Informationssysteme

Kapitel 17 – Einführung in XML

XML - Snake Oil of this Century?

- *Snake Oil* is the all-curing drug these strange guys in wild-west movies sell, travelling from town to town, but visiting each town only once.
- Google: „snake oil“ xml

  ⇒ some 2000 hits
  - „XML revolutionizes software development“
  - „XML is the all-healing, world-peace inducing tool for computer processing“
  - „XML enables application portability“
  - „Forget the Web, XML is the new way to business“
  - „XML is the cure for your data exchange, information integration, data exchange, [x-2-y], [you name it] problems“
  - „XML, the Mother of all Web Application Enablers“
  - „XML has been the best invention since sliced bread“

XML is not…

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

But then – what is it?

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

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

XML by Example

```
<article>
  <author>Gerhard Weikum</author>
  <title>The Web in 10 Years</title>
</article>
```

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

This looks nice, but…

XML by Example

… this is XML, too:

```
<t108>
  <x87>Gerhard Weikum</x87>
  <g10>The Web in 10 Years</g10>
</t108>
```

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

XML by Example

… and what about this XML document:

```
<data>
  ch37fhgks73j5mv9d63h5mgfkds8d984lgnsmcns983
</data>
```

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

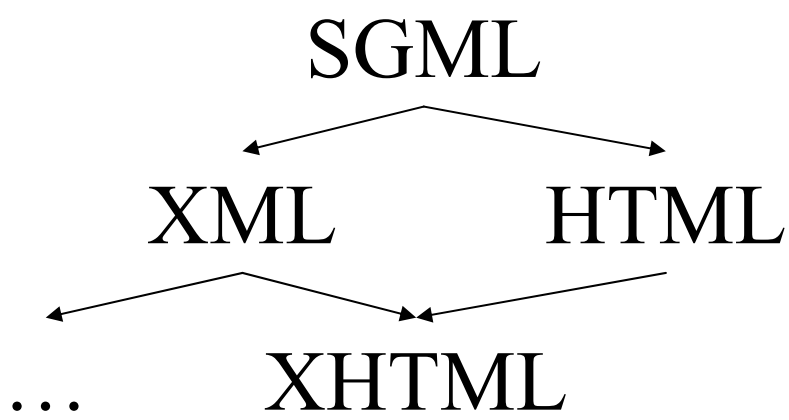
The actual benefit of using XML highly depends on the design of the application.

Possible Advantages of Using XML

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

History of XML

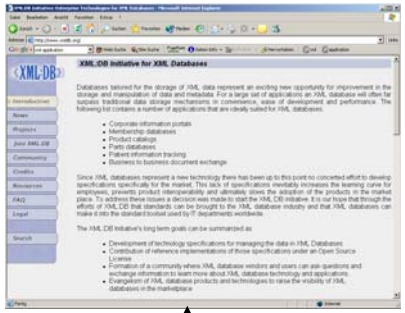
- Based on (i.e., an application of) SGML (**S**tandard **G**eneralized **M**arkup **L**anguage) from 1970ies
  - HTML most prominent SGML application
  - Too complicated and complex (>150 pages of spec)
  - Hardly ever fully implemented, only incompatible subsets
- XML as „SGML light“ on February 10th,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



Design Goals for XML

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

App. Scenario 1: Content Mgt.



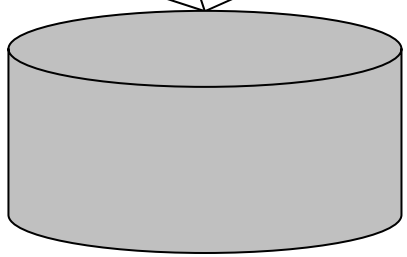
Clients

XML2HTML

XML2WML

XML2PDF

Converters



Database with
XML documents

App. Scenario 1: Examples

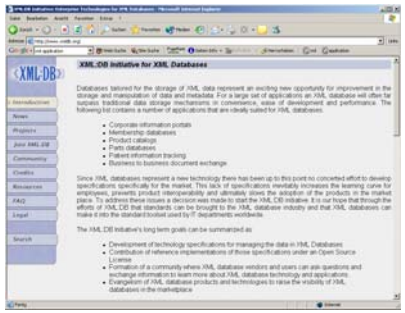
- The Linux Documentation Project **LDP**
- Digital Bibliography & Library Project **DBLP**
- Many Content Management Systems internally store information as XML and convert on-the-fly, among them **Apache Cocoon** (but this is typically not visible from the generated HTML pages)

Advantages:

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

But: Additional Load on the Server (→Caching)

