Server-side Programming: Java Servlets

Server-side Programming

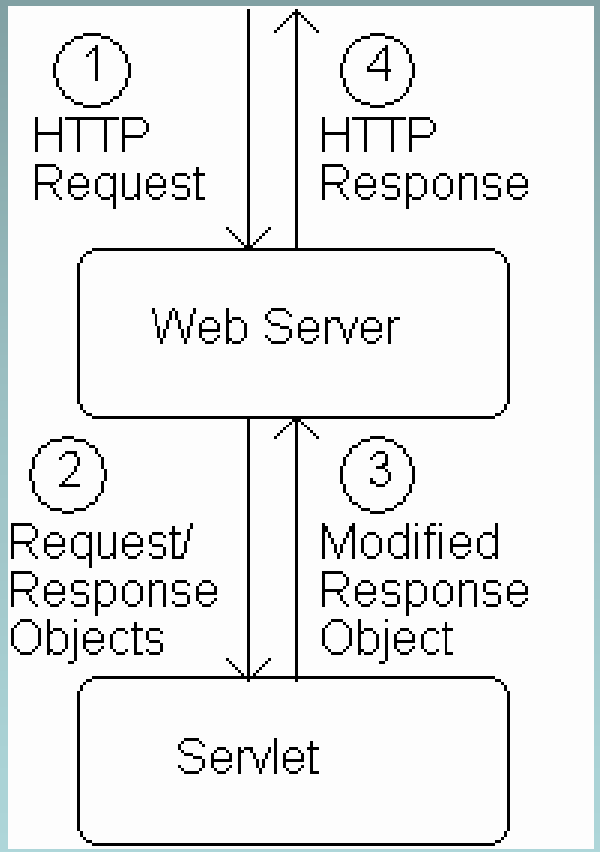
- The combination of
  - HTML
  - JavaScript
  - DOM

  is 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 Java 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:
  - Writing a trace for the exception in `logs/jwsdp_log.*.txt` log file
  - Returning HTML page to client that may (or may not) contain partial exception trace
- If servlet prints a stack trace itself by calling `printStackTrace()`, or if it writes debugging output to `System.out or System.err`, this output will be appended to the file `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

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
```

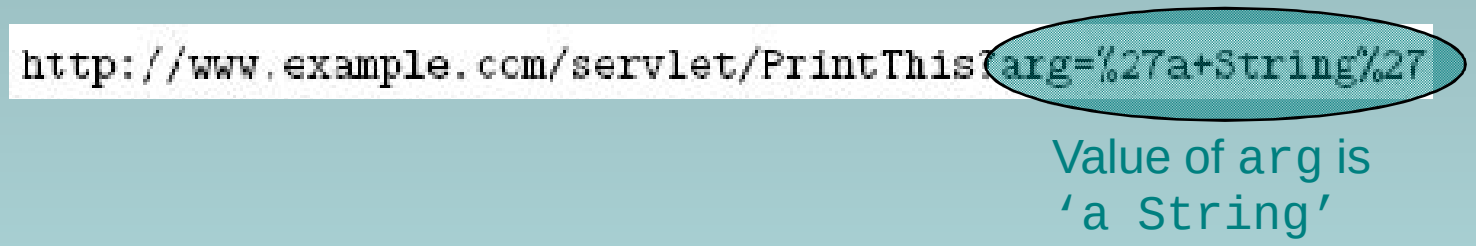
  - All parameter values are strings

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

Value of `arg` is empty string

Parameter Data

- A parameter name or value can be any sequence of 8-bit characters
- URL encoding is used to represent non-alphanumeric characters:



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

Parameter Data

- URL encoding algorithm

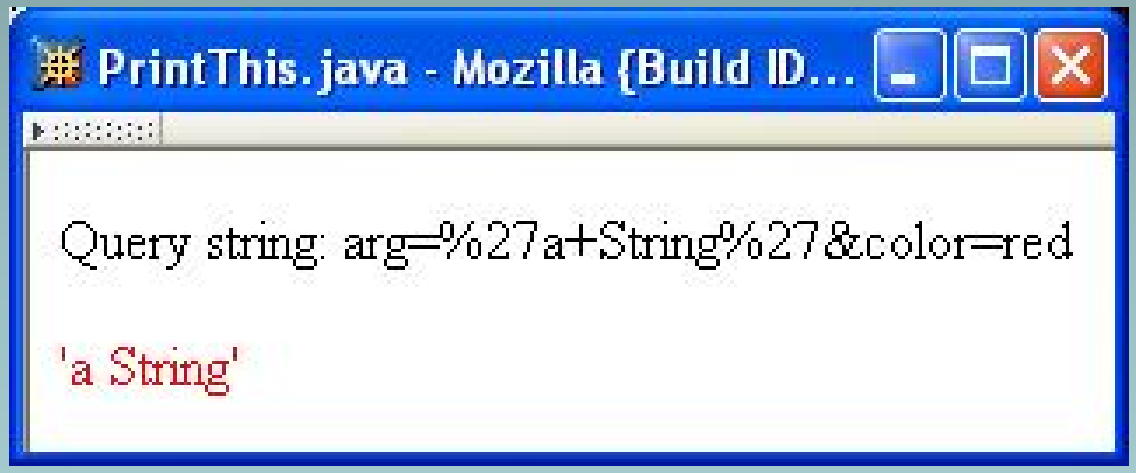
```
initialize the result to the empty string
for each 8-bit character in the original string
  if the character is an alphanumeric
    concatenate the character to the result
  else if the character is a space
    concatenate a plus sign (+) to the result
  else
    concatenate a percent sign (%) followed 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
`String getQueryString()`	Returns the entire query string in its original (URL encoded) form.
`Enumeration getParameterNames()`	Returns Enumeration of String values representing all parameter names (URL decoded) in the query string.
`String getParameter (String name)`	Returns String representing value (URL decoded) of parameter named **name**, or **null** if parameter is not present in the query string.
`String[] getParameterValues (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)) +
"'> " );
          }
```

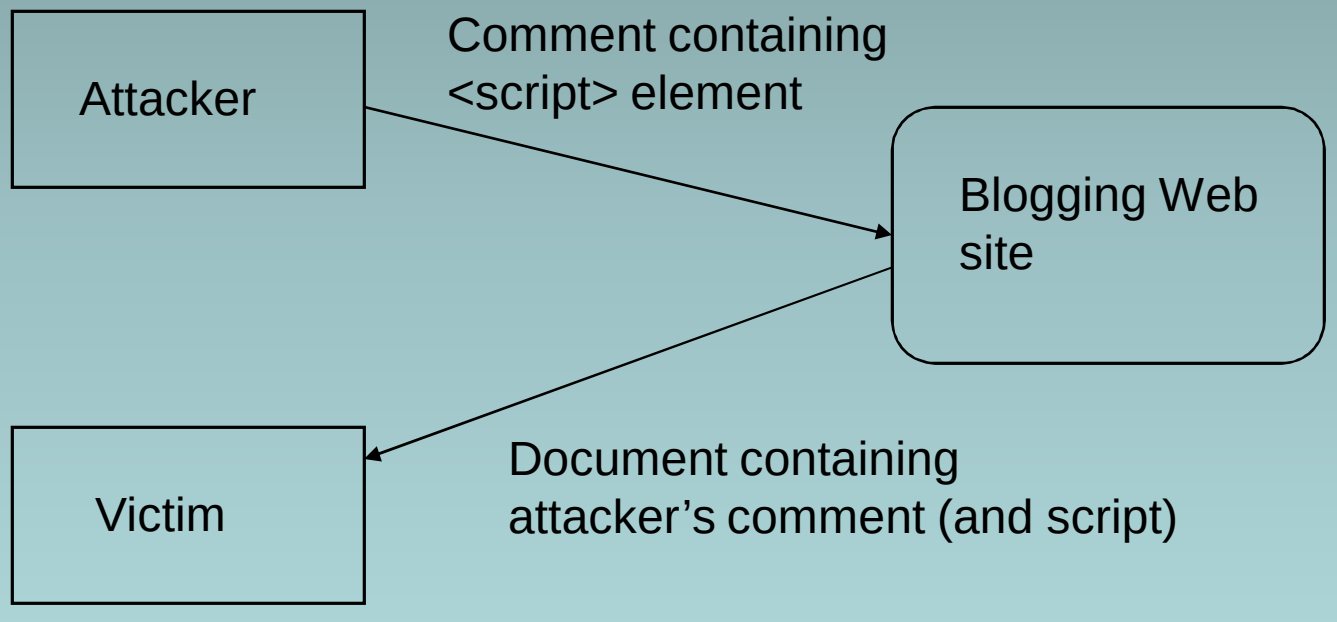
Parameter Data

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

```
<input type="text" name="username" size="40" />
```

  - Parameter value depends on control type

