Lecture 11

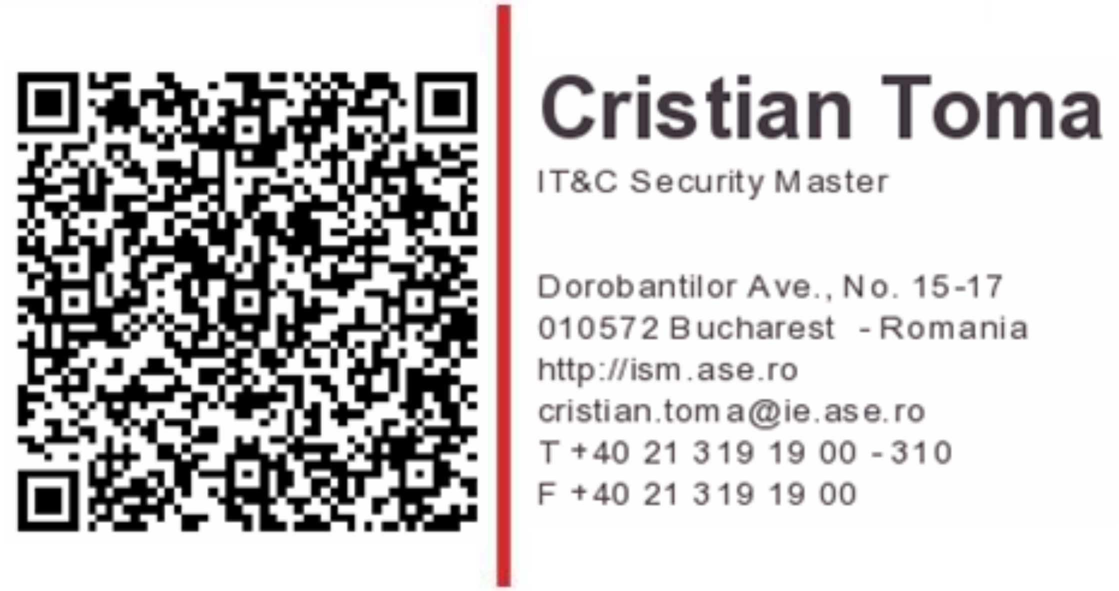
presentation

DAD – Distributed Applications Development

Cristian Toma

D.I.C.E/D.E.I.C – Department of Economic Informatics & Cybernetics
www.dice.ase.ro

Cristian Toma – Business Card



Cristian Toma

IT&C Security Master

Dorobantilor Ave., No. 15-17
010572 Bucharest - Romania
http://ism.ase.ro
cristian.toma@ie.ase.ro
T +40 21 319 19 00 - 310
F +40 21 319 19 00

Agenda for Lecture 11

1. EJB 3.x – Enterprise Java Beans
2. Akka Actors
3. Exchange Ideas

1

EJB – Enterprise Java Beans, Session – Stateless vs. Stateful, Entity – JPA, MDB – Message Driven Bean

EJB Overview

1. EJB 2.x Overview

EJB Products – Java Web & App Servers:

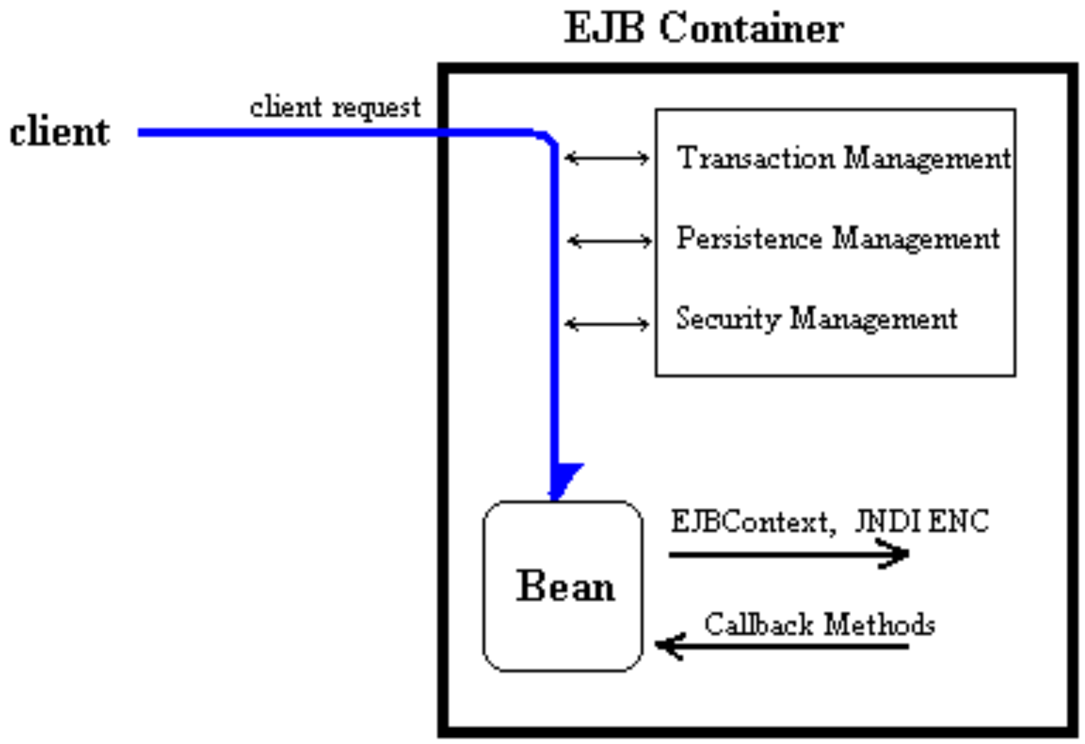
The Enterprise JavaBeans (EJB) 1.1, 2.0 and 3.0 specifications define an architecture for the development and deployment of transactional, distributed object applications-based, server-side software components.

Java Web & App Servers with EJB Containers:

- JBoss – RedHat Linux Division – Portlet + *BPM/BPEL – Rules Engine*
- BEA Web Logic – purchased by Oracle
- GlassFish – Sun Microsystems – purchased by Oracle
- Oracle 9iAS – Oracle
- Apache GERONIMO – the only openSource compliant with JEE 5.0
- IBM Web Sphere - Portlet

1. Middleware & JMS Technology Overview

EJB Container:



EJB Containers manage enterprise beans at runtime

The **Enterprise JavaBeans** specification defines an architecture for a transactional, distributed object system based on components.

The specification mandates a programming model; that is, conventions or protocols and a set of classes and interfaces which make up the **EJB API**.

Lecture 11

Part I – EJB Container

Enterprise beans are software components that run in a special environment called an EJB container.

The EJB container hosts and manages an enterprise bean in the same manner that the Java Web Server hosts a servlet or an HTML browser hosts a Java applet. An enterprise bean cannot function outside of an EJB container.

The EJB container manages every aspect of an enterprise bean at runtimes including remote access to the bean, security, persistence, transactions, concurrency, and access to and pooling of resources.

Lecture 11

An enterprise bean depends on the container for everything it needs. If an enterprise bean needs to access a JDBC connection or another enterprise bean, it does so through the container; if an enterprise bean needs to access the identity of its caller, obtain a reference to itself, or access properties it does so through the container.

The *enterprise bean interacts with its container* through 1 of 3 mechanisms (from JNDI are triggered Callback methods or EJBContext):

- JNDI – Java Naming and Directory Interface – ENC – Environment Naming Context like: rmiregistry 4 RMI, COS 4 CORBA, UDDI 4 WS
  - Callback Methods
  - EJBContext

Lecture 11

Part I – EJB Interaction with EJB Container

JNDI ENC

JNDI – Java Naming and Directory Interface is a standard extension to the Java platform for accessing naming systems like LDAP, NetWare, file systems, etc. Every bean automatically has access to a special naming system called the *ENC – Environment Naming Context*. The ENC is managed by the container and accessed by beans using JNDI. The JNDI ENC allows a bean to access resources like JDBC connections, other enterprise beans, and properties specific to that bean.

Lecture 11

Part I – EJB Interaction with EJB Container

Callback Methods

Every bean implements a subtype of the *EnterpriseBean* interface which defines several methods, called callback methods. Each callback method alerts the bean TO a different event in its lifecycle and the container will invoke these methods to notify the bean when it's about to activate the bean, persist its state to the database, end a transaction, remove the bean from memory, etc. The callback methods give the bean a chance to do some housework immediately before or after some event.

Lecture 11

Part I – EJB Interaction with EJB Container

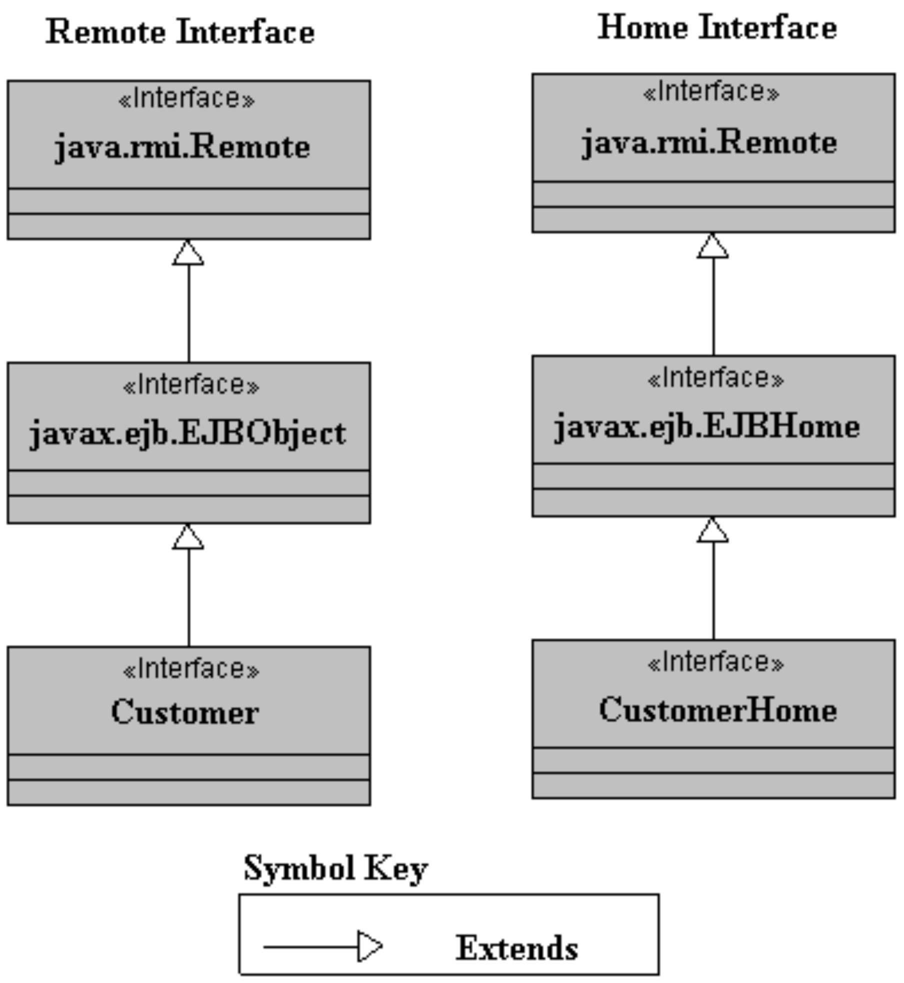
EJBContext

Every bean obtains an ***EJBContext*** object, which is a reference directly to the container. The ***EJBContext*** interface provides **methods** for interacting with the container so that that bean can request information about its environment like the identity of its client, the status of a transaction, or to obtain remote references to itself.

The EJBContext could be EntityContext or SessionContext;

Lecture 11

Part I – Software Conceptual View of EJB



Class Diagram of Remote and Home Interfaces

The *home interface* represents the life-cycle methods of the component (create, destroy, find)
while the *remote interface* represents the business method of the bean.

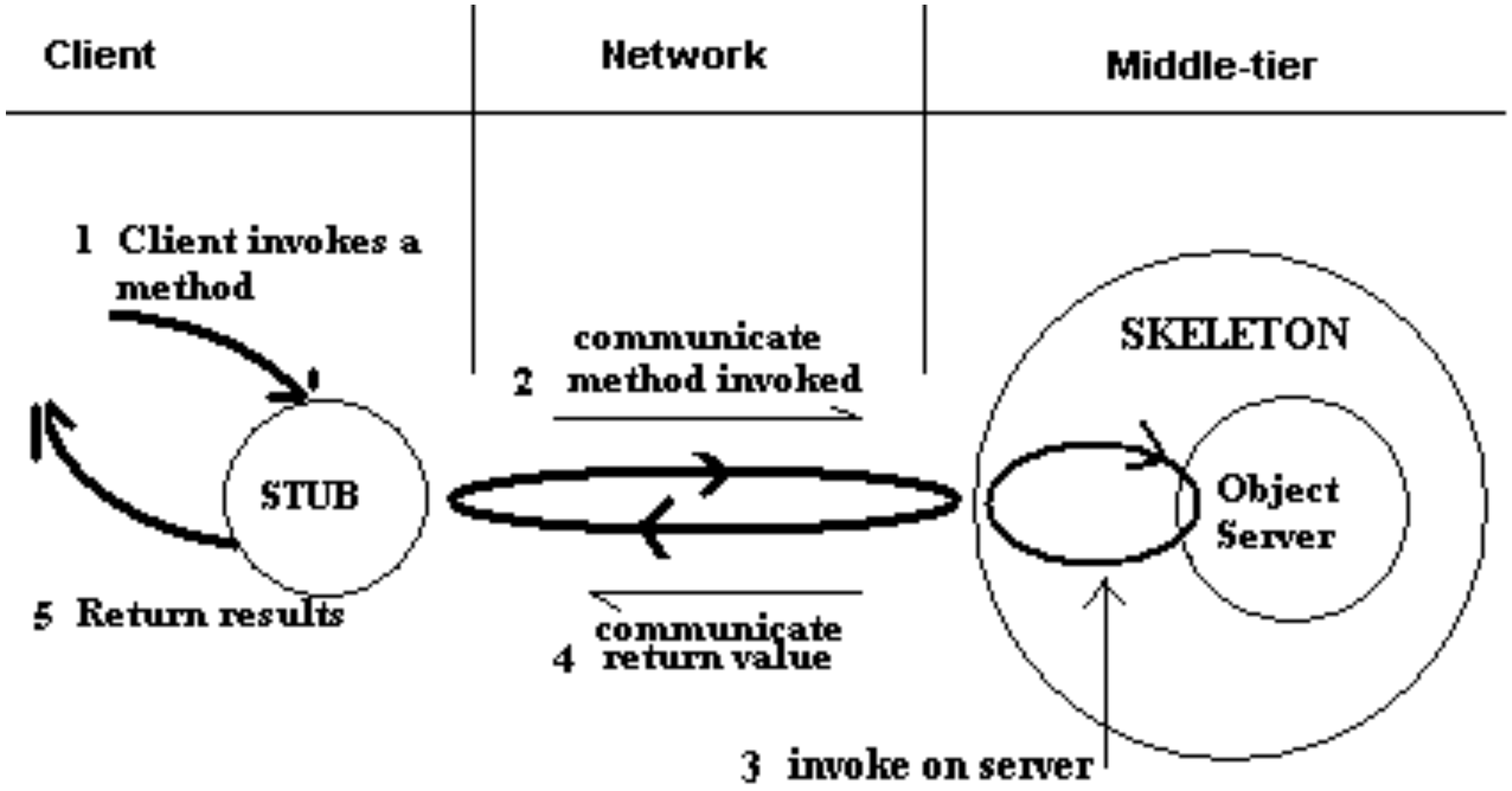
The *remote interface* extends the javax.ejb.EJBObject.

The *home interface* extends javax.ejb.EJBHome interface.

These EJB interface types define a standard set of utility methods and provide common base types for all remote and home interfaces.