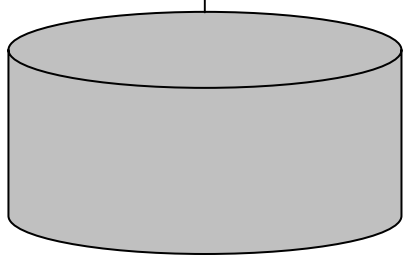
App. Scenario 1 in the future



Clients

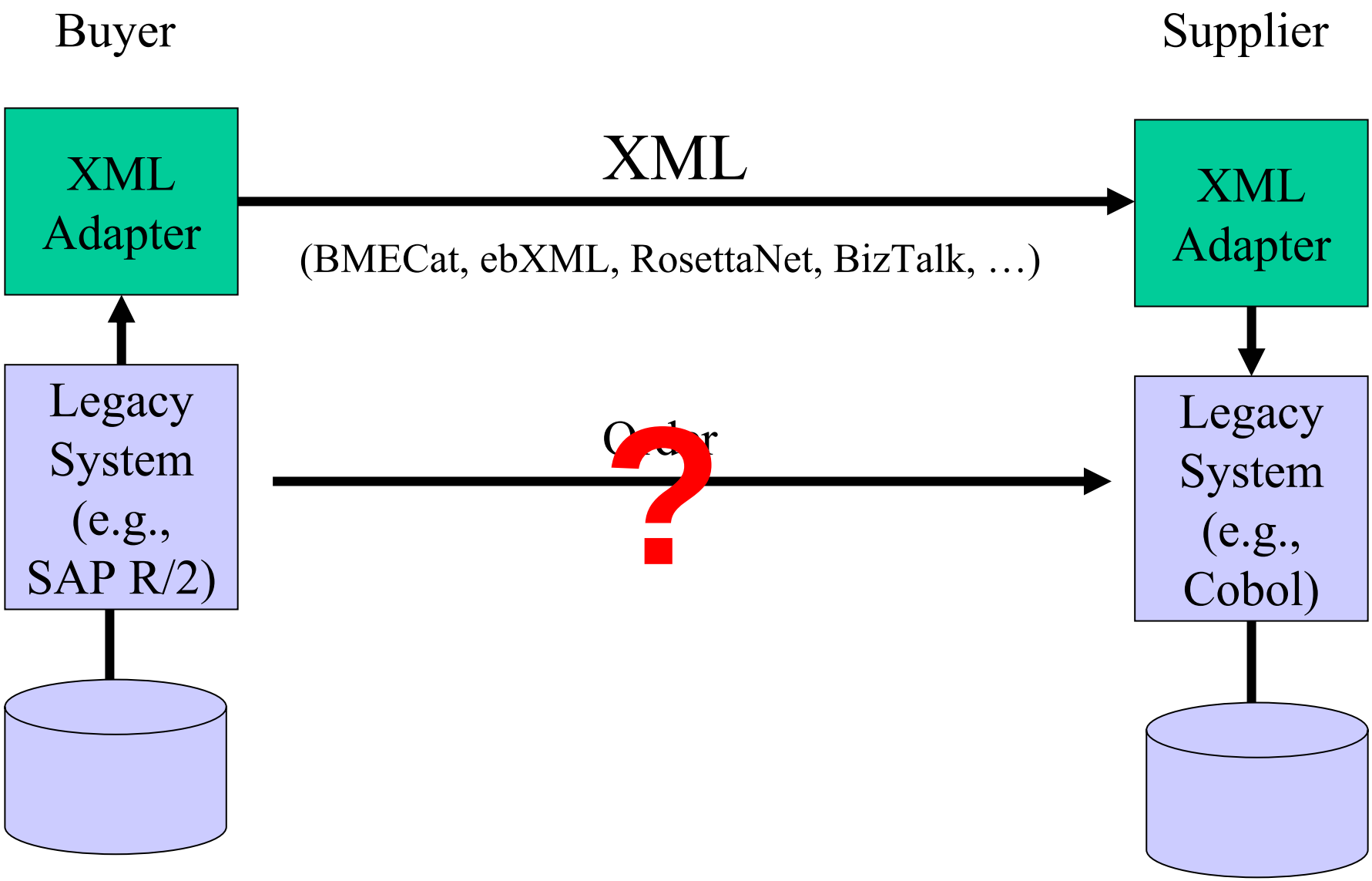
XML + converter

Web Server



Database with
XML documents

App. Scenario 2: Data Exchange



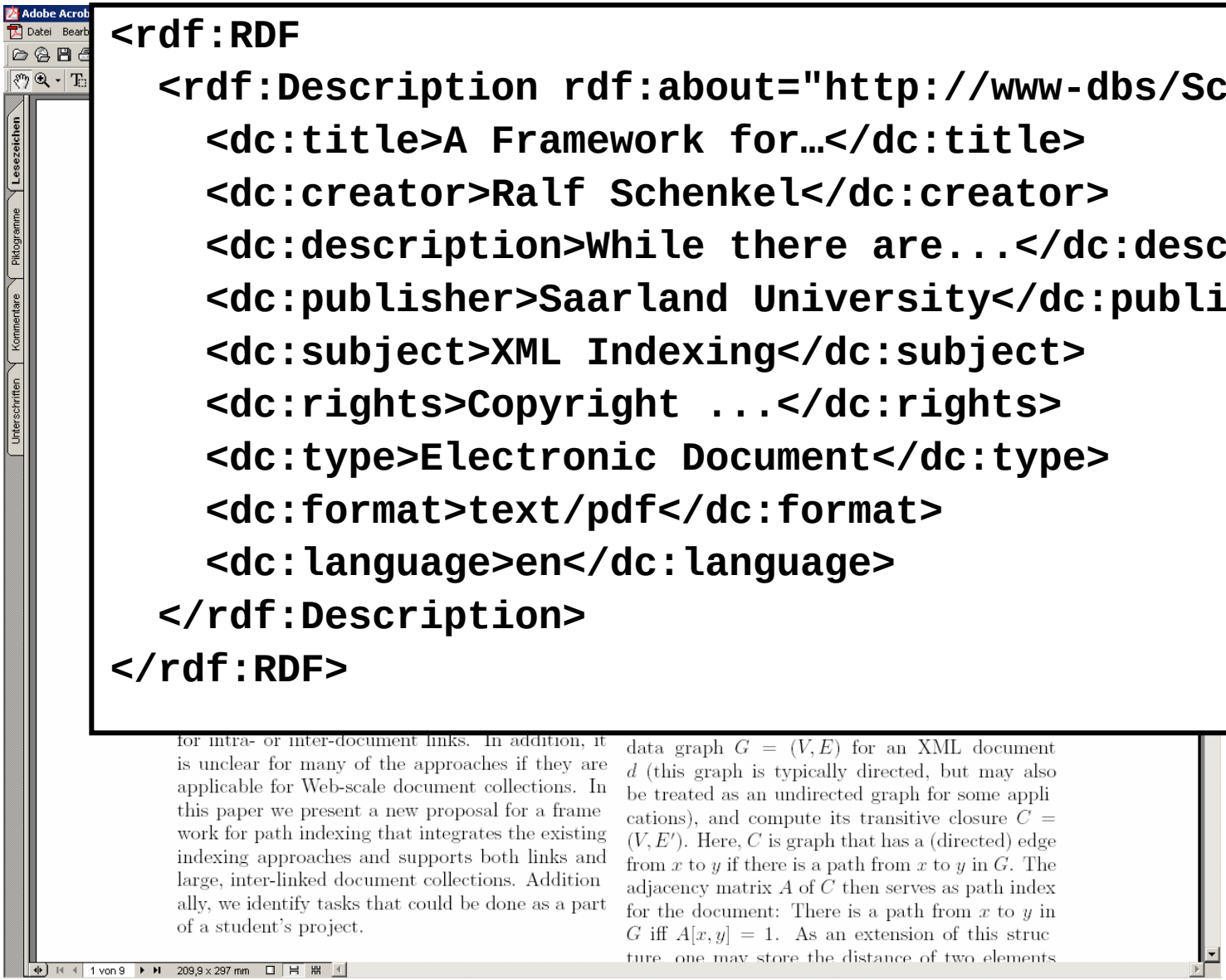
App. Scenario 2: Data Exchange

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

But: All Adapters must support the same XML application

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

App. Scenario 3: XML for Metadata



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

App. Scenario 4: Document Markup

```
<article>
  <section id=„1“ title=„Intro“>
    This article is about <index>XML</index>.
  </section>
  <section id=„2“ title=„Main Results“>
    <name>Weikum</name> <cite idref=„Weik01“/> shows
   the following theorem (see Section <ref idref=„1“/>)
    <theorem id=„theo:1“ source=„Weik01“>
      For any XML document x, ...
    </theorem>
  </section>
  <literature>
    <cite id=„Weik01“><author>Weikum</author></cite>
  </literature>
</article>
```

App. Scenario 4: Document Markup

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

XML Standards – an Overview

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

17.3 XML Documents

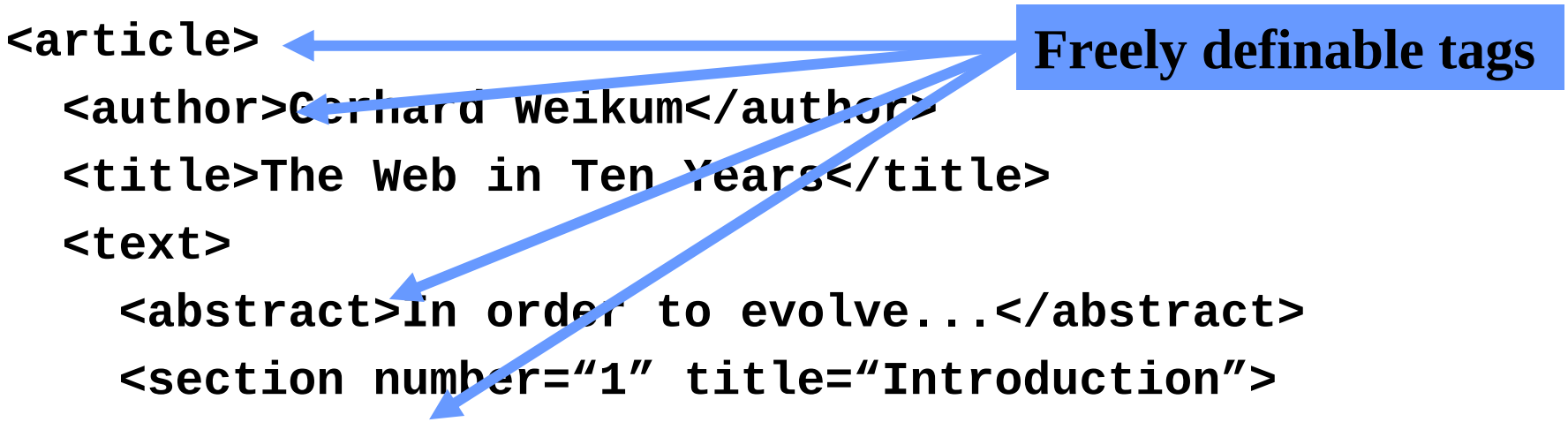
What‘s in an XML document?

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

A Simple XML Document

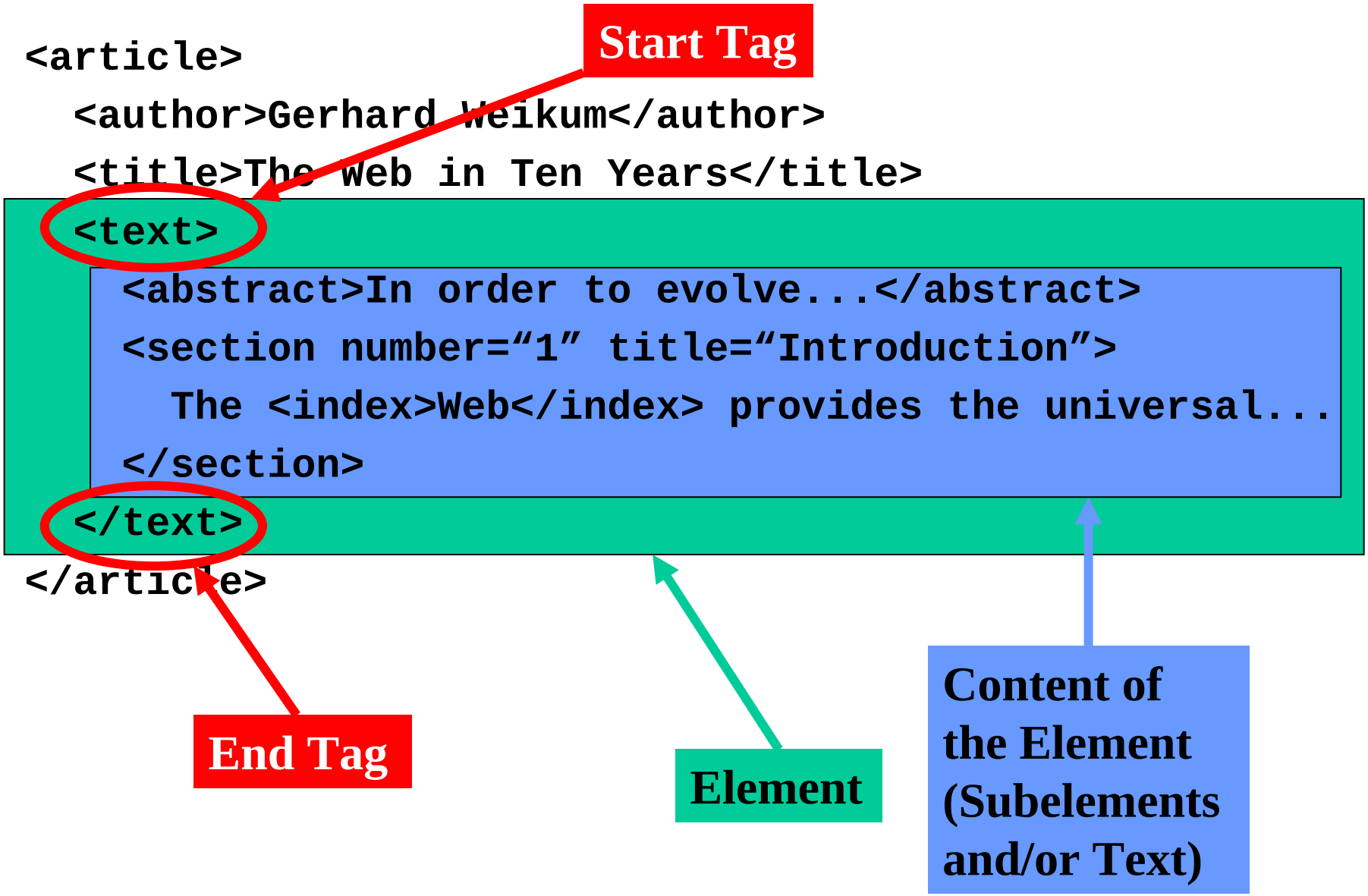
```
<article>
  <author>Gerhard Weikum</author>
  <title>The Web in Ten Years</title>
  <text>
    <abstract>In order to evolve...</abstract>
    <section number="1" title="Introduction">
      The <index>Web</index> provides the universal...
    </section>
  </text>
</article>
```

