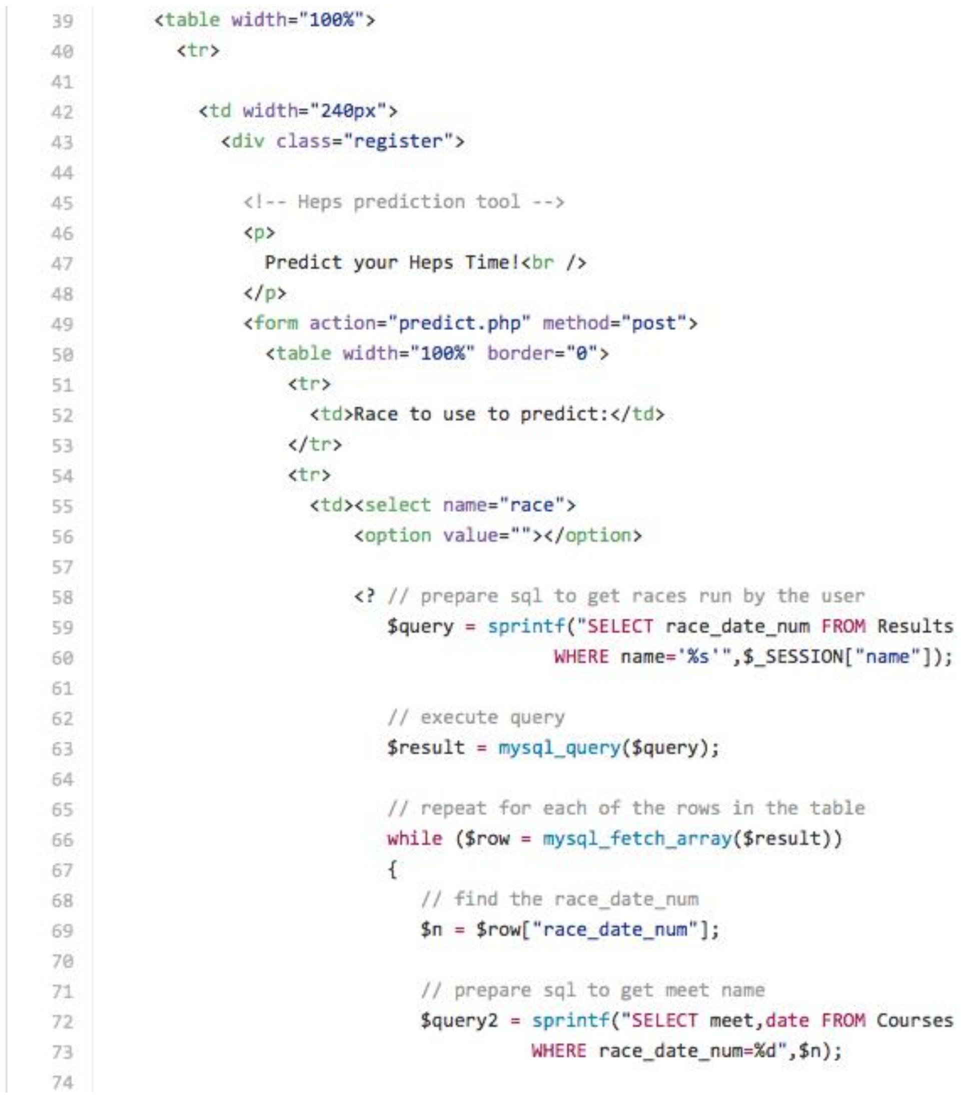
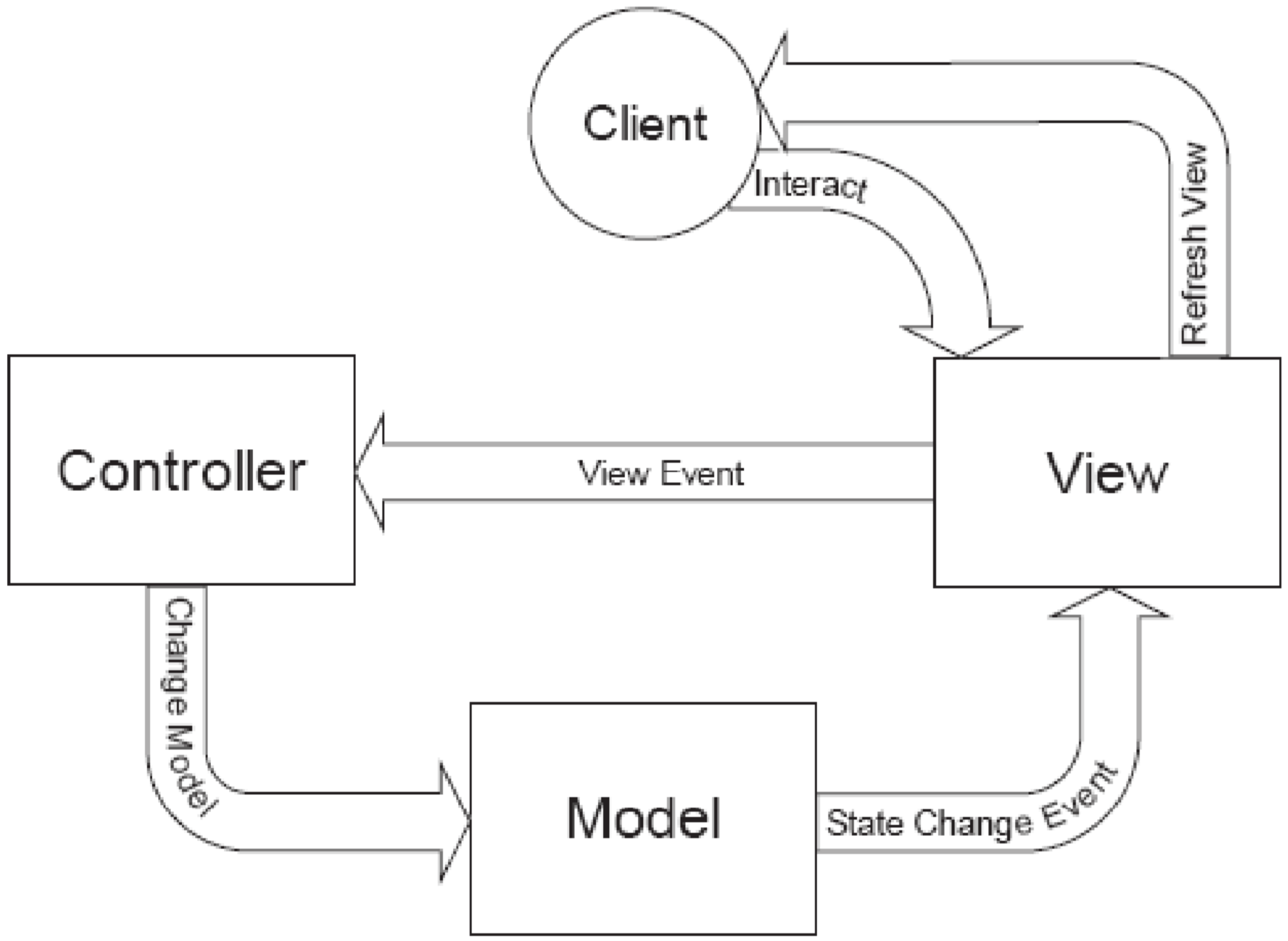
WEB- MVC FRAMEWORKS