```
<label>
  <input type="checkbox" name="boxgroup1" value="tall" />tall
</label>
```

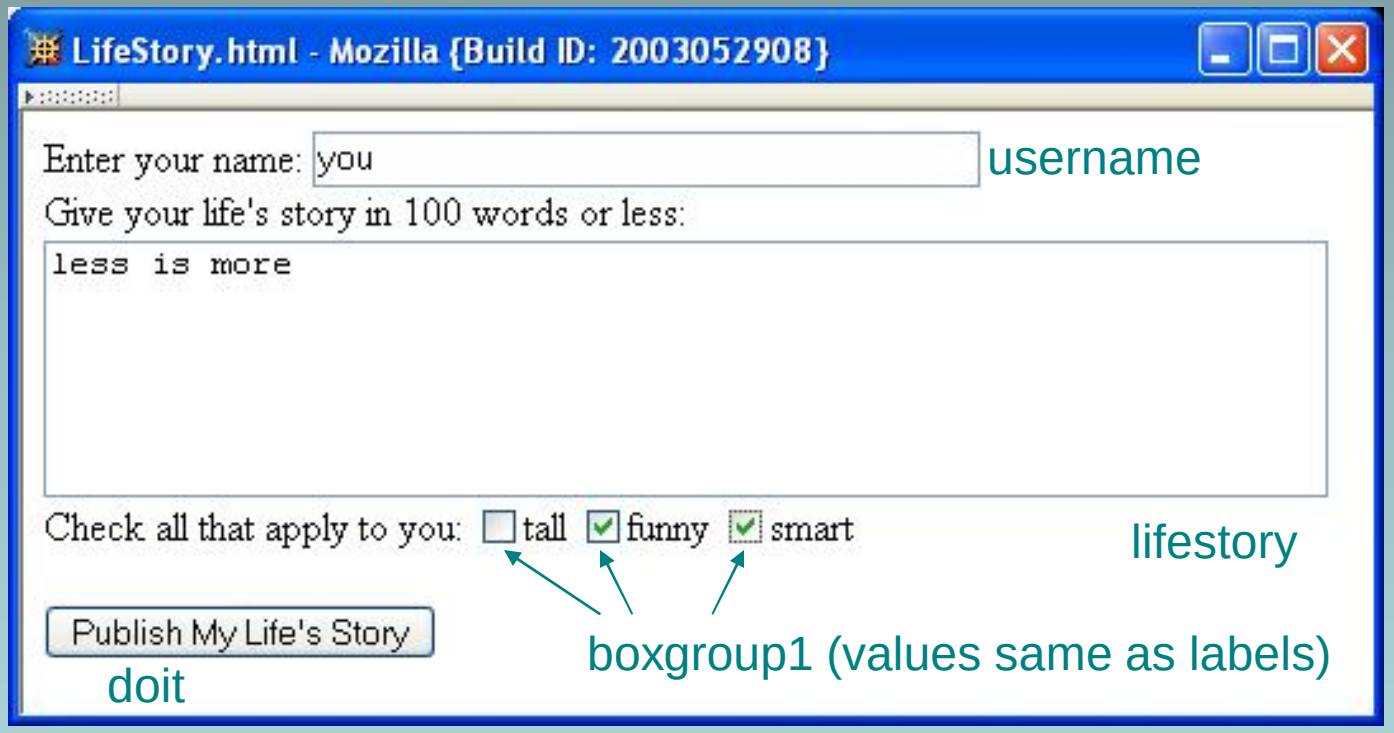
Value for checkbox specified by `value` attribute

Parameter Data

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

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

Parameter Data



Parameter Data

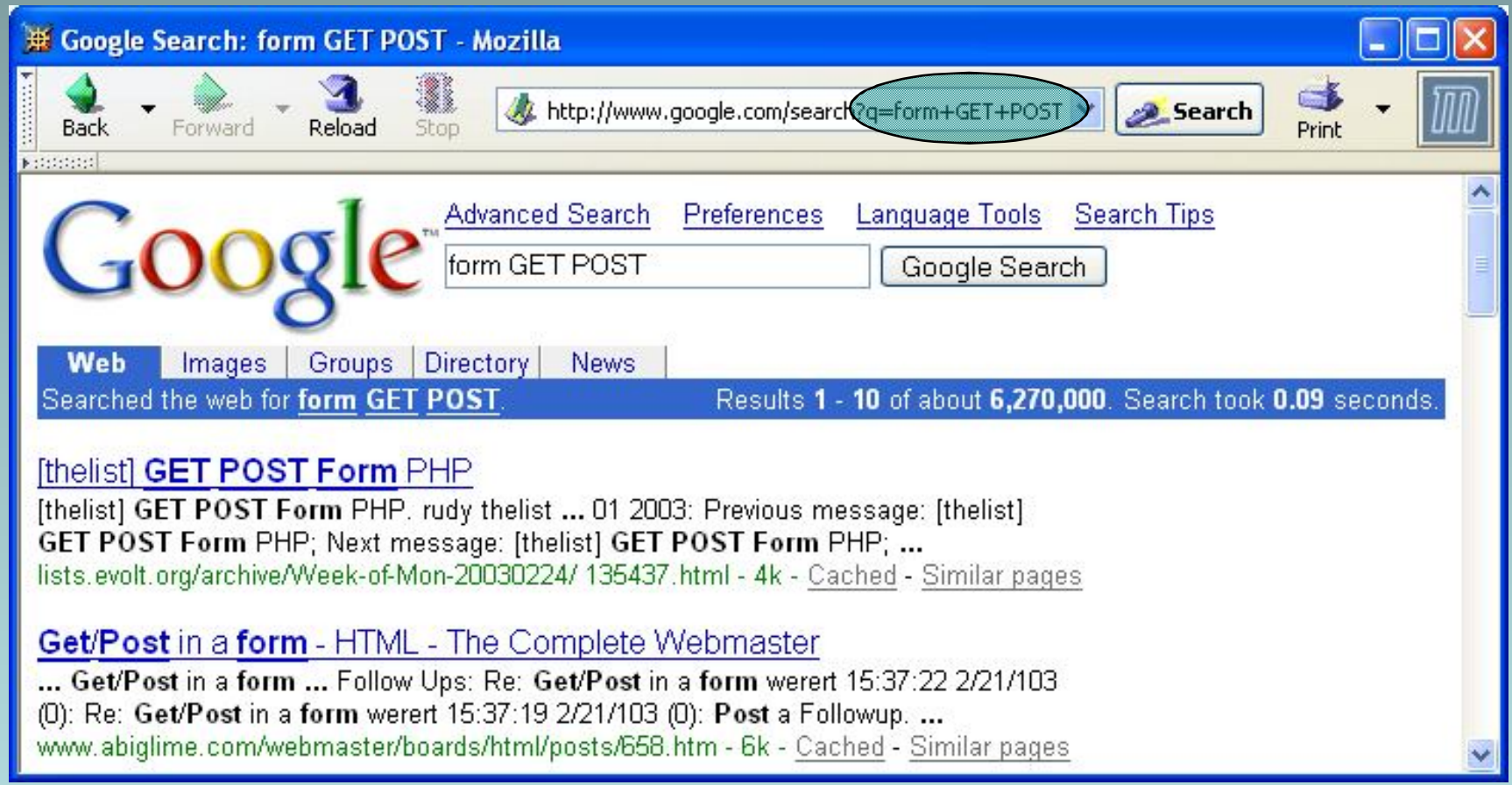
- Query string produced by browser (all one line):

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

Parameter Data

- GET vs. POST for the `method` attribute of forms:
  - GET:
    - Query string is part of URL
    - Length of query string may be limited
    - Recommended when parameter data is not stored or updated on the server, 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



Parameter Data

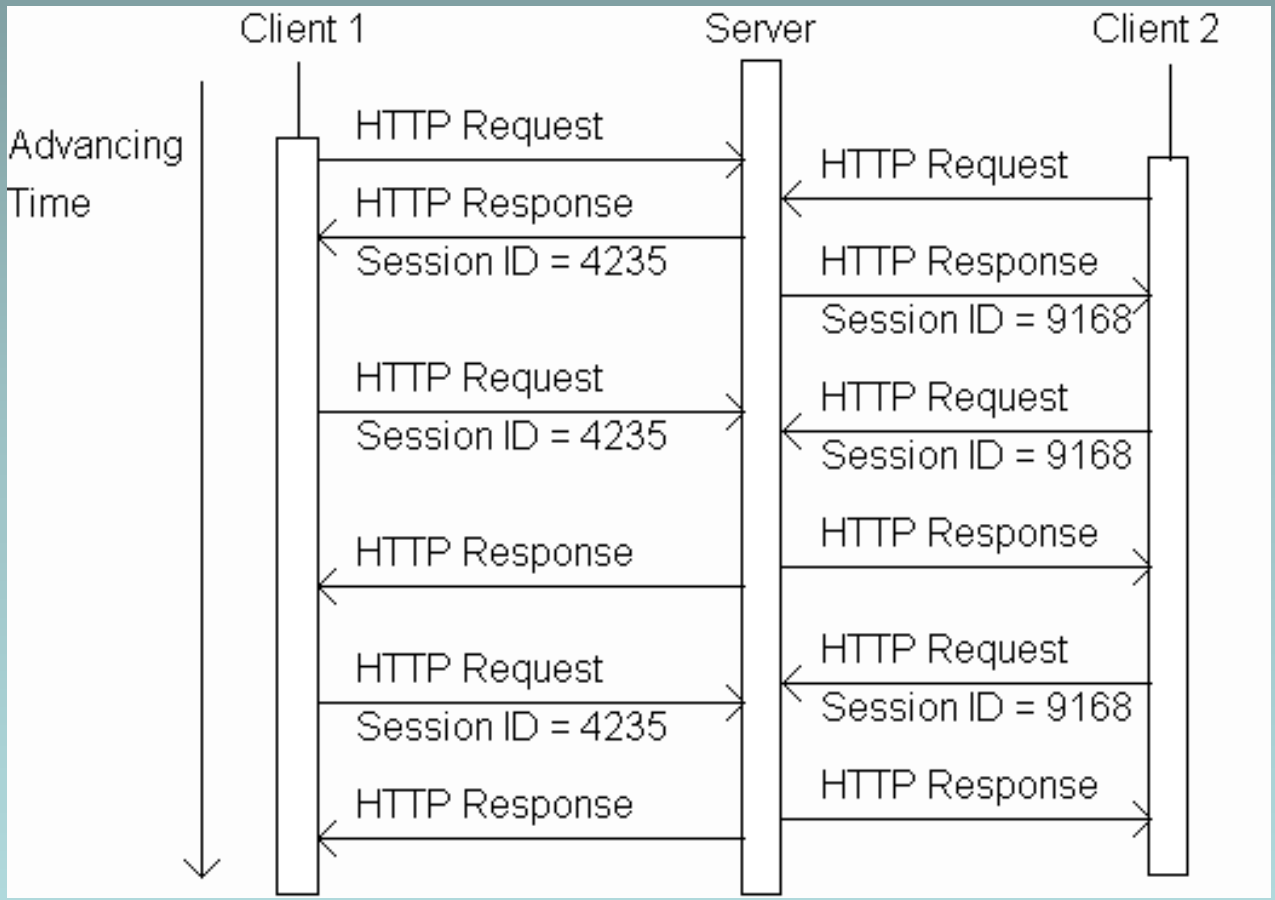
- GET vs. POST method for forms:
  - POST:
    - Query string is sent as body of HTTP request
    - Length of query string is unlimited
    - Recommended if parameter data is intended to cause the server to update stored data
    - Most browsers will warn you if they are about to resubmit POST data to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

Parameter Data

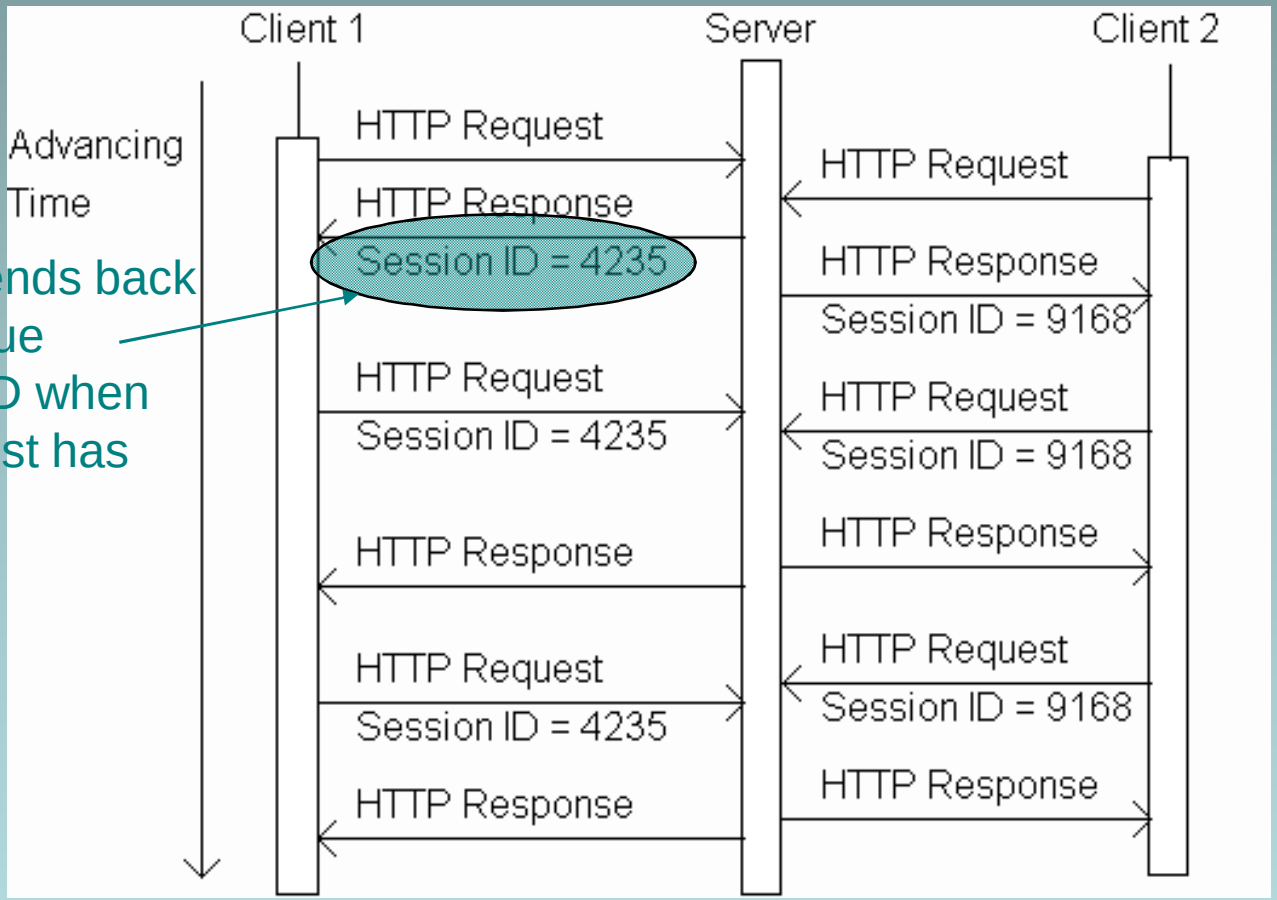
Sessions

- 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

