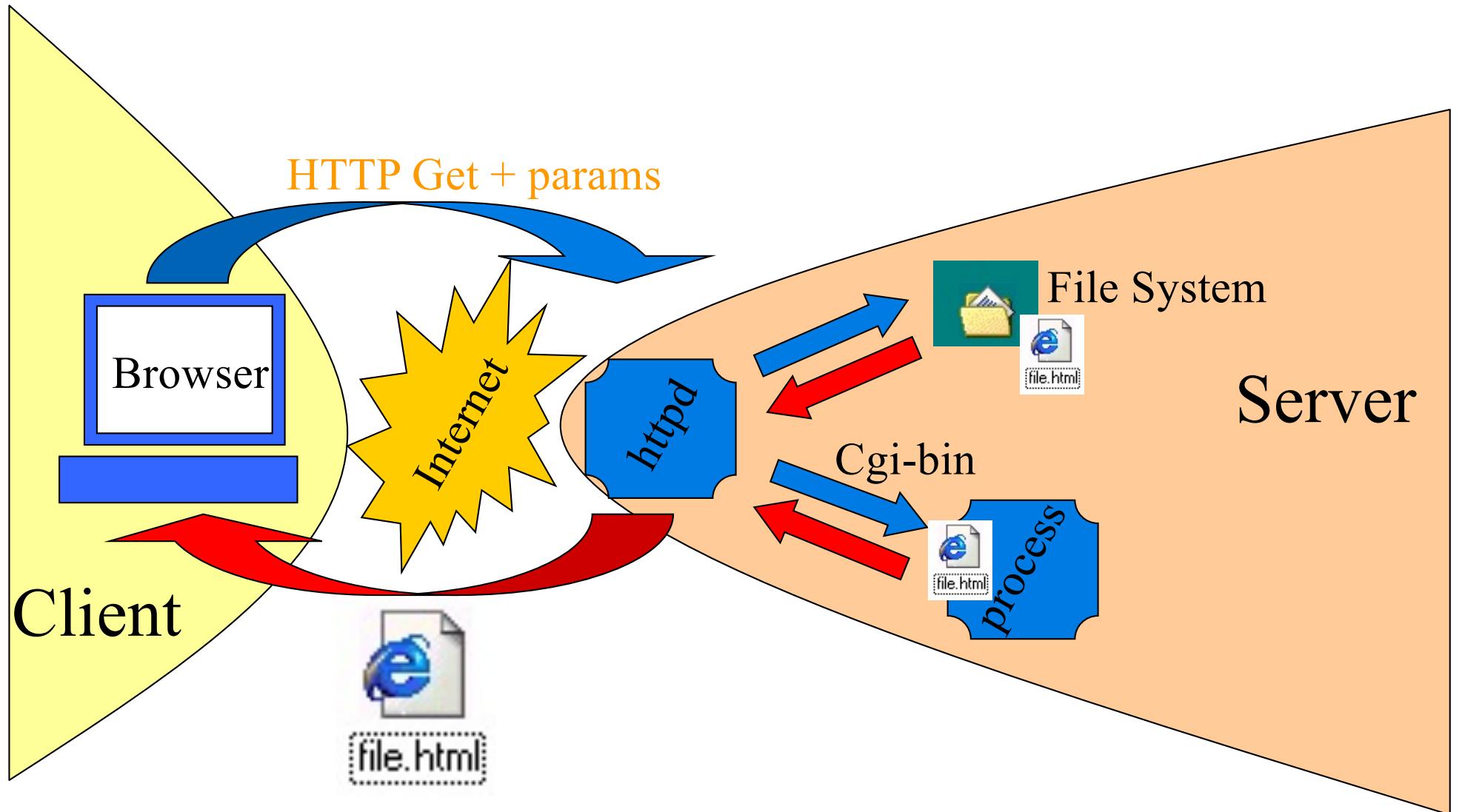


Dynamic content: programming the web servers (recap)

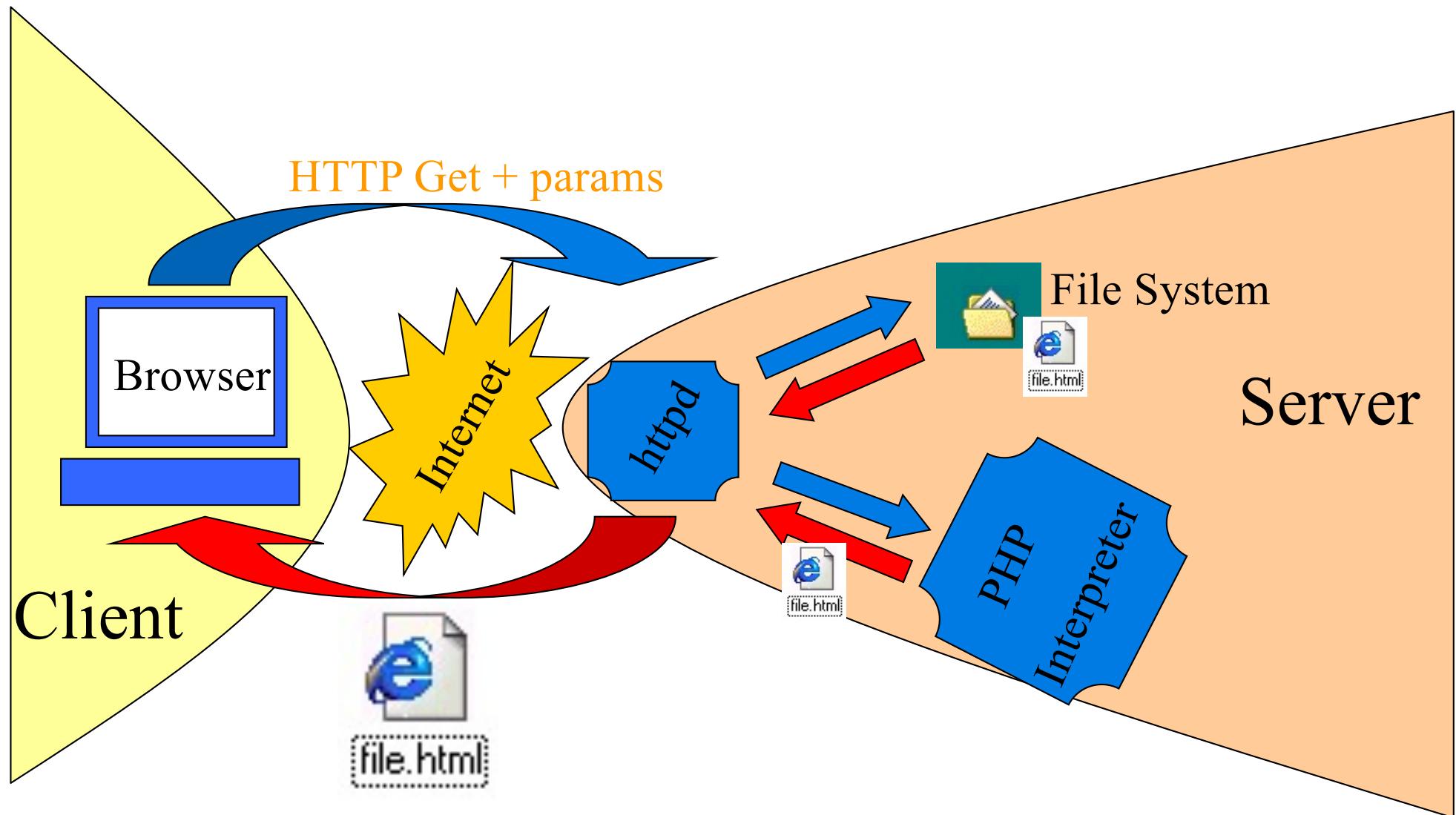
The original web architecture: dynamic pages



