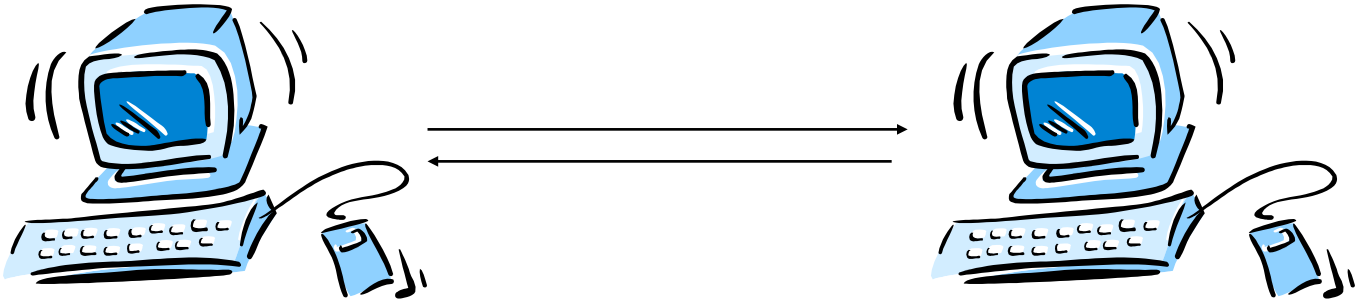


95-702 Distributed Systems

Persistence

Recall Four Styles of Integration

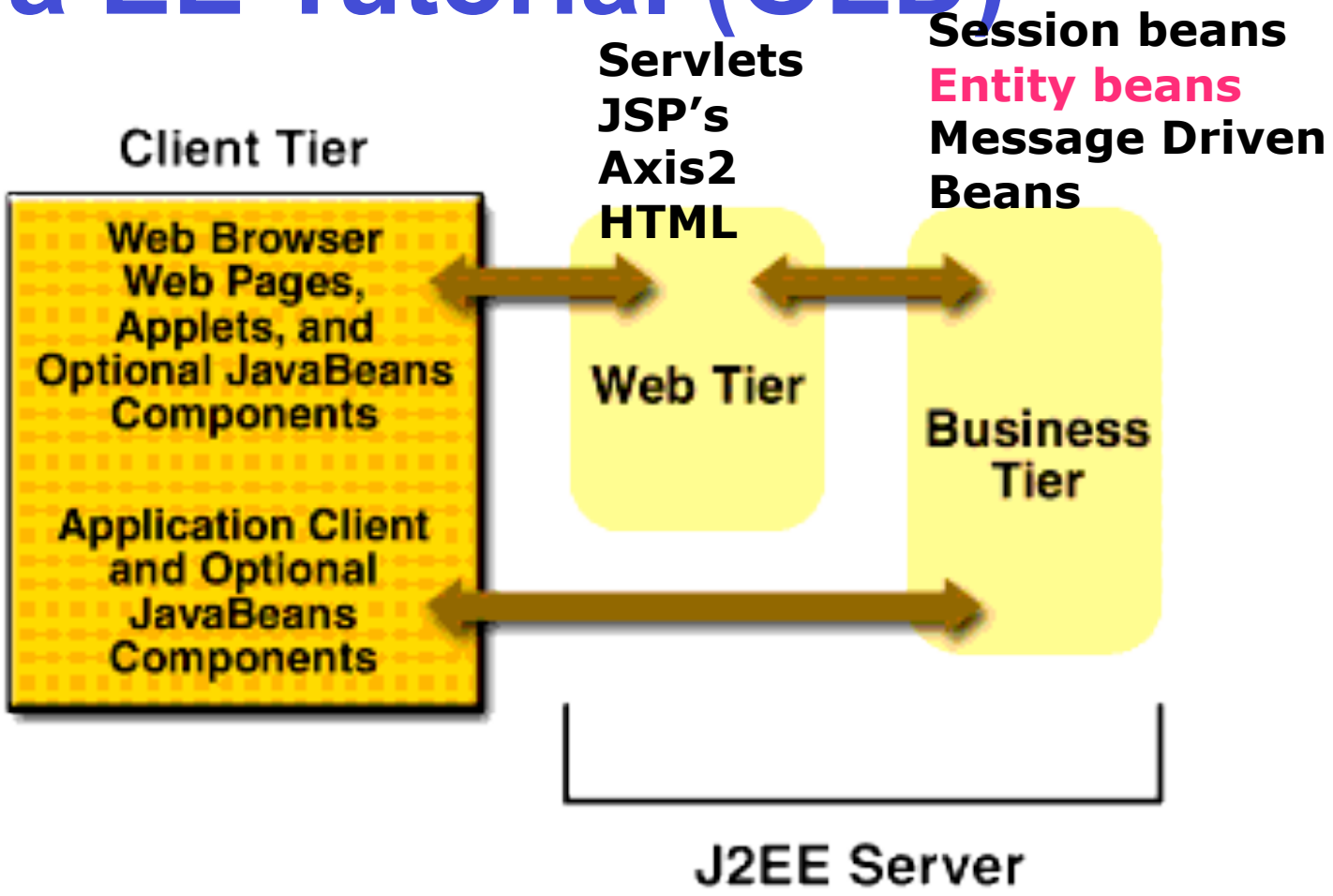
- RPC/RMI
- Shared File
- Messaging
- Share database