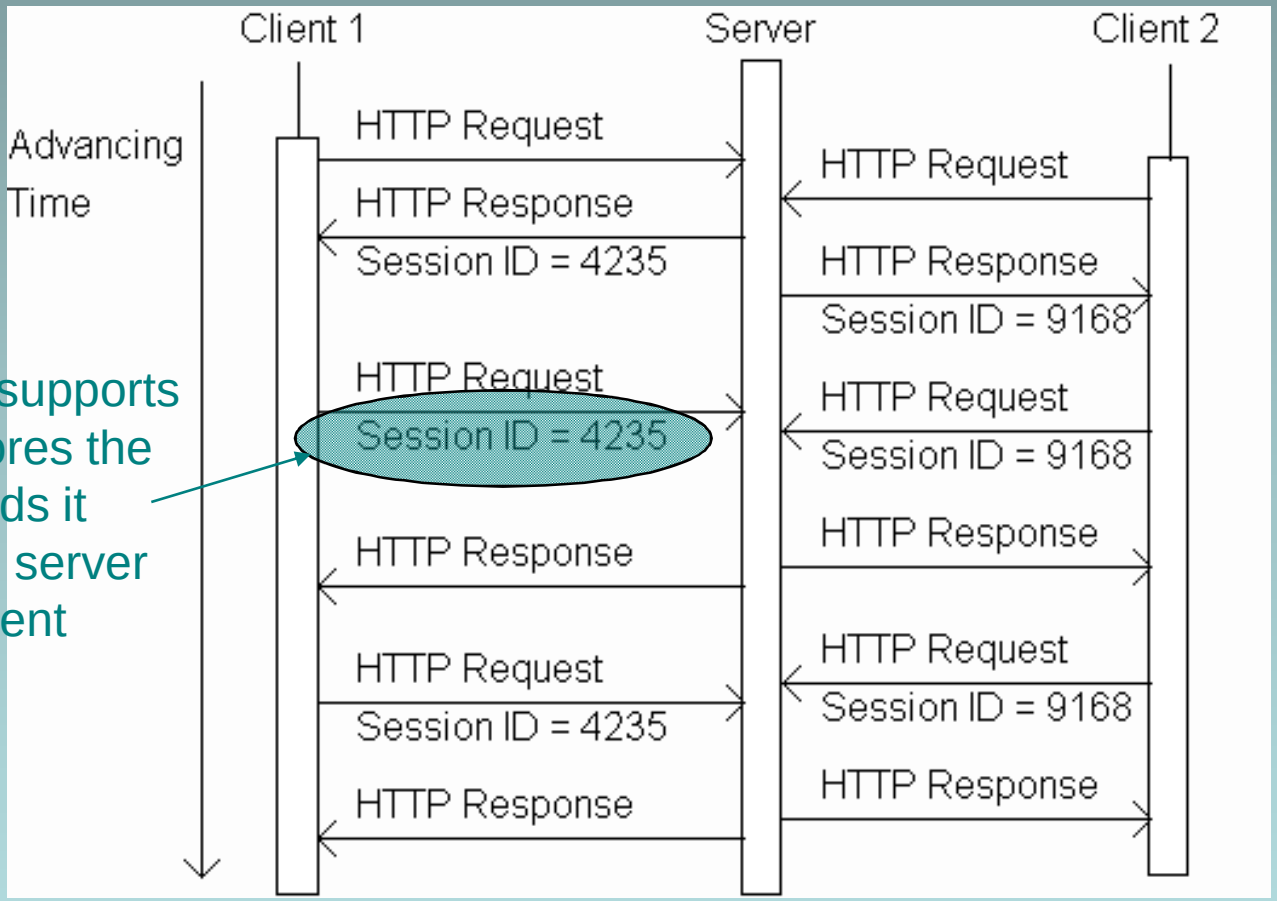

Client 1
Server
Client 2
Advancing
Time
HTTP Request
HTTP Response
Session ID = 4235
HTTP Request
HTTP Response
Session ID = 9168
HTTP Request
Session ID = 4235
HTTP Request
Session ID = 9168
HTTP Response
HTTP Response
HTTP Request
Session ID = 4235
HTTP Request
Session ID = 9168
HTTP Response
HTTP Response

Sessions



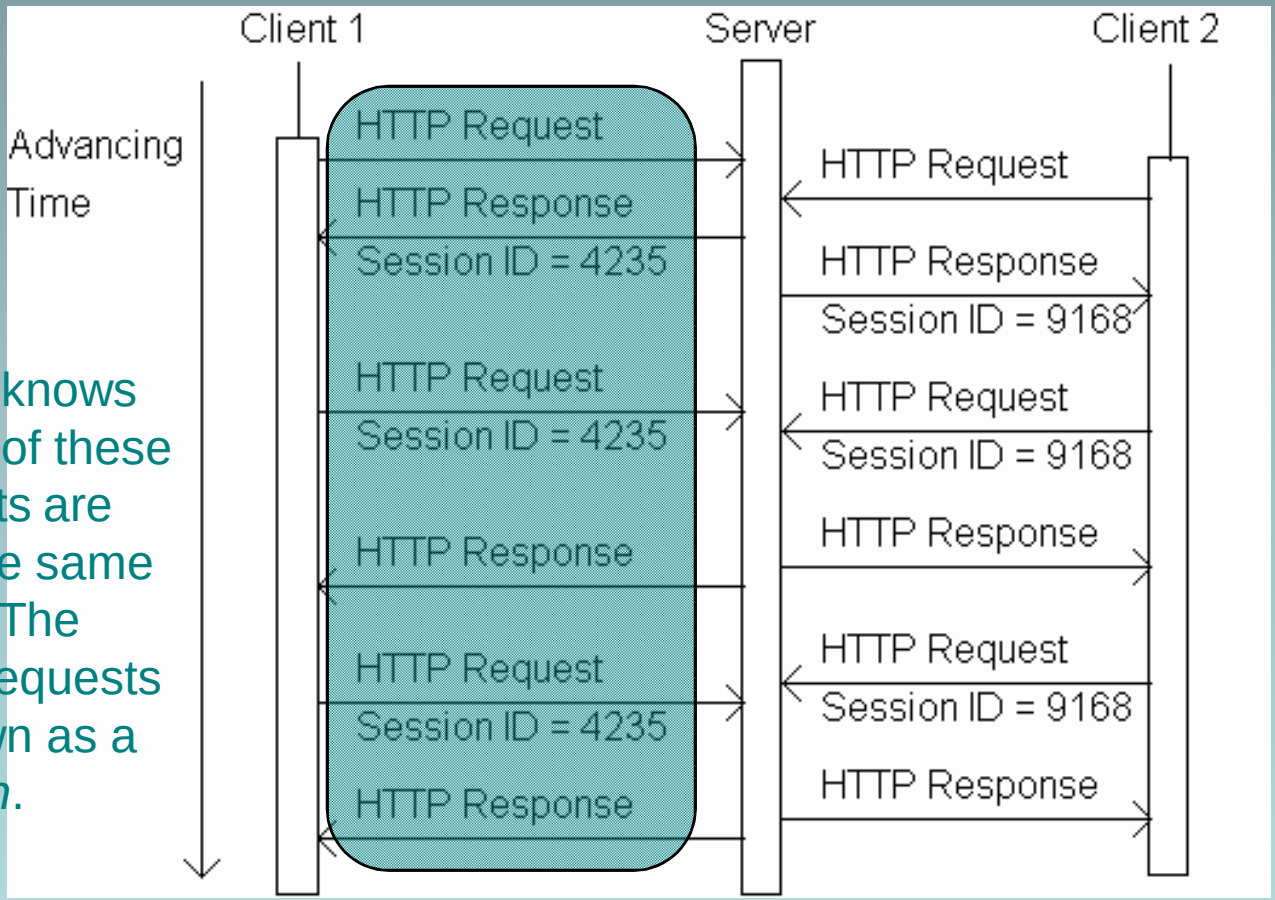
Server sends back new unique session ID when the request has none

Sessions



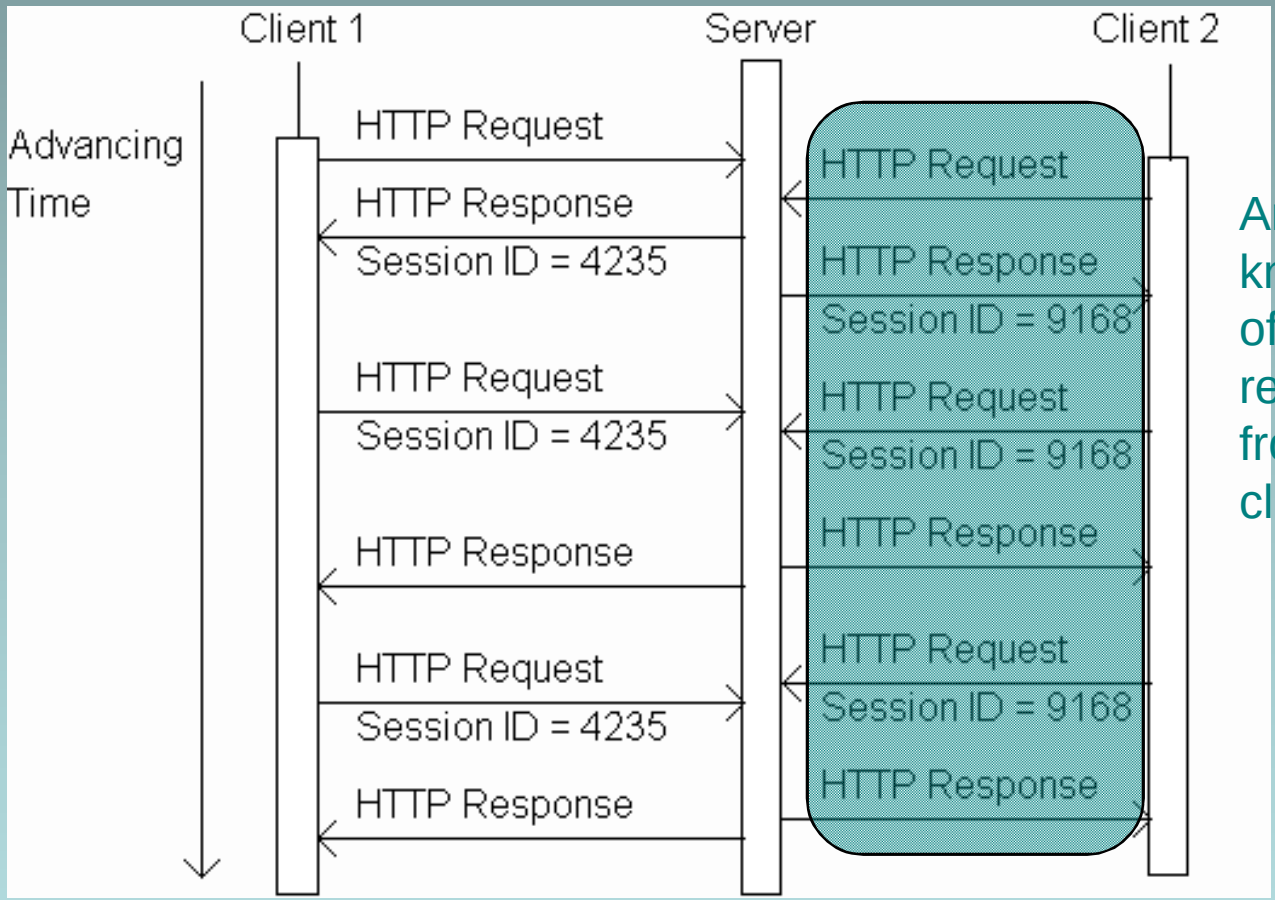
Client that supports session stores the ID and sends it back to the server in subsequent requests

Sessions



Server knows that all of these requests are from the same client. The set of requests is known as a *session*.

Sessions



And the server knows that all of these requests are from a different client.

Sessions

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

HttpSession object is created by server when a servlet calls getSession() method on its HttpServletRequest parameter.

getSession() method returns HttpSession object associated with this HTTP request.

- Creates new `HttpSession` object if no valid session ID in HTTP request
- Otherwise, returns previously created HttpSession object containing the session ID

