CSC309: Introduction to Web Programming

Lecture 2

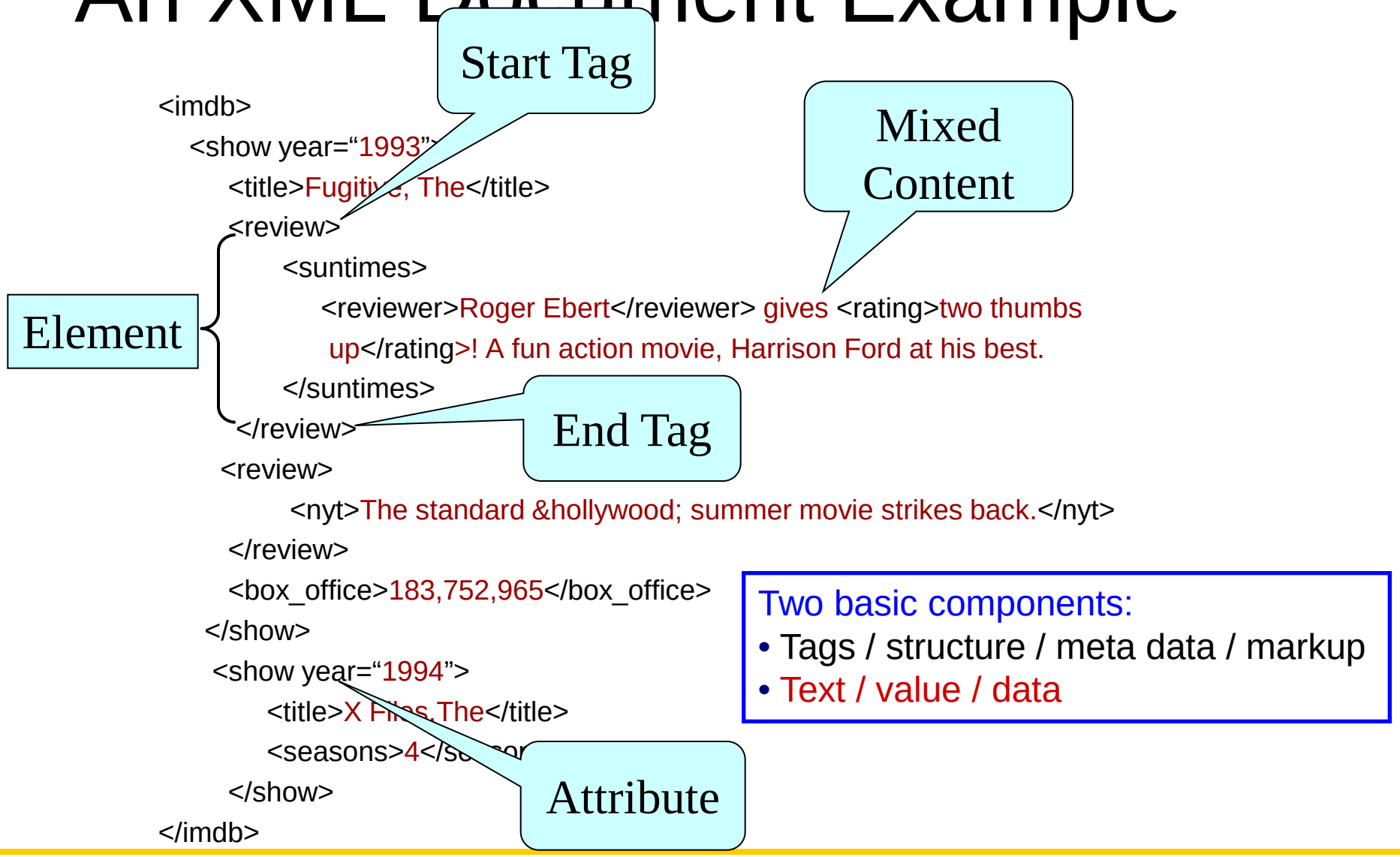
Wael Aboulsaadat

XML: Extensible Markup Language

- http://www.w3.org/XML/
- XML 1.0, W3C Recommendation Feb '98

- XML is just a syntax,
  But a standardized, extensible syntax
  - Compared with HTML, XML allows specification of new dialects by inventing tags.

An XML Document Example



Two basic components:

- Tags / structure / meta data / markup
- Text / value / data

Tags

- Describes the meaning of the text.
- Tags come in pairs: start tags and end tags.
  E. g .<show> ...</show>
- They must be properly nested
  <show> <title> ... </title> ... </show> --- good
  <show> <title> ... </show>... </title> --- bad
- The region between start tag and end tag defines an element.

Structure of XML Data

- Nesting tags can be used to express various structures.

  I.e. Elements are nested.

- E.g.

```
<show>
  <title> Fugitive, The </title>
</show>
```

Structure of XML Data (cont.)

- We can represent a list by using the *same* tag repeatedly:

```
<show>
   < review > ... </ review >
   < review > ... </ review >
   < review > ... </ review >
   ...
</ show>
```

Order matters!

Structure of XML Data (cont.)

- Premise: A text is the sum of its component parts
  - A <Book> could be defined as containing:
    <FrontMatter>, <Chapter>s, <BackMatter>
  - <FrontMatter> could contain:
    <BookTitle> <Author>s <PubInfo>
  - A <Chapter> could contain:
    <ChapterTitle> <Paragraph>s
  - A <Paragraph> could contain:
    <Sentence>s or <Table>s or <Figure>s …
- Components chosen should reflect anticipated use

Structure of XML Data (cont.)

- A useful, albeit imperfect, model
  - Exposes an object's intellectual structure
  - Supports reuse & abstraction of components
  - Better than a bit-mapped page image
  - Better than a model of text as a stream of characters plus formatting instructions
  - Data management system for document-like objects
  - Does not allow overlapping content objects
  - Incomplete; requires infrastructure

Content objects in a book

- **Book**
  - **FrontMatter**
    - **BookTitle**
    - **Author(s)**
    - **PubInfo**
  - **Chapter(s)**
    - **ChapterTitle**
    - **Paragraph(s)**
  - **BackMatter**
    - **References**
    - **Index**

```
<Book>
   <FrontMatter>
     <BookTitle>XML Is Easy</BookTitle>
     <Author>Tim Cole</Author>
     <Author>Tom Habing</Author>
     <PubInfo>CDP Press, 2002</PubInfo>
   </FrontMatter>
   <Chapter>
     <ChapterTitle>First Was SGML</Chapt
     <Paragraph>Once upon a time …</Para
   </Chapter>
</Book>
```

Attributes

- We can have attributes with name and value pairs within a start tag.

