# *Internet Applications*

## Chapter 7

# *Lecture Overview*

- Internet Concepts
- Web data formats
  - HTML, XML, DTDs
- Introduction to three-tier architectures
- The presentation layer
  - HTML forms; HTTP Get and POST, URL encoding; Javascript; Stylesheets. XSLT
- The middle tier
  - CGI, application servers, Servlets, JavaServerPages, passing arguments, maintaining state (cookies)

# *Uniform Resource Identifiers*

- Uniform naming schema to identify *resources* on the Internet
- A resource can be anything:
  - Index.html
  - mysong.mp3
  - picture.jpg

- Example URIs:

  http://www.cs.wisc.edu/~dbbook/index.html
  mailto:webmaster@bookstore.com

# *Structure of URIs*

http://www.cs.wisc.edu/~dbbook/index.html

- URI has three parts:
  - Naming schema (http)
  - Name of the host computer (www.cs.wisc.edu)
  - Name of the resource (~dbbook/index.html)
- URLs are a subset of URIs

# *Hypertext Transfer Protocol*

- What is a communication protocol?
  - Set of standards that defines the structure of messages
  - Examples: TCP, IP, HTTP
- What happens if you click on www.cs.wisc.edu/~dbbook/index.html?
- Client (web browser) sends HTTP request to server
- Server receives request and replies
- Client receives reply; makes new requests

# *HTTP (Contd.)*

Client to Server:

GET ~/index.html HTTP/1.1
User-agent: Mozilla/4.0
Accept: text/html, image/gif, image/jpeg

Server replies:

HTTP/1.1 200 OK
Date: Mon, 04 Mar 2002 12:00:00 GMT
Server: Apache/1.3.0 (Linux)
Last-Modified: Mon, 01 Mar 2002 09:23:24 GMT
Content-Length: 1024
Content-Type: text/html
<HTML> <HEAD></HEAD>
<BODY>
<h1>Barns and Nobble Internet Bookstore</h1>
Our inventory:
<h3>Science</h3>
The Character of Physical Law
...

# *HTTP Protocol Structure*

HTTP Requests

- Request line: GET ~/index.html HTTP/1.1
  - GET: Http method field (possible values are GET and POST, more later)
  - ~/index.html: URI field
  - HTTP/1.1: HTTP version field
- Type of client: User-agent: Mozilla/4.0
- What types of files will the client accept:
  Accept: text/html, image/gif, image/jpeg

# *HTTP Protocol Structure (Contd.)*

HTTP Responses

- Status line: HTTP / 1.1 200 OK
  - HTTP version: HTTP / 1.1
  - Status code: 200
  - Server message: OK
  - Common status code / server message combinations:
    - 200 OK: Request succeeded
    - 400 Bad Request: Request could not be fulfilled by the server
    - 404 Not Found: Requested object does not exist on the server
    - 505 HTTP Version not Supported
- Date when the object was created:
  Last-Modified: Mon, 01 Mar 2002 09:23:24 GMT
- Number of bytes being sent: Content-Length: 1024
- What type is the object being sent: Content-Type: text / html
- Other information such as the server type, server time, etc.

# *Some Remarks About HTTP*

- HTTP is stateless
  - No “sessions”
  - Every message is completely self-contained
  - No previous interaction is “remembered” by the protocol
  - Tradeoff between ease of implementation and ease of application development: Other functionality has to be built on top
- Implications for applications:
  - Any state information (shopping carts, user login-information) need to be encoded in every HTTP request and response!
  - Popular methods on how to maintain state:
    - Cookies (later this lecture)
    - Dynamically generate unique URL’s at the server level (later this lecture)

# *Web Data Formats*

- HTML
  - The presentation language for the Internet
- Xml
  - A self-describing, hierarchal data model
- DTD
  - Standardizing schemas for Xml
- XSLT (not covered in the book)

# *HTML: An Example*

```
<HTML>
  <HEAD></HEAD>
  <BODY>
  <h1>Barns and Nobble Internet
    Bookstore</h1>
  Our inventory:

  <h3>Science</h3>
  <b>The Character of Physical
    Law</b>
  <UL>
    <LI>Author: Richard
  Feynman</LI>
    <LI>Published 1980</LI>
    <LI>Hardcover</LI>
  </UL>

  <h3>Fiction</h3>
  <b>Waiting for the Mahatma</b>
  <UL>
    <LI>Author: R.K. Narayan</LI>
    <LI>Published 1981</LI>
  </UL>
  <b>The English Teacher</b>
  <UL>
    <LI>Author:  R.K. Narayan</LI>
    <LI>Published 1980</LI>
    <LI>Paperback</LI>
  </UL>

  </BODY>
</HTML>
```

