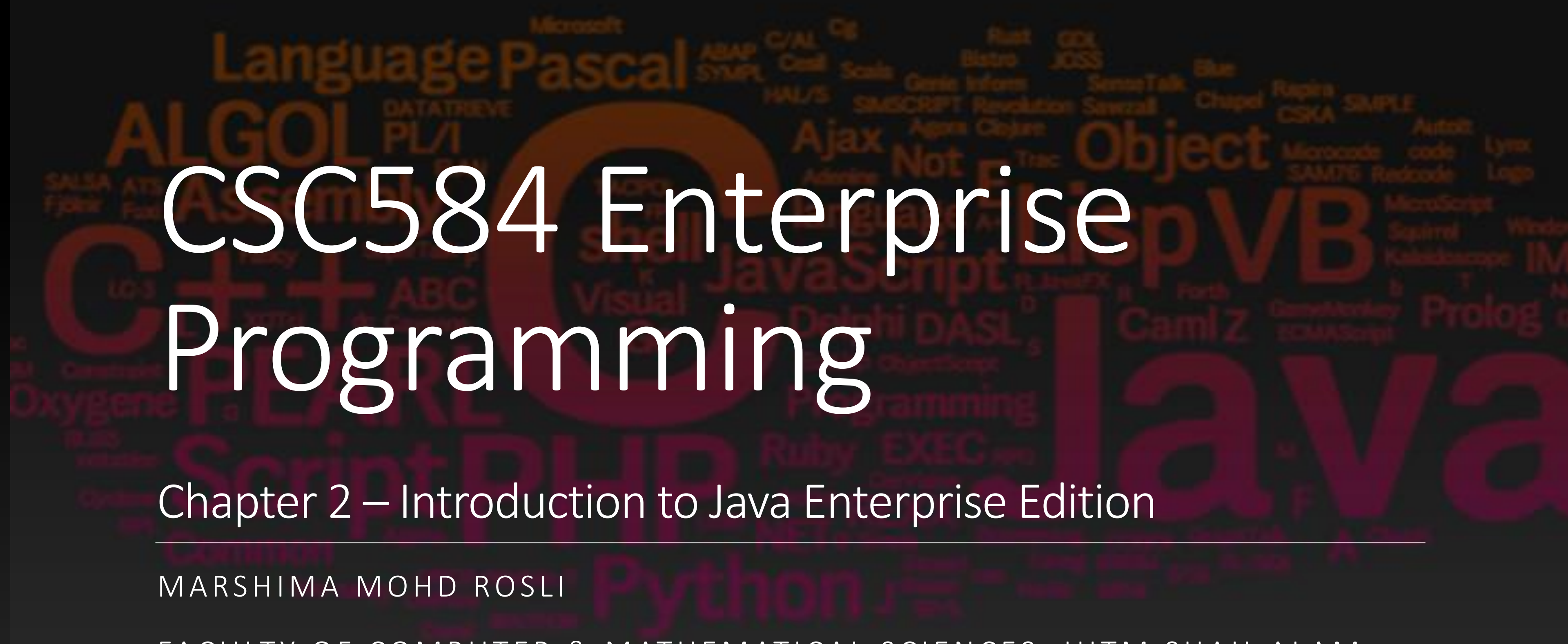


CSC584 Enterprise Programming

Chapter 2 – Introduction to Java Enterprise Edition

MARSHIMA MOHD ROSLI

FACULTY OF COMPUTER & MATHEMATICAL SCIENCES, UITM SHAH ALAM

Chapter Outline

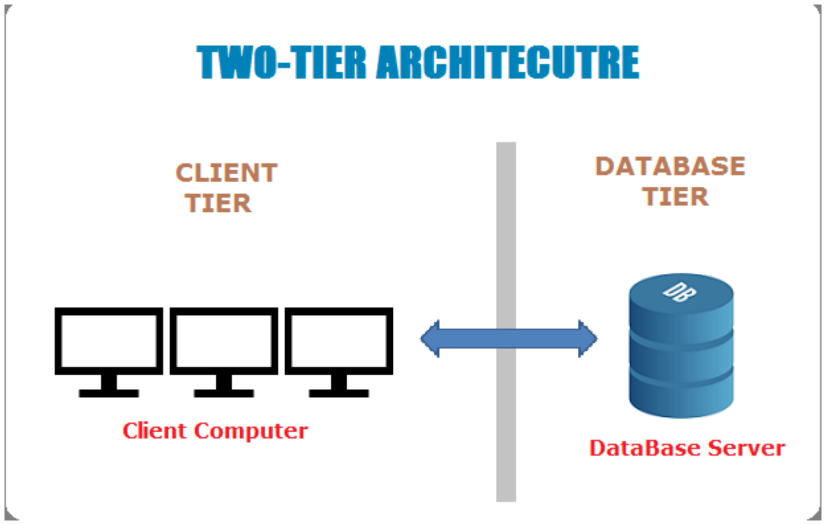
- Overview of Java EE Platform
- Role of Application Servers
- Java EE Architecture (MVC, Spring)
- Java EE Patterns
- Java EE Components (Web components, EJB)

Past History

Initially two tier architecture (client server applications)

Client is responsible for data access applying business logic and presentation of data

Only service provided by Server was that of database server.



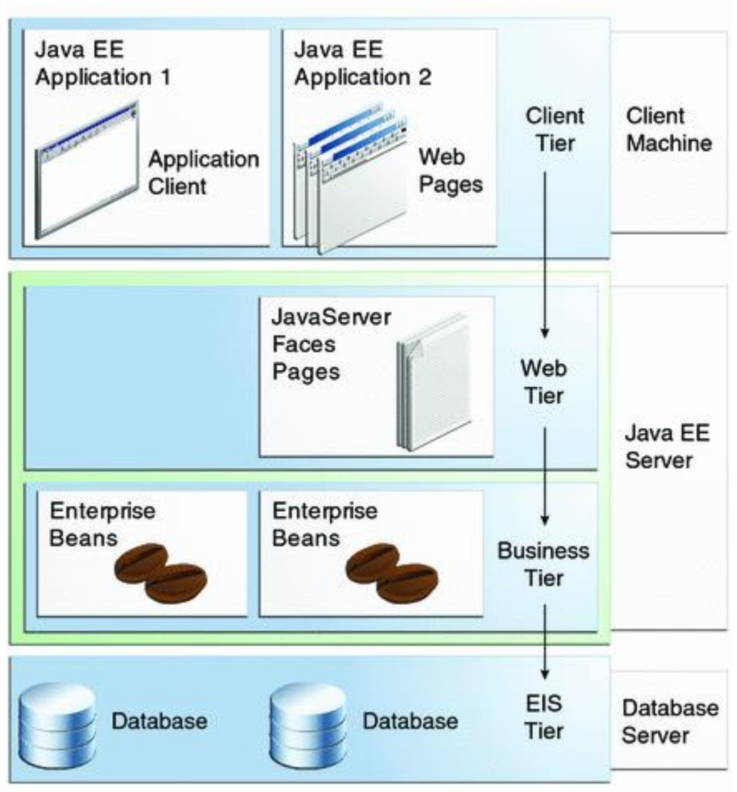
Drawbacks

- Easy to deploy but difficult to enchance or upgrade.
- It makes reuse of business and presentation logic difficult
- Not scalable and not suited for internet

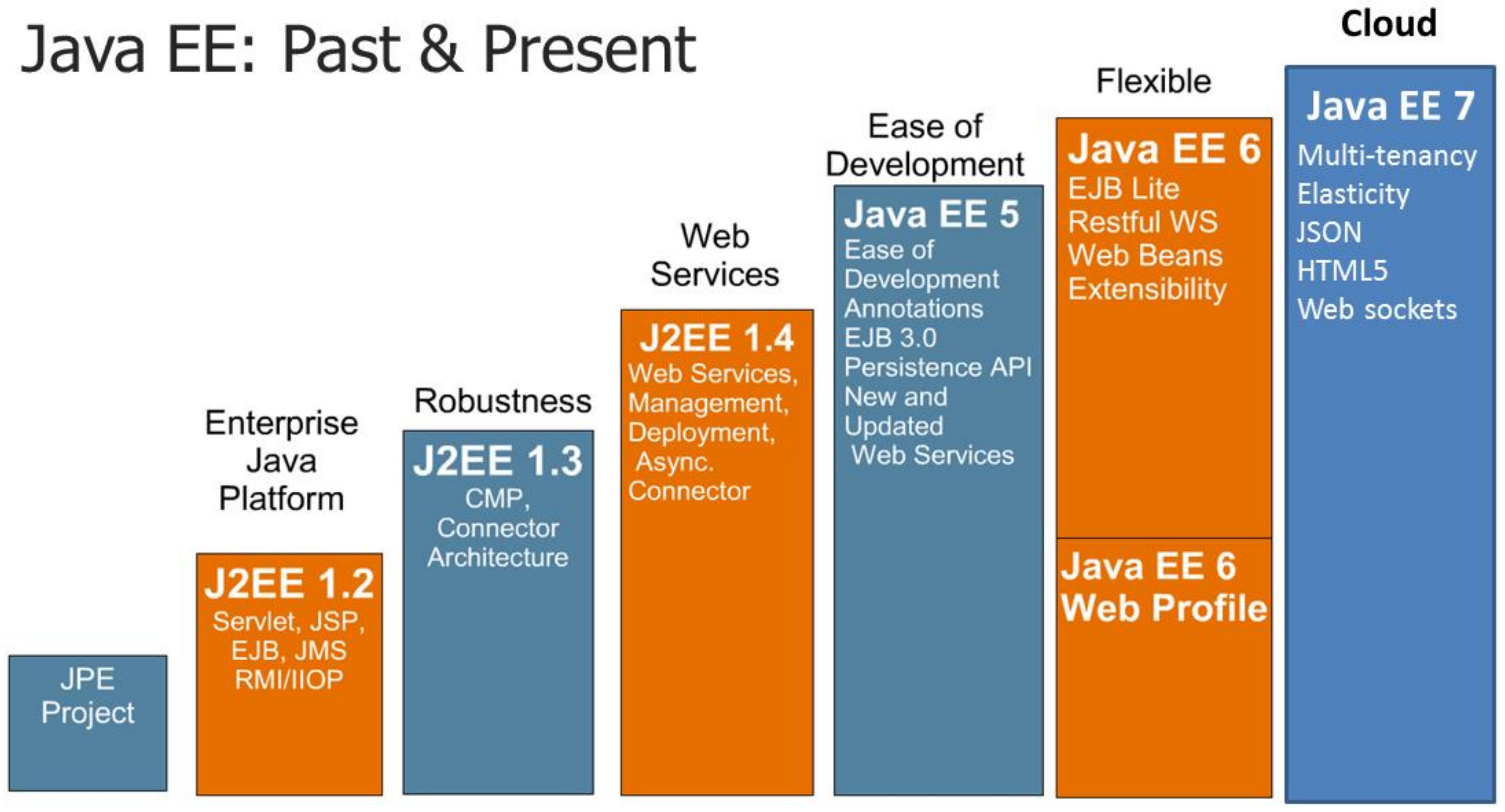
Java EE Platform

Java EE is an architecture for implementing enterprise class applications using Java and Internet Technology

- Solves problems of two tier architecture



Overview of Java EE Platform


Java EE: Past & Present
JPE Project
Enterprise Java Platform
J2EE 1.2
Servlet, JSP, EJB, JMS RMI/IIOP
Robustness
J2EE 1.3
CMP, Connector Architecture
Web Services
J2EE 1.4
Web Services, Management, Deployment, Async. Connector
Ease of Development
Java EE 5
Ease of Development Annotations EJB 3.0 Persistence API New and Updated Web Services
Flexible
Java EE 6
EJB Lite Restful WS Web Beans Extensibility
Java EE 6 Web Profile
Cloud
Java EE 7
Multi-tenancy Elasticity JSON HTML5 Web sockets

Java Platform

- A Java platform comprises the **JVM** together with **supporting class libraries**.

