

Getting deeper with Servlets

Let's build a hit counter

```
@Override
```

```
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
        response.setContentType("text/html;charset=UTF-8");  
    }
```

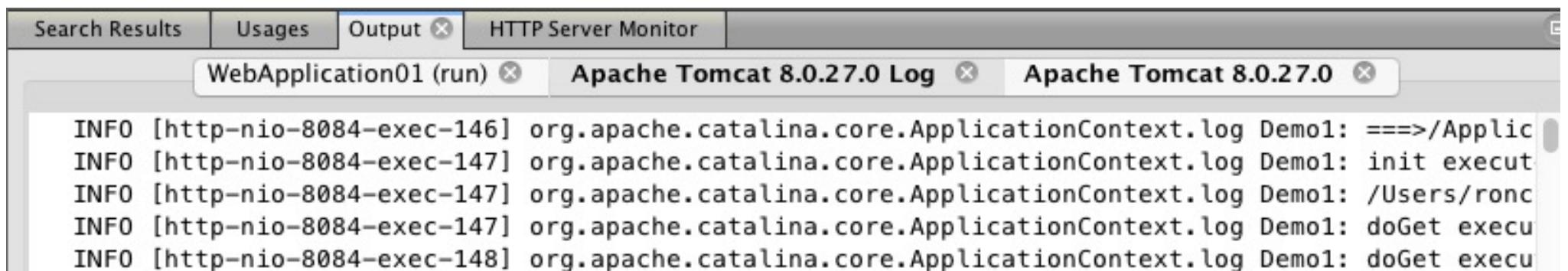
```
@Override
```

```
    public void destroy() {  
        log("destroy executed");  
    }
```

```
@Override
```

```
    public void init() {  
        log("init executed");  
    }
```

Logging



The screenshot shows the Eclipse IDE interface with the 'Output' tab selected. Below it, the 'Apache Tomcat 8.0.27.0 Log' tab is active, displaying the following log entries:

```
INFO [http-nio-8084-exec-146] org.apache.catalina.core.ApplicationContext.log Demo1: ===>/Applic  
INFO [http-nio-8084-exec-147] org.apache.catalina.core.ApplicationContext.log Demo1: init execut  
INFO [http-nio-8084-exec-147] org.apache.catalina.core.ApplicationContext.log Demo1: /Users/ronc  
INFO [http-nio-8084-exec-147] org.apache.catalina.core.ApplicationContext.log Demo1: doGet execut  
INFO [http-nio-8084-exec-148] org.apache.catalina.core.ApplicationContext.log Demo1: doGet execut
```

Let us build a hit counter - 1

```
public class Counter {  
    int count = 0;  
    Calendar timeStamp = Calendar.getInstance();  
    public void increase() {  
        count++;  
        timeStamp = Calendar.getInstance();  
    }  
    @Override  
    public String toString() {  
        StringBuffer s = null;  
        if (count == 0)  
            s = new StringBuffer("<p>no hits yet</p>");  
        else {  
            s = new StringBuffer("<p>hits = ");  
            s.append(count)  
                .append("<br>last hit on ")  
                .append(timeStamp.getTime().toString());  
        }  
        return s.toString();  
    }  
}
```

Let us build a hit counter - 2

```
@WebServlet(name = "Demo1", urlPatterns = {"/Demo1"})
public class Demo1 extends HttpServlet {
    Counter counter=new Counter();
    @Override
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            request.getRequestDispatcher("/fragment1.html")
                .include(request, response);
            counter.increase();
            out.println(counter);
            request.getRequestDispatcher("/fragment2.html")
                .include(request, response);
        }
    }
}
```



Output

fragment 1

hits = 1
last hit on Sun Mar 15 14:41:01 CET 2020

fragment 2

hits = 2
last hit on Sun Mar 15 14:41:24 CET 2020

fragment 1

hits = 3
last hit on Sun Mar 15 14:41:43 CET 2020

fragment 2

But if we restart the server, counter restarts from 1!

How can we persist the counter?

In a file (named `counterData`)!

- 1) In init, let us check if file exists. If yes, let us resume the counter, else, let us create a new one.
- 2) In destroy, let us save counter in conterData.

Java serialization

```
class A implements Serializable {...}
```

```
A a1=new A();
```

```
A a2;
```

```
...
```

```
File myFile = new File(filePath);
```

```
...
```

```
ObjectOutputStream oi = new ObjectOutputStream(new  
    FileOutputStream(myFile));
```

```
oi.writeObject(a1);
```

(throws various exceptions...)

```
...
```

```
ObjectInputStream oi = new ObjectInputStream(new  
    FileInputStream(myFile));
```

```
a2 = (A) oi.readObject();
```

(throws various exceptions...)



Let us build a hit counter - 1

```
public class Counter implements Serializable {  
    int count = 0;  
    Calendar timeStamp = Calendar.getInstance();  
    public void increase() {  
        count++;  
        timeStamp = Calendar.getInstance();  
    }  
    @Override  
    public String toString() {  
        StringBuffer s = null;  
        if (count == 0)  
            s = new StringBuffer("<p>no hits yet</p>");  
        else {  
            s = new StringBuffer("<p>hits = ");  
            s.append(count)  
                .append("<br>last hit on ")  
                .append(timeStamp.getTime().toString());  
        }  
        return s.toString();  
    }  
}
```



