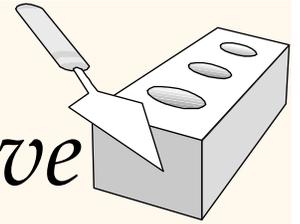


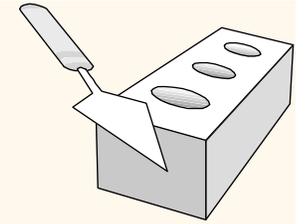
Cursor that gets names of sailors who've reserved a red boat, in alphabetical order



```
EXEC SQL DECLARE sinfo CURSOR FOR
  SELECT S.sname
  FROM Sailors S, Boats B, Reserves R
  WHERE S.sid=R.sid AND R.bid=B.bid AND B.color='red'
  ORDER BY S.sname
```

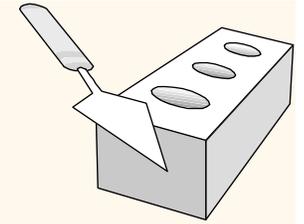
Most implementations differ from SQL standard:

- ❖ Can use order by without cursor.
- ❖ Can have many expressions for order criteria.
- ❖ Including input columns or expressions that depend on input columns.



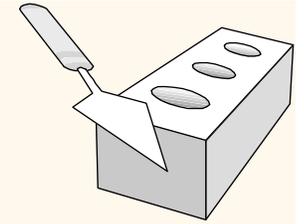
Internet Applications

Chapter 7



Lecture Overview

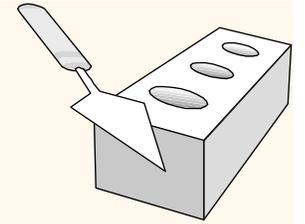
- ❖ Internet Concepts: Data Flow from Client to DBMS
- ❖ Introduction to three-tier architectures
- ❖ Web data formats
 - HTML, XML, DTDs
- ❖ The presentation layer
 - HTML forms; HTTP Get and POST, URL encoding; Javascript; Stylesheets. XSLT
- ❖ The middle tier
 - CGI, application servers, Servlets,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



Lecture Overview

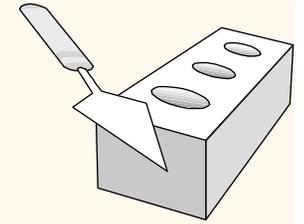
- ❖ Internet Concepts
- ❖ Web data formats
 - HTML, XML, DTDs
- ❖ Introduction to three-tier architectures
- ❖ The presentation layer
 - HTML forms; HTTP Get and POST, URL encoding; Javascript; Stylesheets. XSLT
- ❖ The middle tier
 - CGI, application servers, Servlets, JavaServerPages, passing arguments,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

Components of Data-Intensive Systems

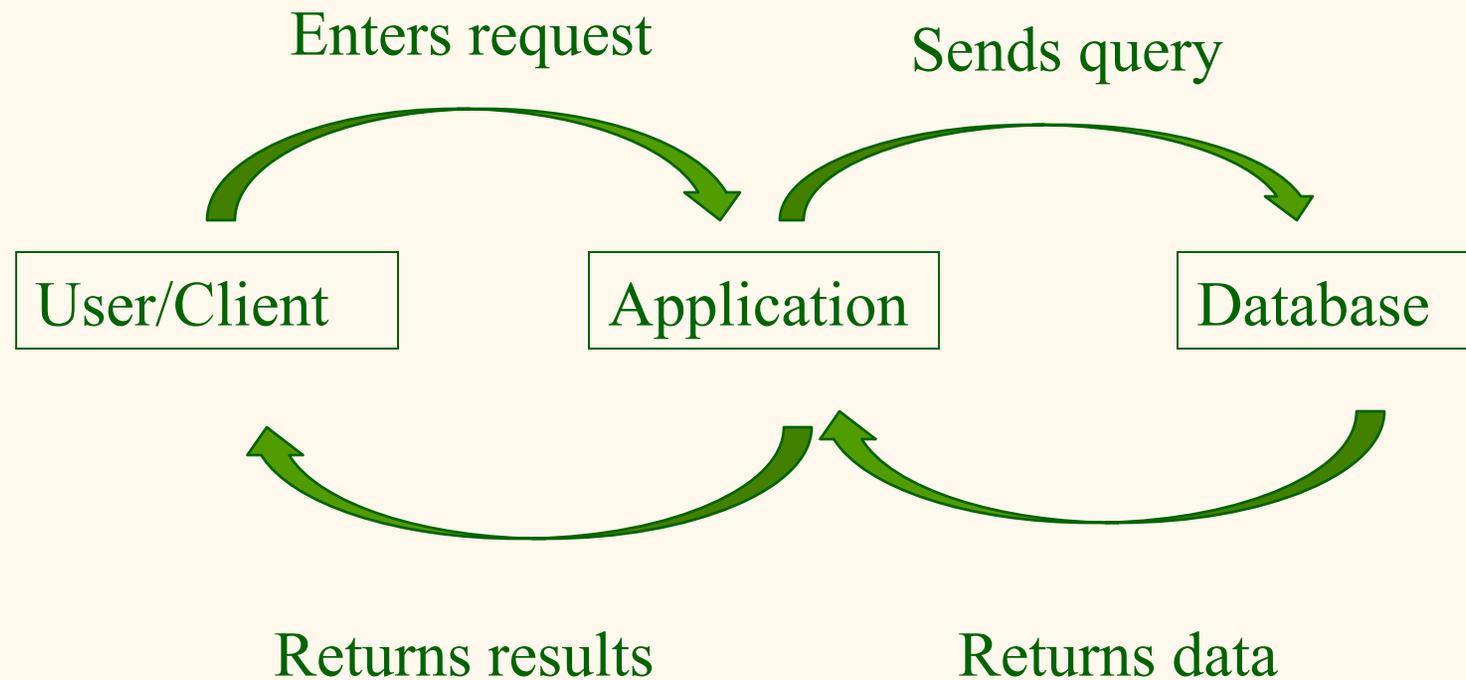


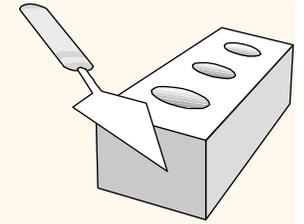
Three separate types of functionality:

- ❖ Data management
 - ❖ Application logic
 - ❖ Presentation
-
- ❖ The system architecture determines whether these three components reside on a single system (“tier”) or are distributed across several tiers

