PEOPLE COME FIRST
INFORMATIKAI SZAKÉRTŐK EGYESÜLETE

Application Infrastructure

Objectives

- Standards, containers, APIs, and services
- Application component functionalities mapped to tiers and containers
  - Web container technologies
  - Business logic implementation technologies
  - Web service technologies
- Packaging and deployment
- Enterprise JavaBeans, managed beans, and CDI beans
  - Understanding lifecycle and memory scopes
- Linking components together with annotations, injections, and JNDI

Requirements of Enterprise Applications

- The Java EE platform:
  - Is an architecture for implementing enterprise-class applications
  - Uses Java and Internet technology
  - Has a primary goal of simplifying development of enterprise-class applications through an application model that is:
    - Vendor-neutral
    - Component-based

Separation of Business Logic from Platform Services

Build from the ground up.

Use Application Component Server.

Developer’s Checklist

- □ Business services
- □ Persistence and Transaction management
- □ Multithreading
- □ Security management
- □ Networking

Developer’s Checklist

- □ Business services

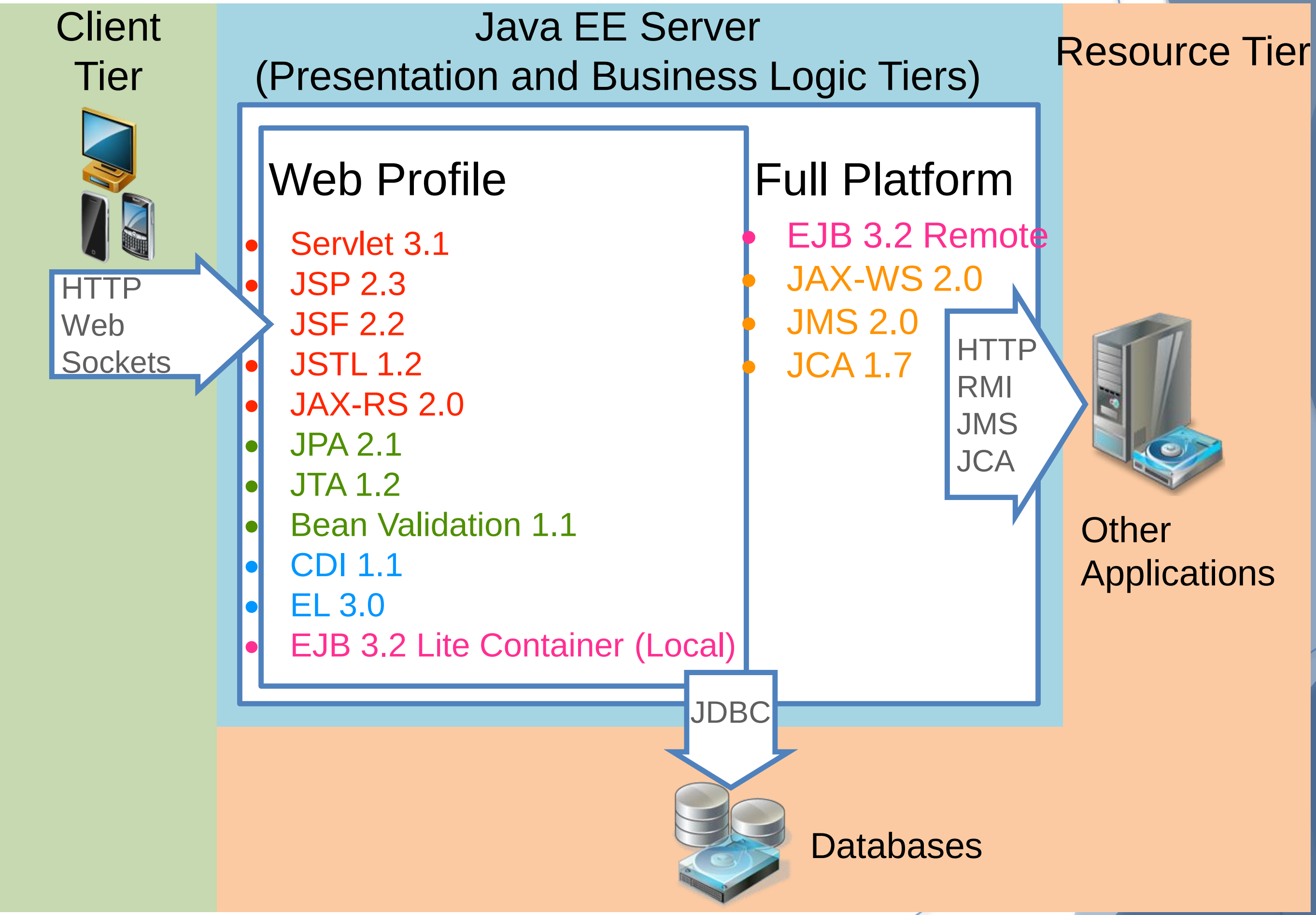
Services Provided by Server

- ☑ Persistence and Transaction management
- ☑ Multithreading
- ☑ Security management
- ☑ Networking

Structure and Purpose of Java EE 7 Server, Containers, and APIs

The Java EE platform describes Web and EJB containers and various APIs:

- → Web Container Technologies
- → Java SE Technologies
- → Technologies in all containers
- → EJB Container Technologies
- → Technologies supported with Full Platform server implementation



EJB Lite and EJB Full Containers

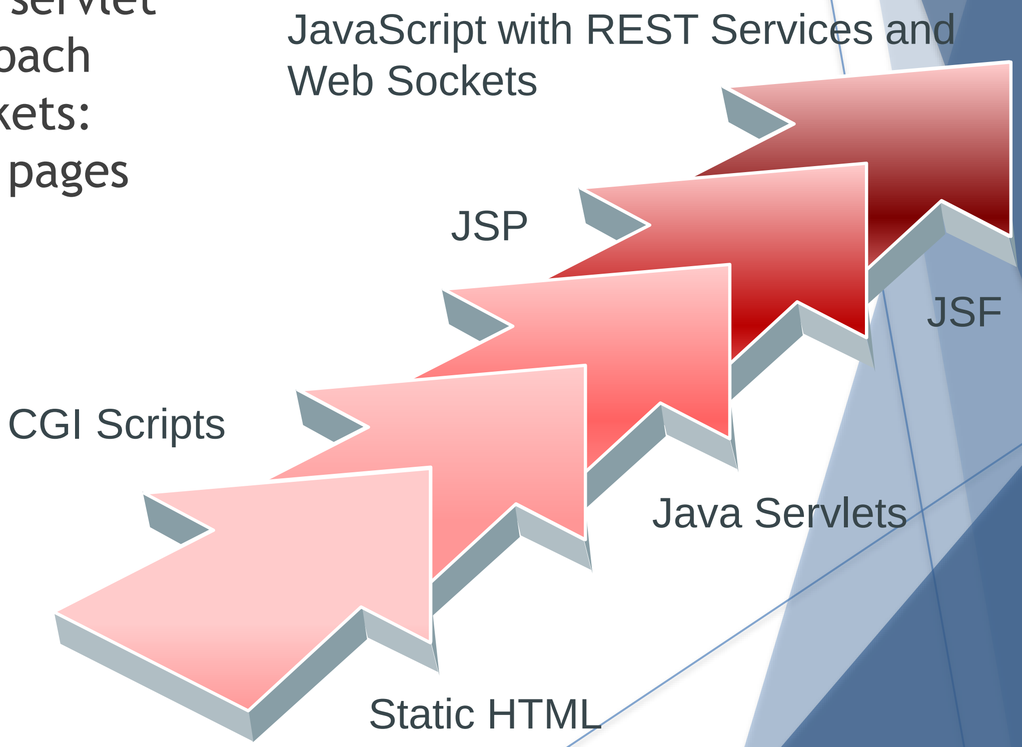
- EJB Lite features:
- Required by the Web Profile
- Session beans components:
  - Stateless
  - Stateful
  - Singleton
- Support local clients
- Method invocations:
  - Synchronous
  - Asynchronous
- Transaction modes:
  - Container-managed
  - Bean-managed
- Declarative and programmatic security
- Automatically created EJB timers

EJB Full = EJB Lite + additional features:

- Required by Full Platform
- Message-driven beans
- Remote and local clients
- JAX-WS web service endpoints
- Persistent EJB timer service
- Support legacy services and EJB APIs

Evolution of Web Design

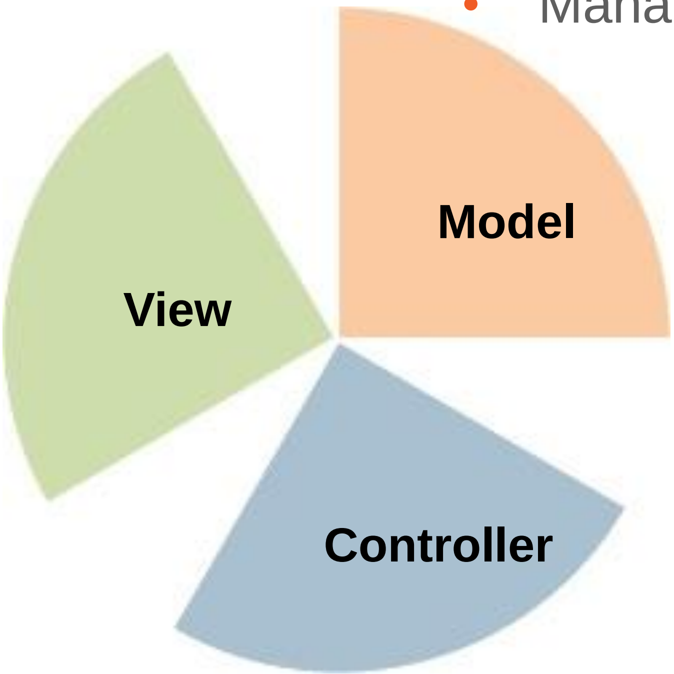
- Web: Started as static HTML documents
- CGI scripts: Introduced dynamically generated content
- Java servlets: Multithreaded and scalable solution
- Java Server Pages: Improved UI design of a servlet
- Java Server Faces: Implemented MVC approach
- JavaScript with REST services and web sockets: Added client-side UI and event handling to pages that are likely to be produced by using Servlet/JSP/JSF



MVC (Model View Controller)

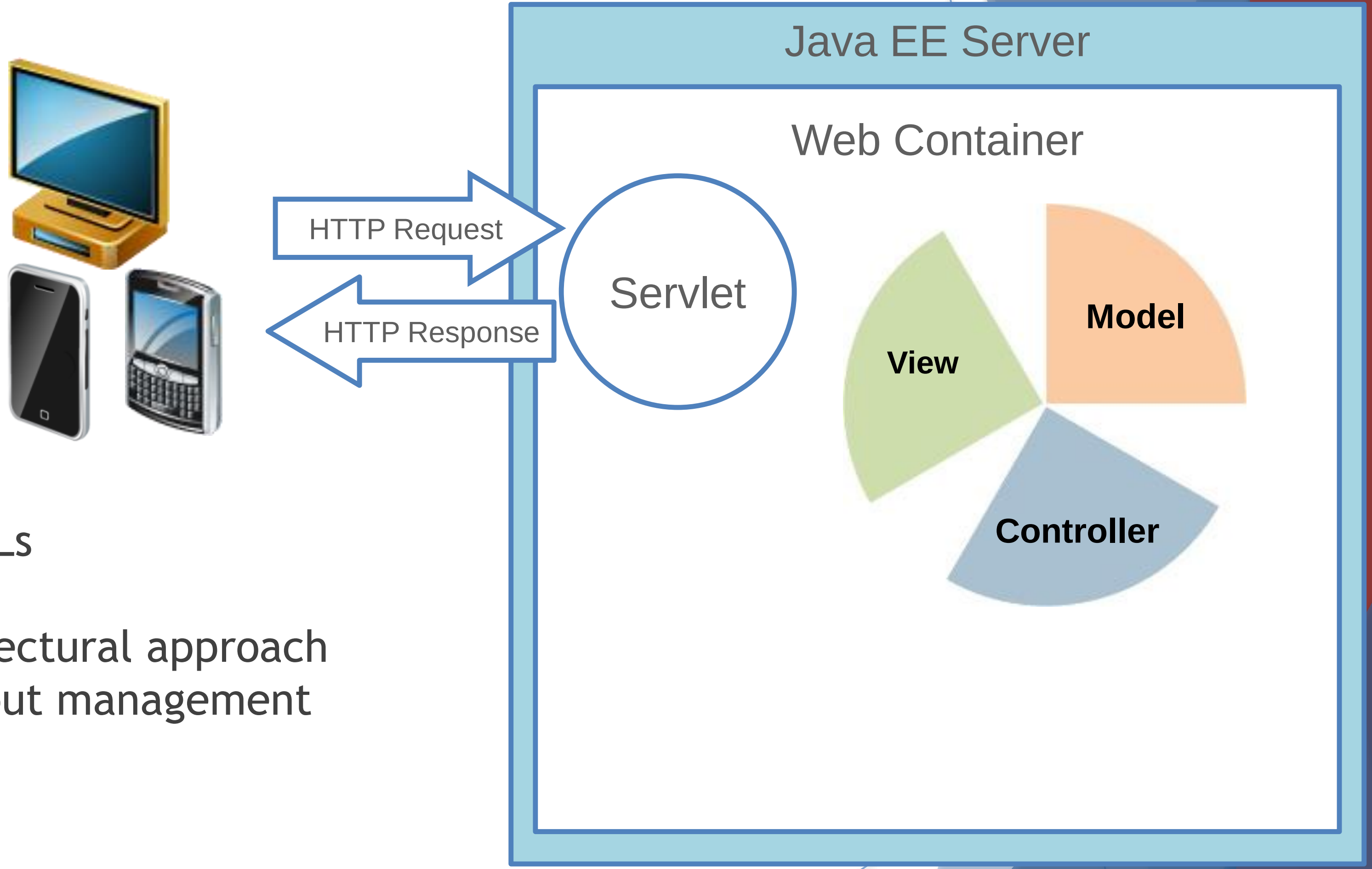
- Implement Business Logic
- Manage Data Persistency

- Produce User Interface
- Manage Presentation
- Generate Events



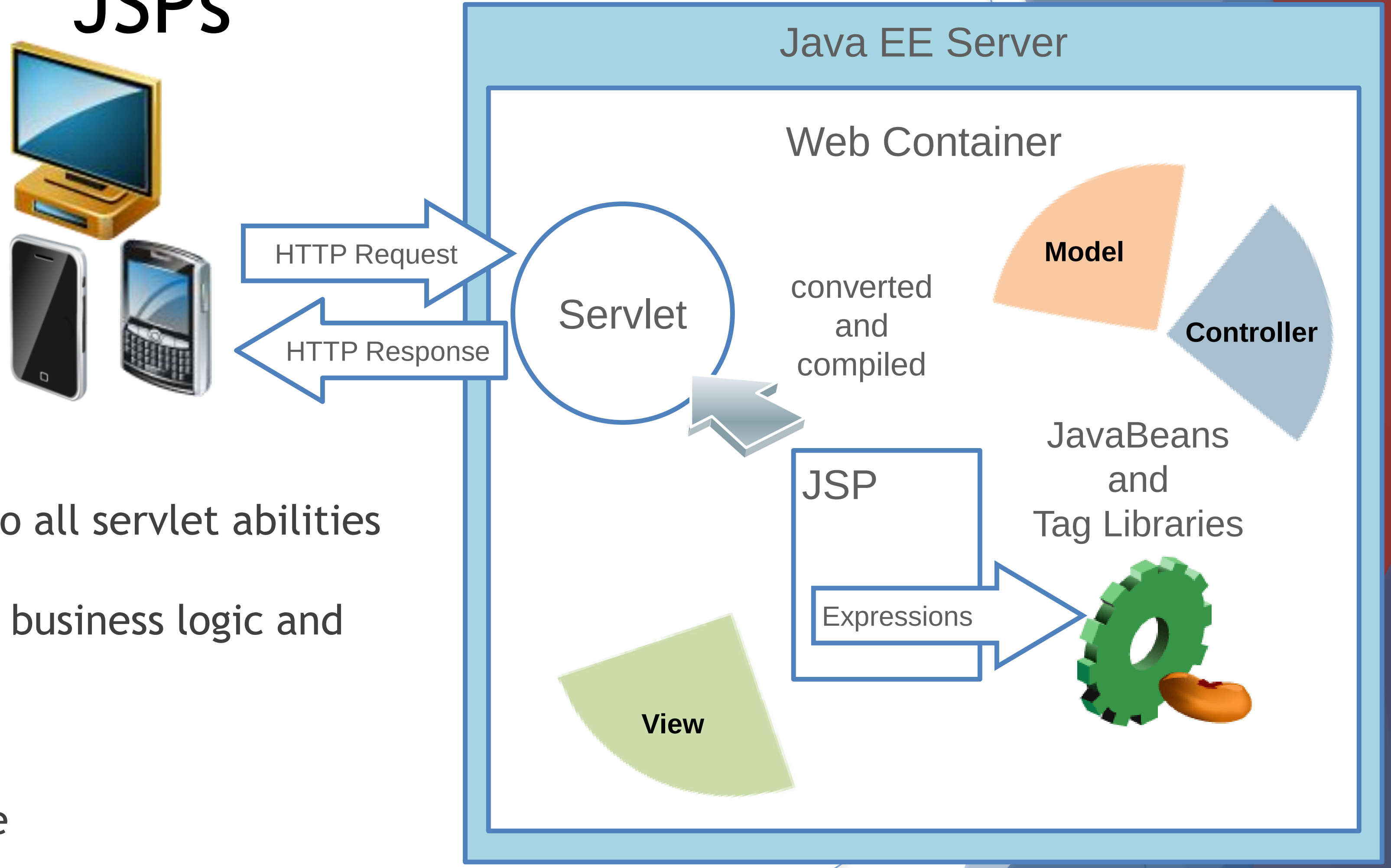
- Handle Events
- Send Commands to the Model
- Control View

Java EE Web Container Components: Servlets



- Servlets:
  - Are Java classes mapped to URLs
  - Are typically invoked via HTTP
  - Utilize request-response architectural approach
  - Can mix business logic and layout management

