Nagarjuna College of Engineering and Technology

Information Science and Engineering

Module 5

XML: Extensible Markup Language

Lecture 33
Introduction, Syntax

What is XML

- XML stands for eXtensible Markup Language.
- A markup language is used to provide information about a document.
- Tags are added to the document to provide the extra information.
- HTML tags tell a browser how to display the document.
- XML tags give a reader some idea what some of the data means.

What is XML?

- a meta language that allows you to create and format your own document markups
- a method for putting structured data into a text file; these files are
  - easy to read
  - unambiguous
  - extensible
  - platform-independent
- XML documents are used to transfer data from one place to another often over the Internet.

Difference Between HTML and XML

- HTML tags have a fixed meaning and browsers know what it is.
- XML tags are different for different applications, and users know what they mean.
- HTML tags are used for display.
- XML tags are used to describe documents and data.

Quick Comparison

- HTML

- tags and attributes are pre-determined and rigid
- content and formatting can be placed together

  <p><font=”Arial”>text
- Designed to represent the presentation structure of a document. Thus, more effective for machine-human interaction

- XML

- allows user to specify what each tag and attribute means
  - content and format are separate; formatting is contained in a stylesheet
  - Designed to represent the logical structure of a document. Thus, more effective for machine-machine interaction
- Syntax is strictly defined

Advantages of XML

- XML is text (Unicode) based.
  - Takes up less space.
  - Can be transmitted efficiently.
- One XML document can be displayed differently in different media.
  - Html, video, CD, DVD,
  - You only have to change the XML document in order to change all the rest.
- XML documents can be modularized. Parts can be reused.

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

Example of an HTML Document

```
<html>
   <head><title>Example</title></head.
<body>
   <h1>This is an example of a page.</h1>
   <h2>Some information goes here.</h2>
</body>
</html>
```

Lecture 35
Namespaces

Example of an XML Document

```
<?xml version="1.0"/>
<address>
   <name>Alice Lee</name>
   <email>alee@aol.com</email>
   <phone>212-346-1234</phone>
   <birthday>1995-03-22</birthday>
</address>
```

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

Lecture 36
XML Schemas

App. Scenario 1: Content Mgt.