  E.g. <show year="1993"> … </show>

- Alternatively, we can represent the information as a nested element (instead of an attribute)

  E.g. <show> <year>1993</year> … </show>

- Differences between elements and attributes:
  - Elements are ordered, attributes are not
  - For an element, each attribute has a distinct name
- Which one to choose?

Text and Mixed Content

- XML has only one “basic” type -- text.
- It is bounded by tags e.g.
  <title> The Big Sleep </title>
- XML text is called PCDATA (for parsed character data).
- Can be mixed with other subelements --- Mixed Content
  <review> <reviewer>Roger Ebert</reviewer> gives <rating>two thumbs up</rating>! A fun action movie, Harrison Ford at his best. </review>
- Mixed content is very usefl for “document“ data
  - People speak in sentences. XML can preserve the structure of natural language, while adding semantic markup that can be interpreted by machines.

Continuous spectrum between text, semi-structured data, and structured data

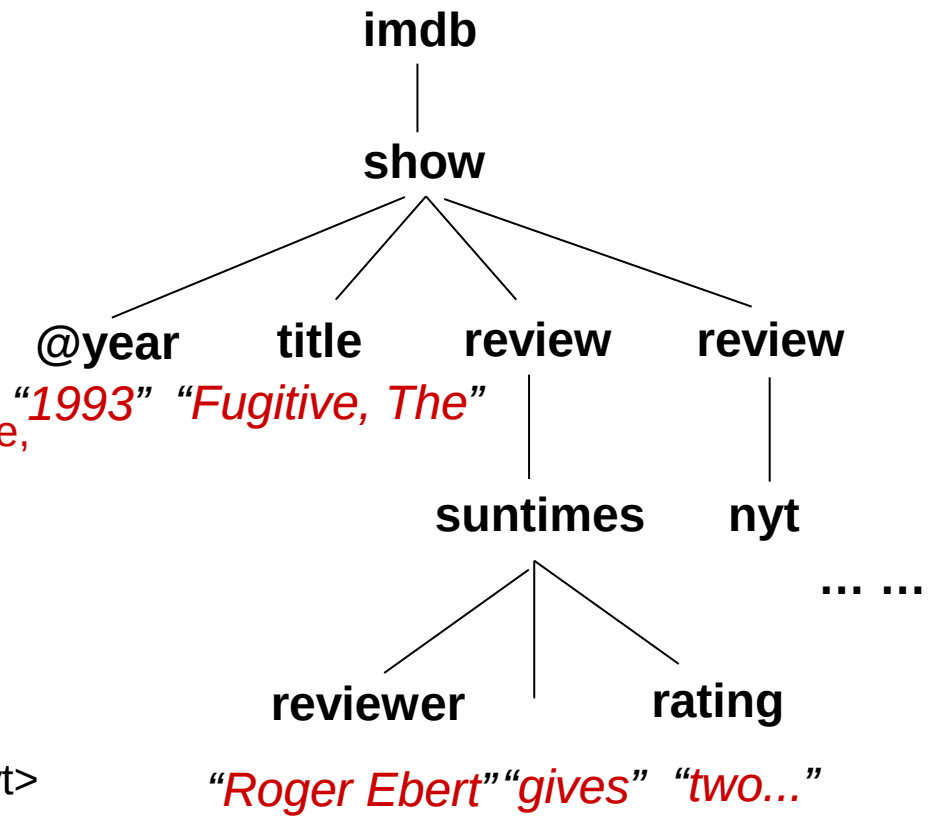
1 Roger Ebert gives two thumbs up! The Fugitive is a fun action movie, Harrison Ford at his best.

2 <review reviewer= “Roger Ebert”> two thumbs up! The Fugitive is a fun action movie, Harrison Ford at his best. </review>

3
```
<review>
    <reviewer>Roger Ebert</reviewer>
    <movie>
            <rating>two thumbs up</rating>
            <title> The Fugitive </title>
            <genre> action </genre>
            <star> Harrison Ford </star>
    </movie>
</review>
```

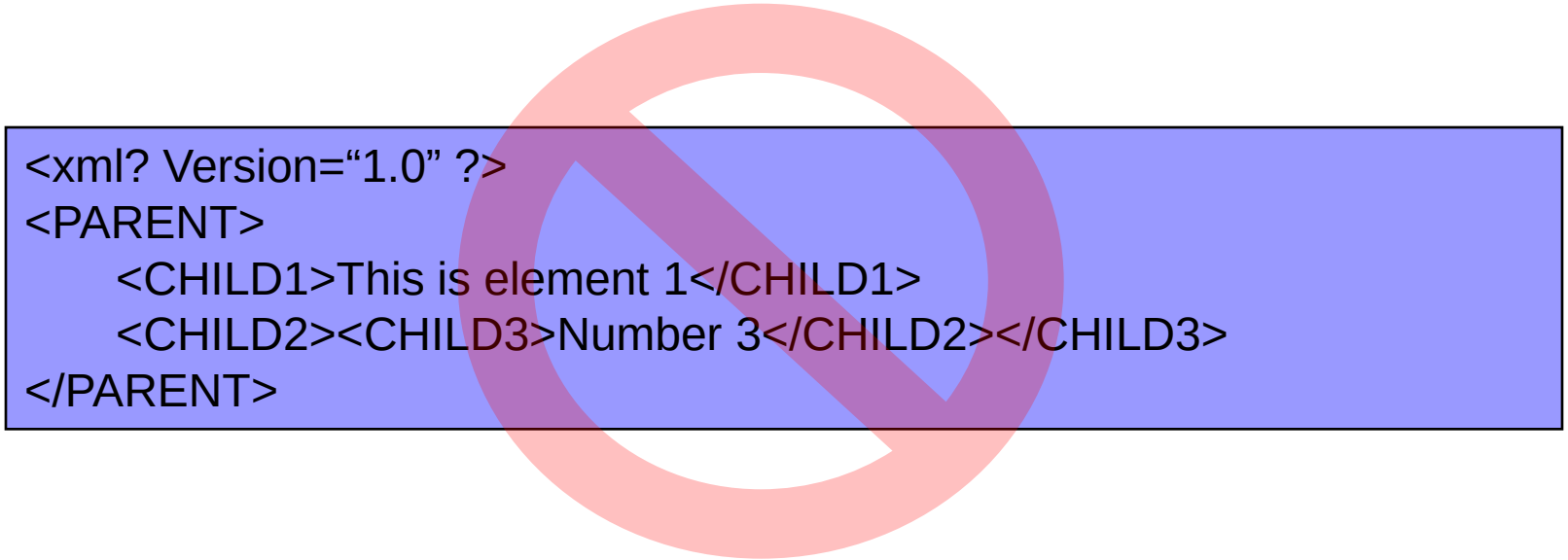
Representing XML Data as Trees

```
<imdb>
   <show year="1993">
      <title>Fugitive, The</title>
      <review>
          <suntimes>
             <reviewer>Roger
             Ebert</reviewer> gives
             <rating>two thumbs
             up</rating>! A fun action movie,
             Harrison Ford at his best.
          </suntimes>
      </review>
      <review>
          <nyt>The standard &hollywood;
          summer movie strikes back.</nyt>
      </review>
</imdb>
```