Evolution 1: dynamically create (interlinked) documents



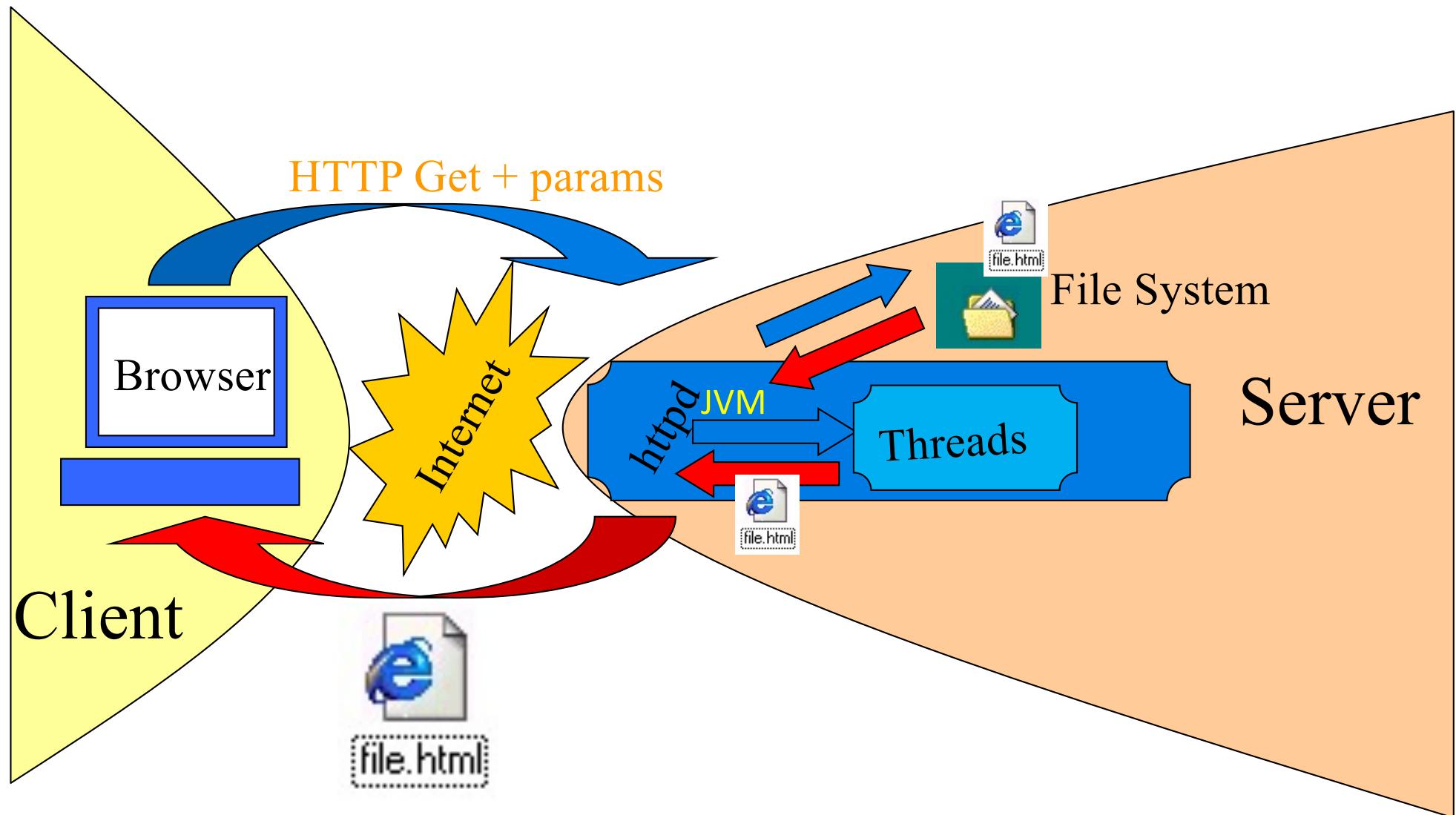
The original web architecture: dynamic pages



Evolution 1: dynamically create (interlinked) documents



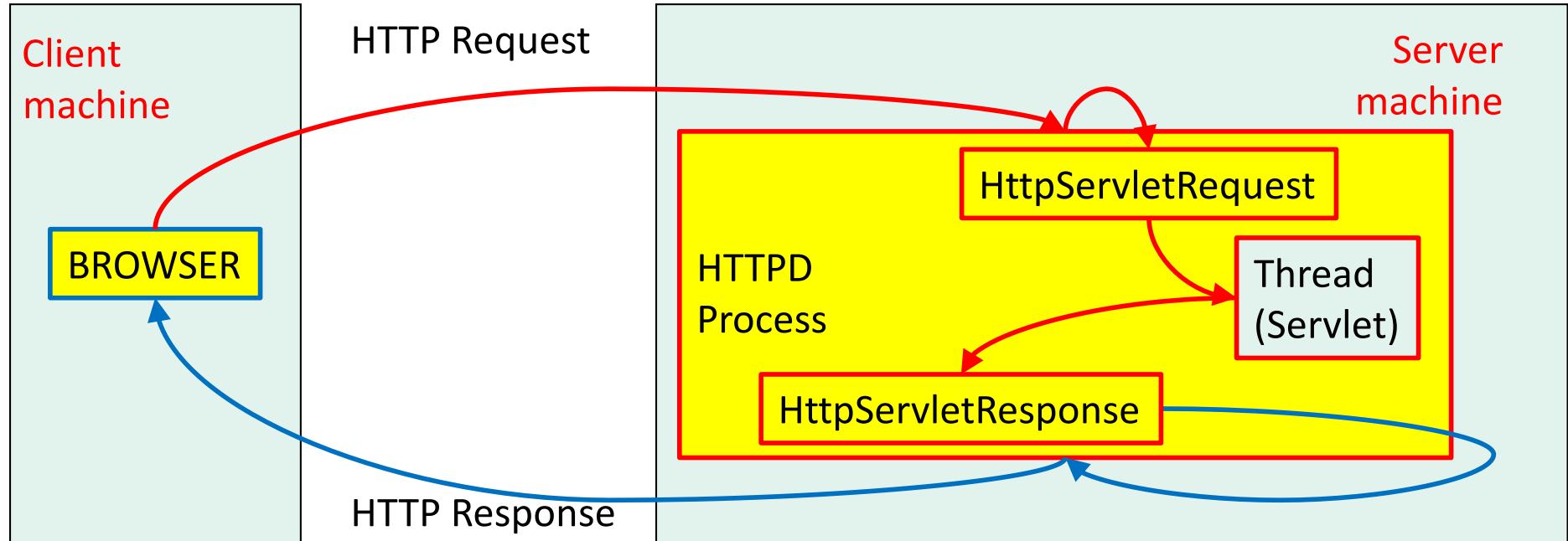
The original web architecture: dynamic pages