Cookies

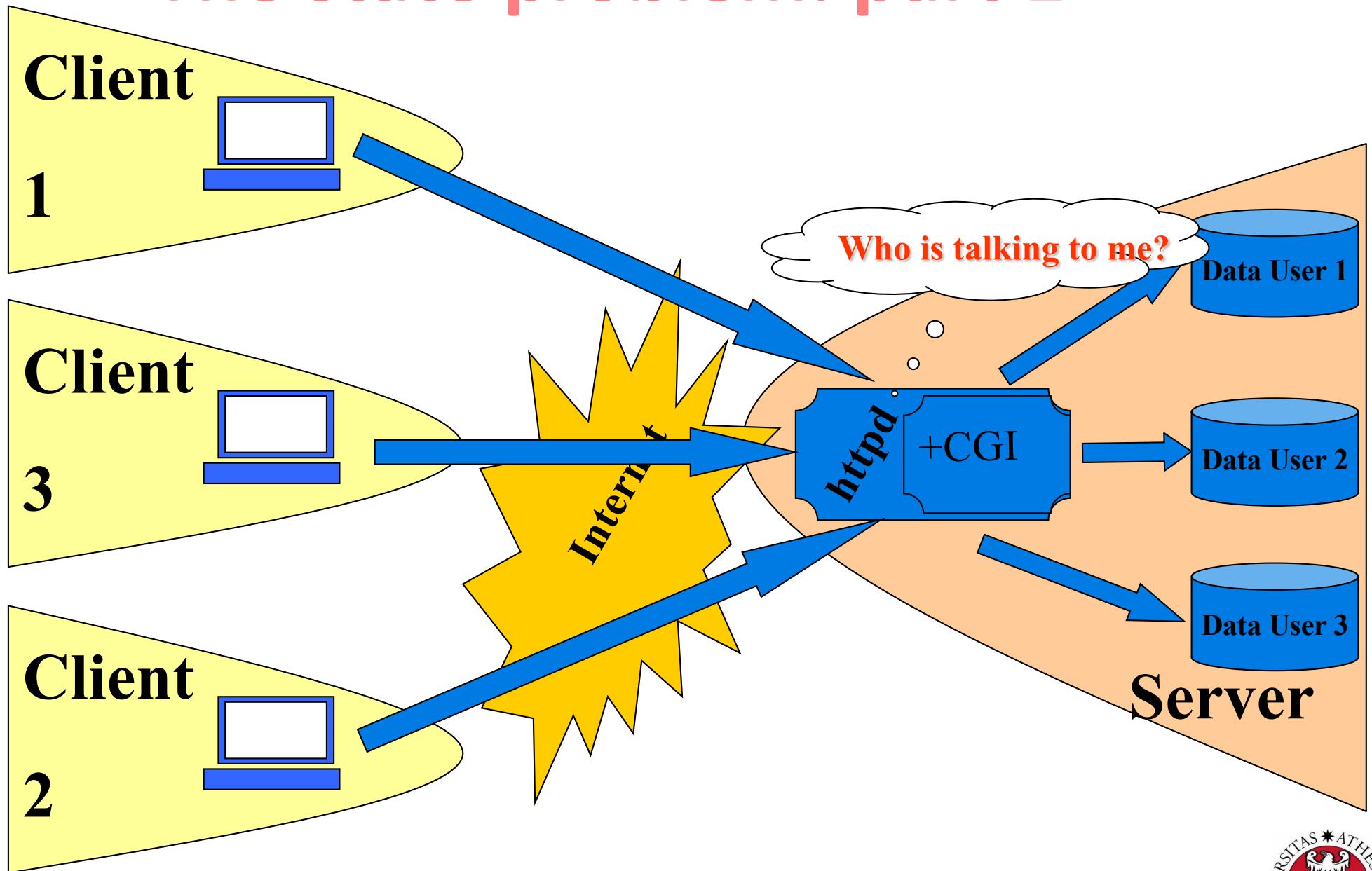
HTTP is stateless

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

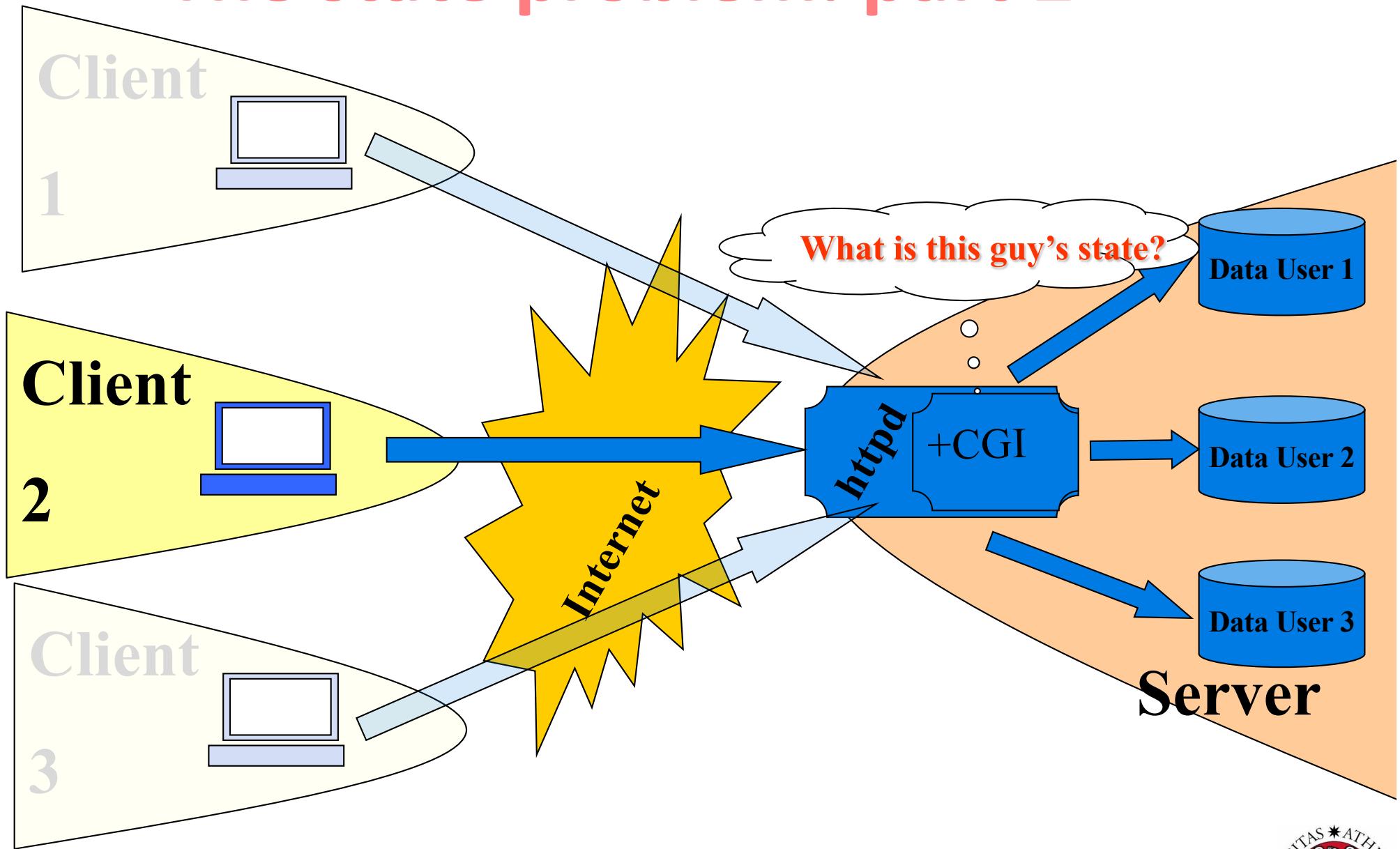
There is no way of doing it server-side...



