

WEB TECHNOLOGIES  
A COMPUTER SCIENCE PERSPECTIVE

JEFFREY C. JACKSON

Chapter 6  
Server-side Programming:  
Java Servlets

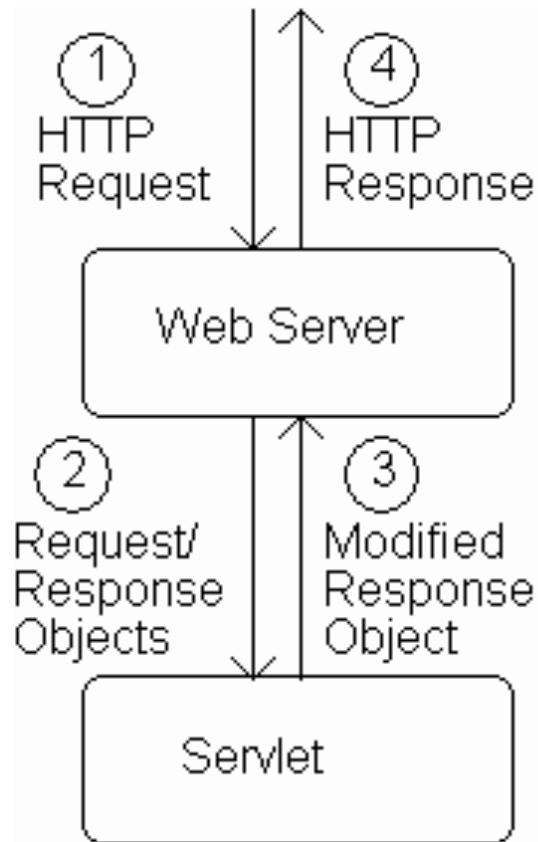
# Server-side Programming

- The combination of
  - HTML
  - JavaScript
  - DOMis sometimes referred to as **Dynamic HTML** (DHTML)
- Web pages that include scripting are often called **dynamic** pages (vs. **static**)

# Server-side Programming

- Similarly, web server response can be static or dynamic
  - **Static**: HTML document is retrieved from the file system and returned to the client
  - **Dynamic**: HTML document is generated by a program in response to an HTTP request
- Java servlets are one technology for producing dynamic server responses
  - **Servlet** is a class instantiated by the server to produce a dynamic response

# Servlet Overview



# Servlet Overview

1. When server starts it **instantiates servlets**
2. Server receives HTTP request, **determines need** for dynamic response
3. Server **selects the appropriate servlet** to generate the response, creates request/response objects, and passes them to a method on the servlet instance
4. Servlet **adds information** to response object via method calls
5. Server **generates HTTP response** based on information stored in response object

# Hello World! Servlet

```
public class ServletHello extends HttpServlet
{
    /**
     * Respond to any HTTP GET request with an
     * HTML Hello World! page.
     */
    public void doGet (HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException
    {
        // Set the HTTP content type in response header
        response.setContentType("text/html; charset=\"UTF-8\"");

        // Obtain a PrintWriter object for creating the body
        // of the response
        PrintWriter servletOut = response.getWriter();
    }
}
```

# Hello World! Servlet

All servlets we will write  
are subclasses of  
**HttpServlet**

```
public class ServletHello extends HttpServlet
{
    /**
     * Respond to any HTTP GET request with an
     * HTML Hello World! page.
     */
    public void doGet (HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException
    {
        // Set the HTTP content type in response header
        response.setContentType("text/html; charset=\"UTF-8\"");

        // Obtain a PrintWriter object for creating the body
        // of the response
        PrintWriter servletOut = response.getWriter();
    }
}
```

# Hello World! Servlet

```
public class ServletHello extends HttpServlet
{
    /**
     * Respond to any HTTP GET request with an
     * HTML Hello World! page.
     */
    public void doGet (HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException
    {
        // Set the HTTP content type in response header
        response.setContentType("text/html; charset=\"UTF-8\"");

        // Obtain a PrintWriter object for creating the body
        // of the response
        PrintWriter servletOut = response.getWriter();
    }
}
```

Server calls doGet() in response to GET request



# Hello World! Servlet

```
public class ServletHello extends HttpServlet
{
    /**
     * Respond to any HTTP GET request with an
     * HTML Hello World! page.
     */
    public void doGet (HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException
    {
        // Set the HTTP content type in response header
        response.setContentType("text/html; charset=\"UTF-8\"");

        // Obtain a PrintWriter object for creating the body
        // of the response
        PrintWriter servletOut = response.getWriter();
    }
}
```

Interfaces implemented by request/response objects

# Hello World! Servlet

```
public class ServletHello extends HttpServlet
{
    /**
     * Respond to any HTTP GET request with an
     * HTML Hello World! page.
     */
    public void doGet (HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException
    {
        // Set the HTTP content type in response header
        response.setContentType("text/html; charset=\"UTF-8\"");

        // Obtain a PrintWriter object for creating the body
        // of the response
        PrintWriter servletOut = response.getWriter();
    }
}
```

Production servlet should catch these exceptions

# Hello World! Servlet

- JWSDP Tomcat server **exception handling**:
  - Stack trace appended to `logs/jwsdp_log.*.txt`
  - HTML document returned to client may (or may not) contain partial stack trace
- **Servlet output** to `System.out.print()`, `printStackTrace()`, *etc.* is appended to `logs/launcher.server.log`

# Hello World! Servlet

```
public class ServletHello extends HttpServlet
{
    /**
     * Respond to any HTTP GET request with an
     * HTML Hello World! page.
     */
    public void doGet (HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException
    {
        // Set the HTTP content type in response header
        response.setContentType("text/html; charset=\"UTF-8\"");

        // Obtain a PrintWriter object for creating the body
        // of the response
        PrintWriter servletOut = response.getWriter();
    }
}
```

First two  
things done  
by typical servlet;  
must be in this  
order

# Hello World! Servlet

```
        // Create the body of the response
        servletOut.println(
"<!DOCTYPE html \n" +
"    PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN\" \n" +
"    \"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd\"> \n" +
"<html xmlns='http://www.w3.org/1999/xhtml'> \n" +
"  <head> \n" +
"    <title> \n" +
"      ServletHello.java \n" +
"    </title> \n" +
"  </head> \n" +
"  <body> \n" +
"    <p> \n" +
"      Hello World! \n" +
"    </p> \n" +
"  </body> \n" +
"</html> ");
        servletOut.close();
    }
}
```

# Hello World! Servlet

```
        // Create the body of the response
        servletOut.println(
"<!DOCTYPE html \n" +
"    PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN\" \n" +
"    \"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd\"> \n" +
"<html xmlns='http://www.w3.org/1999/xhtml'> \n" +
"    <head> \n" +
"        <title> \n" +
"            ServletHello.java \n" +
"        </title> \n" +
"    </head> \n" +
"    <body> \n" +
"        <p> \n" +
"            Hello World! \n" +
"        </p> \n" +
"    </body> \n" +
"</html> ");
        servletOut.close();
    }
}
```

HTML generated by calling `print()` or `println()` on the servlet's `PrintWriter` object

# Hello World! Servlet

```
        // Create the body of the response
        servletOut.println(
"<!DOCTYPE html \n" +
"    PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN\" \n" +
"    \"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd\"> \n" +
"<html xmlns='http://www.w3.org/1999/xhtml'> \n" +
"    <head> \n" +
"        <title> \n" +
"            ServletHello.java \n" +
"        </title> \n" +
"    </head> \n" +
"    <body> \n" +
"        <p> \n" +
"            Hello World! \n" +
"        </p> \n" +
"    </body> \n" +
"</html> ");
        servletOut.close();
    }
}
```

Good practice to explicitly close  
the **PrintWriter** when done

# Servlets vs. Java Applications

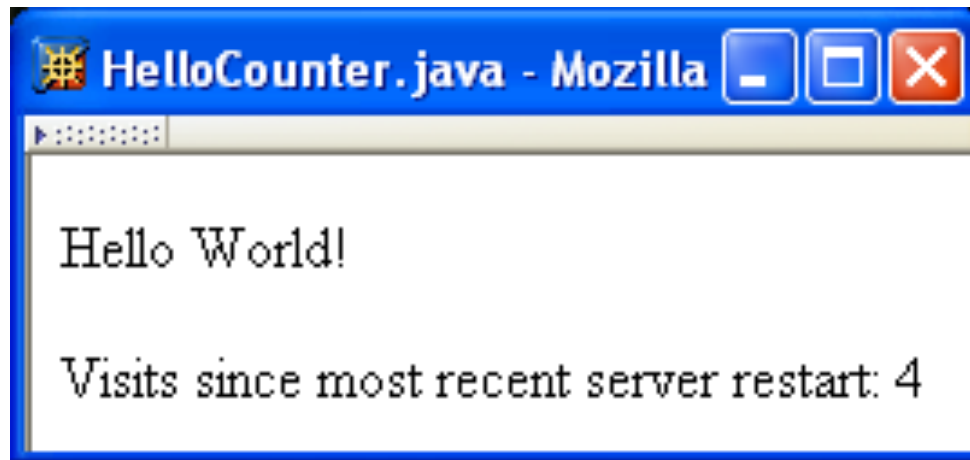
- Servlets **do not have a main()**
  - The `main()` is in the server
  - Entry point to servlet code is via call to a method (`doGet()` in the example)
- Servlet **interaction with end user is indirect** via request/response object APIs
  - Actual HTTP request/response processing is handled by the server
- Primary servlet **output is typically HTML**



# Running Servlets

- Simple way to run a servlet (better later):
  1. Compile servlet (make sure that JWSDP libraries are on path)
  2. Copy .class file to `shared/classes` directory
  3. (Re)start the Tomcat web server
  4. If the class is named `ServletHello`, browse to  
`http://localhost:8080/servlet/ServletHello`

# Dynamic Content



# Dynamic Content

```
public class HelloCounter extends HttpServlet
{
    // Number of times the servlet has been executed since
    // the program (web server) started
    private int visits=0;

    [...] // removed doGet() declaration and initialization

    // Obtain a PrintWriter object for creating the body
    // of the response
    PrintWriter servletOut = response.getWriter();

    // Compute the number of visits to the URL for this servlet
    visits++;
}
```

# Dynamic Content

```
" <p> \n" +  
"   Hello World! \n" +  
" </p> \n" +  
" <p> \n" +  
"   This page has been viewed \n" +  
"     visits +  
"   times since the most recent server restart. \n" +  
" </p> \n" +
```

# Dynamic Content

- Potential problems:
  - Assuming **one instance** of servlet on **one server**, but
    - Many Web sites are distributed over multiple servers
    - Even a single server can (not default) create multiple instances of a single servlet
  - Even if the assumption is correct, this servlet does not handle **concurrent accesses** properly
    - We'll deal with this later in the chapter

# Servlet Life Cycle

- Servlet API life cycle methods
  - **init()**: called when servlet is instantiated; must return before any other methods will be called
  - **service()**: method called directly by server when an HTTP request is received; default **service()** method calls **doGet()** (or related methods covered later)
  - **destroy()**: called when server shuts down

# Servlet Life Cycle

```
public void init()
    throws ServletException
{
    try {
        BufferedReader br =
            new BufferedReader(new FileReader("aFile"));
        visits = (new Integer(br.readLine())).intValue();
    }
    catch (FileNotFoundException fnfe) {
        throw new UnavailableException("File not found: " +
            fnfe.toString());
    }
    catch (Exception e) {
        throw new UnavailableException("Data problem: " +
            e.toString());
    }
}
```

Example life cycle method:  
attempt to initialize **visits** variable  
from file

# Servlet Life Cycle

```
public void init()
    throws ServletException
{
    try {
        BufferedReader br =
            new BufferedReader(new FileReader("aFile"));
        visits = (new Integer(br.readLine())).intValue();
    }
    catch (FileNotFoundException fnfe) {
        throw new UnavailableException("File not found: " +
            fnfe.toString());
    }
    catch (Exception e) {
        throw new UnavailableException("Data problem: " +
            e.toString());
    }
}
```

**Exception to be thrown  
if initialization fails and servlet  
should not be instantiated**



# Parameter Data

- The request object (which implements **HttpServletRequest**) provides information from the HTTP request to the servlet
- One type of information is **parameter data**, which is information from the query string portion of the HTTP request

`http://www.example.com/servlet/PrintThis?arg=aString`

Query string with  
one parameter

# Parameter Data

- Parameter data is the Web analog of arguments in a method call:

```
System.out.println("aString");  
http://www.example.com/servlet/PrintThis?arg=aString
```

- Query string syntax and semantics

# Parameter Data

- Query string syntax and semantics

- Multiple parameters separated by &

`http://www.example.com/servlet/PrintThis?arg=aString&color=red`

- Order of parameters does not matter

`http://www.example.com/servlet/PrintThis?color=red&arg=aString`

- All parameter values are strings

`http://www.example.com/servlet/PrintThis(arg=&color=red`

Value of arg is empty string

# Parameter Data

- Parameter names and values can be any 8-bit characters
- **URL encoding** is used to represent non-alphanumeric characters:

`http://www.example.com/servlet/PrintThis?arg=%27a+String%27`

Value of arg is  
'a String'

- **URL decoding** applied by server to retrieve intended name or value

# Parameter Data

- URL encoding algorithm

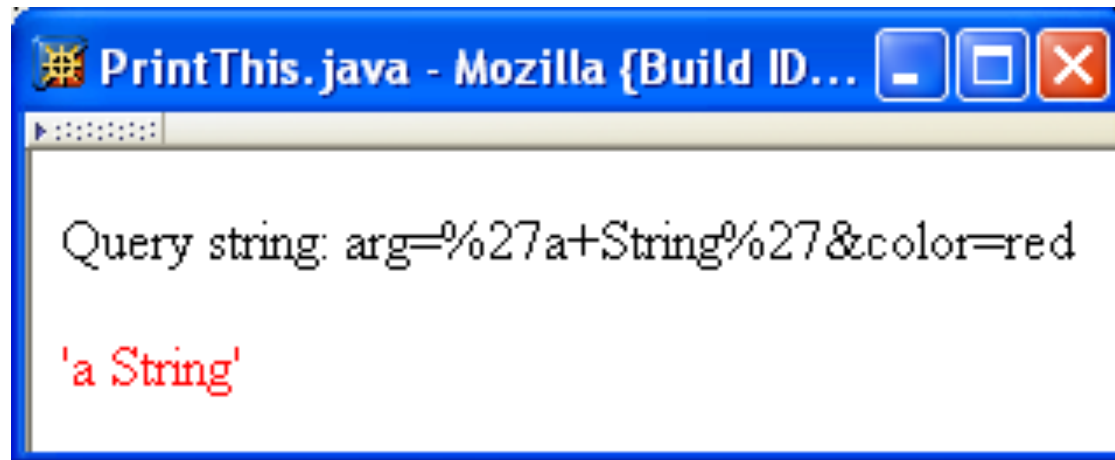
```
initialize the result to the empty string
for each 8-bit character in the original string
  if the character is an alphanumeric
    concatenate the character to the result
  else if the character is a space
    concatenate a plus sign (+) to the result
  else
    concatenate a percent sign (%) followed by
      the two-digit hexadecimal value of the character
    to the result
  endif
endfor
return result
```

# Parameter Data

TABLE 6.1: Some `HttpServletRequest` methods for accessing parameter data.