Sessions

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

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

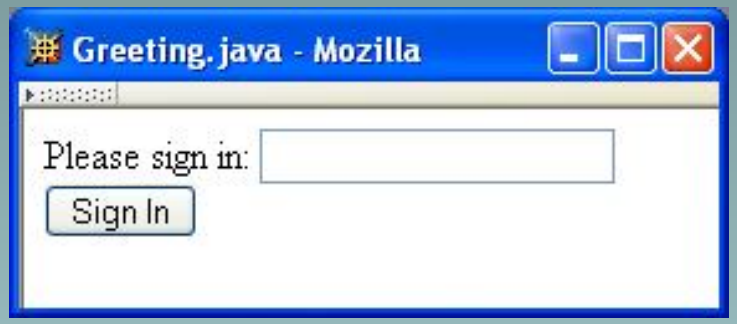
Sessions

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

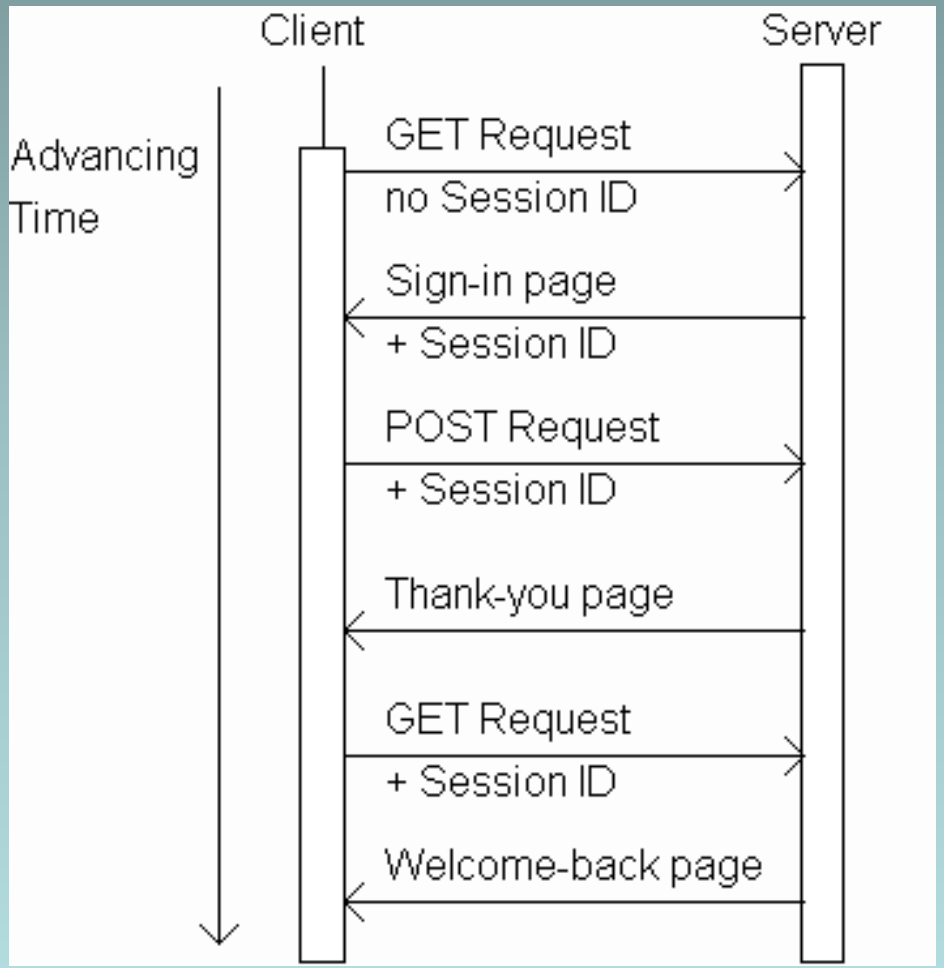
Incremented once per session

Sessions

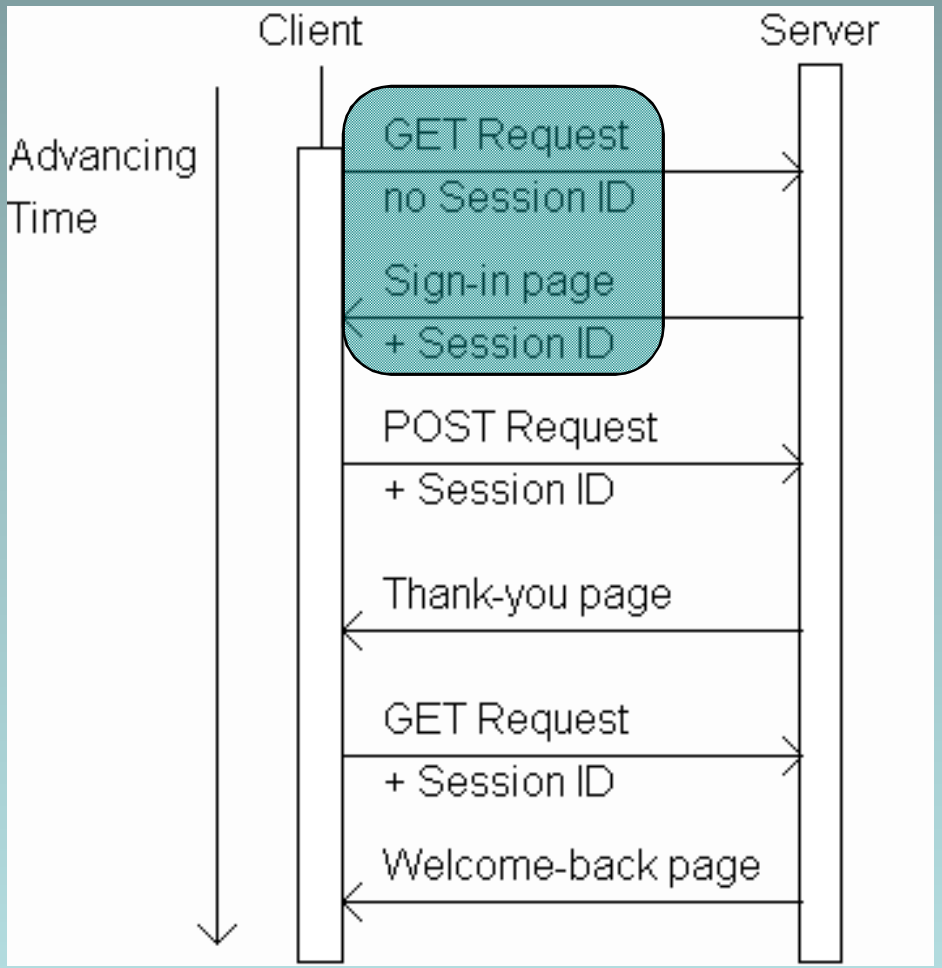
Three web pages produced by a single servlet



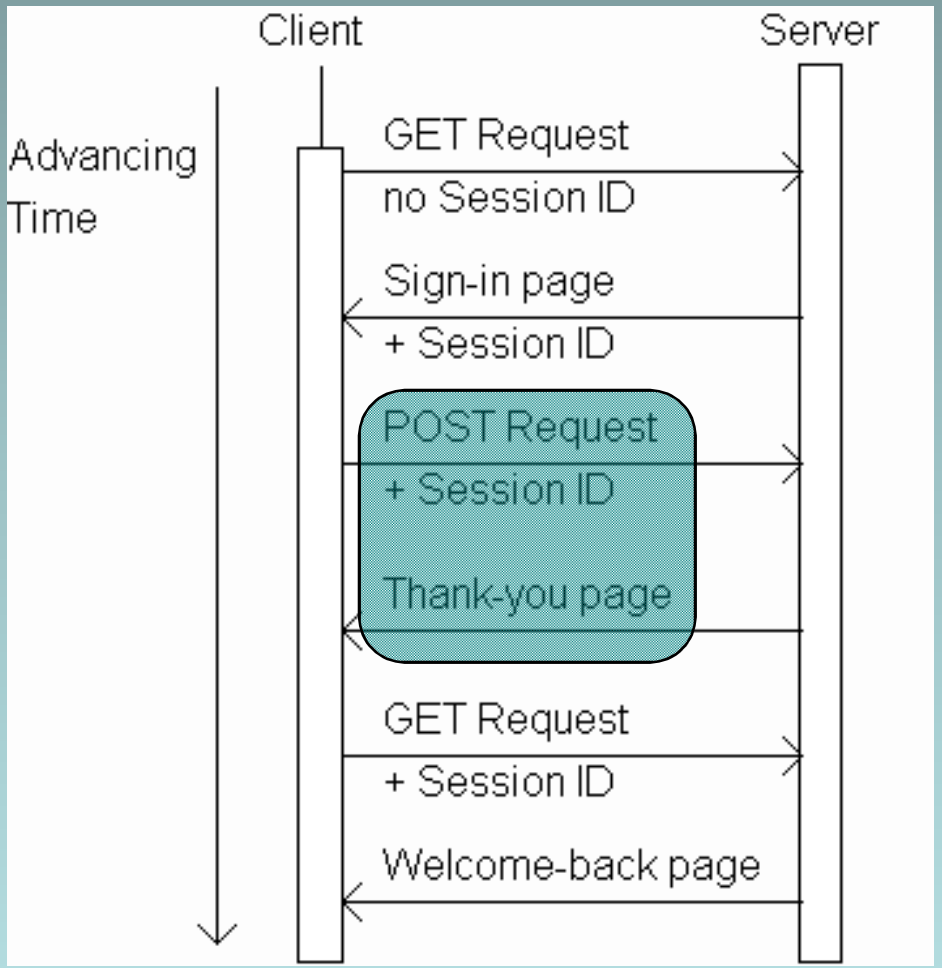
Sessions


Client
Server
Advancing
Time
GET Request
no Session ID
Sign-in page
+ Session ID
POST Request
+ Session ID
Thank-you page
GET Request
+ Session ID
Welcome-back page

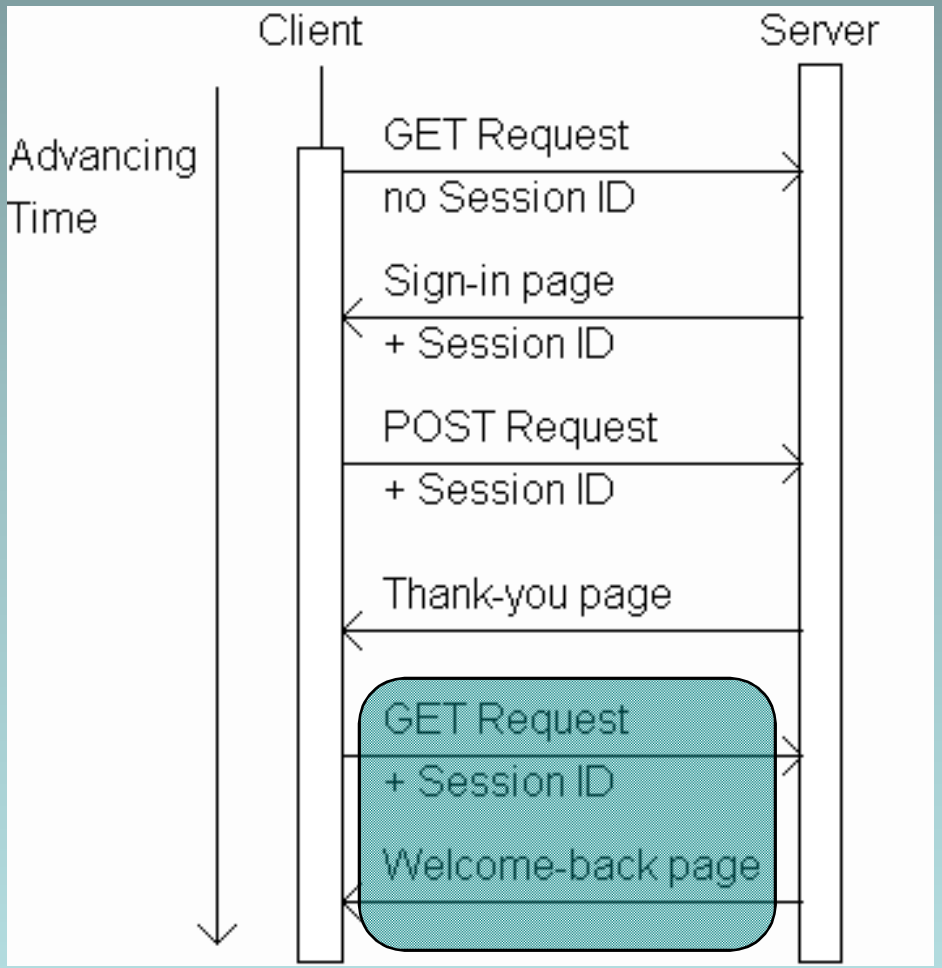
Sessions


Client
Server
Advancing
Time
GET Request
no Session ID
Sign-in page
+ Session ID
POST Request
+ Session ID
Thank-you page
GET Request
+ Session ID
Welcome-back page

Sessions


Client
Server
Advancing
Time
GET Request
no Session ID
Sign-in page
+ Session ID
POST Request
+ Session ID
Thank-you page
GET Request
+ Session ID
Welcome-back page

Sessions


Client
Server
Advancing
Time
GET Request
no Session ID
Sign-in page
+ Session ID
POST Request
+ Session ID
Thank-you page
GET Request
+ Session ID
Welcome-back page

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

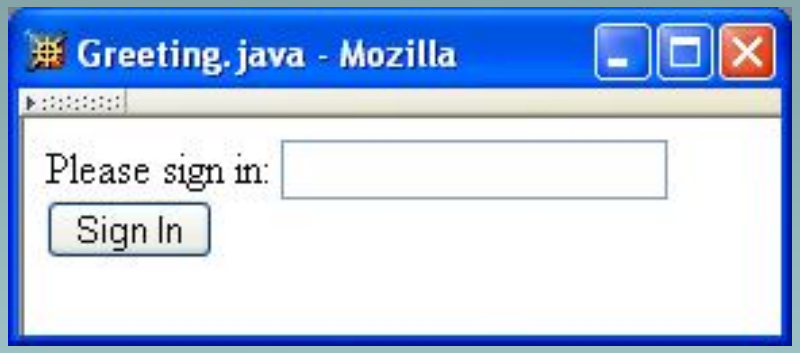
Sessions