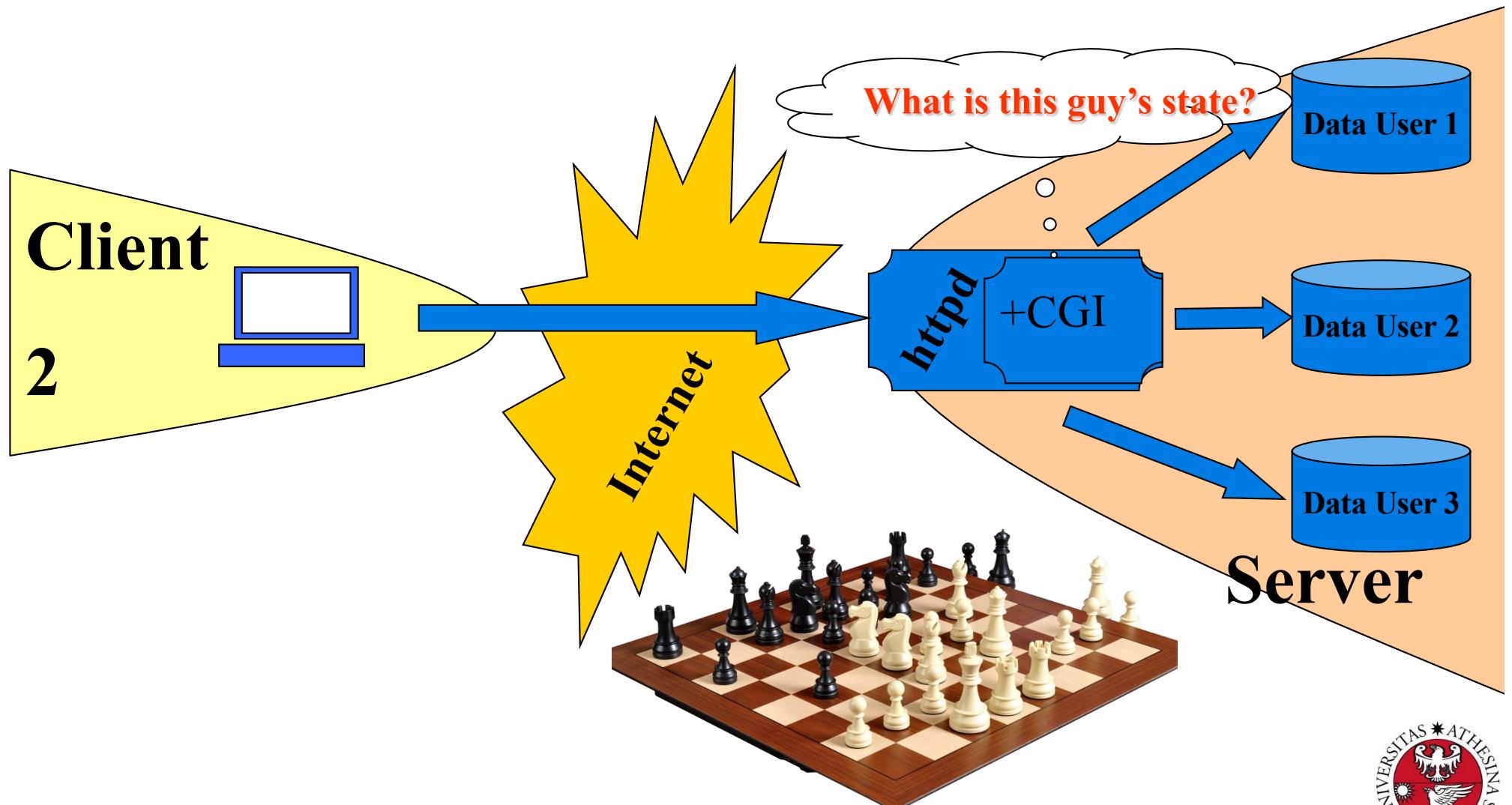
The state problem: part 1



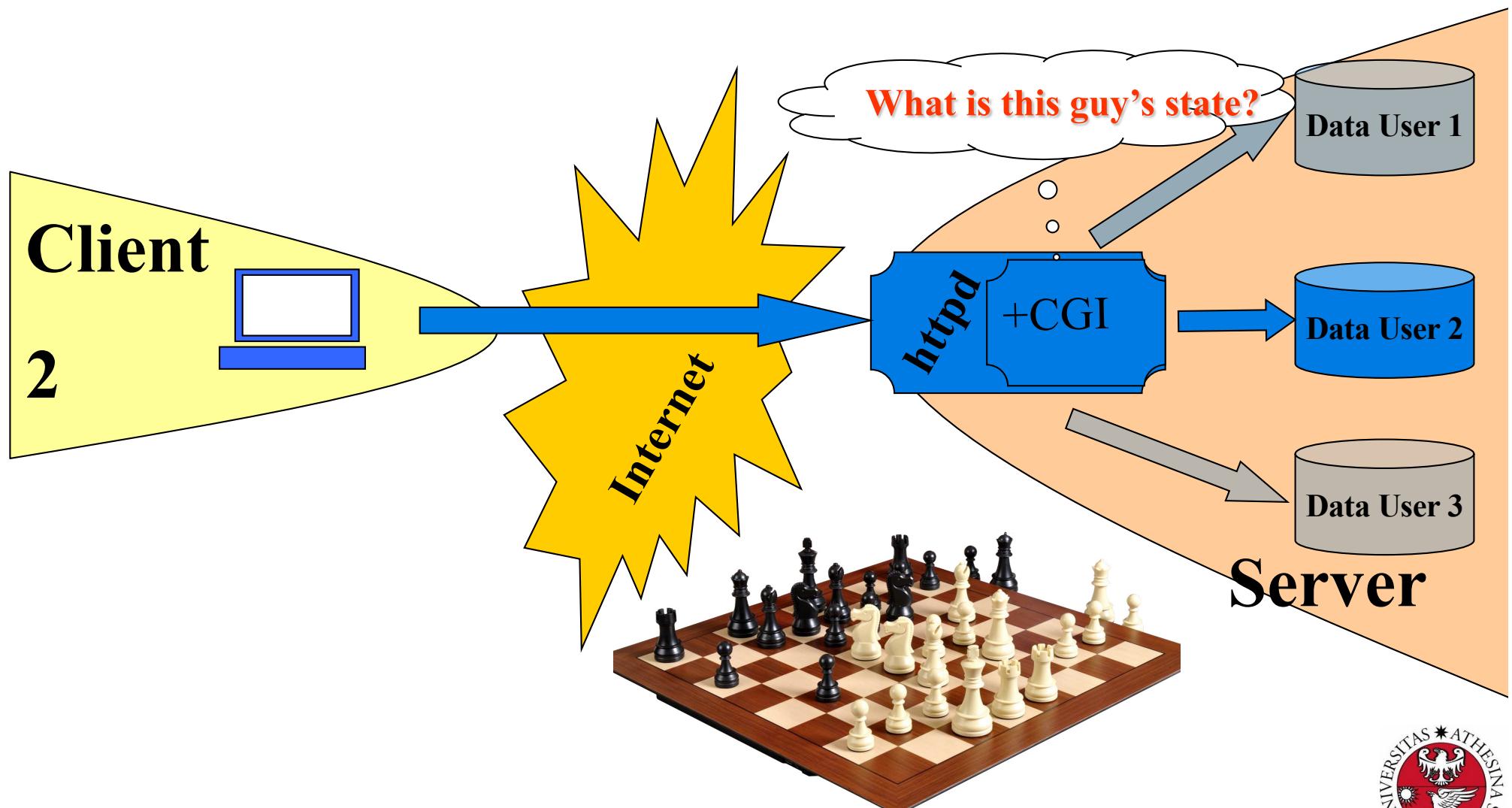
The state problem: part 2



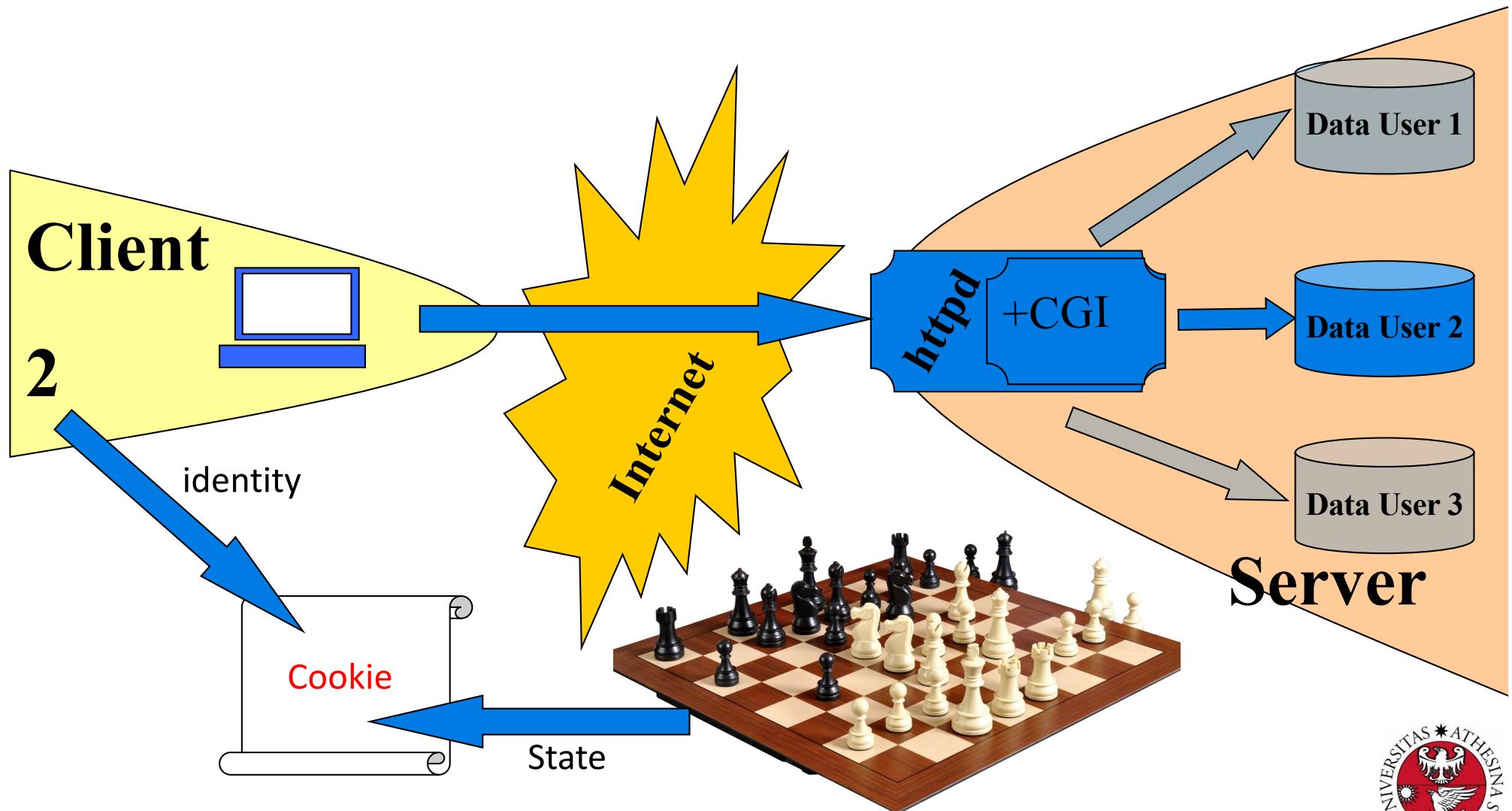
The state problem: part 2 - example



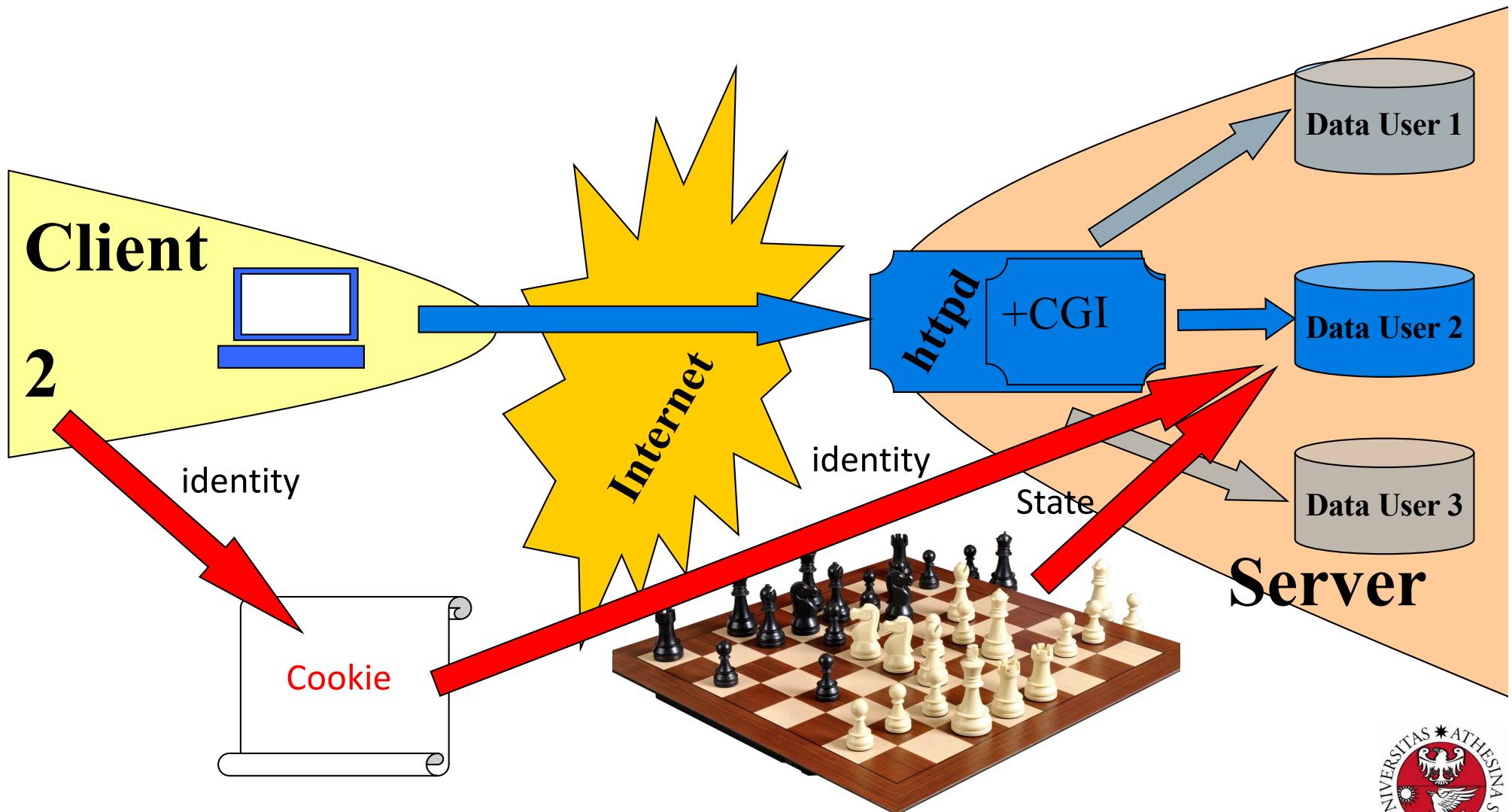
The state problem: part 2 - example



The state problem: solution 1



The state problem: solution 2



Cookies: what are they

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.



Where is the name coming from...



Cookies: why?

- To maintain status across a “user session”
- To maintain infos across multiple interactions:
- Customer identification
- Targeted advertisement
- Elimination of username e password



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.



Cookies and caching

- Cookies affect the caching of the Web pages that use them. HTTP 1.0 does not cache pages that use cookies created with this class.
- The Java class “**Cookie**” does not support the cache control defined with HTTP 1.1. This class supports both the Version 0 (by Netscape) and Version 1 (by RFC 2109) cookie specifications. By default, cookies are created using Version 0 to ensure the best interoperability



Attribute summary

- **String getComment() / void setComment(String s)**

- Gets/sets a comment associated with this cookie.

- **String getDomain() / setDomain(String s)**

- Gets/sets the domain to which cookie applies. Normally, cookies are returned only to the exact hostname that sent them. You can use this method to instruct the browser to return them to other hosts within the same domain. Note that the domain should start with a dot (e.g. .prenhall.com), and must contain two dots for non-country domains like .com, .edu, and .gov, and three dots for country domains like .co.uk and .edu.es.