Method	Purpose
<code>String getQueryString()</code>	Returns the entire query string in its original (URL encoded) form.
Enumeration <code>getParameterNames()</code>	Returns Enumeration of String values representing all parameter names (URL decoded) in the query string.
<code>String getParameter</code> (String name)	Returns String representing value (URL decoded) of parameter named name, or null if parameter is not present in the query string.
<code>String[]</code> <code>getParameterValues</code> (String name)	Returns array of String's representing all values (URL decoded) of parameter named name, or null if parameter is not present in the query string.

# Parameter Data



# Parameter Data

```
" <body> \n" +
"   <p>Query string: " +
    WebTechUtil.escapeXML(request.getQueryString()) + "</p>" );

    // Decide whether or not to set color
    String color = request.getParameter("color");
    if (color == null) {
        servletOut.println(
"   <p> " );
        } else {
            servletOut.println(
"   <p style='color:" +
                WebTechUtil.escapeQuotes(WebTechUtil.escapeXML(color)) +
" '> " );
        }
    }
```



# Parameter Data

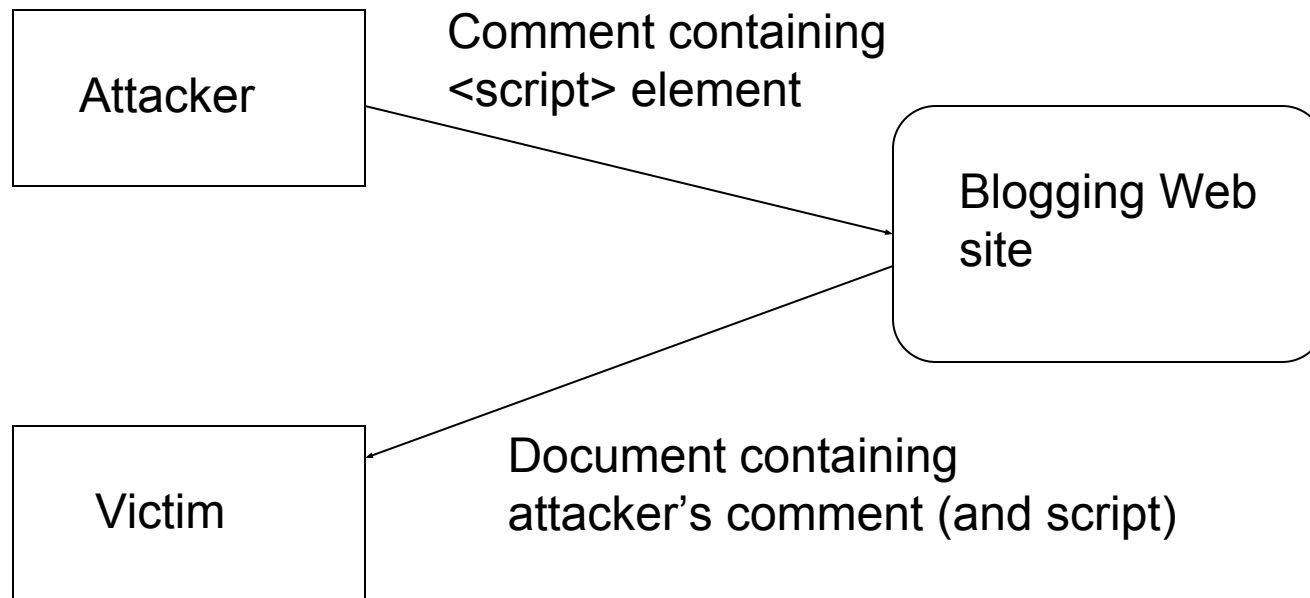
Must escape XML special characters in all user-supplied data before adding to HTML to avoid *cross-site scripting* attacks

```
" <body> \n" +
" <p>Query string: " +
  WebTechUtil.escapeXML(request.getQueryString()) + "</p>" );

    // Decide whether or not to set color
    String color = request.getParameter("color");
    if (color == null) {
        servletOut.println(
" <p> " );
    } else {
        servletOut.println(
" <p style='color:" +
    WebTechUtil.escapeQuotes(WebTechUtil.escapeXML(color)) +
"> " );
    }
}
```

# Parameter Data

- Cross-site scripting



# Parameter Data

```
" <body> \n" +
"   <p>Query string: " +
    WebTechUtil.escapeXML(request.getQueryString()) + "</p>" );

    // Decide whether or not to set color
    String color = request.getParameter("color");
    if (color == null) {
        servletOut.println(
"   <p> " );
    } else {
        servletOut.println(
"   <p style='color:" +
        WebTechUtil.escapeQuotes(WebTechUtil.escapeXML(color)) +
" '> " );
    }
}
```

Also need to escape quotes within attribute values.

# Parameter Data

```
// Decide which string to output
String arg = request.getParameter("arg");
if (arg == null) {
    arg = "Hello World!";
}

// Create remainder of response body
servletOut.println(
"    " + WebTechUtil.escapeXML(arg) + "\n" +
"    </p> \n" +
" </body> \n" +
```

# Parameter Data

- A **form** automatically generates a query string when submitted
  - Parameter **name** specified by value of name attributes of form controls

- Parameter **value** depends on control type

```
<input type="text" name="username" size="40" />  
  
<label>  
  <input type="checkbox" name="boxgroup1" value="tall" />tall  
</label>
```

Value for checkbox  
specified by **value** attribute

# Parameter Data

TABLE 6.2: Values for HTML form controls (except input/file)

Control(s)	Value
<code>input/text</code> , <code>input/password</code> , <code>textarea</code>	text present in control when form is submitted
<code>input/checkbox</code> , <code>input/radio</code> , <code>input/submit</code> , <code>input/image</code> , <code>button/submit</code>	String assigned to corresponding <code>value</code> attribute. Control must be selected/clicked for parameter to be returned.
<code>input/hidden</code>	String assigned to corresponding <code>value</code> attribute.
<code>select</code>	String assigned to <code>value</code> attribute of selected <code>option(s)</code> , or content of any selected option for which <code>value</code> is not defined.

# Parameter Data

LifeStory.html - Mozilla {Build ID: 2003052908}

Enter your name:  username

Give your life's story in 100 words or less:

Check all that apply to you:  tall  funny  smart lifestory

doit

boxgroup1 (values same as labels)

# Parameter Data

- Query string produced by browser (all one line):

```
username=you&lifestory=less+is+more&boxgroup1=funny  
&boxgroup1=smart&doit=Publish+My+Life%27s+Story
```

Checkbox parameters have same name values;  
only checked boxes have corresponding parameters



# Parameter Data

- GET vs. POST method for forms:
  - GET:
    - Query string is part of URL
    - Length of query string may be limited
    - Recommended when parameter data is not stored but used only to request information (e.g., search engine query)
      - The URL can be bookmarked or emailed and the same data will be passed to the server when the URL is revisited

# Parameter Data

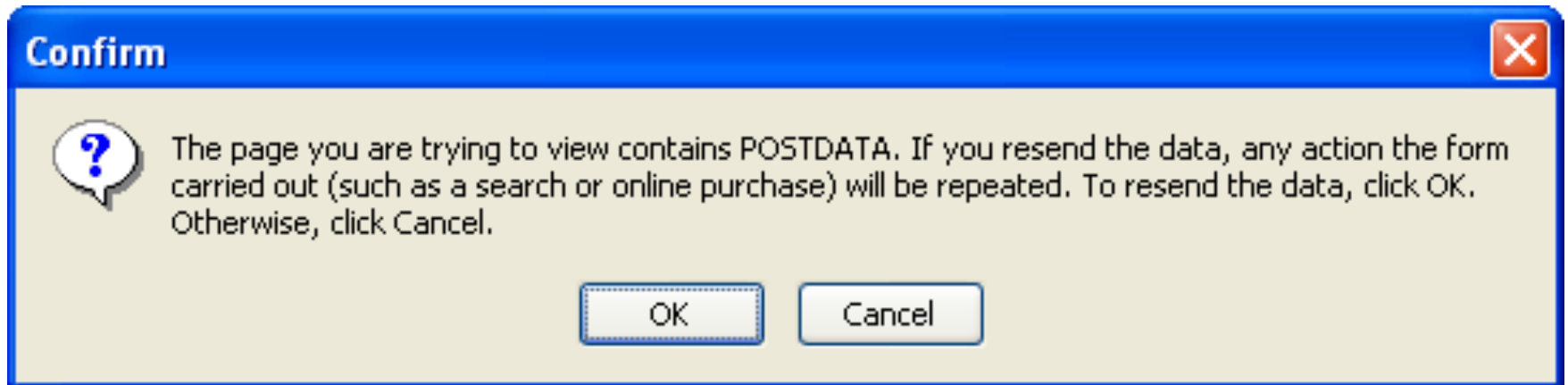
The screenshot shows a Mozilla browser window with the title "Google Search: form GET POST - Mozilla". The address bar contains the URL "http://www.google.com/search?q=form+GET+POST", with the search query "form GET POST" circled in green. The browser's navigation bar includes buttons for Back, Forward, Reload, and Stop. The main content area displays the Google logo, a search input field containing "form GET POST", and a "Google Search" button. Below the search bar, there are links for "Advanced Search", "Preferences", "Language Tools", and "Search Tips". A blue banner indicates the search results: "Searched the web for form GET POST. Results 1 - 10 of about 6,270,000. Search took 0.09 seconds." The first search result is "[thelist] [GET POST Form PHP](#)", with a snippet: "[thelist] GET POST Form PHP. rudy thelist ... 01 2003: Previous message: [thelist] GET POST Form PHP; Next message: [thelist] GET POST Form PHP; ... lists.evoit.org/archive/Week-of-Mon-20030224/135437.html - 4k - [Cached](#) - [Similar pages](#)". The second result is "[Get/Post in a form - HTML - The Complete Webmaster](#)", with a snippet: "... Get/Post in a form ... Follow Ups: Re: **Get/Post** in a **form** werert 15:37:22 2/21/103 (0): Re: **Get/Post** in a **form** werert 15:37:19 2/21/103 (0): **Post** a Followup. ... www.abiglime.com/webmaster/boards/html/posts/658.htm - 6k - [Cached](#) - [Similar pages](#)".

Browser content copyright 2004 Google, Inc. Used by permission.

# Parameter Data

- GET vs. POST method for forms:
  - POST:
    - Query string is sent as body of HTTP request
    - Length of query string is unlimited
    - Recommended if parameter data is intended to cause the server to update stored data
    - Most browsers will warn you if they are about to resubmit POST data to avoid duplicate updates

