Informationssysteme

Grundvorlesung Informatik
Sommersemester 2005
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/ss05/is05/

Organisation

- **Vorlesung:** Di 9-11 und Do 9 -11 in 45/002
  Sprechstunde Dr. Schenkel: Do 14-15 in 46/404 oder n.V. (ggf. e-mail)
  Sprechstunde Prof. Weikum: Di 14-15 in 46/401 oder n.V. (ggf. e-mail)
- **Übungsgruppen**: siehe Webseite
  Übungsgruppenleiter:
  Übungskoordination: 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 11.6., 2) Do 21.7., 3) Mitte Oktober
  - erfolgreiche Bearbeitung der praktischen Übungen
    (Teamarbeit in Dreiergruppen möglich)
  - Präsentation mindestens zweier Übungslösungen in der Übungsgruppe

  + Bonuspunkte für weitere Präsentationen
  + Bonuspunkte für besonders gute Lösung der praktischen Übungen

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: Such-maschinen

Teil II: Datenbank-schnittstellen

Teil III: Entwurf von Datenbanken und Anwendungen

Teil IV: Implementierungs-konzepte von DBS

Teil V: Informations-dienste

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

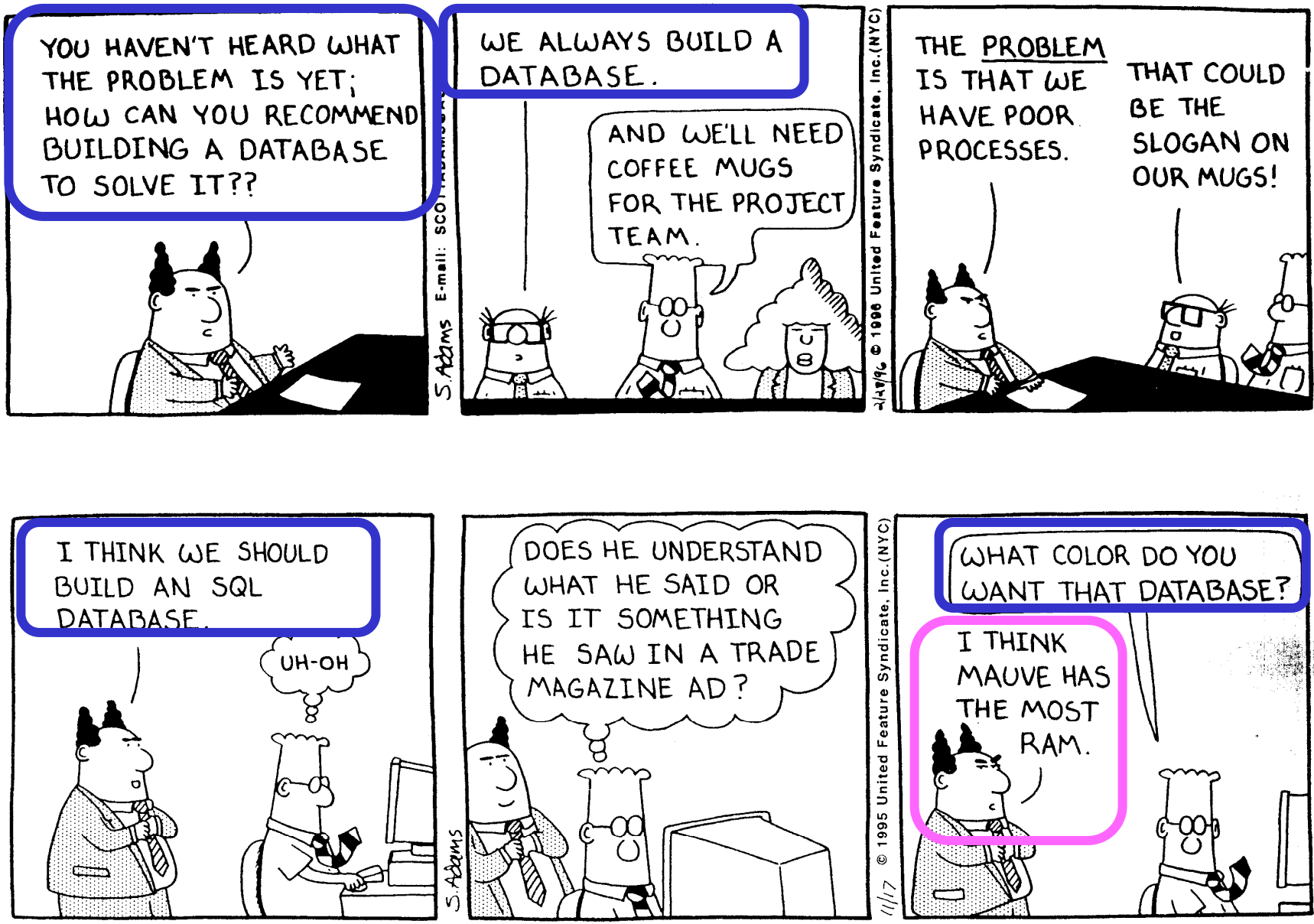
1.1 Anwendungen

1.2 Systemarchitekturen

1.3 Grundprinzipien von Datenbanksystemen

1.4 Grundprinzipien von Suchmaschinen

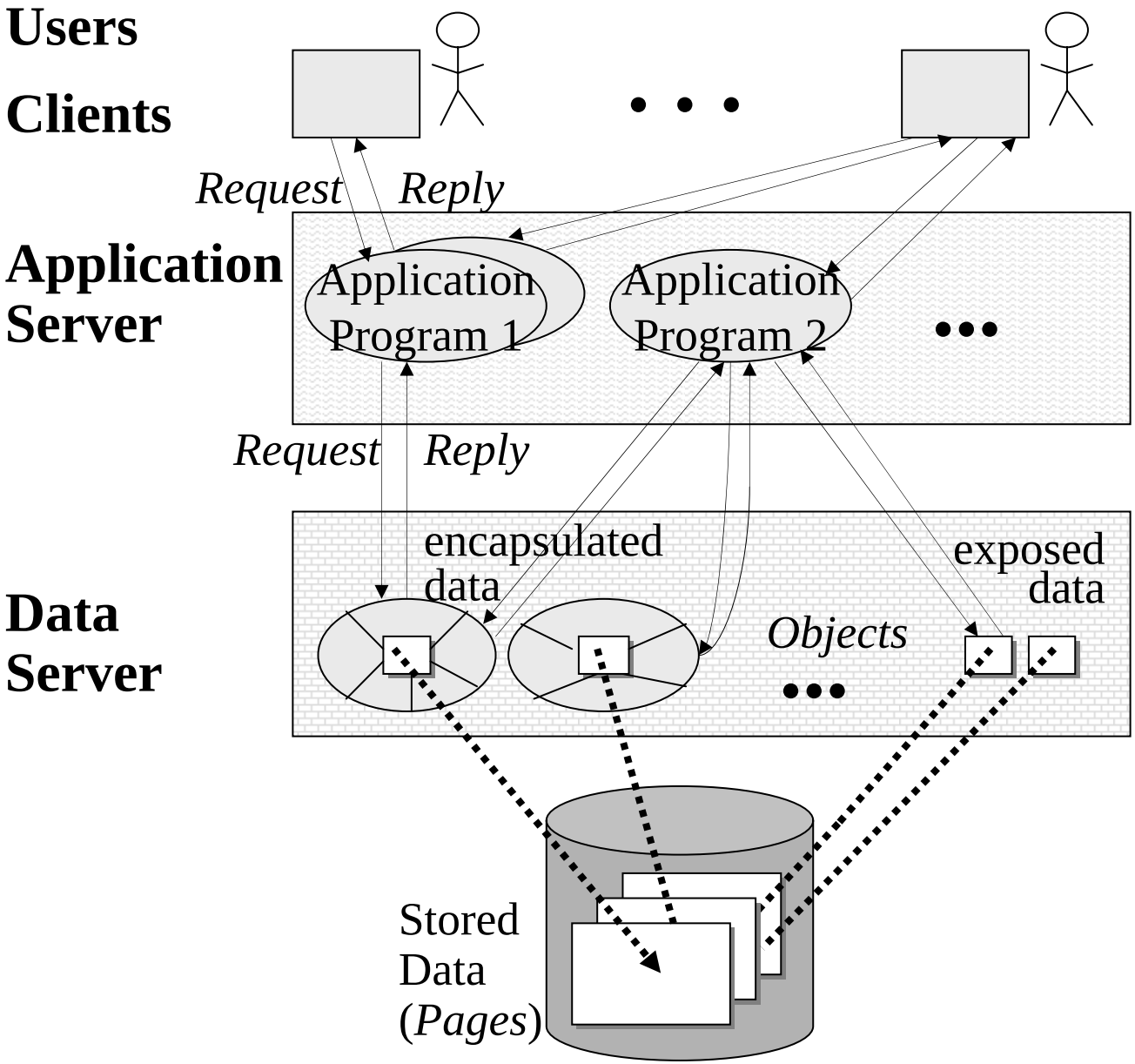
Warum sind Datenbanken und IS so wichtig ?



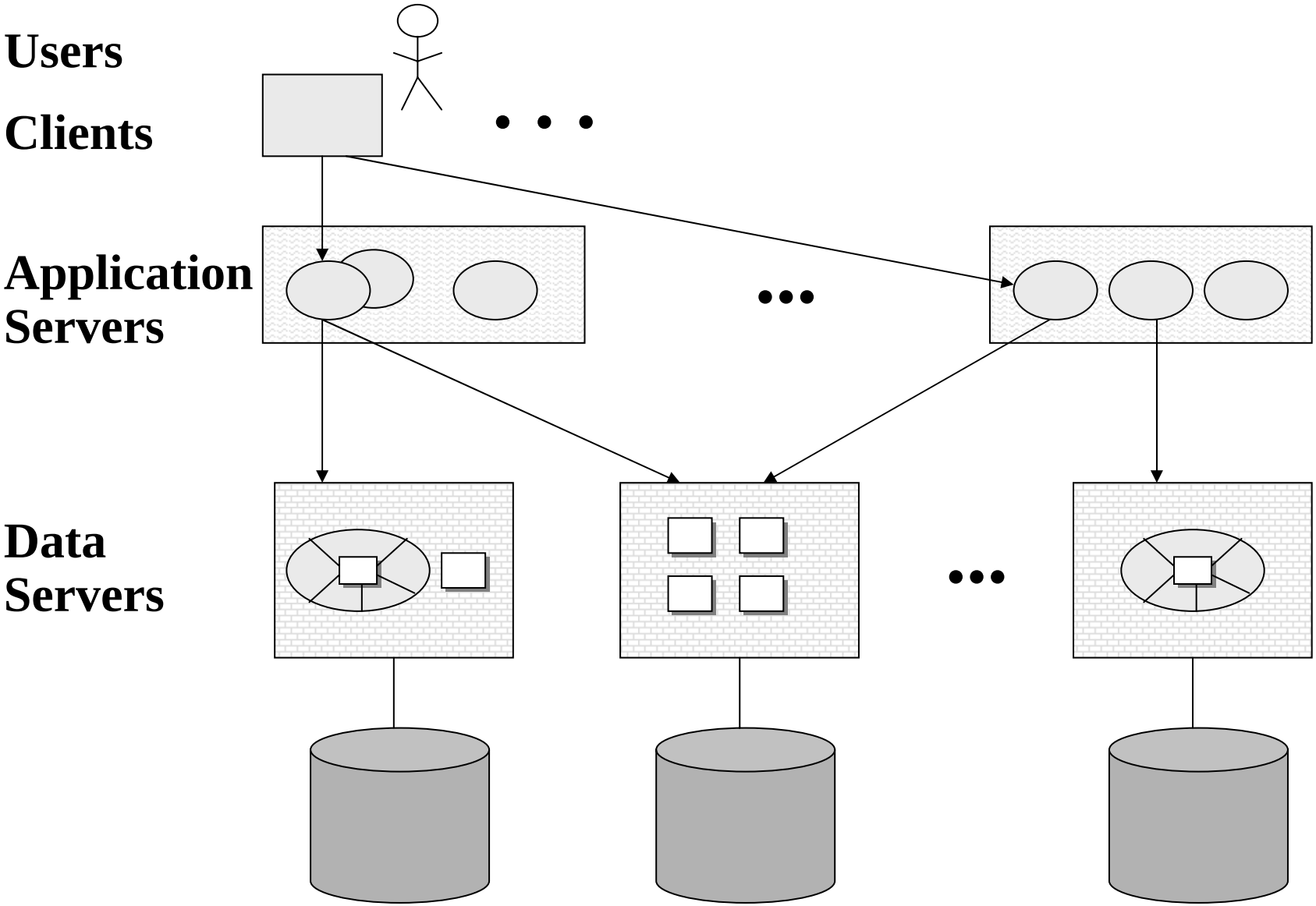
Technische Anforderungen an IS

- 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

IS mit 3-Ebenen-Architektur



Föderatives IS



Grundtugenden von DBS

- **Effiziente Verwaltung großer, persistenter Datenmengen mit Optimierung der Sekundärspeicherzugriffe**

- **Programm-Daten-Unabhängigkeit: Kapselung der Speicherungsstrukturen, so daß Optimierungen transparent für Anwendungsprogramme möglich sind**

- **Gewährleistung der Datenkonsistenz durch das DBS**

- **Datenbankänderungen innerhalb von Transaktionen: atomar, konsistenzerhaltend, isoliert, dauerhaft**

Grundtugenden von Suchmaschinen

- **Kompakte Repräsentation von Textdokumenten und Multimedia-Daten als Feature-Vektoren**
- **Ähnlichkeitssuche mit Resultats-Ranking nach Relevanz**
- **Algebraische Analysen und statistische Lernverfahren zur Organisation und Bewertung von Dokumenten**

Gliederung

1. Einführung und Überblick: Anwendungen, Systeme, Prinzipien
2. Vektorraummodell für Suchmaschinen
3. Automatische Klassifikation von Dokumenten
4. Linkanalyse für Autoritäts-Ranking

Teil I: Such-maschinen

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

Teil II: Datenbank-schnittstellen

11. Relationale Entwurfstheorie
12. Datenbankentwurf mit UML
13. Prozessmodellierung mit Statecharts

Teil III: Entwurf von Datenbanken und Anwendungen

14. Datenspeicherung, Indexstrukturen, Anfrageauswertung
15. Transaktionsverwaltung: Concurrency-Control
16. Transaktionsverwaltung: Recovery

Teil IV: Implementierungs-konzepte von DBS

17. OLAP und Data-Warehouses
18. Daten-Mining: Klassifikation, Assoziationsregeln
19. Semistrukturierte Daten und XML-Anfragesprachen

Teil V: Informations-dienste

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

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)

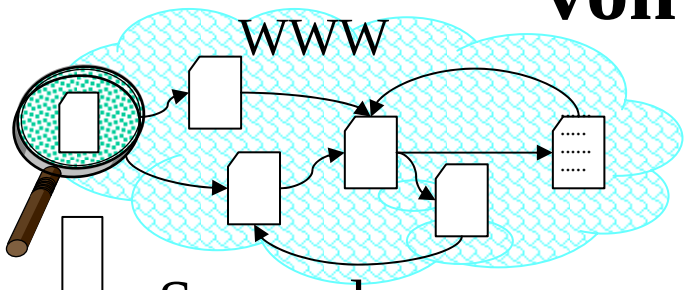
Globales Ziel:
Informationsbedürfnisse befriedigen - und dabei
Beseitigung des Engpasses (teurer) intellektueller Zeit !

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 Oracle/Text, XPath Full-Text)

Sammeln, Analyse und Indexierung von Web-Dokumenten



Sammeln (Crawling)

Web Surfing:
In Saarbrücker Internet-Cafes mit oder ohne Web Suit ...

Dokumente

Surfing
Saarbrücker
Internet
Cafes
...

Extraktion relevanter Wörter

Surf
Saarbrücken
Internet
Cafe
...

Linguist. Methoden: Reduktion auf Stamm-formen

Surf
Wellenritt
Internet
WWW
eService
Cafe
Bistro
...

Bildung von Features (Deskriptoren, Terme) mit Gewichten

Index-aufbau

Thesaurus (Ontologie)

Synonyme, Hypernyme, Hyponyme

Index (B*-Baum)

Bistro | Cafe | ...

URLs

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)

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.

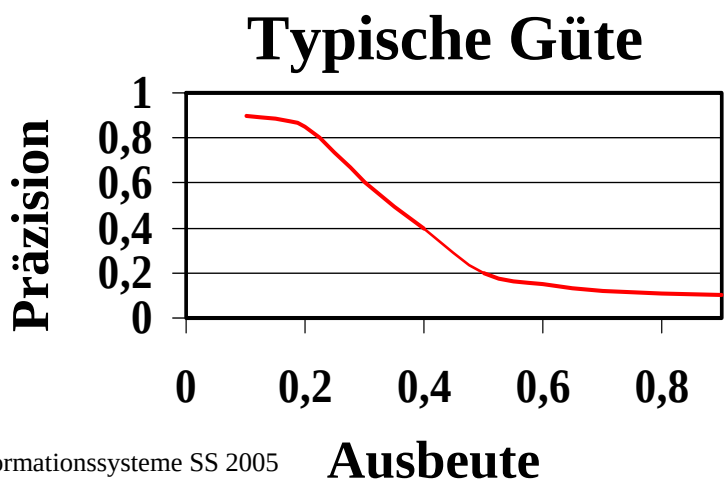
Bewertung der Retrieval-Güte (Effektivität)

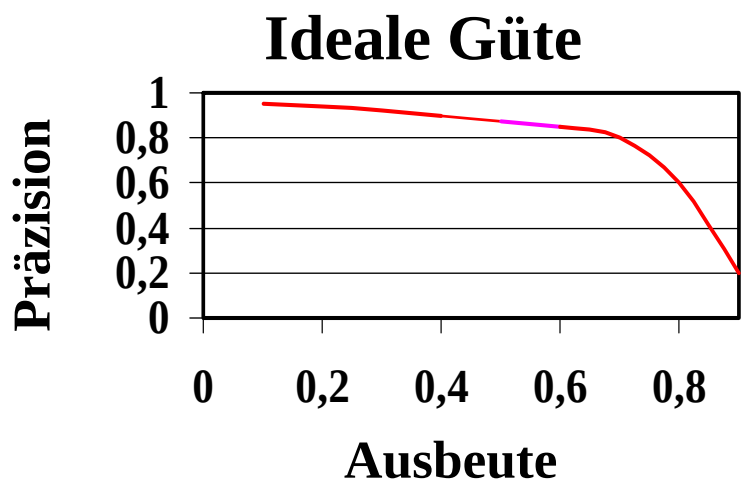
Fähigkeit, zu einer Anfrage **nur** relevante Dokumente zu liefern:

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

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

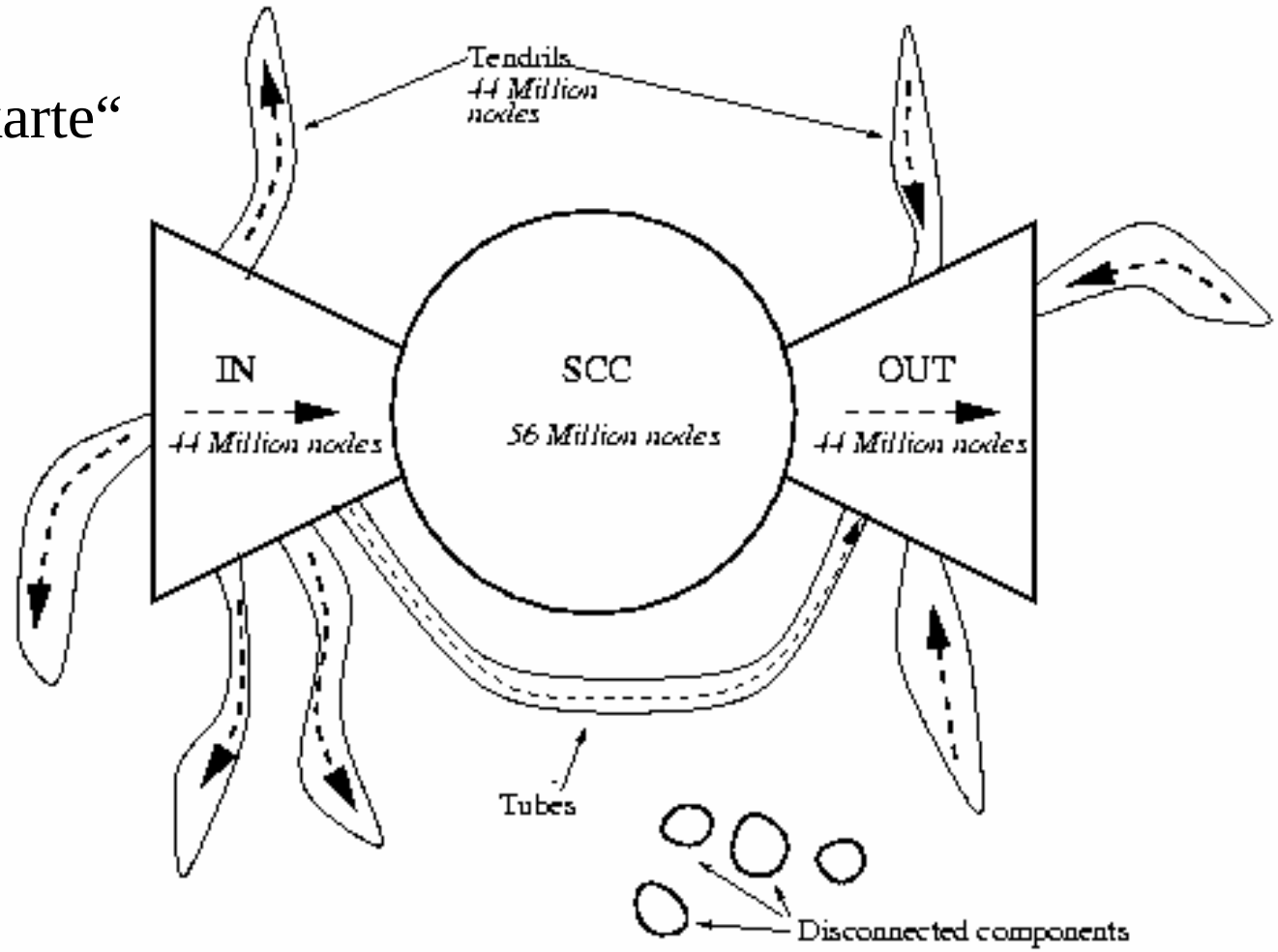
$$\textbf{\textit{Ausbeute (recall)}} = \frac{Anzahl\ relevanter\ Dokumente}{Anzahl\ aller\ relevanten\ Dokumente}$$





Problem: Effizienz und Skalierbarkeit

Aktuelle „Landkarte“ des Webs:



!!! Aber:

Suchmaschinen überdecken das „Surface Web“:
8 Mrd. Dokumente, 40 TBytes
Die meisten Daten sind im „Deep Web“ hinter Portalen:
> 500 Mrd. „Dokumente“, > 8 PBytes

Dimensionen sehr großer Web-Suchmaschinen

- > 8 Mrd. Web-Dokumente + 1 Mrd. News-Dokumente
  > 40 Terabytes Rohdaten
- > 10 Mio. Terme
  > 8 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.

2.2 Web-Crawling und Indexierung

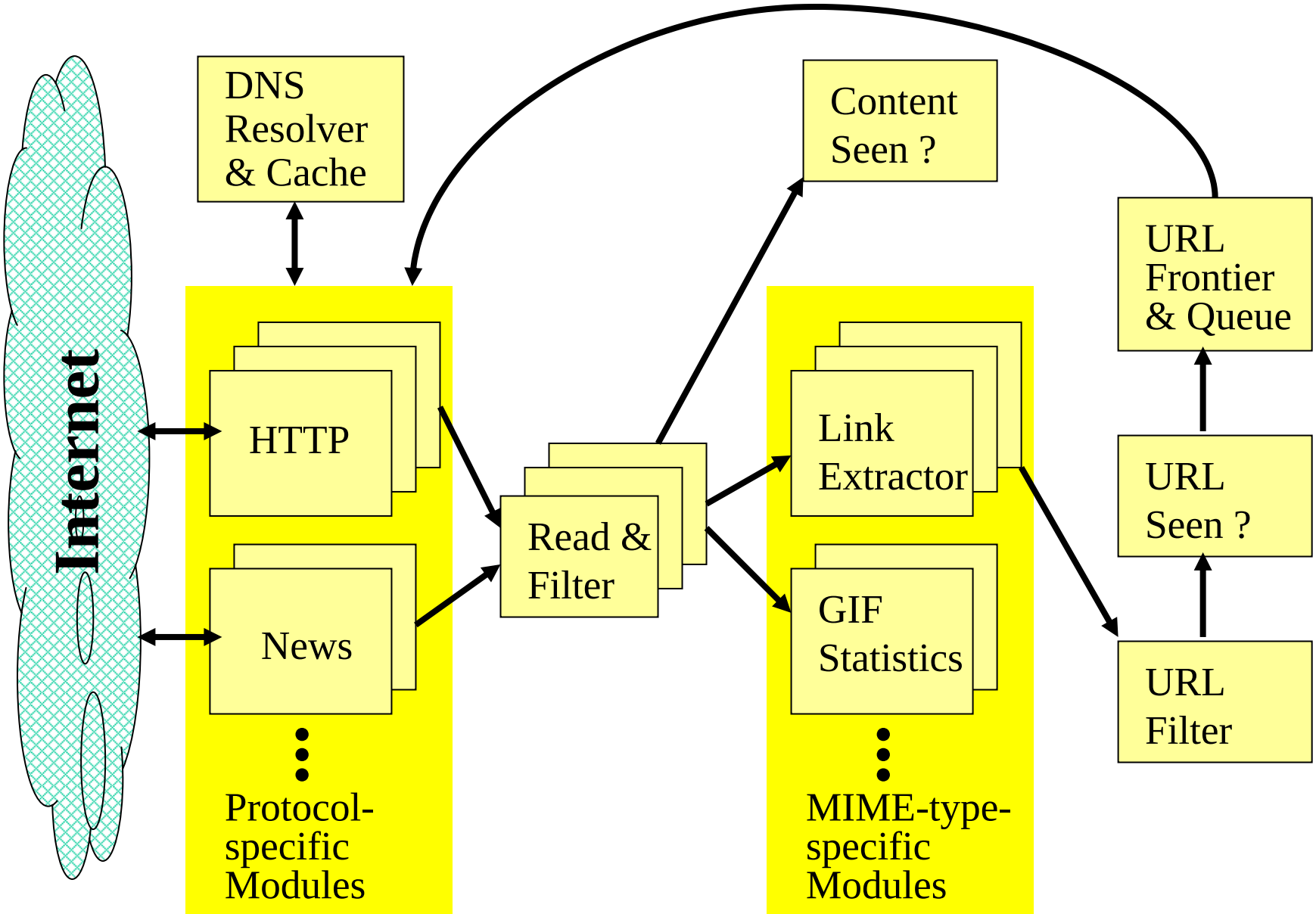
Komponenten einer Web-Suchmaschine:

- **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

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

Architektur eines skalierbaren Crawlers



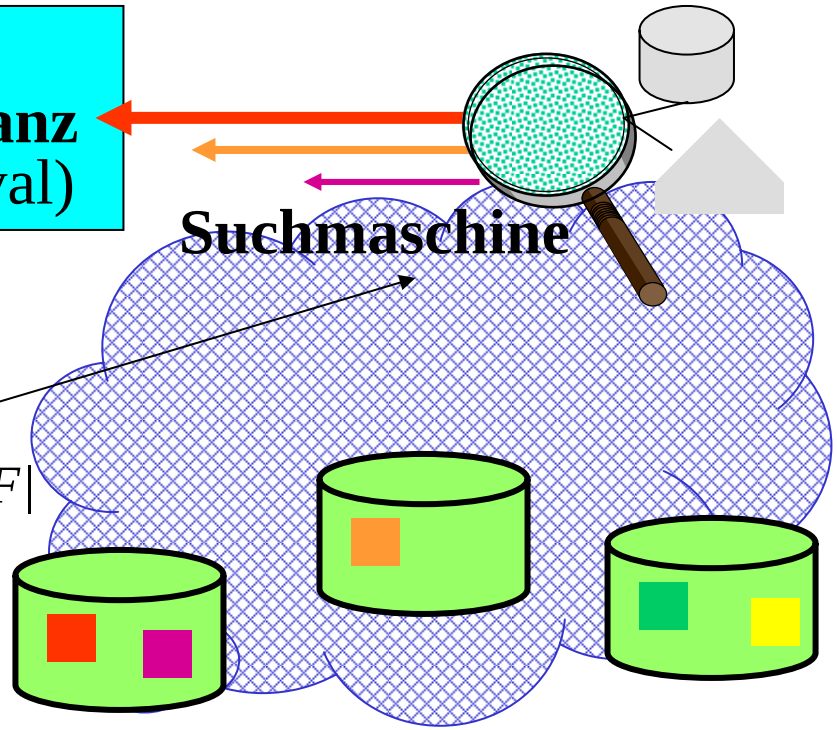
2.3 Vektorraummodell für IR mit Ranking

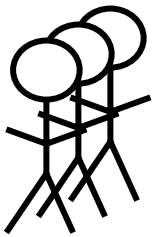
Grundprinzipien:

- **Featureraum**: Wörter in Dokumenten werden auf Terme reduziert.
- **Dokumentenmodell:** Jedes Dokument di wird als Vektor $\in [0,1]^{|F|}$ repräsentiert, wobei dij das Gewicht des j-ten Terms in di 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 analyiseren 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 Einzelterme, um Trefferkandidatenlisten zu bestimmen.

Vektorraummodell für Relevanz-Ranking

Rangliste, nach fallender **Relevanz** (Ranked Retrieval)





Query $q \in [0,1]^{|F|}$ (Menge gewichteter Such-Features)

Ähnlichkeitsfunktion

$sim(d_i, q)$

→ Scores (RSVs = Retrieval Status Values)

Dokumente sind **Feature-Vektoren**

$d_i \in [0,1]^{|F|}$

Verwendete Ähnlichkeitsfunktionen sind z.B.:

$$sim(d_i, q) := \sum_{j=1}^{|F|} d_{ij}\, q_j$$

(Skalarprodukt)

$$sim(d_i, q) := \frac{\sum_{j=1}^{|F|} d_{ij}\, q_j}{\sqrt{\sum_{j=1}^{|F|} d_{ij}^2 \sum_{j=1}^{|F|} q_j^2}}$$

(Cosinus-Maß)

Termgewichtung in Dokumenten

Betrachtet werden die folgenden Werte
(für N Dokumente und M Terme):

- tf_{ij}: Häufigkeit (term frequency) des Terms ti in Dokument dj
- df_i: Anzahl der Dokumente mit Term ti (doc. frequency)
- idf_i: N / df_i (inverse document frequency)
- cf_i: Häufigkeit von ti in allen Dokumenten (corpus frequency)

(ggf. mit separater Berücksichtigung von Termen in title u.ä.)

Grundprinzip:

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

$\rightarrow$ erster Ansatz: wij = tf_{ij} * idf_i **(tf-idf-Formel)**

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

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

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:

$$w_{ij} := \frac{tf_{ij}}{\max_k tf_{kj}} \log \frac{N}{df_i}$$

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

tf*idf-Formel

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.

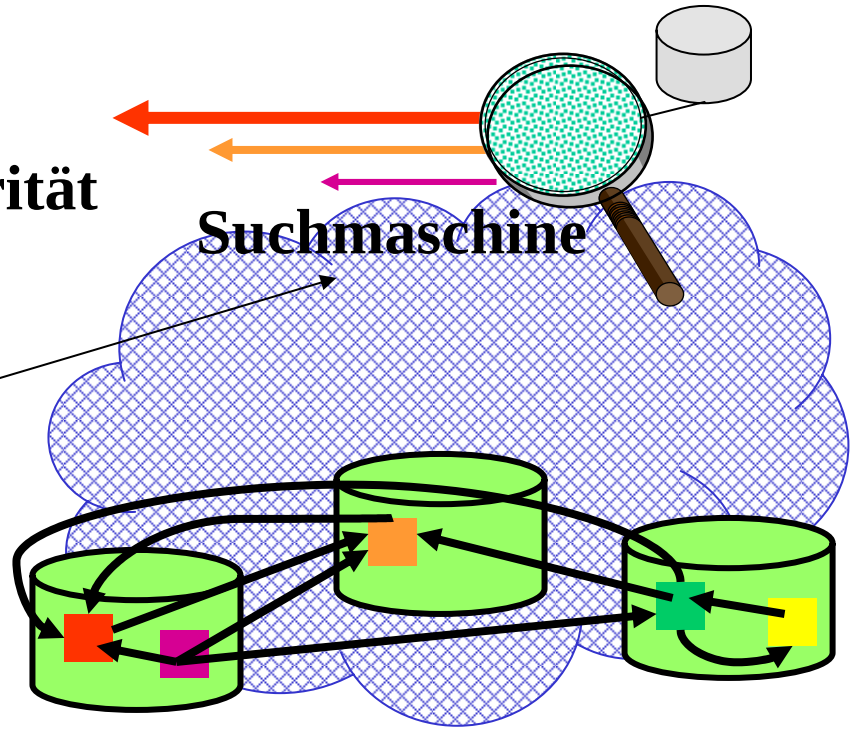
Generiere verfeinerte Query q‘:

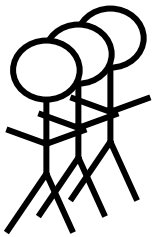
$$\vec{q}' := \alpha\vec{q} + \frac{\beta}{\left|D^+\right|} \sum_{d_j \in D^+} \vec{d}_j - \frac{\gamma}{\left|D^-\right|} \sum_{d_j \in D^-} \vec{d}_j$$

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

Linkanalyse für Autoritäts-Ranking

Rangliste nach fallender **Relevanz & Autorität**





Query $q \in [0,1]^{|F|}$ (Menge gewichteter Such-Features)

+ Berücksichtige Fanin und Fanout von Web-Seiten:

Autoritätsrang (d_i) :=

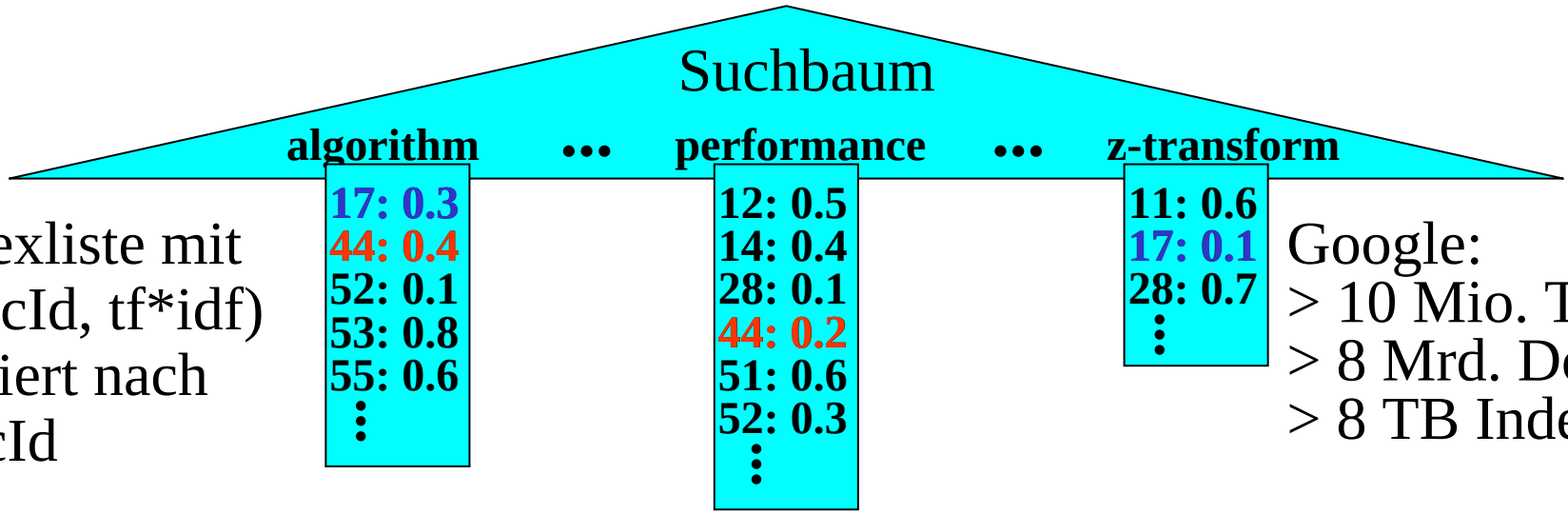
Stationäre Besuchswahrscheinlichkeit P [d_i] bei Random Walk auf dem Web

„Integration“ von Relevanz- und Autoritätsmaßen durch (ad hoc) gewichtete Linearkombination

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

2.4 Anfrageausführung mit Ranking



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

Google:
> 10 Mio. Terme
> 8 Mrd. Dok.
> 8 TB Index

Gegeben: Query $q = t_1\ t_2\ ...\ t_z$ mit z Keywords
Ähnlichkeitsfunktion score(q,d) für Dok. $d \in D$, z.B.: $\vec{q} \cdot \vec{d}$

Finde: Top-k-Resultate bzgl. score(q,d) (z.B.: $\Sigma_{i \in q}\ s_i(d)$)

Naiver QP Algorithmus:

```
candidate-docs := ∅;
for i=1 to z do {
    candidate-docs := candidate-docs ∪ index-lookup(ti) };
for each dj ∈ candidate-docs do {compute score(q,dj)};
sort candidate-docs by score(q,dj) descending;
```

Top-k-Anfragen über m-dimensionalen Datenräumen

Für ***lokale Scores*** s_i für Query q, Datenobjekt d und Dimension i

berechne ***globale Scores s*** der Form $s(q,d) = aggr\{ s_i(q,d) | i = 1..m \}$

mit ***monotoner*** Aggregationsfunktion $aggr : [0,1]^m \rightarrow [0,1]$

Beispiele: $s(q,d) = \sum_{i=1}^{m} s_i(q,d)$ $s(q,d) = max\{ s_i(q,d) | i = 1..m \}$

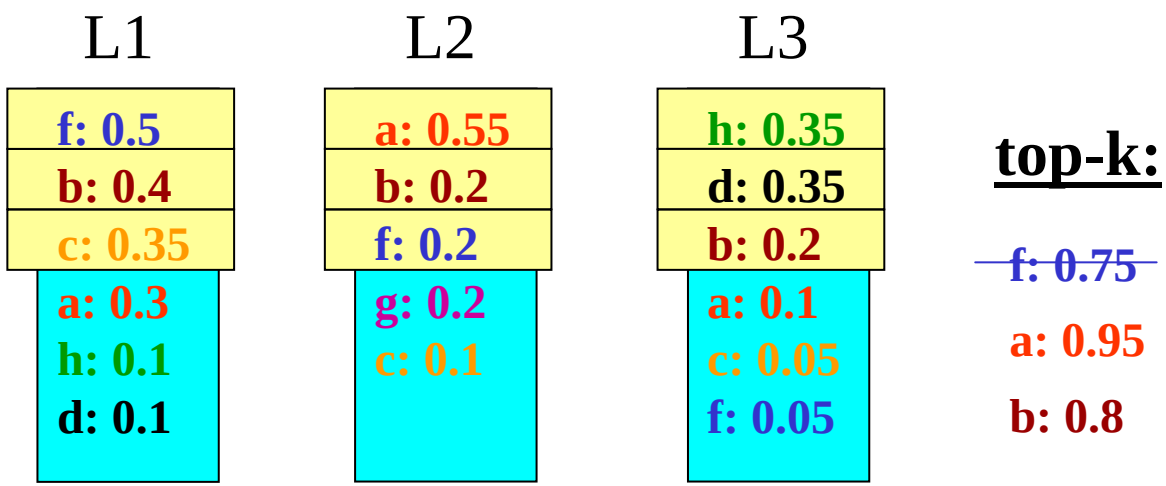
Finde die besten k Datenobjekte (Top-k) bzgl. globaler Scores:

- Traversiere m ***Indexlisten*** Li per ***sortiertem Zugriff (sorted access)*** auf Einträge (d, s_i(q,d)) in absteigender Sortierung von s_i(q,d)
- Verwalte für Kandidaten d eine Menge E(d) evaluierter Dimensionen und einen „Akkumulator“ für den Gesamt-Score
- Für Kandidaten d mit unvollständigem E(d) erwäge Nachschlagen von d in Li für alle $i \in R(d)$ per ***wahlfreiem Zugriff*** (***random access)***
- Terminiere Scans der Indexlisten, wenn genügend Kandidaten gesehen
- Falls nötig, sortiere endgültige Kandidatenliste nach globalem Score

TA: Threshold Algorithm (Fagin; Güntzer et al.)

scan all lists L_i (i=1..m) in parallel:
consider dj at position pos_i in Li;
$high_i := s_i(dj)$;
if dj $\notin$ top-k then {
look up $s_\nu(dj)$ in all lists L_ν with $\nu \neq i$; // random access
compute $s(dj) := aggr \{s_\nu(dj) \mid \nu=1..m\}$;
if s(dj) > min score among top-k then
add dj to top-k and remove min-score d from top-k; };
if min score among top-k $\geq$ aggr $\{high_\nu \mid \nu=1..m\}$ then exit;

m=3
aggr: sum
k=2



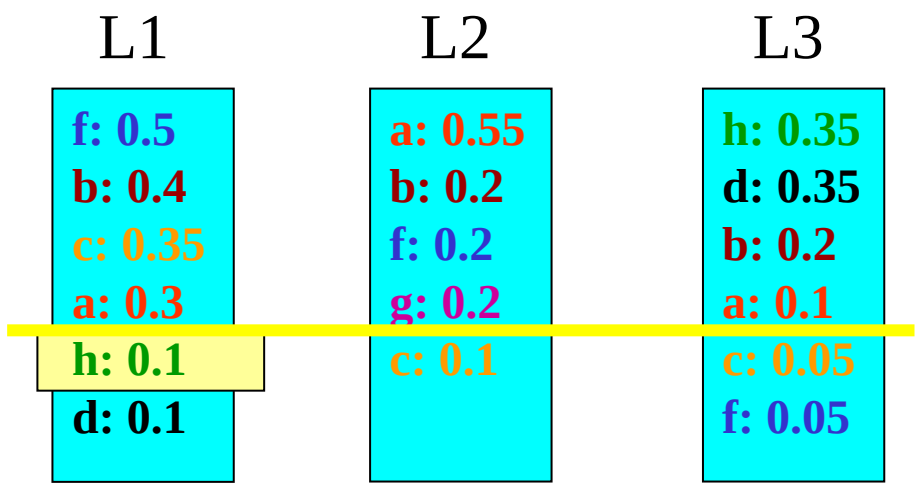
top-k:

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

TA-Sorted / NRA (Fagin; Güntzer et al.)

scan index lists in parallel:
consider dj at position pos_i in Li;
E(dj) := E(dj) ∪ {i}; $high_i$:= si(q,dj);
bestscore(dj) := aggr{x1, ..., xm)
with xi := si(q,dj) for i∈E(dj), $high_i$ for i ∉E(dj);
worstscore(dj) := aggr{x1, ..., xm)
with xi := si(q,dj) for i∈E(dj), 0 for i ∉E(dj);
top-k := k docs with largest worstscore;
if min worstscore among top-k ≥ bestscore{d | d not in top-k}
then exit;

m=3
aggr: sum
k=2



top-k:

a: 0.95
b: 0.8

candidates:

~~**f: 0.7 + ? ≤ 0.7 + 0.1**~~
~~**h: 0.45 + ? ≤ 0.45 + 0.2**~~
~~**c: 0.35 + ? ≤ 0.35 + 0.3**~~
~~**d: 0.35 + ? ≤ 0.35 + 0.3**~~
~~**g: 0.2 + ? ≤ 0.2 + 0.4**~~

Datenintensive Anwendungen mit Top-k-Queries

Top-k-Resultate von Anfragen mit Ranking über

- ***Multimedia-Daten***: Aggr. über Features wie Farbe, Kontur, Textur, etc.
- ***Produktkatalogen***: Aggr. über Ähnlichkeits-Scores für
  numerische Attribute wie Jahr, Preis, Bewertung, etc. und
  kategoriale Attribute wie Lage (von Häusern), Schnittstellen (von PCs)
- ***Textdokumente***: Aggr. über Termgewichte
- ***Web-Dokumente***: Aggr. über (Inhalts-) Relevanz, Autorität, Aktualität
- ***Intranet-Dokumente***: Aggr. über Features wie
  Text, Titel, Ankertext, Autorität, Aktualität, URL-Länge, URL-Tiefe,
  URL-Typ (z.B. mit „index.html“ oder „~“ vs. „?“)
- ***Metasuchmaschinen:*** Aggr. über Ranglisten
  verschiedener Suchmaschinen
- ***Verteilte Deep-Web-Quellen***: Aggr. über Eigenschaften von Objekten
  auf dynamischen Webseiten wie z.B. Restaurantbewertungen,
  Restaurantpreisen, Entfernung lt. streetfinder.com u.ä.
- ***Peer-to-Peer-basierte Empfehlungen***

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×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×1-Vektoren heißt **Orthonormalbasis**, wenn für alle $x, y \in S$ gilt:

$$\|x\|_2 := \sqrt{\sum_{i=1}^{n} X_i^2} = 1 = \|y\|_2 \quad und \quad x \cdot y = 0$$

Eigenwerte und Eigenvektoren

Seien A eine reellwertige n×n-Matrix, x ein reellwertiger n×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(λ) von A: $f(\lambda) = |A - \lambda I| = 0$

mit der Determinantenentwicklung nach der i-ten Zeile:

$$|A| = \sum_{j=1}^{n} (-1)^{i+j} a_{ij} \left|A^{(ij)}\right|$$

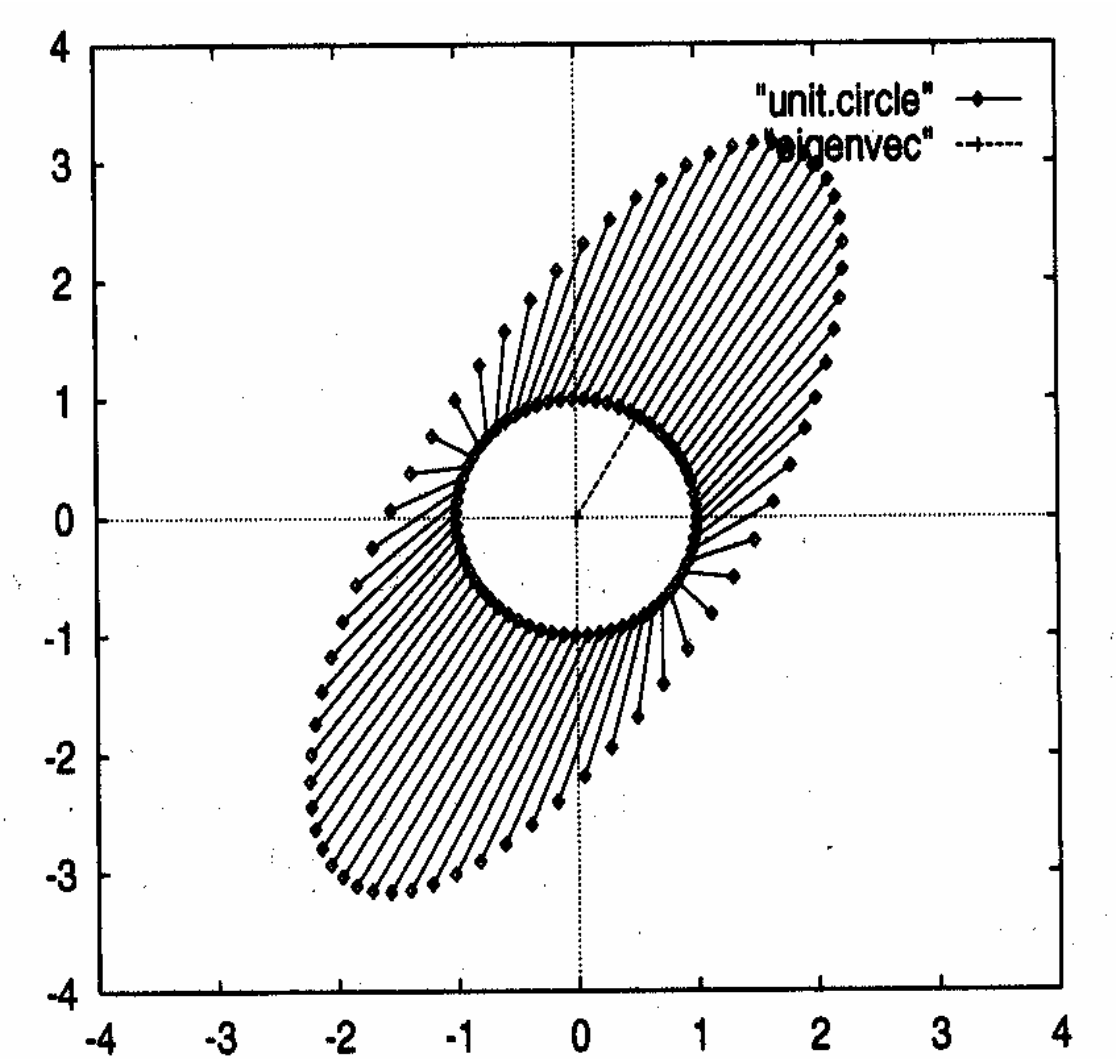
wobei man die Matrix $A^{(ij)}$ aus A durch Streichung der i. Zeile und der j. Spalte erhält

Die reellwertige n×n-Matrix A heißt **symmetrisch**, wenn $a_{ij}=a_{ji}$ für alle i, j. A heißt **positiv definit**, wenn für alle n×1-Vektoren $x \neq \mathcal{O}$ 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.

Illustration von Eigenvektoren

Matrix $A = \begin{pmatrix} 2 & 1 \\ 1 & 3 \end{pmatrix}$

beschreibt
affine Abb. $x \mapsto Ax$



Eigenvektor
x1 = $(0.52 \;\; 0.85)^T$
zum Eigenwert $\lambda 1 = 3.62$

Eigenvektor
x2 = $(0.85 \;\; -0.52)^T$
zum Eigenwert $\lambda 2 = 1.38$

Spektralsatz der Linearen Algebra

Spektralsatz

(Hauptachsentransformation, Principal Component Analysis, PCA):

Sei A eine symmetrische n×n-Matrix mit Eigenwerten λ1, ..., λn
und Eigenvektoren x1, ..., xn, 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 λ1, ..., λn
und Q aus den Spaltenvektoren x1, ..., xn besteht.

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×n-Term-Dokument-Ähnlichkeitsmatrix A,

gesucht: möglichst gute – ähnlichkeitsbewahrende – Abbildung der Spaltenvektoren von A in einen k-dimensionalen Vektorraum (k << m) für gegebenes k

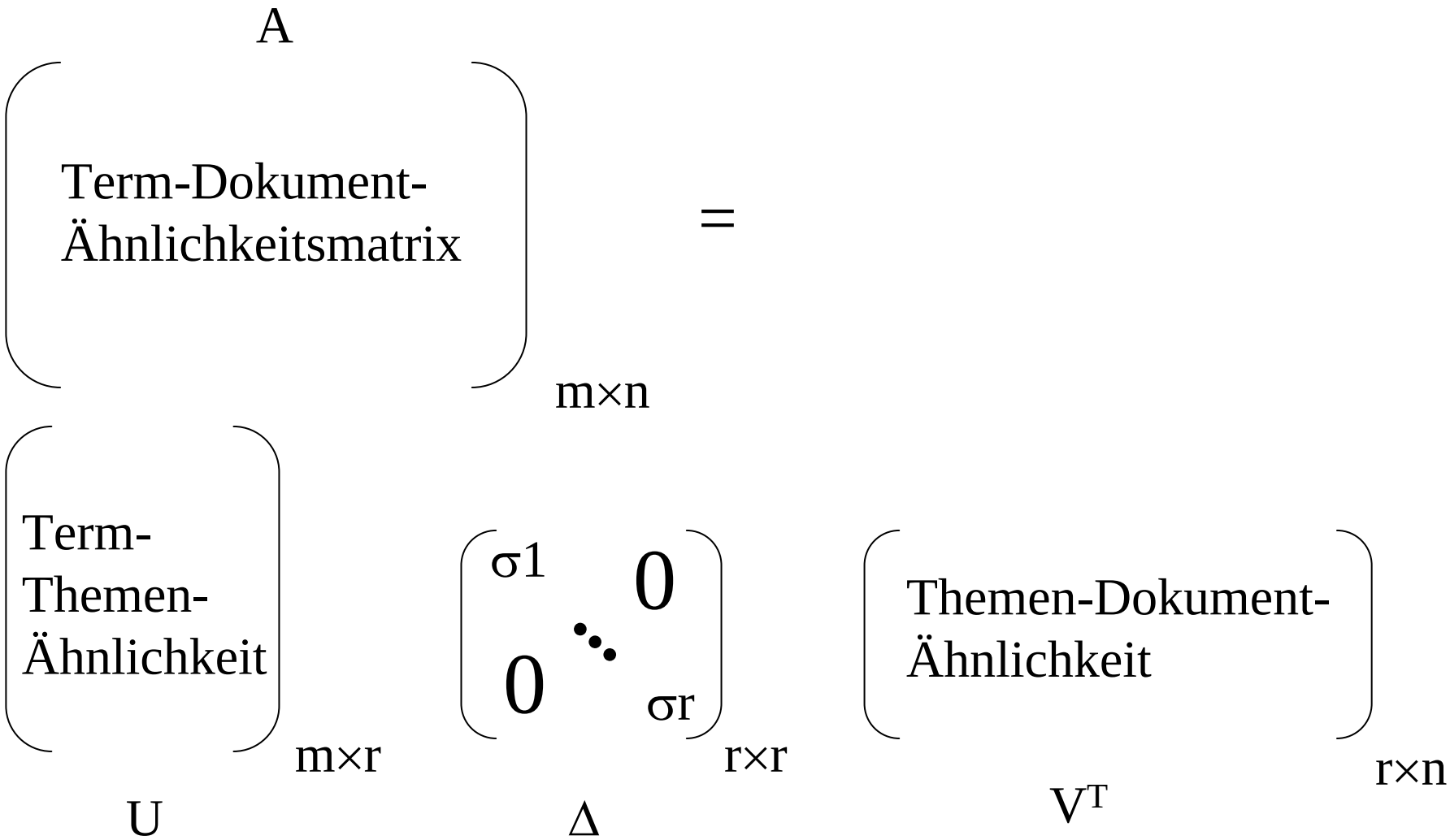
Exkurs: Singulärwertdekomposition (SVD)

Satz:
Jede reellwertige m×n-Matrix A mit Rang r kann zerlegt werden in die Form $\mathbf{A = U \times \Delta \times V^T}$
mit einer **m×r-Matrix U** mit orthonormalen Spaltenvektoren,
einer **r×r-Diagonalmatrix Δ** und
einer **n×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 $A^T \times A$.

Schematische Darstellung der SVD



Exkurs: SVD als Regressionsverfahren

Satz:

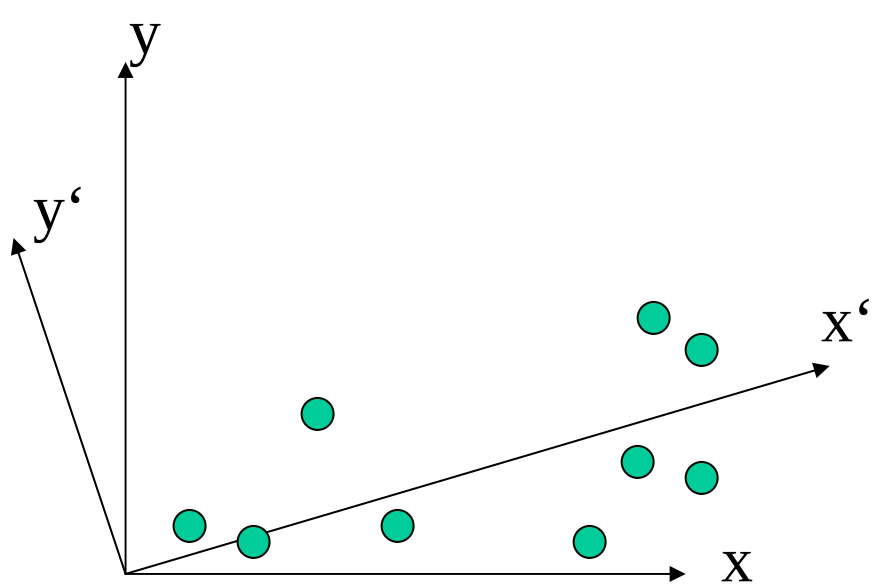
Sei A eine m×n-Matrix mit Rang r und sei Ak = $U_k \times \Delta_k \times V_k^T$, wobei die k×k-Diagonalmatrix Δ_k die k größten Singulärwerte von A enthält und die m×k-Matrix U_k sowie die n×k-Matrix V_k aus den zugehörigen Eigenvektoren der Singulärwertdekomposition von A bestehen.

Unter allen m×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



Anwendung der SVD auf das Vektorraummodell

A ist die m×n-Term-Dokument-Ähnlichkeitsmatrix. Dann sind:

- U bzw. U_k die m×r- bzw. m×k-Term-Themen-Ähnlichkeitsmatrix,
- V bzw. V_k die n×r- bzw. n×k-Dokument-Themen-Ähnlichkeitsmatrix
- $A \times A^T$ bzw. $A_k \times A_k^T$ die m×m-Term-Term-Ähnlichkeitsmatrix,
- $A^T \times A$ bzw. $A_k^T \times A_k$ die n×n-Dokument-Dokument-Ähnlichkeitsmatrix

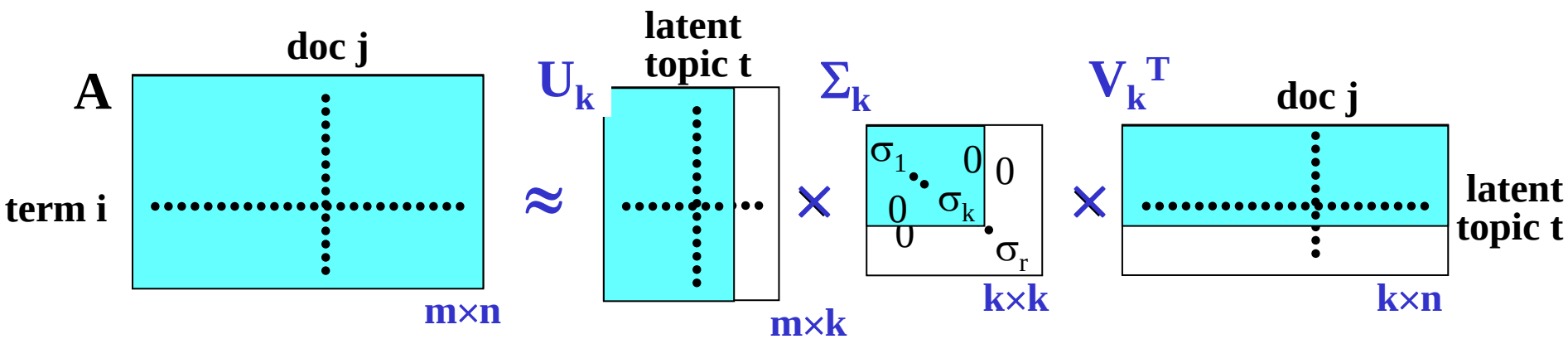


Abbildung von m×1-Vektoren in Themenraum: $d_j \mapsto U_k^T \times d_j =: d_j'$

$q \mapsto U_k^T \times q =: q'$

Skalarprodukt-Ähnlichkeit im Themenraum: $d_j{}^{\prime T} \times q' = ((\Delta_k V_k^T)_{*j})^T \times q'$

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×1-Spaltenvektor) im Termvektorraum wird in die Anfrage **$q' = U_k^T \times q$** (ein k×1-Spaltenvektor) transformiert und dann im Themenvektorraum (also V_k) ausgewertet (z.B. mittels Skalarproduktmaß $V_k^T \times q'$ oder Cosinusmaß)
- Ein **neues Dokument d** (ein m×1-Spaltenvektor) wird in **$d' = U_k^T \times d$** (ein k ×1-Spaltenvektor) transformiert und als neue Spalte an den „Index“ V_k^T angefügt („folding-in“)

Beispiel 1 für Latent Semantic Indexing

m=5 (Bush, Schröder, Korea, Klose, Völler), n=7

$$A = \begin{pmatrix} 1&2&1&5&0&0&0 \\ 1&2&1&5&0&0&0 \\ 1&2&1&5&0&0&0 \\ 0&0&0&0&2&3&1 \\ 0&0&0&0&2&3&1 \end{pmatrix} = \underbrace{\begin{pmatrix} 0.58 & 0.00 \\ 0.58 & 0.00 \\ 0.58 & 0.00 \\ 0.00 & 0.71 \\ 0.00 & 0.71 \end{pmatrix}}_{U} \times \underbrace{\begin{pmatrix} 9.64 & 0.00 \\ 0.00 & 5.29 \end{pmatrix}}_{\Delta} \times \underbrace{\begin{pmatrix} 0.18 & 0.36 & 0.18 & 0.90 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.53 & 0.80 & 0.27 \end{pmatrix}}_{V^T}$$

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.

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

$$A = \begin{pmatrix} 0.5774 & 0.0000 & 0.0000 & 0.4082 & 0.0000 \\ 0.5774 & 0.0000 & 1.0000 & 0.4082 & 0.7071 \\ 0.5774 & 0.0000 & 0.0000 & 0.4082 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.4082 & 0.0000 \\ 0.0000 & 1.0000 & 0.0000 & 0.4082 & 0.7071 \\ 0.0000 & 0.0000 & 0.0000 & 0.4082 & 0.0000 \end{pmatrix}$$

Beispiel 2 für Latent Semantic Indexing (2)

$$A = \begin{pmatrix} 0.2670 & -0.2567 & 0.5308 & -0.2847 \\ 0.7479 & -0.3981 & -0.5249 & 0.0816 \\ 0.2670 & -0.2567 & 0.5308 & -0.2847 \\ 0.1182 & -0.0127 & 0.2774 & 0.6394 \\ 0.5198 & 0.8423 & 0.0838 & -0.1158 \\ 0.1182 & -0.0127 & 0.2774 & 0.6394 \end{pmatrix} \quad \mathrm{U}$$

$$\times \begin{pmatrix} 1.6950 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 1.1158 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.8403 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.4195 \end{pmatrix} \quad \Delta$$

$$\times \begin{pmatrix} 0.4366 & 0.3067 & 0.4412 & 0.4909 & 0.5288 \\ -0.4717 & 0.7549 & -0.3568 & -0.0346 & 0.2815 \\ 0.3688 & 0.0998 & -0.6247 & 0.5711 & -0.3712 \\ -0.6715 & -0.2760 & 0.1945 & 0.6571 & -0.0577 \end{pmatrix} \quad \mathrm{V}^{\mathrm{T}}$$

Beispiel 2 für Latent Semantic Indexing (3)

$$A_3 = \begin{pmatrix} 0.4971 & -0.0330 & 0.0232 & 0.4867 & -0.0069 \\ 0.6003 & 0.0094 & 0.9933 & 0.3858 & 0.7091 \\ 0.4971 & -0.0330 & 0.0232 & 0.4867 & -0.0069 \\ 0.1801 & 0.0740 & -0.0522 & 0.2320 & 0.0155 \\ -0.0326 & 0.9866 & 0.0094 & 0.4402 & 0.7043 \\ 0.1801 & 0.0740 & -0.0522 & 0.2320 & 0.0155 \end{pmatrix} = U_3 \times \Delta_3 \times V_3^T$$

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:

$sim(q, d1) = Vk_{*1}^T \times q' \approx 0.86$ $\quad sim(q, d2) = Vk_{*2}^T \times q' \approx -0.12$

$sim(q, d3) = Vk_{*3}^T \times q' \approx -0.24$ $\quad$ 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 Vk^T anhängen

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.

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

Kapitel 3: Automatische Klassifikation von Dokumenten

Automatische Klassifikation von Dokumenten

Ziel:

Organisation von Dokumenten in (hierarchischen) Taxonomien mit möglichst geringem intellektuellem Aufwand

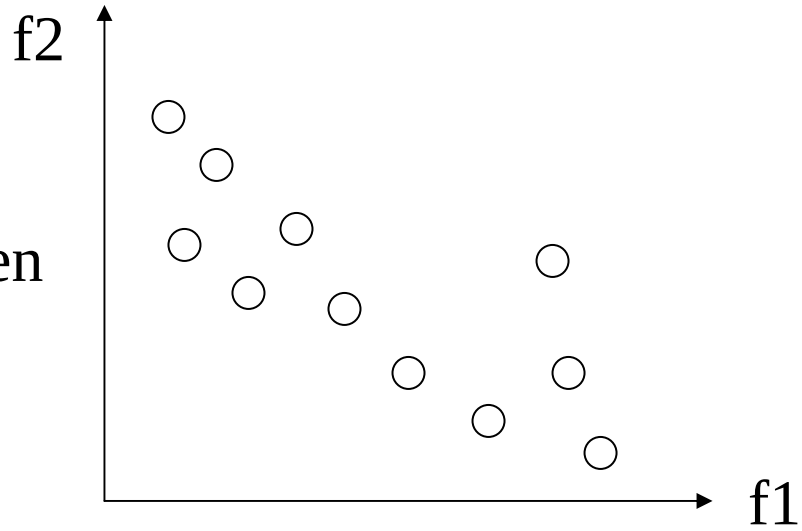
Techniken:

- **Klassifikation mit Training (Supervised Learning)**
  - kNN-Verfahren
  - Rocchio-Verfahren
  - Naives Bayes-Verfahren
  - Support Vector Machines (SVM)
  - ...
- **Klassifikation ohne Training (Unsupervised Learning)**
  - Verfahren der Clusteranalyse

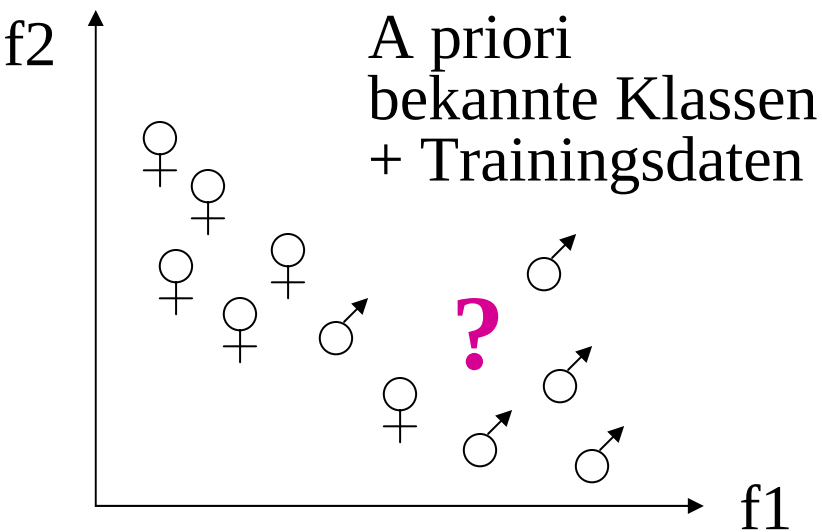
Orthogonal dazu gibt es flache vs. hierarchische Verfahren

Klassifikationsproblem (Kategorisierung)

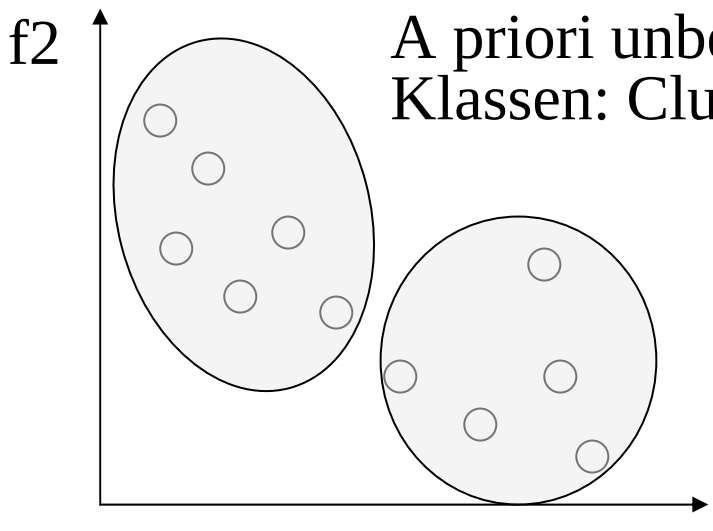
gegeben:
Featurevektoren



bestimme
Klassenzugehörigkeit
von Feature-Vektoren



A priori
bekannte Klassen
+ Trainingsdaten



A priori unbekannte
Klassen: Clustering

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 Themenbaum

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

Genauigkeit (acccuracy) = $\frac{a+d}{a+b+c+d}$

Präzision (precision) = $\frac{a}{a+b}$ **Ausbeute** (recall) = $\frac{a}{a+c}$

Für mehrstellige Klassifikation bzgl. Klassen C1, ..., Ck:
- Makrodurchschnitt über k Klassen oder
- Mikrodurchschnitt über k Klassen

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

$$f(\vec{d}, C_j) = \sum_{\vec{v} \in kNN(\vec{d})} sim(\vec{d}, \vec{v}) * \begin{cases} 1 & falls\ \vec{v} \in C_j \\ 0 & sonst \end{cases}$$

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.

Klassifikationsverfahren von Rocchio

Schritt 1:

Repräsentiere die Trainingsdokumente einer Klasse Cj
- 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. α=16, β=4)

Schritt 2:

Ordne ein neues Dokument $\vec{d}$ derjenigen Klasse Cj zu, für die Cosinus-Ähnlichkeit $cos(\vec{c}_j, \vec{d})$ maximal ist.

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 $P[\cup_i Ai] = \sum_i P[Ai]$ für abzählbar viele, paarweise disjunkte Ai

Eigenschaften von P:

$P[A] + P[\neg A] = 1$

$P[A \cup B] = P[A] + P[B] - P[A \cap B]$

$P[\varnothing] = 0$

$P[\Omega] = 1$

Zufallsvariable

Eine **Zufallsvariable** X über einem W.raum (Ω, E, P) ist eine Funktion X: $\Omega \rightarrow$ M mit M$\subseteq$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.

Momente

Für eine **diskrete Zufallsvariable** X mit Dichtefunktion f_X sind

$E[X] = \sum_{k \in M} k \, f_X(k)$ der ***Erwartungswert*** von X

$E[X^i] = \sum_{k \in M} k^i \, f_X(k)$ das ***i. Moment*** von X

$V[X] = E[(X - E[X])^2] = E[X^2] - E[X]^2$ die ***Varianz*** von X

Für eine **kontinuierliche Zufallsvariable** X mit Dichtefunktion f_X sind

$E[X] = \int_{-\infty}^{+\infty} x \, f_X(x) \, dx$ der ***Erwartungswert*** von X

$E[X^i] = \int_{-\infty}^{+\infty} x^i \, f_X(x) \, dx$ das ***i. Moment*** von X

$V[X] = E[(X - E[X])^2] = E[X^2] - E[X]^2$ die ***Varianz*** von X

Erwartungswerte sind additiv, $E[X + Y] = E[X] + E[Y]$
Verteilungen nicht

Wichtige diskrete Verteilungen

- **Gleichverteilung** über {1, 2, ..., m}:

$$P[X = k] = f_X(k) = \frac{1}{m} \quad for\, 1 \le k \le m$$

- **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):

$$P[X = k] = f_X(k) = (1-p)^k \; p$$

- **2-Poisson-Mix** (mit $a_1+a_2=1$):

$$P[X = k] = f_X(k) = a_1 e^{-\lambda_1} \frac{{\lambda_1}^k}{k!} + a_2 e^{-\lambda_2} \frac{{\lambda_2}^k}{k!}$$

Kontinuierliche Verteilungen

- **Gleichverteilung** über dem Intervall [a,b]

$$f_X(x) = \frac{1}{b-a} \quad \text{für } a \le x \le b \ (0 \text{ sonst})$$

- **Exponentialverteilung** (z.B. Zeit bis zum nächsten Ereignis eines Poisson-Prozesses) mit Rate $\lambda = \lim_{\Delta t \to 0}$ (# Ereignisse in Δt) / Δt :

$$f_X(x) = \lambda e^{-\lambda x} \quad \text{für } x \ge 0 \ (0 \text{ sonst})$$

- **Hyperexponential-Verteilung**: $f_X(x) = p\lambda_1 e^{-\lambda_1 x} + (1-p)\lambda_2 e^{-\lambda_2 x}$

- **Pareto-Verteilung**: $f_X(x) = \frac{a}{b}\left(\frac{b}{x}\right)^{a+1}$ für $x > b, 0$ sonst

Beispiel einer „Heavy-tailed“-Verteilung mit $f_X(x) \to \frac{c}{x^{\alpha+1}}$

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.

Dann ist $Y := \frac{X-\mu}{\sigma}$

normalverteilt mit Erwartungswert 0 und Varianz 1.

Zentraler Grenzwertsatz

Satz:

Seien X1, ..., Xn unabhängig und identisch verteilte Zufallsvariablen mit Erwartungswert μ und Varianz σ^2.
Die Verteilungsfunktion Fn der Zufallsvariablen Zn := X1 + ... + Xn konvergiert gegen eine Normalverteilung $N(n\mu, n\sigma^2)$
mit Erwartungswert $n\mu$ und Varianz $n\sigma^2$:

$$lim_{n\to\infty} P[\, a \le \frac{Z_n - n\mu}{\sqrt{n\sigma}} \le b \,] = \Phi(b) - \Phi(a)$$

Korollar:

$$\overline{X} := \frac{1}{n} \sum_{i=1}^{n} X_i$$

konvergiert gegen eine Normalverteilung $N(\mu, \sigma^2/n)$

mit Erwartungswert μ und Varianz σ^2/n .

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: $$P[A \mid B] = \frac{P[A \cap B]}{P[B]}$$

Satz von der totalen Wahrscheinlichkeit:
Für eine Partitionierung von Ω in Ereignisse B1, ..., Bn gilt:

$$P[A] = \sum_{i=1}^{n} P[A \mid Bi]\,P[Bi]$$

Satz von Bayes: $$P[A \mid B] = \frac{P[B \mid A]\,P[A]}{P[B]}$$

A-Posteriori-W. von A

3.3 Naives Bayes-Verfahren mit binären Features Xi

Schätze: $P[\, d \in c_k \,|\, d\ hat\ \vec{X}\,] = \dfrac{P[\, d\ hat\ \vec{X} \,|\, d \in c_k\,]\, P[\, d \in c_k\,]}{P[\, d\ hat\ \vec{X}\,]}$

$$\sim P[\, X \,|\, d \in c_k\,]\, P[\, d \in c_k\,]$$

$$= \Pi_{i=1}^{m} P[\, X_i \,|\, d \in c_k\,]\, P[\, d \in c_k\,]$$

bei Featureunabhängigkeit bzw. Linked Dependence:

$$\frac{P[\, X \,|\, d \in c_k\,]}{P[\, X \,|\, d \notin c_k\,]} = \Pi_i \frac{P[\, X_i \,|\, d \in c_k\,]}{P[\, X_i \,|\, d \notin c_k\,]}$$

$$= \Pi_{i=1}^{m} p_{ik}^{\,Xi} (1 - p_{ik})^{1-Xi}\, p_k$$

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[...]
weitere Vereinfachung möglich

Naives Bayes-Verfahren mit Bag-of-Words-Modell

Schätze: $P[\, d \in c_k \,|\, d \text{ hat } \vec{f}\,] \sim P[\, \vec{f} \,|\, d \in c_k\,]\, P[\, d \in c_k\,]$

mit Termhäufigkeitsvektor $\vec{f}$

$= \Pi_{i=1}^{m} P[\, f_i \,|\, d \in c_k\,]\, P[\, d \in c_k\,]$ bei Featureunabhängigkeit

$= \Pi_{i=1}^{m} \binom{length(d)}{f_i} p_{ik}^{\,f_i} (1 - p_{ik})^{length(d) - f_i}\; p_k$

mit Binomialverteilung für jedes Feature

bzw. besser: $= \binom{length(d)}{f_1\, f_2 \ldots f_m} p_{1k}^{\,f_1}\, p_{2k}^{\,f_2} \ldots p_{mk}^{\,f_m}\; p_k$

mit Multinomialverteilung der Featurevektoren und $\sum_{i=1}^{m} f_i = length(d)$

mit $\binom{n}{k_1\, k_2 \ldots k_m} := \frac{n!}{k_1!\, k_2! \ldots k_m!}$

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

$\Rightarrow$ p1=2/6, p2=2/6, p3=2/6

	Gruppe	Homomorphismus	Vektor	Integral	Limes	Varianz	Wahrscheinlichkeit	Würfel
	f1	f2	f3	f4	f5	f6	f7	f8
d1:	3	2	0	0	0	0	0	1
d2:	1	2	3	0	0	0	0	0
d3:	0	0	0	3	3	0	0	0
d4:	0	0	1	2	2	0	1	0
d5:	0	0	0	1	1	2	2	0
d6:	1	0	1	0	0	0	2	2

	Algebra	Analysis	Stochastik
	k=1	k=2	k=3
p1k	4/12	0	1/12
p2k	4/12	0	0
p3k	3/12	1/12	1/12
p4k	0	5/12	1/12
p5k	0	5/12	1/12
p6k	0	0	2/12
p7k	0	1/12	4/12
p8k	1/12	0	2/12

Beispiel für das naive Bayes-Verfahren (2)

Klassifikation von d7: (0 0 1 2 0 0 3 0)

$$P[\,\vec{f}\,|d \in c_k\,]\,P[\,d \in c_k\,] = \binom{length(d)}{f_1\ f_2 \cdots f_m}\, p_{1k}^{\,f_1}\ p_{2k}^{\,f_2} \ldots p_{mk}^{\,f_m}\quad p_k$$

für k=1 (Algebra): $= \binom{6}{1\ 2\ 3} \left(\frac{3}{12}\right)^1 0^2\, 0^3 \quad \frac{2}{6} \qquad = 0$

für k=2 (Analysis): $= \binom{6}{1\ 2\ 3} \left(\frac{1}{12}\right)^1 \left(\frac{5}{12}\right)^2 \left(\frac{1}{12}\right)^3 \quad \frac{2}{6} \qquad = 20 * \frac{25}{12^6}$

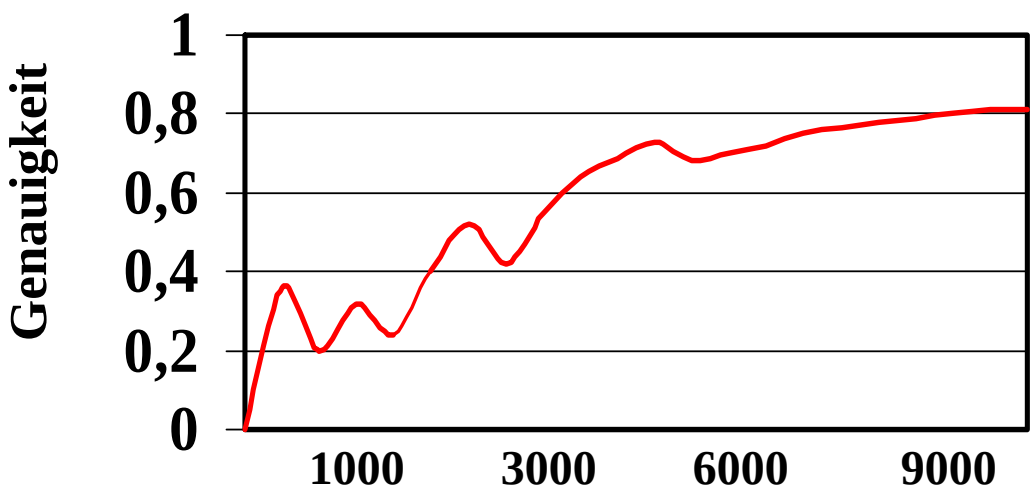
für k=3 (Stochastik): $= \binom{6}{1\ 2\ 3} \left(\frac{1}{12}\right)^1 \left(\frac{1}{12}\right)^2 \left(\frac{4}{12}\right)^3 \quad \frac{2}{6} \qquad = 20 * \frac{64}{12^6}$

Resultat: Ordne d7 der Klasse C3 (Stochastik) zu

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



Typisches Verhalten

max. Genauigkeit liegt je nach Kategorie zwischen 50 und 90 Prozent

Verbesserung des naiven Bayes-Verfahrens

1) geglättete Schätzung der p_{ik} durh 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

Erweiterung um Semisupervised Learning

Motivation:

- Klassifikator nur so gut wie seine Trainingsdaten
- Trainingsdaten teuer wegen intellektueller Klassifikation
- Trainingsdaten sind im Featureraum 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 „autombobile“, aber nicht „car“.

