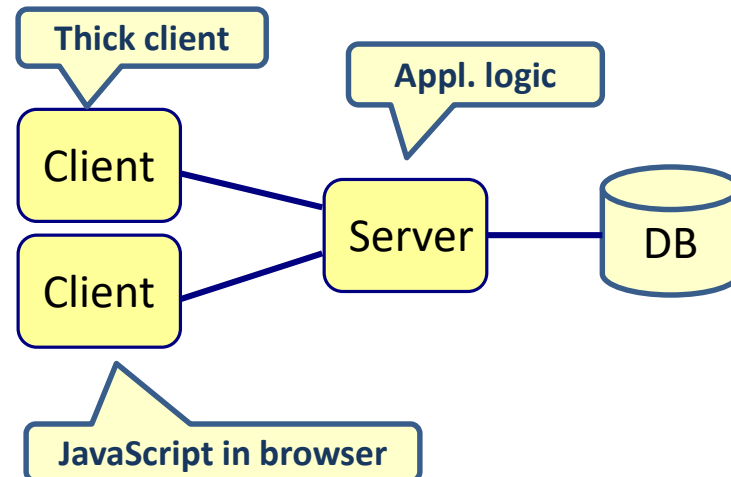
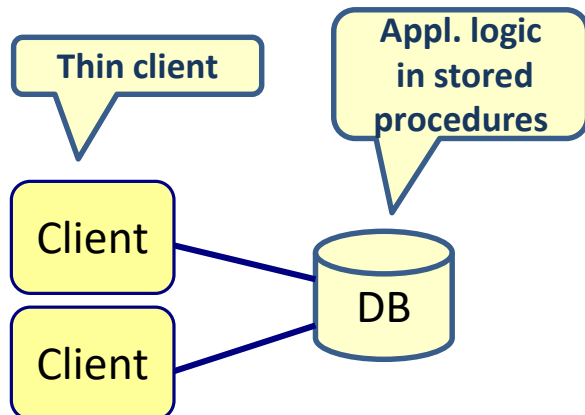
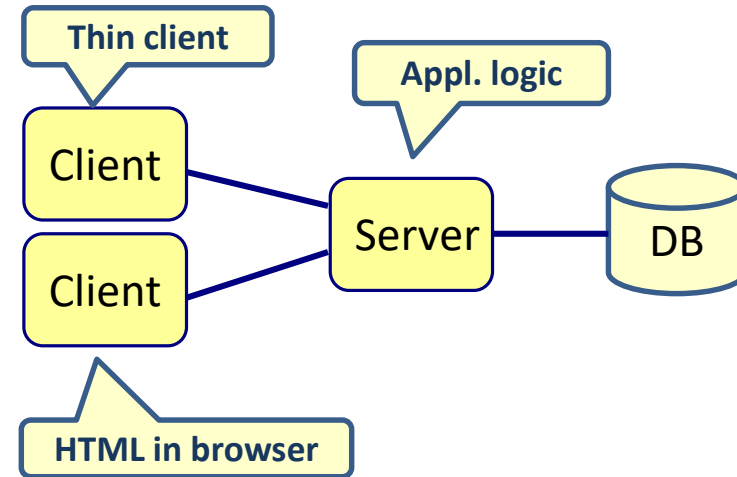
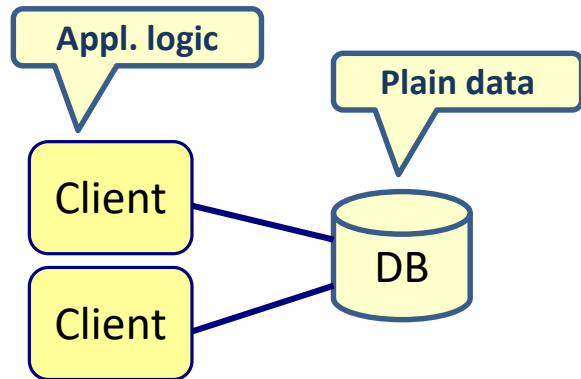


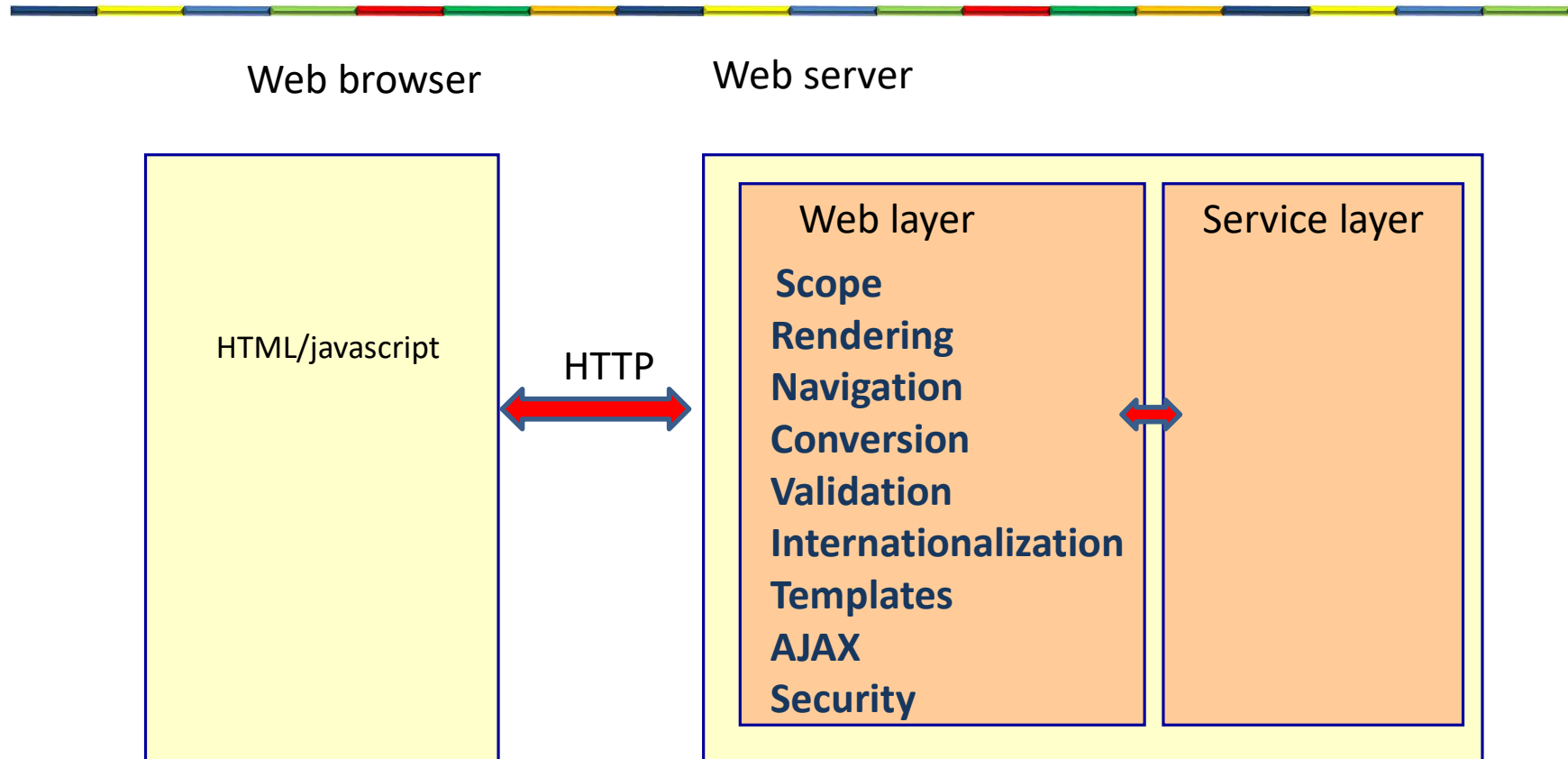
# LESSON 1 WEB DEVELOPMENT

# **CLIENT-SIDE VS. SERVER-SIDE WEB FRAMEWORK**

# Thin client vs. thick client

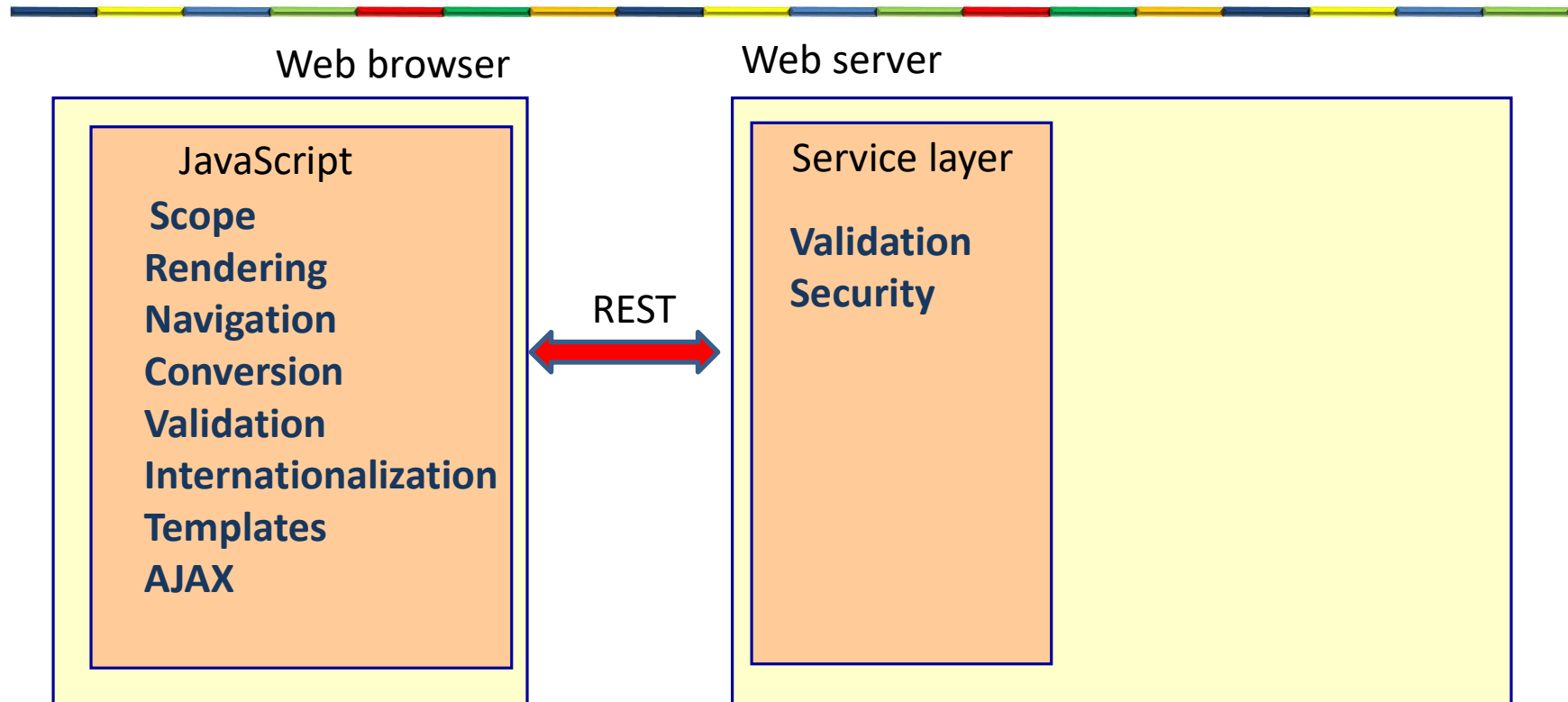


# Server side web framework



- Every action is executed on the server

# Client side web framework



- You only go to the server if you need to.

# Server centric versus client centric



- Remove a stock from the watch list:
  - Server centric: send a request to the server and execute on the server
  - Client centric: execute within the browser

# Server centric versus client centric

---

- Server centric

- Servlets/JSP
- JSF
- Spring MVC

- Client centric

- Angular
- React
- Vue

# Client centric

---

- Advantages

- Javascript runs in the browser. It only goes to the server when it needs data.
  - Less network traffic between client and server, which makes the application more scalable
    - Less burden on the server
  - Faster response times in the client because we don't need to go to the server all the time
  - Separation between front-end and back-end allows different programmers/teams to work on the front-end or back-end, independent from each other
  - Separation between front-end and back-end support using different front-end channels

- Disadvantage

- The frameworks and techniques change very fast

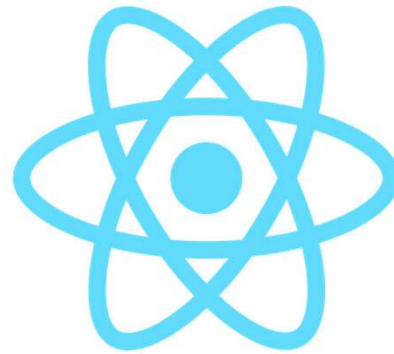


# Client side web frameworks

---



Angular



React

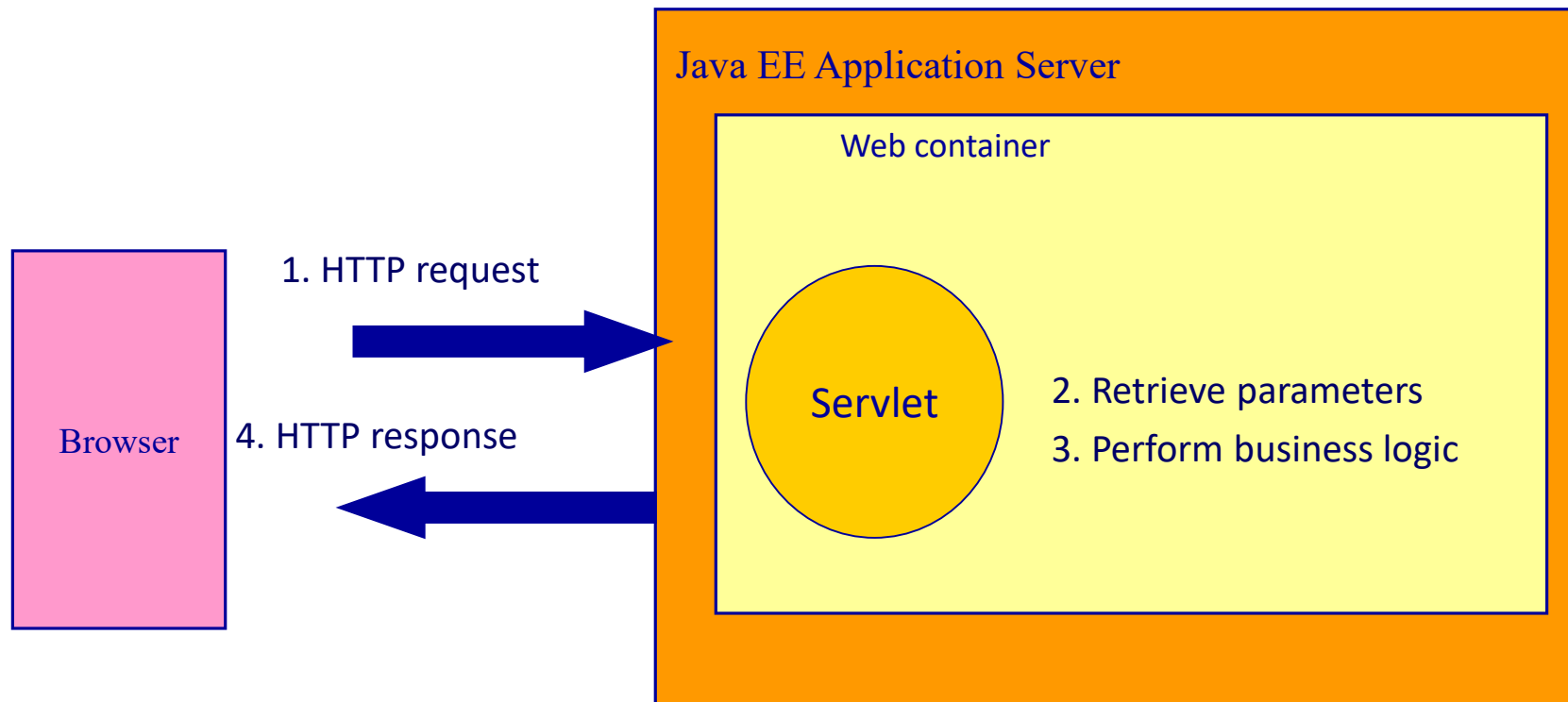


Vue

# **SERVLET**

# Servlet

- Java code that runs within a webcontainer



# Hello World Servlet

```
package hello;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class HelloWorld extends HttpServlet {

    protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        out.println("<html>");
        out.println("<body>");
        out.println("Hello World");
        out.println("</body>");
        out.println("</html>");
    }
}
```

The diagram consists of four orange speech bubbles with blue outlines, each pointing to a specific part of the Java code. The first bubble points to the `extends HttpServlet` line and contains the text "Extend from HttpServlet". The second bubble points to the `HttpServletRequest request` parameter in the `doGet` method signature and contains the text "Request object". The third bubble points to the `HttpServletResponse response` parameter in the `doGet` method signature and contains the text "Response object". The fourth bubble points to the HTML output lines (`out.println("<html>");`, `out.println("<body>");`, `out.println("Hello World");`, `out.println("</body>");`, and `out.println("</html>");`) and contains the text "HTML is written in the response object".

