Databases and Internet Applications

Part 1

Chapter 7.1-7.5

© Mark Parisi, Permission required for use.

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://compgen.unc.edu/Courses
  mailto:webmaster@bookstore.com
  ftp://ftp.sanger.ac.uk/pub/

Structure of URIs

http://www.cs.unc.edu/Courses/comp521-f10/

- URI has three parts:
  - Name of the protocol used to access the resource (http)
  - Name of the host computer (www.cs.unc.edu)
  - Name of the resource (Courses/comp521-f10/)
    (in this case the default document "index.php")
- URLs are a subset of URIs
  - URL (*Universal Resource Locator*)
  - The distinction is not important for our purposes

Hypertext Transfer Protocol (HTTP)

- **What is a communication protocol?**
  - Set of standards that defines the structure of messages
  - Examples: TCP, IP, HTTP, FTP

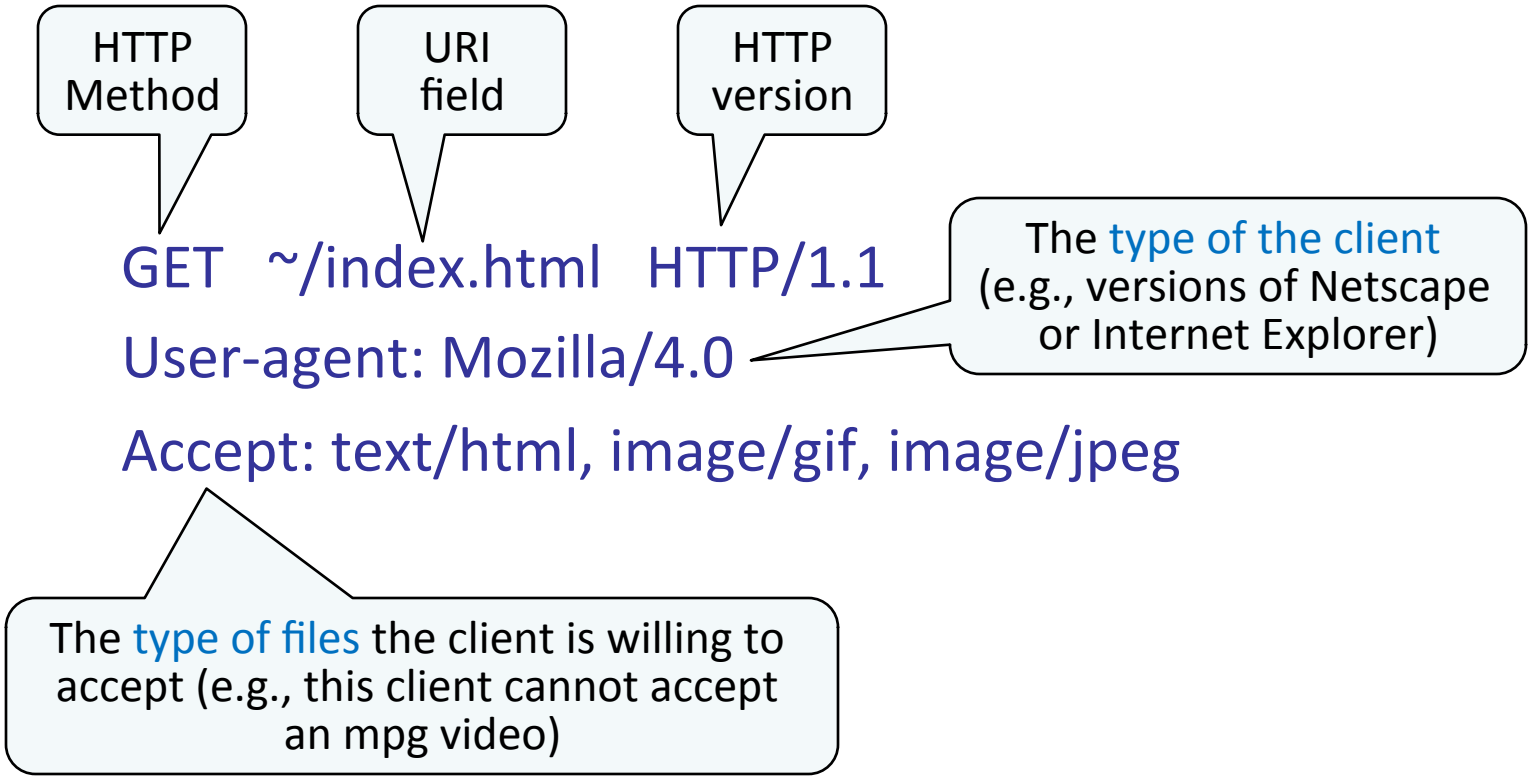
- **What happens if you click on**

  http://compgen.unc.edu/Courses ?