Evolution 1: dynamically create (interlinked) documents

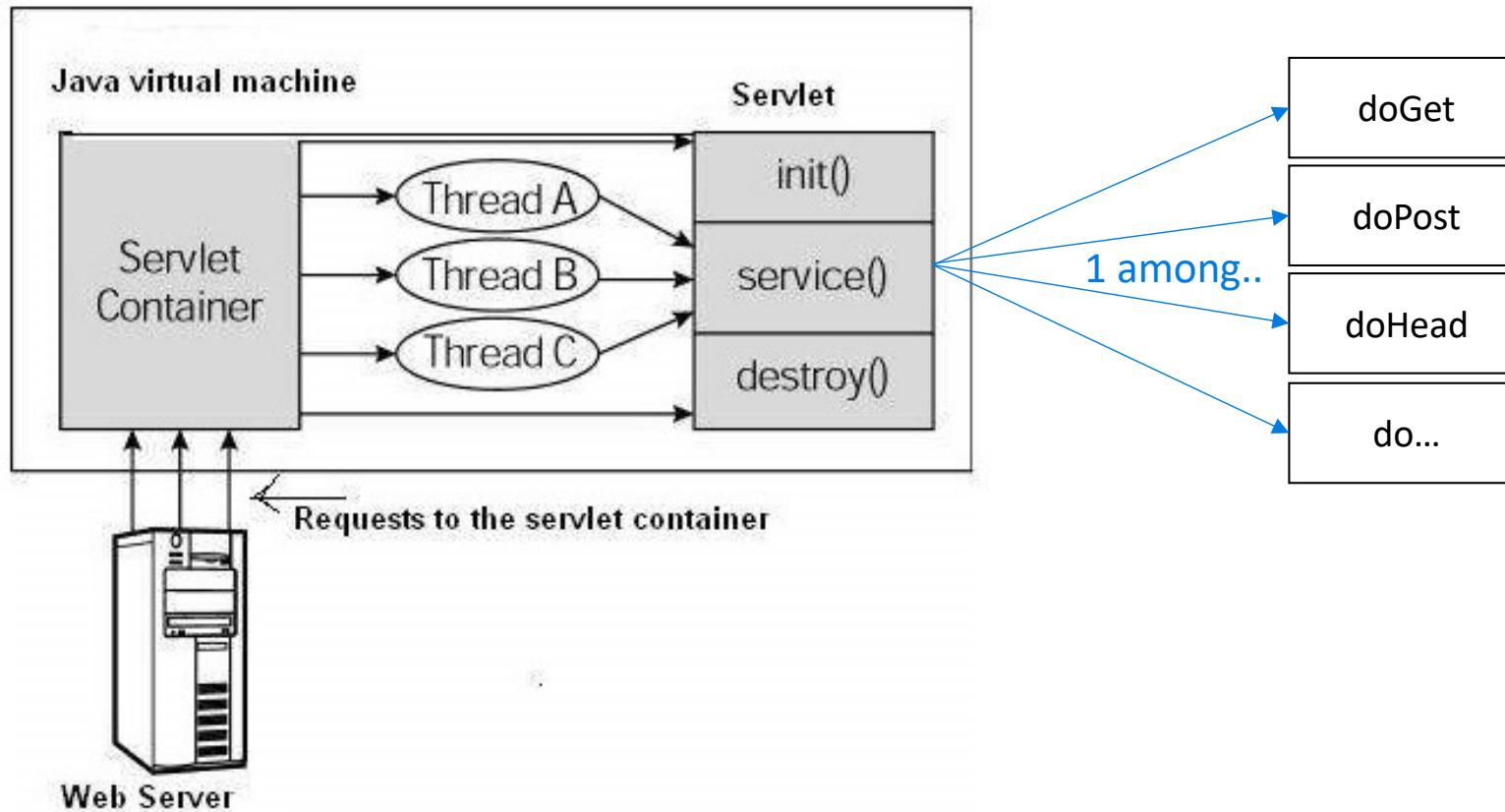


Getting info about the request



- Apache **Tomcat** is a **Java servlet container**, and is run on a **Java Virtual Machine**, or **JVM**.
- **Tomcat** utilizes the Java servlet specification to execute servlets generated by requests, often with the help of JSP pages, allowing dynamic content to be generated much more efficiently than with a CGI script.