A Simple XML Document



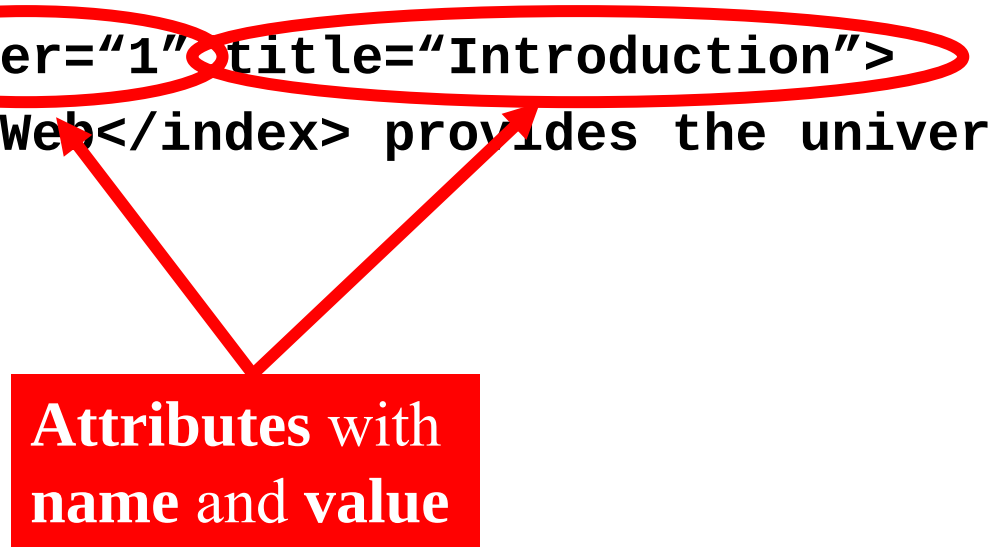

```
<article>
  <author>Gerhard Weikum</author>
  <title>The Web in Ten Years</title>
  <text>
    <abstract>In order to evolve...</abstract>
    <section number="1" title="Introduction">
      The <index>Web</index> provides the universal...
    </section>
  </text>
</article>
```

A Simple XML Document



A Simple XML Document

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



Elements in XML Documents

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

Elements vs. Attributes

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

What is the difference between elements and attributes?

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

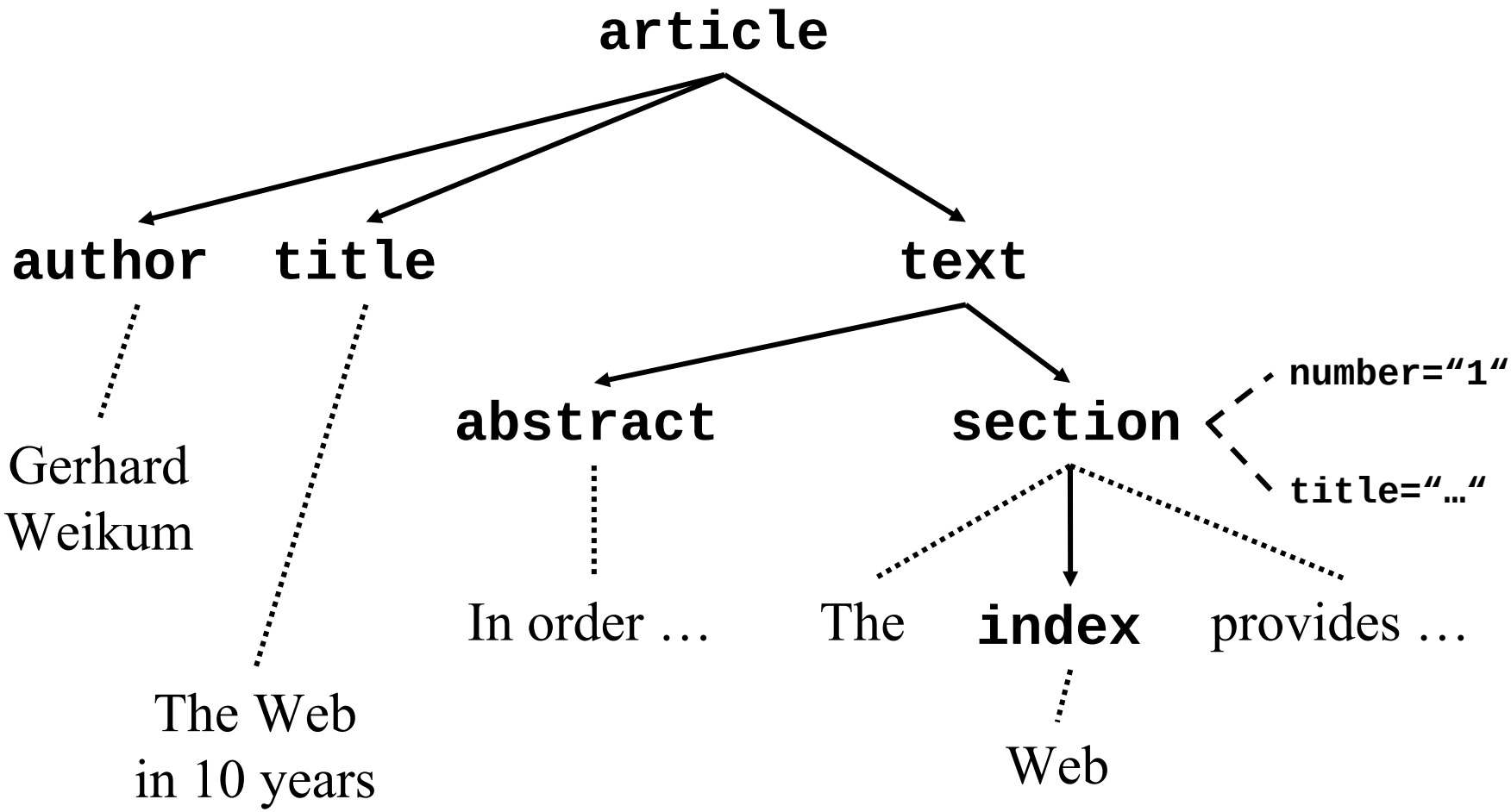
As a *rule of thumb*:

- Content into elements
- Metadata into attributes

Example:

```
<person born="1912-06-23" died="1954-06-07">
Alan Turing</person> proved that…
```