Simples 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

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.

$\rightarrow$ Betrachtung statistischer bzw. informationstheoretischer Maße zur Selektion geeigneter Features

Feature-Selektion auf der Basis der Mutual Information (MI)

Mutual Information (Relative Entropie, Kullback-Leibler-Distanz):
Zur Entscheidung für Klasse cj wähle diejenigen binären Features Xi (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 c1, ..., ck:

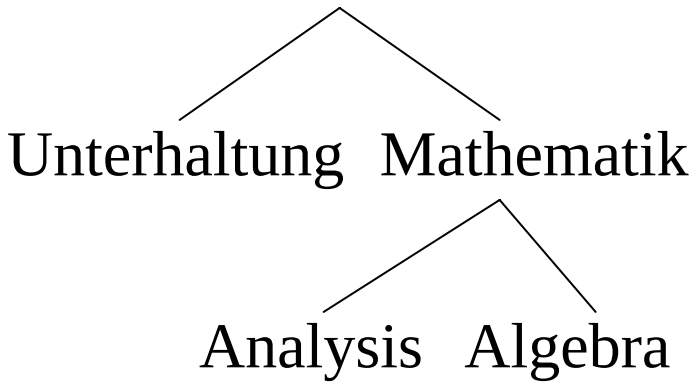
$$MI(X_i) = \sum_{j=1}^{k} P[c_j]\, MI(X_i, c_j)$$

Berechnung in Zeit O(n)+O(mk)
für n Trainingsdokumente,
m Terme und k Klassen

Beispiel für Feature-Selektion

	Film	Hit	Chart	Theorem	Limes	Integral	Gruppe	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	1	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

Beispielrechnung für Feature-Selektion auf der Basis des MI-Maßes

Unterhaltung (d1-d4) vs. Mathematik (d5-d12):

MI(Film) = 2/12 log [2/12 / (2/12 * 1/3)] + 0 log 0 +
2/12 log [2/12 / (2/12 * 1/3)] + 8/12 log[8/12 / (10/12 * 2/3)]

MI(Chart) = 3/12 log [3/12 / (4/12 * 1/3)] + 1/12 log [1/12 / (4/12 * 2/3)] +
1/12 log [1/12 / (8/12 * 1/3)] + 7/12 log[7/12 / (8/12 * 2/3)]

MI(Theorem) = 0 log 0 + 6/12 log [6/12 / (6/12 * 2/3)] +
4/12 log [4/12 / (6/12 * 1/3)] + 2/12 log[2/12 / (6/12 * 2/3)]

Analysis (d5-d8) vs. Algebra (d9-d12):

MI(Film) = 0 log 0 + 0 log 0 +
4/8 log [4/8 / (8/8 * 1/2)] + 4/8 log[4/8 / (8/8 * 1/2)]

MI(Theorem) = 3/8 log [3/8 / (6/8 * 1/2)] + 3/8 log [3/8 / (6/8 * 1/2)] +
1/8 log [1/8 / (2/8 * 1/2)] + 1/8 log[1/8 / (2/8 * 1/2)]

MI(Vektor) = 0 log 0 + 3/8 log [3/8 / (3/8 * 1/2)] +
4/8 log [4/8 / (5/8 * 1/2)] + 1/8 log[1/8 / (5/8 * 1/2)]

Kapitel 4: Linkanalyse für Autoritäts-Ranking

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)

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)∈E, 0 sonst

Idee: $$r(q) \approx k \sum_{(p,q)\in G} r(p)/outdegree(p)$$

Def.: $$r(q) = \varepsilon/n + (1-\varepsilon) \sum_{(p,q)\in G} r(p)/outdegree(p)$$ mit $0 < \varepsilon \leq 0.25$

Satz: Mit A‘$_{ij}$ = 1/outdegree(j) falls (j,i)∈E, 0 sonst, gilt:

$$\vec{r} = \frac{\vec{\varepsilon}}{n} + (1-\varepsilon)A'\vec{r} \quad \Leftrightarrow \vec{r} = \left(\frac{\vec{\varepsilon}}{n}\vec{1}^T + (1-\varepsilon)A'\right)\vec{r}$$

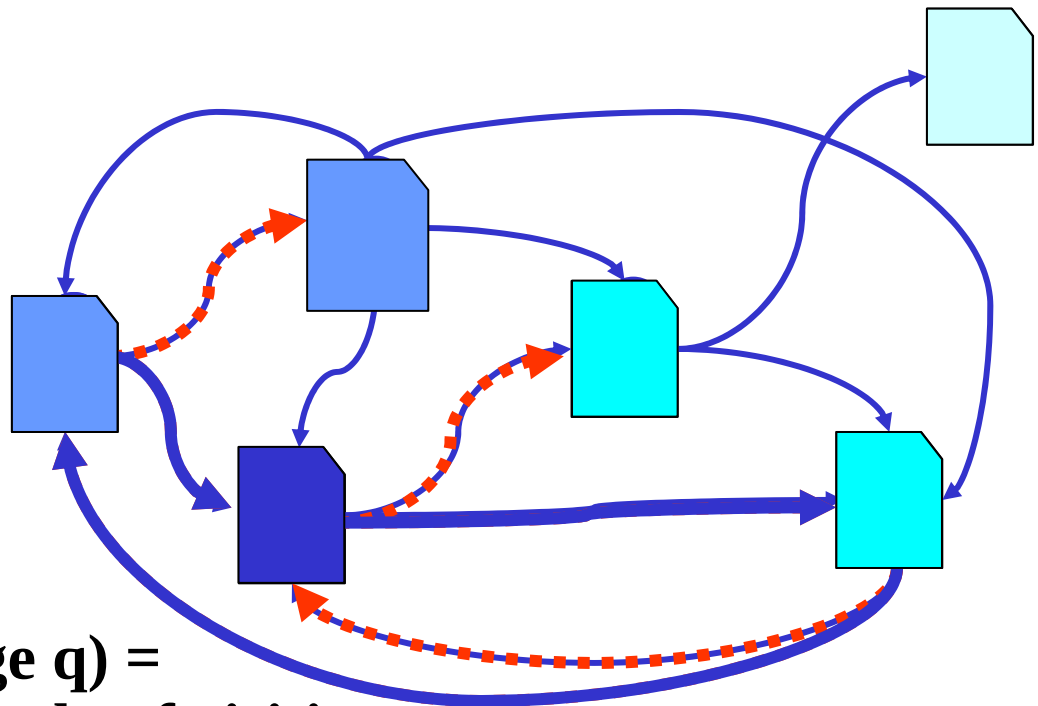
d.h. **r ist Eigenvektor** einer modifizierten Transitionsmatrix

Iterative Berechnung von r(q):
- Initialisierung mit r(q) := 1/n
- Verbesserung durch Auswerten der rekursiven Definitionsgleichung

konvergiert typischerweise mit ca. 100 Iterationen

PageRank als Random Walk

$$PR(q) = \varepsilon \cdot j(q) + (1-\varepsilon) \cdot \sum_{p \in IN(q)} PR(p) \cdot t(p,q)$$



Authority (page q) =
stationary prob. of visiting q

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 $t_1, ..., t_{n+1}$ aus dem Parameterraum und
für beliebige $x_1, ..., x_{n+1}$ aus dem Zustandsraum gilt:

$$P[\, X(t_{n+1}) = x_{n+1} \mid X(t_1) = x_1 \wedge X(t_2) = x_2 \wedge \ldots \wedge X(t_n) = x_n \,]$$
$$= P[\, X(t_{n+1}) = x_{n+1} \mid X(t_n) = x_n \,]$$

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, ...

Exkurs: Eigenschaften von Markov-Ketten mit diskretem Parameterraum (1)

Die Markov-Kette Xn mit diskretem Parameterraum heißt

homogen, wenn die Übergangswahrscheinlichkeiten
pij := $P[X_{n+1} = j \mid X_n = i]$ unabhängig von n sind

irreduzibel, wenn jeder Zustand von jedem Zustand mit positiver Wahrscheinlichkeit erreichbar ist:

$$\sum_{n=1}^{\infty} P[\, X_n = j \mid X_0 = i \,] > 0 \quad \text{für all i, j}$$

aperiodisch, wenn alle Zustände i die Periode 1 haben, wobei die Periode von i der ggT aller Werte n ist, für die gilt:

$$P[\, X_n = i \wedge X_k \neq i \ \text{für}\ k = 1,\ldots,n-1 \mid X_0 = i \,] > 0$$

Exkurs: Eigenschaften von Markov-Ketten mit diskretem Parameterraum (2)

Die Markov-Kette Xn mit diskretem Parameterraum heißt

positiv rekurrent, wenn für jeden Zustand i die Rückkehr-wahrscheinlichkeit gleich 1 ist und mittlere Rekurrenzzeit endlich:

$$\sum_{n=1}^{\infty} P[\, X_n = i \wedge X_k \neq i \ \textit{für}\ k = 1,\ldots,n-1 \,|\, X_0 = i\,] = 1$$

$$\sum_{n=1}^{\infty} n\, P[\, X_n = i \wedge X_k \neq i \ \textit{für}\ k = 1,\ldots,n-1 \,|\, X_0 = i\,] < \infty$$

ergodisch, wenn sie homogen, irreduzibel, aperiodisch und positiv rekurrent ist.

Resultate über Markov-Ketten mit diskretem Parameterraum (1)

Für die **n-Schritt-Transitionswahrscheinlichkeiten**

$$p_{ij}^{(n)} := P[\, X_n = j \mid X_0 = i \,] \quad \text{gilt:}$$

$$p_{ij}^{(n)} = \sum_k p_{ik}^{(n-1)}\, p_{kj} \quad \text{mit} \quad p_{ij}^{(1)} := p_{ik}$$

$$= \sum_k p_{ik}^{(n-l)}\, p_{kj}^{(l)} \quad \text{für } 1 \le l \le n-1$$

in Matrix-Notation: $P^{(n)} = P^n$

Für die **Zustandswahrscheinlichkeiten nach n Schritten**

$$\pi_j^{(n)} := P[\, X_n = j \,] \quad \text{gilt:}$$

$$\pi_j^{(n)} = \sum_i \pi_i^{(0)}\, p_{ij}^{(n)} \quad \text{mit Anfangswahrscheinlichkeiten } \pi_i^{(0)}$$

in Matrix-Notation: $\Pi^{(n)} = \Pi^{(0)} P^{(n)}$ *(Chapman-Kolmogorov-Gleichung)*

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 $\pi_j := \lim_{n\to\infty} \pi_j^{(n)}$
Diese sind unabhängig von $\Pi^{(0)}$
und durch das folgende lineare Gleichungssystem bestimmt:

$$\pi_j = \sum_i \pi_i \, p_{ij} \quad \textit{für alle } j$$

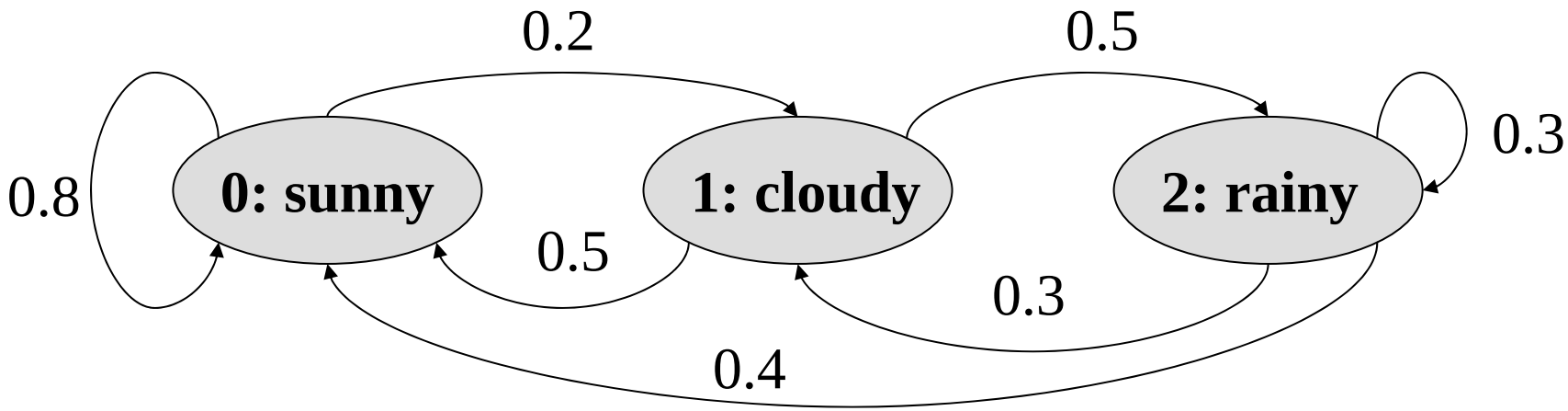
(Gleichgewichts-gleichungen)

$$\sum_j \pi_j = 1$$

in Matrix-Notation $\Pi = \Pi \, \mathrm{P}$

(mit 1×n-Vektor Π): $\Pi \vec{1} = 1$

Beispiel: Markov-Kette



$\pi 0 = 0.8\ \pi 0 + 0.5\ \pi 1 + 0.4\ \pi 2$

$\pi 1 = 0.2\ \pi 0 + 0.3\ \pi 2$

$\pi 2 = 0.5\ \pi 1 + 0.3\ \pi 2$

$\pi 0 + \pi 1 + \pi 2 = 1$

$\Rightarrow \pi 0 = 330/474 \approx 0.696$

$\pi 1 = 84/474 \approx 0.177$

$\pi 2 = 10/79 \approx 0.126$

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

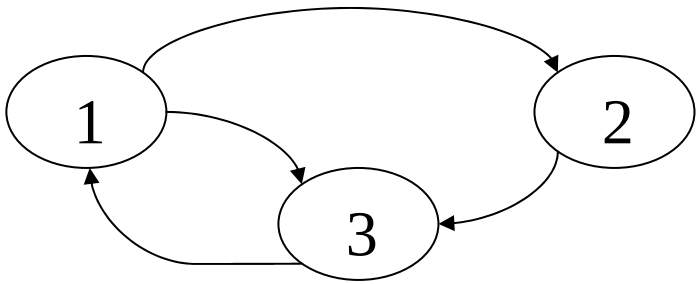
→ ergodische Markov-Kette

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

Beispiel: Page-Rank-Berechnung



$\varepsilon = 0.2$

$$P = \begin{pmatrix} 0.0 & 0.5 & 0.5 \\ 0.1 & 0.0 & 0.9 \\ 0.9 & 0.1 & 0.0 \end{pmatrix}$$

$$\Pi^{(0)} \approx \begin{pmatrix} 0.333 \\ 0.333 \\ 0.333 \end{pmatrix}^T \Rightarrow \Pi^{(1)} \approx \begin{pmatrix} 0.333 \\ 0.200 \\ 0.466 \end{pmatrix}^T \Rightarrow \Pi^{(2)} \approx \begin{pmatrix} 0.439 \\ 0.212 \\ 0.346 \end{pmatrix}^T \Rightarrow \Pi^{(3)} \approx \begin{pmatrix} 0.332 \\ 0.253 \\ 0.401 \end{pmatrix}^T$$

$$\Rightarrow \Pi^{(4)} \approx \begin{pmatrix} 0.385 \\ 0.176 \\ 0.527 \end{pmatrix}^T \Rightarrow \Pi^{(5)} \approx \begin{pmatrix} 0.491 \\ 0.244 \\ 0.350 \end{pmatrix}^T$$

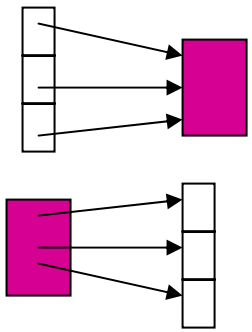
$\pi1 = 0.1\ \pi2 + 0.9\ \pi3$
$\pi2 = 0.5\ \pi1 + 0.1\ \pi3$
$\pi3 = 0.5\ \pi1 + 0.9\ \pi2$
$\pi1 + \pi2 + \pi3 = 1$

$\Rightarrow \pi1 \approx 0.3776,\ \pi2 \approx 0.2282,\ \pi3 \approx 0.3942$

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

Authority-Score $x_q = \sum_{(p,q)\in E} y_p$ und

Hub-Score $y_p = \sum_{(p,q)\in E} x_q$

HITS-Algorithmus (2)

Authority- und Hub-Scores in Matrix-Notation:

$$\vec{x} = A^T \vec{y} \qquad \vec{y} = A\vec{x}$$

Iteration mit Adjazenz-Matrix A:

$$\vec{x} := A^T \vec{y} := A^T A \vec{x} \qquad \vec{y} := A\vec{x} := A A^T \vec{y}$$

x und y sind also **Eigenvektoren** von A^TA bzw. AA^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

$M^{(hub)} := AA^T$ ist die Bibliographic-Coupling-Matrix: $M^{(hub)}_{ij}$ ist die Anzahl der Knoten, auf die i und j zeigen

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
   $\rightarrow$ 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

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

Authority-Score $$x_q = \sum_{(p,q)\in E} y_p * topic\,score(p) * xweight(p,q)$$

Hub-Score $$y_p = \sum_{(p,q)\in E} x_q * topic\,score(q) * yweight(p,q)$$

Bestimmung verwandter URLs

Cocitation-Algorithmus:

- Bestimme bis zu B Vorgänger der gegebenen URL u
- Für jeden Vorgänger p bestimme bis zu BF Nachfolger ≠ 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

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 ≠ u sowie
   - bis zu F Nachfolger von u und
     für jeden Nachfolger c bis zu FB Vorgänger ≠ 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

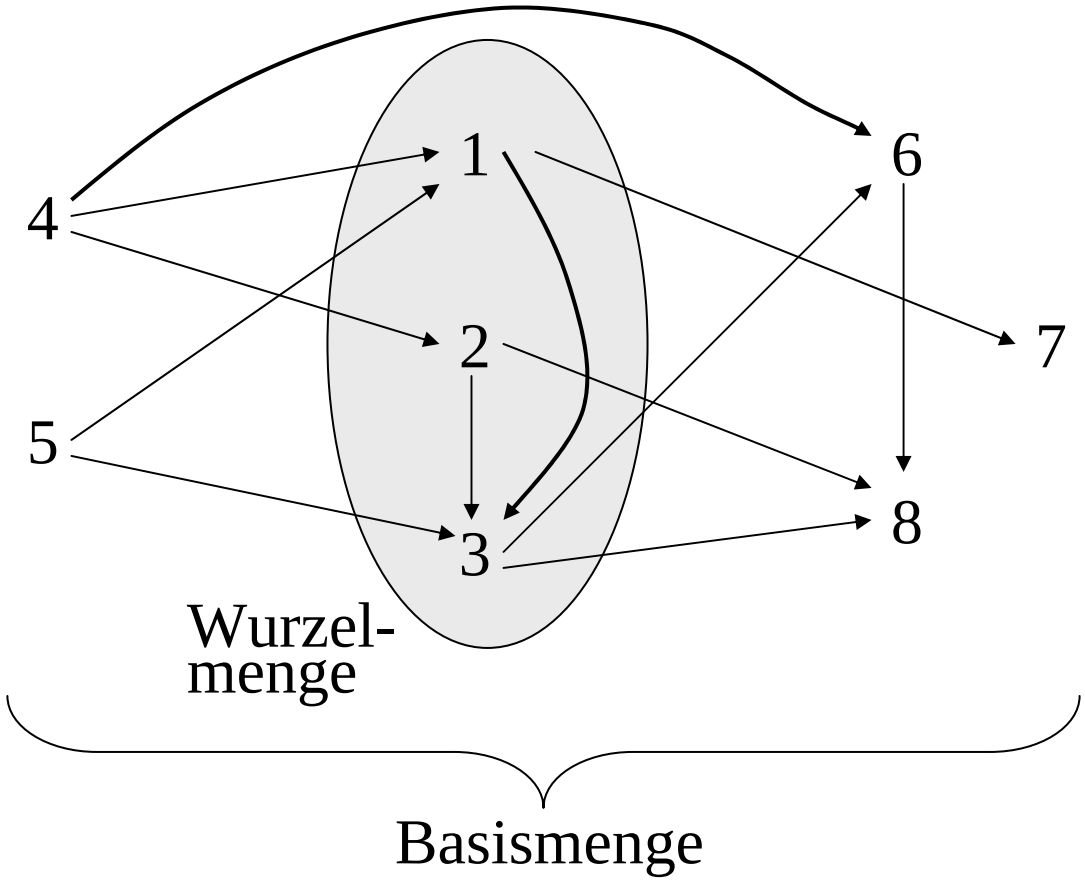
HITS-Algorithmus zur „Community Detection“

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“

Beispiel: HITS-Algorithmus



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:

$$\vec{r}_k = \varepsilon \, \vec{p}_k + (1-\varepsilon) A' \vec{r}_k$$ mit $A'_{ij} = 1/\text{outdegree}(i)$ für $(i,j) \in E$, 0 sonst

mit $(p_k)_j = 1/|T_k|$ für $j \in T_k$, 0 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
   $\rightarrow$ Wahrscheinlichkeit $w_k := P[c_k \mid q]$
3) Der Autoritäts-Score von Seite d ist $\sum_k w_k \, r_k(d)$

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 $\tau 1$ und $\tau 2$ (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 $U = \text{top}(k,\tau 1) \cup \text{top}(k,\tau 2)$

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

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.johnholleman.com	www.artistinformation.com	www.ncpamd.com/Women's_M
4 www.majorleaguebaseball	www.billboard.com	www.wingofmadness.com
5 www.mp3.com	www.soul-patrol.com	www.countrynurse.com

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 u1 und u2 persönliche Präferenzvektoren (für Random-Jump-Ziele) und seien r1 und r2 die zugehörigen Page-Rank-Vektoren. Dann gilt für alle $\alpha 1, \alpha 2 \geq 0$ mit $\alpha 1 + \alpha 2 = 1$:
$\alpha 1\ r1 + \alpha 2\ r2 = (1-\varepsilon)\ A' (\alpha 1\ r1 + \alpha 2\ r2) + \varepsilon\ (\alpha 1\ u1 + \alpha 2\ u2)$

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≠p gilt:

$u = \sum_{p=1}^{m} \alpha_p \cdot e_p$ mit Konstanten $\alpha_1 \ldots \alpha_m$

und $r = \sum_{p=1}^{m} \alpha_p \cdot r_p$ für den persönlichen Page-Rank-Vektor r

Kapitel 5: Relationenmodell und algebraorientierte Anfragesprachen

5.1 Grundbegriffe des Relationenmodells

5.2 Relationenalgebra

5.3 Erweiterte Relationenalgebra

5.1 Grundbegriffe des Relationenmodells

DB: Relationen (Tables)

Bücher

ISBN	Autor	Titel	...	Kategorie
1111	Grass	Blechtrommel		Roman
2222	Gates	Robin Hood		Märchen
3333	King	Windows 2000		Horror

Verkäufe

ISBN	KNr	Datum	...
3333	999	1.4.00	...

Kunden

KNr	Name	Ort	PLZ	Strasse	...
901	Becker	Saarlouis	...		
902	Caesar	Rom			

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

$\subseteq$ ***Dom (ISBN)*** $\times$ ***Dom (Autor)*** $\times$ ***... Dom (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)

Relationenmodell: Definitionen

Definition:
Gegeben sei eine Menge von Wertebereichen primitiver Datentypen {D1, ..., Dm}, die als ***"Domains"*** bezeichnet werden.
Eine ***Relation*** R ist ein Paar R = (s,v) mit
- einem *Schema* s = {A1, ..., An}, das aus einer Menge von ***Attributen (Attributnamen)*** besteht und für jedes Attribut Ai einen Domain dom(Ai) $\in$ {D1, ..., Dm} festlegt,
- und einer ***Ausprägung (Wert)*** v $\subseteq$ dom(A1) $\times$ dom(A2) $\times$ ... $\times$ dom(An).

Schema und Ausprägung von R werden mit *sch(r)* und *val(R)* bezeichnet.

Häufige Schreibweise für Relationenschemata:

R (A1, ..., An)
Bücher (ISBN, Autor, Titel, ...)
Kunden (KNr, Name, Stadt, ...)

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

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

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 !

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

Integritätsbedingungen des Relationenmodells (1)

Definitionen:

Eine Attributmenge $K \subseteq sch(R)$ einer Relation R heißt ***Schlüsselkandidat***, wenn zu jedem Zeitpunkt für je zwei Tupel $t1, t2 \in val(R)$ gelten muß:

$$t1.K = t2.K \Rightarrow t1 = t2$$

und wenn es keine echte Teilmenge von K gibt, die diese Eigenschaft hat.
(Dabei bedeutet $t1.K = t2.K: \forall A \in K: t1.A = t2.A$.)
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 $F \subseteq sch(S)$ 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.

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.

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 sch(R) = sch(S) sind die üblichen Mengenoperationen definiert:

- **Vereinigung (Union) $R \cup S$:**
  $sch(R \cup S) = sch(R)$
  $val(R \cup S) = \{t \mid t \in val(R) \vee t \in val(S)\}$
- **Durchschnitt (Intersection) $R \cap S$:**
  $sch(R \cap S) = sch(R)$
  $val(R \cap S) = \{t \mid t \in val(R) \wedge t \in val(S)\}$
- **Differenz (Difference) $R - S$:**
  $sch(R - S) = sch(R)$
  $val(R - S) = \{t \mid t \in val(R) \wedge t \notin val(S)\}$

Selektion und Projektion

Selektion σ (Filterung, Auswahl von Zeilen einer Tabelle):
Das Resultat einer Selektion $\sigma[F](R)$ auf einer Relation R ist:

$$sch(\sigma[F](R)) = sch(R)$$
$$val(\sigma[F](R)) = \{t \mid t \in R \wedge F(t)\}$$

Die Menge der möglichen Filterformeln F ist:

1) Für Attr. A, B von R mit dom(A)=dom(B), Konstanten $c \in dom(A)$ 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 $A \subseteq sch(R)$. Das Resultat einer Projektion $\pi[A](R)$ ist:

$$sch(\pi[A](R)) = A$$
$$val(\pi[A](R)) = \{t \mid \exists r \in val(R): t.A = r.A\}$$

Beispiele Selektion

1) Finde alle Homburger Kunden.

$\rightarrow \sigma$[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.

$\rightarrow \sigma$[Stadt='Homburg' $\wedge$ Rabatt >= 0.15] (Kunden)

KNr	Name	Stadt	Saldo	Rabatt
2	Schneider	Homburg	-800.00	0.20

Beispiele Projektion

1) Gib alle Produktbezeichnungen aus.
   → π[Bez] (Produkte)

Bez
Papier
Platte
Drucker
Bildschirm
Disketten
Maus
Speicher

2) Gib alle Lagerorte von Produkten aus.
   → π[Lagerort] (Produkte)

Lagerort
Homburg
Saarbrücken
Merzig

(Natural) Join |×| und Zuweisung

(Natural) Join |×| (Natürlicher Verbund
von Relationen über gleiche Attributnamen und Attributwerte):
Das Resultat von R |×| S mit A=sch(R) und B=sch(S) ist:

sch(R |×| S) = sch(R) ∪ sch(S)

val(R |×| S) = {t | ∃ r ∈ val(R) ∃ s ∈ val(S): t.A = r.A ∧ t.B = s.B}

Zuweisung:

Seien R, S Relationen mit sch(R)={A1, ..., An} und sch(S)={B1, ..., Bn},
so daß für alle i gilt: dom(Ai)=dom(Bi).
Die Zuweisung *R := S* bedeutet: val(R) = val(S).
Ausführlicher schreibt man auch R(A1, ..., An) := S(B1, ..., Bn).
Für Ausdrücke E1, ..., En, die über B1, ..., Bn und k-stelligen Operatoren
ψ: dom(B_{i1}) x … x dom(B_{ik}) $\rightarrow$ D mit skalarem Resultat im Bereich W
bedeutet ***R(A1, ..., An) := S(E1, ..., En):***

val(R) = {t | es gibt s ∈ val(S) und t.Ai = Ei(s) für alle i},

wobei der Wertebereich von Ei gleich dom(Ai) sein muß.

Beispiele für Join

1) Alle Bestellungen zusammen mit den dazugehörigen Produktdaten.
   → Bestellungen |×| Produkt

2) R |×| S

R

R1	J
a	1
b	2
c	2
d	3

S

S1	J
w	2
x	2
y	3
z	4

→

R1	J	S1
b	2	w
b	2	x
c	2	w
c	2	x
d	3	y

Kartesisches Produkt und Division

Kartesisches Produkt × :

Seien R, S Relationen mit Schemata A=sch(R) und B=sch(S).
Sei A' ein Schema, bei dem alle Ai, die auch in B vorkommen, unbenannt sind in R.Ai, und sei B' ein analoges Schema mit Attributnamen S.Ai.
Das Resultat von R × S ist:

$$sch(R \times S) = A' \cup B'$$

$$val(R \times S) = \{t \mid \exists r \in val(R)\ \exists s \in val(S):\ t.A' = r.A \text{ und } t.B' = s.B\}$$

Division ÷ :

Seien R, S Relationen mit A=sch(R) und B=sch(S), so daß $B \subset A$.
Das Resultat der Division R ÷ S ist:

$$sch(R \div S) = A - B = Q$$

$$val(R \div S) = \{t \mid \forall s \in val(S)\ \exists r \in val(R): r.B = s.B \wedge t = r.Q\}$$

Intuitive Bedeutung: ein Tupel t ist in R ÷ S genau dann, wenn für alle S-Tupel s ein Tupel <t,s> in R enthalten ist.

Satz: Für R(A,B,C), T(A,B), S(C) mit R=T × S gilt: R ÷ S = T.

Beispiel Division

Kundennummern derjenigen Kunden, die alle überhaupt lieferbaren Produkte irgendwann bestellt haben

$\rightarrow \pi[KNr, PNr]\ (Bestellungen) \div \pi[PNr]\ (Produkte)$

KNr	PNr
1	1
1	2
1	3
1	4
1	5
1	6
1	7
2	1
2	5
3	5
3	7
4	1
6	1
6	7

PNr
1
2
3
4
5
6
7

Rückführung Division auf andere Operationen

Satz:

Seien R und S Relationen mit A=sch(R) und B=sch(S) mit $A \supset B$, und sei Q=sch(R) - 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):

$R := \pi[KNr, PNr]$ (Bestellungen)	
$S := \pi[PNr]$ (Produkte)	
$T1 := \pi[KNr](R) \times S$	alle überhaupt möglichen Bestellungen
$T2 := T1 - R$	potentiell mögliche, aber nicht erfolgte Bestellungen
$T3 := \pi[KNr]$ (T2)	Kunden, die nicht alle Prod. bestellt haben
$T4 := \pi[KNr]$ (R) - T3	Kunden, die alle Produkte bestellt haben

θ-Join

θ-Join:

Seien R, S Relationen mit $sch(R) \cap sch(S) = \emptyset$. Seien $A \subseteq sch(R)$ und $B \subseteq sch(S)$, und sei θ eine der Operationen $=, \neq, <, >, \leq, \geq$.
Das Resultat des θ-Joins $R \,|\times|[A\ \theta\ B]\, S$ ist:

$$sch(R \,|\times|[A\ \theta\ B]\, S) = sch(R \times S) = sch(R) \cup sch(S)$$
$$val(R \,|\times|[A\ \theta\ B]\, S) = val(\sigma[A\ \theta\ B](R \times S))$$

Beispiele:

1) alle Kunden, die an einem Lagerort wohnen.
   → π[KNr, Name, Stadt, ...] (Kunden |×|[Stadt=Lagerort] Produkte)
2) alle bisherigen Bestellungen, die den momentanen Vorrat erschöpfen würden.
   → B(BestNr, KNr,B.PNr, ...) := Bestellungen(BestNr, KNr, PNr, ...)
   π[BestNr, ...] (B |×|[B.PNr=PNr ∧ Menge >= Vorrat] Produkte)
3) alle Paare von Kunden, die in derselben Stadt wohnen.
   → K1(K1.KNr, K1.Name, ...) := Kunden(KNr, Name, ...)
   K2(K2.KNr, K2.Name, ...) := Kunden(KNr, Name, ...)
   π[K1.KNr, K2.KNr]
   (K1 |×|[K1.Stadt=K2.Stadt ∧ K1.KNr < K2.KNr] K2)

Outer Joins

Seien R, S Relationen mit A=sch(R), B=sch(S) und J=sch(R) $\cap$ sch(S).
Bezeichne ferner ω den Nullwert.
Das Resultat des **Outer Joins R |*| S** ist:

$$\text{sch}(R \,|*|\, S) = \text{sch}(R) \cup \text{sch}(S)$$

$$\text{val}(R \,|*|\, S) = \text{val}(R \,|\times|\, S) \cup$$
$$\{t \mid \exists\, r \in \text{val}(R) : t.A = r.A \wedge \neg(\exists\, s \in \text{val}(S) : t.J = s.J) \wedge t.(B-J) = \omega\} \cup$$
$$\{t \mid \exists\, s \in \text{val}(S) : t.B = s.B \wedge \neg(\exists\, r \in \text{val}(R) : t.J = r.J) \wedge t.(A-J) = \omega\}.$$

Left Outer Join:

$$\text{sch}(R \,||*|\, S) = \text{sch}(R) \cup \text{sch}(S)$$

$$\text{val}(R \,||*|\, S) = \text{val}(R \,|\times|\, S) \cup$$
$$\{t \mid \exists\, r \in \text{val}(R) : t.A = r.A \wedge \neg(\exists\, s \in \text{val}(S) : t.J = s.J) \wedge t.(B-J) = \omega\}$$

Right Outer Join:

$$\text{sch}(R \,|*||\, S) = \text{sch}(R) \cup \text{sch}(S)$$

$$\text{val}(R \,|*||\, S) = \text{val}(R \,|\times|\, S) \cup$$
$$\{t \mid \exists\, s \in \text{val}(S) : t.B = s.B \wedge \neg(\exists\, r \in \text{val}(R) : t.J = r.J) \wedge t.(A-J) = \omega\}.$$

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.

σ [Rabatt = 0.05] (Kunden |*| 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	...

Äquivalenzregeln (“Rechenregeln”) der RA

Kommutativitätsregeln:

1) $\pi[R1]\ \sigma[P]\ (R) = \sigma[P]\ \pi[R1]\ (R)$ falls P nur R1-Attribute enthält

2) $R \ |x|\ S = S \ |x|\ R$

Assoziativitätsregeln:

3) $R \ |x|\ (\ S \ |x|\ T\ ) = (\ R \ |x|\ S\ ) \ |x|\ T$

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) $\sigma[P]\ (\ R \ |x|\ S\ ) = \sigma[P](R) \ |x|\ S$ falls P nur R-Attribute enthält

9) $\pi[R1,S1](R \ |x|\ S) = \pi[R1](R) \ |x|\ \pi[S1](S)$ falls Joinattribute $\subseteq R1 \cup S1$

10) $R \ |x|\ (\ S \cup T\ ) = (R \ |x|\ S) \cup (R \ |x|\ T)$

Invertierungsregeln:

11) $\pi[sch(R)]\ (R \ ||^{*}\ S) = R$

Äquivalenzregeln ("Rechenregeln") der RA

Kommutativitätsregeln:

1) $\pi[R1]\ \sigma[P]\ (R) = \sigma[P]\ \pi[R1]\ (R)$ falls P nur R1-Attribute enthält

2) $R \ |x|\ S = S \ |x|\ R$

Assoziativitätsregeln:

3) $R \ |x|\ (\ S \ |x|\ T\ ) = (\ R \ |x|\ S\ ) \ |x|\ T$

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) $\sigma[P]\ (\ R \ |x|\ S\ ) = \sigma[P](R) \ |x|\ S$ falls P nur R-Attribute enthält

9) $\pi[R1,S1](R \ |x|\ S) = \pi[R1](R) \ |x|\ \pi[S1](S)$ falls Joinattribute $\subseteq R1 \cup S1$

10) $R \ |x|\ (\ S \cup T\ ) = (R \ |x|\ S) \cup (R \ |x|\ T)$

Invertierungsregeln:

11) $\pi[sch(R)]\ (R \ ||^* \ S) = R$

Ausdrucksmächtigkeit der RA

Die Menge der **relationenalgebraischen Ausdrücke** über einer Menge vo
Relationen R1, ..., Rn ist wie folgt definiert:

(i) R1, ..., Rn sind Ausdrücke.

(ii) Wenn R, S, T, Q Ausdrücke sind, F eine Filterformel über sch(R) ist,
A $\subseteq$ sch(R), sch(S)=sch(T und sch(R) $\supset$ sch(Q) gilt, dann sind
σ[F](R), π[A](R), R |x| S, R x S, R |*| 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:

$\times$, π, σ, $\cup$ und - 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) Relationenalgebra ausdrücken lassen.

5.3 Erweiterte Relationenalgebra

Definition:

Eine ***Multimenge*** (engl. multiset, bag) M über einer Grundmenge G ist eine Abbildung $M: G \rightarrow N_0$.

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 = {A1, ..., An} und einer Ausprägung v, die eine Multimenge über dom(A1) × ... × dom(An) 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:
val $(\chi(R)) = \{t \mid R(t)>0\}$.

Operationen auf Multimengen und -relationen

Für Multirelationen R, S mit sch(R) = sch(S) sind:

Vereinigung $R \cup_+ S$:

$sch(R \cup_+ S) = sch(R)$, $val(R \cup_+ S) = t \mapsto R(t) + S(t)$

Durchschnitt $R \cap_+ S$:

$sch(R \cap_+ S) = sch(R)$, $val(R \cap_+ S) = t \mapsto \min(R(t), S(t))$

Differenz $R -_+ S$:

$sch(R -_+ S) = sch(R)$, $val(R -_+ S) = t \mapsto R(t)\text{-}S(t)$ falls $R(t) \geq S(t)$, 0 sonst

Das Resultat der **Selektion $\sigma_+ [F](R)$** auf einer Multirelation R ist:

$sch(\sigma_+ [F](R)) = sch(R)$,

$val(\sigma_+ [F](R)) = t \mapsto R(t)$ falls $R(t) \wedge F(t)$, 0 sonst

Das Resultat der **Projektion $\pi_+ [A](R)$** auf einer Multirelation R mit $A \subseteq sch(R)$ ist:

$sch(\pi_+ [A](R)) = A$,

$val(\pi_+ [A](R)) = t \mapsto \Sigma \{R(r) \mid r \in val(R) \text{ und } r.A = t.A\}$

Aggregation und Gruppierung

Aggregation α_+ (Zusammenfassen von Zeilen einer Tabelle):
Sei $A \in sch(R)$ und sei f eine Funktion, die Multimengen über dom(A) in einen Wertebereich W abbildet (z.B. max, min, sum, count, median).
Das Resultat der Aggregation **α_+ [A,f](R)** auf der Multirelation R ist:

$sch(\alpha_+ [A,f](R)) = A'$ mit $dom(A')=W$
$val(\alpha_+ [A,f](R)) = f(\pi_+[A](R))$

Gruppierung γ_+ (Zusammenfassen von Äquivalenzklassen von Zeilen):
Sei $X \subseteq sch(R)$, $A \in sch(R)$, und sei f eine Funktion, die Multimengen über dom(A) in einen Wertebereich W abbildet.
Das Resultat einer Gruppierung **γ_+ [X,A,f](R)** auf der Multirelation R ist:

$sch(\gamma_+ [X,A,f](R)) = X \cup \{A'\}$ mit $dom(A')=W$
$val(\gamma_+ [X,A,f](R)) = \{ t$ | es gibt eine Äquivalenzklasse G von $\pi_+[X](R)$ unter der Wertegleichheit und t.X ist der Wert der Tupel in G und $t.A' = f(\pi_+[A](G)) \}$

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:

$\sigma_{+}[\text{Preis}<50.00](\pi_{+}[\text{Lagerort, Preis}](\text{Produkte}))$

Lagerort	Preis
Homburg	20.00
Homburg	20.00

2) Gesamtstückzahl (aller Produkte) über alle Lager:

Resultat(Gesamtvorrat):=α_{+}[Vorrat,sum](Produkte)

Gesamtvorrat
17880

Beispielanfragen auf Multirelationen (2)

3) Bestimme für jedes Lager die Gesamt-stückzahl aller dort gelagerten Produkte:

Resultat (Lagerort, Gesamtvorrat) :=
γ+ [{Lagerort}, Vorrat, sum](Produkte)

Lagerort	Gesamtvorrat
Homburg	15200
Saarbrücken	2400
Merzig	280

4) Bestimme für jedes Lager die Gesamtkapitalbindung (Stückzahl * Preis) aller dort gelagerten Produkte:

Lagerort	Kapitalbindung
Homburg	320 000
Saarbrücken	1 400 000
Merzig	640 000

P1 := π+ [Lagerort, Preis, Vorrat] (Produkte)
P2 (Lagerort, Wert) := P1(Lagerort, Preis*Vorrat)
Resultat (Lagerort, Kapitalbindung) :=
γ+ [{Lagerort}, Wert, sum](P2)

Weitere Erweiterungen der RA

Transitive Hülle R^+ einer binären Relation R:

Sei R(A, B) eine binäre Relation mit dom(A)=dom(B).

Das Resultat der transitiven Hülle R^+ ist:

sch(R^+) = sch(R)

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 := (π[Abflugort, Zielort] (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
...	...

Kapitel 6: Logikorientierte Anfragesprachen

6.1 Überblick: Dualismus der Logik

Syntax
(beweistheoretische Sicht):

Formeln F1, ..., Fn
mit Aussagenvariablen,
Prädikat-/Funktionssymbolen

Ableitungsregeln
(Beweis-/Deduktionsregeln):
F1, ..., Fn ⊢ G

Semantik
(modelltheoretische Sicht):

Interpretation ψ in
(mathematischer) Struktur S

Gültigkeit in S unter ψ
(G ist in S wahr):
ψ ⊨ 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 Ai (i=1, 2, ...).

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 Formelmenge 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 ψ: D $\rightarrow$ P, für die gilt
ψ(True)=1, ψ(False)=0, ψ(Ai)=1 (0), falls ψ(Ai) 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 F1, ..., Fk (F1,...,Fk |= G), wenn für jedes ψ gilt:
falls ψ Modell von F1, ..., Fk 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 F1, ..., Fk g.d.w. $(F1 \wedge F2 \wedge ... \wedge Fk) \Rightarrow G$ 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) (v) $F \Rightarrow G, F \vdash G$

Definition:

Sei H eine endl. Formelmenge. F heißt aus H ***ableitbar (H |— F),*** wenn es eine endl. Sequenz F0, F1, ..., Fn von Formeln gibt mit Fn=F, so daß für alle Fi gilt: Fi ist Element von H, Fi ist eine der Formeln (i) bis (iii) oder Fi ist aus F0, ..., F(i-1) mittels (iv) oder (v) abgeleitet. Für H=∅ 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):

$G \Leftrightarrow (B \vee F) \wedge (\underline{F \wedge B} \vee \underline{E}) \wedge (\underline{(E \vee \neg B)} \vee \underline{F})$	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 xi, Prädikatsymbole Pi*** der Stelligkeit ki und ***Funktionssymbole fi*** der Stelligkeit li. ***Terme*** sind:

(i) Jede Variable ist ein Term

(ii) Wenn t1, ..., tk Terme sind und fi ein k-stelliges Funktionssymbol ist, dann ist auch fi(t1, ..., tk) ein Term.

Eine ***atomare Formel*** hat die Form ***Pi(t1, ..., tk)*** mit einem k-stelligen Prädikatsymbol Pi und Termen t1, ..., tk.

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 xi und eine Formel F sind auch $\forall$ ***xi (F)*** und $\exists$ ***xi (F)*** Formeln (mit ***gebundener*** Variable xi).

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, fun, pred)*** mit

- einem ***Universum*** (einer ***Trägermenge***) ***U*** von Individuen,
- einer Menge ***fun*** von ***Funktionen fi:*** $U \times ... \times U \rightarrow U$ der Stelligkeit li, so daß für jedes li-stellige Symbol eine entspr. Funktion existiert
- einer Menge ***pred*** von ***Prädikaten Pi:*** $U \times ... \times U \rightarrow \{0,1\}$ der Stelligkeit ki, so daß für jedes ki-stellige Symbol ein entspr. Prädikat existiert.

Eine ***Interpretation*** ψ einer Formel F in einer passenden Struktur S = (U, fun, 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) $\psi(f(t1, ..., tk)) = \psi(f)(\psi(t1), ..., \psi(tk))$
für ein Funktionssymbol f und Terme t1, ..., tk,

(vi) $\psi(x) = a$ mit einem beliebigen $a \in U$ für eine (freie) Variable x

(vii) $\psi(P(t1, ..., tk)) = \psi(P)(\psi(t1), ..., \psi(tk))$
für ein Prädikatsymbol P und Terme t1, ..., tk,

(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 $S = (N_0, \{+, \cdot\}, \{>, =\})$.
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, Lauer, Merzig) $\wedge$ K(2, Schneider, Homburg) $\wedge$
P(1, 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 $N_0 \cup \Sigma^*$
Interpretation ψ (F) = 1, falls es die vier Tupel in der Datenbank gibt,
0 sonst.

Erweiterungen der Prädikatenlogik

Mehrsortige Logik mit mehreren Trägermengen (Sorten)
und einer Signatur für die Prädikate und Funktionen.
Beispiel: U= {U1, U2} mit U1=R^n, U2=R und

Addition: U1 × U1 → U1, Multiplikation: U1 × U2 → U1,
Skalarprodukt: U1 × U1 → U2,
Vektorprodukt: U1 × U1 → U1, Nullvektor: → U1,
Prädikaten Orthogonal: U1 × U1 → {0,1} usw.

Prädikate als Argumente von Prädikaten oder Funktionen,
Quantifizierung von Prädikaten oder Funktionen
→ ***Prädikatenlogik höherer Ordnung***
Beispiel:

$$\forall x,y\ (\ H(x,y) \Leftrightarrow \forall P\ (\ (R(x,y) \Rightarrow P(x,y)) \wedge \forall u,w,z\ (P(u,w) \wedge P(w,z) \Rightarrow P(u,z))) \Rightarrow (H(x,y) \Rightarrow P(x,y))\ )$$

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 F1, ..., Fk (F1,...,Fk $\models$ G),*** wenn für jedes ψ gilt: falls ψ Modell von F1, ..., Fk 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 F1, ..., Fk g.d.w. $(F1 \wedge F2 \wedge ... \wedge Fk) \Rightarrow G$ eine Tautologie ist. F und G sind äquivalent g.d.w. F Folgerung von G ist und umgekehrt.

Prädikatenlogik: Beispiele (2)

Formeln F1, ..., F5 bzw. Formel F= F1 $\wedge$ ... $\wedge$ 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 $H \Leftrightarrow H[F/G]$

$\neg\ \forall(F) \Leftrightarrow \exists x\ (\neg F)$

$\neg\ \exists\ (F) \Leftrightarrow \forall x\ (\neg F)$

$(\forall x\ (F)) \wedge G \Leftrightarrow \forall x\ (F \wedge G)$ falls x in G nicht frei vorkommt

$(\forall x\ (F)) \vee G \Leftrightarrow \forall x\ (F \vee G)$ falls x in G nicht frei vorkommt

$(\exists x\ (F)) \wedge G \Leftrightarrow \exists x\ (F \wedge G)$ falls x in G nicht frei vorkommt

$(\exists x\ (F)) \vee G \Leftrightarrow \exists x\ (F \vee G)$ falls x in G nicht frei vorkommt

$(\forall x\ (F)) \wedge (\forall y\ (G)) \Leftrightarrow \forall x\ (F \wedge G[y/x])$, falls x in G nicht frei vorkommt

$(\exists x\ (F)) \vee (\exists y\ (G)) \Leftrightarrow \exists x\ (F \vee G[y/x])$, falls x in G nicht frei vorkommt

$\forall x\ (\forall y\ (F)) \Leftrightarrow \forall y\ (\forall x\ (F))$

$\exists x\ (\exists y\ (F)) \Leftrightarrow \exists y\ (\exists x\ (F))$

$\forall x\ (F) \Rightarrow F[x/a]$ für alle Konstanten a

$(\ (\forall x\ (F)) \wedge (F \Rightarrow G)\ )\ \Rightarrow\ (\forall x\ (G))$

Vollständige und korrekte Kalküle: Resolutionskalkül, Tableaukalkül

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)

SindVerwandt 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):

$$\forall x\ \ \forall y\ (\ I(hb) \wedge (S(x) \Rightarrow V(x,hb)) \wedge (V(x,y) \wedge I(y)) \Rightarrow M(x))\ )$$
$$\vdash\ \ \forall x\ \ (\ I(hb) \wedge (S(x) \Rightarrow V(x,hb)) \wedge (V(x,hb) \wedge I(hb)) \Rightarrow M(x))\ \ )$$
$$\vdash\ \ \forall x\ \ (\ S(x) \Rightarrow M(x)\ )\ \ (*)$$

(*) $\wedge$ (ii) $\wedge$ (i):

$$\forall x\ \ (\ (S(x) \Rightarrow M(x))\ \wedge\ (M(x) \Rightarrow I(x))\ \wedge\ (I(x) \Rightarrow P(x))\ )$$
$$\vdash\ \ \forall x\ \ (\ S(x) \Rightarrow P(x)\ )$$

Fazit: alle Saarländer können programmieren!

6.2 Domain-Relationenkalkül (DRK)

Anfragen sind prädikatenlogische Mengenspezifikationen der Form

{<x1, ..., xn> | F(x1, ..., xn)}

wobei F ein prädikatenlogischer Ausdruck über einer Menge von ***Domain-Variablen*** ist mit x1, ..., xn als einzigen freien Variablen.

Die Menge der zulässigen prädikatenlogischen Ausdrücke F ist:

1) Für Variablen x1, ..., xn und eine Relation R mit n Attributen ist $<x1,...,xn> \in R$ (auch R(x1, ..., xn)) 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 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 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
R1(A11, ..., A1n_1), ..., Rm(Am1, ..., Amn$_m$) ist eine Formelmenge H,
die für jede Relation Ri ein Prädikatsymbol Ri verwendet und für jedes Tupel ***<a1, ..., an$_i$> ∈ val(Ri)*** eine atomare Formel ***Ri(a1, ..., an$_i$)*** enthält, bzw. ein Modell von H.

Das Ergebnis einer ***Anfrage {<x1, ..., xn> | F(x1, ..., xn)}***
über der Datenbank H ist:
{<a1, a2, ... an> | a1, a2, ..., an ∈ U und H |= F[x1/a1, x2/a2, ..., xn/an]}

Sonderfall:
wenn F keine freien Variablen enthält, ist das Ergebnis
1 wenn H |= F und 0 sonst.

Beispiele für DRK-Anfragen (1)

1) Finden Sie (die Namen) alle(r) Kunden mit negativem Saldo.

$\rightarrow$ {<k,n,st,sa,r> | <k,n,st,sa,r> $\in$ Kunden $\wedge$ sa < 0.0} bzw.

$\rightarrow$ {<n> | $\exists$ k,st,sa,r: <k,n,st,sa,r> $\in$ Kunden $\wedge$ sa < 0.0}

2) Finden Sie die Namen aller Kunden, die eine unbezahlte Bestellung haben, die vor Anfang Oktober erfolgte.

$\rightarrow$ {<n> | $\exists$ k,st,sa,r: <<u>k</u>,n,st,sa,r> $\in$ Kunden $\wedge$
$\exists$ b,m,t,p,me,su,stat: <b,m,t,<u>k</u>,p,me,su,stat> $\in$ Best. $\wedge$
m < 10 $\wedge$ stat $\neq$ 'bezahlt'}

3) Namen der Homburger Kunden, die seit Sept. ein Prod. aus Homburg geliefert bekommen haben, jeweils mit der Bez. des Produkts.

$\rightarrow$ {<n,bez> | $\exists$ k,st,sa,r: <<u>k</u>,n,st,sa,r> $\in$ Kunden $\wedge$
$\exists$ p,g,pr,l,v: <<u>p</u>,bez,g,pr,l,v> $\in$ Produkte $\wedge$
$\exists$ b,m,t,me,su,stat: <b,m,t,<u>k</u>,<u>p</u>,me,su,stat> $\in$ Best. $\wedge$
st='Homburg' $\wedge$ mw9 $\wedge$ l='Homburg'}

Beispiele für DRK-Anfragen (2)

4) Kunden, von denen mindestens eine Bestellung registriert ist.

$\rightarrow \{<k,n,st,sa,r> \mid <\underline{k},n,st,sa,r> \in \text{Kunden} \wedge$
$\exists\, b,m,t,p,me,su,stat: <b,m,t,\underline{k},p,me,su,stat> \in \text{Best.}\}$

5) Kunden, von denen keine Bestellung registriert ist.

$\rightarrow \{<k,n,st,sa,r> \mid <k,n,st,sa,r> \in \text{Kunden} \wedge$
$\forall\, b,m,t,k',p,me,su,stat:$
$<b,m,t,k',p,me,su,stat> \in \text{Best.} \Rightarrow k \neq k'\}$

6) Kunden, die alle überhaupt lieferbaren Produkte bestellt haben.

$\rightarrow \{<k,n,st,sa,r> \mid <\underline{k},n,st,sa,r> \in \text{Kunden} \wedge$
$\forall\, p: (\exists\, bez,g,pr,l,v: <\underline{p},bez,g,pr,l,v> \in \text{Produkte})$
$\Rightarrow$
$(\exists\, b,m,t,me,su,stat: <b,m,t,\underline{k},\underline{p},me,su,stat>$
$\in \text{Bestellungen}) \}$

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 | F(t)} bzw. {<t1.A1, ..., tn.An> | F(t1, ..., tn)},

wobei F eine prädikatenlogische Formel über einer Menge von ***Tupelvariablen*** ist mit t bzw. t1, ..., tn als einzigen freien Variablen und A1, ..., 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 dom(A)=dom(B), Konst. $c \in dom(A)$ und Vergleichsoperationen $\theta \in \{=, \neq, <, >, \leq, \geq\}$ sind r.A θ s.B und r.A θ 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 | F(t)}*** ***oder*** ***{<t.B1, ..., t.Bn> | F(t)}*** wird

- jeder Term der Form ***r.Ai*** 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, ..., Am)
  durch einen Term $R(ra_1, ..., ra_m)$ mit Domain-Variablen ra_1, ..., ra_m.

Beispiele für TRK-Anfragen (1)

1) Finden Sie (die Namen) alle(r) Kunden mit negativem Saldo.

$\rightarrow \{t \mid t \in \text{Kunden} \wedge t.\text{Saldo} < 0.0\}$ 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.

$\rightarrow \{t.\text{Name} \mid t \in \text{Kunden} \wedge \exists\, b: b \in \text{Bestellungen} \wedge$
$t.\text{KNr}=b.\text{KNr} \wedge b.\text{Monat} < 10 \wedge b.\text{Status} \neq \text{'bezahlt'}\}$

3) Namen der Homburger Kunden, die seit Sept. ein Prod. aus Homburg geliefert bekommen haben, jeweils mit der Bez. des Produkts.

$\rightarrow \{k.\text{Name}, p.\text{Bez} \mid k \in \text{Kunden} \wedge p \in \text{Produkte} \wedge$
$\exists\, b: b \in \text{Bestellungen} \wedge$
$k.\text{Stadt}=\text{'Homburg'} \wedge b.\text{Monat} \geq 9 \wedge p.\text{Lagerort}=\text{'Homburg'} \wedge$
$k.\text{KNr}=b.\text{KNr} \wedge b.\text{PNr}=p.\text{PNr}\}$

Beispiele für TRK-Anfragen (2)

4) Kunden, von denen mindestens eine Bestellung registriert ist.

$\rightarrow \{t \mid t \in \text{Kunden} \wedge \exists\, b: b \in \text{Bestellungen} \wedge b.KNr=t.KNr\}$

5) Kunden, von denen keine Bestellung registriert ist.

$\rightarrow \{t \mid t \in \text{Kunden} \wedge \neg(\exists\, b: b \in \text{Bestellungen} \wedge b.KNr=t.KNr)\}$

oder

$\{t \mid t \in \text{Kunden} \wedge \forall\, b: b \in \text{Bestellungen} \Rightarrow b.KNr \neq t.KNr\}$

6) Kunden, die alle überhaupt lieferbaren Produkte bestellt haben.

$\rightarrow \{t \mid t \in \text{Kunden} \wedge \forall\, p: p \in \text{Produkte} \Rightarrow$
$(\exists\, b: b \in \text{Bestellungen} \wedge b.PNr=p.PNr \wedge$
$b.KNr=t.KNr)\}$

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 sch(R)={A}, dom(A)={1,2}, val(R)={<1>}.

Der Ausdruck $\{t \mid \neg (t \in R)\}$ hat
als unbeschränkte Interpretation den Wert {<2>} und
als beschränkte Interpretation den Wert Ø.

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 Ø.

Der Ausdruck $\{t \mid t \in R \wedge \forall\ x : x=t\}$ hat
als unbeschränkte Interpretation den Wert Ø 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 | F(t)} F(t) nur für t aus ADom erfüllbar ist),
- **durch Existenzquantoren gebundene Variablen beschränkt** werden
  (so daß in ∃ x : G(x) G(x) nur für x aus ADom erfüllbar ist) und
- **durch Allquantoren gebundene Variablen unbeschränkt** werden
  (so daß in ∀ 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 sch(R)={A,B}, val(R)={<2,2>}.

$\{t \mid \neg(t \in R) \vee t.A=2 \vee t.B=2\}$

$\{t \mid \neg(t \in R) \wedge t.A=2\}$

$\{t \mid \neg(t \in R) \wedge t.A=2 \wedge t.B=2\}$

$\{t \mid \exists s: (s \in R \wedge s.A=t.A)\}$

$\{t \mid \exists s: (s \in R \vee s=t)\}$

$\{t \mid \exists s: (s \in R \wedge s=t)\}$

$\{t \mid \neg(\exists s: (s \in R \wedge s=t))\}$

$\{t \mid \forall s: (s \in R \wedge s=t)\}$

$\{t \mid \forall s: (s \in R \Rightarrow s=t)\}$

$\{t \mid \forall s: (s \in R \Rightarrow (s=t \wedge t.A=2 \wedge t.B=2))\}$

$\{t \mid t \in R \wedge \forall s: (s \in R \Rightarrow (s=t \wedge t.A=2 \wedge t.B=2))\}$

$\{t \mid \forall s: (s \in R \wedge (s=t \wedge t.A=2 \wedge t.B=2))\}$

$\{t \mid \forall s: (s \in R \Rightarrow \neg(s=t))\}$

$\{t \mid t \in K \wedge \forall p: p \in P \Rightarrow (\exists b: b \in B \wedge b.PNr=p.PNr \wedge b.KNr=t.KNr)\}$

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 |x| [V.Zielort=F.Abflugort] F)

Datalog: Definitionen (1)

Eine ***Hornklausel*** über einer endlichen Menge von Prädikatsymbolen ist eine logische Formel der Form

$$\forall X1, ..., Xn: P1 (Z11, Z12, ..., Z1k_1) \wedge ... \wedge Pm (Zm1, Zm2, ..., Zmk_m) \Rightarrow P0 (Z01, Z02, ..., Z0k_0)$$

mit ki-stelligen Prädikaten Pi, Variablen X1, ..., Xn und Konstanten oder Variablen Zij.

Die Formel

$P0 (Z01, Z02, ..., Z0k_0)$

wird als ***"Kopf"*** (engl.: head) der Hornklausel bezeichnet, die Formel

$P1 (Z11, Z12, ..., Z1k_1) \wedge ... \wedge Pm (Zm1, Zm2, ..., Zmk_m)$

als ***"Rumpf"*** (engl.: body).

Datalog: Definitionen (2)

Ein ***Datalog-Programm*** besteht aus
- einer Menge von Fakten der Form Pi (V1, V2, ..., Vk)
  mit einem k-stelligen Prädikat Pi und Konstanten V1, ..., Vk 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(Z1, ..., Zk)
mit einem k-stelligen Prädikat P und Konstanten oder Variablen
Z1, ..., Zk, so daß mindestens eines der Zj eine (freie) Variable ist.

Eine Regel r heißt ***rekursiv***,
wenn es Regeln r1, r2, ..., rn gibt mit r1 = rn = r, so daß für alle i
ein Rumpfprädikat von ri im Kopf von r(i+1) steht.

Mächtigkeit von Datalog

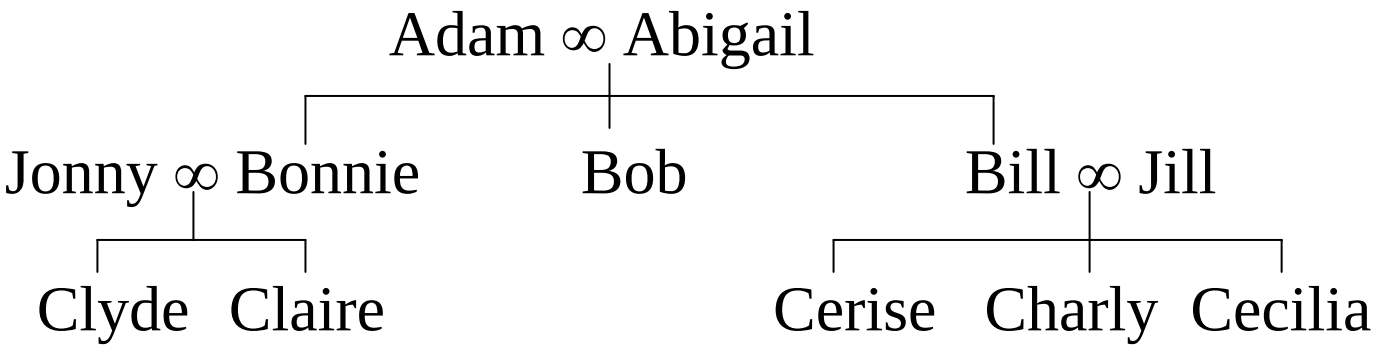
Satz:

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)

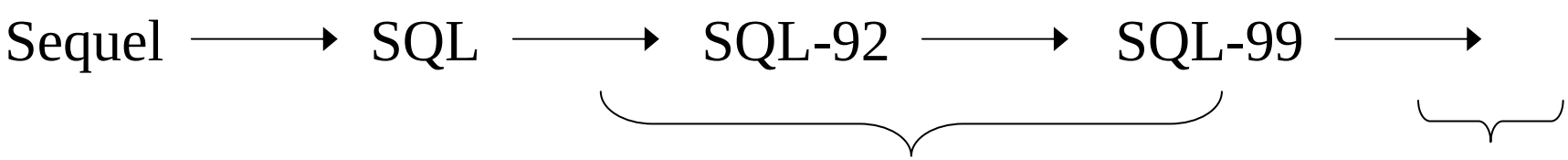
Kapitel 7: Die Datenbanksprache SQL

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 (Literale)
- 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 $L(G) \subseteq \Sigma^* =$
$\{w \in \Sigma^* \mid$ es gibt endliche Ableitung
$S = x0 \rightarrow x1 \rightarrow ... \rightarrow xn=w$ mit $xi \in (V \cup \Sigma)^*$
und $x(i-1) \rightarrow xi$ als Anwendung einer Regel aus P

Exkurs: Syntaxbeschreibung – einfaches Beispiel

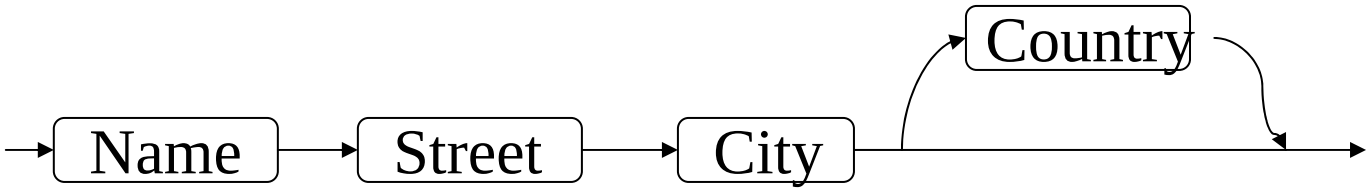
V={Address, Name, Firstname, Lastname, Street, City, Country,
String, Letter, Number, Digit},
Σ={a, b, c, ..., 0, 1, …},
S=Address,
P={Address → Name Street City,
Address → Name Street City Country,
Name → Firstname Lastname,
Name → Letter Lastname,
Firstname → String,
Lastname → String,
Street → String Number,
Country → String,
String → Letter,
String → Letter String,
Number → Digit Number,
Number → ε,
Letter → a, Letter → b, …,
Digit → 0, Digit → 1, …}

Exkurs: Syntaxbeschreibung – EBNF

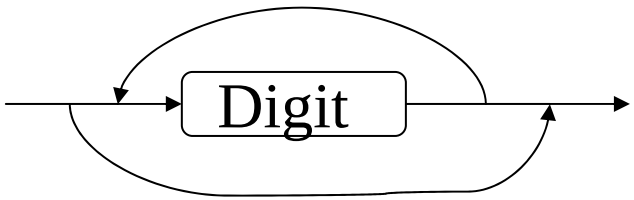
Address ::= Name Street City [Country],
String ::= Letter {Letter …},
Number ::= {Digit …},
Digit ::= 0|1|2|3|4|5|6|7|8|9,
usw.

Exkurs: Syntaxbeschreibung – Syntaxdiagramme

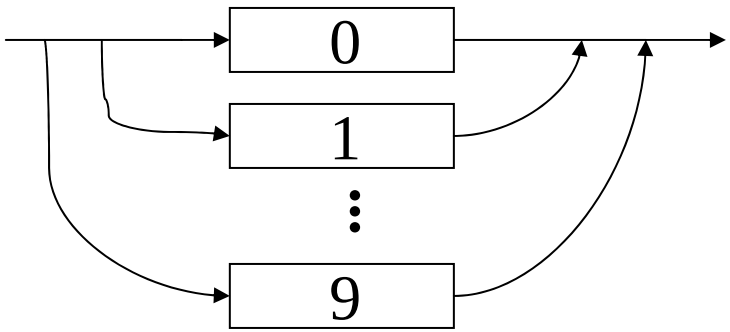
Address ::=



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)

$\rightarrow$ RA: $\pi[A, B, ...] (\sigma [F] (R \times S \times ... \times T \times ...)$

$\rightarrow$ TRK: $\{x.A, y.B, ... \mid x \in R \wedge y \in S \wedge ... \wedge \exists z: z \in T ... \wedge F(x, y, ..., z, ...)\}$

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 Rechnungssume 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 → trc query

von select_block-Konstrukten auf TRK-Anfragen
unter Verwendung der Funktion

sql2trc': sql where clause → trc formula

von search_condition-Konstrukten auf TRK-Formeln.