❖And, we want to interact with databases even when we are not intent on integration.

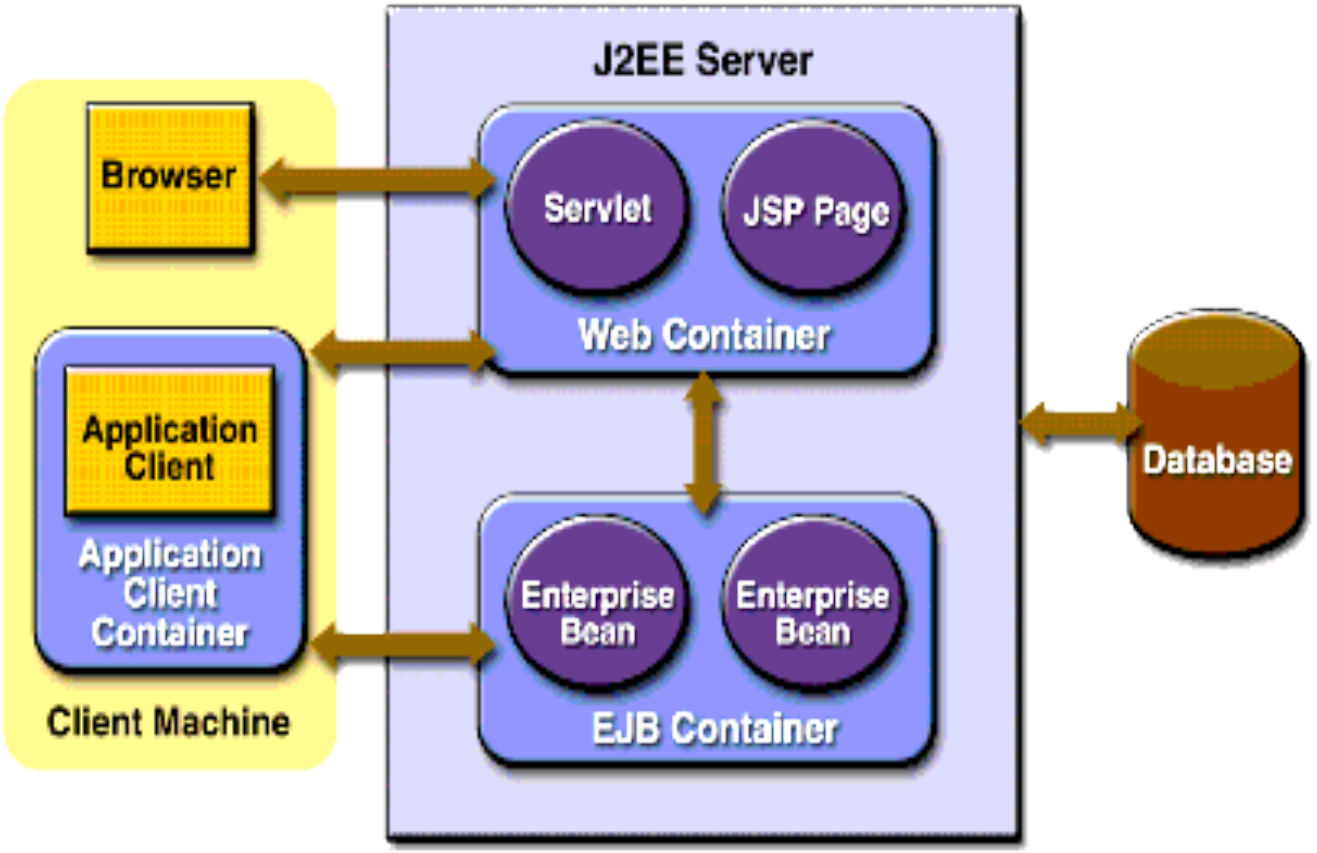
Accessing Databases

- The Java Persistence API is meant to replace EJB Entity Beans.
