Antoni Batchelli
Staff Software Engineer
Citrix Online
Santa Barbara, CA

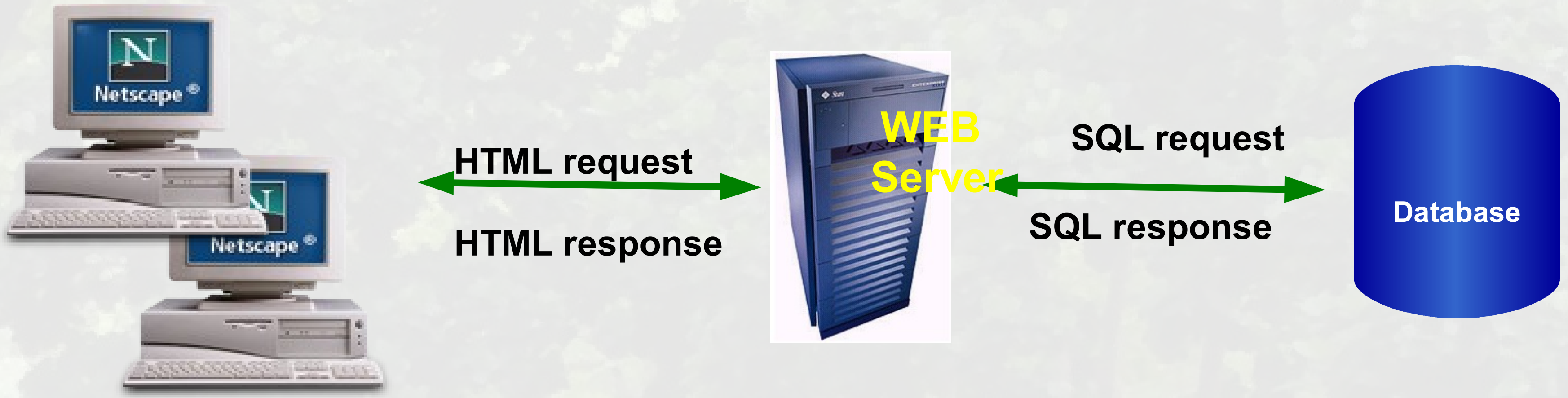
A Brief Introduction to J2EE

11/27/2006

Outline

- **N-Tier Model and Containers**
- What is J2EE?
- What Makes Up J2EE?
- Architecture
- Development and Deployment of Applications
- Business Tier: EJBs
- Enterprise Integration: Distributed Messaging, JMS and MDB
- Presentation Tier: Servlets and JSP
- Data Tier: JDBC

The Three-Tier Model



- Browser handles presentation logic
- Browser talks Web server via HTTP protocol

- Business logic and data model are handled by dynamically by the middle tier

- Data is stored in Databases and other repositories

The Three Tier Model: Pros & Cons

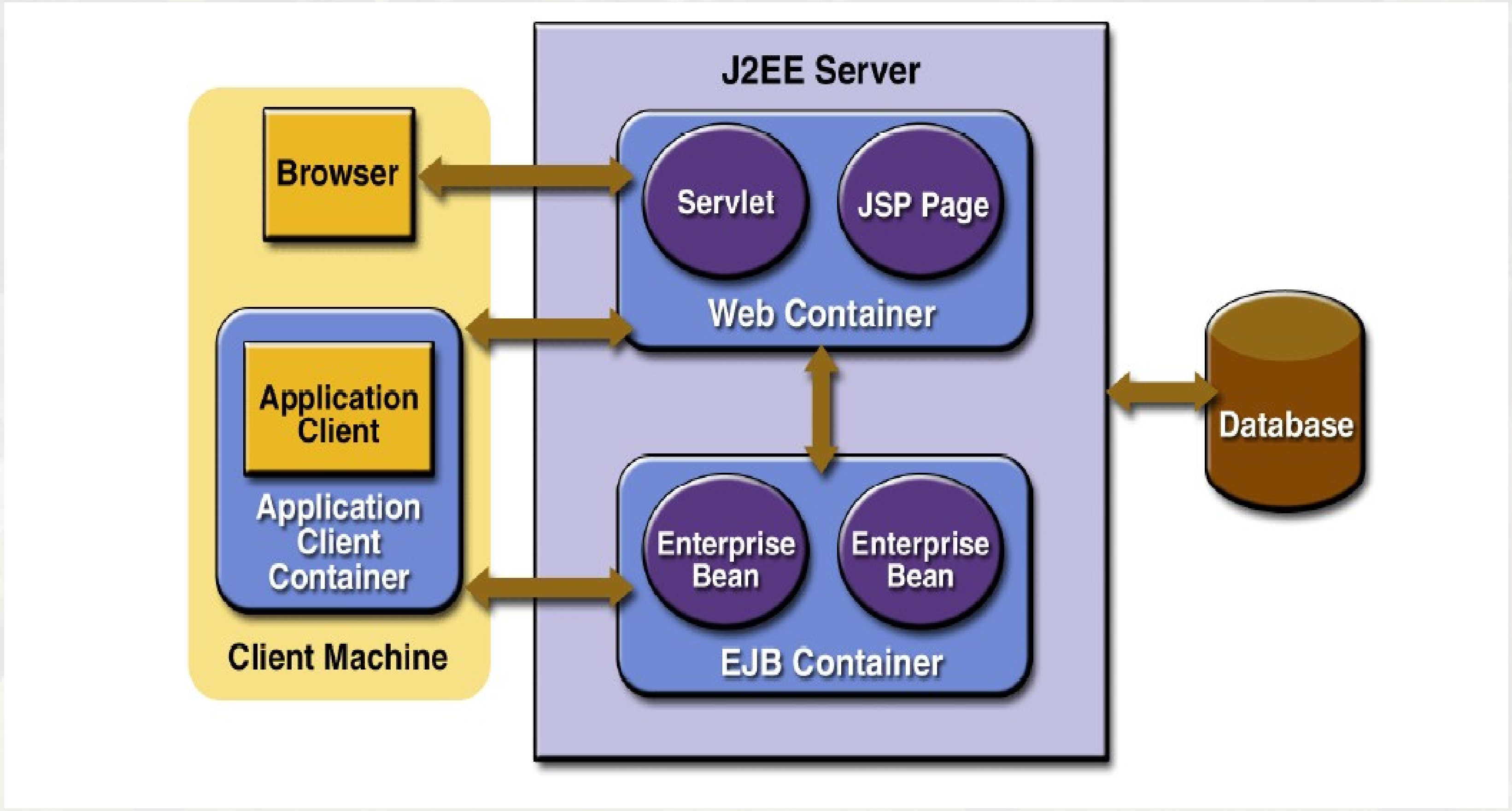
- Pros:
  - Thin clients deployed everywhere
  - Zero client management
  - Support various client devices
    - i.e. Web browsers, phones, handhelds, etc...
- Cons:
  - Complexity is moved from the client into the middle-tier still but it still **needs** to be addressed

Middle Tier Issues

- Complexity moved into the middle tier
- New problems arise (scalability, concurrency, availability)
- Enhanced system services need to be provided for the middle tier applications to be manageable
  - Concurrency control, Transactions
  - Load-balancing, Security
  - Resource management, Connection pooling

Component-Container Model

- Developers program components that “live” inside a container. The container provides services for scalability, load balancing, security, etc..

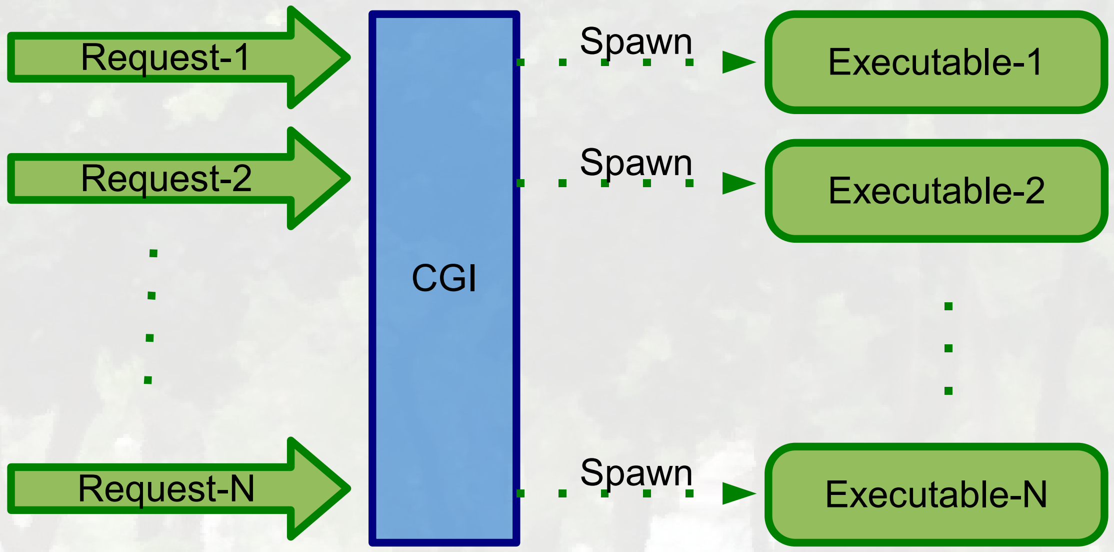


What does a container usually provide?

- Component Life Cycle
- Session Life Cycle
- Distribution Support
- Cluster Support
- Distributed Transaction Support
- Authentication and Authorization
- Management

Component Lifecycle

- Create or reuse? The CGI way

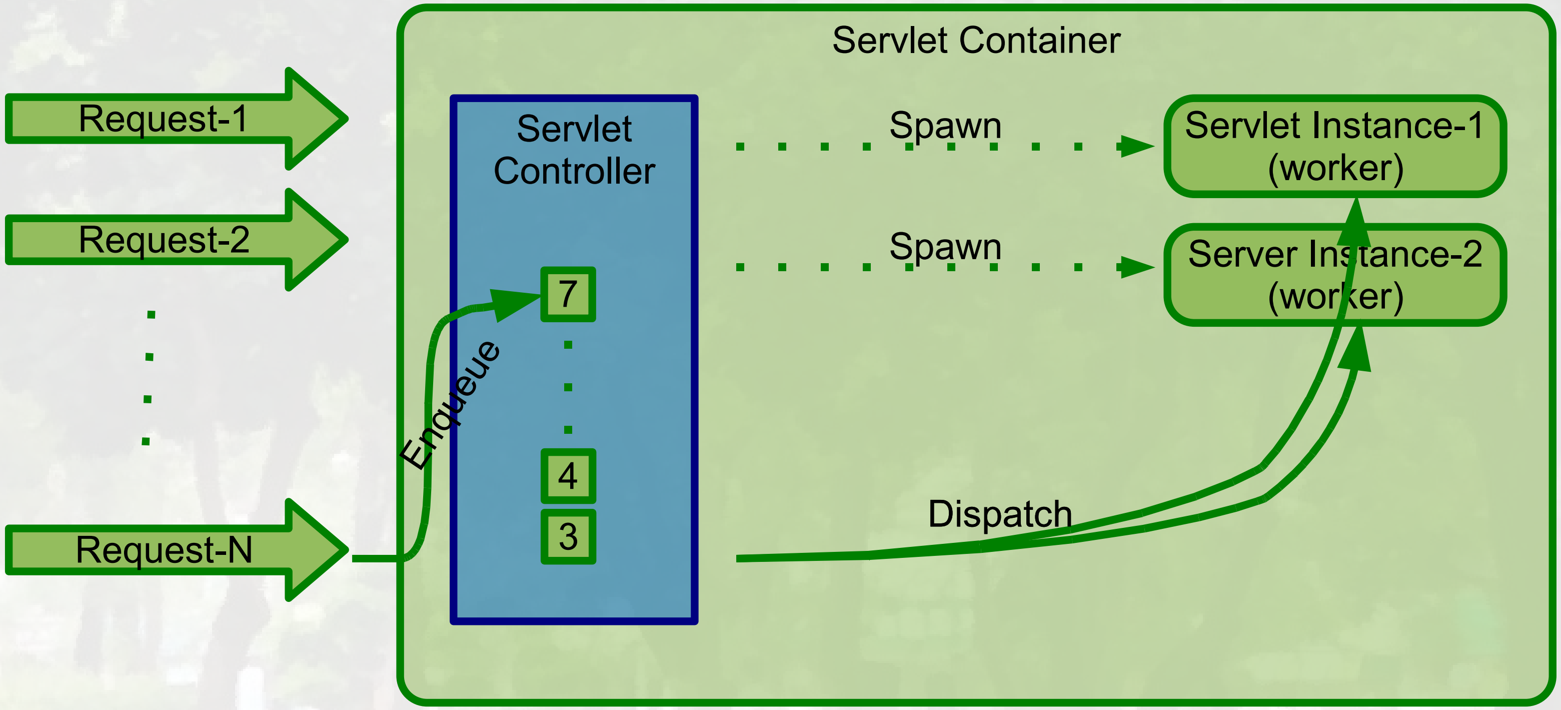


N Concurrent requests ==> N concurrent processes

What if N $\rightarrow \infty$?

Component Lifecycle (2)

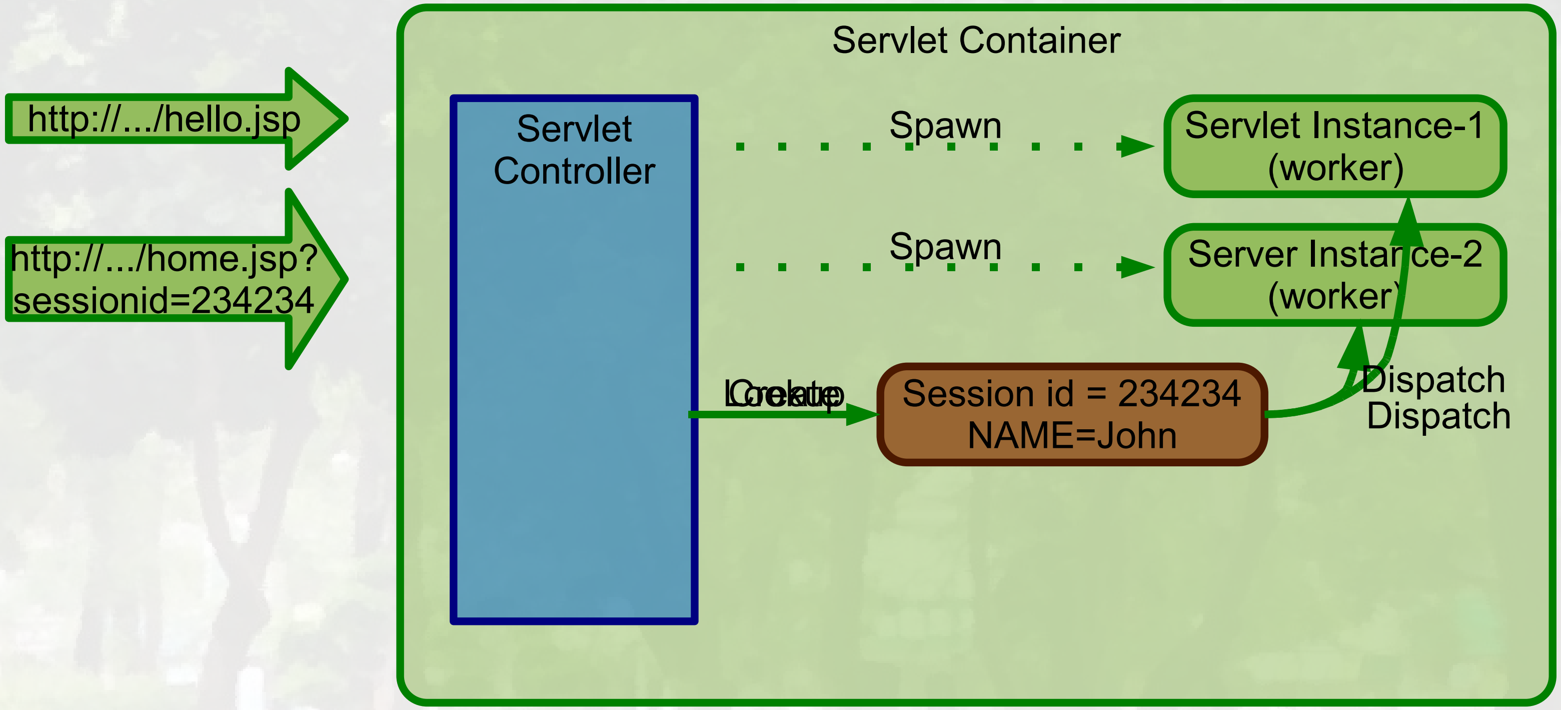
- Create or reuse?