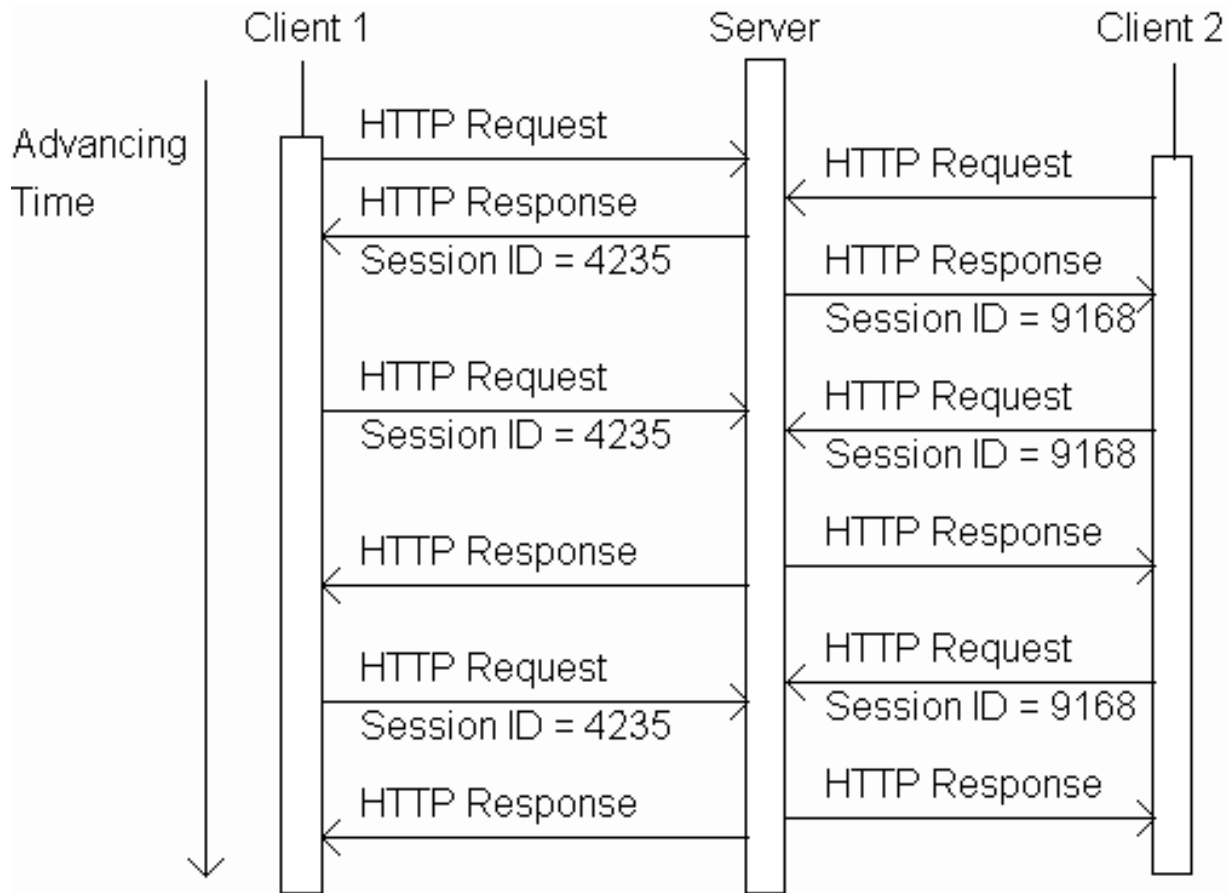
# Parameter Data



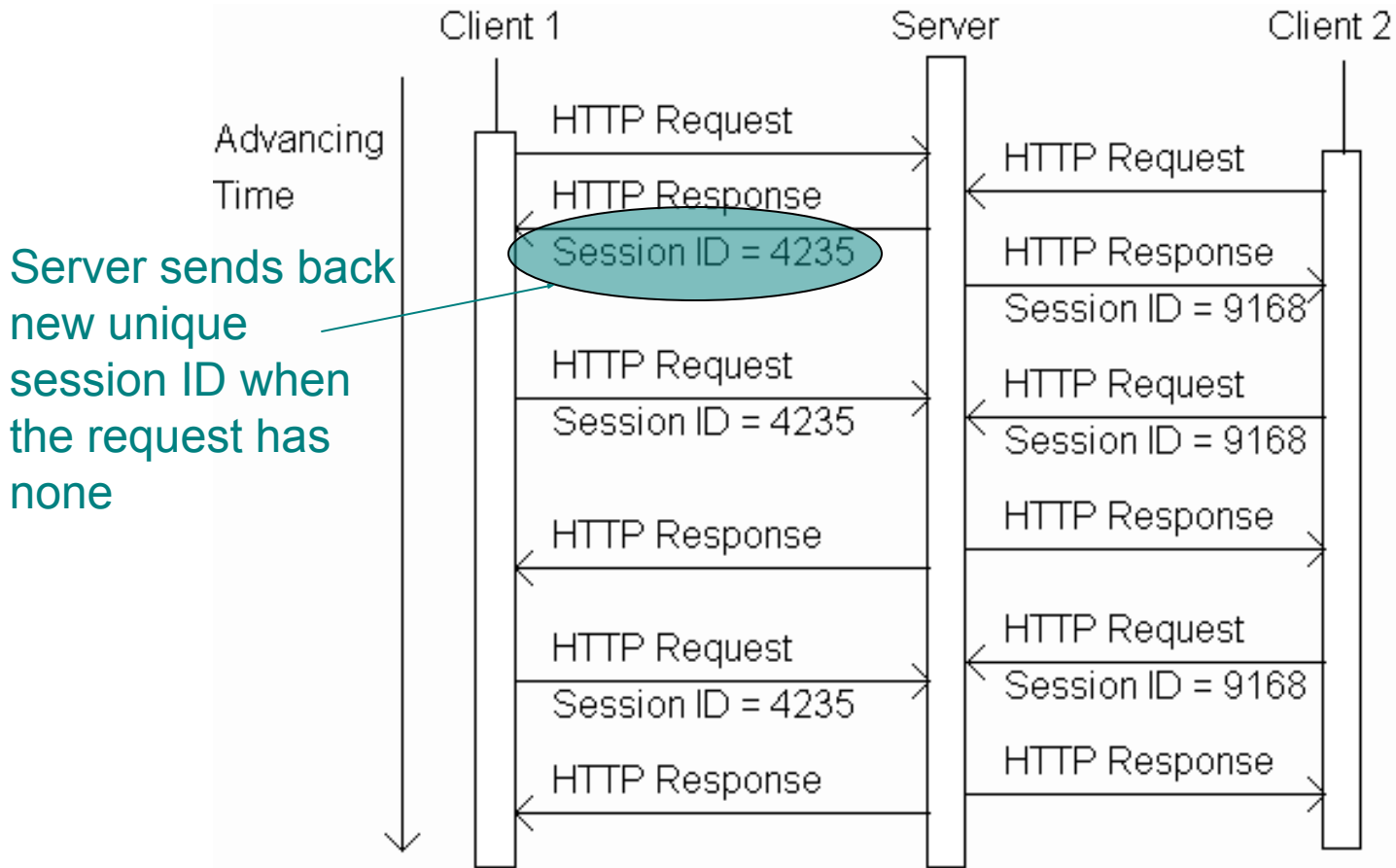
# Sessions

- Many interactive Web sites spread user data entry out over **several pages**:
  - Ex: add items to cart, enter shipping information, enter billing information
- Problem: how does the server know which users generated which HTTP requests?
  - Cannot rely on standard HTTP headers to identify a user

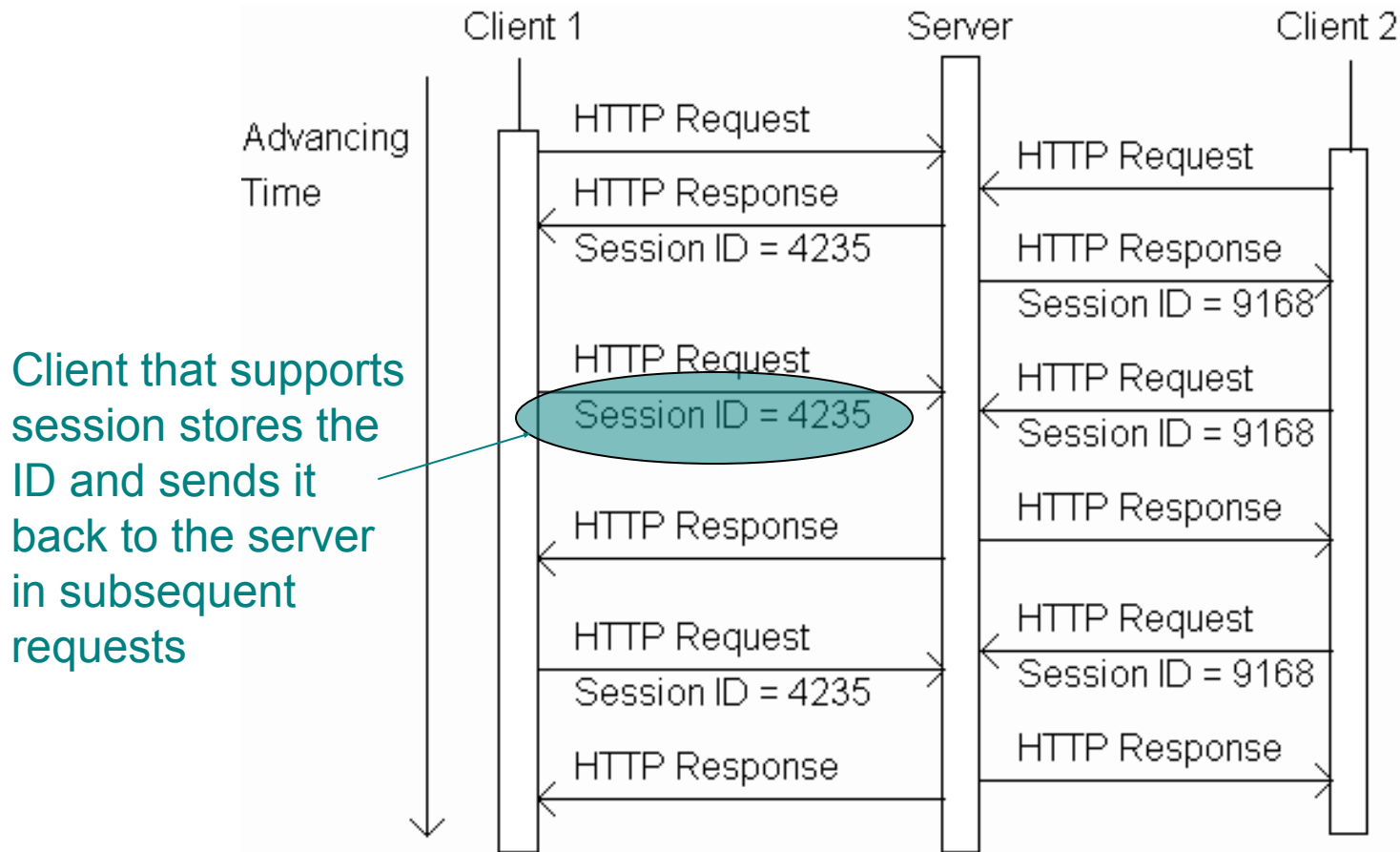
# Sessions



# Sessions

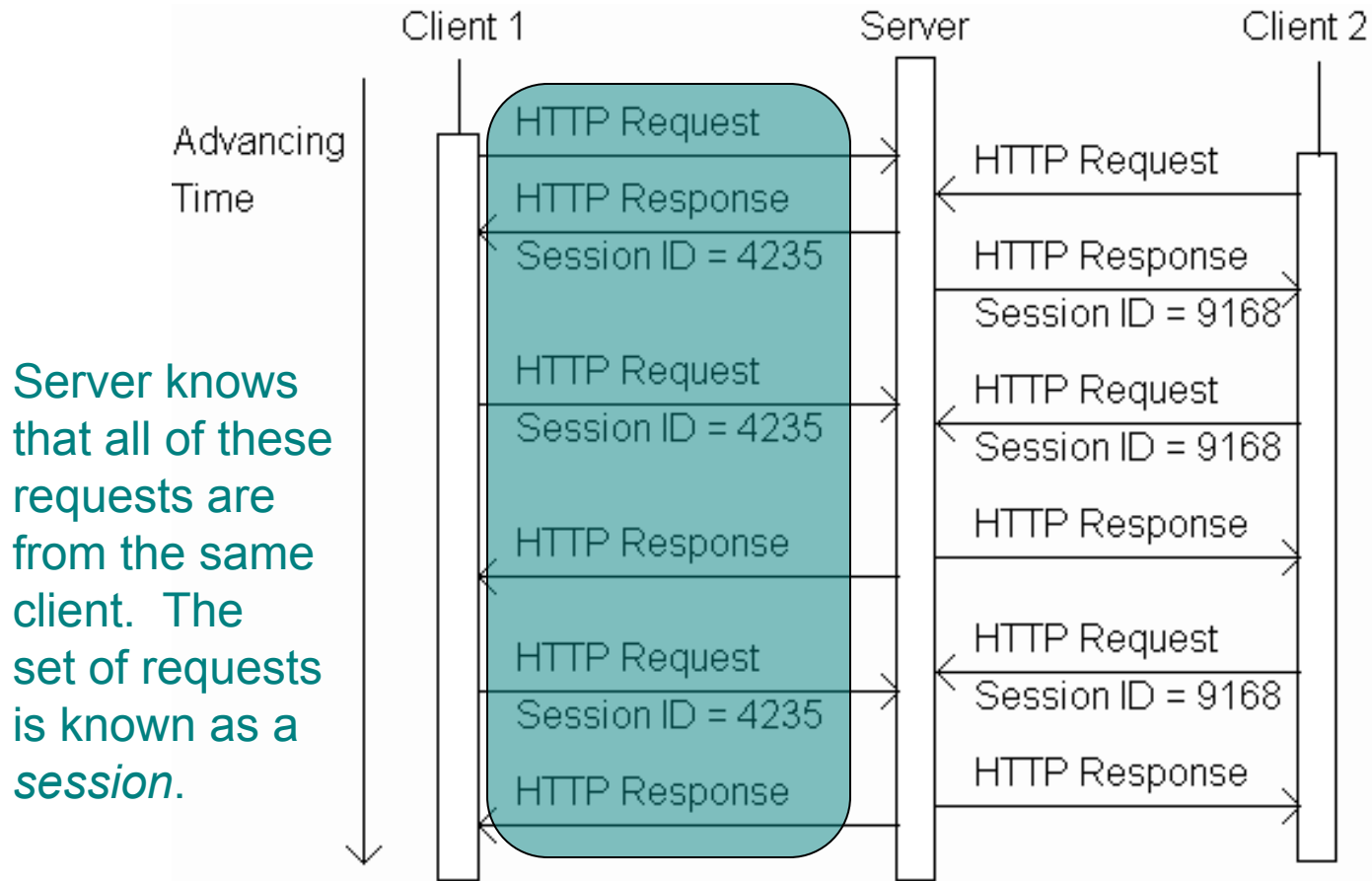


# Sessions

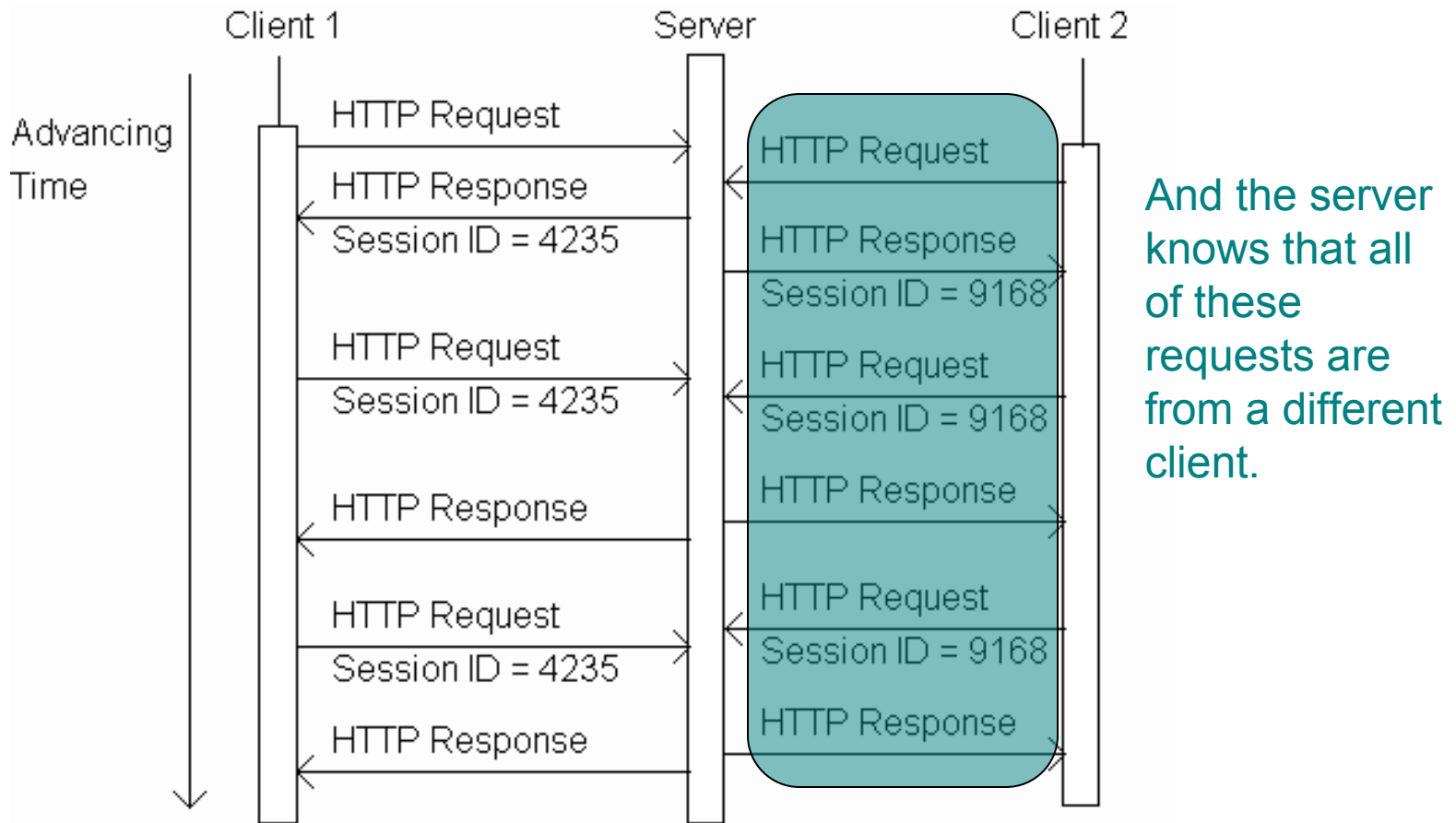




# Sessions



# Sessions



# Sessions

```
HttpSession session = request.getSession();  
if (session.isNew()) {  
    visits++;  
}
```

Returns HttpSession object associated with this HTTP request.

- Creates new HttpSession object if no session ID in request or no object with this ID exists
- Otherwise, returns previously created object

# Sessions

```
HttpSession session = request.getSession();  
if (session.isNew()) {  
    visits++;  
}
```

Boolean indicating whether returned object was newly created or already existed.

