

Informationssysteme

Grundvorlesung Informatik
Sommersemester 2004
Universität des Saarlandes, Saarbrücken

Dr. Ralf Schenkel und Prof. Dr. Gerhard Weikum
{schenkel, weikum}@mpi-sb.mpg.de
<http://www.mpi-sb.mpg.de/units/ag5/teaching/ss04/is.htm>

Informationssysteme SS2004

1-1

Organisation

- **Vorlesung:** Di 9-11 und Do 9-11 in 27/001
Sprechstunde Dr. Schenkel:
Sprechstunde Prof. Weikum: Di 14-15 in 36/326 oder n.V. (ggf. e-mail)
- **Übungsgruppen:** Mi und Fr
Übungsgruppenleiter:
Übungs koordinierung: Christian Zimmer (czimmer@mpi-sb.mpg.de)
Sergej Sizov (sizov@mpi-sb.mpg.de)
Erster Übungsgruppentermin: in der zweiten Semesterwoche
- **Leistungsprüfung:**
 - erfolgreiche Teilnahme an zwei von drei Klausuren:
1) Sa 12.6., 2) Sa 24.7., 3) Mitte Oktober
 - erfolgreiche Bearbeitung der praktischen Übungen
(Teamarbeit in Dreiergruppen möglich)
 - Präsentation mindestens einer Übungslösung in der Übungsgruppe

Informationssysteme SS2004

1-2

Gliederung

1. Einführung und Überblick: Anwendungen, Systeme, Prinzipien
 2. Vektorraummodell für Suchmaschinen
 3. Automatische Klassifikation von Dokumenten
 4. Linkanalyse für Autoritäts-Ranking
 5. Relationenmodell und algebraorientierte Anfragesprachen
 6. Logikorientierte Anfragesprachen
 7. Datenbanksprache SQL
 8. Anwendungsentwicklung mit SQL und JDBC
 9. Integritätssicherung mit SQL
 10. Objektorientierte und objekt-relationale Datenmodelle
 11. Relationale Entwurfstheorie
 12. Datenbankentwurf mit UML
 13. Prozessmodellierung mit Statecharts
 14. Datenspeicherung, Indexstrukturen, Anfrageauswertung
 15. Transaktionsverwaltung: Concurrency-Control
 16. Transaktionsverwaltung: Recovery
 17. OLAP und Data-Warehouses
 18. Daten-Mining: Klassifikation, Assoziationsregeln
 19. Semistrukturierte Daten und XML-Anfragesprachen
- Teil I: Suchmaschinen
Teil II: Datenbankschnittstellen
Teil III: Entwurf von Datenbanken und Anwendungen
Teil IV: Implementierungskonzepte von DBS
Teil V: Informationsdienste

Informationssysteme SS2004

1-3

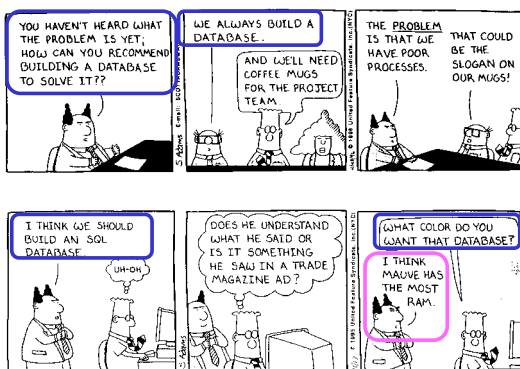
Kapitel 1: Einführung und Übersicht – Anwendungen, Systeme, Prinzipien

- 1.1 Anwendungen
- 1.2 Systemarchitekturen
- 1.3 Grundprinzipien von Datenbanksystemen
- 1.4 Grundprinzipien von Suchmaschinen

Informationssysteme SS2004

1-4

Warum sind Datenbanken und IS so wichtig ?

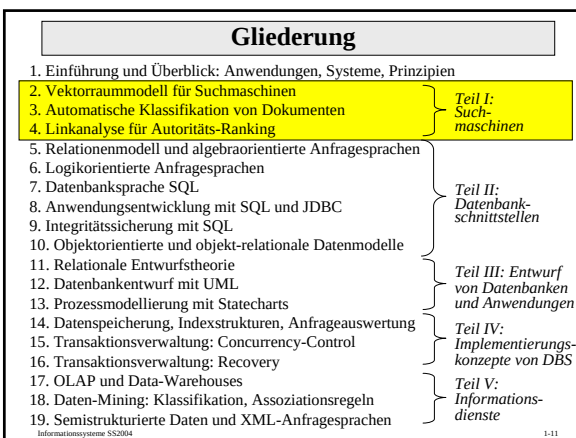
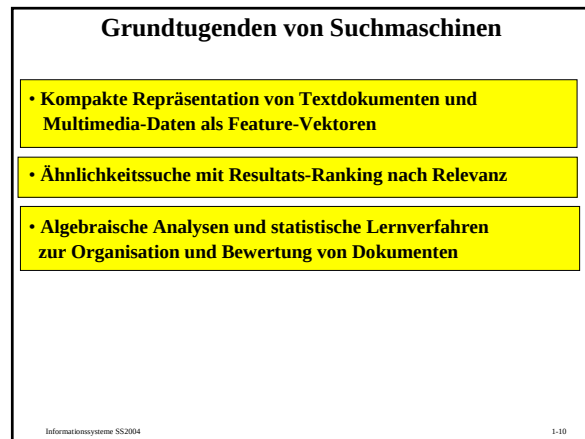
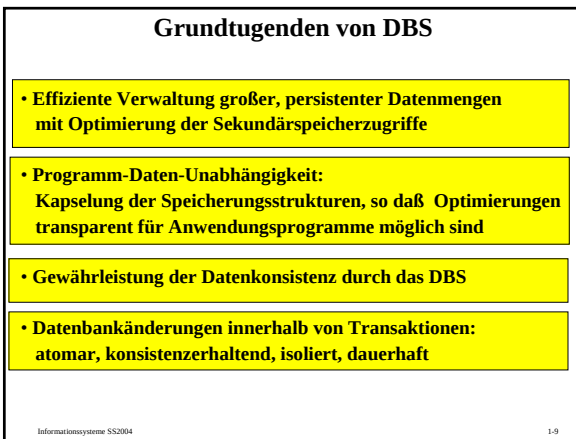
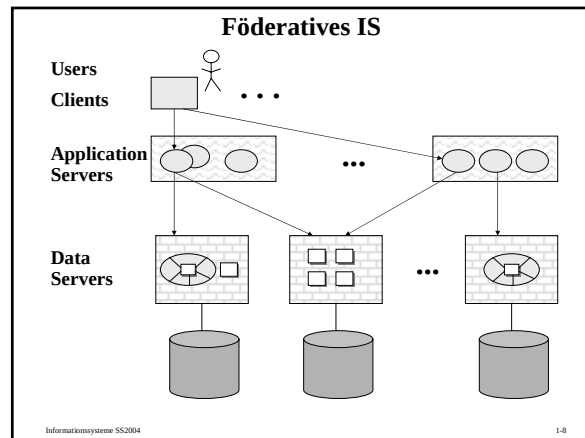
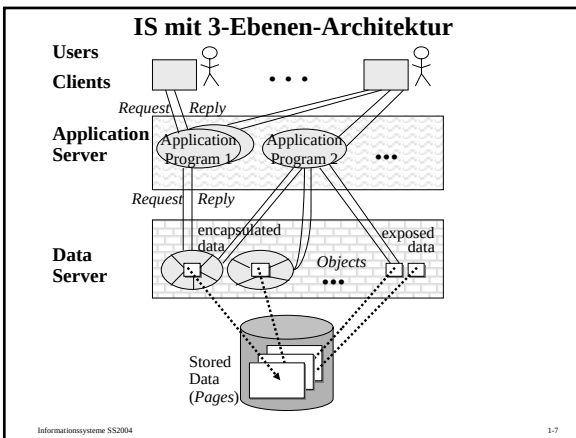


Technische Anforderungen an IS

- Textsuche mit nach Relevanz geordneten Trefferranglisten
- Automatische Organisation von Dokumenten
- Verwaltung von (Deep-)Web-Daten, strukturierten Datenbanken und semistrukturierten (XML-)Daten
- Komfortable GUIs (Graphical User Interfaces)
- deklarative Anfragesprachen
- Zuverlässige Verwaltung sehr großer, persistenter Daten (> 5 TB)
- Hohe Verfügbarkeit (7 x 24) und langfristige Archivierung
- Gute - vorhersagbare und garantierbare - Leistung:
hoher Durchsatz, kurze Antwortzeiten („Quality of Service“)
- Konsistenz verteilter Daten
- Integrierter Zugriff auf heterogene Daten
- Daten-Mining nach Korrelationen, Regeln, Klassifikationen, etc.
- Aktive Regeln und Prozesskoordination
- Komplexe Datentypen (Landkarten, Satellitenbilder, Himmelskarten, ...)
- Ähnlichkeitssuche auf Bildern, wissenschaftlichen Daten, etc.
- Integration mit Web-Applikation und Web-Services

Informationssysteme SS2004

1-6



Kapitel 10: Objektorientierte Datenbanksprachen und Erweiterungen von SQL

10.1 Schwachpunkte relationaler DBS

10.2 Grundkonzepte objektorientierter DBS

10.3 ODMG-Modell

10.4 OO-Anfragesprache OQL

10.5 SQL-Erweiterungen für objekt-relationale DBS

Von relationalen zu objektorientierten DBS

Schwachpunkte relationaler DBS

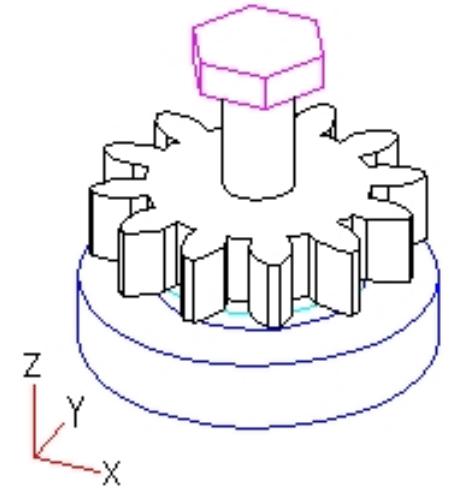
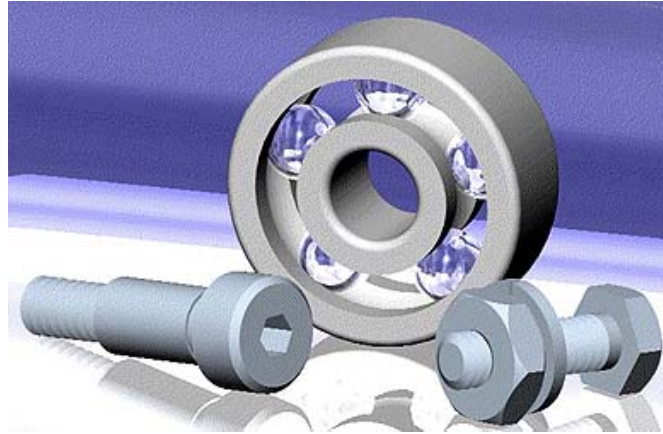
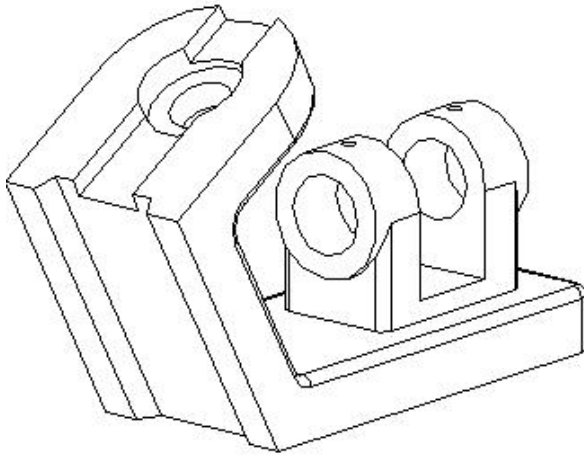
bei technisch-wissenschaftlichen Daten, z.B. bei:

- CAD
- Geowissenschaften
- Desktop Publishing

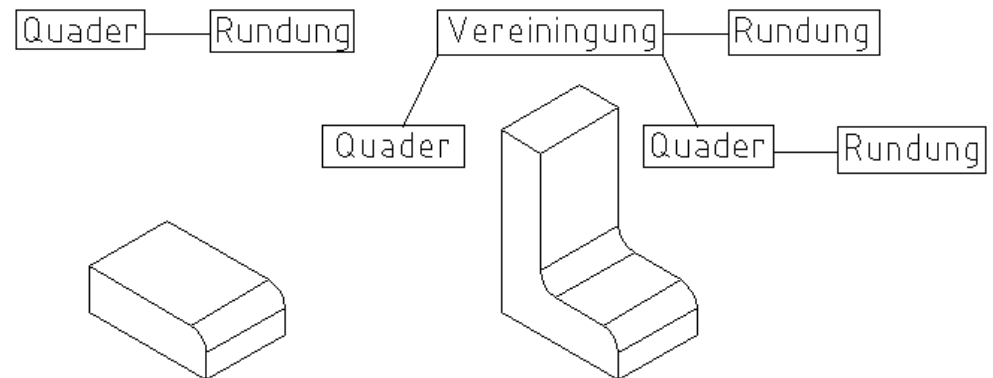
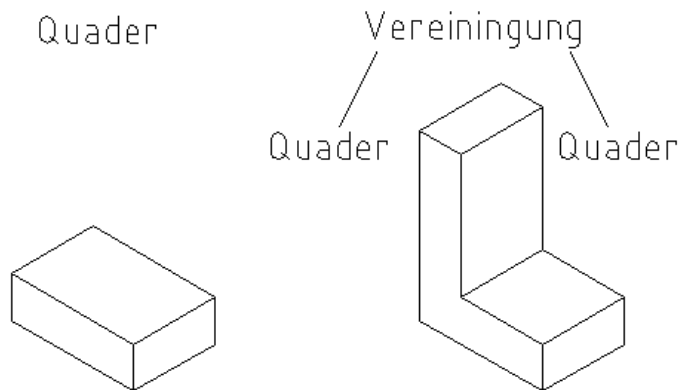
Forderungen an „postrelationale“ DBS:

- Unterstützung komplexer Objektstrukturen
- Erweiterbarkeit des DBS um anwendungsspezifische Methoden
- Wiederverwendbarkeit von Datenschemata und Methoden

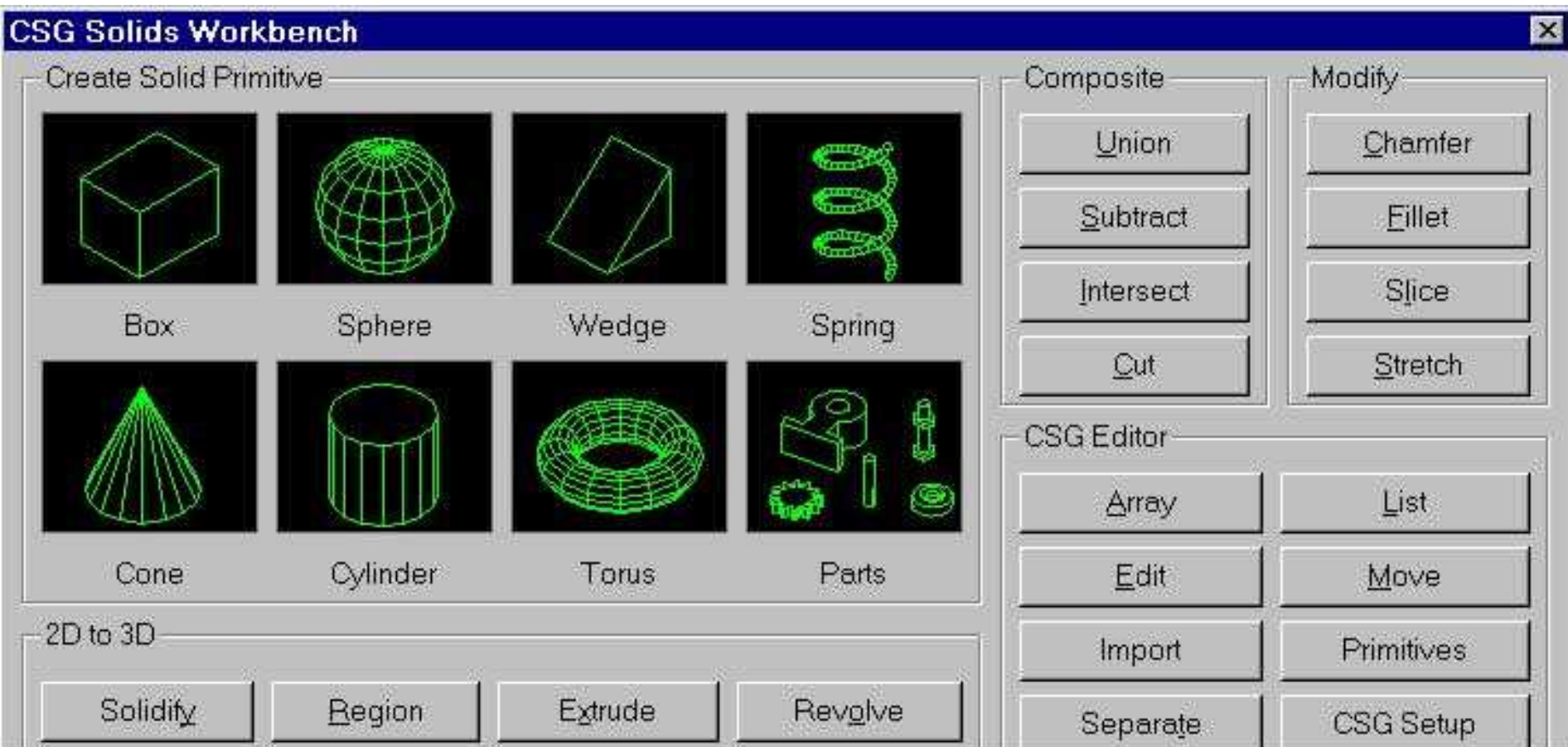
Beispiel 1: CAD (1)



CSG-Modell (Constructive Solid Geometry):

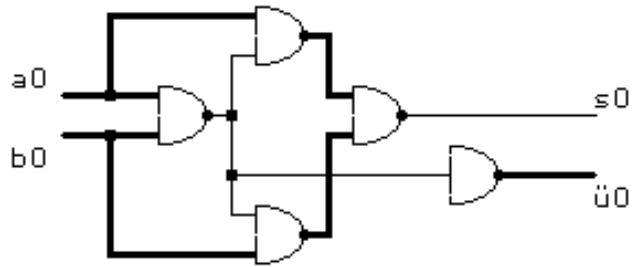


Beispiel 1: CAD (2)

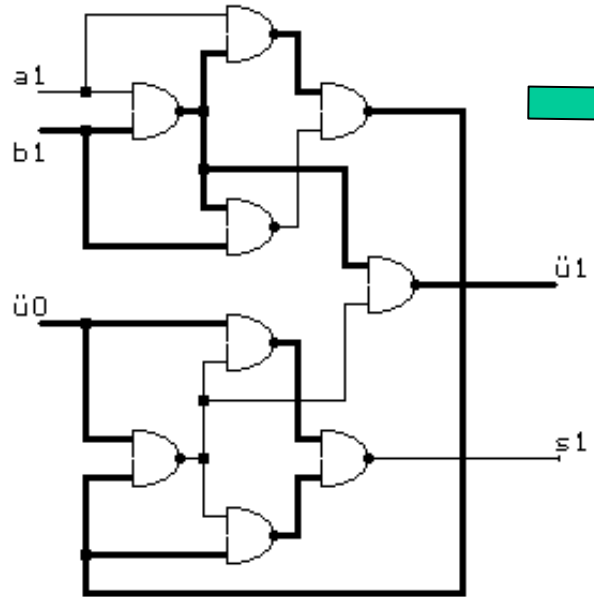


Beispiel 2: Schaltkreisentwurf

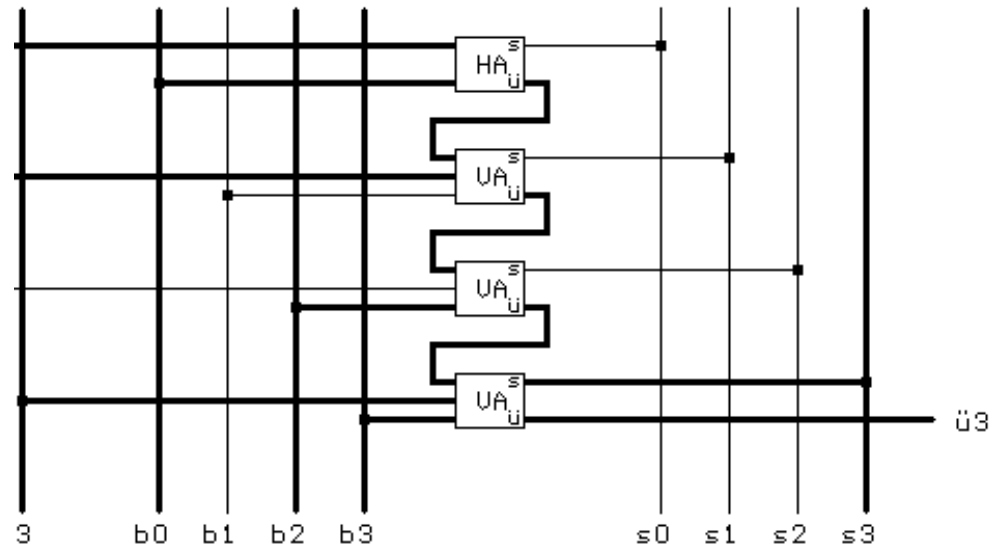
Halbaddierer



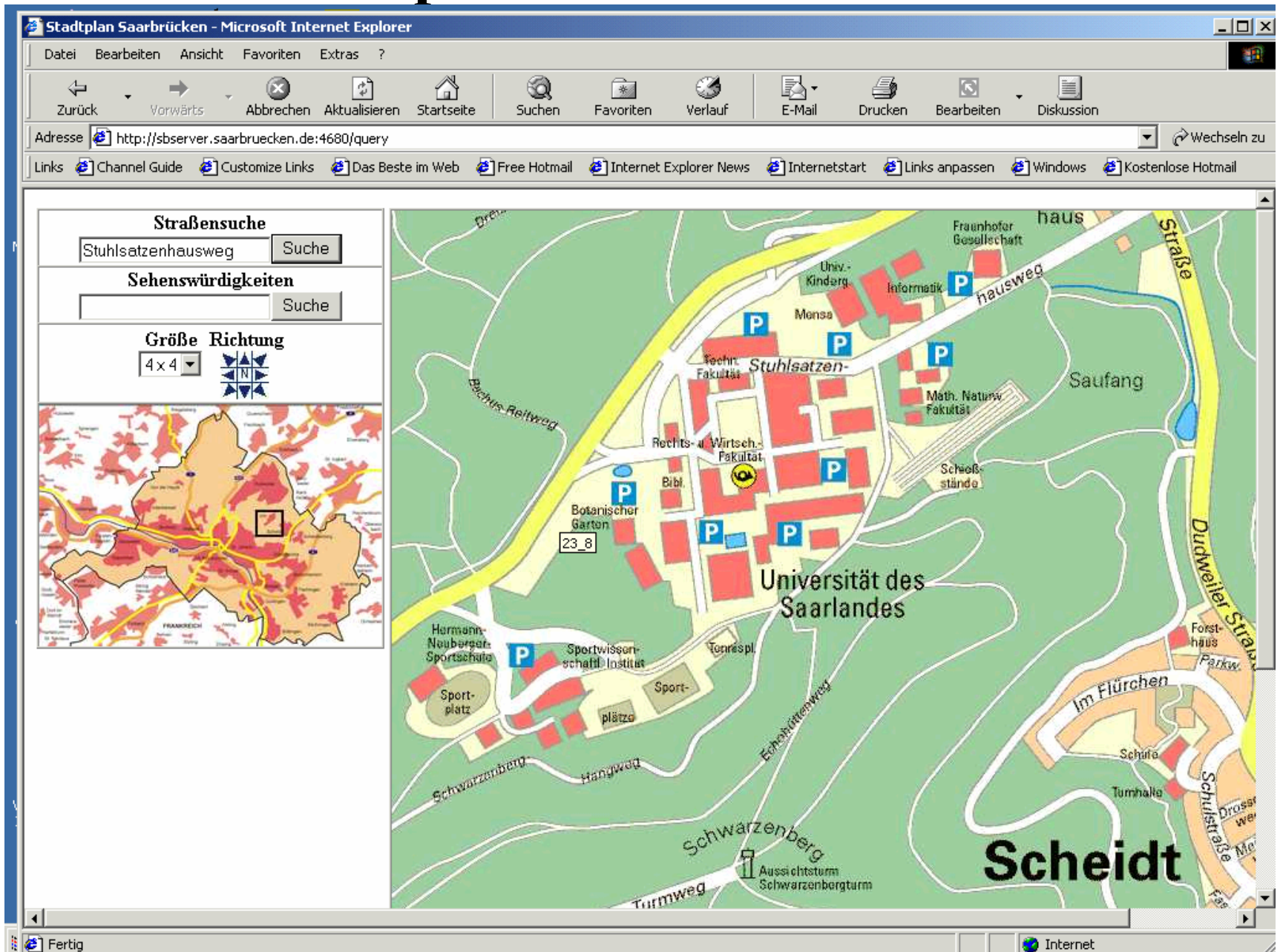
Volladdierer



4 Bit Addierer



Beispiel 3: Straßenkarten



Beispiel 4: Topographische Karten

Internet Explorer interface showing a topographic map of the Grand Canyon area.

Menu: Datei, Bearbeiten, Ansicht, Favoriten, Extras, ?

Toolbar: Zurück, Vorwärts, Abbrechen, Aktualisieren, Startseite, Suchen, Favoriten, Verlauf, E-Mail, Drucken, Bearbeiten, Diskussion

Adresse: <http://www.topozone.com/map.asp?z=12&n=3997450&e=396242&size=m&symshow=n>

Links: Channel Guide, Customize Links, Das Beste im Web, Free Hotmail, Internet Explorer News, Internetstart, Links anpassen, Windows, Kostenlose Hotmail

Map Center: UTM 12 396242E 3997450N - GRAND CANYON quad [\[Quad Info\]](#)
[Click here to download a custom topographic map with TopoFactory](#)

Map Scale: 1:25,000, 1:50,000, 1:100,000, 1:200,000

Map Size: Small, Medium, Large

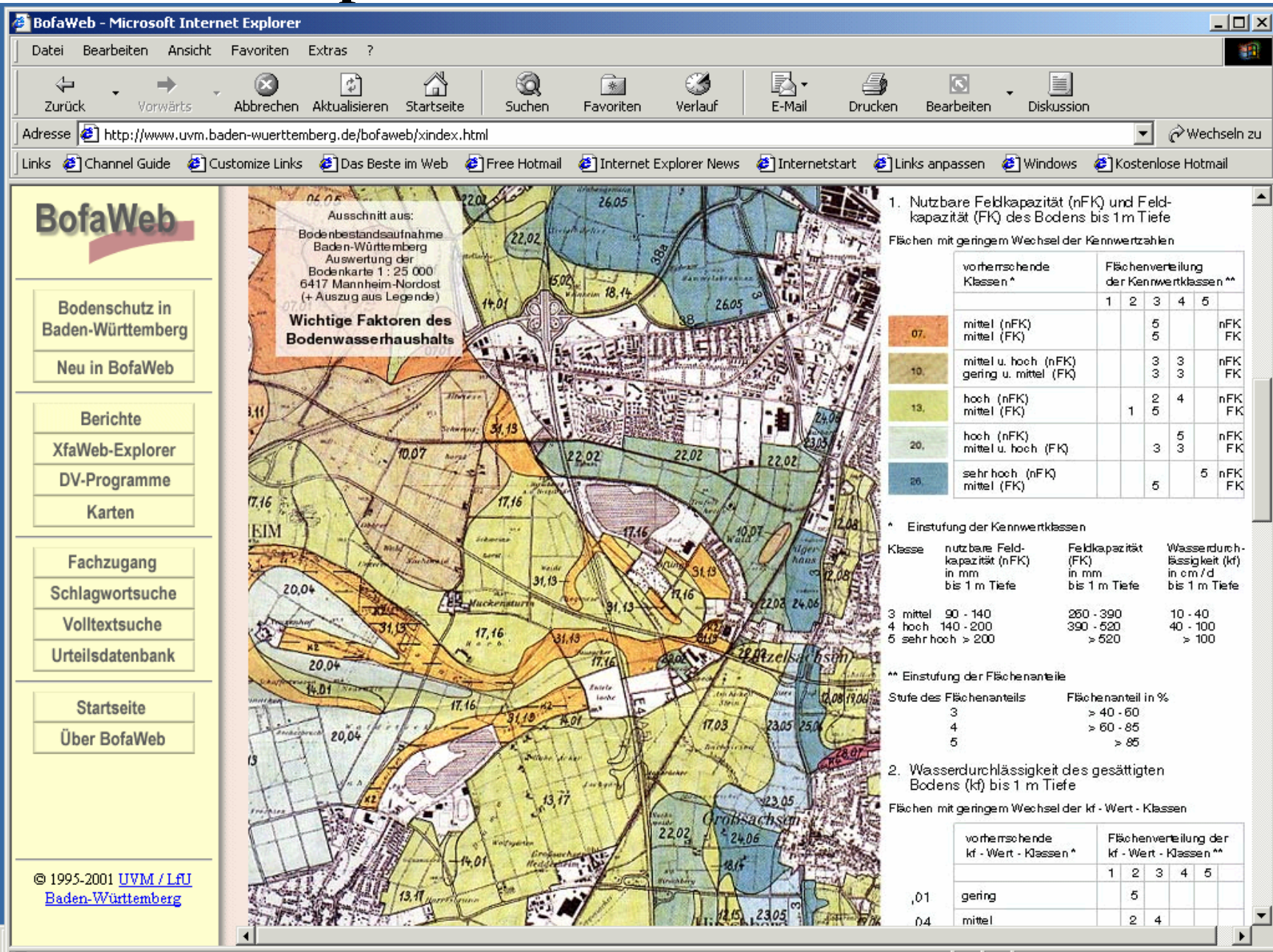
Map Content: Topographic map of the Grand Canyon area, showing the Colorado River, various canyons (e.g., Grand Canyon, Haunted Canyon), and numerous landmarks (e.g., Tower of Ra, Tower of Set, Cheops Pyramid, Buddha Temple, Shiva Temple, Hindu Amphitheater, Claude Birdseye Point, Osiris Temple, Harus Temple, Isis Temple, Schellbach Butte, Clement Powell Butte, Hillers Butte, Johnson Point, Sturdevant Point, The Box, Campground, Ranger Station, Gaging Stations, Plateau Point, Dana Butte, Horn Creek Rapids, Salt Creek Rapids, Granite Rapids, Hermit Rapids, White Canyon, Marsh Butte, Crystal Rapids, Braucher Rapids, Cocopa Point, Yesta temple, Cope, The Colonnade, Flvo Point, Widde Point, Manu Temp, Buddha Temple, Schellbach Butte, Clement Powell Butte, Hillers Butte, Johnson Point, Sturdevant Point, The Box, Campground, Ranger Station, Gaging Stations, Plateau Point, Dana Butte, Horn Creek Rapids, Salt Creek Rapids, Granite Rapids, Hermit Rapids, White Canyon, Marsh Butte, Crystal Rapids, Braucher Rapids, Cocopa Point, Yesta temple, Cope).

Left Sidebar:

- Get a Map**
 - [Place name search](#)
 - [Decimal degrees](#)
 - [Deg/min/sec](#)
 - [UTM coords](#)
- Download Maps**
 - [TopoFactory Login](#)
 - [About TopoFactory](#)
 - [Specifications](#)
 - [TopoFactory Store](#)
- How to...**
 - [Put topo maps on your Web site](#)
 - [Get digital data](#)
 - [Link to us](#)
- What's New?**
 - [Get the TopoTimes](#)
 - [Press releases](#)
 - [New @ TopoZone](#)
 - [Awards](#)
- Help**
 - [Map tips](#)
 - [Topo map symbols](#)
 - [FAQ](#)
 - [Support](#)
 - [Privacy policy](#)
 - [About us](#)
- Current Weather**
 - for Supai
 -
 - Mostly Clear 74 F
 - [More](#)
 - powered by AccuWeather.com

Bottom Status Bar: Cursor is UTM 12 396809E 4001760N, Internet

Beispiel 5: Thematische Karten



Beispiel 6: Desktop Publishing

Dokument1 - Microsoft Word

Datei Bearbeiten Ansicht Einfügen Format Extras Tabelle Fenster ?

Standard Times New Roman 11 F K U

75%

1 Rethinking Database System Architecture: Towards a Self-tuning RISC-style Database System
Abstract

2 Crisis Indicators

Observation 1: Featurism drives products beyond manageability

Observation 2: SQL is painful

3 Explanations and Previous Attempts for Architectural Departures
3.1 Explanations
3.2 Previous Attempts

4 Towards RISC-style Data Management Components
4.1 Notable Departures from Today's Architectures
4.2 Prerequisites of Success
4.3 Evaluation of Success

5 Concluding Remarks

they did not catch on. Section 4 outlines the envisioned architecture with emphasis on RISC-style simplification of data-management components and consequences for the viability of auto-tuning!
This part needs to be modified as we add new sections.

To begin our analysis, let us put together a few important observations on how database systems are perceived by customers, software vendors, and the research community.

Database systems offer more and more features, leading to extremely broad and thus complex interfaces. Quite often novel features are more a marketing issue rather than a real application need or technological advance; for example, a database system vendor may decide to support a fancy type of join or spatial index ~~or certain input/output interfaces for OO development~~ in the next product release because the major competitors have already announced this feature. As a result, database systems become overburdened with functionality, increasing the complexity of maintaining the system's code base as well as installing and managing the system. The irony of this trend lies in the fact that each individual customer (e.g., a small company or a single department of a large enterprise) only makes use of a tiny fraction of the system's features.

A big headache that comes with a database system is the SQL language. It is the union of all conceivable features (many of which are rarely used or should be discouraged to use anyway) and is way too complex for the typical application developer. Its core, say projection-selection-join queries plus grouping and aggregation, is undebated extremely useful, but we doubt that there is wide and wise use of all the bells and whistles. ~~We are not aware of any textbook or course notes that give a precise Understanding~~ semantics of SQL (not even of SQL-92), covering all combinations of nested (and correlated) subqueries, null values, triggers, ADT functions, etc. ~~is a nightmare~~. Teaching SQL typically focuses on the core, and leaves the featurism as a "learning-on-the-job" life experience. Some trade magazines occasionally pose SQL quizzes where the challenge is to express a complicated information request in a single SQL statement. Those statements run over several pages, and are hardly comprehensible. When programmers adopt this style in real applications and given the inherent difficulty of debugging a very high-level "declarative" statement, it is extremely hard if not impossible to gain high confidence that the query is correct in capturing the users' information needs. ~~So In fact~~, good SQL programming typically in many cases decomposes complex requests into a sequence of smaller simpler SQL statements, and there is no demand for much of the complexity of full-fledged SQL.

¹ Throughout the paper, we prefer somewhat "radical", hopefully thought-provoking statements. We do realize that the world is not just black-and-white and some of our arguments or conclusions are oversimplified, but new departures do require a certain amount of radicalism to achieve an effect.
² By precise semantics we mean a mathematically rigorous and objective description, e.g., a translation into some form of logic, algebraic framework, or even a procedural language like Java (whose semantics is much better defined). English prose or "semantics by example" are ambiguous and typically boil down to handwaving.

Seite 2 Ab 3 3/20 Bei 16 cm Ze 24 Sp 6 MAK AND ERW UB English (US)

10.2.1 Komplexe Objekte

Objekt: Menge von Attributen (attribute, property, member)

Typ eines Objekts: Menge der Attribute des Objekts

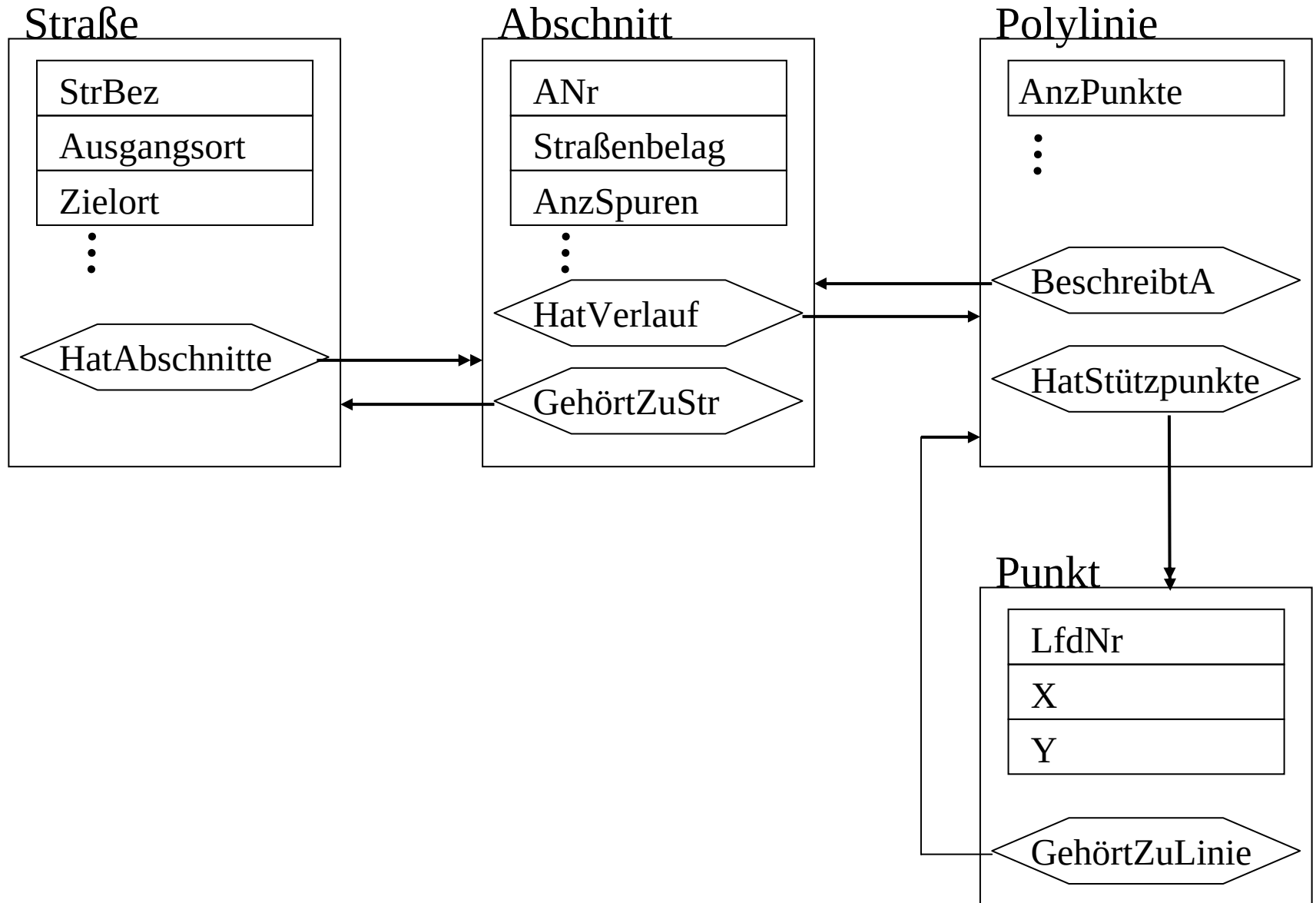
Klasse: Menge von Objekten desselben Typs

OO-DB: Menge von Klassen

Typ eines Attributs:

- elementar (Integer, Real, String, Boolean)
- erzeugt mit Typkonstruktoren (mit Parametertypen):
 - Relationship T: Referenz auf Objekt vom Typ T
 - Struct<T1, ..., Tk>: Tupel von Attributen vom Typ T1, ..., Tk
 - Set<T>: Menge von Elementen des Typs T
 - Bag<T>: Multimenge von Elementen vom Typ T
 - List<T>: Liste von Elementen vom Typ T
 - Array<T>: Abbildung (eines Intervalls) von natürlichen Zahlen auf Elemente vom Typ T

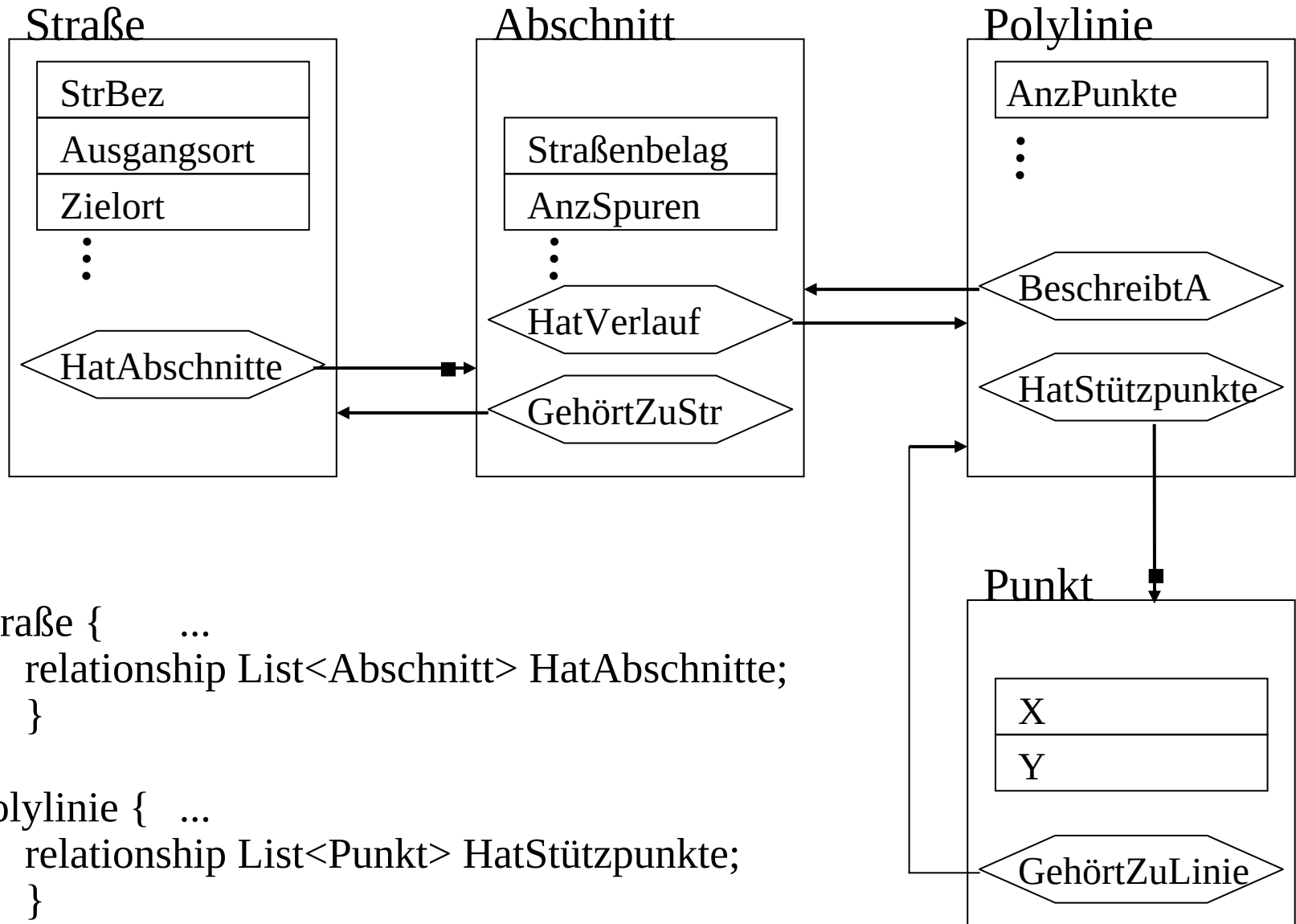
Beispiel – Alternative 1



Beispiel – Alternative 1 (textuell)

```
class Straße { extent Straßen;  
    key StrBez;  
    attribute String StrBez;  
    attribute String Ausgangsort;  
    attribute String Zielort;  
    ...  
    relationship Set<Abschnitt> HatAbschnitte;  
}  
class Abschnitt { extent Abschnitte;  
    key (GehörtZuStr.StrBez, ANr);  
    attribute Integer ANr;  
    attribute String Straßenbelag;  
    attribute Integer AnzSpuren;  
    attribute Integer Tempolimit;  
    ..  
    relationship Polylinie HatVerlauf;  
    relationship Straße GehörtZuStr;  
}
```

Beispiel – Alternative 2



```

class Straße {
    ...
    relationship List<Abschnitt> HatAbschnitte;
}

```

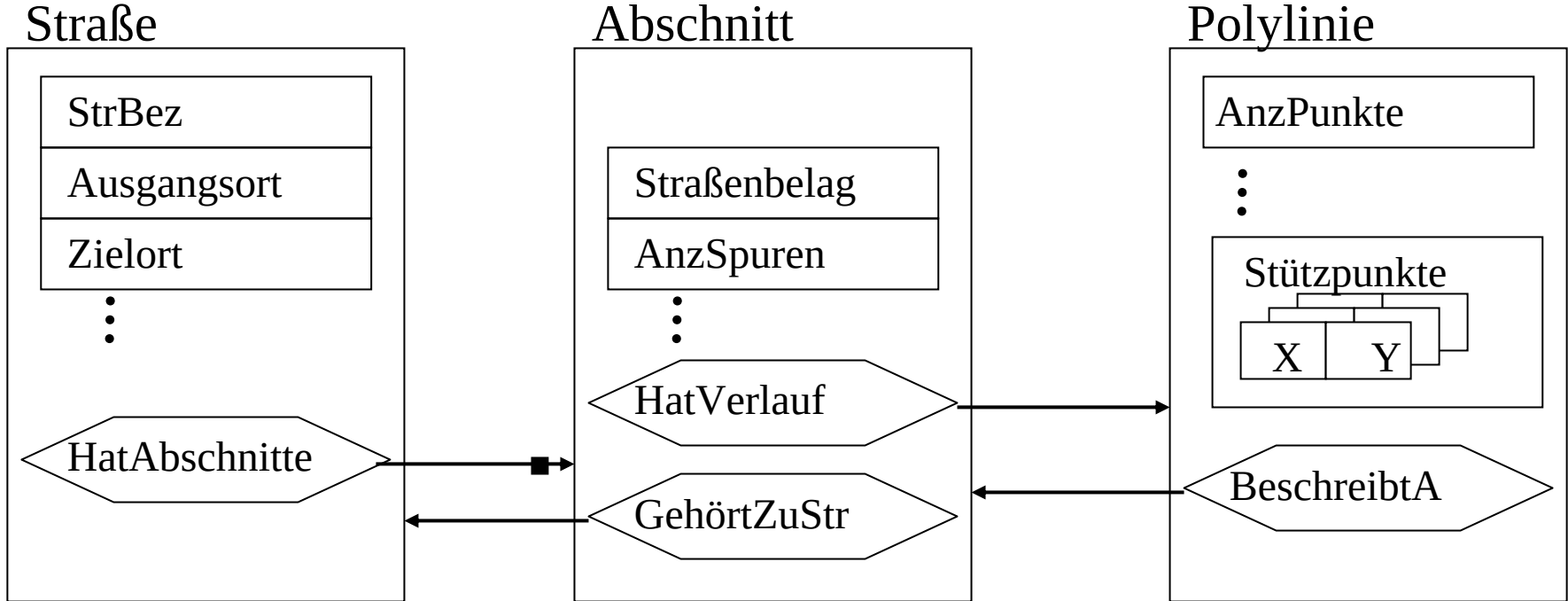
```

...
class Polylinie {
    ...
    relationship List<Punkt> HatStützpunkte;
}

```

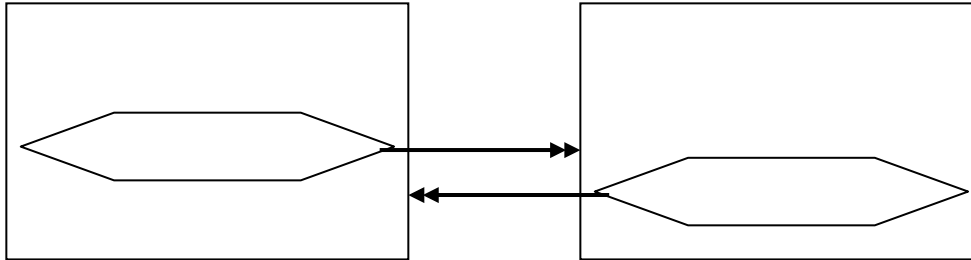
...

Beispiel – Alternative 3

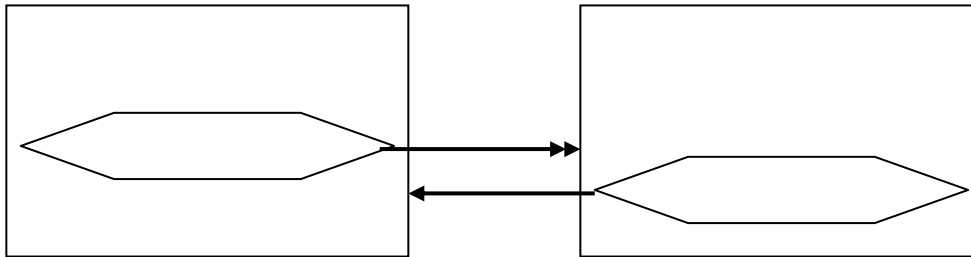


```
...  
class Polylinie { ...  
    attribute Integer AnzPunkte;  
    attribute List<Struct<X: Real; Y: Real>> Stützpunkte;  
}
```

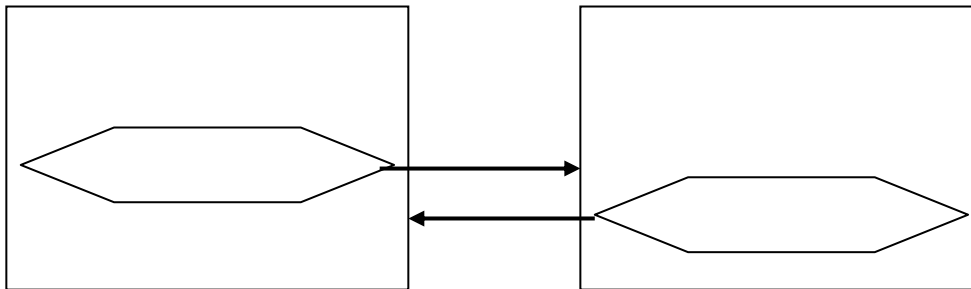
Verschiedene Arten von Relationships



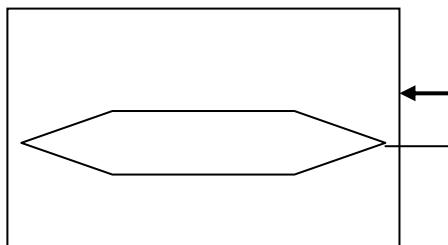
M:N-Relationship



1:N-Relationship
N:1-Relationship



1:1-Relationship



Relationship zwischen
Objekten derselben Klasse

Integritätsbedingungen für Relationships

Sei $R \subseteq A \times B$.

Dies entspricht: $R_A: A \rightarrow 2^B$ und $R_B: B \rightarrow 2^A$.

Die folgende Invariante muß gelten:

$$\forall x \in A: x \in \bigcup_{y \in R_A(x)} R_B(y) \quad \text{und} \quad \forall y \in B: y \in \bigcup_{x \in R_B(y)} R_A(x)$$

bzw.: $R_A(x) = \{y \in B \mid (x, y) \in R\}$ und $R_B(y) = \{x \in A \mid (x, y) \in R\}$

Beispiele:

- 1)

```
class Männer { ...  
    relationship Frauen Ehefrau inverse Frauen::Ehemann; }  
class Frauen { ...  
    relationship Männer Ehemann inverse Männer::Ehefrau; }
```
- 2)

```
class Straße { ...  
    relationship Set<Abschnitt> HatAbschnitte inverse Abschnitt::GehörtZuStr; }  
class Abschnitt { ...  
    relationship Straße GehörtZuStr inverse Straße::HatAbschnitte; ... }
```

Objekt-Identität und Objekt-Sharing

Es kann mehrere Objekten geben, die in allen Attributwerten übereinstimmen, aber verschiedene *Objekt-Ids (OIDs)* haben.

Ein Objekt kann von mehreren Objekten referenziert werden. Damit sind Änderungen auf einem Subobjekt eines Objekts für andere Objekte mit demselben Subobjekt unmittelbar sichtbar („*Objekt-Sharing*“).

Beispiel:

```
class Abschnitt { ...  
    relationship Set<Straße> GehörtZuStr inverse Straße::HatAbschnitte; ... }
```

Straße [StrBez="Mainzer Str."].HatAbschnitte.first().AnzSpuren = 4
wirkt sich sofort auf Abschnitt der B51 aus.

10.2.2 Objektmethoden und Kapselung

Zu jeder Klasse kann eine Menge von *Methoden* definiert werden:
Funktionen mit Signatur $T_1 \times T_2 \times \dots \times T_n \rightarrow T_{(n+1)}$ und
Prozeduren mit Seiteneffekten auf den Objektzustand.
Ein Methodenaufruf $x.m(\dots)$ hat das Objekt x als impliziten Parameter.

Die Gesamtheit der zu einer Klasse definierten – öffentlich sichtbaren – Methoden bildet die *Schnittstelle eines Abstrakten Datentyps (ADT)*, und jedes Objekt der Klasse ist dann eine Instanz dieses ADTs.
Attribute (d.h. der konkrete Objektzustand) können als privat definiert werden und sind dann nicht mehr öffentlich sichtbar, sondern „gekapselt“.

Vorteil der Kapselung:

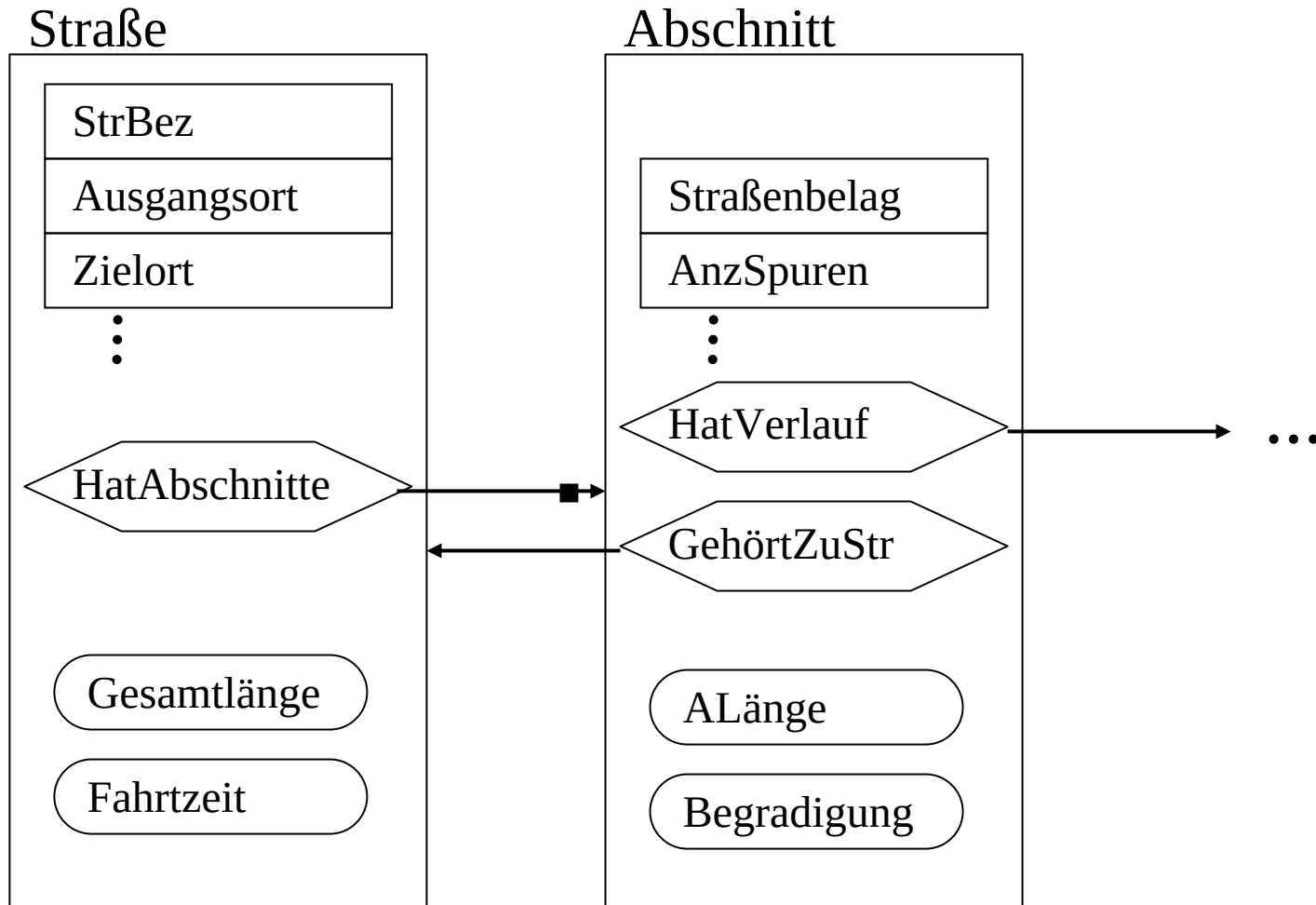
Die Implementierung einer Klasse und seiner Methoden kann geändert werden, ohne dass sich die Schnittstelle der Klasse ändert.

Beispiel: Objektmethoden

```
class Straße { ... private attribute Integer Durchschnittstempo;  
                  Real Gesamtlänge ( );  
                  Integer Fahrtzeit ( ); }  
class Abschnitt { ... Real ALänge ( );  
                   Boolean Begradigung (in Punkt, in Punkt); }
```

```
Straße:: Gesamtlänge () {  
    float l = 0.0;  
    Set<Ref<Abschnitt>> MeineAbschnitte = this->HatAbschnitte;  
    Iterator<Abschnitt> it = MeineAbschnitte->create_iterator();  
    Ref<Abschnitt> a;  
    while (a=it.next()) { l += a->ALänge(); } return l };  
Abschnitt:: ALänge () {  
    float l = 0.0;  
    List<Ref<Punkt>> MeinePunkte = this->HatVerlauf->HatStützpunkte;  
    Iterator<Punkt> it = MeinePunkte->create_iterator();  
    Ref<Punkt> p, q;  
    p = it->next();  
    while (q=it->next()) {  
        l += sqrt ((p->X - q->X) * (p->X - q->X) + (p->Y - q->Y) * (p->Y - q->Y));  
        p = q; };  
    return l };
```

Beispiel: Objektmethoden (Diagramm)



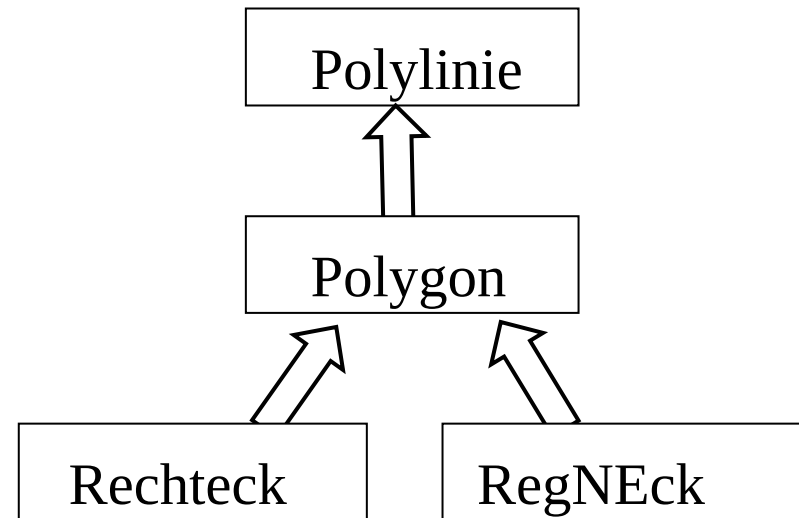
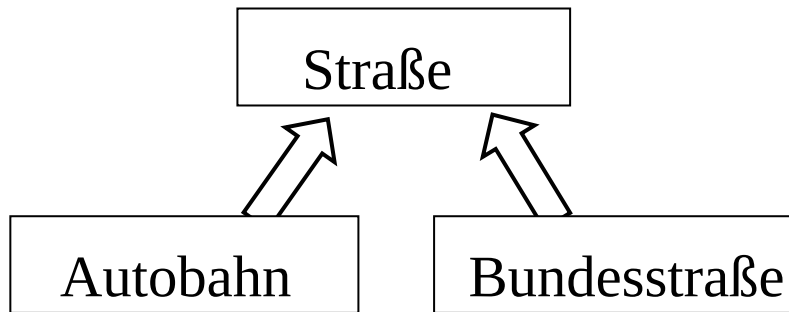
10.2.3 Vererbung

Klasse B heißt Subklasse von Klasse A (A Superklasse von B), wenn

- Attribute und Methoden von B eine Obermenge der von A bilden und
- die Objektmenge von B eine Teilmenge der von A ist.

B heißt Spezialisierung von A, und A heißt Generalisierung von B.

Beispiele:



Beispiele: Vererbung

```
class Bundesstraße: Straße { extent Bundesstraßen;  
    attribute Integer Verkehrsdichte; }
```

```
class Autobahn: Straße { extent Autobahnen;  
    attribute Integer Mindesttempo;  
    attribute Array<Integer> VerkehrsdichteProTag;  
    relationship Set<Punkt> Auffahrten; }
```

```
class Polygon: Polylinie { extent Polygone;  
    relationship Punkt Zentrum;  
    relationship Stadt BeschreibtS inverse Stadt::Stadtgrenze;  
    relationship Land BeschreibtL inverse Land::Landesgrenze;  
    Real Fläche ( ); }
```

```
class Rechteck: Polygon { extent Rechtecke;  
    attribute Real Diagonallänge; }
```

```
class RegNEck: Polygon { extent RegNEcke;  
    attribute Integer AnzEcken;  
    Real Inkreisradius ();  
    Real Umkreisradius (); }
```

Umgang mit geerbten Methoden

Überschreiben der Implementierung einer Methode

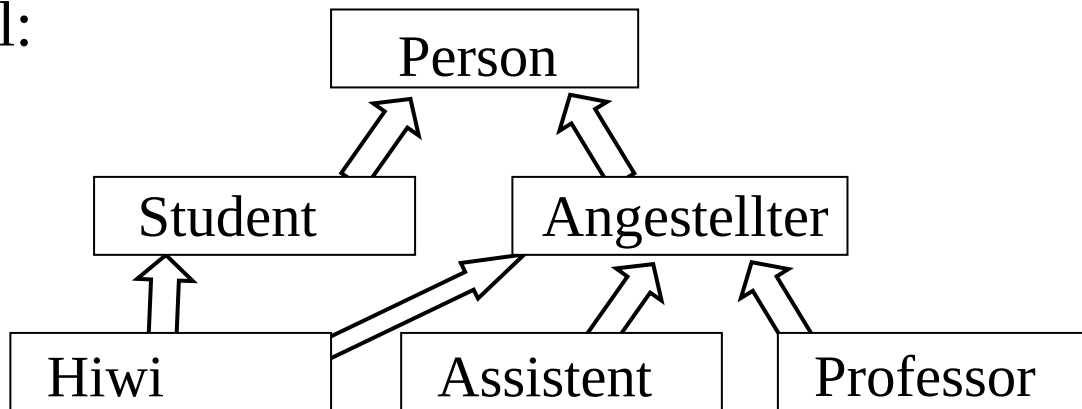
wegen speziellerer Semantik oder Effizienz

Beispiele:

- 1) Fläche () der Subklasse „Polygone mit Löchern“
- 2) Fläche () der Subklassen „Rechtecke“ und „RegNEcke“
- 3) SchneidetBox (in Rechteck) der Subklasse „Polygone“

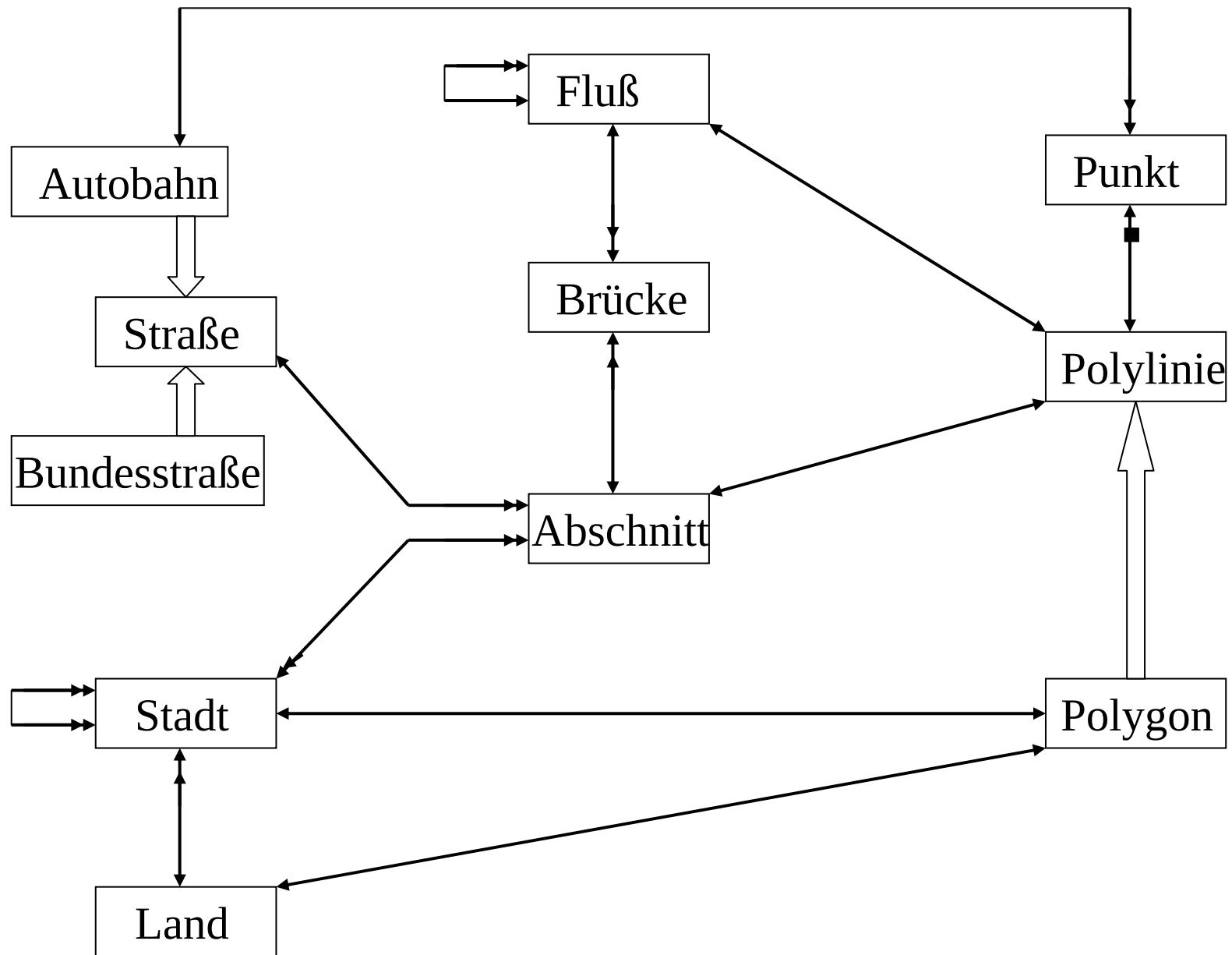
Mehrfachvererbung erfordert ggf. Umbenennung

Beispiel:



Bei der Verarbeitung **polymorpher Objektmengen** (z.B. Polygone) wird die jeweils passende Methodenimplementierung zur Laufzeit bestimmt („late binding“)

10.3 ODMG-Standard: ODL



Beispiel: ODL-Schema (1)

```
class Straße { extent Straßen; key StrBez;
    attribute String StrBez;
    attribute String Ausgangsort; attribute String Zielort;
    relationship Set<Abschnitt> HatAbschnitte inverse Abschnitt::GehörtZuStr;
    Real Gesamtlänge ( ); Integer Fahrtzeit ( ); }

class Bundesstraße: Straße { extent Bundesstraßen;
    attribute Integer Verkehrsdichte; }

class Autobahn: Straße { extent Autobahnen;
    attribute Integer Mindesttempo;
    attribute Array<Integer> VerkehrsdichteProTag;
    relationship Set<Punkt> Auffahrten; }

class Abschnitt { extent Abschnitte; key (GehörtZuStr.StrBez, ANr);
    attribute Integer ANr; attribute String Straßenbelag;
    attribute Integer AnzSpuren; attribute Integer Tempolimit;
    relationship Polylinie HatVerlauf inverse Polylinie::BeschreibtA;
    relationship Straße GehörtZuStr inverse Straße::HatAbschnitte;
    relationship Set<Brücke> HatBrücken inverse Brücke::GehörtZuA;
    relationship Set<Stadt> FührtDurch inverse Stadt::LiegtAn;
    Real ALänge ( ); }
```

Beispiel: ODL-Schema (2)

```
class Polylinie { extent Polylinien;
    attribute Integer AnzPunkte;
    relationship List<Punkt> HatStützpunkte inverse Punkt::GehörtZuLinie;
    relationship BeschreibtA inverse Abschnitt::HatVerlauf;
    relationship BeschreibtF inverse Fluß::HatVerlauf;
    Real Länge ( );
    void Begradigung (in Punkt, in Punkt) Raises (Zu_große_Anschlußkrümmung);
    Boolean Schneidet (in Polylinie);
    Boolean SchneidetBox (in Rechteck);
    Boolean LiegtInBox (in Rechteck); }

class Punkt { extent Punkte;
    attribute Real X; attribute Real Y;
    relationship Polylinie GehörtZuLinie inverse Polylinie::HatStützpunkte
    Boolean LiegtInBox (in Rechteck);
    Boolean LiegtInRegion (in Polygon); }

class Polygon: Polylinie { extent Polygone;
    relationship Punkt Zentrum;
    relationship Stadt BeschreibtS inverse Stadt::Stadtgrenze;
    relationship Land BeschreibtL inverse Land::Landesgrenze;
    Real Fläche ( ); }
```

Beispiel: ODL-Schema (3)

```
class Fluß { extent Flüsse; key Flußbez;  
    attribute String Flußbez; attribute Array<Real> Verschmutzungsgrad;  
    relationship Punkt Quelle; relationship Punkt Mündung;  
    relationship Fluß MündetIn inverse Fluß::HatZuflüsse;  
    relationship Set<Fluß> HatZuflüsse inverse Fluß::MündetIn;  
    relationship Polylinie HatVerlauf inverse Polylinie::BeschreibtF;  
    relationship Set<Brücke> FließtUnter inverse Brücke::ÜberquertF; }  
class Brücke { extent Brücken;  
    attribute Real Höhe; attribute Real Spannweite;  
    relationship Punkt Anfang; relationship Punkt Ende;  
    relationship Fluß ÜberquertF inverse Fluß::FließtUnter;  
    relationship Abschnitt GehörtZuA inverse Abschnitt::HatBrücken; }  
class Stadt { extent Städte;  
    attribute String Stadtname; attribute Integer Einwohnerzahl;  
    relationship Person Bürgermeister;  
    relationship Set<Stadt> Partnerstädte inverse Stadt::Partnerstädte;  
    relationship Land GehörtZuL inverse Land::HatStädte;  
    relationship Polygon Stadtgrenze inverse Polygon::BeschreibtS;  
    relationship Set<Abschnitt> LiegtAn inverse Abschnitt::FührtDurch;  
    Real Einwohnerdichte ( ); void Eingemeindung (Inout Stadt); }  
class Land { extent Länder; key Landesbez;  
    attribute String Landesbez; relationship Person Staatsoberhaupt;  
    relationship Set<Stadt> HatStädte inverse Stadt::GehörtZuL;  
    relationship Polygon Landesgrenze inverse Polygon::BeschreibtL;  
    Integer Bevölkerung ( ); }
```

Generische Typen (“Templates”) in ODL

Collection<T>:

attribute Integer cardinality; attribute Boolean empty?;
Collection<T> create (); void delete ();
void insert_element (in T); void remove_element (in T);
Collection<T> select (in String);
Boolean exists? (in String); Boolean contains_element? (in T);
Iterator create_iterator ();

...

Set<T>:

Set<T> union (in Set<T>); Set<T> intersection (in Set<T>); Set<T> difference (in Set<T>);
Boolean is_subset? (in Set<T>); Boolean is_proper_subset? (in Set<T>);
Boolean is_superset? (in Set<T>); Boolean is_proper_superset? (in Set<T>);

...

List<T>:

T retrieve_first_element (); T retrieve_last_element ();
T retrieve_element_at (in Integer);
void insert_first_element (in T); void insert_last_element (in T);
void insert_element_after (in T, in Integer); void insert_element_before (in T, in Integer);

...

10.4 ODMG-Standard: OQL

- angelehnt an SQL, aber mit signifikanten Abweichungen und OO-Erweiterungen
- Pfadausdrücke zur Traversierung von Relationships („implizite Joins“):
C0.R1.R2.Rn.A in Select-Klausel entspricht
$$\{t_n.A, \dots \mid \dots \wedge \exists t_0 \exists t_1 \dots \exists t_{(n-1)} (t_0.R1 = t_1.OID \wedge t_1.R2 = t_2.OID \wedge \dots \wedge t_{(n-1)}.Rn = t_n.OID)\}$$

C0.R1.R2.Rn.A θ wert in Where-Klausel entspricht
$$\exists t_0 \exists t_1 \dots \exists t_{(n-1)} (t_0.R1 = t_1.OID \wedge \dots \wedge t_{(n-1)}.Rn = t_n.OID \wedge t_n.A \theta \text{ wert})$$
- Subqueries auch in Select- und From-Klausel:
konstruieren geschachtelte Ergebnisse (z.B. vom Typ $\text{Set}<\text{Set}<T>>$)
- Methodenaufrufe: wie Attribute einer Klasse, z.B. C.F(x,y) oder C.G()
- Automatische Berücksichtigung von Subklassen
- Erweiterte Aggregationen und Gruppierung

Beispiele: Pfadausdrücke in OQL (1)

1) Welche Flüsse überquert die B51?

```
SELECT FLATTEN (S.HatAbschnitte.HatBrücken.ÜberquertF)
FROM Straßen S
WHERE S.StrBez="B51"
```

oder:

```
SELECT B.ÜberquertF
FROM ( SELECT FLATTEN(A.HatBrücken)
      FROM ( SELECT FLATTEN (S.HatAbschnitte)
            FROM S IN Straßen
            WHERE S.StrBez="B51" ) AS A
      ) AS B
```

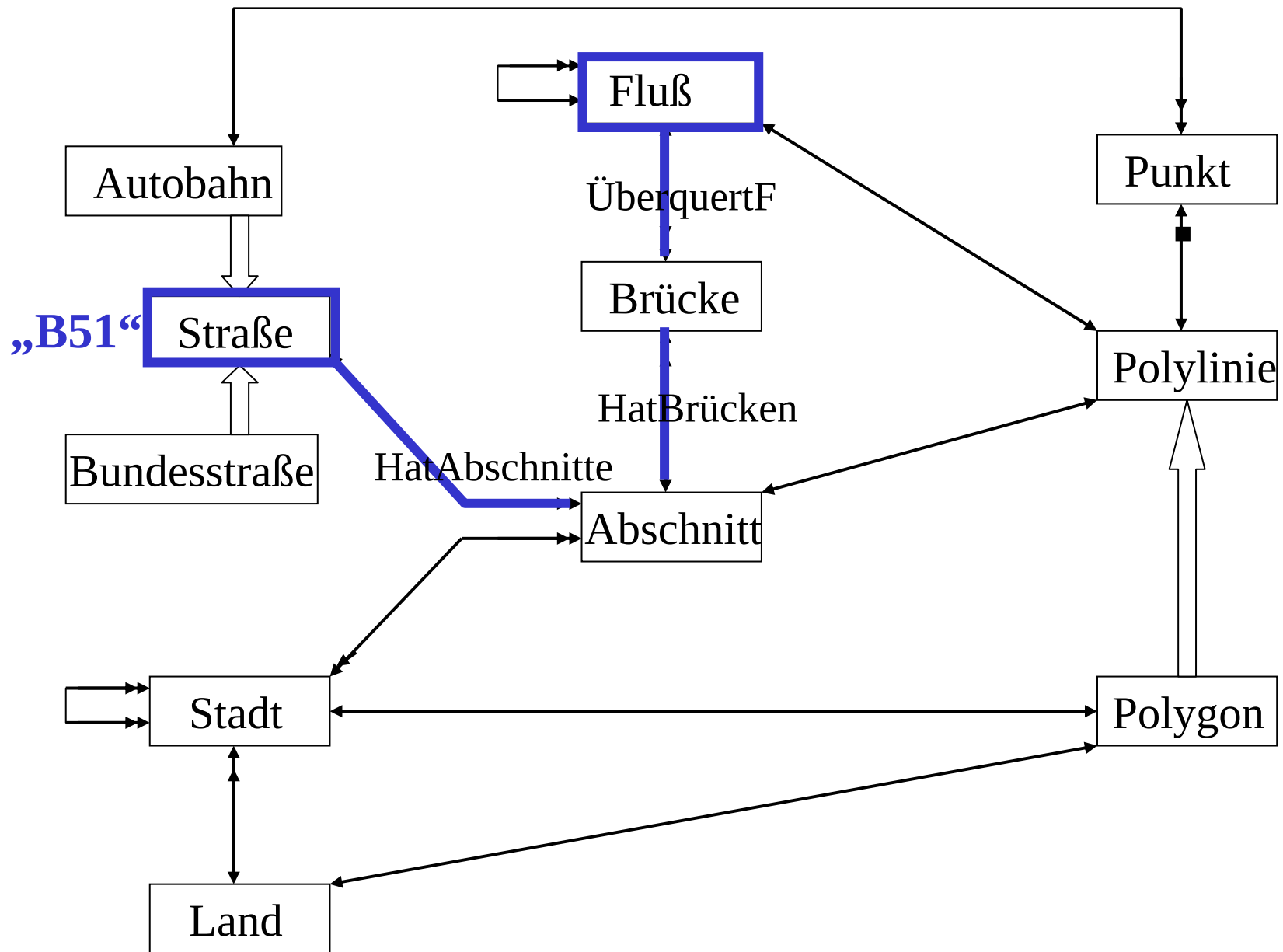
2) Welche Straßen überqueren die Mosel?

```
SELECT S
FROM Straßen S
WHERE S.HatAbschnitte.HatBrücken.ÜberquertF.Flußbez="Mosel"
```

3) Welche Partnerstädte haben die Städte der Elfenbeinküste?

```
SELECT FLATTEN(L.HatStädte.Partnerstädte)
FROM Länder L
WHERE L.Landesbez="Elfenbeinküste"
```

Pfadausdrücke in OQL – Beispiel 1



Beispiele: Pfadausdrücke in OQL (2)

4) In welchen Ländern haben Städte der Elfenbeinküste Partnerstädte?

```
SELECT DISTINCT FLATTEN ( L.HatStädte.Partnerstädte.GehörtZuL )  
FROM Länder L  
WHERE L.Landesbez="Elfenbeinküste"
```

5) Ausgabe des Verlaufs der B51:

```
SELECT S.HatAbschnitte.HatVerlauf.HatStützpunkte  
FROM Straßen S  
WHERE S.StrBez="B51"
```

oder:

```
SELECT A.HatVerlauf.HatStützpunkte  
FROM ( SELECT S.HatAbschnitte  
        FROM Straßen S  
        WHERE S.StrBez="B51"  
        SORT A IN S.HatAbschnitte BY A.ANr ) AS A
```


Beispiele: Methodenaufrufe in OQL (1)

1) Wieviele Kilometer hat das gesamte Straßennetz?

```
SELECT SUM ( S.Gesamtlänge() ) FROM Straßen S
```

2) Welche Städte haben eine Fläche von mehr als 30 Quadratkilometern?

```
SELECT S.Stadtname FROM Städte S  
WHERE S.Stadtgrenze.Fläche() > 30
```

3) In welchen Städten mit einer Fläche von mehr als 30 km² haben alle durch die Stadt führenden Straßenabschnitte ein Tempolimit von max. 80 km/h?

```
SELECT S.Stadtname FROM Städte S  
WHERE S.Stadtgrenze.Fläche() > 30  
AND ( FOR ALL A IN S.LiegtAn : A.Tempolimit <= 80 )
```

4) In welchen Ländern mit mehr als 100 Millionen Einwohnern haben Städte der Elfenbeinküste Partnerstädte?

```
SELECT DISTINCT PL  
FROM ( SELECT FLATTEN (L.HatStädte.Partnerstädte.GehörtZuL)  
      FROM L IN Länder  
      WHERE L.Landesbez="Elfenbeinküste" ) AS PL  
WHERE PL.Bevölkerung() > 100000000
```

Beispiele: Methodenaufrufe in OQL (2)

- 5) Bestimmung aller Straßen mit einem Stützpunkt innerhalb des Rechtecks mit der linken unteren Ecke (1,2) und der rechten oberen Ecke (5,8)

```
SELECT S.StrBez
FROM Straßen S
WHERE S.HatAbschnitte.Verlauf.HatStützpunkte.LiegtInBox (
    STRUCT(lu:STRUCT(x:1,y:2), ro:STRUCT(x:5,y:8)) )
```

- 6) Bestimmung aller Straßen, die das Rechteck mit der linken unteren Ecke (1,2) und der rechten oberen Ecke (5,8) schneiden

```
SELECT S.StrBez
FROM Straßen S
WHERE S.HatAbschnitte.Verlauf.SchneidetBox (
    STRUCT(lu:STRUCT(x:1,y:2), ro:STRUCT(x:5,y:8)) )
```

Beispiele: Generalisierungshierarchie in OQL

1) Welche Polylinien, inkl. Polygonen, schneiden das Rechteck ... ?

```
SELECT P FROM Polylinien P  
WHERE P.SchneidetBox ( STRUCT(lu:STRUCT(x:1,y:2), ro:STRUCT(x:5,y:8)) )
```

2) Das niedrigste Mindesttempo auf einer Straße von mehr als 1000 km Gesamtlänge?

```
SELECT MIN ( ((Autobahn) S).Mindesttempo ) FROM Straßen S  
WHERE S.Gesamtlänge() > 1000
```

3) Autobahnen über die Mosel mit einem Abschnitt mit Tempolimit 60 km/h ?

```
SELECT A.GehörtZuStr.StrBez  
FROM ( SELECT F.FließtUnter.GehörtZuA FROM Flüsse F  
        WHERE F.Flußbez = "Mosel" ) AS A  
WHERE ( EXISTS B IN Autobahnen: B.StrBez = A.GehörtZuStr.StrBez )  
AND A.Tempolimit = 60
```

oder:

```
SELECT ((Autobahn) A.GehörtZuStr).StrBez  
FROM ( SELECT F.FließtUnter.GehörtZuA FROM Flüsse F  
        WHERE F.Flußbez = "Mosel" ) AS A  
WHERE A.Tempolimit = 60
```

Erweiterte Gruppierung in OQL

Resultat von ... GROUP BY G1: Gruppierungsattribut1, ...
hat den Typ set<struct(G1: type_of(Gruppierungsattribut1), ...),
partition: bag<type_of(Nichtgruppierungsattribute)>>

Beispiel: Gesamteinwohnerzahl aller Städte an der E12 pro Land

SELECT Land,

Gesamteinwohner: SUM(SELECT R.Einwohnerzahl FROM partition),

Stadteinwohner: (SELECT R.Stadtname, R. Einwohnerzahl FROM partition)

FROM (SELECT S FROM Städte S

WHERE S.LiegtAn.GehörtZu.StrBez = "E12") AS R

GROUP BY Land: R.GehörtZuL

<i>Land</i>	<i>Gesamteinwohner</i>	<i>Stadteinwohner</i>	
Deutschland	450 000	<i>Stadtname</i>	<i>Einwohnerzahl</i>
		Kaiserslautern	200 000
		Saarbrücken	250 000
Frankreich	5 350 000	<i>Stadtname</i>	<i>Einwohnerzahl</i>
		Metz	200 000
		Reims	150 000
		Paris	5 000 000

Erweiterte Aggregation in OQL

Anwendungsspezifische Aggregationsfunktionen

Beispiel:

```
class Zahlen { attribute bag<Zahl: real> Zahlenmultimenge;  
               Real mean(); Real median(); Real stddev(); }
```

a) Mittelwert und Standardabweichung der Bestellungssummen

```
SELECT Z.mean(), Z.stddev()  
FROM ( SELECT B.Summe FROM Bestellungen B ) AS Z
```

b) Mittelwert und Standardabweichung der Bestellungssummen
für jedes einzelne Produkt

```
SELECT G.Prod, G.Best.mean(), G.Best.stddev()  
FROM ( SELECT Prod, (SELECT B.Summe FROM partition)  
          AS Best  
        FROM Bestellungen B  
        GROUP BY Prod: B.PNr ) AS G
```

8.5.1 Komplexe Datentypen in SQL-99

Typkonstruktoren ROW, SET, LIST, MULTISSET
Subklassen UNDER Superklasse

Beispiele:

```
CREATE ROW TYPE Punkt (X FLOAT, Y FLOAT);
```

```
CREATE ROW TYPE Polylinie
```

```
  (AnzPunkte INTEGER, HatStützpunkte LIST OF Punkt);
```

```
CREATE ROW TYPE Straße
```

```
  (StrBez VARCHAR(10), Ausgangsort VARCHAR(30), Zielort VARCHAR(30),  
   HatVerlauf Polylinie);
```

```
CREATE TABLE Straßen OF TYPE Straße;
```

```
CREATE TYPE Autobahn UNDER Straße
```

```
  (Mindesttempo FLOAT);
```

```
CREATE TABLE Autobahnen OF Autobahn;
```

```
SELECT * FROM Autobahnen
```

liefert alle Autobahnen

```
SELECT * FROM Straßen
```

liefert alle Straßen inklusive der Autobahnen

Komplexe Datentypen in Oracle8i

Typkonstruktoren OBJECT, REF, VARRAY (ohne Schachtelung)
TABLE (mit 1 Stufe von NESTED TABLES)

Beispiele:

```
CREATE TYPE Kenntnissetyp AS VARRAY(5) OF VARCHAR(30);
CREATE TABLE Angestellte
  (AngNr NUMBER, Name VARCHAR(50), Kenntnisse Kenntnissetyp);
INSERT INTO Angestellte VALUES (0815, 'Heinz Becker',
                                Kenntnissetyp ('Oracle', 'Java', 'Urpils') );
```

```
CREATE TYPE Pruefungstyp AS OBJECT
  (Fach VARCHAR(30), Datum DATE, Note NUMBER);
CREATE TYPE Pruefungstabellentyp AS TABLE OF Pruefungstyp;
CREATE TABLE Studenten
  (MatrNr NUMBER, Name VARCHAR(30),
   AbgelegtePruefungen Pruefungstabellentyp)
  NESTED TABLE AbgelegtePruefungen STORE AS Pruefungstabelle;
INSERT INTO Studenten VALUES (4711, 'Jacques Bistro',
                               Pruefungstabellentyp (Pruefungstyp ('Praktische Informatik', 11.11.1998, 1.3),
                                                       Pruefungstyp ('Nebenfach', 24.12.1998, 1.7) ) );
```

Mehrfache Schachtelung in Oracle8i

Beispiel:

```
CREATE TYPE Frageantworttyp AS OBJECT
  (LfdNr NUMBER, Frage VARCHAR(200), Antwort VARCHAR(2000));
CREATE TYPE Protokolltyp AS TABLE OF Frageantworttyp;
CREATE TYPE Pruefungstyp AS OBJECT
  (Fach VARCHAR(30), Datum DATE, Note NUMBER, Protokoll Protokolltyp);
CREATE TYPE Pruefungstabellentyp AS TABLE OF Pruefungstyp;
CREATE TABLE Studenten
  (MatrNr NUMBER, Name VARCHAR(30),
   AbgelegtePruefungen Pruefungstabellentyp)
  NESTED TABLE AbgelegtePruefungen STORE AS Pruefungstabelle;
INSERT INTO Studenten VALUES (4711, 'Jacques Bistro',
  Pruefungstabellentyp
    (Pruefungstyp ('Praktische Informatik', 11.11.1998, 1.3,
      Protokolltyp (
        Frageantworttyp (1, 'Wie baut man ... ?', 'Das macht ... '),
        Frageantworttyp (2, 'Wieso ist ...? ', 'Der Grund ist ... '))),
    Pruefungstyp ('Nebenfach', 24.12.1998, 1.7,
      Protokolltyp (
        Frageantworttyp (1, 'Was ist ... ? ', 'Das ist ... '),
        Frageantworttyp (2, 'Wie funktioniert ...? ', 'Das ... '))) ));
```


Oracle8i: Anfragen auf Nested Tables (1)

Beispiele:

```
CREATE TYPE Namenstyp AS OBJECT
  (Vorname VARCHAR(30), Nachname VARCHAR(30), Spitzname VARCHAR(30))
CREATE TYPE Pruefungstyp AS OBJECT
  (Fach VARCHAR(30), Datum DATE, Note NUMBER);
CREATE TYPE Pruefungstabellentyp AS TABLE OF Pruefungstyp;
CREATE TABLE Studenten
  (MatrNr NUMBER, Name Namenstyp,
  AbgelegtePruefungen Pruefungstabellentyp)
  NESTED TABLE AbgelegtePruefungen STORE AS Pruefungstabelle;
```

Oracle8i: Anfragen auf Nested Tables (2)

1) Vollständige Namen von Studenten mit Spitznamen „Schlumpf“

```
SELECT S.Name.Vorname, S.Name.Nachname FROM Studenten S  
WHERE S.Name.Spitzname = 'Schlumpf'
```

2) Gib alle Prüfungen des Studenten mit Matrikelnummer 1234 aus

```
SELECT S.AbgelegtePruefungen FROM Studenten S WHERE S.MatrNr = 1234
```

bzw. besser:

```
SELECT * FROM TABLE ( SELECT S.AbgelegtePruefungen  
                        FROM Studenten S WHERE S.MatrNr = 1234 )
```

3) Note des Studenten mit Matrikelnr. 1234 im Fach Informationssysteme

```
SELECT P.Note  
FROM TABLE ( SELECT S.AbgelegtePruefungen FROM Studenten S  
              WHERE S.MatrNr = 123456 ) P  
WHERE P.Fach = 'Informationssysteme';
```

Oracle8i: Anfragen auf Nested Tables (3)

4) Gib die Noten aller Studenten in allen Fächern aus.

```
SELECT S.MatrNr, P.Fach, P.Note  
FROM Studenten S, TABLE (S.AbgelegtePruefungen) P
```

5) Gib die Noten aller Studenten im Fach Informationssysteme aus.

```
SELECT S.MatrNr, P.Note  
FROM Studenten S, TABLE (S.AbgelegtePruefungen) P  
WHERE P.Fach = 'Informationssysteme';
```

6) Gib für jeden Studenten das letzte Prüfungsdatum aus
bzw. einen Nullwert für Studenten ohne abgelegte Prüfungen

```
SELECT S.MatrNr, Max(P.Datum)  
FROM Studenten S, TABLE (S.AbgelegtePruefungen) (+) P
```

oder:

```
SELECT S.MatrNr,  
       (SELECT Max(P.Datum) FROM TABLE(S.AbgelegtePruefungen))  
FROM Studenten S
```

Funktionen und ADTs in SQL-99

Beispiele:

```
CREATE TYPE Rechteck (lu Punkt, ro Punkt);  
CREATE FUNCTION SchneidetBox (p Polylinie, r Rechteck) RETURNS BOOLEAN  
BEGIN ... END;  
CREATE FUNCTION Gesamtlänge (p Polylinie) RETURNS FLOAT  
BEGIN ... END;
```

```
SELECT S.StrBez, Gesamtlänge(S.HatVerlauf) FROM Straßen S  
WHERE S.Ausgangsort='Saarbrücken'  
AND SchneidetBox (S.HatVerlauf, <lu: <X:5.71, Y:6.24>, ro: <X:17.5, Y:22>>);
```

```
CREATE TYPE Straße  
(StrBez VARCHAR(10), Ausgangsort VARCHAR(30), Zielort VARCHAR(30),  
PRIVATE HatVerlauf Polylinie,  
PUBLIC FUNCTION Gesamtlänge (s Straße) RETURNS FLOAT  
...  
END FUNCTION );  
CREATE TABLE Straßen OF TYPE Straße;
```

Funktionen und ADTs in Oracle8i

Beispiele:

```
CREATE TYPE Kontotyp AS OBJECT
```

```
  (KontoNr NUMBER, Inhaber VARCHAR(30), Kontostand MONEY,
```

```
  MEMBER FUNCTION Tageszinsen (Zinssatz IN NUMBER) RETURN MONEY,
```

```
  MEMBER PROCEDURE Einzahlung (Betrag IN MONEY),
```

```
  MEMBER PROCEDURE Auszahlung (Betrag IN MONEY) RETURN Fehlertyp );
```

```
CREATE OR REPLACE TYPE BODY Kontotyp AS
```

```
  MEMBER FUNCTION Tageszinsen (Zinssatz IN NUMBER) RETURN MONEY
```

```
    IS BEGIN ... RETURN Kontostand*Zinssatz/100; END;
```

```
  MEMBER PROCEDURE Einzahlung (...) IS BEGIN ... END; ...
```

```
END;
```

```
CREATE TABLE Konto OF Kontotyp;
```

```
SELECT KontoNr, Tageszinsen (0.05) FROM Konto WHERE Kontostand > 100000;
```

```
SELECT KontoNr FROM Konto WHERE Tageszinsen (0.05) > 1000;
```

```
UPDATE Konto K SET K = K.Einzahlung (100) WHERE KontoNr = 1234;
```

Kapitel 11: Relationale Entwurfstheorie

11.1 Funktionalabhängigkeiten

11.2 Normalformen

11.3 Relationendekomposition

11.4 Relationensynthese

11.5 Ergänzungen zum algorithmischen Ansatz

Problem

Bestellungen (Datum, KNr, PNr, Bez, Preis, Gewicht, Menge)

"Typische" Ausprägung:

Datum	KNr	PNr	Bez	Preis	Gewicht	Menge
16.7.	1	1	Papier	20.00	2.000	100
21.7.	1	1	Papier	20.00	2.000	200
26.10.	2	1	Papier	20.00	2.000	100
26.10.	2	5	Disketten	20.00	0.500	50

11.1 Funktionalabhängigkeiten (FAs, FDs)

Definition:

Für $X = \{X_1, \dots, X_m\} \subseteq \text{sch}(R)$ und $Y = \{Y_1, \dots, Y_n\} \subseteq \text{sch}(R)$ gilt die *Funktionalabhängigkeit* $X \rightarrow Y$, wenn jederzeit für je zwei Tupel $r, s \in \text{val}(R)$ gelten muß:

$$(r.X_1=s.X_1 \wedge \dots \wedge r.X_m=s.X_m) \Rightarrow (r.Y_1=s.Y_1 \wedge \dots \wedge r.Y_n=s.Y_n)$$

Beispiel:

Bestellungen (Datum, KNr, PNr, Bez, Preis, Gewicht, Menge)

Es gelten:

Datum, KNr, PNR $\rightarrow$ Menge

PNr $\rightarrow$ Bez, Preis, Gewicht

Es gelten *nicht*:

KNr, PNR $\rightarrow$ Menge

PNr $\rightarrow$ Datum

Entwurfsziel: Alle FAs in Primärschlüsselbedingungen ausdrücken

Ableitungsregeln für Funktionalabhängigkeiten

Definition:

Sei F eine Menge von FAs über $\text{sch}(R)$. Die **transitive Hülle** F^+ ist die Menge der aus F logisch ableitbaren FAs.

Ableitungskalkül (Armstrong-Regeln):

Seien $X \subseteq \text{sch}(R)$, $Y \subseteq \text{sch}(R)$, $Z \subseteq \text{sch}(R)$.

Regel 1 (Reflexivität): Wenn $Y \subseteq X$, dann gilt: $X \rightarrow Y$.

Regel 2 (Erweiterung): Wenn $X \rightarrow Y$, dann gilt: $XZ \rightarrow YZ$

Regel 3 (Transitivität): Wenn $X \rightarrow Y$ und $Y \rightarrow Z$, dann gilt: $X \rightarrow Z$.

Weitere Ableitungsregeln:

Regel 4 (Vereinigung): Wenn $X \rightarrow Y$ und $X \rightarrow Z$, dann gilt: $X \rightarrow YZ$.

Regel 5 (Pseudotrans.): Wenn $X \rightarrow Y$ und $WY \rightarrow Z$ mit $W \subseteq \text{sch}(R)$, dann gilt: $XW \rightarrow Z$.

Regel 6 (Zerlegung): Wenn $X \rightarrow Y$ und $Z \subseteq Y$, dann gilt: $X \rightarrow Z$.

Satz: Die Armstrong-Regeln bilden einen korrekten und vollständigen Ableitungskalkül für F^+ .

Beispiel: Ableitung von Funktionalabhängigkeiten

Bestellungen (Datum, KNr, PNr, Bez, Preis, Gewicht, Menge)

$F = \{ \text{Datum, KNr, PNR} \rightarrow \text{Menge},$
 $\text{PNr} \rightarrow \text{Bez, Preis, Gewicht} \}.$

Es folgen:

Datum, KNr, PNR $\rightarrow$ PNr
Bez, Preis, Gewicht $\rightarrow$ Bez
PNr $\rightarrow$ Bez
Datum, KNr, PNr $\rightarrow$ Bez

nach der Reflexivitätsregel,
nach der Reflexivitätsregel,
nach der Transitivitätsregel,
nach der Transitivitätsregel

Ableitbarkeitstest für FAs: Xplus-Algorithmus

Definition:

Sei R eine Relation mit $\text{sch}(R)$ und FA-Menge F , und sei $X \subseteq \text{sch}(R)$. Die *Hülle* X^+ von X ist die Menge aller $A \in \text{sch}(R)$ mit $X \rightarrow A \in F^+$.

Xplus-Algorithmus:

```
procedure xplus (X: set of attribute): set of attribute;  
var H: set of attribute;  
    newattr: Boolean;  
begin  
    H := X; newattr := true;  
    while newattr do (* Invariante:  $X \rightarrow H$  *)  
        newattr := false;  
        for each  $Y \rightarrow Z \in F$  do  
            if  $Y \subseteq H$  then  
                if not ( $Z \subseteq H$ ) then  $H := H \cup Z$ ; newattr := true; fi; fi;  
        od  
    od  
    return H;  
end xplus;
```

Beispiel: Xplus-Algorithmus

$\text{sch}(R) = \{A, B, C, D, E, G\}$

$F = \{AB \rightarrow C, ACD \rightarrow B,$
 $BC \rightarrow D, BE \rightarrow C,$
 $C \rightarrow A, CG \rightarrow BD, CE \rightarrow AG,$
 $D \rightarrow EG\}$

$X = \{B, D\}$

Bestimmung aller Schlüsselkandidaten

Definition:

Sei R eine Relation mit $\text{sch}(R)$ und FA-Menge F , und sei $X \subseteq \text{sch}(R)$.

$X \subseteq \text{sch}(R)$ ist ein *Schlüsselkandidat*, wenn $X \rightarrow \text{sch}(R) \in F^+$ gilt und es keine echte Teilmenge von X gibt, die diese Eigenschaft hat.

$A \in \text{sch}(R)$ ist Schl.attribut, wenn es in einem Schl.kandidaten vorkommt.

procedure findkeys: **set of set of** attribute;

var K : **set of set of** attribute; iskey: Boolean;

$K := \emptyset$;

for each $S \in 2^{\text{sch}(R)}$ **do**

 iskey := true;

if $\text{xplus}(S) \neq \text{sch}(R)$ **then** iskey := false **fi**;

for each $A \in S$ **while** iskey **do**

if $\text{xplus}(S - \{A\}) = \text{sch}(R)$ **then** iskey := false **fi**

od;

if iskey **then** $K := K \cup \{S\}$ **fi**

od;

return K ;

11.2 Relationale Normalformen

Definition:

Sei R eine Relation mit $\text{sch}(R)$ und FA-Menge F .

R ist in **Boyce-Codd-Normalform (BCNF)**, wenn für jede FA $X \rightarrow A \in F^+$ mit $X \subseteq \text{sch}(R)$, $A \in \text{sch}(R)$ und $A \notin X$ gilt, daß X einen Schlüsselkandidaten enthält.

Definition:

Sei R eine Relation mit $\text{sch}(R)$ und FA-Menge F .

R ist in **3. Normalform (3NF)**, wenn für jede FA $X \rightarrow A \in F^+$ mit $X \subseteq \text{sch}(R)$, $A \in \text{sch}(R)$ und $A \notin X$ gilt, daß X einen Schlüsselkandidaten enthält oder A ein Schlüsselattribut ist.

Satz:

Jede Relation in BCNF ist auch in 3NF.

Algorithmus für BCNF-Test

```
procedure BCNFtest: Boolean;  
var isbcnf: Boolean;  
begin  
    isbcnf := true;  
    for each  $X \rightarrow A \in F$  while isbcnf do  
        if  $A \notin X$  then if  $XPlus(X) \neq sch(R)$  then isbcnf := false fi fi;  
    od  
    return isbcnf;  
end BCNFtest;
```

Beispiele: Normalformen

- 1) Best (Datum, KNr, PNr, Menge) mit
 $F = \{\text{Datum, KNr, PNr} \rightarrow \text{Menge}\}$
und
Prod (PNr, Bez, Preis, Gewicht) mit
 $F = \{\text{PNr} \rightarrow \text{Bez, Preis, Gewicht}\}$
- 2) Vorlesungen (Titel, Zeit, Raum) mit
 $F = \{\text{Titel} \rightarrow \text{Raum,}$
 $\text{Zeit, Raum} \rightarrow \text{Titel}\}$

Beispielausprägung:

Titel	Zeit	Raum
Datenbanken	Di 9-11	HS 2
Datenbanken	Do 9-11	HS 2
Compiler	Di 9-11	HS 3
Compiler	Mi 13-15	HS 3
Betriebssysteme	Mi 13-15	HS 2

11.3 Relationendekomposition

Definition:

Sei R eine Relation mit sch(R) und FA-Menge F.

Eine Zerlegung von R in R1 (X1), ..., Rk (Xk) mit $X_i \subseteq \text{sch}(R)$ und $X_1 \cup \dots \cup X_k = \text{sch}(R)$ heißt **abhängigkeitsbewahrend**, wenn gilt:

$$\left((F^+ \mid_{X_1})^+ \cup \dots \cup (F^+ \mid_{X_k})^+ \right)^+ = F^+$$

Definition:

Sei R eine Relation mit sch(R) und FA-Menge F.

Eine Zerlegung von R in R1 (X1), ..., Rk (Xk) mit $X_i \subseteq \text{sch}(R)$ und $X_1 \cup \dots \cup X_k = \text{sch}(R)$ heißt **(projektion-join-) verlustfrei**, wenn für alle möglichen Ausprägungen, die F erfüllen, gilt:

$$\pi[X_1](R) \bowtie \dots \bowtie \pi[X_k](R) = R.$$

Beispiele: Abhängigkeitsbewahrung

- 1) Bestellungen (Datum, KNr, PNr, Bez, Preis, Gewicht, Menge) mit
 $F = \{\text{Datum, KNr, PNR} \rightarrow \text{Menge},$
 $\text{PNr} \rightarrow \text{Bez, Preis, Gewicht}\}$

Zerlegung in

Best (Datum, KNr, PNr, Menge) mit

$F1 = \{\text{Datum, KNr, PNR} \rightarrow \text{Menge}\}$

und

Prod (PNr, Bez, Preis, Gewicht) mit

$F2 = \{\text{PNr} \rightarrow \text{Bez, Preis, Gewicht}\}$

- 2) Vorlesungen (Titel, Zeit, Raum) mit
 $F = \{\text{Titel} \rightarrow \text{Raum},$
 $\text{Zeit, Raum} \rightarrow \text{Titel}\}$

Zerlegung in

Vorlesungsräume (Titel, Raum) mit

$F1 = \{\text{Titel} \rightarrow \text{Raum}\}$

und

Vorlesungszeiten (Titel, Zeit) mit $F2 = \emptyset$

Beispiele: Verlustfreiheit

Bestellungen (Datum, KNr, PNr, Bez, Preis, Gewicht, Menge) mit
 $F = \{\text{Datum, PNr, KNr} \rightarrow \text{Menge, PNr} \rightarrow \text{Bez, Preis, Gewicht}\}$

Datum	KNr	PNr	Bez	Preis	Gewicht	Menge
16.7.	1	1	Papier	20.00	2.000	100
16.7.	1	5	Disketten	20.00	0.500	50

Zerlegung in:

Datum	KNr	PNr	Menge
16.7.	1	1	100
16.7.	1	5	50

und

Datum	Bez	Preis	Gewicht
16.7.	Papier	20.00	2.000
16.7.	Disketten	20.00	0.500

Verlustfreie BCNF-Dekomposition

Satz: Zu jeder Relation R gibt es eine abhängigkeitsbewahrende und verlustfreie Zerlegung von R in 3NF-Relationen.

Satz: Zu jeder Relation R gibt es eine verlustfreie Zerlegung von R in BCNF-Relationen..

Satz: Sei R eine Relation mit $\text{sch}(R)$ und einer FA-Menge F .

Sei $X \rightarrow Y \in F^+$ mit $X \cap Y = \emptyset$. Die Zerlegung von R in $R'(X, Y)$ und $R''(X, \text{sch}(R) - Y)$ ist verlustfrei.

Algorithmus:

Teste, ob R in BCNF ist

Falls R nicht in BCNF ist

(* es gibt eine Funktionalabhängigkeit $X \rightarrow Y \in F^+$ mit
 $X \cap Y = \emptyset$ und $X \rightarrow \text{sch}(R) \notin F^+$ *)

dann

zerlege R in $R'(X, Y)$ und $R''(X, \text{sch}(R) - Y)$

wiederhole den Zerlegungsalgorithmus für R' und R''

Beispiel: Relationendekomposition

R (FlugNr, Datum, Abflugzeit, FSNr, SitzNr, TicketNr, Name, Adresse, GNr, Gewicht)

F = {FlugNr, Datum $\rightarrow$ Abflugzeit, FSNr,
FlugNr, Datum, TicketNr $\rightarrow$ SitzNr,
TicketNr $\rightarrow$ Name, Adresse,
GNr $\rightarrow$ Gewicht }

Schlüsselkandidat:

FlugNr, Datum, TicketNr, GNr

Zerlegungsschritte:

1. Zerlegung von R entlang von FlugNr, Datum $\rightarrow$ Abflugzeit, FSNr:
R1 (FlugNr, Datum, Abflugzeit, FSNr) und
R2 (FlugNr, Datum, SitzNr, TicketNr, Name, Adresse, GNr, Gewicht)
2. Zerlegung von R2 entlang von FlugNr, Datum, TicketNr $\rightarrow$ SitzNr:
R21 (FlugNr, Datum, TicketNr, SitzNr) und
R22 (FlugNr, Datum, TicketNr, Name, Adresse, GNr, Gewicht)
3. Zerlegung von R22 entlang von TicketNr $\rightarrow$ Name, Adresse:
R221 (TicketNr, Name, Adresse) und
R222 (TicketNr, FlugNr, Datum, GNr, Gewicht)
4. Zerlegung von R222 entlang von GNr $\rightarrow$ Gewicht:
R2221 (GNr, Gewicht) und
R2222 (GNr, FlugNr, Datum, TicketNr)

11.4 Relationensynthese

Definition:

Sei eine Relation mit $\text{sch}(R)$ und FA-Menge F . Eine FA-Menge F' heißt **Überdeckung** (engl. cover) von F , wenn gilt: $(F')^+ = F^+$.

Definition:

Eine Überdeckung F' von F heißt **minimal**, wenn gilt:

- 1) In jeder FA von F' hat die rechte Seite ein Attribut.
- 2) Keine echte Teilmenge von F' ist eine Überdeckung von F .
- 3) Für jede FA $X \rightarrow A \in F'$ und jede echte Teilmenge X' von X
 $F'' := F' - \{X \rightarrow A\} \cup \{X' \rightarrow A\}$ keine Überdeckung mehr von F .

