Assignment 1

Dynamic Web site, without using Web Languages

Assignment 1 – part 1

Modify the simple web server so that all the URLs that start with the token "process " (e.g. http://localhost:8000/process) launch an external process.

http://localhost:8000/process/reverse?par1=*string*
should activate an (**external**) process that takes the parameter string.
and returns the reversed string (e.g. ROMA -> AMOR).

Suggestions:
To see how to start an external process from Java, take a look at one of these:
- http://www.rgagnon.com/javadetails/java-0014.html
- https://www.mkyong.com/java/how-to-execute-shell-command-from-java/
- https://www.baeldung.com/run-shell-command-in-java

To split a string into tokens, and to reverse a string see:
- https://www.javatpoint.com/how-to-reverse-string-in-java
- https://www.javatpoint.com/string-tokenizer-in-java

Assignment 1 – part 2

Install and run an Apache WebServer

Create a script that launches the java program defined in part 1.

Suggestion:

- write a .sh or a .bat (depending on what platform you're on) that:
  - retrieves the parameters
  - launches the java program "prog" as
    java prog params

Delivery

Deliver the (zipped) IntelliJ project of Part 1

Deliver a report using the following structure:

Title Page containing date, title, your name

For each of the two parts:

Section x:

x.1 Introduction (problem statement, description of the domain)

x.2 Short description of what you did

x.3 screen shots of your app running, documenting the various steps

x.4 Comments and notes (optional: any problems encountered during project development, any other comment)

References (if any)

Delivery has to be done by Oct.3, 23:59 on
https://didatticaonline.unitn.it/dol/mod/publication/view.php?id=997920

Q

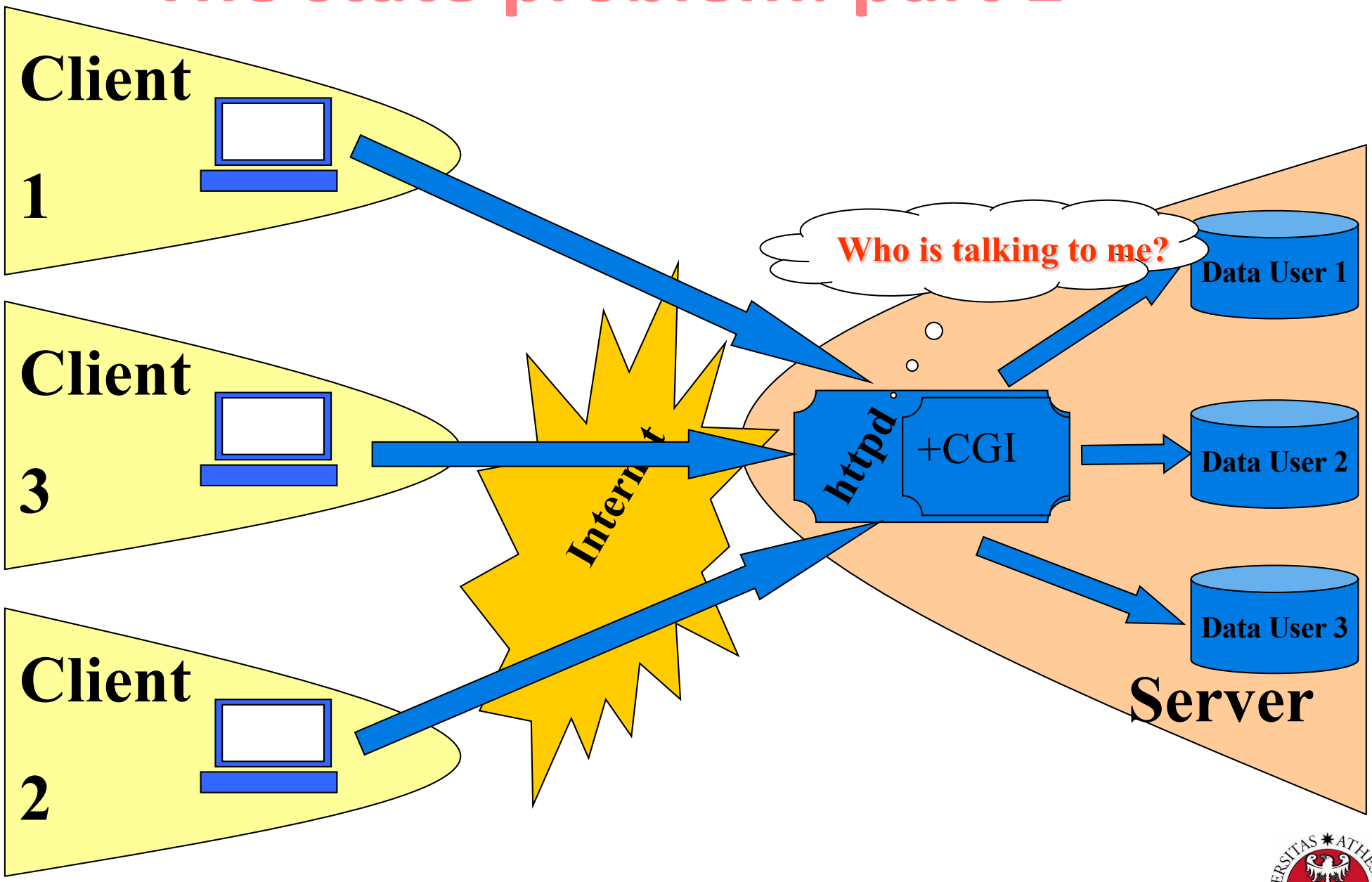
What is the HTTP state problem?

HTTP is stateless

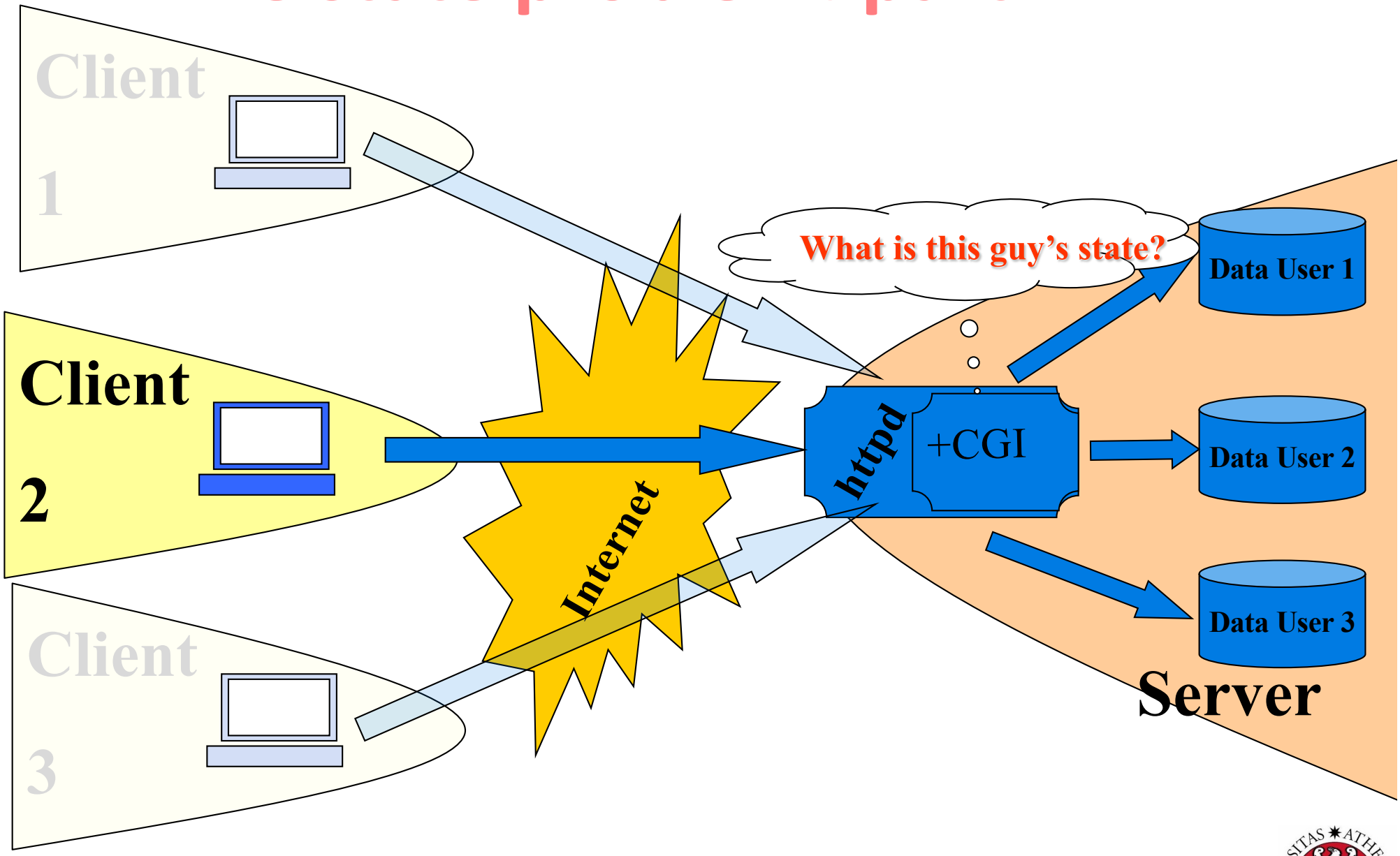
How can we keep track of who is who, and which state of the process s/he is at?