# *HTML: A Short Introduction*

- HTML is a markup language
- Commands are tags:
  - Start tag and end tag
  - Examples:
    - <HTML> … </HTML>
    - <UL> … </UL>
- Many editors automatically generate HTML directly from your document (e.g., Microsoft Word has an “Save as html” facility)

# *HTML: Sample Commands*

- <HTML>:
- <UL>: unordered list
- <LI>: list entry
- <h1>: largest heading
- <h2>: second-level heading, <h3>, <h4> analogous
- <B>Title</B>: Bold

# *XML: An Example*

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<BOOKLIST>
  <BOOK genre="Science" format="Hardcover">
    <AUTHOR>
      <FIRSTNAME>Richard</FIRSTNAME><LASTNAME>Feynman</LASTNAME>
    </AUTHOR>
    <TITLE>The Character of Physical Law</TITLE>
    <PUBLISHED>1980</PUBLISHED>
  </BOOK>
  <BOOK genre="Fiction">
    <AUTHOR>
      <FIRSTNAME>R.K.</FIRSTNAME><LASTNAME>Narayan</LASTNAME>
    </AUTHOR>
    <TITLE>Waiting for the Mahatma</TITLE>
    <PUBLISHED>1981</PUBLISHED>
  </BOOK>
  <BOOK genre="Fiction">
    <AUTHOR>
      <FIRSTNAME>R.K.</FIRSTNAME><LASTNAME>Narayan</LASTNAME>
    </AUTHOR>
    <TITLE>The English Teacher</TITLE>
    <PUBLISHED>1980</PUBLISHED>
  </BOOK>
</BOOKLIST>
```

# *XML – The Extensible Markup Language*

- Language
  - A way of communicating information
- Markup
  - Notes or meta-data that describe your data or language
- Extensible
  - Limitless ability to define new languages or data sets

# *XML – What's The Point?*

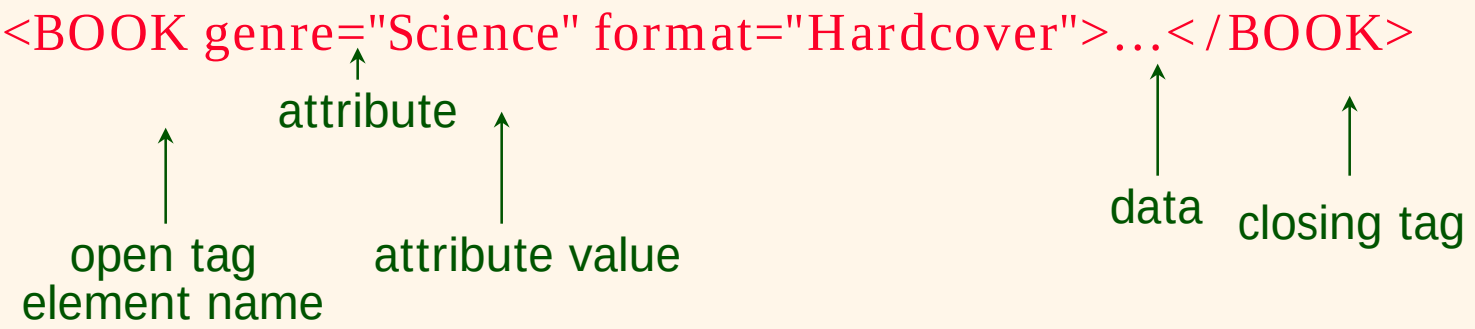
- You can include your data and a description of what the data represents
  - This is useful for defining your own language or protocol
- Example: Chemical Markup Language