# Concurrent Requests >> # Workers >> # Processes

Session Lifecycle

- What about stateful services?

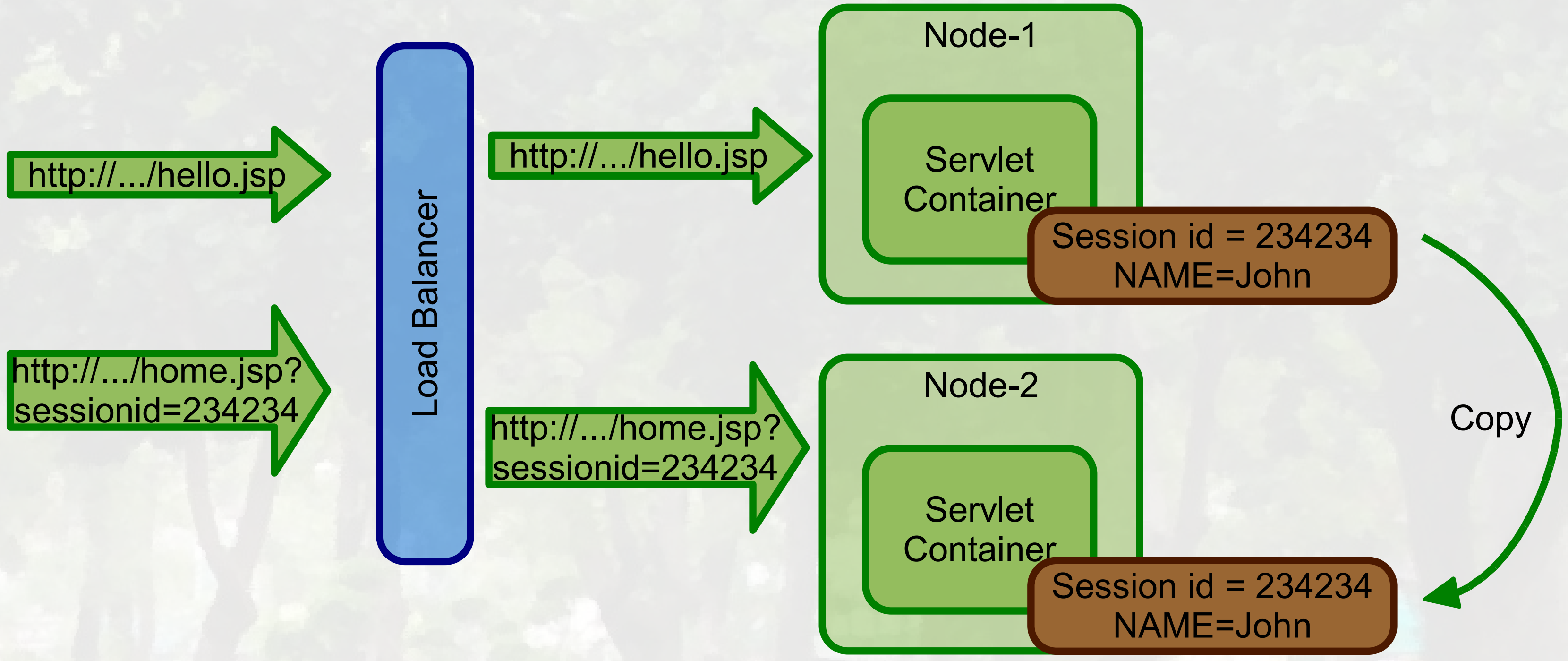


http://.../hello.jsp
http://.../home.jsp?sessionid=234234
Servlet Container
Servlet Controller
Spawn
Spawn
Servlet Instance-1 (worker)
Server Instance-2 (worker)
Create
Lookup
Session id = 234234 NAME=John
Dispatch
Dispatch

# Concurrent Sessions >> # Workers >> # Processes

Distribution/Cluster Support

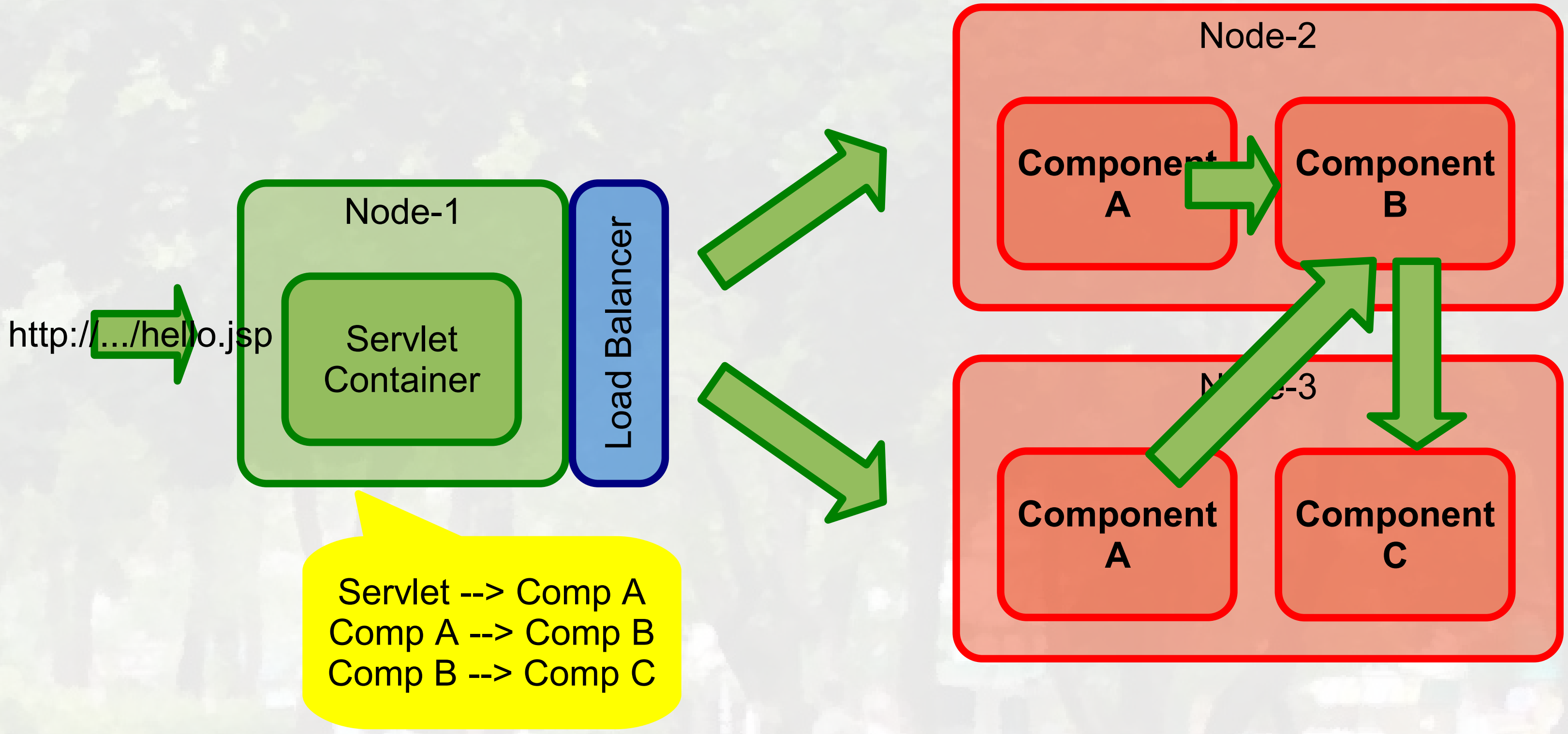
- What about stateful services?



State synchronization is taken care of by the container

Distribution/Cluster Support

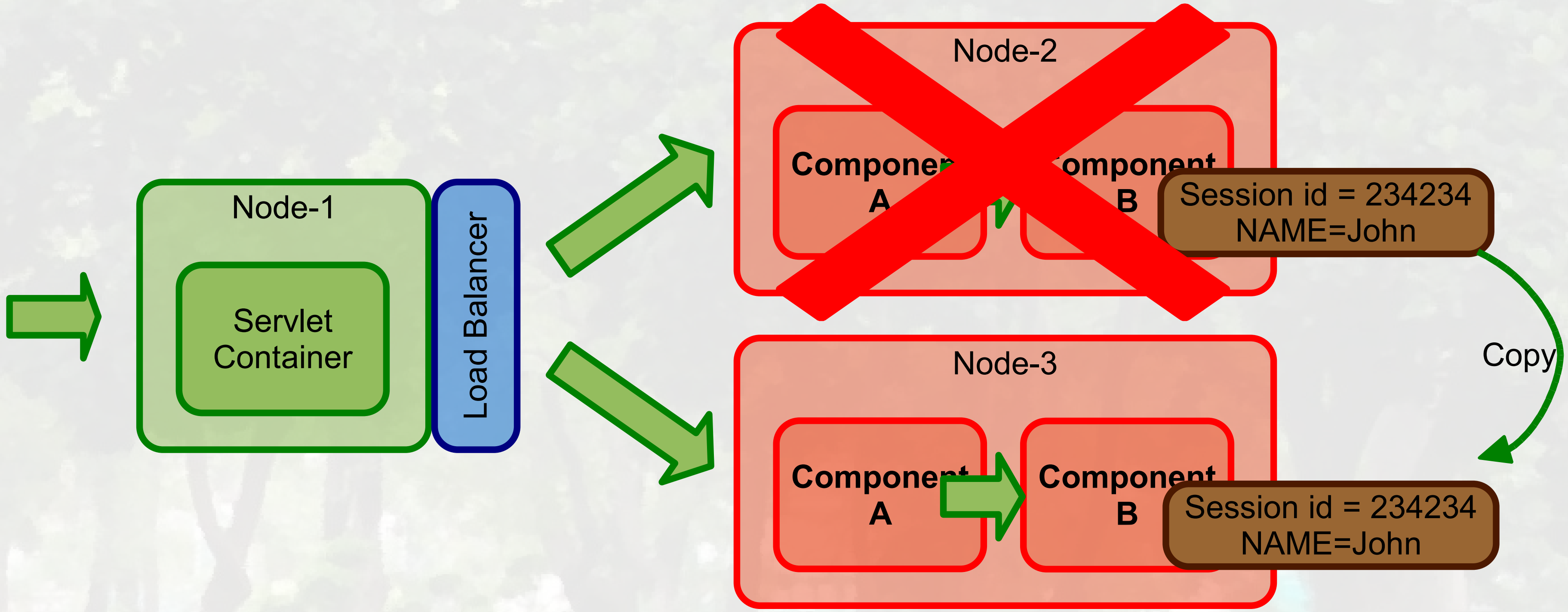
- Load Balancing and Distribution



Distribution and Load Balancing are taken care of by the Container

High Availability

- What about stateful services?



Re-routing and session replication is taken care of by the container

Outline

What Is the J2EE?

- **Open** and **standard** based platform for **developing**, **deploying** and **managing** n-tier, Web-enabled, server-centric, and **component**-based **enterprise** applications

J2EE: An Open and Standard Solution

- A component and container model in which container provides system services through a set of **well-defined and industry standard services**
- J2EE provides code portability; an application written for a particular brand of J2EE application server will work on any other implementation of a J2EE application server.

What's In It For Developers?

- Developers can use any J2EE implementation for development and deployment
  - Use a small scale J2EE server for development
  - Use high-end commercial J2EE server for scalability and fault-tolerance in production
- There is a vast amount of J2EE community resources
  - Books, articles, tutorials, source code, best practice guidelines, design patterns etc.
- **Can use off-the-shelf 3rd-party business components**

What's in it for Vendors?

- Vendors work together on **specifications** (JSR) and then compete in **implementations**:
  - More standards --> more users
  - More users --> more momentum
  - More momentum --> more sales
- Do not have create/maintain their own proprietary APIs

What's In It For Business Customers?

- **Application portability**
- Many implementation choices are possible based on various requirements
  - Price, scalability (single CPU to clustered model), reliability, performance, tools, and more
  - Best of breed of applications and platforms
- Large developer pool :)

Outline

- N-Tier Model and Containers
- What is J2EE?
- **What Makes Up J2EE?**
- Architecture
- Development and Deployment of Applications
- Business Tier: EJBs
- Enterprise Integration: Distributed Messaging, JMS and MDB
- Presentation Tier: Servlets and JSP
- Data Tier: JDBC

What Makes Up J2EE?

- A set of **APIs** and Technology specifications
- A **Development** and **Deployment** Platform
- A **Standard** and production-quality reference implementation
- A **Compatibility** Test Suite (CTS)
- Extensive documentation:
  - J2EE Blueprints
  - Sample source code

J2EE APIs and Technologies

- J2SE
- JAX-RPC
- Web Services for J2EE
- J2EE Management
- J2EE Deployment
- JMX 1.1
- JMS 1.1
- JTA 1.0
- Servlet 2.4
- JSP 2.0
- EJB 2.1
- JAXR
- Connector 1.5
- JACC
- JAXP 1.2
- JavaMail 1.3
- JAF 1.0

What are Servlets?

- Java™ objects which extend the functionality of a HTTP server
- **Dynamic content generation**
- Better alternative to CGI, NSAPI, ISAPI, etc.
  - Efficient
  - Platform and server independent
  - Session management
  - Java-based

What is JSP Technology?

- Enables **separation** of **business logic** from **presentation**
  - Presentation is in the form of HTML or XML/XSLT
  - Business logic is implemented as **Java Beans or custom tags**
  - Better maintainability, reusability
- Extensible via custom tags
- Builds on Servlet technology

What is EJB Technology?

- A **server-side component** technology
- Easy development and deployment of Java technology-based application that are:
  - Transactional, distributed, multi-tier, portable, scalable, secure, …

Java Message Service (JMS)

- Messaging systems provide
  - De-coupled communication
  - Asynchronous communication
  - Plays a role of centralized post office
- Benefits of Messaging systems
  - Flexible, Reliable, Scalable communication systems
- Point-to-Point, Publish and Subscribe
- JMS defines standard Java APIs to messaging systems

JNDI

- Java Naming and Directory Interface
- Utilized by J2EE applications to locate resources and objects in **portable** fashion
  - Applications use symbolic names to find object references to resources via JNDI
  - The symbolic names and object references have to be configured by system administrator when the application is deployed.