There is no way of doing it server-side…

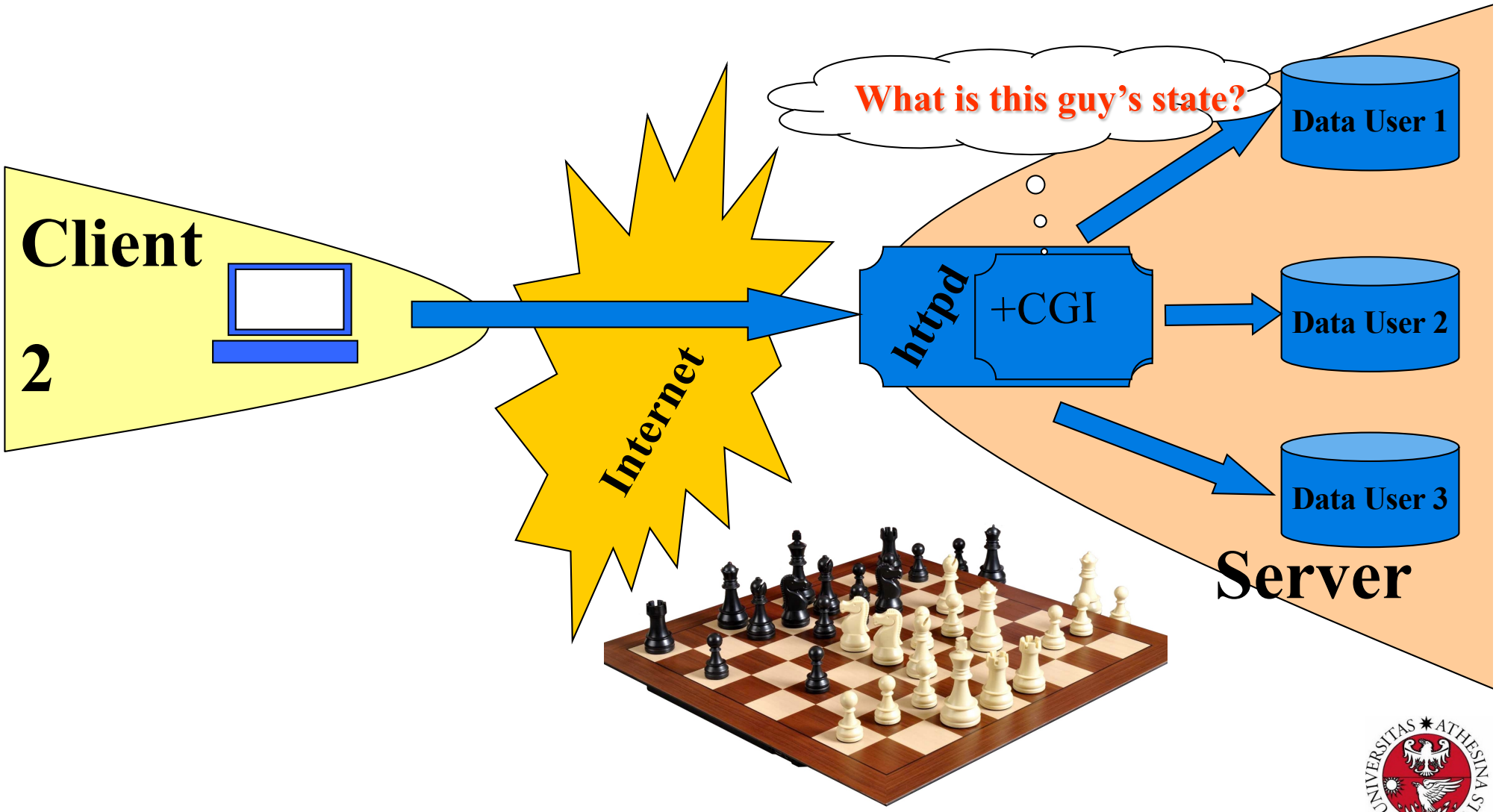
The state problem: part 1



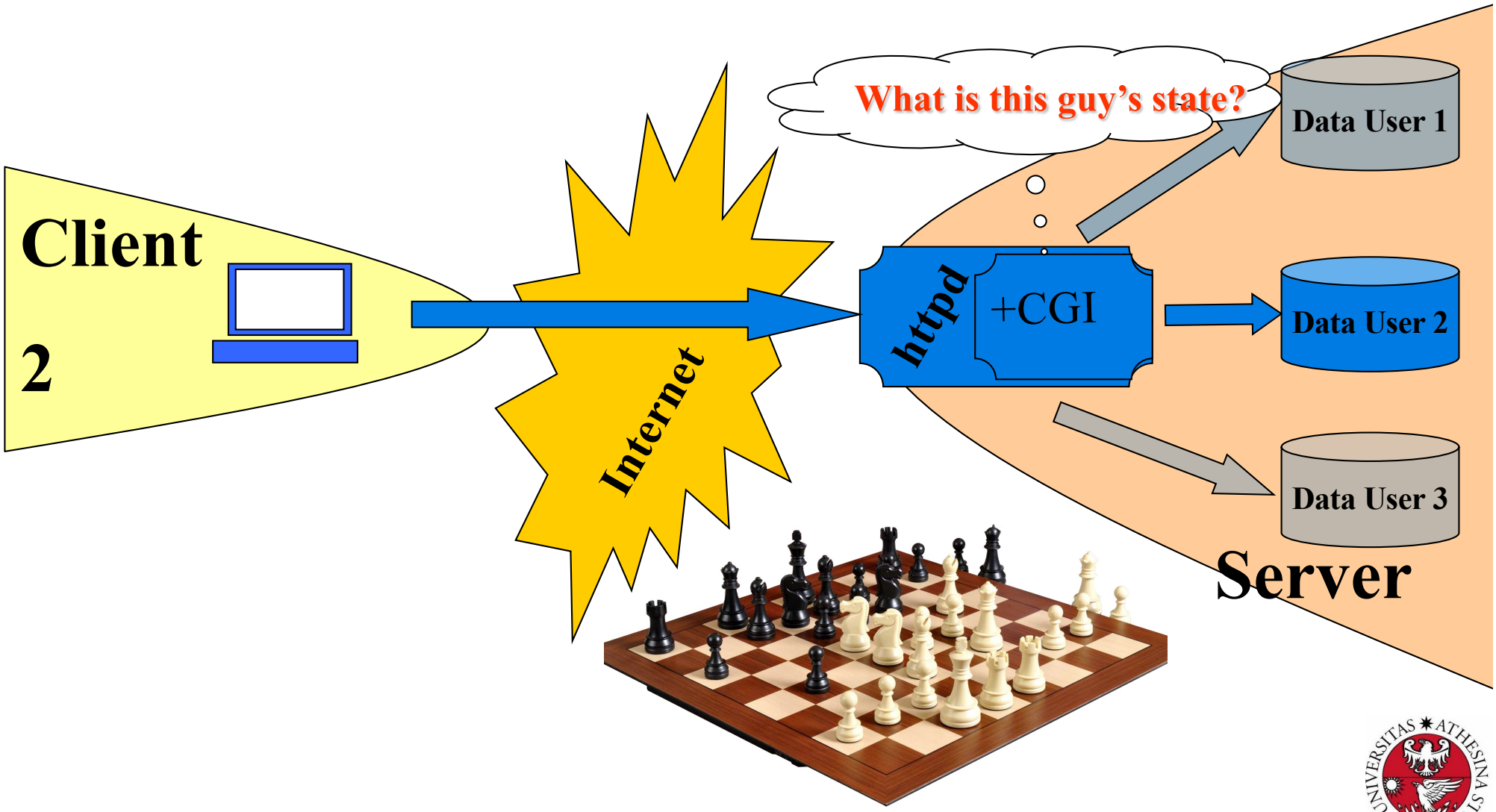
The state problem: part 2



The state problem: part 2 - example



The state problem: part 2 - example

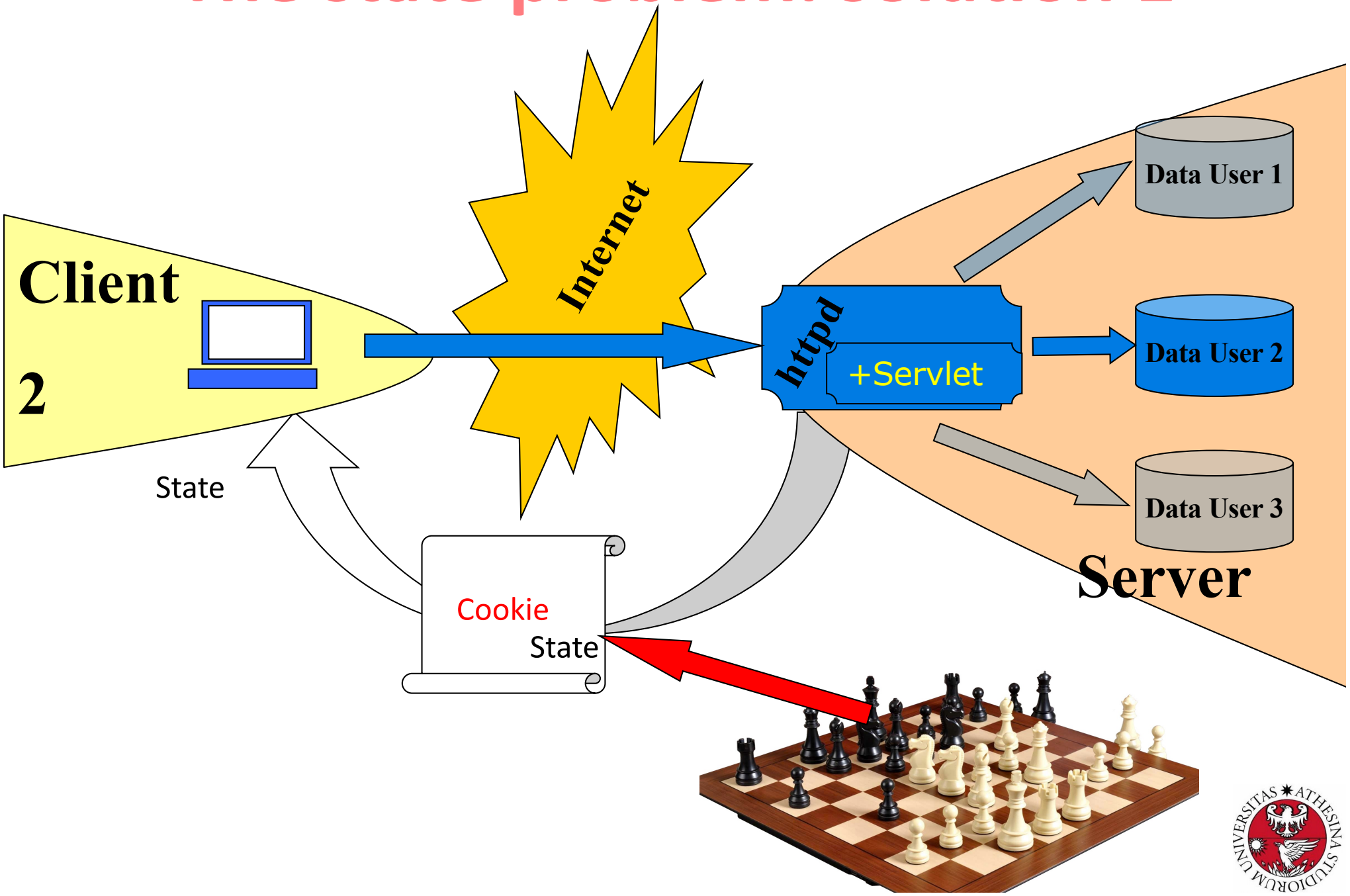


Q

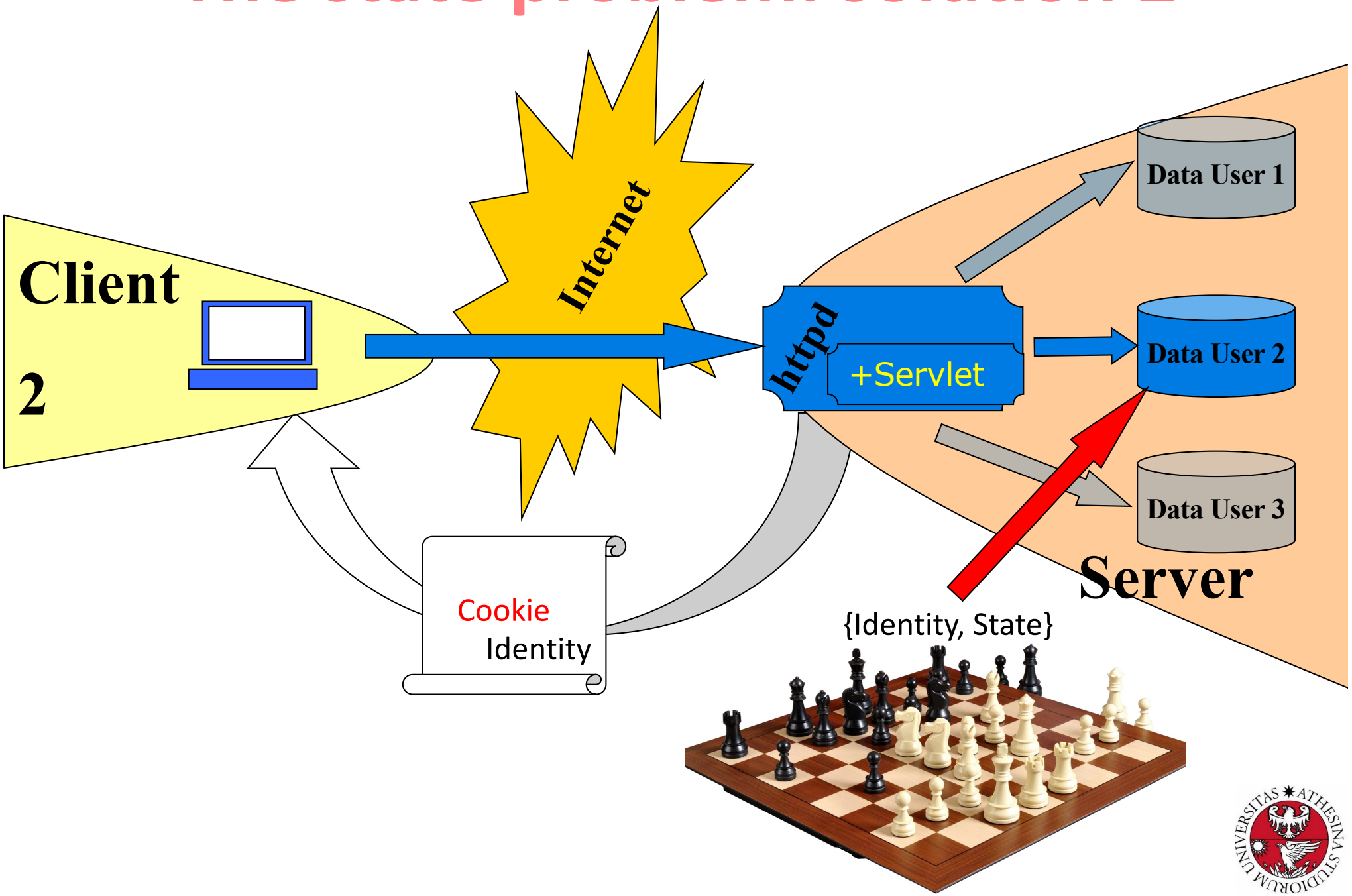
