23. Enterprise Java Beans

Lecturer: Dr. Sebastian Götz

Prof. Dr. Uwe Aßmann
Technische Universität Dresden
Institut für Software- und Multimediatechnik
http://st.inf.tu-dresden.de/teaching/cbse

1. Mai 2017

Obligatory Reading

- Oracle's enterprise bean tutorial
  http://docs.oracle.com/javaee/5/tutorial/doc/bnbls.html
  http://docs.oracle.com/javaee/
  http://docs.oracle.com/javaee/5/tutorial/doc/javaeetutorial5.pdf
- Szyperski, Chap 14
- http://xdoclet.sourceforge.net
- EJB 3.0 Features
  http://www.oracle.com/technetwork/java/index.html
- JBoss has a EJB 3.0 tutorial
  http://docs.jboss.org/ejb3/docs/tutorial/1.0.7/html/index.html
- Red Hat JBoss documentation
  - https://access.redhat.com/site/products/red-hat-jboss-enterprise-application-platform/

Other Literature

- JBoss EJB 3.0 Documentation
  http://docs.jboss.org/ejb3/app-server/
- Ed Roman: Mastering EJB. Wiley & Sons.
  http://www.theserverside.com/books/wiley/masteringEJB/index.jsp
- B. Tate, M. Clark, B. Lee, P. Linskey: Bitter EJB. Manning Publications Co.

23.1 Basics of EJB

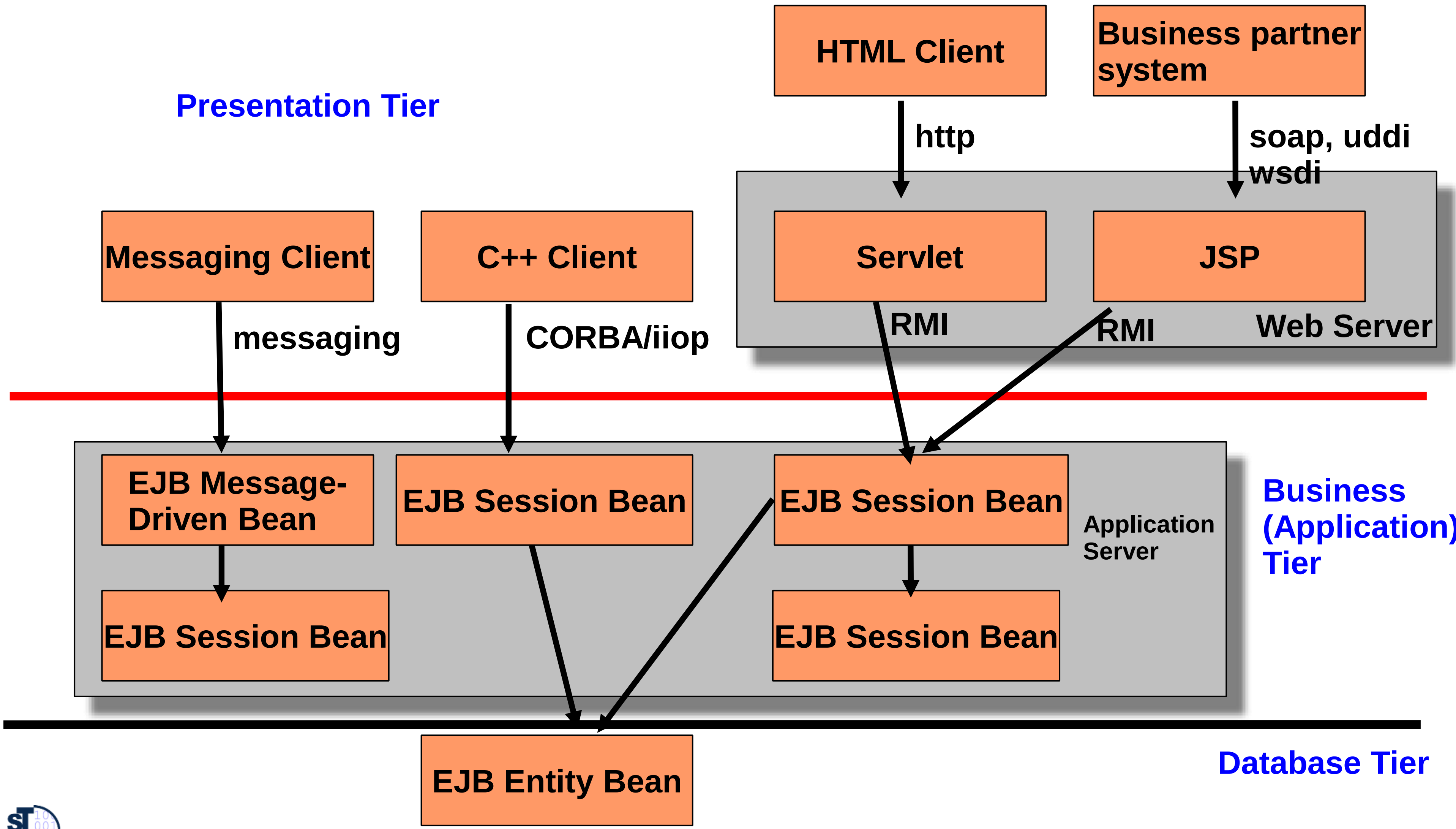
Basics of Enterprise Java Beans (EJB)

- Developed by SUN, now Oracle
  - Server-side component architecture for building distributed OO business applications in Java
  - Separation of business logic and lower-level concerns (e.g., networking, transactions, persistence, ...) into *implicit middleware*
- EJB 1.0 1998, EJB 2.0 2001, current version is 3.2
- EJB integrates several concepts for **Dynamic deployment:**
  - Deployment-time middleware code generation (implicit middleware)
  - Containers as application servers for transparency of transaction and persistency
  - Annotation-based (metadata-based) middleware code generation
  - A simple XML-based composition language
- Some common EJB application servers
  - OSS: JBoss – free software www.jboss.org
    - Apache Geronimo
  - Commercial: BEA's WebLogic, IBM's WebSphere, Oracle's Oracle 11g

Ingredients of EJB

- Java-based Component Model (language specific)
  - Static components contain classes
  - Dynamic components contain objects
- Component Types:
  - **Session Beans:** for business logic and application algorithms (Tools)
  - **Message-Driven Beans:** Same function as session beans
    - Called by sending messages instead of calling methods
    - Have a message queue, react to an asynchronous message connector
  - **Entity Beans:** for business objects (data, Materials)
    - Persistent object that caches database information (an account, an employee, an order, etc)
  - Component factory (*Home bean*), following Abstract Factory pattern
  - Customization possible by metadata and configuration files (deployment descriptors)
- Composition Technique
  - Adaptation/Glue:
    - Distribution (not transparent, see local/remote interfaces)
    - Transparent network protocols
    - Transparent transactions via Containers
    - Transparent persistency via Containers
  - No connectors

Interactions in an EJB Component System (Where are the Beans?)



23.2 The Parts of a Bean Infrastructure

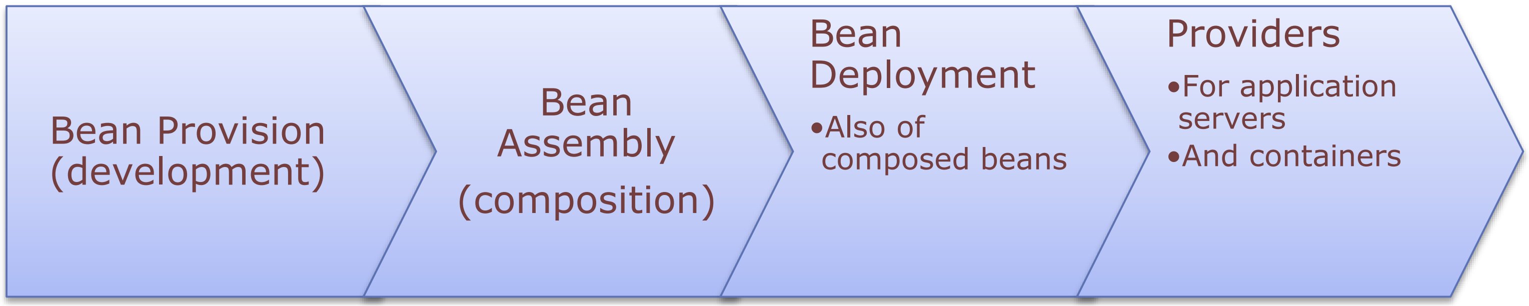
- Container
- Bean class
- Home – a factory
- Remote interface [3.0: annotation]
- Local interface [3.0: annotation]
- Deployment descriptor (2.0)

The Bean Container/Application Server

- The **bean container** is a **run-time façade** for all beans on a server with **infrastructure (application server)**
  - In a container, some business logic may run on the server, hiding the direct data access
  - The container manages the beans with
    - Factory: create bean
    - Repository: find, remove bean
  - The container provides run-time middleware services for the beans
- The bean container is a **deployment infrastructure**
  - The container generates *dynamically* middleware code for the bean when it is deployed on a machine (*implicit middleware*)
    - Bean developer *only* writes business logic and declares the middleware services (transactions, persistence, security, resource management, ...etc) by specifying metadata (annotations)
    - The middleware services are provided automatically by code generation
      - In explicit middleware (e.g., CORBA), middleware services have to be addressed by the programmer

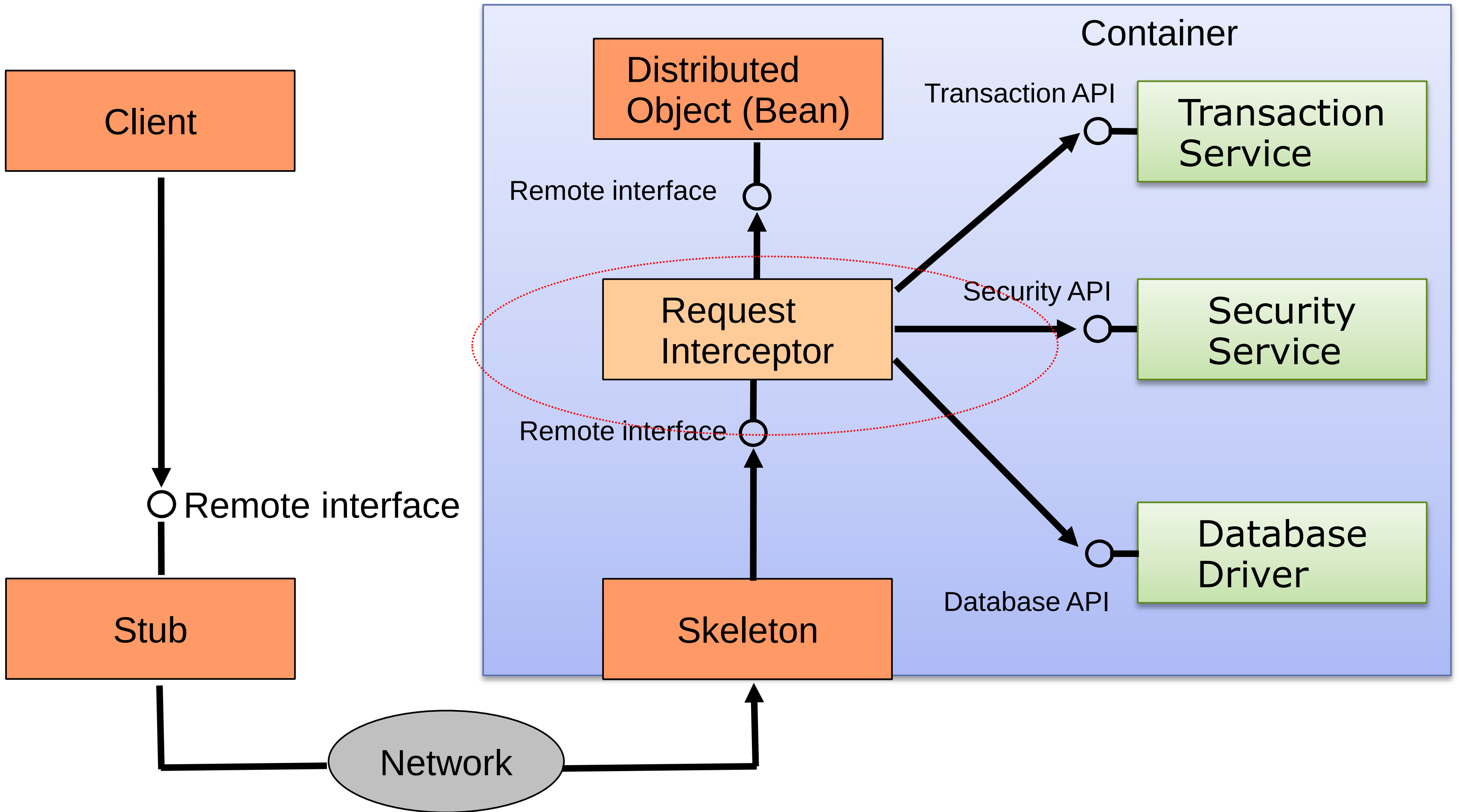
Resulting Roles in the EJB Software Process