  1. Client (web browser) sends an *HTTP request* to server (compgen.unc.edu)
  2. Server replies with an *HTTP response*

HTTP Requests

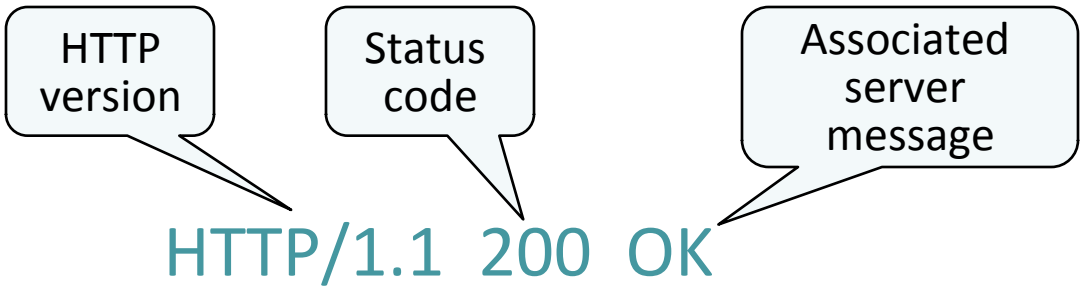
HTTP Requests consists of several lines of ASCII text, with an empty line at the end.



HTTP Responses

- The server retrieves the page "index.html" and uses it to assemble the HTTP response message
- The HTTP response message has three parts:
  - status line
  - several header lines
  - body of the message
    (which contains the requested object)

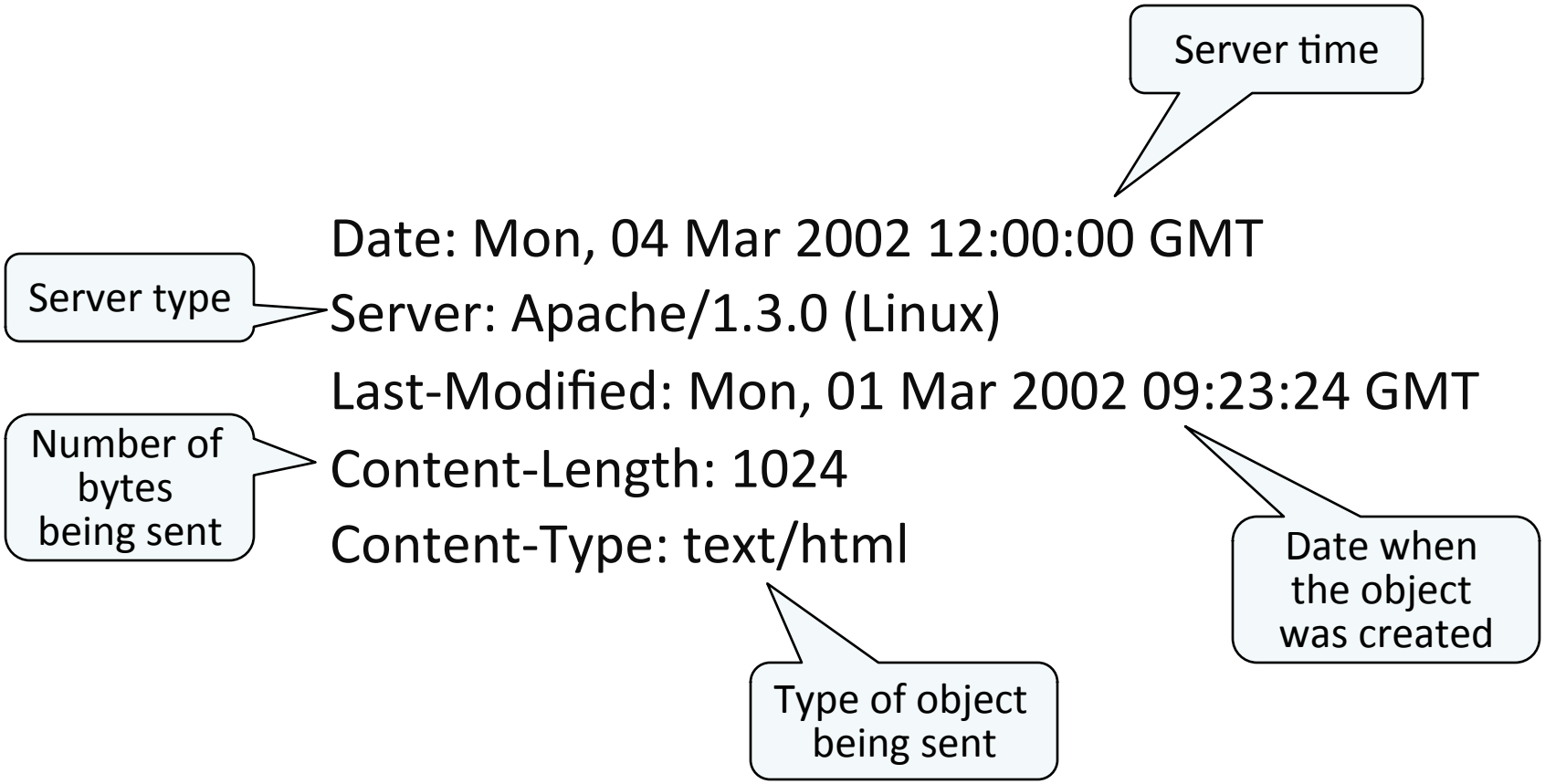
HTTP Response: Status Line



Common status codes and associated messages:

- 200 OK: The request succeeded and the object is in the body of the message
- 400 Bad Request: The request could not be fulfilled
- 404 Not Found: The requested object does not exist
- 505 HTTP Version Not Supported: The protocol version used by the client is not supported by the server

HTTP Responses: Header Lines



HTTP Response: Body

<HTML> <HEAD></HEAD>

<BODY>

<h1>Barns and Nobble Internet Bookstore</h1>

Our inventory:

<h3>Science</h3>

The Character of Physical Law

…

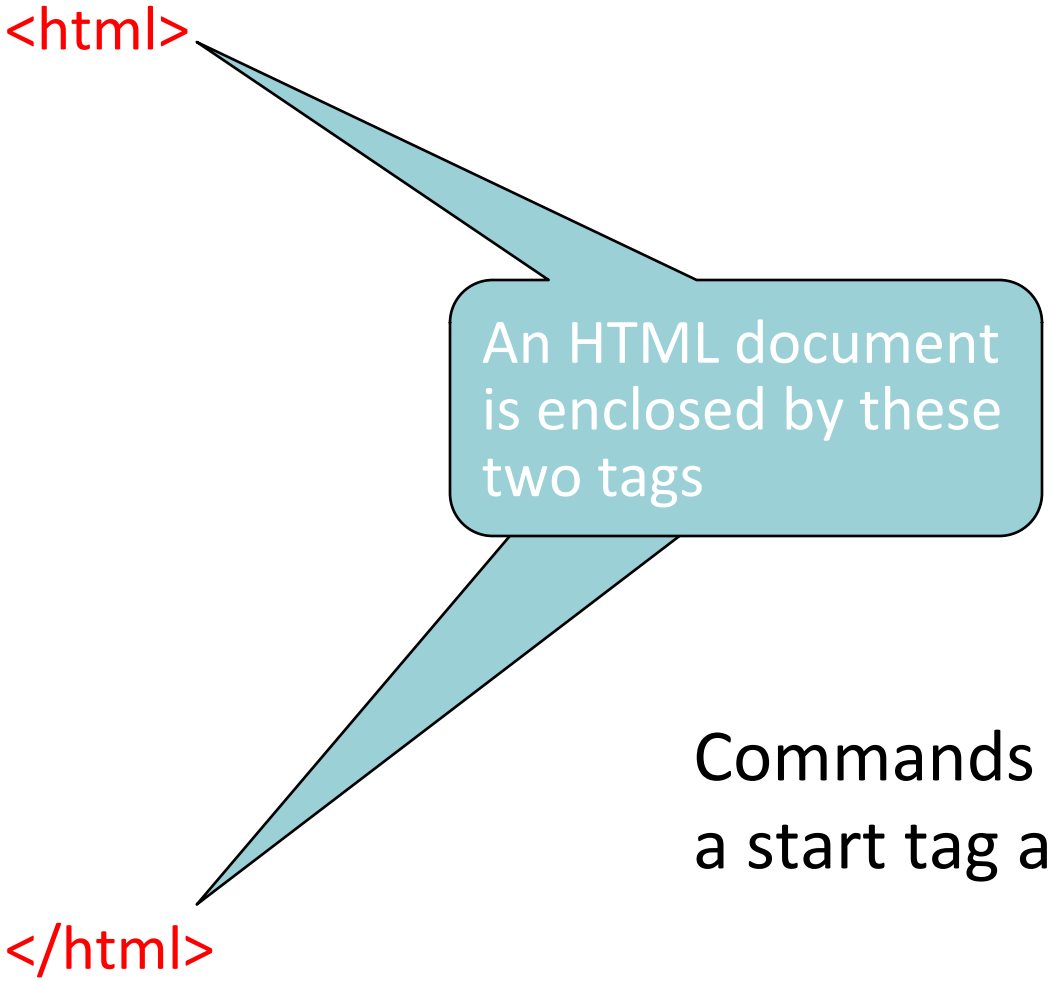
HTTP is Stateless

- HTTP is stateless
  - No "sessions"
  - Every message is self-contained
  - No previous interaction are "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 (more on them next lecture)
    - Generate unique URL's dynamically at the server level