JDBC

- Provides standard Java programming API to relational database (via SQL)
- Vendors provide JDBC compliant driver which can be invoked via standard Java programming API
- A separate API provides pooling of JDBC connections

Standard Implementation

- Under J2EE 1.4 SDK, it is Sun Java Application Server Platform Edition 8
- Production-quality J2EE 1.4 compliant app server
- Free to develop and free to deploy
- Seamless upgrade path to Sun Java Application Server Enterprise Edition

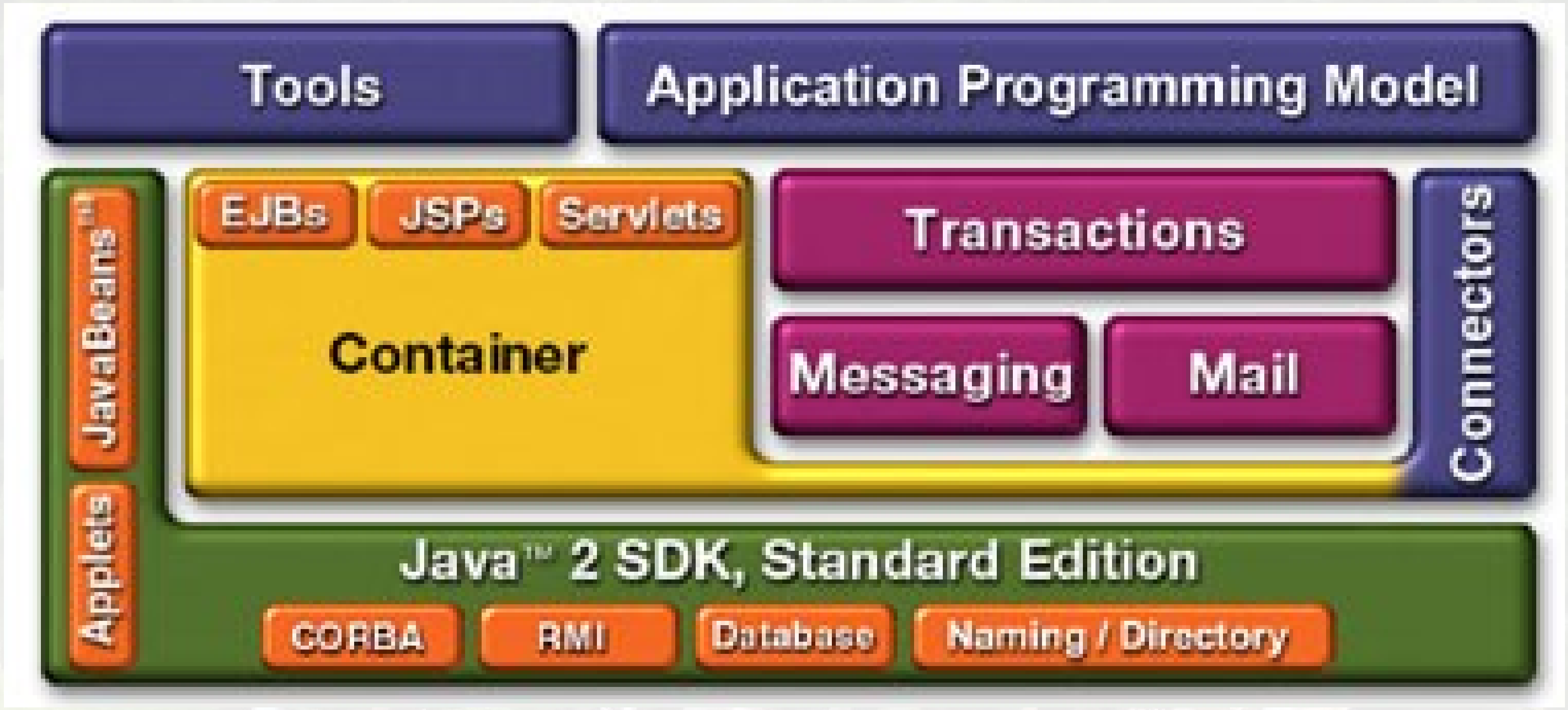
Compatibility Test Suite (CTS)

- Ultimate Java™ technology mission:
  - Write Once, Run Anywhere™
  - My Java-based application runs on any compatible Java virtual machines
  - My J2EE based technology-based application will run on any J2EE based Compatible platforms

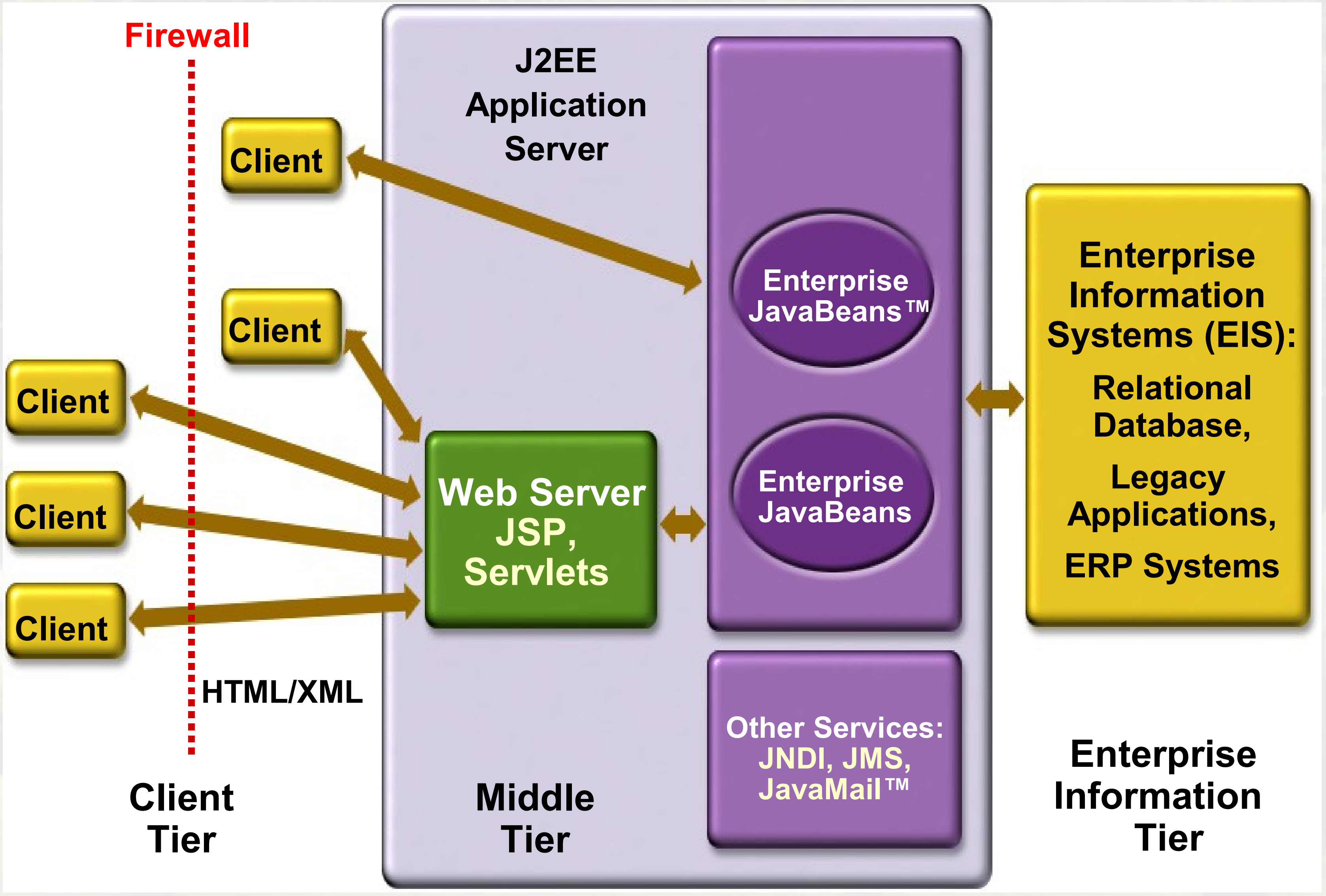
Outline

- N-Tier Model and Containers
- What is J2EE?
- What Makes Up J2EE?
- **Architecture**
- Development and Deployment of Applications
- Business Tier: EJBs
- Enterprise Integration: Distributed Messaging, JMS and MDB
- Presentation Tier: Servlets and JSP
- Data Tier: JDBC

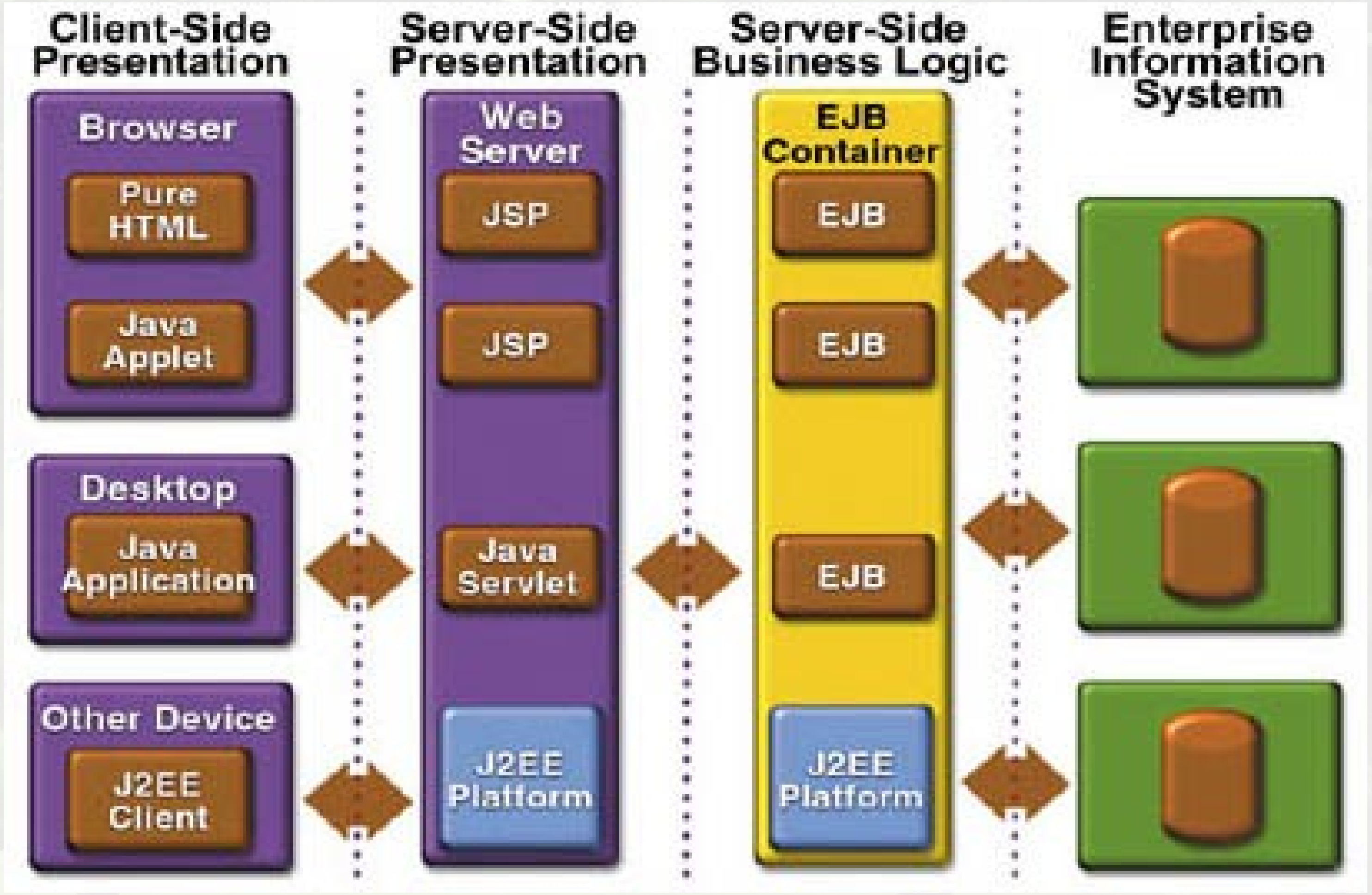
The J2EE Platform Architecture



J2EE is End-to-End Solution



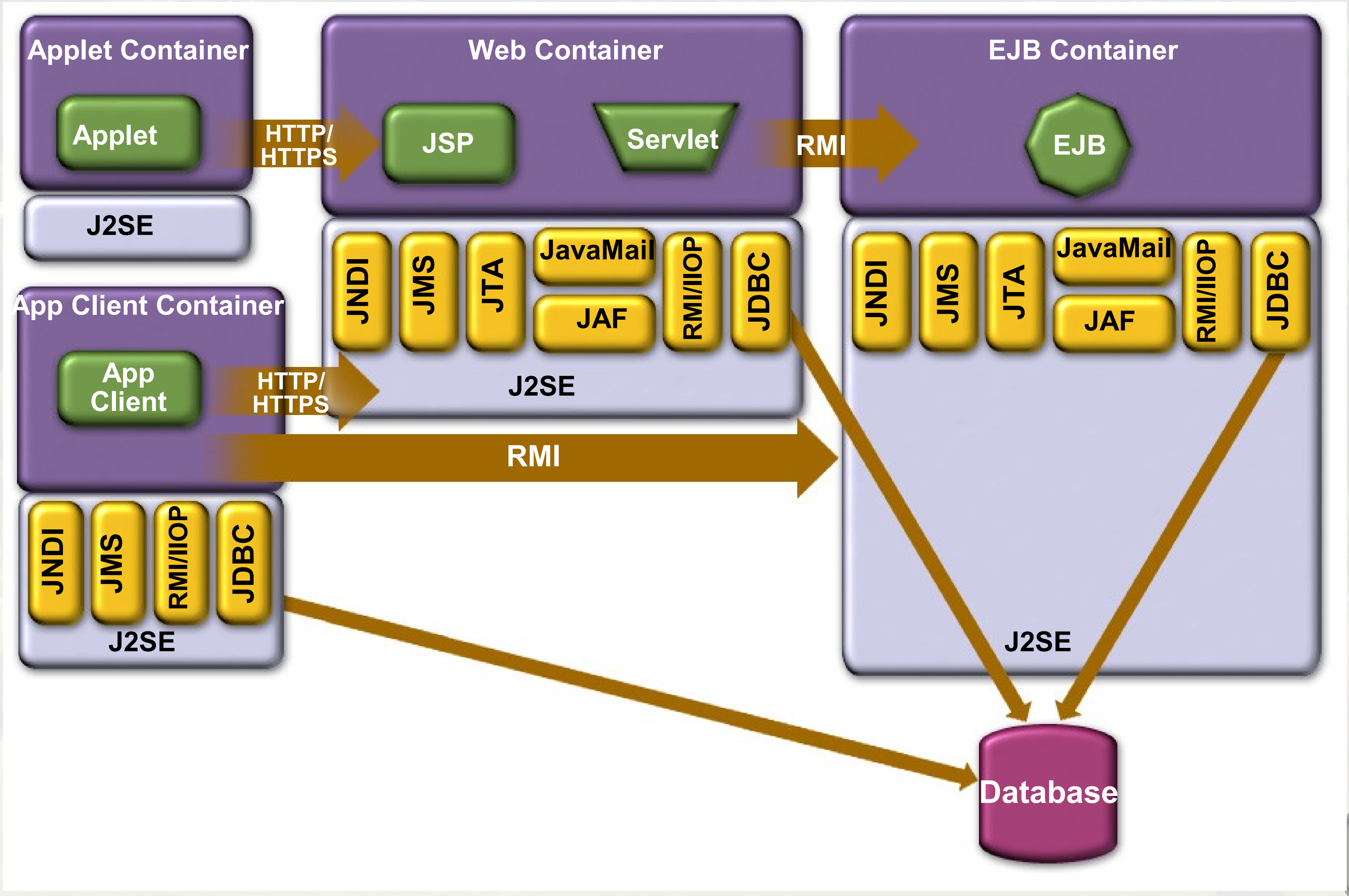
N-tier J2EE Architecture



Web Tier

EJB Tier

J2EE Containers & Components



Containers and Components

Containers Handle

- Concurrency
- Security
- Availability
- Scalability
- Persistence
- Transaction
- Life-cycle management
- Management

Components Handle

- Presentation
- Business Logic

Containers & Components

- Containers do their work invisibly
  - No complicated APIs
  - They control by interposition
- Containers implement J2EE
  - Look the same to components
  - Vendors making the containers have great freedom to innovate