- ***Bean provider*** (bean producer, programmer) is an application expert
  - Builds a EJB-jar with application specific methods, deployment-descriptor, remote, home interface
- ***Application assembler*** composes EJB to larger EJB, i.e., applications units.
  - She extends the deployment-descriptors
- ***Bean deployer*** (employer) puts the EJB-jar into a deployment environment, consisting of a EJB Server and Container
  - Preparing the EJB for use, generating middleware code
  - Is the EJB connected to a EJB-Container, it is configured and usable
- ***Server provider*** is a specialist in transaction management and distributed systems.
  - Provides basic functionality for middleware services
- ***Container provider*** delivers the container tools for configuration and for run time inspection of EJB
  - The Container manages persistency of Entity Beans, generation of communication code (glue code) to underlying data bases



Implicit Middleware by Interceptors (Bean Decorators)

- **Interceptors** are special server decorators (server skeletons) treating transparency problems
- Implementations of interceptors can be generated by the container

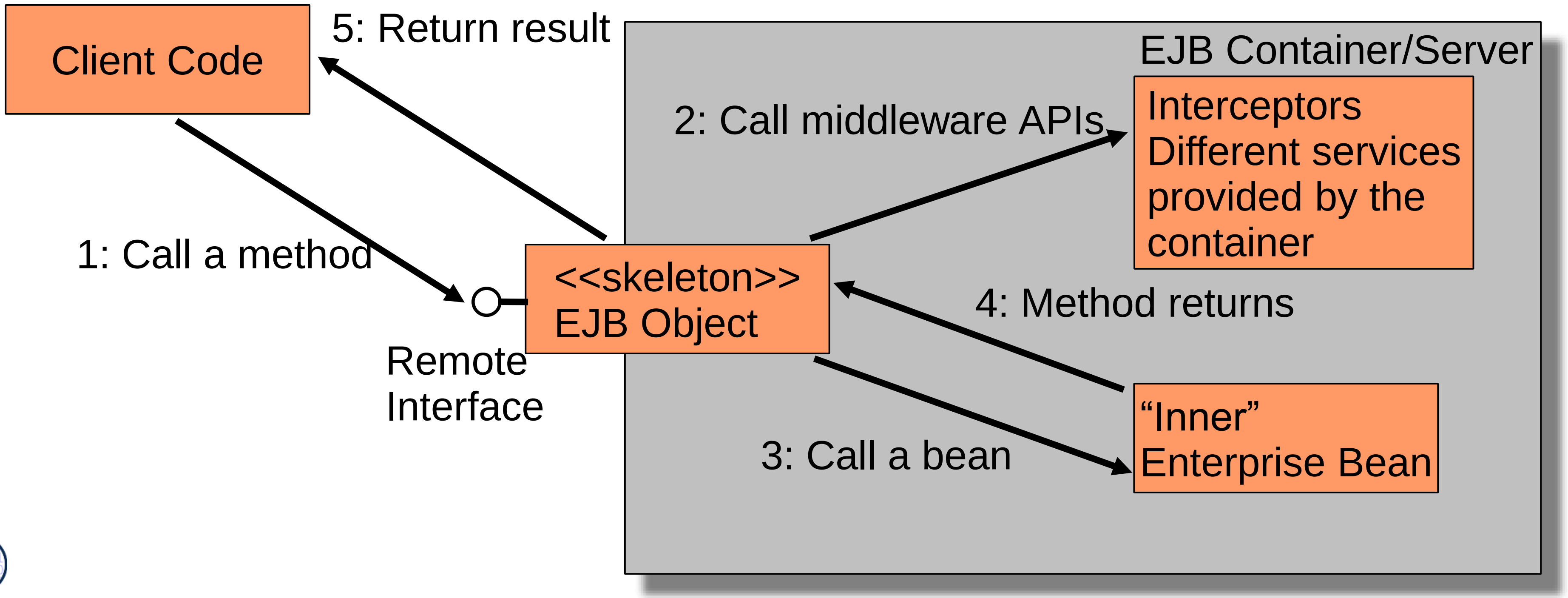


The Parts of an EJB
- The Enterprise Bean Class

- The implementation of the bean looks different depending on which kind of bean
- Three different beans, with different families of interfaces, exist:
- **Session beans**
  - Business-process-related logic, e.g., compute prices, transfer money between accounts (“Business methods”)
  - **Stateless**: call-oriented, runs to completion without interruption
  - **Stateful**: may be interrupted and keep state by functions ejbPassivate(), ejbActivate()
- **Message-driven beans**
  - Message-oriented logic, e.g., receive a message and call a session bean
- **Entity beans**
  - Data-related logic, e.g., change name of a customer, withdraw money from an account

The Parts of an EJB
- The EJB Object as a Skeleton

- The EJB is not called directly, but via an EJB object (skeleton, facade object, proxy)
  - whose implementation is generated by the container
  - It filters the input and intercepts calls and delegates them to the inner bean
    - Interceptors can be generated by the container
    - The EJB object is responsible for providing middleware services



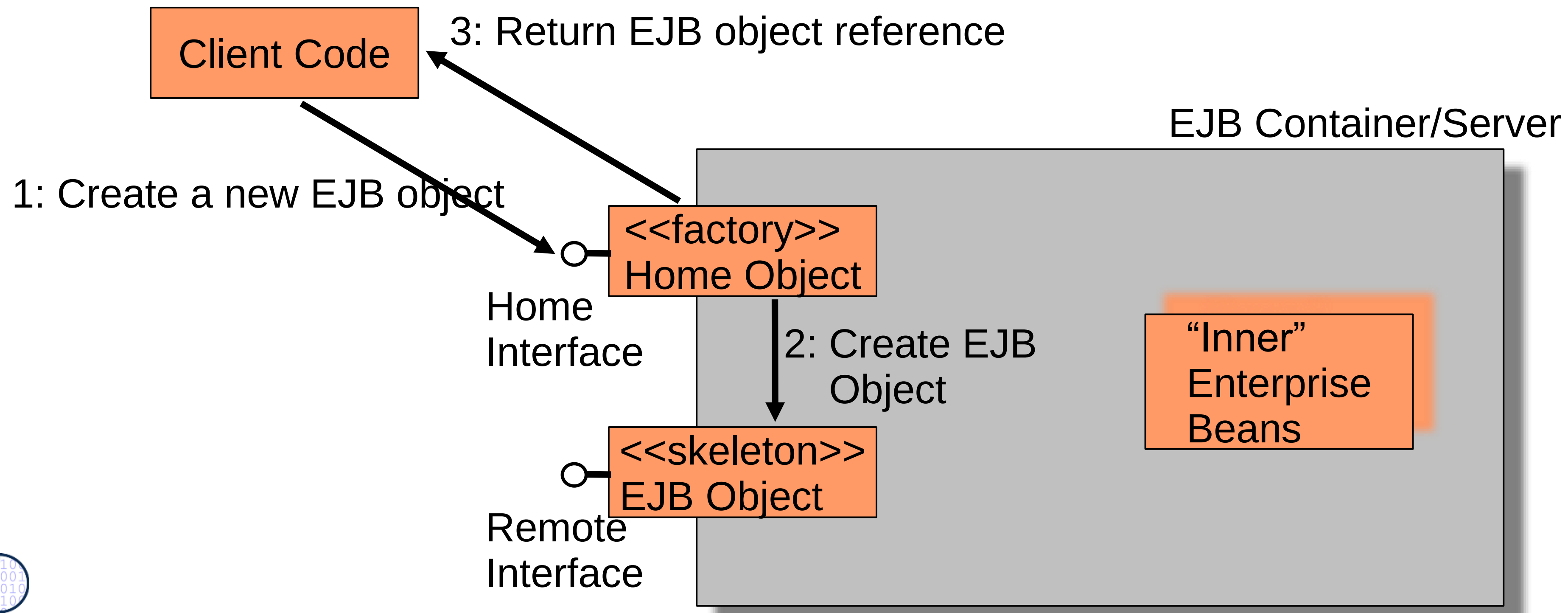
The Parts of an EJB
- The Remote Object Interface

- The interface to the bean that the client sees from remote
  - Must contain all methods the bean should expose
  - As the EJB object lies between the client and the bean, it has to implement this interface
  - Must extend **`javax.ejb.EJBObject`**

```
public interface Bank extends javax.ejb.EJBObject {

        // Bean business methods
public Account getAccount(String name)
        throws java.rmi.RemoteException;

public void openAccount(String name)
        throws java.rmi.RemoteException;
}
```