Beispiel: minimale Überdeckung

$\text{sch}(R) = \{ \text{Ang}(\text{estellten})\text{Nr}, \text{Name}, \text{Gehalt}, \text{Leistung}(\text{sbeurteilung}), \text{Tarif}(\text{klasse}), \text{Abt}(\text{eilungs})\text{Nr}, \text{ProjektNr}, \text{Projektleiter}, \text{Abt}(\text{eilungs})\text{leiter} \}$

$F = \{ \text{AngNr} \rightarrow \text{Name},$
 $\text{AngNr} \rightarrow \text{Gehalt},$
 $\text{AngNr} \rightarrow \text{Tarif},$
 $\text{AngNr} \rightarrow \text{AbtNr},$
 $\text{AngNr} \rightarrow \text{Abtleiter},$
 $\text{ProjektNr} \rightarrow \text{Projektleiter},$
 $\text{AbtNr} \rightarrow \text{Abtleiter},$
 $\text{AngNr}, \text{Abtleiter} \rightarrow \text{Leistung},$
 $\text{Tarif}, \text{Leistung} \rightarrow \text{Gehalt} \}$

Schritt 1: $\rightarrow \{ \text{AngNr} \rightarrow \text{Name},$
 $\text{AngNr} \rightarrow \text{Gehalt},$
 $\text{AngNr} \rightarrow \text{Tarif},$
 $\text{AngNr} \rightarrow \text{AbtNr},$
 $\text{AngNr} \rightarrow \text{Abtleiter},$
 $\text{ProjektNr} \rightarrow \text{Projektleiter},$
 $\text{AbtNr} \rightarrow \text{Abtleiter},$
 $\text{AngNr} \rightarrow \text{Leistung},$
 $\text{Tarif}, \text{Leistung} \rightarrow \text{Gehalt} \}$

Schritt 2: $\rightarrow \{ \text{AngNr} \rightarrow \text{Name},$
 $\text{AngNr} \rightarrow \text{Tarif},$
 $\text{AngNr} \rightarrow \text{AbtNr},$
 $\text{ProjektNr} \rightarrow \text{Projektleiter},$
 $\text{AbtNr} \rightarrow \text{Abtleiter},$
 $\text{AngNr} \rightarrow \text{Leistung},$
 $\text{Tarif}, \text{Leistung} \rightarrow \text{Gehalt} \} = F'$

Algorithmus zur Berechnung der Min-Cover

```
procedure mincover: set of FDs;  
  var G: set of FDs;  
  procedure xplus (arg1: set of FDs, arg2: set of attribute): set of attribute;  
  procedure reduce (arg: set of FDs): set of FDs;  
    var H: set of FDs;  
    H := arg;  
    for each  $X \rightarrow A \in H$  do  
      for each  $C \in X$  do  
         $H_{red} := H - \{X \rightarrow A\} \cup \{(X - \{C\}) \rightarrow A\};$   
        if  $A \in \text{xplus}(H, X - \{C\})$  then  $H := H_{red}$  fi;  
      od;  
    od;  
  return H;  
  
G := reduce(F);  
for each  $X \rightarrow A \in G$  do  
  G := G - {  $X \rightarrow A$  };  
  if not ( $A \in \text{xplus}(G, X)$ ) then  $G := G \cup \{X \rightarrow A\}$  fi;  
od;  
return G;
```


Algorithmus zur Relationensynthese

- Schritt 1: Berechne eine minimale Überdeckung F' von F .
- Schritt 2: Partitioniere die FAs in F' nach gleichen linken Seiten, d.h.
für jedes X_i fasse alle $X_i \rightarrow A_1, \dots, X_i \rightarrow A_{k_i}$ zusammen
- Schritt 3: Bilde für jede Partition von Schritt 2 eine
Relation R_i mit $\text{sch}(R_i) = X_i \cup \{A_i, \dots, A_{k_i}\}$.
- Schritt 4: Falls $\text{sch}(R) - (\cup_i \text{sch}(R_i))$ nichtleer ist, bilde
eine weitere Relation R' mit $\text{sch}(R') = \text{sch}(R) - (\cup_i \text{sch}(R_i))$.
- Schritt 5: Falls keine der Relationen R_i, R' einen Schlüssel von R enthält,
erzeuge eine Relation R'' , deren Schema ein Schlüssel von R ist.

Satz:

Der Algorithmus zur Relationensynthese erzeugt eine
abhängigkeitsbewahrende und verlustfreie 3NF-Zerlegung von R .

Beispiel: Relationensynthese

$\text{sch}(R) = \{ \text{Ang}(\text{estellten})\text{Nr}, \text{Name}, \text{Gehalt},$
 $\text{Leistung}(\text{sbeurteilung}), \text{Tarif}(\text{klasse}),$
 $\text{Abt}(\text{eilungs})\text{Nr}, \text{ProjektNr}, \text{Projektleiter}, \text{Abt}(\text{eilungs})\text{leiter} \}$

$F' = \{ \text{AngNr} \rightarrow \text{Name},$
 $\text{AngNr} \rightarrow \text{Tarif},$
 $\text{AngNr} \rightarrow \text{AbtNr},$
 $\text{ProjektNr} \rightarrow \text{Projektleiter},$
 $\text{AbtNr} \rightarrow \text{Abtleiter},$
 $\text{AngNr} \rightarrow \text{Leistung},$
 $\text{Tarif}, \text{Leistung} \rightarrow \text{Gehalt} \}$

Ergebnis der Relationensynthese:

Angestellte (AngNr, Name, Tarif, Leistung, AbtNr)

Projekte (ProjektNr, Projektleiter)

Abteilungen (AbtNr, Abtleiter)

Lohn (Tarif, Leistung, Gehalt)

Projektmitarbeit (AngNr, ProjektNr)

11.5 Ergänzungen (1)

1) weitere Zerlegung von BCNF-Relationen:

Angestellte (ANr, Informatikkenntnisse, Kinder) mit $F = \emptyset$

ANr	Informatikkenntnisse	Kinder
1	Leda	Sabrina
1	Oracle	Sabrina
2	Leda	Pierre
2	Leda	Sabrina

sollte weiter zerlegt werden in

AngInfKenntnisse (ANr, Informatikkenntnisse) und
AngKinder (ANr, Kinder)

2) unnötig feine Zerlegungen:

Produkte (PNr, Bez, Preis) mit $F = \{PNr \rightarrow Bez, PNr \rightarrow Preis\}$

sollte nicht zerlegt werden in

Produktbez (PNr, Bez) und
Produktpreise (PNr, Preis)

11.5 Ergänzungen (2)

3) Zerlegung aufgrund irrelevanter Funktionalabhängigkeiten:

Bestellungen (Datum, KNr, PNr, Bez, Preis, Gewicht, Menge, Summe)

mit $F = \{ \text{Datum, KNr, PNr} \rightarrow \text{Menge},$

$\text{PNr} \rightarrow \text{Bez, Preis, Gewicht},$

$\text{Preis, Menge} \rightarrow \text{Summe} \}$

sollte nicht zur Relatione führen

Preise (Preis, Menge, Summe)

4) Nicht-BCNF-Relationen, die unproblematisch sind:

Kunden (KNr, Name, Postleitzahl, Stadt, Strasse)

mit $F = \{ \text{KNr} \rightarrow \text{Name, Postleitzahl, Stadt, Strasse},$

$\text{Postleitzahl} \rightarrow \text{Stadt} \}$

muß nicht notwendigerweise zerlegt werden in

Kundenadressen (KNr, Name, Postleitzahl, Strasse) und

Postleitzahlenverzeichnis (Postleitzahl, Stadt)

5) M:N-Relationships ohne Attribute:

Sitze (FlugNr, Datum, SitzNr)

muß bei der Relationendekomposition ggf. hinzugefügt werden

Kapitel 12: Datenmodellierung mit ERM & UML

12.1 Datenmodellierung mit ERM

12.2 Entwurfsmethodologie

12.3 Abbildung ERM auf Relationenmodell

12.4 Datenmodellierung mit UML

12.5 Anforderungsanalyse und Entwurfsmethodologie

Einordnung

Ziel und Infrastruktur der Datenmodellierung:

Beschreibung der für ein IS relevanten Informationszusammenhänge der realen Welt in einer hinreichend präzisen und konzisen Weise unter Verwendung von:

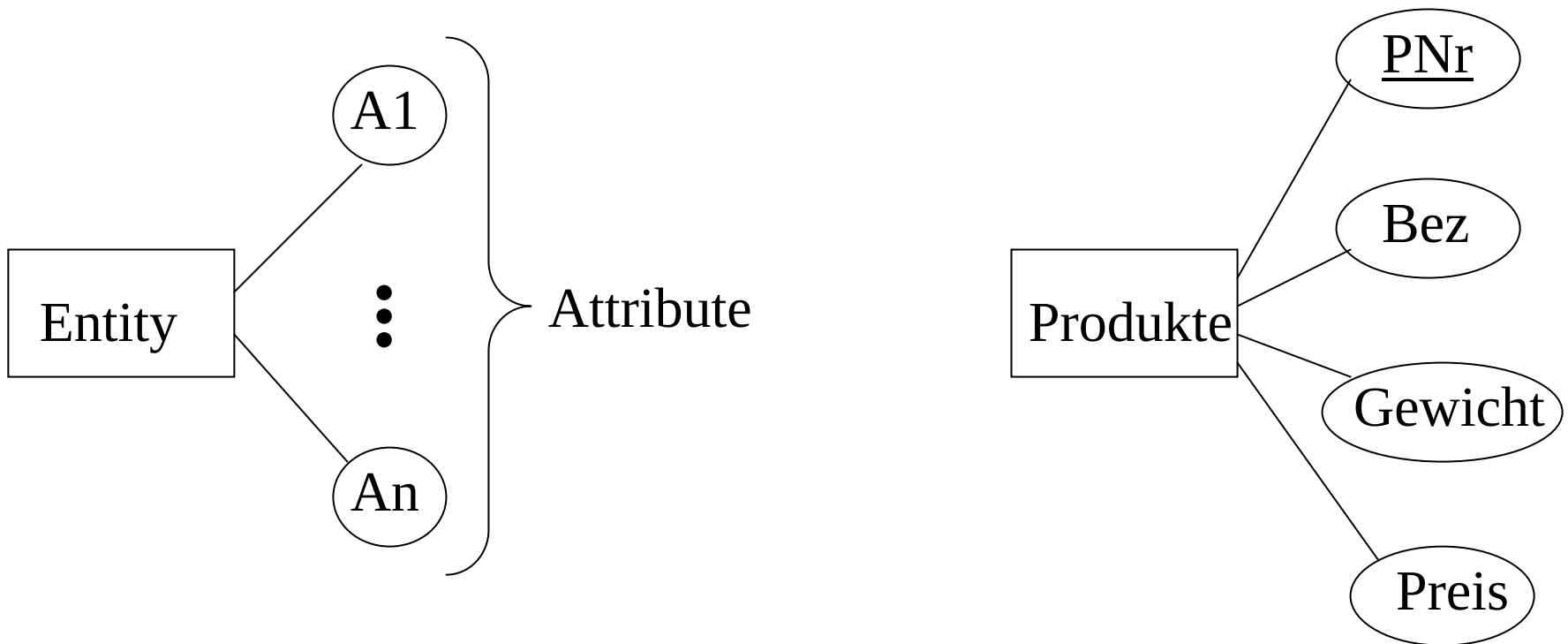
- einer „semantischen“ Modellierungssprache
- einer (i.d.R. informell-groben) Entwurfsmethodologie
- Softwarewerkzeugen (DB-Design-Tools, Repositories)

Grobe Vorgehensweise bei der Entwicklung von IS:

- 1) Anforderungsanalyse
- 2) Datenmodellierung
(plus Funktions- und Prozessmodellierung,
→ Kapitel 13 sowie Vorlesungen über Softwaretechnik)
- 3) Abbildung des „semantischen“ Datenmodells
auf (relationale) DB-Schema

12.1 Entities (Objekte) und Entity-Mengen

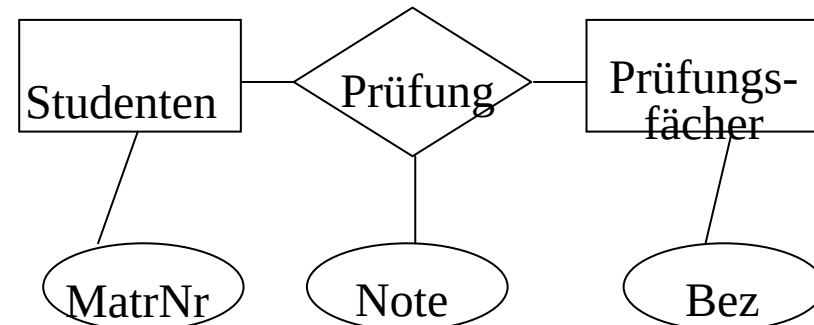
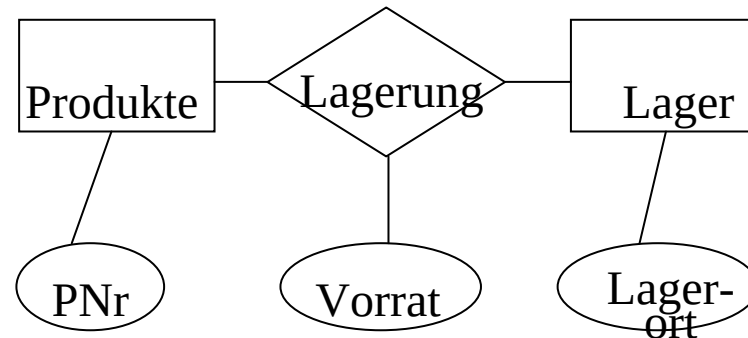
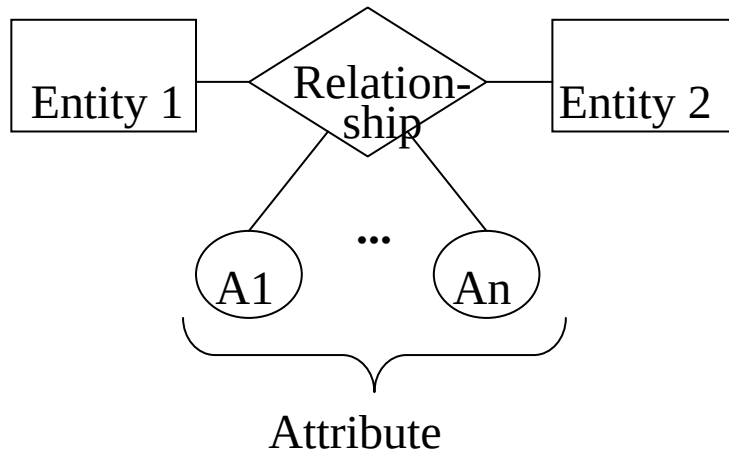
Ein **Entity** ist ein durch **Attribute** beschriebenes Objekt der realen Welt. Gleichartige Entities werden zu einem Entity-Typ zusammengefasst. Entities desselben Typs bilden eine **Entity-Menge**. Eine minimale Attributmenge, die jedes Entity einer Entity-Menge eindeutig identifiziert, heißt **Schlüssel(kandidat)**.



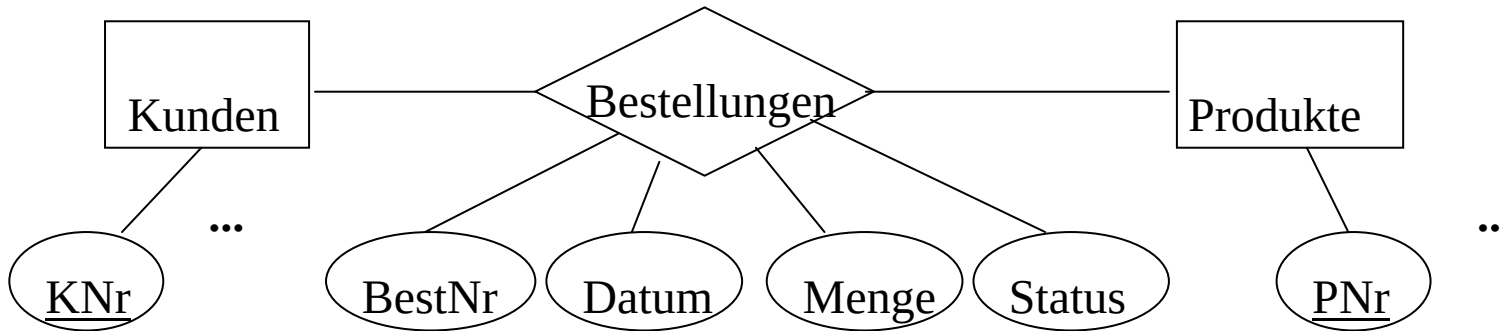
Relationships (Beziehungen, Assoziationen)

Eine **Relationship-Menge** R zwischen Entity-Mengen $E_1, \dots, E_n$ ist eine Teilmenge von $E_1 \times \dots \times E_n$. Ein Element von $\langle e_1, \dots, e_n \rangle$ von R heißt **Relationship** zwischen den Entities $e_1, \dots, e_n$.

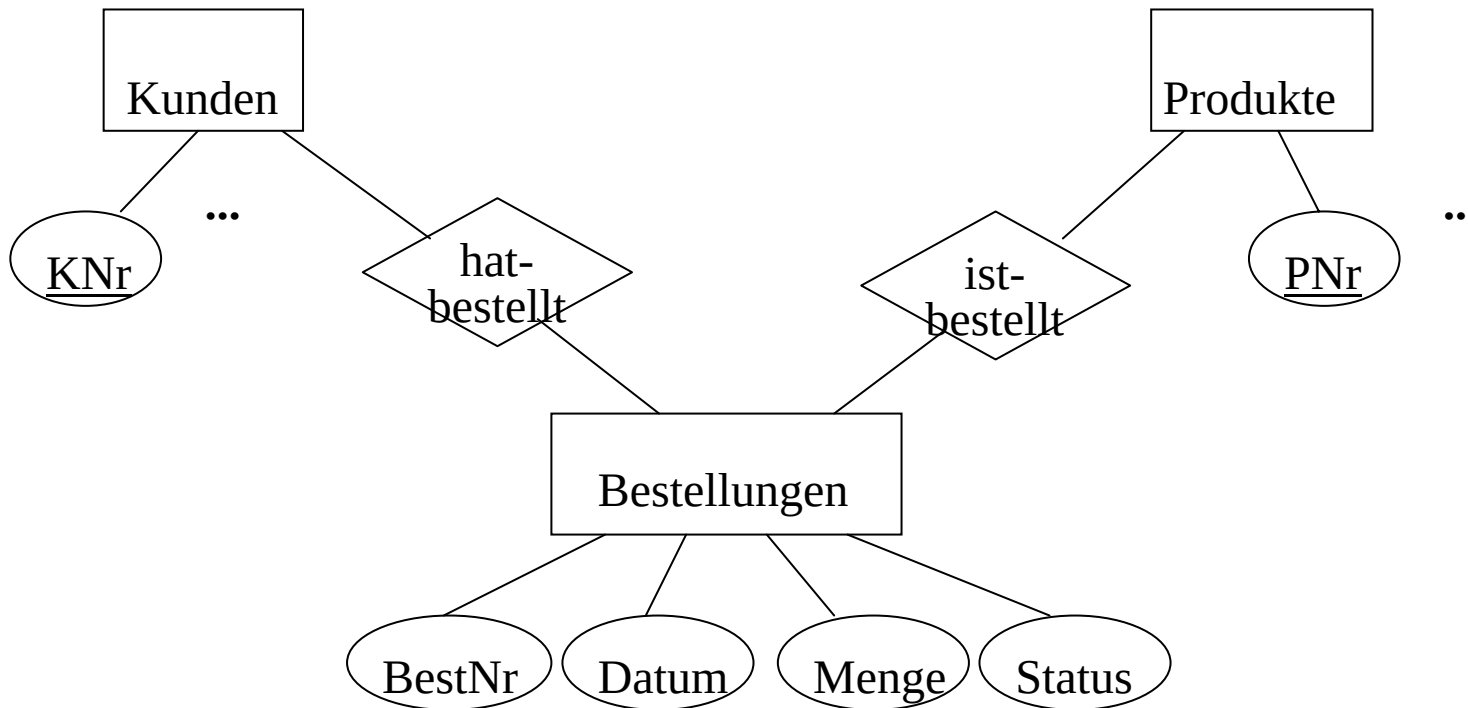
Eine minimale Teilmenge der Schlüssel von $E_1, \dots, E_n$, die jede Relationship in R eindeutig identifiziert, heißt **Schlüssel(kandidat)**.



Entity oder Relationship ? (Varianten 1 und 2)



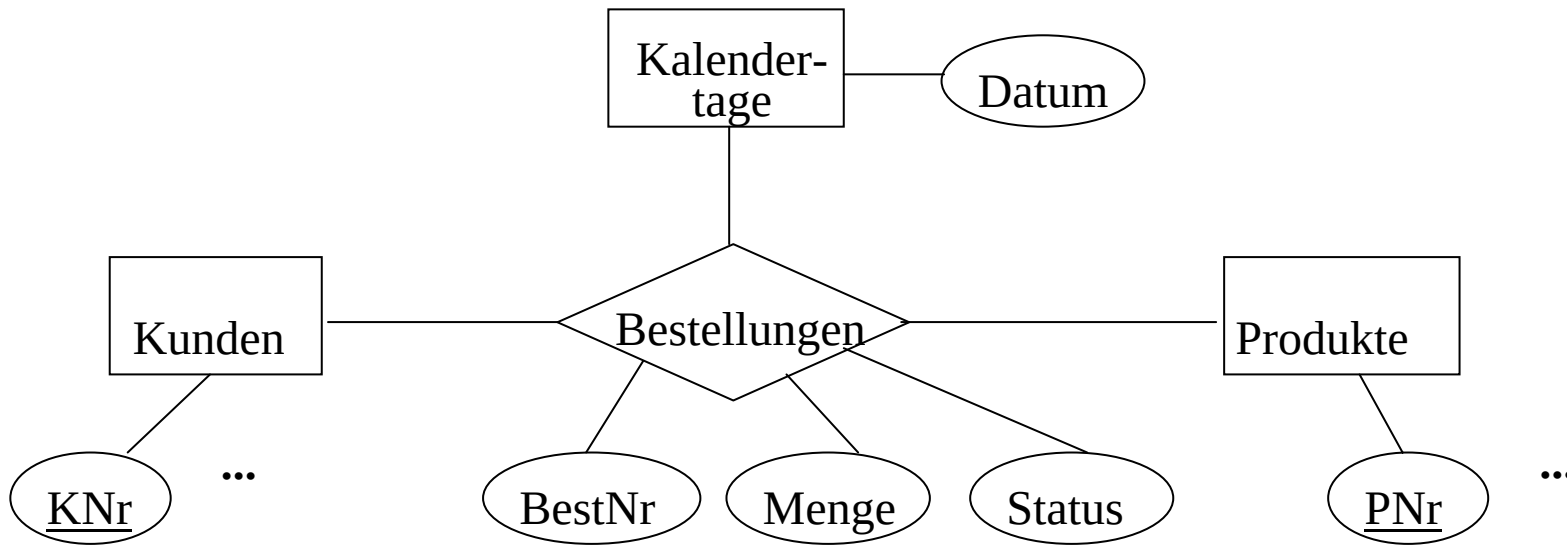
Variante 1



Variante 2

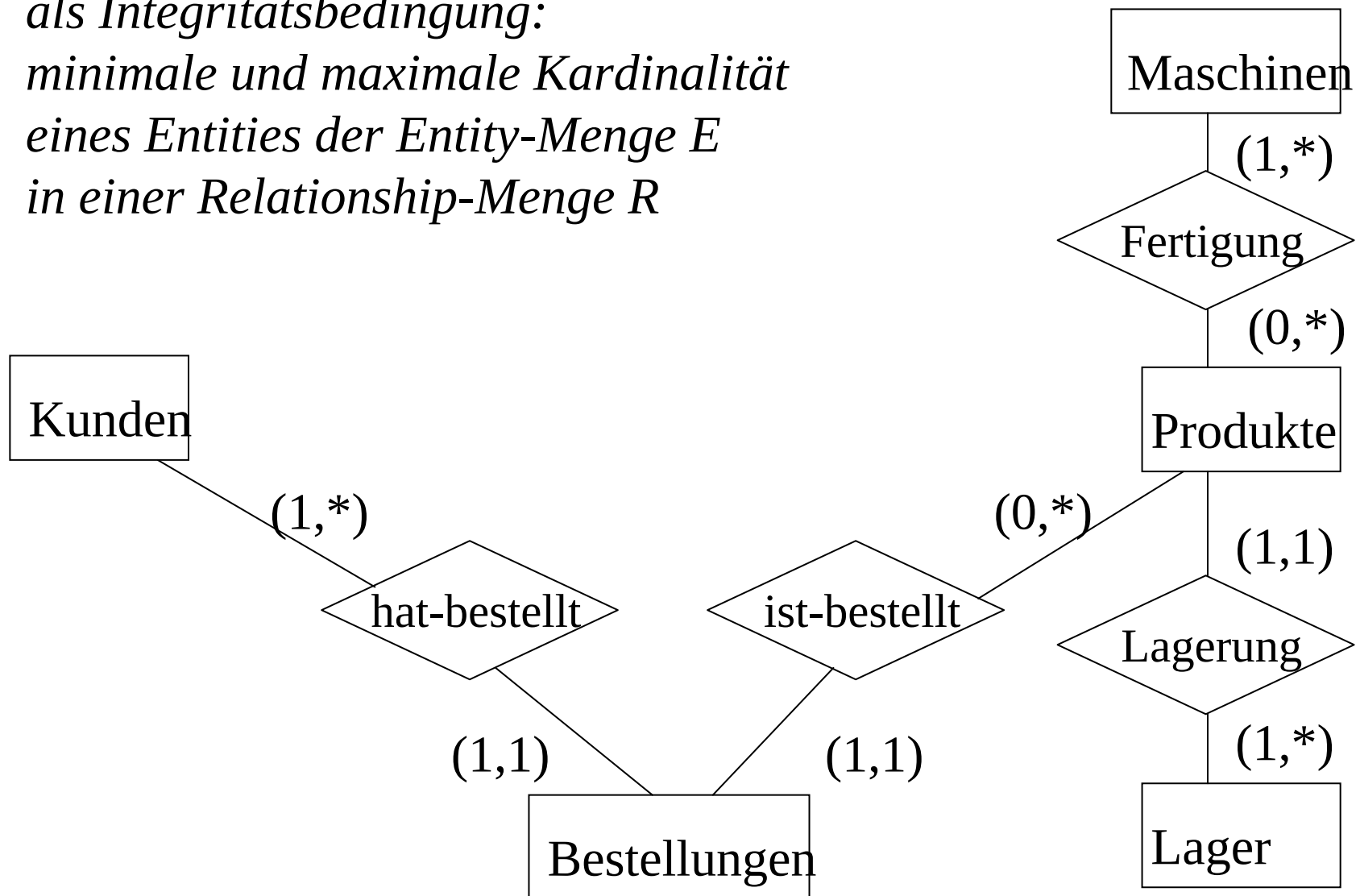
Entity oder Relationship ? (Variante 3)

Variante 3

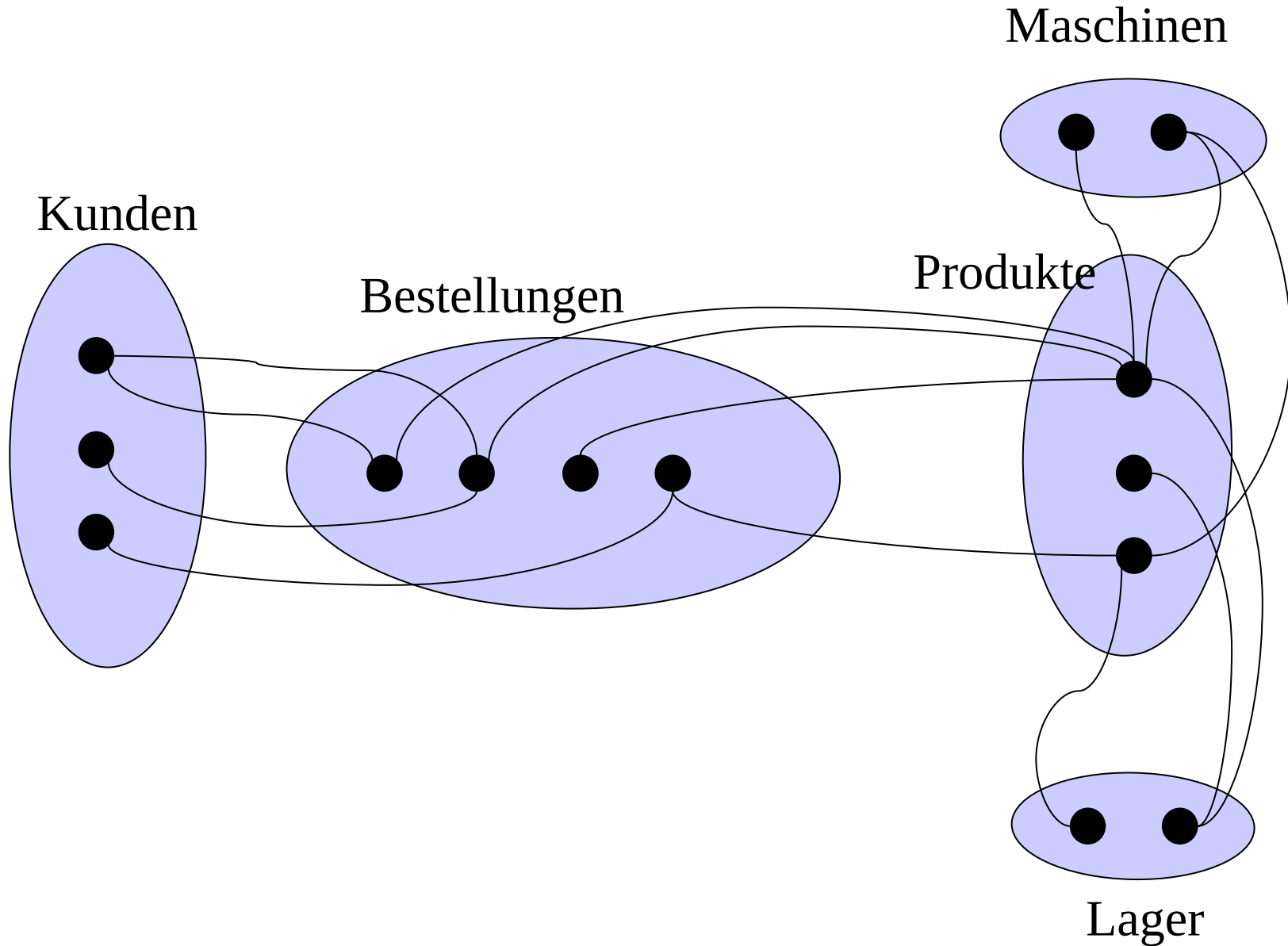


Kardinalitätsbedingungen für Relationships (1)

*als Integritätsbedingung:
minimale und maximale Kardinalität
eines Entities der Entity-Menge E
in einer Relationship-Menge R*

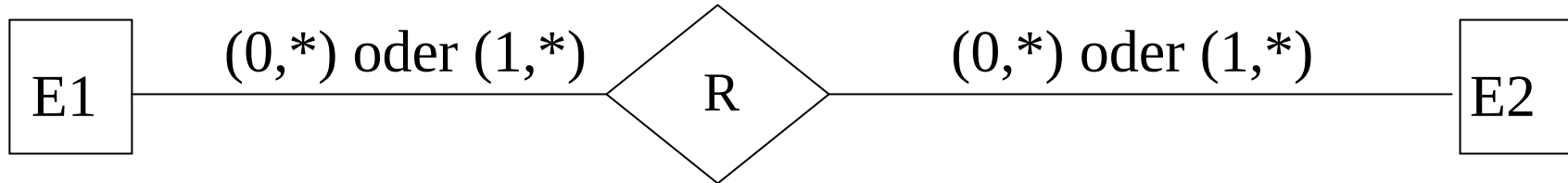


Kardinalitätsbedingungen für Relationships (2)

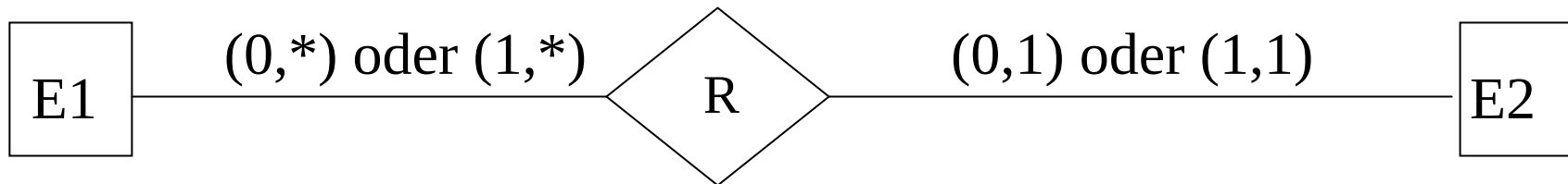


Typen von Kardinalitätsbedingungen (1)

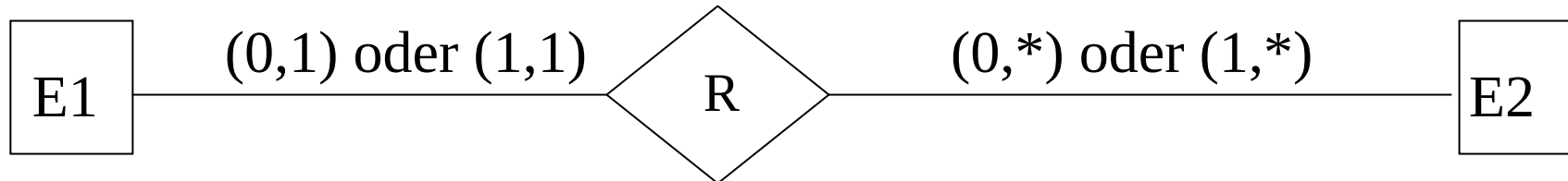
M:N-Relationship



1:N-Relationship

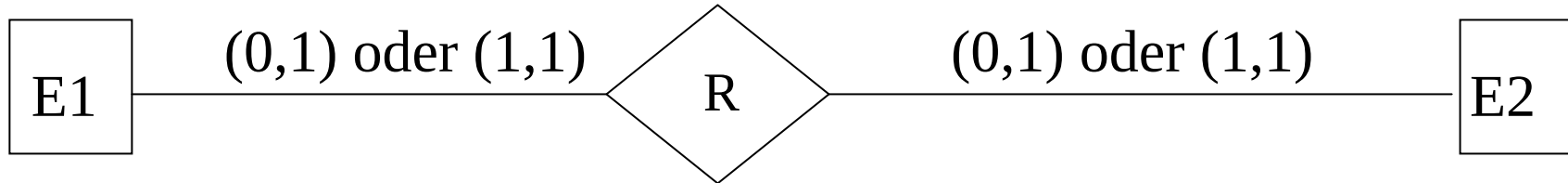


N:1-Relationship

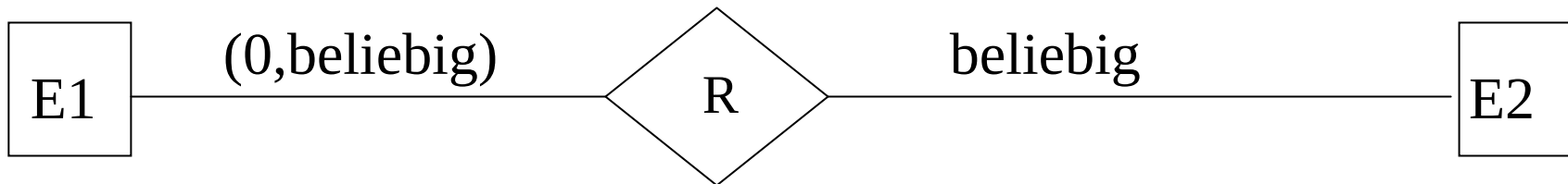


Typen von Kardinalitätsbedingungen (2)

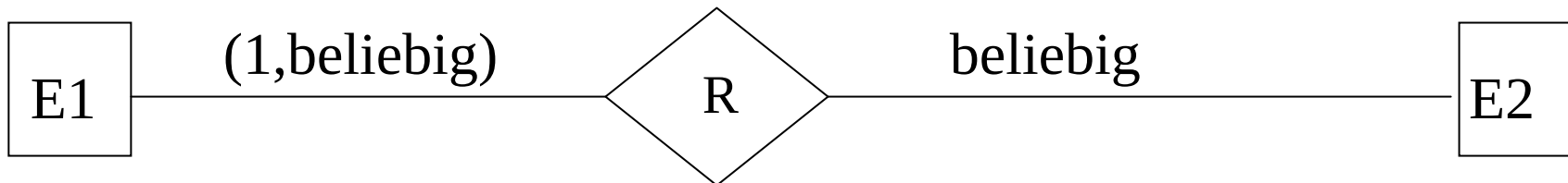
1:1-Relationship



Optionale Rolle von E1 in R

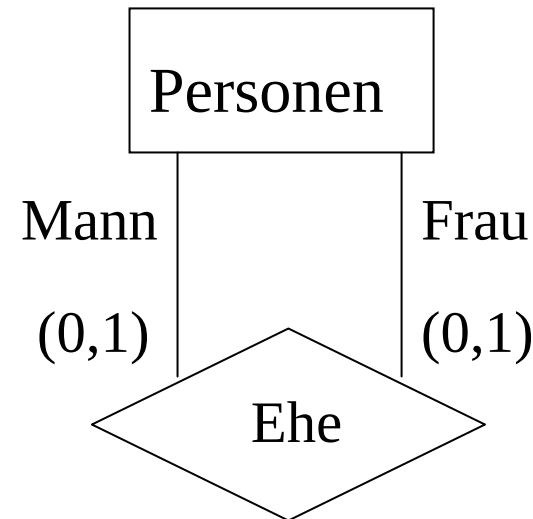
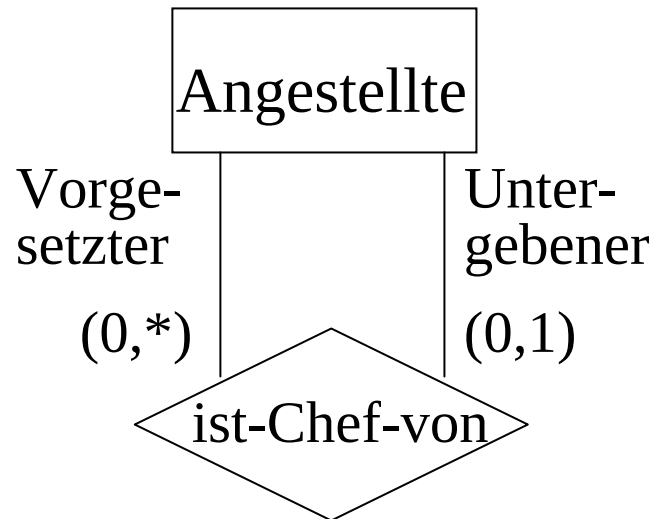


Verpflichtende Rolle von E1 in R

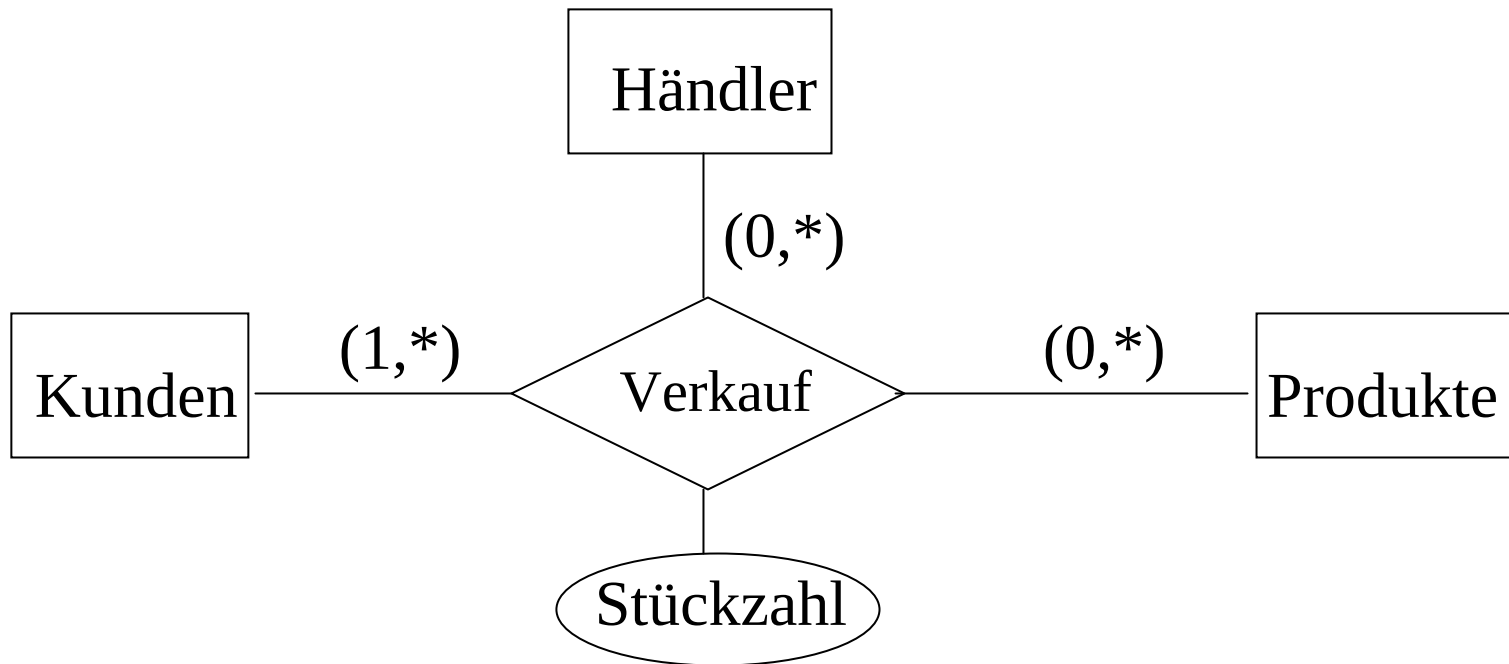


Rekursive Relationships

Relationships auf einer Menge erfordern Rollennamen

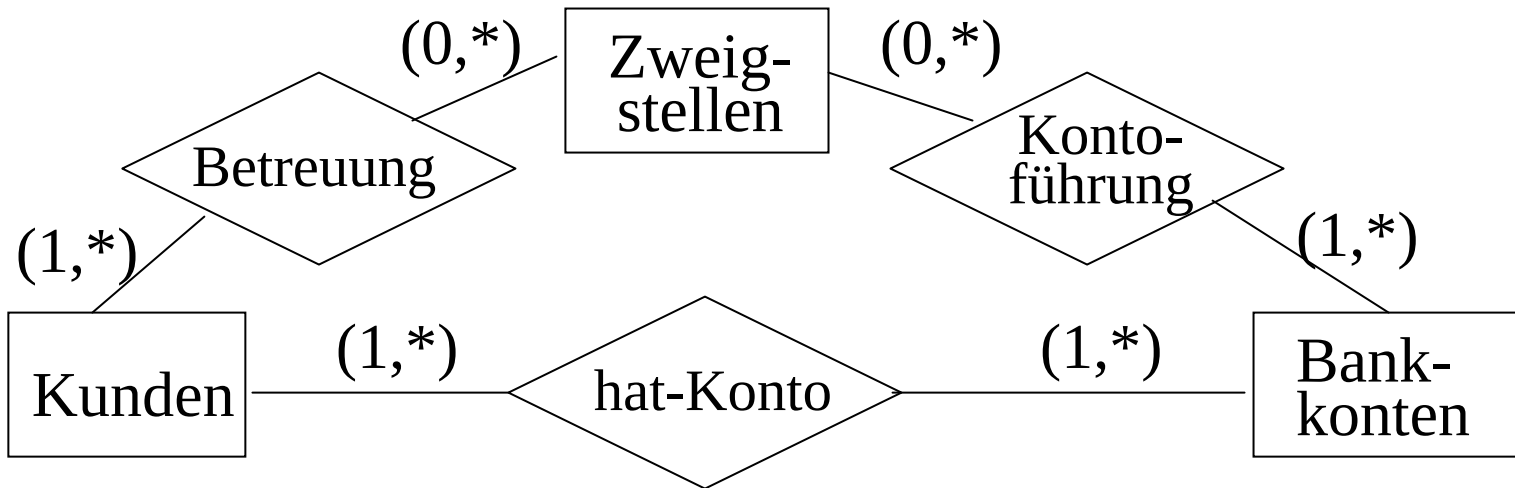
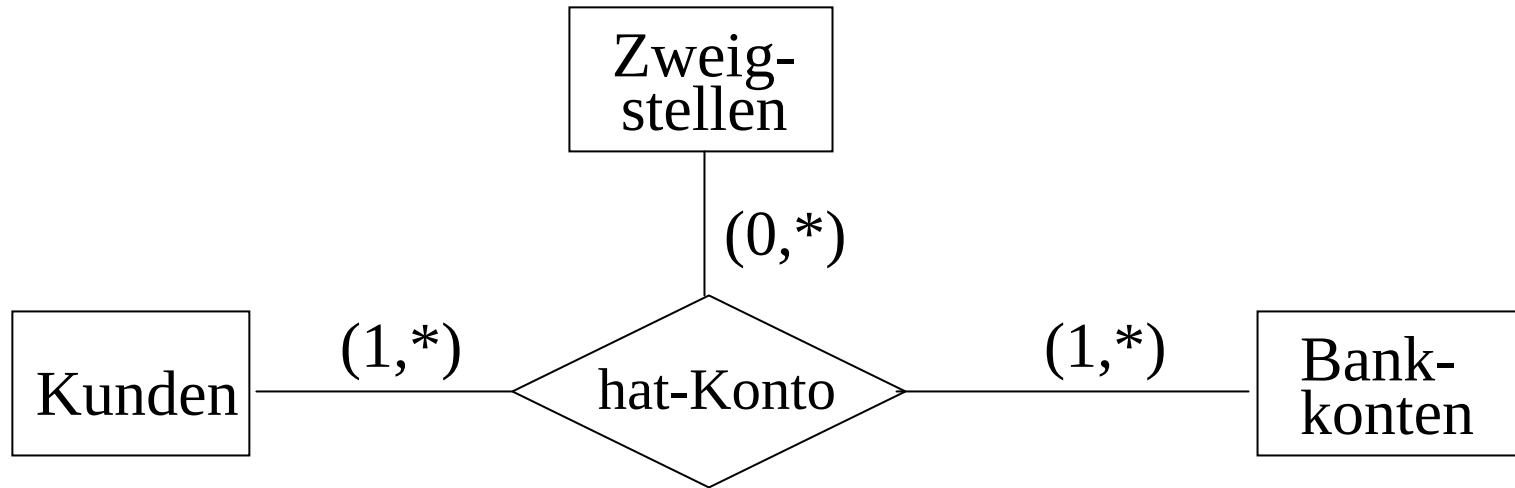


Ternäre Relationships: Beispiel 1

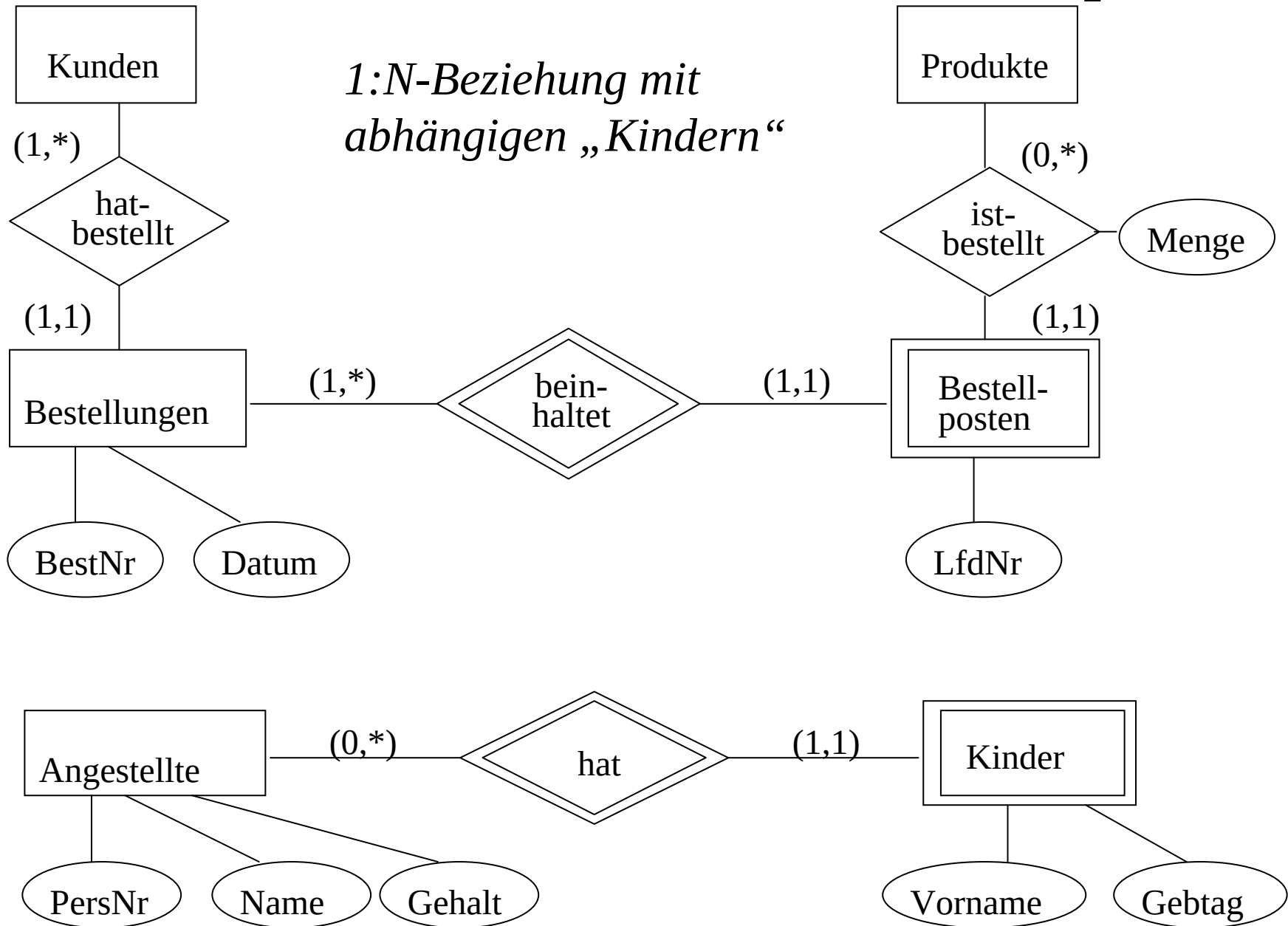


Achtung: Ternäre (und höherstellige) Relationships sind nicht immer zerlegbar in mehrere binäre Relationships

Ternäre Relationships: Beispiel 2

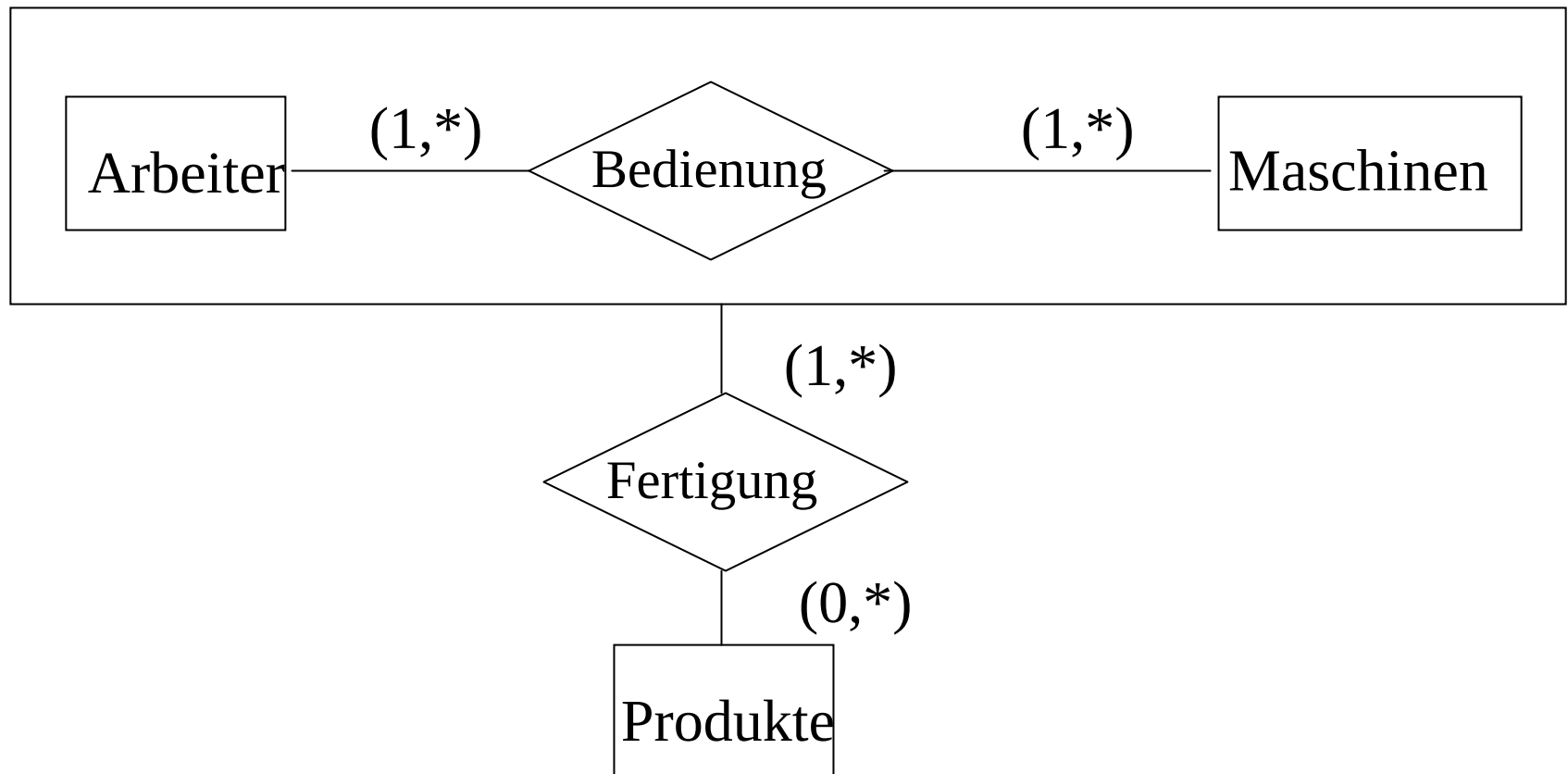


Schwache Entities und Relationships



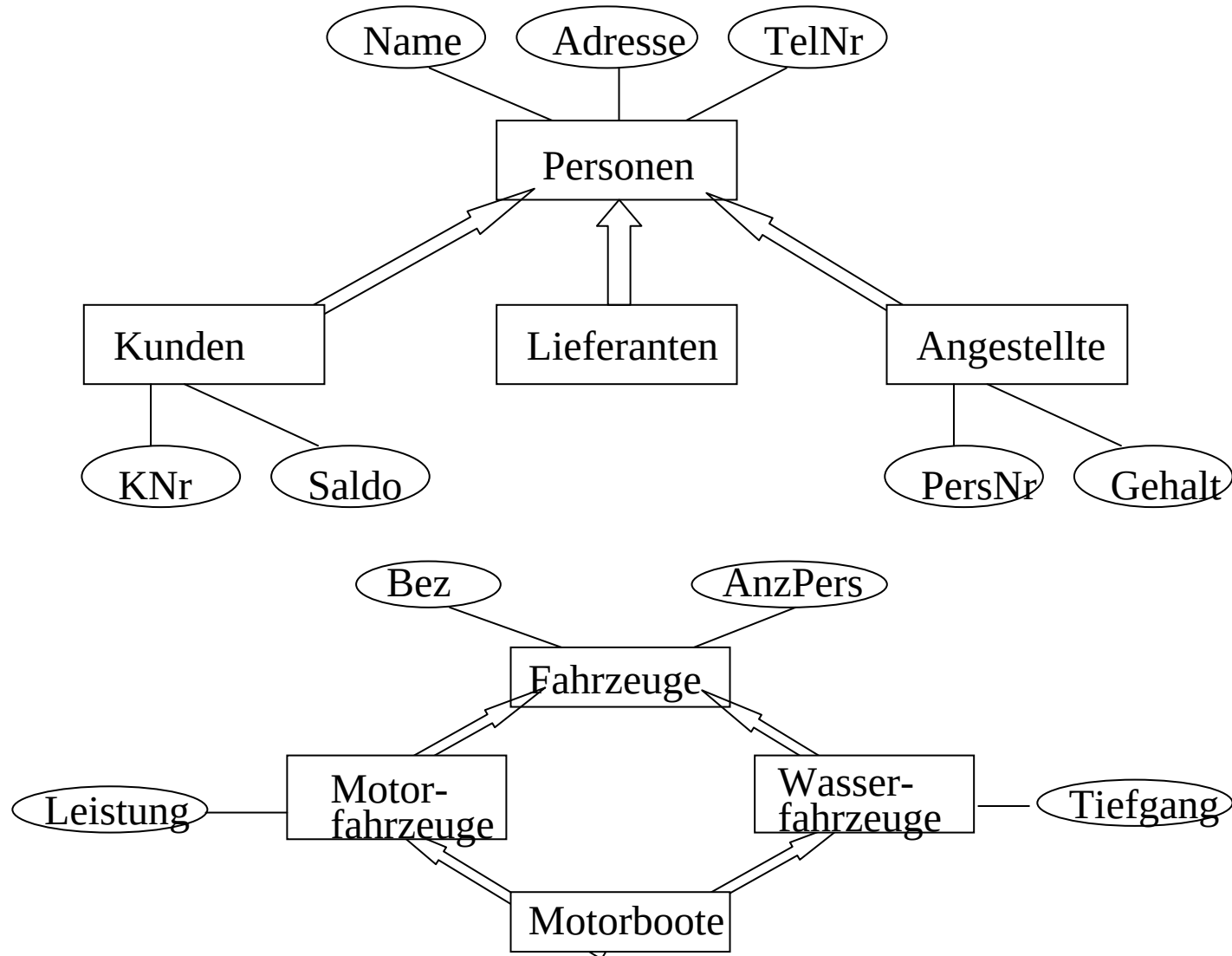
Aggregation

*Zusammengesetzte Entities oder Relationships
(→ ERM-Erweiterungen wie z.B. SERM,
siehe z.B. Aris Toolset)*



Generalisierung (ISA-Relationships)

*1:1-Beziehung zwischen E (Superklasse) und F (Subklasse)
mit $F \subseteq E$ und Vererbung der E-Attribute an F*



12.2 Entwurfsmethodologie (grob)

- 1) Analyse des Informationsbedarfs
- 2) Festlegung von Entity-Mengen mit ihren Attributen
- 3) Festlegung von Primärschlüsseln
- 4) Bildung von Generalisierungsbeziehungen
- 5) Festlegung von Relationship-Mengen
- 6) Spezifikation von Kardinalitätsbedingungen
und ggf. weiteren Integritätsbedingungen

Beispiel Flugbuchungssystem

Es sollen die Informationszusammenhänge für ein Flugbuchungssystem einer Fluggesellschaft modelliert werden.

Flüge werden durch eine Flugnummer identifiziert, die für Flüge am selben Tag eindeutig ist.

Passagiere können einen Flug reservieren, was durch eine Reservationsnummer bestätigt wird.

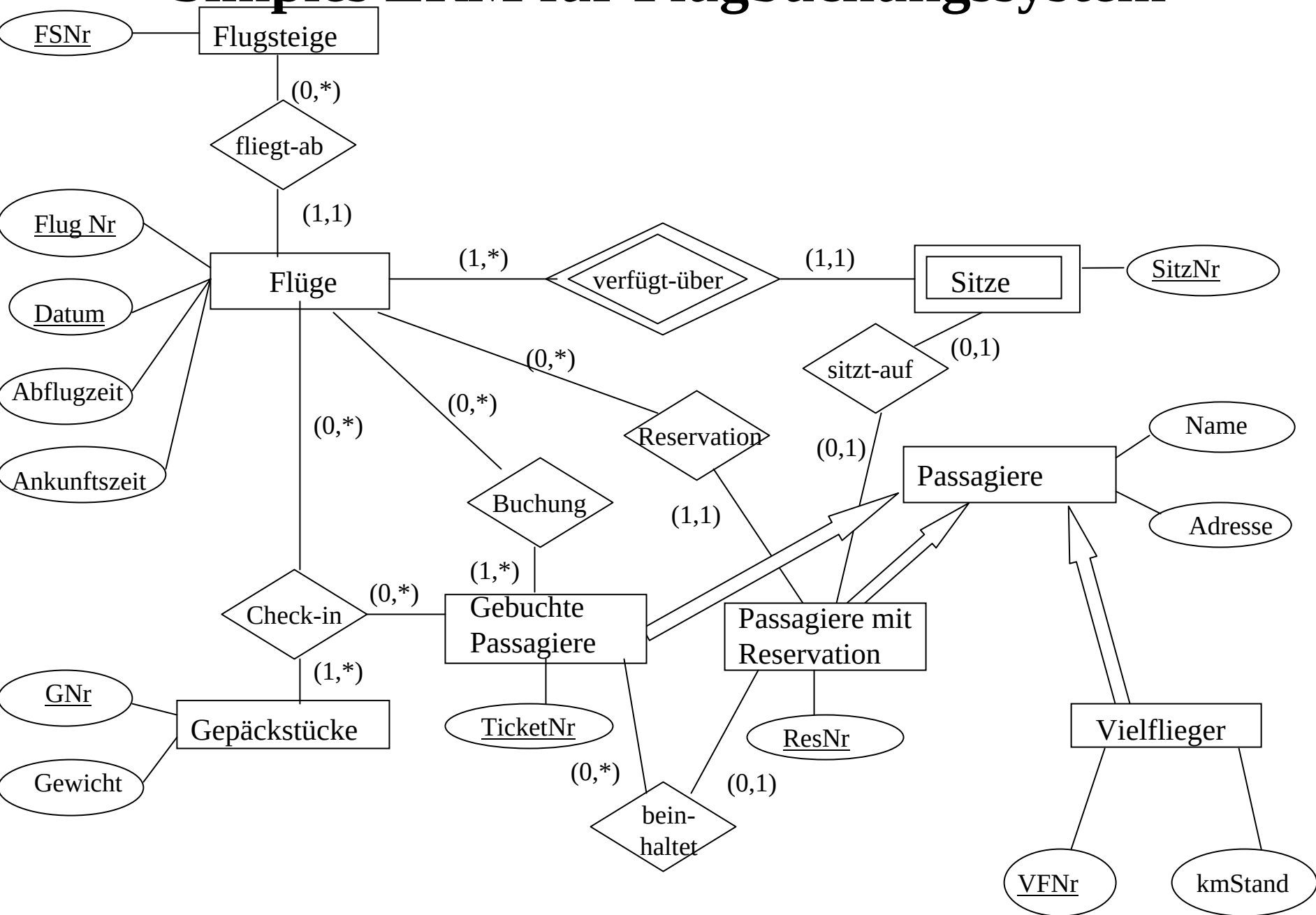
Eine Reservation wird zu einer festen Buchung, indem man ein Ticket kauft. Bei der Reservation oder später können Passagiere auch eine Sitzplatzreservierung vornehmen.

Für Teilnehmer des Vielfliegerprogramms ist die gesamte mit der Fluggesellschaft geflogene Kilometerzahl von Bedeutung.

Flüge fliegen von einem bestimmten Flugsteig ab.

Passagiere müssen vor dem Abflug eine Check-in-Prozedur durchlaufen. Dabei können sie auch Gepäckstücke aufgeben.

Simple ERM für Flugbuchungssystem

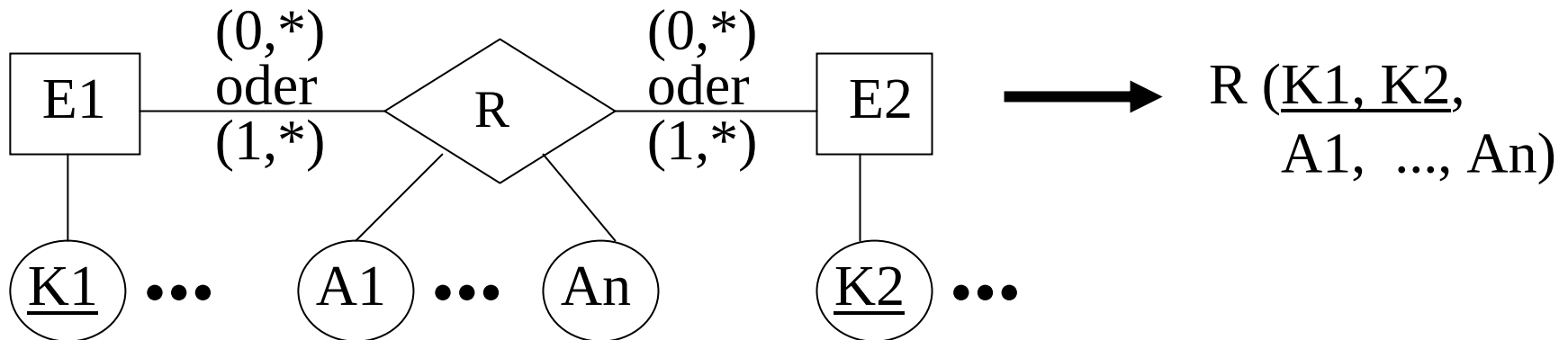


12.3 Abbildung ERM auf Relationenschema (1)

Regel 1: Jede Entity-Menge bildet eine Relation

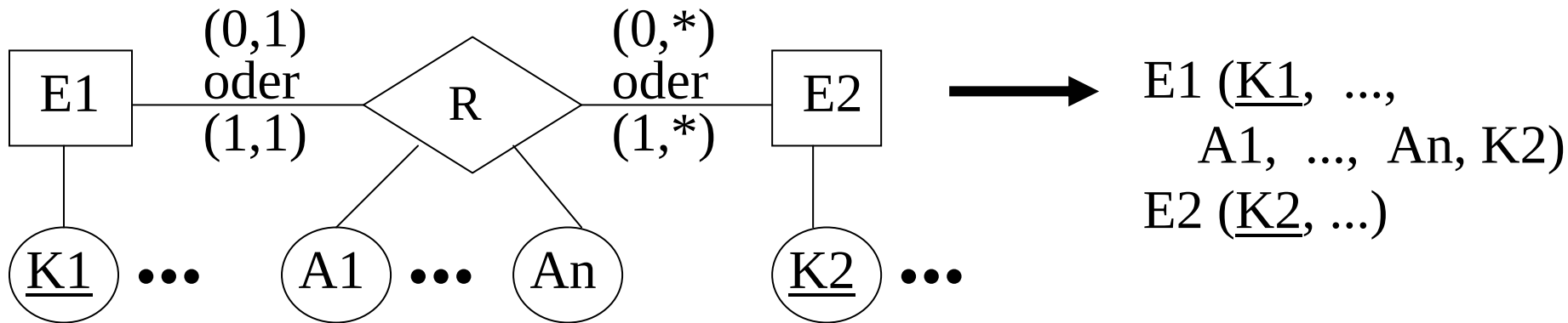


Regel 2: M:N-Relationships bilden eigene Relationen

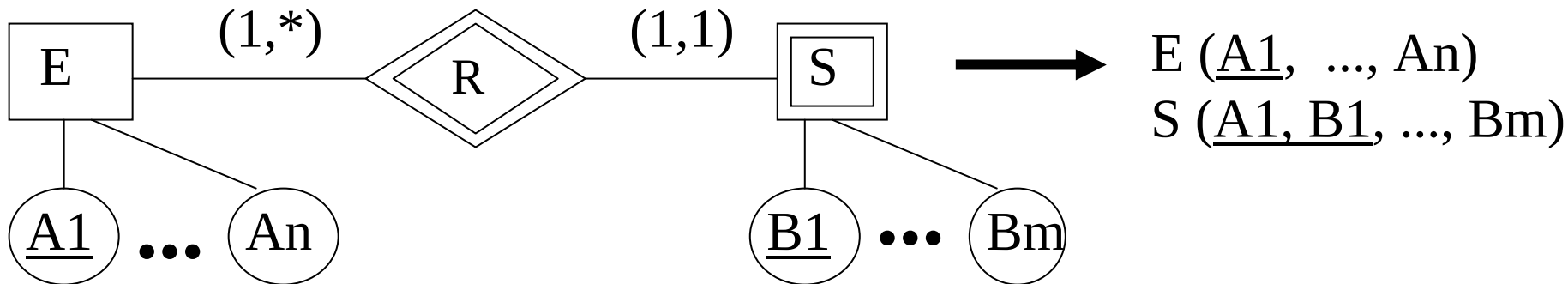


12.3 Abbildung ERM auf Relationenschema (2)

Regel 3: N:1-, 1:N- und 1:1-Relationships können in eine Entity-Relation integriert werden

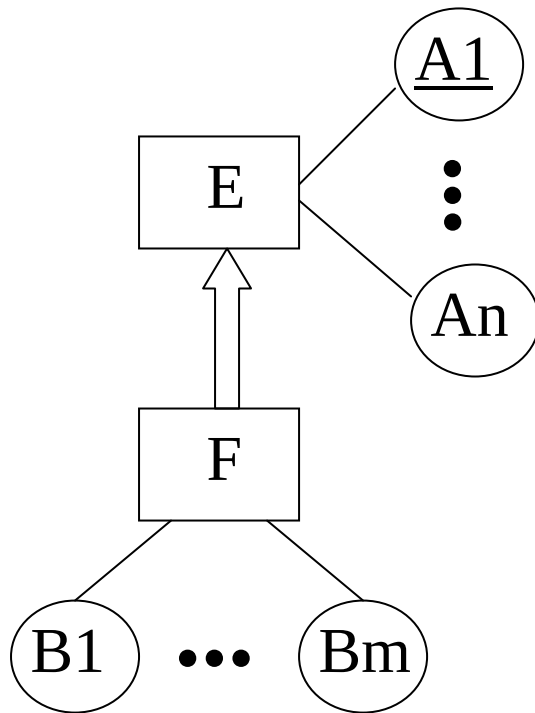


Regel 4: Eine schwache Entity-Menge bildet eine eigene Relation



12.3 Abbildung ERM auf Relationenschema (3)

Regel 5: Generalisierung und Spezialisierung bilden jeweils eigene Relationen



→ $E(\underline{A1}, \dots, An)$
 $F(\underline{A1}, B1, \dots, Bm)$

→ $E(\underline{A1}, \dots, An)$
 $F(\underline{A1}, \dots, An, B1, \dots, Bm)$
wobei E nur Tupel
enthält für Entities
aus E - F

Abbildung für Flugbuchungs-Beispiel

Flüge (FlugNr, Datum, Abflugzeit, Ankunftszeit, FSNr)

Flugsteige (FSNr)

Sitze (FlugNr, Datum, SitzNr)

PassagiereMitReservation (ResNr, Name, Adresse, FlugNr, Datum,
SitzNr, TicketNr)

GebuchtePassagiere (TicketNr, Name, Adresse)

Buchungen (TicketNr, FlugNr, Datum)

Vielflieger (VFNr, Name, Adresse, kmStand)

Gepäckstücke (GNr, Gewicht)

Check-In (FlugNr, Datum, TicketNr, GNr)

12.4 Datenmodellierung mit UML

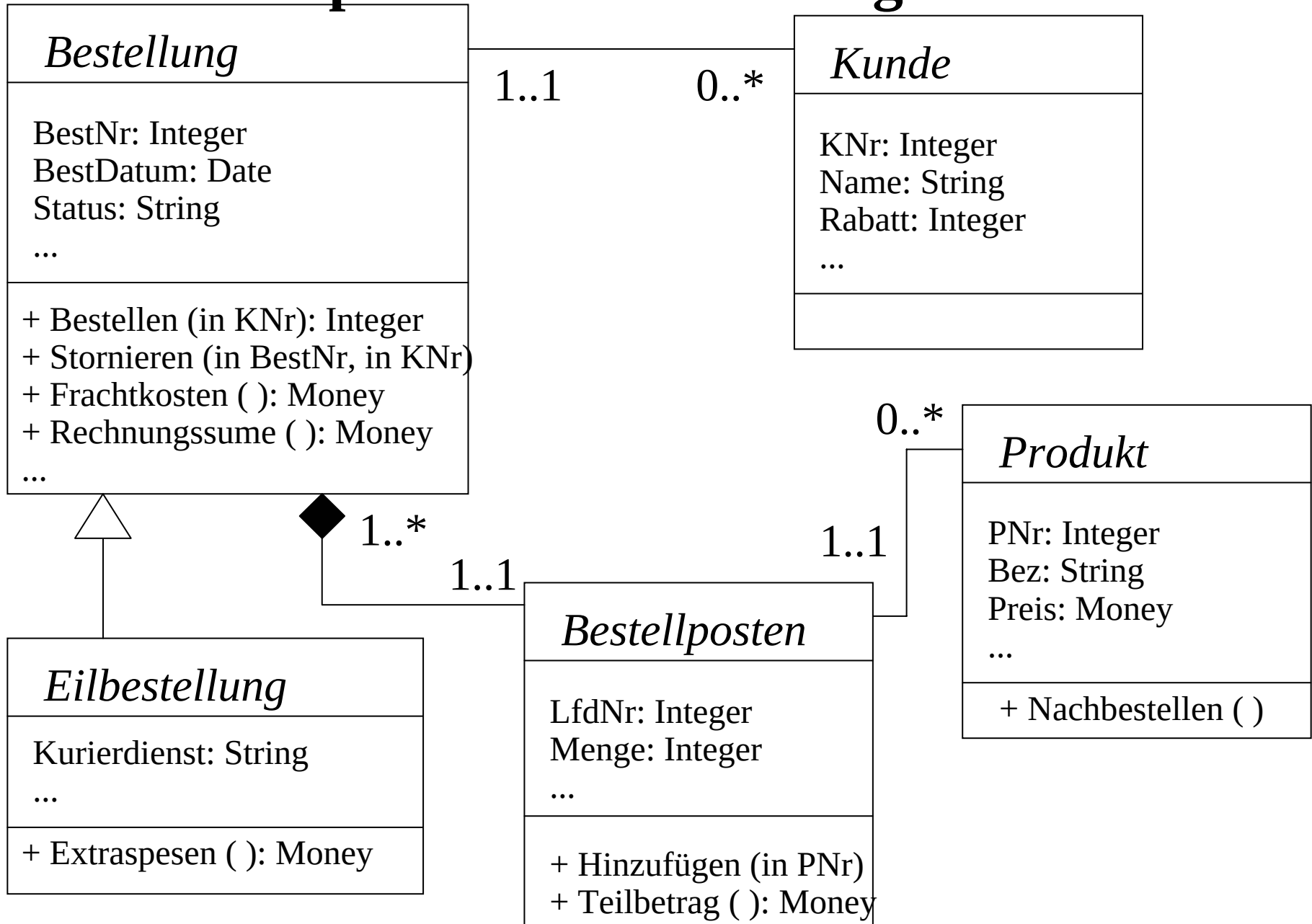
UML (Unified Modeling Language) ist der Industriestandard zur objektorientierten Anforderungsanalyse, Modellierung, Spezifikation und Dokumentation von Daten und Programmen

Gegenüber der Datenmodellierung mit ERM kann man mit UML sowohl statische als auch dynamische Aspekte eines IS beschreiben; insbesondere werden unterstützt:

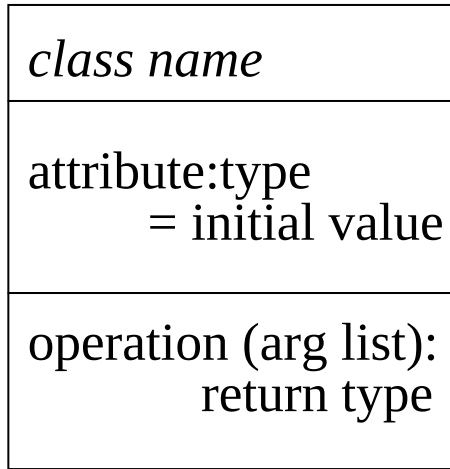
- Objektaggregationen
- Funktionen auf Objekten (Methoden)
- Interaktionen zwischen Objekten und ganze (Geschäfts-) Prozesse (→ Kapitel 11)

UML zielt auf das ganze Spektrum von der informellen Kommunikation mit Anwendern bis hin zur Programmdokumentation. Dazu wird eine Vielfalt an Diagrammtypen angeboten, deren formale Semantik jedoch nicht immer klar ist.

Beispiel eines Klassendiagramms



Klassen und Objekte



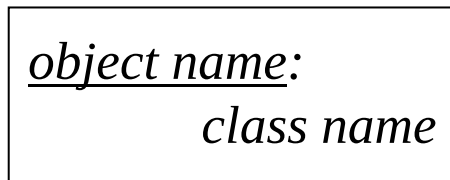
constraints

Klasse (Objekttyp, Komponente)

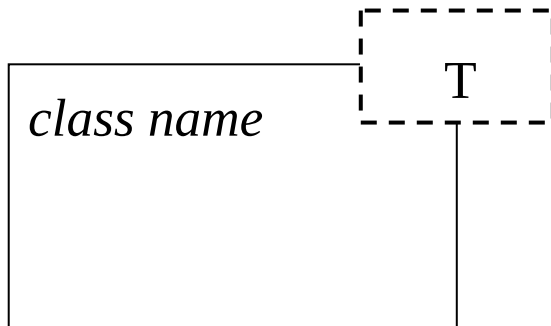
Attribute

Methoden mit in-, out- und inout-Parametern
+: public, -: private, #: protected

Konsistenzbedingungen, Vor- und Nachbedingungen
(constraints) in Textform

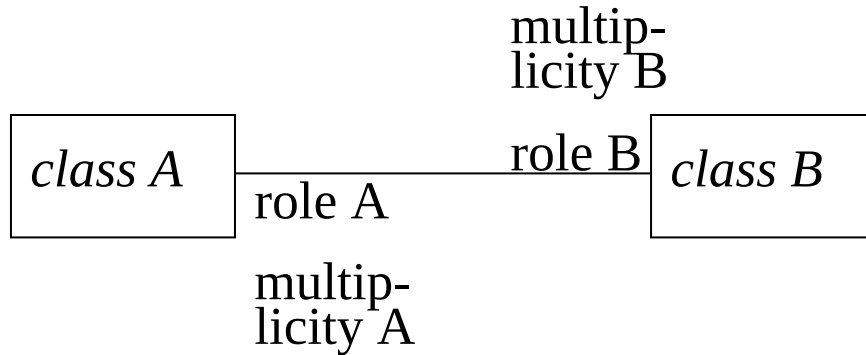


Objekt (Instanz einer Klasse)

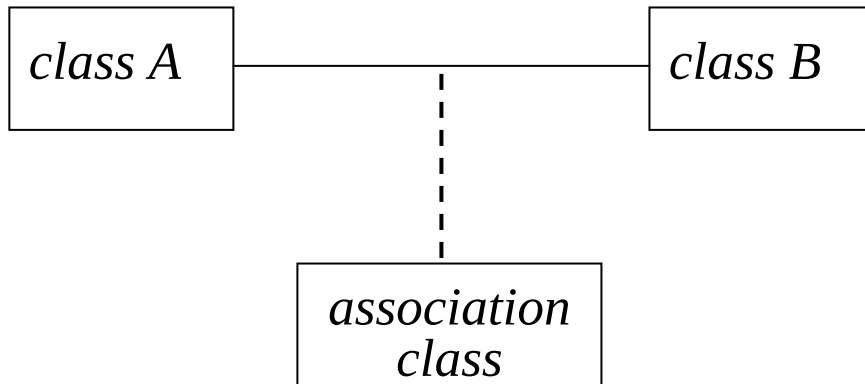


Parametrisierte Klasse (Template Class)
z.B. List <T>

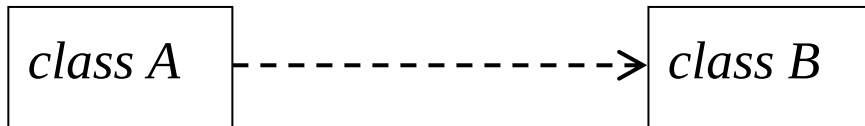
Assoziationen (Relationships)



Assoziation / Relationship (associations) zwischen Klassen mit Rollennamen und Kardinalitäten (min..max) und ggf. der Option ordered

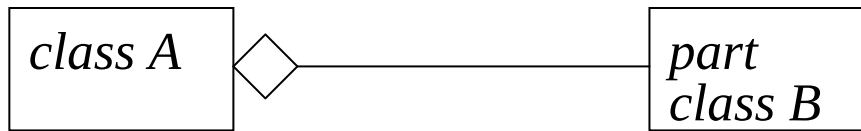


Beziehungsklasse (association class): entspricht den ERM-Relationships mit eigenen Attributen

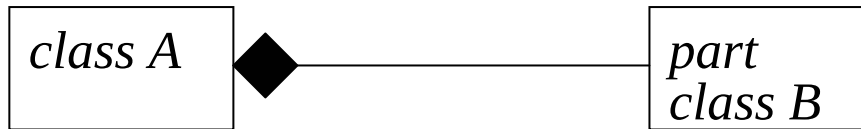


Abhängigkeit (dependency) der Klasse A von Klasse B (z.B. Server-API-Klasse B und Client-Klasse A)

Aggregationen und Kompositionen

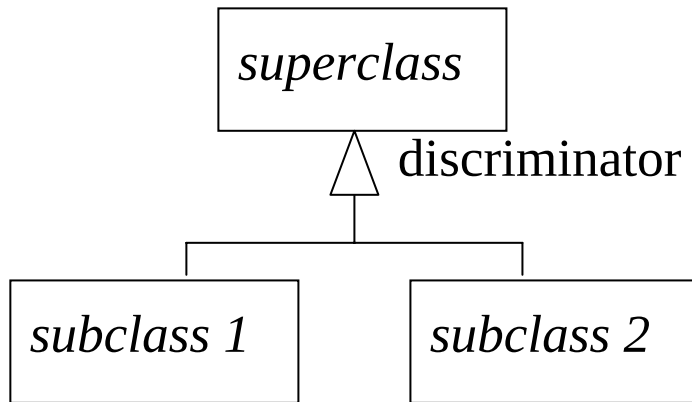


Aggregation (aggregation)
mit shared objects
(Sonderfall der Assoziation)

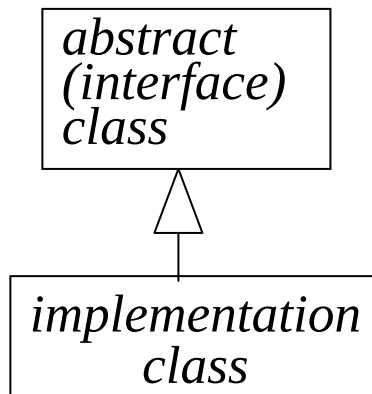


Komposition (composition)
mit non-shared objects
(Sonderfall der Assoziation)

Generalisierungen



Generalisierungshierarchie
(generalization)

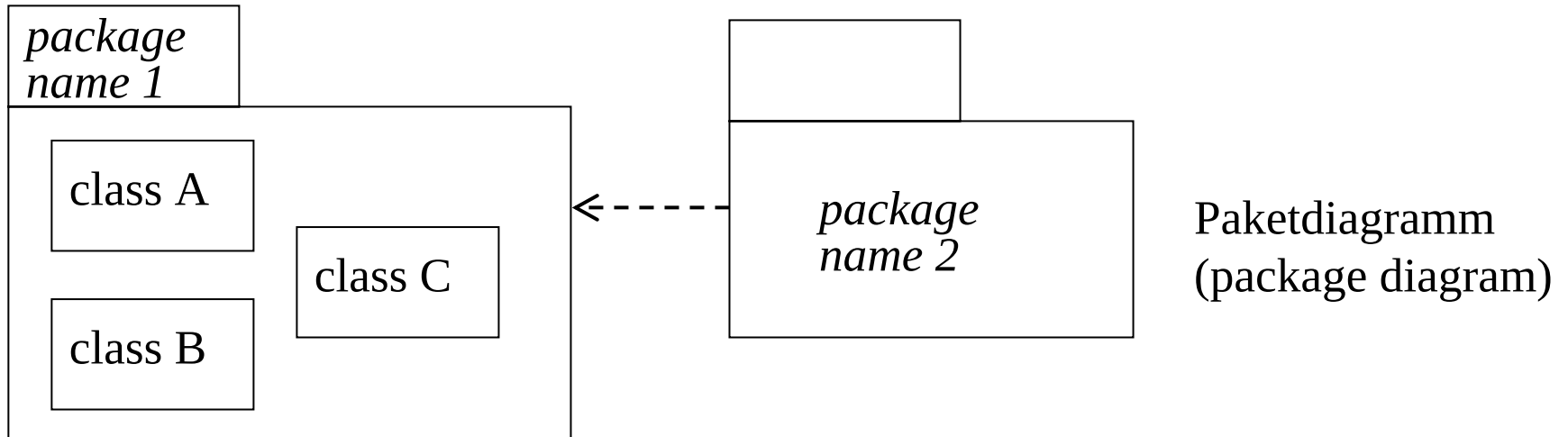


Beziehung zwischen Schnittstelle (ADT
und Implementierung
(Sonderfall der Generalisierung /
Spezialisierung)

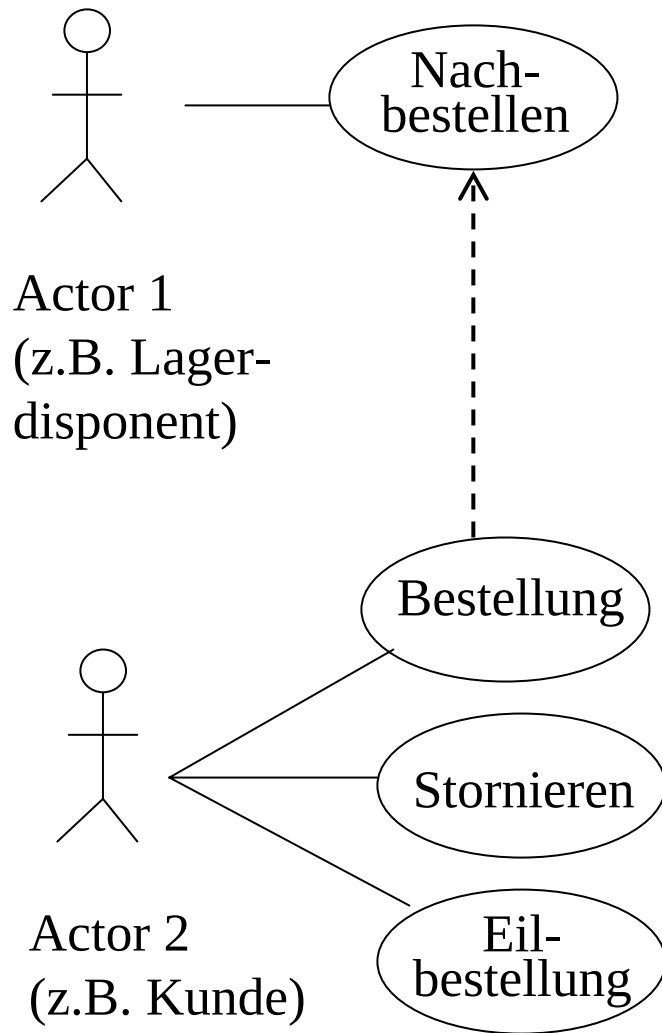
Weitere Diagrammtypen in UML

- ***Paket-Diagramme (Package Diagrams)*** als vergrößerte Sicht auf große Klassendiagramme,
- ***Anwendungsfall-Diagramme (Use-Case Diagrams)*** zur groben
 - Darstellung der Beziehung zwischen Benutzern und Funktionen,
- ***Interaktionsdiagramme (Sequence Diagrams)*** zur Darstellung der
 - Methodenaufrufe zwischen Objektklassen,
- ***Zustandsübergangsdigramme (State-Transition Diagrams)*** zur
 - Darstellung des Verhaltens von Objekten in Anwendungsabläufen,
- ***Aktivitätsdiagramme (Activity Diagrams)*** zur Darstellung des
 - Kontroll- und Datenflusses zwischen Objekten.

Paketdiagramme

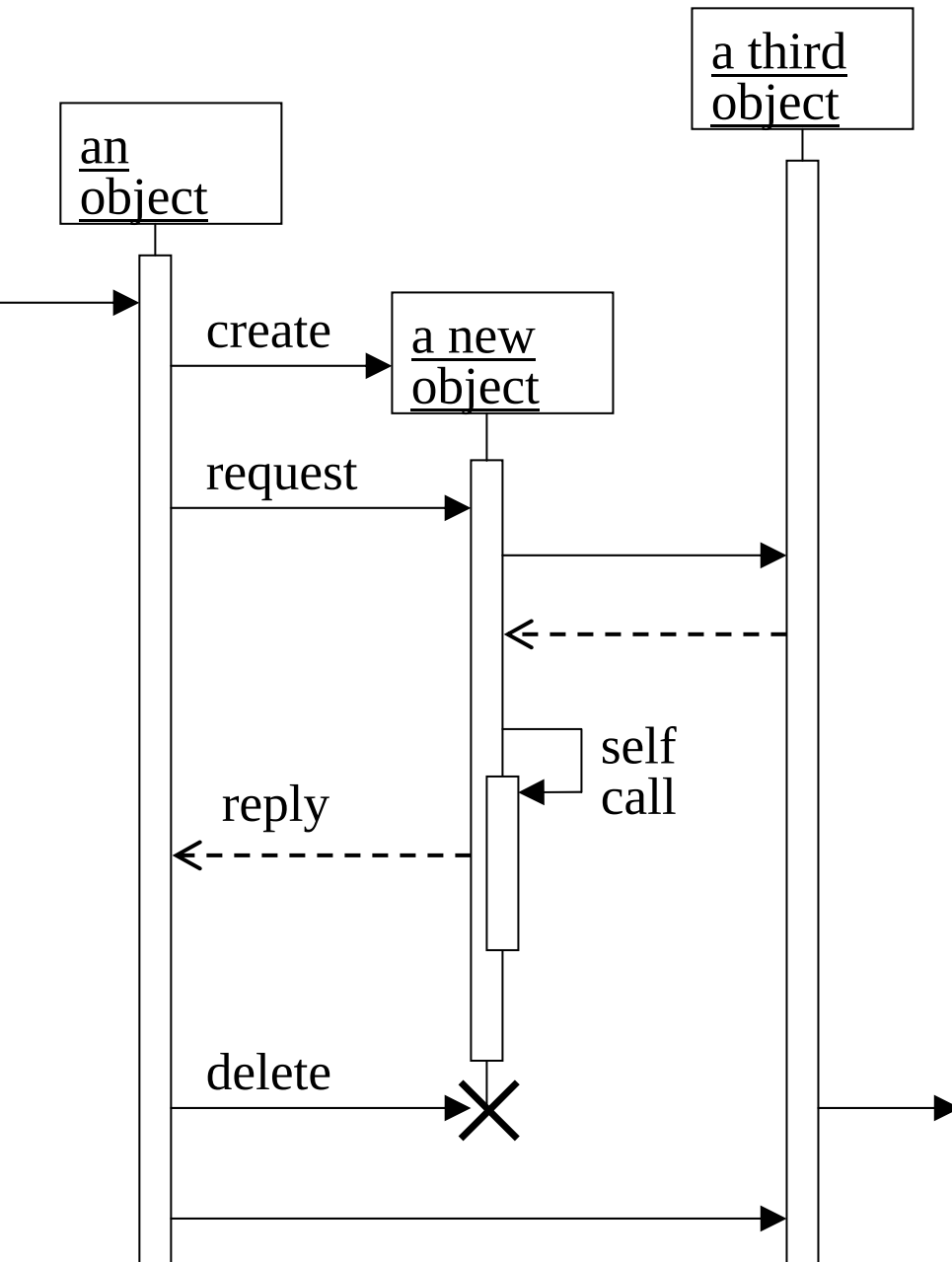


Anwendungsfalldiagramme



Anwendungsfalldiagramm
(use-case diagram):
Zusammenhang zwischen
Benutzern bzw. Rollen
und Funktionen / Ereignissen

Interaktionsdiagramme



Interaktionsdiagramm
(sequence diagram):
Nachrichtenfluss zwischen
Objekten (bzw. Threads)

12.5 Objektorientierte Analyse- und Entwurfsmethodologie

z.B. OMT (Object Modeling Technique):

- 1) strukturelle Modellierung der Datenobjekte
 - 2) Dynamikmodellierung
 - 3) Funktionsmodellierung
- mit iterativen Verfeinerungen

Softwarewerkzeuge (z.B. Rational Rose)
begleiten diesen Entwurfsprozess.

Kapitel 13: Prozessmodellierung und Workflow-Management

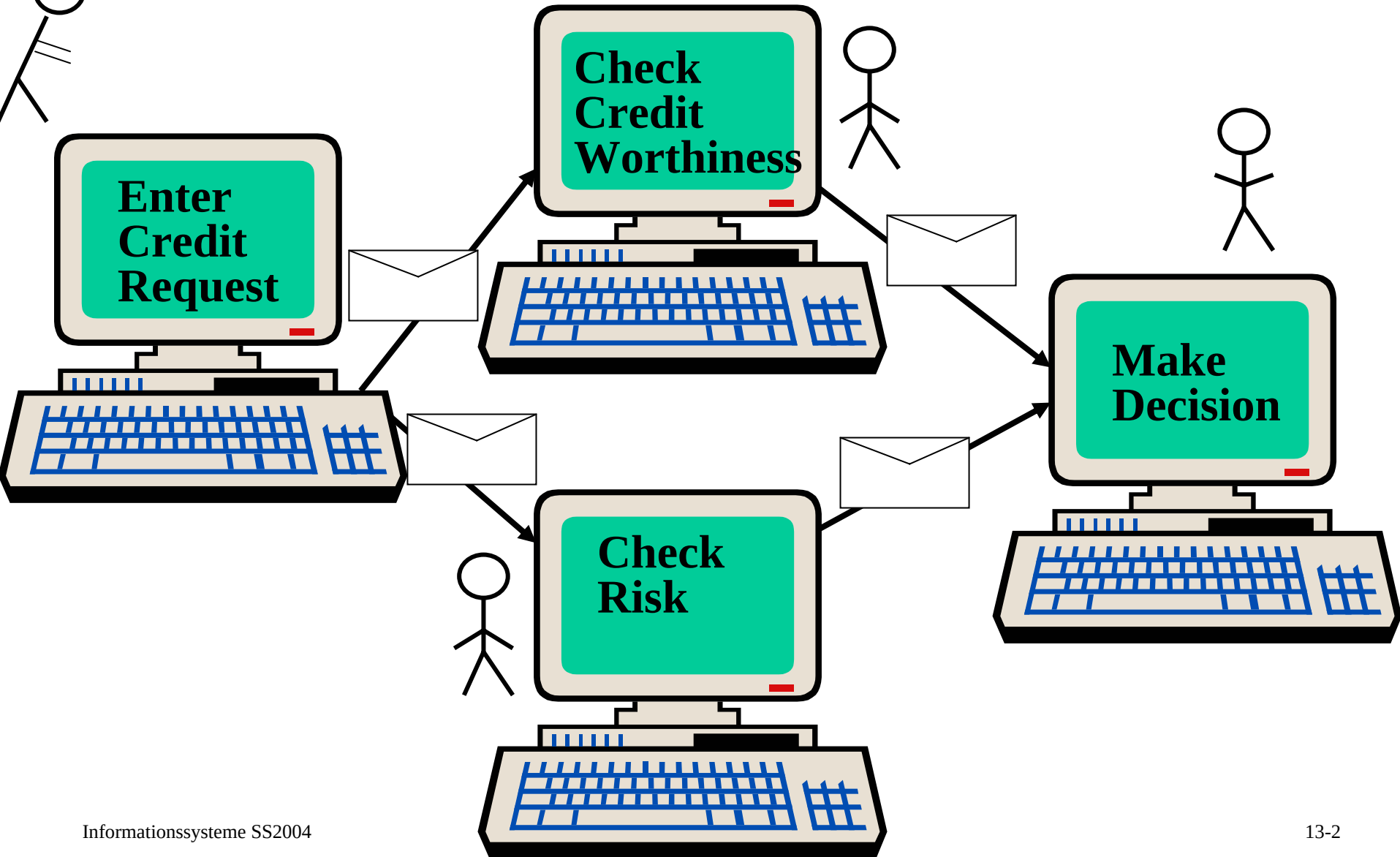
13.1 Prozessmodellierung

- **Statecharts**
- **Ereignis-Prozess-Ketten**

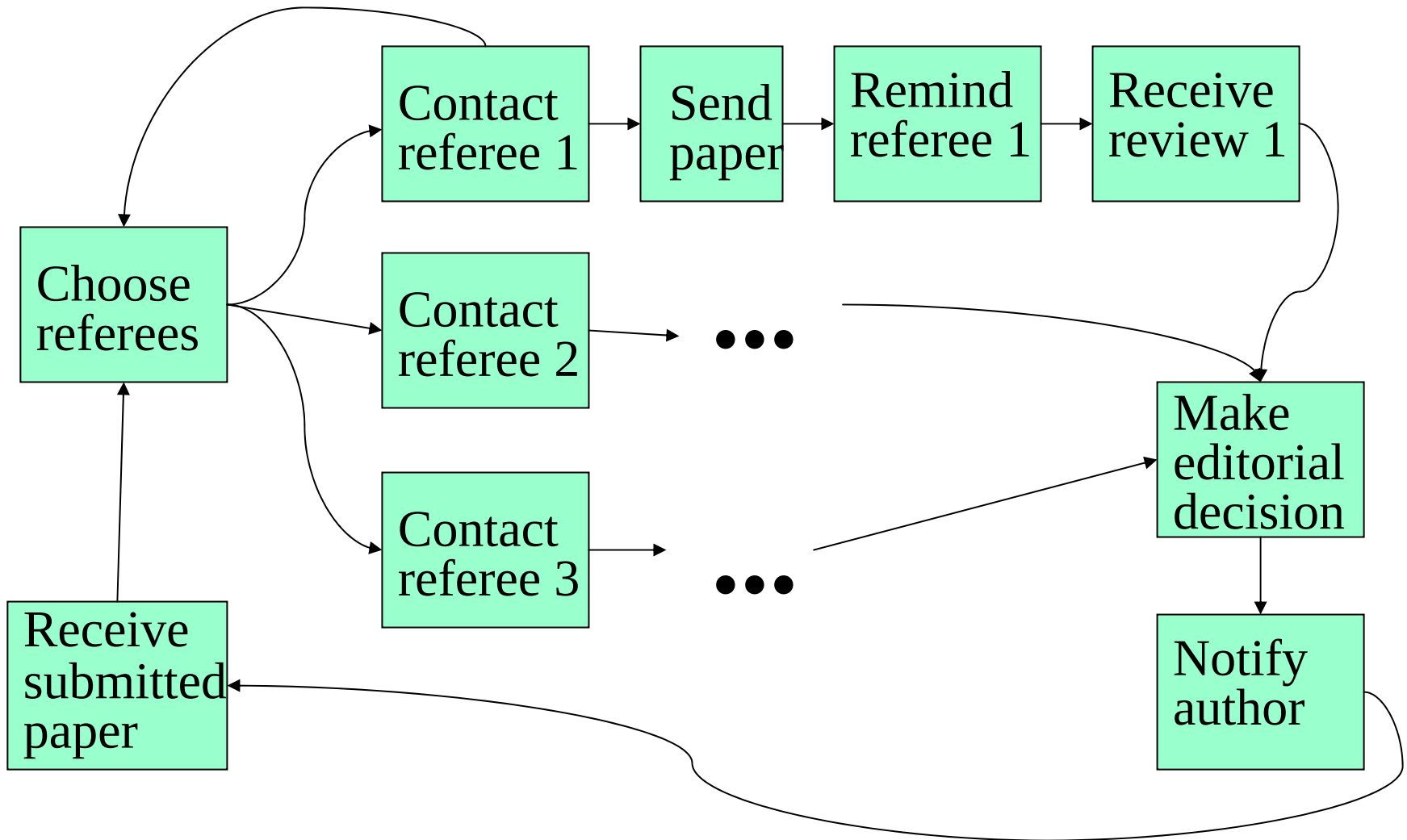
13.2 Workflow-Management für Geschäftsprozesse

13.3 Semantik von Statecharts

13.4 Eigenschaften von Statecharts und deren Verifikation



Workflow Application Example 2: Journal Refereeing Process



What is Workflow Management?

*Computer-supported business processes:
coordination of control and data flow between
distributed - automated or intellectual - activities*

Application examples:

- ★ Credit requests, insurance claims, etc.
- ★ Tax declaration, real estate purchase, etc.
- ★ Student exams, journal refereeing, etc.
- ★ Electronic commerce, virtual enterprises, etc.

