

Open Source Frameworks (OSF)

An Introduction to Java Enterprise Edition (Java EE)

Open Source Frameworks (OSF)
Master of Science in Engineering (MSE)
Olivier Liechti
olivier.liechti@heig-vd.ch



MASTER OF SCIENCE
IN ENGINEERING

The Notion of Systemic Quality

The notion of systemic quality

- How do we specify the **requirements** of a system?
 - We always build systems to deliver some sort of "**service**" (web, messaging, access control, customer management, etc.).
 - We first have to specify "**what**" the system should do. In other words, we have to specify **functional requirements**.
 - We also have to specify "**how**" the system should behave, i.e. what qualities it should exhibit. These are the **non-functional requirements**.
- Non-functional requirements characterize **systemic qualities, or "-ilities"**:
 - There are lots of different systemic qualities. Depending on the system, some are more important than others.
 - Choices made when defining the system architecture have a large impact on the systemic qualities.
 - Your life as an architect will be to deal with **trade-offs** in addressing conflicting systemic qualities.

Example

System

Vehicle

Functional requirements

Move people around

Non-functional requirements

Performance

Capacity

Reliability

Cost

Aesthetics

Ease of use



Different systemic qualities often create **opposing forces**.

Defining the “right” architecture for a system means finding the right **balance** between these forces.



Some systemic qualities...

- **Response time**
 - Measures the time required to present a result to the user
 - Important for the end-user
- **Throughput**
 - Measures the number of requests that can be processed in a given time frame
 - Important for the service provier
- **Scalability**
 - Measures how easy/costly it is to adapt the system in order to handle additional load
 - Ideally: linear scalability. "2 x more users => 2 x more servers"
- **Availability**
 - Measures the percentage of time during which the system can be used
 - 99% availability = average unavailability of 3.65 jours per year, i.e 1 hour and 41 minutes per week.

Some systemic qualities...

- **Reliability**

- Mesure la capacité du système à "remplir sa fonction" sur une période.
- Un système peut avoir une fiabilité plus élevée que celle d'un de ces sous-système (notamment grâce à la redondance).
- Mesure exprimée en "temps moyen entre 2 pannes" (Mean Time Between Failures).

- **Evolvutivity**

- Mesure la facilité avec laquelle le système pourra être adapté pour satisfaire à de nouvelles exigences (fonctionnelles et non-fonctionnelles!).

- **Security**

- Mesure la capacité du système à empêcher une utilisation abusive.

- **Manageability**

- Mesure la facilité avec laquelle l'état du système pourra être surveillé, les pannes détectées, des opérations de maintenance réalisées, etc.

Java Enterprise Edition (Java EE)

What is Java Enterprise Edition?

- It is a **development platform**: it provides high-level APIs to develop software components.
- It is an **execution platform**: it provides an environment to deploy and bring these components “to live”.
- It is an **enterprise platform**: it provides support for distributed transactions, security, integration, etc.
- **Separation of concerns**: "The developer takes care of the business logic. Java EE takes care of the systemic qualities".



http://flickr.com/photos/decade_null/427124229/sizes/m/#cc_license

Java EE and standards

- Java EE is a specification
 - Defined through the JCP, it is a specification that software editors can decide to implement. Java EE 5 is defined in JSR 244.
- Java EE is an “umbrella” specification
 - Java EE builds upon other specifications (servlets, EJBs, JDBC, etc.) and specifies which specifications (and which versions) need to be implemented by a Java EE certified application server.
 - Java EE also defines a programming model and defines several roles (developer, assembler, deployer, etc.).



Specification Lead

★ Bill Shannon Sun Microsystems, Inc.

Expert Group

Barreto, Charlton
Capgemini
Dudney, Bill
Hewlett-Packard
Kohen, Erika S.
Oracle
Pratap, Rama Murthy Amar
Reinshagen, Dirk
Shah, Suneet
Tiwari, Ashish
Umapathy, Sivasundaram

BEA Systems
Chandrasekaran, Muralidharan
E.piphany, Inc.
IBM
Leme, Felipe
OW2
Raible, Matt
SAP AG
Sun Microsystems, Inc.
Tmax Soft, Inc.

Borland Software Corporation
Crawford, Scott
Genender, Jeff
Ironflare AB
Novell, Inc.
Pramati Technologies
Red Hat Middleware LLC
SeeBeyond Technology Corp.
Sybase
Trifork

J2EE or Java EE?

- Java Enterprise Edition is **not** a recent specification.
- The SDK 1.2 was published in 1999.
- The specification is managed through the JCP since version 1.3
- We used to talk about "Java 2 Enterprise Edition 1.3", or J2EE 1.3
- We then moved from J2EE 1.3 to J2EE 1.4 to... **Java EE 5**
- Today, we should speak of Java EE, but J2EE is still sometimes used...
- Today, most application servers implement Java EE 6
- The early draft 2 for Java EE 7 has been submitted in November 2012; final spec expected for Q2 2013



Specification Lead

★ Bill Shannon Sun Microsystems, Inc.

Expert Group

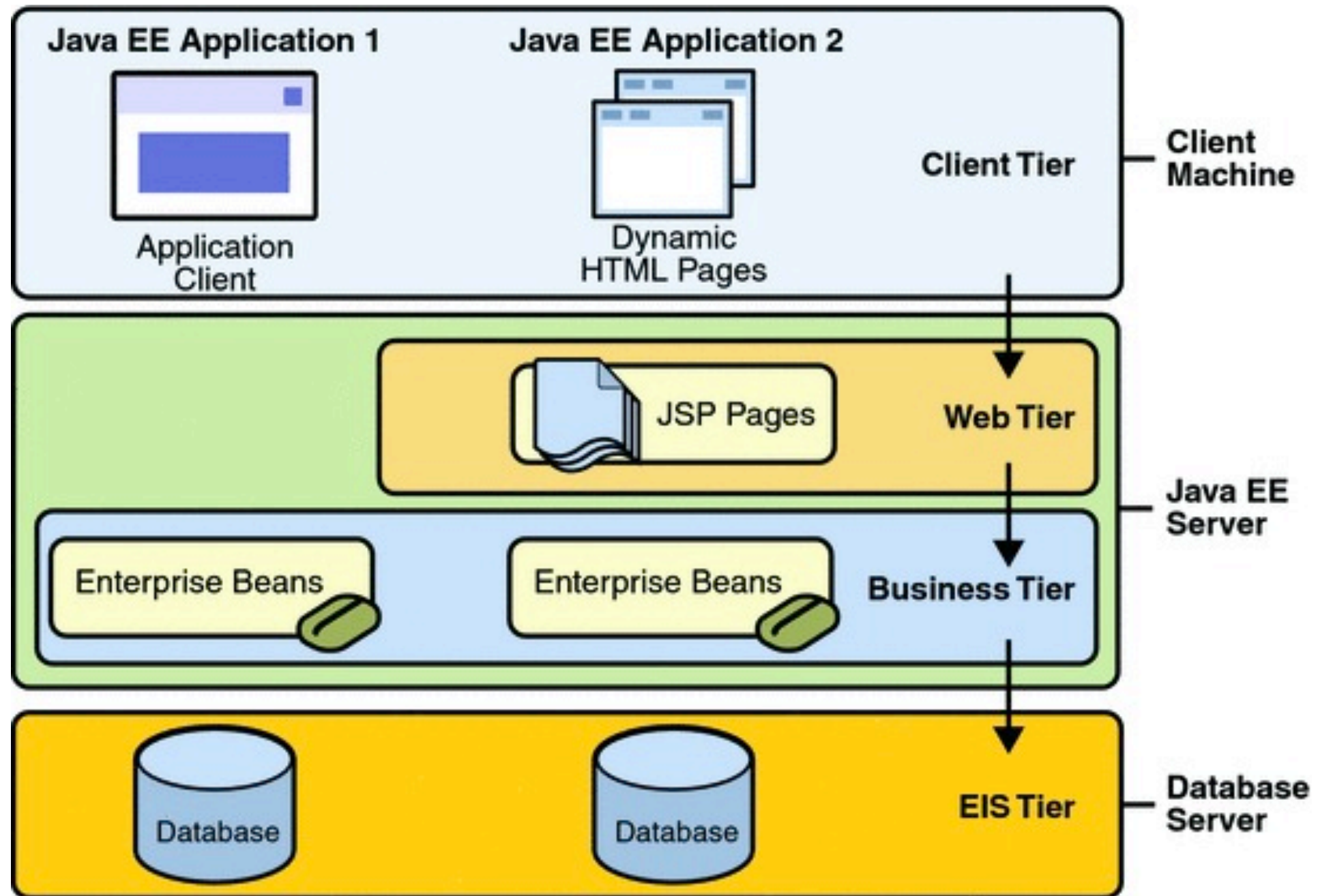
Barreto, Charlton
Capgemini
Dudney, Bill
Hewlett-Packard
Kohen, Erika S.
Oracle
Pratap, Rama Murthy Amar
Reinshagen, Dirk
Shah, Suneet
Tiwari, Ashish
Umapathy, Sivasundaram