Web Data Formats

- HTML: HyperText Markup Language
  - The presentation language for the Internet
- XML: eXtensible Markup Language
  - A self-describing, hierarchal data model
- DTD: Document Type Declarations
  - Standardizing rules/schemas for XML
- CSS: Cascading Style Sheets
  - Page layout and formatting hints
- XSL: eXtensible Style Language
  - not covered

HTML: Basic Contructs



Commands in HTML consist of a start tag and an end tag

HTML: Basic Constructs

```
<html>
<head>
…
</head>
```

The head section contains information about the page including the title, author, etc.

```
</html>
```

HTML: Basic Contructs

```
<html>
<head>
…
</head>
<body>
```

The body section contains the parts of the web page the browser will display: text, images, links, etc.

```
</body>
</html>
```

HTML: Basic Contructs

```
<html>
<head>
…
</head>
<body>
<h1>Section 1</h1>


</body>
</html>
```

There are six levels of section headers: h1 through h6

HTML: Basic Contructs

```
<html>
<head>
…
</head>
<body>
<h1>Section 1</h1>
  <ul>
    <li>This is the first item</li>
  </ul>
</body>
</html>
```

- Ordered List

```
<ol>
  <li>Coffee</li>
  <li>Tea</li>
</ol>
```

- Definition List

```
<dl>
  <dt>Coffee</dt>
  <dd> …  </dd>
  <dt>Tea</dt>
  <dd> … </dd>
</dl>
```

This is an unordered list

HTML: Basic Contructs

```
<html>
<head>
…
</head>
<body>
<h1>Section 1</h1>
  <ul>
    <li> This is the <b>first</b> item <\li>
  </ul>
</body>
</html>
```

Display "first" in boldface

HTML: An Example

```
<html>
  <head></head>
  <body>
  <h1>Barns and Nobble Internet Bookstore</h1>

  <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>Oliver Twist</b>
    <ul>
      <li>Author: Charles Dickens</li>
      <li>Published 2002</li>
    </ul>

  <b>Pride and Prejudice</b>
    <ul>
      <li>Author:  Jane Austen</li>
      <li>Published 1983</li>
      <li>Paperback</li>
    </ul>
  </body>
</html>
```

HTML: Summary

- HTML is a markup language for describing content
- Commands are tag enclosures:
  - 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 Web Page" facility)

HTML vs XML

- HTML
  - Supports a fixed set of predefined tags
  - Not enough tags to describe the structures of the content of specific applications (e.g., what part of the content are names?, etc.)
- XML
  - Allows users to define new tags to structure any type of data or document
  - It makes database systems more tightly integrated into Web applications

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 Elements

- Elements are also called tags
- Elements are primary building blocks of an XML document
- Each element of a user-defined type ELM is enclosed by <ELM> and <\ELM>

  Example: <FIRSTNAME>Jessica</FIRSTNAME>
- Elements can be nested (forming a tree structure)

  Example:
```
<BOOK>
    <AUTHOR>
        <FIRSTNAME>Charles</FIRSTNAME>
        <LASTNAME>Dickens</LASTNAME>
    <AUTHOR>
</BOOK>
```
- EXML elements are case sensitive: BOOK ≠ Book

XML Elements /w Attributes

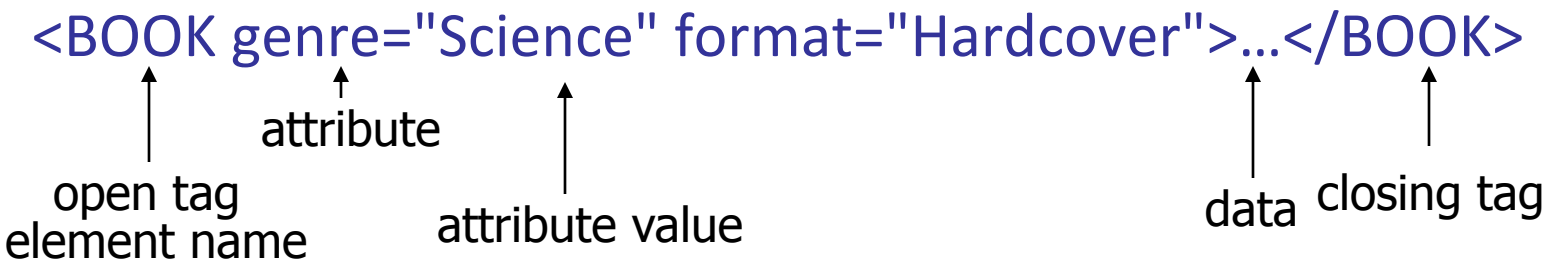
- An Element can have descriptive attributes
- The values of attributes are set inside the start tag of the element
- All attribute values must be enclosed in quotes

Example:

```
<BOOK GENRE="Fiction" FORMAT="Hardcover">
    <AUTHOR>
        <FIRSTNAME>Charles</FIRSTNAME>
        <LASTNAME>Dickens</LASTNAME>
    <AUTHOR>
</BOOK>
```