# web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
                            http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

  <display-name>ServletProject</display-name>
  <servlet>
    <servlet-name>HelloWorld</servlet-name>
    <servlet-class>hello.HelloWorld</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>HelloWorld</servlet-name>
    <url-pattern>/HelloWorld</url-pattern>
  </servlet-mapping>
</web-app>
```

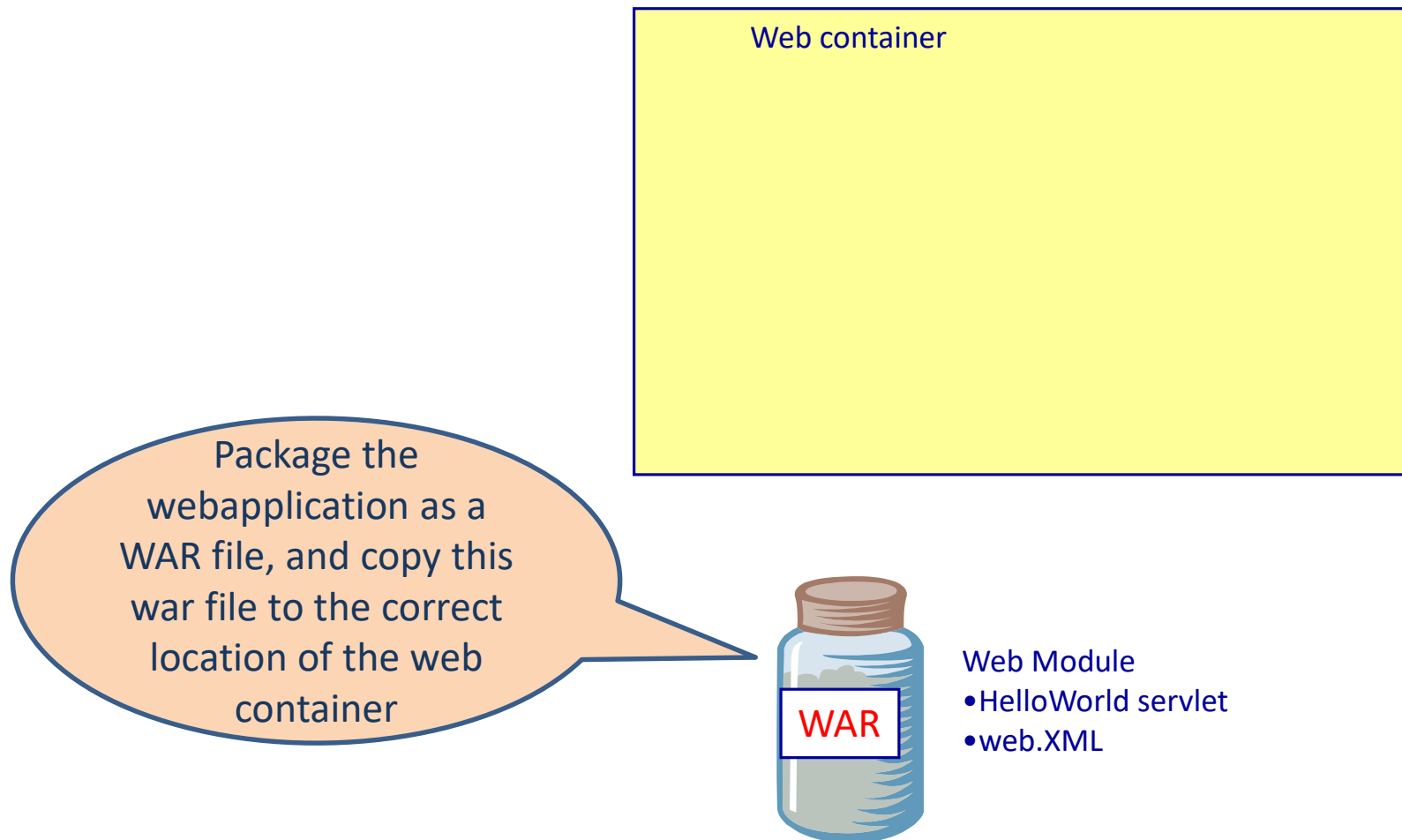
Map the servlet with  
name HelloWorld to  
the servlet class  
hello.HelloWorld

Map the URL  
/HelloWorld to the  
servlet with name  
HelloWorld

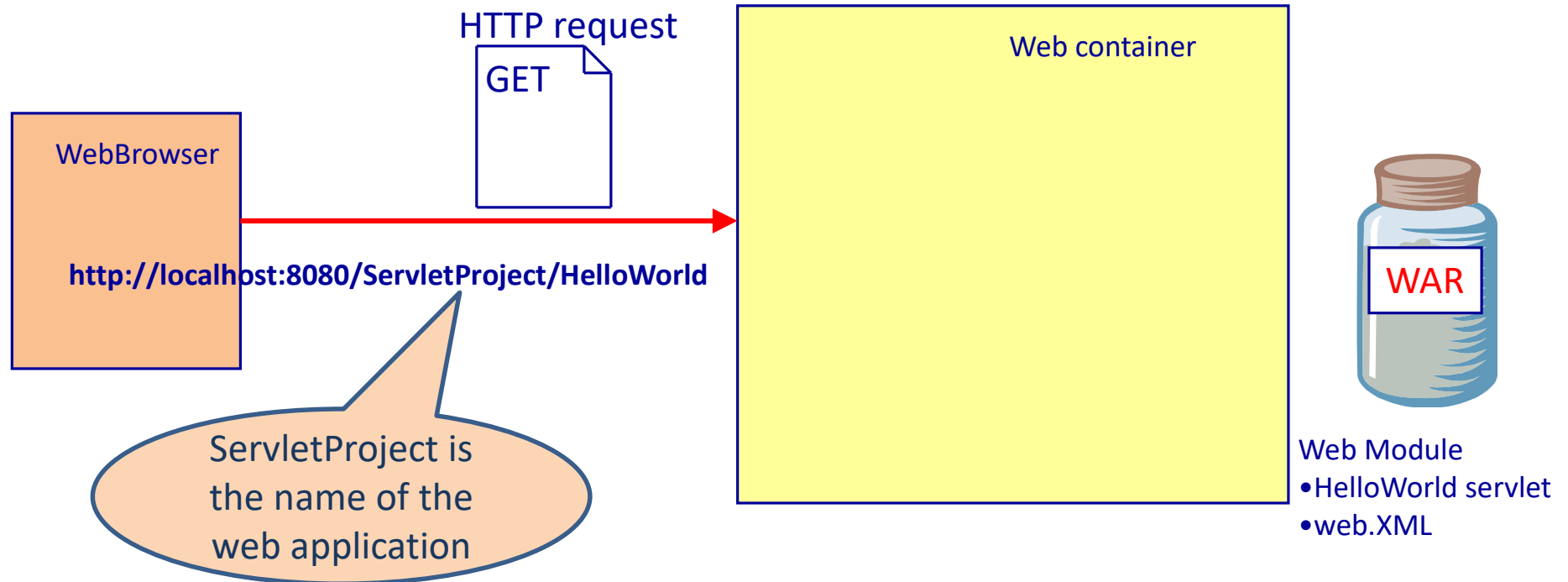
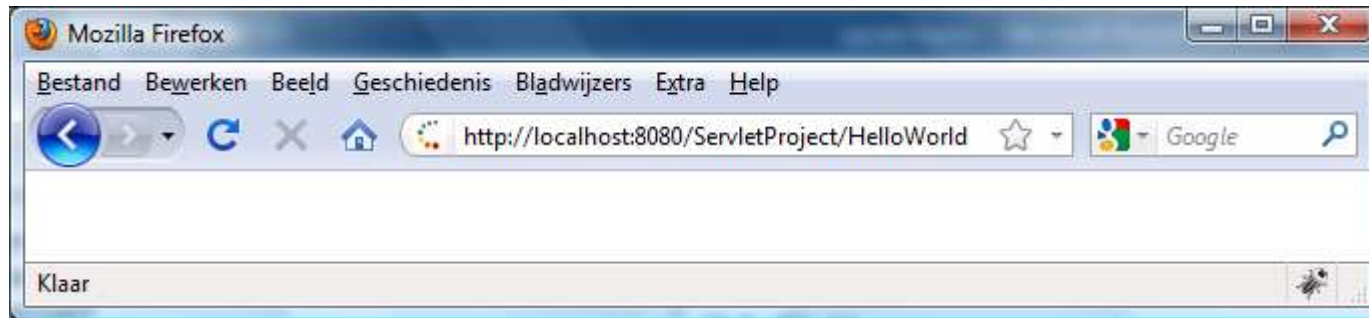
This basically says: send all  
requests with url /HelloWorld  
to the servlet class  
hello.HelloWorld

# Deploy the servlet

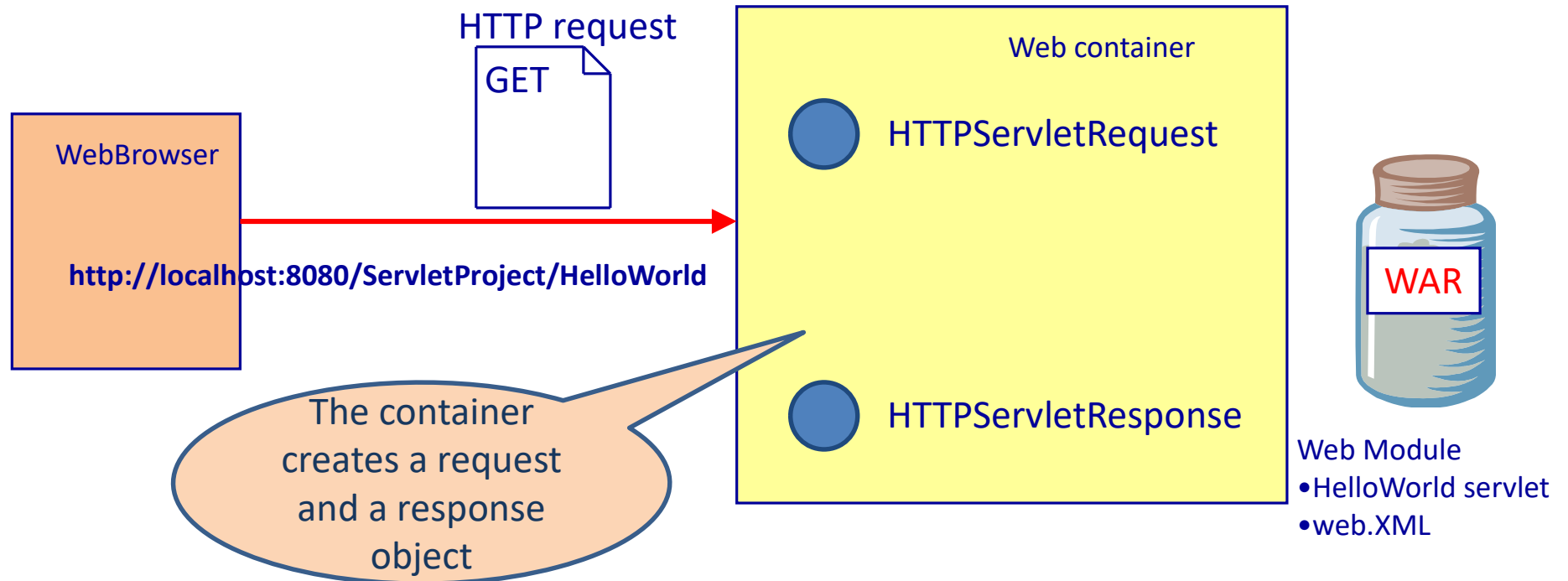
---



# Call the servlet 1

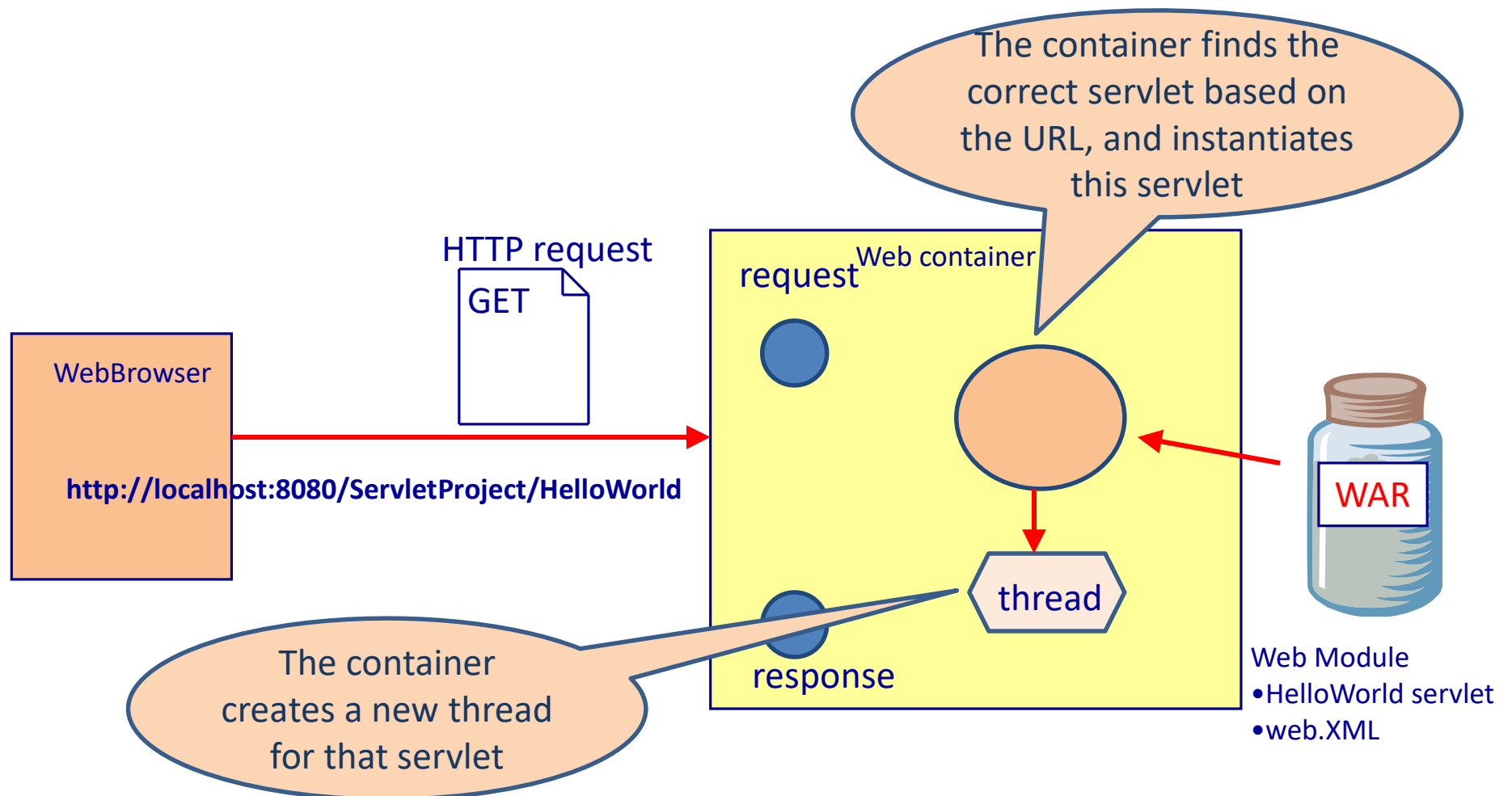


# Call the servlet 2

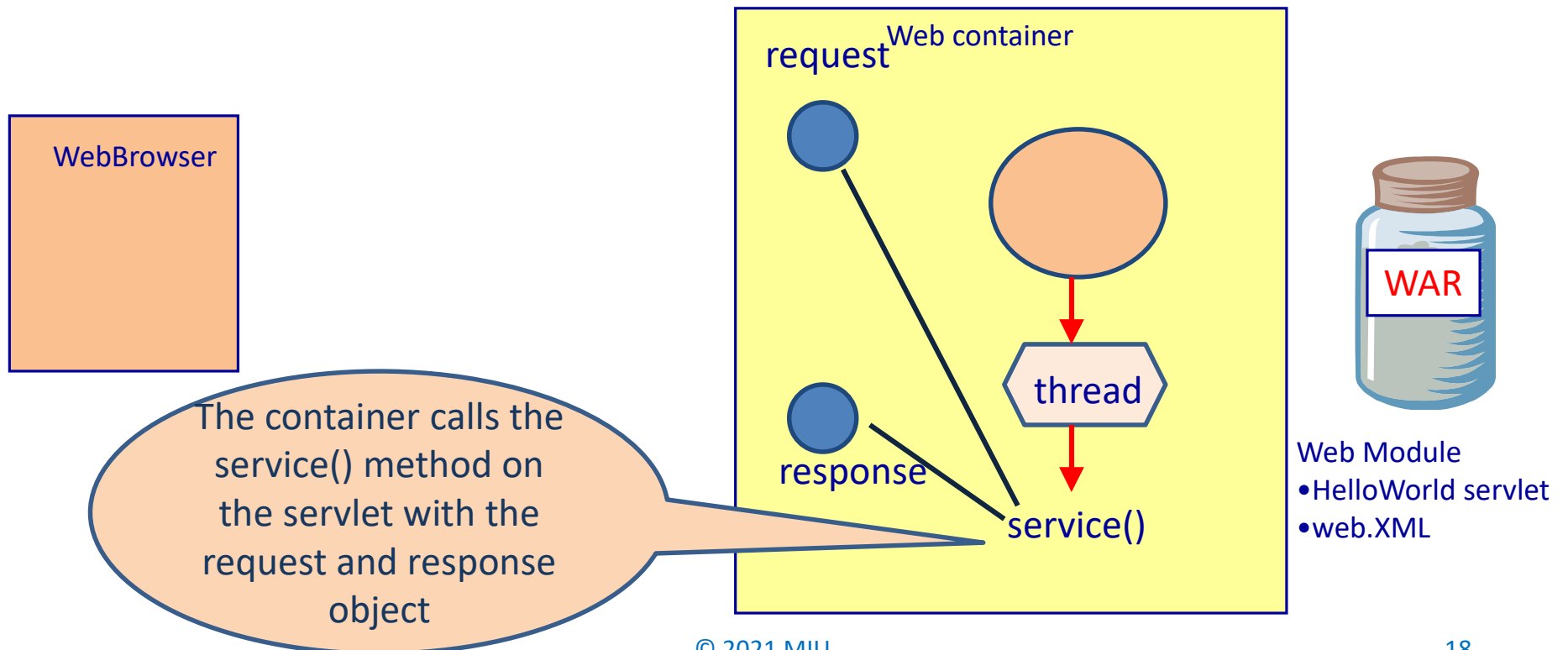




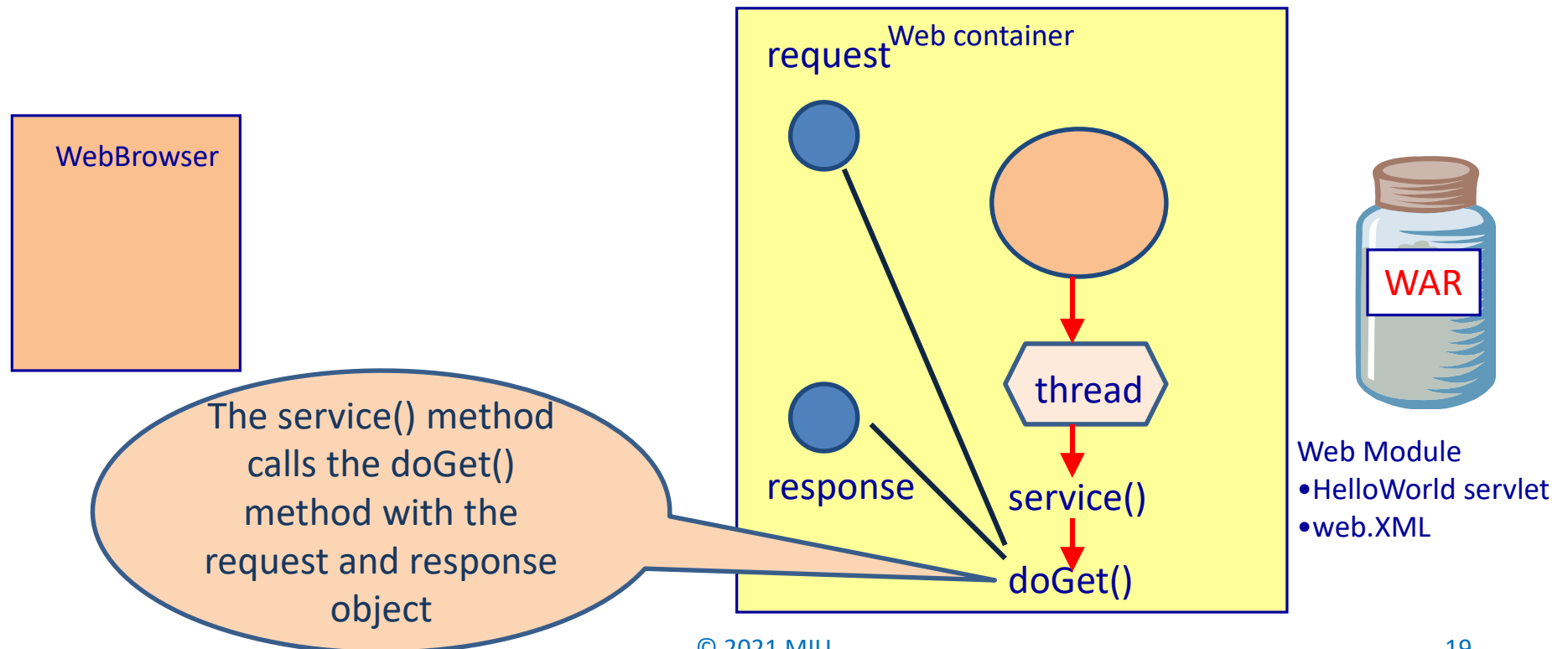
# Call the servlet 3



# Call the servlet 4

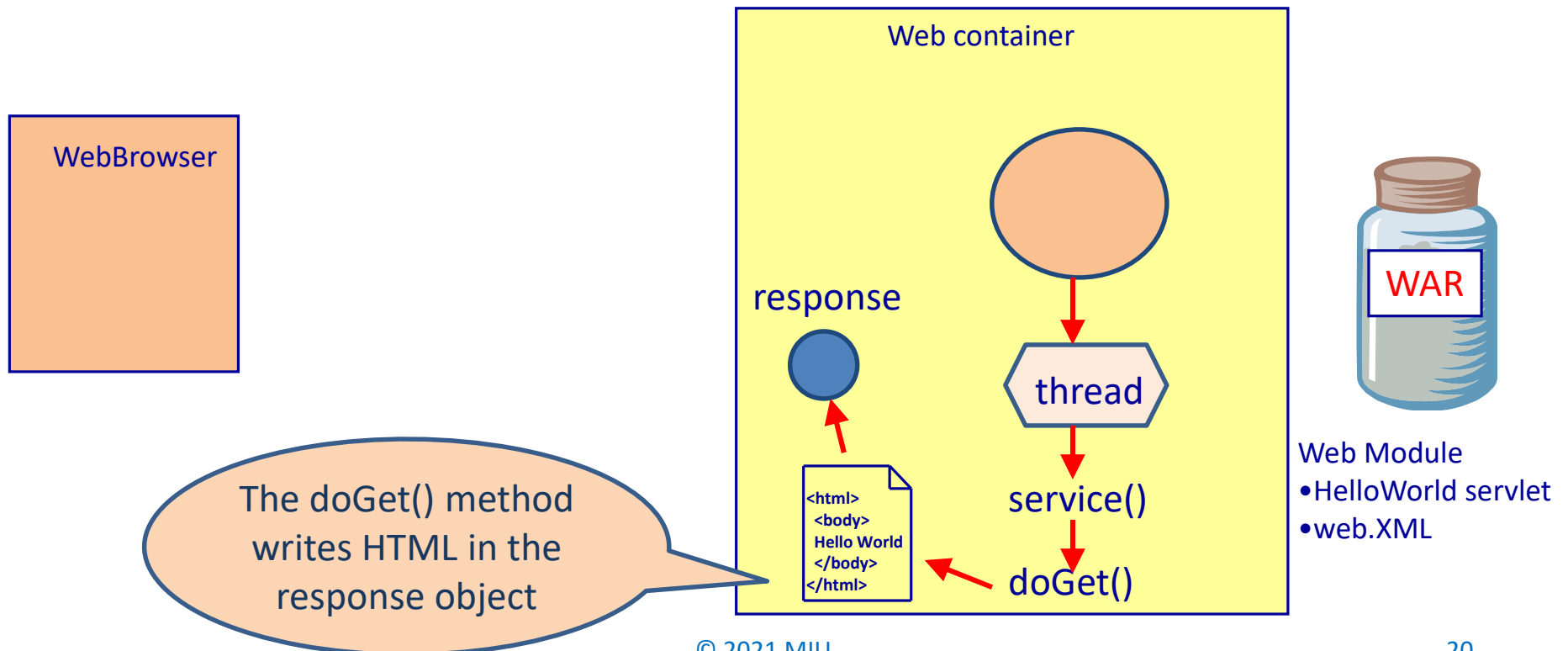


# Call the servlet 5

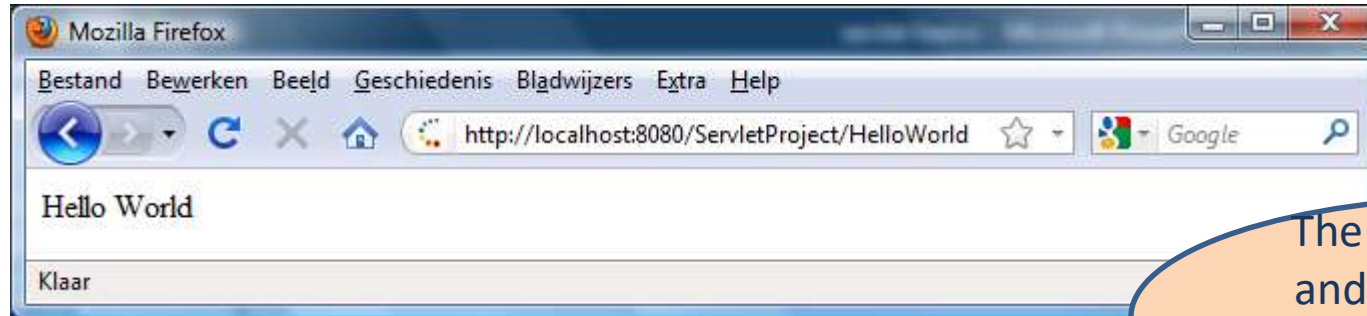


# Call the servlet 6

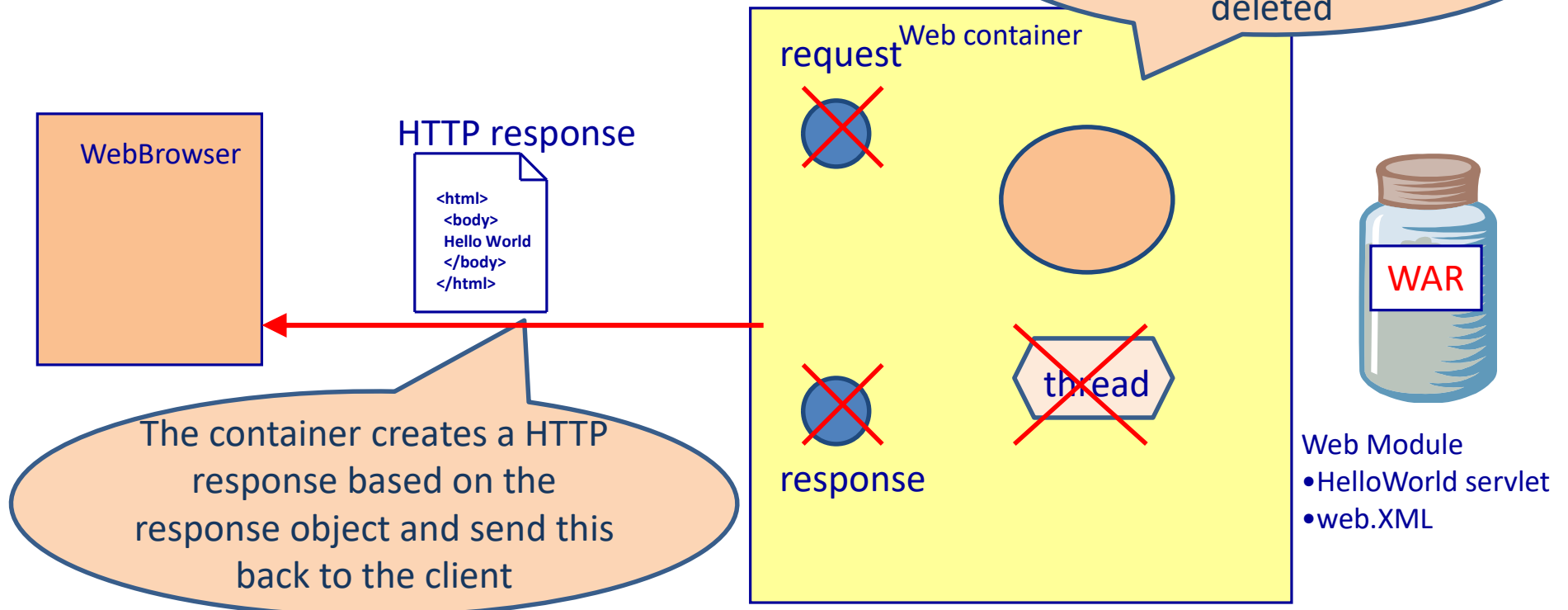
```
out.println("<html>");  
out.println("<body>");  
out.println("Hello World");  
out.println("</body>");  
out.println("</html>");
```



# Call the servlet 7



The tread completes and the request and response object are deleted



# REQUEST PARAMETERS

# Request parameters

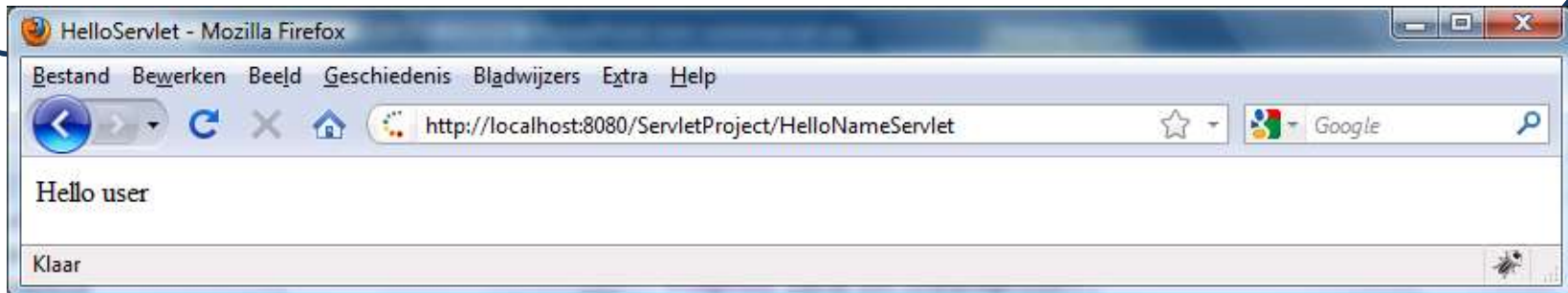
- URL parameters



# Receiving parameters