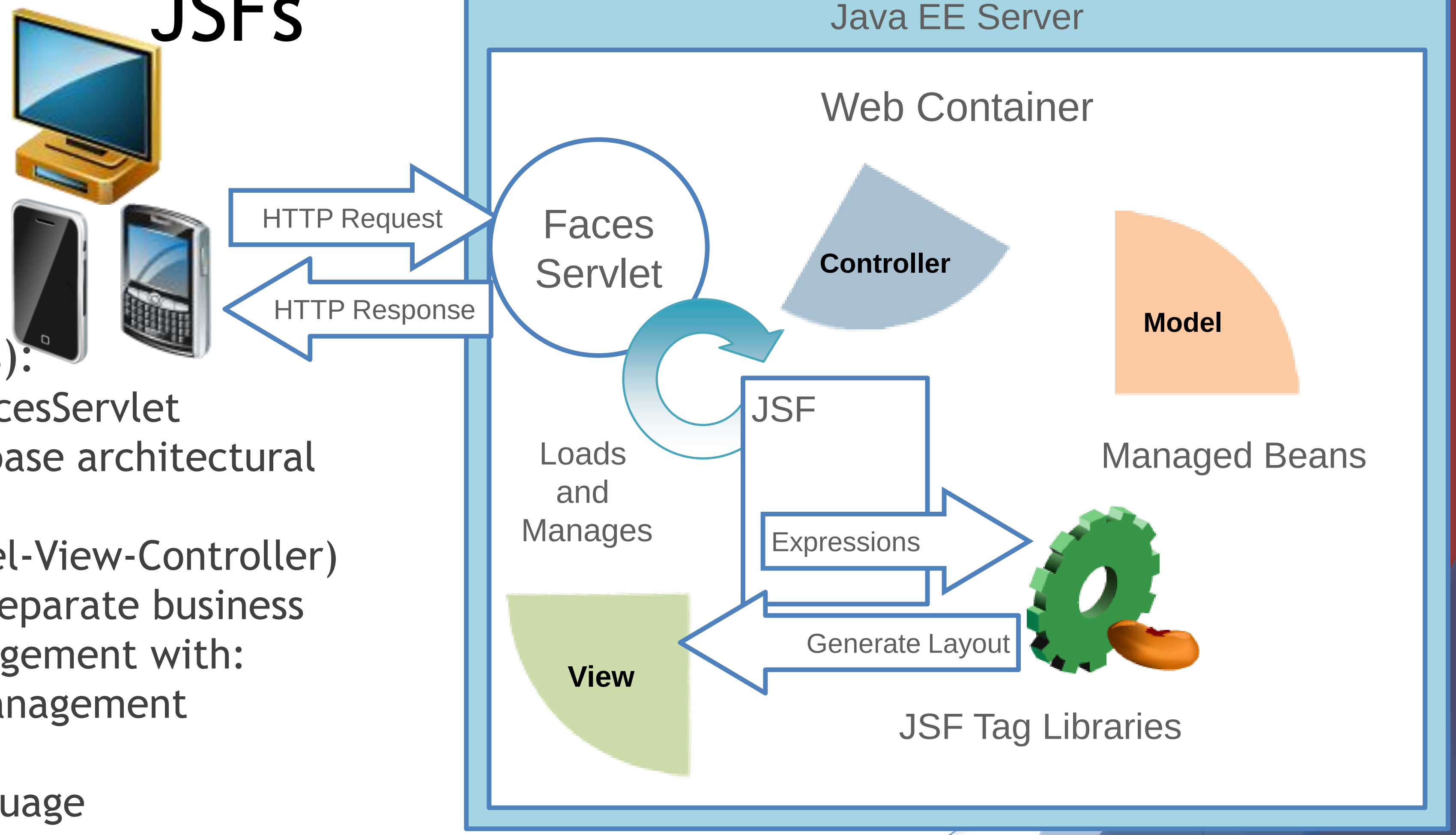
Java EE Web Container Components: JSPs



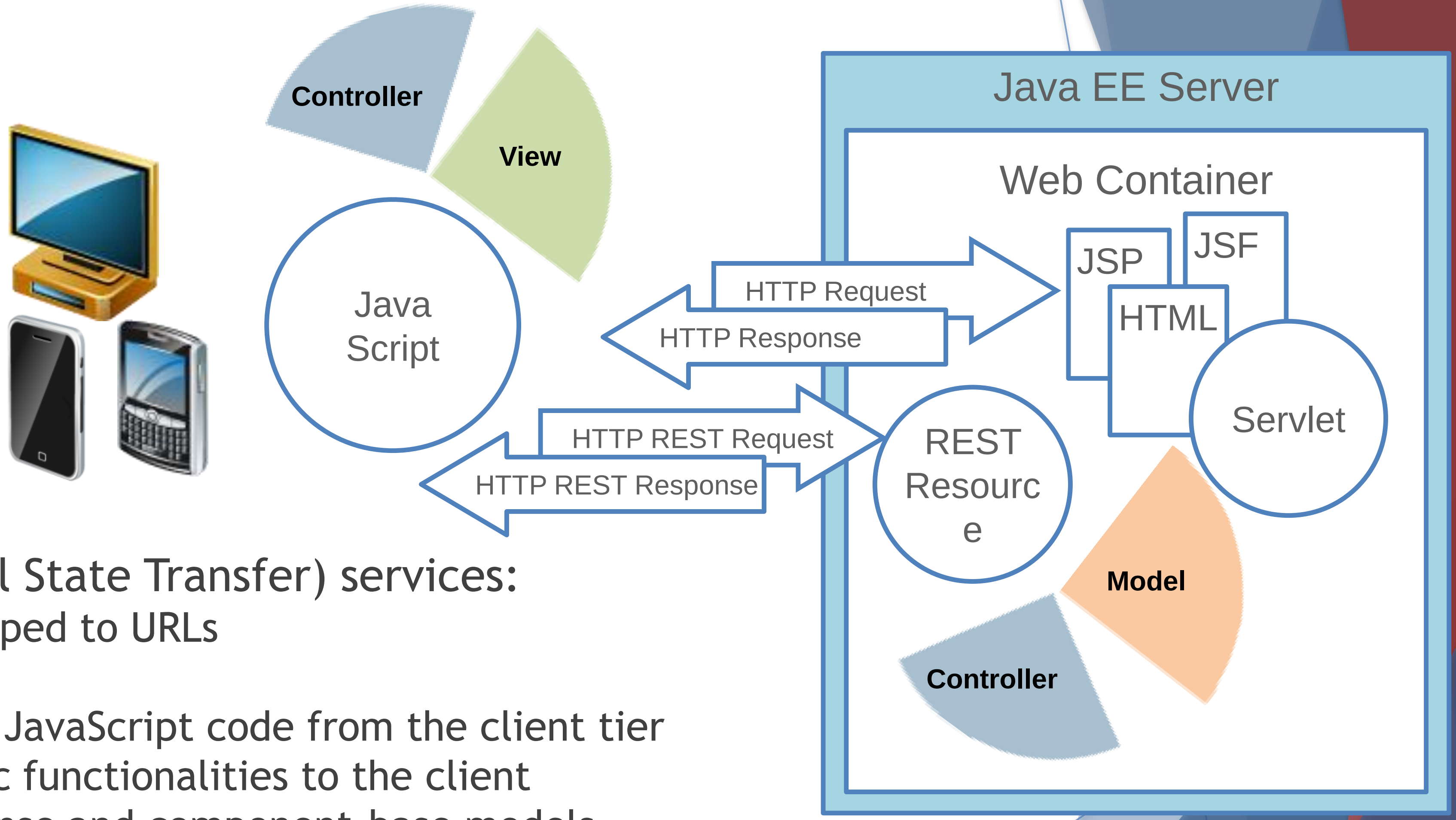
- JSPs (Java Server Pages):
  - Are turned into servlets (so all servlet abilities still apply)
  - Offer better separation of business logic and layout management with:
    - JavaBeans
    - Tag libraries
    - Expression Language

Java EE Web Container Components: JSFs

- JSFs (Java Server Faces):
  - Are interpreted by FacesServlet
  - Utilize a component-base architectural approach
  - Implement MVC (Model-View-Controller) architecture to fully separate business logic and layout management with:
    - JSF lifecycle management
    - Managed beans
    - Expression Language

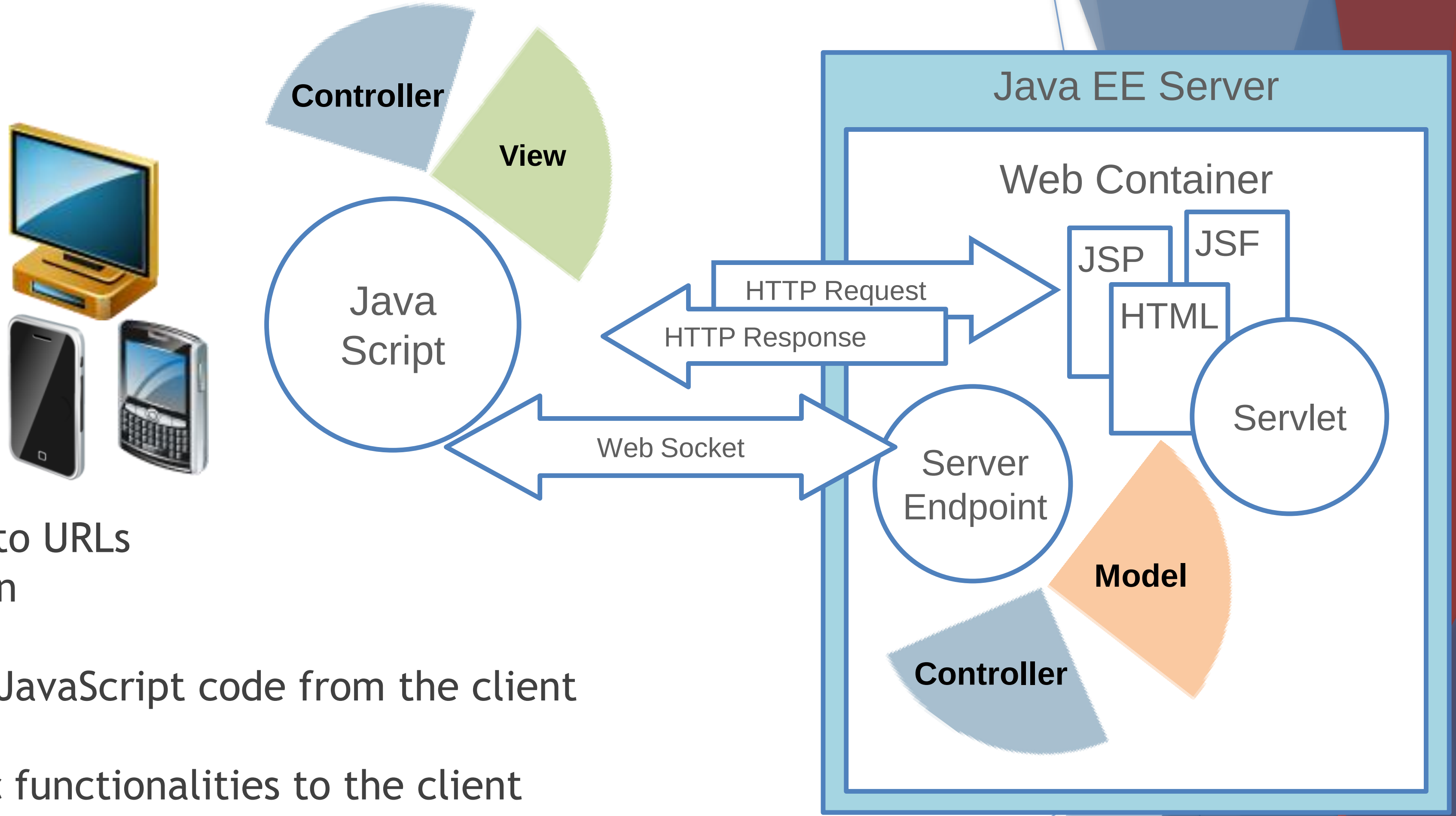


Java EE Web Container Components: REST Services



- REST (Representational State Transfer) services:
  - Are Java Classes mapped to URLs
  - Are invoked via HTTP
  - Are typically used by JavaScript code from the client tier
  - Present business-logic functionalities to the client
  - Utilize request-response and component-base models

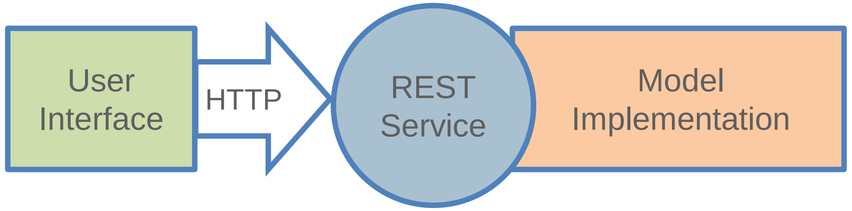
Java EE Web Container Components: Web Sockets

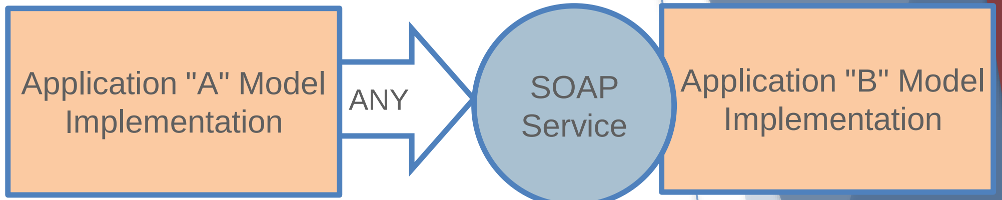


- Web sockets:
  - Java classes mapped to URLs
  - Full duplex connection
  - Can use server push
  - Are typically used by JavaScript code from the client tier
  - Present business-logic functionalities to the client
  - Utilize component-base models

Java EE 7 Web Services

- All Web Services provide business functionalities in a way that disguises their implementation.




Application "A" Model Implementation
ANY
SOAP Service
Application "B" Model Implementation

JAX-RS (REST services):

- Utilize HTTP protocol methods, such as:
  - GET
  - PUT
  - POST
  - DELETE
- Can transport any data, for example:
  - XML
  - JSON
- Typically used by browser and mobile UI

JAX-WS (SOAP services):

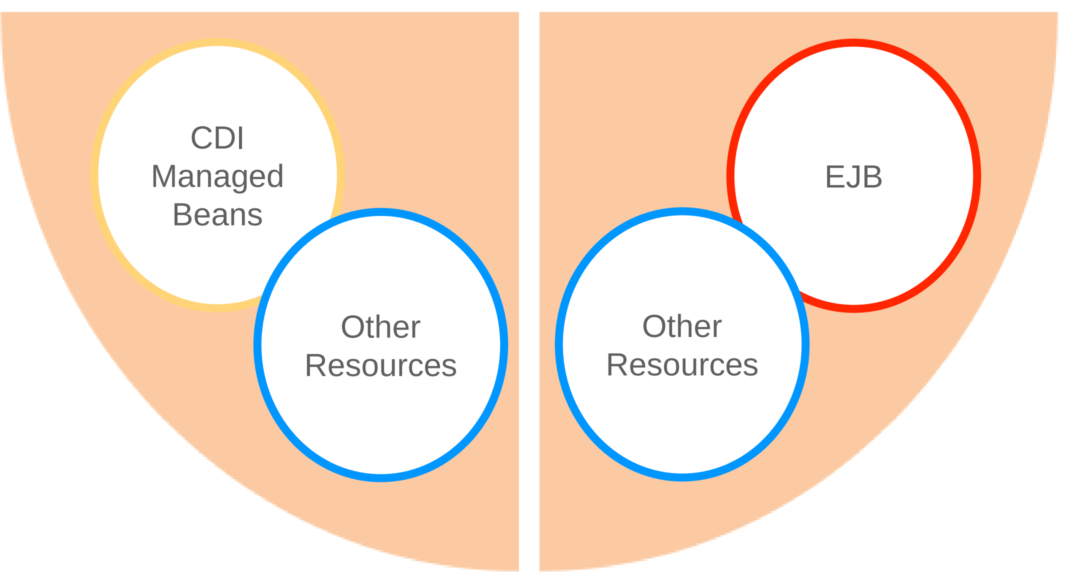
- Are transport protocol–independent
- Use standard WSDL descriptors
- Can transport XML described via XSD
- Provide a range of WS-* standard policies, such as:
  - WS-Security
  - WS-Reliability
  - WS-Addressing
  - WS-Transactions
- Typically used for system integration purposes and in SOA architecture

Java EE 7 Business Logic Handling Technologies

- CDI managed bean:
  - Life cycle determined by memory scope context: Request, View, Session, Application, Dependent, Conversation, and so on
  - Can be invoked only locally
- EJB (Enterprise JavaBean):
  - Life cycle determined by the type of bean:
    - Session Beans: Stateless, Stateful, Singleton
    - Message Beans: Message-Driven
  - Can be invoked locally or remotely
- Other resources:
  - EntityManager, JMS Queue or Topic, DataSource, EJBContext and so on represent container-managed resources.