Attribute summary

- **int getMaxAge() / void setMaxAge(int i)**
 - Gets/sets how much time (in seconds) should elapse before the cookie expires. If you don't set this, the cookie will last only for the current session (i.e. until the user quits the browser), and will not be stored on disk. See the `LongLivedCookie` class below, which defines a subclass of `Cookie` with a maximum age automatically set one year in the future.

- **String getName() / void setName(String s)**
 - Gets/sets the name of the cookie. The name and the value are the two pieces you virtually always care about. Since the `getCookies` method of `HttpServletRequest` returns an array of `Cookie` objects, it is common to loop down this array until you have a particular name, then check the value with `getValue`. See the `getCookieValue` method shown below.



Attribute summary

- **String getPath() / void setPath(String s)**

- Gets/sets the path to which this cookie applies. If you don't specify a path, the cookie is returned for all URLs in the same directory as the current page as well as all subdirectories. This method can be used to specify something more general. For example, someCookie.setPath("/") specifies that all pages on the server should receive the cookie. Note that the path specified must include the current directory.

- **boolean getSecure / setSecure(boolean b)**

- Gets/sets the boolean value indicating whether the cookie should only be sent over encrypted (i.e. SSL) connections.



Attribute summary

- **String getValue() / void setValue(String s)**
- Gets/sets the value associated with the cookie. Again, the name and the value are the two parts of a cookie that you almost always care about, although in a few cases a name is used as a boolean flag, and its value is ignored (i.e the existence of the name means true).

- **int getVersion() / void setVersion(int i)**
- Gets/sets the cookie protocol version this cookie complies with. Version 0, the default, adheres to the original Netscape specification. Version 1, not yet widely supported, adheres to RFC 2109.



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: examples

- **Cookie userCookie = new Cookie("user","uid1234");**
- **userCookie.setMaxAge(60*60*24*365);**
- **response.addCookie(userCookie);**



Cookies

Demo: setCookies - showCookies

SetCookies - 1

```
import ...  
  
public class SetCookies extends HttpServlet {  
  
    public void doGet(HttpServletRequest request,  
                      HttpServletResponse response) throws ServletException,  
                      IOException {  
  
        for(int i=0; i<3; i++) {  
  
            Cookie cookie = new Cookie("Session-Cookie-" + i,  
                                         "Cookie-Value-S" + i);  
  
            response.addCookie(cookie);  
  
            cookie = new Cookie("Persistent-Cookie-" + i,  
                                "Cookie-Value-P" + i);  
  
            cookie.setMaxAge(3600);  
  
            response.addCookie(cookie);  
        }  
    }  
}
```

Sets six cookies:

- three that apply only to the current session (regardless of how long that session lasts)
- three that persist for five minutes (regardless of whether the browser is restarted).

Default maxAge is -1, indicating cookie applies only to current browsing session

Cookie is valid for an hour, regardless of whether user quits browser, reboots computer, or whatever.