- Entity beans are heavyweight and require an EJB container.
- Why not use JDBC?
- Separation of concerns: separate the integration logic from the business logic.

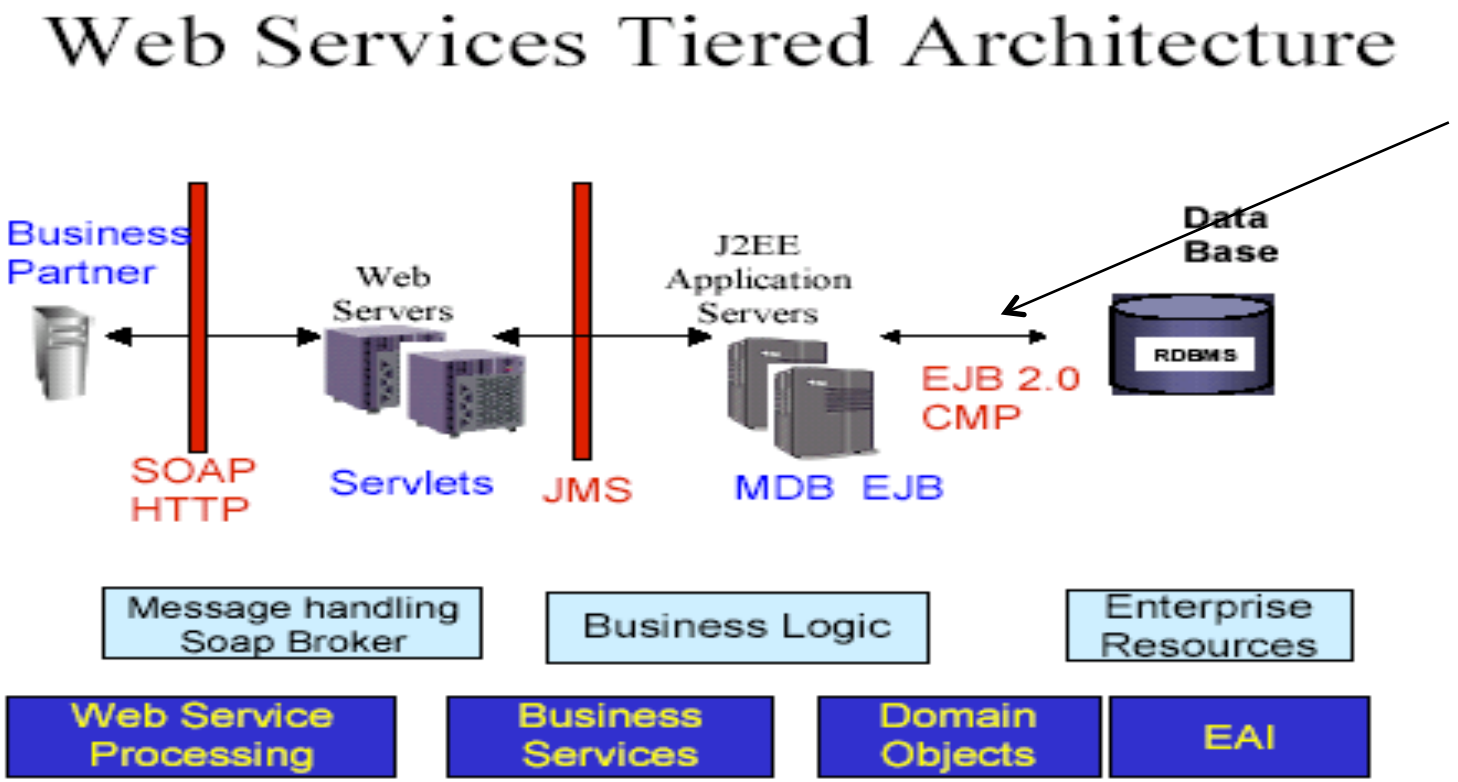
Java EE Tutorial (OLD)