What are cookies ?

How do they solve the state problem?

The state problem: solution 1

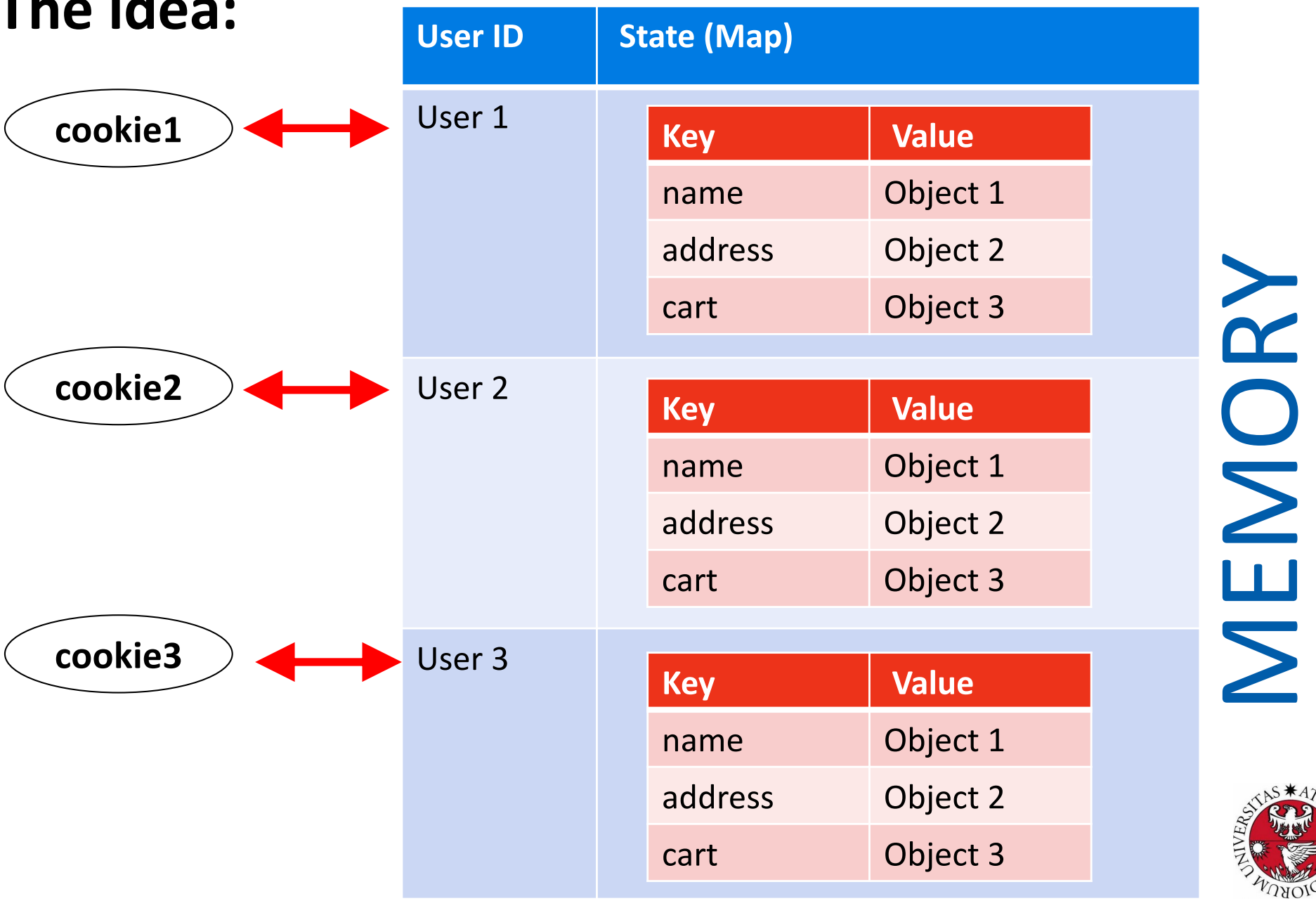


The state problem: solution 2



Keeping the state by using cookies

The idea:



Cookies:

A Cookie is a small amount of information sent by a servlet to a Web browser, saved by the browser, and later sent back to the server.