XML – Structure

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



XML Entity References

- XML data can't contain the reserved characters
- Whenever an *entity reference* appears in the document, it is textually replaced by its content
- Format: < "lt" is an entity reference for the character "<"

Reserved Characters	Entity References
<	lt
>	gt
&	amp
"	quot
'	apos

'1<5'

'1 < 5'

XML: Comments

- Comments start with <!-- and end with -->
- Comments can contain arbitrary text except the string --
- Example: <!- comment ->

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>Charles</FIRSTNAME><LASTNAME>Dickens</
    LASTNAME>
    </AUTHOR>
    <TITLE>Oliver Twist</TITLE>
    <PUBLISHED>2002</PUBLISHED>
  </BOOK>
  <BOOK genre="Fiction">
    <AUTHOR>
      <FIRSTNAME>Jane</FIRSTNAME><LASTNAME>Austen</
    LASTNAME>
    </AUTHOR>
    <TITLE>Pride and Prejudice</TITLE>
    <PUBLISHED>1983</PUBLISHED>
  </BOOK>
</BOOKLIST>
```

Should begin with an XML declaration

A root element contains all other elements

All elements must be properly nested

XML – What's The Point?

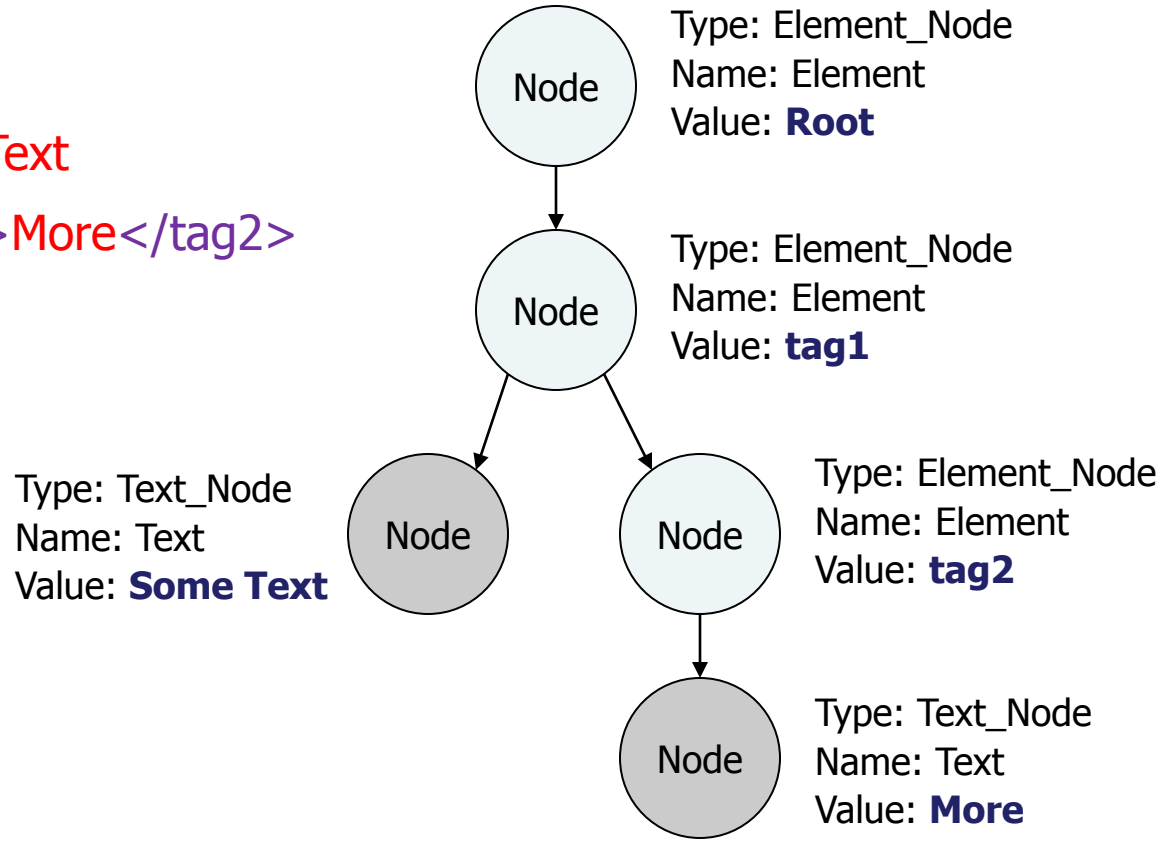
- You can include your data and a description of what the data represents
  - This is useful for defining your own language or protocol
- Example: Chemical Markup Language