From The Java EE Tutorial



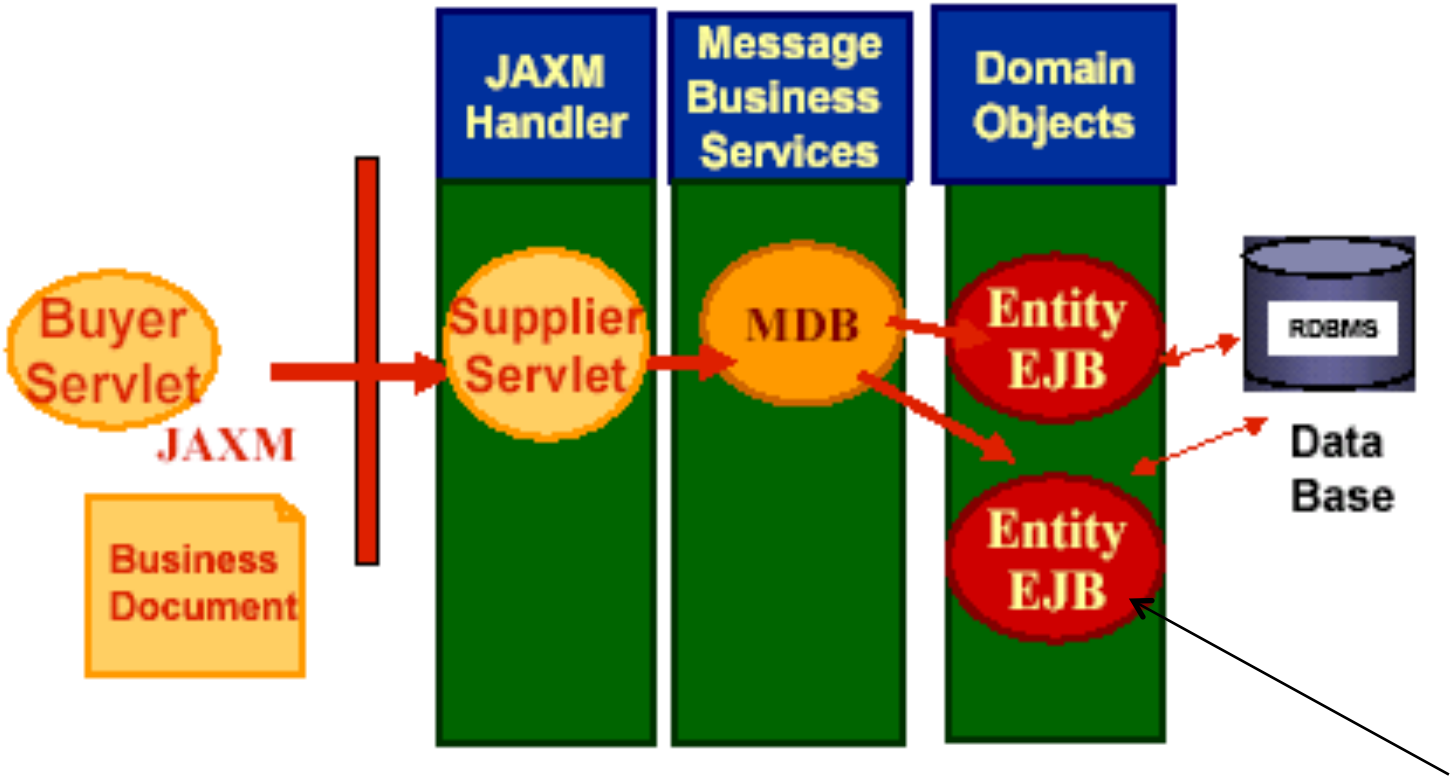
SOAP and Java EE



Replaced by JPA.

Slide from JAXM/JMS tutorial at Sun Microsystems

JAXM Web Services & J2EE



Slide from JAXM/JMS tutorial at Sun Microsystems

Replaced by JPA.