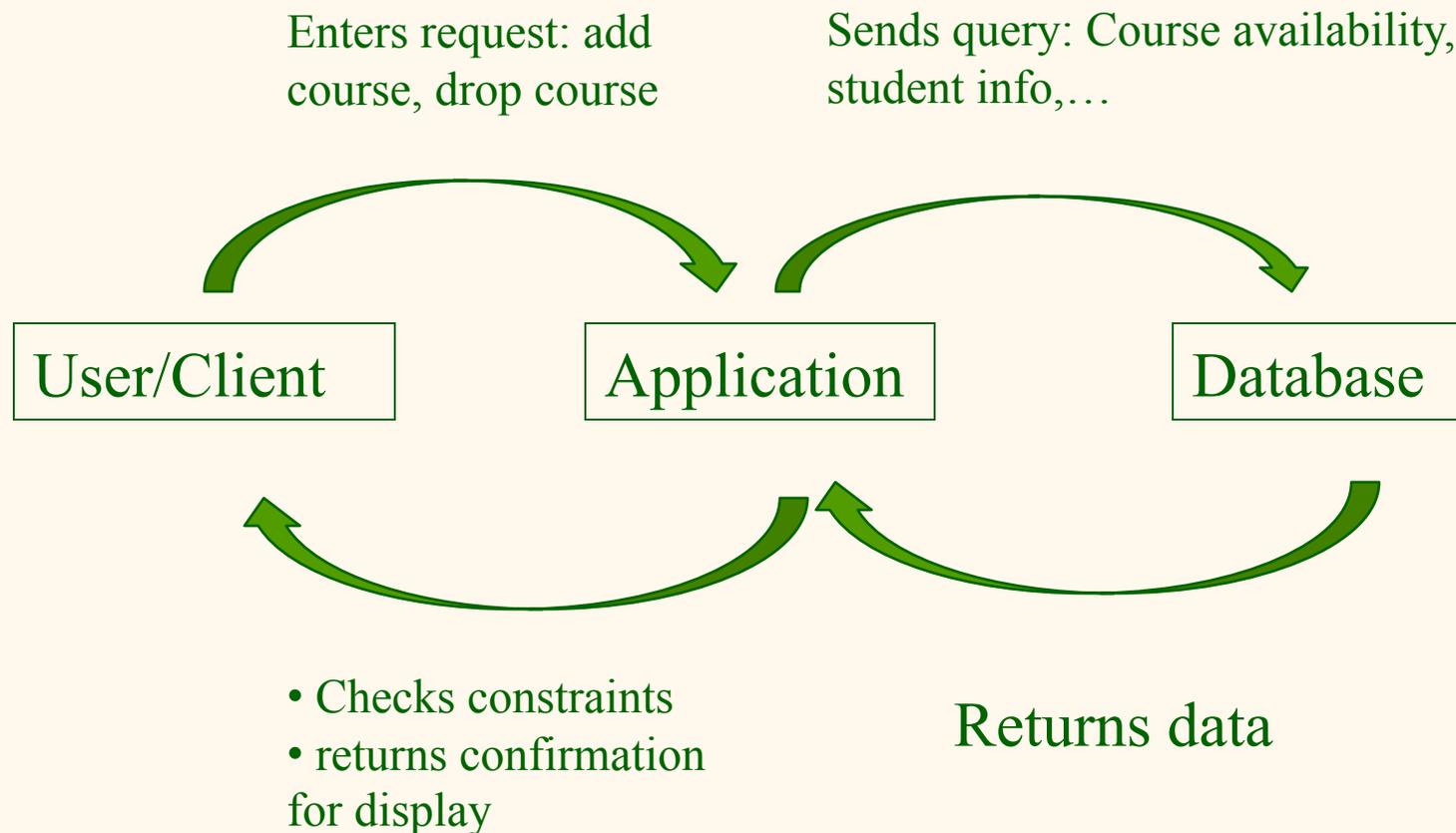


Process/Data Flow in Network

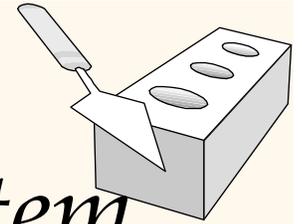




Example: Course Enrolment

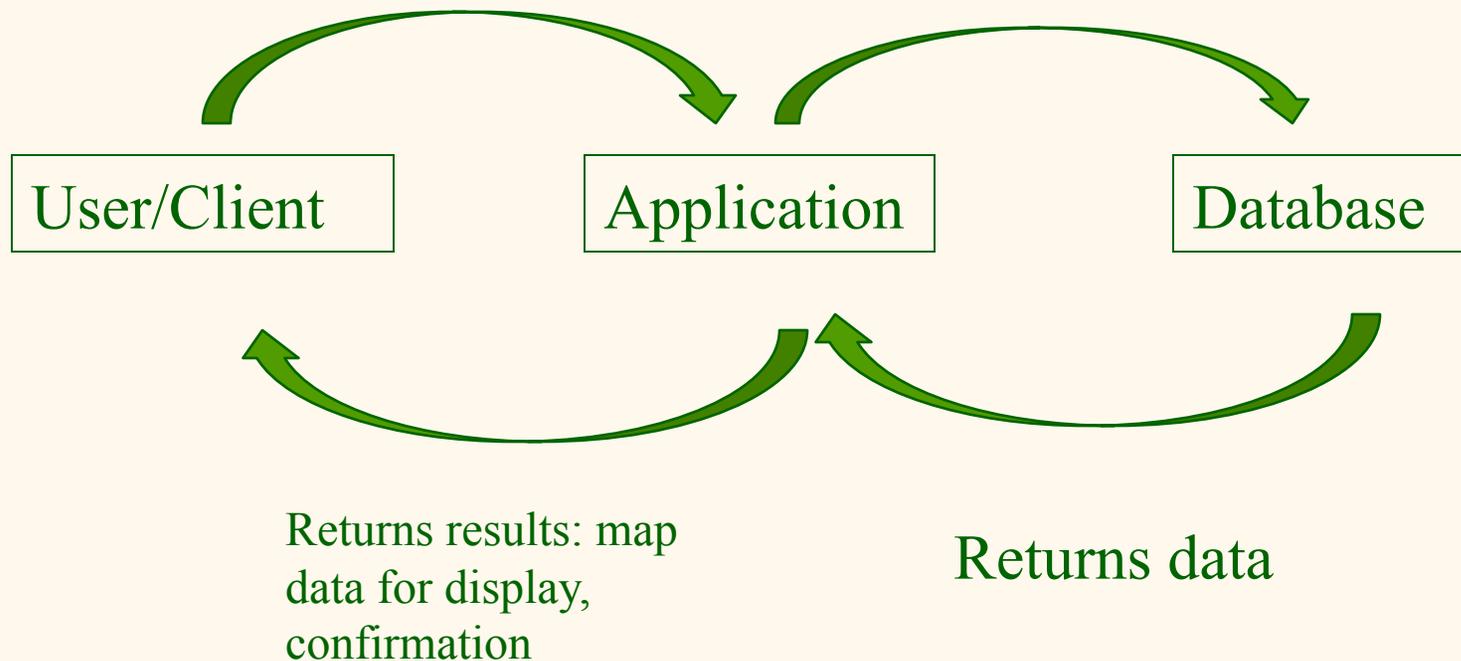


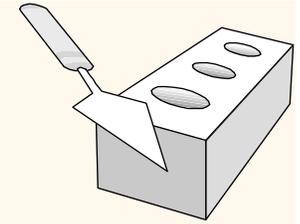
Example: Airline Reservation System



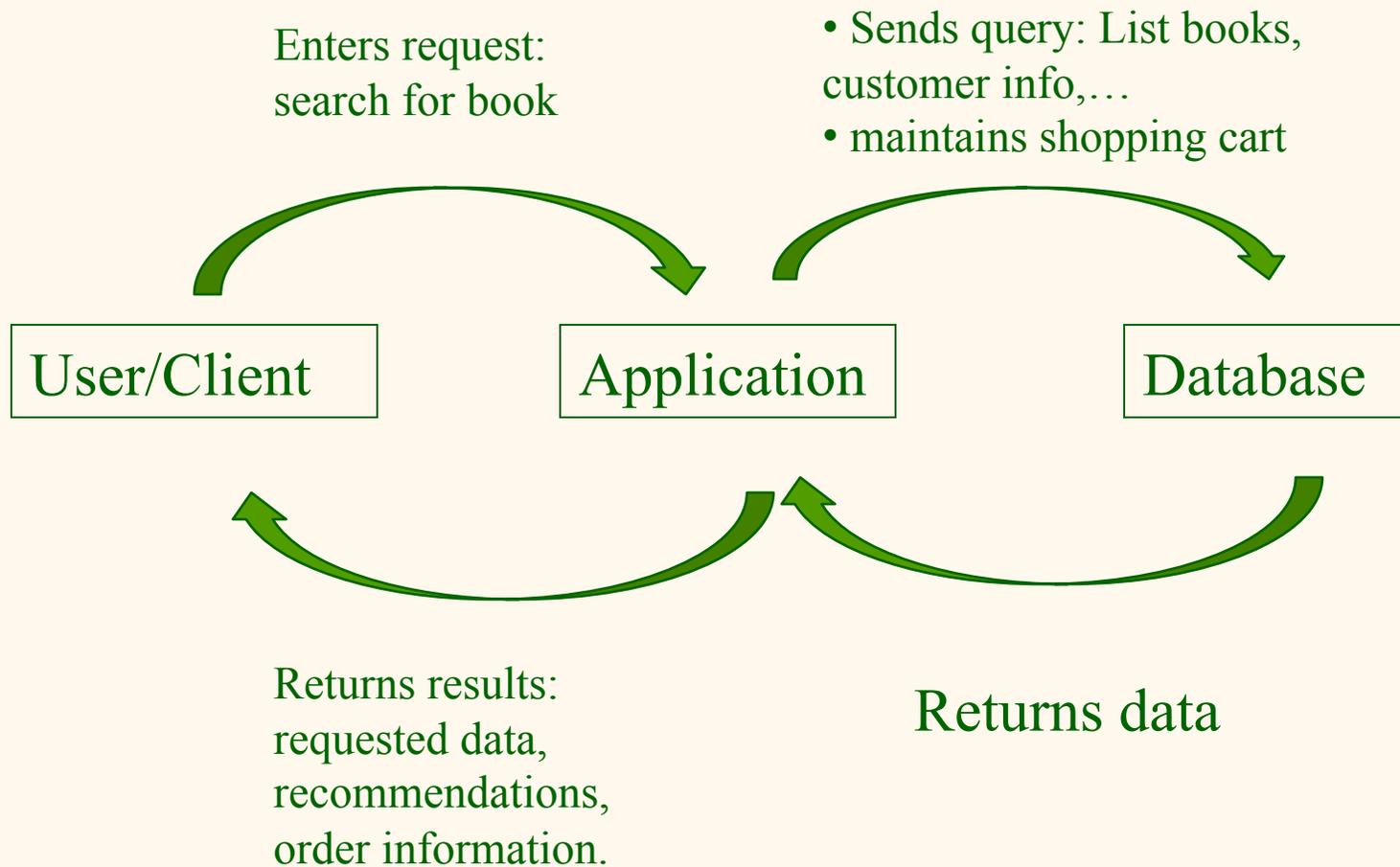
Enters request: log in, show seat map

Sends query: Airline info, available seats, customer info...

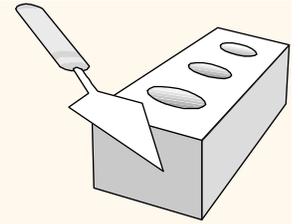




Example: Book Order System

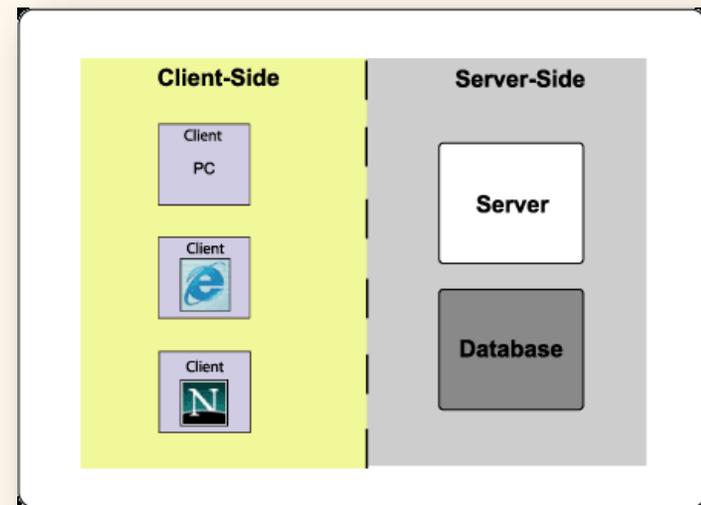


Client-Server Architectures



Work division: Thin client

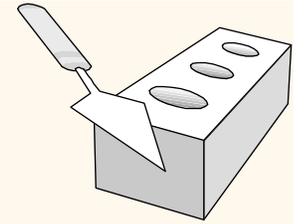
- Client implements only the graphical user interface
- Server implements business logic and data management.
- Development supported by Visual Studio, Sybase Powerbuilder.



❖ Work division: Thick client

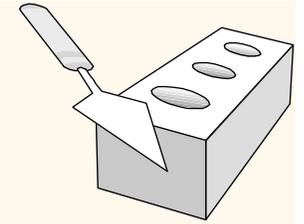
- Client implements both the graphical user interface and the business logic
- Server implements data management

Discussion Question



- ❖ What are advantages of thin clients?
- ❖ What are disadvantages of thin clients?

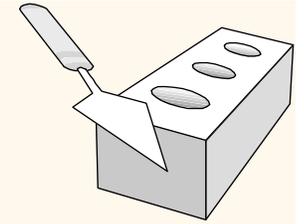
- ❖ What are advantages of thick clients?
- ❖ What are disadvantages of thick clients?



Client-Server Architectures

Disadvantages of thick clients

- No central place to update the business logic
- Security issues: Server needs to trust clients
 - Access control and authentication needs to be managed at the server
 - 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



The Three-Tier Architecture

Presentation tier

Client Program (Web Browser)

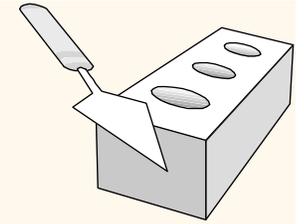
Middle tier

Application Server

Data management
tier

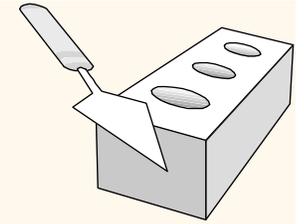
Database System

C
l
o
u
d



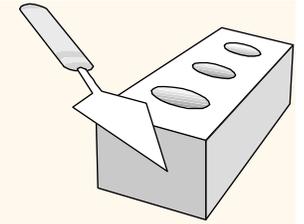
Example 1: Airline reservations

- ❖ Build a system for making airline reservations
- ❖ What is done in the different tiers?
- ❖ 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