View-Model Mix

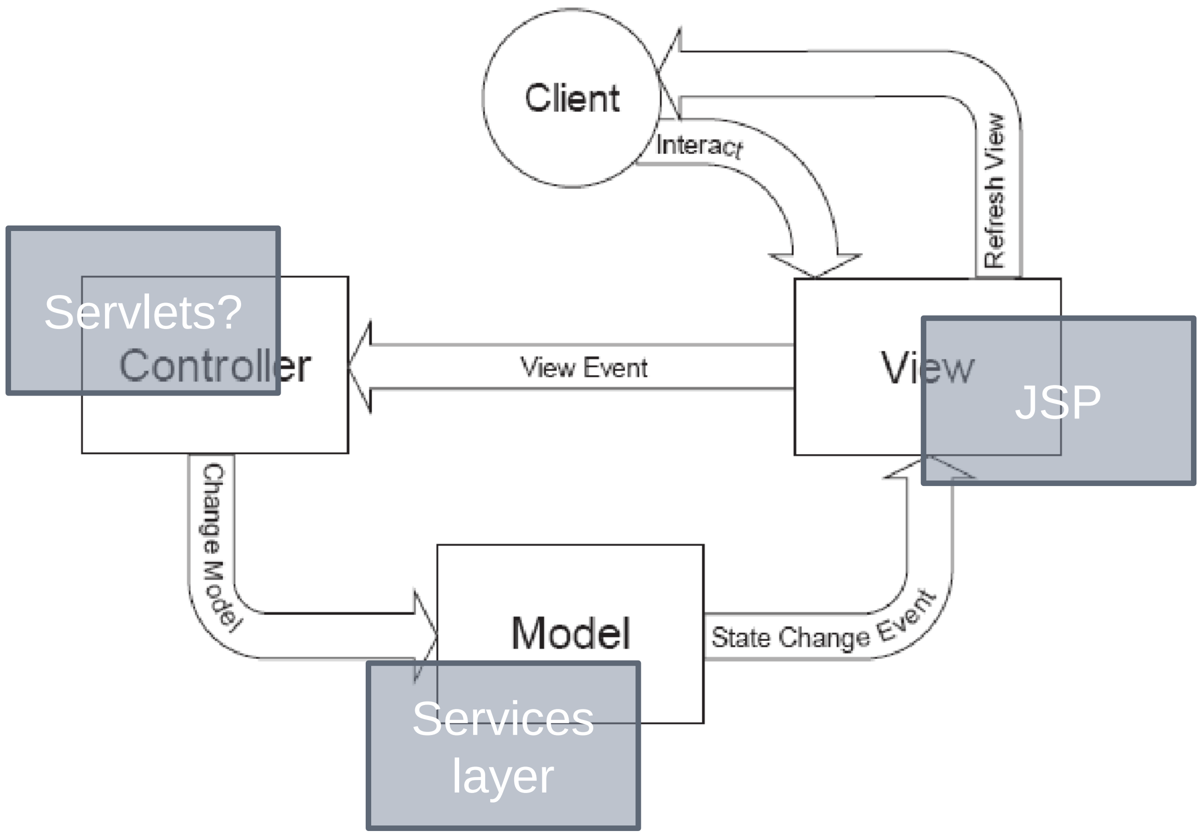


```
<table width="100%">
 <tr>

  <td width="240px">
   <div class="register">

    <!-- Heps prediction tool -->
    <p>
     Predict your Heps Time!<br />
    </p>
    <form action="predict.php" method="post">
     <table width="100%" border="0">
      <tr>
       <td>Race to use to predict:</td>
      </tr>
      <tr>
       <td><select name="race">
         <option value=""></option>

         <? // prepare sql to get races run by the user
            $query = sprintf("SELECT race_date_num FROM Results
                  WHERE name='%s'",$_SESSION["name"]);

            // execute query
            $result = mysql_query($query);

            // repeat for each of the rows in the table
            while ($row = mysql_fetch_array($result))
            {
              // find the race_date_num
              $n = $row["race_date_num"];

              // prepare sql to get meet name
              $query2 = sprintf("SELECT meet,date FROM Courses
                   WHERE race_date_num=%d",$n);