```
public class HelloNameServlet extends HttpServlet {  
  
    protected void doGet(HttpServletRequest request, HttpServletResponse  
                           response) throws ServletException, IOException {  
        response.setContentType("text/html");  
        PrintWriter out = response.getWriter();  
  
        String userName = request.getParameter("userName");  
        if (userName == null) {  
            userName= "user";  
        }  
        out.println("<html>");  
        out.println("<head><title>HelloServlet</title></head>");  
        out.println("<body bgcolor=\"#ffffff\">");  
        out.println("Hello "+userName);  
        out.println("</body>");  
        out.println("</html>");  
    }  
}
```

Get the parameter from  
the request object

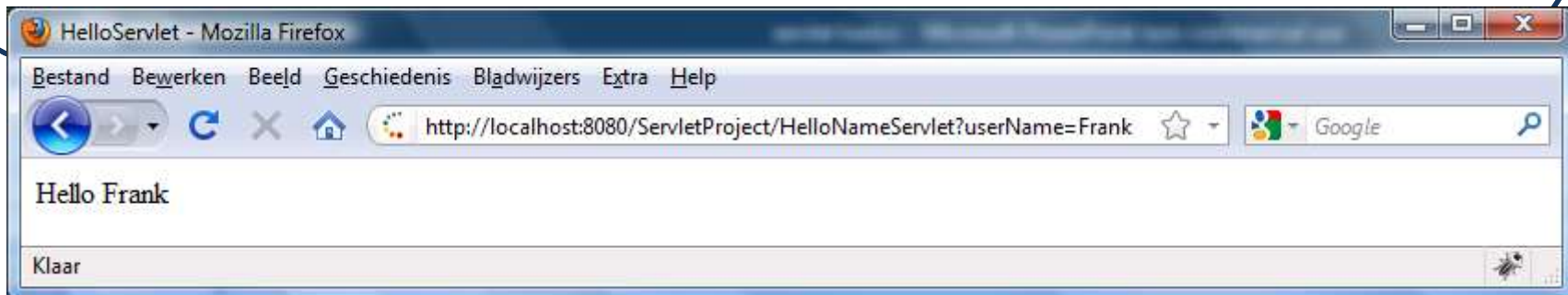




# Receiving parameters