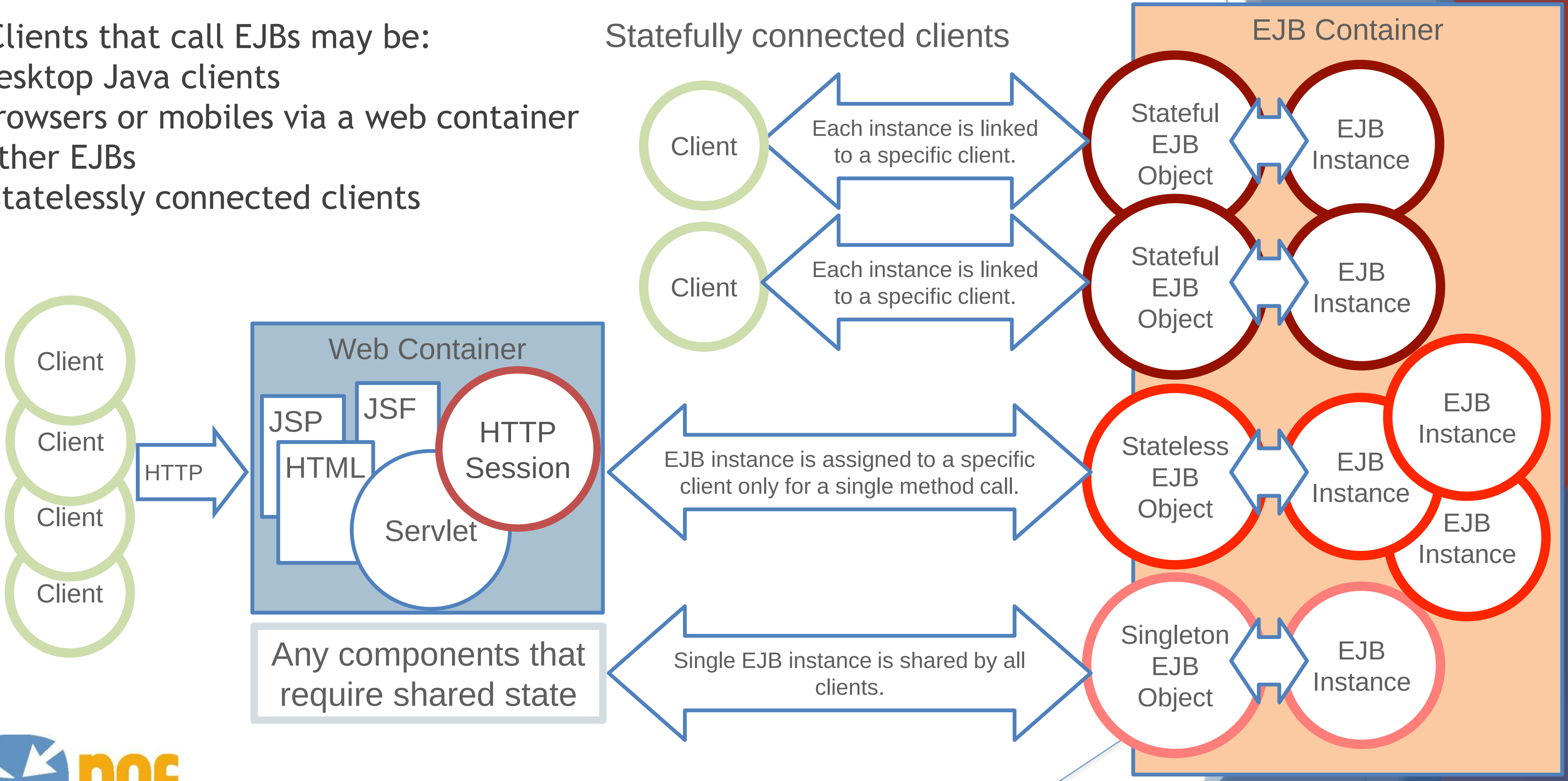
Model implementation:

- Contains your business logic
- Contained within CDI managed beans
- or Enterprise JavaBeans
- or both



Session EJB Types

- Clients that call EJBs may be:
  - Desktop Java clients
  - Browsers or mobiles via a web container
  - Other EJBs
- Statelessly connected clients



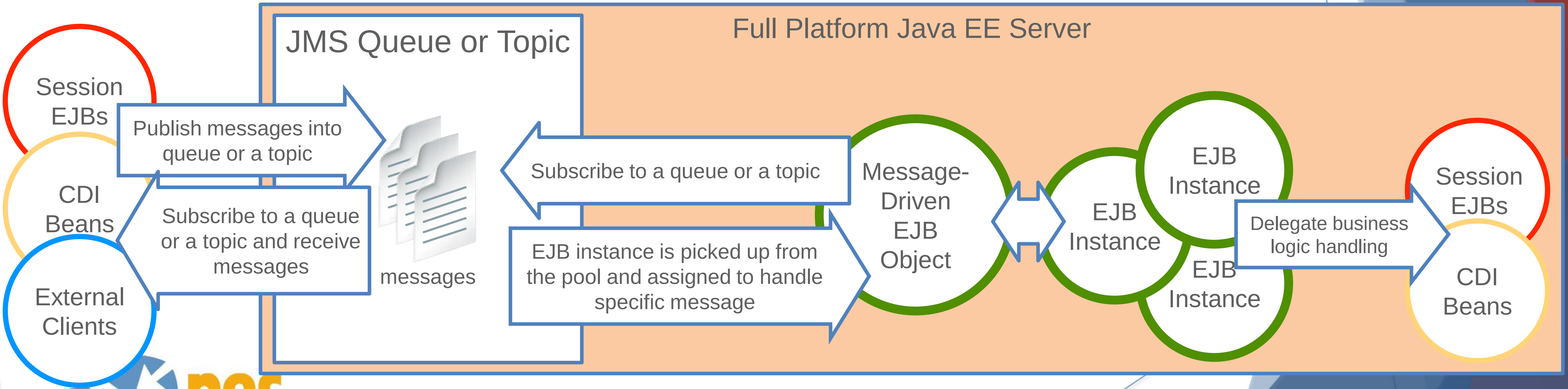
Statefully connected clients
EJB Container
Client
Each instance is linked to a specific client.
Stateful EJB Object
EJB Instance
Client
Each instance is linked to a specific client.
Stateful EJB Object
EJB Instance
Client
Client
Client
Client
HTTP
Web Container
JSP
JSF
HTML
Servlet
HTTP Session
EJB instance is assigned to a specific client only for a single method call.
Stateless EJB Object
EJB Instance
EJB Instance
EJB Instance
Any components that require shared state
Single EJB instance is shared by all clients.
Singleton EJB Object
EJB Instance

Message-Driven EJB

- Message producers and consumers:
  - Java EE components
  - External clients

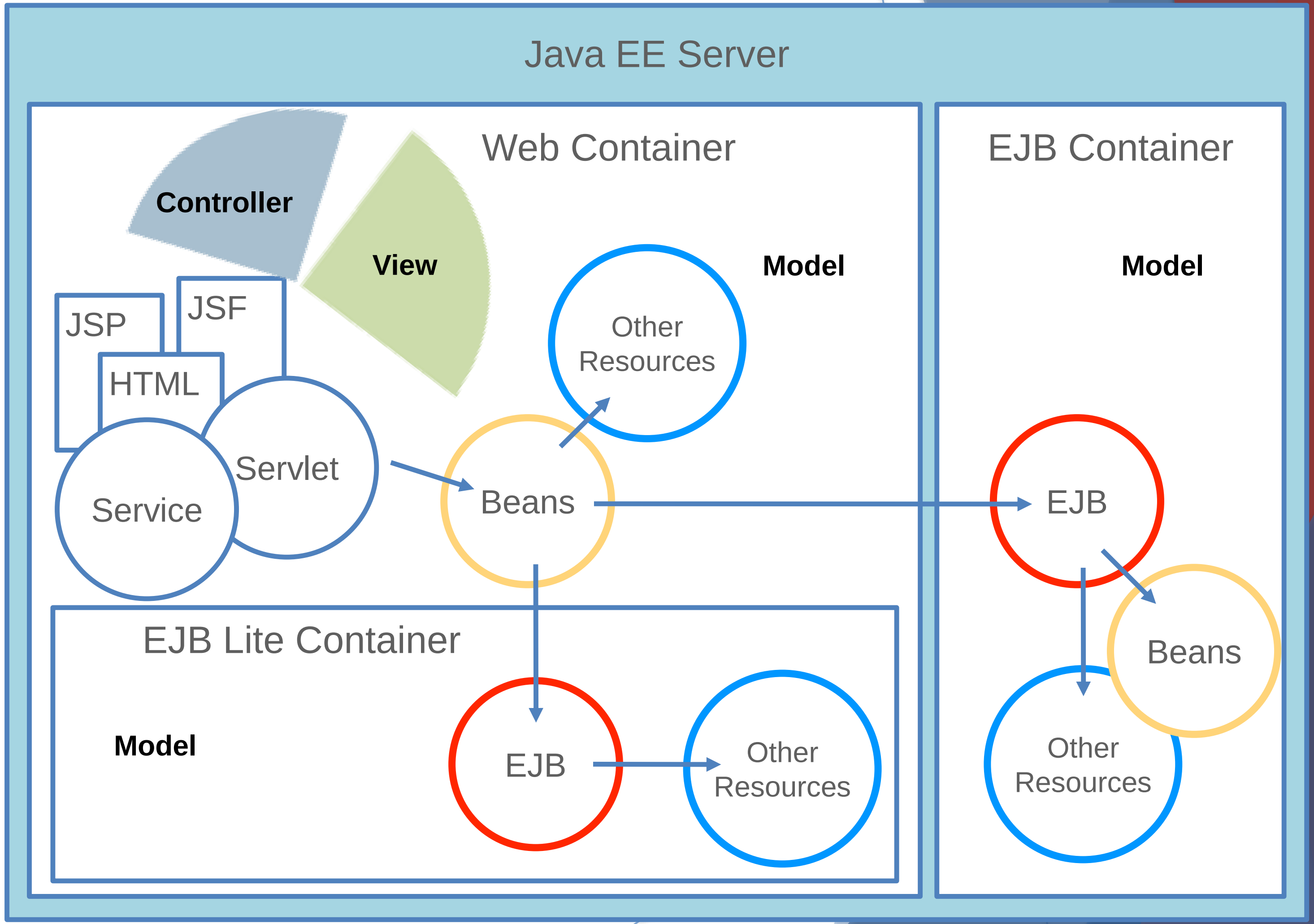
Message-driven bean:

- Stateless asynchronous message consumer
- Can be subscribed to receive messages from queues or topics
- Can never be directly invoked by a client
- Typically is responsible for acquiring, validating, and preparing messages before passing on the business logic handling classes


Session EJBs
CDI Beans
External Clients
Publish messages into queue or a topic
Subscribe to a queue or a topic and receive messages
JMS Queue or Topic
messages
Full Platform Java EE Server
Subscribe to a queue or a topic
EJB instance is picked up from the pool and assigned to handle specific message
Message-Driven EJB Object
EJB Instance
EJB Instance
EJB Instance
Delegate business logic handling
Session EJBs
CDI Beans

Assembling Application Components with CDIs

- Context Dependency Injections and annotations are used to reference objects such as:
  - POJOs
  - JSF managed beans or CDI beans
  - EJBs
  - Other resources such as
    - EntityManager
    - JMS queues or topics
    - EJBContext
    - ServletContext
    - And so on



JSF Managed Beans, CDI Beans, EJBs

- Enterprise JavaBeans (EJBs):
  - Described by the javax.ejb package
  - Can be used in the EJB Java EE container
  - Can be called locally and remotely
  - Can be stateful and be passivated
  - Can work with timers
  - Can be invoked asynchronously

CDI beans:

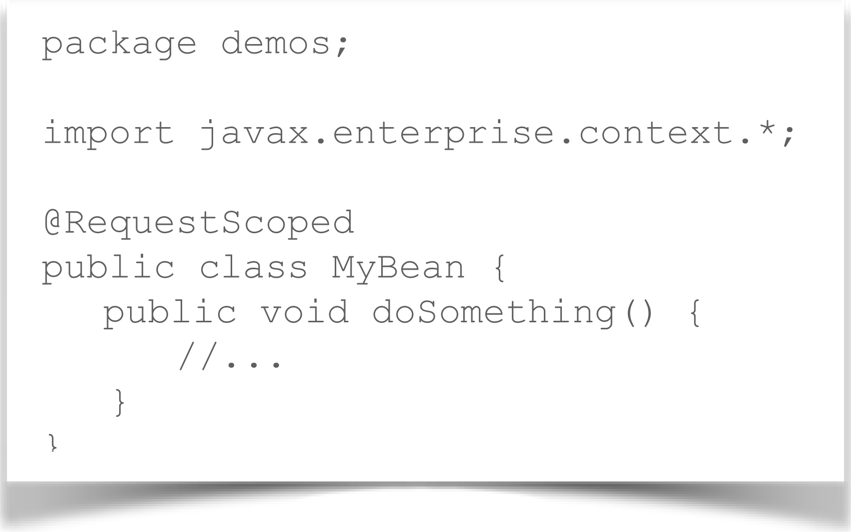
- Described by the javax.enterpise.context package
- Can be used in all Java EE Containers—not limited to JSF
- Support interceptors, events, and so on, and are more flexible than JSF Managed Beans
- In addition to annotations, may use beans.xml deployment descriptor

JSF managed beans:

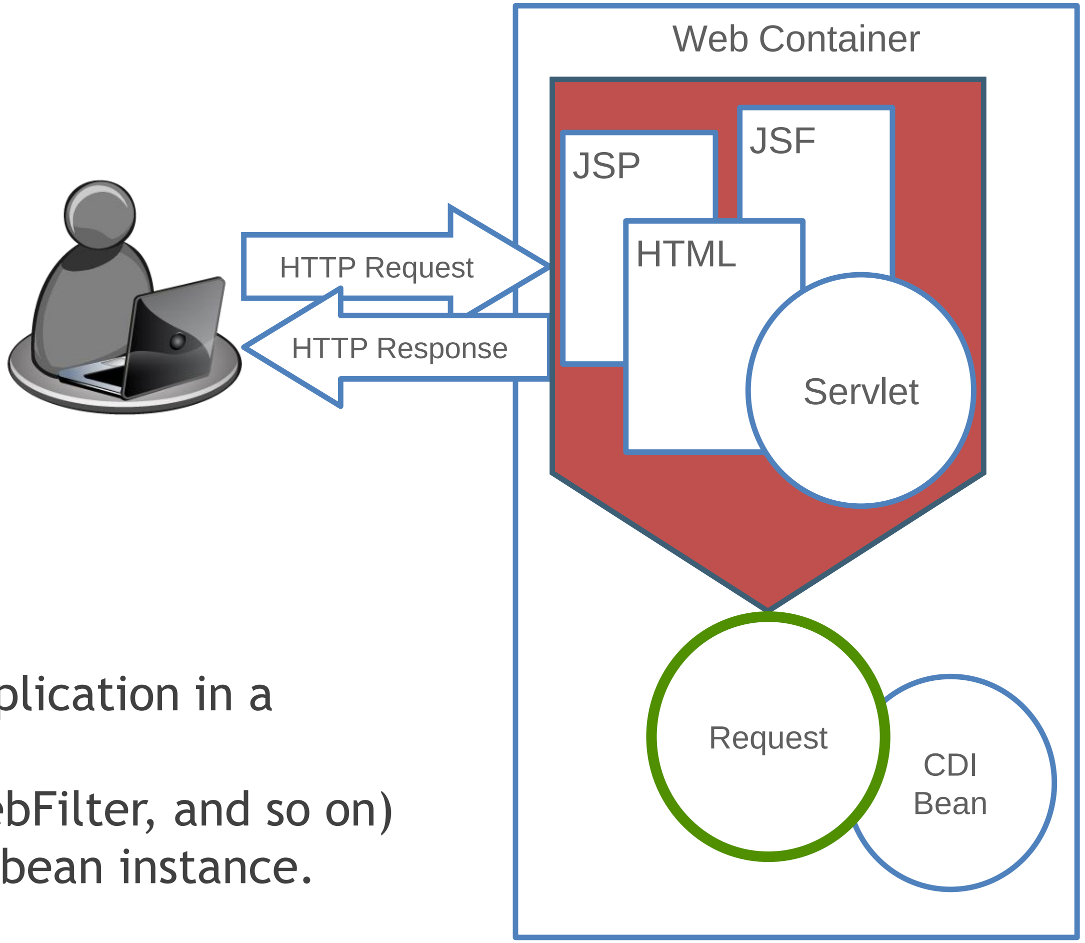
- Described by the javax.faces.bean package
- Used in the web container by JSF Components
- CDI Beans "evolved" from JSF managed beans

❖ The following examples demonstrate the use of CDI beans in a context of a web container. However, they could also be used in EJB container.

Request Scope

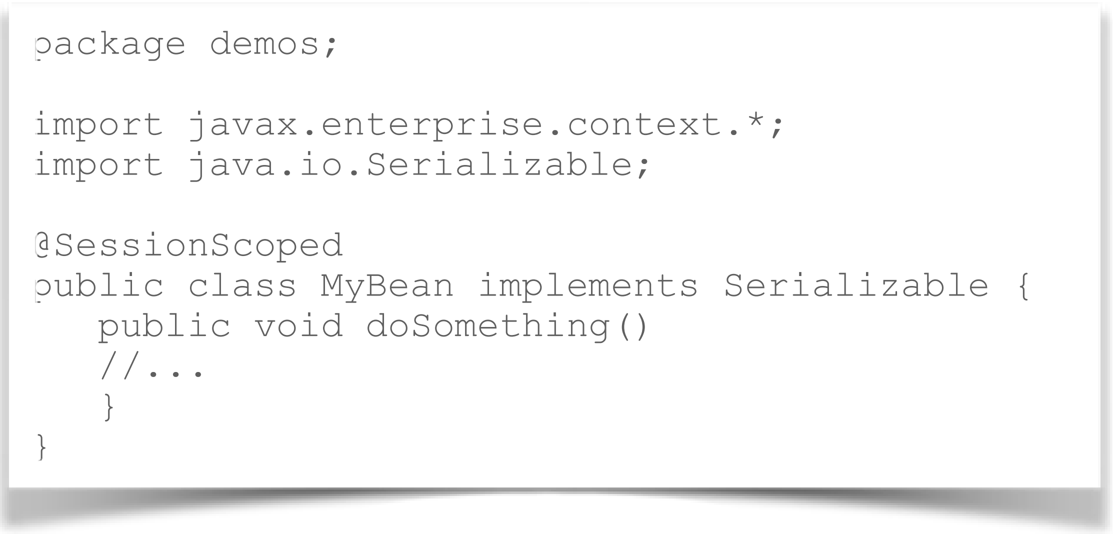


```
package demos;

import javax.enterprise.context.*;

@RequestScoped
public class MyBean {
   public void doSomething() {
       //...
    }
}
```

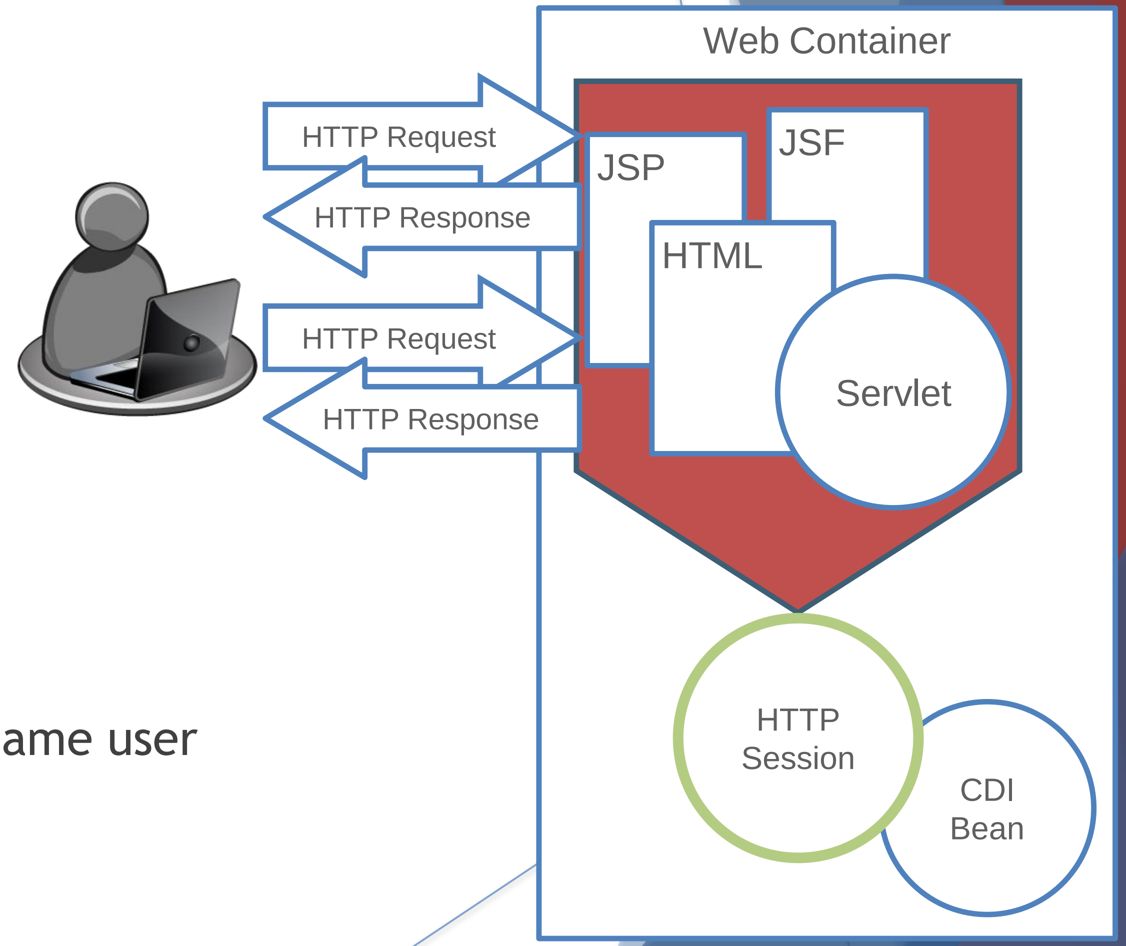


- Request scoped beans:
  - Instantiated once per user's interaction with a web application in a single HTTP request
  - More than one server component (servlet, JSP, JSF, WebFilter, and so on) can handle the same client request, sharing the same bean instance.