BEA Systems
Chandrasekaran, Muralidharan
E.piphany, Inc.
IBM
Leme, Felipe
OW2
Raible, Matt
SAP AG
Sun Microsystems, Inc.
Tmax Soft, Inc.

Borland Software Corporation
Crawford, Scott
Genender, Jeff
Ironflare AB
Novell, Inc.
Pramati Technologies
Red Hat Middleware LLC
SeeBeyond Technology Corp.
Sybase
Trifork

Architecture

- The software that implements the Java EE specification is called an “**application server**”
 - There are **open source** and **proprietary** application servers.
 - Glassfish, JBoss, WebSphere, BEA WebLogic are examples of application servers.
 - Editors compete on aspects that are not defined the specification (clustering, administration, etc.).
- Key notion in the Java EE architecture: the **containers**
 - a container is an **environment** in which we deploy components;
 - a container **provides services** (transactions, security, etc.) through APIs;
 - there are different containers in Java EE: the “**web**” container, the “**ejb**” container and even a “**client**” container that can be used for rich clients.



<http://java.sun.com/javaee/5/docs/tutorial/doc/bnaay.html>

Enterprise JavaBeans Technology

Java Servlet Technology

JavaServer Pages Technology

JavaServer Pages Standard Tag
Library

JavaServer Faces

Java Message Service API

Java Transaction API

JavaMail API

JavaBeans Activation Framework

Java API for XML Processing

Java API for XML Web Services
(JAX-WS)Java Architecture for XML Binding
(JAXB)SOAP with Attachments API for
Java