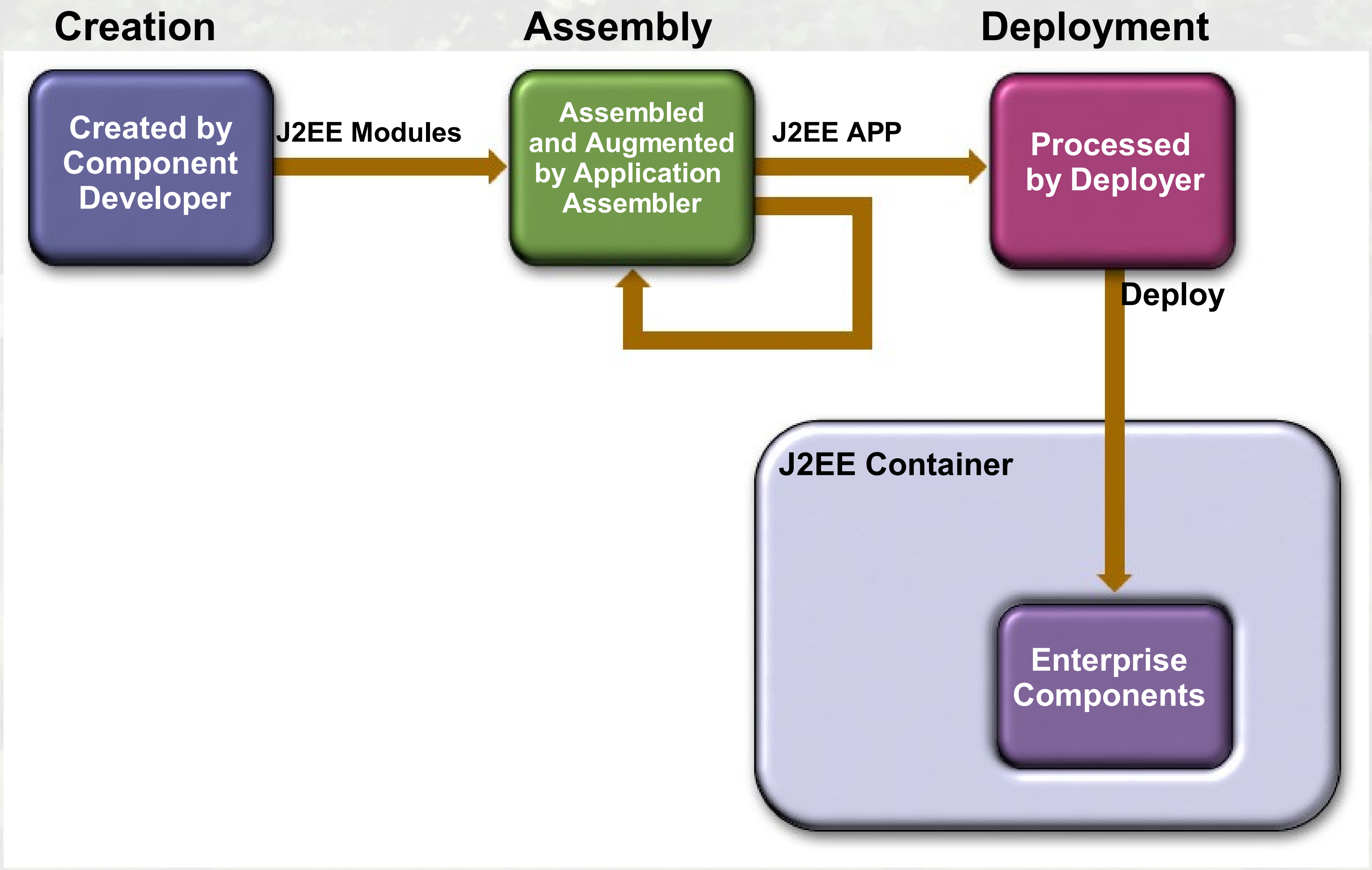
Outline

- N-Tier Model and Containers
- What is J2EE?
- What Makes Up J2EE?
- Architecture
- **Development and Deployment of Applications**
- Business Tier: EJBs
- Enterprise Integration: Distributed Messaging, JMS and MDB
- Presentation Tier: Servlets and JSP
- Data Tier: JDBC

J2EE Application Development Lifecycle

- Write and compile component code
  - Servlet, JSP, EJB
- Write deployment descriptors for components
- Assemble components into ready-to-deployable package
- Deploy the package on a server

Life-cycle Illustration



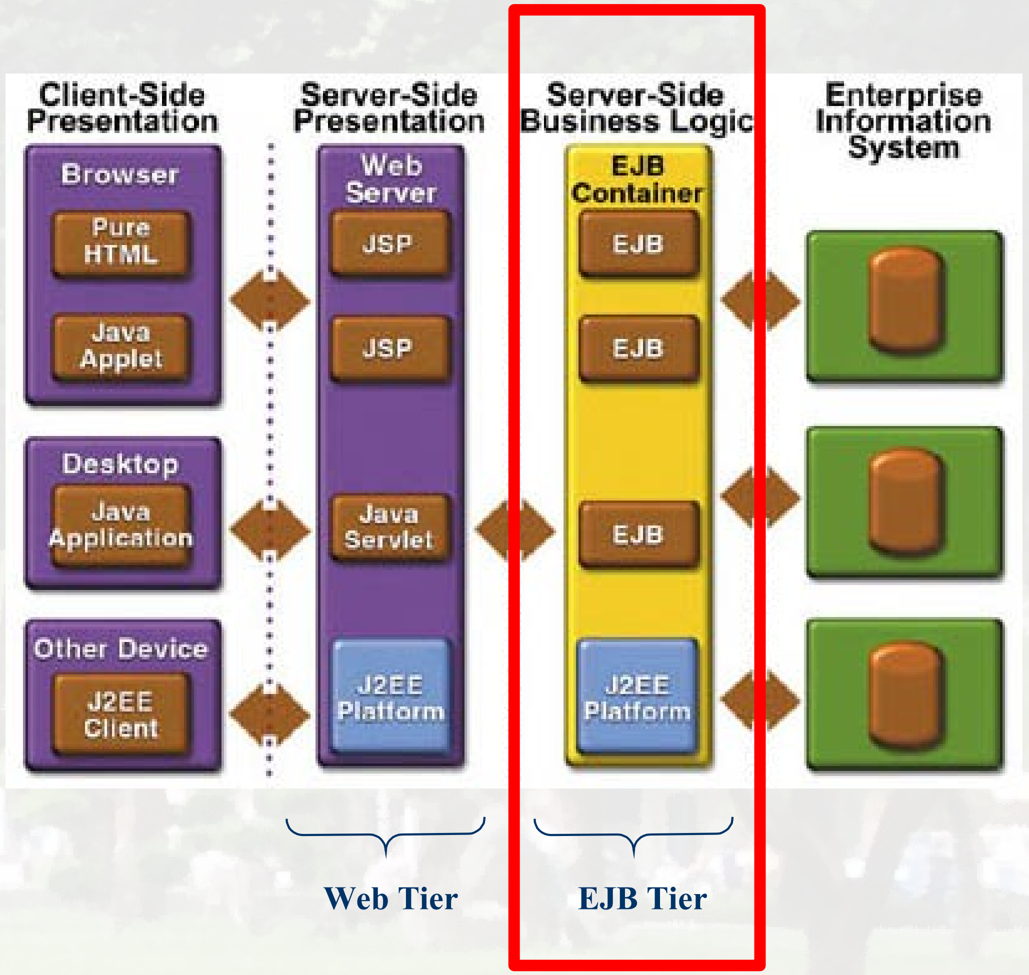
The Deployment Descriptor

- Gives the container instructions on how to manage and control behaviors of the J2EE components
  - Transaction
  - Security
  - Persistence
- Allows declarative customization (as opposed to programming customization)
  - XML file
- Enables portability of code

Outline

- N-Tier Model and Containers
- What is J2EE?
- What Makes Up J2EE?
- Architecture
- Development and Deployment of Applications
- **Business Tier: EJBs**
- Enterprise Integration: Distributed Messaging, JMS and MDB
- Presentation Tier: Servlets and JSP
- Data Tier: JDBC

Where are Enterprise Java Beans?



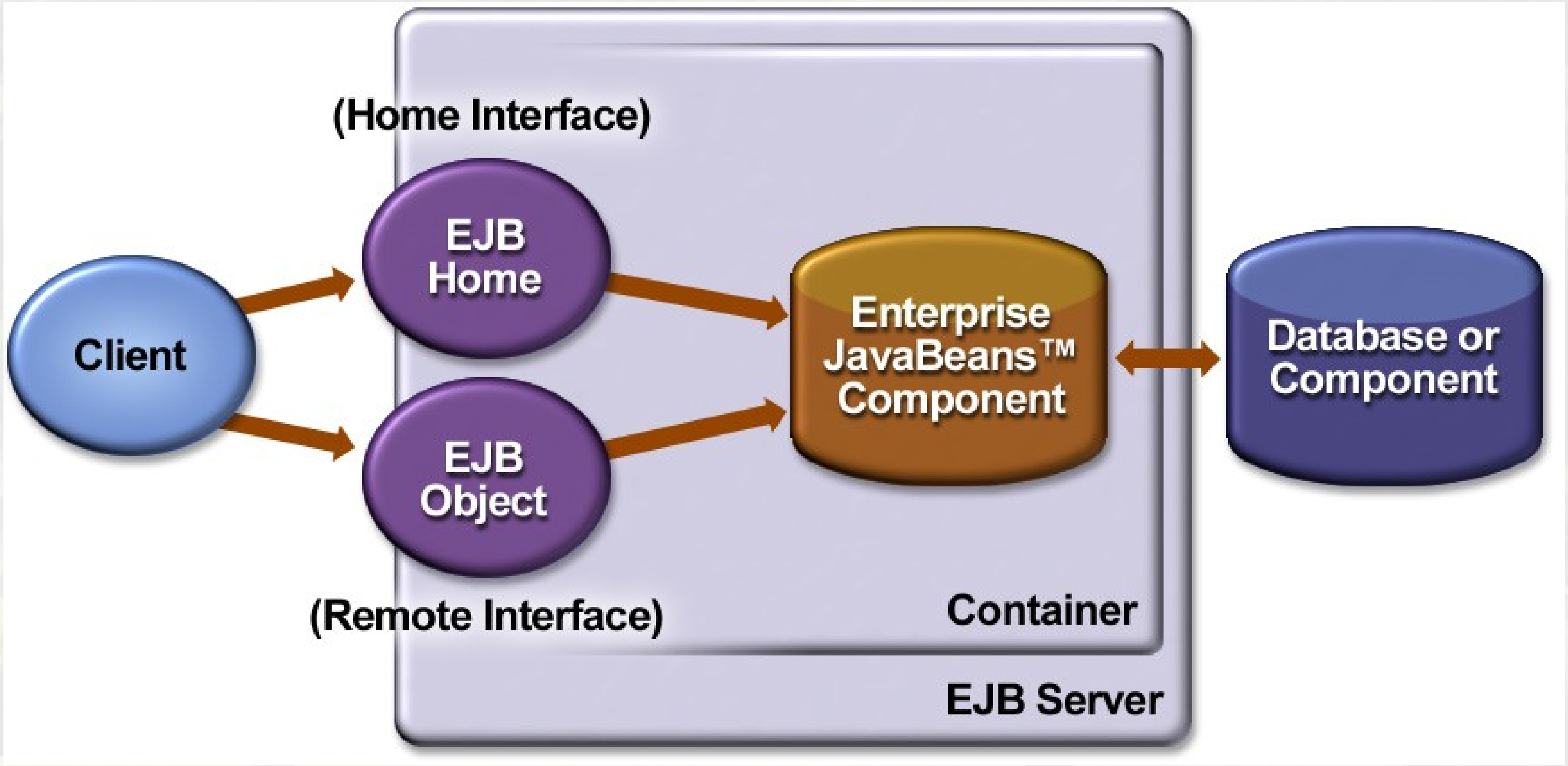
Why EJB Technology?

- Leverages the benefits of **component-model** on the server side
- Separates **business logic** from system code
  - Container provides system services
- Provides framework for **portable components**
  - Over different J2EE-compliant servers
  - Over different operational environments
- Enables **deployment-time configuration**
  - Deployment descriptor

Types of Beans

- Session Beans
  - Stateful session beans
  - Stateless session beans
- Entity Beans
  - Bean Managed Persistence (BMP)
  - Container Managed Persistence (CMP)
- Message Driven Beans
  - JMS (Java Message Service)
  - JAXM (Java API for XML Messaging), SMTP

EJB Architecture



Session Beans

- Does work on behalf of a single client
  - life typically is that of its client
- Is not persistent and hence relatively short lived
  - Is gone when the EJB™ server crashes
- Does not represent data in data store, although can access/update such data
- Can be transaction aware
  - Can perform transaction demarcation

2 Types of Session Beans

- Stateless: execute a request and return a result without saving any client specific state information
  - transient
  - temporary piece of business logic needed by a specific client for a limited time span
- Stateful: maintains client specific state

Examples of Stateless Session Bean

- Catalog
  - No client specific state needs to be preserved
  - Common catalog data for all clients
    - The data can be retrieved from database the first time it is accessed
- Interest calculator
  - No client specific state needs to be preserved
  - Common business logic for all clients

Examples of Stateful Session Bean

- Shopping cart
  - Client specific state needs to be preserved for each client
    - Items that a user wants to buy
  - State will be lost when the server crashes
- Travel ticket purchasing
  - Client specific state needs to be preserved for each client
    - Tickets to purchase and then confirm/cancel

Reusability of Stateless Session Bean Instances

- Container transparently reuses bean instances to serve different clients
  - Pool of bean instances are created by container at appropriate time (ex: at the time of system boot or when the size of pool becomes too small)
  - Bean instances are then recycled
  - Smaller number of bean instances (pool of bean instances) can serve larger number of clients at a single time – Improves scalability of the system
    - clients can be idle between calls

Resource usage of Stateless Session Beans

- Load-balancing & Failover (between EJB servers) is easier since no state needs to be preserved
  - Any bean instance in any EJB server can serve any client call
- High scalability since a client call can be served by any EJB server in a clustered architecture
  - In order to handle increased number of clients, just add more memory or more EJB servers

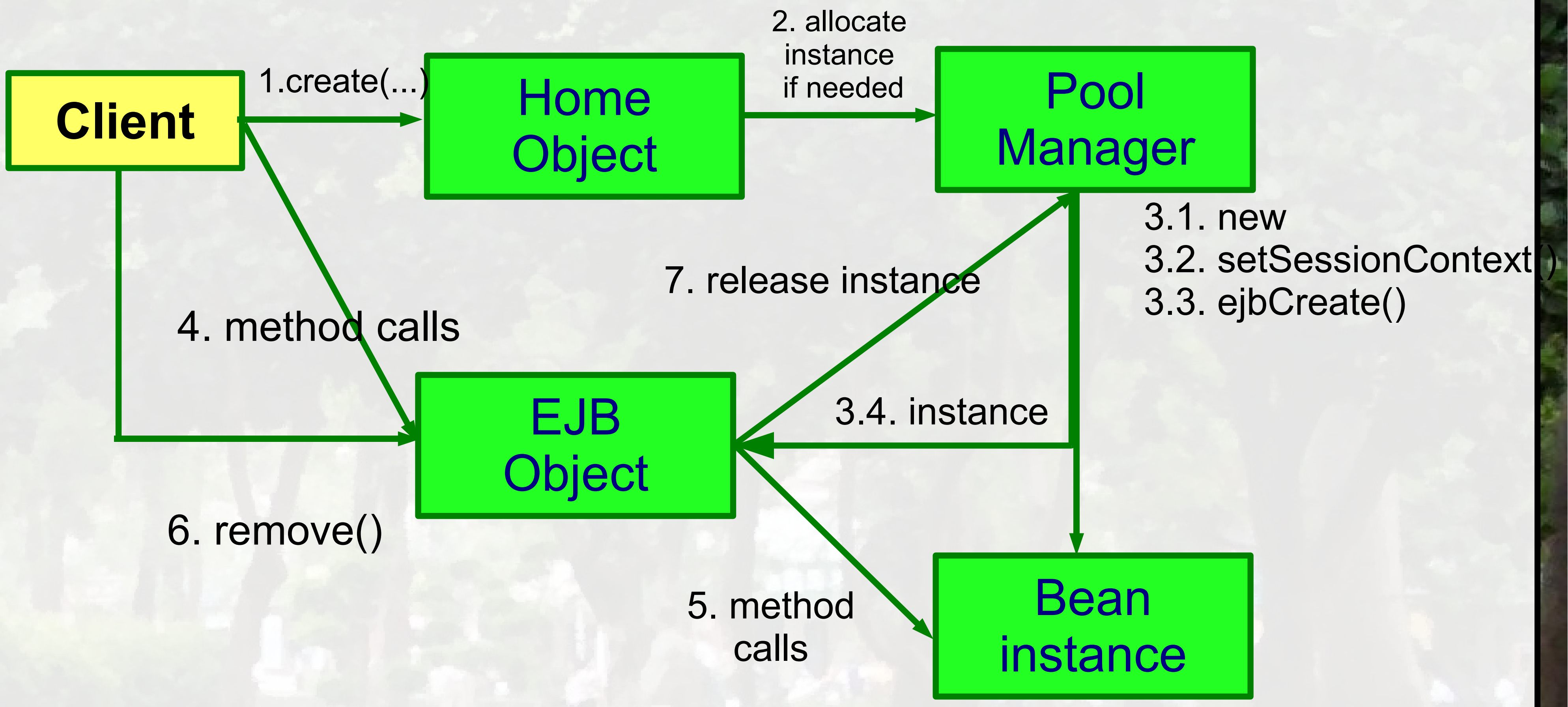
Usage Model of Stateless Session Bean

- Use it when no client specific state needs to be preserved between calls
- If stateless session bean has to deal with client specific request
  - Client then has to pass any needed information as parameters to the business methods
  - But may require the client to maintain state information on the client side which can mean more complex client code