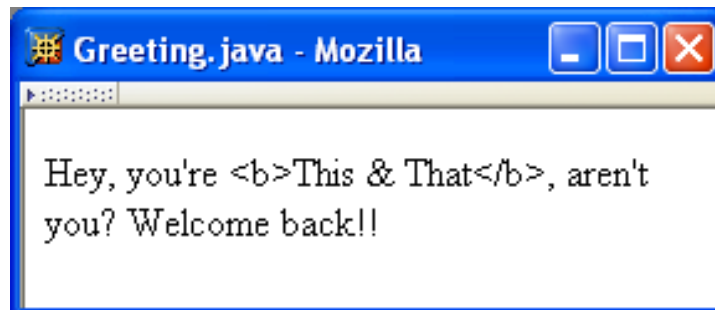
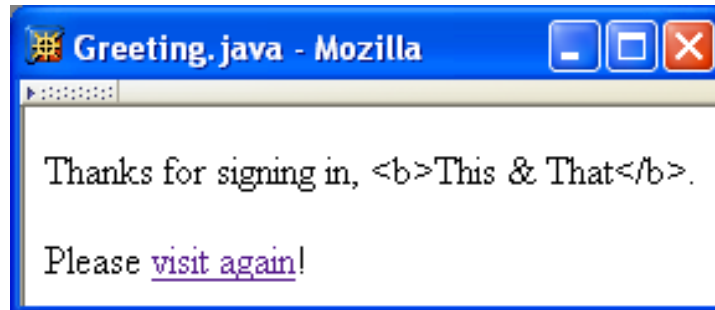
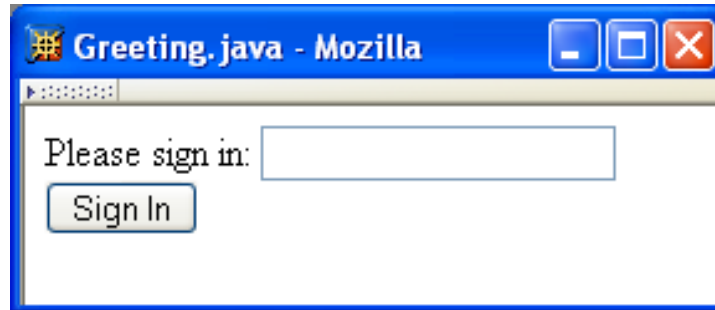
# Sessions

```
HttpSession session = request.getSession();  
if (session.isNew()) {  
    visits++;  
}
```

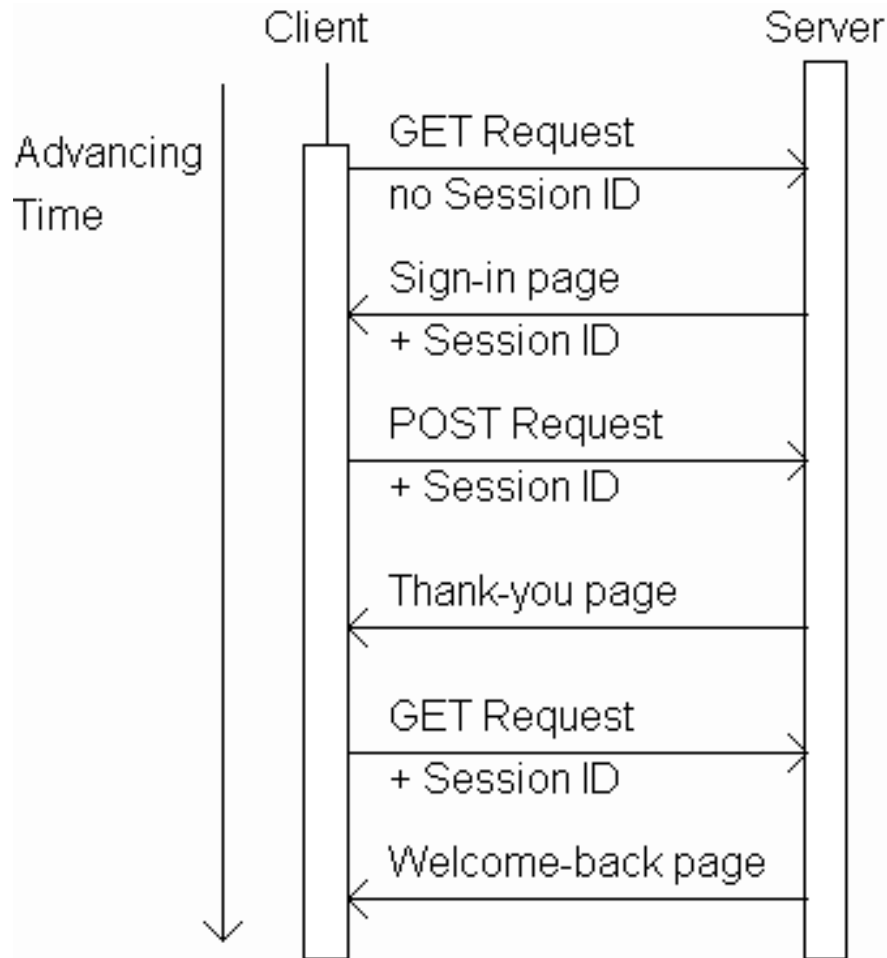
Incremented once per session

# Sessions

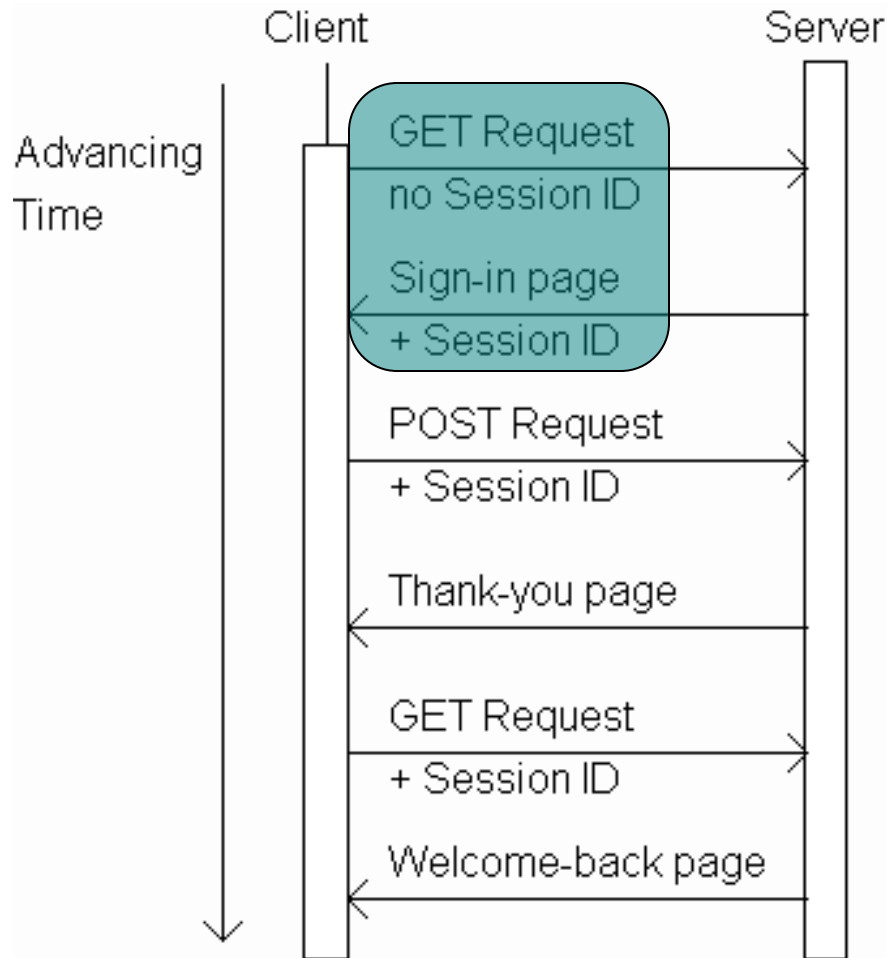
Three web pages produced by a single servlet



# Sessions

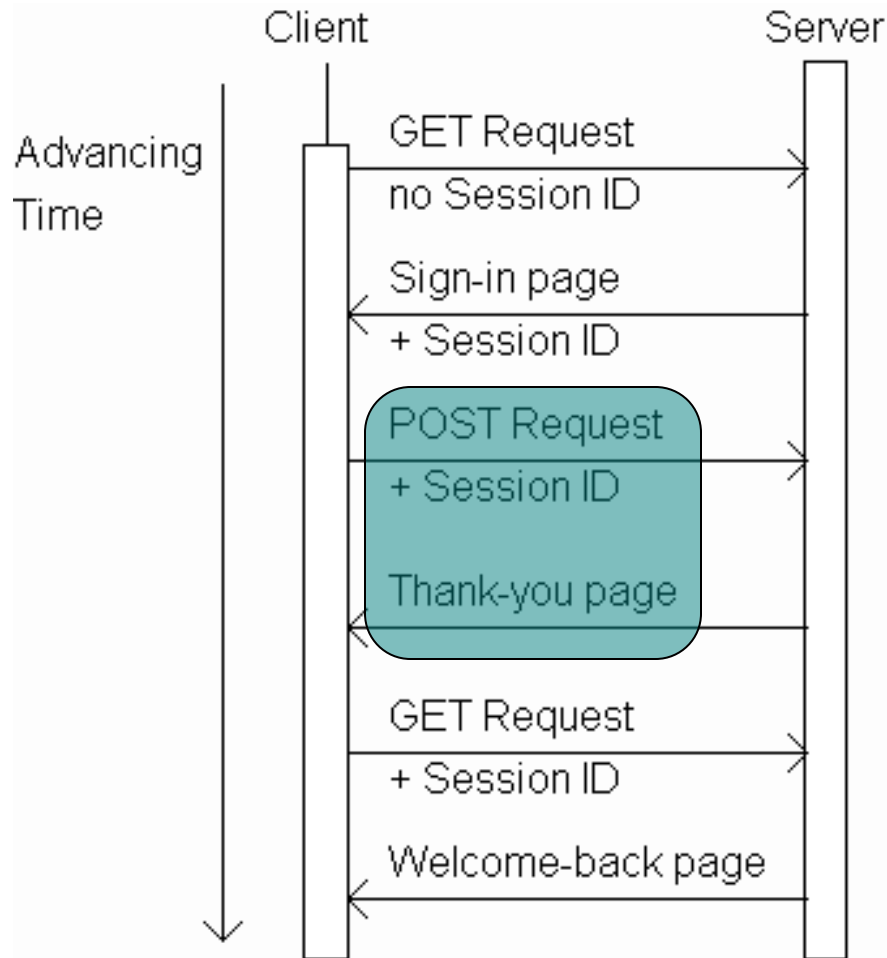


# Sessions

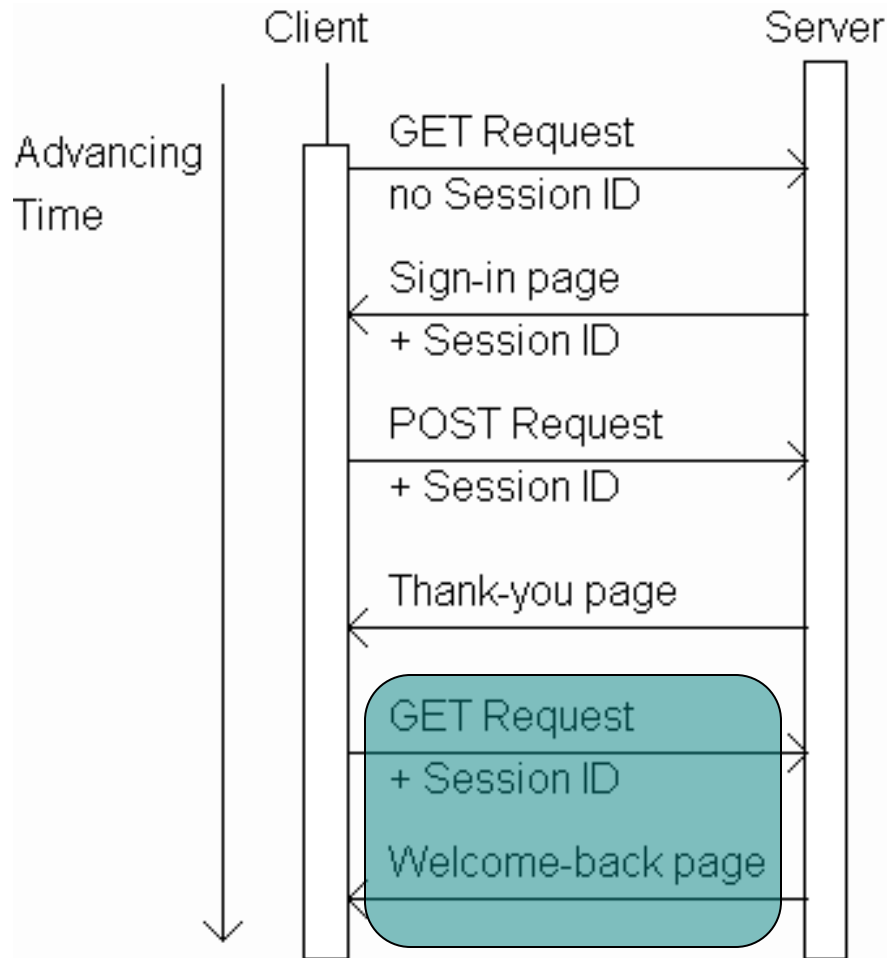




# Sessions



# Sessions



# Sessions

```
public void doGet (HttpServletRequest request,
                  HttpServletResponse response)
    ""
    HttpSession session = request.getSession();
    String signIn = (String)session.getAttribute("signIn");
    if (session.isNew() || (signIn == null)) {
        printSignInForm(servletOut, "Greeting");
    } else {
        printWelcomeBack(servletOut, signIn);
    }
}
```

# Sessions

```
public void doGet (HttpServletRequest request,  
                  HttpServletResponse response)
```

```
    {
```

```
        HttpSession session = request.getSession();  
        String signIn = (String)session.getAttribute("signIn");  
        if (session.isNew() || (signIn == null)) {  
            printSignInForm(servletOut, "Greeting");  
        } else {  
            printWelcomeBack(servletOut, signIn);  
        }  
    }
```

*Session attribute is a  
name/value pair*

# Sessions

```
public void doGet (HttpServletRequest request,  
                  HttpServletResponse response)
```

```
    {
```

```
        HttpSession session = request.getSession();
```

```
        String signIn = (String)session.getAttribute("signIn");
```

```
        if (session.isNew() || (signIn == null)) {
```

```
            printSignInForm(servletOut, "Greeting");
```

```
        } else {
```

```
            printWelcomeBack(servletOut, signIn);
```

```
        }
```

Session attribute will  
have null value until  
a value is assigned

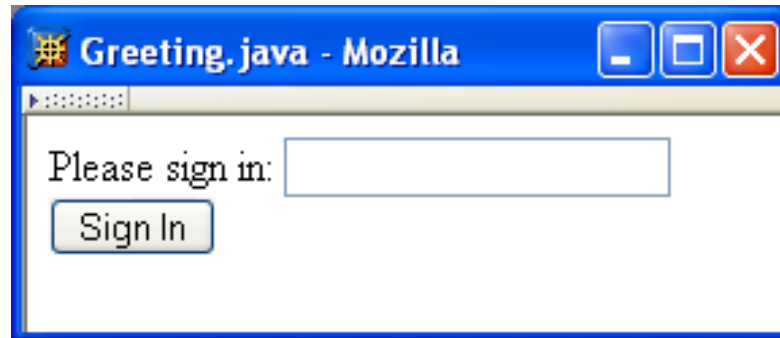
# Sessions

```
public void doGet (HttpServletRequest request,
                  HttpServletResponse response)
{
    ""
    HttpSession session = request.getSession();
    String signIn = (String)session.getAttribute("signIn");
    if (session.isNew() || (signIn == null)) {
        printSignInForm(servletOut, "Greeting");
    } else {
        printWelcomeBack(servletOut, signIn);
    }
}
```

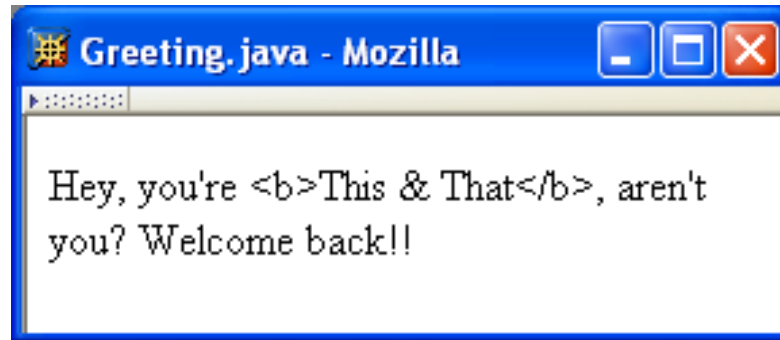
Generate  
sign-in form  
if session is  
new or  
**signIn**  
attribute has no value,  
welcome-back page  
otherwise.