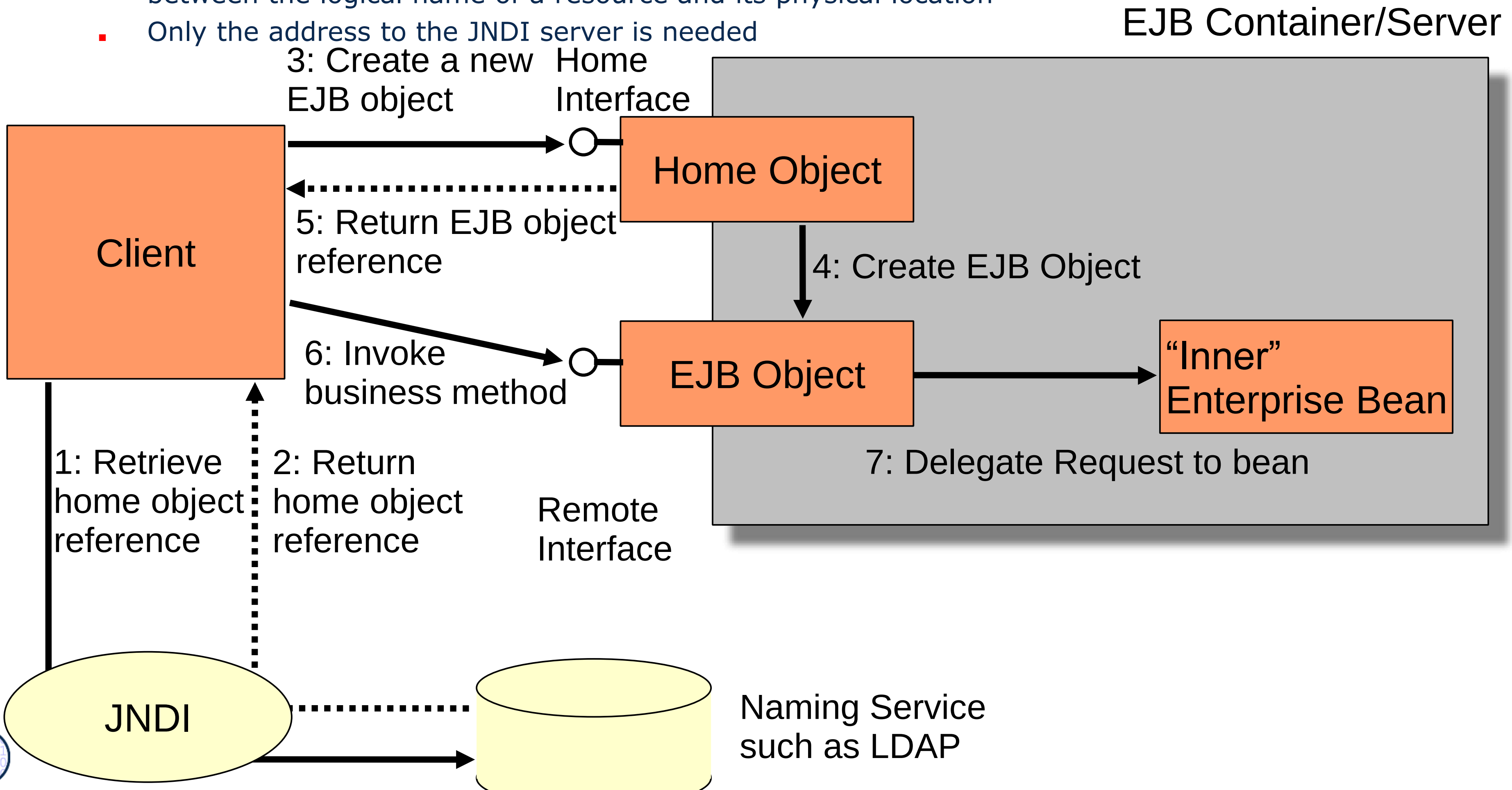
The Parts of an EJB - The Home Object and Interfaces

- An EJB object *factory* and *repository* is needed: The home object with the *home interface*
  - Defines methods for creating, finding and removing EJB objects
- The communication uses Java RMI over IIOP
  - If an argument is serializable, it is sent as pass-by-value
  - RMI can also simulate pass-by-reference
  - A serialized stub for the remote object is sent instead



Name Service for Name Transparency

- The Java Naming and Directory Interface (JNDI) is used to lookup home objects
  - JNDI is a standard interface for locating resources (name service), providing a mapping between the logical name of a resource and its physical location
  - Only the address to the JNDI server is needed


EJB Container/Server
3: Create a new EJB object
Home Interface
Client
Home Object
5: Return EJB object reference
4: Create EJB Object
6: Invoke business method
EJB Object
"Inner" Enterprise Bean
7: Delegate Request to bean
1: Retrieve home object reference
2: Return home object reference
Remote Interface
JNDI
Naming Service such as LDAP

The Parts of an EJB - Local Interfaces

- Beans do not support location transparency
  - For a local call, you must provide local interfaces
    - local interface corresponding to remote interface
    - local home interface corresponding to home interface
  - To switch between local and remote calls it is necessary to change the code
- Horrible: this should be encapsulated in a connector!

Remote:

- Client calls a local stub
- Marshalling
- Stub calls skeleton over a network connection
- Unmarshalling
- EJB object is called, performs middleware services
- Bean is called
- Repeat to return result

Local:

- Client calls a local object
- Local object performs middleware services
- Bean is called
- Control is returned to the client

The Parts of an EJB – Putting Together an EJB Component File

- All the above mentioned files are put into an **EJB-jar** file (Java archive, zipped)
  - bean class
  - home (and local home) interface
  - remote (and local) interface
  - (possibly vendor-specific files)
- Additionally in EJB 2.0:
  - Deployment descriptor, i.e., the specification for the implicit middleware and the composition of beans

Deployment of an EJB Component File

- The ***deployment*** of an EJB is a new step in component systems we have not yet seen

- Deployment: The application server is notified of the new EJB component file by
  - using a command-line tool,
  - dropping the EJB file in a specific directory,
  - or in some other way
- The EJB-jar file is verified by the container
- The container generates an EJB object and home object
- The container generates any necessary further RMI stubs, skeletons, and interceptors

23.3 A Closer Look at the Different Kinds of Enterprise JavaBeans

Session Beans Overview

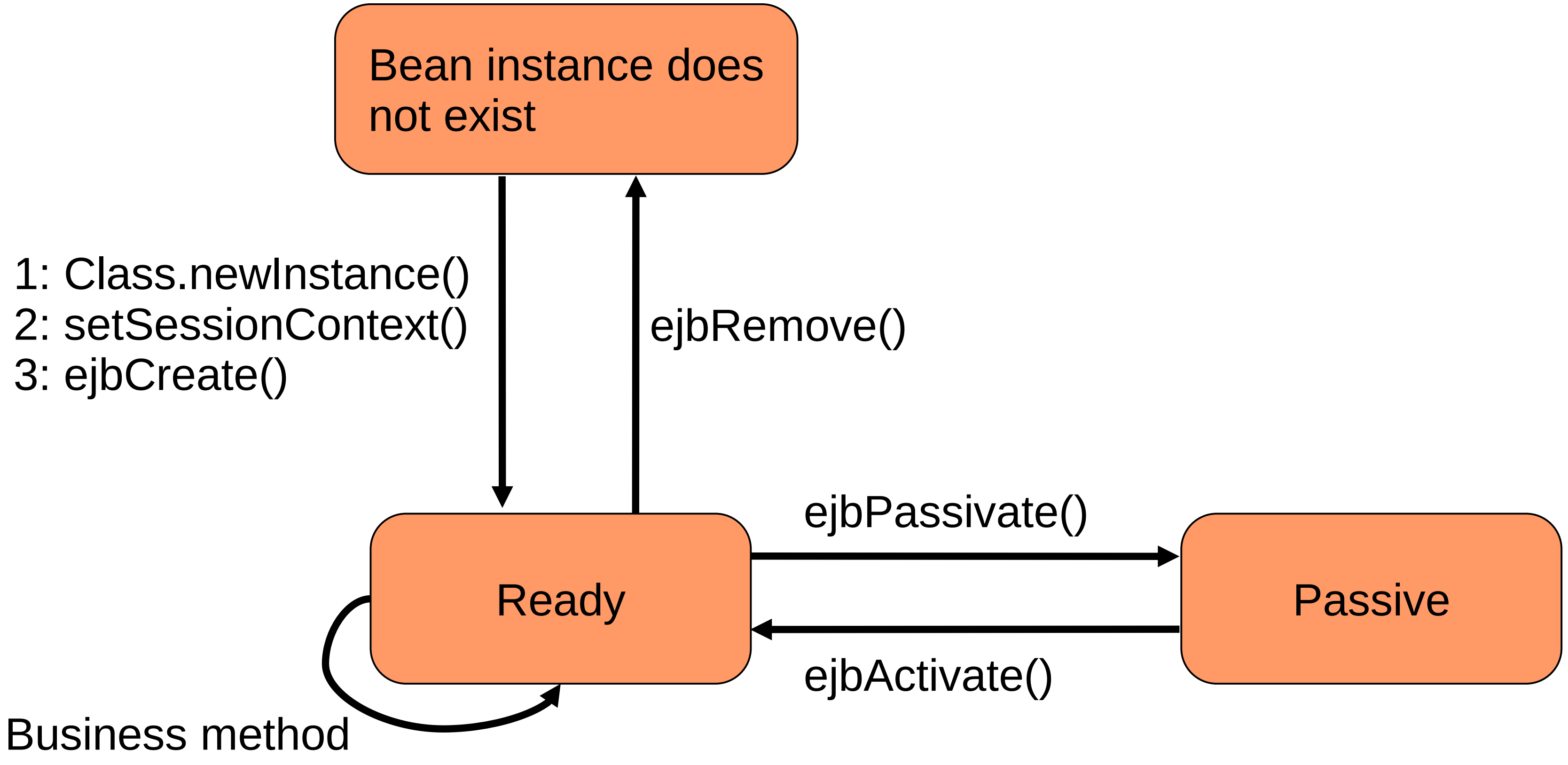
- Reusable components that contain logic for business processes
  - The lifetime of a session bean is roughly equivalent to the lifetime of the client code calling it
  - A session bean is nonpersistent

java.ejb.SessionBean

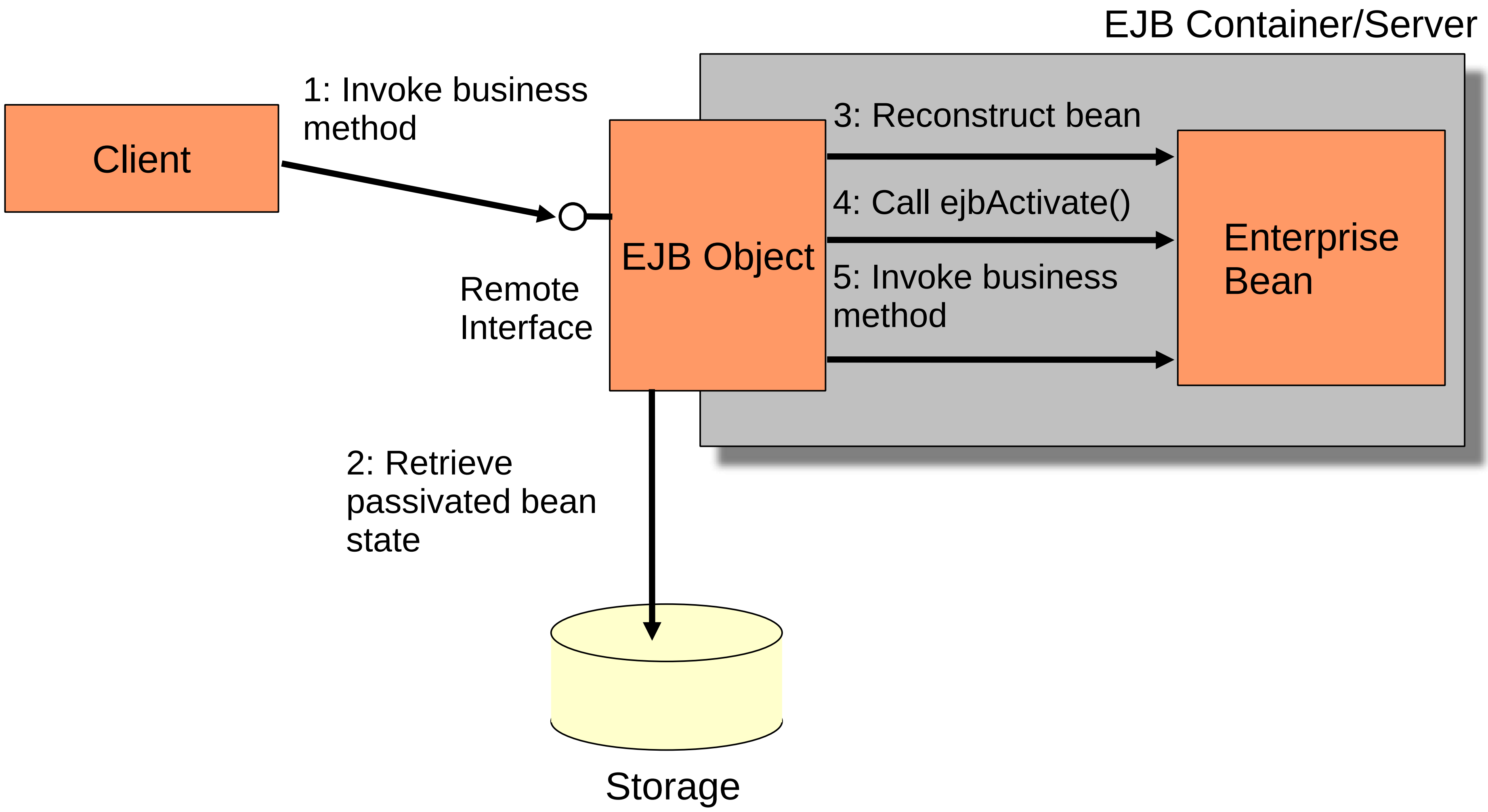
- **`setSessionContext(SessionContext context)`**
  - The bean can query the SessionContext for information concerning the container
- **`ejbCreate()`**
  - Used to perform initialization when the bean is created
- **`ejbPassivate()`**
  - Used by stateful session beans, explained later
- **`ejbActivate()`**
  - Used by stateful session beans, explained later
- **`ejbRemove()`**
  - Used to release any resources the bean has been holding before it is removed

