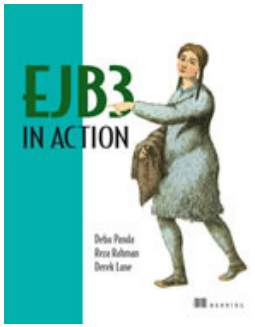
ORACLE



EJB 3 In Action

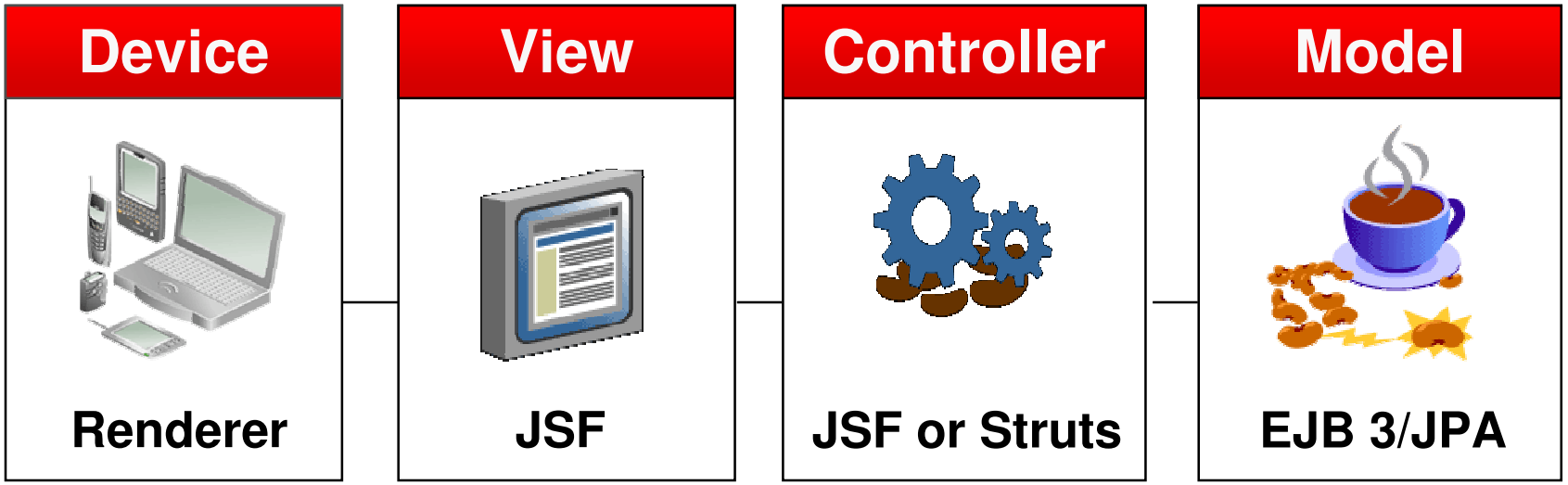
Debu Panda
http://debupanda.com
Author : EJB 3 In Action (http://manning.com/panda)

EJB 2 vs EJB 3

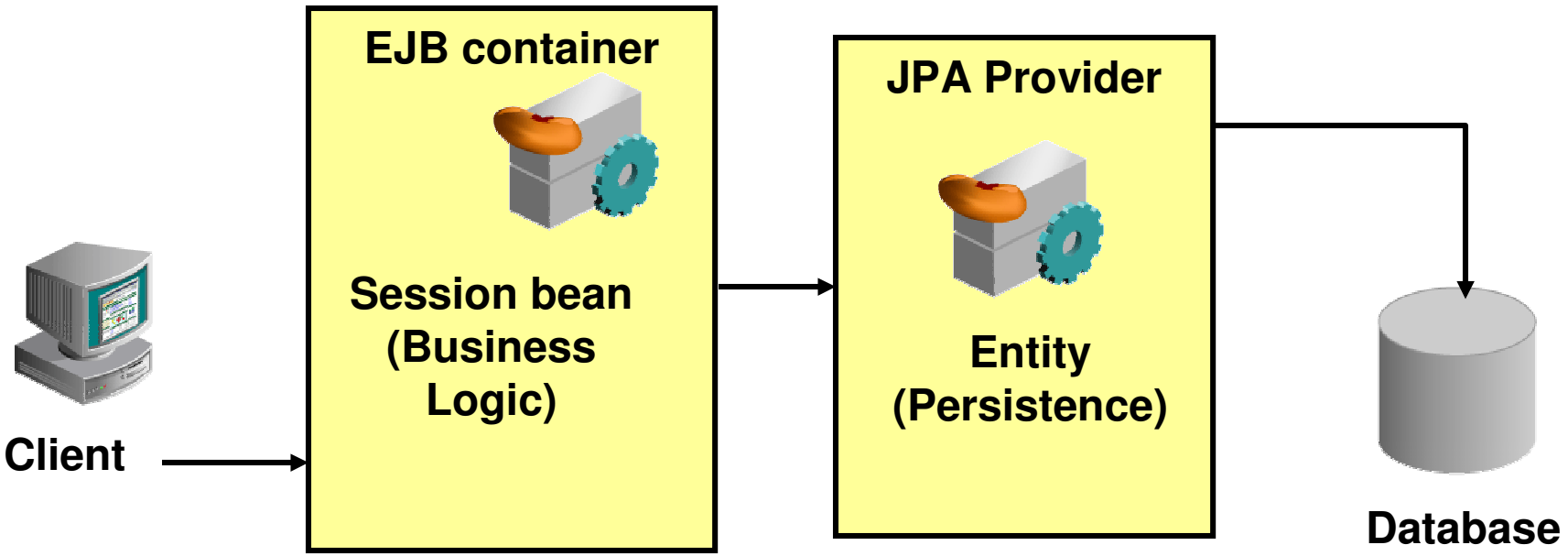
EJB 3 Goals

- Simplify developers life
  - EJB now resembles Plain Java Object (POJO)
  - Use metadata annotations
    - XML descriptors are no longer necessary
    - Default
  - Unnecessary artifacts are optional
  - Simplify client view using dependency injection
- Standardize persistence API for Java platform
  - Based on success of leading ORM solutions
    - Including Oracle TopLink, Hibernate