- ❖ Build a system using which students can enroll in courses
- ❖ 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



The Three Layers

Presentation tier

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

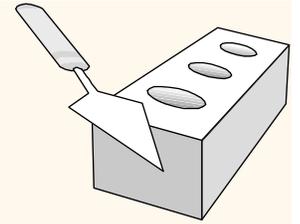
Middle tier

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

Data management tier

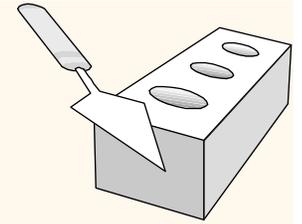
- One or more standard database management systems

Advantages of the Three-Tier Architecture



- ❖ Heterogeneous systems
 - Tiers can be independently maintained, modified, and replaced
- ❖ Thin clients
 - Only presentation layer at clients (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

Technologies



Client Program
(*Web Browser*)

HTML
Javascript

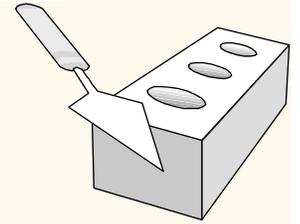
Application Server
(*Tomcat, Apache*)

JSP
Servlets
Cookies
CGI

Database System
(*DB2*)

XML
Stored Procedures

Overview of the Presentation Tier

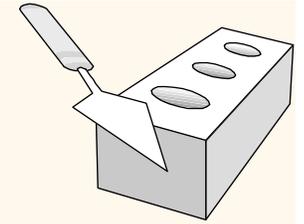


❖ Functionality of the presentation tier

- Primary interface to the user
- Needs to adapt to different display devices (PC, PDA, cell phone, voice access?)
- Simple functionality, such as field validity checking

❖ We will cover:

- Http protocol.
- XML, HTML Forms: How to pass data to the middle tier
- JavaScript: Simple functionality at the presentation tier.



Uniform Resource Identifiers

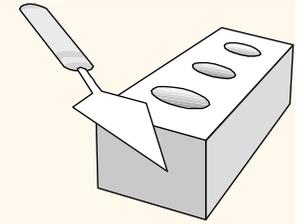
❖ Uniform naming schema to identify *resources* on the Internet

❖ 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>

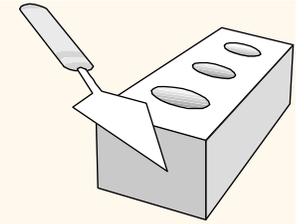


Structure of URIs

<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](http://www.cs.wisc.edu/~dbbook/index.html))

- ❖ URLs are a subset of URIs

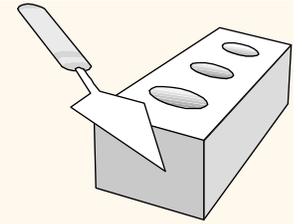


Hypertext Transfer Protocol

- ❖ 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?
 1. Client (web browser) sends HTTP request to server
 2. Server receives request and replies
 3. Client receives reply; makes new requests

HTTP (Contd.)