Lecture 11

Part I – EJB as Distributed Objects – RMI/IIOP



Lecture 11

Part I – EJB as Distributed Objects – RMI/IIOP

The *remote* and *home interfaces* are types of *Java RMI Remote interfaces*. The *java.rmi.Remote interface* is used by distributed objects to represent the bean in a different address space (process or machine). An enterprise bean is a distributed object.

That means that the bean class is instantiated and lives in the container but it can be accessed by applications that live in other address spaces – in other JVMs and other computer machines.

Lecture 11

Part I – EJB as Distributed Objects – RMI procedure

To make an object instance in one address space available in another requires a little trick involving network sockets. To make the trick work, wrap the instance in a special object called a *skeleton* that has a network connection to another special object called a *stub*. The stub implements the remote interface so it looks like a business object. But the stub doesn't contain business logic; it holds a network socket connection to the skeleton. Every time a business method is invoked on the stub's remote interface, the stub sends a network message to the skeleton telling it which method was invoked. When the skeleton receives a network message from the stub, it identifies the method invoked and the arguments, and then invokes the corresponding method on the actual instance. The instance executes the business method and returns the result to the skeleton, which sends it to the stub.

Lecture 11

Part I – EJB Types in EJB 1.1 and 2.1

EJB Types:

- **1. Entity EJBs**
  - **1.1 CMP – Content Managed Persistence**
  - **1.2 BMP – Bean Managed Persistence**
- **2. Session EJBs**
  - **2.1 Stateless**
  - **2.2 Stateful**
- **3. Message Driven Beans – see JMS and JTA**

Lecture 11

Part I – EJB Types in EJB 3.x

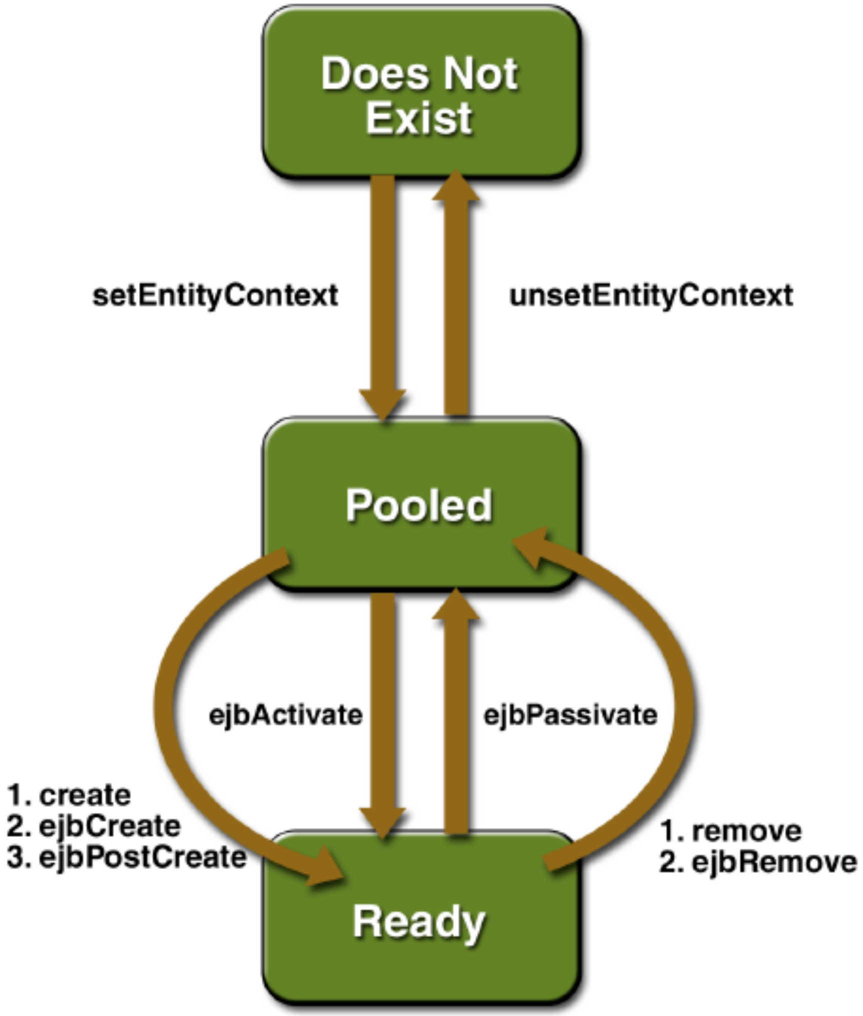
EJB Types:

- **1. Session EJBs**
  - **1.1 Stateless**
  - **1.2 Stateful**
- **2. Message Driven Beans – see JMS and JTA**

- **Entity EJBs => are included within Java Persistence API**
  - **CMP – Container Managed Persistence**
  - **BMP – Bean Managed Persistence**

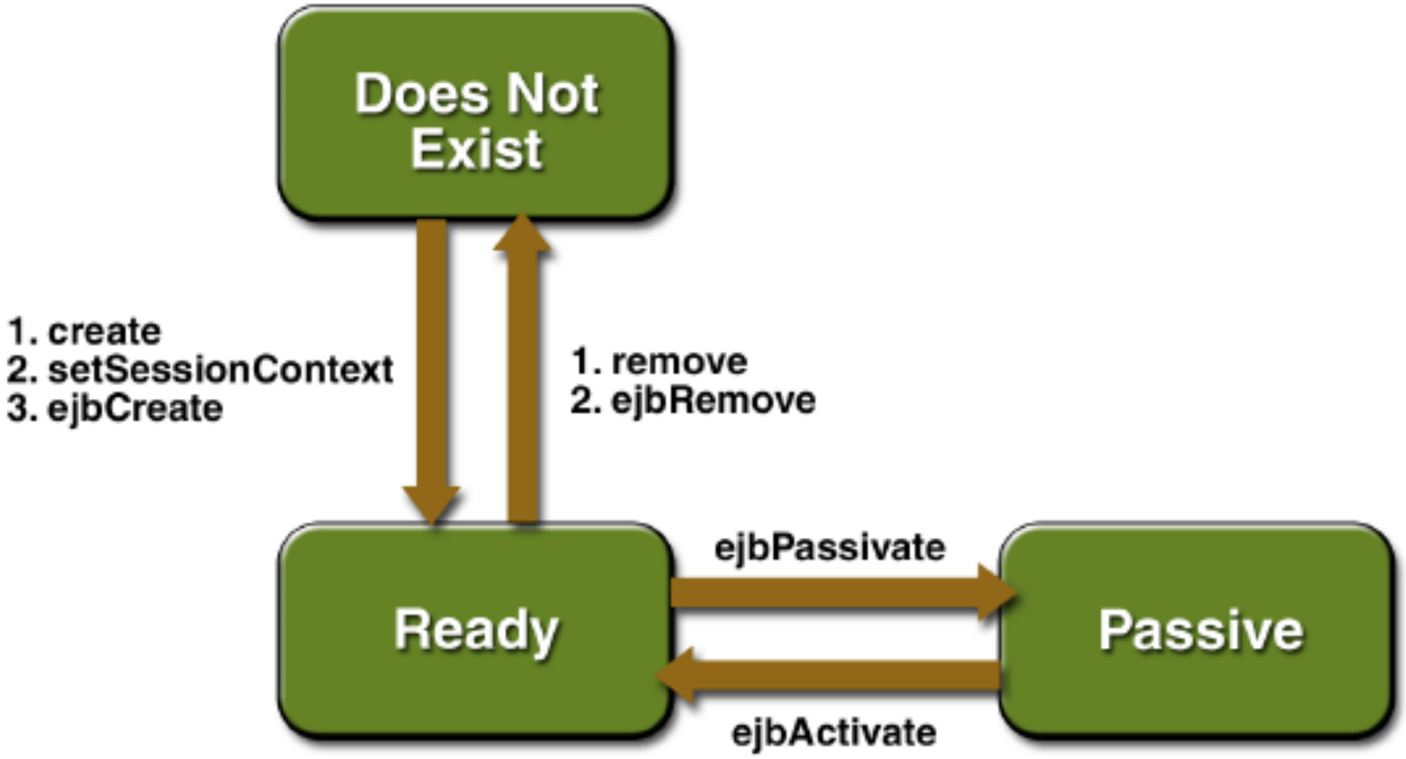
Lecture 11

Part I – Life-cycle of a Entity Bean



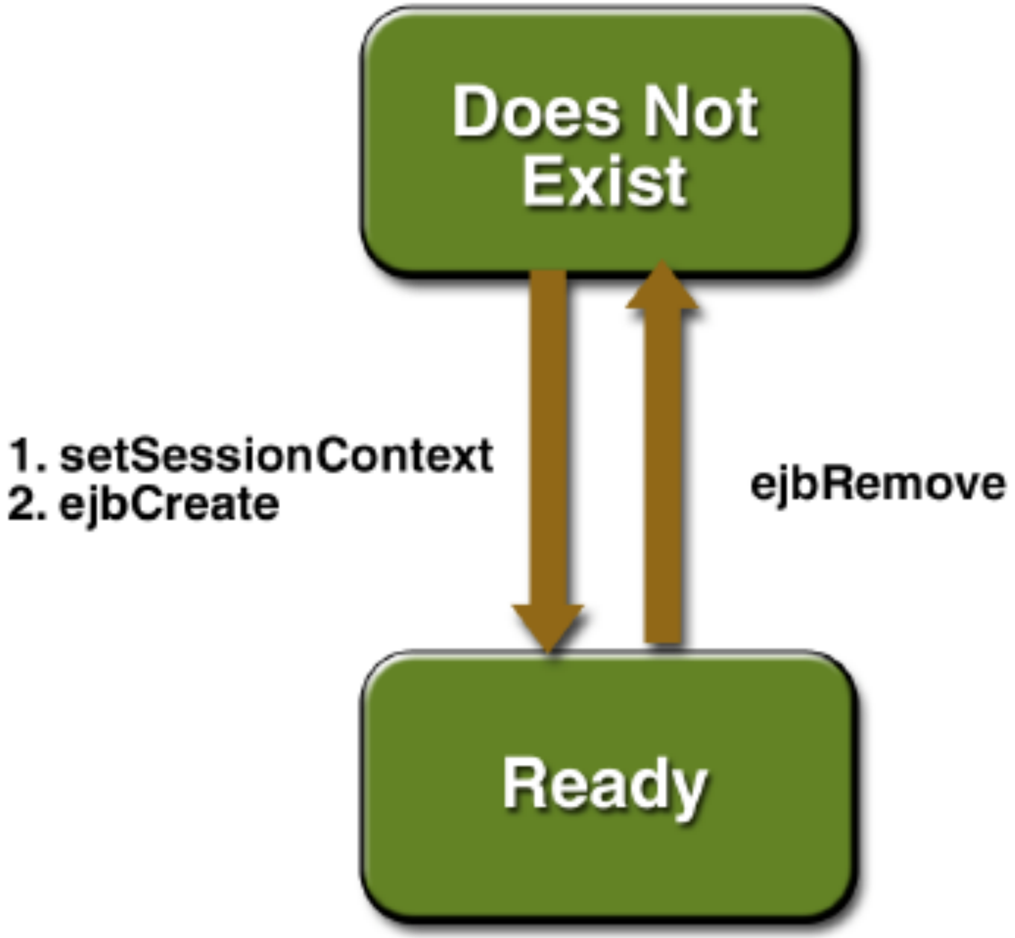
Lecture 11

Part I – Life-cycle of a Stateful Session Bean



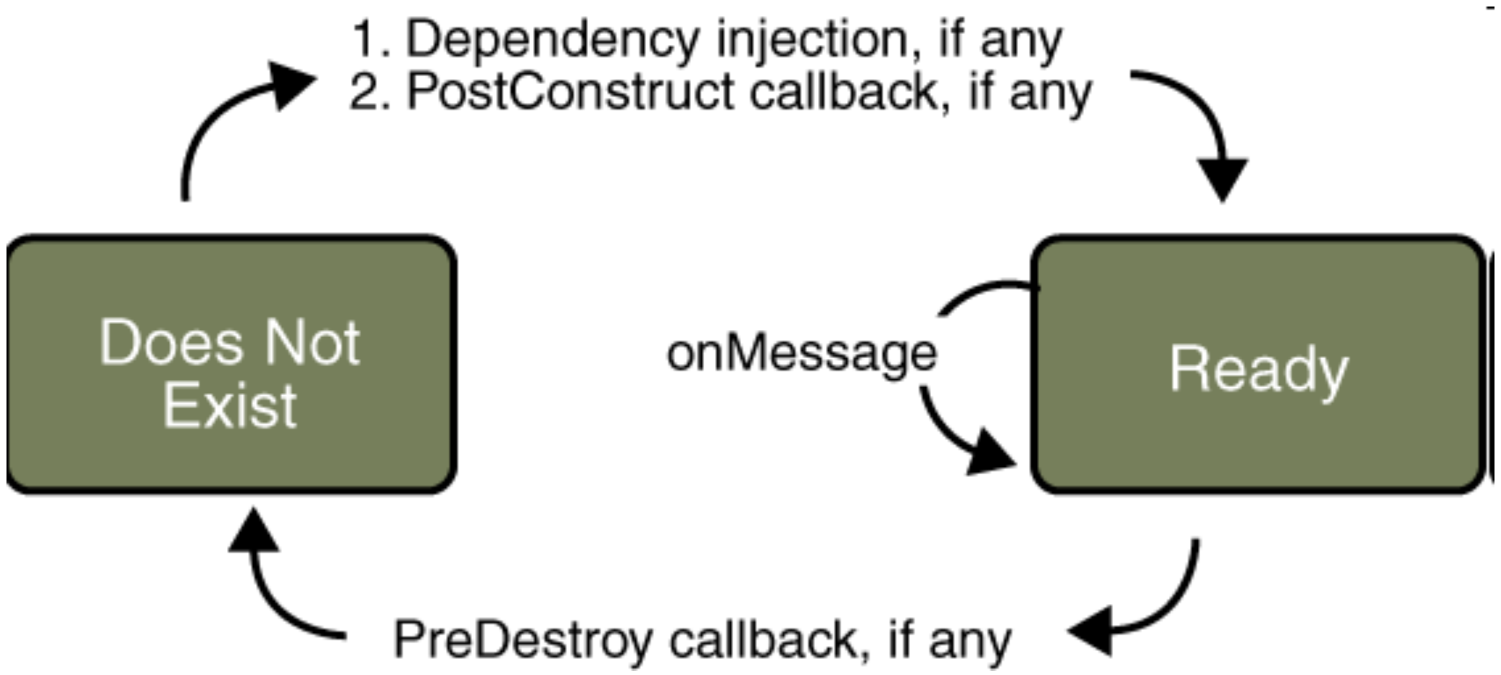
Lecture 11

Part I – Life-cycle of a Stateless Session Bean



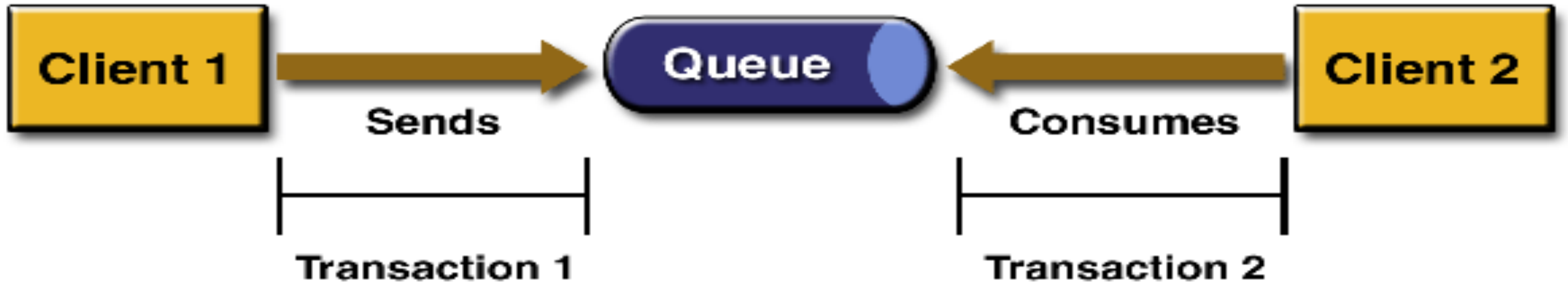
Lecture 11

Part I – Life-cycle of a Message Driven Bean


1. Dependency injection, if any
2. PostConstruct callback, if any
Does Not Exist
onMessage
Ready
PreDestroy callback, if any

