XML

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)

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

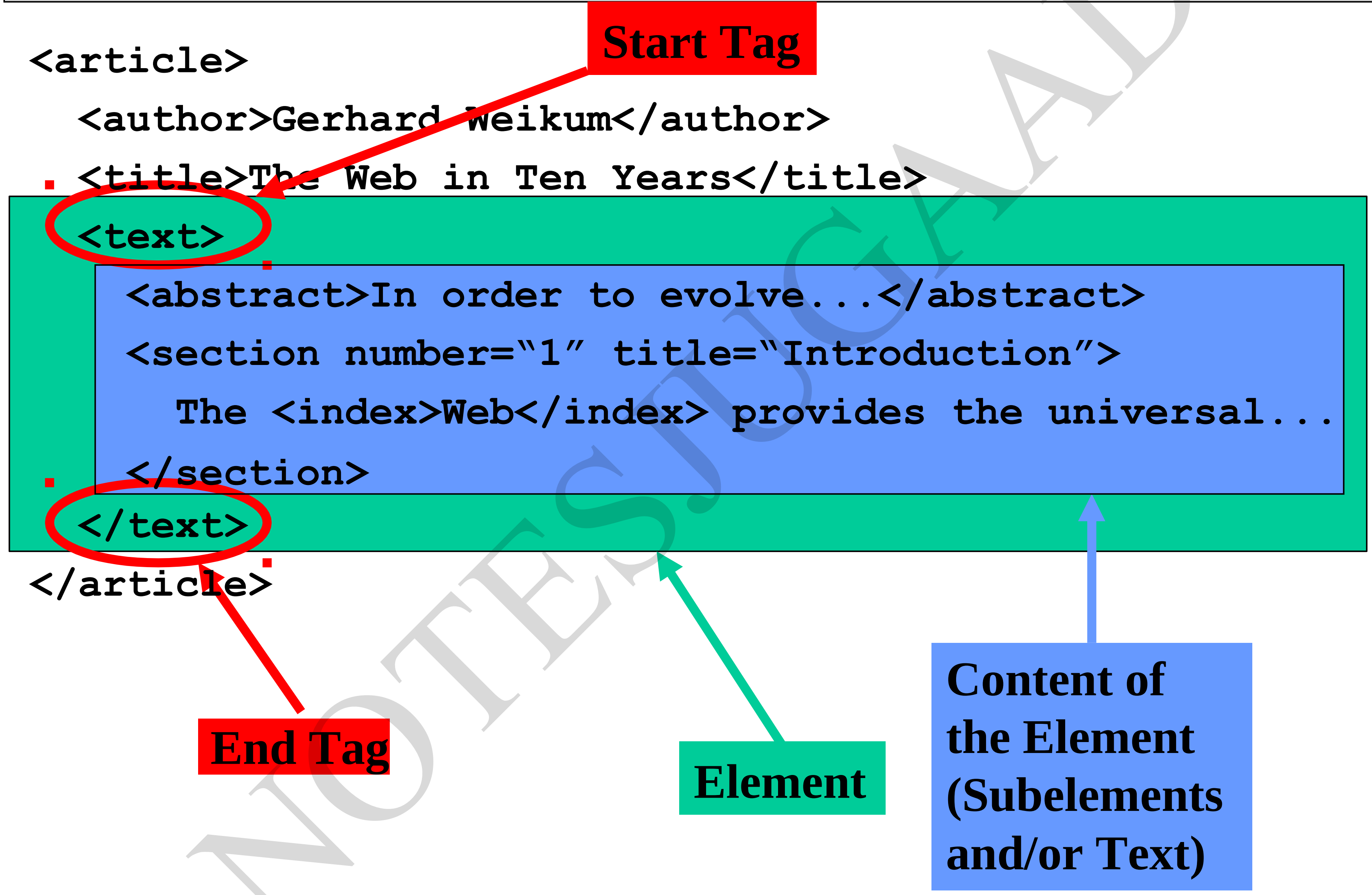
- Truly Portable Data
- Easily readable by human users
- Very expressive (semantics near data)
- Very flexible and customizable (no finite tag set)
- Easy to use from programs (libs available)
- Easy to convert into other representations (XML transformation languages)
- Many additional standards and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