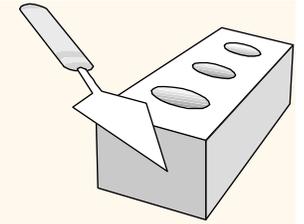


Client to Server:

```
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>
<b>The Character of Physical Law</b>
...
```

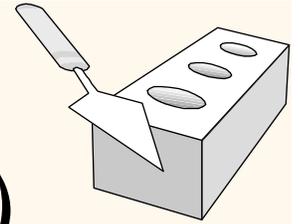


HTTP Protocol Structure

HTTP Requests

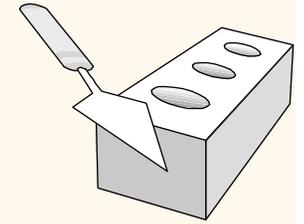
- ❖ Request line: **GET ~/index.html HTTP/1.1**
 - **GET**: Http method field (possible values are GET and POST, more later)
 - **~/index.html**: URI field
 - **HTTP/1.1**: HTTP version field
- ❖ Type of client: **User-agent: Mozilla/4.0**
- ❖ What types of files will the client accept:
 Accept: text/html,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



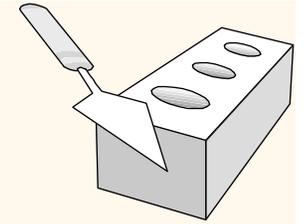
Some Remarks About HTTP

❖ 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

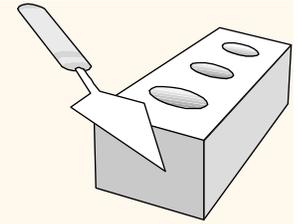
- The presentation language for the Internet

❖ XML

- A self-describing, hierarchical data model.
- [XML Examples and Exercises](#)

❖ And others, e.g. SGML, not covered.

HTML: An Example

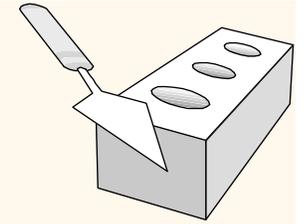


```
<HTML>
  <HEAD></HEAD>
  <BODY>
    <h1>Barns and Nobble Internet
      Bookstore</h1>
    Our inventory:

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

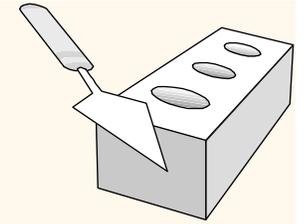
```
<h3>Fiction</h3>
<b>Waiting for the Mahatma</b>
<UL>
  <LI>Author: R.K. Narayan</LI>
  <LI>Published 1981</LI>
</UL>
<b>The English Teacher</b>
<UL>
  <LI>Author: R.K. Narayan</LI>
  <LI>Published 1980</LI>
  <LI>Paperback</LI>
</UL>

</BODY>
</HTML>
```



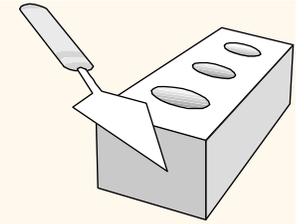
HTML: A Short Introduction

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



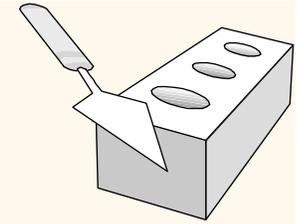
HTML: Sample Commands

- ❖ `<HTML>`:
- ❖ ``: unordered list
- ❖ ``: list entry
- ❖ `<h1>`: largest heading
- ❖ `<h2>`: second-level heading, `<h3>`, `<h4>`
analogous
- ❖ `Title`: Bold



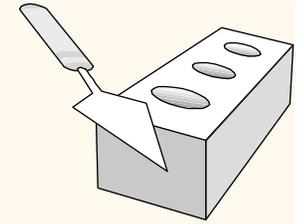
XML – The Extensible Markup Language

- ❖ Language
 - A way of communicating information
 - Part of the Semantic Web.
- ❖ Markup
 - Notes or meta-data that describe your data or language
- ❖ Extensible
- ❖ Limitless ability to define new languages or data sets.
- ❖ Sophisticated query languages for XML are available:
 - Xquery
 - XPath



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>
```

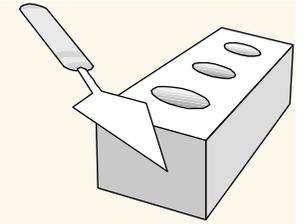


XML – What's The Point?

- ❖ 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

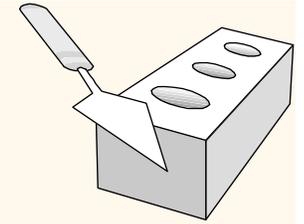


XML – Structure

- ❖ XML 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**

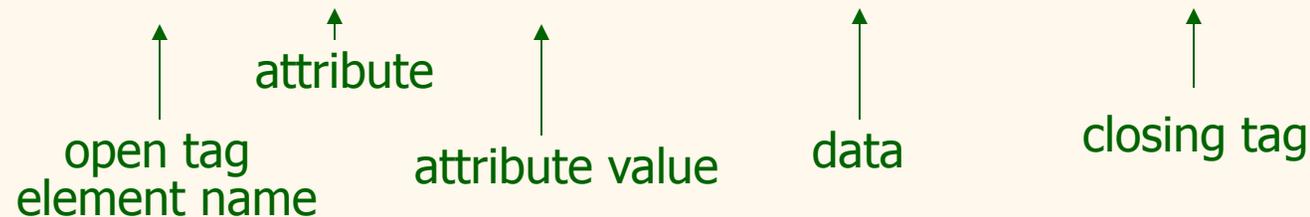
`<BOOK genre="Science" format="Hardcover">...</BOOK>`



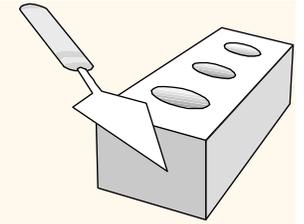


XML – Elements

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

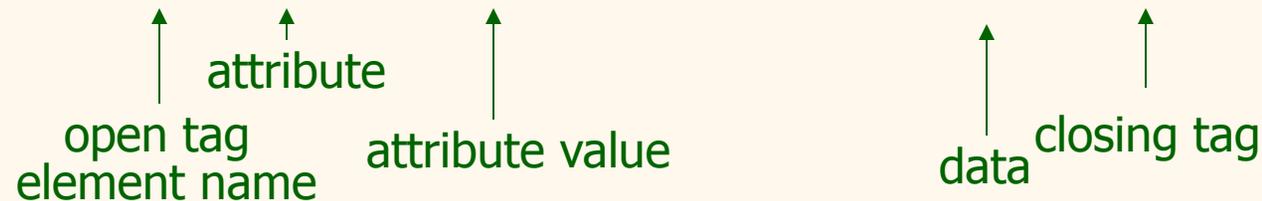


- ❖ XML is case and space sensitive
- ❖ Element opening and closing tag names must be identical
- ❖ Opening tags: “`<`” + element name + “`>`”
- ❖ Closing tags: “`</`” + element name + “`>`”

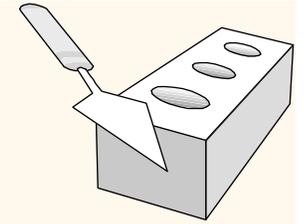


XML – Attributes

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

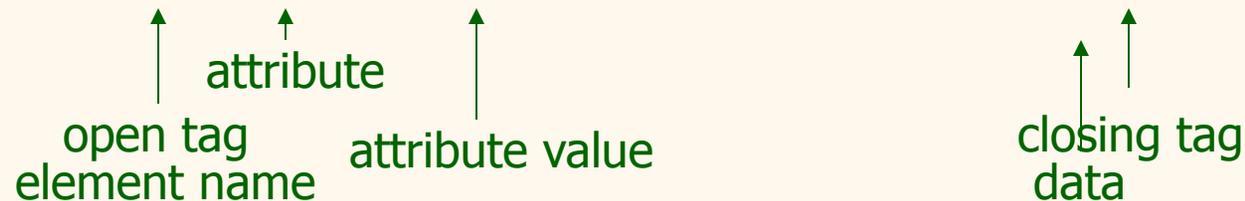


- ❖ Attributes provide additional information for element tags.
- ❖ There can be zero or more attributes in every element; each one has the the form:
 - attribute_name='attribute_value'*
 - There is no space between the name and the `"='"`
 - Attribute values must be surrounded by `"` or `'` characters
- ❖ Multiple attributes are separated by white space (one or more spaces or tabs).