EJB Types (Old)

- Entity Beans
- Session Beans
- Message-Driven Beans

EJB Types (Old)

- Entity Beans
- Session Beans

} RMI-based server side components Accessed using distributed object Protocols (RMI IIOP)

- Message-Driven Beans

} New since EJB 2.0 Asynchronous server side component that responds to JMS asynchronous messages (Think provider like JAXM)

EJB Entity Beans

- Represent real world entities (customers, orders, etc.).
- Persistent objects typically stored in a relational database using CMP (Container Managed Persistence) or BMP (Bean Managed Persistence).
- The client sees no difference between CMP and BMP beans.
- CMP promotes component portability (more reliant on the container to handle detail).
- CMP uses its own Query Language EJB QL.
- CMP relies on an Abstract Schema in the deployment descriptor.

Implementing Entity and Session Beans

- Defining the component interfaces:
  - You may choose to define all or only some of these depending on how you want your bean used.
  - local interfaces do not require RMI overhead.
- Define a bean class.
- For entity beans define a primary key.

Implementing Entity and Session Beans

- Define the component interfaces
  - The remote interface specifies how the outside world can access the bean's business methods
  - The remote home interface specifies how the outside world can access the bean's life-cycle methods (for creating, removing and finding)
  - The local interface specifies how the inside world (same EJB container) can access the bean's business methods
  - The local home interface specifies how the inside world can access the bean's life-cycle methods

Implementing Entity and Session Beans

- Implement the bean
  - Fill in the code for the business and life-cycle methods.
  - It's not normal to directly implement the interfaces as we do in standard Java (though you must provide many of the methods). The calls to methods are not normal Java calls. They first go through the container.
  - Session beans implement javax.ejb.SessionBean.
  - Entity beans implement javax.ejb.EntityBean.
  - Both beans extend javax.ejb.EnterpriseBean

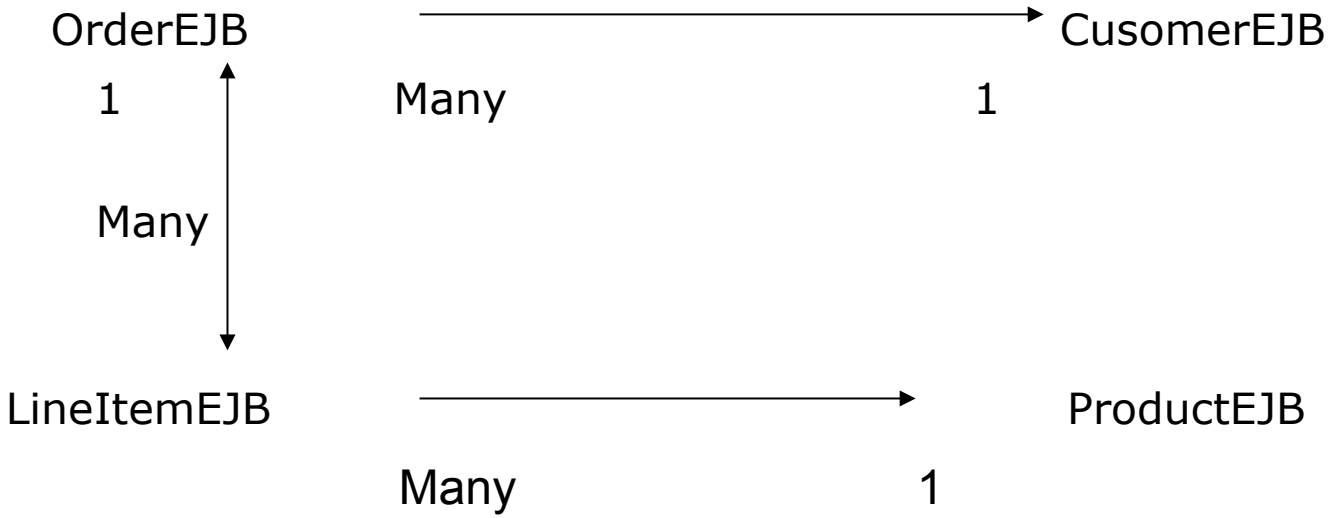
For Entity Beans

- Define a primary key class:
  - Required for entity beans.
  - Provides a pointer into the database.
  - Must implement Java.io.Serializable.
- The EJB instance represents a particular row in the corresponding database table.
- The home interface for the entity EJB represents the table as a whole (has finder methods.)