A Simple XML Document

Freely definable tags

```
<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

```
<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>
```

Attributes with **name** and **value**

Elements in XML Documents

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

Elements vs. Attributes

Elements may have **attributes** (in the start tag) that have a **name** and

a **value**, e.g. **`<section number=“1“>`**.

What is the difference between elements and attributes?

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

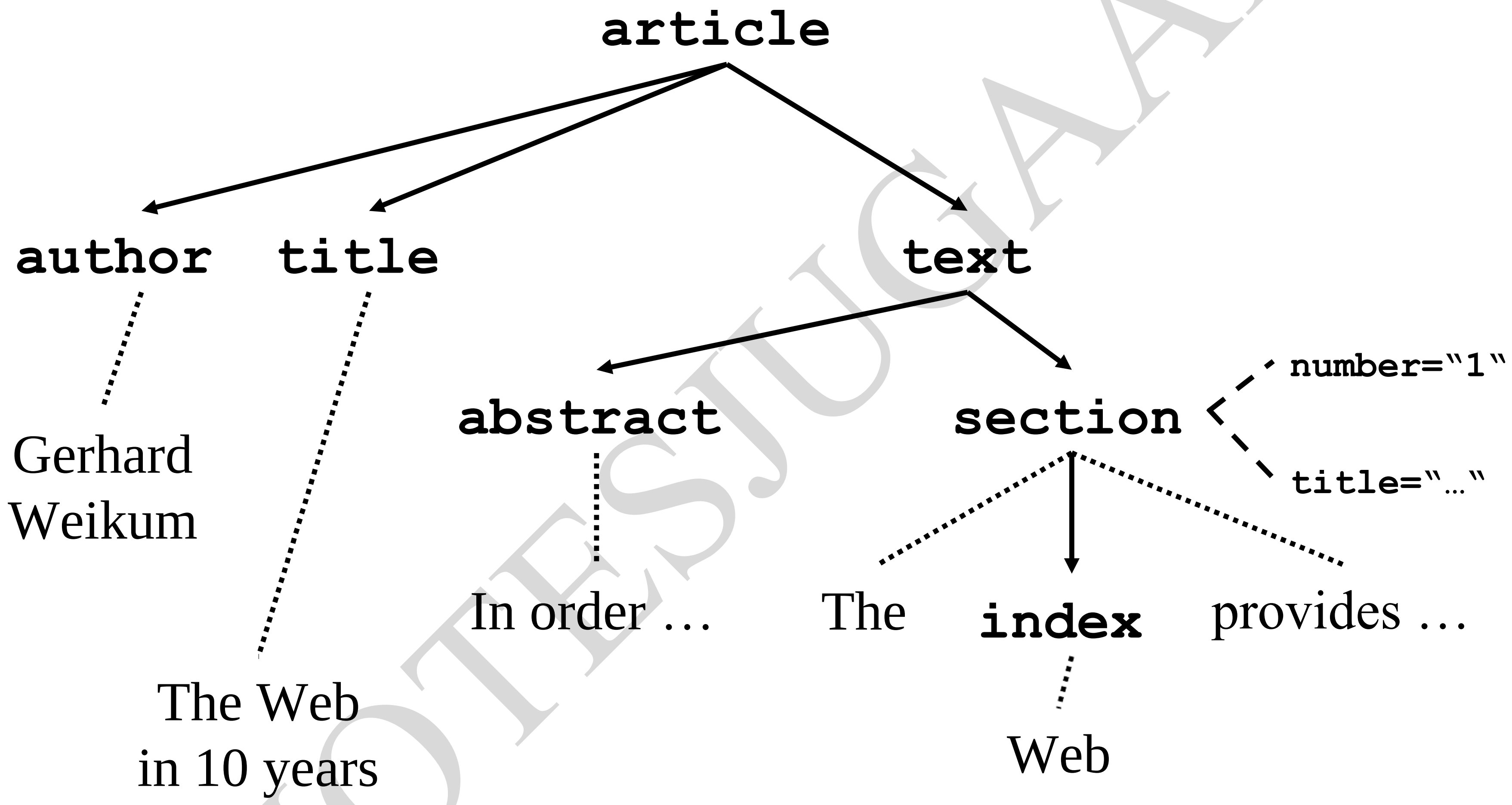
As a *rule of thumb*:

- Content into elements
- Metadata into attributes

Example:

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

XML Documents as Ordered Trees



More on XML Syntax

- <root> <child>   <subchild>.....</subchild> </child></root>
