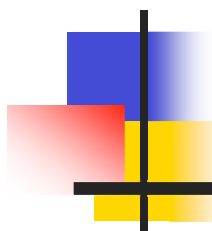


# Migrating to EJB 3.0: Panama Canal or Northwest Passage?



---

Mike Keith

Oracle Corp.

[michael.keith@oracle.com](mailto:michael.keith@oracle.com)





# About Me

---

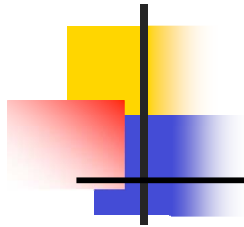
- Co-spec Lead of EJB 3.0 (JSR 220)
- Java EE 5 (JSR 244) expert group member
- Architect for OracleAS TopLink and OracleAS EJB Container in OracleAS OC4J
- 15+ years experience in OO persistence and numerous persistence implementations
- Presenter at JavaOne, JavaPolis, TSS Java Symposium, JA00, CSS, JavaPro Live, *etc.*



# Audience Poll

---

1. How many people will be migrating from EJB 2.0/2.1 systems?
2. How many still have EJB 1.1 systems?
3. How many are using other persistence solutions?
4. How many think you will be migrating within a year of when the spec is final?



# Goal

---

Understand the issues involved in migrating existing EJB and other applications to EJB 3.0



# Agenda

---

1. EJB 3.0 Goals
2. Motivation
3. Migrating Session Beans to EJB 3.0
4. Migrating Entity Beans to EJB Persistence API
5. Migrating POJOs to EJB Persistence API
6. Preparing for the Future
7. Summary



# Agenda

---

1. EJB 3.0 Goals
2. Motivation
3. Migrating Session Beans to EJB 3.0
4. Migrating Entity Beans to EJB Persistence API
5. Migrating POJOs to EJB Persistence API
6. Preparing for the Future
7. Summary



# EJB 3.0 Goals

---

- Simplify developer experience
  - EJB's now Plain Old Java Objects (POJO)
  - Can use metadata annotations
    - XML descriptors are no longer necessary
    - Use defaults when possible
  - Unnecessary artifacts are optional
  - Simplify client view using dependency injection
- Standardize persistence API for Java platform
  - Based on success of leading ORM solutions (*e.g.* Hibernate, TopLink)



# Compatibility/Interoperability

---

- **Backward Compatibility with EJB 2.1**
  - Existing Applications will continue to work
- **Interoperability with EJB 2.1**
  - Use of EJB 3.0 persistence API from EJB 2.x
  - Access EJB 2.1 modules from EJB 3.0
  - Access EJB 3.0 style EJBs from EJB 2.1



# Agenda

---

1. EJB 3.0 Goals
2. Motivation
3. Migrating Session Beans to EJB 3.0
4. Migrating Entity Beans to EJB Persistence API
5. Migrating POJOs to EJB Persistence API
6. Preparing for the Future
7. Summary



# Motivation

---

- Efficiency
  - Improves maintainability (or makes it possible!)
  - Cheaper dev costs?
- Extensibility
  - Offers better/more integration with other components/subsystems
  - Other libraries
- Life-prolonging
  - Standardization — future path
  - Access to bigger developer pool
  - Availability of support, dev tools, resources



# Existing Systems

---

- J2EE
  - EJB 2.1 — session beans, MDB
  - CMP
- Mix of J2EE and proprietary
  - Session beans, JSP, *etc.*
  - Commercial and open source components
- Non-J2EE
  - Non-standard components throughout



# Agenda

---

1. EJB 3.0 Goals
2. Motivation
3. Migrating Session Beans to EJB 3.0
4. Migrating Entity Beans to EJB Persistence API
5. Migrating POJOs to EJB Persistence API
6. Preparing for the Future
7. Summary



# Migration Approaches

---

- Can leave them just the way they are
  - EJB 2.1 components are still supported in EJB 3.0 Containers
- Can migrate server side only
  - Make use of adapted home interfaces
- Can migrate client side only
  - Assume EJB 3.0 session beans all the way
- Can migrate part of the way or all the way
  - Vendors are going to be supporting part-way migration