Abbildung auf TRK (1)

sql2trc [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)

$= \{ri_1.A1, ri_2.A2, ... \mid r1 \in REL1 \wedge r2 \in REL2 \wedge \ ... \wedge rm \in RELm$
$\wedge \exists t1 ... \exists tk (t1 \in TAB1 \wedge ... \wedge tk \in TABk \wedge sql2trc'[F])\}$

sql2trc [select-block1 UNION select-block2]

(mit select-block1: SELECT A1, A2, … FROM REL1 R1, …, RELm Rm,
TAB1 T1, …, TABk Tk WHERE F
und select-block2: SELECT B1, B2, ... FROM SET1 S1, ..., SETm' Sm',
PAR1 P1, ..., PARk' Pk' WHERE G)

$= \{u1, u2, ... \mid$
$(\exists r1 ... \exists rm (u1 = r1.A1 \wedge \ u2 = r2.A2 \wedge \ldots \wedge$
$r1 \in REL1 \wedge r2 \in REL2 \wedge \ ... \wedge rm \in RELm \wedge$
$\exists t1 ... \exists tk \ (t1 \in TAB1 \wedge ... \wedge tk \in TABk \wedge sql2trc'[F]))$
$\vee \ (\exists s1 ... \exists sm' (u1 = s1.B1 \wedge \ u2 = s2.B2 \wedge \ldots \wedge$
$s1 \in SET1 \wedge s2 \in SET2 \wedge \ ... \wedge sm' \in SETm' \wedge$
$\exists p1 ... \exists pk' (p1 \in PAR1 \wedge ... \wedge pk' \in PARk' \wedge sql2trc'[G]))\}$

Abbildung auf TRK (2)

sql2trc' [Ri.Aj θ Tk.Bl] = ri.Aj θ tk.Bl
sql2trc' [Ri.Aj θ c] (mit einer Konstanten c) = ri.Aj θ c
sql2trc' [F AND G] = sql2trc'[F] $\wedge$ sql2trc'[G]
sql2trc' [F OR G] = sql2trc'[F] $\vee$ sql2trc'[G]
sql2trc' [NOT F] = $\neg$ sql2trc'[F]

sql2trc' [Ri.Aj IN subquery]
(so daß subquery die Form SELECT Qk.C
FROM QUELL1 Q1, ..., QUELLm' Qm' WHERE H hat)
= $\exists$ q1 … $\exists$ qm' (qk.C = ri.Aj $\wedge$ q1$\in$QUELL1 $\wedge$ … qm'$\in$QUELLm'
$\wedge$ sql2trc'[H])

Abbildung auf TRK (3)

sql2trc' [Ri.Aj θANY subquery]

$= \exists qk\,(qk \in QUELLk \wedge (\exists q1 \ldots \exists q(k-1)\ \exists q(k+1) \ldots \exists qm'$
$(ri.Aj\ \theta\ qk.C\ \wedge q1 \in QUELL1 \wedge \ldots \wedge q(k-1) \in QUELL(k-1) \wedge$
$q(k+1) \in QUELL(k+1) \wedge \ldots \wedge qm' \in QUELLm' \wedge$ sql2trc'[H])))

sql2trc' [Ri.Aj θALL subquery]

$= \forall qk\,(\,(\,qk \in QUELLk \wedge\ (\exists q1 \ldots \exists q(k-1)\ \exists q(k+1) \ldots \exists qm'$
$(q1 \in QUELL1 \wedge \ldots \wedge q(k-1) \in QUELL(k-1) \wedge$
$q(k+1) \in QUELL(k+1) \wedge \ldots \wedge qm' \in QUELLm' \wedge$ sql2trc'[H])))
$\Rightarrow (\ ri.Aj\ \theta\ qk.C\ )\ )$

sql2trc' [EXISTS subquery]

(so daß subquery die Form SELECT C1, C2, ...
FROM QUELL1 Q1, ..., QUELLm' Qm' WHERE H hat)

$= \exists q1 \ldots \exists qm'\ (q1 \in QUELL1 \wedge \ldots \wedge qm' \in QUELLm'$
$\wedge$ 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]

$= \{k.KNr, k.Name \mid k \in Kunden \wedge sql2trc'[K.Ort=... AND NOT EXISTS \ldots]\}$

$= \{k.KNr, k.Name \mid k \in Kunden \wedge k.Ort=... \wedge \neg\ sql2trc'[EXISTS ...]\}$

$= \{k.KNr, k.Name \mid k \in Kunden \wedge k.Ort=... \wedge \neg\ (\exists\ p\ \exists\ b\ (\ p \in Produkte \wedge b \in Bestellungen \wedge sql2trc'[P.Preis>... AND ... AND ...]\ ))\ \}$

$= \{k.KNr, k.Name \mid k \in Kunden \wedge k.Ort=... \wedge \neg\ (\exists\ p\ \exists\ b\ (\ p \in Produkte \wedge b \in Bestellungen \wedge p.Preis>... \wedge p.PNr=b.PNr \wedge b.KNr=k.KNr\ ))\ \}$

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 → ra query

von select_block-Konstrukten auf RA-Anfragen
unter Verwendung der Funktion

sql2ra': sql where clause × ra query → ra query

von search_condition-Konstrukten auf RA-Ausdrücke
sowie der Hilfsfunktion

sql2ra-: sql where clause × ra query → ra query

mit sql2ra- [F](E) = E - π[sch(E)] (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;
π[Ri_1.A1, Ri_2.A2, ...] (sql2ra'[F](R1 × … × Rm × T1 × … × 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] ∪ sql2ra[select-block2]
(mit ggf. notwendigen Umbenennungen von Attributen)

Abbildung auf RA (2)

sql2ra' [Ri.Aj θ Tk.Bl] (E) = σ[Ri.Aj θ Tk.Bl](E)
sql2ra' [Ri.Aj θ c] (E) (mit einer Konstanten c) = σ[Ri.Aj θ c](E)
sql2ra' [F AND G] (E) = sql2ra'[F](E) ∩ sql2tra'[G](E)
sql2ra' [F OR G] (E) = sql2ra'[F](E) ∪ sql2ra'[G](E)
sql2ra' [NOT F] (E) = sql2ra-[F](E) = E - π[sch(E)] (sql2ra'[F](E))

sql2ra' [Ri.Aj IN subquery](E)
(so daß subquery die Form SELECT Qk.C
FROM QUELL1 Q1, ..., QUELLm' Qm' WHERE H hat)
= Q1 := QUELL1; … Qm' := QUELLm';
π[sch(E)] (sql2ra'[H] (σ[Ri.Aj = Qk.C] (E × Q1 × … × Qm')))

Abbildung auf RA (3)

sql2ra' [Ri.Aj θANY subquery] (E)
= Q1 := QUELL1; … Qm' := QUELLm';
π[sch(E)] (sql2ra'[H] (σ[Ri.Aj θ Qk.C] (E × Q1 × … × Qm')))

sql2ra' [Ri.Aj θALL subquery] (E)
= sql2ra- [Ri.Aj θ'ANY subquery](E)
mit θ' gleich ≠ für θ gleich =, = für ≠, < für ≥, > für ≤, usw.
= E - π[sch(E)] (sql2ra'[H] (σ[Ri.Aj θ' Qk.C] (E × Q1 × … × Qm')))

sql2ra' [EXISTS subquery]
(so daß subquery die Form SELECT C1, C2, ...
FROM QUELL1 Q1, ..., QUELLm' Qm' WHERE H hat)
= Q1 := QUELL1; … Qm' := QUELLm';
π[sch(E)] (sql2ra'[H] (E × Q1 × … × 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]
= K := Kunden; B := Bestellungen; P := Produkte;
π[K.KNr, K.Name] (sql2ra’[K.Ort=… AND NOT EXISTS …] (K))
= … π[K.KNr, K.Name] (σ[Ort=…](K) $\cap$ sql2ra’[NOT EXISTS …](K))
= … π[K.KNr, K.Name] (σ[Ort=…](K) $\cap$ sql2ra- [EXISTS ...](K))
= … π[K.KNr, K.Name] (σ[Ort=…](K)
$\cap$ (K - π[sch(E)](sql2ra’ [EXISTS ...](K))))
= ... π[K.KNr, K.Name] (σ[Ort=…](K)
$\cap$ (K - π[sch(K)](sql2ra’[P.Preis...AND…AND…](K $\times$ P $\times$ B)))
= ... π[K.KNr, K.Name] (σ[Ort=…](K)
$\cap$ (K - π[sch(K)](σ[P.Preis>100](K $\times$ P $\times$ B) $\cap$
σ[P.PNr=B.PNr](K $\times$ P $\times$ B) $\cap$
σ[P.PNr=B.PNr](K $\times$ P $\times$ B))))

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
```

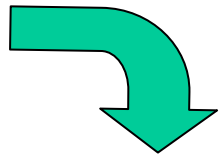
Aggregation mit Gruppierung

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

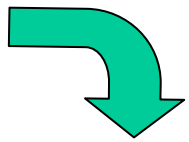
Aggregation mit Gruppierung und Gruppenauswahl

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

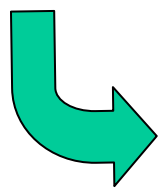
Zwischenresultat 1:

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):

sql2ra [SELECT A', f(B) FROM … WHERE … GROUP BY A]
= R(A, F) := γ_+[A,B,f] (sql2ra[SELECT A,B FROM … WHERE …]);
π_+ [A', F] (R)

Abbildung von

SELECT A', f(B) FROM ... WHERE ...
GROUP BY A HAVING cond(A, g(C))

auf RA-Programm:

Q(A,B,C) := sql2ra[SELECT … FROM … WHERE …]; R (A,F) := $\emptyset$;
for each x $\in$ π [A](Q) do
Gx := π_+ [A,B,C] (σ_+ [A=x] (Q))
if sql2ra'[cond](Gx) $\neq \emptyset$ then y := f (π_+ [B](Gx)); R := R $\cup_+$ {(x,y)} fi;
od;
π_+[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 Java

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;
        Produkterelationentyp = RELATION OF Produktrecordtyp;
VAR     Produkte: PERSISTENT Produkterelationentyp;
        Ergebnis: Produkterelationentyp;
        x: Produktrecordtyp;

Ergebnis :=        SELECT * FROM Produkte WHERE ...;
FOR EACH x IN Ergebnis  DO ... END;
```

8.1 Embedded SQL (ESQL)

Kernproblem:

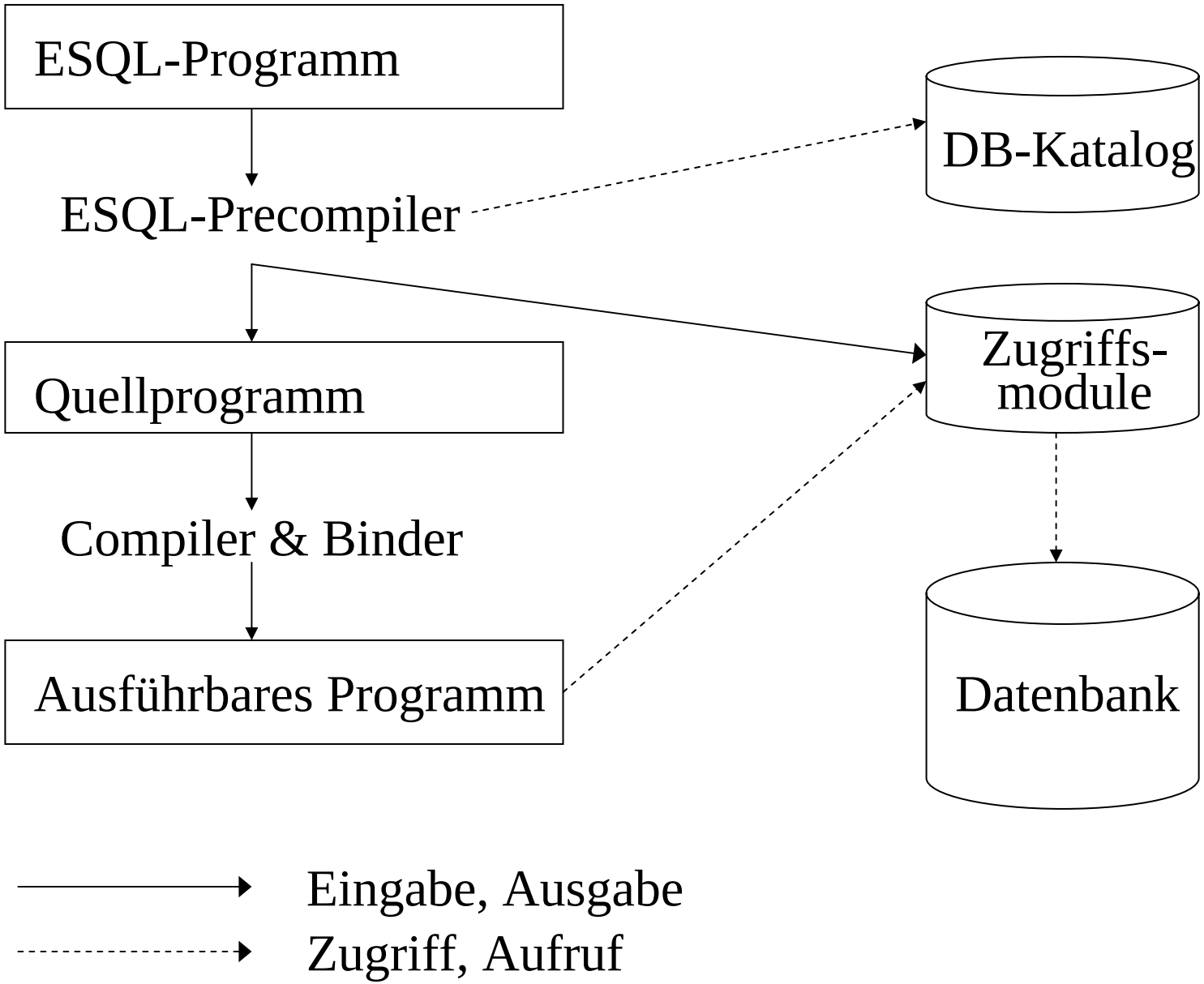
Abbildung von Tupelmengen auf
Datentypen der Wirtsprogrammiersprache

Lösungsansatz:

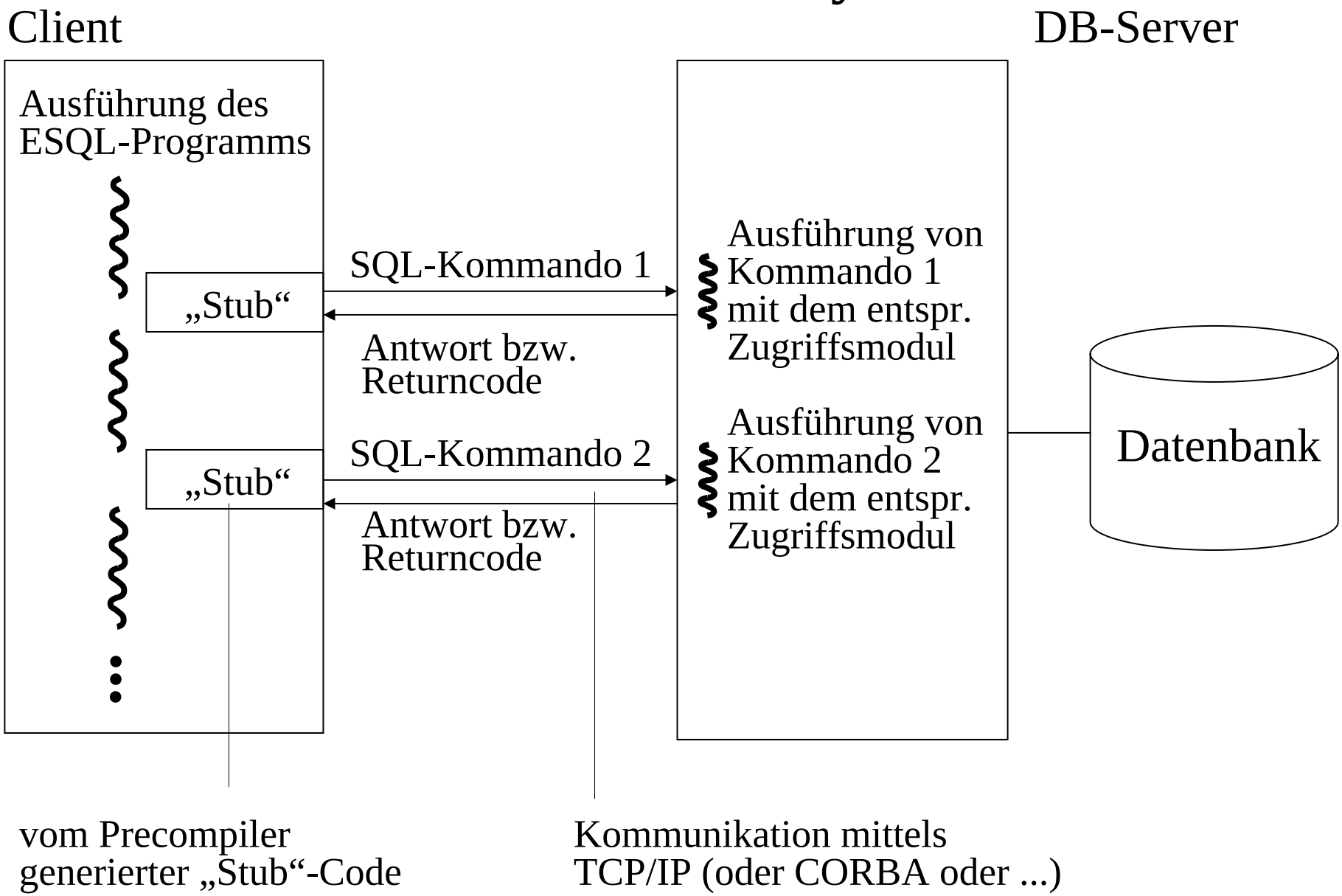
1) Abbildung von Tupeln bzw. Attributen auf die
Datentypen der Wirtsprogrammiersprache
→ ***Wirtsprogrammvariablen (Host Variables)***
2) Iteratoren (Schleifen) zur Verarbeitung von Tupelmengen
→ ***Cursor-Konzept***

ein universeller Lösungsansatz
für alle Wirtsprogrammiersprachen !

Architektur des ESQL-Precompilers



Laufzeitarchitektur für ESQL-Programme in Client-Server-System



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 vo 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 ("%.70s\n", sqlca.sqlerrm.sqlerrmc);
    exit (-1);
} /*main*/
```

Stored Procedures in 4GL PL/SQL

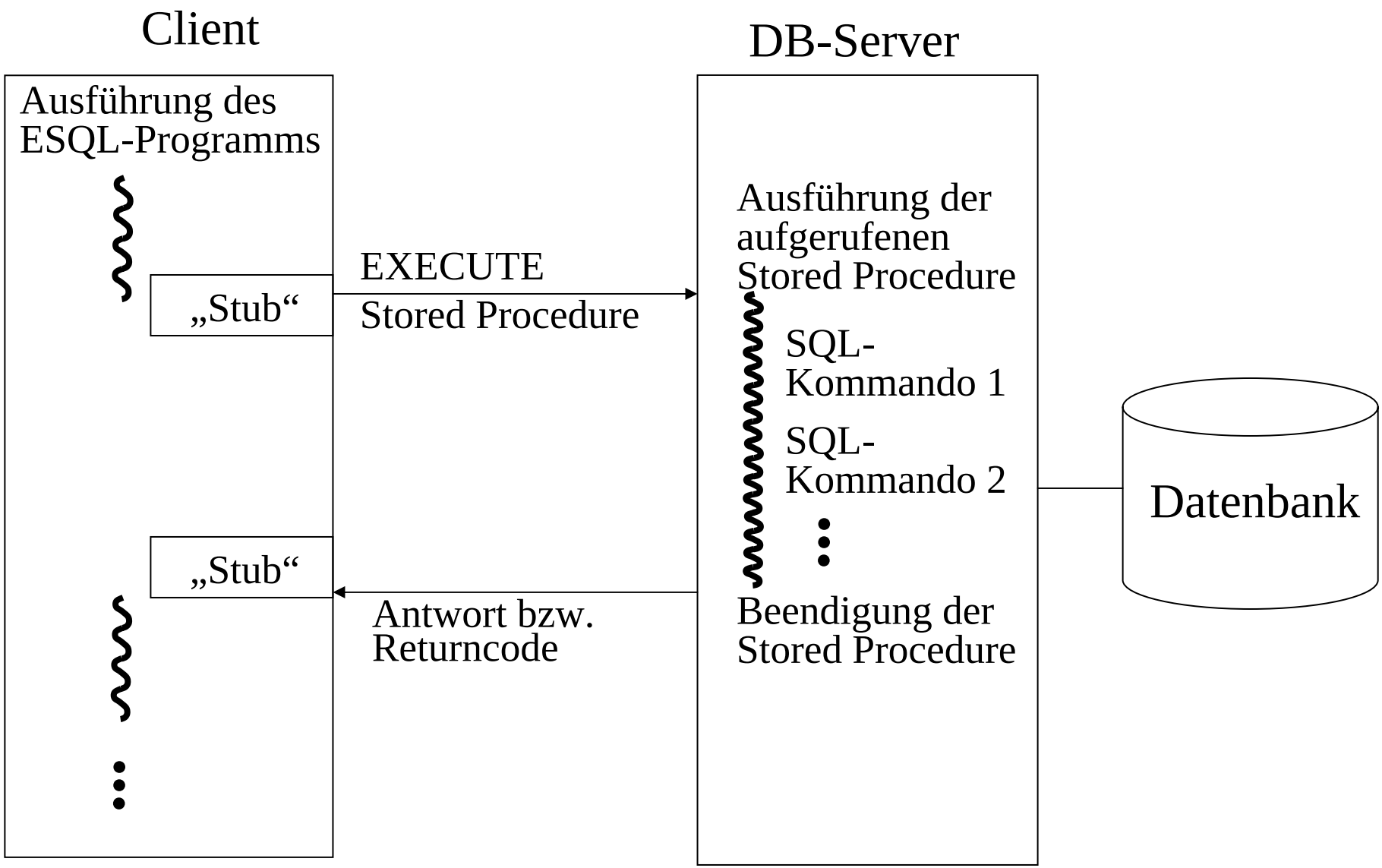
```
CREATE PROCEDURE Lager_Auffuellen AS
DECLARE ...;
BEGIN
DECLARE CURSOR Knappe_Produkte IS
    SELECT PNr, Vorrat FROM Produkte WHERE Vorrat < 100;
KP Knappe_Produkte%ROWTYPE;
BEGIN
    OPEN Knappe_Produkte;
    LOOP
        FETCH Knappe_Produkte INTO KP;
        EXIT WHEN Knappe_Produkte%NOTFOUND;
        ...
        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;
```

Prozedur-deklaration

```
EXECUTE Lager_Auffuellen;
```

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

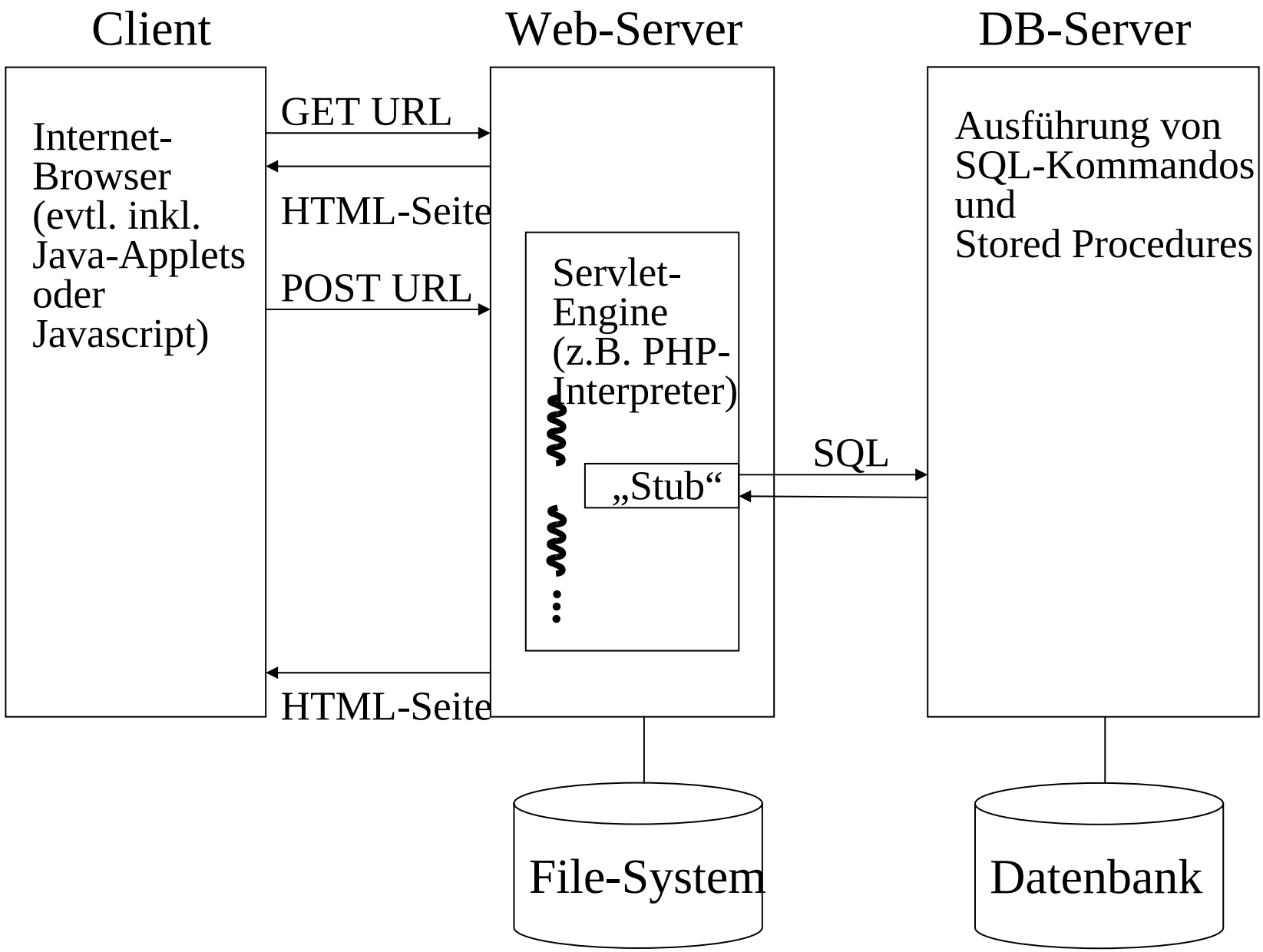
3-Schichten-Architektur (3-Tier Architecture):

- 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-Identifier

```
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 sql2trc‘[searchcond] = F(t1, ..., tn) mit freien Variablen t1, ..., tn:

$$\forall t1 \ldots \forall tn \; (\; (\; t1 \in R1 \wedge \ldots \wedge tn \in Rn \;) \Rightarrow F(t1, \ldots, tn) \;) \quad \Big\} \; Ij$$

Semantik der Datenbank aus logischer Sicht: $H \wedge I1 \wedge \ldots \wedge Im$
mit elementarer Formel (Fakten) H und Integritätsbed. I1, ..., Im

Mögliche Implementierung:
bei potentieller Verletzung von Ij 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):

$$\forall t \, (t \in R \Rightarrow ((t.A1=\omega \wedge \ldots \wedge t.Am=\omega) \vee (\exists t1 \ldots \exists tm \, (t1 \in R1 \wedge \ldots \wedge tm \in Rm \wedge t1.B1=t.A1 \wedge \ldots \wedge tm.Bm=t.Am))))$$

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			1 Null		
1002			2		
1003			1 Null		
...					

Bestellposten

BestNr	PNr	Menge
1001	1 Null	
1002	2	
1003	1 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 Integritätsssicherung, 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]
= π[A1, ..., Am] (sql2ra' [F] (TAB1 × … × TABk × sql2ra[viewquery]))

Beispiel:
sql2ra [SELECT * FROM SuperBestellungsInfo WHERE Rabatt>0.3]
= σ[Rabatt>0.3] (SuperBestellungsInfo)
= σ[Rabatt>0.3] (σ[Summe>10000.0] (BestellungsInfo))
= σ[Rabatt>0.3] (σ[Summe>10000.0]
(π[BestNr, ...] (Kunden |x| Bestellungen |x| 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)

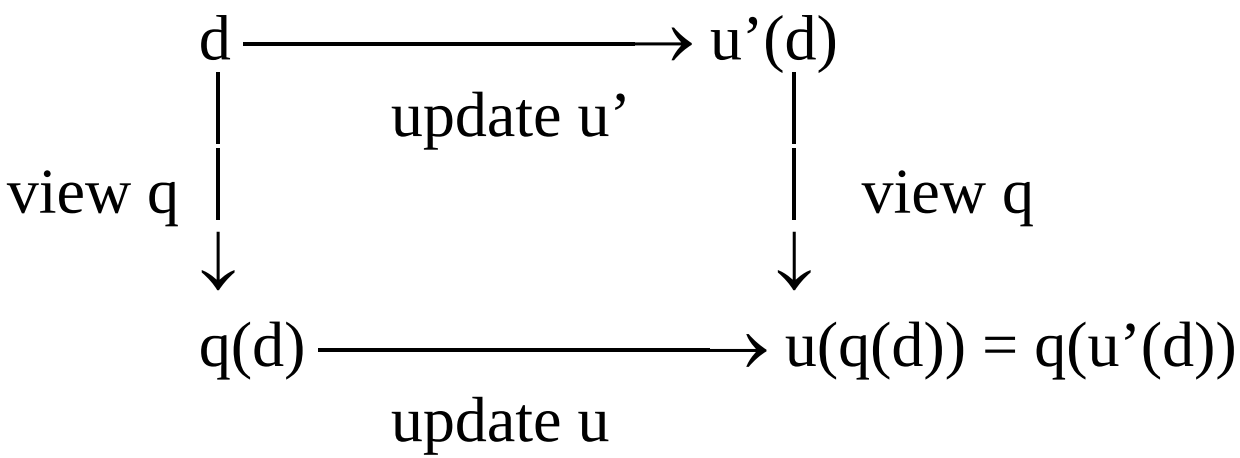
oder einfacher:

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 BestellungsKurzInfo (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 BestellungsKurzInfo 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

Kapitel 10: Objektorientierte Datenbanksprachen und Erweiterungen von SQL

Von relationalen zu objektorientierten DBS

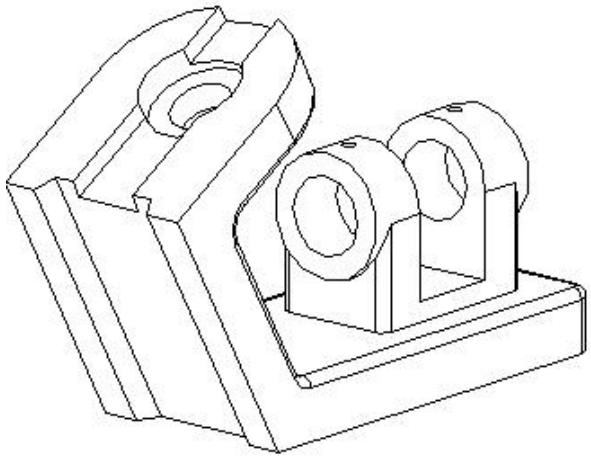
Schwachpunkte relationaler DBS
bei technisch-wissenschaftlichen Daten, z.B. bei:

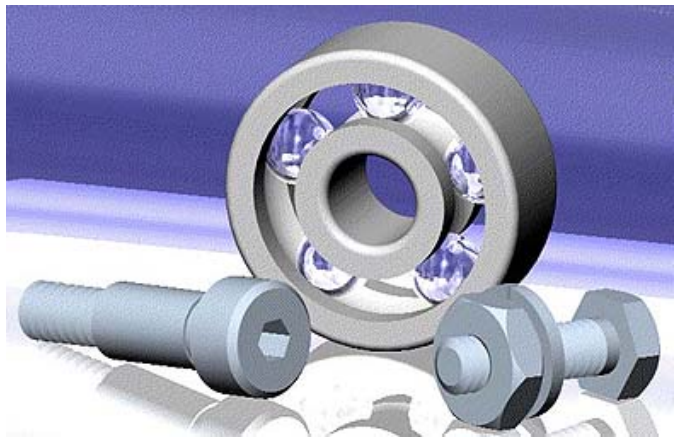
- CAD
- Geowissenschaften
- Desktop Publishing

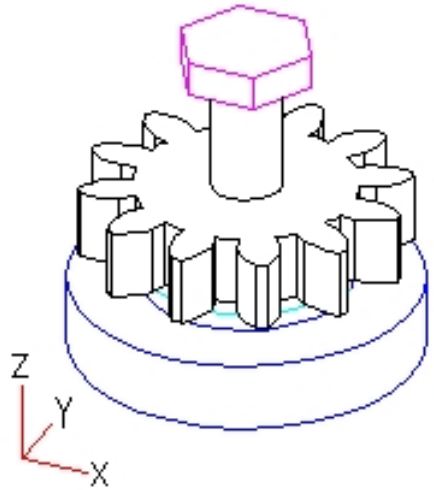
Forderungen an „postrelationale“ DBS:

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

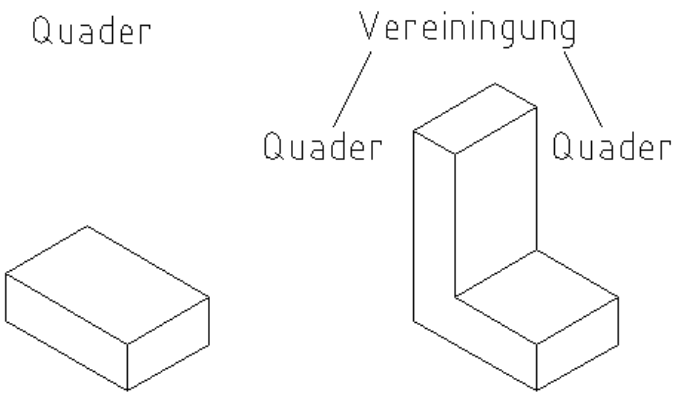
Beispiel 1: CAD (1)

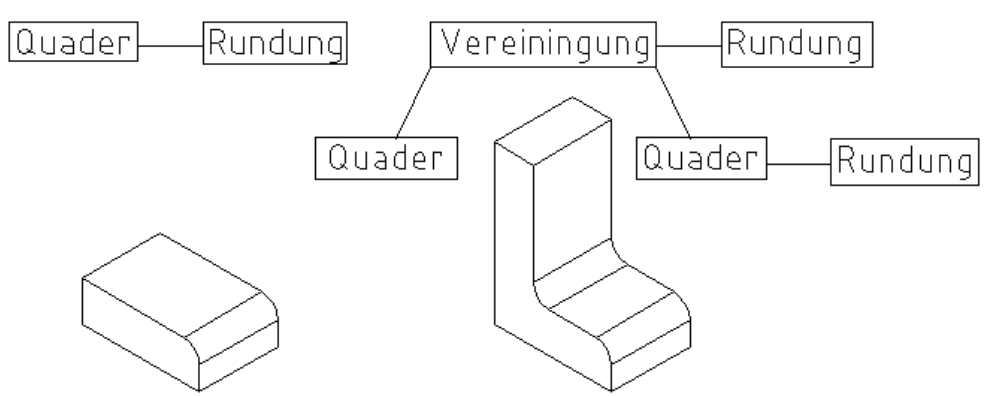




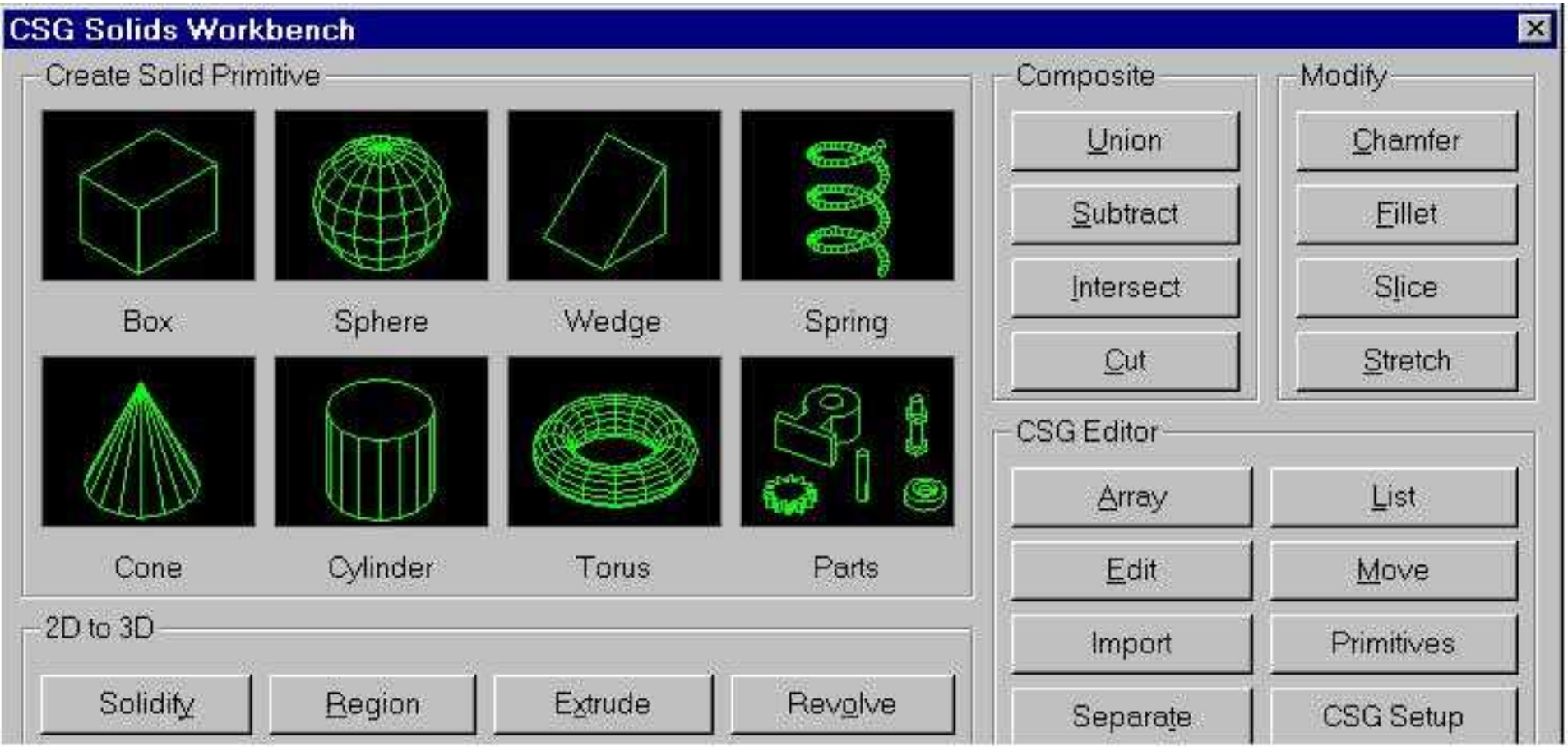


CSG-Modell (Constructive Solid Geometry):

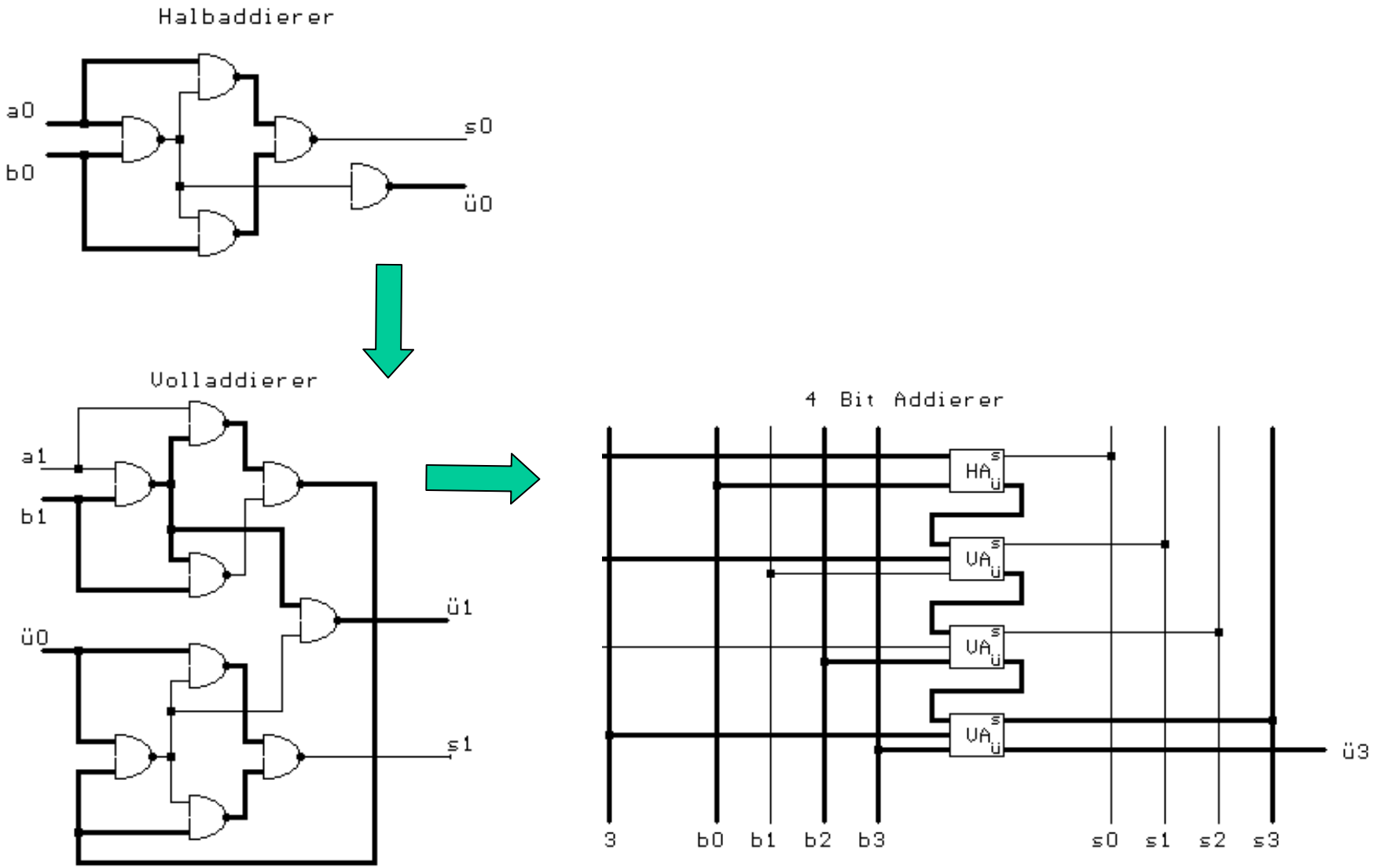




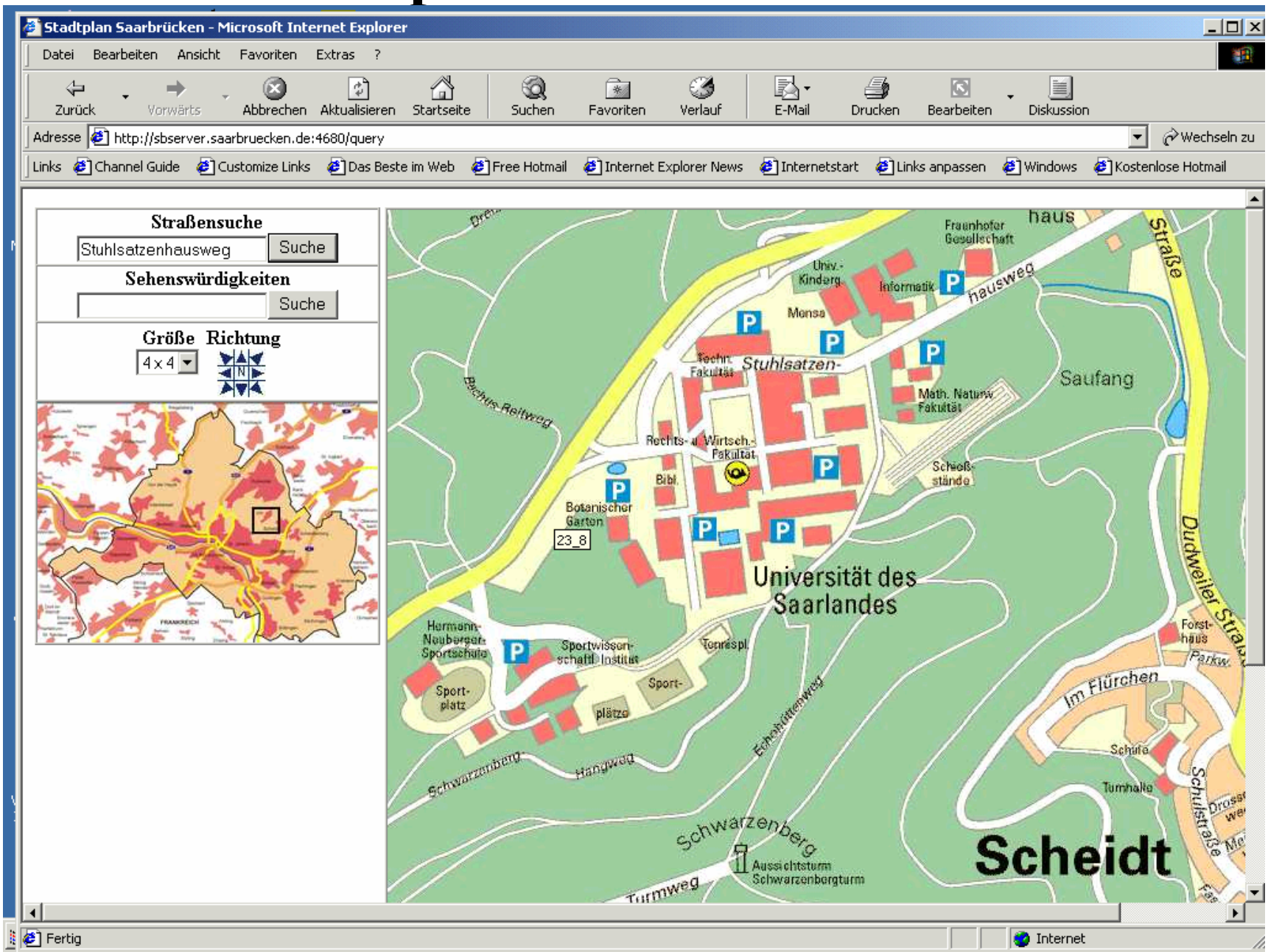
Beispiel 1: CAD (2)



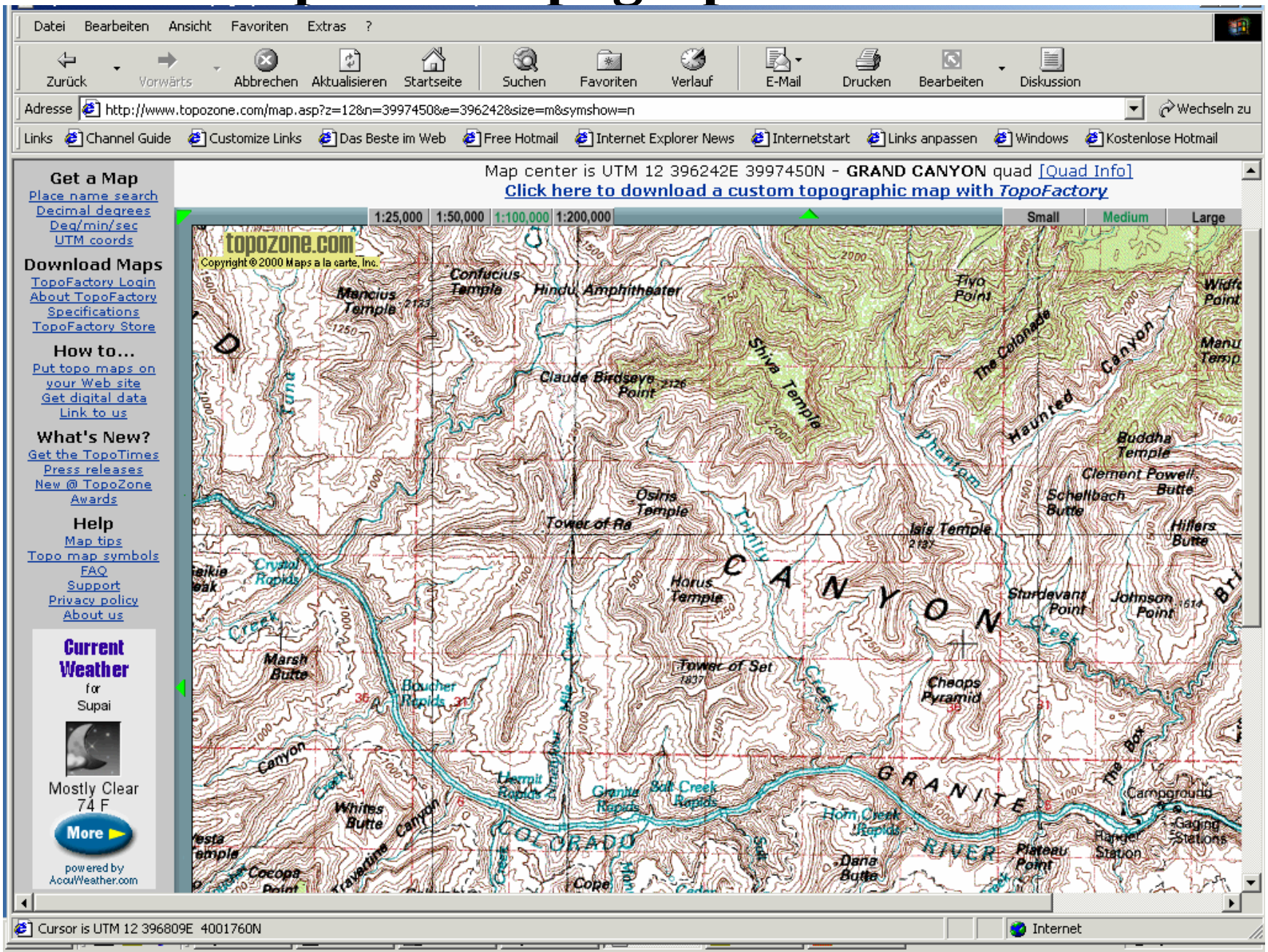
Beispiel 2: Schaltkreisentwurf



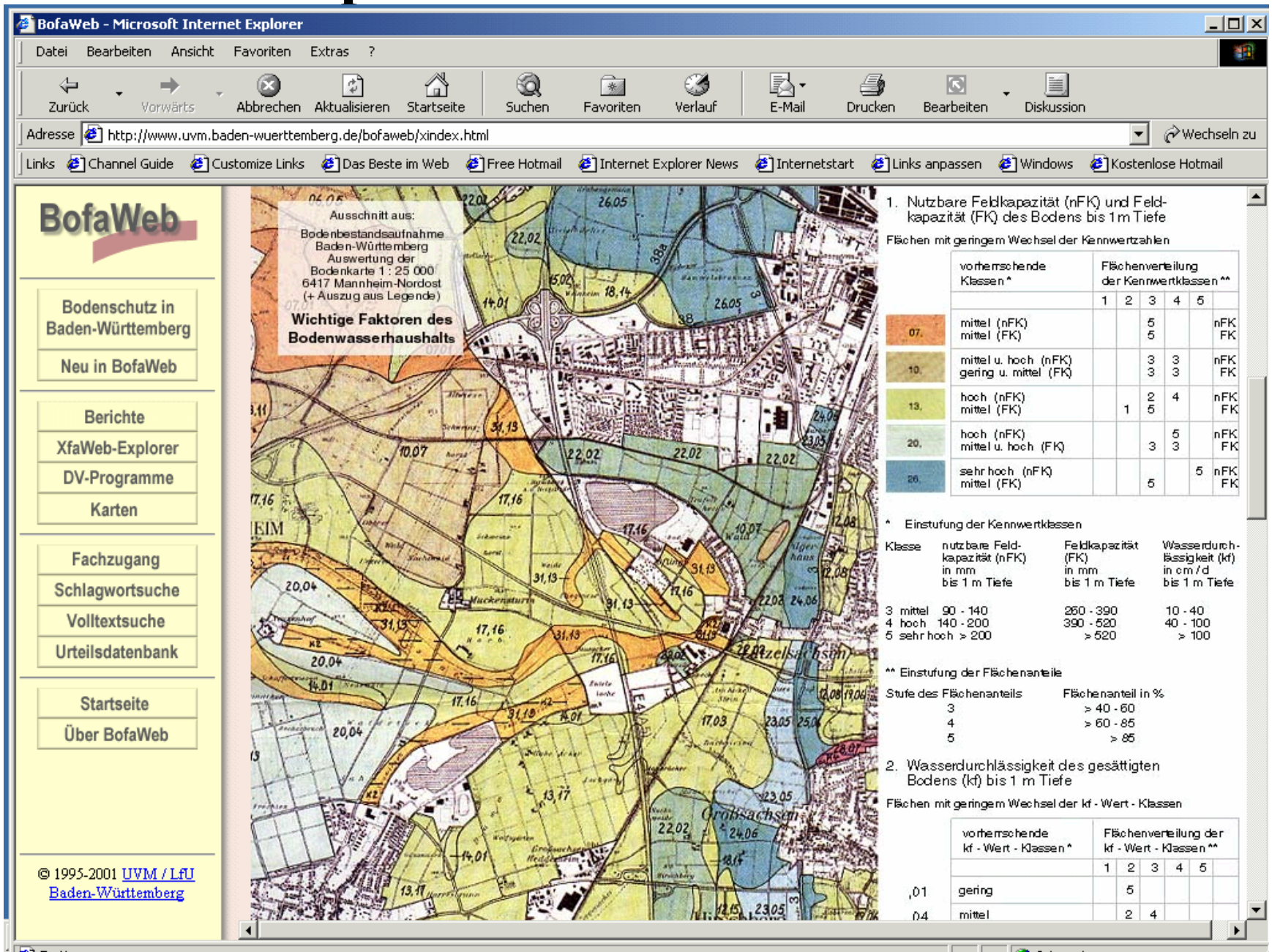
Beispiel 3: Straßenkarten



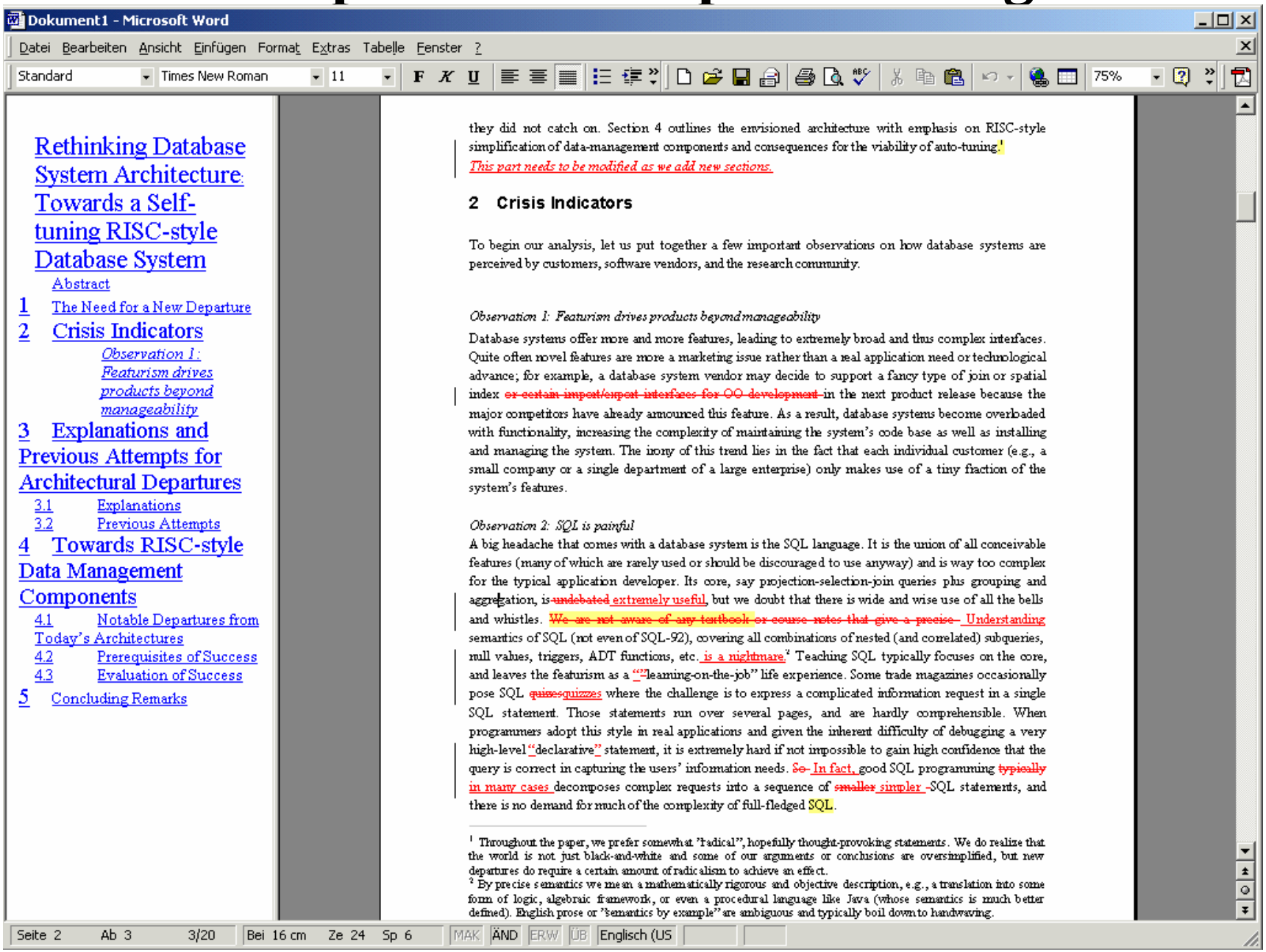
Beispiel 4: Topographische Karten



Beispiel 5: Thematische Karten



Beispiel 6: Desktop Publishing

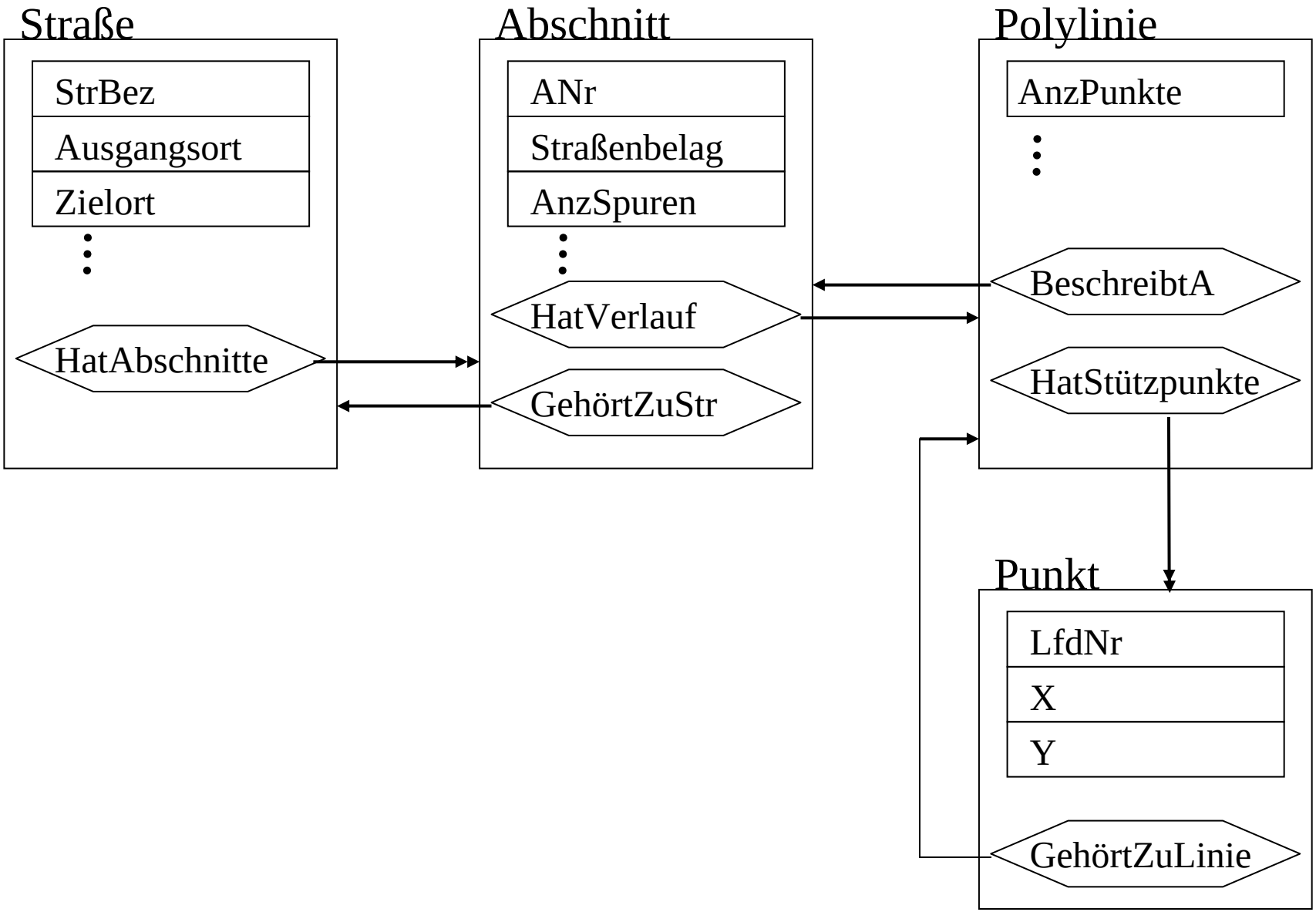


10.2.1 Komplexe Objekte

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

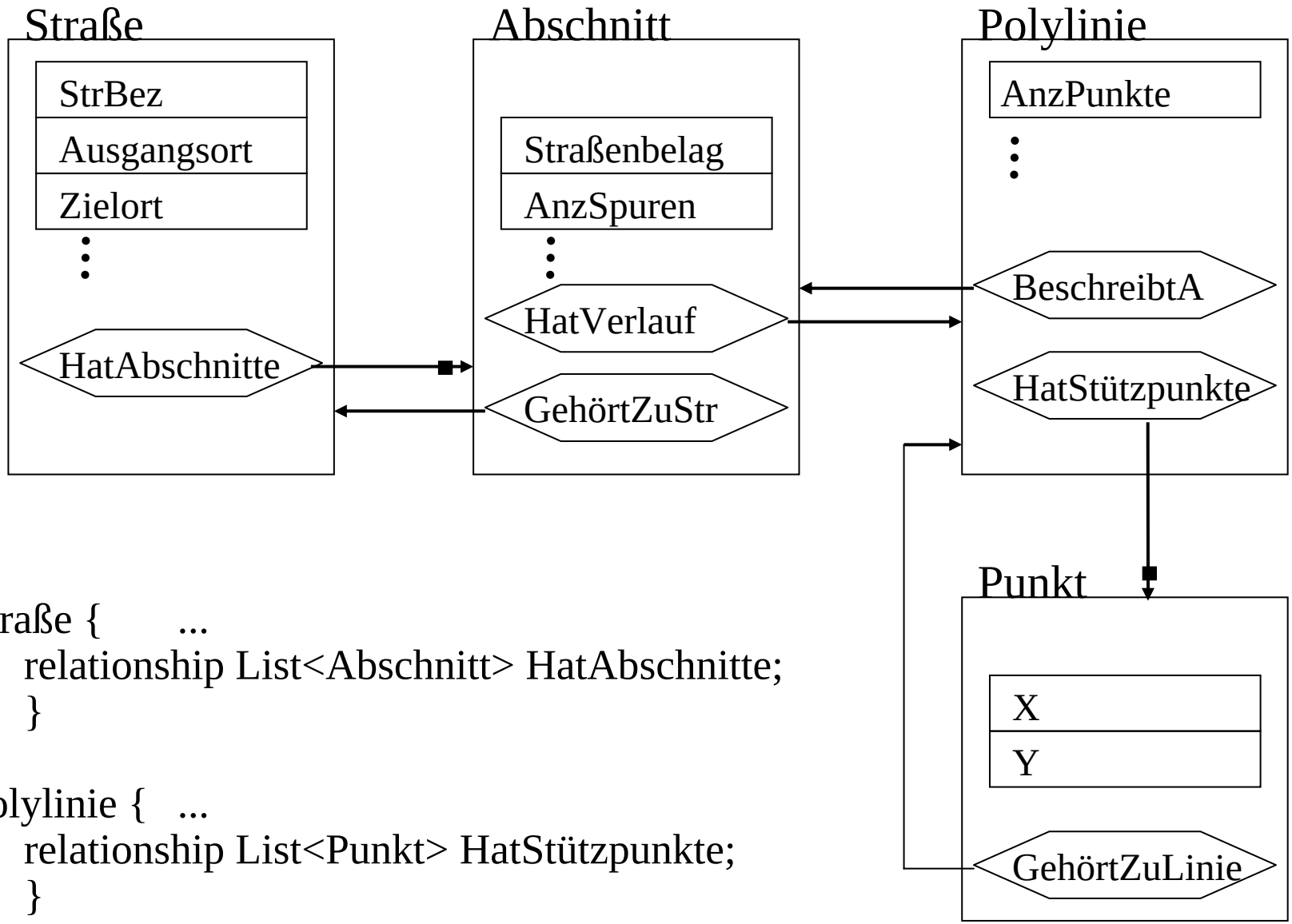
Beispiel – Alternative 1



Beispiel – Alternative 1 (textuell)

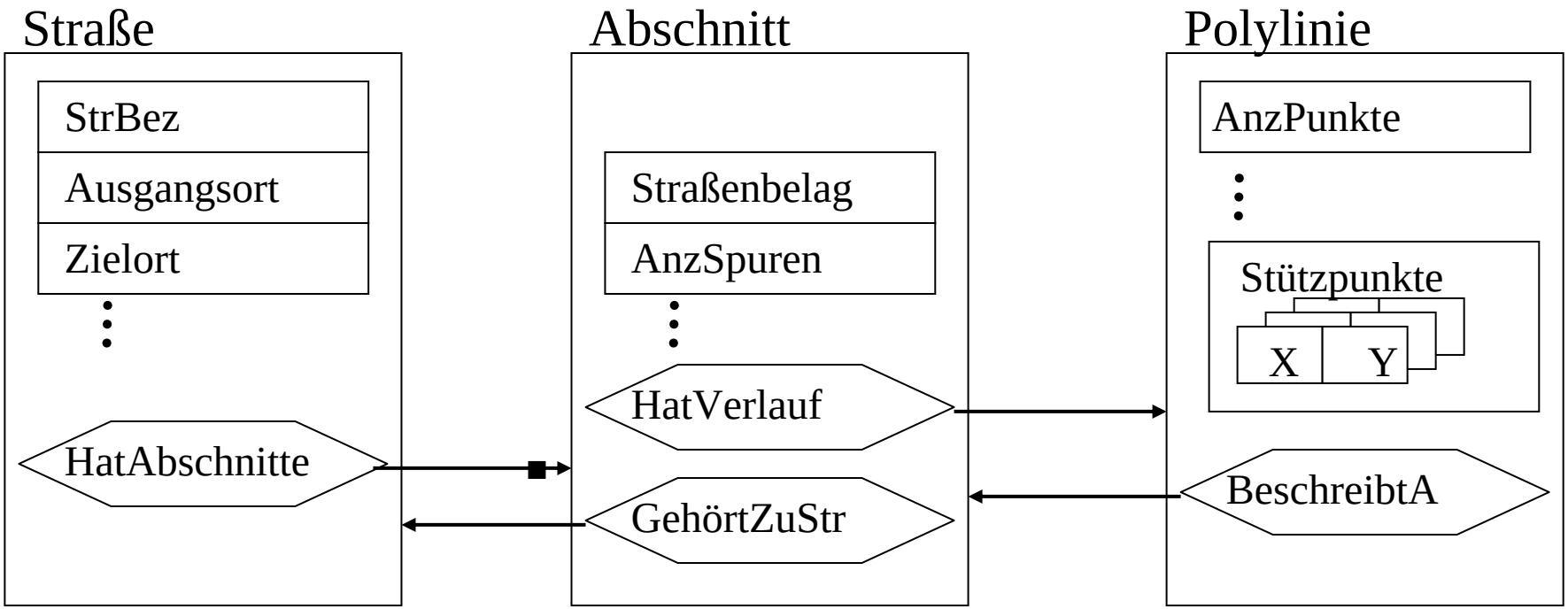
```
class Straße { extent Straßen;
        key StrBez;
        attribute String StrBez;
        attribute String Ausgangsort;
        attribute String Zielort;
        ...
        relationship Set<Abschnitt> HatAbschnitte;
        }
class Abschnitt { extent Abschnitte;
        key (GehörtZuStr.StrBez,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

Beispiel – Alternative 2



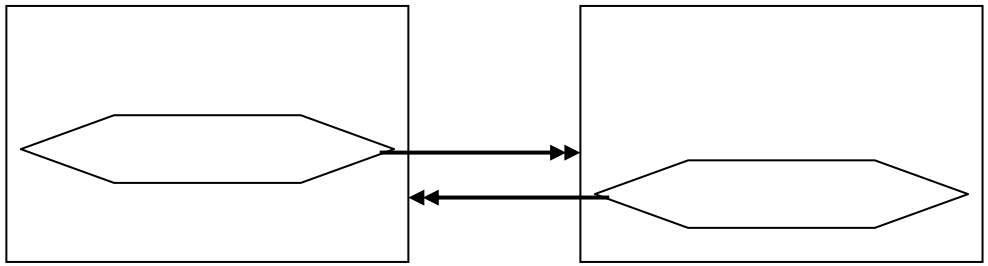

```
class Straße {      ...
        relationship List<Abschnitt> HatAbschnitte;
        }
...
class Polylinie {   ...
        relationship List<Punkt> HatStützpunkte;
        }
...
```

Beispiel – Alternative 3

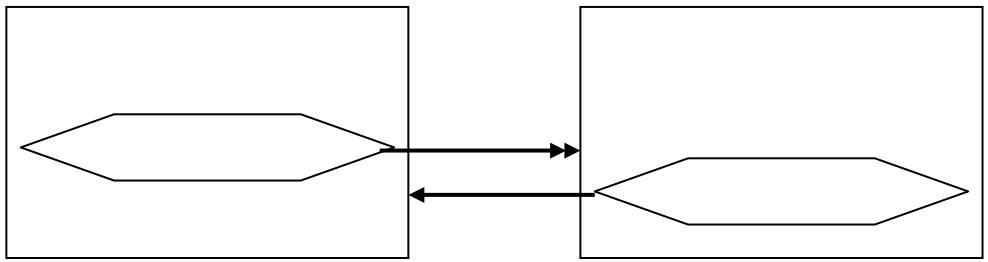



```
...
class Polylinie {   ...
        attribute Integer AnzPunkte;
        attribute List<Struct<X: Real; Y: Real>> Stützpunkte;
        }
```

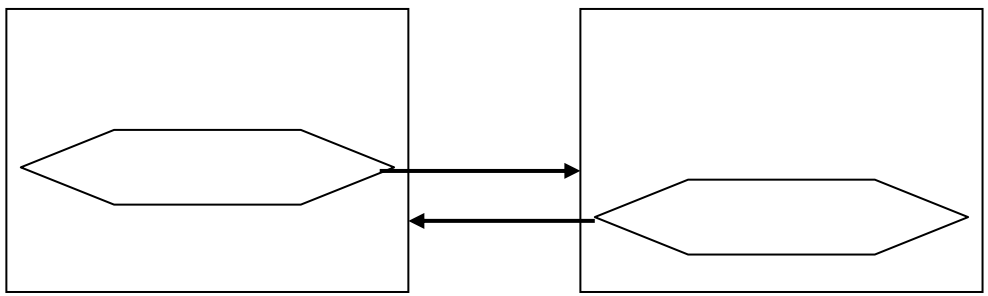
Verschiedene Arten von Relationships



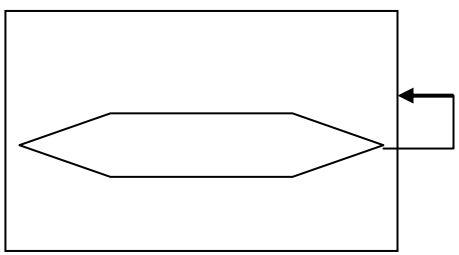
M:N-Relationship



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 denselben 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 $T1 \times T2 \times ... \times Tn \rightarrow T(n+1)$ und
Prozeduren mit Seiteneffekten auf den Objektzustand.
Ein Methodenaufruf x.m(...) hat das Objekt x als impliziten Parameter.

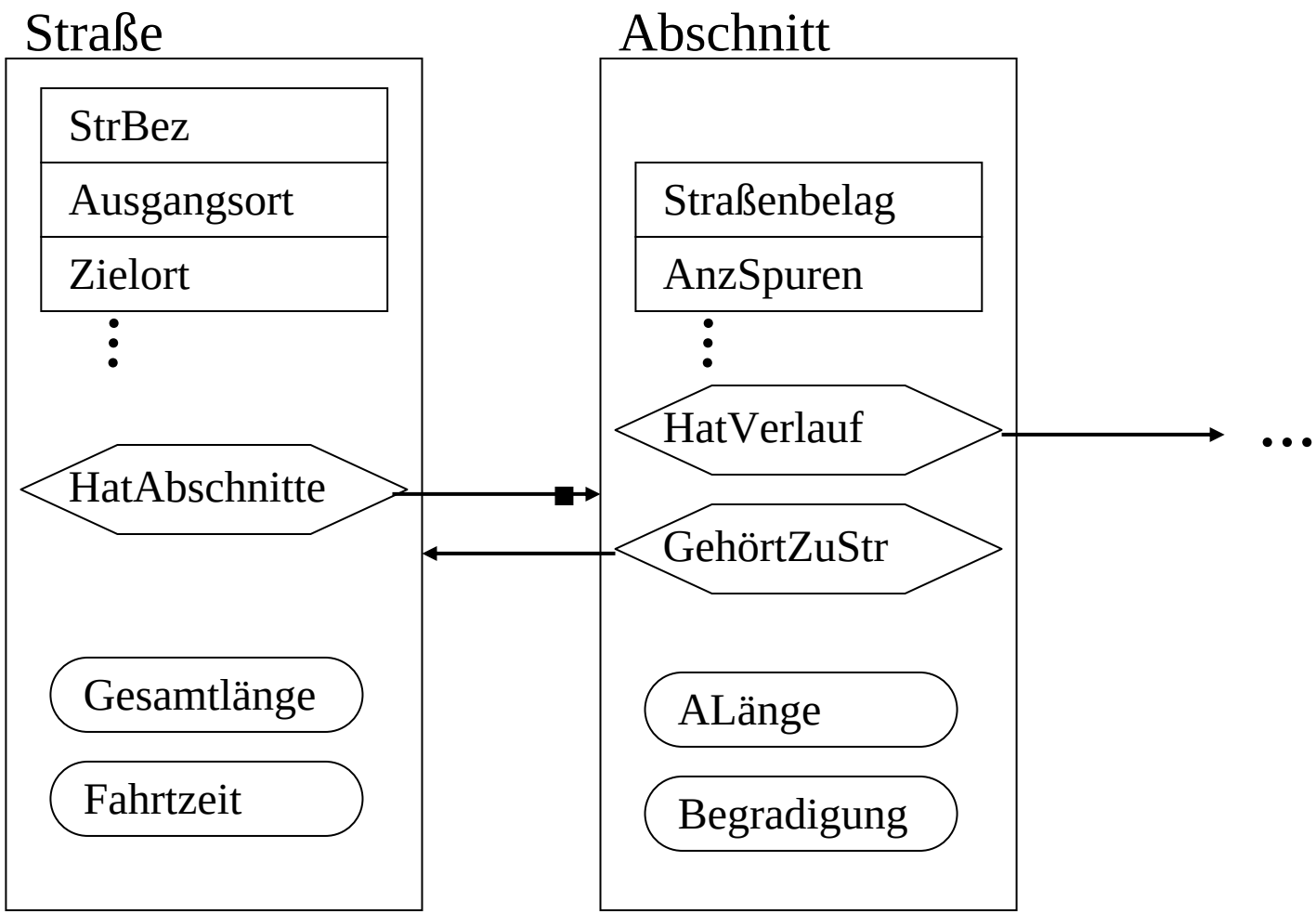
Die Gesamtheit der zu einer Klasse definierten – öffentlich sichtbaren – Methoden bildet die *Schnittstelle eines Abstrakten Datentyps (ADT)*, und jedes Objekt der Klasse ist dann eine Instanz dieses ADTs.
Attribute (d.h. der konkrete Objektzustand) können als privat definiert werden und sind dann nicht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

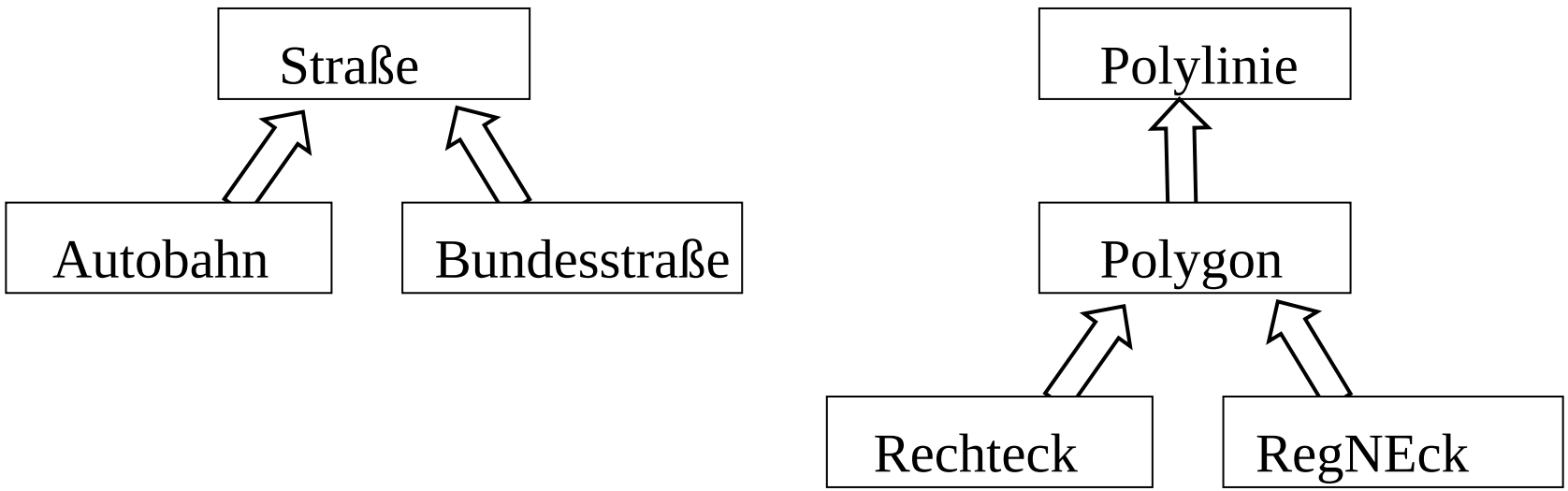


10.2.3 Vererbung

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

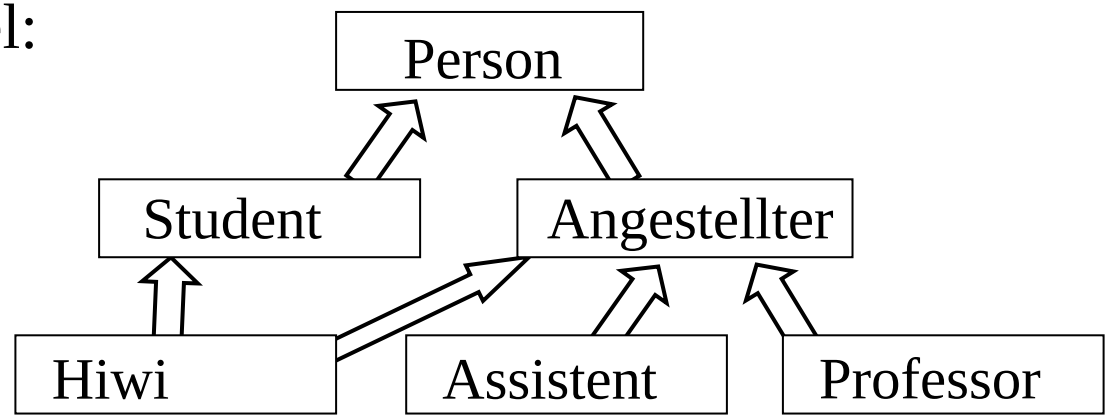
```
class Bundesstraße: Straße { extent Bundesstraßen;
        attribute Integer Verkehrsdichte; }
class Autobahn: Straße { extent Autobahnen;
        attribute Integer Mindesttempo;
        attribute Array<Integer> VerkehrsdichteProTag;
        relationship Set<Punkt> Auffahrten; }

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:

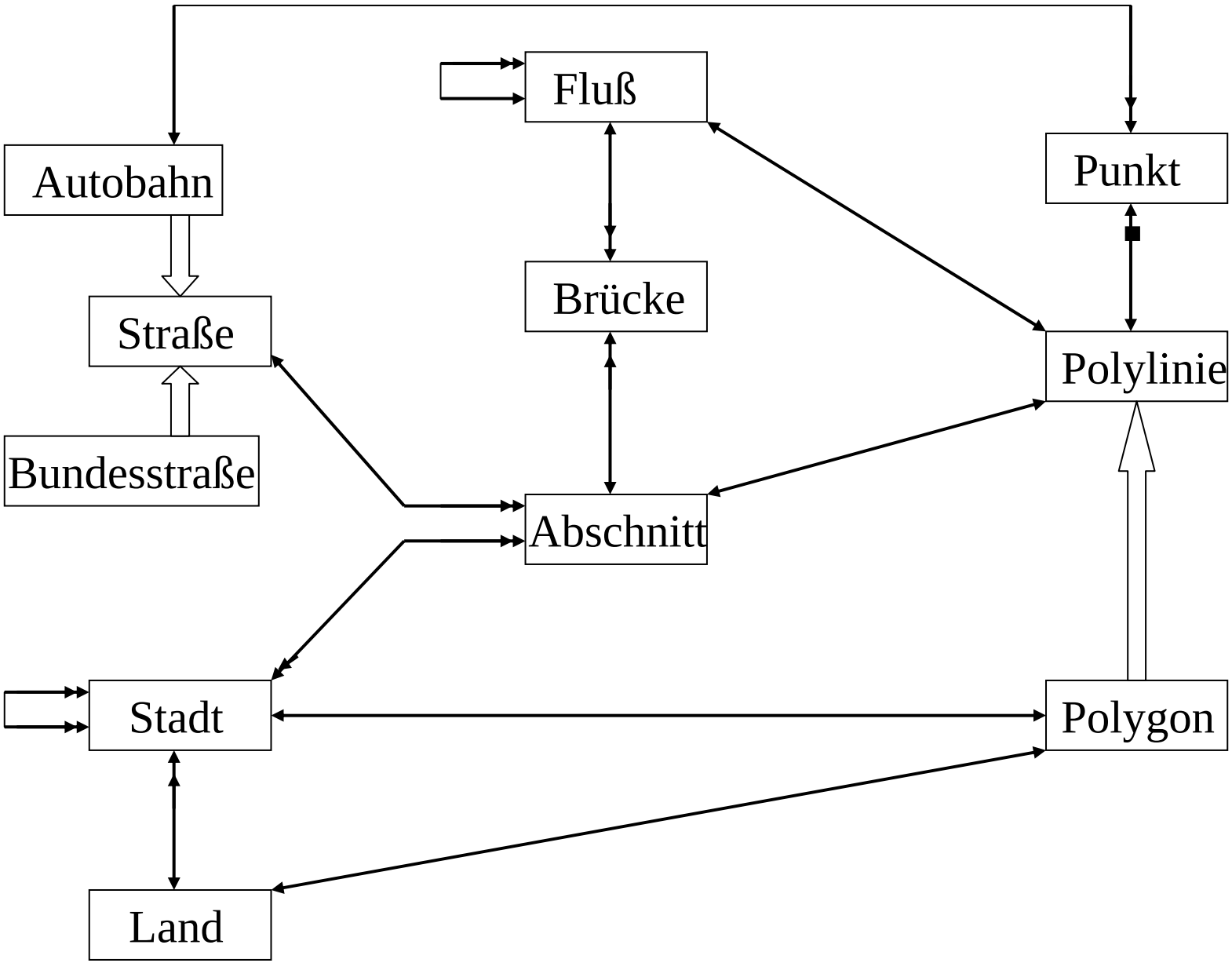
1) Fläche ( ) der Subklasse „Polygone mit Löchern“
2) Fläche ( ) der Subklassen „Rechtecke“ und „RegNEcke“
3) SchneidetBox (in Rechteck) der Subklasse „Polygone“

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
  $\{tn.A, ... \mid ... \wedge \exists\, t0\, \exists\, t1 ... \exists\, t(n-1)$
  $(t0.R1 = t1.OID \wedge t1.R2 = t2.OID \wedge ... \wedge t(n-1).Rn = tn.OID)\}$

  C0.R1.R2. ... .Rn.A θ wert in Where-Klausel entspricht
  $\exists\, t0\, \exists\, t1 ... \exists\, t(n-1)\ (t0.R1 = t1.OID \wedge ... \wedge t(n-1).Rn = tn.OID \wedge tn.A\ \theta$ wert)

- Subqueries auch in Select- und From-Klausel: konstruieren geschachtelte Ergebnisse (z.B. vom Typ Set<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
```

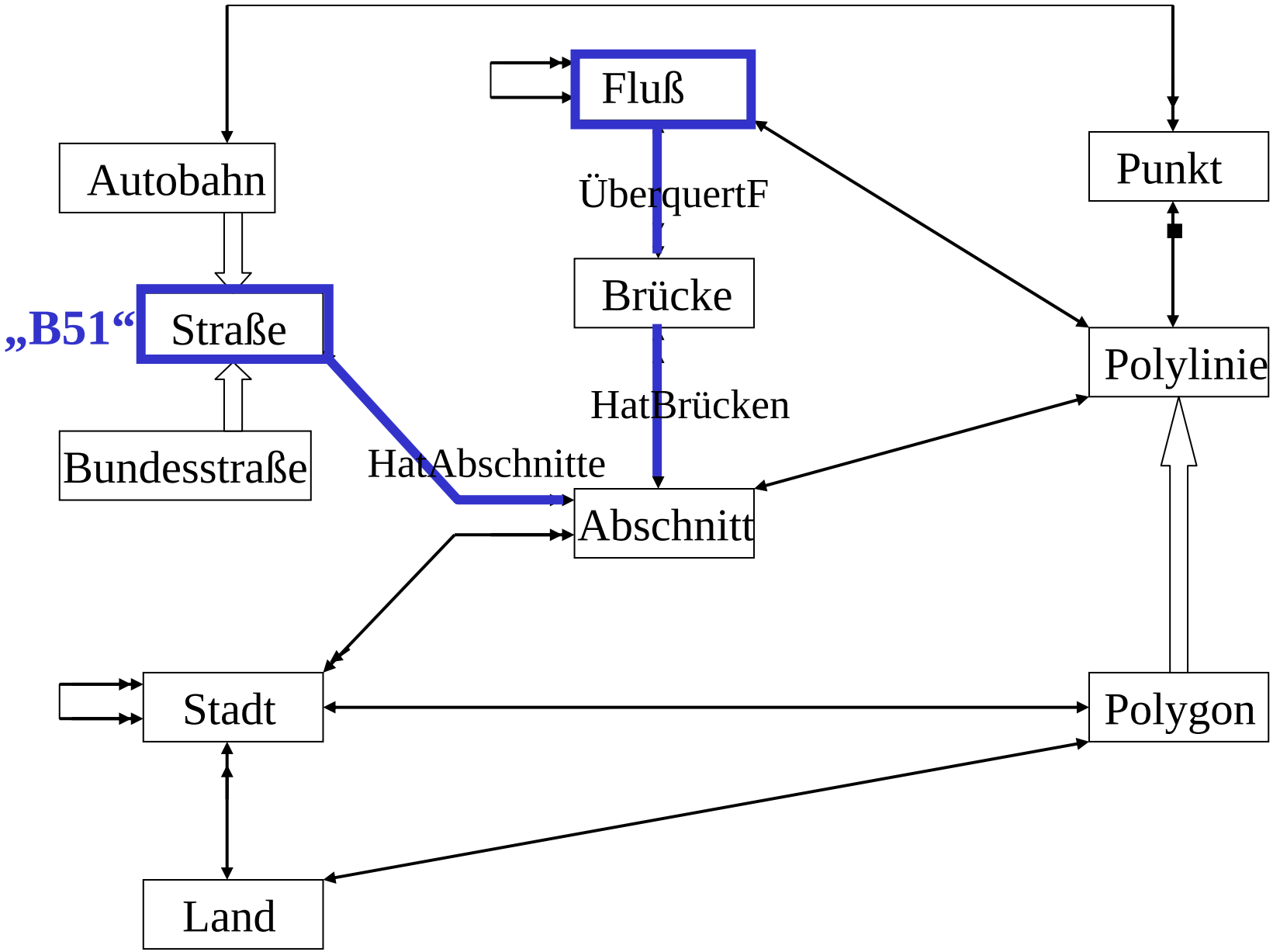
2) Welche Straßen überqueren die Mosel?