- <?xml version="1.0" encoding="UTF-8"?>
- XML Tags are Case Sensitive
- XML Elements Must be Properly Nested
- XML Attribute Values Must be Quoted
- <!-- This is a comment -->
- Some special characters must be escaped using **entities:**

  `<` → **`&lt;`**

  `&` → **`&amp;`**

  (will be converted back when reading the XML doc)
- Some other characters may be escaped, too:

  `>` → **`&gt;`**

  `“` → **`&quot;`**

  `‘` → **`&apos;`**

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

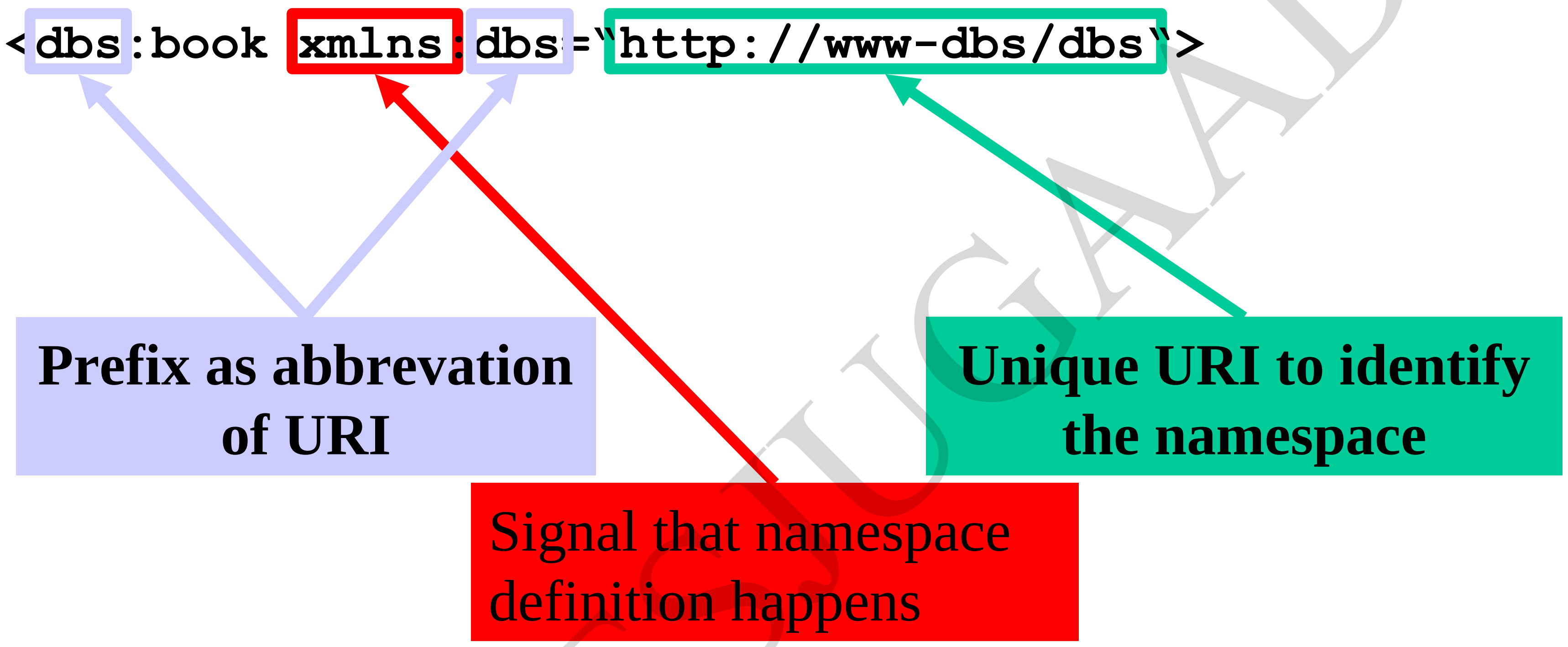
A **well-formed** document must adher to, among others, the following rules:

- Every start tag has a matching end tag.
- Elements may nest, but must not overlap.
- The
- Attr
- 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.

Namespace Syntax