Life Cycle of a Stateful Session Bean

- Handles state-based conversations with users
  - E-commerce web store with a shopping cart
  - Online bank
  - Tax declaration



Activation of a Stateful Session Bean



Characteristics of Message-Driven Beans (MDB)

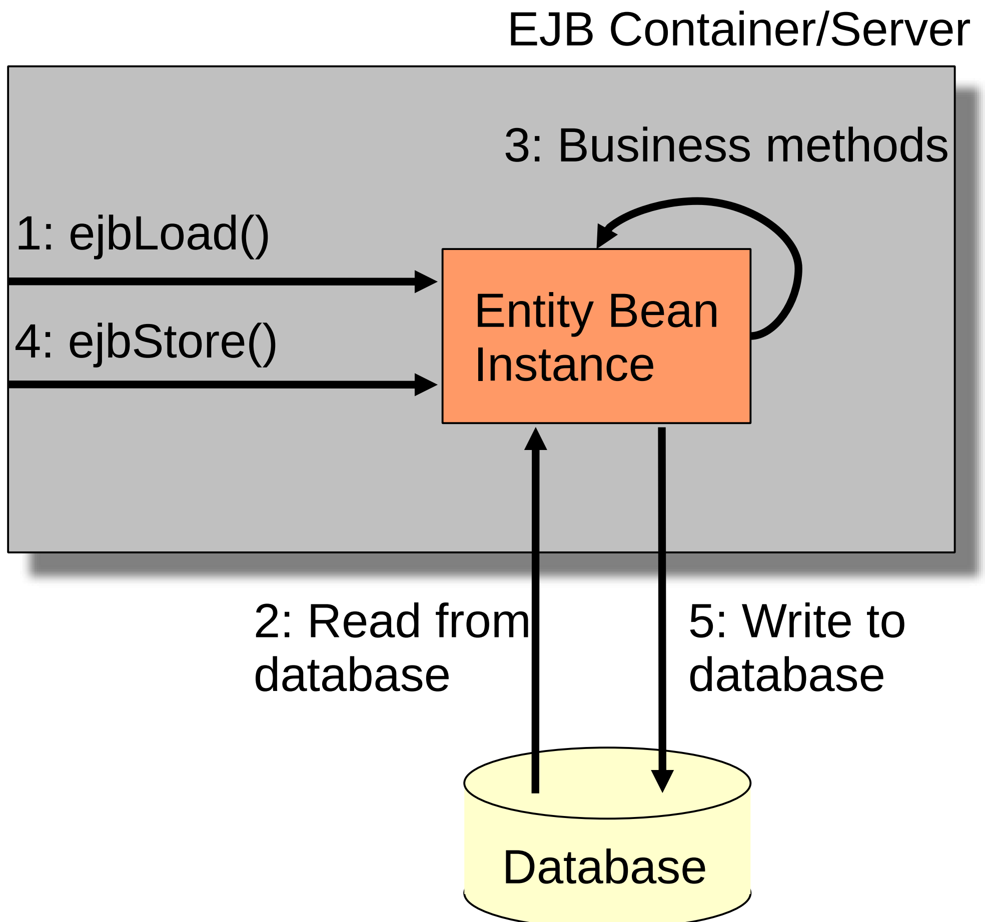
- MDBs are also stateless
  - MDBs don't have a home, local home, remote or local interface
- MDBs have a single, weakly typed business method
  - onMessage() is used to process messages
  - MDBs don't have any return values
  - However, it is possible to send a response to the client
  - MDBs cannot send exceptions back to clients
- MDBs can be durable or nondurable subscribers
  - durable means that the subscriber receives all messages, even if it is inactive
- Why use MDB instead of Session Beans?
  - **Asynchronous processing** means that clients don't have to wait for the bean to finish
  - Reliability
    - With RMI-IIOP the server has to be up when the client is calling it.
    - With a message-oriented middleware (MOM) that supports guaranteed delivery, the message is delivered when the server gets back online
  - Support for subscription of multiple senders and receivers
    - RMI-IIOP is limited to one client talking to one server

Overview of Entity Beans

- An **entity bean** is a persistent material
  - It consists of the same files as a session bean
- Object-relational mapping necessary (from Java classes to relational databases)
  - Map the object to a relational database when it is stored
  - Queries possible by using an special EJB query language (EJB-QL) that is translated to specific query languages of relational databases
  - The mapping is either hand-coded or achieved by finished products
- Several entity bean instances may represent the same underlying data
  - An entity bean has a primary key to uniquely identify the database data
  - Entity bean instances can be put to database by ejbStore() and ejbLoad()
- Two kinds of entity beans
  - *Bean-managed persistent* or *container-managed persistent*

Loading and Storing an Entity Bean

- Entity beans are persistent objects that can be stored in permanent storage
  - Live on the entity or database layer of the 3-tier architecture
  - The entity bean data is the physical set of data stored in the database



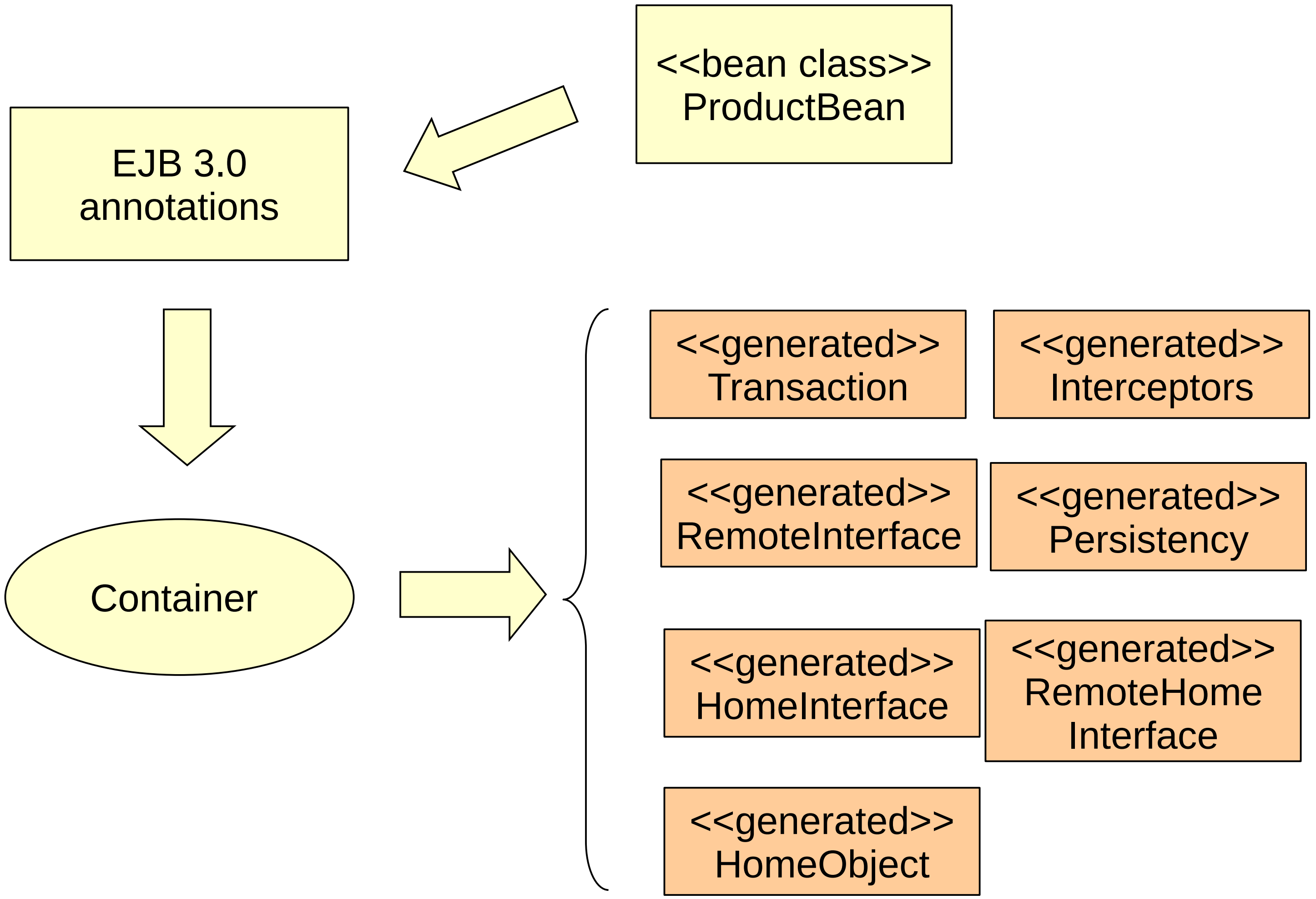
23.4. Generation of Implicit Middleware in EJB 3.X

•EJB heavily use metadata markup to generate all dependent middleware interfaces and code