Lecture 11

Part I – JMS API API Local Transaction vs. EJB distributed Transaction



Sun: "Distributed transactions can be either of two kinds:

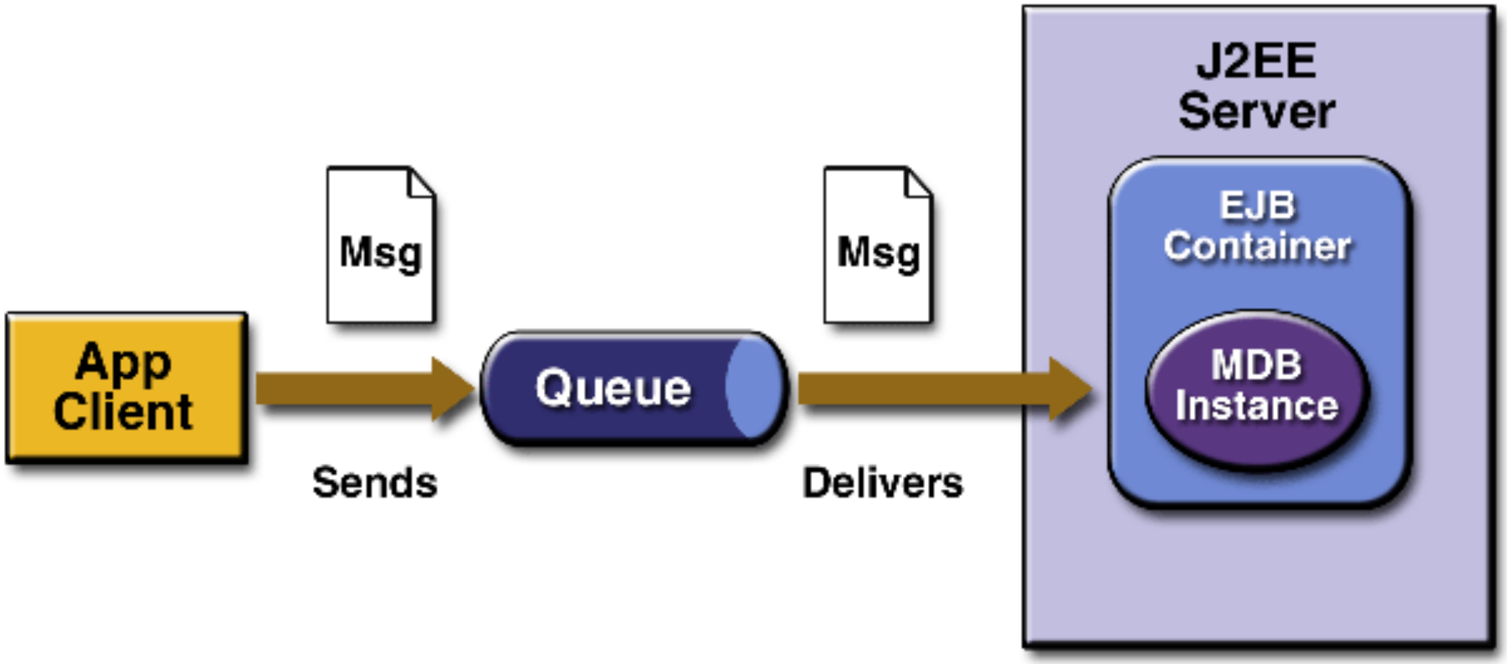
* ***Container-managed transactions***. The EJB container controls the integrity of your transactions without your having to call commit or rollback. Container-managed transactions are recommended for J2EE applications that use the JMS API. You can specify appropriate transaction attributes for your enterprise bean methods. Use the Required transaction attribute to ensure that a method is always part of a transaction. If a transaction is in progress when the method is called, the method will be part of that transaction; if not, a new transaction will be started before the method is called and will be committed when the method returns.

* ***Bean-managed transactions***. You can use these in conjunction with the javax.transaction.UserTransaction interface, which provides its own commit and rollback methods that you can use to delimit transaction boundaries."

Lecture 11

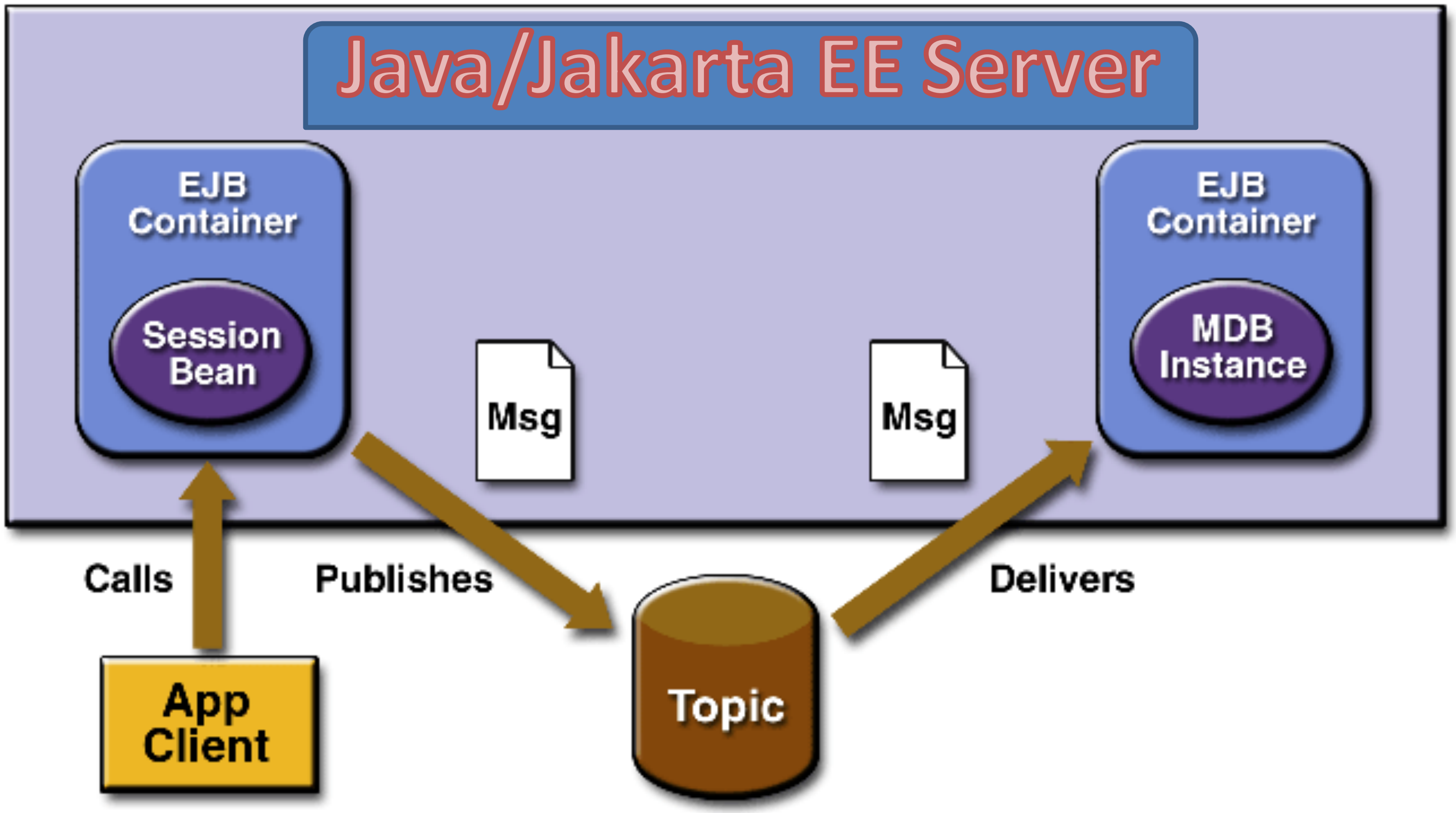
Part I – EJB Life-cycle of a Message Driven Bean

Sun: "Like a stateless session bean, a message-driven bean can have many interchangeable instances running at the same time. The container can pool these instances to allow streams of messages to be processed concurrently. Concurrency can affect the order in which messages are delivered, so you should write your application to handle messages that arrive out of sequence."



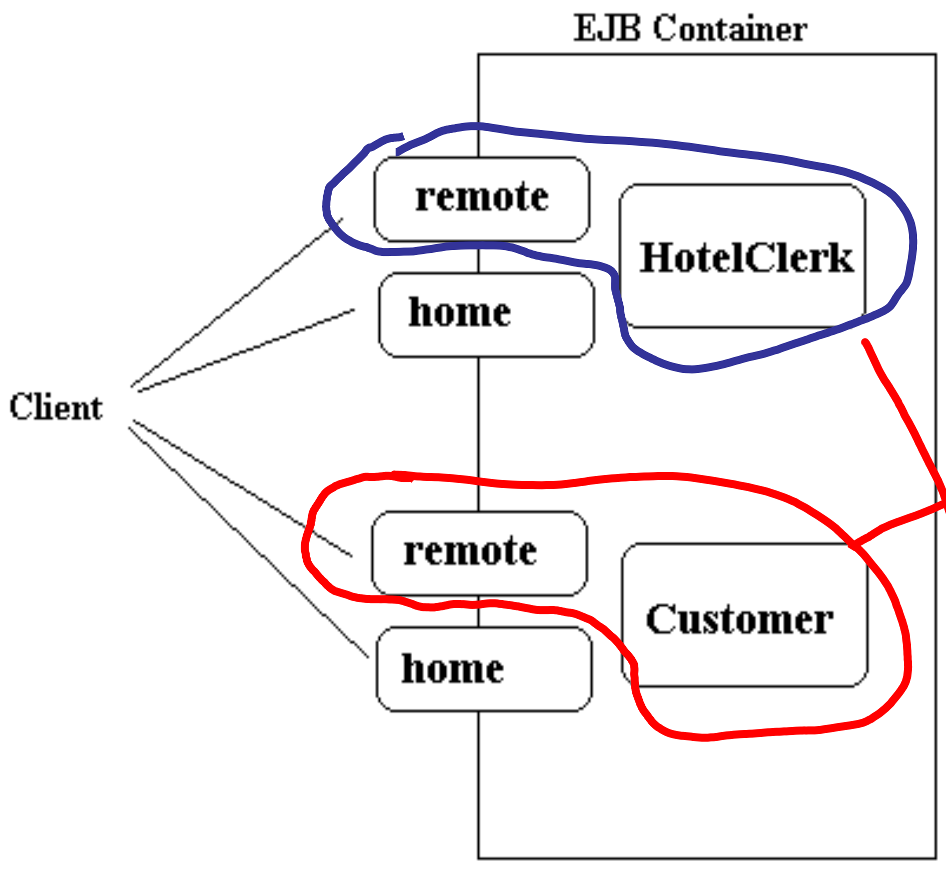
Lecture 11

Part I – EJB Sample for Session + Message Driven Beans



Lecture 11

Part I – Entity vs. Session EJB



Conceptual View of EJB Architecture

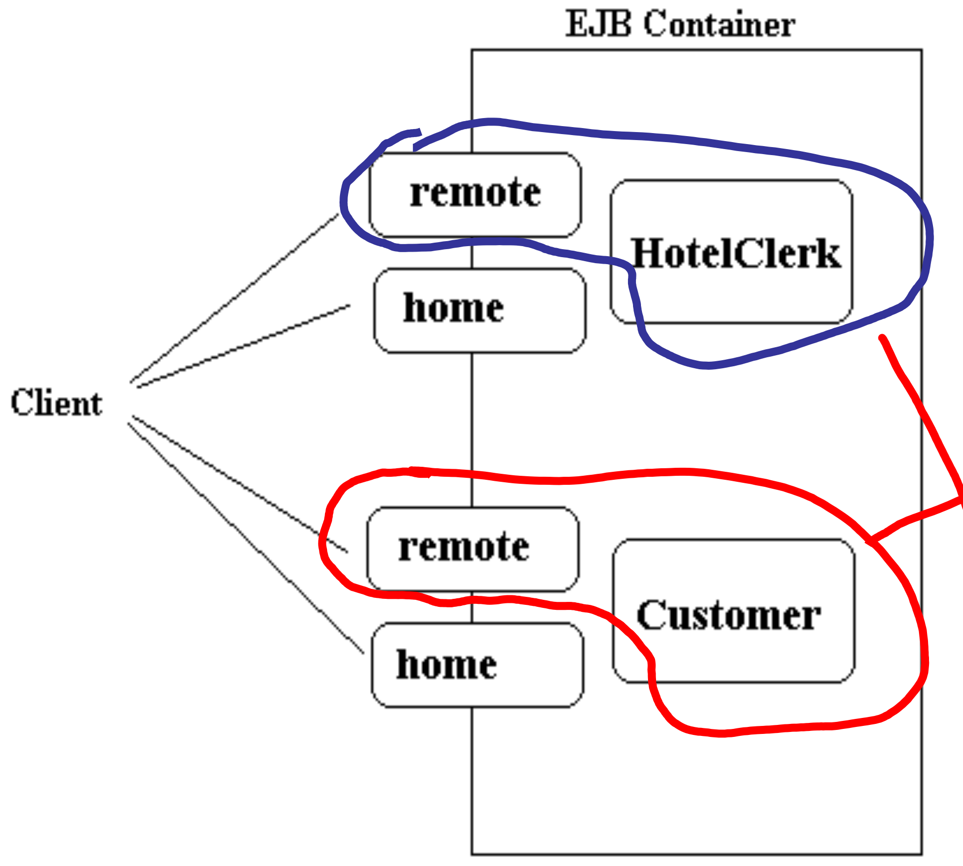
The *remote interface* defines business methods.

The business methods could be:

- *accessor and mutator methods* (get/set) to read and update information about a business concept – like in Customer interface => **entity bean**
- *tasks* that a bean performs - tasks are more typical of a type of bean called a session bean. Session beans do not represent data like entity beans. They represent business processes or agents that perform a service, like making a reservation at a hotel – like in HotelClerk interface => **session bean**.