SetCookies - 2

```
response.setContentType("text/html");

PrintWriter out = response.getWriter();

String title = "Setting Cookies";

out.println ("<HTML><HEAD><TITLE>" +title+ "</TITLE></HEAD>" +
"<BODY BGCOLOR=\\"#FDF5E6\\">" +"<H1 ALIGN=\\"CENTER\\">" +
title + "</H1>" +
"There are six cookies associated with this page.\n" +
"</BODY></HTML>");

}

}
```



ShowCookies - 1

Creates a table of the cookies associated with the current page

```
import ...;

public class ShowCookies extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse
                       response) throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String title = "Active Cookies";
        out.println("<HTML><HEAD><TITLE>" + title + "</TITLE></HEAD>" +
                   "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                   "<H1 ALIGN=\"CENTER\">" + title + "</H1>\n" +
                   "<TABLE BORDER=1 ALIGN=\"CENTER\">\n" +
                   "<TR BGCOLOR=\"#FFAD00\">\n" +
                   "  <TH>Cookie Name\n" + "  <TH>Cookie Value");
    }
}
```



ShowCookies - 2

```
Cookie[] cookies = request.getCookies();

Cookie cookie;

for(int i=0; i<cookies.length; i++) {

    cookie = cookies[i];

    out.println("<TR>\n" +
               "  <TD>" + cookie.getName() + "\n" +
               "  <TD>" + cookie.getValue()));

}

out.println("</TABLE></BODY></HTML>");

}
```



Cookies

Demo: Cookies in action

Output

Chrome

Hi! What is your name? Marco

Hi Marco, nice to meet you!
Delete Cookies?

All cookies have been deleted
Go to the [initial page](#).

Safari

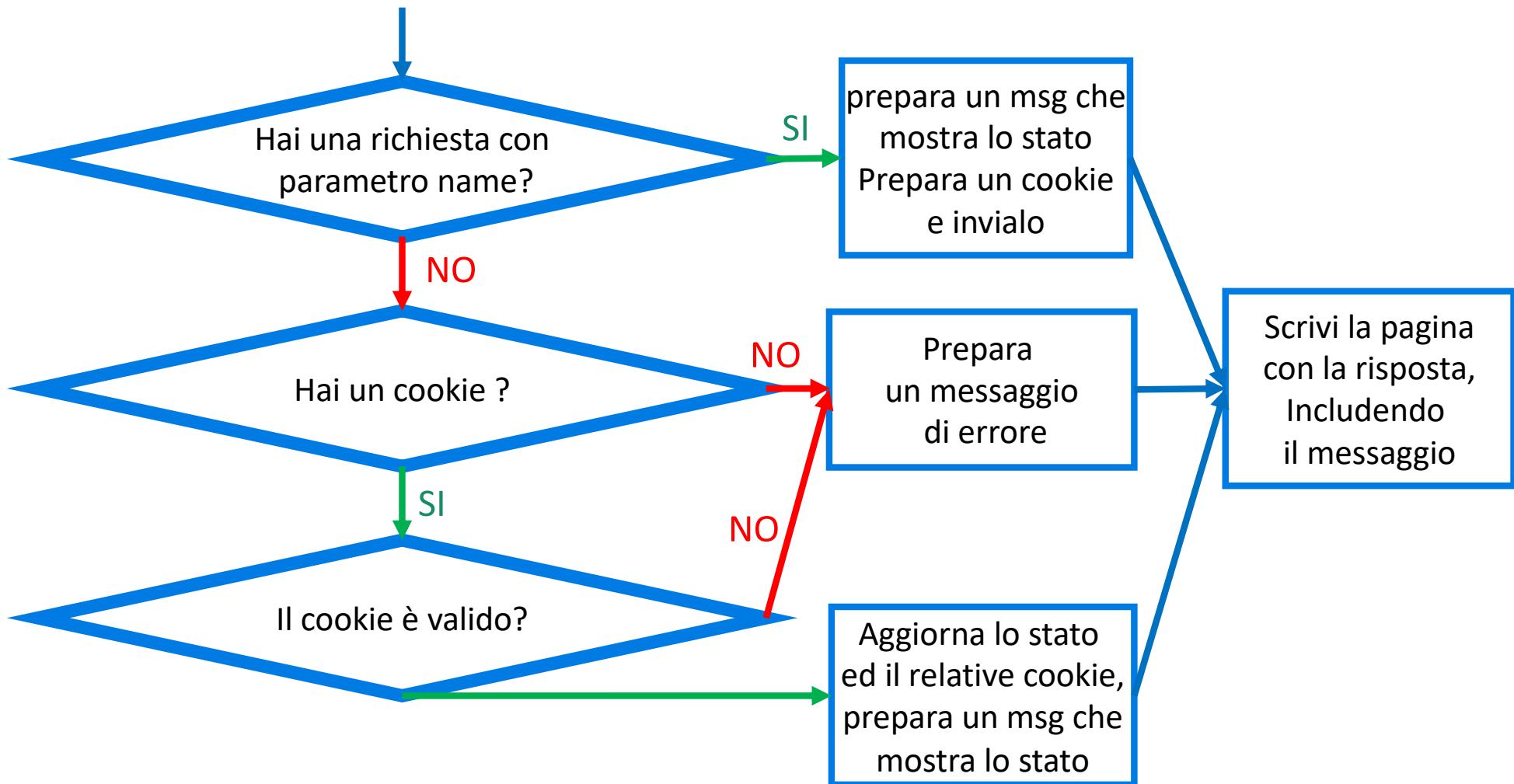
Hi! What is your name? Pietro

Hi Pietro, nice to meet you!
Delete Cookies?

Hi Pietro, welcome back! (5)
Delete Cookies?

Sorry, we do not know each other...
Please introduce yourself.
What is your name?

