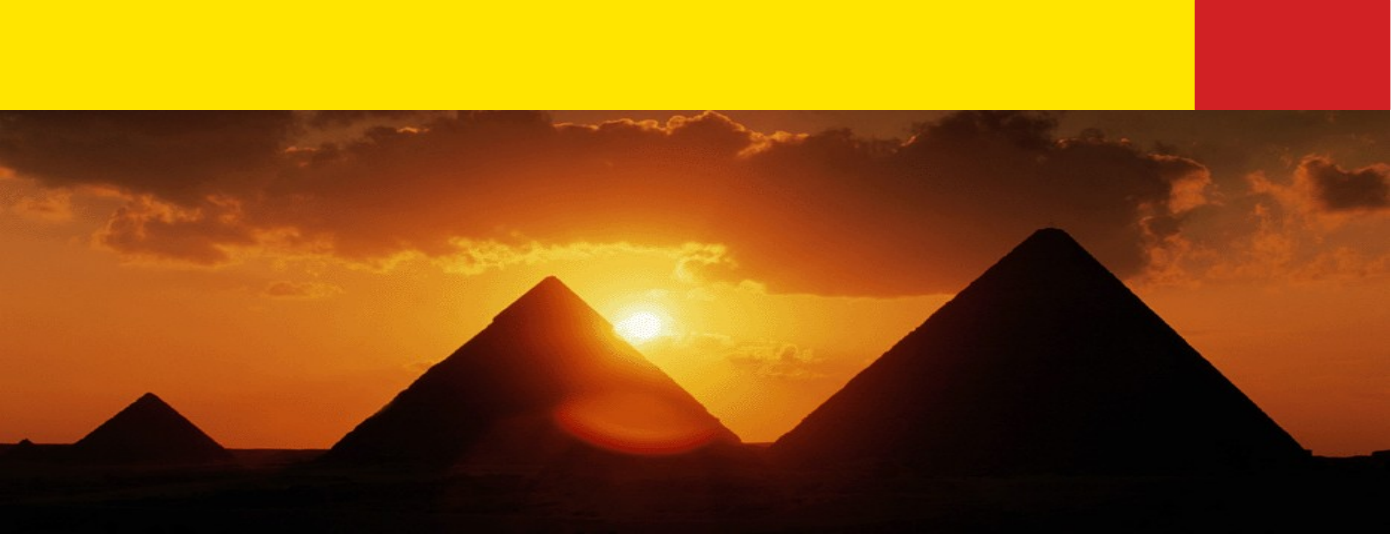
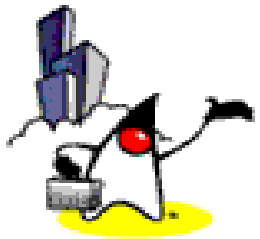
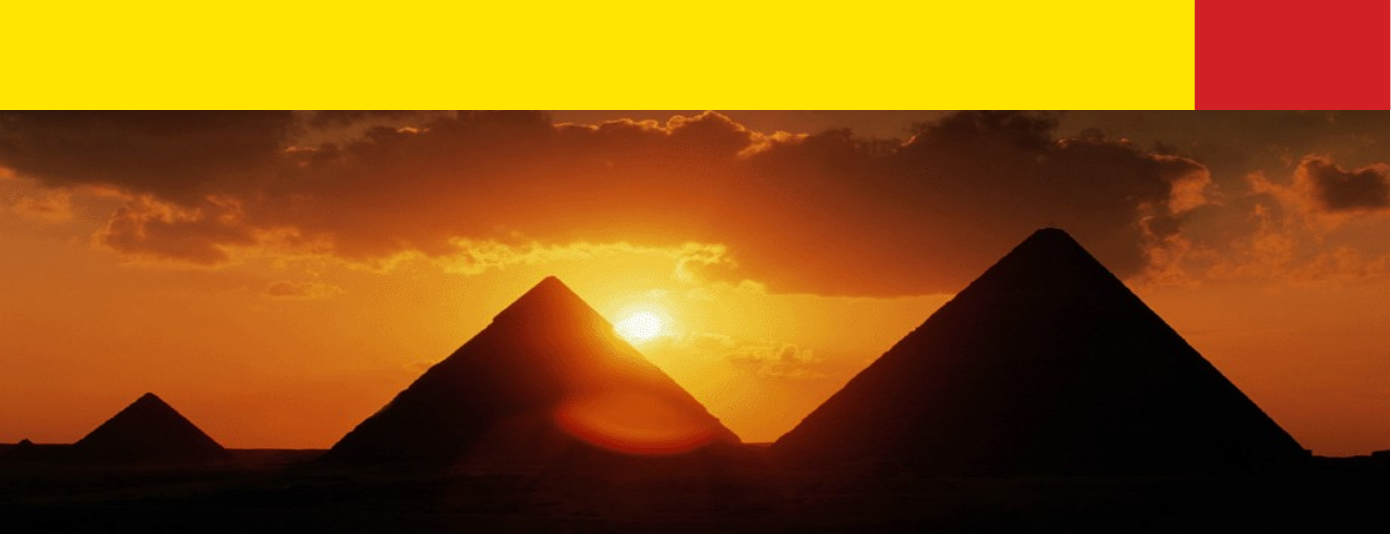


EJB Overview

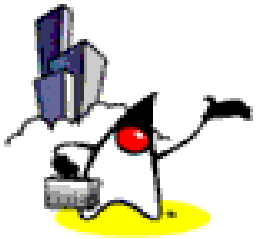


Agenda

- What is and Why EJB?
- EJB Architecture
- Component and Container Architecture
- Types of Beans
  - Session beans, Entity beans, Message-Driven Beans
- Roles
- Anatomy of a EJB module
- How client invokes methods of EJB
- RMI communication model
- Deployment Descriptor & Packaging



What is EJB?



What is EJB? (From EJB Spec.)

- “The Enterprise JavaBeans architecture is a component architecture for the development and deployment of component-based distributed business applications."
- “Applications written using the Enterprise JavaBeans architecture are scalable, transactional, and multi-user secure."
- “These applications may be written once, then then deployed on any server platform that supports the Enterprise JavaBeans specification."

What is EJB Technology?

- Cornerstone of J2EE
- A server-side component technology
- Easy development and deployment of Java technology-based application that are:
  - Transactional, distributed, multi-tier, portable, scalable, secure, …

EJB Design Principles

- EJB applications are loosely coupled
- EJB behavior is specified by interfaces
- EJB applications do not manage resources
- The container supports the application developer
- EJB applications are tiered
- The session tier is the API to the application
- The entity tier is the API to the data sources

EJB applications are Loosely Coupled

- Support the integration of components from different vendors
- EJBs refer to other components and services to which they have access by arbitrary names
- EJB can be authored without a detailed knowledge of the environment
- Applications can be assembled from separate components

EJB behavior is specified by Java interfaces

- An EJB's interaction with its clients is specified entirely in terms of Java interfaces
  - Interfaces expose the methods that clients can call, and thereby set out a 'contract' between the client and the EJB
  - Implementation is hidden from the client
  - Supports portability and modularity

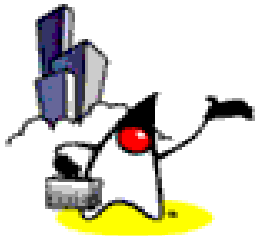
EJB applications do not manage resources

- EJBs get access to external resources (databases, legacy systems) through their container
  - Programmer does not have to worry about resource allocation and de-allocation
- It is the container's job to manage these resources, and make the access as efficient as possible
  - Container is configured by system admin not through programming APIs

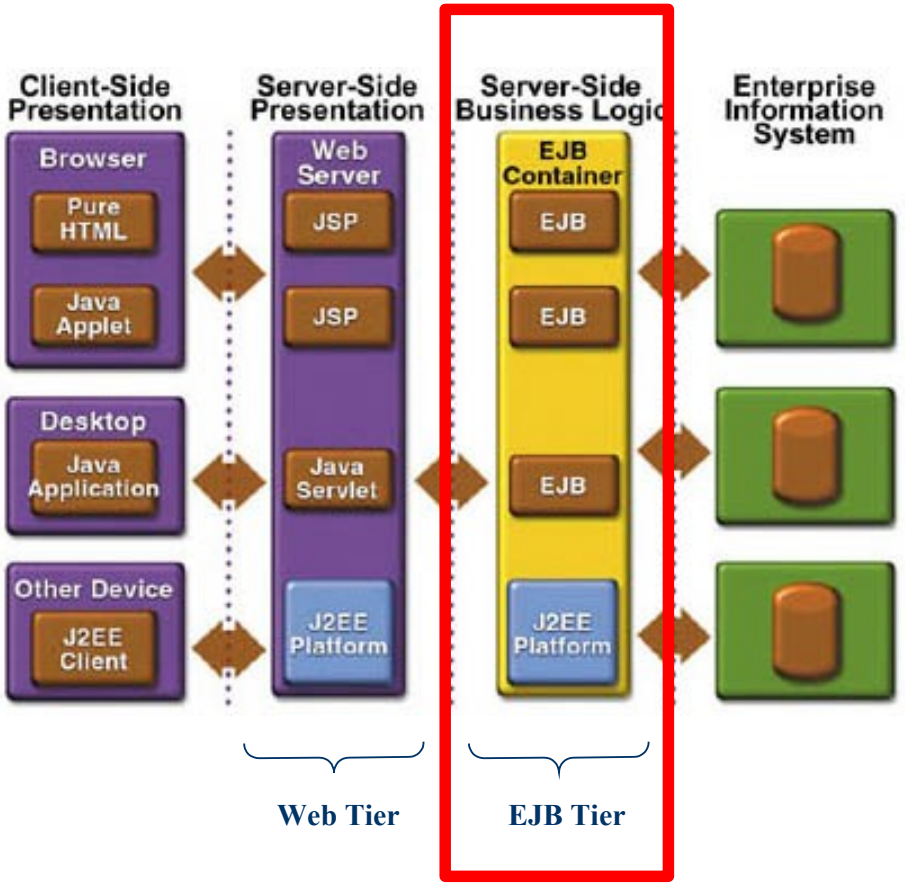
Container supports application developer

- Container provides system services
  - Persistence
  - Security
  - Transaction
  - Connection pooling
  - Component lifecycle management
  - Threading
- Application developer and deployer specifies his/her requirements in declarative fashion (in deployment descriptor)

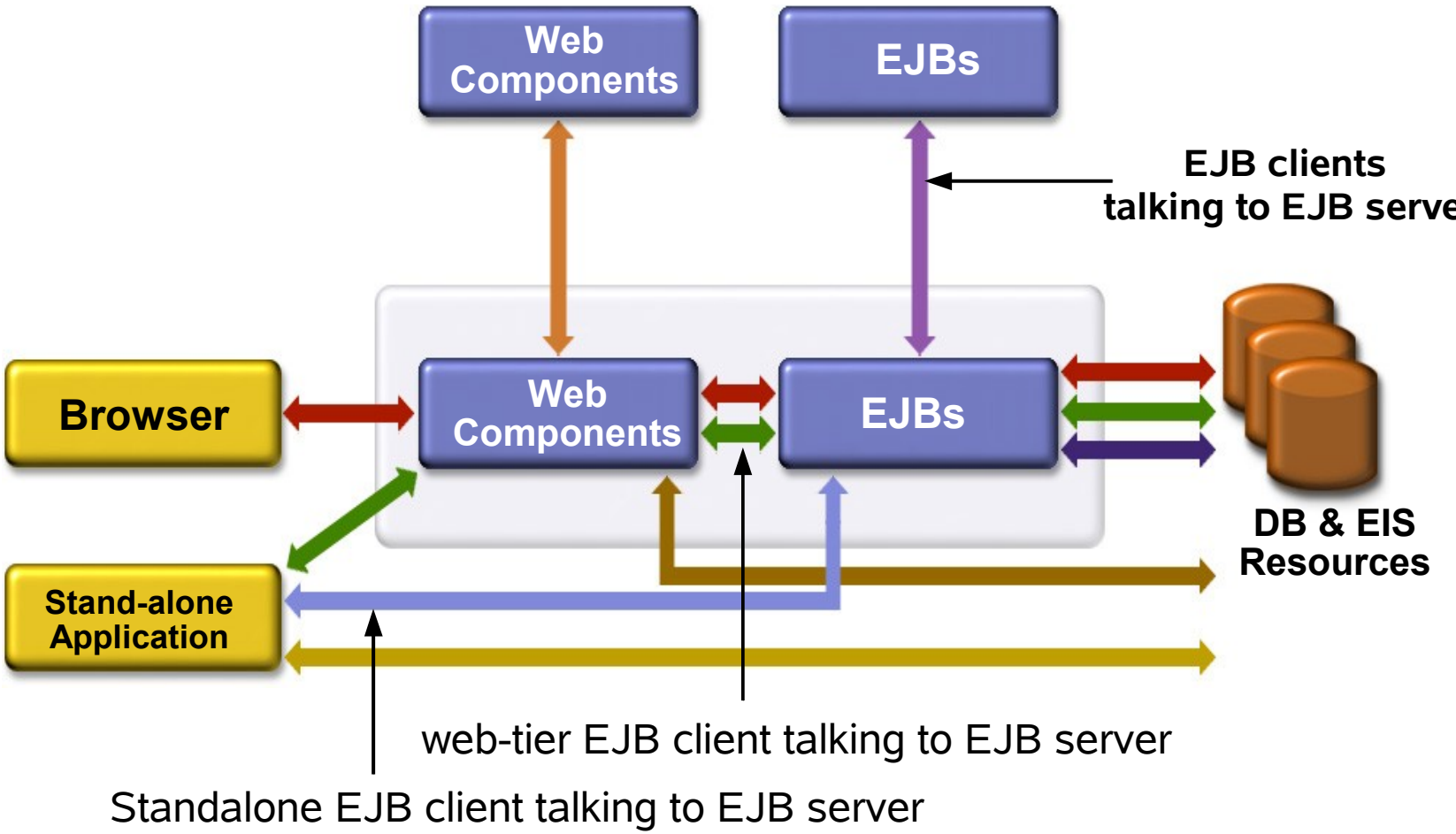
EJB in a Big Picture of J2EE

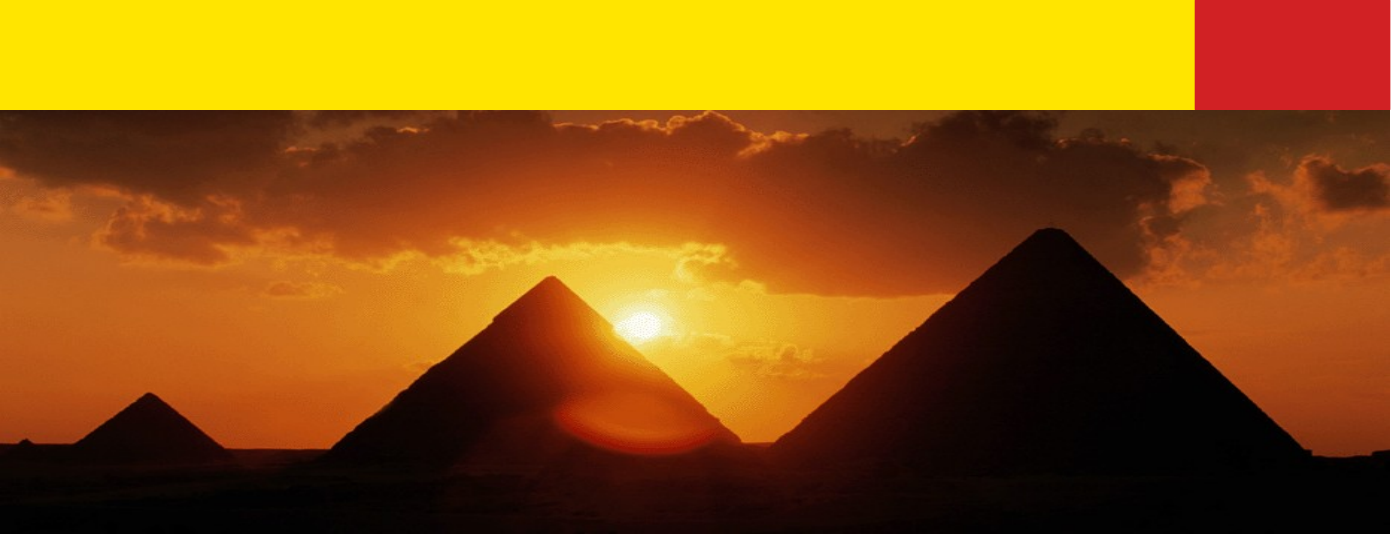


Where is EJB?

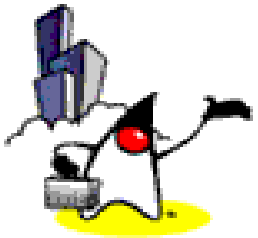

Client-Side Presentation
Server-Side Presentation
Server-Side Business Logic
Enterprise Information System
Browser
Pure HTML
Java Applet
Desktop
Java Application
Other Device
J2EE Client
Web Server
JSP
JSP
Java Servlet
J2EE Platform
EJB Container
EJB
EJB
EJB
J2EE Platform
Web Tier
EJB Tier

EJB, EJB Server, EJB Client





Why EJB?



Why EJB Technology?

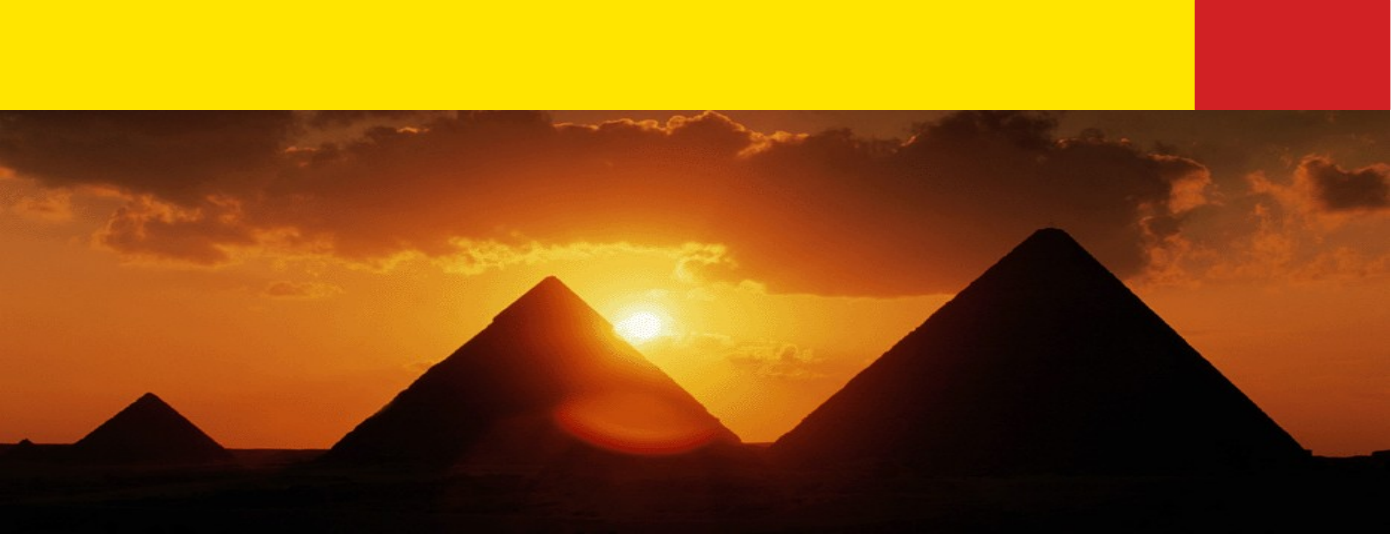
- Leverages the benefits of component-model on the server side
- Separates business logic from system code
- Provides framework for portable components
  - Over different J2EE-compliant servers
  - Over different operational environments
- Enables deployment-time configuration
  - Deployment descriptor

Enterprise JavaBeans

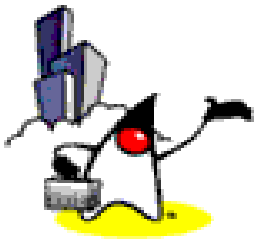
- Defined as business logic ONLY
- No low-level plumbing
- Reusable across multiple EJB Servers
- Implement interfaces that allow container to manage them

Do You Need an EJB Tier?