Business Benefits of Workflow Technology



Business process automation
(to the extent possible and reasonable)

- ➡ shorter turnaround time, less errors, higher customer satisfaction
- ➡ better use of intellectual resources for exceptional cases



Transparency

- ➡ understanding & analyzing the enterprise



Fast & easy adaptation

- ➡ Business Process Reengineering (BPR)

13.1 Specification Method and Environment

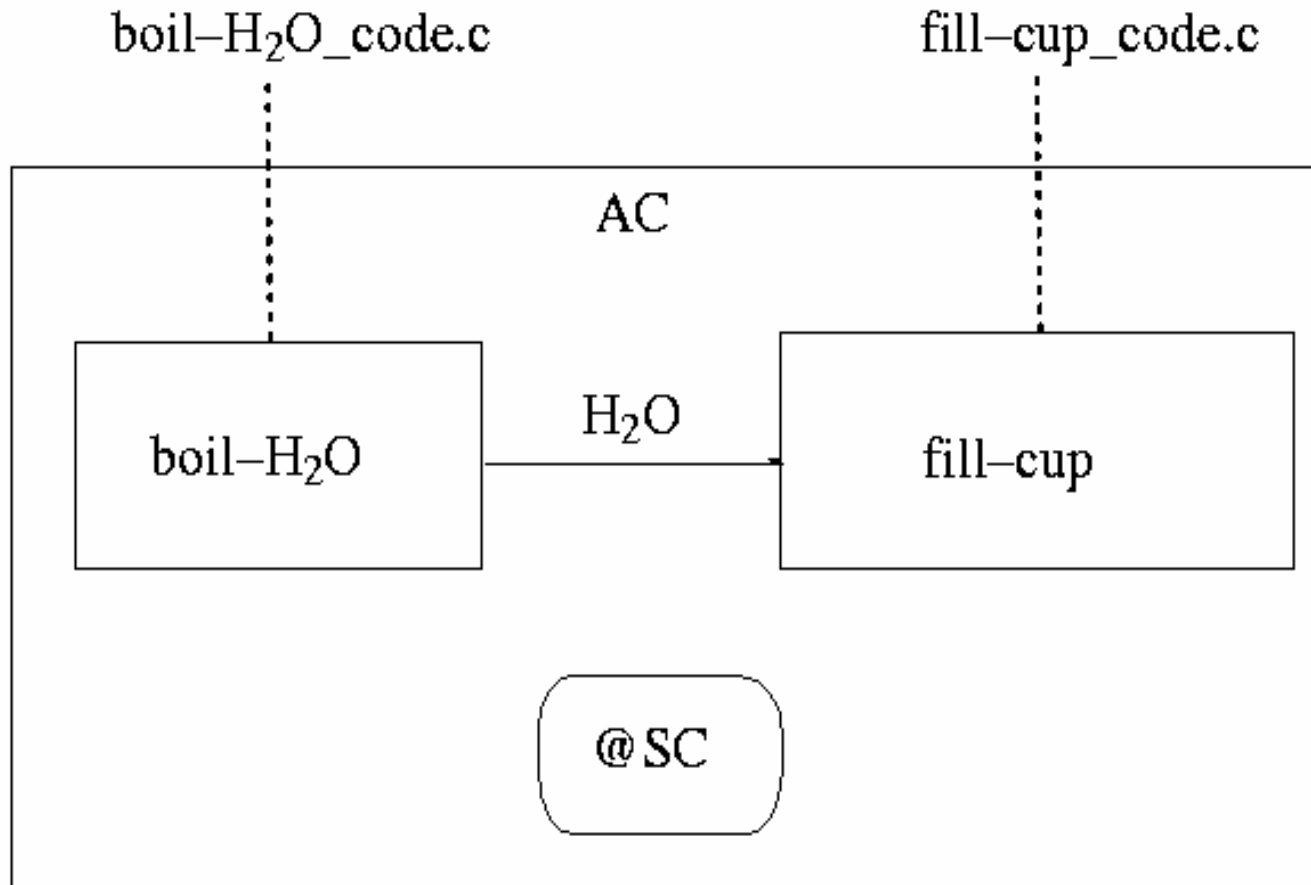
Requirements:

- Visualization
 - Refinement & Composability
 - Rigorous Semantics
- 
- Interoperability with other methods & tools
 - Wide acceptance & standard compliance

Solutions:

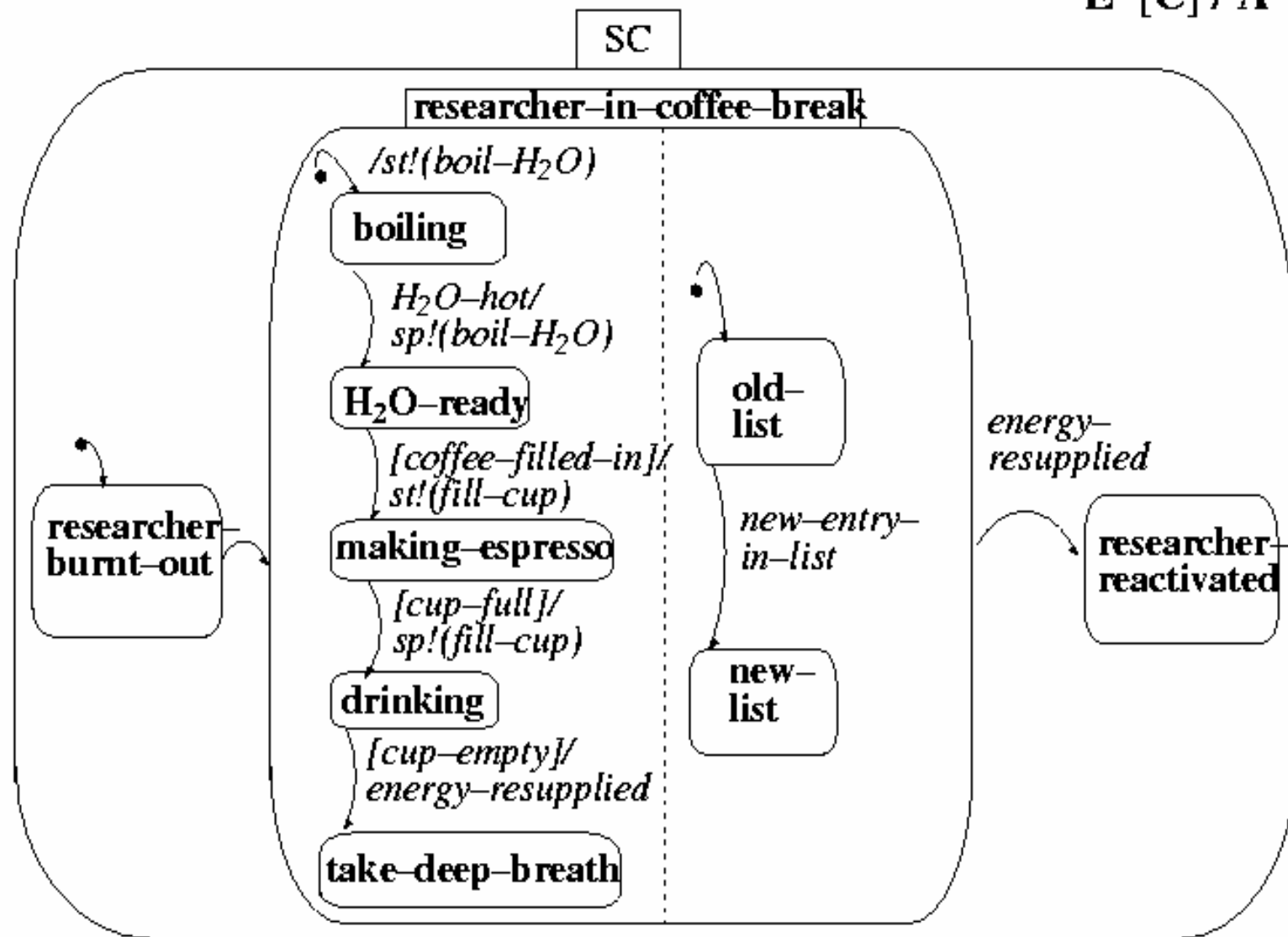
- ➡ **Statecharts** (Harel et al. 1987)
(alt.: Petri Net variants, temporal logic, process algebra, script language)
- ➡ Import / export
BPR tools → WFMS ↔ WFMS
- ➡ Statecharts included in UML industry standard (Unified Modeling Language, OMG 1997))

Example of Activitychart

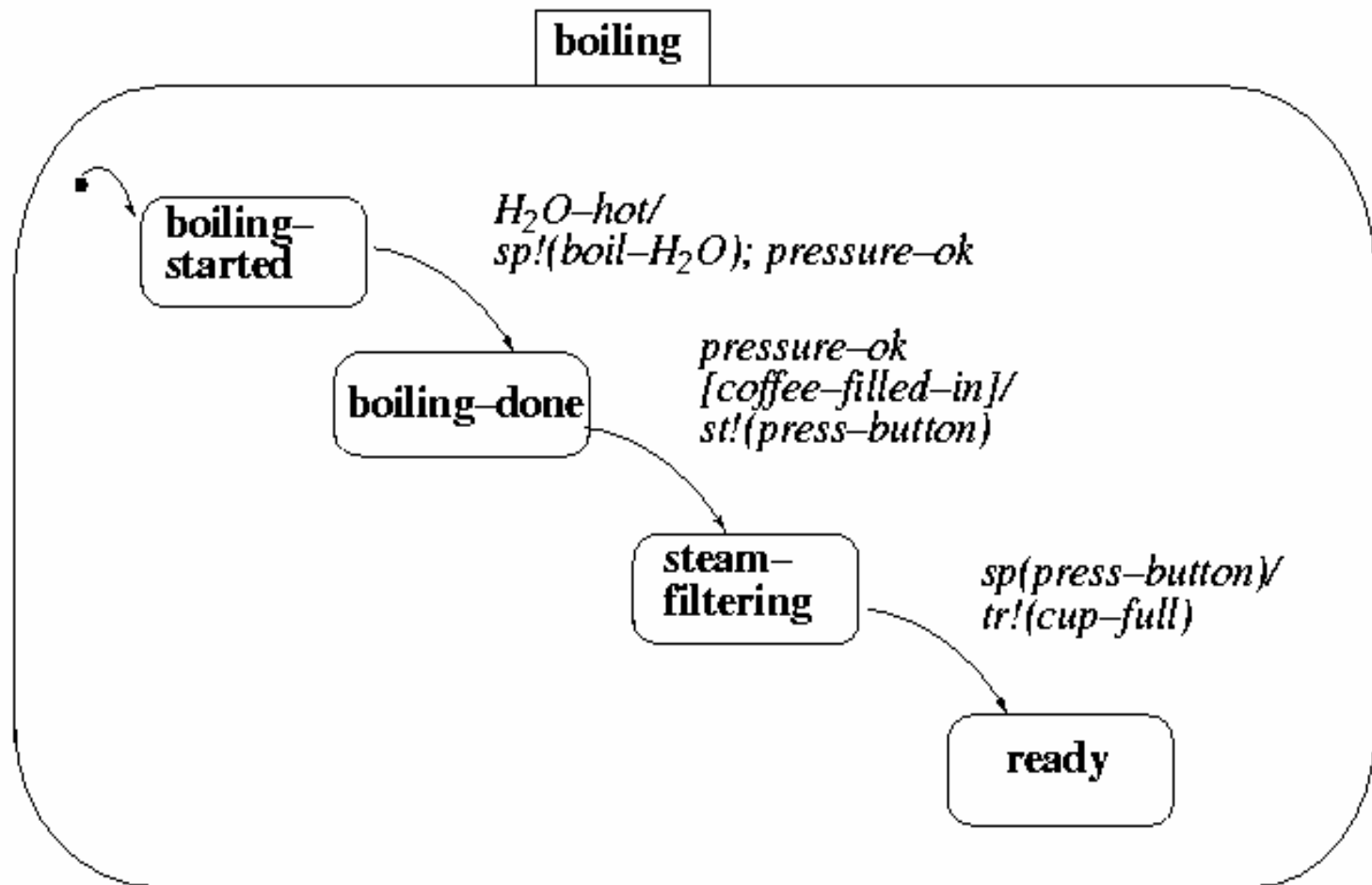


Example of Statechart

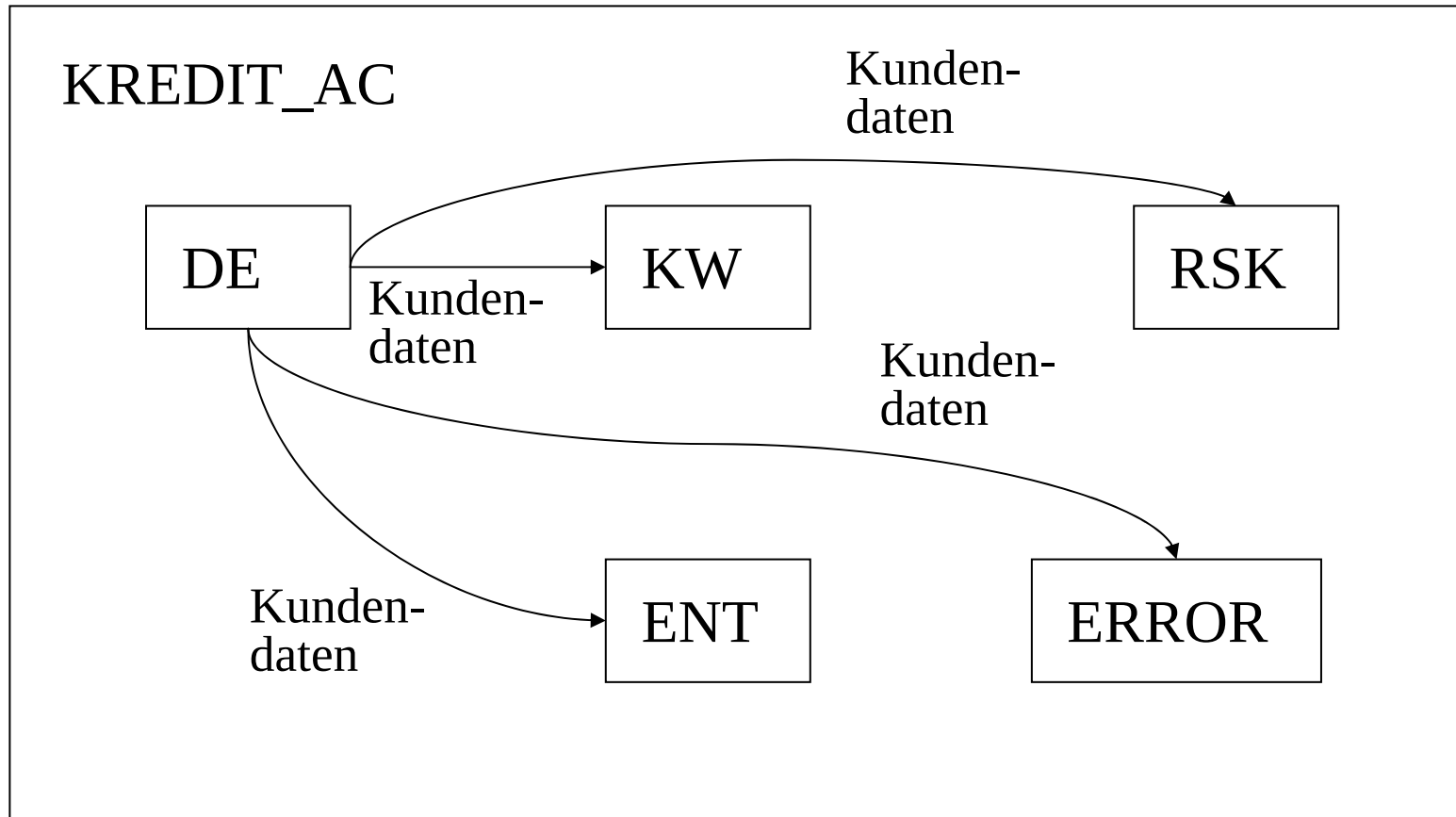
E [C] / A



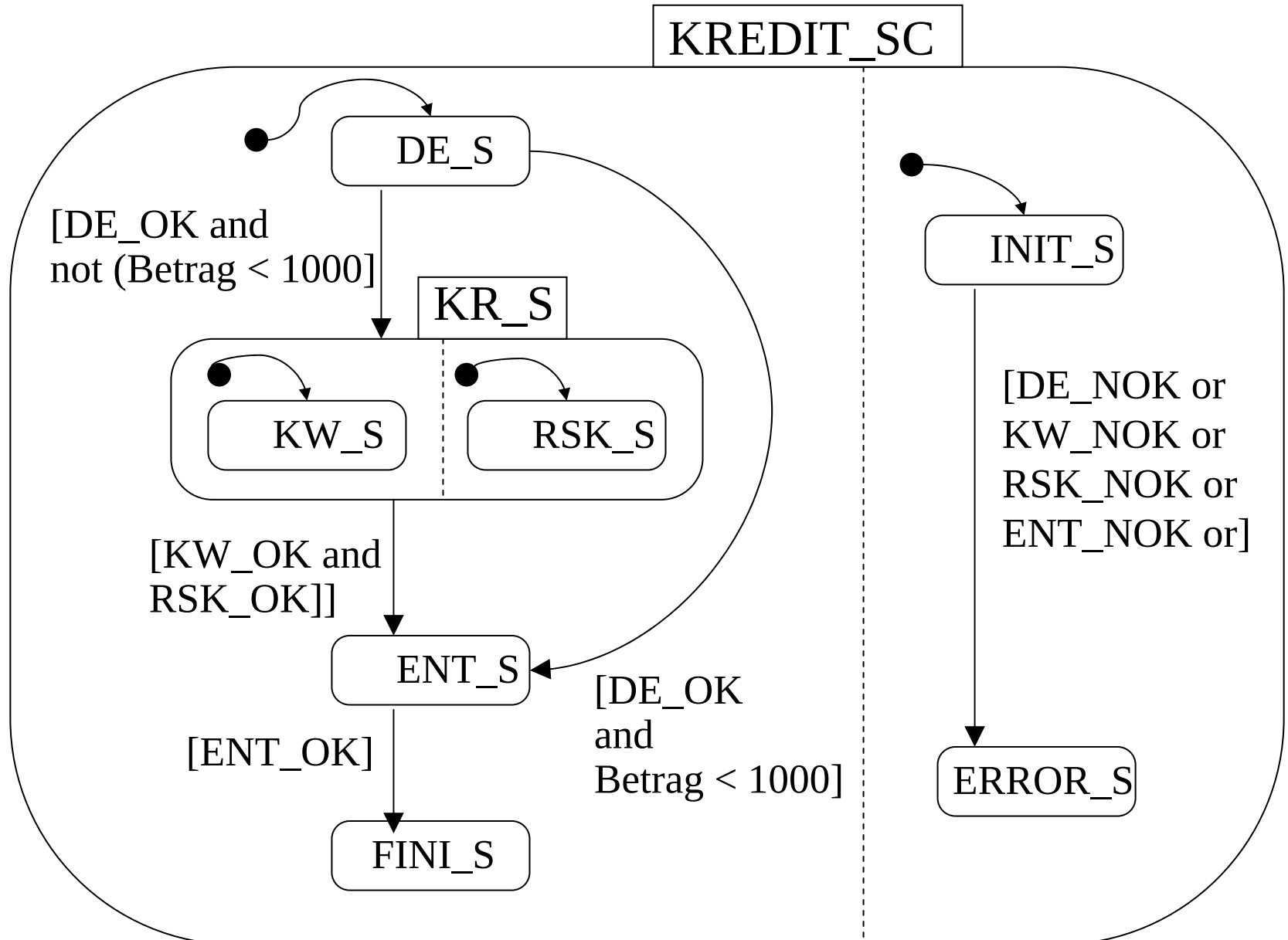
Refinement of States



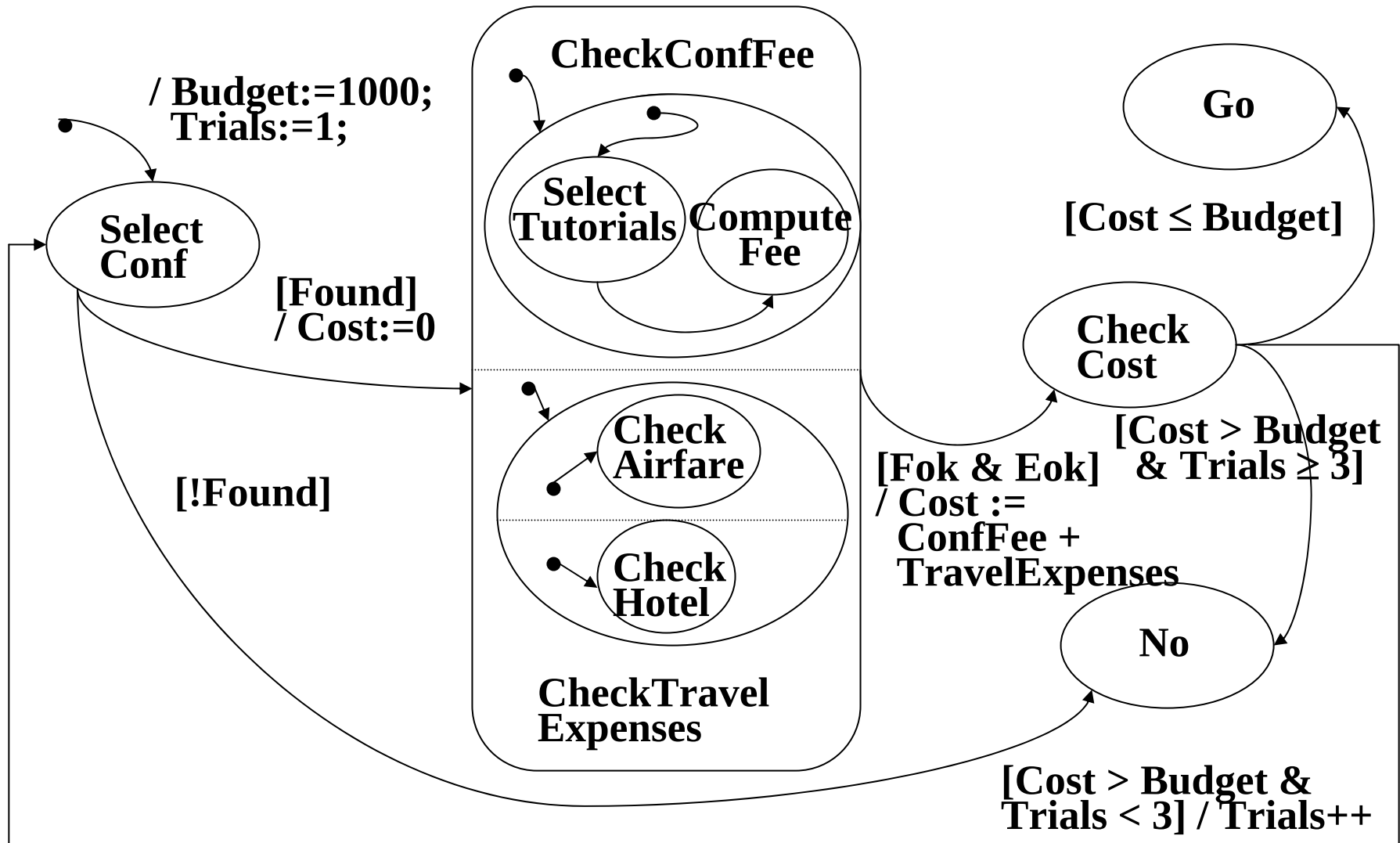
Activitychart Example 1



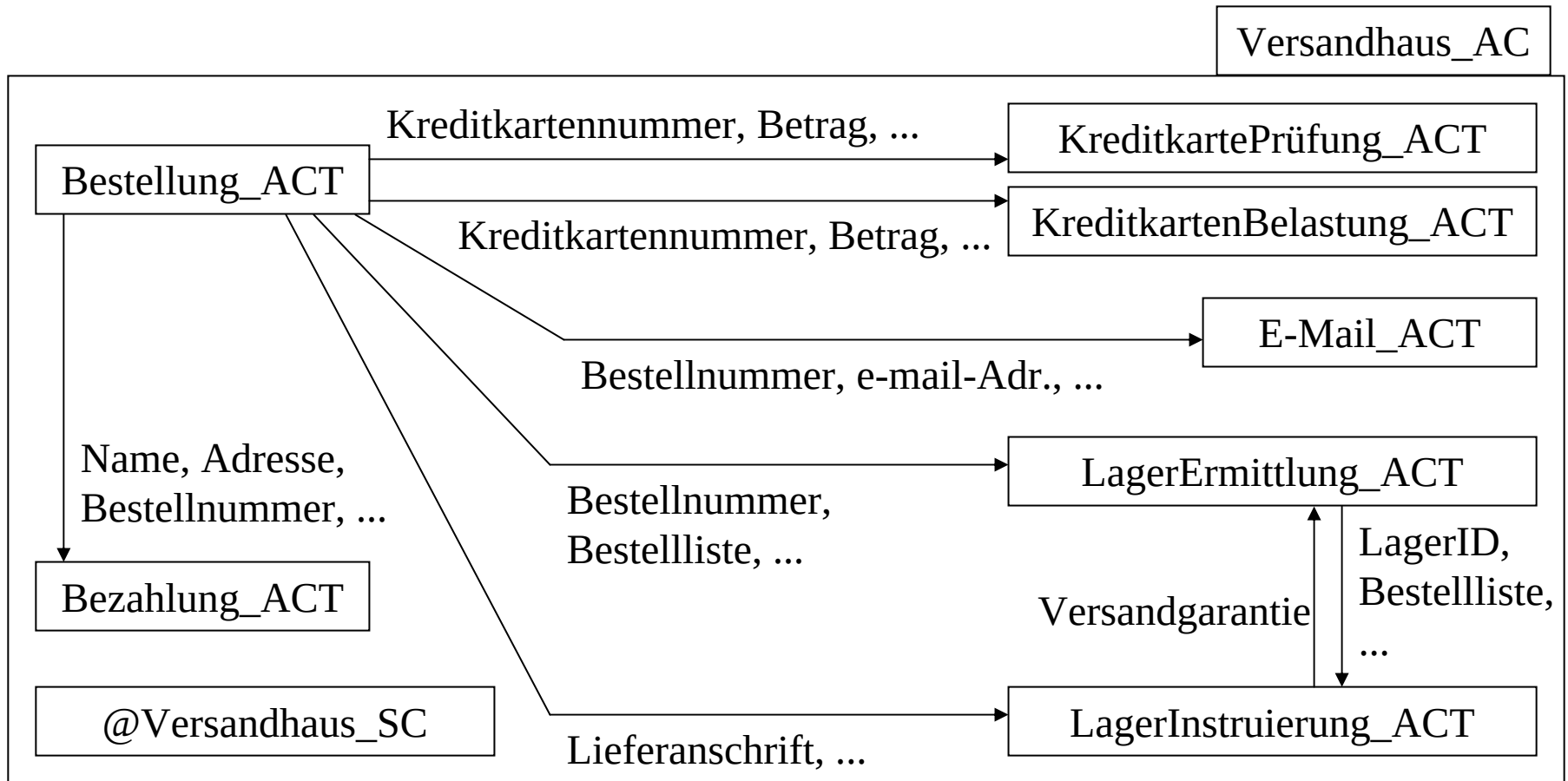
Statechart Example 1



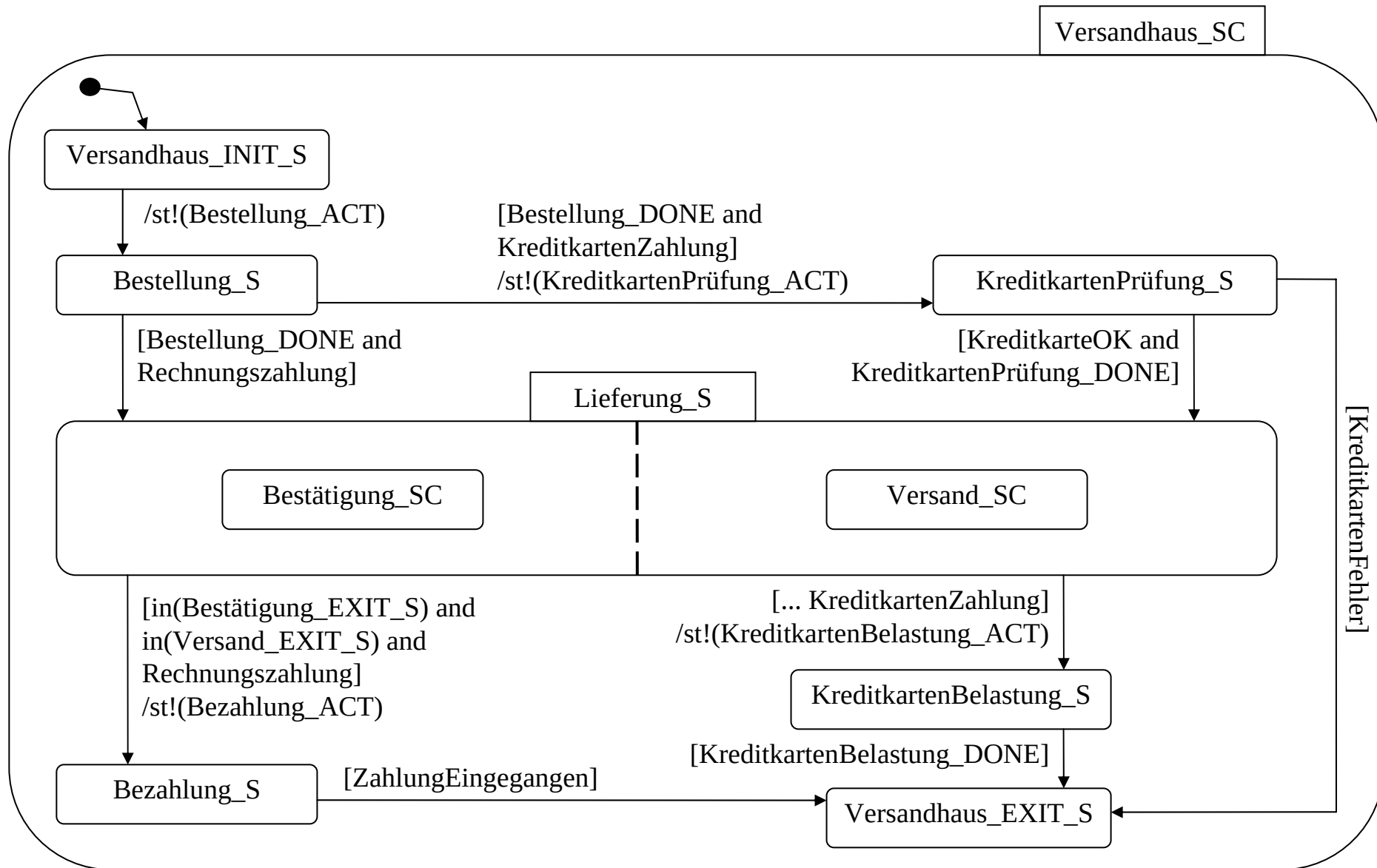
Statechart Example 2



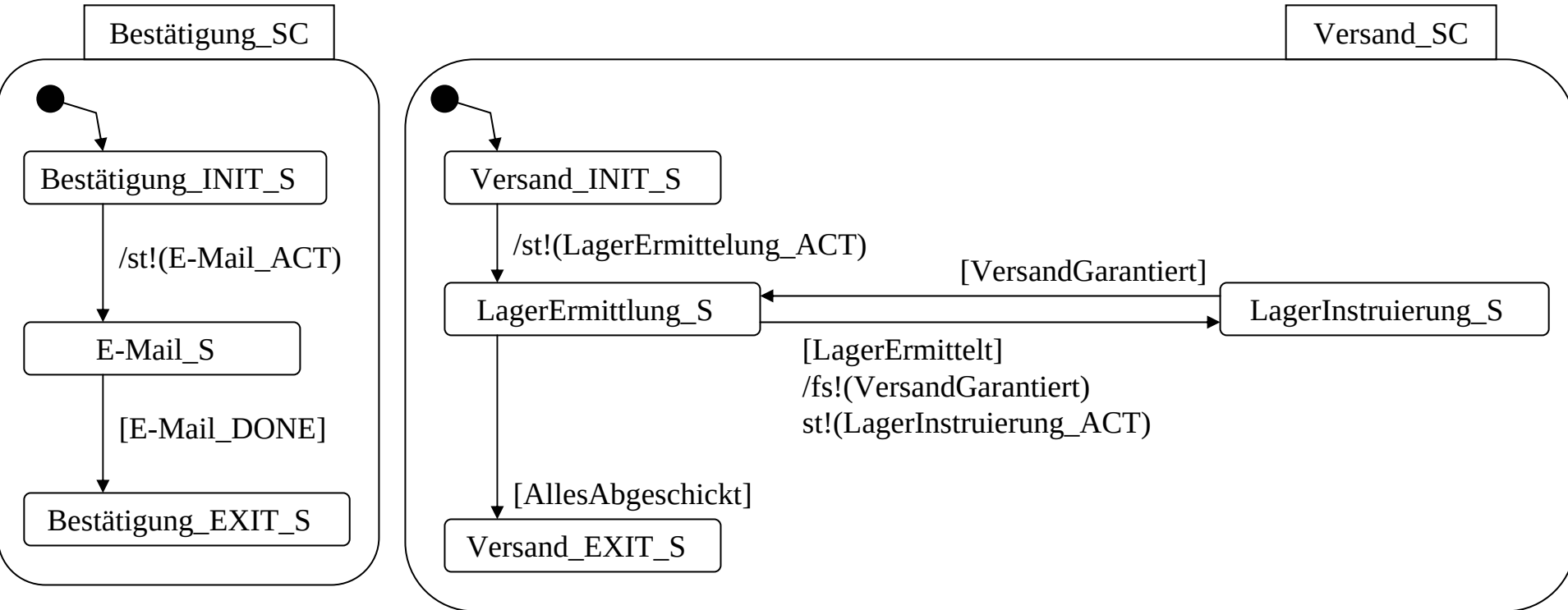
E-Commerce Workflow: Activitychart



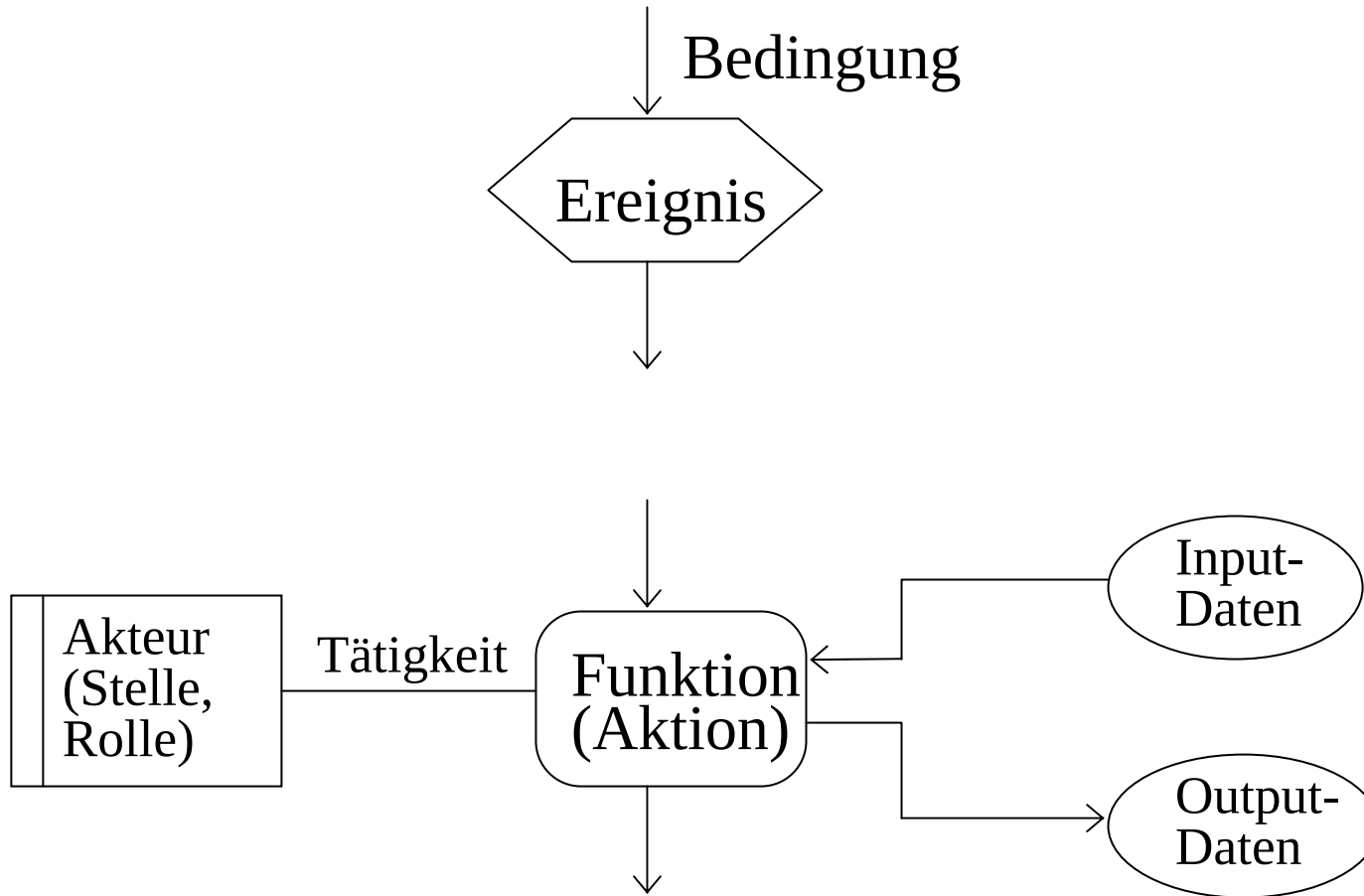
E-Commerce Workflow: Statechart



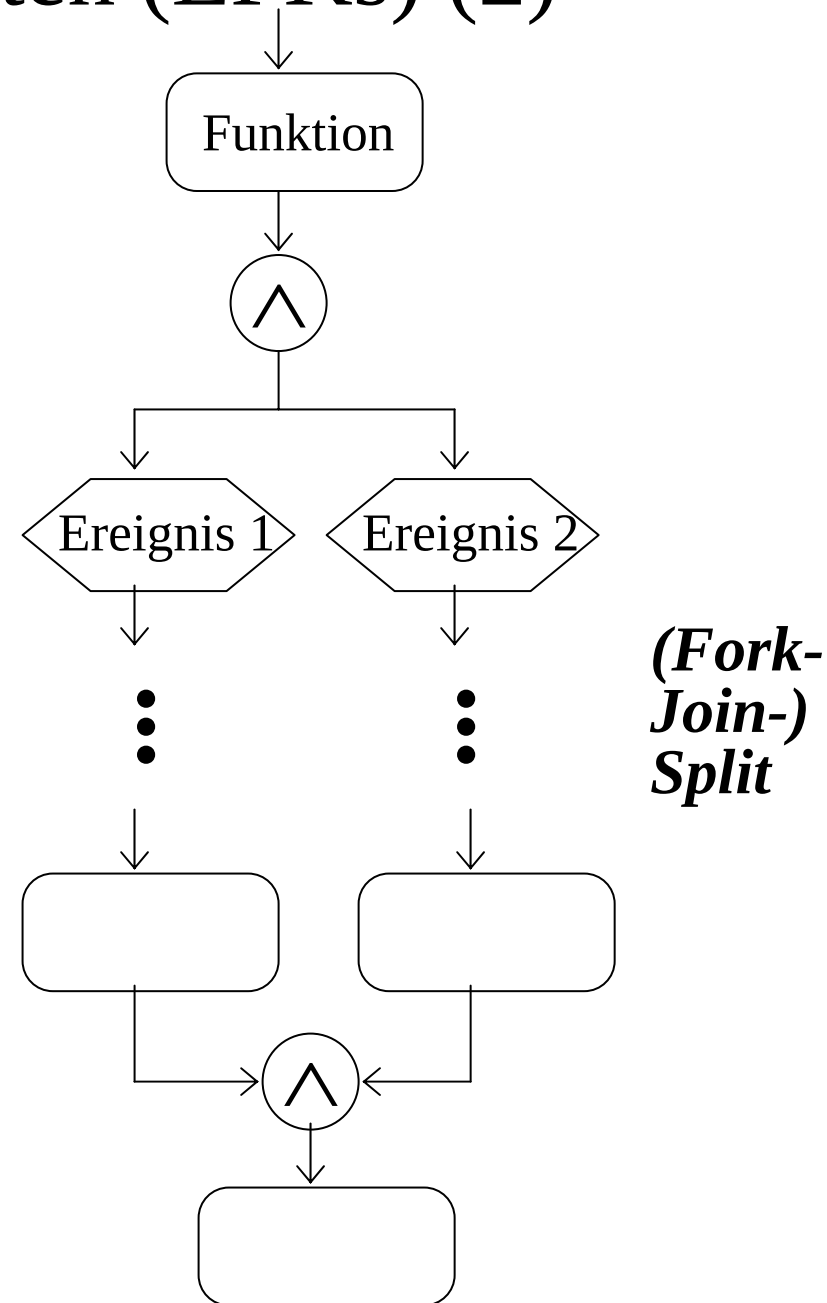
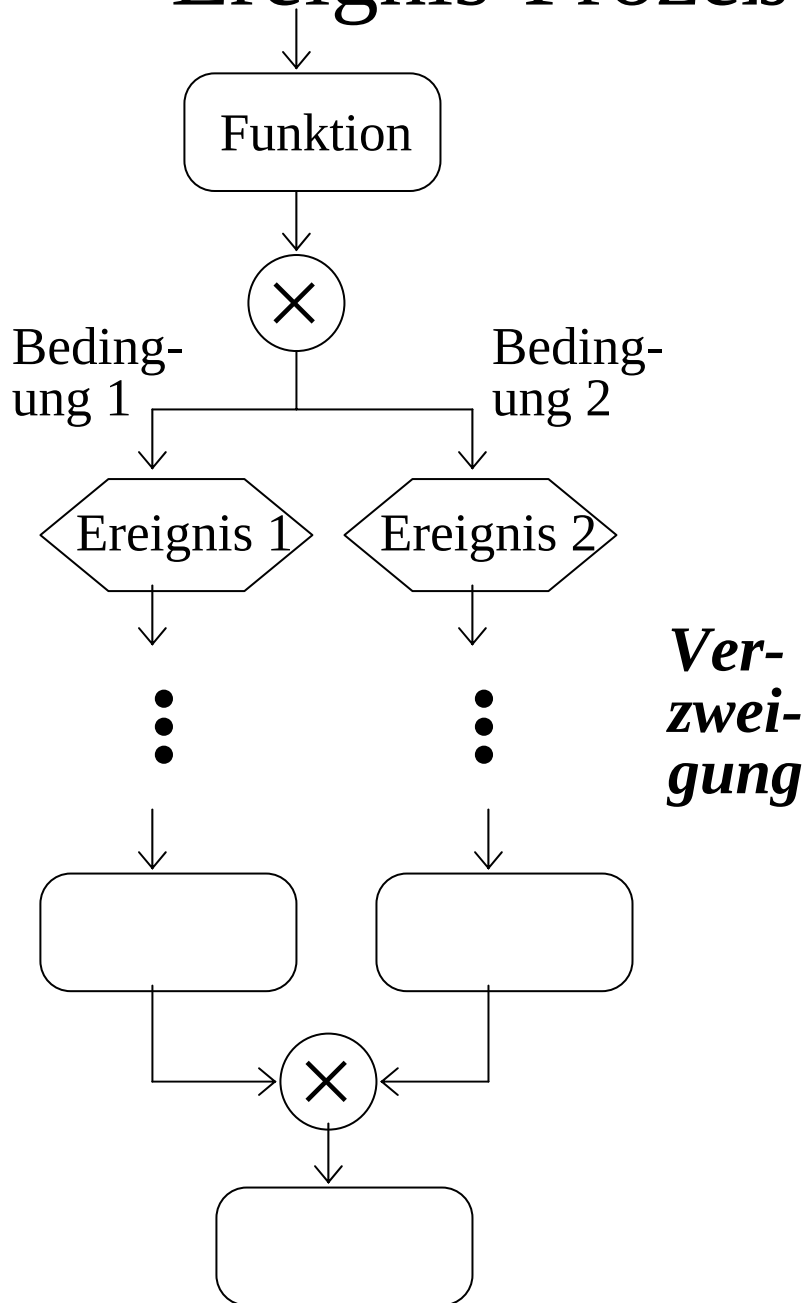
E-Commerce Sub-Workflows



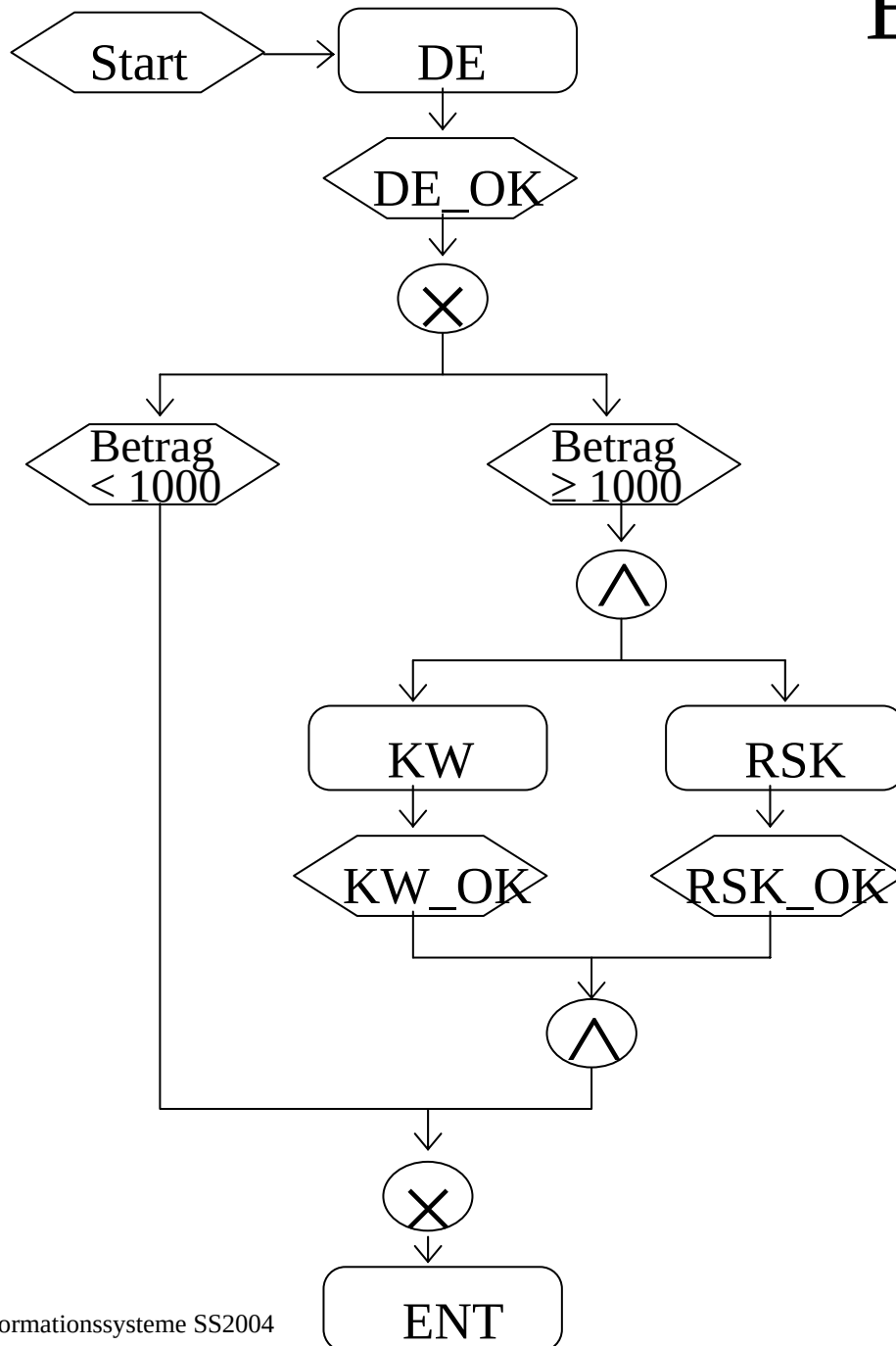
Ereignis-Prozeß-Ketten (EPKs) (1)



Ereignis-Prozeß-Ketten (EPKs) (2)

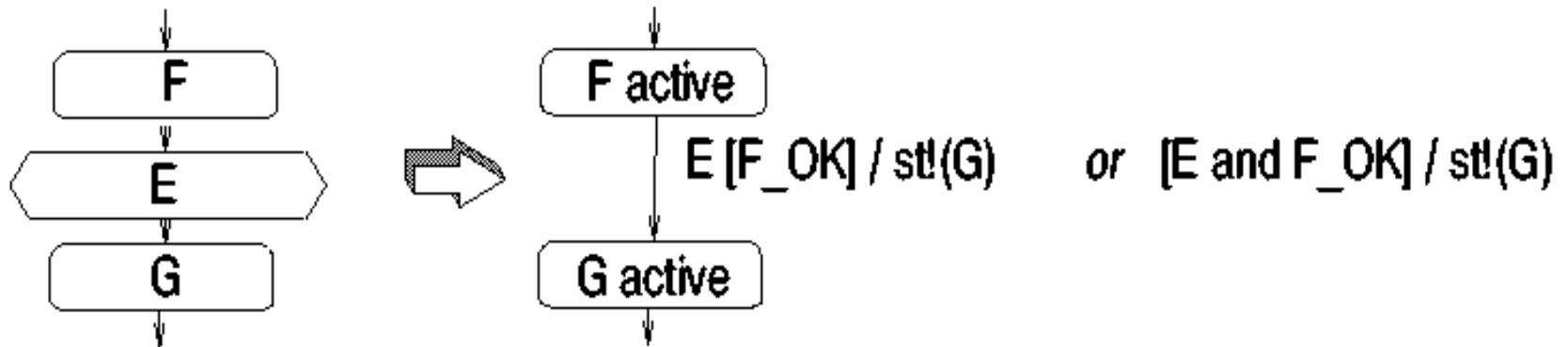


EPK-Beispiel



Import from BPR Tools

Principle:

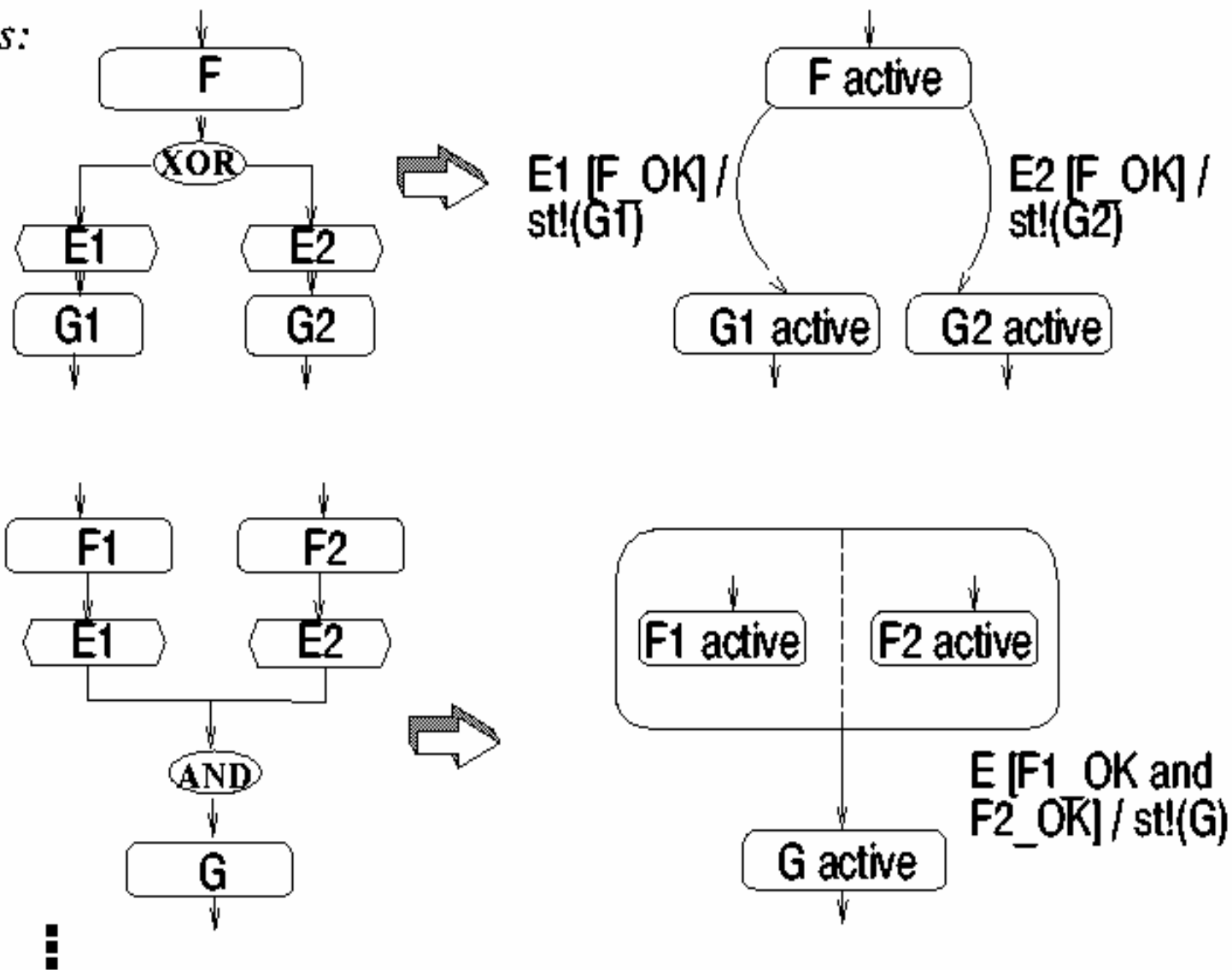


Event process chains
(EPCs à la Aris Toolset):

- process decomposed into functions
- completed functions raise events that trigger further functions
- control-flow connectors

Import from BPR Tools (continued)

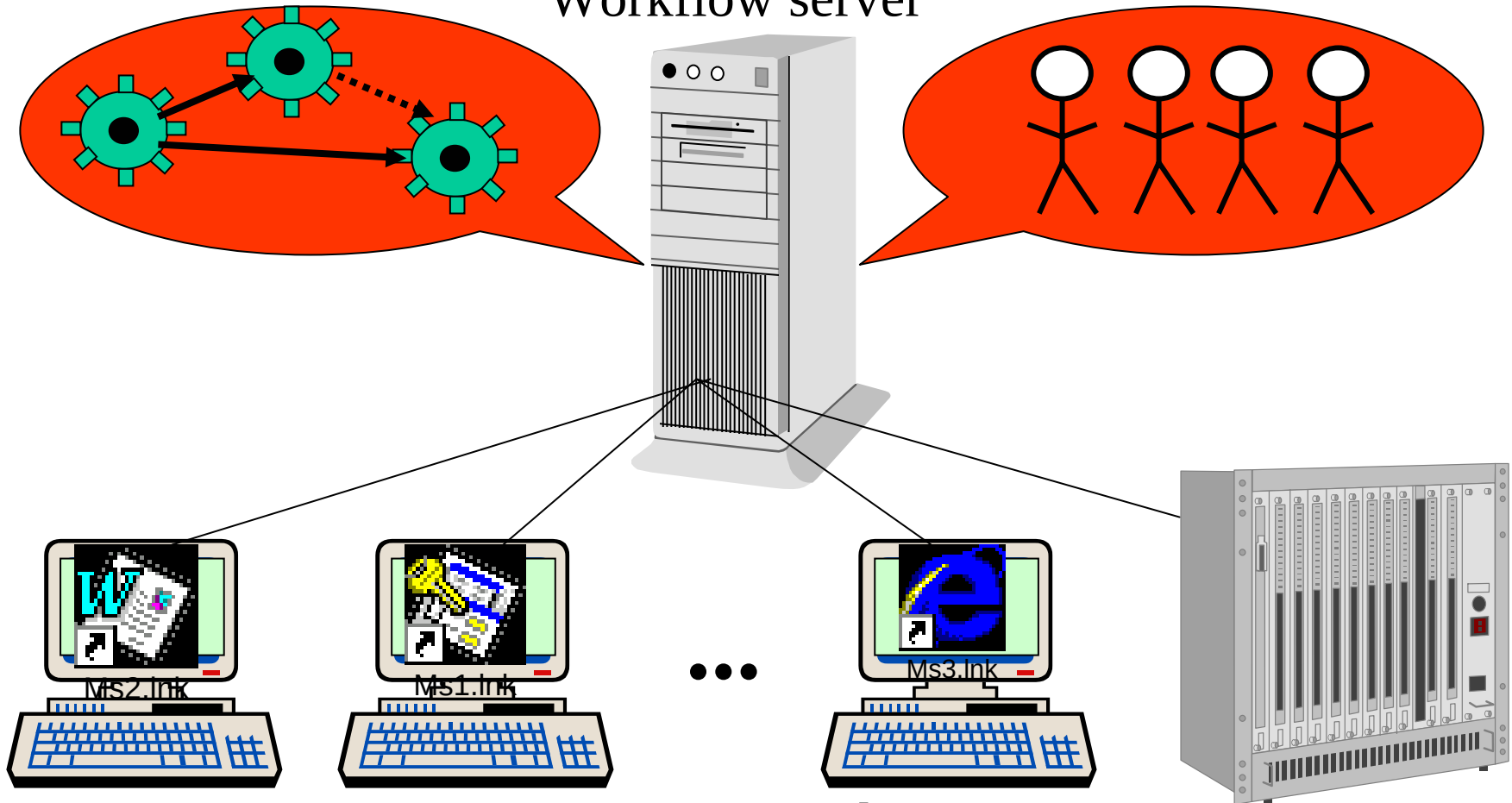
Some Subtleties:



13.2 Workflow Management System Architecture

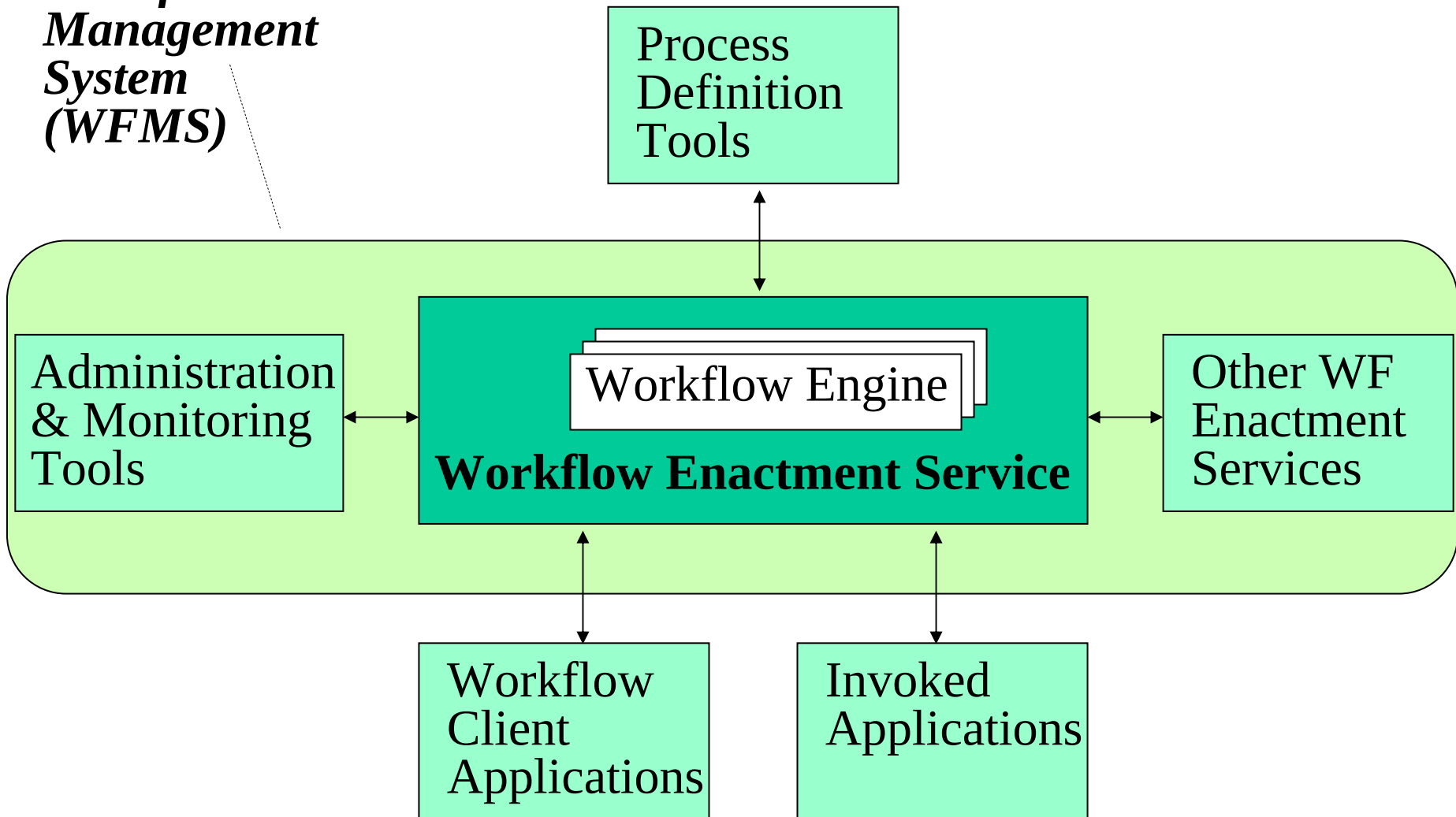
Workflow specification

Workflow server

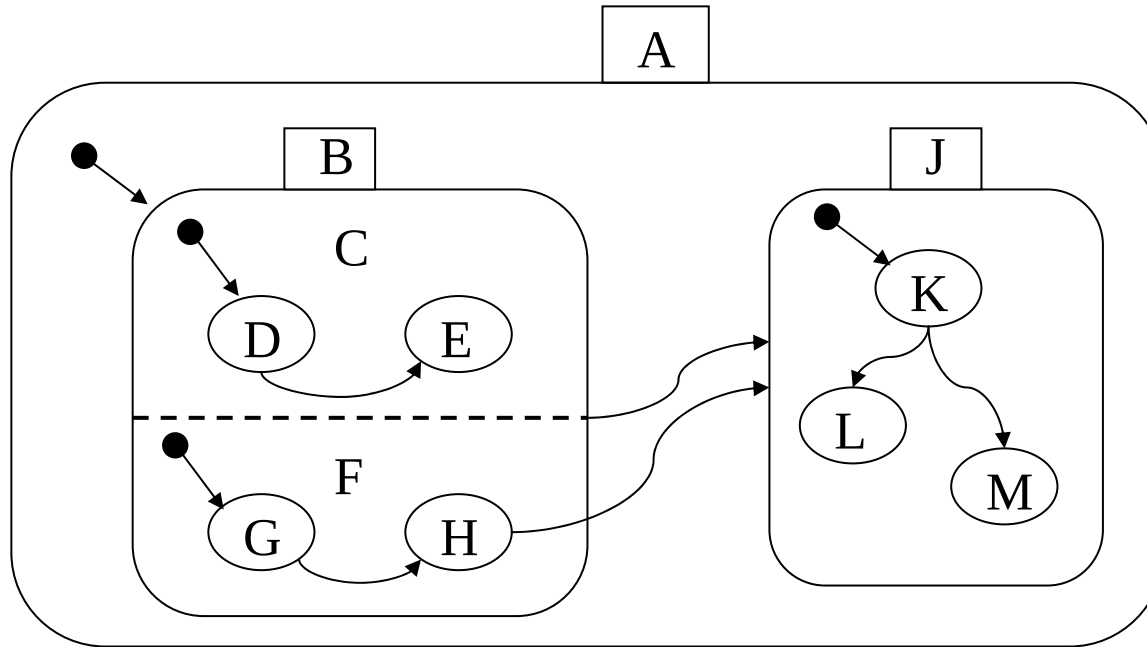


WfMC Reference Architecture

*Workflow
Management
System
(WFMS)*



13.3 Abstract Syntax of Statecharts (1)



State set S

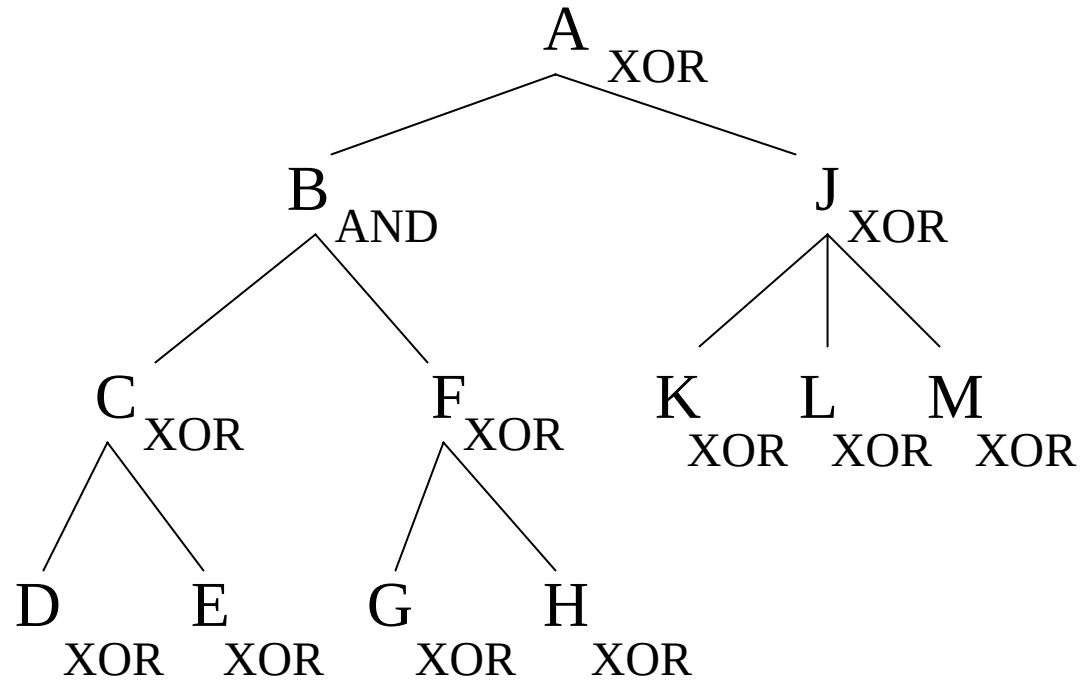
State tree (with node types AND or XOR)

Transition t : (source, target, $[c]/a$)

Transition set T

Variable set V

Abstract Syntax of Statecharts (2)



Operational Semantics of Statecharts (1)

Execution state of statechart (S, T, V) :

subset $states \subseteq S$ of currently active states s.t.

- root of S is in $states$
- if s in $states$ and type of s is AND then all children of s are in $states$
- if s in $states$ and type of s is XOR then exactly one child of s is in $states$

Execution context of statechart (S, T, V) :

current values of variables defined by $val: V \rightarrow \text{Dom}$

Configuration of statechart (S, T, V) : $(states, val)$

Initial configuration

Operational Semantics of Statecharts (2)

Evaluation of expression in configuration:
 $\text{eval}(\text{expr}, \text{conf})$ defined inductively

Effect of action on context:
modification of variable values in val

fire(conf) = set of transitions
 $t = (\text{source}, \text{target}, [\text{cond}]/\text{action})$
with $\text{source}(t)$ in states for which $\text{eval}(\text{cond}, \text{conf}) = \text{true}$

Operational Semantics of Statecharts (3)

for transition t :

- $a = \text{lca}(\text{source}(t), \text{target}(t))$
- $\text{src}(t) = \text{child of } a \text{ in subtree of } \text{source}(t)$
- $\text{tgt}(t) = \text{child of } a \text{ in subtree of } \text{target}(t)$

when t fires:

- set of left states **$\text{source}^*(t)$** :
 - $\text{src}(t)$ is in $\text{source}^*(t)$
 - if s in $\text{source}^*(t)$ then all children of s are in $\text{source}^*(t)$
- set of entered states **$\text{target}^*(t)$** :
 - $\text{tgt}(t)$ and $\text{target}(t)$ are in $\text{target}^*(t)$
 - if s in $\text{target}^*(t)$ and type of s is AND then all children of s are in $\text{target}^*(t)$
 - if s in $\text{target}^*(t)$ and type of s is XOR then exactly one child of s with initial transition is in $\text{target}^*(t)$

Operational Semantics of Statecharts (4)

For a given configuration $\text{conf} = (\text{states}, \text{val})$ a **successor configuration** $\text{conf}' = (\text{states}', \text{val}')$ is derived by selecting one transition t from $\text{fire}(\text{conf})$ with the effect:

- $\text{states}' = \text{states} - \text{source}^*(t) \cup \text{target}^*(t)$
- val' captures the effect of $\text{action}(t)$ and equals val otherwise

The operational semantics of a statechart (S, V, T) is the set of all possible executions along configurations $\text{conf}_0, \text{conf}_1, \text{conf}_2, \dots$ with

- initial configuration conf_0 and
- conf_{i+1} being a successor configuration of conf_i

Digression: Finite State Automata

Definition:

Ein endlicher Automat (finite state automaton) ist ein 5-Tupel $M = (Z, \Sigma, \delta, z_0, E)$ mit

- einer endlichen Zustandsmenge Z
- einem Alphabet (d.h. einer endlichen Menge von Zeichen) Σ
- einer Transitionsfunktion $\delta: Z \times \Sigma \rightarrow Z$
- einem Startzustand z_0
- einer Menge von Endzuständen $E \subseteq Z$

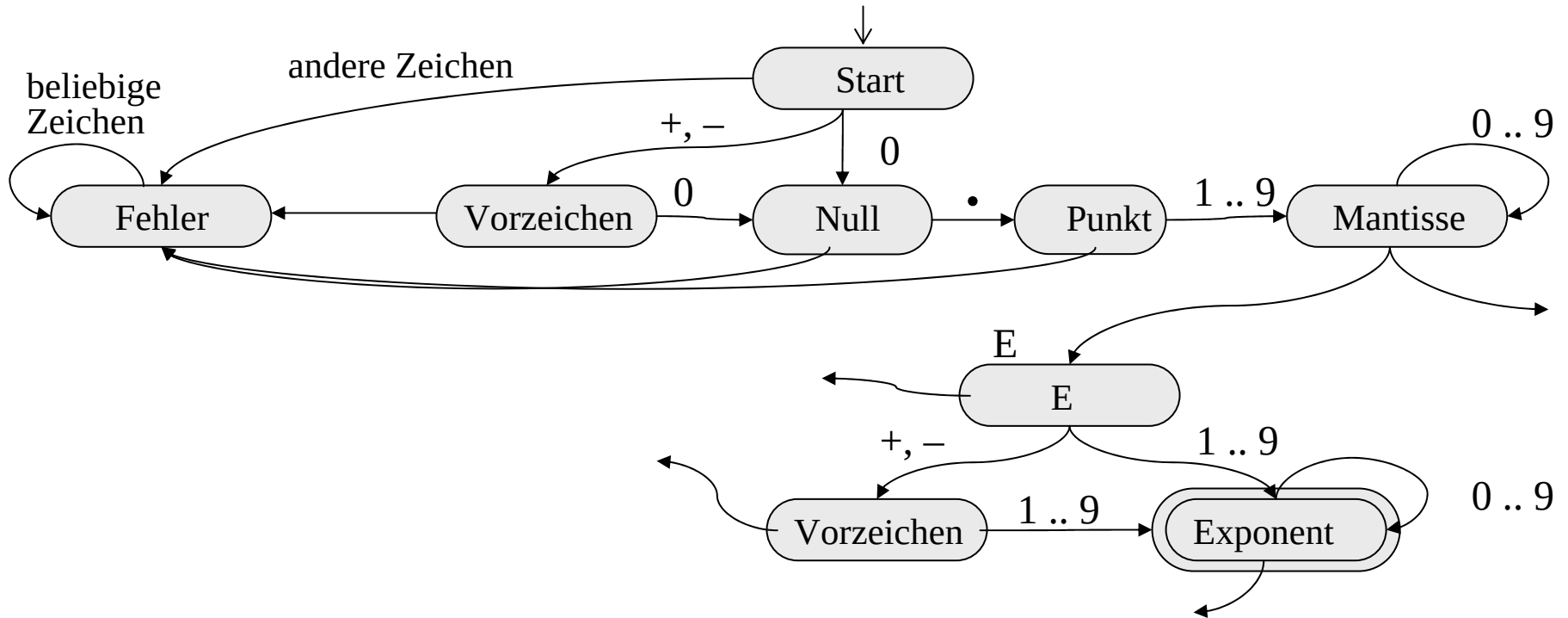
M geht in $z \in Z$ mit Eingabe $x \in \Sigma$ in $\delta(z, x) \in Z$ über.

δ wird homomorph zur Funktion $\delta^*: Z \times \Sigma^* \rightarrow Z$ erweitert:

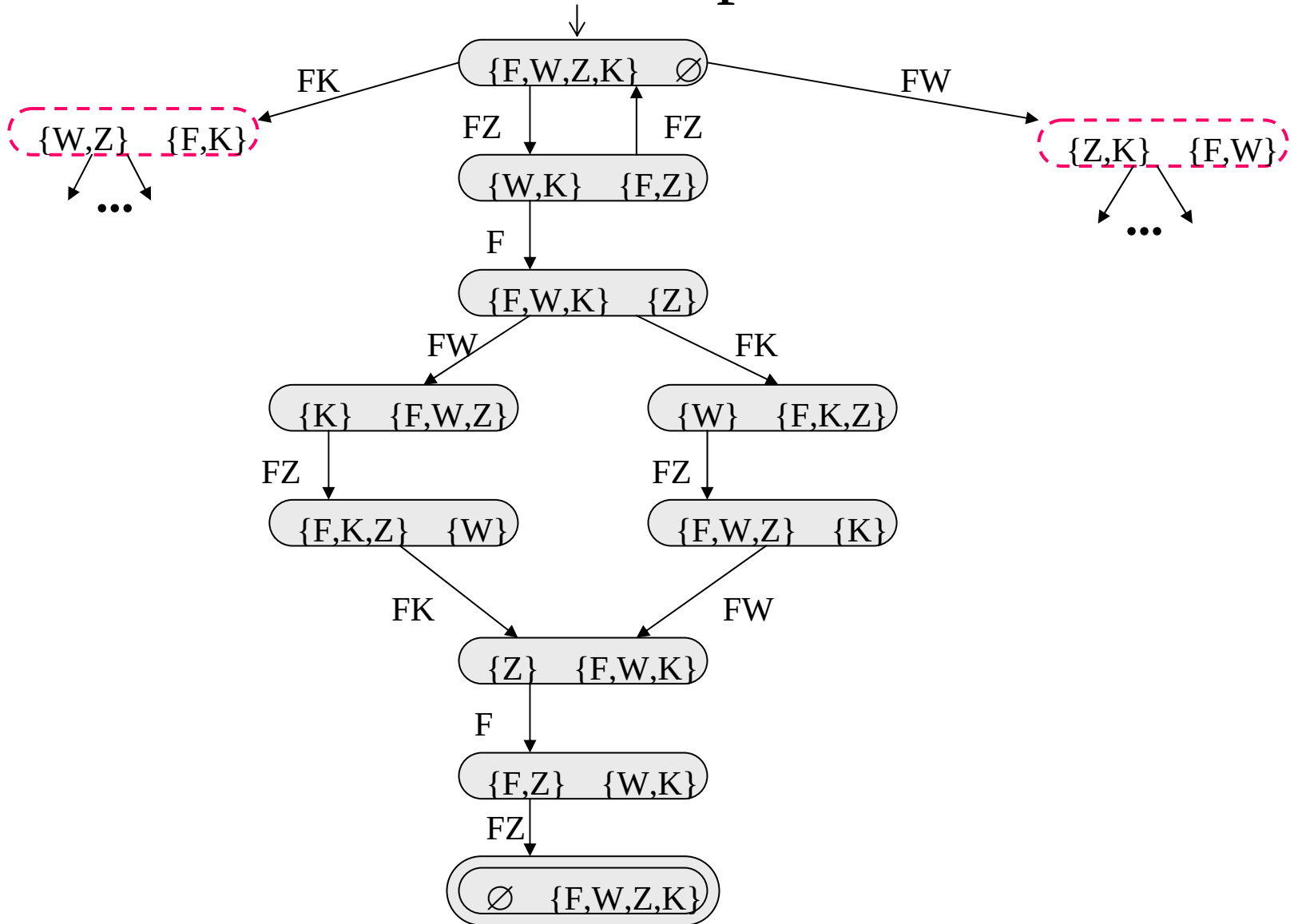
$$\delta^*(z, au) = \delta^*(\delta(z, a), u) \text{ mit } z \in Z, a \in \Sigma, u \in \Sigma^*.$$

Die Menge $L(M) = \{w \in \Sigma^* \mid \delta^*(z_0, w) \in E\} \subseteq \Sigma^*$
ist die vom Automat M akzeptierte Sprache.

FSA Example 1



FSA Example 2



Mapping Statecharts into FSAs

Represent SC configurations as states of a FSA:

Step 1:

abstract conditions on infinite-domain variables into Boolean vars

formal mapping: $\psi_1: \text{val} \rightarrow B_1 \times B_2 \times \dots \times B_m$

Step 2:

capture set of active SC states (in SC hierarchy and in components)

by powerset automaton $\psi_2: \text{states} \rightarrow 2^S =: Z$

Step 3:

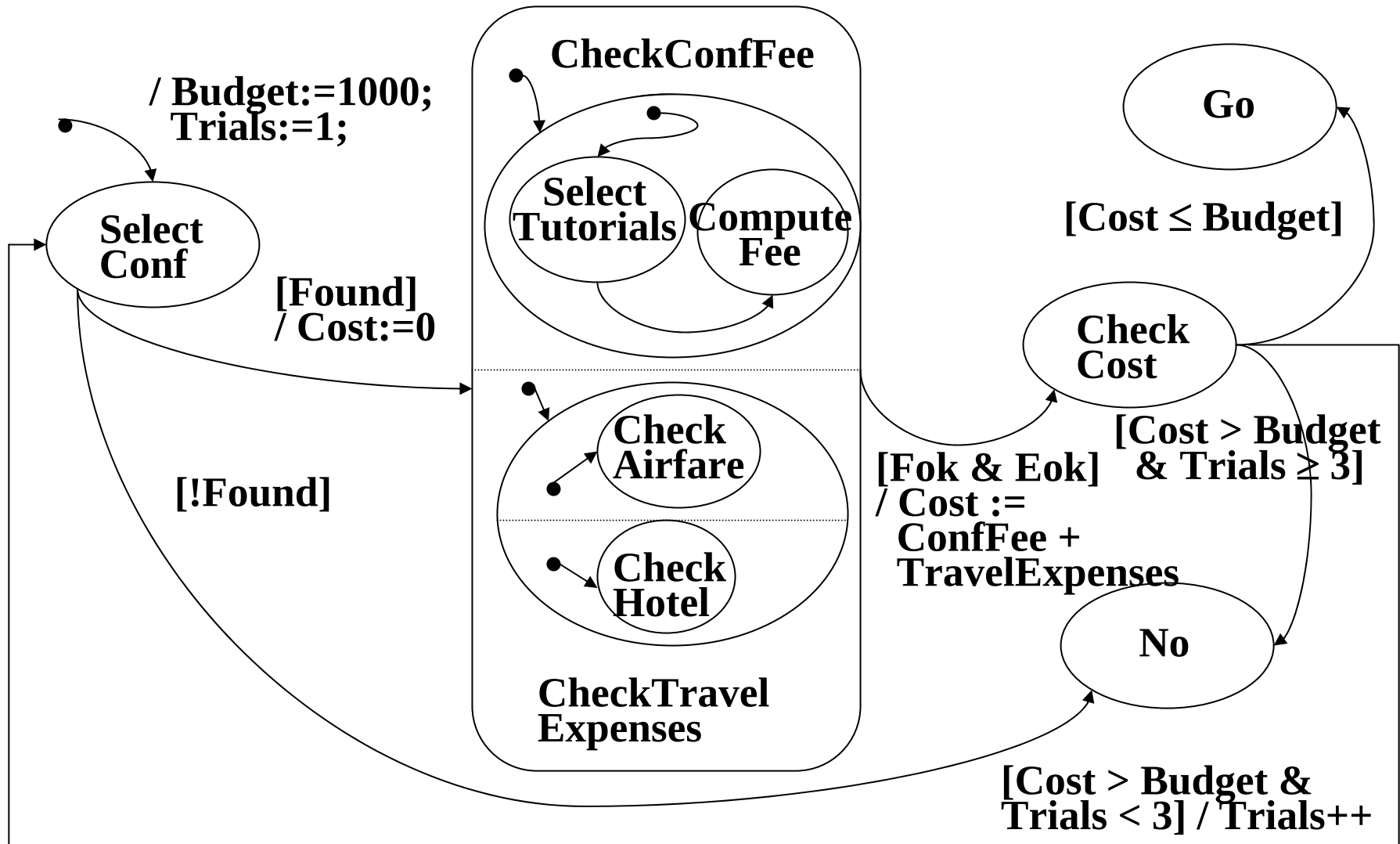
encode SC context into extended state space of FSA

by an injective mapping $\psi_3: Z \times B_1 \times B_2 \times \dots \times B_m \rightarrow Z'$

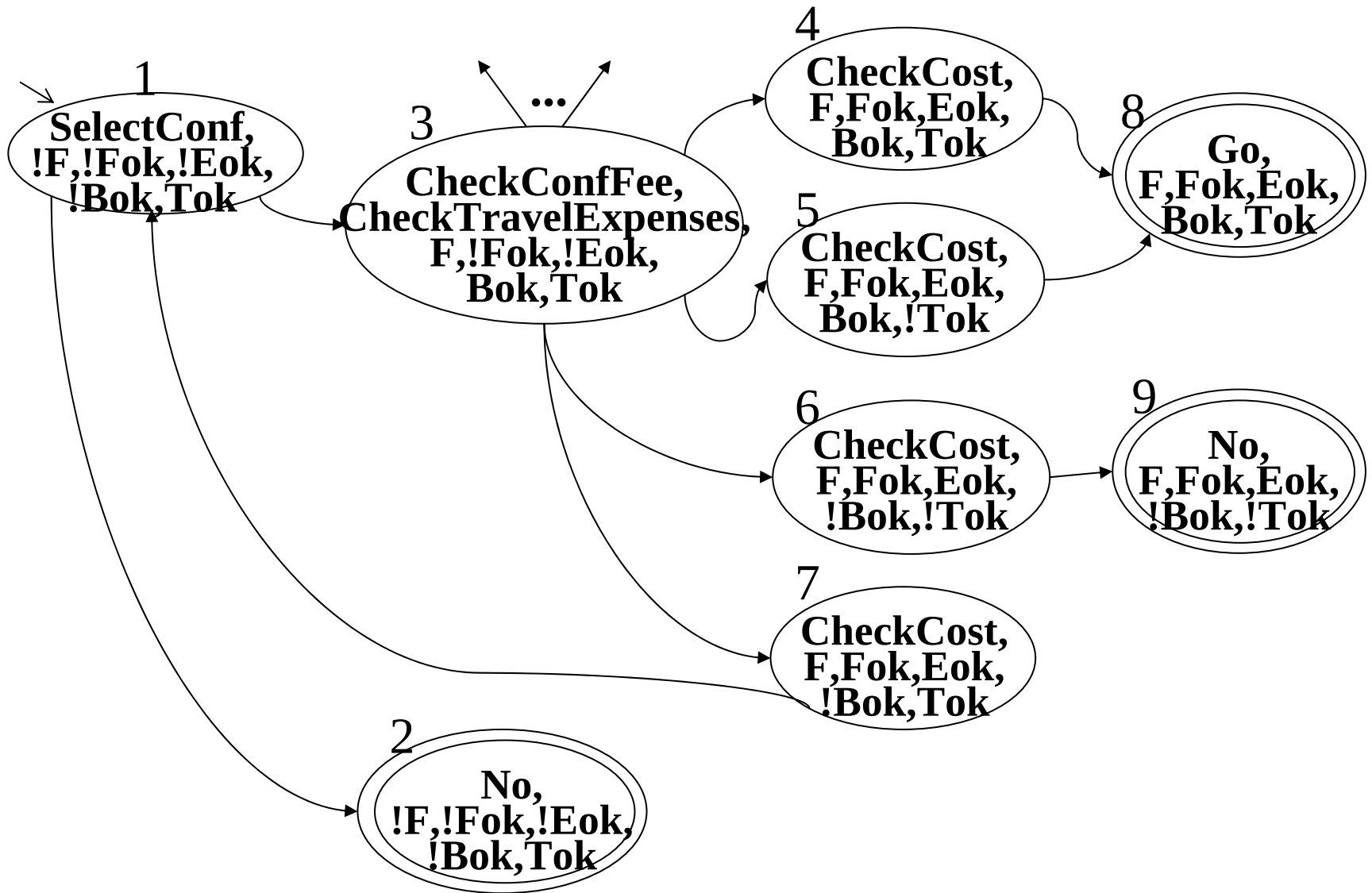
such that there is a transition from z_1 to z_2 in the FSA

iff $\psi_3^{-1}(z_2)$ is a possible successor configuration of $\psi_3^{-1}(z_1)$ in the SC

Example: From SC To FSA (1)



Example: From SC To FSA (2)



13.4 Guaranteed Behavior and Outcome of Mission-critical Workflows

Crucial for workflows in
banking, medical applications, electronic commerce, etc.

- **Safety** properties (invariants):
nothing bad ever happens
- **Liveness** properties (termination, fairness, etc.):
something good eventually happens

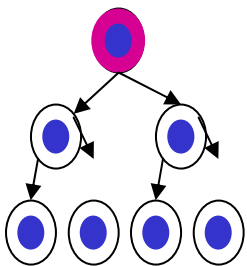
● Mathematical model	→	Finite-state automaton
● Formalization of properties	→	Temporal logic
● Verification method	→	Model checking

CTL: Computation Tree Logic

- propositional logic formulas
- quantifiers ranging over execution paths
- modal operators referring to future states

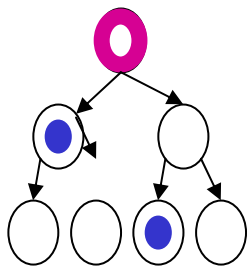
all
next:

$AX\ p$



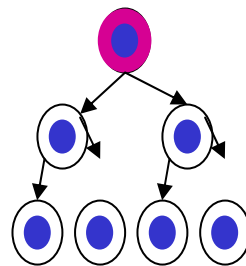
exists
next:

$EX\ p$



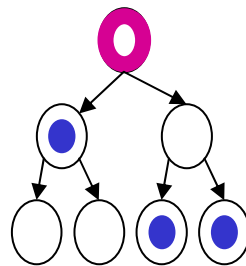
all
globally:

$AG\ p$



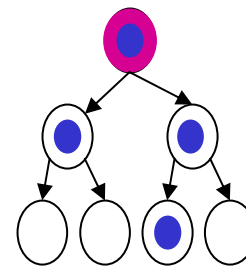
all finally
(inevitably): globally:

$AF\ p$



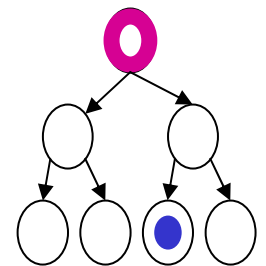
exists
globally:

$EG\ p$



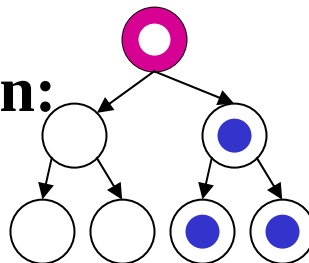
exists finally
(possibly):

$EF\ p$



combination:

$EF\ AG\ p$



Critical Properties of the Example Workflow

formalized in CTL (Computation Tree Logic)

- *Can we ever exceed the budget ?*

$\text{not EF (in(Go) and !Bok)}$

$\equiv \text{AG (not in(Go) or Bok)}$

- *Do we always eventually reach a decision ?*

$\text{AF (in(Go) or in(No))}$

- *Can the trip still be approved after a proposal that would have exceeded the budget ?*

$\text{EF ((in(CheckCost) and !Bok) } \Rightarrow \text{ (EF (in(Go))))}$

CTL Syntax

Definition:

Eine atomare *CTL-Formel* ist eine aussagenlogische Formel über elementaren Aussagen (bzw. Booleschen Variablen).

Die Menge der in CTL erlaubten Formeln ist induktiv wie folgt definiert:

- Jede atomare CTL-Formel ist eine Formel.
- Wenn P und Q Formeln sind, dann sind auch EX (P), AX (P), EG (P), AG (P), EF (P), AF (P), (P), $\neg P$, $P \wedge Q$, $P \vee Q$, $P \Rightarrow Q$ und $P \Leftrightarrow Q$ Formeln.

CTL Semantik (1)

Definition:

Gegeben sei eine Menge P atomarer aussagenlogischer Formeln.

Eine *Kripke-Struktur* M über P ist ein 4-Tupel (S, s_0, R, L) mit

- einer endlichen Zustandsmenge S ,
- einem Startzustand $s_0 \in S$,
- einer Transitionsrelation $R \subseteq S \times S$,
- einer Funktion $L: S \rightarrow 2^P$, die einem Zustand wahre Aussagen zuordnet.

Definition:

Eine Kripke-Struktur $M = (S, s_0, R, L)$ ist ein *Modell* einer Formel F , wenn $M, s_0 \models F$.

Eine Formel heißt *erfüllbar*, wenn sie mindestens ein Modell hat, ansonsten *unerfüllbar*.

Eine Formel F heißt *allgemeingültig* (oder Tautologie), wenn jede Kripke-Struktur über den atomaren Aussagen von F ein Modell von F ist.

CTL Semantik (2)

Definition:

Die *Interpretation* ψ einer Formel F mit atomaren Aussagen P ist eine Abbildung auf eine Kripke-Struktur $M=(S, s_0, R, L)$ über Aussagen P so dass die Wahrheitswerte von Teilformeln p bzw. p_1, p_2 von F in den Zuständen s von M , in Zeichen: $M, s \models p$, wie folgt sind:

- (i) $M, s \models p$ mit einer aussagenlogischen Formel p gilt g.d.w. $p \in L(s)$;
- (ii) $M, s \models \neg p$ g.d.w. nicht $M, s \models p$ gilt;
- (iii) $M, s \models p_1 \wedge p_2$ g.d.w. $M, s \models p_1$ und $M, s \models p_2$;
- (iv) $M, s \models p_1 \vee p_2$ g.d.w. $M, s \models p_1$ oder $M, s \models p_2$;
- (v) $M, s \models EX\ p$ g.d.w. es $t \in S$ gibt mit $(s, t) \in R$ und $M, t \models p$;
- (vi) $M, s \models AX\ p$ g.d.w. für alle $t \in S$ mit $(s, t) \in R$ gilt: $M, t \models p$;
- (vii) $M, s \models EG\ p$ g.d.w. es $t_1, \dots, t_k \in S$ gibt mit $t_1 = s, (t_i, t_{i+1}) \in R$ für alle i und $t_k = t_j$ für ein $j: 1 \leq j < k$ oder t_k ohne Nachfolger, so dass $M, t_i \models p$ für alle i ;
- (viii) $M, s \models AG\ p$ g.d.w. für alle $t \in S$ mit $(s, t) \in R^*$ gilt: $M, t \models p$;
- (ix) $M, s \models EF\ p$ g.d.w. es $t \in S$ gibt mit $(s, t) \in R^*$ und $M, t \models p$;
- (x) $M, s \models AF\ p$ g.d.w. es für alle $t \in S$ mit $(s, t) \in R^*$ einen Zustand $t' \in S$ gibt mit a) $(t, t') \in R^*$ oder b) $(s, t') \in R^*$ und $(t', t) \in R^*$, so dass $M, t' \models p$ gilt.

Model Checking

Für CTL-Formel F und Transitionssystem (Kripke-Struktur) M teste, ob M ein Modell von F ist, indem man
induktiv alle Zustände von M mit q markiert, in denen
die Teilformel q von F wahr ist.

Sei q eine Teilformel von F , seien $p, p1, p2$ direkte Teilformeln von q und seien $P, P1, P2$ die mit $p, p1, p2$ markierten Zustände von M .

(i) q ist eine atomare Aussage (Boolesche Variable):

Markiere alle Zustände s mit $q \in L(s)$ mit q

(ii) q hat die Form $\neg p$: Markiere $S - P$ mit q

(iii) q hat die Form $p1 \wedge p2$: Markiere $P1 \cap P2$ mit q

(iv) q hat die Form $p1 \vee p2$: Markiere $P1 \cup P2$ mit q

(v) q hat die Form $EX p$:

**Markiere alle Vorgänger von P mit q , also alle $s \in S$,
für die es ein $x \in P$ gibt mit $R(s, x)$**

(vi) q hat die Form $AX p$:

Markiere s mit q , wenn alle Nachfolger von s mit p markiert sind

Model Checking: Fall EF

(vii) q hat die Form $EF\ p$:

Löse Rekursion $EF\ p \Leftrightarrow p \vee EX\ (EF\ p)$.

(Fixpunktgleichung $Q = P \cup \text{pred}(Q)$)

```
Q := P;
```

```
Qnew := Q  $\cup$  pred(Q);
```

```
while not (Q = Qnew) do
```

```
    Q := Qnew;
```

```
    Qnew := Q  $\cup$  pred(Q);
```

```
od;
```


Model Checking: Fall EG

(viii) q hat die Form $EG\ p$:

Löse Rekursion $EG\ p \Leftrightarrow p \wedge EX\ (EG\ p)$:

```
Q := P;  
Qnew := Q ;  
repeat  
  for each s in Q do  
    if s has successors and  
       no successor of s is in Q  
    then Qnew := Q - {s}; fi;  
od;  
until (Q = Qnew);
```

Model Checking: Fall AG

(ix) q hat die Form $AG\ p$:

Löse Rekursion $AG\ p \Leftrightarrow p \wedge AX\ (AG\ p)$

```
Q := P;  
repeat  
    Qnew := Q;  
    for each s in Q do  
        if s has successors and  
           one successor of s is not in Q  
        then Q := Q - {s} fi;  
    od;  
until (Q = Qnew);
```

Alternativ wegen $AG\ p \Leftrightarrow \neg EF\ (\neg p)$:

Berechne Zustandsmenge Q' zur Formel $EF\ (\neg p)$
und markiere dann die Zustandsmenge $S - Q'$ mit q .

Model Checking: Fall AF

(x) q hat die Form AF p:

Löse Rekursion $AF\ p \Leftrightarrow p \vee AX\ (AF\ p)$

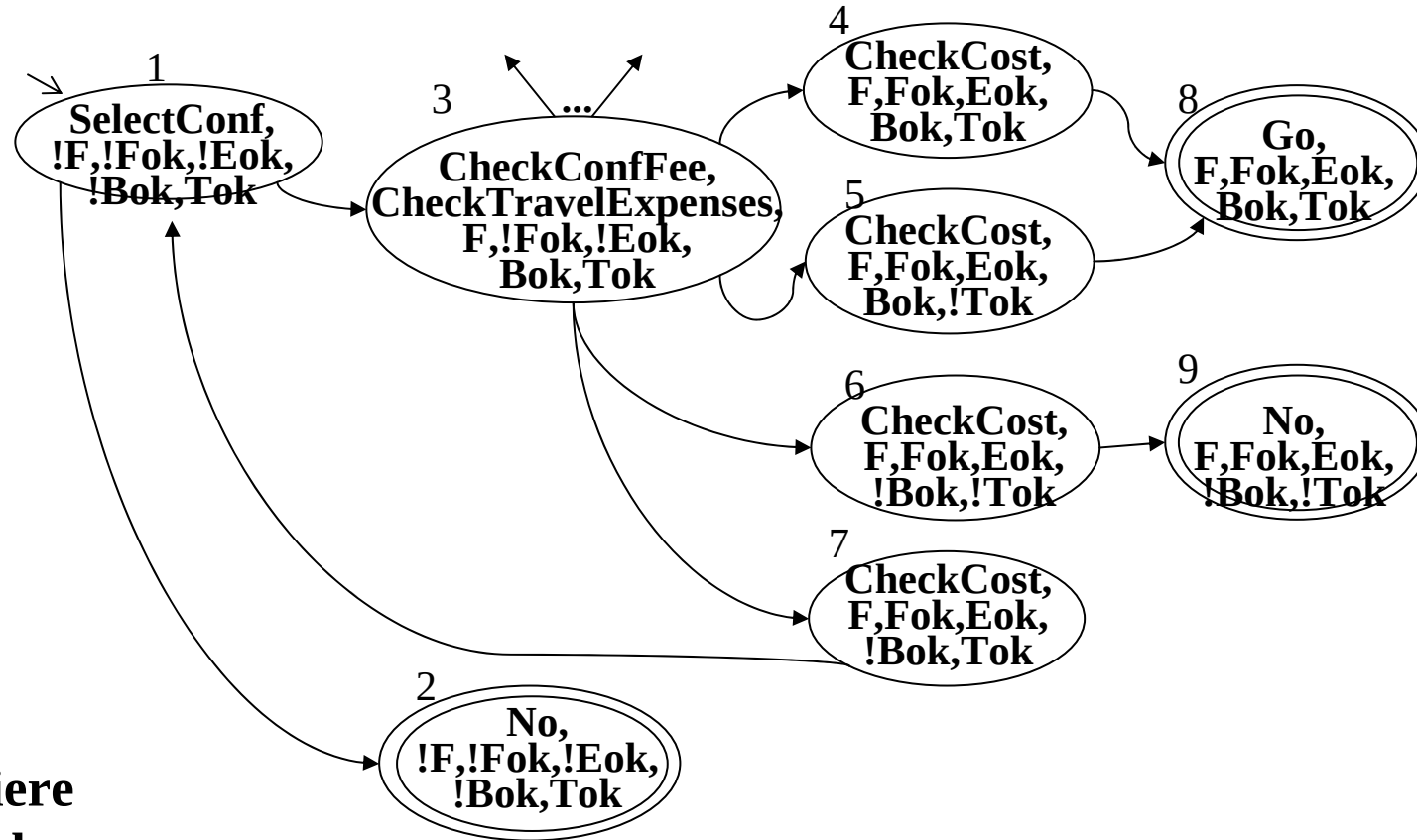
```
Q := P;  
repeat  
    Qnew := Q;  
    for each s in pred(Q) do  
        if all successors of s are in Q  
        then Q := Q  $\cup$  {s}; fi;  
    od;  
until (Q = Qnew);
```

Alternativ wegen $AF\ p \Leftrightarrow \neg EG\ (\neg p)$:

**Berechne Zustandsmenge Q' zur Formel $EG\ (\neg p)$
und markiere dann die Zustandsmenge $S - Q'$ mit q.**

Model Checking: Beispiel 1

AG (not in(Go) or Bok)



Markiere

mit Bok:

mit in(Go):

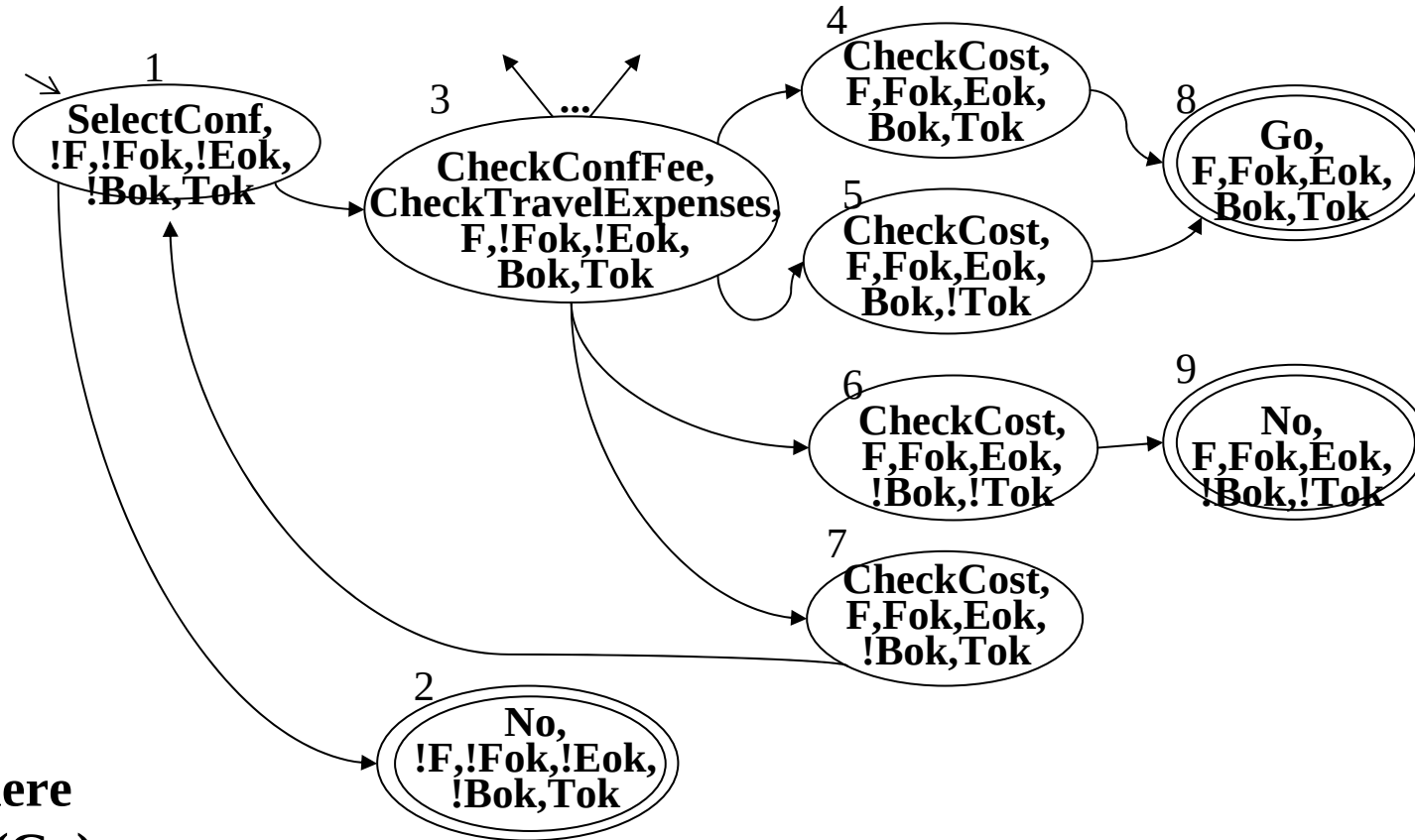
mit $\neg$ in(Go):

mit $(\neg \text{Bok} \Rightarrow \neg \text{in(Go)})$:

mit AG $(\neg \text{Bok} \Rightarrow \neg \text{in(Go)})$:

Model Checking: Beispiel 2

AF (in(Go) $\vee$ in(No))



Markiere

mit in(Go):

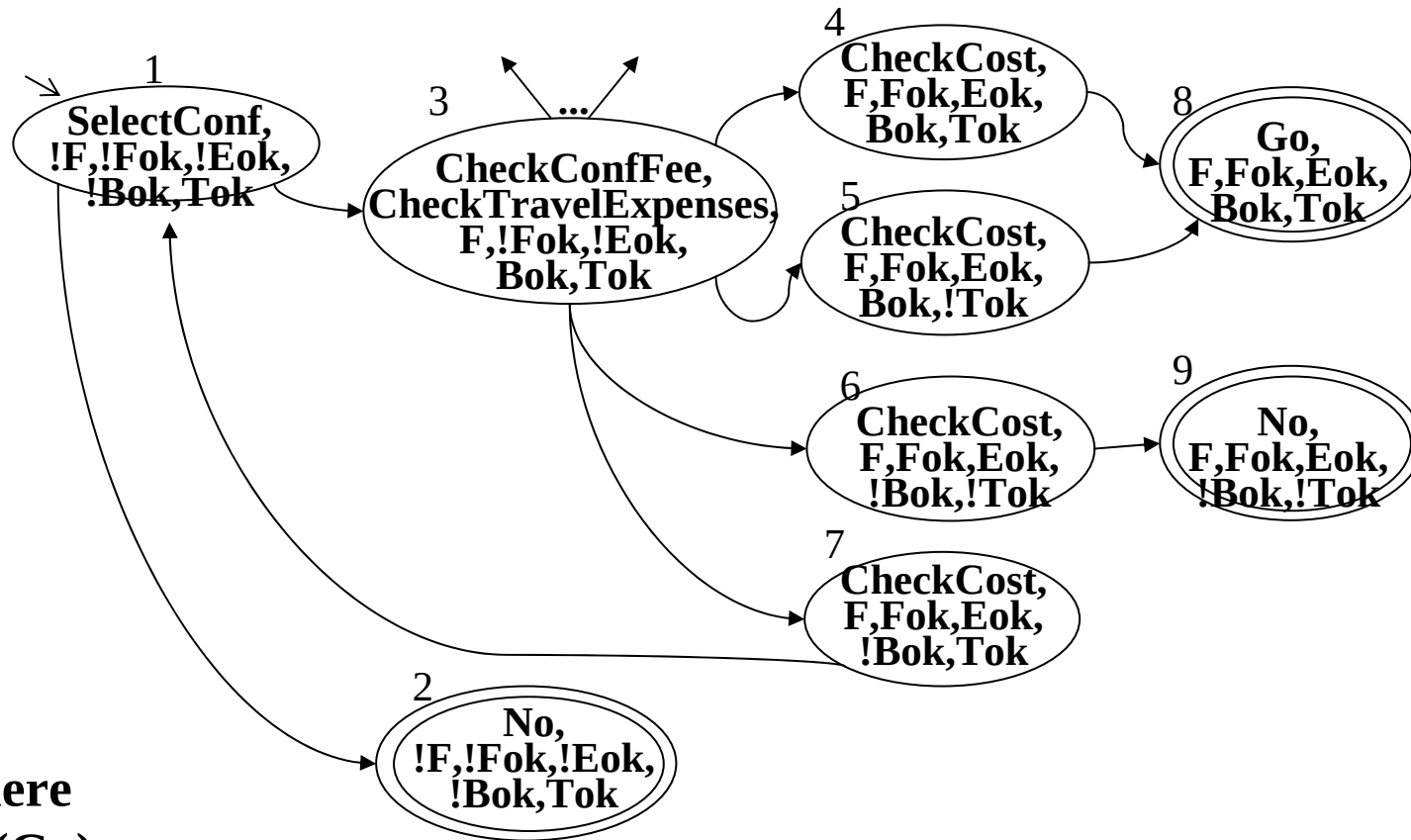
mit in(No):

mit in(Go) $\vee$ in(No):

mit AF (in(Go) $\vee$ in(No)):

Model Checking: Beispiel 3

$EF \ ((in(CheckCost) \text{ and } !Bok) \Rightarrow (EF (in(Go))))$



Markiere

mit in(Go):

mit EF (in(Go)):

mit not in(CheckCost) or Bok:

mit (in(CheckCost) and !Bok) => (EF (in(Go)):

mit EF ((in(CheckCost) and !Bok) => (EF (in(Go)))):

Guaranteed Behavior of Workflows

- **Leverage computer-aided verification techniques for finite-state concurrent systems**
 - **Efficiency gain with encoding of FSM as OBDD**
 - **Further requirements:**
 - **User-friendly macros for CTL**
 - **More expressive logic**
 - **Adding assertions on behavior of invoked apps**
 - **Adding real-time (clock variables)**
- **Preserving guaranteed behavior in distributed, failure-prone system environment**
→ **System guarantees**

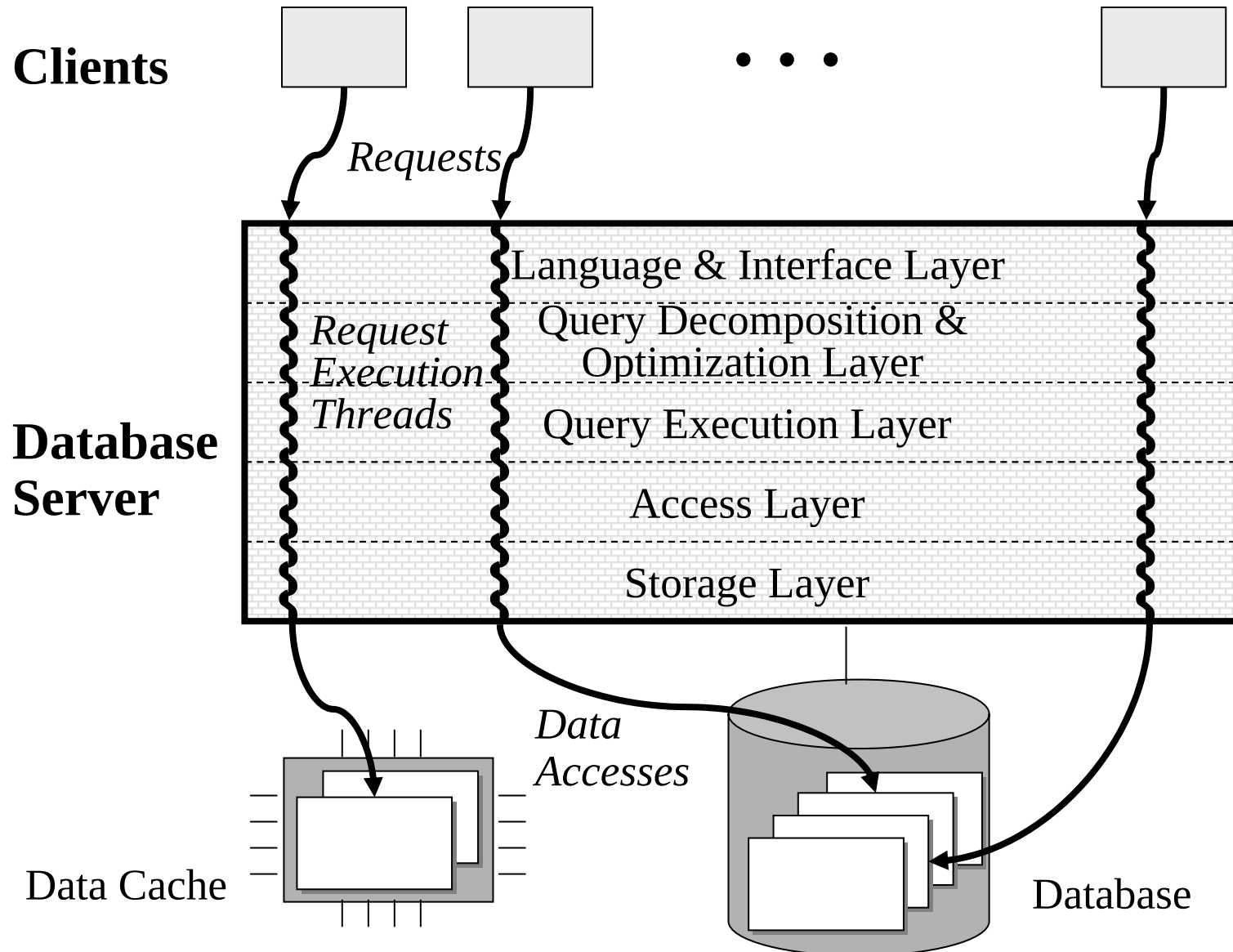
Kapitel 14: Datenspeicherung, Indexstrukturen und Anfrageausführung

14.1 Wie Daten gespeichert werden

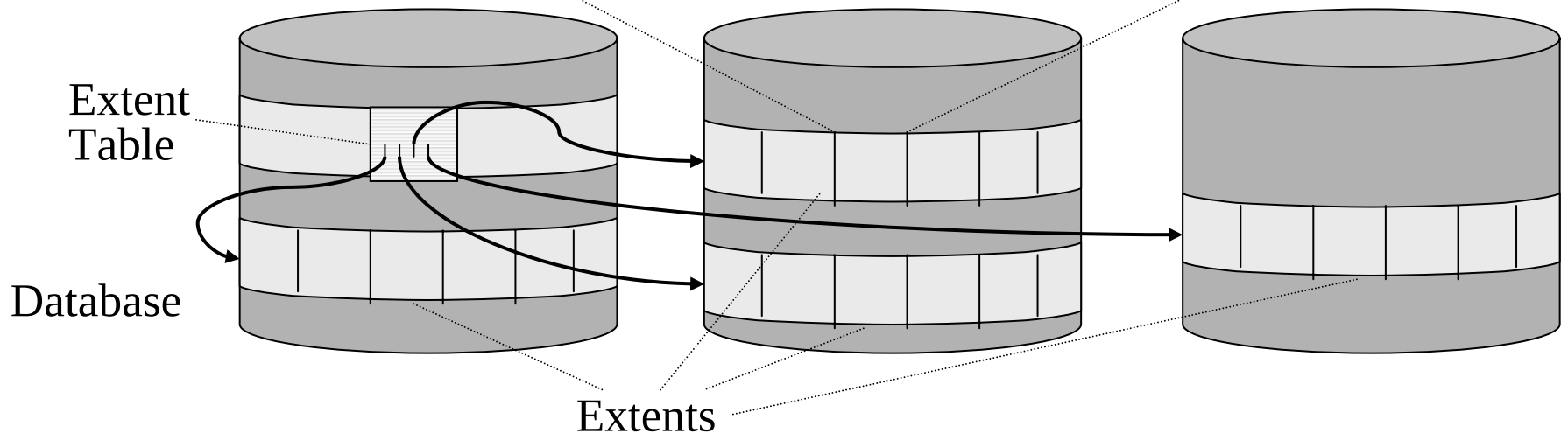
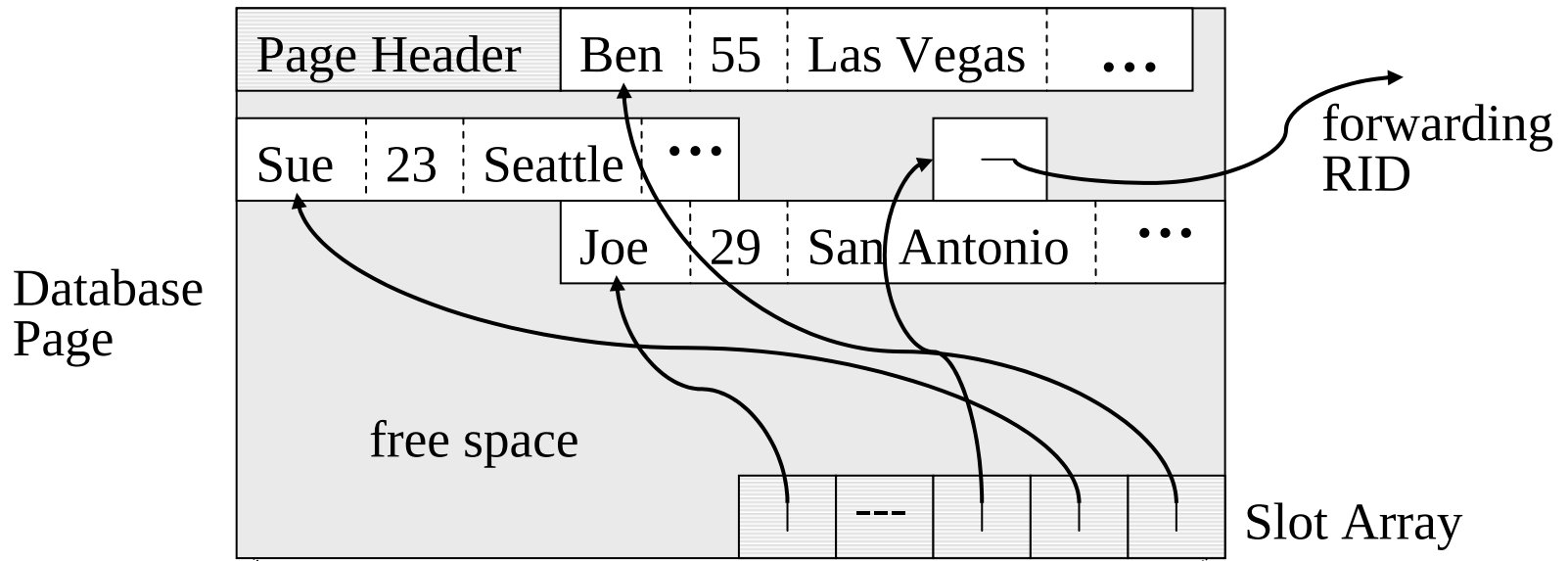
**14.2 Wie auf Daten effizient zugegriffen wird
(Indexstrukturen)**

14.3 Wie Anfragen ausgeführt werden

Interne (idealisierte) Schichtenarchitektur eines DBS



14.1 Wie Daten gespeichert werden

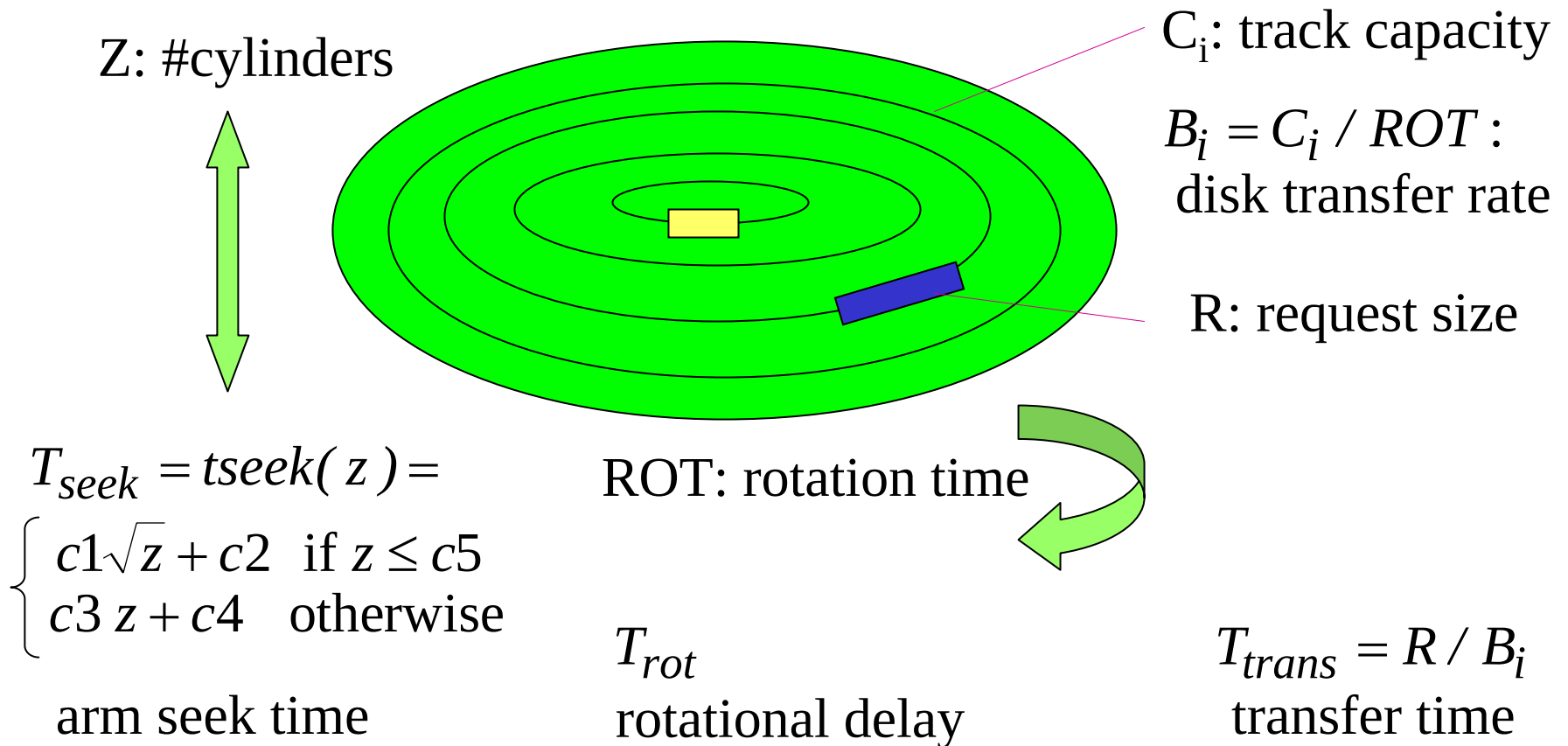


Charakteristika moderner Magnetplatten

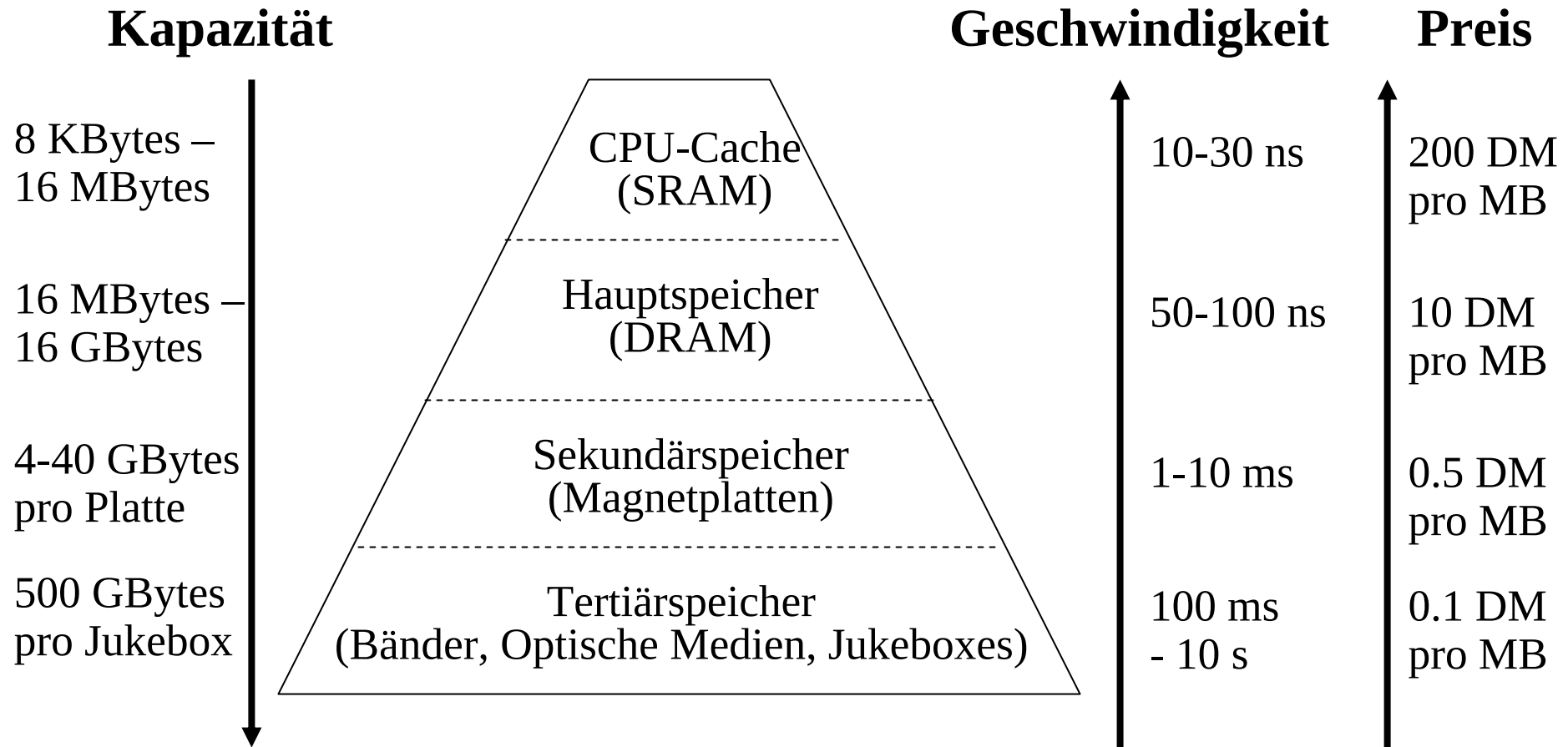
(z.B. Seagate Cheetah 18 mit Ultra-SCSI- oder Fibre-Channel-Schnittstelle)

Durchmesser	3.5 Zoll
Speicherkapazität	18.2 GBytes
Preis	ca. 4000 DM
Größe	ca. 14 x 10 x 4 cm
Gewicht	ca. 1.2 kg
Energieverbrauch	15 Watt
Zuverlässigkeit (MTTF)	800 000 Stunden (> 75 Jahre)
Anzahl Oberflächen	24
Anzahl Zylinder	6962
Spurkapazität	87 bis 128 KBytes
Rotationszeit	6 ms (10000 U/min)
Mittlere Armpositionierungszeit	5.7 ms
Minimale Armpositionierungszeit	0.6 ms
Übertragungsrate	14.5 bis 21.3 MBytes / s
Random I/O auf 1 Block a 4 KB	ca. 9 ms
Sequential I/O auf 30 Blöcke a 4 KB	ca. 12 ms

Leistungsparameter einer Magnetplatte



Charakteristika von Speicherhierarchien



14.2 Wie auf Daten effizient zugegriffen wird (Indexstrukturen)

CREATE [UNIQUE] INDEX index-name
ON table (column, {, column ...})

zur Beschleunigung von:

- *Exact-Match-Selektionen*: $A1 = \text{wert1} \wedge A2 = \text{wert2} \wedge \dots \wedge Ak = \text{wertk}$
(z.B. City = 'Miami' oder City = 'Paris' $\wedge$ State = 'Texas')
- *Bereichs-Selektionen*: $u1 \leq A1 \leq o1 \wedge \dots \wedge uk \leq Ak \leq ok$
(z.B. $21 \leq \text{Age} \leq 30$ oder $21 \leq \text{Age} \leq 30 \wedge \text{Salary} \geq 100\,000$)
- *Prefix-Match-Selektionen*:
 $u1 \leq A1 \leq o1 \wedge u2 \leq A2 \leq o2 \wedge \dots \wedge uj \leq Aj \leq oj$ (mit $j < k$)
(z.B. $21 \leq \text{Age} \leq 30$ bei einem Index über Age, Salary).