Syntax and Structure

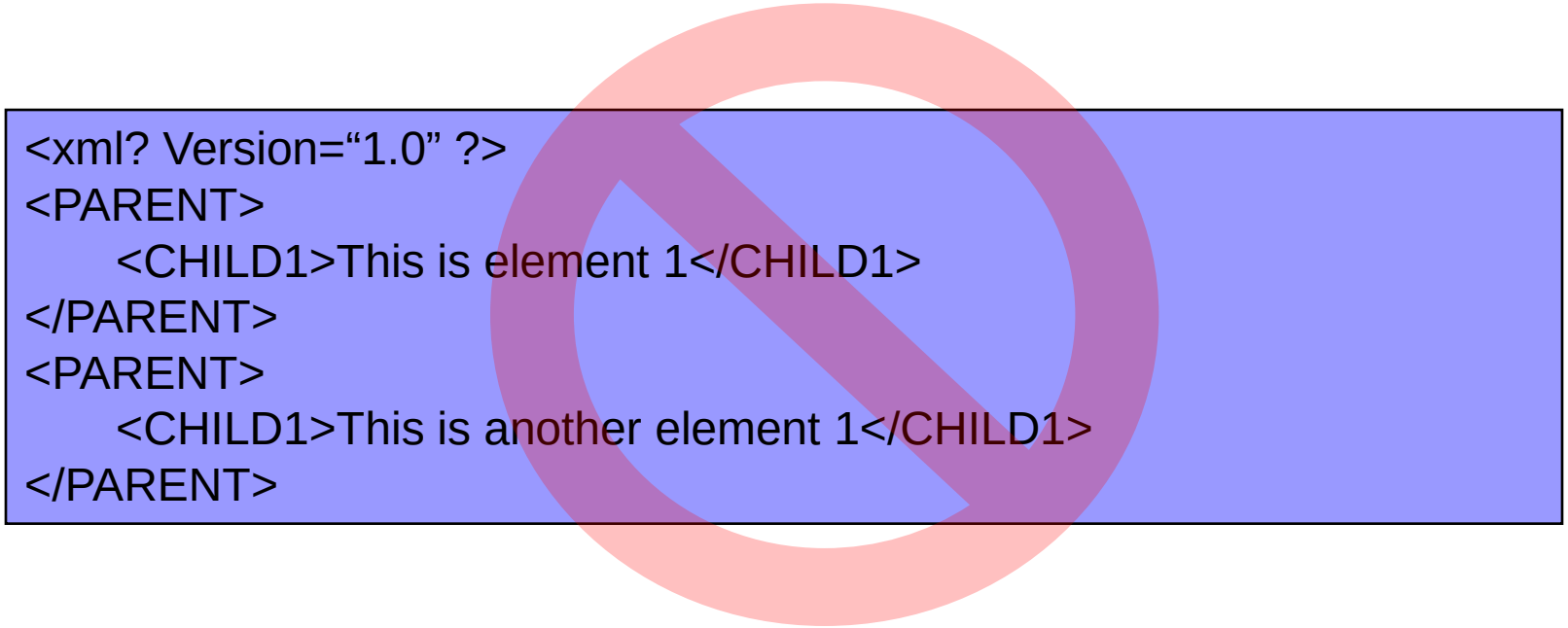
Well-Formed XML?



```
<xml? Version="1.0" ?>
<PARENT>
    <CHILD1>This is element 1</CHILD1>
    <CHILD2><CHILD3>Number 3</CHILD2></CHILD3>
</PARENT>
```

Syntax and Structure

Well-Formed XML?

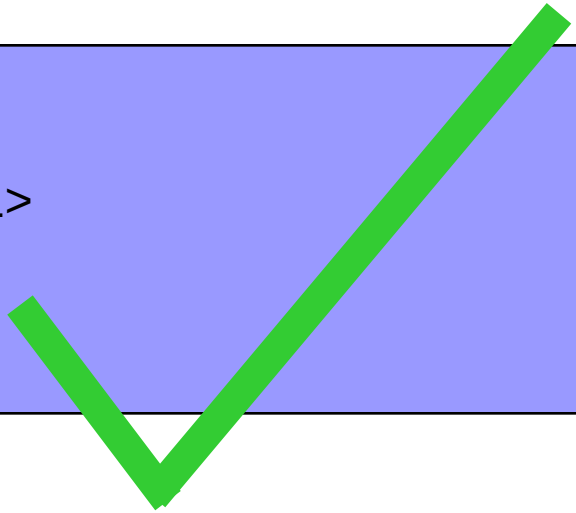


```
<xml? Version="1.0" ?>
<PARENT>
    <CHILD1>This is element 1</CHILD1>
</PARENT>
<PARENT>
    <CHILD1>This is another element 1</CHILD1>
</PARENT>
```

Syntax and Structure

Well-Formed XML?

```
<xml? Version="1.0" ?>
<PARENT>
        <CHILD1>This is element 1</CHILD1>
        <CHILD2/>
        <CHILD3></CHILD3>
</PARENT>
```



XML Dialects in Vertical Application Domains: Basically everywhere!