- Yes, if you want to leverage middleware features provided by container
  - Resource management, instance life-cycle management, concurrency control and threading
  - Persistence, transaction and security management
  - Messaging
  - Scalability, availability
- Yes, if you want to build portable and reusable business components
- Maybe not, for a simple application whose main function is reading database tables



Controversies of EJB 2.x



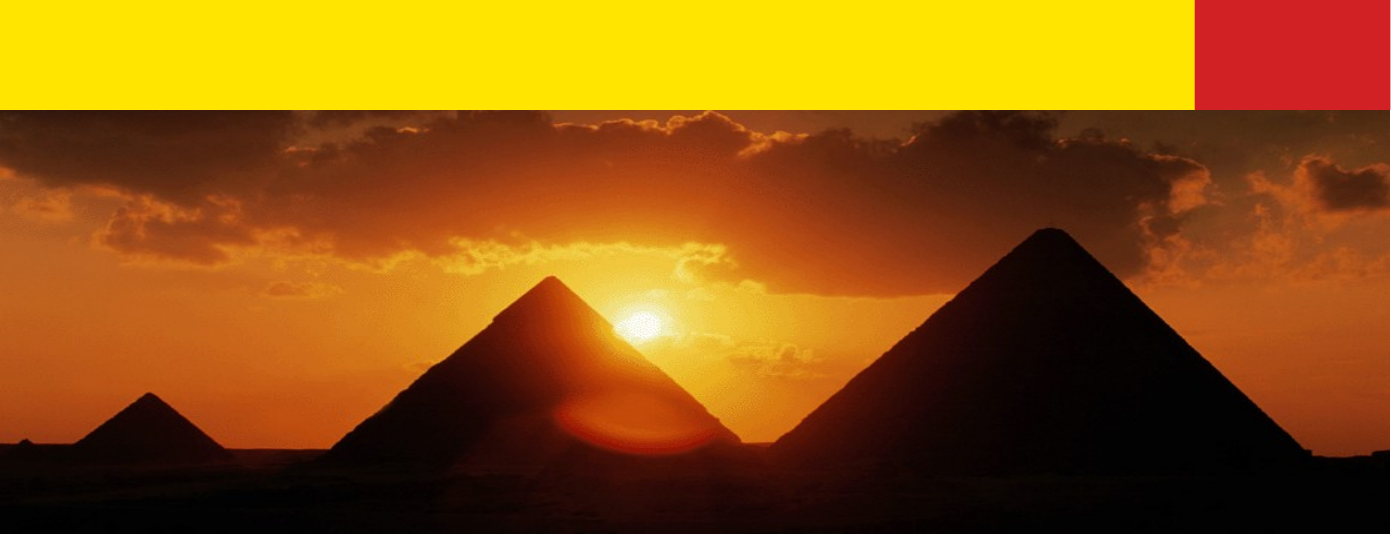
Controversies of EJB 2.x

- It is too heavy-weight
- Its programming model is complex
- It does not follow OO programming model
- It does not let you test the code easily

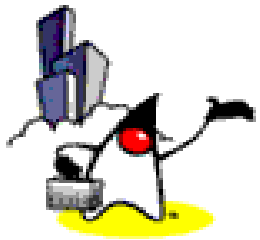
Generally, people agree that EJB 2.x is useful only for distributed transactional applications.

EJB 3.0

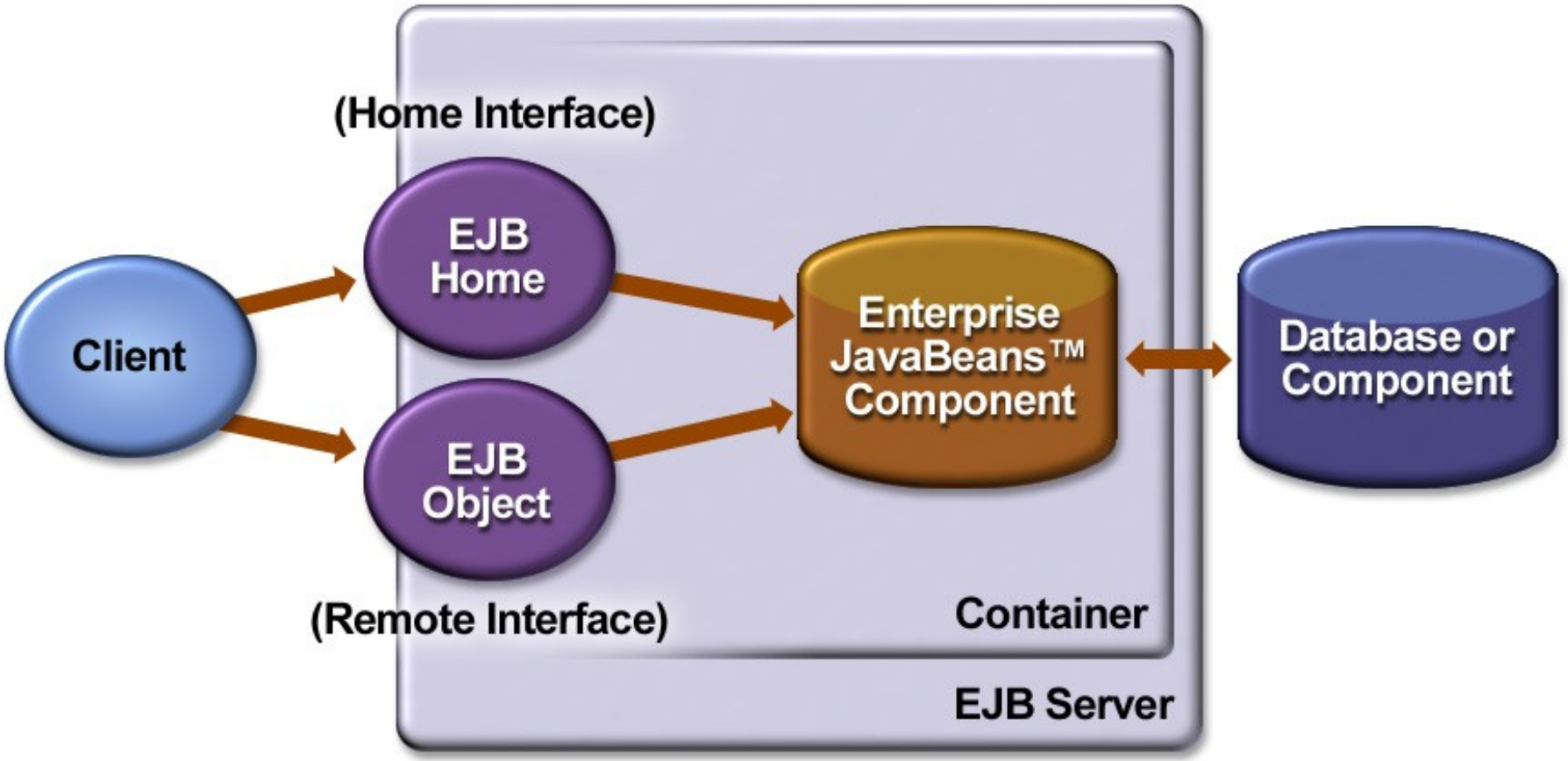
- Good news is that EJB 3.0 addresses all of the issues mentioned in the previous slide



EJB Architecture



EJB Architecture



EJB Architecture Contracts

- Contracts are specified in EJB specification
- Client view contract
  - Contract between client and container
- Component contract
  - Contract between an Enterprise Bean and its Container

Client View Contract

- Client of an EJB can be
  - Web tier components: Servlet and JSP
  - Standalone Java application
  - Applet
  - Another EJB in same or different container
  - Web services client (in EJB 2.1)
- Provides development model for clients using EJB services

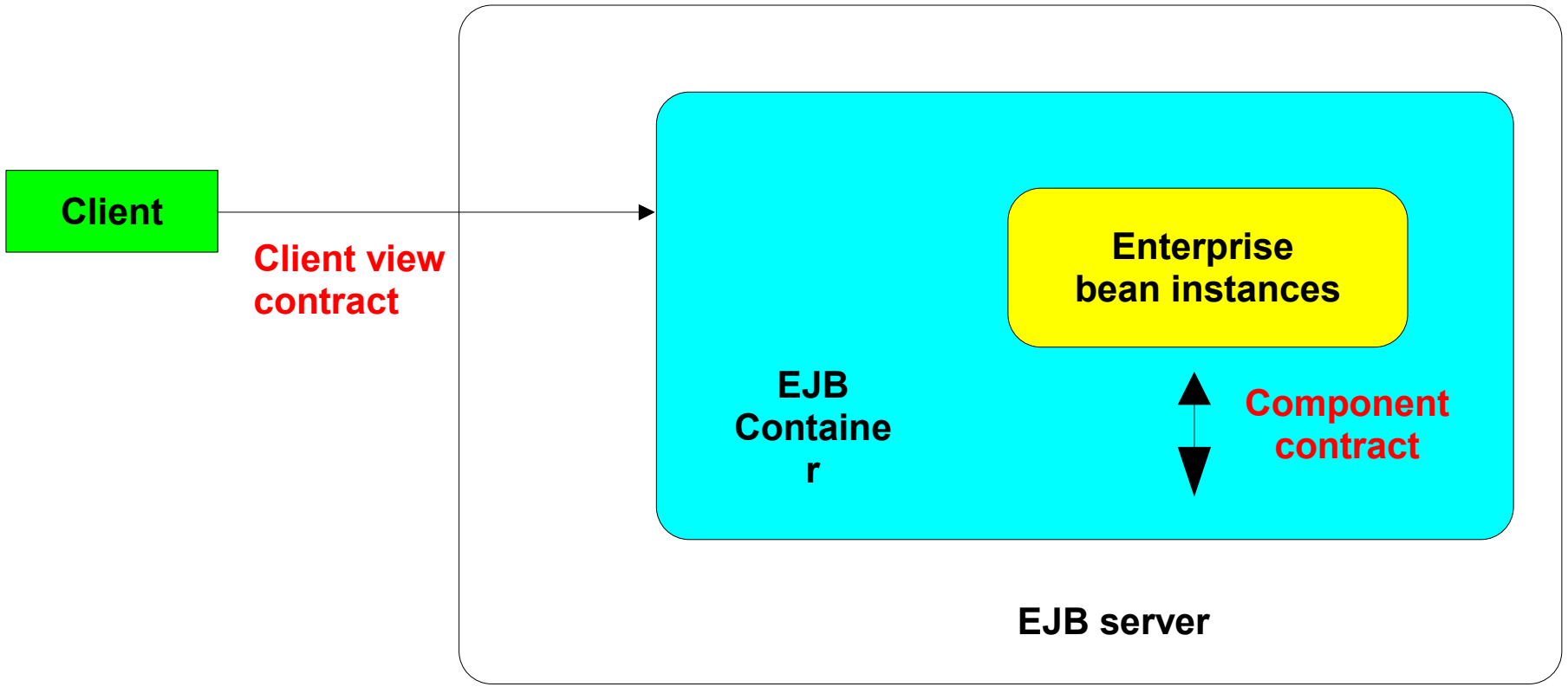
Client View Contract (Contd.)

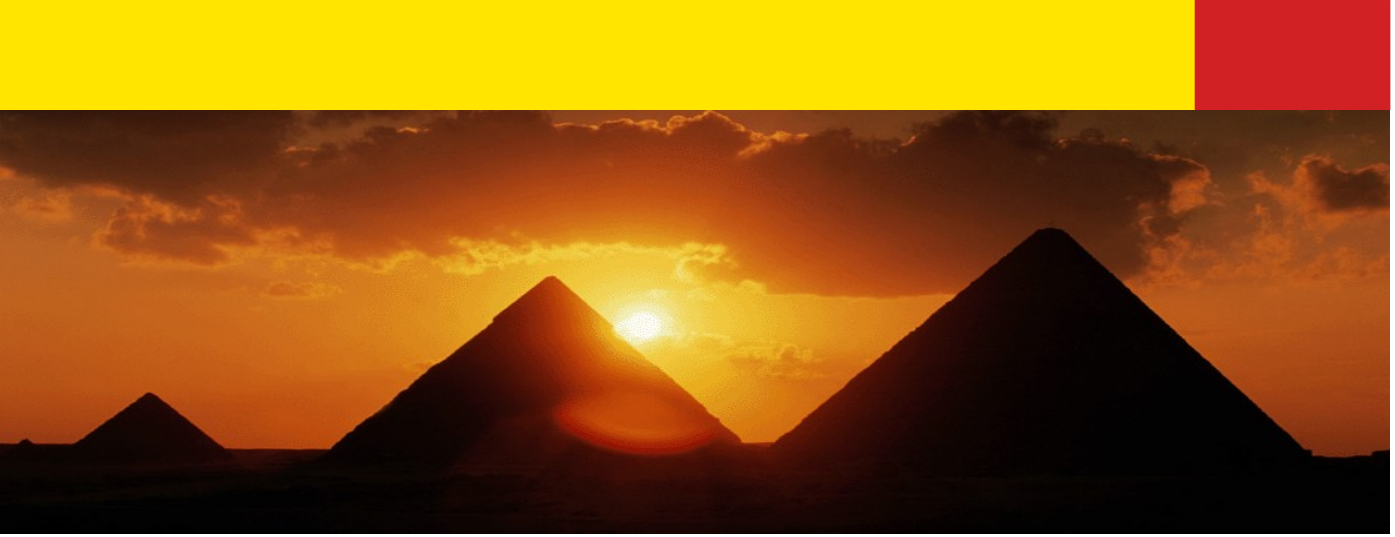
- Client view contract is comprised of
  - Home interface
    - For local or remote clients
    - Contains methods for creating and locating beans
  - Logic Interrface (Is called Remote interface)
    - For local or remote clients
    - Contains business methods
  - Object identity
  - Metadata interface
  - Handle

Component Contract: What Container does (for Beans)

- Enables EJB method invocations from clients
- Manage the life cycle of EJB bean instances
- Implements home and remote interfaces
- Provide persistence for CMP entity beans
- Provide runtime context information to beans
- Manage transactions, security, exceptions, etc...
- Implements callbacks

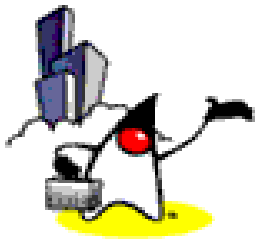
EJB Contracts





Fundamentals of EJB Architecture

(source: Applied Enterprise JavaBeans written by Kevin Boone)



Fundamentals of EJB Architecture

- Client view of an EJB is defined strictly by interfaces
- EJBs are isolated and supported by an EJB container
- EJB container provides an illusion of a single threaded environment
- EJB container manages database transactions
- EJB container manages access and security

Fundamentals of EJB Architecture

- Creating and locating EJBs is standardized
- Instance can be pooled for efficiency
- Container manages resources
- There is a standard deployment process

Client view of an EJB is defined strictly by Interfaces

- Clients can call only those methods exposed by EJB's interfaces
  - Factory interface (home interface)
  - Business method interface (remote interface)
- These interfaces are collectively referred to as “client view”
- An interface is a simply specification of method signatures
  - You bean is not an implementation of the interface

EJBs are isolated and supported by an EJB container

- EJB method calls from client are intercepted by EJB container before they are delegated to EJB beans
  - Proxy objects (Home object and EJB object) which are generated by container
- Container encapsulates EJB beans and acts as its security manager
- Container provides system services to EJB beans

EJB container provides an illusion of a single threaded environment

- Bean developer does not have to worry about concurrency control
- Bean developer write bean as if it is used in a single threaded environment

EJB container manages database transactions

- Container handles both local and distributed transactions
- Container supports declarative transactions

EJB container manages access and security

- Container handles access control
  - Which methods are accessible to which roles
  - Access control is declaratively specified in deployment descriptor
  - Programmatic access control is allowed
- Container also provide authentication scheme
  - Bean provider should never have to code authentication procedures

Creating and locating EJBs is standardized

- There is a well-defined way for the client to create new EJBs, or to find existing ones
  - Client uses JNDI to get a proxy object (actually a reference to stub of a EJB Home object)
  - Client then calls either create() or find() method of the Home object to get another proxy object (actually a reference to stub of a EJB object)
  - Clients always deal with proxy objects - never directly with EJB bean instance

Instances can be pooled for efficiency by Container

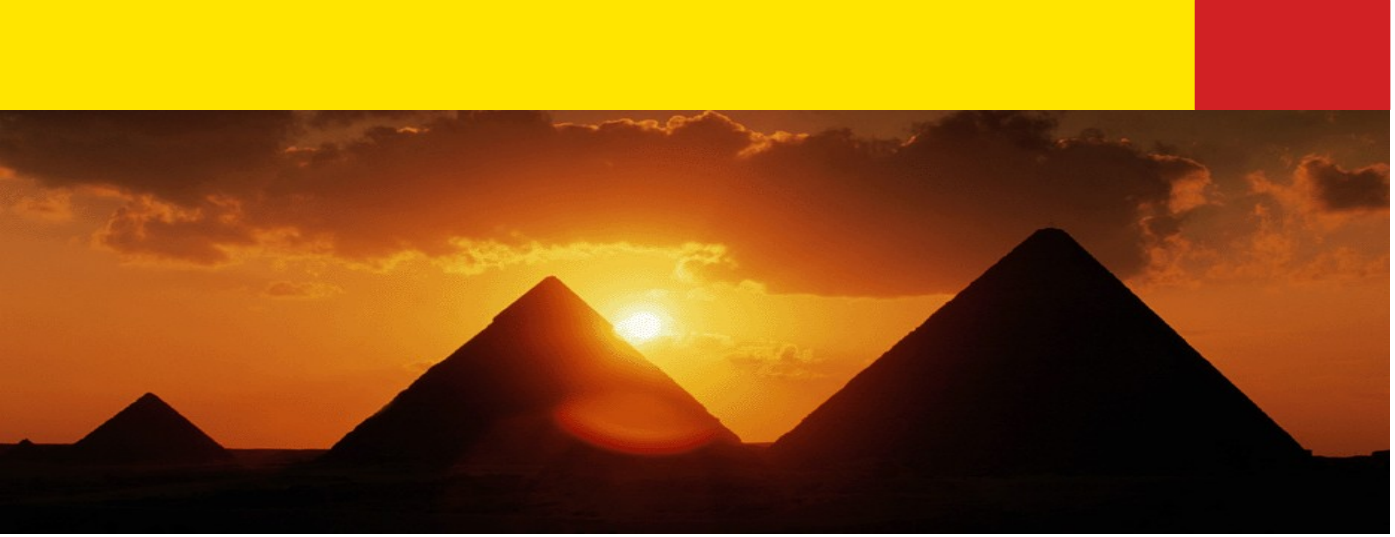
- Container may pool bean instances
  - Container knows when is the good time to create and remove bean instances
- When a client asks container to create a bean via create() method, the container is likely to return a bean instance from the pool
- This is all transparent to client

Container manages external resources

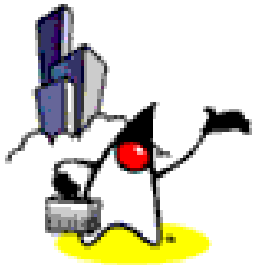
- External resources include
  - Databases
  - Enterprise information systems
  - Messaging systems
- External resources are shared among EJBs
- Container handles pooling these resources

There is a Standard Deployment Process

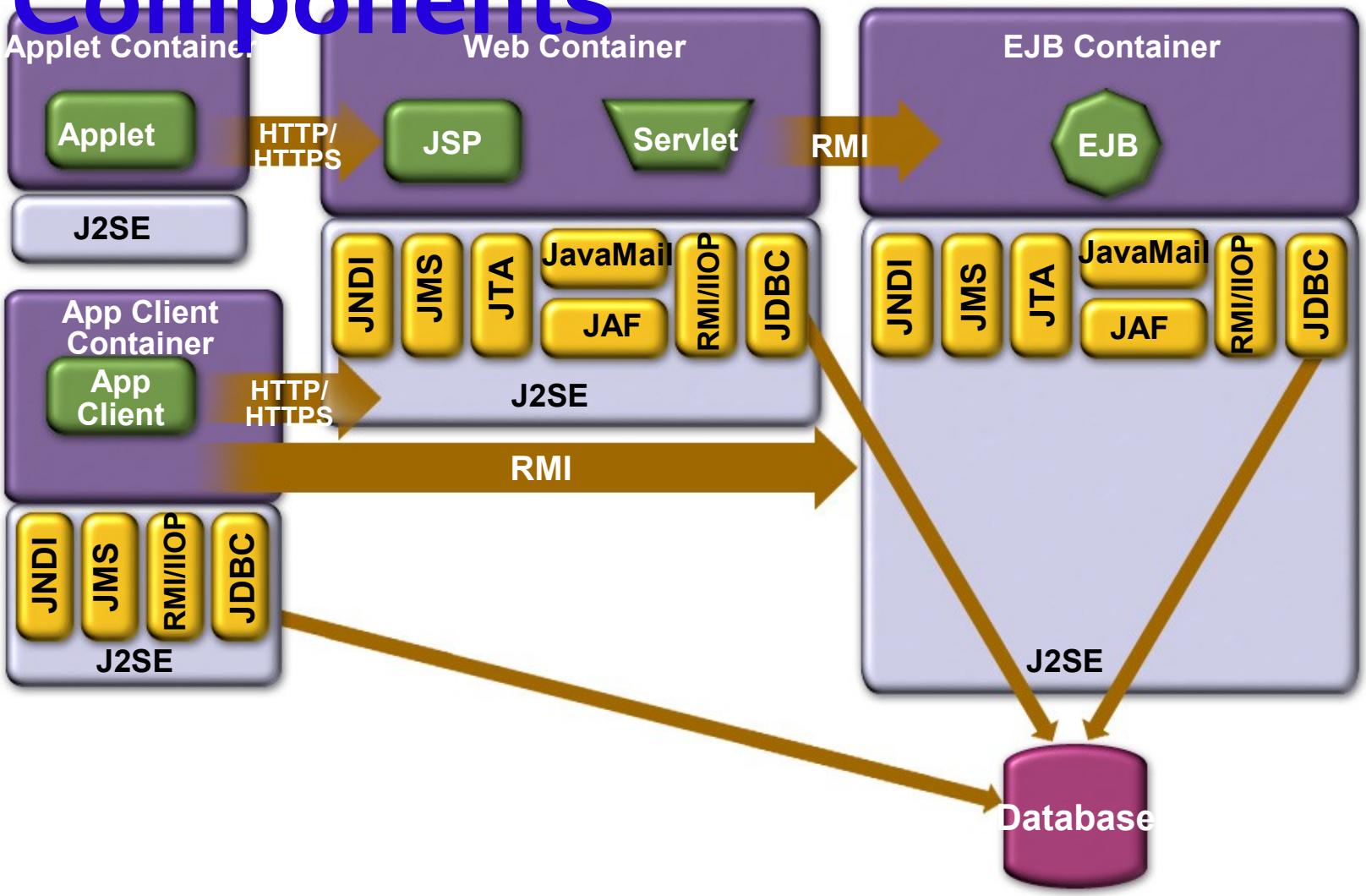
- EJB specification standardize
  - Packaging EJB application
  - Deployment descriptor
- Any EJB compliant platform should be able to deploy any EJB compliant application