A cookie's value can uniquely identify a client, so cookies are commonly used for session management.

A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

Cookies

- The servlet sends cookies to the browser by using the HttpServletResponse.addCookie(javax.servlet.http.Cookie) method, which adds fields to HTTP response headers to send cookies to the browser, one at a time. The browser is expected to support 20 cookies for each Web server, 300 cookies total, and may limit cookie size to 4 KB each.

- The browser returns cookies to the servlet by adding fields to HTTP request headers. Cookies can be retrieved from a request by using the HttpServletRequest.getCookies() method.

Q

Which operations can we perform on cookies ?

main operations on cookies

String getName() / void setName(String s)

String getValue() / void setValue(String s)

boolean getSecure / setSecure(boolean b) (encript the cookie)

int getMaxAge() / void setMaxAge(int i) (Positive: seconds to live. Zero: delete cookie. Negative: only as long as browser quits.

Control cookie scope: Normally, cookies are returned only to the exact hostname that sent them.

- String getDomain() / setDomain(String s):
- instruct the browser to return them to other hosts within the same domain.
- String getPath() / void setPath(String s)
- restrict the path of URL which can obtain the cookie

Placing Cookies in the Response Headers

The cookie is added to the Set-Cookie response header by means of the addCookie method of HttpServletResponse. Here's an example:

```
  Cookie userCookie = new Cookie("user",
"uid1234");
  response.addCookie(userCookie);
// before opening the body of response!
// i.e. before any out.print
```

Reading Cookies from the Client

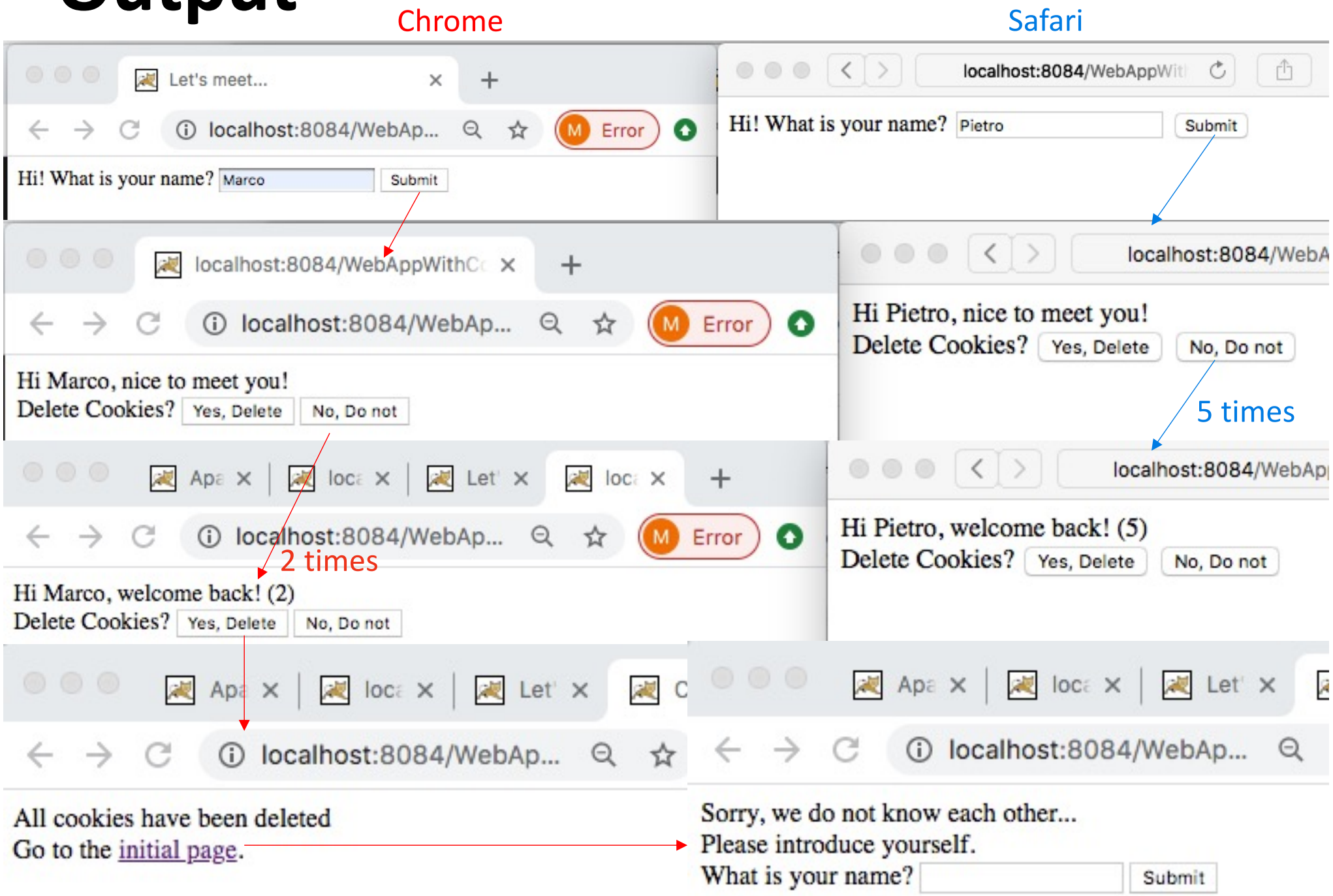
- To read the cookies that come back from the client, you call getCookies on the HttpServletRequest. This returns an array of Cookie objects corresponding to the values that came in on the Cookie HTTP request header.

- Once you have this array, you typically loop down it, calling getName on each Cookie until you find one matching the name you have in mind. You then call getValue on the matching Cookie, doing some processing specific to the resultant value.

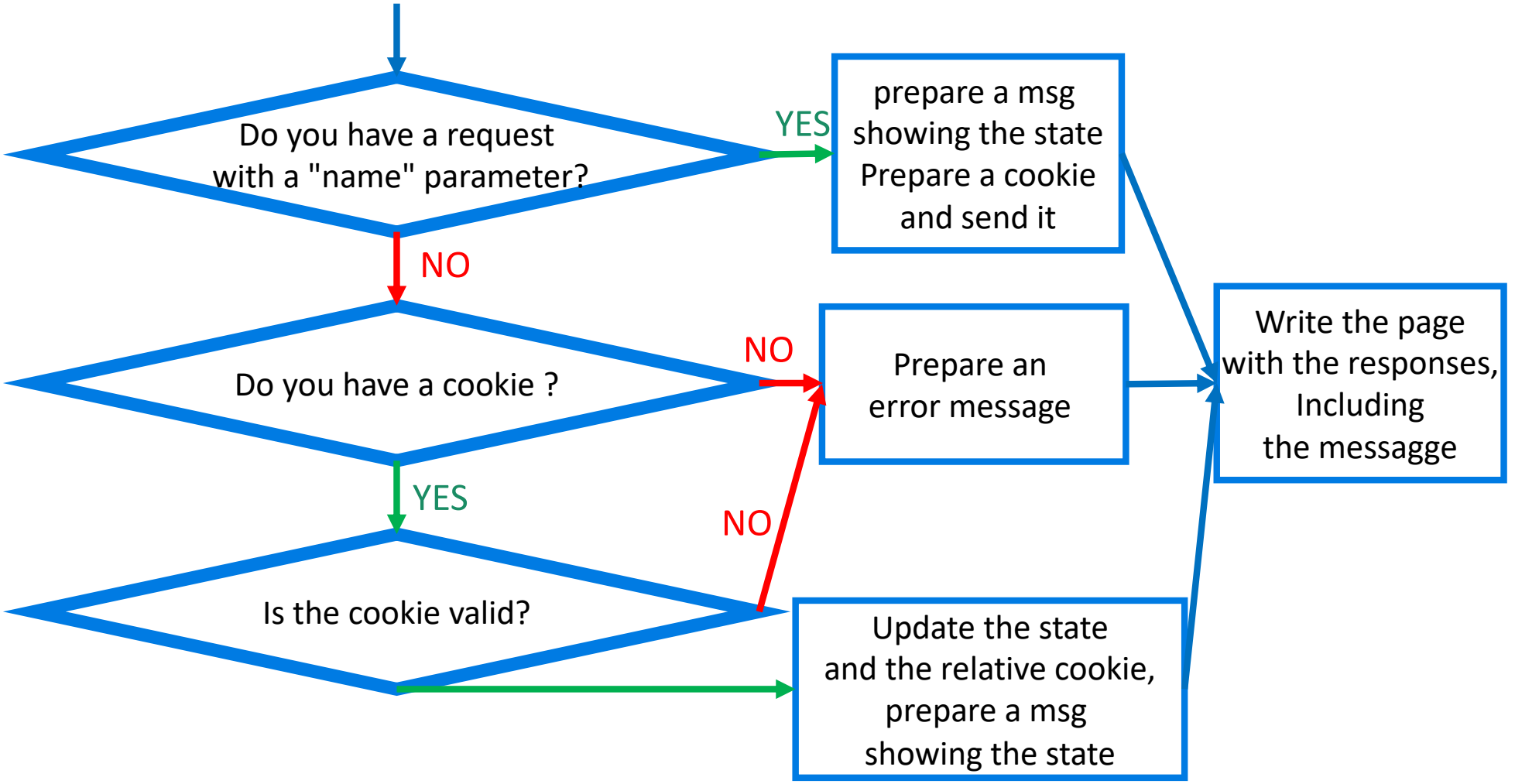
Cookies

Demo: Cookies in action

Output


Chrome
Safari
Let's meet...
localhost:8084/WebAp...
Error
Hi! What is your name? Marco
Submit
localhost:8084/WebAppWit
Hi! What is your name? Pietro
Submit
localhost:8084/WebAppWithC
Hi Marco, nice to meet you!
Delete Cookies? Yes, Delete
No, Do not
Hi Pietro, nice to meet you!
Delete Cookies? Yes, Delete
No, Do not
5 times
2 times
Hi Marco, welcome back! (2)
Delete Cookies? Yes, Delete
No, Do not
Hi Pietro, welcome back! (5)
Delete Cookies? Yes, Delete
No, Do not
All cookies have been deleted
Go to the initial page.
Sorry, we do not know each other...
Please introduce yourself.
What is your name?
Submit