- HealthCare Level Seven (HL7)
- Geography Markup Language (GML)
- Systems Biology Markup Language (SBML)
- Digital photography metadata (XMP)
- Extensible Financial Reporting Markup Language (XFRML)
- MusicXML,
- Spacecraft Markup Language (SML)
- Bank Internet Payment System (BIPS),
- Bioinformatic Sequence Markup Language (BSML),
- Chemical Markup Language (CML),
- Electronic Business XML Initiative (ebXML),
- FinXML, Financial Information eXchange protocol (FIX),
- Scalable Vector Graphics (SVG),
- Real Estate Listing Markup Language (RELML), . . .
- More at: http://xml.coverpages.org/gen-apps.html
  http://www.xml.org/xml/industry_industrysectors.jsp

RSS

- RSS 2.0 (Really Simple Syndication): web feed formats used to publish frequently updated digital content, such as blogs, news feeds.

- RSS delivers its information as an XML file called an "RSS feed", "webfeed", "RSS stream", or "RSS channel".

- RSS Users: 'subscribes' to a feed by supplying to their reader a link to the feed;
- Feed readers (or aggregators): client softwares that regularly check a list of feeds on behalf of a user, pull and display any updated content that they find.

- RSS readers/ feed readers / feed aggregators / news readers / search aggregators:

  Google Reader, My Yahoo!, Bloglines, web browsers

RSS - sample

```
<rss version="0.91">
  <channel>
    <title>XML.com</title>
    <link>http://www.xml.com/</link>
    <description>XML.com features a rich mix of information and services for the XML community.</description>
    <language>en-us</language>
    <item>
      <title>Normalizing XML, Part 2</title>
      <link>http://www.xml.com/pub/a/2002/12/04/normalizing.html</link>
      <description>In this second and final look at applying relational normalization techniques to W3C XML Sche
    </item>
    <item>
      <title>The .NET Schema Object Model</title>
      <link>http://www.xml.com/pub/a/2002/12/04/som.html</link>
      <description>Priya Lakshminarayanan describes in detail the use of the .NET Schema Object Model for prog
    </item>
    <item>
      <title>SVG's Past and Promising Future</title>
      <link>http://www.xml.com/pub/a/2002/12/04/svg.html</link>
      <description>In this month's SVG column, Antoine Quint looks back at SVG's journey through 2002 and look
    </item>
  </channel>
</rss>
```

Microsoft Office in XML

- Office 2003 was able to import/export all documents into XML
- Office 2007 models the documents NATIVELY in XML (Microsoft Office Open XML, i.e. OOXML)
- Examples of vocabularies and schemas:
  - WordprocessingML (the XML file format for Word 2003), SpreadsheetML (Excel 2003), and DataDiagramingML (Visio 2003)

Web Services

- Web service: a software system designed to support interoperable machine to machine interaction over a network.

- Web service protocol stack
  - Service transport: HTTP, SMTP, ..
  - XML messaging: SOAP(Simple Object Access Protocol)
  - Service description: WSDL(web service descripton language), an XML based language that provides a model for describing web services
  - Service discovery: UDDI (universal description, discovery and integration), an XML based registry for business to list themselves on the Internet

XML Isn't Enough on Its Own

It's is just a data format, but we care about data!

- How to design XML format for a given domain? Sometimes more structure about the data is helpful.
- How can we know when we are getting garbage?
- How can we query the data?
- How can we understand what we got?

XML Standards Landscape

- Schema languages
  - DTDs: http://www.w3schools.com/dtd/default.asp
  - XML Schemas: http://www.w3.org/XML/Schema
- Programming APIs
  - DOM (Document Object Model): http://www.w3.org/DOM/
  - SAX (Simple API for XML): http://www.saxproject.org/
  - JAXP (Java API for XML Processing): http://java.sun.com/webservices/jaxp/
- Query languages
  - XPath: http://www.w3.org/TR/xpath
  - XQuery: http://www.w3.org/TR/xquery/
  - XSLT: http://www.w3.org/TR/xslt
- Standard organizations
  - W3C (the World Wide Web Consortium)
  - OASIS (Organization for the Advancement of Structured Information Standards)

XML Schema

- Schema: a model of the data
  - Structural definitions
  - Type definitions
  - Defaults
- Useful for
  - validating data
  - facilitate the writing of applications that process data
  - facilitate the writing of queries
  - designing storage and query evaluation strategies
  - defining prior agreements between parties for data exchange
  - mapping to programming languages (e.g. Java, C#)

DTD: Document Type Descriptors

- Part of the original XML 1.0 specification
- Describe the “grammar” of the XML file
  - Element declarations: how elements are allowed to nest within each other by rules
  - Attributes lists: describe what attributes are allowed on which element
  - Some constraints on the value of elements and attributes
  - the root element of the XML tree

Sample XML Data

```
<imdb>
   <show year="1993">
      <title>Fugitive, The</title>
      <review>
            <suntimes>
               <reviewer>Roger Ebert</reviewer>
                  <rating>two thumbs up</rating>
                  ! A fun action movie, Harrison Ford at his best.
            </suntimes>
       </review>
      <review>
             <nyt>The standard &hollywood; summer movie strikes back.</nyt>
      </review>
      <box_office>183,752,965</box_office>
    </show>
     <show year="1994">
          <title>X Files,The</title>
          <seasons>4</seasons>
      </show>
</imdb>
```

Exactly one title

As many reviews as needed after title

Box office

or seasons info

Specifying the Structure

- title — to specify a title element
- director? — to specify an optional (0 or 1) director elements
- review* — to specify 0 or more review
- title, review+ — to specify a title followed by 1 or more review
- box_office | seasons — to specify a box_office *or* a seasons element

Specifying the Structure (cont)

- So the whole structure of a movie element is specified by

  <!ELEMENT show (title, review*, (season|box_office))>

- This is known as a regular expression

- PCDATA: only textual content allowed

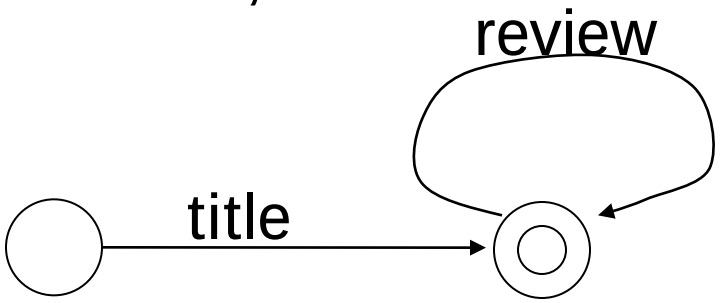
  <!ELEMENT title(#PCDATA)>

Regular Expressions

- Each regular expression determines a corresponding finite state automaton

- Let's start with a simpler example:

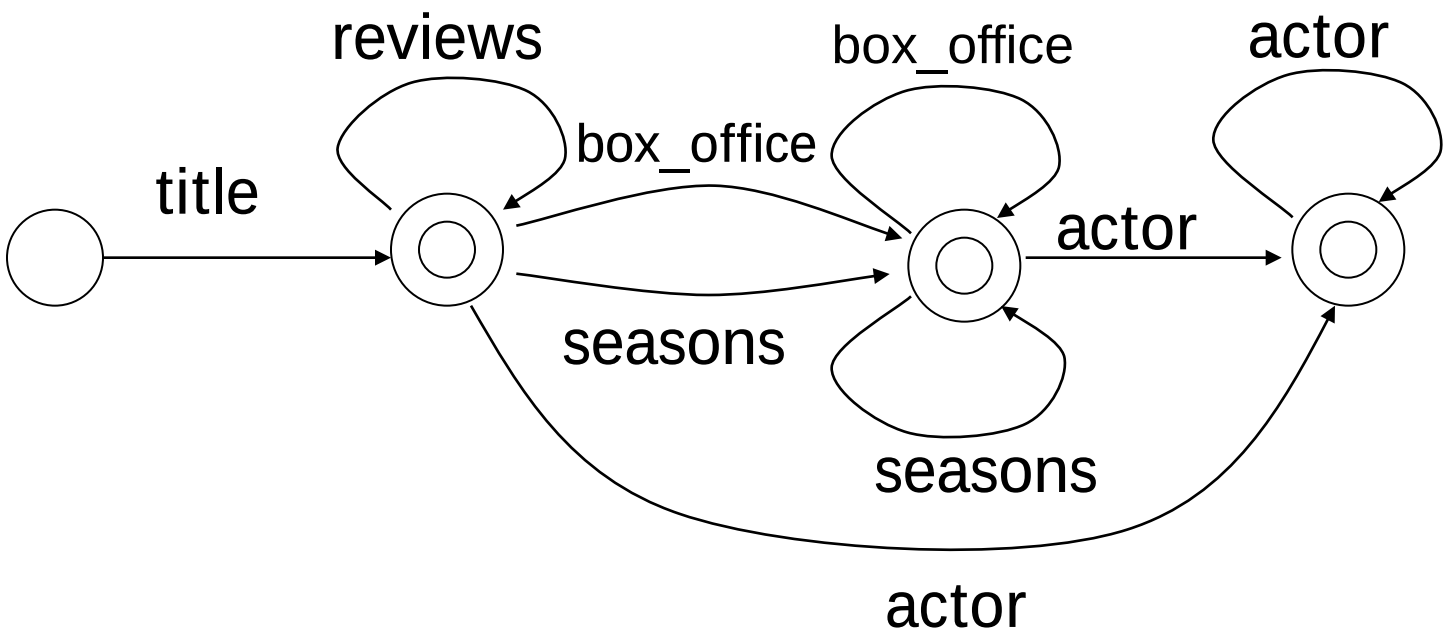
title, review*



- This suggests a simple parsing program to validate XML data

A More Complicated Example

title,review*,(box_offcie | seasons)*,actor*



Defining the attribute lists

- <!ATTLIST element-name attribute-name attribute-type default-value>
  - attribute-type: CDATA, ...
  - default value:
    - value: the default value of the attribute
    - #REQUIRED: the attribute value must be included
    - #IMPLIED: attribute is optional
    - #FIXED value: the attribute value is fixed.

- E.g. <!ATTLIST show year CDATA #IMPLIED>

The DTD for the Sample XML Data

- DTD

```
<!DOCTYPE imdb [
    <!ELEMENT imdb (show*)>
    <!ELEMENT show (title, review*, (box_office |seasons))>
    <!ATTLIST show year CDATA #IMPLIED>
    <!ELEMENT title (#PCDATA)>
……
]>
```

- Indicating DTD in an XML file:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE imdb SYSTEM “my.dtd">
<imdb>…
```

DTDs Aren't Enough Sometimes

DTDs capture grammatical structure, but have some drawbacks:

- Not themselves in XML – inconvenient to build tools for them
- No built-in data types and domains
- Limited abilities to specify constraints on values
- No way of defining OO-like inheritance

XML Schema of the Data

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="show">
     <xs:complexType>
       <xs:sequence>
          <xs:element name="title" type="xs:string"/>
          <xs:element name="review" mixed="true" minoccurs="1"
          maxoccurs="unbounded" />
          <xs:choice>
               <xs:element name="box_office" type="xs:integer"/>
               <xs:element name="seasons" type="xs:integer"/>
          </xs:choice>
       </xs:sequence>
       <xs:attribute name="year" type="xs:integer" use="optional"/>
     </xs:complexType>
  </xs:element>
</xs:schema>
```

XML namespace, specified by a URI

By default, minoccurs="1" maxoccurs="1" type = "xsd:anyType"

The value of "use" can be "optional" or "required". We can also have attribute "fixed", "default"

DTD vs XML Schema

XML Schema is more expressive

- It defines data type information
  - Simple types and complex types
  - Built-in types and user-defined types
- Can specify the cardinality of an element within its parent element
- Can specify expressive value constraints, such as keys, and foreign keys (more on this later!)
- XML Schema is an also XML file!