J2EE Component & Container Architecture



J2EE Containers & Components



Containers and Components

Containers Handle

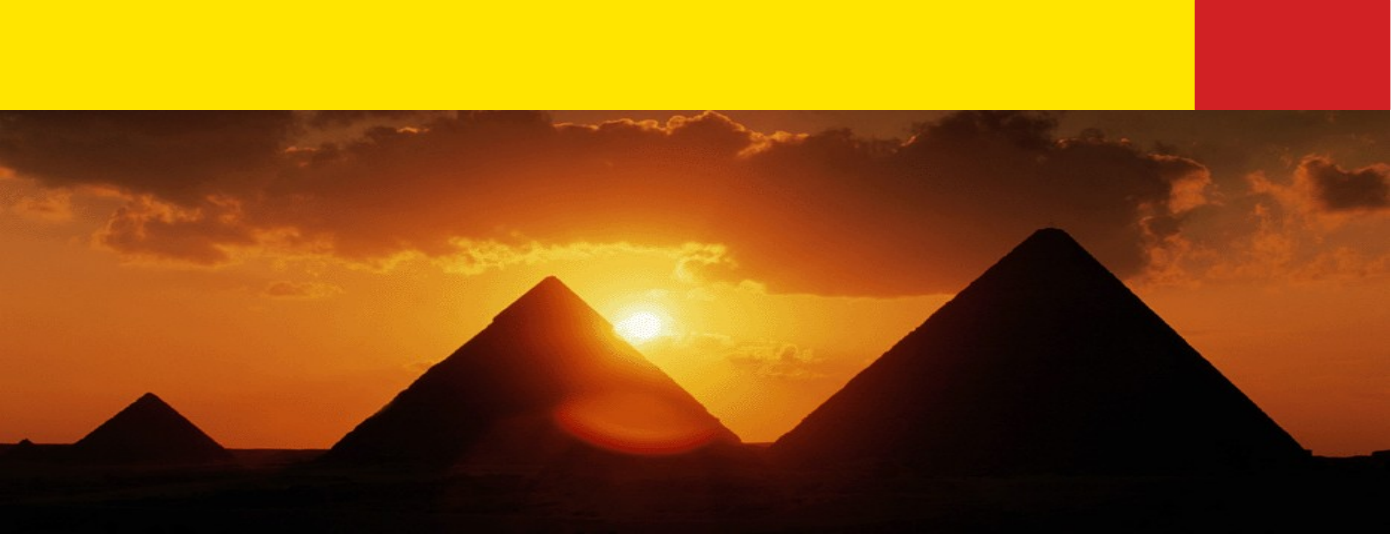
- Concurrency
- Security
- Availability
- Scalability
- Persistence
- Transaction
- Lifecycle management
- Management

Components Handle

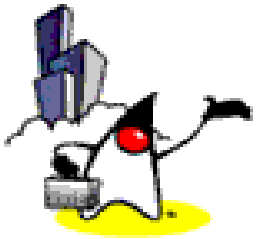
- Presentation
- Business Logic

What the EJB Container does

- Generates concrete class for
  - Remote (or Local) home Interface
  - Remote (or Local) Logic Interrface
- Binds home object to Naming service
  - Clients can lookup the home object using JNDI
- Creates “free beans” pool.
- Caches recently accessed beans
- Provides JDBC Connection pooling



Terminology



Terminology Confusion

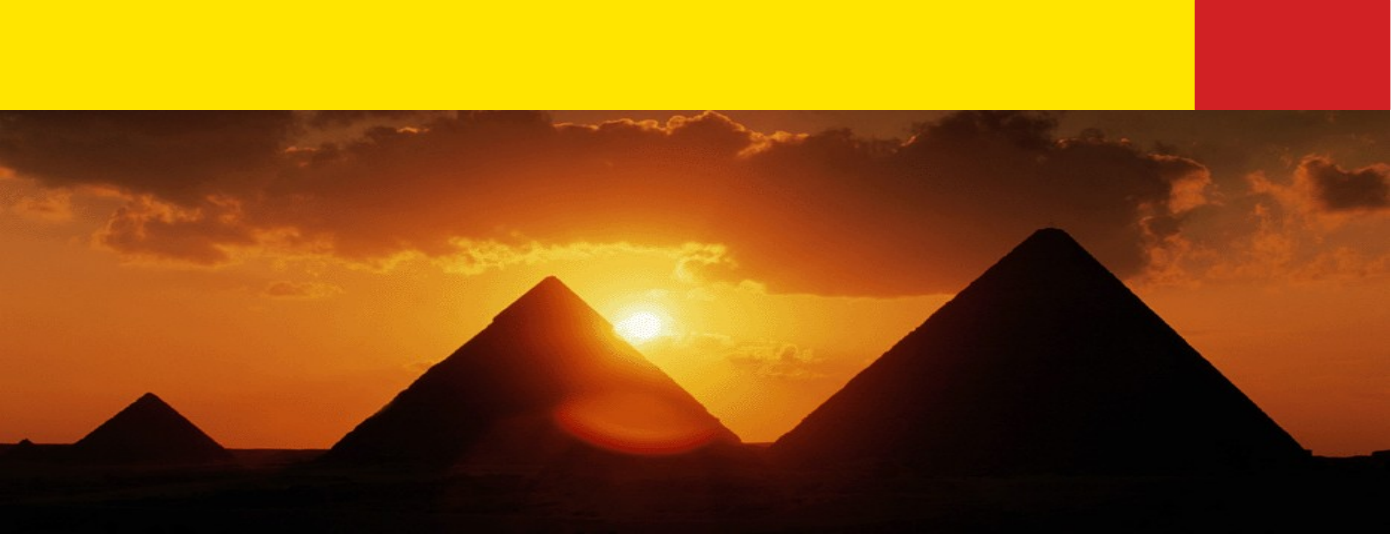
- The term “EJB” is used in many different context
  - Technology, Architecture, Specification, Implementation, Product, Server, Container
  - Bean class
  - Bean instance
  - EJB module
  - EJB application
  - EJB proxy objects

We will use these terms

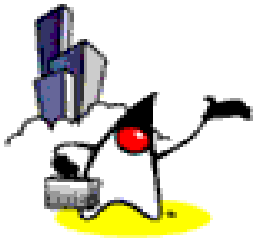
- Bean class (or Implementation class)
  - Java class that represents a bean
- Bean instance
  - Actual bean object instance within a EJB container
- EJB module (or EJB jar file)
  - ejb-jar file
  - Collection of bean classes
- EJB application (or J2EE application)
  - *.ear file

We will use these terms

- Home interface
  - Java interface that contains creation/location methods
- EJB Home object (implements Home interface)
  - Sometimes called factory object
- Logic Interrface (called Remote interface)
  - Java inteface that contains business methods
- EJB object (implements Logic Interrface)
  - Implemented by the container
- EJB Home object and EJB object are



Types of Beans



Types of Beans

- Session Beans
  - Stateful session beans
  - Stateless session beans
- Entity Beans
  - Bean Managed Persistence (BMP)
  - Container Managed Persistence (CMP)
- Message Driven Beans
  - JMS
  - JAXM

Session Beans

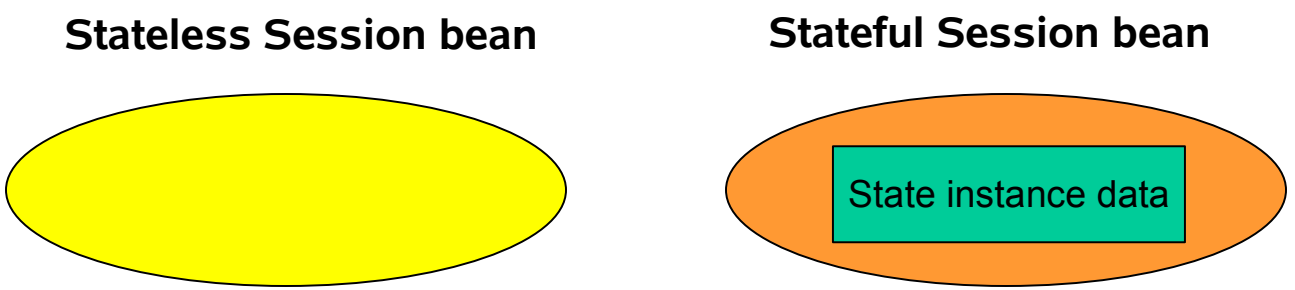
- Does work on behalf of a single client
- Is not persistent and hence relatively short lived
  - Is removed when the EJB™ server crashes
- Does not represent data in data store, although can access/update such data
- Bean class implements javax.ejb.SessionBean interface

When to Use Session Beans?

- Use Session beans to model process or control objects specific to a particular client.
- To model workflow, processes or tasks, manage activities (make reservation, purchase...).
- To Coordinate processes between entity beans, control interactions of beans
- To Move business application logic from Client to the Server Side

2 Types of Session Beans

- Stateless: execute a request and return a result without saving any client specific state information
  - transient
  - temporary piece of business logic needed by a specific client for a limited time span
- Stateful: maintains client specific state



Examples of Session Beans

- Stateless session beans
  - Catalog
    - No client specific state needs to be preserved
  - Interest calculator
    - No client specific state needs to be preserved
    - Business logic with no need for database access
- Stateful session beans

Entity Beans

- Provides object view of data in data store
  - Its lifetime not related to the duration of interaction with clients
  - Lives as long as data exists in database i.e. Long lived
  - In most cases, synchronized with relational databases
- Shared access among clients
- Bean class implements javax.ejb.EntityBean interface

Entity Beans

- Clients normally look up (find) an existing entity EJB
  - Creation means adding a row to a database table
  - Finding means finding a row in a existing database table
  - Removing means removing a row from a database table
- Entity bean instance has unique identifier called primary key
  - Primary key can be any class

Examples of Entity Beans

- Customer
  - Customer data has to persist, thus is maintained in the database
  - Customer data has to survive server crash
  - Customer data is shared by many clients
  - Each customer has unique identification such as customer number

2 Types of Entity Beans

- CMP (Container Managed Persistence)
  - Persistence is managed by Container
  - Persistence requirements are specified in deployment descriptor
  - Bean developer does not have to worry about providing persistence logic in his code
- BMP (Bean Managed Persistence)
  - Persistence logic code is provided by Bean developer

When to Use CMP vs. BMP?

- CMP entity beans
  - With CMP 2.0, there is no reason not to use CMP
  - Database independence
  - Higher performance
  - Easy to develop and deploy
- BMP entity beans
  - More programmatic control is desired

Session Beans and Entity Beans

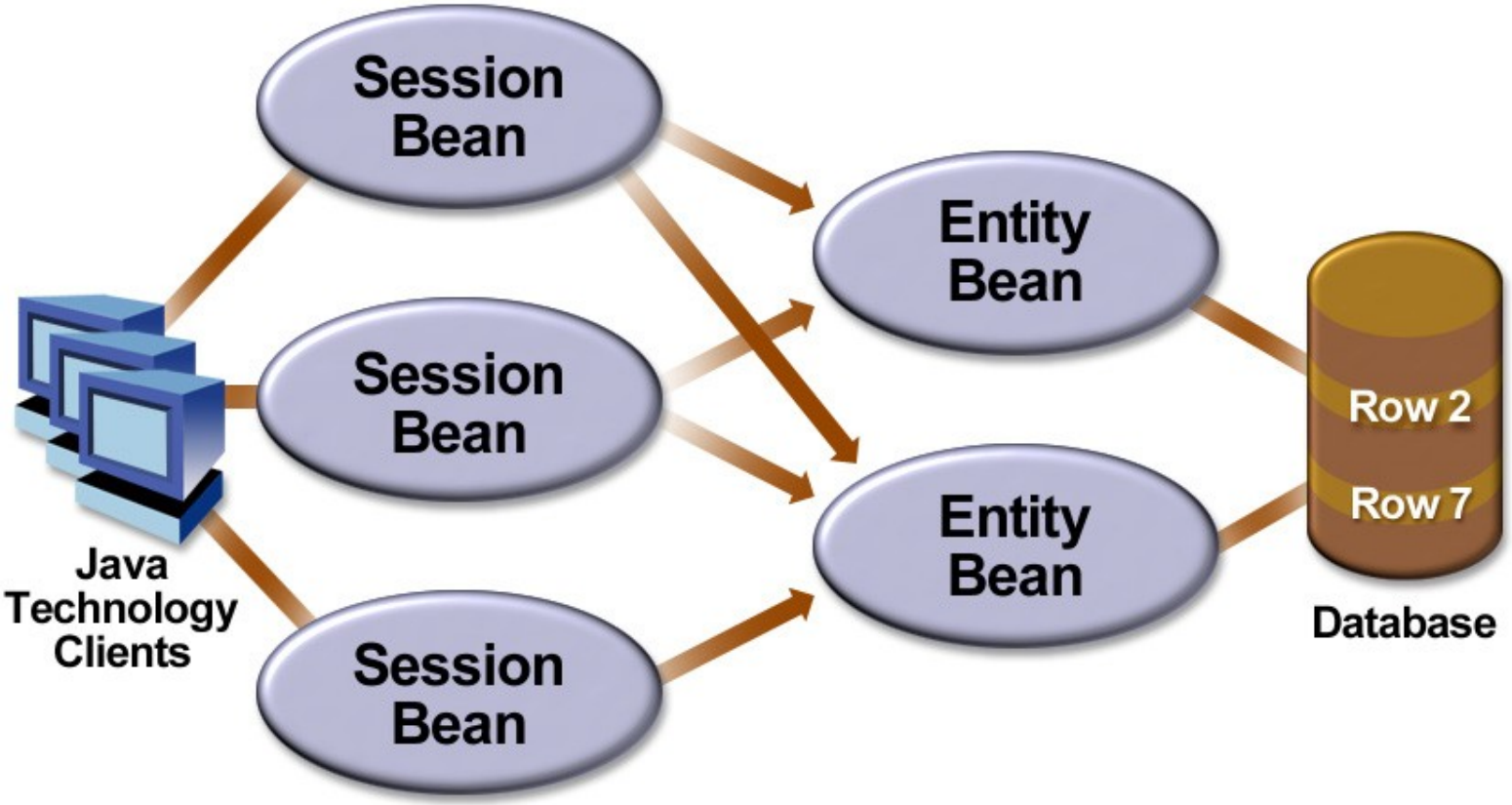
Session Beans

- Represent a business process
- One instance per client
- Short-lived: Life of client is life of bean
- Transient
- Doesn't survive server crashes
- May be transactional

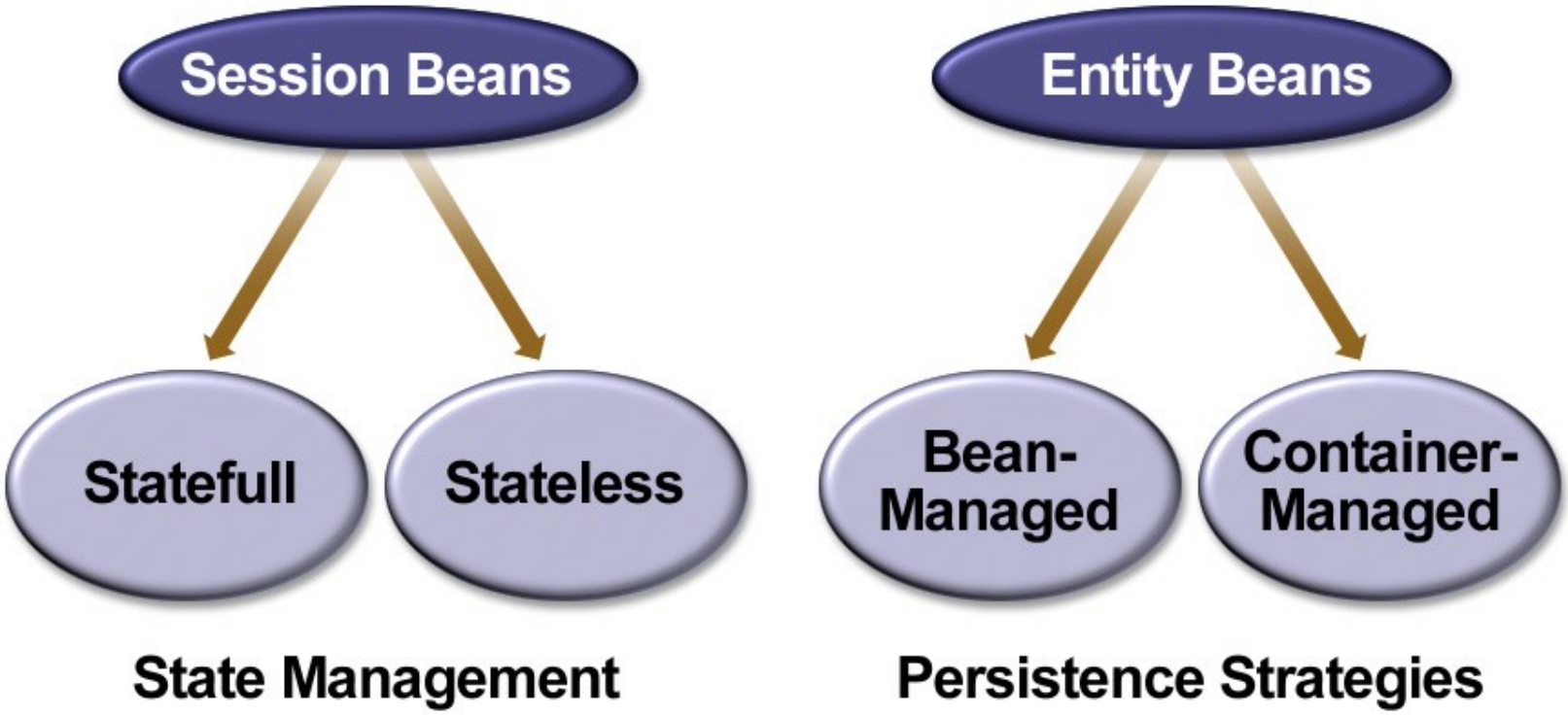
Entity Beans

- Represent business data
- Shared instance for multiple clients
- Long-lived: as long as data in database
- Persistent
- Survive server crashes
- Always transactional

Entity and Session Beans — Typical Architecture

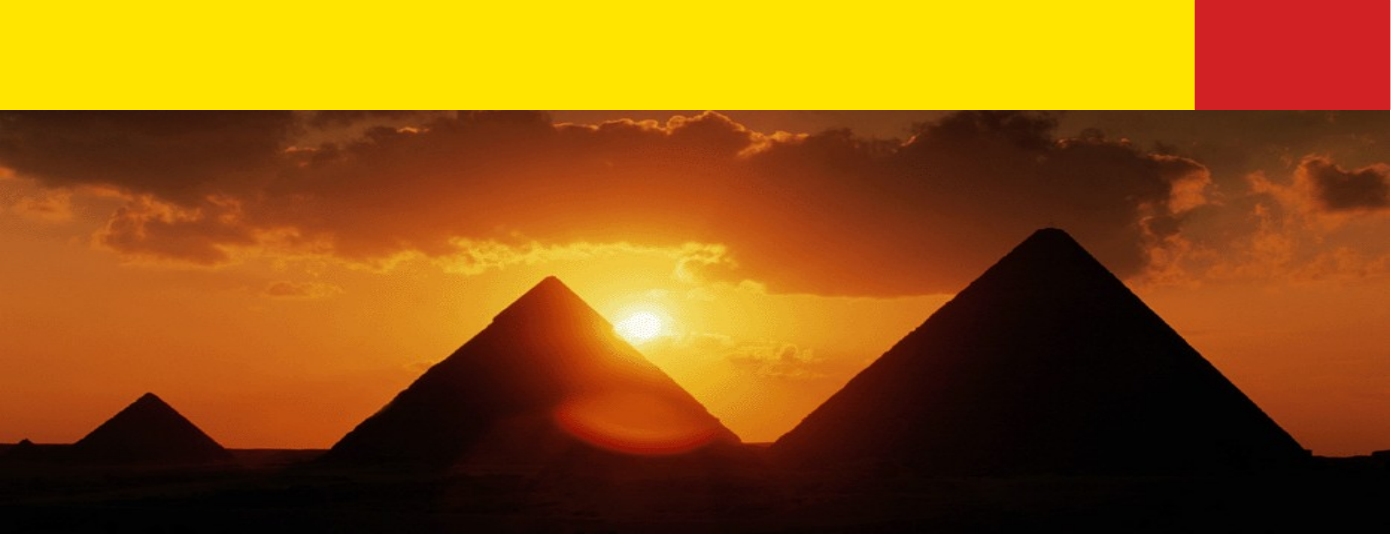


Entity and Session Beans

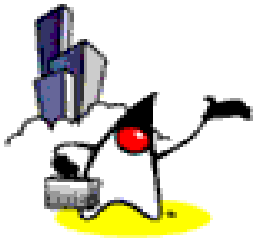


Message-Driven Beans (MDB)