```
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 km^2 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
```

Land	*Gesamteinwohner*	*Stadteinwohner*	
Deutschland	450 000	*Stadtname*	*Einwohnerzahl*
		Kaiserslautern	200 000
		Saarbrücken	250 000
Frankreich	5 350 000	*Stadtname*	*Einwohnerzahl*
		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, MULTISET
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 $X = \{X1, ..., Xm\} \subseteq sch(R)$ und $Y = \{Y1, ..., Yn\} \subseteq sch(R)$ gilt die *Funktionalabhängigkeit* $X \rightarrow Y$, wenn jederzeit für je zwei Tupel $r, s \in val(R)$ gelten muß:

$(r.X1=s.X1 \wedge ... \wedge r.Xm=s.Xm) \Rightarrow (r.Y1=s.Y1 \wedge ... \wedge r.Yn=s.Yn)$

Beispiel:

Bestellungen (<u>Datum, KNr, PNr</u>, 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 sch(R). Die **transitive Hülle F^+** ist die Menge der aus F logisch ableitbaren FAs.

Ableitungskalkül (Armstrong-Regeln):

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

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 $W \subseteq sch(R)$, 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 = { Datum, KNr, PNR $\rightarrow$ Menge,
PNr $\rightarrow$ Bez, Preis, Gewicht }.

Es folgen:

Datum, KNr, PNR $\rightarrow$ PNr	nach der Reflexivitätsregel,
Bez, Preis, Gewicht $\rightarrow$ Bez	nach der Reflexivitätsregel,
PNr $\rightarrow$ Bez	nach der Transitivitätsregel,
Datum, KNr, PNr $\rightarrow$ Bez	nach der Transitivitätsregel

Ableitbarkeitstest für FAs: Xplus-Algorithmus

Definition:

Sei R eine Relation mit sch(R) und FA-Menge F, und sei $X \subseteq sch(R)$.
Die *Hülle* X^+ von X ist die Menge aller $A \in sch(R)$ 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 → H *)
        newattr := false;
        for each Y → Z ∈ F do
            if Y ⊆ H then
                if not (Z ⊆ H) then H := H ∪ Z; newattr := true; fi; fi;
        od
     od
     return H;
end xplus;
```

Beispiel: Xplus-Algorithmus

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 sch(R) und FA-Menge F, und sei $X \subseteq sch(R)$.
$X \subseteq sch(R)$ ist ein *Schlüsselkandidat*, wenn $X \rightarrow sch(R) \in F+$ gilt und es keine echte Teilmenge von X gibt, die diese Eigenschaft hat.
$A \in sch(R)$ 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 := ∅;
     for each S ∈ 2^sch(R) do
         iskey := true;
         if xplus (S) ≠ sch(R) then iskey := false fi;
         for each A ∈ S while iskey do
             if xplus (S - {A}) = sch(R) then iskey := false fi
         od;
         if iskey then K := K ∪ {S} fi
     od;
     return K;
```

11.2 Relationale Normalformen

Definition:
Sei R eine Relation mit sch(R) und FA-Menge F.
R ist in ***Boyce-Codd-Normalform (BCNF),*** wenn für jede FA
$X \rightarrow A \in F+$ mit $X \subseteq sch(R)$, $A \in sch(R)$ und $A \notin X$ gilt,
daß X einen Schlüsselkandidaten enthält.

Definition:
Sei R eine Relation mit sch(R) und FA-Menge F.
R ist in ***3. Normalform (3NF),*** wenn für jede FA
$X \rightarrow A \in F+$ mit $X \subseteq sch(R)$, $A \in sch(R)$ 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 → A ∈ F while isbcnf do
        if A ∉ X then if XPlus(X) ≠ sch(R) then isbcnf := false fi fi;
    od
    return isbcnf;
end BCNFtest;
```

Beispiele: Normalformen

1) Best (Datum, KNr, PNr, Menge) mit
   F = {Datum, KNr, PNr $\rightarrow$ Menge}
   und
   Prod (PNr, Bez, Preis, Gewicht) mit
   F = {PNr $\rightarrow$ Bez, Preis, Gewicht}

2) Vorlesungen (Titel, Zeit, Raum) mit
   F = {Titel $\rightarrow$ Raum,
   Zeit, Raum $\rightarrow$ 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 Xi $\subseteq$ sch(R) und X1 $\cup$ .. $\cup$ Xk = sch(R) heißt ***abhängigkeitsbewahrend***, wenn gilt:

$$\left((F^+|_{X1})^+ \cup ... \cup (F^+|_{Xk})^+ \right)^+ = F^+$$

Definition:
Sei R eine Relation mit sch(R) und FA-Menge F.
Eine Zerlegung von R in R1 (X1), ..., Rk (Xk) mit Xi $\subseteq$ sch(R) und X1 $\cup$ ... $\cup$ Xk = sch(R) heißt ***(projektion-join-) verlustfrei***,
wenn für alle möglichen Ausprägungen, die F erfüllen, gilt:

π[X1](R) |×| ... |×| π[Xk](R) = R.

Beispiele: Abhängigkeitsbewahrung

1) Bestellungen (Datum, KNr, PNr, Bez, Preis, Gewicht, Menge}mit
   F = {Datum, KNr, PNR $\rightarrow$ Menge,
   PNr $\rightarrow$ Bez, Preis, Gewicht}
   Zerlegung in
   Best (Datum, KNr, PNr, Menge) mit
   F1 = {Datum, KNr, PNR $\rightarrow$ Menge}
   und
   Prod (PNr, Bez, Preis, Gewicht) mit
   F2 = {PNr $\rightarrow$ Bez, Preis, Gewicht}

2) Vorlesungen (Titel, Zeit, Raum) mit
   F = {Titel $\rightarrow$ Raum,
   Zeit, Raum $\rightarrow$ Titel}
   Zerlegung in
   Vorlesungsräume (Titel, Raum) mit
   F1 = {Titel $\rightarrow$ Raum}
   und
   Vorlesungszeiten (Titel, Zeit) mit F2 = $\varnothing$

Beispiele: Verlustfreiheit

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

Datum	**KNr**	**PNr**	**Bez**	**Preis**	**Ge-wicht**	**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.	Disket-ten	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 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, 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 $X \rightarrow sch(R) \notin F+$ *)
dann
zerlege R in R' (X, Y) und R" (X, 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 → Abflugzeit, FSNr,
FlugNr, Datum, TicketNr → SitzNr,
TicketNr → Name, Adresse,
GNr → Gewicht }

Schlüsselkandidat:
<u>FlugNr, Datum, TicketNr, GNr</u>

Zerlegungsschritte:

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

11.4 Relationensynthese

Definition:

Sei eine Relation mit 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

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

F = { AngNr → Name,
AngNr → Gehalt,
AngNr → Tarif,
AngNr → AbtNr,
AngNr → Abtleiter,
ProjektNr → Projektleiter,
AbtNr → Abtleiter,
AngNr, Abtleiter → Leistung,
Tarif, Leistung → Gehalt}

Schritt 1: → { AngNr → Name,
AngNr → Gehalt,
AngNr → Tarif,
AngNr → AbtNr,
AngNr → Abtleiter,
ProjektNr → Projektleiter,
AbtNr → Abtleiter,
AngNr → Leistung,
Tarif, Leistung → Gehalt}

Schritt 2: → { AngNr → Name,
AngNr → Tarif,
AngNr → AbtNr,
ProjektNr → Projektleiter,
AbtNr → Abtleiter,
AngNr → Leistung,
Tarif, Leistung → 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→A∈H do
      for each C∈X do
        Hred := H - { X→A} ∪ {(X-{C})→A};
        if A ∈ xplus (H, X-{C})) then H := Hred  fi;
      od;
    od;
    return H;