Java 2 Standard Edition (J2SE)

- (1999) provides core libraries for data structures, xml parsing, security, internationalization, db connectivity, RMI

Java 2 Platform, Enterprise Edition (J2EE)

- provides more class libraries for servlets, JSPs, Enterprise Java Beans, advanced XML

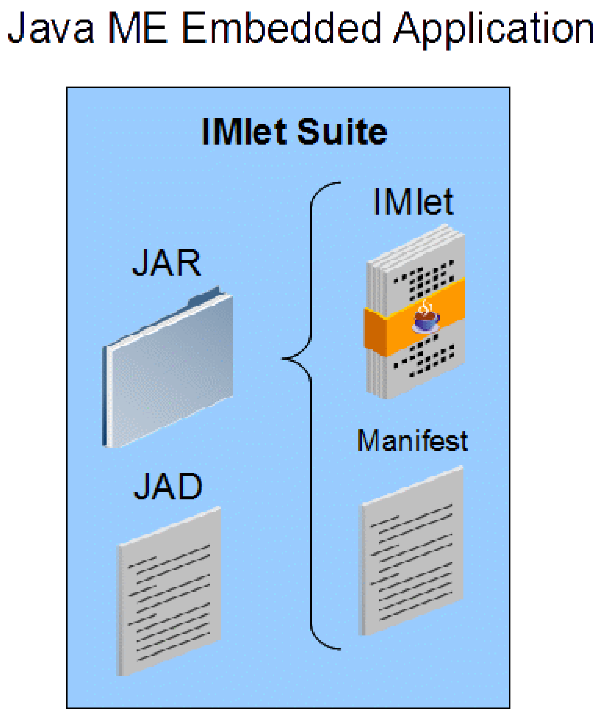
Java Platform, Enterprise Edition (Java EE)

- When Java Platform 5.0 was released (2004) the ‘2’ was dropped from these titles.

Java Platform

Java Micro Edition (Java ME)

- comprises the necessary core libraries and tools for writing Java for embedded systems and other small footprint platforms, along with some **specialised libraries** for specific types of device such as **mobile phones**.



Java Web Application

A **Java web application** generates **interactive web pages** containing various types of markup language (**HTML**, **XML**, and so on) and **dynamic content**.

It is typically comprised of web components such as:

- **JavaServer Pages (JSP)**
- **Servlets**
- **JavaBeans**

to **modify** and temporarily **store data**, **interact with databases** and **web services**, and **render content** in response to **client requests**.

Java EE (Enterprise Edition)

Java EE (Enterprise Edition) is a widely used **platform** containing a **set of coordinated technologies** that significantly reduce the cost and complexity of:

- **developing**
- **deploying and**
- **managing**

Java EE 6 is supported only by the GlassFish server v3.x.

multitier, server-centric applications.

Java EE builds upon the Java SE platform and **provides a set of APIs** (application programming interfaces) for developing and running portable, robust, scalable, reliable and secure server-side applications.

Java EE Platform

- The Java EE platform uses a simplified programming model.
- **XML deployment descriptors** are **optional**. Instead, a developer can simply enter the information as an **annotation** directly into a Java source file, and the **Java EE server** will configure the component at deployment and runtime
- With **annotations**, you put the specification information in your code next to the program element affected.

Java EE application model

- an architecture for implementing **services as multitier applications** that deliver the scalability, accessibility, and manageability needed by enterprise-level applications.
- With this structure you can more easily change one of the tiers without compromising your entire application.
- **Business and presentation logic** - to be implemented by the **developer**
- **Standard system services** – to be provided by the **Java EE platform**

Role of Application Servers

- A Java EE server is **a server application** that the implements the Java EE platform APIs and provides the standard Java EE services.
- Java EE servers are sometimes called **application servers**, because they allow you to serve application data to clients, much as how web servers serve web pages to web browsers.
- The Java EE server provides services to these components in the form of a container.

Java EE Containers

• are the **interface** between a **Java component** and the **low-level platform-specific functionality** (*i.e.* **transaction and state management, multithreading, resource pooling, etc.**) that supports the component.

• provide for the separation of **business logic** from **resource** and **lifecycle management.**

• this allows developers to focus on writing business logic rather than writing **enterprise infrastructure**.

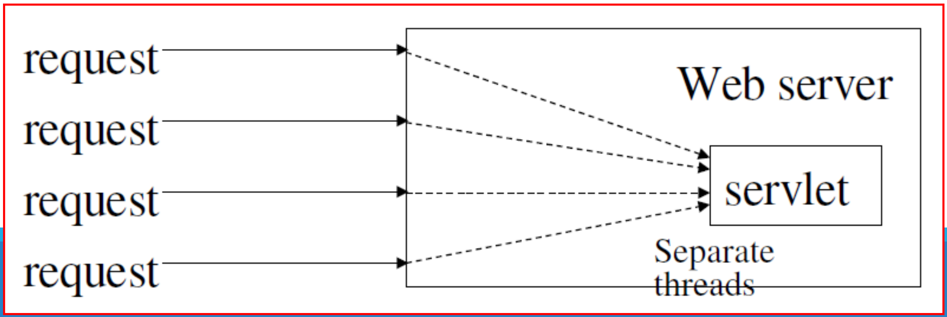
The **Java EE platform** uses "**containers**" to simplify development.

Java EE Containers

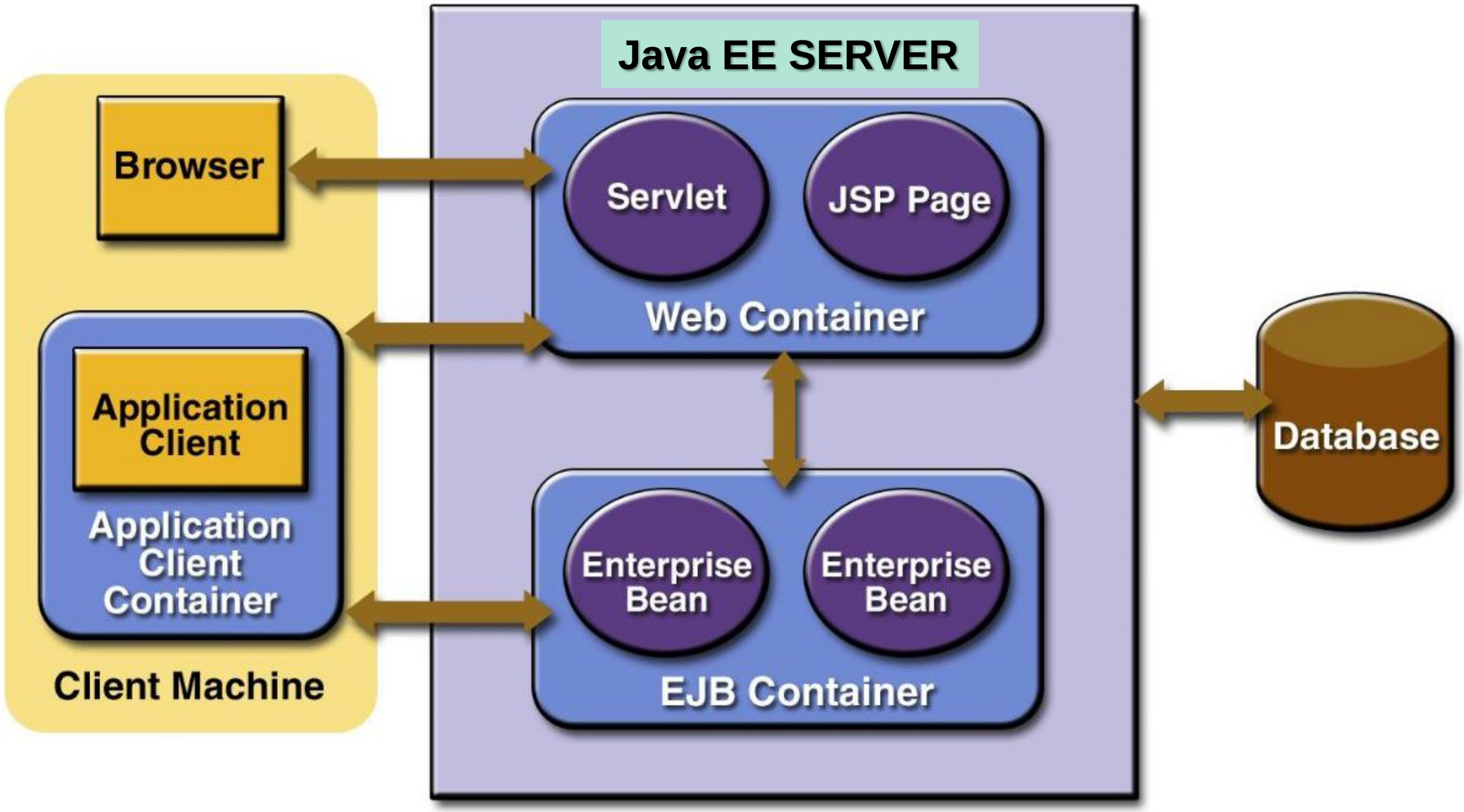
When a request comes in:

- a **Servlet** needs to be **instantiated** and create a **new thread** to handle the request.
- call the **Servlet's** **doPost()**or **doGet()** method and pass the **HTTP request** and **HTTP response** **objects**
- get the request and the response to the **Servlet**
- manage the life, death and resources of the **Servlet**

* All of the above are the tasks of the **web container**.



Java EE Containers



Container Types

Java EE server:

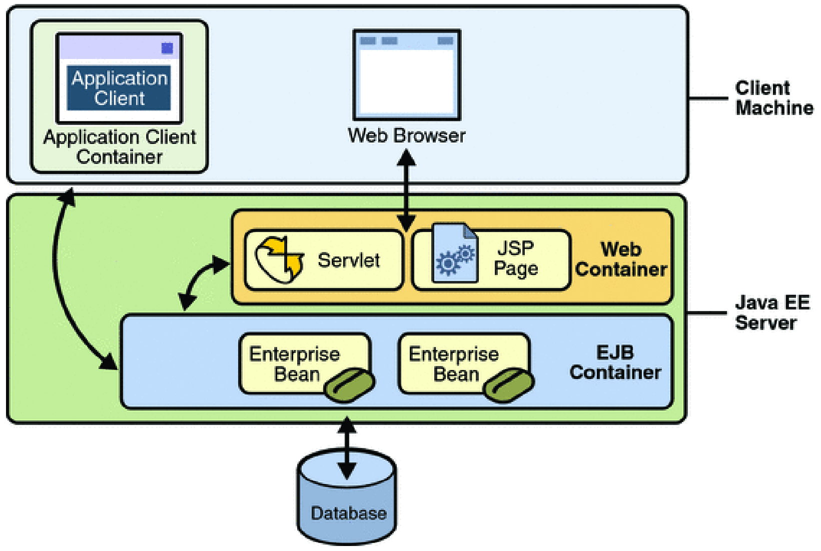
- The runtime portion of a Java EE product. A Java EE server provides EJB and web containers.

Enterprise JavaBeans (EJB) container:

- Manages the execution of enterprise beans for Java EE applications.

Web container:

- Manages the execution of JSP page and servlet components for Java EE applications.



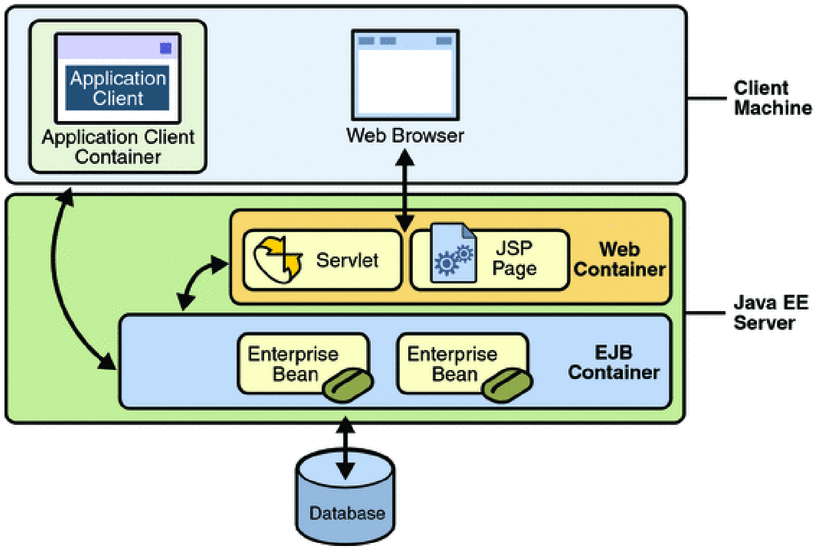
Container Types

Application client container:

- Manages the execution of application client components.

