

# Introduction to Enterprise JavaBeans (EJB)



---

Kimberly Bobrow Jennery

Sun Microsystems

[kimberly.bobrow@sun.com](mailto:kimberly.bobrow@sun.com)

[kimberly@bobrow.net](mailto:kimberly@bobrow.net)

[kimberly@jennery.com](mailto:kimberly@jennery.com)





# Overview of This Session

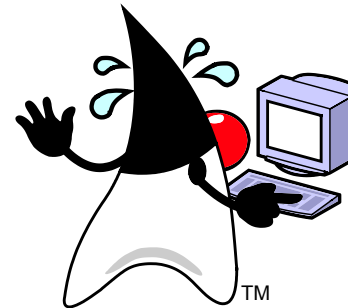
---

- What is an EJB component?
  - What's the point, anyway?
  - Classes and interfaces involved
- Session Beans
  - Stateless
  - Stateful
- Entity Beans
  - Container-managed persistence (CMP)
  - Bean-managed persistence (BMP)
- Message Driven Beans (MDBs)
  - Java Messaging Service (JMS) overview
- EJB Lifecycle
- EJB Limitations

# Why Use EJB Components?

- Most enterprise-level applications have similar infrastructure needs
- J2EE provides a framework for:
  - J2EE providers – vendors of J2EE products that include component containers
  - Application component providers – produce components and applications which run in J2EE component containers
- Most server-provided services are defined or set declaratively – no coding
- EJBs make a difficult job tractable

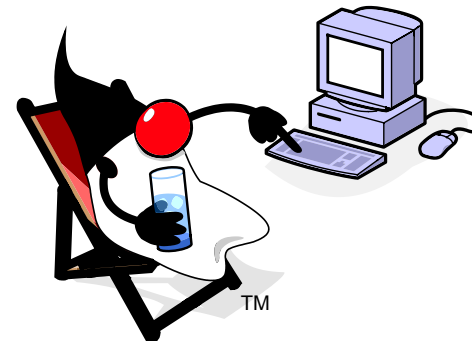
## Build From the Ground Up



### Developer's Checklist

- Business service
- Persistence
- Transaction
- Threading
- Security
- Networking
- Service publishing

## Use Application Component Server



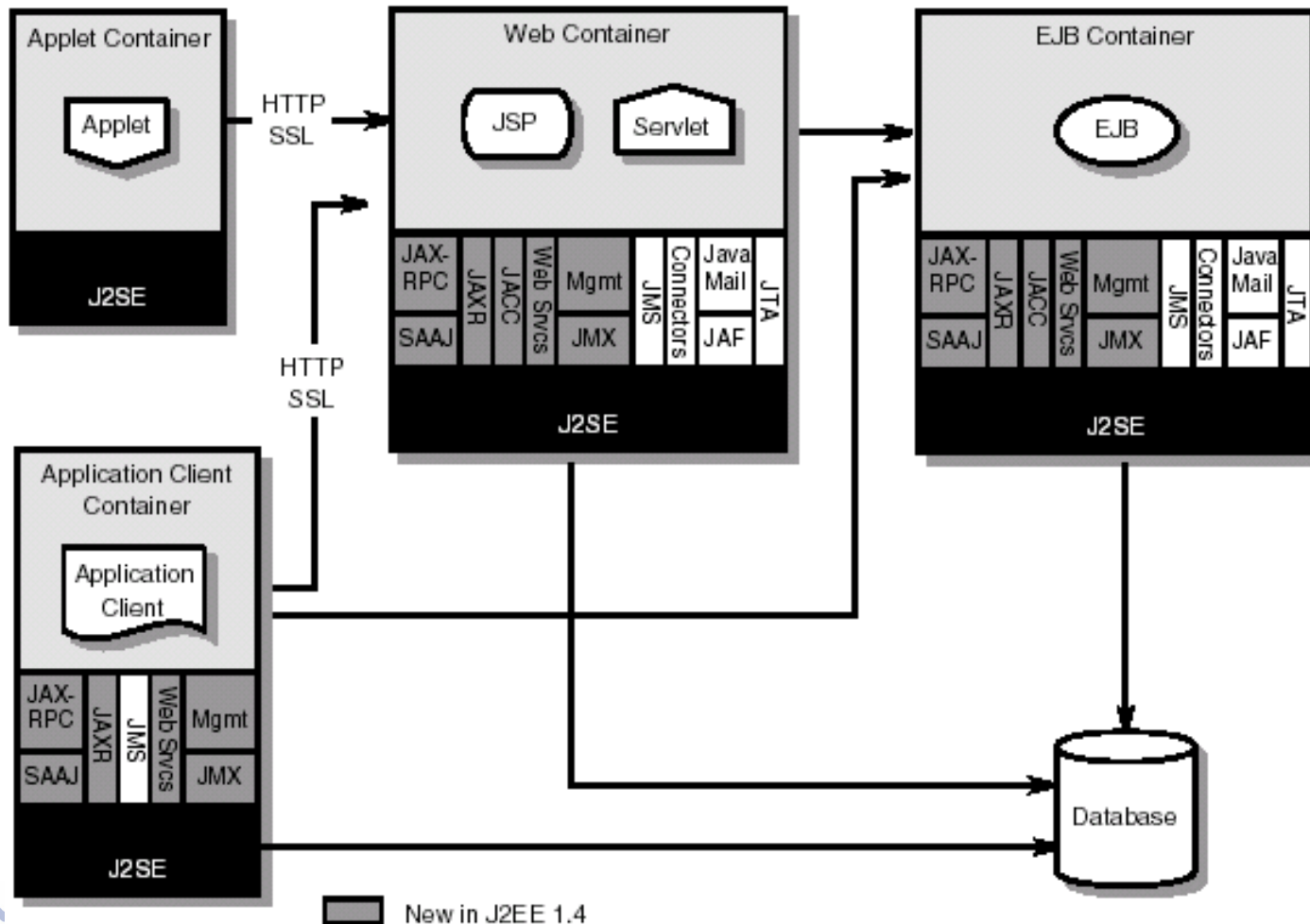
### Developer's Checklist

- Business service

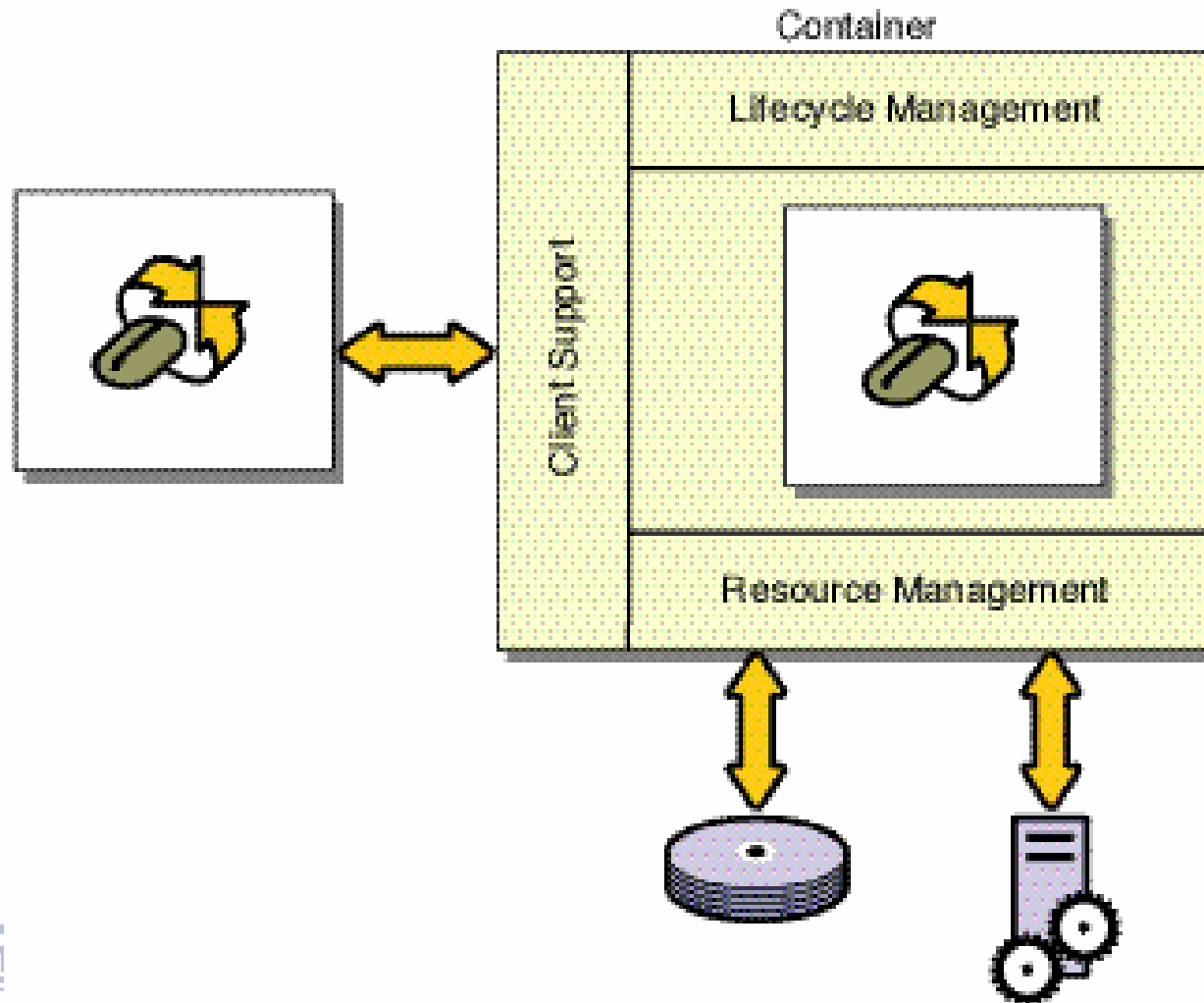
### Services Provided by Server

- Persistence
- Transaction
- Threading
- Security
- Networking
- Service publishing

# J2EE Architecture & Containers



# Role of the EJB Container in the EJB Component Model





# Containers

---

- Containers provide the runtime support for J2EE application components
- Containers provide a view of the underlying J2EE APIs to the application components
- J2EE application components never interact directly with other J2EE application components
- They use the protocols and methods of the container for interacting with each other and with platform services



# Punch Line

---

- You develop an EJB component, focusing on the business logic for your business.
- The container intercepts and/or dispatches every method call to its components, and takes care of all the underlying services (security, persistence, transactions, *etc.*)



# Three Types of EJBs

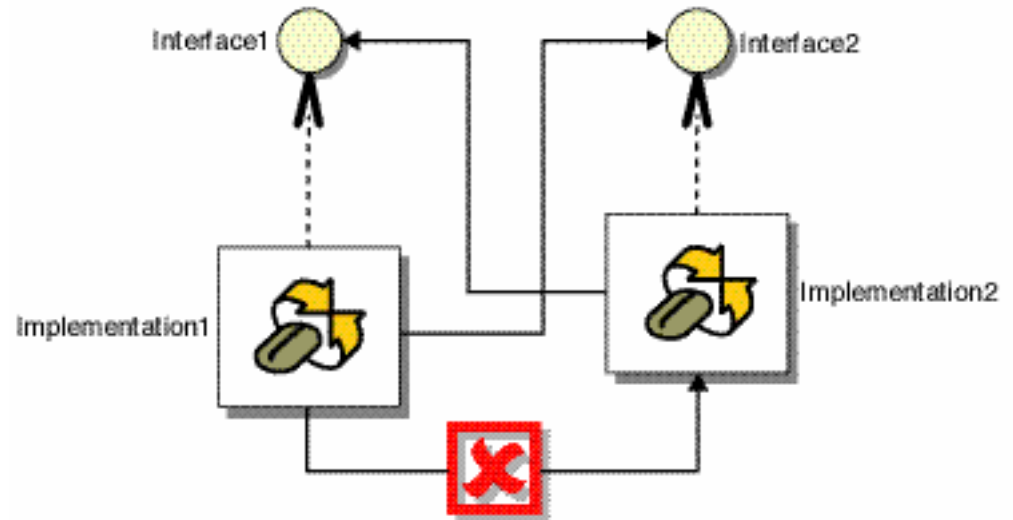
---

- Session Beans
  - Process oriented service providers
  - Examples: Shopping carts, Credit Authorization
- Entity Beans
  - Represent data in the persistent store.
  - May represent a row in a database, or data from multiple sources
  - Examples: Customer, Account
- Message-Driven Beans
  - JMS consumer
  - Never called directly by clients, only receives messages from a JMS messaging service
  - Example: Transaction Logging
- Coding is similar for all three types. Many of the development concepts are the same.