XML Documents as Ordered Trees



Preorder Traversal of the XML Tree

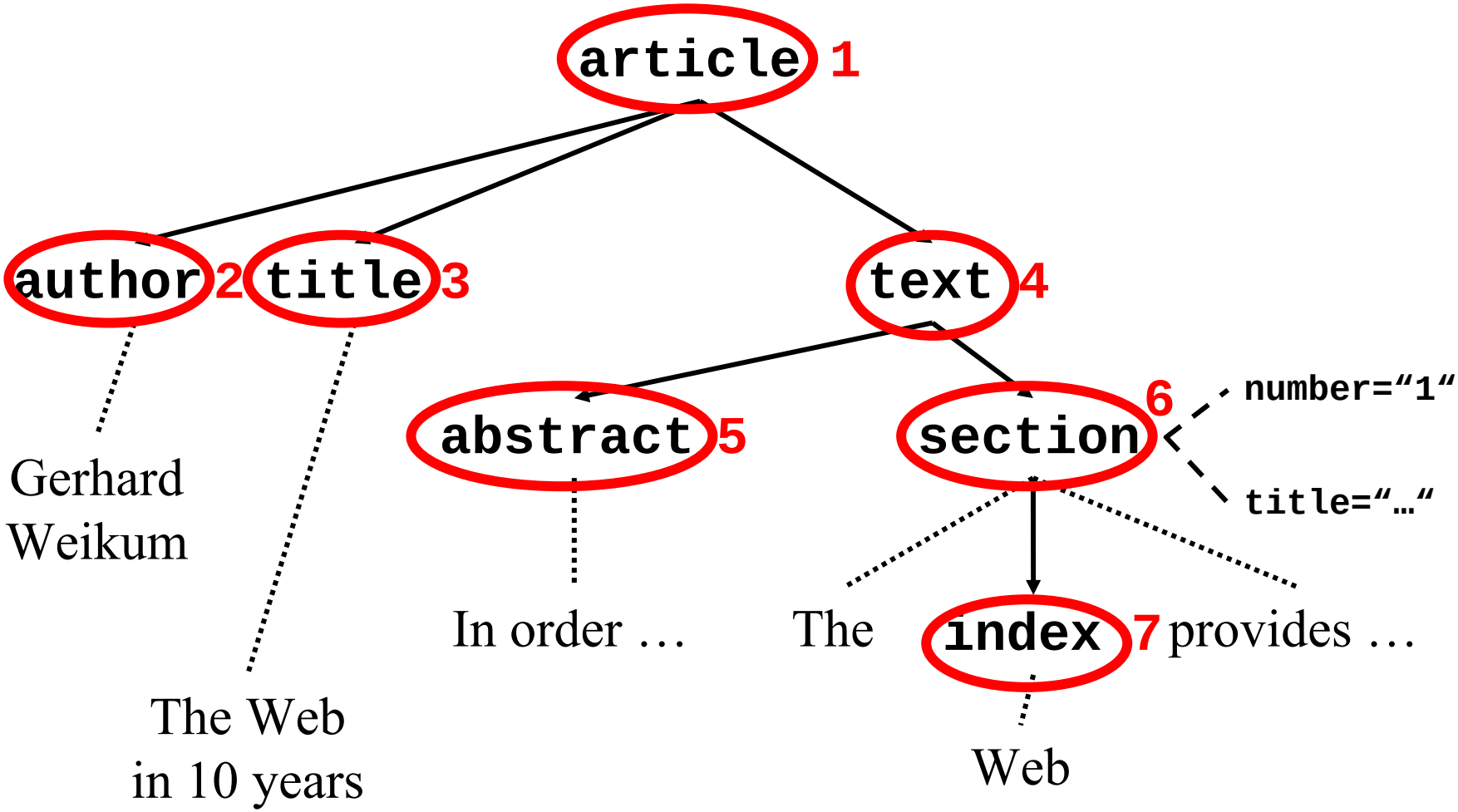
Goal: Find a *numbering* for the nodes in the XML Tree such that it can be used for linearizing the tree (i.e., create a document from it)

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

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

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

Preorder example



Well-Formed XML Documents

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

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

Well-Formed XML Documents

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

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

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

Documents vs. Data

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

17.4 Document Type Definitions

Sometimes XML is *too* flexible:

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

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

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

Most XML parsers can be configured to validate.

DTD Example: Elements

```
<!ELEMENT article    (title,author+,text)>
<!ELEMENT title      (#PCDATA)>
<!ELEMENT author     (#PCDATA)>
<!ELEMENT text       (abstract,section*,literature?)>
<!ELEMENT abstract   (#PCDATA)>
<!ELEMENT section    (#PCDATA|index)*>
<!ELEMENT literature (#PCDATA)>
<!ELEMENT index      (#PCDATA)>
```

Content of the `title` element is parsed character data

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

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

Element Declarations in DTDs

One element declaration for each element type:

`<!ELEMENT element_name content_specification>`

where **`content_specification`** can be

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

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

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

Plus arbitrary combinations using parenthesis:

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

More on Element Declarations

- Elements with mixed content:

```
<!ELEMENT text (#PCDATA|index|cite|glossary)*>
```

- Elements with empty content:

```
<!ELEMENT image EMPTY>
```

- Elements with arbitrary content (this is nothing for production-level DTDs):

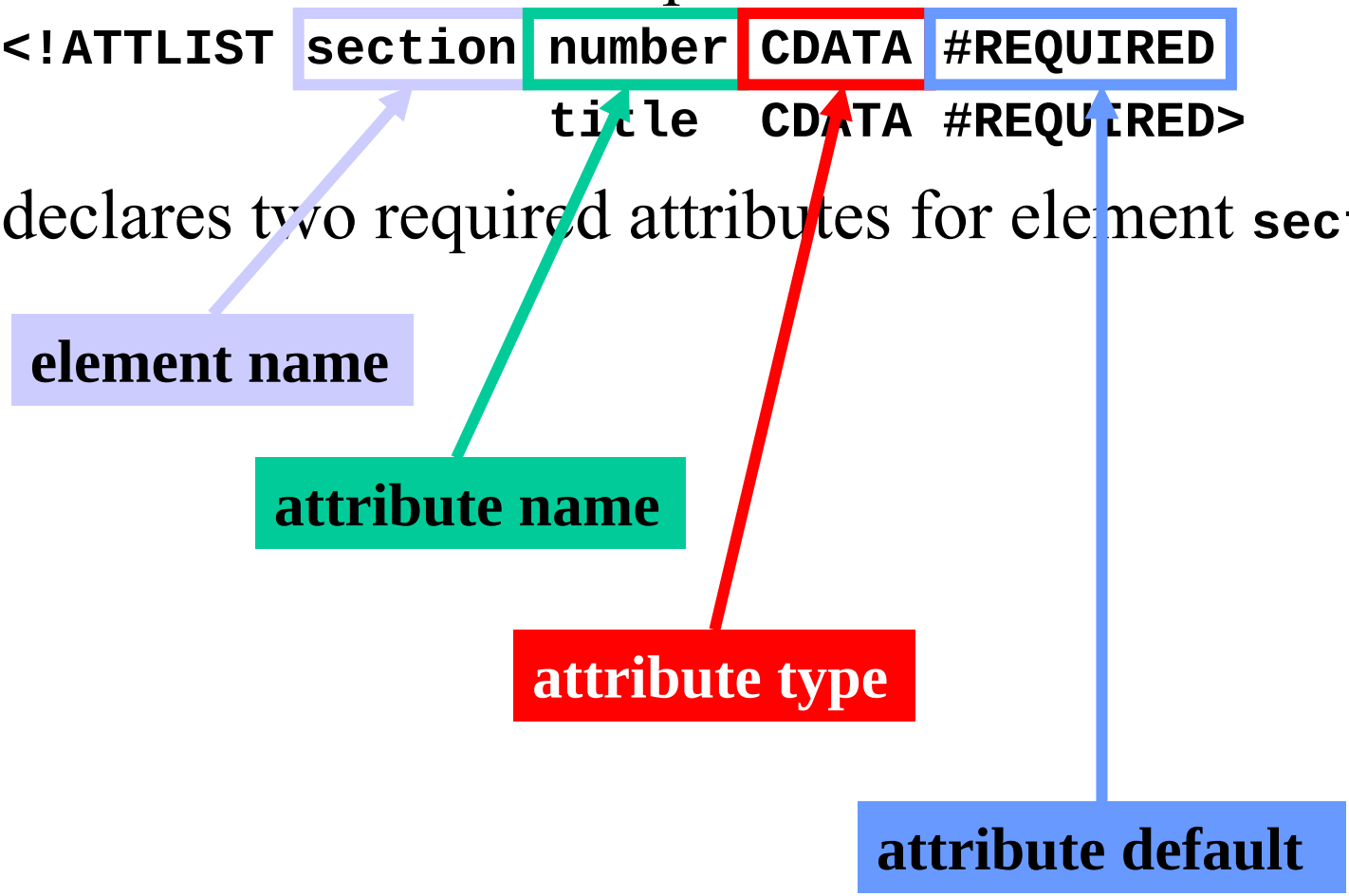
```
<!ELEMENT thesis ANY>
```

Attribute Declarations in DTDs

Attributes are declared per element:

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

declares two required attributes for element `section`.



Attribute Declarations in DTDs

Attributes are declared per element:

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

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

Possible attribute defaults:

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

Important Attribute Types in DTDs

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

Attribute Examples

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

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

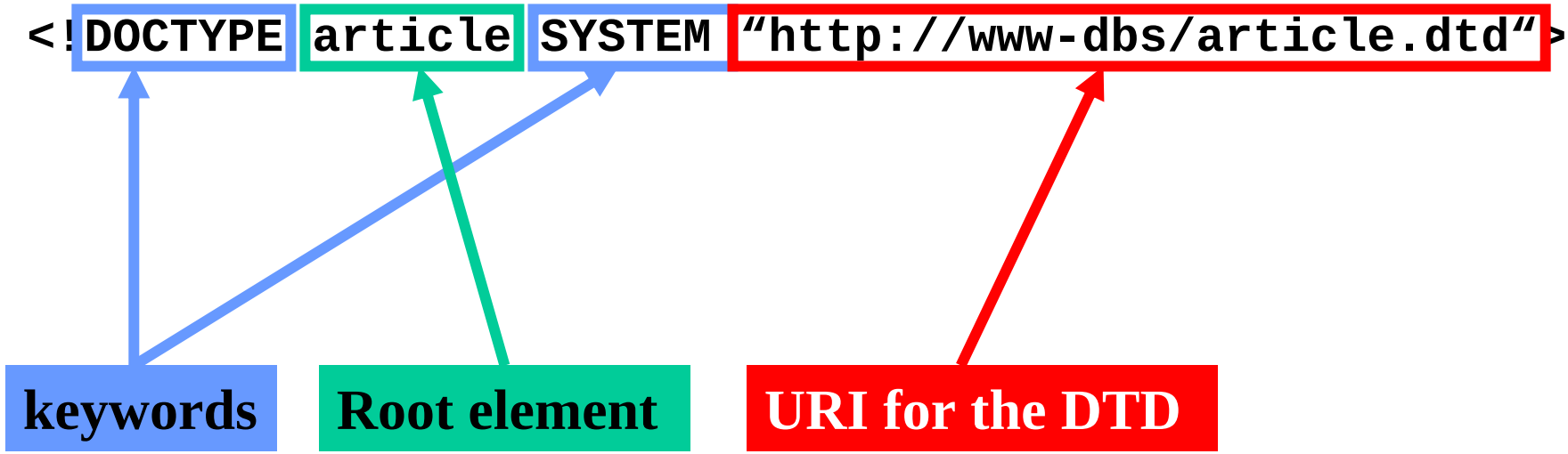
Attribute Examples

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


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

Linking DTD and XML Docs

- Document Type Declaration in the XML document:



Flaws of DTDs

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

$\Rightarrow$ XML Schema

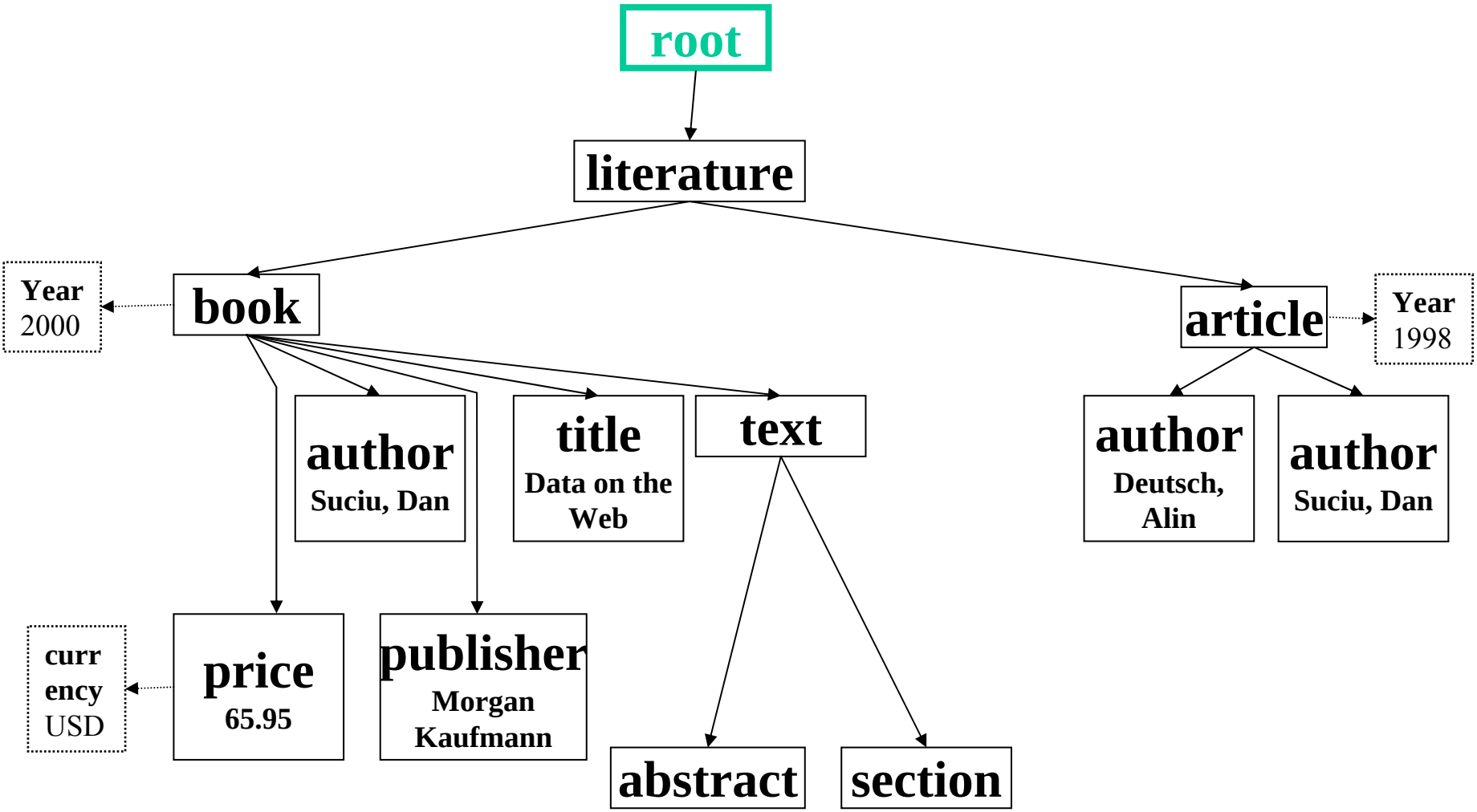
17.5 XPath

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

XML Example for XPath

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

Extended XML Tree Example



Elements of XPath

- A common XPath expression is a **location path** that consists of **location steps**, separated by /:

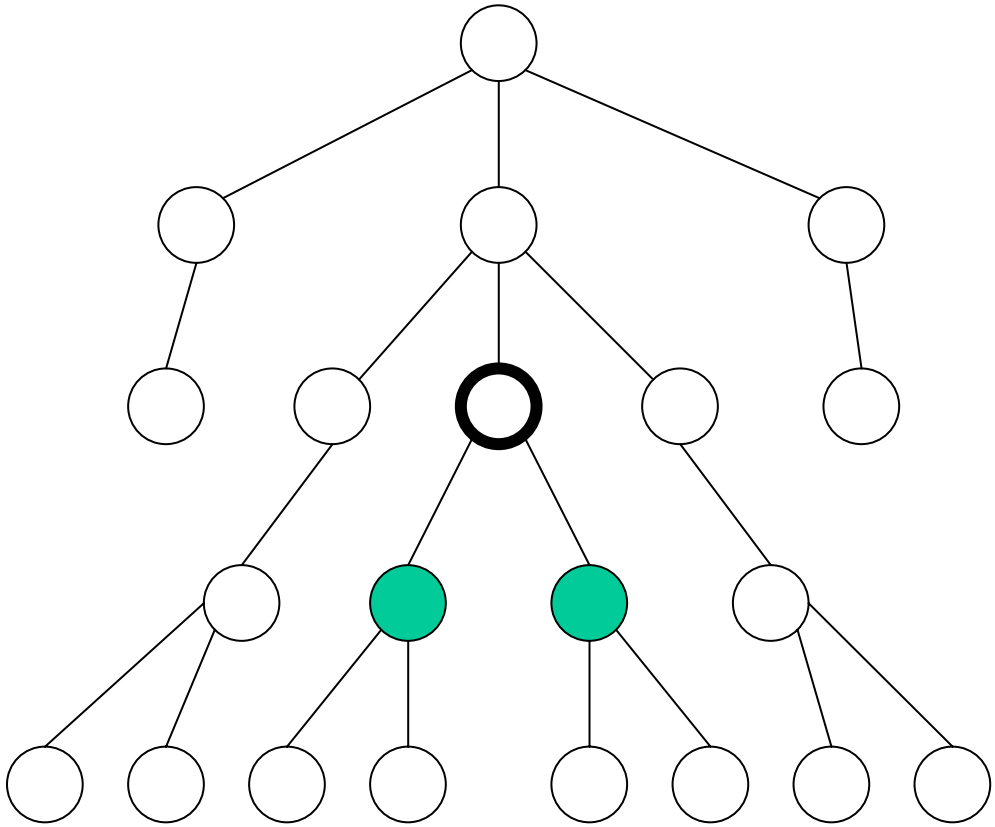
  `//article/text/abstract`: selects all **`abstract`** elements of articles

- A leading / always means the root element
- Each location step is evaluated in the context of a node in the tree, the so-called **context node** (which is among the results of the previous location step)
- The general form of a location step is ***`axis::test[predicate]`***
  - **`axis`** selects the node set for this step, starting from the context node
  - **`test`** is an additional restriction on the nodes
  - **`predicate`** is a filtering condition on the nodes