Entity Beans

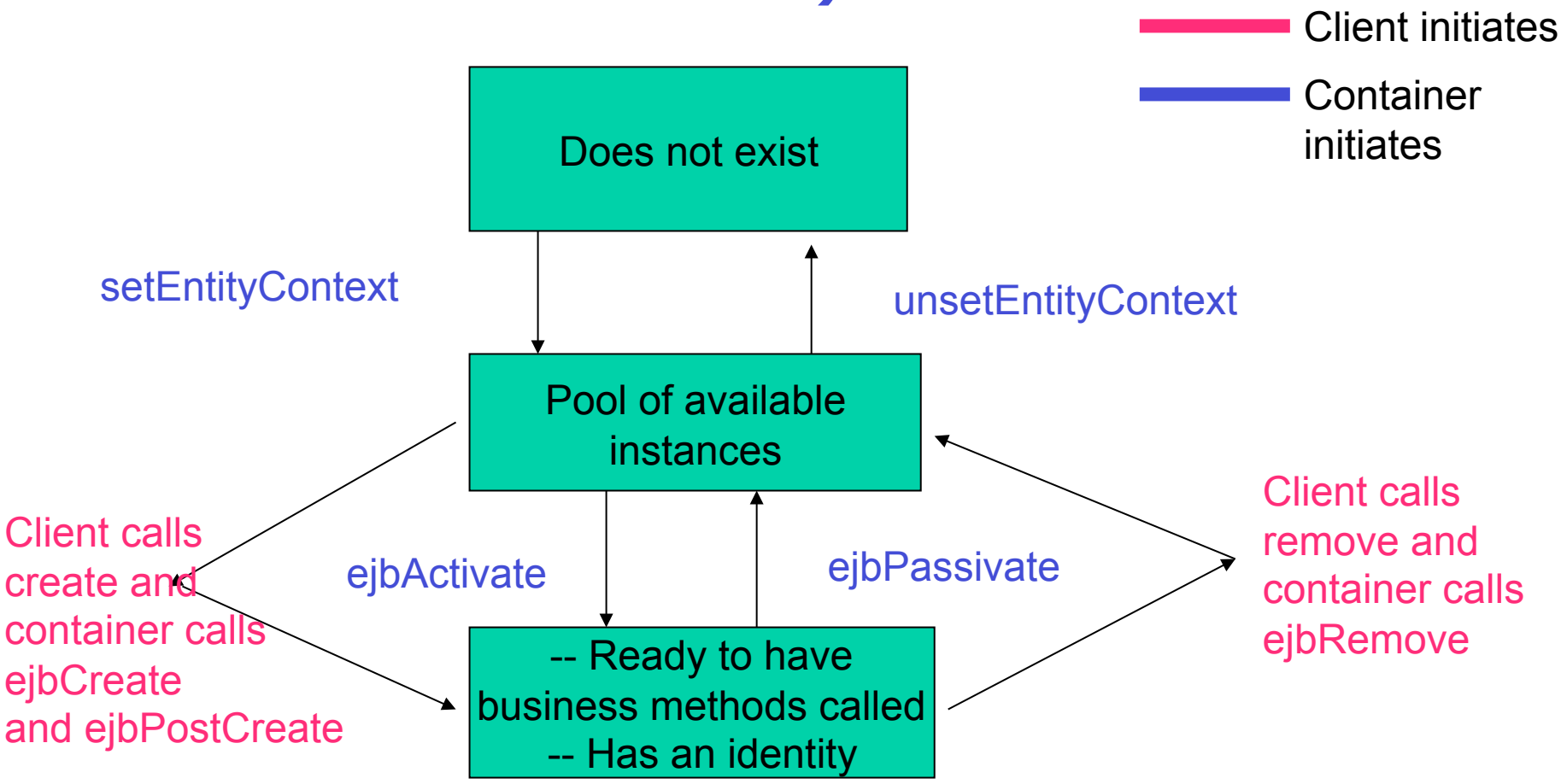
- Each entity bean has a unique identifier called its *primary key*
- The primary key can be used by the client to locate the bean
- Each bean represents a row in its table
- Rows may have columns that reference other entity beans (students take courses)
- These relationships may be managed by the bean or by the container (container managed relationships)

Entity Bean Relationships



A relationship field is like a foreign key in a database.
If a ⟶ b then a “knows about” or “holds a pointer to” b.