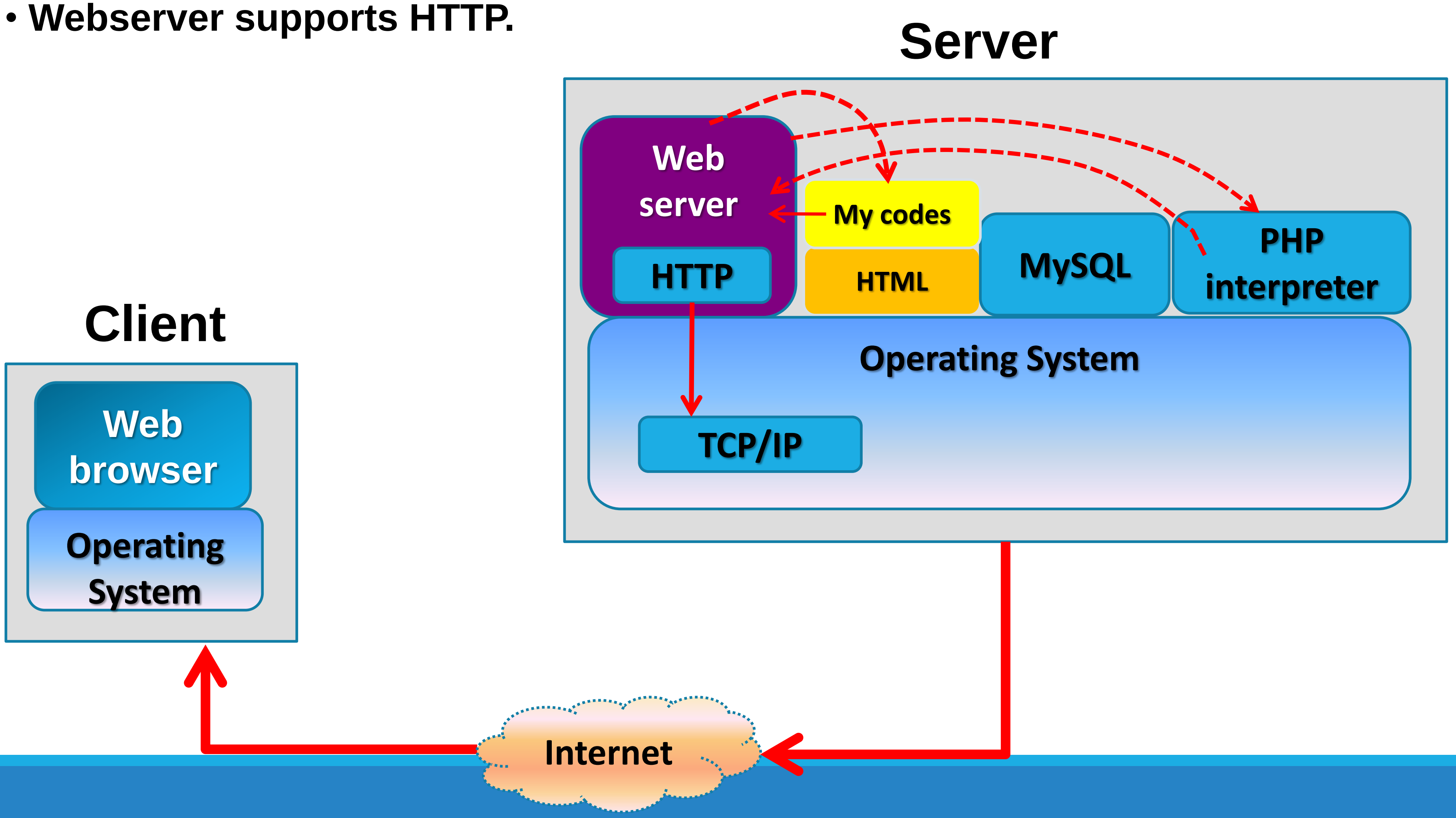
Applet container:

- Manages the execution of applets. Consists of a web browser and Java Plug-in running on the client together.



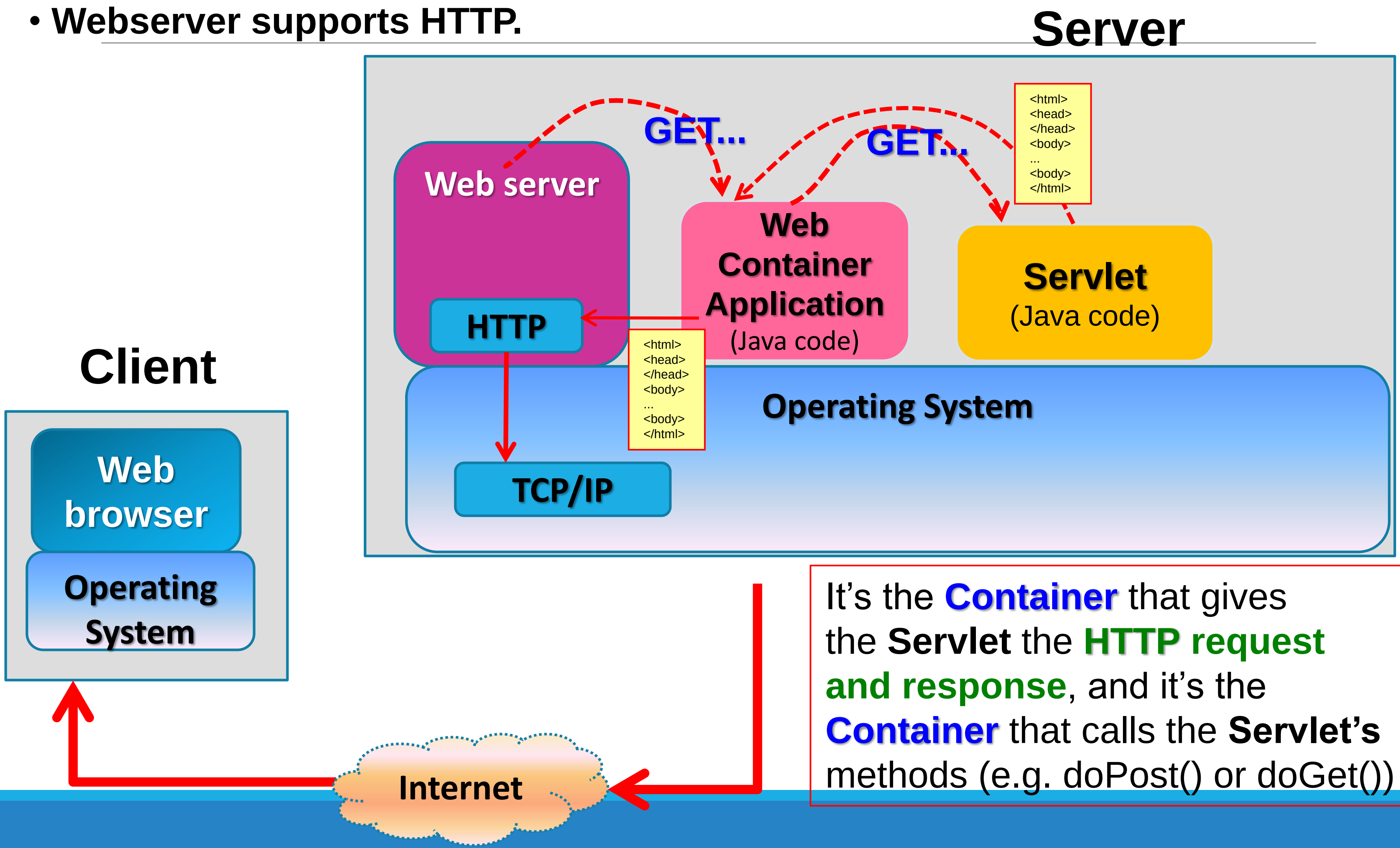
Recall: (PHP-MySQL) Server: response

- Webserver supports HTTP.



Historically (Java Web App)
Server: response

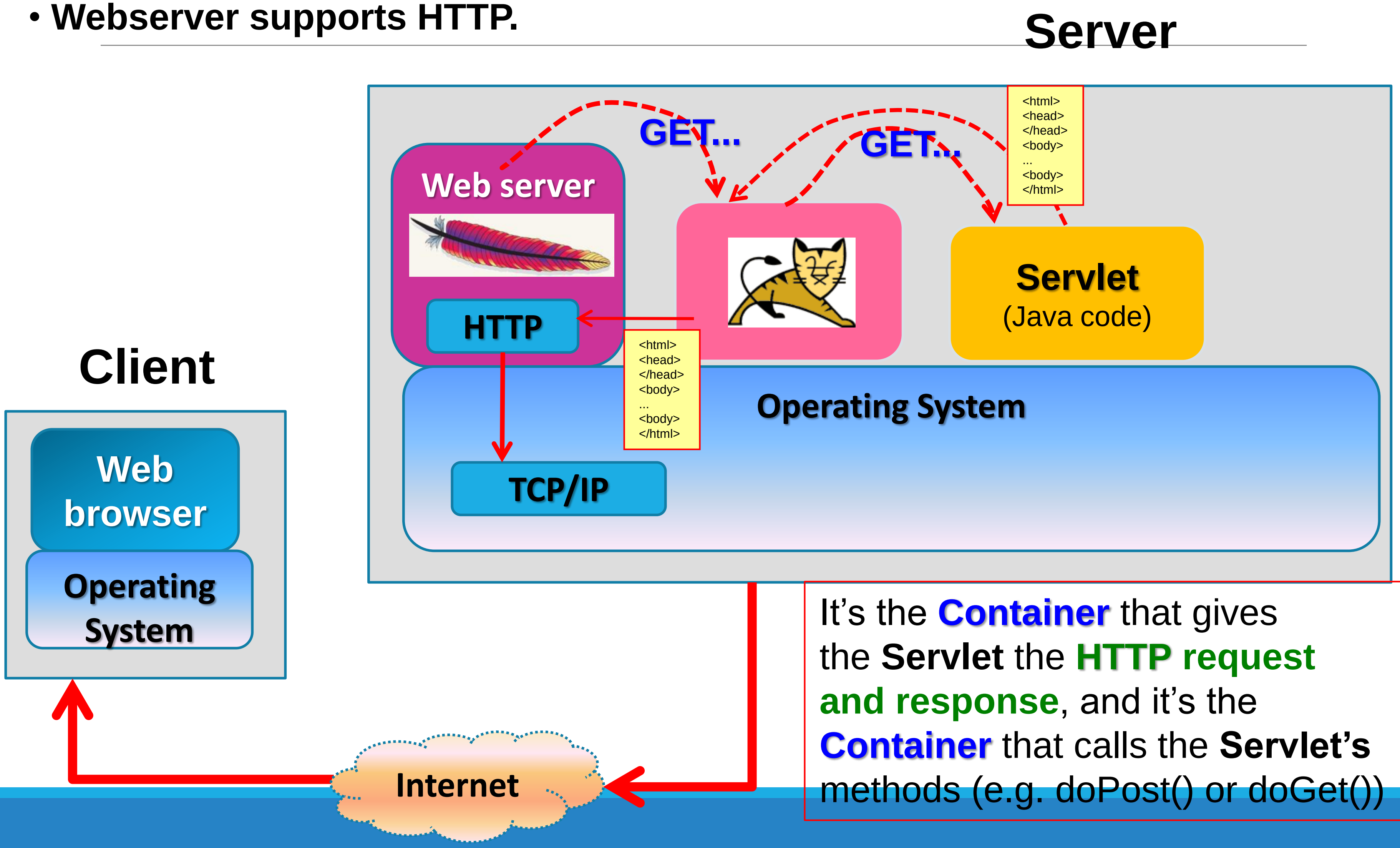
- **Webserver supports HTTP.**



It's the **Container** that gives the **Servlet** the **HTTP request and response**, and it's the **Container** that calls the **Servlet's** methods (e.g. doPost() or doGet())

Historically (Java Web App)
Server: response

- Webserver supports HTTP.