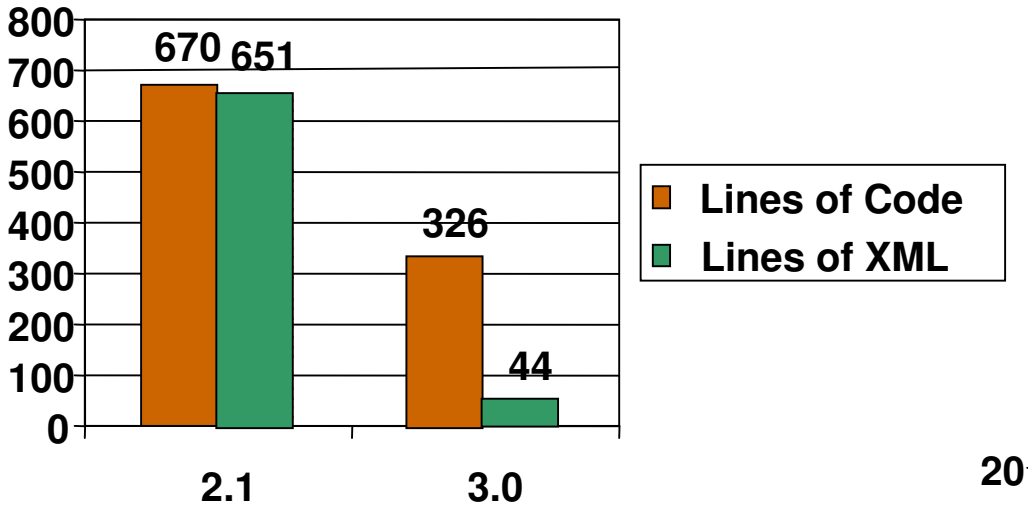
Where EJB fits

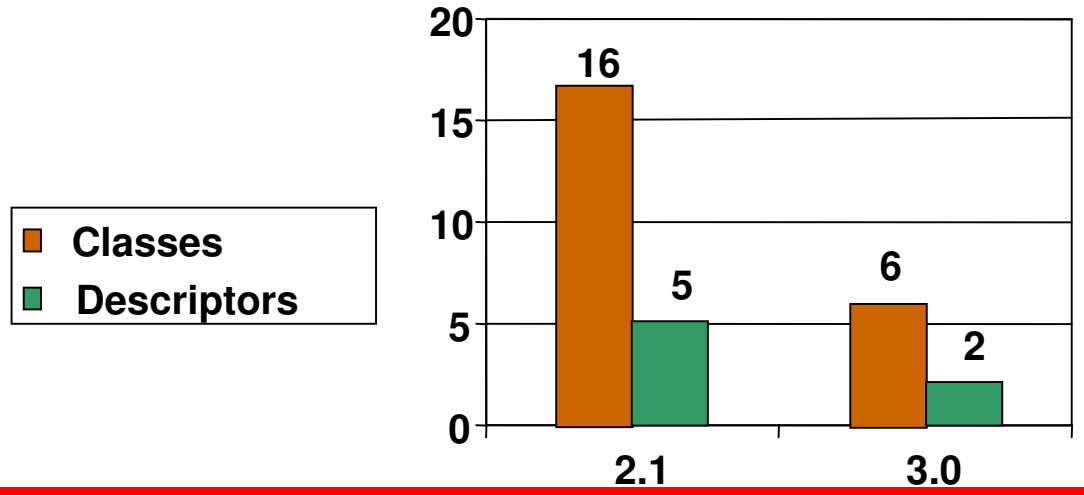


Working with Enterprise JavaBean Types



EJB 2 Versus EJB 3: Simplifying Complexity





Simplify EJB Development

- POJO (Plain Old Java Object) Class
  - EJB Class will be a plain java class
- POJI (Plain Old Java interface)
  - Regular business interface
  - EJB interface does not have to implement EJBObject
- No need of home interface
- Annotations for type of EJB and interface

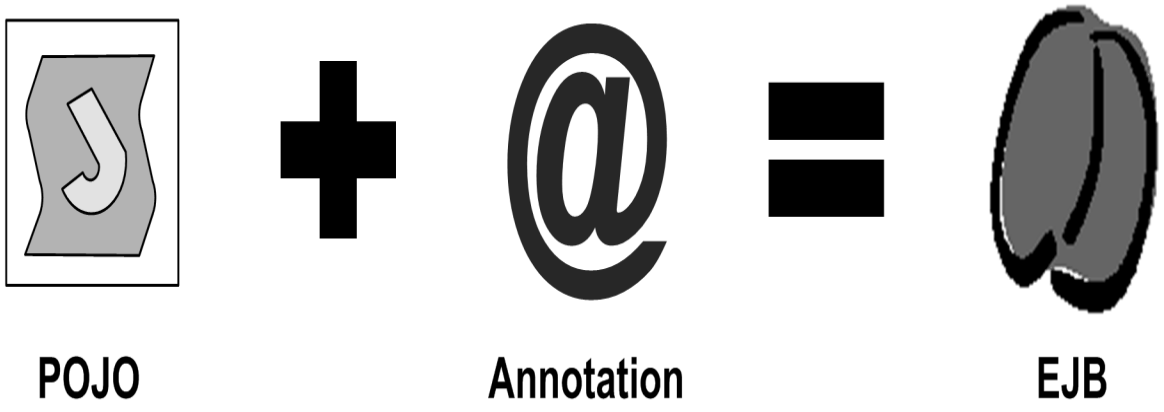
EJB 2.1 Session Bean Class

```
public class CartEJB implements SessionBean
{
  protected Collection items = new ArrayList();
  public void add(String item)
  {
       items.add(item);
  }
  public Collection getItems()
  {
       return items;
  }
  public void completeOrder(){ .. }
  public void ejbCreate(){}
  public void ejbActivate(){}
  public void ejbPassivate(){}
  public void ejbRemove(){}
  public void setSessionContext(SessionContext context){}
}
```

EJB 2 Deployment Descriptor

```
<session>
   <display-name>Shopping Cart</display-name>
   <ejb-name>MyCart</ejb-name>
   <home>CartHome</home>
   <remote>Cart</remote>
   <ejb-class>CartEJB</ejb-class>
   <session-type>Stateful</session-type>
   <transaction-type>Container</transaction-type>
</session>
```

EJB 3 : Simplifying with annotations



@Stateless

@Stateful

@MessageDriven

@Entity

Same Example: EJB 3

```
@Stateful
public class CartBean implements Cart {
    private ArrayList items;

    public void add(String item) {
        items.add(item);
    }

    public Collection getItems() {
        return items;
    }
   @Remove
   public void completeOrder()
   {
   }
}
```

Same Example: EJB 3

```
@Remote
public interface Cart {
    public void addItem(String item);
    public void completeOrder();
    public Collection getItems();
}
```

Deployment Descriptor

EJB 3 Simplifications

- Eliminated requirement for Home Interface
  - Not needed for session beans
- Business interface is a POJI
  - Bean can implement it
  - Bean can have more than one business interface
  - Can support remote access
  - EJB(Local)Object removed from client view
  - RemoteExceptions are removed from programmer and client view
- Eliminated requirement for unnecessary callback methods
  - Removed requirement to implement javax.ejb.SessionBean

General Changes in Session Beans and Message Driven Beans

- Dependency Injection
  - Field and property injection supported
  - Most J2EE resource types supported: ejb-ref, ejb-local-ref, resource-ref, resource-env-ref and environment-entry
- Enhanced lifecycle methods
  - Custom methods for standard lifecycle events
  - Callback listener classes may be used to delegate lifecycle management
- Interceptors
  - Interceptor classes may be registered to intercept business methods
  - Provides equivalent of AOP around advice

Simplification Through Defaults

- Minimize use of metadata that provide defaults for:
  - Names
  - Use of transaction management types
  - Transaction attributes
  - Unchecked methods
  - Use of caller identity
  - Etc.

Enhanced Lifecycle Methods

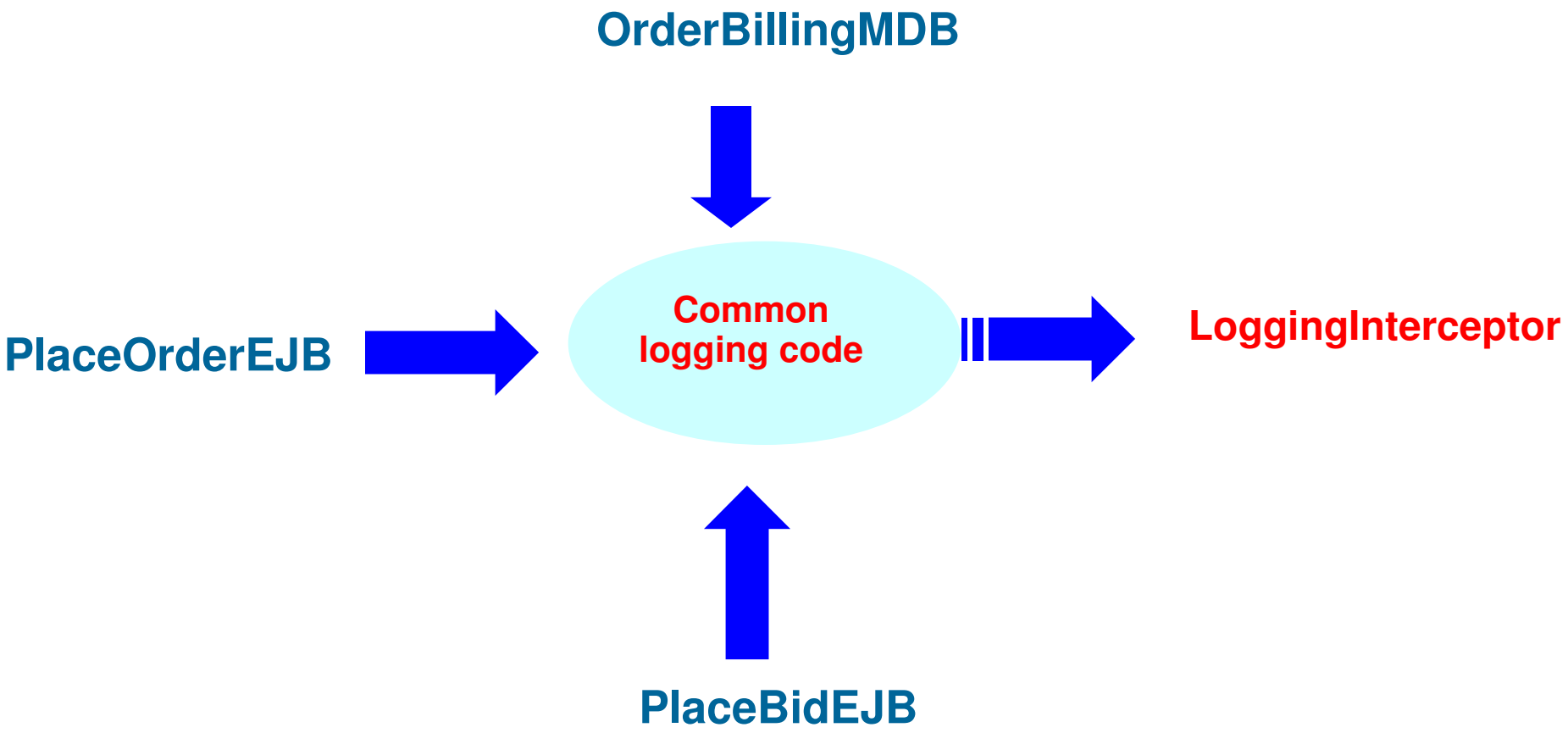
- No need to implement unnecessary call back methods
- Mark any arbitrary methods as callback method using annotations or XML

```
@PostConstruct
    public void initialize() {
        items = new ArrayList();
    }
```