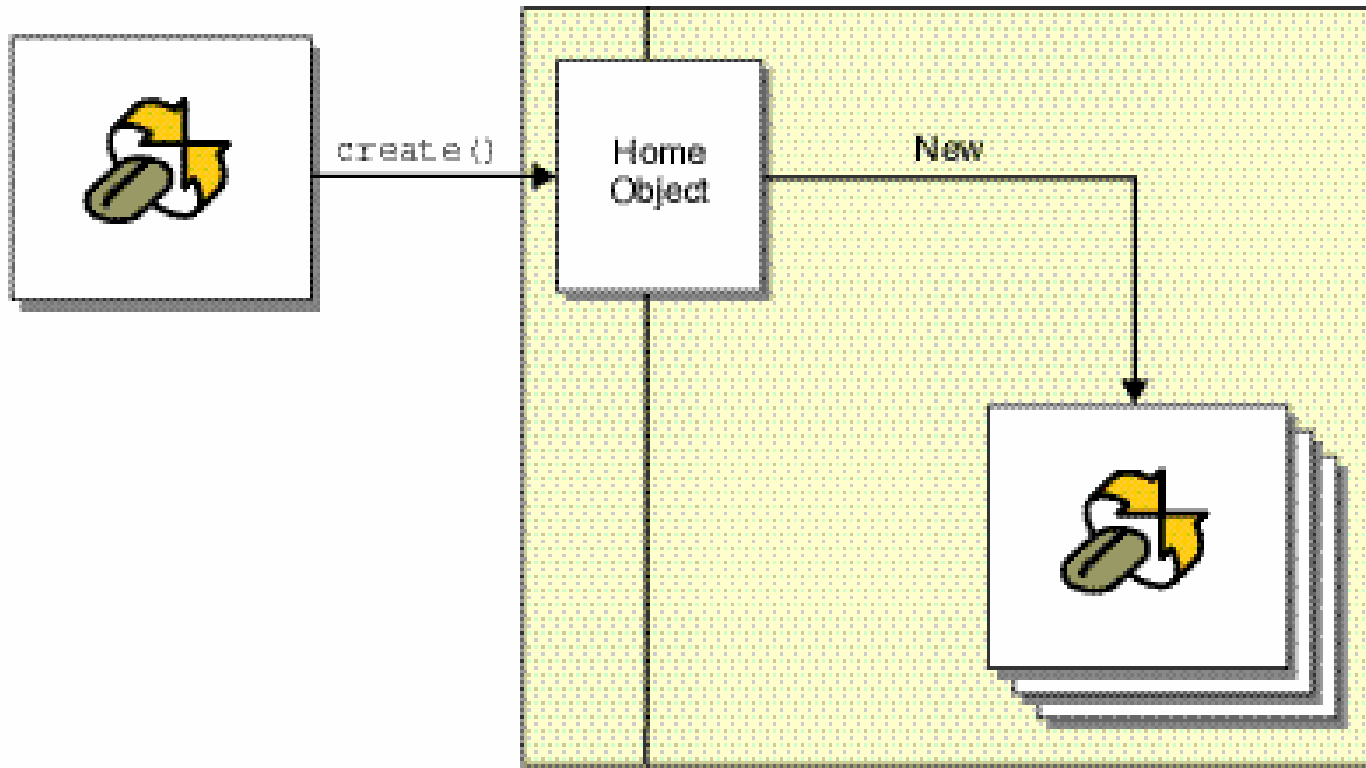


# Building an EJB Component

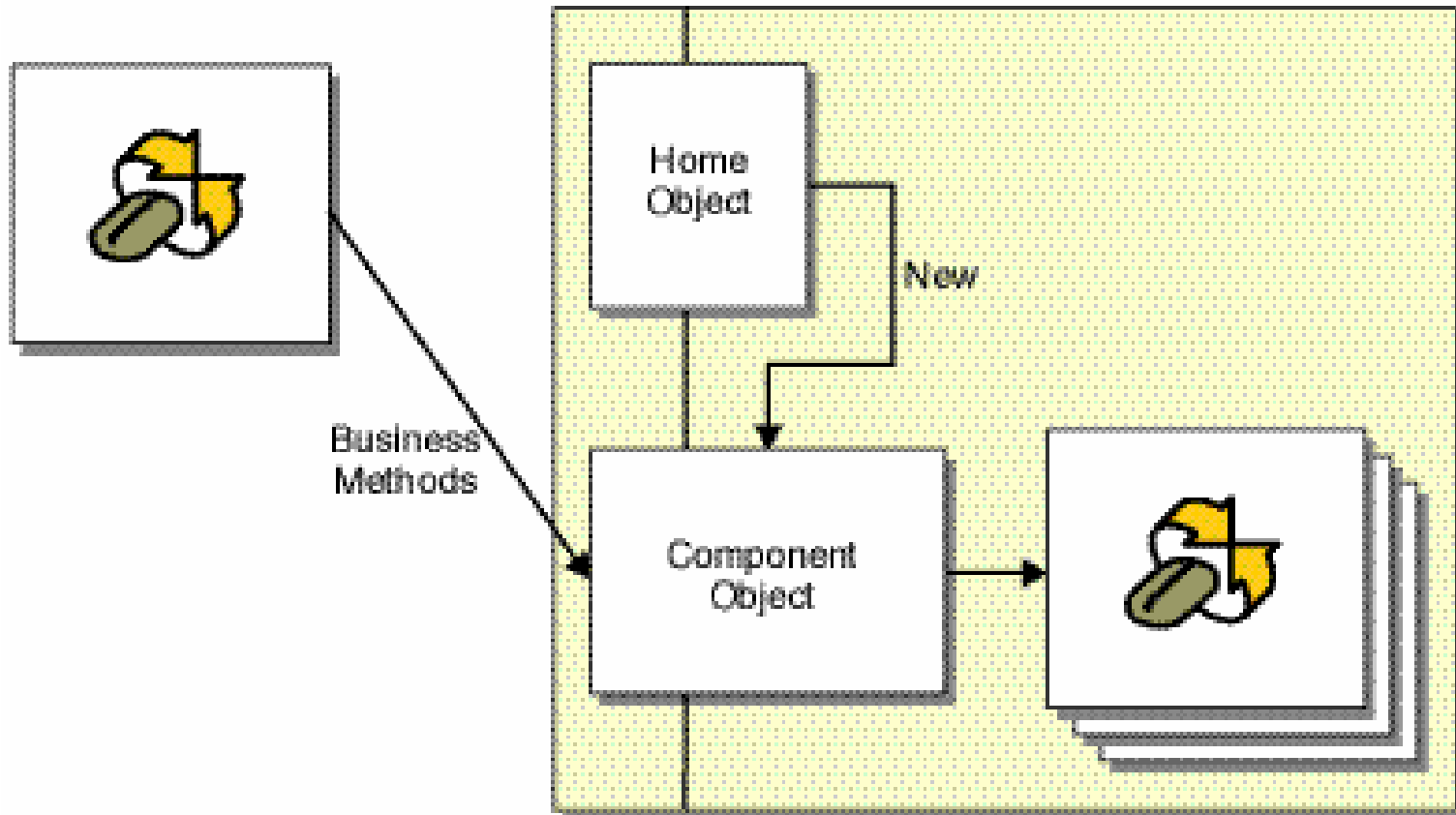
- Only the container EVER talks directly to an EJB component
- Clients communicate with EJBs entirely through interfaces
- Therefore, besides coding the implementation of an EJB in a Java class, you must also code the interfaces that clients will use to interact with them



# Home Object as a Factory for EJB Component Model Instances



# EJB Object as a Proxy for the EJB Component Model Implementation





# Building an EJB Component

---

1. Code:
  - a. The bean implementation class with the business logic
  - b. For each view (remote or local) code two interfaces
    - i. Home interface – for factory methods
    - ii. Component interface – for business methods
2. Create an XML deployment descriptor that provides information about the bean to the container. Must be named `ejb-jar.xml`
3. Package the bean class, interfaces, any additional helper or primary key classes, and the deployment descriptor into an `ejb-jar` file. This is a packaged component.
4. Deploy the bean. This process will be different for each vendor's server.



# Building an EJB Component

---

- Example case:

- FortuneTeller

- First step: create a session bean with one business method which returns a random fortune String from an array of Strings

- We'll take a look at this sample as we go through the steps of building an EJB component



# Building an EJB Component

---

1.a Code the bean implementation class with the business logic:

- Create class: FortuneTellerBean.java
- Must *implement* javax.ejb.SessionBean, EntityBean or MessageDrivenBean
- Implement container callback methods:
  - public void ejbActivate()
  - public void ejbRemove()
  - public void setSessionContext()
  - public void ejbCreate()
  - *etc.*



Add business methods

# The Implementation Class – Container Callbacks

---

- **setSessionContext or setEntityContext**
  - Save the context if you need it in other methods
  - EJBContext gives Bean access to Transaction, Security, and other information
- **unsetSessionContext or unsetEntityContext**
  - Set saved context object to null
  - Release any resources allocated during setSessionContext or setEntityContext

# The Implementation Class – Container Callbacks

- **ejbActivate/ejbPassivate**
  - Called when container brings instance into or out of active state from the bean pool
    - State of the bean is being preserved
  - Never called for stateless session beans
  - Often empty
  - If you have open resources which cannot be stored during passivation, you must release them and reacquire them during these methods

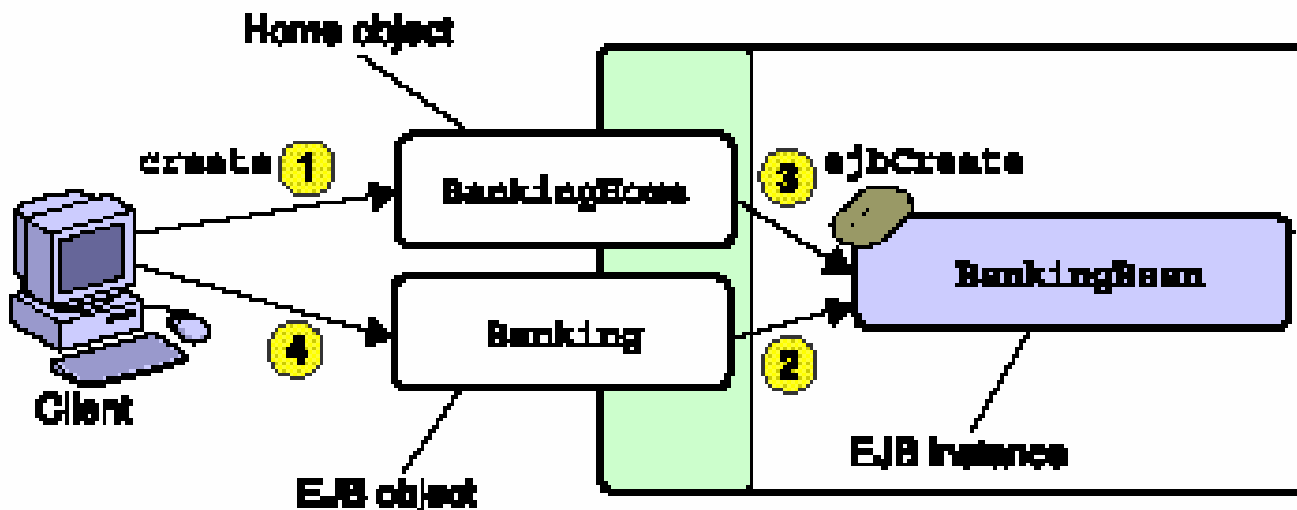


# The Implementation Class – Container Callbacks

- **ejbCreate** – Matches create(...) methods from the Home interface
  - Creates and initializes the bean instance
    - Verify parameters, throw CreateException if invalid
  - Often empty for stateless session bean
  - **BMP** – Perform SQL INSERT, return primary key
    - **CMP** – return null or 0 (according to data type)
  - **ejbPostCreate** – Entity Beans
    - Called after EJB created in container and after it has its identity
    - Do things that require EJB reference – such as establish relationships with other entities
  - **ejbRemove**
    - **BMP** – Perform SQL DELETE

# Creator Methods

1. The client calls a create method on a home object.
2. An EJB object is created and associated with an implementation instance.
3. The home object or container calls a corresponding ejbCreate method on the implementation instance.
4. The home object returns the EJB object to the client, and the client makes business method calls on the EJB object.