- Acts as a consumer of asynchronous messages
- Cannot be called directly by clients
  - Activated upon message arrival
  - No home or remote interface
- Clients interact with MDB by sending messages to the queues or topics to which they are listening
- Stateless



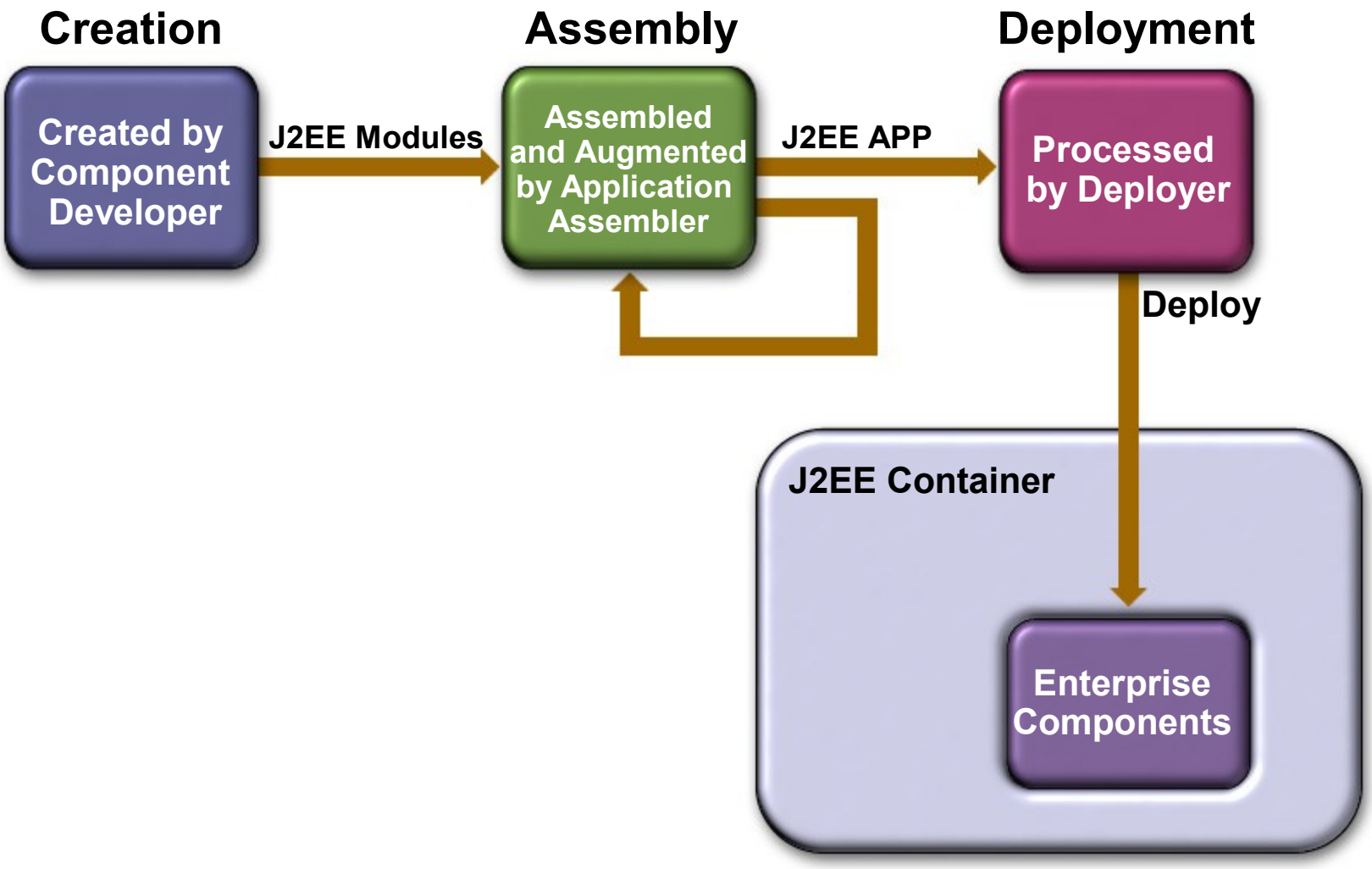
Roles



Roles

- Distinct roles in the application development and deployment life cycle
  - EJB Bean Provider (Component provider)
  - Application Assembler
  - Deployer
  - EJB Server Provider
  - Container Provider
  - System Administrator
  - Tool provider

Lifecycle Illustration

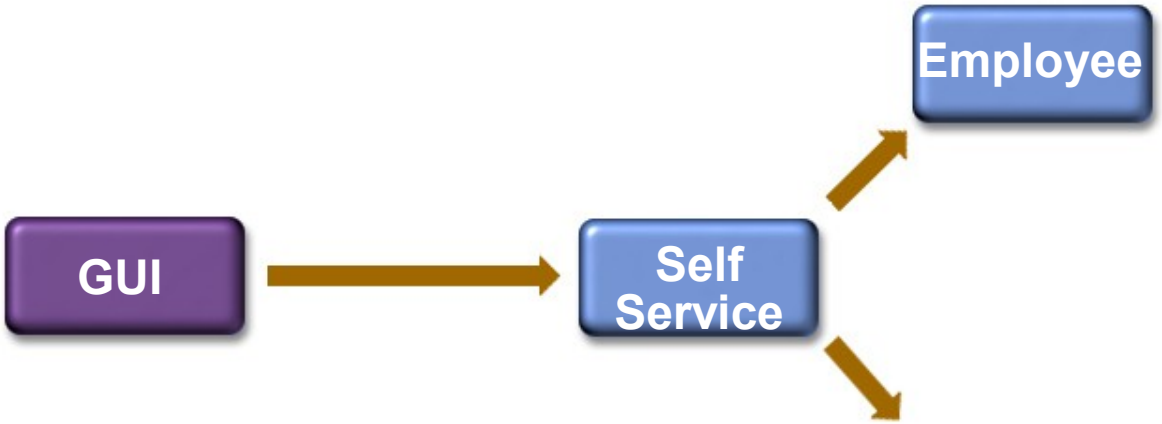


Example Scenario: Step 1

Vendor A: Bean Provider creates Payroll bean

Example Scenario: Step 2

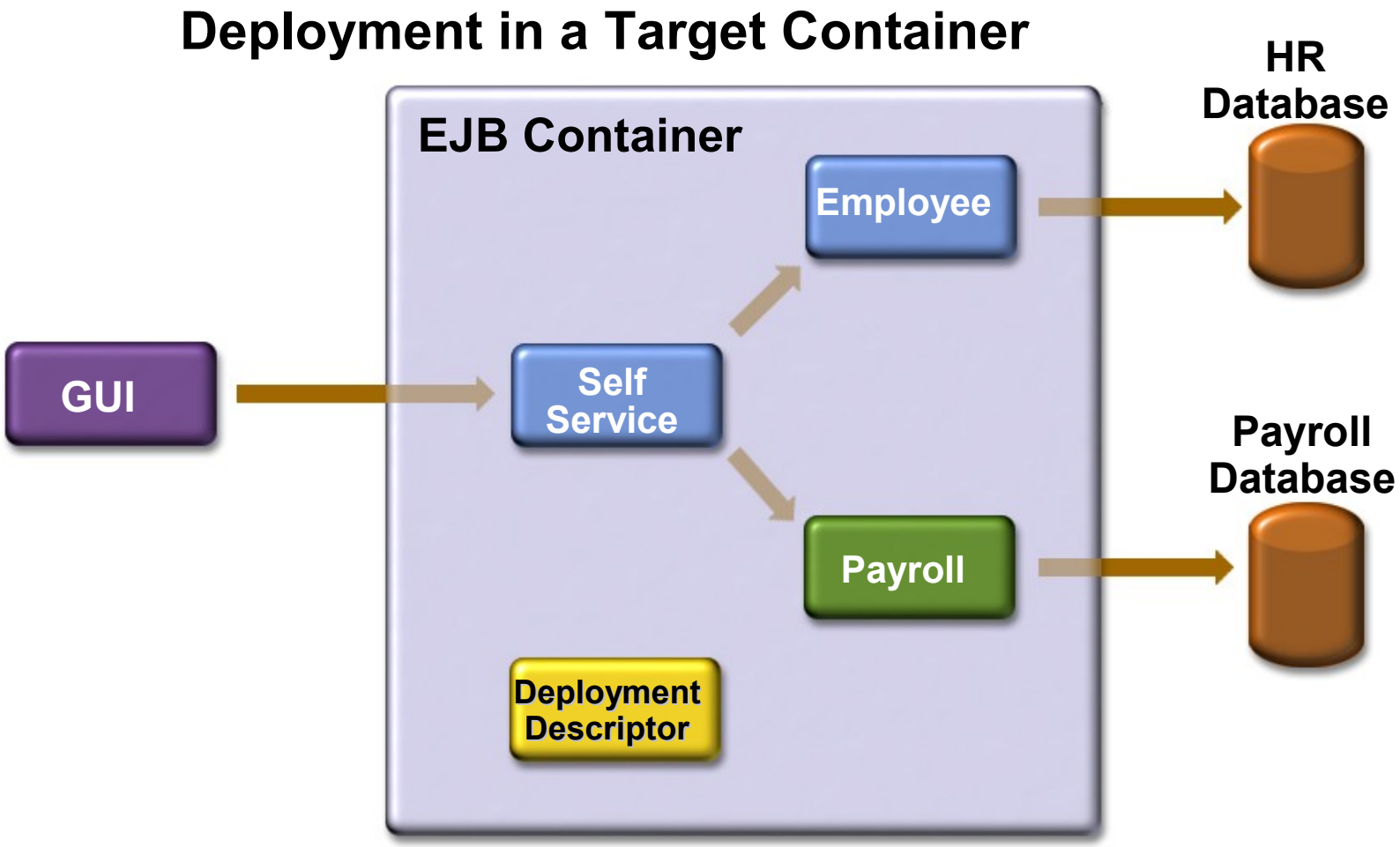
Vendor B: Bean Provider and Application Assembler

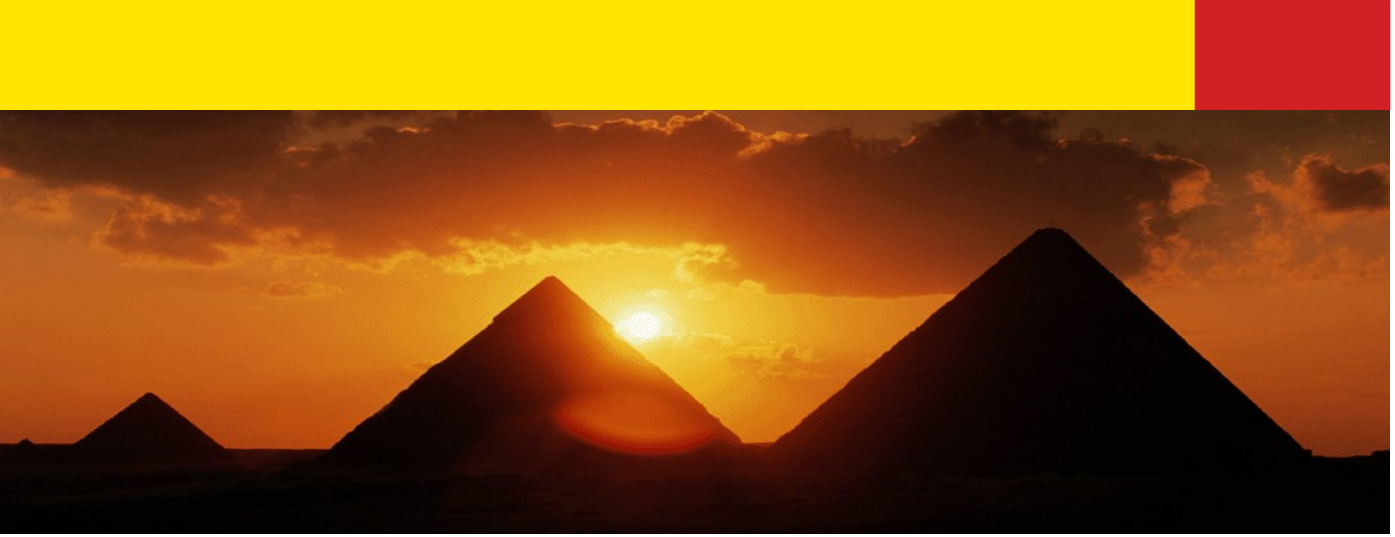


Depends on: **Payroll**

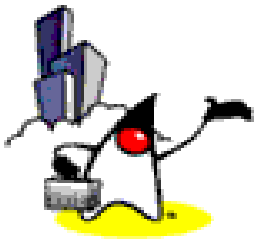
Deployment Descriptor

Example Scenario: Step 3





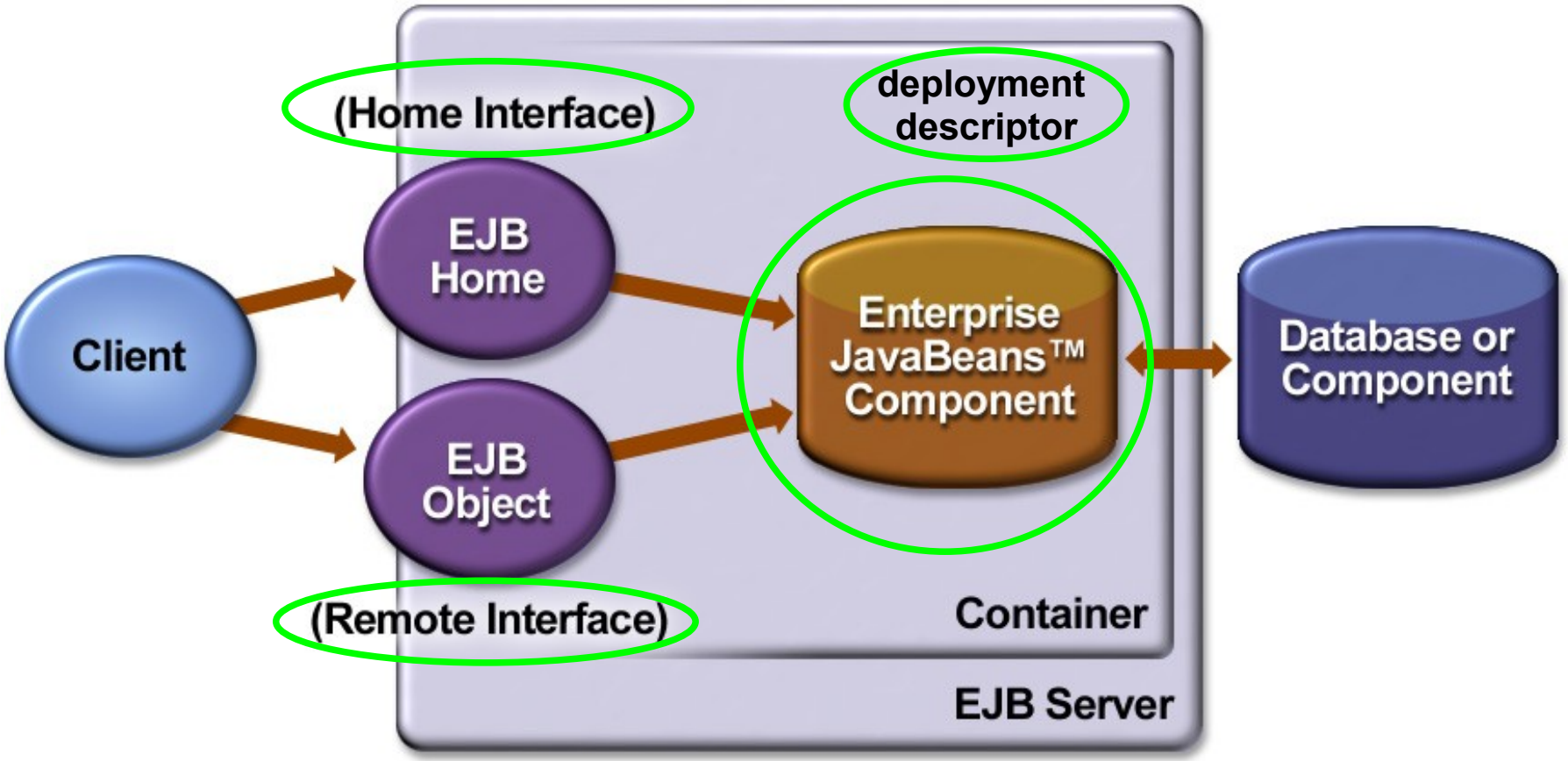
Anatomy of a EJB Module



Contents of EJB Module

- Bean developer creates EJB Modules (EJB-JAR files)
- A EJB module contains
  - Interface classes (must be present)
    - Home interface
    - Remote interface
  - Bean classes (must be present)
  - Deployment descriptor (must be present)
  - Helper classes (present only when needed by Bean class)

Anatomy of EJB Module (EJB-JAR file)



Home Interface

- Defines methods for creating, finding and removing beans
  - Factory interface
- Implemented by Container
  - EJB home object
- Client gets “reference to stub object of the EJB home object” via JNDI
- Can be remote or local

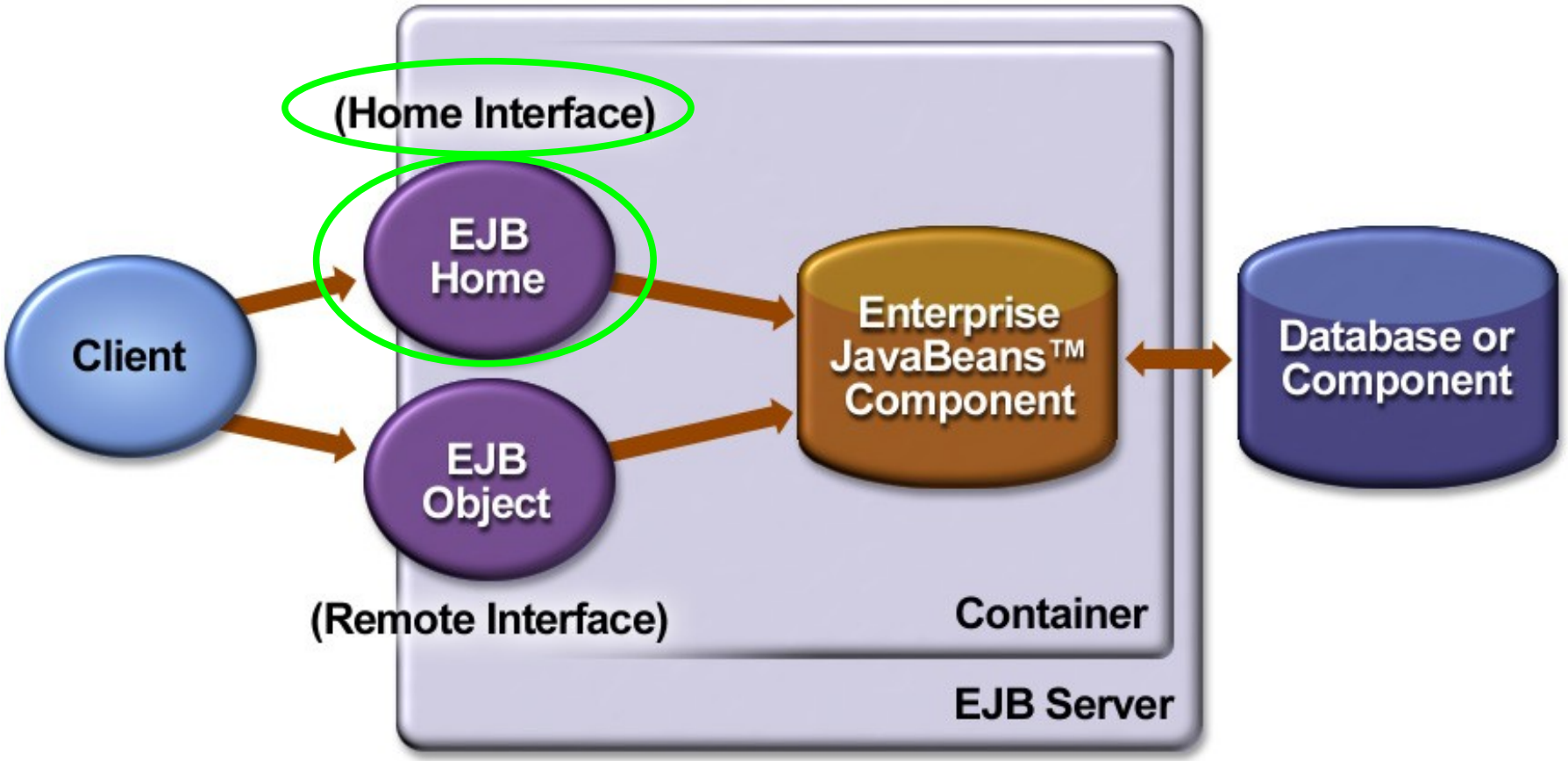
Example: (remote) Home interface

```
package com.kevinboone.ejb_book.interest;
import javax.ejb.*;
import java.rmi.*;

// (Remote) home interface for the `Interest' EJB (c)2002 Kevin Boone
public interface InterestHome extends EJBHome{

   //Create an instance of the EJB
   public Interest create()
         throws CreateException, RemoteException;
}
```