```
<dbs:book xmlns:dbs=“http://www-dbs/dbs“>
```


Prefix as abbrevation of URI
Signal that namespace definition happens
Unique URI to identify the namespace

Namespace Example

```
<dbs:book xmlns:dbs="http://www-dbs/dbs">
  <dbs:description> ... </dbs:description>
  <dbs:text>
    <dbs:formula>
      <mathml:math
   xmlns:mathml="http://www.w3.org/1998/Math/MathML">
        ...
      </mathml:math>
    </dbs:formula>
  </dbs:text>
</dbs:book>
```

Default Namespace

- Default namespace may be set for an element and its content (but *not* its attributes):

```
<book xmlns=“http://www-dbs/dbs“>
  <description>...</description>
<book>
```

- Can be overridden in the elements by specifying the namespace there (using prefix or default namespace)

3.1 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.

```
<!DOCTYPE element DTD identifier
[ declaration1 declaration2 ......... ]>
```

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):

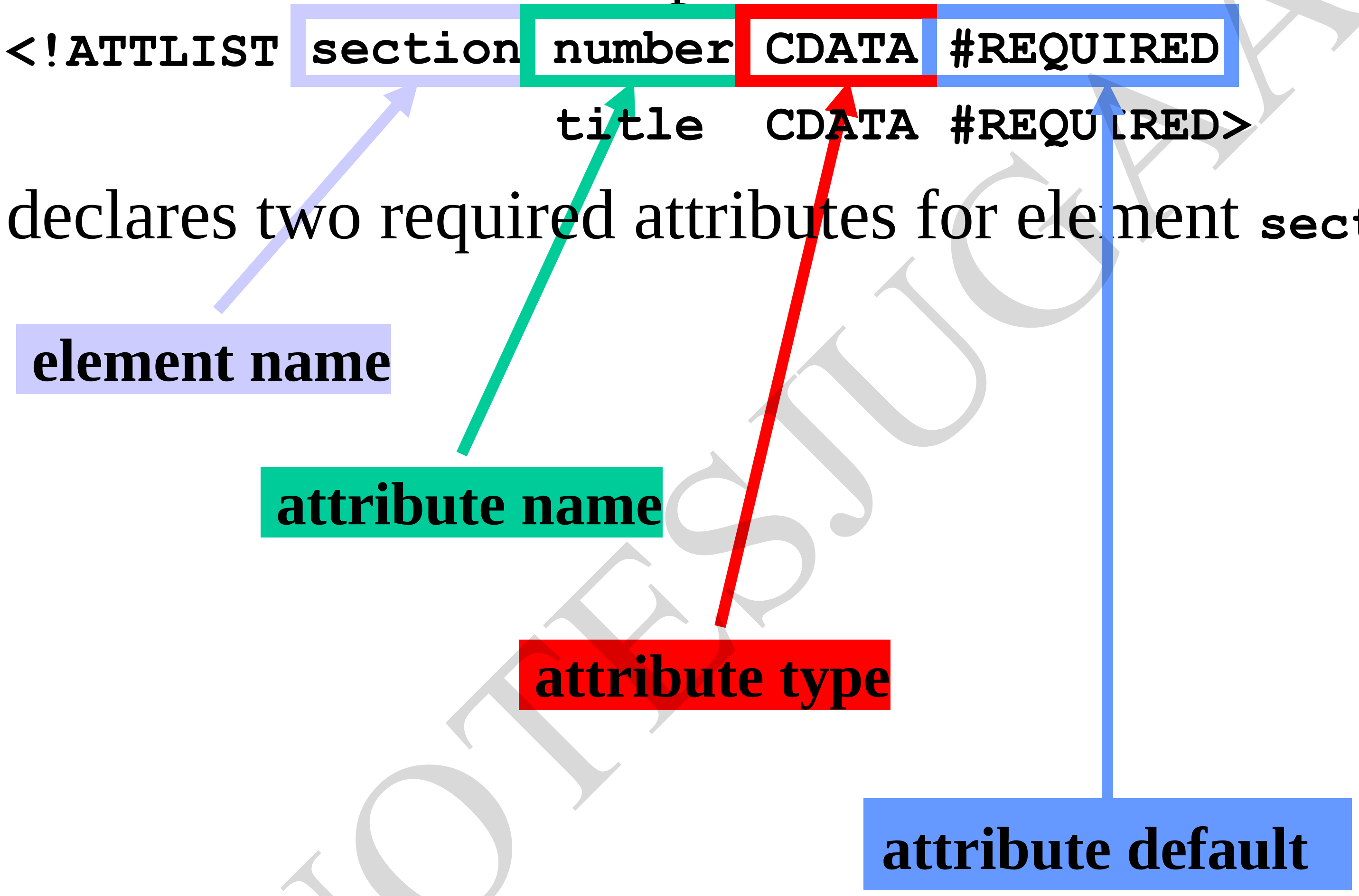
```
<!ELEMENT thesis ANY>
```

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

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>
```