Welche Indexstrukturen für welche Attributkombinationen?

→ Problem des physischen Datenbankentwurfs
für DBA oder „Index Wizard“ !

Binäre Suchbäume (für Hauptspeicher)

Schlüssel (mit den ihnen zugeordneten Daten)
bilden die Knoten eines binären Baums
mit der Invariante:

für jeden Knoten t mit Schlüssel $t.key$ und alle Knoten l im linken Teilbaum von t , $t.left$, und alle Knoten r im rechten Teilbaum von t gilt: $l.key \leq t.key \leq r.key$

Suchen eines Schlüssels k :

Traversieren des Pfades von der Wurzel bis zu k bzw. einem Blatt

Einfügen eines Schlüssels k :

Suchen von k und Anfügen eines neuen Blatts

Löschen eines Schlüssel k :

Ersetzen von k durch das „rechteste“ Blatt links von k

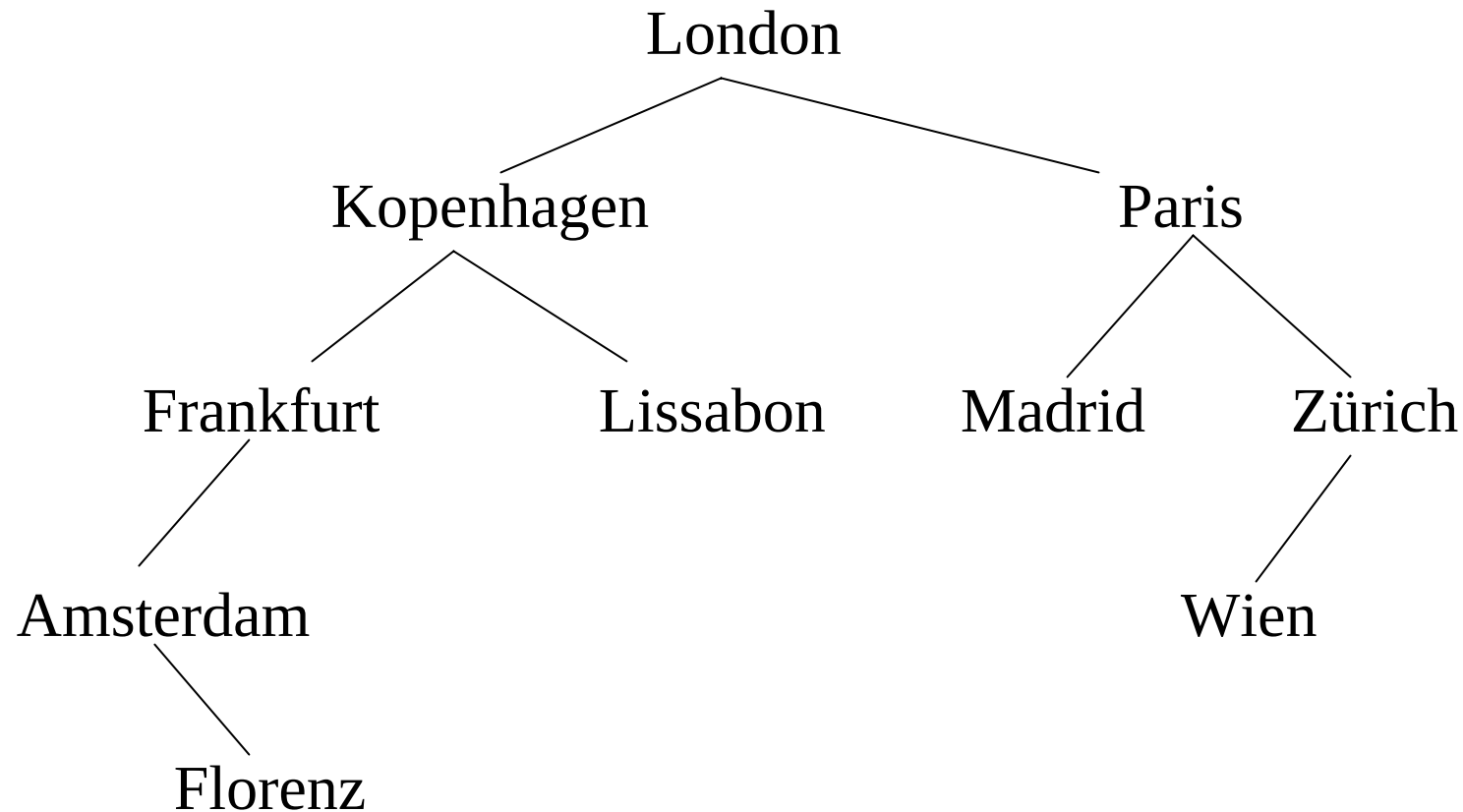
Worst-Case-Suchzeit für n Schlüssel: $O(n)$

bei geeigneten Rebalancierungsalgorithmen

(AVL-Bäume, Rot-Schwarz-Bäume, usw.): $O(\log n)$

Beispiel für einen binären Suchbaum

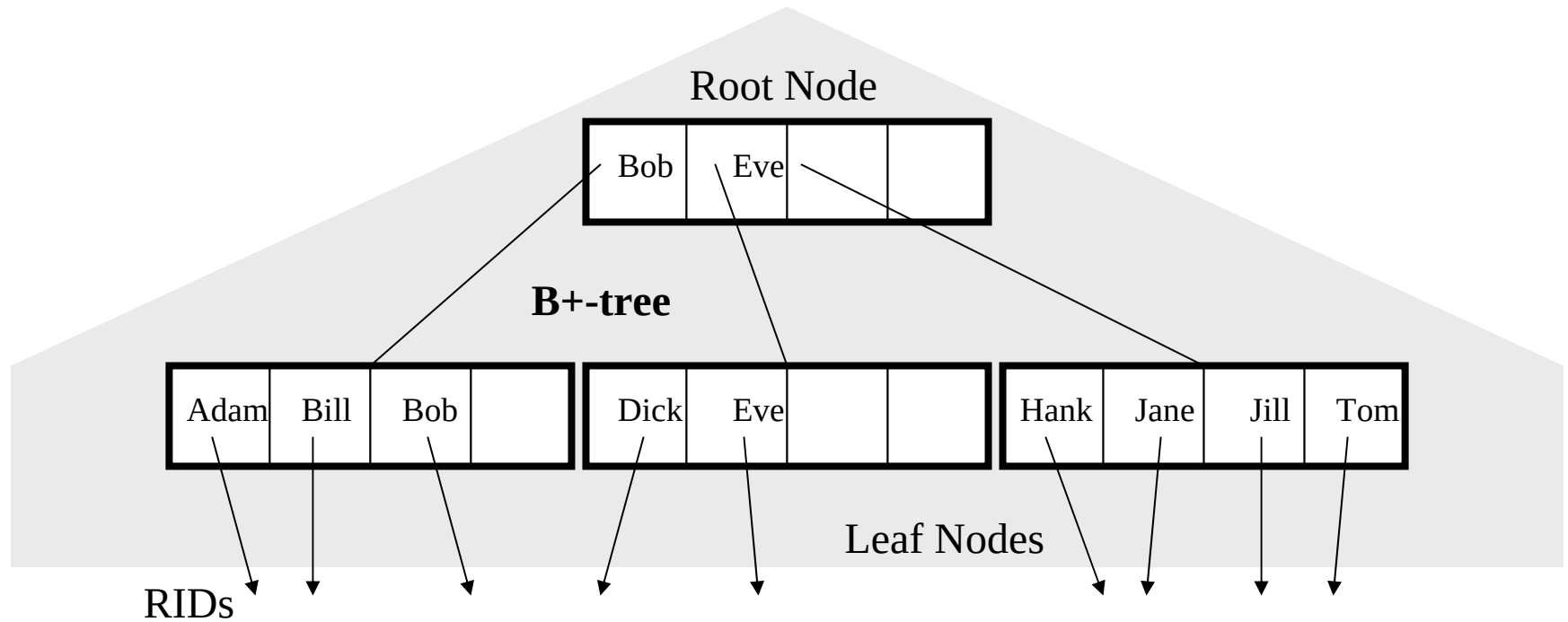
London, Paris, Madrid, Kopenhagen, Lissabon, Zürich, Frankfurt, Wien, Amsterdam, Florenz



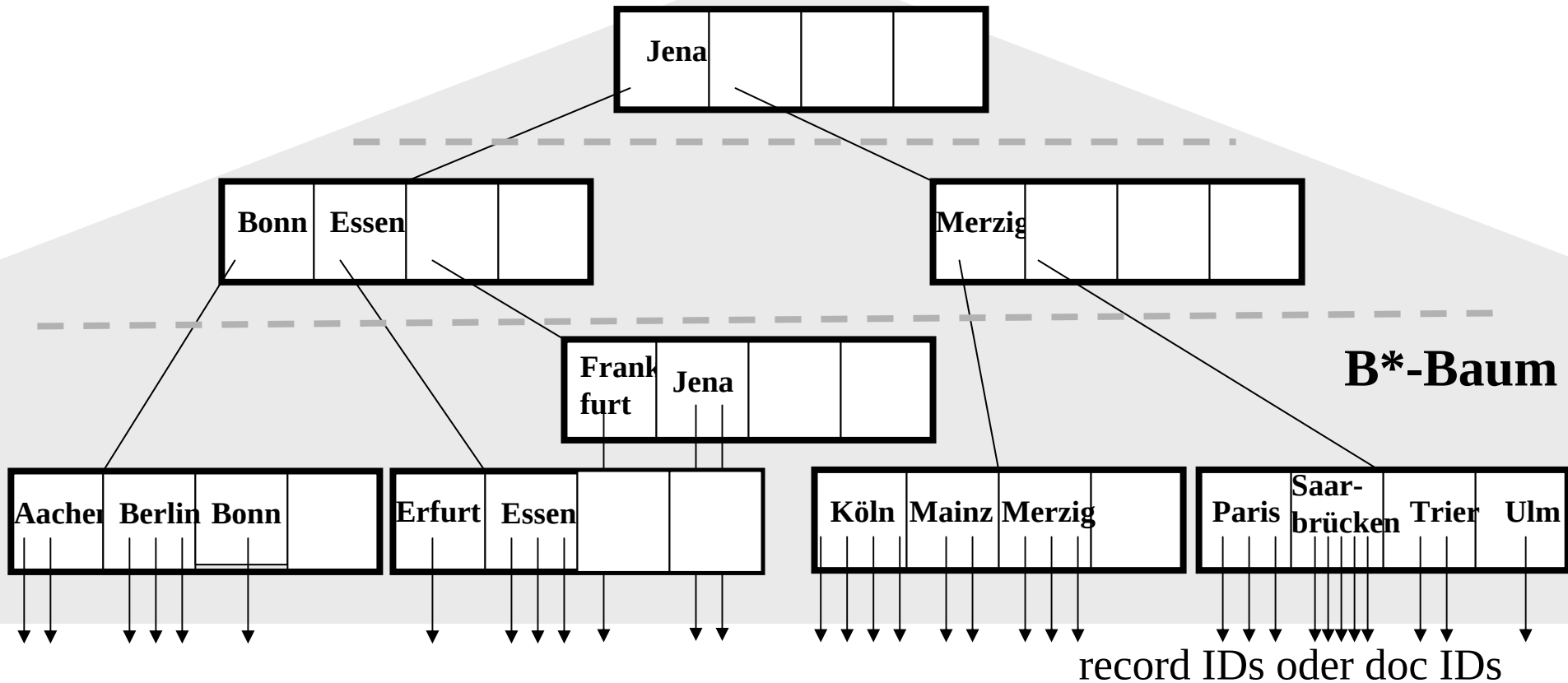
B*-Bäume: Seitenstrukturierte Mehrwegbäume

- Hohler Mehrwegebaum mit hohem Fanout ($\Rightarrow$ kleiner Tiefe)
- Knoten = Seite auf Platte
- Knoteninhalt:
 - (Sohnzeiger, Schlüssel)-Paare in inneren Knoten
 - Schlüssel (mit weiteren Daten) in Blättern
- perfekt balanciert: alle Blätter haben dieselbe Distanz zur Wurzel
- Suche effizienz $O(\log_k n/C)$ Seitenzugriffe (Platten-I/Os)
bei n Schlüsseln, Seitenkapazität C und Fanout k
pro Baumniveau: bestimme kleinsten Schlüssel $\geq q$ und
suche weiter im Teilbaum links von q
- Kosten einer Einfüge- oder Löschoperation $O(\log_k n/C)$
- mittlere Speicherplatzauslastung bei zufälligem Einfügen: $\ln 2 \approx 0.69$

B*-Baum-Beispiel



B*-Baum-Beispiel (2)



B*-Baum-Definition

Ein Mehrwegbaum heißt B*-Baum der Ordnung (m, m^*) , wenn gilt:

- Jeder Nichtblattknoten außer der Wurzel enthält mindestens $m \geq 1$ und höchstens $2m$ Schlüssel (Wegweiser).
- Ein Nichtblattknoten mit k Schlüssel $x_1, \dots, x_k$ hat genau $k+1$ Söhne $t_1, \dots, t_{k+1}$, so daß
 - für alle Schlüssel s im Teilbaum t_i ($2 \leq i \leq k$) gilt $x_{i-1} < s \leq x_i$ und
 - für alle Schlüssel s im Teilbaum t_1 gilt $s \leq x_1$ und
 - für alle Schlüssel im Teilbaum t_{k+1} gilt $x_k < s$.
- Alle Blätter haben dasselbe Niveau (Distanz von der Wurzel)
- Jedes Blatt enthält mindestens $m^* \geq 1$ und höchstens $2m^*$ Schlüssel.

Achtung: Implementierungen verwenden **variabel lange Schlüssel** und eine Knotenkapazität in Bytes statt Konstanten $2m$ und $2m^*$

Sonderfall $m=m^*=1$: **2-3-Bäume** als Hauptspeicherdatenstruktur

Pseudocode für B*-Baum-Suche

Suchen von Schlüssel s in B*-Baum mit Wurzel t :

t habe k Schlüssel $x_1, \dots, x_k$ und $k+1$ Söhne $t_1, \dots, t_{(k+1)}$

(letzteres sofern t kein Blatt ist)

Bestimme den kleinsten Schlüssel x_i , so daß $s \leq x_i$

if $s = x_i$ (für ein $i \leq k$) und t ist ein Blatt

then Schlüssel gefunden

else

if t ist kein Blatt then

if $s \leq x_i$ (für ein $i \leq k$)

then suche s im Teilbaum t_i

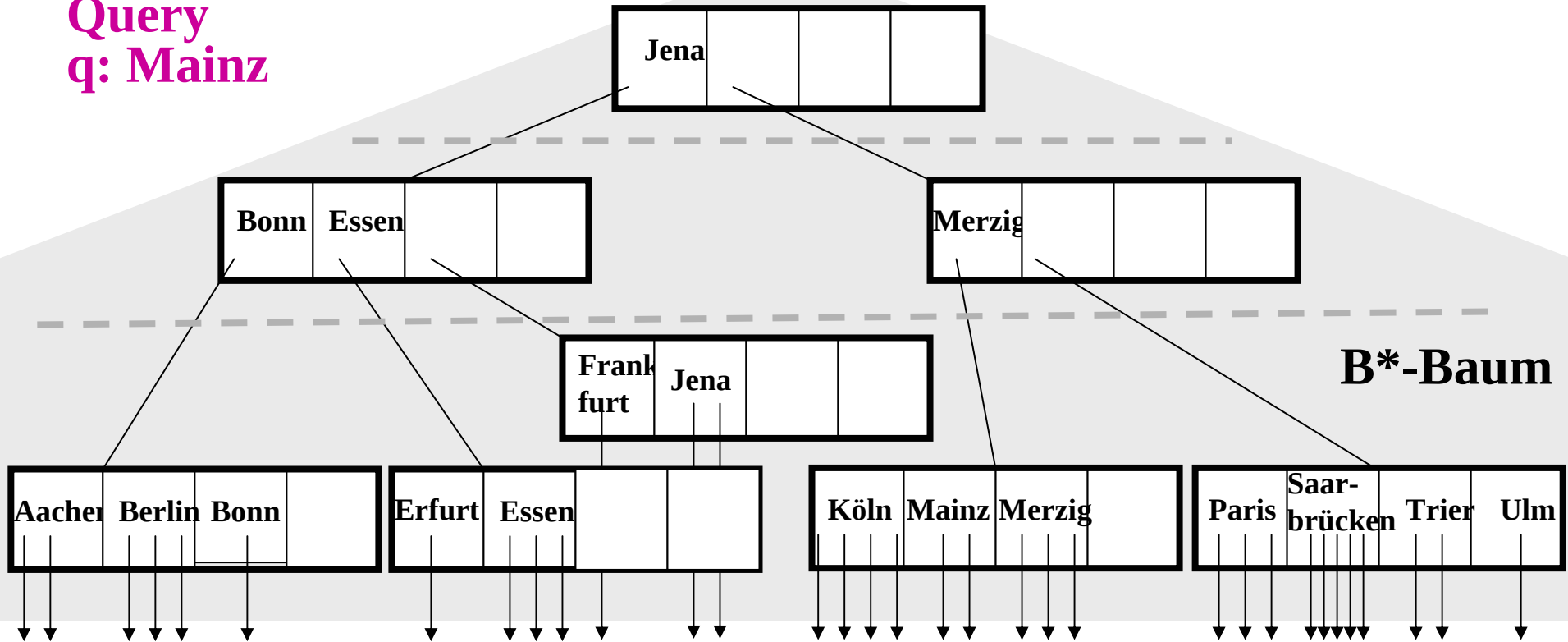
else suche s im Teilbaum $t_{(k+1)}$ fi

else Schlüssel s ist nicht vorhanden fi

fi

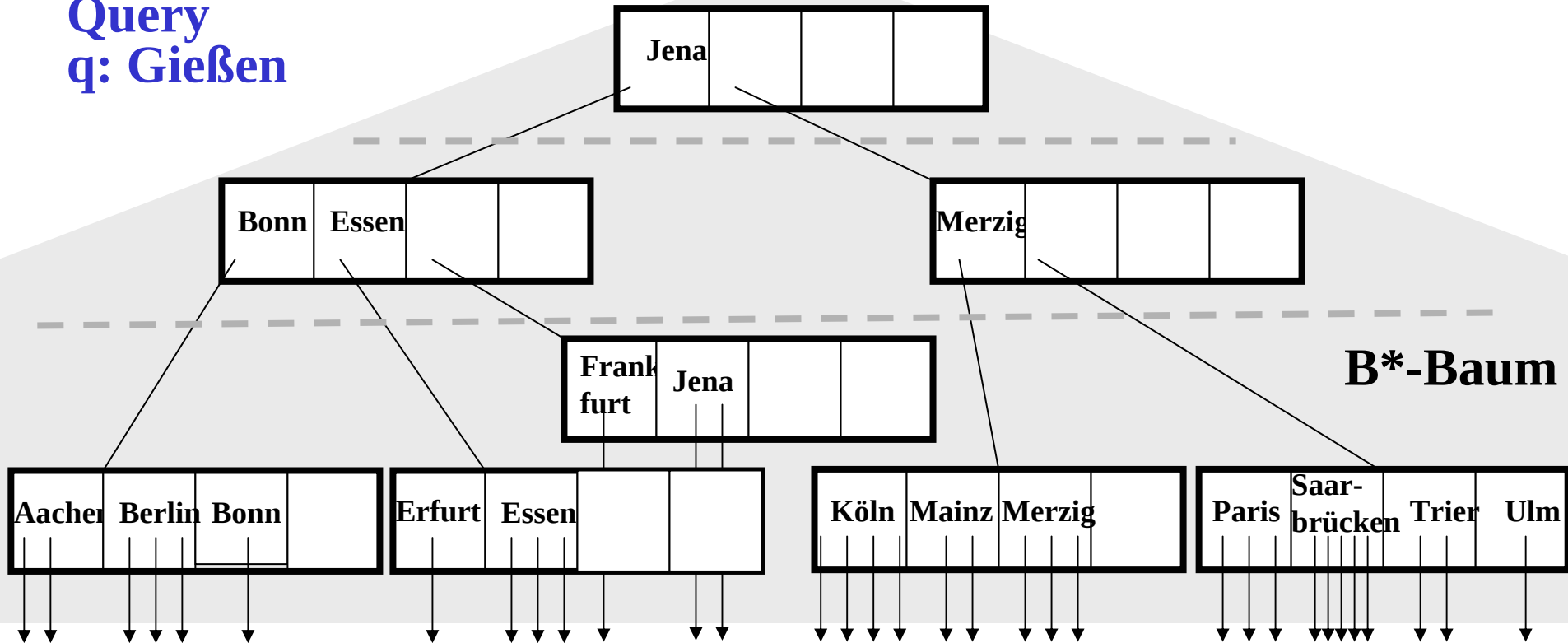
B*-Baum-Suche (1)

Query q: Mainz



B*-Baum-Suche (2)

Query
q: Gießen



Pseudocode für Einfügen in B*-Baum (Grow&Post)

Suche nach einzufügendem Schlüssel e

if e ist noch nicht vorhanden then

Sei t das Blatt, bei dem die Suche erfolglos geendet hat

repeat

if t hat weniger als $2m^*$ bzw. $2m$ Schlüssel (d.h. ist nicht voll)

then füge e in t ein

else /* *Knoten-Split* */

Bestimme Median s der $2m^* + 1$ bzw. $2m + 1$ Schlüssel inkl. e

Erzeuge Bruderknoten t' /* *Grow-Phase* */

if t ist Blattknoten then

Speichere Schlüssel $\leq s$ in t und Schlüssel $> s$ in t'

else Speichere Schlüssel $< s$ (mit Sohnzeigern) in t und

Schlüssel $> s$ (mit Sohnzeigern) in t' fi

if t ist Wurzel /* *Post-Phase* */

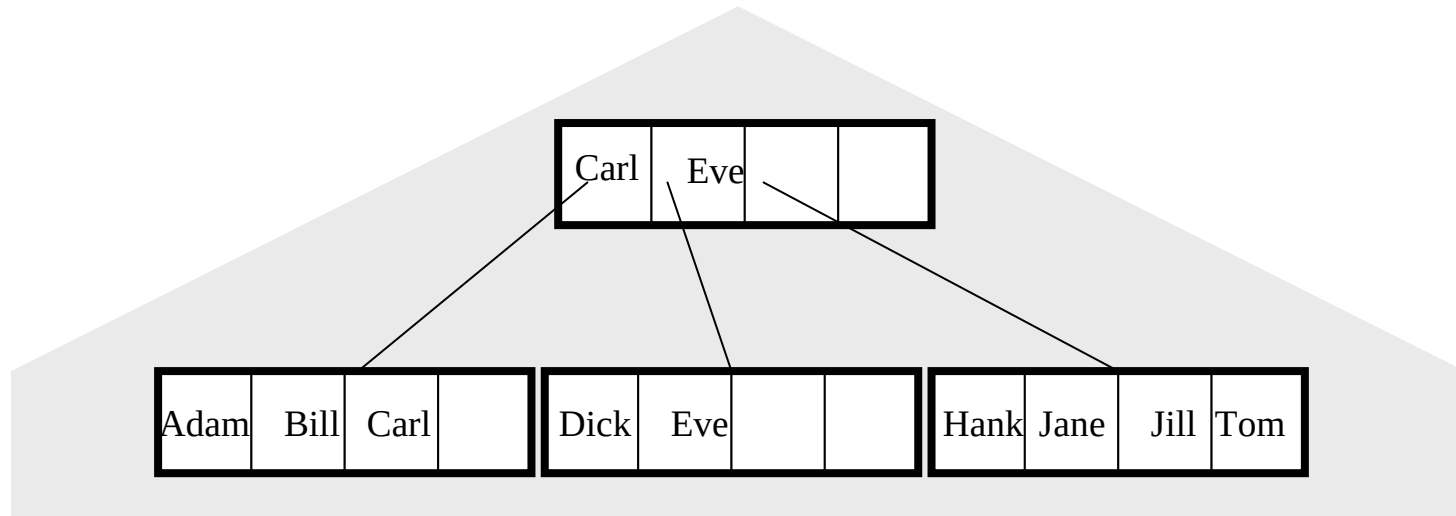
then Erzeuge neue Wurzel r mit Schlüssel s und Zeigern auf t und t'

else Betrachte Vater von t als neues t und s (mit Zeiger auf t') als e fi

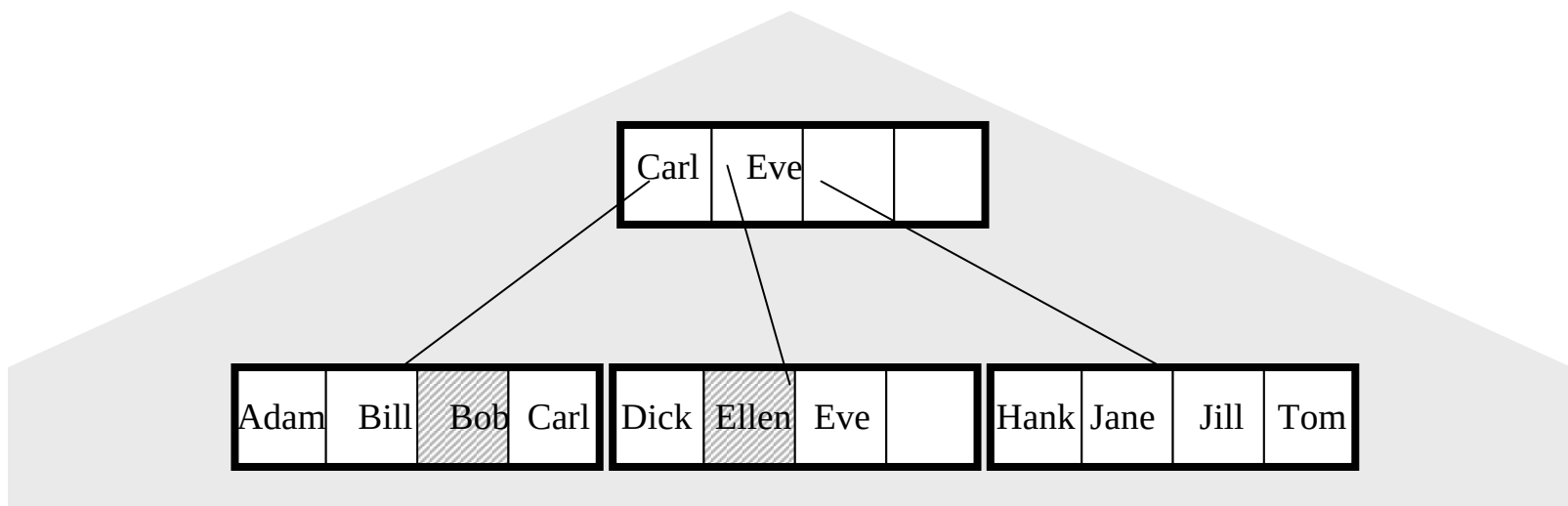
fi

until kein Knoten-Split mehr erfolgt

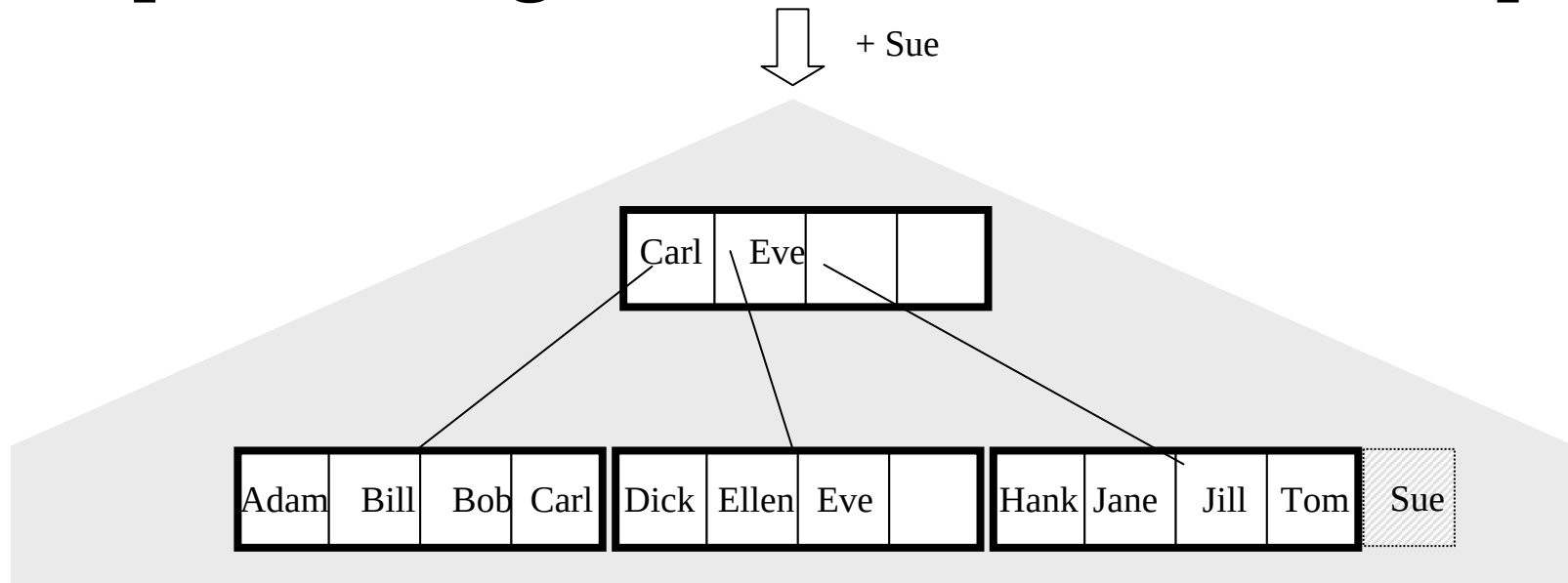
Beispiel: Einfügen in B*-Baum



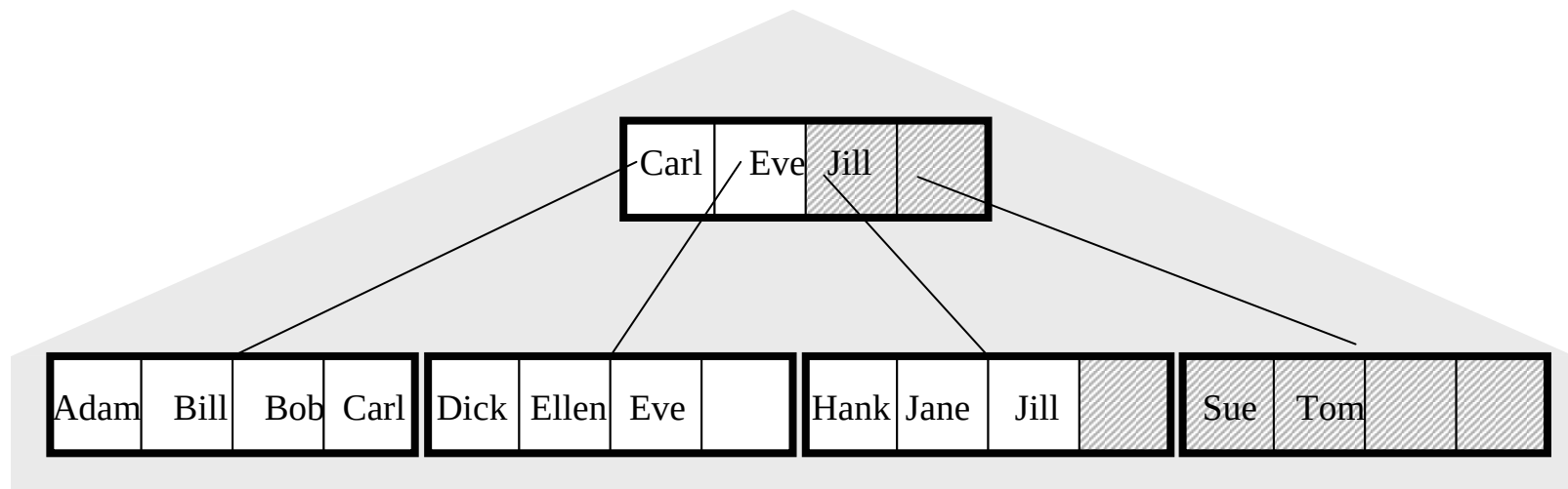
↓ + Ellen, + Bob



Beispiel: Einfügen in B*-Baum mit Blatt-Split

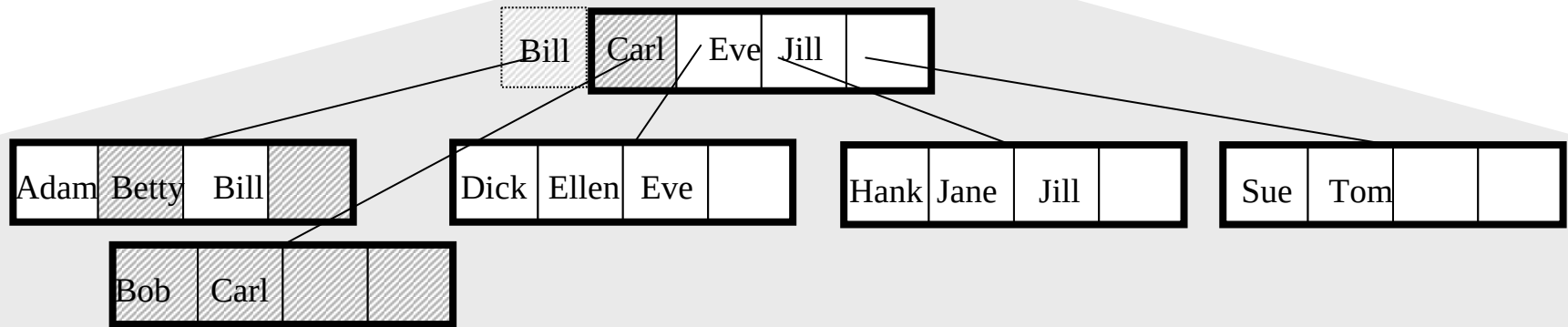


Leaf Node Split ↓

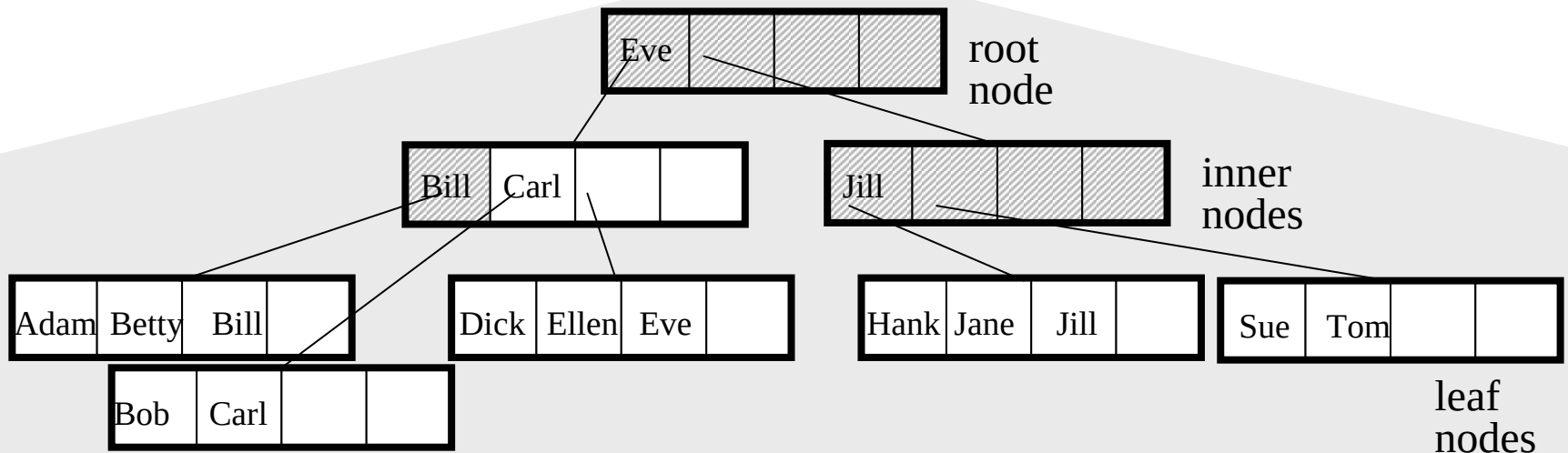


Beispiel: Einfügen in B*-Baum mit Wurzel-Split

↓ + Betty



Root Node Split

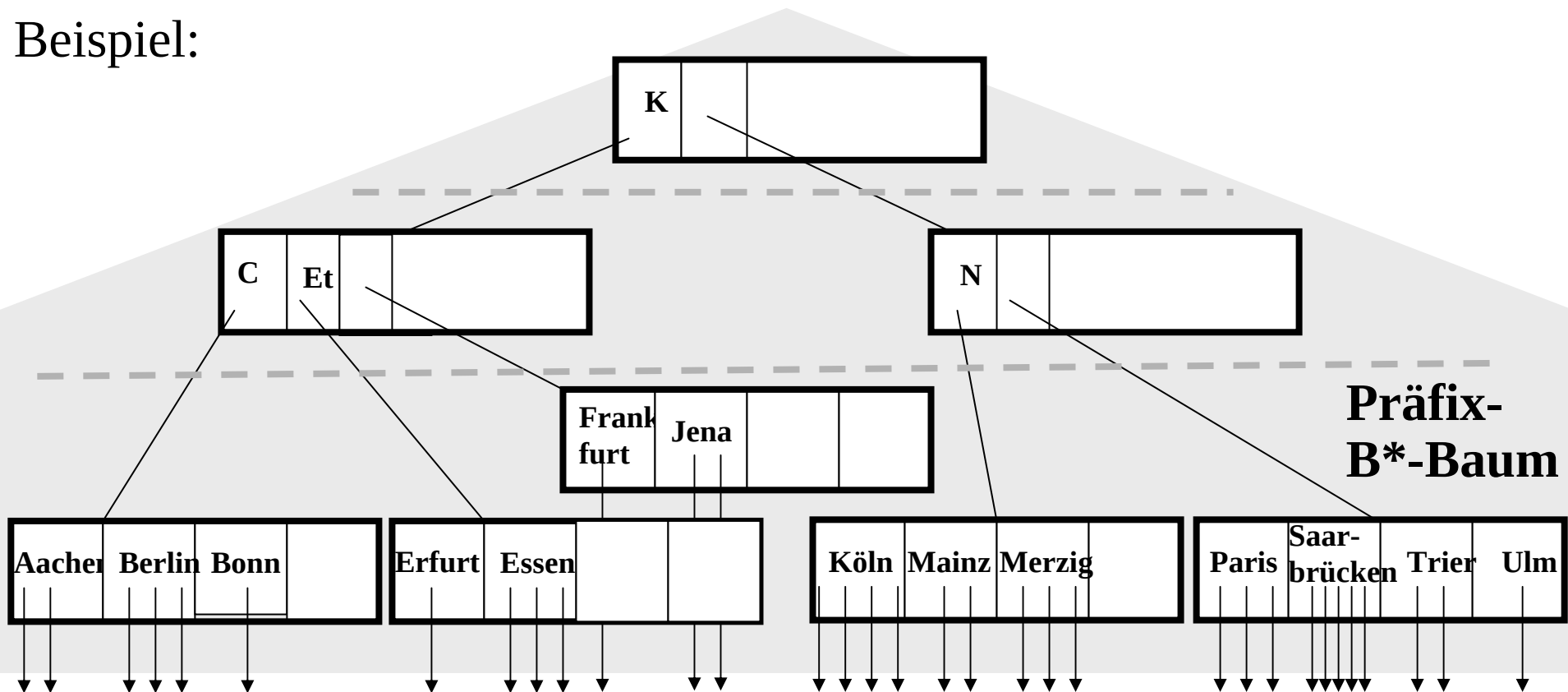


Präfix-B*-Bäume für Strings als Schlüssel

Schlüssel in inneren Knoten sind nur **Wegweiser (Router)** zur Partitionierung des Schlüsselraums.

Statt $x_i = \max\{s: s \text{ ist ein Schlüssel im Teilbaum } t_i\}$ genügt ein (kürzerer) Wegweiser x_i' mit $s_i \leq x_i' < x_{i+1}$ für alle s_i in t_i und alle s_{i+1} in t_{i+1} .
Eine Wahl wäre $x_i' = \text{kürzester String mit der o.a. Eigenschaft.}$
→ höherer Fanout, potentiell kleinere Baumhöhe

Beispiel:



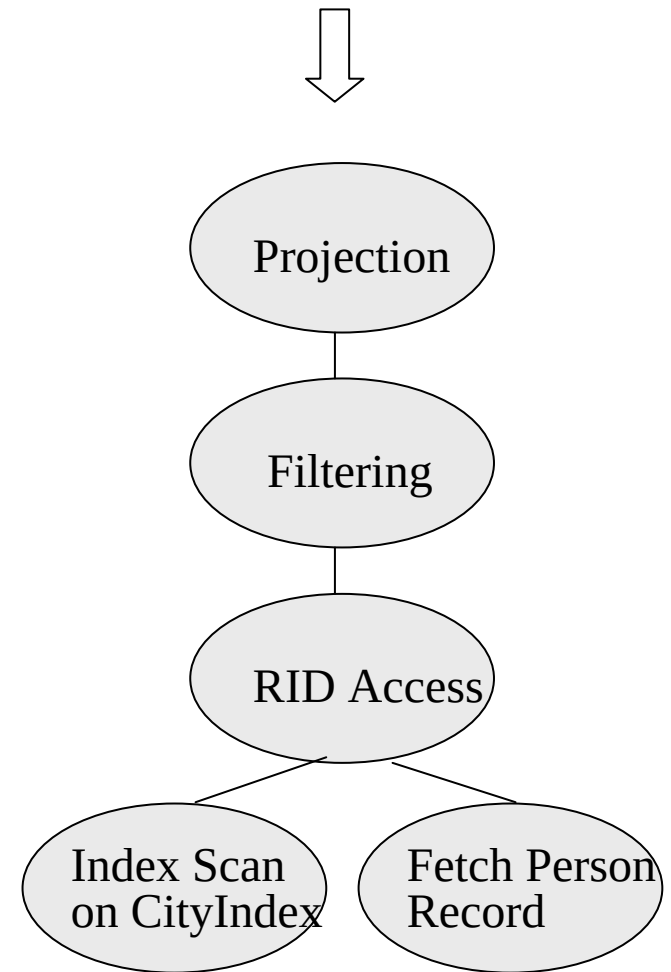
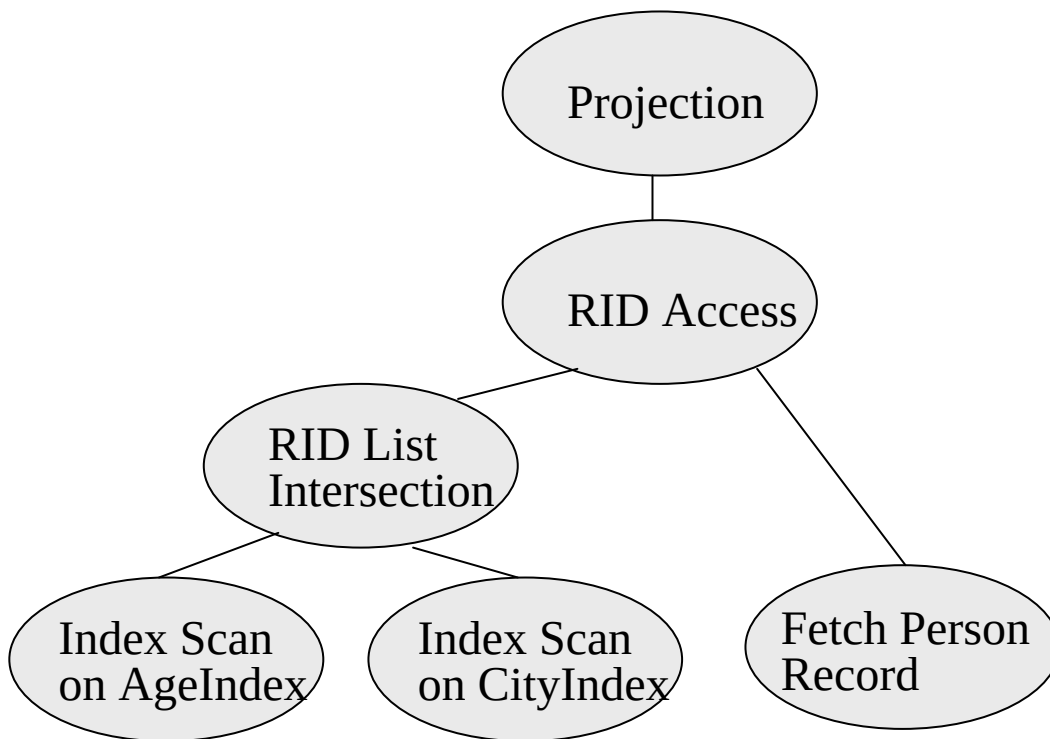
14.3 Wie Anfragen ausgeführt werden

Interne Repräsentation einer Anfrage bzw. eines Ausführungsplans als Operatorbaum mit algebraischen Operatoren der Art:

- Table-Scan
- RID-Zugriff
- Index-Scan
- Sortieren
- Durchschnitt, Vereinigung, Differenz
- Filter (Selektion)
- Projektion für Mengen
- Projekten für Multimengen
- usw.

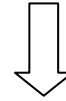
Ausführungspläne (Operatorbäume): Beispiel 1

Select Name, City, Zipcode, Street
From Person
Where Age < 30
And City = "Austin" ↓



Ausführungspläne (Operatorbäume): Beispiel 1

Select Name, City, Zipcode, Street
From Person
Where Age < 30
And City = "Austin"



in Oracle8i:

SELECT STATEMENT
TABLE ACCESS BY ROWID Person
INTERSECT
INDEX RANGE SCAN AgeIndex
INDEX RANGE SCAN CityIndex

SELECT STATEMENT
FILTER
TABLE ACCESS BY ROWID Person
INDEX RANGE SCAN CityIndex

Query-Optimierung

Weitere interne Operatoren als algorithmische Alternativen:

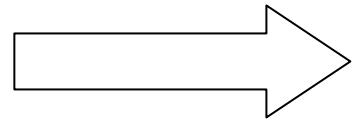
- Nested-Loop-Join
- Merge-Join
- Hash-Join
- Gruppierung mit Aggregation
- Hash-Gruppierung mit Aggregation
- usw.

Der Query-Optimierer

- generiert algebraisch äquivalente Ausführungspläne,
- bewertet deren Ausführungskosten (insbesondere #Plattenzugriffe)
- und wählt den (vermutlich) besten Plan aus.

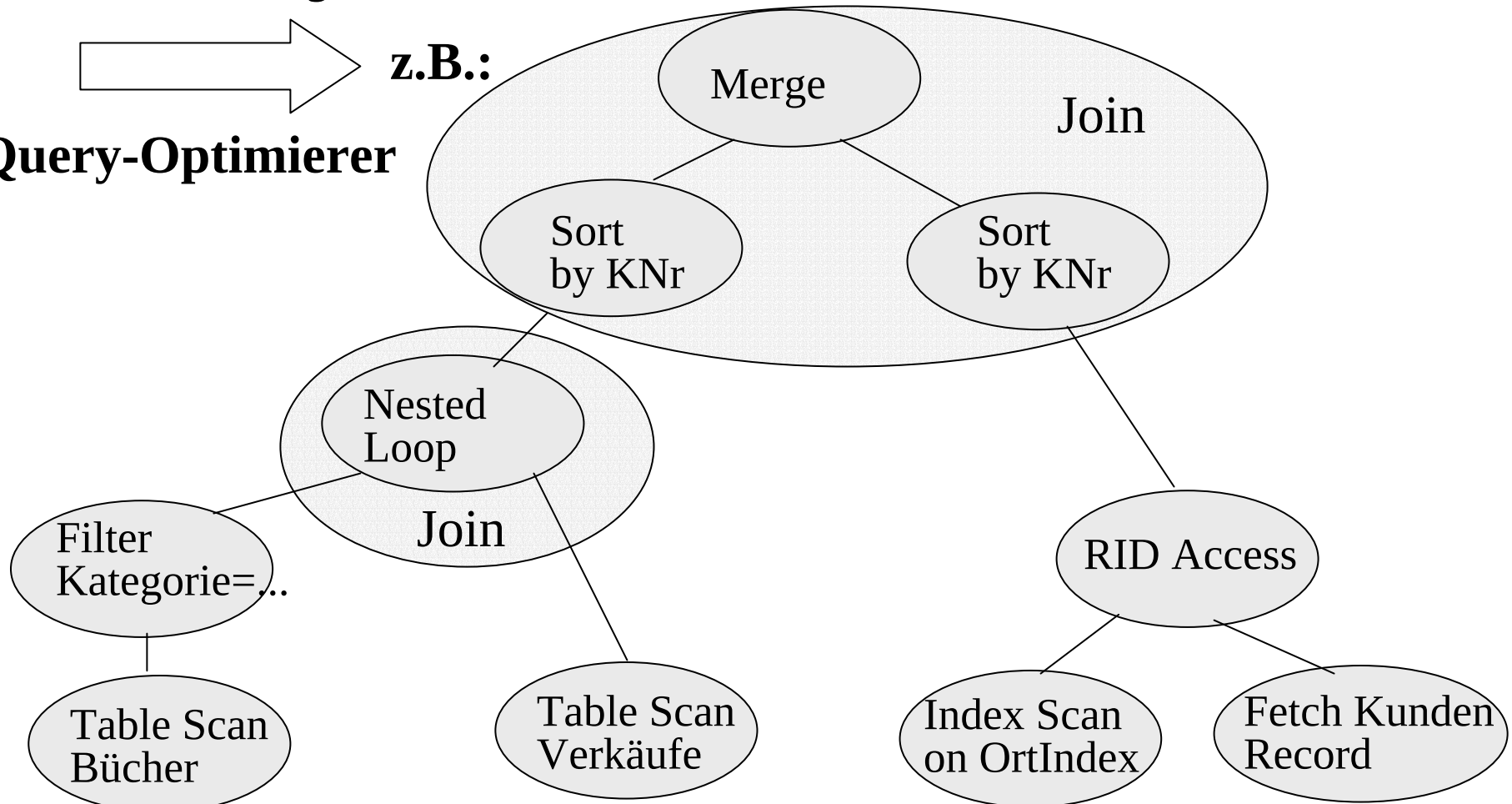
Ausführungspläne (Operatorbäume): Beispiel 2

**SELECT KNr, Name FROM Kunden K, Verkäufe V, Bücher B
WHERE K.Ort = „Saarbrücken“
AND K.KNr = V.KNr AND V.ISBN = B.ISBN
AND B.Kategorie = „Politik“**



z.B.:

Query-Optimierer



Ausführungspläne (Operatorbäume): Beispiel 2

```
SELECT KNr, Name FROM Kunden K, Verkäufe V, Bücher B
WHERE K.Ort = „Saarbrücken“
AND K.KNr = V.KNr AND V.ISBN = B.ISBN
AND B.Kategorie = „Politik“
```

Repräsentation in Oracle8i:

SELECT STATEMENT

MERGE JOIN

SORT

NESTED LOOP

FILTER

TABLE ACCESS FULL

Bücher

TABLE ACCESS FULL

Verkäufe

SORT

TABLE ACCESS BY ROWID

Kunden

INDEX RANGE SCAN

OrtIndex

Kapitel 15: Concurrency Control – Synchronisation von Prozessen und Transaktionen

15.1 Synchronisation nebenläufiger Prozesse

15.2 Synchronisation nebenläufiger Transaktionen

15.2.1 Probleme

15.2.2 Modellbildung

15.2.3 Protokolle

15.1 Synchronisation nebenläufiger Prozesse

Nebenläufigkeit (Quasi-Parallelität, Concurrency):

Verschränkte Ausführung der Schritte mehrerer Prozesse

z.B.: p11, p21, p12, p13, p22 für Prozesse p1, p2

→ Gefährdung der Konsistenz gemeinsamer Datenstrukturen

Lösung:

wechselseitiger Ausschluss (mutual exclusion)

bei kritischen Abschnitten (critical sections)

Beispiel: Programm mit Schritten p1, p2, p3, p4 und
kritischem Abschnitt p2, p3

2 Instantiierungen: Prozesse p, p'

→ p1, p1', p2, p3, p2', p3', p4', p4 ist erlaubt

→ p1, p2, p1', p2', p3, p3', p4, p4' ist nicht erlaubt

Wechselseitiger Ausschluss mit Semaphoren

Semaphor (Ampel):

ganzzahlige, nichtnegative Variable mit unteilbaren Operationen

```
Boolean P (Semaphore x) {  
    if (x == 1) {x = x-1; return True}  
    else return False;}
```

```
void V (Semaphore x) {x = x+1;};
```

Wechselseitiger Ausschluss für kritischen Abschnitt von Programm p:

```
p1; while (P(mutex)!=True) { }; p2; p3; V(mutex); p4;
```

Implementierung von Semaphoren

- 1) unteilbare Maschineninstruktion Test-and-Set <addr> <val>
- 2) ununterbrechbare Prozedur des Betriebssystems-kerns
(z.B. System-Call semop in Unix)
- 3) selbst implementierte leichtgewichtige Prozedur

jeweils mit

- a) busy wait oder
- b) lazy wait

Latches: leichtgewichtige Semaphore

```
struct latch {
    int Status;
    int NumProcesses;
    int Pids[MaxProcesses]; };

int P (latch) {
    while ( Test-and-Set(latch.Status, 0) != True ) { };
    latch.Pids[latch.NumProcesses] = getOwnProcessId();
    latch.NumProcesses++;
    if (latch.NumProcesses == 1) {latch.Status = 1}
    else {latch.Status = 1; sleep ( )};
    return True; };

int V (latch) {
    while ( Test-and-Set(latch.Status, 0) != True) { };
    for (i=0; i < latch.NumProcesses; i++)
        {latch.Pids[i] = latch.Pids[i+1]};
    latch.NumProcesses--;
    if (latch.NumProcesses == 0) {latch.Status = 1}
    else {latch.Status = 1, wakeup (Pids[0])};
    return True;};
```

15.2 Synchronisation nebenläufiger Transaktionen

Eine Transaktion ist eine Folge von Server-Aktionen mit ACID-Eigenschaften:

- Atomarität (atomicity): Alles oder Nichts
- Dauerhaftigkeit/Persistenz (durability)
- Isolation (isolation)
- Integritätserhaltung (consistency preservation)

ACID-Vertrag zwischen Client und transaktionalem Server:

- Alle Aktionen vom Beginn einer Transaktion bis zum Commit Work bilden eine ACID-Einheit (Alles-Fall bei Atomarität).
- Alle Aktionen vom Beginn bis zum Rollback Work (Abort) sind isoliert und werden rückgängig gemacht (Nichts-Fall bei Atomarität)

Transaktionsprogramm Debit/Credit

```
void main ( ) {  
    EXEC SQL BEGIN DECLARE SECTION  
        int b /*balance*/, a /*accountid*/, amount;  
    EXEC SQL END DECLARE SECTION;  
    /* read user input */  
    scanf (, "%d %d", &a, &amount);  
    /* read account balance */  
    EXEC SQL Select Balance into :b From Account  
        Where Account_Id = :a;  
    /* add amount (positive for debit, negative for credit) */  
    b = b + amount;  
    /* write account balance back into database */  
    EXEC SQL Update Account  
        Set Balance = :b Where Account_Id = :a;  
    EXEC SQL Commit Work;  
}
```

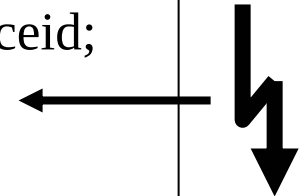
OLTP Example 1: Concurrent Executions

P1	Time	P2
Select Balance Into :b1	1	
From Account		
Where Account_Id = :a		
		Select Balance Into :b2
	2	From Account
		Where Account_Id = :a
b1 = b1-50		
	3	
		b2 = b2 + 100
	4	
Update Account		
Set Balance = :b1	5	
Where Account_Id = :a		
		Update Account
	6	Set Balance = :b2
		Where Account_Id = :a

*Observation: concurrency or parallelism may cause inconsistencies
requires concurrency control for „isolation“*

Transaktionsprogramm Überweisung

```
void main ( ) {  
    /* read user input */  
    scanf („%d %d %d“, &sourceid, &targetid, &amount);  
    /* subtract amount from source account */  
    EXEC SQL Update Account  
        Set Balance = Balance - :amount Where Account_Id = :sourceid;  
    /* add amount to target account */  
    EXEC SQL Update Account  
        Set Balance = Balance + :amount Where Account_Id = :targetid;  
    EXEC SQL Commit Work; }
```



*Observation: failures may cause inconsistencies
require recovery for „atomicity“ and „durability“*

Problemtyp „Verlorene Änderung“

P1	Time	P2
r (x) x := x+100 w (x)	/* x = 100 */ 1 2 4 5 /* x = 200 */ 6 /* x = 300 */	r (x) x := x+200 w (x)



„lost update“

Kern des Problems: R1(x) R2(x) W1(x) W2(x)

Problemtyp „Inkonsistentes Lesen“

P1	Time	P2
	1	r (x)
	2	x := x - 10
	3	w (x)
sum := 0	4	
r (x)	5	
r (y)	6	
sum := sum + x	7	
sum := sum + y	8	
	9	r (y)
	10	y := y + 10
	11	w (y)

↑
„inconsistent read“

Kern des Problems: $R2(x)$ $W2(x)$ $R1(x)$ $R1(y)$ $R2(y)$ $W2(y)$

Problemtyp „Dominoeffekt“

P1	Time	P2
r (x)	1	
x := x + 100	2	
w (x)	3	
	4	r (x)
	5	x := x - 100
failure & rollback	6	
	7	w (x)



„dirty read“

Kern des Problems: $R1(x)$ $W1(x)$ $R2(x)$ $A1$ $W2(x)$... $[C2]$

Warum Semaphore nicht genügen

- ein Semaphor für ganze DB führt zur Zwangssequentialisierung
- jeweils ein Semaphor pro Tupel ist (noch) keine Lösung:
 - unklar, wann V-Operation möglich ist
 - unklar, wie Insert-/Delete-Operationen und prädiktorientierte Operationen zu behandeln sind
 - Umgang mit Deadlocks unklar
 - unakzeptabler Overhead bei vielen Semaphoren

Prozess 1: FundsTransfer von a nach b *Prozess 2: Prüfen der Summe von a und b*
shared semaphore mutex_a; shared semaphore mutex_b;

P(mutex_a);
Update Konto Set Stand = Stand – 3000
Where KontoNr = a;
V(mutex_a);

P(mutex_a);
Select Stand From Konto
Where KontoNr = a;
V(mutex_a);
P(mutex_b); ...; V(mutex_b);

P(mutex_b); ...; V(mutex_b);

Modellbildung und Korrektheitskriterium

Intuitiv:

nur Abläufe zulassen, die äquivalent zu sequentieller Ausführung sind

Formalisierung:

- Modellbildung für (quasi-) parallele Abläufe → Schedules
- Definition der Äquivalenz → Konfliktäquivalenz
- Definition zulässiger Abläufe → (Konflikt-) Serialisierbarkeit
- Entwicklung beweisbar korrekter Concurrency-Control-Verfahren zur automatischen Synchronisation durch das DBS

Schedules und Konflikte

Ein **Schedule** s ist ein Tripel $(T, A, <)$ mit:

- Transaktionsmenge T
- Aktionsmenge A mit Aktionen der Form $R_i(x)$ oder $W_i(x)$
- Partieller Ordnung $<$ auf der Aktionenmenge A ,
wobei für jedes Paar $\langle a, b \rangle \in A \times A$ gilt:
wenn a und b auf dasselbe Objekt zugreifen und a oder b schreibt,
muß $(a < b) \vee (b < a)$ gelten.

Aktionen $a, b \in A$ in einem Schedule $s = (T, A, <)$ sind in **Konflikt**, wenn beide auf dasselbe Objekt zugreifen und a oder b schreibt und a, b zu verschiedenen Transaktionen gehören.

Schedule $s = (T, A, <)$ heißt **seriell**, wenn für je zwei $T_i, T_j \in T$ gilt: entweder liegen alle Aktionen von T_i bezüglich $<$ vor allen Aktionen von T_j oder umgekehrt.

Konflikt-Äquivalenz und Serialisierbarkeit

Schedules s und s' mit denselben Transaktionen und Aktionen sind **(konflikt-) äquivalent**, wenn sie dieselben Konfliktpaare haben.

Ein Schedule s heißt (konflikt-) **serialisierbar**, wenn er (konflikt-) äquivalent zu einem seriellen Schedule ist.

Sei $s = (T, A, <)$ ein Schedule. Der **Abhängigkeitsgraph** $G(s)$ von s ist ein gerichteter Graph mit

- Knotenmenge T und
- einer Kante von T_i nach T_j , wenn es ein Konfliktpaar $\langle a, b \rangle$ in s gibt, bei dem a zu T_i und b zu T_j gehört.

Satz: s ist serialisierbar g.d.w. $G(s)$ azyklisch ist.

Beispiele

s1: R1 (a) W1 (a) R2 (a) R2 (b) R1 (b) W1 (b)

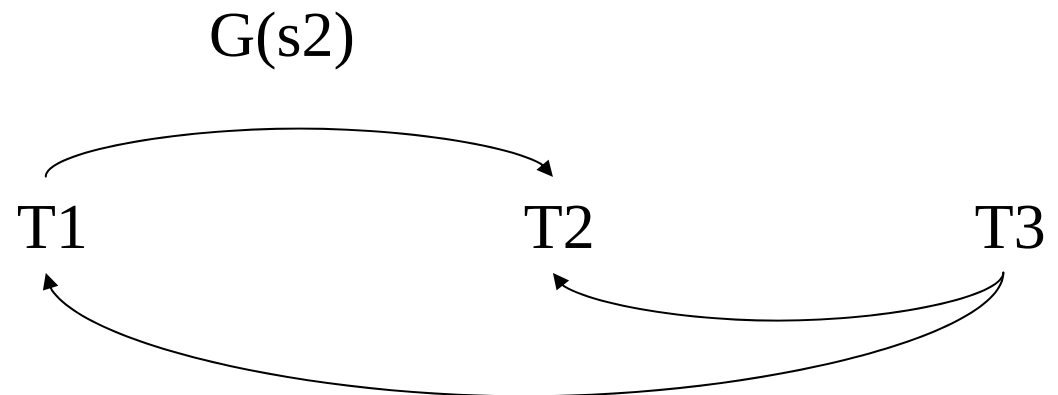
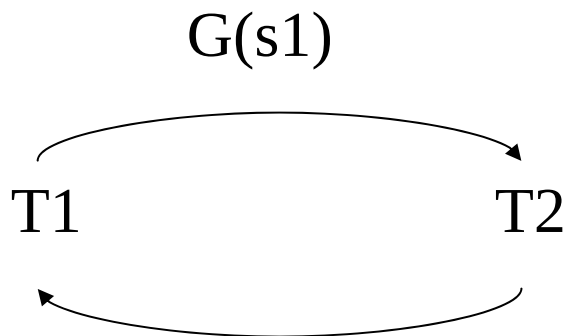
s2: R1 (a) W1 (a) R2 (a) R3 (b) R2 (b) W2 (b) R3 (c) W3 (c) R1 (c)

s2': R3 (b) R3 (c) W3 (c) R1 (a) W1 (a) R1 (c) R2 (a) R2 (b) W2 (b)

Konfliktpaare s1: $\langle W1(a), R2(a) \rangle, \langle R2(b), W1(b) \rangle$

Konfliktpaare s2: $\langle W1(a), R2(a) \rangle, \langle R3(b), W2(b) \rangle, \langle W3(c), R1(c) \rangle$

Konfliktpaare s3: $\langle W3(c), R1(c) \rangle, \langle R3(b), W2(b) \rangle, \langle W1(a), R2(a) \rangle$



Zweiphasen-Sperrprotokoll

DBS-interne Synchronisationsprimitive:

- Lock (Transaktion T_i , Objekt x , Modus m)
- Unlock (Transaktion T_i , Objekt x)

Beispiel:

L1 (a) W1 (a)

L1 (c) W1 (c)

U1 (a) U1 (c)

L2 (b) W2 (b)

L2 (c) - - - - - W2 (c) U2 (b) U2 (c)

Zweiphasen-Sperrprotokoll (Two-Phase Locking, 2PL):

Regel 1: Bevor eine Transaktion auf ein Objekt zugreift, muß das Objekt für die Transaktion gesperrt werden (Wachstumsphase).

Regel 2: Nachdem eine Transaktion eine Sperre freigegeben hat, darf sie keine weiteren Sperren mehr erwerben (Schrumpfungsphase).

Satz: 2PL garantiert Serialisierbarkeit.

Striktes Zweiphasen-Sperrprotokoll

Striktes Zweiphasen-Sperrprotokoll (Strict Two-Phase Locking, S2PL):

Regel 1: wie bei 2PL

Regel 2: Alle jemals erworbenen (exklusiven) Sperren müssen bis zum Commit oder Abort der Transaktion gehalten werden.

Satz: S2PL garantiert Serialisierbarkeit und vermeidet Dominoeffekte.

Sperrprotokolle verwenden in der Regel operationsspezifische Sperrmodi (z.B. Shared-Sperren für Lesen und Exclusive-Sperren für Schreiben) mit einer entsprechenden Sperrmoduskompatibilitätsmatrix:

		Transaktion fordert eine Sperre an im Modus	
		<i>Shared</i>	<i>Exclusive</i>
Objekt ist gesperrt im Modus	<i>Shared</i>	+	-
	<i>Exclusive</i>	-	-

Deadlocks

Zyklische Wartebeziehungen zwischen Transaktionen
→ Zykluserkennung auf Waits-For-Graphen
→ Opferauswahl und Rücksetzen

Beispiele:

X1 (a)	W1 (a)		X1 (b)	- - - - -	
		X2 (b)	W2 (b)	X2 (a)	- - - - -

S1 (a)	R1 (a)		X1 (a)	- - - - -	
		S2 (a)	R2 (a)	X2 (a)	- - - - -

Kapitel 16: Daten-Recovery – Wie Systemausfälle behandelt werden

Fehlerkategorien:

1. Fehler im Anwendungsprogramm
2. Ausfall der Systemsoftware (BS, DBS, usw.): Bohrbugs, Heisenbugs
3. Stromausfall und transiente Hardwarefehler
4. Plattenfehler
5. Katastrophen

Behandlung durch das DBS:

- 1 → Rollback
- 2, 3 → Crash Recovery (basierend auf Logging)
- 4 → Media Recovery (basierend auf Backup und Logging)
- 5 → Remote Backup/Log, Remote Replication

Goal of Crash Recovery

Failure-resilience:

- **redo** recovery for committed transactions
- **undo** recovery for uncommitted transactions

Failure model:

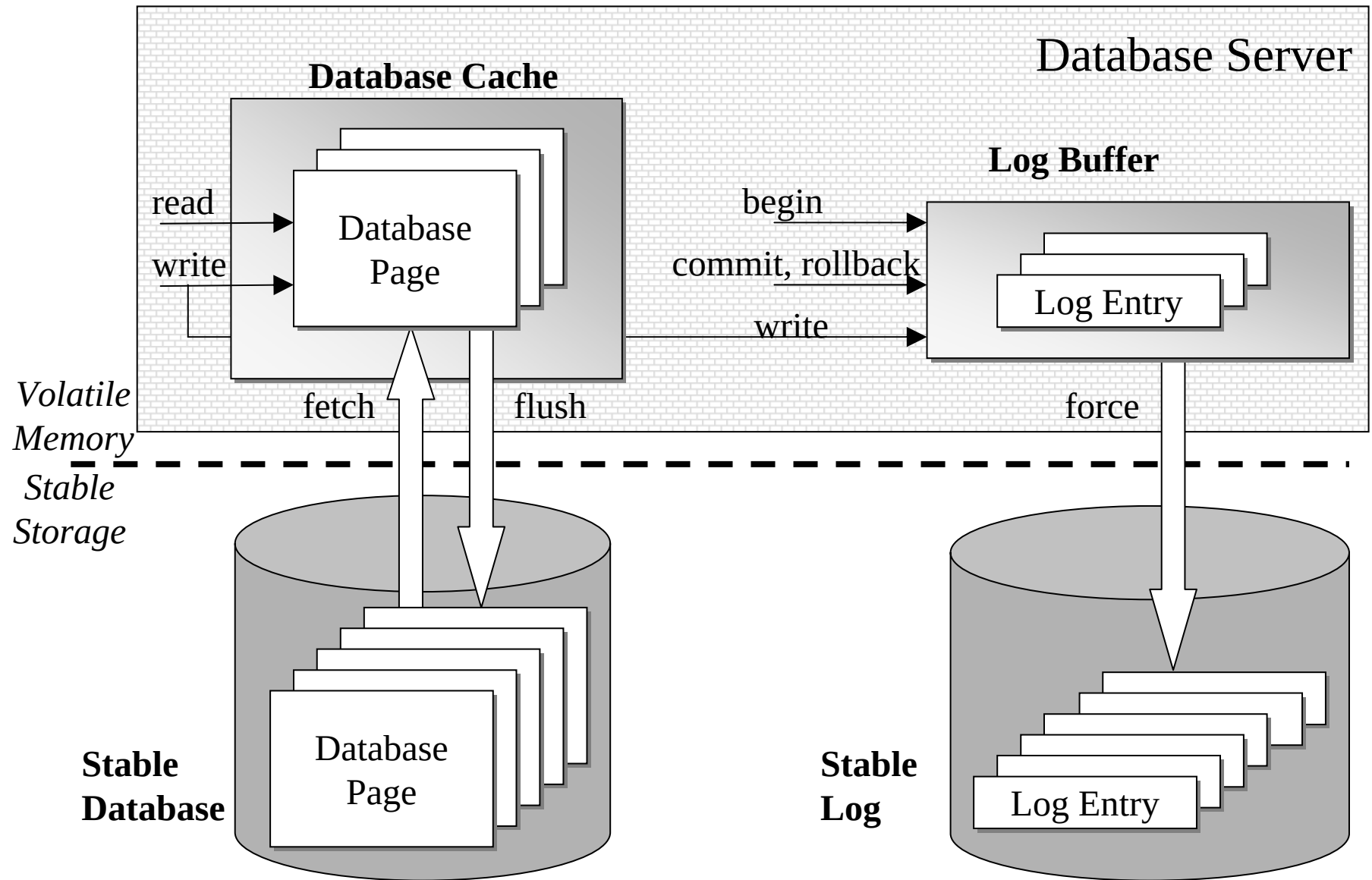
- soft (no damage to secondary storage)
- fail-stop (no unbounded failure propagation)

captures most (server) software failures,
both Bohrbugs and Heisenbugs

Requirements:

- fast restart for high availability ($= \text{MTTF} / (\text{MTTF} + \text{MTTR})$)
- low overhead during normal operation
- simplicity, testability, very high confidence in correctness

Overview of System Architecture



Overview of Simple Three-Pass Algorithm

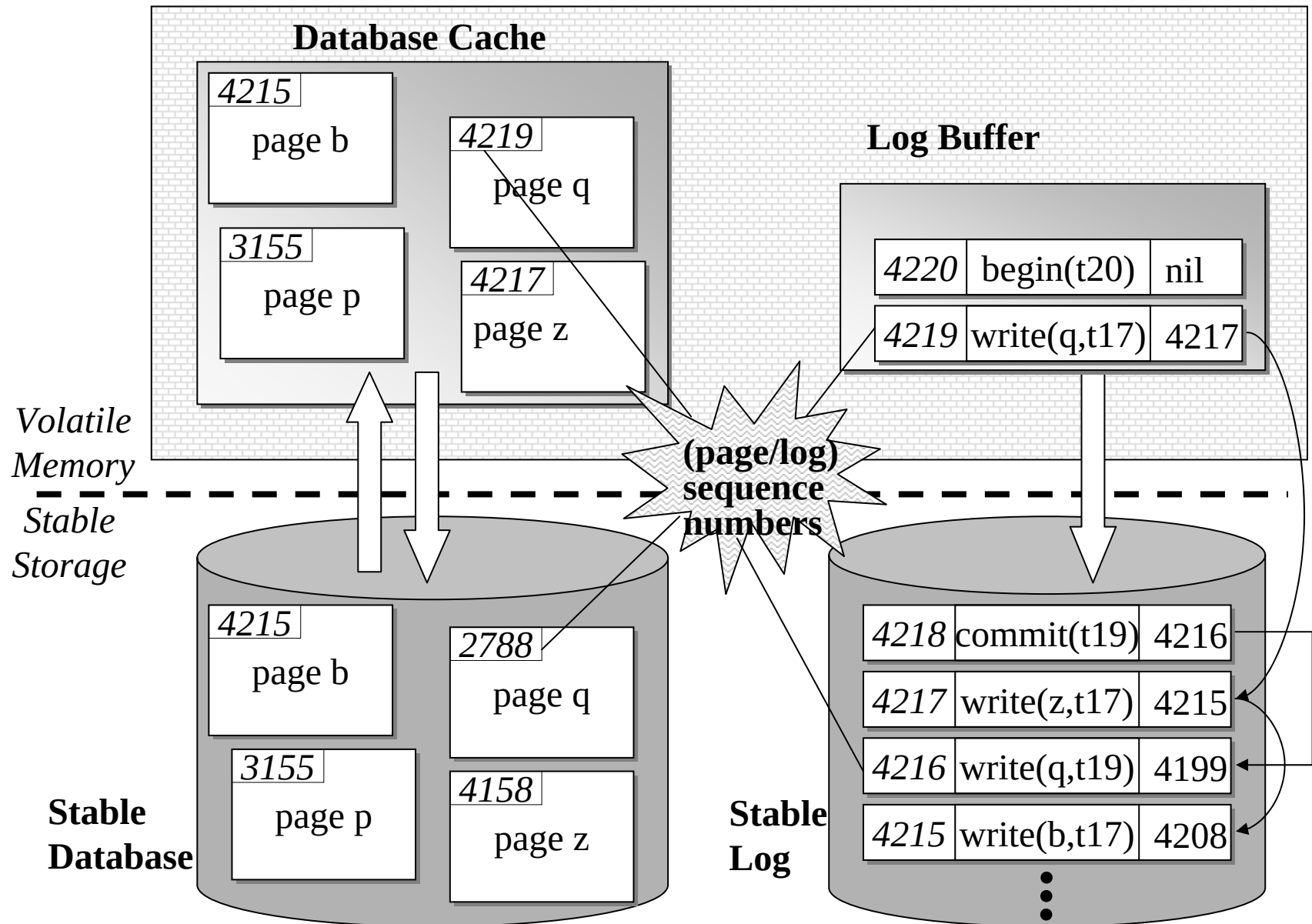
- **Analysis pass:**
 - determine start of stable log from master record
 - perform forward scan
 - to determine winner and loser transactions
- **Redo pass:**
 - perform forward scan
 - to redo all winner actions in chronological (LSN) order (until end of log is reached)
- **Undo pass:**
 - perform backward scan
 - to traverse all loser log entries in reverse chronological order and undo the corresponding actions

Incorporating General Writes As Physiological Log Entries

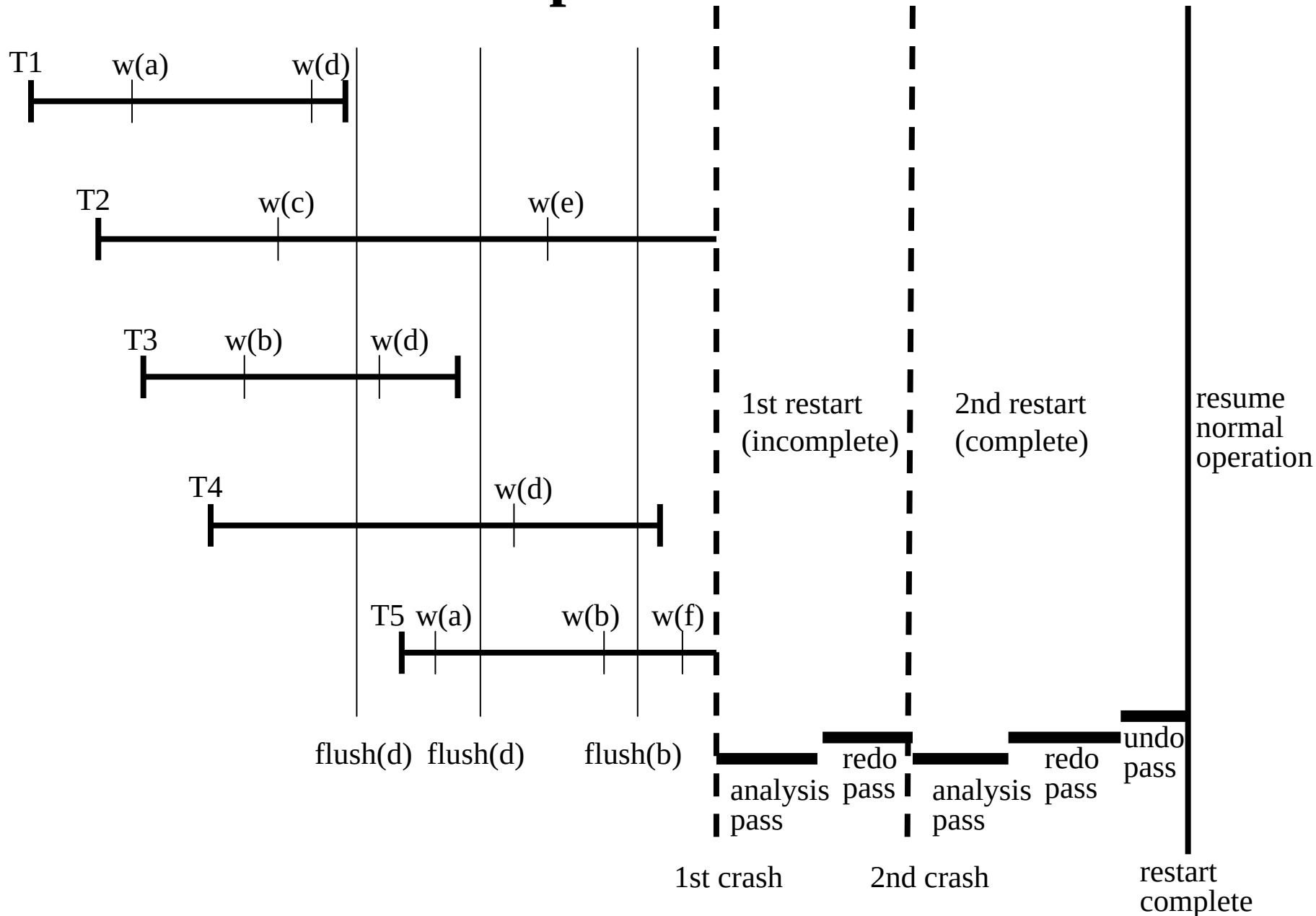
Principle:

- state testing during the redo pass:
for log entry for page p with log sequence number i ,
redo write only if $i > p.\text{PageSeqNo}$
and subsequently set $p.\text{PageSeqNo} := i$
- state testing during the undo pass:
for log entry for page p with log sequence number i ,
undo write only if $i \leq p.\text{PageSeqNo}$
and subsequently set $p.\text{PageSeqNo} := i-1$

Usage of (Log) Sequence Numbers



Example Scenario



Example under Simple Three-Pass Algorithm with General Writes

Sequence number: action	Change of cached database [PageNo: SeqNo]	Change of stable database [PageNo: SeqNo]	Log entry added to log buffer [LogSeqNo: action]	Log entries added to stable log [LogSeqNo's]
1: begin(T1)			1: begin(T1)	
2: begin(T2)			2: begin(T2)	
3: write(a,T1)	a: 3		3: write(a,T1)	
4: begin(T3)			4: begin(T3)	
5: begin(T4)			5: begin(T4)	
6: write(b,T3)	b: 6		6: write(b,T3)	
7: write(c,T2)	c: 7		7: write(c,T2)	
8: write(d,T1)	d: 8		8: write(d,T1)	
9: commit(T1)			9: commit(T1)	1,2,3,4,5,6,7,8,9
10: flush(d)		d: 8		
11: write(d,T3)	d: 11		11: write(d,T3)	
12: begin(T5)			12: begin(T5)	
13: write(a,T5)	a: 13		13: write(a,T5)	
14: commit(T3)			14: commit(T3)	11,12,13,14
15: flush(d)		d: 11		
16: write(d,T4)	d: 16		16: write(d,T4)	
17: write(e,T2)	e: 17		17: write(e,T2)	
18: write(b,T5)	b: 18		18: write(b,T5)	
19: flush(b)		b: 18		16,17,18
20: commit(T4)			20: commit(T4)	20
21: write(f,T5)	f: 21		21: write(f,T5)	
system crash				

restart				
analysis pass: losers = {T2,T5}				
redo(3)	a: 3			
consider-redo(6)	b: 18			
flush (a)		a: 3		
consider-redo(8)	d: 11			
consider-redo(11)	d: 11			
second system crash				
second restart				
analysis pass: losers = {T2,T5}				
consider-redo(3)	a:3			
consider-redo(6)	b: 18			
consider-redo(8)	d: 11			
consider-redo(11)	d: 11			
redo(16)	d: 16			
undo(18)	b: 17			
consider-undo(17)	e: 0			
consider-undo(13)	a: 3			
consider-undo(7)	c: 0			
second restart complete: resume normal operation				

Need and Opportunity for Log Truncation

Major cost factors and potential availability bottlenecks:

- 1) analysis pass and redo pass scan entire log
- 2) redo pass performs many random I/Os on stable database

Improvement:

continuously advance the log start pointer (garbage collection)

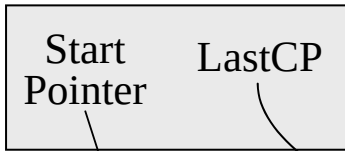
- for redo, can drop all log entries for page p that precede the last flush action for $p =: \text{RedoLSN}(p)$;
 $\min\{\text{RedoLSN}(p) \mid \text{dirty page } p\} =: \text{SystemRedoLSN}$
- for undo, can drop all log entries that precede the oldest log entry of a potential loser $=: \text{OldestUndoLSN}$

Remarks:

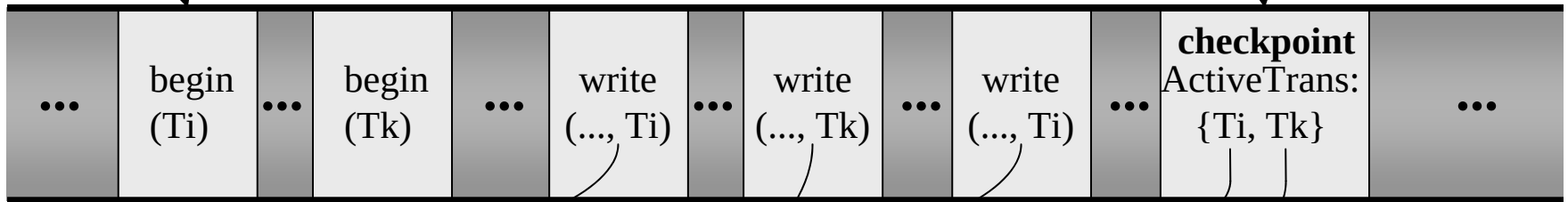
*for full-writes, all but the most recent after-image can be dropped
log truncation after complete undo pass requires global flush*

Heavy-Weight Checkpoints

master record



stable
log



LastSeqNo's

analysis pass

redo pass

undo pass

Dirty Page List for Redo Optimization

Keep track of

- the set of dirty cached pages
- for each such page the sequence number of the oldest write action that followed the most recent flush action (redo sequence numbers)

Avoid very old RedoSeqNo's by write-behind demon

```
type DirtyPageListEntry: record of
    PageNo: identifier;
    RedoSeqNo: identifier;
end;
var DirtyPages:
    set of DirtyPageListEntry indexed by PageNo;
```

Record dirty page list in checkpoint log entry and reconstruct (conservative approximation of) dirty page list during analysis pass

→ exploit knowledge of dirty page list and redo sequence numbers for I/O optimizations during redo

Light-Weight Checkpoints

master record

Start
Pointer

LastCP

RedoSeqNo's

checkpoint

Active

Dirty

Trans:

Pages:

{Ti, Tk}

{p, q, x}

...