```
<molecule>
        <weight>234.5</weight>
        <Spectra>...</Spectra>
        <Figures>...</Figures>
</molecule>
```

- XML design goals:
  - XML should be compatible with SGML
  - It should be easy to write XML processors
  - The design should be formal and precise

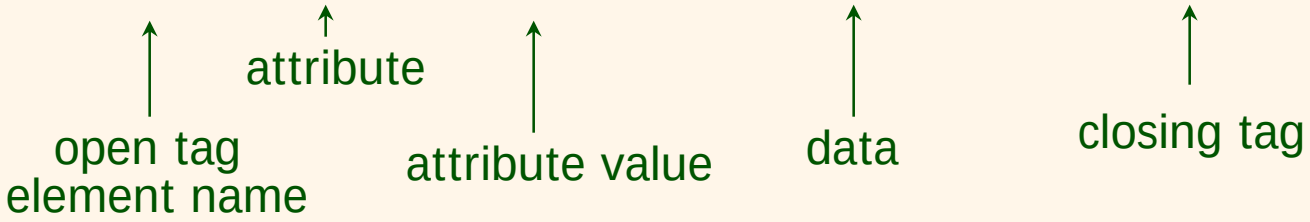
# *XML – Structure*

- XML: Confluence of SGML and HTML
- Xml looks like HTML
- Xml is a hierarchy of user-defined tags called elements with attributes and data
- Data is described by elements, elements are described by attributes



# *XML – Elements*

<BOOK genre="Science" format="Hardcover">...</BOOK>

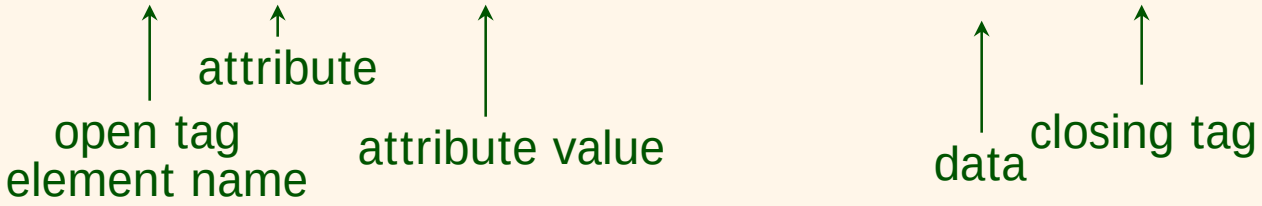



- Xml is case and space sensitive
- Element opening and closing tag names must be identical
- Opening tags: "<" + element name + ">"
- Closing tags: "</" + element name + ">"
- Empty Elements have no data and no closing tag:
  - They begin with a "<" and end with a "/>"

  <BOOK/>

# *XML – Attributes*

<BOOK genre="Science" format="Hardcover">…</BOOK>




- Attributes provide additional information for element tags.
- There can be zero or more attributes in every element; each one has the the form:

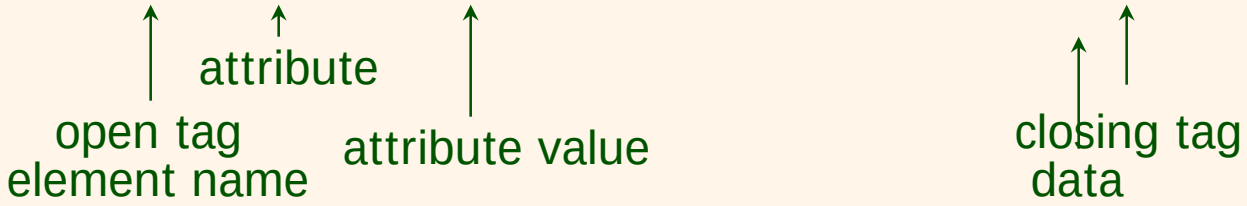
  *attribute_name='attribute_value'*

  - There is no space between the name and the "='"
  - Attribute values must be surrounded by " or ' characters

- Multiple attributes are separated by white space (one or more spaces or tabs).

# *XML – Data and Comments*

<BOOK genre="Science" format="Hardcover">…</BOOK>




- Xml data is any information between an opening and closing tag
- Xml data must not contain the '<' or '>' characters

- Comments:
  <!- comment ->

# *XML – Nesting & Hierarchy*

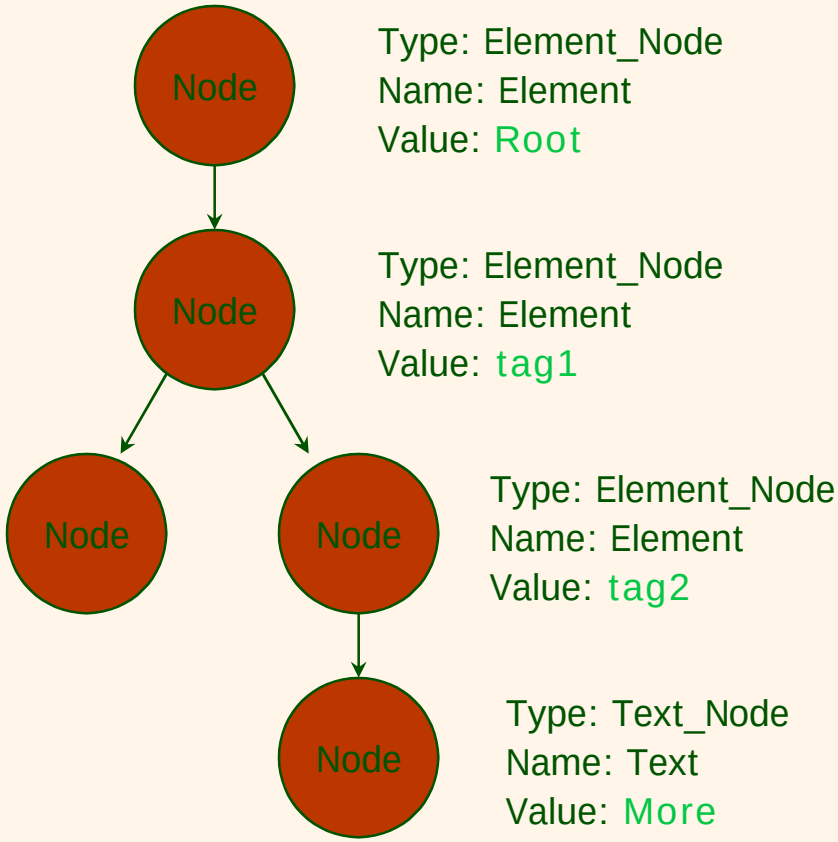
- Xml tags can be nested in a tree hierarchy
- Xml documents can have only one root tag
- Between an opening and closing tag you can insert:
    1. Data
    2. More Elements
    3. A combination of data and elements