Lecture 11

Part I – Entity vs. Session EJB



Conceptual View of EJB Architecture

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface Customer extends EJBObject {

    public Name getName()
                throws RemoteException;
    public void setName(Name name)
            throws RemoteException;
    public Address getAddress()
                throws RemoteException;
    public void setAddress(Address address)
                    throws RemoteException;

}
```

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface HotelClerk
               extends EJBObject {

   public void reserveRoom(Customer cust,
               RoomInfo ri,Date from, Date to)
                      throws RemoteException;

   public RoomInfo availableRooms(
        Location loc, Date from, Date to)
                 throws RemoteException;
}
```

- **1. Entity EJBs**

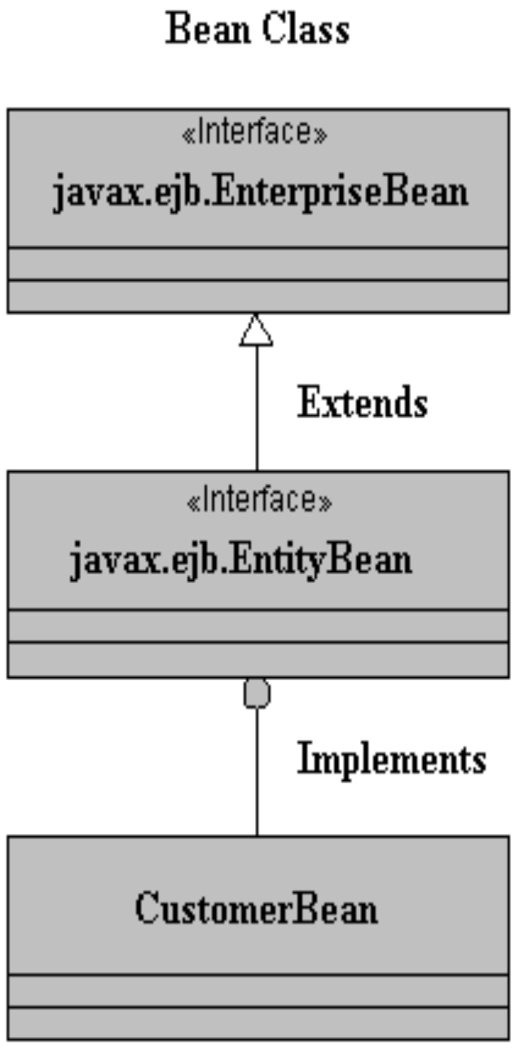
CustomerHome.java – Home interface EJB life-cycle methods.

```
import javax.ejb.EJBHome;
import javax.ejb.CreateException;
import javax.ejb.FinderException;
import java.rmi.RemoteException;

public interface CustomerHome
                  extends EJBHome }

    public Customer create(Integer
                     customerNumber)
                    throws RemoteException,
                           CreateException;

    public Customer findByPrimaryKey(Integer
                              customerNumber)
                    throws RemoteException,
                            FinderException;

    public Enumeration findByZipCode(int zipCode)
                       throws RemoteException,
                              FinderException;
}
```

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface Customer extends EJBObject {

    public Name getName()
                throws RemoteException;
    public void setName(Name name)
           throws RemoteException;
    public Address getAddress()
             throws RemoteException;
    public void setAddress(Address address)
                     throws RemoteException;
```

Customer.java – Remote Interface – business methods – here get/set.

- **1. Entity EJBs**



Class Diagram for the Bean Class

```
import javax.ejb.EntityBean;

public class CustomerBean
                implements EntityBean {

    Address    myAddress;
    Name       myName;
    CreditCard myCreditCard;

    public Name getName() {
         return myName;
    }

    public void setName(Name name) {
         myName = name;
    }

    public Address getAddress() {
         return myAddress;
    }

    public void setAddress(Address address) {
         myAddress = address;
    }

    ...
}
```

CustomerBean.java – EJB entity class

Lecture 11

Part I – Entity EJB / JPA – Java Persistence API

- **1. Entity EJBs –** **The entity bean is used to represent data in the database. It provides an object-oriented interface to data that would normally be accessed by the JDBC or some other back-end API.**
  - ***1.1 CMP – Container Managed Persistence*** **– the container manages the persistence of the entity bean. Vendor tools are used to map the entity fields to the database and absolutely no database access code is written in the bean class.**
  - ***1.2 BMP – Bean Managed Persistence*** **– the entity bean contains database access code (usually JDBC) and is responsible for reading and writing its own state to the database. BMP entities have a lot of help with this since the container will alert the bean as to when it's necessary to make an update or read its state from the database.**

Lecture 11

Partea II – EJB Types

- **2. Session EJBs – Session beans are used to manage the interactions of entity and other session beans, access resources, and generally perform tasks on behalf of the client. Session beans are not persistent business objects as are entity beans. They do not represent data in the database.**
  - ***2.1 Stateless*** **– session beans are made up of business methods that behave like procedures; they operate only on the arguments passed to them when they are invoked. Stateless beans are called "stateless" because they are transient; they do not maintain business state between method invocations.**
  - ***2.2 Stateful*** **– Stateful session beans encapsulate business logic and state specific to a client. Stateful beans are called "stateful" because they do maintain business state between method invocations, held in memory and not persistent.**

- **2. Session EJBs**

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface HotelClerk
               extends EJBObject {

    public void reserveRoom(Customer cust,
                     RoomInfo ri,Date from, Date to)
                              throws RemoteException;

    public RoomInfo availableRooms(
         Location loc, Date from, Date to)
                        throws RemoteException;
}
```

HotelClerkHome.java – Home interface – EJB life-cycle methods.

HotelClerk.java – Remote Interface – business methods – here TASKS.

```
import java.rmi.RemoteException;
import javax.ejb.*;


public interface HotelClerkHome extends EJBHome {
    public HotelClerk create() throws CreateException, RemoteException;
}
```

```java
import javax.ejb.SessionBean;
import javax.naming.InitialContext;

//STATELESS
public interface HotelClerkBean implements SessionBean {
    InitialContext jndiContext;
    public void ejbCreate() {}

    public void reserveRoom(Customer cust, RoomInfo ri, Date from, Date to) {
        CreditCard card = cust.getCreditCard();
        RoomHome roomHome = (RoomHome)
        getHome("java:comp/env/ejb/RoomEJB", RoomHome.class);
        Room room = roomHome.findByPrimaryKey(ri.getID());
        double amount = room.getPrice(from,to);

        CreditServiceHome creditHome = (CreditServiceHome)
        getHome("java:comp/env/ejb/CreditServiceEJB", CreditServiceHome.class);
        CreditService creditAgent = creditHome.create();
        creditAgent.verify(card, amount);

        ReservationHome resHome =
                (ReservationHome) getHome("java:comp/env/ejb/ReservationEJB",
                                   ReservationHome.class);
        Reservation reservation = resHome.create(cust.getName(), room,from,to);
    }
```

- **2. Session EJBs - Stateless**

HotelClerkBean.java – Stateless Session EJB.

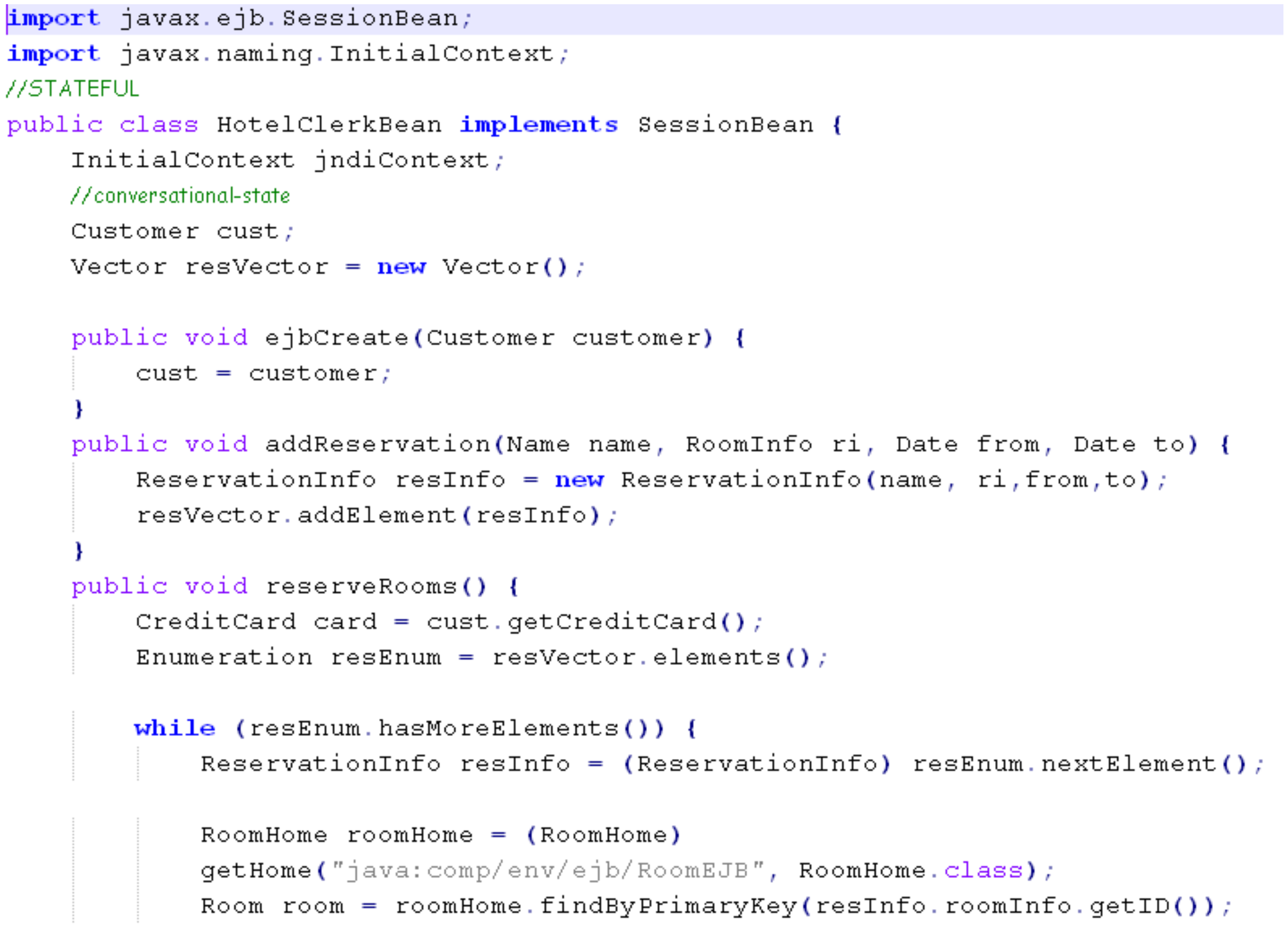
Lecture 11

Part I – Session Stateless EJB

```
public RoomInfo[] availableRooms(Location loc, Date from, Date to) {
    // do a SQL call to find available rooms
    Connection con = db.getConnection();
    Statement stmt = con.createStatement();
    ResultSet results = stmt.executeQuery("SELECT ...");
    ...
    return roomInfoArray;
}

private Object getHome(String path, Class type) {
    Object ref = jndiContext.lookup(path);
    return PortableRemoteObject.narrow(ref,type);
}
```

- **2. Session EJBs - Stateless**

HotelClerkBean.java – Stateless Session EJB.



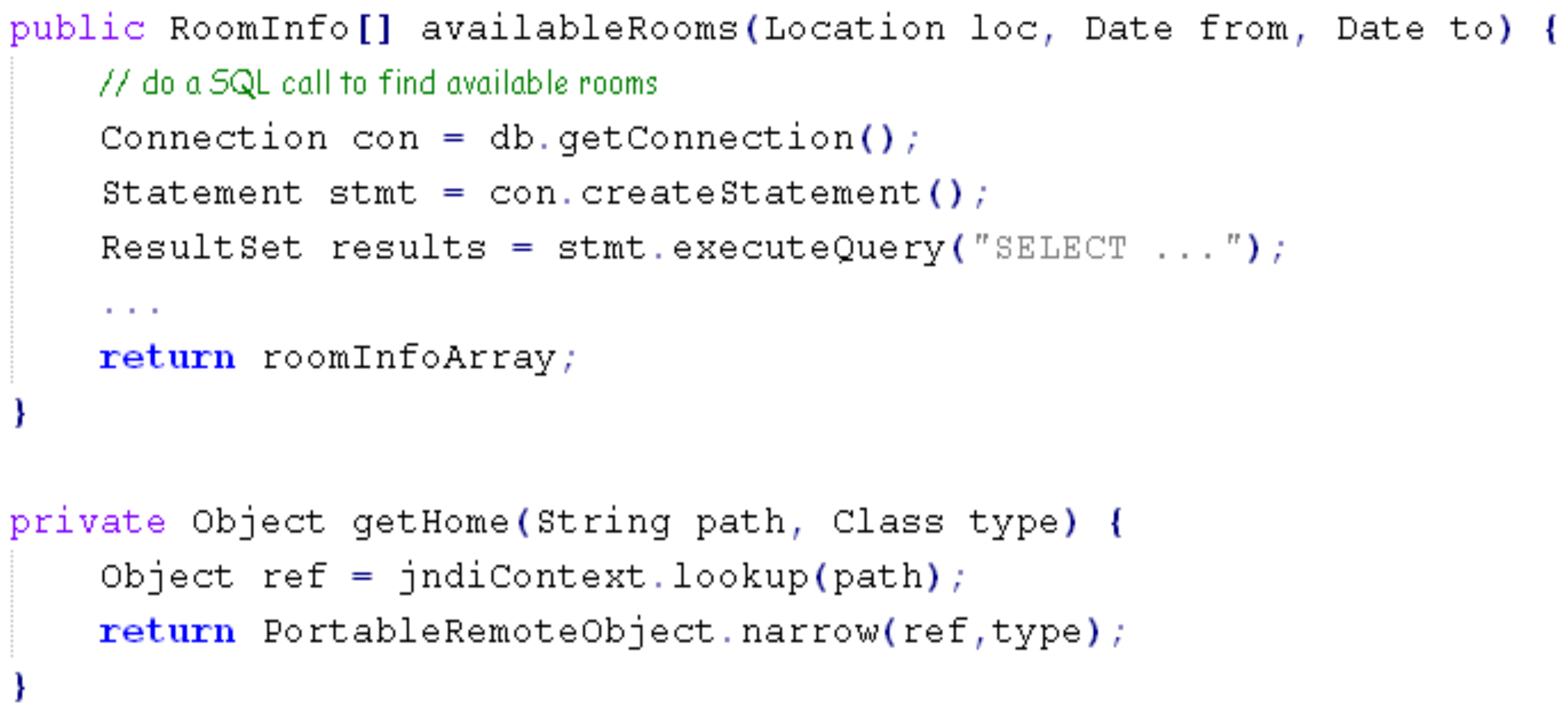
```java
import javax.ejb.SessionBean;
import javax.naming.InitialContext;
//STATEFUL
public class HotelClerkBean implements SessionBean {
    InitialContext jndiContext;
    //conversational-state
    Customer cust;
    Vector resVector = new Vector();

    public void ejbCreate(Customer customer) {
        cust = customer;
    }
    public void addReservation(Name name, RoomInfo ri, Date from, Date to) {
        ReservationInfo resInfo = new ReservationInfo(name, ri,from,to);
        resVector.addElement(resInfo);
    }
    public void reserveRooms() {
        CreditCard card = cust.getCreditCard();
        Enumeration resEnum = resVector.elements();

        while (resEnum.hasMoreElements()) {
            ReservationInfo resInfo = (ReservationInfo) resEnum.nextElement();

            RoomHome roomHome = (RoomHome)
            getHome("java:comp/env/ejb/RoomEJB", RoomHome.class);
            Room room = roomHome.findByPrimaryKey(resInfo.roomInfo.getID());
```

- ## 2. Session EJBs - Stateful

HotelClerkBean.java – Stateful Session EJB.