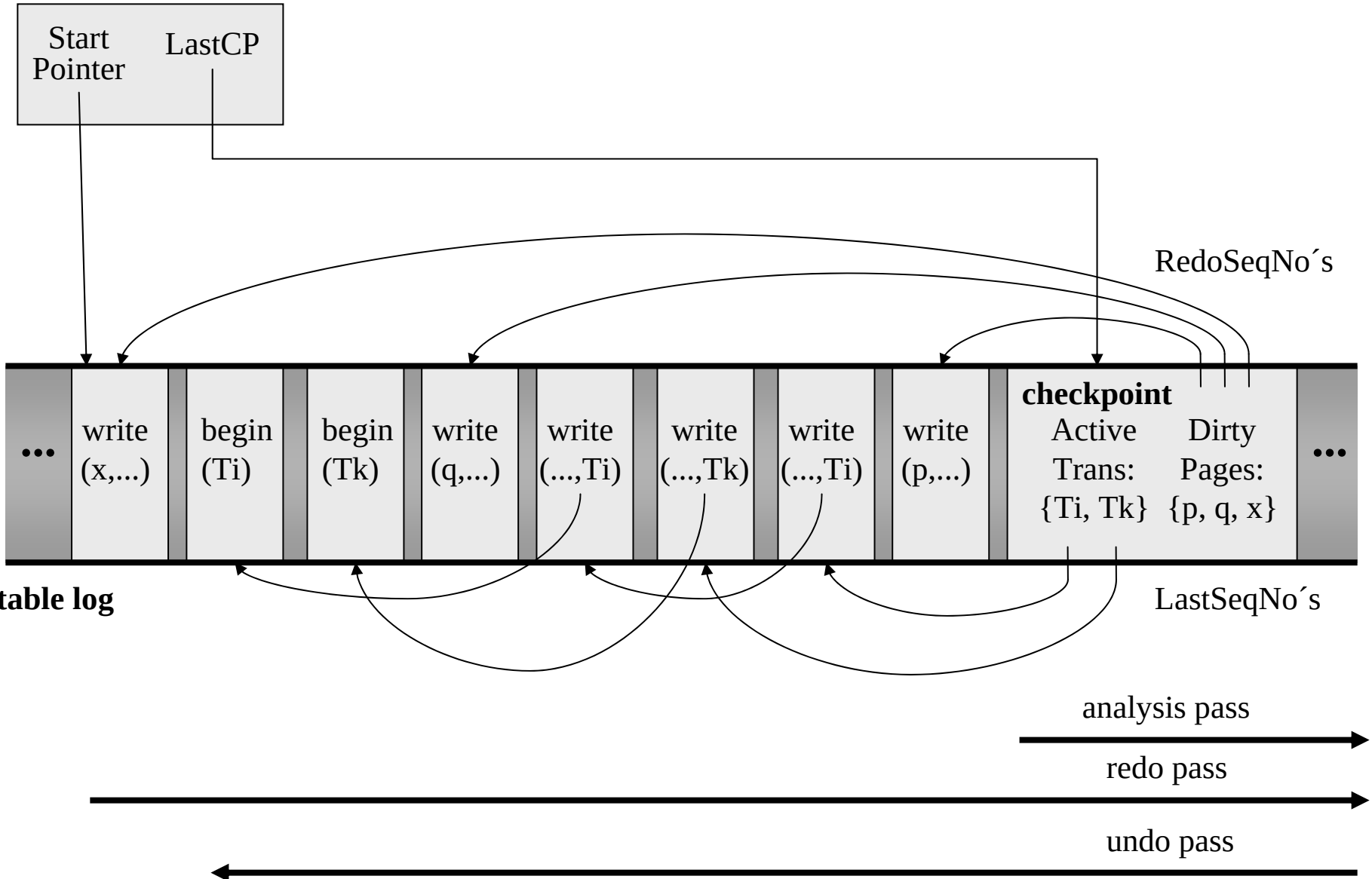
stable log

LastSeqNo's

analysis pass

redo pass

undo pass



Example with Optimizations

Sequence number: action	Change of cached database [PageNo: SeqNo]	Change of stable database [PageNo: SeqNo]	Log entry added to log buffer [LogSeqNo: action]	Log entries added to stable log [LogSeqNo's]
1: begin(T1)			1: begin(T1)	
2: begin(T2)			2: begin(T2)	
3: write(a,T1)	a: 3		3: write(a,T1)	
4: begin(T3)			4: begin(T3)	
5: begin(T4)			5: begin(T4)	
6: write(b,T3)	b: 6		6: write(b,T3)	
7: write (c,T2)	c: 7		7: write(c,T2)	
8: write(d,T1)	d: 8		8: write(d,T1)	
9: commit(T1)			9: commit(T1)	1,2,3,4,5,6,7,8,9
10: flush(d)		d:8	10: flush(d)	
11: write(d,T3)	d: 11		11: write(d,T3)	
12: begin(T5)			12: begin(T5)	
13: write(a,T5)	a: 13		13: write(a,T5)	
14: checkpoint			14: CP DirtyPages: {a,b,c,d} RedoLSNs: a:3, b:6, c:7, d:11 ActiveTrans: {T2,T3,T4,T5}	10,11,12,13,14
15: commit(T3)			15: commit(T3)	15
16: flush(d)		d: 11	16: flush(d)	
17: write(d,T4)	d: 17		17: write(d,T4)	
18: write(e,T2)	e: 18		18: write(e,T2)	
19: write(b,T5)	b: 19		19: write(b,T5)	
20: flush(b)		b: 19	20: flush(b)	16,17,18,19
21: commit(T4)			21: commit(T4)	20,21
22: write(f,T5)	f: 22		22: write(f,T5)	
system crash				

restart				
analysis pass: losers = {T2,T5}				
DirtyPages = {a,c,d,e,f}				
RedoLSNs: a:3, c:7, d:17, e:18				
redo(3)	a:3			
consider-redo(6)	b: 19			
skip-redo(8)				
skip-redo(11)				
redo(17)	d:17			
undo(19)	b: 18			
consider-undo(18)	e: 0			
consider-undo(13)	a: 3			
consider-undo(7)	c: 0			
restart complete: resume normal operation				

Pseudocode: Data Structures (1)

```
type Page: record of
    PageNo: identifier;
    PageSeqNo: identifier;
    Status: (clean, dirty);
    Contents: array [PageSize] of char;
end;
persistent var StableDatabase:
    set of Page indexed by PageNo;
var DatabaseCache:
    set of Page indexed by PageNo;
type LogEntry: record of
    LogSeqNo: identifier;
    TransId: identifier;
    PageNo: identifier;
    ActionType: (write, full-write, begin, commit,
        rollback, compensate, checkpoint, flush);
    ActiveTrans: set of TransInfo;
    DirtyPages: set of DirtyPageInfo;
    UndoInfo: array of char;
    RedoInfo: array of char;
    PreviousSeqNo: identifier;
    NextUndoSeqNo: identifier;
end;
```

Pseudocode: Data Structures (2)

```
persistent var StableLog:
    ordered set of LogEntry indexed by LogSeqNo;
var LogBuffer:
    ordered set of LogEntry indexed by LogSeqNo;
persistent var MasterRecord: record of
    StartPointer: identifier;
    LastCP: identifier;
end;
type TransInfo: record of
    TransId: identifier;
    LastSeqNo: identifier;
end;
var ActiveTrans:
    set of TransInfo indexed by TransId;
type DirtyPageInfo: record of
    PageNo: identifier;
    RedoSeqNo: identifier;
end;
var DirtyPages:
    set of DirtyPageInfo indexed by PageNo;
```

Pseudocode: Actions During Normal Operation (1)

```
write or full-write (pageno, transid, s):  
    DatabaseCache[pageno].Contents := modified contents;  
    DatabaseCache[pageno].PageSeqNo := s;  
    DatabaseCache[pageno].Status := dirty;  
    newlogentry.LogSeqNo := s;  
    newlogentry.ActionType := write or full-write;  
    newlogentry.TransId := transid;  
    newlogentry.PageNo := pageno;  
    newlogentry.UndoInfo := information to undo update;  
    newlogentry.RedoInfo := information to redo update;  
    newlogentry.PreviousSeqNo :=  
        ActiveTrans[transid].LastSeqNo;  
    ActiveTrans[transid].LastSeqNo := s;  
    LogBuffer += newlogentry;  
    if pageno not in DirtyPages then  
        DirtyPages += pageno;  
        DirtyPages[pageno].RedoSeqNo := s;  
    end /*if*/;
```


Pseudocode: Actions During Normal Operation (2)

fetch (pageno):

```
DatabaseCache += pageno;  
DatabaseCache[pageno].Contents :=  
    StableDatabase[pageno].Contents;  
DatabaseCache[pageno].PageSeqNo :=  
    StableDatabase[pageno].PageSeqNo;  
DatabaseCache[pageno].Status := clean;
```

flush (pageno):

```
if there is logentry in LogBuffer  
    with logentry.PageNo = pageno  
then force ( ); end /*if*/;  
StableDatabase[pageno].Contents :=  
    DatabaseCache[pageno].Contents;  
StableDatabase[pageno].PageSeqNo :=  
    DatabaseCache[pageno].PageSeqNo;  
DatabaseCache[pageno].Status := clean;  
newlogentry.LogSeqNo := next sequence number;  
newlogentry.ActionType := flush;  
newlogentry.PageNo := pageno;  
LogBuffer += newlogentry;  
DirtyPages -= pageno;
```

Pseudocode: Actions During Normal Operation (3)

```
force ( ):
    StableLog += LogBuffer;
    LogBuffer := empty;

begin (transid, s):
    ActiveTrans += transid;
    ActiveTrans[transid].LastSeqNo := s;
    newlogentry.LogSeqNo := s;
    newlogentry.ActionType := begin;
    newlogentry.TransId := transid;
    newlogentry.PreviousSeqNo := nil;
    LogBuffer += newlogentry;

commit (transid, s):
    newlogentry.LogSeqNo := s;
    newlogentry.ActionType := commit;
    newlogentry.TransId := transid;
    newlogentry.PreviousSeqNo :=
        ActiveTrans[transid].LastSeqNo;
    LogBuffer += newlogentry;
    ActiveTrans -= transid;
    force ( );
```

Pseudocode: Actions During Normal Operation (4)

```
abort (transid):
    logentry :=
        ActiveTrans[transid].LastSeqNo;
    while logentry is not nil and
        logentry.ActionType = write or full-write
    do
        newlogentry.LogSeqNo := new sequence number;
        newlogentry.ActionType := compensation;
        newlogentry.PreviousSeqNo := ActiveTrans[transid].LastSeqNo;
        newlogentry.RedoInfo :=
            inverse action of the action in logentry;
        newlogentry.NextUndoSeqNo := logentry.PreviousSeqNo;
        ActiveTrans[transid].LastSeqNo := newlogentry.LogSeqNo;
        LogBuffer += newlogentry;
        write (logentry.PageNo) according to logentry.UndoInfo;
        logentry := logentry.PreviousSeqNo;
    end /*while*/
    newlogentry.LogSeqNo := new sequence number;
    newlogentry.ActionType := rollback;
    newlogentry.TransId := transid;
    newlogentry.PreviousSeqNo := ActiveTrans[transid].LastSeqNo;
    newlogentry.NextUndoSeqNo := nil;
    LogBuffer += newlogentry;
    ActiveTrans -= transid;
    force ( );
```

Pseudocode: Actions During Normal Operation (5)

```
log truncation ( ):  
  OldestUndoLSN := min{i|StableLog[i].TransId is in ActiveTrans}  
  SystemRedoLSN := min {DirtyPages[p].RedoSeqNo};  
  OldestRedoPage := page p such that  
    DirtyPages[p].RedoSeqNo = SystemRedoLSN;  
  NewStartPointer := min{OldestUndoLSN, SystemRedoLSN};  
  OldStartPointer := MasterRecord.StartPointer;  
  while OldStartPointer - NewStartPointer is not large enough  
    and SystemRedoLSN < OldestUndoLSN  
  do  
    flush (OldestRedoPage);  
    SystemRedoLSN := min{DatabaseCache[p].RedoLSN};  
    OldestRedoPage := page p such that  
      DatabaseCache[p].RedoLSN = SystemRedoLSN;  
    NewStartPointer := min{OldestUndoLSN, SystemRedoLSN};  
  end /*while*/;  
  MasterRecord.StartPointer := NewStartPointer;  
  
checkpoint ( ):  
  logentry.ActionType := checkpoint;  
  logentry.ActiveTrans := ActiveTrans (as maintained in memory);  
  logentry.DirtyPages := DirtyPages (as maintained in memory);  
  logentry.LogSeqNo := next sequence number to be generated;  
  LogBuffer += logentry;  
  force ( ); MasterRecord.LastCP := logentry.LogSeqNo;
```

Pseudocode: Recovery Procedure (1)

```
restart ( ):  
    analysis pass ( ) returns losers, DirtyPages;  
    redo pass ( );  
    undo pass ( );
```

Pseudocode: Recovery Procedure (2)

analysis pass () returns losers, DirtyPages:

```
var losers: set of record
    TransId: identifier; LastSeqNo: identifier;
end indexed by TransId;
cp := MasterRecord.LastCP;
losers := StableLog[cp].ActiveTrans;
DirtyPages := StableLog[cp].DirtyPages;
max := LogSeqNo of most recent log entry in StableLog;
for i := cp to max do
    case StableLog[i].ActionType:
        begin: losers += StableLog[i].TransId;
            losers[StableLog[i].TransId].LastSeqNo := nil;
        commit: losers -= StableLog[i].TransId;
        full-write:
            losers[StableLog[i].TransId].LastSeqNo := i;
    end /*case*/;
    if StableLog[i].ActionType = write or full-write or compensat
        and StableLog[i].PageNo not in DirtyPages
    then
        DirtyPages += StableLog[i].PageNo;
        DirtyPages[StableLog[i].PageNo].RedoSeqNo := i;
    end /*if*/;
    if StableLog[i].ActionType = flush
    then DirtyPages -= StableLog[i].PageNo; end /*if*/;
end /*for*/;
```

Pseudocode: Recovery Procedure (3)

```
redo pass ( ):
  SystemRedoLSN := min {DirtyPages[p].RedoSeqNo};
  max := LogSeqNo of most recent log entry in StableLog;
  for i := SystemRedoLSN to max do
    if StableLog[i].ActionType =
      write or full-write or compensate
    then
      pageno = StableLog[i].PageNo;
      if pageno in DirtyPages and
        DirtyPages[pageno].RedoSeqNo < i
      then
        fetch (pageno);
        if DatabaseCache[pageno].PageSeqNo < i
        then
          read and write (pageno)
            according to StableLog[i].RedoInfo;
          DatabaseCache[pageno].PageSeqNo := i;
        end /*if*/;
      end /*if*/;
    end /*if*/;
  end /*for*/;
```

Pseudocode: Recovery Procedure (4)

```
undo pass ( ):
  ActiveTrans := empty;
  for each t in losers
  do
    ActiveTrans += t;
    ActiveTrans[t].LastSeqNo := losers[t].LastSeqNo;
  end /*for*/;
  while there exists t in losers
    such that losers[t].LastSeqNo <> nil
  do
    nexttrans := TransNo in losers
      such that losers[nexttrans].LastSeqNo =
        max {losers[x].LastSeqNo | x in losers};
    nextentry := losers[nexttrans].LastSeqNo;

    if StableLog[nextentry].ActionType = compensation
    then
      losers[nexttrans].LastSeqNo :=
        StableLog[nextentry].NextUndoSeqNo;
    end /*if*/;
```


Pseudocode: Recovery Procedure (5)

```
if StableLog[nextentry].ActionType = write or full-write
then
    pageno = StableLog[nextentry].PageNo;
    fetch (pageno);
    if DatabaseCache[pageno].PageSeqNo >= nextentry.LogSeqNo
    then
        newlogentry.LogSeqNo := new sequence number;
        newlogentry.ActionType := compensation;
        newlogentry.PreviousSeqNo :=
            ActiveTrans[transid].LastSeqNo;
        newlogentry.NextUndoSeqNo := nextentry.PreviousSeqNo;
        newlogentry.RedoInfo :=
            inverse action of the action in nextentry;
        ActiveTrans[transid].LastSeqNo := newlogentry.LogSeqNo;
        LogBuffer += newlogentry;
        read and write (StableLog[nextentry].PageNo)
            according to StableLog[nextentry].UndoInfo;
        DatabaseCache[pageno].PageSeqNo := newlogentry.LogSeqNo;
    end /*if*/;
    losers[nextttrans].LastSeqNo =
        StableLog[nextentry].PreviousSeqNo;
end /*if*/;
```

Pseudocode: Recovery Procedure (6)

```
if StableLog[nextentry].ActionType = begin
    then
        newlogentry.LogSeqNo := new sequence number;
        newlogentry.ActionType := rollback;
        newlogentry.TransId := StableLog[nextentry].TransId;
        newlogentry.PreviousSeqNo :=
            ActiveTrans[transid].LastSeqNo;
        LogBuffer += newlogentry;
        ActiveTrans -= transid;
        losers -= transid;
    end /*if*/;

end /*while*/;
force ( );
```

Fundamental Problem of Distributed Commit

Problem:

- Transaction operates on multiple servers (**resource managers**)
- Global commit needs unanimous local commits of all **participants (agents)**
- Distributed system may fail partially (server crashes, network failures) and creates the potential danger of inconsistent decisions

Approach:

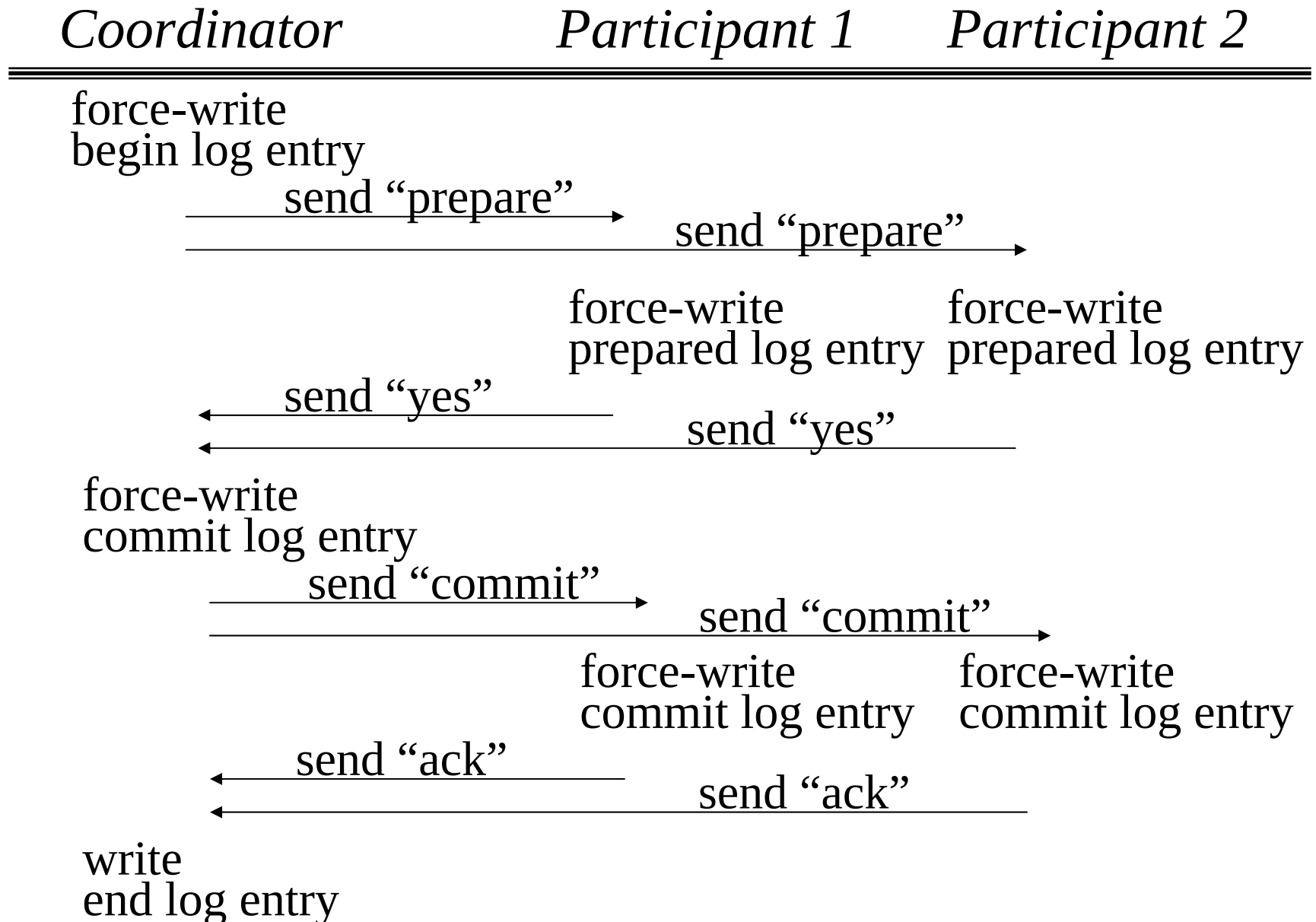
- Distributed handshake protocol known as **two-phase commit (2PC)**
- with a **coordinator** taking responsibility for unanimous outcome
- Recovery considerations for in-doubt transactions

2PC During Normal Operation

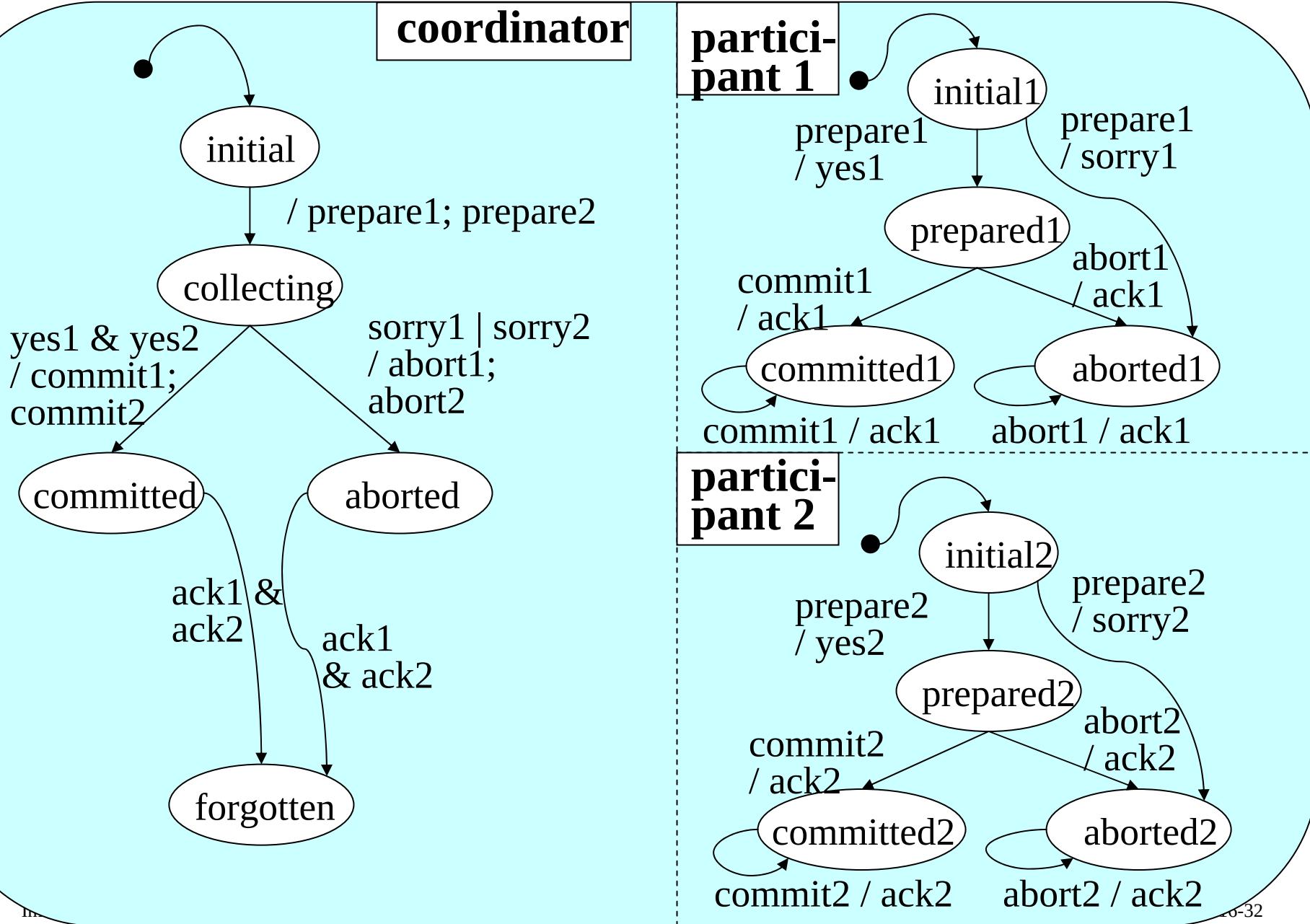
- **First phase (voting):**
coordinator sends *prepare* messages to participants and waits for *yes* or *no* votes
- **Second phase (decision)**
coordinator sends *commit* or *rollback* messages to participants and waits for *acks*
- **Participants** write *prepared* log entries in voting phase and become *in-doubt (uncertain)*
→ potential **blocking** danger, breach of local autonomy
- Participants write *commit* or *rollback* log entry in decision phase
- **Coordinator** writes *begin* log entry
- Coordinator writes *commit* or *rollback* log entry and can now give return code to the client's commit request
- Coordinator writes *end (done, forgotten)* log entry to facilitate **garbage collection**

→ $4n$ messages, $2n+2$ forced log writes, 1 unforced log write with n participants and 1 coordinator

Illustration of 2PC



Statechart for Basic 2PC



Restart and Termination Protocol

Failure model:

- process failures: transient server crashes
- network failures: message losses, message duplications
- assumption that there are no malicious commission failures
→ Byzantine agreement
- no assumptions about network failure handling
→ can use datagrams or sessions for communication

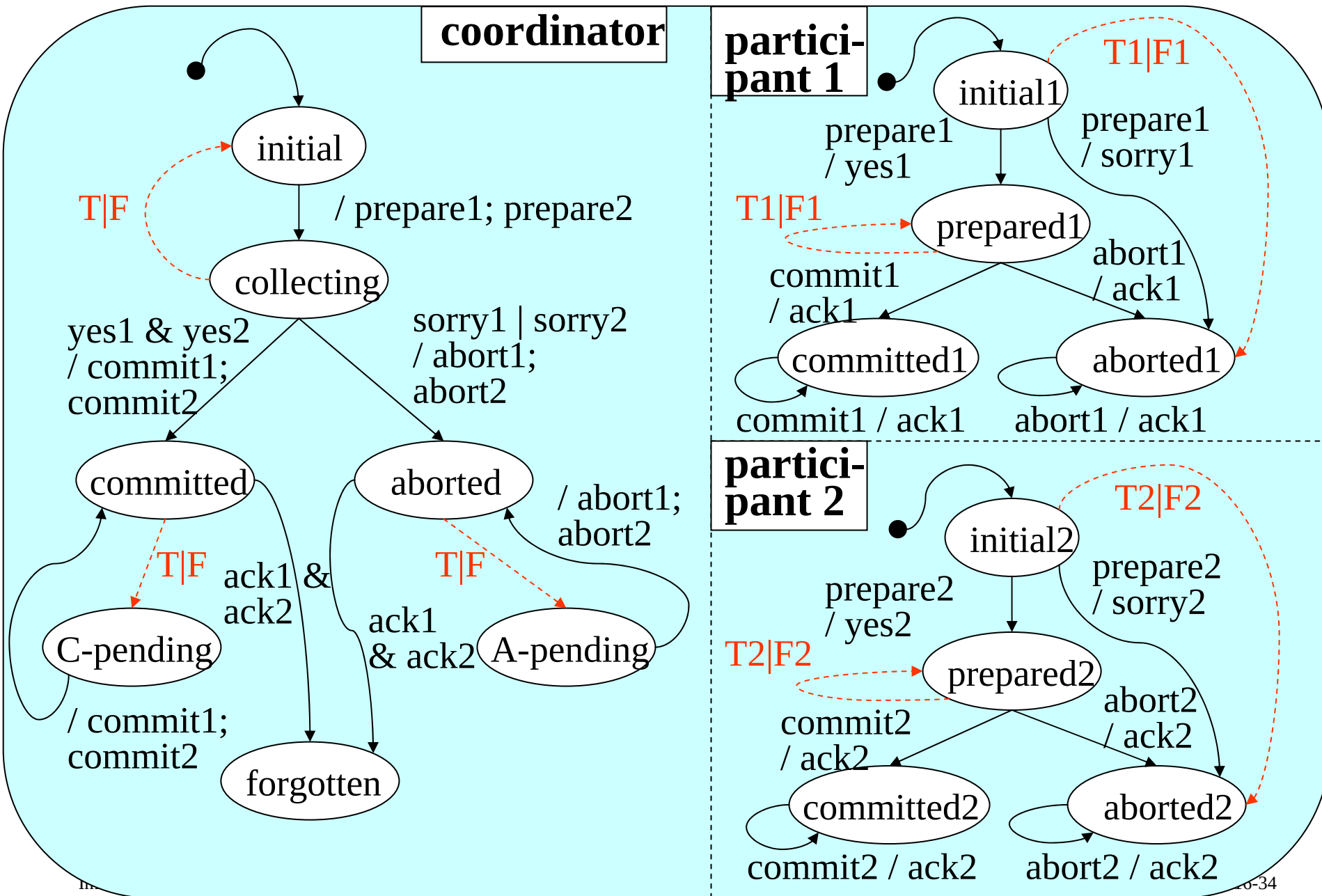
Restart protocol after failure (F transitions):

- coordinator restarts in last remembered state and resends messages
- participant restarts in last remembered state and resends message or waits for message from coordinator

Termination protocol upon timeout (T transitions):

- coordinator resends messages
and may decide to abort the transaction in first phase
- participant can unilaterally abort in first phase and wait for or may contact coordinator in second phase

Statechart for Basic 2PC with Restart/Termination



Correctness of Basic 2PC

Theorem 19.1 (Safety):

2PC guarantees that if one process is in a final state, then either all processes are in their committed state or all processes are in their aborted state.

Proof methodology:

Consider the set of possible computation paths starting in global state (initial, initial, ..., initial) and reason about invariants for states on computation paths.

Theorem 19.2 (Liveness):

For a finite number of failures the 2PC protocol will eventually reach a final global state within a finite number of state transitions.

Independent Recovery

Independent recovery: ability of a failed and restarted process to terminate his part of the protocol without communicating to other processes.

Theorem:

There exists no distributed commit protocol that can guarantee independent process recovery in the presence of multiple failures (e.g., network partitionings).

Informationssysteme

Kapitel 17 – Einführung in XML

17.1 XML?

17.2 Beispielanwendungen für XML

17.3 Grundlagen von XML

17.4 Schemabeschreibung mit DTDs

17.5 XPath

XML - Snake Oil of this Century?

- *Snake Oil* is the all-curing drug these strange guys in wild-west movies sell, travelling from town to town, but visiting each town only once.
- Google: „snake oil“ xml
 - ⇒ some 2000 hits
 - „XML revolutionizes software development“
 - „XML is the all-healing, world-peace inducing tool for computer processing“
 - „XML enables application portability“
 - „Forget the Web, XML is the new way to business“
 - „XML is the cure for your data exchange, information integration, data exchange, [x-2-y], [you name it] problems“
 - „XML, the Mother of all Web Application Enablers“
 - „XML has been the best invention since sliced bread“

XML is not...

- **A replacement for HTML**
(but HTML can be generated from XML)
- **A presentation format**
(but XML can be converted into one)
- **A programming language**
(but it can be used with almost any language)
- **A network transfer protocol**
(but XML may be transferred over a network)
- **A database**
(but XML may be stored into a database)

But then – what is it?

**XML is a meta markup language
for text documents / textual data**



**XML allows to define languages
(„applications“) to represent text
documents / textual data**

XML by Example

```
<article>
```

```
  <author>Gerhard Weikum</author>
```

```
  <title>The Web in 10 Years</title>
```

```
</article>
```

- Easy to understand for human users
- Very expressive (semantics along with the data)
- Well structured, easy to read and write from programs

This looks nice, but...

XML by Example

... this is XML, too:

<t108>

<x87>Gerhard Weikum</x87>

<g10>The Web in 10 Years</g10>

</t108>

- **Hard** to understand for human users
- **Not** expressive (**no** semantics along with the data)
- Well structured, easy to read and write from programs

XML by Example

... and what about this XML document:

<data>

ch37fhgks73j5mv9d63h5mgfkds8d984lgnsmcns983

</data>

- **Impossible** to understand for human users
- **Not** expressive (**no** semantics along with the data)
- **Unstructured**, read and write only with **special** programs

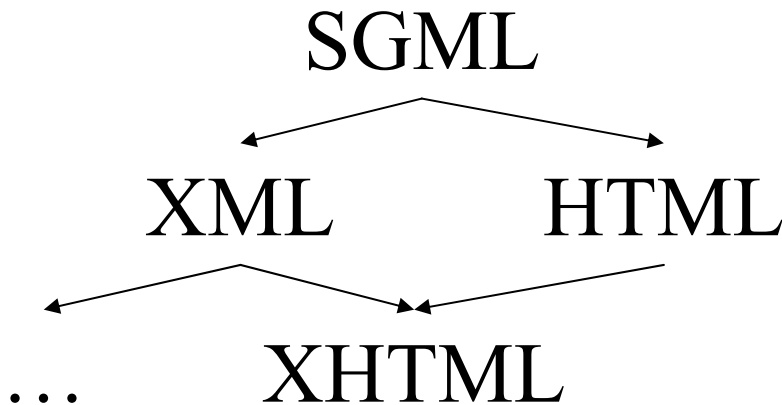
The actual benefit of using XML highly depends on the design of the application.

Possible Advantages of Using XML

- Truly Portable Data
- Easily readable by human users
- Very expressive (semantics near data)
- Very flexible and customizable (no finite tag set)
- Easy to use from programs (libs available)
- Easy to convert into other representations (XML transformation languages)
- Many additional standards and tools
- Widely used and supported

History of XML

- Based on (i.e., an application of) SGML (Standard **G**eneralized **M**arkup **L**anguage) from 1970ies
 - HTML most prominent SGML application
 - Too complicated and complex (>150 pages of spec)
 - Hardly ever fully implemented, only incompatible subsets
- XML as „SGML light“ on February 10th, 1998
- Many additional „standards“ since then



Design Goals for XML

- XML shall be straightforwardly usable over the Internet.
- XML shall support a wide variety of applications.
- XML shall be compatible with SGML.
- It shall be easy to write programs which process XML documents.
- The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
- XML documents should be human-legible and reasonably clear.
- The XML design should be prepared quickly.
- The design of XML shall be formal and concise.
- XML documents shall be easy to create.
- Terseness in XML markup is of minimal importance.

App. Scenario 1: Content Mgt.



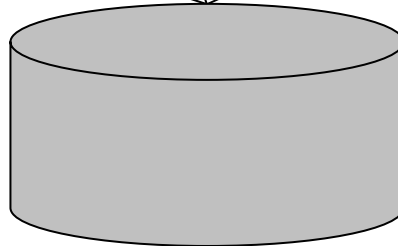
Clients

XML2HTML

XML2WML

XML2PDF

Converters



Database with
XML documents

App. Scenario 1: Examples

- The Linux Documentation Project **LDP**
- Digital Bibliography & Library Project **DBLP**
- Many Content Management Systems internally store information as XML and convert on-the-fly, among them **Apache Cocoon** (but this is typically not visible from the generated HTML pages)

Advantages:

- Separation of Content and Presentation
- New Client Types easily added

But: Additional Load on the Server (→Caching)

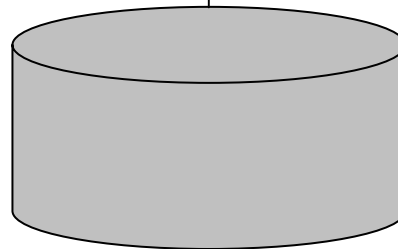
App. Scenario 1 in the future



Clients

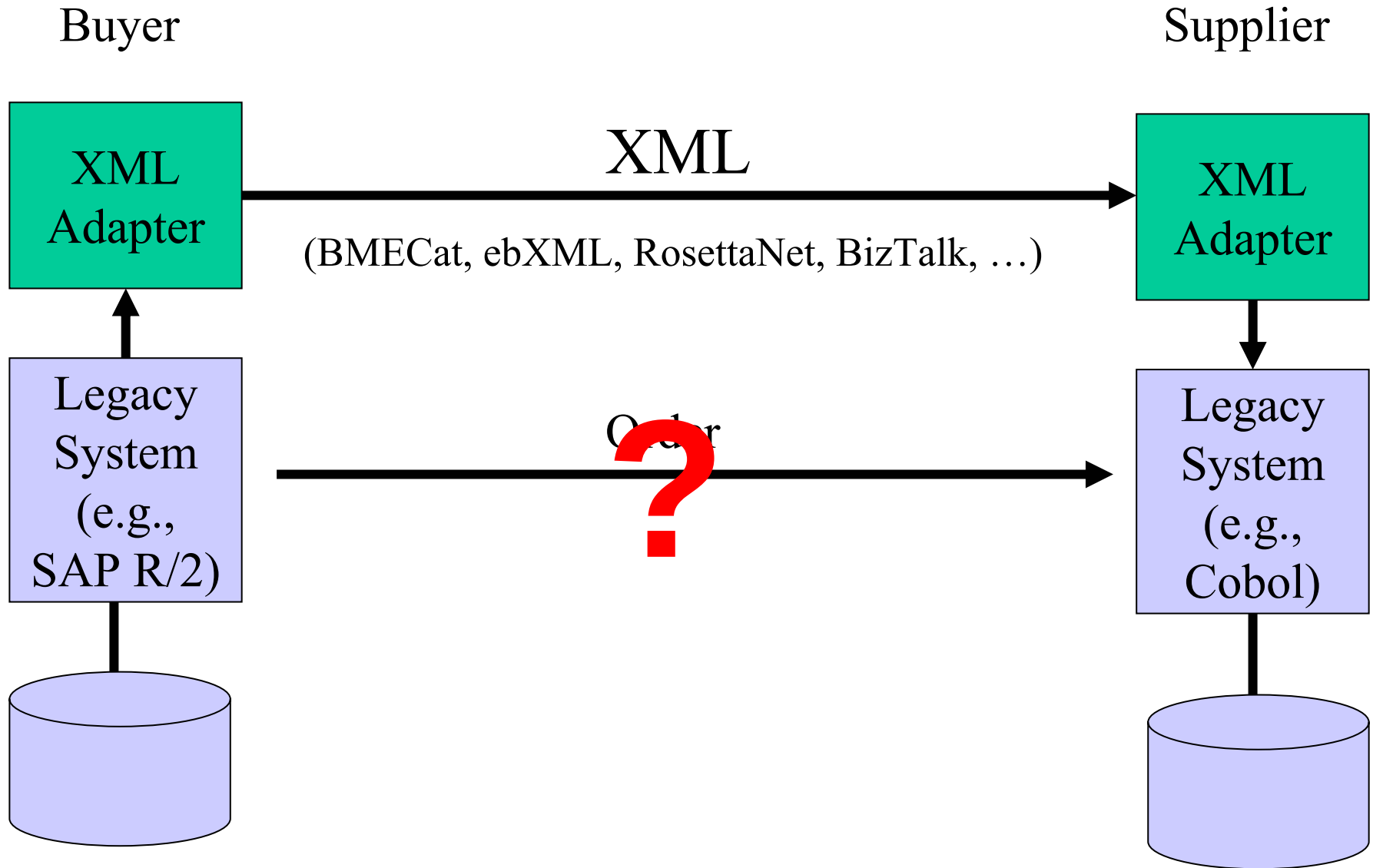
XML + converter

Web Server



Database with
XML documents

App. Scenario 2: Data Exchange



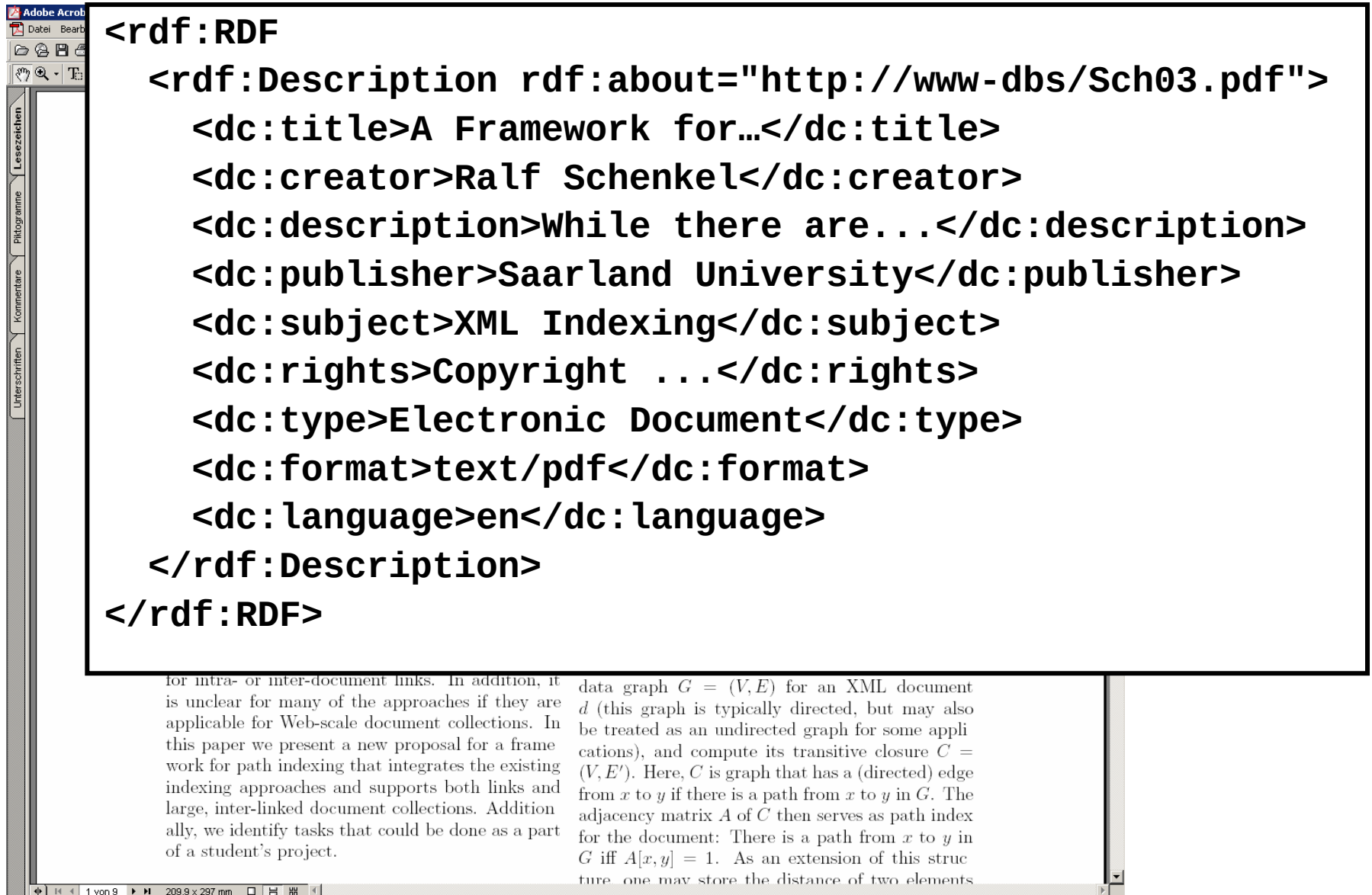
App. Scenario 2: Data Exchange

- Federated Information Systems based on XML
- For n legacy systems, only n system-specific adapters required (instead of n^2 system-to-system adapters)
- Easily extendible

But: All Adapters must support the same XML application

Another important XML application: Web Services, SOAP
(XML used for specifying function calls, their parameters, and the results)

App. Scenario 3: XML for Metadata



The screenshot shows an Adobe Acrobat window with a sidebar on the left containing icons for 'Datei', 'Bearbeiten', 'Leserzeichen', 'Piktogramme', 'Kommentare', and 'Unterschriften'. The main content area displays an XML structure for an RDF description of a PDF document.

```
<rdf:RDF
  <rdf:Description rdf:about="http://www-dbs/Sch03.pdf">
    <dc:title>A Framework for...</dc:title>
    <dc:creator>Ralf Schenkel</dc:creator>
    <dc:description>While there are...</dc:description>
    <dc:publisher>Saarland University</dc:publisher>
    <dc:subject>XML Indexing</dc:subject>
    <dc:rights>Copyright ...</dc:rights>
    <dc:type>Electronic Document</dc:type>
    <dc:format>text/pdf</dc:format>
    <dc:language>en</dc:language>
  </rdf:Description>
</rdf:RDF>
```

for intra- or inter-document links. In addition, it is unclear for many of the approaches if they are applicable for Web-scale document collections. In this paper we present a new proposal for a framework for path indexing that integrates the existing indexing approaches and supports both links and large, inter-linked document collections. Additionally, we identify tasks that could be done as a part of a student's project.

data graph $G = (V, E)$ for an XML document d (this graph is typically directed, but may also be treated as an undirected graph for some applications), and compute its transitive closure $C = (V, E')$. Here, C is graph that has a (directed) edge from x to y if there is a path from x to y in G . The adjacency matrix A of C then serves as path index for the document: There is a path from x to y in G iff $A[x, y] = 1$. As an extension of this structure, one may store the distance of two elements

App. Scenario 4: Document Markup

```
<article>
```

```
  <section id=„1“ title=„Intro“>
```

```
    This article is about <index>XML</index>.
```

```
  </section>
```

```
  <section id=„2“ title=„Main Results“>
```

```
    <name>Weikum</name> <cite idref=„Weik01“/> shows  
    the following theorem (see Section <ref idref=„1“/>)
```

```
    <theorem id=„theo:1“ source=„Weik01“>
```

```
      For any XML document x, ...
```

```
    </theorem>
```

```
  </section>
```

```
  <literature>
```

```
    <cite id=„Weik01“><author>Weikum</author></cite>
```

```
  </literature>
```

```
</article>
```

App. Scenario 4: Document Markup

- Document Markup adds structural and semantic information to documents, e.g.
 - Sections, Subsections, Theorems, ...
 - Cross References
 - Literature Citations
 - Index Entries
 - Named Entities
- This allows queries like
 - Which articles cite Weikum's XML paper from 2001?
 - Which articles talk about (the named entity) „Weikum“?

XML Standards – an Overview

- XML Core Working Group:
 - XML 1.0 (Feb 1998), 1.1 (candidate for recommendation)
 - XML Namespaces (Jan 1999)
 - XML Inclusion (candidate for recommendation)
- XSLT Working Group:
 - XSL Transformations 1.0 (Nov 1999), 2.0 planned
 - XPath 1.0 (Nov 1999), 2.0 planned
 - eXtensible Stylesheet Language XSL(-FO) 1.0 (Oct 2001)
- XML Linking Working Group:
 - XLink 1.0 (Jun 2001)
 - XPointer 1.0 (March 2003, 3 substandards)
- XQuery 1.0 (Nov 2002) plus many substandards
- XMLSchema 1.0 (May 2001)
- ...

17.3 XML Documents

What's in an XML document?

- Elements
- Attributes
- Entity References
- CDATA Sections
- Comments
- Processing Instructions

A Simple XML Document

```
<article>
  <author>Gerhard Weikum</author>
  <title>The Web in Ten Years</title>
  <text>
    <abstract>In order to evolve...</abstract>
    <section number="1" title="Introduction">
      The <index>Web</index> provides the universal...
    </section>
  </text>
</article>
```

A Simple XML Document

The diagram illustrates the concept of 'Freely definable tags' in XML. A blue box on the right contains the text 'Freely definable tags'. Four blue arrows originate from this box and point to specific XML tags in the document: the opening <article> tag, the <author> tag, the <abstract> tag, and the <section number='1' title='Introduction'> tag. This highlights that these tags are not part of a fixed schema but are instead defined by the document creator.

```
<article>  
  <author>Gerhard Weikum</author>  
  <title>The Web in Ten Years</title>  
  <text>  
    <abstract>in order to evolve...</abstract>  
    <section number="1" title="Introduction">  
      The <index>Web</index> provides the universal...  
    </section>  
  </text>  
</article>
```


A Simple XML Document

```
<article>
```

Start Tag

```
<author>Gerhard Weikum</author>
```

```
<title>The web in Ten Years</title>
```

```
<text>
```

```
<abstract>In order to evolve...</abstract>
```

```
<section number="1" title="Introduction">
```

```
  The <index>Web</index> provides the universal...
```

```
</section>
```

```
</text>
```

```
</article>
```

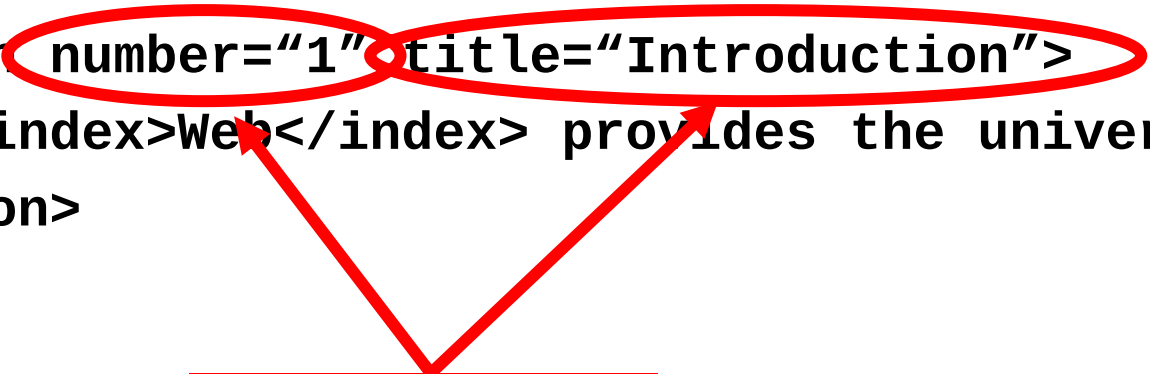
End Tag

Element

**Content of
the Element
(Subelements
and/or Text)**

A Simple XML Document

```
<article>
  <author>Gerhard Weikum</author>
  <title>The Web in Ten Years</title>
  <text>
    <abstract>In order to evolve...</abstract>
    <section number="1" title="Introduction">
      The <index>Web</index> provides the universal...
    </section>
  </text>
</article>
```

A red oval highlights the attributes 'number="1"' and 'title="Introduction"' of the <section> tag. Two red arrows originate from a red box at the bottom and point to the equals signs in these two attributes, illustrating the structure of an attribute (name=value).

**Attributes with
name and value**

Elements in XML Documents

- (Freely definable) **tags**: **article**, **title**, **author**
 - with start tag: **<article>** etc.
 - and end tag: **</article>** etc.
- **Elements**: **<article> ... </article>**
- Elements have a **name** (**article**) and a **content** (...)
- Elements may be nested.
- Elements may be empty: **<this_is_empty/>**
- Element content is typically parsed character data (PCDATA), i.e., strings with special characters, and/or nested elements (*mixed content* if both).
- Each XML document has exactly one root element and forms a tree.
- Elements with a common parent are ordered.

Elements vs. Attributes

Elements may have **attributes** (in the start tag) that have a **name** and a **value**, e.g. `<section number="1">`.

What is the difference between elements and attributes?

- Only one attribute with a given name per element (but an arbitrary number of subelements)
- Attributes have no structure, simply strings (while elements can have subelements)

As a rule of thumb:

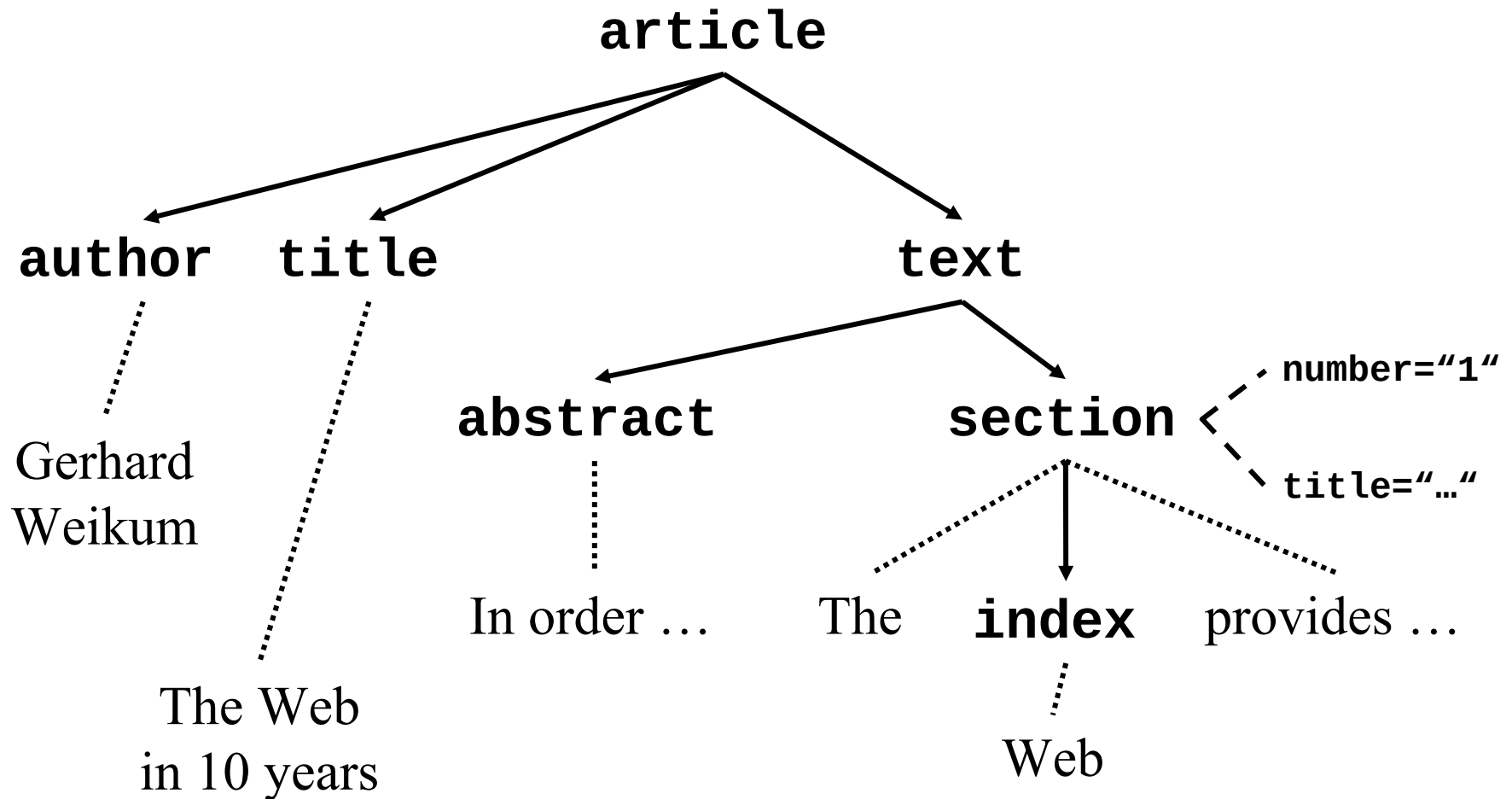
- Content into elements
- Metadata into attributes

Example:

`<person born="1912-06-23" died="1954-06-07">`

`Alan Turing</person>` proved that...

XML Documents as Ordered Trees



Preorder Traversal of the XML Tree

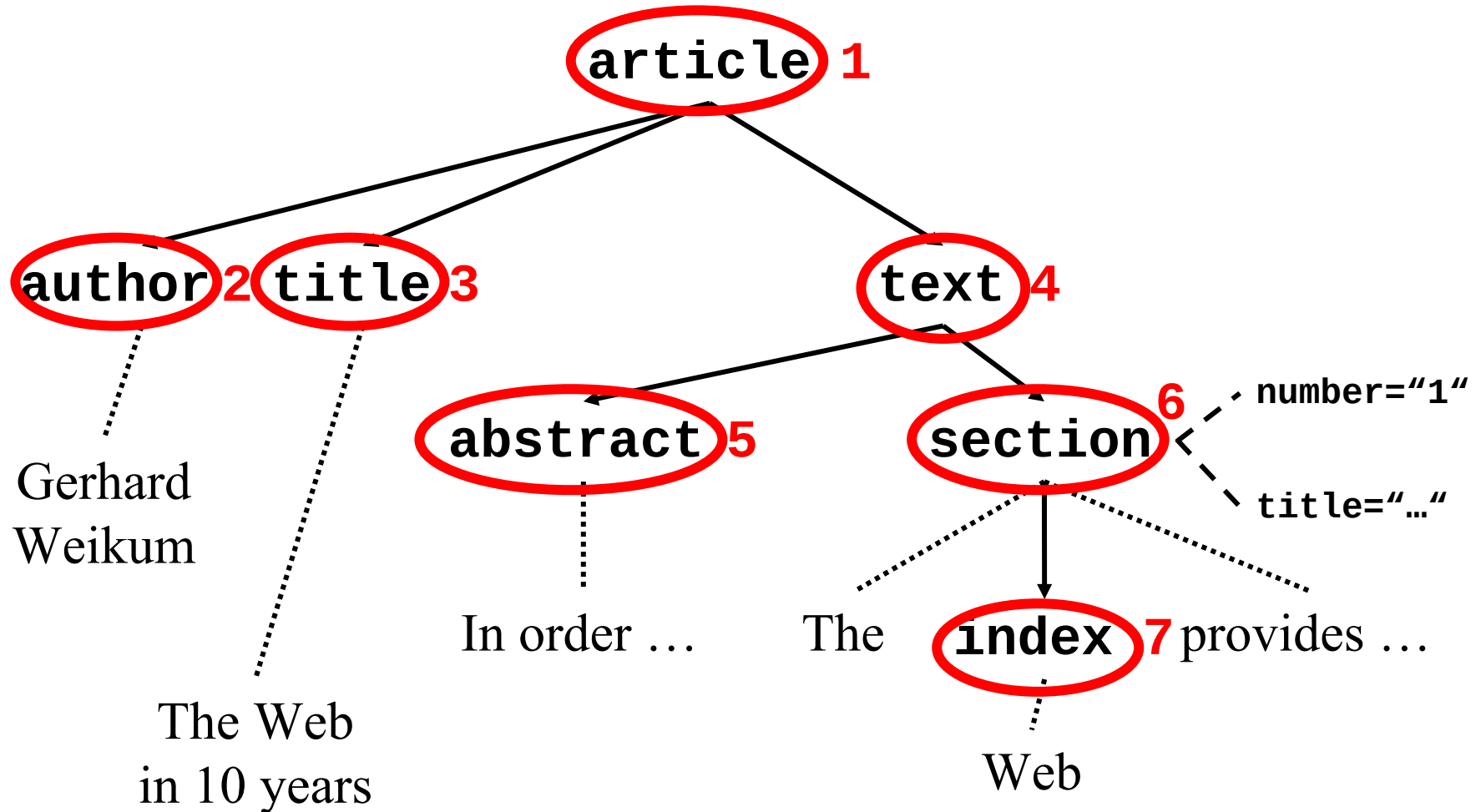
Goal: Find a *numbering* for the nodes in the XML Tree such that it can be used for linearizing the tree (i.e., create a document from it)

Solution: Preorder Traversal of the XML Tree (order in which the traversal enters nodes)

```
int traverse(node n,int rank)
{ n.rank=rank;
  for all child elements c of n from left to right
  do
    rank=traverse(c,rank+1);
  od
  return rank;
}
```

Start of the algorithm: **traverse(root,1)**

Preorder example



Well-Formed XML Documents

A **well-formed** document must adhere to, among others, the following rules:

- Every start tag has a matching end tag.
- Elements may nest, but must not overlap.
- There must be exactly one root element.
- Attribute values must be quoted.
- An element may not have two attributes with the same name.
- Comments and processing instructions may not appear inside tags.
- No unescaped < or & signs may occur inside character data.

Well-Formed XML Documents

A **well-formed** document must adhere to, among others, the following rules:

- Every start tag has a matching end tag.
- Elements may nest, but must not overlap.
- The
- Att
- An
- nan
- Comments and processing instructions may not appear inside tags.
- No unescaped < or & signs may occur inside character data.

**Only well-formed documents
can be processed by XML
parsers.**

Documents vs. Data

	Document	Data
Structure	semistructured, unstructured	well structured
Content	text (with markup) + meta data	data (numbers, strings, dates, ...)
Mixed Content	yes	no
Element order	matters	does not matter
Audience	humans	computers
Permanence	permanent	often transitory
Elements	often generic	application-specific
Example elements	section, title, heading, ...	age, salary, amount, price, ...

17.4 Document Type Definitions

Sometimes XML is *too* flexible:

- Most Programs can only process a subset of all possible XML applications
- For exchanging data, the format (i.e., elements, attributes and their semantics) must be fixed

⇒ **Document Type Definitions (DTD)** for establishing the vocabulary for one XML application (in some sense comparable to *schemas* in databases)

A document is **valid with respect to a DTD** if it conforms to the rules specified in that DTD.

Most XML parsers can be configured to validate.

DTD Example: Elements

```
<!ELEMENT article (title,author+,text)>
```

```
<!ELEMENT title (#PCDATA)>
```

```
<!ELEMENT author (#PCDATA)>
```

```
<!ELEMENT text (abstract,section*,literature?)>
```

```
<!ELEMENT abstract (#PCDATA)>
```

```
<!ELEMENT section (#PCDATA|index)*>
```

```
<!ELEMENT literature (#PCDATA)>
```

```
<!ELEMENT index (#PCDATA)>
```

Content of the title element is parsed character data

Content of the text element may contain zero or more section elements in this position

Content of the article element is a title element, followed by one or more author elements, followed by a text element

Element Declarations in DTDs

One element declaration for each element type:

<!ELEMENT element_name content_specification>

where **content_specification** can be

- **(#PCDATA)** parsed character data
- **(child)** one child element
- **(c1,...,cn)** a sequence of child elements c1...cn
- **(c1|...|cn)** one of the elements c1...cn

For each component **c**, possible counts can be specified:

- **c** exactly one such element
- **c+** one or more
- **c*** zero or more
- **c?** zero or one

Plus arbitrary combinations using parenthesis:

<!ELEMENT f ((a|b)*,c+,(d|e))*>

More on Element Declarations

- Elements with mixed content:
<!ELEMENT text (#PCDATA|index|cite|glossary)*>
- Elements with empty content:
<!ELEMENT image EMPTY>
- Elements with arbitrary content (this is nothing for production-level DTDs):
<!ELEMENT thesis ANY>

Attribute Declarations in DTDs

Attributes are declared per element:

```
<!ATTLIST section number CDATA #REQUIRED  
               title  CDATA #REQUIRED>
```

declares two required attributes for element **section**.

element name

```
graph TD
    EN[element name] --> EN1[section]
    AN[attribute name] --> AN1[number]
    AT[attribute type] --> AT1[CDATA]
    AD[attribute default] --> AD1[#REQUIRED]
```

attribute name

attribute type

attribute default

Attribute Declarations in DTDs

Attributes are declared per element:

```
<!ATTLIST section number CDATA #REQUIRED  
                title  CDATA #REQUIRED>
```

declares two required attributes for element **section**.

Possible attribute defaults:

- **#REQUIRED** is required in each element instance
- **#IMPLIED** is optional
- **#FIXED default** always has this default value
- **default** has this default value if the attribute is omitted from the element instance

Important Attribute Types in DTDs

- **CDATA** string data
- **(A1|...|An)** enumeration of all possible values of the attribute
- **ID** unique ID to identify the element
- **IDREF** refers to **ID** attribute of some other element („intra-document link“)
- **IDREFS** list of **IDREF**, separated by white space

Attribute Examples

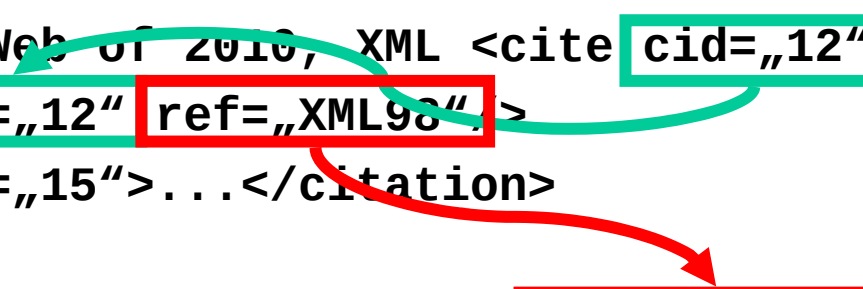
```
<ATTLIST publication type (journal|inproceedings) #REQUIRED
                        pubid ID #REQUIRED>
<ATTLIST cite          cid IDREF #REQUIRED>
<ATTLIST citation      ref IDREF #IMPLIED
                        cid ID #REQUIRED>
```

```
<publications>
  <publication type="journal" pubid="Weikum01">
    <author>Gerhard Weikum</author>
    <text>In the Web of 2010, XML <cite cid="12"/>...</text>
    <citation cid="12" ref="XML98"/>
    <citation cid="15">...</citation>
  </publication>
  <publication type="inproceedings" pubid="XML98">
    <text>XML, the extended Markup Language, ...</text>
  </publication>
</publications>
```

Attribute Examples

```
<ATTLIST publication type (journal|inproceedings) #REQUIRED
                        pubid ID #REQUIRED>
<ATTLIST cite          cid IDREF #REQUIRED>
<ATTLIST citation      ref IDREF #IMPLIED
                        cid ID #REQUIRED>
```

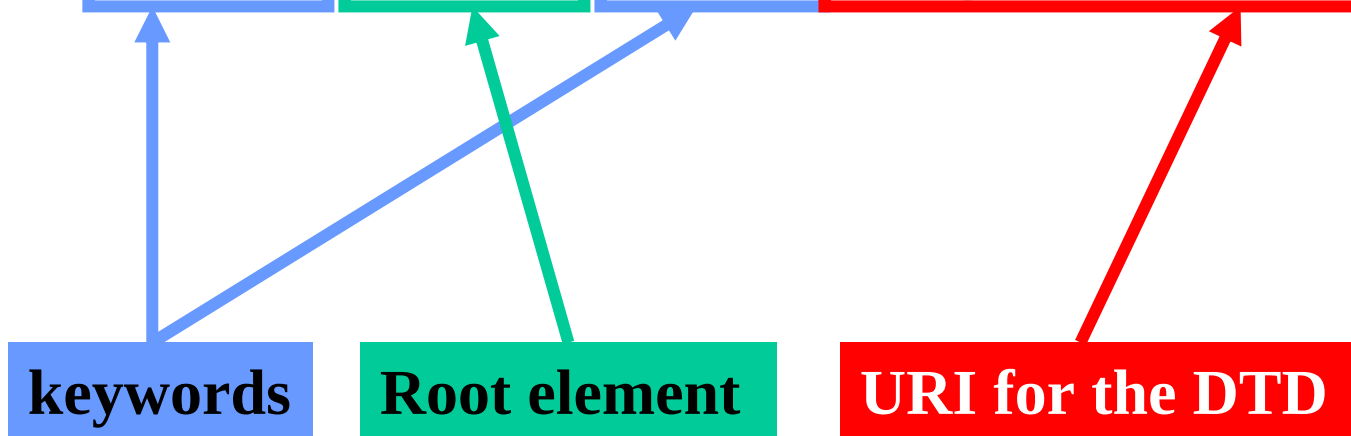
```
<publications>
  <publication type="journal" pubid="Weikum01">
    <author>Gerhard Weikum</author>
    <text>In the Web of 2010, XML <cite cid="12"/>...</text>
    <citation cid="12" ref="XML98">
      <citation cid="15">...</citation>
    </publication>
  <publication type="inproceedings" pubid="XML98">
    <text>XML, the extended Markup Language, ...</text>
  </publication>
</publications>
```



Linking DTD and XML Docs

- Document Type Declaration in the XML document:

`<!DOCTYPE article SYSTEM "http://www-dbs/article.dtd">`



Flaws of DTDs

- No support for basic data types like integers, doubles, dates, times, ...
- No structured, self-definable data types
- No type derivation
- id/idref links are quite loose (target is not specified)

⇒ XML Schema

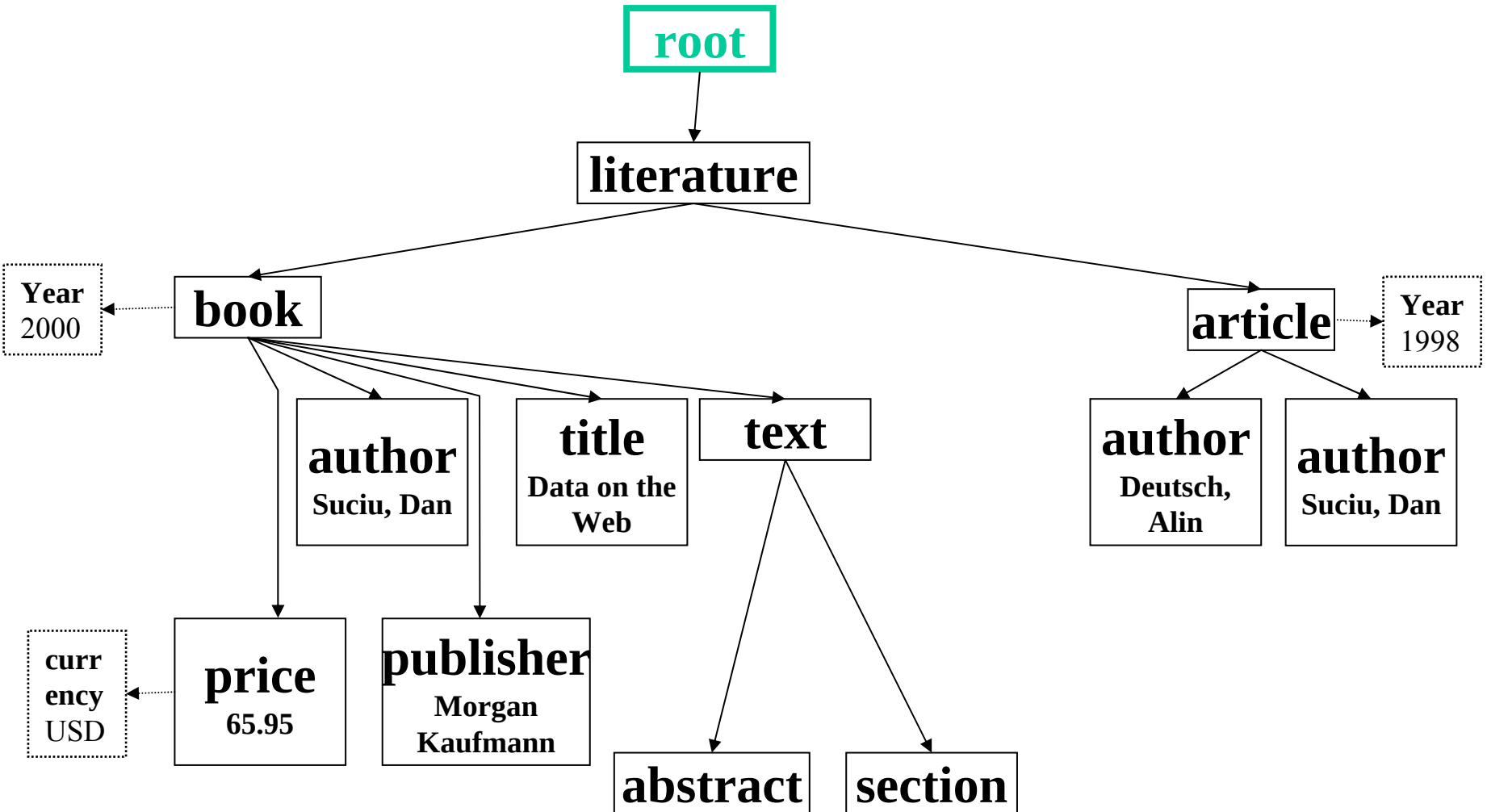
17.5 XPath

- XPath is a simple language to identify parts of the XML document (for further processing)
- XPath operates on an extended tree representation of the document, including comments and processing instructions (but no CDATA, entity references, DTD)
- Result of an XPath expression is a set of elements or attributes (**node-set**)

XML Example for XPath

```
<literature>
  <book year="2000" key="AS00">
    <author>Suciu, Dan</author>
    <title>Data on the Web</title>
    <text>
      <abstract>...</abstract><section>...</section>
    </text>
    <publisher>Morgan Kaufmann</publisher>
    <price currency="USD">65.95</price>
  </book>
  ...
  <article year="1998" key="DS98">
    <author>Deutsch, Alin</author>
    <author>Suciu, Dan</author>
    <title>XML-QL: A Query Language for XML</title>
  </article>
  ...
</literature>
```

Extended XML Tree Example



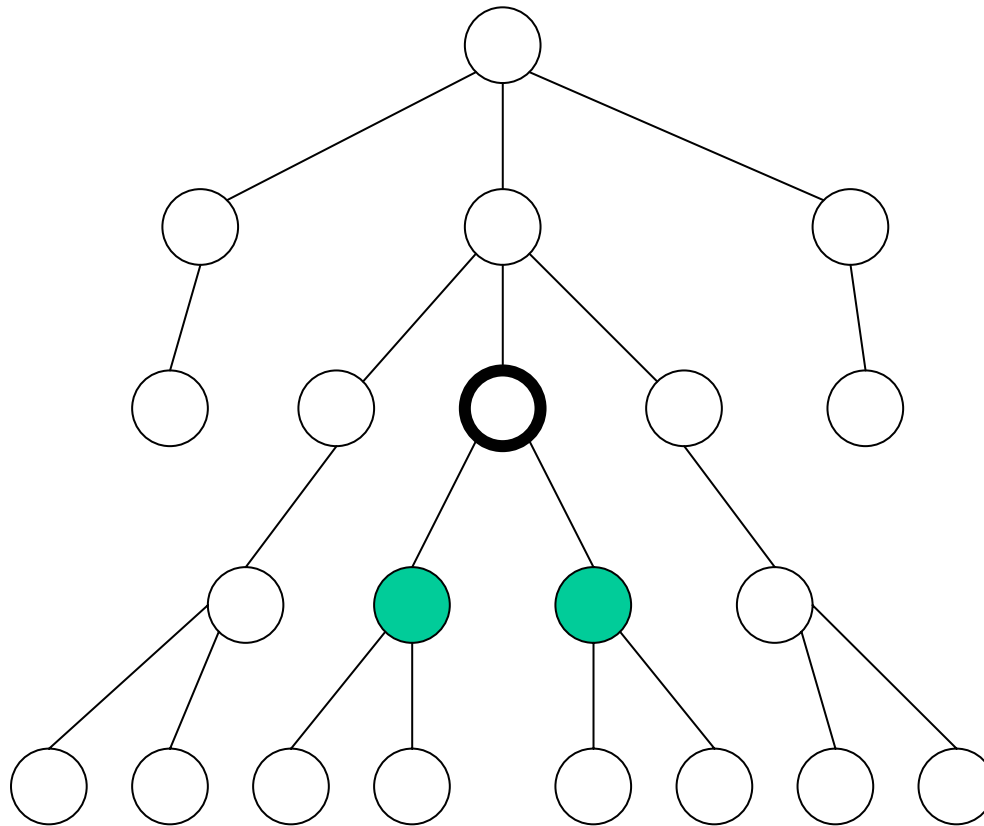
Elements of XPath

- A common XPath expression is a **location path** that consists of **location steps**, separated by /:
 //article/text/abstract: selects all **abstract** elements of articles
- A leading / always means the root element
- Each location step is evaluated in the context of a node in the tree, the so-called **context node** (which is among the results of the previous location step)
- The general form of a location step is ***axis::test[predicate]***
 - **axis** selects the node set for this step, starting from the context node
 - **test** is an additional restriction on the nodes
 - **predicate** is a filtering condition on the nodes

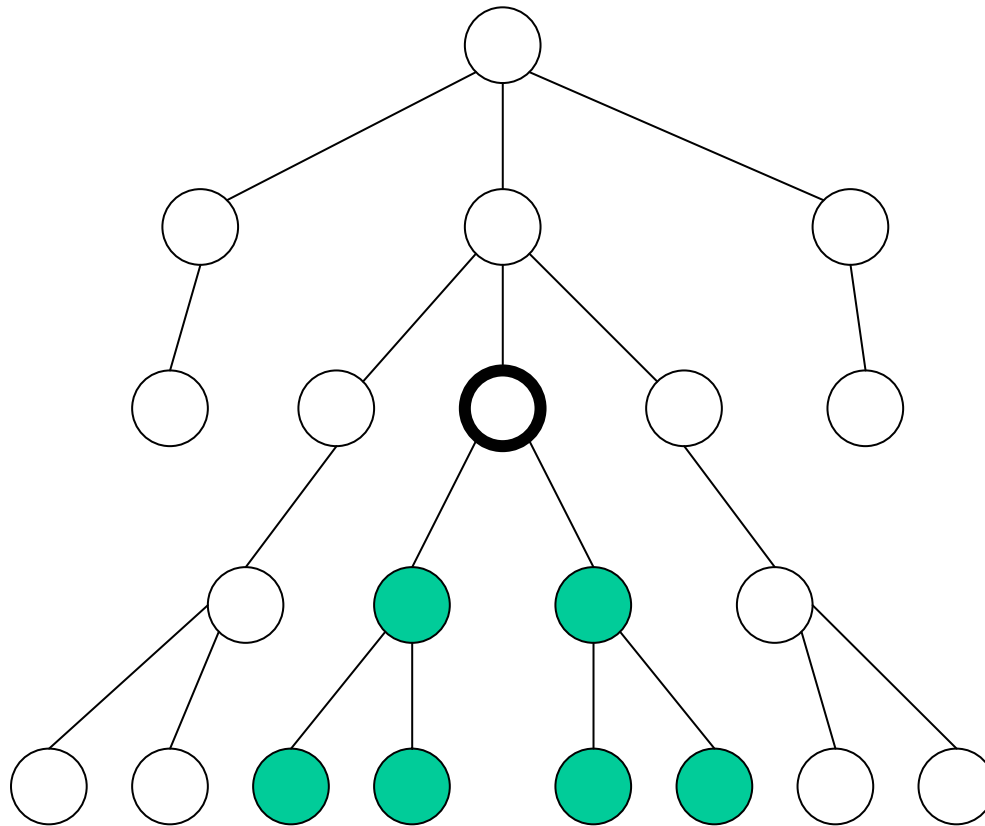
XPath Location Axes

Axis	Shortcut	Comment
child		
descendant	//	
descendant-or-self		
self	.	
parent	..	
ancestor		nodes in reverse preorder
ancestor-or-self		nodes in reverse preorder
following		
following-sibling		
preceding		nodes in reverse preorder
preceding-sibling		nodes in reverse preorder
attribute	@	for attributes

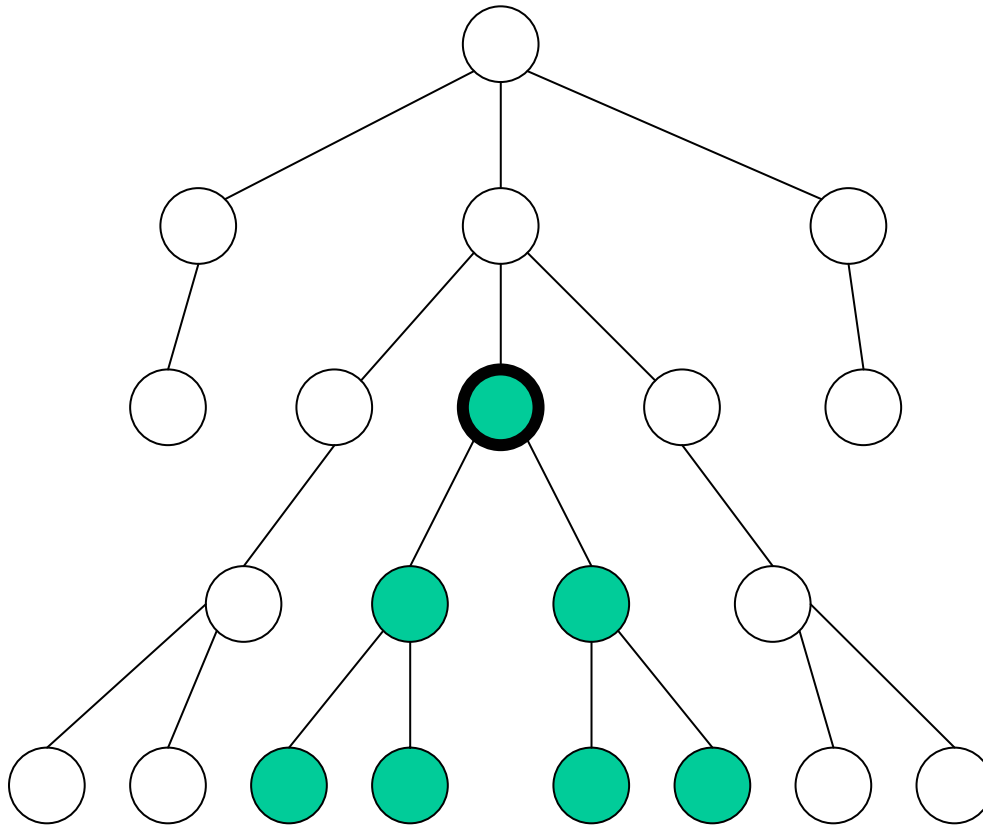
The child axis



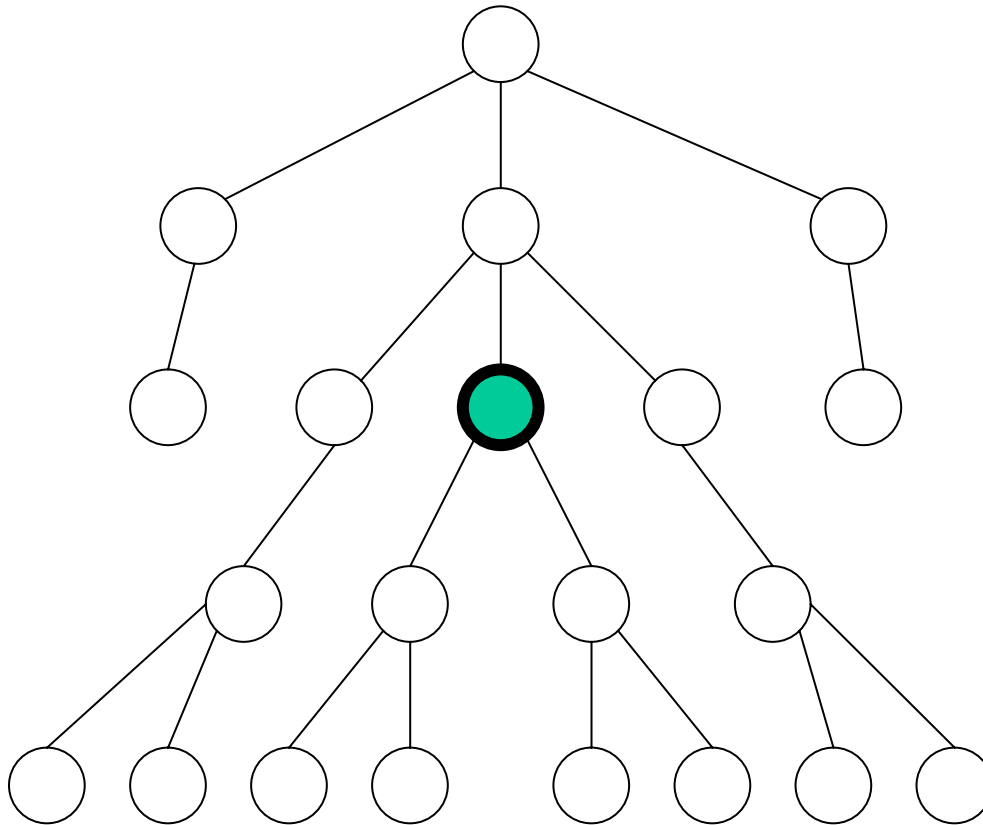
The descendant axis



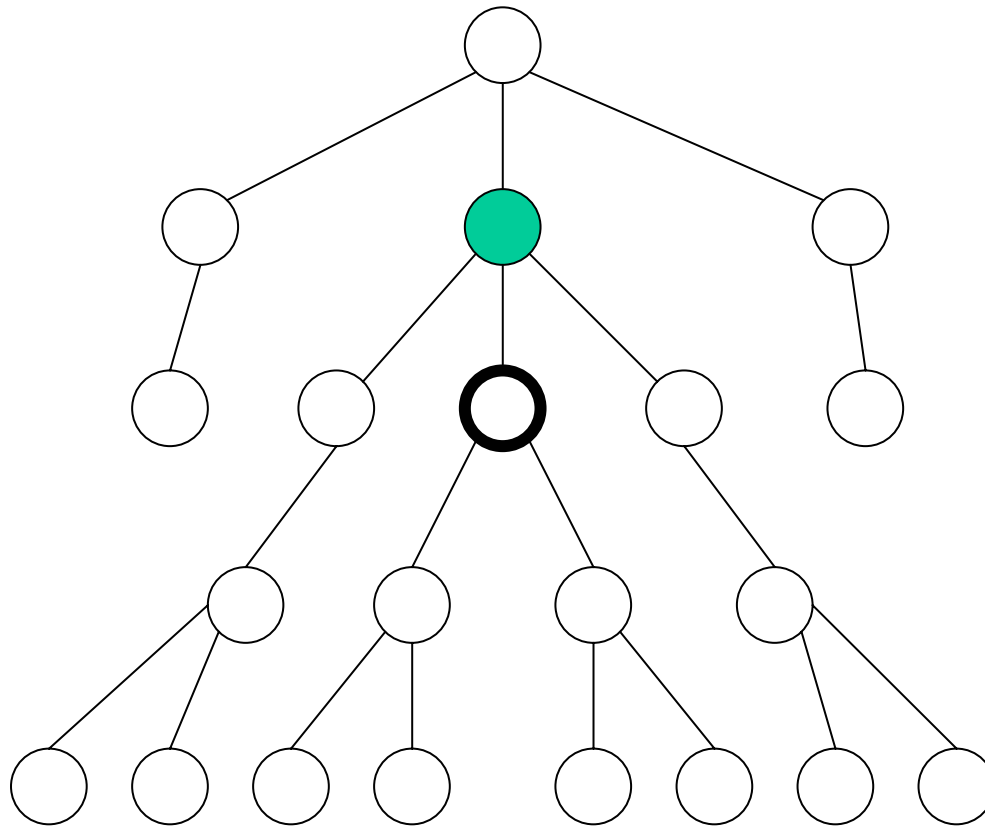
The descendant - or - self axis



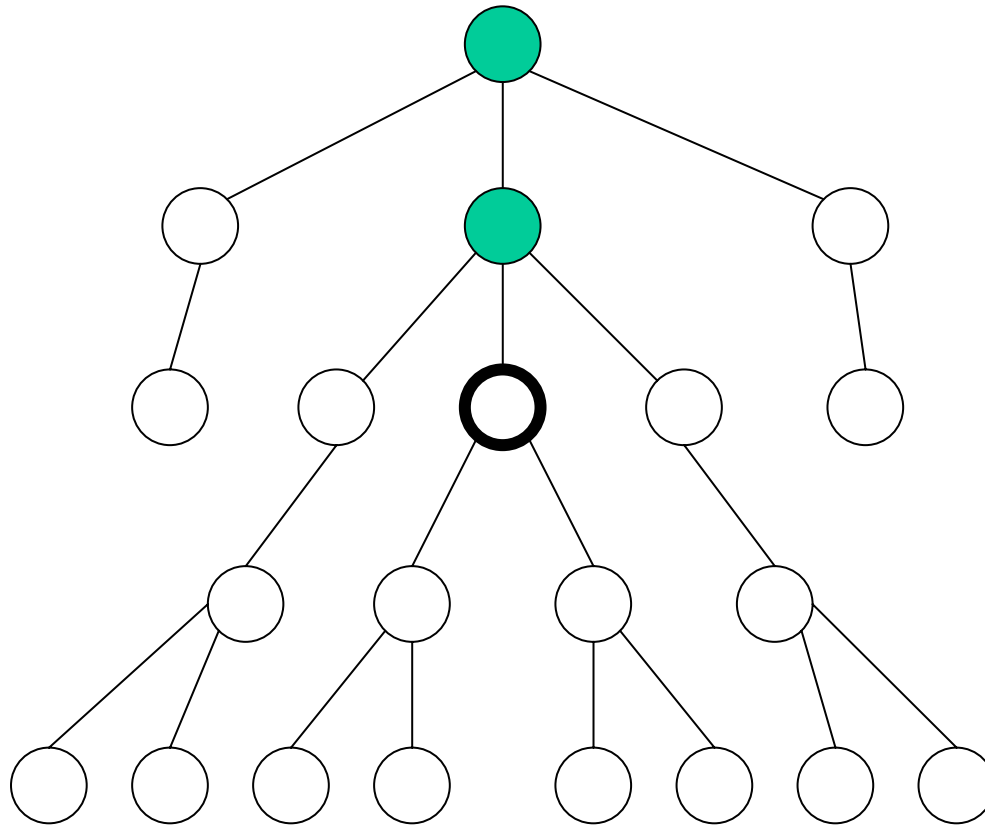
The self axis



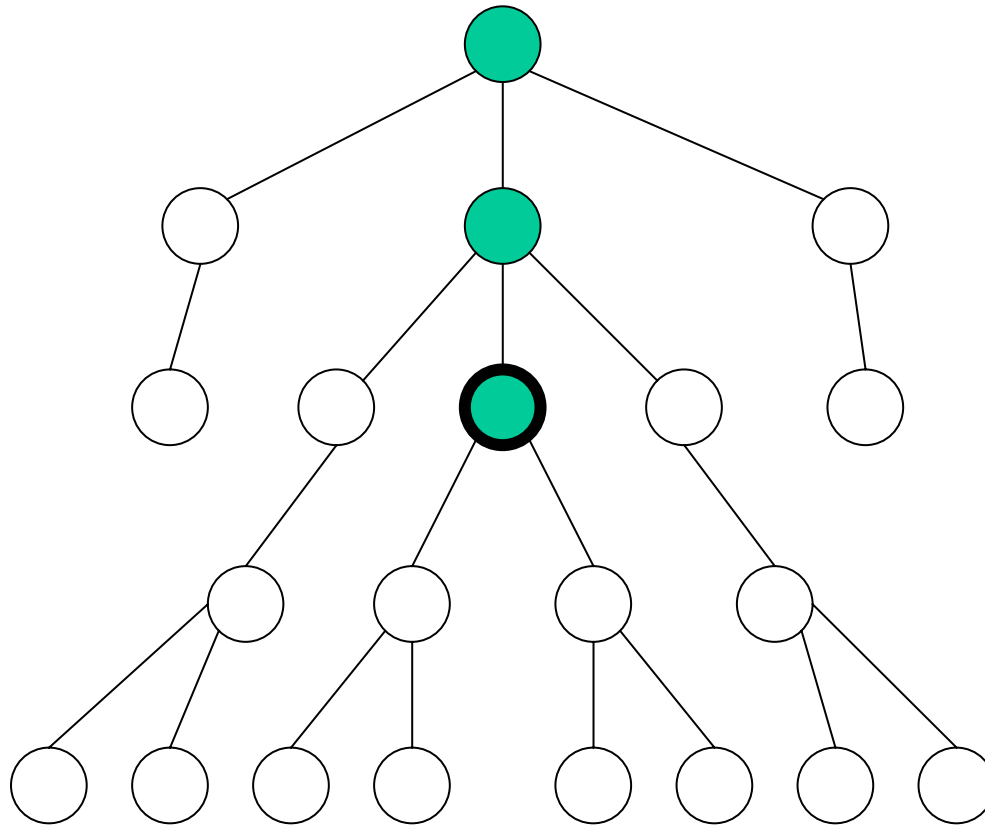
The parent axis



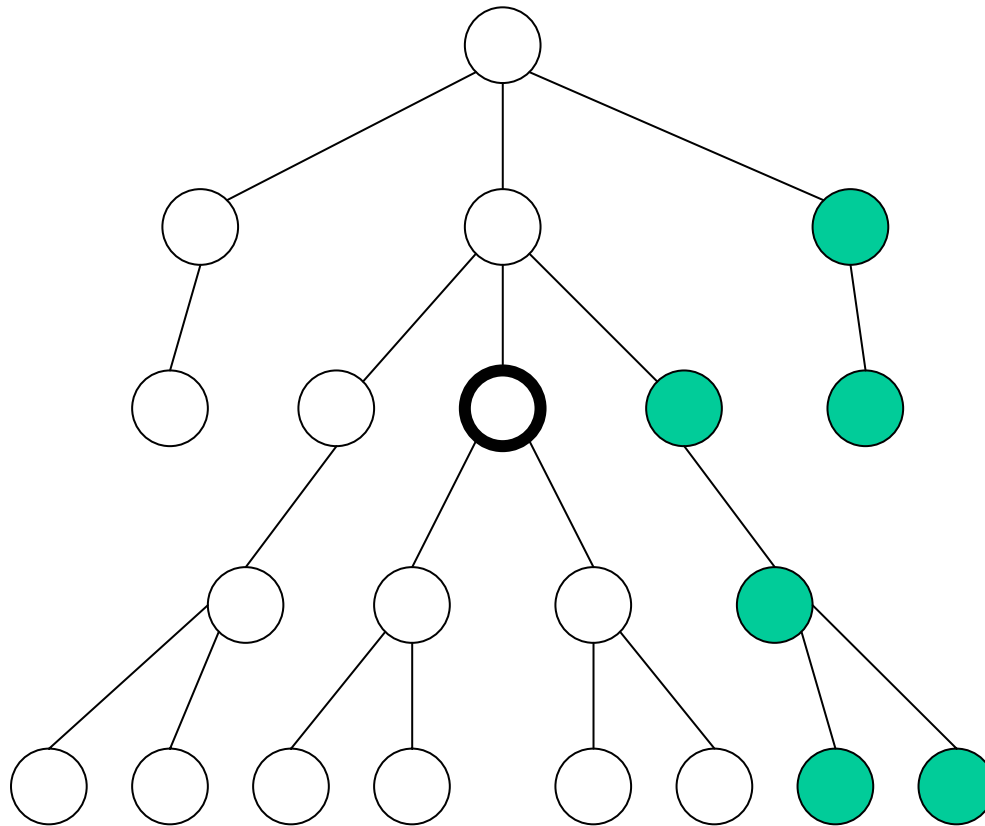
The ancestor axis



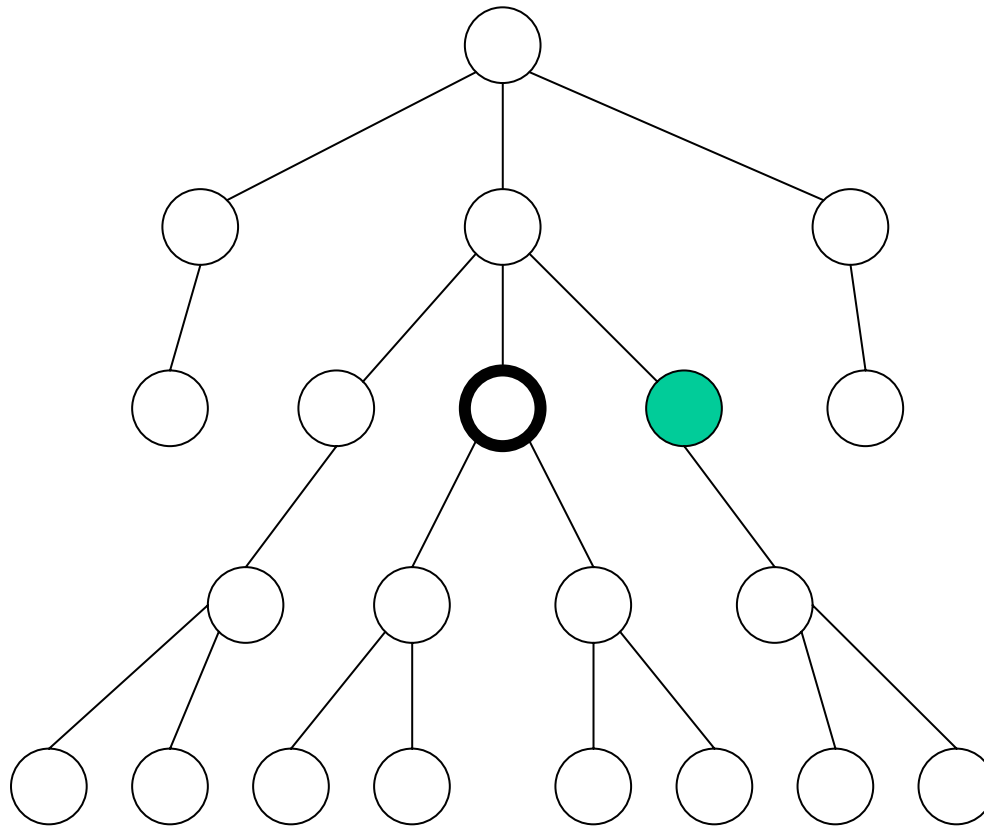
The ancestor-or-self axis



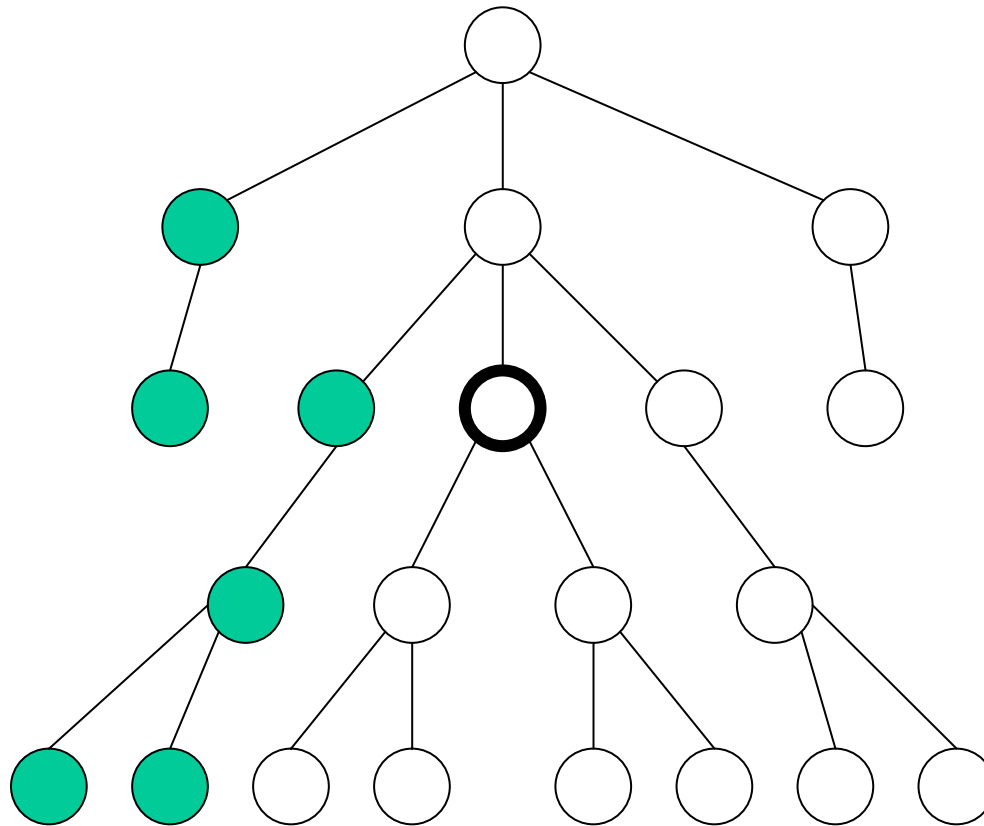
The following axis



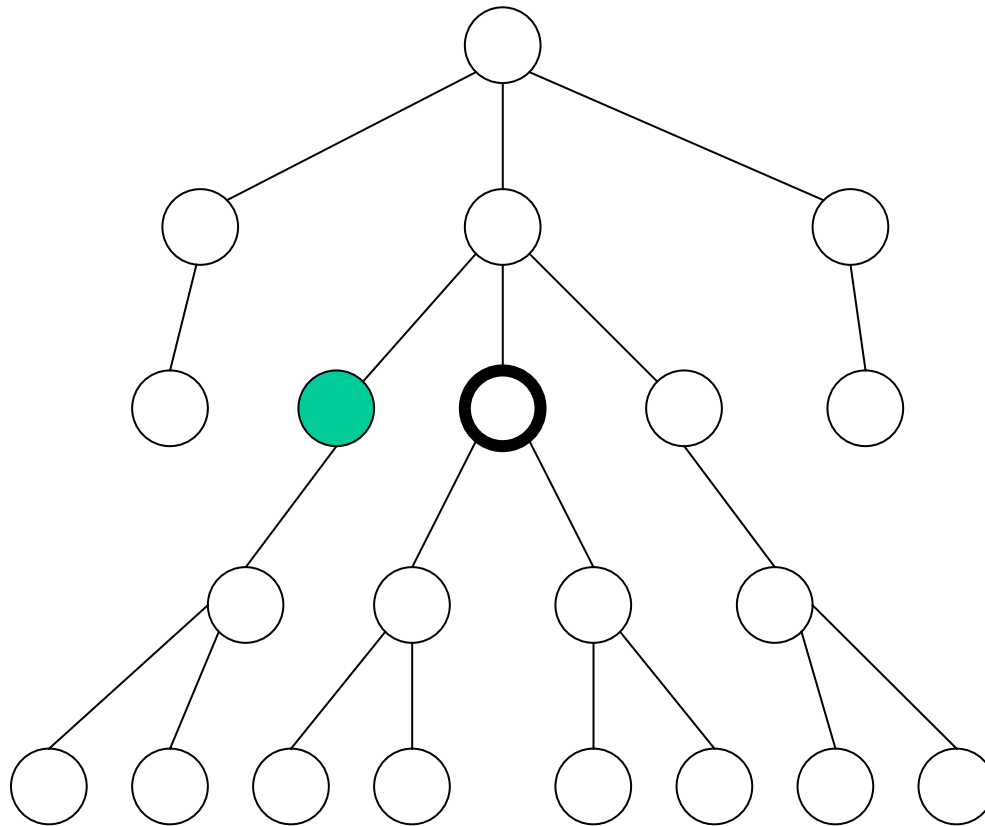
The following-sibling axis



The preceding axis



The preceding-sibling axis



Important XPath Location Tests

Axis	Nodes that qualify for the result
name	elements/attributes with name name
*	any elements/attributes
node()	any nodes
text()	nodes with string content

Simple XPath Examples

<code>/literature/book/author</code>	retrieves all book authors: starting with the root, traverses the tree, matches element names literature, book, author, and returns elements <code><author>Suciu, Dan</author></code> , <code><author>Abiteboul, Serge</author></code> , ..., <code><author><firstname>Jeff</firstname></code> <code><lastname>Ullman</lastname></author></code>
<code>/literature/*/author</code>	authors of books, articles, essays, etc.
<code>/literature//author</code>	authors that are descendants of literature
<code>/literature//@year</code>	value of the year attribute of descendants of literature

Predicates in Location Steps

- Added with [] to the location step
- Predicates are Boolean expressions over conditions on the results of location paths and elementary expressions (strings, numbers)
- Context nodes for the start of relative location paths is the result of the location step with the predicate
- Used to restrict elements that qualify as result of a location step to those that fulfil the predicate, e.g.:
 - **a[b]** elements **a** that have a subelement **b**
 - **a[@d]** elements **a** that have an attribute **d**
 - Plus conditions on content/value:
 - **a[b="c"]**
 - **a[@d>7]**
 - **<, <=, >=, !=, ...**
 - Boolean combination of conditions:
 - **a[b="c" and (@d>7 or @d<2)]**
 - Fancy combinations:
 - **a[b//c/d//f and (d="text1" or e/@g="text2")]**
- A location step may have more than one predicate

Some XPath Functions

- **count (//book)**: number of nodes
- **id ("DS98")**: choose nodes with this id
- **last ()**: size of the context
- **name (.)**: name of the first node of the argument
- **position ()**: position of the current context node

Shortcut for **[position()=i]: [i]**

Ex.: all second authors of articles

//article/author[2]

XPath Examples with Predicates

`/literature//author[firstname]` authors that have a subelement firstname

`/literature/book[price < „50“]` low priced books

`/literature/book[author//country = „Germany“]` books with German author

Sophisticated XPath Examples

Articles by Jeff Ullman that are older than 1970
and have appeared in the Journal of the ACM

```
/literature/article[author = "Ullman" and @year<„1970  
and * = "Journal of the ACM"]
```

Coauthors of Jeff Ullman:

```
/literature/*[author = „Ullman“]/author[.!="Ullman"]
```

Second author of books with Jeff Ullman being the first author

```
/literature/book/author[2][../author[1] = „Ullman“]
```

Design Rules for Efficient XPath

- Be as specific as possible: Use any information about document structure that is available
not `//article//author[2]`
but `/literature/article/author[2]`
Reduces complexity for evaluating the query
- Filter as early as possible: Try to decrease the number of intermediate results in early steps of the query
not `//article[@year=2000][author="Schenkel"]`
but `//article[author="Schenkel"][@year=2000]`
Benefit depends on *selectivity* of predicates, rule of thumb:
selective predicates to the front for increasing performance
(in the ex.: many articles from 2000, but only a few articles in the literature database authored by Schenkel)

Sources and Further Literature

- D. Hollander and C.M. Sperberg-McQueen: **Happy Birthday, XML**. <http://www.w3c.org/2003/02/xml-at-5.html>
- **The Linux Documentation Project**, <http://www.tldp.org/>
- **DBLP Computer Science Bibliography**, <http://dblp.uni-trier.de/>
- **The Resource Document Framework**, <http://www.w3.org/RDF/>
- Elliotte R. Harold, W. Scott Means: **XML in a Nutshell**, 2nd Edition, O'Reilly, 2002. Chapters 1-5, 9, 20, 22, 26.
- E. Wilde and D. Lowe: **XPath, XLink, XPointer, and XML**. Addison-Wesley, 2003.
- Slides and lecture notes of „XML Technologies“, SS03, <http://www.mpi-sb.mpg.de/units/ag5/teaching/ss03/xml/>

Informationssysteme

Kapitel 18 – Verwaltung von XML

18.1 Speicherung von XML in relationalen DBS

18.2 Indizierung von XML

18.3 Komprimierung von XML

18.4 XML-Datenbanken

18.1 Requirements for Storing XML

- Lossless reconstruction of documents
 - order matters only for document-centric documents
 - ignoring comments, processing instructions, entities, ...
- Efficient reconstruction of documents
- Efficient query evaluation on documents
 - Boolean queries with XPath
- Efficient updates of documents
 - Addition, deletion of documents
 - Modification of documents

Classification of Storage Methods

- Store complete XML documents in the file system
- Store in relational databases (RDBMS):
 - Store the structure of XML documents (i.e., the XML data graph) in generic tables
 - Derive a schema-specific database schema for storing XML documents (DTD or XML Schema required)

Big advantage: reuse existing RDBMS infrastructure and experience from the last 20 years

- Natively store XML documents in a dedicated database
But: re-invent all the auxiliary structures: indexes, cache, disk organization, transaction management, log, ...

XML and RDBMS

XML	Relational databases
hierarchical, arbitrary deep structure	flat, unnested tables
Elements may appear several times	Columns have one fixed value per row
Elements are ordered	Rows are unordered
Schema is optional and may be open	Schema is mandatory
Complex element content with choices	Single schema definition per row

Unclear how to map semistructured XML to well-structured RDBMS

Storing Complete XML Documents

Documents are stored as files or as CLOBs (Character Large Object) in the database

- Very ineffective to answer queries (scan complete document collections for answers)
- Build additional index structures:
 - Inverted File Index (where does a term appear)
 - Inverted File plus structure index to answer structural queries

Inverted File Index

Store appearance of terms in documents (like index of a book)

alphabet	→	(15,42);(26,186);(31,86)
database	→	(41,10)
index	→	(15,76);(51,164);(76,641);(81,64)
information	→	(16,76)
retrieval	→	(16,88)
semistructured	→	(5,61);(15,174);(25,41)
XML	→	(1,108);(2,65);(15,741);(21,421)
XPath	→	(5,90) (21,301)

(document-ID, position in the doc)

Answer queries like „xml and index“, „information near retrieval“

But: not suitable for evaluating path expressions

Structure Index

- Compact representation of structural information for evaluating path expressions

Element	(DocID,Pos)	Order	Parent
dblp	(1,1)	1	
article	(1,10)	1	(1,1)
article	(1,251)	2	(1,1)
author	(1,21)	1	(1,10)
author	(1,64)	2	(1,10)

Used in combination with inverted file to answer queries like
//article[CONTAINS(author, "Weikum")]

But: only for XML trees

descendant-or-self axis is hard to evaluate

result must be constructed from original document

Generic Relational Tables for XML

Store both structure and content in relational tables

Elements

DocID	ID	ElementName	Type	Value	Order	Parent
1	1	dblp			1	
1	2	article			1	1
1	3	author	String	P. Muth	1	2
1	4	author	String	G. Weikum	2	2
1	5	article			2	1

Attributes

ElementDocID	ElementID	AttributeName	Type	Value
1	2	key	ID	MuthW00

Mapping XPath to SQL

Queries (e.g. in XPath) are mapped to equivalent SQL queries on the generic tables:

XPath: //article[author="G. Weikum"]

```
SELECT e1.DocID, e1.ID  
FROM Elements e1, Elements e2  
WHERE e1.ElementName="article" AND  
       e2.ElementName="author" AND  
       e2.Value="G. Weikum" AND  
       e2.parent=e1.ID AND  
       e1.DocID=e2.DocID;
```

Generic Table Summary

- No DTD/Schema required to store XML documents in a relational database
- Result documents are constructed from the SQL result
- descendant-or-self axis is hard to evaluate (cannot be done directly in SQL!)
- Supports only XML trees (without links)

Generic Representation of Graphs

- Only one table for all kinds of nodes in the graph
- One additional table for edges of the graph
- Ignore order (use only for data-centric documents)

Nodes

DocID	ID	NodeName	Type	Value
1	1	dblp	Element	
1	2	article	Element	
1	3	author	Element	
1	4	key	Attribute	MuthW00
1	5	author	Text	P. Muth

Edges

SourceDocID	SourceID	TargetDocID	TargetID	Type
1	1	1	2	containment
1	3	1	4	attribute

...

DTD-Based Schema Generation (1)

Exploit DTD information to derive table definitions

Simple Example (just **#PCDATA** subelements):

```
<!ELEMENT article ((author)+, title, journal, year)>
```

```
<!ATTLIST article key ID #REQUIRED>
```

```
<!ELEMENT title (#PCDATA)> ...
```

1. Generate a table for the root element **article**

article					
DocID	ID	title	journal	year	key

2. Add DocumentID plus automatically maintained ID to the table

3. Add a column for each **#PCDATA** subelement that occurs at most once

4. Add a column for each attribute

DTD-Based Schema Generation (2)

Subelements that occur multiple times are moved to separate relations linked with foreign keys

Simple Example (ctd.):

```
<!ELEMENT article ((author)+, title, journal, year)>
```

```
<!ATTLIST article key ID #REQUIRED>
```

```
<!ELEMENT title (#PCDATA)>
```

```
<!ELEMENT author (firstname, lastname)>
```

article					
DocID	ID	title	journal	year	key

author					
pDocID	pID	DocID	ID	firstname	lastname

Foreign key

**ID for this
element**

Generation for Complex Content

Complex Content is

- inlined, if it occurs at most once
- stored in an external table, if it may occur more often

Example:

```
<!ELEMENT dblp (meta,(article)*)>
```

```
<!ELEMENT meta (version,lastchange)>
```

```
<!ELEMENT article (author+,title,journal,year)>
```

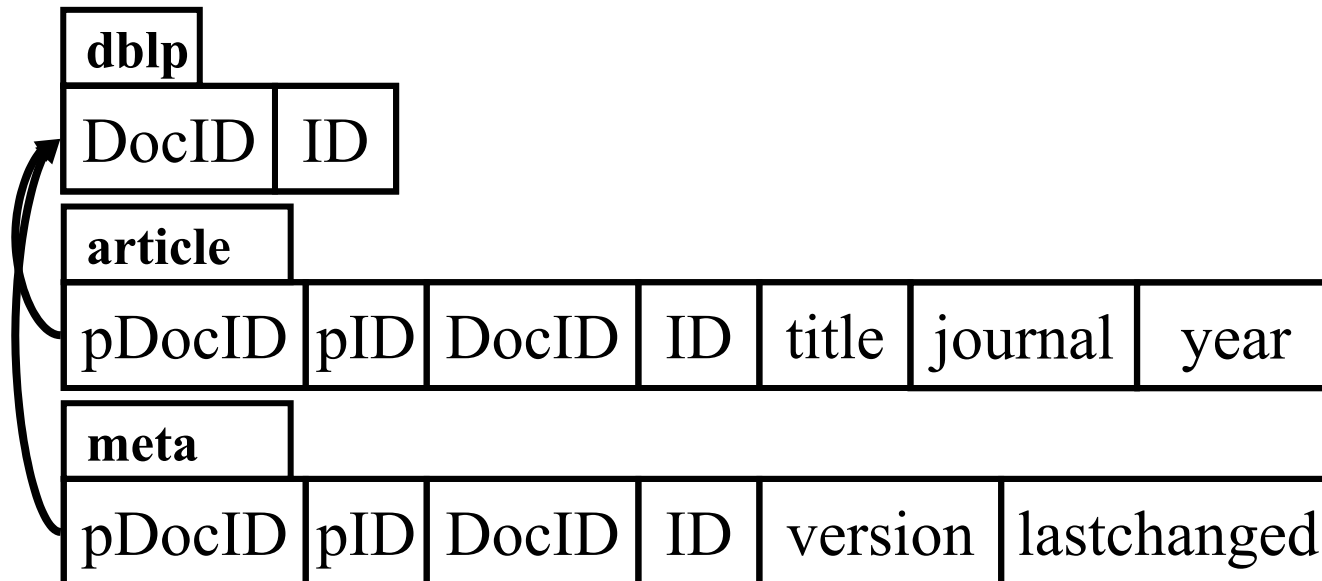
dblp							
DocID		ID	version	lastchanged			
article							
pDocID	pID	DocID	ID	title	journal	year	
author							
pDocID	pID	DocID	ID	firstname	lastname		

Alternative for Complex Content

Complex Content can also always be stored externally and point to its parent relation, regardless of its occurrence
(Makes creation easier, but query evaluation less efficient)

Example:

```
<!ELEMENT dblp (meta,(article)*)>  
<!ELEMENT meta (version,lastchange)>  
<!ELEMENT article (title,journal,year)>
```



Problem 1: Alternatives

Which schema should we generate for

<!ELEMENT vehicle (car|bike|truck|ship)>

Solution 1: encode all alternatives into one relation

⇒ many empty columns, not space-effective

Solution 2: store only the alternative that is used in an external table and use foreign keys for linkage

⇒ saves space, but requires more time to evaluate queries

Problem 2: Recursions

Problematic Scenario:

<!ELEMENT E1 (A, B, E2)>

<!ELEMENT E2 (C, D, E1?)>

Solution:

Break recursion by storing information externally and linking via foreign keys (as if **E1** occurred multiple times)

Problem 3: ANY

What is a good database schema for

<!ELEMENT description ANY>

⇒ Cannot be efficiently converted to a database schema

⇒ Store complete content of this element as XML (using a CLOB)

More on Schema Generation

- XPath queries can be mapped to SQL queries (typically containing many joins)
- Very (space and time) efficient if XML is well structured (data-centric)
- Not so efficient if XML is unstructured (document-centric)
- Can be extended to automatically decide which children to inline (based on query statistics: children that are often used are likely to be inlined)

18.2 Indexing XML

Problem: How to evaluate an XPath expression like **`//article/author[name="Weikum"]`**

Two options:

- Traverse the complete XML graph and search for matching subgraphs (but: *very inefficient* for large graphs and small result sets)
- Maintain appropriate index structures to speed up query evaluation; two kinds:
 - Indexes on the content of elements and value of attributes
 - Indexes on the structure of the XML graph

Content Index (CI)

Retrieval methods of CI:

- Find elements and/or attributes that have a string in their content/value
- Find elements and/or attributes whose content/value satisfies a given template

Result of these methods is a list of nodes (or node IDs) in the XML graph that satisfy the search condition

Common implementation:

- **Inverted Lists** (maybe augmented with additional information like tf- and idf- values) plus efficient string search index on index entries
- Evaluation of arbitrary template expressions can be hard

Structure Index (SI)

SI should support all structural XPath axes:

child, descendant, descendant-or-self, self, parent, ancestor, ancestor-or-self, following, preceding, following-sibling, preceding-sibling, attribute

Most important axes: ($\Rightarrow$ Path Index PI)

- **child (/)**
- **attribute (@)**
- **parent (..)**
- **descendant-or-self (//)**

Interface Method of SI:

Given a node set **N** and an XPath axis **A** as input, compute the nodes reachable from the nodes in **N** via the axis **A** and return the set **N'** of these nodes.