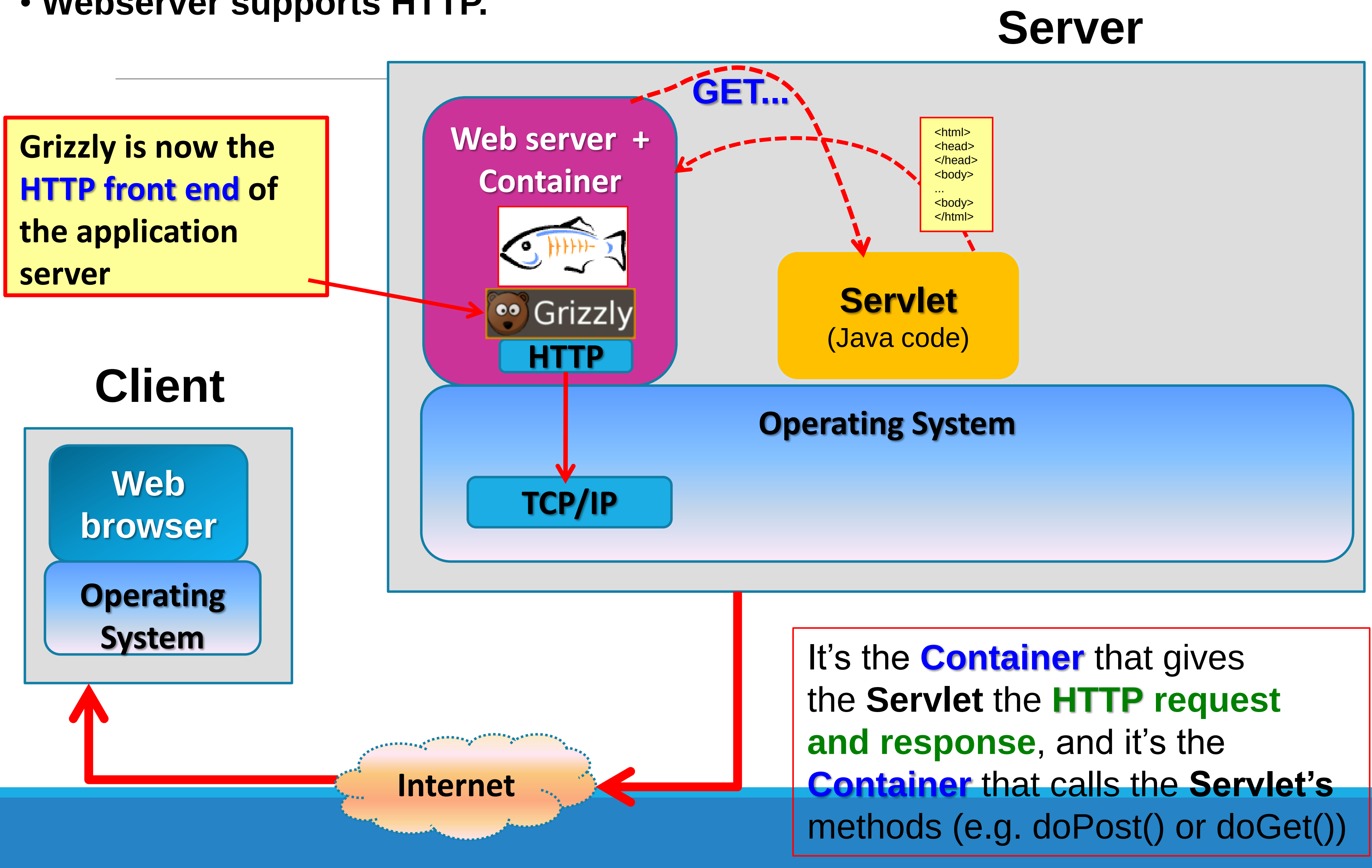


It's the **Container** that gives the **Servlet** the **HTTP request and response**, and it's the **Container** that calls the **Servlet's** methods (e.g. doPost() or doGet())

(Java Web App) Server: response

• **Webserver supports HTTP.**

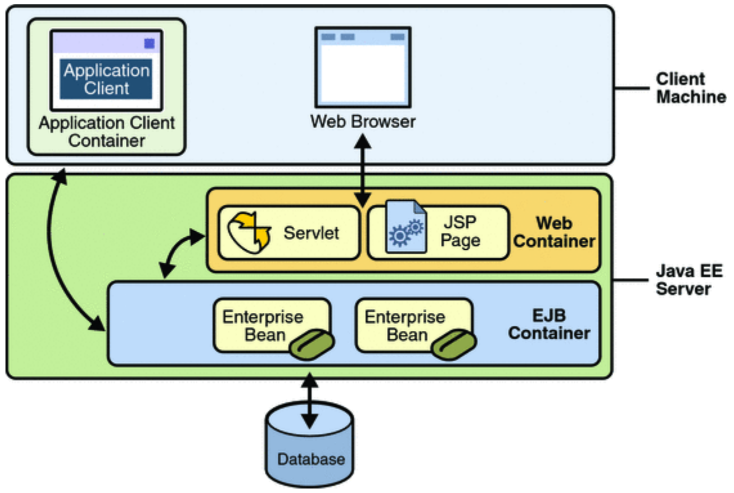
Grizzly is now the **HTTP front end** of the application server

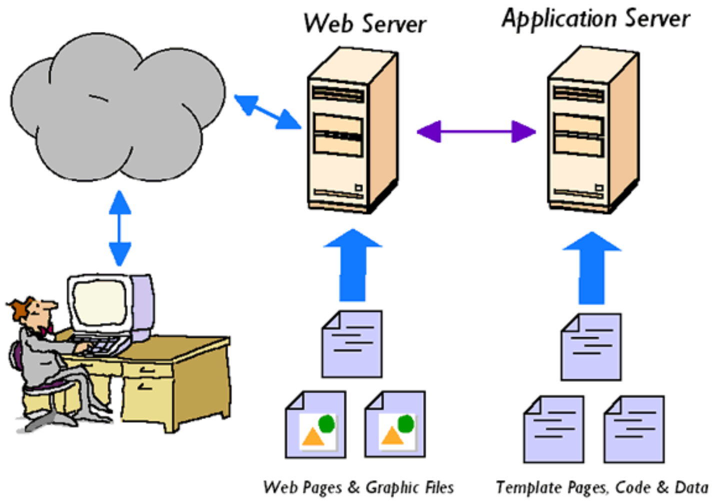


It's the **Container** that gives the **Servlet** the **HTTP request and response**, and it's the **Container** that calls the **Servlet's** methods (e.g. doPost() or doGet())