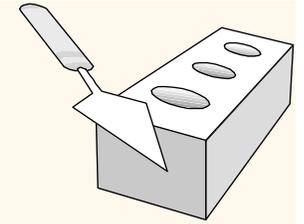


XML – Data and Comments

`<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 -->`

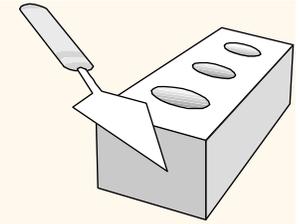


XML – Nesting & Hierarchy

- ❖ 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

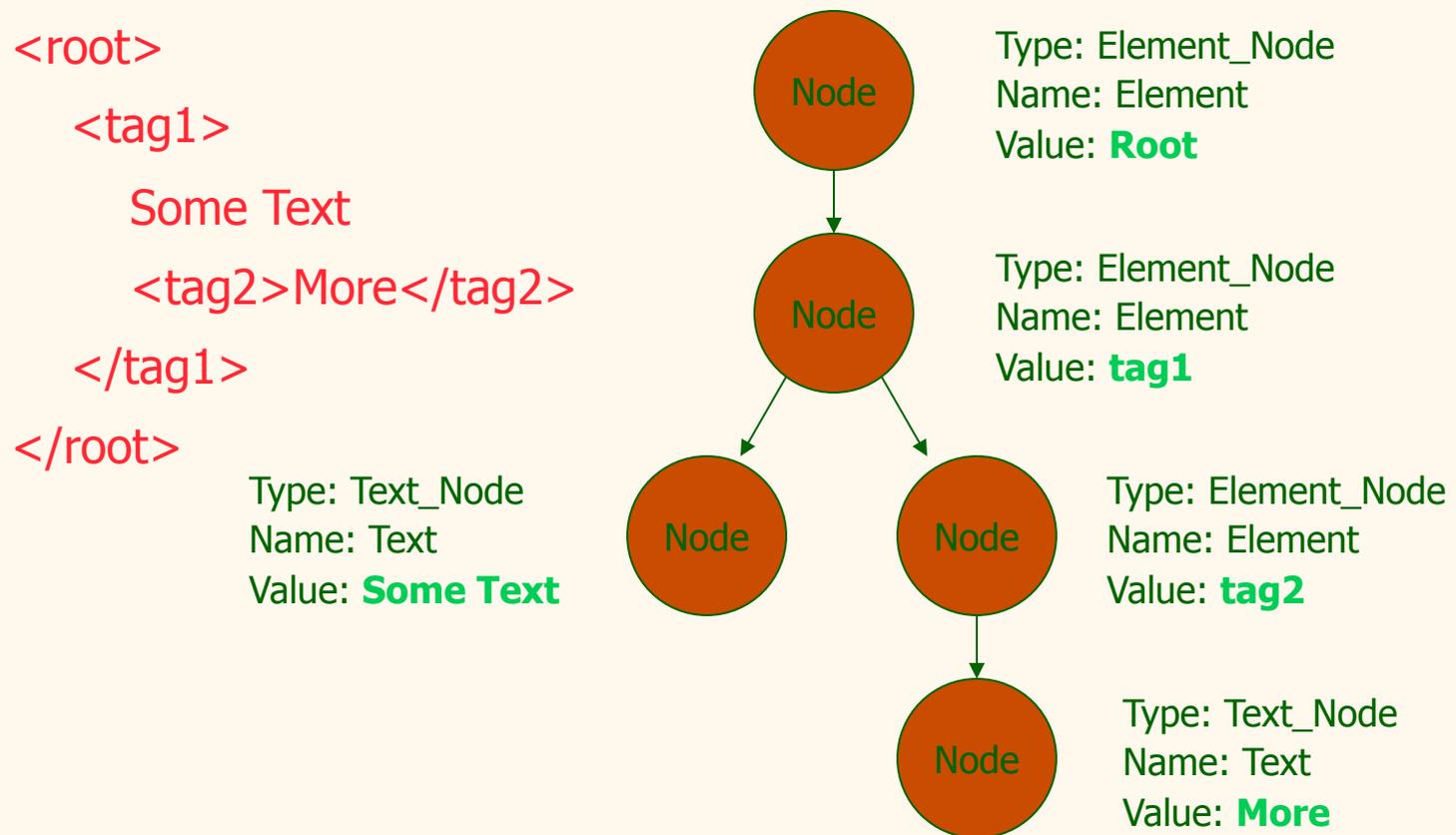
```
<root>  
  <tag1>  
    Some Text  
    <tag2>More</tag2>  
  </tag1>  
</root>
```

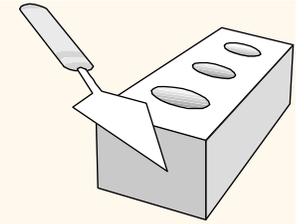
[XML Examples and Exercises](#)



XML – Storage

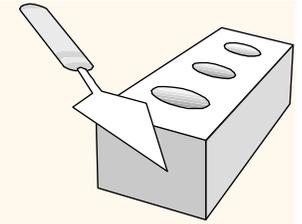
- ❖ Storage is done just like an n-ary tree (DOM)





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>



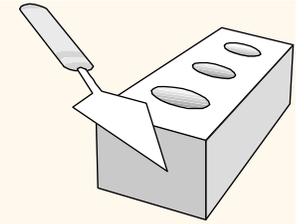
<Basket>

<Apple/>

<Cherry flavor='good'/>

<Orange/>

</Basket>



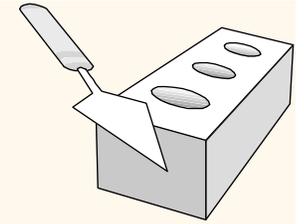
DTD - !ELEMENT

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

Name

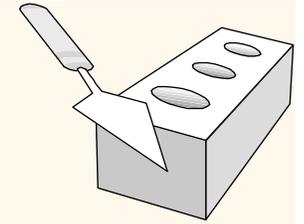
Children

- ❖ **!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:
 - $exp_1, exp_2, exp_3, \dots, exp_k$: A list of regular expressions
 - exp^* : An optional expression with zero or more occurrences
 - exp^+ : An optional expression with one or more occurrences
 - $exp_1 | exp_2 | \dots | exp_k$: A disjunction of expressions



DTD - !ATTLIST

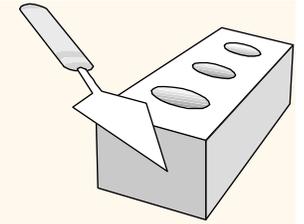
```
<!ATTLIST Cherry flavor CDATA #REQUIRED>
```

Diagram illustrating the components of the DTD declaration:

- Cherry: Element
- flavor: Attribute
- CDATA: Type
- #REQUIRED: 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

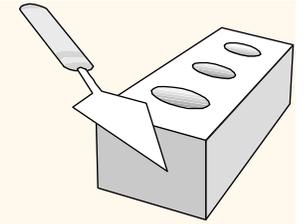
```
<basket>  
  <Cherry flavor=good>  
</Basket>
```

Well-Formed but Invalid

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

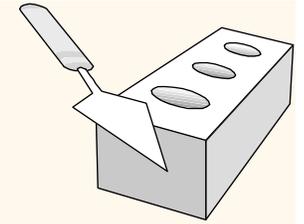
Well-Formed and Valid

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



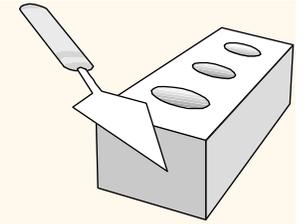
XML and DTDs

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



HTML Forms

- ❖ Common way to communicate data from client to middle tier
- ❖ General format of a form:
 - `<FORM ACTION="page.jsp" METHOD="GET" NAME="LoginForm">`
 - ...
 - `</FORM>`
- ❖ Components of an HTML FORM tag:
 - **ACTION**: Specifies URI that handles the content
 - **METHOD**: Specifies HTTP GET or POST method
 - **NAME**: Name of the form; can be used in client-side scripts to refer to the form