Session Scope

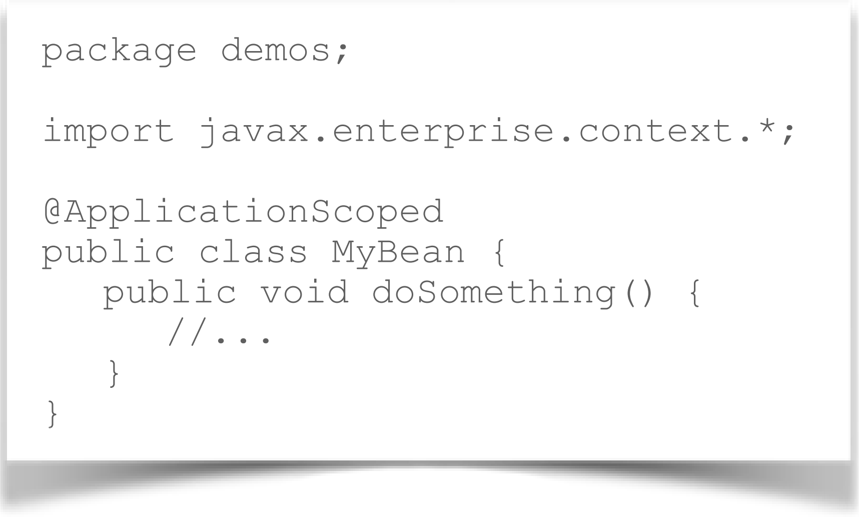


```
package demos;

import javax.enterprise.context.*;
import java.io.Serializable;

@SessionScoped
public class MyBean implements Serializable {
   public void doSomething()
   //...
   }
}
```

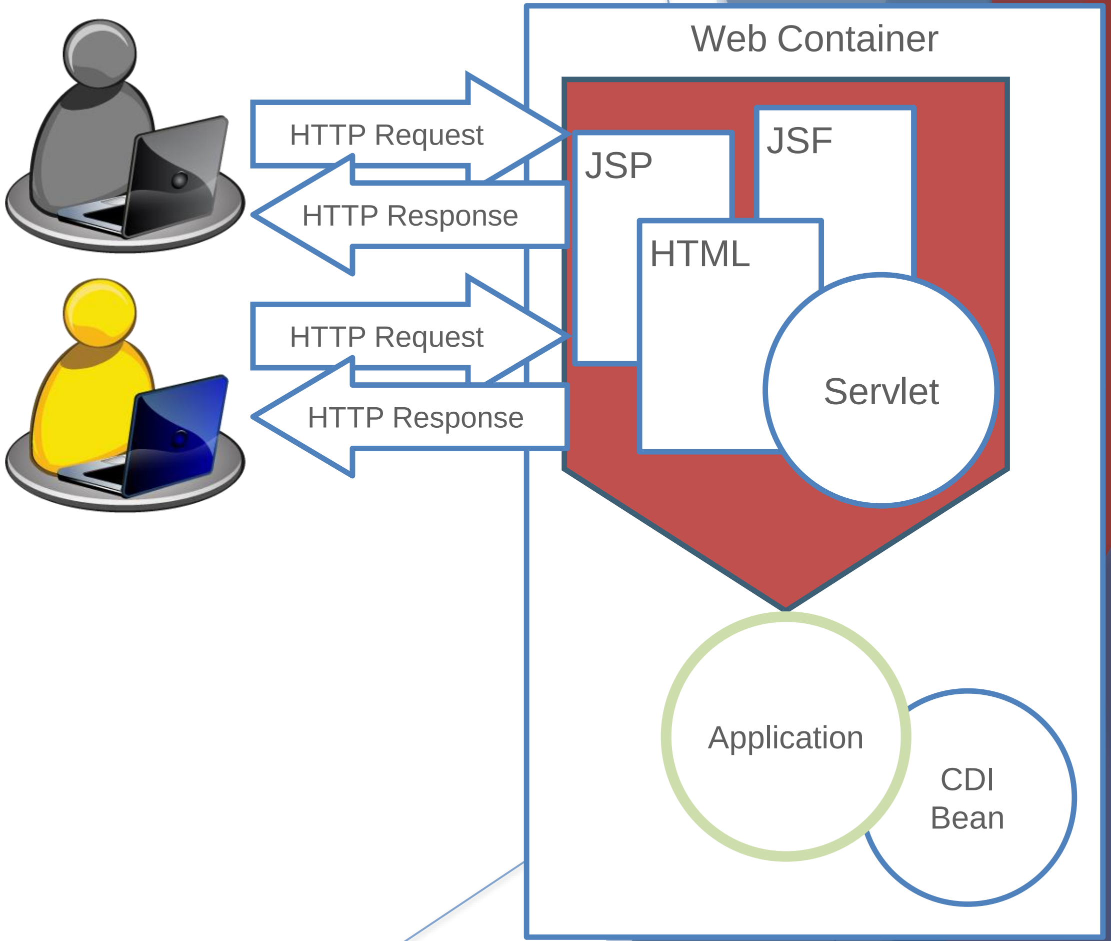


- Session Scoped Beans:
  - Instantiated once per user session
  - Shared across multiple HTTP requests from the same user

Application Scope

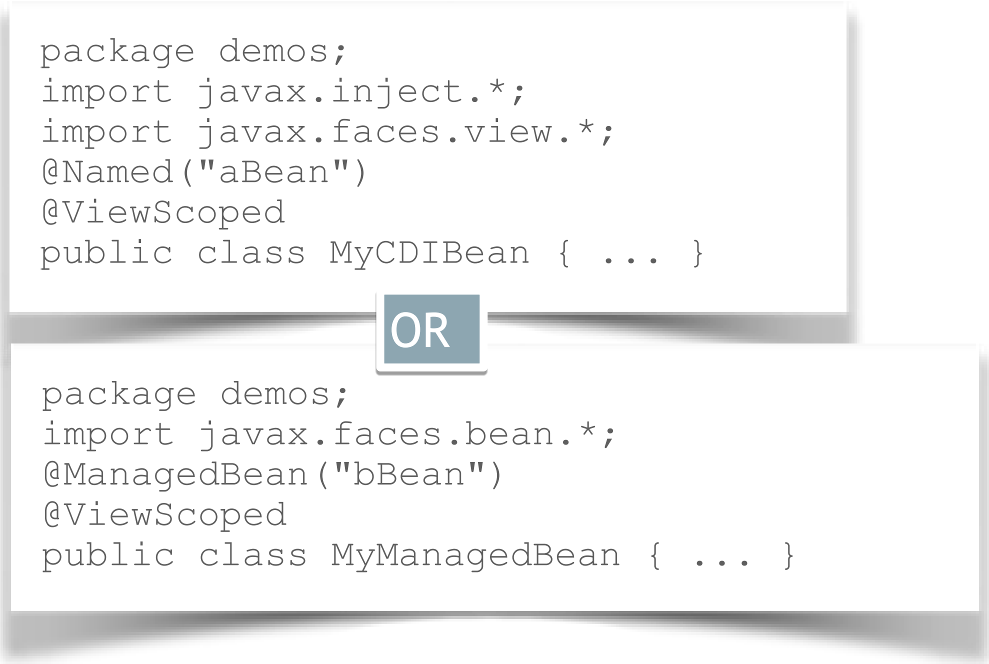


```
package demos;

import javax.enterprise.context.*;

@ApplicationScoped
public class MyBean {
   public void doSomething() {
      //...
   }
}
```



- Application Scoped Beans:
  - Instantiated once per application
  - Shared by all application users

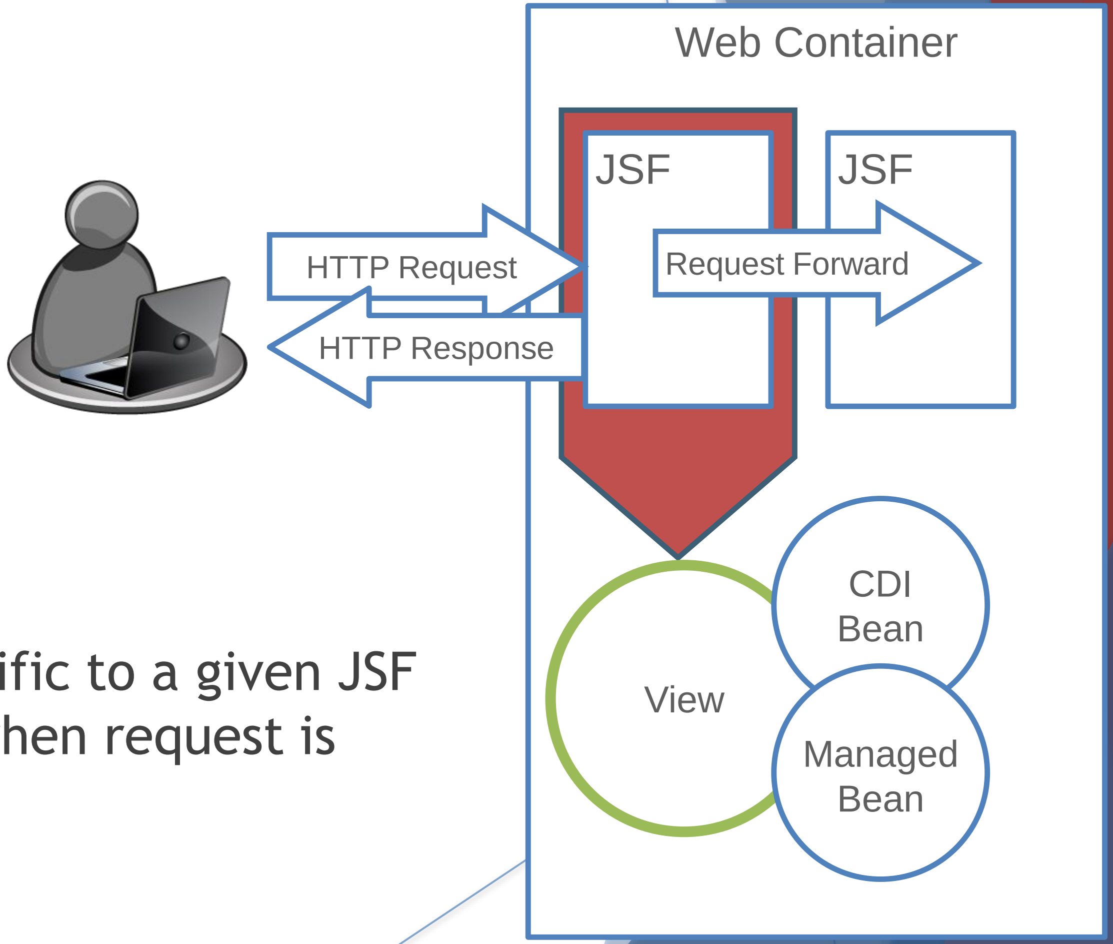
View Scope



```
package demos;
import javax.inject.*;
import javax.faces.view.*;
@Named("aBean")
@ViewScoped
public class MyCDIBean { ... }
```

OR

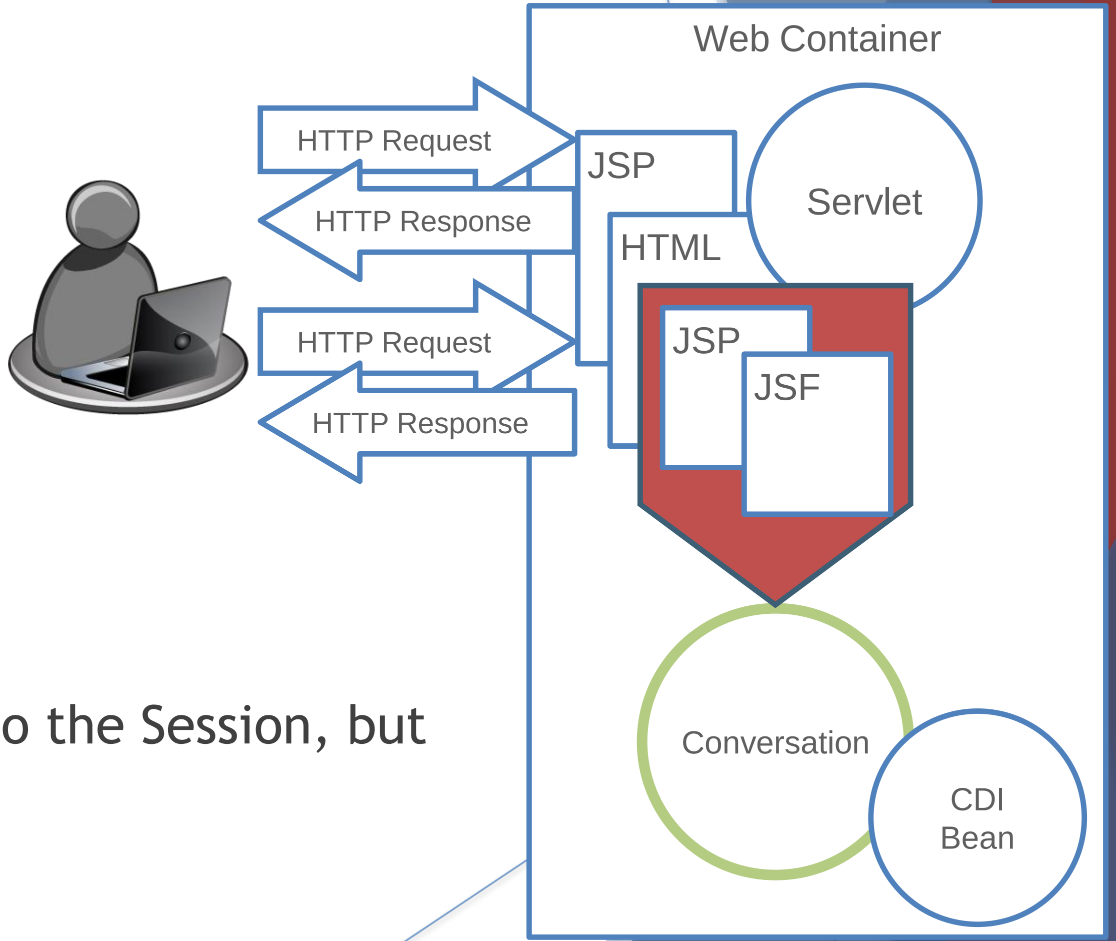
```
package demos;
import javax.faces.bean.*;
@ManagedBean("bBean")
@ViewScoped
public class MyManagedBean { ... }
```



- View Scoped is a JSF-specific scope that is specific to a given JSF page. It is not "shared" with another JSF page when request is forwarded to it during navigation.

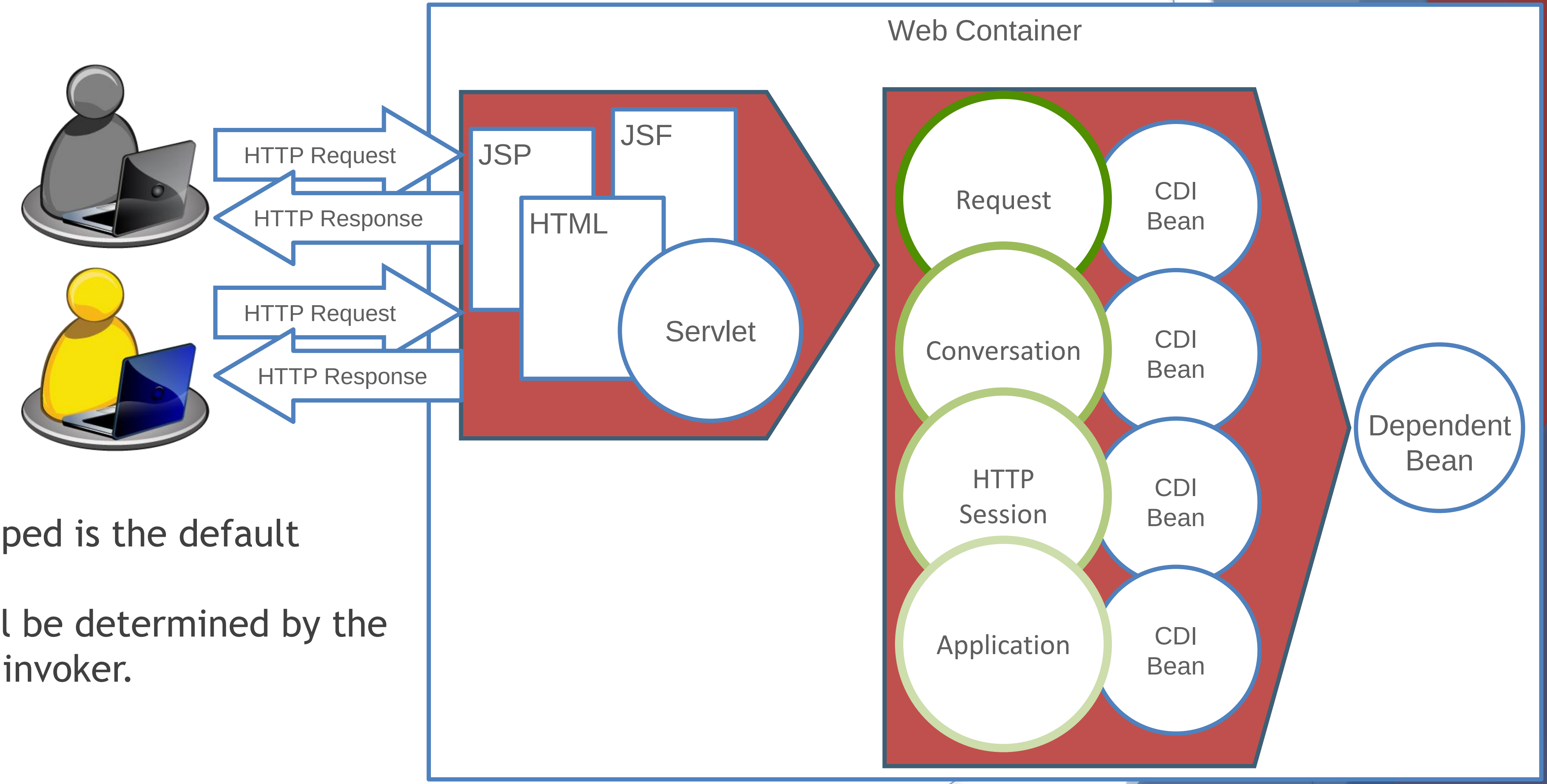
Conversation Scope

```
package demos;

import javax.enterprise.context.*;
import java.io.Serializable;

@ConversationScoped
public class MyBean implements Serializable {
   @Inject
  private Conversation conversation;
  public void startConversation() {
     conversation.begin();
   }
   public void endConversation() {
     conversation.end();
   }
}
```



- Conversation Scoped is a custom scope similar to the Session, but can be limited to specific part of application.