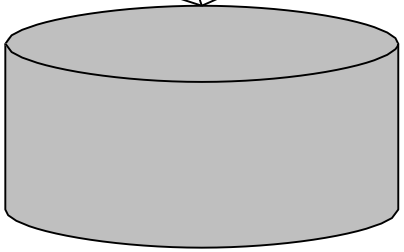
Clients

XML2HTML

XML2WML

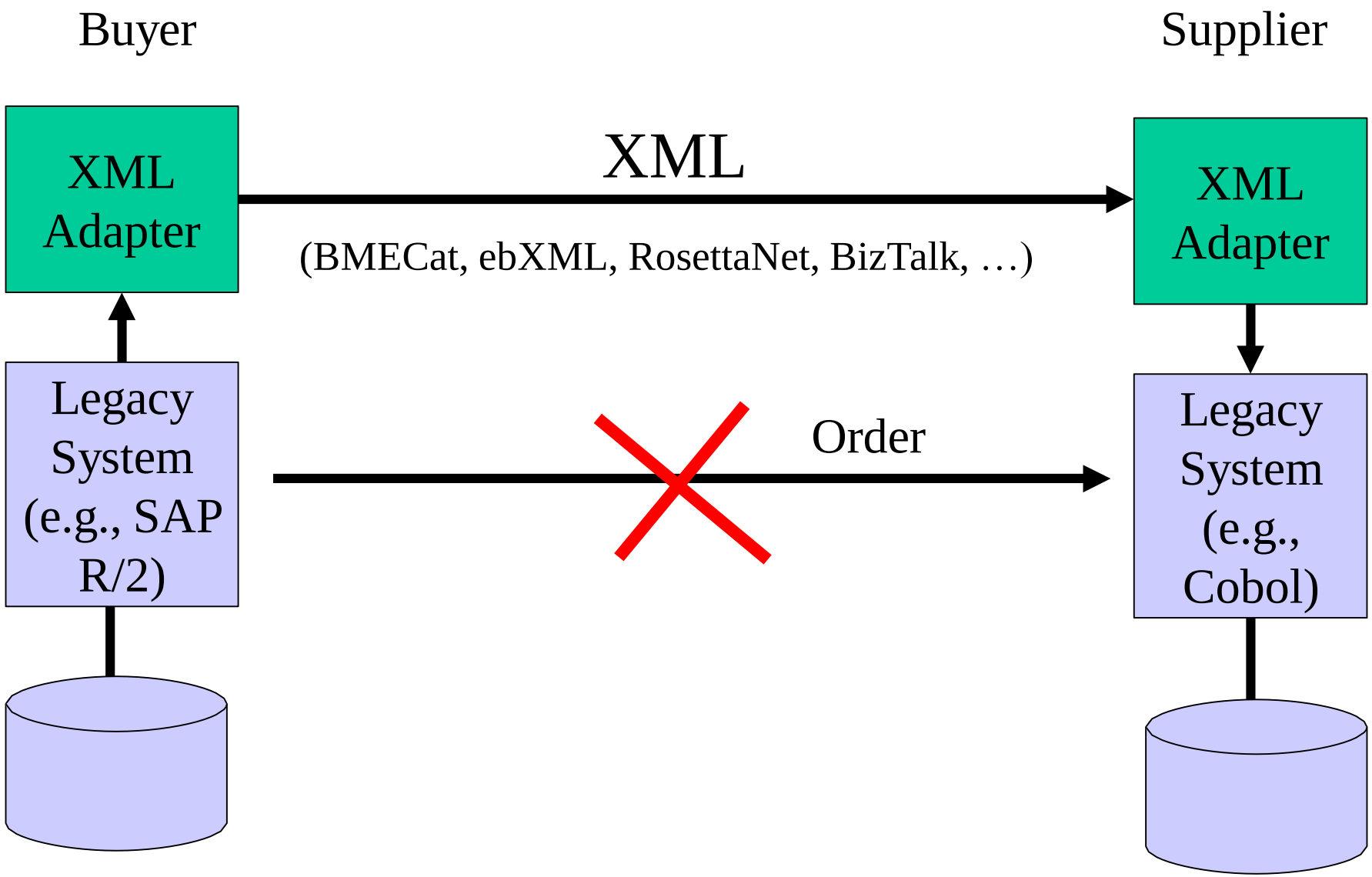
XML2PDF

Converters

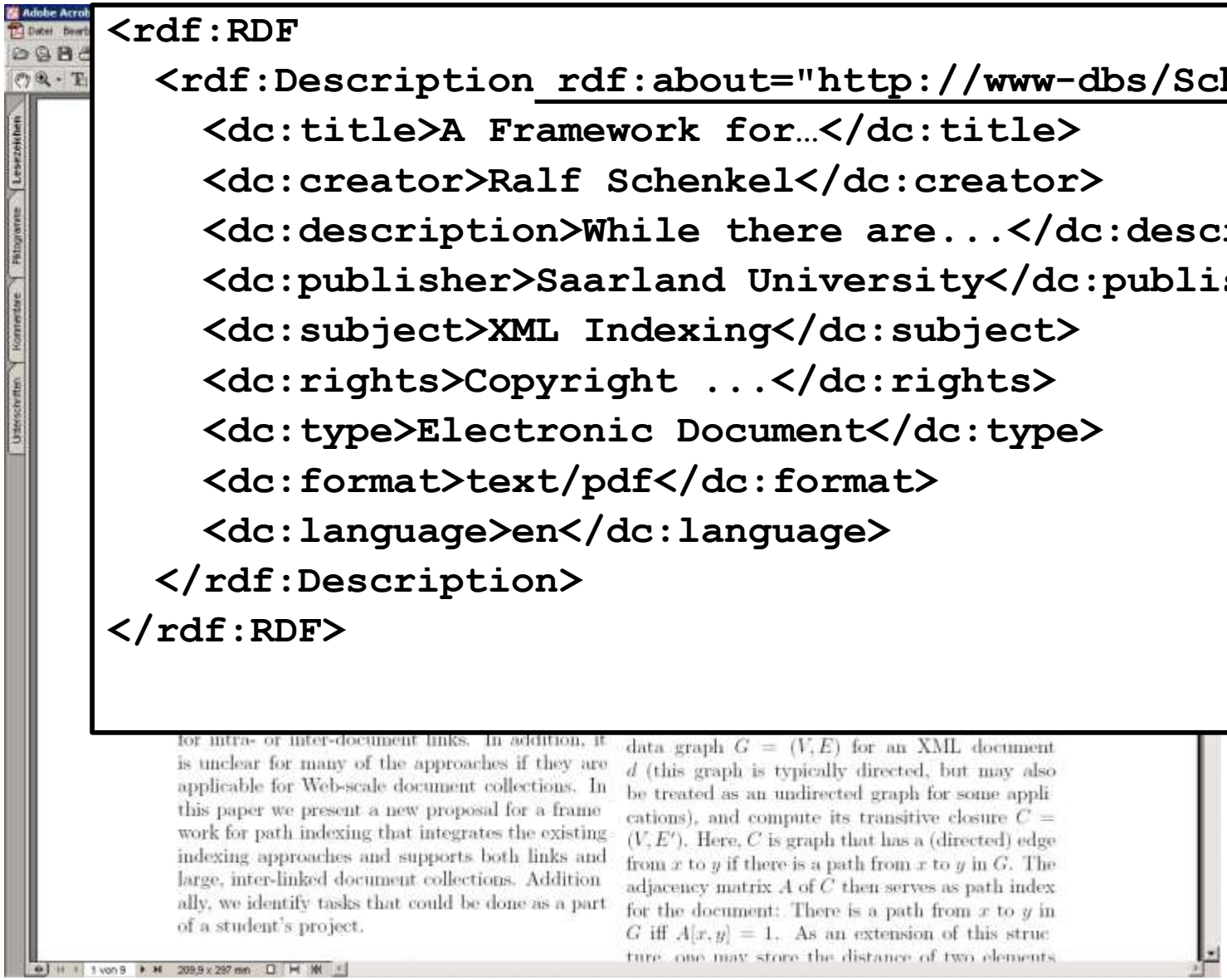


Database with
XML documents