# The Implementation Class – Container Callbacks

- **ejbLoad / ejbStore**
  - Used with entity beans
  - Reading from/writing to database
  - Usually empty for CMP – that's the CM in CMP!
  - BMP – Usually JDBC calls
    - `SELECT * FROM table WHERE key = primary`
      - ✓ Set bean fields from database data
    - `UPDATE table SET col1=? col2=?... WHERE key = primary`
      - ✓ Set database values from bean fields

# The Implementation Class – Home Methods

- These methods are implementations of “home” methods you create in the home interface (which will have similar signatures)
  - Method names prefix with “ejb” and next letter uppercased
  - create() becomes ejbCreate()
  - findByPrimaryKey() becomes ejbFindByPrimaryKey
  - Home methods prefix with “ejbHome”
    - ✓ totalAllAccounts() becomes ejbHomeTotalAllAccounts()
- Return types:
  - Session ejbCreate() returns void
  - Entity ejbCreate returns primary key type
    - ✓ BMP returns primary key instance, CMP returns null
  - Entity ejbFindXXX() returns primary key type or Collection (of primary key type)

# The Implementation Class – Business Methods

- These methods are implementations of business methods you create in the component interface (which will have similar signatures)
  - Same signature as the component interface, except in the implementation class:
    - ✓ Don't throw new RemoteException()
      - ❖ Pick a better, more applicable exception. Remote exception is for network/remote exceptional conditions
    - ✓ If you call another EJB remote, and catch a RemoteException, it's ok to propagate it



# Building an EJB Component

---

1.b For each view (remote or local) code two interfaces:

- **Home interface:**

- Factory for creating, finding, deleting EJBs
- Looked up using JNDI

- **Component interface**

- Client makes method call on these interfaces



# Local vs. Remote

---

- Remote
  - Distributed calls (RMI/CORBA)
  - Pass-by-value (Serialized)
- Local (EJB 2.0)
  - Must be in same JVM
  - Pass-by-reference
  - No RemoteException
  - Required for CMR entity beans
- EJB can have either or both
- Remote and local interfaces can have the same or different methods
- Typical best practice usage is to have a session bean with a remote interface talk to entity beans or other session beans with (only) local interfaces



# Building an EJB Component

---

1.b.i For each view (remote or local) code two interfaces:

- Home interface (remote and/or local)
  - Extends EJBHome or EJBLocalHome
    - FortuneTellerHome.java / FortuneTellerLocalHome.java
    - One for each interface
  - Requirements
    - Create methods (not required, but common for entity beans)
    - Entity beans require findByPrimaryKey
  - Optional
    - Several create methods
      - ✓ Just one create method with no args for stateless session bean
    - Entity Beans
      - ✓ Other finder methods
    - ✓ Home methods





# EJBHome Create Methods

---

- `create(...)`
  - Returns EJB Component Interface
  - Throws `CreateException`
    - And `RemoteException` if not Local
- How Many?
  - Stateless Sessions: One, with no arguments
  - Stateful Sessions: One or more, with appropriate arguments
  - Entity Beans: Zero or more, with appropriate arguments



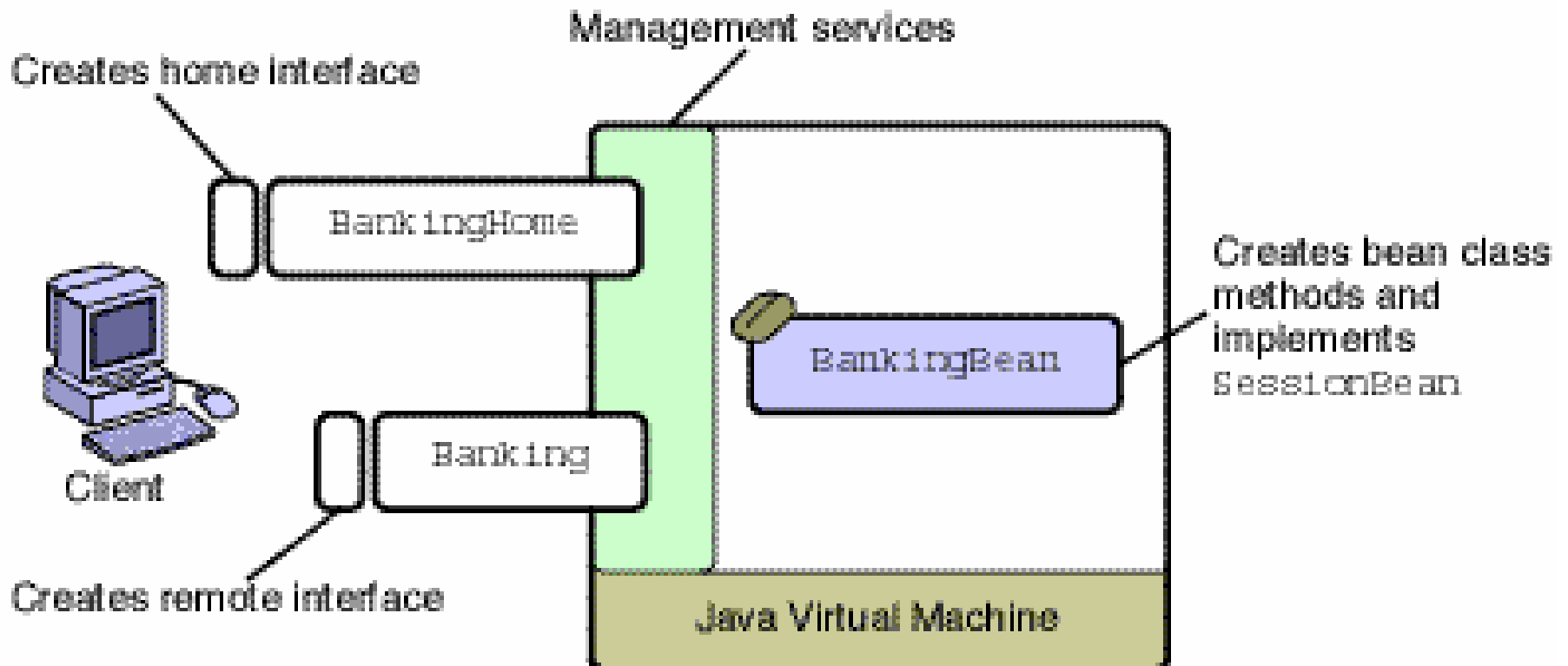
# Building an EJB Component

---

1.b.ii For each view (remote or local) code two interfaces:

- **Component interface (remote and/or local)**
  - Extends EJBObject or EJBLocalObject
    - FortuneTeller.java / FortuneTellerLocal.java
  - Add your methods
    - Session beans have business methods for clients
    - Entities usually have get/set methods for attributes/fields
    - If Remote, methods must throw `java.rmi.RemoteException`
  - All arguments and return values for Remote must be **Serializable**

# Bean Provider Responsibilities



# Developer Implemented Methods

Home Interface	Component Interface	Bean Class	Method Type
create		ejbCreate	Factory
		setSessionContext	Infrastructure
	<businessMethods>	<businessMethods>	Business
		ejbPassivate	Infrastructure
		ejbActivate	Infrastructure
remove	remove	ejbRemove	Factory

Called by client

Called by container



# Building an EJB Component

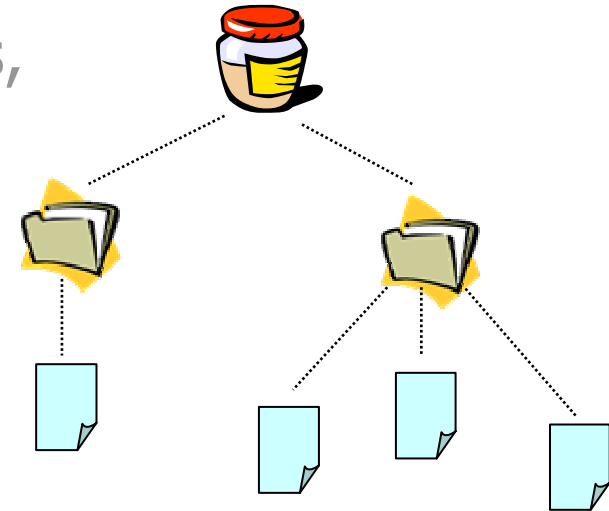
---

2. Create an XML deployment descriptor that provides information about the bean to the container
  - EJB jar file: META-INF/ejb-jar.xml
  - Declares each EJB, its related classes, and:
    - References to other EJBs (ejb-ref)
    - Environment properties (env-entry)
    - Database and other resources (resource-ref)
    - Security restrictions
    - Transaction settings
  - CMP Definitions
    - Fields and Queries
  - Vendor specific descriptor
    - Server specific configurations, tunings, *etc.*
  - There are tools to assist in the development of deployment descriptors – they do NOT need to be coded by hand, unless you really want to!

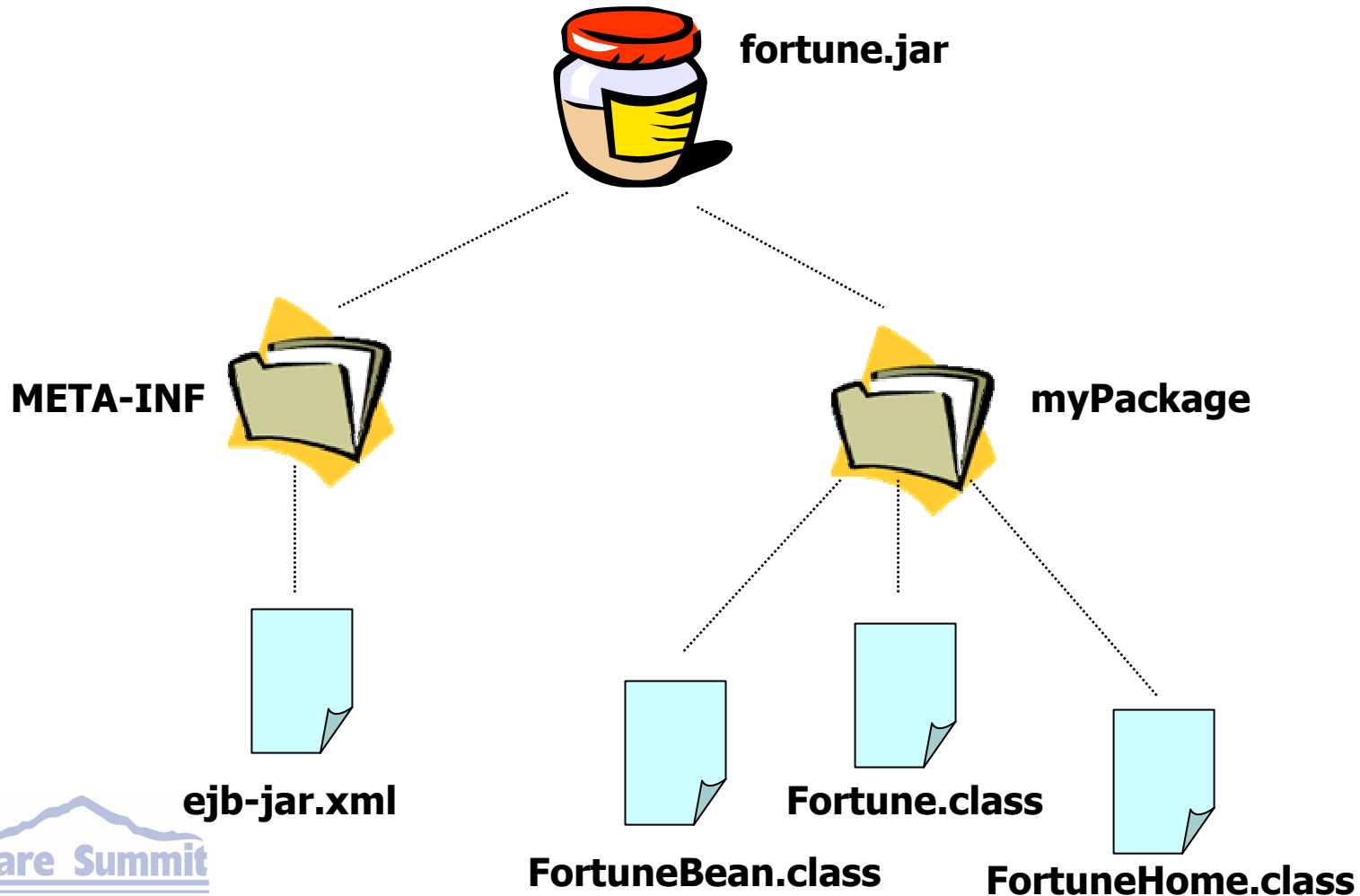
# Building an EJB Component

3. Package the bean class, interfaces, any additional helper or primary key classes, and the deployment descriptor into an `ejb-jar` file. This is a packaged component.