```
<root>
  <tag1>
    Some Text
    <tag2>More</tag2>
  </tag1>
</root>
```

# *Xml – Storage*

- Storage is done just like an n-ary tree (DOM)

```
<root>
   <tag1>
      Some Text
      <tag2>More</tag2>
   </tag1>
</root>
```



# *DTD – Document Type Definition*

- A DTD is a schema for Xml data
- Xml protocols and languages can be standardized with DTD files
- A DTD says what elements and attributes are required or optional
  - Defines the formal structure of the language

# *DTD – An Example*

```
<?xml version='1.0'?>
<!ELEMENT Basket (Cherry+, (Apple | Orange)*) >
    <!ELEMENT Cherry EMPTY>
        <!ATTLIST Cherry flavor CDATA #REQUIRED>
    <!ELEMENT Apple EMPTY>
        <!ATTLIST Apple color CDATA #REQUIRED>
    <!ELEMENT Orange EMPTY>
        <!ATTLIST Orange location 'Florida'>
```

---

```
<Basket>
    <Cherry flavor='good'/>
    <Apple color='red'/>
    <Apple color='green'/>
</Basket>
```

```
<Basket>
    <Apple/>
    <Cherry flavor='good'/>
    <Orange/>
</Basket>
```

# *DTD - !ELEMENT*

<!ELEMENT Basket (Cherry+, (Apple | Orange)*) >

Name Children

- !ELEMENT declares an element name, and what children elements it should have
- Content types:
  - Other elements
  - #PCDATA (parsed character data)
  - EMPTY (no content)
  - ANY (no checking inside this structure)
  - A regular expression

# *DTD - !ELEMENT (Contd.)*

- A regular expression has the following structure:
  - $exp_1$, $exp_2$, $exp_3$, …, $exp_k$: A list of regular expressions
  - exp*: An optional expression with zero or more occurrences
  - exp+: An optional expression with one or more occurrences
  - $exp_1$ | $exp_2$ | … | $exp_k$: A disjunction of expressions

# *DTD - !ATTLIST*

<!ATTLIST Cherry flavor CDATA #REQUIRED>

Element Attribute Type Flag

<!ATTLIST Orange location CDATA #REQUIRED
color 'orange'>

- !ATTLIST defines a list of attributes for an element
- Attributes can be of different types, can be required or not required, and they can have default values.

# *DTD – Well-Formed and Valid*

```
<?xml version='1.0'?>
<!ELEMENT Basket (Cherry+)>
    <!ELEMENT Cherry EMPTY>
        <!ATTLIST Cherry flavor CDATA #REQUIRED>
```

---

Not Well-Formed

```
<basket>
  <Cherry flavor=good>
</Basket>
```

Well-Formed but Invalid

```
<Job>
  <Location>Home</Location>
</Job>
```

Well-Formed and Valid

```
<Basket>
  <Cherry flavor='good'/>
</Basket>
```

# *XML and DTDs*

- More and more standardized DTDs will be developed
  - MathML
  - Chemical Markup Language
- Allows light-weight exchange of data with the same semantics

- Sophisticated query languages for XML are available:
  - Xquery
  - XPath