Lecture 11

Part I – Session Stateful EJB



```
public RoomInfo[] availableRooms(Location loc, Date from, Date to) {
    // do a SQL call to find available rooms
    Connection con = db.getConnection();
    Statement stmt = con.createStatement();
    ResultSet results = stmt.executeQuery("SELECT ...");
    ...
    return roomInfoArray;
}

private Object getHome(String path, Class type) {
    Object ref = jndiContext.lookup(path);
    return PortableRemoteObject.narrow(ref,type);
}
```

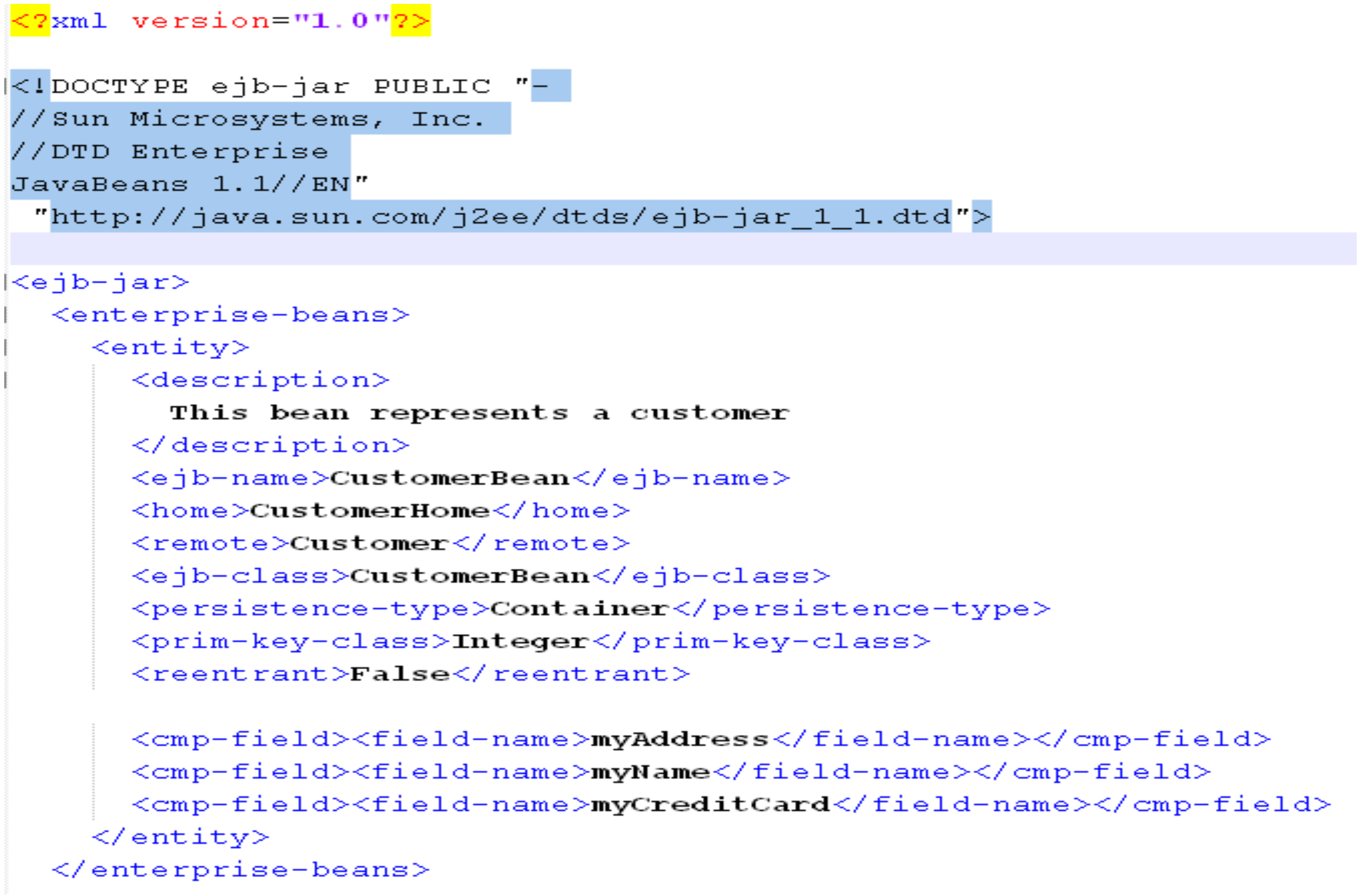
- **2. Session EJBs - Stateful**

HotelClerkBean.java – Stateful Session EJB.

Lecture 11

Part I – Deprecated Deploy EJB 1.1 and 2.1

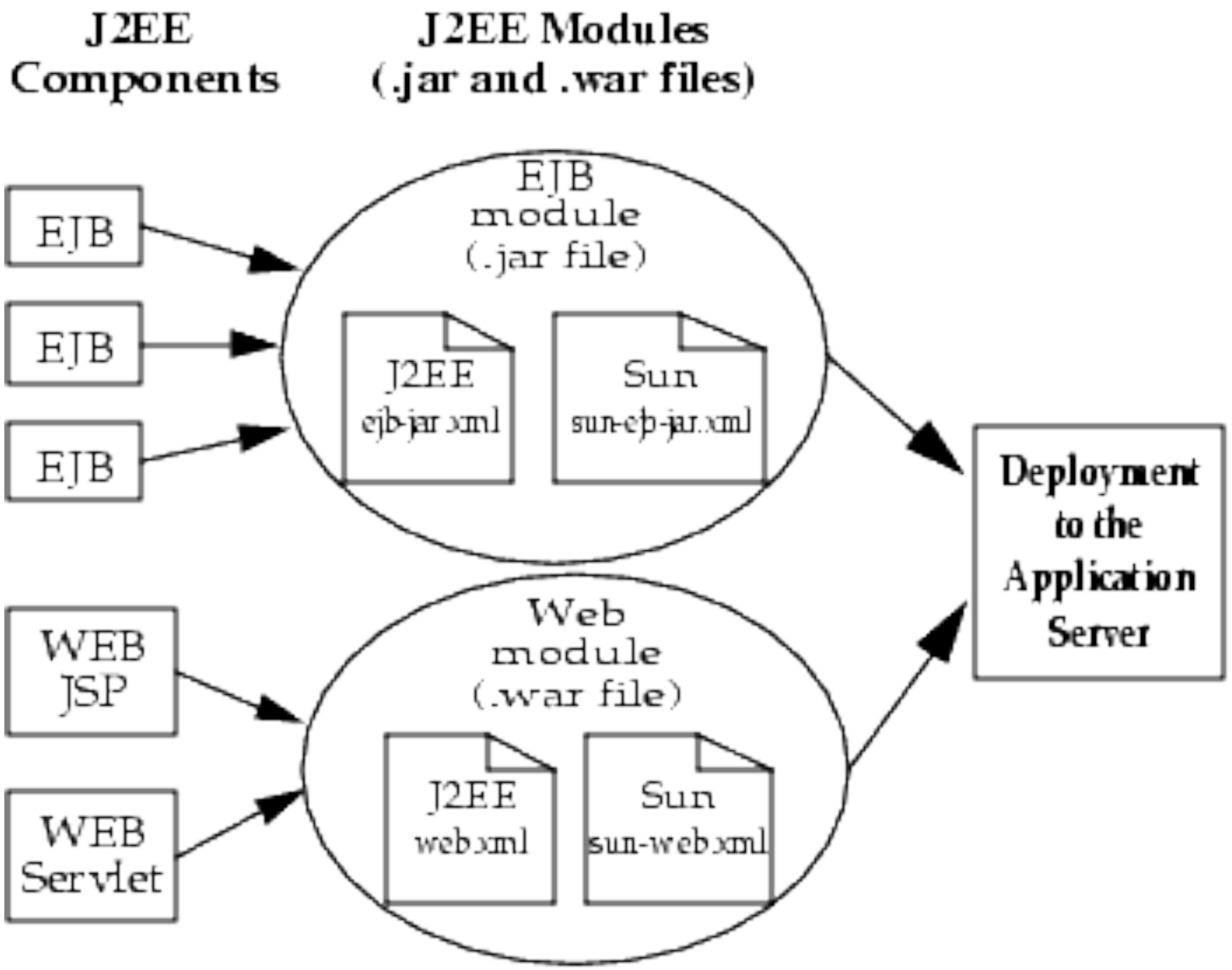
In EJB JAR file – within META-INF/ejb-jar.xml.

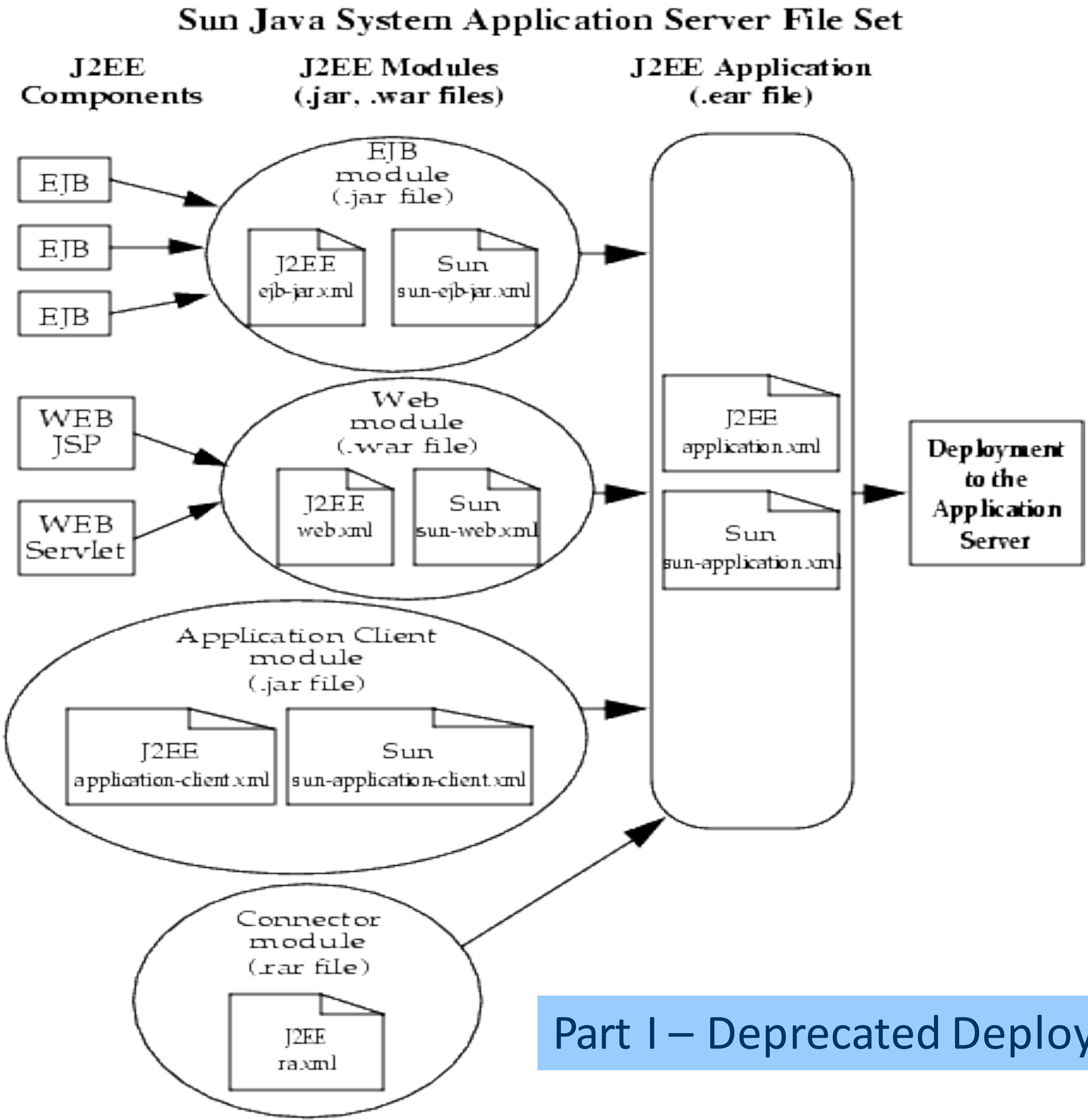


```
<?xml version="1.0"?>

<!DOCTYPE ejb-jar PUBLIC "-
//Sun Microsystems, Inc.
//DTD Enterprise
JavaBeans 1.1//EN"
 "http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd">

<ejb-jar>
  <enterprise-beans>
    <entity>
      <description>
        This bean represents a customer
      </description>
      <ejb-name>CustomerBean</ejb-name>
      <home>CustomerHome</home>
      <remote>Customer</remote>
      <ejb-class>CustomerBean</ejb-class>
      <persistence-type>Container</persistence-type>
      <prim-key-class>Integer</prim-key-class>
      <reentrant>False</reentrant>

      <cmp-field><field-name>myAddress</field-name></cmp-field>
      <cmp-field><field-name>myName</field-name></cmp-field>
      <cmp-field><field-name>myCreditCard</field-name></cmp-field>
    </entity>
  </enterprise-beans>
```

Lecture 11

Part I – Deprecated Deploy EJB 1.1 and 2.1

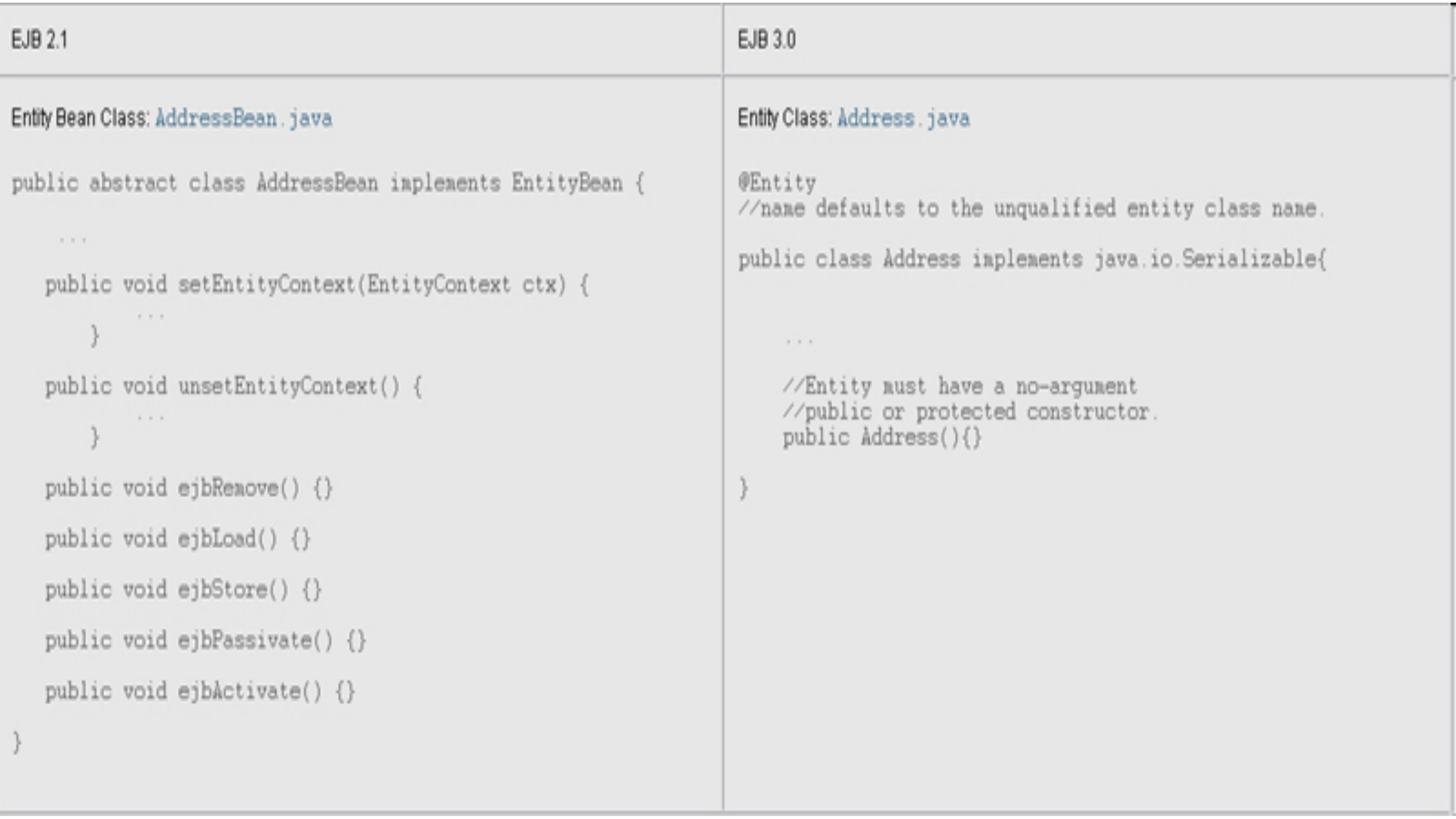



Sun Java System Application Server File Set
J2EE Components
J2EE Modules (.jar, .war files)
J2EE Application (.ear file)
EJB
EJB
EJB
EJB module (.jar file)
J2EE ejb-jar.xml
Sun sun-ejb-jar.xml
WEB JSP
WEB Servlet
Web module (.war file)
J2EE web.xml
Sun sun-web.xml
Application Client module (.jar file)
J2EE application-client.xml
Sun sun-application-client.xml
Connector module (.rar file)
J2EE ra.xml
J2EE application.xml
Sun sun-application.xml
Deployment to the Application Server