App. Scenario 2: Data Exchange



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“?

Lecture 37 and 38
Displaying raw XML documents

XML Documents

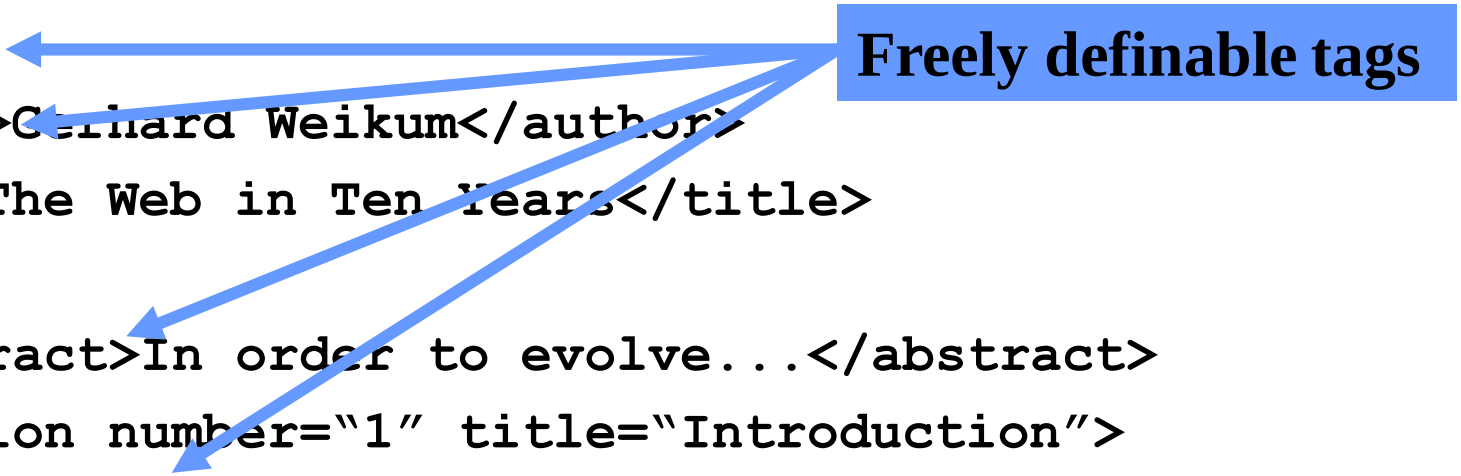
What‘s in an XML document?