Well-Formed XML

Well-formed applies to any XML document (with or without a DTD)

- All open-tags have matching close-tags, or a special case:

  <tag/> is a shortcut for <tag></tag>
- Attributes (which are unordered, in contrast to elements) only appear once within an element
- There's a single root element
- XML is case-sensitive

Valid XML

- *Valid* specifies that the document conforms to the DTD or XML Schema.
  - conforms to the grammar
  - the types of attributes are correct
  - constraints on references are satisfied

Referring to a schema

- To refer to a DTD in an XML document, the reference goes *before* the root element:
  - <?xml version="1.0"?>
    <!DOCTYPE rootElement SYSTEM "***url***">
    <rootElement> … </rootElement>
- To refer to an XML Schema in an XML document, the reference goes *in* the root element:
  - <?xml version="1.0"?>
    <rootElement
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    (The XML Schema Instance reference is required)
    xsi:noNamespaceSchemaLocation="***url***.xsd">
    (This is where *your* XML Schema definition can be found)
    …
    </rootElement>

The XSD document

- Since the XSD is written in XML, it can get confusing which we are talking about
- Except for the additions to the root element of our XML data document, the rest of this lecture is about the XSD schema document
- The file extension is .xsd
- The root element is <schema>
- The XSD starts like this:
  - <?xml version="1.0"?>
    <xs:schema xmlns:xs="http://www.w3.rg/2001/XMLSchema">

<schema>

- The <schema> element may have attributes:
  - xmlns:xs="http://www.w3.org/2001/XMLSchema"
    - This is necessary to specify where all our XSD tags are defined
  - elementFormDefault="qualified"
    - This means that all XML elements must be qualified (use a namespace)
    - It is highly desirable to qualify all elements, or problems will arise when another schema is added

“Simple” and “complex” elements

- A “simple” element is one that contains text and nothing else
  - A simple element cannot have attributes
  - A simple element cannot contain other elements
  - A simple element cannot be empty (!)
  - However, the text can be of many different types, and may have various restrictions applied to it
- If an element isn’t simple, it’s “complex”
  - A complex element may have attributes
  - A complex element may be empty, or it may contain text, other elements, or both text and other elements

Defining a simple element

- A simple element is defined as

  <xs:element name="***name***" type="***type***" />

  where:
  - ***name*** is the name of the element
  - the most common values for ***type*** are

    xs:boolean xs:integer
    xs:date xs:string
    xs:decimal xs:time
- Other attributes a simple element may have:
  - default="***default value***" *if no other value is specified*
  - fixed="***value***" *no other value may be specified*

Defining an attribute

- Attributes themselves are always declared as simple types
- An attribute is defined as
  <xs:attribute name="***name***" type="***type***" />
  where:
  - ***name*** and ***type*** are the same as for xs:element
- Other attributes a simple element may have:
  - default="***default value***" *if no other value is specified*
  - fixed="***value***" *no other value may be specified*
  - use="optional" *the attribute is not required (default)*
  - use="required" *the attribute must be present*

Restrictions, or “facets”

- The general form for putting a restriction on a text value is:
  - <xs:element name="*name*">

```
    <xs:restriction base="type">
        ... the restrictions ...
    </xs:restriction>
</xs:element>
```

(or xs:attribute*)*

- For example:
  - <xs:element name="age">

```
    <xs:restriction base="xs:integer">
        <xs:minInclusive value="0">
        <xs:maxInclusive value="140">
    </xs:restriction>
</xs:element>
```

Restrictions on numbers

- minInclusive -- number must be ≥ the given ***value***
- minExclusive -- number must be > the given ***value***
- maxInclusive -- number must be ≤ the given ***value***
- maxExclusive -- number must be < the given ***value***
- totalDigits -- number must have exactly ***value*** digits
- fractionDigits -- number must have no more than ***value*** digits after the decimal point

Restrictions on strings

- length -- the string must contain exactly ***value*** characters
- minLength -- the string must contain at least ***value*** characters
- maxLength -- the string must contain no more than ***value*** characters
- pattern -- the ***value*** is a regular expression that the string must match
- whiteSpace -- not really a “restriction”--tells what to do with whitespace
  - value="preserve" Keep all whitespace
  - value="replace" Change all whitespace characters to spaces
  - value="collapse" Remove leading and trailing whitespace, and replace all sequences of whitespace with a single space

Enumeration

- An enumeration restricts the value to be one of a fixed set of values
- Example:

```
<xs:element name="season">
    <xs:simpleType>
        <xs:restriction  base="xs:string">
            <xs:enumeration value="Spring"/>
            <xs:enumeration value="Summer"/>
            <xs:enumeration value="Autumn"/>
            <xs:enumeration value="Fall"/>
            <xs:enumeration value="Winter"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

Complex elements

- A complex element is defined as

```
<xs:element   name="name">
    <xs:complexType>
        ... information about the complex type...
    </xs:complexType>
</xs:element>
```

- Example:

```
<xs:element   name="person">
    <xs:complexType>
        <xs:sequence>
            <xs:element  name="firstName"  type="xs:string" />
            <xs:element  name="lastName"  type="xs:string" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

- <xs:sequence> says that elements must occur in this order
- Remember that attributes are always simple types

Global and local definitions

- Elements declared at the “top level” of a <schema> are available for use throughout the schema
- Elements declared within a xs:complexType are local to that type
- Thus, in

```
<xs:element   name="person">
    <xs:complexType>
        <xs:sequence>
            <xs:element  name="firstName"  type="xs:string" />
            <xs:element  name="lastName"  type="xs:string" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

  the elements firstName and lastName are only locally declared
- The order of declarations at the “top level” of a <schema> *do not* specify the order in the XML data document

Declaration and use

- So far we've been talking about how to *declare* types, not how to *use* them
- To *use* a type we have declared, use it as the value of type="…"
  - Examples:
    - <xs:element name="student" type="person"/>
    - <xs:element name="professor" type="person"/>
  - Scope is important: you cannot use a type if is local to some other type

xs:sequence

- We've already seen an example of a complex type whose elements must occur in a specific order:
-
```
<xs:element   name="person">
   <xs:complexType>
     <xs:sequence>
       <xs:element  name="firstName"  type="xs:string" />
       <xs:element  name="lastName"  type="xs:string" />
     </xs:sequence>
   </xs:complexType>
 </xs:element>
```

xs:all

- xs:all allows elements to appear in any order
- ```
  <xs:element   name="person">
     <xs:complexType>
        <xs:all>
           <xs:element  name="firstName"  type="xs:string" />
           <xs:element  name="lastName"  type="xs:string" />
        </xs:all>
     </xs:complexType>
  </xs:element>
  ```
- Despite the name, the members of an xs:all group can occur once or not at all
- You can use minOccurs="0" to specify that an element is optional (default value is 1)
  - In this context, maxOccurs is always 1
  ```