Java API for XML Registries

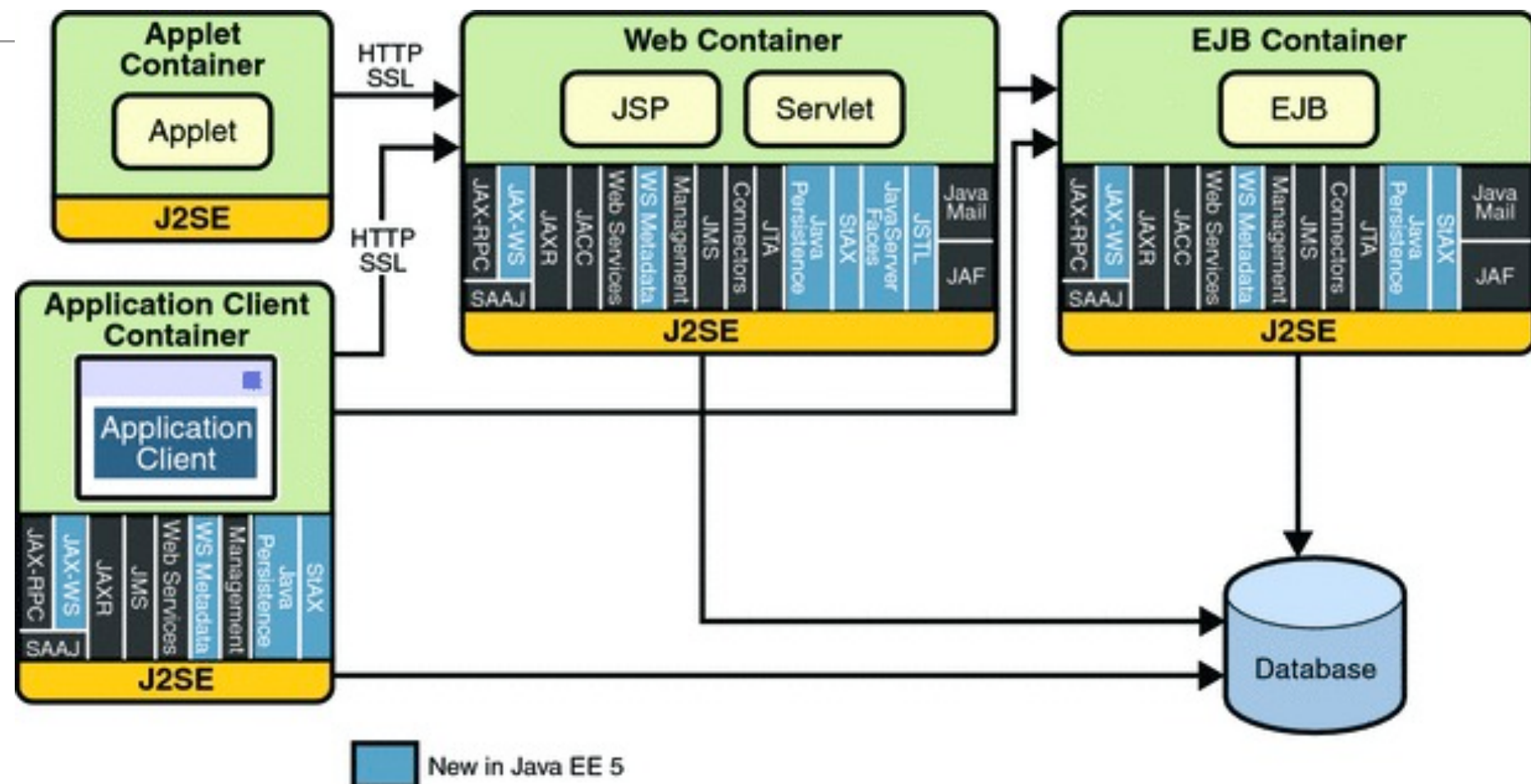
J2EE Connector Architecture

Java Database Connectivity API

Java Persistence API

Java Naming and Directory
InterfaceJava Authentication and
Authorization Service

Simplified Systems Integration



<http://java.sun.com/javaee/5/docs/tutorial/doc/bnacj.html>

Components, packaging & deployment

- The first version of J2EE put a lot of emphasis on the separation of roles:
 - **Component developer** builds self-contained, reusable **components** and focuses on business logic (“Here is a customer management brick”, “Here is a pricing brick”).
 - **Application assemblers** create **applications** by gluing together different components (“I connect the customer management and the pricing bricks in order to have an insurance application”).
 - **Deployers** have to take applications and to put them into production. Every application needs resources (DBs, user repositories, etc.). The deployer has to connect the application to the resources (declaratively!!).
- We also want to be able to manage the life-cycle of applications and to have control on the release of new versions. It really helps to create **packages**!

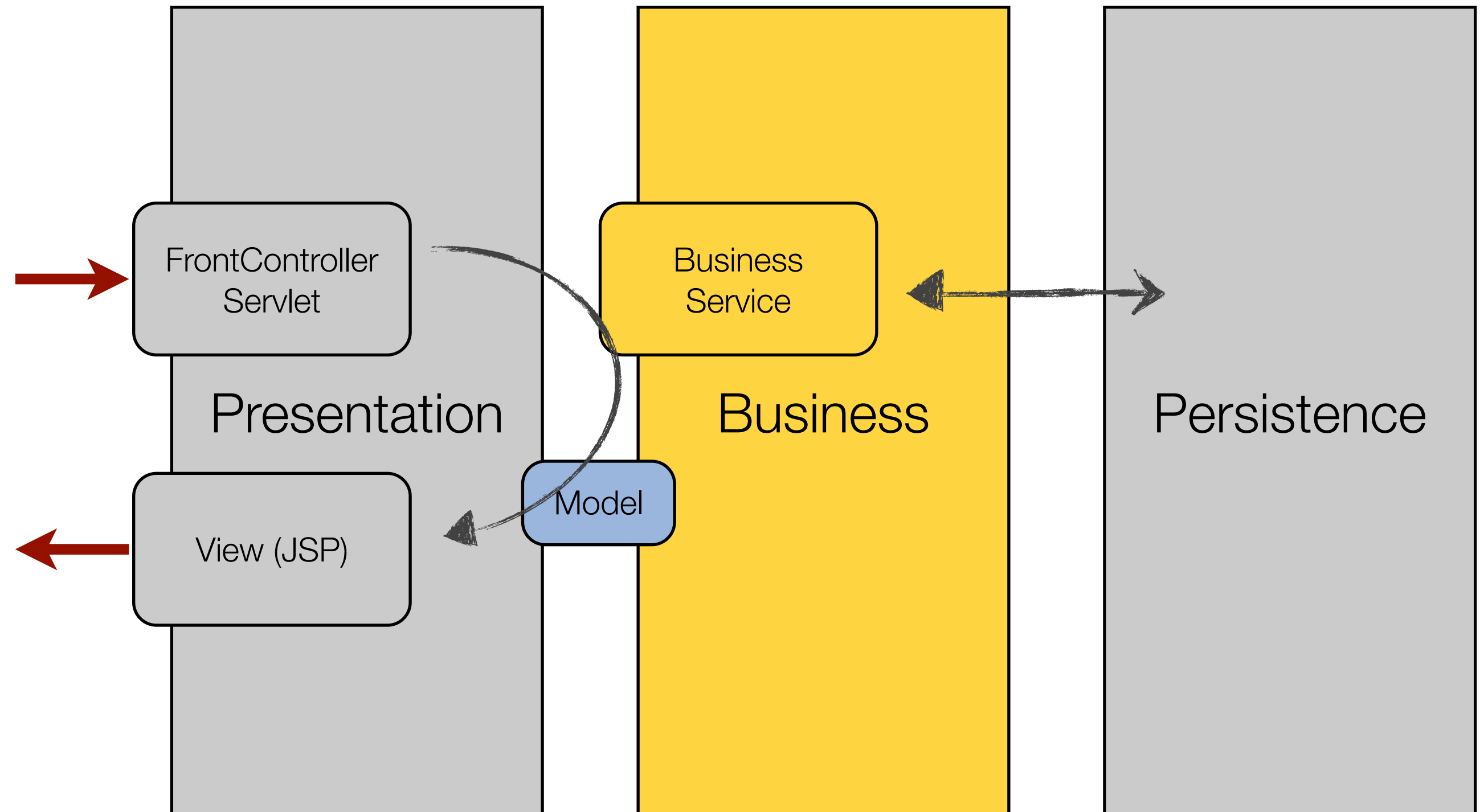
Components, packaging & deployment

- Different types of components, different types of packages/archives:
 - Java libraries are useful in enterprise applications; we can bundle classes into **traditional .jar files (Java ARchives)** and integrate them in our applications.
 - Some applications do not require all the features provided by the Java EE platform; they are “lightweight” can live entirely in the web container. We package these applications in **.war files (Web ARchives)**.
 - Business services can be packaged and deployed independently from a presentation front-end. We can package **EJBs in .jar files**.
 - Full-blown Java EE applications bundle together web-tier components, service components, libraries. A Java EE application is packaged in a **.ear file (Enterprise ARchive)**. The .ear file contains a .war files and one or more .jar files.

Content of archives

- In .war, .jar and .ear files, we find:
 - **Compiled classes and libraries**
 - **Resources** (images, property files, configuration files, etc.)
 - **Deployment descriptors**
- Deployment descriptors make it possible to alter the system-level behavior of the application **declaratively**, at deployment time:
 - Change security mappings, change transaction isolation level, change database connection parameters, change URL mappings, etc.
- In J2EE, deployment descriptors were mandatory and quite a pain to write (IDE added support over time, and XDoclet introduced a way to generate them automatically).
- In Java EE, deployment descriptors are still available but are largely replaced by @annotations. In Java EE 5, annotations are generally used for EJBs. The web.xml deployment descriptor is still used.

The Business Tier



Agenda

- Containers & components
- Enterprise Java Beans (EJB)
 - Stateless and stateful session beans (SLSB and SF SB)
 - Message-driven beans (MDB) will be presented later (JMS)
- Resource pooling
- Life cycle
- Getting a reference to the service: lookup vs. dependency injection
- Interceptors

Reference: the Java EE 7 Tutorial

<http://docs.oracle.com/javaee/7/tutorial/doc/>



The screenshot shows the Oracle Java EE 7 Tutorial page. At the top, the Oracle logo is displayed in red. Below it, a breadcrumb trail reads "Java > Java EE > Java EE Documentation". The main heading is "The Java EE 7 Tutorial" with a small icon of a book. Below the heading, the authors are listed: Eric Jendrock, Ricardo Cervera-Navarro, Ian Evans, Kim Haase, William Markito, and Chinmayee Srivathsa. To the right of the authors, there is a "Next Page" link with a right-pointing arrow. Further right, the Java logo is shown, followed by the text "Release: Java EE 7.0", "Last Updated: August, 2013", and "Download as PDF". Below this, a table of contents is listed, including "Preface", "Part I Introduction", "1 Overview", "2 Using the Tutorial Examples", "Part II Platform Basics", "3 Resource Creation", "4 Injection", "5 Packaging", and "Part III The Web Tier". Another "Next Page" link with a right-pointing arrow is located between the authors and the table of contents.