- Elements
- Attributes
- plus some other details

A Simple XML Document

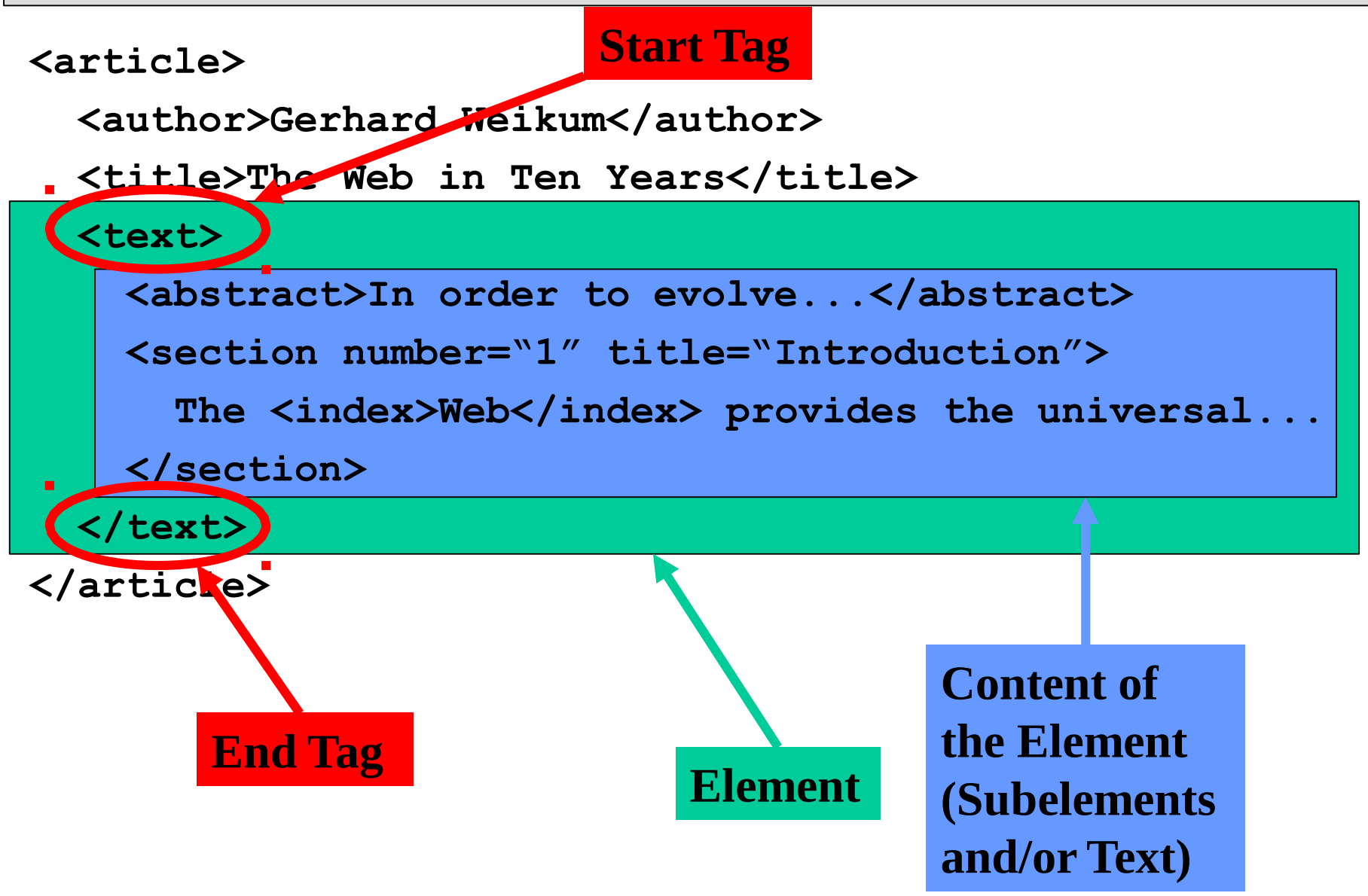
```
<article>
  <author>Gerhard Weikum</author>
  <title>The Web in Ten Years</title>
  <text>
    <abstract>In order to evolve...</abstract>
    <section number="1" title="Introduction">
      The <index>Web</index> provides the universal...
    </section>
  </text>
</article>
```