```
public class HelloNameServlet extends HttpServlet {  
  
    protected void doGet(HttpServletRequest request, HttpServletResponse  
                           response) throws ServletException, IOException {  
        response.setContentType("text/html");  
        PrintWriter out = response.getWriter();  
  
        String userName = request.getParameter("userName");  
        if (userName == null) {  
            userName= "user";  
        }  
        out.println("<html>");  
        out.println("<head><title>HelloServlet</title></head>");  
        out.println("<body bgcolor=\"#ffffff\">");  
        out.println("Hello "+userName);  
        out.println("</body>");  
        out.println("</html>");  
    }  
}
```

Get the parameter from  
the request object

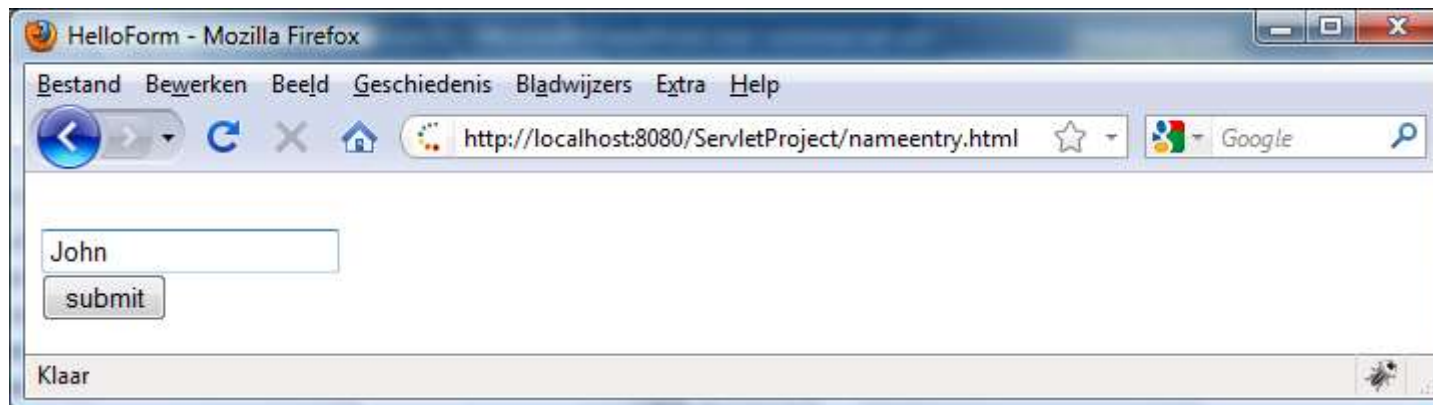


# HTML FORMS

# HTML Forms

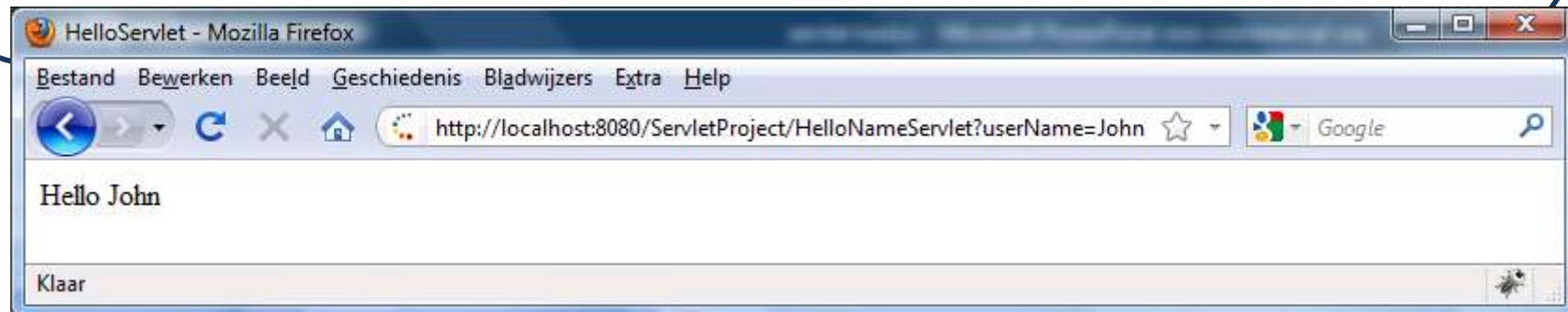
```
<html>
<head>
  <title>HelloForm</title>
</head>
<body bgcolor="#ffffff">
  <form method="get" action="HelloNameServlet">
    <br>
    <INPUT TYPE="TEXT" NAME="userName">
    <br/>
    <input type="submit" value="submit">
  </form>
</body>
</html>
```

Create a GET request  
and send this to the  
servlet with the servlet  
mapping  
/HelloNameServlet



# The servlet

```
public class HelloNameServlet extends HttpServlet {  
  
    protected void doGet(HttpServletRequest request, HttpServletResponse  
                           response) throws ServletException, IOException {  
        response.setContentType("text/html");  
        PrintWriter out = response.getWriter();  
  
        String userName = request.getParameter("userName");  
        if (userName == null) {  
            userName= "user";  
        }  
        out.println("<html>");  
        out.println("<head><title>HelloServlet</title></head>");  
        out.println("<body bgcolor=\"#ffffff\">");  
        out.println("Hello "+userName);  
        out.println("</body>");  
        out.println("</html>");  
    }  
}
```



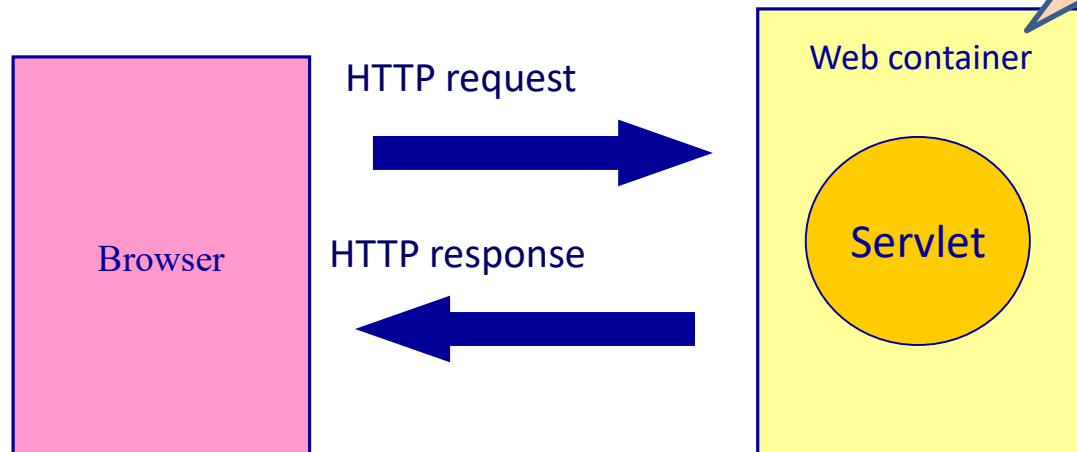
**STATE**

# HTTP is stateless

- Browser makes connection
- Browser sends a HTTPRequest
- Browser receives a HTTPResponse
- Browser closes the connection

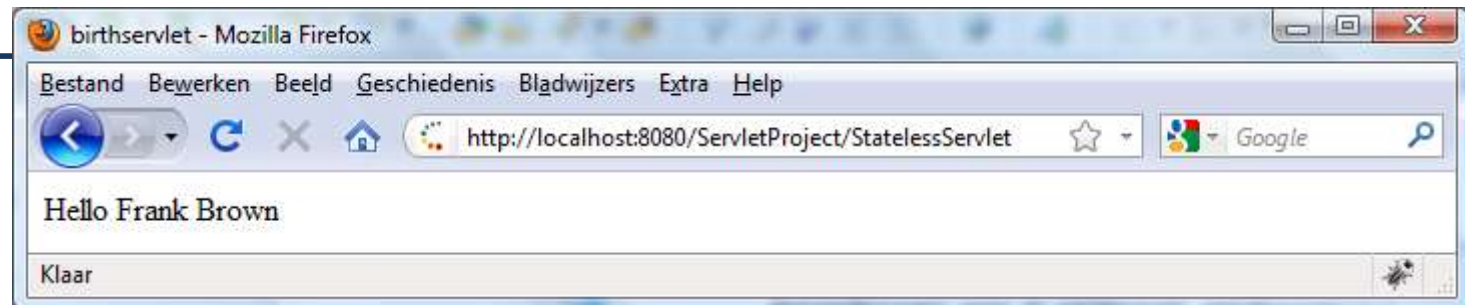
Every request-response cycle has its own connection

Every request is a unique request for the webcontainer

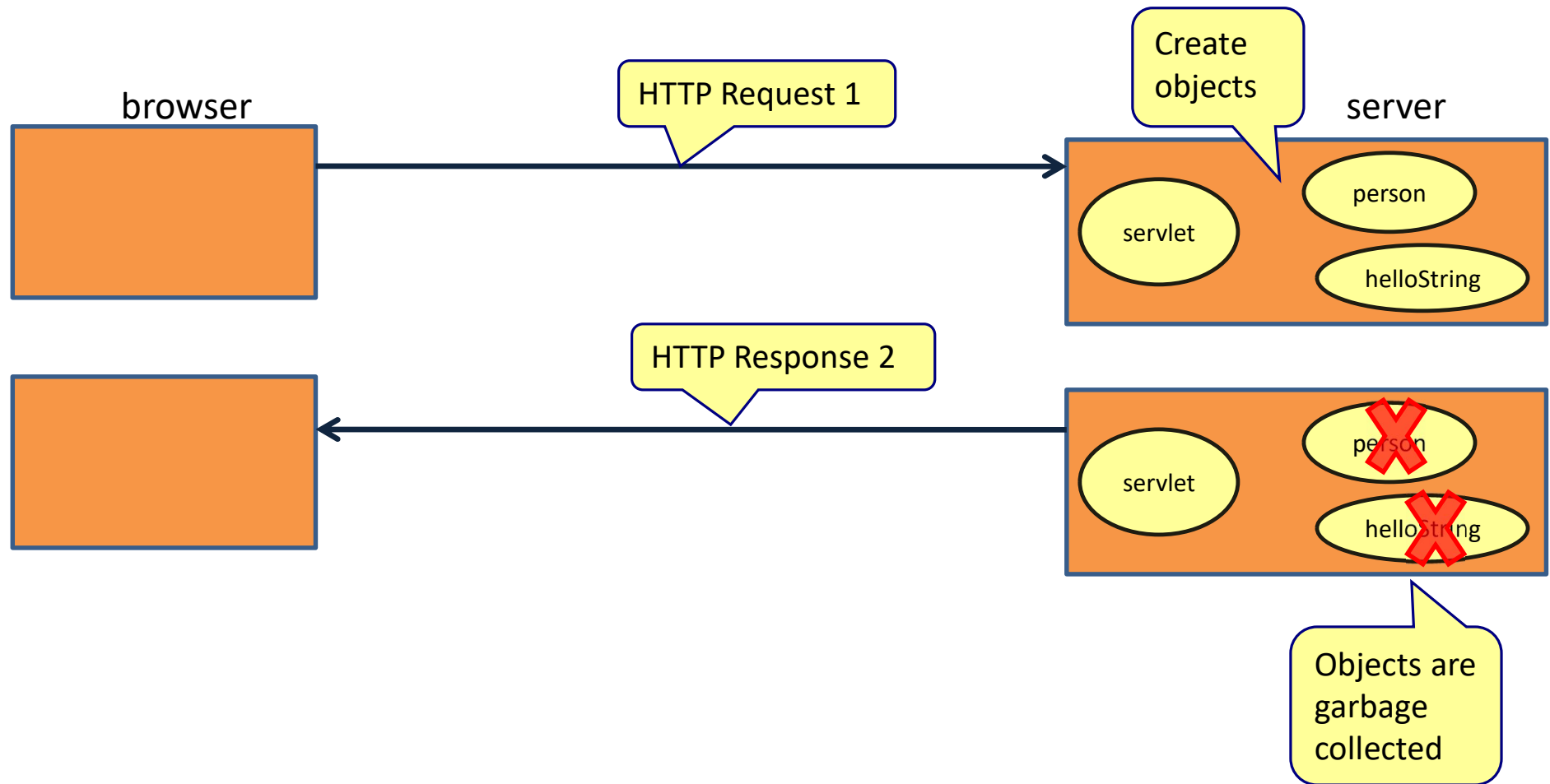


# StatelessServlet

```
public class StatelessServlet extends HttpServlet {  
    protected void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        response.setContentType("text/html");  
        PrintWriter out = response.getWriter();  
        // create a Person object  
        Person person = new Person("Frank", "Brown");  
        String hellostring = "Hello";  
  
        out.println("<html>");  
        out.println("<head><title>birthservlet</title></head>");  
        out.println("<body>");  
        out.println(hellostring+person.getFirstName()+" "+person.getLastName());  
        out.println("</body>");  
        out.println("</html>");  
    }  
}
```



# StatelessServlet





# HttpSession

---

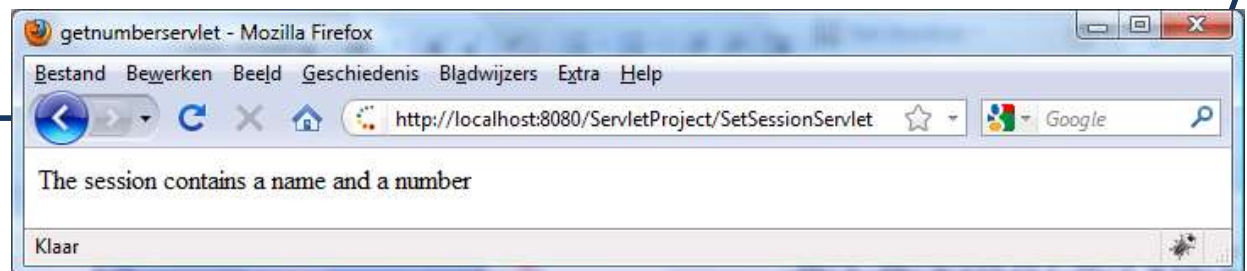
- An object that is unique for every client (browser)
- Can hold objects between client request
- Is created by the container
- Is destroyed by the container
  - During a session timeout
- The container makes sure every client has a unique HttpSession object
  - The container connects clients and HttpSession objects with a unique session id for every client.

# SetSessionServlet

```
public class SetSessionServlet extends HttpServlet {  
    public void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        response.setContentType("text/html");  
        PrintWriter out = response.getWriter();  
        out.println("<html>");  
        out.println("<head><title>getnumberservlet</title></head>");  
        out.println("<body>");  
        //get the session object  
        HttpSession session = request.getSession();  
        //add a number and a name in the session  
        Integer number = new Integer(6);  
        session.setAttribute("mycount", number);  
        String name="Frank Brown";  
        session.setAttribute("thename", name);  
  