```
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, generate weclome-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)

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

Retrieve `signIn` parameter value

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

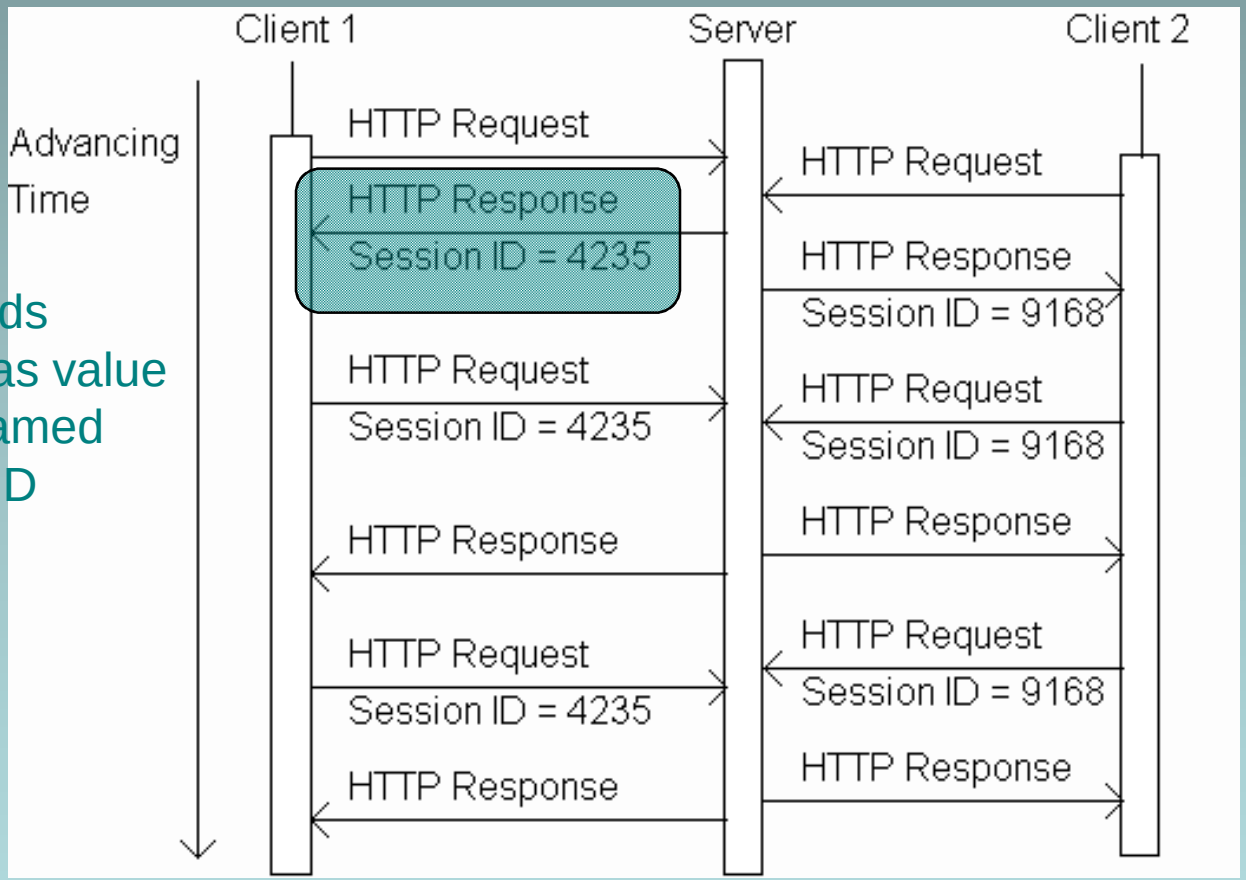
- By default, each session expires if a server-determined length of time elapses between a session's HTTP requests
  - Server destroys the corresponding session object
- Servlet code can:
  - Terminate a session by calling `invalidate()` method on session object
  - Set the expiration time-out duration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

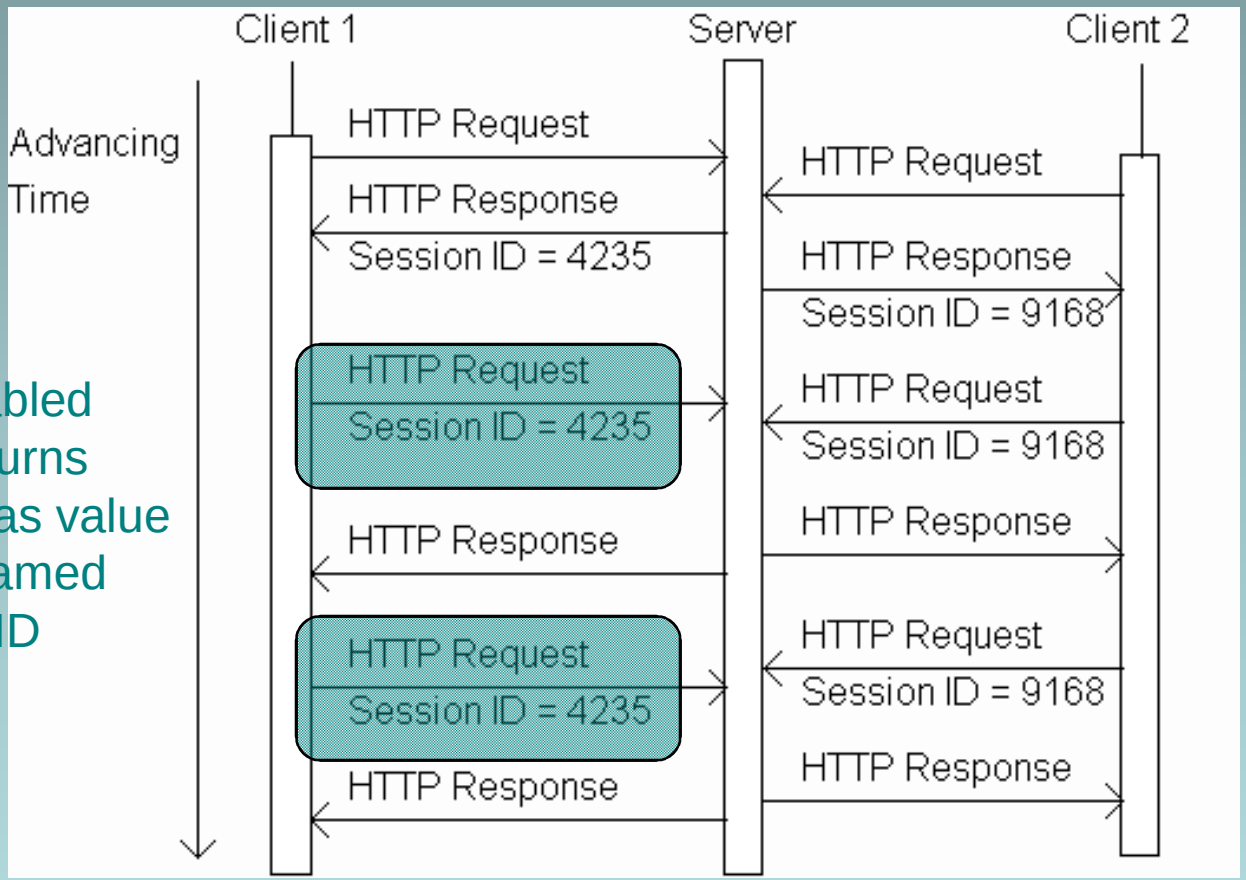
Cookies

Tomcat sends session ID as value of cookie named JSESSIONID



Cookies

Cookie-enabled browser returns session ID as value of cookie named JSESSIONID



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
`Cookie(String name, String value)`	Constructor to create a cookie with given name and value
`String getName()`	Return name of this cookie
`String getValue()`	Return value of this cookie
`void setMaxAge(int seconds)`	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 are expired by client (server can request expiration date)

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)