Cookies in action - outline



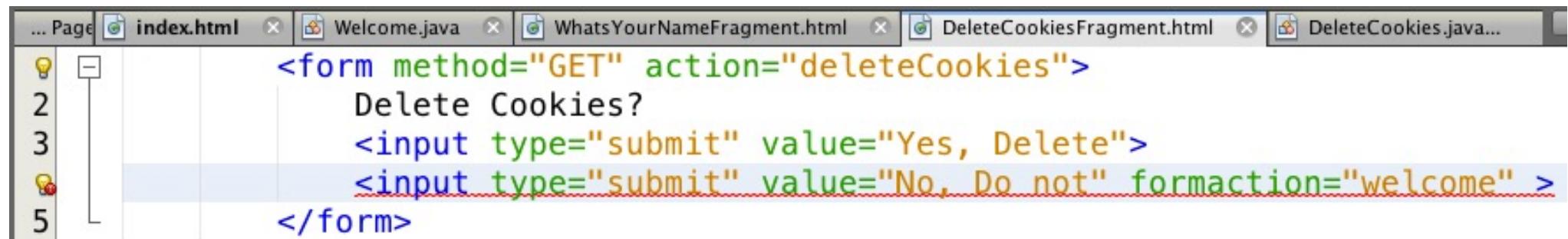
Cookies in action - 1

```
... Page 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">
10       <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>
```

```
... Page index.html Welcome.java WhatsYourNameFragment.html DeleteCookiesFragment.html
1 <form method="GET" action="welcome">
2   <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



The screenshot shows a code editor with several tabs at the top: "... Page", "index.html", "Welcome.java", "WhatsYourNameFragment.html", "DeleteCookiesFragment.html", and "DeleteCookies.java...". The "index.html" tab is active. The code in the editor is:

```
<form method="GET" action="deleteCookies">
    Delete Cookies?
    <input type="submit" value="Yes, Delete">
    <input type="submit" value="No, Do not" formaction="welcome" >
</form>
```



The screenshot shows a code editor with several tabs at the top: "...java", "WhatsYourNameFragment.html", "DeleteCookiesFragment.html", "DeleteCookies.java", and "CookiesHaveBeenDeleted.html". The "CookiesHaveBeenDeleted.html" tab is active. The code in the editor is:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Cookies have 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="/WebAppWithCookies/welcome">initial page</a>.
    </body>
</html>
```

The left sidebar shows the project structure of "WebAppWithCookies" with files like "index.html", "DeleteCookiesF", "WhatsYourNameFragment.html", and "Welcome.java".

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);
    }
}
```



Cookies in PHP

https://www.w3schools.com/php/php_cookies.asp



Cookies: legal aspects

Italian regulations

Individuazione delle modalità semplificate per l'informativa e l'acquisizione del consenso per l'uso dei cookie - 8 maggio 2014

- <http://garanteprivacy.it/web/guest/home/docweb/-/docweb-display/docweb/3118884>
- <https://www.garanteprivacy.it/home/doveri>
- <https://it.wikipedia.org/wiki/Regolamento GENERALE SULLA PROTEZIONE DEI DATI>



Italian regulations

- a. **Cookie tecnici.**

- I cookie tecnici sono quelli utilizzati al solo fine di "effettuare la trasmissione di una comunicazione su una rete di comunicazione elettronica, o nella misura strettamente necessaria al fornitore di un servizio della società dell'informazione esplicitamente richiesto dall'abbonato o dall'utente a erogare tale servizio" (cfr. art. 122, comma 1, del Codice).



Italian regulations

- Essi non vengono utilizzati per scopi ulteriori e sono normalmente installati direttamente dal titolare o gestore del sito web. Possono essere suddivisi in
- **cookie di navigazione o di sessione**, che garantiscono la normale navigazione e fruizione del sito;
- **cookie analytics**, assimilati ai cookie tecnici laddove utilizzati direttamente dal gestore del sito per raccogliere informazioni, in forma aggregata, sul numero degli utenti e su come questi visitano il sito stesso;



Italian regulations

- **cookie di funzionalità**, che permettono all'utente la navigazione in funzione di una serie di criteri selezionati (ad esempio, la lingua, i prodotti selezionati per l'acquisto) al fine di migliorare il servizio reso allo stesso.
- Per l'installazione di tali cookie non è richiesto il preventivo consenso degli utenti, mentre resta fermo l'obbligo di dare l'informativa ai sensi dell'art. 13 del Codice, che il gestore del sito, qualora utilizzi soltanto tali dispositivi, potrà fornire con le modalità che ritiene più idonee.



Italian regulations

- b. **Cookie di profilazione.**
- I cookie di profilazione sono volti a creare profili relativi all'utente e vengono utilizzati al fine di inviare messaggi pubblicitari in linea con le preferenze manifestate dallo stesso nell'ambito della navigazione in rete. In ragione della particolare invasività che tali dispositivi possono avere nell'ambito della sfera privata degli utenti, la normativa europea e italiana prevede che l'utente debba essere adeguatamente informato sull'uso degli stessi ed esprimere così il proprio valido consenso.



Italian regulations

- 2. **Soggetti coinvolti: editori e "terze parti".**
- Un ulteriore elemento da considerare, ai fini della corretta definizione della materia in esame, è quello soggettivo. Occorre, cioè, tenere conto del differente soggetto che installa i cookie sul terminale dell'utente, a seconda che si tratti dello stesso gestore del sito che l'utente sta visitando (che può essere sinteticamente indicato come "editore") o di un sito diverso che installa cookie per il tramite del primo (c.d. "terze parti").



Italian regulations

- Nel momento in cui l'utente accede a un sito web, **deve essergli presentata una prima informativa "breve"**, contenuta in un banner a comparsa immediata sulla home page (o altra pagina tramite la quale l'utente può accedere al sito), **integrata da un'informativa "estesa"**, alla quale si accede attraverso un link cliccabile dall'utente.



Session

Introduction

Session tracking using cookies

The idea:



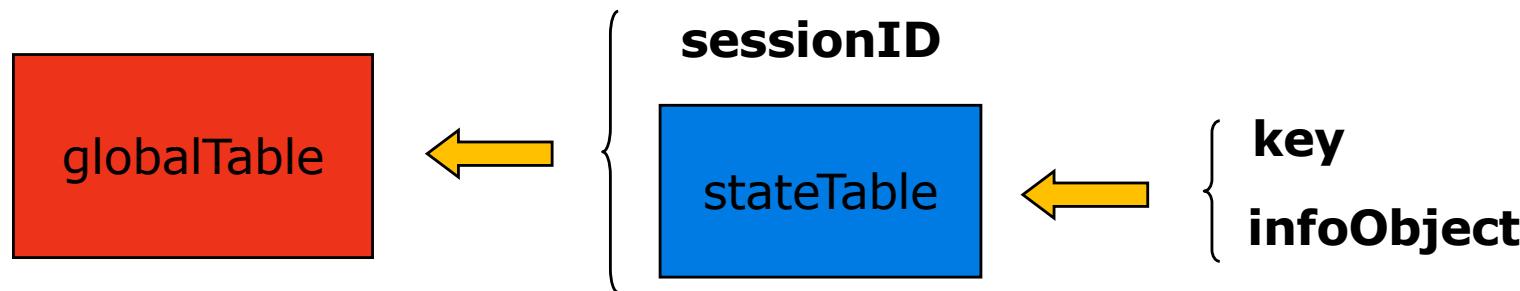
User ID	State (Map)	
	Key	Value
User 1	name	Object 1
	address	Object 2
	cart	Object 3
User 2	name	Object 1
	address	Object 2
	cart	Object 3
User 3	name	Object 1
	address	Object 2
	cart	Object 3

MEMORY



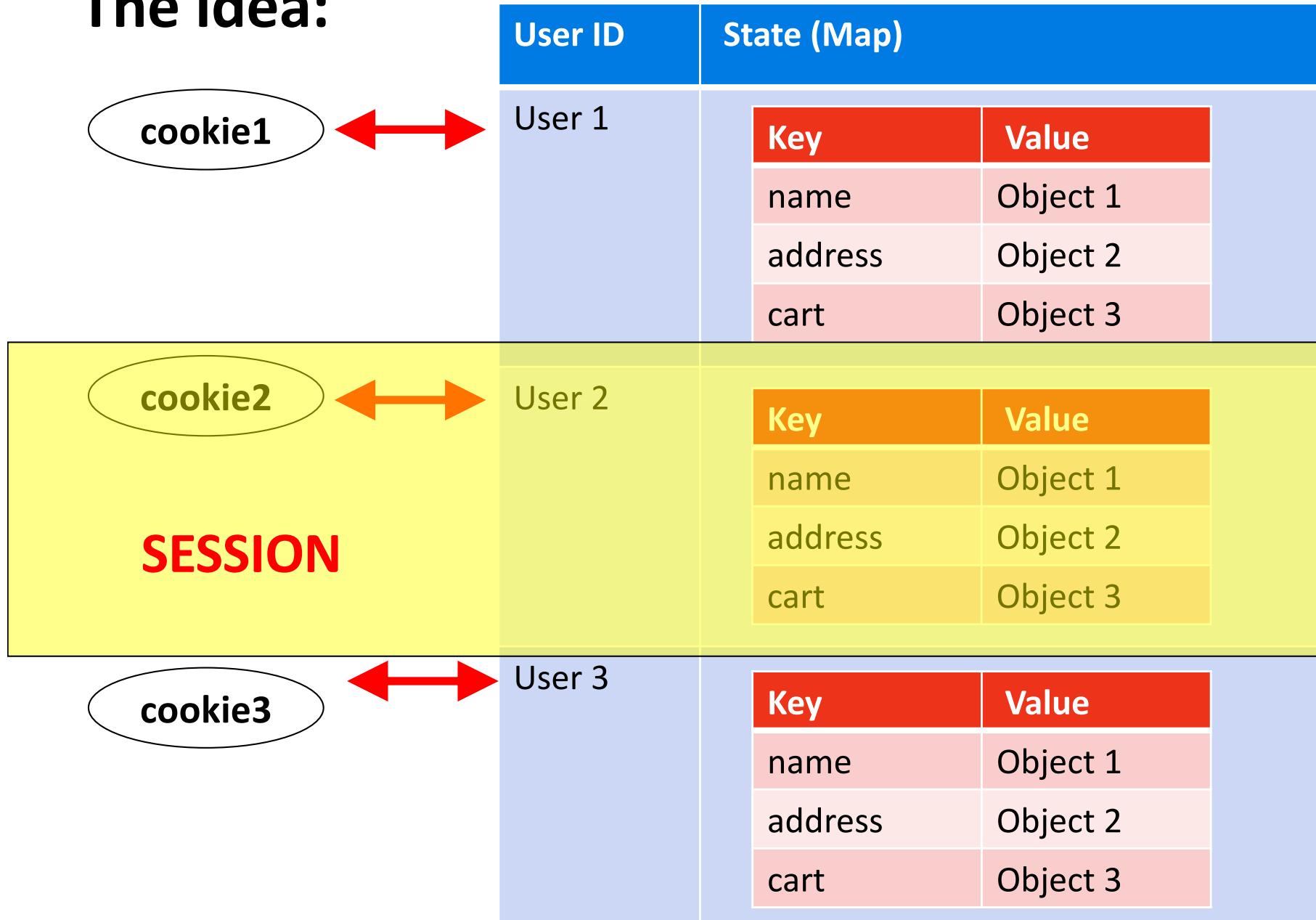
A possible implementation (pseudocode)