Servlet lifecycle

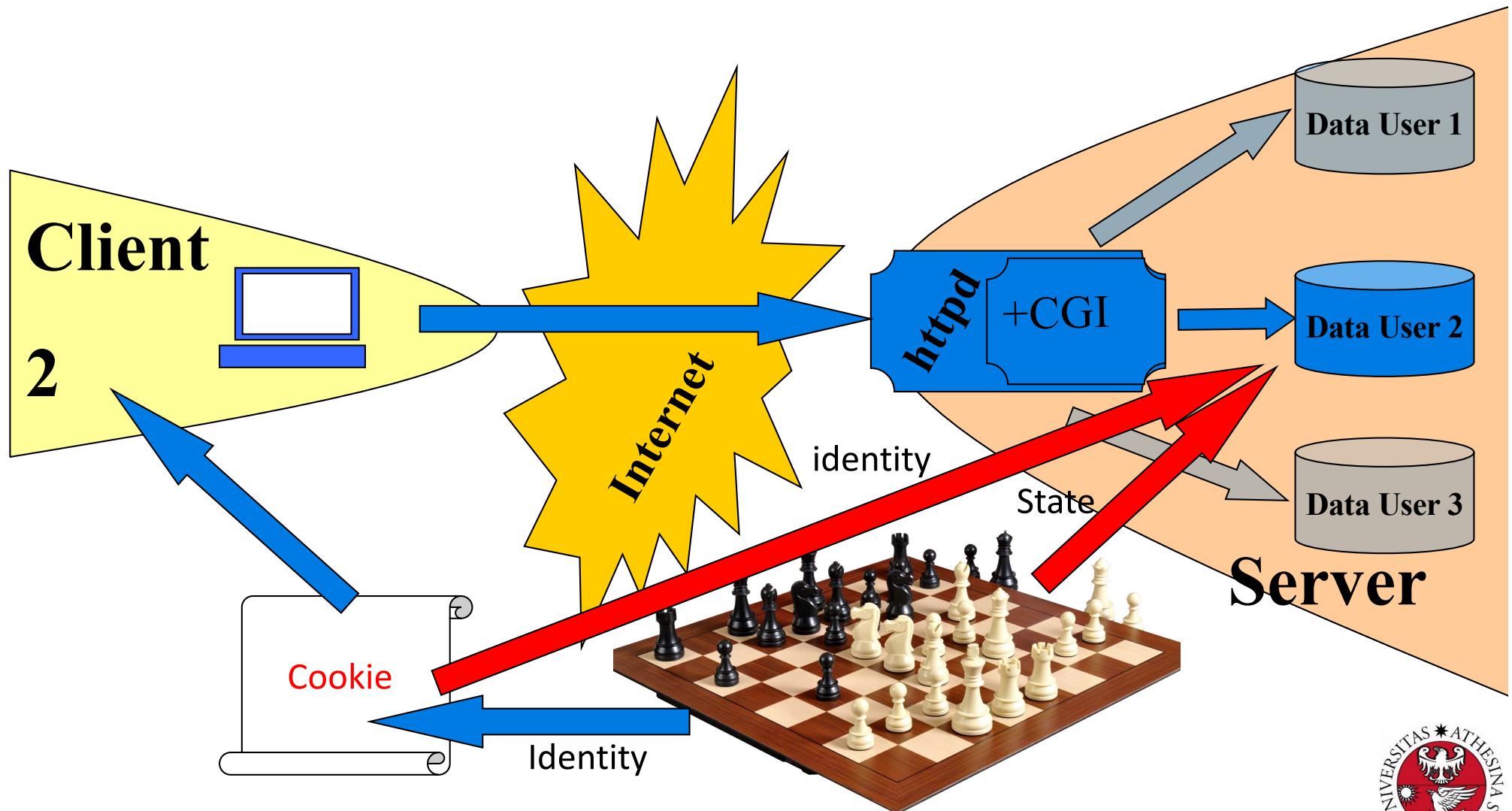


See <https://www.tutorialspoint.com/servlets/servlets-life-cycle.htm>

The state problem

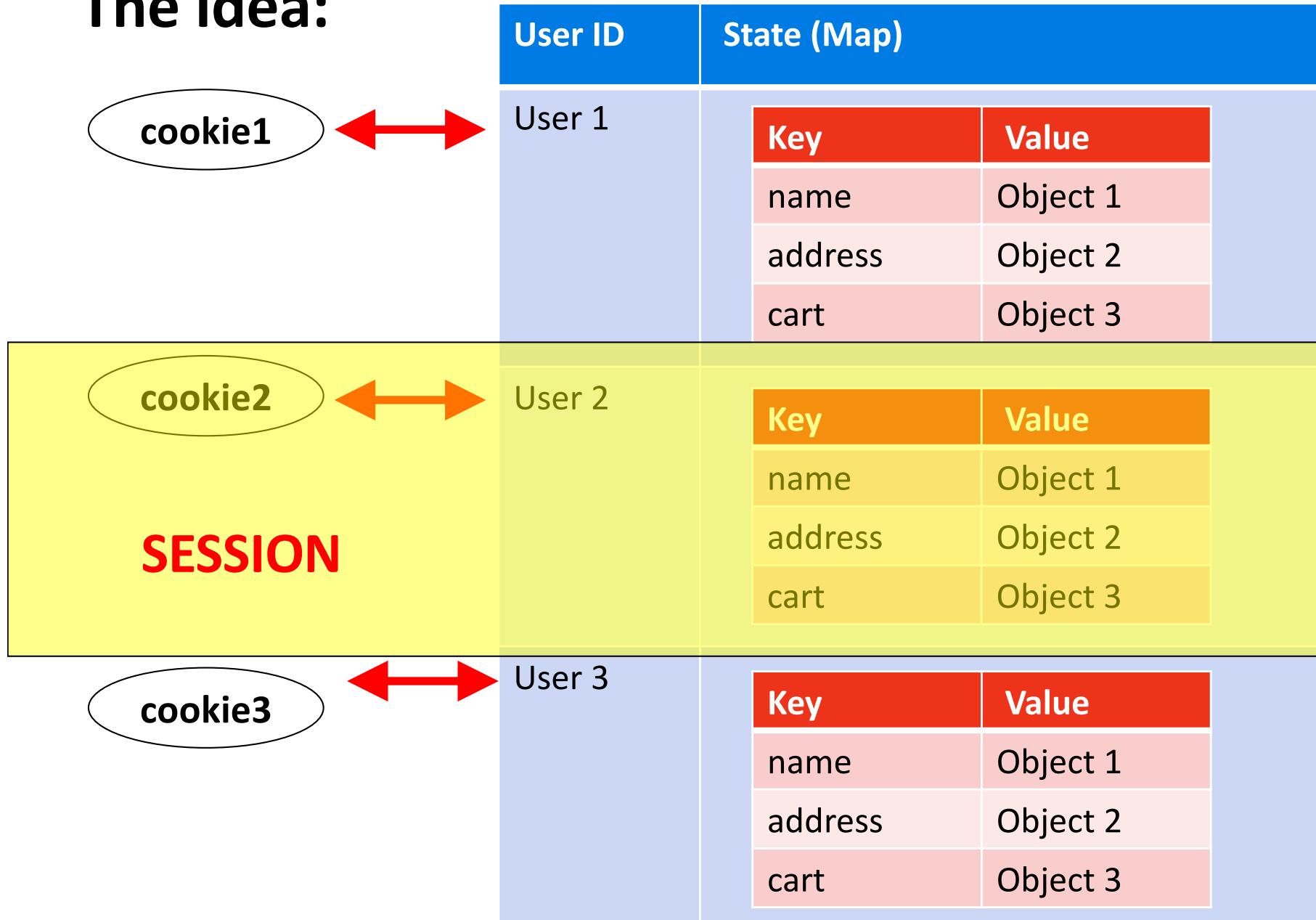
The state problem: solution

Cookies



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 `jsessionid` part of the URL. Below the browser, the University of Trento logo and the ESSE3 online services logo are visible. The page content shows a navigation menu with links like 'Elenco Registri' and 'Dati Registro', and a section titled 'Dettaglio Registro'. At the bottom, there is a footer with the university logo and the text 'Attività: Introduzione alla Programmazione per il web [145325]'.

← → ⌂ 🔒 esse3.unitn.it/auth/docente/RegistroDocente/ChangeStatus.do;jsessionid=D2C0C7DBB9070ACD9E94742084...

UNIVERSITÀ DI TRENTO

ESSE³
servizi *online*
per la didattica

Elenco Registri » Dati Registro

Dettaglio Registro

Attività: Introduzione alla Programmazione per il web [145325]

UNIVERSITAS * ATHENISNA
STUDIORUM UNITN

URL-Rewriting

```
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
```

Servlet Deployment

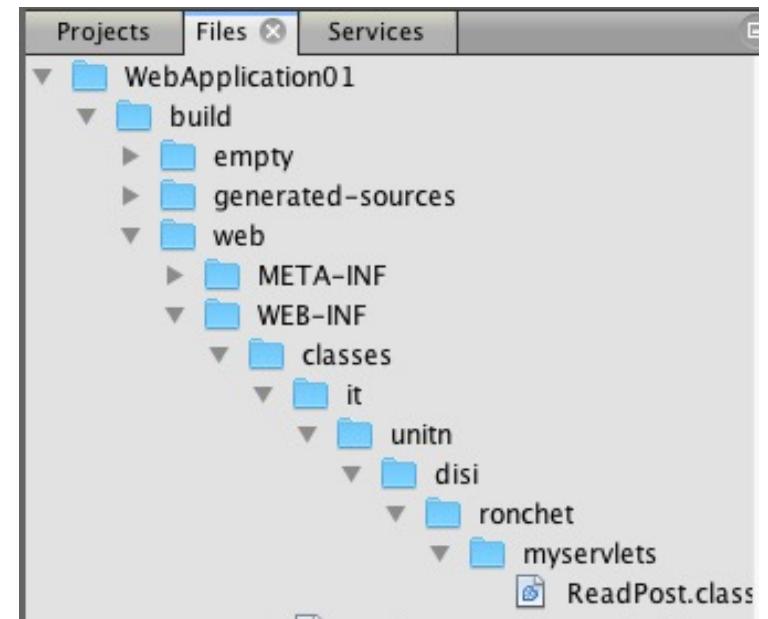
- By default, a servlet application is located at the path

<Tomcat-installationdirectory>/webapps/ROOT

and the class file would reside in

<Tomcat-installationdirectory>/webapps/ROOT/WEB-INF/classes.

If you have a fully qualified class name
of **com.myorg.MyServlet**, then this servlet class must
be located in
WEB-INF/classes/com/myorg/MyServlet.class.



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>

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.

JSP



Basics

Tutorial

<https://www.tutorialspoint.com/jsp/index.htm>

JSP Technology

A technology somehow similar to PHP or ASP,
ASP.net, but Java-based.

Dual to Servlets

Has been the basis for JSP-CustomTags

Has been the basis for JSF



ASP.Net Core

ASP (Active Server Pages) was a Microsoft, Windows-only technology based on the same idea.

Its evolution was ASP.Net first, and now ASP.Net Core.

ASP.Net Core is based on .Net Core, a portable implementation of the previously Windows-only DLLs.

<https://docs.microsoft.com/it-it/dotnet/core/>

A taste of servlet programming-2

```
import java.util.Calendar;  
  
public class SimpleServlet extends HttpServlet {  
  
    public void doGet (HttpServletRequest request,  
                      HttpServletResponse response)  
        throws ServletException, IOException {  
  
        PrintWriter out=response.getWriter();  
  
        response.setContentType("text/html");  
  
        out.println("<HTML><BODY>");  
        out.println(Calendar.get(Calendar.HOUR_OF_DAY));  
  
        out.println("</BODY></HTML>");  
  
        out.close();  
    }  
}
```