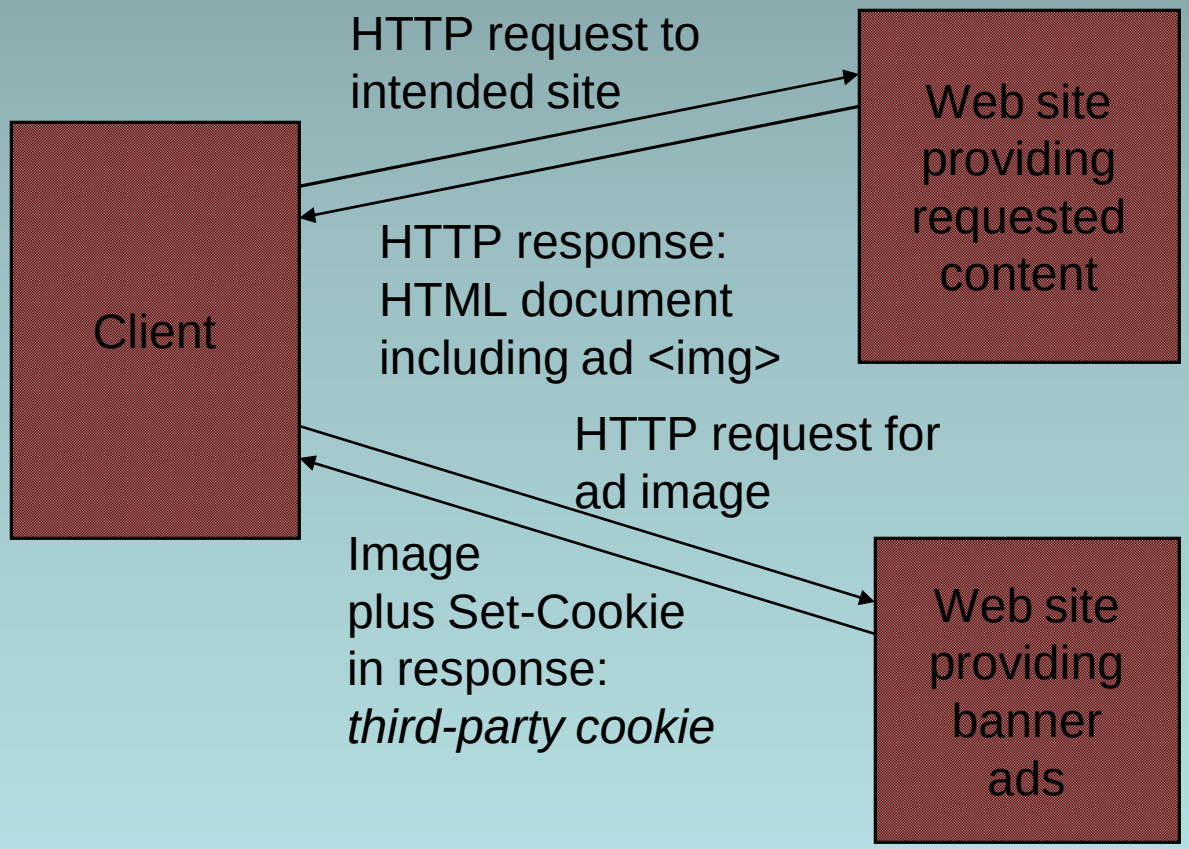
```
        // Increment the count and add request to client to store it
        // for one year.
        count++;
        Cookie cookie = new Cookie("COUNT",
                                   new Integer(count).toString());
        cookie.setMaxAge(oneYear);
        response.addCookie(cookie);

        // 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

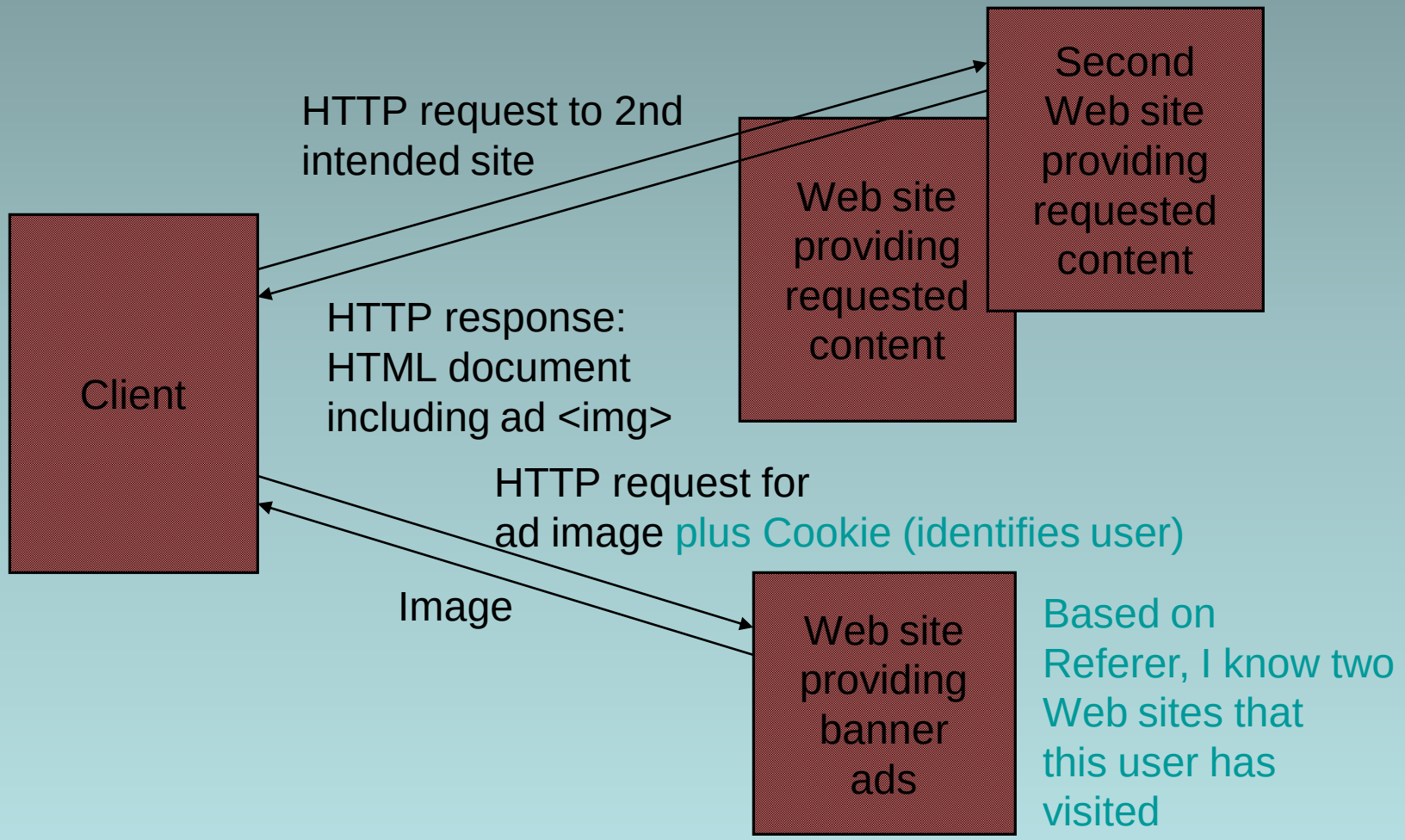
```
        // Increment the count and add request to client to store it
        // for one year.
        count++;
        Cookie cookie = new Cookie("COUNT",
                                   new Integer(count).toString());
        cookie.setMaxAge(oneYear);
        response.addCookie(cookie);