Failover of Stateful Session Bean

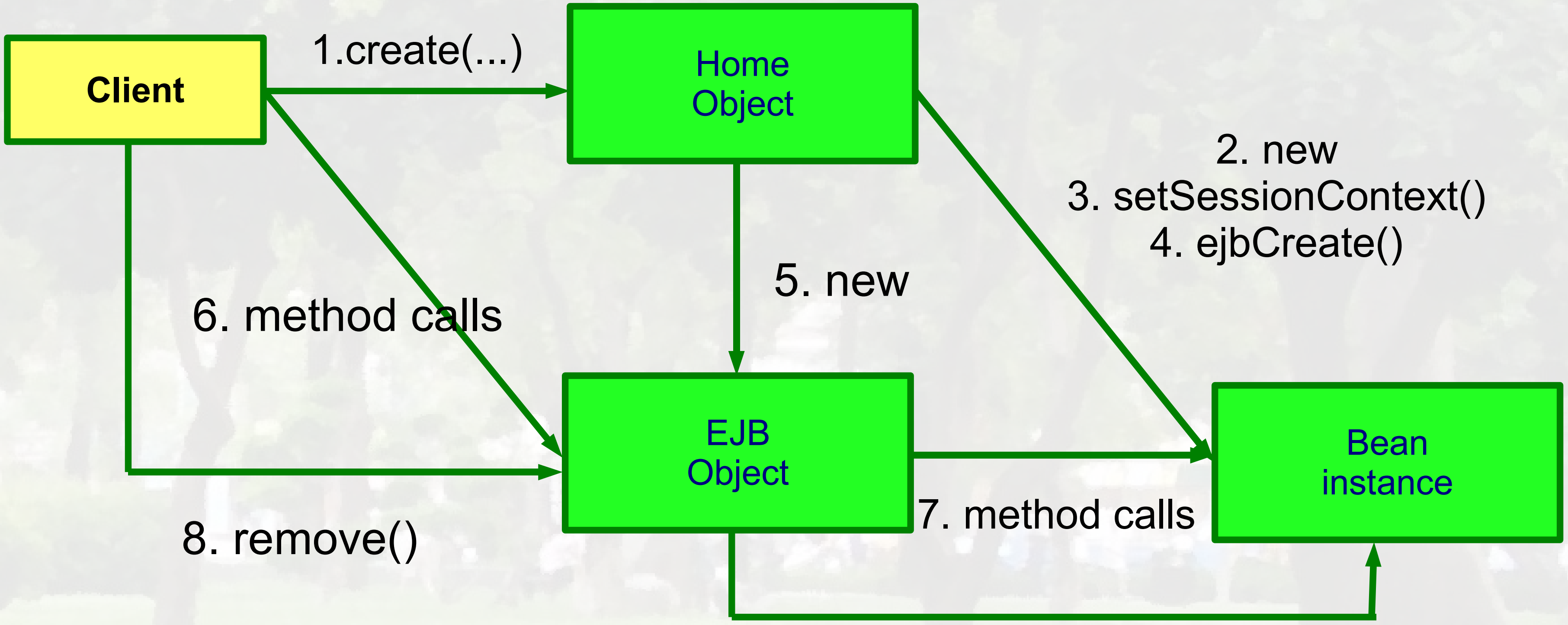
- State is not preserved when a server crashes
- High-end commercial servers can maintain session state even at the time of server failure by
  - maintaining server state in persistent storage
  - maintaining the same state in multiple servers

Interaction between Client, Bean instance, Container for Stateless Session Bean



source: Applied Enterprise JavaBeans

Interaction between Client, Bean instance, Container for Stateful Session Bean



source: Applied Enterprise JavaBeans

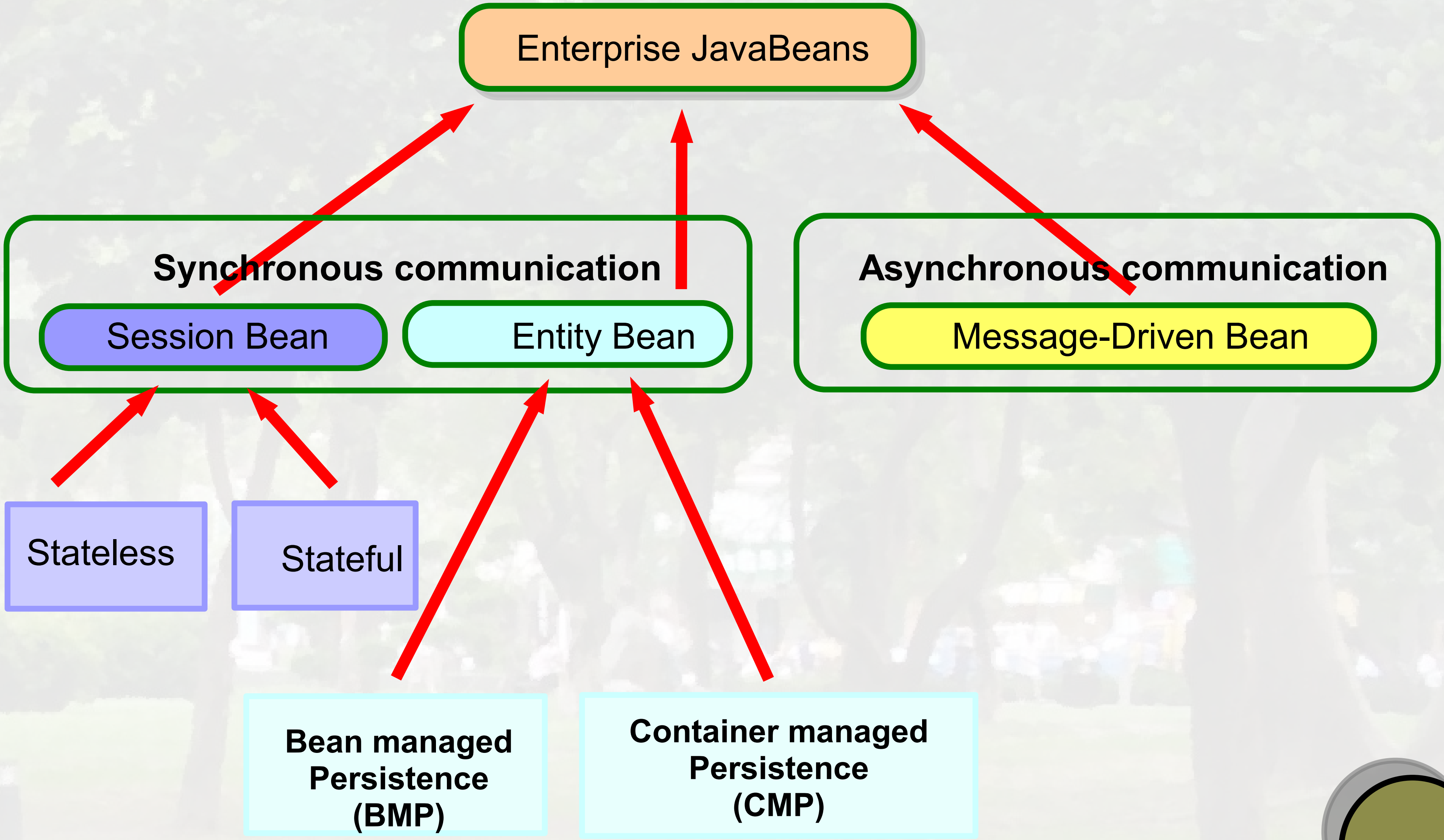
Atm Interface Business Methods

```
public class AtmBean implements SessionBean {
   // implement atm interface business methods
   public void transfer(
          int fromAcctId,
          int toAcctId,
          double amount)
     throws ... {
   try {
     fromAccount = accountHome.findByPrimaryKey(
       new Integer(fromAcctId));
     toAccount = accountHome.findByPrimaryKey(
       new Integer(toAcctId));
     fromAccount.withdraw(amount);
     toAccount.deposit(amount);
   } catch(...) {
       ...
   }
}
```

ATM Client Code

```
// create an initial context (starting point in name tree)
javax.naming.Context ic =new
      javax.naming.InitialContext();

// lookup jndi name (set by deployer in deployment
// descriptor)
java.lang.Object objref = ic.lookup("Atm");

AtmHome home = (AtmHome)PortableRemoteObject.narrow(
objref, AtmHome.class);

//call AtmHome Create method to get Atm interface
Atm atm = home.create();

// call Atm business methods
atm.transfer(41476633, 4443332121, 100000);
```

Enterprise JavaBeans


Enterprise JavaBeans
Synchronous communication
Session Bean
Entity Bean
Asynchronous communication
Message-Driven Bean
Stateless
Stateful
Bean managed Persistence (BMP)
Container managed Persistence (CMP)

Outline

- N-Tier Model and Containers
- What is J2EE?
- What Makes Up J2EE?
- Architecture
- Development and Deployment of Applications
- Business Tier: EJBs
- **Enterprise Integration: Distributed Messaging, JMS and MDB**
- Presentation Tier: Servlets and JSP
- Data Tier: JDBC

Messaging System Concepts

- De-coupled (Loosely-coupled) communication
- Asynchronous communication
- Messages are the means of communication between applications.
- Underlying messaging software provides necessary support
  - MOM (Message Oriented Middleware), Messaging system, Messaging server, Messaging provider, JMS provider: they all mean this underlying messaging software

Messaging System Features

- Support of two messaging models
  - Point-to-point
  - Publish/Subscribe
- Reliability
- Transactional operations
- Distributed messaging
- Security

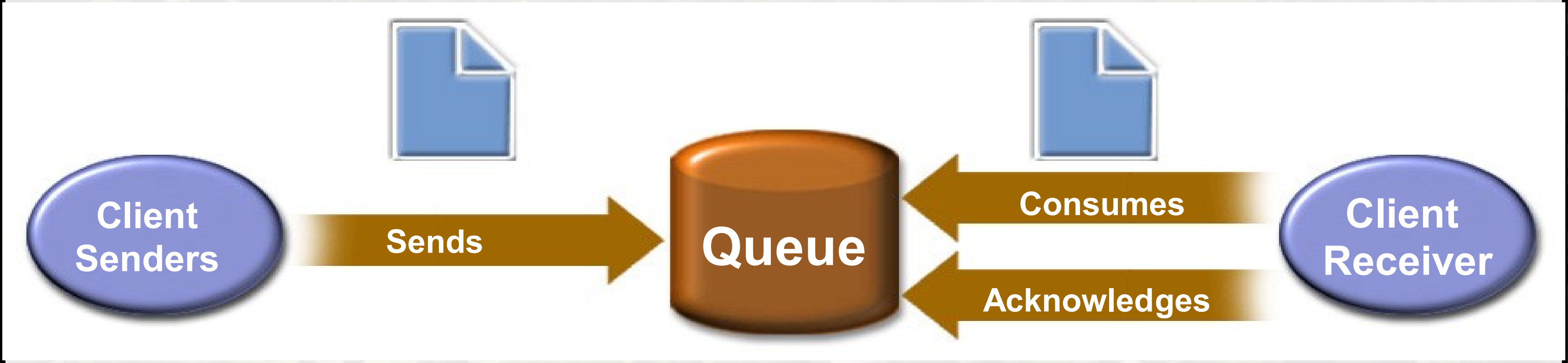
Additional Features

- Some Messaging System vendors support
  - Guaranteed real-time delivery
  - Secure transactions
  - Auditing
  - Metering
  - Load balancing

Point-to-Point

- A message is consumed by a **single** consumer
- "Destination" of a message is a named <u>queue</u>
- First in, first out (at the same priority level)
- Sender (producer) sends a message to a named queue (with a priority level)
- Receiver (consumer) extracts a message from the queue

Point-to-Point



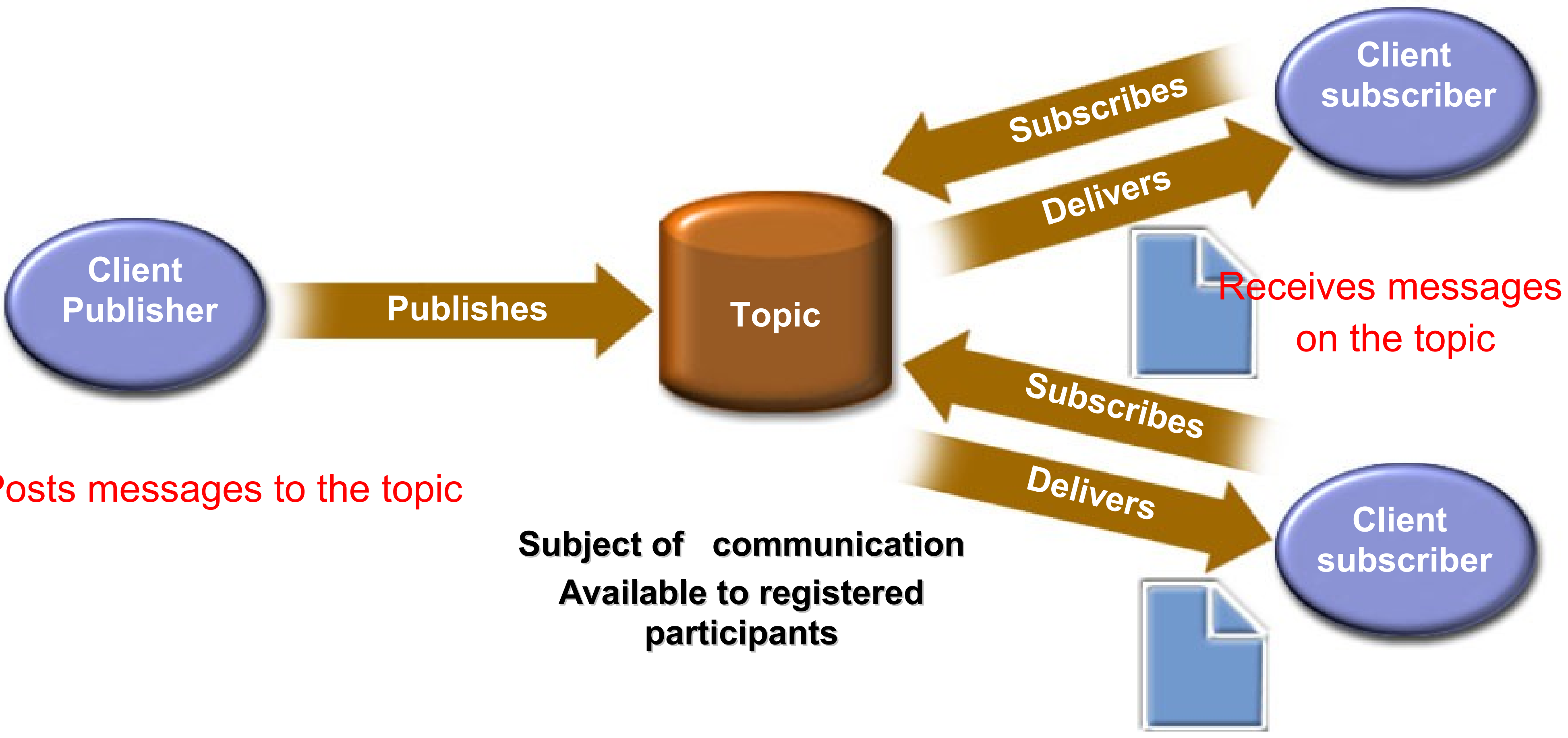
Posts messages to the queue

Receives messages on the queue

Publish/Subscribe (Pub/Sub)

- A message is consumed by **multiple** consumers
- "Destination" of a message is a named topic
- Producers “publish” to topic
- Consumers “subscribe” to topic

Publish-and-Subscribe



Receives messages on the topic

Posts messages to the topic

Subject of communication
Available to registered participants

When to use Pub/Sub?