XPath Evaluation

Evaluation of an XPath location step **axis::test[condition]** with input node set **N** (result of the previous location step or root node for the first location step in the location path):

- Compute the result set **N1** by following **axis** from nodes in **N**
- Compute set **N2** of nodes in **N1** that satisfy **test**
- Compute set **N3** of nodes in **N2** that satisfy **condition** (which is again an XPath location path)
- If **condition** has the form **.="string"**
 - Compute (using CI) set **N4** of nodes that satisfy the condition
 - Compute **N3** by intersecting **N2** and **N4**

Using Pre- and Postorder as SI

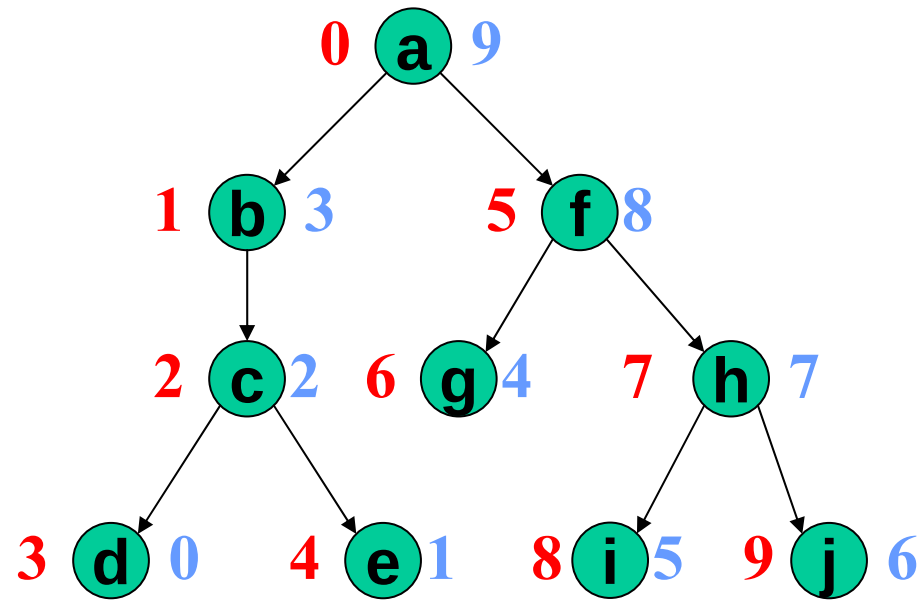
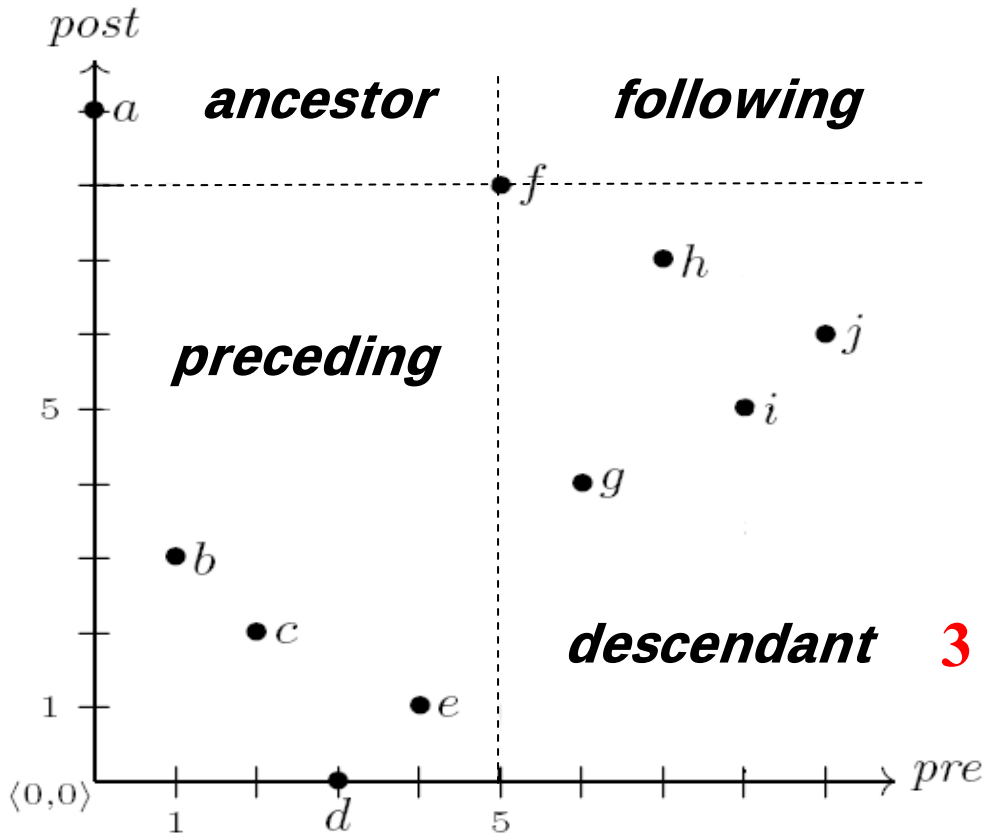
Idea:

- Use two numbering schemes (pre- and postorder) for the nodes in the XML tree
- Compute result of following an XPath axis from a given node by evaluating conditions on pre- and postorder ranks

How to compute pre- and postorder:

- Compute depth-first traversal of XML tree
 - Preorder rank = order in which the traversal enters nodes
 - Postorder rank = order in which the traversal exits nodes
- (see Slides 17-28 and 17-29)

Details on Pre- and Postorder SI



Additionally store preorder rank of the parent

Pre- and Postorder Conditions

Database Schema for storing nodes:

Pre	Post	Par	Tag
Preorder rank	Postorder rank	Parents preorder rank	Stores the element tag or attribute tag

SQL Query conditions for important axes:

Axis	Pre	Post	Par	Tag
child	$(\text{pre}(v), \infty)$	$[0, \text{post}(v))$	$\text{pre}(v)$	*
descendant	$(\text{pre}(v), \infty)$	$[0, \text{post}(v))$	*	*
descendant-or-self	$[\text{pre}(v), \infty)$	$[0, \text{post}(v)]$	*	*
parent	$[\text{par}(v), \text{par}(v)]$	$(\text{post}(v), \infty)$	*	*
ancestor	$[0, \text{pre}(v))$	$(\text{post}(v), \infty)$	*	*
ancestor-or-self	$[0, \text{pre}(v)]$	$[\text{post}(v), \infty)$	*	*
following	$(\text{pre}(v), \infty)$	$(\text{post}(v), \infty)$	*	*
following-sibling	$(\text{pre}(v), \infty)$	$(\text{post}(v), \infty)$	$\text{par}(v)$	*

Pre-/Postorder Summary

Advantages:

- Supports all XPath axes
- Efficiently handles **ancestor-or-self** queries (//)

Problems:

- Does not support links
- Numbering scheme has to be recomputed upon updates
- Cannot compute the distance between nodes (would be interesting for ranking results)

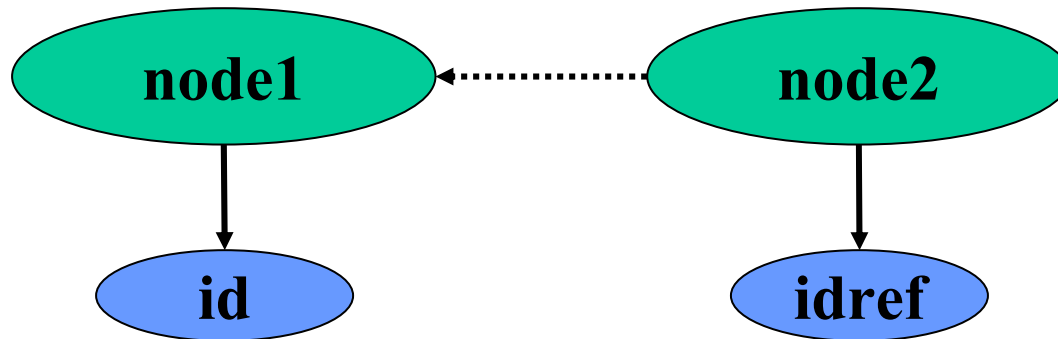
Modelling Links in the XML Graph

For simplicity, we model ID/IDREF-links by one single edge (useful for information retrieval):

```
<node1 id="sample"> ... </node1>
```

...

```
<node2 idref="sample"> ... </node2>
```



APEX

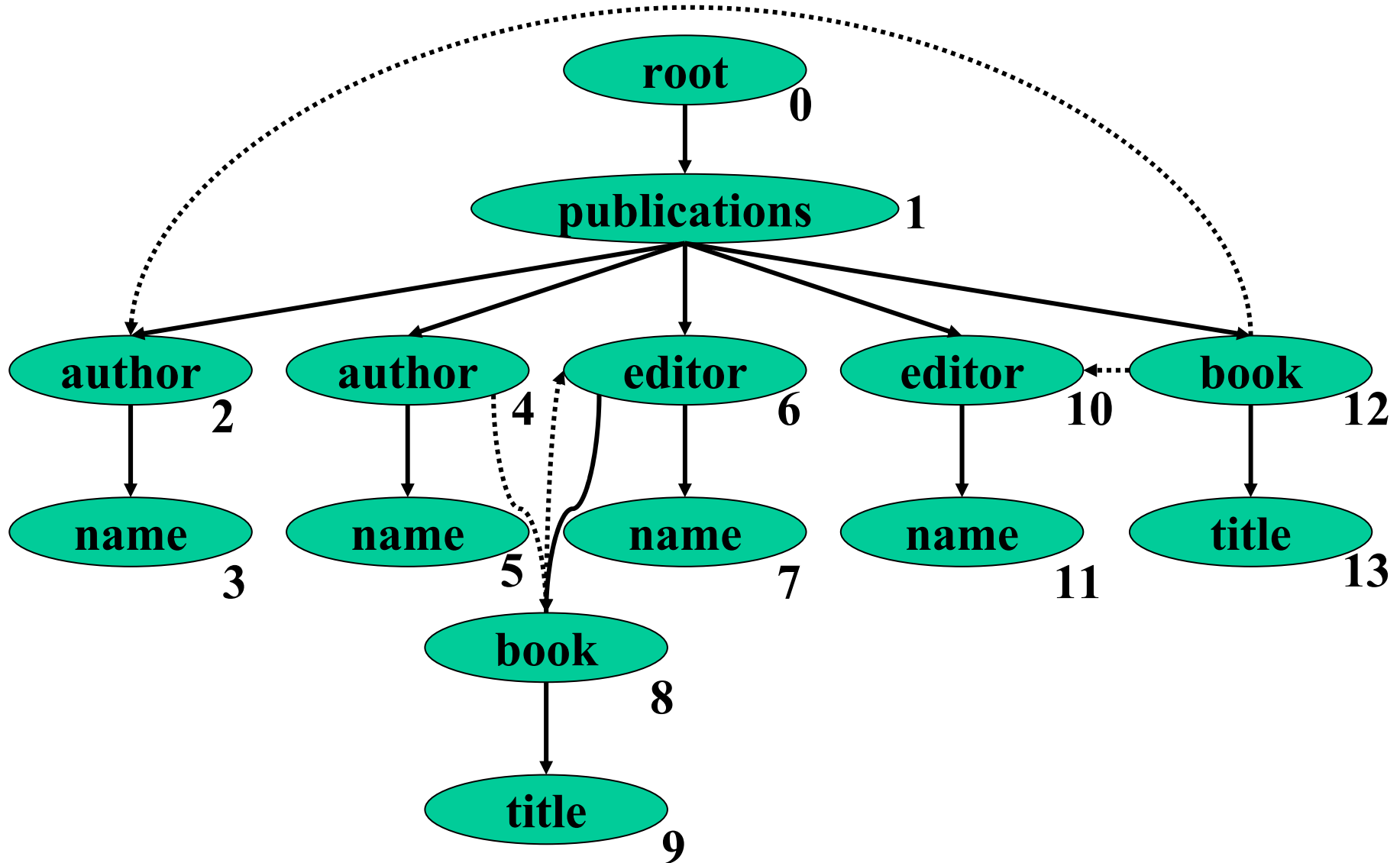
APEX: Adaptive Path Index for XML Data

Components:

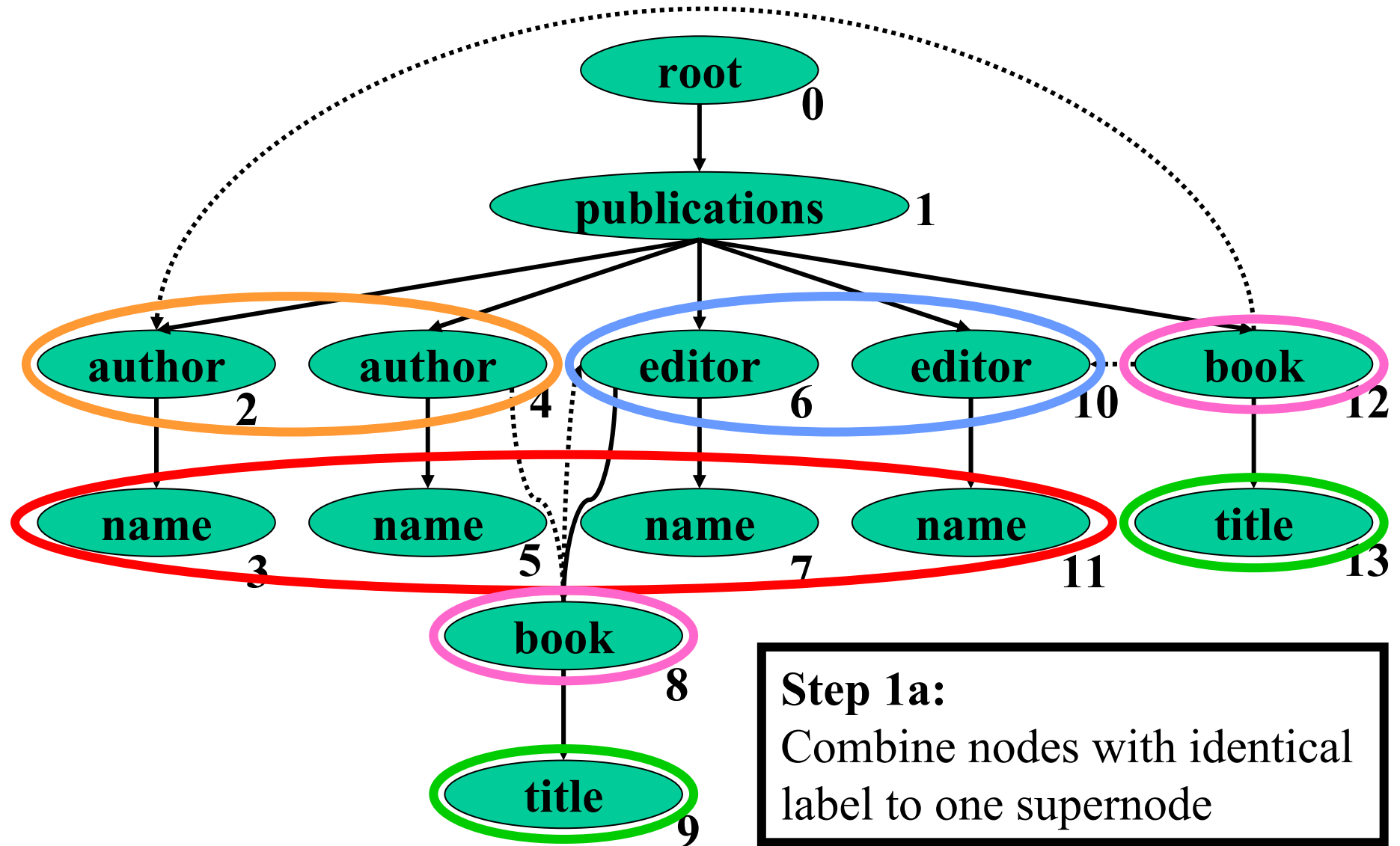
- Structural Summary: Compactly store information about connections in-between node types
- Collect information about workload
- Refine structural summary to efficiently support often used subqueries (at the price of possibly higher cost for less frequently used queries)
- Adaptive to changing workloads

We consider an adaption of APEX to our XML data model

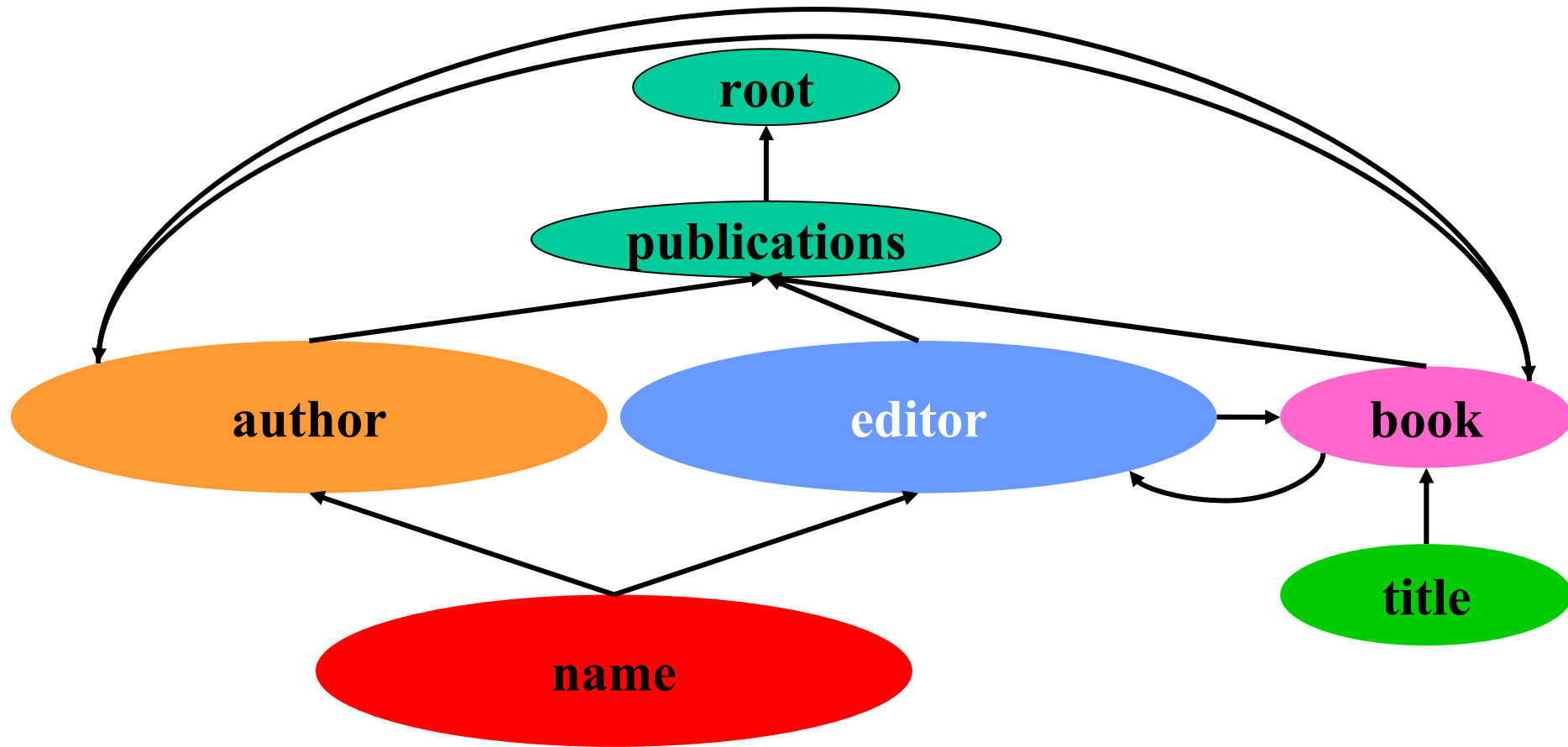
Step 1: Create Structural Summary



Step 1: Create Structural Summary



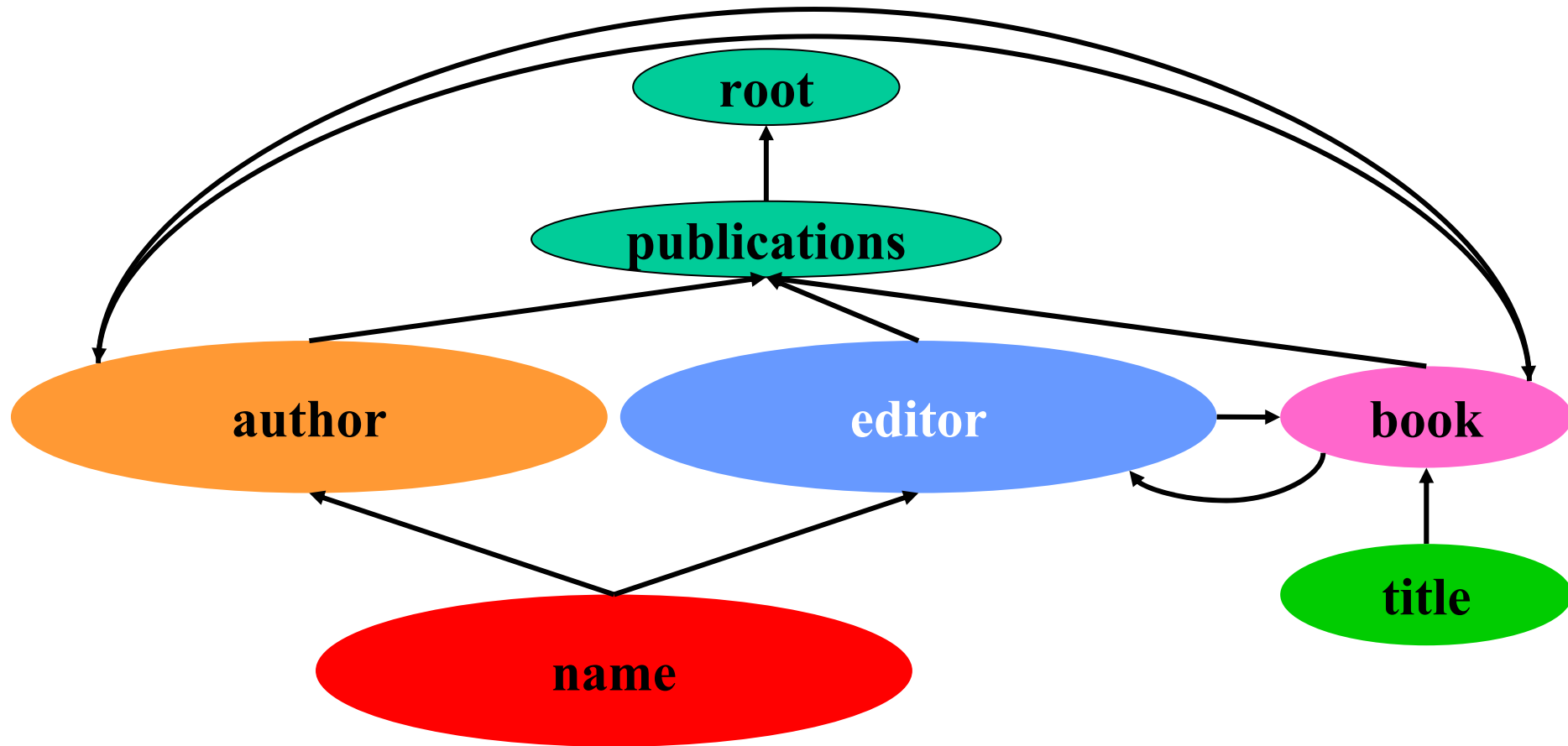
Step 1: Create Structural Summary



Step 1b:

Add edges between
supernodes in reverse order

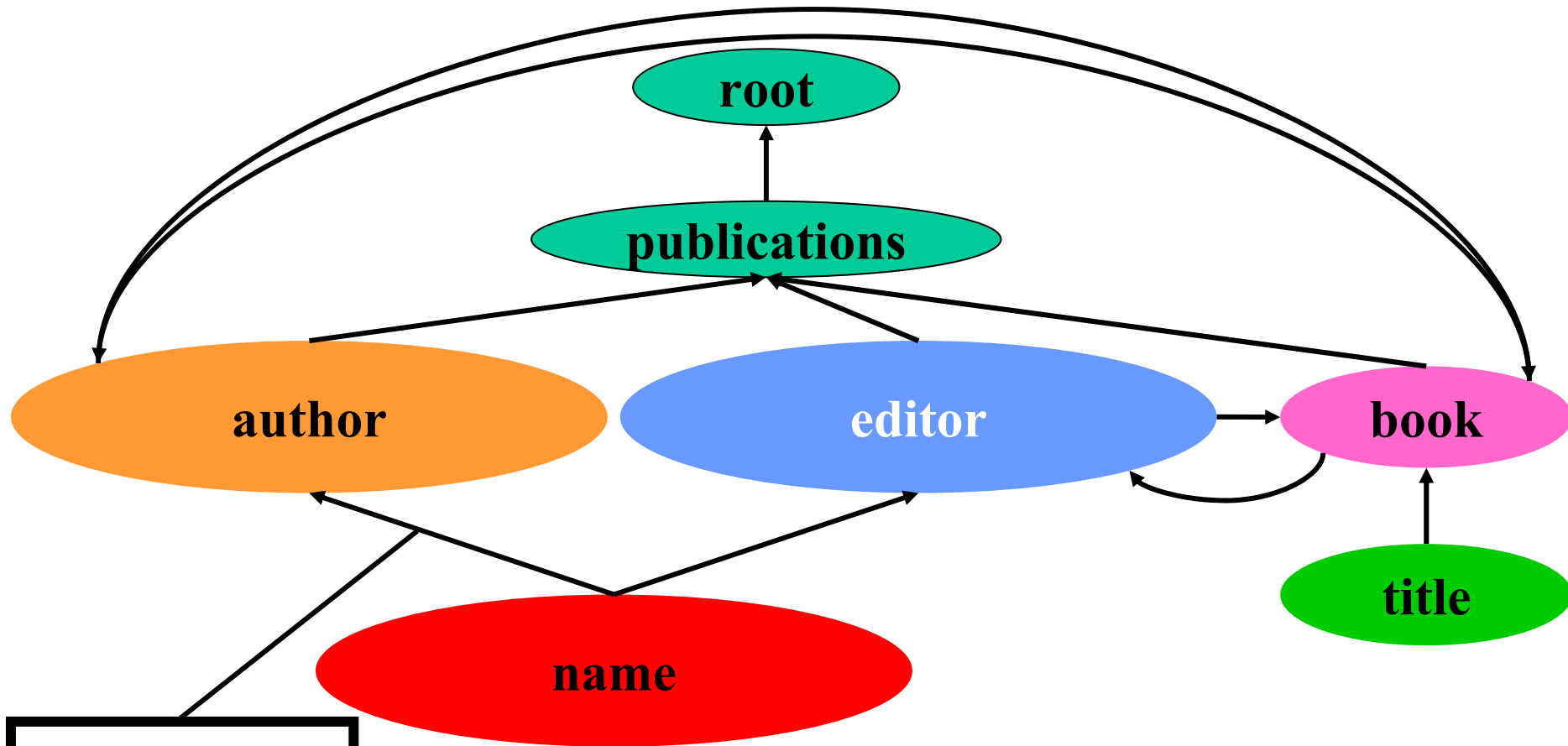
Step 1: Create Structural Summary



instances:
3,5,7,11

Step 1c:
augment supernodes with
sorted instance information

Step 1: Create Structural Summary



connections:

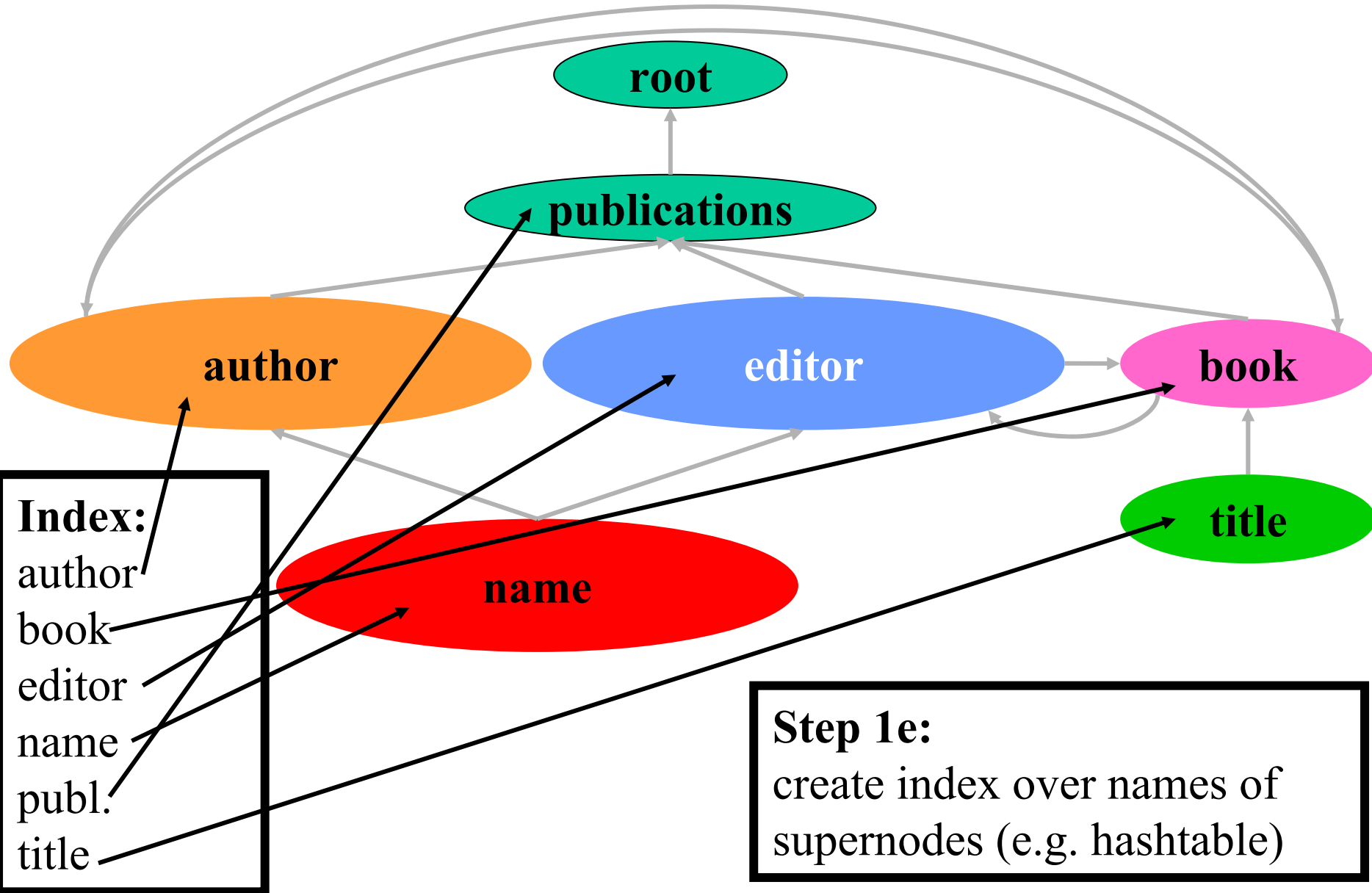
(2,3)

(4,5)

Step 1d:

augment edges with sorted
connection information

Step 1: Create Structural Summary



Evaluation with Structural Summary

- **//name:**
search for “name” in supernode name index, go to supernode for “name“, extract instance list
- **//author/name:**
get supernode for “name“ as before, follow edge to “author“ supernode, get instances from edge annotation
- **author/name:** (starting at one specific author node)
get supernode for “name“ as before, follow edge to “author“ supernode, get instance for specified starting node from edge annotation

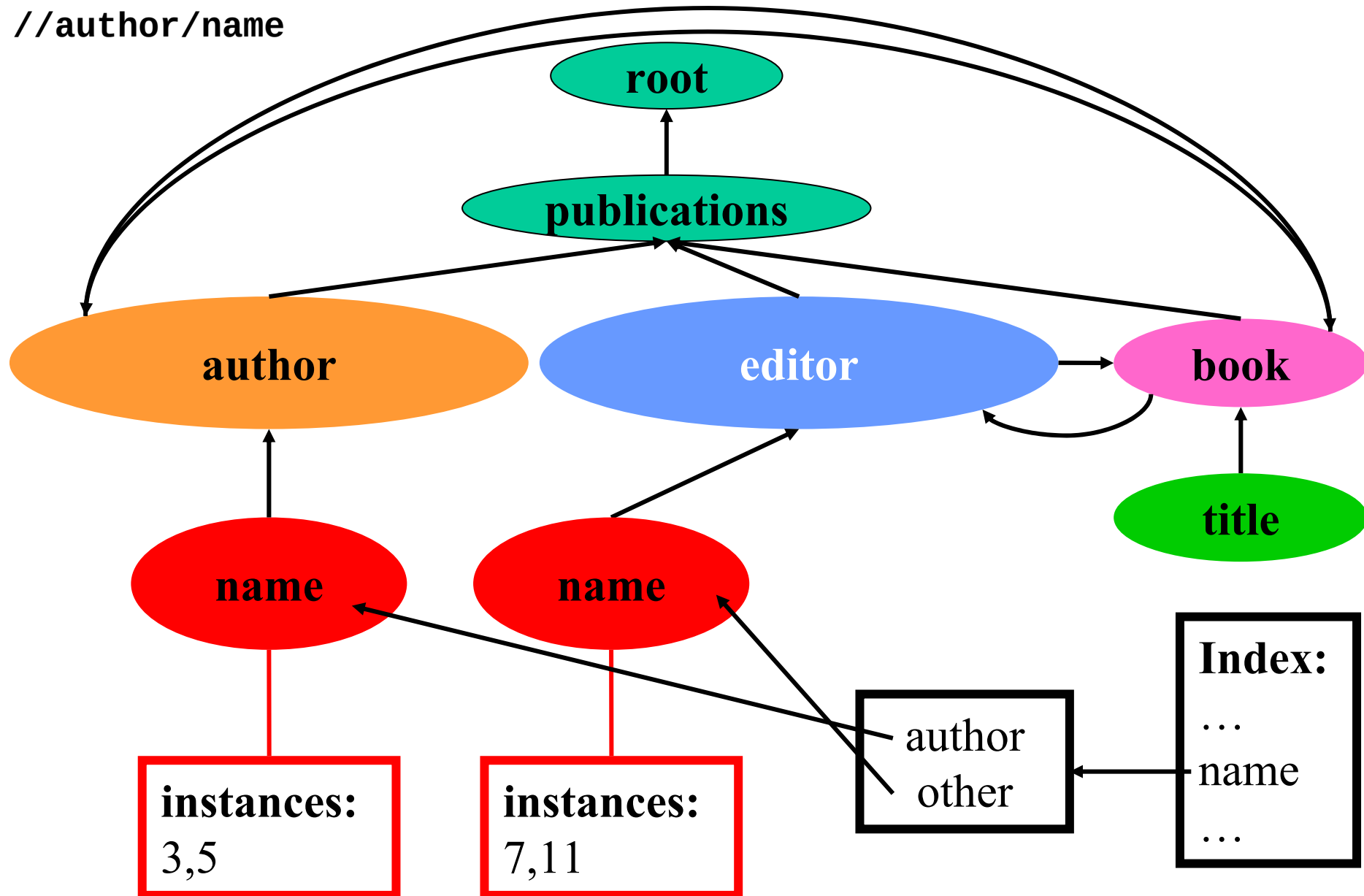
Structural Summary is initial version of APEX‘

Step 2: Collect Workload Info

- Count how often paths appear in the query workload:
10x **//author/name**
3x **//director/title**
2x **//movie/title**
- Update structural summary to efficiently support often used paths (with usage over a fixed threshold):
Split associated supernodes such that often used paths can be evaluated without further evaluation
in our example: **//author/name**

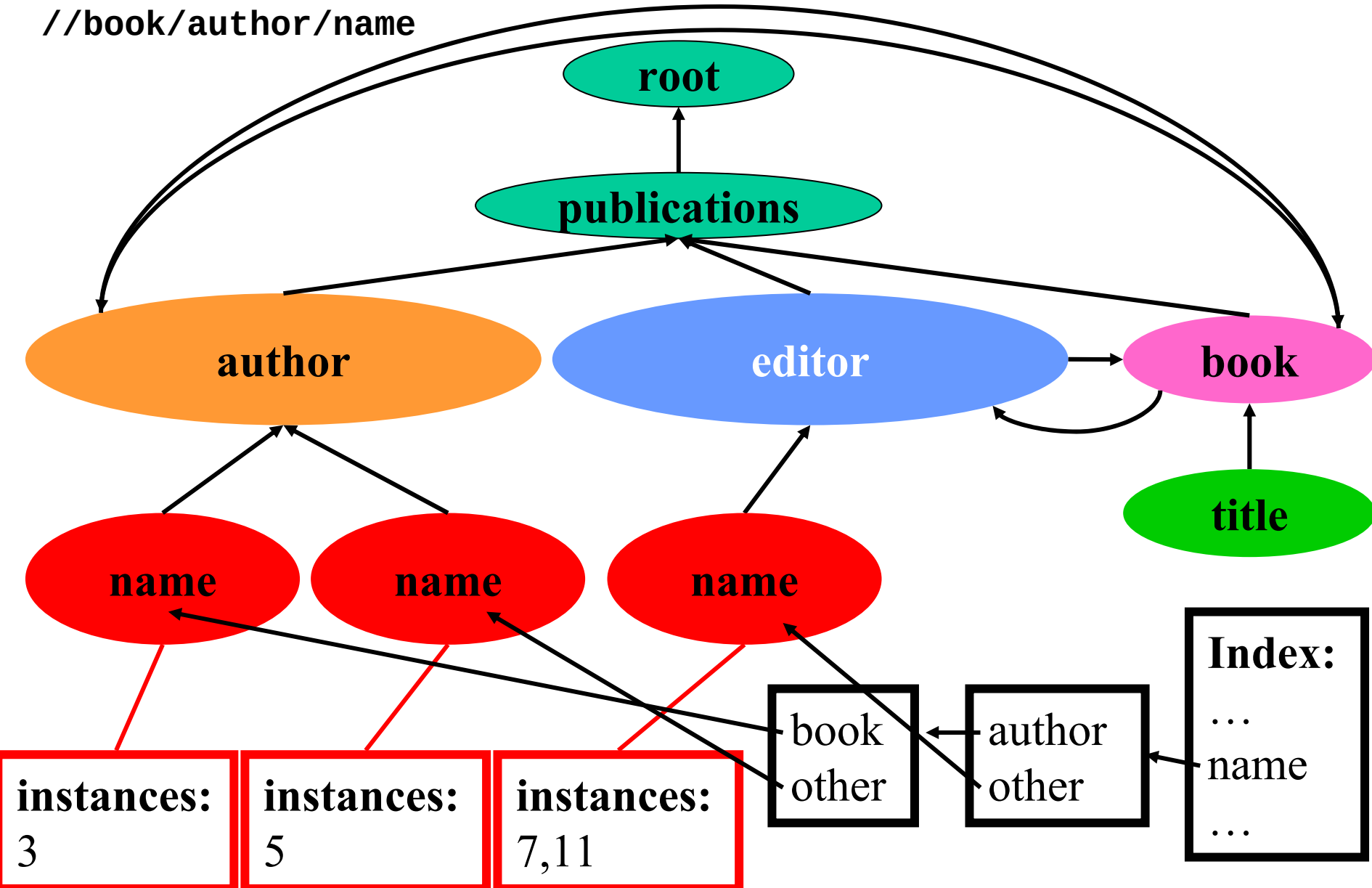
Step 3: Update Structural Summary

//author/name



Most Effective on Longer Paths

//book/author/name



APEX Summary

Advantages:

- Adaptive to workload changes
- Efficiently handles **ancestor-or-self** queries (//)
- Efficiently handles often used queries
- Supports links

Problems:

- How to set threshold? (Guru required!)
- Performance under structural updates unclear
- Performs less efficiently for less frequently used queries
- Performs poorly for queries like **a//b**

Indexing the XML-based Web

How to index the XML-based Web of the future?

- Too large for existing index structures ($>10^{10}$ pages)
- Inter-document queries (following XLinks)
- Optimized for IR (non-boolean queries)
- Distance-based ranking for `//` expressions
- Varying structure and element naming
- First results should be returned very quickly ($\Rightarrow$ architecture should be pipelined)

“Normal“ queries (like this XXL-query) are no longer sufficient:

SELECT \$p FROM INDEX

WHERE publication AS \$p AND \$p.author AS \$a

AND \$a LIKE “Weikum” AND \$p.content LIKE “XML”

AND \$p.citation.author LIKE “Jim Gray”

Indexing the XML-based Web

How to index the XML-based Web of the future?

- Too large for existing index structures ($>10^{10}$ pages)
- Inter-document queries (following XLinks)
- Optimized for IR (non-boolean queries)
- Distance-based ranking for // expressions
- Varying structure and element naming
- First results should be returned very quickly ($\Rightarrow$ architecture should be pipelined)

Automatically add structural vagueness:

```
SELECT $p FROM INDEX
```

```
WHERE #.~publication AS $p AND $p.#.~author AS $a  
AND $a LIKE "Weikum" AND $p.#.~content LIKE "XML"  
AND $p.#.~citation.#.~author LIKE "Jim Gray"
```

Connection Index

Evaluation of path expressions with structural vagueness requires support for inner path wildcards (`//` or `#`)

Basic approach:

- Compute transitive closure **C** of XML graph (containing all the *connections* between nodes)
- Find compact representation **C'** of **C**
- Evaluate wildcard expressions using **C'**

Computing the Transitive Closure

Given a directed graph $G=(V,E)$, the transitive closure $C=(V,T)$ can be computed in $O(|V|^3)$ using the Warshal algorithm or in $O(|T|*\max(\text{outdeg}))$ using the following alg:

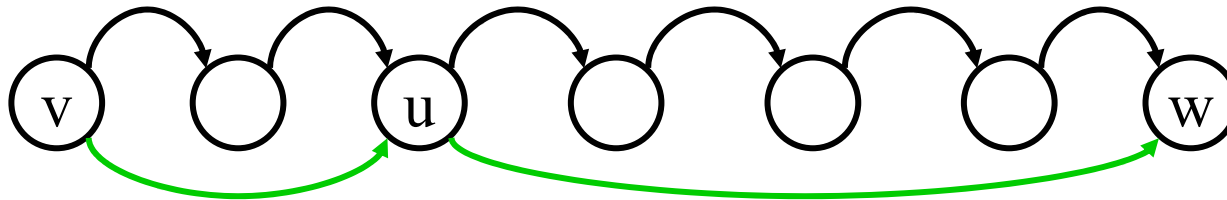
```
Set    $T := E \cup \{(v, v) \mid v \in V\};$   
Set  $\Delta T := T;$   
while  $(\Delta T \neq \emptyset)$   
{ Set  $T' := \emptyset;$   
  for all  $p := (v, u)$  in  $\Delta T$   
    for all  $e := (u, w)$  in  $E$   
      if  $(v, w) \notin T$  then  $T' := T' \cup \{(v, w)\};$   
   $T := T \cup T';$   
   $\Delta T := T';$   
}
```

Idea:

In round i of the loop,
compute all connections over
paths of length $i+1$ by
extending existing connections
over paths of length i

Compact Representation

- For each node v , maintain two sets of labels (which are node names): $L_{\text{in}}(v)$ and $L_{\text{out}}(v)$
- For each connection (v, w) ,
 - choose a node u on the path from v to w
 - add u to $L_{\text{out}}(v)$ and to $L_{\text{in}}(w)$
- Then $(v, w) \in T \Leftrightarrow L_{\text{out}}(v) \cap L_{\text{in}}(w) \neq \emptyset$



Two-hop Cover of T

- Minimize the sum of the label sizes
(NP-complete $\Rightarrow$ approximation required)

Query Evaluation with Two-Hop

- **//author:**
search for “author“ among all nodes (can be done efficiently using index on node names)
- **//author/name:**
get nodes for “author“ as before, go to connected nodes with distance 1 and look for “name“ nodes (distance information may be included in the index)
- **author/name:** (starting at one specific author node)
as before, for one specific author node
- **//author//editor:**
get “author“ nodes and the union of their L_{out} sets, get “editor“ nodes and the union of their L_{in} sets, compute intersection

Web-Scale XML Indexing

Existing indexing strategies are not usable for Web-scale data:

- Hardly support for inner-query path wildcards (necessary for expressing structural vagueness)
- Documents are very heterogeneous (no single indexing technique can be the best for all documents)
- Sometimes no support for intra-document links
- No explicit support for inter-document links (treat links as if they were ordinary containment edges) $\Rightarrow$ few, very large documents to index
- Space, build and execution time for large-scale XML data is unacceptable

FLIX: Index Framework for XML

Current Research Project: *FLIX* (Framework for Indexing Large Collections of Interlinked XML Documents)

Basic Principles:

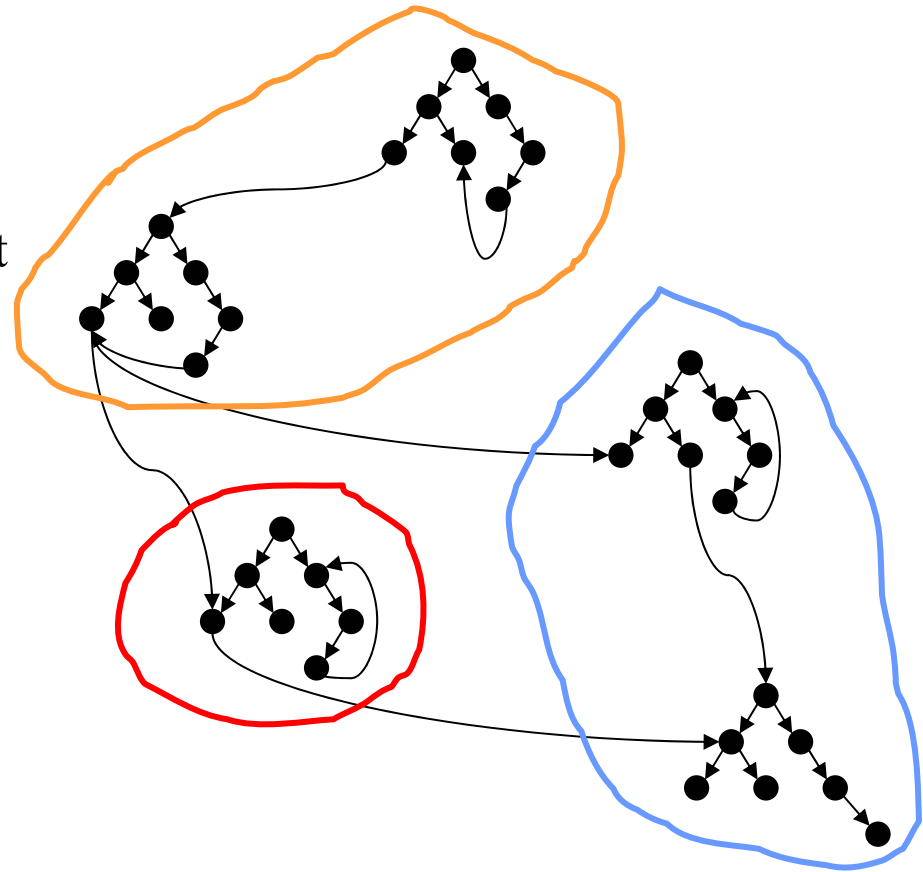
- Automatically construct fragments of the set of interlinked XML documents
- Choose an „optimal“ connection index for each meta document (depending on workload and available space)
- Incrementally build results for path queries with inner path wildcards in approximate order of distance ($\Rightarrow$ most relevant results first)

FLIX Example

Step 1:

Choose meta documents

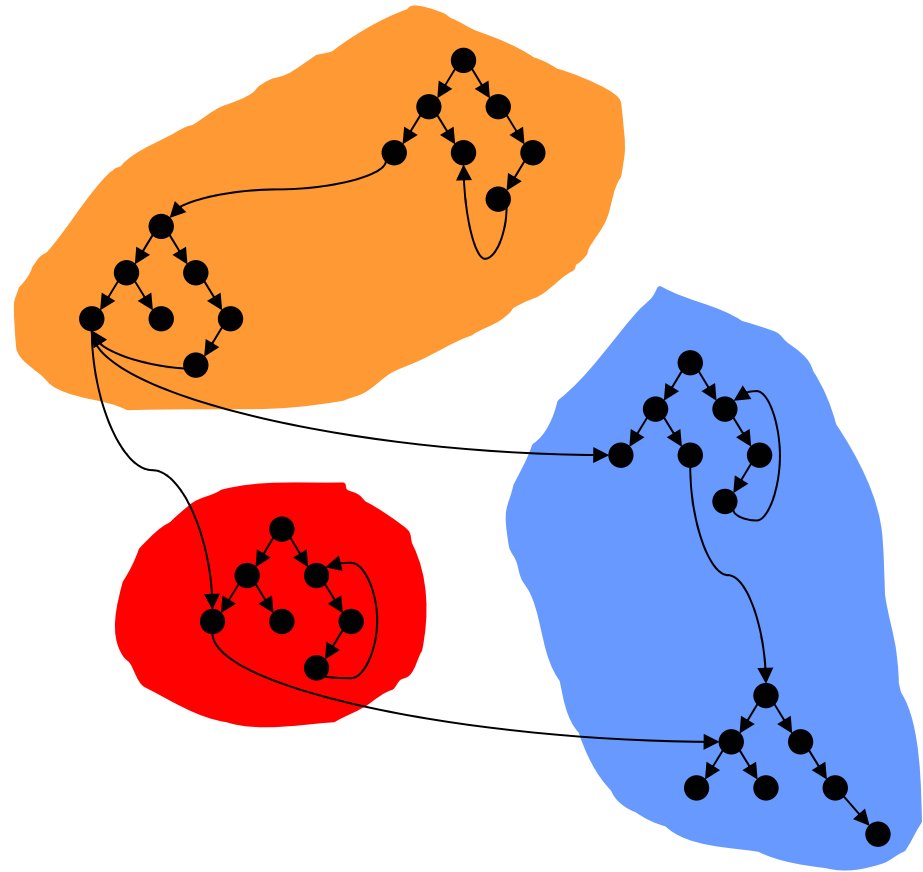
- using a graph-theoretic measure
- by identifying units with coherent information
- by analyzing query workload



FLIX Example

Step 2:

Choose and build optimal connection index for each meta document

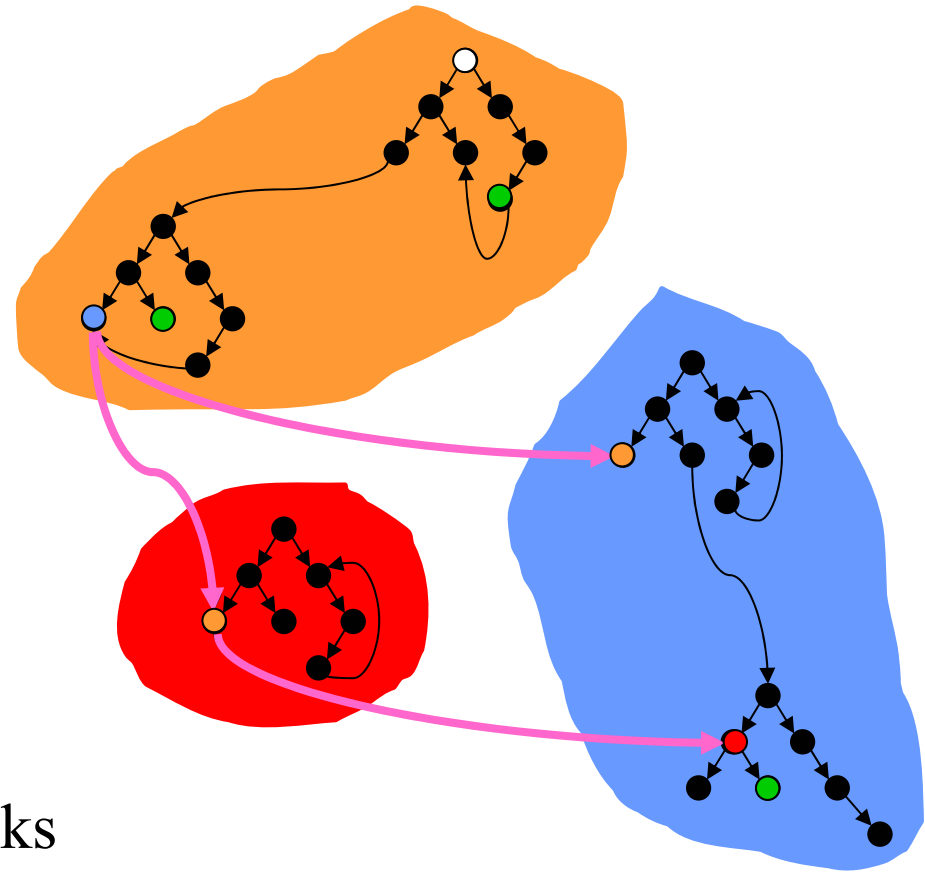


FLIX Example

Step 3:

Incrementally evaluate query
with inner path wildcard
//a//b

- Find set of instances of **a**
- Evaluate query within resulting meta documents
- Find elements with outgoing links in resulting meta documents and follow the links in ascending distance
- Evaluate rest of query (**//b**) in target meta document



18.3 XML Compression

XML is not very storage efficient:

- contains a lot of redundancy (increases readability for human users):
 - opening and closing tags
 - element and attribute names
- character-based:
 - 4294967295 (10 bytes) vs. 0xFFFFFFFF (4 bytes)

⇒ Compression can save much storage space

Text-based compressors (like gzip): ~80-90% compr. ratio

Using information about XML structure gives even better compression ratios

Example: Web Log Data

ASCII File 15.9 MB (gzipped 1.6MB):

```
202.239.238.16|GET / HTTP/1.0|text/html|200|1997/10/01-00:00:02|-|4478|-|http://www.net.jp/|Mozilla/3.1
```

XML-ized inflates to 24.2 MB (gzipped 2.1MB):

```
<apache:entry>
  <apache:host> 202.239.238.16 </apache:host>
  <apache:requestLine> GET / HTTP/1.0 </apache:requestLine>
  <apache:contentType> text/html </apache:contentType>
  <apache:statusCode> 200</apache:statusCode>
  <apache:date> 1997/10/01-00:00:02</apache:date>
  <apache:byteCount> 4478</apache:byteCount>
  <apache:referer> http://www.net.jp/ </apache:referer>
  <apache:userAgent> Mozilla/3.1[$ja$]$(I)</apache:userAgent>
</apache:entry>
```

XMill

- Specialized compressor for XML data
 - Utilizes three basic principles:
 - Compress the structure separately from the data
 - Group the data values per element type
 - Apply semantic (specialized) compressors
- ⇒ Even better compression ratios (optimality of compression can be proven)

(Slides for XMill partly taken from the original presentation at SIGMOD 2000)

XMill Principles (1)

Compress the structure separately from the data:

gzip Structure

```
<apache:entry>  
  <apache:host> </apache:host>  
  
  ...  
</apache:entry>
```

+

gzip Data

```
202.239.238.16  
GET / HTTP/1.0  
text/html  
200  
...
```

=1.75MB

XMill Principles (2)

Group the data values per element type:

gzip Structure

```
<apache:entry>  
...  
</apache:entry>
```

+

gzip host

```
202.23.23.16  
224.42.24.55  
...
```

+

gzip url

```
GET / HTTP/1.0  
GET / HTTP/1.1  
...
```

=1.33MB

XMill Principles (3)

Apply semantic (specialized) compressors for known types:

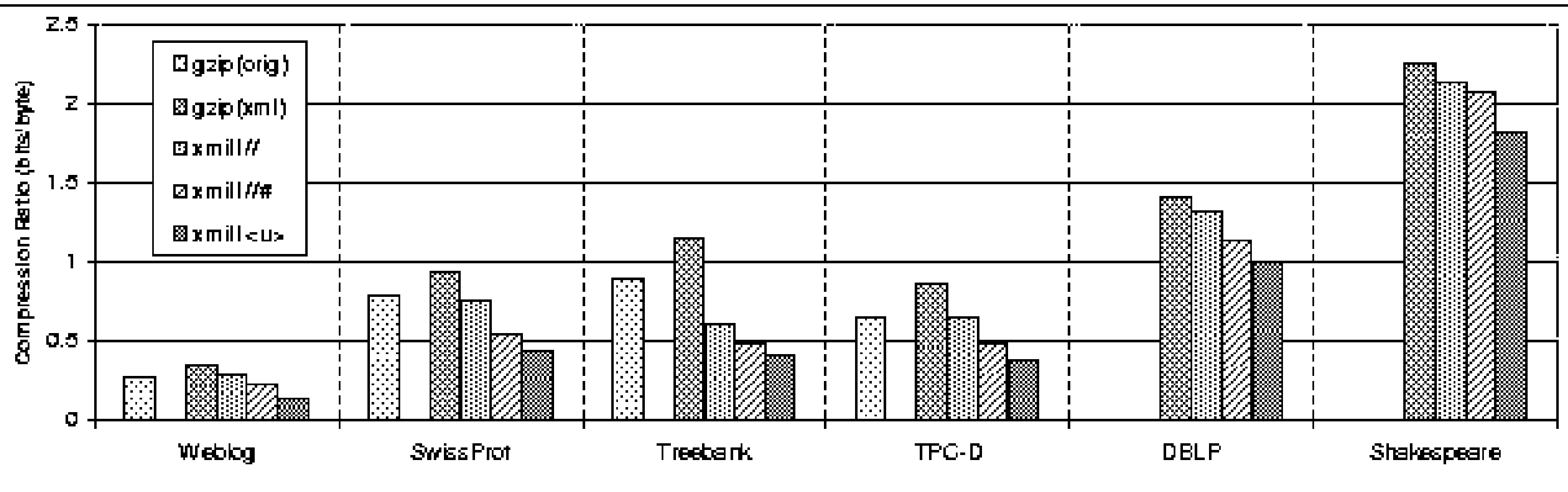
gzip Structure + gzip c1(host) + gzip c2(url) + ... = 0.82MB

Examples:

- 8, 16, 32-bit integer encoding (signed/unsigned)
- differential compressing (e.g. 1999, 1995, 2001, 2000, 1995, ...)
- compress lists, records (e.g. 104.32.23.1 → 4 bytes)

semantic compressor selection by user and/or schema information

XMill Compression Factors



- gzip (orig) works on the original data
- gzip (xml) works on the XMLified data
- XMill // separates structure and data
- XMill // # additionally groups the data by element names
- XMill <u> additionally applies type-specific compressors

18.4 Native XML Databases

Key concepts of native XML Databases:

- (Logically) store XML documents without converting them
- Logical unit of information is a single XML Document
- Efficient query evaluation on the XML documents (with index structures for structure and content)
- Update, insert and delete XML documents
- All „standard“ database features:
 - Transactions
 - Distribution, replication
 - Parallel server
 - Multiuser
 - Security
 - Query optimization, Performance
 - ...

Native XML vs. XML-Enabled RDMBS

XML-Enabled RDBMS:

- XML mapped to relational data model
- Optimizer tuned for relational algebra (projection, join, selection)
- Evaluation of query may require numerous SQL joins

Native XML-DB:

- XML stored natively
- Optimizer tuned for tree algebra (e.g., tree traversal)
- Evaluation of query using specific index structures

XML Schema Design

- Basic unit is one XML document
- What should go into a document:
 - Document = individual thing, event, ...(about a product, report, order, measurement, treatment)
 - Document collects related information (about a project, process, career, ...)
- Principle: Store exactly the information within a single document that is most often requested together
- Refer to less frequently used information via XLinks

XML Databases: Tamino

- **Tamino:** Transactional Architecture for Managing Internet Objects
- Available since 1999 from Software AG
- Evaluation version available (252MB!)
- Best-known native XML database
- Key concepts:
 - Extension of XPath for querying
 - Updates, Inserts, Deletes
 - Built-in Extensions for Information Retrieval
 - Support for optional validation against XML Schema
 - Transparent integration of non-XML data from other sources (automatic mapping to XML view of the data)
 - Extensible Architecture

Transaction Support in Tamino

- A transaction consists of a set of operations (queries and/or updates) that form a unit
- Transactional principles (ACID):
 - Atomicity (all-or-nothing)
 - Consistency (maintain database consistency)
 - Isolation (run transaction as if it was alone in the DB)
 - Durability (data must survive failures)
- Transaction support in Tamino:
 - A+D using logging (+restoring information from logs) plus two-phase commit for distributed transactions (Windows only)
 - I using document-level locks ($\Rightarrow$ potential performance bottleneck)
 - C by checking modified documents against constraints

Other XML Databases: Timber

- Research prototype from Uni Michigan
 - Applies existing backend system *Shore* for disk management, buffering, concurrency control
 - XML documents are stored in fragments, where each fragment roughly corresponds to a DOM node
 - Supports content indexes on elements and attributes, variant of pre/post-ordering as structure index
 - Uses tree algebra for optimizing XQuery expressions (focus of the project)
 - Updates of XML documents supported (but with problems with the numbering when too many updates occur)

Other XML Databases: Natix

- Research prototype from Uni Mannheim
 - XML documents are split into fragments in a clever way (minimizing reconstruction effort and maximizing query performance) $\Rightarrow$ one of the focusses of the project
 - Sophisticated transaction management techniques (e.g., element-level locking, timestamps)
 - Interfaces to Java, C++ (using DOM and SAX), WebDAV, HTTP and Filesystem
 - Inverted File as content index plus variant of pre/postorder scheme as structure index

XML Databases: Xindice

- Open-source project from Apache
 - Available from <http://xml.apache.org>
 - Source code available (but not well documented)
 - Optimized for many, small XML documents (max document size: 5MB)
 - Implements XPath and update operations
 - API available
 - Supports only content indexes, no structure index
⇒ poor query performance
 - May be a starting point for using XML databases

Sources and Further Literature for Part 18

- M. Klettke, H. Meyer: **Speicherung von XML-Dokumenten – eine Klassifikation**. Datenbank-Spektrum 3(5), 2003.
- H. Schöning: **XML und Datenbanken**. Hanser, 2003.
- D. Florescu, D. Kossmann: **Storing and Querying XML Data using an RDBMS**. IEEE Data Engineering Bulletin 22(3), 1999.
- G. Kappel et al.: **X-Ray – Towards Integrating XML and Relational Database Systems**. Technical Report, Uni Linz, 2000.
- T. Grust: **Accelerating XPath Location Steps**. SIGMOD Conference, 2002.
- C. Chun et al.: **APEX: An Adaptive Path Index for XML data**. SIGMOD Conference, 2002.
- Q. Chen et al.: **D(K)-Index: An Adaptive Structural Summary for Graph-Structured Data**. SIGMOD Conference, 2003.
- H. Liefke and D. Suciu: **XMILL: An Efficient Compressor for XML Data**. SIGMOD Conference, 2000.
- J.-K. Min et al.: **XPRESS: A Queriable Compression for XML Data**. SIGMOD Conference, 2003.

Sources and Further Literature for Part 18

- Tamino: <http://www.softwareag.com/tamino/>
- Timber: <http://www.eecs.umich.edu/db/timber/>
- **Th. Fiebig et al.** : Natix: A Technology Overview. NODe 2002, Springer LNCS 2593.
- Xindice: <http://xml.apache.org/xindice/>

Kapitel 2: Suchmaschinentechnologie für Intranets und das Web

- 2.1 Information-Retrieval-Systeme
- 2.2 Web-Crawling und Indexierung
- 2.3 Vektorraummodell für IR mit Ranking
- 2.4 Anfrageausführung mit Ranking
- 2.5 Grundlagen aus der Linearen Algebra
- 2.6 Latent Semantic Indexing

Informationssysteme SS 2004

2-1

2.1 Information-Retrieval-Systeme

Information Retrieval (IR) ist die Technologie zum Suchen in Kollektionen (Korpora, Intranets, Web) schwach strukturierter Dokumente: Text, HTML, XML, ...

Darunter fällt auch:

- Text- und Strukturanalyse
- Inhaltserschließung und -repräsentation
- Gruppierung und Klassifikation
- Zusammenfassung
- Filtern und Personalisieren (z.B. von Nachrichten-„Feeds“)
- „Routing“ (Metasuche)

*Globales Ziel:
Informationsbedürfnisse befriedigen - und dabei
Beseitigung des Engpasses (teurer) intellektueller Zeit !*

Informationssysteme SS 2004

2-2

Schnittstellen von IR-Systemen

- **Ausgabe:**
 - Menge von Dokumenten, die Suchstring(s) enthalten: **Freitextsuche**
 - Menge inhaltlich relevanter Dokumente: **Inhaltssuche**
 - ungeordnete Menge: **Boolesches Retrieval**
 - nach Relevanz absteigend sortierte Rangliste: **Ranked Retrieval** (Ähnlichkeitssuche)
- **Eingabe:**
 - Keywords (positiv/negativ) (plus Phrasen, ganze Sätze)
 - (Boolesche) Ausdrücke über Keyword-Bedingungen
 - Strukturbedingungen (z.B. Tags, Links)
 - ontologisch basierte Bedingungen
 - Suchsprache (z.B. SQL mit interMedia)

Informationssysteme SS 2004

2-3

Beispiel: SQL in Oracle/Text

Beispielanfrage: `Select URL, Content, Year From Docs
Where Year > 1995 And Category Like '%drama%'
And Contains (Content, 'BT(king)', 1) > 10
And Contains (Content, 'SYN(traitor)NT(traitor)', 2) > 0
Order By Score(1)*Score(2)`

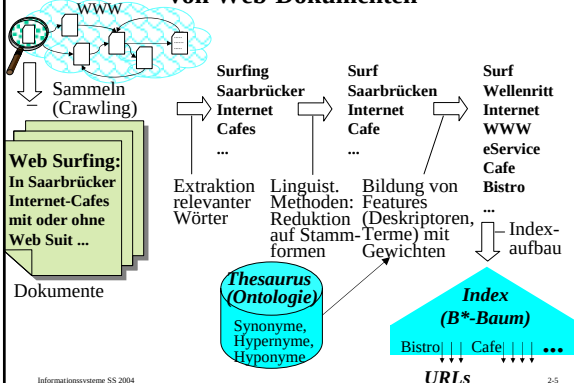
Weitere Operatoren (anhand von Beispielen):

~, &,	Not, And, Or
NEAR (king, David, 10)	höchstens 10 Wörter auseinander
king&David WITHIN Sentence	im selben Satz
!dog	ähnliche Aussprache (z.B. doc, dock)
\$sing	gleicher Wortstamm (z.B. singer, sings, sang)
?apple	ähnliche Schreibweise (z.B. applet, apply)
NTP(computer)	narrower term partative (z.B. hard drive)
NTG(rodent)	narrower term generic (z.B. rat)
NTI(fairytale)	narrower term instance (z.B. Cinderella)
ABOUT(miracles by Jesus)	thematische Suche (verwendet intern selbst andere Operatoren)

Informationssysteme SS 2004

2-4

Sammeln, Analyse und Indexierung von Web-Dokumenten



Informationssysteme SS 2004

2-5

Problem: Inhaltserschließung

Umgang mit „unscharfen“ Daten (und „unscharfen“ Anfragen)

→ Dokumente werden typischerweise durch

Features charakterisiert, z.B.:

- Wörter, Wortpaare oder Phrasen
- Worthäufigkeiten
- Anzahl eingehender Hyperlinks
- title, weitere Tags, Struktur von HTML- oder XML-Seiten
- Farbhäufigkeiten in Bildern (Bildmitte, oberer Rand, etc.)
- usw. usw.

→ Abbildung von natürlichsprachlichem Text auf Features:

- Behandlung von morphologischer Variation
- Behandlung von Synonymen, Hypernymen/Hyponymen und Polysemen (u.a. mittels Thesaurus)

Informationssysteme SS 2004

2-6

Problem: Effektivität (Suchresultatsgüte)
query = „Chernoff theorem“

AltaVista: Fermat's last theorem. Previous topic. Next topic. ...
 URL: www.groups.dcs.st-and.ac.uk/~history/His...st_theorem.html

Northernlight: J. D. Biggins- Publications. Articles on the Branching Random Walk
<http://www.shef.ac.uk/~st1jdb/bibliog.html>

Excite: The Official Web Site of Playboy Lingerie Model Mikki Chernoff <http://www.mikkichernoff.com/>

Google: ...strong convergence \cite{Chernoff}. \begin{theorem} \label{T1} Let...
http://mpej.unige.ch/mp_arc/p/00-277

Yahoo: Moment-generating Functions; Chernoff's Theorem;
<http://www.siam.org/catalog/mcc10/bahadur.htm>

Mathsearch: No matches found.

Informationssysteme SS 2004 2-7

Problem: Effizienz und Skalierbarkeit

Aktuelle „Landkarte“ des Webs:

!!! Aber:

Suchmaschinen überdecken das „Surface Web“:
 4 Mrd. Dokumente, 20 TBytes
 Die meisten Daten sind im „Deep Web“ hinter Portalen:
 500 Mrd. Dokumente, 8 PBytes

Informationssysteme SS 2004 2-8

Bewertung der Retrieval-Güte (Effektivität)

Fähigkeit, zu einer Anfrage **nur** relevante Dokumente zu liefern:

$$\text{Präzision (precision)} = \frac{\text{Anzahl relevanter Dokumente unter Top } r}{r}$$

Fähigkeit, zu einer Anfrage **alle** relevanten Dokumente zu liefern:

$$\text{Ausbeute (recall)} = \frac{\text{Anzahl relevanter Dokumente}}{\text{Anzahl aller relevanten Dokumente}}$$

Typische Güte

Ideale Güte

Informationssysteme SS 2004 2-9

2.2 Web-Crawling und Indexierung

Komponenten einer Web-Suchmaschine:

- **URL Queue Server:** verwaltet Priority-Queue (noch) zu traversierender Links
- **Crawler (Robot, Spider):** holt Dokumente unter Beachtung von Nebenbedingungen (Filetyp, Robot Exclusion Protocol, usw.)
- **Repository Server:** verwaltet DocumentRepository
- **Indexer (inkl. Parser, Stemmer):** analysiert Dokumente und erzeugt Einträge in Lexicon, Anchors und DocumentIndex
- **URL Resolver:** übersetzt URLs in DocIds
- **Link Analyzer:** berechnet Autoritäts-Ranking aufgrund von Links
- **Query Processor:** wertet Anfragen durch Index-Lookups aus und berechnet Resultats-Ranking

Informationssysteme SS 2004 2-10

Datenstrukturen einer Web-Suchmaschine

- **DocumentRepository (DocId, DocContent):**
 alle (HTML-) Dokumente in komprimierter Form
- **Lexicon (TermId, Term):**
 alle vorkommenden Stammformen jeweils mit TermId
- **DocumentIndex (DocId, TermId, Weight, ...):**
 alle Vorkommen von Termen in Dokumenten, optimiert für Zugriff nach DocId
- **TermIndex (TermId, DocId, Weight, ...):**
 alle Dokumente zu allen Termen, optimiert für Zugriff nach TermId
- **Anchors (SourceDocId, TargetDocId, AnchorText):**
 alle Hyperlinks
- **URLIndex (URL, DocId):**
 Umsetzung von URLs auf interne Ids

Informationssysteme SS 2004 2-11

Architektur eines skalierbaren Crawlers

Informationssysteme SS 2004 2-12

Dimensionen sehr großer Web-Suchmaschinen

- > 4 Mrd. Web-Dokumente + 1 Mrd. News-Dokumente
- > 20 Terabytes Rohdaten
- > 10 Mio. Terme
- > 4 Terabytes Index
- > 150 Mio. Anfragen pro Werktag
- < 1 Sek. mittlere Antwortzeit
- < 30 Tage Indexaktualität
- > 1000 Webseiten pro Sek. Crawling

High-End-Server-Farm:

- > 10 000 Intel-Server mit jeweils
- > 1 GB Hauptspeicher, 2 Platten und
- partitionierten, gespiegelten Daten, die über alle Server verteilt sind,
- sowie Lastbalancierung der Queries, Remote-Administration, usw.

Informationssysteme SS 2004

2-13

2.3 Vektorraummodell für IR mit Ranking

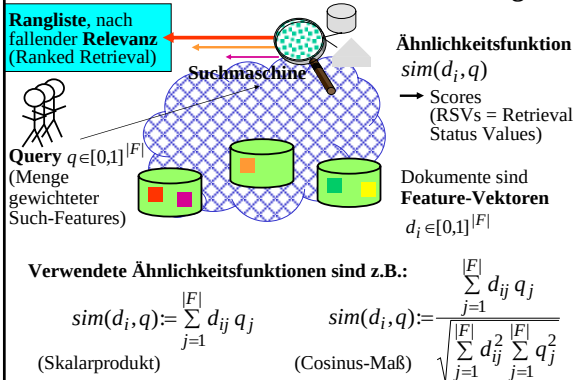
Grundprinzipien:

- Featureraum:** Wörter in Dokumenten werden auf Terme reduziert.
- Dokumentenmodell:** Jedes Dokument d_i wird als Vektor $\in [0,1]^{|F|}$ repräsentiert, wobei d_{ij} das Gewicht des j -ten Terms in d_i angibt.
- Anfragemodell:** Anfragen sind Vektoren $q \in [0,1]^{|F|}$
- Relevanz:** Suchresultatsranking basiert auf einer Ähnlichkeitsfunktion im Vektorraum $[0,1]^{|F|}$
- Crawling:** Das Web wird entlang von Hyperlinks traversiert, um Dokumente zu analysieren und zu indexieren.
- Indexierung:** Zu jedem Term wird eine Liste von Dokumenten-Ids (z.B. URLs) mit dem jeweiligen Gewicht in einem „invertierten File“ (Suchbaum oder Hash-File) angelegt.
- Anfrageverarbeitung:** Anfragen werden zerlegt in Index-Lookups für Einzeltermine, um Trefferkandidatenlisten zu bestimmen.

Informationssysteme SS 2004

2-14

Vektorraummodell für Relevanz-Ranking



Informationssysteme SS 2004

2-15

Termgewichtung in Dokumenten

Betrachtet werden die folgenden Werte

(für N Dokumente und M Terme):

- tf_{ij} : Häufigkeit (term frequency) des Terms t_i in Dokument d_j
- df_i : Anzahl der Dokumente mit Term t_i (doc. frequency)
- idf_i : N / df_i (inverse document frequency)
- cf_i : Häufigkeit von t_i in allen Dokumenten (corpus frequency) (ggf. mit separater Berücksichtigung von Termen in title u.ä.)

Grundprinzip:

Das Gewicht w_{ij} von Term t_i in Dokument d_j sollte mit tf_{ij} und mit idf_i monoton wachsen.

→ erster Ansatz: $w_{ij} = tf_{ij} * idf_i$ (**tf-idf-Formel**)

Ggf. sollten die Gewichte w_{ij} wie folgt zu ω_{ij} normiert werden:

$$\omega_{ij} := w_{ij} / \sqrt{\sum_k w_{kj}^2}$$

Informationssysteme SS 2004

2-16

Variationen der Termgewichtung mit tf und idf

Empirische Resultate zeigen, daß in der Regel die tf- und idf-Werte normalisiert und/oder gedämpft sein sollten.

Normalisierung tf-Werte: $tf_{ij} := \frac{tf_{ij}}{\max_k tf_{kj}}$

Gedämpfte tf-Werte: $tf_{ij} := (1 + \log tf_{ij})$

Gedämpfte idf-Werte: $idf_i := \log \frac{N}{df_i}$

Häufige Variante:

$$w_{ij} := \frac{tf_{ij}}{\max_k tf_{kj}} \log \frac{N}{df_i} \quad \text{tf*idf-Formel}$$

$$\omega_{ij} := w_{ij} / \sqrt{\sum_k w_{kj}^2}$$

Informationssysteme SS 2004

2-17

Query-Verfeinerung mit Relevanz-Feedback nach Rocchio

Für Resultat D der Query q bestehe das Relevanz-Feedback des Benutzers aus einer Partitionierung von D in

- D^+ : die Menge der relevanten Dokumente in D und
- D^- : die Menge der nicht relevanten Dokumente in D.

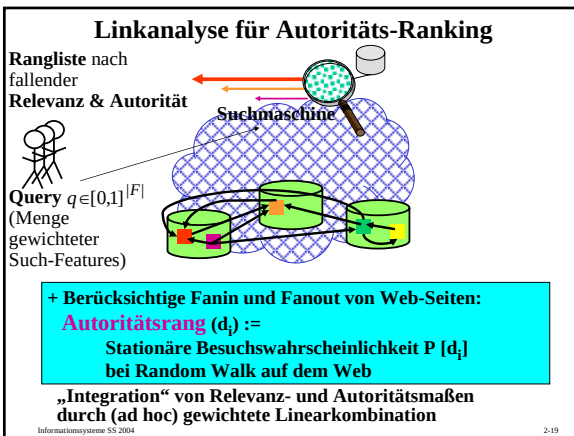
Generiere verfeinerte Query q' :

$$\bar{q}' := \alpha \bar{q} + \frac{\beta}{|D^+|} \sum_{d_j \in D^+} \bar{d}_j - \frac{\gamma}{|D^-|} \sum_{d_j \in D^-} \bar{d}_j$$

mit geeigneten Gewichten $\alpha, \beta, \gamma \in [0,1]$ (typischerweise $\alpha > \beta > \gamma$)

Informationssysteme SS 2004

2-18



Weitere IR-Modelle

- **Probabilistisches Retrieval & Statistische Sprachmodelle:**
Ranking aufgrund von Relevanzwahrscheinlichkeiten, die aus - geschätzten - Basisparametern abgeleitet werden.
- **Fuzzy-Set-Modell:**
Queries (inkl. einzelner Terme) beschreiben Fuzzy-Mengen mit Dokumenten als Elementen vom Grad $\mu \in [0,1]$. Mengenoperationen verwenden Funktionen $\max$, $\min$, $1-\mu$.
- **Latent Semantic Indexing:**
Berücksichtigung von Termkorrelationen durch Transformation des Term-Vektorraums in einen Themen-Vektorraum niedrigerer Dimensionalität
- **Neuronale Netze** und andere Inferenznetze zum Lernen von Termgewichten

Informationssysteme SS 2004 2-20

2.4 Anfrageausführung mit Ranking

Suchbaum

algorithm	...	performance	...	z-transform
17: 0.3		12: 0.5		11: 0.6
44: 0.4		14: 0.4		17: 0.1
52: 0.1		28: 0.1		28: 0.7
53: 0.8		44: 0.2		...
55: 0.6		51: 0.6		
...		52: 0.3		

Indexliste mit (DocId, tf*idf)
sortiert nach DocId

Gegeben: Query $q = t_1 t_2 \dots t_z$ mit z Keywords
 Ähnlichkeitsfunktion $\text{score}(q,d)$ für Dok. $d \in D$, z.B.: $\bar{q} \cdot \bar{d}$
 Finde: Top-k-Resultate bzgl. $\text{score}(q,d)$ (z.B.: $\sum_{i \in q} s_i(d)$)

Naïver QP Algorithmus:
 candidate-docs := $\emptyset$;
 for $i=1$ to z do {
 candidate-docs := candidate-docs $\cup$ index-lookup(t_i) };
 for each $d_j \in$ candidate-docs do {compute $\text{score}(q,d_j)$ };
 sort candidate-docs by $\text{score}(q,d_j)$ descending;

Informationssysteme SS 2004 2-21

Fagin's TA (Threshold Algorithm)

scan all lists L_i ($i=1..m$) in parallel:
 consider d_j at position pos_i in L_i ;
 $\text{high}_i := s_i(d_j)$;
 if $d_j \notin \text{top-k}$ then {
 look up $s_v(d_j)$ in all lists L_v with $v \neq i$; // random access
 compute $s(d_j) := \text{aggr} \{s_v(d_j) \mid v=1..m\}$;
 if $s(d_j) > \min \text{score among top-k}$ then
 add d_j to top-k and remove min-score d from top-k; }
 if $\min \text{score among top-k} \geq \text{aggr} \{\text{high}_v \mid v=1..m\}$ then exit;

	L1	L2	L3	top-k:
f: 0.5	a: 0.55	h: 0.35		
b: 0.4	b: 0.2	d: 0.35		
c: 0.35	f: 0.2	b: 0.2		
a: 0.3	g: 0.2	a: 0.1		
h: 0.1	c: 0.1	c: 0.05		
d: 0.1		f: 0.05		

m=3
aggr: sum
k=2

~~f: 0.75~~
a: 0.95
b: 0.8

Informationssysteme SS 2004 2-22

TA-Sorted (Fagin, Güntzer et al., ...)

scan index lists in parallel:
 consider d_j at position pos_i in L_i ;
 $E(d_j) := E(d_j) \cup \{i\}$; $\text{high}_i := s_i(q,d_j)$;
 $\text{bestscore}(d_j) := \text{aggr}\{x_1, \dots, x_m\}$
 with $x_i := s_i(q,d_j)$ for $i \in E(d_j)$, high_i for $i \notin E(d_j)$;
 $\text{worstscore}(d_j) := \text{aggr}\{x_1, \dots, x_m\}$
 with $x_i := s_i(q,d_j)$ for $i \in E(d_j)$, 0 for $i \notin E(d_j)$;
 top-k := k docs with largest worstscore;
 if $\min \text{worstscore among top-k} \geq \text{bestscore}\{d \mid d \text{ not in top-k}\}$
 then exit;

	L1	L2	L3	top-k:
f: 0.5	a: 0.55	h: 0.35		a: 0.95
b: 0.4	b: 0.2	d: 0.35		b: 0.8
c: 0.35	f: 0.2	b: 0.2		
a: 0.3	g: 0.2	a: 0.1		
h: 0.1	c: 0.1	c: 0.05		
d: 0.1		f: 0.05		

m=3
aggr: sum
k=2

~~f: 0.7 + ? $\leq$ 0.7 + 0.1~~
~~h: 0.45 + ? $\leq$ 0.45 + 0.2~~
~~c: 0.35 + ? $\leq$ 0.35 + 0.3~~
~~d: 0.35 + ? $\leq$ 0.35 + 0.3~~
~~g: 0.2 + ? $\leq$ 0.2 + 0.4~~

Informationssysteme SS 2004 2-24

2.5 Grundlagen aus der Linearen Algebra

Eine Menge S von Vektoren heißt **linear unabhängig**, wenn sich kein $x \in S$ als Linearkombination der anderen Vektoren aus S schreiben lässt. Der **Rang** einer Matrix A ist die maximale Anzahl linear unabhängiger Zeilen- oder Spaltenvektoren.

Eine **Basis** einer $n \times n$ -Matrix A ist eine Menge S von Zeilen- bzw. Spaltenvektoren, so dass alle Zeilen bzw. Spalten Linearkombinationen der Vektoren aus S ist.

Eine Menge S von $n \times 1$ -Vektoren heißt **Orthonormalbasis**, wenn für alle $x, y \in S$ gilt:

$$\|x\|_2 := \sqrt{\sum_{i=1}^n x_i^2} = 1 = \|y\|_2 \quad \text{und} \quad x \cdot y = 0$$

Informationssysteme SS 2004 2-24

Eigenwerte und Eigenvektoren

Seien A eine reellwertige $n \times n$ -Matrix, x ein reellwertiger $n \times 1$ -Vektor und λ ein reeller Skalarwert. Die Lösungen x und λ der Gleichung $A \times x = \lambda x$ heißen **Eigenvektor** bzw. **Eigenwert** von A . Die Eigenvektoren von A sind Vektoren, deren Richtungen bei der durch A beschriebenen Linearabbildung erhalten bleiben.

Die Eigenwerte von A sind die Nullstellen des charakteristischen Polynoms $f(\lambda)$ von A : $f(\lambda) = |A - \lambda I| = 0$

mit der Determinantenentwicklung nach der i -ten Zeile:

$$|A| = \sum_{j=1}^n (-1)^{i+j} a_{ij} |A^{(ij)}| \quad \text{wobei man die Matrix } A^{(ij)} \text{ aus } A \text{ durch Streichung der } i. \text{ Zeile und der } j. \text{ Spalte erhält}$$

Die reellwertige $n \times n$ -Matrix A heißt **symmetrisch**, wenn $a_{ij} = a_{ji}$ für alle i, j . A heißt **positiv definit**, wenn für alle $n \times 1$ -Vektoren $x \neq 0$ gilt: $x^T \times A \times x > 0$.

Wenn A symmetrisch ist, sind alle Eigenwerte von A reell.

Wenn A symmetrisch und positiv definit ist, sind alle Eigenwerte positiv.

Informationssysteme SS 2004

2-25

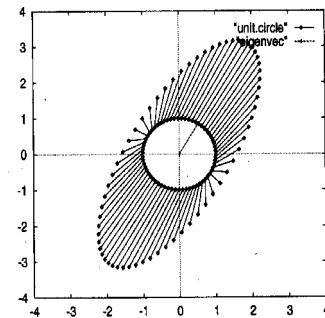
Illustration von Eigenvektoren

$$\text{Matrix } A = \begin{pmatrix} 2 & 1 \\ 1 & 3 \end{pmatrix}$$

beschreibt
affine Abb. $x \mapsto Ax$

Eigenvektor
 $x_1 = (0.52 \ 0.85)^T$
zum Eigenwert $\lambda_1 = 3.62$

Eigenvektor
 $x_2 = (0.85 \ -0.52)^T$
zum Eigenwert $\lambda_2 = 1.38$



Informationssysteme SS 2004

2-26

Spektralsatz der Linearen Algebra

Spektralsatz

(**Hauptachsentransformation, Principal Component Analysis, PCA**):

Sei A eine symmetrische $n \times n$ -Matrix mit Eigenwerten $\lambda_1, \dots, \lambda_n$ und Eigenvektoren $x_1, \dots, x_n$, so dass $\|x_i\|_2 = 1$ für alle i .

Die Eigenvektoren bilden eine Orthonormalbasis von A .

Dann gilt:

$$D = Q^T \times A \times Q,$$

wobei D eine Diagonalmatrix ist mit den Diagonalelementen $\lambda_1, \dots, \lambda_n$ und Q aus den Spaltenvektoren $x_1, \dots, x_n$ besteht.

Informationssysteme SS 2004

2-27

2.6 Latent Semantic Indexing (LSI): Grundidee

Ziel:

Transformation der Dokumentvektoren vom hochdimensionalen Termvektorraum in einen **Themenvektorraum** niedrigerer Dimensionalität unter

- Ausnutzung von Korrelationen zwischen Termen (z.B. „Web“ und „Internet“ häufig zusammen)
- implizite Differenzierung von Polysemen, die sich in ihren Korrelationen mit anderen Termen unterscheiden (z.B. „Java“ mit „Library“ vs. „Java“ mit „Kona Blend“ vs. „Java“ mit „Borneo“)

mathematisch:

gegeben: m Terme, n Dokumente (i.d.R. $n > m$) und eine $m \times n$ -Term-Dokument-Ähnlichkeitsmatrix A ,
gesucht: möglichst gute – ähnlichkeitsbewahrende – Abbildung der Spaltenvektoren von A in einen k -dimensionalen Vektorraum ($k \ll m$) für gegebenes k

Informationssysteme SS 2004

2-28

Exkurs: Singulärwertdekomposition (SVD)

Satz:

Jede reellwertige $m \times n$ -Matrix A mit Rang r kann zerlegt werden in die Form $A = U \times \Delta \times V^T$ mit einer $m \times r$ -Matrix U mit orthonormalen Spaltenvektoren, einer $r \times r$ -Diagonalmatrix Δ und einer $n \times r$ -Matrix V mit orthonormalen Spaltenvektoren. Diese Zerlegung heißt Singulärwertdekomposition und ist eindeutig, wenn die Elemente von Δ der Größe nach geordnet werden.

Satz:

In der Singulärwertdekomposition $A = U \times \Delta \times V^T$ der Matrix A sind U , Δ und V wie folgt bestimmt:

- Δ besteht aus den Singulärwerten von A , d.h. den positiven Wurzeln der Eigenwerte von $A^T \times A$,
- die Spaltenvektoren von U sind die Eigenvektoren von $A \times A^T$,
- die Spaltenvektoren von V sind die Eigenvektoren von $A^T \times A$.

Informationssysteme SS 2004

2-29

Schematische Darstellung der SVD

$$\begin{pmatrix} \text{Term-Dokument-Ähnlichkeitsmatrix} \end{pmatrix}_{m \times n} = \begin{pmatrix} \text{Term-Themen-Ähnlichkeit} \end{pmatrix}_{m \times r} \begin{pmatrix} \sigma_1 & & 0 \\ & \ddots & \\ 0 & & \sigma_r \end{pmatrix}_{r \times r} \begin{pmatrix} \text{Themen-Dokument-Ähnlichkeit} \end{pmatrix}_{r \times n}$$

$U \qquad \qquad \Delta \qquad \qquad V^T$

Informationssysteme SS 2004

2-30

Exkurs: SVD als Regressionsverfahren

Satz:

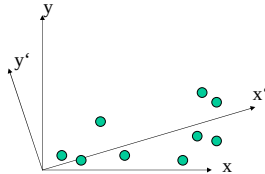
Sei A eine $m \times n$ -Matrix mit Rang r und sei $A_k = U_k \times \Delta_k \times V_k^T$, wobei die $k \times k$ -Diagonalmatrix Δ_k die k größten Singulärwerte von A enthält und die $m \times k$ -Matrix U_k sowie die $n \times k$ -Matrix V_k aus den zugehörigen Eigenvektoren der Singulärwertdekomposition von A bestehen.

Unter allen $m \times n$ -Matrizen C mit einem Rang, der nicht größer als k ist, ist A_k diejenige Matrix, die den Wert

$$\|A - C\|_F^2 = \sum_{i=1}^m \sum_{j=1}^n (A_{ij} - C_{ij})^2$$

minimiert (die Frobenius-Norm).

Beispiel:
 $m=2, n=8, k=1$
 Projektion auf x' -Achse
 minimiert „Fehler“ bzw.
 maximiert Varianz
 im k -dimensionalen Raum



Informationssysteme SS 2004

2-31

$p \times q$

Anwendung der SVD auf das Vektorraummodell

A ist die $m \times n$ -Term-Dokument-Ähnlichkeitsmatrix. Dann sind:

- U bzw. U_k die $m \times r$ - bzw. $m \times k$ -Term-Themen-Ähnlichkeitsmatrix,
- V bzw. V_k die $n \times r$ - bzw. $n \times k$ -Dokument-Themen-Ähnlichkeitsmatrix
- $A \times A^T$ bzw. $A_k \times A_k^T$ die $m \times m$ -Term-Term-Ähnlichkeitsmatrix,
- $A^T \times A$ bzw. $A_k^T \times A_k$ die $n \times n$ -Dokument-Dokument-Ähnlichkeitsmatrix

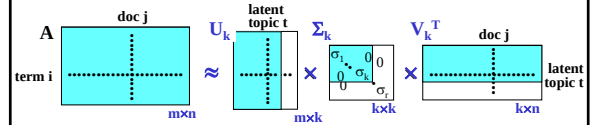


Abbildung von $m \times 1$ -Vektoren in Themenraum: $d_j \mapsto U_k^T \times d_j =: d_j'$

$$q \mapsto U_k^T \times q =: q'$$

Skalarprodukt-Ähnlichkeit im Themenraum: $d_j'^T \times q' = ((\Delta_k V_k^T)_{ij})^T \times q'$

Informationssysteme SS 2004

2-32

Indexierung und Anfrageauswertung

- Die Matrix $\Delta_k V_k^T$ entspricht einem „Themen-Index“ und ist in einer geeigneten Datenstruktur zu verwalten.
 Statt $\Delta_k V_k^T$ kann man auch vereinfachend V_k^T als Index verwenden.
- Zusätzlich muß die Term-Themen-Abbildung U_k gespeichert werden.
- Eine Anfrage q (ein $m \times 1$ -Spaltenvektor) im Termvektorraum wird in die Anfrage $q' = U_k^T \times q$ (ein $k \times 1$ -Spaltenvektor) transformiert und dann im Themenvektorraum (also V_k) ausgewertet (z.B. mittels Skalarproduktsmaß $V_k^T \times q'$ oder Cosinusmaß)
- Ein neues Dokument d (ein $m \times 1$ -Spaltenvektor) wird in $d' = U_k^T \times d$ (ein $k \times 1$ -Spaltenvektor) transformiert und als neue Spalte an den „Index“ V_k^T angefügt („folding-in“)

Informationssysteme SS 2004

2-33

Beispiel 1 für Latent Semantic Indexing

$m=5$ (Bush, Schröder, Korea, Klose, Völler), $n=7$

$$A = \begin{pmatrix} 1 & 2 & 1 & 5 & 0 & 0 & 0 \\ 1 & 2 & 1 & 5 & 0 & 0 & 0 \\ 1 & 2 & 1 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 3 & 1 \\ 0 & 0 & 0 & 2 & 3 & 1 \end{pmatrix} = \begin{pmatrix} 0.58 & 0.00 \\ 0.58 & 0.00 \\ 0.58 & 0.00 \\ 0.00 & 0.71 \\ 0.00 & 0.71 \end{pmatrix} \times \begin{pmatrix} 9.64 & 0.00 \\ 0.00 & 5.29 \end{pmatrix} \times \begin{pmatrix} 0.18 & 0.36 & 0.18 & 0.90 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.53 & 0.80 & 0.27 \end{pmatrix}$$

$U \quad \Delta \quad V^T$

Anfrage $q = (0 \ 0 \ 1 \ 0 \ 0)^T$ wird in

$q' = U^T \times q = (0.58 \ 0.00)^T$ transformiert und gegen V^T evaluiert.

Neues Dokument $d8 = (1 \ 1 \ 0 \ 0 \ 0)^T$ wird in

$d8' = U^T \times d8 = (1.16 \ 0.00)^T$ transformiert und an V^T angefügt.

Informationssysteme SS 2004

2-34

Beispiel 2 für Latent Semantic Indexing

$m=6$ terms

- t1: bak(e,ing)
- t2: recipe(s)
- t3: bread
- t4: cake
- t5: pastr(y,ies)
- t6: pie

$n=5$ documents

- d1: How to bake bread without recipes
- d2: The classic art of Viennese Pastry
- d3: Numerical recipes: the art of scientific computing
- d4: Breads, pastries, pies and cakes: quantity baking recipes
- d5: Pastry: a book of best French recipes

$$A = \begin{pmatrix} 0.5774 & 0.0000 & 0.0000 & 0.4082 & 0.0000 \\ 0.5774 & 0.0000 & 1.0000 & 0.4082 & 0.7071 \\ 0.5774 & 0.0000 & 0.0000 & 0.4082 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.4082 & 0.0000 \\ 0.0000 & 1.0000 & 0.0000 & 0.4082 & 0.7071 \\ 0.0000 & 0.0000 & 0.0000 & 0.4082 & 0.0000 \end{pmatrix}$$

Informationssysteme SS 2004

2-35

Beispiel 2 für Latent Semantic Indexing (2)

$$A = \begin{pmatrix} 0.2670 & -0.2567 & 0.5308 & -0.2847 \\ 0.7479 & -0.3981 & -0.5249 & 0.0816 \\ 0.2670 & -0.2567 & 0.5308 & -0.2847 \\ 0.1182 & -0.0127 & 0.2774 & 0.6394 \\ 0.5198 & 0.8423 & 0.0838 & -0.1158 \\ 0.1182 & -0.0127 & 0.2774 & 0.6394 \end{pmatrix} \times \begin{pmatrix} 1.6950 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 1.1158 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.8403 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.4195 \end{pmatrix} \times \begin{pmatrix} 0.4366 & 0.3067 & 0.4412 & 0.4909 & 0.5288 \\ -0.4717 & 0.7549 & -0.3568 & -0.0346 & 0.2815 \\ 0.3688 & 0.0998 & -0.6247 & 0.5711 & -0.3712 \\ -0.6715 & -0.2760 & 0.1945 & 0.6571 & -0.0577 \end{pmatrix}$$

$U \quad \Delta \quad V^T$

Informationssysteme SS 2004

2-36

Beispiel 2 für Latent Semantic Indexing (3)

$$A_3 = \begin{pmatrix} 0.4971 & -0.0330 & 0.0232 & 0.4867 & -0.0069 \\ 0.6003 & 0.0094 & 0.9933 & 0.3858 & 0.7091 \\ 0.4971 & -0.0330 & 0.0232 & 0.4867 & -0.0069 \\ 0.1801 & 0.0740 & -0.0522 & 0.2320 & 0.0155 \\ -0.0326 & 0.9866 & 0.0094 & 0.4402 & 0.7043 \\ 0.1801 & 0.0740 & -0.0522 & 0.2320 & 0.0155 \end{pmatrix} = U_3 \times \Delta_3 \times V_3^T$$

Informationssysteme SS 2004

2-37

Beispiel 2 für Latent Semantic Indexing (4)

Anfrage q: baking bread

$$q = (1 \ 0 \ 1 \ 0 \ 0 \ 0)^T$$

Transformation in den Themenraum mit k=3

$$q' = U_k^T \times q = (0.5340 \ -0.5134 \ 1.0616)^T$$

Skalarprodukt-Ähnlichkeit im Themenraum mit k=3:

$$\text{sim}(q, d1) = V_{k_1}^T \times q' \approx 0.86 \quad \text{sim}(q, d2) = V_{k_2}^T \times q' \approx -0.12$$

$$\text{sim}(q, d3) = V_{k_3}^T \times q' \approx -0.24 \quad \text{usw.}$$

Folding-in eines neuen Dokuments d6:

algorithmic recipes for the computation of pie

$$d6 = (0 \ 0.7071 \ 0 \ 0 \ 0 \ 0.7071)^T$$

Transformation in den Themenraum mit k=3

$$d6' = U_k^T \times d6 \approx (0.5 \ -0.28 \ -0.15)$$

d6' als neue Spalte an V_k^T anhängen

Informationssysteme SS 2004

2-38

Mehrsprachiges Retrieval mit LSI

- Konstruiere LSI-Modell (U_k, Δ_k, V_k^T) anhand von Trainingsdokumenten, die mehrsprachig vorliegen:
 - Betrachte alle Sprachversionen eines Dokuments als ein einziges Dokument und
 - extrahiere zur Indexierung alle Terme oder Wörter unabhängig von der Sprache.
- Indexiere weitere Dokumente durch „folding-in“, also Abbildung in den Themen-Vektorraum und Anhängen an V_k^T .
- Anfragen können dann in beliebiger Sprache gestellt werden und liefern Antworten in allen Sprachen.

Beispiel:

d1: How to bake bread without recipes.

Wie man ohne Rezept Brot backen kann.

d2: Pastry: a book of best French recipes.

Gebäck: eine Sammlung der besten französischen Rezepte.

Terme sind dann z.B. bake, bread, recipe, backen, Brot, Rezept, usw. Dokumente und Terme werden auf einen kompakten Themenraum abgebildet.

Informationssysteme SS 2004

2-39

Zusammenfassung zu LSI

- + Elegantes, mathematisch wohlfundiertes Modell
- + „Automatisches Lernen“ von Termkorrelationen (inkl. morphologischer Wortvarianten, Mehrsprachigkeit)
- + Impliziter Thesaurus (durch Korrelation von Synonymen)
- + Implizite Diskriminierung der verschiedenen Bedeutungen von Polysemen (durch verschiedene Korrelationen)
- + Verbesserte Retrievalgüte auf „geschlossenen“ Korpora (z.B. TREC-Benchmark, Finanznachrichten, Patentkollektionen u.ä.) mit empirisch günstigstem k in der Größenordnung 100-200
- Schwierige Wahl von günstigem k
- Rechen- und Speicheraufwand für sehr große (z.T. aber dünn besetzte) Matrizen
- Keine überzeugenden Resultate für Web-Suchmaschinen

Informationssysteme SS 2004

2-40

Kapitel 3: Automatische Klassifikation von Dokumenten

- 3.1 Einfache Klassifikatoren
- 3.2 Grundlagen aus der Wahrscheinlichkeitsrechnung
- 3.3 Naive-Bayes-Klassifikator
- 3.4 Feature-Selektion

Informationssysteme SS2004

3-1

Automatische Klassifikation von Dokumenten

Ziel:

Organisation von Dokumenten in (hierarchischen) Ontologien mit möglichst geringem intellektuellem Aufwand

Techniken:

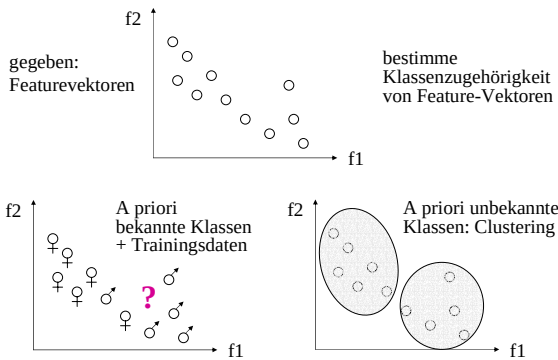
- Klassifikation mit Training (Supervised Learning)
 - kNN-Verfahren
 - Rocchio-Verfahren
 - Naives Bayes-Verfahren
 - Support Vector Machines (SVM)
 - ...
- Klassifikation ohne Training (Unsupervised Learning)
 - Verfahren der Clusteranalyse

Orthogonal dazu gibt es flache vs. hierarchische Verfahren

Informationssysteme SS2004

3-2

Klassifikationsproblem (Kategorisierung)



Informationssysteme SS2004

3-3

Anwendung von Klassifikationsverfahren im IR

- **Filtern:** teste eintreffende Dokumente (z.B. Mail, News), ob sie in eine interessante Klasse fallen
- **Übersicht:** organisiere Query-/Crawler-Resultate, Verzeichnisse, Feeds, etc.
- **Query-Expansion:** ordne Query einer Klasse zu und ergänze dementsprechende Suchterme
- **Relevanz-Feedback:** klassifiziere Treffer und lasse Benutzer relevante Klassen identifizieren, um bessere Query zu generieren
- **Query-Effizienz:** beschränke (Index-)Suche auf relevante Klasse(n)

Klassifikationsvarianten:

- mit Termen, Termhäufigkeiten, Linkstruktur als Features
- binär: Gehört ein Dokument zu einer Klasse c oder nicht?
- mehrstellig: In welche von k Klassen passt ein Dokument am besten?
- hierarchisch: Iteration der Klassifikation über ontologischen Baum

Informationssysteme SS2004

3-4

Bewertung der Klassifikationsgüte

empirisch durch automatische Klassifikation von Dokumenten, die nicht zu den Trainingsdaten gehören

Für binäre Klassifikation bzgl. Klasse C:

a = #Dok., die zu C klassifiziert wurden und zu C gehören

b = #Dok., die zu C klassifiziert wurden, aber nicht zu C gehören

c = #Dok., die nicht zu C klassifiziert wurden, aber zu C gehören

d = #Dok., die nicht zu C klassifiziert wurden und nicht zu C gehören

$$\text{Genauigkeit (accuracy)} = \frac{a + d}{a + b + c + d}$$

$$\text{Präzision (precision)} = \frac{a}{a + b} \quad \text{Ausbeute (recall)} = \frac{a}{a + c}$$

Für mehrstellige Klassifikation bzgl. Klassen C1, ..., Ck:

- Makrodurchschnitt über k Klassen oder
- Mikrodurchschnitt über k Klassen

Informationssysteme SS2004

3-5

3.1 Einfache Klassifikatoren: k-Nearest-Neighbor-Verfahren (kNN)

Schritt 1:

Finde unter den Trainingsdokumenten aller Klassen die k (z.B. 10-100) bzgl. der (Cosinus-) Ähnlichkeit nächsten Nachbarn eines neuen Dokuments $\vec{d}$

Schritt 2:

Ordne $\vec{d}$ derjenigen Klasse Cj zu, für die die Funktion

$$f(\vec{d}, C_j) = \sum_{\vec{v} \in kNN(\vec{d})} \text{sim}(\vec{d}, \vec{v}) * \begin{cases} 1 & \text{falls } \vec{v} \in C_j \\ 0 & \text{sonst} \end{cases}$$

maximal wird

Bei binärer Klassifikation ordne $\vec{d}$ der Klasse C zu, falls $f(\vec{d}, C)$ über einem Schwellwert δ ($\delta > 0.5$) liegt.

Informationssysteme SS2004

3-6

Klassifikationsverfahren von Rocchio

Schritt 1:

Repräsentiere die Trainingsdokumente einer Klasse C_j
- mit tf*idf-Vektorkomponenten – durch den **Prototypvektor**:

$$\vec{c}_j := \alpha \frac{1}{|C_j|} \sum_{\vec{d} \in C_j} \frac{\vec{d}}{\|\vec{d}\|} - \beta \frac{1}{|D - C_j|} \sum_{\vec{d} \in D - C_j} \frac{\vec{d}}{\|\vec{d}\|}$$

mit geeigneten Koeffizienten α und β (z.B. $\alpha=16$, $\beta=4$)

Schritt 2:

Ordne ein neues Dokument $\vec{d}$ derjenigen Klasse C_j zu, für die Cosinus-Ähnlichkeit $\cos(\vec{c}_j, \vec{d})$ maximal ist.

Satz:

Für $\alpha=\beta=1$ maximiert $\vec{c}_j$ die Funktion:

$$f(\vec{c}_j) = \frac{1}{|C_j|} \sum_{\vec{d} \in C_j} \cos(\vec{c}_j, \vec{d}) - \frac{1}{|D - C_j|} \sum_{\vec{d} \in D - C_j} \cos(\vec{c}_j, \vec{d})$$

Informationssysteme SS2004

3-7

3.2 Grundlagen aus der Wahrscheinlichkeitsrechnung

Ein **Wahrscheinlichkeitsraum** ist ein Tripel (Ω, E, P) mit

- einer Menge Ω elementarer Ereignisse,
- einer Familie E von Teilmengen von Ω mit $\Omega \in E$, die unter $\cap$, $\cup$ und $-$ mit abzählbar vielen Operanden abgeschlossen ist (bei endlichem Ω ist in der Regel $E=2^\Omega$), und
- einem W.maß $P: E \rightarrow [0,1]$ mit $P[\Omega]=1$ und $P[\cup_i A_i] = \sum_i P[A_i]$ für abzählbar viele, paarweise disjunkte A_i

Eigenschaften von P :

$$\begin{aligned} P[A] + P[\bar{A}] &= 1 & P[\emptyset] &= 0 \\ P[A \cup B] &= P[A] + P[B] - P[A \cap B] & P[\Omega] &= 1 \end{aligned}$$

Informationssysteme SS2004

3-8

Zufallsvariable

Eine **Zufallsvariable** X über einem W.raum (Ω, E, P) ist eine Funktion $X: \Omega \rightarrow M$ mit $M \subseteq \mathbb{R}$, so daß $\{e \mid X(e) \leq x\} \in E$ für alle $x \in M$.

$F_X: M \rightarrow [0,1]$ mit $F_X(x) = P[X \leq x]$ heißt **Verteilungsfunktion** von X ; bei abzählbarer Menge M heißt $f_X: M \rightarrow [0,1]$ mit $f_X(x) = P[X = x]$

Dichtefunktion von X , ansonsten ist $f_X(x)$ durch $F'_X(x)$ gegeben.

Informationssysteme SS2004

3-9

Momente

Für eine diskrete Zufallsvariable X mit Dichtefunktion f_X sind

$$E[X] = \sum_{k \in M} k f_X(k) \quad \text{der Erwartungswert von } X$$

$$E[X^i] = \sum_{k \in M} k^i f_X(k) \quad \text{das } i. \text{ Moment von } X$$

$$V[X] = E[(X - E[X])^2] = E[X^2] - E[X]^2 \quad \text{die Varianz von } X$$

Für eine kontinuierliche Zufallsvariable X mit Dichtefunktion f_X sind

$$E[X] = \int_{-\infty}^{+\infty} x f_X(x) dx \quad \text{der Erwartungswert von } X$$

$$E[X^i] = \int_{-\infty}^{+\infty} x^i f_X(x) dx \quad \text{das } i. \text{ Moment von } X$$

$$V[X] = E[(X - E[X])^2] = E[X^2] - E[X]^2 \quad \text{die Varianz von } X$$

Erwartungswerte sind additiv, $E[X + Y] = E[X] + E[Y]$
Verteilungen nicht

Informationssysteme SS2004

3-10

Wichtige diskrete Verteilungen

- Gleichverteilung über $\{1, 2, \dots, m\}$:

$$P[X = k] = f_X(k) = \frac{1}{m} \quad \text{für } 1 \leq k \leq m$$

- Binomialverteilung (Münzwurf n -mal wiederholt; X : #Köpfe):

$$P[X = k] = f_X(k) = \binom{n}{k} p^k (1-p)^{n-k}$$

- Poisson-Verteilung (mit Rate λ):

$$P[X = k] = f_X(k) = e^{-\lambda} \frac{\lambda^k}{k!}$$

- Geometrische Verteilung (# Münzwürfe bis zum ersten Kopf):

$$P[X = k] = f_X(k) = (1-p)^{k-1} p$$

- 2-Poisson-Mix (mit $a_1 + a_2 = 1$):

$$P[X = k] = f_X(k) = a_1 e^{-\lambda_1} \frac{\lambda_1^k}{k!} + a_2 e^{-\lambda_2} \frac{\lambda_2^k}{k!}$$

Informationssysteme SS2004

3-11

Kontinuierliche Verteilungen

- Gleichverteilung über dem Intervall $[a,b]$

$$f_X(x) = \frac{1}{b-a} \quad \text{für } a \leq x \leq b \quad (0 \text{ sonst})$$

- Exponentialverteilung (z.B. Zeit bis zum nächsten Ereignis eines Poisson-Prozesses) mit Rate $\lambda = \lim_{\Delta t \rightarrow 0} (\# \text{ Ereignisse in } \Delta t) / \Delta t$:

$$f_X(x) = \lambda e^{-\lambda x} \quad \text{für } x \geq 0 \quad (0 \text{ sonst})$$

- Hyperexponential-Verteilung: $f_X(x) = p\lambda_1 e^{-\lambda_1 x} + (1-p)\lambda_2 e^{-\lambda_2 x}$

- Pareto-Verteilung: $f_X(x) \rightarrow \frac{a}{b} \left(\frac{b}{x}\right)^{a+1} \quad \text{für } x > b, 0 \text{ sonst}$

Beispiel einer „Heavy-tailed“-Verteilung mit $f_X(x) \rightarrow \frac{c}{x^{\alpha+1}}$

Informationssysteme SS2004

3-12

Normalverteilung

- Normalverteilung $N(\mu, \sigma^2)$ (Gauß-Verteilung; approximiert Summen unabhängiger, identisch verteilter Zufallsvariablen):

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$



- Verteilungsfunktion von $N(0,1)$: $\Phi(z) = \int_{-\infty}^z \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx$

Sei X normalverteilt mit Erwartungswert μ und Varianz σ^2 .

$$\text{Dann ist } Y := \frac{X - \mu}{\sigma}$$

normalverteilt mit Erwartungswert 0 und Varianz 1.

Informationssysteme SS2004

3-13

Zentraler Grenzwertsatz

Satz:

Seien $X_1, \dots, X_n$ unabhängig und identisch verteilte Zufallsvariablen mit Erwartungswert μ und Varianz σ^2 .

Die Verteilungsfunktion F_n der Zufallsvariablen $Z_n := X_1 + \dots + X_n$ konvergiert gegen eine Normalverteilung $N(n\mu, n\sigma^2)$ mit Erwartungswert $n\mu$ und Varianz $n\sigma^2$:

$$\lim_{n \rightarrow \infty} P[a \leq \frac{Z_n - n\mu}{\sqrt{n\sigma^2}} \leq b] = \Phi(b) - \Phi(a)$$

Korollar:

$$\bar{X} := \frac{1}{n} \sum_{i=1}^n X_i$$

konvergiert gegen eine Normalverteilung $N(\mu, \sigma^2/n)$

mit Erwartungswert $n\mu$ und Varianz σ^2/n .

Informationssysteme SS2004

3-14

Satz von Bayes

Zwei Ereignisse A, B eines W.raums heißen **unabhängig**, wenn gilt $P[A \cap B] = P[A]P[B]$.

Die **bedingte Wahrscheinlichkeit** $P[A|B]$ von A unter der Bedingung (Hypothese) B ist definiert als: $P[A|B] = \frac{P[A \cap B]}{P[B]}$

Satz von der totalen Wahrscheinlichkeit:

Für eine Partitionierung von Ω in Ereignisse $B_1, \dots, B_n$ gilt:

$$P[A] = \sum_{i=1}^n P[A|B_i]P[B_i]$$

$$\text{Satz von Bayes: } P[A|B] = \frac{P[B|A]P[A]}{P[B]}$$

A-Posteriori-W.
von A

Informationssysteme SS2004

3-15

3.3 Naives Bayes-Verfahren mit binären Features X_i

$$\text{Schätze: } P[d \in c_k | d \text{ hat } \bar{X}] = \frac{P[d \text{ hat } \bar{X} | d \in c_k] P[d \in c_k]}{P[d \text{ hat } \bar{X}]}$$

$$\sim P[X | d \in c_k] P[d \in c_k]$$

$$= \prod_{i=1}^m P[X_i | d \in c_k] P[d \in c_k] \quad \text{bei Featureunabhängigkeit bzw. Linked Dependence:}$$

$$\frac{P[X | d \in c_k]}{P[X | d \notin c_k]} = \prod_i \frac{P[X_i | d \in c_k]}{P[X_i | d \notin c_k]}$$

$$= \prod_{i=1}^m p_{ik}^{X_i} (1 - p_{ik})^{1 - X_i} p_k \quad \text{mit empirisch zu schätzenden } p_{ik} = P[X_i = 1 | c_k], p_k = P[c_k]$$

$$\Rightarrow \log P[c_k | d] \sim \sum_{i=1}^m X_i \log \frac{p_{ik}}{(1 - p_{ik})} + \sum_{i=1}^m \log(1 - p_{ik}) + \log p_k$$

für binäre Klassifikation mit Quote $P[d \in c_k] / P[d \notin c_k]$ statt $P[\dots]$
weitere Vereinfachung möglich

Informationssysteme SS2004

3-16

Naives Bayes-Verfahren mit Bag-of-Words-Modell

$$\text{Schätze: } P[d \in c_k | d \text{ hat } \bar{f}] \sim P[\bar{f} | d \in c_k] P[d \in c_k]$$

mit Termhäufigkeitsvektor $\bar{f}$

$$= \prod_{i=1}^m P[f_i | d \in c_k] P[d \in c_k] \quad \text{bei Featureunabhängigkeit}$$

$$= \prod_{i=1}^m \binom{\text{length}(d)}{f_i} p_{ik}^{f_i} (1 - p_{ik})^{\text{length}(d) - f_i} p_k$$

mit Binomialverteilung
für jedes Feature

$$\text{bzw. besser: } = \binom{\text{length}(d)}{f_1 f_2 \dots f_m} p_{1k}^{f_1} p_{2k}^{f_2} \dots p_{mk}^{f_m} p_k$$

$$\text{mit } \binom{n}{k_1 k_2 \dots k_m} := \frac{n!}{k_1! k_2! \dots k_m!}$$

mit Multinomialverteilung
der Featurevektoren und
 $\sum_{i=1}^m f_i = \text{length}(d)$

Informationssysteme SS2004

3-17

Beispiel für das naive Bayes-Verfahren (1)

3 Klassen: c1 – Algebra, c2 – Analysis, c3 – Stochastik

8 Terme, 6 Trainingsdokumente d1, ..., d6: je 2 in jeder Klasse

$$\Rightarrow p_1 = 2/6, p_2 = 2/6, p_3 = 2/6$$

	Gruppe	Homomorphismus	Vektor	Integral	Limex	Varianz	Wahrscheinlichkeit	Würfel		Algebra	Analysis	Stochastik
	f1	f2	f3	f4	f5	f6	f7	f8		k=1	k=2	k=3
d1:	3	2	0	0	0	0	0	1	p1k	4/12	0	1/12
d2:	1	2	3	0	0	0	0	0	p2k	4/12	0	0
d3:	0	0	0	3	3	0	0	0	p3k	3/12	1/12	1/12
d4:	0	0	1	2	2	0	1	0	p4k	0	5/12	1/12
d5:	0	0	0	1	1	2	2	0	p5k	0	5/12	1/12
d6:	1	0	1	0	0	0	2	2	p6k	0	0	2/12
									p7k	0	1/12	4/12
									p8k	1/12	0	2/12

Informationssysteme SS2004

3-18

Beispiel für das naive Bayes-Verfahren (2)

Klassifikation von d7: (0 0 1 2 0 0 3 0)

$$P[\tilde{f}|d \in c_k] P[d \in c_k] = \left(\frac{\text{length}(d)}{f_1 f_2 \dots f_m} \right) p_{1k}^{f_1} p_{2k}^{f_2} \dots p_{mk}^{f_m} p_k$$

für k=1 (Algebra): $= \left(\frac{6}{1 \cdot 2 \cdot 3} \right) \left(\frac{3}{12} \right)^1 0^2 0^3 \frac{2}{6} = 0$

für k=2 (Analysis): $= \left(\frac{6}{1 \cdot 2 \cdot 3} \right) \left(\frac{1}{12} \right)^1 \left(\frac{5}{12} \right)^2 \left(\frac{1}{12} \right)^3 \frac{2}{6} = 20 * \frac{25}{12^6}$

für k=3 (Stochastik): $= \left(\frac{6}{1 \cdot 2 \cdot 3} \right) \left(\frac{1}{12} \right)^1 \left(\frac{1}{12} \right)^2 \left(\frac{4}{12} \right)^3 \frac{2}{6} = 20 * \frac{64}{12^6}$

Resultat: Ordne d7 der Klasse C3 (Stochastik) zu

Informationssysteme SS2004

3-19

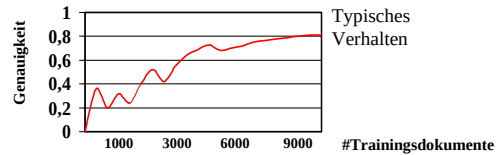
Typisches Verhalten des naiven Bayes-Verfahrens

Reuters Benchmark (siehe trec.nist.gov):

12902 kurze Artikel (Wirtschaftsnachrichten)

aus 90 Kategorien (acq, corn, earn, grain, interest, money-fx, ship, ...)