```
<molecule>
        <name>Methionine</name>
        <formula>C<sub>5</sub>H<sub>11
        </sub>NO<sub>2</sub>S</formula>
        <weight>149.2</weight>
        <spectra>…</spectra>
        <figures>…</figures>
</molecule>
```

XML – Storage

Storage is just an n-ary tree

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



DTD – Document Type Definition

- Unlike HTML, XML has user-defined elements (tags)
  - → the user needs to describe these elements
- DTD is a set of rules that defines the user-defined elements for an XML document
  - → DTD is the schema for the XML data
  - → DTD says what elements and attributes are required or optional (the formal structure of the language)
- A document is valid if it is structured according to the rules set by the DTD

DTD Structure

```
<!DOCTYPE BOOKLIST [

]>
```

A DTD is enclosed in:
<!DOCTYPE name [DTDdeclaration]>

DTD Structure

```
<!DOCTYPE BOOKLIST [
<!ELEMENT BOOKLIST (BOOK)*>
]>
```

- A DTD starts with the root element
- The root element BOOKLIST consists of zero or more BOOK elements
  - * : zero or more occurrences
  - + : one or more occurrences
  - ? : zero or one occurrence

DTD Structure

```
<!DOCTYPE BOOKLIST [
<!ELEMENT BOOKLIST (BOOK)*>
   <!ELEMENT BOOK (AUTHOR,TITLE,PUBLISHED?)>
]>
```

- An element can have nested elements
- This rule says that a BOOK element contains an AUTHOR element, a TITLE element, and an optional PUBLISHED element

DTD Structure

```
<!DOCTYPE BOOKLIST [
<!ELEMENT BOOKLIST (BOOK)*>
    <!ELEMENT BOOK (AUTHOR,TITLE,PUBLISHED?)>
        <!ELEMENT AUTHOR (FIRSTNAME,LASTNAME)>
            <!ELEMENT FIRSTNAME (#PCDATA)>
            <!ELEMENT LASTNAME (#PCDATA)>
]>
```

- Instead of containing other elements, an element can contain actual text
  - #PCDATA indicates character data
  - EMPTY indicates the element has no content
  - ANY indicates that any content is permitted. No checking inside this structure (avoided whenever possible)

DTD Structure

```
<!DOCTYPE BOOKLIST [
<!ELEMENT BOOKLIST (BOOK)*>
    <!ELEMENT BOOK (AUTHOR,TITLE,PUBLISHED?)>
        <!ELEMENT AUTHOR (FIRSTNAME,LASTNAME)>
            <!ELEMENT FIRSTNAME (#PCDATA)>
            <!ELEMENT LASTNAME (#PCDATA)>
        <!ELEMENT TITLE (#PCDATA)>
        <!ELEMENT PUBLISHED (#PCDATA)>
    <!ATTLIST BOOK GENRE (Science|Fiction) #REQUIRED>
    <!ATTLIST BOOK FORMAT (Paperback|Hardcover) "Paperback">
]>
```

DTD Structure

```
<!DOCTYPE BOOKLIS
<!ELEMENT BOOKLIS
   <!ELEMENT BOO
         <!ELEMENT
               <!EL
               <!EL
         <!ELEMENT TI
         <!ELEMENT PUBLISHED (#PCDATA)>
   <!ATTLIST BOOK GENRE (Science|Fiction) #REQUIRED>
   <!ATTLIST BOOK FORMAT (Paperback|Hardcover) "Paperback">
]>
```

- Attributes of elements are declared outside the element
- The BOOK element has two attributes
  - The GENRE attribute is required and can have the value 'Science' or 'Fiction'
  - The FORMAT attribute can have the value 'Paperback' or 'Hardcover', and 'Paperback' is the default value
  - #REQUIRED is the default option

The whole DTD

```
<!DOCTYPE BOOKLIST [
<!ELEMENT BOOKLIST (BOOK)*>
    <!ELEMENT BOOK (AUTHOR,TITLE,PUBLISHED?)>
        <!ELEMENT AUTHOR (FIRSTNAME,LASTNAME)>
            <!ELEMENT FIRSTNAME (#PCDATA)>
            <!ELEMENT LASTNAME (#PCDATA)>
        <!ELEMENT TITLE (#PCDATA)>
        <!ELEMENT PUBLISHED (#PCDATA)>
    <!ATTLIST BOOK GENRE (Science|Fiction) #REQUIRED>
    <!ATTLIST BOOK FORMAT (Paperback|Hardcover) “Paperback”>
]>
```

Five Possible Content Types

<!ELEMENT (contentType)>

- Other elements
- Special symbol #PCDATA, EMPTY, or ANY
- A regular expression constructed from the preceding four choices
  - exp1, exp2, exp3: An ordered list of regular expressions
  - Exp*: An optional expression (zero or more occurrence)
  - Exp?: An optional expression (zero or one occurrences)
  - Exp+: A mandatory expression (one or more occurrences)
  - Exp1 | exp2: exp1 or exp2

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' >
```

Apple's color is required. Cherry should go first.