- Use it when a message you send need to be processed by multiple consumers
- Example: HR application
  - Create “new hire” topic
  - Many applications (“facilities”, “payroll”, etc.) subscribe “new hire” topic

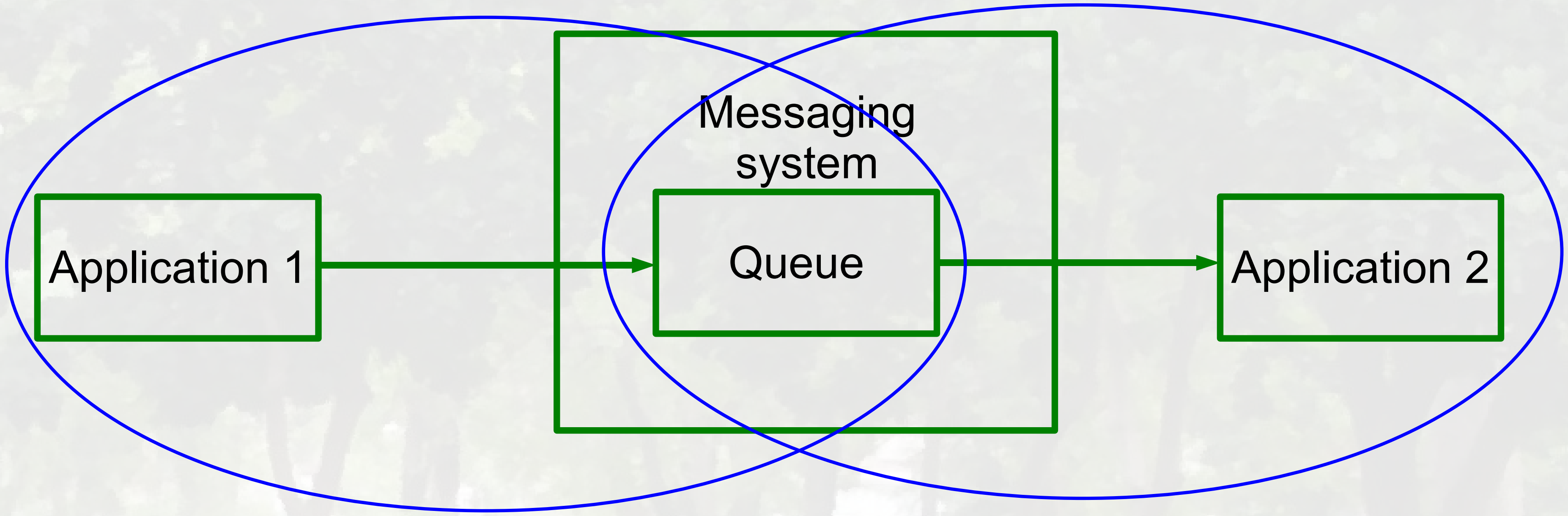
Transactional Operations

- Transactional production
  - Sender groups a series of messages into a transaction
  - Either all messages are enqueued successfully or none are
- Transactional consumption
  - Consumer retrieves a group of messages as a transaction
  - Unless all messages are retrieved successfully, the messages remain in a queue or topic

Transactional Scope

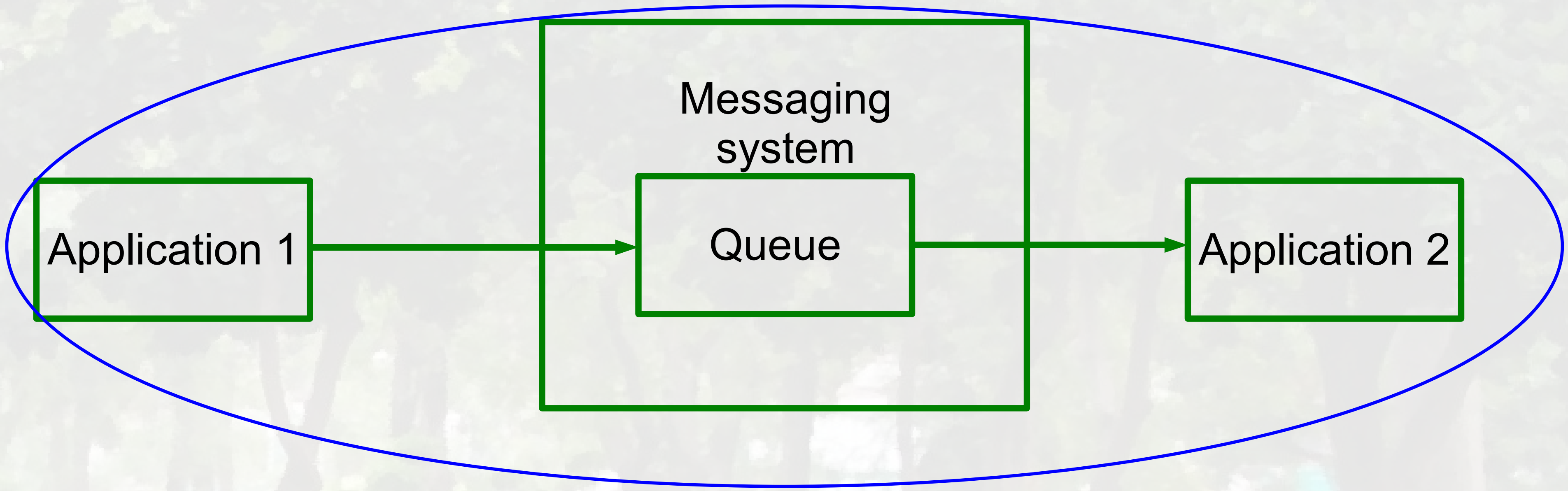
- Client-to-Messaging system scope
  - Transaction encompasses the interaction between each messaging client (applications) and the messaging system
  - JMS supports this
- Client-to-Client scope
  - Transaction encompasses both clients
  - **JMS does not support this**

Client-to-Messaging System Transactional Scope



source: Applied Enterprise JavaBeans[1]

Client-to-Client Transactional Scope



source: Applied Enterprise JavaBeans[1]

What is JMS?

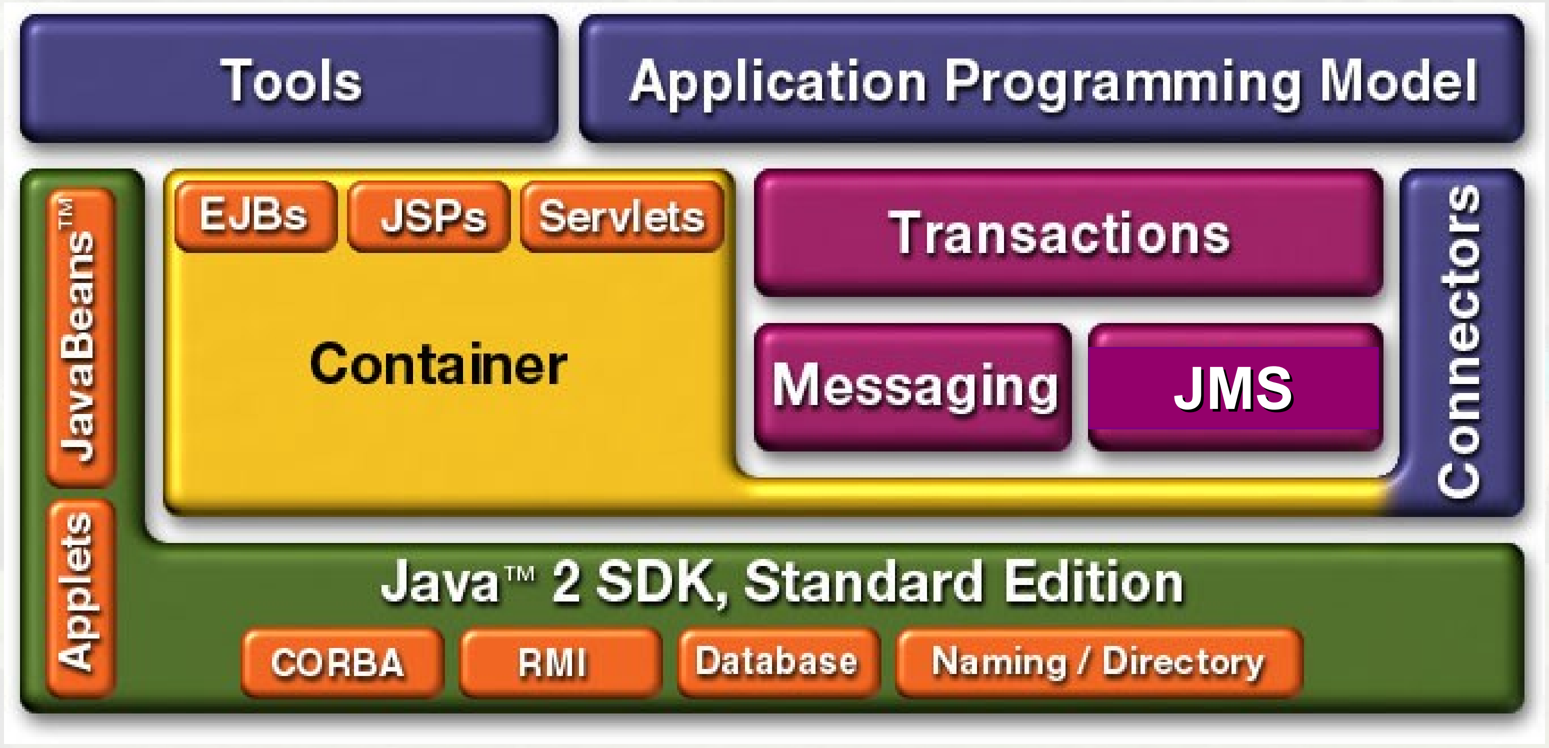
- **JMS** is a set of Java interfaces and associated semantics (APIs) that define how a JMS client accesses the facilities of a messaging system
- Supports message production, distribution, delivery
- Supported message delivery semantics
  - Synchronous or Asynchronous
  - transacted
  - Guaranteed
  - Durable

What is JMS? (Continued)

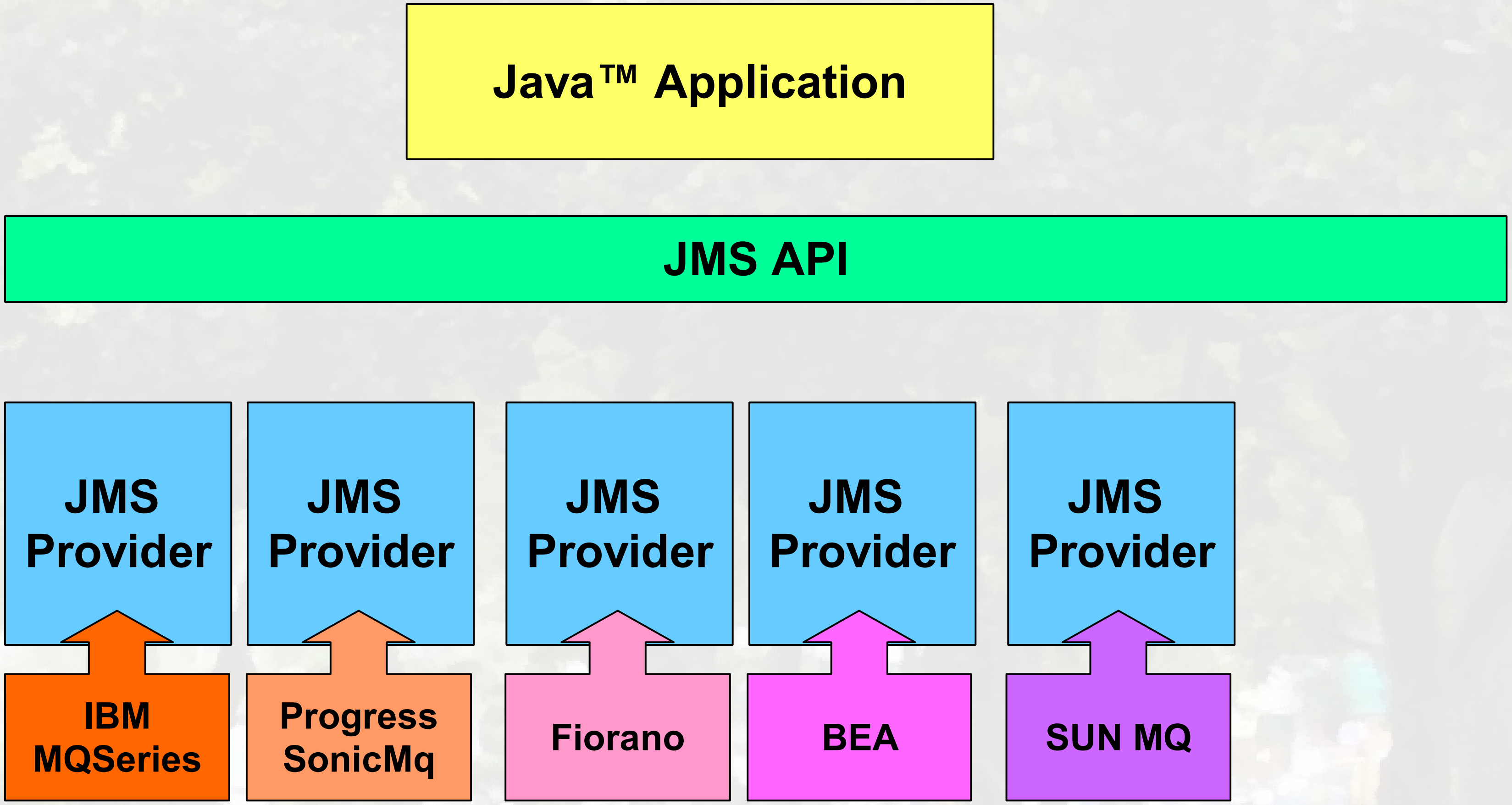
- Supports existing messaging models
  - Point-to-Point (reliable queue)
  - Publish/Subscribe
- Message selectors (on the receiver side)
- 5 Message types

JMS and J2EE

- Allows Java Developers to access the power of messaging systems
- Part of the J2EE Enterprise Suite



JMS is an API


Java™ Application
JMS API
JMS Provider
JMS Provider
JMS Provider
JMS Provider
JMS Provider
IBM MQSeries
Progress SonicMq
Fiorano
BEA
SUN MQ

Steps for Building a JMS Sender Application

1. Get ConnectionFactory and Destination object (Topic or Queue) through JNDI
2. Create a Connection
3. Create a Session to send/receive messages
4. Create a MessageProducer (TopicPublisher or QueueSender)
5. Start Connection
6. Send (publish) messages
7. Close Session and Connection

Locate ConnectionFactory and Destination objects via JNDI

```
// Get JNDI InitialContext object
Context jndiContext =  new InitialContext();

// Locate ConnectionFactory object via JNDI
TopicConnectionFactory factory =
     (TopicConnectionFactory) jndiContext.lookup(
     "MyTopicConnectionFactory");

// Locate Destination object (Topic or Queue)
// through JNDI
Topic weatherTopic =
   (Topic) jndiContext.lookup("WeatherData");
```

Create Connection Object, Session and Publisher

```
// Create a Connection object from ConnectionFactory object
TopicConnection topicConnection =
      factory.createTopicConnection();

// Create a Session from Connection object.
//       1st parameter controls transaction
//       2nd parameter specifies acknowledgment type
TopicSession session =
      topicConnection.createTopicSession (false,
                        Session.CLIENT_ACKNOWLEDGE);

// Create MessageProducer from Session object
//   TopicPublisher for Pub/Sub
//   QueueSender for Point-to-Point
TopicPublisher publisher =
         session.createPublisher(weatherTopic);
```

Start Connection and Publish Message

```
// Until Connection gets started, message flow
// is inhibited: Connection must be started
// before messages will be transmitted.
topicConnection.start();

// Create a Message
TextMessage message = session.createMessage();
message.setText("text:35 degrees");

// Publish the message
publisher.publish(message);
```