Part I – Deprecated Deploy EJB 1.1 and 2.1

Lecture 11

Part I – EJB 1.1/2.1 vs. EJB 3.0



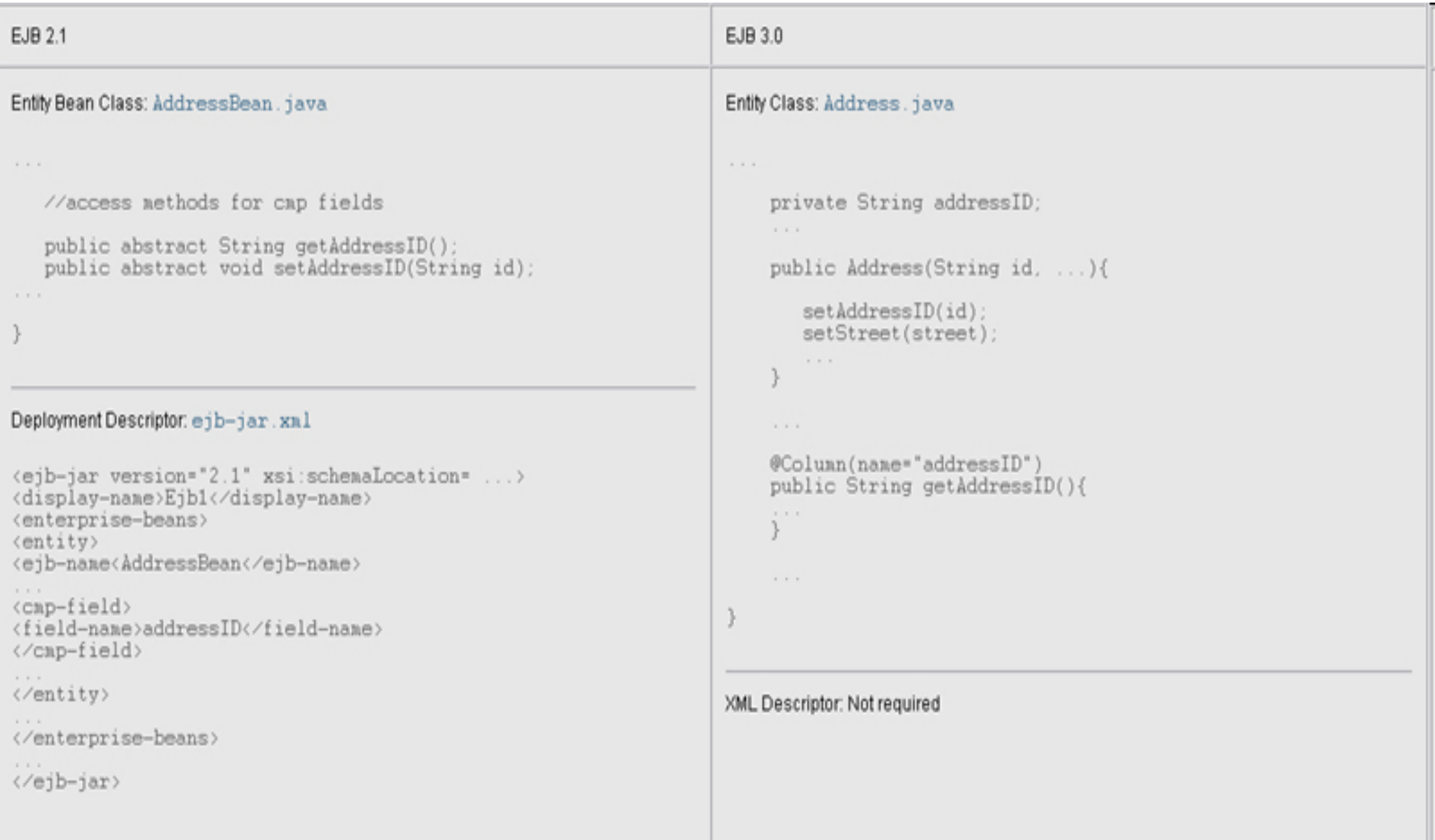
EJB 2.1	EJB 3.0
Entity Bean Class: AddressBean.java	Entity Class: Address.java

```
public abstract class AddressBean implements EntityBean {

    ...

   public void setEntityContext(EntityContext ctx) {
          ...
      }

   public void unsetEntityContext() {
          ...
      }

   public void ejbRemove() {}

   public void ejbLoad() {}

   public void ejbStore() {}

   public void ejbPassivate() {}

   public void ejbActivate() {}

}
```

```
@Entity
//name defaults to the unqualified entity class name.

public class Address implements java.io.Serializable{


    ...

    //Entity must have a no-argument
    //public or protected constructor.
    public Address(){}

}
```

Lecture 11

Part I – EJB 1.1/2.1 vs. EJB 3.0



EJB 2.1

Entity Bean Class: `AddressBean.java`

```
...

   //access methods for cmp fields

   public abstract String getAddressID();
   public abstract void setAddressID(String id);
...

}
```

Deployment Descriptor: `ejb-jar.xml`

```
<ejb-jar version="2.1" xsi:schemaLocation= ...>
<display-name>Ejb1</display-name>
<enterprise-beans>
<entity>
<ejb-name<AddressBean</ejb-name>
...
<cmp-field>
<field-name>addressID</field-name>
</cmp-field>
...
</entity>
...
</enterprise-beans>
...
</ejb-jar>
```

EJB 3.0

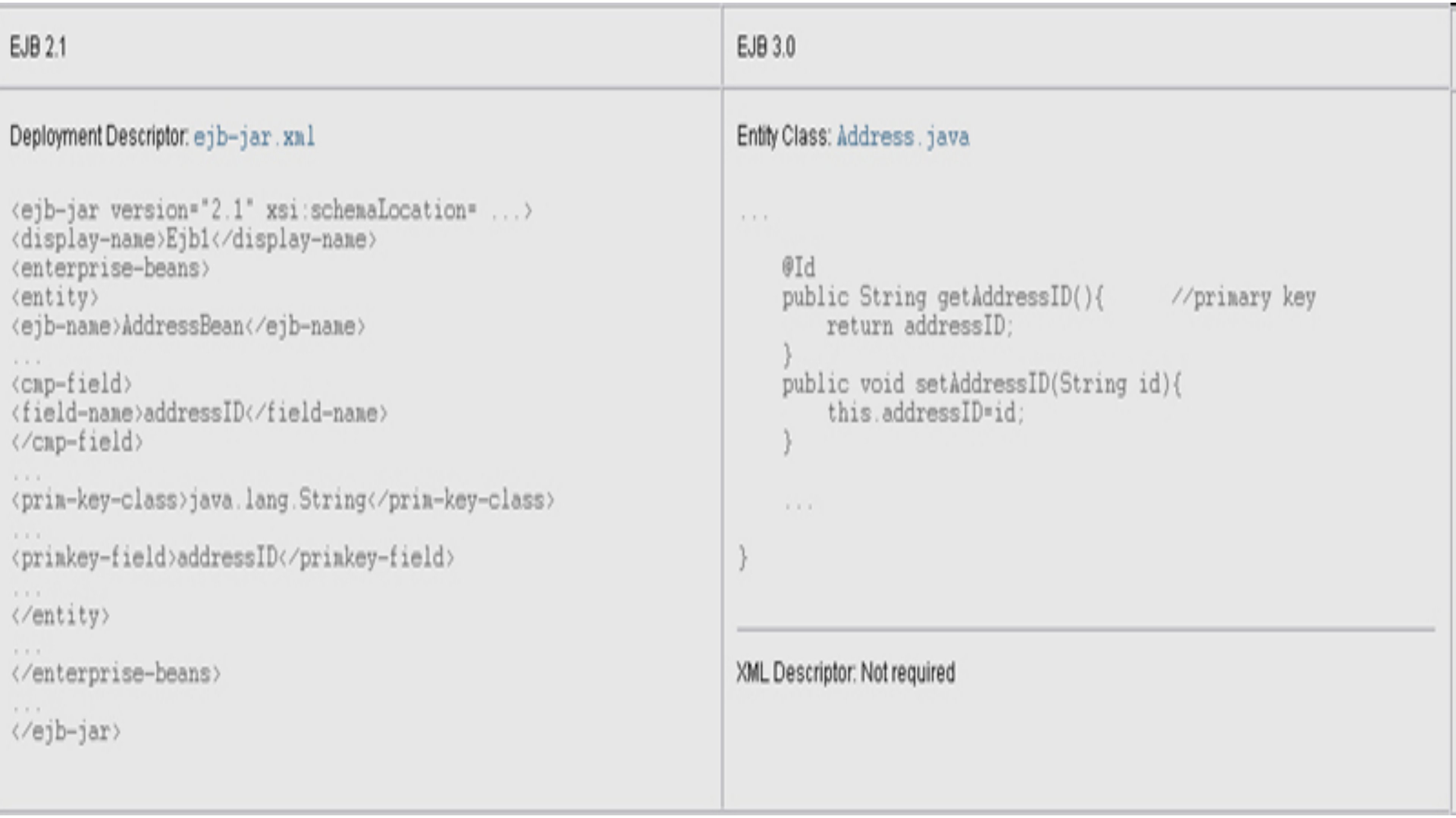
Entity Class: `Address.java`

```
...

    private String addressID;
    ...

    public Address(String id, ...){

       setAddressID(id);
       setStreet(street);
       ...
    }

    ...

    @Column(name="addressID")
    public String getAddressID(){
    ...
    }

    ...

}
```

XML Descriptor: Not required

Lecture 11

Part I – EJB 1.1/2.1 vs. EJB 3.0



EJB 2.1

Deployment Descriptor: ejb-jar.xml

```
<ejb-jar version="2.1" xsi:schemaLocation= ...>
<display-name>Ejb1</display-name>
<enterprise-beans>
<entity>
<ejb-name>AddressBean</ejb-name>
...
<cmp-field>
<field-name>addressID</field-name>
</cmp-field>
...
<prim-key-class>java.lang.String</prim-key-class>
...
<primkey-field>addressID</primkey-field>
...
</entity>
...
</enterprise-beans>
...
</ejb-jar>
```

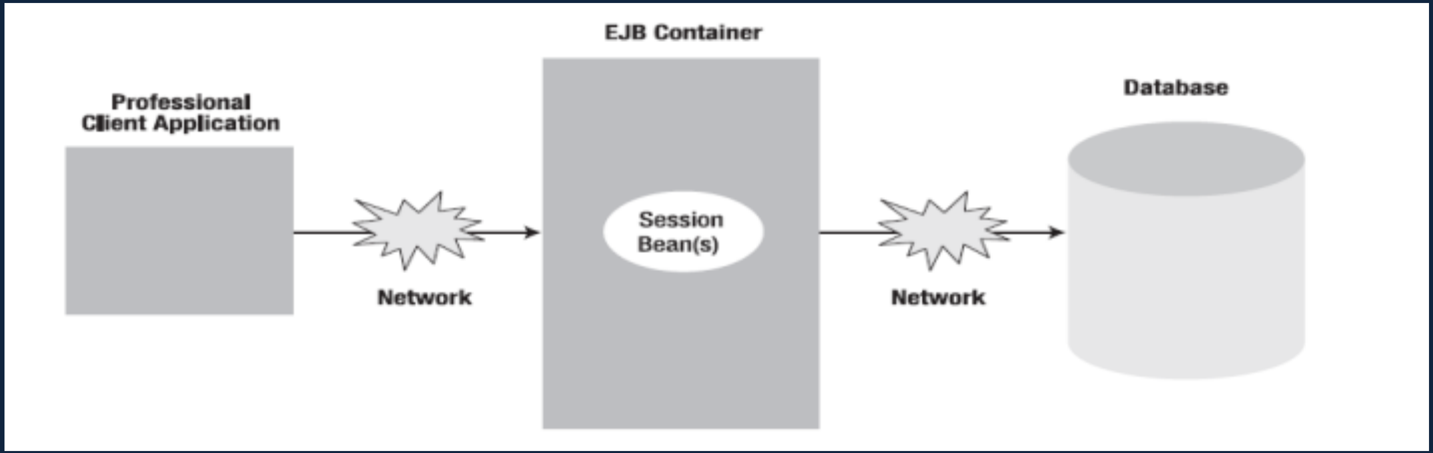
EJB 3.0

Entity Class: Address.java

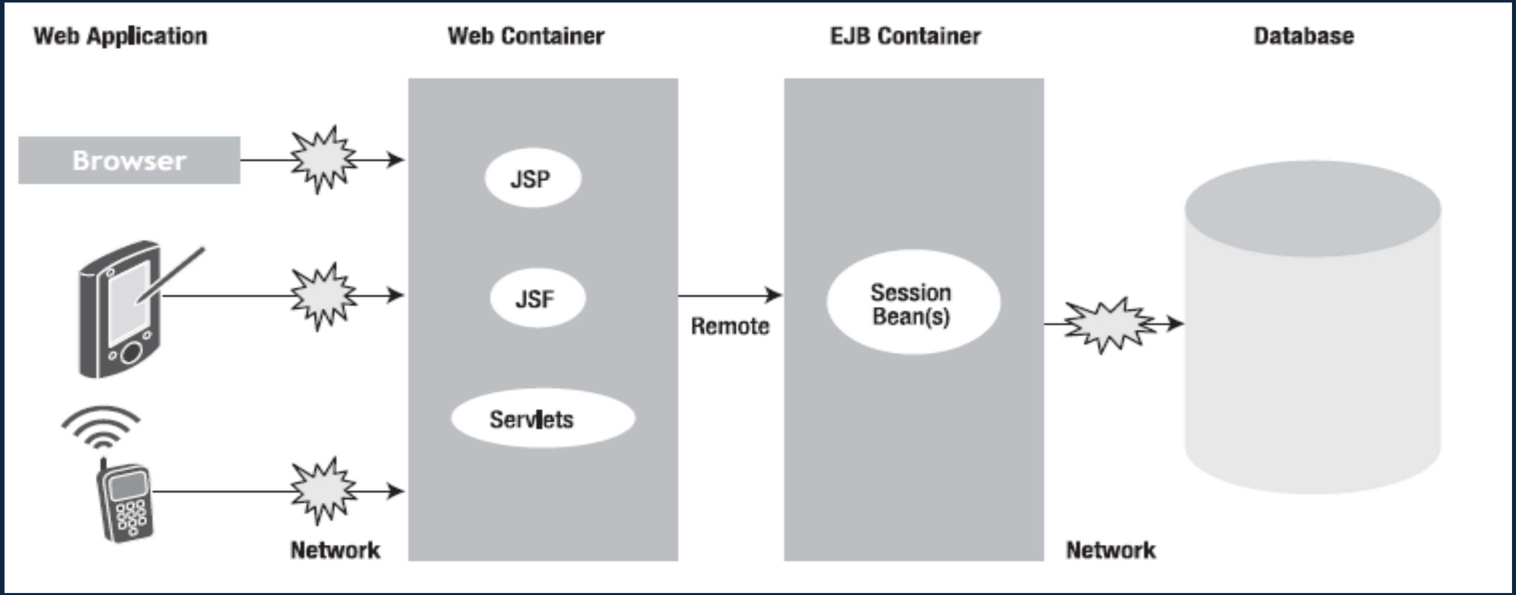
```
...

    @Id
    public String getAddressID(){      //primary key
        return addressID;
    }
    public void setAddressID(String id){
        this.addressID=id;
    }

    ...

}
```