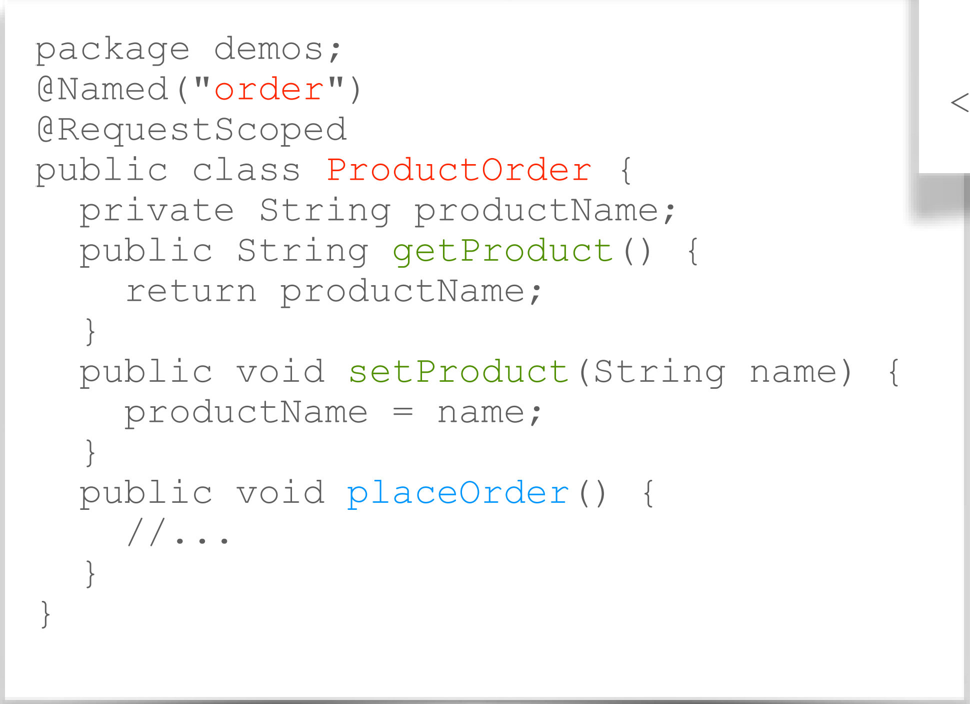
Dependent Scope



- Dependent Scoped is the default scope.
- Bean scope will be determined by the context of the invoker.

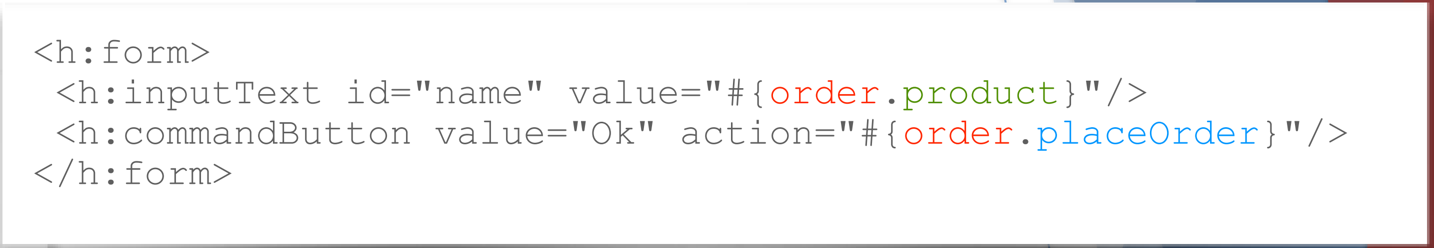
Injecting Beans

ProductOrder bean definition with "Product" property, "placeOrder" operation, and optional alias "order":



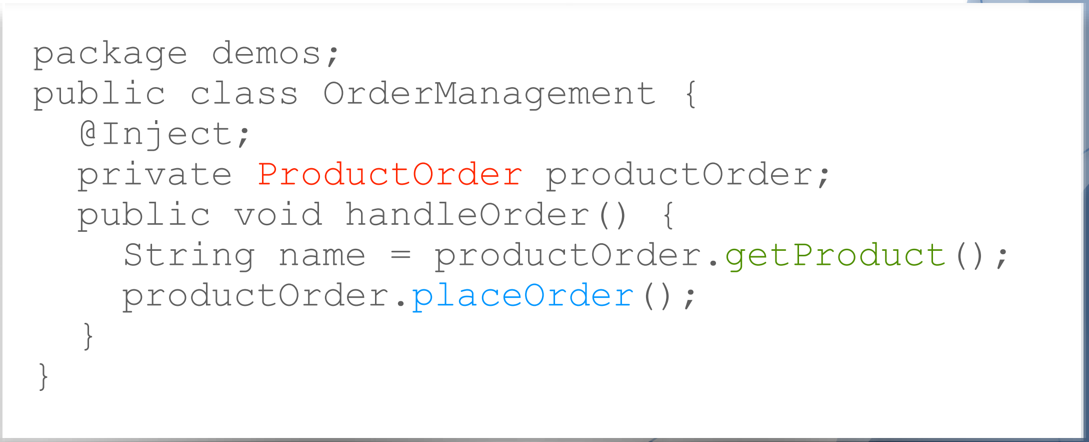
```
package demos;
@Named("order")
@RequestScoped
public class ProductOrder {
  private String productName;
  public String getProduct() {
    return productName;
  }
  public void setProduct(String name) {
    productName = name;
  }
  public void placeOrder() {
    //...
  }
}
```

Injection of the ProductOrder bean into JSF page using "order" alias:



```
<h:form>
 <h:inputText id="name" value="#{order.product}"/>
 <h:commandButton value="Ok" action="#{order.placeOrder}"/>
</h:form>
```

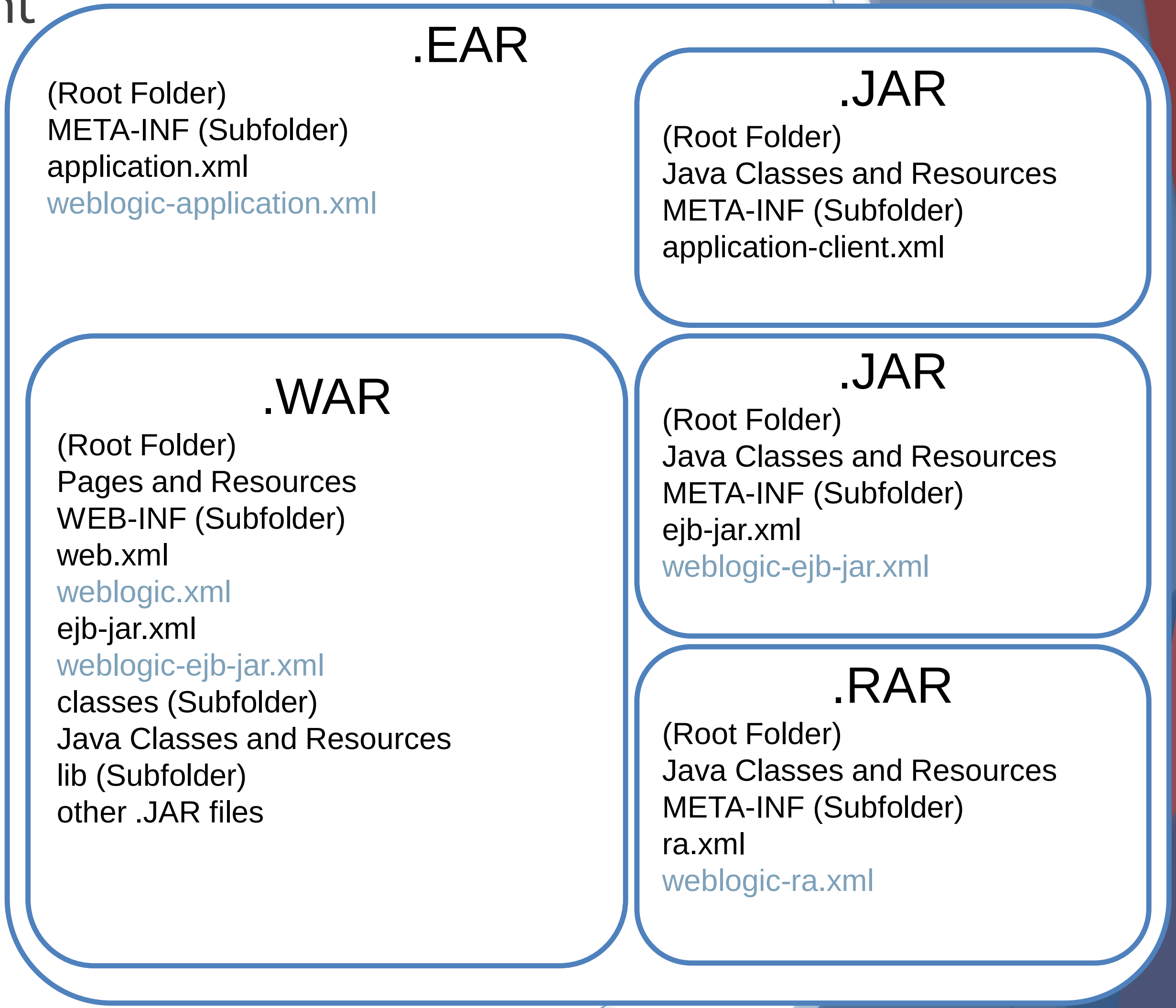
Injection of the ProductOrder bean into OrderManagement class:



```
package demos;
public class OrderManagement {
  @Inject;
  private ProductOrder productOrder;
  public void handleOrder() {
    String name = productOrder.getProduct();
    productOrder.placeOrder();
  }
}
```

Java EE Packaging and Deployment

- JSR 88: Java EE Application Deployment
  - Enterprise Archive
    - Packaged as EAR files
    - Contains other modules
  - Web Module - WAR files
    - Packaged as WAR files
    - Contains web content such as HTML, servlet, JSP, JSF etc..
    - May contain library or EJB JAR files
  - EJB Module
    - Packaged as JAR files
    - Contains Enterprise JavaBeans
  - Resource Adapter Module
    - Packaged as RAR files
    - Contains JCA adapters
  - Application Client Module
    - Packaged as JAR files
    - Contains Java client applications



.EAR

(Root Folder)
META-INF (Subfolder)
application.xml
weblogic-application.xml

.JAR

(Root Folder)
Java Classes and Resources
META-INF (Subfolder)
application-client.xml

.WAR

(Root Folder)
Pages and Resources
WEB-INF (Subfolder)
web.xml
weblogic.xml
ejb-jar.xml
weblogic-ejb-jar.xml
classes (Subfolder)
Java Classes and Resources
lib (Subfolder)
other .JAR files

.JAR

(Root Folder)
Java Classes and Resources
META-INF (Subfolder)
ejb-jar.xml
weblogic-ejb-jar.xml

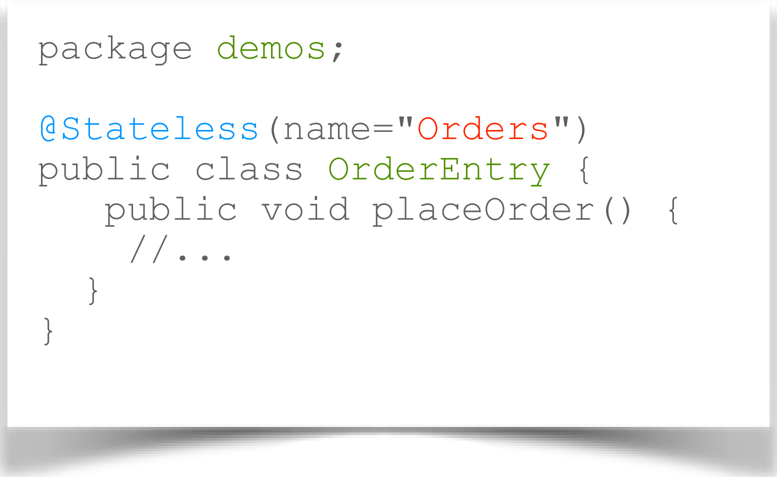
.RAR

(Root Folder)
Java Classes and Resources
META-INF (Subfolder)
ra.xml
weblogic-ra.xml

Annotations or Deployment Descriptors

- In Java EE 7, deployment descriptors are optional - a developer may use annotations instead.
- This example shows an EJB component described both ways.

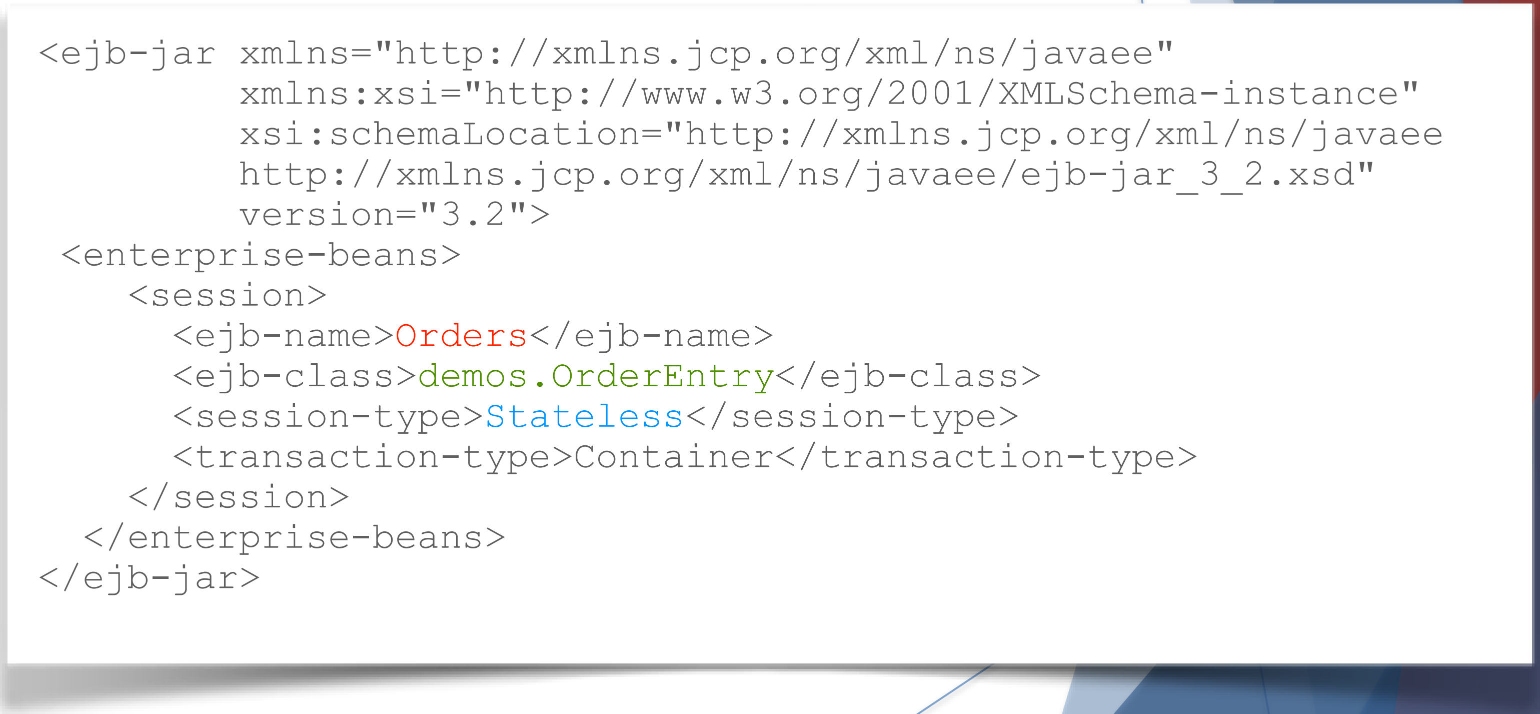
Using Annotations:



```
package demos;

@Stateless(name="Orders")
public class OrderEntry {
   public void placeOrder() {
    //...
  }
}
```

These two options are available for all Java EE components.