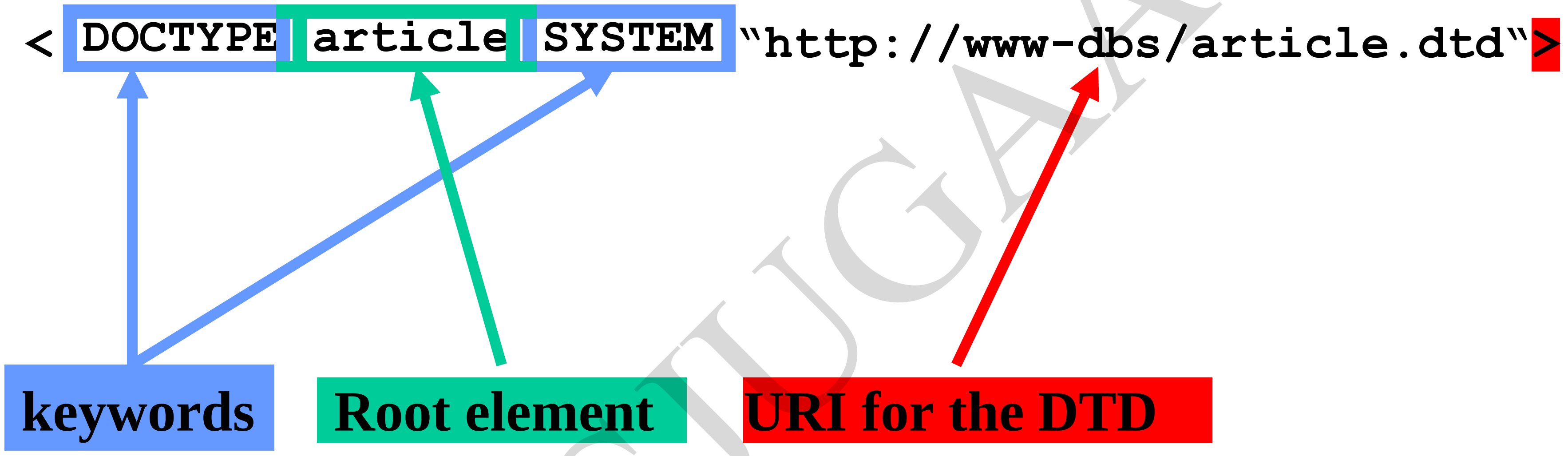
Attribute Examples

```
<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:



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

Linking DTD and XML Docs

- Internal DTD:

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

- Both ways can be mixed, internal DTD overwrites external entity information:

```
<!DOCTYPE article SYSTEM „article.dtd" [
  <!ENTITY % pub_content (title+,author*,text)
]>
```

Internal & External DTD

- <?xml version="1.0" encoding="UTF-8"?>
  <!DOCTYPE note SYSTEM "Note.dtd">

- <u>Note.dtd</u>

```
<!DOCTYPE note
  [
  <!ELEMENT note (to,from,heading,body)>
  <!ELEMENT to (#PCDATA)>
  <!ELEMENT from (#PCDATA)>
  <!ELEMENT heading (#PCDATA)>
  <!ELEMENT body (#PCDATA)>
  ]>
```

```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend</body>
</note>
```

XSD-Schema

- Syntax

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="x" type="y"/>
<xs:attribute name="x" type="y"/>
```

Example

Simple Type

```
        <xs:element name="phone_number" type="xs:int”/>
```

Complex Type

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="Address">
<xs:complexType>
<xs:sequence>
<xs:element name="name" type="xs:string" />
<xs:element name="company" type="xs:string" />
<xs:element name="phone" type="xs:int" />
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

Global Type

```
<xs:element name="AddressType">
<xs:complexType>
<xs:sequence>
<xs:element name="name" type="xs:string" />
<xs:element name="company" type="xs:string" />
</xs:sequence>
</xs:complexType>
</xs:element>
```

Loading XML DOC

```
<script type="text/javascript" language="javascript">
Fn fnname()
{
  var xmldoc;
  xmldoc=new ActiveXobject("Microsoft.XMLDOM")
  xmldoc.load("src")
  noderootelement=xmldoc.element;
  nodeelement=nodeelementname.firstchild;
  nodeelement=nodeelement.nextsibiling
  ……………………………………………..}
```

<u>**Example address.xml**</u>

```
<?xml version="1.0"?>
<contact-info>
<name>Tanmay Patil</name>
<company>TutorialsPoint</company>
 <phone>(011) 123-4567</phone>
 </contact-info>
```

<!DOCTYPE html> <html> <body> <h1>TutorialsPoint DOM example </h1> <div> <b>Name:</b> <span id="name"></span><br> <b>Company:</b> <span id="company"></span><br> <b>Phone:</b> <span id="phone"></span> </div>

```
 <script> if (window.XMLHttpRequest)
 {     // code for IE7+, Firefox, Chrome, Opera, Safari
xmlhttp = new XMLHttpRequest();
} else
{// code for IE6, IE5
xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
} xmlhttp.open("GET","/xml/address.xml",false);
  xmlhttp.send(); xmlDoc=xmlhttp.responseXML;
document.getElementById("name").innerHTML=
xmlDoc.getElementsByTagName("name")[0].childNodes[0].nodeValue;
document.getElementById("company").innerHTML=
xmlDoc.getElementsByTagName("company")[0].childNodes[0].nodeValue;
document.getElementById("phone").innerHTML=
xmlDoc.getElementsByTagName("phone")[0].childNodes[0].nodeValue;
```

</script> </body> </html>

XML - Namespaces

```
<?xml version="1.0" encoding="UTF-8"?>
<cont: contactxmlns:cont="www.tutorialspoint.com/profile">
 <cont:name>Tanmay Patil</cont:name>
<cont:company>TutorialsPoint</cont:company>
 <cont:phone>(011) 123-4567</cont:phone>
</cont:contact>
```

Xml with CSS

CATALOG.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<?xml-stylesheet type="text/css" href="CATALOG.css"?>
<CATALOG>
  <CD>
     <TITLE>Empire Burlesque</TITLE>
     <ARTIST>Bob Dylan</ARTIST>
     <COUNTRY>USA</COUNTRY>
     <COMPANY>Columbia</COMPANY>
     <PRICE>10.90</PRICE>
     <YEAR>1985</YEAR> </CD>
  <CD>
     <TITLE>Hide your heart</TITLE>
     <ARTIST>Bonnie Tyler</ARTIST>
     <COUNTRY>UK</COUNTRY>
     <COMPANY>CBS Records</COMPANY>
     <PRICE>9.90</PRICE>
     <YEAR>1988</YEAR> </CD> </CATALOG>
```

CATALOG.css