        // 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

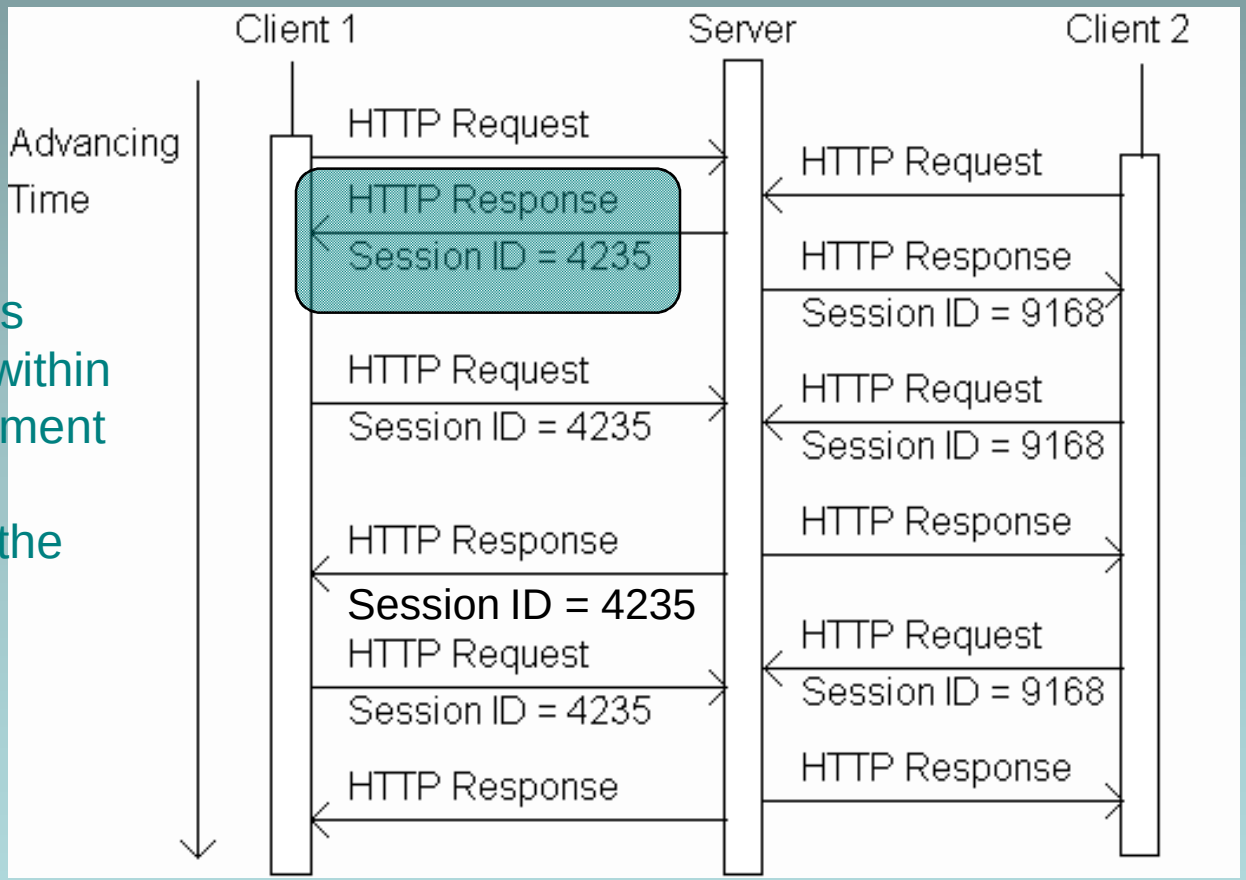


Cookies
Privacy issues

- 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

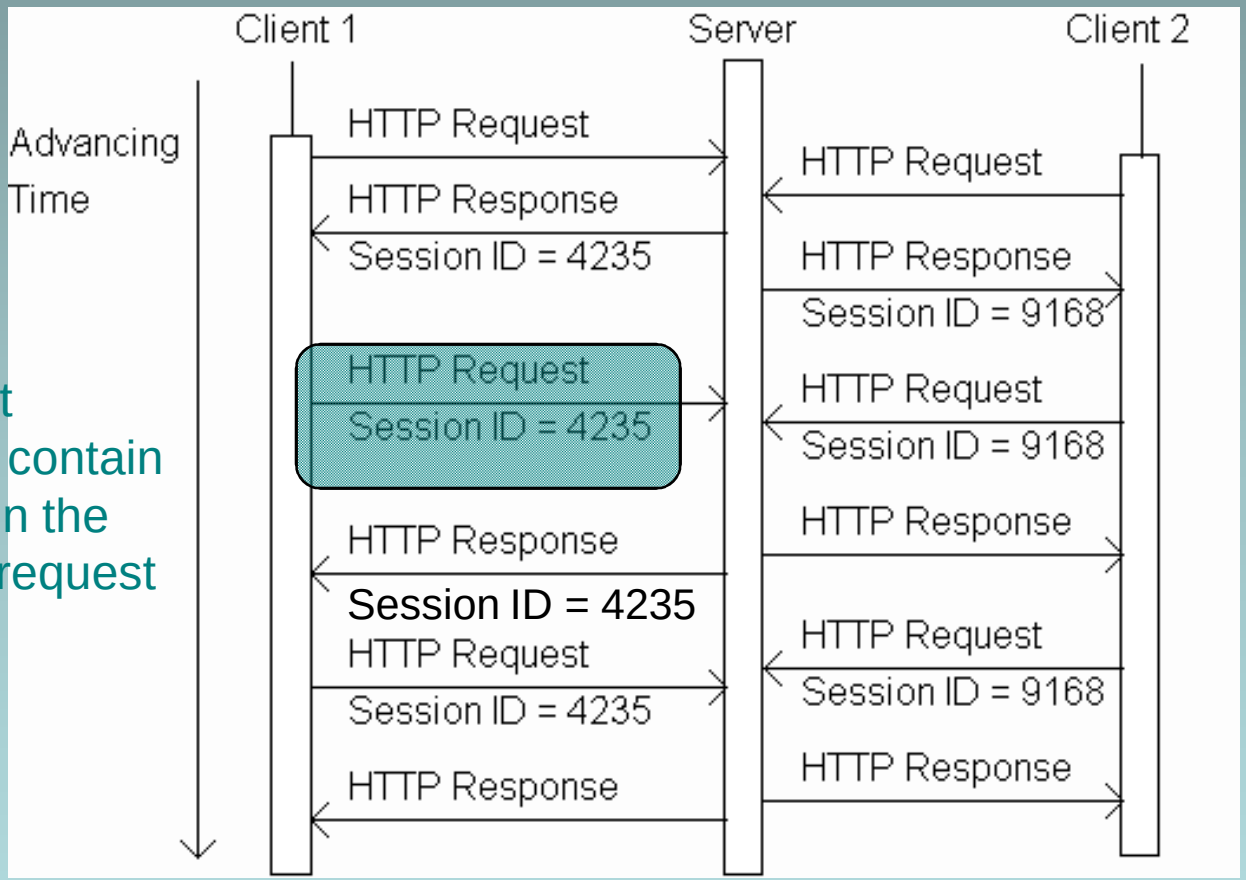
URL Rewriting

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

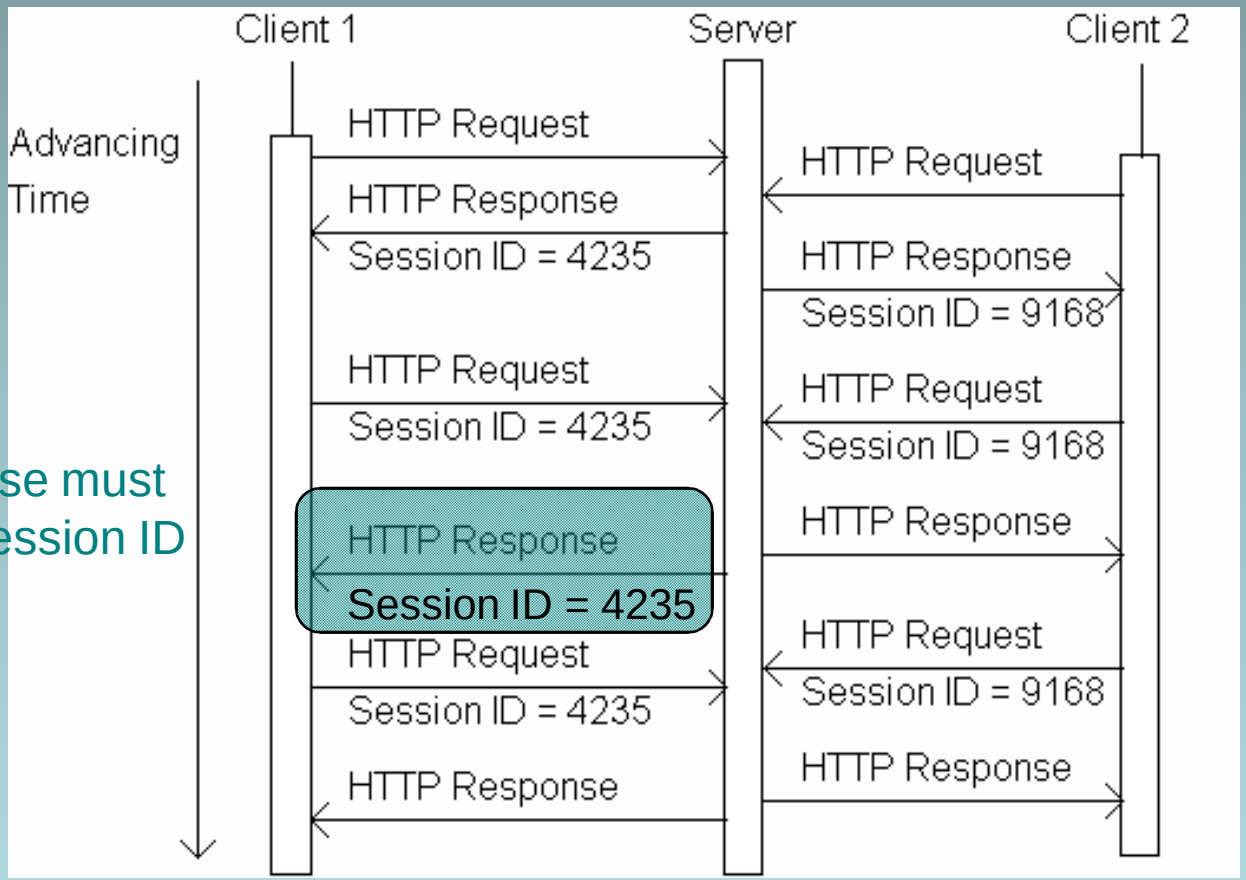


URL Rewriting

Subsequent request will contain session ID in the URL of the request



URL Rewriting



Next response must again add session ID to all URL's

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");
```

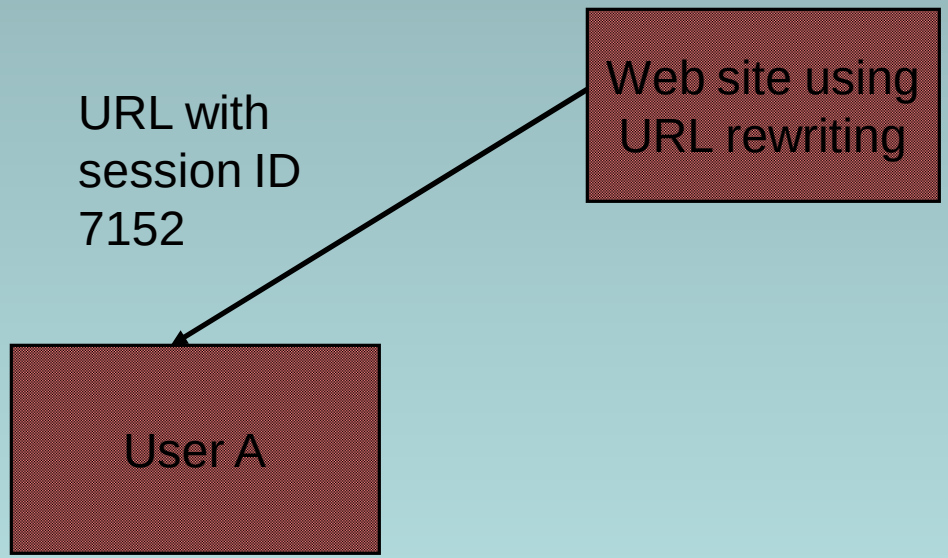
Relative URL of servlet

Servlet using URL rewriting

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

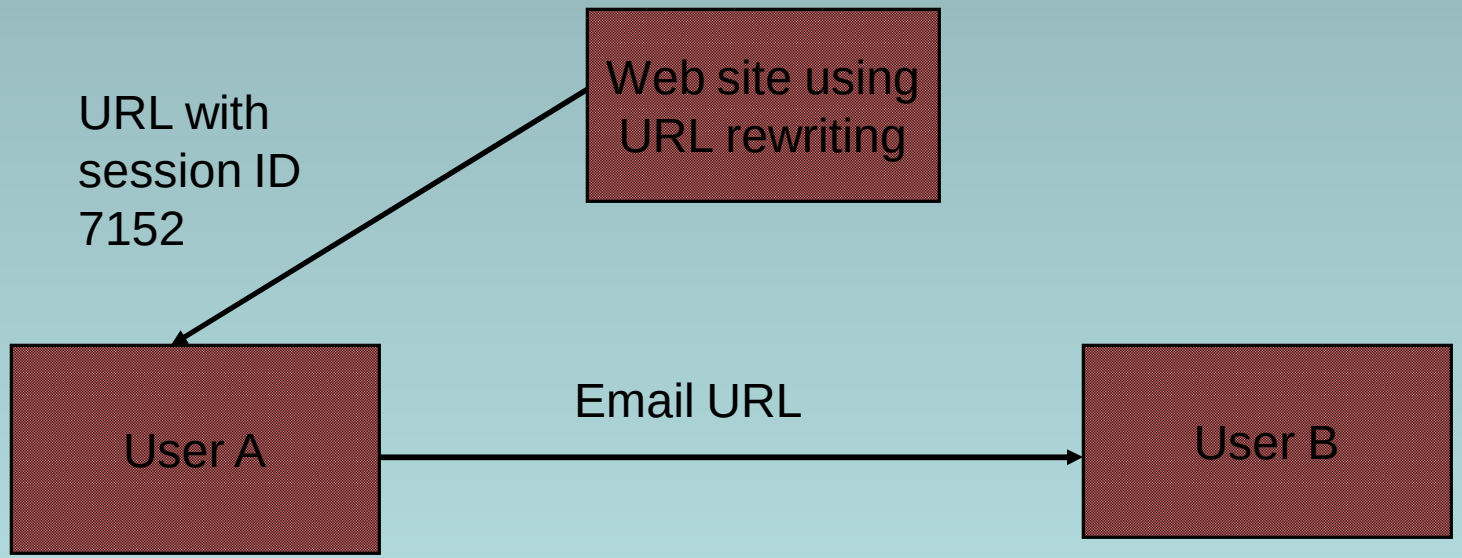
URL Rewriting

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



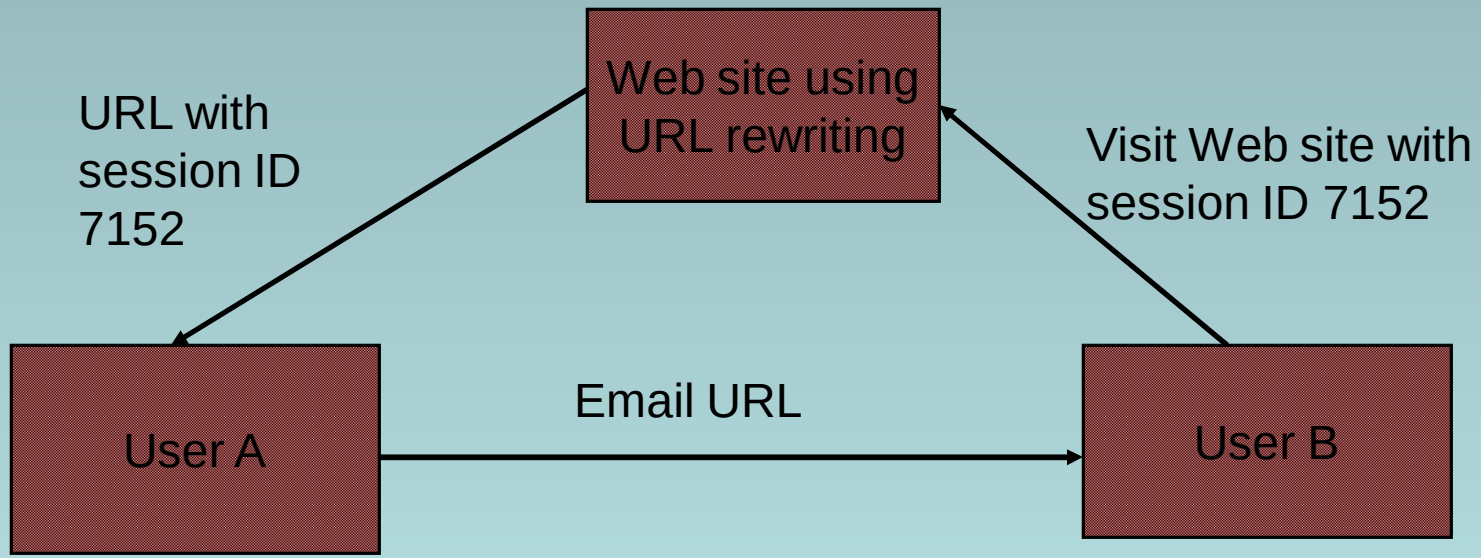
URL Rewriting

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



URL Rewriting

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

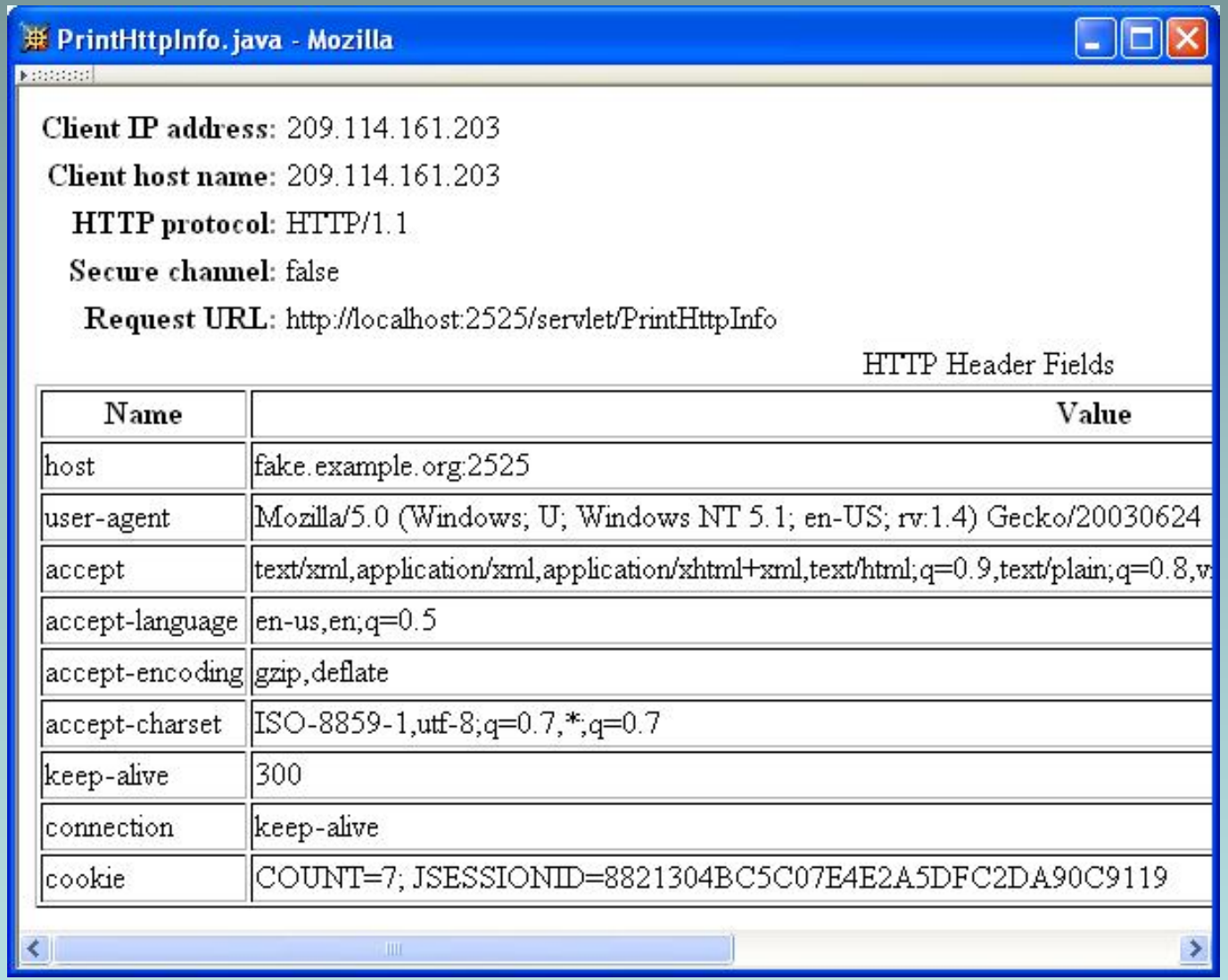


More Servlet Methods

TABLE 6.4: Additional `HttpServletRequest` methods.

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

More Servlet Methods



More Servlet Methods

TABLE 6.5: Additional `HttpServletResponse` methods.

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