Inside HTML Forms

❖ INPUT tag

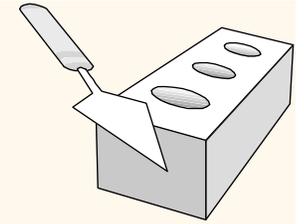
▪ Attributes:

- TYPE: text (text input field), password (text input field where input is, reset (resets all input fields)
- NAME: symbolic name, used to identify field value at the middle tier
- VALUE: default value

- Example: `<INPUT TYPE="text" Name="title">`

❖ Example form:

```
<form method="POST" action="TableOfContents.jsp">  
  <input type="text" name="userid">  
  <input type="password" name="password">  
  <input type="submit" value="Login" name="submit">  
  <input type="reset" value="Clear">  
</form>
```



Passing Arguments

Two methods: GET and POST

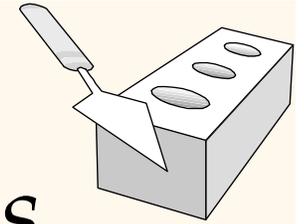
❖ GET

- Form contents go into the submitted URI

- Structure:

`action?name1=value1&name2=value2&name3=value3`

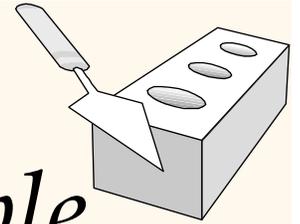
- Action: name of the URI specified in the form
- (name,value)-pairs come from INPUT fields in the form; empty fields have empty values ("name=")
- Example from previous password form:
`TableOfContents.jsp?userid=john&password=johnpw`
- Note that the page named action needs to be a program, script, or page that will process the user input



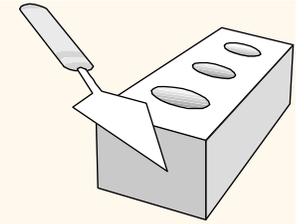
HTTP GET: Encoding Form Fields

- ❖ Form fields can contain general ASCII characters that cannot appear in an URI
- ❖ A special encoding convention converts such field values into “URI-compatible” characters:
 1. Convert all “special” characters to %xyz, where xyz is the ASCII code of the character. Special characters include &, =, +, %, etc.
 2. Convert all spaces to the “+” character
 3. Glue (name,value)-pairs from the form INPUT tags together with “&” to form the URI

HTML Forms: A Complete Example

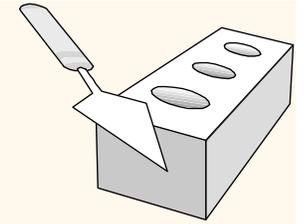


```
<form method="POST" action="TableOfContents.jsp">
  <table align = "center" border="0" width="300">
    <tr>
      <td>Userid</td>
      <td><input type="text" name="userid" size="20"></td>
    </tr>
    <tr>
      <td>Password</td>
      <td><input type="password" name="password" size="20"></td>
    </tr>
    <tr>
      <td align = "center"><input type="submit" value="Login"
        name="submit"></td>
    </tr>
  </table>
</form>
```



JavaScript

- ❖ Goal: Add functionality to the presentation tier.
- ❖ Sample applications:
 - Detect browser type and load browser-specific page
 - Form validation: Validate form input fields
 - Browser control: Open new windows, close existing windows (example: pop-up ads)
- ❖ Usually embedded directly inside the HTML with the `<SCRIPT> ... </SCRIPT>` tag.
- ❖ `<SCRIPT>` tag has several attributes:
 - LANGUAGE: specifies language of the script (such as javascript)
 - SRC: external file with script code
 - Example:
`<SCRIPT LANGUAGE="JavaScript" SRC="validate.js">
</SCRIPT>`



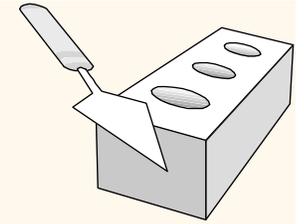
JavaScript (Contd.)

- ❖ If `<SCRIPT>` tag does not have a SRC attribute, then the JavaScript is directly in the HTML file.

- ❖ Example:

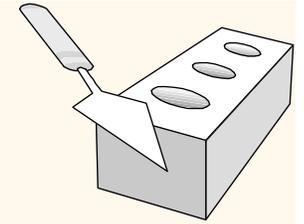
```
<SCRIPT LANGUAGE="JavaScript" >  
<!-- alert("Welcome to our bookstore")  
//-->  
</SCRIPT>
```

- ❖ Two different commenting styles
 - `<!--` comment for HTML, since the following JavaScript code should be ignored by the HTML processor
 - `//` comment for JavaScript in order to end the HTML comment



JavaScript (Contd.)

- ❖ JavaScript is a complete scripting language
 - Variables
 - Assignments (=, +=, ...)
 - Comparison operators (<, >, ...), boolean operators (&&, ||, !)
 - Statements
 - if (condition) {statements;} else {statements;}
 - for loops, do-while loops, and while-loops
 - Functions with return values
 - Create functions using the function keyword
 - f(arg1, ..., argk) {statements;}



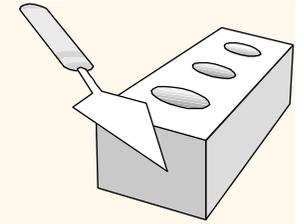
JavaScript: A Complete Example

HTML Form:

```
<form method="POST"
  action="TableOfContents.jsp">
  <input type="text"
    name="userid">
  <input type="password"
    name="password">
  <input type="submit"
    value="Login"
    name="submit">
  <input type="reset"
    value="Clear">
</form>
```

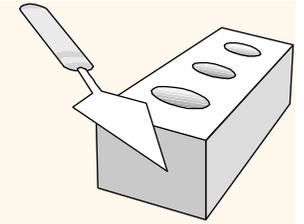
Associated JavaScript:

```
<script language="javascript">
function testLoginEmpty()
{
  loginForm = document.LoginForm
  if ((loginForm.userid.value == "") ||
    (loginForm.password.value == ""))
  {
    alert('Please enter values for userid and
      password. ');
    return false;
  }
  else return true;
}
</script>
```



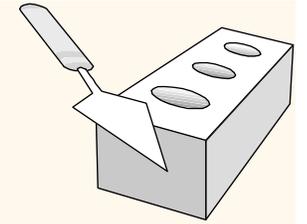
Lecture Overview

- ❖ 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)



Overview of the Middle Tier

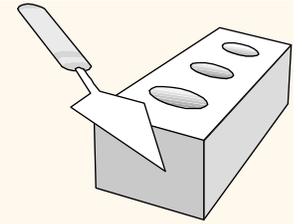
- ❖ **Functionality of the middle tier**
 - Encodes business logic
 - Connects to database system(s)
 - Accepts form input from the presentation tier
 - Generates output for the presentation tier
- ❖ **We will cover**
 - CGI: Protocol for passing arguments to programs running at the middle tier
 - Application servers: Runtime environment at the middle tier
 - Maintaining state: How to maintain state at the middle tier.
Main focus: Cookies.



W3Schools vs. W3Fools

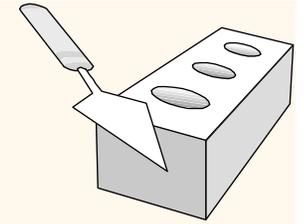
- ❖ W3Schools is actually *not* related to W3C.
- ❖ Mondial data in SQL vs. XML.

CGI: *Common Gateway Interface*



- ❖ Transmits arguments from HTML forms to application programs running at the middle tier
- ❖ Details of the actual CGI protocol unimportant → libraries implement high-level interfaces

- ❖ Example: Implementing a wiki.
 - The user agent requests the name of an entry.
 - The server retrieves the source of that entry's page.
 - Transforms it into HTML
 - Sends the result.