```

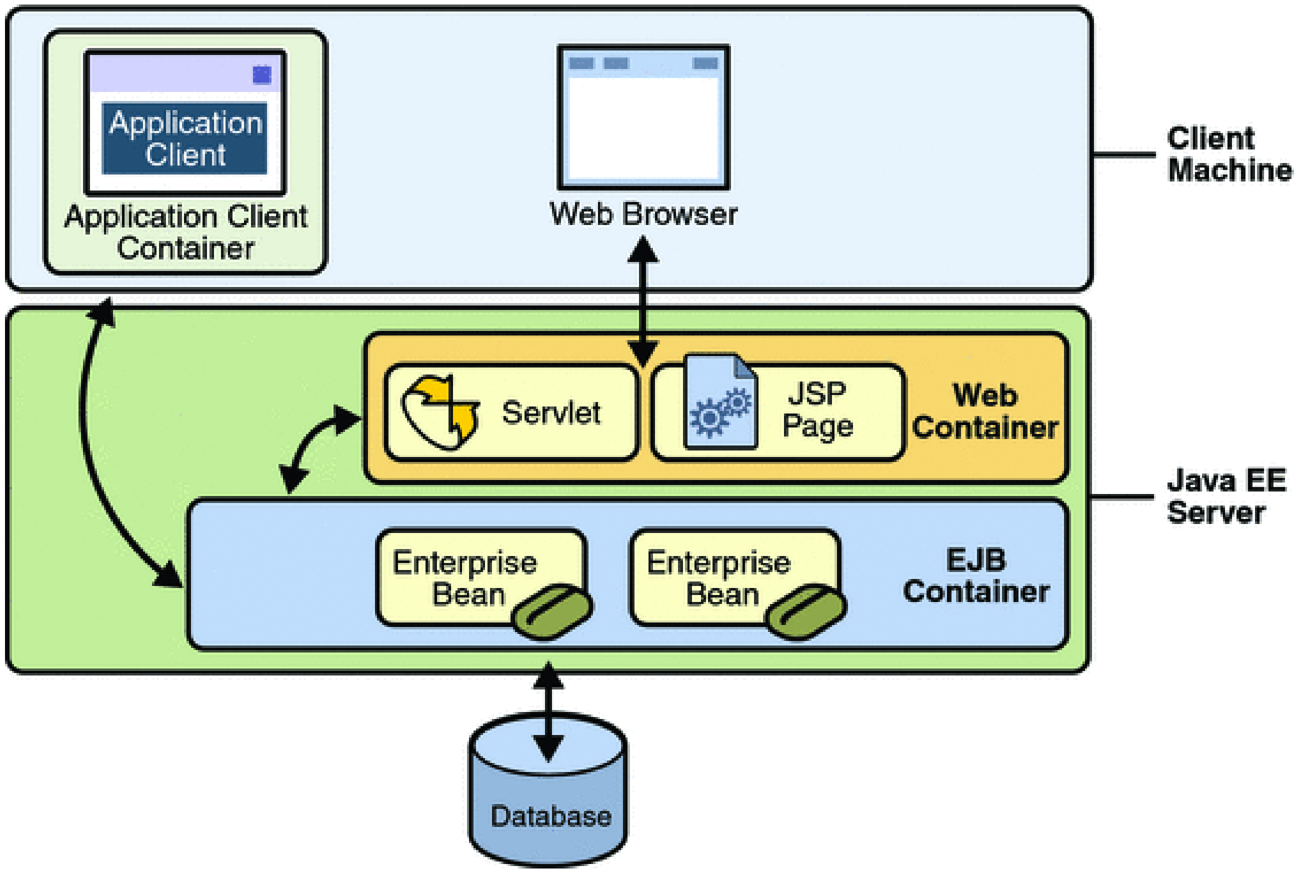
MVC Meta-pattern



Web-based (JEE) MVC?



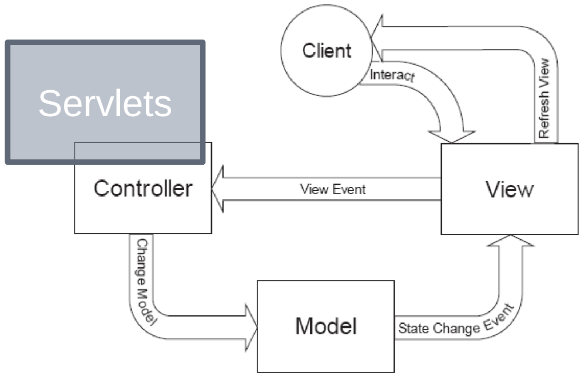
Web-based (JEE) MVC?



Controller-servlet

The old-way:

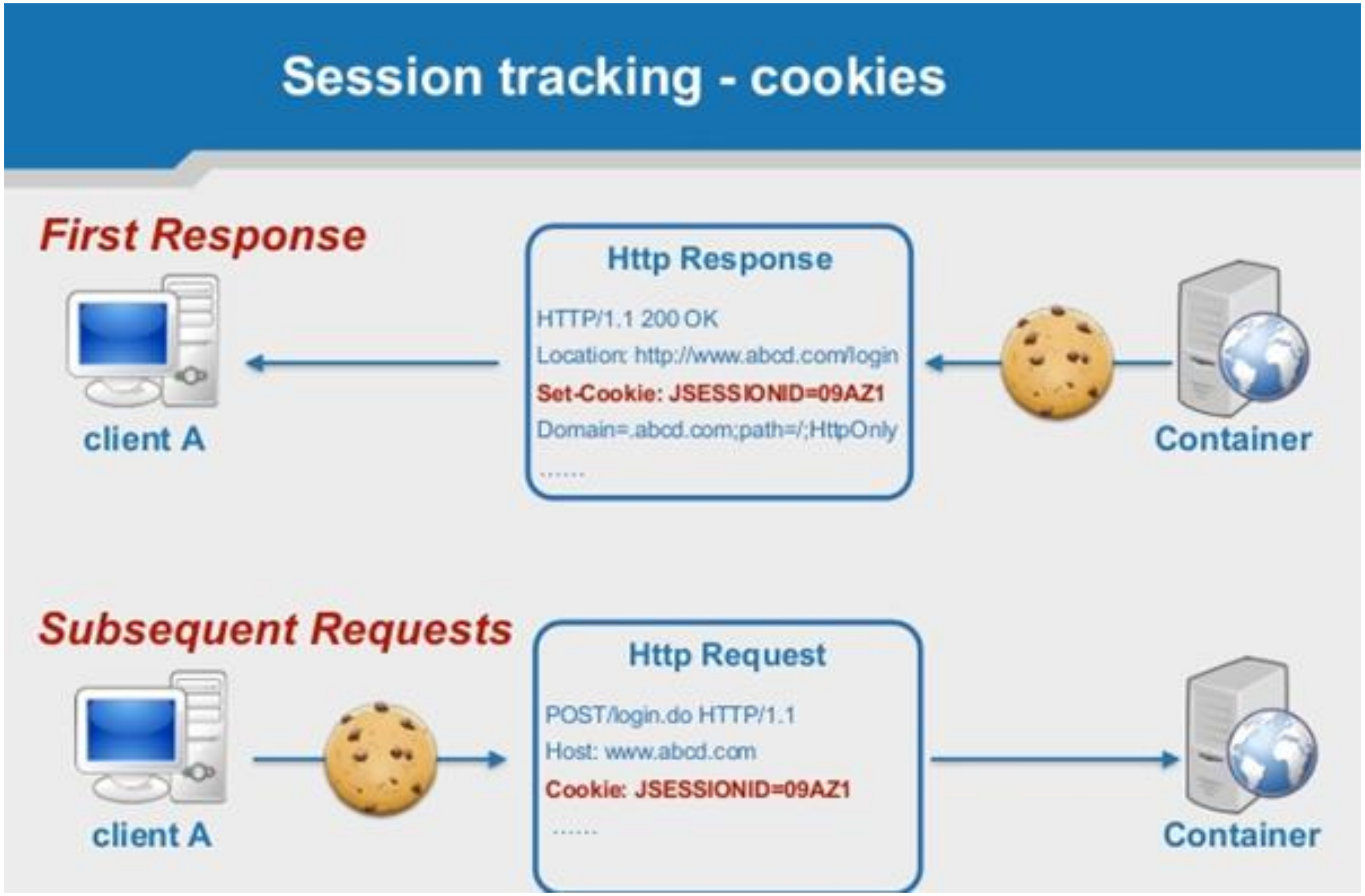
```
protected void doPost(HttpServletRequest request, HttpServletResponse response) {
  String v1=request.getParameter("p1");
  String v2=request.getParameter("p2");

  Integer answer=Services.getInstance().process(convert(v1),convert(v2));

  request.setAttribute("answerObject", answer);

  this.getServletContext().getRequestDispatcher("").forward(request, response);
}
```