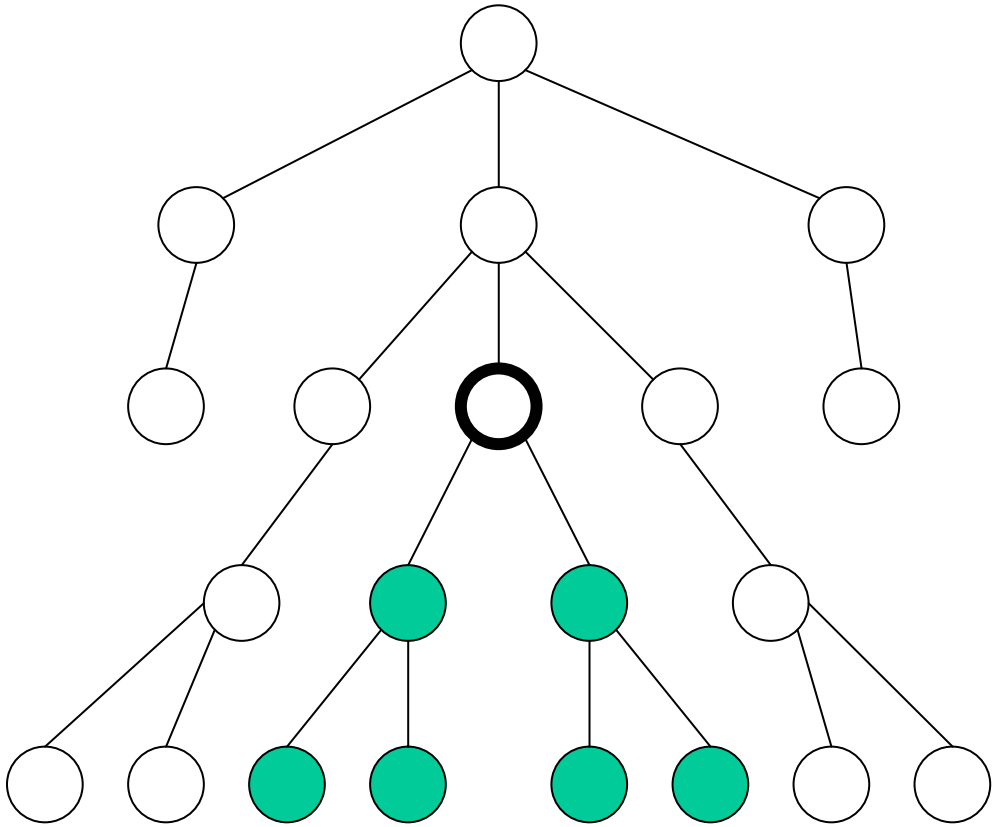
XPath Location Axes

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

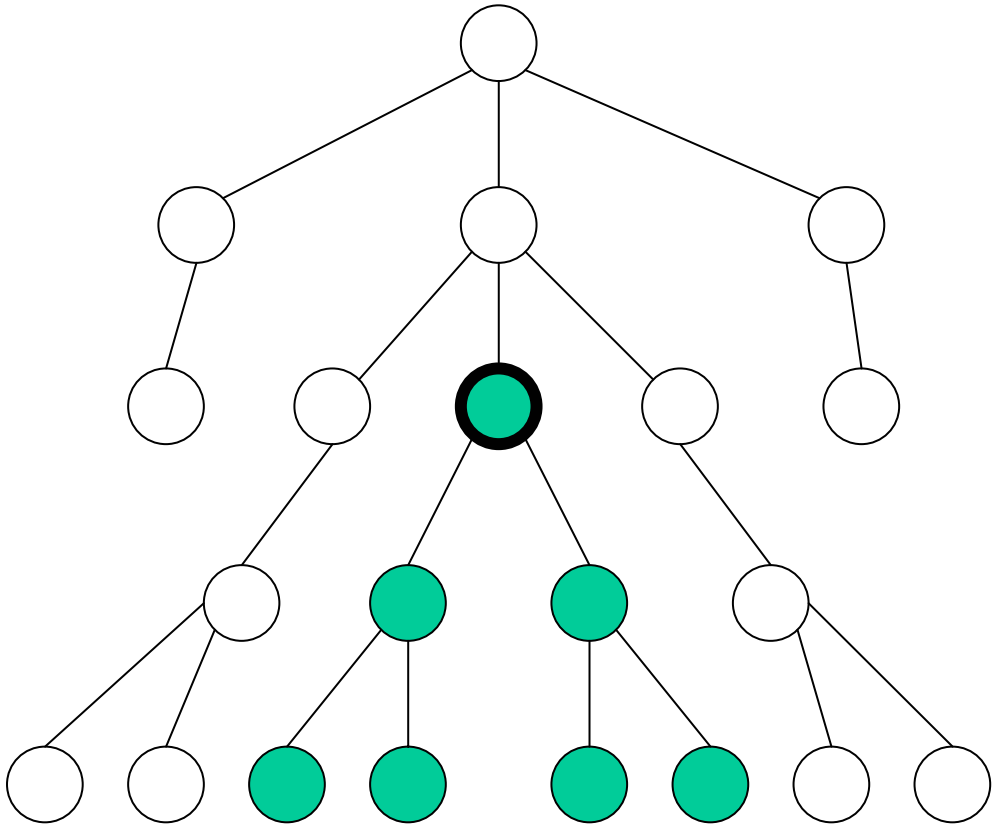
The `child` axis



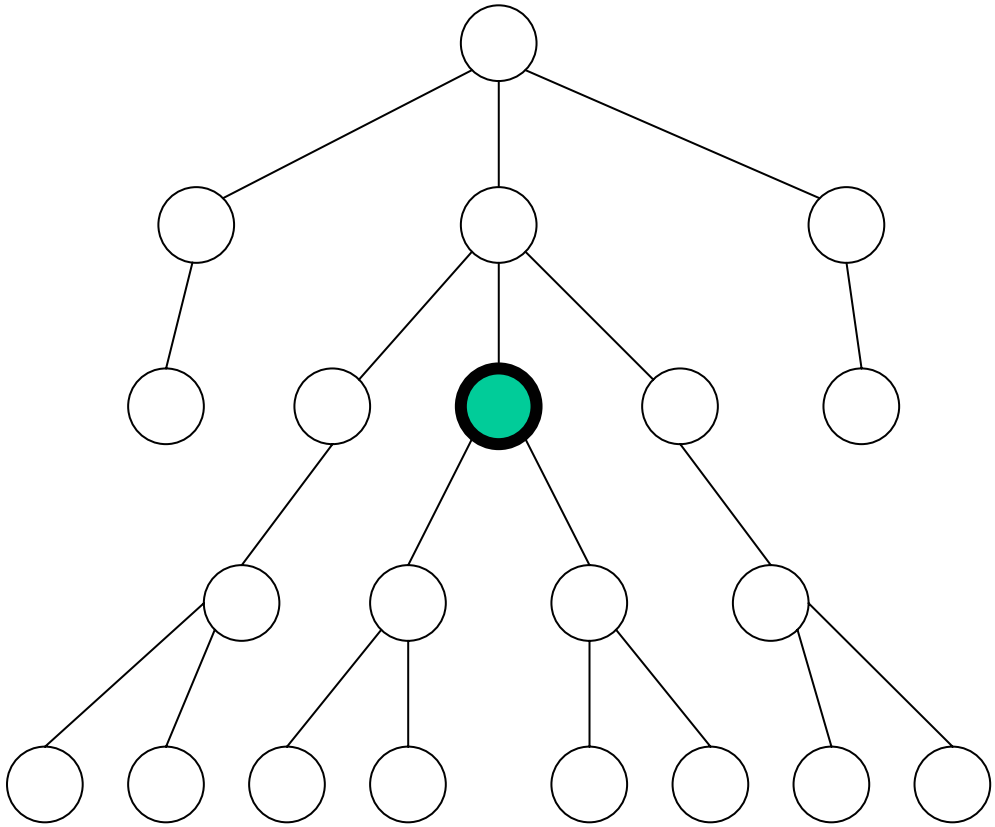
The descendant axis



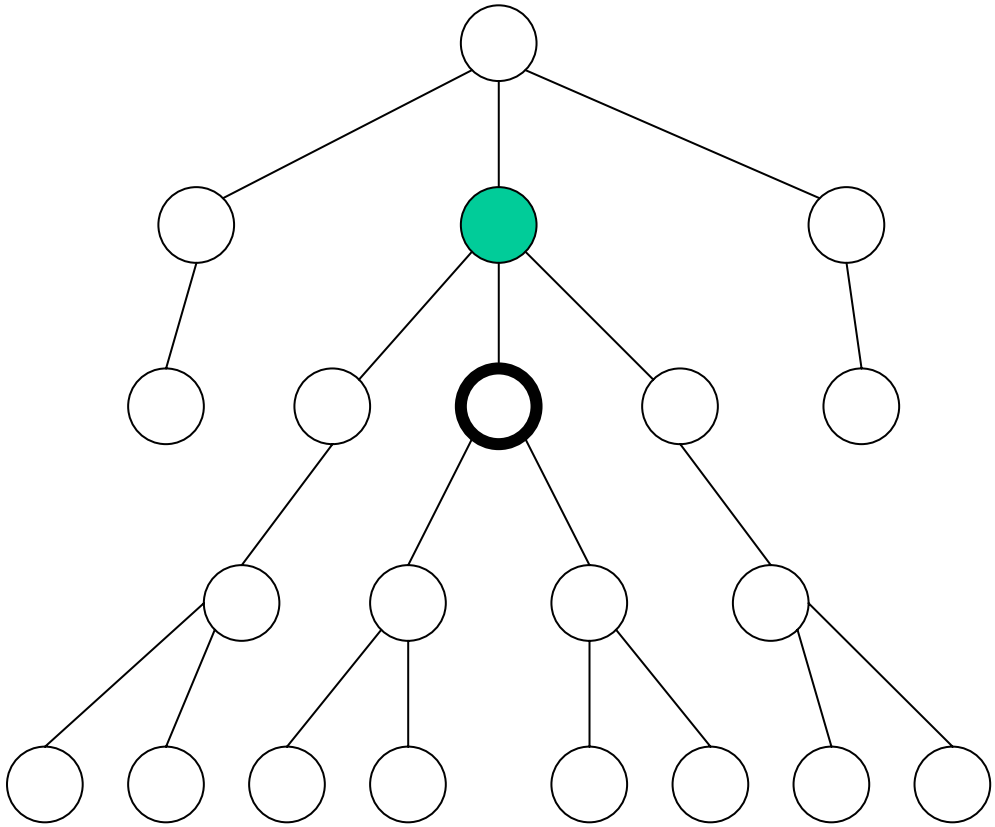
The descendant-or-self axis



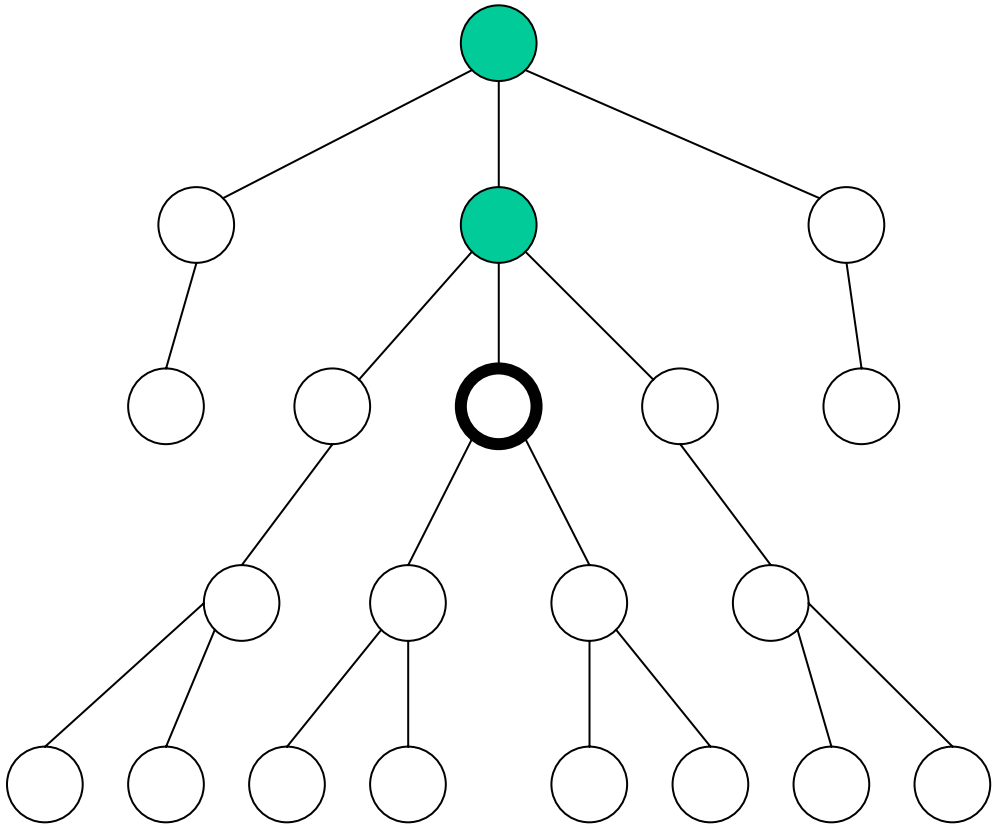
The `self` axis



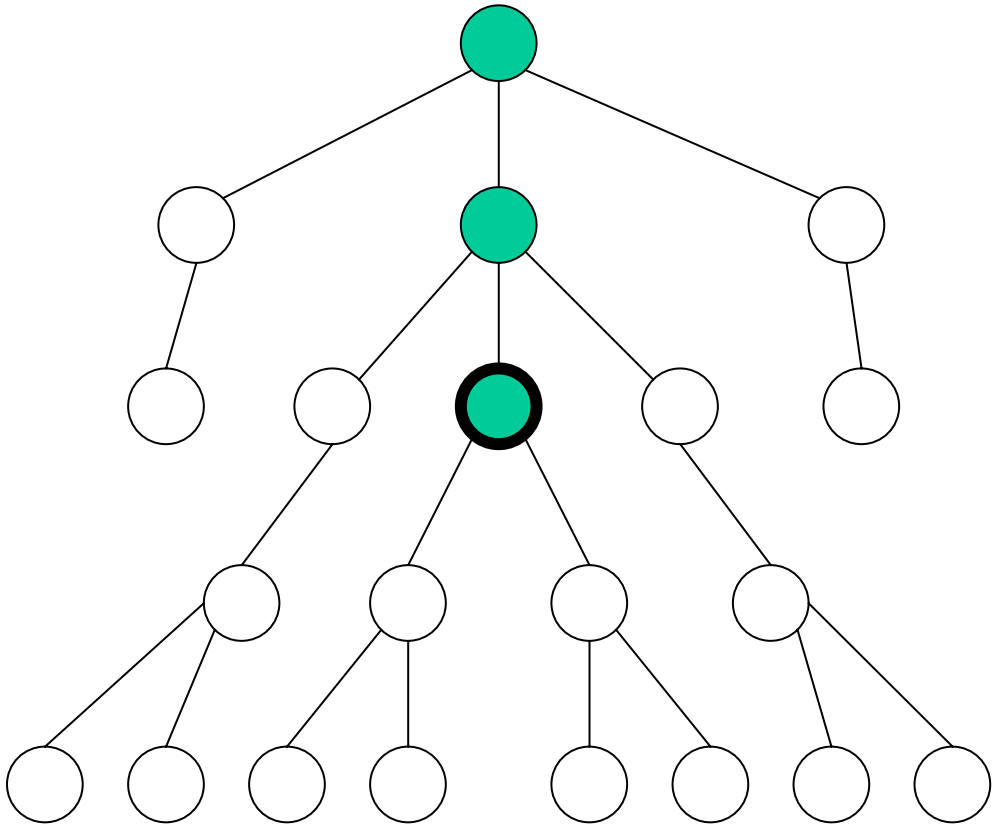
The `parent` axis



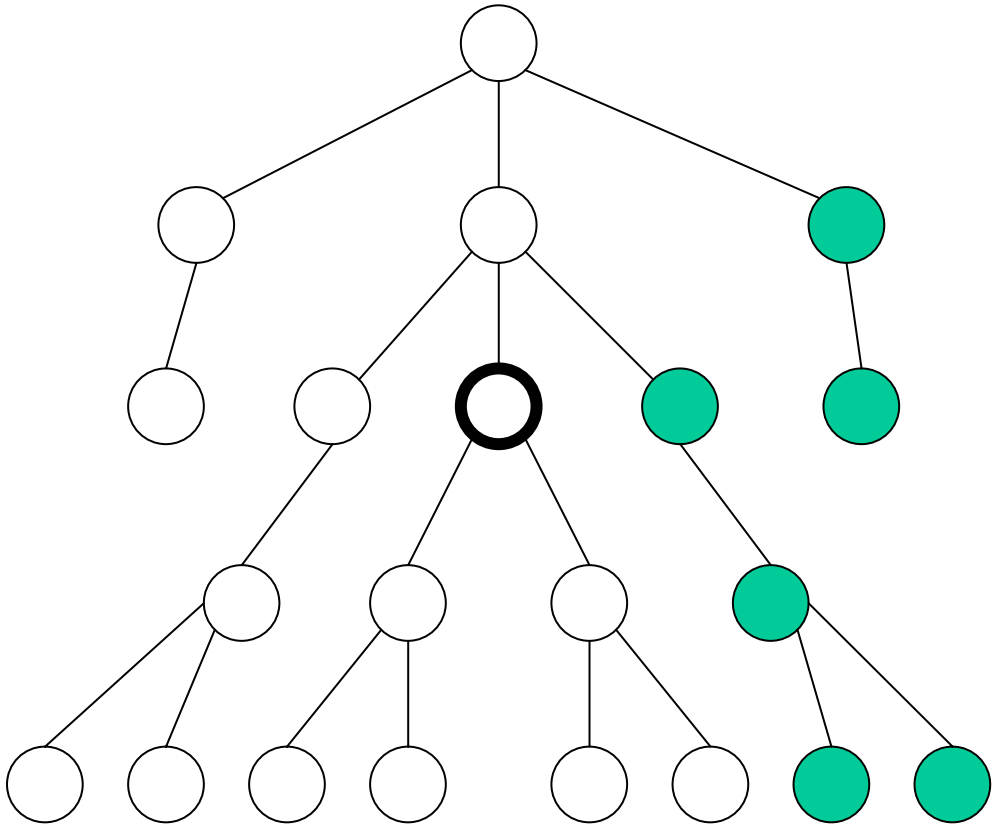
The ancestor axis



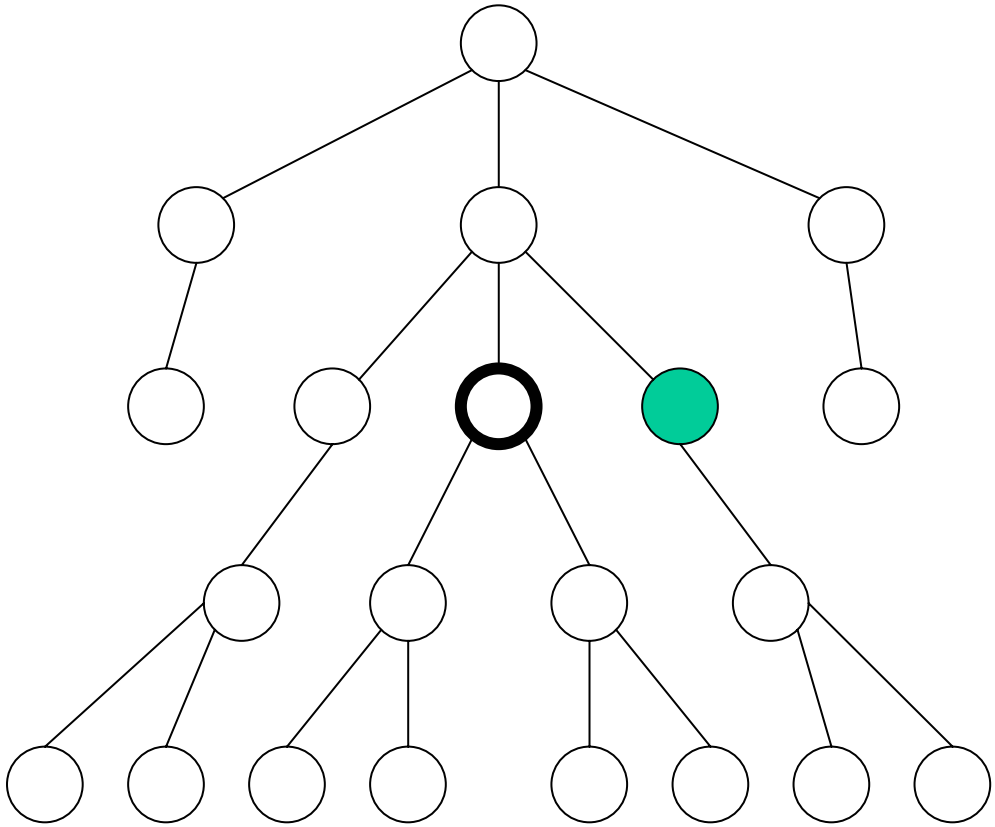
The `ancestor-or-self` axis



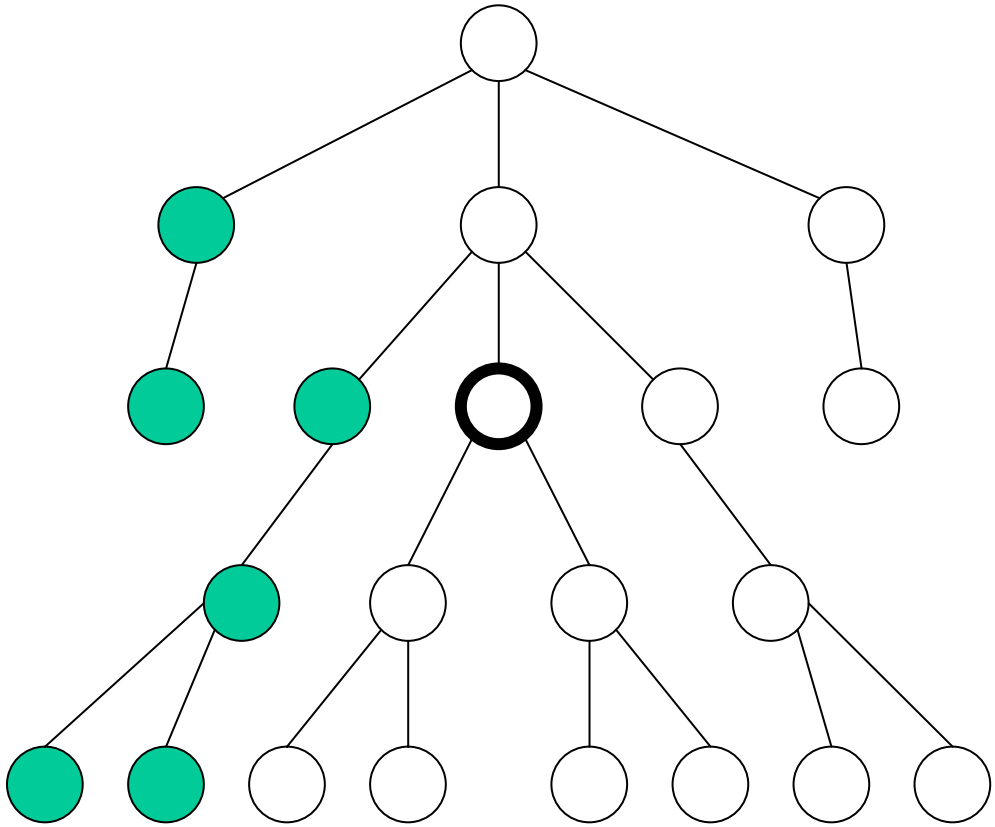
The following axis



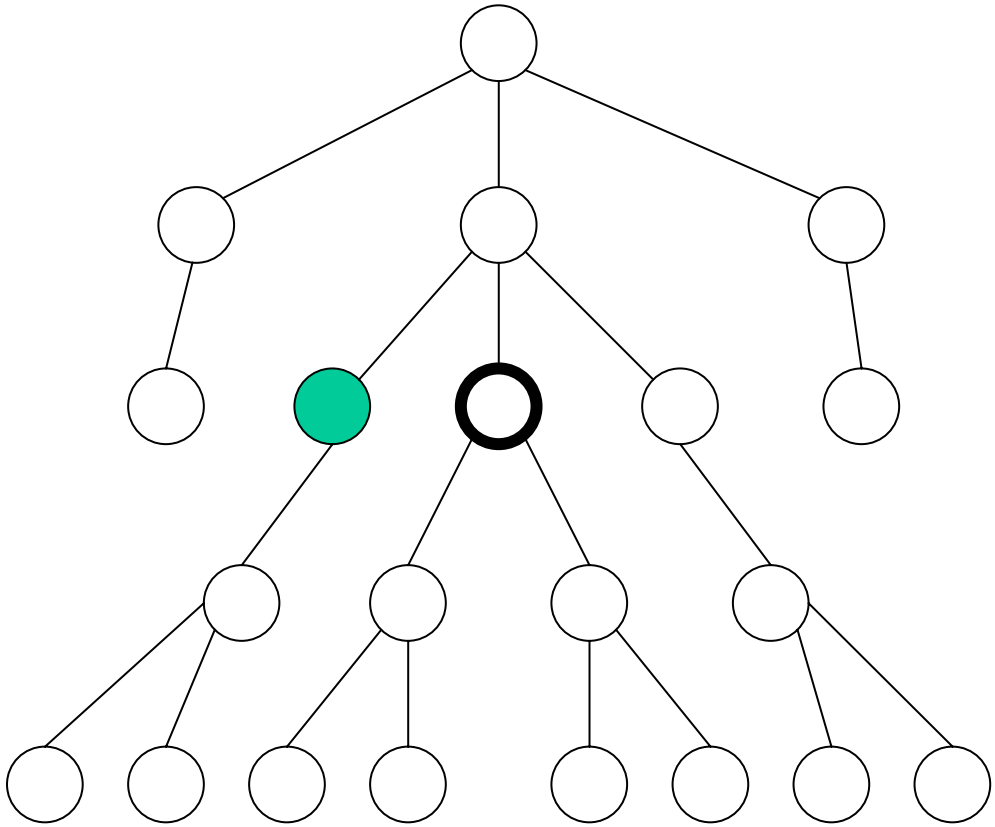
The following-sibling axis



The `preceding` axis



The `preceding-sibling` axis



Important XPath Location Tests

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

Simple XPath Examples

/literature/book/author — retrieves all book authors:
starting with the root, traverses the tree, matches element names literature, book, author, and returns elements