Example: Local Home interface

```
package com.kevinboone.ejb_book.interest;
import javax.ejb.*;
import java.rmi.*;


// (Local) home interface for the `Interest' EJB (c)2002 Kevin Boone
public interface InterestLocalHome extends EJBLocalHome {

  // Create an instance of the EJB
  public InterestLocal create()
            throws CreateException;
}
```

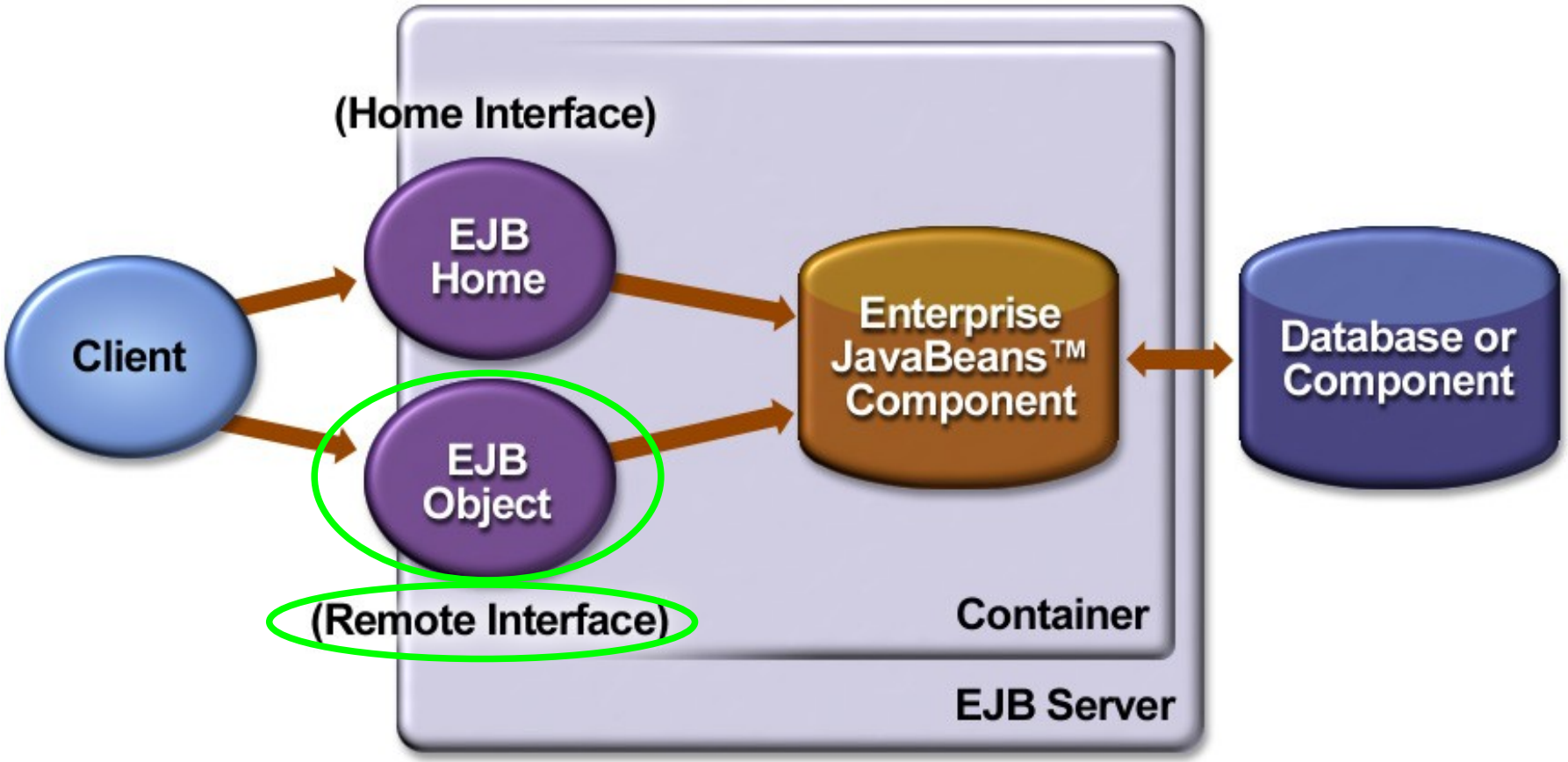
Home Interface and EJB Home object



Logic Interrface (Remote Interface)

- Defines business methods
  - Business methods are methods that deals with application specific business logic
- Implemented by Container
  - EJB object
- Client gets “reference to stub object of the EJB object” through create() or find() method of EJB Home interface
- Can be remote or local

Remote Interface and EJB object



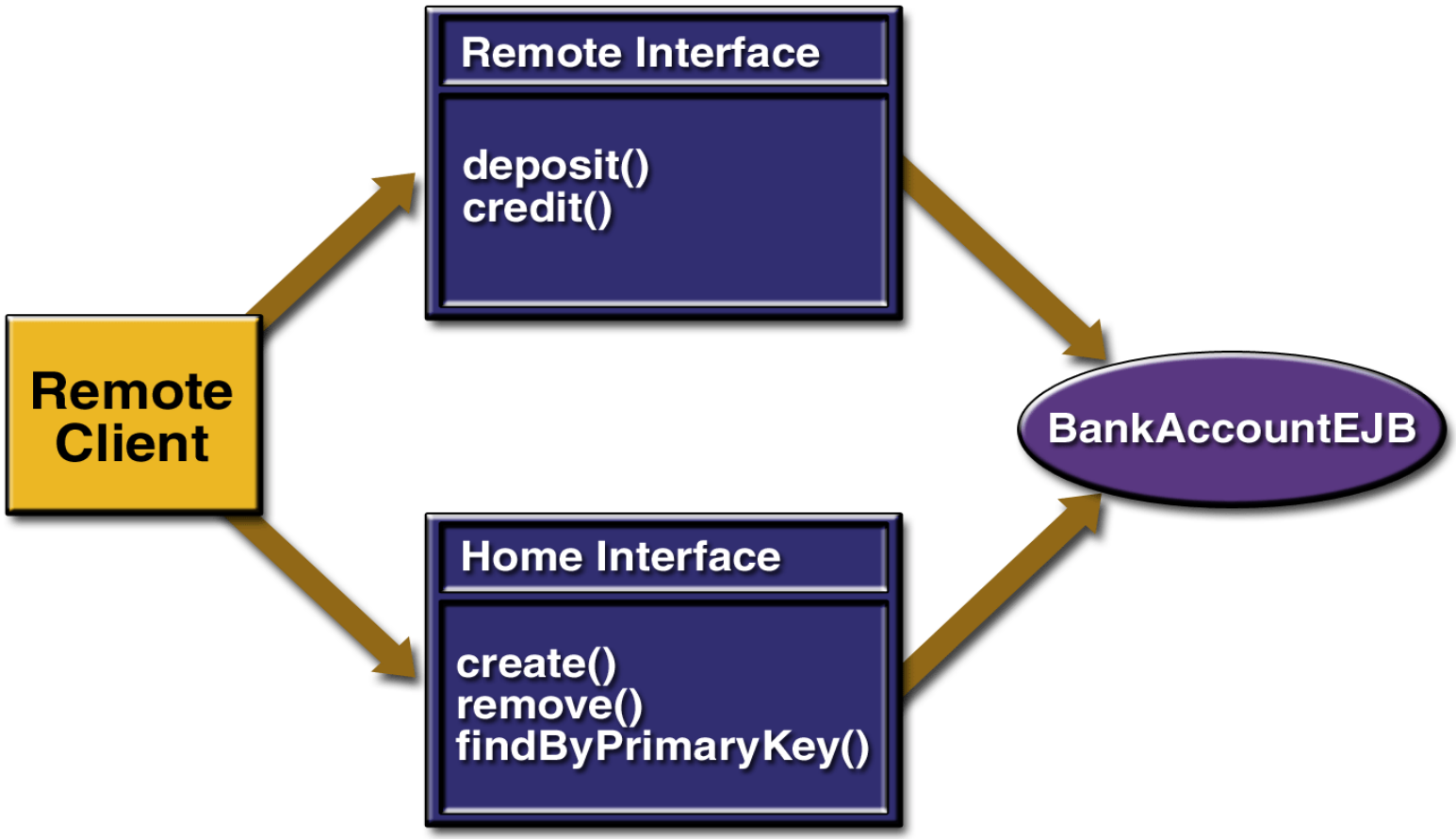
Example: Logic interface (Remote interface)

```
package com.kevinboone.ejb_book.interest;
import javax.ejb.*;
import java.rmi.*;

// Remote interface for the `Interest' EJB (c)2002 Kevin Boone
public interface Interest extends EJBObject{

  // Calculate the interest payable on a certain principal, at a
  // specified  (percent) interest rate per term, for a specific number of terms
  public double getInterestOnPrincipal
     (double principal, double interestPerTerm, int terms)
                                    throws RemoteException;

  // Calculate the total amount payable on a certain principal,
  // at a specified (percent) interest rate per term, for a specific number of terms
  public double getTotalRepayment
     (double principal, double interestPerTerm, int terms)
                                    throws RemoteException;

}
```

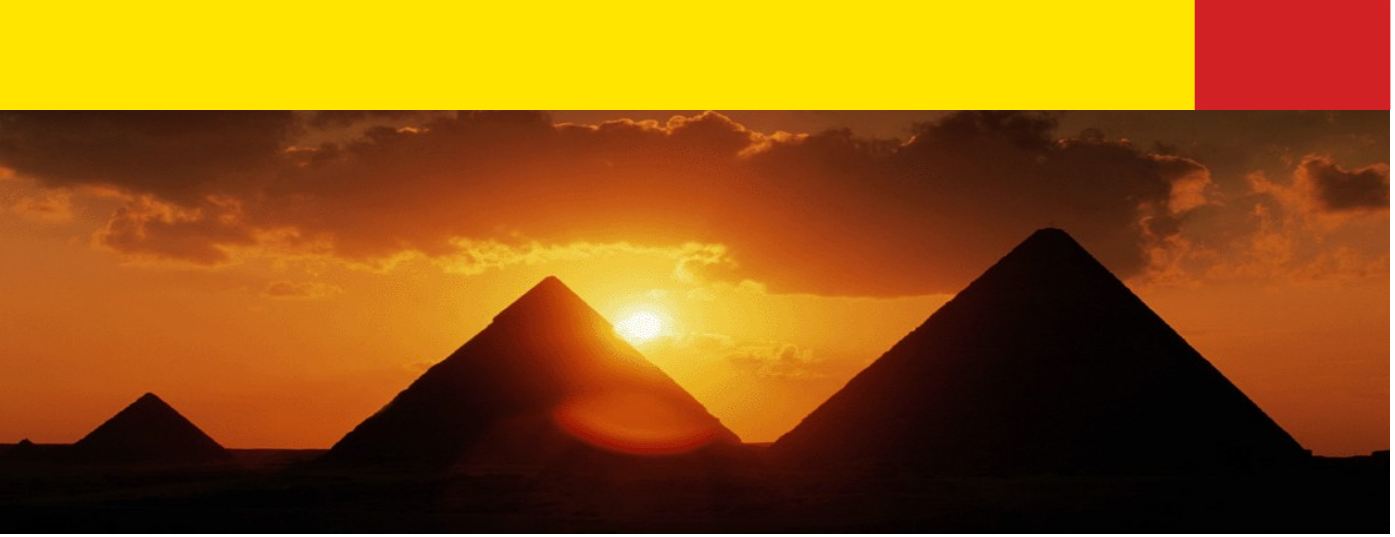
Example: Local Logic interface

```
package com.kevinboone.ejb_book.interest;
import javax.ejb.*;
import java.rmi.*;

// Local interface for the `Interest' EJB (c)2002 Kevin Boone
public interface InterestLocal extends EJBLocalObject {

  // Calculate the interest payable on a certain principal, at a
  // specified interest rate per term, for a specific number of terms
  public double getInterestOnPrincipal
      (double principal, double interestPerTerm, int terms);

  // Calculate the total amount payable on a certain principal,
  // at a specified interest rate per term, for a specific number of terms
  public double getTotalRepayment
     (double principal, double interestPerTerm, int terms);
}
```

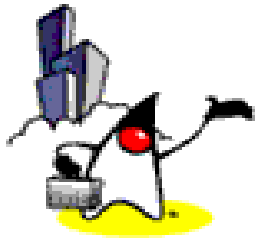
Logic (Remote) & Home Interface



source:J2EE tutorial in java.sun.com



How Client invokes Methods of EJB Bean



Steps for Client To Invoke Methods of EJB

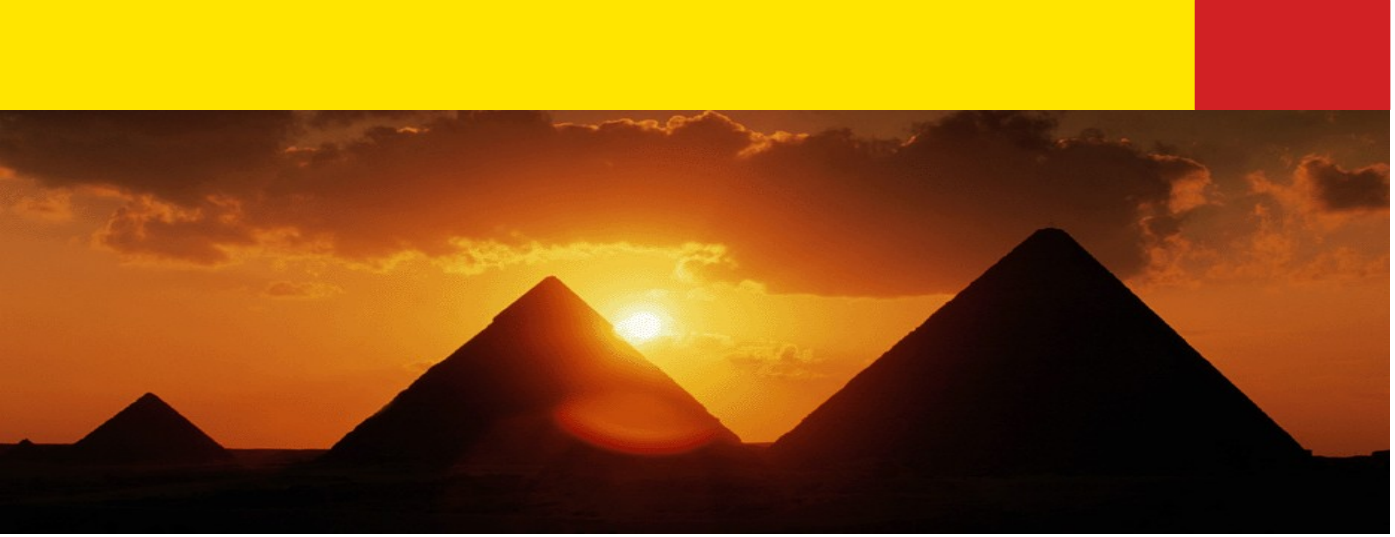
1. Get EJB home object (actually a stub object of the EJB home object) via JDNI
   - Get initial context
   - Perform lookup
   - Perform narrowing
2. From EJB home object, get a EJB object (actually a stub object of the EJB object)
3. Invoke Business methods through EJB object
4. Clean up

Example: Client (page 1)

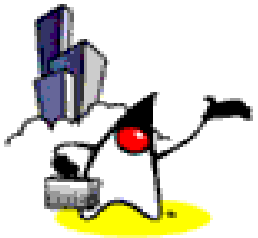
```
package com.kevinboone.ejb_book.interest;
import javax.ejb.*; import javax.naming.*;
import javax.rmi.*;
import java.rmi.*;

// Test client class for the `Interest' EJB (c)2002 Kevin Boone
public class InterestTestClient {
    public static void main (String[] args) throws Exception {
       // Step 1 and Step 2 are captured in a subroutine called getInterest()
       Interest interest = getInterest();
       double principal = 10000.0;double rate = 10.0;int terms = 10;
       System.out.println ("Principal = $" + principal);
       System.out.println ("Rate(%) = " + rate);
       System.out.println ("Terms = " + terms);
       // Step 3: Invoke business methods
       System.out.println ("Interest = $" +
                           interest.getInterestOnPrincipal (principal, rate,
    terms));
       System.out.println ("Total = $" +
                           interest.getTotalRepayment( principal, rate, terms));
       // Step 4: Clean up, remove instance of stateless session bean
```

Example: Client (page 2)

```
    // Get an instance of the Interest EJB. Note that the
    // EJB-specific stuff is wrapped up in this method,
    // so that the main logic can just be plain Java
    public static Interest getInterest()
            throws CreateException, RemoteException, NamingException {

        // Step 1: Get an instance of EJB home object (actually a
        // stub object to EJB home object) via JNDI
        InitialContext initialContext = new InitialContext();
        Object o = initialContext.lookup ("Interest");
        InterestHome home = (InterestHome)
            PortableRemoteObject.narrow (o, InterestHome.class);

        // Step 2: Create a EJB remote object (actually a stub object to EJB
        // remote object) through EJB home object
        return home.create();
    }

}
```