        out.println("<html>");  
        out.println("<body>The session contains a name and a number");  
        out.println("</body></html>");  
    }  
}
```

Get the HttpSession

Add in the session

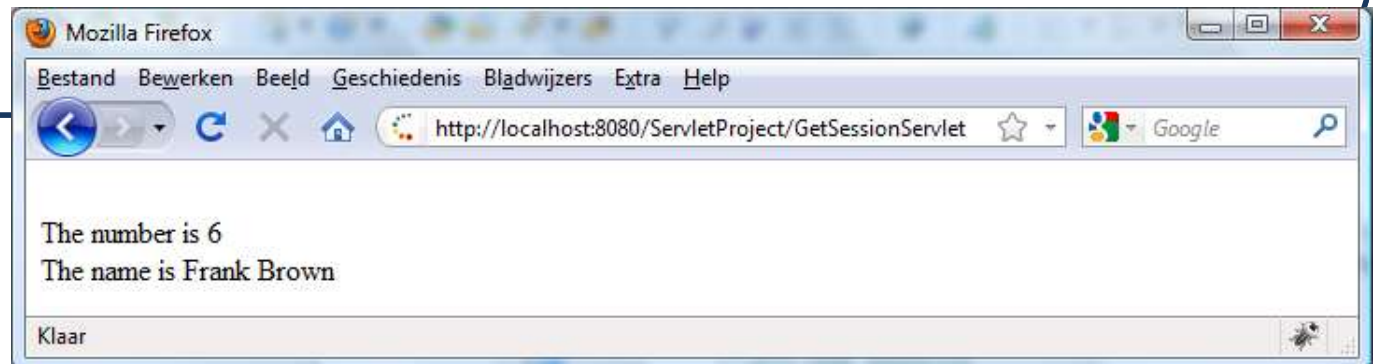


# GetSessionServlet

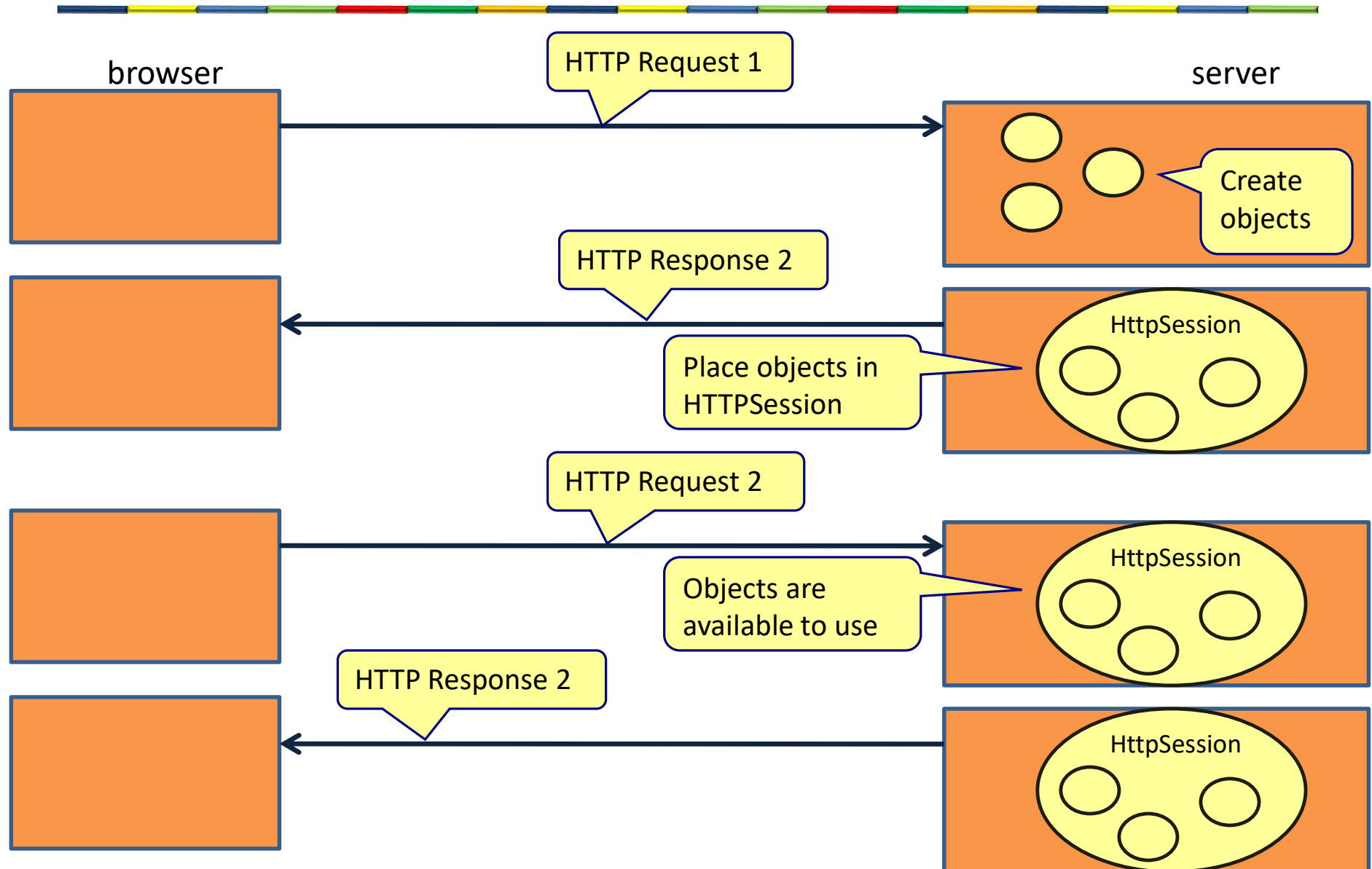
```
public class GetSessionServlet extends HttpServlet {  
    public void doGet(HttpServletRequest request, HttpServletResponse  
        response) throws ServletException, IOException {  
        response.setContentType("text/html");  
        PrintWriter out = response.getWriter();  
        //get the session  
        HttpSession session = request.getSession();  
        //get the name and number from the session  
        Integer number = (Integer) session.getAttribute("mycount");  
        String name = (String) session.getAttribute("thename");  
  
        out.println("<html>");  
        out.println("<body>");  
        out.println("<br/>The number is "+number);  
        out.println("<br/>The name is "+name);  
        out.println("</body>");  
        out.println("</html>");  
    }  
}
```

Get the HttpSession

Get from the  
session



# HttpSession



# **SERVLETS WITH SPRING BOOT**

# Spring and Spring Boot

---

- Spring framework
  - Java framework that makes programming enterprise Java applications simpler
- Spring Boot framework
  - Makes writing spring applications even simpler
  - Convention over configuration

# Spring web dependency

---

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-web</artifactId>  
</dependency>
```

Adds the Tomcat webcontainer in the application

# Containerless deployment

---



## Container Deployments

- Pre-setup and configuration
- Need to use files like web.xml to tell container how to work
- Environment configuration is external to your application



## Application Deployments

- Runs anywhere Java is setup (think cloud deployments)
- Container is embedded and the app directs how the container works
- Environment configuration is internal to your application



# HelloWorldServlet

Servlet mapping

```
@WebServlet("/hello")
```

```
public class HelloWorldServlet extends HttpServlet {
```

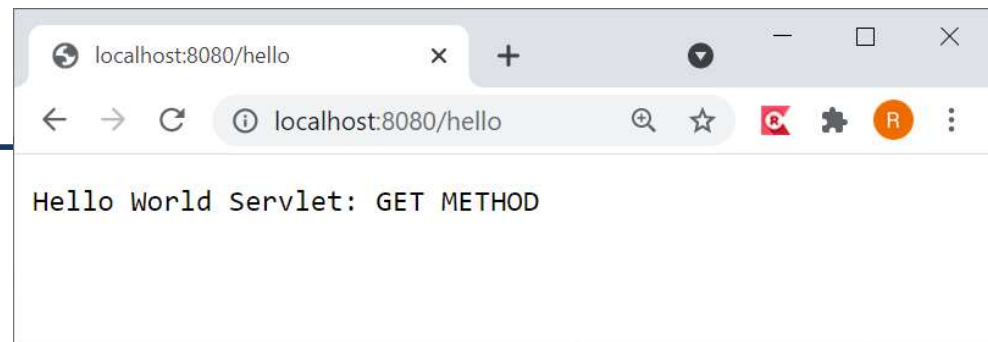
```
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
                                                                    IOException {
```

```
        PrintWriter out = response.getWriter();
        out.println("Hello World Servlet: GET METHOD");
        out.flush();
    }
```

```
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException,
                                                                    IOException {
```

```
        PrintWriter out = response.getWriter();
        out.println("Hello World Servlet: POST METHOD");
        out.flush();
    }
```

```
}
```



# The application class

Scan for classes with  
@WebSevlet annotation

```
@ServletComponentScan
@SpringBootApplication(exclude = { DispatcherServletAutoConfiguration.class, ErrorMvcAutoConfiguration.class })
public class SpringBootWebApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringBootWebApplication.class, args);
    }
}
```

# Main point

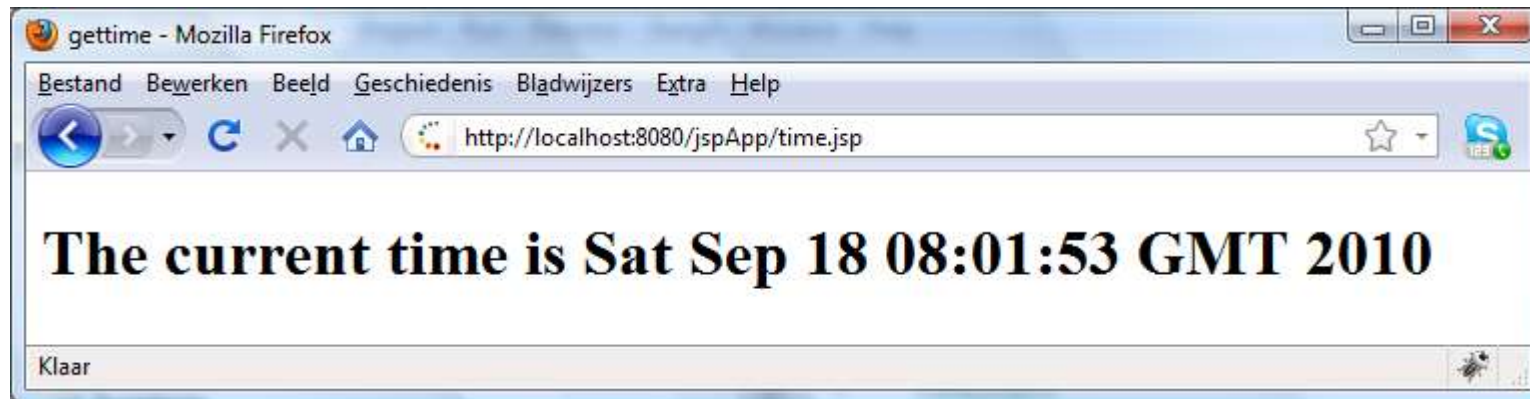
---

- Servlets are the basis of dynamic web applications. Servlets process information from a request object and generate new information in a response object. *For every action in nature there is always a reaction.*

**JSP**

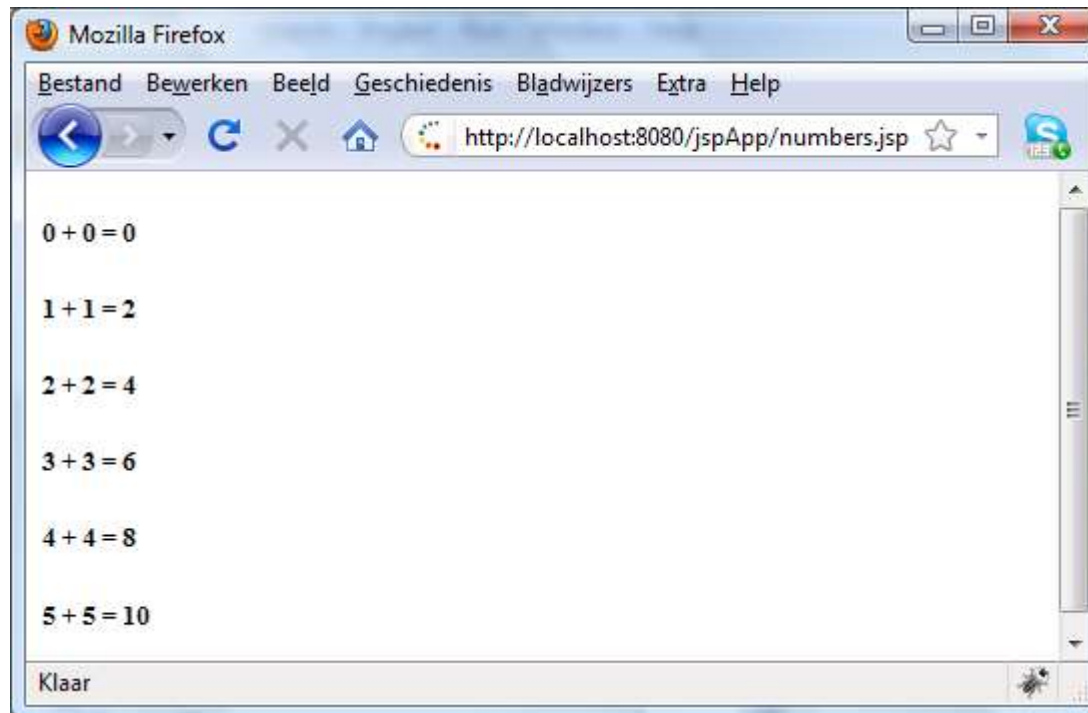
# time.jsp

```
<html>
  <head>
    <title>
      gettime
    </title>
  </head>
  <body>
    <h1>
      The current time is <%= new java.util.Date() %>
    </h1>
  </body>
</html>
```



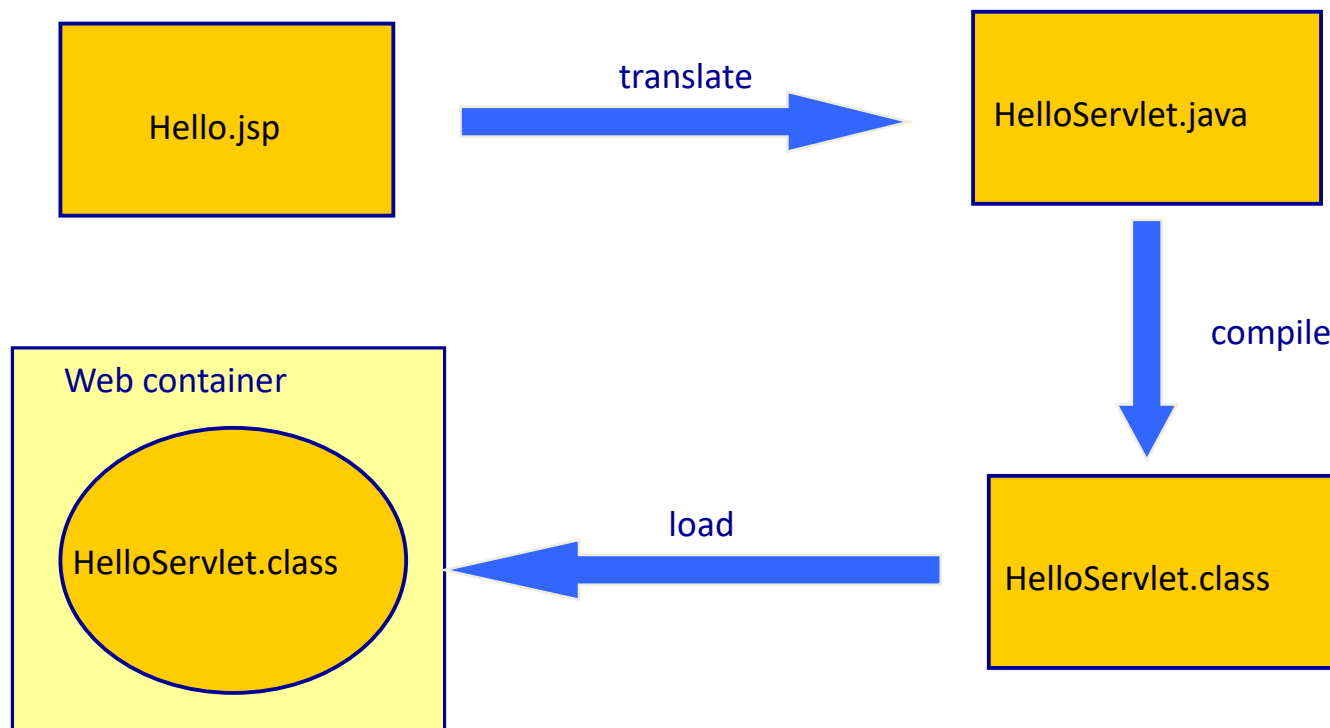
# numbers.jsp

```
<html>
  <body>
    <% for(int x=0; x<6; x++){ %>
      <h5><%= x %> + <%= x %> = <%= x+x %></h5>
    <% } %>
  </body>
</html>
```



# Every JSP becomes a servlet

---



# The servlet class

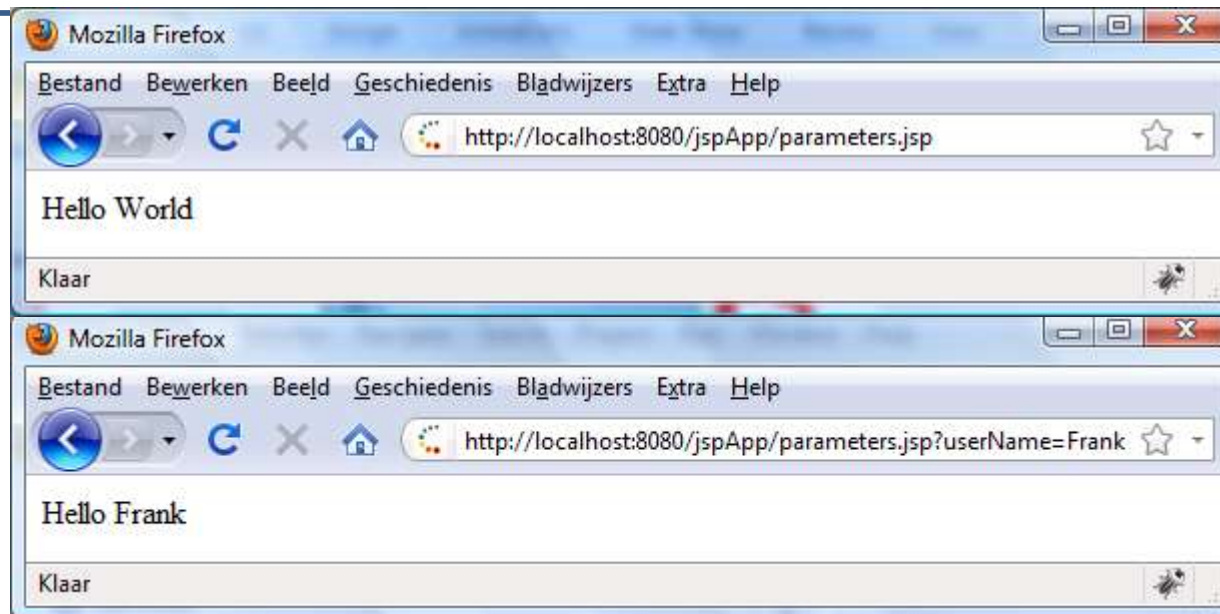
```
<html>
  <head>
    <title>
      gettime
    </title>
  </head>
  <body>
    <h1>
      The current time is <%= new java.util.Date() %>
    </h1>
  </body>
</html>
```

```
public final class time_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {
    ...
    public void _jspInit() {... }
    public void _jspDestroy() {}
    public void _jspService(HttpServletRequest request, HttpServletResponse
        response) throws java.io.IOException, ServletException {
        ...
    }
}
```