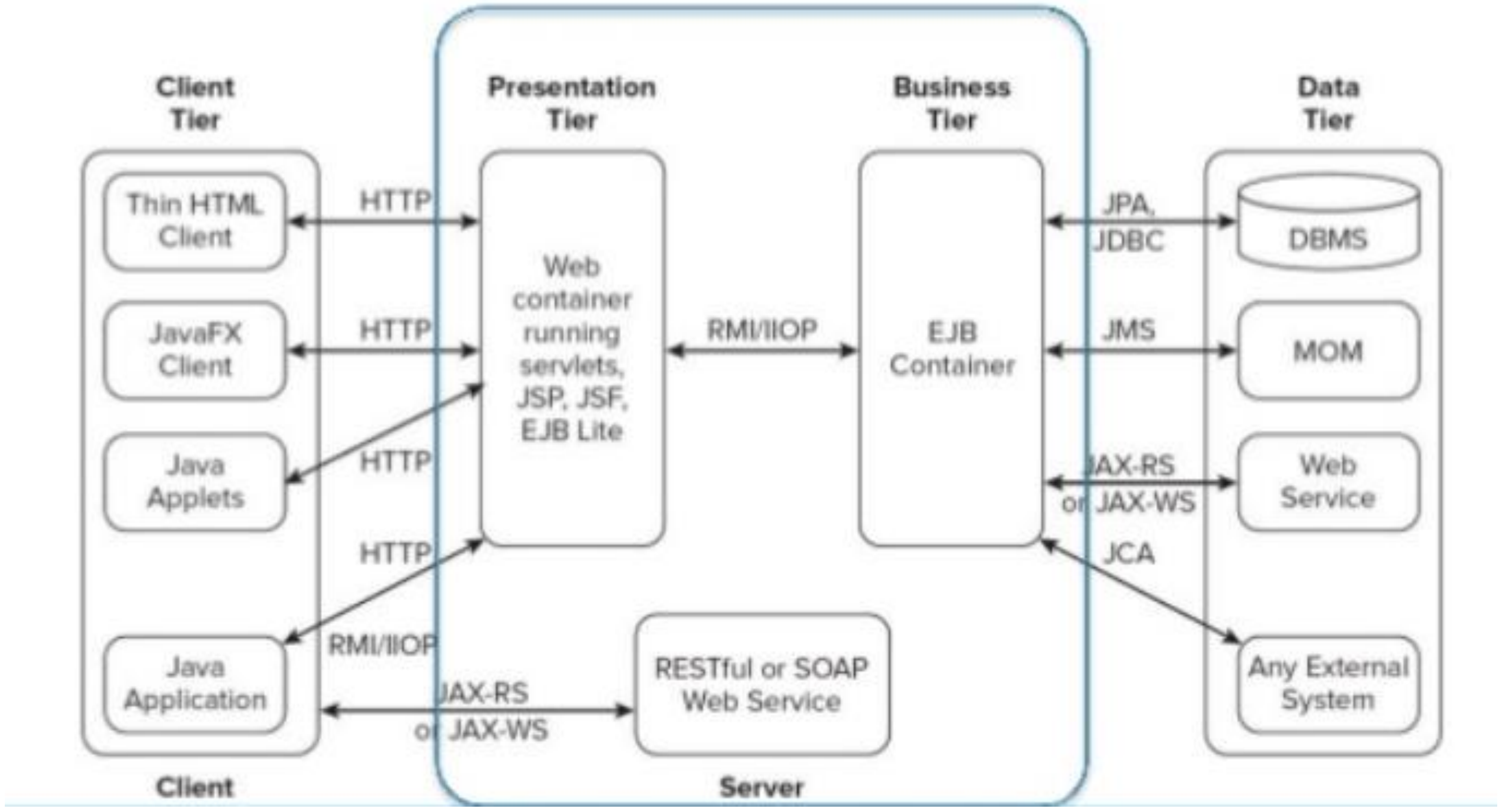
Activity 1

Find the difference between Web Server, Web Container and Application Server





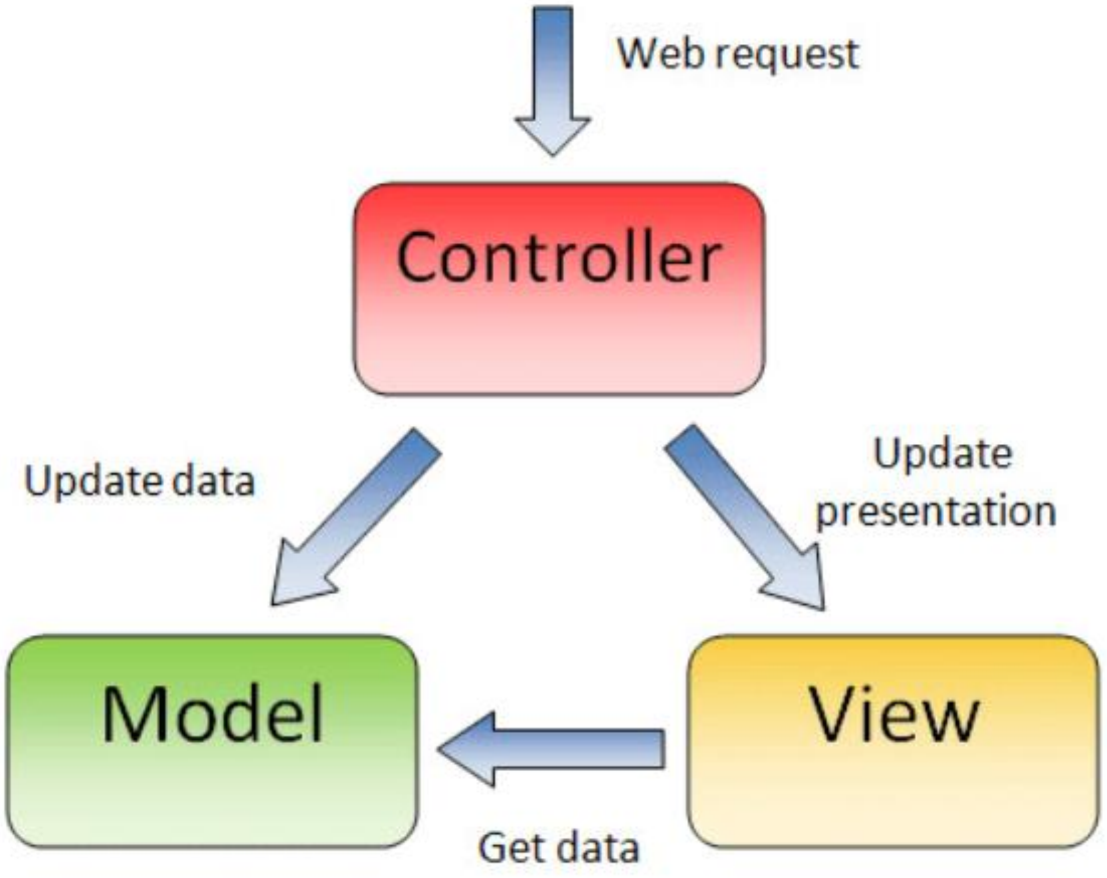
Java EE Architecture …the big picture



Java Frameworks (Patterns)

PRIME FACES
DELTA SPIKE
arquillian
RichFaces
NetBeans
AeroGear
myeclipse
vaadin }>
Apache Camel
PicketLink
eclipseLink
HIBERNATE
APACHE SHIRO
vraptor
spring
errai
Infinispan
OSGi Alliance
eclipse
APACHE WICKET
ZEROTURNAROUND
Forge
equinox OSGi
HAZELCAST SOFTWARE
Apache Maven Project

Model View Controller



Model View Controller

- **Model-View-Controller** (MVC) is a pattern used in software engineering to separate the application logic from the user interface. As the name implies, the MVC pattern has three layers.
- **The Model defines the business layer of the application, the Controller manages the flow of the application, and the View defines the presentation layer of the application.**
- Although the MVC pattern isn't specific to web applications, it fits very well in this type of applications. In a Java context, the **Model** consists of simple Java classes, the **Controller** consists of servlets and the **View** consists of JSP pages.

The Model Layer

1. This is the data layer which contains business logic of the system, and also represents the state of the application.
2. It’s independent of the presentation layer, the controller fetches the data from the Model layer and sends it to the View layer.

Example: Model layer

To implement a web application based on MVC design pattern, we'll create the *Student* and *StudentService* classes – which will act as our Model layer.

```java
public class Student {
    private int id;
    private String firstName;
    private String lastName;

    // constructors, getters and setters goes here
}
```

```java
public class StudentService {

    public Optional<Student> getStudent(int id) {
        switch (id) {
            case 1:
                return Optional.of(new Student(1, "John", "Doe"));
            case 2:
                return Optional.of(new Student(2, "Jane", "Goodall"));
            case 3:
                return Optional.of(new Student(3, "Max", "Born"));
            default:
                return Optional.empty();
        }
    }
}
```

The Controller Layer

1. Controller layer acts as an interface between View and Model. It receives requests from the View layer and processes them, including the necessary validations.
2. The requests are further sent to Model layer for data processing, and once they are processed, the data is sent back to the Controller and then displayed on the View.

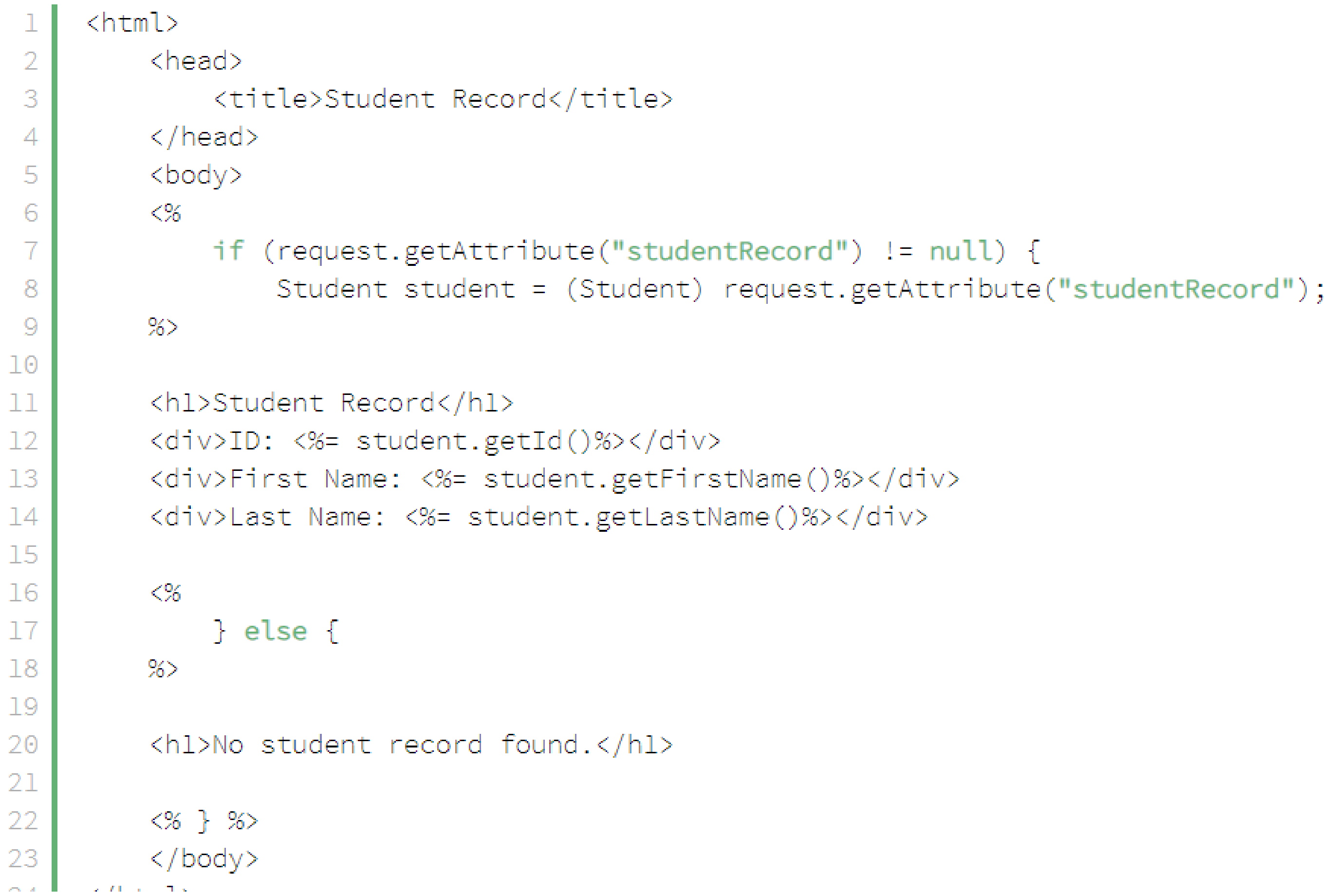
Example: Controller layer

```java
@WebServlet(
  name = "StudentServlet",
  urlPatterns = "/student-record")
public class StudentServlet extends HttpServlet {

    private StudentService studentService = new StudentService();

    private void processRequest(
      HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {

        String studentID = request.getParameter("id");
        if (studentID != null) {
            int id = Integer.parseInt(studentID);
            studentService.getStudent(id)
              .ifPresent(s -> request.setAttribute("studentRecord", s));
        }

        RequestDispatcher dispatcher = request.getRequestDispatcher(
          "/WEB-INF/jsp/student-record.jsp");
        dispatcher.forward(request, response);
    }
```

The View Layer

1. This layer represents the output of the application, usually some form of UI. The presentation layer is used to display the Model data fetched by the Controller.