ORACLE


Java > Java EE > Java EE Documentation

 The Java EE 7 Tutorial

The Java EE 7 Tutorial

Eric Jendrock
Ricardo Cervera-Navarro
Ian Evans
Kim Haase
William Markito
Chinmayee Srivathsa

Next Page →

 **Release: Java EE 7.0**
Last Updated: August, 2013
Download as PDF

Preface
Part I Introduction
1 Overview
2 Using the Tutorial Examples
Part II Platform Basics
3 Resource Creation
4 Injection
5 Packaging
Part III The Web Tier

Next Page →

Containers and components

- There are **different ways** to design, build and deploy the “business service” components.
- In the **simplest scenario**, think of a plain vanilla web application, where you create a “controllers”, a “services” and a “model” packages. Here, you implement services as POJOs and **do everything yourself**:
 - Instantiation of services, concurrency control, transactions, security, etc.
- When the application/system has some complexity (because of the functionality, because of the volume of transactions, because of the integration, etc.), it becomes interesting to use “**managed**” **components**.
- The idea is to let “something” in the **infrastructure** manage the life cycle of your service components. That “something” can take different shapes.
- In Java EE, that “something” is the Enterprise Java Bean **container**.

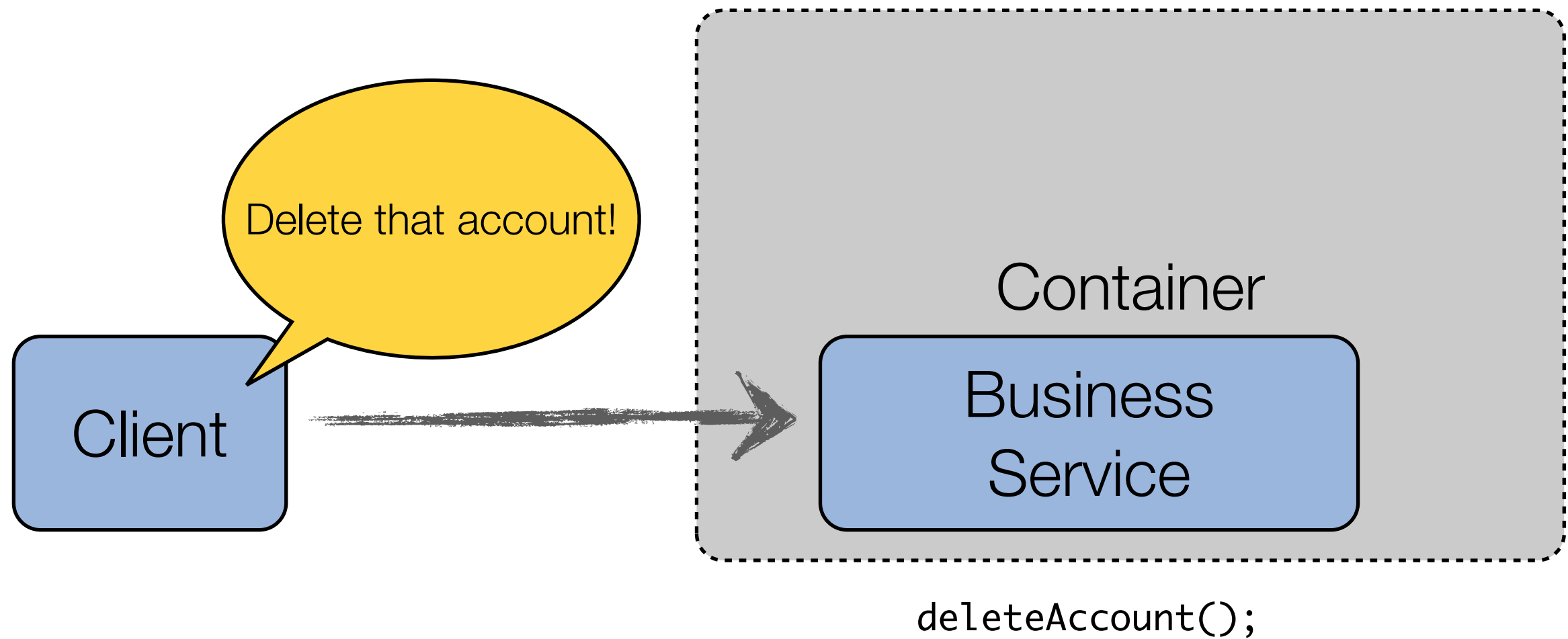
Containers and components

- The **EJB container** is one sub-system of the Java EE container.
- It provides a **runtime environment** for the business services, implemented as Enterprise Java Beans.
- The **developer** knows that services will be “available” to its components, once they are deployed in the container. He can thus use these services through standard **APIs**.
- The **production manager** configures the EJB container and tunes it in different ways (this is implementation specific).
- With **some application servers**, the EJB container can be **distributed** over several physical nodes (for scalability and availability purposes). This distribution is **transparent to the clients** - they still only see “one runtime environment”.