Entity Bean Life Cycle (From Sun)



A Typical Entity Bean Needs:

- A Home interface defining the create and finder methods.
- A Component interface defining the business methods a client may call.
- An implementation of the Component interface.
- Deployment descriptors.

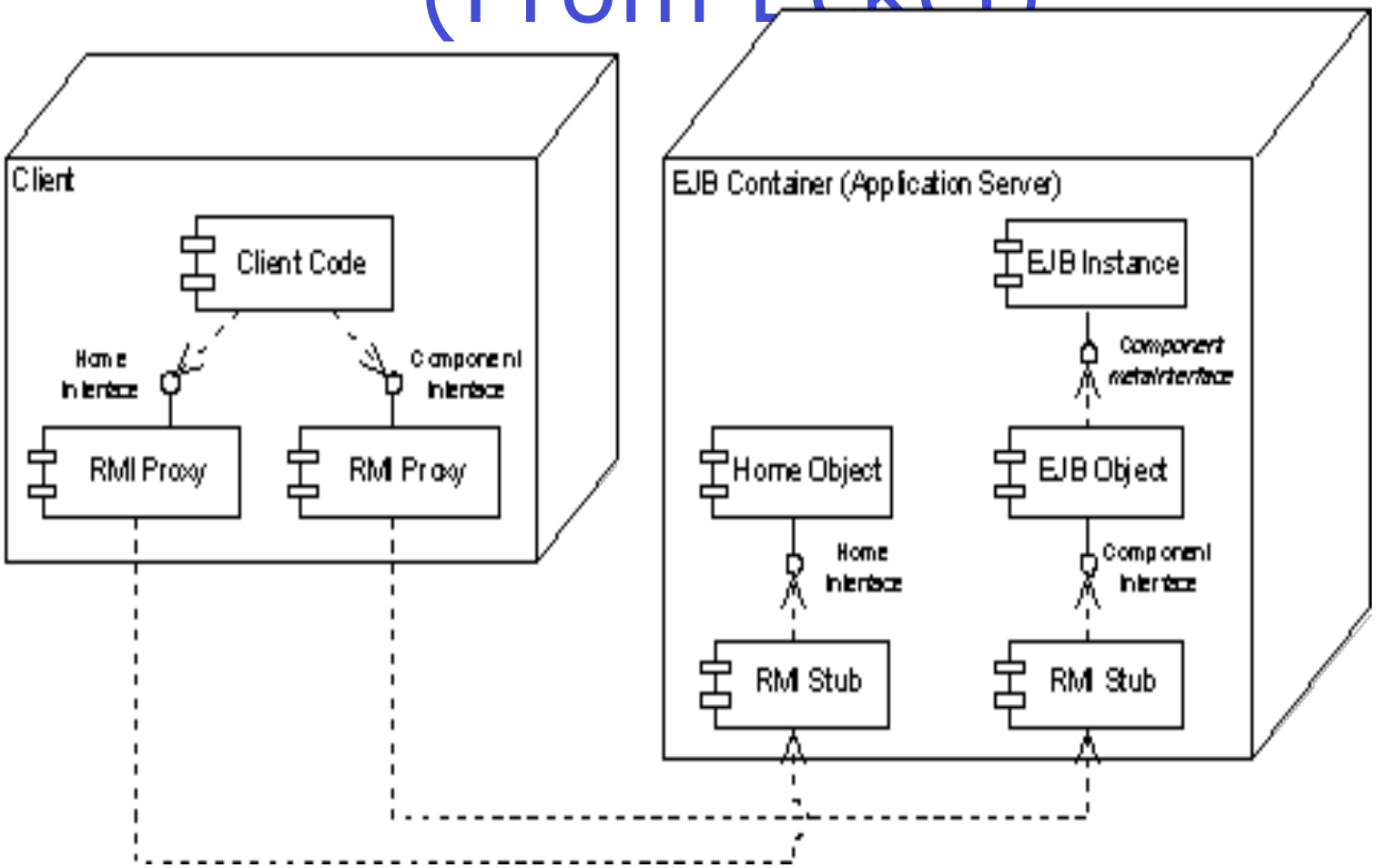
A Home Interface

```
import javax.ejb.*;                          // From Eckel
import java.util.Collection;
import java.rmi.RemoteException;

public interface MovieHome extends EJBHome {
  public Movie create(Integer id, String title)
    throws RemoteException, CreateException;

  public Movie findByPrimaryKey(Integer id)
    throws RemoteException, FinderException;
}
```