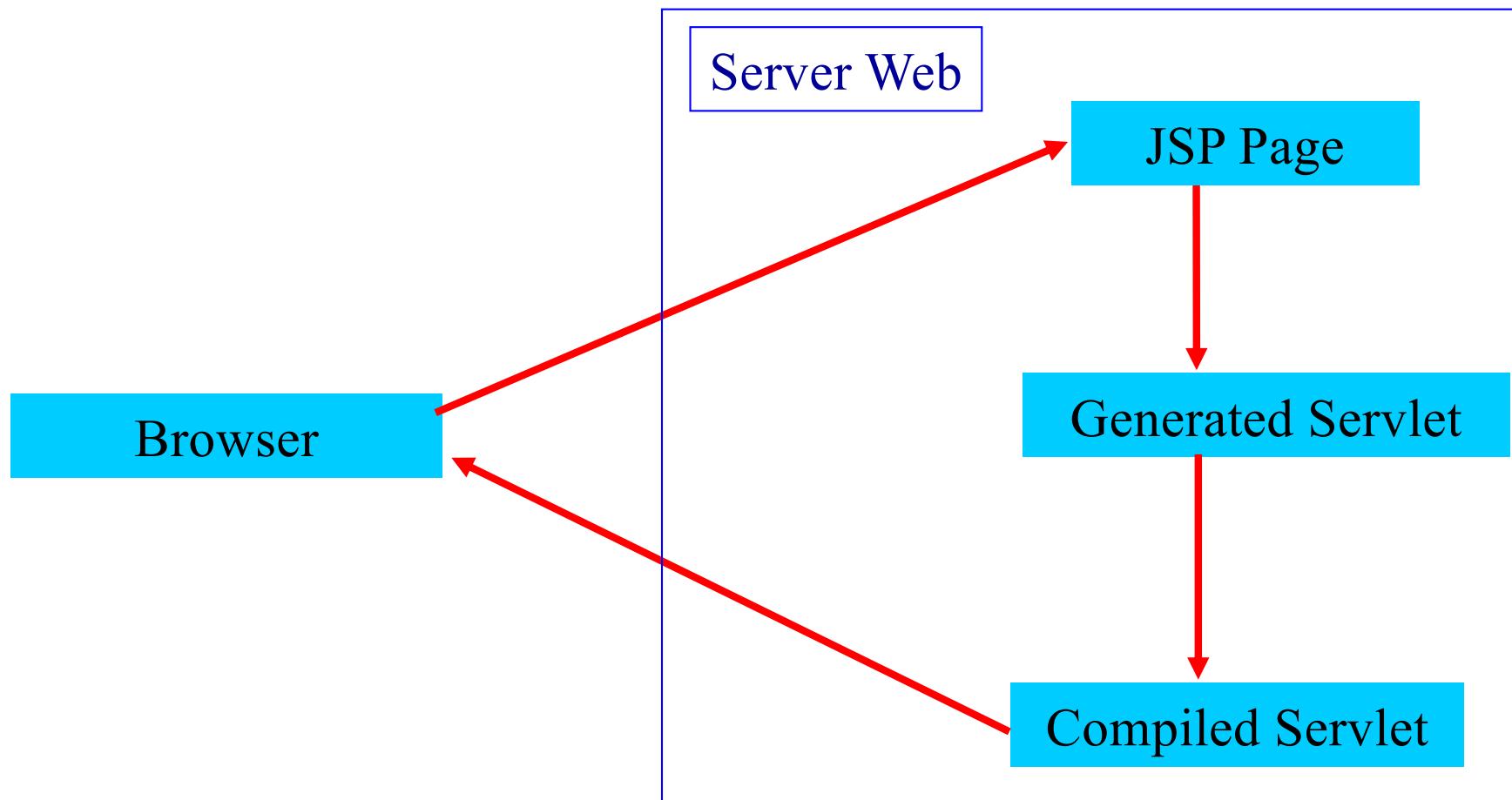


Simple.jsp

```
<%@ page import=java.util.* %>
<html>
<body>
    <% out.println(Calendar.get(Calendar.HOUR_OF_DAY)); %>
</body>
</html>
```



JSP Lifecycle



JSP nuts and bolts

Syntactic elements:

<%@ directives %>
<%! declarations %>
<% scriptlets %>
<%= expressions %>
<jsp:actions/>
<%-- Comment --%>

Implicit Objects:

- request
- response
- pageContext
- session
- application
- out
- config
- page



JSP nuts and bolts

Syntactic elements:

<%@ directives %> → Interaction with the CONTAINER

<%! declarations %> → In the initialization of the JSP

<% scriptlets %> → In the service method

<%= expression %> → (Syntactic sugar)

same as scriptlet: <% out.println(expression %)>

<jsp:actions/>



Scriptlets

A **scriptlet** is a block of Java code **executed during the request-processing time**.

In Tomcat all the scriptlets gets put into the [service\(\)](#) method of the servlet. They are therefore processed for every request that the servlet receives.



Scriptlet

Examples :

```
<% z=z+1; %>
```

```
<%
    // Get the Employee's Name from the request
    out.println("<b>Employee: </b>" +
    request.getParameter("employee"));

    // Get the Employee's Title from the request
    out.println("<br><b>Title: </b>" +
    request.getParameter("title"));

%>
```



Declarations

- A **declaration** is a block of Java code used to:
- **define class-wide variables and methods in the generated servlet.**
- They are **initialized** when the JSP page is initialized.
- `<%! DECLARATION %>`
- Examples:
- `<%! String nome="pippo"; %>`
- `<%! public String getName() {return nome;} %>`



Directives

A **directive** is used as a message mechanism to:

pass information from the JSP code to the container

Main directives:

`page`

`include` (for including other **STATIC** resources at compilation time)



Directives

```
<%@ DIRECTIVE{attributo=valore} %>
```

main attributes:

```
<%@ page language=java session=true %>
<%@ page import=java.awt.* ,java.util.* %>
<%@ page errorPage=URL %>
<%@ page isErrorPage=true %>
```



Standard actions

- Standard action are tags that affect the runtime behavior of the JSP and the response sent back to the client.
- <jsp:include page="URL" />
- For including STATIC or DYNAMIC resources at request time
- <jsp:forward page="URL" />



Predefined Objects

out	Writer
request	HttpServletRequest
response	HttpServletResponse
session	HttpSession
page	this nel Servlet
application	servlet.getServletContext area condivisa tra i servlet
config	ServletConfig
exception	solo nella errorPage
pageContext	sorgente degli oggetti, raramente usato



request

```
<%@ page errorPage="errorpage.jsp" %>
<html>
    <head>
        <title>UseRequest</title>
    </head>
    <body>
        <%
            // Get the User's Name from the request
            out.println("<b>Hello: " + request.getParameter("user") + "</b>");
        %>
    </body>
</html>
```



<%@include@%> or <jsp:include> ?

- When should I use a JSP <%@include@%> directive, and when should I use a <jsp:include> action?
- A JSP <%@include@%> directive (for example, <%@ include file="myfile.jsp" @%>) includes literal text "as is" in the JSP page and is not intended for use with content that changes at runtime. The include occurs only when the servlet implementing the JSP page is being built and compiled.
- The <jsp:include> action allows you to include either static or dynamic content in the JSP page. Static pages are included just as if the <%@include@%> directive had been used. Dynamic included files, though, act on the given request and return results that are included in the JSP page. The include occurs each time the JSP page is served.



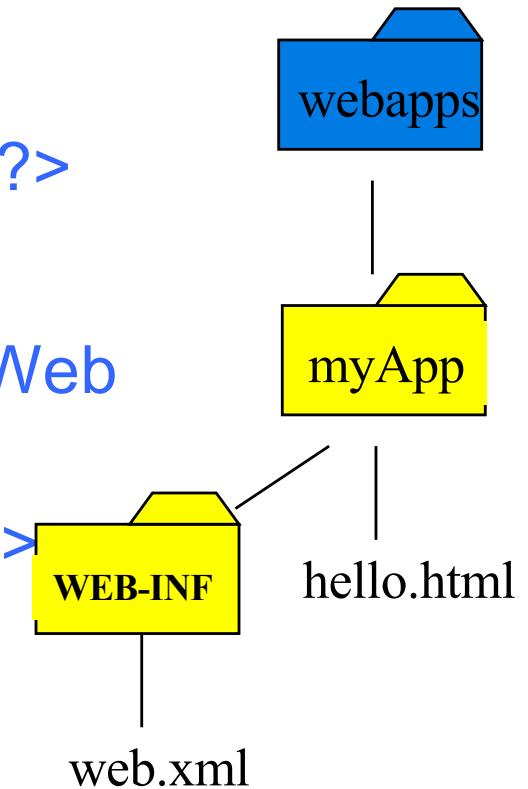


WebApps

(Tomcat configuration)