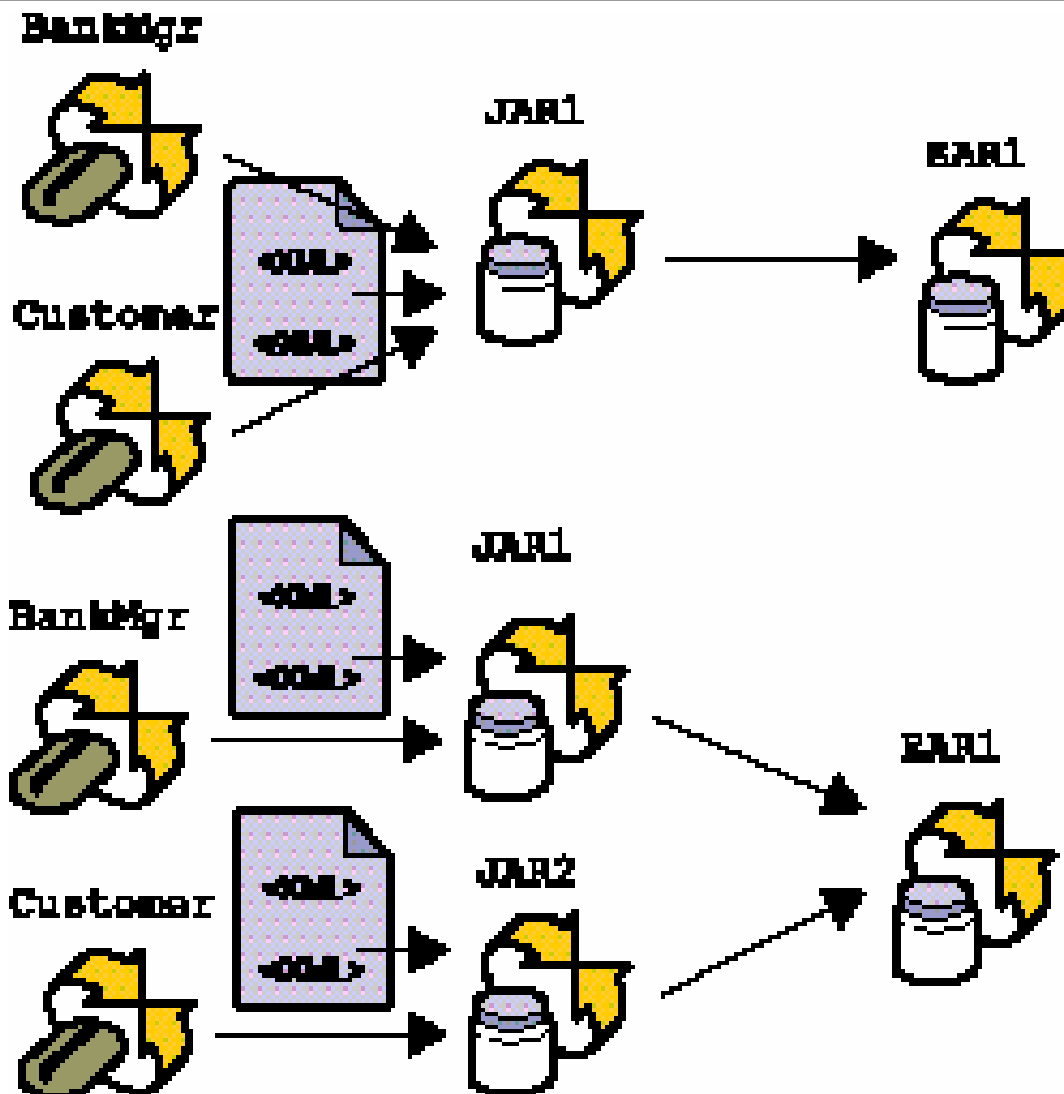
- You can use any `jar` utility for this process.
- There are many tools which can assist in the packaging process and will create the `ejb-jar` file for you.
- This process is sometimes combined with deployment, depending on the tool(s) you are using



# Packaged EJB Component



# Packaging Options







# Building an EJB Component

---

Deploy the bean. This process will be different for each vendor server.

- The tools provided by your vendor server will dictate how you deploy your components.
- Discussion of the details of any particular deployment environment is beyond the scope of this talk.



# Session Beans

---

- Service providers/request handlers
- Often are used to provide coarse-grained access to back-end services
- Commonly used as the interface to entity beans



# Session Beans and State

---

- There are two kinds of session beans:
  - Stateless
    - Most common
    - Any session bean from the “pool” can service a request
    - Most efficient
  - Stateful
    - State maintained between method calls from the same client
    - Each client is guaranteed the same session bean for subsequent requests
    - “People don’t kill people, stateful session beans kill people” – Dion Gillard, CSS 2001



# Let's Look at Some Sample Code

---

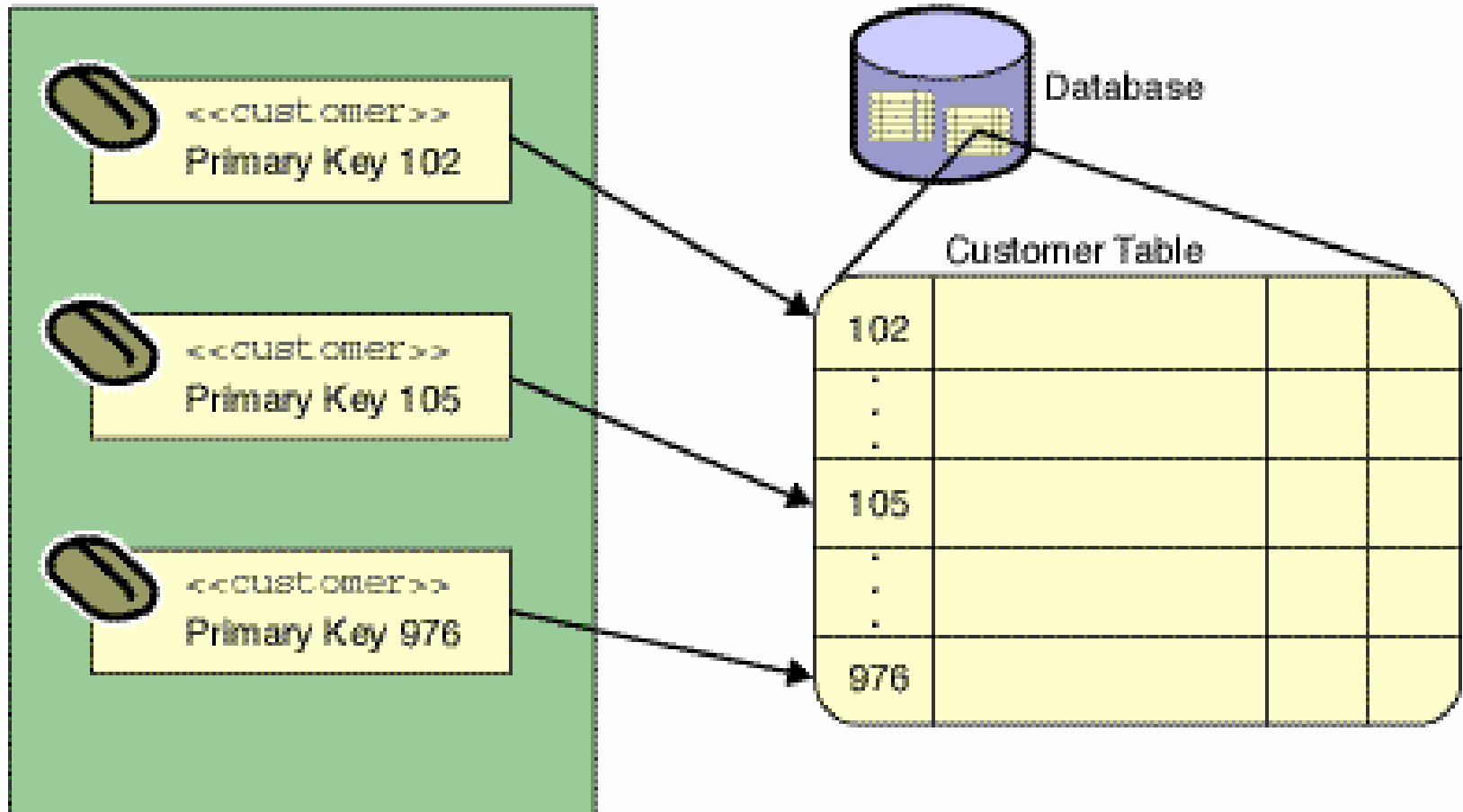


# Entity Beans

---

- Durable persistence
  - Survives container crashes
  - Container-managed (CMP)
  - Bean-managed (BMP)
- Unique primary keys identify individual entities
- Can have container-managed relationships with other entities (CMR)

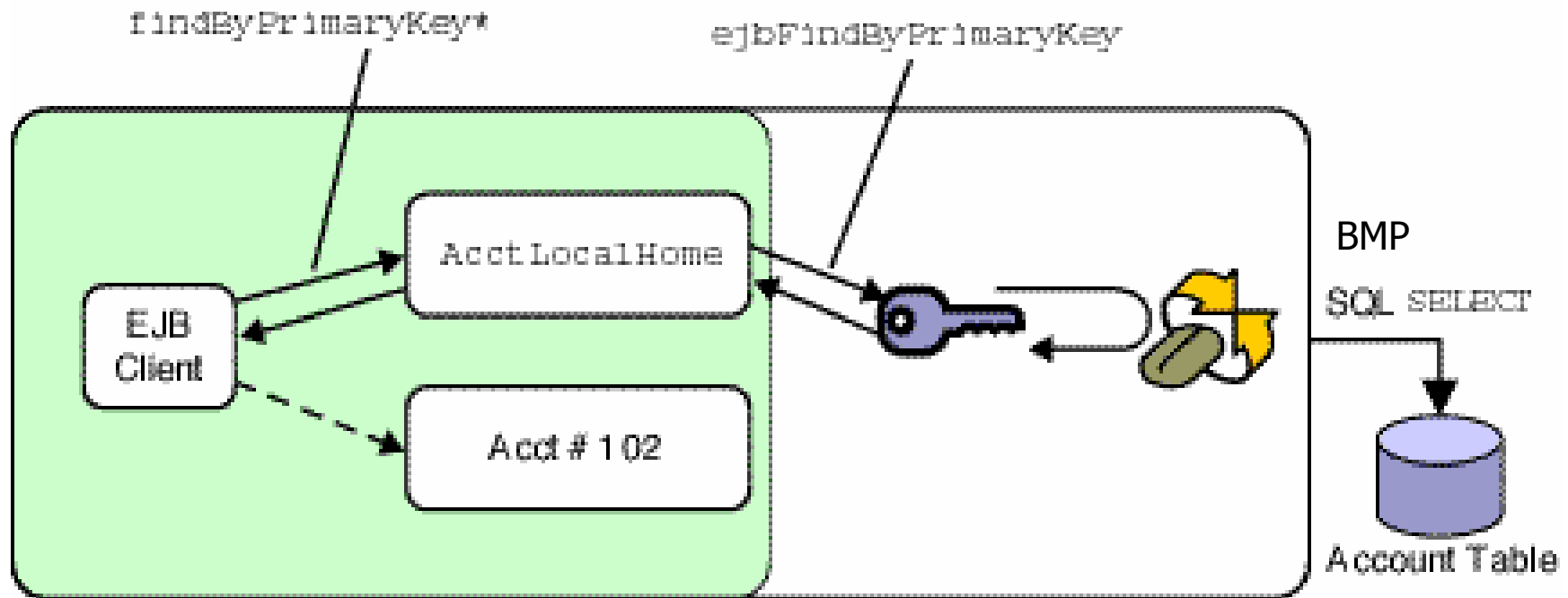
# Primary Key and Database Rows



# Entity Home Finder Methods

- To find a particular entity
  - SQL: `SELECT * FROM table WHERE <condition>`
- Must have `findByPrimaryKey(PrimaryKeyType)`
- Others named `findXXX(...)`
  - Appropriate arguments to perform query
  - Returns component interface – local or remote
    - Or Collection of component interfaces
  - Throws `FinderException`
    - And `RemoteException` if not local