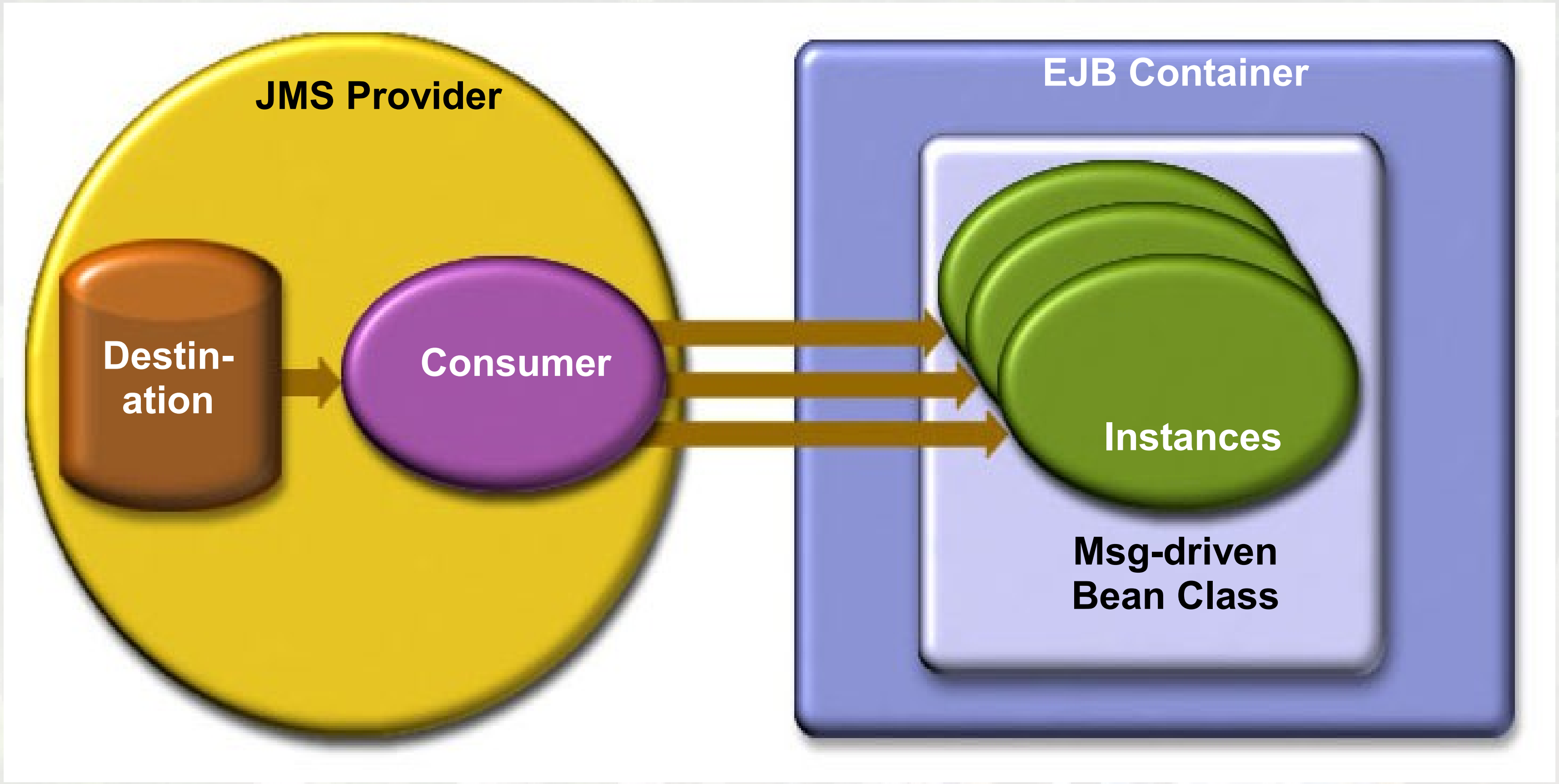
Steps for Building a JMS Receiver Application (non-blocking mode)

1. Get ConnectionFactory and Destination object (Topic or Queue) through JNDI
2. Create a Connection
3. Create a Session to send/receive messages
4. Create a MessageConsumer (TopicSubscriber or QueueReceiver)
5. Register MessageListener for non-blocking mode
6. Start Connection
7. Close Session and Connection

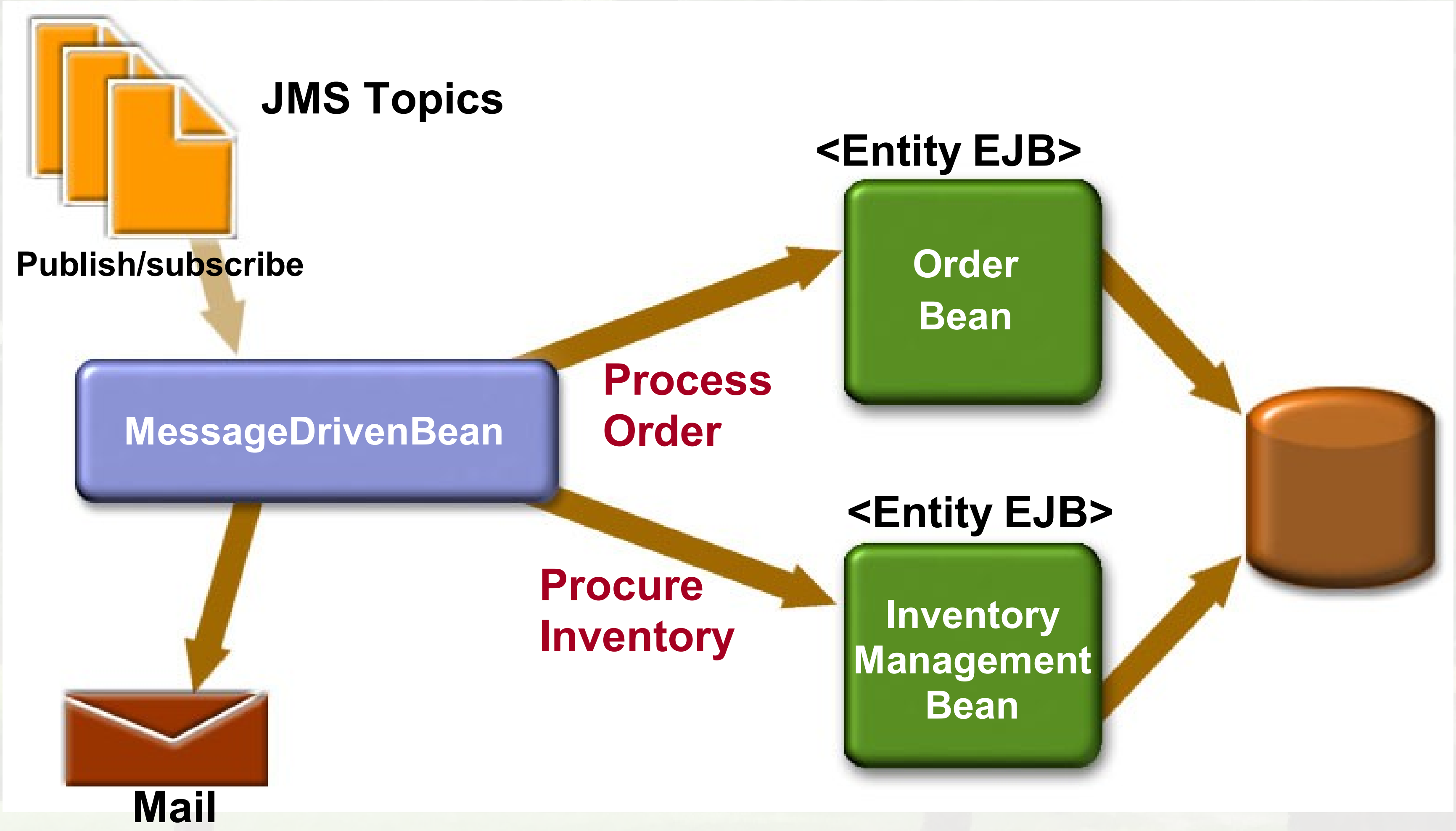
Create Message Subscriber, non-blocking listener and

```
// Create Subscriber from Session object
TopicSubscriber subscriber =
   session.createSubscriber(weatherTopic);

// Create MessageListener object
WeatherListener myListener
       = new WeatherListener();

// Register MessageListener with
// TopicSubscriber object
subscriber.setMessageListener(myListener);
```

JMS and MDB


JMS Provider
Destin-
ation
Consumer
EJB Container
Instances
Msg-driven
Bean Class

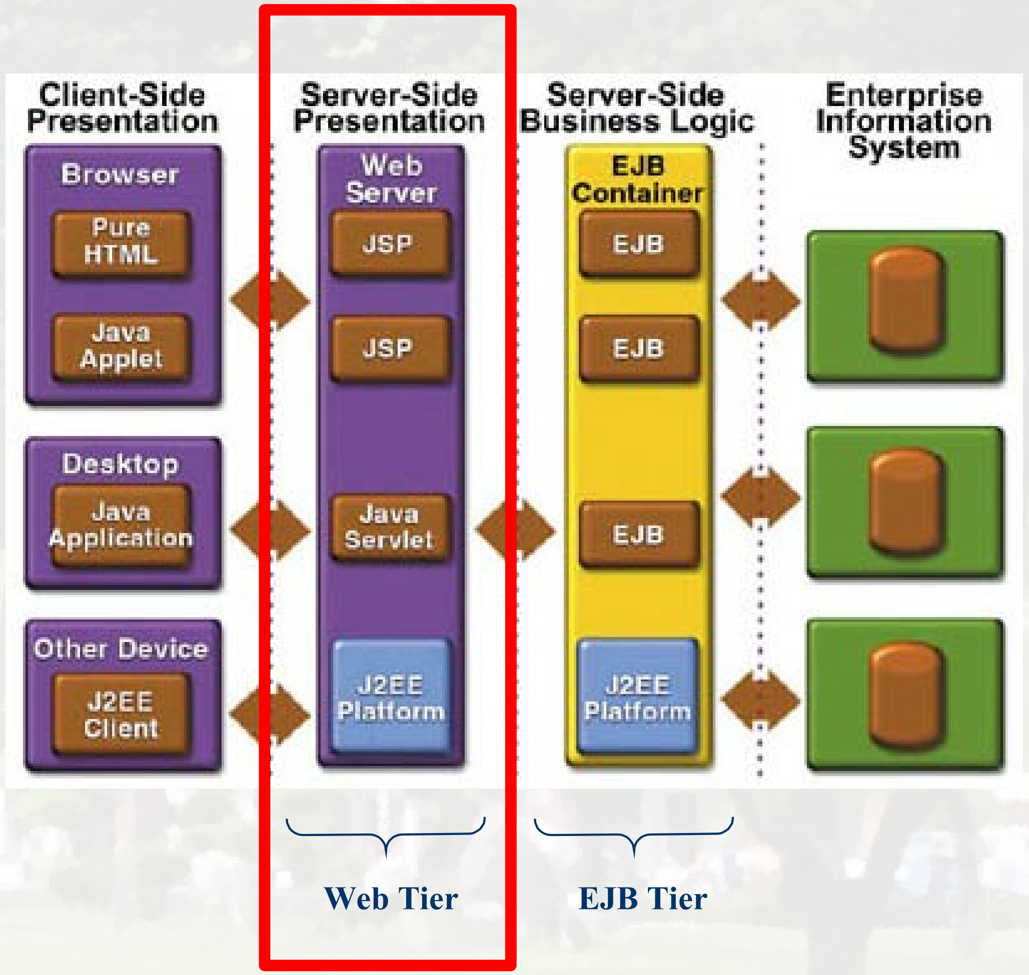
MDB Example


JMS Topics
Publish/subscribe
<Entity EJB>
Order Bean
MessageDrivenBean
Process Order
<Entity EJB>
Procure Inventory
Inventory Management Bean
Mail

Outline

- N-Tier Model and Containers
- What is J2EE?
- What Makes Up J2EE?
- Architecture
- Development and Deployment of Applications
- Business Tier: EJBs
- Enterprise Integration: Distributed Messaging, JMS and MDB
- **Presentation Tier: Servlets and JSP**
- Data Tier: JDBC

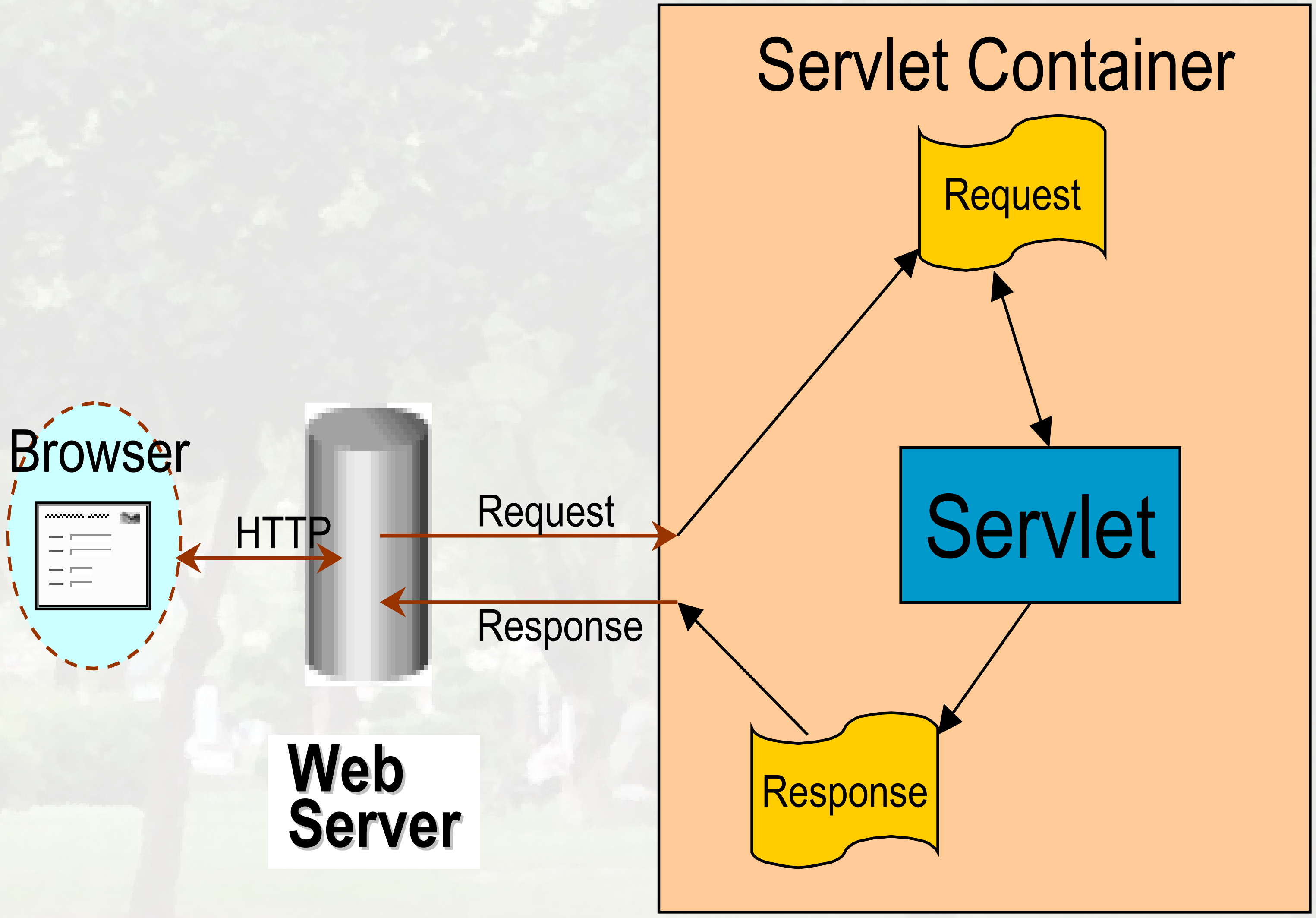
Where are Servlet and JSP?



What is Servlet?

- Java™ objects which are based on servlet framework and APIs and extend the functionality of a HTTP server.
- Mapped to URLs and managed by container with a simple architecture
- Available and running on all major web servers and app servers
- Platform and server independent

Servlet Request and Response Model



First Servlet Code

```
Public class HelloServlet extends HttpServlet {

   public void doGet(HttpServletRequest request,
                     HttpServletResponse response){
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<title>Hello World!</title>");
   }
   ...
}
```

Advantages of Servlet

- No CGI limitations
- Abundant third-party tools and Web servers supporting Servlet
- Access to entire family of Java APIs
- Reliable, better performance and scalability
- Platform and server independent
- Secure
- Most servers allow automatic reloading of Servlet's by administrative action

What is JSP Technology?

- Enables separation of business logic from presentation
  - Presentation is in the form of HTML or XML/XSLT
  - Business logic is implemented as Java Beans or custom tags
  - Better maintainability, reusability
- Extensible via custom tags
- Builds on Servlet technology

What is JSP page?