Static pages

- A web.xml file **MUST** be provided:
- <?xml version="1.0" encoding="ISO-8859-1"?>
- <!DOCTYPE web-app
 - PUBLIC "-//Sun Microsystems, Inc//DTD Web Application 2.3//EN"
 - "http://java.sun.com/dtd/web-app_2_3.dtd">
- <web-app>
- </web-app>



JSP pages

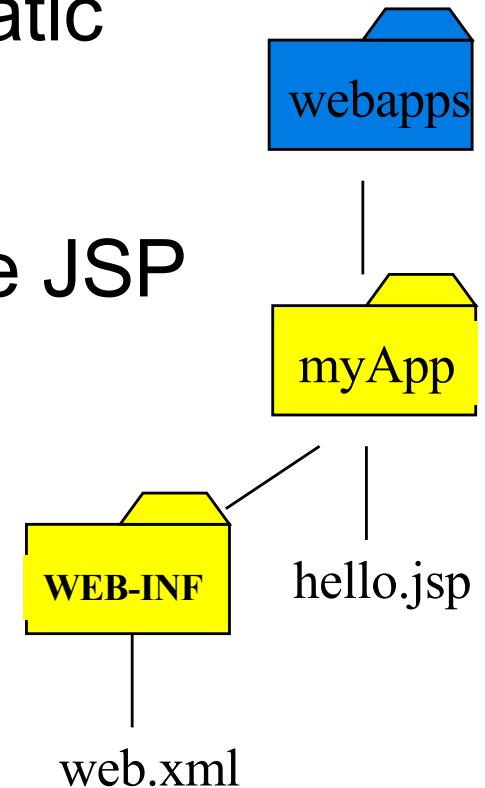
To let Tomcat serve JSP pages, we follow the same procedure that we use for static pages.

In the `myApp` folder we can deposit the JSP files.

On our Tomcat server, the URL for the `hello.jsp` file becomes:

`http://machine/port/myApp/hello.jsp`

The `WEB-INF` directory can be empty.



JSP in action - 1

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page import="java.util.Date"%>
<%@page language="java" session="true" %>

<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type"
              content="text/html; charset=UTF-8">
        <title>Session Test JSP</title>
    </head>
    <body>
        <%! Integer accessCount; %>
        <%
            accessCount=(Integer)session.getAttribute("accessCount");
            if (accessCount == null) {
                accessCount = 0; // autobox int to Integer
            } else {
                accessCount = new Integer(accessCount + 1);
            }
            session.setAttribute("accessCount", accessCount);
        %>
    </body>
</html>
```



JSP in action - 2

```
Session is new? <% out.println(session.isNew()); %>
    <h2>You accessed this site " <%= accessCount %>
        times in this session.</h2>
    <ul><li>Your session ID is " <%= session.getId() %></li>
        <li>Session creation time is
            <%= new Date(session.getCreationTime()) %> </li>
        <li>Session last access time is <%
            new Date(session.getLastAccessedTime()) %> </li>
        <li>Session max inactive interval is <%
            session.getMaxInactiveInterval() %> seconds</li>
    </ul>

    <p><a href='<%= request.getRequestURI() %>'>Refresh</a>
    <p><a href='
        <%= response.encodeURL(request.getRequestURI()) %>'>
            Refresh with URL rewriting</a>
    <form method="GET" action="endSession.jsp">
        <input type="submit" value="End Session">
    </form>
</body>
</html>
```



Where is the generated code?

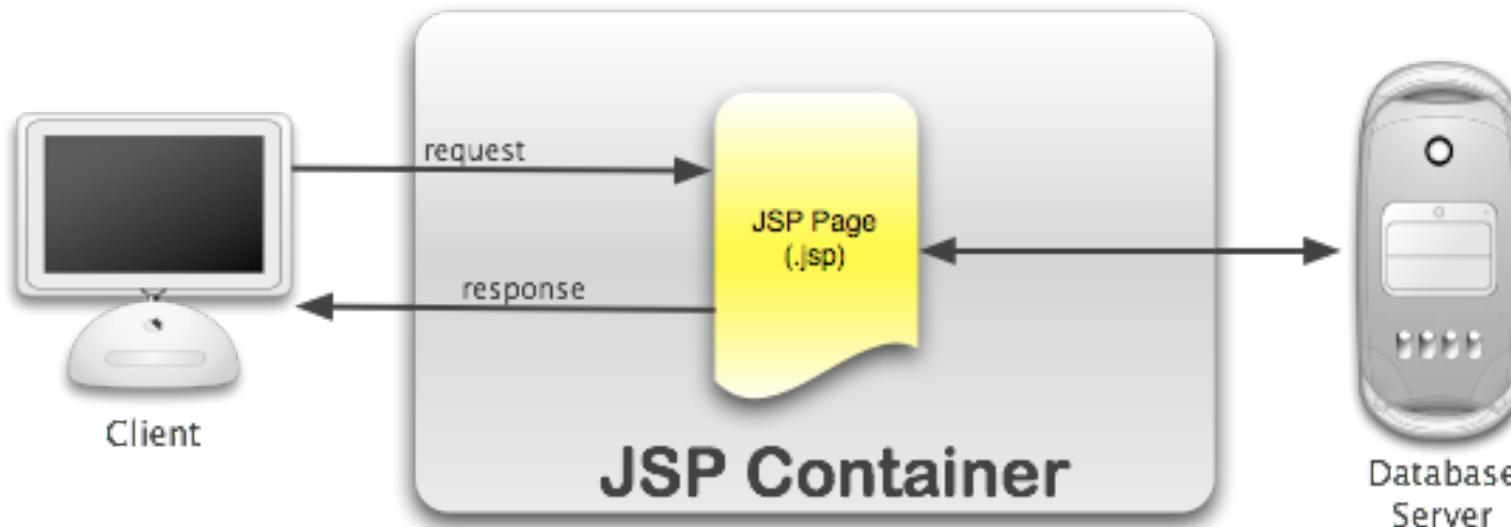
`/Users/ronchet/Library/Application Support/NetBeans/8.2/
apache-tomcat-8.0.27.0_base/work/Catalina/localhost/
WebAppJSPwithSession/org/apache/jspDemoSession.class`

See generatedServlet.java on the web site



JSP usage: MVC pattern

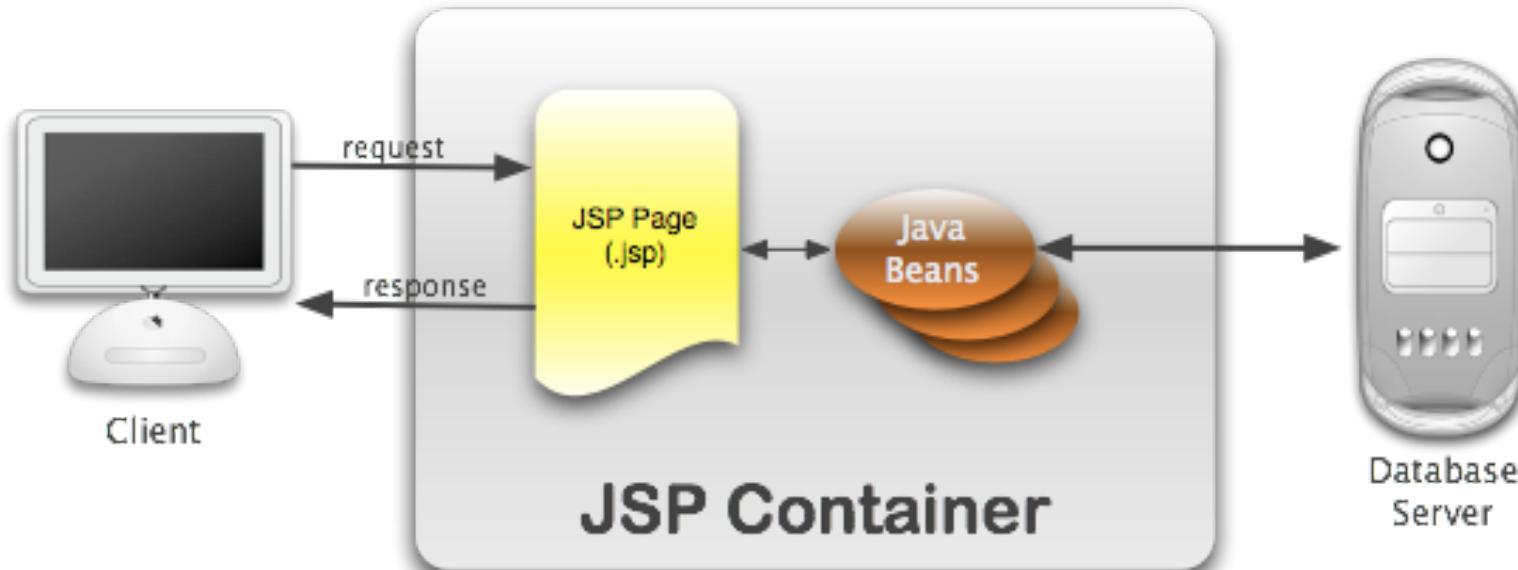
The simple (but wrong) approach



by Bear Bibeault, March 2006

control, data management and presentation
in the same page!

A better solution



by Bear Bibeault, March 2006

control logic is delegated to Java classes

What is a Java bean?

A **bean** is a Java class that:

- Provides a public no-argument constructor
- Implements `java.io.Serializable`
- Follows JavaBeans design patterns
 - Has Set/get methods for properties
 - (Has Add/remove methods for events)
- Is thread safe/security conscious
 - Can run in an applet, application, servlet, ...

Example:

```
public class SimpleBean implements Serializable {  
    private int counter;  
    SimpleBean() {counter=0;}  
    int getCounter() {return counter;}  
    void setCounter(int c) {counter=c;}  
}
```



Standard actions involving beans

```
<jsp:useBean id="name" class="fully_qualified_pathname"  
scope="{page|request|session|application}" />
```

```
<jsp:setProperty name="nome" property="value" />  
<jsp:getProperty name="nome" property="value" />
```

See: https://www.tutorialspoint.com/jsp/jsp_java_beans.htm



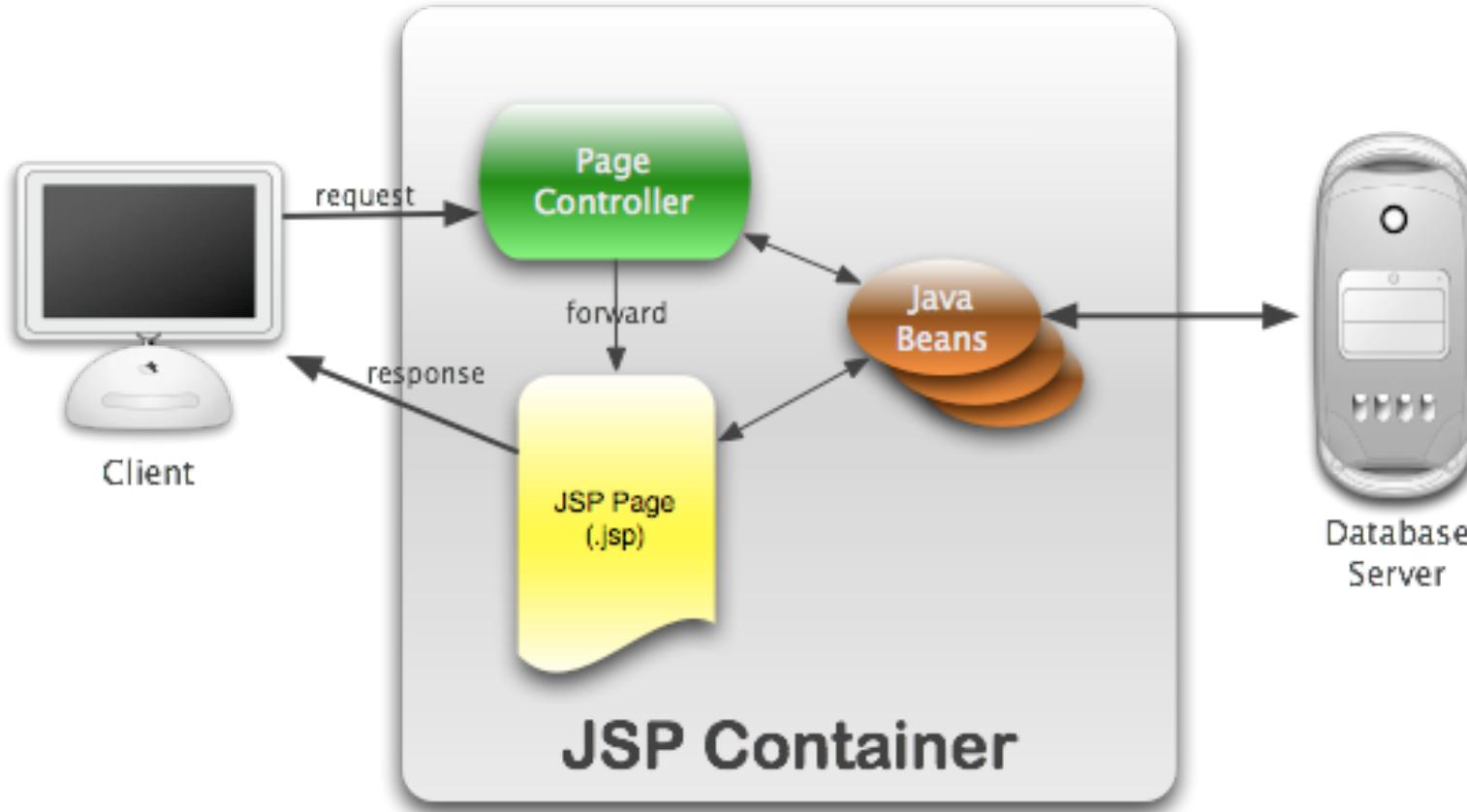
Example