Cookies in action - outline



Cookies in action - 1

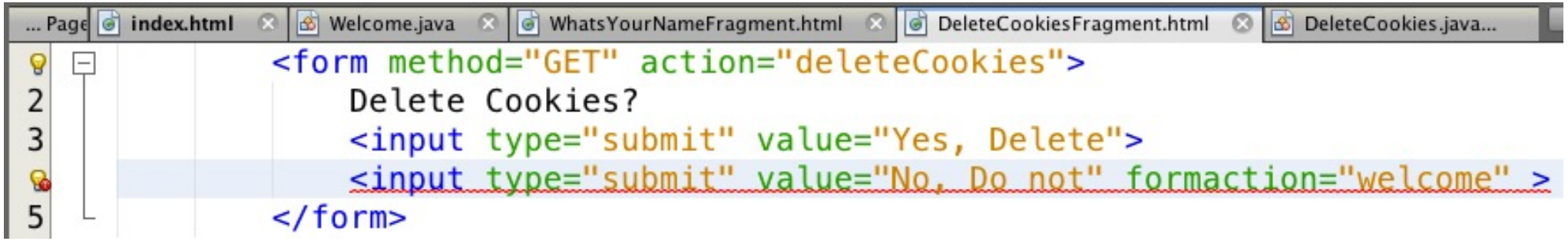
index.html | Welcome.java | WhatsYourNameFragment.html | DeleteCookiesFragment.html | DeleteCookies.java...

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>Let's meet...</title>
5          <meta charset="UTF-8">
6          <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      </head>
8      <body>
9          <form method="GET" action="welcome">
           <label for="name">Hi! What is your name?</label>
11         <input type="text" name="name">
12         <input type="submit" >
13         </form>
14     </body>
15 </html>
```

index.html | Welcome.java | WhatsYourNameFragment.html | DeleteCookiesFragment.html

```
        <form method="GET" action="welcome">
            <label for="name">What is your name?</label>
3           <input type="text" name="name" value="">
4           <input type="submit" >
5       </form>
```

Cookies in action - 2



Cookies in action - 3

```
package it.unitn.disi.ronchet.myservlets;

import …

@WebServlet(urlPatterns = {"/welcome"})
public class Welcome extends HttpServlet {

    String msg;
    boolean isInitialIteration ;

    private void dealWithInvalidCookie() {
        msg = "Sorry, we do not know each other...<br>"
                + "Please introduce yourself.<br>";
        isInitialIteration = true;
    }
```

Cookies in action - 4

```
 protected void doGet(HttpServletRequest request,
`              HttpServletResponse response)
               throws ServletException, IOException {
        isInitialIteration=false;
        // manage params and cookies
        String name = request.getParameter("name");
        if (name != null && ! name.equals("")) {
            // there is the right parameter,
            // no need to read cookie, but we set them
            log("name != null && ! name.equals(\"\")");
            Cookie cookie = new Cookie("name", name);
            msg = "Hi " + name + ", nice to meet you!";
            response.addCookie(cookie); // identity
            Cookie cookie1 = new Cookie("counter", "0");
            response.addCookie(cookie1); // state
            // finished! Go to end
        } else {
```

Cookies in action - 5

```
//} else {
      // no parameter, let's try with cookies
      Cookie cookies[] = request.getCookies();
      if (cookies==null || cookies.length == 0) {
          // no cookies
          log(”no cookies found");
          dealWithInvalidCookie();
      } else {
          Cookie n_Cookie=null; // cookie con il nome
          Cookie c_Cookie=null; // cookie con il contatore
          for (Cookie c:cookies) {
              String cookieName = c.getName();
              if (cookieName.equals("name")) {
                  n_Cookie=c;
              } else if (cookieName.equals("counter")) {
                  c_Cookie=c;
              }
          }
          if (n_Cookie==null) {
              log ("valid cookies not found");
              // invalid cookie
              dealWithInvalidCookie();
          } else {
```

Cookies in action - 6

```
            } else {
                // ok, the cookie is good!
                String userName=n_Cookie.getValue();
                String counterAsString=c_Cookie.getValue();
                log ("name == "+userName);
                // let's update the counter, and the cookie
                int counter=Integer.valueOf(counterAsString)+1;
                c_Cookie.setValue(""+counter);
                response.addCookie(c_Cookie);
                msg = "Hi " + userName + ", welcome back! (”
                       +counter+")";}
        }
    }
```

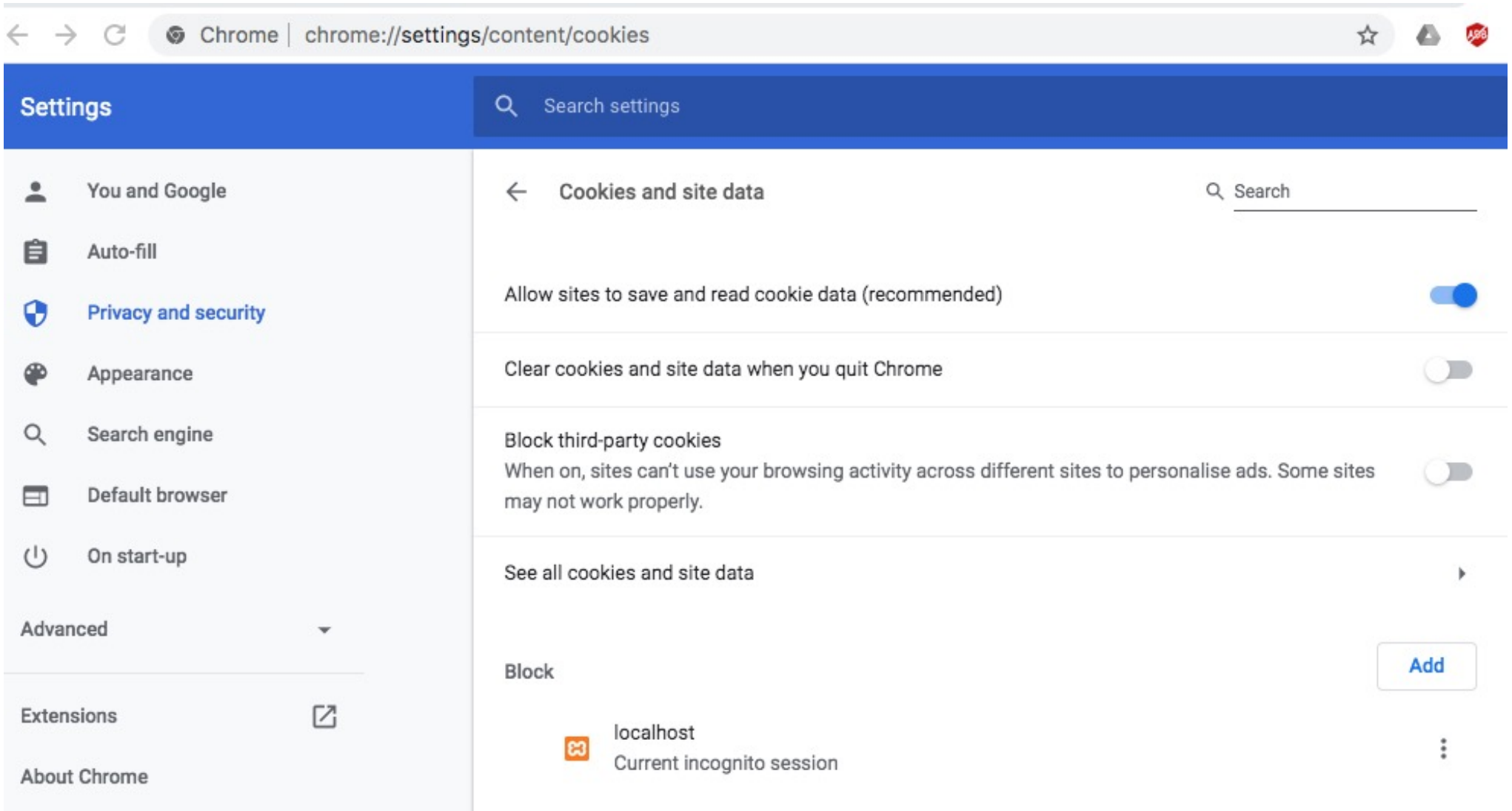
Cookies in action - 7

```
// prepare response and send it
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            out.println("<!DOCTYPE html>");
            out.println("<html><body>");
            out.println(msg);
            if (isInitialIteration) {
                request.getRequestDispatcher(
                    "WhatsYourNameFragment.html")
                        .include(request, response);
            } else {
                request.getRequestDispatcher(
                    "DeleteCookiesFragment.html")
                        .include(request, response);
            }
            out.println("</body></html>");
        }