- Persistency
- Transactions

EJB 3.0

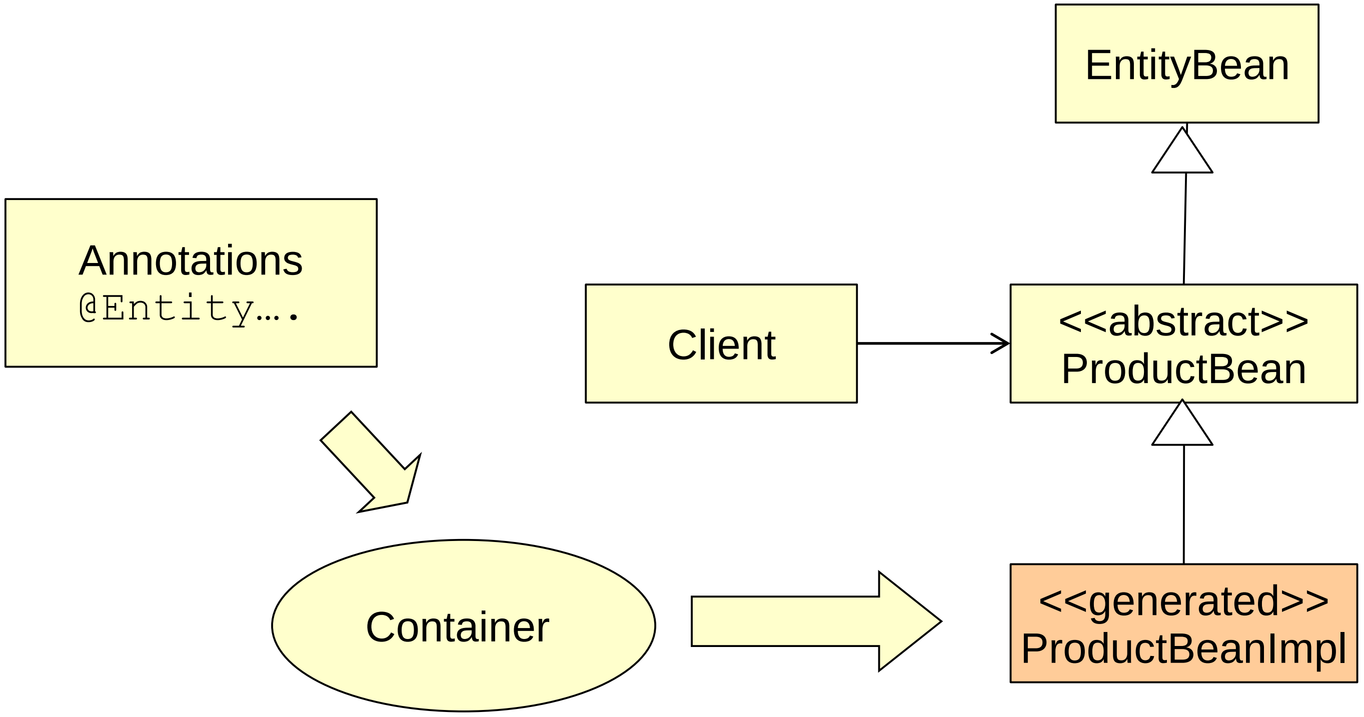
- Only the bean class is specified
  - Rest of the classes is generated from metadata annotations



<<bean class>> ProductBean
EJB 3.0 annotations
Container
<<generated>> Transaction
<<generated>> Interceptors
<<generated>> RemoteInterface
<<generated>> Persistency
<<generated>> HomeInterface
<<generated>> RemoteHome Interface
<<generated>> HomeObject

Persistency is Container-Managed in 3.0

- *TemplateMethod* design pattern with generated hook class implementation
- The container performs the storage operations
  - The container generates the persistence (ProductBeanImpl) and does the run-time service
- The CMP entity bean is always abstract (ProductBean)
  - The container generates a concrete subclass (ProductBeanImpl)
  - An abstract persistence schema is declared in the deployment descriptor so the container will know what to generate



Metadata Annotations in EJB 3.0 – Annotation Types

- Bean class annotations refer to classes and create interfaces with adapters:

```
@Entity
public class AccountBean implements Account {
   public void deposit (double money) {...}
}

@Stateless
@Stateful
@MessageDriven

// adding interfaces for beans
@Local
@Remote
@RemoteHome
@LocalHome
```

From [EJB 3.0 Features]

Method Callback Annotations

- The default methods can be adorned with user-written *filters (before, after,* and *around fragments (advices))*
- Filter methods are part of Interceptor objects

```
@PrePassivate
void signalPassivation() {
  System.out.writeln("passivating bean now...");
}
```

```
@PreDestroy
@PrePersist
@PostPersist
@PreActivate
@PostActivate
@PrePassivate
@PostPassivate
@CallbackListener
```

```
[from EJB 3.0 Features]

/* Callback method defined inside a Listener class*/

public class AccountListener{

  @PostPersist
  insertAccountDetails(AccountDetails accountDetails){}

}
```

Custom Interceptors

```
[from EJB 3.0 Features]
// Provides profiling logic in a business method (with interceptors)
/* The interceptor class */
public class ProfilingInterceptor {
  @AroundInvoke  // indicates that this is the interceptor method
  public Object profile(InvocationContext invocation) throws Exception {
    long start = System.currentTimeMillis();
    try {
      return invocation.proceed(); // this statement would call the withdraw method
    } finally  {
      long time = start - System.currentTimeMillis();
      Method method = invocation.getMethod();
      System.out.println(method.toString() + "took" + time + " (ms)");
} } }
/* The bean class */
@Stateless
public class BankAccountBean implements BankAccount {
  @PersistenceContext EntityManager entityManager;
  @Interceptors(ProfilingInterceptor.class)
  public void withdraw(int acct, double amount) {  … }
  public void deposit(int acct, double amount) {  …   }
}
```

Prologue
(Down action
of recursion)

Epilogue
(up action
of recursion)

Transaction Control with Metadata Attributes

- Classes and methods may receive transaction attributes
  - **Required**: bean joins the client's transaction, otherwise signals error
  - **RequiresNew**: bean starts new transaction
  - **NotSupported**: interrupt transaction, execute without it
  - **Supported**: bean joins the client's transaction, otherwise executes without transaction

```
[The Java 2 EE tutorial]
@TransactionAttribute(NOT_SUPPORTED)
@Stateful
public class TransactionBean implements Transaction {
...
    @TransactionAttribute(REQUIRES_NEW)
    public void firstMethod() {...}
    @TransactionAttribute(REQUIRED)
    public void secondMethod() {...}
    public void thirdMethod() {...}
    public void fourthMethod() {...}
}
```

23.5 Evaluation of EJB

- as composition system

Component Model

- Mechanisms for secrets and transparency: very good
  - Interface and implementation repository
  - Location, transaction, persistence transparency
  - Life-time of service hidden, states hidden
  - Deployment-time generation of implicit middleware code
  - Communication protocol can be replaced (RMI-IIOP, CORBA-IIOP)
- Parameterization by metadata annotations
  - The services to use are specified: transaction protocol, filters
- Deployment of EJB supported
  - Code generation of stubs
- Standardization: de-facto standard in the Java world
  - Good tutorials
  - Technical vs. application specific vs. business components

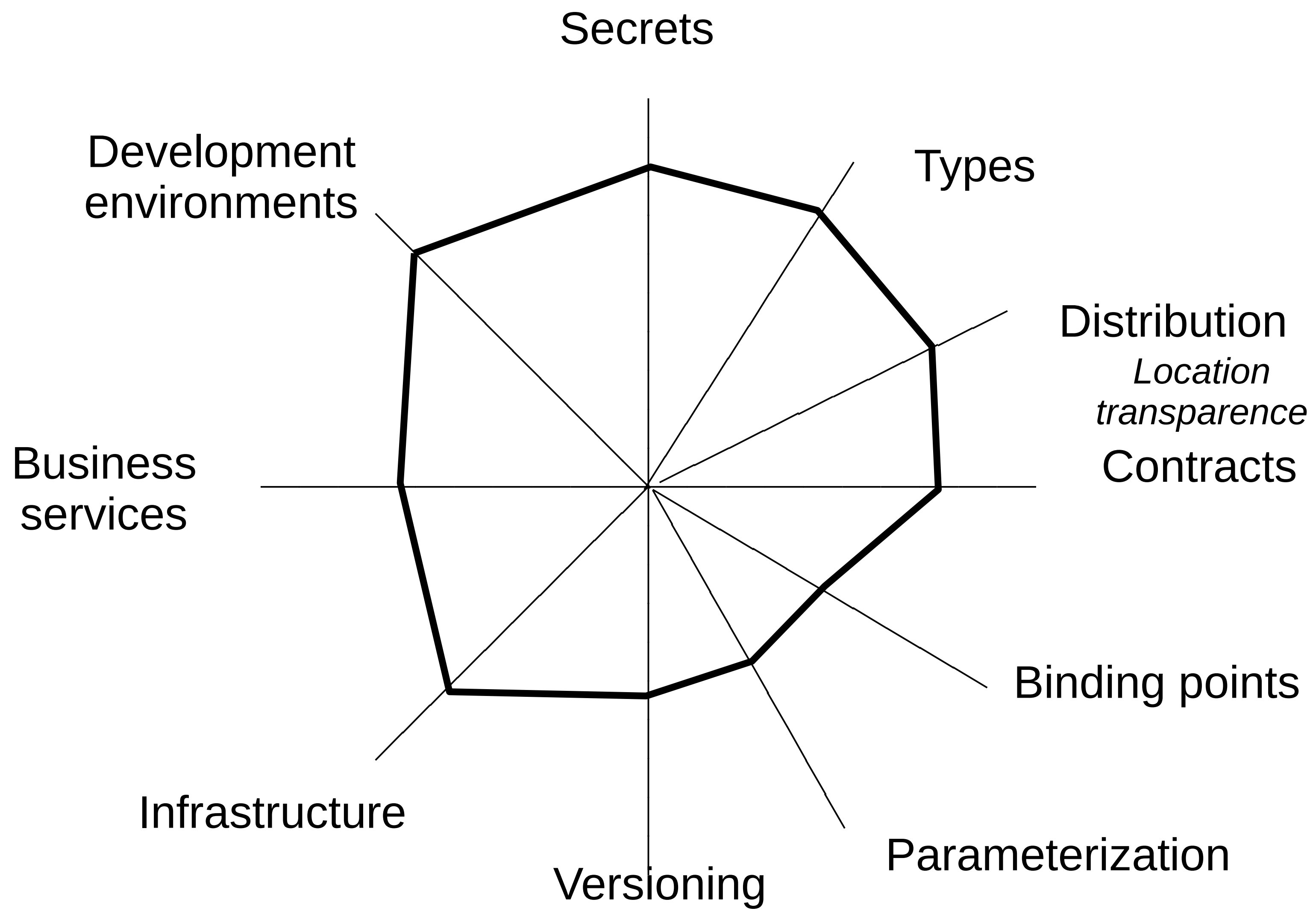
Composition Technique

- Mechanisms for connection
  - Mechanisms for locating
    - JNDI
  - Mechanisms for adaptation
    - Interceptors (server-side skeletons)
  - Mechanisms for glueing
    - Container producing glue code
- Mechanisms for aspect separation
  - Middleware services declared in the deployment descriptor
- Mechanisms for meta-modeling
  - with Java reflection and metadata annotations
- Scalability
  - Pooling ensures scaling