Interceptors

- Provides fine grained control over the method invocation flow
  - may be either a method in the same bean class or an external class
  - Used with SLSB, SFSB, MDB
- Usage
  - Modify parameters before they're passed to the bean
  - Modify the value returned from the bean
  - Catch and swallow method exceptions
  - Interrupt the call completely (handy for a home-grown security framework)
  - Provide method profiling

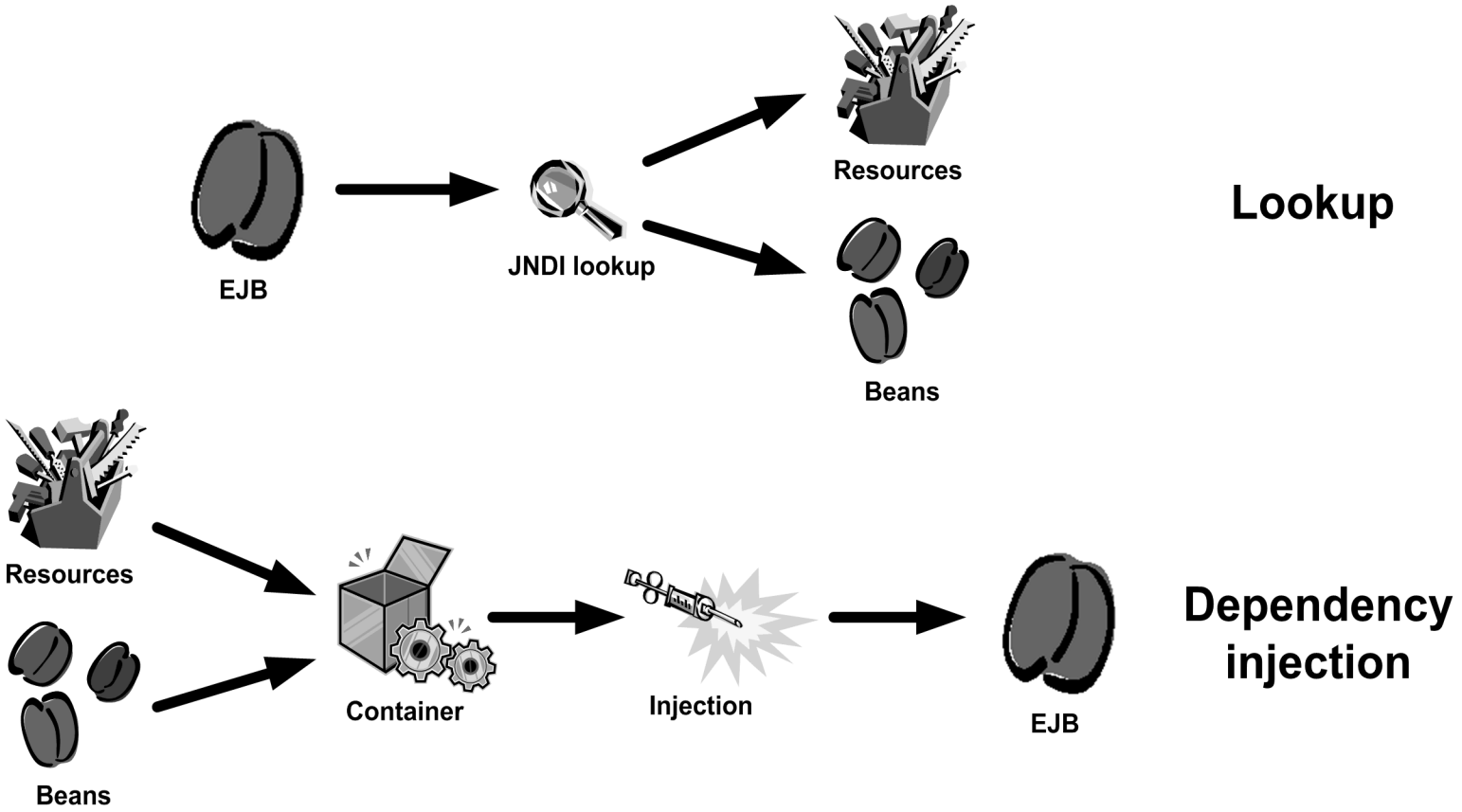
Interceptors



Interceptor Example

```
@Stateless
@Interceptor(value="oracle.ejb30.ProfilingInterceptor")
// identify external interceptors
public class HelloWorldBean implements HelloWorld {
}
public class ProfilingInterceptor {
..
  @AroundInvoke // mark this method as a bean
   interceptor
    public Object checkPermission(InvocationContext
   ctx) throws Exception {
        System.out.println("*** checkPermission
   interceptor invoked");
       … }
}
```

Dependency Injection



Injection

- Container can initialize instance variables at time bean is made available
- Setter injection is better technique
  - Better testability
  - Considered constructor injection, but found it not as simple / flexible
- These techniques can be used to inject:
  - EJBContext
  - EntityManager
  - Resources
  - Session bean references

Injection Examples

```
@EJB AdminService bean;
public void privilegedTask()
{
   bean.adminTask();
}
```

```
@Resource(name="myDB")
public void setDataSource(DataSource myDB) {
  customerDB = myDB;
}
```

```
@Resource javax.ejb.SessionContext sc;
…
TimerService ts = sc.getTimerService();
```

Client View

- Homes eliminated
  - With metadata, injection, easy lookup(), etc.,
    Homes not needed for session beans
    (either stateless or stateful)
  - Stateless SessionBean homes
    not very useful anyway
- Stateful SessionBean homes have useful create methods
  - But: shifting functionality to “initialization” business method
    enables home to be eliminated
  - @Remove annotation completes the picture

EJB 2 Complex Client View

- Need ejb-ref entry:

```
<ejb-ref-name>MyCart</ejb-ref-name>
<ejb-ref-type>Session</ejb-ref-type>
<home>CartHome</home>
<remote>Cart</remote>
```

- Complex Lookup:

```
Object homeObject = context.lookup("java:comp/env/MyCart");
CartHome home = (CartHome)
    PortableRemoteObject.narrow(homeObject,CartHome.class);
Cart cart = (Cart)
    PortableRemoteObject.narrow(home.create(), Cart.class);
cart.addItem("Item1");
```