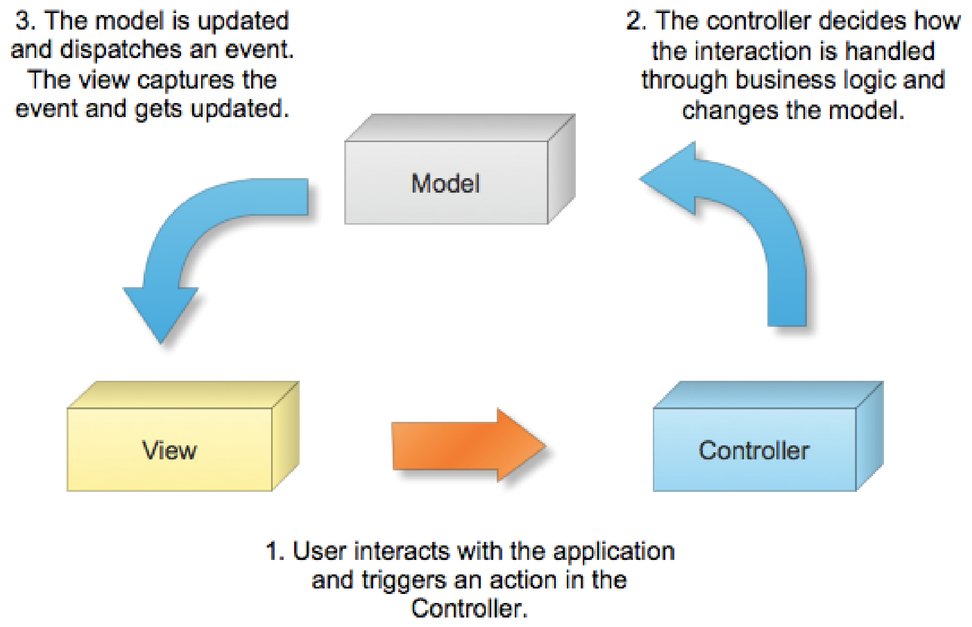
Example: View layer



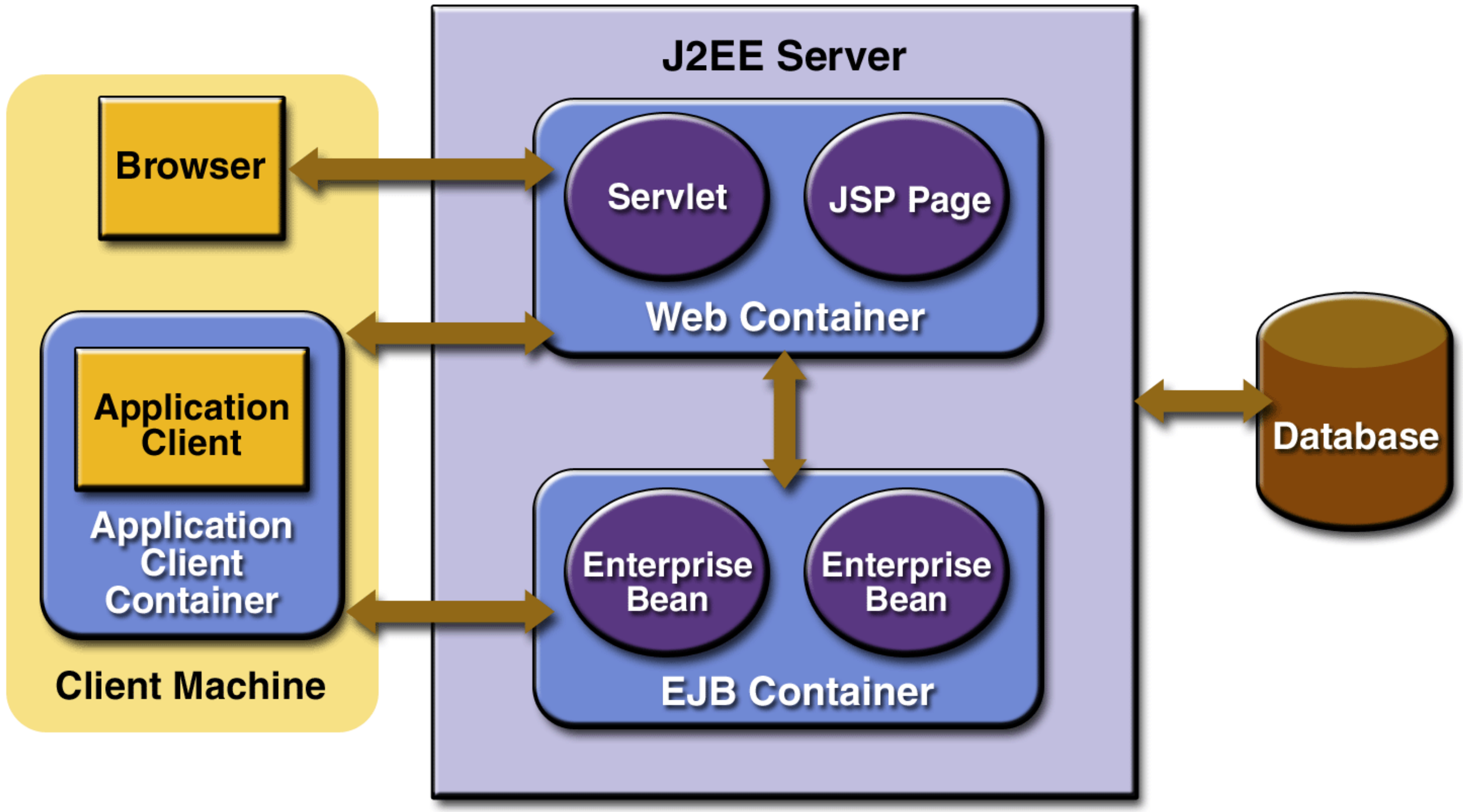
```
<html>
    <head>
        <title>Student Record</title>
    </head>
    <body>
    <%
        if (request.getAttribute("studentRecord") != null) {
            Student student = (Student) request.getAttribute("studentRecord");
    %>

    <h1>Student Record</h1>
    <div>ID: <%= student.getId()%></div>
    <div>First Name: <%= student.getFirstName()%></div>
    <div>Last Name: <%= student.getLastName()%></div>

    <%
        } else {
    %>

    <h1>No student record found.</h1>

    <% } %>
    </body>
```

Discussions

List the advantages and disadvantages of MVC framework.


3. The model is updated and dispatches an event. The view captures the event and gets updated.
2. The controller decides how the interaction is handled through business logic and changes the model.
Model
View
Controller
1. User interacts with the application and triggers an action in the Controller.

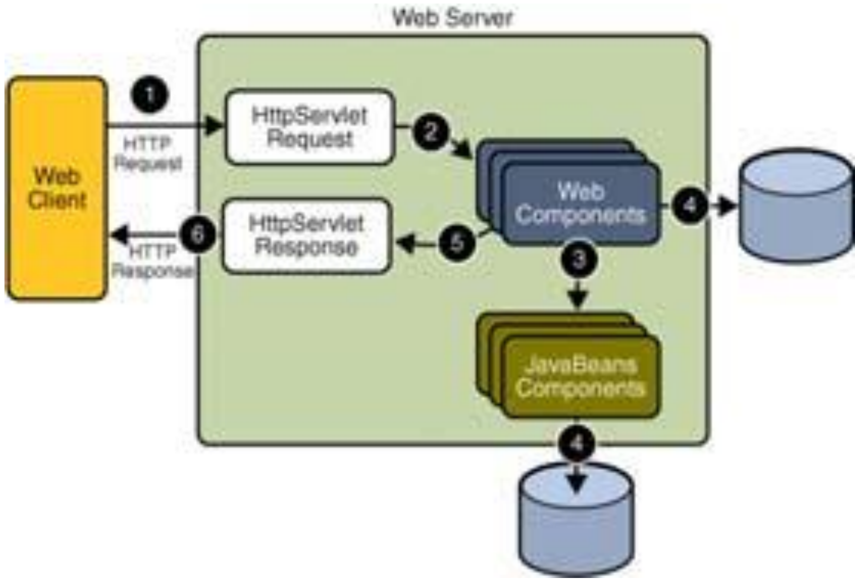
Java EE Components (Web components, EJB)



Web Applications

Client sends HTTP request

- Web server that implements JavaServlet and JavaServer Pages technology converts request into an HTTPServletRequest object
- Object is delivered to a web component, which can interact with JavaBeans components or a database to generate dynamic content
- Web component can then generate an HTTPServletResponse or pass request to another web component
- Eventually a web component generates a HTTPServletResponse object, which is converted to an HTTP response and returned to the client



Web Application Life Cycle

1. Develop the web component code
2. Develop the web application deployment descriptor
3. Compile the web application components and helper classes referenced by the components
4. Optionally package the application into a deployable unit
5. Deploy the application into a web container
6. Access a URL that references the web application

Web Modules

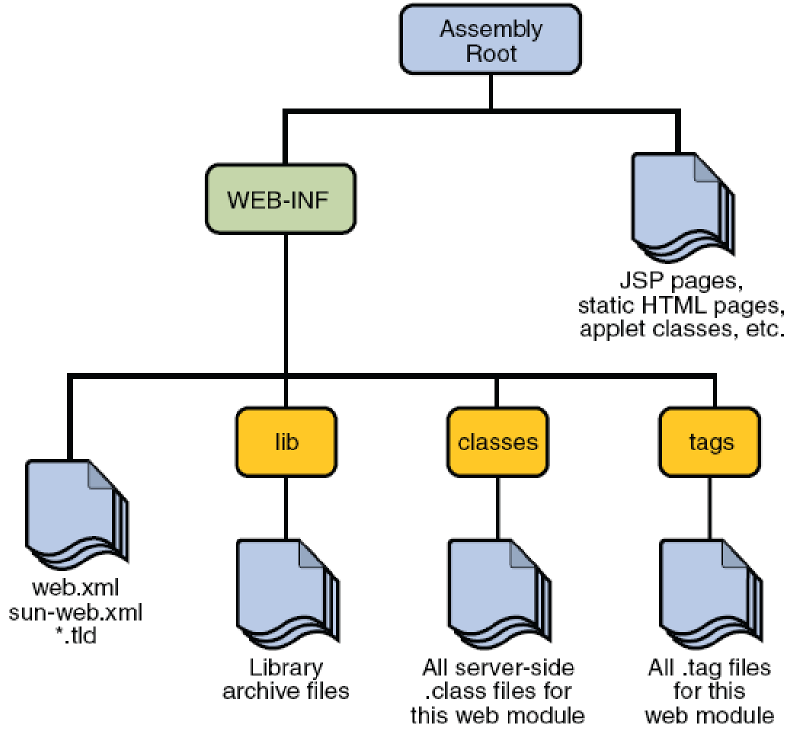
Smallest deployable and usable unit of web resources

web.xml: Web application deployment descriptor

- Not needed if module does not contain any servlets, filter, orlistener components (i.e., only has JSP pages and static files)

sun-web.xml: runtime deployment descriptor

- Context root of web application, mapping of names of application resources to Application Serverresources


Assembly Root
WEB-INF
JSP pages, static HTML pages, applet classes, etc.
web.xml sun-web.xml *.tld
lib
Library archive files
classes
All server-side .class files for this web module
tags
All .tag files for this web module

Java Servlets

Java Servlets simplify web development by providing **infrastructure** for **component**, **communication**, and **session management** in a web container that is integrated with a **web server**.

- Writing **Servlets** is like writing Java codes that place an HTML page inside a Java class (this is the **worst part** of Servlets!)

- (**Historically!**) requires a **deployment descriptor** (**DD**). This is in the form of an **XML file**.

- **Servlets** do not have a **main()** method.
- **Servlets** are under the control of another Java application called a **Container**

Java Servlets

- **Servlets are Java classes that dynamically process requests and construct responses.**
- Server side replacement for CGI
- Extensions to Java enabled web-servers
- Inherently **multi-threaded**.
- One thread per request.
- Very efficient.
- Platform independent.

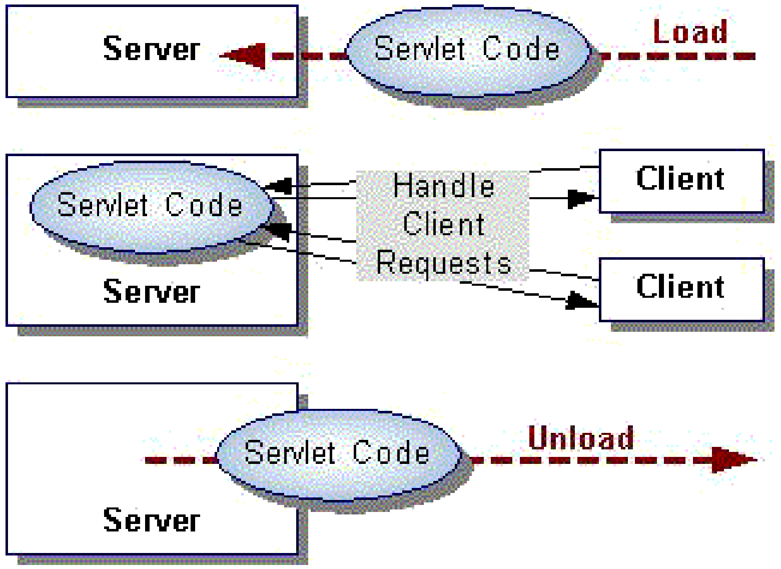
How do Servlets work?

- **Servlets** run inside a **Web Container** - the component of the web server that runs and interacts with Servlets
- **Servlet** is running on the server listening for requests
- When a **request** comes in, a **new thread** is generated by the **web container**.

Anatomy of a Servlet

init() – the init() function is called when the servlet is initialized by the server. This often happens on the first doGet() or doPut() call of the servlet.

destroy() – this function is called when the servlet is being destroyed by the server, typically when the server process is being stopped.