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

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

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 (domain-specific) DTDs will be developed
  - MathML (Mathematical Markup Language)
  - Chemical Markup Language
- Enable seamless data exchange among heterogeneous sources
- Sophisticated query languages for XML are available:
  - Xquery
  - XPath

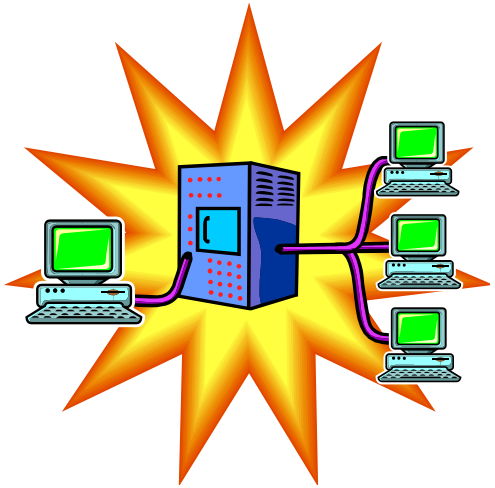
Web Application Architectures

- Model encompassing most web-based apps
- Three separate types of functionality:
  - Data management (Model)
  - Application logic (Controller)
  - Presentation (View)
- The system architecture determines whether these three components reside on a single system (tier) or are distributed across several tiers

Single-Tier Architectures

All functionality combined into a single tier, usually on a mainframe

- User access through dumb terminals



Advantages:

- Easy maintenance and administration

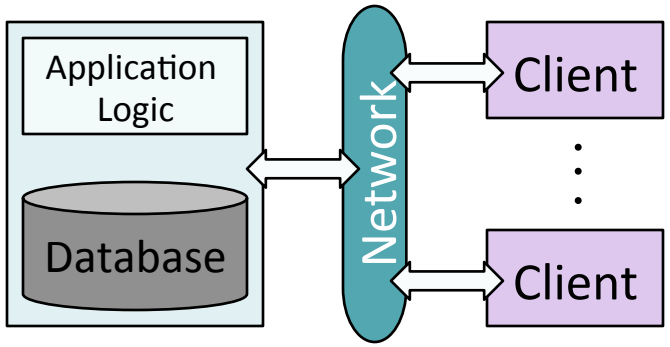
Disadvantages:

- Today, users expect graphical user interfaces.
- Centralized computation of all of them is too much for a central system

Client-Server Architectures

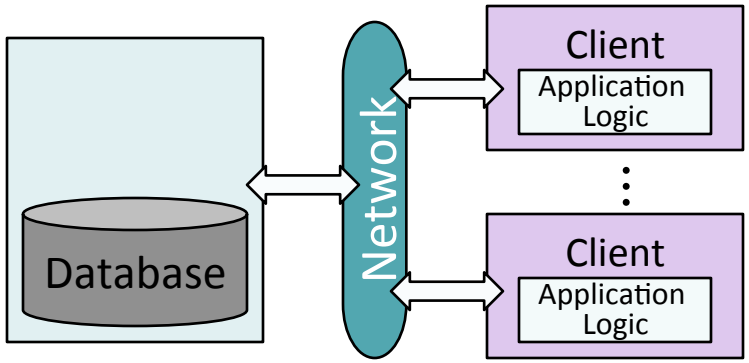
Work division: Thin client

- Client implements only the graphical user interface
- Server implements business logic and data management



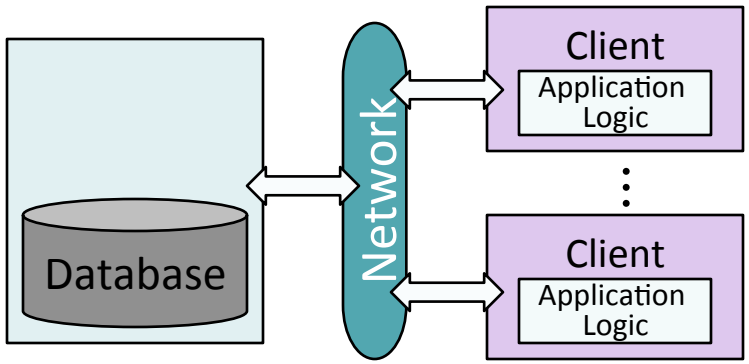
❖ Work division: Thick client

- Client implements both the graphical user interface and the business logic
- Server implements data management