```
CATALOG { background-color: #ffffff; width: 100%; }
CD { display: block; margin-bottom: 30pt; margin-left: 0; }
TITLE { display: block; color: #ff0000; font-size: 20pt; }
ARTIST { display: block; color: #0000ff; font-size: 20pt; }
COUNTRY, PRICE, YEAR, COMPANY { display: block; color: #000000; margin-left: 20pt; }
```

Xml file

```xml
<?xml version="1.0" encoding="UTF-8"?>
<breakfast_menu>

<food>
<name>Belgian Waffles</name>
<price>$5.95</price>
<description>Two of our famous Belgian Waffles with
plenty of real maple syrup</description>
<calories>650</calories>
</food>
</ breakfast_menu>
```

Xml with xslt

```
<?xml version="1.0" encoding="UTF-8"?>
<html xsl:version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<body style="font-family:Arial;font-size:12pt;background-color:#EEEEEE">
<xsl:for-each select="breakfast_menu/food">
  <div style="background-color:teal;color:white;padding:4px">
    <span style="font-weight:bold"> <xsl:value-of select="name"/> - </span>
    <xsl:value-of select="price"/>
    </div>
  <div style="margin-left:20px;margin-bottom:1em;font-size:10pt">
    <p>
    <xsl:value-of select="description"/>
    <span style="font-style:italic"> (<xsl:value-of select="calories"/> calories per
serving)</span>
    </p>
  </div>
</xsl:for-each>
</body>
</html>
```

Xpath

```
/bookstore/book[1]
/bookstore/book[last()]
/bookstore/book[last()-1]
/bookstore/book[position()<3]
//title[@lang]
//title[@lang='en‘]
/bookstore/book[price>35.00]
/bookstore/book[price>35.00]/title
```

Xml Link

XLink Syntax

```
<?xml version="1.0" encoding="UTF-8"?>
<homepages xmlns:xlink="http://www.w3.org/1999/xlink">
 <homepage xlink:type="simple" xlink:href="http://www.w3schools.com">Visit W3Schools</homepage>
 <homepage xlink:type="simple" xlink:href="http://www.w3.org">Visit W3C</homepage>
</homepages>
```

Specifications

- xlink:actuate
- xlink:href
- xlink:show
- xlink:type

example

- <?xml version="1.0" encoding="UTF-8"?>

  <bookstore xmlns:xlink="http://www.w3.org/1999/xlink">
  <book title="Harry Potter">
  <description
  xlink:type="simple"
  xlink:href="/images/HPotter.gif"
  xlink:show="new">
  As his fifth year at Hogwarts School of Witchcraft and Wizardry approaches, 15-year-old Harry Potter is.......
  </description>
  </book>
- </bookstore>

Xml with ids

- <?xml version="1.0" encoding="UTF-8"?>

  <dogbreeds>

  <dog breed="Rottweiler" id="Rottweiler">
    <picture url="http://dog.com/rottweiler.gif" />
    <history>The Rottweiler's ancestors were probably Roman
    drover dogs.... </history>
    <temperament>Confident, bold, alert and imposing, the Rottweiler
    is a popular choice for its ability to protect....</temperament>
  </dog>
- </dogbreeds>

Xml with xpointer

```
• <?xml version="1.0" encoding="UTF-8"?>

  <mydogs xmlns:xlink="http://www.w3.org/1999/xlink">

  <mydog>
    <description>
    Anton is my favorite dog. He has won a lot of.....
    </description>
    <fact xlink:type="simple" xlink:href="http://dog.com/dogbreeds.xml#Rottweiler">
    Fact about Rottweiler
    </fact>
  </mydog>
• </mydogs>
```

Xml into the server

-
  ```
  <%
  'Load XML
  set xml = Server.CreateObject("Microsoft.XMLDOM")
  xml.async = false
  xml.load(Server.MapPath("simple.xml"))

  'Load XSL
  set xsl = Server.CreateObject("Microsoft.XMLDOM")
  xsl.async = false
  xsl.load(Server.MapPath("simple.xsl"))

  'Transform file
  Response.Write(xml.transformNode(xsl))
  %>
  ```