EJB 3 Client View

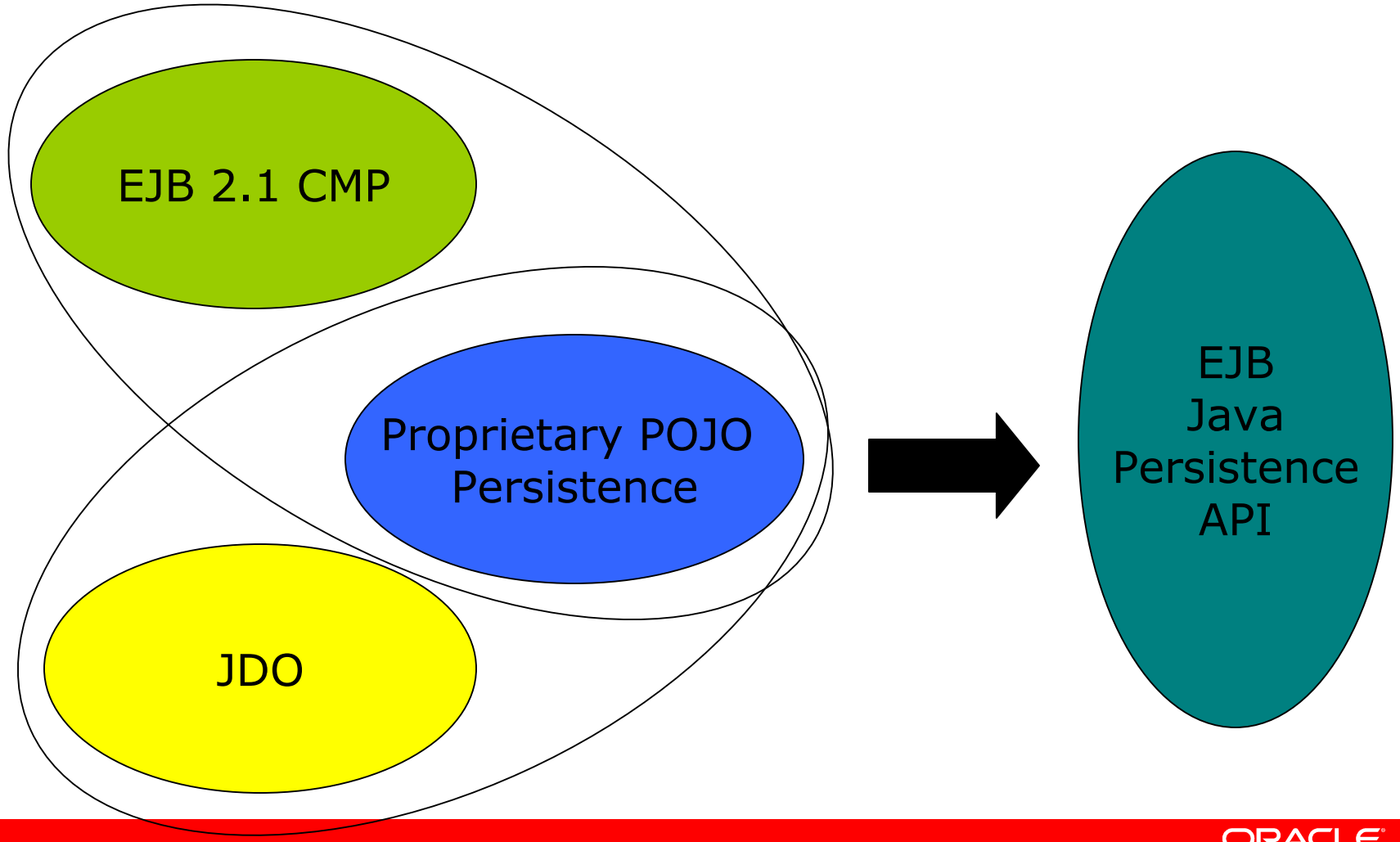
```
@Stateful
public class OrderBean {
@EJB CartEJB cart;
 public void addItems() {
     cart.addItem("Item1");
  }
}
```

EJB3 Java Persistence API

EJB3 JPA: Goals

- Simplify programming model
- Improve modelling capabilities
  - Inheritance and polymorphism
- Standardize O/R mapping
  - Annotations and O-R XML
- Make entities usable outside the container
- Facilitate testability

Migrating Persistent Systems

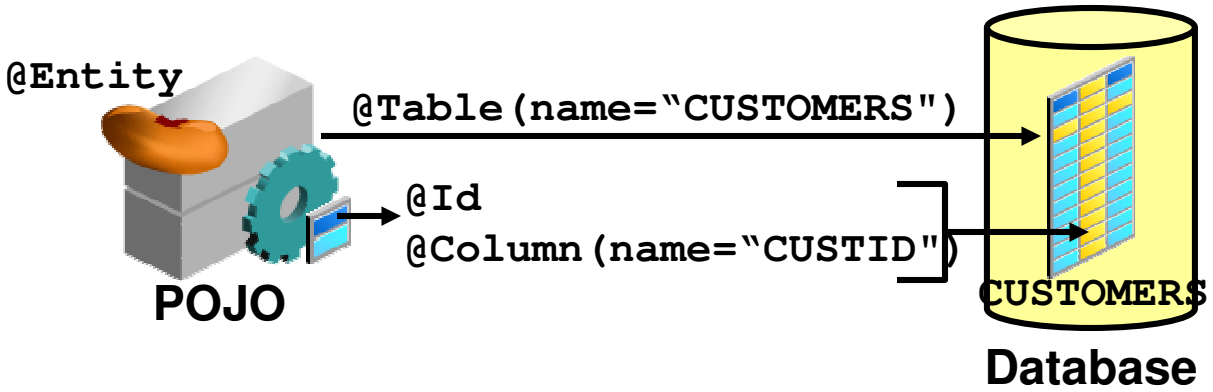


EJB 3.0 Java Persistence API

- Simplification of container-managed persistence
  - POJO / JavaBeans architecture approach
  - Support use of new()
  - Allow for use of entities outside the container
    - Web container
    - Java SE
- Support for light-weight domain modelling, including
  - Inheritance and polymorphism
  - Object-relational mapping metadata
- Elimination of need for data transfer objects and related anti-patterns

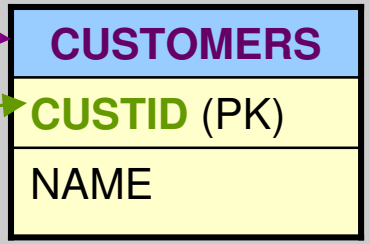
What Are JPA Entities?

- An JPA entity:
  - Is a lightweight object that manages persistent data
  - Is defined as a plain old Java object (POJO) marked with the `Entity` annotation (no interfaces required)
  - Must implement the `java.io.Serializable` interface to be passed by value to a remote application
  - Is mapped to a database by using annotations