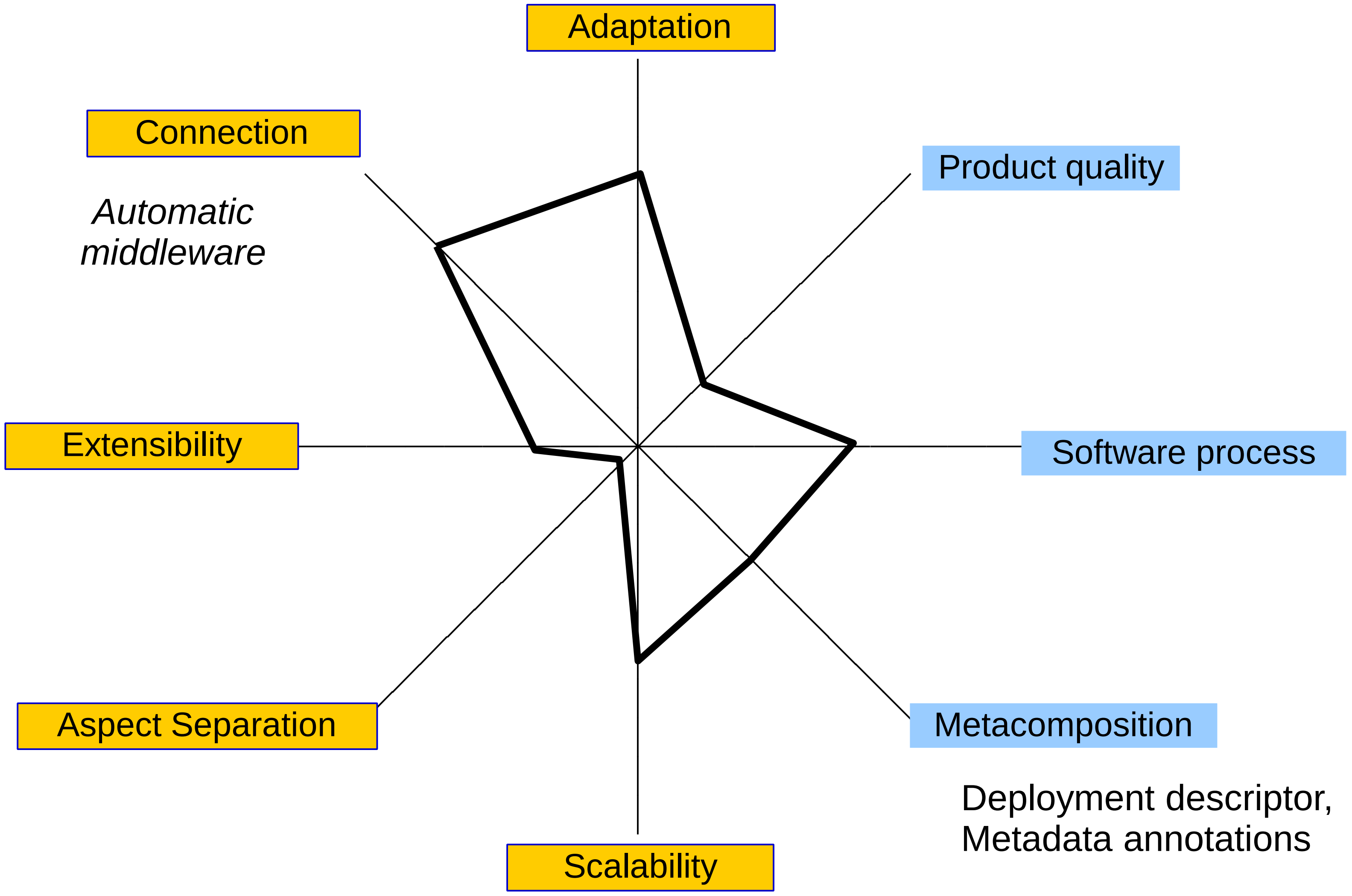
Composition Language

- The deployment descriptor language of EJB 2.0 is a simple composition language
- Limited:
  - Glue code is provided by the container
  - Services can be added/removed/modified by changing the deployment descriptor
  - CMP entity beans can be customized by changing the deployment descriptor

EJB - Component Model



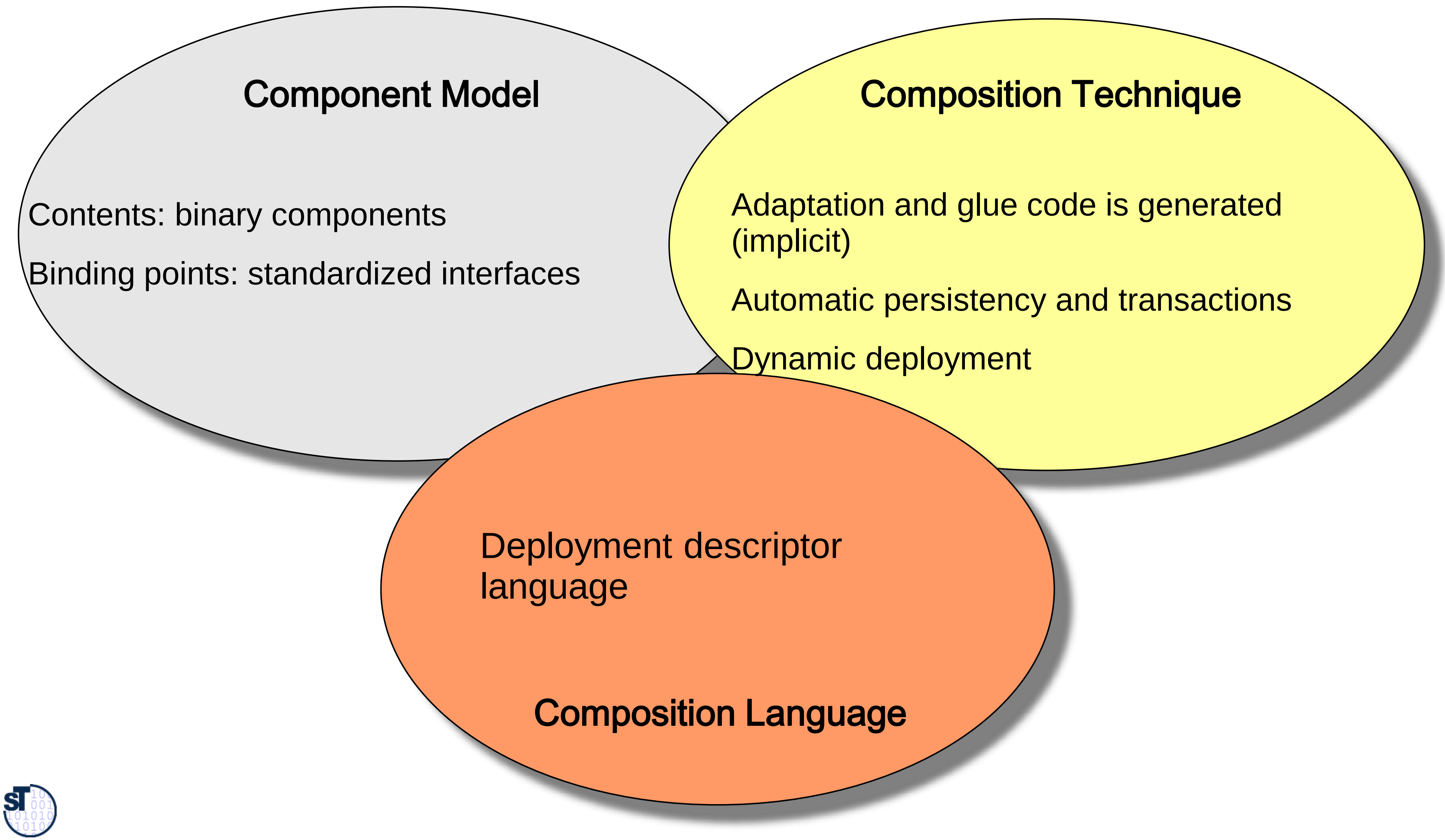
EJB – Composition Technique and Language



Automatic middleware

Deployment descriptor,
Metadata annotations

EJB as Composition Systems



Component Model

Contents: binary components

Binding points: standardized interfaces

Composition Technique

Adaptation and glue code is generated (implicit)

Automatic persistency and transactions

Dynamic deployment

Deployment descriptor language

Composition Language

The End - What Have We Learned

- EJB is big, not for everything
  - Allows the developer to focus on business logic
  - Provides very useful services, like transparency, persistence, security, networking independence, etc.
  - Can interoperate with CORBA
- It is a well-defined standard by Oracle
- It works in symbiosis with several other APIs
  - JNDI, RMI, JDBC, JMS, etc

Appendix: The Parts of an EJB - The Deployment Descriptor (EJB 2.0)

- An XML file in which the middleware service requirements are declared (There is a DD-DTD)
  - Bean management and lifecycle requirements
  - Transaction, persistence, and security requirements
- Composition of beans (references to other beans)
  - Names: Name, class, home interface name, remote-interface name, class of the primary key
  - States: type (session, entity, message), state, transaction state, persistency management - how?
- The application assembler may allocate or modify additional different information
  - Name, environments values, description forms
  - Binding of open references to other EJB
  - Transaction attributes

Example of a Deployment Descriptor

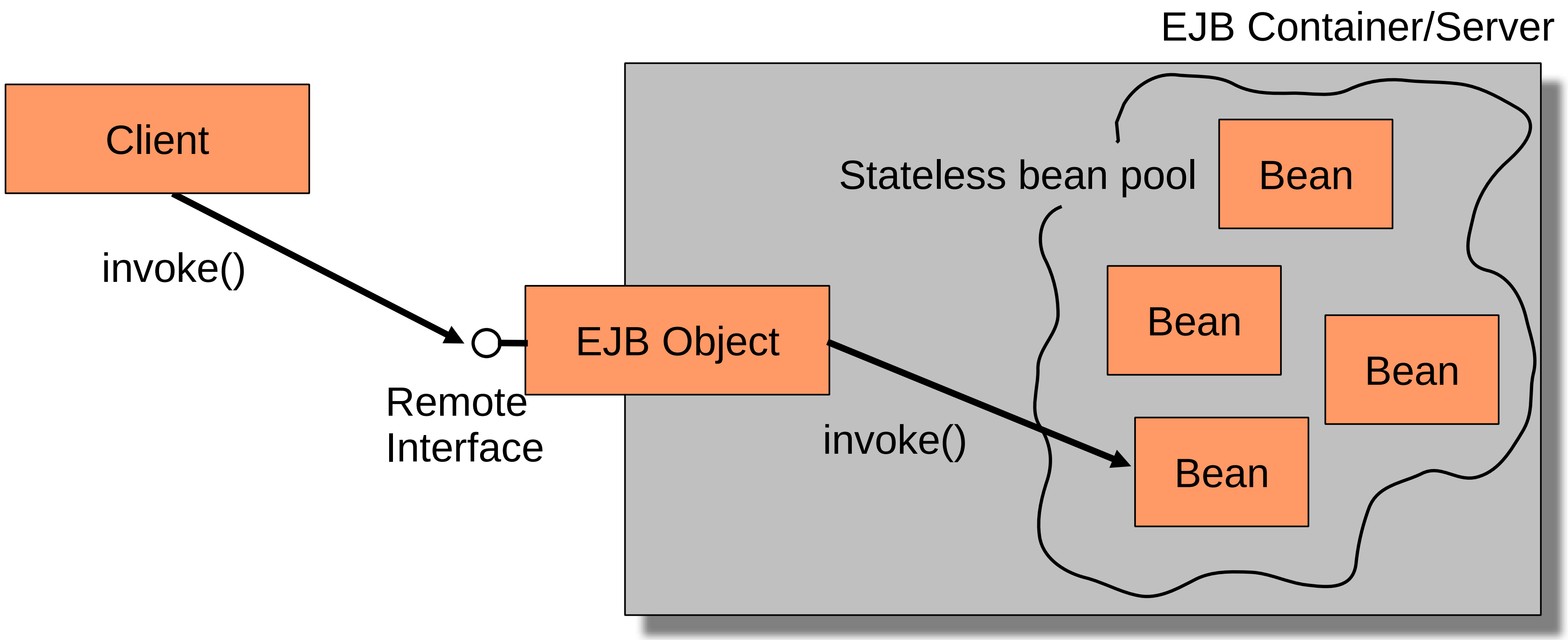
```
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise
JavaBeans 2.0//EN" "http://java.sun.com/dtd/ejb-jar_2_0.dtd">

<ejb-jar>
  <enterprise-beans>
    <session>
      <ejb-name>Bank</ejb-name>
      <home>com.somedomain.BankHome</home>
      <remote>com.somedomain.Bank</remote>
      <local-home>com.somedomain.BankLocalHome</local-home>
      <local>com.somedomain.BankLocal</local>
      <ejb-class>com.somedomain.BankBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
    </session>
  </enterprise-beans>
</ejb-jar>
```