- Verwende die (bzw. einen Teil der) ältesten 9603 Artikel zum Trainieren des Klassifikators
- Verwende die neuesten 3299 Artikel zur Evaluation der Klassifikationsgenauigkeit



max. Genauigkeit liegt je nach Kategorie zwischen 50 und 90 Prozent

Informationssysteme SS2004

3-20

Verbesserung des naiven Bayes-Verfahrens

- 1) geglättete Schätzung der p_{ik} durch Laplace-Smoothing:

$1/(m + \sum_{d \in C_k} \text{length}(d))$ statt 0 für in den Trainingsdokumenten einer Klasse überhaupt nicht auftretende Features

- 2) Anreicherung des Trainingsmaterials durch unbenannte, automatisch klassifizierte Dokumente zur besseren Schätzung der p_{ik}

mit unterschiedlicher Gewichtung der intellektuell und der automatisch klassifizierten „Trainingsdokumente“

- 3) Berücksichtigung von Abhängigkeiten zwischen Features durch Verallgemeinerung auf Bayessche Netze

Informationssysteme SS2004

3-21

Erweiterung um Semisupervised Learning

Motivation:

- Klassifikator nur so gut wie seine Trainingsdaten
 - Trainingsdaten teuer wegen intellektueller Klassifikation
 - Trainingsdaten sind im Feature-Raum nur dünnbesetzt
- Verwendung zusätzlicher nichtklassifizierter Daten zum impliziten Lernen von Korrelationen

Beispiel:

- Klassifikator für Thema „cars“ wurde auf Dokumenten trainiert, die „car“ enthalten, aber nicht „automobile“.
- In den nichtklassifizierten Daten sind „car“ und „automobile“ stark korreliert.
- Testdokumente enthalten „automobile“, aber nicht „car“.

Informationssysteme SS2004

3-22

Simple Iteratives Labeling

Sei D^K die Menge der Dok. mit bekannten Klassen (Trainingsdaten) und sei D^U die Menge der Dok. mit unbekannten Klassen.

Algorithm:

train classifier with D^K as training data

classify docs in D^U

repeat

re-train classifier with D^K and the now labeled docs in D^U

classify docs in D^U

until labels do not change anymore (or changes are marginal)

Robustheitsproblem:

einige wenige Dokumente aus D^U können

den Klassifikator zu einem „Drift“ verleiten

→ bessere, aber komplexere Iterationsverfahren basierend auf dem Expectation-Maximization-Verfahren für Parameterschätzung

Informationssysteme SS2004

3-23

3.4 Feature-Selektion

Zur Entscheidung zwischen Klassen einer Stufe werden geeignete Features ausgewählt (aus Effizienzgründen und zur Vermeidung von Overfitting).

Beispiel:

Terme wie „Definition“, „Theorem“, „Lemma“ sind gute Diskriminatoren zwischen Arts, Entertainment, Science, etc.; sie sind schlechte Diskriminatoren zwischen den Unterklassen von Mathematics wie z.B. Algebra, Stochastics, etc.

→ Betrachtung statistischer bzw. informationstheoretischer Maße zur Selektion geeigneter Features

Informationssysteme SS2004

3-24

Feature-Selektion auf der Basis der Mutual Information (MI)

Mutual Information (Relative Entropie, Kullback-Leibler-Distanz):

Zur Entscheidung für Klasse c_j wähle diejenigen binären Features X_i (Termvorkommen) mit dem größten Wert von

$$MI(X_i, c_j) = P[X_i \wedge c_j] \log \frac{P[X_i \wedge c_j]}{P[X_i]P[c_j]}$$

oder

$$MI(X_i, c_j) = \sum_{X \in \{X_i, \bar{X}_i\}} \sum_{C \in \{c_j, \bar{c}_j\}} P[X \wedge C] \log \frac{P[X \wedge C]}{P[X]P[C]}$$

und für die Entscheidung bzgl. Klassen $c_1, \dots, c_k$:

$$MI(X_i) = \sum_{j=1}^k P[c_j] MI(X_i, c_j)$$

Berechnung in Zeit $O(n) + O(mk)$
für n Trainingsdokumente,
 m Terme und k Klassen

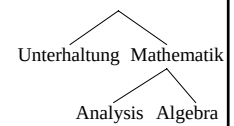
Informationssysteme SS2004

3-25

Beispiel für Feature-Selektion

	Film	Hit	Chart	Theorem	Lines	Integral	Groupe	Vektor
	f1	f2	f3	f4	f5	f6	f7	f8
d1:	1	1	0	0	0	0	0	0
d2:	0	1	1	0	0	0	1	0
d3:	1	0	1	0	0	0	0	0
d4:	0	1	1	0	0	0	0	0
d5:	0	0	0	1	1	1	0	0
d6:	0	0	0	1	0	1	0	0
d7:	0	0	0	0	1	0	0	0
d8:	0	0	0	1	0	1	0	0
d9:	0	0	0	0	0	0	0	1
d10:	0	0	0	1	0	0	1	1
d11:	0	0	0	1	0	1	0	1
d12:	0	0	1	1	1	0	1	0

Klassenbaum:



Trainingsdokumente:

d1, d2, d3, d4
→ Unterhaltung
d5, d6, d7, d8
→ Analysis
d9, d10, d11, d12
→ Algebra

Informationssysteme SS2004

3-26

Beispielrechnung für Feature-Selektion auf der Basis des MI-Maßes

Unterhaltung (d1-d4) vs. Mathematik (d5-d12):

$$MI(\text{Film}) = \frac{2}{12} \log \left[\frac{2/12}{(2/12 * 1/3)} \right] + 0 \log 0 + \frac{2}{12} \log \left[\frac{2/12}{(2/12 * 1/3)} \right] + \frac{8}{12} \log \left[\frac{8/12}{(10/12 * 2/3)} \right]$$

$$MI(\text{Chart}) = \frac{3}{12} \log \left[\frac{3/12}{(4/12 * 1/3)} \right] + \frac{1}{12} \log \left[\frac{1/12}{(4/12 * 2/3)} \right] + \frac{1}{12} \log \left[\frac{1/12}{(8/12 * 1/3)} \right] + \frac{7}{12} \log \left[\frac{7/12}{(8/12 * 2/3)} \right]$$

$$MI(\text{Theorem}) = 0 \log 0 + \frac{6}{12} \log \left[\frac{6/12}{(6/12 * 2/3)} \right] + \frac{4}{12} \log \left[\frac{4/12}{(6/12 * 1/3)} \right] + \frac{2}{12} \log \left[\frac{2/12}{(6/12 * 2/3)} \right]$$

Analysis (d5-d8) vs. Algebra (d9-d12):

$$MI(\text{Film}) = 0 \log 0 + 0 \log 0 + \frac{4}{8} \log \left[\frac{4/8}{(8/8 * 1/2)} \right] + \frac{4}{8} \log \left[\frac{4/8}{(8/8 * 1/2)} \right]$$

$$MI(\text{Theorem}) = \frac{3}{8} \log \left[\frac{3/8}{(6/8 * 1/2)} \right] + \frac{3}{8} \log \left[\frac{3/8}{(6/8 * 1/2)} \right] + \frac{1}{8} \log \left[\frac{1/8}{(2/8 * 1/2)} \right] + \frac{1}{8} \log \left[\frac{1/8}{(2/8 * 1/2)} \right]$$

$$MI(\text{Vektor}) = 0 \log 0 + \frac{3}{8} \log \left[\frac{3/8}{(3/8 * 1/2)} \right] + \frac{4}{8} \log \left[\frac{4/8}{(5/8 * 1/2)} \right] + \frac{1}{8} \log \left[\frac{1/8}{(5/8 * 1/2)} \right]$$

Informationssysteme SS2004

3-27

Kapitel 4: Linkanalyse für Autoritäts-Ranking

- 4.1 Page-Rank-Verfahren
- 4.2 Exkurs: Grundlagen aus der Stochastik
- 4.3 HITS-Verfahren
- 4.4 Themenspezifisches Page-Rank-Verfahren

Informationssysteme SS2004

4-1

Verbessertes Ranking durch Autoritäts-Scores

Ziel:

Höheres Ranking von URLs mit hoher Autorität bzgl. Umfang, Signifikanz, Aktualität und Korrektheit von Information
→ verbesserte Präzision von Suchresultaten

Ansätze (mit Interpretation des Web als gerichtetem Graphen G):

- Citation- oder Impact-Rank (q) ~ indegree (q)
- Page-Rank (nach Lawrence Page)
- HITS-Algorithmus (nach Jon Kleinberg)

Kombination von Relevanz- und Autoritäts-Ranking:

- gewichtete Summe mit geeigneten Koeffizienten (Google)
- initiales Relevanz-Ranking und iterative Verbesserung durch Autoritäts-Ranking (HITS)

Informationssysteme SS2004

4-2

4.1 Page-Rank $r(q)$

gegeben: gerichteter Web-Graph $G=(V,E)$ mit $|V|=n$ und Adjazenzmatrix $A: A_{ij} = 1$ falls $(i,j) \in E$, 0 sonst

Idee: $r(q) \approx k \sum_{(p,q) \in G} r(p) / \text{outdegree}(p)$

Def.: $r(q) = \varepsilon / n + (1 - \varepsilon) \sum_{(p,q) \in G} r(p) / \text{outdegree}(p)$ mit $0 < \varepsilon \leq 0.25$

Satz: Mit $A'_{ij} = 1/\text{outdegree}(i)$ falls $(i,j) \in E$, 0 sonst, gilt:

$$\vec{r} = \frac{\varepsilon}{n} \vec{1} + (1 - \varepsilon) A' \vec{r} \Leftrightarrow \vec{r} = \left(\frac{\varepsilon}{n} \vec{1}^T + (1 - \varepsilon) A' \right) \vec{r}$$

d.h. $\vec{r}$ ist Eigenvektor einer modifizierten Transitionsmatrix

Iterative Berechnung von $r(q)$:

- Initialisierung mit $r(q) := 1/n$
- Verbesserung durch Auswerten der rekursiven Definitionsgleichung konvergiert typischerweise mit ca. 100 Iterationen

Informationssysteme SS2004

4-3

4.2 Exkurs: Markov-Ketten

Ein **stochastischer Prozeß** ist eine Familie von

Zufallsvariablen $\{X(t) \mid t \in T\}$.

T heißt Parameterraum, und der Definitionsbereich M der $X(t)$

heißt Zustandsraum. T und M können diskret oder kontinuierlich sein.

Ein stochastischer Prozeß heißt **Markov-Prozeß**, wenn

für beliebige $t_1, \dots, t_{n+1}$ aus dem Parameterraum und

für beliebige $x_1, \dots, x_{n+1}$ aus dem Zustandsraum gilt:

$$P[X(t_{n+1}) = x_{n+1} \mid X(t_1) = x_1 \wedge X(t_2) = x_2 \wedge \dots \wedge X(t_n) = x_n] \\ = P[X(t_{n+1}) = x_{n+1} \mid X(t_n) = x_n]$$

Ein Markov-Prozeß mit diskretem Zustandsraum heißt **Markov-Kette**.

O.B.d.A. werden die natürlichen Zahlen als Zustandsraum gewählt.

Notation für Markov-Ketten mit diskretem Parameterraum:

X_n statt $X(t_n)$ mit $n = 0, 1, 2, \dots$

Informationssysteme SS2004

4-4

Exkurs: Eigenschaften von Markov-Ketten mit diskretem Parameterraum (1)

Die Markov-Kette X_n mit diskretem Parameterraum heißt

homogen, wenn die Übergangswahrscheinlichkeiten $p_{ij} := P[X_{n+1} = j \mid X_n = i]$ unabhängig von n sind

irreduzibel, wenn jeder Zustand von jedem Zustand mit positiver Wahrscheinlichkeit erreichbar ist:

$$\sum_{n=1}^{\infty} P[X_n = j \mid X_0 = i] > 0 \quad \text{für alle } i, j$$

aperiodisch, wenn alle Zustände i die Periode 1 haben, wobei die Periode von i der ggT aller Werte n ist, für die gilt:

$$P[X_n = i \wedge X_k \neq i \text{ für } k = 1, \dots, n-1 \mid X_0 = i] > 0$$

Informationssysteme SS2004

4-5

Exkurs: Eigenschaften von Markov-Ketten mit diskretem Parameterraum (2)

Die Markov-Kette X_n mit diskretem Parameterraum heißt

positiv rekurrent, wenn für jeden Zustand i die Rückkehrwahrscheinlichkeit gleich 1 ist und mittlere Rekurrenzzzeit endlich:

$$\sum_{n=1}^{\infty} P[X_n = i \wedge X_k \neq i \text{ für } k = 1, \dots, n-1 \mid X_0 = i] = 1 \\ \sum_{n=1}^{\infty} n P[X_n = i \wedge X_k \neq i \text{ für } k = 1, \dots, n-1 \mid X_0 = i] < \infty$$

ergodisch, wenn sie homogen, irreduzibel, aperiodisch und positiv rekurrent ist.

Informationssysteme SS2004

4-6

Resultate über Markov-Ketten mit diskretem Parameterraum (1)

Für die **n-Schritt-Transitionswahrscheinlichkeiten**

$$p_{ij}^{(n)} := P[X_n = j | X_0 = i] \text{ gilt:}$$

$$p_{ij}^{(n)} = \sum_k p_{ik}^{(n-1)} p_{kj} \text{ mit } p_{ij}^{(1)} := p_{ik}$$

$$= \sum_k p_{ik}^{(n-1)} p_{kj}^{(1)} \text{ für } 1 \leq l \leq n-1$$

in Matrix-Notation: $P^{(n)} = P^n$

Für die **Zustandswahrscheinlichkeiten nach n Schritten**

$$\pi_j^{(n)} := P[X_n = j] \text{ gilt:}$$

$$\pi_j^{(n)} = \sum_i \pi_i^{(0)} p_{ij}^{(n)} \text{ mit Anfangswahrscheinlichkeiten } \pi_i^{(0)}$$

in Matrix-Notation: $\Pi^{(n)} = \Pi^{(0)} P^{(n)}$ (Chapman-Kolmogorov-Gleichung)

Informationssysteme SS2004

4-7

Resultate über Markov-Ketten mit diskretem Parameterraum (2)

Jede homogene, irreduzible, aperiodische Markov-Kette mit endlich vielen Zuständen ist positiv rekurrent und ergodisch.

Für jede ergodische Markov-Kette existieren **stationäre Zustandswahrscheinlichkeiten** $\pi_j := \lim_{n \rightarrow \infty} \pi_j^{(n)}$
Diese sind unabhängig von $\Pi^{(0)}$
und durch das folgende lineare Gleichungssystem bestimmt:

$$\pi_j = \sum_i \pi_i p_{ij} \text{ für alle } j \text{ (Gleichgewichtsgleichungen)}$$

$$\sum_j \pi_j = 1$$

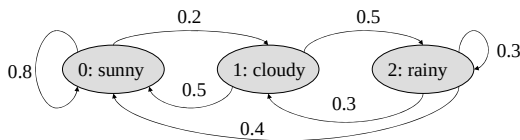
in Matrix-Notation $\Pi = \Pi P$

(mit $1 \times n$ -Vektor Π): $\Pi \bar{1} = 1$

Informationssysteme SS2004

4-8

Beispiel: Markov-Kette



$$\pi_0 = 0.8 \pi_0 + 0.5 \pi_1 + 0.4 \pi_2$$

$$\pi_1 = 0.2 \pi_0 + 0.3 \pi_2$$

$$\pi_2 = 0.5 \pi_1 + 0.3 \pi_2$$

$$\pi_0 + \pi_1 + \pi_2 = 1$$

$$\Rightarrow \pi_0 = 330/474 \approx 0.696$$

$$\pi_1 = 84/474 \approx 0.177$$

$$\pi_2 = 10/79 \approx 0.126$$

Informationssysteme SS2004

4-9

Page-Ranks im Kontext von Markov-Ketten

Modellierung des **Random Walks** eines Web-Surfers durch

- Verfolgen von Hyperlinks mit gleichverteilten Wahrscheinlichkeiten
- „Random Jumps“ mit Wahrscheinlichkeit ϵ

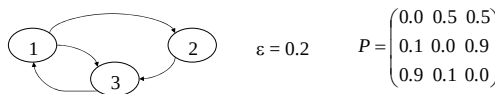
→ ergodische Markov-Kette
Der Page-Rank einer URL ist die stationäre Besuchswahrscheinlichkeit der URL für diese Markov-Kette.
Verallgemeinerungen sind denkbar (z.B. Random Walk mit Back-Button u.ä.)

Kritik am Page-Rank-Verfahren:
Page-Rank ist query-unabhängig und orthogonal zur Relevanz

Informationssysteme SS2004

4-10

Beispiel: Page-Rank-Berechnung



$$P = \begin{pmatrix} 0.0 & 0.5 & 0.5 \\ 0.1 & 0.0 & 0.9 \\ 0.9 & 0.1 & 0.0 \end{pmatrix}$$

$$\Pi^{(0)} \approx \begin{pmatrix} 0.333 \\ 0.333 \\ 0.333 \end{pmatrix}^T \Rightarrow \Pi^{(1)} \approx \begin{pmatrix} 0.333 \\ 0.200 \\ 0.466 \end{pmatrix}^T \Rightarrow \Pi^{(2)} \approx \begin{pmatrix} 0.439 \\ 0.212 \\ 0.346 \end{pmatrix}^T \Rightarrow \Pi^{(3)} \approx \begin{pmatrix} 0.332 \\ 0.253 \\ 0.401 \end{pmatrix}^T$$

$$\Rightarrow \Pi^{(4)} \approx \begin{pmatrix} 0.385 \\ 0.176 \\ 0.527 \end{pmatrix}^T \Rightarrow \Pi^{(5)} \approx \begin{pmatrix} 0.491 \\ 0.244 \\ 0.350 \end{pmatrix}^T$$

$$\pi_1 = 0.1 \pi_2 + 0.9 \pi_3$$

$$\pi_2 = 0.5 \pi_1 + 0.1 \pi_3$$

$$\pi_3 = 0.5 \pi_1 + 0.9 \pi_2$$

$$\pi_1 + \pi_2 + \pi_3 = 1 \Rightarrow \pi_1 \approx 0.3776, \pi_2 \approx 0.2282, \pi_3 \approx 0.3942$$

Informationssysteme SS2004

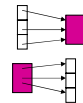
4-11

4.3 HITS-Algorithmus: Hyperlink-Induced Topic Search (1)

Idee:

Bestimme

- gute Inhaltsquellen: **Authorities** (großer indegree)
- gute Linkquellen: **Hubs** (großer outdegree)



Finde

- bessere Authorities mit guten Hubs als Vorgängern
- bessere Hubs mit guten Authorities als Nachfolgern

Für Web-Graph $G=(V,E)$ definiere für Knoten $p, q \in V$

$$\text{Authority-Score } x_q = \sum_{(p,q) \in E} y_p \text{ und}$$

$$\text{Hub-Score } y_p = \sum_{(p,q) \in E} x_q$$

Informationssysteme SS2004

4-12

HITS-Algorithmus (2)

Authority- und Hub-Scores in Matrix-Notation:

$$\bar{x} = A^T \bar{y} \quad \bar{y} = A \bar{x}$$

Iteration mit Adjazenz-Matrix A:

$$\bar{x} := A^T \bar{y} := A^T A \bar{x} \quad \bar{y} := A \bar{x} := A A^T \bar{y}$$

x und y sind also Eigenvektoren von $A^T A$ bzw. $A A^T$.

Intuitive Interpretation:

$M^{(auth)} := A^T A$ ist die Cocitation-Matrix: $M^{(auth)}_{ij}$ ist die Anzahl der Knoten, die auf i und j zeigen

$M^{(hub)} := A A^T$ ist die Bibliographic-Coupling-Matrix: $M^{(hub)}_{ij}$ ist die Anzahl der Knoten, auf die i und j zeigen

Informationssysteme SS2004

4-13

Implementierung des HITS-Algorithmus

- 1) Bestimme hinreichend viele (z.B. 50-200) „Wurzelseiten“ per Relevanz-Ranking (z.B. mittels tf*idf-Ranking)
- 2) Füge alle Nachfolger von Wurzelseiten hinzu
- 3) Füge für jede Wurzelseite max. d Vorgänger hinzu
- 4) Bestimme durch Iteration die Authority- und Hub-Scores dieser „Basismenge“ (von 1000-5000 Seiten) mit Initialisierung $x_q := y_p := 1 / |\text{Basismenge}|$ und Normalisierung nach jedem Schritt
→ konvergiert gegen die Eigenvektoren mit dem betragsgrößten Eigenwert (falls dieser Multiplizität 1 hat)
- 5) Gib Seiten nach absteigend sortierten Authority-Scores aus (z.B. die 10 größten Komponenten von x)

Kritik am HITS-Algorithmus:

Relevanz-Ranking innerhalb der Wurzelmenge bleibt unberücksichtigt

Informationssysteme SS2004

4-14

Verbesserter HITS-Algorithmus

Potentielle Schwachstellen des HITS-Algorithmus:

- irritierende Links (automatisch generierte Links, Spam, etc.)
- Themendrift (z.B. von „Jaguar car“ zu „car“ generell)

Verbesserung:

- Einführung von Kantengewichten:
0 für Links auf demselben Host,
1/k bei k Links von k URLs desselben Host zu 1 URL (xweight)
1/m bei m Links von 1 URL zu m URLs desselben Host (yweight)
- Berücksichtigung von thematischen Relevanzgewichten (z.B. tf*idf)

→ Iterative Berechnung von

$$\text{Authority-Score } x_q = \sum_{(p,q) \in E} y_p * \text{topic score}(p) * xweight(p,q)$$

$$\text{Hub-Score } y_p = \sum_{(p,q) \in E} x_q * \text{topic score}(q) * yweight(p,q)$$

Informationssysteme SS2004

4-15

Bestimmung verwandter URLs

Cocitation-Algorithmus:

- Bestimme bis zu B Vorgänger der gegebenen URL u
- Für jeden Vorgänger p bestimme bis zu BF Nachfolger $\neq u$
- Bestimme unter allen Geschwistern s von u diejenigen mit der größten Anzahl von Vorgängern, die sowohl auf s als auch auf u zeigen (Cocitation-Grad)

Companion-Algorithmus:

- Bestimme geeignete Basismenge um die gegebene URL u herum
- Wende den HITS-Algorithmus auf diese Basismenge an

Informationssysteme SS2004

4-16

Companion-Algorithmus zur Bestimmung verwandter URLs

- 1) Bestimmung der Basismenge: u sowie
 - bis zu B Vorgänger von u und für jeden Vorgänger p bis zu BF Nachfolger $\neq u$ sowie
 - bis zu F Nachfolger von u und für jeden Nachfolger c bis zu FB Vorgänger $\neq u$ mit Elimination von Stop-URLs (wie z.B. www.yahoo.com)
- 2) Duplikateliminierung:
Verschmelze Knoten, die jeweils mehr als 10 Nachfolger haben und mehr als 95 % ihrer Nachfolger gemeinsam haben
- 3) Bestimme Authority-Scores mit dem verbesserten HITS-Algorithmus

Informationssysteme SS2004

4-17

HITS-Algorithmus zur „Community Detection“

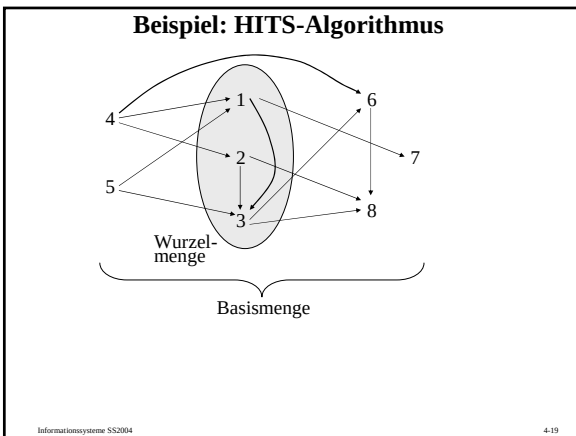
Wurzelmenge kann mehrere Themen bzw. „Communities“ beinhalten, z.B. bei Queries „jaguar“, „Java“ oder „randomized algorithm“

Ansatz:

- Bestimmung der k betragsgrößten Eigenwerte von $A^T A$ und der zugehörigen Eigenvektoren x
- In jedem dieser k Eigenvektoren x reflektieren die größten Authority-Scores eine eng vernetzte „Community“

Informationssysteme SS2004

4-18



4.4 Themenspezifisches Page-Rank-Verfahren

für verschiedene thematische Klassen (Sport, Musik, Jazz, etc.), wobei jede Klasse c_k durch eine Menge T_k einschlägiger Autoritäten charakterisiert ist (z.B. aus Verzeichnissen von yahoo.com, dmoz.org)

Kernidee :
Ändere den Random Walk durch themenspezifische Random-Jump-Wahrscheinlichkeiten für Seiten aus T_k :

$$\vec{r}_k = \varepsilon \vec{p}_k + (1 - \varepsilon) A' \vec{r}_k \quad \text{mit } A'_{ij} = 1/\text{outdegree}(i) \text{ für } (i,j) \in E, 0 \text{ sonst}$$

$$\text{mit } (p_k)_j = 1/|T_k| \text{ für } j \in T_k, 0 \text{ sonst (anstatt } p_j = 1/n)$$

Verfahren:

- 1) Berechne für jede Klasse c_k thematische Page-Rank-Vektoren r_k
- 2) Klassifiziere Query q (inkl. Kontext) bzgl. Klasse c_k
→ Wahrscheinlichkeit $w_k := P[c_k | q]$
- 3) Der Autoritäts-Score von Seite d ist $\sum_k w_k r_k(d)$

Informationssysteme SS2004 4-20

Experimentelle Evaluation: Qualitätsmaße

basierend auf Stanford WebBase (120 Mio. Seiten, Jan. 2001) enthält ca. 300 000 von 3 Mio. Seiten aus dmoz.org aus 16 Themen der obersten Stufe von dmoz.org;
Link-Graph mit 80 Mio. Knoten und der Größe 4 GB
auf 1.5 GHz Dual Athlon mit 2.5 GB Speicher und 500 GB RAID
25 Iterationen für alle 16+1 PR-Vektoren brauchen 20 Stunden
Random-Jump-W. ε gesetzt auf 0.25 (themenspezifisch?)
35 Test-Queries, z.B.: classical guitar, lyme disease, sushi, etc.

Qualitätsmaße: Betrachte Top-k zweier Ranglisten τ_1 und τ_2 ($k=20$)

- **Überlappung** $OSim(\tau_1, \tau_2) = |\text{top}(k, \tau_1) \cap \text{top}(k, \tau_2)| / k$
- **Kendall's τ** $KSim(\tau_1, \tau_2) = \frac{|\{(u, v) | u, v \in U, u \neq v, \text{ und } \tau_1, \tau_2 \text{ haben dieselbe Ordnung von } u, v\}|}{|U| \cdot (|U| - 1)}$
mit $U = \text{top}(k, \tau_1) \cup \text{top}(k, \tau_2)$

Informationssysteme SS2004 4-21

Experimentelle Resultate (1)

- Ranglistenähnlichkeit zwischen den ähnlichsten PR Vektoren:

	OSim	KSim
(Games, Sports)	0.18	0.13
(No Bias, Regional)	0.18	0.12
(Kids&Teens, Society)	0.18	0.11
(Health, Home)	0.17	0.12
(Health, Kids&Teens)	0.17	0.11

- Präzision für Top-10 (# relevante Dok. / 10) von 5 Benutzern:

	Standard	Themenspezifisch
alcoholism	0.12	0.7
bicycling	0.36	0.78
death valley	0.28	0.5
HIV	0.58	0.41
Shakespeare	0.29	0.33
micro average	0.276	0.512

Informationssysteme SS2004 4-22

Experimentelle Resultate (2)

- Top-3 für Query "bicycling"
(klassifiziert auf sports mit W. 0.52, regional mit 0.13, health mit 0.07)

Standard	Recreation	Sports
1 www.RailRiders.com	www.gorp.com	www.multisports.com
2 www.waypoint.org	www.GrownupCamps.com	www.BikeRacing.com
3 www.gorp.com	www.outdoor-pursuits.com	www.CycleCanada.com

- Top-5 für Query-Kontext "blues" (Benutzer wählt Seite aus)
(klassifiziert auf arts mit W. 0.52, shopping mit 0.12, news mit 0.08)

No Bias	Arts	Health
1 news.tucows.com	www.britannia.com	www.baltimorepsych.com
2 www.emusic.com	www.bandhunt.com	www.ncpamd.com/seasonal
3 www.johnhollman.com	www.artistinformation.com	www.ncpamd.com/Women's
4 www.majorleaguebaseball	www.billboard.com	www.wingofmadness.com
5 www.mp3.com	www.soul-patrol.com	www.countrynurse.com

Informationssysteme SS2004 4-23

Persönliche Page-Rank-Werte

Ziel: Effiziente Berechnung und Speicherung auf einzelne Benutzerpräferenzen zugeschnittener Page-Rank-Vektoren

Page-Rank-Gleichung: $\vec{r}_k = \varepsilon \vec{p}_k + (1 - \varepsilon) A' \vec{r}_k$

Theorem:
Seien u_1 und u_2 persönliche Präferenzvektoren (für Random-Jump-Ziele) und seien r_1 und r_2 die zugehörigen Page-Rank-Vektoren. Dann gilt für alle $\alpha_1, \alpha_2 \geq 0$ mit $\alpha_1 + \alpha_2 = 1$:
 $\alpha_1 r_1 + \alpha_2 r_2 = (1 - \varepsilon) A' (\alpha_1 r_1 + \alpha_2 r_2) + \varepsilon (\alpha_1 u_1 + \alpha_2 u_2)$

Korollar:
Für einen Präferenzvektor u mit m von 0 verschiedenen Komponenten und Basisvektoren e_p mit $(e_p)_i = 1$ für $i=p$, 0 für $i \neq p$ gilt:
 $u = \sum_{p=1}^m \alpha_p \cdot e_p$ mit Konstanten $\alpha_1 \dots \alpha_m$
und $r = \sum_{p=1}^m \alpha_p \cdot r_p$ für den persönlichen Page-Rank-Vektor r

Informationssysteme SS2004 4-24

Kapitel 5: Relationenmodell und algebraorientierte Anfragesprachen

5.1 Grundbegriffe des Relationenmodells

5.2 Relationenalgebra

5.3 Erweiterte Relationenalgebra

Informationssysteme SS2004

5-1

5.1 Grundbegriffe des Relationenmodells

Bücher

ISBN	Autor	Titel	...	Kategorie
1111	Grass	Blechtrommel		Roman
2222	Gates	Robin Hood		Märchen
3333	King	Windows 2000		Horror

DB: Relationen
(Tables)

Verkäufe

ISBN	KNr	Datum	...
3333	999	1.4.00	...

Kunden

KNr	Name	Ort	PLZ	Strasse	...
901	Becker	Saarlouis	...		
902	Caesar	Rom			

Informationssysteme SS2004

5-2

Grundbegriffe des Relationenmodells

Bücher

ISBN	Autor	Titel	...	Kategorie
1111	Grass	Blechtrommel		Roman
2222	Gates	Robin Hood		Märchen
3333	King	Windows 2000		Horror

DB: Relationen
(Tables)

Ausprägung

$\subseteq \text{Dom}(\text{ISBN}) \times \text{Dom}(\text{Autor}) \times \dots \times \text{Dom}(\text{Kategorie})$

Verkäufe

ISBN	KNr	Datum	...
3333	999	1.4.00	...

Kunden

KNr	Name	Ort	PLZ	Strasse	...
901	Becker	Saarlouis	...		
902	Caesar	Rom			

Schema

Attribut (Column)

Tupel
(Record, Row)

Informationssysteme SS2004

5-3

Relationenmodell: Definitionen

Definition:

Gegeben sei eine Menge von Wertebereichen primitiver Datentypen $\{D_1, \dots, D_m\}$, die als "**Domains**" bezeichnet werden.

Eine **Relation** R ist ein Paar $R = (s, v)$ mit

- einem Schema $s = \{A_1, \dots, A_n\}$, das aus einer Menge von **Attributen** (**Attributnamen**) besteht und für jedes Attribut A_i einen Domain $\text{dom}(A_i) \in \{D_1, \dots, D_m\}$ festlegt,
- und einer **Ausprägung** (**Wert**) $v \subseteq \text{dom}(A_1) \times \text{dom}(A_2) \times \dots \times \text{dom}(A_n)$

Schema und Ausprägung von R werden mit $\text{sch}(R)$ und $\text{val}(R)$ bezeichnet.

Häufige Schreibweise für Relationenschemata:

$R(A_1, \dots, A_n)$

Bücher (ISBN, Autor, Titel, ...)

Kunden (KNr, Name, Stadt, ...)

Informationssysteme SS2004

5-4

Beispieldatenbank (1)

Kunden

KNr	Name	Stadt	Saldo	Rabatt
1	Lauer	Merzig	-1080.00	0.10
2	Schneider	Homburg	-800.00	0.20
3	Kirsch	Homburg	0.00	0.10
4	Schulz	Merzig	0.00	0.10
5	Becker	Dillingen	0.00	0.05
6	Meier	Saarlouis	-3800.00	0.05

Produkte

PNr	Bez	Gewicht	Preis	Lagerort	Vorrat
1	Papier	2.000	20.00	Homburg	10000
2	Platte	1.000	2500.00	Saarbrücken	400
3	Drucker	5.000	2000.00	Merzig	200
4	Bildschirm	5.000	3000.00	Merzig	80
5	Disketten	0.500	20.00	Homburg	5000
6	Maus	0.250	100.00	Homburg	200
7	Speicher	0.100	200.00	Saarbrücken	2000

Informationssysteme SS2004

5-5

Beispieldatenbank (2)

Bestellungen

BestNr	Monat	Tag	KNr	PNr	Menge	Summe	Status
1	7	16	1	1	100	1800.00	bezahlt
2	7	21	1	1	100	1800.00	bezahlt
3	9	30	1	2	4	9000.00	bezahlt
4	9	30	1	3	1	1800.00	bezahlt
5	9	30	1	4	10	27000.00	bezahlt
6	10	15	1	5	50	900.00	bezahlt
7	10	28	1	6	2	180.00	geliefert
8	11	2	1	7	5	900.00	neu
9	10	26	2	1	100	1600.00	bezahlt
10	11	2	2	5	50	800.00	neu
11	9	28	3	5	50	900.00	bezahlt
12	10	28	3	7	10	1800.00	bezahlt
13	4	15	4	1	50	900.00	bezahlt
14	5	31	6	1	200	3800.00	bezahlt
15	6	30	6	7	10	1900.00	geliefert
16	7	31	6	1	100	1900.00	geliefert

Informationssysteme SS2004

5-6

Einschränkungen der "1. Normalform"

Studenten

Name	Fach	... Systemkenntnisse
Meier	Informatik	{Oracle, mySQL, PHP}
Schmidt	Informatik	{Java, Oracle}
Kunz	Informatik	{Oracle}
Müller	Mathematik	∅

ist im („flachen“) Relationenmodell nicht erlaubt !

Informationssysteme SS2004

5-7

Repräsentation in "1. Normalform"

Studenten

Name	Fach	... Systemkenntnis
Meier	Informatik	Oracle
Meier	Informatik	mySQL
Meier	Informatik	PHP
Schmidt	Informatik	Java
Schmidt	Informatik	Oracle
Kunz	Informatik	Oracle
Müller	Mathematik	---

oder besser:

Studenten

Name	Fachbereich	...
Meier	Informatik	
Schmidt	Informatik	
Kunz	Informatik	
Müller	Mathematik	

Kenntnisse

Name	Systemkenntnis
Meier	Oracle
Meier	mySQL
Meier	PHP
Schmidt	Java
Schmidt	Oracle
Kunz	Oracle

Informationssysteme SS2004

5-8

Integritätsbedingungen des Relationenmodells (1)

Definitionen:

Eine Attributmenge $K \subseteq \text{sch}(R)$ einer Relation R heißt **Schlüsselkandidat** wenn zu jedem Zeitpunkt für je zwei Tupel $t_1, t_2 \in \text{val}(R)$ gelten muß:

$$t_1.K = t_2.K \Rightarrow t_1 = t_2$$

und wenn es keine echte Teilmenge von K gibt, die diese Eigenschaft hat.

(Dabei bedeutet $t_1.K = t_2.K$: $\forall A \in K: t_1.A = t_2.A$.)

Ein Attribut einer Relation R , das in mind. einem Schlüsselkandidaten vorkommt, heißt **Schlüsselattribut**.

Der **Primärschlüssel** einer Relation ist ein Schlüsselkandidat, der explizit ausgewählt wird.

Attributmenge $F \subseteq \text{sch}(S)$ einer Relation S ist ein **Fremdschlüssel** in S , wenn es eine Relation R gibt, in der F Primärschlüssel ist.

Ein Attribut A eines Tupels t hat einen **Nullwert**, wenn der Wert $t.A$ undefiniert oder unbekannt ist.

Informationssysteme SS2004

5-9

Integritätsbedingungen des Relationenmodells (2)

Primärschlüsselbedingung (Entity Integrity):

Für jede Relation muß ein Primärschlüssel festgelegt sein.

Der Primärschlüssel eines Tupels darf niemals den Nullwert annehmen.

Fremdschlüsselbedingung (Referential Integrity):

Für jeden Wert eines Fremdschlüssels in einer Relation R muß in den referenzierten Relationen jeweils ein Tupel mit demselben Wert als Primärschlüssel existieren, oder der Wert des Fremdschlüssels muß der Nullwert sein.

Informationssysteme SS2004

5-10

Integritätsbedingungen des Relationenmodells (2)

Primärschlüsselbedingung (Entity Integrity):

Für jede Relation muß ein Primärschlüssel festgelegt sein.

Der Primärschlüssel eines Tupels darf niemals den Nullwert annehmen.

Fremdschlüsselbedingung (Referential Integrity):

Für jeden Wert eines Fremdschlüssels in einer Relation R muß in den referenzierten Relationen jeweils ein Tupel mit demselben Wert als Primärschlüssel existieren, oder der Wert des Fremdschlüssels muß der Nullwert sein.

Informationssysteme SS2004

5-11

5.2 Relationenalgebra (RA)

Eine Operation der RA hat eine oder mehrere Relationen als Operanden und liefert eine Relation als Ergebnis.
(Abgeschlossenheit der Algebra)

Mengenoperationen:

Für zwei Relationen R, S mit $\text{sch}(R) = \text{sch}(S)$ sind die üblichen Mengenoperationen definiert:

• Vereinigung (Union) $R \cup S$:

$$\text{sch}(R \cup S) = \text{sch}(R)$$

$$\text{val}(R \cup S) = \{t \mid t \in \text{val}(R) \vee t \in \text{val}(S)\}$$

• Durchschnitt (Intersection) $R \cap S$:

$$\text{sch}(R \cap S) = \text{sch}(R)$$

$$\text{val}(R \cap S) = \{t \mid t \in \text{val}(R) \wedge t \in \text{val}(S)\}$$

• Differenz (Difference) $R - S$:

$$\text{sch}(R - S) = \text{sch}(R)$$

$$\text{val}(R - S) = \{t \mid t \in \text{val}(R) \wedge t \notin \text{val}(S)\}$$

Informationssysteme SS2004

5-12

Selektion und Projektion

Selektion σ (Filterung, Auswahl von Zeilen einer Tabelle):

Das Resultat einer Selektion $\sigma[F](R)$ auf einer Relation R ist:

$$\text{sch}(\sigma[F](R)) = \text{sch}(R)$$

$$\text{val}(\sigma[F](R)) = \{t \mid t \in R \wedge F(t)\}$$

Die Menge der möglichen Filterformeln F ist:

- 1) Für Attr. A, B von R mit $\text{dom}(A)=\text{dom}(B)$, Konstanten $c \in \text{dom}(A)$ und Vergleichsoperationen $\theta \in \{=, \neq, <, >, \leq, \geq\}$ sind $A \theta B$ und $A \theta c$ zulässige Filterbedingungen.
- 2) Falls F1 und F2 zulässige Filterbedingungen sind, dann sind auch $F1 \wedge F2$, $F1 \vee F2$, $\neg F1$ und $(F1)$ zulässig.
- 3) Nur die von 1) und 2) erzeugten Filterbedingungen sind zulässig.

Projektion π (Auswahl von Spalten einer Tabelle):

Sei $A \subseteq \text{sch}(R)$. Das Resultat einer Projektion $\pi[A](R)$ ist:

$$\text{sch}(\pi[A](R)) = A$$

$$\text{val}(\pi[A](R)) = \{t \mid \exists r \in \text{val}(R): t.A = r.A\}$$

Informationssysteme SS2004

5-13

Beispiele Selektion

- 1) Finde alle Homburger Kunden.

→ $\sigma[\text{Stadt}='Homburg']$ (Kunden)

KNr	Name	Stadt	Saldo	Rabatt
2	Schneider	Homburg	-800.00	0.20
3	Kirsch	Homburg	0.00	0.10

- 2) Finde alle Homburger Kunden, die einen Rabatt von mind. 15 % haben

→ $\sigma[\text{Stadt}='Homburg' \wedge \text{Rabatt} \geq 0.15]$ (Kunden)

KNr	Name	Stadt	Saldo	Rabatt
2	Schneider	Homburg	-800.00	0.20

Informationssysteme SS2004

5-14

Beispiele Projektion

- 1) Gib alle Produktbezeichnungen aus.

→ $\pi[\text{Bez}]$ (Produkte)

Bez
Papier
Platte
Drucker
Bildschirm
Disketten
Maus
Speicher

- 2) Gib alle Lagerorte von Produkten aus.

→ $\pi[\text{Lagerort}]$ (Produkte)

Lagerort
Homburg
Saarbrücken
Merzig

Informationssysteme SS2004

5-15

(Natural) Join $\bowtie$ und Zuweisung

(Natural) Join $\bowtie$ (Natürlicher Verbund

von Relationen über gleiche Attributnamen und Attributwerte):

Das Resultat von $R \bowtie S$ mit $A=\text{sch}(R)$ und $B=\text{sch}(S)$ ist:

$$\text{sch}(R \bowtie S) = \text{sch}(R) \cup \text{sch}(S)$$

$$\text{val}(R \bowtie S) = \{t \mid \exists r \in \text{val}(R) \exists s \in \text{val}(S): t.A = r.A \wedge t.B = s.B\}$$

Zuweisung:

Seien R, S Relationen mit $\text{sch}(R)=\{A_1, \dots, A_n\}$ und $\text{sch}(S)=\{B_1, \dots, B_n\}$,

so daß für alle i gilt: $\text{dom}(A_i)=\text{dom}(B_i)$.

Die Zuweisung $R := S$ bedeutet: $\text{val}(R) = \text{val}(S)$.

Ausführlicher schreibt man auch $R(A_1, \dots, A_n) := S(B_1, \dots, B_n)$.

Für Ausdrücke $E_1, \dots, E_n$, die über $B_1, \dots, B_n$ und k-stelligen Operatoren

ψ : $\text{dom}(B_{i_1}) \times \dots \times \text{dom}(B_{i_k}) \rightarrow D$ mit skalarem Resultat im Bereich W

bedeutet $R(A_1, \dots, A_n) := S(E_1, \dots, E_n)$:

$$\text{val}(R) = \{t \mid \text{es gibt } s \in \text{val}(S) \text{ und } t.A_i = E_i(s) \text{ für alle } i\},$$

wobei der Wertebereich von E_i gleich $\text{dom}(A_i)$ sein muß.

Informationssysteme SS2004

5-16

Beispiele für Join

- 1) Alle Bestellungen zusammen mit den dazugehörigen Produktdaten.

→ Bestellungen $\bowtie$ Produkt

- 2) $R \bowtie S$

R	S
R1	J
a	1
b	2
c	2
d	3

→

R1	J	S1
b	2	w
b	2	x
c	2	w
c	2	x
d	3	y

Informationssysteme SS2004

5-17

Kartesisches Produkt und Division

Kartesisches Produkt $\times$:

Seien R, S Relationen mit Schemata $A=\text{sch}(R)$ und $B=\text{sch}(S)$.

Sei A' ein Schema, bei dem alle A_i , die auch in B vorkommen, unbenannt

sind in $R.A_i$, und sei B' ein analoges Schema mit Attributnamen $S.A_i$.

Das Resultat von $R \times S$ ist:

$$\text{sch}(R \times S) = A' \cup B'$$

$$\text{val}(R \times S) = \{t \mid \exists r \in \text{val}(R) \exists s \in \text{val}(S): t.A' = r.A \text{ und } t.B' = s.B\}$$

Division $\div$:

Seien R, S Relationen mit $A=\text{sch}(R)$ und $B=\text{sch}(S)$, so daß $B \subset A$.

Das Resultat der Division $R \div S$ ist:

$$\text{sch}(R \div S) = A - B = Q$$

$$\text{val}(R \div S) = \{t \mid \forall s \in \text{val}(S) \exists r \in \text{val}(R): r.B = s.B \wedge t = r.Q\}$$

Intuitive Bedeutung: ein Tupel t ist in $R \div S$ genau dann, wenn

für alle S-Tupel s ein Tupel $\langle t, s \rangle$ in R enthalten ist.

Satz: Für $R(A,B,C)$, $T(A,B)$, $S(C)$ mit $R=T \times S$ gilt: $R \div S = T$.

Informationssysteme SS2004

5-18

Beispiel Division

Kundennummern derjenigen Kunden, die alle überhaupt lieferbaren Produkte irgendwann bestellt haben
 $\rightarrow \pi[\text{KNr}, \text{PNr}] (\text{Bestellungen}) \div \pi[\text{PNr}] (\text{Produkte})$

KNr	PNr	PNr
1	1	1
1	2	2
1	3	3
1	4	4
1	5	5
1	6	6
1	7	7
2	1	
2	5	
3	5	
3	7	
4	1	
6	1	
6	7	

Informationssysteme SS2004

5-19

Rückführung Division auf andere Operationen

Satz:

Seien R und S Relationen mit $A = \text{sch}(R)$ und $B = \text{sch}(S)$ mit $A \supset B$, und sei $Q = \text{sch}(R) - \text{sch}(S)$. Es gilt (bis auf Umbenennungen von Attributen):
 $R \div S = \pi[Q](R) - \pi[Q] ((\pi[Q](R) \times S) - R)$

Beweisskizze (kanonisches Beispiel):

$R := \pi[\text{KNr}, \text{PNr}] (\text{Bestellungen})$

$S := \pi[\text{PNr}] (\text{Produkte})$

$T1 := \pi[\text{KNr}](R) \times S$

$T2 := T1 - R$

$T3 := \pi[\text{KNr}] (T2)$

$T4 := \pi[\text{KNr}] (R) - T3$

alle überhaupt möglichen Bestellungen

potentiell mögliche,

aber nicht erfolgte Bestellungen

Kunden, die nicht alle Prod. bestellt haben

Kunden, die alle Produkte bestellt haben

Informationssysteme SS2004

5-20

θ -Join

θ -Join:

Seien R, S Relationen mit $\text{sch}(R) \cap \text{sch}(S) = \emptyset$. Seien $A \subseteq \text{sch}(R)$ und $B \subseteq \text{sch}(S)$, und sei θ eine der Operationen $=, \neq, <, >, \leq, \geq$.

Das Resultat des θ -Joins $R \bowtie_{\theta} [A \theta B] S$ ist:

$\text{sch}(R \bowtie_{\theta} [A \theta B] S) = \text{sch}(R \times S) = \text{sch}(R) \cup \text{sch}(S)$

$\text{val}(R \bowtie_{\theta} [A \theta B] S) = \text{val}(\sigma[A \theta B](R \times S))$

Beispiele:

1) alle Kunden, die an einem Lagerort wohnen.

$\rightarrow \pi[\text{KNr}, \text{Name}, \text{Stadt}, \dots] (\text{Kunden} \bowtie_{\theta} [\text{Stadt} = \text{Lagerort}] \text{Produkte})$

2) alle bisherigen Bestellungen, die den momentanen Vorrat erschöpfen würden.

$\rightarrow B(\text{BestNr}, \text{KNr}, B.\text{PNr}, \dots) := \text{Bestellungen}(\text{BestNr}, \text{KNr}, \text{PNr}, \dots)$

$\pi[\text{BestNr}, \dots] (B \bowtie_{\theta} [B.\text{PNr} = \text{PNr} \wedge \text{Menge} >= \text{Vorrat}] \text{Produkte})$

3) alle Paare von Kunden, die in derselben Stadt wohnen.

$\rightarrow K1(K1.\text{KNr}, K1.\text{Name}, \dots) := \text{Kunden}(\text{KNr}, \text{Name}, \dots)$

$K2(K2.\text{KNr}, K2.\text{Name}, \dots) := \text{Kunden}(\text{KNr}, \text{Name}, \dots)$

$\pi[K1.\text{KNr}, K2.\text{KNr}]$

$(K1 \bowtie_{\theta} [K1.\text{Stadt} = K2.\text{Stadt} \wedge K1.\text{KNr} < K2.\text{KNr}] K2)$

Informationssysteme SS2004

5-21

Outer Joins

Seien R, S Relationen mit $A = \text{sch}(R)$, $B = \text{sch}(S)$ und $J = \text{sch}(R) \cap \text{sch}(S)$. Bezeichne ferner ω den Nullwert.

Das Resultat des **Outer Joins** $R \bowtie^* S$ ist:

$\text{sch}(R \bowtie^* S) = \text{sch}(R) \cup \text{sch}(S)$

$\text{val}(R \bowtie^* S) = \text{val}(R \bowtie S) \cup$

$\{t \mid \exists r \in \text{val}(R) : t.A = r.A \wedge \neg (\exists s \in \text{val}(S) : t.J = s.J) \wedge t.(B-J) = \omega\} \cup$

$\{t \mid \exists s \in \text{val}(S) : t.B = s.B \wedge \neg (\exists r \in \text{val}(R) : t.J = r.J) \wedge t.(A-J) = \omega\}.$

Left Outer Join:

$\text{sch}(R \bowtie^*_L S) = \text{sch}(R) \cup \text{sch}(S)$

$\text{val}(R \bowtie^*_L S) = \text{val}(R \bowtie S) \cup$

$\{t \mid \exists r \in \text{val}(R) : t.A = r.A \wedge \neg (\exists s \in \text{val}(S) : t.J = s.J) \wedge t.(B-J) = \omega\}$

Right Outer Join:

$\text{sch}(R \bowtie^*_R S) = \text{sch}(R) \cup \text{sch}(S)$

$\text{val}(R \bowtie^*_R S) = \text{val}(R \bowtie S) \cup$

$\{t \mid \exists s \in \text{val}(S) : t.B = s.B \wedge \neg (\exists r \in \text{val}(R) : t.J = r.J) \wedge t.(A-J) = \omega\}.$

Informationssysteme SS2004

5-22

Beispiel Outer Join

Gib alle Kunden mit 5 % Rabatt zusammen mit ihren Bestellungen aus, und zwar auch, wenn für einen Kunden gar keine Bestellungen vorliegen.
 $\sigma[\text{Rabatt} = 0.05] (\text{Kunden} \bowtie^* \text{Bestellungen})$

KNr	Name	Stadt	Saldo	Rabatt	BestNr	...
5	Becker	Dillingen	0.00	0.05	ω	...
6	Meier	Saarlouis	-3800.00	0.05	14	...
6	Meier	Saarlouis	-3800.00	0.05	15	...
6	Meier	Saarlouis	-3800.00	0.05	16	...

Informationssysteme SS2004

5-23

Äquivalenzregeln ("Rechenregeln") der RA

Kommutativitätsregeln:

1) $\pi[R1] \sigma[P] (R) = \sigma[P] \pi[R1] (R)$ falls P nur R1-Attribute enthält

2) $R \bowtie S = S \bowtie R$

Assoziativitätsregeln:

3) $R \bowtie (S \bowtie T) = (R \bowtie S) \bowtie T$

Idempotenzregeln:

4) $\pi[R1] (\pi[R2] (R)) = \pi[R1] (R)$ falls $R1 \subseteq R2$

5) $\sigma[P1] (\sigma[P2] (R)) = \sigma[P1 \wedge P2] (R)$

Distributivitätsregeln:

6) $\pi[R1] (R \cup S) = \pi[R1](R) \cup \pi[R1](S)$

7) $\sigma[P] (R \cup S) = \sigma[P](R) \cup \sigma[P](S)$

8) $\sigma[P] (R \bowtie S) = \sigma[P](R) \bowtie S$ falls P nur R-Attribute enthält

9) $\pi[R1, S1](R \bowtie S) = \pi[R1](R) \bowtie \pi[S1](S)$ falls Joinattribute $\subseteq R1 \cup S1$

10) $R \bowtie (S \cup T) = (R \bowtie S) \cup (R \bowtie T)$

Invertierungsregeln:

11) $\pi[\text{sch}(R)] (R \bowtie^* S) = R$

Informationssysteme SS2004

5-24

Äquivalenzregeln ("Rechenregeln") der RA

Kommutativitätsregeln:

1) $\pi[R1] \sigma[P](R) = \sigma[P] \pi[R1](R)$ falls P nur R1-Attribute enthält

2) $R \bowtie S = S \bowtie R$

Assoziativitätsregeln:

3) $R \bowtie (S \bowtie T) = (R \bowtie S) \bowtie T$

Idempotenzregeln:

4) $\pi[R1] (\pi[R2](R)) = \pi[R1](R)$ falls $R1 \subseteq R2$

5) $\sigma[P1] (\sigma[P2](R)) = \sigma[P1 \wedge P2](R)$

Distributivitätsregeln:

6) $\pi[R1] (R \cup S) = \pi[R1](R) \cup \pi[R1](S)$

7) $\sigma[P] (R \cup S) = \sigma[P](R) \cup \sigma[P](S)$

8) $\sigma[P] (R \bowtie S) = \sigma[P](R) \bowtie S$ falls P nur R-Attribute enthält

9) $\pi[R1, S1](R \bowtie S) = \pi[R1](R) \bowtie \pi[S1](S)$ falls Joinattribute $\subseteq R1 \cup S1$

10) $R \bowtie (S \cup T) = (R \bowtie S) \cup (R \bowtie T)$

Invertierungsregeln:

11) $\pi[\text{sch}(R)] (R \parallel^* S) = R$

Informationssysteme SS2004

5-25

Ausdrucks-mächtigkeit der RA

Die Menge der relationalen algebraischen Ausdrücke über einer Menge von Relationen $R_1, \dots, R_n$ ist wie folgt definiert:

(i) $R_1, \dots, R_n$ sind Ausdrücke.

(ii) Wenn R, S, T, Q Ausdrücke sind, F eine Filterformel über $\text{sch}(R)$ ist

$A \subseteq \text{sch}(R)$, $\text{sch}(S) = \text{sch}(T)$ und $\text{sch}(R) \supseteq \text{sch}(Q)$ gilt, dann sind

$\sigma[F](R)$, $\pi[A](R)$, $R \bowtie S$, $R \times S$, $R \parallel^* S$, $S \cap T$, $S \cup T$, $S - T$, $R \div Q$ auch Ausdrücke.

(iii) Nur die von (i) und (ii) erzeugten Ausdrücke sind RA-Ausdrücke.

Satz:

$\times, \pi, \sigma, \cup$ und $\div$ bilden eine minimale Menge von Operationen, mit denen sich alle Operationen der RA ausdrücken lassen.

Eine Anfragesprache heißt *relational vollständig*, wenn sich damit alle Anfragen der (minimalen) Relationalenalgebra ausdrücken lassen.

Informationssysteme SS2004

5-26

5.3 Erweiterte Relationalenalgebra

Definition:

Eine **Multimenge** (engl. multiset, bag) M über einer Grundmenge G ist eine Abbildung $M: G \rightarrow \mathbb{N}_0$.

$M(x)$ wird als die Häufigkeit von x in M bezeichnet.

Definition:

Eine **Multirelation** R ist ein Paar $R = (s, v)$ mit einem Schema $s = \{A_1, \dots, A_n\}$ und einer Ausprägung v , die eine Multimenge über $\text{dom}(A_1) \times \dots \times \text{dom}(A_n)$ ist.

Beispiel:

Name	Stadt	Rabatt
Lauer	Merzig	0.10
Schneider	Homburg	0.20
Schneider	Homburg	0.20
Schulz	Merzig	0.10
Schulz	Merzig	0.05
Meier	Saarlouis	0.05

Die Funktion χ konvertiert eine Multirelation R in eine Relation $\chi(R)$ mit:
 $\text{val}(\chi(R)) = \{t \mid R(t) > 0\}$.

Informationssysteme SS2004

5-27

Operationen auf Multimengen und -relationen

Für Multirelationen R, S mit $\text{sch}(R) = \text{sch}(S)$ sind:

Vereinigung $R \cup_+ S$:

$\text{sch}(R \cup_+ S) = \text{sch}(R)$, $\text{val}(R \cup_+ S) = t \mapsto R(t) + S(t)$

Durchschnitt $R \cap_+ S$:

$\text{sch}(R \cap_+ S) = \text{sch}(R)$, $\text{val}(R \cap_+ S) = t \mapsto \min(R(t), S(t))$

Differenz $R -_+ S$:

$\text{sch}(R -_+ S) = \text{sch}(R)$, $\text{val}(R -_+ S) = t \mapsto R(t) - S(t)$ falls $R(t) \geq S(t)$, 0 sonst

Das Resultat der **Selektion** $\sigma_+[F](R)$ auf einer Multirelation R ist:

$\text{sch}(\sigma_+[F](R)) = \text{sch}(R)$,

$\text{val}(\sigma_+[F](R)) = t \mapsto R(t)$ falls $R(t) \wedge F(t)$, 0 sonst

Das Resultat der **Projektion** $\pi_+[A](R)$ auf einer Multirelation R mit $A \subseteq \text{sch}(R)$ ist:

$\text{sch}(\pi_+[A](R)) = A$,

$\text{val}(\pi_+[A](R)) = t \mapsto \sum \{R(r) \mid r \in \text{val}(R) \text{ und } r.A = t.A\}$

Informationssysteme SS2004

5-28

Aggregation und Gruppierung

Aggregation α_+ (Zusammenfassen von Zeilen einer Tabelle):

Sei $A \in \text{sch}(R)$ und sei f eine Funktion, die Multimengen über $\text{dom}(A)$ in einen Wertebereich W abbildet (z.B. max, min, sum, count, median).

Das Resultat der Aggregation $\alpha_+[A, f](R)$ auf der Multirelation R ist:

$\text{sch}(\alpha_+[A, f](R)) = A'$ mit $\text{dom}(A') = W$

$\text{val}(\alpha_+[A, f](R)) = f(\pi_+[A](R))$

Gruppierung γ_+ (Zusammenfassen von Äquivalenzklassen von Zeilen):

Sei $X \subseteq \text{sch}(R)$, $A \in \text{sch}(R)$, und sei f eine Funktion, die Multimengen über $\text{dom}(A)$ in einen Wertebereich W abbildet.

Das Resultat einer Gruppierung $\gamma_+[X, A, f](R)$ auf der Multirelation R ist:

$\text{sch}(\gamma_+[X, A, f](R)) = X \cup \{A'\}$ mit $\text{dom}(A') = W$

$\text{val}(\gamma_+[X, A, f](R)) = \{t \mid \text{es gibt eine Äquivalenzklasse } G \text{ von } \pi_+[X](R) \text{ unter der Wertgleichheit und } t.X \text{ ist der Wert der Tupel in } G \text{ und } t.A' = f(\pi_+[A](G))\}$

Informationssysteme SS2004

5-29

Beispielanfragen auf Multirelationen (1)

PNr	Bez	Gewicht	Preis	Lagerort	Vorrat
1	Papier	2.000	20.00	Homburg	10000
2	Platte	1.000	2500.00	Saarbrücken	400
3	Drucker	5.000	2000.00	Merzig	200
4	Bildschirm	5.000	3000.00	Merzig	80
5	Disketten	0.500	20.00	Homburg	5000
6	Maus	0.250	100.00	Homburg	200
7	Speicher	0.100	200.00	Saarbrücken	2000

1) Produkte unter 50 DM sowie

deren Lagerorte und Preise:

$\sigma_+[\text{Preis} < 50.00](\pi_+[\text{Lagerort}, \text{Preis}](\text{Produkte}))$

Lagerort	Preis
Homburg	20.00
Homburg	200.00

2) Gesamtstückzahl (aller Produkte)

über alle Lager:

Resultat(GesamtVorrat): $= \alpha_+[\text{Vorrat}, \text{sum}](\text{Produkte})$

GesamtVorrat
17880

Informationssysteme SS2004

5-30

Beispielanfragen auf Multirelationen (2)

- 3) Bestimme für jedes Lager die Gesamtstückzahl aller dort gelagerten Produkte:
 Resultat (Lagerort, Gesamtbestand) :=
 $\gamma_+ [\{ \text{Lagerort} \}, \text{Vorrat}, \text{sum}] (\text{Produkte})$

Lagerort	Gesamtbestand
Homburg	15200
Saarbrücken	2400
Merzig	280

- 4) Bestimme für jedes Lager die Gesamtkapitalbindung
 (Stückzahl * Preis)
 aller dort gelagerten Produkte:
 $P1 := \pi_+ [\text{Lagerort}, \text{Preis}, \text{Vorrat}] (\text{Produkte})$
 $P2 (\text{Lagerort}, \text{Wert}) := P1(\text{Lagerort}, \text{Preis} * \text{Vorrat})$
 Resultat (Lagerort, Kapitalbindung) :=
 $\gamma_+ [\{ \text{Lagerort} \}, \text{Wert}, \text{sum}] (P2)$

Lagerort	Kapitalbindung
Homburg	320 000
Saarbrücken	1 400 000
Merzig	640 000

Informationssysteme SS2004

5-31

Weitere Erweiterungen der RA

Transitive Hülle R^+ einer binären Relation R:

Sei $R(A, B)$ eine binäre Relation mit $\text{dom}(A) = \text{dom}(B)$.

Das Resultat der transitiven Hülle R^+ ist:

$\text{sch}(R^+) = \text{sch}(R)$

$\text{val}(R^+)$ ist die kleinste Menge, für die gilt:

- (i) für jedes $r \in R$ gilt $r \in R^+$ und
- (ii) für $t \in R^+$ und $r \in R$ mit $t.B = r.A$ gilt $(t.A, r.B) \in R^+$

Beispiel: Flugverbindungen := $(\pi[\text{Abflugort}, \text{Zielort}] (\text{Flüge}))^+$

Flüge

FlugNr	Abflugort	Zielort	...
LH58	Frankfurt	Chicago	
AA371	Chicago	Phoenix	
DA77	Phoenix	Yuma	
AA70	Frankfurt	Dallas	
AA351	Dallas	Phoenix	
UA111	Chicago	Dallas	

Flugverbindungen

Abflugort	Zielort
Frankfurt	Chicago
Frankfurt	Dallas
Frankfurt	Phoenix
Frankfurt	Yuma
...	...

Informationssysteme SS2004

5-32

Kapitel 6: Logikorientierte Anfragesprachen

6.1 Grundlagen aus der Logik

6.1.1 Aussagenlogik

6.1.2 Prädikatenlogik 1. Ordnung

6.2 Domain-Relationenkalkül (DRK)

6.3 Tupel-Relationenkalkül (TRK)

6.4 Mächtigkeit relationaler Anfragesprachen

6.5 Datalog: Deduktionsregeln als Anfragesprache

6.1 Überblick: Dualismus der Logik

Syntax

(beweistheoretische Sicht):

Formeln $F1, \dots, F_n$
mit Aussagenvariablen,
Prädikat-/Funktionssymbolen

Ableitungsregeln
(Beweis-/Deduktionsregeln):
 $F1, \dots, F_n \vdash G$

Semantik

(modelltheoretische Sicht):

Interpretation ψ in
(mathematischer) Struktur S

Gültigkeit in S unter ψ
(G ist in S wahr):
 $\psi \models G$

idealerweise:

wenn G ableitbar, dann G wahr (Korrektheit)

wenn G wahr, dann G ableitbar (Vollständigkeit)

Aussagenlogik: Syntax

Definition:

Eine atomare **Formel** der Aussagenlogik ist eine Aussagenkonstante True oder False oder eine Aussagevariable der Form A_i ($i=1, 2, \dots$). Die Menge der Formeln der Aussagenlogik ist:

- (i) Jede atomare Formel ist eine Formel.
- (ii) Wenn F, G Formeln sind, dann auch (F) , $\neg F$, $F \wedge G$, $F \vee G$ sowie $F \Rightarrow G$ (kurz für $\neg F \vee G$) und $F \Leftrightarrow G$ (kurz für $(F \wedge G) \vee (\neg F \wedge \neg G)$).

Notation und Präzedenzen:

Statt $\neg F$ schreibt man auch $\overline{F}$.

$\neg$ bindet stärker als $\wedge$ und $\vee$, und $\wedge$ und $\vee$ binden stärker als $\Rightarrow$, $\Leftrightarrow$.

Aussagenlogik: Semantik

Definition:

Sei F eine Formel der Aussagenlogik mit atomarer Formelmengende D .

Eine **passende Struktur** S zu F ist eine Menge P von Aussagen, so daß jede Variable in D einer dieser Aussagen zugeordnet werden kann.

Eine **Interpretation** von F ist eine Abbildung $\psi: D \rightarrow P$, für die gilt $\psi(\text{True})=1$, $\psi(\text{False})=0$, $\psi(A_i)=1$ (0), falls $\psi(A_i)$ in S wahr (falsch) ist. ψ wird auf beliebige Formeln F , G , usw. über D fortgesetzt:

(i) $\psi((F)) = \psi(F)$

(ii) $\psi(F \wedge G) = 1$ falls $\psi(F) = \psi(G) = 1$, 0 sonst

(iii) $\psi(F \vee G) = 1$ falls $\psi(F) = 1$ oder $\psi(G) = 1$, 0 sonst

(iv) $\psi(\neg F) = 1$ falls $\psi(F) = 0$, 0 sonst.

Aussagenlogik: Modelle und Folgerungen

Definition:

Sei F eine Formel und ψ eine Interpretation von F .

Wenn $\psi(F)=1$ ist, heißt ψ **Modell** von F ($\psi \models F$ oder $\models_{\psi} F$).

F heißt **erfüllbar**, falls F mindestens ein Modell hat, sonst **unerfüllbar**.

F heißt **(allgemein-)gültig** oder **Tautologie**, falls jede Interpretation in einer zu F passenden Struktur ein Modell von F ist.

Satz: F ist Tautologie genau dann, wenn $\neg F$ unerfüllbar ist

Definition:

G heißt **Folgerung** von $F_1, \dots, F_k$ ($F_1, \dots, F_k \models G$), wenn für jedes ψ gilt: falls ψ Modell von $F_1, \dots, F_k$ ist, dann ist ψ auch Modell von G .

F und G heißen **äquivalent** ($F \equiv G$), falls für jedes ψ gilt: $\psi(F) = \psi(G)$.

Satz: G ist Folgerung von $F_1, \dots, F_k$ g.d.w. $(F_1 \wedge F_2 \wedge \dots \wedge F_k) \Rightarrow G$ eine Tautologie ist. F und G sind äquivalent g.d.w. F Folgerung von G ist und umgekehrt.

Aussagenlogik: Beispiele

1) F1: $T \Rightarrow \neg P$, F2: $P \wedge \neg T$ bzw. F: $(T \Rightarrow \neg P) \wedge (P \wedge \neg T)$

$\psi(T)$ = „7 ist durch 2 teilbar“, $\psi(P)$ = „7 ist eine Primzahl“

2) F: $(S \Rightarrow \neg R) \wedge (\neg R \Rightarrow S) \wedge (A \Rightarrow \neg S) \wedge (A)$

$\psi(S)$ = „die Sonne scheint“, $\psi(R)$ = „es regnet“ (wahr),

$\psi(A)$ = „es ist April“

3) a) $\neg(\neg F) \Leftrightarrow F$

b) $(X \wedge Y) \vee (\neg X \wedge Z)$

c) $F \wedge \neg F$

Aussagenlogik: Deduktionskalkül

Ein einfacher Deduktionskalkül:

- (i) $\vdash F \Rightarrow (F \Rightarrow F)$
- (ii) $\vdash (F \Rightarrow (G \Rightarrow H)) \Rightarrow ((F \Rightarrow G) \Rightarrow (F \Rightarrow H))$
- (iii) $\vdash (\neg F \Rightarrow \neg G) \Rightarrow (G \Rightarrow F)$
- (iv) $F \Leftrightarrow G, H \vdash H[F/G]$
- (v) $F \Rightarrow G, F \vdash G$

Definition:

Sei H eine endl. Formelmenge. F heißt aus H **ableitbar** ($H \vdash F$), wenn es eine endl. Sequenz $F_0, F_1, \dots, F_n$ von Formeln gibt mit $F_n = F$, so daß für alle F_i gilt: F_i ist Element von H , F_i ist eine der Formeln (i) bis (iii) oder F_i ist aus $F_0, \dots, F_{i-1}$ mittels (iv) oder (v) abgeleitet. Für $H = \emptyset$ heißt F **Theorem** des Deduktionskalküls.

Satz: Der einfache Deduktionskalkül ist korrekt und vollständig, d.h.:

$H \vdash F$ genau dann, wenn $H \models F$
(und $\vdash F$ genau dann, wenn $\models F$)

Aussagenlogik: alternative Kalküle bzw. Äquivalenzregeln

$\neg\neg F \Leftrightarrow F$	(Doppelnegation)
$F \wedge F \Leftrightarrow F$	(Idempotenz)
$F \vee F \Leftrightarrow F$	
$F \wedge G \Leftrightarrow G \wedge F$	(Kommutativität)
$F \vee G \Leftrightarrow G \vee F$	
$F \wedge (G \wedge H) \Leftrightarrow (F \wedge G) \wedge H$	(Assoziativität)
$F \vee (G \vee H) \Leftrightarrow (F \vee G) \vee H$	
$F \wedge (F \vee G) \Leftrightarrow F$	(Absorption)
$F \vee (F \wedge G) \Leftrightarrow F$	
$F \wedge (G \vee H) \Leftrightarrow (F \wedge G) \vee (F \wedge H)$	(Distributivität)
$F \vee (G \wedge H) \Leftrightarrow (F \vee G) \wedge (F \vee H)$	
$\neg (F \wedge G) \Leftrightarrow (\neg F \vee \neg G)$	(Gesetz von de Morgan)
$\neg (F \vee G) \Leftrightarrow (\neg F \wedge \neg G)$	
$F \vee 1 \Leftrightarrow 1$	(Tautologieregeln)
$F \wedge 1 \Leftrightarrow F$	
$F \vee 0 \Leftrightarrow F$	(Unerfüllbarkeitsregeln)
$F \wedge 0 \Leftrightarrow 0$	
$F \vee (\neg F) \Leftrightarrow 1$	(Tertium non datur)
$F \wedge (\neg F) \Leftrightarrow 0$	(Kontradiktion)
$((F \Leftrightarrow G) \wedge H) \Leftrightarrow ((F \Leftrightarrow G) \wedge H[F/G])$	(Substitution)

Beispiel für aussagenlogische Deduktion

„Worin besteht das Geheimnis Ihres Lebens?“:

Wenn ich kein Bier zu einer Mahlzeit trinke, dann habe ich immer Fisch.

Wenn ich Fisch und Bier zur selben Mahlzeit habe, verzichte ich auf Eiscreme.

Wenn ich Eiscreme habe oder Bier meide, dann rühre ich Fisch nicht an.

Modellierung:

Formel G: $(\neg B \Rightarrow F) \wedge ((F \wedge B) \Rightarrow \neg E) \wedge ((E \vee \neg B) \Rightarrow \neg F)$

Vereinfachung, sprich Deduktion (mit Unterstreichung als Negation):

$G \Leftrightarrow (B \vee F) \wedge (\underline{F} \wedge \underline{B} \vee \underline{E}) \wedge ((\underline{E} \vee \neg B) \vee \underline{F})$	Def. Implikation
$\Leftrightarrow (B \vee F) \wedge (\underline{F} \vee \underline{B} \vee \underline{E}) \wedge ((\underline{E} \wedge B) \vee \underline{F})$	de Morgan
$\Leftrightarrow (B \vee F) \wedge (\underline{F} \vee \underline{B} \vee \underline{E}) \wedge (\underline{E} \vee \underline{F}) \wedge (B \vee \underline{F})$	Distributivität
$\Leftrightarrow ((B \vee F) \wedge (B \vee \underline{F})) \wedge ((\underline{F} \vee \underline{B} \vee \underline{E}) \wedge (\underline{E} \vee \underline{F}))$	Komm., Assoz.
$\Leftrightarrow ((B \vee (F \wedge \underline{F})) \wedge ((\underline{F} \vee \underline{E}) \vee (\underline{B} \wedge 0)))$	Distrib., Tautologie
$\Leftrightarrow ((B \vee 0) \wedge ((\underline{F} \vee \underline{E}) \vee 0))$	Kontradikt., Unerf.
$\Leftrightarrow B \wedge (\underline{F} \vee \underline{E})$	Unerfüllbarkeit

Fazit: Trinke zu jeder Mahlzeit Bier, und iss niemals Fisch mit Eiscreme!

Prädikatenlogik: Syntax

Definition:

Gegeben seien **Variablen** x_i , **Prädikatsymbole** P_i der Stelligkeit k_i und **Funktionssymbole** f_i der Stelligkeit l_i . **Terme** sind:

- (i) Jede Variable ist ein Term
- (ii) Wenn $t_1, \dots, t_k$ Terme sind und f_i ein k -stelliges Funktionssymbol ist, dann ist auch $f_i(t_1, \dots, t_k)$ ein Term.

Eine **atomare Formel** hat die Form $P_i(t_1, \dots, t_k)$ mit einem k -stelligen Prädikatsymbol P_i und Termen $t_1, \dots, t_k$.

Formeln der Prädikatenlogik 1. Ordnung sind:

- (i) Jede atomare Formel ist eine Formel.
- (ii) Für Formeln F und G sind auch $\neg F$, $F \wedge G$, $F \vee G$ und (F) Formeln.
- (iii) Für eine Variable x_i und eine Formel F sind auch $\forall x_i (F)$ und $\exists x_i (F)$ Formeln (mit **gebundener** Variable x_i).

Notation und Präzedenzen:

Statt $\forall x \forall y \forall z (F)$ schreibt man auch $\forall x, y, z (F)$.

Junktoren binden stärker als Quantoren.

Alle gebundenen Variablen sind paarweise verschieden.

Prädikatenlogik: Semantik (1)

Definition:

Geg. sei eine Menge von Funktions- und Prädikatsymbolen mit entspr. Stelligkeiten. Eine passende **Struktur** ist ein Tripel $S = (U, \text{fun}, \text{pred})$ mit

- einem **Universum** (einer **Trägermenge**) U von Individuen,
- einer Menge fun von **Funktionen** $f_i: U \times \dots \times U \rightarrow U$ der Stelligkeit l_i , so daß für jedes l_i -stellige Symbol eine entspr. Funktion existiert
- einer Menge pred von **Prädikaten** $P_i: U \times \dots \times U \rightarrow \{0,1\}$ der Stelligkeit k_i , so daß für jedes k_i -stellige Symbol ein entspr. Prädikat existiert.

Eine **Interpretation** ψ einer Formel F in einer passenden Struktur $S = (U, \text{fun}, \text{pred})$ ist eine Abbildung, die U festlegt und jedem Funktions- bzw. Prädikatsymbol in F eine Funktion aus fun bzw. ein Prädikat aus pred zuordnet.

Prädikatenlogik: Semantik (2)

ψ wird auf beliebige Formeln F , G , usw. und Terme fortgesetzt:

(i) $\psi((F)) = \psi(F)$

(ii) $\psi(F \wedge G) = 1$ falls $\psi(F) = \psi(G) = 1$, 0 sonst

(iii) $\psi(F \vee G) = 1$ falls $\psi(F) = 1$ oder $\psi(G) = 1$, 0 sonst

(iv) $\psi(\neg F) = 1$ falls $\psi(F) = 0$, 0 sonst

(v) $\psi(f(t_1, \dots, t_k)) = \psi(f)(\psi(t_1), \dots, \psi(t_k))$

für ein Funktionssymbol f und Terme $t_1, \dots, t_k$,

(vi) $\psi(x) = a$ mit einem beliebigen $a \in U$ für eine (freie) Variable x

(vii) $\psi(P(t_1, \dots, t_k)) = \psi(P)(\psi(t_1), \dots, \psi(t_k))$

für ein Prädikatsymbol P und Terme $t_1, \dots, t_k$,

(viii) $\psi(\forall x (F)) = 1$ falls $\psi(F[x/a]) = 1$ für alle $a \in U$

(ix) $\psi(\exists x (F)) = 1$ falls $\psi(F[x/a]) = 1$ für ein beliebiges $a \in U$.

Prädikatenlogik: Beispiele (1)

- 1) $F: \forall q \exists p \forall x, y \text{ (greater}(p, q) \wedge ((\text{greater}(x, 1) \wedge \text{greater}(y, 1)) \Rightarrow (\neg \text{equal}(\text{mult}(x, y), p))))$

Struktur $S = (\mathbb{N}_0, \{+, \cdot\}, \{>, =\})$.

Interpretation ψ :

$\text{add} \mapsto +, \text{mult} \mapsto \cdot, \text{greater} \mapsto >, \text{equal} \mapsto =,$

$q \mapsto 1, p \mapsto 2, x \mapsto 3, y \mapsto 4$

- 2) $F: K(1, \text{Lauer}, \text{Merzig}) \wedge K(2, \text{Schneider}, \text{Homburg}) \wedge P(1, \text{Papier}) \wedge B(7, 16, 1, 1, 100)$

Struktur: Datenbank mit dem Schema

Kunden (KNr, Name, Stadt),

Produkte (PNr, Bez),

Bestellungen (Monat, Tag, KNr, PNr, Menge).

über dem Universum $\mathbb{N}_0 \cup \Sigma^*$

Interpretation $\psi(F) = 1$, falls es die vier Tupel in der Datenbank gibt,
0 sonst.

Erweiterungen der Prädikatenlogik

Mehrsortige Logik mit mehreren Trägersmengen (Sorten) und einer Signatur für die Prädikate und Funktionen.

Beispiel: $U = \{U_1, U_2\}$ mit $U_1 = \mathbb{R}^n$, $U_2 = \mathbb{R}$ und

Addition: $U_1 \times U_1 \rightarrow U_1$, Multiplikation: $U_1 \times U_2 \rightarrow U_1$,

Skalarprodukt: $U_1 \times U_1 \rightarrow U_2$,

Vektorprodukt: $U_1 \times U_1 \rightarrow U_1$, Nullvektor: $\rightarrow U_1$,

Prädikaten Orthogonal: $U_1 \times U_1 \rightarrow \{0,1\}$ usw.

Prädikate als Argumente von Prädikaten oder Funktionen,
Quantifizierung von Prädikaten oder Funktionen

→ **Prädikatenlogik höherer Ordnung**

Beispiel:

$$\begin{aligned} \forall x,y (H(x,y) \Leftrightarrow \forall P ((R(x,y) \Rightarrow P(x,y)) \wedge \\ \forall u,w,z (P(u,w) \wedge P(w,z) \Rightarrow P(u,z))) \\ \Rightarrow (H(x,y) \Rightarrow P(x,y))) \end{aligned}$$

Prädikatenlogik: Modelle und Folgerungen

Definition:

Sei F eine Formel und ψ eine Interpretation von F .

Wenn $\psi(F)=1$ ist, heißt ψ **Modell** von F ($\psi \models F$ oder $\models_{\psi} F$).

F heißt **erfüllbar**, falls F mindestens ein Modell hat, sonst **unerfüllbar**.

F heißt **(allgemein-)gültig** oder **Tautologie**, falls jede Interpretation in einer zu F passenden Struktur ein Modell von F ist.

Satz: F ist Tautologie genau dann, wenn $\neg F$ unerfüllbar ist

Definition:

G heißt **Folgerung** von $F_1, \dots, F_k$ ($F_1, \dots, F_k \models G$), wenn für jedes ψ gilt: falls ψ Modell von $F_1, \dots, F_k$ ist, dann ist ψ auch Modell von G .

F und G heißen **äquivalent** ($F \equiv G$), falls für jedes ψ gilt: $\psi(F) = \psi(G)$.

Satz: G ist Folgerung von $F_1, \dots, F_k$ g.d.w. $(F_1 \wedge F_2 \wedge \dots \wedge F_k) \Rightarrow G$ eine Tautologie ist. F und G sind äquivalent g.d.w. F Folgerung von G ist und umgekehrt.

Prädikatenlogik: Beispiele (2)

Formeln $F1, \dots, F5$ bzw. Formel $F = F1 \wedge \dots \wedge F5$:

$$F1 = \forall x, y, z \ (G(f(f(x,y),z), f(x,f(y,z))))$$

$$F2 = \forall x \ (G(f(x,e),x))$$

$$F3 = \forall x, y \ (G(x, y) \Leftrightarrow G(y, x))$$

$$F4 = \forall x \ \forall y \ \forall z \ (G(x,y) \wedge G(y,z) \Rightarrow G(x,z))$$

$$F5 = \forall x \ \exists y \ (G(f(x,y), e))$$

Prädikatenlogik: Deduktionskalküle

alle aussagenlogischen Äquivalenzen

Wenn $F \Leftrightarrow G$ und F als Teilformel in H vorkommt, dann ist $H \Leftrightarrow H[F/G]$

$$\neg \forall (F) \Leftrightarrow \exists x (\neg F)$$

$$\neg \exists (F) \Leftrightarrow \forall x (\neg F)$$

$$(\forall x (F)) \wedge G \Leftrightarrow \forall x (F \wedge G) \quad \text{falls } x \text{ in } G \text{ nicht frei vorkommt}$$

$$(\forall x (F)) \vee G \Leftrightarrow \forall x (F \vee G) \quad \text{falls } x \text{ in } G \text{ nicht frei vorkommt}$$

$$(\exists x (F)) \wedge G \Leftrightarrow \exists x (F \wedge G) \quad \text{falls } x \text{ in } G \text{ nicht frei vorkommt}$$

$$(\exists x (F)) \vee G \Leftrightarrow \exists x (F \vee G) \quad \text{falls } x \text{ in } G \text{ nicht frei vorkommt}$$

$$(\forall x (F)) \wedge (\forall y (G)) \Leftrightarrow \forall x (F \wedge G[y/x]), \text{ falls } x \text{ in } G \text{ nicht frei vorkommt}$$

$$(\exists x (F)) \vee (\exists y (G)) \Leftrightarrow \exists x (F \vee G[y/x]), \text{ falls } x \text{ in } G \text{ nicht frei vorkommt}$$

$$\forall x (\forall y (F)) \Leftrightarrow \forall y (\forall x (F))$$

$$\exists x (\exists y (F)) \Leftrightarrow \exists y (\exists x (F))$$

$$\forall x (F) \Rightarrow F[x/a] \text{ für alle Konstanten } a$$

$$((\forall x (F)) \wedge (F \Rightarrow G)) \Rightarrow (\forall x (G))$$

Vollständige und korrekte Kalküle: Resolutionskalkül, Tableaunkalkül

Beispiel für prädikatenlogische Deduktion (1)

1. i) *Informatikstudenten können programmieren.*
2. ii) *Studenten mit guten Mathematiknoten studieren Informatik.*
3. iii) *Studenten, die mit einem Informatiker verwandt sind,*
4. *haben gute Mathematiknoten.*
5. iv) *Alle saarländischen Studenten sind mit Heinz Becker verwandt.*
6. v) *Die Verwandtschaftsbeziehung ist symmetrisch.*
7. vi) *Heinz Becker hat Informatik studiert.*

Universum: Menge aller Studenten

Prädikate:

KannProgrammieren $P(x)$

StudiertInformatik $I(x)$

HatGuteMathenoten $M(x)$

SindVerwandtschaft $V(x,y)$

IstSaarländer $S(x)$

Funktionen und Konstanten:

HeinzBecker $hb()$

Beispiel für prädikatenlogische Deduktion (2)

Formeln:

1. i) $\forall x (I(x) \Rightarrow P(x)) \wedge$
2. ii) $\forall x (M(x) \Rightarrow I(x)) \wedge$
3. iii) $\forall x \forall y ((V(x,y) \wedge I(y)) \Rightarrow M(x)) \wedge$
4. iv) $\forall x (S(x) \Rightarrow V(x,hb)) \wedge$
5. v) $\forall x \forall y ((V(x,y) \Leftrightarrow V(y,x)) \wedge$
6. vi) $I(hb)$

Vereinfachung, sprich Deduktion:

(iii) $\wedge$ (iv) $\wedge$ (vi):

$$\begin{array}{l} \forall x \forall y (I(hb) \wedge (S(x) \Rightarrow V(x,hb)) \wedge (V(x,y) \wedge I(y)) \Rightarrow M(x)) \\ \hline \forall x (I(hb) \wedge (S(x) \Rightarrow V(x,hb)) \wedge (V(x,hb) \wedge I(hb)) \Rightarrow M(x)) \\ \hline \forall x (S(x) \Rightarrow M(x)) \quad (*) \end{array}$$

(*) $\wedge$ (ii) $\wedge$ (i):

$$\begin{array}{l} \forall x ((S(x) \Rightarrow M(x)) \wedge (M(x) \Rightarrow I(x)) \wedge (I(x) \Rightarrow P(x))) \\ \hline \forall x (S(x) \Rightarrow P(x)) \end{array}$$

Fazit: alle Saarländer können programmieren!

6.2 Domain-Relationenkalkül (DRK)

Anfragen sind prädikatenlogische Mengenspezifikationen der Form

$$\{ \langle x_1, \dots, x_n \rangle \mid F(x_1, \dots, x_n) \}$$

wobei F ein prädikatenlogischer Ausdruck über einer Menge von **Domain-Variablen** ist mit $x_1, \dots, x_n$ als einzigen freien Variablen.

Die Menge der zulässigen prädikatenlogischen Ausdrücke F ist:

- 1) Für Variablen $x_1, \dots, x_n$ und eine Relation R mit n Attributen ist $\langle x_1, \dots, x_n \rangle \in R$ (auch $R(x_1, \dots, x_n)$) ein zulässiger Ausdruck.
- 2) Für Variablen x, y , Konstanten c und Vergleichsoperationen $\theta \in \{=, \neq, <, >, \leq, \geq\}$ sind $x \theta y$ und $x \theta c$ zulässige Ausdrücke.
- 3) Falls F_1 und F_2 zulässige Ausdrücke sind, dann sind auch $F_1 \wedge F_2$, $F_1 \vee F_2$, $\neg F_1$ und (F_1) zulässig.
- 4) Falls F ein zulässiger Ausdruck mit einer freien Variable x ist, dann sind auch $\exists x: F(x)$ und $\forall x: F(x)$ zulässige Ausdrücke.
- 5) Falls F ein zulässiger Ausdruck ist, dann ist auch (F) zulässig.
- 6) Nur die durch 1) bis 5) erzeugten Ausdrücke sind zulässig.

Semantik des DRK

Eine **Datenbank** über einem Schema mit Relationen $R_1(A_{11}, \dots, A_{1n_1}), \dots, R_m(A_{m1}, \dots, A_{mn_m})$ ist eine Formelmengende H , die für jede Relation R_i ein Prädikatsymbol R_i verwendet und für jedes Tupel $\langle a_1, \dots, a_{n_i} \rangle \in \text{val}(R_i)$ eine atomare Formel $R_i(a_1, \dots, a_{n_i})$ enthält, bzw. ein Modell von H .

Das Ergebnis einer **Anfrage** $\{\langle x_1, \dots, x_n \rangle \mid F(x_1, \dots, x_n)\}$ über der Datenbank H ist:
 $\{\langle a_1, a_2, \dots, a_n \rangle \mid a_1, a_2, \dots, a_n \in U \text{ und } H \models F[x_1/a_1, x_2/a_2, \dots, x_n/a_n]\}.$

Sonderfall:
wenn F keine freien Variablen enthält, ist das Ergebnis
1 wenn $H \models F$ und 0 sonst.

Beispiele für DRK-Anfragen (1)

- 1) Finden Sie (die Namen) alle(r) Kunden mit negativem Saldo.
→ $\{ \langle k, n, st, sa, r \rangle \mid \langle k, n, st, sa, r \rangle \in \text{Kunden} \wedge sa < 0.0 \}$ bzw.
→ $\{ \langle n \rangle \mid \exists k, st, sa, r: \langle k, n, st, sa, r \rangle \in \text{Kunden} \wedge sa < 0.0 \}$
- 2) Finden Sie die Namen aller Kunden, die eine unbezahlte Bestellung haben, die vor Anfang Oktober erfolgte.
→ $\{ \langle n \rangle \mid \exists k, st, sa, r: \langle \underline{k}, n, st, sa, r \rangle \in \text{Kunden} \wedge$
 $\exists b, m, t, p, me, su, stat: \langle b, m, t, \underline{k}, p, me, su, stat \rangle \in \text{Best.} \wedge$
 $m < 10 \wedge stat \neq \text{'bezahlt'} \}$
- 3) Namen der Homburger Kunden, die seit Sept. ein Prod. aus Homburg geliefert bekommen haben, jeweils mit der Bez. des Produkts.
→ $\{ \langle n, bez \rangle \mid \exists k, st, sa, r: \langle \underline{k}, n, st, sa, r \rangle \in \text{Kunden} \wedge$
 $\exists p, g, pr, l, v: \langle \underline{p}, bez, g, pr, l, v \rangle \in \text{Produkte} \wedge$
 $\exists b, m, t, me, su, stat: \langle b, m, t, \underline{k}, \underline{p}, me, su, stat \rangle \in \text{Best.} \wedge$
 $st = \text{'Homburg'} \wedge mw9 \wedge l = \text{'Homburg'} \}$

Beispiele für DRK-Anfragen (2)

- 4) Kunden, von denen mindestens eine Bestellung registriert ist.
→ $\{ \langle k, n, st, sa, r \rangle \mid \langle \underline{k}, n, st, sa, r \rangle \in \text{Kunden} \wedge$
 $\exists b, m, t, p, me, su, stat: \langle b, m, t, \underline{k}, p, me, su, stat \rangle \in \text{Best.} \}$
- 5) Kunden, von denen keine Bestellung registriert ist.
→ $\{ \langle k, n, st, sa, r \rangle \mid \langle k, n, st, sa, r \rangle \in \text{Kunden} \wedge$
 $\forall b, m, t, k', p, me, su, stat:$
 $\langle b, m, t, k', p, me, su, stat \rangle \in \text{Best.} \Rightarrow k \neq k' \}$
- 6) Kunden, die alle überhaupt lieferbaren Produkte bestellt haben.
→ $\{ \langle k, n, st, sa, r \rangle \mid \langle \underline{k}, n, st, sa, r \rangle \in \text{Kunden} \wedge$
 $\forall p: (\exists bez, g, pr, l, v: \langle \underline{p}, bez, g, pr, l, v \rangle \in \text{Produkte})$
 $\Rightarrow$
 $(\exists b, m, t, me, su, stat: \langle b, m, t, \underline{k}, \underline{p}, me, su, stat \rangle$
 $\in \text{Bestellungen}) \}$

Anfrage-GUI Query-by-Example

Finden Sie die Namen der Homburger Kunden, die seit Anfang Sept. ein Produkt aus Homburg oder Saarbrücken geliefert bekommen haben, dessen Preis über 100 DM liegt, jeweils mit der Bez. des Produkts.

Kunden

KNr	Name	Stadt	Saldo	Rabatt
_55	.print	Homburg		

Bestellungen

BestNr	Monat	Tag	KNr	PNr	Menge	Summe	Status
	>= 9		_55	_33			

Produkte

PNr	Bez	Gewicht	Preis	Lagerort	Vorrat
_33	.print		> 100	Homburg Saarbrücken	

6.3 Tupel-Relationenkalkül (TRK)

Anfragen sind prädikatenlogische Mengenspezifikation der Form

$$\{t \mid F(t)\} \text{ bzw. } \{ \langle t1.A1, \dots, tn.An \rangle \mid F(t1, \dots, tn) \},$$

wobei F eine prädikatenlogische Formel über einer Menge von **Tupelvariablen** ist mit t bzw. $t1, \dots, tn$ als einzigen freien Variablen und $A1, \dots, An$ Attributnamen sind.

Die Menge der zulässigen prädikatenlogischen Ausdrücke F ist:

- 1) Für eine Variable r und eine Relation R ist $r \in R$ (auch $R(r)$) ein zulässiger Ausdruck.
- 2) Für Var. r, s , Attr. A, B mit $\text{dom}(A) = \text{dom}(B)$, Konst. $c \in \text{dom}(A)$ und Vergleichsoperationen $\theta \in \{=, \neq, <, >, \leq, \geq\}$ sind $r.A \theta s.B$ und $r.A \theta c$ zulässige Ausdrücke.
- 3) Falls $F1$ und $F2$ zulässige Ausdrücke sind, dann sind auch $F1 \wedge F2$, $F1 \vee F2$, $\neg F1$ und $(F1)$ zulässig.
- 4) Falls F ein zulässiger Ausdruck mit einer freien Tupelvariable r ist, dann sind auch $\exists r: F(r)$ und $\forall r: F(r)$ zulässige Ausdrücke.
- 5) Falls F ein zulässiger Ausdruck ist, dann ist auch (F) zulässig.
- 6) Nur die aufgrund von 1) bis 5) erzeugten Ausdrücke sind zulässig.

Semantik des TRK

Übersetzung in den DRK:

In einer Anfrage der Form $\{t \mid F(t)\}$ oder $\{<t.B1, \dots, t.Bn> \mid F(t)\}$ wird

- jeder Term der Form $r.A_i$ mit einer Tupelvariablen r durch eine Domain-Variable ra_i ersetzt und
- jeder Term der Form $r \in R$
über einer Relation mit Schema $R(A1, \dots, Am)$
durch einen Term $R(ra_1, \dots, ra_m)$ mit Domain-Variablen $ra_1, \dots, ra_m$.

Beispiele für TRK-Anfragen (1)

- 1) Finden Sie (die Namen) alle(r) Kunden mit negativem Saldo.
→ $\{t \mid t \in \text{Kunden} \wedge t.\text{Saldo} < 0.0\}$ bzw.
 $\{t.\text{Name} \mid t \in \text{Kunden} \wedge t.\text{Saldo} < 0.0\}$
- 2) Finden Sie die Namen aller Kunden, die eine unbezahlte Bestellung haben, die vor Anfang Oktober erfolgte.
→ $\{t.\text{Name} \mid t \in \text{Kunden} \wedge \exists b: b \in \text{Bestellungen} \wedge$
 $t.\text{KNr}=b.\text{KNr} \wedge b.\text{Monat} < 10 \wedge b.\text{Status} \neq \text{'bezahlt'}\}$
- 3) Namen der Homburger Kunden, die seit Sept. ein Prod. aus Homburg geliefert bekommen haben, jeweils mit der Bez. des Produkts.
→ $\{k.\text{Name}, p.\text{Bez} \mid k \in \text{Kunden} \wedge p \in \text{Produkte} \wedge$
 $\exists b: b \in \text{Bestellungen} \wedge$
 $k.\text{Stadt}=\text{'Homburg'} \wedge b.\text{Monat} \geq 9 \wedge p.\text{Lagerort}=\text{'Homburg'} \wedge$
 $k.\text{KNr}=b.\text{KNr} \wedge b.\text{PNr}=p.\text{PNr}\}$

Beispiele für TRK-Anfragen (2)

- 4) Kunden, von denen mindestens eine Bestellung registriert ist.
→ $\{t \mid t \in \text{Kunden} \wedge \exists b: b \in \text{Bestellungen} \wedge b.\text{KNr}=t.\text{KNr}\}$
- 5) Kunden, von denen keine Bestellung registriert ist.
→ $\{t \mid t \in \text{Kunden} \wedge \neg(\exists b: b \in \text{Bestellungen} \wedge b.\text{KNr}=t.\text{KNr})\}$
oder
 $\{t \mid t \in \text{Kunden} \wedge \forall b: b \in \text{Bestellungen} \Rightarrow b.\text{KNr} \neq t.\text{KNr}\}$
- 6) Kunden, die alle überhaupt lieferbaren Produkte bestellt haben.
→ $\{t \mid t \in \text{Kunden} \wedge \forall p: p \in \text{Produkte} \Rightarrow$
 $(\exists b: b \in \text{Bestellungen} \wedge b.\text{PNr}=p.\text{PNr} \wedge$
 $b.\text{KNr}=t.\text{KNr}) \}$

Domain-unabhängige Anfragen

Problem: Anfragen liefern u.U. unendliche Resultatmengen.

Beispiel: $\{t \mid \neg(t \in R)\}$ für eine endliche Relation R.

Lösungsidee:

Eine Anfrage des TRK heißt **domain-unabhängig** g.d.w. für jede mögliche Ausprägung der Datenbank die Resultatmenge von der Datenbank, nicht aber von den zugrundeliegenden Domains abhängt.

Wahl des Universums U bei der Interpretation einer TRK-Formel F:

- **unbeschränkte Interpretation:**

U besteht aus der Vereinigung aller Domains der Datenbank

- **beschränkte Interpretation:**

U besteht aus dem **aktiven Domains** der Datenbank und der Formel F

Definition:

F heißt **domain-unabhängig** g.d.w. ihre unbeschränkte Interpretation für jeden Domain Dom, der den aktiven Domain ADom umfasst, mit ihrer beschränkten Interpretation übereinstimmt.

Domain-unabhängige Anfragen: Beispiele

Sei R eine Relation mit $\text{sch}(R)=\{A\}$, $\text{dom}(A)=\{1,2\}$, $\text{val}(R)=\{<1>\}$.

Der Ausdruck $\{t \mid \neg (t \in R)\}$ hat

als unbeschränkte Interpretation den Wert $\{<2>\}$ und
als beschränkte Interpretation den Wert $\emptyset$.

Der Ausdruck $\{t \mid \exists x : \neg (x=t) \wedge x.A=1\}$ hat

als unbeschränkte Interpretation den Wert $\{<2>\}$ und
als beschränkte Interpretation den Wert $\emptyset$.

Der Ausdruck $\{t \mid t \in R \wedge \forall x : x=t\}$ hat

als unbeschränkte Interpretation den Wert $\emptyset$ und
als beschränkte Interpretation den Wert $\{<1>\}$.

Sichere Anfragen: Idee

$F(x)$ mit Var. x heißt **beschränkt**, wenn für jede Interpretation von F gilt:

- wenn $F(x)$ wahr ist,
dann liegt die Interpretation von x in ADom von F .

$F(x)$ mit Var. x heißt **unbeschränkt**, wenn für jede Interpret. von F gilt:

- wenn die Interpretation von x nicht in ADom von F liegt,
muß $F(x)$ wahr sein.

Durch syntaktische Einschränkungen von Formeln können

- **freie Variablen beschränkt** werden
(so daß in $\{t \mid F(t)\}$ $F(t)$ nur für t aus ADom erfüllbar ist),
- **durch Existenzquantoren gebundene Variablen beschränkt** werden
(so daß in $\exists x : G(x)$ $G(x)$ nur für x aus ADom erfüllbar ist) und
- **durch Allquantoren gebundene Variablen unbeschränkt** werden
(so daß in $\forall x : G(x)$ $G(x)$ für Interpret. von x , die nicht im ADom liegen, immer wahr ist).

Sichere Anfragen: Syntaktische Einschränkung

Definition:

Sei $F(t)$ eine Formel des TRK mit freier Variable t . Zu jeder Var. in F läßt sich ein Schema aus F und dem Datenbankschema ableiten.

Die **Beschränktheit** und die **Unbeschränktheit** der Attr. von Var. in F sind:

- 1) In $t \in R$ sind alle Attribute von t beschränkt.
- 2) In $t.A=c$ mit einer Konstanten c ist $t.A$ beschränkt.
- 3) In $r.A=s.B \wedge F$ ist $r.A$ beschränkt, wenn $s.B$ in F beschränkt ist, und $s.B$ beschränkt, wenn $r.A$ in F beschränkt ist..
- 4a) In $F \wedge G$ ist $t.A$ beschränkt g.d.w. $t.A$ in F oder in G beschränkt ist.
- 4b) In $F \wedge G$ ist $t.A$ unbeschränkt g.d.w. $t.A$ in F und in G unbeschr. ist.
- 5a) In $F \vee G$ ist $t.A$ beschränkt g.d.w. $t.A$ in F und in G beschränkt ist.
- 5b) In $F \vee G$ ist $t.A$ unbeschränkt g.d.w. $t.A$ in F oder in G unbeschr. ist.
- 6a) In $\neg F$ ist $t.A$ beschränkt g.d.w. $t.A$ in F unbeschränkt ist.
- 6b) In $\neg F$ ist $t.A$ unbeschränkt g.d.w. $t.A$ in F beschränkt ist.
- 7a) In $\exists x: F(x)$ ist $t.A$ beschränkt g.d.w. $t.A$ in F beschränkt ist.
- 7b) In $\exists x: F(x)$ ist $t.A$ unbeschränkt g.d.w. $t.A$ in F unbeschränkt ist.
- 8a) In $\forall x: F(x)$ ist $t.A$ beschränkt g.d.w. $t.A$ in F beschränkt ist.
- 8b) In $\forall x: F(x)$ ist $t.A$ unbeschränkt g.d.w. $t.A$ in F unbeschränkt ist.

Sichere und unsichere Anfragen: Beispiele

Sei R eine Relation mit $\text{sch}(R)=\{A,B\}$, $\text{val}(R)=\{<2,2>\}$.

$$\{t \mid \neg(t \in R) \vee t.A=2 \vee t.B=2\}$$

$$\{t \mid \neg(t \wedge R) \wedge t.A=2\}$$

$$\{t \mid \neg(t \wedge R) \wedge t.A=2 \wedge t.B=2\}$$

$$\{t \mid \exists s: (s \in R \wedge s.A=t.A) \}$$

$$\{t \mid \exists s: (s \in R \vee s=t) \}$$

$$\{t \mid \exists s: (s \in R \wedge s=t) \}$$

$$\{t \mid \neg(\exists s: (s \in R \wedge s=t)) \}$$

$$\{t \mid \forall s: (s \in R \wedge s=t) \}$$

$$\{t \mid \forall s: (s \in R \Rightarrow s=t) \}$$

$$\{t \mid \forall s: (s \in R \Rightarrow (s=t \wedge t.A=2 \wedge t.B=2)) \}$$

$$\{t \mid t \in R \wedge \forall s: (s \in R \Rightarrow (s=t \wedge t.A=2 \wedge t.B=2)) \}$$

$$\{t \mid \forall s: (s \in R \wedge (s=t \wedge t.A=2 \wedge t.B=2)) \}$$

$$\{t \mid \forall s: (s \in R \Rightarrow \neg(s=t)) \}$$

$$\{t \mid t \in K \wedge \forall p: p \in P \Rightarrow (\exists b: b \in B \wedge b.PNr=p.PNr \wedge b.KNr=t.KNr)\}$$

Rezept für sichere Anfragen

Satz:

Eine Anfrage des TRK ist sicher

und damit garantiert domain-unabhängig, wenn

- die Anfrage die Form $\{t \mid t \in R \wedge F(t)\}$ hat und.
- jede existenzquantifizierte Teilformel
die Form $\exists x: x \in R \wedge F(t)$ hat und
- jede allquantifizierte Teilformel
die Form $\forall x: x \in R \Rightarrow F(x)$ hat.

6.4 Mächtigkeit relationaler Anfragesprachen

Satz:

- Alle Anfragen, die sich mit der **RA** ausdrücken lassen, lassen sich auch mit dem **sicheren TRK** oder dem **sicheren DRK** ausdrücken.
- Alle Anfragen, die sich mit dem **sicheren TRK** ausdrücken lassen, lassen sich auch mit der **RA** oder dem **sicheren DRK** ausdrücken.
- Alle Anfragen, die sich mit dem **sicheren DRK** ausdrücken lassen, lassen sich auch mit der **RA** oder dem **sicheren TRK** ausdrücken.

6.5 Datalog: Deduktionsregeln als Anfragesprache

Extensionale Repräsentation von Information: Fakten, Tabellen

Intensionale Repräsentation von Information: Regeln, Ableitungen

Beispiel:

intensionale Repräsentation von Flugverbindungen (V)

auf der Grundlage einer extensionalen Repräsentation von Flügen (F)

als Formeln des DRK:

$$\forall a, z: F(a, z) \Rightarrow V(a, z)$$

$$\forall a, z, o: V(a, o) \wedge F(o, z) \Rightarrow V(a, z)$$

als Fixpunktgleichung in der RA:

$$V = F \cup (V \mid x \mid [V.Zielort = F.Abflugort] F)$$

Datalog: Definitionen (1)

Eine **Hornklausel** über einer endlichen Menge von Prädikatsymbolen ist eine logische Formel der Form

$$\forall X_1, \dots, X_n: P_1(Z_{11}, Z_{12}, \dots, Z_{1k_1}) \wedge \dots \wedge P_m(Z_{m1}, Z_{m2}, \dots, Z_{mk_m}) \Rightarrow P_0(Z_{01}, Z_{02}, \dots, Z_{0k_0})$$

mit k_i -stelligen Prädikaten P_i , Variablen $X_1, \dots, X_n$ und Konstanten oder Variablen Z_{ij} .

Die Formel

$$P_0(Z_{01}, Z_{02}, \dots, Z_{0k_0})$$

wird als "**Kopf**" (engl.: head) der Hornklausel bezeichnet, die Formel

$$P_1(Z_{11}, Z_{12}, \dots, Z_{1k_1}) \wedge \dots \wedge P_m(Z_{m1}, Z_{m2}, \dots, Z_{mk_m})$$

als "**Rumpf**" (engl.: body).