Controller-servlet

- The old-way:
  - Get request parameters (you must know their names!).
  - Convert and process with such parameters (you must know their types!).
  - You must decide how to keep the state between page transitions: cookies?, session objects?, request objects?, URL-rewriting?, hidden-form-fields?...
  - The servlet (the controller) must know the next page to redirect!

Session tracking



```
HTTP/1.1 200 OK
Location: http://www.abcd.com/login
Set-Cookie: JSESSIONID=09AZ1
Domain=.abcd.com;path=/;HttpOnly
......
```

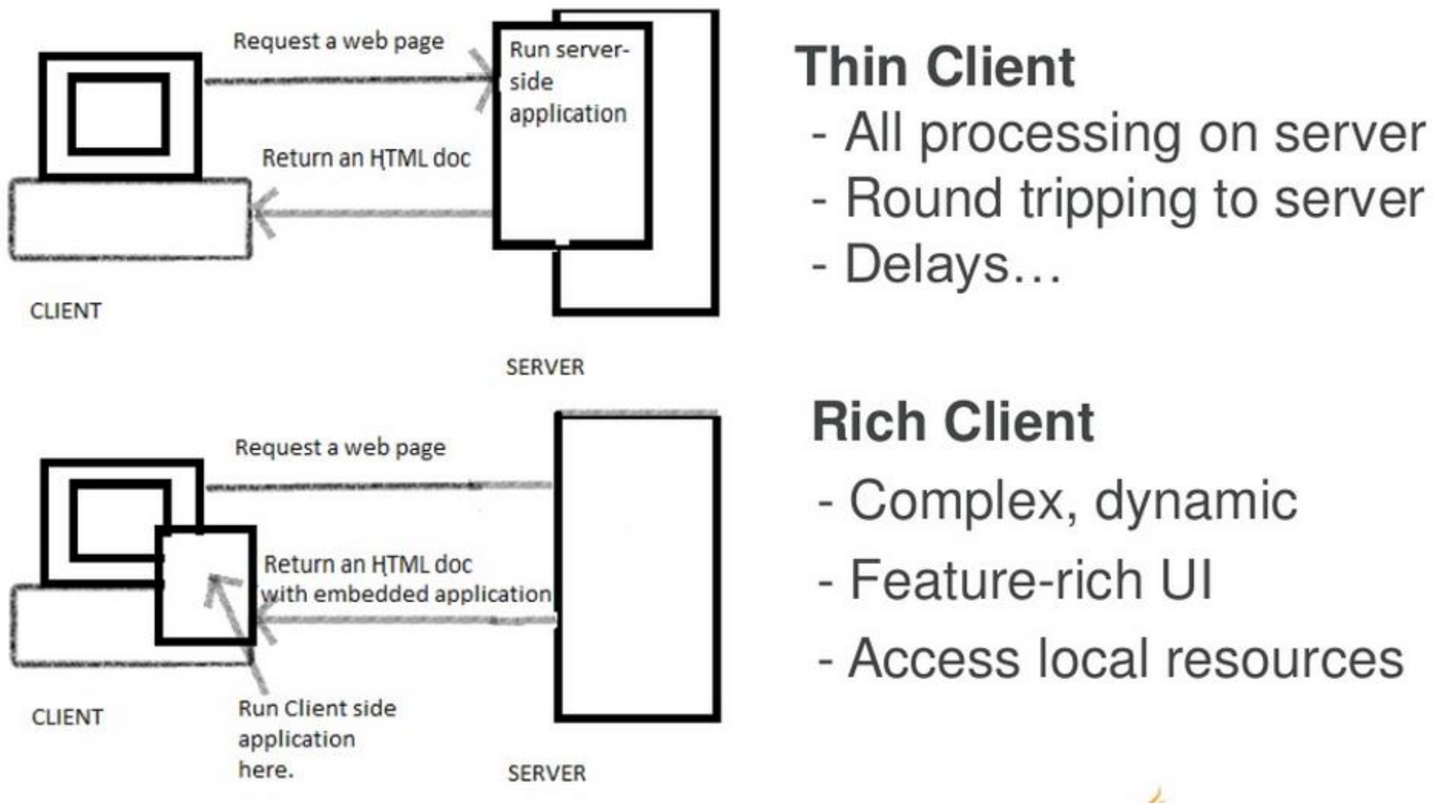
```
POST/login.do HTTP/1.1
Host: www.abcd.com
Cookie: JSESSIONID=09AZ1
......
```

KISS Principle

KEEP IT SIMPLE, STUPID (AND SHORT) ·
KISS

Web-MVC frameworks

- Rich client vs. thin clients.