# Sessions

Sign-in form



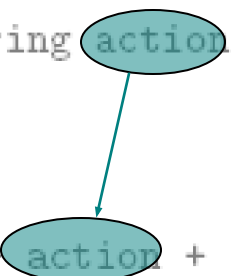
Welcome-back page



# Sessions

```
private void printSignInForm(PrintWriter servletOut,  
                             String action) {  
    ...  
    servletOut.println(  
"    <form method='post' action='" + action + "'><div> \n" +  
"    <label> \n" +  
"        Please sign in: <input type='text' name='signIn' /> \n" +  
    ...  
}
```

Second argument  
("Greeting") used as  
action attribute value  
(relative URL)

A diagram consisting of two light blue ovals. The top oval contains the word 'action' from the method signature. A blue arrow points from this oval down to a second oval containing the word 'action' from the HTML code snippet, illustrating how the parameter is passed to the action attribute.



# Sessions

```
private void printSignInForm(PrintWriter servletOut,
                             String action)
{
    ...
    servletOut.println(
"    <form method=post action='" + action + "'><div> \n" +
"    <label> \n" +
"        Please sign in: <input type='text' name='signIn' /> \n" +
    ...

```

Form will be sent using POST HTTP method (doPost() method will be called)

# Sessions

```
private void printSignInForm(PrintWriter servletOut,
                             String action)
{
    ...
    servletOut.println(
"    <form method='post' action='" + action + "'><div> \n" +
"    <label> \n" +
"    Please sign in: <input type='text' name='signIn' /> \n" +
    ...

```

Text field containing  
user name is named  
**signIn**

# Sessions

```
public void doPost (HttpServletRequest request,
                   HttpServletResponse response)
    ...
    String signIn = request.getParameter("signIn");
    HttpSession session = request.getSession();
    if (signIn != null) {
        printThanks(servletOut, signIn, "Greeting");
        session.setAttribute("signIn", signIn);
    } else {
        printSignInForm(servletOut, "Greeting");
    }
}
```

# Sessions

```
public void doPost (HttpServletRequest request,  
                   HttpServletResponse response)
```

```
...
```

Retrieve  
signIn  
parameter value

```
{ String signIn = request.getParameter("signIn");  
  HttpSession session = request.getSession();  
  if (signIn != null) {  
      printThanks(servletOut, signIn, "Greeting");  
      session.setAttribute("signIn", signIn);  
  } else {  
      printSignInForm(servletOut, "Greeting");  
  }  
}
```

# Sessions

```
public void doPost (HttpServletRequest request,
                   HttpServletResponse response)
    ...
    String signIn = request.getParameter("signIn");
    HttpSession session = request.getSession();
    if (signIn != null) {
        printThanks(servletOut, signIn, "Greeting");
        session.setAttribute("signIn", signIn);
    } else {
        printSignInForm(servletOut, "Greeting");
    }
}
```

Normal  
processing:  
**signIn**  
parameter  
is present in  
HTTP request

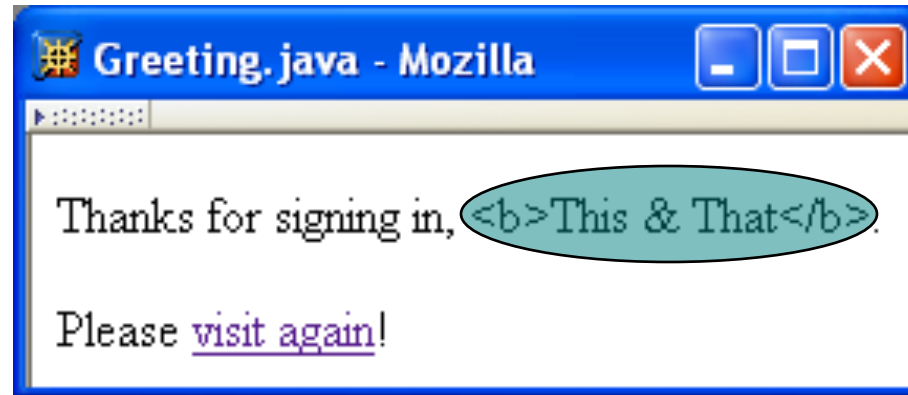
# Sessions

```
public void doPost (HttpServletRequest request,
                   HttpServletResponse response)
    ...
    String signIn = request.getParameter("signIn");
    HttpSession session = request.getSession();
    if (signIn != null) {
        printThanks(servletOut, signIn, "Greeting");
        session.setAttribute("signIn", signIn);
    } else {
        printSignInForm(servletOut, "Greeting");
    }
}
```

Generate  
HTML for  
response

# Sessions

Thank-you page



Must escape XML special characters in user input

# Sessions

```
public void doPost (HttpServletRequest request,
                   HttpServletResponse response)
    ...
    String signIn = request.getParameter("signIn");
    HttpSession session = request.getSession();
    if (signIn != null) {
        printThanks(servletOut, signIn, "Greeting");
        session.setAttribute("signIn", signIn);
    } else {
        printSignInForm(servletOut, "Greeting");
    }
}
```

Assign a  
value to the  
**signIn** session  
attribute {



# Sessions

- Session attribute methods:
  - **setAttribute(String name, Object value):** creates a session attribute with the given name and value
  - **Object getAttribute(String name):** returns the value of the session attribute named name, or returns null if this session does not have an attribute with this name

# Sessions

```
public void doPost (HttpServletRequest request,
                   HttpServletResponse response)
    ...
    String signIn = request.getParameter("signIn");
    HttpSession session = request.getSession();
    if (signIn != null) {
        printThanks(servletOut, signIn, "Greeting");
        session.setAttribute("signIn", signIn);
    } else {
        printSignInForm(servletOut, "Greeting");
    }
}
```

Error  
processing  
(return user  
to sign-in form)

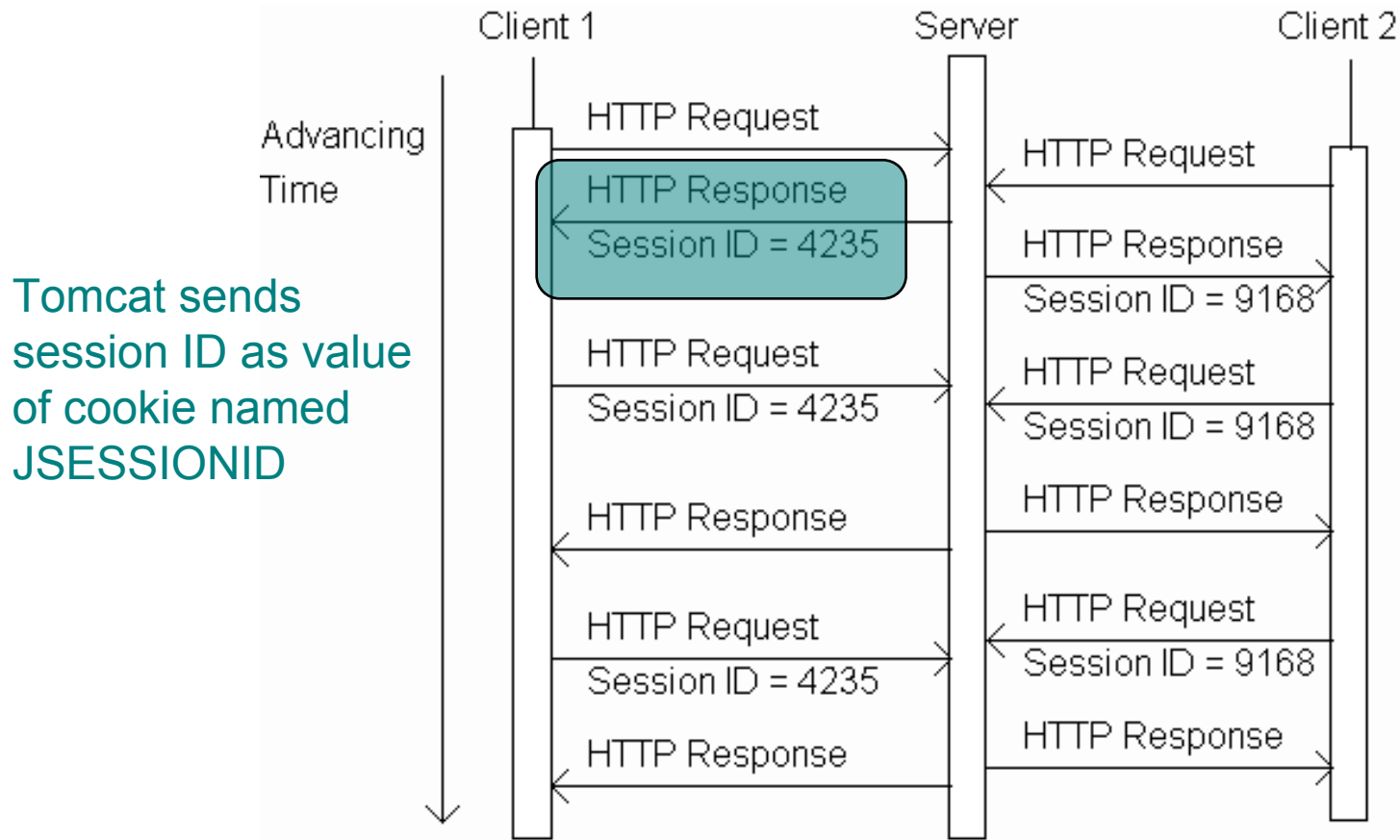
# Sessions

- By default, each session **expires** if a server-determined length of time elapses between a session's HTTP requests
  - Server destroys the corresponding session object
- Servlet code can:
  - **Terminate** a session by calling `invalidate()` method on session object
  - Set the **expiration time-out duration** (secs) by calling `setMaxInactiveInterval(int)`

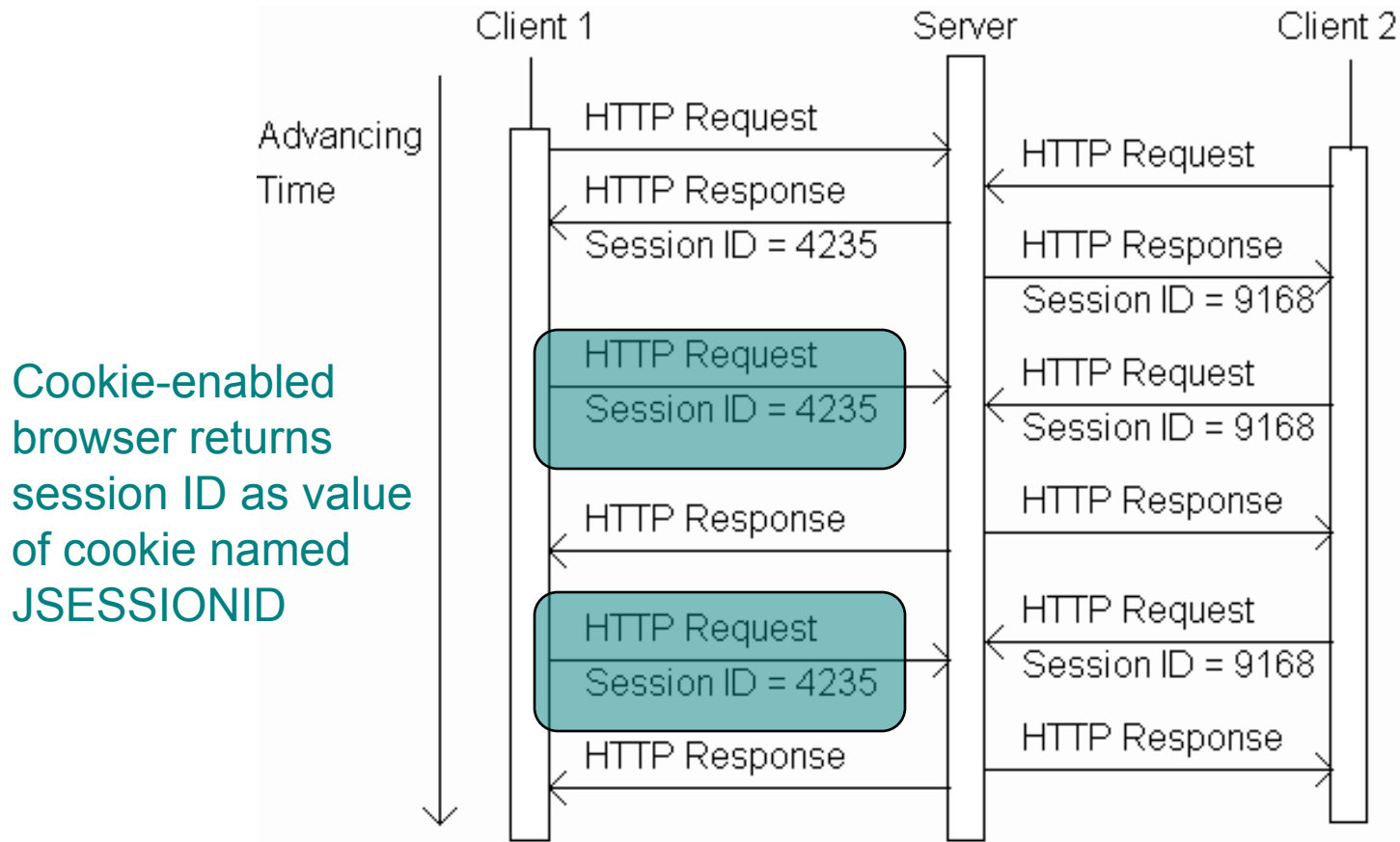
# Cookies

- A **cookie** is a name/value pair in the Set-Cookie header field of an HTTP response
- Most (not all) clients will:
  - **Store** each cookie received in its file system
  - **Send** each cookie back to the server that sent it as part of the Cookie header field of subsequent HTTP requests

# Cookies



# Cookies



# Cookies

- Servlets can set cookies explicitly
  - `Cookie` class used to represent cookies
  - `request.getCookies()` returns an array of `Cookie` instances representing cookie data in HTTP request
  - `response.addCookie(Cookie)` adds a cookie to the HTTP response

# Cookies

TABLE 6.3: Key Cookie class methods.