G := reduce(F);
for each X→A∈G do
  G := G - { X→A};
  if not (A ∈ xplus (G, X)) then G := G ∪ { X→A} fi;
od;
return G;
```

Algorithmus zur Relationensynthese

Schritt 1: Bereche eine minimale Überdeckung F' von F.
Schritt 2: Partitioniere die FAs in F' nach gleichen linken Seiten, d.h.
für jedes Xi fasse alle $Xi \rightarrow A1, ..., Xi \rightarrow Ak_i$ zusammen
Schritt 3: Bilde für jede Partition von Schritt 2 eine
Relation Ri mit $sch(Ri) = Xi \cup \{Ai, ..., Ak_i\}$.
Schritt 4: Falls $sch(R) - (\cup_i sch(Ri))$ nichtleer ist, bilde
eine weitere Relation R' mit $sch(R') = sch(R) - (\cup_i sch(Ri))$.
Schritt 5: Falls keine der Relationen Ri, 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

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

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

Ergebnis der Relationensynthese:

Angestellte (<u>AngNr</u>, Name, Tarif, Leistung, AbtNr)
Projekte (<u>ProjektNr</u>, Projektleiter)
Abteilungen (<u>AbtNr</u>, Abtleiter)
Lohn (<u>Tarif, Leistung</u>, Gehalt)
Projektmitarbeit (<u>AngNr, ProjektNr</u>)

11.5 Ergänzungen (1)

1) weitere Zerlegung von BCNF-Relationen:

Angestellte (ANr, Informatikkenntnisse, Kinder) mit F = ∅

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 = {*Datum, KNr, PNr* $\rightarrow$ *Menge,*
PNr $\rightarrow$ *Bez, Preis, Gewicht,*
Preis, Menge $\rightarrow$ *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 = {*KNr* $\rightarrow$ *Name, Postleitzahl, Stadt, Strasse,*
Postleitzahl $\rightarrow$ *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

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:

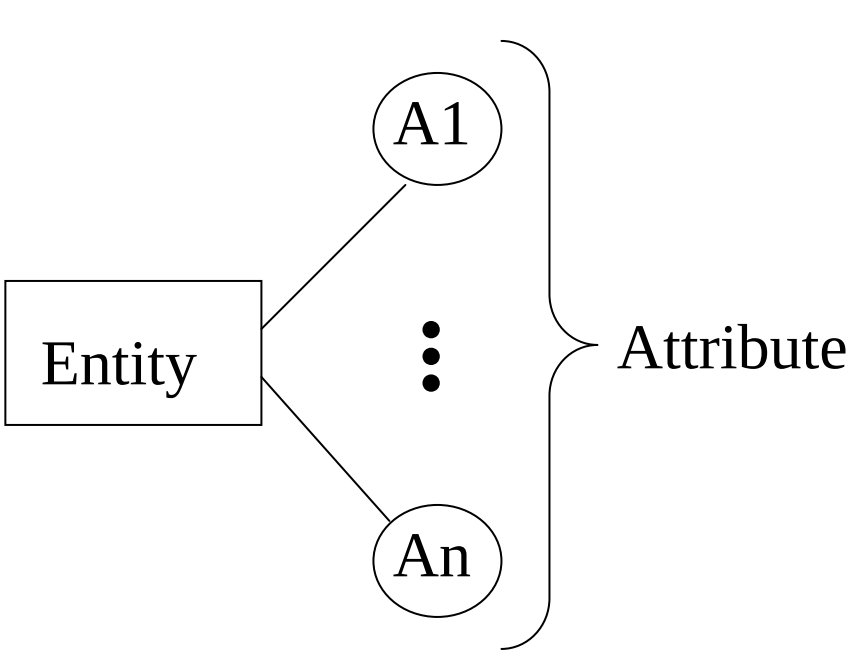
- einer „semantischen“ Modellierungssprache
- einer (i.d.R. informell-groben) Entwurfsmethodologie
- Sofwarewerkzeugen (DB-Design-Tools, Repositories)

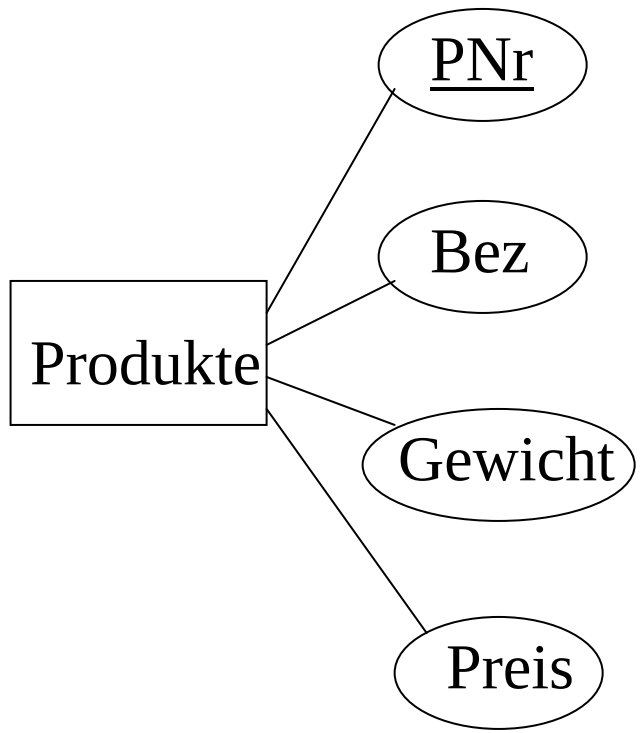
Grobe Vorgehensweise bei der Entwicklung von IS:

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

12.1 Entities (Objekte) und Entity-Mengen

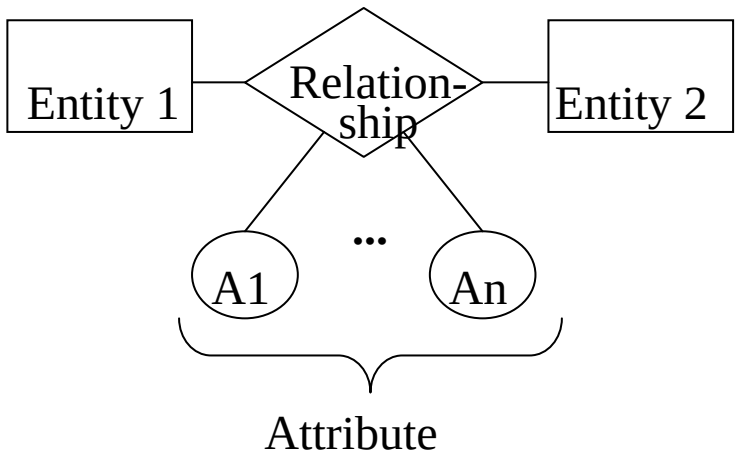
Ein ***Entity*** ist ein durch ***Attribute*** beschriebenes Objekt der realen Welt.
Gleichartige Entities werden zu einem Entity-Typ zusammengefasst.
Entities desselben Typs bilden eine ***Entity-Menge***.
Eine minimale Attributmenge, die jedes Entity einer Entity-Menge eindeutig identifiziert, heißt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

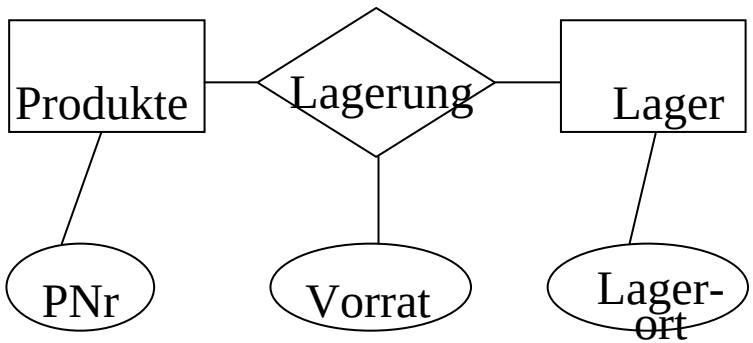


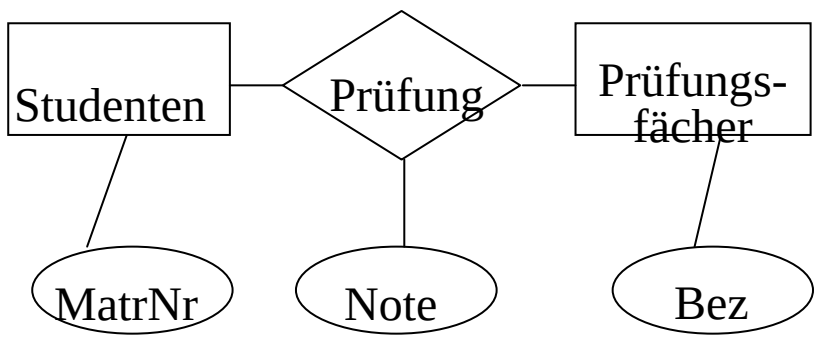


Relationships (Beziehungen, Assoziationen)

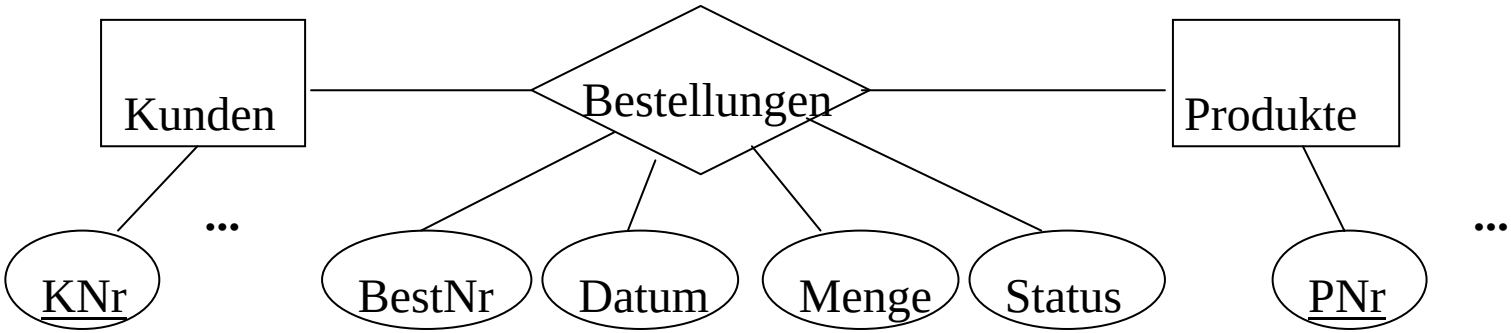
Eine ***Relationship-Menge*** R zwischen Entity-Mengen E1, ..., En ist eine Teilmenge von E1 × ... × En. Ein Element von <e1, ..., en> von R heißt ***Relationship*** zwischen den Entities e1, ..., en.
Eine minimale Teilmenge der Schlüssel von E1, ..., En, die jede Relationship in R eindeutig identifiziert, heißt ***Schlüssel(kandidat).***



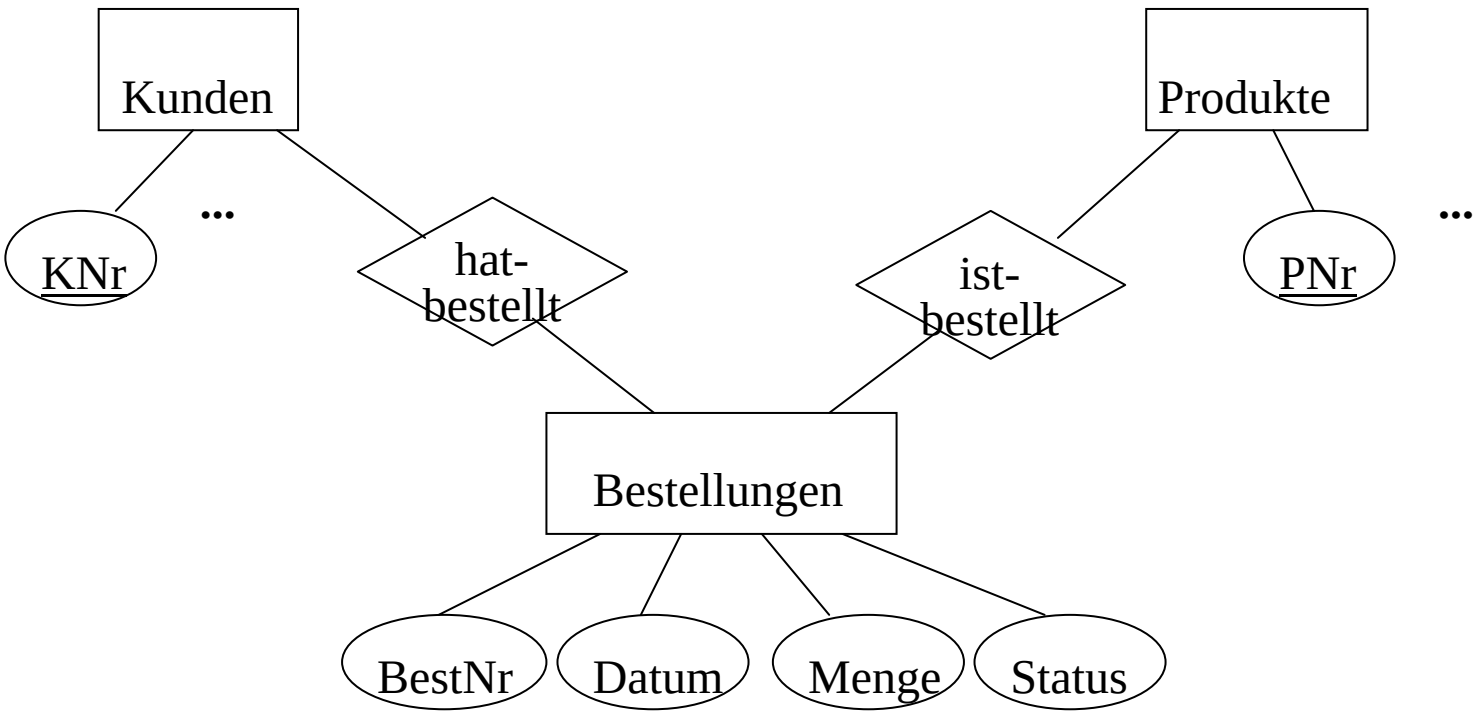




Entity oder Relationship ? (Varianten 1 und 2)



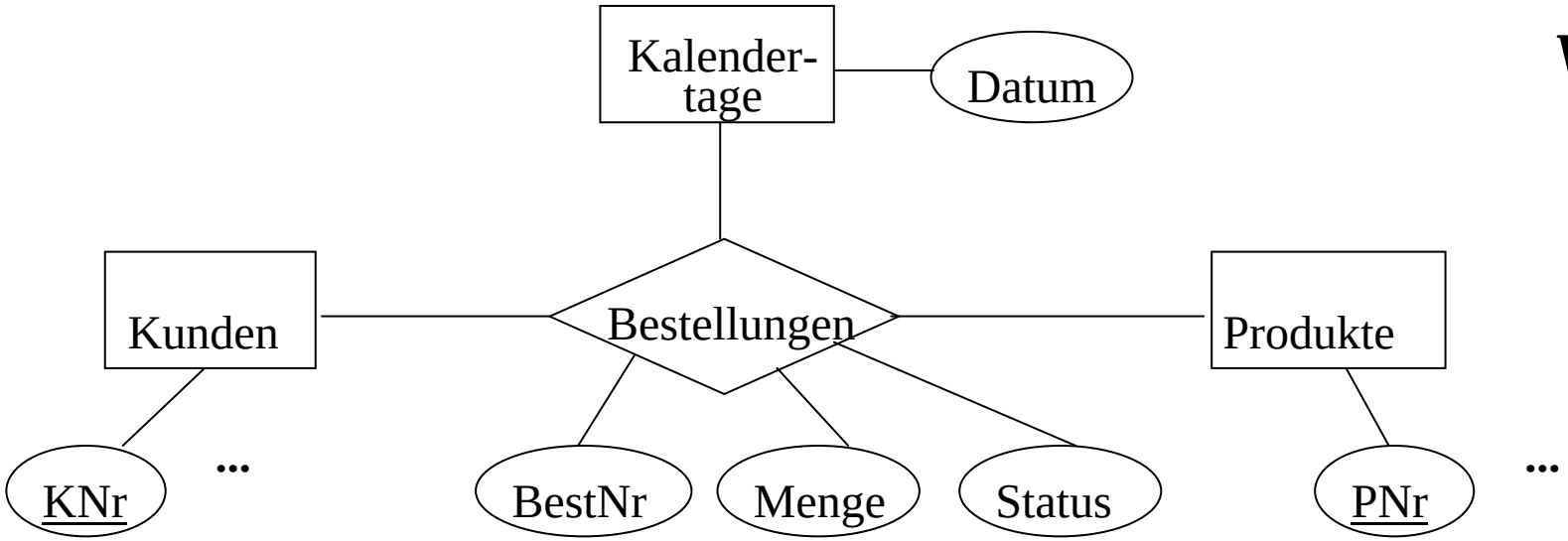
Variante 1



Variante 2

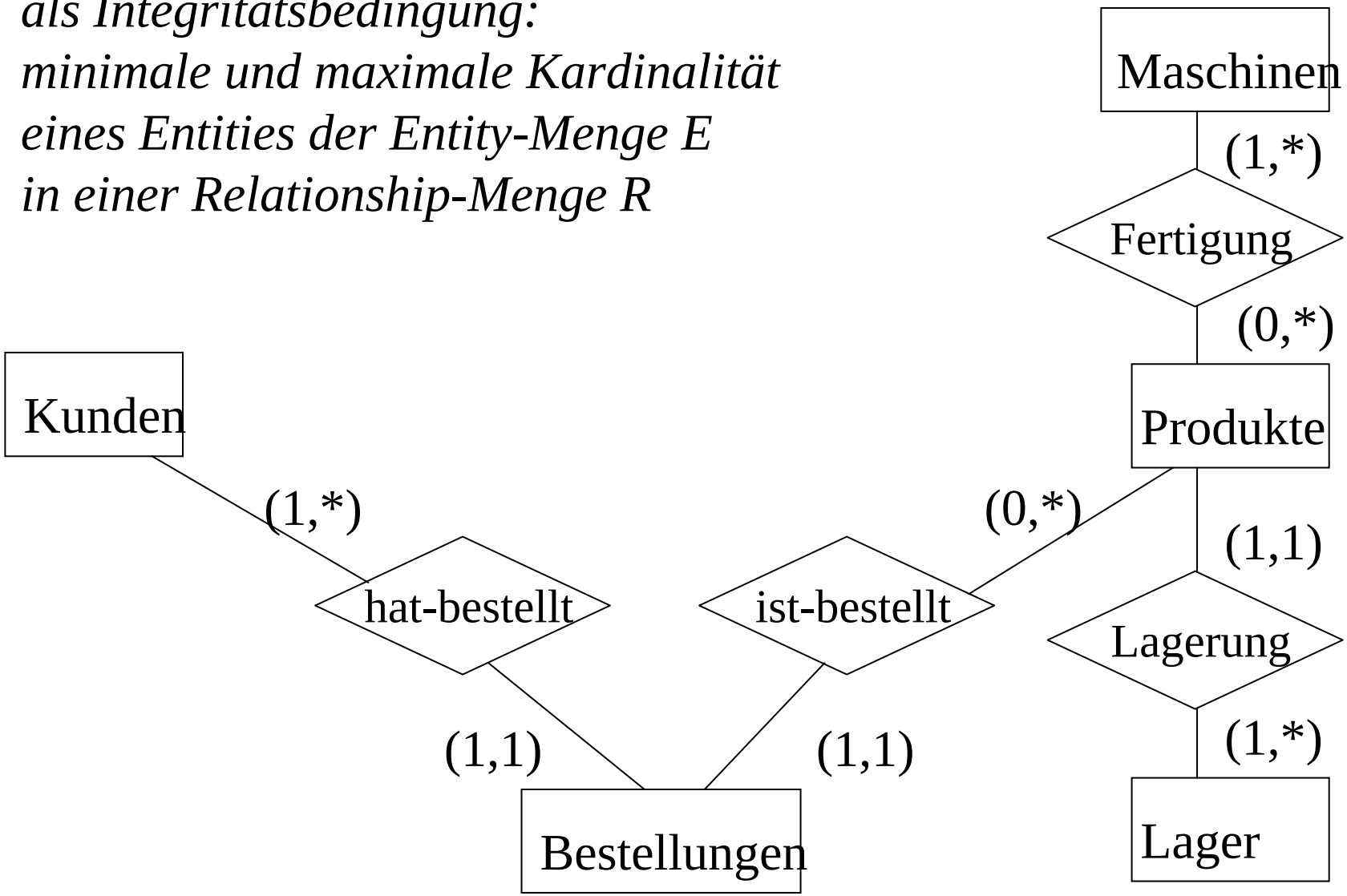
Entity oder Relationship ? (Variante 3)

Variante 3

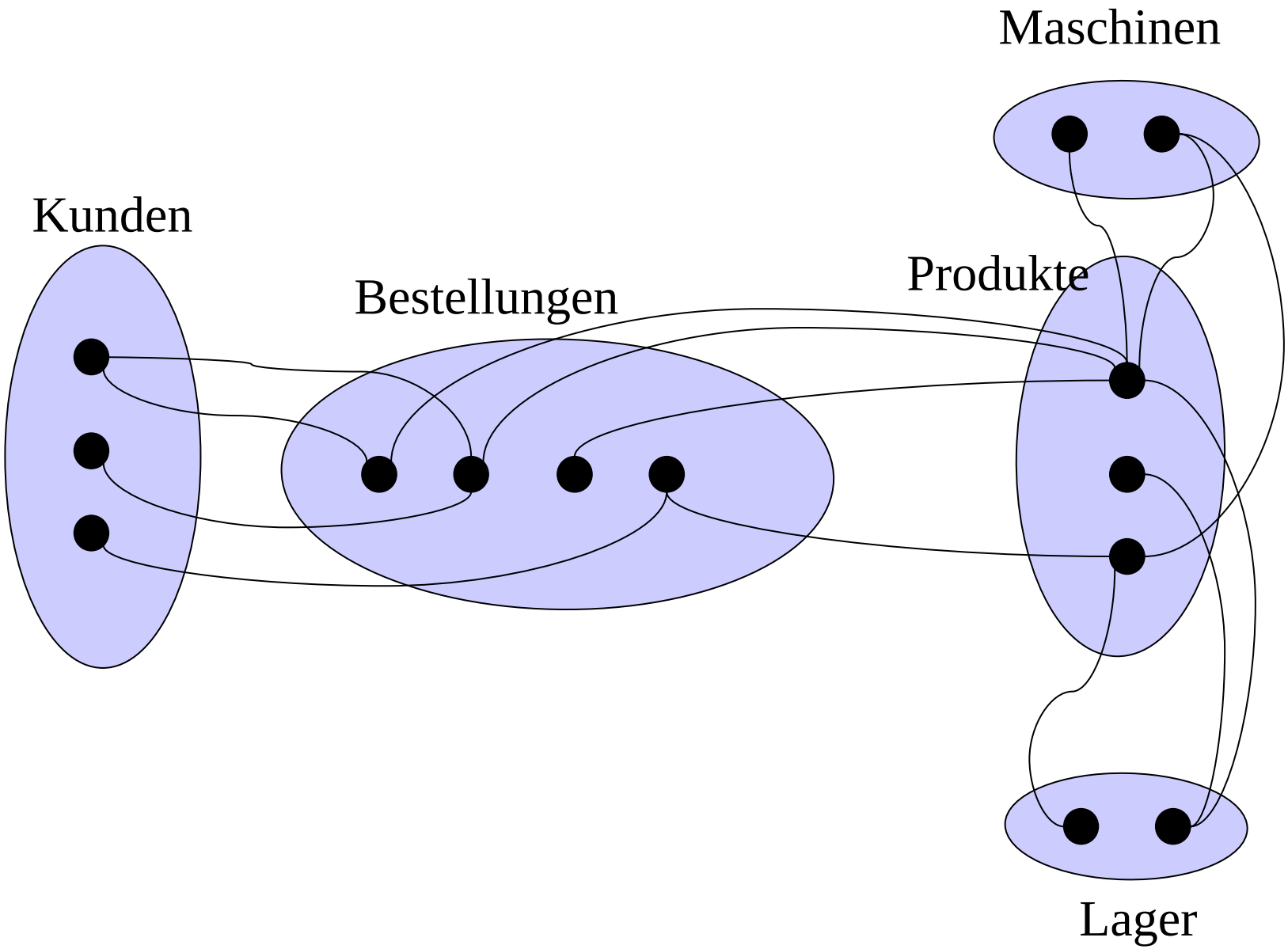


Kardinalitätsbedingungen für Relationships (1)

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

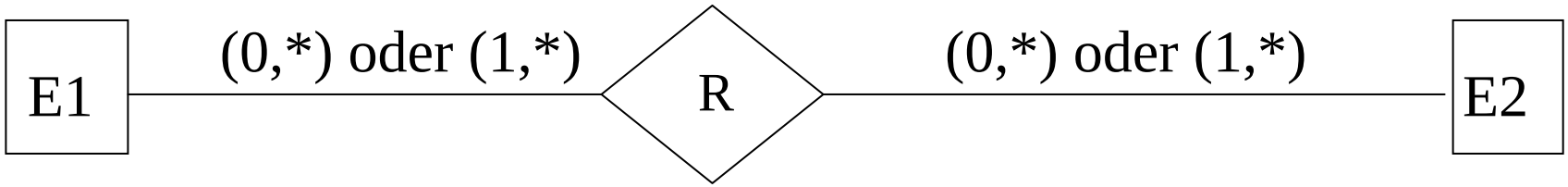


Kardinalitätsbedingungen für Relationships (2)

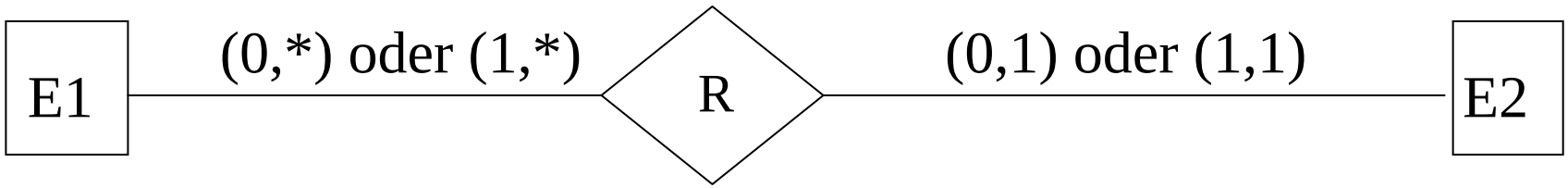


Typen von Kardinalitätsbedingungen (1)

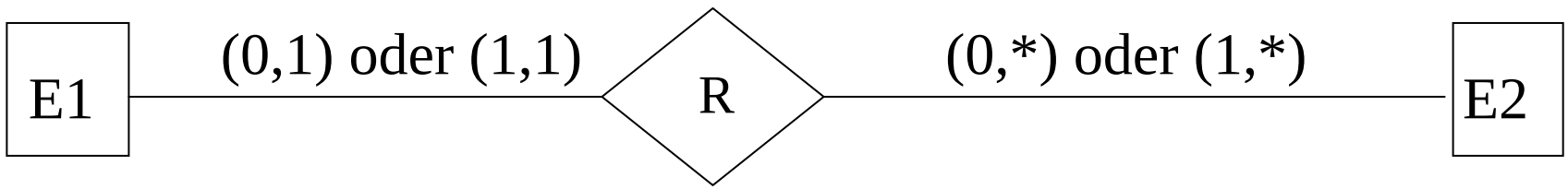
M:N-Relationship



1:N-Relationship

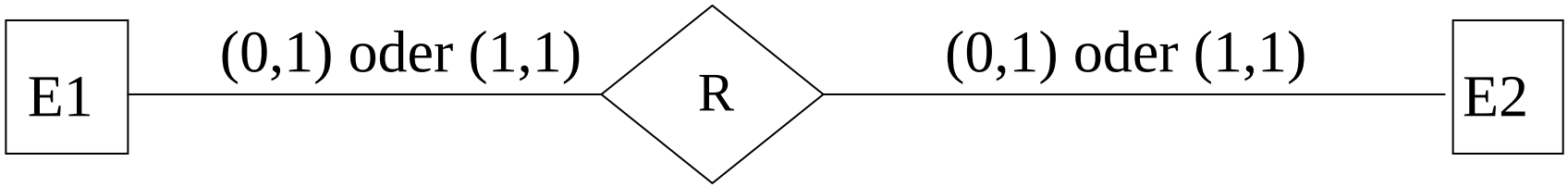


N:1-Relationship

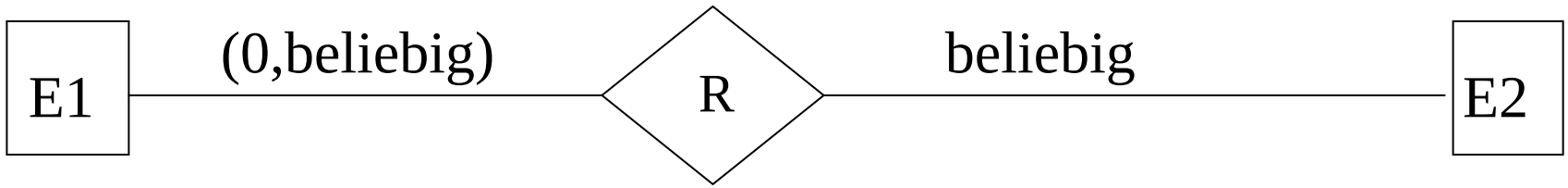


Typen von Kardinalitätsbedingungen (2)

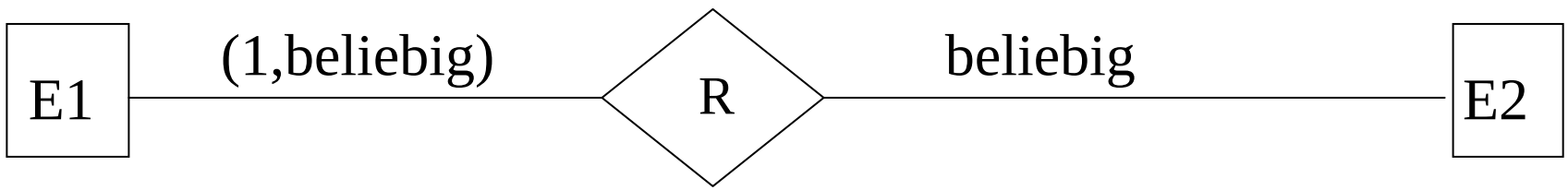
1:1-Relationship



Optionale Rolle von E1 in R

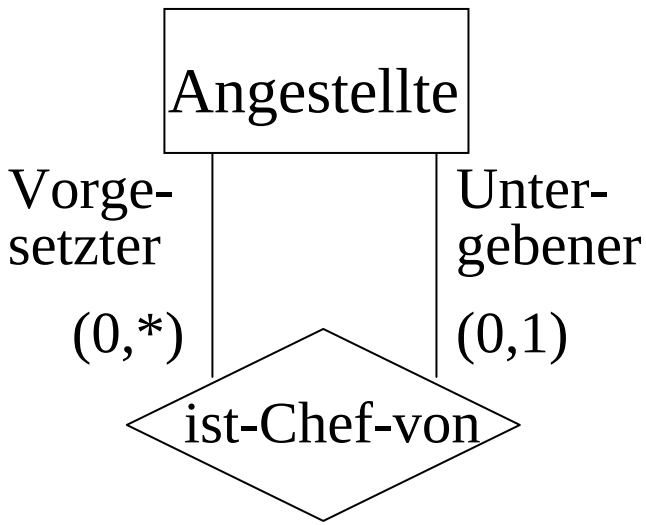


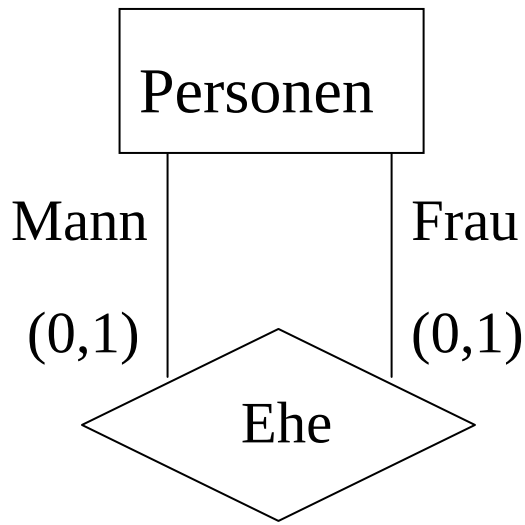
Verpflichtende Rolle von E1 in R



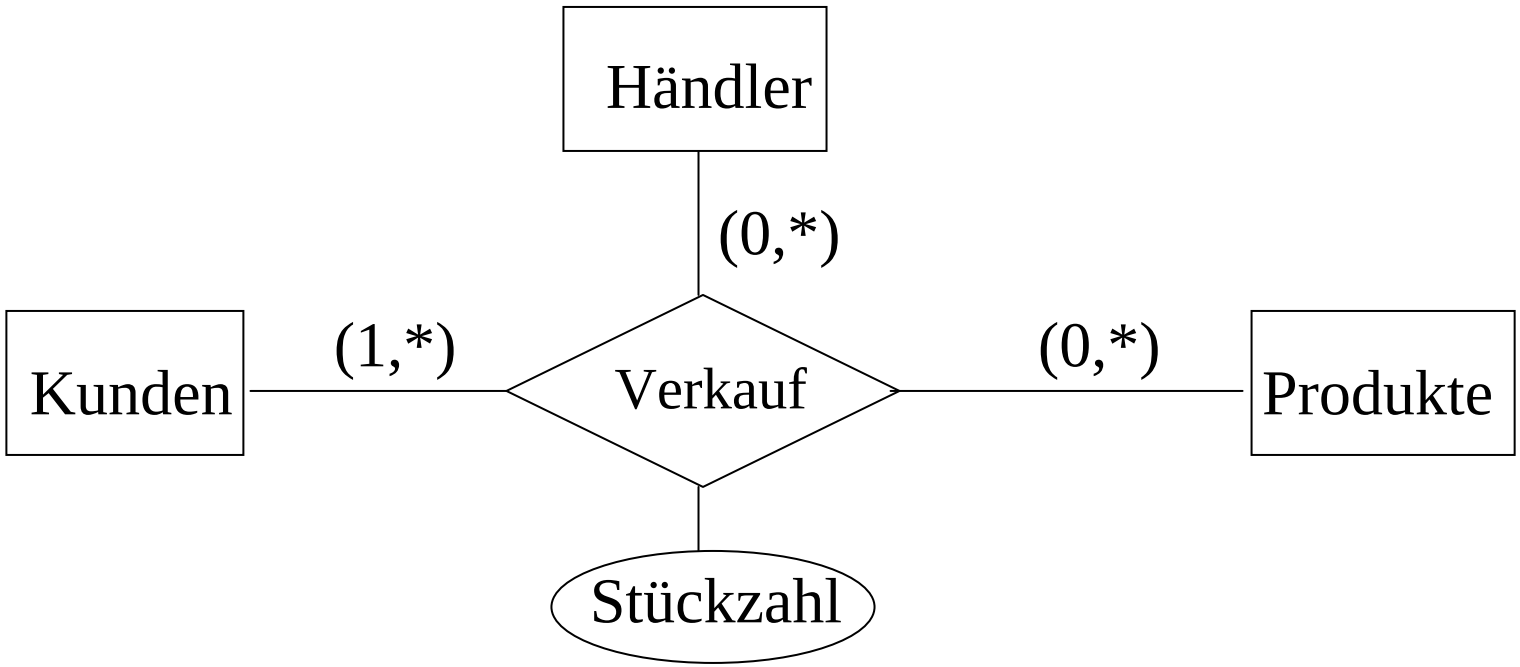
Rekursive Relationships

Relationships auf einer Menge erfordern Rollennamen



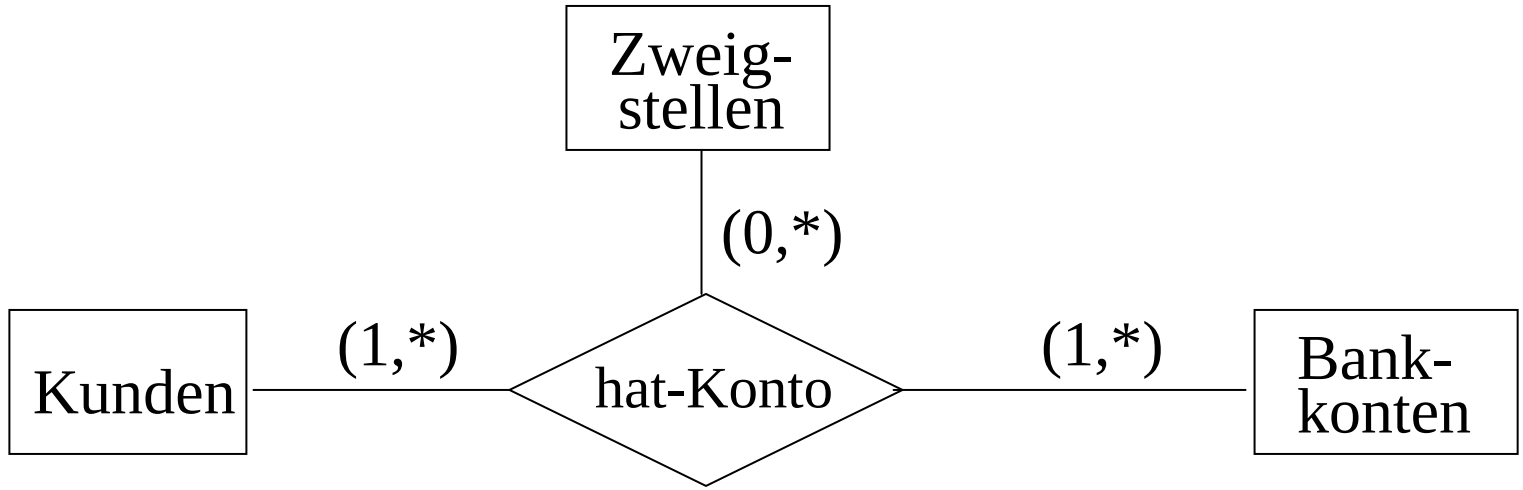


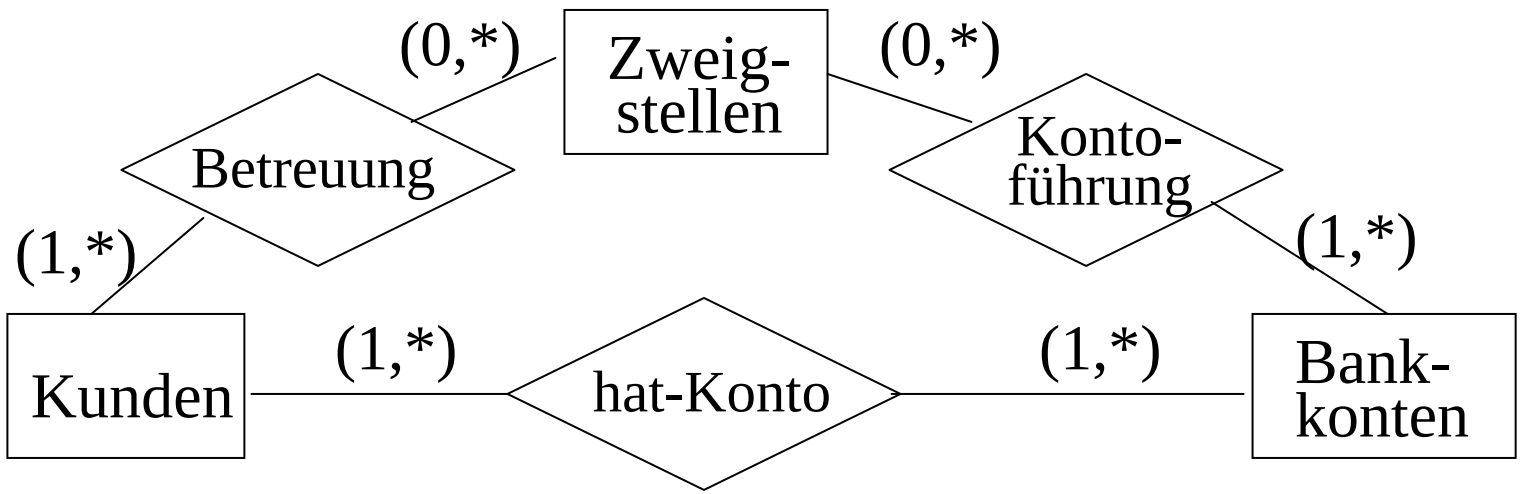
Ternäre Relationships: Beispiel 1



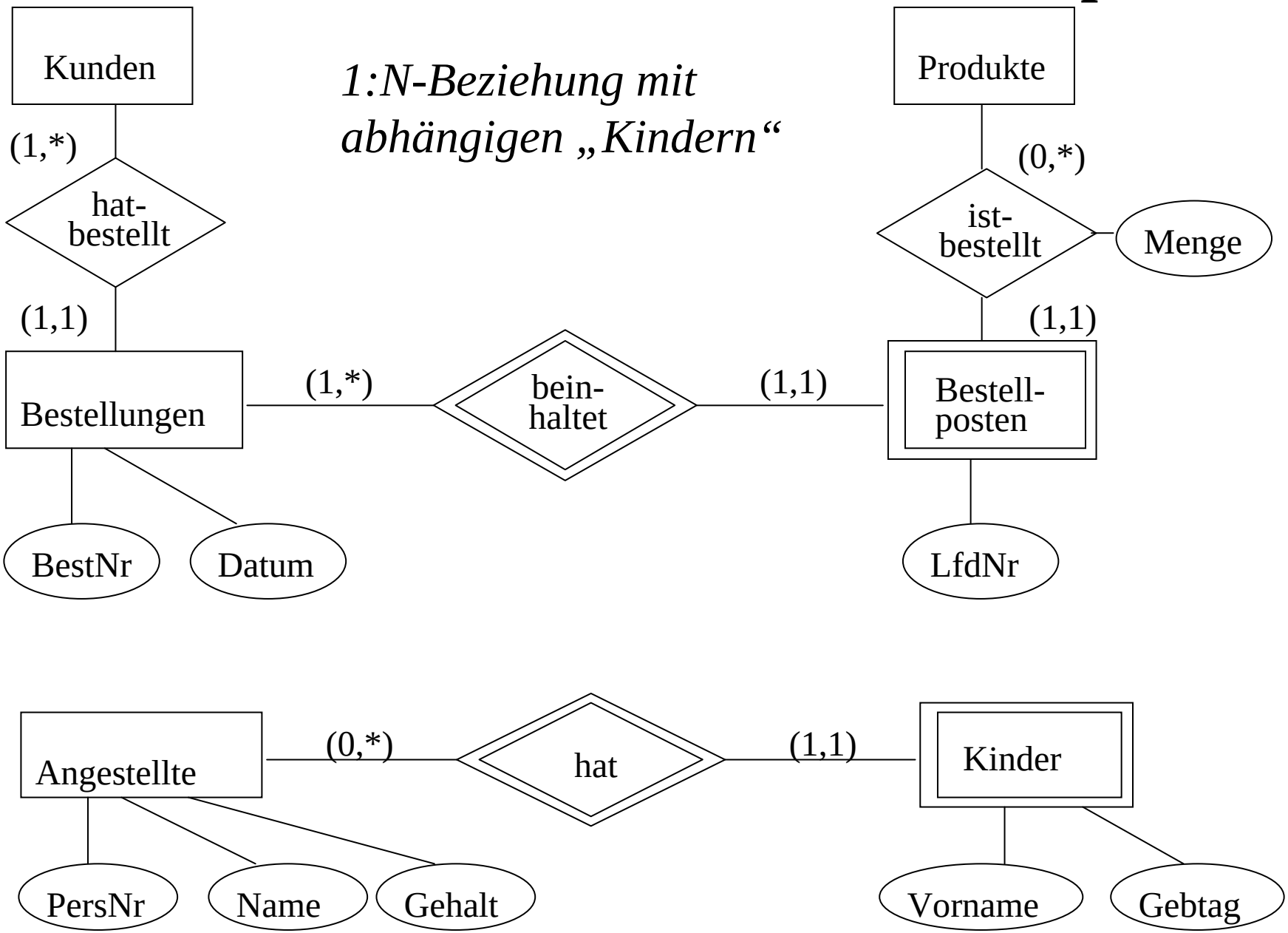
Achtung: Ternäre (und höherstellige) Relationships sind nicht immer zerlegbar in mehrere binäre Relationships

Ternäre Relationships: Beispiel 2



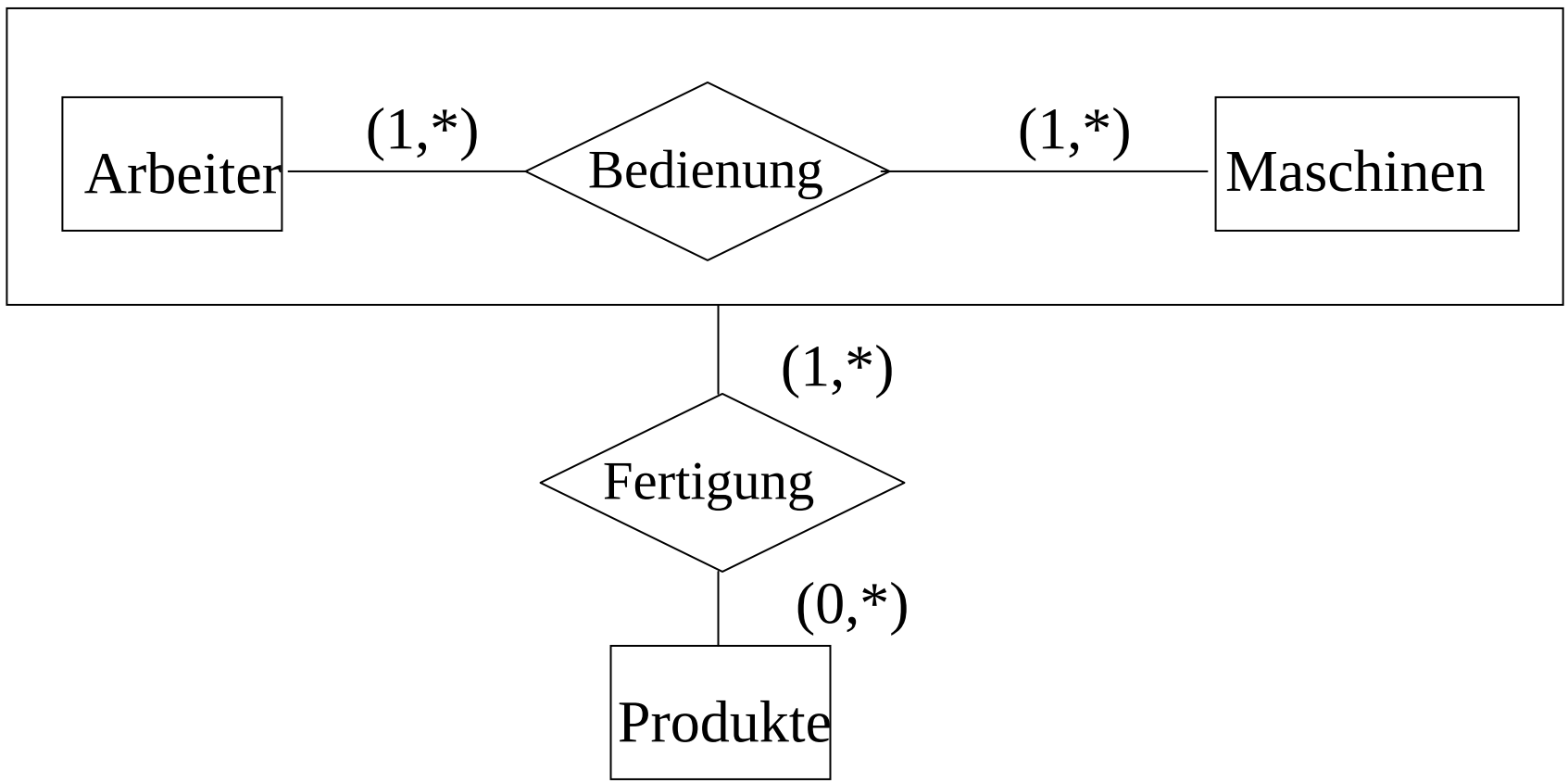


Schwache Entities und Relationships



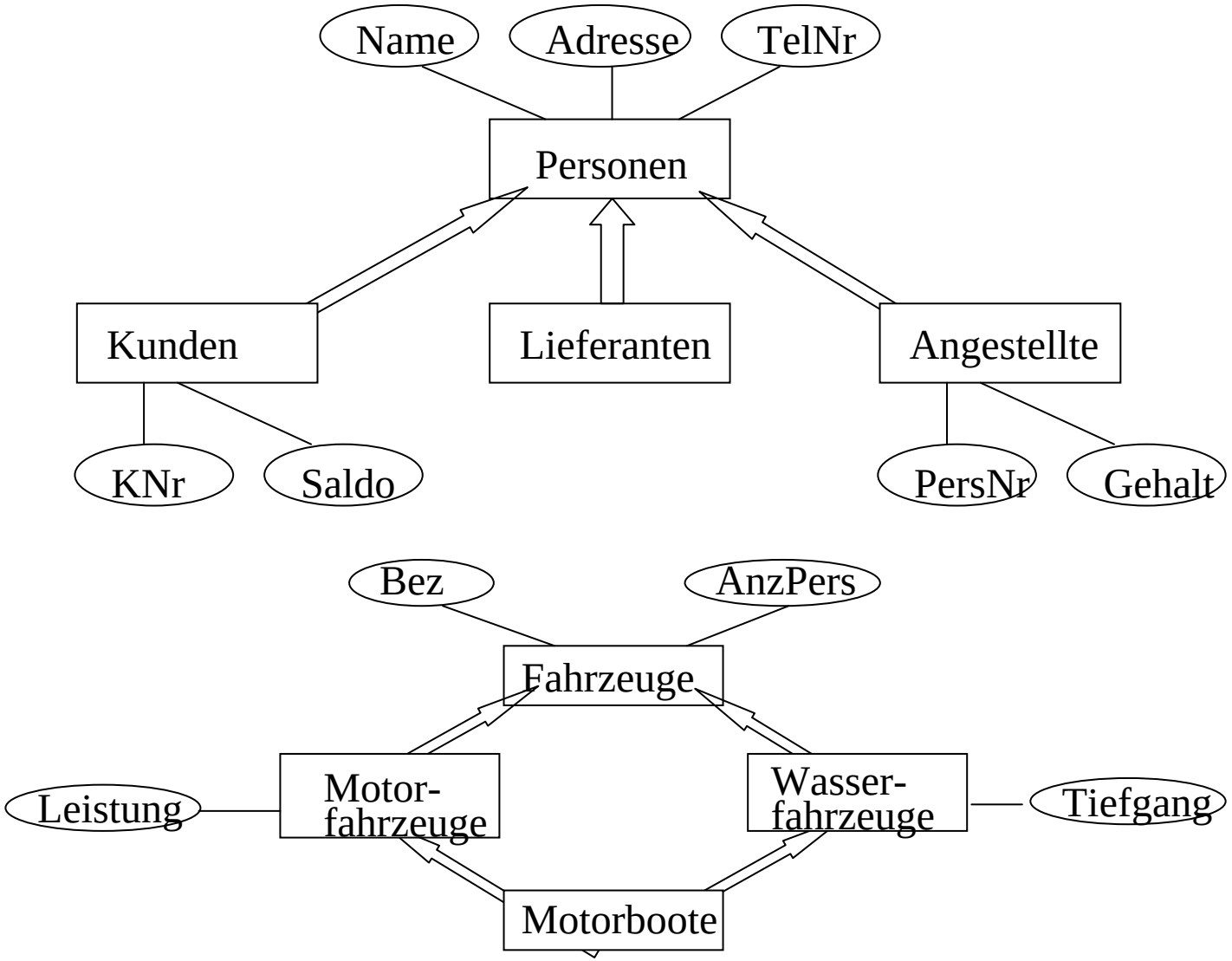
Aggregation

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)

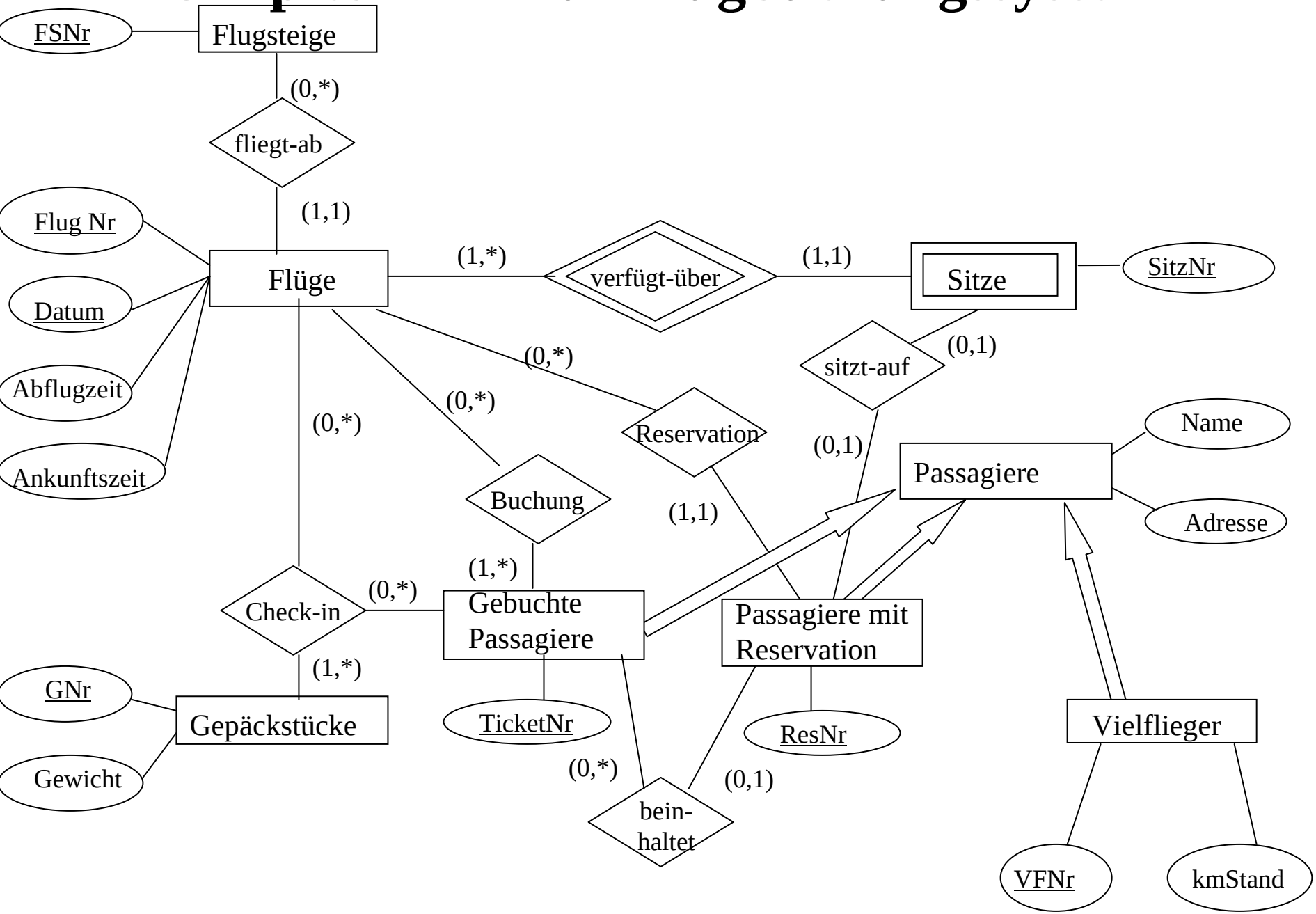
1) Analyse des Informationsbedarfs
2) Festlegung von Entity-Mengen mit ihren Attributen
3) Festlegung von Primärschlüsseln
4) Bildung von Generalisierungsbeziehungen
5) Festlegung von Relationship-Mengen
6) Spezifikation von Kardinalitätsbedingungen
   und ggf. weiteren Integritätsbedingungen

Beispiel Flugbuchungssystem

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

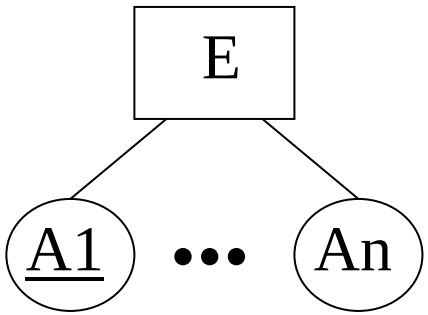
Flüge werden durch eine Flugnummer identifiziert, die für Flüge am selben Tag eindeutig ist.
Passagiere können einen Flug reservieren, was durch eine Reservationsnummer bestätigt wird.
Eine Reservation wird zu einer festen Buchung, indem man ein Ticket käuft.
Bei der Reservation oder später können Passagiere 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.

Simples ERM für Flugbuchungssystem



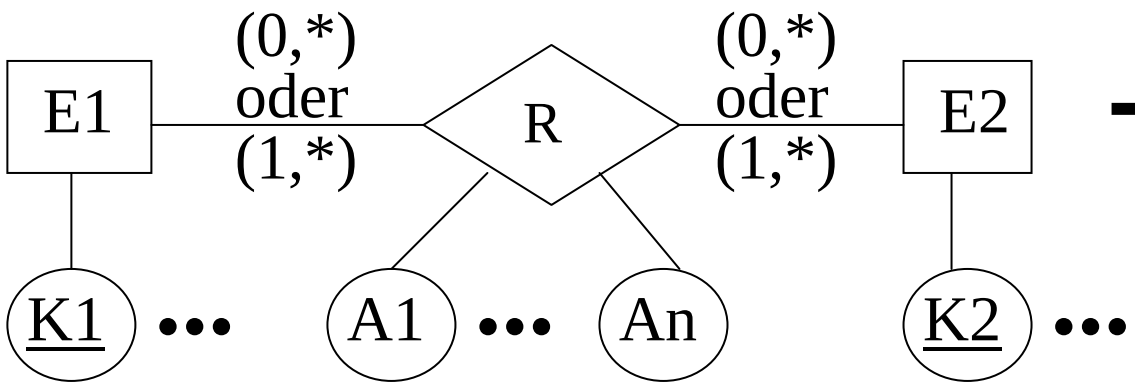
12.3 Abbildung ERM auf Relationenschema (1)

Regel 1: Jede Entity-Menge bildet eine Relation



E (A1, ..., An)

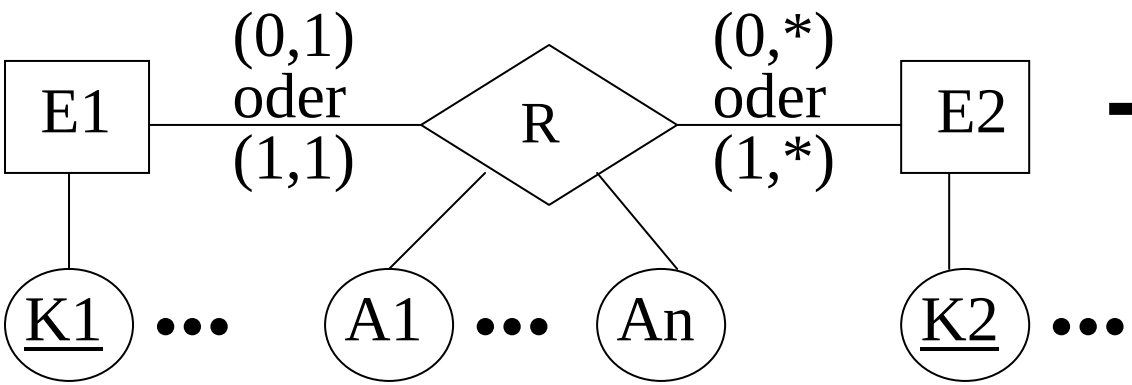
Regel 2: M:N-Relationships bilden eigene Relationen



R (K1, K2,
A1, ..., An)

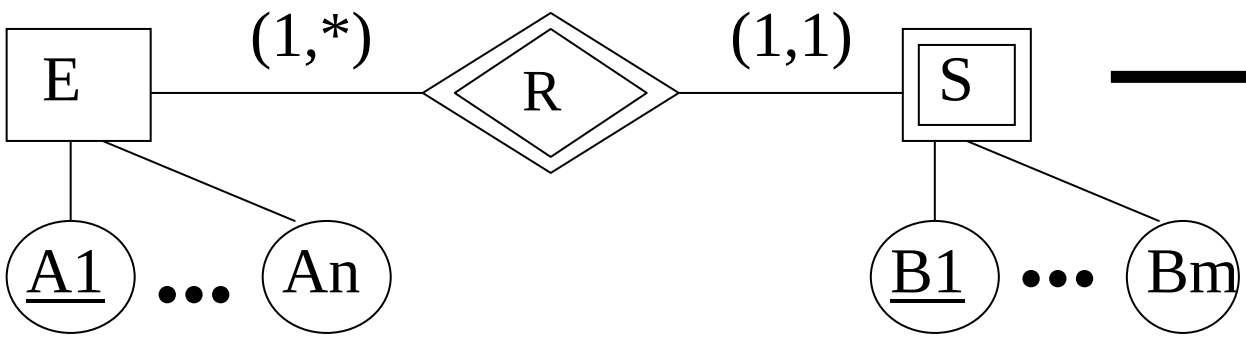
12.3 Abbildung ERM auf Relationenschema (2)

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



E1 (<u>K1</u>, ..., A1, ..., An, K2)
E2 (<u>K2</u>, ...)

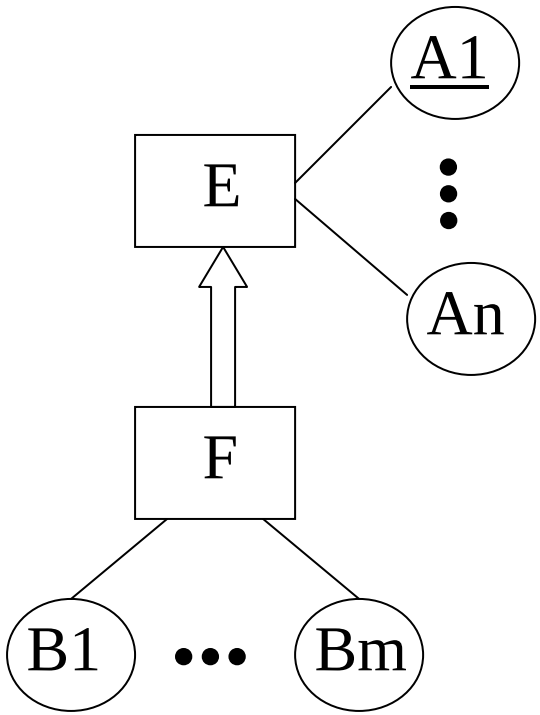
Regel 4: Eine schwache Entity-Menge bildet eine eigene Relation



E (<u>A1</u>, ..., An)
S (<u>A1, B1</u>, ..., Bm)

12.3 Abbildung ERM auf Relationenschema (3)

Regel 5: Generalisierung und Spezialisierung bilden jeweils eigene Relationen



→ E (A1, ..., An)
F (A1, B1, ..., Bm)

→ E (A1, ..., An)
F (A1, ..., An,
B1, ..., 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

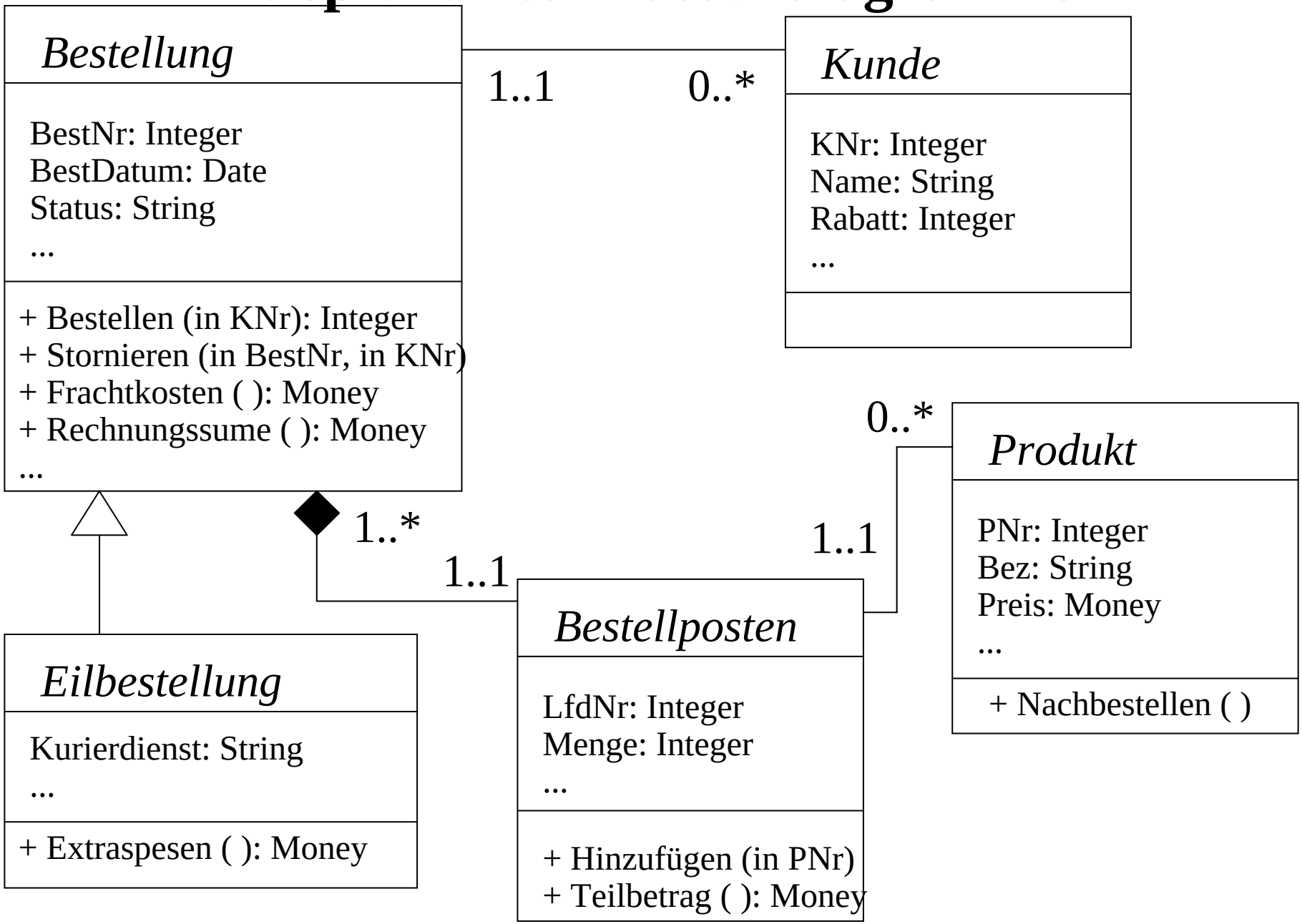
UML (Unified Modeling Language) ist der Industriestandard zur objektorientierten Anforderungsanalyse, Modellierung, Spezifikation und Dokumentation von Daten und Programmen

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

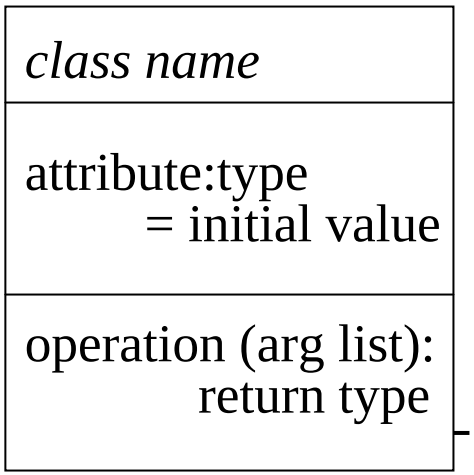
- Objektaggregationen
- Funktionen auf Objekten (Methoden)
- Interaktionen zwischen Objekten und ganze (Geschäfts-) Prozesse ($\rightarrow$ Kapitel 11)

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



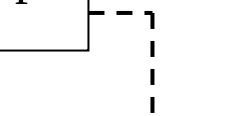
Klassen und Objekte



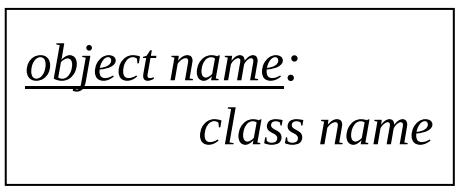
Klasse (Objekttyp, Komponente)

Attribute

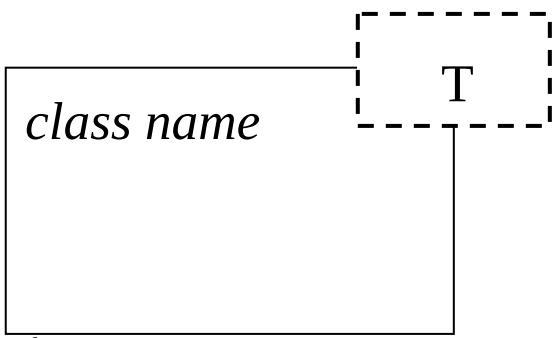
Methoden mit in-, out- und inout-Parametern
+: public, -: private, #: protected



Konsistenzbedingungen, Vor- und Nachbedingungen (constraints) in Textform

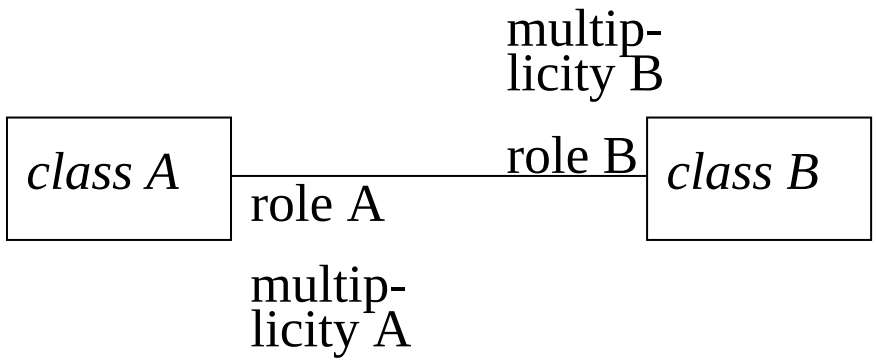


Objekt (Instanz einer Klasse)

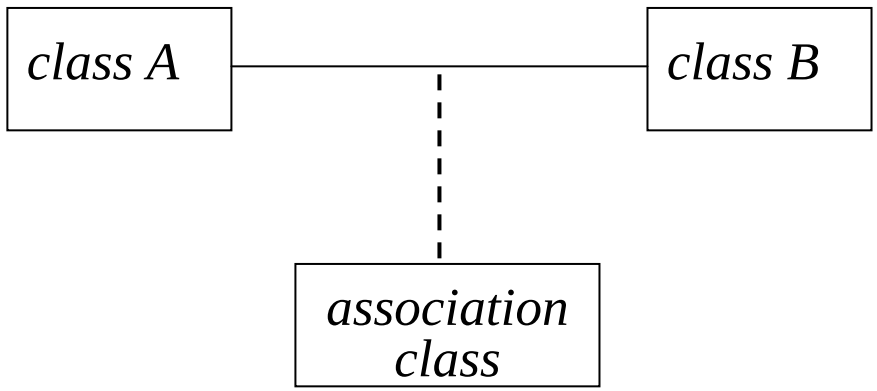


Parametrisierte Klasse (Template Class)
z.B. List <T>

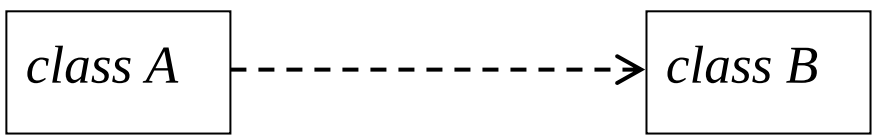
Assoziationen (Relationships)



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

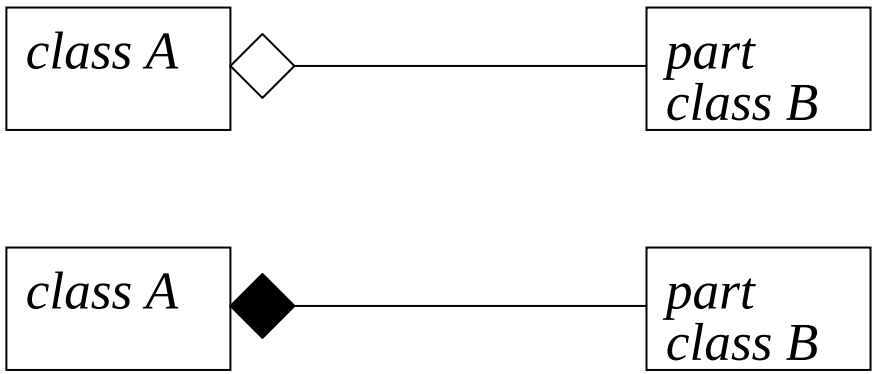


Beziehungsklasse (assocation class): entspricht den ERM-Relationships mit eigenen Attributen



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

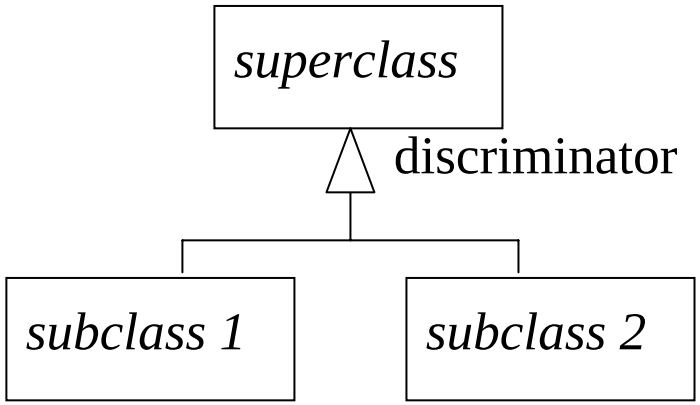
Aggregationen und Kompositionen



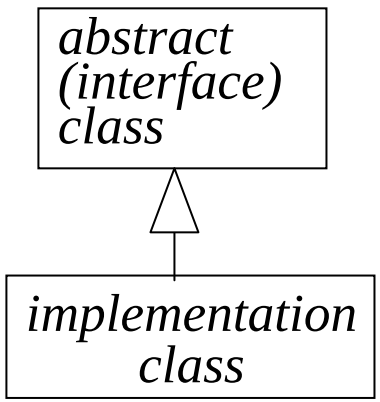
Aggregation (aggregation)
mit shared objects
(Sonderfall der Assoziation)

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

Generalisierungen



Generalisierungshierarchie
(generalization)

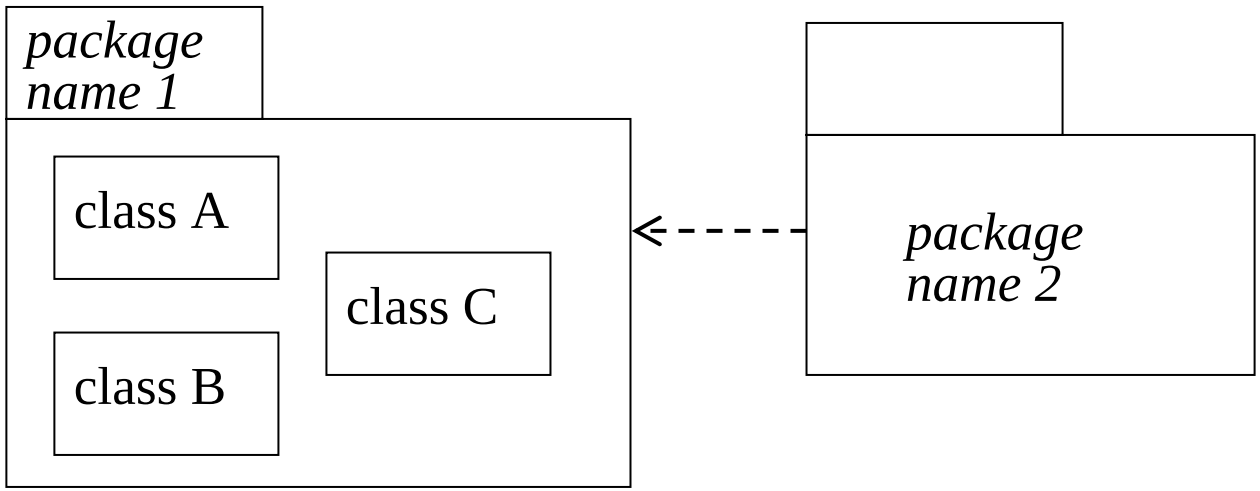


Beziehung zwischen Schnittstelle (ADT
und Implementierung
(Sonderfall der Generalisierung /
Spezialisierung)

Weitere Diagrammtypen in UML

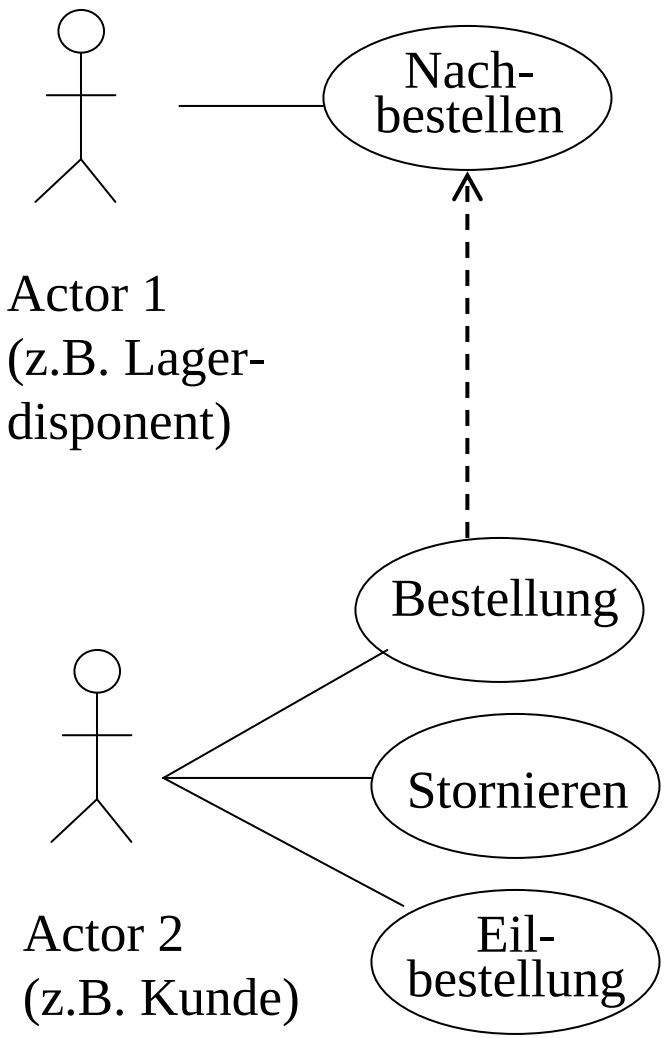
- ***Paket-Diagramme (Package Diagrams)*** als vergröberte Sicht auf große Klassendiagramme,
- ***Anwendungsfall-Diagramme (Use-Case Diagrams)*** zur groben Darstellung der Beziehung zwischen Benutzern und Funktionen,
- ***Interaktionsdiagramme (Sequence Diagrams)*** zur Darstellung der Methodenaufrufe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

Paketdiagramme



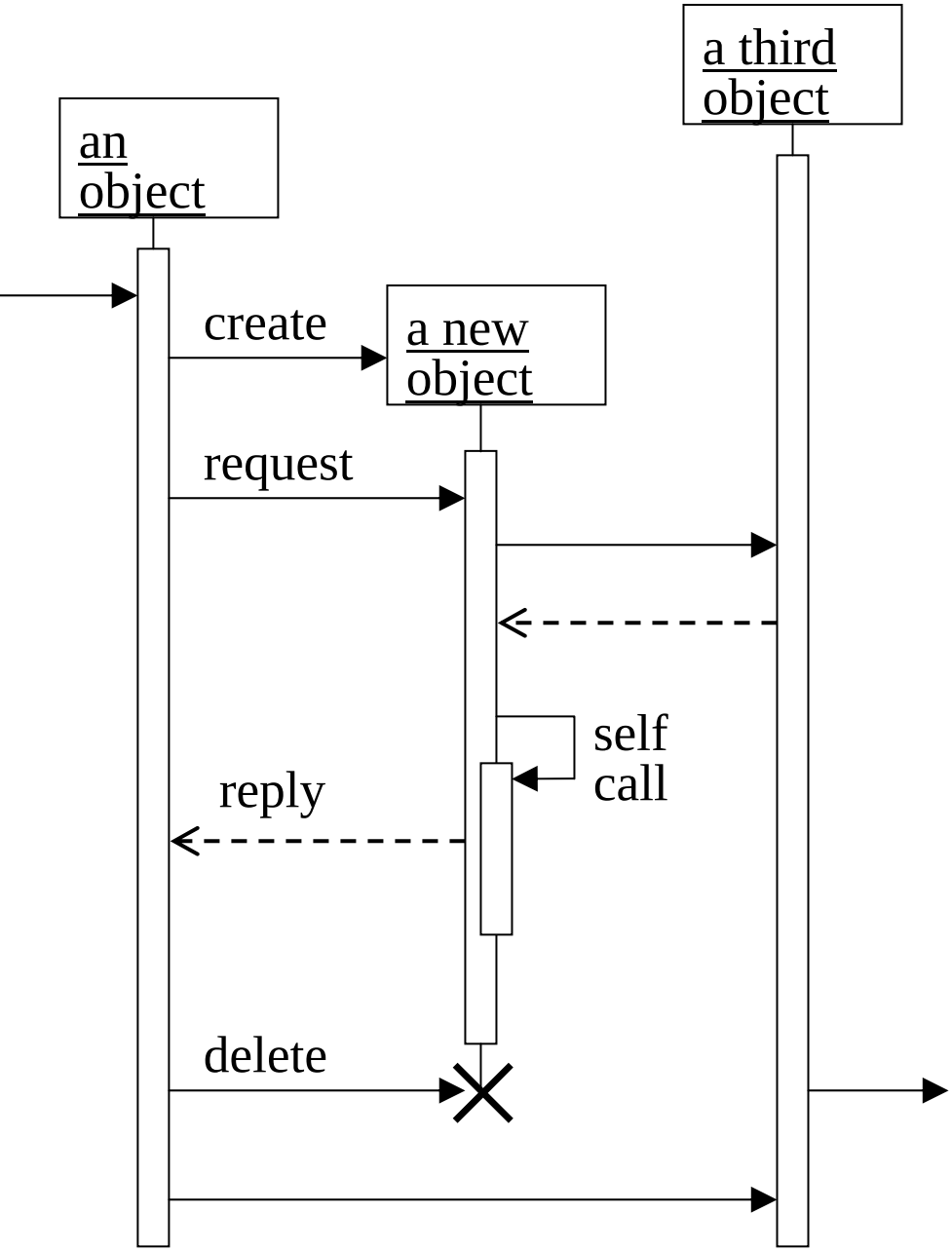
Paketdiagramm
(package diagram)

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

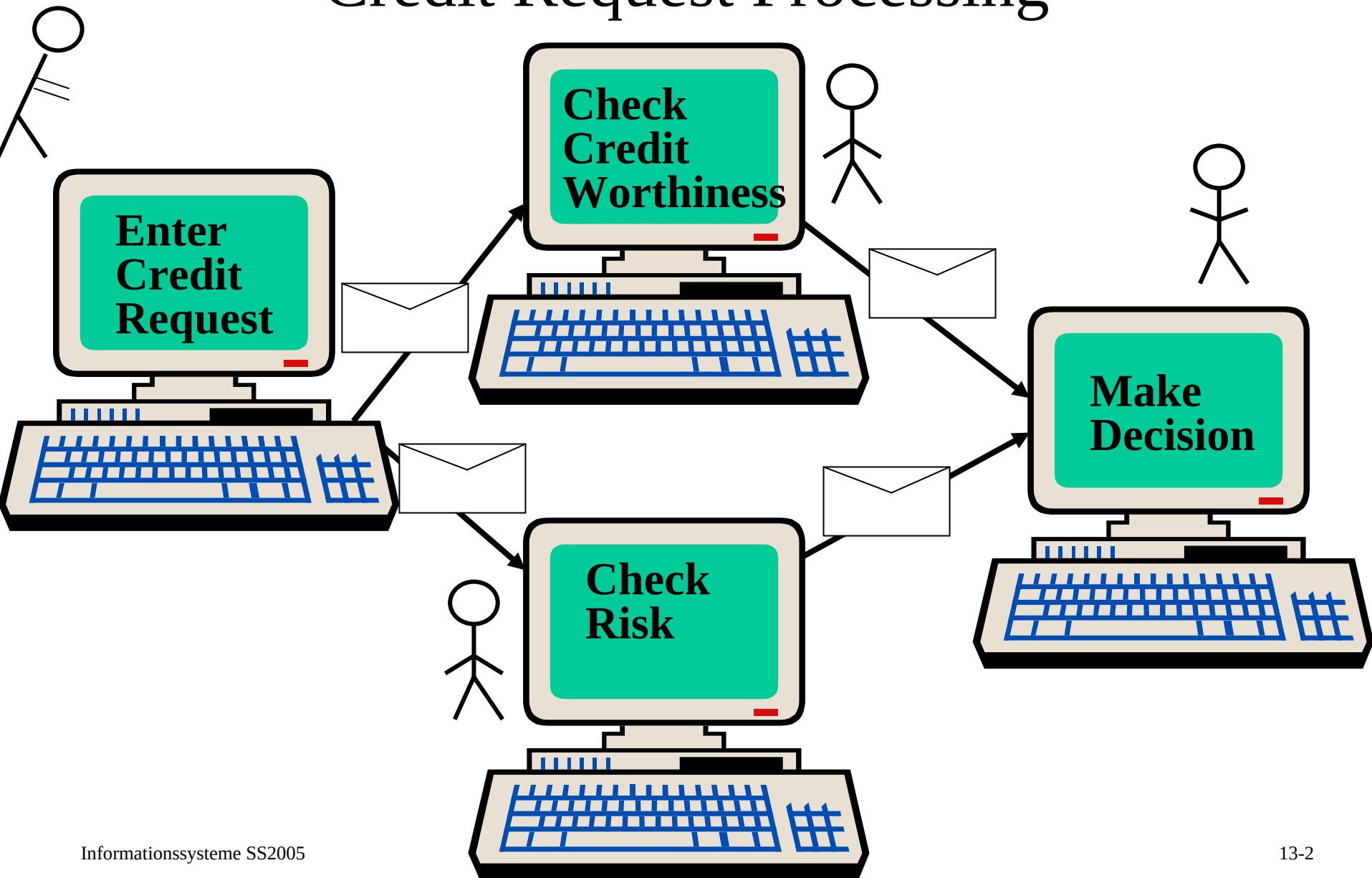
- **Statecharts**
- **Ereignis-Prozess-Ketten**

13.2 Workflow-Management für Geschäftsprozesse

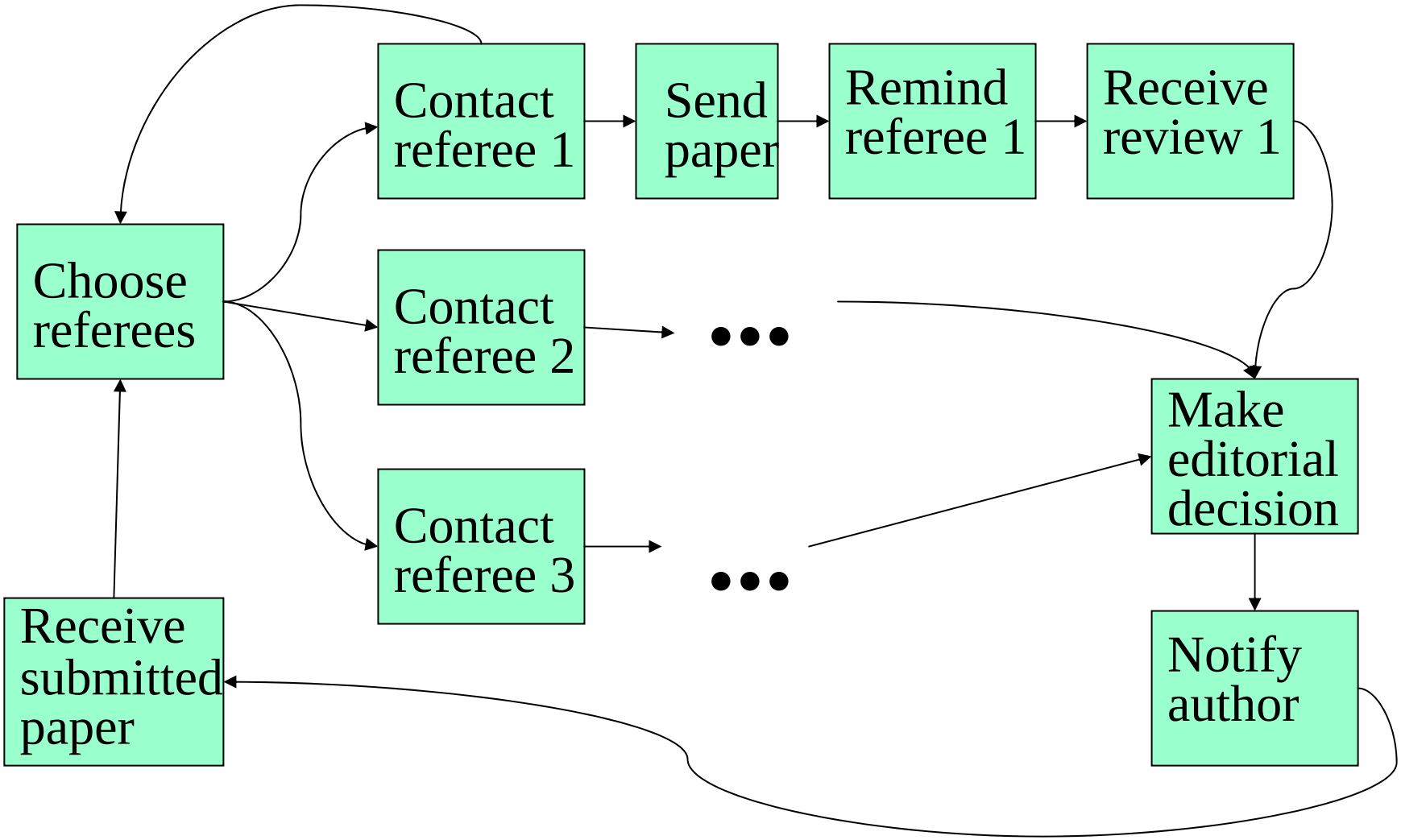
13.3 Semantik von Statecharts

13.4 Eigenschaften von Statecharts und deren Verifikation

Workflow Application Example 1: Credit Request Processing



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:

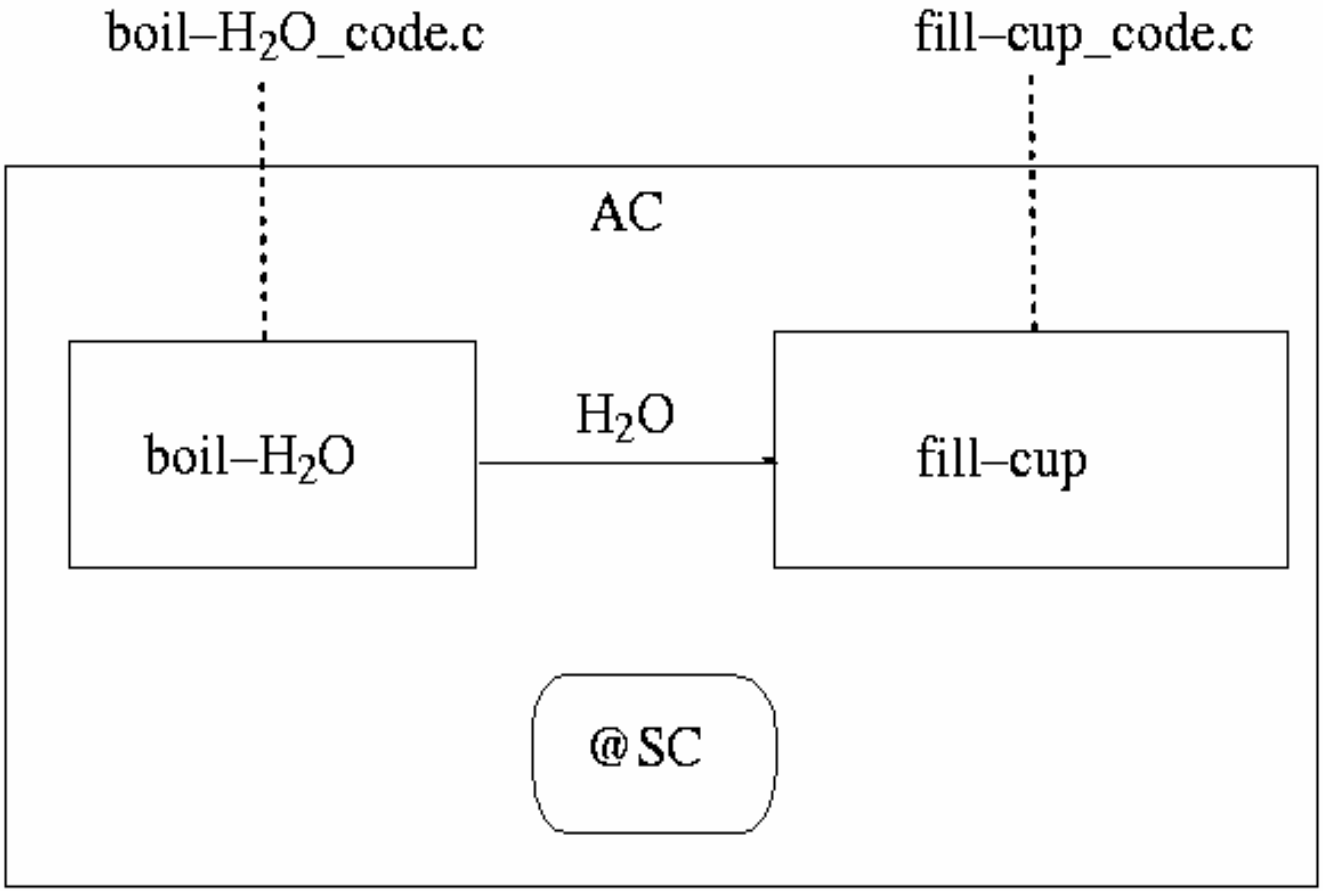
➡ ***Statecharts*** (Harel et al. 1987)

(alt.: Petri Net variants, temporal logic, process algebra, script language)

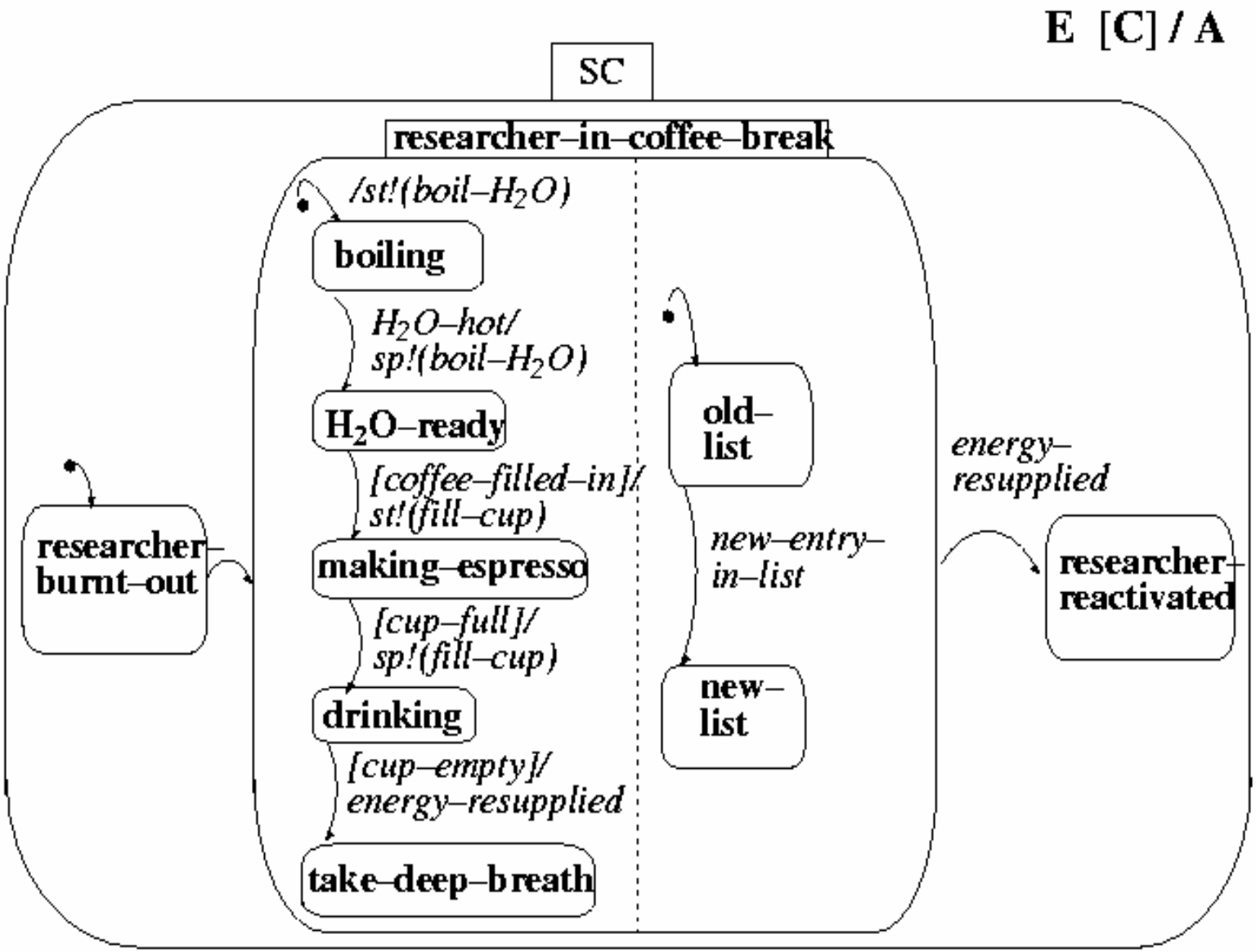
➡ Import / export
BPR tools → WFMS ↔ WFMS

➡ Statecharts included in UML industry standard (Unified Modeling Language, OMG 1997))

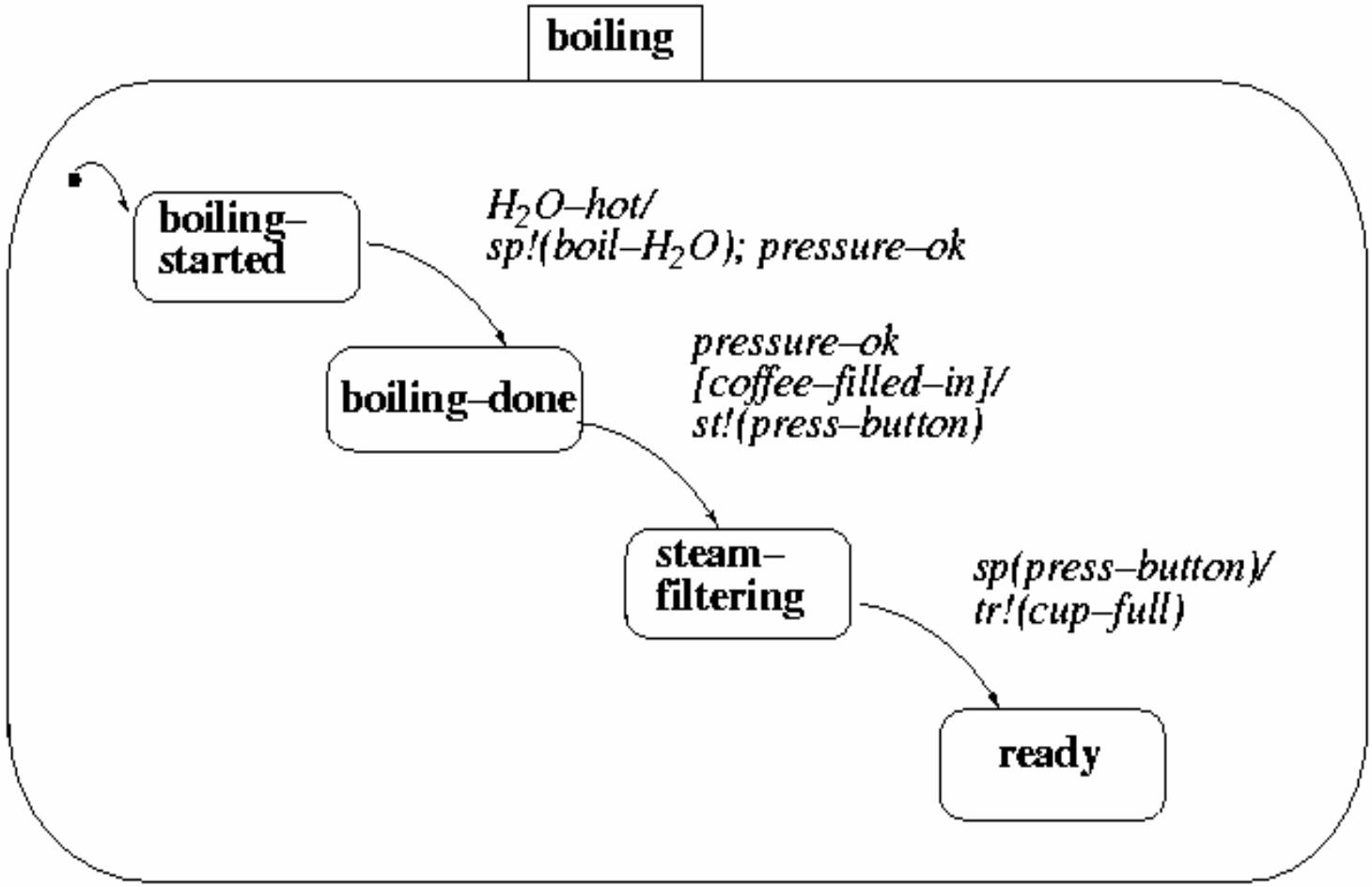
Example of Activitychart



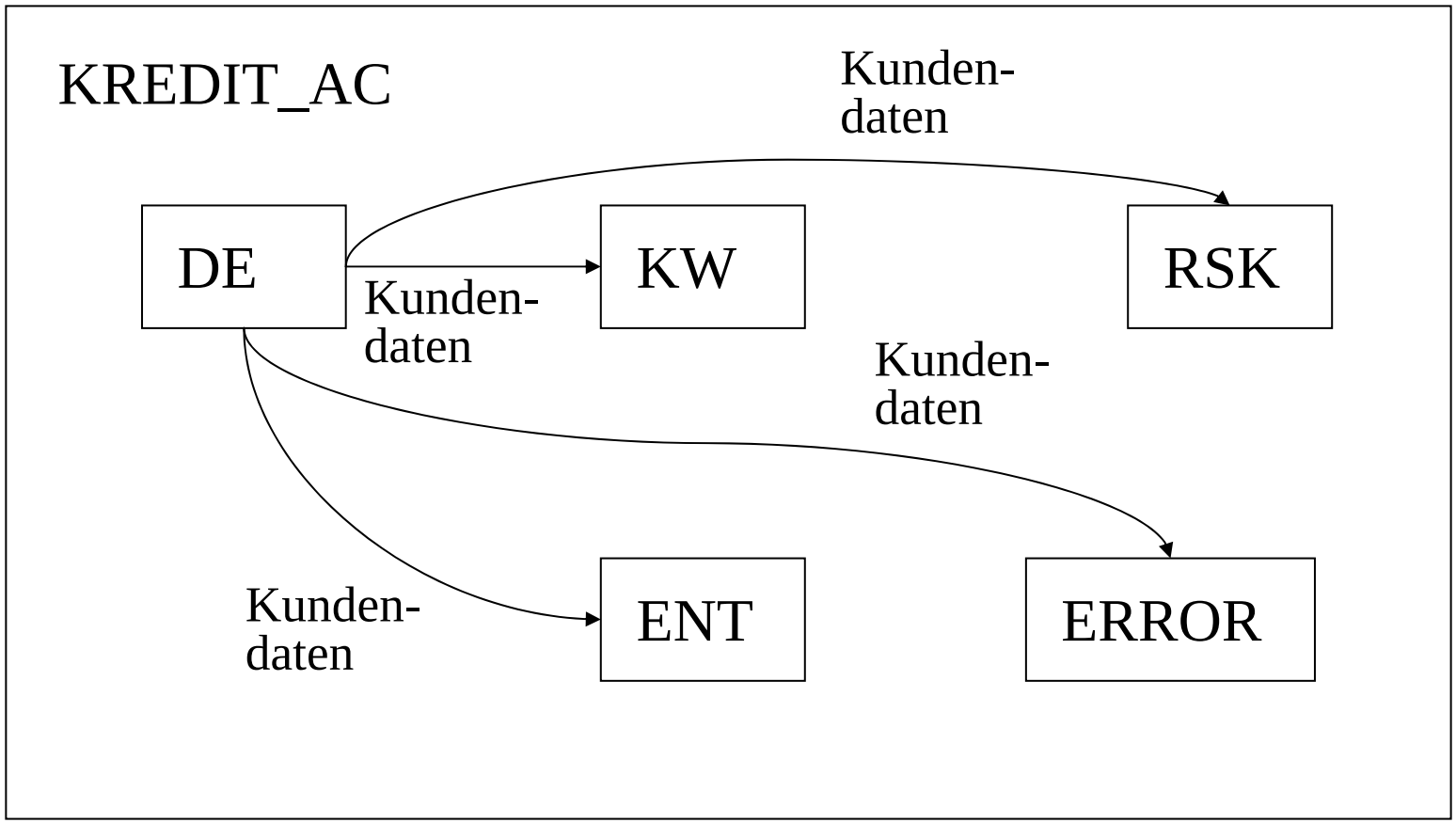
Example of Statechart



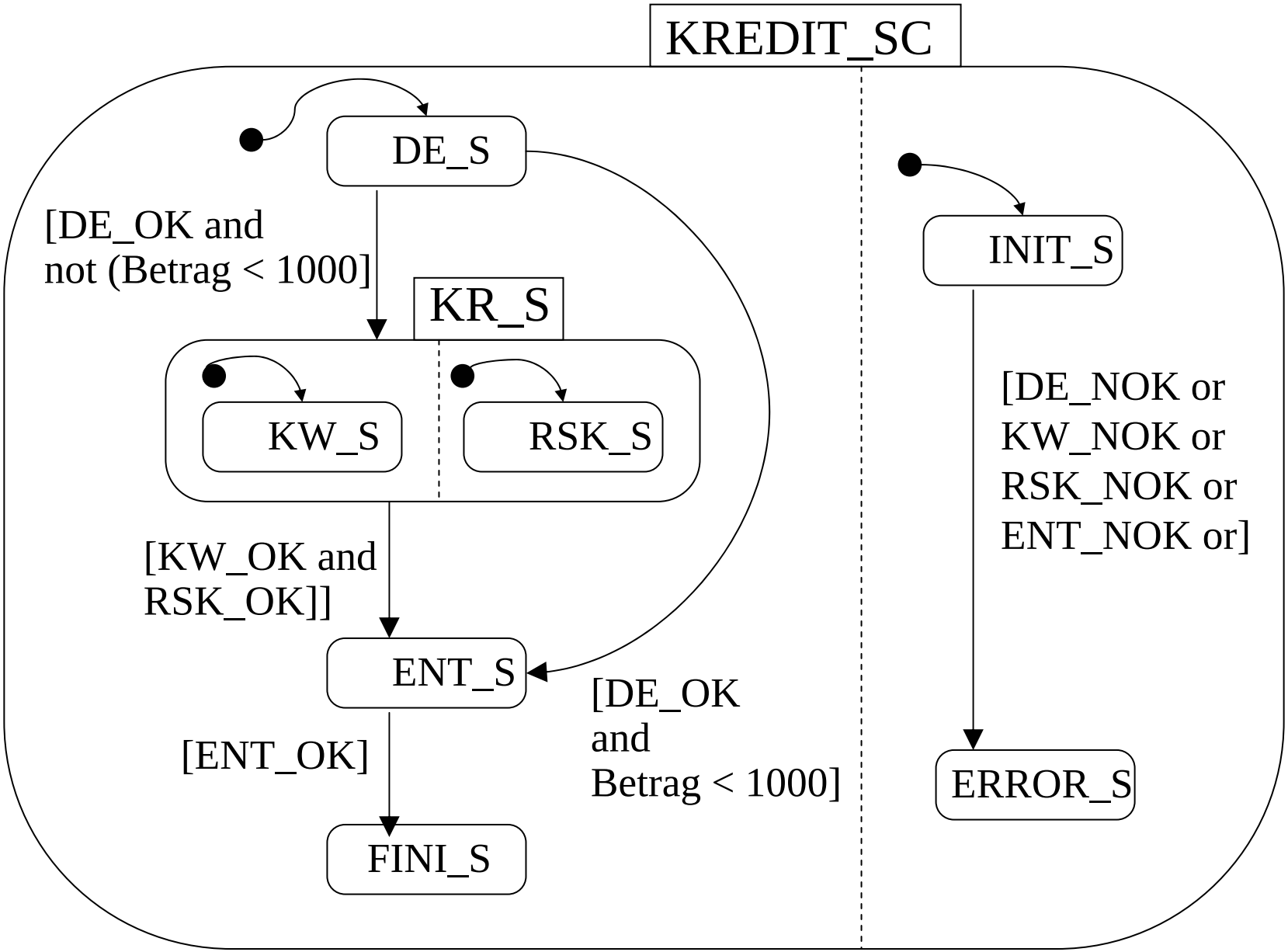
Refinement of States



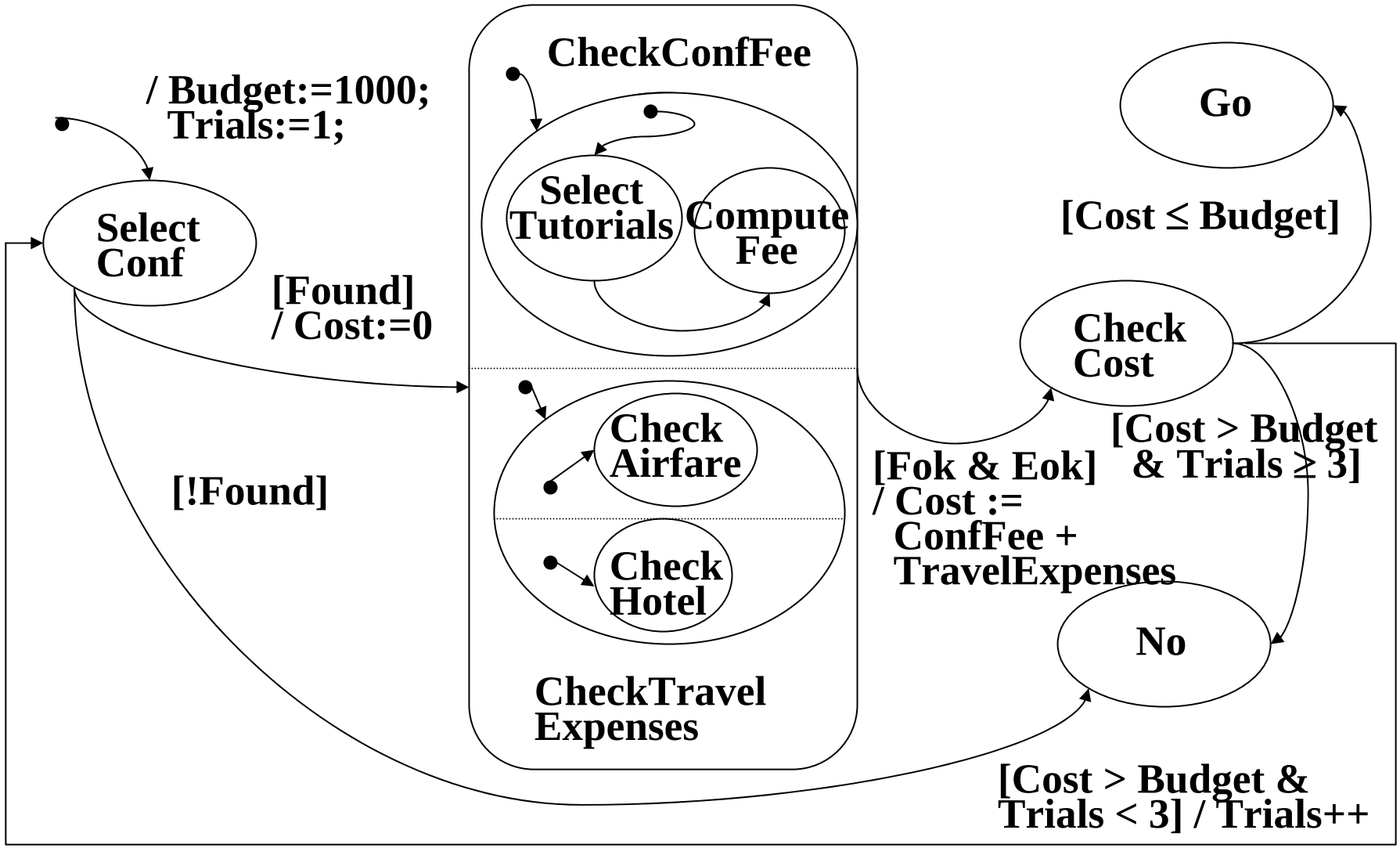
Activitychart Example 1



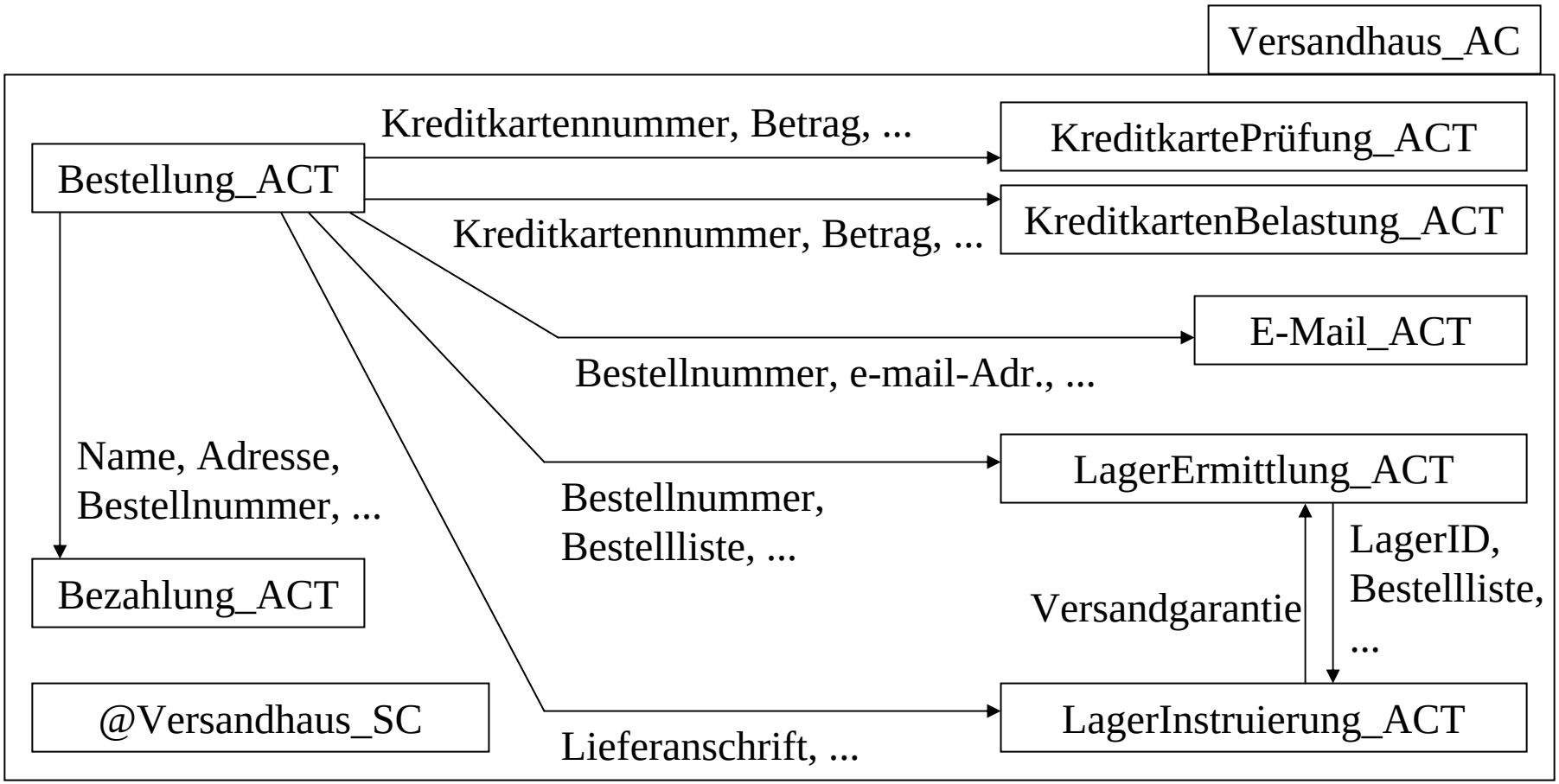
Statechart Example 1



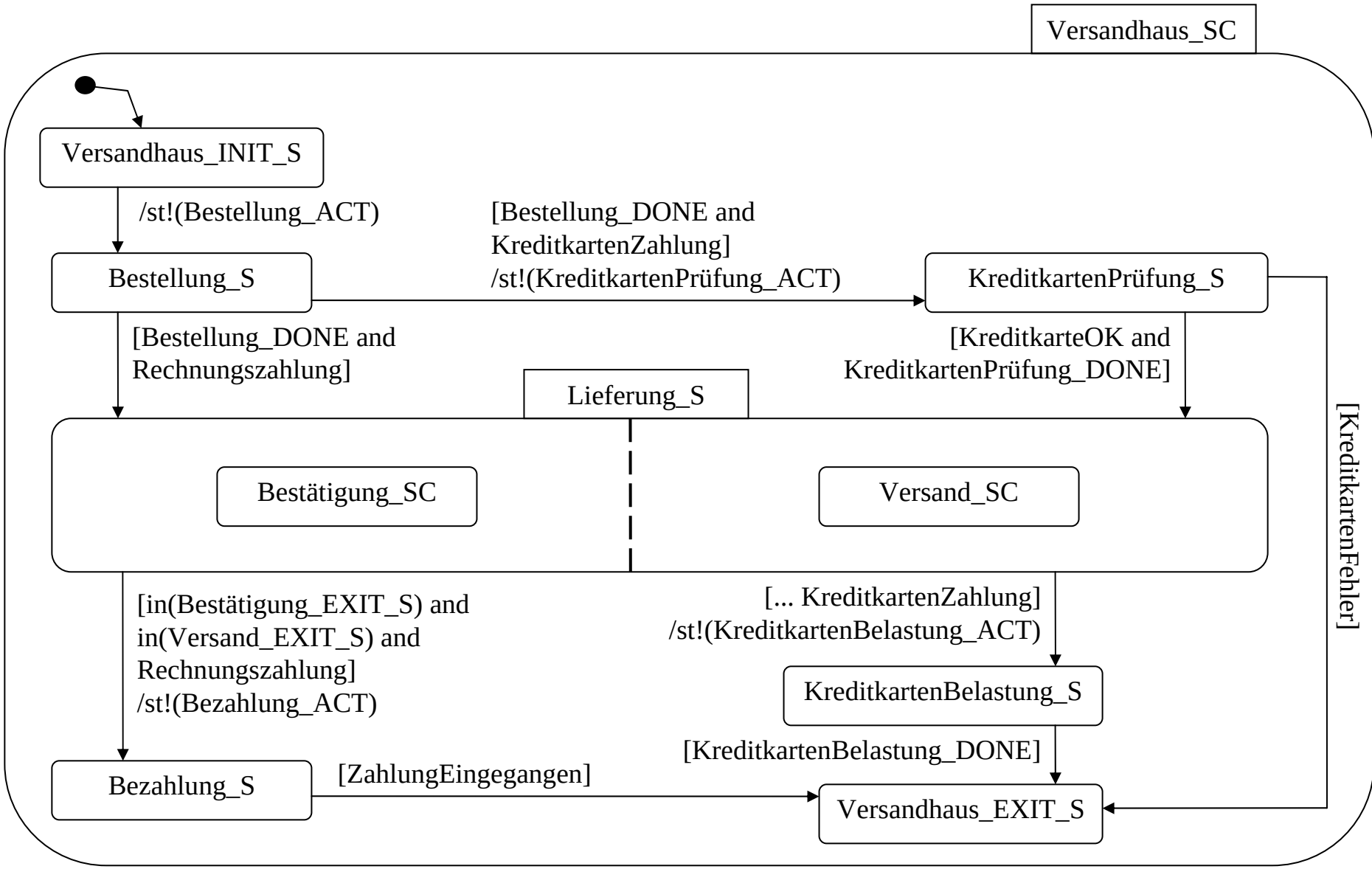
Statechart Example 2



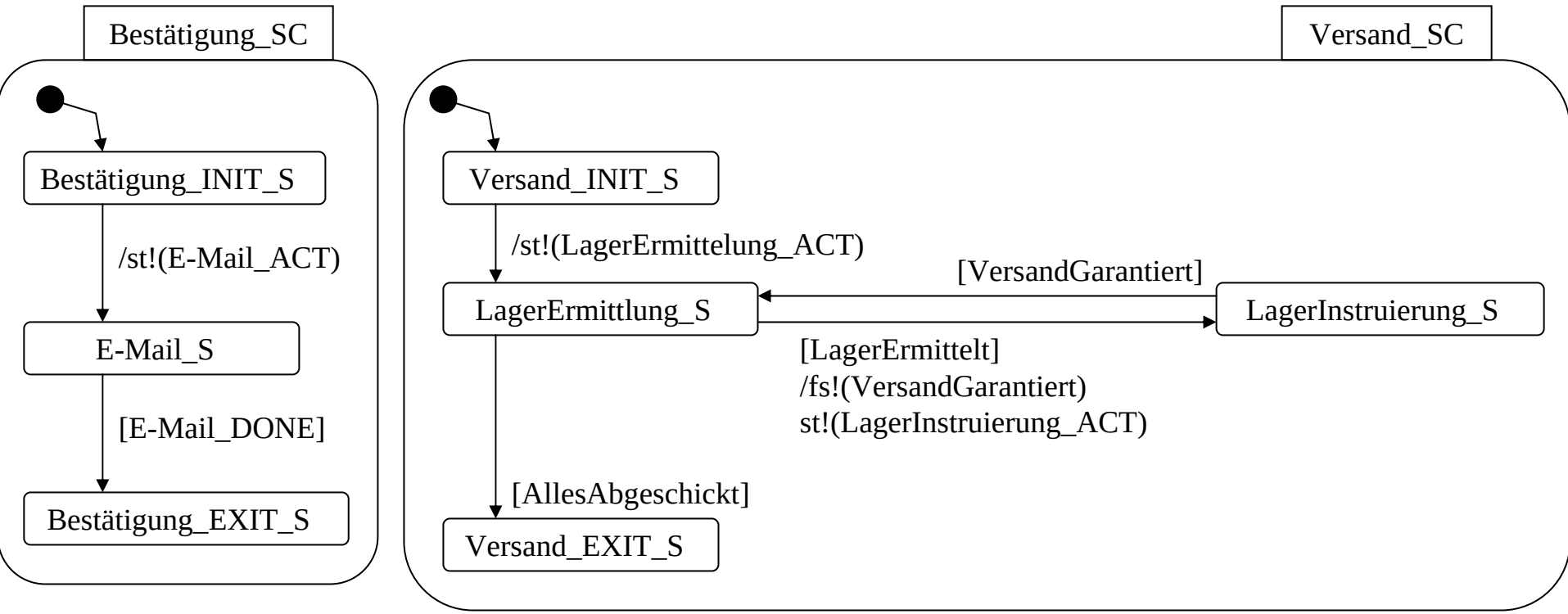
E-Commerce Workflow: Activitychart



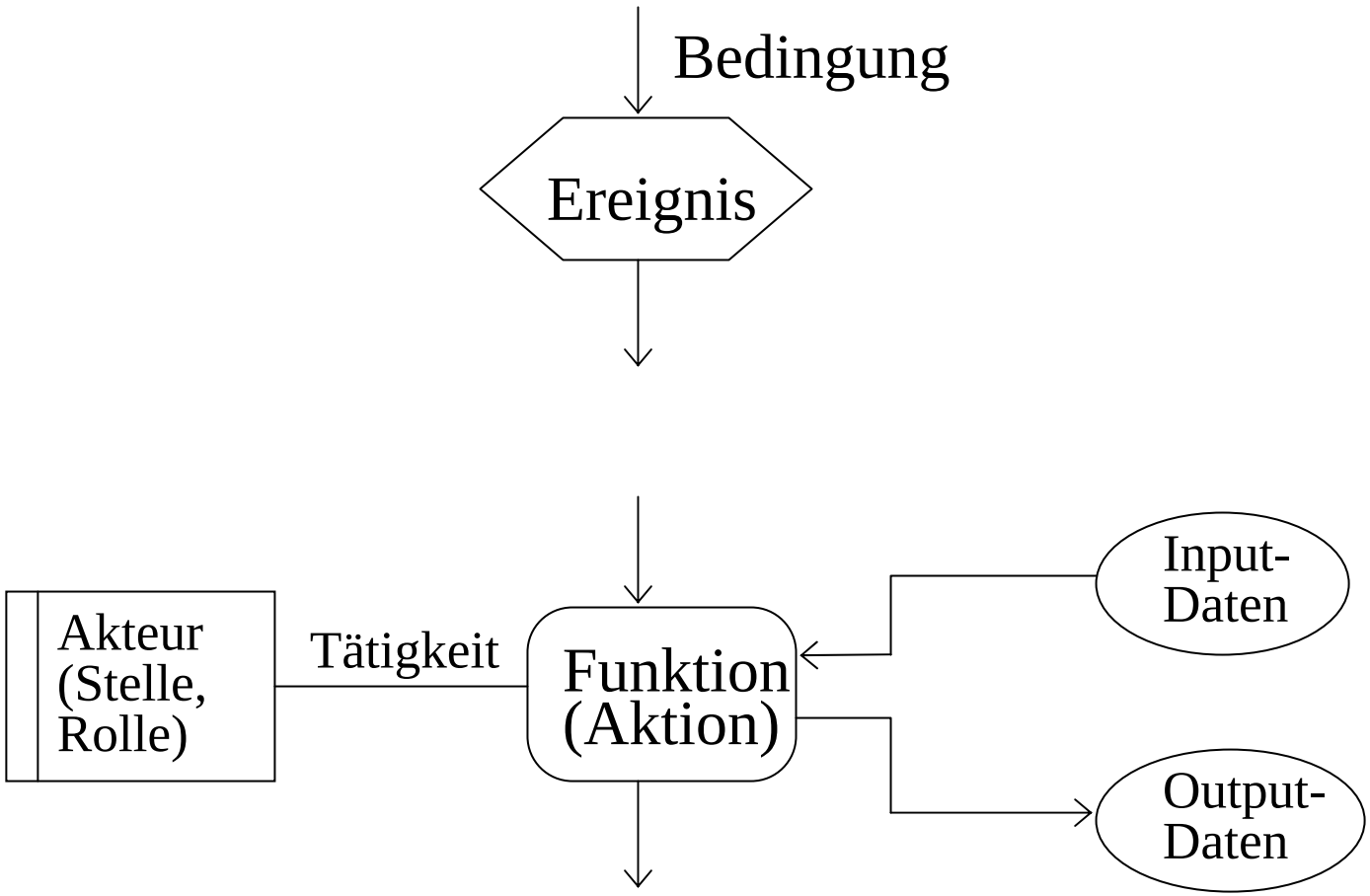
E-Commerce Workflow: Statechart



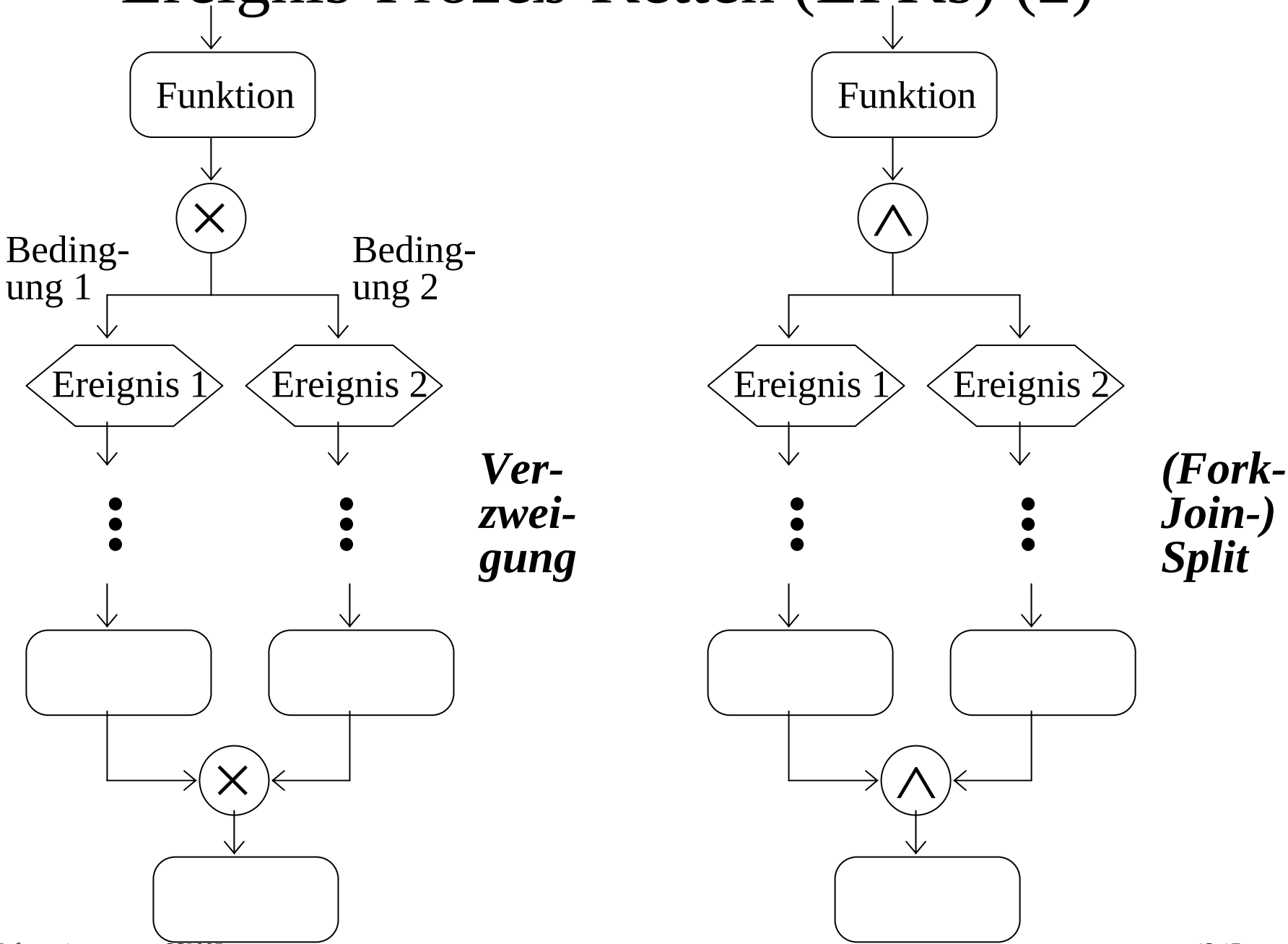
E-Commerce Sub-Workflows



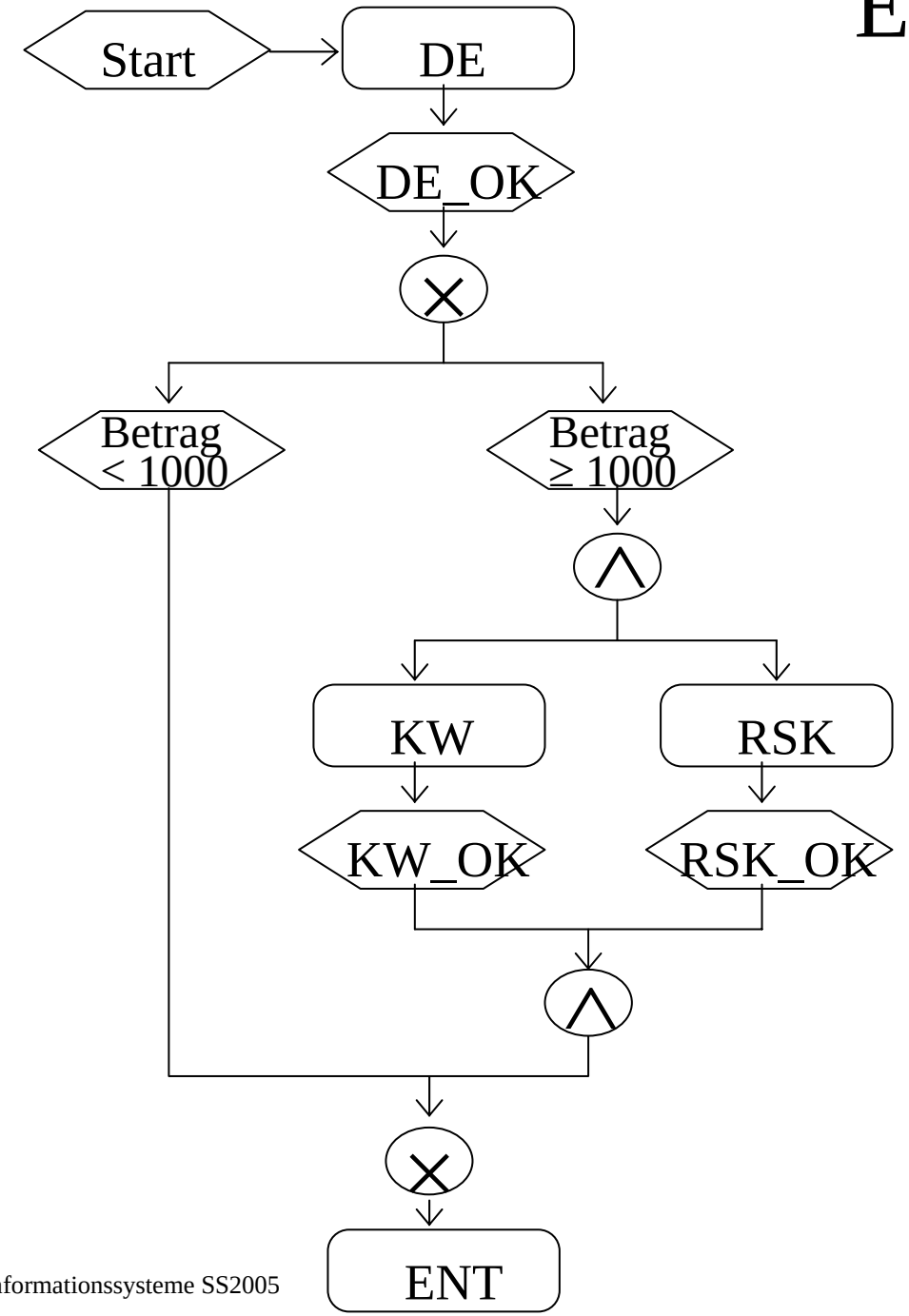
Ereignis-Prozeß-Ketten (EPKs) (1)



Ereignis-Prozeß-Ketten (EPKs) (2)

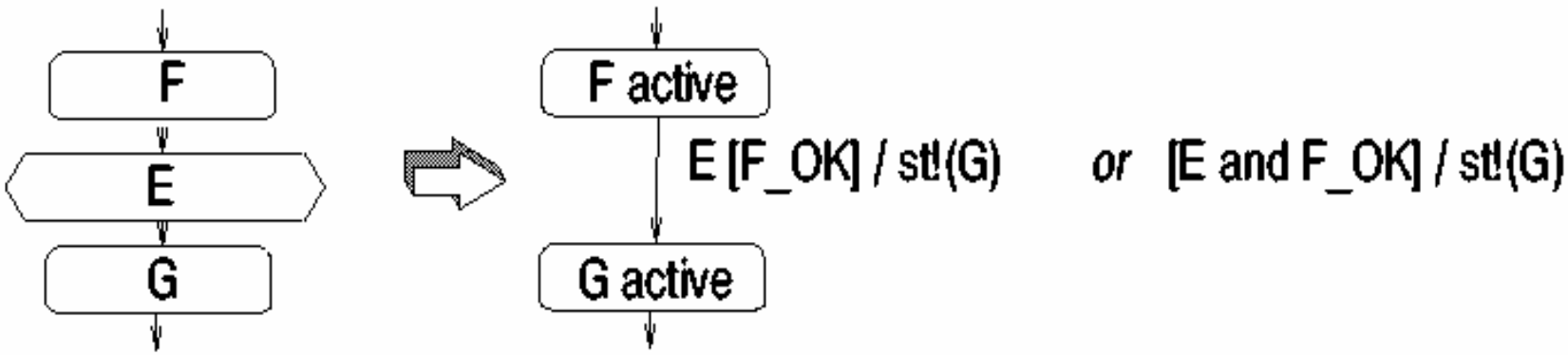


EPK-Beispiel



Import from BPR Tools

Principle:

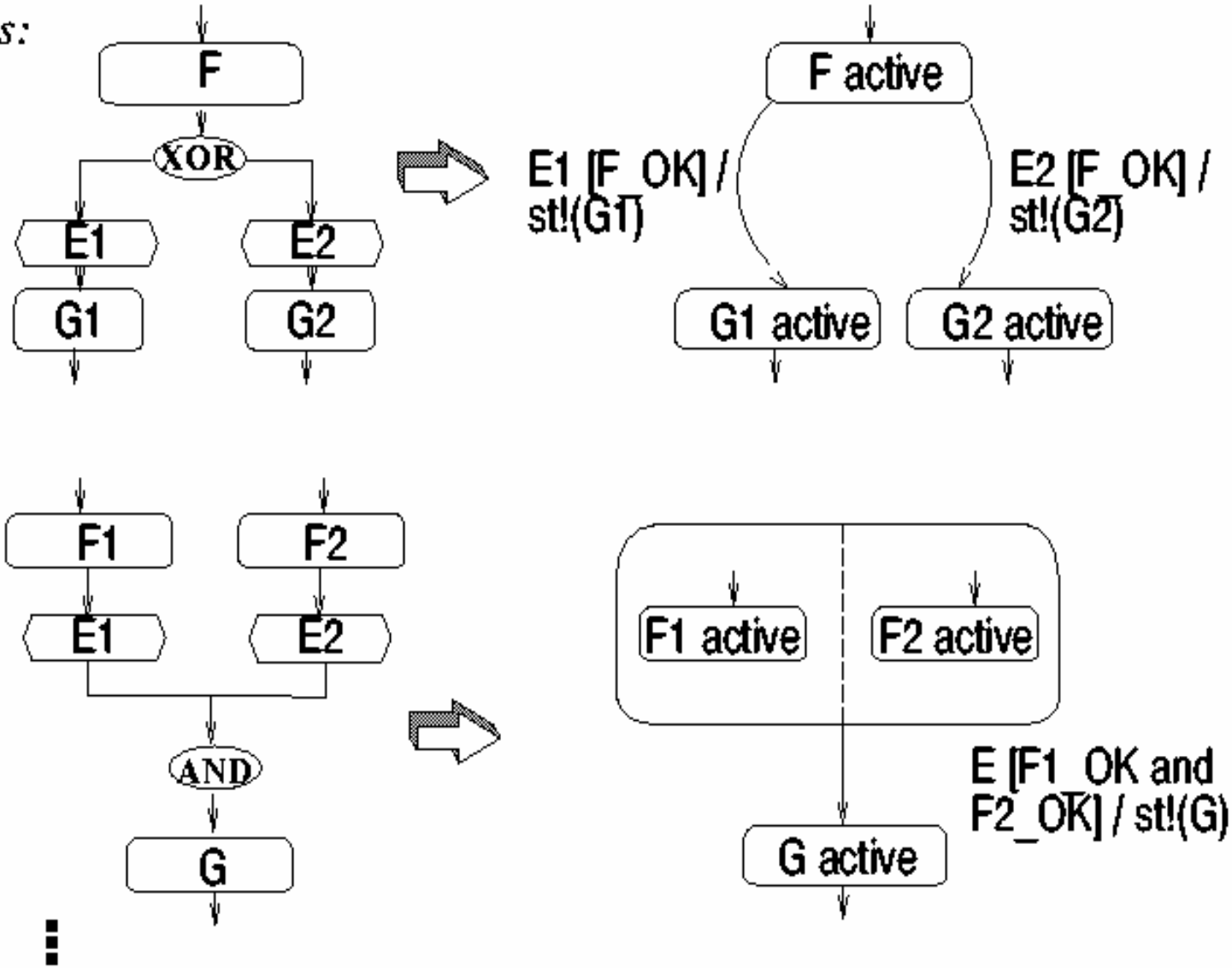


Event process chains
(EPCs à la Aris Toolset):

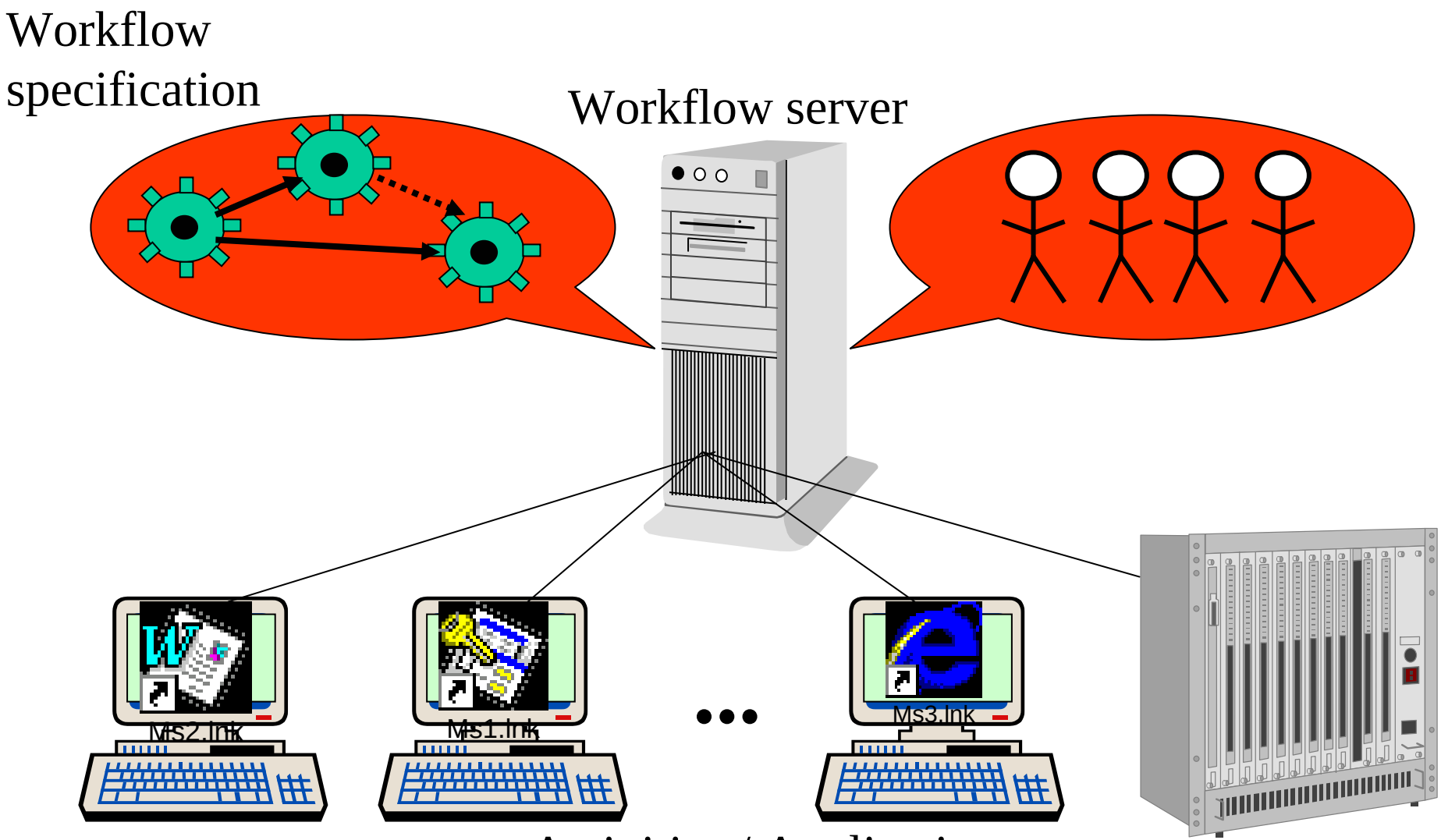
- process decomposed into functions
- completed functions raise events
  that trigger further functions
- control-flow connectors

Import from BPR Tools (continued)

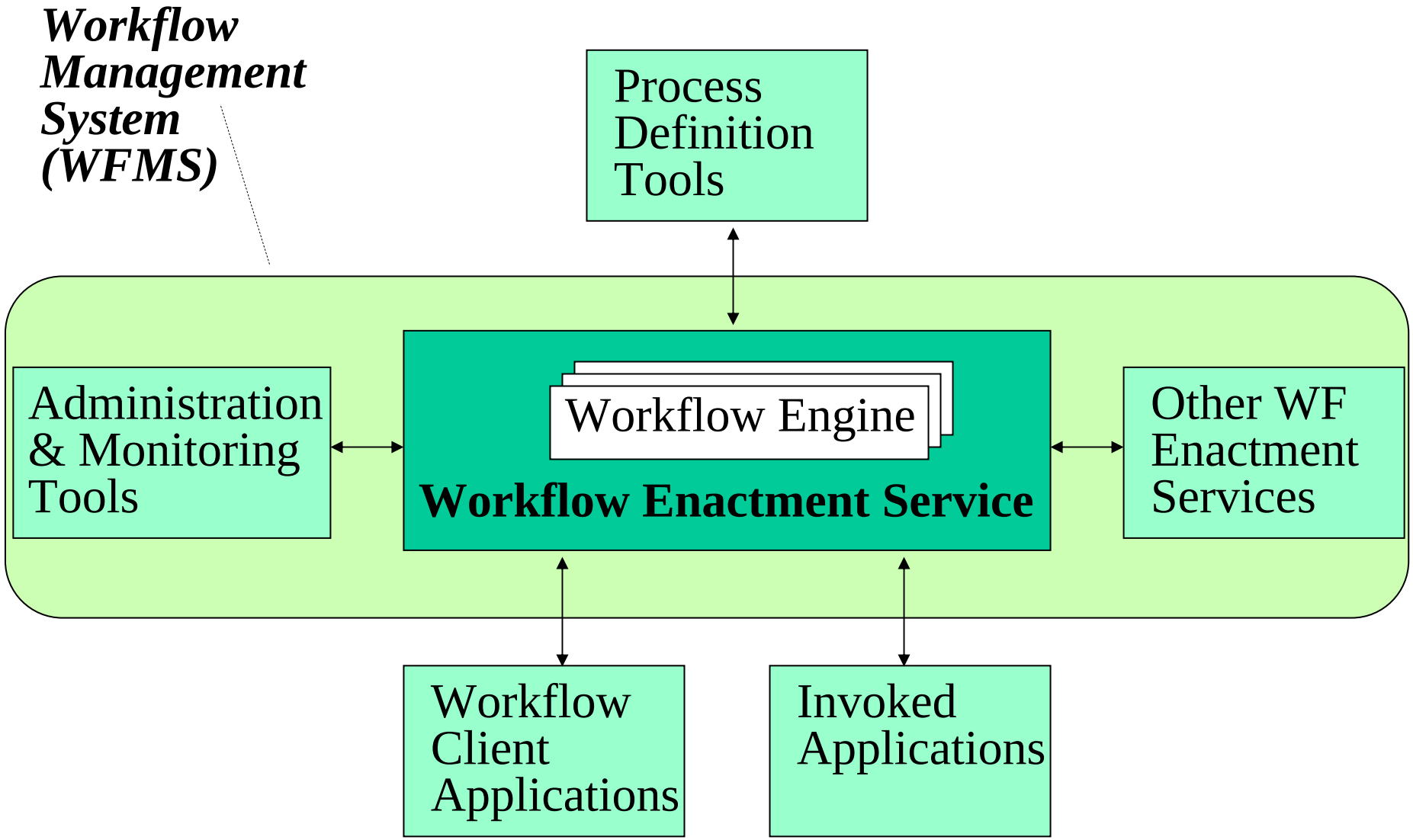
Some Subtleties:



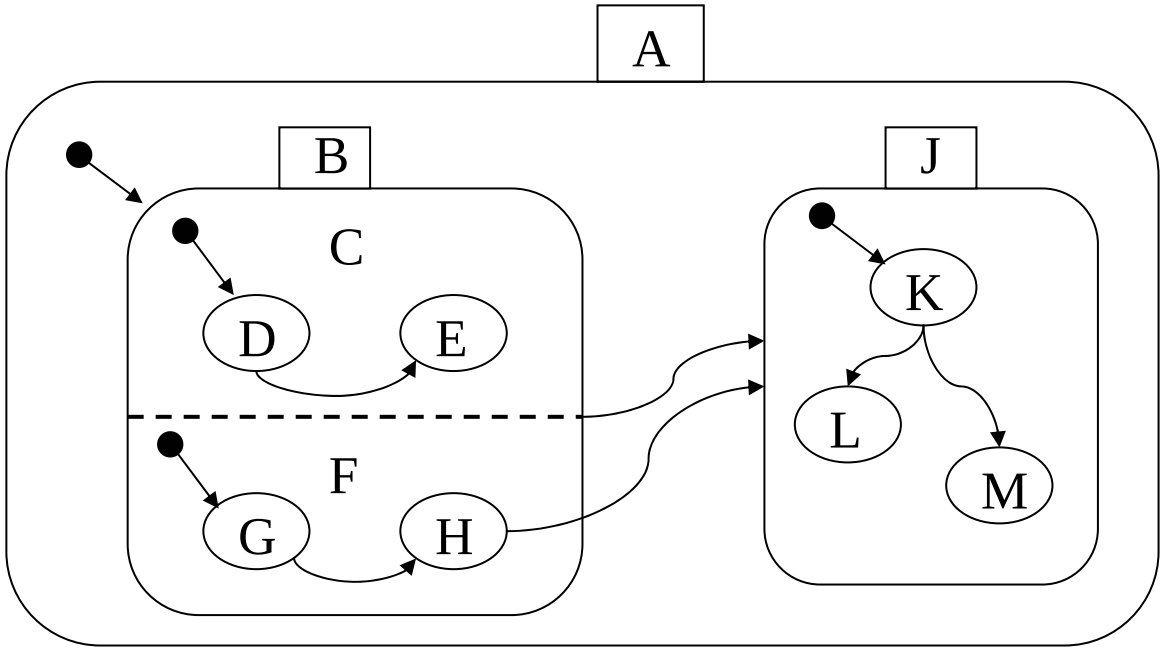
13.2 Workflow Management System Architecture



WfMC Reference Architecture



13.3 Abstract Syntax of Statecharts (1)



State set S

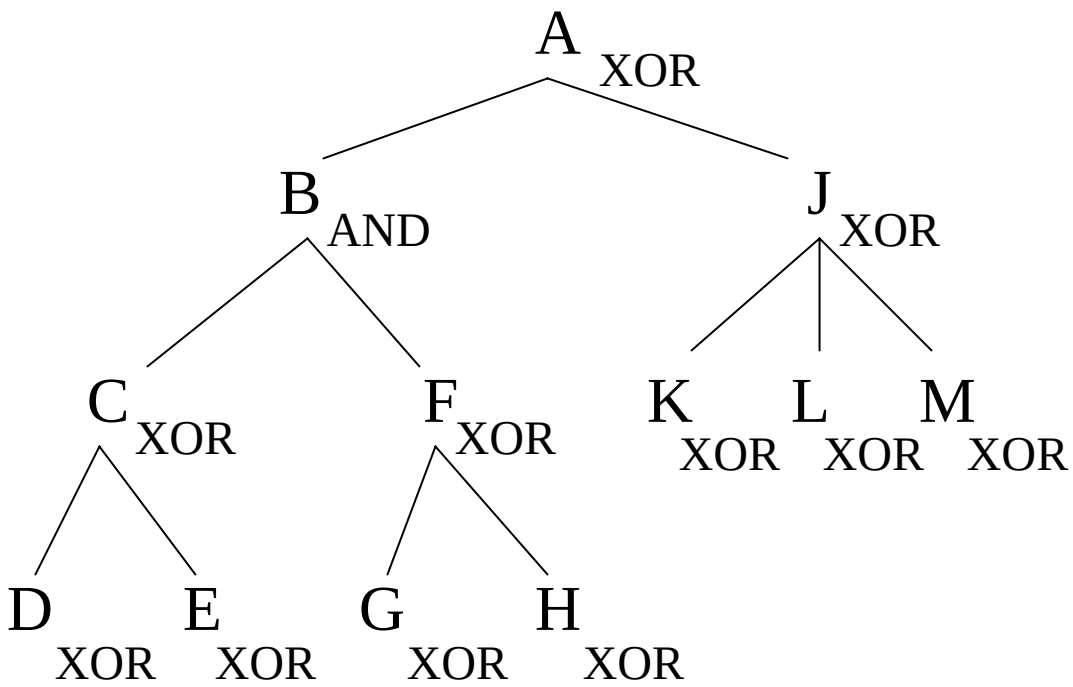
State tree (with node types AND or XOR)

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 $\rightarrow$ Dom

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

Initial configuration

Operational Semantics of Statecharts (2)

Evaluation of expression in configuration:
eval (expr, conf) defined inductively

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

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

Operational Semantics of Statecharts (3)

for transition t:

- a = lca (source(t), target(t))
- src(t) = child of a in subtree of source(t)
- tgt(t) = child of a in subtree of target(t)

when t fires:

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

Operational Semantics of Statecharts (4)

For a given configuration conf = (states, val) a **successor configuration** conf‘ = (states‘, val‘) is derived by selecting one transition t from fire(conf) with the effect:

- states‘ = states – source*(t) ∪ target*(t)
- val‘ captures the effect of action(t) and equals val otherwise

The operational semantics of a statechart (S,V,T) is the set of all possible executions along configurations $conf_0$, $conf_1$, $conf_2$, ... with

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

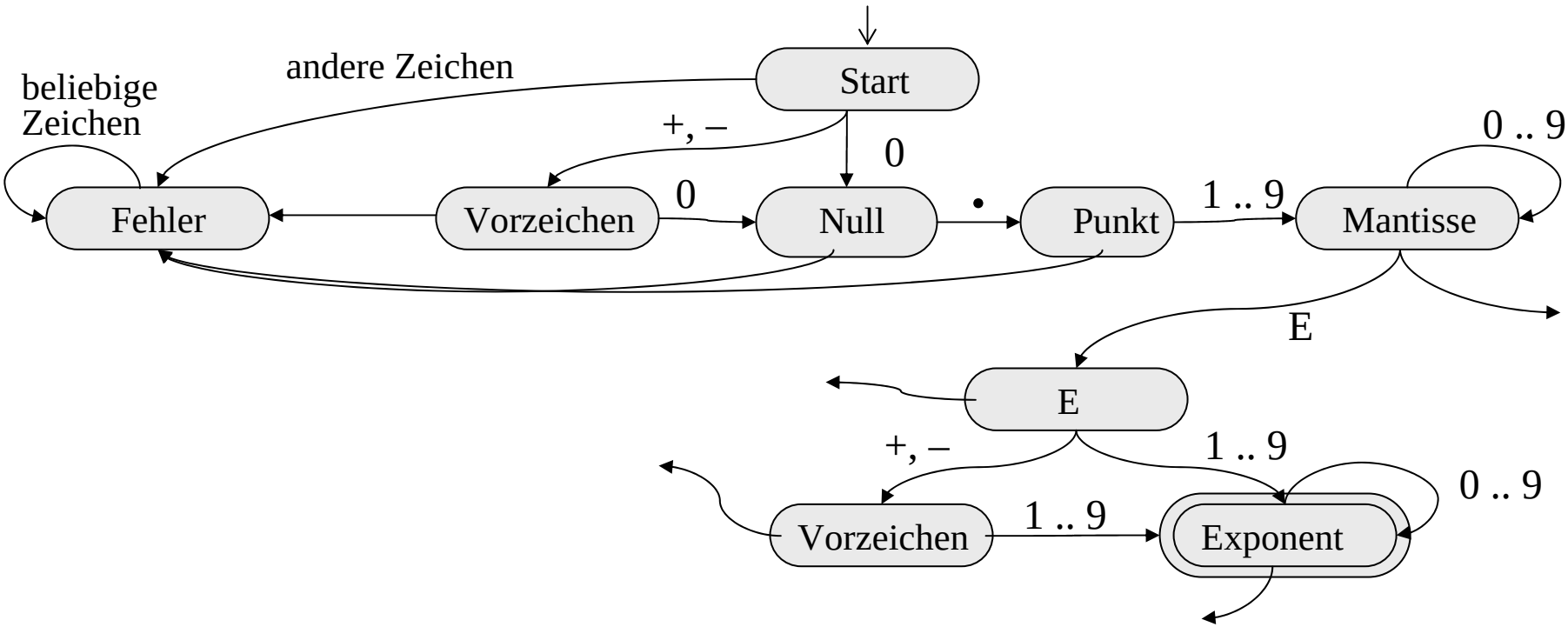
Digression: Finite State Automata

Definition:
Ein endlicher Automat (finite state automaton) ist ein 5-Tupel
$M = (Z, \Sigma, \delta, z0, E)$ 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 z0
- einer Menge von Endzuständen $E \subseteq Z$

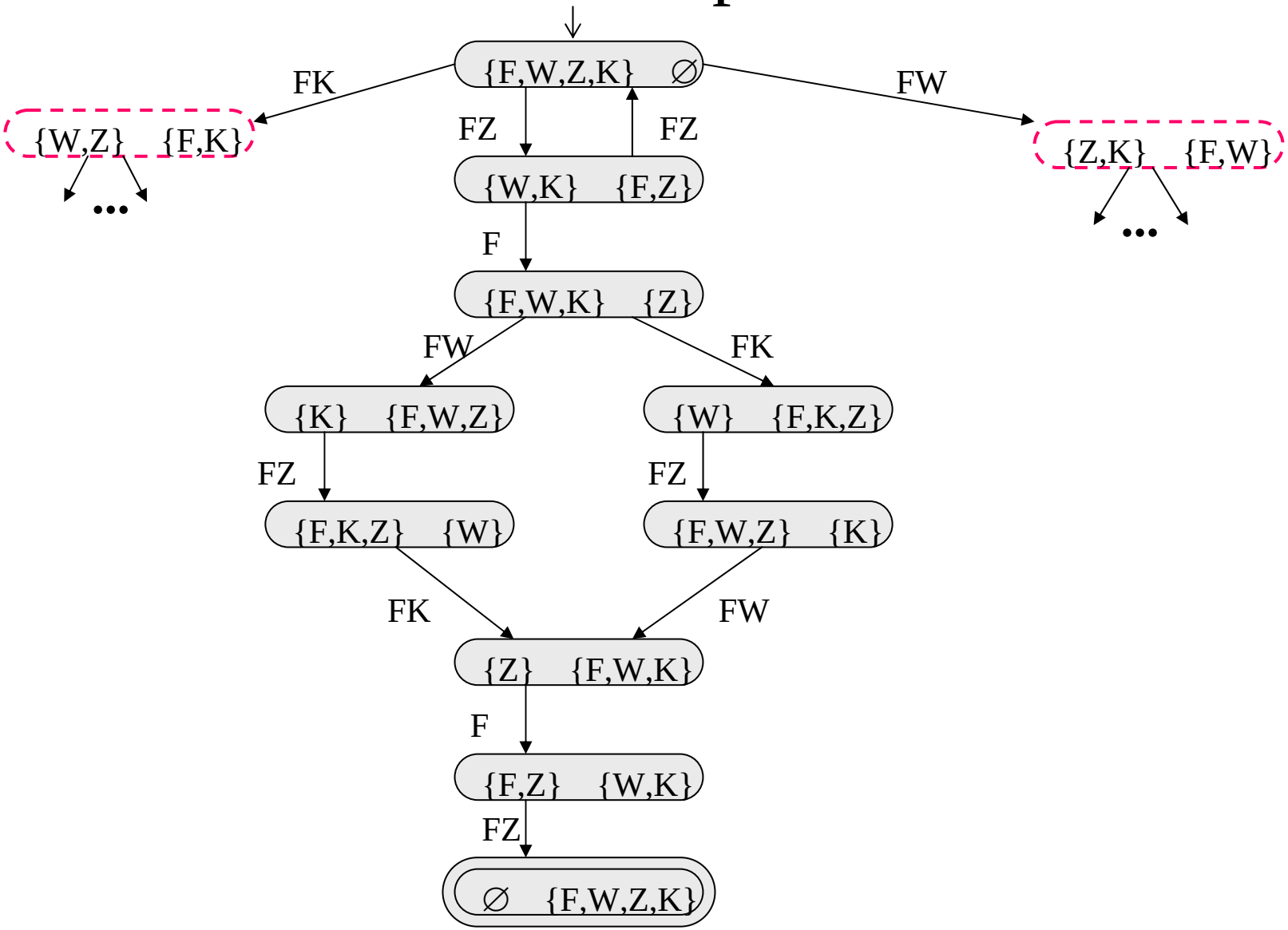
M geht in $z \in Z$ mit Eingabe $x \in \Sigma$ in $\delta(z,x) \in Z$ über.
δ wird homomorph zur Funktion $\delta^*: Z \times \Sigma^* \rightarrow Z$ erweitert:
$\delta^*(z, au) = \delta^*(\delta(z,a),u)$ mit $z \in Z$, $a \in \Sigma$, $u \in \Sigma^*$.
Die Menge $L(M) = \{w \in \Sigma^* \mid \delta^*(z0,w) \in E\} \subseteq \Sigma^*$
ist die vom Automat M akzeptierte Sprache.

FSA Example 1



(offene Transitionen gehen zum Zustand „Fehler“)

FSA Example 2



Mapping Statecharts into FSAs

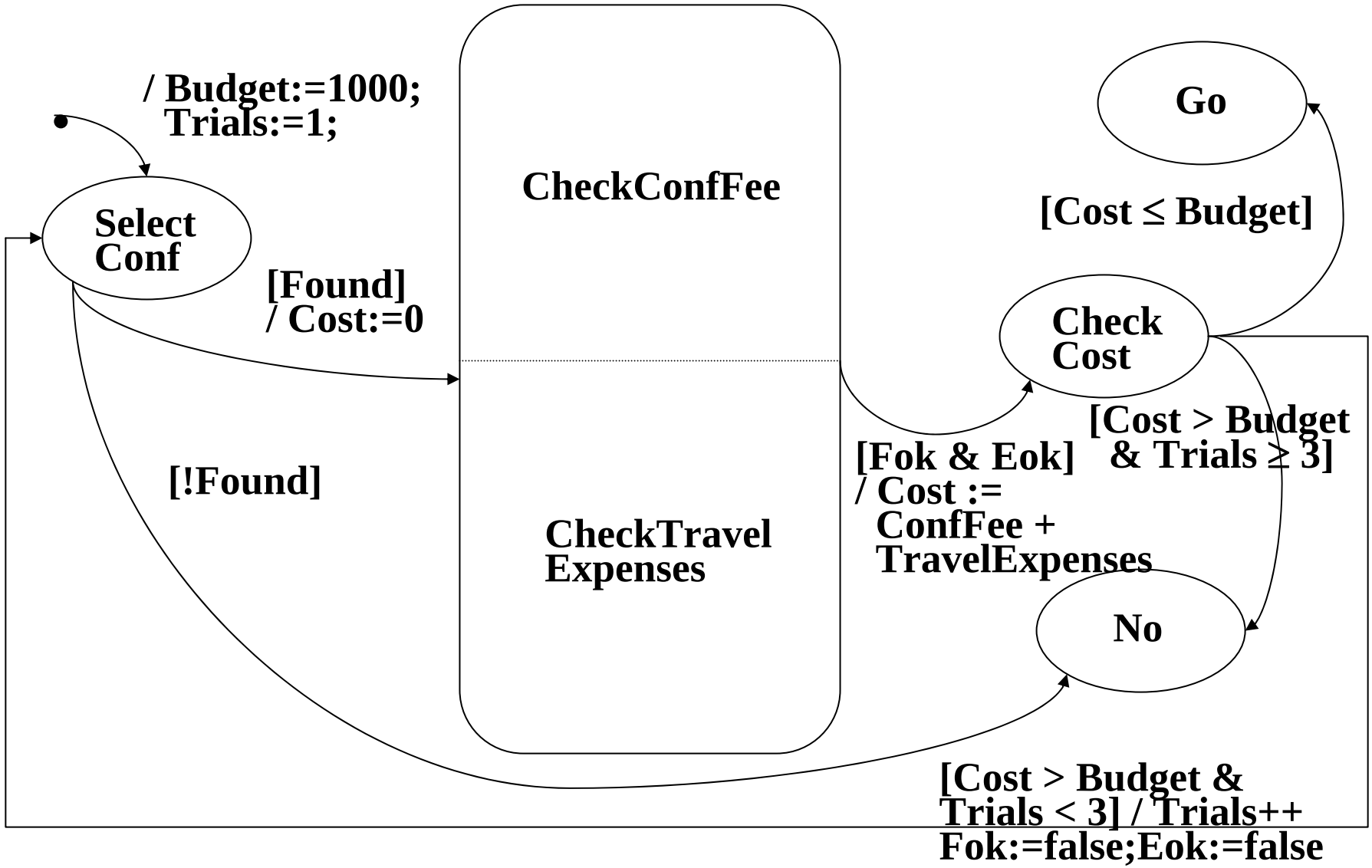
Represent SC configurations as states of a FSA:

Step 1:
abstract conditions on infinite-domain variables into Boolean vars
formal mapping: $\psi 1$: val $\rightarrow B1 \times B2 \times ... \times Bm$

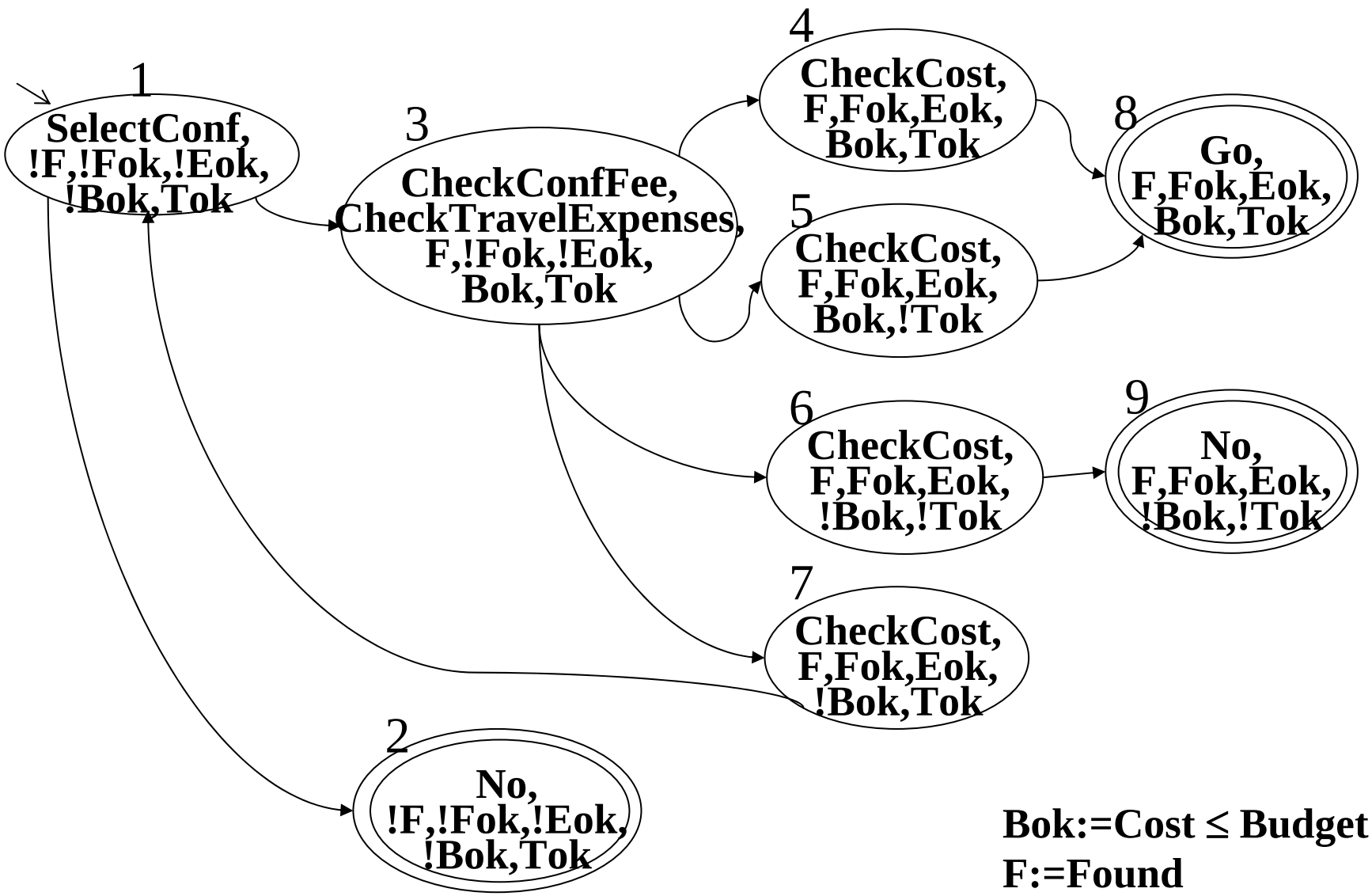
Step 2:
capture set of active SC states (in SC hierarchy and in components)
by powerset automaton $\psi 2$: states $\rightarrow 2^S =: Z$

Step 3:
encode SC context into extended state space of FSA
by an injective mapping $\psi 3$: $Z \times B1 \times B2 \times ... \times Bm \rightarrow Z'$
such that there is a transition from z1 to z2 in the FSA
iff $\psi 3^{-1}(z2)$ is a possible successor configuration of $\psi 3^{-1}(z1)$ in the SC

Example: From SC To FSA (1)



Example: From SC To FSA (2)



Bok:=Cost ≤ Budget
F:=Found
Tok:=Trials<3

13.4 Guaranteed Behavior and Outcome of Mission-critical Workflows

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

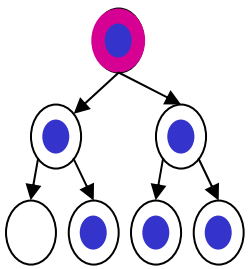
- **Safety** properties (invariants):
  nothing bad ever happens
- **Liveness** properties (termination, fairness, etc.):
  something good eventually happens

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

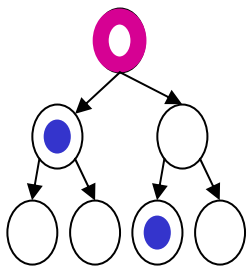
CTL: Computation Tree Logic

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

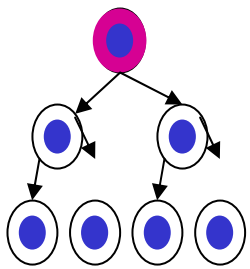
all next: AX p



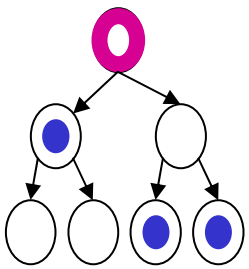
exists next: EX p



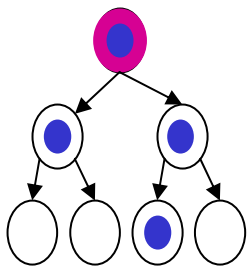
all globally: AG p



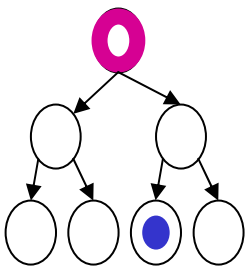
all finally (inevitably): AF p



exists globally: EG p

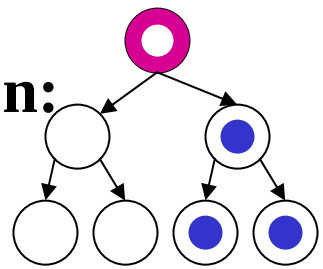


exists finally (possibly): EF p



combination:

EF AG p



Critical Properties of the Example Workflow

formalized in CTL (Computation Tree Logic)

- ***Can we ever exceed the budget ?***

  not EF (in(Go) and !Bok)

  ≡ AG (not in(Go) or Bok)

- ***Do we always eventually reach a decision ?***

  AF (in(Go) or in(No))

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

  EF ((in(CheckCost) and !Bok) => (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, s0, R, L) mit

- **einer endlichen Zustandsmenge S,**
- **einem Startzustand $s0 \in S$,**
- **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, s0, R, L) ist ein *Modell* einer Formel F, wenn $M,s0 \models F$.
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. p1, p2 von F in den Zuständen s von M, in Zeichen: M,s |= p, wie folgt sind:

(i) M,s |= p mit einer aussagenlogischen Formel p gilt g.d.w. $p \in L(s)$;
(ii) M,s |= $\neg$p g.d.w. nicht M,s |= p gilt;
(iii) M,s |= p1 $\wedge$ p2 g.d.w. M,s |= p1 und M,s |= p2;
(iv) M,s |= p1 $\vee$ p2 g.d.w. M,s |= p1 oder M,s |= p2;
(v) M,s |= EX p g.d.w. es $t \in S$ gibt mit $(s,t) \in R$ und M,t |= p;
(vi) M,s |= AX p g.d.w. für alle $t \in S$ mit $(s,t) \in R$ gilt: M,t |= p;
(vii) M,s |= EG p g.d.w. es $t_1, ..., t_k \in S$ 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 |= p für alle i;
(viii) M,s |= AG p g.d.w. für alle $t \in S$ mit $(s,t) \in R^*$ gilt: M,t |= p;
(ix) M,s |= EF p g.d.w. es $t \in S$ gibt mit $(s,t) \in R^*$ und M,t |= p;
(x) M,s |= 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' |= 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

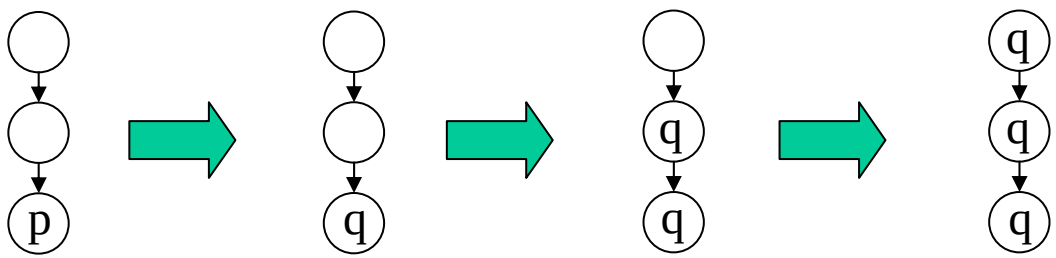
Model Checking: Fall EF

(vii) q hat die Form EF p:

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

(Fixpunktgleichung Q = P $\cup$ pred(Q))