# Migrating Session Beans

---

## Server side:

- Session beans become POJO's as described in the Simplified API
- Beans don't implement `javax.ejb.SessionBean`
- Strip out old (unused) life cycle methods and annotate any callbacks that may be used
- Beans implement the remote/local business interface(s)



# Migrating Session Beans

---

## Server side:

- RemoteExceptions no longer needed on remote interfaces
- For SFSB change ejbCreates to business methods and add removal method annotated with @Remove
- Optionally add annotations for transaction and security



# EJB 2.1 Session Bean

---

```
public interface Cart extends EJBObject {
    public void add(String item) throws
        RemoteException;
    public Collection.getItems() throws RemoteException;
    public void completeOrder() throws RemoteException,

        NotInCartException;
}
```

```
public interface CartHome extends EJBHome {
    public Cart create() throws CreateException,
        RemoteException;
}
```



# EJB 2.1 Session Bean

---

```
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) {}
}
```



# Same Example: EJB 3.0

---

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



# Same Example: EJB 3.0

---

```
@Stateful
public class CartBean implements Cart {
    private ArrayList items = new ArrayList();

    public void add(String item) {
        items.add(item);
    }
    public Collection getItems() {
        return items;
    }
    @Remove
    public void completeOrder() {...}
}
```



# Migrating Session Beans

---

## Client side:

- Use instance directly
- Do not specify home interface in ejb-ref
- Change bean home refs to use business interface
- EJB[Local]Object methods map to business methods
- RemoteException, CreateException, *etc.* no longer need to be handled



# Migrating Session Beans

---

## Client side:

- For SFSB can use business interface for create/remove
- Optionally replace JNDI lookup code with dependency injection



## EJB 2.1 Session Bean

---

```
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 2.1 Client View

---

## 1. Need ejb-ref entry:

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



# EJB 2.1 Client View

---

## 2. Complex lookup:

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



# EJB 3.0 Client View

---

Becomes:

```
@Stateful
public class OrderBean {
    @EJB Cart cart;
    public void addItem(String item) {
        cart.addItem(item);
    }
}
```



# Migrating Session Beans

---

## Partial Migration:

- Incrementally adopt Simplified API features
  - Don't implement SessionBean interface and make use of magic life cycle methods
  - Make use of dependency injection (in XML descriptor to obtain bean references
  - Use XML without annotations to stay on JDK 1.4



# Migrating Session Beans

---

## Partial Migration:

- Different vendors may offer different paths to get there
  - Tools for doing some of the more automated things
  - Visual editors/IDE to read in old formats and be able to write in new formats
- Is it reasonable to expect to be able to change vendors part-way through migration?



# EJB 3.0 Support with XML

---