Method	Purpose
<code>Cookie(String name, String value)</code>	Constructor to create a cookie with given name and value
<code>String getName()</code>	Return name of this cookie
<code>String getValue()</code>	Return value of this cookie
<code>void setMaxAge(int seconds)</code> <b>Cookies are expired by client (server can request expiration date)</b>	Set delay until cookie expires. Positive value is delay in seconds, negative value means that the cookie expires when the browser closes, and 0 means delete the cookie.



# Cookies

```
public void doGet (HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException
{
    // Get count from cookie if available, otherwise
    // use initial value.
    int count = 0;
    Cookie[] cookies = request.getCookies();
    if (cookies != null) {
        for (int i=0; (i<cookies.length) && (count==0); i++) {
            if (cookies[i].getName().equals("COUNT")) {
                count = Integer.parseInt(cookies[i].getValue());
            }
        }
    }
}
```

# Cookies

```
public void doGet (HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException
{
    // Get count from cookie if available, otherwise
    // use initial value.
    int count = 0;
    Cookie[] cookies = request.getCookies();
    if (cookies != null) {
        for (int i=0; (i<cookies.length) && (count==0); i++) {
            if (cookies[i].getName().equals("COUNT")) {
                count = Integer.parseInt(cookies[i].getValue());
            }
        }
    }
}
```

Return array of cookies  
contained in HTTP request

# Cookies

```
public void doGet (HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException
{
    // Get count from cookie if available, otherwise
    // use initial value.
    int count = 0;
    Cookie[] cookies = request.getCookies();
    if (cookies != null) {
        for (int i=0; (i<cookies.length) && (count==0); i++) {
            if (cookies[i].getName().equals("COUNT")) {
                count = Integer.parseInt(cookies[i].getValue());
            }
        }
    }
}
```

Search for  
cookie  
named  
COUNT and  
extract value  
as an int

# Cookies

```
// Increment the count and add request to client to store it
// for one year.
count++;
Cookie cookie = new Cookie("COUNT",
                           new Integer(count).toString());
cookie.setMaxAge(oneYear);
response.addCookie(cookie);

// Set the HTTP content type in response header
response.setContentType("text/html; charset=\"UTF-8\"");
. . .
" <body> \n" +
"   <p>You have visited this page " + count + " time(s) \n" +
"     in the past year, or since clearing your cookies.</p> \n" +
" </body> \n" +
```

# Cookies

Send replacement cookie value to client (overwrites existing cookie)

```
// Increment the count and add request to client to store it
// for one year.
count++;
Cookie cookie = new Cookie("COUNT",
                           new Integer(count).toString());
cookie.setMaxAge(oneYear);
response.addCookie(cookie);

// Set the HTTP content type in response header
response.setContentType("text/html; charset=\"UTF-8\"");
. . .
" <body> \n" +
"   <p>You have visited this page " + count + " time(s) \n" +
"     in the past year, or since clearing your cookies.</p> \n" +
" </body> \n" +
```

# Cookies

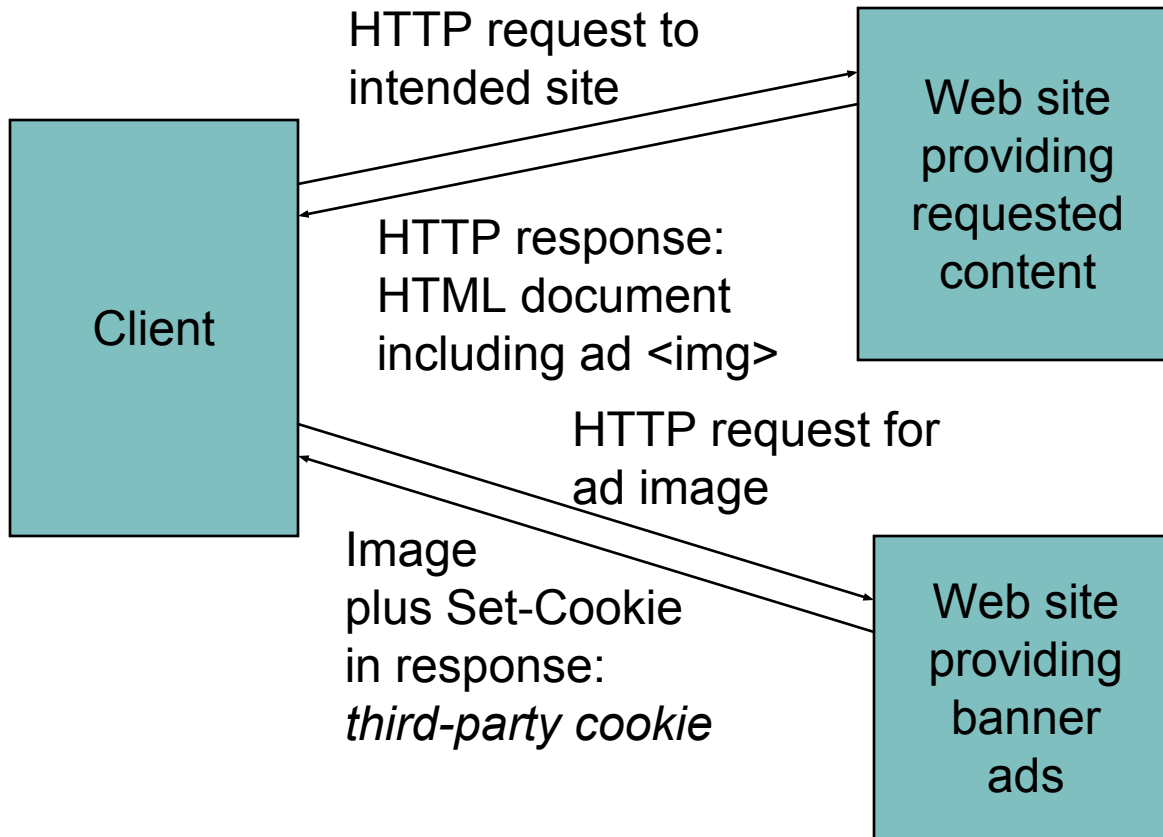
```
// Increment the count and add request to client to store it
// for one year.
count++;
Cookie cookie = new Cookie("COUNT",
                           new Integer(count).toString());
cookie.setMaxAge(oneYear);
response.addCookie(cookie);

// Set the HTTP content type in response header
response.setContentType("text/html; charset=\"UTF-8\"");
. . .
" <body> \n" +
"   <p>You have visited this page " + count + " time(s) \n" +
"     in the past year, or since clearing your cookies.</p> \n" +
" </body> \n" +
```