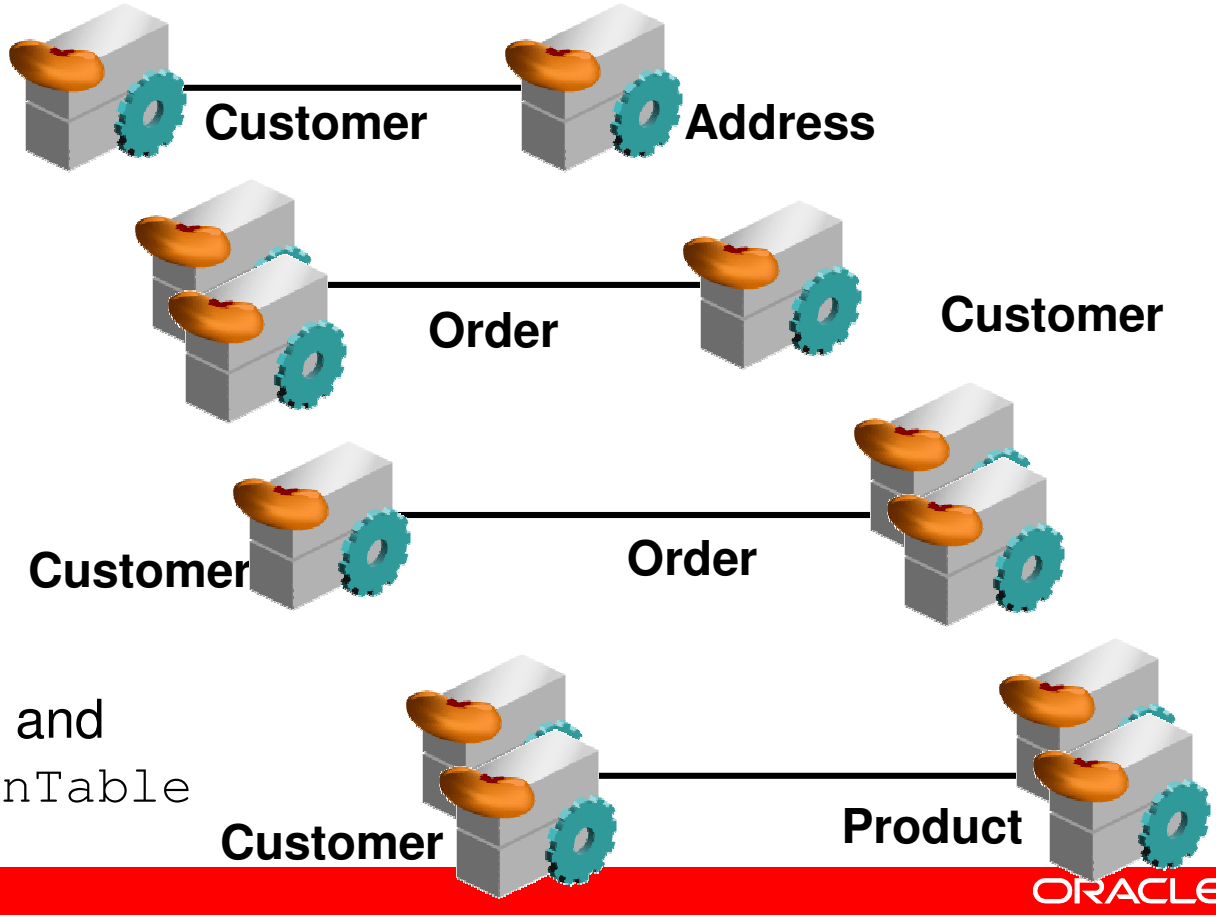
Example: JPA Entity

```
@Entity
@Table(name="CUSTOMERS")
 public class Customer {
    @Id
    @Column(name="CUSTID")
    private Long id;
    private String name;
    private Address address;
    private HashSet orders = new HashSet();

    public Long getId() {
      return id;
    }

   protected void setId (Long id) {
     this.id = id;
   }
...
```



Mapping Relationships Between Entities

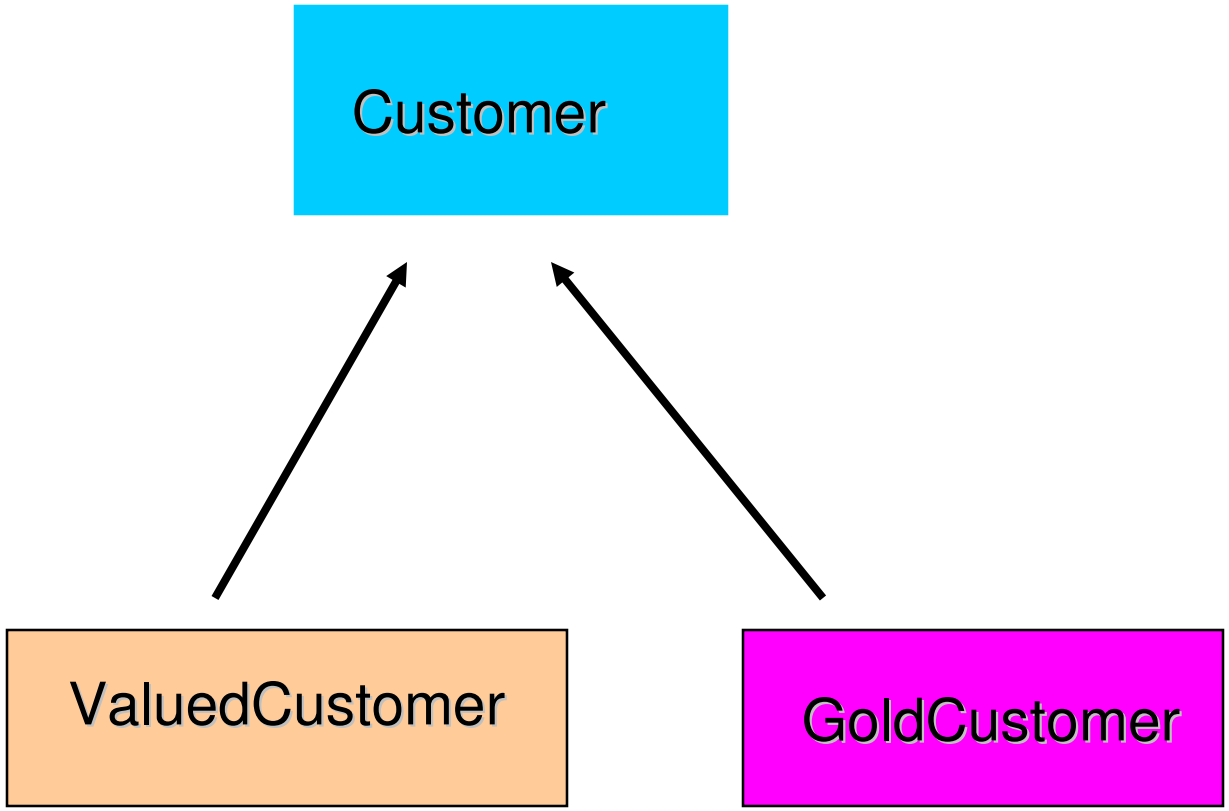
- Annotations for entity relationships:
  - `OneToOne`
  - `ManyToOne`
  - `OneToMany`
  - `ManyToMany` and `AssociationTable`



Mapping Entity Relationships

```
// In the Order class

@ManyToOne
@JoinColumn(name="CUSTID")
public Customer getCustomer() {
  return customer;
}
...
// In the Customer class
@OneToMany(mappedBy="customer")
public Set<Order> getOrders() {
  return orders;
}

public void setOrders(Set<Order> orders) {
  this.orders = orders;
}

// other business methods, etc.
}
```