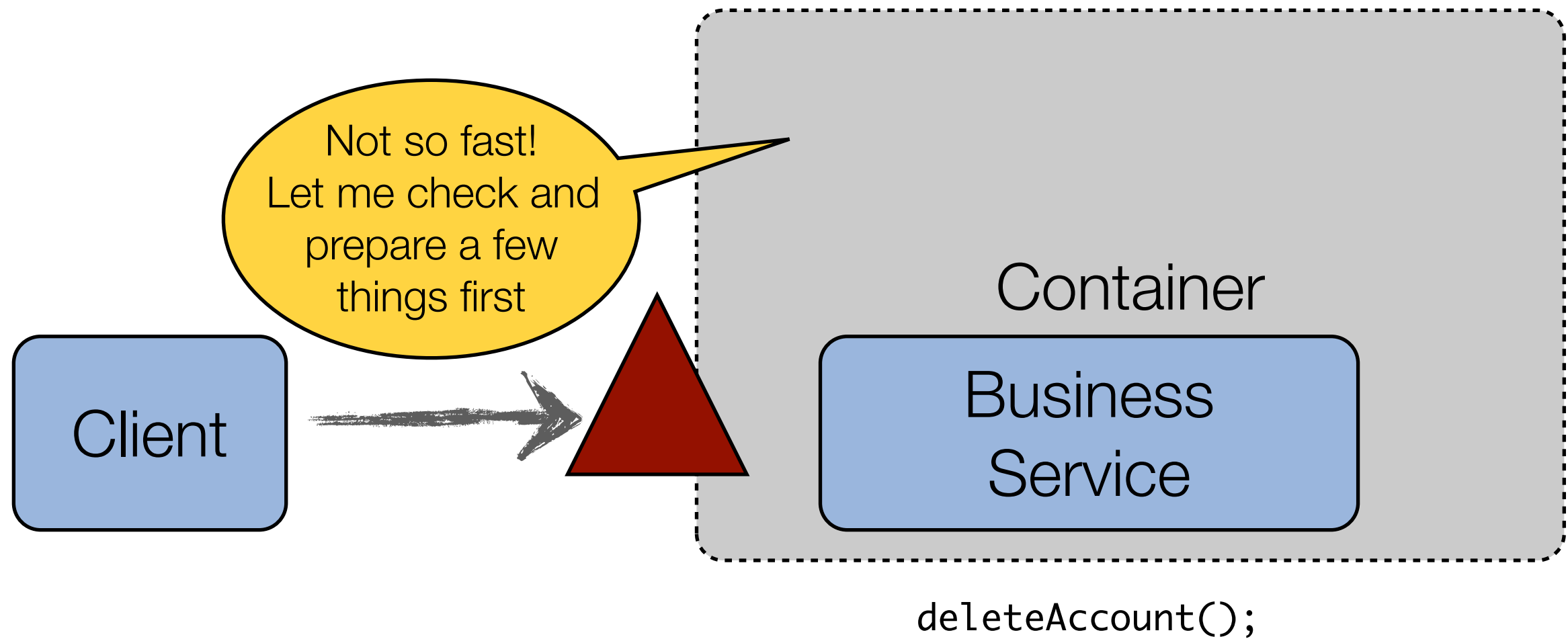
From the specs: what are EJBs (v3)

- “An enterprise bean typically contains **business logic** that operates on the **enterprise’s data**.
- An enterprise bean’s instances are **managed at runtime by a container**.
- An enterprise bean can be **customized at deployment time** by editing its environment entries.
- Various service information, such as **transaction** and **security** attributes, may be specified **together** with the business logic of the enterprise bean class in the form of **metadata annotations**, or **separately**, in an **XML deployment descriptor**. This service information may be extracted and managed by tools during application assembly and deployment.
- Client **access is mediated by the container** in which the enterprise bean is deployed.”