# ejbFindByPrimaryKey

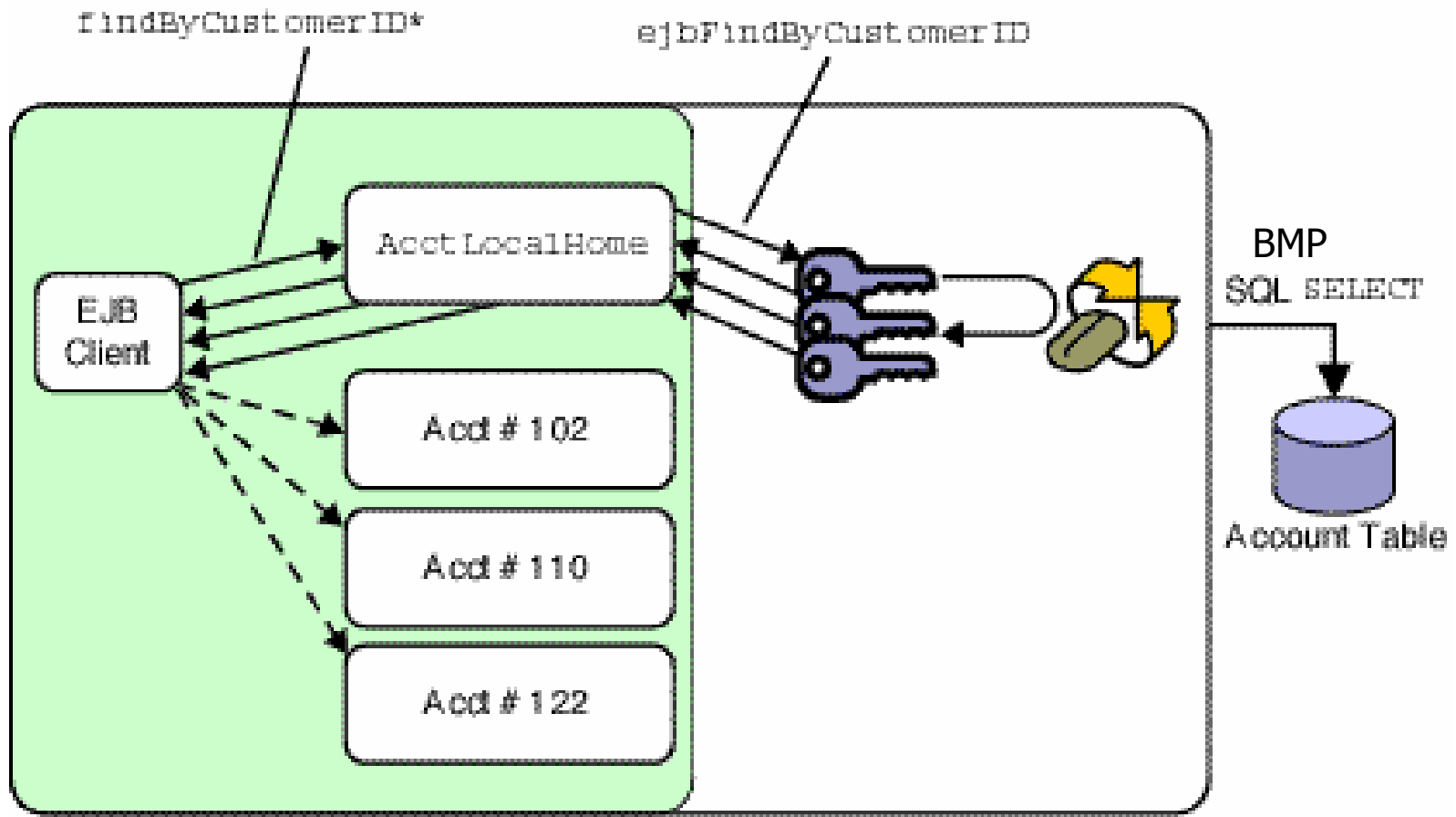


```
* findByPrimaryKey (int acctNum)
```

For CMP: EJB-QL and container-managed retrieval



# Multi-object Finder Methods





# Entity Home Methods

---

- General methods appropriate to the entity
- Do not require a specific entity instance
  - Similar in concept to static methods on an object
- Only available in EJB 2.0 and above
  - In prior implementations, this functionality was typically performed in a companion session bean



# Entity Primary Key

---

- Unique identifier for entity instance
- Can be a standard Java class
  - Usually String, BigDecimal, Integer, Long, *etc.*
- Can be a custom class
  - Compound keys
    - More than one database column defines uniqueness
      - ✓ First Name, Last Name
      - ✓ Year, Month, Day, Hour, Minute
    - Serializable & properly implemented equals() and hashCode() methods
      - ✓ Immutability and toString() are also helpful

# Container-managed Persistence

- Each attribute has get/set methods
  - Attributes are 'abstract'
  - Implementation of attributes done by the container
- You bind an attribute to database using EJB-Query Language (EJB-QL )
  - SQL-like
  - Defined in deployment descriptor
- Select methods allow EJB-QL to be used from business methods
- Can also set up relationships with other entity beans (CMR)

# Container-managed Relationships

- Natural relationships occur between entities:
  - Customer has multiple accounts
  - An account has an owner (Customer)
- CMR relationships are established:
  - with set/get methods in the implementation class to indicate the presence of an “attribute”
    - `getAccounts()`
  - With deployment descriptor entries indicating the type of relationships between related entities (one-to-one, one-to-many, *etc.*) whether deletion is prohibited or cascades, *etc.*
  - During `ejbPostCreate()`



# Container-managed Relationships

---

- Entity beans without a local interface can only have unidirectional relationships from itself to another entity bean.
- The lack of a local interface prevents other entity beans from having a relationship with an entity bean.
- Related entity beans must be packaged together – the deployment descriptor for the ejb-jar file has the relationship information between the beans.

# Bean-managed Persistence

- You write the data CRUD code (Create, Retrieve, Update, Delete) – usually database, but could be something else
- JDBC and SQL in the following container callback methods:
  - `ejbCreate`
  - `ejbRemove`
  - `ejbLoad`
  - `ejbStore`
- Good if:
  - Binding is too complex for CMP/EJB-QL
  - Persistence is not to a database
- Disadvantage:
  - You lose all the advantages of the container's optimizations for caching, loading, *etc.*

■ In general, CMP is the "recommended" way



# Let's Look at Some Sample Code

---





# Message-Driven Beans (MDB)

---

- Asynchronous communication
- Relevant JMS information
- Implementing MDBs

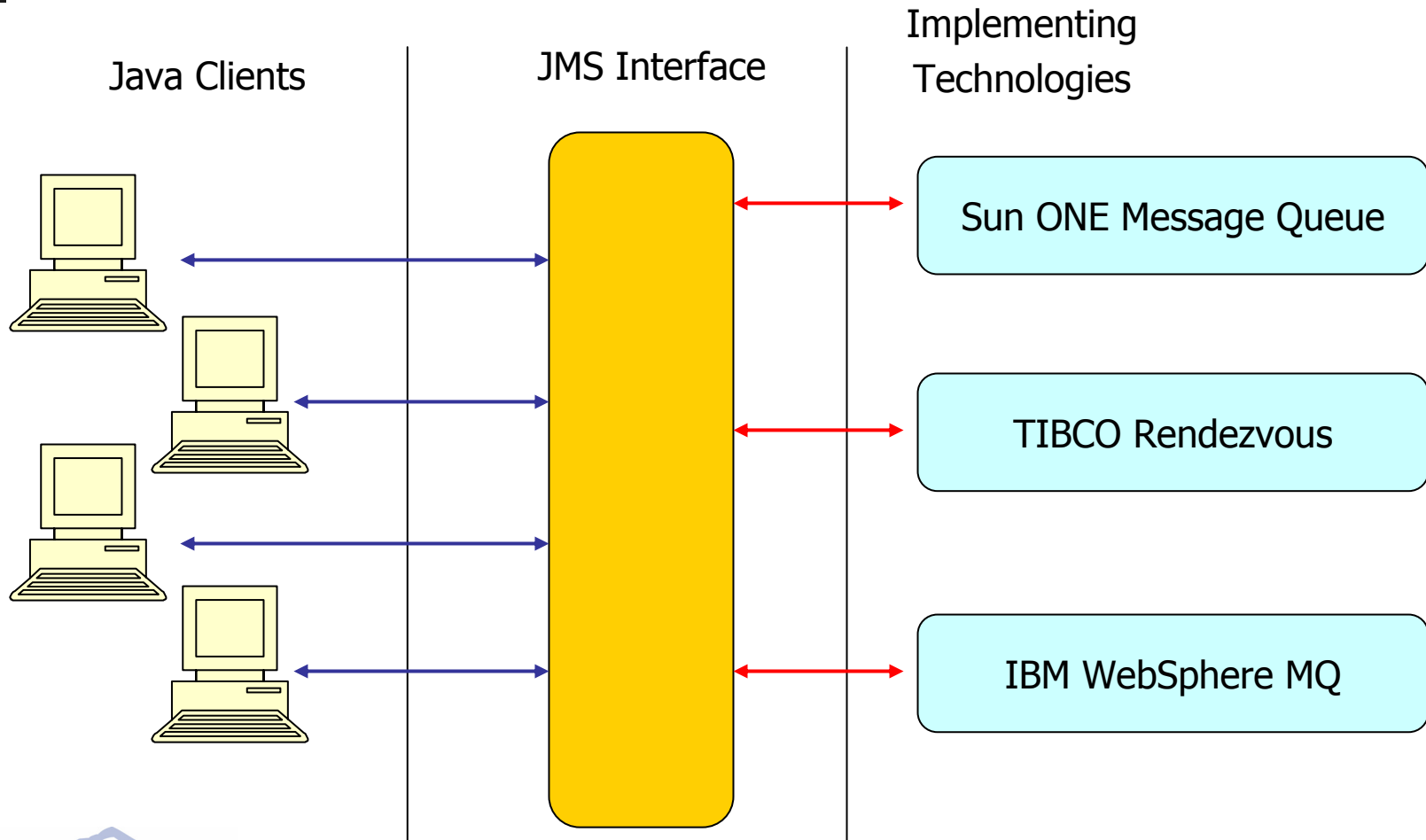


# Asynchronous Communication

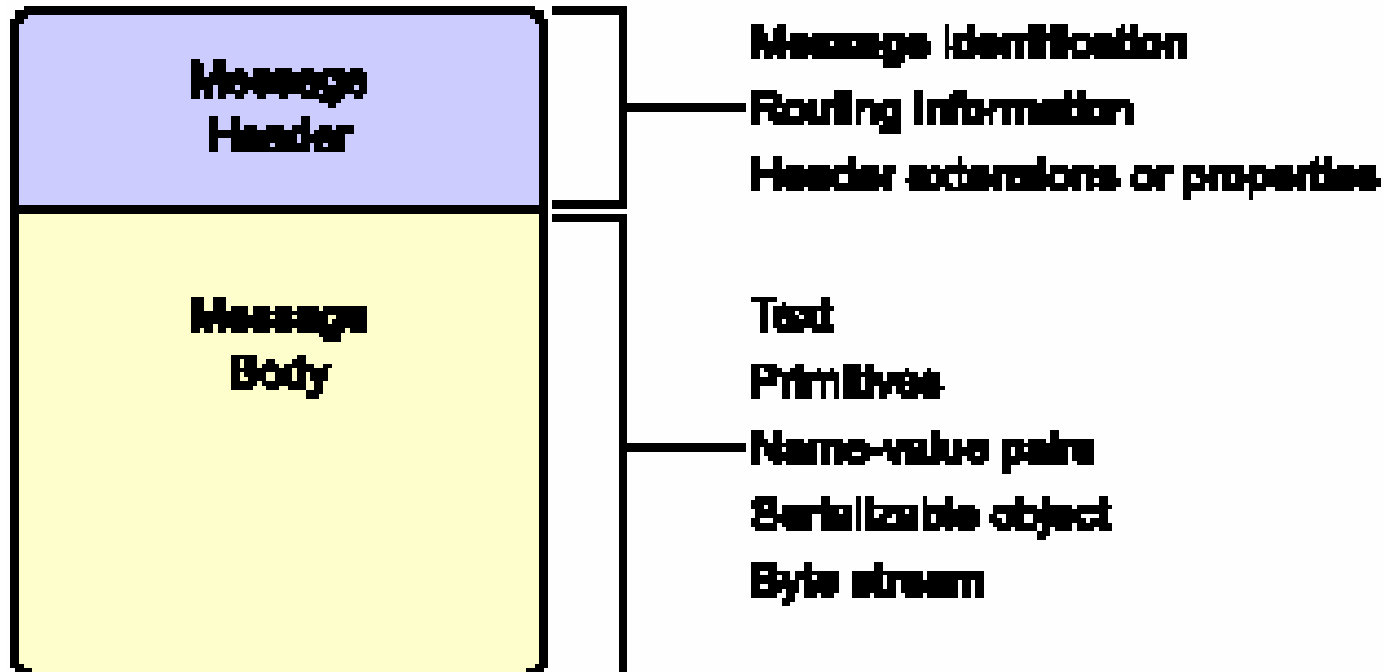
---

- Entity beans and Session beans are ill-suited to be message recipients
  - There is no good place in their lifecycle to 'block' while waiting for a message
  - There is no good place in the lifecycle to poll for a message
- MDBs fill this hole in the EJB world