- A text-based document capable of returning dynamic content to a client browser
- Contains both static and dynamic content
  - Static content: HTML, XML
  - Dynamic content: programming code, and JavaBeans, custom tags

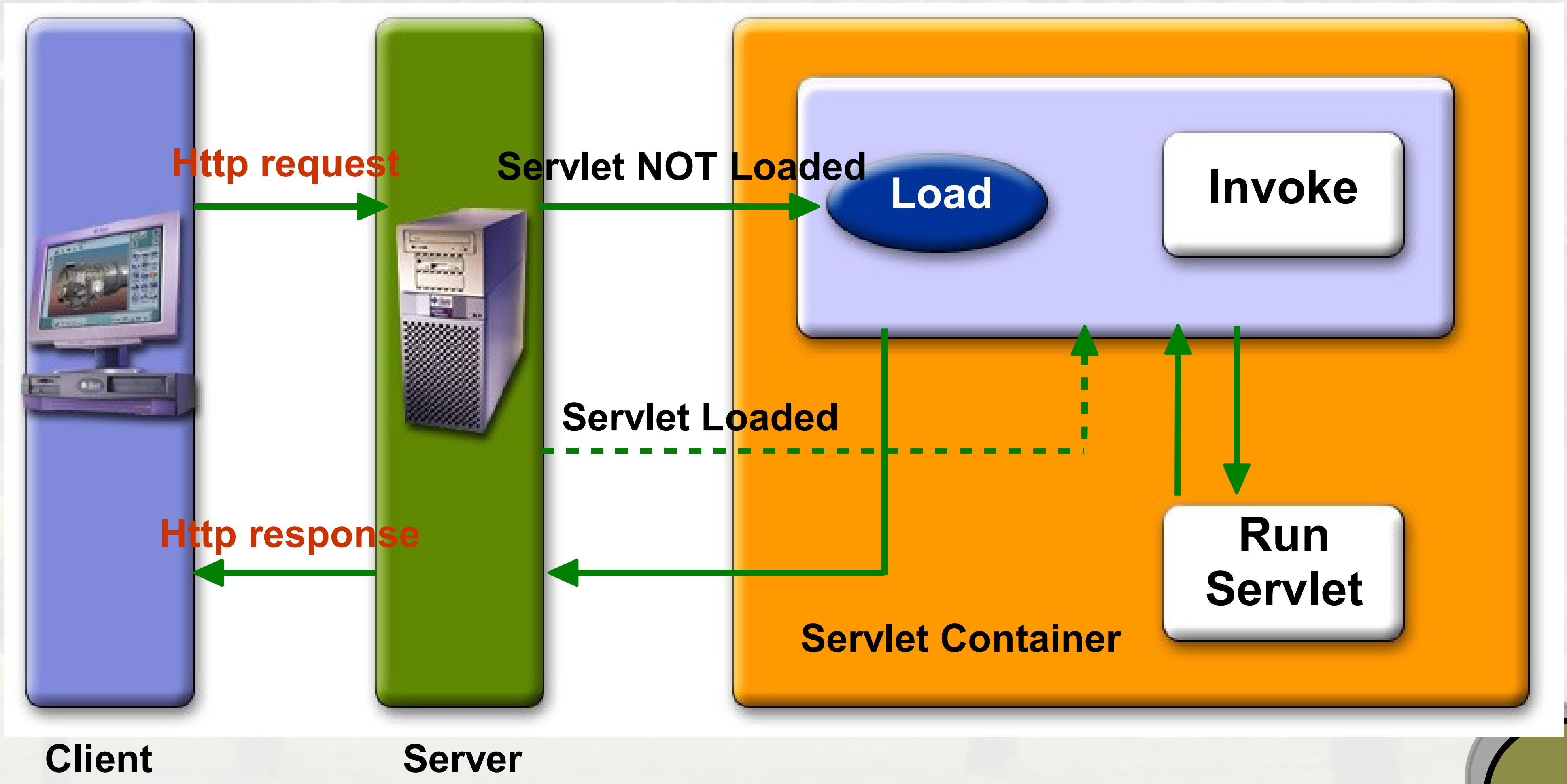
JSP Sample Code

```
<html>
   Hello World!
  <br>
  <jsp:useBean id="clock"
               class=“calendar.JspCalendar” />
   Today is
  <ul>
  <li>Day of month: <%= clock.getDayOfMonth() %>
  <li>Year: <%= clock.getYear() %>
  </ul>
</html>
```

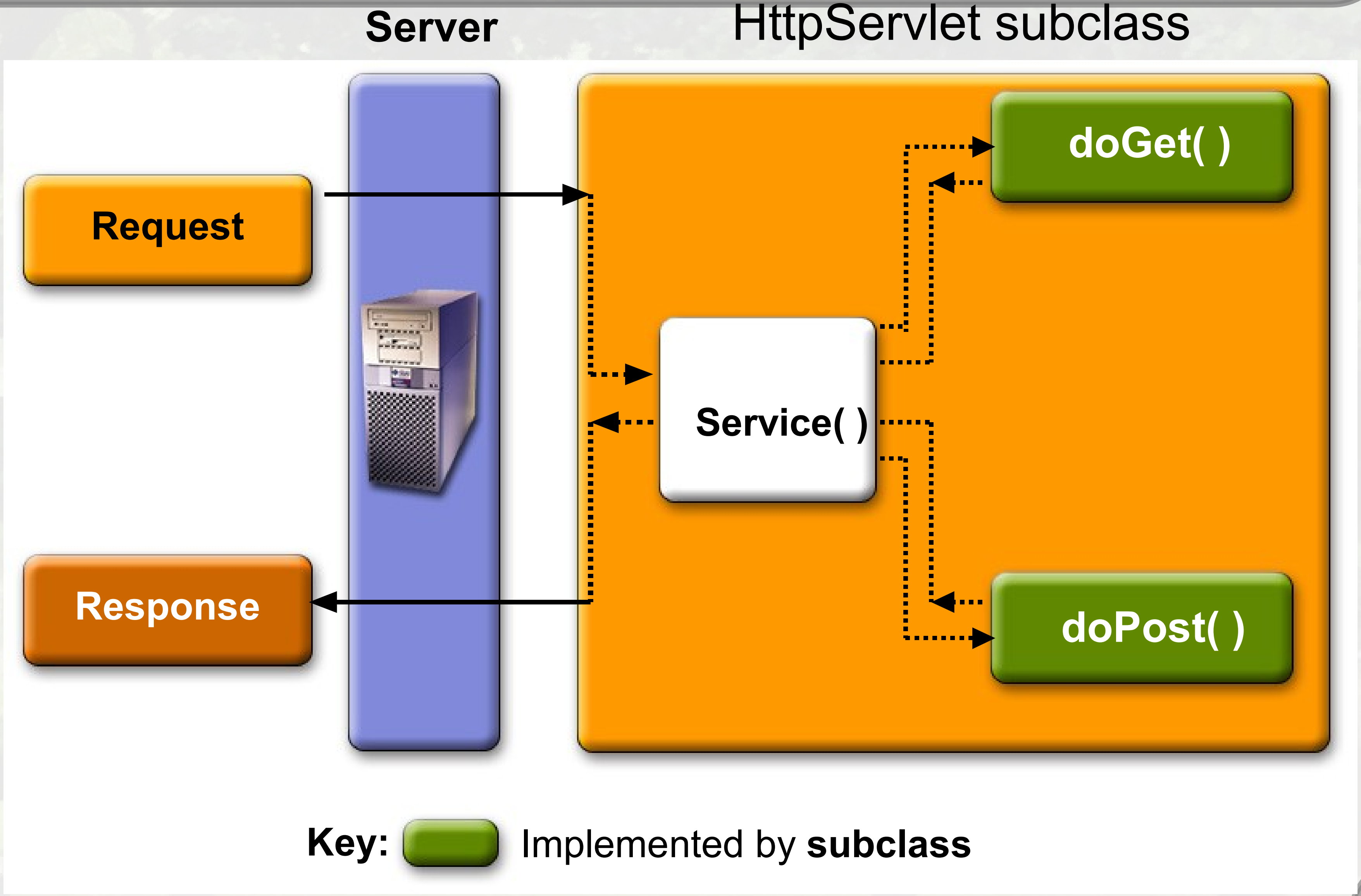
Should I Use Servlet or JSP?

- In practice, servlet and JSP are used together
  - via MVC (Model, View, Controller) architecture
  - Servlet handles Controller
  - JSP handles View

Servlet Life-Cycle



doGet() and doPost() Methods



Things You Do in doGet() & doPost()

- Extract client-sent information (HTTP parameter) from HTTP request
- Set (Save) and get (read) attributes to/from Scope objects
- Perform some business logic or access database
  - **EJB**
  - JDBC
- Optionally forward the request to other Web components (Servlet or JSP)
- Populate HTTP response message and send it to client

Scope Objects

- Enables sharing information among collaborating web components via attributes maintained in Scope objects
  - Attributes are name/object pairs
- Attributes maintained in the Scope objects are accessed with
  - getAttribute() & setAttribute()
- 4 Scope objects are defined
  - Web context, session, request, page

Four Scope Objects: Accessibility

- Web context (ServletConext)
  - Accessible from Web components within a Web context
- Session
  - Accessible from Web components handling a request that belongs to the session
- Request
  - Accessible from Web components handling the request
- Page
  - Accessible from JSP page that creates the object

What is ServletContext For?

- Used by servlets to
  - Set and get context-wide (application-wide) object-valued attributes
  - Get request dispatcher
    - To forward to or include web component
  - Access Web context-wide initialization parameters set in the web.xml file
  - Access Web resources associated with the Web context
  - Log
  - Access other misc. information

Why HttpSession?

- Need a mechanism to **maintain client state** across a series of requests from a same user (or originating from the same browser) over some period of time
  - Example: Online shopping cart
- Yet, HTTP is stateless
- HttpSession maintains client state
  - Used by Servlets to set and get the values of session scope attributes

What is Servlet Request?

- Contains data passed from client to servlet
- All servlet requests implement ServletRequest interface which defines methods for accessing
  - Client sent parameters
  - Object-valued attributes
  - Locales
  - Client and server
  - Input stream
  - Protocol information
  - Content type
  - If request is made over secure channel (HTTPS)

HTTP Request URL: [request path]

- http://[host]:[port]/[request path]?[query string]
- [request path] is made of
  - Context: /<context of web app>
  - Servlet name: /<component alias>
  - Path information: the rest of it
- Examples
  - http://localhost:8080/hello1/greeting
  - http://localhost:8080/hello1/greeting.jsp
  - http://daydreamer/catalog/lawn/index.html

What is Servlet Response?

- Contains data passed from servlet to client
- All servlet responses implement ServletResponse interface
  - Retrieve an output stream
  - Indicate content type
  - Indicate whether to buffer output
  - Set localization information
- HttpServletResponse extends ServletResponse
  - HTTP response status code
  - Cookies

Outline

- N-Tier Model and Containers
- What is J2EE?
- What Makes Up J2EE?
- Architecture
- Development and Deployment of Applications
- Business Tier: EJBs
- Enterprise Integration: Distributed Messaging, JMS and MDB
- Presentation Tier: Servlets and JSP
- **Data Tier: JDBC**

What is JDBC?

- Standard Java API for accessing relational database
  - Hides database specific details from application
- Part of J2SE

JDBC API

- Defines a set of Java Interfaces, which are implemented by vendor-specific JDBC Drivers
  - Applications use this set of Java interfaces for performing database operations
- Majority of JDBC API is located in java.sql package
  - DriverManager, Connection, ResultSet, DatabaseMetaData, ResultSetMetaData, PreparedStatement, CallableStatement and Types
- Other advanced functionality exists in the javax.sql package
  - DataSource

JDBC Driver

- Database specific implemention of JDBC interfaces
  - Every database server has corresponding JDBC driver(s)
- http://industry.java.sun.com/products/jdbc/drivers

Database URL

- Used to make a connection to the database
  - Can contain server, port, protocol etc…
- jdbc:subprotocol_name:driver_dependant_databasename
  - Oracle thin driver
    1. jdbc:oracle:thin:@machinename:1521:dbname
  - Pointbase
    - jdbc:pointbase:server://localhost/sample

Steps of Using JDBC

1. Load DB-specific JDBC driver
2. Get a Connection object
3. Get a Statement object
4. Execute queries and/or updates
5. Read results
6. Read Meta-data (optional step)
7. Close Statement and Connection objects

JNDI Registration of a DataSource (JDBC Resource) Object

- The JNDI name of a JDBC resource is expected in the java:comp/env/jdbc subcontext
  - For example, the JNDI name for the resource of a BookDB database could be java:comp/env/jdbc/BookDB
- Because all resource JNDI names are in the java:comp/env subcontext, when you specify the JNDI name of a JDBC resource enter only jdbc/name. For example, for a payroll database, specify jdbc/BookDB

Why Connection Pooling?

- Database connection is an expensive and limited resource
  - Using connection pooling, a smaller number of connections are shared by a larger number of clients
- Creating and destroying database connections are expensive operations
  - Using connection pooling, a set of connections are pre-created and are available as needed basis cutting down on the overhead of creating and destroying database connections

Connection Pooling & DataSource

- DataSource objects that implement connection pooling also produce a connection to the particular data source that the DataSource class represents
- The connection object that the getConnection method returns is a handle to a PooledConnection object rather than being a physical connection
  - The application code works the same way

Retrieval and Usage of a DataSource Object

- Application perform JNDI lookup operation to retrieve DataSource object
- DataSource object is then used to retrieve a Connection object
- In the application's web.xml, information on external resource, DataSource object in this case, is provided
- For Sun Java System App server, the mapping of external resource and JNDI name is provided
  - This provides further flexibility

Example: Retrieval of DataSource Object via JNDI

- BookDBAO.java in bookstore1 application

```
public class BookDBAO {
    private ArrayList books;
    Connection con;
    private boolean conFree = true;

    public BookDBAO() throws Exception {
        try {
            Context initCtx = new InitialContext();
            Context envCtx = (Context) initCtx.lookup(
                 "java:comp/env");
            DataSource ds = (DataSource) envCtx.lookup(
                 "jdbc/BookDB");
            con = ds.getConnection();
        } catch (Exception ex) {
            ...
        }
    }
```

Transaction

- One of the main benefits to using a PreparedStatement is executing the statements in a transactional manner
- The committing of each statement when it is first executed is very time consuming
- By setting AutoCommit to false, the developer can update the database more then once and then commit the entire transaction as a whole
- Also, if each statement is dependant on the other, the entire transaction can be rolled back and the user notified.

JDBC Transaction Methods

- setAutoCommit()
  - If set true, every executed statement is committed immediately
- commit()
  - Relevant only if setAutoCommit(false)
  - Commit operations performed since the opening of a Connection or last commit() or rollback() calls
- rollback()
  - Relevant only if setAutoCommit(false)
  - Cancels all operations performed

Transactions Example

```
Connection connection = null;
try {
    connection = DriverManager.getConnection
        ("jdbc:oracle:thin:@machinename:1521:dbname",
         "username","password");
    connection.setAutoCommit(false);
    PreparedStatement updateQty =
        connection.prepareStatement(
            "UPDATE STORE_SALES SET QTY = ?”
            +” WHERE ITEM_CODE = ? ");
    int [][] arrValueToUpdate = { {123, 500} , {124, 250},
            {125, 10}, {126, 350} };
```

Transaction Example cont.

```
int iRecordsUpdate = 0;
for ( int items=0 ; items < arrValueToUpdate.length ;
   items++) {
    int itemCode = arrValueToUpdate[items][0];
    int qty = arrValueToUpdate[items][1];
    updateQty.setInt(1,qty);
    updateQty.setInt(2,itemCode);
    iRecordsUpdate += updateQty.executeUpdate();
}
connection.commit();
System.out.println(iRecordsUpdate + " record(s) have
   been updated");
```

Transaction Example cont.

```
try  {
    connection.rollback();
} catch(SQLException sqleRollback) {
    System.out.println("" + sqleRollback);
} finally {
    try  {
        connection.close();
    } catch(SQLException sqleClose) {
        System.out.println("" + sqleClose);
    }
}
```

Resources

- Partially based on Shang Shin's Java Passion Slides
  - http://www.javapassion.com/j2ee/
- J2EE Home page
  - java.sun.com/j2ee
- J2EE 1.4 SDK
  - java.sun.com/j2ee/1.4/download.html#appserv
- J2EE 1.4 Tutorial
  - java.sun.com/j2ee/1.4/download.html#appserv
- J2EE Blueprints
  - java.sun.com/blueprints/enterprise/index.html

A Brief Introduction to J2EE

Thank You