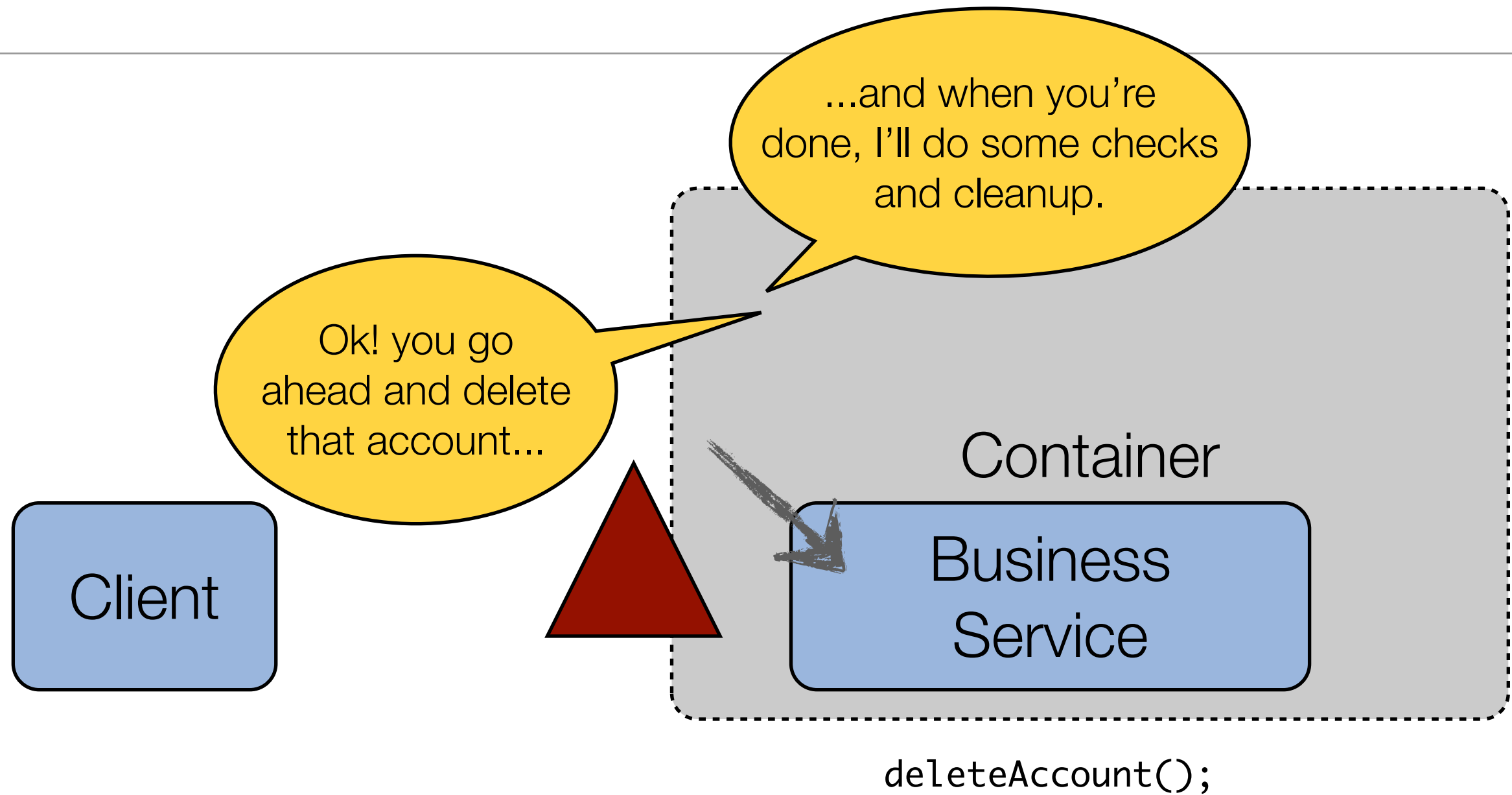
Mediated access



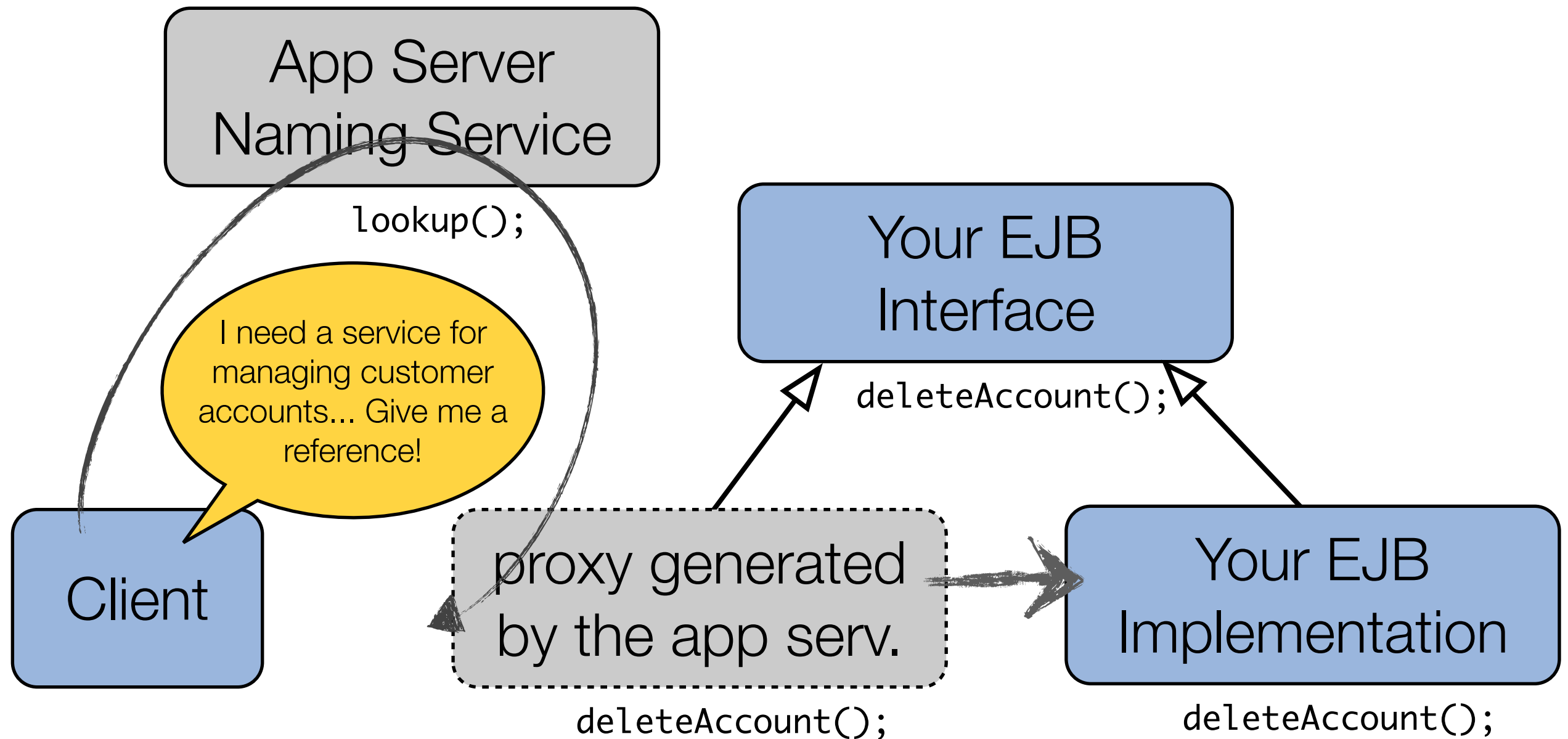
Mediated access



Mediated access

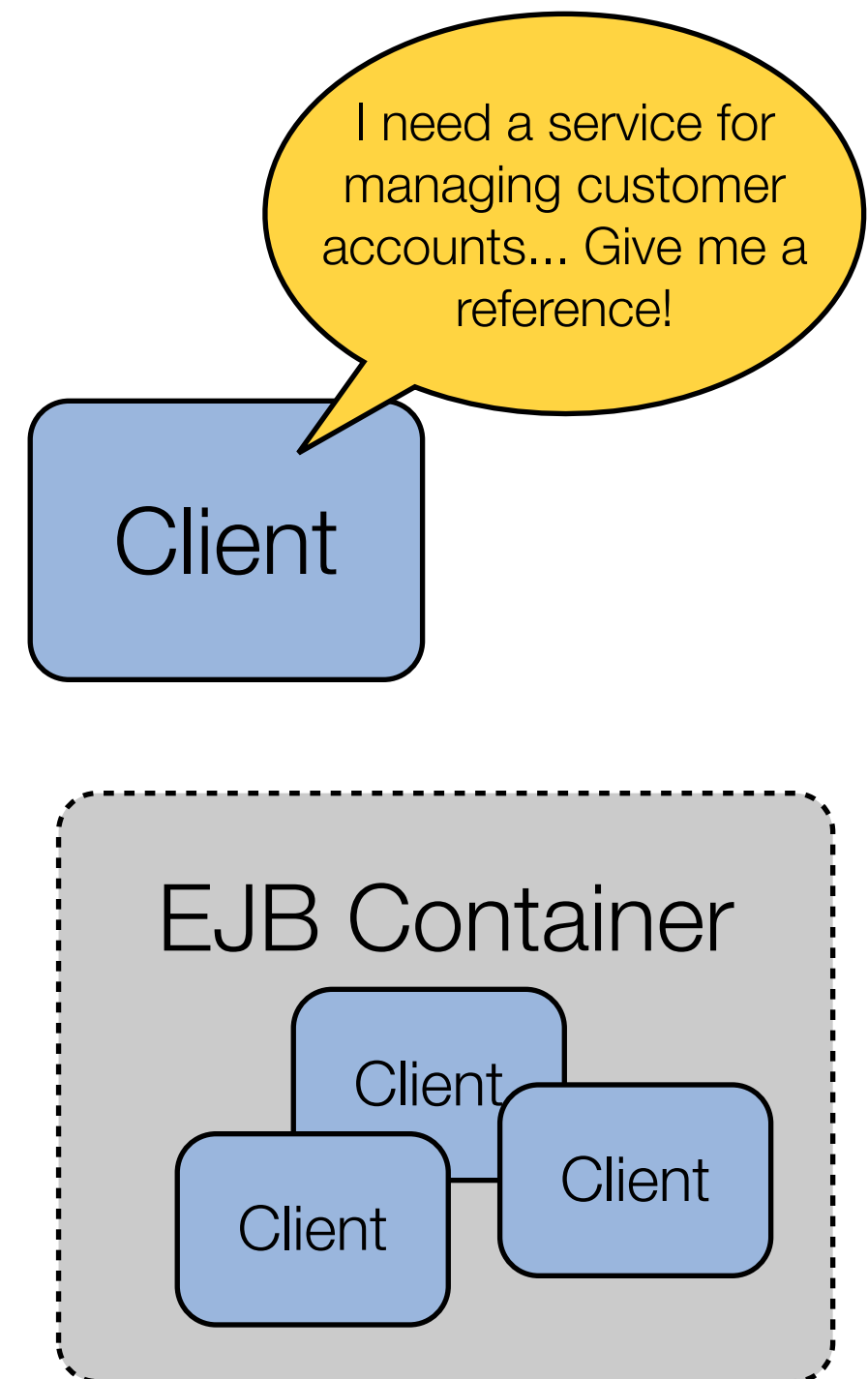


Mediated access



Resource pooling

- As a developer, you define a **service interface** and provide an **implementation**.
- **You do not instantiate the service** - the application server takes care of that.
- The container does not instantiate only one instance of a class implementing your interface; **it creates several instances that are placed into a pool**.
- When a client comes along, a **free instance** is picked from the pool and **dedicated** to the client (no concurrency issue).
- In production, you can tune the size of pools (tradeoff between processing **bandwidth** and resource **consumption**).