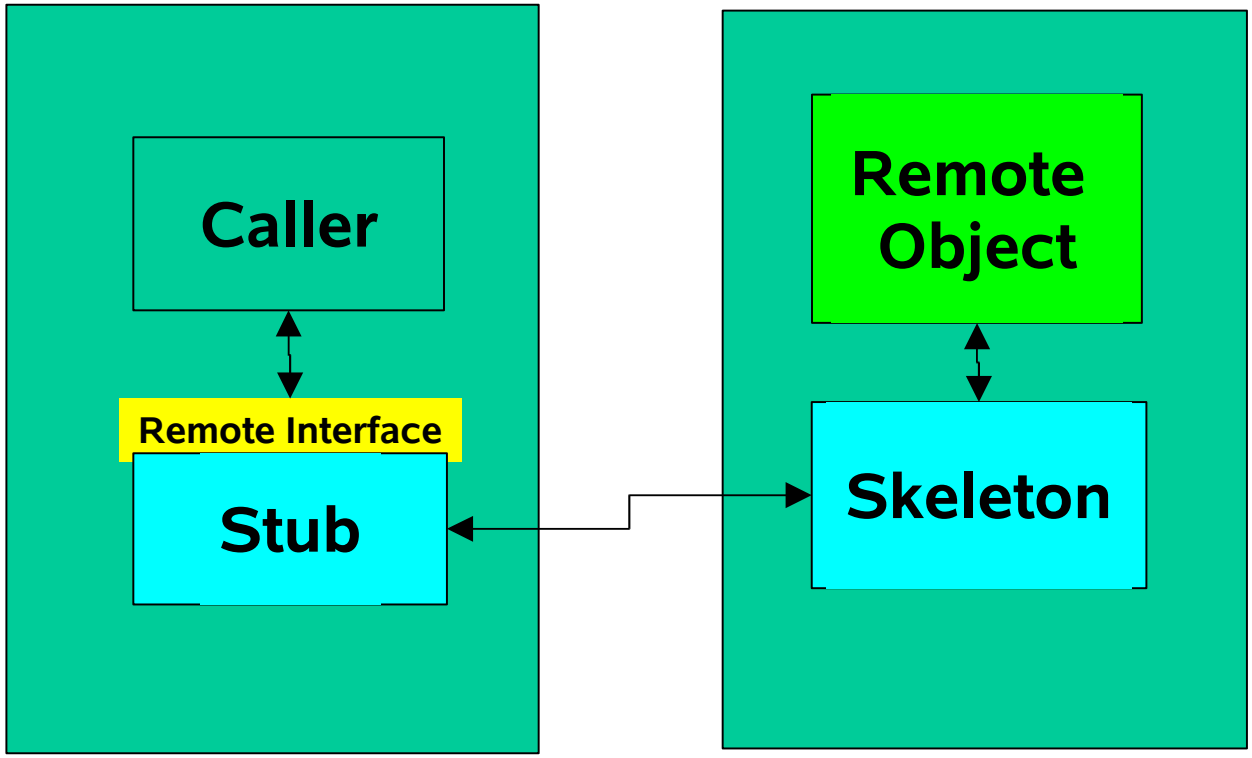
RMI Communication Model



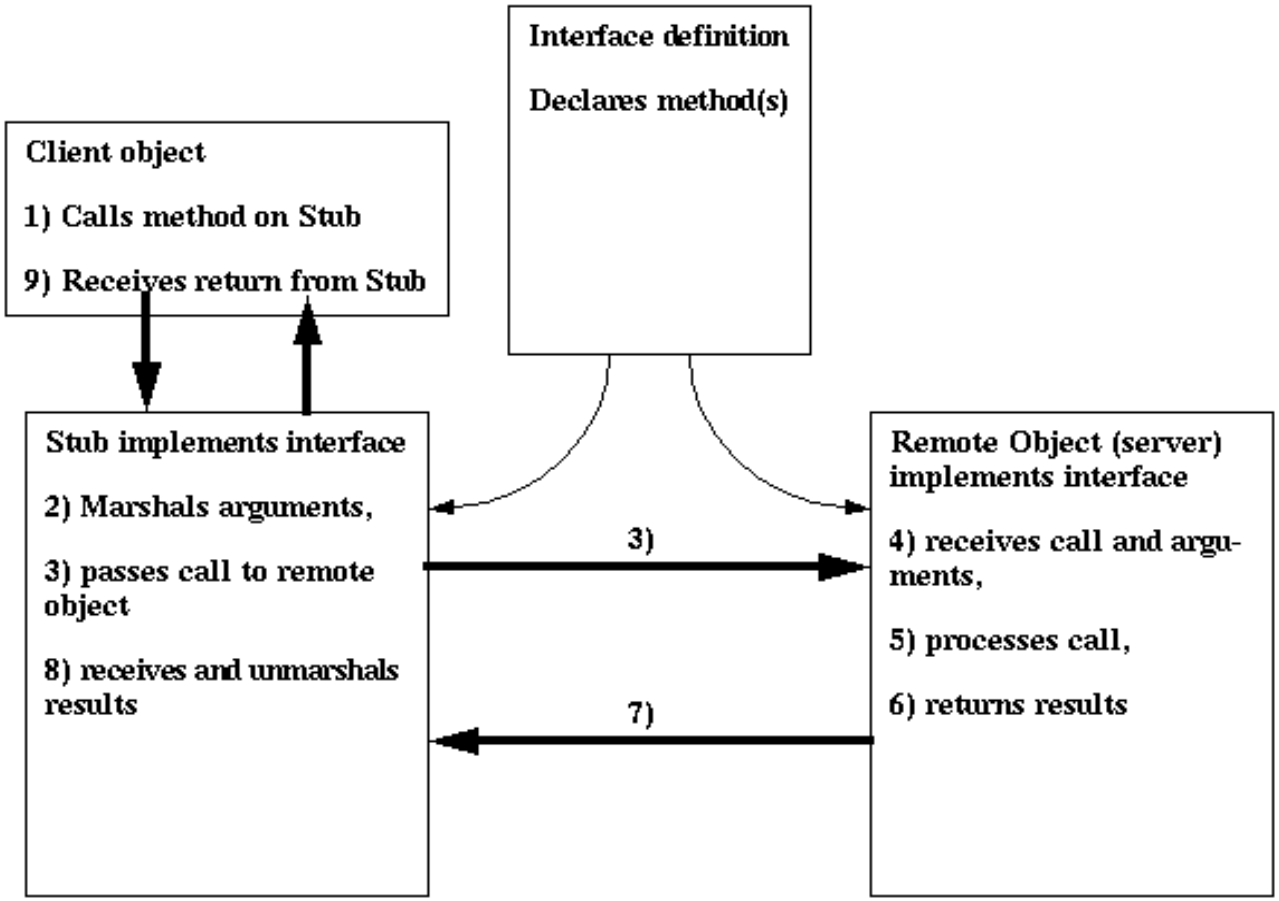
Distributed Objects

- Collaborating objects reside in different computers (different VMs)
  - client object invokes methods of server object
- There have to be some mechanisms
  - passing method signature from client to server
  - marshalling parameters from client to server
  - unmarsalling received parameters at the server
  - marshalling return values from server to client
  - unmarshalling received results at the client

RMI Communication Model



RMI Control Flow



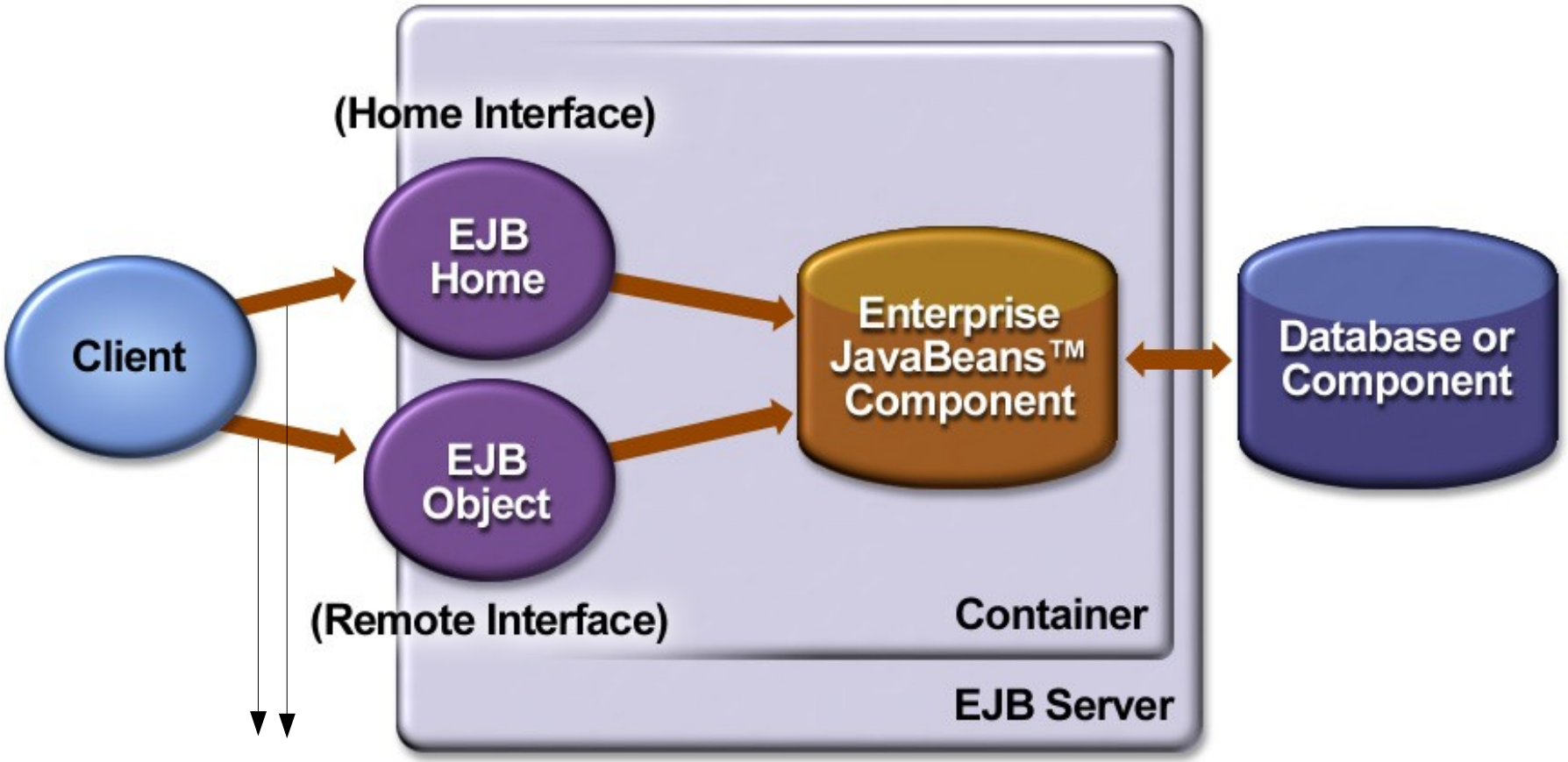
RMI Control Flow

- Caller (Client)
  1. invokes a method of a remote object
- Stub of the remote object
  1. intercepts the method call
  2. marshals the arguments
  3. makes calls to remote object

RMI Control Flow

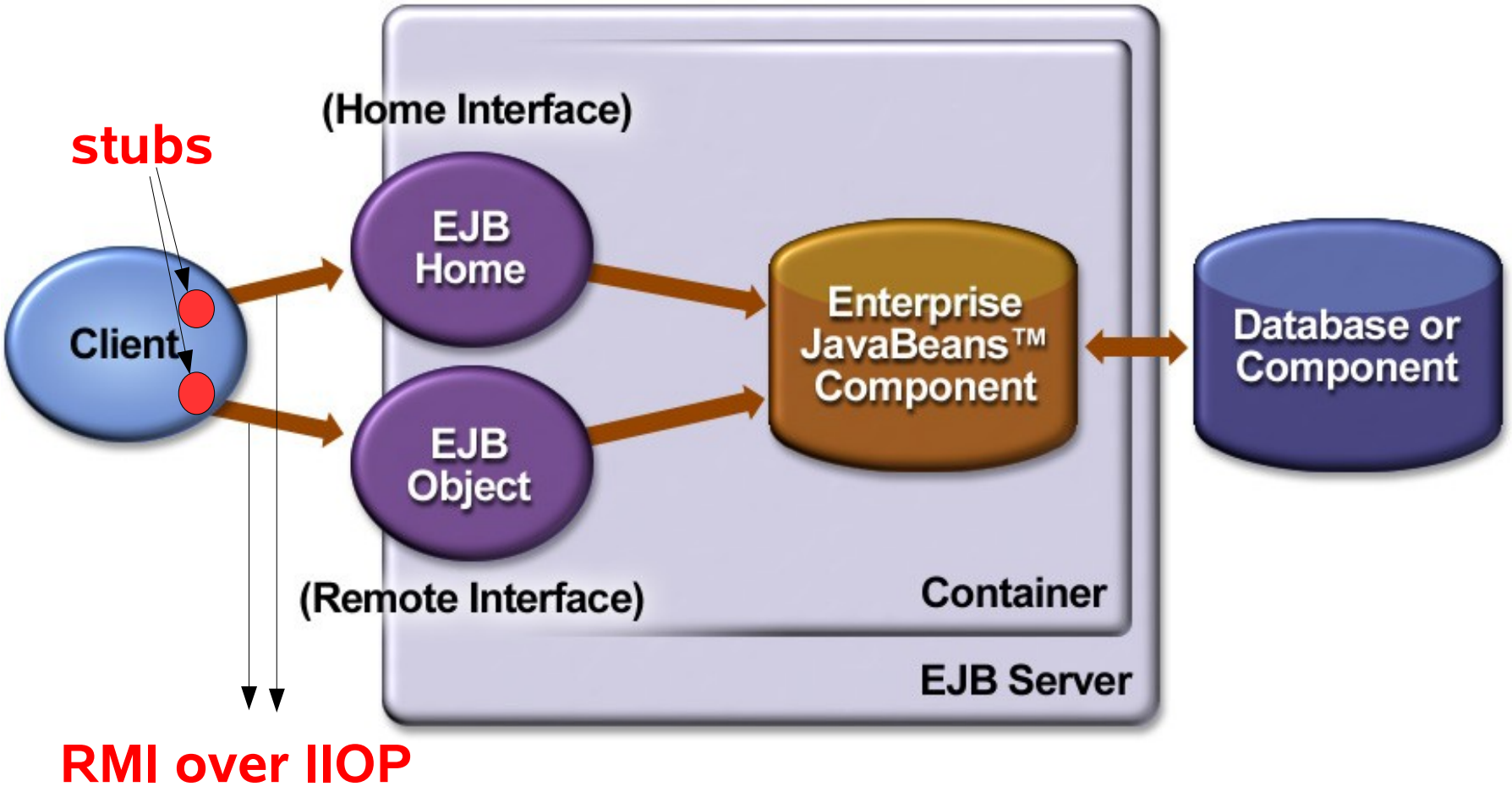
- Remote object
  1. Receives the calls via Skeleton
  2. Unmarshals the arguments
  3. Performs the call locally
  4. Marshals the result
  5. Send the result to client
- Stub
  1. Receives result
  2. Unmarshal result
  3. Return result to client

Where does RMI Communication occur?

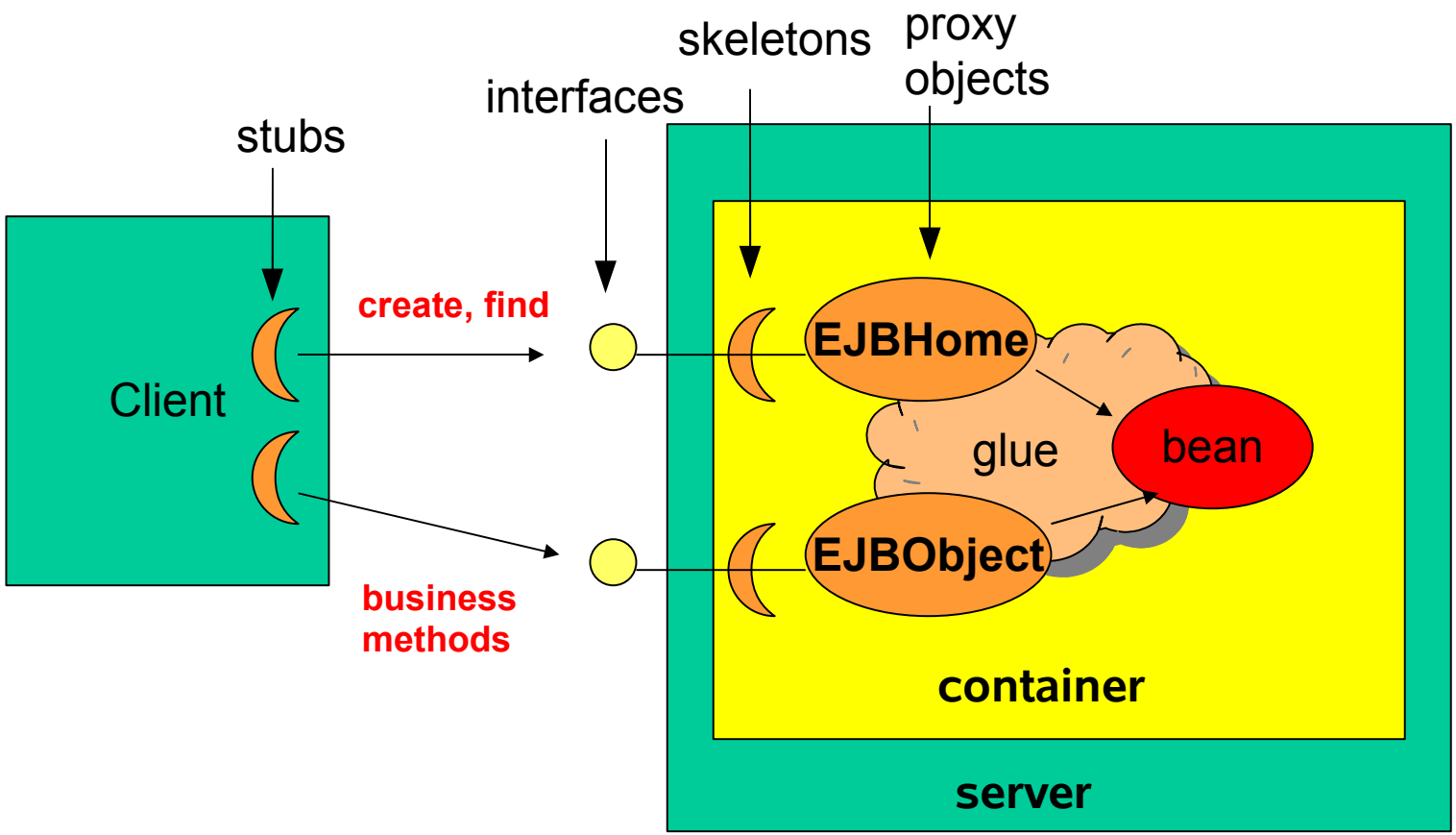


RMI over IIOP

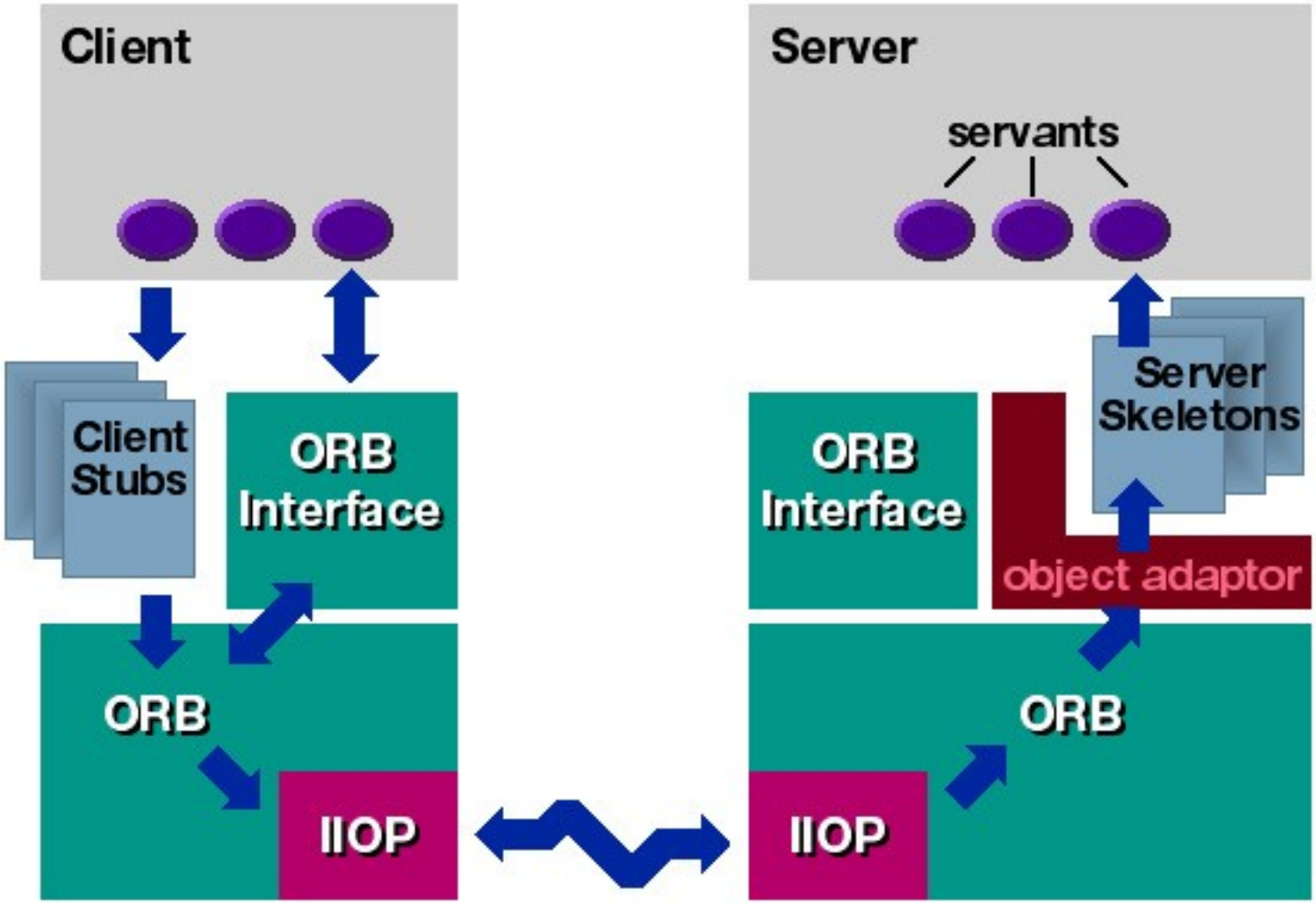
Where are Stubs?