Inheritance and polymorphism



Mapping Classes to Tables

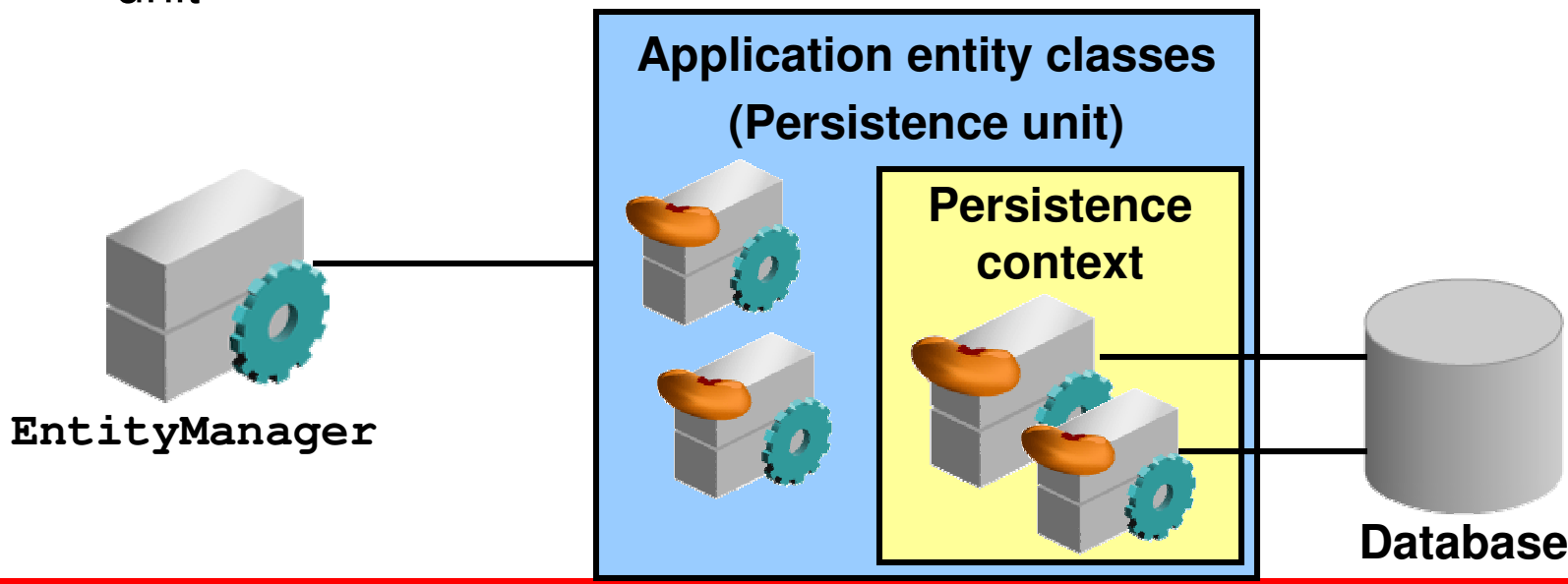
- Use Java™ application metadata to specify mapping
- Support for usual inheritance strategies
  - Single table per class hierarchy
  - Table per class
  - Joined subclass
- Default type mappings defined by spec
- Custom type mappings for finer control and flexibility

Inheritance Mapping Example

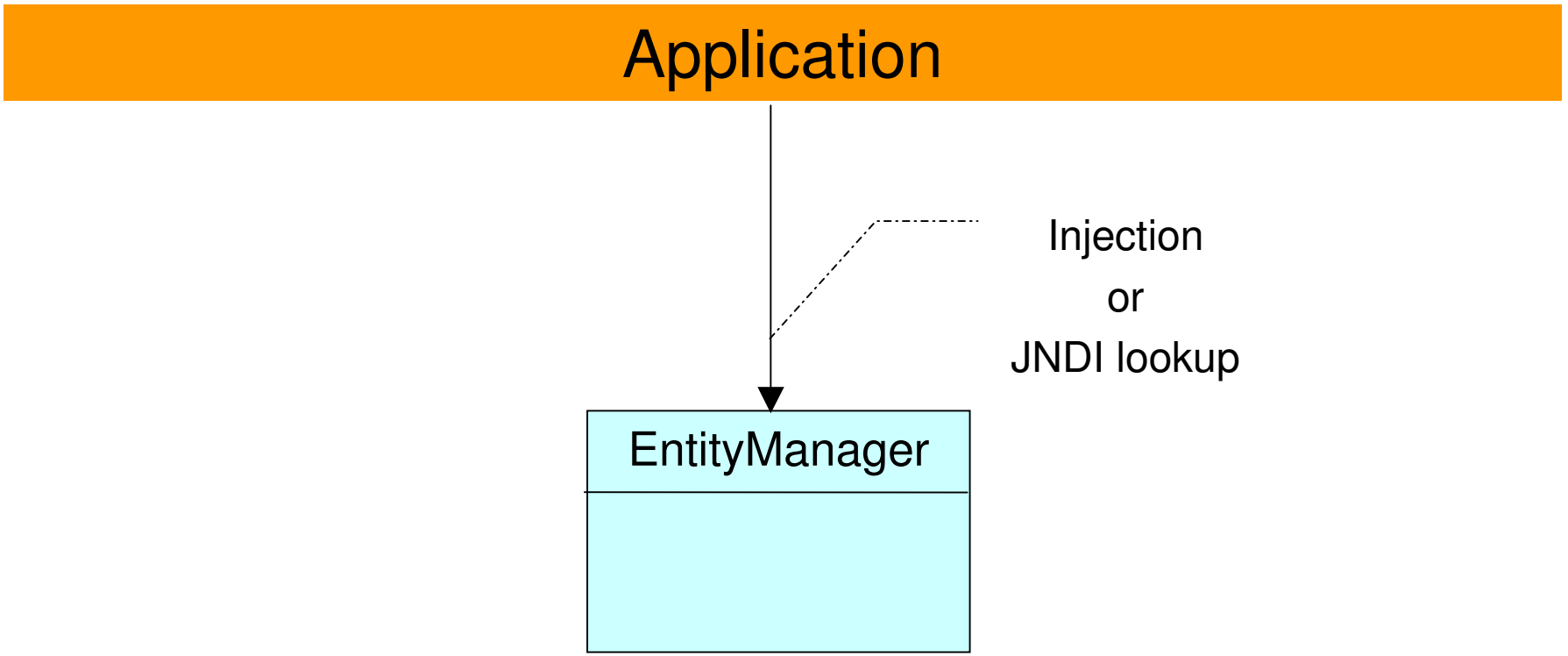
```
@Entity
@Table(name="CUSTOMERS")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE),
@DiscriminatorColumn(name="CUST_TYPE",
                     discriminatorType=STRING)
public class Customer {
...
}

@Entity
@DiscriminatorValue(value="V")
public class ValuedCustomer extends Customer{...}
```

What Is `EntityManager`?

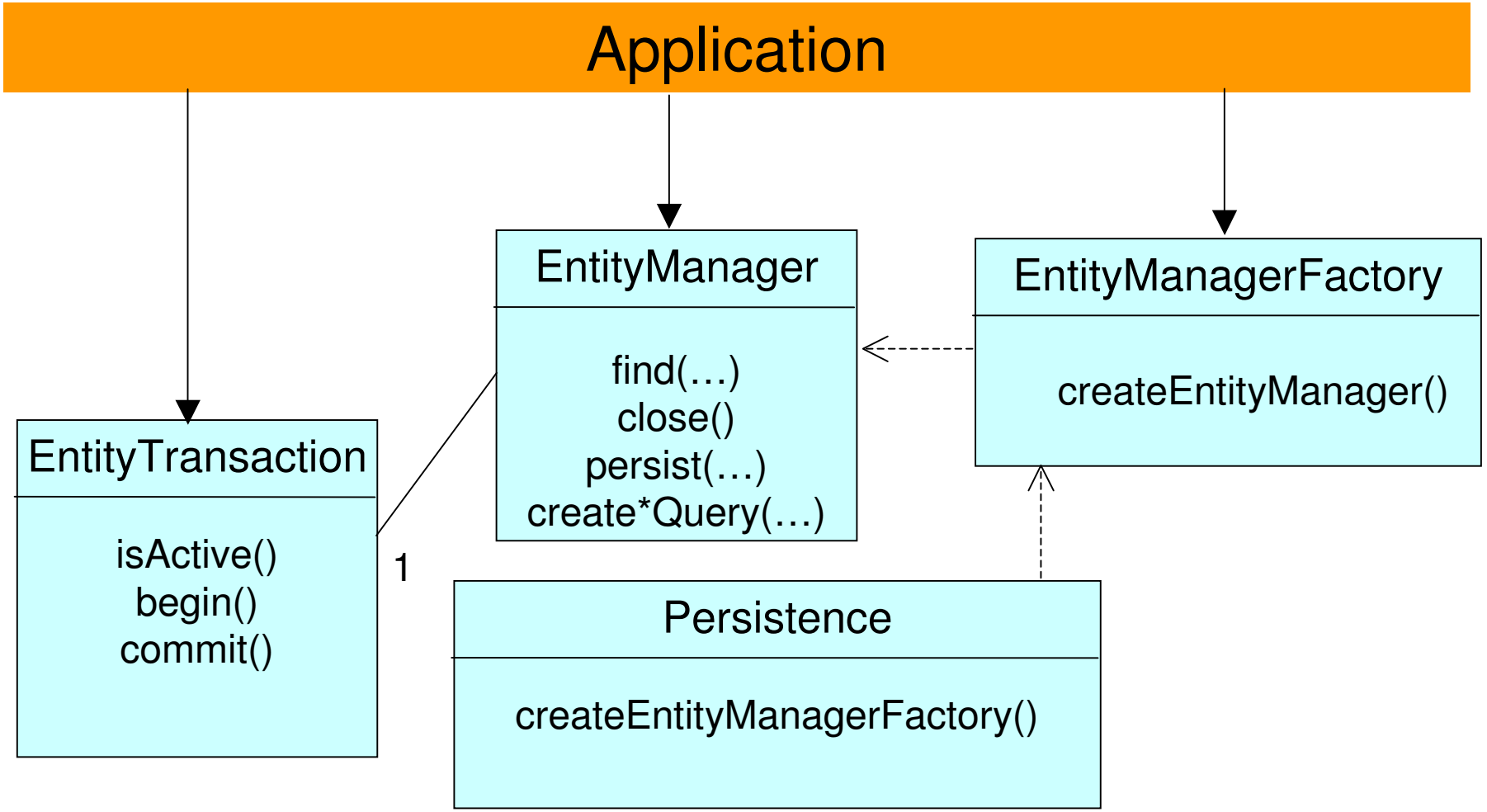
- `EntityManager` is:
  - An API that manages the life cycle of entity instances
  - Associated with a persistence context
  - An object that manages a set of entities defined by a persistence unit