# JMS Abstracts the Underlying Implementation

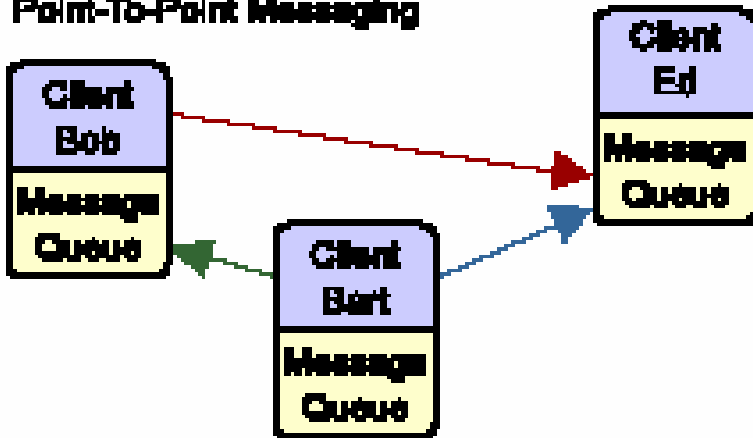


# Message Structure

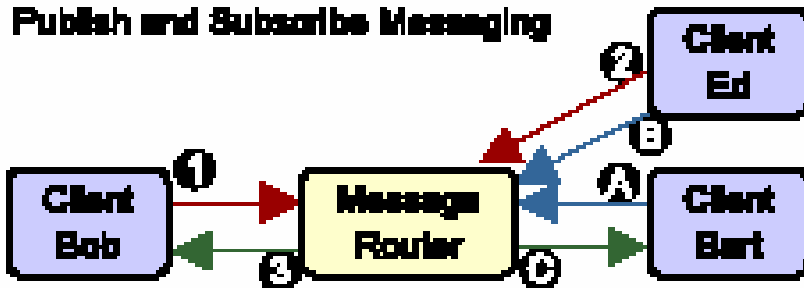


# Two Messaging Models

## Point-To-Point Messaging

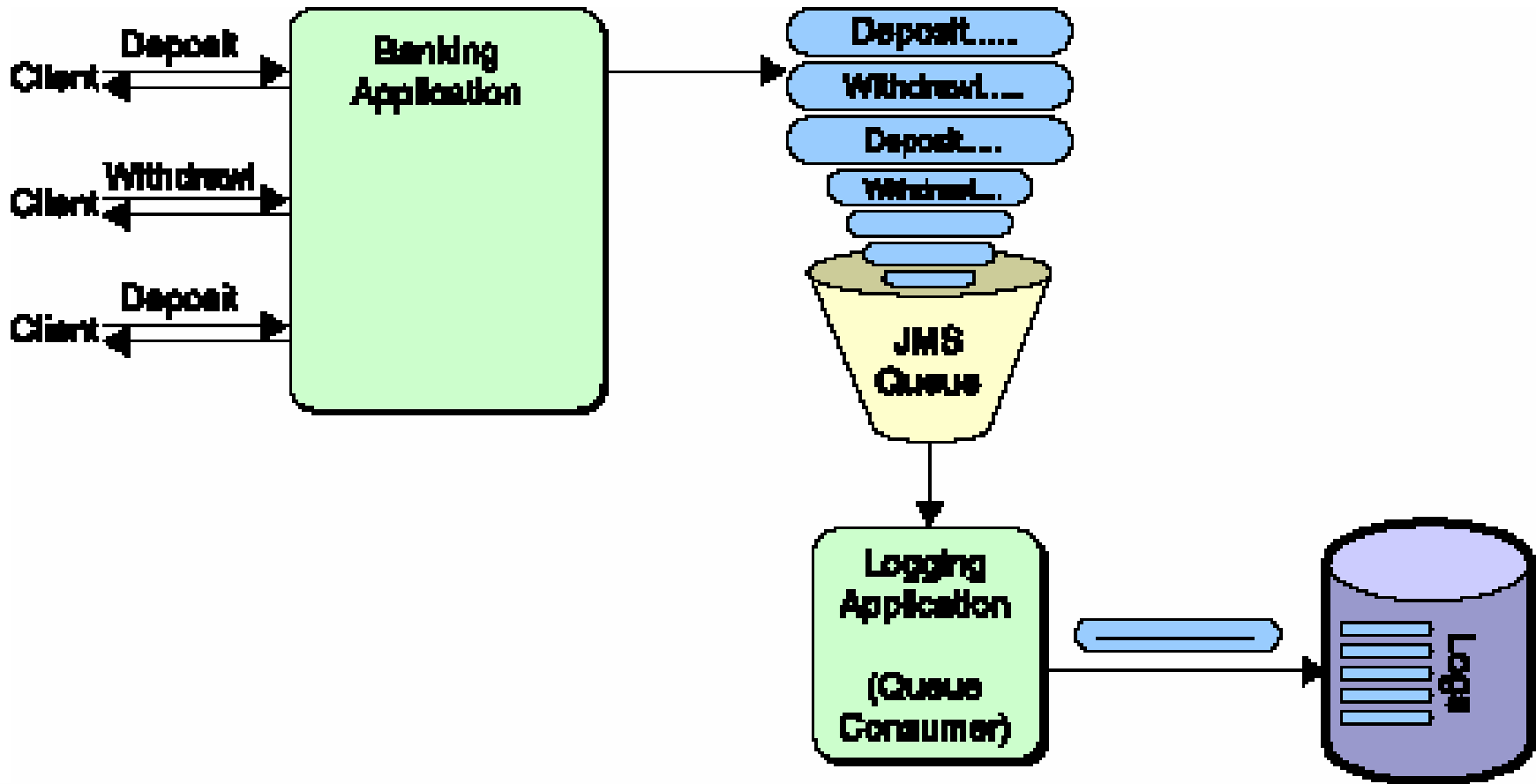


## Publish and Subscribe Messaging



- Point-to-point
  - One recipient gets the message
  - Similar to email model
- Publish and subscribe
  - Anyone interested subscribes to a topic, and is notified when a message of interest is available
  - Similar to mailing-list model

# Messaging Example



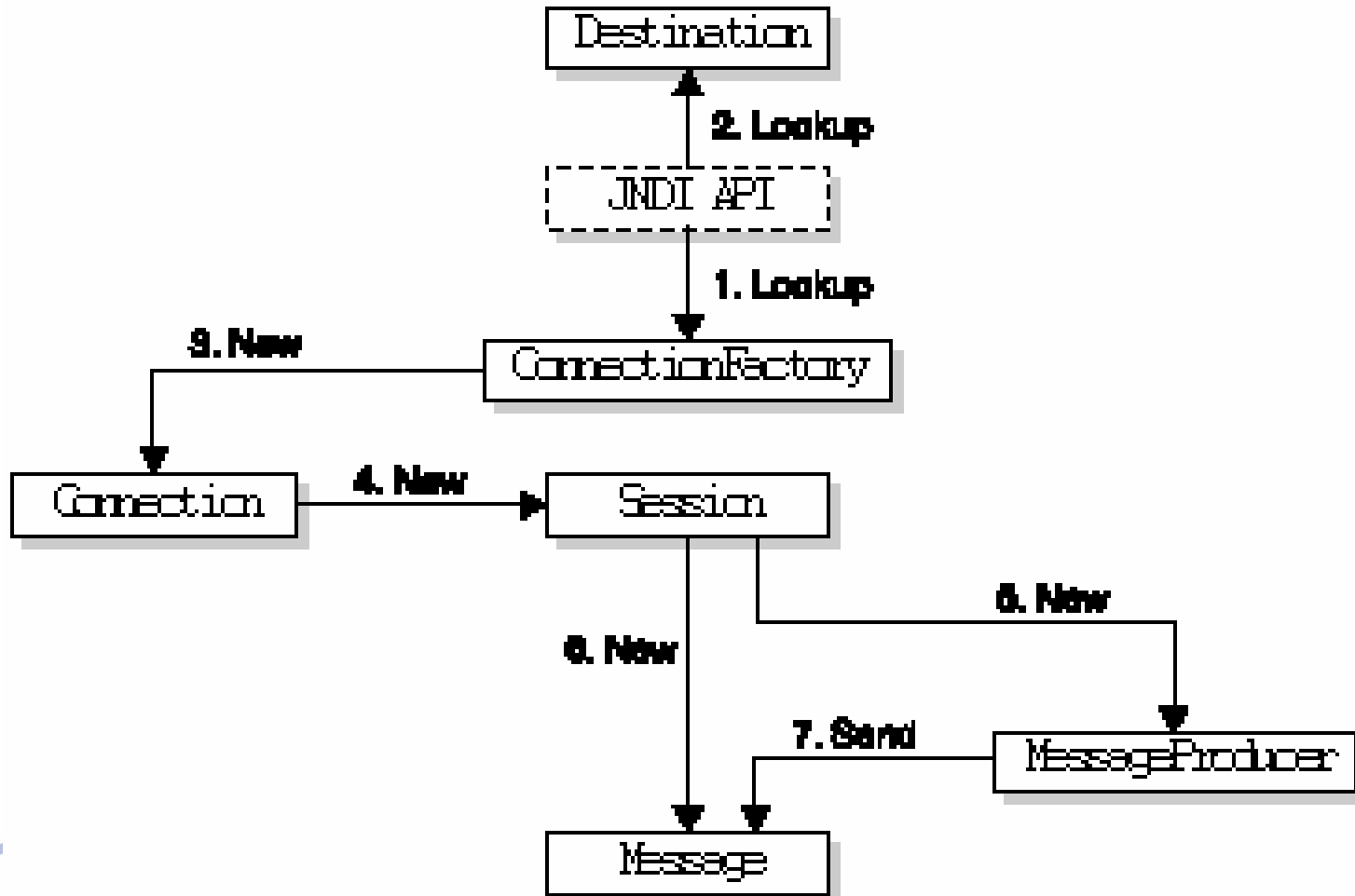


# Message-Driven Beans

---

- No client interfaces (component or home)
- Implement
  - MessageDrivenBean
  - javax.jms.MessageListener
- Only one method to implement
  - onMessage(javax.jms.Message m)
- Associated with JMS destination during deployment

# Process of Sending a Message to a Queue





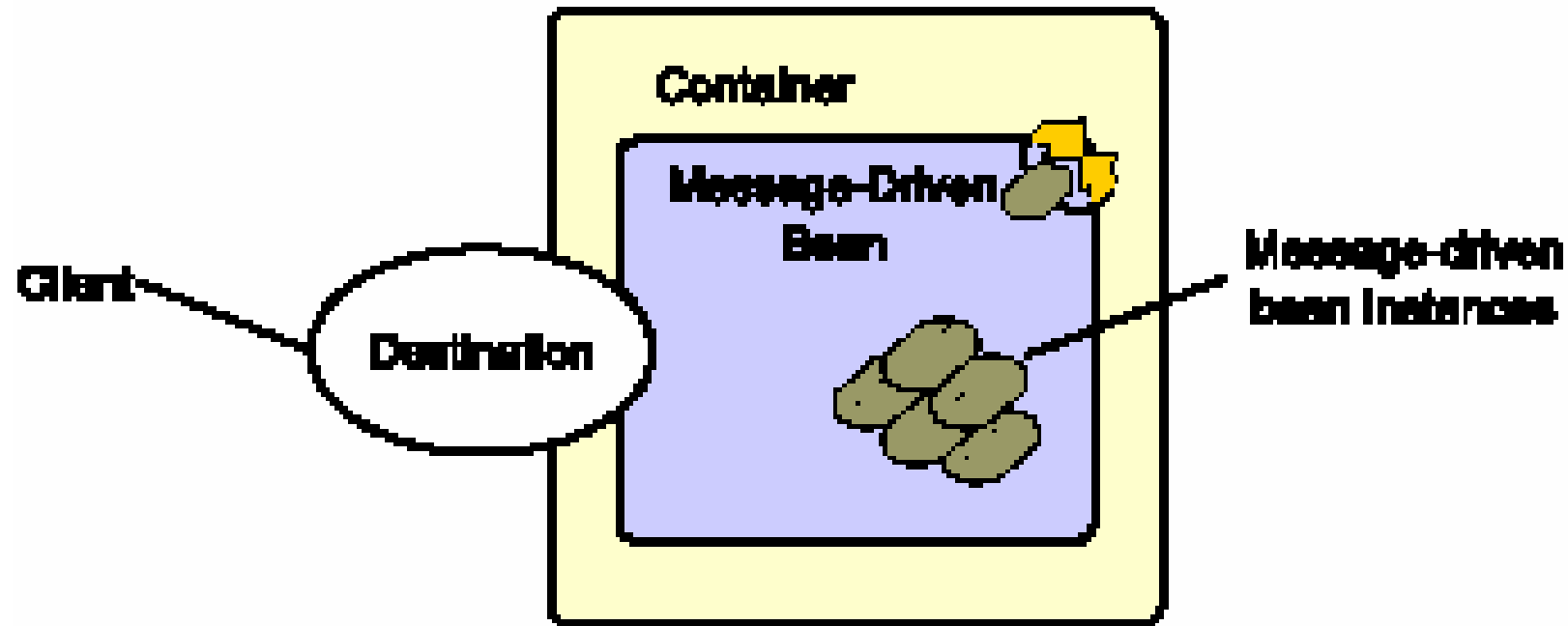


# Sending a Message to a Queue

---

- 1) ...
- 2) `Context ctx = new InitialContext();`
- 3) `QueueConnectionFactory factory =  
(QueueConnectionFactory) ctx.lookup  
("jms/QueueConnectionFactory");`
- 4) `Queue q = (Queue) ctx.lookup("jms/Queue");`
- 5) `QueueConnection qc = factory.createQueueConnection();`
- 6) `QueueSession qSess = qc.createQueueSession(false,  
Session.AUTO_ACKNOWLEDGE);`
- 7) `QueueSender qSend = qSess.createSender(q);`
- 8) `TestMessage tm = qSess.createTextMessage("Hello  
World");`
- 9) `qSend.send(tm);`
- 10) ...