```
<remote>com.acme.WeatherReport</remote>
<ejb-class>com.acme.WeatherReportBean</ejb-class>
<env-entry>
  <env-entry-name>CelsiusString</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
  <env-entry-value>degrees Celsius</env-entry-value>
  <inject-field>celsiusString</inject-field>
</env-entry>
...
<interceptor-binding>
  <ejb-name>WeatherReportBean</ejb-name>
  <interceptor-class>
    com.acme.SubscriberVerifier
  </interceptor-class>
</interceptor-binding>
```



# Migrating Message-Driven Beans

---

- Simple to migrate
- Remove implementing MessageDrivenBean
- Use annotations for specifying configuration properties
- Use dependency injection for resources



# Agenda

---

1. EJB 3.0 Goals
2. Motivation
3. Migrating Session Beans to EJB 3.0
4. Migrating Entity Beans to EJB Persistence API
5. Migrating POJOs to EJB Persistence API
6. Preparing for the Future
7. Summary

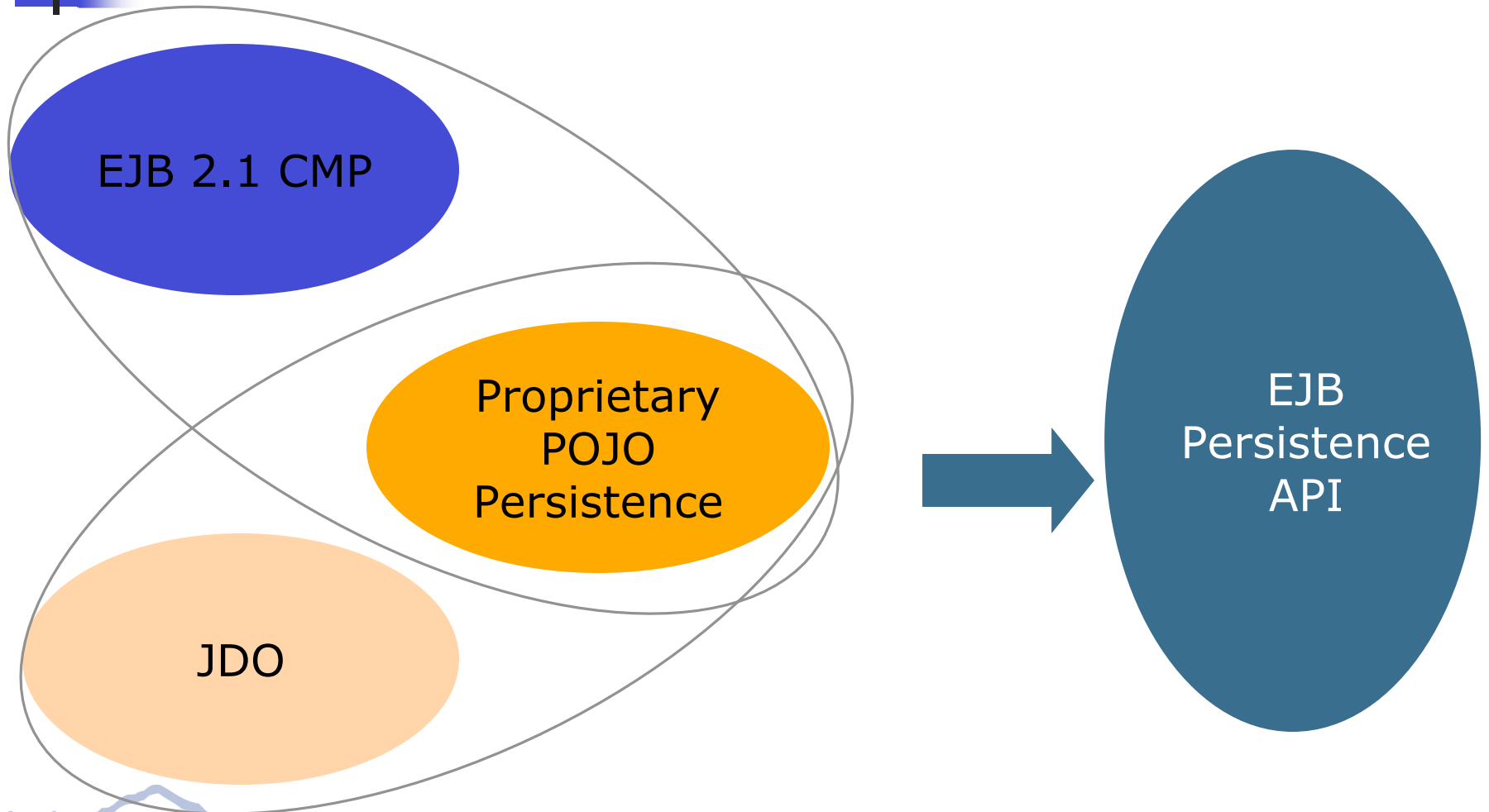


# EJB 3.0 Persistence

---

- EJB 3.0 is a HUGE leap forward w.r.t persistence
- “Modernized” to be lightweight POJO persistence model (already in proven products like TopLink)
- Some major differences from previous approach
- Convergence upon the “correct” model means some short-term efforts

# Migrating Persistence Systems





# Migrating Entity Beans

---

- Abstract/concrete bean class
  - Make class and methods concrete
  - Define and use state attributes

```
public abstract class CustomerBean {  
...  
    public abstract int getStatus();  
    public abstract void setStatus(int status);  
}
```



# Migrating Entity Beans

---

```
public abstract class CustomerBean {  
...  
    public abstract int getStatus();  
    public abstract void setStatus(int status);  
}
```

```
public class CustomerBean {  
    private int status;  
    public int getStatus() { return status; }  
    public void setStatus(int status); {  
        this.status = status;  
    }  
}
```



# Migrating Entity Beans

---

## Home interfaces:

- Local/remote home interfaces no longer apply – remove them
- create
  - `ejbCreate` method to pass in state becomes a constructor