A Simple XML Document



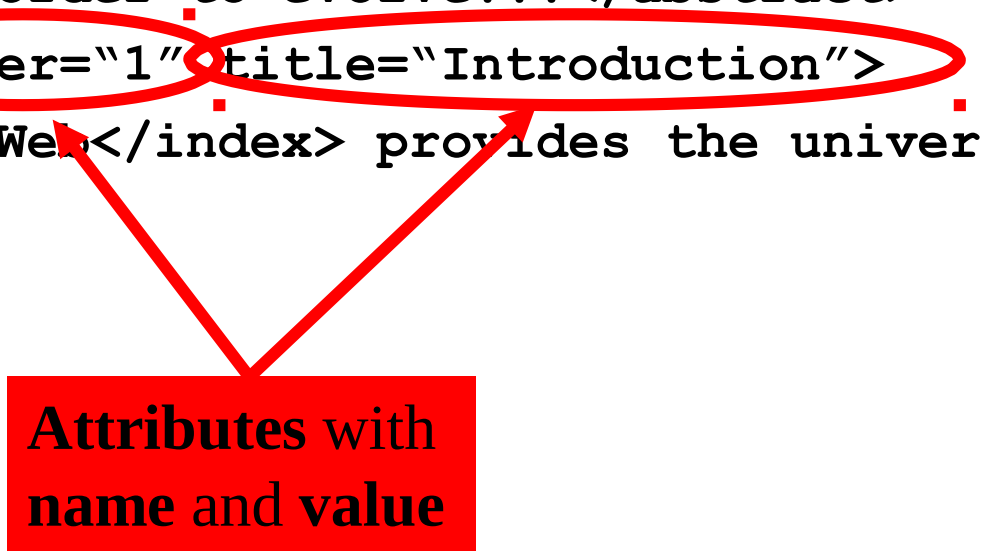

```
<article>
  <author>Gerhard Weikum</author>
  <title>The Web in Ten Years</title>
  <text>
    <abstract>In order to evolve...</abstract>
    <section number="1" title="Introduction">
      The <index>Web</index> provides the universal...
    </section>
  </text>
</article>
```

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)

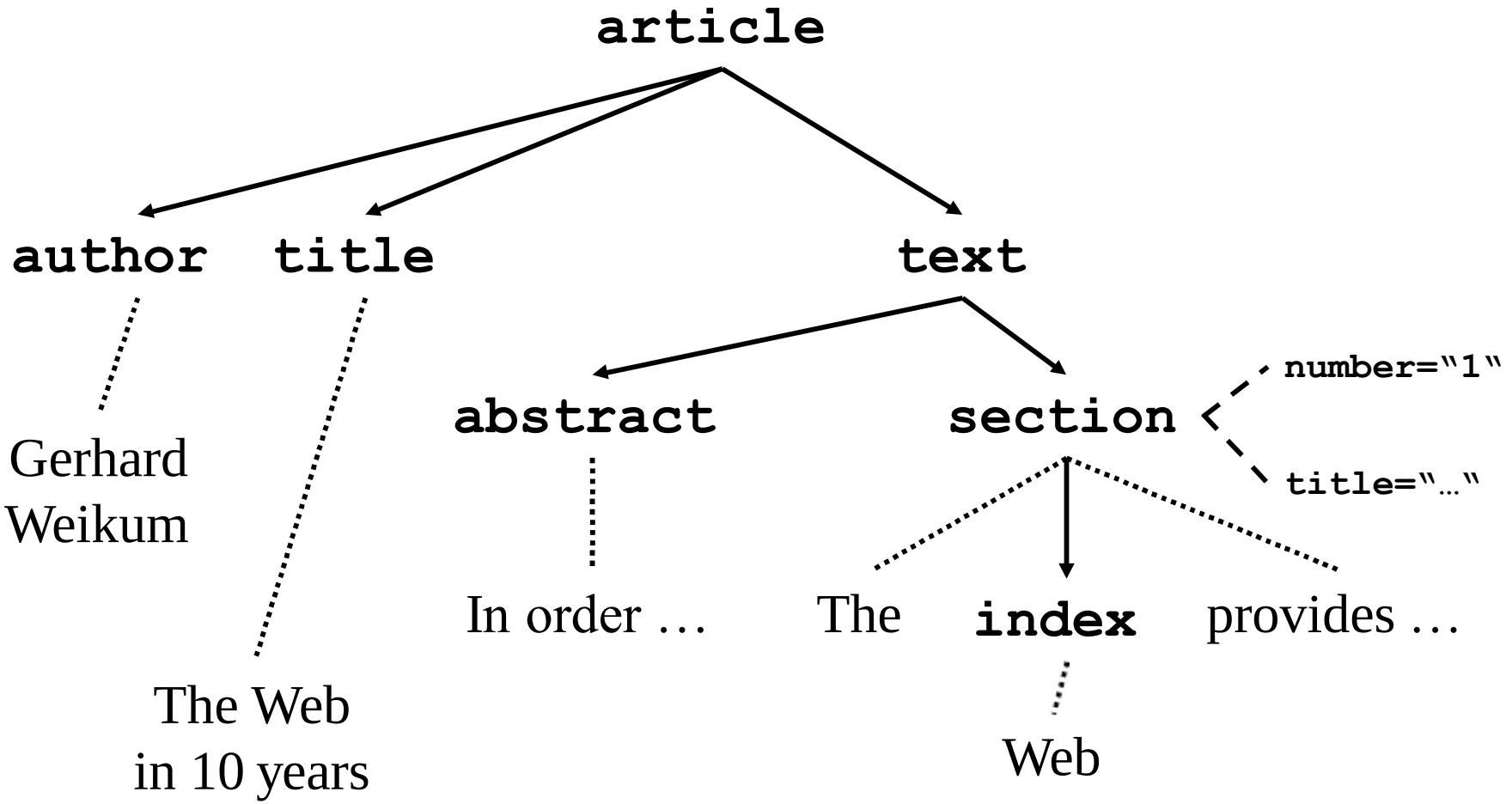
As a *rule of thumb*:

- 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



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.

Lecture 34
Document type definitions

Document Type Definitions

Sometimes XML is *too* flexible:

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

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

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

Most XML parsers can be configured to validate.

DTD Example: Elements

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

Content of the `title` element is parsed character data

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

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

Element Declarations in DTDs

One element declaration for each element type:

`<!ELEMENT element_name content_specification>`

where **`content_specification`** can be

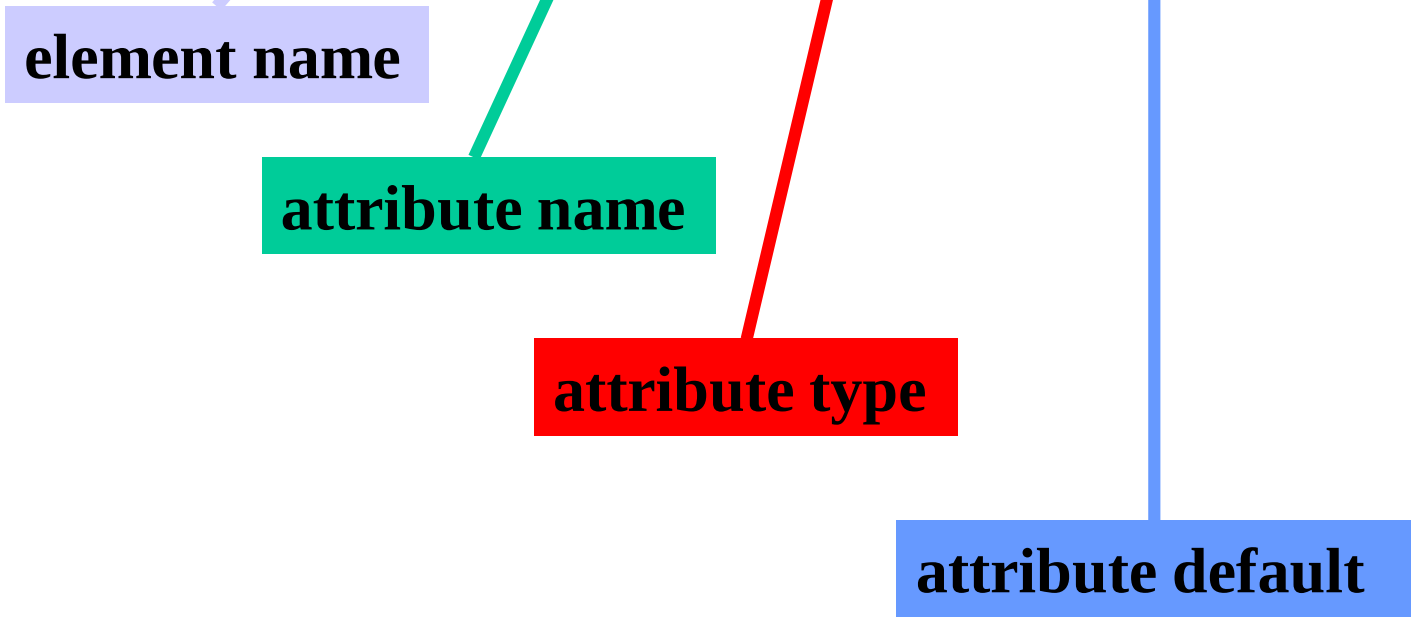
- **`(#PCDATA)`** parsed character data
- **`(child)`** one child element
- **`(c1,…,cn)`** a sequence of child elements c1…cn
- **`(c1|…|cn)`** one of the elements c1…cn