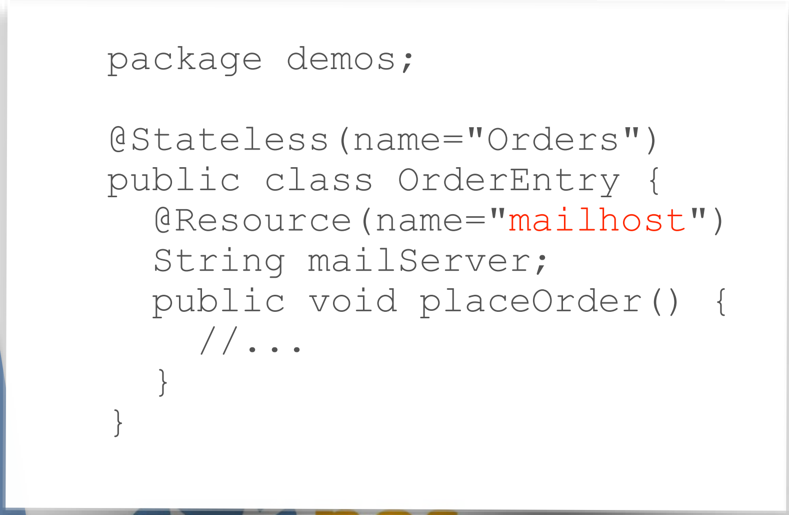
Using ejb-jar.xml Deployment Descriptor:



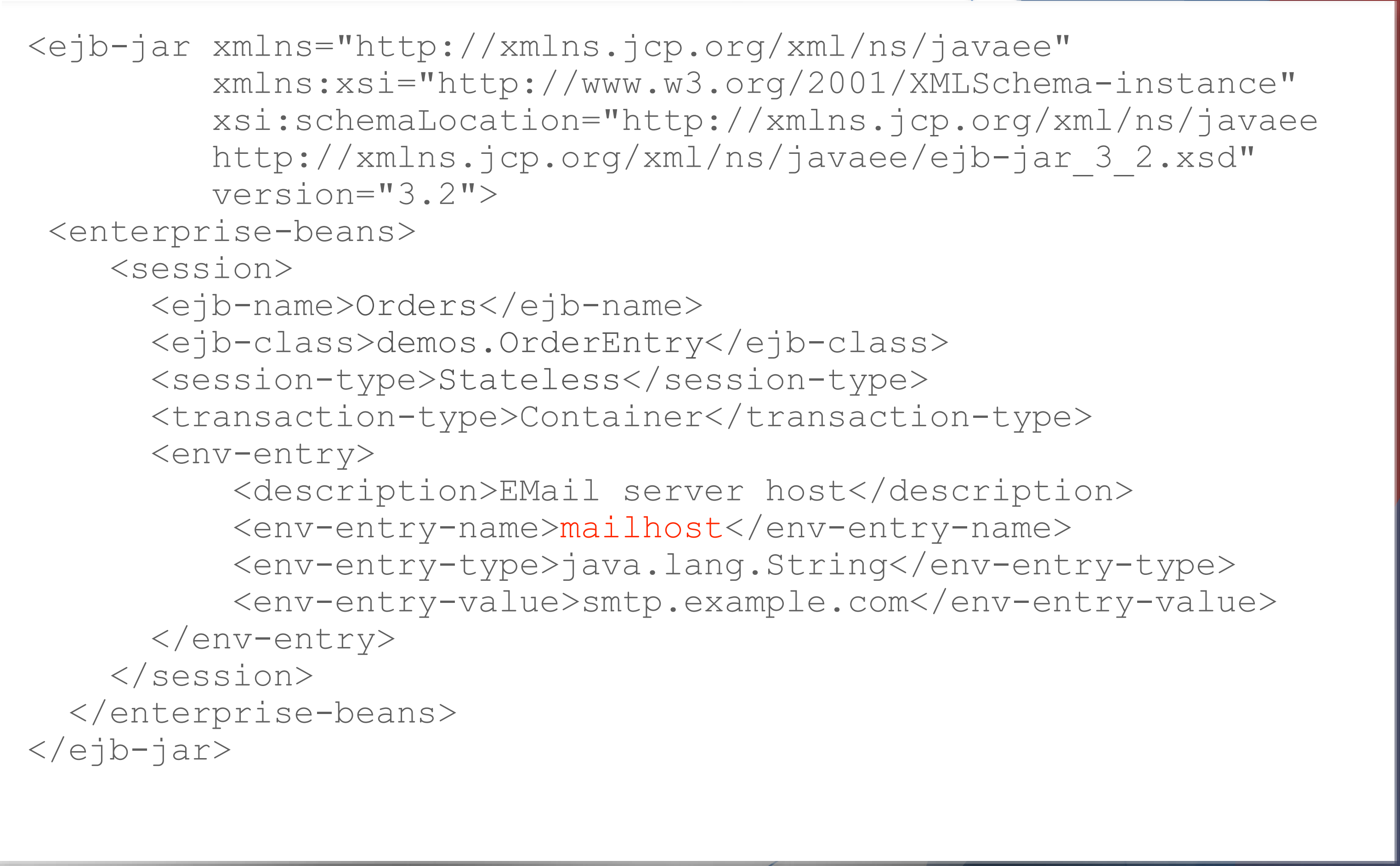
```
<ejb-jar xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
         http://xmlns.jcp.org/xml/ns/javaee/ejb-jar_3_2.xsd"
         version="3.2">
 <enterprise-beans>
    <session>
      <ejb-name>Orders</ejb-name>
      <ejb-class>demos.OrderEntry</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
    </session>
  </enterprise-beans>
</ejb-jar>
```

Annotations with Deployment Descriptors

- Deployment descriptors and annotation can be used together.
  - Annotations are convenient.
  - Descriptors are flexible.
  - Properties and behaviors of all
  - Java EE components can be
  - adjusted via XML descriptors
  - without changing annotations
  - in the source code.



```java
package demos;

@Stateless(name="Orders")
public class OrderEntry {
  @Resource(name="mailhost")
  String mailServer;
  public void placeOrder() {
    //...
  }
}
```



```xml
<ejb-jar xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
         http://xmlns.jcp.org/xml/ns/javaee/ejb-jar_3_2.xsd"
         version="3.2">
 <enterprise-beans>
    <session>
      <ejb-name>Orders</ejb-name>
      <ejb-class>demos.OrderEntry</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
      <env-entry>
          <description>EMail server host</description>
          <env-entry-name>mailhost</env-entry-name>
          <env-entry-type>java.lang.String</env-entry-type>
          <env-entry-value>smtp.example.com</env-entry-value>
      </env-entry>
    </session>
  </enterprise-beans>
</ejb-jar>
```

Java Naming Directory Interface Objects

- JNDI is used to catalog various types of objects, such as:
  - EJBs
  - JMS queues, topics, connection factories
  - Data sources
  - LDAP objects
  - etc...

```
@Stateless
public class OrderManagement {...}
```

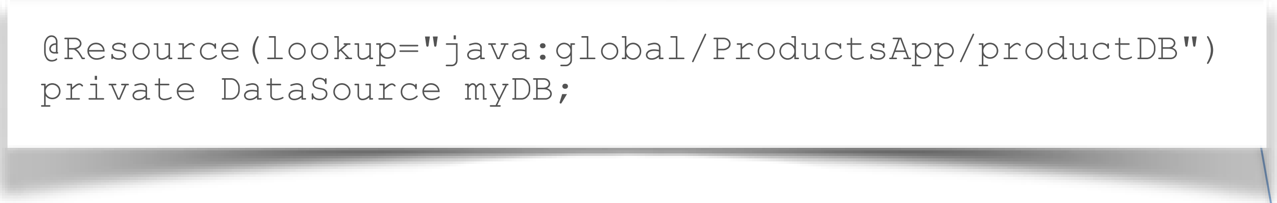
```
@DataSourceDefinition (
      name="java:global/ProductsApp/productDB",
      className="org.apache.derby.jdbc.ClientDataSource",
      url="jdbc:derby://localhost:1527/ProductDB",
      user="...",
      password="...",
      databaseName="ProductDB"
   )
```

Global JNDI naming convention:

```
java:global/<application-name>/<module-name>/<bean-name>
                      java:app/<module-name>/<bean-name>
                                 java:module/<bean-name>
                                             <bean-name>
```

Container-Managed Injections

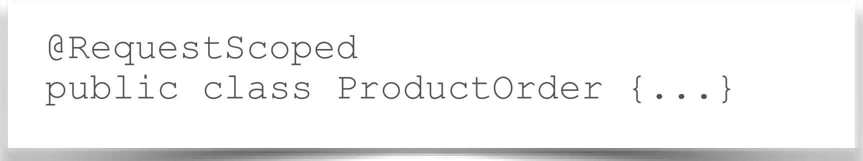
- Components in Java EE environment use annotations to inject resources and dependencies.
- Injecting Resources:
  - Data sources
  - JMS queues, topics
  - etc...



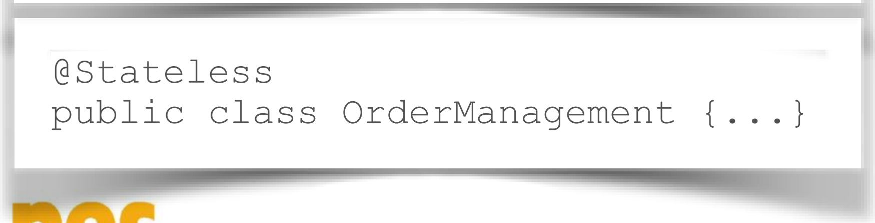
```
@Resource(lookup="java:global/ProductsApp/productDB")
private DataSource myDB;
```

Defining Components:

- CDI beans
- Enterprise JavaBeans



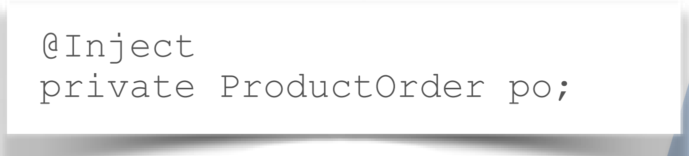
```
@RequestScoped
public class ProductOrder {...}
```



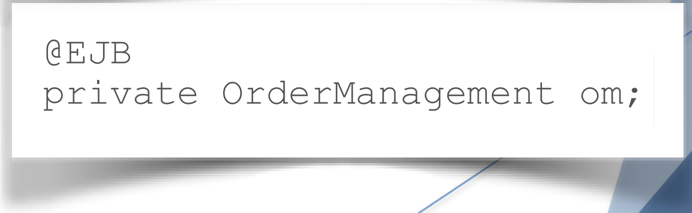
```
@Stateless
public class OrderManagement {...}
```

Injecting Dependencies:

- CDI beans
- Enterprise JavaBeans



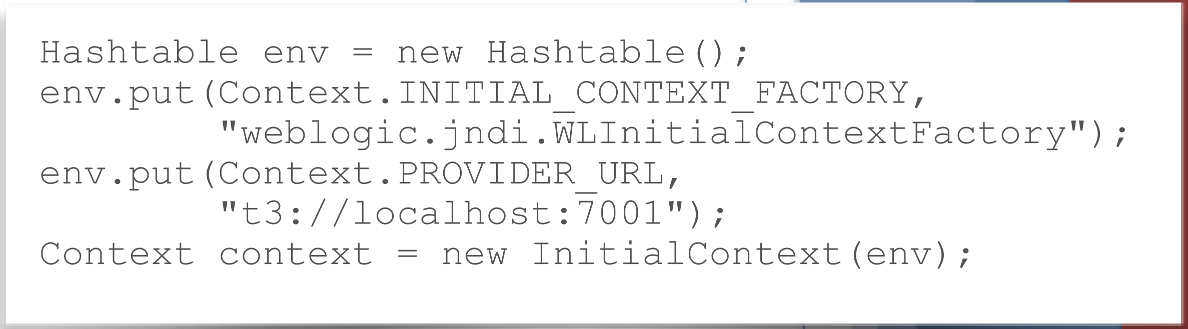
```
@Inject
private ProductOrder po;
```



```
@EJB
private OrderManagement om;
```

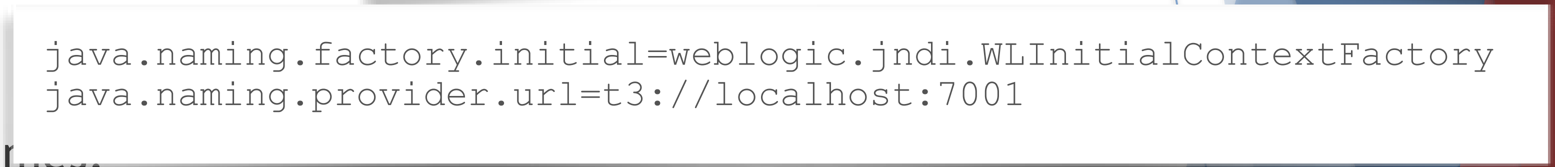
JDNI Lookups

- Components outside of the Java EE container environment perform explicit JNDI lookups.
- Create Initial Context Object to reference JNDI:
  - Using Java code to set server context properties:

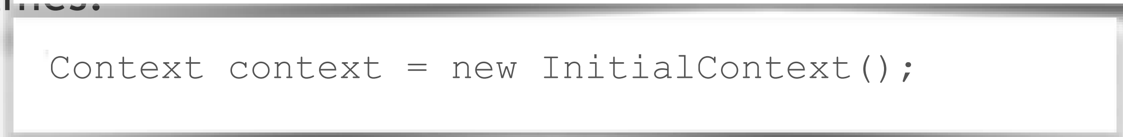


```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
        "weblogic.jndi.WLInitialContextFactory");
env.put(Context.PROVIDER_URL,
        "t3://localhost:7001");
Context context = new InitialContext(env);
```

  - Or using the `jdni.properties` file:

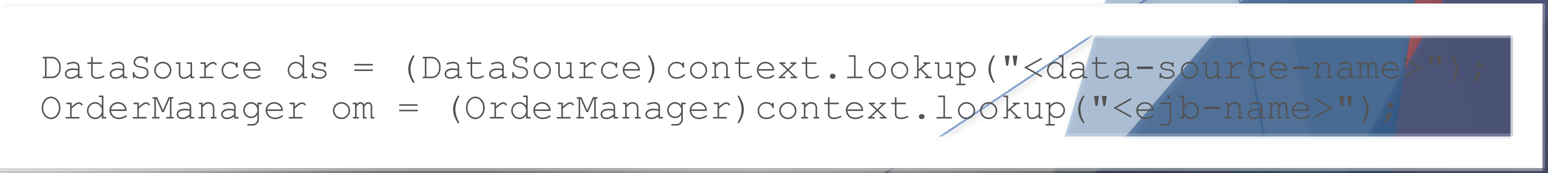


```
java.naming.factory.initial=weblogic.jndi.WLInitialContextFactory
java.naming.provider.url=t3://localhost:7001
```



```
Context context = new InitialContext();
```

Perform lookups using JDNI object names:



```
DataSource ds = (DataSource)context.lookup("<data-source-name>");
OrderManager om = (OrderManager)context.lookup("<ejb-name>");
```

Summary

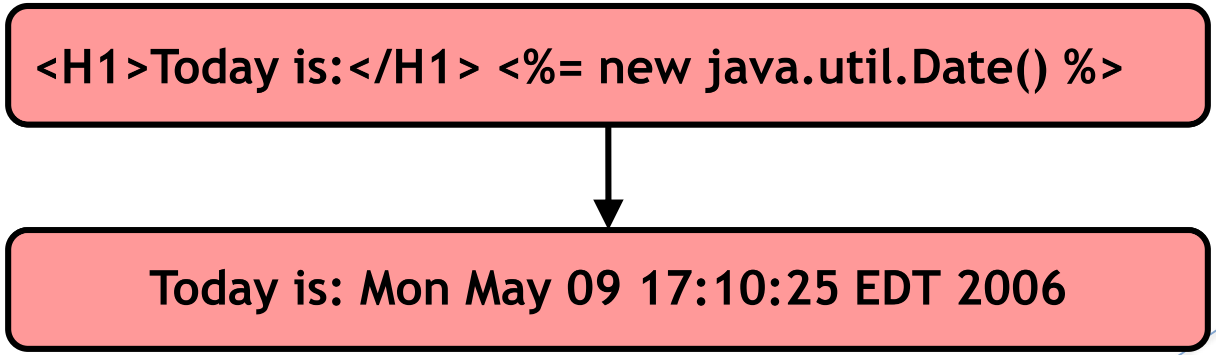
- In this lesson, you studied the fundamentals of Java EE 7 Architecture and its components. After completing this lesson, you should have learned how to describe:
  - Standards, containers, APIs, and services
  - Application component functionalities mapped to tiers and containers
    - Web container technologies
    - Business logic implementation technologies
    - Web service technologies
  - Packaging and deployment
  - Enterprise JavaBeans, managed beans, and CDI beans
    - Understanding lifecycle and memory scopes
  - Linking components together with annotations, injections, and JNDI

Agenda

- Tuning Web Applications
- Tuning Enterprise JavaBeans

What Is a JavaServer Page?

- JavaServer Pages:
  - Are templates for dynamic content
  - Extend HTML with custom Java code
  - Are compiled into servlets by WebLogic Server (WLS)
  - Allow for the division of labor into content production and programming

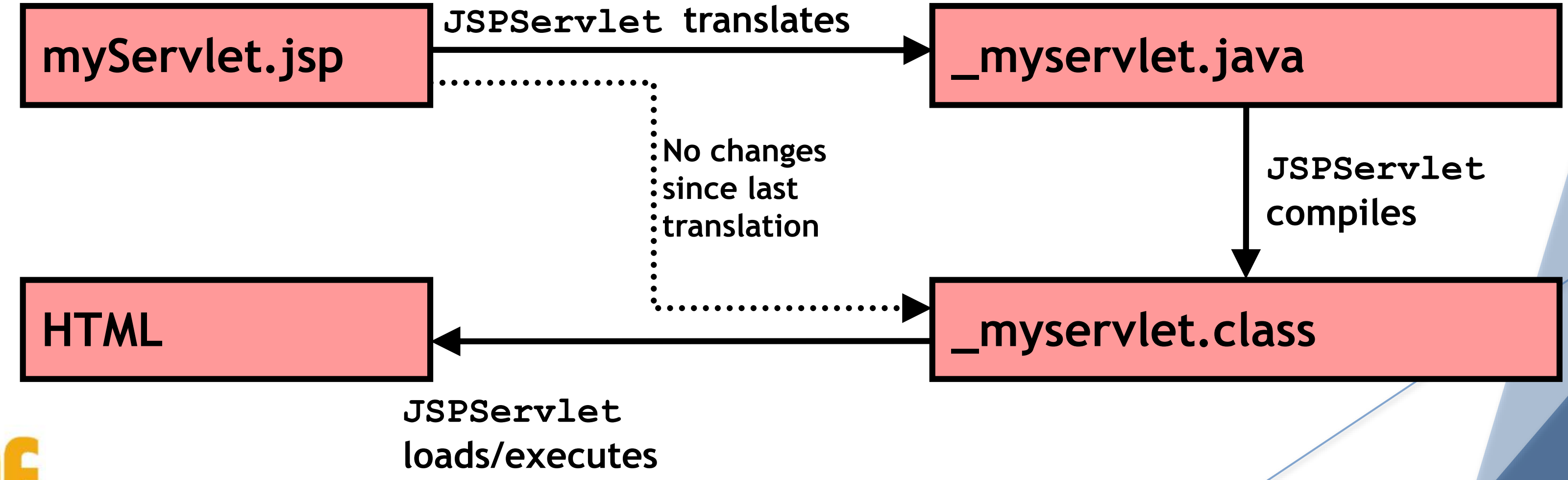


```
<H1>Today is:</H1> <%= new java.util.Date() %>
```

```
Today is: Mon May 09 17:10:25 EDT 2006
```

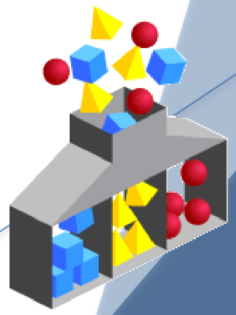
JSP Request Processing

- Requests for JSPs are handled by a special WLS servlet called `JSPServlet`, which handles:
  - Translation of JSP into a servlet
  - Compilation of a resulting servlet into a class
  - Execution of JSP



Precompiled JSP

- Precompiling JSPs save:
  - Initial time for the server translation and compilation
  - Memory required for noncompiled JSPs
  - Management related to noncompiled JSPs
- JSP precompilation can be done:
  - Manually, using WebLogic compilers such as `weblogic.appc`
  - Automatically, using the precompile option in `weblogic.xml`
  - If the precompile option is turned ON, recompilation
- of JSPs occurs each time the server restarts.