From the specs: component types

- “The enterprise bean architecture is flexible enough to implement the following:
 - An object that represents a **stateless service**.
 - An object that represents a stateless service and that implements a **web service endpoint**.
 - An object that represents a stateless service and whose invocation is **asynchronous, driven by the arrival of messages**.
 - An object that represents a **conversational session** with a particular client. Such session objects automatically maintain their **conversational state** across multiple client-invoked methods.
 - An entity object that represents a **fine-grained persistent object**.”

From the specs: component types

- “The enterprise bean architecture is flexible enough to implement the following:
 - An object that represents a **stateless service**.
 - An object that represents a stateless service and that implements a **web service endpoint**.
 - An object that represents a stateless service and whose invocation is **asynchronous, driven by the arrival of messages**.
 - An object that represents a **conversational session** with a particular client. Such session objects automatically maintain their **conversational state** across multiple client-invoked methods.
 - An entity object that represents a **fine-grained persistent object**.”

Stateless Session
Bean

Stateful Session
Bean

Message Driven
Bean

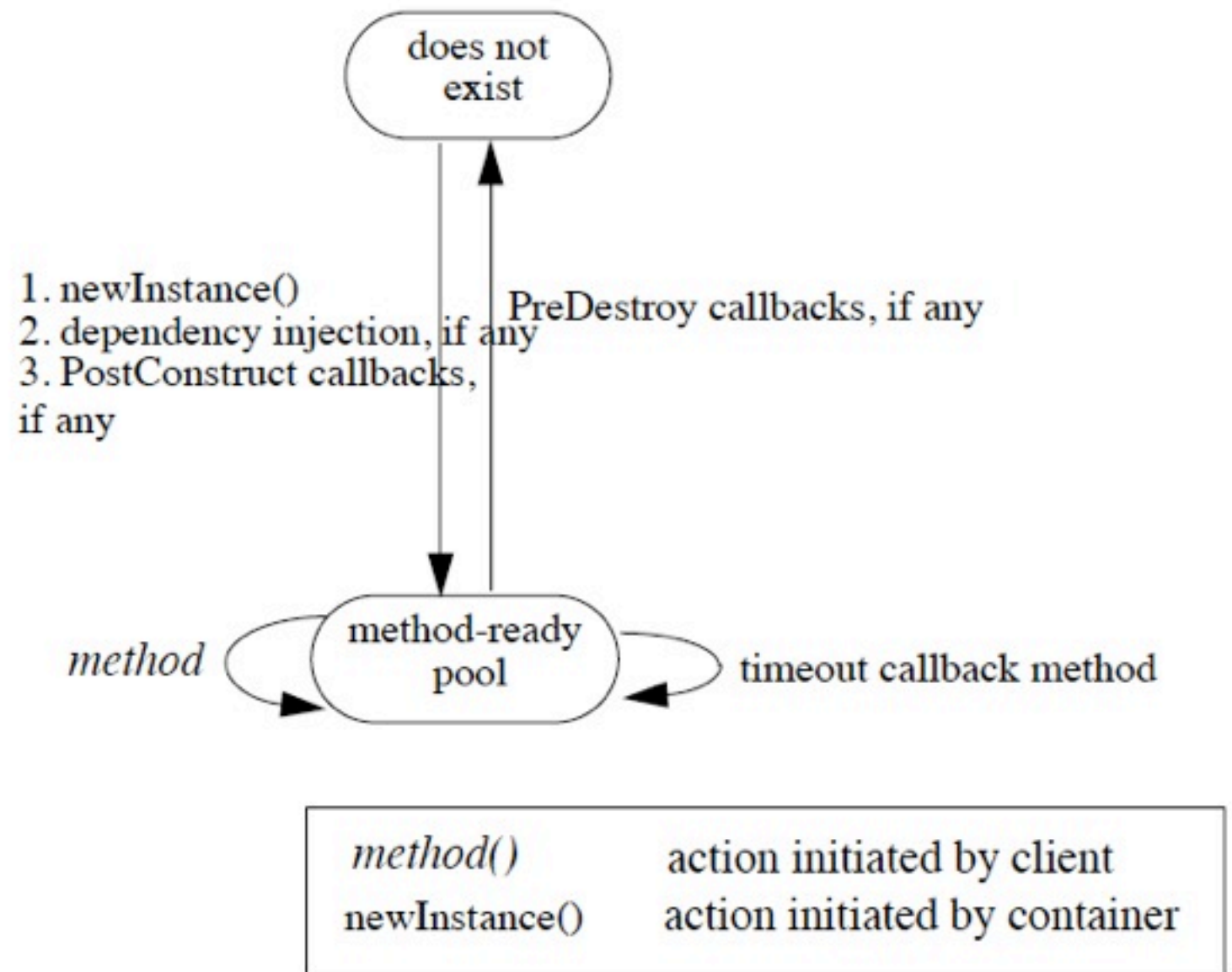
Entity Bean (not the
same as JPA entity!!)

Session Beans (EJB)

- A session bean may be either:
 - **stateless**—the session bean instances contain no conversational state between methods; **any instance** can be used for any client.
 - **stateful**—the session bean instances contain **conversational state** which must be **retained** across methods and transactions.
- Notes:
 - The term “**stateless**” signifies that an instance has no state **for a specific client**. However, the instance variables of the instance **can** contain the state across client-invoked method calls. Examples of such state include an open database connection and an object reference to an enterprise bean object.
 - The lifetime of stateful session bean is controlled by the client (the client call `myStatefulService.remove()`)