```
public class CustomerBean {
    public CustomerBean(int custId) {
        ejbCreate(custId); }
    public int ejbCreate(int custId) {
        this.custId = custId;
    }
}
```



# Migrating Entity Beans

---

- **remove**
  - Home.remove replaced by EntityManager.remove
  - Pass the object to remove
  - ejbRemove method can be annotated with @PreRemove

```
custHome.remove(custId);
```

becomes:

```
em.remove(cust);
```

or use a bulk delete (blech!):

```
"DELETE FROM Customer c WHERE c.custId = :custId"
```



# Migrating Entity Beans

---

- **ejbHome methods**
  - Can be left as instance methods, since no state is assumed, or be turned into static methods

```
public float.ejbHomeCustomerDiscount(int preferredStatus) {  
    return (preferredStatus * 0.05);  
}
```



# Migrating Entity Beans

---

- Component interfaces (EJB[Local]Object)
  - Can keep it as a business interface and implement it
  - `getPrimaryKey` is a simple accessor method on the bean

```
EJBObject c = custHome.findByName("Johnny Depp");  
Integer custId = (Integer) c.getPrimaryKey();
```

becomes:

```
Customer c = (Customer) em.createNamedQuery("findByName")  
    .setParameter("custName", "Johnny Depp")  
    .getSingleResult();  
Integer custId = c.getId();
```



# Migrating Entity Beans

---

- Component interfaces (EJB[Local]Object)
  - Pass “this” as a self reference instead of `EntityContext.getEJBObject()`

```
return getEntityContext().getEJBObject();
```

becomes:

```
return this;
```



# Migrating Entity Beans

---

- Life cycle methods, EntityContext
  - ejbCreate => init methods/constructors
  - ejbPostCreate => in constructor, PrePersist or PostPersist
  - ejbRemove => PreRemove
  - setEntityContext, unsetEntityContext => -
  - ejbActivate, ejbPassivate => PostLoad, -
  - ejbSelect => concrete dynamic query
  - ejbStore => PrePersist, PreUpdate



# Migrating Entity Beans

---

- Exceptions – Bean
  - Change bean logic to throw whatever exception makes sense to the application (checked or unchecked)
  - May also throw standard 2.1 exceptions to avoid changing client logic
- Exceptions – Client
  - Change to handle application-defined exceptions thrown by the bean method
  - EntityManager method calls throw runtime exceptions (typically cause rollbacks)



# Migrating Entity Beans

---

- Finders
  - Finder becomes either named or dynamic query
  - EJB QL string from DD gets put in annotation or in query

```
public Collection findAllByName(String nameString)
throws FinderException;
```

```
<ejb-ql>
    SELECT OBJECT(c) FROM Customer c
        WHERE c.name LIKE ?1
</ejb-ql>
```



# Migrating Entity Beans

---

Using a dynamic query this becomes:

```
public Collection findAllByName(String nameString) {  
    return getEntityManager()  
        .createQuery("SELECT c FROM Customer c "  
            + "WHERE c.name LIKE :custName")  
        .setParameter("custName", nameString)  
        .getResultList();  
}
```



# Migrating Entity Beans

---

Using a named query this becomes:

```
@NamedQuery (name="findAllByName" ,
    query="SELECT OBJECT(c) FROM Customer c
          WHERE c.name LIKE :custName")
public Collection findAllByName(String nameString) {
    return getEntityManager()
        .createNamedQuery("findAllByName")
        .setParameter("custName", nameString)
        .getResultList();
}
```



# Migrating Entity Beans

---

- Container-managed relationships
  - Java code needs to do the management (LinkedList)
  - Easiest to add a single line to the relationship modifier methods

```
public addOrder(Order order) {  
    getOrders().add(order);  
}
```

Becomes:

```
public addOrder(Order order) {  
    getOrders().add(order);  
    order.setCustomer(this);  
}
```



# Migrating Entity Beans

---

- What if applications didn't follow EJB best practices?
  - Non-local entities and remote access
  - Entity level transaction demarcation
  - Entity method level security
- More effort to migrate since these are no longer available to EJB 3.0 entities (POJOs)

How can you offer something that doesn't exist?



# Migrating Entity Beans

---

- “Wrap” each entity in a Simplified SFSB
- Put the remoteness, transactions and security in session bean
- Create a business interface for the session bean and implement both the entity home interface as well as the component interface (with create methods, finders, *etc.*)
- Query processing code either get changed to use session bean, or each entity result must be wrapped in a session bean instance when returned from the query
- State of the entity stays cached in session bean instance or gets looked up each time

❖ Not pretty, and not recommended!



# Migrating Entity Beans

---

## Turn Homes into POJOs

- Change home interface into a POJI
- Create a POJO that implements the Home interface
- Create and finder methods are simple
- Application init code can bind the home POJO into JNDI
- Client code does not have to change (although it does have to be recompiled with new interface defs)



# Migrating Entity Beans

---

## Example:

```
public Customer findByPrimaryKey(CustomerPK pk)
    throws FinderException {

    Object result =
        getEntityManager().find(Customer.class, pk);
    if (result == null)
        throw new ObjectNotFoundException();
    return result;
}
```



# Migrating Entity Beans

---

## Data Transfer Objects

- Lightweight container objects for entity data
- No longer needed in EJB 3.0
- Resemble what an entity is supposed to be
- Turn them into the entities themselves by adding the logic!



# Agenda

---

1. EJB 3.0 Goals
2. Motivation
3. Migrating Session Beans to EJB 3.0
4. Migrating Entity Beans to EJB Persistence API
5. Migrating POJOs to EJB Persistence API
6. Preparing for the Future
7. Summary



# Migrating Persistent POJO's

---

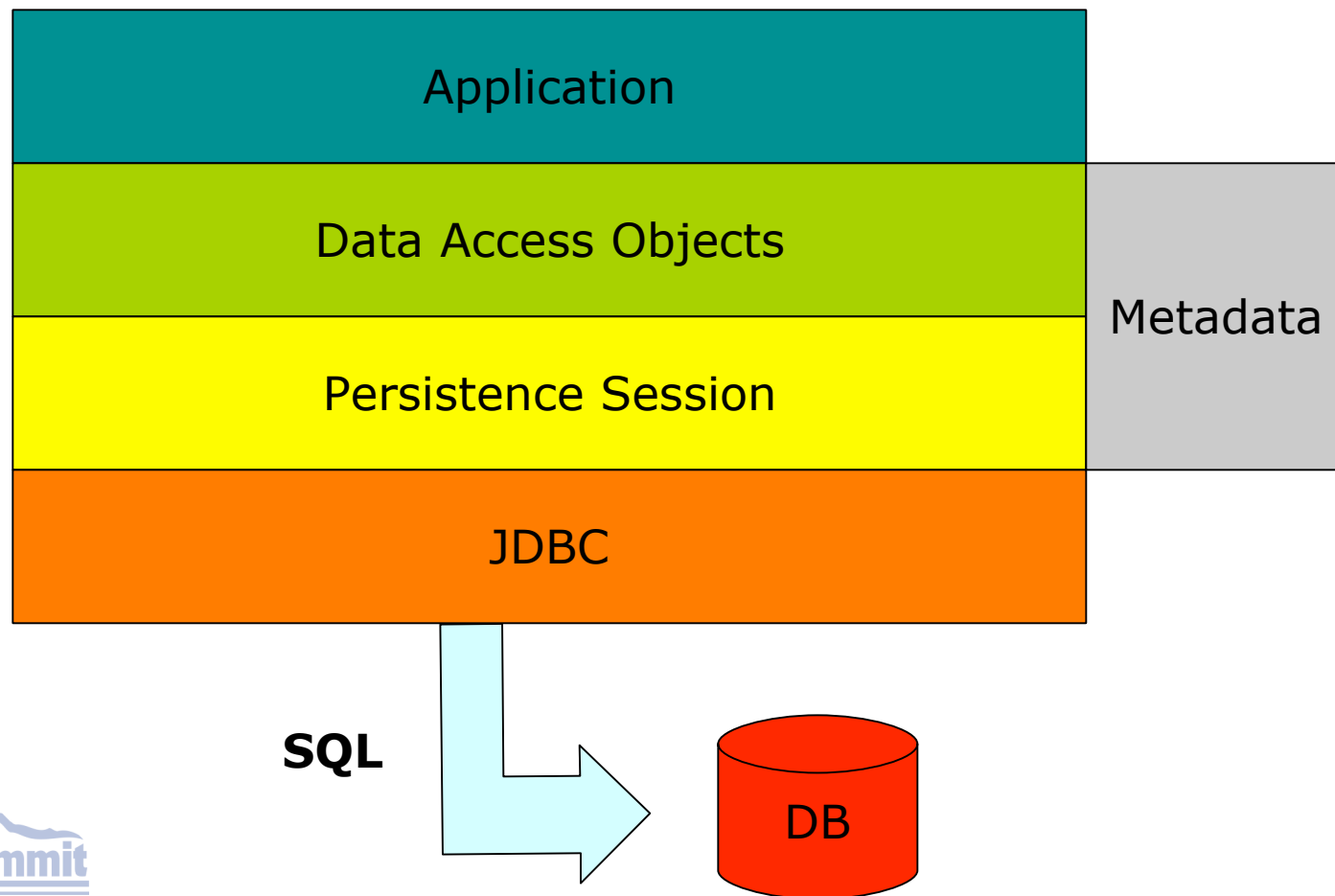
## Good news:

- Already \*are\* persistent POJOs so structure should be same
- Already similar transaction mechanism, session-level API's
- Already does O-R mapping
- This is the majority of applications

## Bad News:

- Session API's are different (object registration, locking, events, queries, *etc.*)
- O-R mapping metadata is different

# Migrating Persistent POJO's





# Migrating Persistent POJO's

---

## Data Access Layer:

- Easy to revamp DAO's to access EJB 3.0 persistence layer
- Calling application code does not need to change
- May not always have one, though