The appc Application Compiler

- The appc compiler compiles EJBs and JSPs, and generates the classes needed to deploy to WebLogic Server.
- The appc compiler provides the following benefits:
  - Flexibility of compiling all modules of an application
  - Validation checks throughout an entire application as well as individual modules
  - Easier identification and correction of errors
  - Reduction in time and effort in repeated compilations—when deploying to multiple servers
- Syntax for using appc:

```
$> java weblogic.appc -options <application archive file or
   directory>
```

Using the `precompile` Parameter

- You can configure WebLogic Server to precompile your JSPs when a web application is deployed or redeployed.
- Set the `precompile` parameter to true in the
- `<jsp-descriptor>` element of the `weblogic.xml` deployment descriptor.
- Snippet from `WEB-INF/weblogic.xml`:

In UNIX environments, the following parameter would help address mixed-case JSP names:

- `-Dweblogic.jsp.windows.caseSensitive=true`

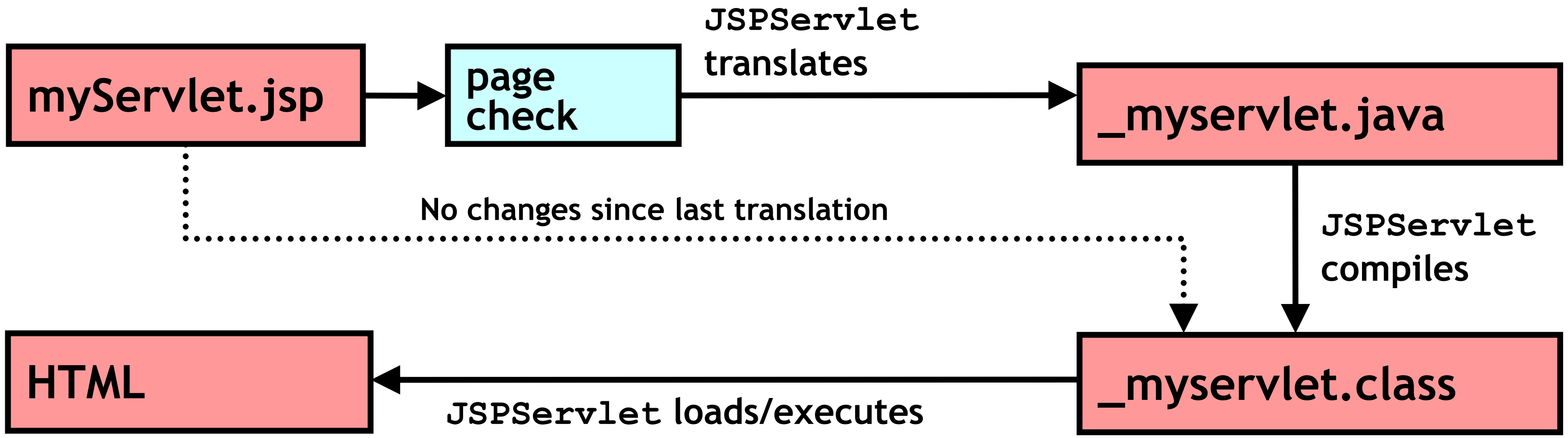
```
<jsp-descriptor>
       <precompile>true</precompile>
</jsp-descriptor>
```

HttpSession Replication Tuning

- WebLogic Server replicates only the session’s new/changed attributes.
- Keep session objects as small as possible.
- It is better to store lots of attributes rather than one.
- Put only Serializable objects in the session.
  - Failure to do so will prevent replication
- Monitor primary distributions across the cluster.
  - Uneven distributions likely means the Load Balancer/Proxy isn’t properly distributing the load.

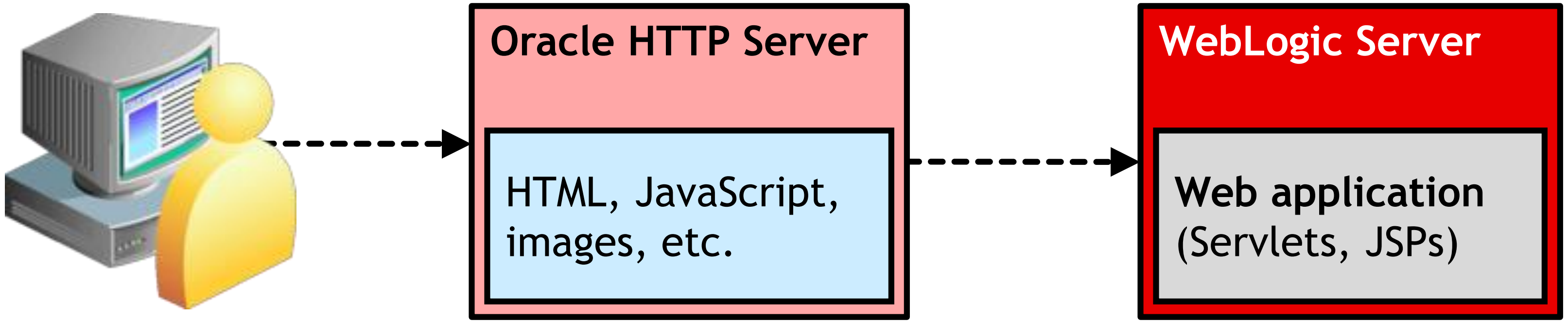
Page Check Interval

- Whenever a JSP or a servlet have been modified, it must be recompiled by the server.
- The frequency at which a server checks for modifications to JSPs or servlets can affect performance.



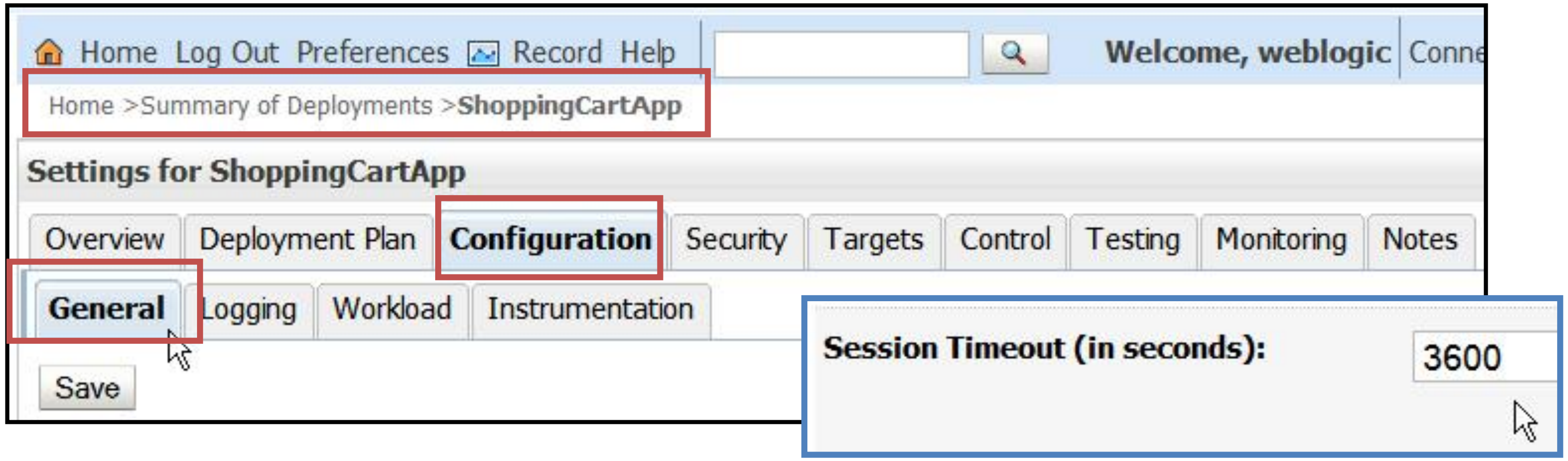
Using Web Servers for Static Content

- Front application servers and clusters with a dedicated web tier to serve static content such as HTML and image files
- Proxy requests for servlets and JSPs to application servers
- Perform additional optimizations at the web tier, such as caching.



Session Timeout

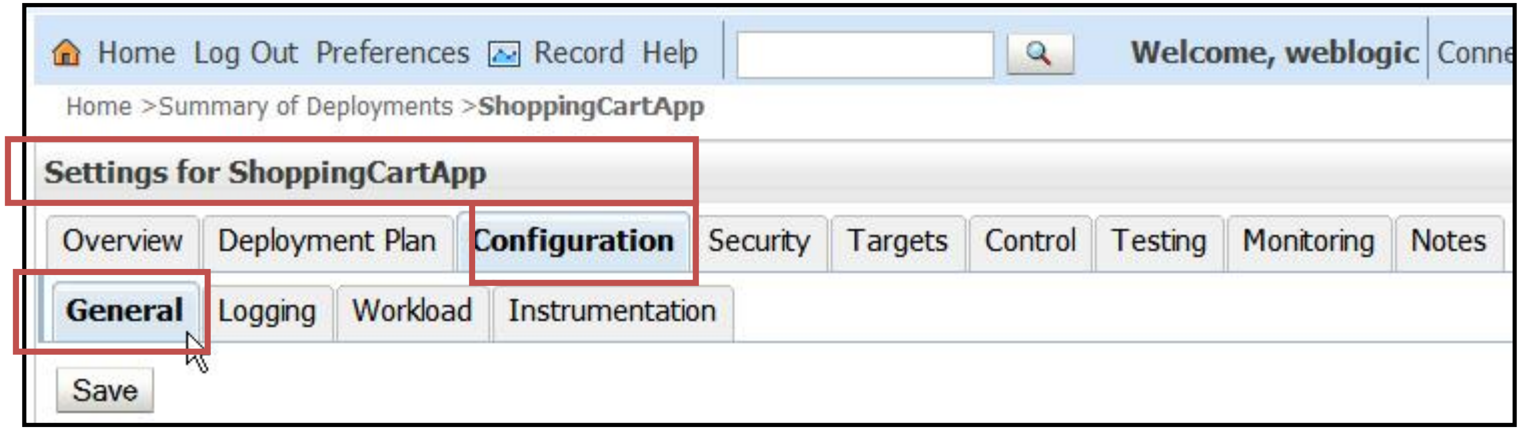
- Whenever an HTTP session object is created, it resides in memory until:
  - The session object is invalidated
  - The server removes it after a certain period of time (`session timeout`)
  - The server comes down
- You can set the session timeout parameter by using the Administration Console:

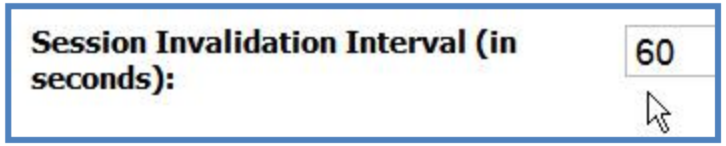


Session Timeout (in seconds): 3600

Session Invalidation

- Set the `invalidation-interval-secs` parameter to an appropriate value.
- Tuning this parameter can improve the performance of applications that have high traffic.





Session Invalidation Interval (in seconds): 60

Using Custom JSP Tags

- Oracle provides three specialized JSP tags that you can use in your JSPs:
  - `cache`: Enables caching the work that is done within the body of the tag
  - `process`: Enables you to control the flow of query parameter-based JSPs
  - `repeat`: Enables you to iterate over many different types of sets, including Enumerations, Iterators, Collections, and Arrays of Objects

Using the WebLogic `cache` Tag

- Caching data can boost performance significantly.
- The WebLogic `cache` tag enables caching the work that is done within the body of the `<wl:cache>` tag.
- The `cache` tag supports both output and input caching.
- The tag supports refreshing and flushing the cache at various scopes.
- Example of using the WebLogic `cache` tag:

```
<wl:cache name="holidaycache"
   key="parameter.holidaytable"scope="application">
  // Retrieve Holidays and output it to the page
</wl:cache>
```

Web Application Tuning JSP and Servlet Output Buffer Tuning

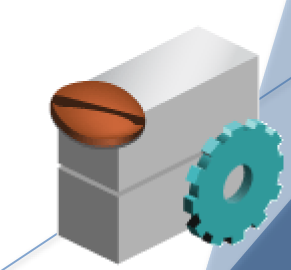
- Web Container buffers output while building response:
  - Data flushed to client once buffer is full
  - Cannot forward request once data sent to client
    - Set buffer size accordingly
- **`ServletResponse.setBufferSize(int)`**
- **`<@page contentType="text/html; buffer="64kb" %>`**
  - WLS default buffer size is 8 kb.
  - WLS default buffer size on Exalogic is 64 kb.
- Avoid calling **`flush()`** and **`close()`** on **`ServletOutputStream`**

Agenda

- Tuning Web Applications
- Tuning Enterprise JavaBeans

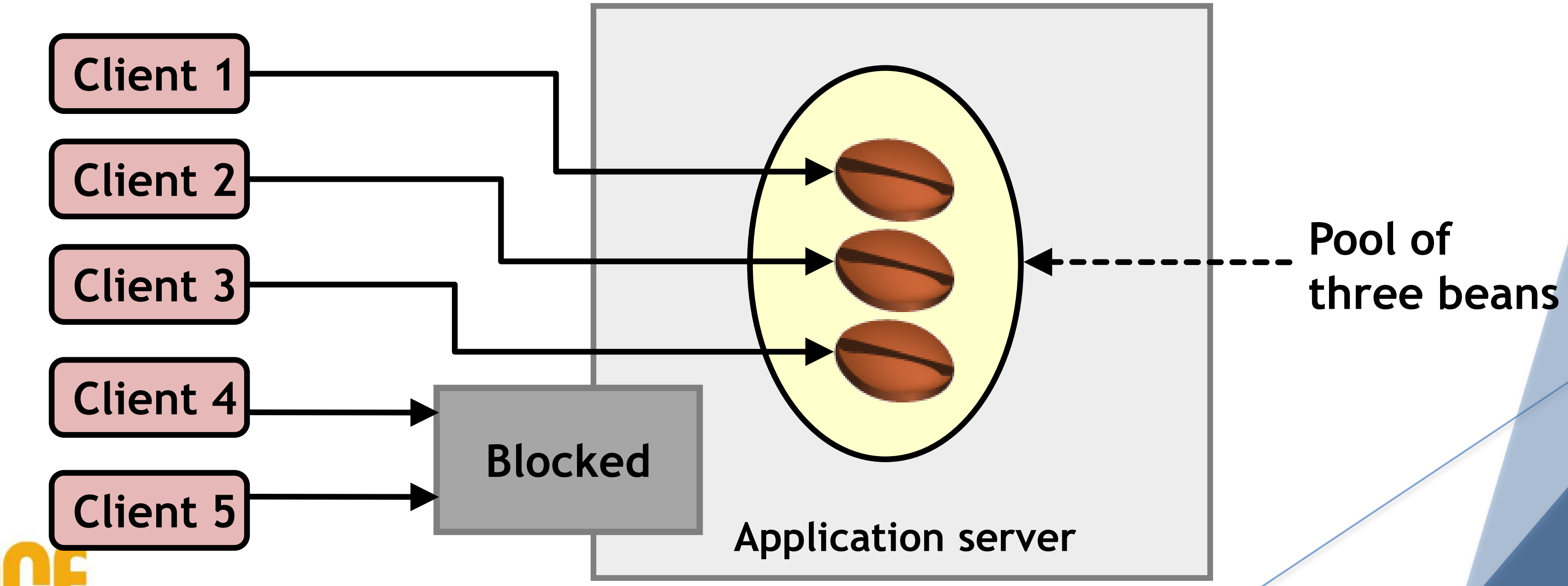
Enterprise JavaBeans

- Enterprise JavaBeans (EJBs) standardize the development and deployment of server components built in Java.
- The EJB specification discusses four types of objects:
    - Stateless session beans
    - Stateful session beans
    - Singleton session beans
    - Message-driven beans
- Note that entity beans specifications are considered legacy and are not present in the current Java EE specifications.



Stateless Session Beans

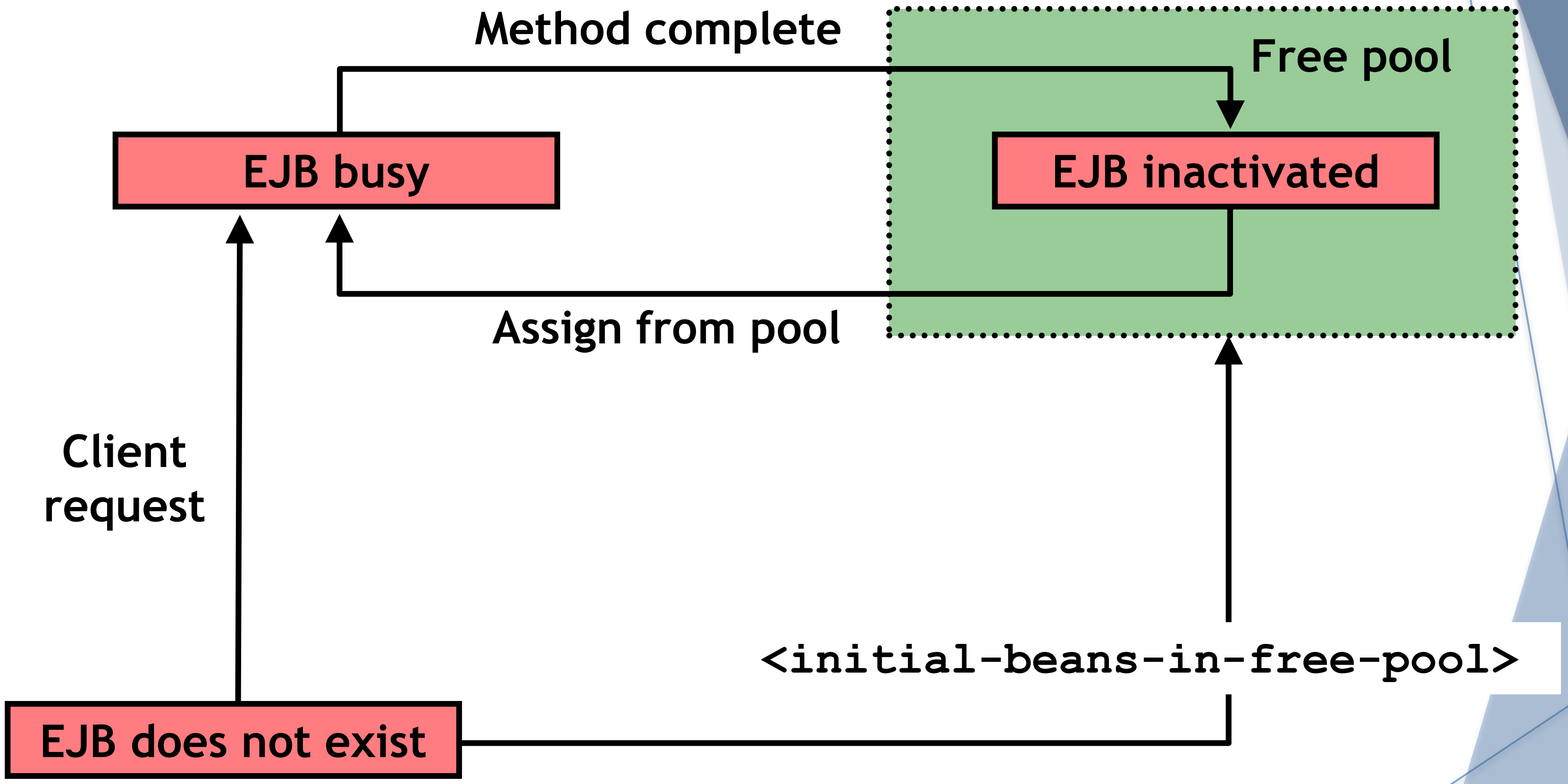
- Stateless session beans are maintained as a pool in memory.
- If the pool size is three, and if five clients attempt to use one EJB type, two clients are blocked.
- Stateless EJBs in a pool are identical. So a server can assign any *available* EJB to any client.