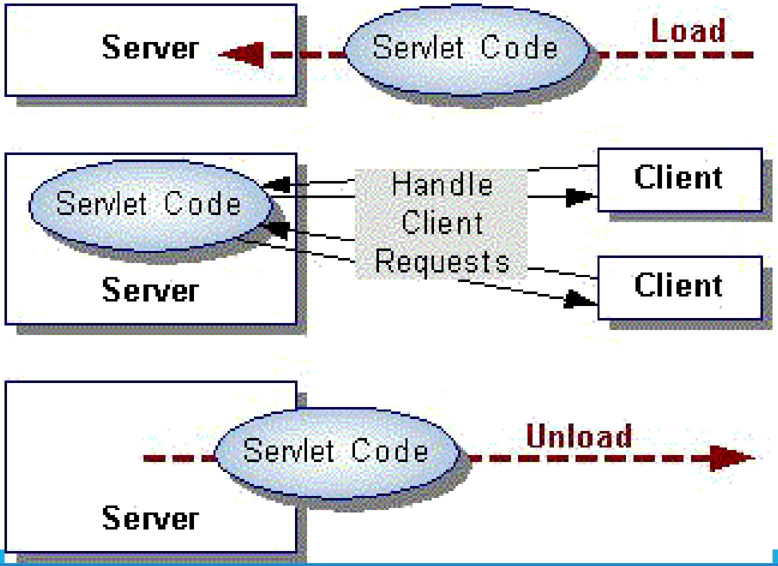


Anatomy of a Servlet

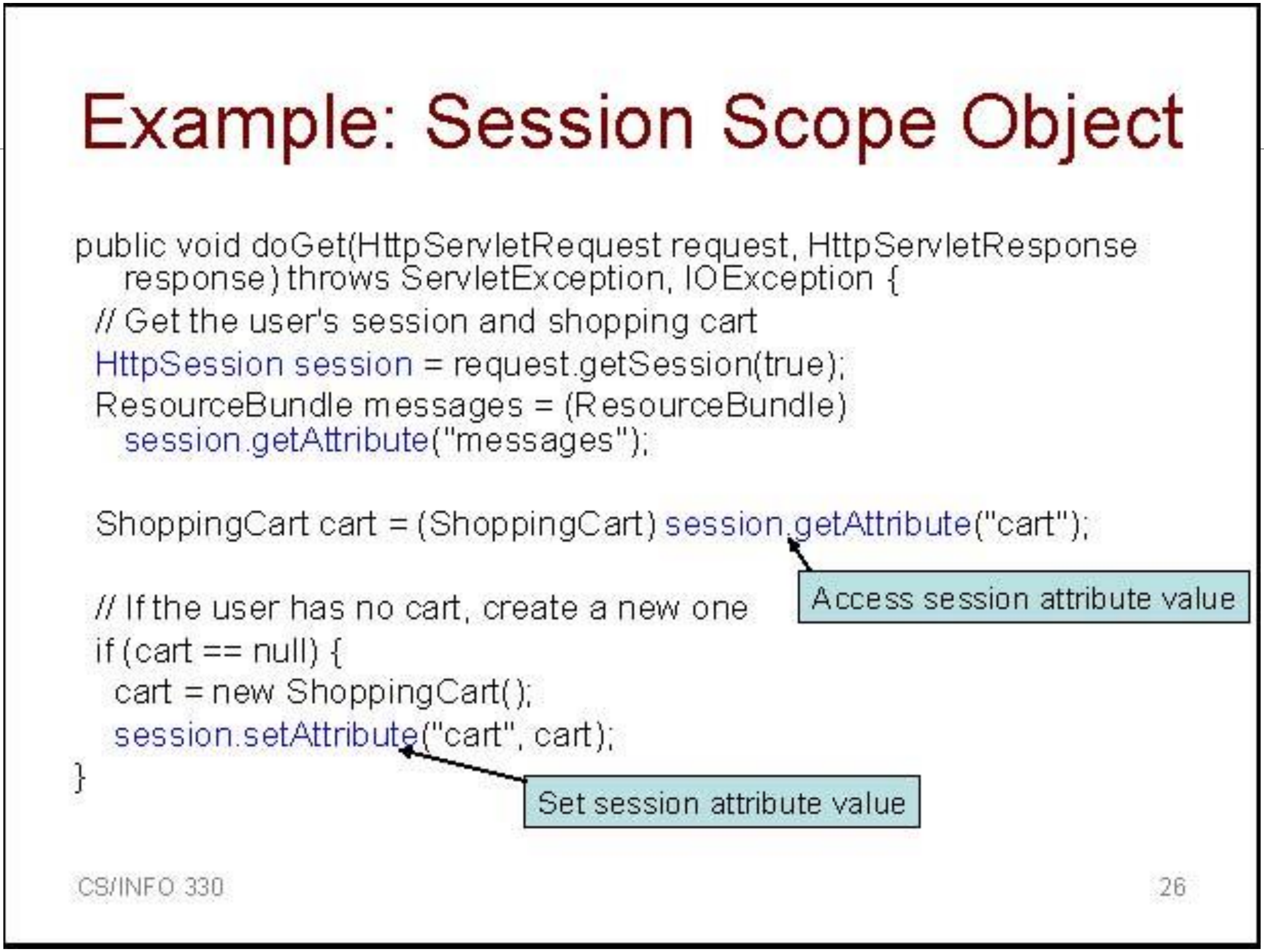
doGet() – the doGet() function is called when the servlet is called via an HTTP GET.

doPost() – the doPost() function is called when the servlet is called via an HTTP POST.

- POSTs are a good way to get input from HTML forms



http://java.sun.com/docs/books/tutorial/servlets/lifecycle/index.html



Example: Session Scope Object

```
public void doGet(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
  // Get the user's session and shopping cart
  HttpSession session = request.getSession(true);
  ResourceBundle messages = (ResourceBundle)
    session.getAttribute("messages");

  ShoppingCart cart = (ShoppingCart) session.getAttribute("cart");

  // If the user has no cart, create a new one
  if (cart == null) {
    cart = new ShoppingCart();
    session.setAttribute("cart", cart);
}
```

Access session attribute value

Set session attribute value

CS/INFO 330 26

JSP – JavaServer Pages

- JavaServer Pages technology uses XML-like tags and scriptlets written in the Java programming language to encapsulate the logic that generates the content for the page.
- Any and all formatting (HTML or XML) tags are passed directly back to the response page.
- By separating the page logic from its design and display and supporting a reusable component-based design, JSP technology makes it faster and easier than ever to build web-based applications.

http://java.sun.com/products/jsp/index.html

SAMPLE JSP

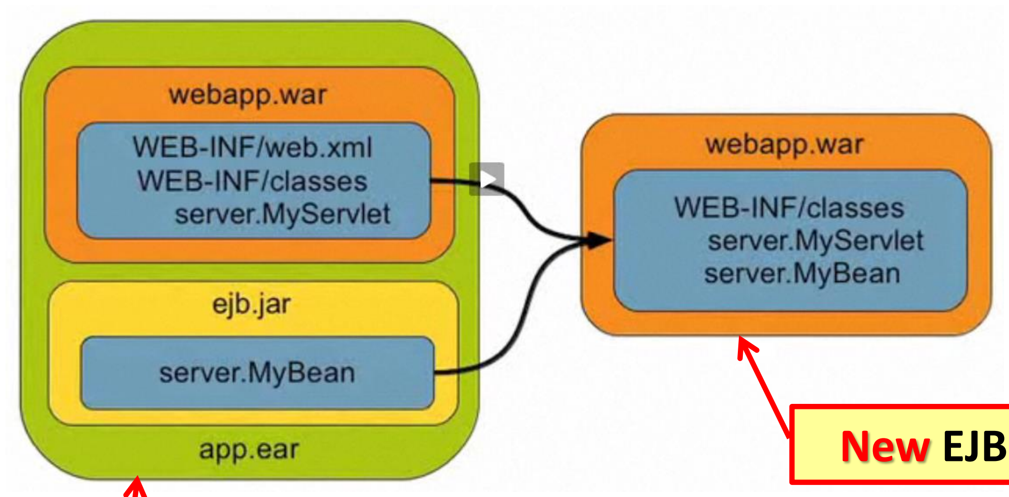
```
<html>                                                  <!- Apache Tomcat
  Samples ->
<!--  Copyright (c) 1999 The Apache Software Foundation.  All rights reserved.-->
<body bgcolor="white">
<jsp:useBean id='clock' scope='page' class='dates.JspCalendar'
  type="dates.JspCalendar" />

<font size=4><ul>
<li>          Day of month: is  <jsp:getProperty name="clock"
  property="dayOfMonth"/>
<li>          Year: is  <jsp:getProperty name="clock" property="year"/>
<li>          Month: is  <jsp:getProperty name="clock" property="month"/>
<li>          Time: is  <jsp:getProperty name="clock" property="time"/>
<li>          Date: is  <jsp:getProperty name="clock" property="date"/>
<li>          Day: is  <jsp:getProperty name="clock" property="day"/>
<li>          Day Of Year: is  <jsp:getProperty name="clock" property="dayOfYear"/>
<li>          Week Of Year: is  <jsp:getProperty name="clock"
  property="weekOfYear"/>
<li>          era: is  <jsp:getProperty name="clock" property="era"/>
<li>          DST Offset: is  <jsp:getProperty name="clock" property="DSTOffset"/>
<li>          Zone Offset: is  <jsp:getProperty name="clock" property="zoneOffset"/>
</ul>
</font>

</body>
</html>
```

Enterprise JavaBeans (EJB)

Enterprise JavaBeans container handles:

- distributed communication
- threading
- scaling
- transaction management, etc.

has a new packaging! (see figure)



New EJB 3.1 Packaging

Older EJB Packaging

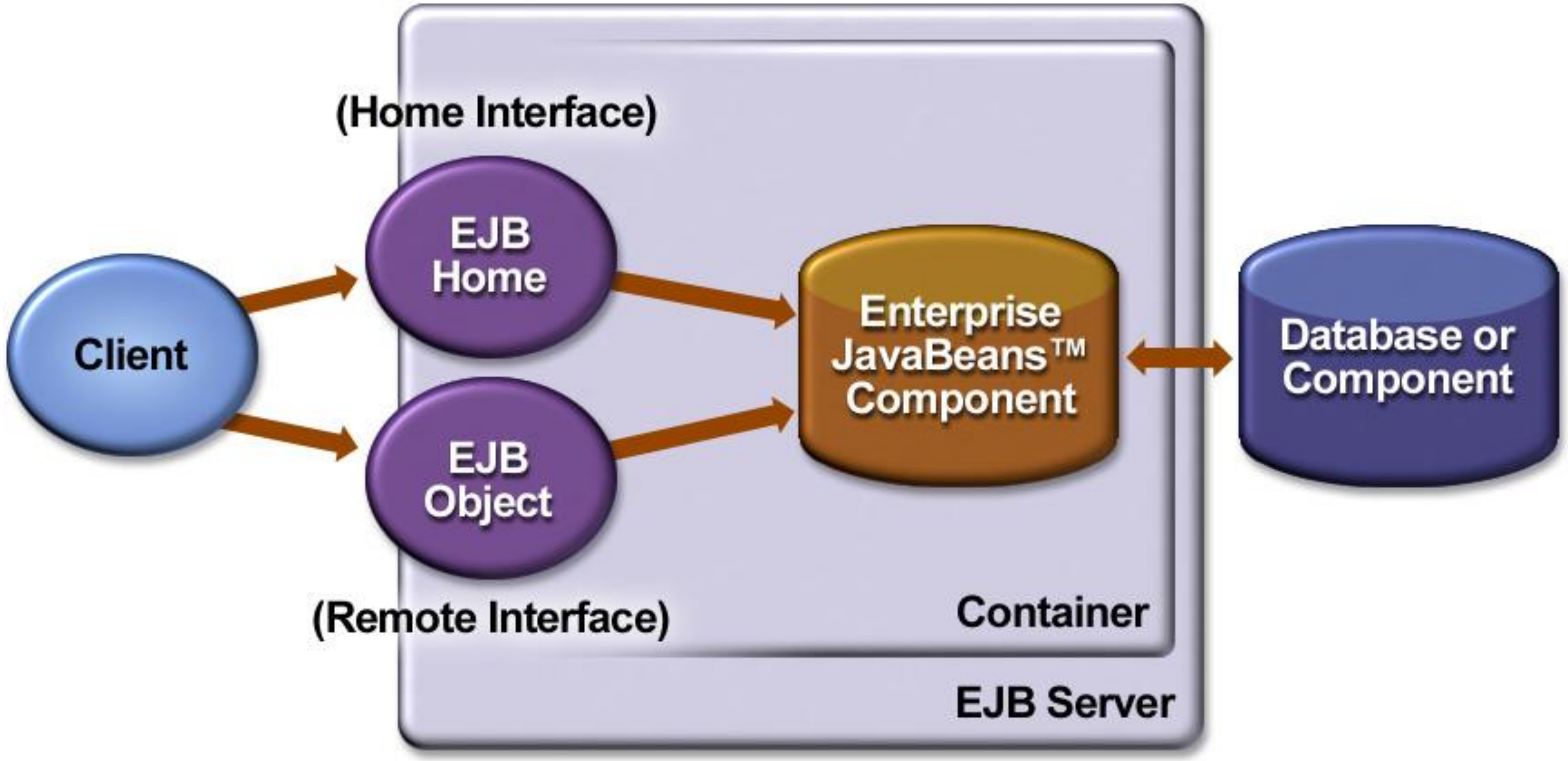
JavaBeans

- **manage the data flow** between the following:

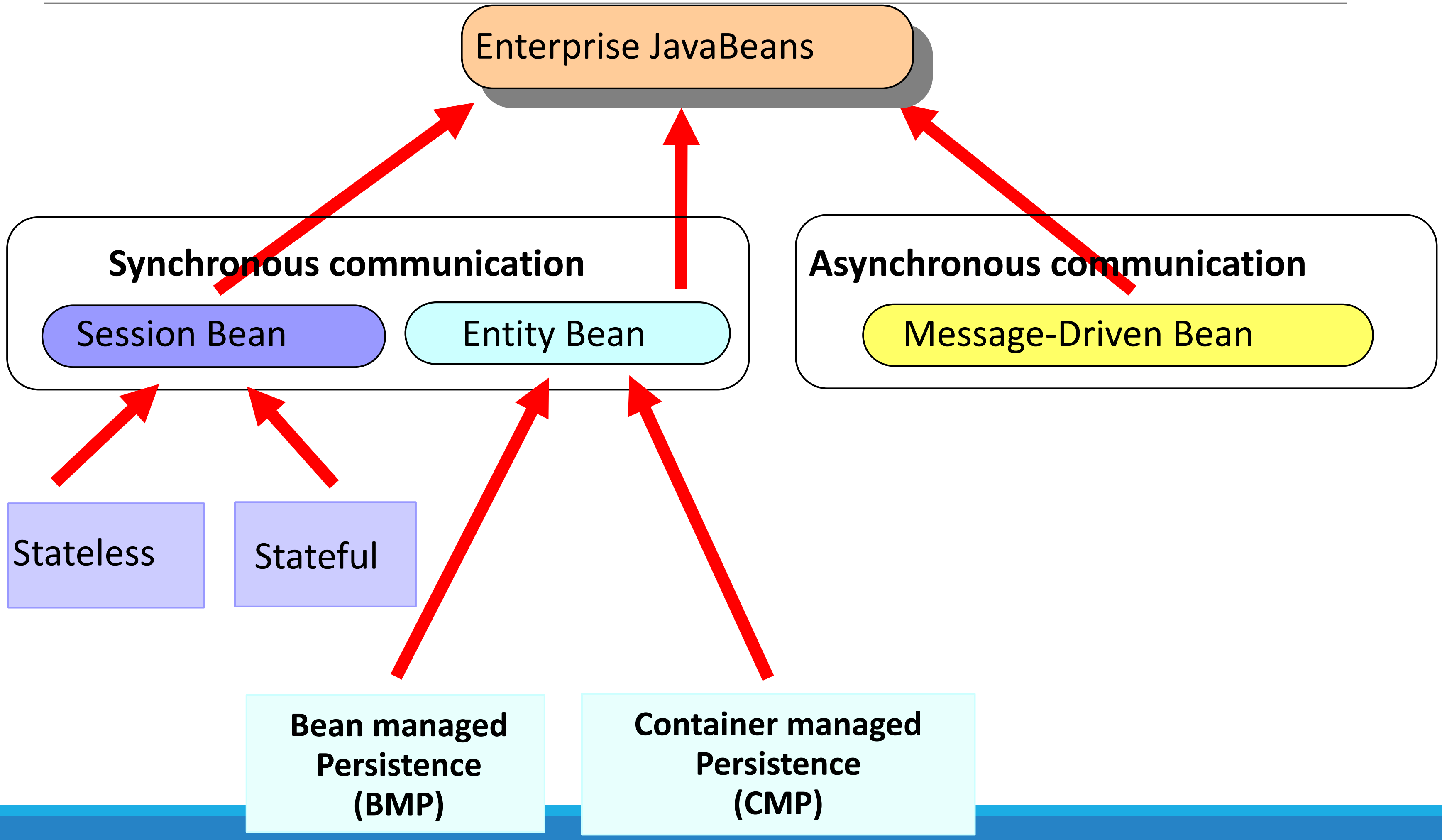
Client/Database	Server
application client or **applet**	components running on the Java EE **server**
database	**Server** components

- **JavaBeans** components are not considered Java EE components by the Java EE specification.
- **JavaBeans** components have properties and have **get** and **set methods** for accessing the **properties**.

EJB Architecture



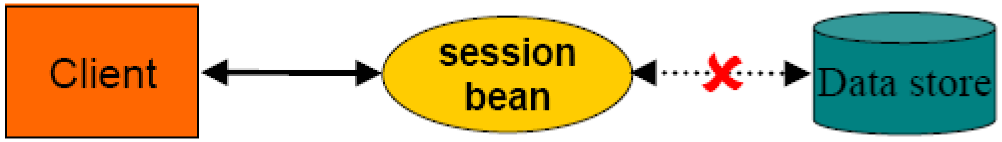
Enterprise JavaBeans



Enterprise JavaBeans
Synchronous communication
Session Bean
Entity Bean
Asynchronous communication
Message-Driven Bean
Stateless
Stateful
Bean managed Persistence (BMP)
Container managed Persistence (CMP)

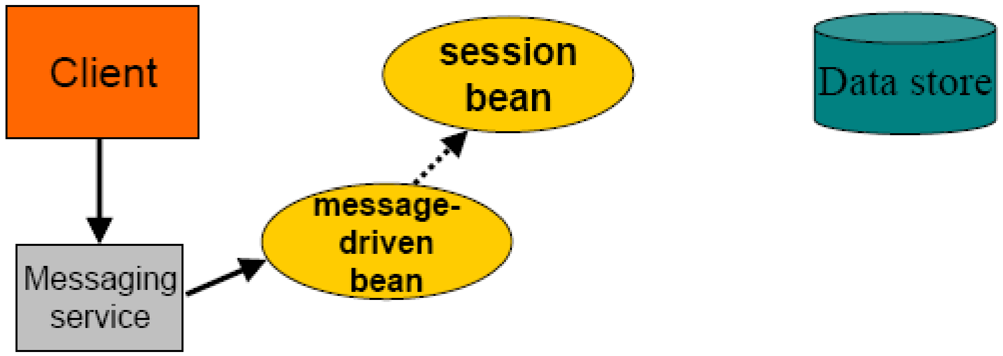
Types of EJB - Session Bean

- ❑A session bean instance is a non-persistent object that implements some business logic and runs on a server
- ❑Session beans can be reused by different clients, but not shared concurrently
- ❑Stateless session beans retain no conversational state between method calls
- ❑Stateful session beans retain conversational state between method calls



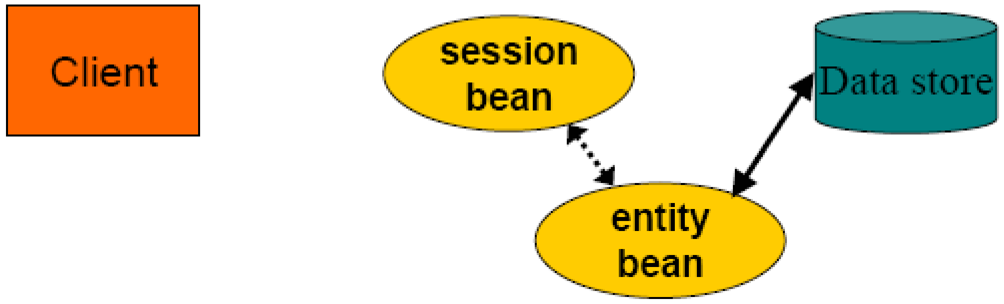
Types of EJB -Message-Driven Bean

- A message-driven bean instance is an asynchronous message consumer
- Message-driven beans have no client visibility



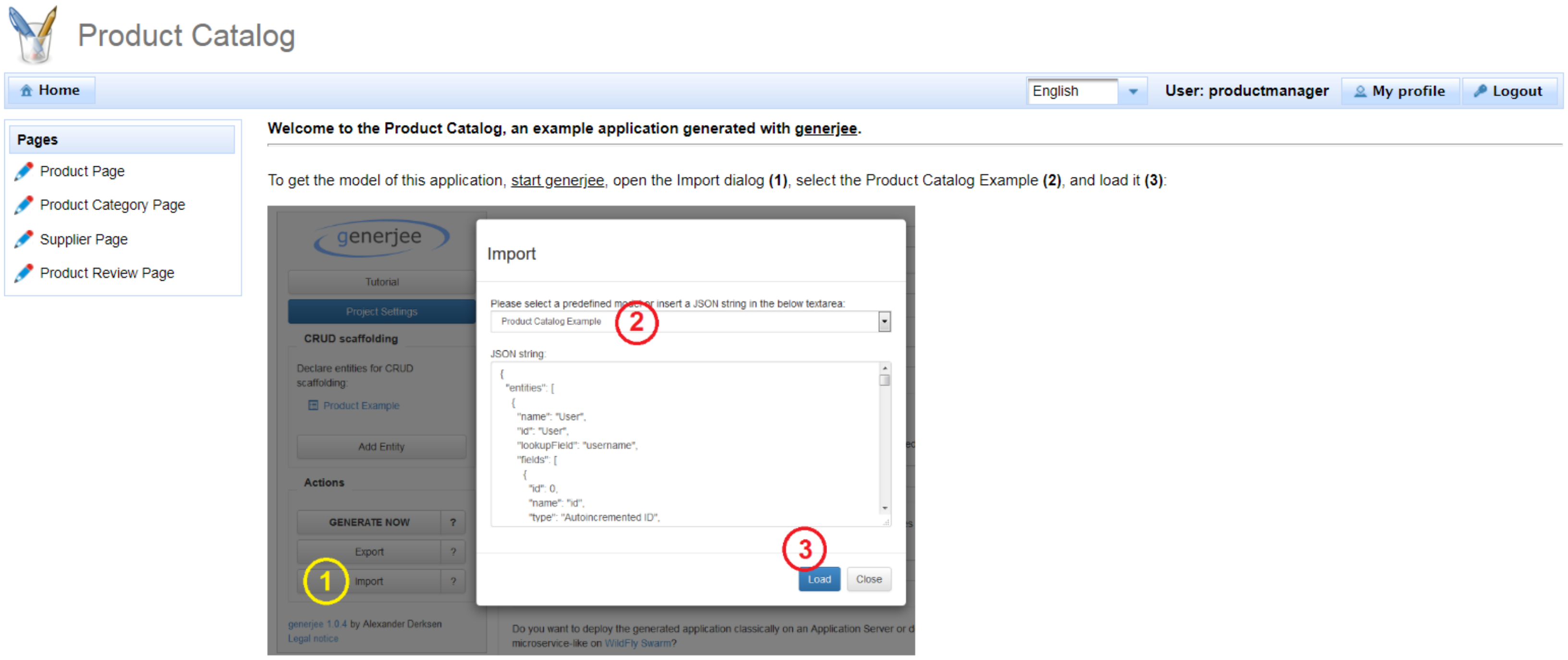
Types of EJB - Entity Bean

- An entity bean instance represents an object-oriented view of data in a persistent storage
- Container-managed persistence (CMP) and bean-managed persistence (BMP) are available



Example online Java EE application

http://generjee.inetseite.de/ProductCatalog/pages/main.xhtml



Product Catalog

Home | English | User: productmanager | My profile | Logout

Pages
- Product Page
- Product Category Page
- Supplier Page
- Product Review Page

Welcome to the Product Catalog, an example application generated with generjee.

To get the model of this application, start generjee, open the Import dialog **(1)**, select the Product Catalog Example **(2)**, and load it **(3)**:

Import

Please select a predefined model or insert a JSON string in the below textarea:

Product Catalog Example

JSON string:

```
{
 "entities": [
  {
   "name": "User",
   "id": "User",
   "lookupField": "username",
   "fields": [
    {
     "id": 0,
     "name": "id",
     "type": "Autoincremented ID",
```

generjee 1.0.4 by Alexander Derksen
Legal notice

Do you want to deploy the generated application classically on an Application Server or d
microservice-like on WildFly Swarm?

Discussion: Final Project