    }
```

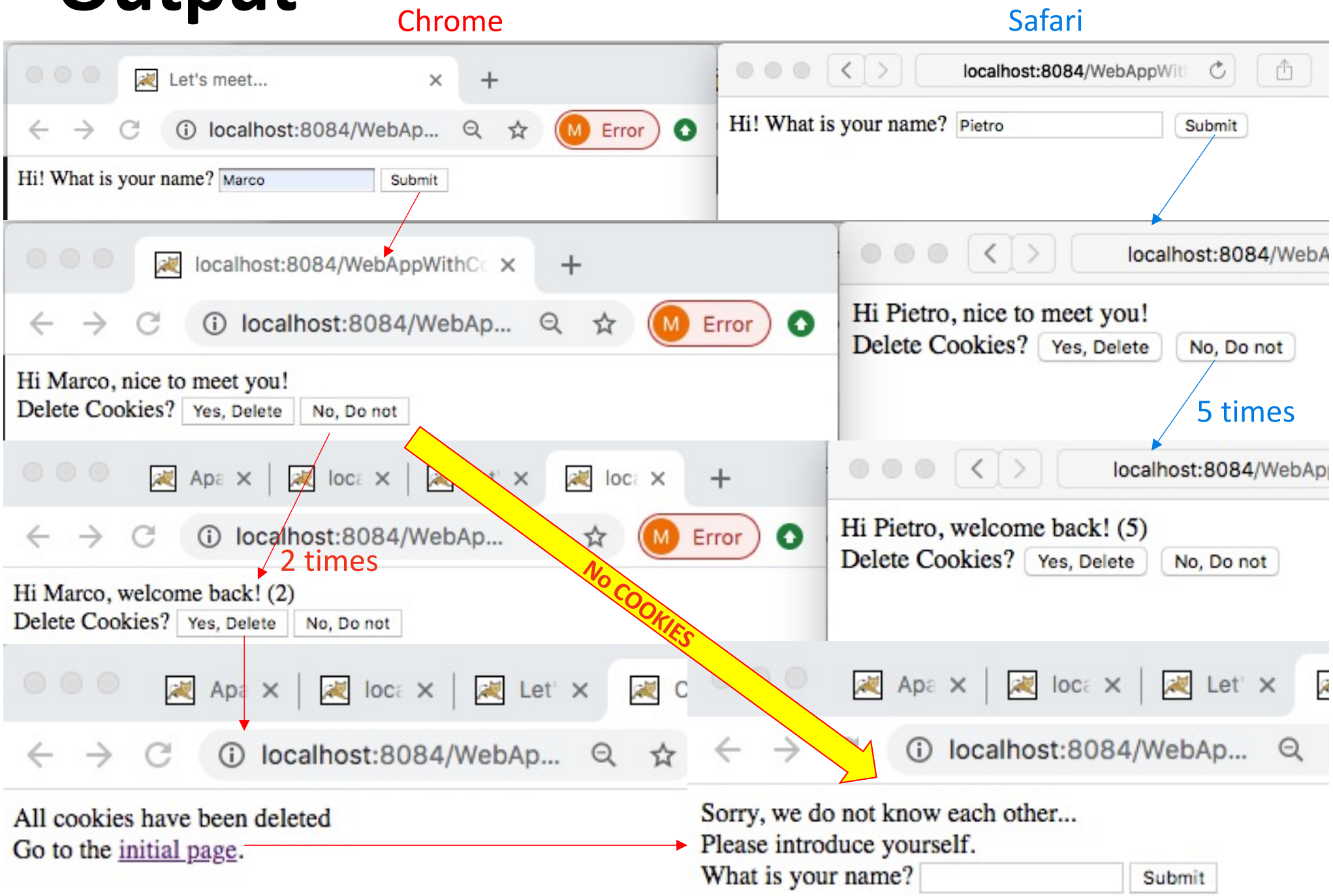
Cookies in action – 8 - deleteCookies

```
@WebServlet(name = "DeleteCookies",
        urlPatterns = {"/deleteCookies"})
public class DeleteCookies extends HttpServlet {

    protected void doGet(HttpServletRequest request,
              HttpServletResponse response)
              throws ServletException, IOException {
        Cookie cookies[]=request.getCookies();
        if (cookies != null) {
            for (Cookie c : cookies) {
                c.setMaxAge(0);
                response.addCookie(c);
            }
        }
        response.setContentType("text/html;charset=UTF-8");
        request.getRequestDispatcher(
              "CookiesHaveBeenDeleted.html")
                 .include(request, response);
    }
}
```

Disallowing cookies



Output


Chrome
Safari
Let's meet...
localhost:8084/WebAp...
Error
Hi! What is your name? Marco
Submit
localhost:8084/WebAppWit
Hi! What is your name? Pietro
Submit
localhost:8084/WebAppWithC
Hi Marco, nice to meet you!
Delete Cookies? Yes, Delete
No, Do not
Hi Pietro, nice to meet you!
Delete Cookies? Yes, Delete
No, Do not
5 times
2 times
Hi Marco, welcome back! (2)
Delete Cookies? Yes, Delete
No, Do not
Hi Pietro, welcome back! (5)
Delete Cookies? Yes, Delete
No, Do not
No COOKIES
All cookies have been deleted
Go to the initial page.
Sorry, we do not know each other...
Please introduce yourself.
What is your name?
Submit

What are sessions ?

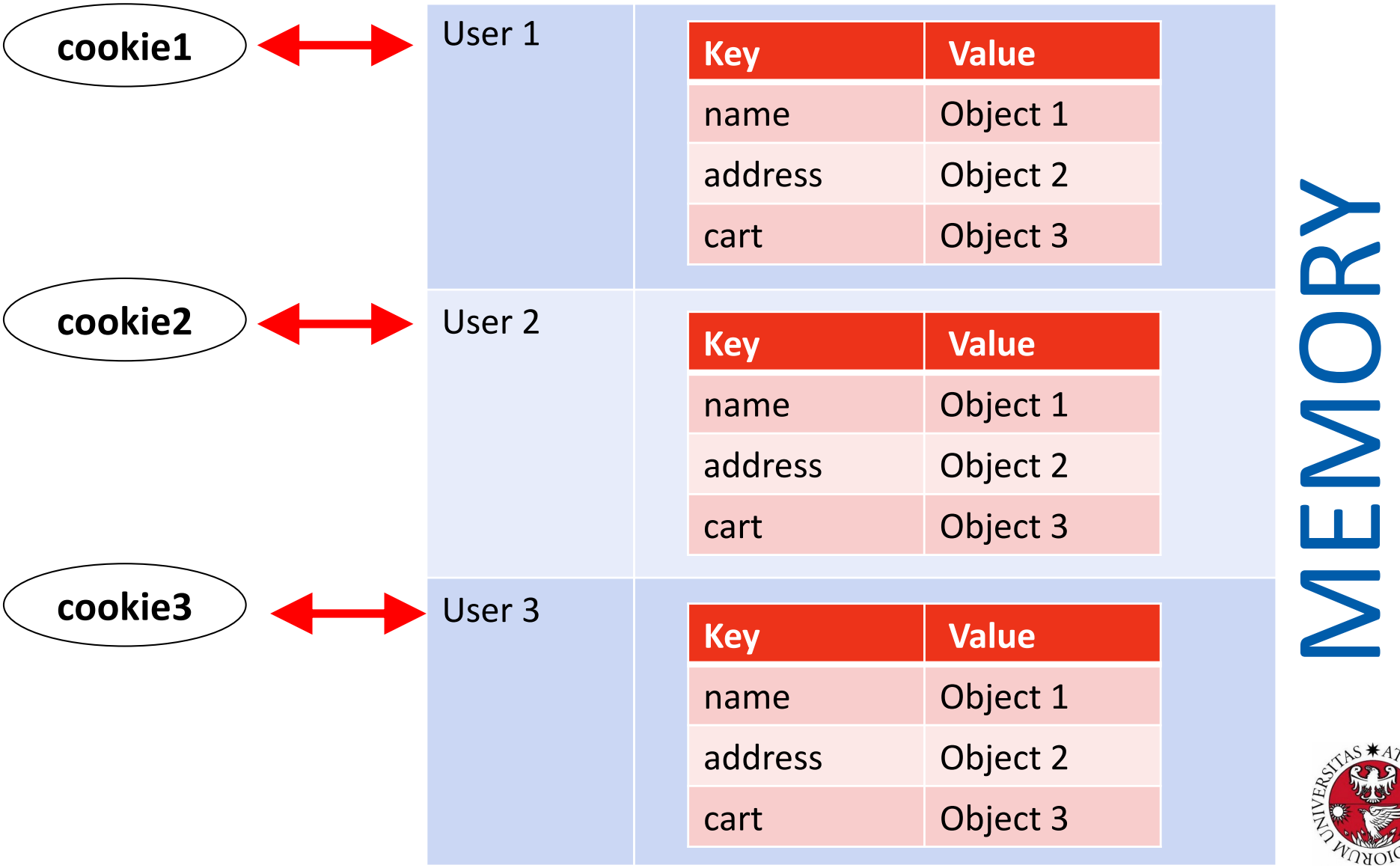
Session concept

- To support applications that need to maintain state, Java Servlet technology provides an API for managing sessions and allows several mechanisms for implementing sessions.

- Sessions are represented by an HttpSession object.

Session tracking using cookies

The idea:



Session tracking

- To associate a session with a user, a web container can use several methods, all of which involve passing an identifier between the client and the server. The identifier can be
- maintained on the client as a cookie, or
- the web component can include the identifier in every URL that is returned to the client. See URL rewriting later)

Session

Java HttpSession

Accessing Session

- You access an HttpSession object by calling the getSession method of a request object.
- This method returns the current session associated with this request; or, if the request does not have a session, this method creates one.

Associating objects with Session

You can associate object-valued attributes with an HttpSession by name.

Such attributes are accessible by any web component that belongs to the same web context *and* is handling a request that is part of the same session.

HttpSession methods: attributes

- public Enumeration **getAttributeNames**()
  - Returns an Enumeration of String objects containing the names of all the objects bound to this session.
- public Object **getAttribute**(String name)
  - Returns the object bound with the specified name in this session, or null if no object is bound under the name.
- public void **setAttribute**(String name, Object value)
  - Binds an object to this session, using the name specified. If an object of the same name is already bound to the session, the object is replaced.
- public void **removeAttribute**(String name)
  - Removes the object bound with the specified name from this session. If the session does not have an object bound with the specified name, this method does nothing.

Session lifecycle

- A session consumes resources (memory), hence it has to be managed. Since http is stateless, there is no notion of “log out”.
- The way of solving the problem, is to decide an expiry time for sessions (timeout)

HttpSession methods: timing

- public long **getCreationTime**()
  - Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
- public long **getLastAccessedTime**()
  - Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT, and marked by the time the container received the request.
- public void **setMaxInactiveInterval**(int interval)
  - Specifies the time, in seconds, between client requests before the servlet container will invalidate this session. A negative time indicates the session should never timeout.
- public int **getMaxInactiveInterval**()
  - Returns the maximum time interval, in seconds, that the servlet container will keep this session open between client accesses. After this interval, the servlet container will invalidate the session.

Setting Session global Timeout

To set the timeout period in the deployment descriptor using NetBeans IDE, follow these steps.

- Expand the node of your project in the **Projects** tab.
- Expand the **Web Pages** and **WEB-INF** nodes that are under the project node.
- If WebInf is empty, select it and right-click new->other->Standard Deployment Descriptor
- Double-click web.xml
- If not present, add