XML Descriptor: Not required

EJB Architecture Distribution Model



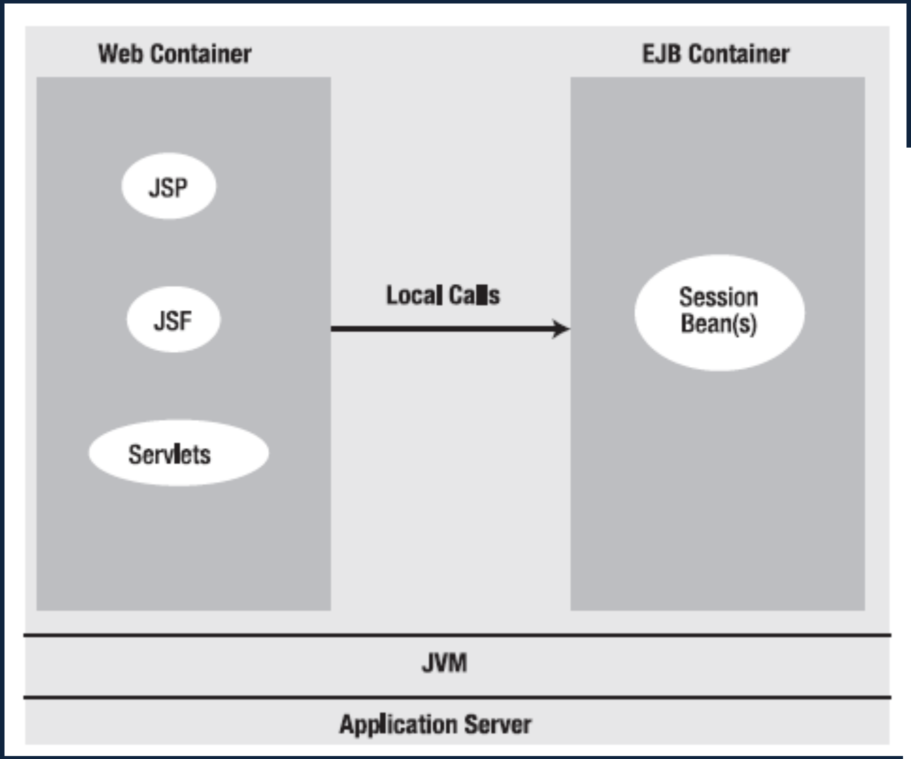
Session beans in 3-tiers architecture

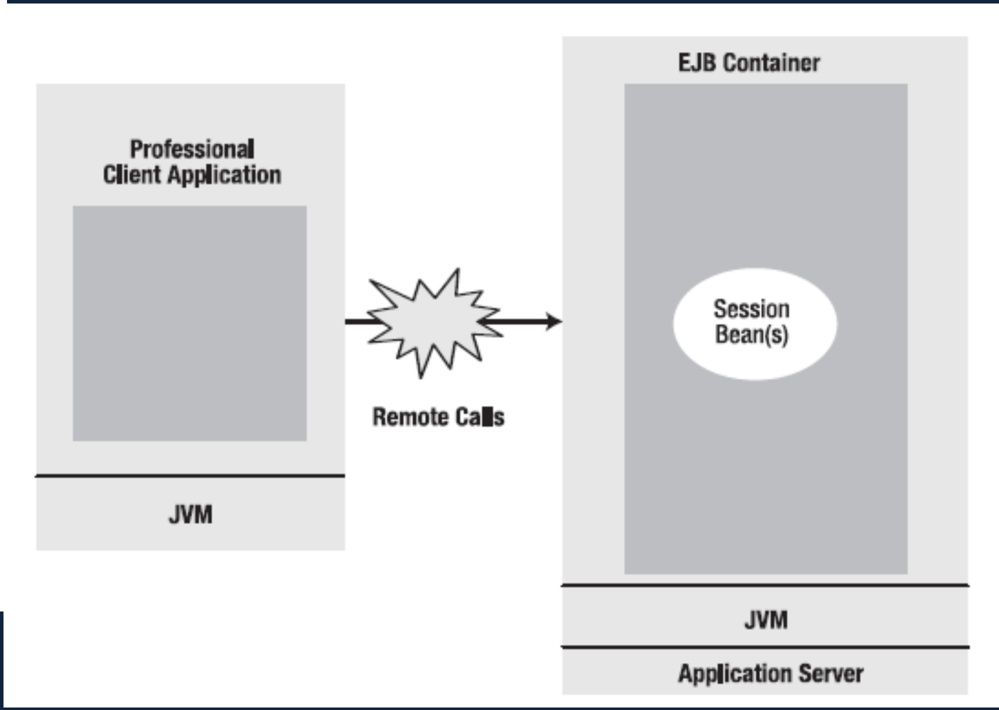


Session beans in 3-tiers architecture with a web application

EJB Architecture Distribution Model

Web Client using local interfaces of session beans





Rich Client using remote interfaces of session beans

Section Conclusion

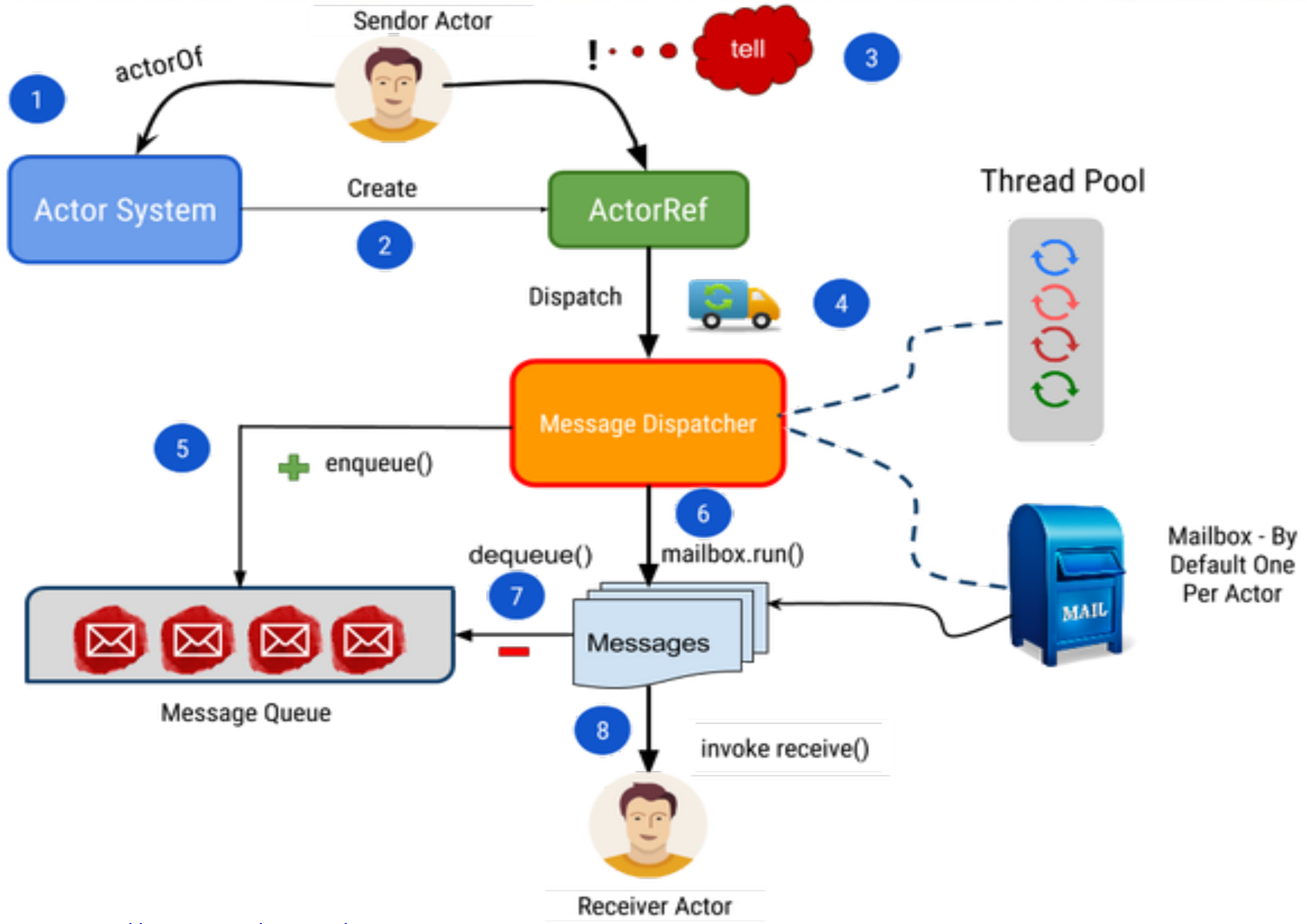
Fact: **DAD middleware is exploring EJB**