# Client Relationship with a MDB



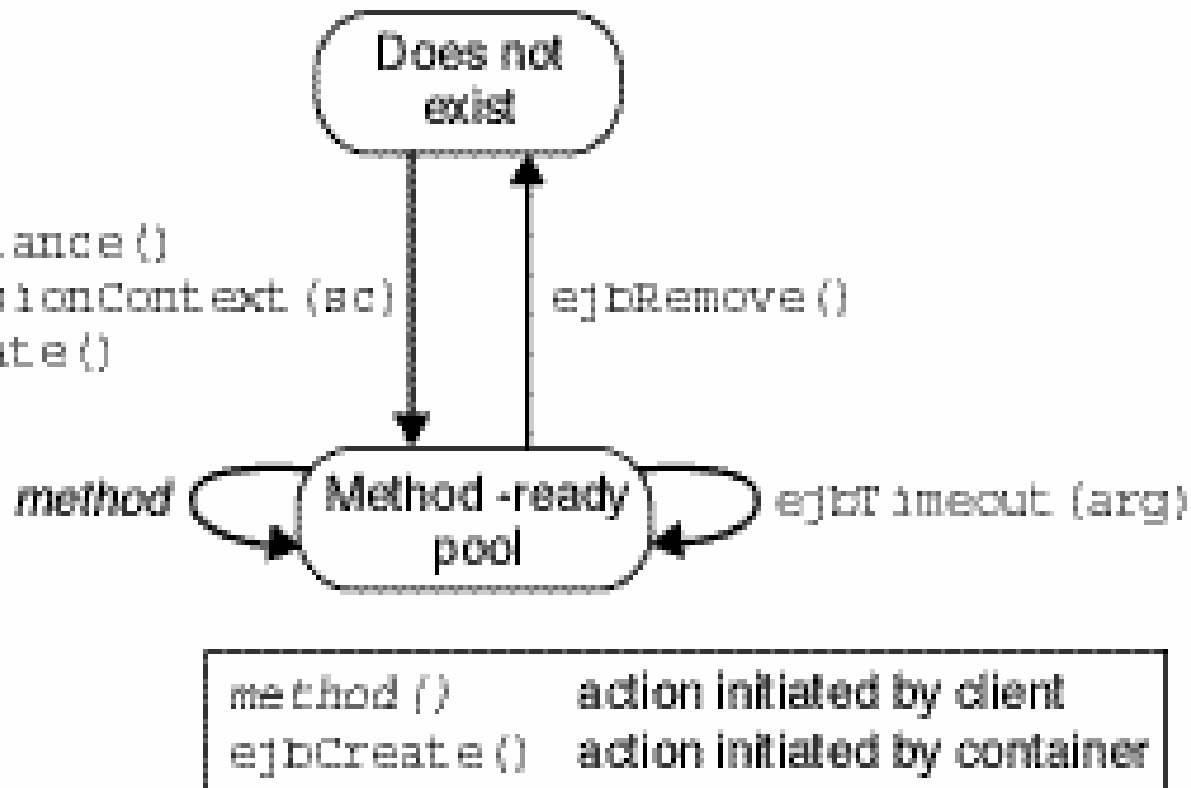


# EJB Lifecycle Issues

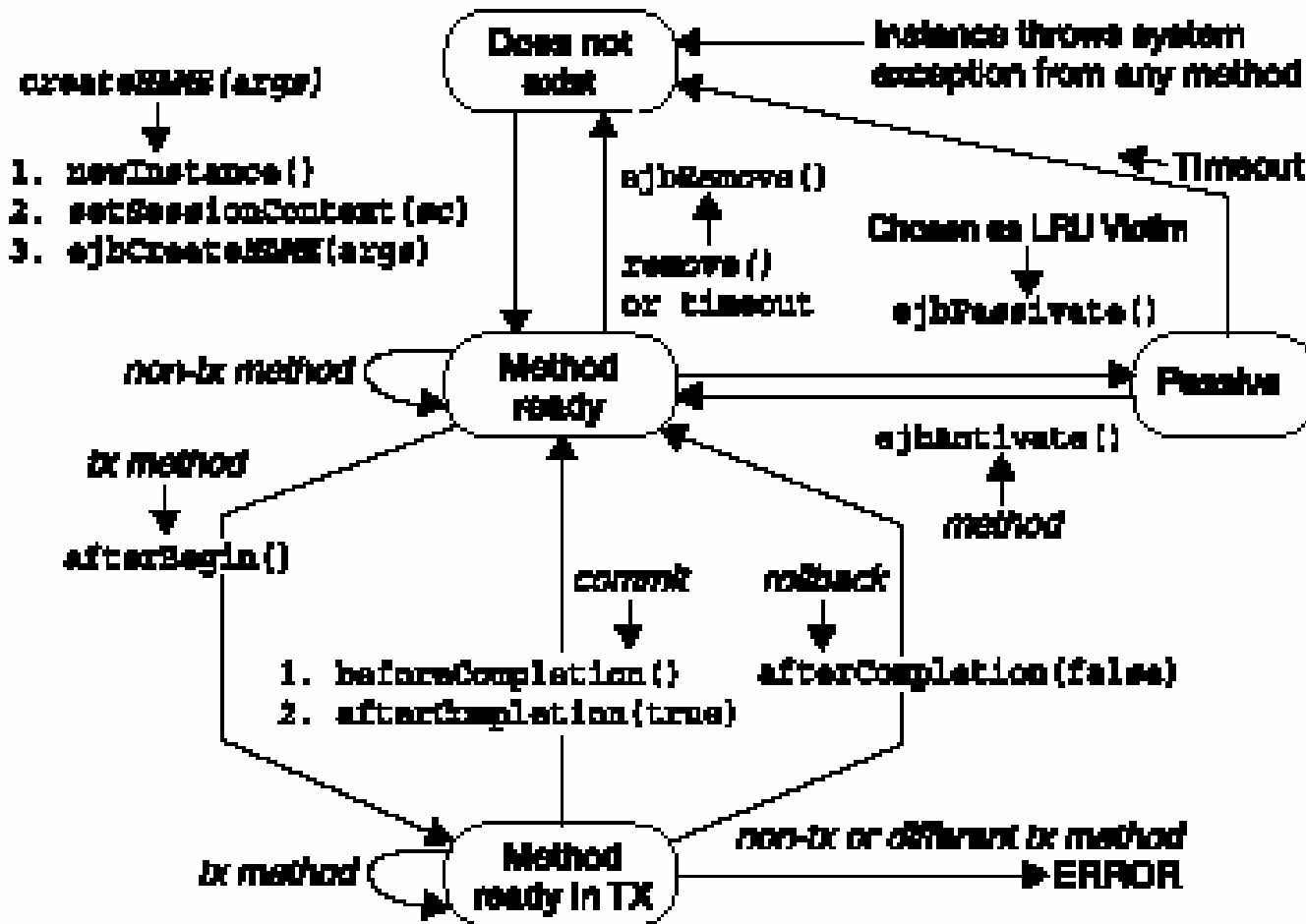
---

- The lifecycle of enterprise bean instances is managed by the container
- EJB components get callbacks at appropriate times
- For beans with more state, the lifecycle is more complicated