## Persistence Engine:

- Reduces to migrating usage of the "Session" API



# Migrating Persistent POJO's

---

- Persistence API:
  - EntityManager is available outside the Container
    - Most persistence engines will offer EntityManager API in addition to proprietary API
    - Can migrate persistence bits piecemeal
- Queries:
  - Most persistence frameworks provide named queries
  - Just expose them as named queries from EntityManager
  - Allows any query language to be “imported” into EJB3
  - Can migrate to EJB QL query by query



# Migrating Persistent POJO's

---

## Tools:

- Tools will be available and will help to perform some of the migration tasks
  - Mapping file migration
  - Some annotation metadata
  - CMP (EJB QL) query migration
- Other features will likely not be part of any migration tool
  - Relationship maintenance
  - Life cycle methods



# Agenda

---

1. EJB 3.0 Goals
2. Motivation
3. Migrating Session Beans to EJB 3.0
4. Migrating Entity Beans to EJB Persistence API
5. Migrating POJOs to EJB Persistence API
6. Preparing for the Future
7. Summary



# Preparing For the Future

---

- How can I prepare now?
- Apply common sense
  - Adopt products that are showing their commitment to help you get to EJB 3.0 now
  - Use POJO-based persistence products
  - Follow well-known and proven patterns of using persistence
- Educate yourselves

“Preparing for EJB 3.0”

<http://www.oracle.com/technology/ejb3>



# Preparing for the Future

---

- The best migration strategy is not to have to migrate at all!
  - Try EJB 3.0 API's now
  - Features are available in production

- EJB 3.0 Tech Preview:

<http://www.oracle.com/technology/ejb3>

- **EJB 3.0 Reference Implementation (Glassfish ):**

<http://www.java.net/glassfish/>



# Agenda

---

1. EJB 3.0 Goals
2. Motivation
3. Migrating Session Beans to EJB 3.0
4. Migrating Entity Beans to EJB Persistence API
5. Migrating POJOs to EJB Persistence API
6. Preparing for the Future
7. Summary



# Summary

---

- ✓ The spec facilitates migration from different perspectives, as well as version interoperability
- ✓ Vendors committed to EJB 3.0 will help to get you there offering migration paths suited to their users
- ✓ Examine your application and determine which features it uses before you start migrating
- ✓ Migrate a small piece to test out your initial investigation and analysis





# Summary

---

- ✓ Using J2EE patterns helps. Not using J2EE anti-patterns helps more!
- ✓ You have already built your own Panama Canal (or NW passage!)
  - ✓ If it is well-architected it should be easy to migrate
  - ✓ If it is less than well-architected stop looking for the NW Passage and keep the waterways that you have flowing
  - ✓ You may be better off keeping what you have and build new canals with new API's as you need them