EJB™ Home & EJB™ Object

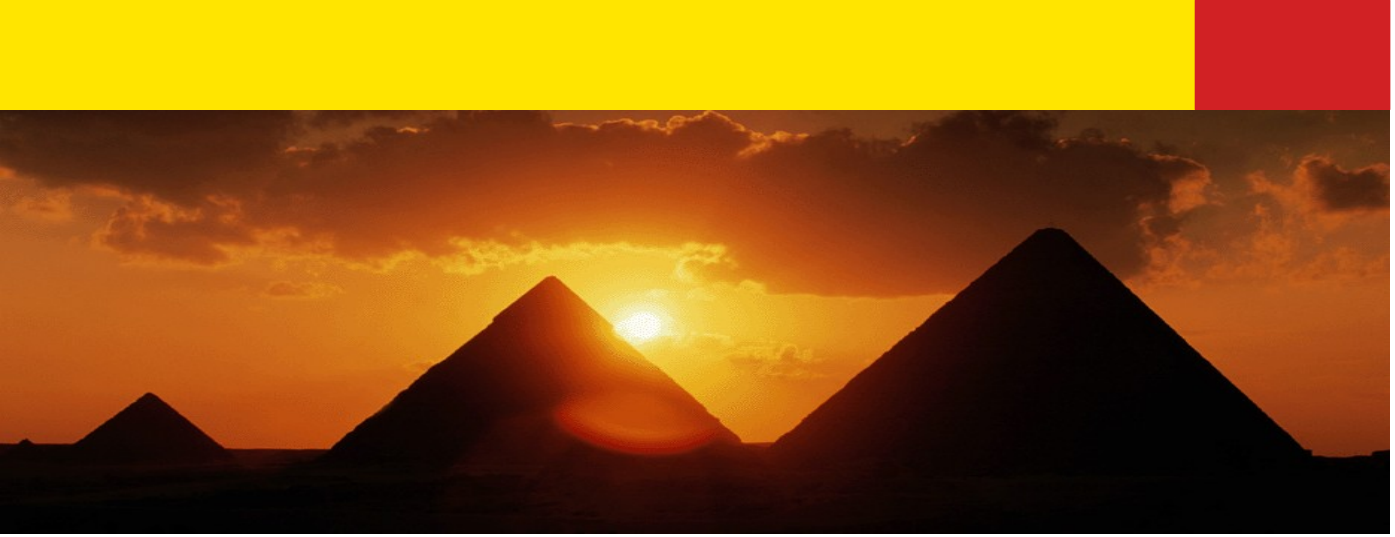


RMI over IIOP

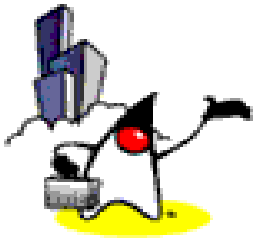


RMI over IIOP

- RMI mechanism used between EJB client and EJB server
  - Before EJB 2.0 (J2EE 1.3), RMI over IIOP has to be used even if EJB client and EJB server reside in the same VM, which results in unnecessary overhead
  - Vendor products provide some optimized communication if client and server reside in a single VM
- RMI operations are in general expensive
  - Reason why “local interface” are introduced in



Local versus Remote Client View



Local Client View

- Local interfaces
- Used when client resides in the same VM with EJB bean (thus with EJB container)
- No overhead of RMI over IIOP communication
- Introduced in EJB 2.0 (J2EE 1.3)
- Methods do not have to throw RemoteException
- "Call-by-reference" is possible

Local Client

- Typically used to make Session beans functioning as local clients to local entity beans

Local Interface

- Advantages
  - More efficient access due to co-location due to no RMI over IIOP overhead
  - Ability to share data between client and bean through call by reference
- Disadvantages
  - Tight coupling of client and bean
  - Less flexibility in distribution
- Use local interface beans (as opposed to remote beans) for fine-grained operations

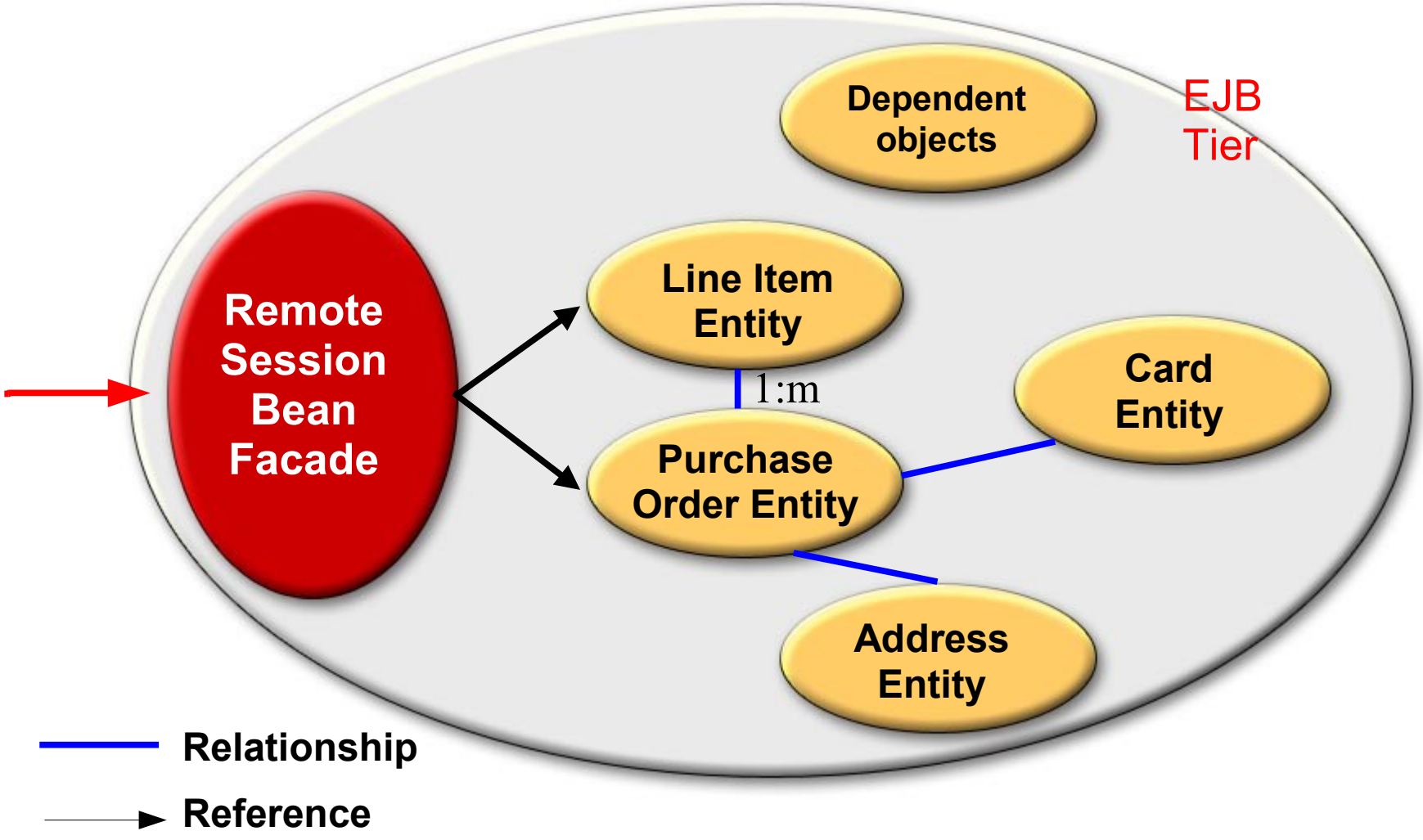
Can Local and Remote Interfaces be used together?

- You can build your bean to support both local and remote interface at the same time
- EJB client cannot be both, however
  - This is compile time decision (rather than runtime decision)

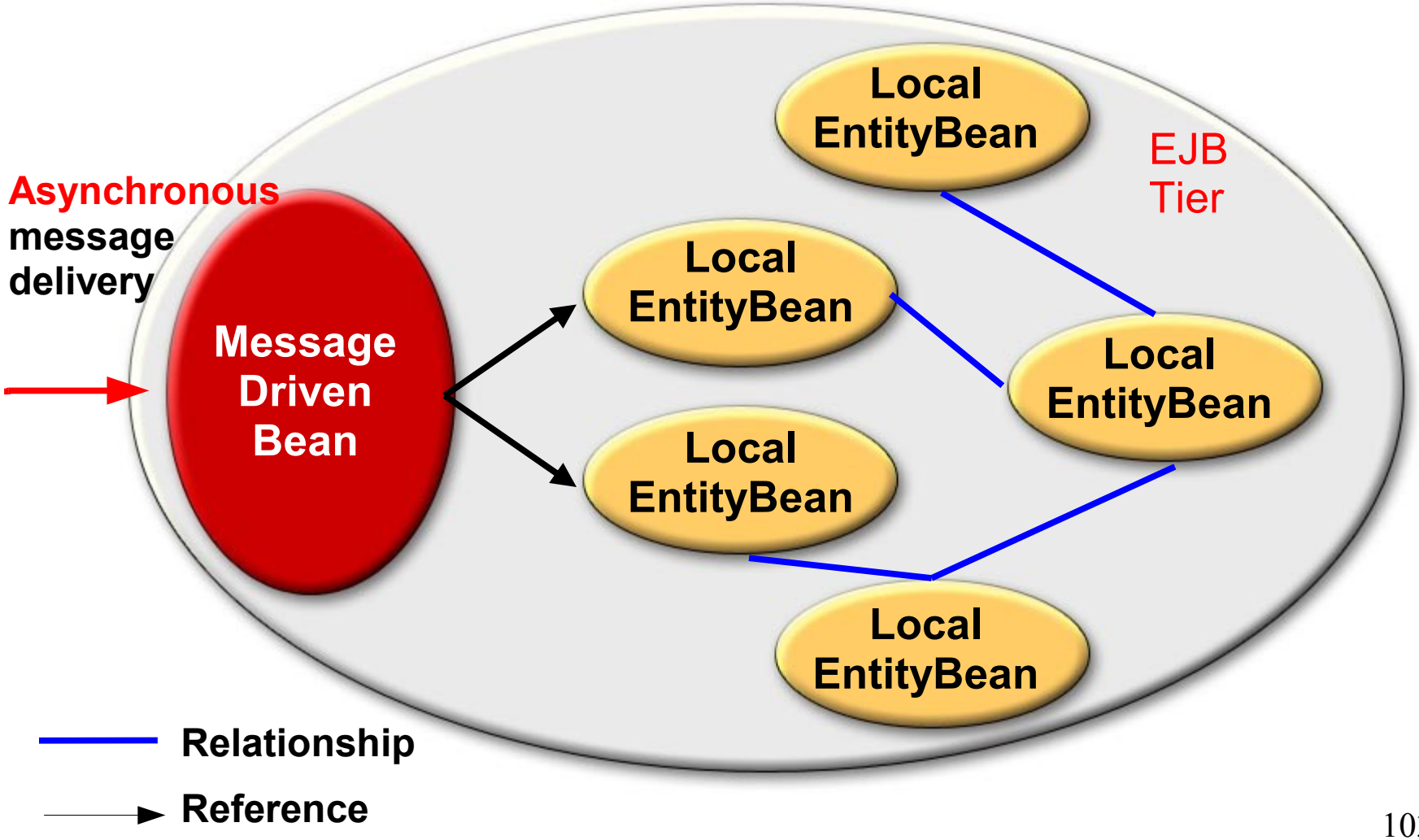
Recommendations

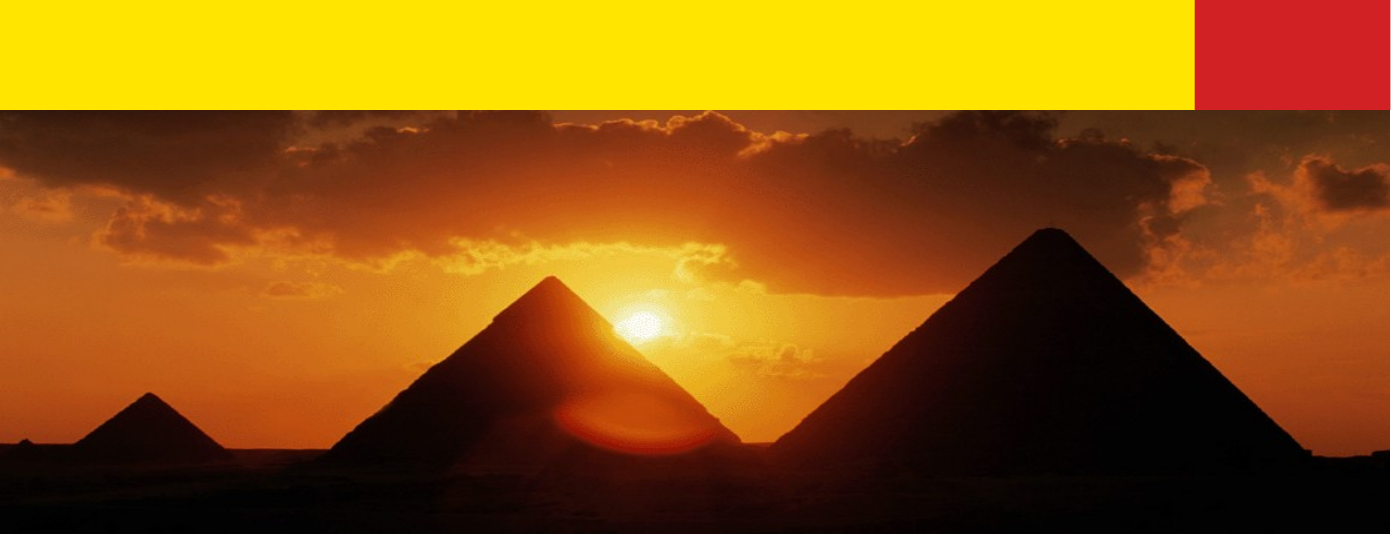
- Use local interface whenever possible
  - Create islands of local components (local entity beans and their dependent objects)
- Use facade pattern in which a remote interface session bean (for synchronous operations) or message driven bean (for asynchronous calls) invokes local entity beans
- Use remote interface for loose coupling

Remote Session Bean Facade With a Network of Local Entity Beans

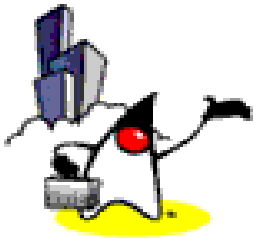


Message Driven Bean Facade With a Network of Local Entity Beans





EJB-JAR Deployment Descriptor



EJB-JAR Deployment Descriptor

- Gives the container instructions on how to manage the enterprise bean
- Allows declarative customization
- Controls behaviors for:
  - Transaction
  - Security
  - Life cycle
  - State management
  - Persistence
  - …

EJB-JAR Deployment Descriptor

- Defines contract between producer and consumer of ejb-jar file
- It is an XML document that must be well formed in XML sense and must be valid with respect to the DTD given in EJB specification
- Must be stored with the name META-INF/ejb-jar.xml in the ejb-jar file
- Captures two basic kinds of information viz. Structural and Application assembly information

EJB-JAR Deployment Descriptor: Structural Information

- Describes structural and external dependencies of enterprise bean
- Mandate on ejb-jar producer to provide structural information
- Should not be changed since it may break the enterprise bean's function
- Example
  - Fully qualified class name of Enterprise bean class, home interface and remote interface
  - State management type of session bean, etc.

EJB-JAR Deployment Descriptor: Application Assembly Information

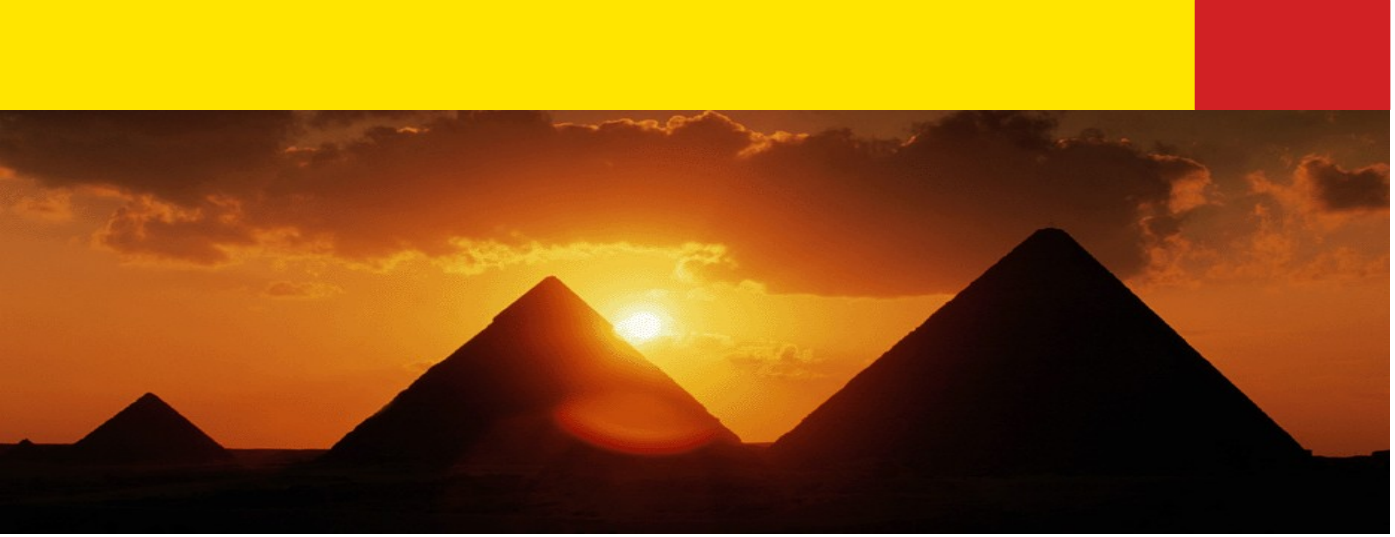
- Describes how enterprise bean(s) in ejb-jar file can be composed into a larger application deployment unit
- Can be optionally provided by ejb-jar producer
- Changes may not break functionality of enterprise bean, however doing so may change its behavior
- Examples
  - Transaction attributes

Example: Deployment Descriptor of EJB-JAR file: ejb-jar.xml (page 1)

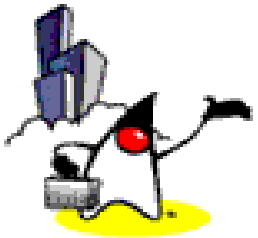
```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN'
    'http://java.sun.com/dtd/ejb-jar_2_0.dtd'>

<ejb-jar>
 <display-name>Interest_ejb</display-name>
 <enterprise-beans>
  <session>
   <display-name>InterestBean</display-name>
   <ejb-name>InterestBean</ejb-name>
   <home>com.kevinboone.ejb_book.interest.InterestHome</home>
   <remote>com.kevinboone.ejb_book.interest.Interest</remote>
   <local-home>com.kevinboone.ejb_book.interest.InterestLocalHome</local-home>
   <local>com.kevinboone.ejb_book.interest.InterestLocal</local>
   <ejb-class>com.kevinboone.ejb_book.interest.InterestBean</ejb-class>
   <session-type>Stateless</session-type>
   <transaction-type>Bean</transaction-type>
   <security-identity>
    <description></description>
    <use-caller-identity></use-caller-identity>
   </security-identity>
  </session>
 </enterprise-beans>
```

Example: Deployment Descriptor of EJB-JAR file: ejb-jar.xml (page 2)

```
<assembly-descriptor>
  <method-permission>
    <unchecked />
    <method>
      <ejb-name>InterestBean</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>getHandle</method-name>
      <method-params />
    </method>
    ...
  </method-permission>
</assembly-descriptor>
</ejb-jar>
```



EJB Packaging



Package Files

- J2EE Application (sometimes called EJB application)
  - *.EAR file
  - Can contain Web tier modules (*.WAR files) and EJB-JAR files
- EJB-JAR (EJB Module)
  - *.jar file
  - Some container allows direct deployment of EJB-JAR files
- EJB Client Jar

*.EAR file

- Contains both Web-tier modules and EJB Modules (EJB-JAR files)
  - It can contain multiple EJB-JAR files
- Has its own deployment descriptor
  - application.xml
- For deploying EJB application over J2EE RI, you have to create *.EAR file even if you have only one EJB-JAR file and no Web modules
  - Some container allows direct deployment of EJB-JAR

Example: Deployment Descriptor of J2EE Application: application.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE application PUBLIC '-//Sun Microsystems, Inc.//DTD J2EE
    Application 1.3//EN' 'http://java.sun.com/dtd/application_1_3.dtd'>