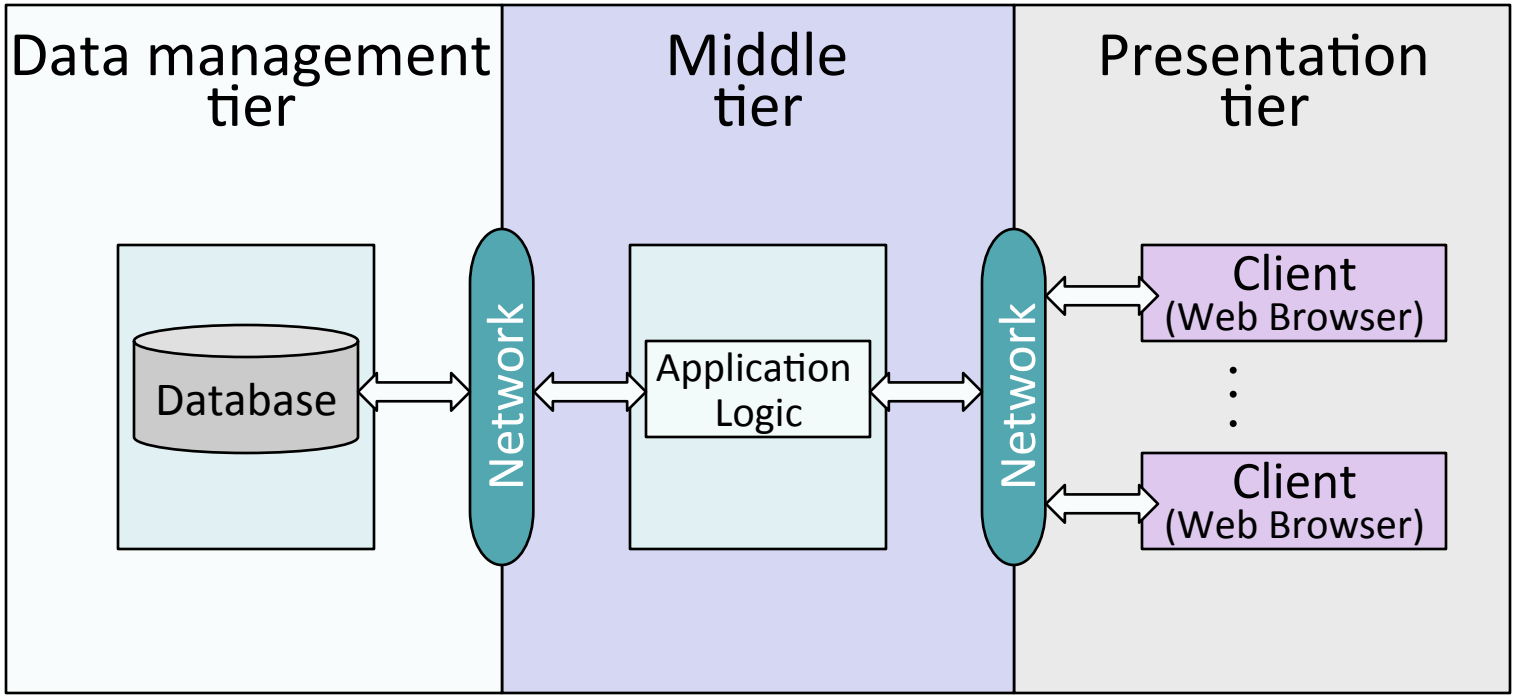


Disadvantages of Thick Clients

- No central place to update the business logic
- Security issues: Server needs to trust clients
  - Clients need to leave server database in consistent state
  - One possibility: Encapsulate all database access into stored procedures
- Does not scale to more than several 100s of clients
  - Large data transfer between server and client
  - More than one server creates a problem: x clients, y servers: x*y connections



Three-Tier Architecture



The Three Layers

Presentation tier

- Primary interface to the user
- Needs to adapt to different display devices (PC, PDA, cell phone, voice access?)

Middle tier

- Implements business logic (implements complex actions, maintains state between different steps of a workflow)
- Accesses different data management systems

Data management tier

- One or more standard database management systems

Example 1: Airline reservations

- Database System
  - Airline info, available seats, customer info, etc.
- Application Server
  - Logic to make reservations, cancel reservations, add new airlines, etc.
- Client Program
  - Log in different users, display forms and human-readable output

Example 2: Course Enrollment

- Database System
  - Student info, course info, instructor info, course availability, pre-requisites, etc.
- Application Server
  - Logic to add a course, drop a course, create a new course, etc.
- Client Program
  - Log in different users (students, staff, faculty), display forms and human-readable output

Technologies

Client Program *(Web Browser)*	*HTML* *Javascript* *CSS*
Application Server *(Tomcat, Php, Apache)*	*JSP* *Servlets* *CGI*
Database System *(DB2)*	*SQL* *XML* *Stored Procedures*

Advantages of the Three-Tier Architecture

- Heterogeneous systems
  - Tiers can be independently maintained, modified, and replaced
- Thin-"ish" clients
  - Clients only need enough computation power for the presentation layer and simple application logic (e.g. form checking) (web browsers)
- Integrated data access
  - Several database systems can be handled transparently at the middle tier
  - Central management of connections
- Scalability
  - Replication at middle tier permits scalability of business logic
- Software development
  - Code for business logic is centralized
  - Interaction between tiers through well-defined APIs: Can reuse standard components at each tier