# Stateless Session Bean Lifecycle

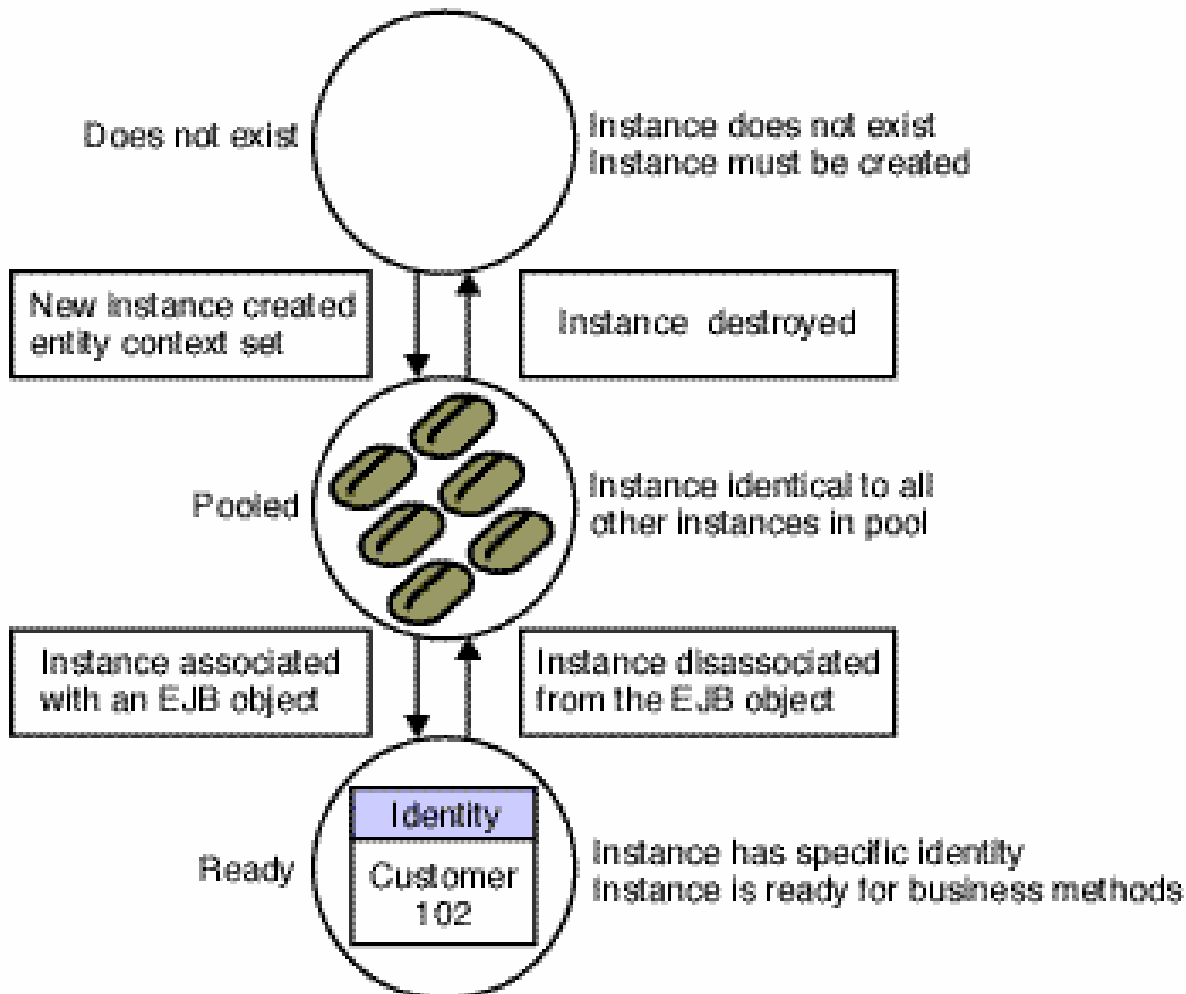


# Stateful Session Bean Lifecycle

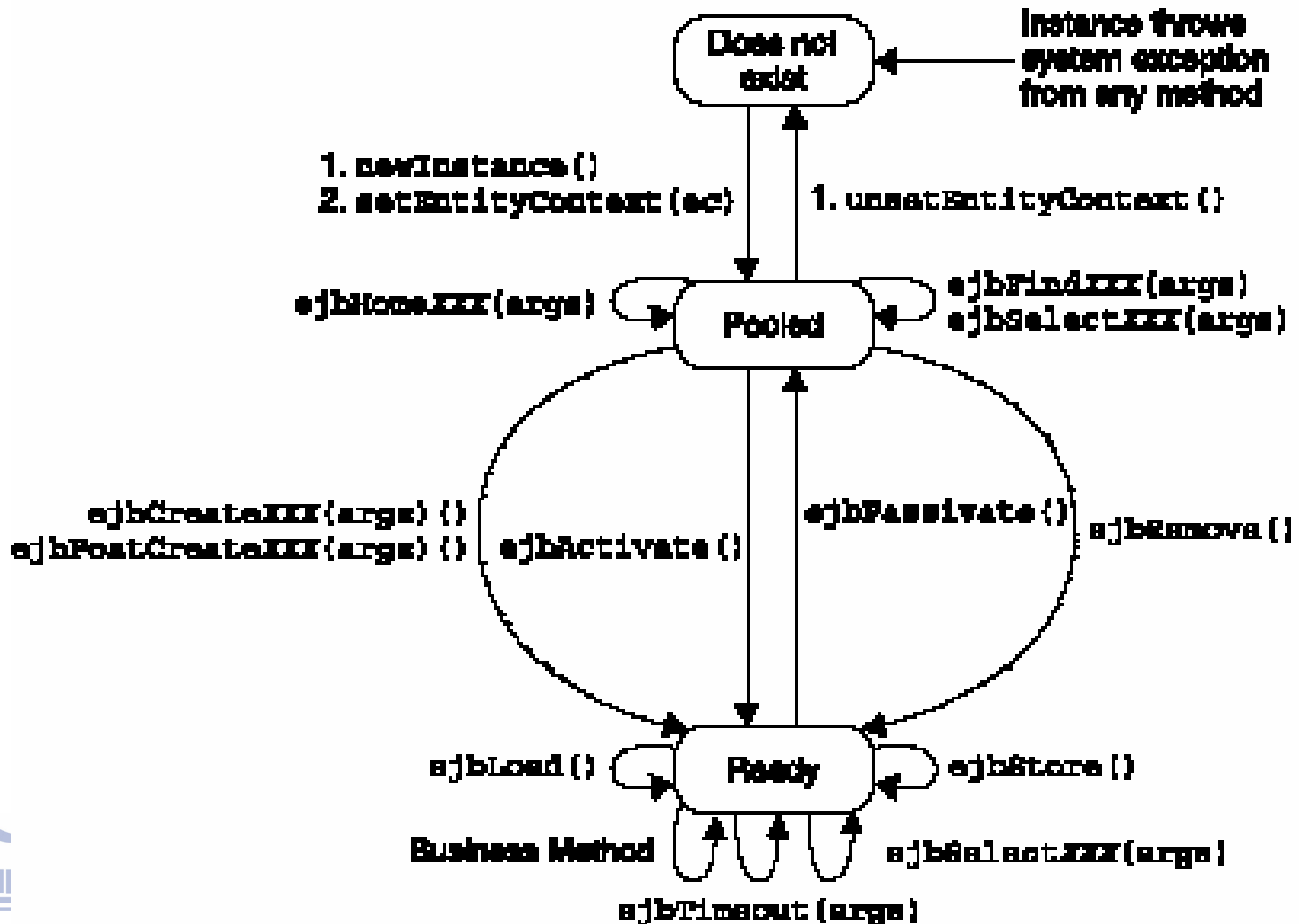


<code>create()</code>	action initiated by client
<code>newInstance</code>	action initiated by container

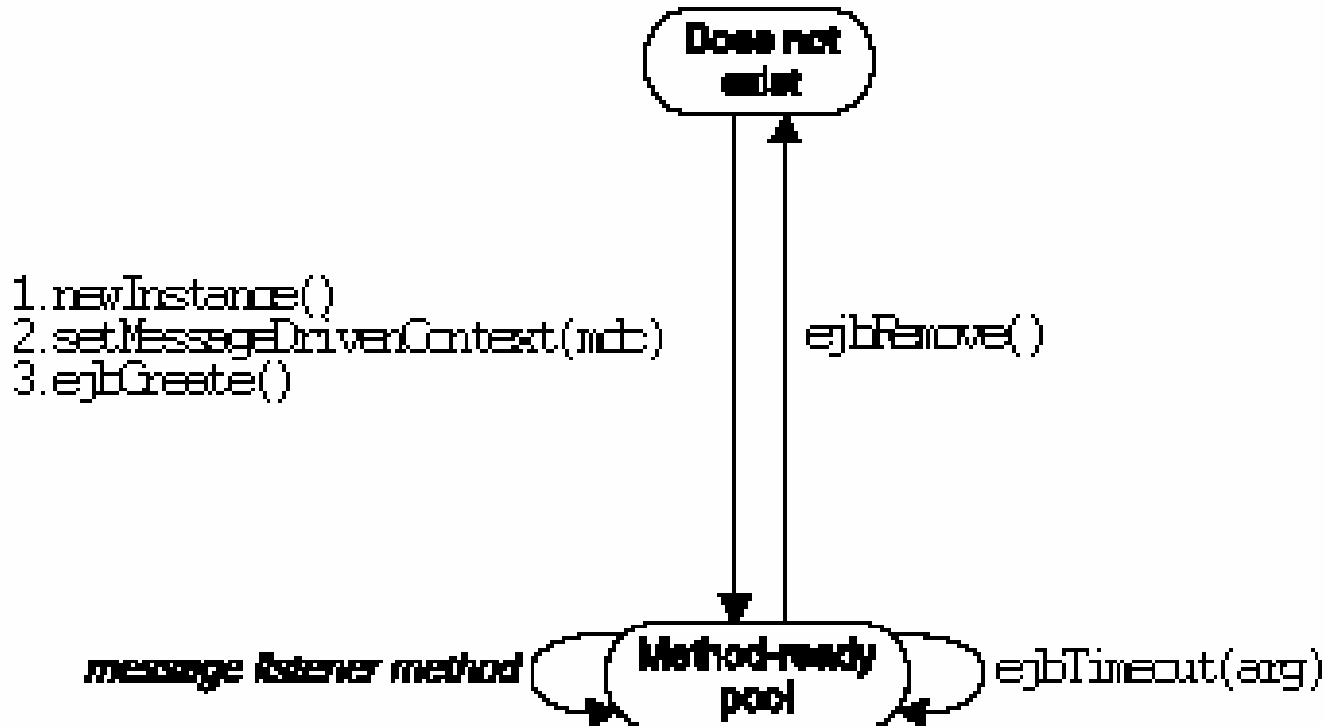
# Entity Bean Lifecycle Overview



# Entity Bean Life Cycle with Method Calls



# MDB Lifecycle



<i>message listener method</i> <code>ejbCreate()</code>	Action resulting from client message arrival
	Action initiated by container





# Client View of EJB

---

- Three steps to using an EJB:
  1. Lookup the home object with JNDI
    - Get InitialContext
    - Perform lookup
    - Narrow (remote objects only)
    - Cast
  2. Use the home object to create or find the bean
  3. Make your business method calls



# Client EJB calls

---

```
Context ic = new InitialContext();
```

- May need to set JNDI parameters in standalone applications

```
Object o = ic.lookup("fortuneJndiName");
```

```
FortuneHome fHome = (FortuneHome)
```

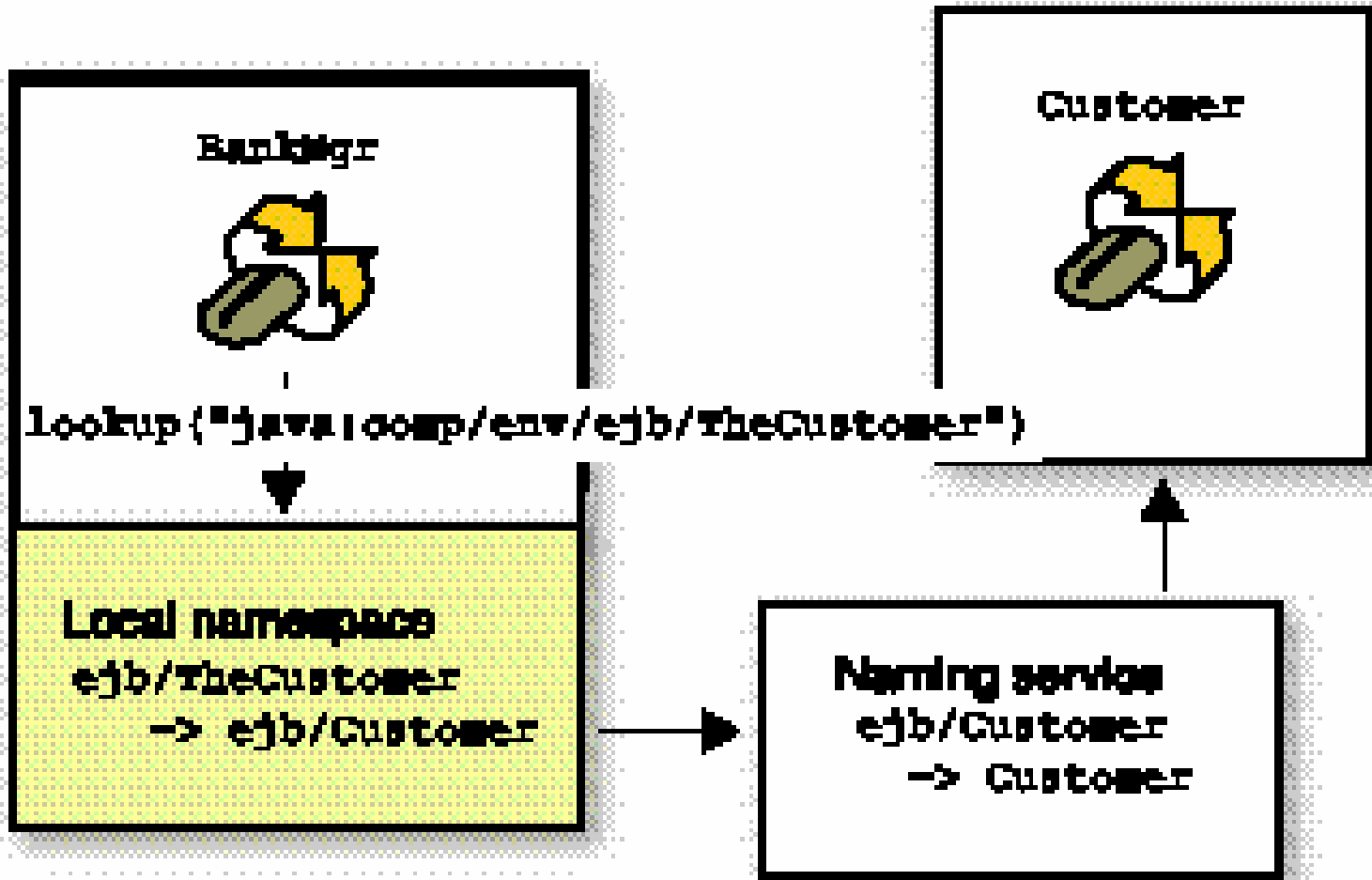
```
    PortableRemoteObject.narrow(o,  
    FortuneHome.class); // remote
```

- Or

```
FortuneLocalHome = (FortuneLocalHome)
```

```
    ic.lookup("localFortuneJndiName"); // local
```

# Resolving References at Deployment Time



# Using the Home Object to Get the EJBObject (Component Interface)

---

- `MyEJB.ejbObj = home.create();`
- `MyEJB.ejbObj = home.create(args);`
- `MyEntity.ejbObj = home.findByPrimaryKey(pk);`
- `MyEntity.ejbObj = home.findTheOne(...);`
- `Collection c = home.findAllThatMatch(...);`



# More Client View Stuff

---

- `ejbObject.remove()`
  - When done with stateful session “conversation”
  - Not necessary for stateless session bean
  - For entities, removes entity from the database (DELETE)
- Don't use `equals()`
  - `ejbObject.isIdentical(EJBObject other)` will test if two objects are referring to the same bean
  - Entity beans or stateful session beans
- `ejbObject.getPrimaryKey()`
  - Returns entity's primary key



# EJB Limitations

---

- The spec is the ultimate authority
- No read/write static fields
  - Might not be accessible by all EJBs
  - Container may distribute instances across multiple JVMs
- No thread synchronization or thread management
  - Might not work as expected
    - Distributed EJBs across multiple JVMs could be a problem
  - Could interfere with the container's pooling, load balancing, *etc.*
- No file I/O
- No server sockets or multicast
- No ClassLoader games or Native Library loading



# EJB Limitations

---

- Since you won't always get warnings or errors when you violate the limitations, some people choose to violate them.
- But:
  - Read the reasons for the restrictions in the spec
    - Chapter 24.1.2 for EJB 2.0
    - Chapter 25.1.1 for EJB 2.1
  - Make decisions based on the rationale found in the spec – violating the spirit of the restriction is asking for trouble



# EJB Limitations

---

- Never pass, return or otherwise give away “this”
  - Only the container can *ever* talk directly to an EJB instance
  - Get a usable reference to “this” from SessionContext or EntityContext:
    - `context.getEJBObject()`
    - `context.getEJBLocalObject()`





# Summary

---

- 3 Class files
  - Implementation Class
    - Container required stuff
    - Your code for component and home implementations
  - Component Interface
    - Contract for the component
    - Local and/or remote
  - Home Interface
    - Factory methods
    - Create, find, *etc.*



## Deployment Descriptor



# References

---

*Head First EJB*, Kathy Sierra & Bert Bates, O'Reilly, ISBN  
0596005717

*Mastering Enterprise Java Beans*, Ed Roman, Wiley, ISBN  
0471417114

*Enterprise Java Beans Specification (2.0 and 2.1)*

*Java 2 Platform Enterprise Specification (1.3 and 1.4)*

# Demos, Discussion, Questions?

---





# With Special Thanks...

---

- Dave Landers



# Contact Info

---

Kimberly Bobrow Jennery

[kimberly.bobrow@sun.com](mailto:kimberly.bobrow@sun.com)

[kimberly@jennery.com](mailto:kimberly@jennery.com)

[kimberly@bobrow.net](mailto:kimberly@bobrow.net)

[geekstress@javasluts.com](mailto:geekstress@javasluts.com) (don't ask)