In few **samples** it is simple to remember: EJB – Enterprise Java Beans

Actors and Distributed Systems

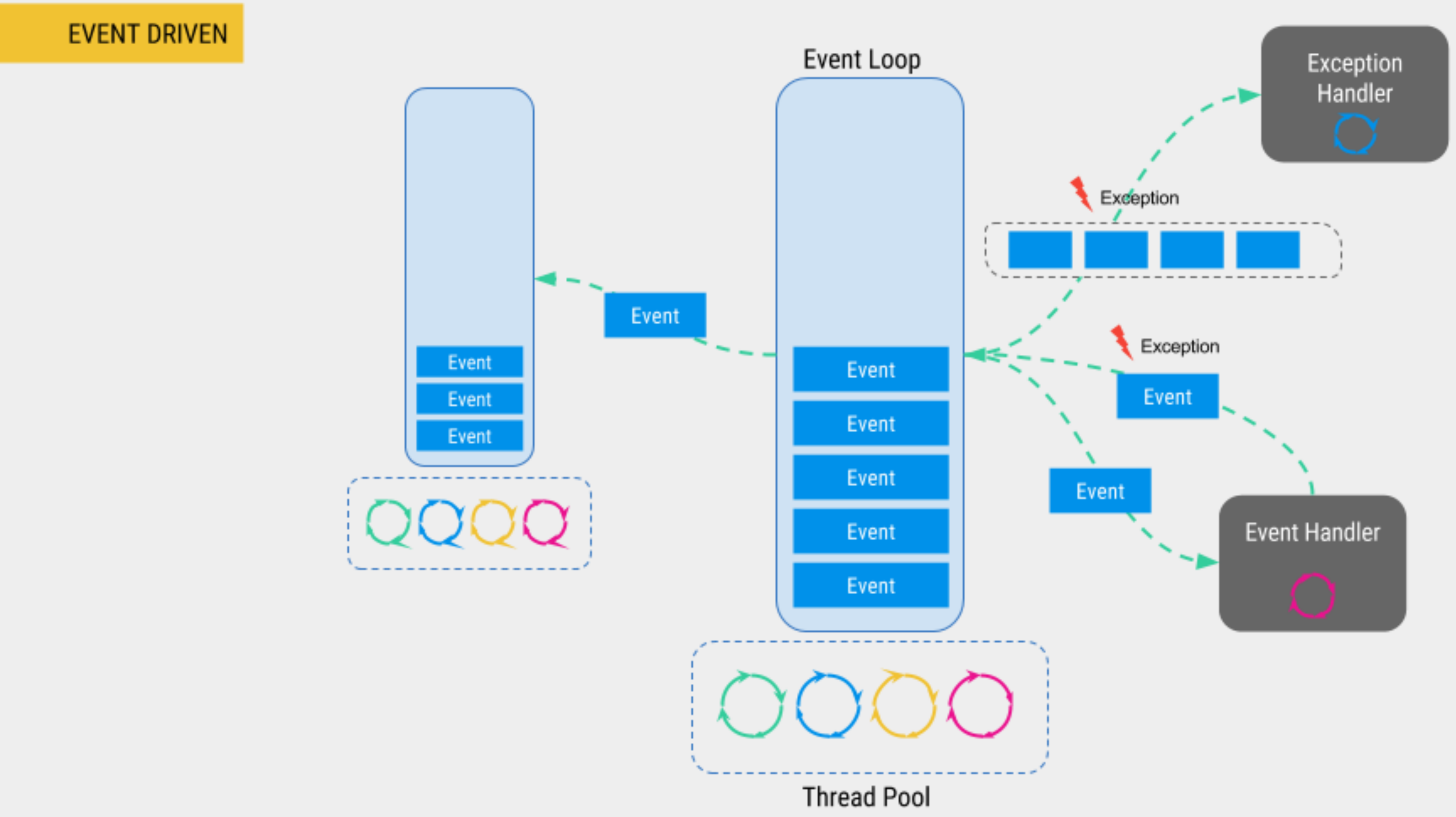
Akka Actors

2. Akka Actors

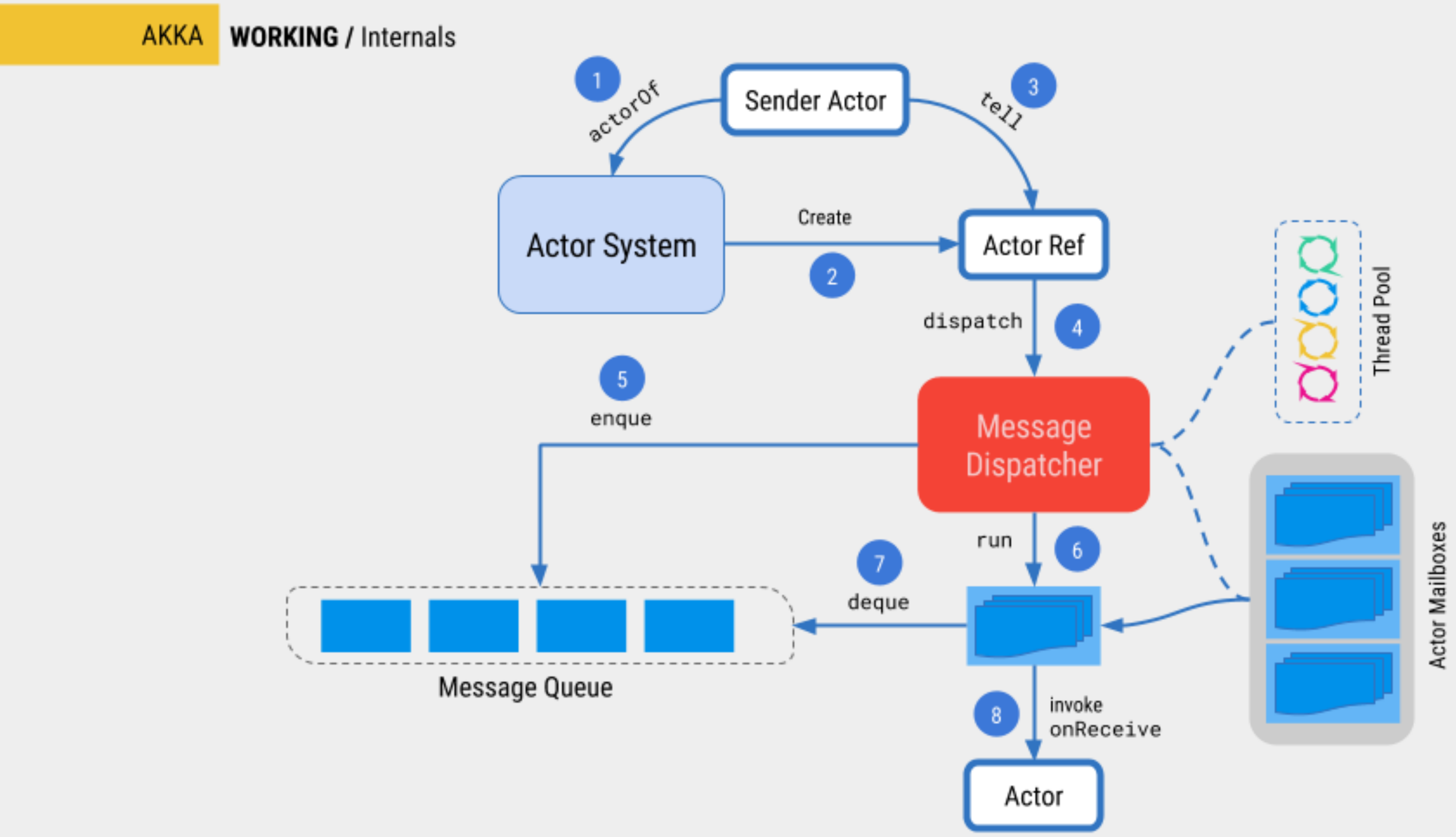


Copyright: https://dzone.com/articles/working-with-akka-actors

2. Event Driven and Reactive

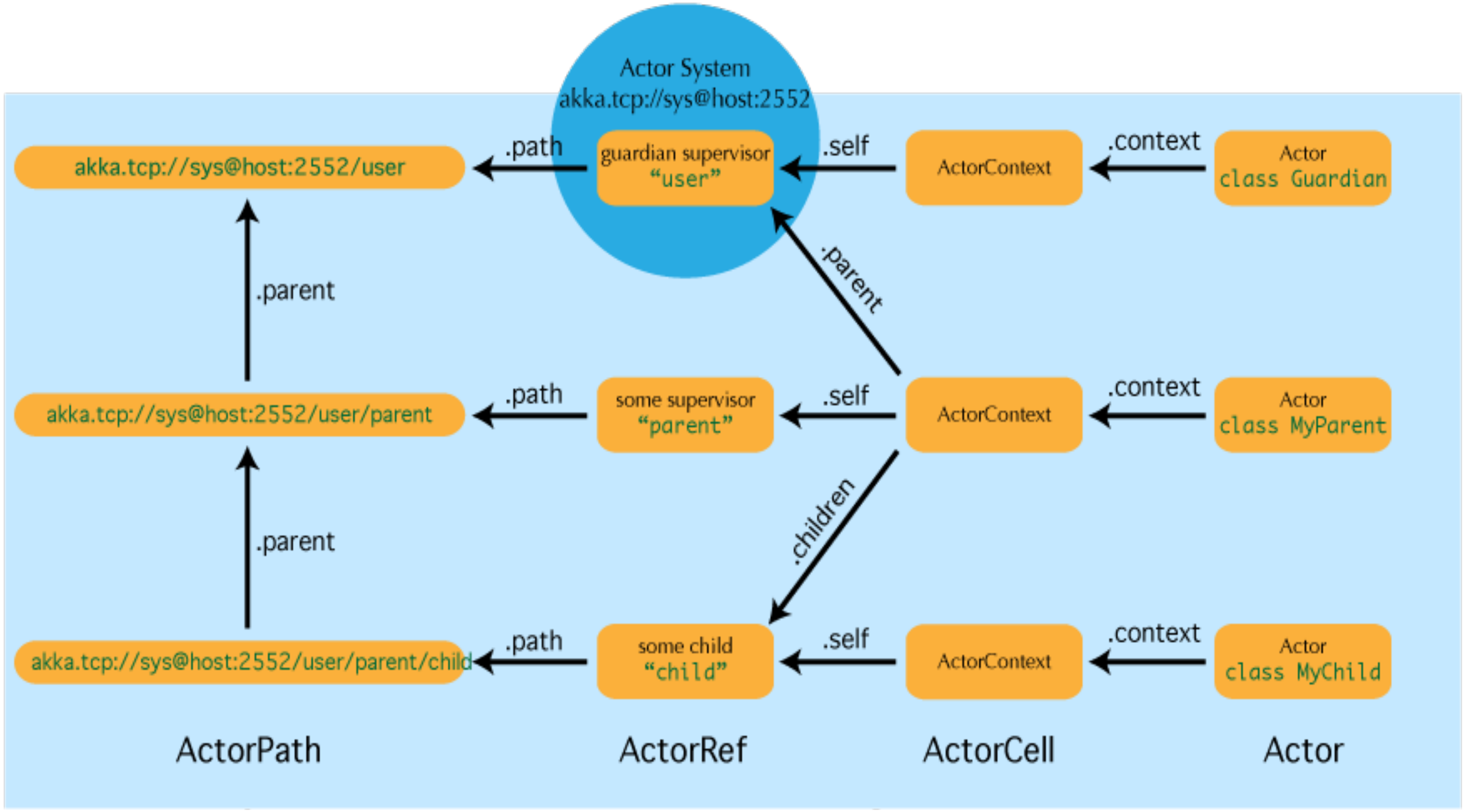


2. Event Driven and Reactive

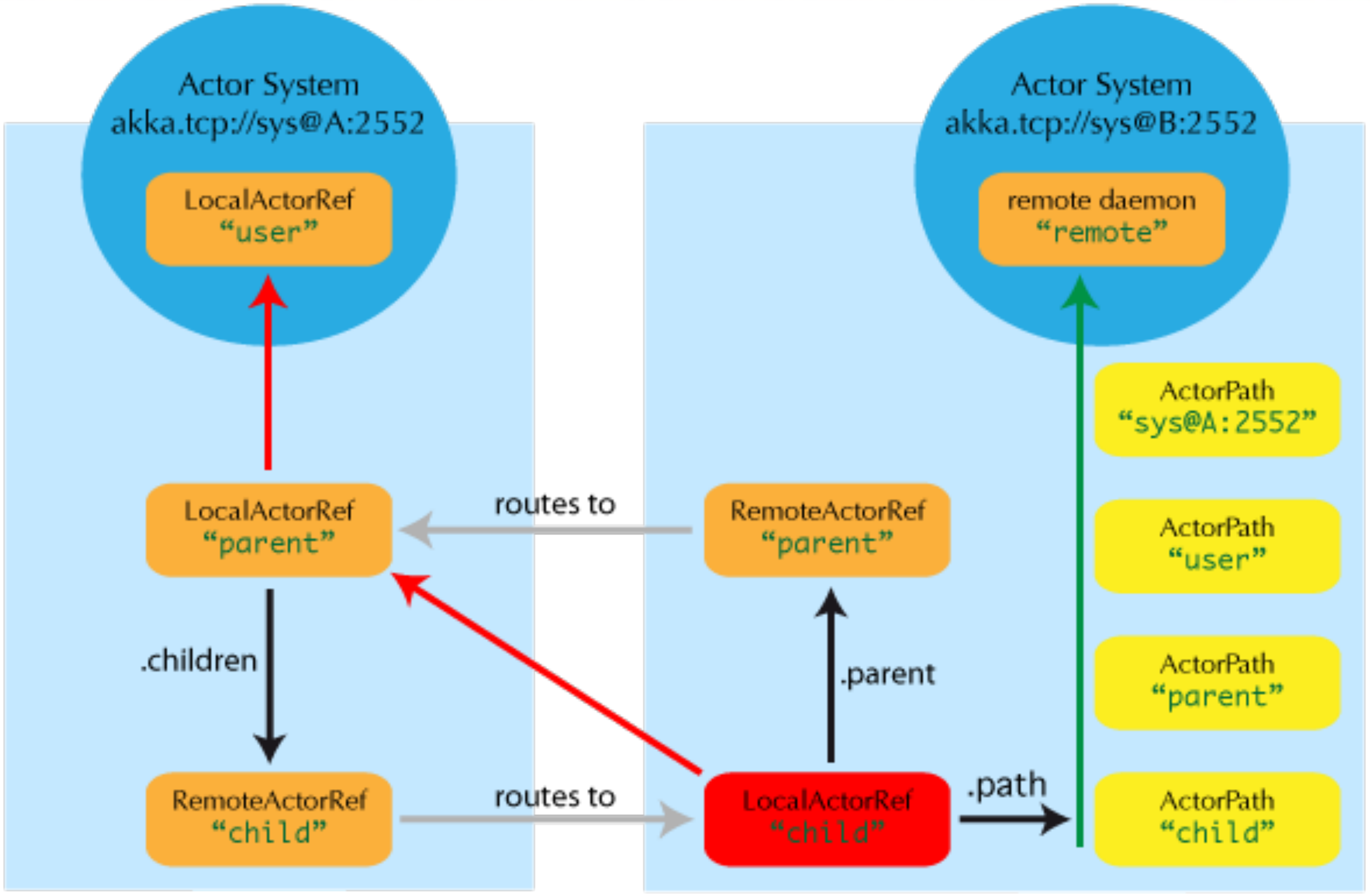


Copyright: https://deepakpol.wordpress.com/2015/09/29/event-driven-and-reactive-architecture/

2. Akka Actors Path



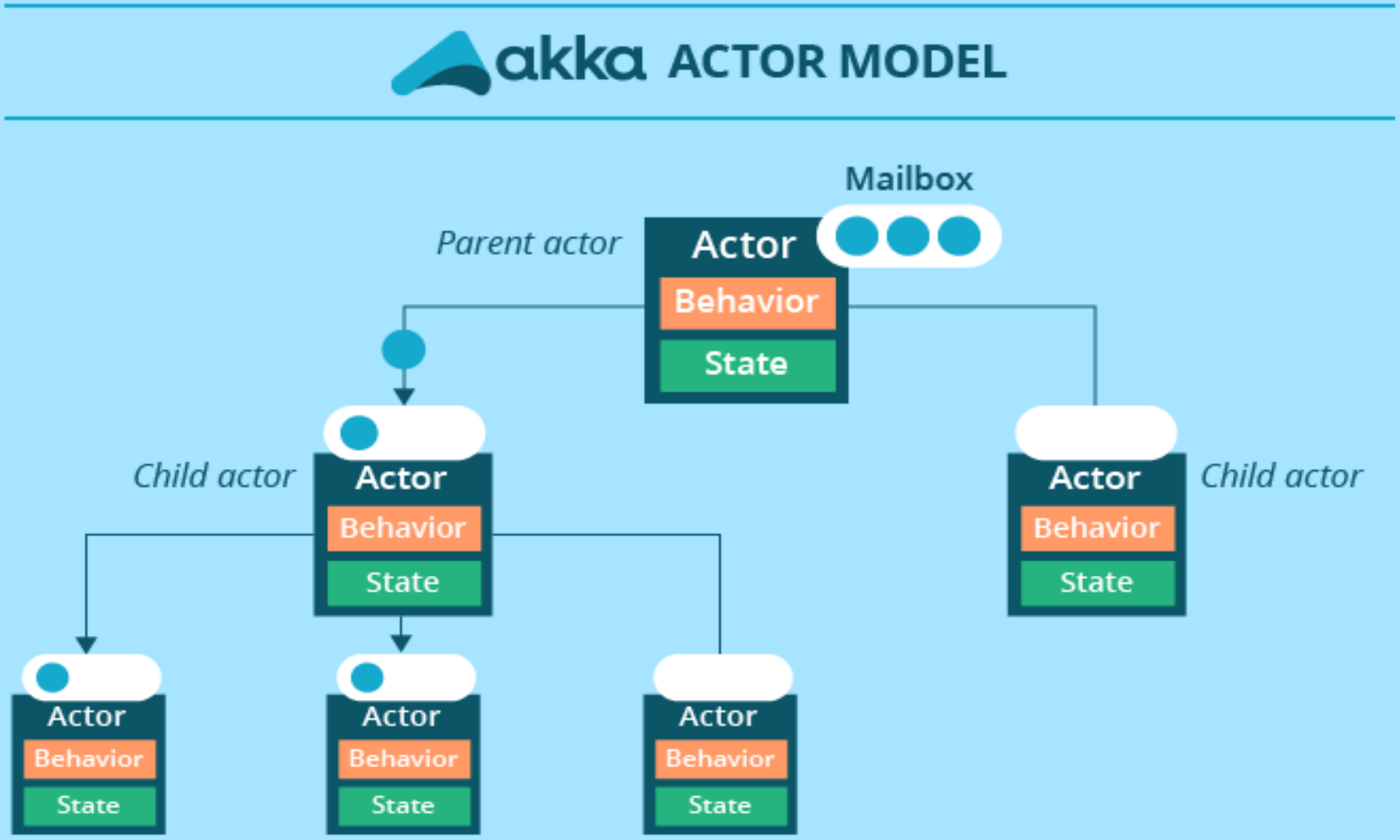
2. Akka Actors – Remote Deployment



logical actor path: akka.tcp://sys@A:2552/user/parent/child

physical actor path: akka.tcp://sys@B:2552/remote/sys@A:2552/user/parent/child

2. Akka Actors



Copyright: https://www.scnsoft.com/blog/akka-actors-for-microservices

Section Conclusions

EJB – Enterprise Java Beans vs. Akka Actors Systems

EJB and Akka DEMO

for easy sharing

3

Distributed Application Development

Communicate & Exchange Ideas

Questions & Answers!

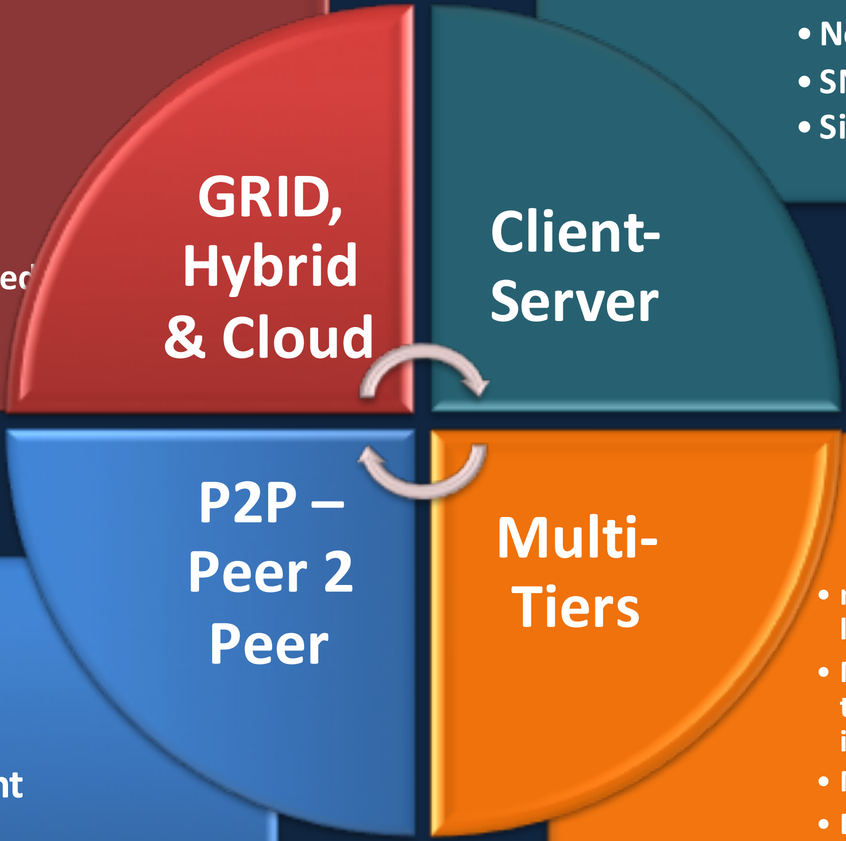
But wait…

There's More!

Distributed Architectures & Protocols Overview

- Cloud IaaS, CaaS, PaaS, SaaS
- Globus Toolkit
- Condor
- OGSA/OGSI/gLite
- ZeroC ICE GRID
- BOINC
- Hybrid: Map-Reduce distributed programming model: Apache Hadoop

- X Window Server
- Network Printers
- SNMP Req/Resp – NO Traps
- Simple WEB App



- Samples: Bitcoin/Ethereum, Torrent
- Old JXTA

- n-Tiers: Presentation / Logic / Data – linear topology
- MVC: Different than 3-Tiers in topology (triangular) – V is directly influenced by M / Spring, Struts, JSF
- MVP: Java Swing, ASP .NET MVP
- ESB – Enterprise Service Bus
- Actors Systems: Akka

Distributed Communications Protocols & Platforms

TCP/UDP-IP Sockets & Protocols Stack	MP-TCP	RPC/RMI	Distributed Components: CORBA/DCOM g-RPC	SOA-Web Services: SOAP vs. REST-JSON	Message Queues Systems: JMS / ActiveMQ	EJB / .NET Spring	P2P Protocols: Bitcoin vs. eth. Deprecated JXTA	AI Agents Systems Protocols: JADE	Actors Systems Protocols: AKKA

Thanks!

DAD – Distributed Application Development
End of Section 4
End of Lecture 12