```
Hashtable globalTable = getGlobalTable();  
String sessionID = makeUniqueString();  
Hashtable stateTable = new Hashtable();  
globalTable.put(sessionID, stateTable);  
Cookie sessionCookie=  
    new Cookie("SessionID", sessionID);  
response.addCookie(sessionCookie);  
stateTable.put("key", infoObject);
```



Session tracking using cookies

The idea:



MEMORY



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

- 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)

The screenshot shows a web browser window. The address bar contains the URL: `esse3.unitn.it/auth/docente/RegistroDocente/ChangeStatus.do;jsessionid=D2C0C7DBB9070ACD9E94742084...`. A yellow oval highlights the part of the URL after `ChangeStatus.do;`, specifically `jsessionid=D2C0C7DBB9070ACD9E94742084...`, which represents the session identifier.

Below the browser window, the University of Trento logo is visible, along with the text "UNIVERSITÀ DI TRENTO". To the right, the "ESSE3" logo is shown with the tagline "servizi online per la didattica". Below the logo is a graphic featuring a laptop and various colorful icons representing different online services.

The main content area of the page shows a breadcrumb navigation: [Home](#) » [Elenco Registri](#) » [Dati Registro](#). The title "Dettaglio Registro" is displayed prominently. At the bottom of the page, a blue bar contains the text "Attività: Introduzione alla Programmazione per il web [145325]".



Disallowing cookies

← → ⌂ Chrome | chrome://settings/content/cookies

Settings

You and Google

Auto-fill

Privacy and security

Appearance

Search engine

Default browser

On start-up

Advanced ▾

Extensions 

About Chrome

← Cookies and site data

Allow sites to save and read cookie data (recommended)

Clear cookies and site data when you quit Chrome

Block third-party cookies
When on, sites can't use your browsing activity across different sites to personalise ads. Some sites may not work properly.

See all cookies and site data 

Block 

localhost 

Current incognito session



Output

Chrome

Let's meet...

Hi! What is your name?

localhost:8084/WebAppWithCoo...

Hi Marco, nice to meet you!

Hi Marco, welcome back! (2)

localhost:8084/WebAppWithCoo...

All cookies have been deleted

Go to the [initial page](#).

Safari

localhost:8084/WebAppWithCoo...

Hi! What is your name?

localhost:8084/WebAppWithCoo...

Hi Pietro, nice to meet you!

Hi Pietro, welcome back! (5)

localhost:8084/WebAppWithCoo...

Sorry, we do not know each other...
Please introduce yourself.

What is your name?