Container Managed EM

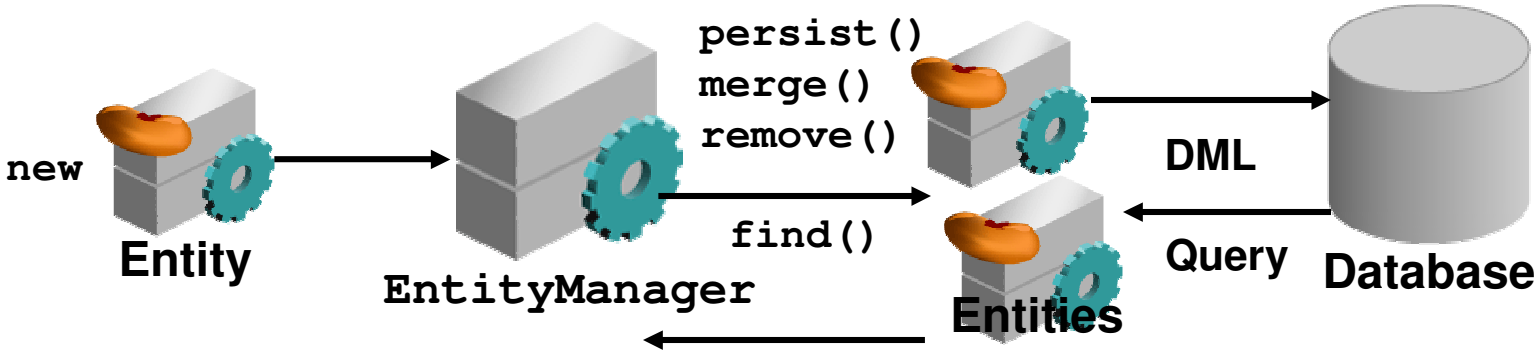


Application Managed EM

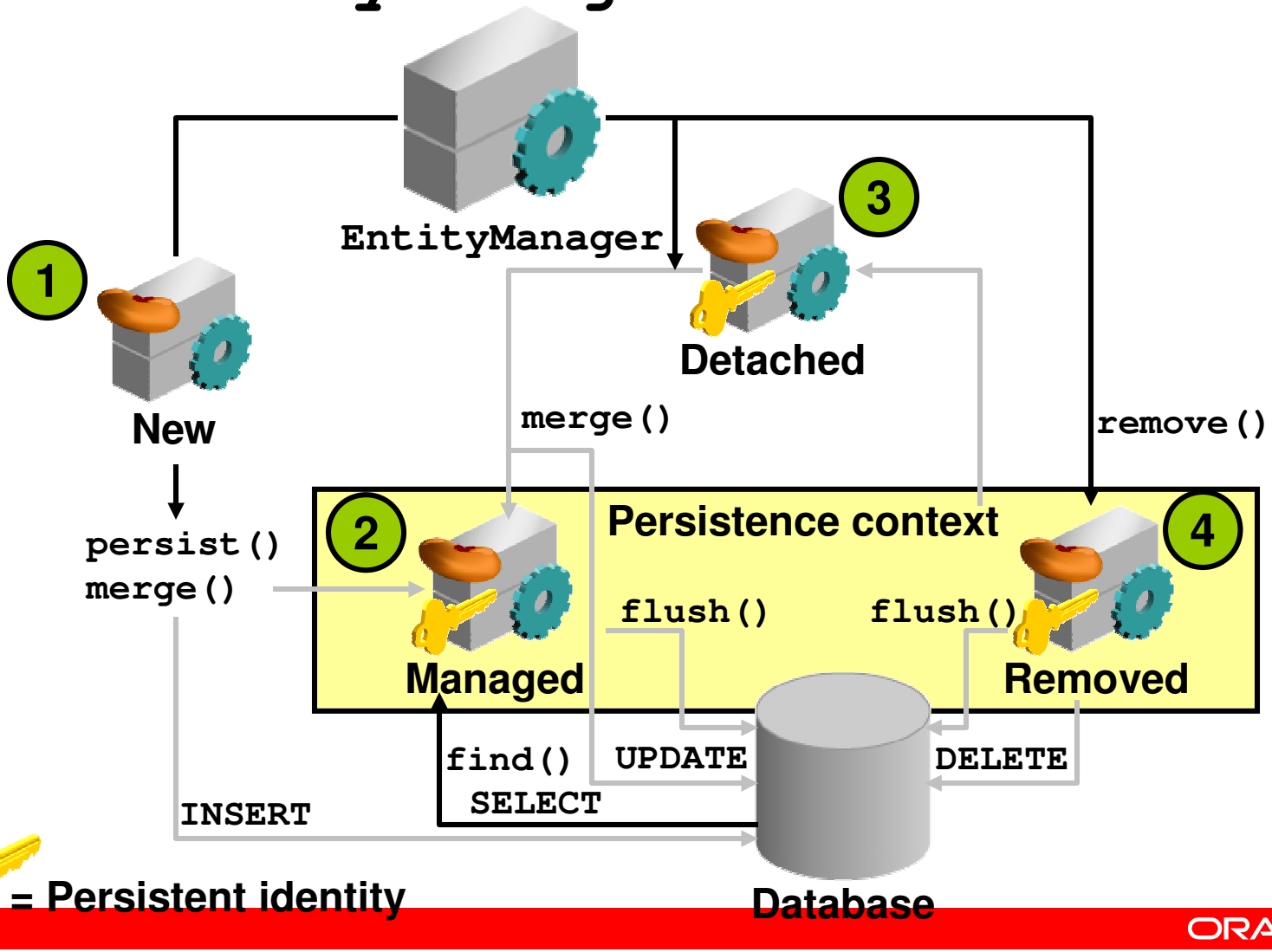


Managing Persistence of Entities

- The life cycle of an entity is managed by using the `EntityManager` interface,
- An entity can be created by using:
  - The `new` operator (creates detached instance)
  - The `EntityManager` Query API (synchronized with the database)
- An entity is inserted, updated, or deleted from a database through the `EntityManager` API.