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

Attribute Types in DTDs

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

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>
```

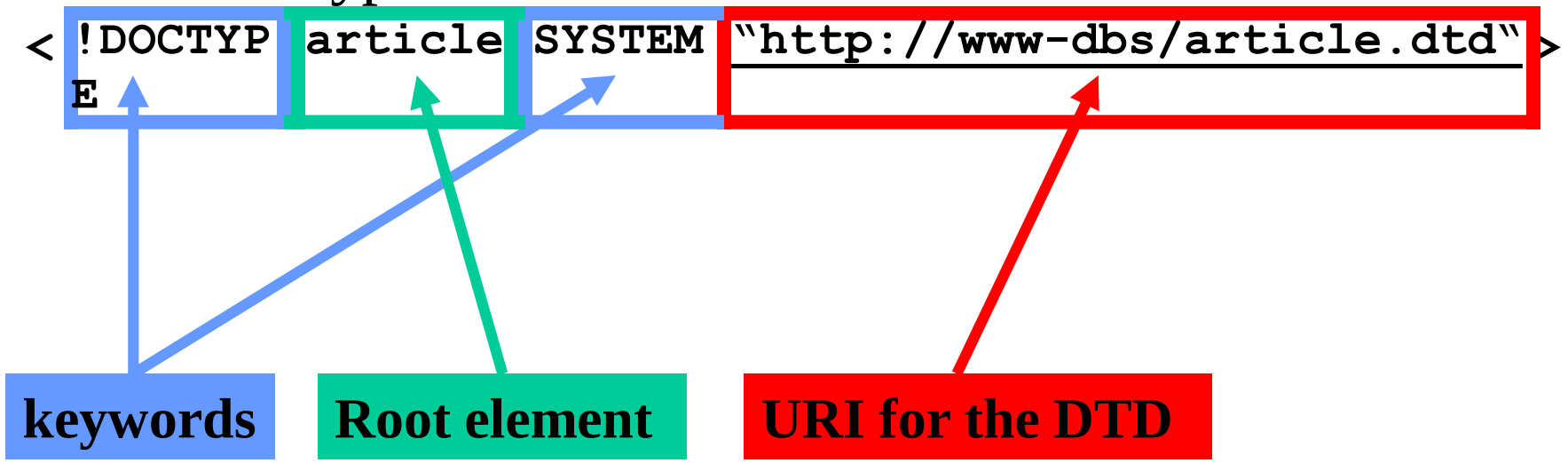
Attribute Examples

```
<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:



Linking DTD and XML Docs

- Internal DTD:

```
<?xml version="1.0"?>
<!DOCTYPE article [
  <!ELEMENT article (title,author+,text)>
  ...
  <!ELEMENT index (#PCDATA)>
]>
<article>
...
</article>
```

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

Lecture 39 and 40
Document type definitions

Displaying XML Documents

- Different approaches-
  - Extensible Style Language (XSL)
  - Document Style & Semantics Language (DSSL)
  - Cascading Style Sheets (CSS)

  CSS were originally designed for displaying HTML but XML browser can also use this.

  Unlike CSS, which utilizes the formatting and style instructions to directly display the XML document, XSL normally first transforms he XML document into a suitable format for display, such as HTML or RTF (Rich Text Format).

  It the sends this to an XSL processor(such as a browser or application), which then generates the required HTML or RTF output.

  Thus XSL style sheet has 2 parts:- Transforming, and Formatting.

Applications of XML

- Database applications
- Document Mark-up(with HTML)
- Mathematical Mark-up language(MATHML)
- Messaging b/w different business platforms
- Channel definition Format (CDF)
- Metacontent definition
- Platform for Internet Context Selection (PICS)
- Platform for Privacy References Syntax Specification (P3P)
- Resource Description Format (RDF)
- Scaleable Vector Graphics (SVG)
- Synchronized Multemedia Integration Language (SMIL)