Should call  
addCookie()  
before writing  
HTML

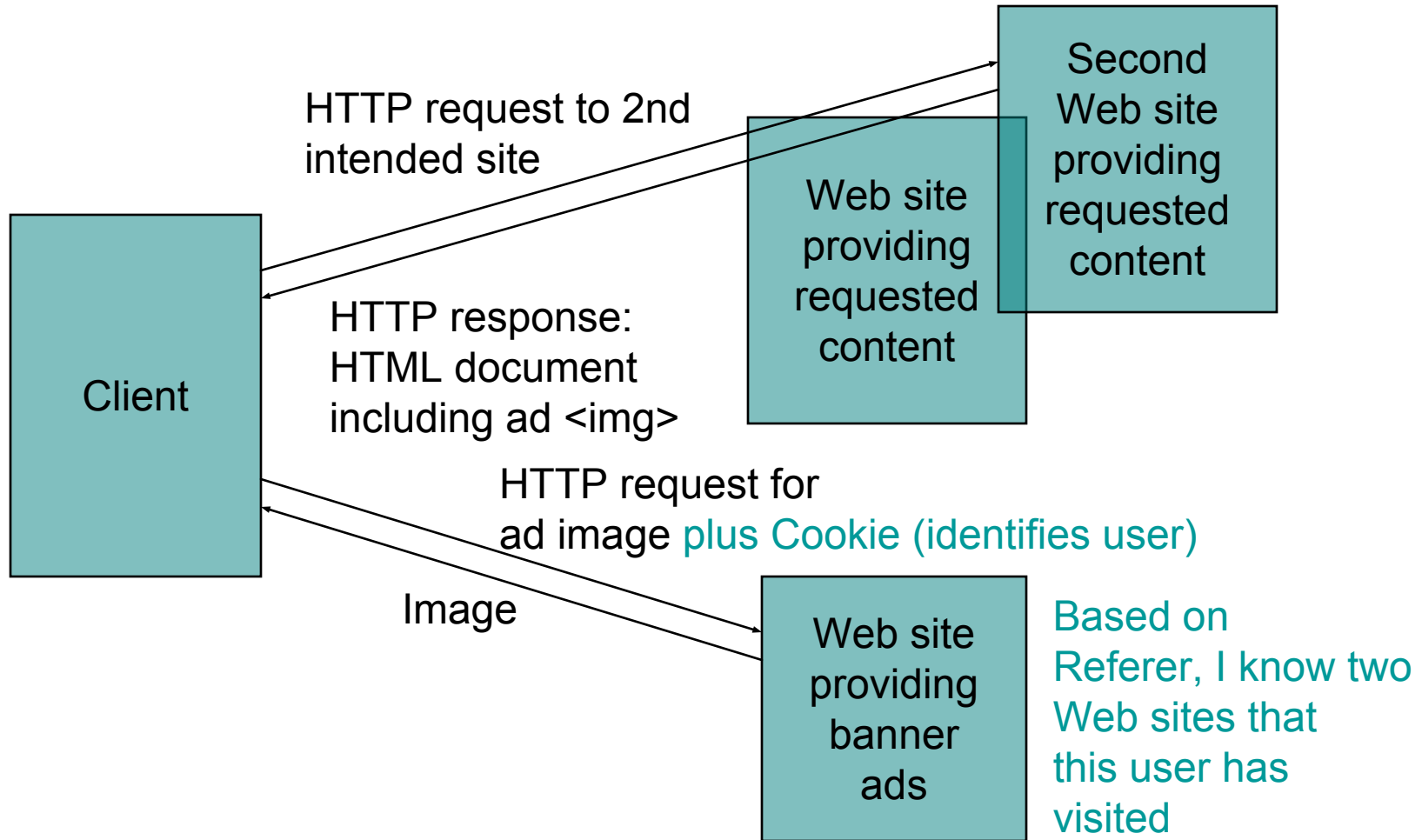
# Cookies

## Privacy issues



# Cookies

## Privacy issues





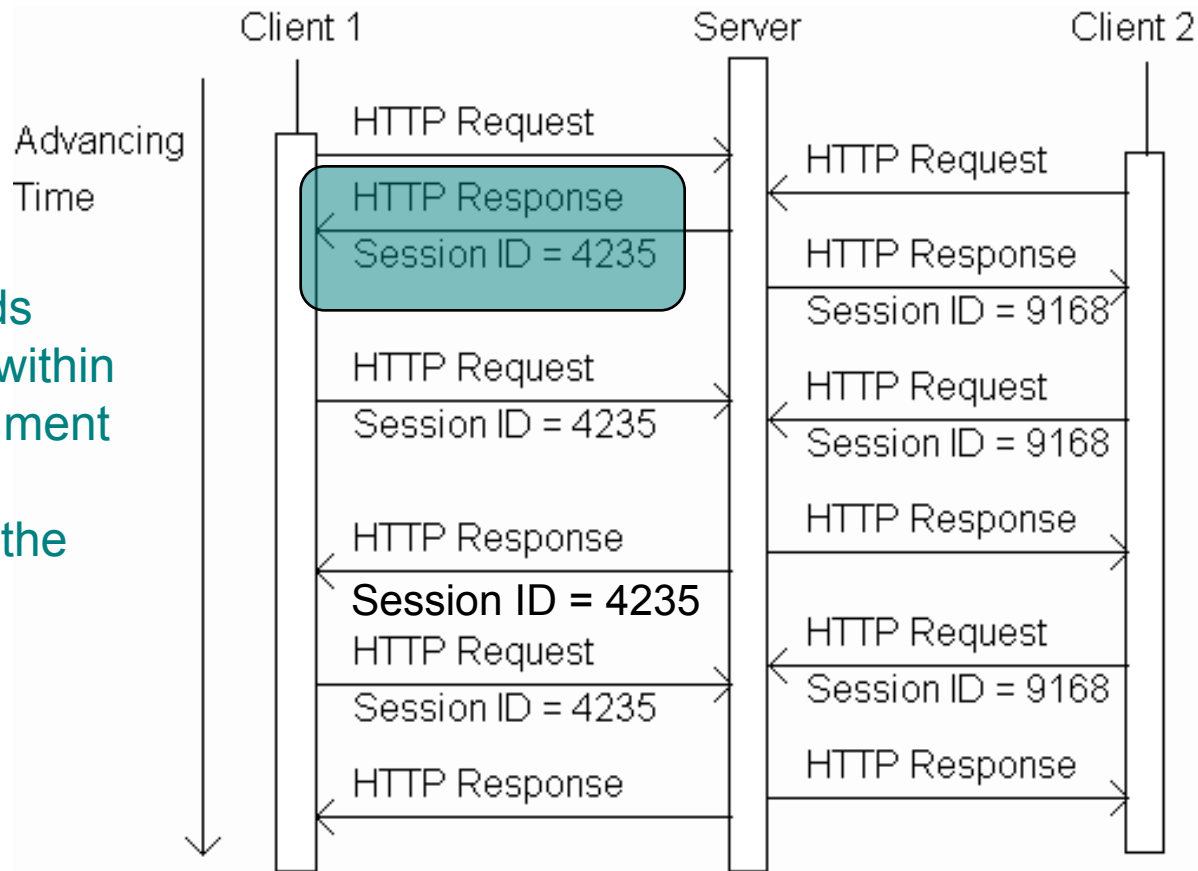
# Cookies

## Privacy issues

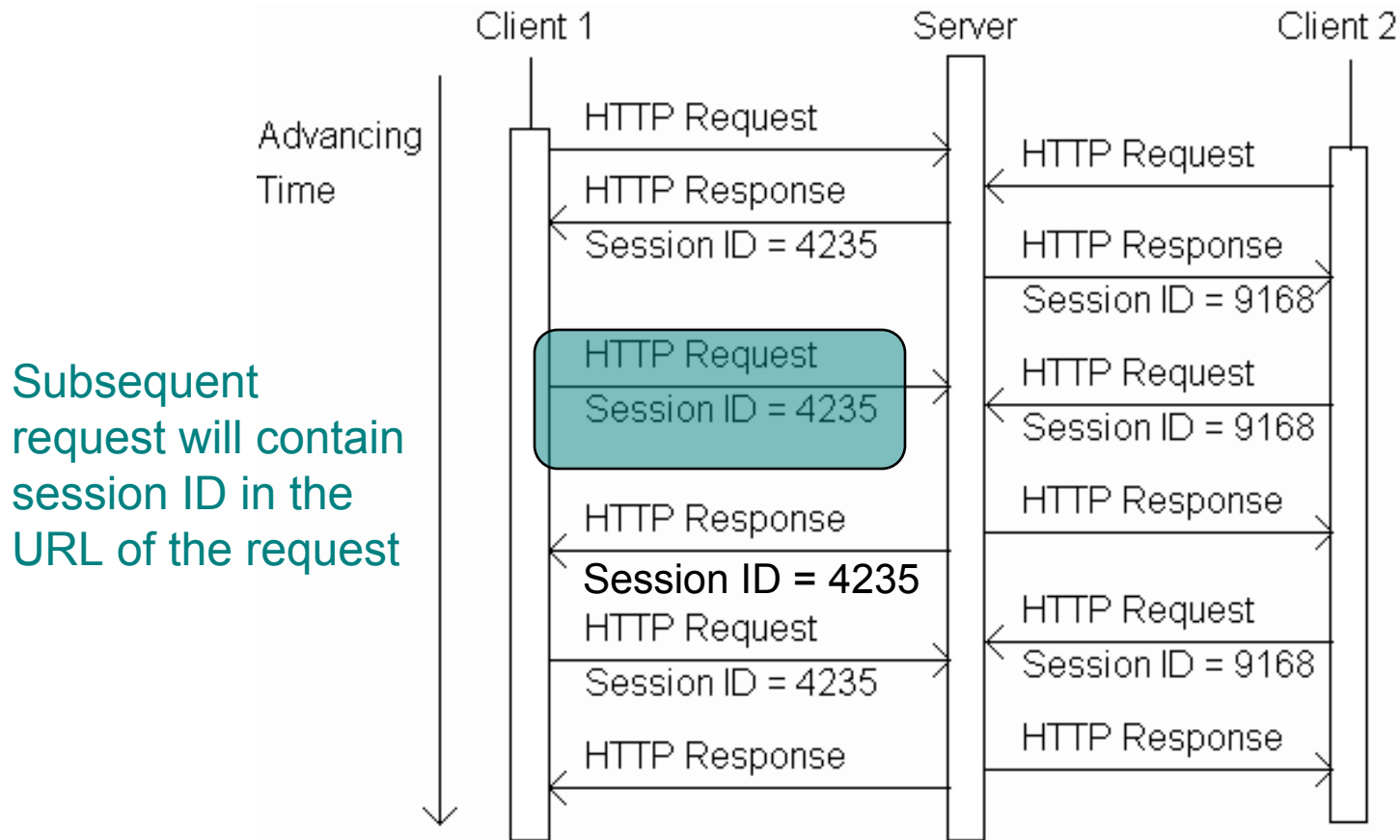
- Due to privacy concerns, many users **block** cookies
  - Blocking may be fine-tuned. Ex: Mozilla allows
    - Blocking of third-party cookies
    - Blocking based on on-line privacy policy
- Alternative to cookies for maintaining session: **URL rewriting**

# URL Rewriting

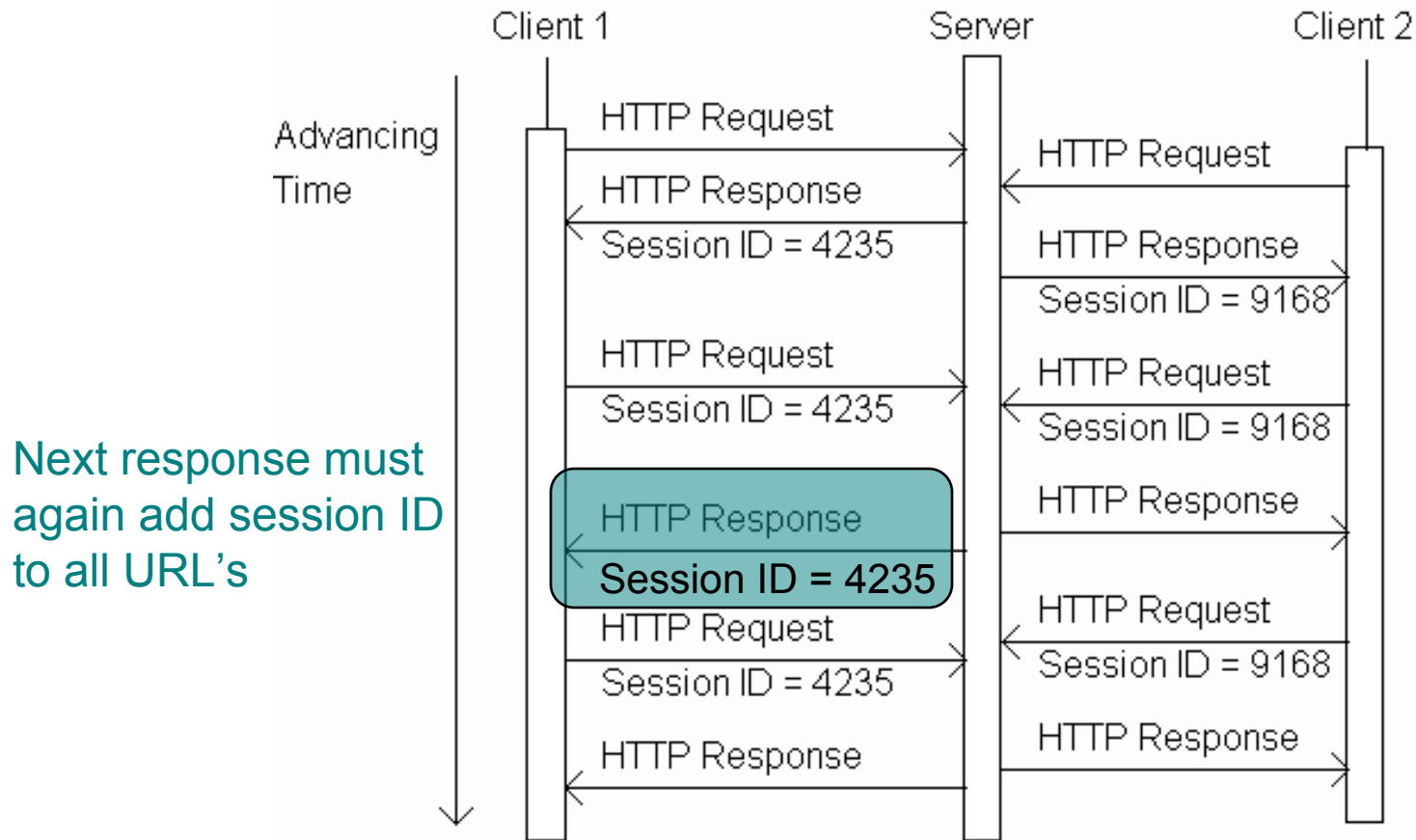
Tomcat adds session ID within HTML document to all URL's referring to the servlet



# URL Rewriting



# URL Rewriting



# URL Rewriting

- Original (relative) URL:

href="URLEncodedGreeting"

- URL containing session ID:

href="URLEncodedGreeting;jsessionId=0157B9E85"



*Path parameter*

- Path parameter is treated differently than query string parameter
  - Ex: invisible to `getParameter()`

# URL Rewriting

- `HttpServletResponse` method `encodeURL()` will add session id path parameter to argument URL

Original  
servlet

```
printSignInForm(servletOut, "Greeting");
```

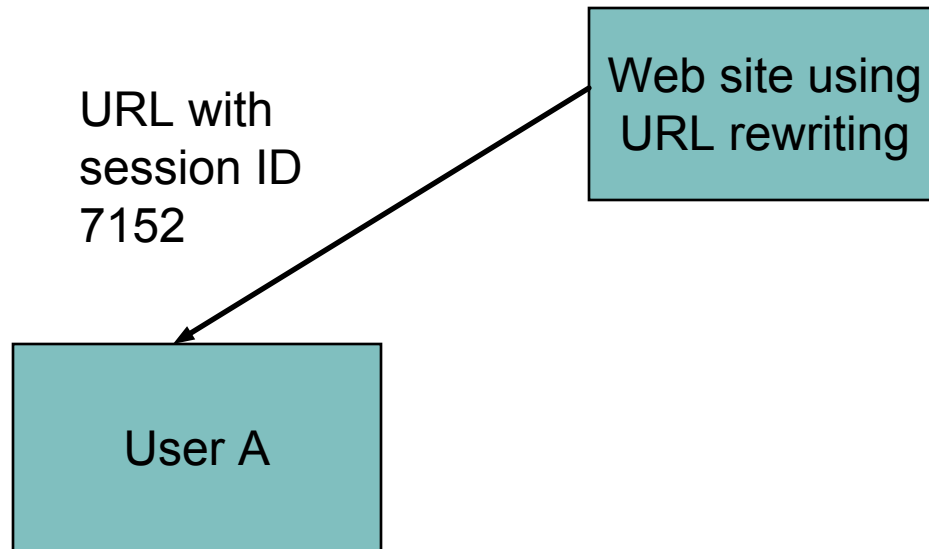
Relative URL of servlet

Servlet  
using URL  
rewriting

```
printSignInForm(servletOut,  
                response.encodeURL("URLEncodedGreeting"));
```

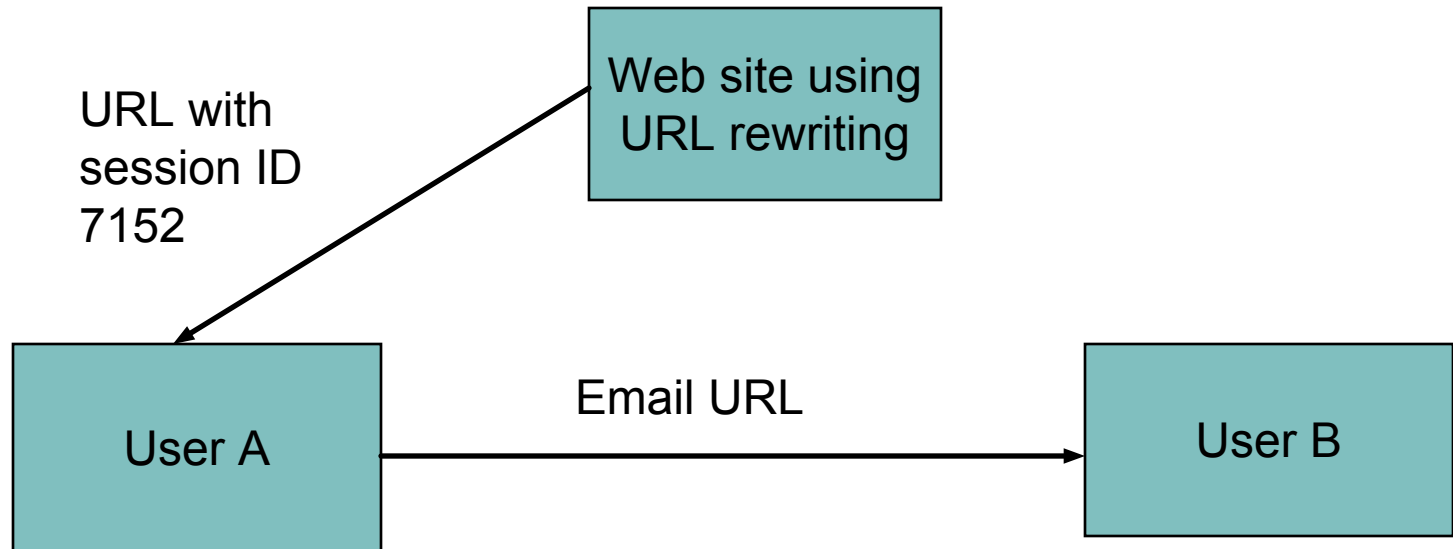
# URL Rewriting

- Must rewrite *every* servlet URL in *every* document
- Security issues



# URL Rewriting

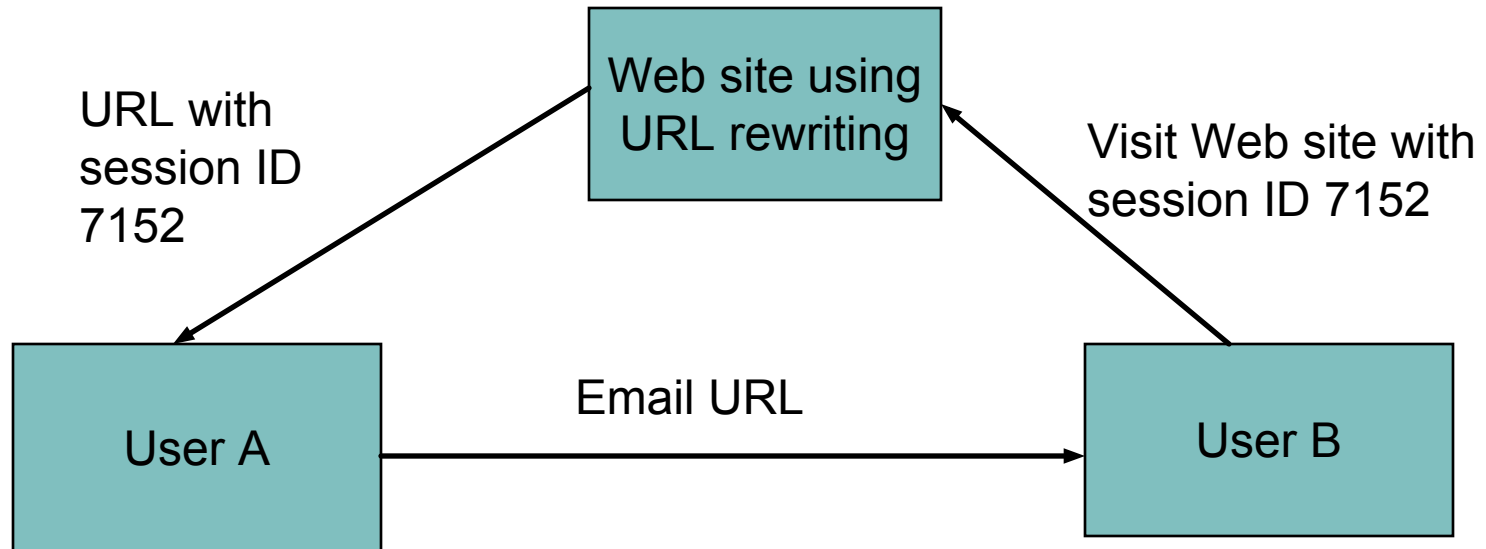
- Must rewrite *every* servlet URL in *every* document
- Security issues