```
Q := P;
Qnew := Q ∪ pred(Q);
while not (Q = Qnew) do
   Q := Qnew;
   Qnew := Q ∪ pred(Q);
od;
```

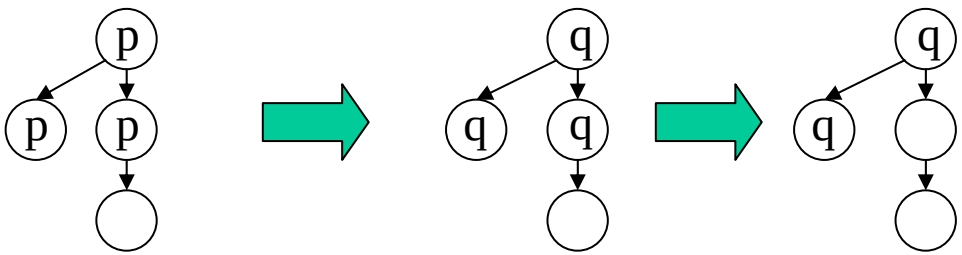


Model Checking: Fall EG

(viii) q hat die Form EG p:

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

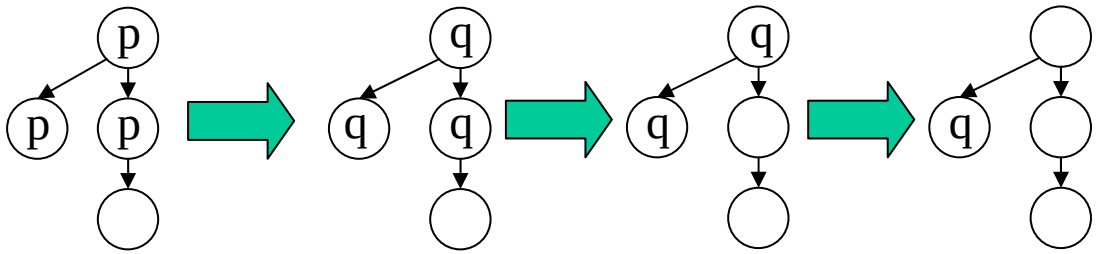
```
Q := P;
repeat
Qnew := Q;
for each s in Q do
      if s has successors and
          no successor of s is in Q
      then Q := Q - {s}; fi;
od;
until (Q = Qnew);
```



Model Checking: Fall AG

(ix) q hat die Form AG p:

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



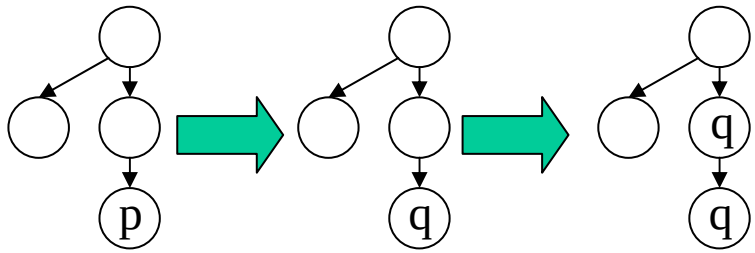

```
Q := P;
repeat
     Qnew := Q;
     for each s in Q do
          if s has successors and
             one successor of s is not in Q
          then Q := Q - {s} fi;
     od;
until (Q = Qnew);
```

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)

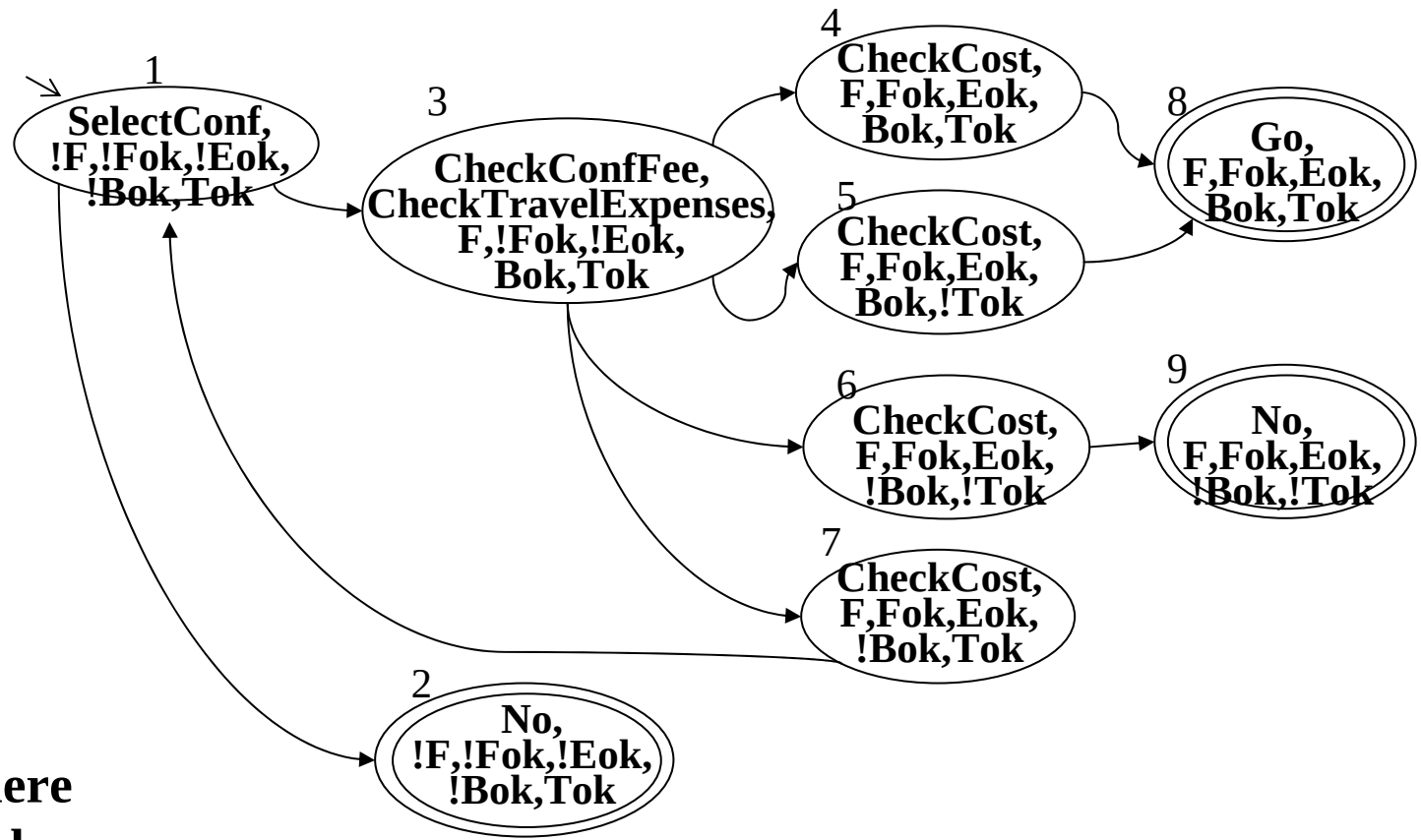



```
Q := P;
repeat
     Qnew := Q;
for each s in pred(Q) do
          if all successors of s are in Q
     then Q := Q ∪ {s}; fi;
od;
until (Q = Qnew);
```

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

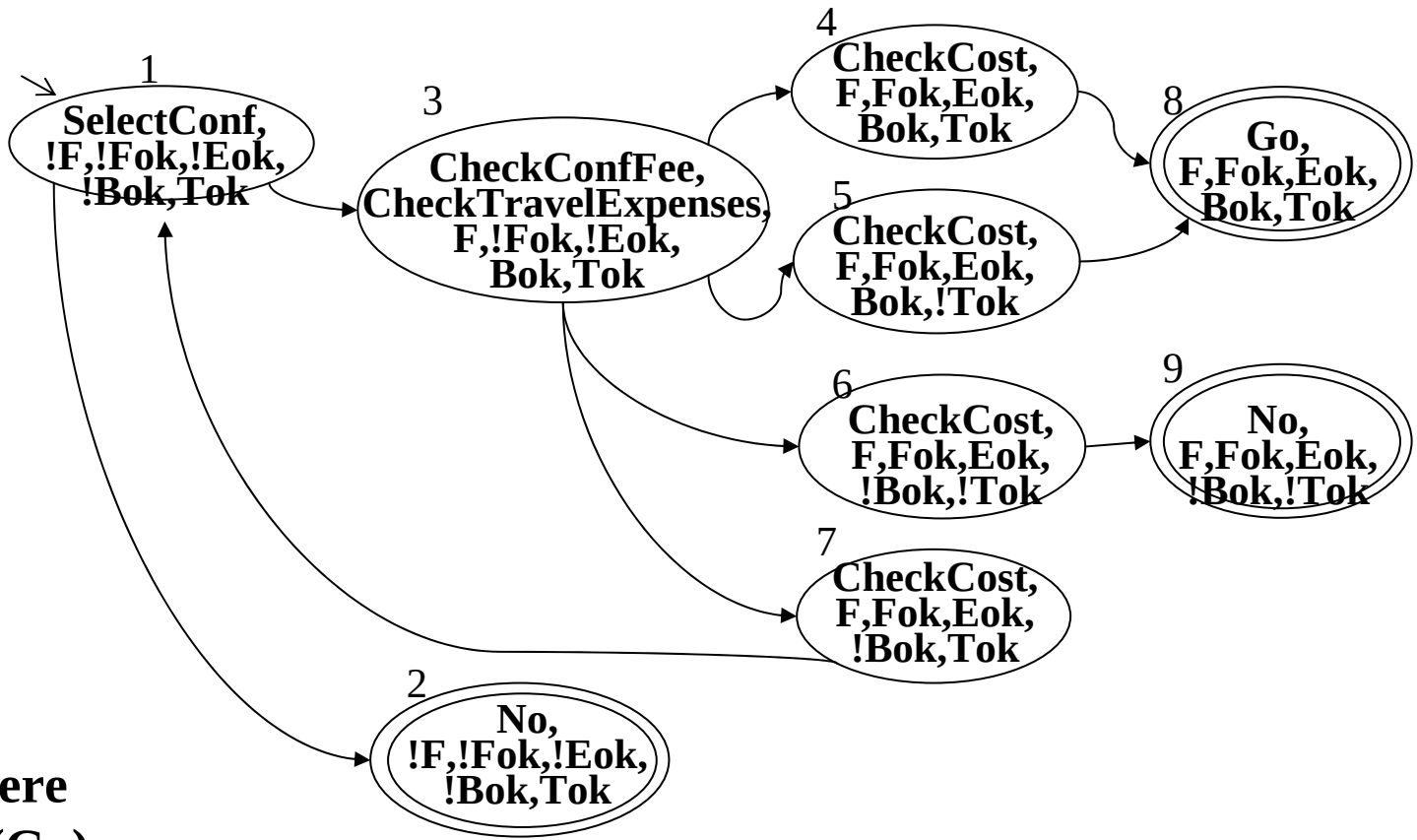
AG (not in(Go) or Bok)



Markiere
mit Bok:
mit in(Go):
mit ¬in(Go):
mit (¬Bok ⇒ ¬in(Go)):
mit AG (¬Bok ⇒ ¬in(Go)):

Model Checking: Beispiel 2

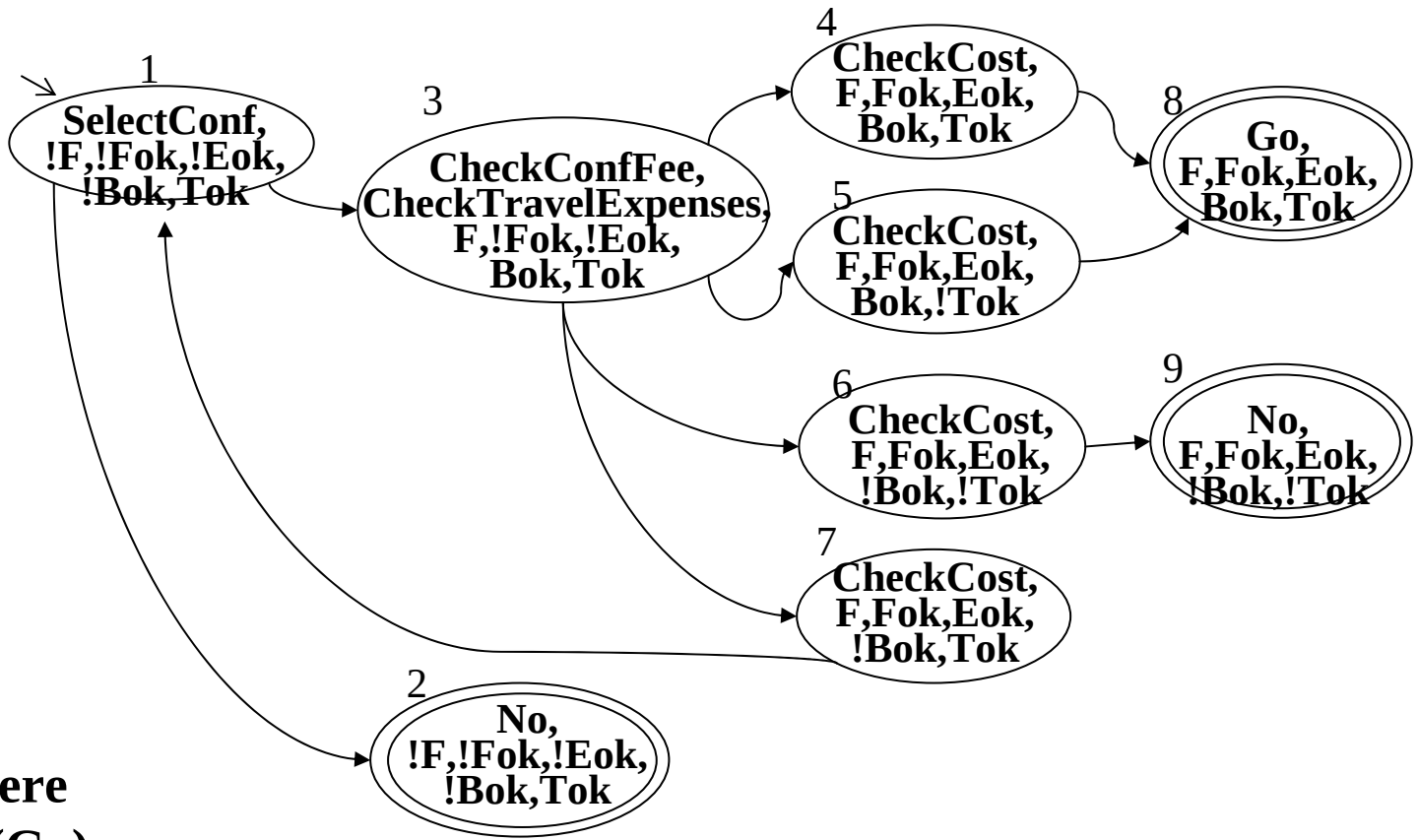
AF (in(Go) $\vee$ in(No))



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) and !Bok) => (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**

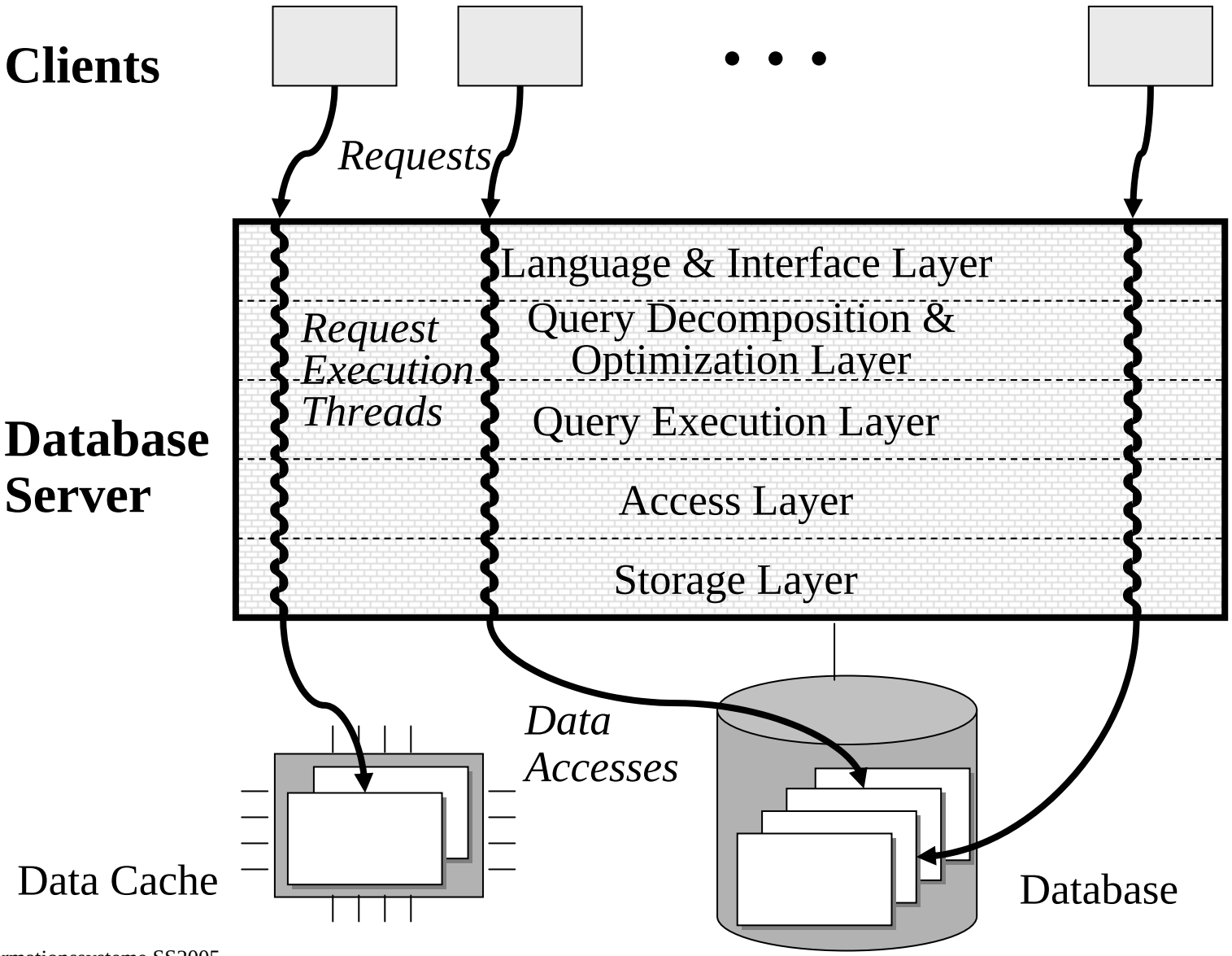
Kapitel 14: Datenspeicherung, Indexstrukturen und Anfrageausführung

14.1 Wie Daten gespeichert werden

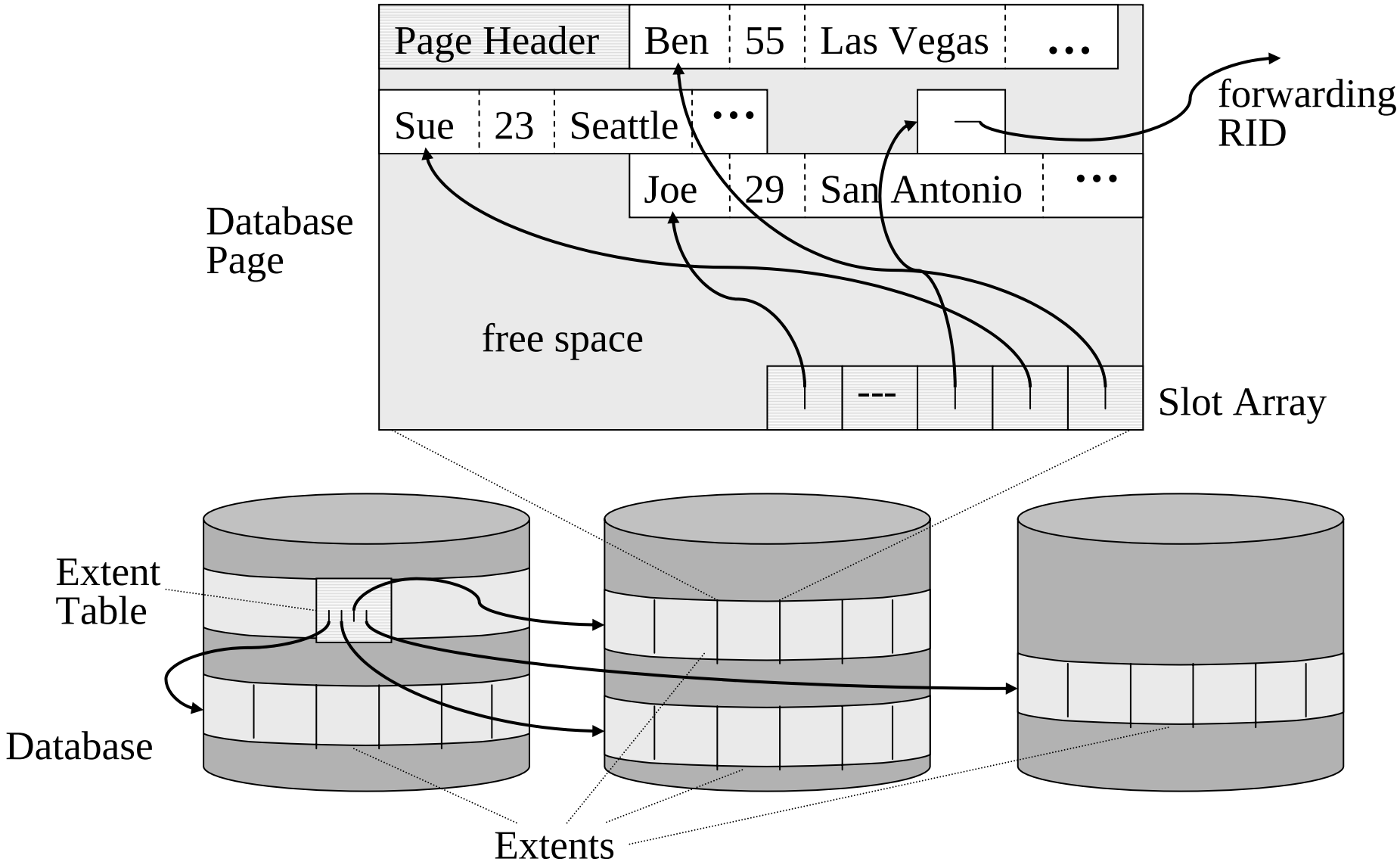
14.2 Wie auf Daten effizient zugegriffen wird (Indexstrukturen)

14.3 Wie Anfragen ausgeführt werden

Interne (idealisierte) Schichtenarchitektur eines DBS



14.1 Wie Daten gespeichert werden



Charakteristika von Magnetplatten (~2001)

(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

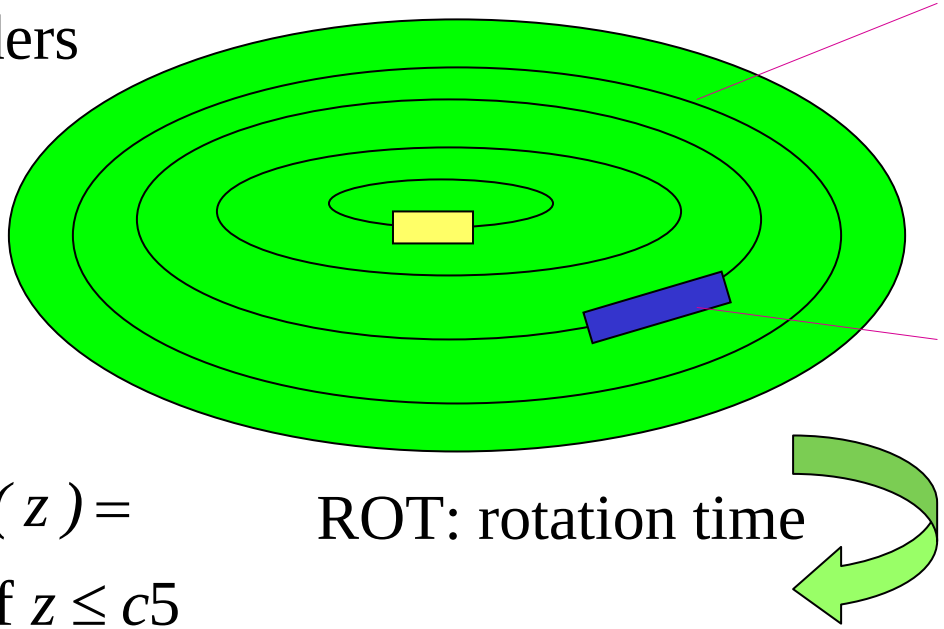
Charakteristika moderner Magnetplatten

(z.B. Seagate Cheetah 15K.4 mit Ultra-SCSI- oder Fibre-Channel-Schnittstelle)

Durchmesser	3.5 Zoll
Speicherkapazität	146,8 GBytes
Preis	ca. 1000 Euro
Größe	ca. 14 x 10 x 2.5 cm
Gewicht	ca. 0,8 kg
Energieverbrauch	17,5 Watt
Zuverlässigkeit (MTTF)	1.400.000 Stunden (160 Jahre)
Anzahl Oberflächen	8
Anzahl Zylinder	50.864
Spurkapazität	471 KBytes im Durchschnitt
Rotationszeit	4 ms (15.000 U/min)
Mittlere Armpositionierungszeit	3.5 ms
Minimale Armpositionierungszeit	0.2 ms
Übertragungsrate	58 bis 96 MBytes / s
Random I/O auf 1 Block a 4 KB	ca. 5,5 ms
Sequential I/O auf 30 Blöcke a 4 KB	ca. 7 ms

Leistungsparameter einer Magnetplatte

Z: #cylinders



C_i: track capacity

$B_i = C_i / ROT$:
disk transfer rate

R: request size

$T_{seek} = tseek(z) =$

$$\begin{cases} c1\sqrt{z} + c2 & \text{if } z \le c5 \\ c3\,z + c4 & \text{otherwise} \end{cases}$$

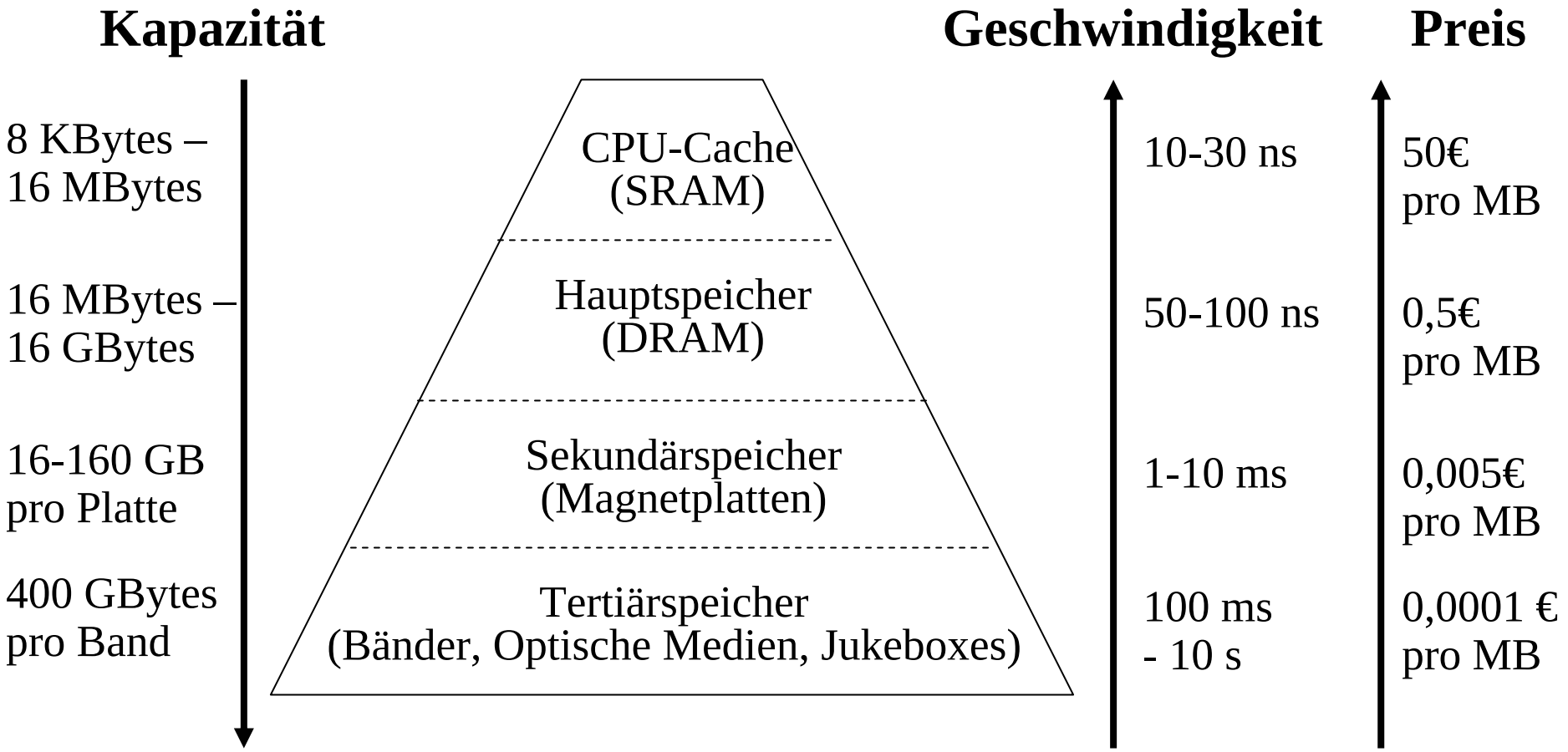
arm seek time

ROT: rotation time

T_{rot}
rotational delay

$T_{trans} = R / B_i$
transfer time

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

Welche Indexstrukturen für welche Attributkombinationen?

$\rightarrow$ 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

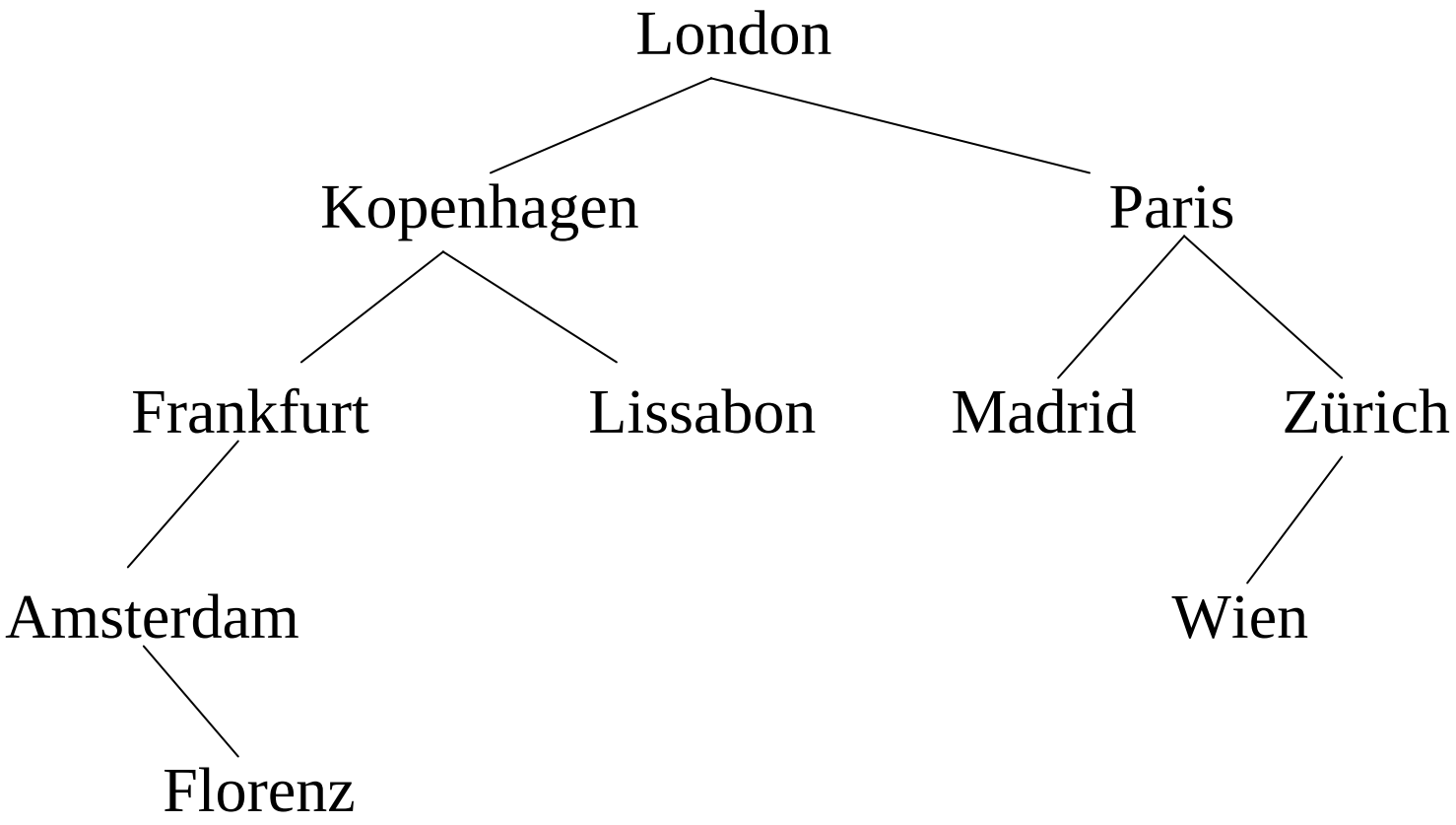
Löschen eines Schlüssel k:

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

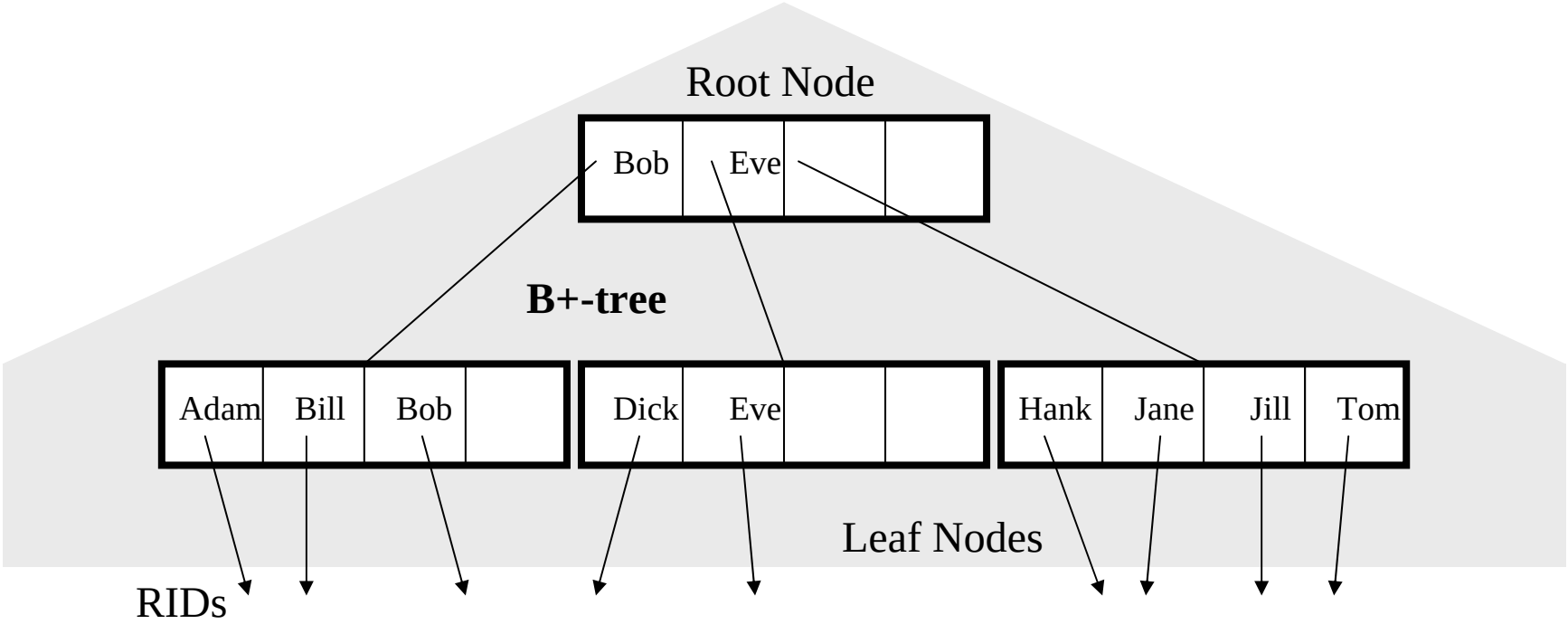
London, Paris, Madrid, Kopenhagen, Lissabon, Zürich, Frankfurt, Wien, Amsterdam, Florenz



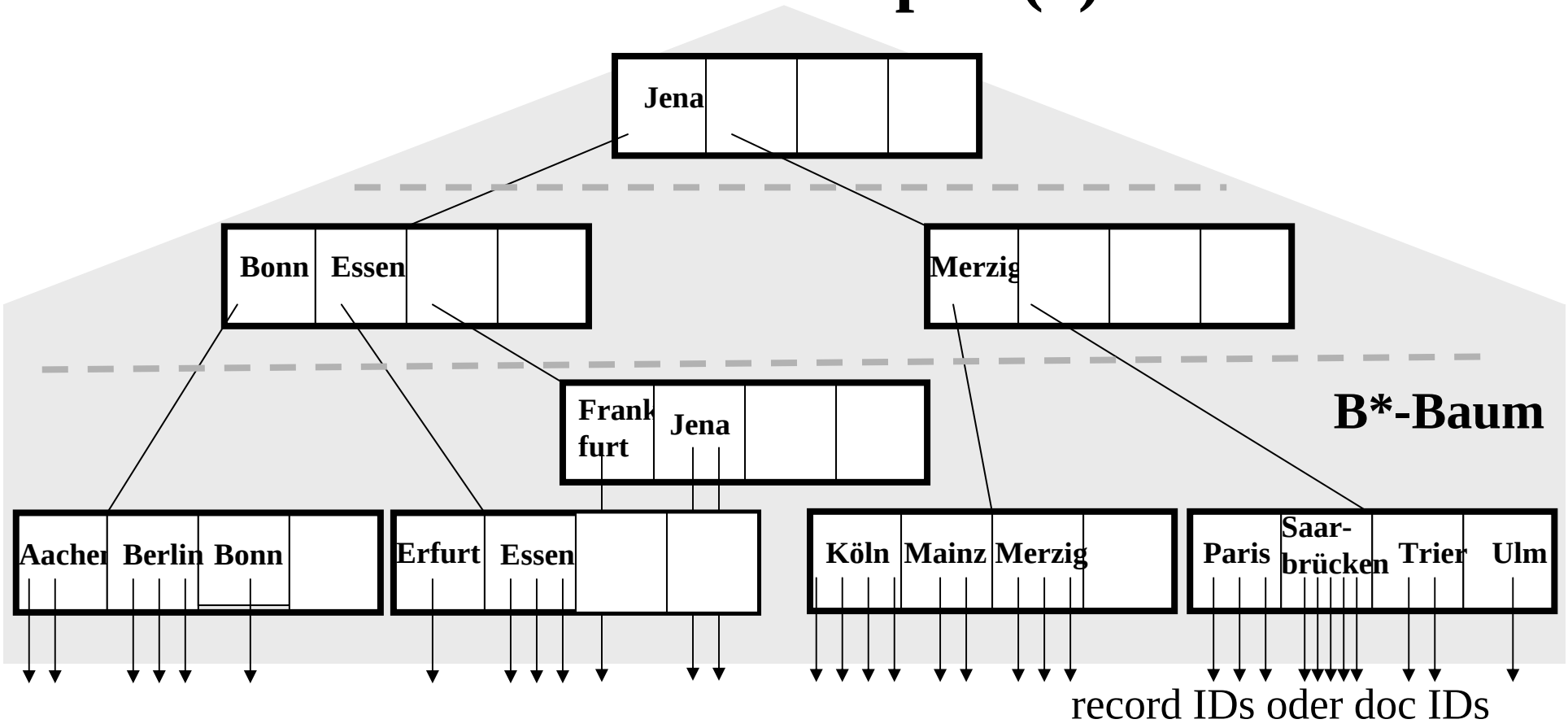
B*-Bäume: Seitenstrukturierte Mehrwegbäume

- 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
- Sucheffizienz $O(\log_k n/C)$ 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 ≥1 und höchstens 2m Schlüssel (Wegweiser).
- Ein Nichtblattknoten mit k Schlüssel x_1, ..., x_k hat genau k+1 Söhne t_1, ..., t_{k+1}, so dass
  - für alle Schlüssel s im Teilbaum t_i ($2 \le i \le k$) gilt $x_{i-1} < s \le x_i$ und
  - für alle Schlüssel s im Teilbaum t_1 gilt $s \le x_1$ und
  - für alle Schlüssel im Teilbaum t_{k+1} gilt xk < s.
- Alle Blätter haben dasselbe Niveau (Distanz von der Wurzel)
- Jedes Blatt enthält mindestens m* ≥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

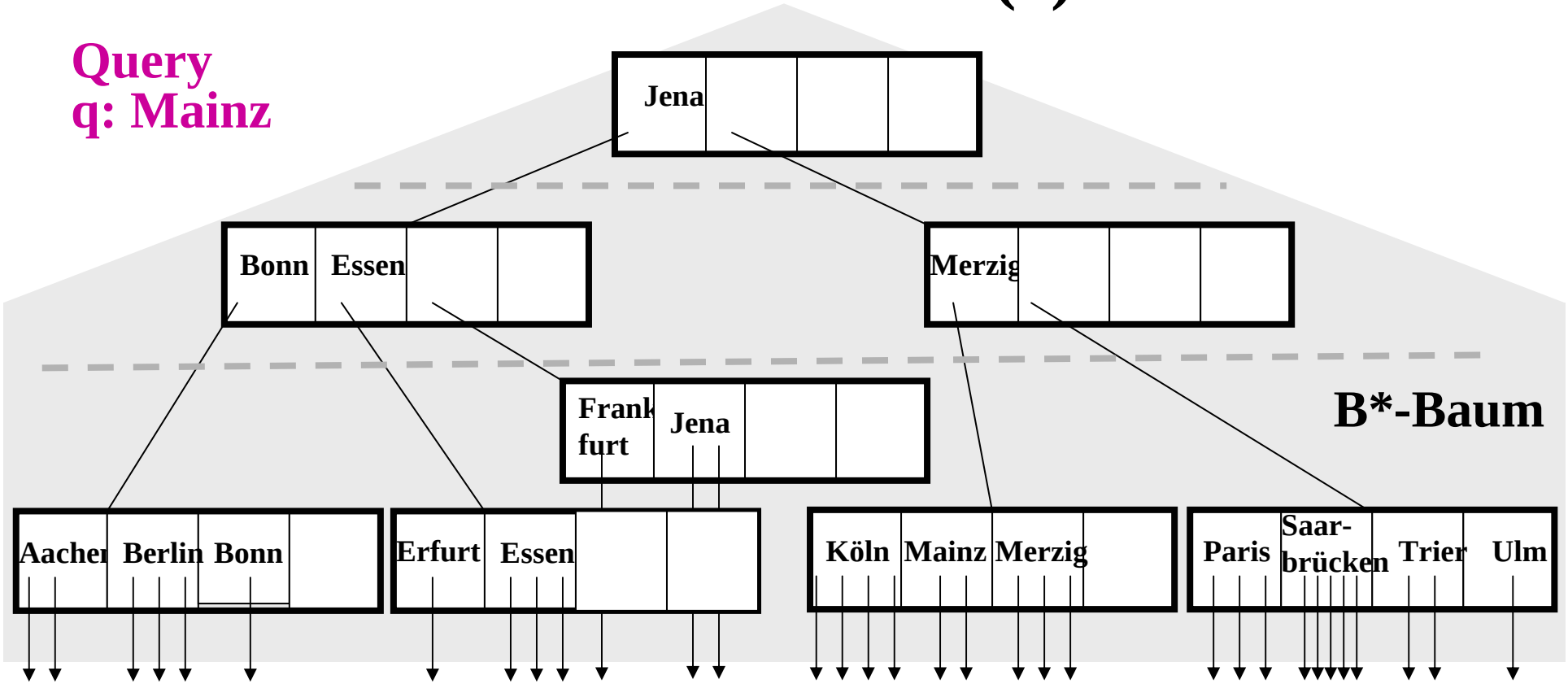
Pseudocode für B*-Baum-Suche

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