Referencing

- Once you have defined an element or attribute (with name="…"), you can refer to it with ref="…"
- Example:
  - <xs:element name="person">
    <xs:complexType>
    <xs:all>
    <xs:element name="firstName" type="xs:string" />
    <xs:element name="lastName" type="xs:string" />
    </xs:all>
    </xs:complexType>
    </xs:element>
  - <xs:element name="student" ref="person">
  - Or just: <xs:element ref="person">

Text element with attributes

- If a text element has attributes, it is no longer a simple type
  -
```
<xs:element  name="population">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension  base="xs:integer">
                <xs:attribute  name="year”
                              type="xs:integer">
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>
```

Empty elements

- Empty elements are (ridiculously) complex

- ```
  <xs:complexType name="counter">
      <xs:complexContent>
          <xs:extension base="xs:anyType"/>
          <xs:attribute name="count" type="xs:integer"/>
      </xs:complexContent>
  </xs:complexType>
  ```
  ```

Mixed elements

- Mixed elements may contain both text and elements
- We add `mixed="true"` to the `xs:complexType` element
- The text itself is not mentioned in the element, and may go anywhere (it is basically ignored)

```
<xs:complexType  name="paragraph"  mixed="true">
    <xs:sequence>
        <xs:element  name="someName”
                     type="xs:anyType"/>
    </xs:sequence>
</xs:complexType>
```

Extensions

- You can base a complex type on another complex type
-
```
<xs:complexType  name="newType">
    <xs:complexContent>
        <xs:extension  base="otherType">
            ...new stuff...
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

Predefined string types

- Recall that a simple element is defined as:
  <xs:element name="***name***" type="***type***" />
- Here are a few of the possible string types:
  - xs:string -- a string
  - xs:normalizedString -- a string that doesn't contain tabs, newlines, or carriage returns
  - xs:token -- a string that doesn't contain any whitespace other than single spaces
- Allowable restrictions on strings:
  - enumeration, length, maxLength, minLength, pattern, whiteSpace

Predefined date and time types

- xs:date -- A date in the format ***CCYY-MM-DD***, for example, 2002-11-05
- xs:time -- A date in the format ***hh*:*mm*:*ss*** (hours, minutes, seconds)
- xs:dateTime -- Format is ***CCYY-MM-DD*T*hh*:*mm*:*ss***
  - The T is part of the syntax
- Allowable restrictions on dates and times:
  - enumeration, minInclusive, minExclusive, maxInclusive, maxExclusive, pattern, whiteSpace

Predefined numeric types

- Here are some of the predefined numeric types:

xs:decimal	xs:positiveInteger
xs:byte	xs:negativeInteger
xs:short	xs:nonPositiveInteger
xs:int	xs:nonNegativeInteger
xs:long	

- Allowable restrictions on numeric types:
  - enumeration, minInclusive, minExclusive, maxInclusive, maxExclusive, fractionDigits, totalDigits, pattern, whiteSpace

Opinion

- If you like C++, you'll love XSD
  - Enjoy the feeling of knowing something arcane that isn't understood by lesser mortals
  - If you hope to rule the world, you'll need a schema language with this much power
- If you dislike debugging, you'll hate XSD
  - XSD is complex and error-prone, with lots of gotchas
- Be prepared
  - XSD is a W3C standard, which makes it important
  - If you work with XML, you may have to use XSD
  - I hope this brief introduction will make it easier to get started
  - XSD is probably one of the reasons that DTDs are still so popular

XML Schemas

- **Simple XML Document "note.xml":**

```
<?xml version="1.0"?>

<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

XML Schemas

```
<?xml version="1.0"?>

<xs:schema xmlns:xs= “http://www.w3.org/2001/XMLSchema”
targetNamespace=     “http://www.w3schools.com”
xmlns=               “http://www.w3schools.com”
elementFormDefault=  "qualified">

<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to"      type="xs:string"/>
      <xs:element name="from"    type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body"    type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

Summary

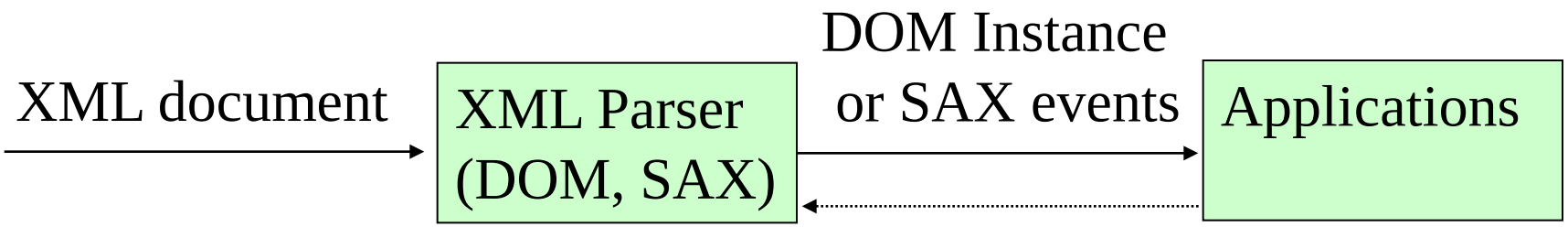
- As a data format, the main virtues of XML are its widespread acceptance and the (important) ability to handle semi-structured data (data without schema)

- Problems remain:
  - How to store large XML documents?
  - How to query them?
  - How to map XML from/to other data representation formats?

XML Standard Landscape

- Schema languages
  - DTDs
  - XML Schemas
- Programming APIs
  - DOM
  - SAX
- Query languages
  - XPath
  - XQuery
  - XSLT