```
<session-config>
    <session-timeout>
        30
    </session-timeout>
</session-config>
```

(30 is the number of minuts after which the session will expire)

web.xml

Java web applications use a deployment descriptor file named web.xml to determine many things, such as how URLs map to servlets, which URLs require authentication, etc..

web.xml resides in the app's WAR under the WEB-INF/ directory.

See https://cloud.google.com/appengine/docs/standard/java/config/webxml

Why did we not use web.xml so far?

Some of the info expected in the web.xml can be provided via annotation. E.g.

```
package it.unitn.disi.ronchet.myservlets;

@WebServlet(name="myServlet",
        urlPatterns = {"/welcome"})
public class Welcome extends HttpServlet
```

Is equivalent to

```
</web-app>
    <servlet>
        <servlet-name>myServlet</servlet-name>
        <servlet-class>it.unitn.disi.ronchet.myservlets.Welcome
        </servlet-class>
    </servlet>
    <servlet-mapping
        <servlet-name>myServlet</servlet-name>
        <url-pattern>/welcome</url-pattern>
    </servlet-mapping>
</web-app>
```

web.xml and annotations together

Whatever is defined in web.xml overwrites annotations.

Try it!

Redefine the URL via web.xml, and see who wins between annotation and configuration.

HttpSession methods: other

- public java.lang.String **getId**()
  - Returns a string containing the unique identifier assigned to this session. The identifier is assigned by the servlet container and is implementation dependent.
- public boolean **isNew**()
  - Returns true if the client does not yet know about the session or if the client chooses not to join the session (e.g., if client had disabled the use of cookies).
- public void **invalidate**()
  - Invalidates this session then unbinds any objects bound to it.

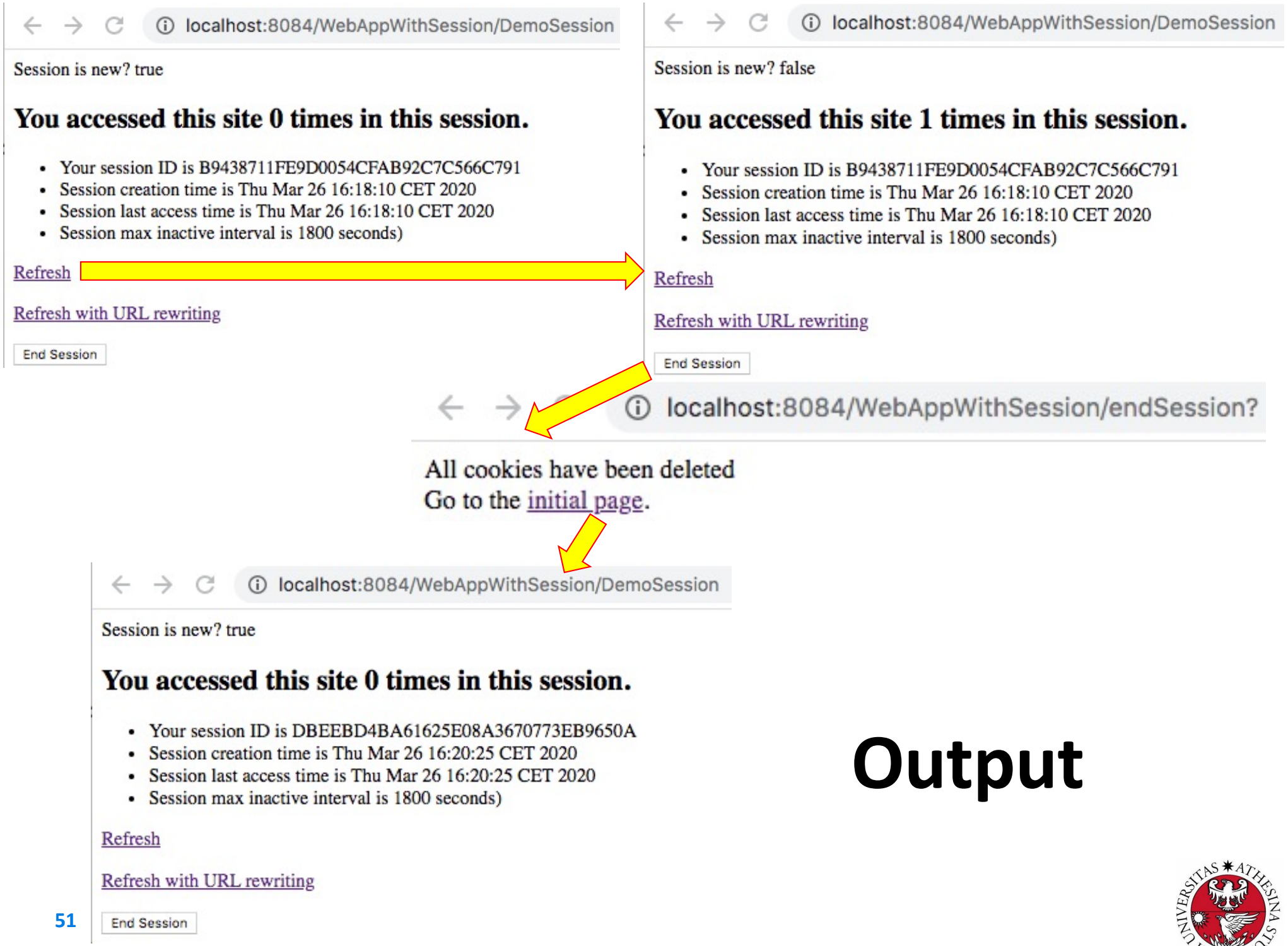
Session tracking

- If your application uses session objects, you must ensure that session tracking is enabled by having the application rewrite URLs whenever the client turns off cookies.

- You do this by calling the response's encodeURL(URL) method on all URLs returned by a servlet.
- This method includes the session ID in the URL only if cookies are disabled; otherwise, the method returns the URL unchanged.

Session

Demo


localhost:8084/WebAppWithSession/DemoSession
Session is new? true
You accessed this site 0 times in this session.
Your session ID is B9438711FE9D0054CFAB92C7C566C791
Session creation time is Thu Mar 26 16:18:10 CET 2020
Session last access time is Thu Mar 26 16:18:10 CET 2020
Session max inactive interval is 1800 seconds)
Refresh
Refresh with URL rewriting
End Session
localhost:8084/WebAppWithSession/DemoSession
Session is new? false
You accessed this site 1 times in this session.
Your session ID is B9438711FE9D0054CFAB92C7C566C791
Session creation time is Thu Mar 26 16:18:10 CET 2020
Session last access time is Thu Mar 26 16:18:10 CET 2020
Session max inactive interval is 1800 seconds)
Refresh
Refresh with URL rewriting
End Session
localhost:8084/WebAppWithSession/endSession?
All cookies have been deleted
Go to the initial page.
localhost:8084/WebAppWithSession/DemoSession
Session is new? true
You accessed this site 0 times in this session.
Your session ID is DBEEBD4BA61625E08A3670773EB9650A
Session creation time is Thu Mar 26 16:20:25 CET 2020
Session last access time is Thu Mar 26 16:20:25 CET 2020
Session max inactive interval is 1800 seconds)
Refresh
Refresh with URL rewriting
End Session

Output

Session in action - 1

```
package it.unitn.disi.ronchet.myservlets;

import …

 @WebServlet(urlPatterns = {"/DemoSession"})
public class DemoSession extends HttpServlet {

    PrintWriter out=null;
    private void p(String s) {
       out.println(s);
   }
```


Session in action - 2

```
@Override
   public void doGet(HttpServletRequest request,
                     HttpServletResponse response)
                     throws IOException, ServletException {
      out = response.getWriter();