```
  t habe k Schlüssel x1, ..., xk und k+1 Söhne t1, ..., t(k+1)
  (letzteres sofern t kein Blatt ist)
  Bestimme den kleinsten Schlüssel xi, so daß s ≤ xi
  if s = xi (für ein i ≤ k) und t ist ein Blatt
  then Schlüssel gefunden
  else
      if t ist kein Blatt then
          if s ≤ xi (für ein i ≤ k)
          then suche s im Teilbaum ti
          else suche s im Teilbaum t(k+1) fi
      else Schlüssel s ist nicht vorhanden fi
   fi
```

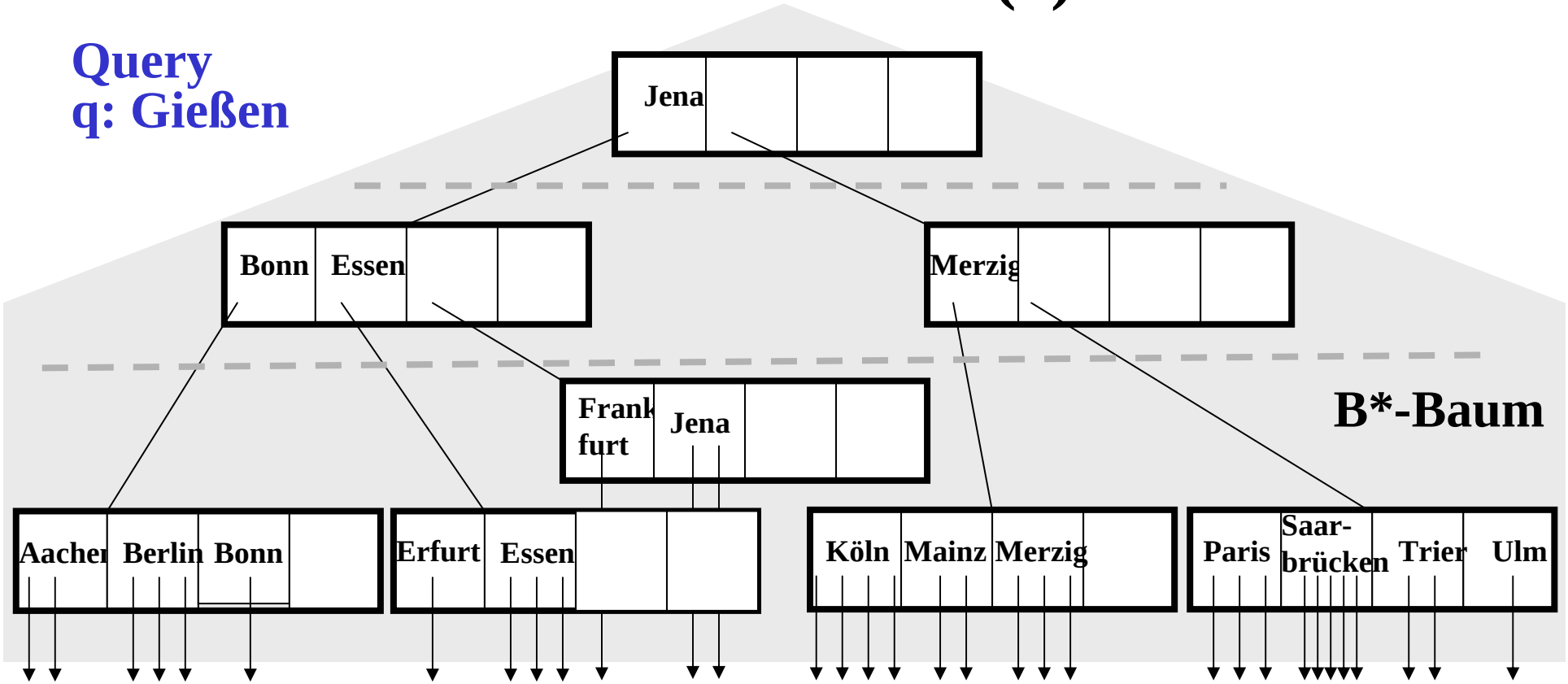
B*-Baum-Suche (1)

Query
q: Mainz



B*-Baum-Suche (2)

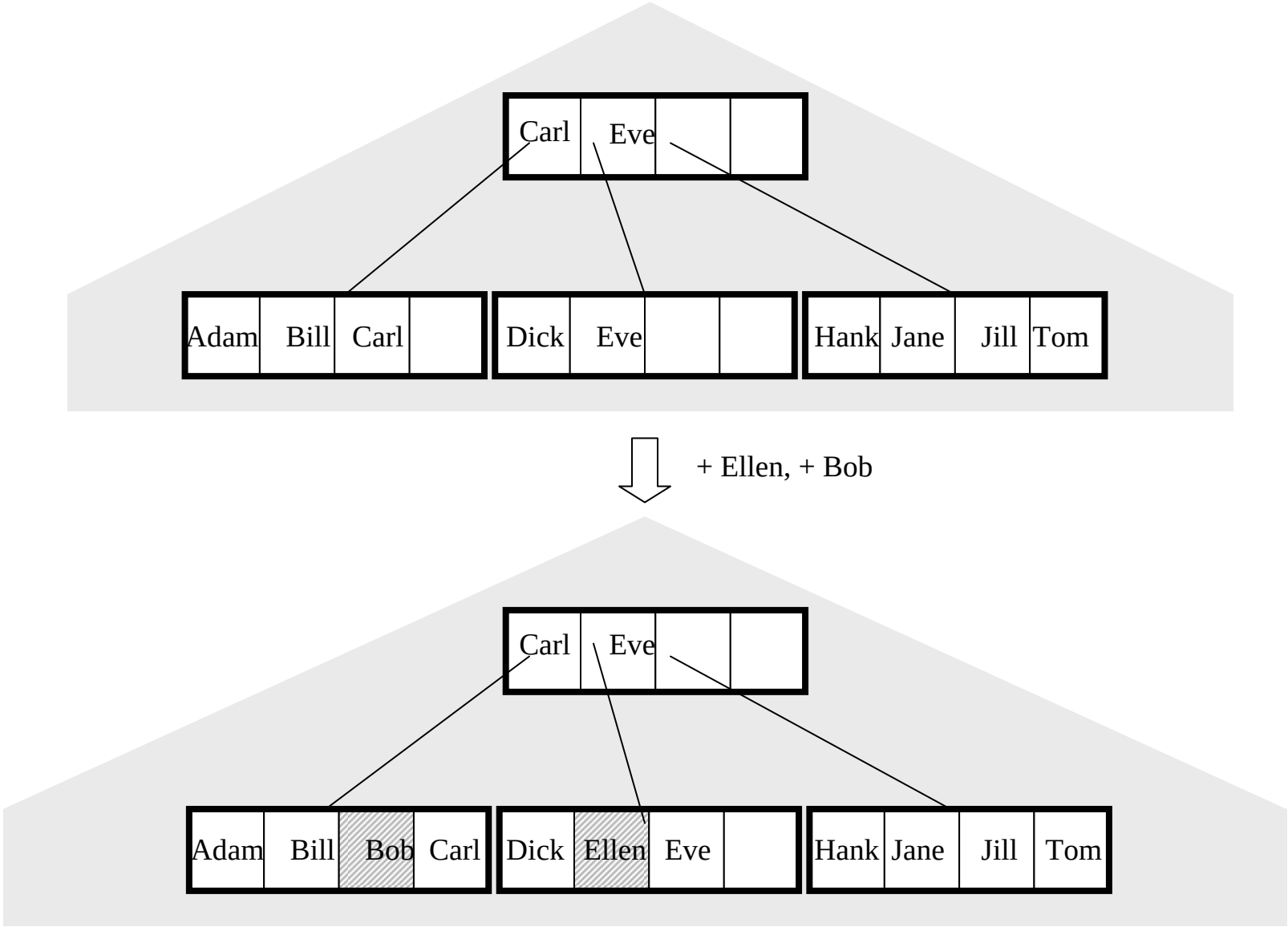
Query
q: Gießen



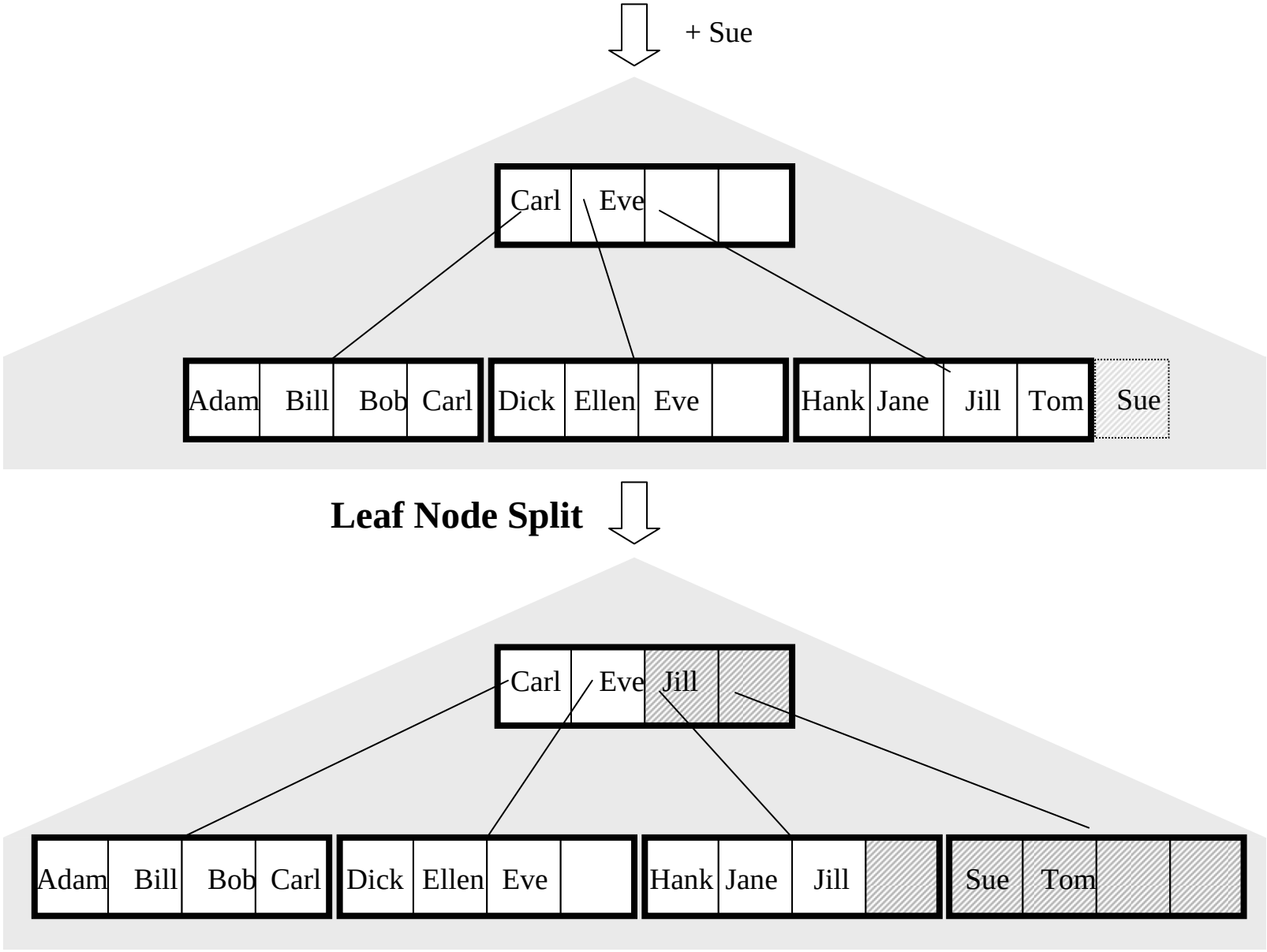
Pseudocode für Einfügen in B*-Baum (Grow&Post)

```
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 ≤ 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
fi
```

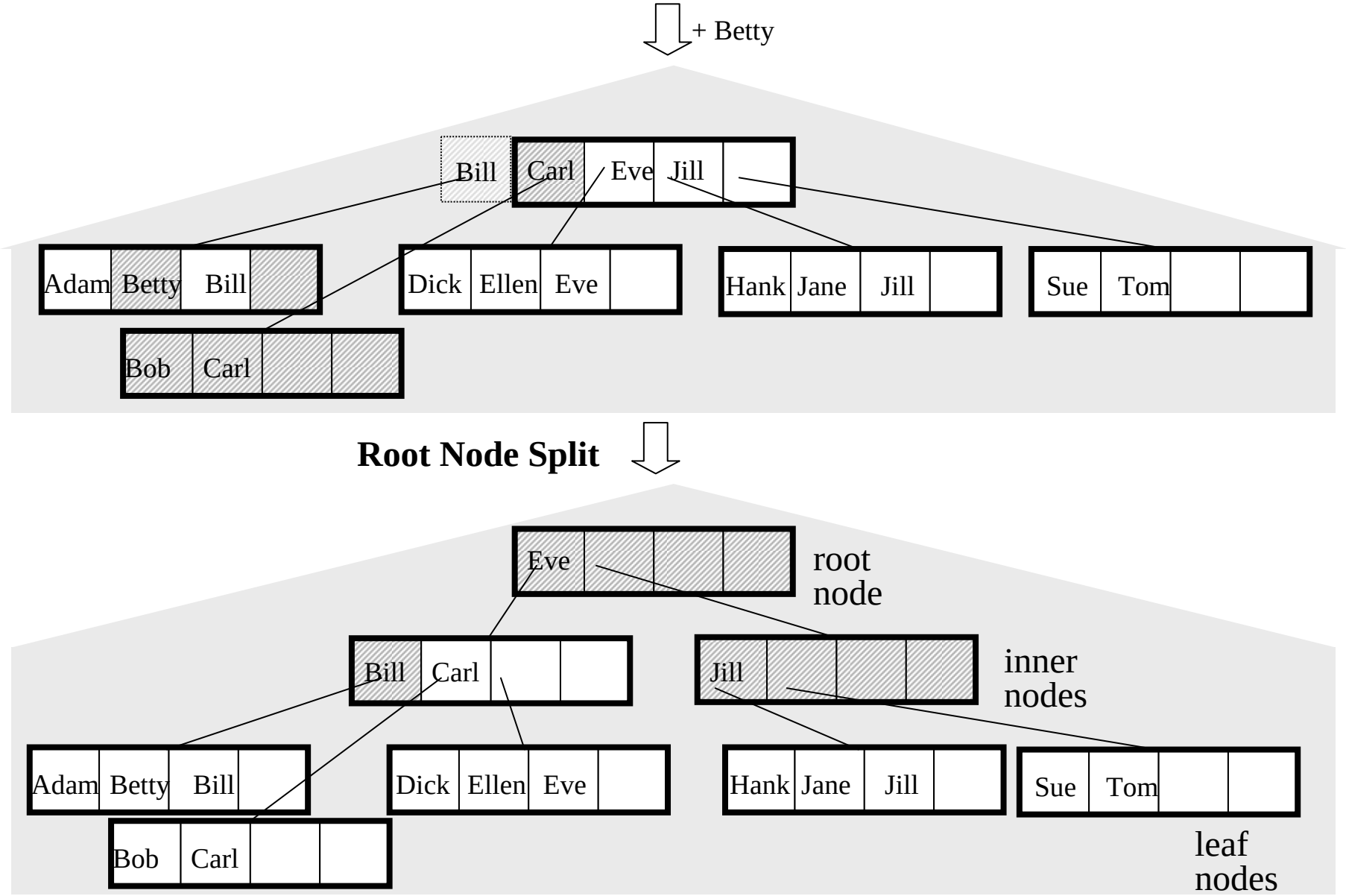
Beispiel: Einfügen in B*-Baum



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



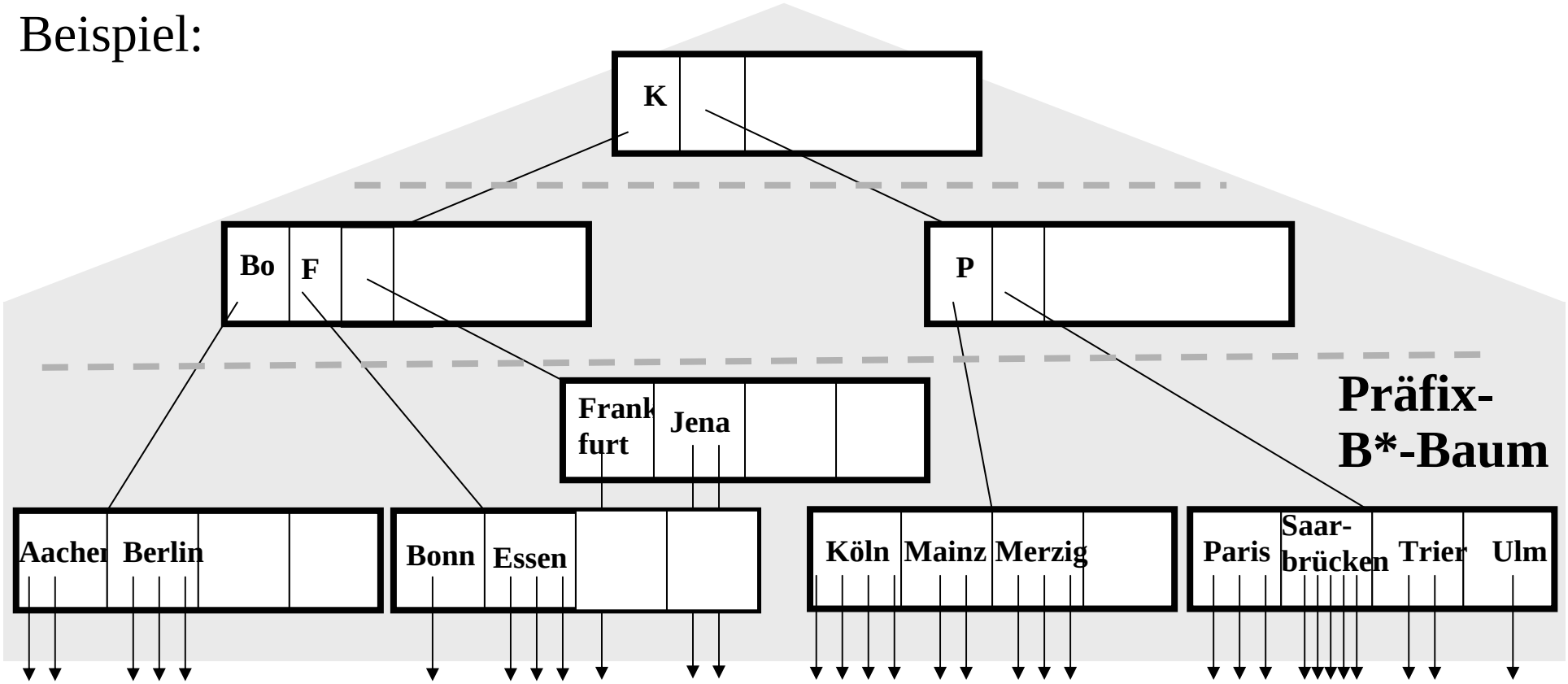
Beispiel: Einfügen in B*-Baum mit Wurzel-Split



Präfix-B*-Bäume für Strings als Schlüssel

Schlüssel in inneren Knoten sind nur **Wegweiser (Router)** zur Partitionierung des Schlüsselraums.
Statt xi = max{s: s ist ein Schlüssel im Teilbaum ti} genügt ein (kürzerer) Wegweiser xi‘ mit si ≤ xi‘< s(i+1) für alle si in ti und alle s(i+1) in t(i+1).
Eine Wahl wäre xi‘ = kürzester String mit der o.a. Eigenschaft.
→ höherer Fanout, potentiell kleinere Baumhöhe

Beispiel:



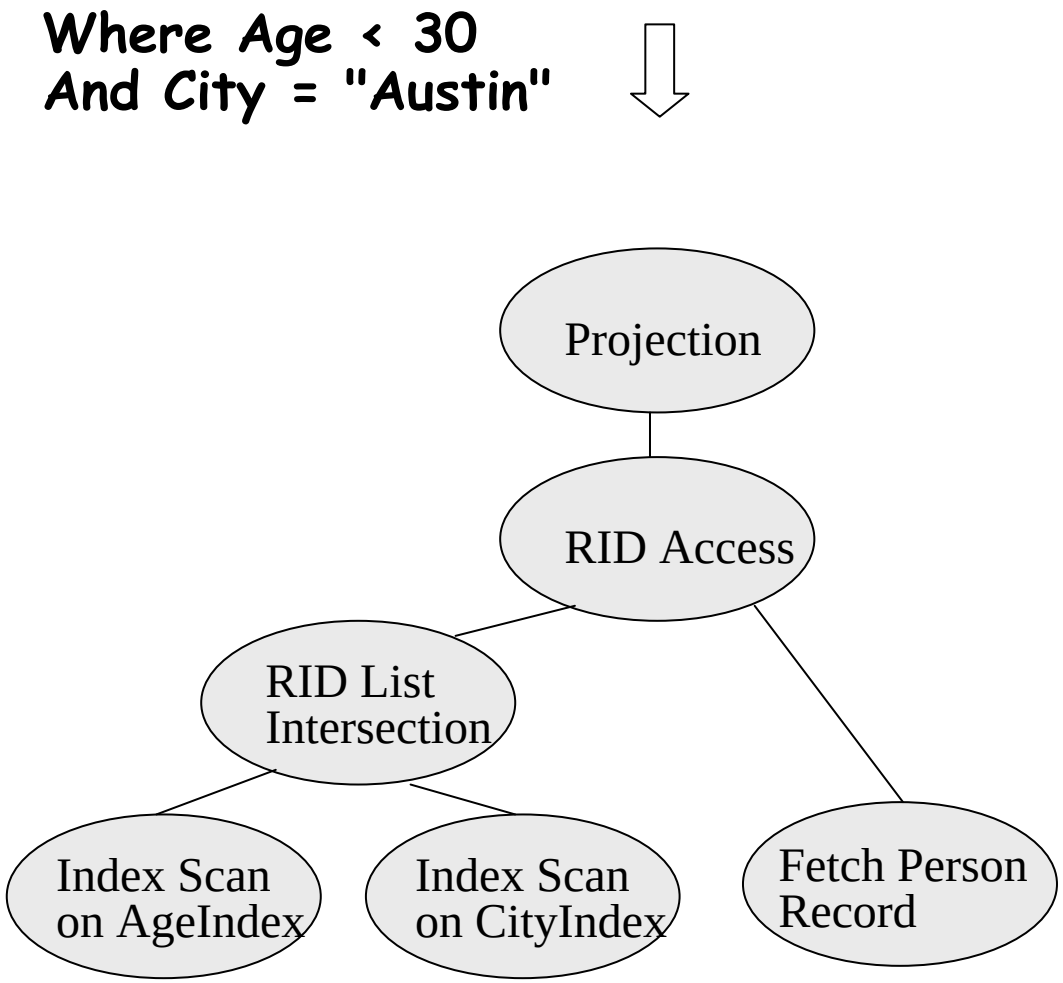
14.3 Wie Anfragen ausgeführt werden

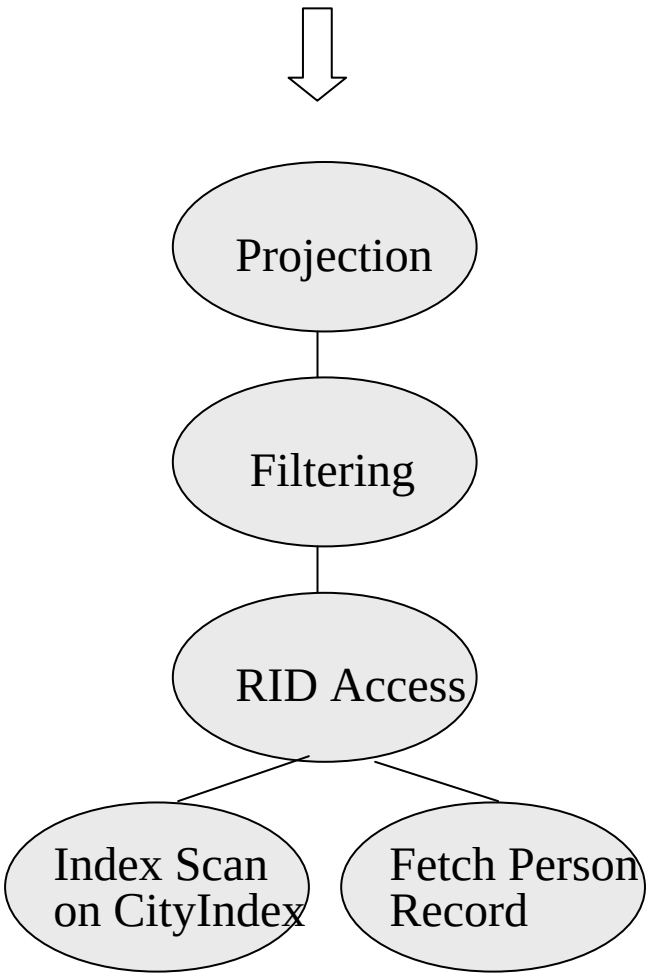
Interne Repräsentation einer Anfrage bzw. eines Ausführungsplans als Operatorbaum mit algebraischen Operatoren der Art:

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

Ausführungspläne (Operatorbäume): Beispiel 1

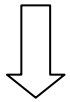
```
Select Name, City, Zipcode, Street
From Person
Where Age < 30
And City = "Austin"
```

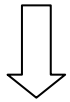




Ausführungspläne (Operatorbäume): Beispiel 1

```
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

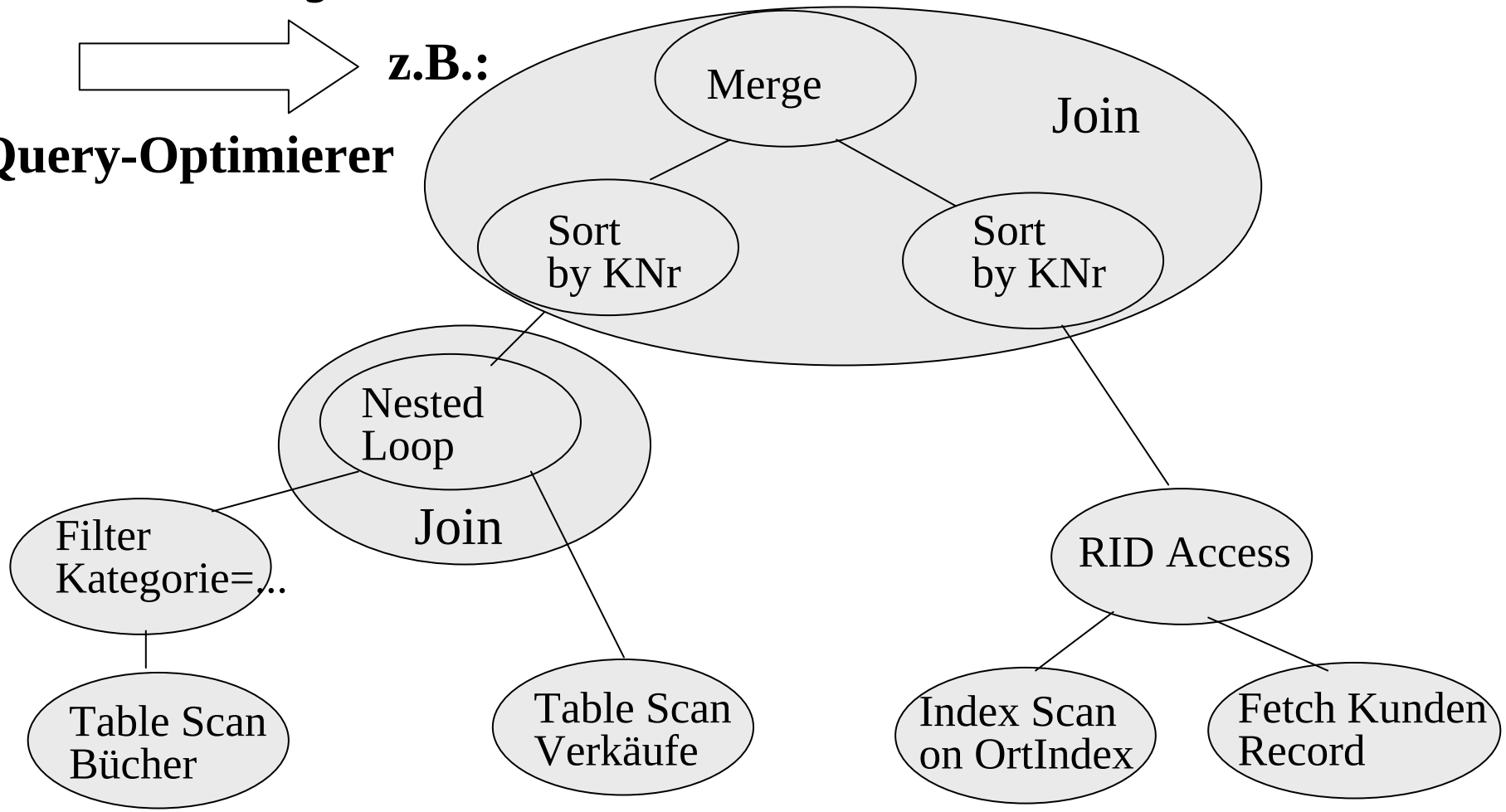
- generiert algebraisch äquivalente Ausführungspläne,
- bewertet deren Ausführungskosten (insbesondere #Plattenzugriffe)
- und wählt den (vermutlich) besten Plan aus.

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

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 Betriebssystemskerns
   (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
- Dauerhauftigkeit/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;
}
```

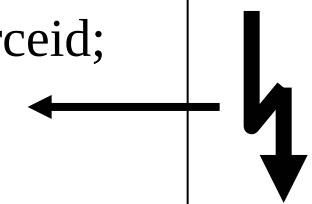
OLTP Example 1: Concurrent Executions

P1	Time	P2
Select Balance Into :b1		
From Account	**1**	
Where Account_Id = :a		
/* b1=0, a.Balance=100, b2=0 */		
		Select Balance Into :b2
	2	**From Account**
		Where Account_Id = :a
/* b1=100, a.Balance=100, b2=100 */		
b1 = b1-50	**3**	
/* b1=50, a.Balance=100, b2=100 */		
	4	**b2 = b2 +100**
/* b1=50, a.Balance=100, b2=200 */		
Update Account		
Set Balance = :b1	**5**	
Where Account_Id = :a		
/* b1=50, a.Balance=50, b2=200 */		
		Update Account
	6	**Set Balance = :b2**
		Where Account_Id = :a
/* b1=50, a.Balance=200, b2=200 */		

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

„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 prädikatorientierte 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 $\rightarrow$ Schedules
- Definition der Äquivalenz $\rightarrow$ Konfliktäquivalenz
- Definition zulässiger Abläufe $\rightarrow$ (Konflikt-) Serialisierbarkeit
- Entwicklung beweisbar korrekter Concurrrency-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 Ri (x) oder Wi (x)
- Partieller Ordnung < auf der Aktionenmenge A,
  wobei für jedes Paar <a, b> $\in$ 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 Ti, Tj $\in$ T gilt: entweder liegen alle Aktionen von Ti bezüglich < vor allen Aktionen von Tj 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 Ti nach Tj, wenn es ein Konfliktpaar <a,b> in s gibt, bei dem a zu Ti und b zu Tj 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)

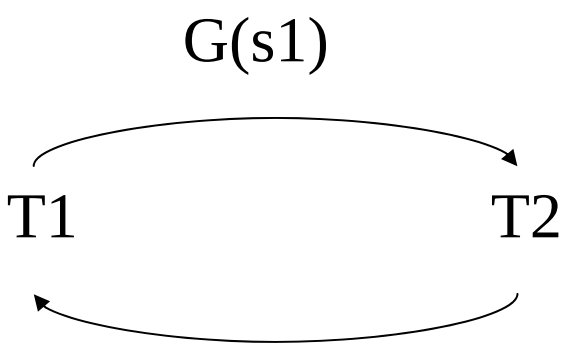
s2: R1 (a) W1 (a) R2 (a) R3 (b) R2 (b) W2 (b) R3 (c) W3 (c) R1 (c)

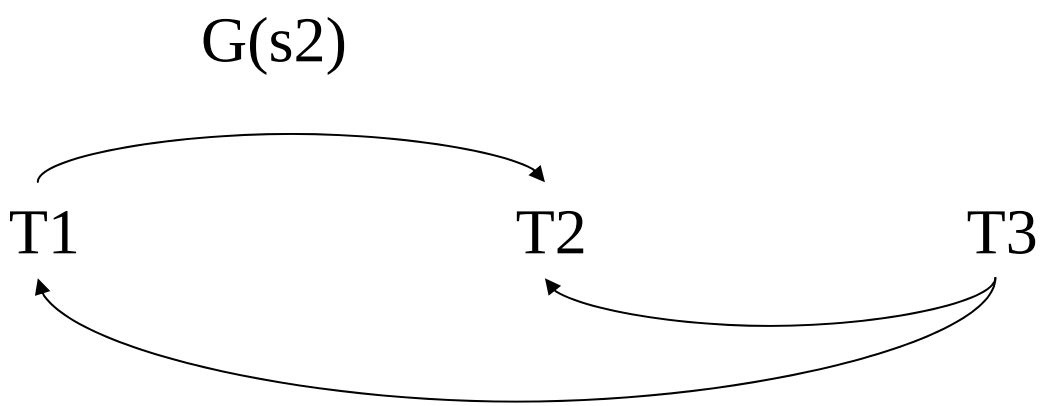
s2‘: R3 (b) R3 (c) W3 (c) R1 (a) W1 (a) R1 (c) R2 (a) R2 (b) W2 (b)

Konfliktpaare s1: <W1(a), R2(a)>, <R2(b), W1(b)>

Konfliktpaare s2: <W1(a), R2(a)>, <R3(b), W2(b)>, <W3(c), R1(c)>

Konfliktpaare s3: <W3(c), R1(c)>, <R3(b), W2(b)>, <W1(a), R2(a)>





Zweiphasen-Sperrprotokoll

DBS-interne Synchronisationsprimitive:
- Lock (Transktion Ti, Objekt x, Modus m)
- Unlock (Transaktion Ti, 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	
		Shared	*Exclusive*
Objekt ist gesperrt im Modus	*Shared*	+	-
	Exclusive	-	-

Deadlocks

Zyklische Wartebeziehungen zwischen Transaktionen
$\rightarrow$ Zykluserkennung auf Waits-For-Graphen
$\rightarrow$ 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 $\rightarrow$ Rollback

2, 3 $\rightarrow$ Crash Recovery (basierend auf Logging)

4 $\rightarrow$ Media Recovery (basierend auf Backup und Logging)

5 $\rightarrow$ Remote Backup/Log, Remote Replication

Goal: Continuous Availability

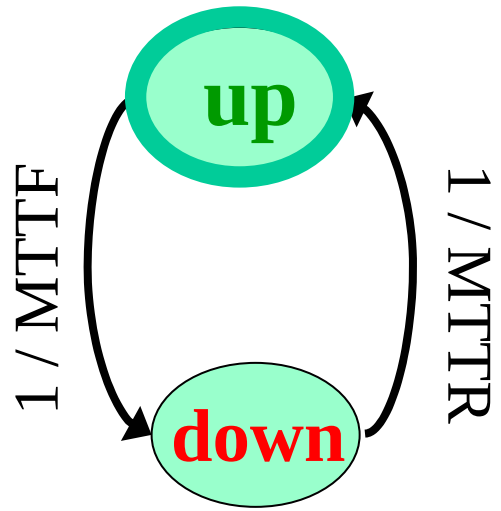
Business apps and e-services demand 24 x 7 availability

99.999 availability would be acceptable (5 min outage/year)

Downtime costs (per hour):

Brokerage operations: $ 6.4 Mio.
Credit card authorization: $ 2.6 Mio.
Ebay: $ 225 000
Amazon: $ 180 000
Airline reservation: $ 89 000
Cell phone activation: $ 41 000

(Source: Internet Week 4/3/2000)



$$\text{Stationary availability} = \frac{MTTF}{MTTF + MTTR}$$

State of the art:

DB servers ≈ 99.99 to 99.999 %

Internet e-service (e.g., Ebay) ≈ 99 %

Heisenbugs and the Recovery Rationale

Failure causes:

- power: 2 000 h MTTF or ∞ with UPS
- chips: 100 000 h MTTF
- system software: 200 – 1 000 h MTTF
- telecomm lines: 4 000 h MTTF
- admin error: ??? or $\rightarrow \infty$ with auto-a
- disks: 800 000 h MTTF
- environment: > 20 000 h MTTF

Transient software failures are the main problem:
Heisenbugs (non-repeatable exceptions caused by stochastic confluence of very rare events)

$\rightarrow$ **Failure model** for crash recovery:
- ***fail-stop*** (no dynamic salvation/resurrection code)
- ***soft failures*** (stable storage survives)

$\rightarrow$ **„Self-healing“ recovery =**
amnesia + data repair (from „trusted“ store) + re-init

Goal of Crash Recovery

Failure-resilience:
- **redo** recovery for committed transactions
- **undo** recovery for uncommitted transactions

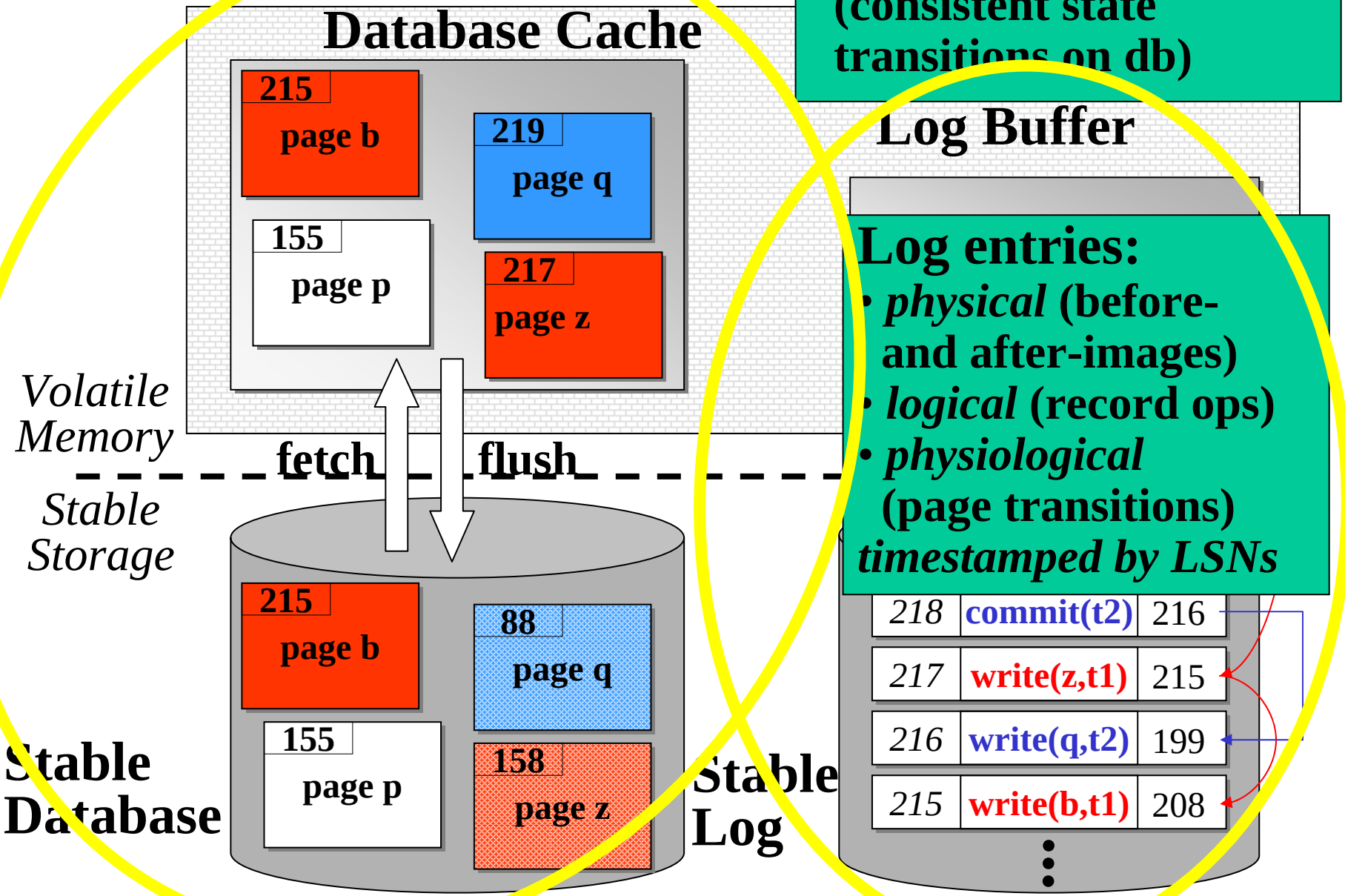
Failure model:
- soft (no damage to secondary storage)
- fail-stop (no unbounded failure propagation)

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

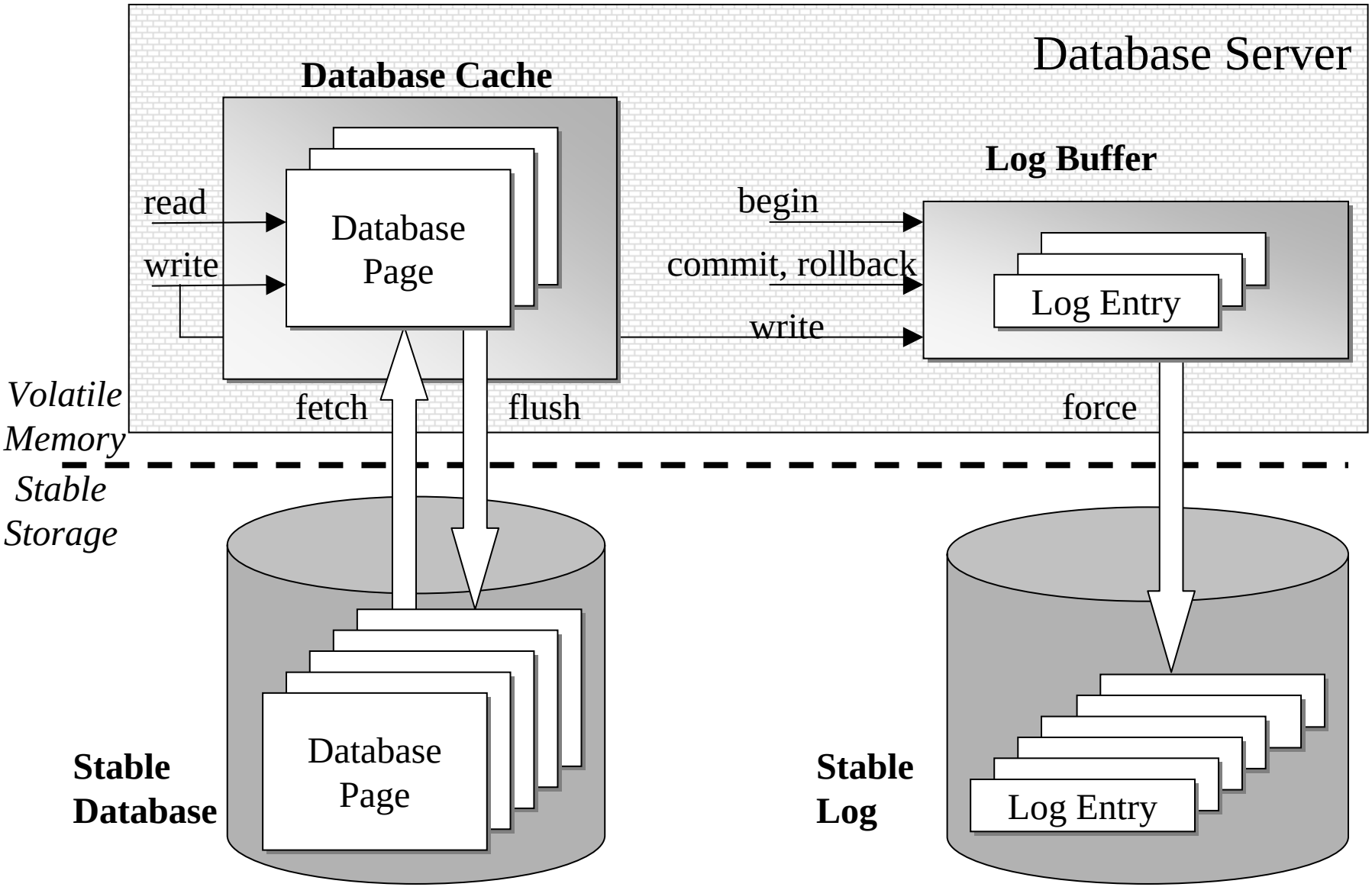
Requirements:
- fast restart for high availability (= MTTF / (MTTF + MTTR)
- low overhead during normal operation
- simplicity, testability, very high confidence in correctness

Why Exactly is Recov

Atomic transactions (consistent state transitions on db)



Overview of System Architecture

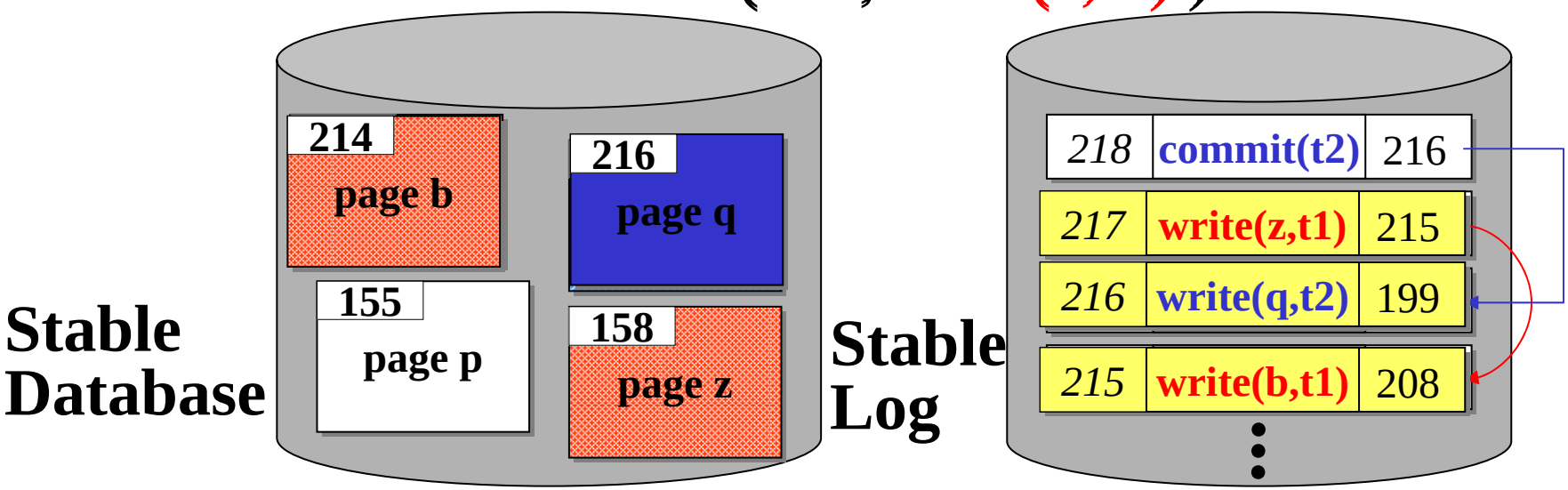


How Does Recovery Work?

- **Analysis pass**: determine winners vs. losers
  (by scanning the stable log)
- **Redo pass**: redo winner writes (b
- **Undo pass**: undo loser writes (by

LSN in page header implements testable state

losers: {t1}, winners: {t2}
redo(216, write(q,t2))
undo (215, write(b,t1)) ?



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

Log Operations in Normal Mode

Goals:

- Avoid random writes to stable log, try to batch writes
- Guarantee entry in stable log to undo change of stable database by potential loser transaction
- Guarentee entry in stable log to redo change by winner transaction

Solution: Flush log buffer when

- Dirty page is flushed from DB cache to stable DB
- Transaction commits

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.PageSeqNo
  and subsequently set p.PageSeqNo := i

- state testing during the undo pass:

  for log entry for page p with log sequence number i,
  undo write only if i ≤ p.PageSeqNo
  and subsequently set p.PageSeqNo := i-1

What Do We Need to Optimize for?

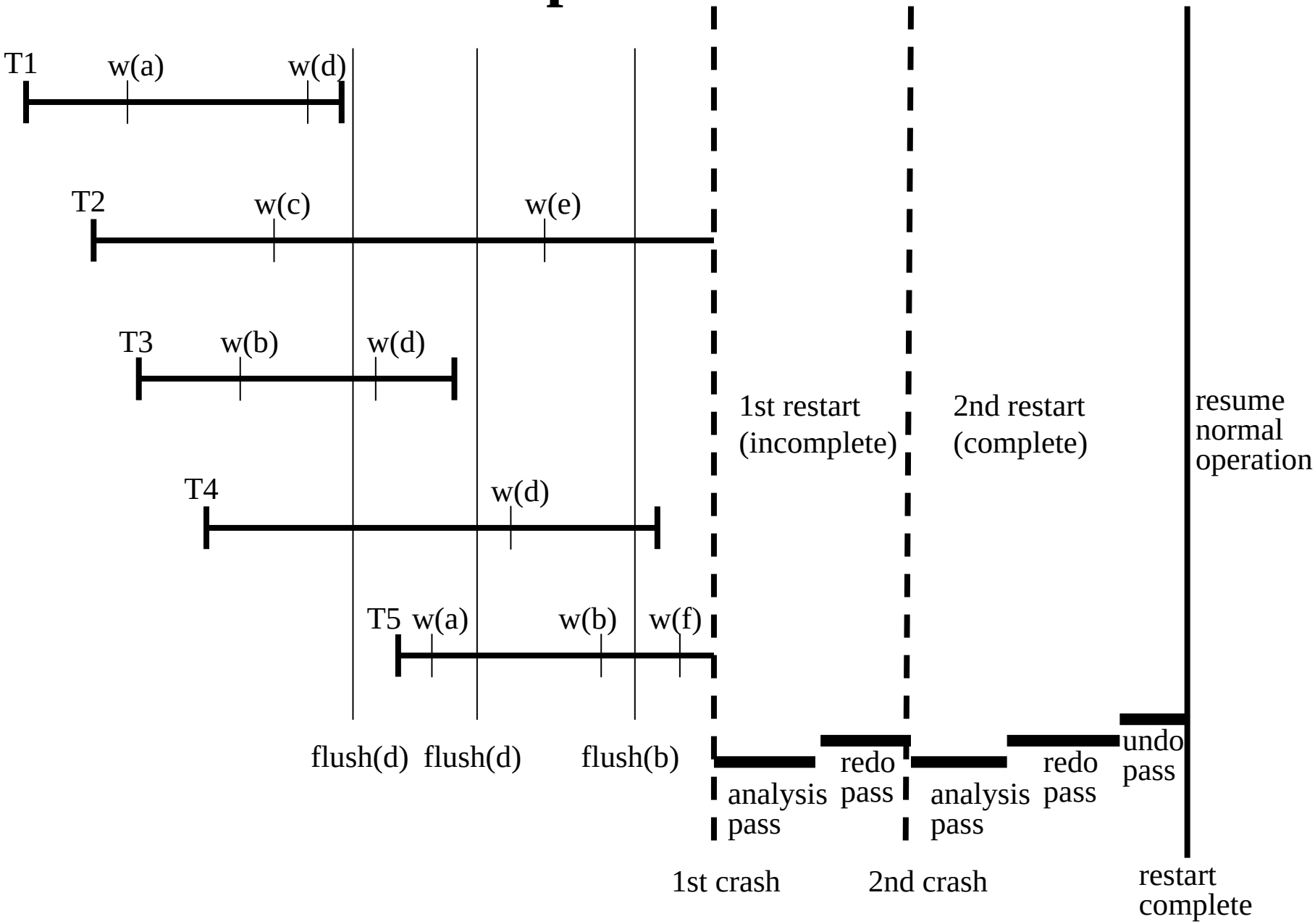
- **Fast restart** (for high **availability**)
  by bounding the log and
  minimizing page fetches

- **Low overhead** during
  normal operation by
  minimizing forced log writes (and page flushes)

- High transaction concurrency
  during normal operation

- Correctness (and simplicity)
  in the presence of many subtleties

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				

Data Structures for Logging & Recovery

```
type Page: record of
   PageNo: id; PageSeqNo: id; Status: (clean, dirty);
   Contents: array [PageSize] of char; end;
persistent var StableDatabase: set[PageNo] of Page;
var DatabaseCache: set[PageNo] of Page;
type LogEntry: record of
   LogSeqNo: id; TransId: id; PageNo: id;
   ActionType:(write, full-write, begin, commit, rollback);
   UndoInfo: array of char; RedoInfo: array of char;
   PreviousSeqNo: id; end;
persistent var StableLog: list[LogSeqNo] of LogEntry;
var LogBuffer: list[LogSeqNo] of LogEntry;
```

modeled in functional manner with test *op* $\in$ *state*:

write s on page p $\in$ StableDatabase $\Leftrightarrow$ StableDatabase[p].PageSeqNo $\geq$ s

write s on page p $\in$ CachedDatabase $\Leftrightarrow$
((p $\in$ DatabaseCache $\wedge$ DatabaseCache[p].PageSeqNo $\geq$ s) $\vee$
(p $\notin$ DatabaseCache $\wedge$ StableDatabase[p].PageSeqNo $\geq$ s))

Correctness Criterion

A crash recovery algorithm is **correct**
if it guarantees that, after a system failure,
the **cached database** will eventually be
equivalent to a
serial order of the **committed transactions**
that coincides with the **serialization order**
of the schedule.

Simple Redo Pass

```
redo pass ( ):
min := LogSeqNo of oldest log entry in StableLog;
max := LogSeqNo of most recent log entry in StableLog;
for i := min to max do
  if StableLog[i].TransId not in losers then
     pageno = StableLog[i].PageNo; fetch (pageno);
     case StableLog[i].ActionType of
        full-write:
           full-write (pageno) with contents
              from StableLog[i].RedoInfo;
        write:
           if DatabaseCache[pageno].PageSeqNo < i then
              read and write (pageno)
                 according to StableLog[i].RedoInfo;
              DatabaseCache[pageno].PageSeqNo := i;
           end /*if*/;
      end /*case*/;
end /*for*/;
```

Correctness of Simple Redo & Undo

Invariants during redo pass (compatible with serialization):

$$\forall s \in StableLog : \Big(\text{all } s' \le s \text{ have been processed} \Rightarrow \big(\forall \text{pages } p \; \forall \text{trans } t \; \forall o \in StableLog : (o.transid = t \wedge o.pageid = p \wedge t \in winners \wedge o \le s) \Rightarrow o \in CachedDb \big) \Big)$$

$$\forall o \in CachedDb : o \in StableLog \vee o \in StableDb$$

Invariant of undo pass:

$$\forall s \in StableLog : \Big(\text{all } s' \in StableLog|_{losers} \text{ with } s' \ge s \text{ have been processed} \Rightarrow \big(\forall \text{pages } p \; \forall \text{trans } t \; \forall o \in StableLog : (o.transid = t \wedge o.pageid = p \wedge t \in losers \wedge o \ge s) \Rightarrow o \notin CachedDb \big) \Big)$$

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)

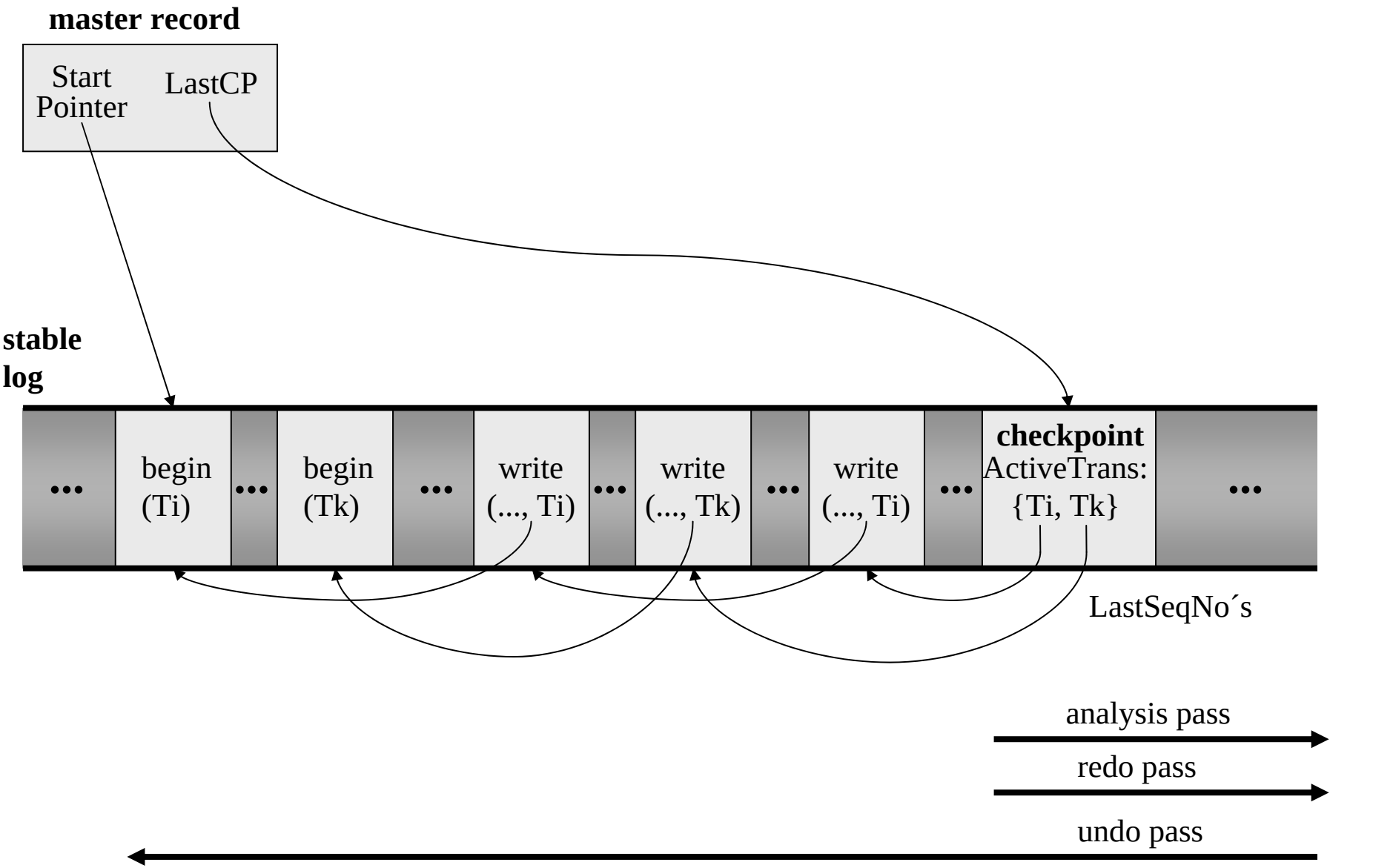
- for redo, can drop all log entries for page p that
  precede the last flush action for p =: RedoLSN (p);
  min{RedoLSN (p) | dirty page p} =: SystemRedoLSN
- for undo, can drop all log entries that
  precede the oldest log entry of a potential loser =: OldestUndoLSN

Remarks:

for full-writes, all but the most recent after-image can be dropped

log truncation after complete undo pass requires global flush

Simplistic Approach: Heavy-Weight Checkpoints



Redo Optimization 1: Light-Weight Checkpoints

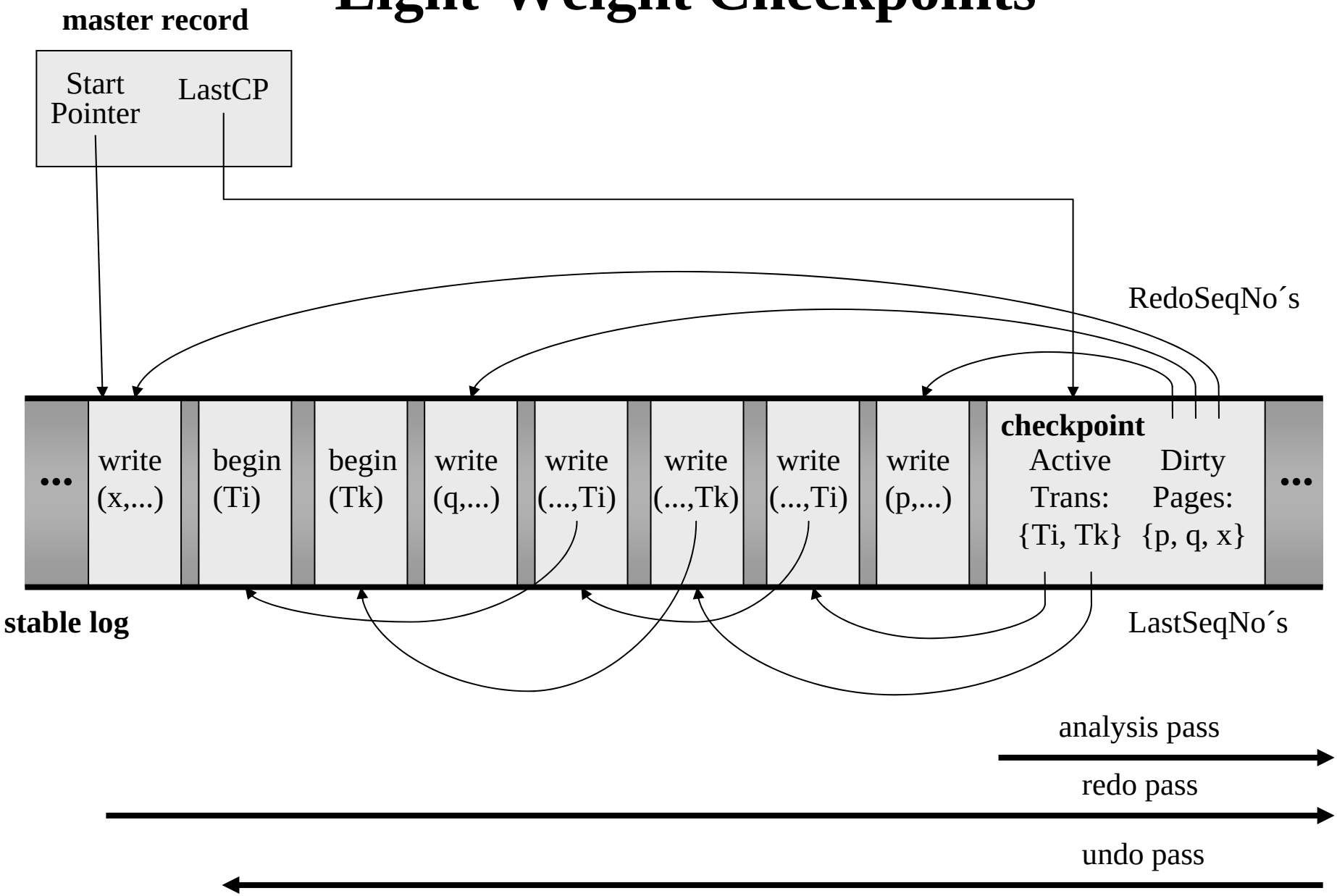
track dirty cache pages and periodically write **DirtyPageTable** (DPT) into **checkpoint log entry:**

```
type DPTEntry: record of
    PageNo: id;
    RedoSeqNo: id; end;
var DirtyPages: set[PageNo] of DPTEntry;
```

+ add potentially dirty pages during analysis pass

+ **flush-behind demon** flushes dirty pages in background

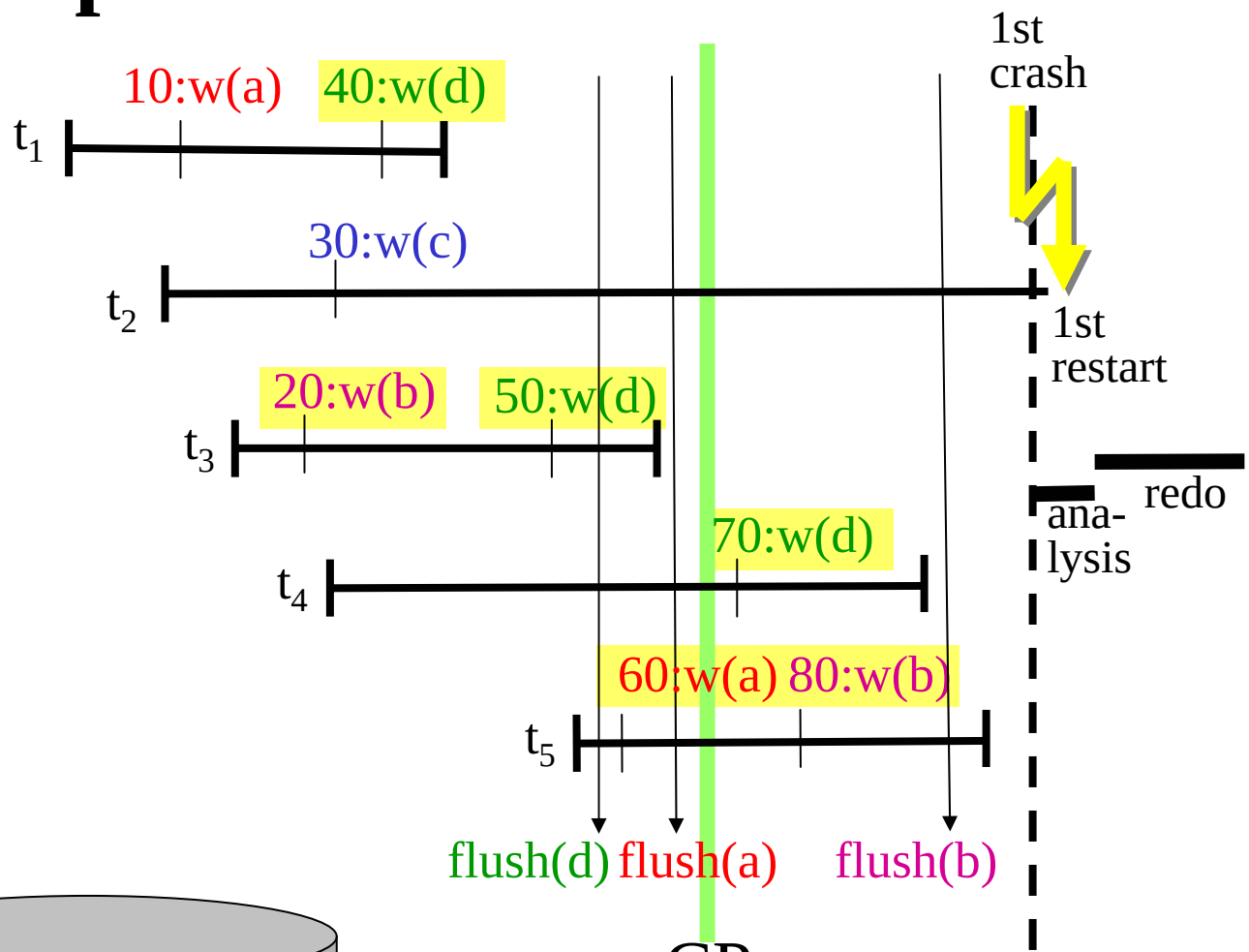
Light-Weight Checkpoints

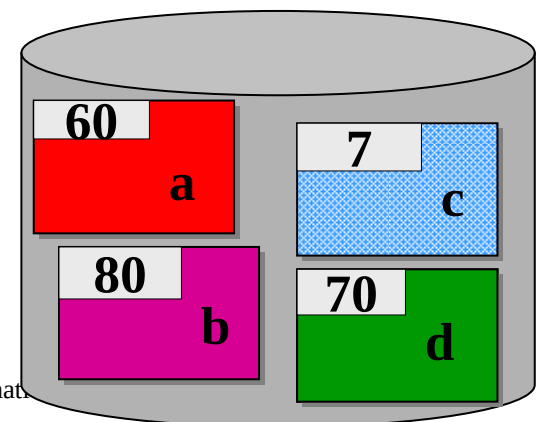


Redo Pass with CP and DPT

```
redo pass ( ):
cp := MasterRecord.LastCP;
SystemRedoLSN := min{cp.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
         and StableLog[i].TransId not in losers then
      pageno := StableLog[i].PageNo;
      if pageno in DirtyPages
            and i >= DirtyPages[pageno].RedoSeqNo then
         fetch (pageno);
         if DatabaseCache[pageno].PageSeqNo < i then
            read and write (pageno)
               according to StableLog[i].RedoInfo;
            DatabaseCache[pageno].PageSeqNo := i;
         else
            DirtyPages[pageno].RedoSeqNo :=
            DatabaseCache[pageno].PageSeqNo + 1;
end; end; end;
```

Example for Redo with CP and DPT





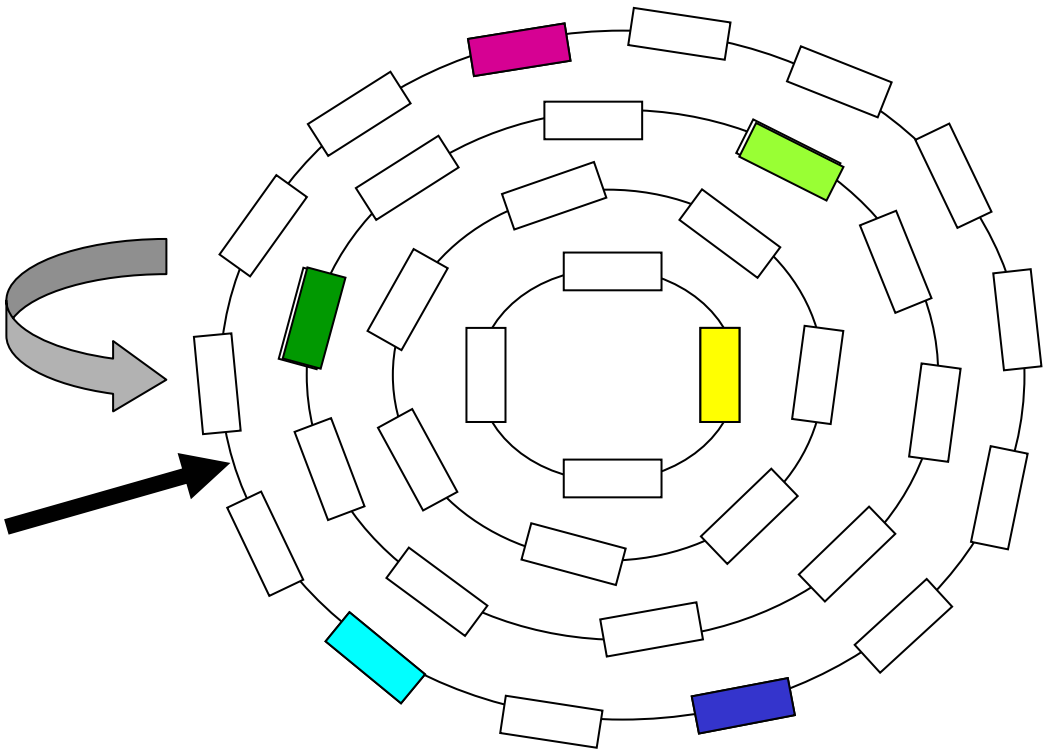
Stable DB

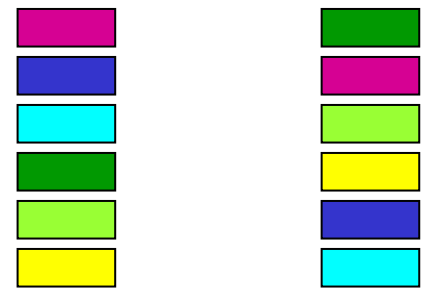
DPT:
b: RedoLSN=20
c: RedoLSN=30
d: RedoLSN=70

Stable Log

Benefits of Redo Optimization

- can save page fetches
- can plan prefetching schedule for dirty pages
  (order and timing of page fetches)
  - for minization of disk-arm seek time
  - and rotational latency





global opt.:
→TSP based on
$t_{seek} + t_{rot}$

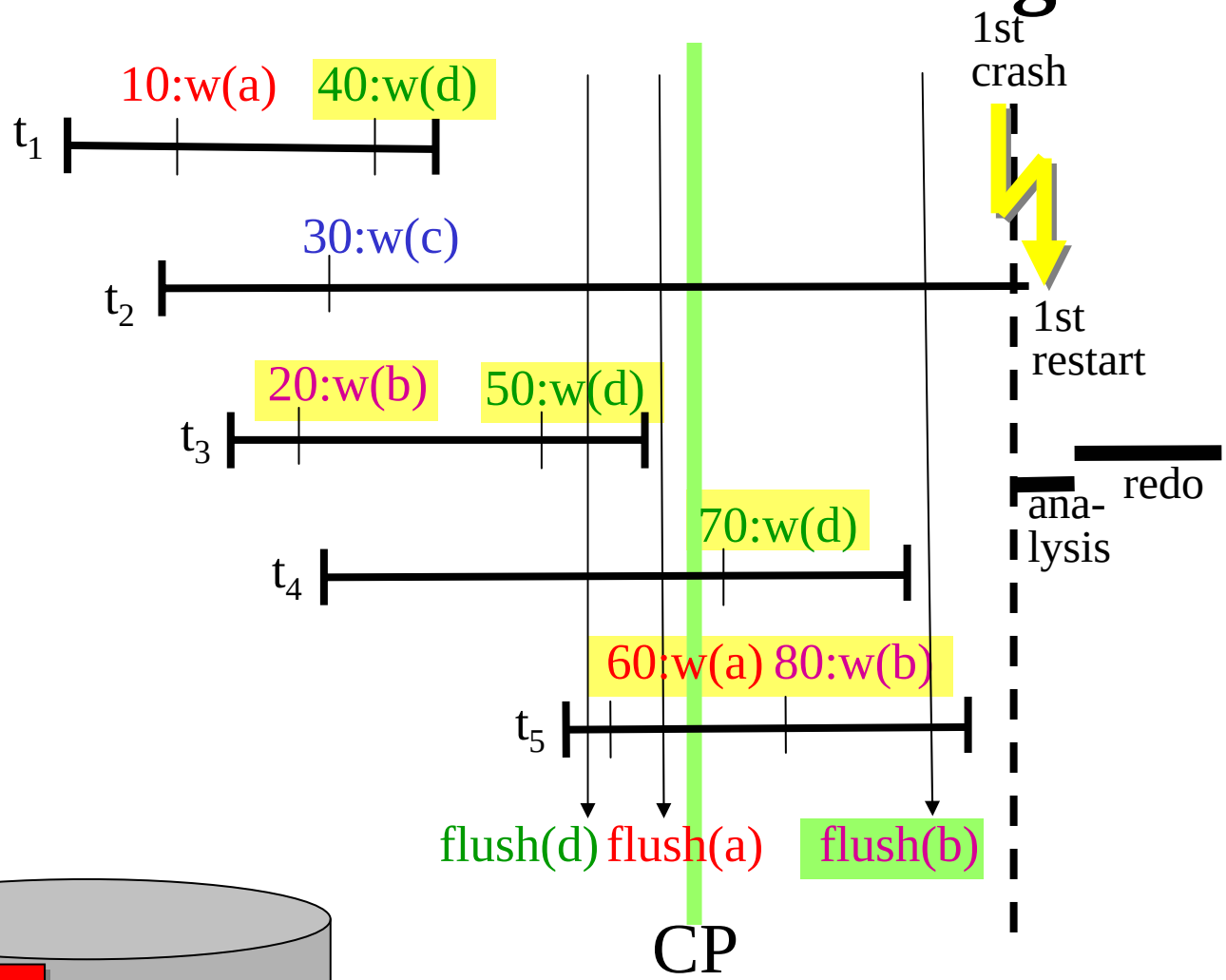
Redo Optimization 2: Flush Log Entries

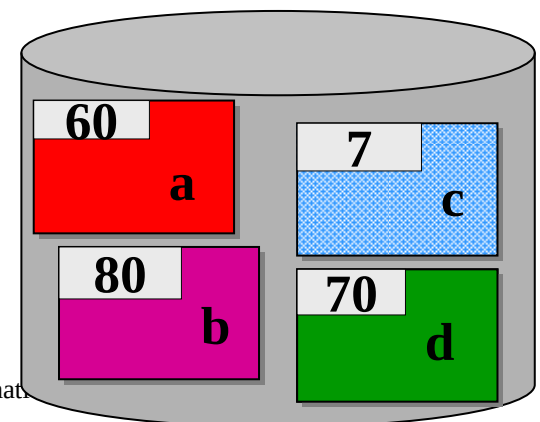
during normal operation:
  log flush actions (without forcing the log buffer)
during analysis pass of restart:
  construct more recent DPT