Stateless Session Beans

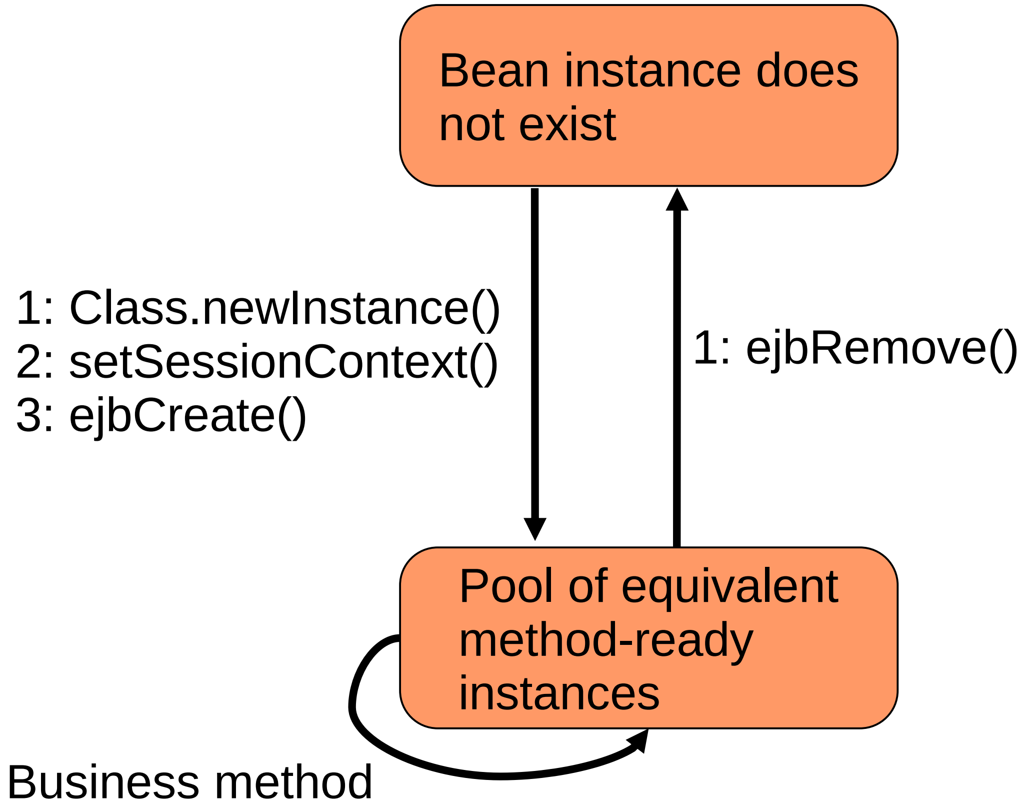
- Handle single requests
  - Conversations that span a single method call
  - Does not hold a conversational state
- The bean may be destroyed by the container after a call or it has to be cleared of old information
- Examples of stateless session beans
  - A user verification service
  - An encoding engine
  - Any service that given some input always produces the same result

Pooling Stateless Session Beans

- Stateless session beans can easily be pooled (reused) to allow better scaling
  - They contain no state

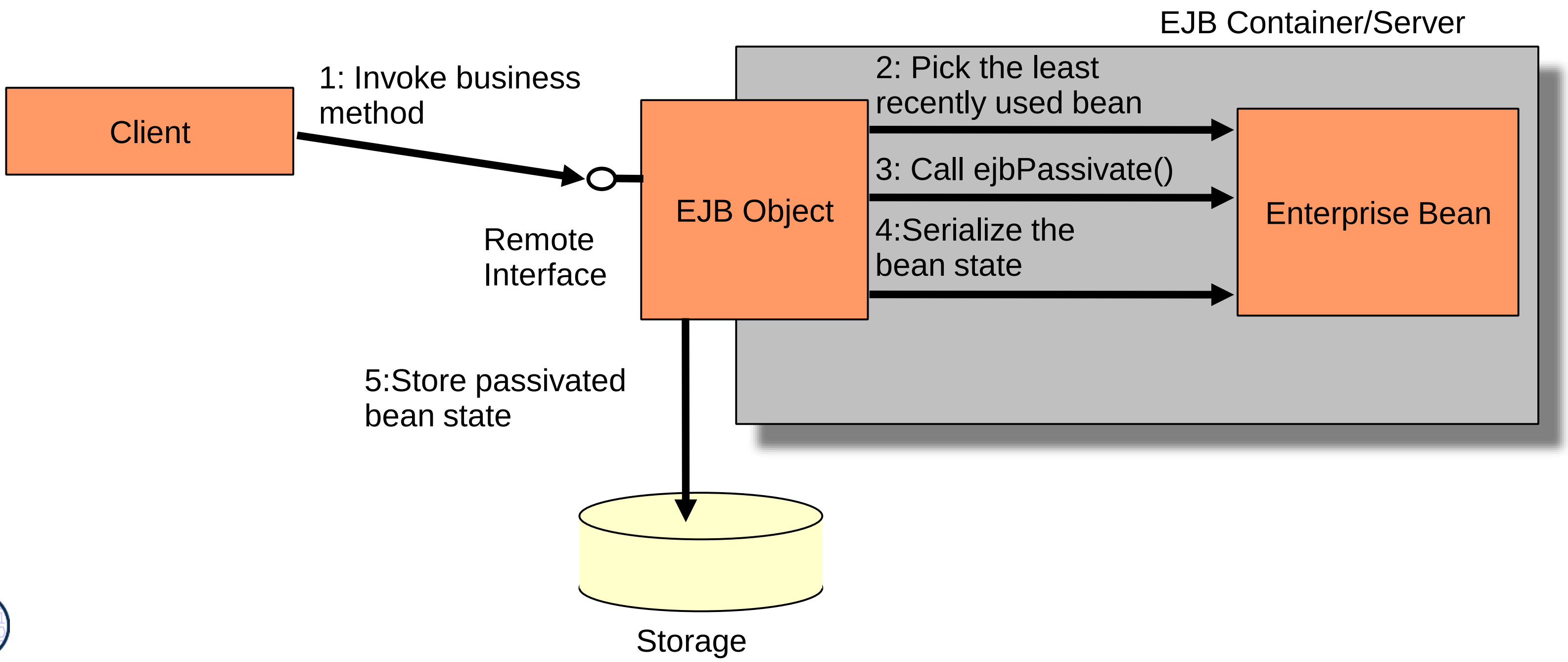


Life Cycle of a Stateless Session Bean

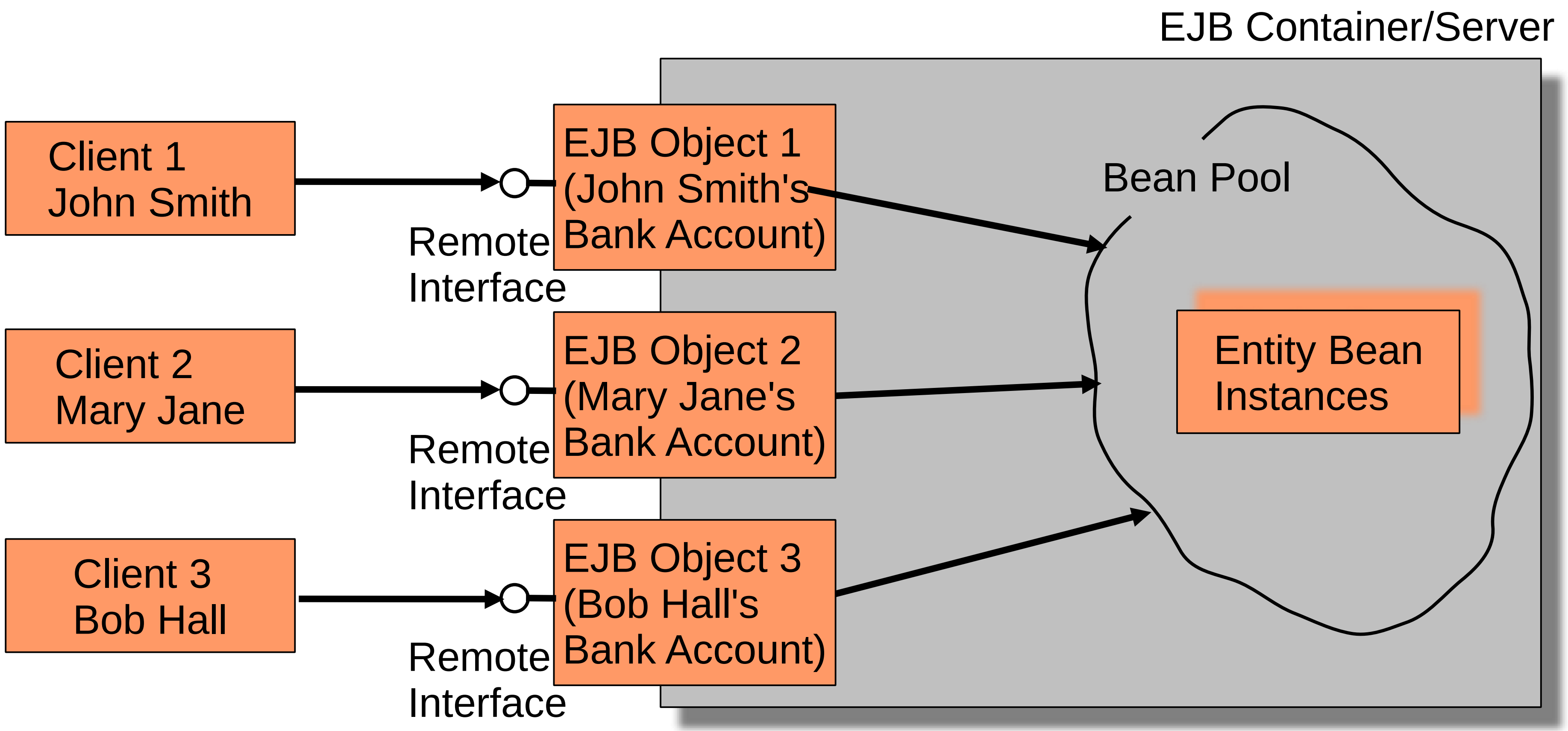


Pooling Stateful Session Beans

- Pooling becomes more complicated
  - Beans must be swapped from physical memory to disk
- A stateful session bean has to implement:
  - ejbPassivate(): Called to let the bean release any resources it holds before it gets swapped out
  - ejbActivate(): Called right after the bean has been swapped in to let it acquire the resources it needs


EJB Container/Server
Client
1: Invoke business method
Remote Interface
EJB Object
2: Pick the least recently used bean
3: Call ejbPassivate()
4:Serialize the bean state
Enterprise Bean
5:Store passivated bean state
Storage