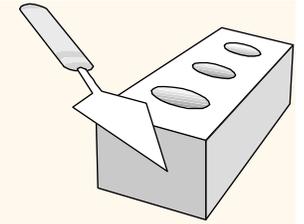
CGI: Example

❖ HTML form:

```
<form action="findbooks.cgi" method=POST>  
Type an author name:  
<input type="text" name="authorName">  
<input type="submit" value="Send it">  
<input type="reset" value="Clear form">  
</form>
```

❖ Perl code:

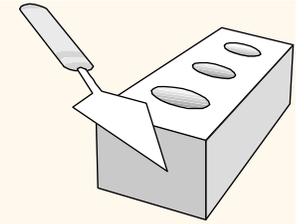
```
use CGI;  
$dataIn=new CGI;  
$dataIn->header();  
$authorName=$dataIn->param('authorName');  
print("<HTML><TITLE>Argument passing test</TITLE>");  
print("The author name is " + $authorName);  
print("</HTML>");  
exit;
```



CGI Disadvantages

❖ Disadvantages:

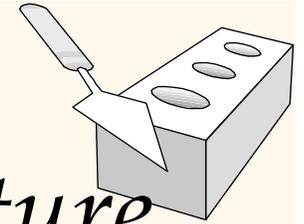
- Each CGI script invocation leads to a new process.
- No resource sharing between application programs (e.g., database connections)
- Remedy: Application servers share threads in process.



Application Servers

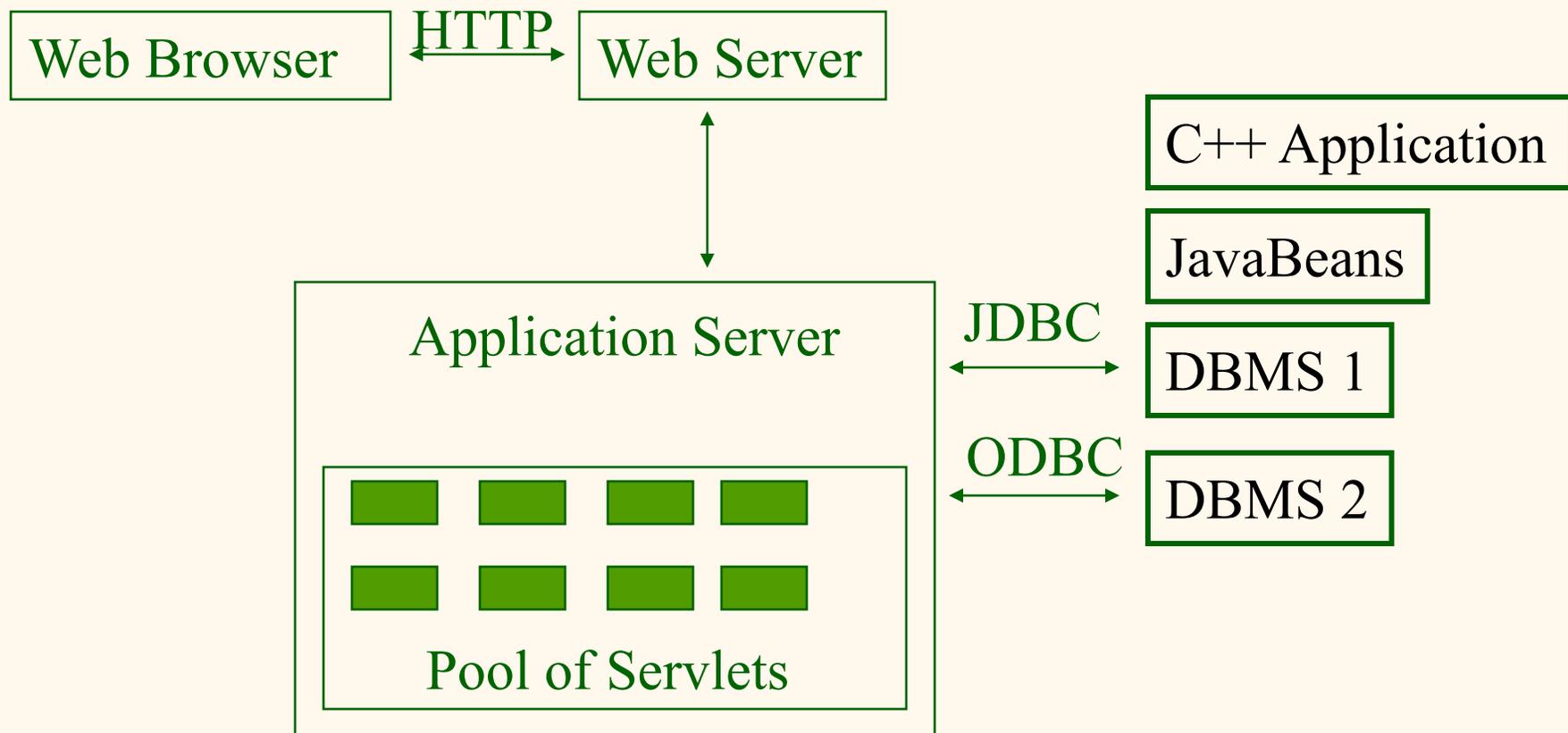
❖ Idea: Avoid the overhead of CGI

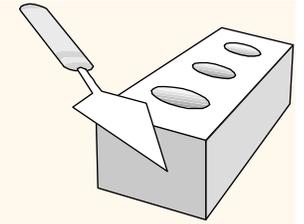
- Main pool of threads inside processes.
- Requests are assigned to threads (cheap) rather than separate processes.
- Manage connections
- Enable access to heterogeneous data sources
- Other functionality such as APIs for session management.
- Servlets handle client requests using Java.



Application Server: Process Structure

Process Structure





Maintaining State

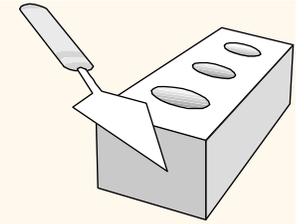
HTTP is stateless.

❖ Advantages

- Easy to use: don't need memory management.
- Great for static-information applications (“fire and forget”)
- Requires no extra memory space

❖ Disadvantages

- No record of previous requests means
 - No shopping baskets
 - No user logins
 - No custom or dynamic content
 - Security is more difficult to implement



Application State

❖ Server-side state

- Information is stored in a database, or in the application layer's local memory

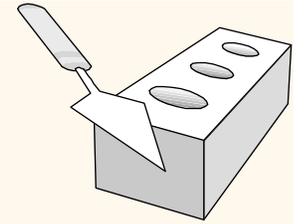
❖ Client-side state

- Information is stored on the client's computer in the form of a cookie

❖ Hidden state

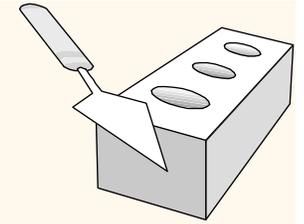
- Information is hidden within dynamically created web pages

Application State



So many kinds of
state...
...how will I choose?

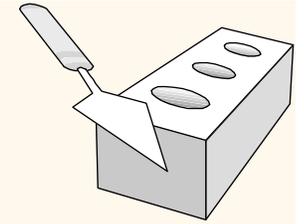




Server-Side State

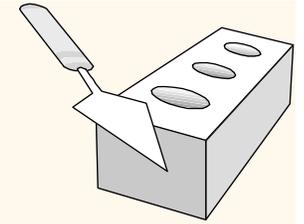
- ❖ Many types of Server side state:
- ❖ 1. Store information in a database
 - Data will be safe in the database
 - BUT: requires a database access to query or update the information
- ❖ 2. Use application layer's local memory
 - Can map the user's IP address to some state
 - BUT: this information is volatile and takes up lots of server main memory

5 million IPs = 20 MB



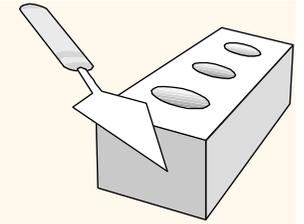
Server-Side State

- ❖ Should use Server-side state maintenance for information that needs to persist
 - Old customer orders
 - “Click trails” of a user’s movement through a site
 - Permanent choices a user makes



Client-side State: Cookies

- ❖ Storing text on the client which will be passed to the application with every HTTP request.
 - Can be disabled by the client.
 - Are wrongfully perceived as "dangerous", and therefore will scare away potential site visitors if asked to enable cookies¹
- ❖ Are a collection of (Name, Value) pairs.
- ❖ Discussion Question: what do you think of cookies?