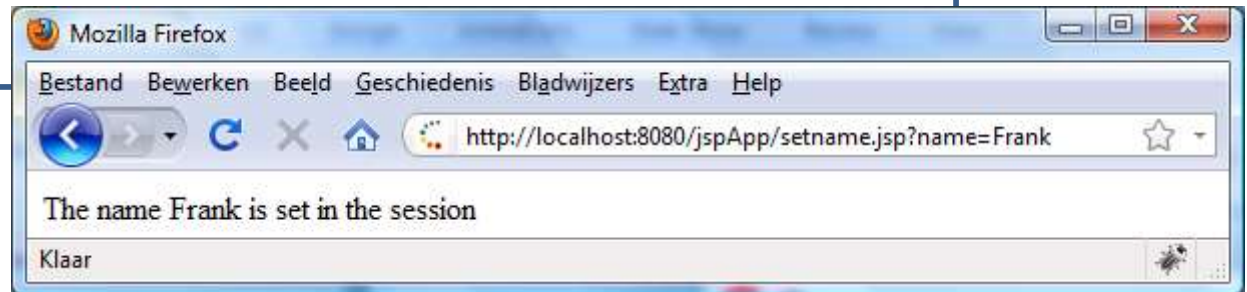
# Getting parameters

```
<html>
<body>
  <%
    String name= request.getParameter ("userName");
    if (name == null){
      name="World";
    }
  %>
  Hello <%= name %>
</body>
</html>
```

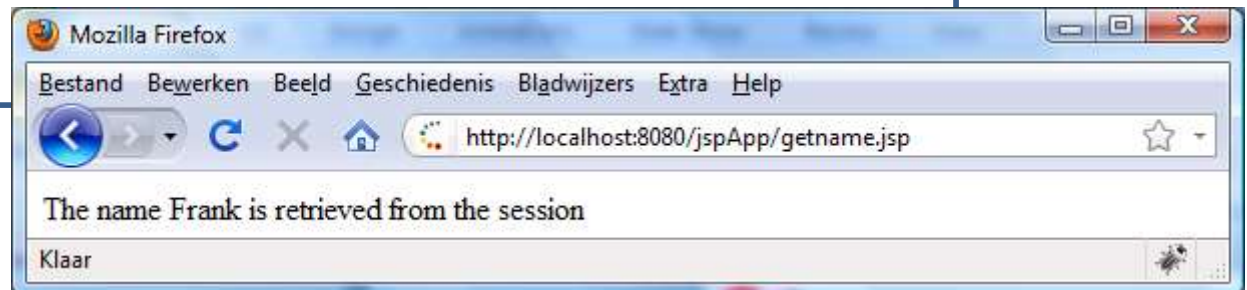


# Session

```
<html>
  <body>
    <%
      String name= request.getParameter ( "name" );
      session.setAttribute( "theName", name );
    %>
    The name <%= name %> is set in the session
  </body>
</html>
```



```
<html>
  <body>
    The name <%= session.getAttribute( "theName" ) %> is retrieved
    from the session
  </body>
</html>
```



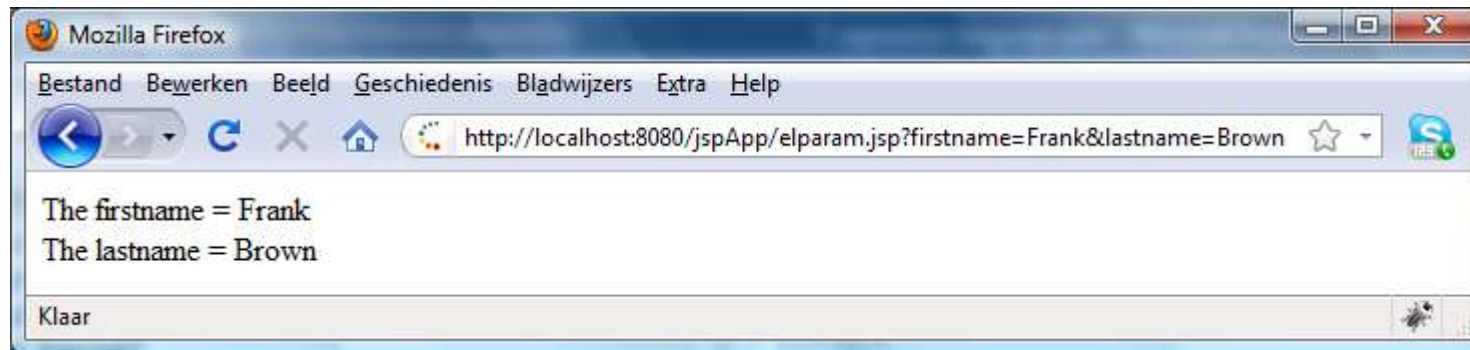
# Expression Language (EL)

---

- The EL provides identifiers, accessors, and operators for retrieving and manipulating data resident in the JSP container
- `${ el expression }`

# Request parameters

```
<html>
  <body>
    The firstname = ${param.firstname}<br/>
    The lastname = ${param.lastname}<br/>
  </body>
</html>
```



# JSP standard actions

---

- `<jsp:useBean>`
- `<jsp:getProperty>`
- `<jsp:setProperty>`
- `<jsp:include>`
- `<jsp:forward>`
- `<jsp:param>`

# <jsp:useBean>, <jsp:getProperty>

```
<html>
<head>
</head>
<body>
  <jsp:useBean id="person" class="domain.Person" />
  Name is: <jsp:getProperty name="person" property="name" />
  <br></br>
  Age is: <jsp:getProperty name="person" property="age" />
</body>
</html>
```

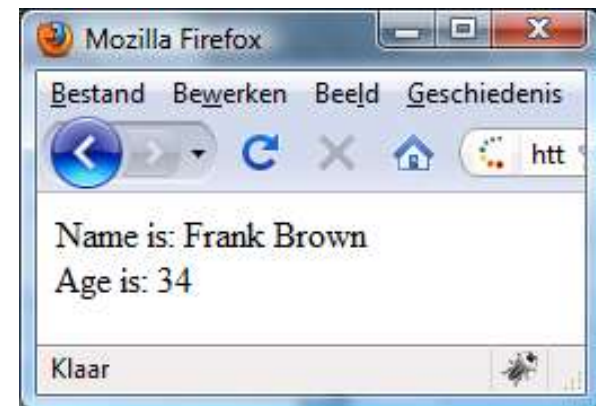
Find the bean with name person in the page scope, if you can't find it, create it.

Get the age property of the bean with name person

```
package domain;

public class Person {
  private String name = "Frank Brown";
  private int age = 34;

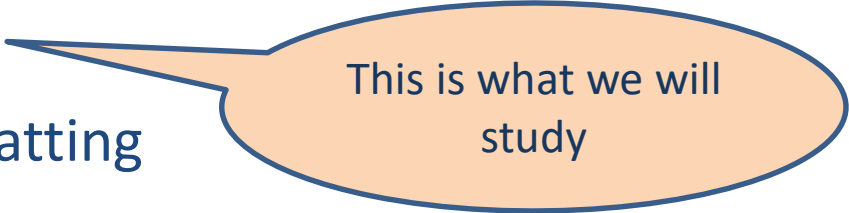
  //getter and setter methods ...
}
```



# JSTL

---

- Java Standard Tag Library
- Contains 5 libraries
  - 4 with custom tags
    - Core
    - Formatting
    - SQL
    - XML
  - 1 with functions for String manipulation

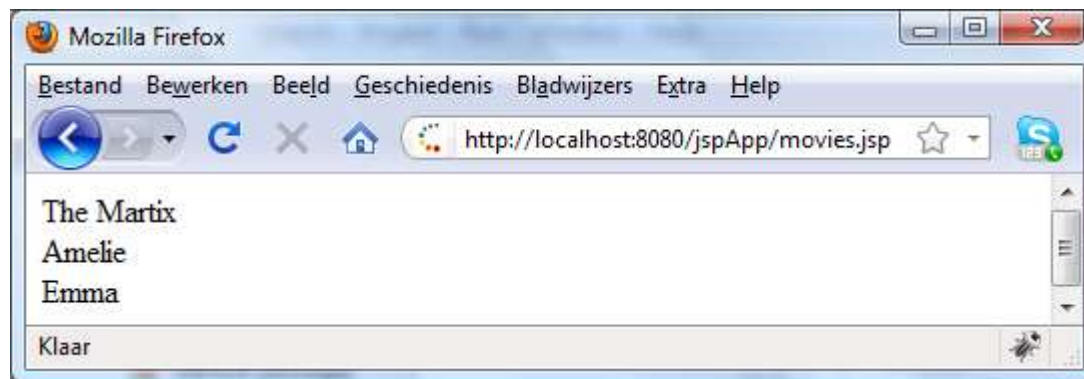


This is what we will study

# <c:forEach>

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
  <body>
    <% String[] movieList = {"The Martix", "Amelie", "Emma"};
       pageContext.setAttribute("movies", movieList); %>

    <c:forEach var="movie" items="${movies}">
      ${movie}<br/>
    </c:forEach>
  </body>
</html>
```





# <c:if>

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
  <body>
    <% pageContext.setAttribute("employee", "Frank Brown");
    pageContext.setAttribute("usertype", "manager");%>

    Employee name= ${employee}<br>
    <c:if test="${usertype eq 'manager' }" >
      The salary = $3200.00
    </c:if>
  </body>
</html>
```



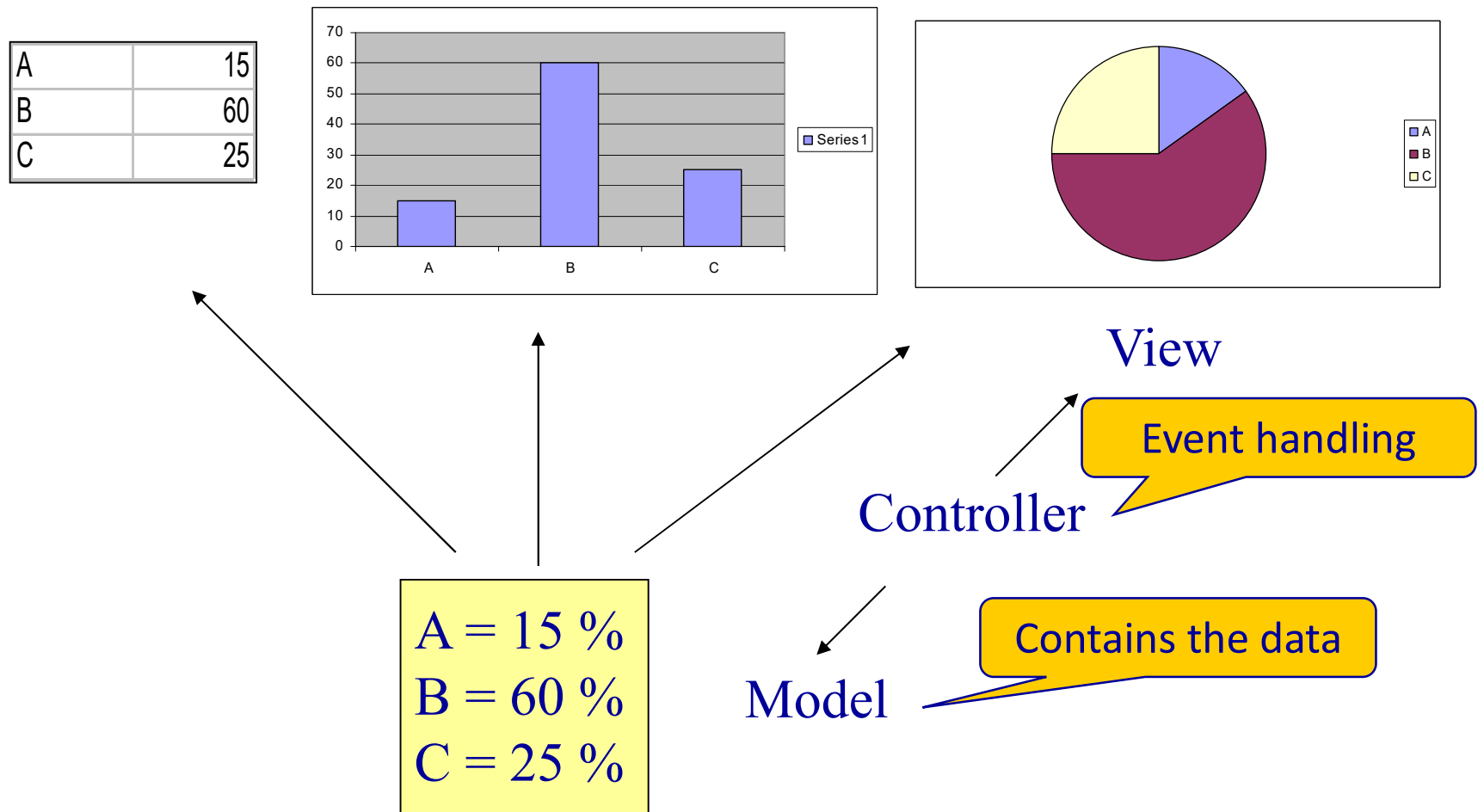
# Main point

---

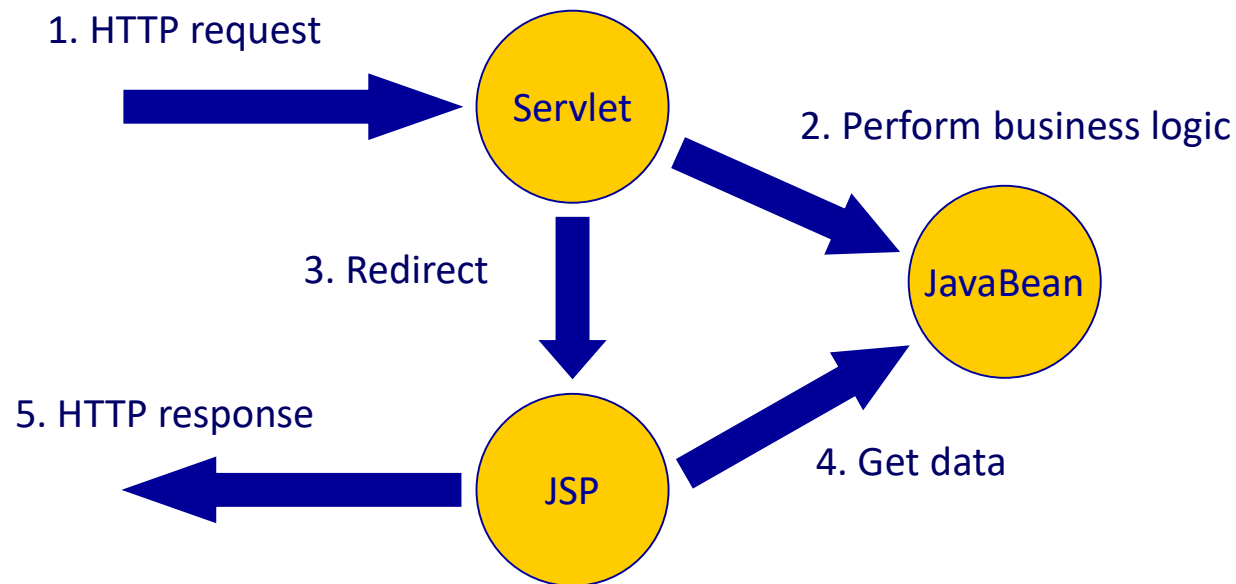
- The web container generates a servlet from a JSP file the first time the JSP is requested from a web application. Since a JSP is essentially a servlet, one should understand servlets to effectively deal with JSPs. *Actions in accord with fundamental levels of intelligence result in success in the relative world.*

# MVC

# Model-View-Controller



# Model-View-Controller with servlets and JSP's



# MVC

```
<body>
<form action="Logonservlet" method="post">
Username <input type="text" name="username" size="20"><br>
Password <input type="text" name="password" size="20"><br>
<input type="submit" value="submit" />
</form>
</body>
```

login.jsp

```
<body>
Welcome
</body>
```

success.jsp

```
<body>
Sorry, wrong userid or password
</body>
```

error.jsp



# LogonServlet

```
public class Logonservlet extends HttpServlet {

    protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
        String username = request.getParameter("username");
        String password = request.getParameter("password");

        LogonHandler handler = new LogonHandler();
        boolean successlogon = handler.handleLogon(username, password);
        if (successlogon){
            response.sendRedirect("success.jsp");
        }
        else{
            response.sendRedirect("error.jsp");
        }
    }
}
```