Programming APIs



- DOM
http://www.w3.org/DOM/
http://www.w3schools.com/dom/

- SAX
http://www.saxproject.org/

DOM: Document Object Model

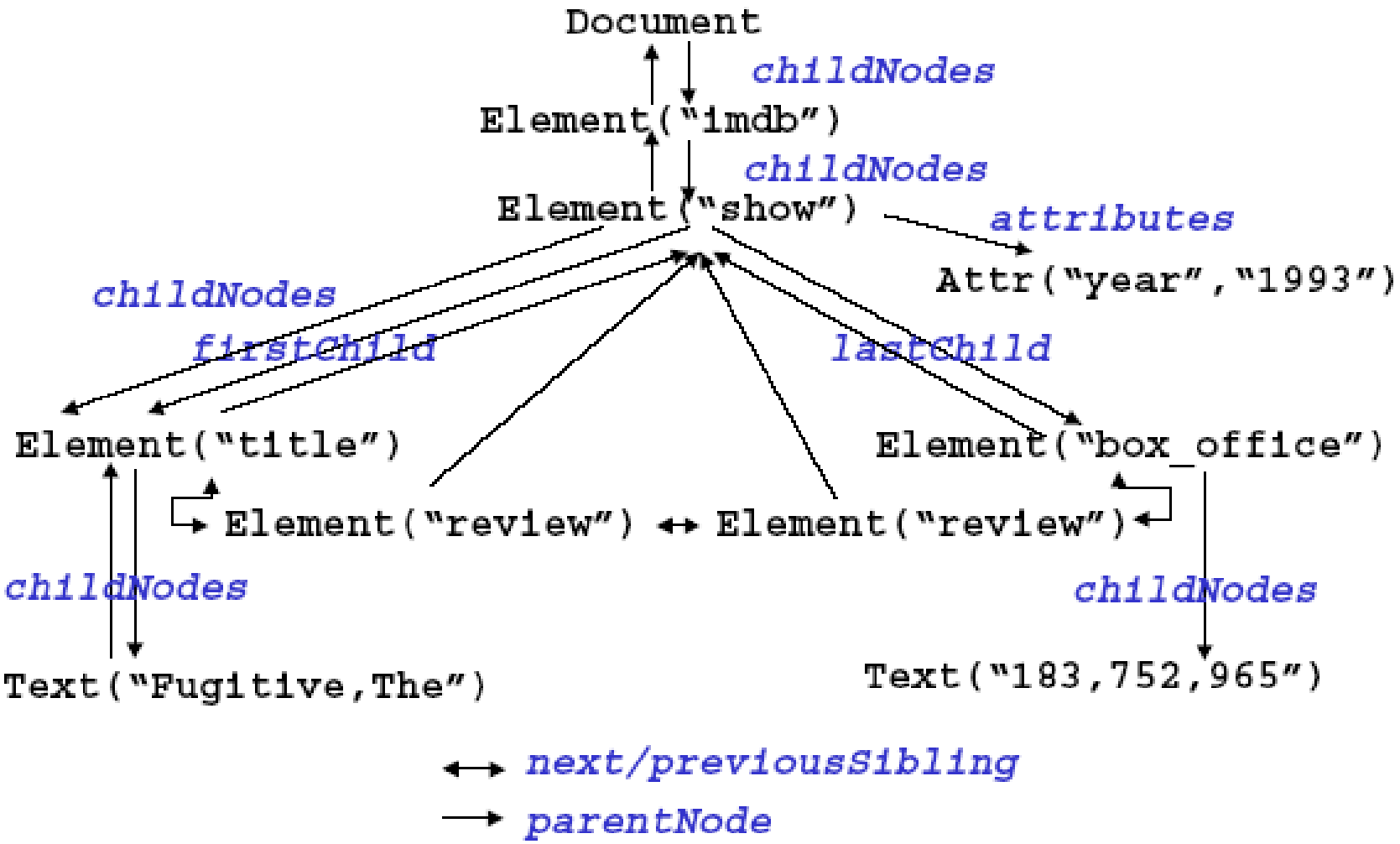
- Build a tree data structure (in memory)
- Provide accesses to nodes in a tree.

- Level 1. Functionality for XML document navigation and manipulation.
  - Level 2. Stylesheet and namespaces
  - Level 3. Document loading and saving DTDs and schemas

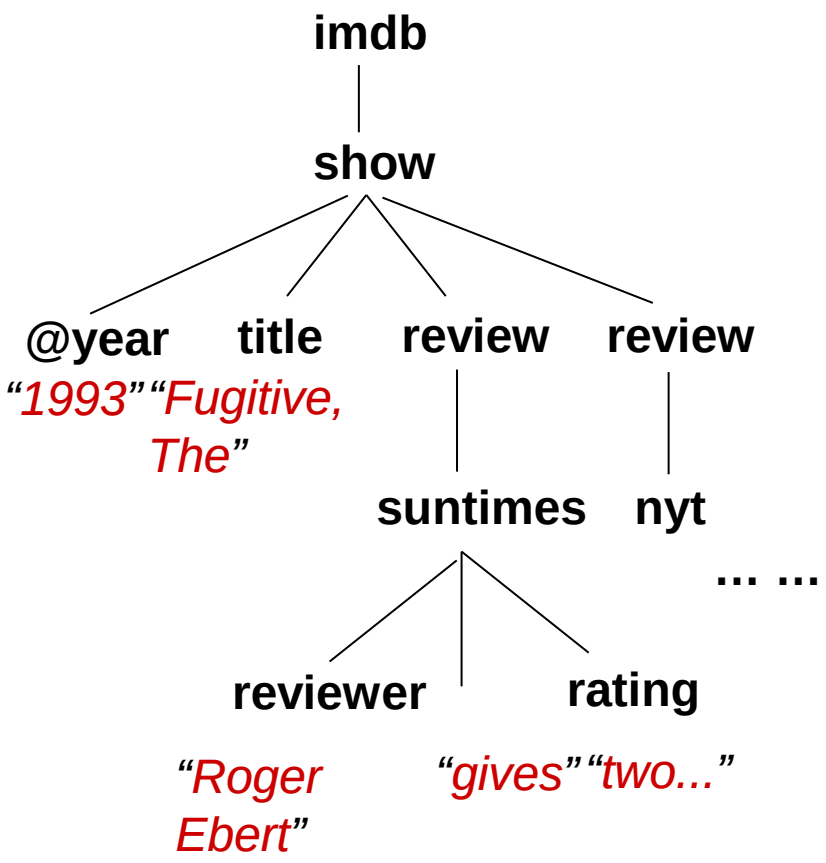
DOM Interfaces

- Interface: Document
  - Method: createElement; getElementsByTagName...
- Interface: Node
  - Attribute: parentNode, childNodes, firstChild, nextSibling, attributes...
  - Method: appendChild; removeChild...
- Interface: Element
  - Method: getAttributeNode, removeAttributeNode….

DOM: an Example



Navigating DOM trees



What does this code fragment do?

```
var x=getElementsByTagName('title')
for (i=0;i<x.length;i++) {
    document.write(x[i].childNodes[0].
    nodeValue)
}
```