Request a web page
Run server-side application
Return an HTML doc
CLIENT
SERVER
Request a web page
Return an HTML doc with embedded application
CLIENT
Run Client side application here.
SERVER

Thin Client
- All processing on server
- Round tripping to server
- Delays…

Rich Client
- Complex, dynamic
- Feature-rich UI
- Access local resources

Web - MVC

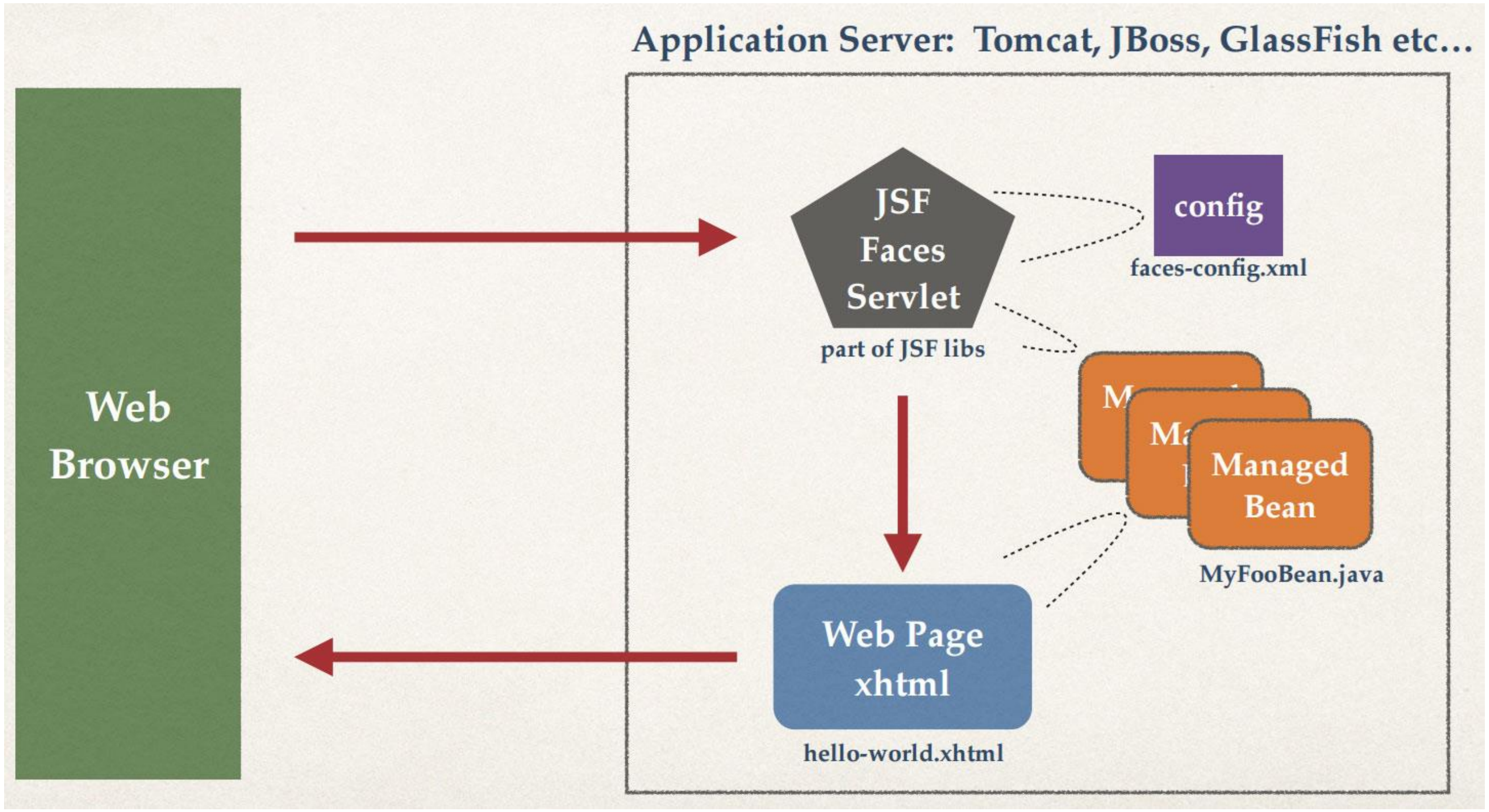
- Thin client / MVC frameworks:
  - Java server faces
  - Struts
  - Vaadin
  - SpringMVC