Managing an Entity Life Cycle with `EntityManager`



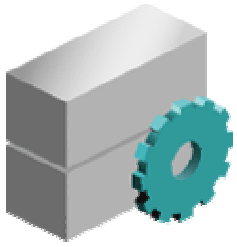
Manipulate Data

- To insert new data:
  1. Create a new entity object.
  2. Call the `EntityManager persist()` method.

```
@PersistenceContext
  private EntityManager em; // inject the EntityManager

...
public void persistUser() {
  User user = new User();
  user.setFirstName("Debu");
  user.setLastName("Panda");
  em.persist(user);
  // On return the user object contains persisted state
  // including fields populated with generated id values
}
```

Retrieving Entities by Using the `Query` API

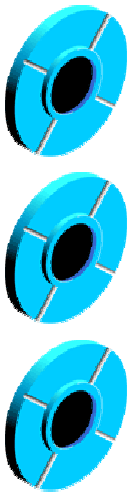
- The `EntityManager` interface provides the `Query` API methods to execute EJB QL statements:



EntityManager

- `createQuery(String jpql)`
- `createNamedQuery (String jpql)`
- `Native Query methods`

`Query` instance methods:



```
setParameter(String, Object)
setParameter(int, Object)
Object getSingleResult()
List getResultList()
Query setMaxResults(int)
Query setFirstResult(int)
```

Query API

- Queries can be expressed in JPQL or native SQL
- Can be dynamic or stored as a named query on entities

```
public List findByName (String name) {
  return em.CreateQuery (
   "SELECT e FROM Employee e " +
   "WHERE e.name LIKE :empName")
   .setParameter("empName", name)
   .setMaxResults(10)
   .listResults();
}
```

Named Queries

```
@NamedQuery(
  name=
    "findEmployeeByName",
  queryString=
    "SELECT e FROM Employee e " +
    "WHERE e.name LIKE :empName"
)
```

```
@PersistenceContext public EntityManager em;
...
List employees =
  em.createNamedQuery("findEmployeeByName")
    .setParameter("empName", "Debu")
    .listResults();
```

JPQL Enhancements over EJBQL

- Simplified syntax
- Bulk update and delete operations
- Projection list (SELECT clause)
- Group by, Having
- Subqueries (correlated and not)
- Additional SQL functions
  - UPPER, LOWER, TRIM, CURRENT_DATE, ...
- Dynamic queries
- Polymorphic queries

Native SQL Queries

- Allow direct SQL over actual database schema
  - Very useful for some applications
  - Database portability overrated for some applications
- Allow SQL query results to be mapped into entity beans and/or instances of other Java™ classes

EJB 3 In Action

Available EJB 3 Containers

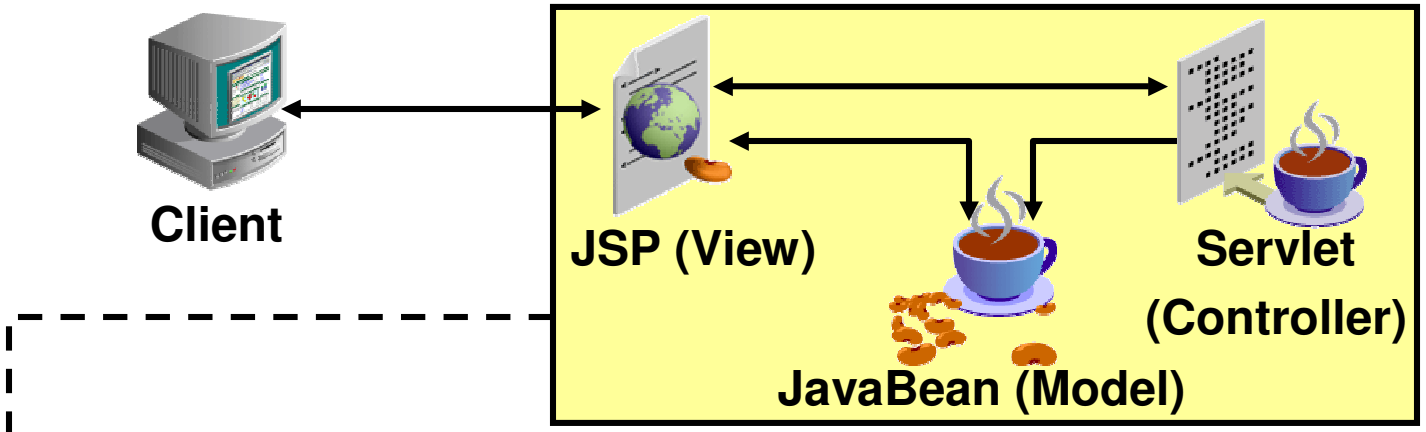
- Sun Glassfish Application Server
  - TopLink Essentials
- Oracle Application Server 10g
  - TopLink Essentials
- JBoss Application Server
  - Hibernate

EJB 3 and Spring 2.0

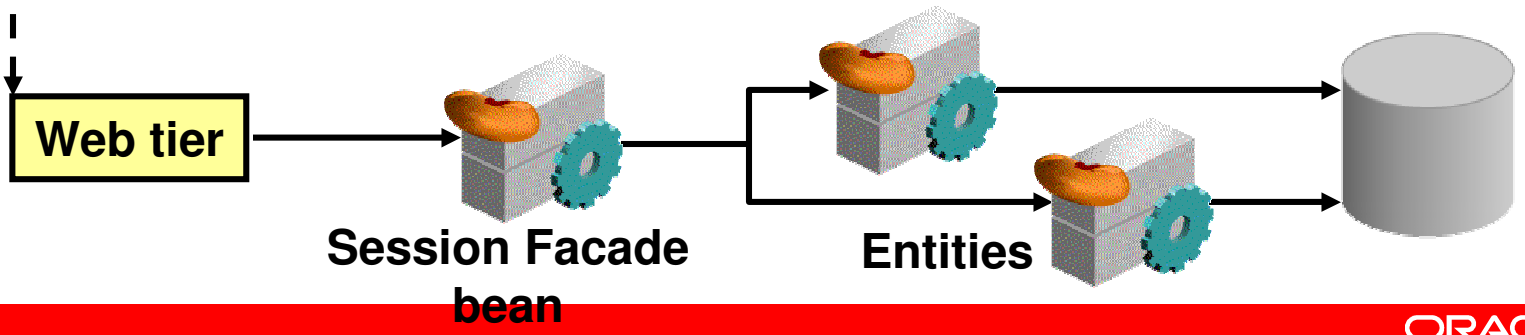
- Spring 2.0 supports JPA side fully
  - TopLink Essentials
  - Simplify use of JPA using JpaTemplate
- Includes support for EJB 3.0 container features
  - Pitchfork project supports EJB 3 annotations
- Great Integration story
  - Spring enabled session beans and MDBs
  - Inject session beans into Spring POJOs

Designing a J2EE Application

- Web-tier design: Use an MVC design pattern



- Business Logic tier: Use a Session Facade pattern



Using EJB 3 Session bean from web app

- May use Injection from managed classes
  - Servlet, Filter, Context Listener, JSP Tag handlers, JSF Managed bean

```
public class ActionServlet .. {
@EJB HelloEJB hello;
 public void doPost() {
     hello.sayHello("Curio George");
  }
}
```

- Avoid injection of stateful session beans from multi-threaded classes

Using JPA from web app

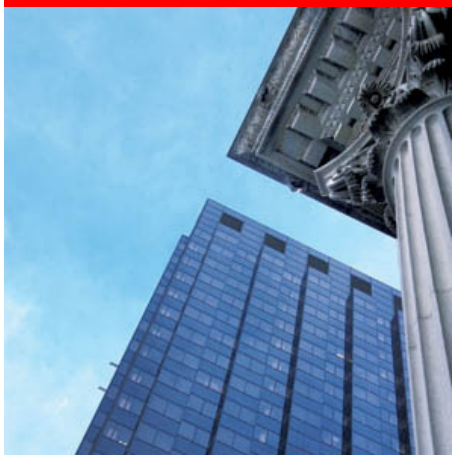
- Package entities in WAR
  - Both Container-managed and application-managed entity manager
- Avoid injection of container-managed entity manager
  - Use JNDI
  - Application-managed entity manager with JTA transaction

Oracle and EJB3

Oracle and EJB3

- Oracle was co-specification lead for EJB 3.0
  - Oracle has built the reference implementation for EJB 3.0 based on TopLink (TopLink Essentials)
  - TopLink Essentials can be used as persistence provider for EJB3 persistence
- Oracle has full implementation of EJB3 and JPA in OAS10.1.3.1
  - JPA has passed CTS
- Great tool support
  - JDeveloper 10g 10.1.3.1 – great support for EJB 3
  - Leads the Eclipse O-R Mapping tooling project

More Info



http://otn.oracle.com/ejb3

http://otn.oracle.com/jpa

http://manning.com/panda

http://debupanda.com