<author>Suciu, Dan</author>,
<author>Abiteboul, Serge</author>, ...,
<author><firstname>Jeff</firstname>
<lastname>Ullman</lastname></author>

/literature/*/author — authors of books, articles, essays, etc.

/literature//author — authors that are descendants of literature

/literature//@year — value of the year attribute of descendants of literature

Predicates in Location Steps

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

Some XPath Functions

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

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

Ex.: all second authors of articles

```
//article/author[2]
```

XPath Examples with Predicates

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

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

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

Sophisticated XPath Examples

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

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

Coauthors of Jeff Ullman:

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

Second author of books with Jeff Ullman being the first author

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

Design Rules for Efficient XPath

- Be as specific as possible: Use any information about document structure that is available

  not **`//article//author[2]`**

  but **`/literature/article/author[2]`**

  Reduces complexity for evaluating the query

- Filter as early as possible: Try to decrease the number of intermediate results in early steps of the query

  not **`//article[@year=2000][author="Schenkel"]`**

  but **`//article[author="Schenkel"][@year=2000]`**

  Benefit depends on *selectivity* of predicates, rule of thumb: selective predicates to the front for increasing performance

  (in the ex.: many articles from 2000, but only a few articles in the literature database authored by Schenkel)

Sources and Further Literature

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