# URL Rewriting

- Must rewrite *every* servlet URL in *every* document
- Security issues

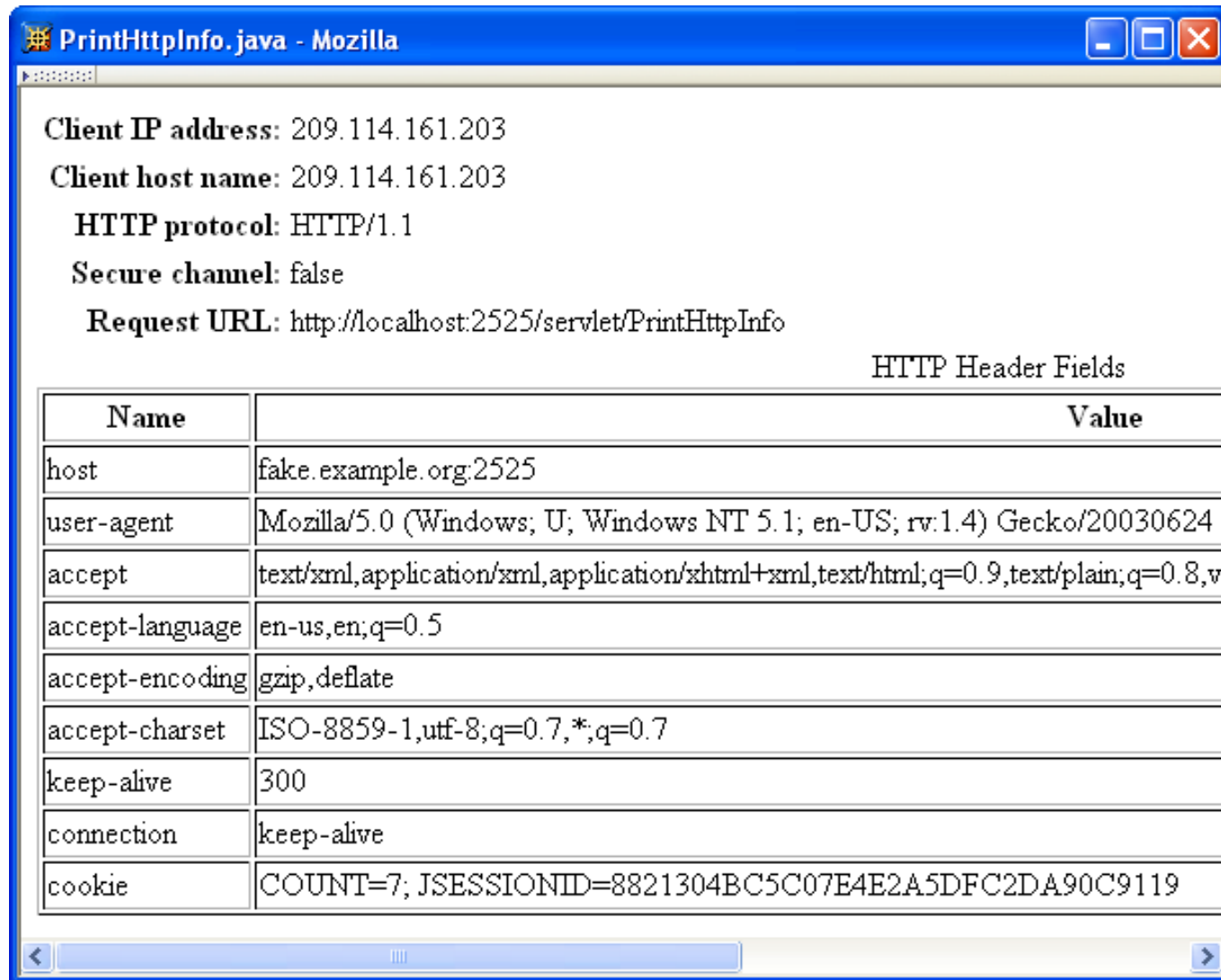


# More Servlet Methods

TABLE 6.4: Additional `HttpServletRequest` methods.

Method	Purpose
<code>String getRemoteAddr()</code>	Return IP address of the client machine making this request.
<code>String getRemoteHost()</code>	Return fully qualified name of the client making this request, or its IP address if name is not available.
<code>String getProtocol()</code>	Return the type and version of communication protocol used by the client to make this request (e.g., "HTTP/1.1").
<code>boolean isSecure()</code>	Return boolean indicating whether or not this request was made over a secure communication channel.
<code>StringBuffer getRequestURL()</code>	Return a <code>StringBuffer</code> containing the URL used to access this servlet, excluding any query string appended to the URL as well as any <code>jsessionid</code> path parameter.
<code>Enumeration getHeaderNames()</code>	Return an <code>Enumeration</code> of <code>String</code> objects representing names of all header fields in the request.
<code>String getHeader(String fieldName)</code>	Given a valid header field name, return a <code>String</code> representing the value of the field, or <code>null</code> if header field is not present in request. The match of <code>fieldName</code> against header field names is case-insensitive.

# More Servlet Methods



PrintHttpInfo.java - Mozilla

Client IP address: 209.114.161.203  
Client host name: 209.114.161.203  
HTTP protocol: HTTP/1.1  
Secure channel: false  
Request URL: http://localhost:2525/servlet/PrintHttpInfo

HTTP Header Fields

Name	Value
host	fake.example.org:2525
user-agent	Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.4) Gecko/20030624
accept	text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,v
accept-language	en-us,en;q=0.5
accept-encoding	gzip,deflate
accept-charset	ISO-8859-1,utf-8;q=0.7,*,q=0.7
keep-alive	300
connection	keep-alive
cookie	COUNT=7; JSESSIONID=8821304BC5C07E4E2A5DFC2DA90C9119

# More Servlet Methods

TABLE 6.5: Additional HttpServletResponse methods.

Method	Purpose
<code>void setHeader(String name, String value)</code>	Include a header field with the given <code>name</code> and <code>value</code> in the HTTP response.
<code>void setDateHeader(String name, long value)</code>	Include a date header field (such as Expires) with the given <code>name</code> in the HTTP response. The given <code>value</code> is converted from milliseconds since 00:00 01 January 1970 UTC to an equivalent time in HTTP date format.
<code>void setContentLength(int len)</code>	Set the Content-Length header field to the given value.
<code>void setBufferSize(int size)</code>	Set the desired size of the response buffer (see below). The server may override the specified <code>size</code> and use a larger value. This method must be called before any data is written into the response buffer.
<code>int getBufferSize()</code>	Return an integer representing the actual size of the response buffer.

# More Servlet Methods

- **Response buffer**
  - All data sent to the **PrintWriter** object is stored in a buffer
  - When the buffer is full, it is automatically **flushed**:
    - Contents are sent to the client (preceded by header fields, if this is the first flush)
    - Buffer becomes empty
  - Note that all header fields must be defined before the first buffer flush

# More Servlet Methods

TABLE 6.5: Additional `HttpServletResponse` methods.

<code>void setStatus(int statusCode)</code>	Set the status code in the HTTP response (status code is 200 (OK) by default). Any information contained in the response buffer is cleared. Use only for non-error status codes.
<code>void sendError(int statusCode, String msg)</code>	Set the status code in the HTTP response to the given error <code>statusCode</code> (status code beginning with 4 or 5), and in the body of the response send a server-generated HTML error page containing the given <code>msg</code> .
<code>void sendRedirect(String url)</code>	Cause HTTP response with status code 307 (Temporary Redirect) to be sent to the client, causing the client to send a new HTTP request to the given <code>url</code> . Client will behave as if it had sent request to the specified <code>url</code> .
<code>void encodeRedirectURL(String url)</code>	Perform URL rewriting (for session management) on <code>url</code> that will be used for redirection.

# More Servlet Methods

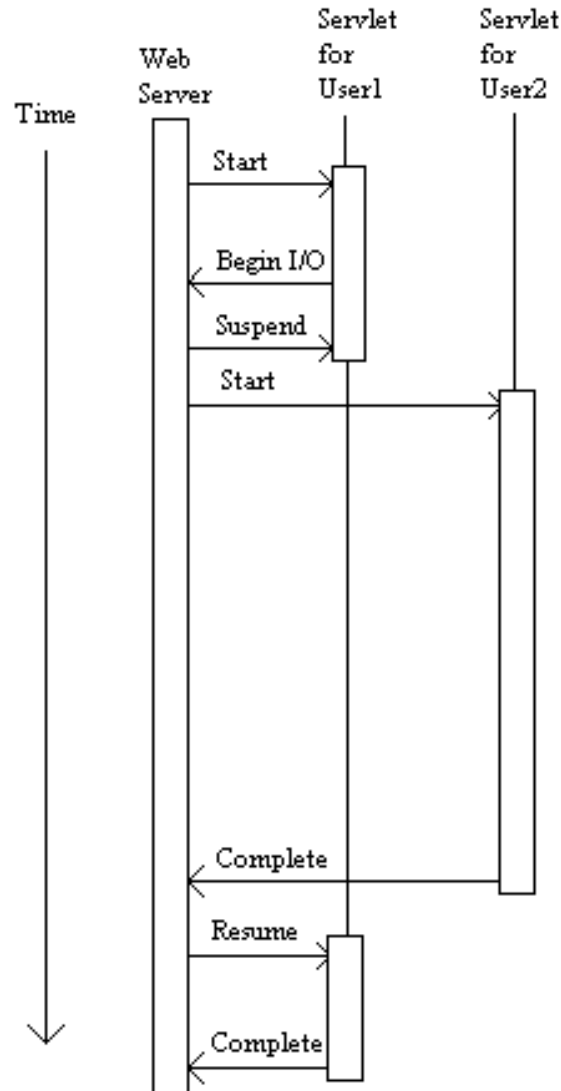
- In addition to **doGet()** and **doPost()**, servlets have methods corresponding to other HTTP request methods
  - **doHead()**: automatically defined if **doGet()** is overridden
  - **doOptions()**, **doTrace()**: useful default methods provided
  - **doDelete()**, **doPut()**: override to support these methods

# Data Storage

- Almost all **web applications** (servlets or related dynamic web server software) store and retrieve data
  - Typical web app uses a data base management system (DBMS)
  - Another option is to use the file system
  - Not web technologies, so beyond our scope
- Some Java data storage details provided in Appendices B (file system) and C (DBMS)
- One common problem: **concurrency**



# Concurrency



# Concurrency

- Tomcat creates a separate **thread** for each HTTP request
- Java thread state saved:
  - Which statement to be executed next
  - The **call stack**: where the current method will return to, where that method will return to, *etc.* plus parameter values for each method
  - The values of local variables for all methods on the call stack

# Concurrency

- Some examples of values that are *not* saved when a thread is suspended:
  - Values of **instance variables** (variables declared outside of methods)
  - Values of **class variables** (variables declared as static outside of methods)
  - Contents of **files** and other external resources

# Concurrency

```
public class HelloCounter extends HttpServlet
{
    // Number of times the servlet has been executed since
    // the program (web server) started
    private int visits=0;

    [...] // removed doGet() declaration and initialization

        // Obtain a PrintWriter object for creating the body
        // of the response
        PrintWriter servletOut = response.getWriter();

        // Compute the number of visits to the URL for this servlet
        visits++;

        // Output HTML document
```

# Concurrency

<u>User1 Thread</u>	<u>User2 Thread</u>
<started>	
.	
.	
.	
visits++;	
<visits now 18>	
<suspended>	
	<started>
	.
	.
	.
	visits++;
	<visits now 19>
	servletOut.println(...
	visits + ... );
	<outputs 19>
	<completed>
<resumed>	
servletOut.println(...	
visits + ... );	
<outputs 19>	

# Concurrency

- Java support thread synchronization

```
synchronized public void doGet (HttpServletRequest request,  
                                 HttpServletResponse response)
```

Only one thread at  
at time can call `doGet()`

- Only one synchronized method within a class  
can be called at any one time

# Concurrency

<u>User1 Thread</u>	<u>User2 Thread</u>
<code>&lt;started&gt;</code>	
<code>...</code>	
<code>synchMethod();</code>	
<code>    synchronized void</code>	
<code>    synchMethod() {</code>	
<code>        ...</code>	
<code>    &lt;suspended,</code>	
<code>    holding lock&gt;</code>	
<code>    </code>	<code>&lt;started&gt;</code>
<code>    </code>	<code>...</code>
<code>    </code>	<code>synchMethod();</code>
<code>    </code>	<code>&lt;blocked, waiting</code>
<code>    </code>	<code>for lock&gt;</code>
<code>    </code>	
<code>    &lt;resumed&gt;</code>	
<code>    </code>	
<code>    </code>	
<code>    ... </code>	
<code>    } // end of method</code>	
	<code>&lt;unblocked&gt;</code>
	<code>synchronized void</code>
	<code>synchMethod() {</code>
	<code>...</code>

# Concurrency

- Web application with multiple servlet classes and shared resource:

```
Servlet 1 (CounterReader)
-----
Input count from file (24)
<suspended>

<resumed>
Overwrite file with 0
Display count (24)
```

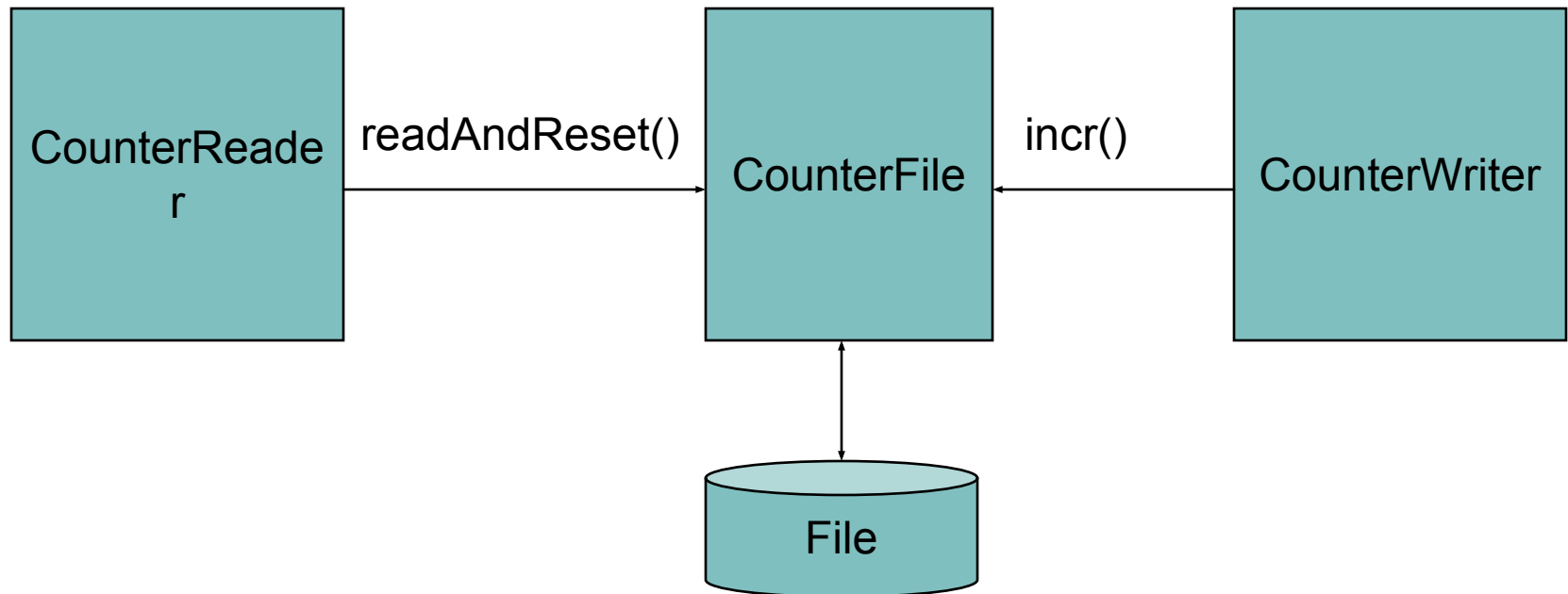
```
Servlet 2 (CounterWriter)
-----

Input count from file (24)
Increment count (25)
Overwrite file with count (25)
Display "Hello World!" page
```



# Concurrency

- Solution: create a shared class with synchronized static methods called by both servlets



# Common Gateway Interface

- **CGI** was the earliest standard technology used for dynamic server-side content
- CGI basics:
  - HTTP request information is stored in environment variables (*e.g.*, `QUERY_STRING`, `REQUEST_METHOD`, `HTTP_USER_AGENT`)
  - Program is executed, output is returned in HTTP response

# Common Gateway Interface

- Advantage:
  - Program can be written in any programming language (**Perl** frequently used)
- Disadvantages:
  - No standard for concepts such as session
  - May be slower (programs normally run in separate processes, not server process)