Client State: Cookies

❖ Advantages

- Easy to use in Java Servlets / JSP
- Provide a simple way to keep non-essential data on the client side even when the browser has closed

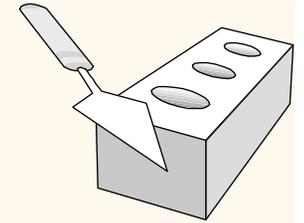
❖ Disadvantages

- Limit of 4 kilobytes of information
- Users can (and often will) disable them

❖ Should use cookies to store interactive state

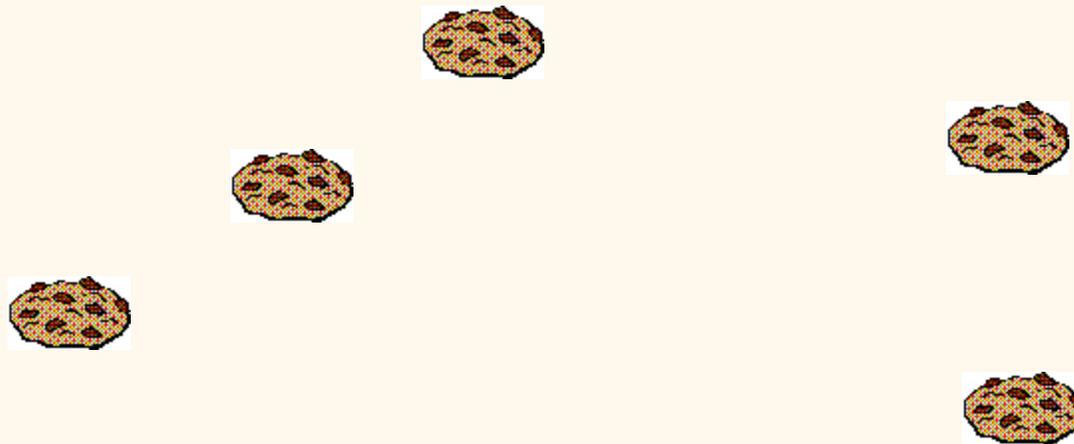
- The current user's login information
- The current shopping basket
- Any non-permanent choices the user has made

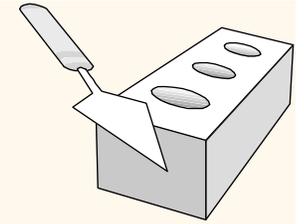
Creating A Cookie



```
Cookie myCookie =  
    new Cookie("username", "jeffd");  
response.addCookie(userCookie);
```

- ❖ You can create a cookie at any time





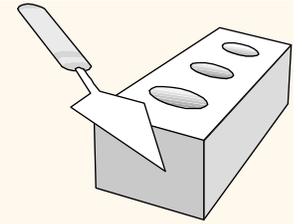
Accessing A Cookie

```
Cookie[] cookies = request.getCookies();
String theUser;
for(int i=0; i<cookies.length; i++) {
    Cookie cookie = cookies[i];
    if(cookie.getName().equals("username"))
        theUser = cookie.getValue();
}
// at this point theUser == "username"
```

- ❖ Cookies need to be accessed BEFORE you set your response header:

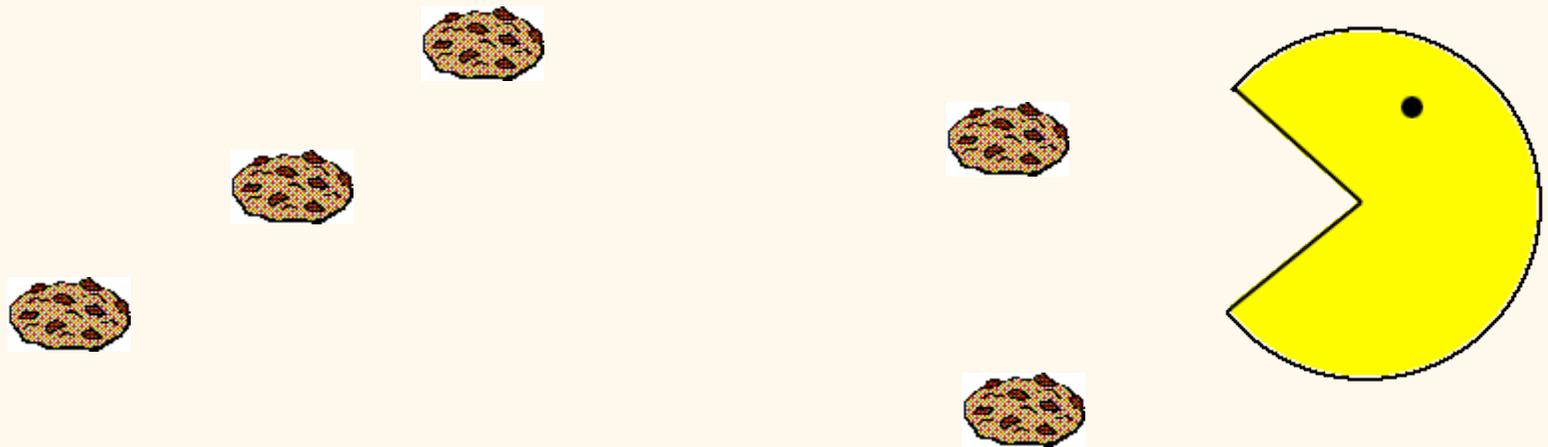
```
response.setContentType("text/html");
PrintWriter out = response.getWriter();
```

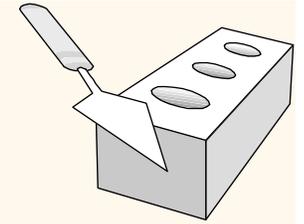
Cookie Features



❖ Cookies can have

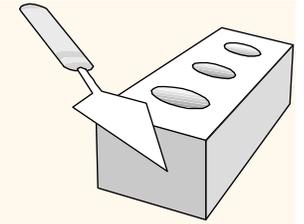
- A duration (expire right away or persist even after the browser has closed)
- Filters for which domains/directory paths the cookie is sent to.





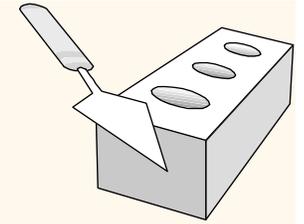
Hidden State

- ❖ Often users will disable cookies
- ❖ You can “hide” data in two places:
 - Hidden fields within a form
 - Using the path information
- ❖ Requires no “storage” of information because the state information is passed inside of each web page



Hidden State: Hidden Fields

- ❖ Declare hidden fields within a form:
 - `<input type='hidden' name='user' value='username' />`
- ❖ Users will not see this information (unless they view the HTML source)
- ❖ If used prolifically, it's a killer for performance since EVERY page must be contained within a form.



Hidden State: Path Information

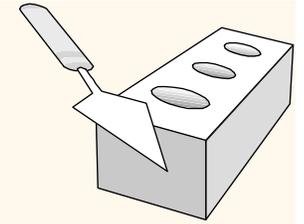
- ❖ Path information is stored in the URL request:

```
http://server.com/index.htm?user=jeffd
```

- ❖ Can separate 'fields' with an & character:

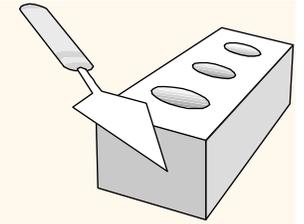
```
index.htm?user=jeffd&preference=pepsi
```

- ❖ There are mechanisms to parse this field in Java. Check out the `javax.servlet.http.HttpUtils` `parseQueryString()` method.



Multiple state methods

- ❖ Typically all methods of state maintenance are used:
 - User logs in and this information is stored in a cookie
 - User issues a query which is stored in the path information
 - User places an item in a shopping basket cookie
 - User purchases items and credit-card information is stored/retrieved from a database
 - User leaves a click-stream which is kept in a log on the web server (which can later be analyzed)



Summary

We covered:

- ❖ Internet Concepts (URIs, HTTP)
- ❖ Web data formats
 - HTML, XML, DTDs
- ❖ Three-tier architectures
- ❖ The presentation layer
 - HTML forms; HTTP Get and POST, URL encoding; Javascript.
- ❖ The middle tier
 - CGI, application servers, Servlets, passing arguments, maintaining state (cookies).
- ❖ Only lecture material will be on exam (not other material from Ch.7).