# LogonHandler

---

```
public class LogonHandler {
    String userFullName;

    public boolean handleLogon(String username, String password){
        if ("user".equals(username) && "pass".equals(password)){
            userFullName = "Frank Brown";
            return true;
        }
        else
            return false;
    }
    public String getUserFullName() {
        return userFullName;
    }
    public void setUserFullName(String userFullName) {
        this.userFullName = userFullName;
    }
}
```



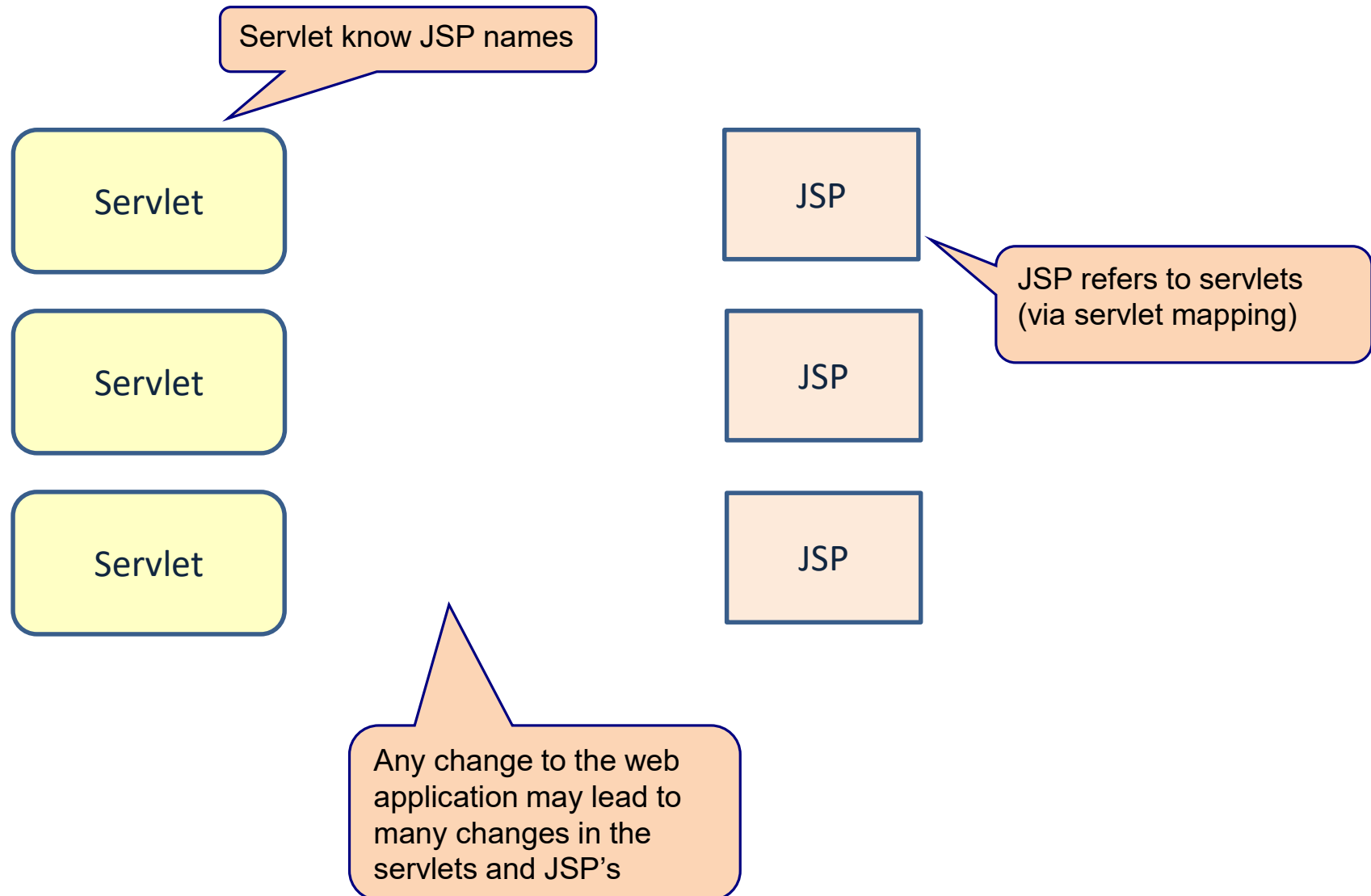
# Main point

---

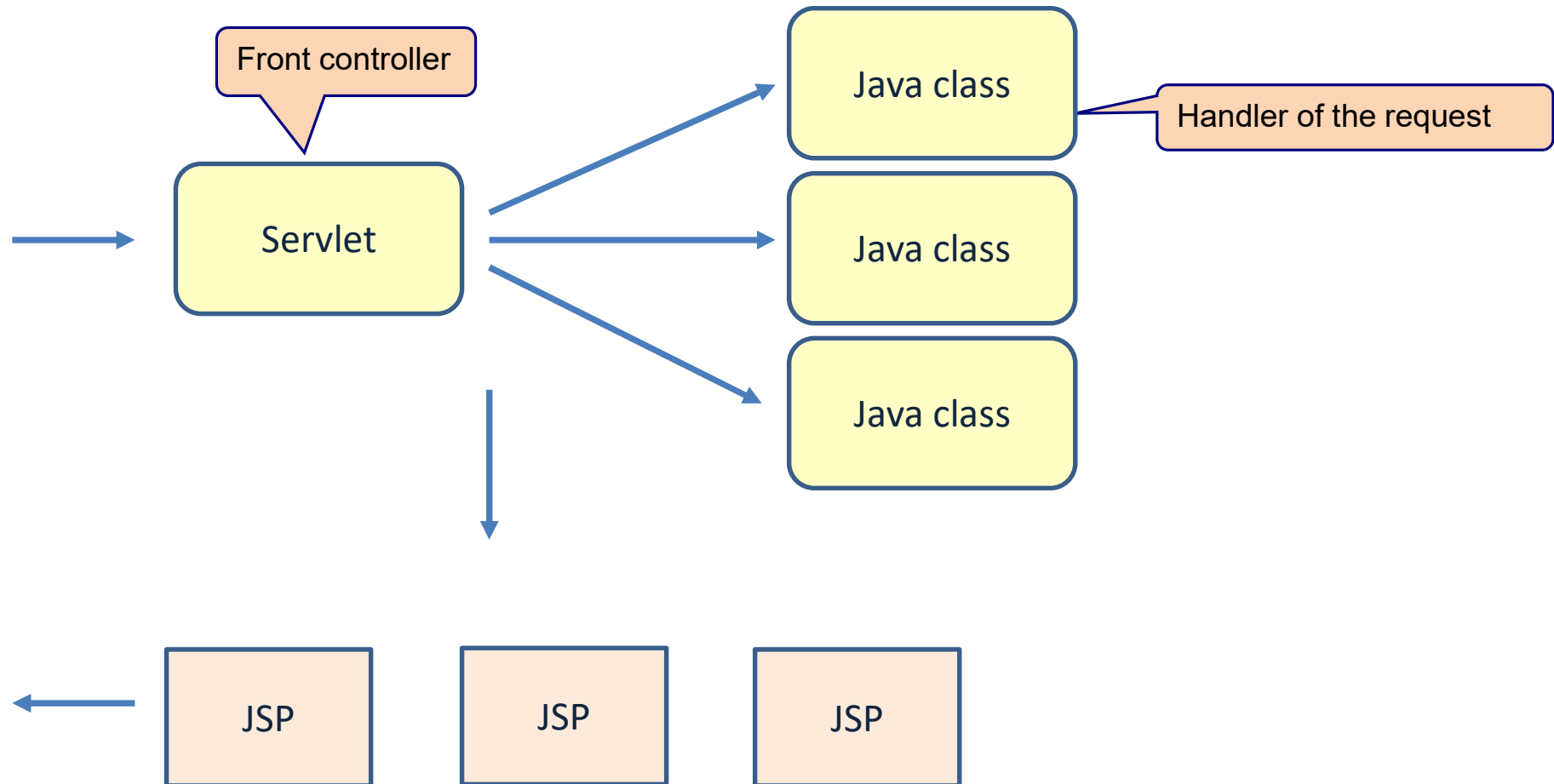
- In model-view-controller (MVC) you separate the responsibilities of the web logic. *Pure consciousness transforms itself into knower (Rishi), the known (Chhandas) and the process of knowing (Devata) while maintaining unity.*

# FRONT CONTROLLER

# Problem with plain MVC



# Front controller



# **SPRING MVC WITH SPRING BOOT**

# View technology

---

- SpringMVC does not contain a view technology
- SpringMVC view technologies
  - JSP
  - Freemarker
  - Thymeleaf
  - Groovy
  - Velocity

# Thymeleaf instead of JSP

---

- JSP
  - Needs to be translated into a servlet before you can view it
    - Only once
  - You can write java into a JSP
    - But you should not
  - You need to compile and deploy the JSP before you can view it.
- Thymeleaf
  - Is only a view template that looks very close to HTML
  - You can view the Thymeleaf page in the browser without compiling

# Spring MVC libraries

---

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-web</artifactId>  
</dependency>  
  
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-thymeleaf</artifactId>  
</dependency>
```



# SpringMVC example

```
@Controller
public class HelloController {
    @RequestMapping("/hello")
    public String hello(Model model, @RequestParam(value="name", required=false,
        defaultValue="World") String name) {

        model.addAttribute("name", name);
        return "hello";
    }
}
```

url = /hello and method is GET (default)

The name is send as an parameter

Place the name in the model

Show the view with name "hello"

```
@SpringBootApplication
public class SpringMvcDemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringMvcDemoApplication.class, args);
    }
}
```

Spring boot application

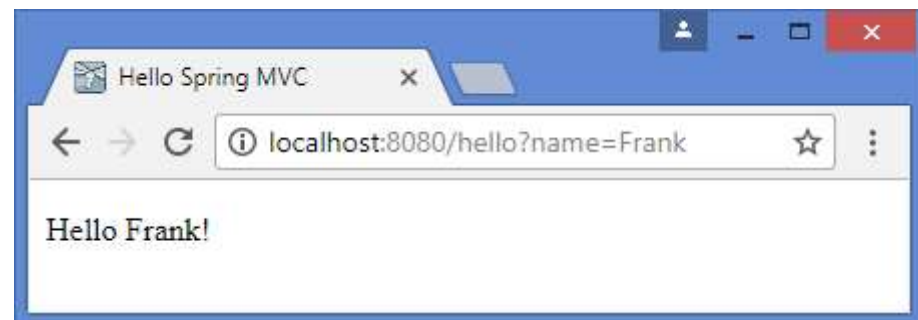
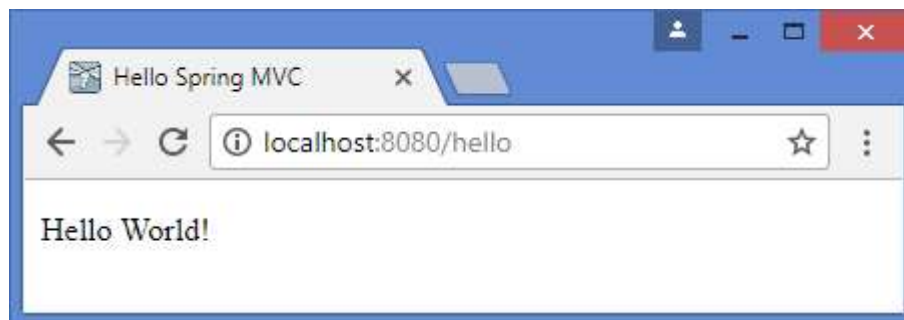
# SpringMVC example

hello.html

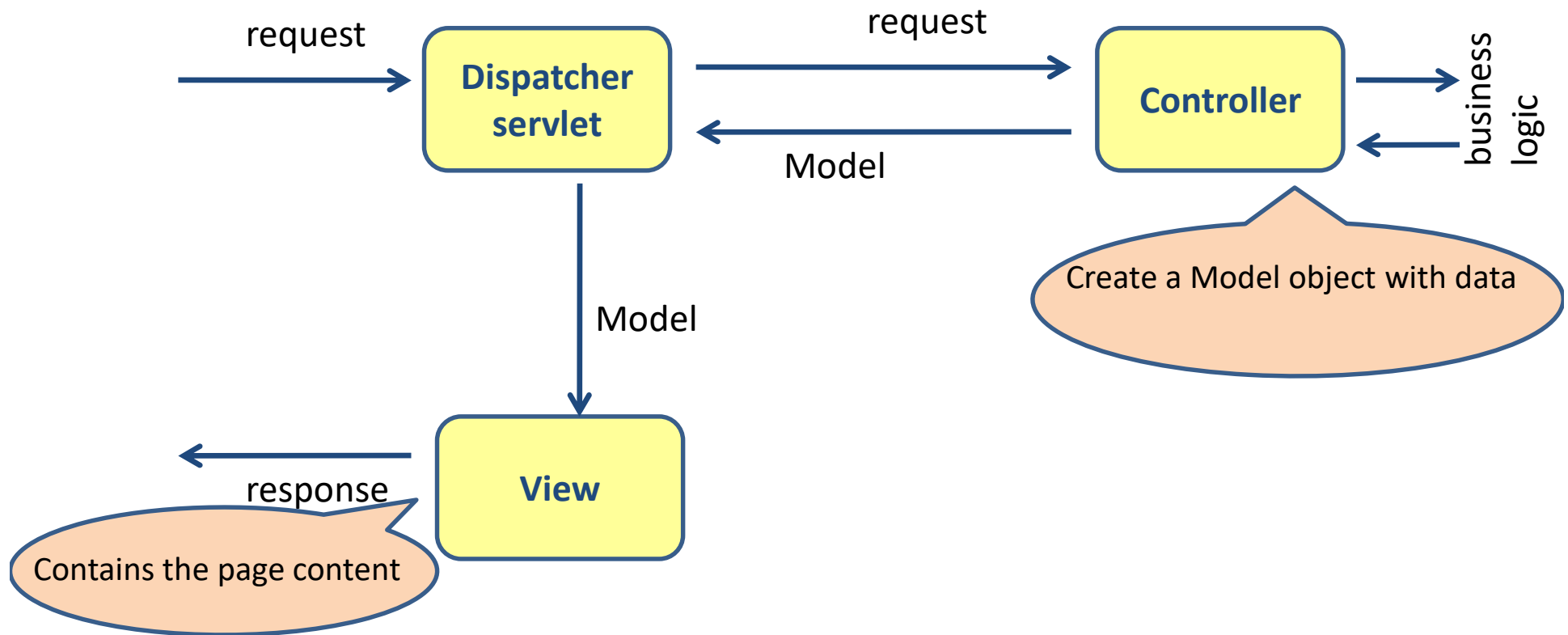
```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <title>Hello Spring MVC</title>
</head>
<body>
  <p th:text="'Hello ' + ${name}+ '!'" />
</body>
</html>
```

Thymeleaf namespace

Get the "name" from the model



# Spring MVC framework



# ModelAndView

```
@Controller
public class HelloController {
    @RequestMapping("/hello")
    public ModelAndView hello(@RequestParam(value="name", required=false,
        defaultValue="World") String name) {
        Map<String, Object> params = new HashMap<>();
        params.put("name", name);
        return new ModelAndView("hello", params);
    }
}
```

Model is a Map

Return a ModelAndView class

# Mapping annotations

---