Pooling Entity Beans



Bean-Managed Persistent Entity Beans (BMP Beans)

- The developer is required to provide the implementation to map the instances to and from storage
  - Java Database Connectivity (JDBC)
- BMP beans have to implement **`javax.ejb.EntityBean`**:
  - **`setEntityContext(javax.ejb.EntityContext)`**
    - The context can be queried of information regarding the container
  - **`unsetEntityContext()`**
  - **`ejbRemove()`**
    - Removes the data from the persistent storage
  - **`ejbActivate()`**
    - Lets the bean allocate resources after being swapped in
  - **`ejbPassivate()`**
    - Called before the bean is swapped out so it can release resources
  - **`ejbLoad()`**
    - Loads database data into the bean
  - **`ejbStore()`**
    - Stores the data in the bean to the database

Bean-Managed Persistent Entity Beans

- BMP beans also have to other kinds of methods relating to storage
  - **`ejbCreate()`**
    - Used to create new entries in the database (optional)
  - Finder methods
    - **`ejbFindXXX()`**
    - Must have at least one: **`ejbFindByPrimaryKey()`**
    - Normally contains database queries
      - e.g., **`SELECT id FROM accounts WHERE balance > 3000`**
  - **`ejbHomeXXX()`** methods
    - Performs simple services over a set of beans
- A BMP entity bean consists of
  - Bean-managed state fields, persistable fields that are loaded from the database
  - Business logic methods: Performs services for clients
  - EJB-required methods: Required methods that the container calls to manage the bean

Example
- Bean-Managed State Fields

- AccountBean.java

```
import java.sql.*;
import javax.naming.*;
import javax.ejb.*;
import java.util.*;

public class AccountBean implements EntityBean {
  protected EntityContext context;

  // Bean-managed state fields
  private String accountID;
  private String ownerName;
  private double balance;

  public AccountBean() { }

  ...
```

```
public void deposit(double amount) {
  balance += amount;
}

public void withdraw(double amount {
  if (amount < balance) {
    balance -= amount;
  }
}

public void getBalance() {
  return balance;
}
```

Example
- Business Logic Methods

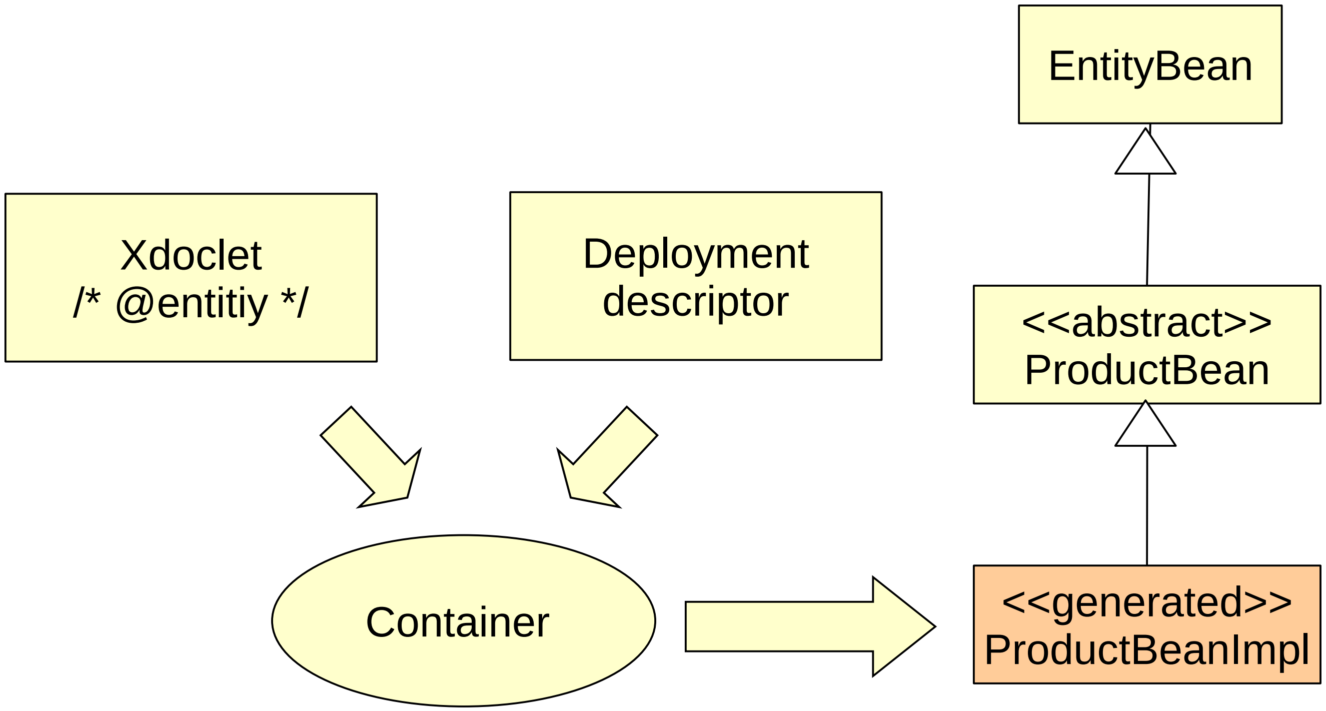
```
...cont...
public void ejbHomeGetTotalBankValue() {
  PreparedStatement pStatement = null;
  Connection connection = null;
  try {
    connection = getConnection();
    pStatement = connection.prepareStatement(
      "select sum(balance) as total from accounts");
    ResultSet rs = pStatement.executeQuery();
    if (rs.next()) { return rs.getDouble("total"); }
  catch (Exception e) { … }
  finally {
    try { if (pStatement != null) pStatement.close(); }
    catch (Exception e) { … }
    try { if (connection != null) connection.close(); }
    catch (Exception e) { … }
  }
}

  ...cont...
```

Example
- Required Methods

```
...cont...
public void ejbRemove {
  PreparedStatement pStatement = null;
  Connection connection = null;
  AccountPK pk = (AccountPK) context.getPrimaryKey();
  String id = pk.accountID;
  try {
    connection = getConnection();
    pStatement = connection.prepareStatement(
      "delete from accounts where id = ?1");
    pStatement.setString(1, id);
    pStatement.executeQuery();
  catch (Exception e) { … }
  finally {
    try { if (pStatement != null) pStatement.close(); }
    catch (Exception e) { … }
    try { if (connection != null) connection.close(); }
    catch (Exception e) { … }
  }  }
...
```

Container-Managed Persistency in 2.0

- TemplateMethod design pattern with generated hook class implementation
- Xdoclet tag comments or deployment descriptor



Container-Managed Persistent Entity Beans (CMB)

- The container performs the storage operations
  - This gives a clean separation between the entity bean and its persistent representation
  - The container generates the persistence logic
- The CMP entity bean is always abstract
  - The container generates a concrete subclass
- The CMP entity beans have no declared fields
  - Also the get/set method implementations are generated by the container from the deployment descriptor
- CMP beans get an abstract persistence schema
  - An abstract persistence schema is declared in the deployment descriptor so the container will know what to generate
- There is a query language, EJB Query Language (EJB-QL)
  - SELECT OBJECT(a) FROM Account AS a WHERE a.balance > ?1

Example: Using the TemplateMethod Pattern in EJB 2.0

```java
import javax.ejb.*;
public abstract class ProductBean implements EntityBean {
  protected EntityContext context;
  public abstract String getName();
  public abstract void setName(String name);
  public abstract String getDescription();
  public abstract void setDescription(String description);
  public abstract double getBasePrice();
  public abstract void setBasePrice(double prise);
  public abstract String getProductID();
  public abstract void setProductID(String productID);

  public void ejbActivate() { }
  public void ejbRemove() { }
  public void ejbPassivate() { }
  public void ejbLoad() { }
  public void ejbStore() { }
  public void setEntityContext(EntityContext ctx) { context = ctx; }
  public void unsetEntityContext() { context = null; }
  public void ejbPostCreate(String productID, String name,
    String description, double basePrice) { }
  public String ejbCreate(String productID, String name,
    String description, double basePrice) {
    setProductID(productID);      setName(name);
    setDescription(description);  setBasePrice(basePrice);
    return productID;
  }
}
```

Hook methods

CMP Entity Beans – Deployment Descriptor

- You have to declare how the container should generate methods and fields

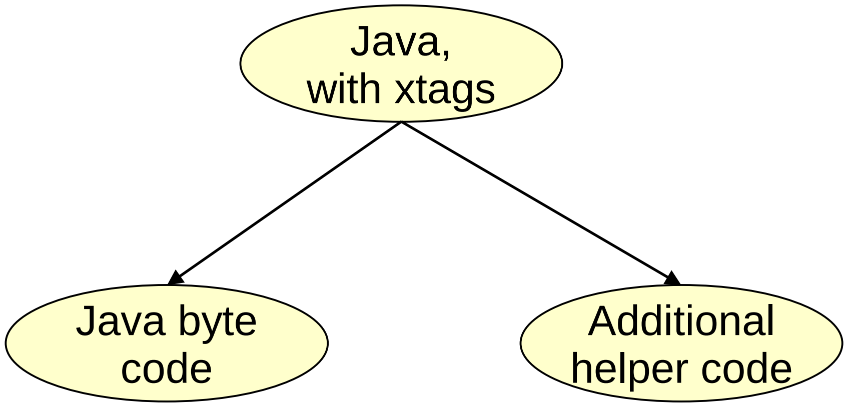
```
....declarations of interfaces, etc ....
  <cmp-field>
    <field-name>productID</field-name>
  </cmp-field>
  <cmp-field>
    <field-name>name</field-name>
  </cmp-field>
  <cmp-field>
    <field-name>description</field-name>
  </cmp-field>
  <cmp-field>
    <field-name>basePrice</field-name>
  </cmp-field>
.. queries ...
  <query>
    <query-method>
      <method-name>findByName</method-name>
      <method-params>
        <method-param>java.lang.String</method-param>
      </method-params>
    </query-method>
    <ejb-ql>
      <![CDATA(SELECT OBJECT(a) FROM ProductBean AS a WHERE name=?1)]>
    </ejb-ql>
  </query>
```

EJB and Others

- Interceptors and Decorators
  - The Interceptor of a bean is like a decorator
  - It can be overwritten and extended from outside the EJB
  - User can write filters for EJB
  - JBoss uses this for aspect-oriented EJB (see later)
- EJB was formed after Microsoft's MTS (now COM+)
  - COM+ is in .NET
  - Models are somewhat similar
- Corba Component Model (CCM) is also similar

XDoclets

- An XDoclet is a plugin into the XDoclet framework
- The XDoclet framework is a doclet, i.e., a Javadoc extension
- XDoclets define new tags (xtags), used for metadata
  - Tags can have attribute lists
  - **`/* @ejb.bean  type = "CMP" name="client" view-type="local" */`**
- Tags steer code generation
  - XDoclet compiler reads the Java source files, evaluates commented tags and generates additional code



Use of Xdoclets in EJB 2.0

- Generation of
  - Deployment descriptors
  - Default interfaces
  - Implementation stubs
- Example [from XDoclet documentation]

```
/** Account
  @see Customer
  @ejb.bean name="bank/Account" type="CMP"
            jndi-name="ejb/bank/Account"
            primkey-field="id"
  @ejb.finder signature="jara.util.collection findAll()"
              unchecked="true"
  @ejb.transaction type="required"
  @ejb.interface remote-class="test.interfaces.Account"
  @version 1.5
*/
```