A Component Interface

```
import javax.ejb.*;                          // From Eckel
import java.rmi.RemoteException;

/**
 * A movie, with id and title.
 *
 * Note that there is no setId() method in the
 * interface, to prevent clients from arbitrarily
 * changing a movie's primary key.
 */
public interface Movie extends EJBObject {
  public Integer getId() throws RemoteException;
  public String getTitle() throws RemoteException;
  public void setTitle(String title)
    throws RemoteException;
}
```

The Container Implements the component interface (From Eckel)



```
import javax.ejb.CreateException;                    // From Eckel
import javax.ejb.EntityBean;
import javax.ejb.EntityContext;

public abstract class MovieBean implements EntityBean {

  // Container notifications methods
  public Integer ejbCreate(Integer id, String title) throws CreateException {
    if (id == null) throw new
        CreateException("Primary key cannot be null");
    if (id.intValue() == 0)
      throw new CreateException("Primary key cannot be zero");

    setId(id);
    setTitle(title);

    return null;
  }
```

```
  // From Eckel

  public void ejbPostCreate(Integer id, String title) {}    // Called by
  public void ejbLoad() {}                                  // container
  public void ejbStore() {}
  public void ejbRemove() {}
  public void ejbActivate() {}
  public void ejbPassivate() {}
  public void setEntityContext(EntityContext ctx) {}
  public void unsetEntityContext() {}

  // Business methods provided by container
  public abstract void setId(Integer id);
  public abstract String getTitle();
  public abstract void setTitle(String title);
  public abstract Integer getId();
}
```

Deployment Descriptor

```
<ejb-jar>                                                    // From Eckel
  <enterprise-beans>
    <entity>
      <ejb-name>Movie</ejb-name>
      <home>javatheater.ejb.MovieHome</home>
      <remote>javatheater.ejb.Movie</remote>
      <ejb-class>
        javatheater.ejb.implementation.MovieBean
      </ejb-class>
      <persistence-type>Container</persistence-type>
      <prim-key-class>java.lang.Integer</prim-key-class>
      <reentrant>False</reentrant>
      <cmp-version>2.x</cmp-version>
      <abstract-schema-name>Movie</abstract-schema-name>
      <cmp-field><field-name>id</field-name>-</cmp-field>
      <cmp-field><field-name>title</field-name>-</cmp-field>
      <primkey-field>id</primkey-field>
    </entity>
  </enterprise-beans>
</ejb-jar>
```

Client (From Eckel)

```
import javax.naming.*;
import javatheater.ejb.*;

public class MovieClient {
  public static void main(String[] args) throws Exception {
    javax.naming.Context initial =
      new javax.naming.InitialContext();

    Object objRef = initial.lookup("javatheater/Movie");
    MovieHome movieHome =
      (MovieHome) javax.rmi.PortableRemoteObject.narrow(
        objRef,
        MovieHome.class);
```

```
    // Generate a primary key value
     int pkValue =
      (int) System.currentTimeMillis() % Integer.MAX_VALUE;

     // Create a new Movie entity
     Movie movie =
       movieHome.create(
         new Integer(pkValue), "A Bug's Life"
       );

     // As a test, locate the newly created entity
     movie =
       movieHome.findByPrimaryKey(new Integer(pkValue));

     // Access the bean properties
     System.out.println(movie.getId());
     System.out.println(movie.getTitle());

     // Remove the entity
     movie.remove();
   }
}
```

Entity Beans Are A Lot Of Work

- The JPA 2.0 specification was released in 2009.
- Implemented by:

  Hibernate

  EclipseLink

  Open JPA

  Object DB

  TopLink and many others

JPA Notes From Oracle

JPA simplifies the programming model for entity persistence:

- Requires fewer classes and interfaces
- Virtually eliminates lengthy deployment descriptors through annotations
- Eliminates the need for lookup code
- Adds support for named (static) and dynamic queries.
- Provides a Java Persistence query language -- an enhanced EJB QL Makes it easier to test entities outside of the EJB container.
- Can be used outside of the container
- Can be used with pluggable, third-party persistence providers

In Class Exercise

- Using Netbeans 7.0
- Using EclipseLink default JPA 2.0 Implementation
- See course schedule for link.