```
@RequestMapping(value = "/add")
```

```
@RequestMapping(value = "/add", method = RequestMethod.GET)
```

```
@GetMapping("/add")
```

```
@RequestMapping(value = "/add", method = RequestMethod.POST)
```

```
@PostMapping("/add")
```

Same

Same

# Externalizing text

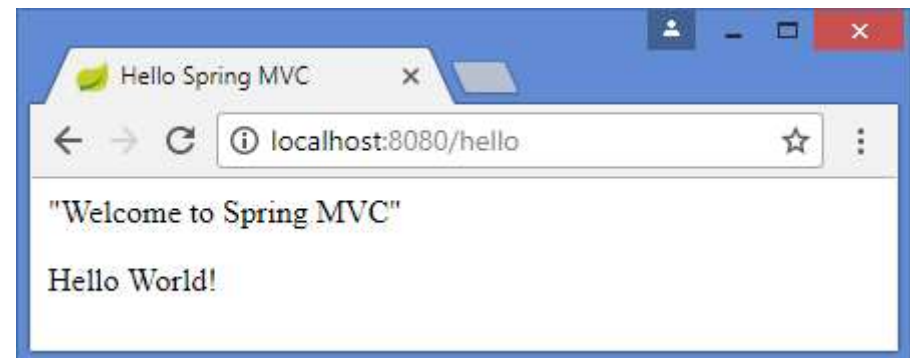
- Spring MVC uses the file **messages.properties** for text in Thymeleaf.

messages.properties

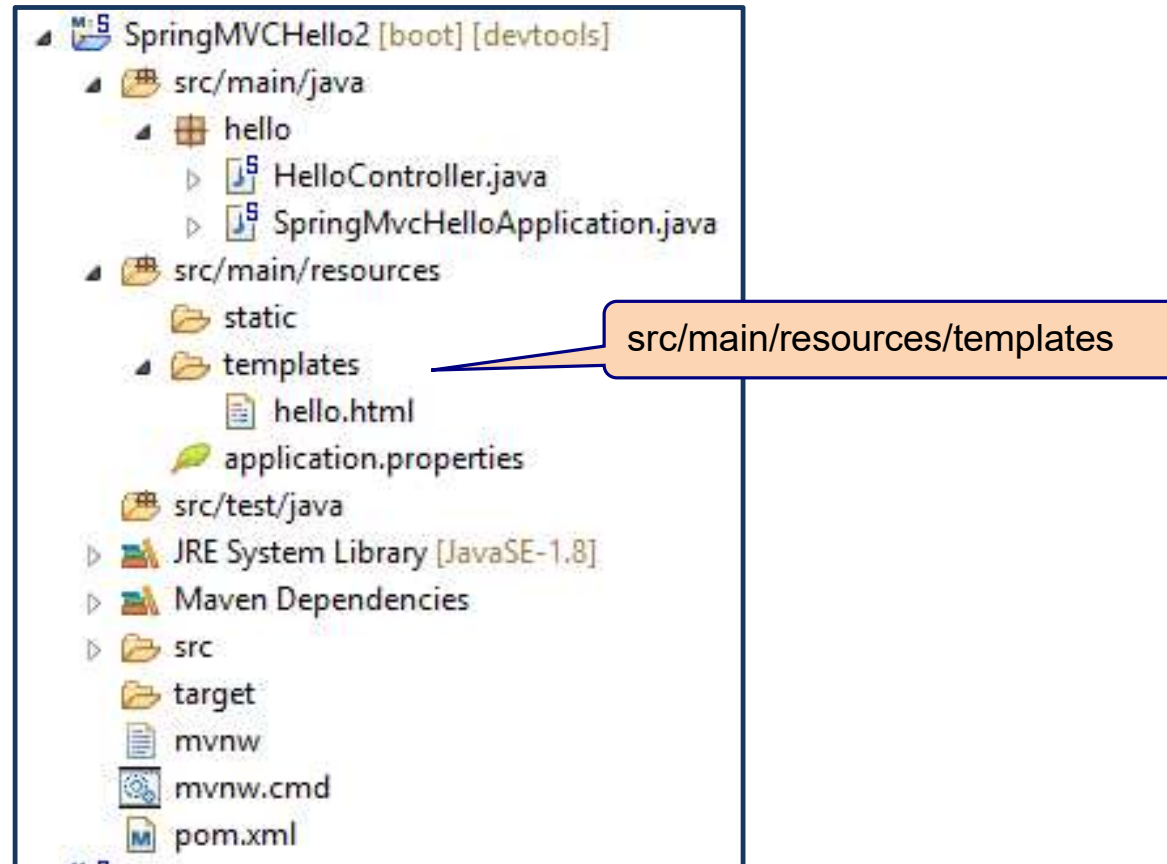
```
mainpage.header="Welcome to Spring MVC"
```

hello.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <title>Hello Spring MVC</title>
</head>
<body>
  <span th:text="#{mainpage.header}">header</span>
  <br/>
  <p th:text="'Hello ' + ${name}+'!'" />
</body>
</html>
```



# Default place for thymeleaf templates



# Set the place for thymeleaf templates

src/main/resources/webfiles

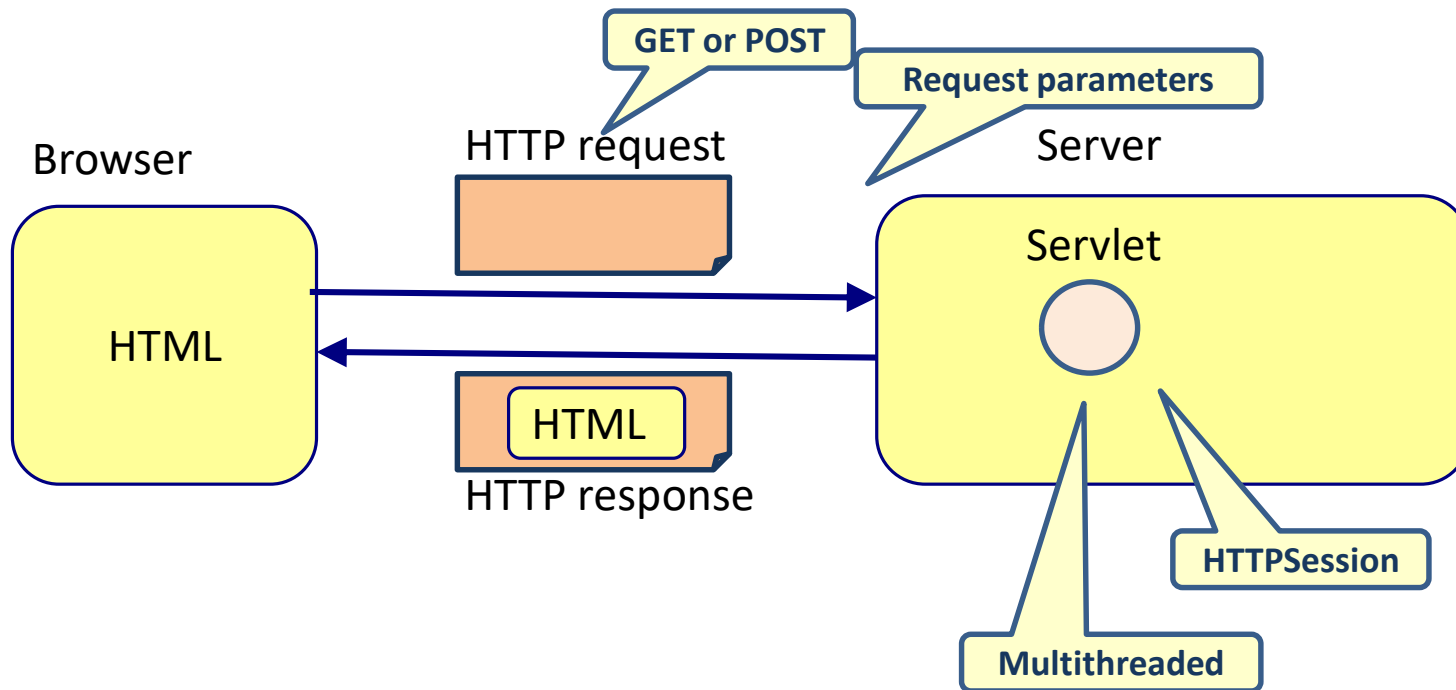
application.properties

```
spring.thymeleaf.check-template-location=true  
spring.thymeleaf.prefix=classpath:/webfiles/  
spring.thymeleaf.suffix=.html
```

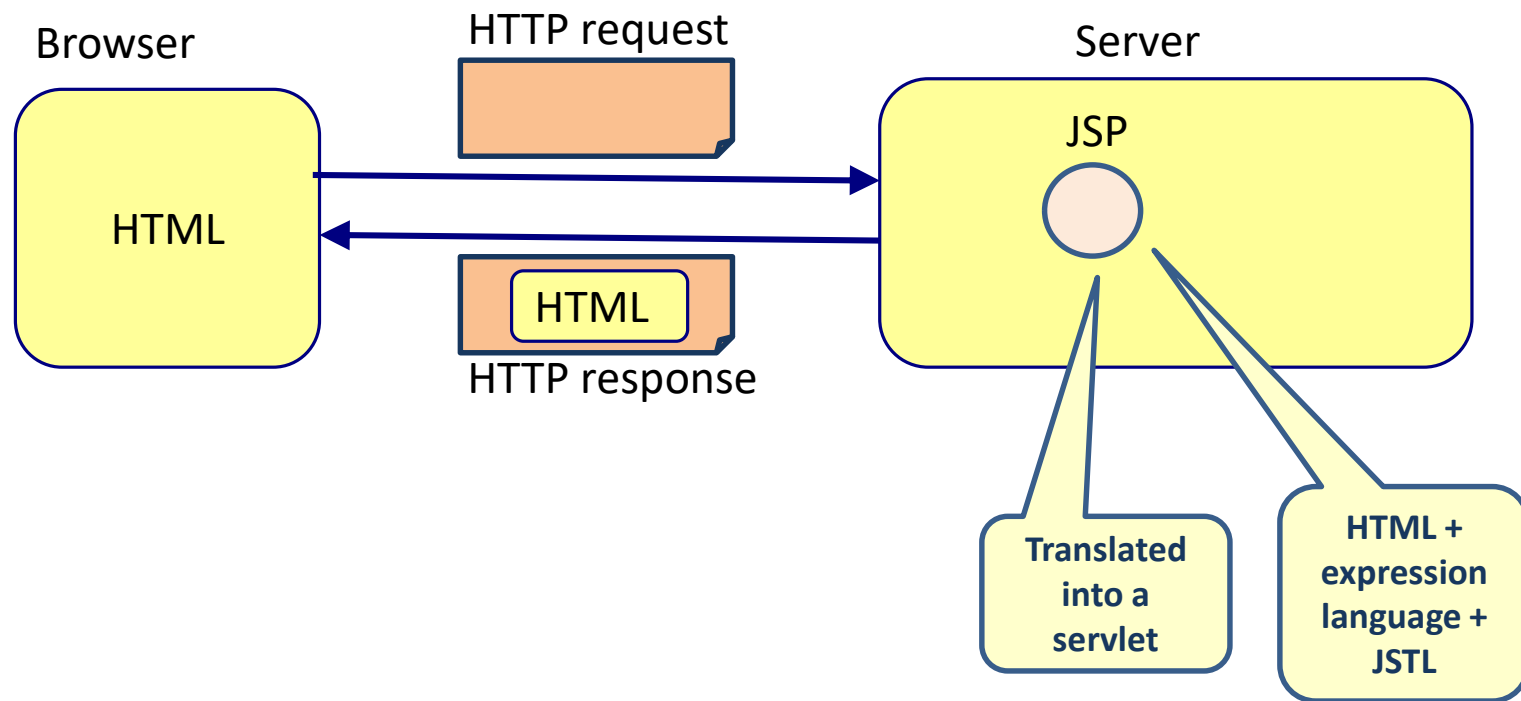


# SUMMARY

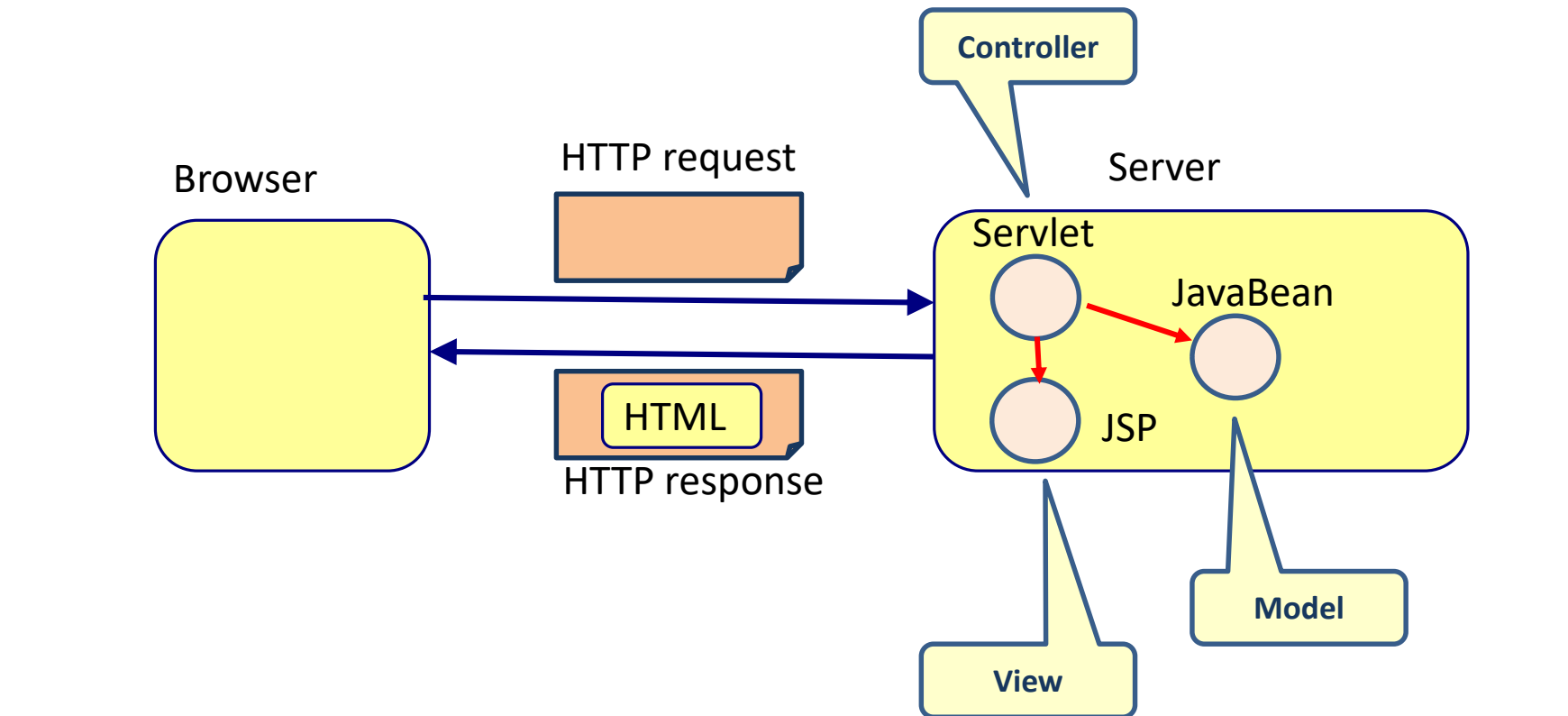
# Servlet



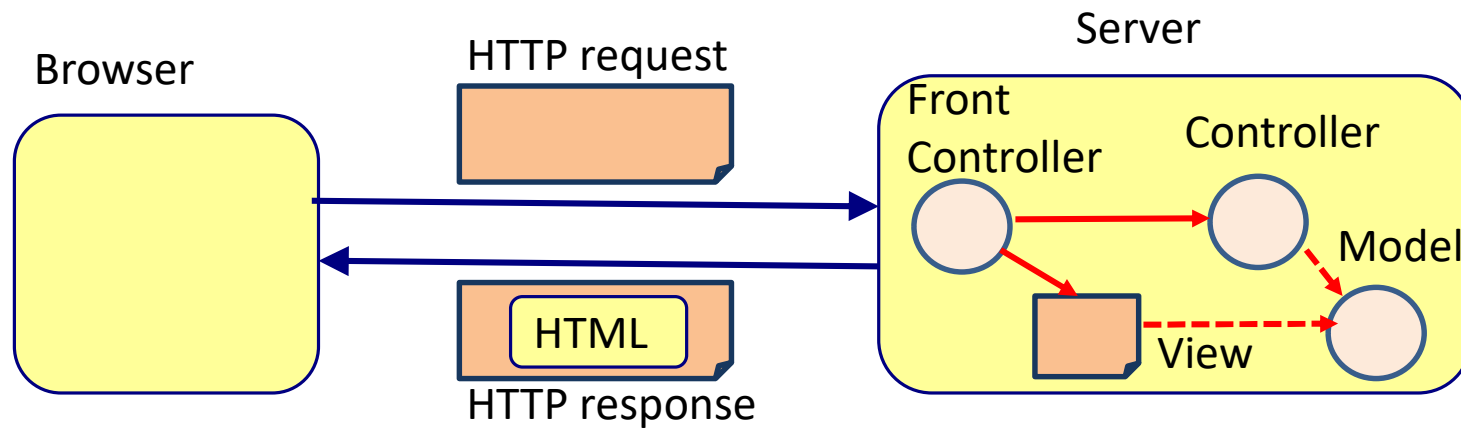
# JSP



# MVC



# Front controller



# Spring MVC with Spring boot

---

- Spring boot
  - Convention over configuration
  - Embedded web container
    - Tomcat
- Spring MVC
  - Build-in front controller
  - @Controller
  - @RequestMapping
  - @RequestParam
  - ModelAndView class
  - Thymeleaf templates

# Connecting the parts of knowledge with the wholeness of knowledge

---

1. A servlet is the basic building block of web applications in Java
2. The separation of Model-View-Controller gives structure to the web application according design best practices

- 
3. **Transcendental consciousness** is the natural experience pure consciousness, the home of all the laws of nature.
  4. **Wholeness moving within itself:** In unity consciousness, one appreciates and enjoys the underlying blissful nature of life even in all the abstract expressions of pure consciousness.