Stateless Session Beans Tuning

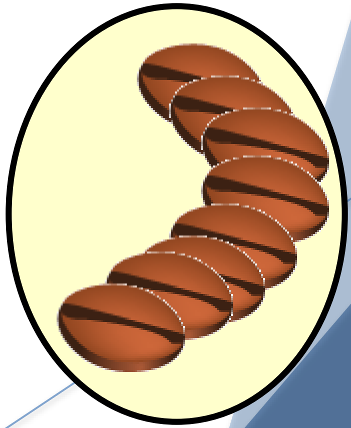
- The defaults are typically good enough.
  - <**`initial-beans-in-free-pool`**> = 0
  - <**`max-beans-in-free-pool`**> = 1000
  - There is a cost with creating a new instance in the pool. Thus, tuning the initial-beans-in-free-pool might be beneficial if the runtime cost is too much.
  - This may increase the time to start up, but reduces the runtime cost.

Pool Management



Determining the Pool Size

- You can improve performance by tuning the number of bean instances in this free pool.
- Factors to consider to determine how many EJBs should be in the pool include:
  - The number of threads set in WebLogic Server (WLS)
  - The number of concurrent clients
  - The number of dependent back-end resources (for example, database connections)



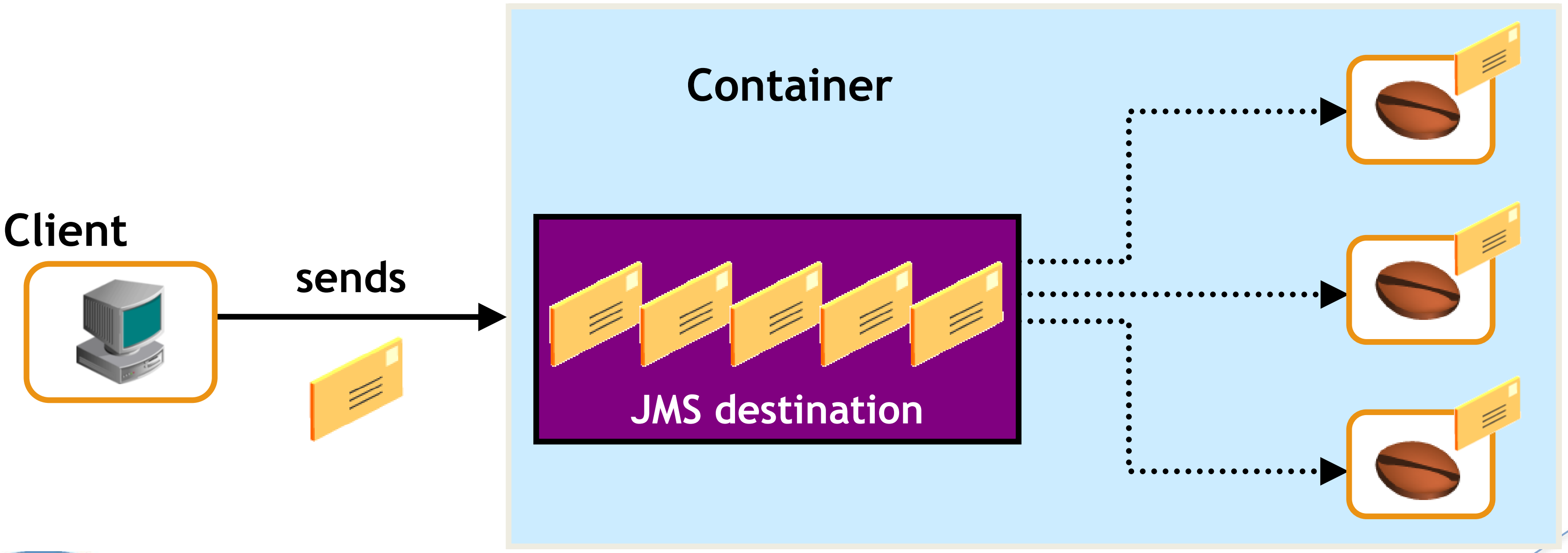
Configuring a Stateless Session EJB Pool

- You can manage the stateless session EJB pool by:
  - Capping the number of instances
  - Setting an initial pool size

```
<weblogic-enterprise-bean>
    <ejb-name>InsuranceQuoteBean</ejb-name>
    <stateless-session-descriptor>
      <pool>
        <max-beans-in-free-pool>15</max-beans-in-free-pool>
        <initial-beans-in-free-pool>5</initial-beans-in-free-
    pool>
      </pool>
    </stateless-session-descriptor>
    ...
</weblogic-enterprise-bean>
OR
@Stateless(name="InsuranceQuoteBean", maxBeansInFreePool="15",
    initialBeansInFreePool="5")
```

Message-Driven Beans

- Message-driven beans (MDBs) are:
  - Asynchronous stateless components
  - JMS message consumers
    - Clients do not interact directly with MDBs.



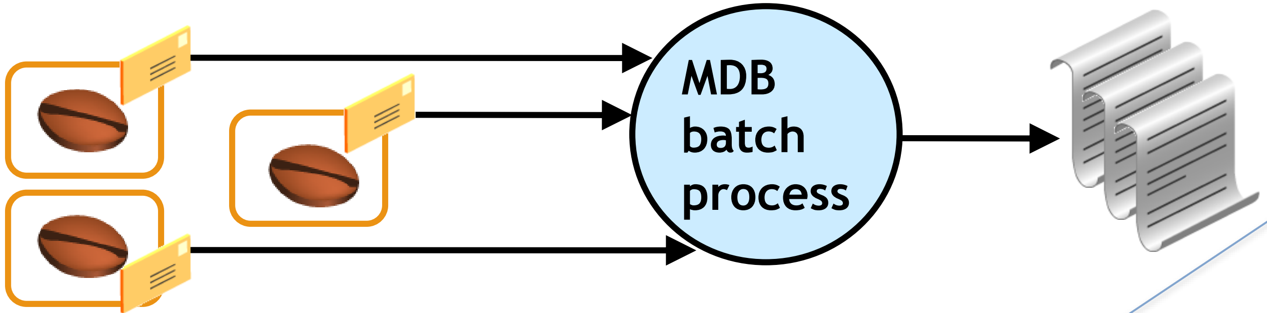
Configuring an MDB Pool

- You can manage the message-driven EJB pool by setting:
  - The initial pool size
  - The maximum size
  - `idle-timeout-seconds`

```
<weblogic-enterprise-bean>
  <ejb-name>InsuranceQuoteBean</ejb-name>
   <message-driven-descriptor>
    <pool>
     <max-beans-in-free-pool>15</max-beans-in-free-pool>
     <initial-beans-in-free-pool>5</initial-beans-in-free-pool>
    </pool>
    </message-driven-descriptor>
</weblogic-enterprise-bean>
OR
@MessageDriven(name="InsuranceQuoteBean",
    maxBeansInFreePool="15", initialBeansInFreePool="5")
```

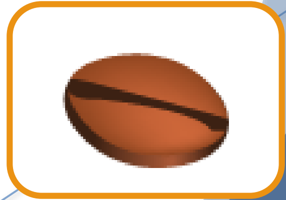
Configuring to Use Batching with an MDB

- Group transactions to reduce writing transaction logs.
- You can configure these parameters in `weblogic-ejb-jar.xml` to optimize MDB processing:
  - `max-messages-in-transaction`
  - `trans-timeout-seconds`
  - Each MDB listening on topic uses a dedicated
- daemon-polling thread.
- Each MDB listening on queue uses at least one thread from the dispatch policy.



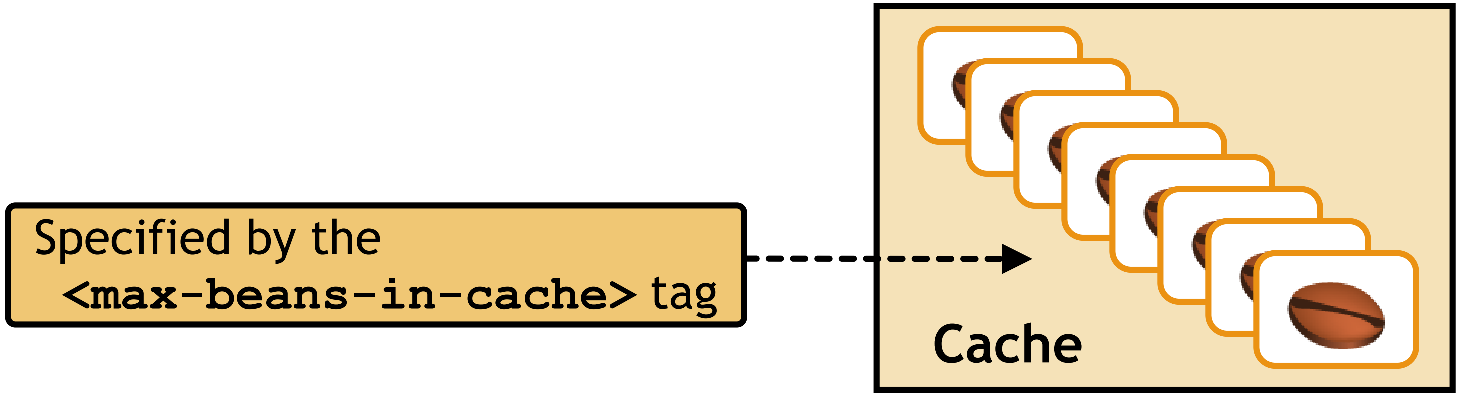
Stateful Session Beans

- Stateful session EJBs:
  - Provide conversational interaction
  - Store state on behalf of the client
  - Are associated with a single client
  - Are synchronous
  - Are maintained in memory
- Processing time includes passivation and activation.



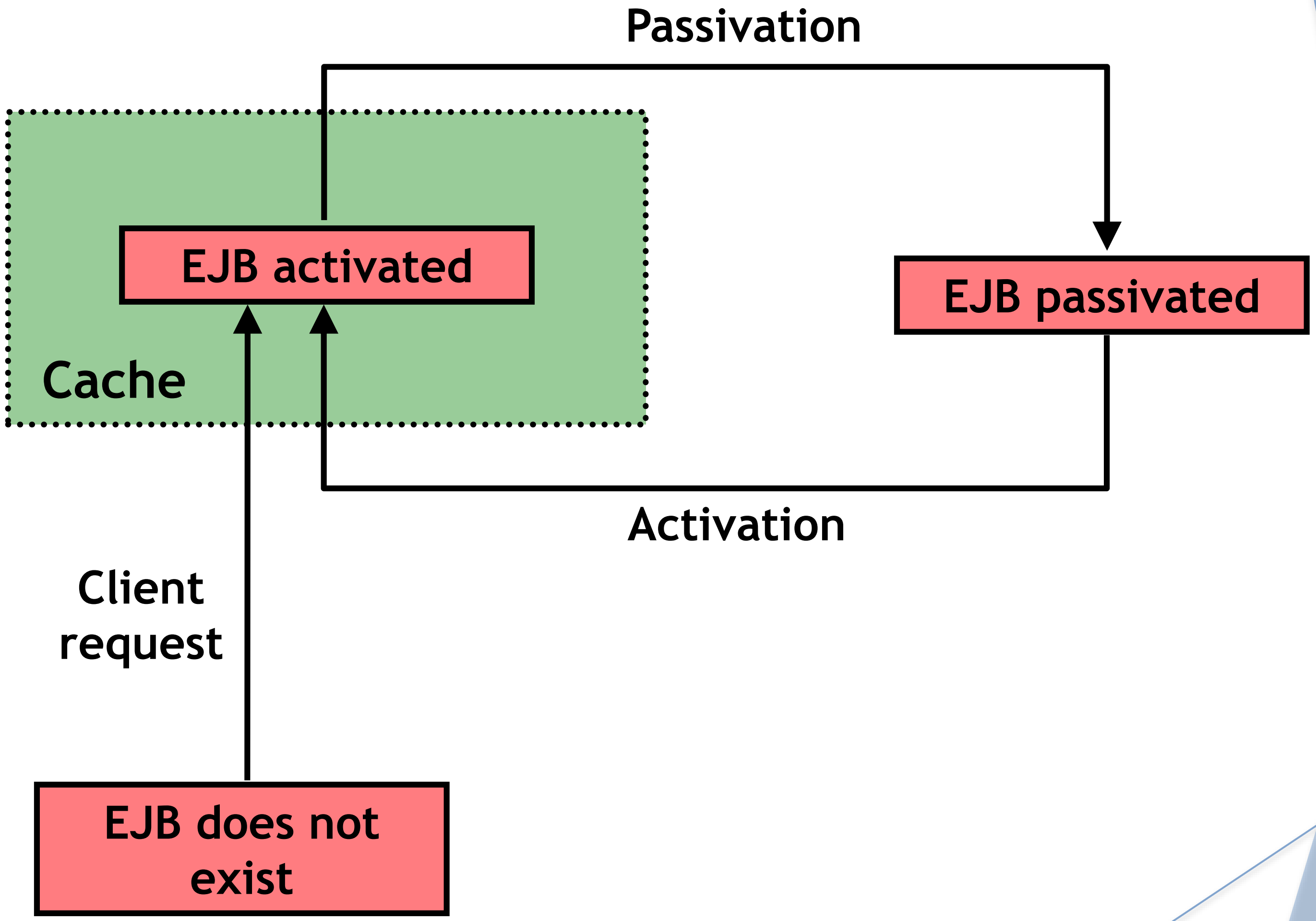
Cache Management

- You can specify the maximum number of stateful session EJB instances allowed in memory for each EJB class.



Specified by the `<max-beans-in-cache>` tag

Cache Management



What Happens When the Cache Fills Up?

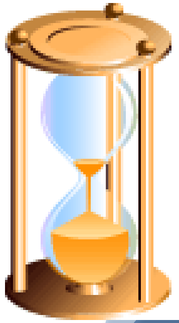
- WLS has two options to make additional memory available in the cache:
  - Destroy timed-out EJBs.
  - Passivate eligible EJBs to the hard drive.
- Passivation is the act of removing a bean instance from the cache and writing it to a temporary store.
- A passivated bean is activated when the requesting client comes back.
- Activation is the act of retrieving the bean instance from the temporary store to the cache.

Determining the Cache Size

- Tuning this cache can have a significant impact on performance.
- The criteria that affect the size of a cache include:
  - The total amount of Java heap space available in the VM
  - The number of different stateful sessions
  - The amount of memory that one stateful session bean consumes while active
  - The number of concurrent clients

Idle Timeout and Eligibility

- An EJB that exceeds its `<idle-timeout-seconds>` value can be destroyed by the WLS.
- An EJB is eligible for passivation if it:
  - Has not exceeded its `<idle-timeout-seconds>`
  - Is not currently executing a method
  - Is not participating in a transaction
- You can tune the `<idle-timeout-seconds>` value for optimal performance.



Cache Type

- The cache type specifies the order in which EJBs are removed from the cache.
- It includes:
  - Not recently used (NRU), known as lazy passivation
  - Least recently used (LRU), known as eager passivation
- The `idle-timeout-seconds` and `max-beans-in-cache` elements also affect passivation and removal behaviors, based on the value of the cache type.

Configuring a Stateful Session EJB Cache

- You can manage the stateful session EJB cache by:
  - Capping the number of instances
  - Setting an `idle-timeout` period
  - Setting the passivation strategy for the cached EJBs

```
<weblogic-enterprise-bean>
    <ejb-name>ShoppingCartBean</ejb-name>
    <stateful-session-descriptor>
      <stateful-session-cache>
       <max-beans-in-cache>1000</max-beans-in-cache>
       <idle-timeout-seconds>60</idle-timeout-seconds>
      </stateful-session-cache>
    </stateful-session-descriptor>
</weblogic-enterprise-bean>
OR
@Stateful(name="ShoppingCartBean",
    cacheType=Session.CacheType.N_R_U)
@CacheConfig (maxSize=100000, idleTimeoutSeconds=300,
    removalTimeoutSeconds=0)
```

Using Filtering ClassLoaders

- Use a Filtering ClassLoader to reduce the number of JAR files WebLogic will search while trying to load a class or resource:
  - Loading class from the current application still has to search all the way up to the system classloader before working its way down to the application's classloader.
  - WLS has hundreds of jar files, making this a somewhat expensive operation.
  - A Filtering ClassLoader short circuits this by searching the Application ClassLoader for configured classes.
    - It will not search parent class loaders for filtered classes!
    - The same concept applies for resources that you know exist only in the application.

General Application Performance Problems

- Most application performance issues are typically contention-related:
  - Overuse of synchronization
  - Insufficient resources (for example, database connections)
  - Overloaded back-end systems:
    - Load testing can identify most contention
    - Take thread dumps at regular intervals (5-10 secs)
    - Attach a profiler, such as HPROF or VisualVM

EJBs: Coding for Performance

- Cache JNDI InitialContext, Data Source, JMS Connection Factory and Destination objects:
  - Use the Service Locator pattern to cache lookups.
  - Use Dependency Injection.
  - Use Coherence to cache frequently used data:
    - Can be used explicitly in your application
    - Can be used as a JPA Level 2 cache

Developer Tip!

Q/A?