      // Return the existing session if there is one.
      // Create a new session otherwise.
      HttpSession session = request.getSession();
      Integer accessCount;
      synchronized(session) {
         accessCount =
              (Integer)session.getAttribute("accessCount");
         if (accessCount == null) {
            accessCount = 0;   // autobox int to Integer
         } else {
            accessCount = new Integer(accessCount + 1);
         }
         session.setAttribute("accessCount", accessCount);
      }
```

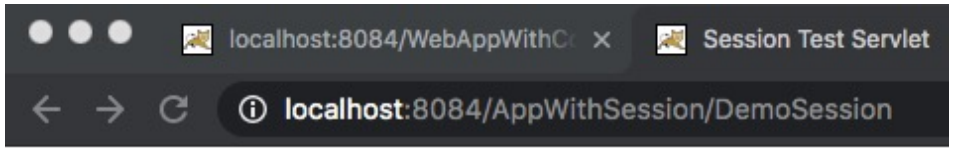

Session in action - 3

```
try {
  response.setContentType("text/html;charset=UTF-8");
  p("<!DOCTYPE html>"
      +"<html>"
      +"<head><title>Session Test Servlet</title></head><body>")
  p("Session is new? "+session.isNew());
  p("<h2>You accessed this site " + accessCount
      + " times in this session.</h2>");
  p("<ul><li>Your session ID is " + session.getId() + "</li>");
  p("<li>Session creation time is " +
      new Date(session.getCreationTime()) + "</li>");
  p("<li>Session last access time is " +
      new Date(session.getLastAccessedTime()) + "</li>");
  p("<li>Session max inactive interval  is " +
      session.getMaxInactiveInterval() + " seconds)</li></ul>");
```

Session in action - 4

```
   p("<p><a  href='" + request.getRequestURI()
      +  "'>Refresh</a>");
   p("<p><a  href='”
      + response.encodeURL(request.getRequestURI())
      + "'>Refresh with  URL rewriting</a>\n");
         p("<form method=\"GET\" action=\"endSession\">\n"
            +"<input type=\"submit\" value=\"End Session\">\n"
            +"</form>");
         p("</body></html>");
 } finally {
     out.close(); // Always close the output writer
 }
} // end DoGet
```

Without cookies…



You have access this site 0 times in this session.

(Session ID is ECB0CF247DD8C156A0EFE73FF9A3AA4A)

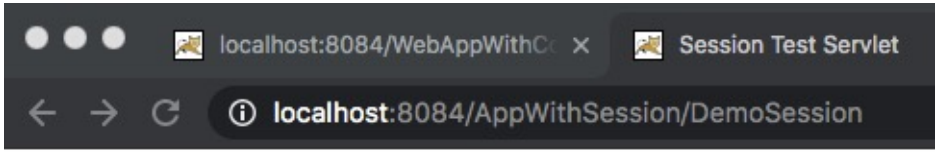
(Session creation time is Thu Mar 26 12:08:11 CET 2020)

(Session last access time is Thu Mar 26 12:08:11 CET 2020)

(Session max inactive interval is 1800 seconds)

Refresh

Refresh with URL rewriting



You have access this site 0 times in this session.

(Session ID is 0E13C8D5828616D80EA34BF2213CAB4C)

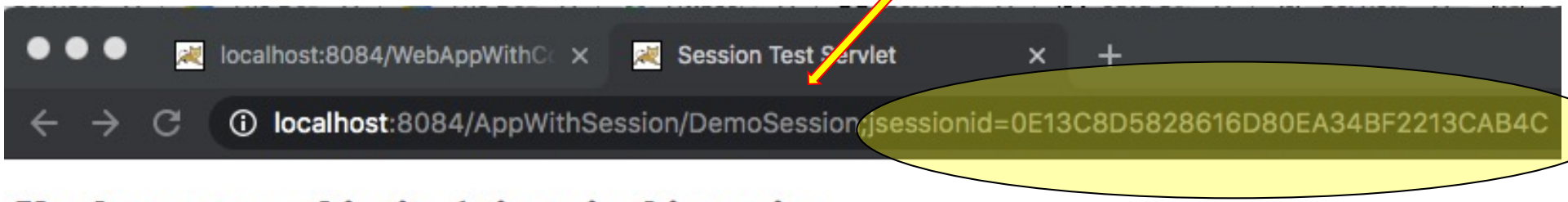
(Session creation time is Thu Mar 26 12:09:34 CET 2020)

(Session last access time is Thu Mar 26 12:09:34 CET 2020)

(Session max inactive interval is 1800 seconds)

Refresh

Refresh with URL rewriting



You have access this site 1 times in this session.

(Session ID is 0E13C8D5828616D80EA34BF2213CAB4C)

(Session creation time is Thu Mar 26 12:09:34 CET 2020)

(Session last access time is Thu Mar 26 12:09:34 CET 2020)

(Session max inactive interval is 1800 seconds)

Refresh

Refresh with URL rewriting

Session in action – 5 - endSession

```
package it.unitn.disi.ronchet.webProg;

import …

@WebServlet(name = "endSession", urlPatterns = {"/endSession"})
public class DeleteSession extends HttpServlet {

    protected void doGet(HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException {
      HttpSession s= request.getSession();
      s.invalidate();
      response.setContentType("text/html;charset=UTF-8");
      request.getRequestDispatcher("SessionHasBeenDeleted.html")
              .include(request, response);

    }
}
```

Session in action – 6 - SessionHasBeenDeleted.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>Session has been deleted</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width,
            initial-scale=1.0">
    </head>
    <body>
        All cookies have been deleted <br>
        Go to the <a href="/WebAppWithSession/DemoSession">
            initial page</a>.
    </body>
</html>
```

Session in action – 7 – web.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
             http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
         version="3.1">
    <welcome-file-list>
        <welcome-file>DemoSession</welcome-file>
    </welcome-file-list>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
</web-app>
```

Session

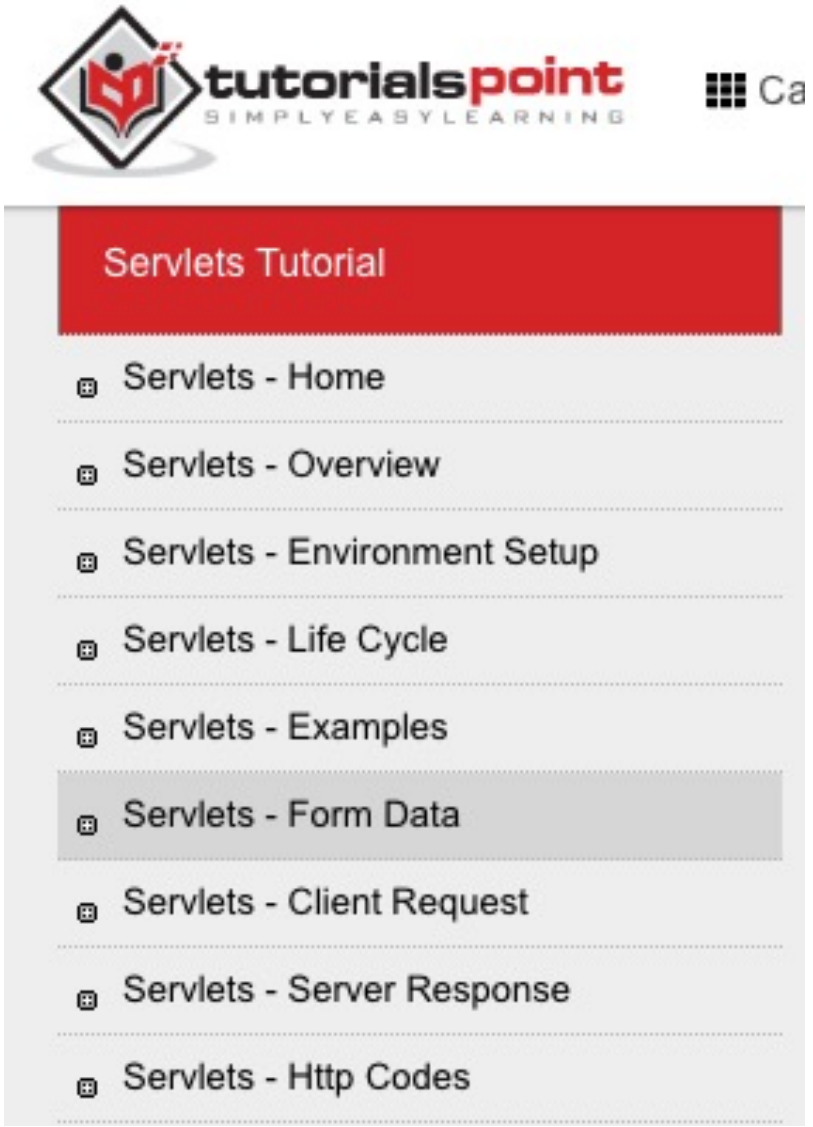
Final notes

Advanced: associating events with session objects

- Your application can notify web context and session listener objects of servlet lifecycle events (Handling Servlet Lifecycle Events). You can also notify objects of certain events related to their association with a session, such as the following:
  - When the object is added to or removed from a session. To receive this notification, your object must implement the javax.servlet.http.HttpSessionBindingListener interface.
  - When the session to which the object is attached will be passivated or activated. A session will be passivated or activated when it is moved between virtual machines or saved to and restored from persistent storage. To receive this notification, your object must implement the javax.servlet.http.HttpSessionActivationListener interface.

Readings

https://www.tutorialspoint.com/servlets/index.htm



Stop here!

ServletContext

Associating objects with the WebApp

```
ServletContext context = request.getSession().getServletContext();

context.setAttribute("someValue", "aValue");

Object attribute = context.getAttribute("someValue");
```

- The attributes stored in the ServletContext are available to all servlets in your application, and between requests and sessions. That means, that the attributes are available to all visitors of the web application. Session attributes are just available to a single user.
- The ServletContext attributes are still stored in the memory of the servlet container. That means that the same problems exists as does with the session attributes, in server clusters.

from http://tutorials.jenkov.com/java-servlets/servletcontext.html