- Rich client / MVC frameworks:
  - Angular.js
  - React.js
  - Ember.js

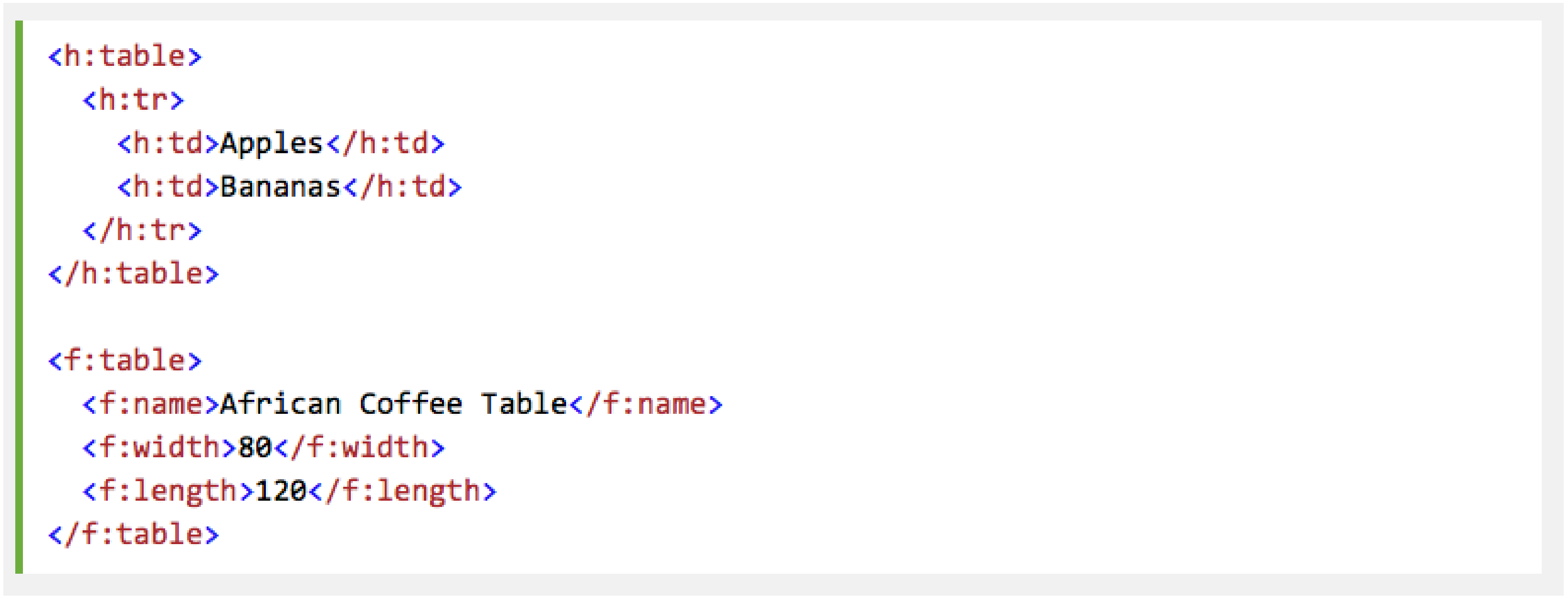
JSF – Java Server Faces

- MVC framework based on JavaBeans/POJO.
- Encapsulates/handles:
  - Request handling.
  - Parameters processing/type conversion.
  - Validations.
  - Application navigation.
- Provides:
  - Large set of custom-tags

JSF – Java Server Faces

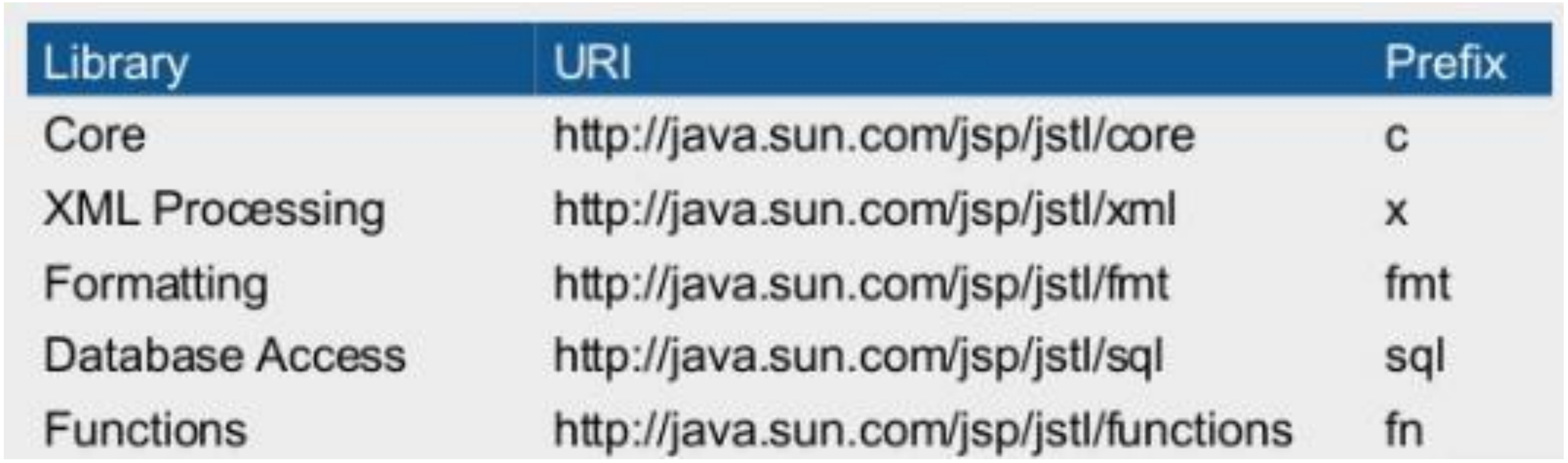


XML Name-Spaces



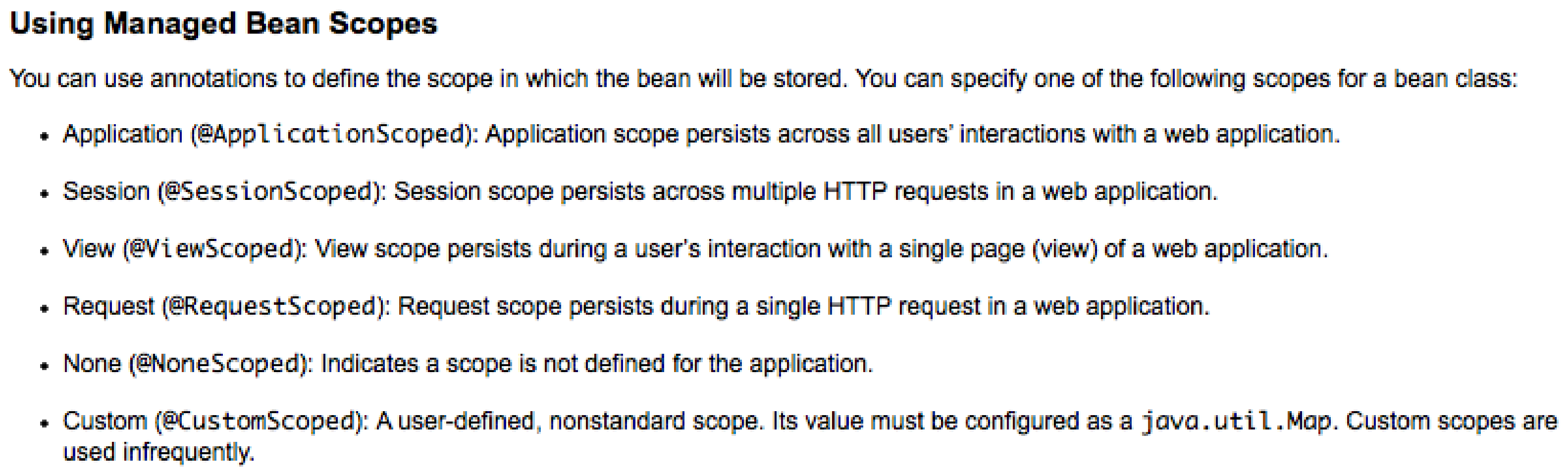
```
<h:table>
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table>
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
```