```
<html> <head>
<title>get and set properties Example</title>
</head> <body>
<jsp:useBean id = "students" class = "com.tutorialspoint.StudentsBean">
    <jsp:setProperty name = "students" property = "firstName" value = "Zara"/>
    <jsp:setProperty name = "students" property = "lastName" value = "Ali"/>
    <jsp:setProperty name = "students" property = "age" value = "10"/>
</jsp:useBean>
<p>Student First Name: <jsp:getProperty name = "students" property = "firstName"/> </p>
<p>Student Last Name: <jsp:getProperty name = "students" property = "lastName"/> </p>
<p>Student Age: <jsp:getProperty name = "students" property = "age"/> </p>
</body> </html>
```

from: https://www.tutorialspoint.com/jsp/jsp_java_beans.htm



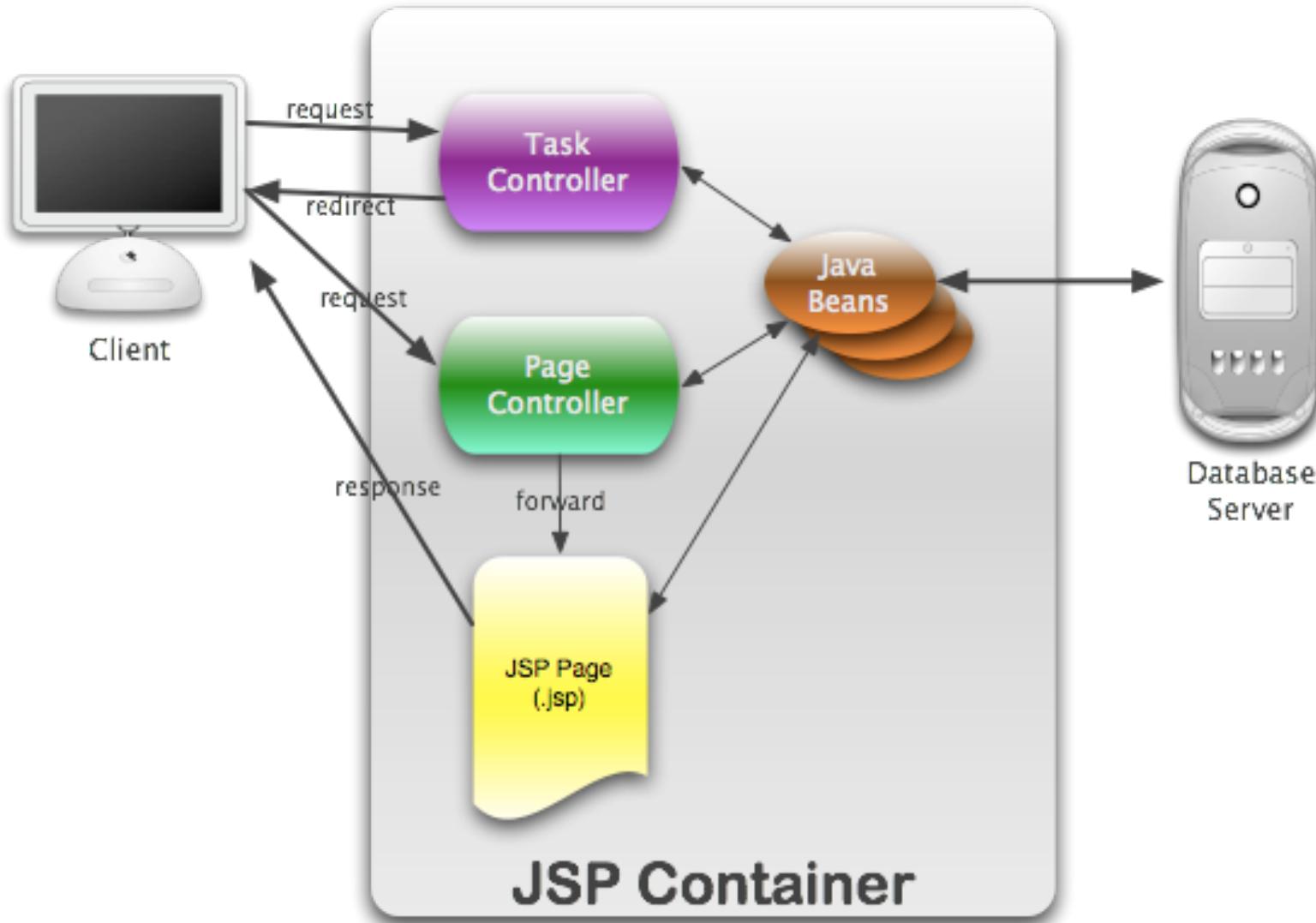
JSP used only for presentation



by Bear Bibeault, March 2006

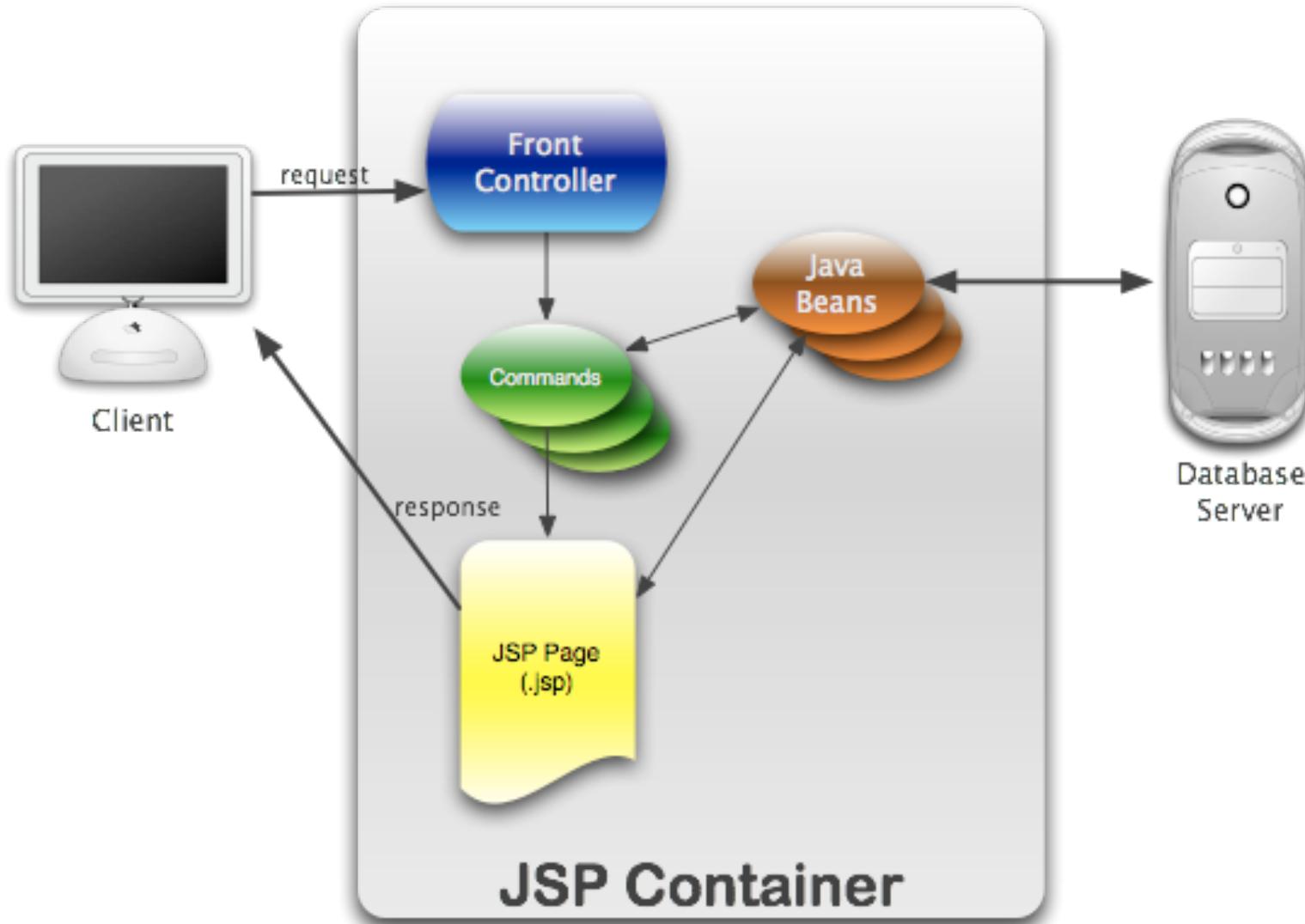
JSP used only for presentation

Let's separate post and get



by Bear Bibeault, March 2006

Front controller pattern



by Bear Bibeault, March 2006



Examples

from:

<https://javaranch.com/journal/200603/frontman.html>

The Front Controller employed by the **Struts** package typically uses a servlet mapping of ***.do** where whatever appears as the prefix of the ".do" is used to lookup the actual class path of the Command (called "actions" in Struts) in an internal configuration map.

Another example, the pattern that I usually use, is to employ a servlet mapping such as **/command/*** where the prefix "command" triggers the front controller, and the rest of the path info is used to lookup the Command class in an internal map.

It would be typical to see URLs along the lines of:

`http://some.server.com/webapp/command/deleteItem`

`http://some.server.com/webapp/command/insertItem`

`http://some.server.com/webapp/command/doSomethingWonderful`



Exercise

Transform the servlets-based web app
of Exercise 2
into a JSP-based web app