`void setStatus(int statusCode)`	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.
`void sendError(int statusCode, String msg)`	Set the status code in the HTTP response to the given error `statusCode` (status code beginning with 4 or 5), and in the body of the response send a server-generated HTML error page containing the given `msg`.
`void sendRedirect(String url)`	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 `url`. Client will behave as if it had sent request to the specified `url`.
`void encodeRedirectURL(String url)`	Perform URL rewriting (for session management) on `url` that will be used for redirection.

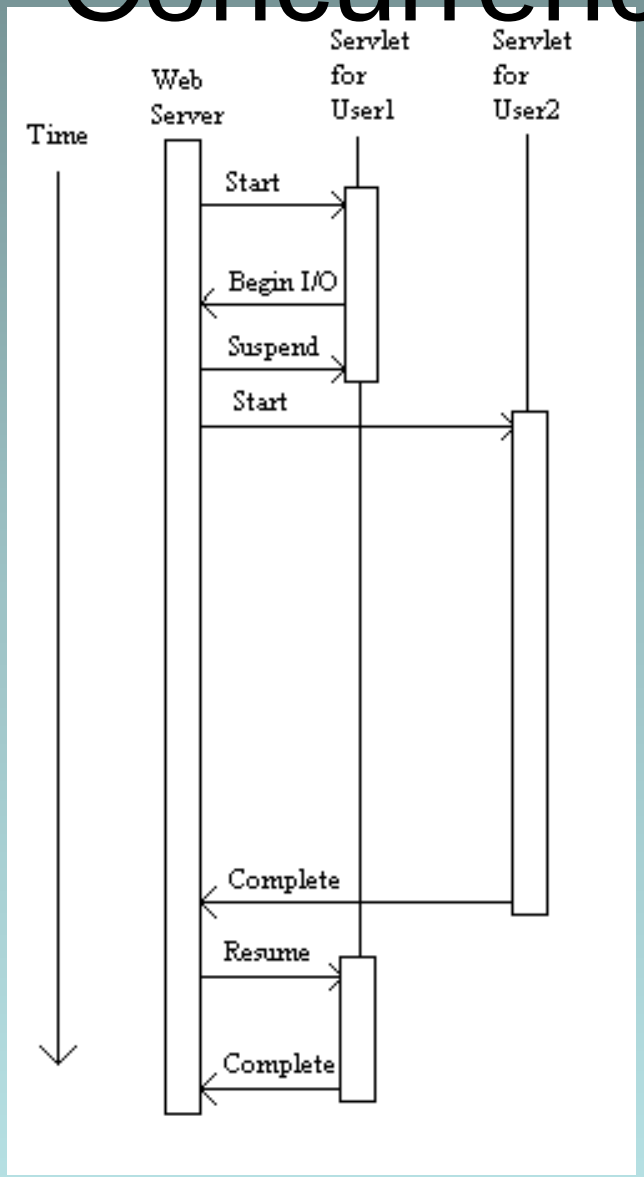
More Servlet Methods

- 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

User1 Thread	User2 Thread
<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

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

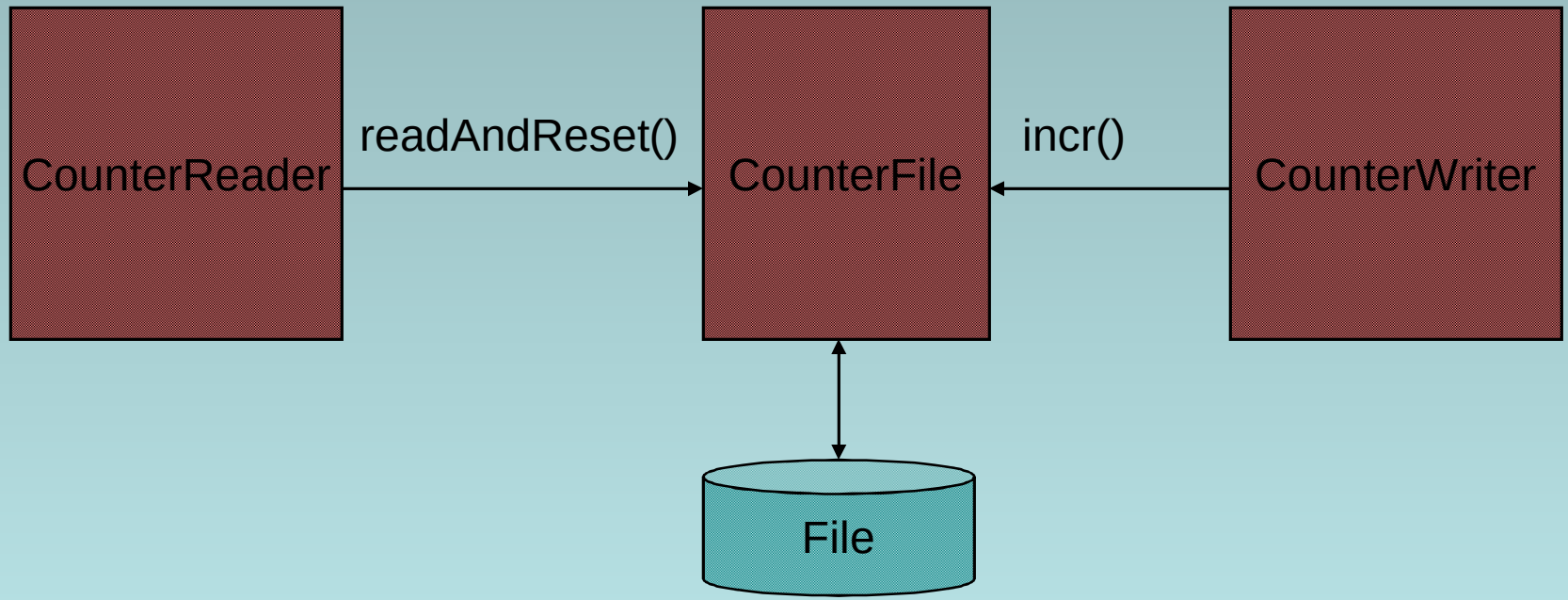
Concurrency

- Web application with multiple servlet classes and shared resource:

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

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

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

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

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)