JSF- Tag Libs



Library	URI	Prefix
Core	http://java.sun.com/jsp/jstl/core	c
XML Processing	http://java.sun.com/jsp/jstl/xml	x
Formatting	http://java.sun.com/jsp/jstl/fmt	fmt
Database Access	http://java.sun.com/jsp/jstl/sql	sql
Functions	http://java.sun.com/jsp/jstl/functions	fn

ManagedBean: scopes



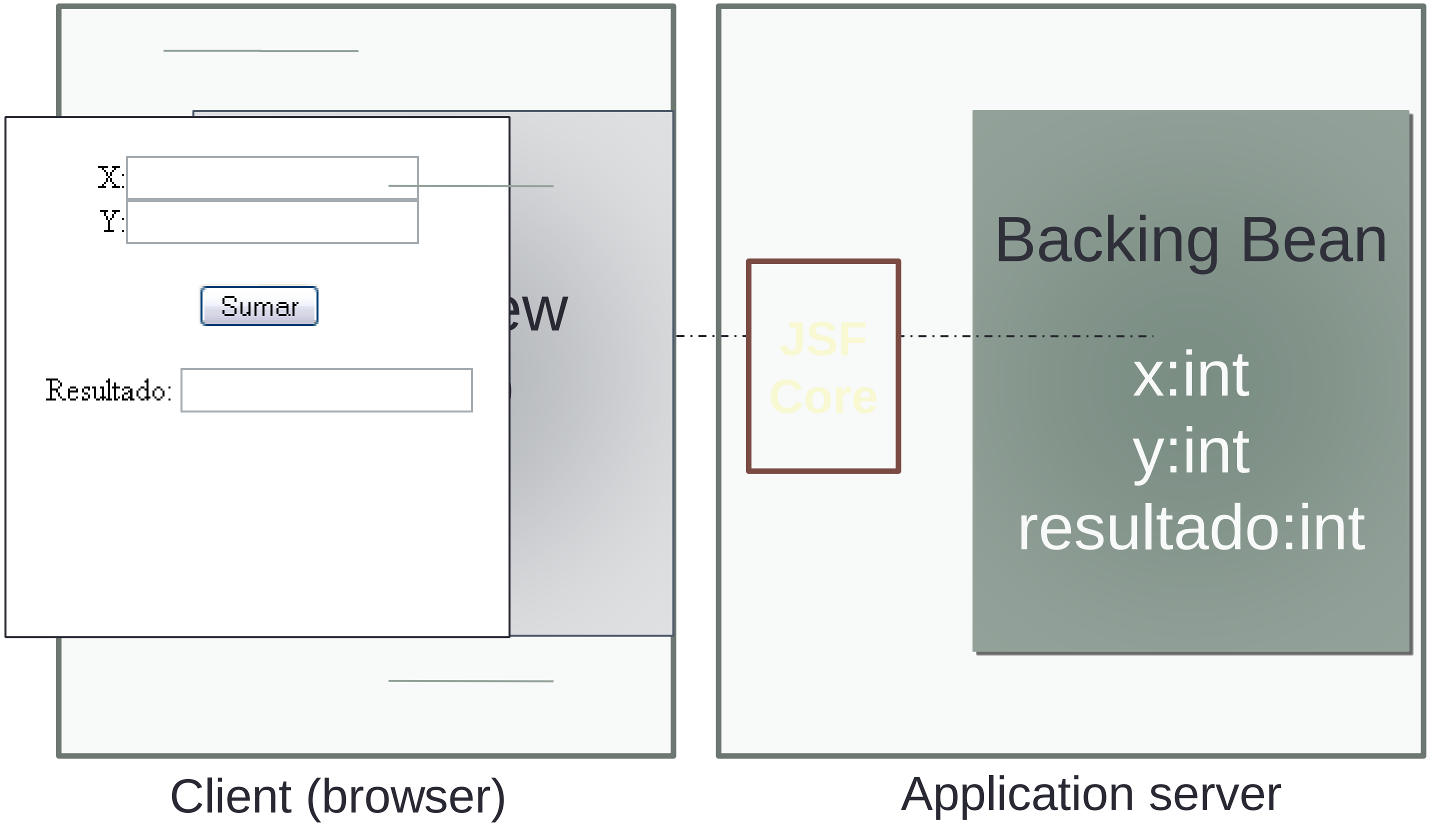
Using Managed Bean Scopes

You can use annotations to define the scope in which the bean will be stored. You can specify one of the following scopes for a bean class:

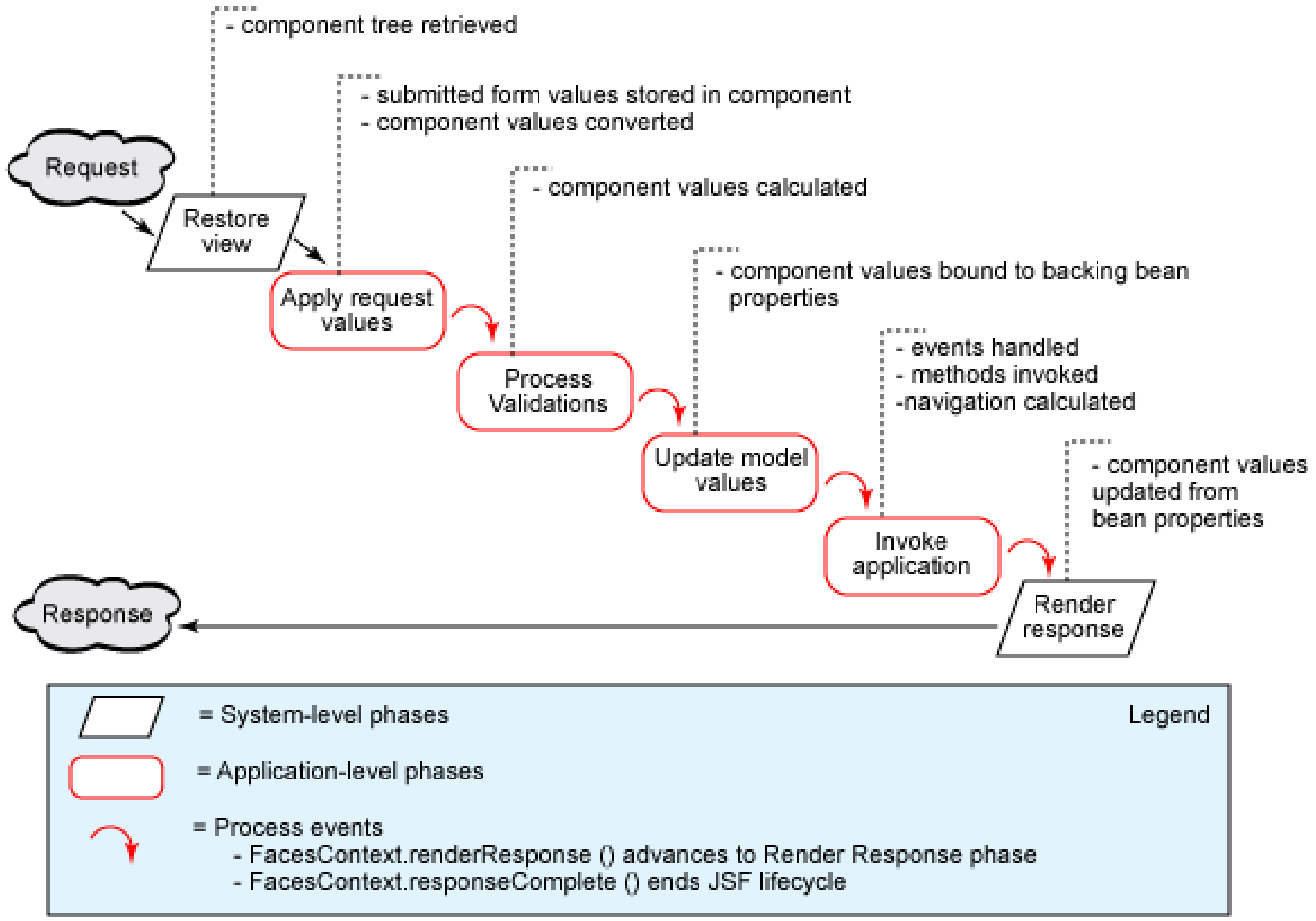
- Application (`@ApplicationScoped`): Application scope persists across all users' interactions with a web application.
- Session (`@SessionScoped`): Session scope persists across multiple HTTP requests in a web application.
- View (`@ViewScoped`): View scope persists during a user's interaction with a single page (view) of a web application.
- Request (`@RequestScoped`): Request scope persists during a single HTTP request in a web application.
- None (`@NoneScoped`): Indicates a scope is not defined for the application.
- Custom (`@CustomScoped`): A user-defined, nonstandard scope. Its value must be configured as a `java.util.Map`. Custom scopes are used infrequently.

JSF – Java Server Faces

- JavaBean in a JSF context: Backing Bean



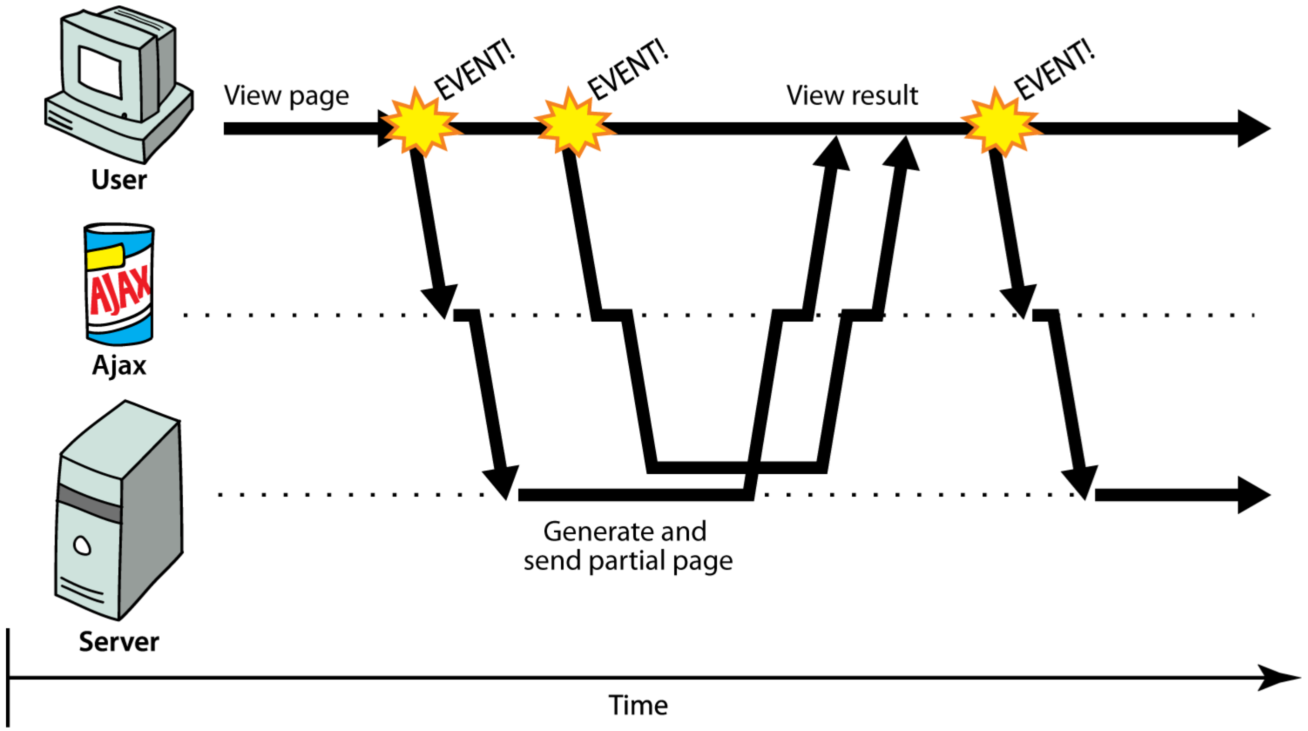
JSF life cycle



- component tree retrieved
- submitted form values stored in component
- component values converted
- component values calculated
- component values bound to backing bean properties
- events handled
- methods invoked
-navigation calculated
- component values updated from bean properties
Request
Restore view
Apply request values
Process Validations
Update model values
Invoke application
Render response
Response
= System-level phases
Legend
= Application-level phases
= Process events
- FacesContext.renderResponse () advances to Render Response phase
- FacesContext.responseComplete () ends JSF lifecycle

JSF life cycle

application
Render response
Response


User
Ajax
Server
View page
EVENT!
EVENT!
View result
EVENT!
Generate and send partial page
Time