```
analysis pass ( ) returns losers, DirtyPages:
...
if StableLog[i].ActionType = write or full-write
      and StableLog[i].PageNo not in DirtyPages then
   DirtyPages += StableLog[i].PageNo;
   DirtyPages[StableLog[i].PageNo].RedoSeqNo := i; end;
if StableLog[i].ActionType = flush then
   DirtyPages -= StableLog[i].PageNo; end;
...
```

advantages:
  → can save many page fetches
  → allows better prefetch scheduling

Example for Redo with Flush Log Entries





Stable DB

DPT:

c: RedoLSN=30

d: RedoLSN=70

Stable Log

Redo Optimization 3: Full-write Log Entries

during normal operation:

„occasionally“ log **full after-image** of a page p
(**absorbs** all previous writes to that page)

during analysis pass of restart:

construct **enhanced DPT**
DirtyPages[p].RedoSeqNo := LogSeqNo of full-write

during redo pass of restart:

can ignore all log entries of a page that precede its most recent full-write log entry

advantages:

→ can skip many log entries
→ can plan better schedule for page fetches
→ allows better log truncation

Correctness of Optimized Redo

Invariant during optimized redo pass:

$$\forall pages\ \ p : \forall o \in stable\ log : ($$
$$(o.pageid = p \wedge$$
$$(p \notin DirtyPages \vee o < DirtyPages[\ p\ ].RedoSeqNo))$$
$$\Rightarrow o \in StableDB\ )$$

builds on correct DPT construction during analysis 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				

Korrektur: Ersetze consider-redo(6) durch skip-redo(6)!

Why is Page-based Redo Logging Good Anyway?

- testable state with very low overhead
- much faster than logical redo

  → logical redo would require replaying high-level operations with many page fetches (random IO)

- can be applied to selective pages independently

  → can exploit perfectly scalable parallelism for redo of large multi-disk db

  → can efficiently reconstruct corrupted pages when disk tracks have errors (without full media recovery for entire db)

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;
typeDirtyPageInfo: 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[nexttrans].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:

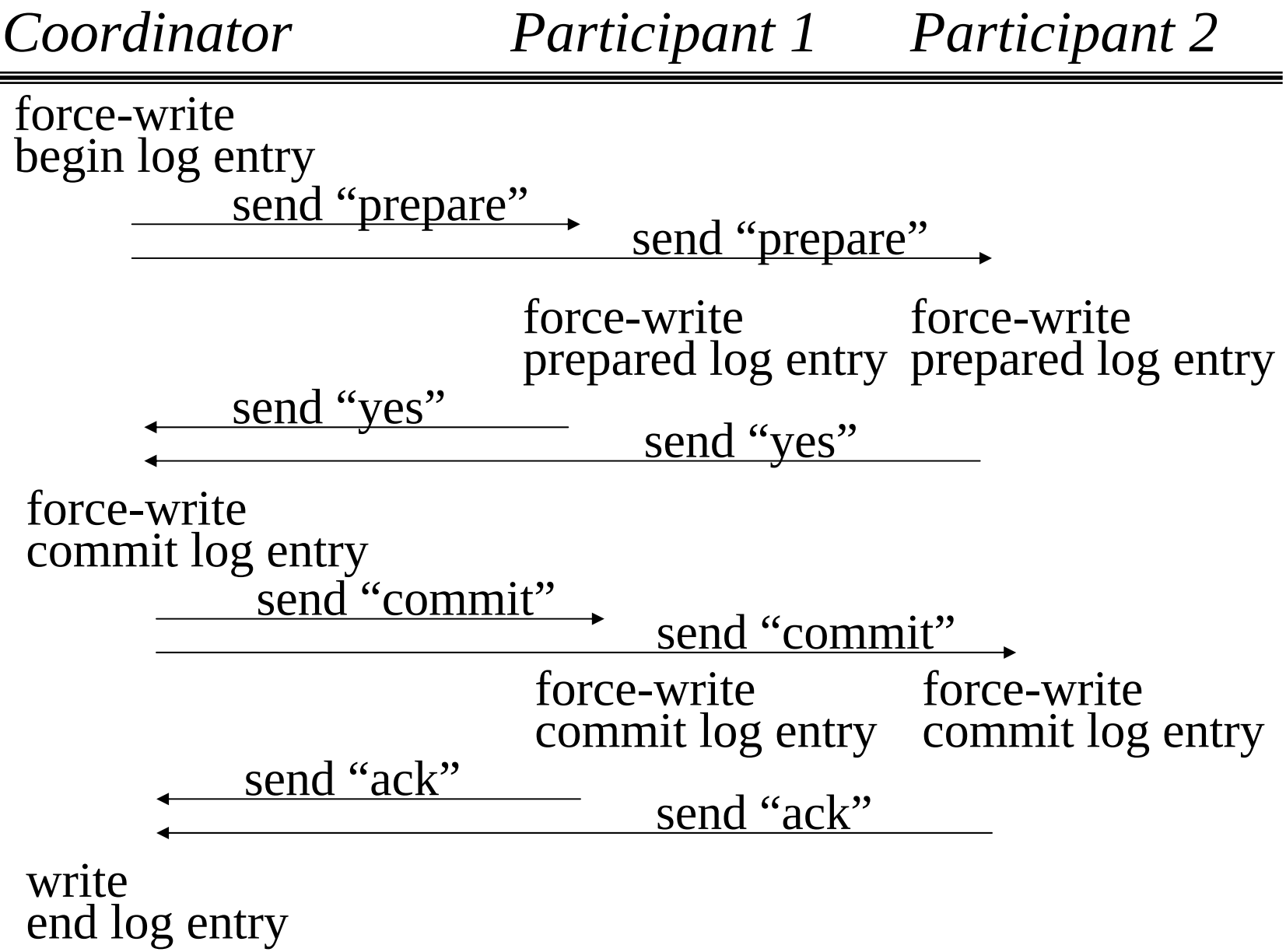
- Distributed handshake protocol
  known as **two-phase commit (2PC)**
- with a **coordinator** taking responsibility for unanimous outcome
- Recovery considerations for in-doubt transactions

2PC During Normal Operation

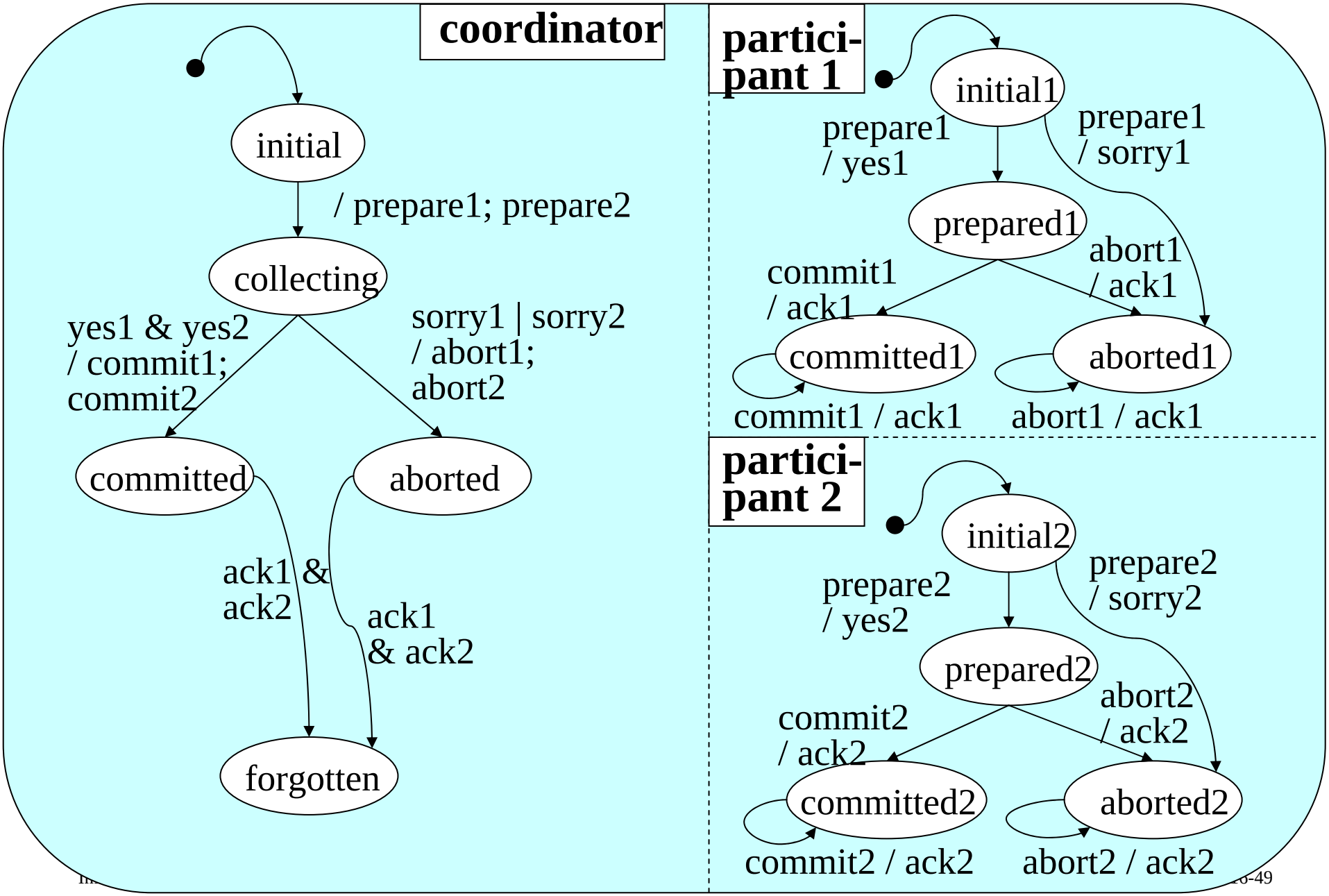
- **First phase (voting):**
  coordinator sends *prepare* messages to participants
  and waits for *yes* or *no* votes
- **Second phase (decision)**
  coordinator sends *commit* or *rollback* messages to participants
  and waits for *acks*
- **Participants** write *prepared* log entries in voting phase
  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
  $\rightarrow$ Byzantine agreement
- no assumptions about network failure handling
  $\rightarrow$ can use datagrams or sessions for communication

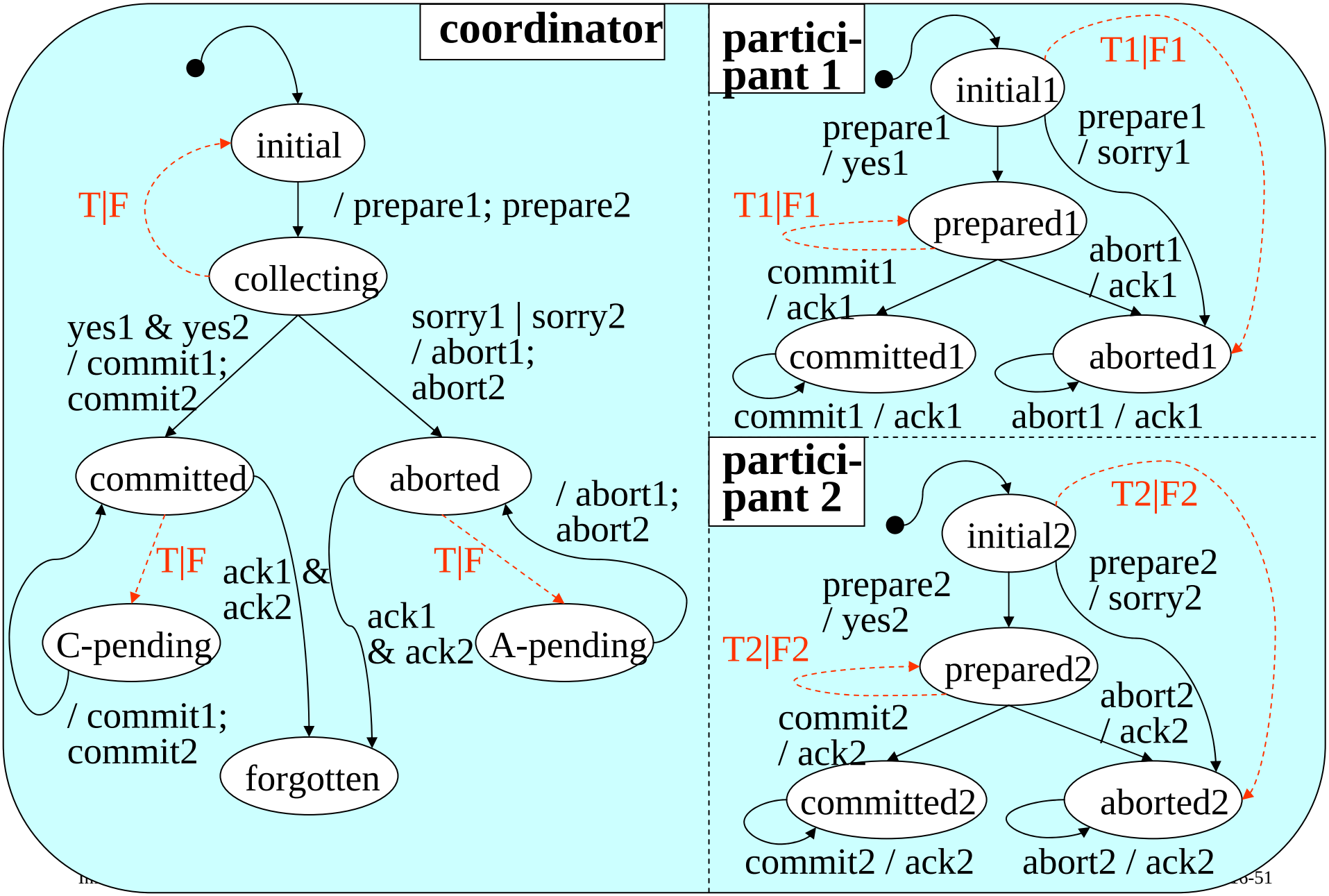
Restart protocol after failure (F transitions):

- 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 (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 (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).

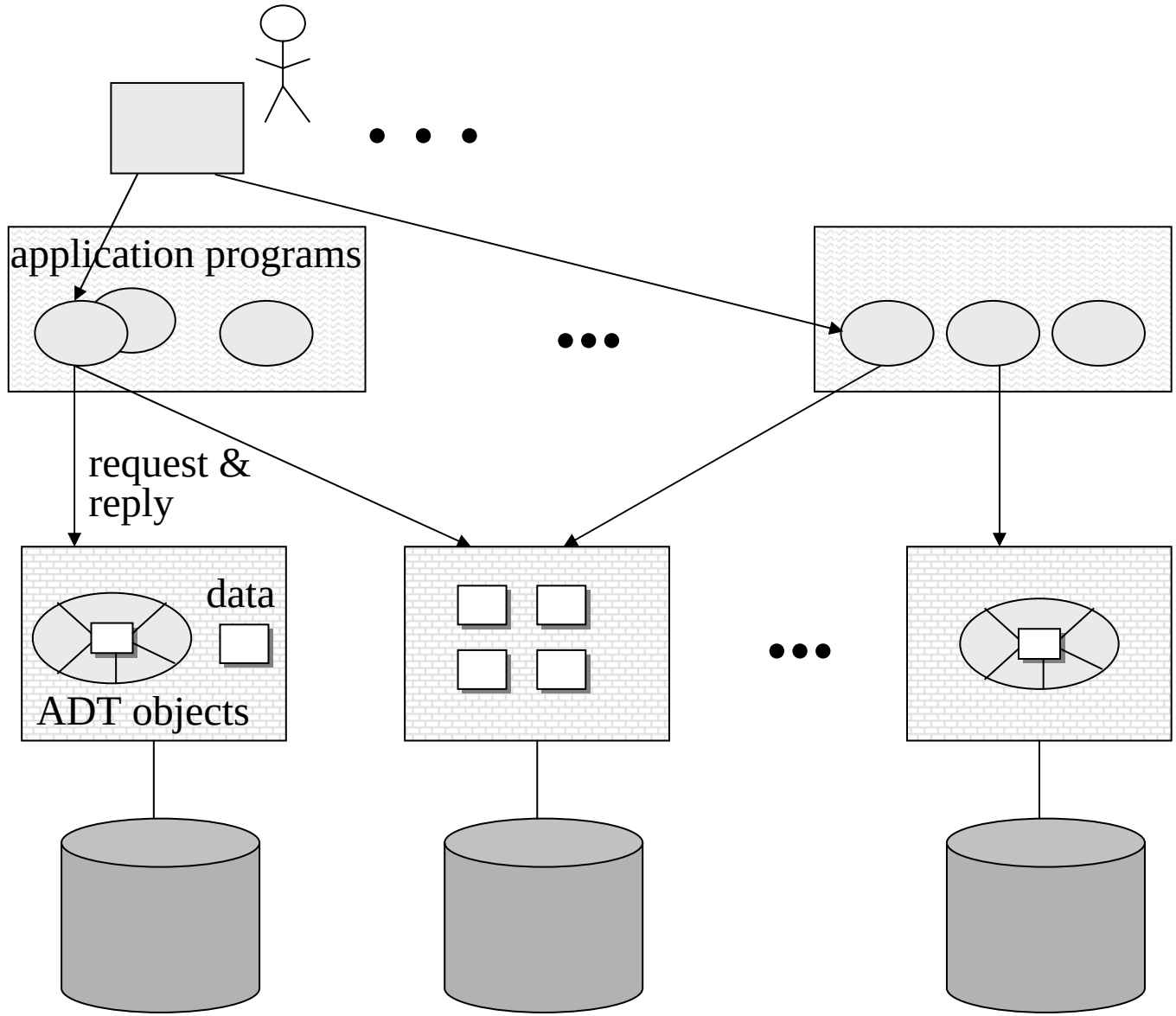
Kapitel 17: Verteilte Informationssysteme

17.1 Verteilte Systemarchitekturen
17.2 Anfrageausführung in verteilten IR-Systemen
17.3 Skalierbare verteilte Indexierung und Suche
17.4 Anfrageausführung in verteilten DB-Systemen

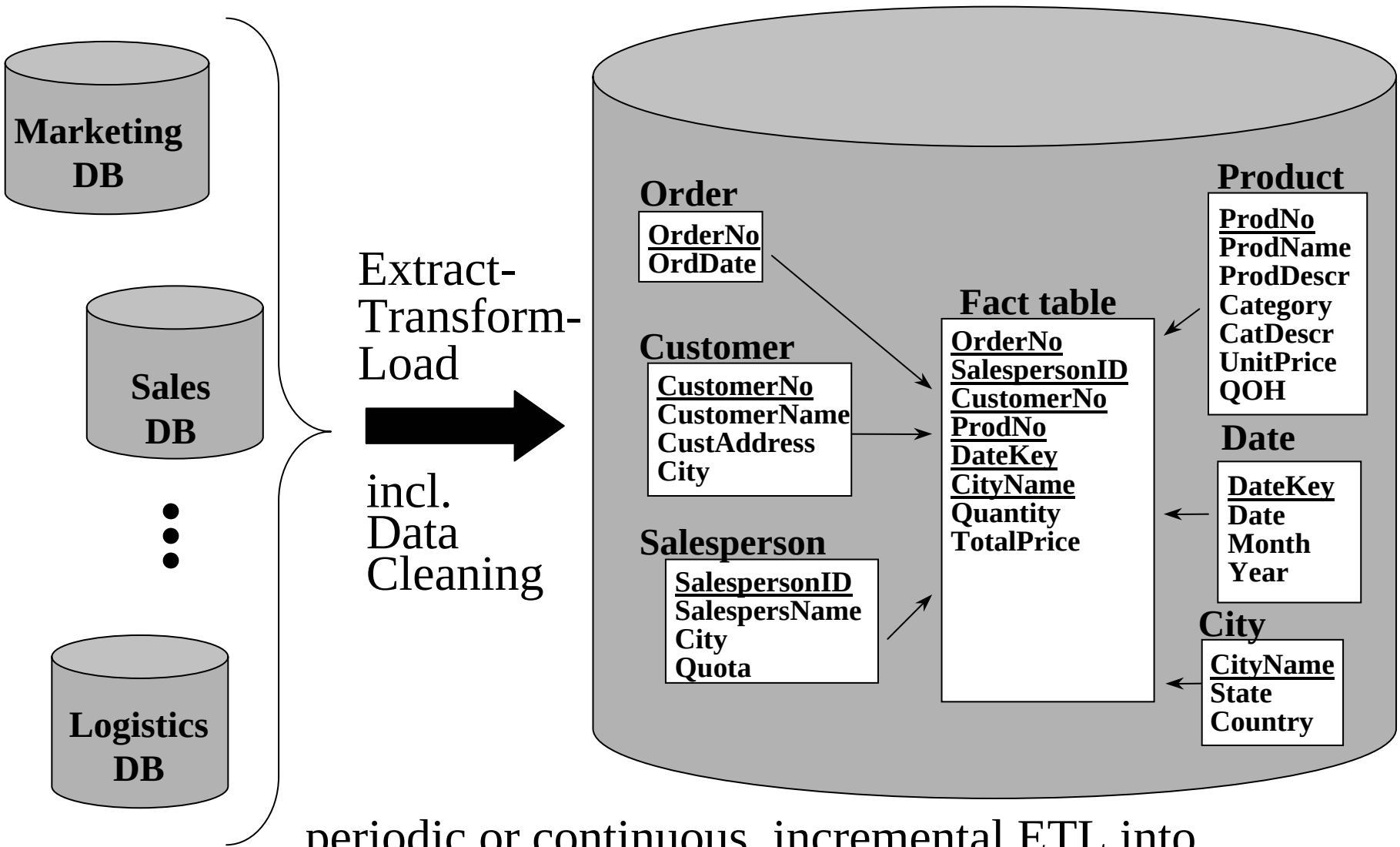
17.1 Verteilte Systemarchitekturen

- **Software-Component-Oriented:**
  - **Client-Server Architecture**
  - **Multi-Tier Architecture**
  - **Federated Systems**
- **Data-Source-Oriented:**
  - **Data Warehouses (and Digital Libraries)**
  - **Wrapper-Mediator Architecture for Information Integration (Example: Internet Portals)**
- **Uncoordinated Decentralization:**
  - **Service-Oriented Architecture for Web Services or Grid**
  - **Peer-to-Peer Systems (P2P)**

Client-Server and (Federated) Multi-Tier Systems

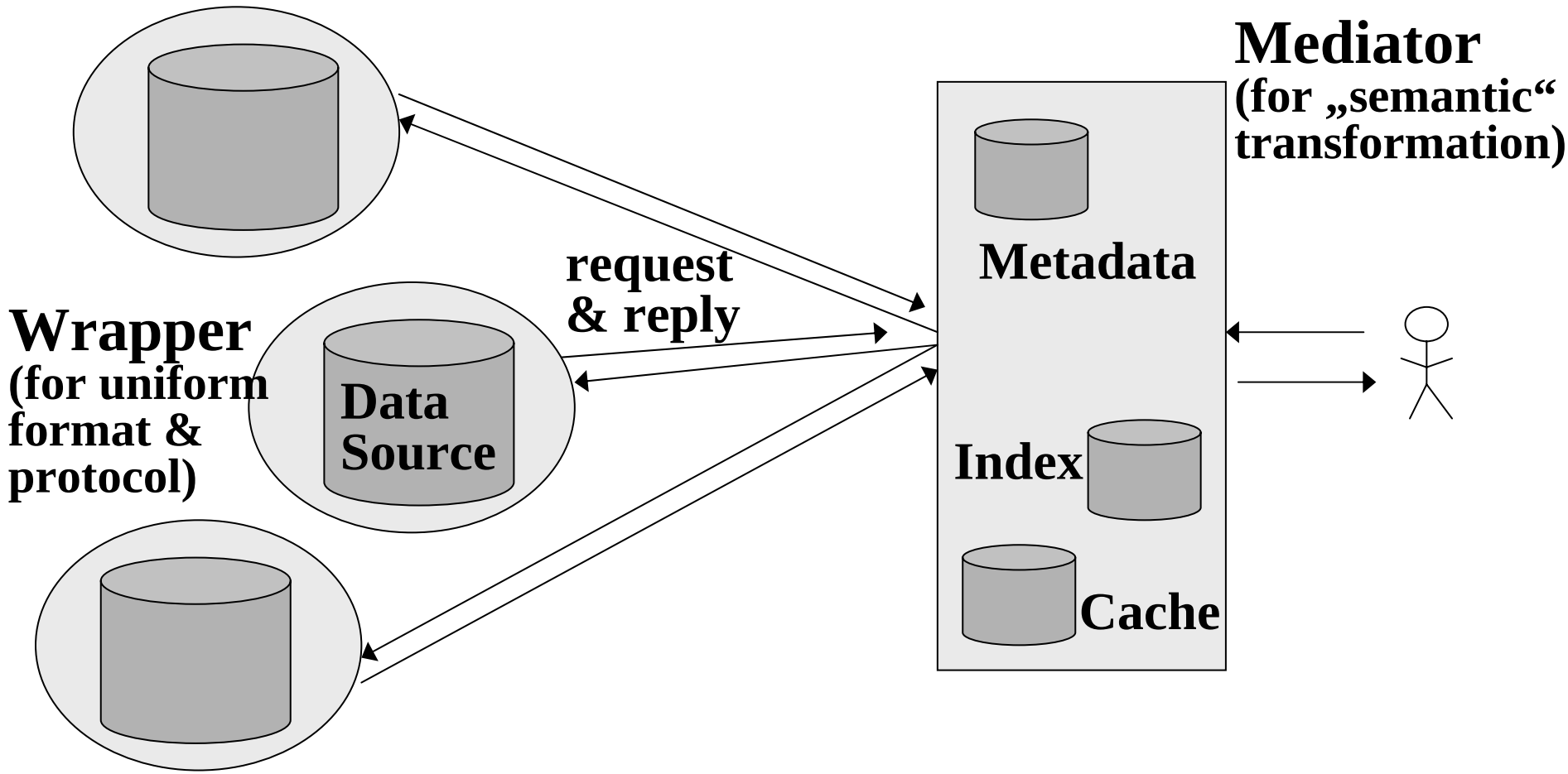


Data Warehouses (and Digital Libraries)



periodic or continuous, incremental ETL into
DW with star or snowflake schema (facts, dimensions)

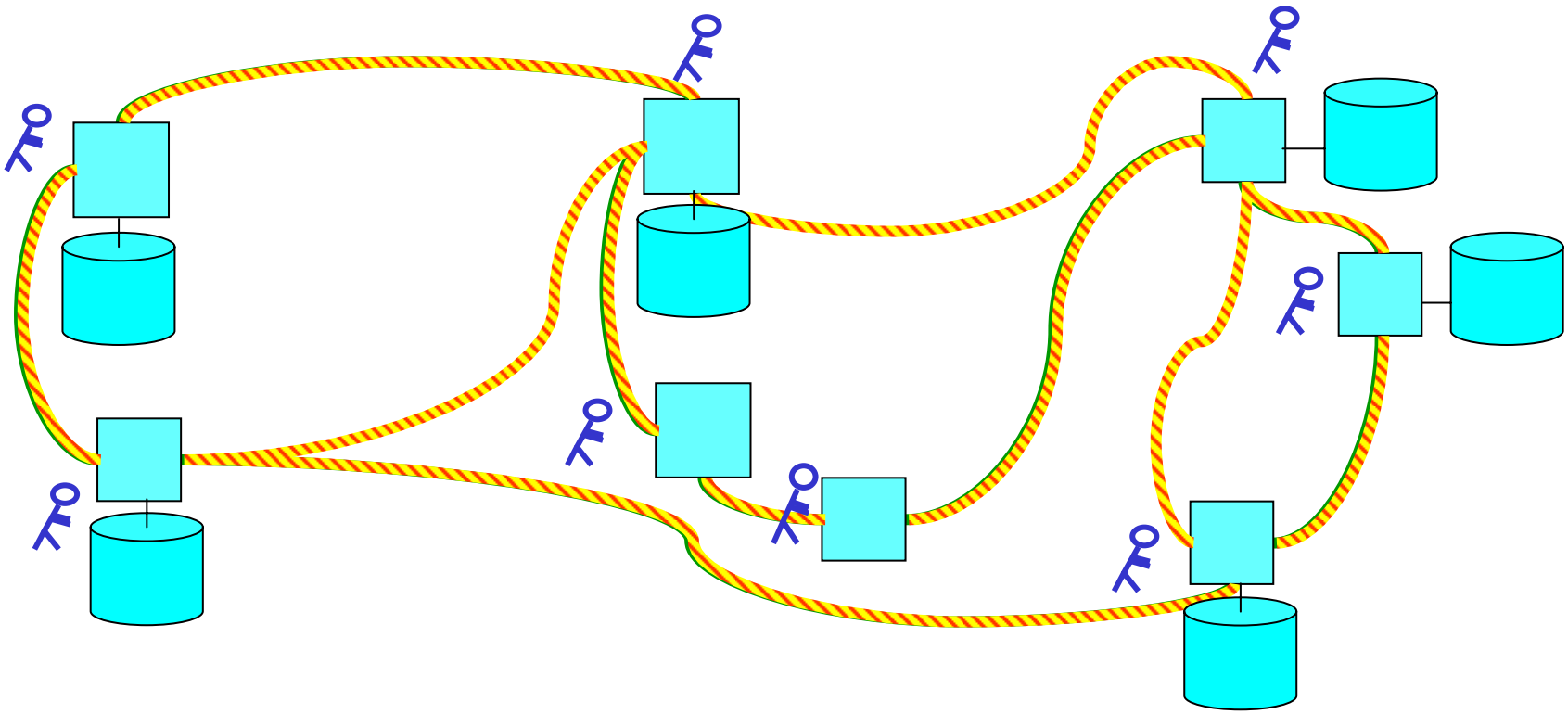
Wrapper-Mediator Architecture for Information Integration Systems



Wrappers may be hand-engineered or generated

Examples:
- **Yahoo for news, business, etc.**
- **SRS for bioinformatics & life sciences**
- **intranet portals for organizations**

Peer-to-Peer (P2P) Information Sharing



User post sharable files on autonomous computers (peers)
P2P system maintains (distributed) directory with file names
Users query file names and download files from other peers

Peer-to-Peer (P2P) Architectures

Decentralized, self-organizing, highly dynamic loose coupling of many autonomous computers

Applications:

- **Large-scale distributed computation (SETI, PrimeNumbers, etc.)**
- **File sharing (Napster, Gnutella, KaZaA, etc.)**
- **Publish-Subscribe Information Sharing (Marketplaces, etc.)**
- **Collaborative Work (Games, etc.)**
- **Collaborative Data Mining**
- **(Collaborative) Web Search**

Goals:

- **make systems ultra-scalable and completely self-organizing**
- **make complex systems manageable and less susceptible to attacks**
- **break information monopolies, exploit small-world phenomenon**

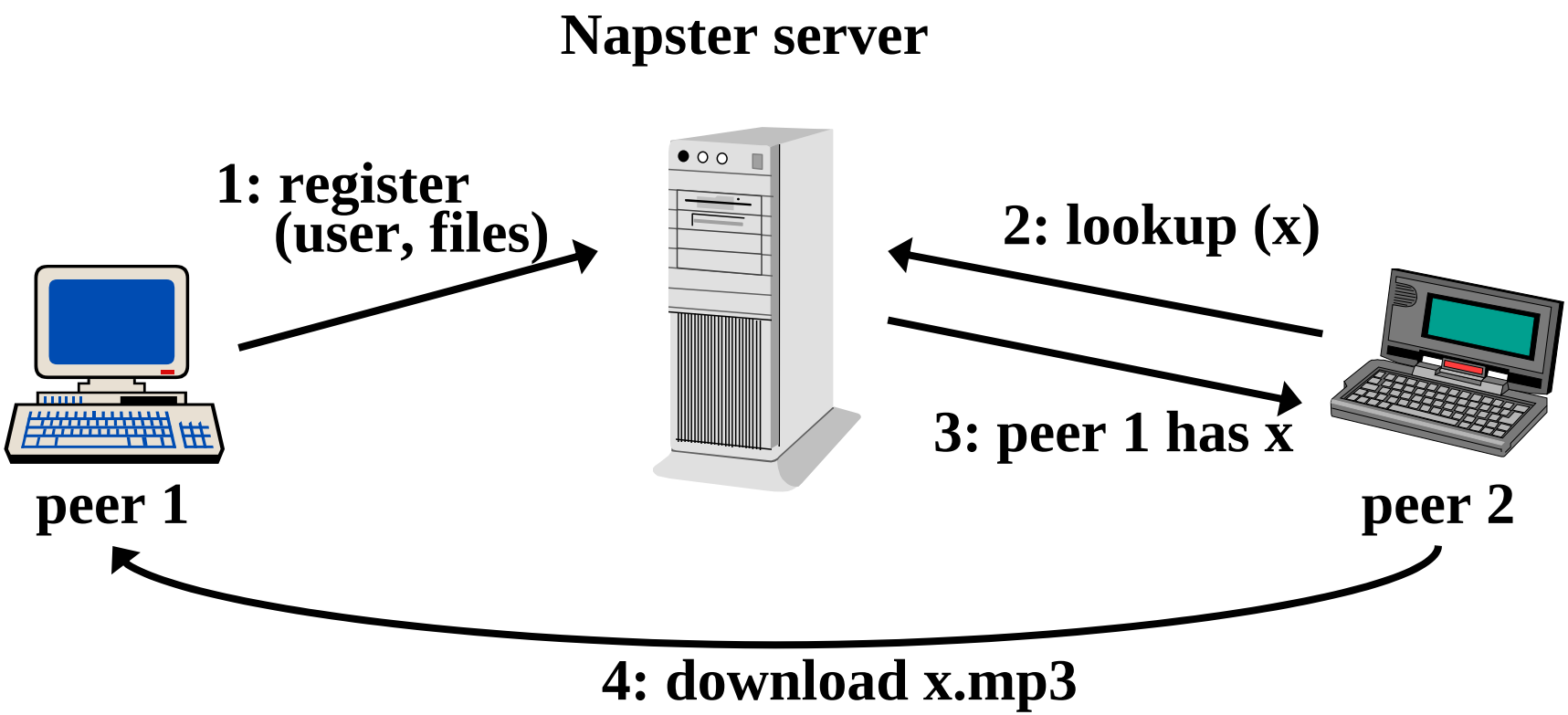
1st-Generation P2P

Napster (1998-2001) and Gnutella (1999-now):
driven by file-sharing for MP3, etc.
very simple, extremely popular

can be seen as a mega-scale but very simple publish-subscribe system:
- **owner of a file makes it available under name x**
- **others can search for x, find copy, download it**

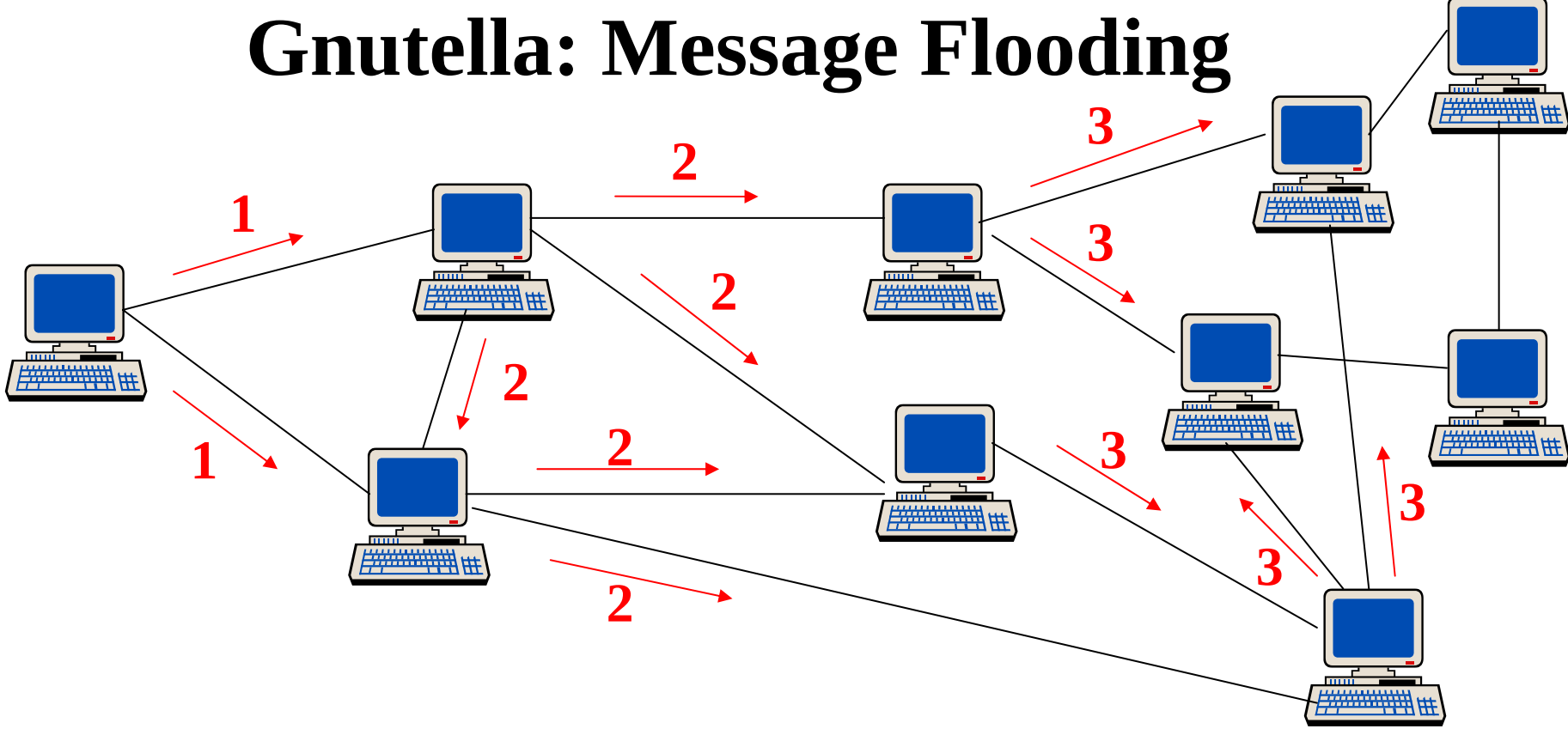
invitation to break the law (piracy, etc.) ?

Napster: Centralized Index



+ chat room, instant messaging, firewall handling, etc.

Gnutella: Message Flooding



all forward messages carry a TTL tag (time-to-live)

1) **contact neighborhood and establish virtual topology (on-demand + periodically): *Ping*, *Pong***
2) **search file: *Query*, *QueryHit***
3) **download file: *Get* or *Push* (behind firewall)**

2nd-Generation P2P

Freenet
emphasizes anonymity

eDonkey, KaZaA (based on FastTrack), Morpheus, MojoNation, AudioGalaxy, etc. etc.
commercial, typically no longer open source; often based on super-peers

JXTA
(Sun-sponsored) open API

Research prototypes (with much more refined architecture and advanced algorithms):
Chord (MIT), CAN (Berkeley), OceanStore/Tapestry (Berkeley), Farsite (MSR), Spinglass/Pepper (Cornell), Pastry/PAST (Rice, MSR), Viceroy (Hebrew U), P-Grid (EPFL), P2P-Net (Magdeburg), Pier (Berkeley), Peers (Stanford), Kademlia (NYU), Bestpeer (Singapore), YouServ (IBM Almaden), Hyperion (Toronto), Piazza (UW Seattle), PlanetP (Rutgers), SkipNet (MSR), Galanx (U Wisconsin), Minerva (MPII), etc. etc.

The Future of P2P: Challenging Requirements

Unlimited scalability with millions of nodes:
O(log n) hops to target, O(log n) state per node

Failure resilience, high availability, self-stabilization
(w.r.t. dynamics: many failures & high churn)

Data placement, routing, load management, etc.
in overlay networks

Robustness to DoS attacks & other traffic anomalies

Trustworthy computing and data sharing

Incentive mechanisms to reconcile selfish behavior
of individual nodes with strategic global goals

P2P-Related Technologies

Web Services (SOAP, WSDL, etc.)
for e-business interoperability (supply chains, etc.)

Grid Computing
for scientific data interoperability

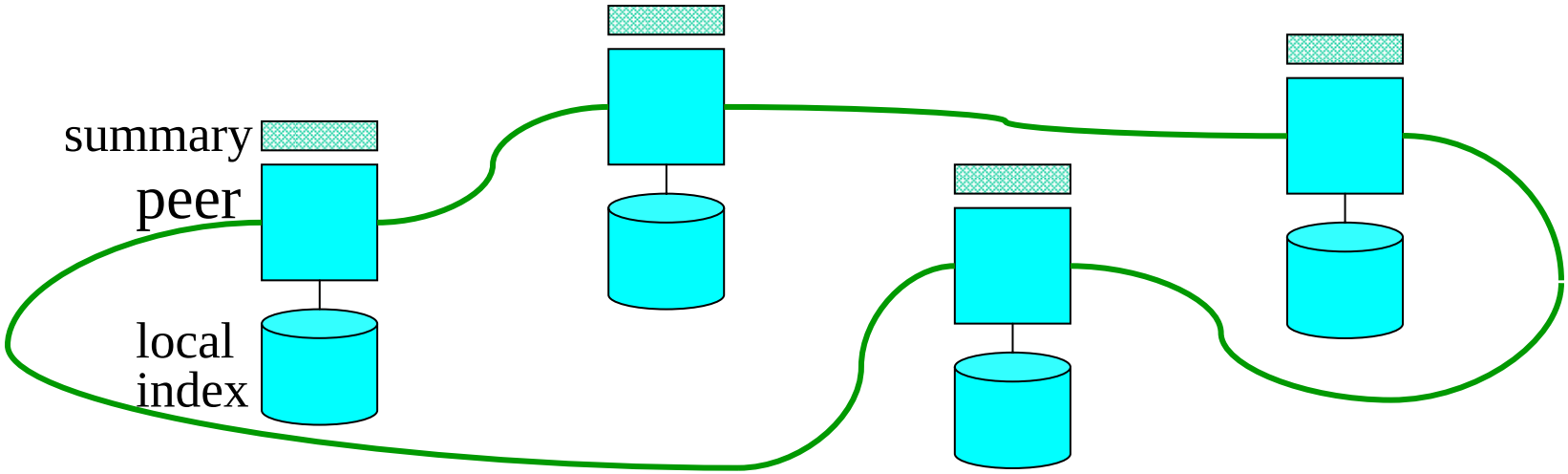
Autonomic / Organic / Introspective Computing
for self-organizing, zero-admin operation

Multi-Agent Technology
for interaction of autonomous, mobile agents

Sensor Networks
for data streams from measurement devices etc.

Content-Delivery Networks (e.g., Akamai)
for large content of popular Web sites

17.2 Anfrageausführung in verteilten IR-Systemen



- every peer is autonomous and has its own local search engine
- every peer posts (statistical) summary info about its contents (index lists, bookmarks, cached docs, QoS properties, ...)
- query routing is driven by similarity to summaries
- summaries are organized into a (distributed) directory
  - mapped onto DHT, random-graph overlay network, etc.
  - lazily replicated at additional peers (via „gossiping“)

querying peer needs to

1. determine interesting peers (query routing)
2. plan, run, monitor, and adapt distributed top-k algorithm
3. reconcile results from different peers

Why Peer-to-Peer Web Search?

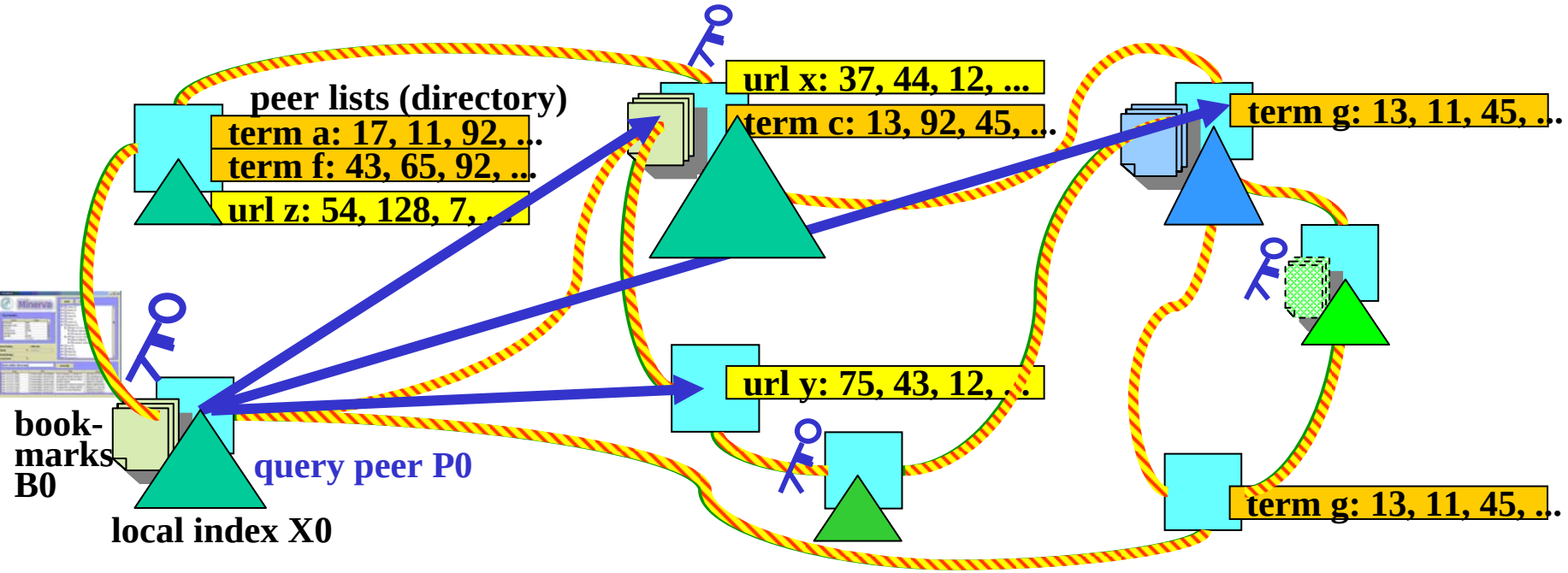
Goal: Self-organizing P2P Web Search Engine
with Google-or-better functionality

- Scalable & Self-Organizing Data Structures and Algorithms
  (DHTs, Semantic Overlay Networks, Epidemic Spreading, Distr. Link Analysis, etc.)
- Better Search Result Quality (Precision, Recall, etc.)
  - Powerful Search Methods for Each Peer
    (Concept-based Search, Query Expansion, Personalization, etc.)
  - Leverage Intellectual Input at Each Peer
    (Bookmarks, Feedback, Query Logs, Click Streams, Evolving Web, etc.)
  - Collaboration among Peers
    (Query Routing, Incentives, Fairness, Anonymity, etc.)
- Small-World Phenomenon
  Breaking Information Monopolies

Differences between Meta and P2P Search Engines

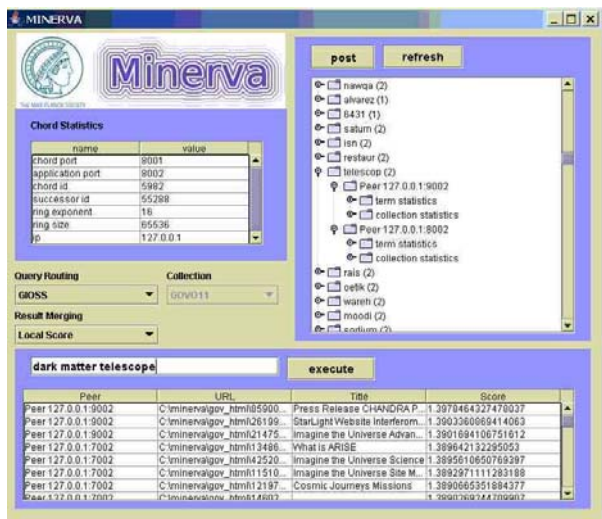
Meta Search Engine	P2P Search Engine
small # sites (e.g., digital libraries)	huge # sites
rich statistics about site contents	poor/limited/stale summaries
static federation of servers	highly dynamic system
each query fully executed at each site	single query may need content from multiple peers
interconnection topology largely irrelevant	highly dependent on overlay network structure

P2P Query Routing



Query routing aims to optimize benefit/cost driven by distributed statistics on peers' content similarity, content overlap, freshness, authority, trust, performability etc.

Dynamically precompute „good peers" to maintain a **Semantic Overlay Network** using random but biased graphs



Framework and Parameters for Query Routing

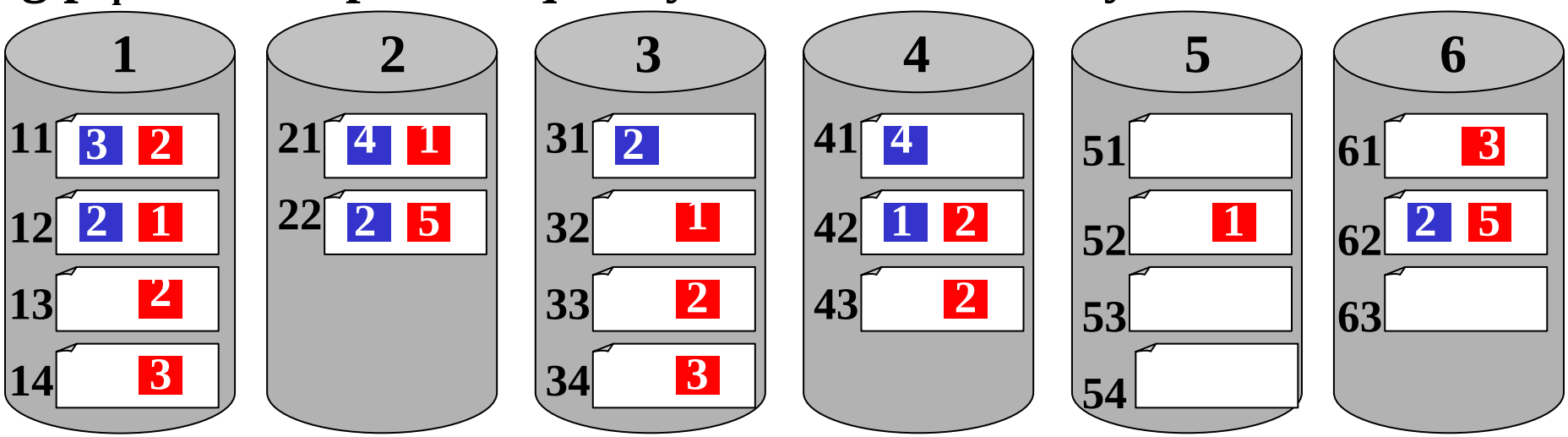
M terms t_i, N documents d_j, P peers with N_k docs at peer p_k

local measures at peer k:

- $tf_i^{(k)}(d)$ – freq. of term i in doc d
- $df_i^{(k)}$ - # docs with term i
- $idf_i^{(k)}$ - inverse doc freq. of term i
- $ttf_i^{(k)}$ – total freq. of term i
- $mtf_i^{(k)}$ – max. term freq.
- $mdf^{(k)}$ – max. doc freq.
- $t^{(k)}$ – # distinct terms

global measures:

- $gidf_i$ - inverse doc frequency of term i in P2P system
- $gipf_i$ - inverse peer frequency of term i in P2P system



select best peers based on expected benefit/cost ratio

Query Routing based on IPF (PlanetP)

Every peer conceptually maintains the global inverse peer frequency (gipf) for each term i:

$$gipf_i = log\left(1 + \frac{\#\ peers}{\#\ peers\ \ with\ term\ i}\right)$$

For multi-keyword query q the quality of peer j is:

$$R_j(q) := \sum_{i \in q} gipf_i \cdot \begin{cases} 1\ if\ \ peer\ \ j\ contains\ term\ i \\ 0\ otherwise \end{cases}$$

To retrieve top k results for query q:

1. **rank peers in descending order of Rj(q)**
2. **contact peers in groups of m in rank order**
3. **merge results**
4. **iterate steps 2 and 3 until no peer contributes to top-k result**

Example: Query Routing based on IPF

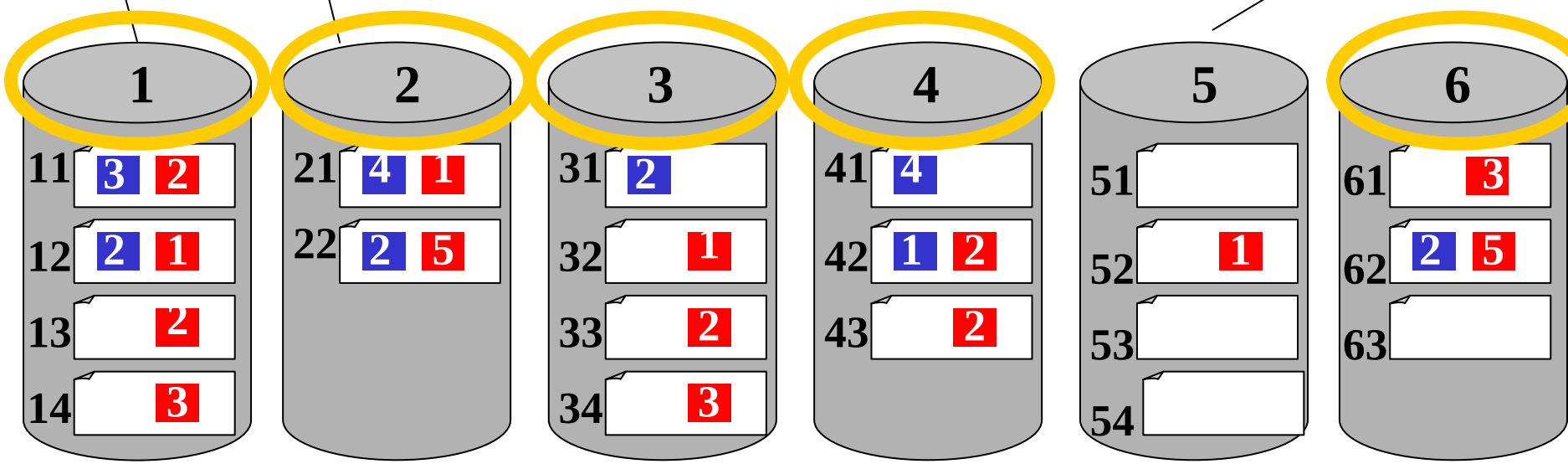
gipf■ = log (1 + 6/5) gipf■ = log (1+1)

R_1(■ ■) = log 11/5 * 1 + log 2 * 1

R_2(■ ■) = log 11/5 * 1 + log 2 * 1

•••

R_5(■ ■) = log 2 * 1



PlanetP Implementation

Each peer posts its summary in the form of a *Bloom-filter signature*:

- **bit vector S[1..s] of fixed length s, initially all bits zero**
- **if peer j has term i it sets bit h(i) to one using a hash function h**
- **other peers can test if peer j holds term set $\{q_1, ..., q_k\}$ by looking up $S[h(q_1)], ..., S[h(q_k)]$ or by computing a bit vector Q[1..s] for $\{q_1, ..., q_k\}$ and ANDing S with Q, both with the risk of *„false positives“***

Summaries are sent to other peers by asynchronous *gossiping* in a combined push/pull mode:

- ***push*: periodically send updates of global registry (small Δs) as „rumors“ to randomly chosen neighbors; stop doing so when n consecutive peers already know the update**
- **(anti-entropy) *pull*: periodically ask randomly chosen neighbor to send an updated summary of the global registry; alternatively ask push-recipient for recent rumors**

Query Routing based on Simple Heuristics

Consider df, ttf, mtf, gipf as quality measures of a peer

Choose peers in descending order of

$$\sum_{i \in q} \alpha_1 \log df_i^{(k)} + \alpha_2 \log mtf_i^{(k)} + \alpha_3 \log ttf_i^{(k)}$$ **or**

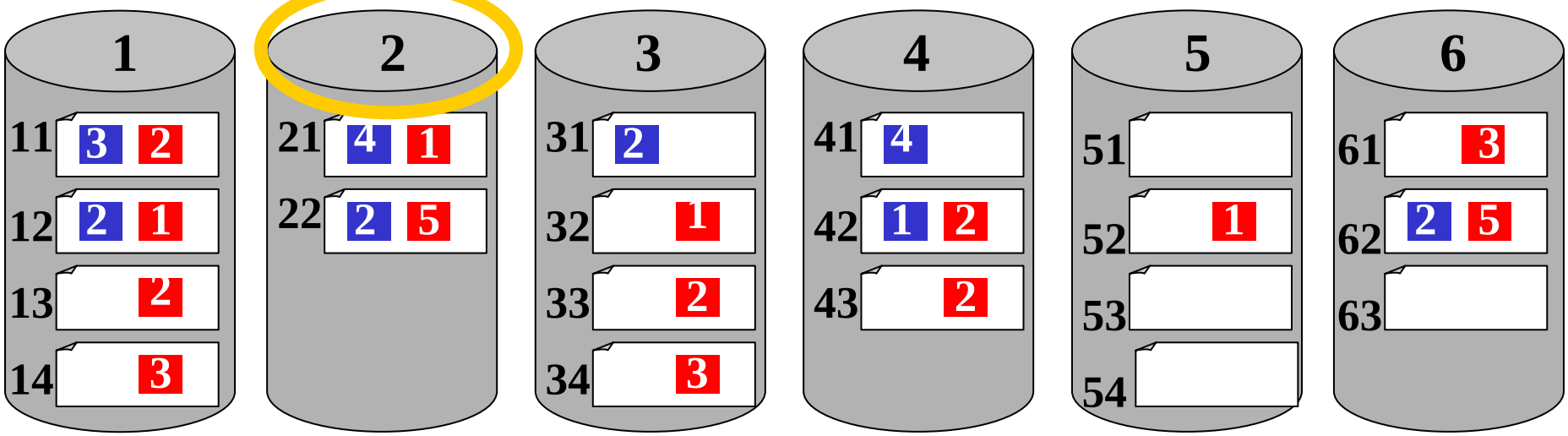
$$\sum_{i \in q} gipf_i \cdot \left(\alpha_1 \log df_i^{(k)} + \alpha_2 \log mtf_i^{(k)} + \alpha_3 \log ttf_i^{(k)} \right)$$

with tunable weights α_1, α_2, α_3 such that $\alpha_1 + \alpha_2 + \alpha_3 = 1$

Example: Query Routing Heuristics

gipf = log (1 + 6/5) gipf = log (1+1)

1	2	3	4	5	6
df = 2	df = 2	df = 1	df = 2	df = 0	df = 1
df = 4	df = 2	df = 3	df = 2	df = 1	df = 2
mtf = 3	mtf = 4	mtf = 2	mtf = 4	mtf = 0	mtf = 2
mtf = 3	mtf = 5	mtf = 3	mtf = 2	mtf = 1	mtf = 5
ttf = 5	ttf = 6	ttf = 2	ttf = 5	ttf = 0	ttf = 2
ttf = 8	ttf = 6	ttf = 6	ttf = 4	ttf = 1	ttf = 8



Query Routing based on Statistical Similarity

For query q select peers p with highest value of sim(q, p), e.g., cosine(q, p) where p is represented by its centroid

Use statistical language model for similarity:

$$KL(q \,||\, p) = \sum_{t \in q} P[t|q] \; log \frac{P[t|q]}{\lambda P[t|C_p] + (1-\lambda) P[t|G]}$$

where P[t|q], P[t|Cp], P[t|G] are the (estimated) probabilities that term t is generated by the language models for the query q, the corpus C_k of peer k, and the general vocabulary, and λ is a smoothing parameter between 0 and 1

Implementation may estimate $P[t|C_k] \approx ttf_t^{(k)} / \Sigma_i \, ttf_i^{(k)}$

The Kullback-Leibler divergence (aka. relative entropy) is a measure for the distance between two probability distributions:

$$KL(f \,\|\, g) := \sum_x f(x) \log \frac{f(x)}{g(x)}$$

CORI Query Routing

Apply probabilistic IR method with heuristic elements (Okapi BM25 term weighting) to peer selection by treating a peer's complete index contents as a „document"

$$P[q \mid p] \sim$$

$$\sum_{i \in q} \frac{ttf_i^{(k)}}{ttf_i^{(k)} + 0.5 + 1.5\, t^{(k)} / avg_v(t^{(v)})}$$

$$\cdot \frac{log(\, gipf_t \cdot (0.5 + \#\, peers\,))}{log(1 + \#\, peers\,)}$$

GlOSS Query Routing based on Goodness

Goodness (q, s, l) = $\sum$ {sim(q, d) | d $\in$ result(q, s) $\wedge$ lsim(q,d)>l}
for query q, source s, and score threshold l

GlOSS (Glossary Of Servers Server) aims to rank sources by goodness

Approximate goodness by using for source s:
- **df_i(s): number of docs in s that contain term i**
- **w_i(s): $\sum$ {tf_i(d)*idf_i | d $\in$ s} (total weight of term i in s)**

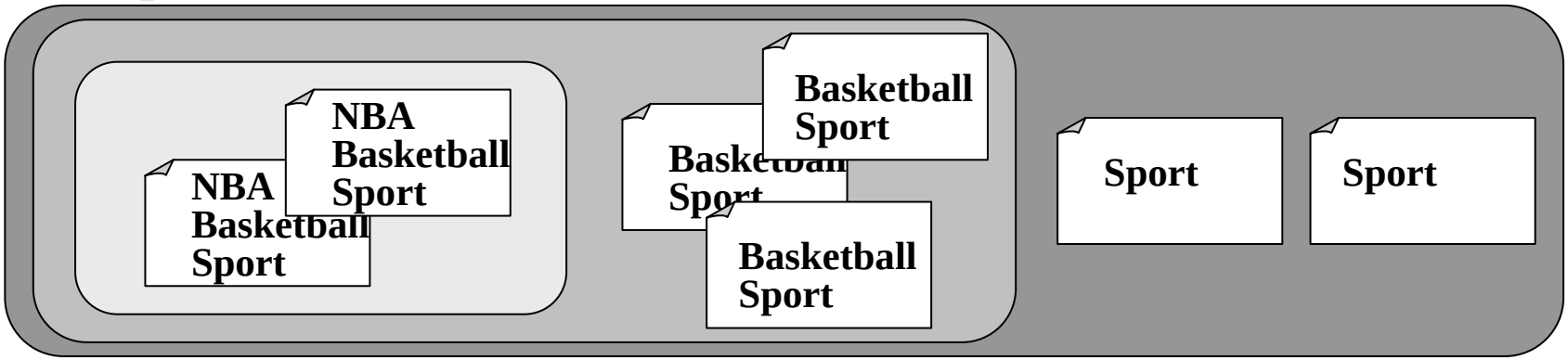
Uniformity assumption:
w_i(s) is distributed uniformly over all docs in s that contain i

GlOSS Goodness with High-correlation Assumption

High-correlation assumption:

$df_i(s) \leq df_j(s) \Rightarrow$ every doc in s that contains i also contains j

Example:



For fixed source s and query $q = t_1 \ldots t_n$ with $df_i \leq df_{i+1}$ for i=1..n-1 consider subqueries $q_p = t_p \ldots t_n$ (p=1..n).
Every doc d in s that contains only $t_p \ldots t_n$ has query similarity

$$sim_p(q,d) = \sum_{i=p..n} t_i \frac{w_i(s)}{df_i(s)}$$

Find p such that $sim_p(q,d) > l$ and $sim_{p+1}(q,d) \leq l$

EstGoodness(q,s,l) = $\sum_{i=1..p} (df_i(s) - df_{i-1}(s)) * sim_i$

GlOSS Goodness with Disjointness Assumption

Disjointness assumption:
{d∈s|d contains term i} ∩ {d∈s|d contains term j} = ∅ for all i,j ∈q

Uniformity assumption:
$w_i(s)$ is distributed uniformly over all docs in s that contain i

$$sim_i(q,d) = t_i \frac{w_i(s)}{df_i(s)}$$

EstGoodness(q,s,l) $$= \sum_{i=1..n \wedge sim_i > l} df_i(s) \cdot t_i \frac{w_i(s)}{df_i(s)}$$

$$= \sum_{i=1..n \wedge sim_i > l} t_i \; w_i(s)$$

Usefulness Estimation Based on MaxSim

Def.: A set S of sources is *optimally ranked* for query q in the order s1, s2, ..., sm if for every n>0 there exists k, $0<k\leq m$, such that s1, ..., sk contain the n best matches to q and each of s1, ..., sk contains at least one of these n matches

Thm.: Let MaxSim(q,s) = max{sim(q,d)|$q \in s$}.
s1, ..., sm are optimally ranked for query q if and only if MaxSim(q,s1) > MaxSim(q,s2) > ... > MaxSim(q,sm).

Practical approach („Fast-Similarity method“):
Capture, for each s, $df_i(s)$, $avgw_i(s)$, $maxw_i(s)$ as source summary.
Estimate for query q = t1 ... tk

$$MaxSim(q,s) := \max_{i=1..k} \{t_i * maxw_i(s) + \sum_{v \neq i} t_v * avgw_v(s)\}$$

estimation time linear in query size,
space for statistical summaries linear in #sources * #terms

Overlap Aware Query Routing

First execute q on initiator peer's local index X0,
then select peers Pi with highest benefit/cost ratio where

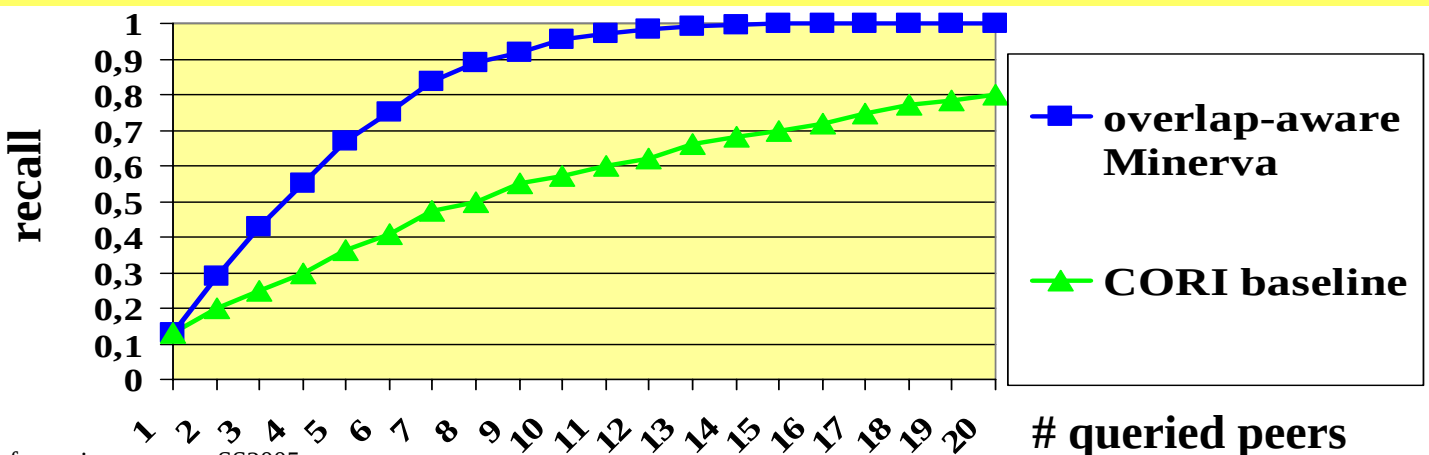
- **benefit(Pi) ~ sim (X0, Xi) and ~ 1/overlap(X0, Xi)**
- **cost(Pi) ~ estimated response time or communication costs**

precompute sim: $$KL(X0, Xi) := \sum_{terms\ x} freq(x, X0)\ log \frac{freq(x, X0)}{freq(x, Xi)}$$

approximate overlap by Bloom filters, hash sketches, etc.

Experiments:

based on 100 .Gov partitions (1.25 Mio. docs), assigned to 50 peers, with each peer holding 10 partitions and 80% overlap for P_i, P_{i+1} with 50 TREC-2003 Web queries, e.g.: „juvenile delinquency"



Distributed Query Execution Issues

Algorithm:

- **Determine the number of results to be retrieved from each source a priori based on the source‘s content quality**

vs.

- **Run distributed version of top-k query processing algorithm**

Dynamic adaptation:

- **Plan query execution only once before initiating it vs.**
- **Dynamic plan adjustment based on sources‘ result quality and responsiveness (incl. failures)**

Parallelism:

- **Start querying all selected sources in parallel vs.**
- **Consider (initial) results from one source when querying the next sources**

Result Reconciliation

Case 1: all peers use the same scoring function,
e.g. cosine similarities based on tf*idf weights

Case 2: peers may use different scoring functions
that are publicly known

Case 3: peers may use different & unknown scoring functions
but provide scored results

Case 4: peers provide only result rankings, no scores

Baseline case:
when gidf values are known at the query initiator,
we can recompute tf values from the different peers‘ result docs
and compute global scores based on gidf and tf values

Techniques for Result Reconciliation (1)

for case 1:

local sim is

$$lsim(\vec{q},\vec{d}) = \sum_i \frac{q_i \cdot tf_i(\vec{d}) \cdot lidf_i}{\sqrt{\sum_i {q_i}^2} \cdot \sqrt{\sum_i tf_i(\vec{d})^2 \cdot {lidf_i}^2}}$$

global sim is

$$sim(\vec{q},\vec{d}) = \sum_i \frac{q_i \cdot tf_i(\vec{d}) \cdot gidf_i}{\sqrt{\sum_i {q_i}^2} \cdot \sqrt{\sum_i tf_i(\vec{d})^2 \cdot {gidf_i}^2}}$$

either recompute *tf* of result docs and compute *sim*

or submit additional single-term queries (one for each query term) such that each result d to the original query q is retrieved:

$$lsim(q_i,\vec{d}) = \frac{q_i \cdot tf_i(\vec{d}) \cdot lidf_i}{q_i \cdot \sqrt{\sum_j tf_j(\vec{d})^2 \cdot {lidf_j}^2}} = \frac{tf_i(\vec{d}) \cdot lidf_i}{\sqrt{\sum_j tf_j(\vec{d})^2 \cdot {lidf_j}^2}}$$

$$\Rightarrow \frac{lidf_i}{\sqrt{\sum_j tf_j(\vec{d})^2 \cdot {lidf_j}^2}} = \frac{lsim(q_i,\vec{d})}{tf_i(\vec{d})}$$

solve equation system for *tf* and *lidf* values (if possible) and compute *sim*

Techniques for Result Reconciliation (2)

for case 4:

set global score of doc j retrieved from source i to

$$g(d_j) := 1 - (r_{local}(d_j) - 1) \cdot \frac{r_{\min}}{m \cdot r_i}$$ **where**

- $r_{local}(dj)$ **is the local rank of** d_j**,**
- r_i **is the score of source i among the queried sources,**
- r_{min} **is the lowest such score, and**
- **m is the number of desired global results**

Intuition:

- **initially local ranks are linearly mapped to scores**
- **the factor** $r_{min} / (m\ r_i)$ **is the score difference for consecutive ranks from source i**

Wrap-Up: P2P Query Routing & Execution

Research on distributed IR has provided many approaches – principled as well as heuristic ones – that can be carried over to a P2P setting

However, the scale, dynamics, and usage patterns of a P2P search engine entail additional issues, many of which are widely open:

- **peer statistics collection and dissemination (frequency, quality, overlap, resource util., response times, etc.)**
- **precomputation of good peers → „semantic overlay network“**
- **consideration of strong correlations**
- **combination with PageRank-style authority etc.**
- **consideration of P0 query execution and feedback**
- **coping with tradeoffs in network bandwidth & latency, per-peer resource consumption, search result quality**

Literature

- Communications of the ACM, Vol 46, No. 2, Special Section on
- Peer-to-Peer Computing, Feb. 2003.
- Weiyi Meng, Clement Yu, King-Lup Liu: Building Efficient and Effective Metasearch Engines, ACM Computing Surveys 34(1), 2002
- F.M. Cuenca-Acuna, C. Peery, R.P. Martin, T.D. Nguyen: PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities, IEEE Symp. on HPDC, 2003
- Jie Lu, Jamie Callan: Content-Based Retrieval in Hybrid Peer-to-Peer Networks, CIKM Conference, 2003.
- Henrik Nottelmann, Norbert Fuhr: Evaluating Different Methods of Estimating Retrieval Quality for Resource Selection, SIGIR 2003
- M. Bender, S. Michel, P. Triantafillou, G. Weikum, C. Zimmer: Improving Collection Selection with Overlap Awareness in P2P Search Engines, SIGIR 2005
- M. Bender, S. Michel, G. Weikum, C. Zimmer: Das MINERVA-Projekt: Datenbankselektion für Peer-to-Peer-Websuche, Informatik Forschung und Entwicklung, 2005

17.3 Skalierbare verteilte Indexierung und Suche

Goals:

Decentralize

data store (MP3 files, Web documents, etc.) or
index (term-doc-score entries) or
directory (statistical info about peers)

across N peers with large N
and provide file-name / keyword lookup

with good properties:

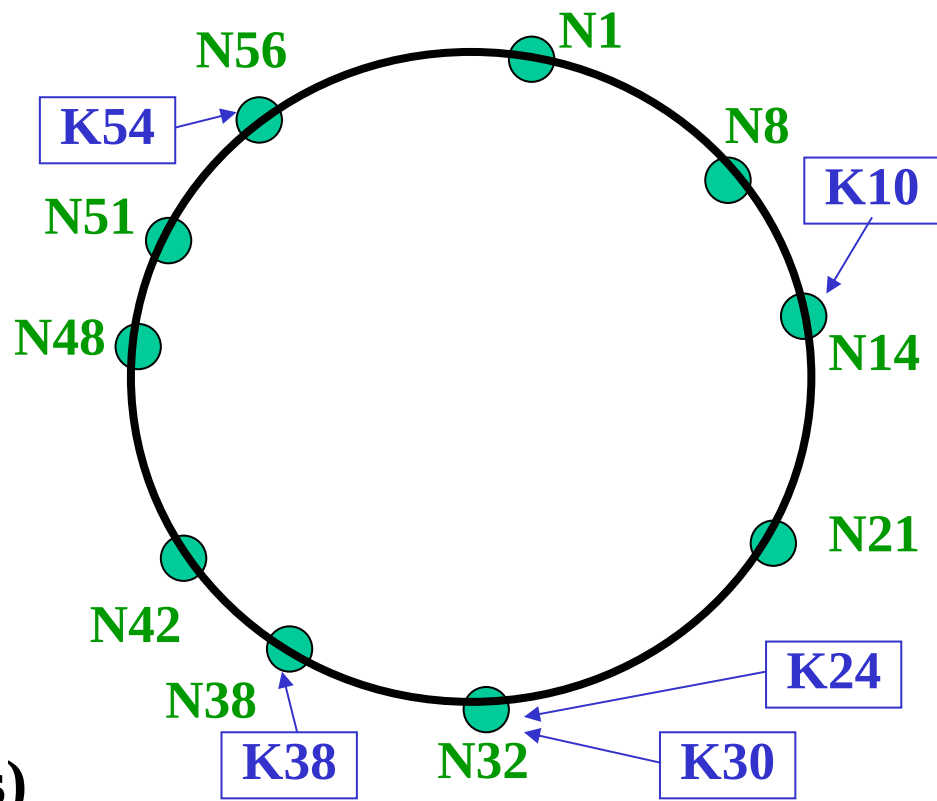
- **scalability: throughput is proportional to N, response time is independent of N (or grows very slowly with N)**
- **efficiency: overhead should be O(1) or ≤ O(log N)**
- **load balance: factor between least loaded and most loaded peer should be O(log N) or even O(log log N)**
- **robustness to failures: peers being temporarily down**
- **robustness to churn: peers joining and leaving at high rate**
- **self-organization regarding**
  data growth, load growth, dynamics of system and load patterns

Structured P2P: Example Chord

Distributed Hash Table (DHT):

map strings (file names, keywords) and numbers (IP addresses) onto very large „cyclic“ key space $0..2^m-1$, the so-called *Chord Ring*

Key k (e.g., *hash*(file name)) is assigned to the node with key n (e.g., *hash*(IP address)) such that k ≤ n and there is no node n‘ with k ≤ n‘ and n‘<n



Properties:

Unlimited scalability (> 10^6 nodes)

O(log n) hops to target, O(log n) state per node

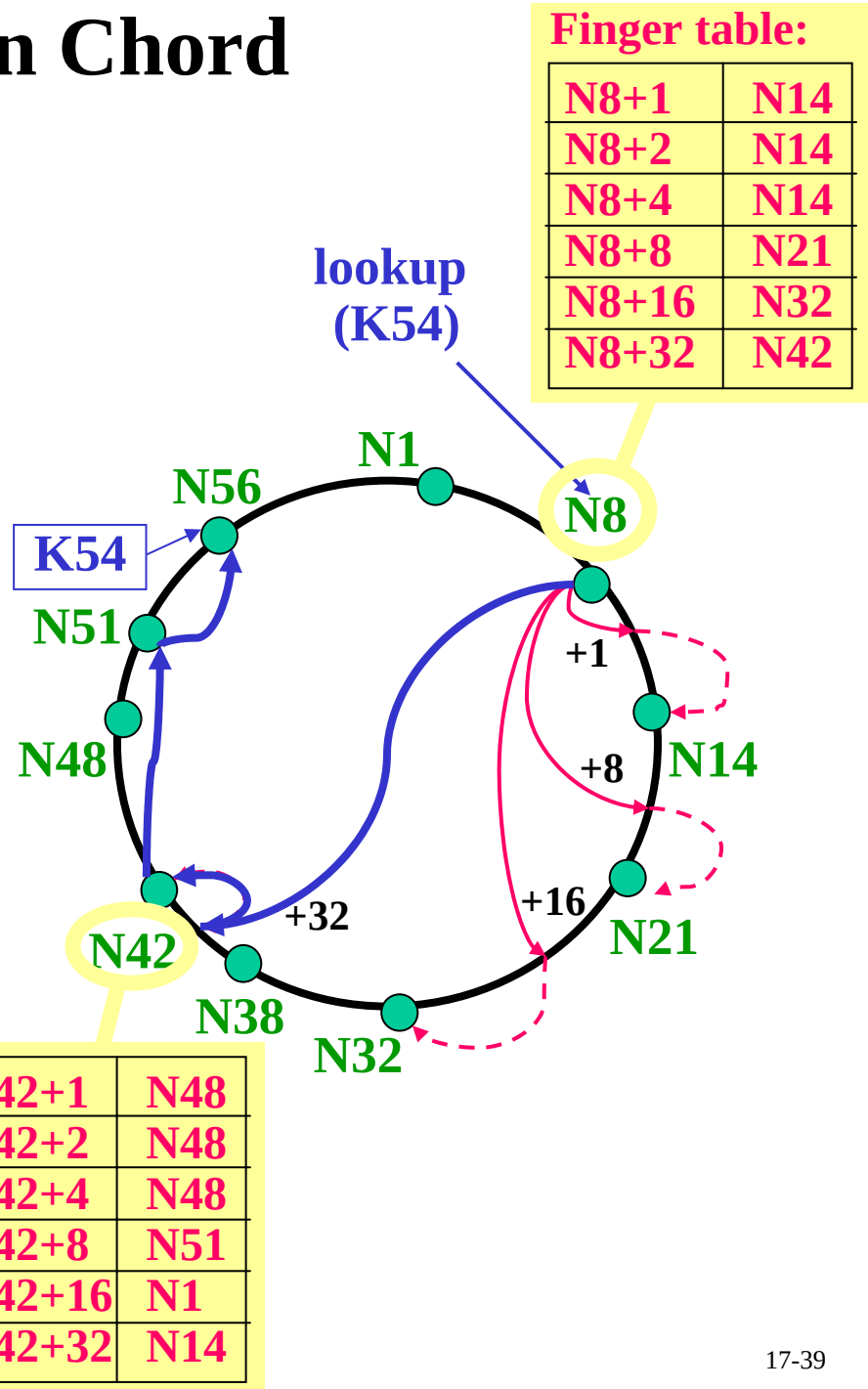
Self-stabilization (many failures, high dynamics)

Request Routing in Chord

Every node knows its pred/succ and has a finger table with log(n) pointers: finger[i] = successor (node number + 2^{i-1}) for i=1..m

For finding key k perform repeatedly: determine current node's largest finger[i] (modulo 2^m) with finger[i] ≤ k

pred/succ ring and finger tables require dynamic maintenance → stabilization protocol



17.4 Anfrageausführung in verteilten DBS

Beispiel: Netzwerkmonitoring