<application>
  <display-name>interest</display-name>
  <description>Application description</description>
  <module>
    <ejb>ejb-jar-ic.jar</ejb>
  </module>
</application>
```

Interest.EAR file structure

- ejb-jar-ic.jar
- META-INF
  - application.xml
  - ejb-jar.xml
  - sun-j2ee-ri.xml
- com
  - com\kevinboone\ejb_book\interest
    - InterestLocal.class
    - InterestLocalHome.class
    - InterestBean.class
    - InterestHome.class
    - Interest.class

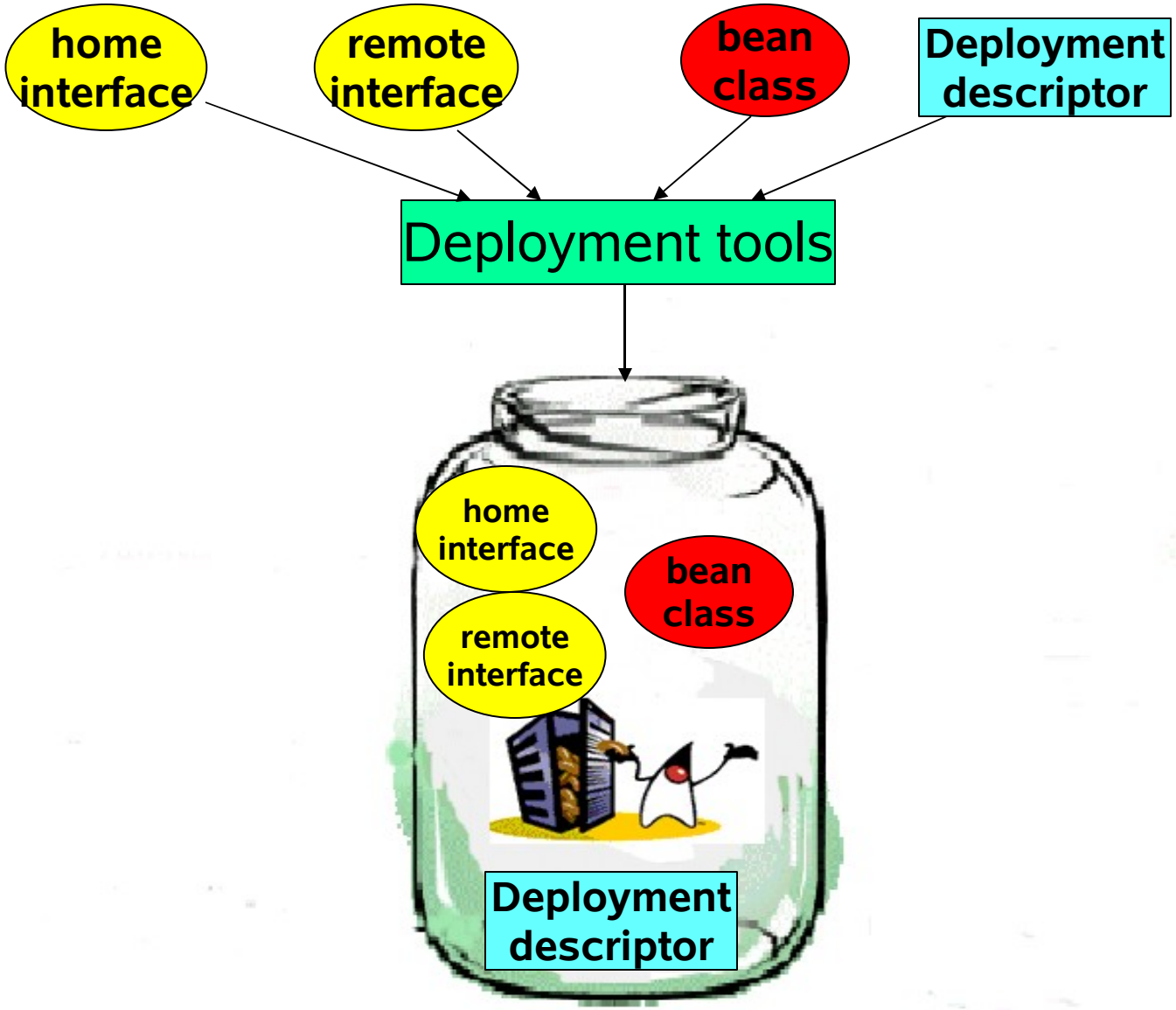
EJB-JAR file

- Standard format for packaging enterprise beans
- Used to package un-assembled and assembled enterprise beans
- Must contain deployment descriptor
- For each enterprise bean, ejb-jar file must contain following class files
  - Enterprise bean class
  - Enterprise bean home and remote interface
  - Primary key class if the bean is entity bean

Ejb-client JAR file

- Ejb-jar file producer can create ejb-client JAR file for ejb-jar file
- Consists of all classes that client program needs to use client view of enterprise beans contained in ejb-jar file
- Can be specified in deployment descriptor of ejb-jar file
- Deployer should ensure that specified ejb-client JAR file is accessible to client program's class loader

EJB™ Role: Bean Provider



EJB™ Role: Application Assembler

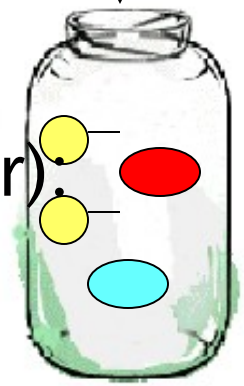
ejb-jar (.jar)

Web jars (.war): servlets, JSP…

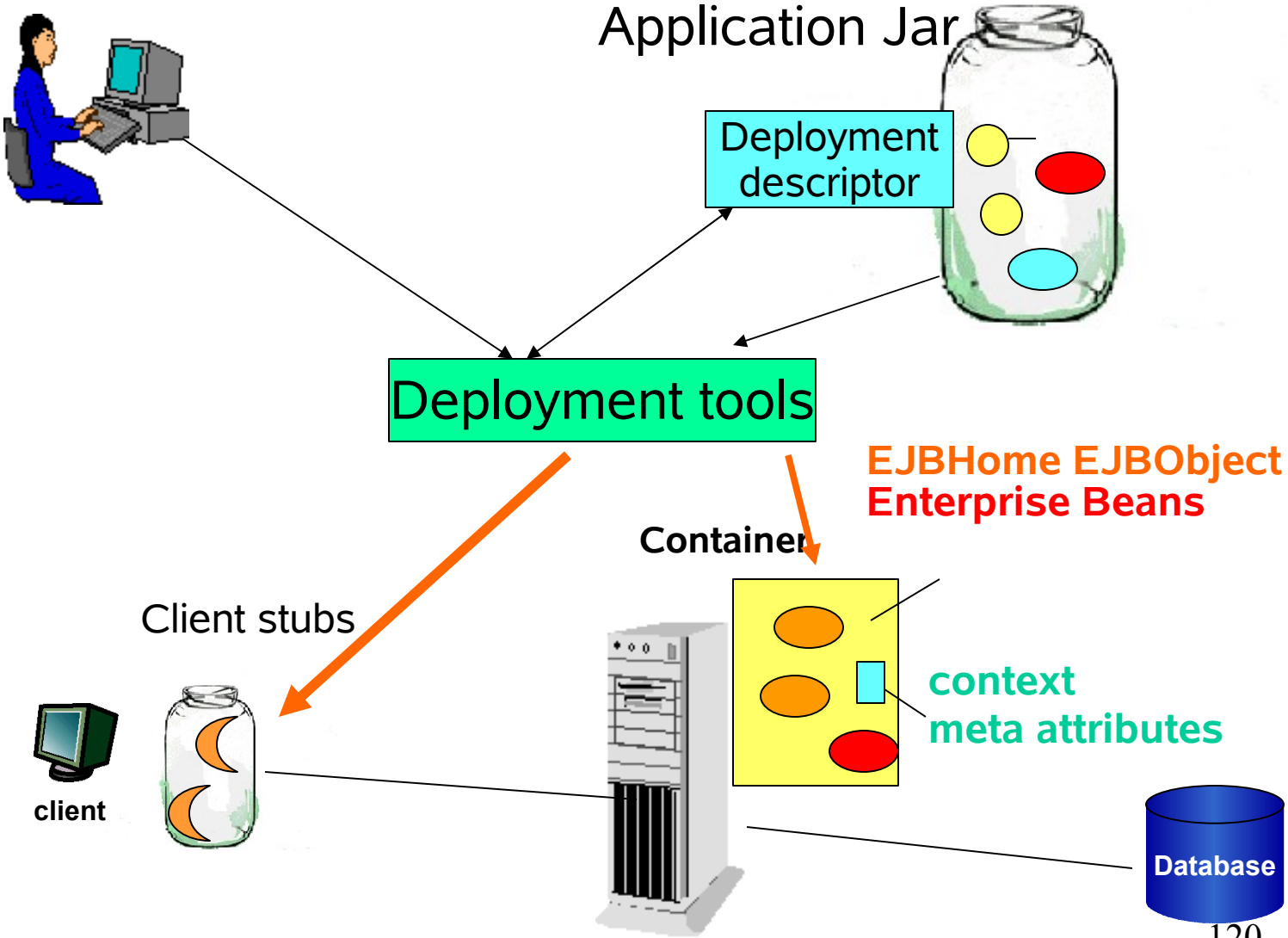
Deployment descriptor

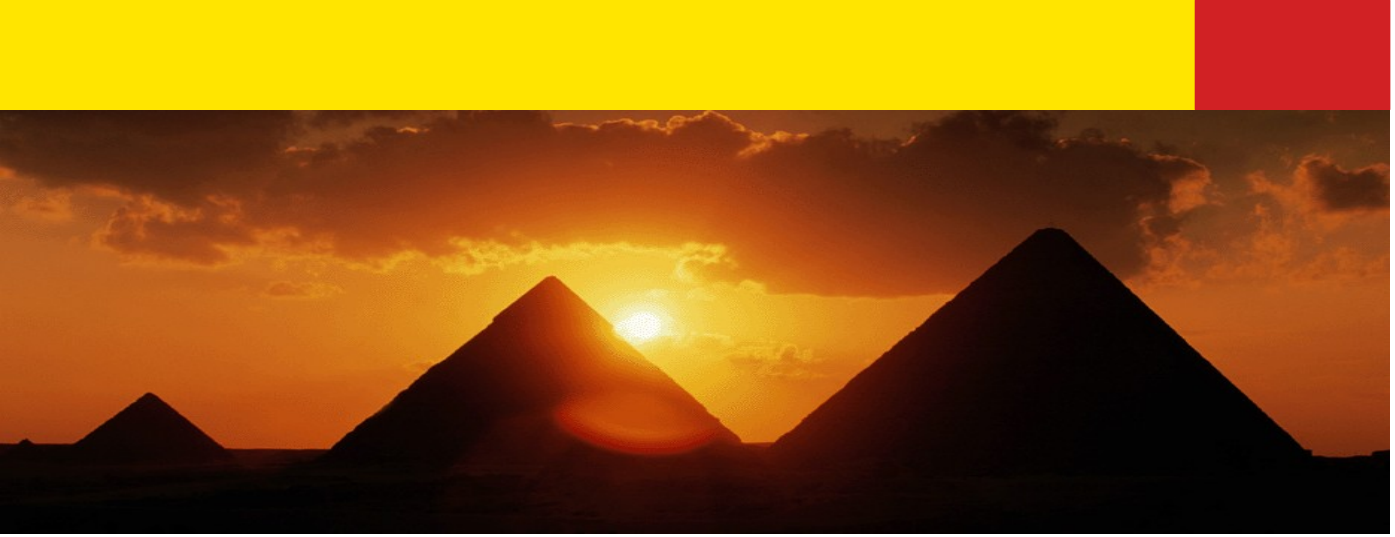
Deployment tools

Application Jar (.ear):

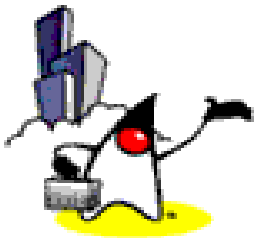


EJB™ Role: Deployer





Naming Conventions For EJB Development

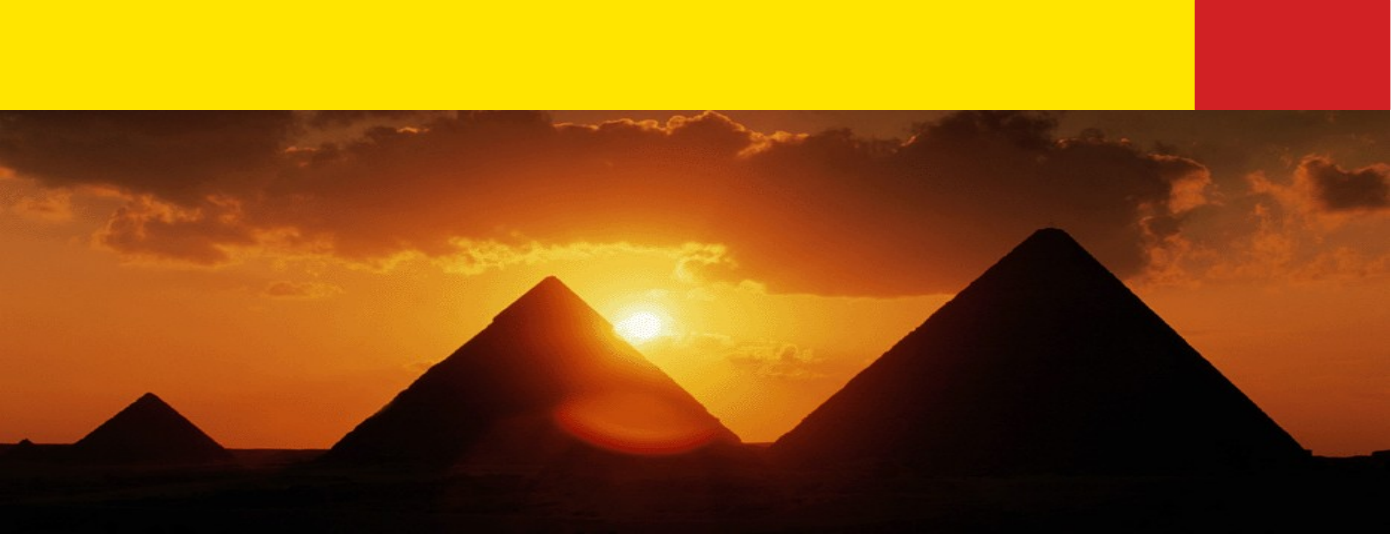


Naming Conventions

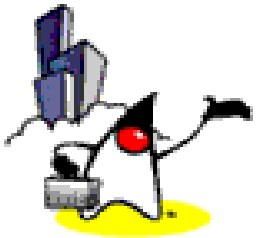
- Home Interface
  - CatalogHome (remote), CatalogLocalHome (local)
- Logic Interface (Remote interface)
  - Catalog (remote), CatalogLoca (local)
- Bean class
  - CatalogBean
- EJB-JAR Deployment Descriptor
  - ejb-jar.xml (Same for all Beans)

Naming Conventions (Only a suggestion)

- EJB-JAR file & display name
  - <custom-name>.jar, ex: petstore.jar
  - <custom-name>JAR.jar, ex: petstoreJAR.jar
- EJB Application & display name
  - <custom-name>.ear, ex: interest.ear
  - <custom-name>EAR.ear, ex:interestEAR.ear
- EJB Application deployment descriptor
  - application.xml (Same for all apps)
- Container specific deployment descriptor
  - <custom-name>.xml, ex: sun-j2ee-ri.xml

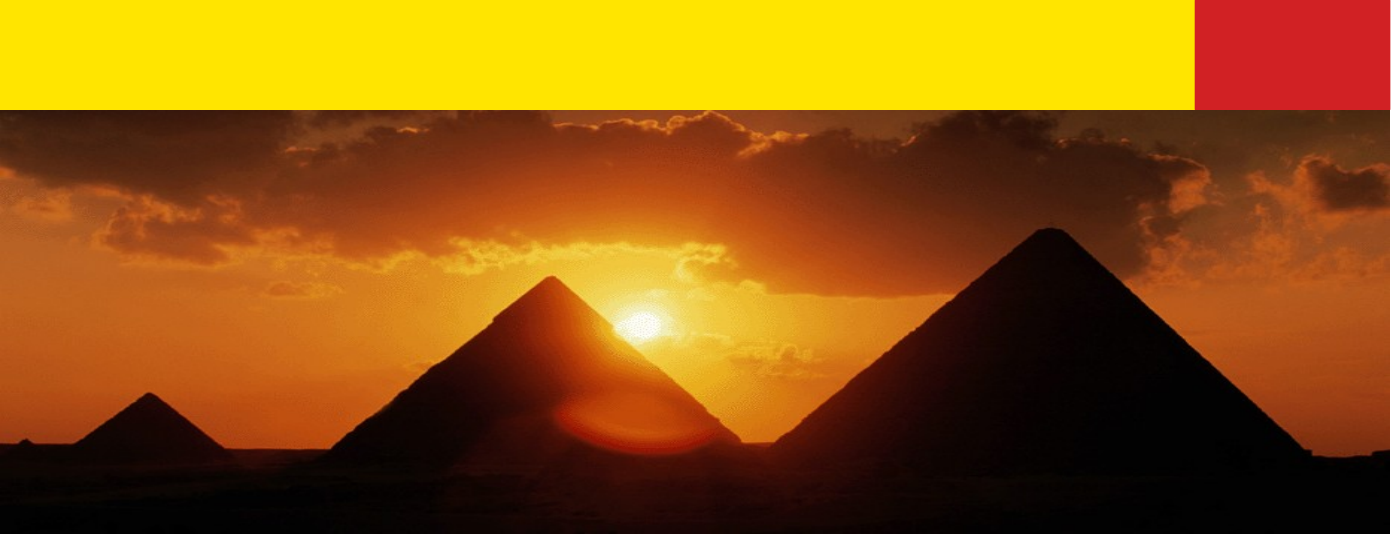


EJB 2.0 Features

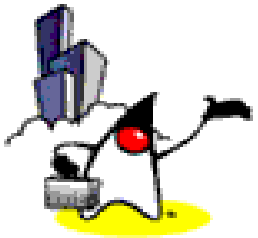


EJB 2.0 New Features

- Integrated support for JMS
- Support for local interfaces
- Improved architecture for Container Managed Persistence
  - Support for container managed relationships among entity beans
- Enterprise JavaBeans™ Query Language (EJB QL)
- Home methods for entity beans
- Network interoperability



EJB and JMS

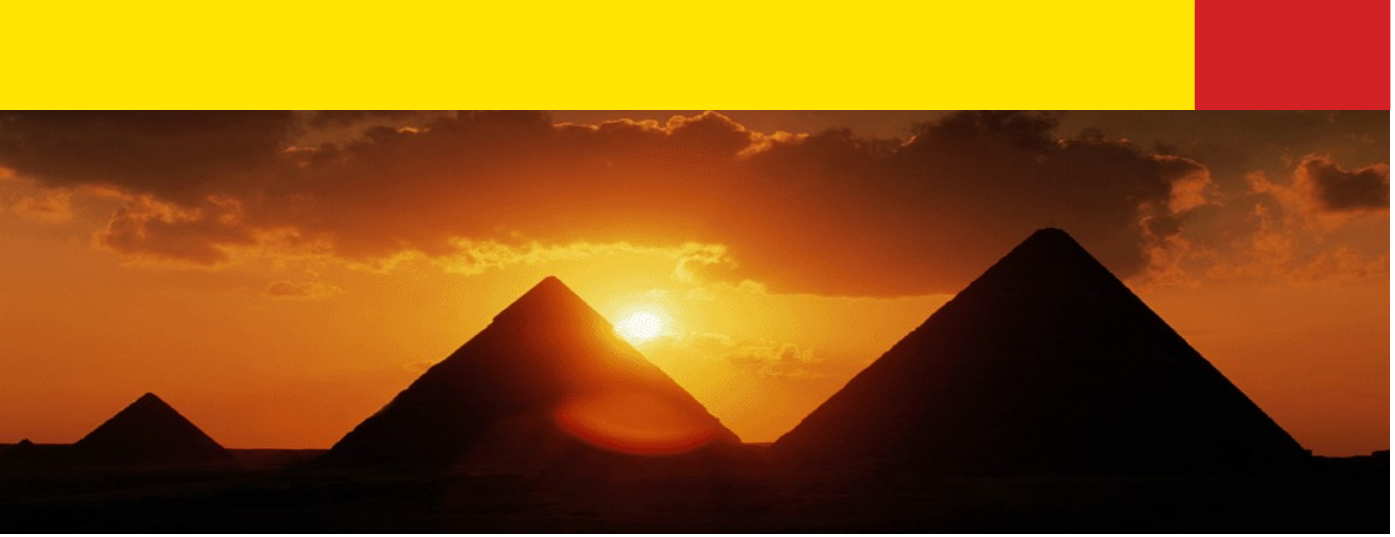


JMS Support

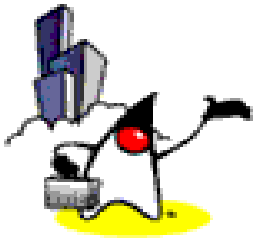
- By adding new enterprise bean type called "MessageDrivenBean"
  - Stateless bean without home and remote interface
  - Activated upon message arrival
  - Bean implements javax.jms.MessageListener interface
  - onMessage() method implementation should contain business logic
  - Configured as listener for queue or topic

JMS and EJB

- JMS APIs for sending messages available to all enterprise beans
  - Use in point-to-point configurations a.k.a Reliable Queuing
  - Use within pub/sub configurations

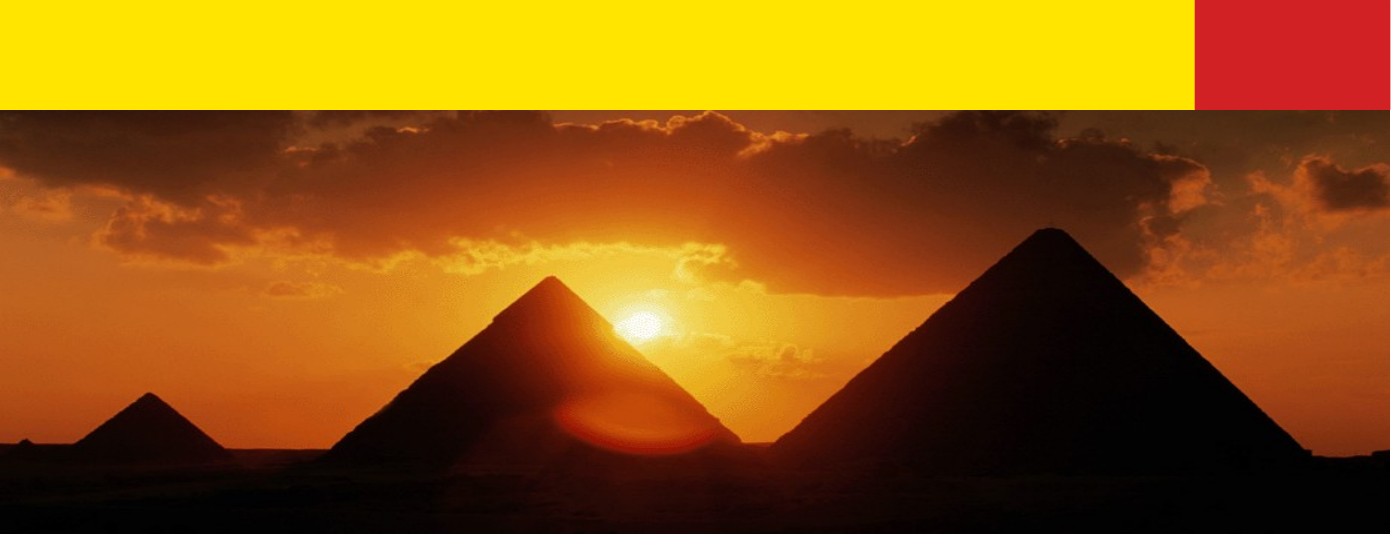


Home Methods



Home Methods

- Additional methods on Home interface for entity beans with both bean and container managed persistence
  - Business methods that are not specific to an entity bean instance
  - Implementation provided by Bean Provider
  - Useful for bulk updates or other aggregate operations



Passion!