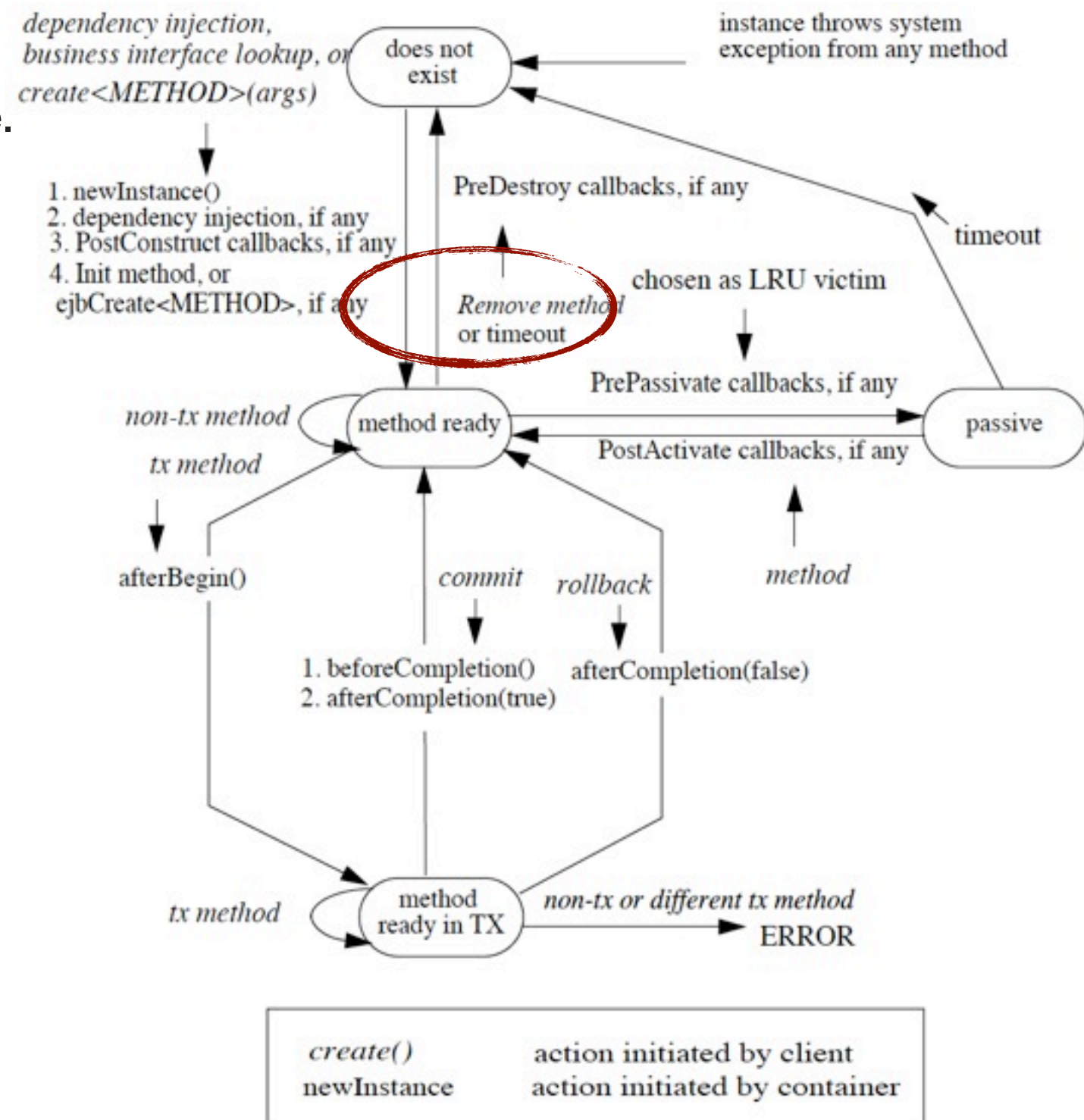
Stateless Session Bean Life Cycle

- The container manages the life cycle of the beans.
- The developer may implement callback methods, invoked when the state changes (e.g. for init/clean-up tasks).
- With EJB3, you can do that with method annotations.
- Remember that you don't know how many instances there are, you don't know which one you're talking to.



Stateful Session Bean Lifecycle

- Things are more complicated here, because we manage client-specific state.
- What happens when we run out of memory? We need to move the state of “some beans” to secondary memory.
- This is called “passivation”.
- When a client comes back and wants to invoke a method on a passivated bean, we need to bring it back to life.
- This is called “activation”.
- You typically “remove” a bean at the end of the use case (conversation).
- Check out the JBoss Seam Framework for an architecture that makes use of SF SB.



The client view: reaching the service

- Before using a service, the client needs to obtain a **reference** to a component that implements the service.
- Two approaches: **lookup** and **dependency injection**
- From the original J2EE model, we can do a lookup - the application server provides a **naming service** through JNDI.
- **Dependency injection** was introduced in the EJB3 / Java EE 5 specification. Here, the app server injects a reference into the client.

```
InitialContext ic = new InitialContext();  
CustomerService service =  
(CustomerService)ic.lookup("customerService")
```

```
@EJB  
CustomerService service
```

Client

I need a service for managing customer accounts... Give me a reference!

Interceptors

- We have seen the notion of “**mediation**” by the container.
- The idea is that the container “**intercepts**” every method call on the EJBs and gets a change to “do its stuff”.
- As a developer, you do not directly control what the container is doing at interception time. However, you **declare your intentions** (the application-level semantics).
 - With some annotations, you declare how transactions should be managed (no transaction, stay in the same transaction, start a new one, etc.).
 - With other annotations, you declare how security should be managed (does the user have to be in a role to invoke the method, etc.).
- We EJB 3, the developer now can also provide its own interceptors.
 - For instance, think about creating an audit log for all method access.
 - This is very similar to the notion of “aspect oriented programming” that we will come back to when looking at the Spring Framework.

From the specs: Interceptors

- “Interceptors are used to **interpose** on **business method invocations** and **lifecycle events** that occur on an enterprise bean instance.
- **Any number** of interceptor classes may be defined for a bean class.
- It is possible to **carry state across multiple interceptor method invocations** for a single business method invocation or lifecycle callback event for a bean in the context data of the `InvocationContext` object.”

```
public interface InvocationContext {  
    public Object getTarget();  
    public Method getMethod();  
    public Object[] getParameters();  
    public void setParameters(Object[] params);  
    public java.util.Map<String, Object> getContextData();  
    public Object proceed() throws Exception;  
}
```


Interceptors

- “Interceptor methods may be defined for business methods of sessions beans and for the message listener methods of message-driven beans.
- Business method interceptor methods are denoted by the **Around-Invoke annotation** or **around-invoke deployment descriptor** element.”

```
import javax.interceptor.AroundInvoke;
public class MethodProfiler {
    public MethodProfiler() {}
```

```
@AroundInvoke
```

```
private Object profile(InvocationContext invCtx) throws Exception {
    long t1 = System.nanoTime();
    Object result = invCtx.proceed();
    long t2 = System.nanoTime();
    System.out.println(invCtx.getMethod().getName() +
        "(" + invCtx.getParameters()[0] + ") took: " +
        ((t2 - t1)/ 1000.0) + " nano seconds.");
    return result;
```

```
}
```

```
}
```

```
@Stateful
```

```
@Interceptors({MethodProfiler.class})
```

```
public class StringServiceBean
    implements StringServiceRemote {
```

```
...
```

```
}
```

Interceptors

- Interceptors can be “attached” at different levels
 - Default interceptors (in deployment descriptor)
 - Class-level
 - Method-level
- The specification gives all the details for the syntax and the rules

@Stateless

```
public class MyBean ... {  
    public void notIntercepted() {}  
}
```

```
@Interceptors(org.acme.MyInterceptor.class)  
public void someMethod() {  
}
```

```
@Interceptors(org.acme.MyInterceptor.class)  
public void anotherMethod() {  
}
```

Environment Setup, part 1

- Objectives **(to be done for next week!)**
 - Have an **IDE** that you can use to develop Java EE applications (**Netbeans 7.4 RC1**)
 - Have a **Java EE 7 application server** that you can use to deploy Java EE applications (**Glassfish 4.0**)
 - Have the required supporting tools installed (**apache maven**)
 - Be able to build, deploy and run the **Java EE 7 Tutorial examples**.
- Information
 - **Basic setup:** <http://docs.oracle.com/javaee/7/tutorial/doc/usingexamples.htm#GFIUD>
 - **Validation**, the hello2 example: <http://docs.oracle.com/javaee/7/tutorial/doc/webapp004.htm#BNAEO>