Datalog: Definitionen (2)

Ein **Datalog-Programm** besteht aus

- einer Menge von Fakten der Form $P_i(V_1, V_2, \dots, V_k)$ mit einem k-stelligen Prädikat P_i und Konstanten $V_1, \dots, V_k$ und
- einer Menge von Regeln in Form von Hornklauseln.

Ein **Datalog-Programm mit Negation** besteht aus

- einer Menge von Fakten und einer Menge von
- Regeln in Form von Klauseln, bei denen Rumpfprädikate negiert sein können.

Eine **Anfrage** (Ziel; engl: goal) ist ein Ausdruck der Form $P(Z_1, \dots, Z_k)$ mit einem k-stelligen Prädikat P und Konstanten oder Variablen $Z_1, \dots, Z_k$, so daß mindestens eines der Z_j eine (freie) Variable ist.

Eine Regel r heißt **rekursiv**,

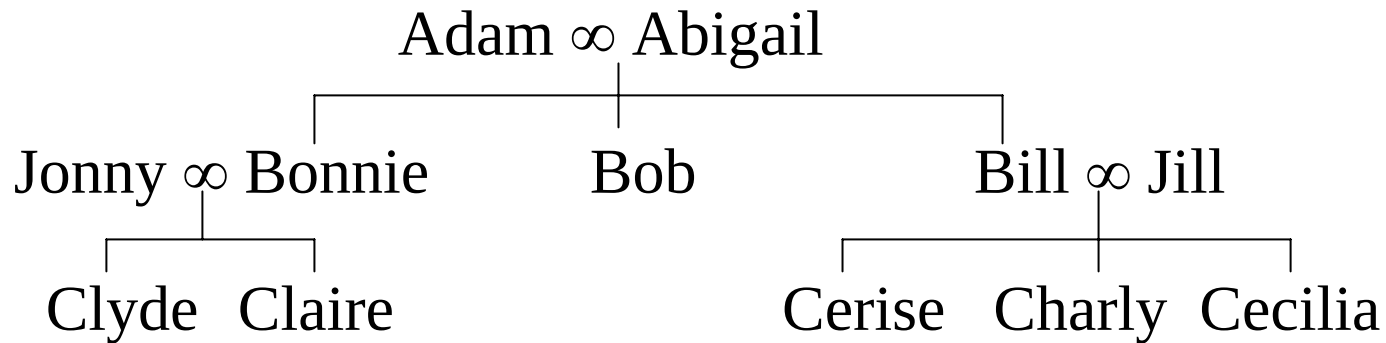
wenn es Regeln $r_1, r_2, \dots, r_n$ gibt mit $r_1 = r_n = r$, so daß für alle i ein Rumpfprädikat von r_i im Kopf von r_{i+1} steht.

Mächtigkeit von Datalog

Satz:

- 1) Die Menge der Anfragen, die sich mit Datalog ***ohne Rekursion und ohne Negation*** ausdrücken lassen, ist eine echte Untermenge der Anfragen, die sich mit dem DRK ausdrücken lassen.
- 2) Die Menge der Anfragen, die sich mit Datalog ***mit Negation, aber ohne Rekursion*** ausdrücken lassen, ist identisch mit der Menge der Anfragen, die sich mit dem DRK ausdrücken lassen.
- 3) Die Menge der Anfragen, die sich mit Datalog ***mit Rekursion und mit Negation*** ausdrücken lassen, ist eine echte Obermenge der Menge von Anfragen, die sich mit dem DRK ausdrücken lassen.

Datalog-Beispiel: Fakten



Mann (Adam), Mann (Jonny), Mann (Bob), Mann (Bill), Mann (Clyde) ,
Mann (Charly), Frau (Abigail), Frau (Bonnie), Frau (Jill), Frau (Claire),
Frau (Cerise), Frau (Cecilia)

Ehepaar (Adam, Abigail), Ehepaar (Jonny, Bonnie), Ehepaar (Bill, Jill)

Elternteil (Adam, Bonnie), Elternteil (Adam, Bob), Elternteil (Adam, Bill),
Elternteil (Abigail, Bonnie), Elternteil (Abigail, Bob), Elternteil (Abigail, Bill),
Elternteil (Jonny, Clyde), Elternteil (Jonny, Claire),

Elternteil (Bonnie, Clyde), Elternteil (Bonnie, Claire),

Elternteil (Bill, Cerise), Elternteil (Bill, Charly), Elternteil (Bill, Cecilia),

Elternteil (Jill, Cerise), Elternteil (Jill, Charly), Elternteil (Jill, Cecilia)

SelbeGeneration (Adam, Abigail),

SelbeGeneration (Adam, Adam), SelbeGeneration (Abigail, Abigail)

SpieltMit (Clyde, Charly)

Datalog-Beispiel: Regeln

Elternteil (X, Y)	$\Rightarrow$ Vorfahren (X, Y)
Elternteil (X, Y) $\wedge$ Vorfahren (Y, Z)	$\Rightarrow$ Vorfahren (X, Z)
Elternteil (X, Y) $\wedge$ Mann (X)	$\Rightarrow$ Vater (X, Y)
Elternteil (X, Y) $\wedge$ Frau (X)	$\Rightarrow$ Mutter (X, Y)
Elternteil (X, Y) $\wedge$ Elternteil (X, Z) $\wedge$ Y $\neq$ Z	$\Rightarrow$ Geschwister (Y, Z)
Elternteil (X, Y) $\wedge$ Elternteil (U, W) $\wedge$ Geschwister (X, U) $\wedge$ Frau (W)	$\Rightarrow$ Cousine (Y, W)
Elternteil (X, Y) $\wedge$ Elternteil (U, W) $\wedge$ SelbeGeneration (X, U)	$\Rightarrow$ SelbeGeneration (Y, W)
Geschwister (X, Y)	$\Rightarrow$ SpieltMit (X, Y)
SpieltMit (Clyde, Y)	$\Rightarrow$ SpieltMit (Claire, Y)
TRUE	$\Rightarrow$ SpieltMit (Cecilia, Y)
Elternteil (X, Y)	$\Rightarrow$ Verwandt (X, Y)
Geschwister (X, Y)	$\Rightarrow$ Verwandt (X, Y)
Verwandt (X, Y)	$\Rightarrow$ Verwandt (Y, X)
Verwandt (X, Y) $\wedge$ Verwandt (Y, Z)	$\Rightarrow$ Verwandt (X, Z)
Mann (X) $\wedge$ $\neg$ Ehepaar (X, Y)	$\Rightarrow$ Ledig (X)
Frau (Y) $\wedge$ $\neg$ Ehepaar (X, Y)	$\Rightarrow$ Ledig (Y)

Kapitel 7: Die Datenbanksprache SQL

7.1 Datendefinition

7.2 Einfache Anfragen

7.3 Semantik einfacher SQL-Anfragen

7.4 Erweiterte Anfragen

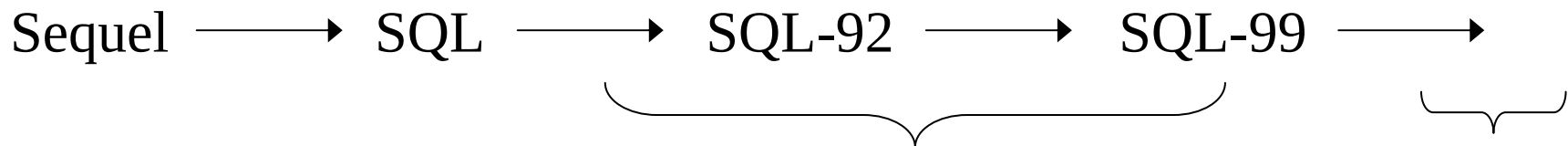
7.5 Datenmodifikation

SQL-Sprachumfang

SQL (Structured Query Language) umfasst:

- Interaktives („stand-alone“) SQL
- Eingebettetes („embedded“) SQL
- Anweisungen zur Integritätssicherung, Zugriffskontrolle
- Anweisung zum Tuning

Historie:



Produkte:
Oracle, DB2, Informix,
Sybase, SQL Server,
(MySQL), ...

5.1 Datendefinition

“Grobsyntax”:

```
CREATE TABLE [user .] table ( column_element {, column_element ...}  
                                {, table_constraint ...} )
```

mit:

```
column_element ::=  
    column data_type [DEFAULT expr] [column_constraint]
```

```
column_constraint ::=  
    [NOT NULL] [PRIMARY KEY | UNIQUE]  
    [REFERENCES [user .] table [ ( column ) ] ]  
    [CHECK ( condition ) ]
```

```
table_constraint ::=  
    [ {PRIMARY KEY | UNIQUE} ( column {, column ...} ) ]  
    [ FOREIGN KEY ( column {, column ...} )  
      REFERENCES [user .] table [ ( column {, column ...} ) ]  
    [CHECK ( condition ) ]
```

Semantik: Anlegen von (leeren) Relationen und
Festlegen von Integritätsbedingungen

Exkurs: Syntaxbeschreibung – kontextfreie Grammatiken

Kontextfreie Grammatik $G = (\Sigma, V, P, S)$ mit

- Σ : Terminalsymbole (Literele)
- V : Nonterminalsymbole (Variablen) (mit $V \cap \Sigma = \emptyset$)
- $P \subseteq V \times (V \cup \Sigma)^*$: Produktionsregeln
- $S \in V$: Startsymbol.

Die von G erzeugte Sprache $L(G) \subseteq \Sigma^* =$

$\{w \in \Sigma^* \mid \text{es gibt endliche Ableitung}$

$S = x_0 \rightarrow x_1 \rightarrow \dots \rightarrow x_n = w \text{ mit } x_i \in (V \cup \Sigma)^*$

und $x_{i-1} \rightarrow x_i$ als Anwendung einer Regel aus P

Exkurs: Syntaxbeschreibung – einfaches Beispiel

$V = \{\text{Address, Name, Firstname, Lastname, Street, City, Country, String, Letter, Number, Digit}\},$

$\Sigma = \{a, b, c, \dots, 0, 1, \dots\},$

$S = \text{Address},$

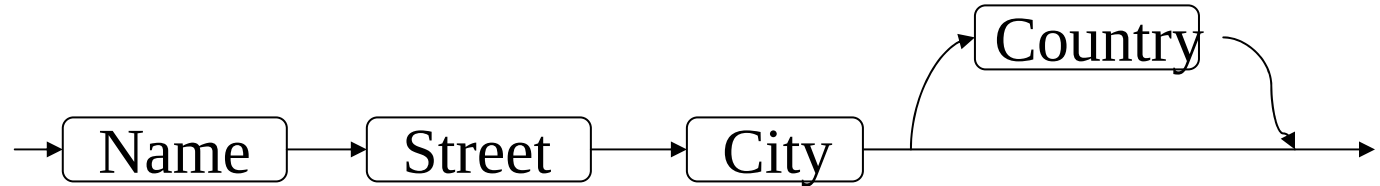
$P = \{\text{Address} \rightarrow \text{Name Street City},$
 $\text{Address} \rightarrow \text{Name Street City Country},$
 $\text{Name} \rightarrow \text{Firstname Lastname},$
 $\text{Name} \rightarrow \text{Letter Lastname},$
 $\text{Firstname} \rightarrow \text{String},$
 $\text{Lastname} \rightarrow \text{String},$
 $\text{Street} \rightarrow \text{String Number},$
 $\text{Country} \rightarrow \text{String},$
 $\text{String} \rightarrow \text{Letter},$
 $\text{String} \rightarrow \text{Letter String},$
 $\text{Number} \rightarrow \text{Digit Number},$
 $\text{Number} \rightarrow \varepsilon,$
 $\text{Letter} \rightarrow a, \quad \text{Letter} \rightarrow b, \quad \dots,$
 $\text{Digit} \rightarrow 0, \quad \text{Digit} \rightarrow 1, \quad \dots\}$

Exkurs: Syntaxbeschreibung – EBNF

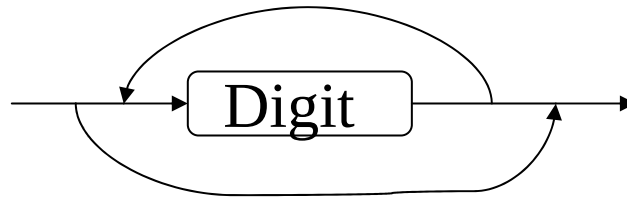
Address ::= Name Street City [Country],
String ::= Letter {Letter ...},
Number ::= {Digit ...},
Digit ::= 0|1|2|3|4|5|6|7|8|9,
usw.

Exkurs: Syntaxbeschreibung – Syntaxdiagramme

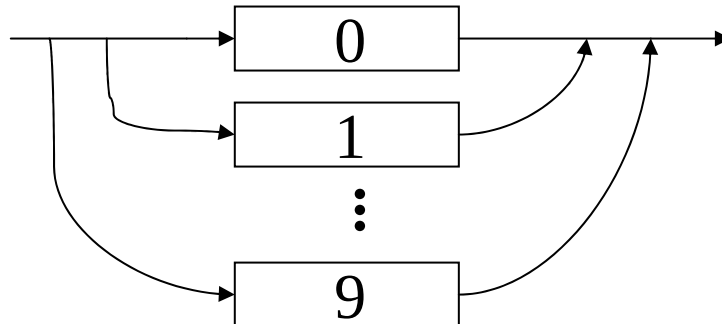
Address ::=



Number ::=



Digit ::=



CREATE TABLE: Beispiele (1)

```
CREATE TABLE Kunden
```

```
(KNr INTEGER CHECK (KNr>0) PRIMARY KEY,  
Name VARCHAR(30),  
Stadt VARCHAR(30),  
Saldo FLOAT, /* oder besser: Saldo MONEY, */  
Rabatt FLOAT CHECK (Rabatt >= 0.0) )
```

```
CREATE TABLE Produkte
```

```
(PNr INTEGER CHECK (PNr>0) PRIMARY KEY,  
Bez VARCHAR(30) NOT NULL UNIQUE,  
Gewicht FLOAT CHECK (Gewicht > 0.0),  
Preis FLOAT CHECK (Preis > 0.0),  
Lagerort VARCHAR(30),  
Vorrat INTEGER CHECK (Vorrat >= 0) )
```


CREATE TABLE: Beispiele (2)

```
CREATE TABLE Bestellungen (  
    BestNr INTEGER CHECK (BestNr > 0) PRIMARY KEY,  
    Monat INTEGER NOT NULL  
        CHECK (Monat BETWEEN 1 AND 12),  
    Tag INTEGER NOT NULL CHECK (Tag BETWEEN 1 AND 31),  
    /* oder besser: Datum DATE, */  
    KNr INTEGER CHECK(KNr > 0),  
    PNr INTEGER CHECK (PNr > 0),  
    Menge INTEGER CHECK (Menge > 0),  
    Summe FLOAT,  
    Status VARCHAR(9) CHECK (Status IN ('neu', 'geliefert', 'bezahlt')),  
    FOREIGN KEY (PNr) REFERENCES Produkte (PNr),  
    FOREIGN KEY (KNr) REFERENCES Kunden (KNr),  
    UNIQUE (Monat, Tag, PNr, KNr) )
```

5.2 Einfache Anfragen

“Grobsyntax”:

```
select_block { { UNION | INTERSECT | EXCEPT } [ALL]  
              select_block ...}  
[ORDER BY result_column [ASC | DESC]  
          {, result_column [ASC | DESC] ...}]
```

mit select_block ::=

```
SELECT [ALL | DISTINCT]  
      {column | {expression [AS result_column]}}  
      {, {column | {expression [AS result_column]}} ...}  
FROM table [correlation_var] {, table [correlation_var] ...}  
[WHERE search_condition]  
[GROUP BY column {, column ...} [HAVING search_condition] ]
```

Abbildung auf TRK und RA

“Grobsemantik”:

SELECT A, B, ... FROM R, S, ..., T, ... WHERE F

(so daß A, B, ... zu R, S, ... gehören, nicht aber zu T, ...,
und F über R, S, ..., T, ... definiert ist)

→ RA: $\pi[A, B, \dots] (\sigma [F] (R \times S \times \dots \times T \times \dots))$

→ TRK: $\{x.A, y.B, \dots \mid x \in R \wedge y \in S \wedge \dots \wedge \exists z: z \in T \dots$
 $\wedge F(x, y, \dots, z, \dots)\}$

Einfache SQL-Anfragen: Beispiele (1)

1) Finden Sie (die Namen) alle(r) Kunden mit negativem Saldo.

→ `SELECT KNr, Name, Stadt, Saldo, Rabatt FROM Kunden
WHERE Saldo < 0.0`

oder: `SELECT * FROM Kunden WHERE Saldo < 0.0`

bzw.: `SELECT Name FROM Kunden WHERE Saldo < 0.0`

2) Namen von Kunden mit unbezahlter Bestellung vor Anfang Oktober

→ `SELECT Name FROM Kunden, Bestellungen
WHERE Monat < 10 AND Status <> 'bezahlt'
AND Kunden.KNr = Bestellung.KNr`

3) Namen der Homburger Kunden, die seit Anfang Sept. ein Produkt aus Homburg bekommen haben, jeweils mit der Bezeichnung des Produkts

→ `SELECT Name, Bez FROM Kunden, Bestellungen, Produkte
WHERE Stadt='Homburg'
AND Monat>=9 AND Status<>'neu'
AND Lagerort='Homburg'
AND Kunden.KNr=Bestellungen.KNr
AND Bestellungen.PNr=Produkt.PNr`

Einfache SQL-Anfragen: Beispiele (2)

- 4) Finden Sie die Rechnungssumme der Bestellung mit BestNr 111 (ohne auf das Attribut Summe der Relation Bestellungen zuzugreifen).
- `SELECT Menge*Preis*(1.0-Rabatt) AS Rechnungssumme
FROM Bestellungen, Produkte, Kunden
WHERE BestNr=111
AND Bestellungen.PNr=Produkte.PNr
AND Bestellungen.KNr=Kunden.KNr`

Allgemeinere Form der FROM-Klausel

Korrelationsvariablen (Tupelvariablen):

Finde alle Paare von Kunden, die in derselben Stadt wohnen.

→ `SELECT K1.Name, K2.Name FROM Kunden K1, Kunden K2
WHERE K1.Stadt=K2.Stadt AND K1.KNr < K2.KNr`

Outer Joins:

Kunden mit 5 % Rabatt zusammen mit ihren Bestellungen und den entsprechenden Produkten, inkl. Kunden ohne Bestellungen

→ `SELECT *
FROM Kunden FULL OUTER JOIN Bestellungen
ON (Kunden.KNr=Bestellungen.KNr),
Produkte
WHERE Rabatt = 0.05
AND Produkte.PNr=Bestellungen.PNr`

Allgemeinere Form der WHERE-Klausel (1)

Bereichsanfragen:

Kunden, deren Rabatt zwischen 10 und 20 Prozent liegt.

→ `SELECT * FROM Kunden
WHERE Rabatt BETWEEN 0.10 AND 0.20`

String-Pattern-Matching:

Kunden, deren Name mit A beginnt.

→ `SELECT * FROM Kunden WHERE Name LIKE 'A%'`

Kunden mit Namen Meier, Maier, Meyer, ...

→ `SELECT * FROM Kunden WHERE Name LIKE 'M__er'`

Test auf Nullwert:

Produkte, die grundsätzlich in keinem Lager geführt werden.

→ `SELECT * FROM Produkte WHERE Lagerort IS NULL`

Allgemeinere Form der WHERE-Klausel (2): Subqueries

Test auf Mitgliedschaft in Menge:

Finden Sie alle Kunden aus Homburg, Merzig und Saarlouis.

```
→ SELECT * FROM Kunden  
   WHERE Stadt IN ('Homburg', 'Merzig', 'Saarlouis')
```

mit Subqueries:

Namen von Kunden mit unbezahlter Bestellung vor Anfang Oktober

```
→ SELECT Name FROM Kunden  
   WHERE KNr IN ( SELECT KNr FROM Bestellungen  
                  WHERE Status <> 'bezahlt' AND Monat < 10 )
```

mit korrelierten Subqueries:

Kunden aus Städten, in denen es mindestens zwei Kunden gibt.

```
→ SELECT * FROM Kunden K1  
   WHERE Stadt IN (SELECT Stadt FROM Kunden K2  
                  WHERE K2.KNr <> K1.KNr )
```


Allgemeinere Form der WHERE-Klausel (3): “quantifizierte” Subqueries

- Die Bedingung *Wert θ ANY Menge* mit $\theta \in \{=, \neq, <, >, \leq, \geq\}$ ist erfüllt, wenn es in der Menge ein Element gibt, für das *Wert θ Element* gilt.
(=ANY ist äquivalent zu IN.)
- Die Bedingung *Wert θ ALL Menge* mit $\theta \in \{=, \neq, <, >, \leq, \geq\}$ ist erfüllt, wenn für alle Elemente der Menge gilt: *Wert θ Element*.
($<>$ ALL ist äquivalent zu NOT IN.)
- Die Bedingung *EXISTS Menge* ist erfüllt, wenn die Menge nicht leer ist
(dies ist äquivalent zur Bedingung `0 < SELECT COUNT(*) FROM ...`)

Beispiele:

Finden Sie die Kunden mit dem geringsten Rabatt.

→ `SELECT * FROM Kunden`
`WHERE Rabatt <= ALL ( SELECT Rabatt FROM Kunden )`

Finden Sie die Kunden, für die keine Bestellung registriert ist.

→ `SELECT * FROM Kunden`
`WHERE NOT EXISTS (SELECT * FROM Bestellungen`
`WHERE Bestellungen.KNr = Kunden.KNr )`

Allgemeinere Form der WHERE-Klausel (4): Simulation allquantifizierter Suchprädikate

Finden Sie die Kunden, die alle überhaupt lieferbaren Produkte irgendwann bestellt haben:

```
→ SELECT * FROM Kunden
   WHERE NOT EXISTS
       ( SELECT * FROM Produkte
         WHERE NOT EXISTS
             ( SELECT * FROM Bestellungen
               WHERE Bestellungen.PNr = Produkte.PNr
                 AND Bestellungen.KNr = Kunden.KNr ) )
```

5.3 Präzise Semantik einfacher SQL-Anfragen: Abbildung auf TRK

Voraussetzungen:

- Vernachlässigung von Multimengen, Nullwerten u.ä.
- Eindeutige Benennung von Tupelvariablen und Zuordnung von Attr.

Beispiel: SELECT A FROM REL

WHERE EXISTS (SELECT B FROM REL WHERE B>0)
umbenennen in

SELECT R1.A FROM REL R1 WHERE EXISTS
(SELECT R2.B FROM REL R2 WHERE R2.B>0)

Definition einer Abbildungsfunktion

sql2trc: sql query $\rightarrow$ trc query

von select_block-Konstrukten auf TRK-Anfragen
unter Verwendung der Funktion

sql2trc': sql where clause $\rightarrow$ trc formula

von search_condition-Konstrukten auf TRK-Formeln.

Abbildung auf TRK (1)

$\text{sql2trc} [\text{SELECT } A_1, A_2, \dots \text{ FROM REL1 } R_1, \text{REL2 } R_2, \dots, \text{RELm } R_m, \\ \text{TAB1 } T_1, \dots, \text{TABk } T_k \text{ WHERE } F]$
 (so daß $A_1, A_2, \dots, A_n$ zu $\text{REL1}, \text{REL2}, \dots, \text{RELm}$ gehören,
 nicht aber zu $\text{TAB1}, \dots, \text{TABk}$
 und F über $\text{REL1}, \dots, \text{RELm}, \text{TAB1}, \dots, \text{TABk}$ definiert ist)
 $= \{r_{i_1}.A_1, r_{i_2}.A_2, \dots \mid r_1 \in \text{REL1} \wedge r_2 \in \text{REL2} \wedge \dots \wedge r_m \in \text{RELm} \\ \wedge \exists t_1 \dots \exists t_k (t_1 \in \text{TAB1} \wedge \dots \wedge t_k \in \text{TABk} \wedge \text{sql2trc}'[F])\}$

$\text{sql2trc} [\text{select-block1 UNION select-block2}]$
 (mit select-block1 : $\text{SELECT } A_1, A_2, \dots \text{ FROM REL1 } R_1, \dots, \text{RELm } R_m, \\ \text{TAB1 } T_1, \dots, \text{TABk } T_k \text{ WHERE } F$
 und select-block2 : $\text{SELECT } B_1, B_2, \dots \text{ FROM SET1 } S_1, \dots, \text{SETm}' S_m', \\ \text{PAR1 } P_1, \dots, \text{PARk}' P_k' \text{ WHERE } G$)
 $= \{u_1, u_2, \dots \mid$
 $(\exists r_1 \dots \exists r_m (u_1 = r_1.A_1 \wedge u_2 = r_2.A_2 \wedge \dots \wedge$
 $r_1 \in \text{REL1} \wedge r_2 \in \text{REL2} \wedge \dots \wedge r_m \in \text{RELm} \wedge$
 $\exists t_1 \dots \exists t_k (t_1 \in \text{TAB1} \wedge \dots \wedge t_k \in \text{TABk} \wedge \text{sql2trc}'[F]))$
 $\vee (\exists s_1 \dots \exists s_m' (u_1 = s_1.B_1 \wedge u_2 = s_2.B_2 \wedge \dots \wedge$
 $s_1 \in \text{SET1} \wedge s_2 \in \text{SET2} \wedge \dots \wedge s_m' \in \text{SETm}' \wedge$
 $\exists p_1 \dots \exists p_k' (p_1 \in \text{PAR1} \wedge \dots \wedge p_k' \in \text{PARk}' \wedge \text{sql2trc}'[G]))\}$

Abbildung auf TRK (2)

$\text{sql2trc}' [Ri.Aj \theta Tk.Bl] = ri.Aj \theta tk.Bl$

$\text{sql2trc}' [Ri.Aj \theta c]$ (mit einer Konstanten c) $= ri.Aj \theta c$

$\text{sql2trc}' [F \text{ AND } G] = \text{sql2trc}'[F] \wedge \text{sql2trc}'[G]$

$\text{sql2trc}' [F \text{ OR } G] = \text{sql2trc}'[F] \vee \text{sql2trc}'[G]$

$\text{sql2trc}' [\text{NOT } F] = \neg \text{sql2trc}'[F]$

$\text{sql2trc}' [Ri.Aj \text{ IN subquery}]$

(so daß subquery die Form $\text{SELECT } Qk.C$

$\text{FROM QUELL1 } Q1, \dots, \text{QUELLm'} Qm' \text{ WHERE } H \text{ hat})$

$= \exists q1 \dots \exists qm' (qk.C = ri.Aj \wedge q1 \in \text{QUELL1} \wedge \dots qm' \in \text{QUELLm'} \\ \wedge \text{sql2trc}'[H])$

Abbildung auf TRK (3)

sql2trc' [Ri.Aj θ ANY subquery]

$$= \exists q_k (q_k \in \text{QUELL}_k \wedge (\exists q_1 \dots \exists q_{(k-1)} \exists q_{(k+1)} \dots \exists q_{m'} \\ (r_i.A_j \theta q_k.C \wedge q_1 \in \text{QUELL}_1 \wedge \dots \wedge q_{(k-1)} \in \text{QUELL}_{(k-1)} \wedge \\ q_{(k+1)} \in \text{QUELL}_{(k+1)} \wedge \dots \wedge q_{m'} \in \text{QUELL}_{m'} \wedge \text{sql2trc}'[H])))$$

sql2trc' [Ri.Aj θ ALL subquery]

$$= \forall q_k ((q_k \in \text{QUELL}_k \wedge (\exists q_1 \dots \exists q_{(k-1)} \exists q_{(k+1)} \dots \exists q_{m'} \\ (q_1 \in \text{QUELL}_1 \wedge \dots \wedge q_{(k-1)} \in \text{QUELL}_{(k-1)} \wedge \\ q_{(k+1)} \in \text{QUELL}_{(k+1)} \wedge \dots \wedge q_{m'} \in \text{QUELL}_{m'} \wedge \text{sql2trc}'[H]))) \\ \Rightarrow (r_i.A_j \theta q_k.C))$$

sql2trc' [EXISTS subquery]

(so daß subquery die Form SELECT C1, C2, ...

FROM QUELL1 Q1, ..., QUELL_{m'} Q_{m'} WHERE H hat)

$$= \exists q_1 \dots \exists q_{m'} (q_1 \in \text{QUELL}_1 \wedge \dots \wedge q_{m'} \in \text{QUELL}_{m'} \\ \wedge \text{sql2trc}'[H])$$

Abbildung auf TRK: Beispiel

```
query = SELECT K.KNr, K.Name FROM Kunden K
        WHERE K.Ort='Saarbrücken'
        AND NOT EXISTS
            (SELECT P.PNr FROM Produkte P, Bestellungen B
             WHERE P.Preis > 100 AND P.PNr=B.PNr AND B.KNr=K.KNr)
```

```
sql2trc [query]
= {k.KNr, k.Name | k∈Kunden ∧ sql2trc'[K.Ort=... AND NOT EXISTS ...]}
= {k.KNr, k.Name | k∈Kunden ∧ k.Ort=... ∧ ¬ sql2trc'[EXISTS ...]}
= {k.KNr, k.Name | k∈Kunden ∧ k.Ort=... ∧
    ¬ (∃ p ∃ b ( p∈Produkte ∧ b∈Bestellungen
    ∧ sql2trc'[P.Preis>... AND ... AND ...] )) }
= {k.KNr, k.Name | k∈Kunden ∧ k.Ort=... ∧
    ¬ (∃ p ∃ b ( p∈Produkte ∧ b∈Bestellungen
    ∧ p.Preis>... ∧ p.PNr=b.PNr ∧ b.KNr=k.KNr )) }
```

Präzise Semantik einfacher SQL-Anfragen: Abbildung auf RA

Voraussetzungen:

- Vernachlässigung von Multimengen, Nullwerten u.ä.
- Eindeutige Benennung von Tupelvariablen und Zuordnung von Attr.

Definition einer Abbildungsfunktion

sql2ra: sql query $\rightarrow$ ra query

von select_block-Konstrukten auf RA-Anfragen
unter Verwendung der Funktion

sql2ra': sql where clause $\times$ ra query $\rightarrow$ ra query

von search_condition-Konstrukten auf RA-Ausdrücke
sowie der Hilfsfunktion

sql2ra-: sql where clause $\times$ ra query $\rightarrow$ ra query

mit sql2ra- $[F](E) = E - \pi[\text{sch}(E)] (\text{sql2ra}'[F](E))$

Erweiterung auf Multirelationen relativ leicht möglich.

Abbildung auf RA (1)

sql2ra [SELECT A1, A2, ... FROM REL1 R1, REL2 R2, ..., RELm Rm,
TAB1 T1, ..., TABk Tk WHERE F]

(so daß A1, A2, ..., An zu REL1, REL2, ..., RELm gehören,
nicht aber zu TAB1, ..., TABk
und F über REL1, ..., RELm, TAB1, ..., TABk definiert ist)

= R1 := REL1; ...; Rm := RELm; T1 := TAB1; ...; Tk := TABk;
 $\pi[R_{i_1}.A1, R_{i_2}.A2, \dots] (\text{sql2ra}'[F](R1 \times \dots \times Rm \times T1 \times \dots \times Tk))$

sql2ra [select-block1 UNION select-block2]

(mit select-block1: SELECT A1, ... FROM REL1 R1, ..., RELm Rm,
TAB1 T1, ..., TABk Tk WHERE F
und select-block2: SELECT B1, ... FROM SET1 S1, ..., SETm' Sm',
PAR1 P1, ..., PARk' Pk' WHERE G)

= sql2ra[select-block1] $\cup$ sql2ra[select-block2]
(mit ggf. notwendigen Umbenennungen von Attributen)

Abbildung auf RA (2)

$\text{sql2ra}' [Ri.Aj \theta Tk.Bl] (E) = \sigma[Ri.Aj \theta Tk.Bl](E)$
 $\text{sql2ra}' [Ri.Aj \theta c] (E) \text{ (mit einer Konstanten } c) = \sigma[Ri.Aj \theta c](E)$
 $\text{sql2ra}' [F \text{ AND } G] (E) = \text{sql2ra}'[F](E) \cap \text{sql2tra}'[G](E)$
 $\text{sql2ra}' [F \text{ OR } G] (E) = \text{sql2ra}'[F](E) \cup \text{sql2ra}'[G](E)$
 $\text{sql2ra}' [\text{NOT } F] (E) = \text{sql2ra}'[F](E) = E - \pi[\text{sch}(E)] (\text{sql2ra}'[F](E))$

$\text{sql2ra}' [Ri.Aj \text{ IN subquery}] (E)$
(so daß subquery die Form $\text{SELECT } Qk.C$
 $\text{FROM QUELL1 } Q1, \dots, \text{QUELLm' } Qm' \text{ WHERE } H \text{ hat}$)
 $= Q1 := \text{QUELL1}; \dots Qm' := \text{QUELLm'};$
 $\pi[\text{sch}(E)] (\text{sql2ra}'[H] (\sigma[Ri.Aj = Qk.C] (E \times Q1 \times \dots \times Qm')))$

Abbildung auf RA (3)

sql2ra' [Ri.Aj θ ANY subquery]
= Q1 := QUELL1; ... Qm' := QUELLm';
 $\pi[\text{sch}(E)] (\text{sql2ra}'[H] (\sigma[\text{Ri.Aj } \theta \text{ Qk.C}] (E \times Q1 \times \dots \times Qm')))$

sql2ra' [Ri.Aj θ ALL subquery] (E)
= sql2ra- [Ri.Aj θ' ANY subquery] (E)
mit θ' gleich $\neq$ für θ gleich $=$, $=$ für $\neq$, $<$ für $\geq$, $>$ für $\leq$, usw.
= $E - \pi[\text{sch}(E)] (\text{sql2ra}'[H] (\sigma[\text{Ri.Aj } \theta' \text{ Qk.C}] (E \times Q1 \times \dots \times Qm')))$

sql2ra' [EXISTS subquery]
(so daß subquery die Form SELECT C1, C2, ...
FROM QUELL1 Q1, ..., QUELLm' Qm' WHERE H hat)
= Q1 := QUELL1; ... Qm' := QUELLm';
 $\pi[\text{sch}(E)] (\text{sql2ra}'[H] (E \times Q1 \times \dots \times Qm'))$

Abbildung auf RA: Beispiel

query = SELECT K.KNr, K.Name FROM Kunden K
WHERE K.Ort='Saarbrücken'
AND NOT EXISTS
(SELECT P.PNr FROM Produkte P, Bestellungen B
WHERE P.Preis > 100 AND P.PNr=B.PNr AND B.KNr=K.KNr)

sql2ra [query]

$$\begin{aligned} &= K := \text{Kunden}; B := \text{Bestellungen}; P := \text{Produkte}; \\ &\quad \pi[K.KNr, K.Name] (\text{sql2ra}'[K.Ort=\dots \text{ AND NOT EXISTS } \dots](K)) \\ &= \dots \pi[K.KNr, K.Name] (\sigma[\text{Ort}=\dots](K) \cap \text{sql2ra}'[\text{NOT EXISTS } \dots](K)) \\ \\ &= \dots \pi[K.KNr, K.Name] (\sigma[\text{Ort}=\dots](K) \\ &\quad \cap (K - \pi[\text{sch}(E)](\text{sql2ra}'[\text{EXISTS } \dots](K)))) \\ &= \dots \pi[K.KNr, K.Name] (\sigma[\text{Ort}=\dots](K) \\ &\quad \cap (K - \pi[\text{sch}(K)](\text{sql2ra}'[P.Preis\dots\text{AND}\dots\text{AND}\dots](K \times P \times B)))) \\ &= \dots \pi[K.KNr, K.Name] (\sigma[\text{Ort}=\dots](K) \\ &\quad \cap (K - \pi[\text{sch}(K)](\sigma[P.Preis>100](K \times P \times B) \cap \\ &\quad \quad \sigma[P.PNr=B.PNr](K \times P \times B) \cap \\ &\quad \quad \sigma[P.PNr=B.PNr](K \times P \times B)))) \end{aligned}$$

5.4 Erweiterte SQL-Anfragen: Aggregationsfunktionen

"Grobsyntax":

{ MAX | MIN | AVG | SUM | COUNT }
({ ALL | DISTINCT } {column | expression | *})

„Grobsemantik“:

Abbildung einer Menge skalarer Werte auf einen skalaren Wert

Aggregationsfunktionen: Beispiele

- 1) Finden Sie den höchsten (durchschnittlichen) Rabatt aller Kunden.
→ `SELECT MAX (Rabatt) FROM Kunden`
→ `SELECT AVG (Rabatt) FROM Kunden`
- 2) An wievielen Lagerorten werden Produkte gelagert?
→ `SELECT COUNT (DISTINCT Lagerort) FROM Produkte`
- 3) Wieviele Kunden haben einen Rabatt von mehr als 15 Prozent?
→ `SELECT COUNT (*) FROM Kunden WHERE Rabatt > 0.15`
- 4) Welche Kunden haben einen überdurchschnittlichen Rabatt?
→ `SELECT * FROM Kunden`
`WHERE Rabatt > SELECT AVG (Rabatt) FROM Kunden`
- 5) Wie hoch ist der Gesamtumsatz?
→ `SELECT SUM (Menge*Preis*(1.0-Rabatt))`
`FROM Bestellungen, Produkte, Kunden`
`WHERE Bestellungen.PNr=Produkte.PNr`
`AND Bestellungen.KNr=Kunden.KNr`

Built-in-Funktionen auf skalaren Werten

Häufig produktspezifisch, z.B.

- Stringmanipulation in Oracle
SELECT SUBSTR (Name, INSTR(Name, ' ')+1) FROM Kunden
- Umwandlung eines Datums (Datentyp DATE) in einen String
SELECT TO_CHAR(SYSDATE, 'DY DD MONTH YYYY, HH24:MI:SS')

usw. usw.

Oracle-spezifische Funktionen zur Textinhaltssuche

mit Oracle8i interMedia

mit Ranking von Suchresultaten:

```
SELECT PNr, Bez FROM Produkte WHERE  
CONTAINS (Beschreibung, 'hard disk ', 1) > 0 ORDER BY Score(1) DESC  
oder: ... CONTAINS (Beschreibung, 'NEAR(hard, disk, 10) ', 1) > 0 ...
```

mit Reduktion auf Wortstämme:

```
SELECT PNr, Bez FROM Produkte WHERE  
CONTAINS (Beschreibung, '$disk', 1) >= 5 ORDER BY Score(1) DESC
```

inkl. Thesaurus zur Berücksichtigung
von Synonymen (SYN) und Unterbegriffen (NT):

```
SELECT PNr, Bez FROM Produkte  
WHERE CONTAINS (Beschreibung, 'SYN(disk)', 1) >= 0  
AND CONTAINS (Beschreibung, 'NT(optics)', 2) >= 0  
ORDER BY Score(1)*Score(2) DESC
```

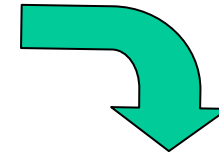

Aggregation mit Gruppierung

Beispiel: Gesamtverkaufszahl von Produkten (ab Anfang September)

→ SELECT PNr, SUM(Menge) FROM Bestellungen
WHERE Monat >= 9 GROUP BY PNr

Zwischenresultat:

PNr	Gruppe		
	BestNr	...	Menge
1	9		100
2	3		4
3	4		1
4	5		10
5	6		50
	10		50
	11		50
6	7		2
7	8		5
	12		10



Endresultat:

PNr	SUM(Menge)
1	100
2	4
3	1
4	10
5	150
6	2
7	15

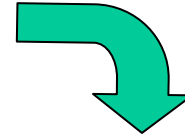
Aggregation mit Gruppierung und Gruppenauswahl

Kunden mit mindestens zwei Bestellungen seit Anfang Oktober, deren Gesamtwert mindestens 2000 DM betragen hat.

→ `SELECT KNr FROM Bestellungen WHERE Monat >= 10
GROUP BY KNr HAVING COUNT(*) >= 2 AND SUM(Summe) >= 2000`

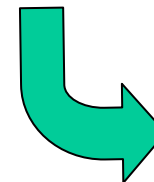
Zwischenresultat 1:

KNr	Gruppe		
	BestNr	...	Summe
1	6		900.00
	7		180.00
	8		900.00
2	9		1600.00
	10		800.00
3	7		1800.00



Zwischenresultat 2:

KNr	COUNT(*)	SUM (Summe)
1	3	1980.00
2	2	2400.00
3	1	1800.00



Endresultat:

KNr
2

Beispiele für GROUP BY ... HAVING ...

1) Gesamtverkaufszahl von Produkten, die ab Anfang September mehr als 1000-mal verkauft worden sind

→ SELECT PNr, SUM(Menge) FROM Bestellungen
WHERE Monat >= 9
GROUP BY PNr HAVING SUM(Menge) > 1000

2) Durchschnittsrabatt für Städte mit mehr als 10 Kunden

→ SELECT Stadt, AVG(Rabatt) AS MittlRabatt FROM Kunden
GROUP BY Stadt HAVING COUNT(*) > 10
ORDER BY 2 DESC bzw. ORDER BY MittlRabatt DESC

3) Kunden, die alle überhaupt lieferbaren Produkte bestellt haben

→ SELECT KNr FROM Bestellungen
GROUP BY KNr
HAVING COUNT (DISTINCT PNr) =
(SELECT COUNT(*) FROM Produkte)

Semantik der Gruppierung

Abbildung auf erweiterte RA (wobei $A' \subseteq A$ gelten muß):

$$\begin{aligned} & \text{sql2ra} [\text{SELECT } A', f(B) \text{ FROM } \dots \text{ WHERE } \dots \text{ GROUP BY } A] \\ &= R(A, F) := \gamma_+[A, B, f] (\text{sql2ra} [\text{SELECT } A, B \text{ FROM } \dots \text{ WHERE } \dots]); \\ & \pi_+[A', F] (R) \end{aligned}$$

Abbildung von

SELECT A', f(B) FROM ... WHERE ...
GROUP BY A HAVING cond(A, g(C))

auf RA-Programm:

$Q(A, B, C) := \text{sql2ra} [\text{SELECT } \dots \text{ FROM } \dots \text{ WHERE } \dots]; R(A, F) := \emptyset;$
for each $x \in \pi[A](Q)$ do
 $G_x := \pi_+[A, B, C] (\sigma_+[A=x] (Q))$
 if $\text{sql2ra}'[\text{cond}](G_x) \neq \emptyset$ then $y := f(\pi_+[B](G_x)); R := R \cup_+ \{(x, y)\}$ fi;
od;
 $\pi_+[A', F] (R)$

Rekursive Anfragen

für transitive Hüllen u.ä.

Beispiel SQL-99-Standard:

```
WITH RECURSIVE Verbindungen (Start, Ziel, Gesamtdistanz) AS
  ( ( SELECT Abflugort, Zielort, Distanz
      FROM Flüge WHERE Abflugort = 'Frankfurt' )
    UNION
    ( SELECT V.Start, F. Ziel, V.Gesamtdistanz + F.Distanz
      FROM Verbindungen V, Flüge F WHERE V.Ziel = F.Abflugort ))
SELECT Ziel, Gesamtdistanz FROM Verbindungen
```

Beispiel Oracle:

```
SELECT F.Zielort FROM Flüge F
START WITH Abflugort = 'Frankfurt'
CONNECT BY Abflugort = PRIOR Zielort
AND PRIOR Ankunftszeit < Abflugzeit - 0.05
```

5.5 Datenmodifikation (1)

Einfügen von Tupeln:

"Grobsyntax":

```
INSERT INTO table [ ( column {, column ...} ) ]  
{ VALUES ( expression {, expression ...} ) | subselect }
```

Beispiele:

- 1) INSERT INTO Kunden VALUES (7, 'Kunz', 'Neunkirchen', 0.0, 0.0)
- 2) INSERT INTO Kunden (KNr, Name, Stadt)
VALUES (7, 'Kunz', 'Neunkirchen')
- 3) CREATE TABLE Mahnungen (BestNr ...)
mit demselben Schema wie Bestellungen
INSERT INTO Mahnungen
(SELECT * FROM Bestellungen
WHERE Status='geliefert' AND Monat<10)

Datenmodifikation (2)

Ändern von Tupeln:

"Grobsyntax":

UPDATE table [correlation_var]

SET column = expression {, column = expression ...}

WHERE search_condition

Beispiele:

1) UPDATE Kunden SET Stadt = 'Saarbrücken' WHERE KNr=1

2) UPDATE Kunden SET Rabatt = Rabatt + 0.05

WHERE Saldo > -10000.0

AND KNr IN (SELECT KNr FROM Bestellungen

GROUP BY KNr HAVING SUM(Summe) > 100000.0)

Datenmodifikation (3)

Löschen von Tupeln:

"Grobsyntax":

```
DELETE FROM table [correlation_var] [WHERE search_condition]
```

Beispiel: DELETE FROM Bestellungen WHERE Monat < 7

Schemaänderung:

"Grobsyntax":

```
ALTER TABLE table
```

```
ADD column datatype [column_constraint]  
    {, column data_type [column_constraint] ...}
```

Beispiel:

```
ALTER TABLE Bestellungen ADD Frist INTEGER CHECK (Frist > 0)
```

Transaktionen:

geklammerte Folgen von SQL-Anweisungen zu einer atomaren Einheit mit Abschluß COMMIT WORK oder ROLLBACK WORK

Varianten der “Alleskäufer”-Query

- 1)

```
SELECT KNr FROM Kunden
WHERE ( SELECT PNr FROM Produkte )
      IN ( SELECT PNr FROM Bestellungen
           WHERE KNr = Kunden.KNr )
```
- 2)

```
SELECT KNr FROM Bestellungen
WHERE PNr = ALL (SELECT PNr FROM Produkte)
```
- 3)

```
SELECT * FROM Kunden K
WHERE NOT EXISTS ( SELECT * FROM Produkte P
                  WHERE NOT EXISTS
                    ( SELECT * FROM Bestellungen B
                      WHERE B.PNr = P.PNr
                        AND B.KNr = K.KNr ) )
```
- 4)

```
SELECT KNr FROM Bestellungen GROUP BY KNr
HAVING COUNT (DISTINCT PNr) =
      (SELECT COUNT(*) FROM Produkte)
```

Kapitel 8: Web-Anwendungen mit SQL und PHP

8.1 Embedded SQL (ESQL)

8.2 Dynamisches ESQL

8.3 JDBC

8.4 Web-Anwendungen mit Java Servlets

Kopplung von Programmier- und DB-Sprachen

- 1) Erweiterung der Programmiersprache um DB-Sprachkonstrukte
→ Persistente Programmiersprachen (z.B. DBPL)
- 2) Erweiterung der DB-Sprache um Programmierkonstrukte
→ 4GLs (z.B. PL/SQL)
- 3) Einbettung der DB-Sprache in die Programmiersprache
→ Embedded SQL (ESQL) für „alle“ Programmiersprachen

Beispiel für 1:

```
TYPE   Produktrecordtyp = RECORD
                                PNR: INTEGER; ...
                                END;
VAR    Produkterrelationentyp = RELATION OF Produktrecordtyp;
VAR    Produkte: PERSISTENT Produkterrelationentyp;
        Ergebnis: Produkterrelationentyp;
        x: Produktrecordtyp;
```

```
Ergebnis :=      SELECT * FROM Produkte WHERE ...;
FOR EACH x IN Ergebnis DO ... END;
```

8.1 Embedded SQL (ESQL)

Kernproblem:

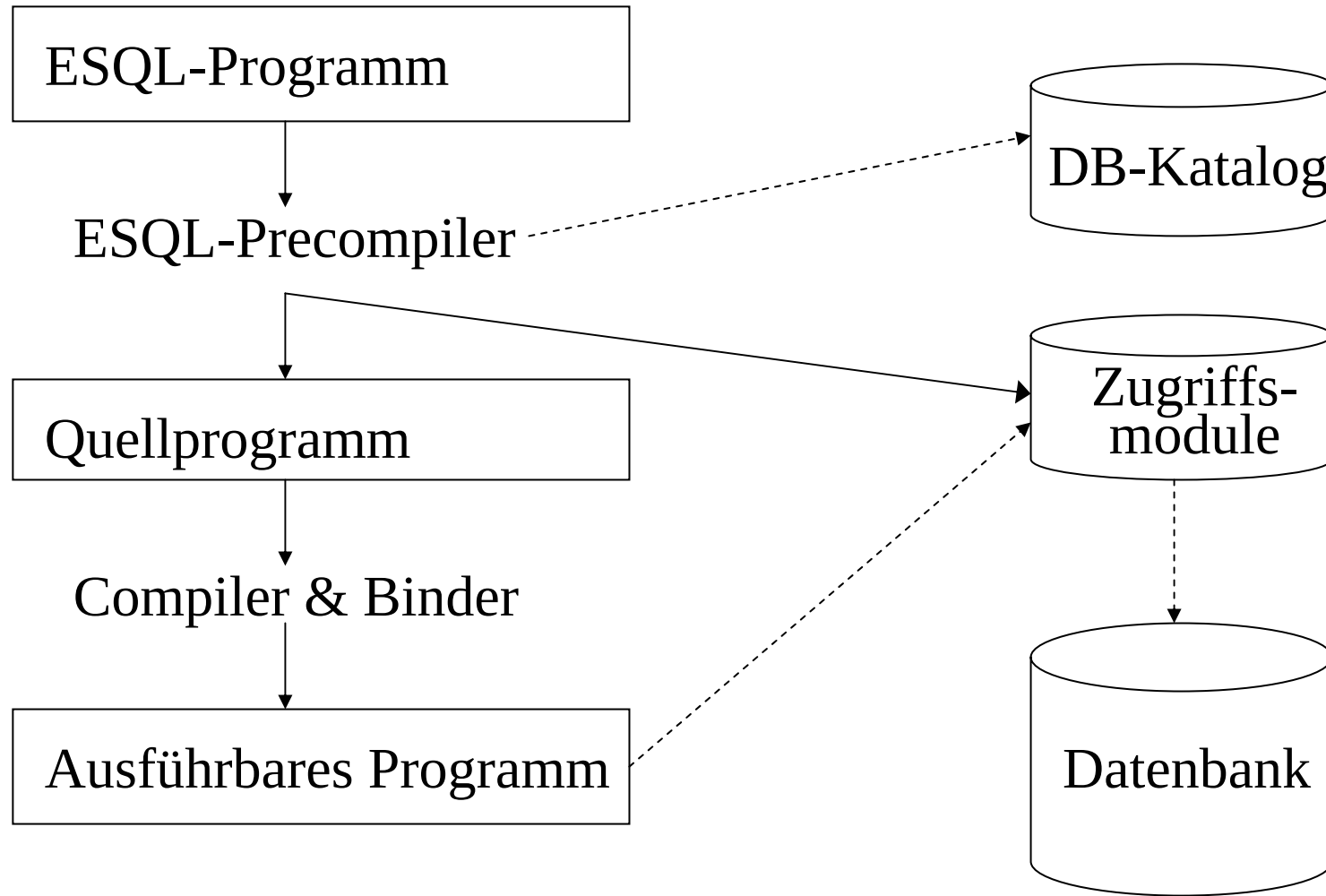
Abbildung von Tupelmengen auf
Datentypen der Wirtsprogrammiersprache

Lösungsansatz:

- 1) Abbildung von Tupeln bzw. Attributen auf die
Datentypen der Wirtsprogrammiersprache
→ ***Wirtsprogrammvariablen (Host Variables)***
- 2) Iteratoren (Schleifen) zur Verarbeitung von Tupelmengen
→ ***Cursor-Konzept***

ein universeller Lösungsansatz
für alle Wirtsprogrammiersprachen !

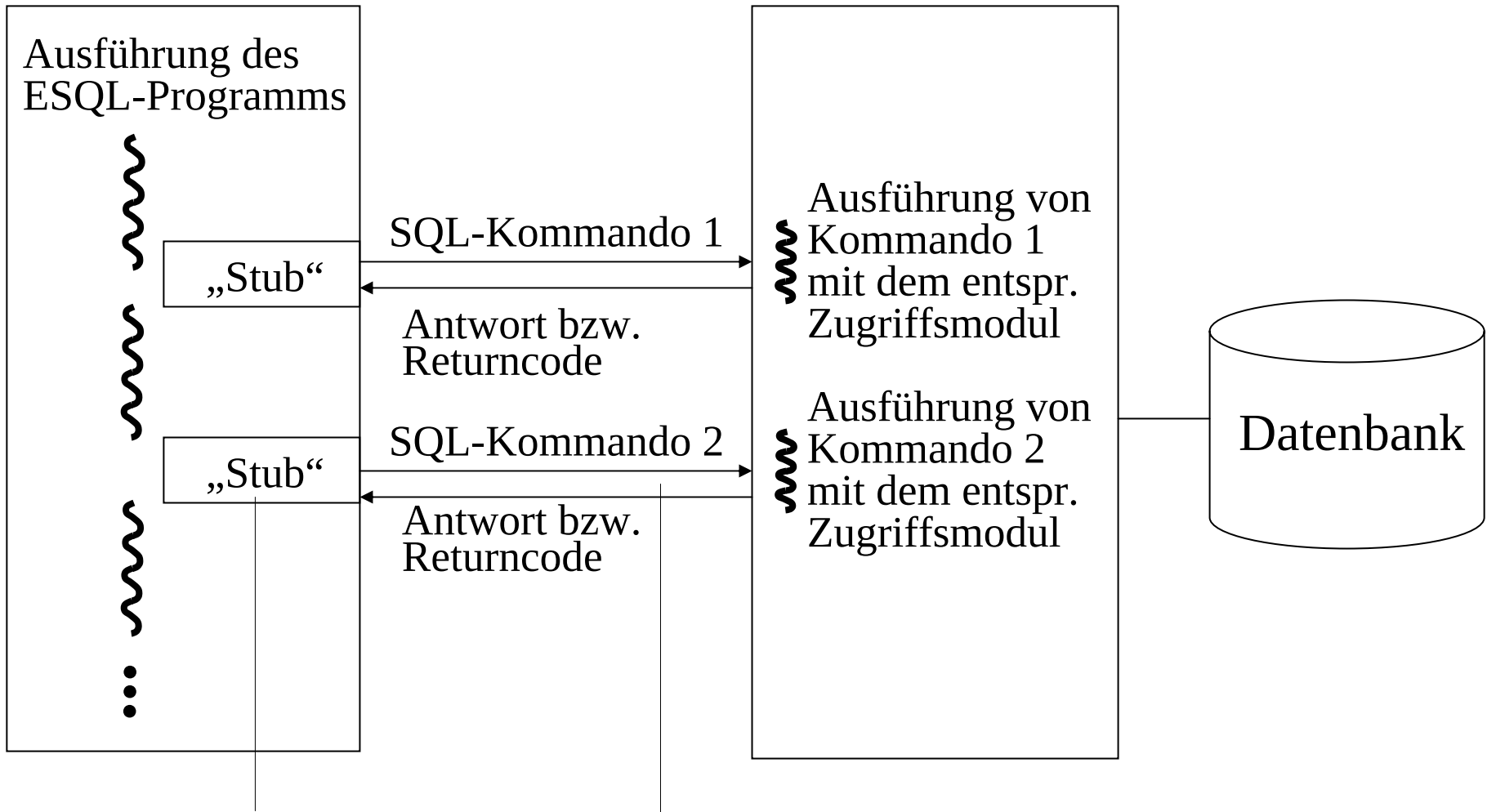
Architektur des ESQL-Precompilers



Laufzeitarbeit für ESQL-Programme in Client-Server-System

Client

DB-Server



vom Precompiler
generierter „Stub“-Code

Kommunikation mittels
TCP/IP (oder CORBA oder ...)

Wirtsprogrammvariable

- Ergänzung von SQL-Anfragen um INTO-Klausel für die Angabe (speziell „markierter“) Wirtsprogrammvariable als Übergabebereich für Anfrageresultate

Beispiel: EXEC SQL SELECT PNr, Menge, Status INTO :pnr, :menge, :status
FROM Bestellungen WHERE BestNr = 555;

- Verwendung von Wirtsprogrammvariablen als Eingabeparameter (an Stelle von Konstanten) in SQL-Anweisungen

Beispiel: EXEC SQL SELECT PNr, Menge, Status INTO :pnr, :menge, :status
FROM Bestellungen WHERE BestNr = :bestnr;

- Verwendung zusätzlicher Indikatorvariablen zur Nullwertdiagnose

Beispiel: EXEC SQL BEGIN DECLARE SECTION;

int pnr; int vorrat; short vorrat_ind;

EXEC SQL END DECLARE SECTION;

...

EXEC SQL SELECT Vorrat INTO :vorrat:vorrat_ind

FROM Produkte WHERE PNr = :pnr;

if (vorrat_ind == 0) { /* kein Nullwert */ } else { /* Nullwert */ };

Beispiel eines ESQL-C-Programms (1)

```
#include <stdio.h>
#include <string.h>
EXEC SQL INCLUDE SQLCA; /* Importieren der SQL Communication Area */
/* Der Precompiler erzeugt an dieser Stelle die folgende Datenstruktur:
    struct sqlca {
        char sqlcaid[8];
        long sqlabc;
        long sqlcode;
        struct { unsigned short sqlerrml; char sqlerrmc[70]; } sqlerrm;
        ...
    }; struct sqlca sqlca; */
main () {
    EXEC SQL BEGIN DECLARE SECTION; /* Wirtsprogrammvariablen */
    int bestnr;
    int pnr;
    int menge;
    VARCHAR status[10];
    /* struct { unsigned short len; unsigned char arr[10]; } status; */
    VARCHAR user[20];
    VARCHAR passwd[10];
    EXEC SQL END DECLARE SECTION;
    EXEC SQL WHENEVER SQLERROR STOP; /* Globale Fehlerbehandlung */
```


Beispiel eines ESQL-C-Programms (2)

```
printf ("Benutzername?"); scanf ("%s", &(user.arr)); user.len = strlen(user.arr);
printf ("Kennwort?"); scanf ("%s", &(passwd.arr)); passwd.len = strlen(passwd.arr);
EXEC SQL CONNECT :user IDENTIFIED BY :passwd; /* Beginn Transaktion */
printf („Bestellnummer? (0 = Programmende)\n"); scanf ("%d", &bestnr);
while (bestnr != 0) {
    EXEC SQL SELECT PNr, Menge, Status INTO :pnr, :menge, :status
        FROM Bestellungen WHERE BestNr = :bestnr;
    if (sqlca.sqlcode == 0) /* Testen des SQL-Returncodes */ {
        status.arr[status.len] = '\0'; /* Konvertierung von VARCHAR in C-String */
        if (strcmp (status.arr, "neu") == 0) {
            EXEC SQL UPDATE Produkte SET Vorrat = Vorrat - :menge
                WHERE PNr = :pnr AND Vorrat >= :menge;
            if (sqlca.sqlcode == 0) {
                strcpy (status.arr, "geliefert"); status.len = strlen (status.arr);
                EXEC SQL UPDATE Bestellungen SET Status = :status
                    WHERE BestNr = :bestnr; }
            else printf ("\n *** Ungenuegender Lagervorrat ***\n"); }
            else printf ("\n *** Lieferung bereits erfolgt ***\n"); }
        else printf ("\n *** Bestellung nicht gefunden ***\n");
        EXEC SQL COMMIT WORK; /* Ende Transaktion und Beginn neue Trans. */
        printf ("Bestellnummer? (0 = Programmende)\n"); scanf ("%d", &bestnr);
    }; /*while*/ } /*main*/
```

Fehlerbehandlung in ESQL

1) Explizites Testen von `sqlca.sqlcode`

(0: korrekt; < 0: Fehler; > 0: Warnung/Ausnahme)

und ggf. Zugriff auf `sqlca.sqlerrd[2]`, `sqlca.sqlerrm.sqlerrmc`, usw.

2) Deklaration von „Exception Handling“-Maßnahmen:

```
EXEC SQL WHENEVER {SQLERROR | NOT FOUND | SQLWARNING}  
                {STOP | GOTO label | CONTINUE}
```

Potentielle Falle: Wirkungsbereich ist statisch!

```
• void f (...) ...  
  { ... EXEC SQL UPDATE ...; ... };  
void g (..) ...  
  { ... EXEC SQL WHENEVER SQLERROR GOTO handle_error; ...  
    f (...); ...  
  };
```

```
• EXEC SQL WHENEVER SQLERROR GOTO handle_error;  
  EXEC SQL CREATE TABLE Mytable ( ... );  
  ...  
  handle_error:  
    EXEC SQL DROP TABLE Mytable; exit (1);
```

Cursor-Konzept für Iteratoren über Tupelmengen

```
#define TRUE 1
#define FALSE 0
printf ("Kundennummer? (0 = Programmende)\n"); scanf ("%d", &knr);
EXEC SQL DECLARE Kundeniterator CURSOR FOR
        SELECT Monat, Tag, Bez, Menge FROM Bestellungen
        WHERE KNr = :knr
        ORDER BY Monat DESC, Tag DESC;

...
EXEC SQL OPEN Kundeniterator;
found = TRUE;
while (found) {
    EXEC SQL FETCH Kundeniterator INTO :monat, :tag, :bez, :menge;
    bez.arr[bez.len] = '\0';
    if ((sqlca.sqlcode >= 0) && (sqlca.sqlcode != 1403))
        printf ("%d.%d.: %d %s", tag, monat, menge, bez);
    else found = FALSE;
}; /*while*/
EXEC SQL CLOSE Kundeniterator;
```

8.2 Dynamisches ESQL

Motivation: Schreibe Programm für Suche

SELECT * FROM * WHERE * LIKE '%SQL%'

Ansatz:

Durchsuche DB-Katalog und
generiere SQL-Queries auf allen Tabellen

Problem:

Anfragen entstehen erst zur Laufzeit des Programms und
können nicht vom ESQL-Precompiler behandelt werden

Lösung:

Dynamisches ESQL

(für alle Anwendungen, bei denen SQL-Statements erst
zur Laufzeit bekannt sind, z.B. für sqlplus u.ä.);

Spezialfall: JDBC (dynamisches ESQL für Java)

Ausschnitt aus DB-Katalog von Oracle

SYSCATALOG bzw. SYS.ALL_TABLES UNION SYS.ALL_VIEWS

OWNER	TABLE_NAME	TABLE_TYPE
DBS	BESTELLUNGEN	TABLE
DBS	KUNDEN	TABLE
DBS	PRODUKTE	TABLE
SYS	ALL_TABLES	VIEW

SYS.ALL_TAB_COLUMNS

OWNER	TABLE_NAME	COLUMN_NAME	DATA_TYPE	DATA_LENGTH	...	COLUMN_ID	...
DBS	KUNDEN	KNR	NUMBER	22		1	
DBS	KUNDEN	NAME	CHAR	30		2	
DBS	KUNDEN	STADT	CHAR	30		3	
DBS	KUNDEN	SALDO	NUMBER	22		4	
DBS	KUNDEN	RABATT	NUMBER	22		5	
DBS	PRODUKTE	PNR	NUMBER	22		1	
SYS	ALL_TABLES	OWNER	CHAR	30		1	
SYS	ALL_TABLES	TABLE_NAME	CHAR	30		2	

Programmieren mit dynamischem ESQL

Einfacher Fall: Resultatschema und Queryparameter statisch festgelegt

- 1) Deklaration der Wirtsprogrammvariablen
- 2) Aufbau der SQL-Anweisung als Zeichenkette
- 3) Dynamische Precompilation: PREPARE DynQuery FROM :sqltext
- 4) DECLARE Iterator CURSOR FOR DynQuery
- 5) Auswertung der Eingabeparameter und Vorbereitung der
"Cursor"-Schleife: OPEN Iterator
- 6) FETCH Iterator INTO Wirtsprogrammvariablen
- 7) Resultatsattribute in den Wirtsprogrammvariablen verarbeiten
- 8) Wiederholung der Schritte 6 und 7 für jedes Resultatstupel
- 9) CLOSE Iterator

Sonderfall: höchstens ein Resultatstupel

statt 3) bis 9) einfacher: EXECUTE IMMEDIATE :sqltext

Komplexer Fall:

Resultatschema oder Queryparameter dynamisch festgelegt

Programmbeispiel mit dynamischem ESQL (1)

```
#include <stdio.h>
#include <string.h>
#define TRUE 1
#define FALSE 0
#define MAXPRODUKTE 10
EXEC SQL INCLUDE SQLCA;
main () {
EXEC SQL BEGIN DECLARE SECTION;
    int monat, tag, pnr, menge;
    VARCHAR bez[31];
    VARCHAR sqltext[512];
    VARCHAR user[20];
    VARCHAR passwd[10];
EXEC SQL END DECLARE SECTION;
EXEC SQL WHENEVER SQLERROR GOTO handle_error;
typedef char prod_bez_t[31];
typedef int bool;
prod_bez_t      prod_bez[10];
int             i;
int             num_of_prod;
bool            found;
```

Programmbeispiel mit dynamischem ESQL (2)

```
printf ("Benutzername?"); scanf ("%s", &(user.arr)); user.len = strlen(user.arr);
printf ("Kennwort?"); scanf ("%s", &(passwd.arr)); passwd.len = strlen(passwd.arr);
EXEC SQL CONNECT :user IDENTIFIED BY :passwd;
i = 0;
printf ("%s%s", "Produktbezeichnung oder \"ende\" \n"); scanf ("%s", &(prod_bez[i]));
while ((i < MAXPRODUKTE) && (strcmp (prod_bez[i], "ende") != 0)) {
    i++;
    printf ("%s%s", "Produktbezeichnung oder \"ende\" \n");
    scanf ("%s", &(prod_bez[i])); }; /*while*/
num_of_prod = i;
if (num_of_prod == 0) exit(-1);
/* Aufbau der SQL-Anfrage */
strcpy (sqltext.arr, "SELECT Monat, Tag, Bestellungen.PNr, Bez, Menge ");
strcat (sqltext.arr, "FROM Bestellungen, Produkte ");
strcat (sqltext.arr, "WHERE Bestellungen.PNr = Produkte.PNr ");
strcat (sqltext.arr, "AND ( "); strcat (sqltext.arr, "Bez LIKE \"");
strcat (sqltext.arr, prod_bez[0]); strcat (sqltext.arr, "%\" ");
for (i=1; i < num_of_prod; i++) {
    strcat (sqltext.arr, "OR Bez LIKE \""); strcat (sqltext.arr, prod_bez[i]);
    strcat (sqltext.arr, "%\" "); }; /*for*/
strcat (sqltext.arr, ") "); strcat (sqltext.arr, "ORDER BY Monat DESC, Tag DESC");
sqltext.len = strlen(sqltext.arr);
```


Programmbeispiel mit dynamischem ESQL (3)

```
EXEC SQL PREPARE DynQuery FROM :sqltext;
EXEC SQL DECLARE BestIterator CURSOR FOR DynQuery;
EXEC SQL OPEN BestIterator;
found = TRUE;
while (found) {
    EXEC SQL FETCH BestIterator INTO :monat, :tag, :pnr, :bez, :menge;
    if (sqlca.sqlcode == 0) {
        bez.arr[bez.len] = '\0';
        printf ("%d.%d.: %d Stueck %s (PNr %d)\n", tag, monat, menge, bez.arr, pnr); }
    else
        found = FALSE;
}; /*while*/
EXEC SQL CLOSE BestIterator;
EXEC SQL COMMIT WORK RELEASE;
exit (0);
/* Fehlerbehandlung */
handle_error:
    printf ("\n*** Error %d in program. ***\n", sqlca.sqlcode);
    printf ("%s\n", sqlca.sqlerrm.sqlerrmc);
    exit (-1);
} /*main*/
```

Stored Procedures in 4GL PL/SQL

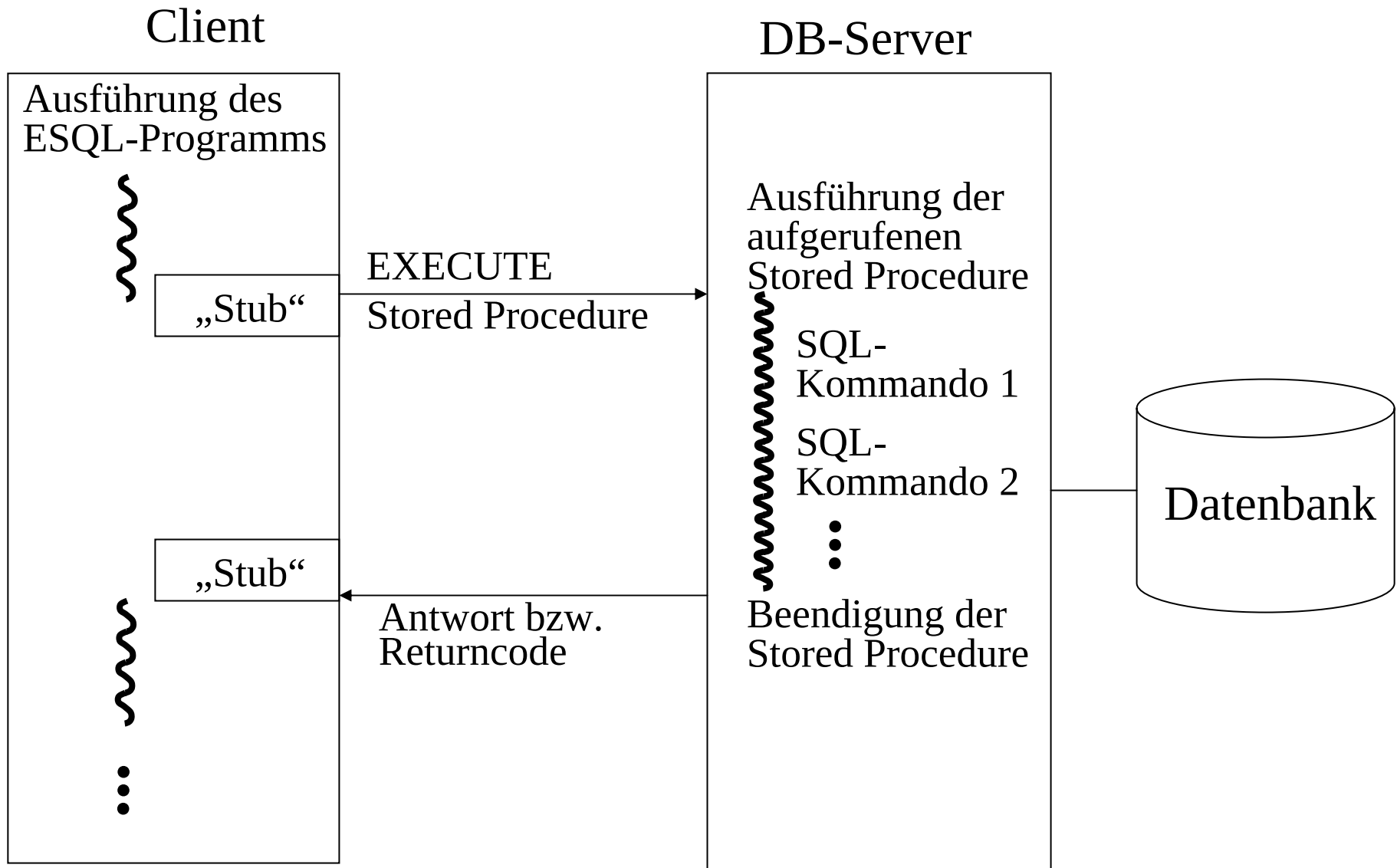
```
CREATE PROCEDURE Lager_Auffuellen AS
DECLARE ...;
BEGIN
DECLARE CURSOR Knappe_Produkte IS
    SELECT PNr, Vorrat FROM Produkte WHERE Vorrat < 100;
KP Knappe_Produkte%ROWTYPE;
BEGIN
    OPEN Knappe_Produkte;
    LOOP
        FETCH Knappe_Produkte INTO KP;
        EXIT WHEN Knappe_Produkte%NOTFOUND;
        ...
        IF KP.Vorrat > 10 THEN
            EXECUTE Nachbestellen (KP.PNr, 1000 - KP.Vorrat)
        ELSE
            EXECUTE Express_Bestellen (KP.PNr, 10);
            EXECUTE Nachbestellen (KP.PNr, 1000 - KP.Vorrat - 10);
        END IF;
    END LOOP;
END; END;

EXECUTE Lager_Auffuellen;
```

Prozedur-
deklara-
tion

Prozedur-
aufruf

Laufzeitarchitektur für Stored Procedures



8.3 JDBC (Java Database Connectivity)

```
import java.io.*; import java.util.*; import java.sql.*;
public class myjdbc {
public static void main (String args[]) {

Connection con = null; Statement stmt = null;
String sqlstring;

String driver = "oracle.jdbc.driver.OracleDriver";
try { Class.forName(driver);}
    catch(Exception e) {
        System.out.println ("error when loading jdbc driver");};

try {
String jdbcURL = "jdbc:oracle:thin:";
String db = "(DESCRIPTION =(ADDRESS_LIST = (ADDRESS = " +
    "(PROTOCOL = TCP)(HOST = TOKYO)(PORT = 1521)))" +
    "(CONNECT_DATA =(SERVICE_NAME = TOKYO.WORLD)))";
String user = "lehre40"; String password = "leere4zig";
con = DriverManager.getConnection
    (jdbcURL + "@" + db, user, password);
con.setAutoCommit (false);
stmt = con.createStatement(); }
    catch (Exception e) {
        System.out.println ("error with jdbc connection"); };
}
```

JDBC-Beispiel (2)

```
String inputline = null;
try{
System.out.println ("\n Please type name. \n");
DataInputStream myinput = new DataInputStream (System.in);
inputline = myinput.readLine(); }
catch (IOException e) {System.out.println ("IO error");};

try{
sqlstring = "SELECT * FROM CUSTOMER " +
            "WHERE NAME LIKE '" + inputline + "'";
ResultSet result = stmt.executeQuery (sqlstring);
System.out.println ("CUSTOMER:");
System.out.println ("=====");
while (result.next()) {
    String namevar = result.getString("NAME");
    String ipnumbervar = result.getString("IPNUMBER");
    String cityvar = result.getString("CITY");
    System.out.println ("NAME: " + namevar + " IPNUMBER: "
                        + ipnumbervar + " CITY: " + cityvar);
}; //while
con.commit(); }
catch (SQLException sqlex)
{System.out.println (sqlex.getMessage());};
System.out.println ("\n");
```

JDBC-Beispiel (3)

```
try {
sqlstring = "SELECT * FROM ACCOUNT " +
            "WHERE NAME LIKE '" + inputline + "'";
ResultSet result = stmt.executeQuery (sqlstring);

System.out.println ("ACCOUNT:");
System.out.println ("=====");
while (result.next()) {
    String namevar = result.getString("NAME");
    int balancevar = result.getInt("BALANCE");
    System.out.println ("NAME: " + namevar +
                        " BALANCE: " + balancevar);
}; //while
con.commit(); }
catch (SQLException sqlex)
{System.out.println (sqlex.getMessage());};

try{ con.close(); }
catch (SQLException sqlex)
{System.out.println (sqlex.getMessage());};

}; //main
} //myjdbc
```

8.4 Web-Anwendungen mit Servlets

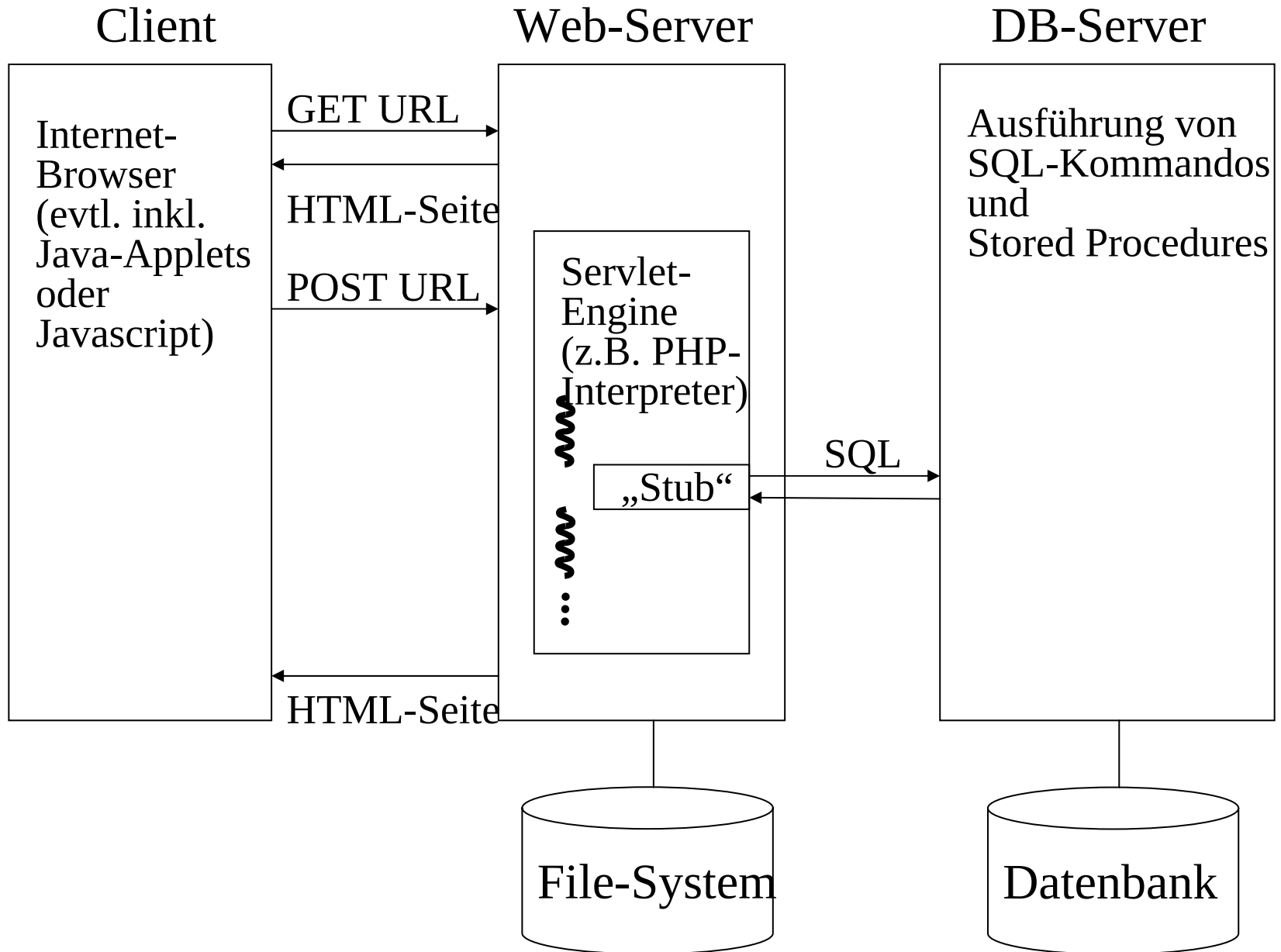
3-Schichten-Architektur (3-Tier Architecture):

- Client (Front-End):
Präsentation / GUI mittels Internet-Browser
(plus ggf. Javascript oder Java Applets)
- Applikations-Server (Middle Tier):
Applikationslogik gesteuert durch Servlet-Engine
(z.B. Apache/Tomcat und JVM mit JSDK,
Apache plus PHP-Interpreter)
mit Datenbankzugriffen via JDBC bzw. ODBC
- Daten(bank)-Server (Back-End)

Vorteile gegenüber Applikationslogik im Client (z.B. Applets):

- einfacheres Management (Installation, Softwarepflege)
- höhere Sicherheit (Applikationscode unsichtbar für Client)
- bessere Effizienz und Skalierbarkeit (z.B. weniger DB-Sessions)

Laufzeitarchitektur für Servlets



Servlets in Java

anhand eines typischen Beispiels einer Web-Anwendung

basierend auf folgender DB:

```
create table customer (name varchar(20) primary key,  
                        ipnumber varchar(20) not null, city varchar(20));  
create table account (name varchar(20) primary key, balance integer);  
create table history (name varchar(20),  
                      bookingdate date, amount integer);
```

mit der folgenden Applikationslogik:

```
look up customer in database  
check if customer has filled form  
if (form) parameters available  
then  
    insert customer info into database  
    send personalized greeting  
else send blank form  
create output with links to „transfer“ and „audit“ servlets
```

Servlet-Beispiel (1)

```
import java.lang.*; import java.io.*; import java.util.*;
import javax.servlet.*; import javax.servlet.http.*;
import java.sql.*; import java.net.*;
public class welcome extends HttpServlet {

    public void doGet
    (HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

        String namevar; String cityvar;
        BufferedReader readervar; String linestring;
        String ipnumbervar = null; String browservar = null;
        boolean newcustomer; String nexturl;

        PrintWriter out = response.getWriter();
        out.println("<html><body>");
        namevar = request.getParameter ("inputname");
        cityvar = request.getParameter ("inputcity");
        ipnumbervar = request.getRemoteAddr();
        // create JDBC connection
        // ...
    }
}
```

Servlet-Beispiel (2)

```
// actual application code
newcustomer = true;
try{
sqlstring = "SELECT * FROM CUSTOMER " +
            "WHERE IPNUMBER='" + ipnumbervar + "'";
ResultSet result = stmt.executeQuery (sqlstring);
if (result.next()) {
    newcustomer = false;
    namevar = result.getString("NAME");
    cityvar = result.getString("CITY");
}; //if
}
catch (SQLException sqlex)
    {System.out.println (sqlex.getMessage());};
```

Servlet-Beispiel (3)

```
if ((namevar != "") & (namevar != null)) {
    // insert customer info into db
    if (newcustomer){
        try {
            sqlstring = "INSERT INTO CUSTOMER " +
                "(NAME,IPNUMBER,CITY) VALUES (' " + namevar +
                "', '" + ipnumbervar + "', '" + cityvar + "')" ;
            stmt.execute (sqlstring);
            sqlstring = "INSERT INTO ACCOUNT (NAME,BALANCE) " +
                " VALUES (' " + namevar + "',0)";
            stmt.execute (sqlstring); con.commit(); }
            catch (SQLException sqlex)
                {System.out.println (sqlex.getMessage());};
        } } //then
    else {
        // post form for inquiring customer info
        out.println("<form method=post action=welcome>");
        out.println("<br> Please type your name and city. <br>");
        out.println("Name: <input type=text name=inputname><br>");
        out.println("City: <input type=text " +
            "name=inputcity value=Saarbruecken><br>");
        out.println("<input type=submit name=inputenter " +
            "value=\"Here you go!\"><br>");
        out.println("</form>");
    }; //if
};
```

Servlet-Beispiel (4)

```
if ((namevar != "") && (namevar != null)) {
    out.println ("Welcome, " + namevar + "!<br>");
    out.println ("Today's date is ");
    out.println (new java.util.Date());
    out.println ("<br>");
    out.println ("Your IP number is " + ipnumbervar + "<br>");
    out.println ("How is the weather in " +
        cityvar + "?<br><br>");
    out.println ("How can we help you today?<br>");
    nexturl = "transfer?customername=" +
        URLEncoder.encode(namevar);
    out.println ("<a href=" + nexturl +
        "> Deposit or withdraw money</a><br>");
    nexturl = "audit?customername=" +
        URLEncoder.encode(namevar);
    out.println ("<a href=" + nexturl +
        "> Audit trail of your account</a><br>");
}; //if
out.println("</body></html>");
} //doGet
```

Servlet-Beispiel (5)

```
public void doPost
(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

    doGet(request, response);

} // doPost

} // class welcome
```

Servlet mit Session-Verwaltung (1)

- Sitzungsdaten über mehrere Dialogschritte hinweg
- Session-Objekt wird von Servlet-Engine verwaltet und ist für Servlet zugreifbar
- Cookie enthält Session-Identifizier

```
import java.lang.*; import java.io.*; import java.util.*;
import javax.servlet.*; import javax.servlet.http.*;
public class sessionexample extends HttpServlet {

public void doGet
(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

Integer mycounter;
PrintWriter out = response.getWriter();
out.println("<html><body>");
out.println("<h1>Welcome!</h1><br><br>");
```

Servlet mit Session-Verwaltung (2)

```
// create session if not yet existing,  
// otherwise fetch session data  
HttpSession session = request.getSession(true);  
if (session.isNew()) {  
    mycounter = new Integer(1);  
    session.putValue("mycounter", mycounter); }  
else {  
    mycounter = (Integer) session.getValue("mycounter");  
    mycounter = new Integer(mycounter.intValue() + 1);  
    out.println("You are visiting us the ");  
    out.println(mycounter.intValue());  
    out.println("th time");  
    session.putValue("mycounter", mycounter);  
}; //if  
out.println("</body></html>");  
} //doGet  
  
} // class sessionexample
```


Kapitel 9: Integritätssicherung

9.1 Integritätsbedingungen

- Constraints und Assertions
- Triggers

9.2 Views

9.3 Zugriffskontrolle

Ziel: Integritätsregeln in DB verankern

- effektivere Integritätssicherung
- einfachere Anwendungsprogrammierung

Typen von Integritätsbedingungen

- Statische Integritätsbedingungen
 - datenmodellinhärente Invarianten
 - Primärschlüsselbedingung, Fremdschlüsselbedingung
 - anwendungsspezifische Invarianten
 - für ein Attribut eines Tupels
 - für ein Tupel
 - für mehrere Tupel einer Relation
 - für mehrere Relationen
- Dynamische Integritätsbedingungen

Zeitpunkt der Prüfung:

- am Ende einer SQL-Anweisung
- am Ende einer Transaktion

Reaktion auf Integritätsverletzungen:

- Nichtausführung bzw. Rückgängigmachen der Anweisung
- Abbruch der Transaktion
- Ausführung von Folgeänderungen

Beispiele

Statische Bedingungen:

- 1) Der Rabatt eines Kunden darf nicht über 50 % liegen.
- 2) Der Rabatt eines ausländischen Kunden darf nicht über 30 % liegen.
(Annahme: Es gibt ein zusätzliches Kundenattribut Land.)
- 3) Der durchschnittliche Rabatt aller Kunden darf 30 % nicht überschreiten.
- 4) Der Gesamtwert aller Produkte im selben Lager darf 1 Mio. DM nicht überschreiten.
- 5) Es muß mindestens ein Produkt geben.
- 6) Die Rechnungssumme einer Bestellung ist das Produkt von Preis und bestellter Menge des bestellten Produkts abzüglich des Kundenrabatts.
- 7) Der Saldo eines Kunden ist die (negative) Summe der Rechnungssummen aller noch nicht bezahlten Bestellungen des Kunden.

Dynamische Bedingungen:

- 8) Der Rabatt eines Kunden darf nie reduziert werden.
- 9) Der Rabatt eines Kunden darf in 1 Jahr um max. 10 % erhöht werden.
- 10) Von Kunden, deren Saldo unter - 100000 DM liegt, werden keine Bestellungen mehr angenommen.
- 11) Der Status einer Bestellung darf sich nur in "geliefert" ändern, der Status einer gelieferten Bestellung nur in "bezahlt".
Der Status einer bezahlten Bestellung darf sich nie mehr ändern.

9.1 Ausdrucksmittel zur Spezifikation von Integritätsbedingungen

- Constraints beim Create Table
- Create Assertion
- Create Trigger

} nur statische
Integritätsbedingungen

Syntax von Constraints und Assertions

```
CREATE TABLE tablename
( colname datatype [DEFAULT {defaultconst | NULL}]
  [colconstraint {, colconstraint ...}],
  {, colname datatype [DEFAULT {defaultconst | NULL}]
    [colconstraint {, colconstraint ...}] ...} )
[tabconstraint {, tabconstraint ...}]
colconstraint ::= NOT NULL |
[CONSTRAINT constrname]
  { UNIQUE | PRIMARY KEY | CHECK (searchcond) |
    REFERENCES tablename [(colname)] [...] }
tabconstraint ::=
[CONSTRAINT constrname]
  { UNIQUE colname {, colname ...} |
    PRIMARY KEY colname {, colname ...} |
    CHECK (searchcond) |
    FOREIGN KEY colname {, colname ...}
    REFERENCES tablename [(colname)] [...] }
CREATE ASSERTION assertname CHECK (searchcond)
[ INITIALLY {DEFERRED | IMMEDIATE} ]
```

Semantik von Constraints und Assertions

Semantik von CHECK (searchcond)

mit $\text{sql2trc}[\text{searchcond}] = F(t_1, \dots, t_n)$ mit freien Variablen $t_1, \dots, t_n$:

$$\left. \forall t_1 \dots \forall t_n ((t_1 \in R_1 \wedge \dots \wedge t_n \in R_n) \Rightarrow F(t_1, \dots, t_n)) \right\} I_j$$

Semantik der Datenbank aus logischer Sicht: $H \wedge I_1 \wedge \dots \wedge I_m$
mit elementarer Formel (Fakten) H und Integritätsbed. $I_1, \dots, I_m$

Mögliche Implementierung:

bei potentieller Verletzung von I_j Ausführen von

`SELECT t1, ..., tn FROM R1, ..., Rn WHERE NOT searchcond`

und Test auf Leerheit des Resultats

Zeitpunkt der Integritätsprüfung:

nach der SQL-Anweisung (... IMMEDIATE) oder
am Ende der Transaktion (... DEFERRED)

Reaktion bei Integritätsverletzung: Rollback

Beispiele: Constraints und Assertions

Bedingungen 1, 2, 3:

```
CREATE TABLE Kunden ( ... Rabatt FLOAT
    CONSTRAINT Rabattbedingung CHECK (Rabatt BETWEEN 0.0 AND 0.5)
    CONSTRAINT Auslandsrabattbedingung CHECK (Land = 'D' OR Rabatt <= 0.3), ...,
    CONSTRAINT Durchschnittsrabattbedingung CHECK
        (0.3 >= (SELECT AVG(Rabatt) FROM Kunden)) )
```

Bedingung 4:

```
CREATE TABLE Produkte ( ...,
    CONSTRAINT Lagerwertbedingung CHECK (1000000.0 >= ALL
        (SELECT SUM(Vorrat*Preis) FROM Produkte GROUP BY Lager)) )
```

Bedingung 5:

```
CREATE ASSERTION Produktexistenzbedingung CHECK
    (EXISTS (SELECT * FROM Produkte) )
```

Bedingung 6:

```
CREATE ASSERTION Rechnungssummenbedingung CHECK (Bestellungen.Summe =
    (SELECT B.Menge * Produkte.Preis * (1.0 - Kunden.Rabatt)
    FROM Bestellungen B, Produkte, Kunden WHERE B.PNr=Produkte.PNr
    AND B.KNr=Kunden.KNr AND B.BestNr=Bestellungen.BestNr ) )
```

Bedingung 7:

```
CREATE ASSERTION Saldobedingung CHECK ( Kunden.Saldo =
    (SELECT SUM(Summe) FROM Bestellungen WHERE
    Bestellungen.KNr=Kunden.KNr AND Status <> 'bezahlt') )
```

Optionen für Fremdschlüsselbedingung

Grobsyntax:

```
... FOREIGN KEY ( column {, column ...} )  
  REFERENCES [user .] table [ ( column {, column ...} ) ]  
  [ON DELETE {NO ACTION | CASCADE | SET NULL | SET DEFAULT}]  
  [ON UPDATE {NO ACTION | CASCADE | SET NULL | SET DEFAULT}]  
  [INITIALLY {IMMEDIATE | DEFERRED}] ...
```

Semantik von

CREATE TABLE R ...

FOREIGN KEY A1, ..., Am REFERENCES R1 (B1), ..., Rm (Bm):

$$\forall t (t \in R \Rightarrow ((t.A1 = \omega \wedge \dots \wedge t.Am = \omega) \vee \\ (\exists t1 \dots \exists tm (t1 \in R1 \wedge \dots \wedge tm \in Rm \wedge \\ t1.B1 = t.A1 \wedge \dots \wedge tm.Bm = t.Am)))))$$

Reaktion bei Integritätsverletzung:

- Zurückweisung der Löschung/Änderung (bei NO ACTION)
- Löschen/Ändern aller "abhängigen" Tupel (bei CASCADE)
- Fremdschlüssel in "abhängigen" Tupeln auf Default-/Nullwert setzen

Beispiel: Fremdschlüsselbedingung

Kunden

KNr	...
1	
2	
...	

Produkte

PNr	...
1	
2	
...	

Bestellungen

BestNr	Monat	Tag	KNr	Summe	Status
1001			1		
1002			2		
1003			1		
...					

Bestellposten

BestNr	PNr	Menge
1001	1	
1002	2	
1003	1	
1003	2	
...		

```
CREATE TABLE Bestellungen ( ... ,  
    FOREIGN KEY KNr REFERENCES Kunden (KNr) ON DELETE SET NULL )  
CREATE TABLE Bestellposten ( ... ,  
    FOREIGN KEY PNr REFERENCES Produkte (PNr) ON DELETE CASCADE,  
    FOREIGN KEY BestNr REFERENCES Bestellungen (BestNr)  
        ON DELETE CASCADE )
```

Beispiel: Fremdschlüsselbedingung

Kunden

KNr	...
1	
2	
...	

Produkte

PNr	...
1	
2	
...	

Bestellungen

BestNr	Monat	Tag	KNr	Summe	Status
1001			Null		
1002			2		
1003			Null		
...					

Bestellposten

BestNr	PNr	Menge
1001	Null	
1002	2	
1003	Null	
1003	2	
...		

```
CREATE TABLE Bestellungen ( ... ,  
    FOREIGN KEY KNr REFERENCES Kunden (KNr) ON DELETE SET NULL )  
CREATE TABLE Bestellposten ( ... ,  
    FOREIGN KEY PNr REFERENCES Produkte (PNr) ON DELETE CASCADE,  
    FOREIGN KEY BestNr REFERENCES Bestellungen (BestNr)  
        ON DELETE CASCADE )
```

Syntax und Semantik von CREATE TRIGGER

Grobsyntax:

```
CREATE TRIGGER trigger-name {BEFORE | AFTER }  
{ DELETE | INSERT | UPDATE [OF column {, column ...}] }  
ON table [ REFERENCING OLD AS corrvar NEW AS corrvar ]  
[ FOR EACH ROW | FOR EACH STATEMENT ]  
[ WHEN ( condition ) ] ( statement-sequence )
```

Semantik mit sql2trc '[condition] = F:

Werte F bei spez. Ereignis aus und starte Aktion, falls F wahr

- Fall 1 (FOR EACH STATEMENT):

F hat keine freien Variablen

- Fall 2 (FOR EACH ROW):

F hat eine oder zwei freie Variablen – told und tnew -, die an den alten bzw. neuen Wert des jeweiligen Tupels gebunden werden

Beispiele: Trigger

Bedingung 7:

```
CREATE TRIGGER Saldoeintrag
AFTER INSERT ON Bestellungen FOR EACH ROW WHEN ( Status = 'neu' )
( UPDATE Kunden SET Saldo = Saldo - Summe
  WHERE Kunden.KNr=Bestellungen.KNr );
CREATE TRIGGER Saldoausgleich
AFTER UPDATE OF Status ON Bestellungen
REFERENCING OLD AS BOLD NEW AS BNew FOR EACH ROW
WHEN ( BNew.Status = 'bezahlt' AND BOLD.Status <> 'bezahlt' )
( UPDATE Kunden SET Saldo = Saldo + Summe
  WHERE Kunden.KNr=Bestellungen.KNr )
```

Bedingung 8:

```
CREATE TRIGGER Rabattmonotonie
AFTER UPDATE OF Rabatt ON Kunden
REFERENCING OLD AS KOld NEW AS KNew FOR EACH ROW
WHEN ( KNew.Rabatt < KOld.Rabatt ) ( ROLLBACK WORK )
```

Bedingung 10:

```
CREATE TRIGGER Kundensperrung
BEFORE INSERT ON Bestellungen FOR EACH ROW
WHEN ( (SELECT Saldo FROM Kunden
        WHERE Kunden.KNr=Bestellungen.KNr) < -100000.0 )
( <Fehlermeldung ausgeben>; ROLLBACK WORK )
```

Trigger zur Steuerung der “Business-Logik”

Beispiel:

```
CREATE TRIGGER Kundenbeförderung
AFTER INSERT ON Bestellungen
REFERENCING NEW AS BNew FOR EACH ROW
WHEN ( ( (SELECT SUM(Summe) FROM Bestellungen
        WHERE Bestellungen.KNr=BNew.KNr) > 100000.0 )
AND
      ( (SELECT SUM(Summe) - BNew.Summe FROM Bestellungen
        WHERE Bestellungen.KNr=BNew.KNr) <= 100000.0 ) )
( UPDATE Kunden SET Rabatt = Rabatt + 0.05
  WHERE Kunden.KNr=BNew.KNr;
  INSERT INTO Stammkunden
    SELECT * FROM Kunden WHERE Kunden.KNr=BNew.KNr )
```

Aber Achtung:

Die Effekte von Triggern, die selbst andere Trigger in bestimmten Reihenfolge „feuern“, sind u.U. sehr schwer vorhersagbar!

ECA-Regeln für Aktive Datenbanken

on event **if** condition **do** action

(mit allgemeineren Formen von Events und Actions)

Beispiele:

- 1) Auffüllen des Lagers für knappe Produkte automatisch initiieren:
on after update Produkte.Vorrat
if Produkte.Vorrat < 100
do execute LagerAuffüllen(...)
- 2) Werbebriefe drucken für Kunden,
die seit einem Jahr nichts mehr bestellt haben:
on 0:00 daily **do execute** JunkMail(...)
- 3) Verschicken eines Mahnbriefs an Kunden,
die dreimal hintereinander die Zahlungsfrist überschreiten.
- 4) Ausgabe einer Meldung, wenn der Kurs einer Aktie
innerhalb der letzten 5 Minuten um mehr als 50% variiert.

9.2 Views (Sichten, Virtuelle Relationen)

simplifizieren Integritätssicherung, Anfragen und Schemaänderungen

Grobsyntax:

```
CREATE VIEW view-name [ ( column {, column ...} ) ]  
AS select-block [ WITH CHECK OPTION ]
```

Beispiel:

gespeicherte Relationen:

Kunden (KNr, Name, Stadt, Rabatt)

Bestellungen (BestNr, Monat, Tag, KNr, PNr, Menge, Summe, Status)

zusätzliche View:

```
CREATE VIEW KundenInfo ( KNr, Name, Stadt, Rabatt, Saldo ) AS  
SELECT K.KNr, Name, Stadt, Rabatt, SUM(Summe) FROM Kunden K, Bestellungen  
WHERE K.KNr = Bestellungen.KNr AND Status <> 'bezahlt'  
GROUP BY K.KNr, Name, Stadt, Rabatt
```

Abfrage des Saldos:

```
SELECT Saldo FROM KundenInfo WHERE KNr=1
```

oder z.B. auch:

```
SELECT Saldo FROM KundenInfo, Bestellungen  
WHERE KundenInfo.KNr=Bestellungen.KNr AND BestNr=1001
```

Views auf Views

- 1) CREATE VIEW BestellungsInfo
(BestNr, Monat, Tag, KNr, Kundenname, Rabatt,
PNr, Produktbez, Menge, Summe, Status) AS
SELECT BestNr, Monat, Tag, Kunden.KNr, Name, Rabatt,
Produkte.PNr, Bez, Menge, Summe, Status
FROM Bestellungen, Kunden, Produkte
WHERE Bestellungen.KNr=Kunden.KNr
AND Bestellungen.PNr=Produkte.PNr
- 2) CREATE VIEW SuperBestellungsInfo AS
SELECT * FROM BestellungsInfo
WHERE Summe > 10000.0

Semantik von Views

Für CREATE VIEW VIRT AS viewquery ist

sql2ra [SELECT A1, ..., Am
FROM TAB1, ..., TABk, VIRT
WHERE F]
= $\pi[A1, \dots, Am] (\text{sql2ra}' [F] (TAB1 \times \dots \times TABk \times \text{sql2ra}[\text{viewquery}]))$

Beispiel:

sql2ra [SELECT * FROM SuperBestellungsInfo WHERE Rabatt>0.3]
= $\sigma[\text{Rabatt}>0.3] (\text{SuperBestellungsInfo})$
= $\sigma[\text{Rabatt}>0.3] (\sigma[\text{Summe}>10000.0] (\text{BestellungsInfo}))$
= $\sigma[\text{Rabatt}>0.3] (\sigma[\text{Summe}>10000.0]$
 $(\pi[\text{BestNr}, \dots] (\text{Kunden } |x| \text{ Bestellungen } |x| \text{ Produkte})))$

Einfachere Queries mit Views

Beispiel:

Prüfungen (Fach, Student, Note)

Welches ist das Fach mit der besten Durchschnittsnote?

```
a) SELECT Fach FROM Prüfungen GROUP BY Fach
    HAVING AVG(Note) <= ALL
                                ( SELECT AVG(Note) FROM Prüfungen
                                  GROUP BY Fach )
```

oder einfacher:

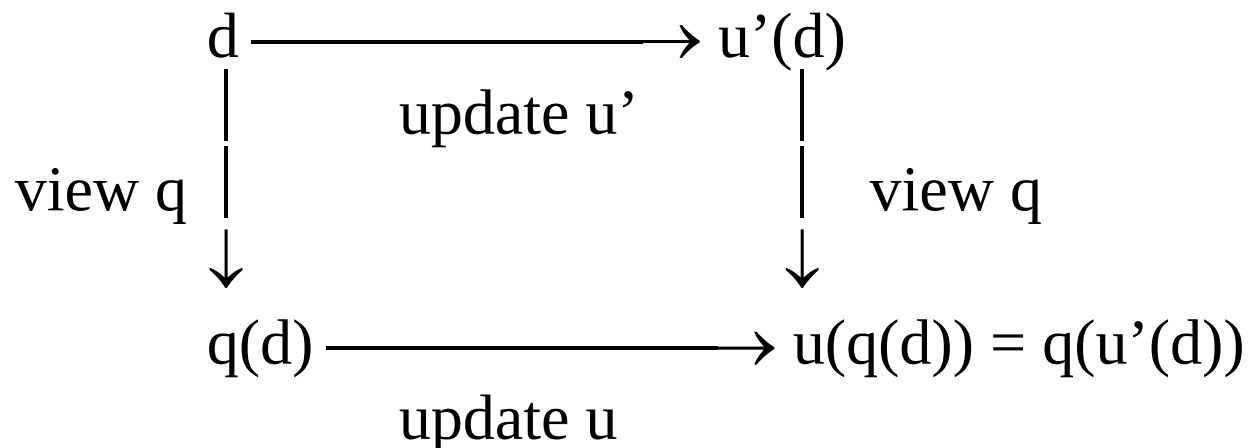
```
b) CREATE VIEW Fachstatistik AS
    SELECT Fach, AVG(Note) AS Durchschnitt FROM Prüfungen
    GROUP BY Fach;
SELECT Fach FROM Fachstatistik
WHERE Durchschnitt =
    (SELECT MIN(Durchschnitt) FROM Fachstatistik )
```

Updates auf Views

Sei D die Menge aller Datenbanken.

Views und Updates sind partielle Funktionen $q: D \rightarrow D$, $u: D \rightarrow D$.

Eine View q heißt *änderbar* genau dann,
wenn es für jede Datenbank d , auf der q definiert ist,
und jeden auf $q(d)$ definierten Update u
einen eindeutigen Update u' gibt,
der auf d definiert ist und für den **$u(q(d)) = q(u'(d))$** gilt.



Beispiele: View-Updates

```
CREATE VIEW KundenInfo ( KNr, Name, Stadt, Rabatt, Saldo ) AS
  SELECT K.KNr, Name,Stadt,Rabatt, SUM(Summe) FROM Kunden K, Bestellungen
  WHERE Kunden.KNr = Bestellungen.KNr AND Status <> 'bezahlt'
  GROUP BY KNr, Name, Stadt, Rabatt;
```

```
CREATE VIEW BestellungsInfo (BestNr, Monat, Tag, KNr, Kundenname, Rabatt,
  PNr, Produktbez, Menge, Summe, Status) AS
  SELECT BestNr, Monat, Tag, Kunden.KNr, Name, Rabatt, Produkte.PNr, Bez,
    Menge, Summe, Status FROM Bestellungen, Kunden, Produkte
  WHERE Bestellungen.KNr=Kunden.KNr AND Bestellungen.PNr=Produkte.PNr
```

- 1) UPDATE KundenInfo SET Stadt='Homburg' WHERE KNr=1
- 2) UPDATE BestellungsInfo SET Produktbez = 'Druckerpapier' WHERE Monat=11
- 3) UPDATE BestellungsInfo SET Produktbez = 'Druckerpapier' WHERE PNr=1
- 4) UPDATE BestellungsInfo SET Produktbez = 'Druckerpapier' WHERE BestNr=1
- 5) CREATE VIEW BestellungenKurzInfo (BestNr, Kundenname,Produktbez, Menge) AS
 SELECT BestNr, Name, Bez, Menge FROM Kunden, Bestellungen, Produkte
 WHERE Bestellungen.KNr=Kunden.KNr AND Bestellungen.PNr=Produkte.PNr;
 INSERT INTO BestellungenKurzInfo VALUES (1111, 'Hempel', 'Maus', 5)
- 6) UPDATE KundenInfo SET Saldo = Saldo + 1000.0 WHERE KNr=1

Views bei Schemaänderungen

bisherige Anfrage:

```
SELECT * FROM Kunden WHERE KNr=1
```

Schemaänderung (plus Datenbank aktualisieren oder neu laden):

1) ALTER TABLE Kunden

```
ADD Vorname CHAR(20), Nachname CHAR(20)
```

2) UPDATE Kunden SET Vorname = SUBSTR (Name, ...),
Nachname = SUBSTR (Name, ...)

3) ALTER TABLE Kunden DROP Name

Vorgehen zur "Bewahrung" der bisherigen Anfrage:

A) Kunden in KundenDaten umbenennen

B) CREATE VIEW Kunden (KNr, Name, Stadt, Rabatt, Saldo) AS
SELECT KNr, Vorname || ' ' || Nachname, Stadt, Rabatt, Saldo
FROM KundenDaten

9.3 Zugriffskontrolle

Begriffsklärung:

- Datenschutz (engl.: data privacy):
Einschränkungen bei Speicherung und Verarbeitung "kritischer" Daten
- Zugriffskontrolle / Autorisierung (engl.: data security, authorization):
Verhinderung von unbefugten Zugriffen auf gespeicherte Daten

Maßnahmen der Zugriffskontrolle:

- 1) Organisatorische Maßnahmen
(Rechner-/Netzzugang)
- 2) Technische Maßnahmen
(Verschlüsselung, Krypto-Protokolle)
- 3) Maßnahmen des Betriebssystems
(Firewalls, File-/Prozessisolation)
- 4) Authentikation des (DB-) Benutzers
- 5) Prüfung der Zugriffsrechte des DB-Benutzers beim Zugriff auf Daten

Rechtevergabe in SQL

Prinzipien:

Subjekte haben *Rechte* (zur Ausführung von Operationen) auf *Objekten*.

Der Erzeuger einer Tabelle ist deren Eigentümer.

Rechte sind minimal, d.h. beschränkt auf den Eigentümer und Subjekte, denen explizit Rechte erteilt wurden.

Vergabe von Rechten mit der GRANT-Anweisung in SQL.

Grobsyntax:

```
GRANT { ALL | privilege {, privilege ...} } ON { table | view }  
TO { PUBLIC | user {, user ...} } [ WITH GRANT OPTION ]
```

Mögliche Rechte sind:

SELECT	lesender Zugriff auf eine Relation
INSERT	Einfügen in eine Relation
UPDATE	Ändern von Tupeln einer Relation (ggf. nur bestimmte Attribute)
DELETE	Löschen von Tupeln einer Relation
CONNECT	Verbindung zum DBS aufnehmen ("Login"-Recht)
RESOURCE	Anlegen neuer Relationen (ggf. mit Limit für den Plattenplatz)
DBA	Datenbankadministration (z.B. Aufruf von Dienstprogrammen)
EXECUTE	Ausführung eines Anwendungsprogramms
IO_LIMIT	Beschränkung des Ressourcenverbrauchs für SQL-Anweisungen

Beispiele: Zugriffskontrolle mit SQL

- 1) Meier hat das SELECT-Recht auf Bestellungen.
GRANT SELECT ON Bestellungen TO Meier
- 2) Meier hat das Recht zur Ausführung des Programms Lieferung.
GRANT EXECUTE Lieferung TO Meier
- 3) Programm Lieferung hat das UPDATE-Recht auf Bestellungen.
GRANT UPDATE Status ON Bestellungen TO Lieferung
- 4) Meier hat das Recht zum Lesen der Kundendaten der Stadt Homburg.
CREATE VIEW KundenHOM AS SELECT * FROM Kunden
WHERE Stadt='Homburg';
GRANT SELECT ON KundenHOM TO Meier
- 5) Meier sei der Eigentümer der Relation MeierTabelle
Meier: GRANT SELECT ON MeierTabelle TO Schmid WITH GRANT OPTION
Schmid: GRANT SELECT ON MeierTabelle TO Schmitz WITH GRANT OPTION
Meier: REVOKE SELECT ON MeierTabelle FROM Schmid
Schmitz: GRANT SELECT ON MeierTabelle TO Schmid
REVOKE zieht (transitiv) auch das Recht von Schmitz zurück