15

Managing Transactions with Session and Message-Driven Beans

Objectives

After completing this lesson, you should be able to do the following:

- Choose the appropriate type of transaction management
- Set the transaction attribute for container-managed transactions
- Create transaction demarcations

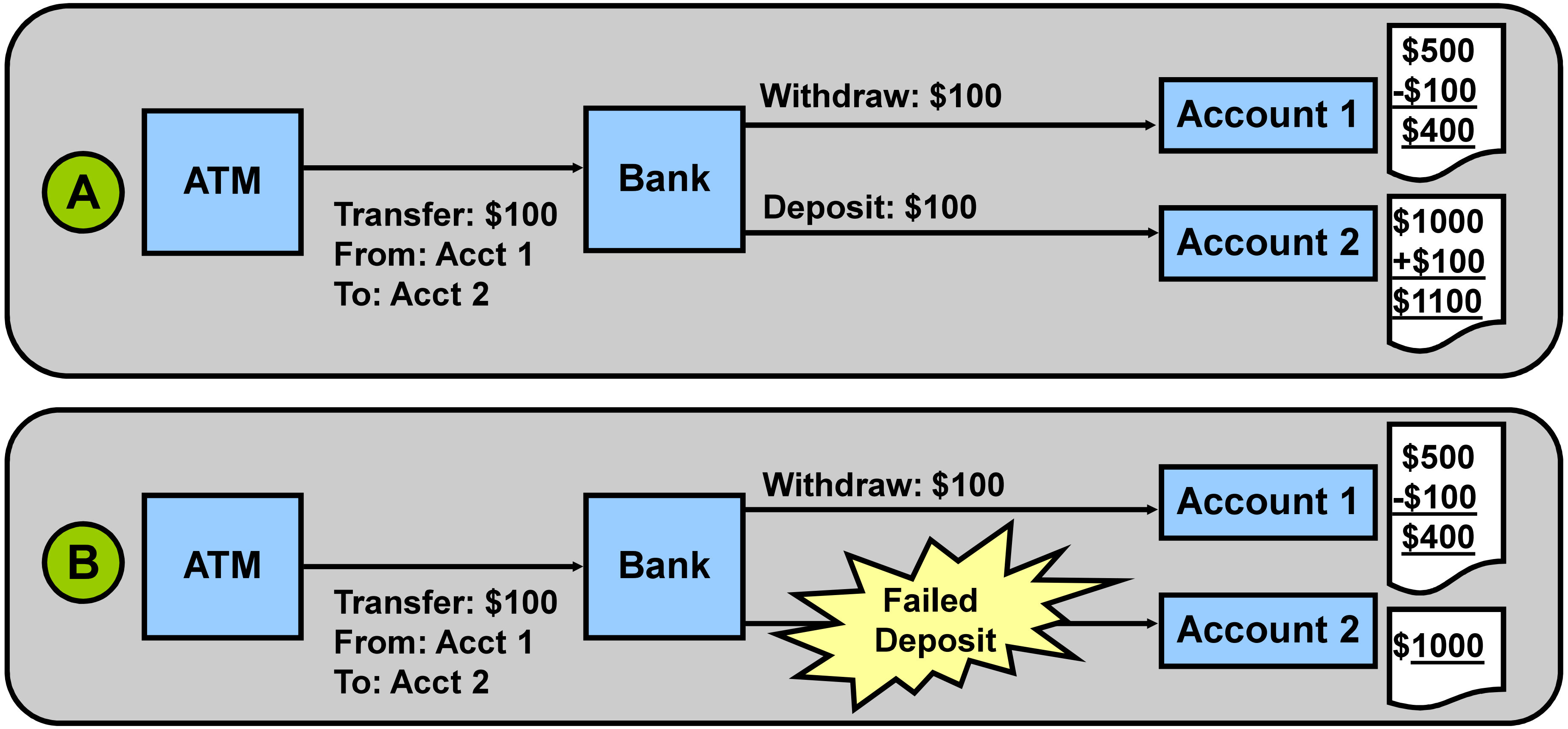
What Is a Transaction?

A transaction:

- Is a single, logical unit of work or a set of tasks that are executed together
- May access one or more shared resources (such as databases)
- Must be atomic, consistent, isolated, and durable (ACID)

Example of a Transaction

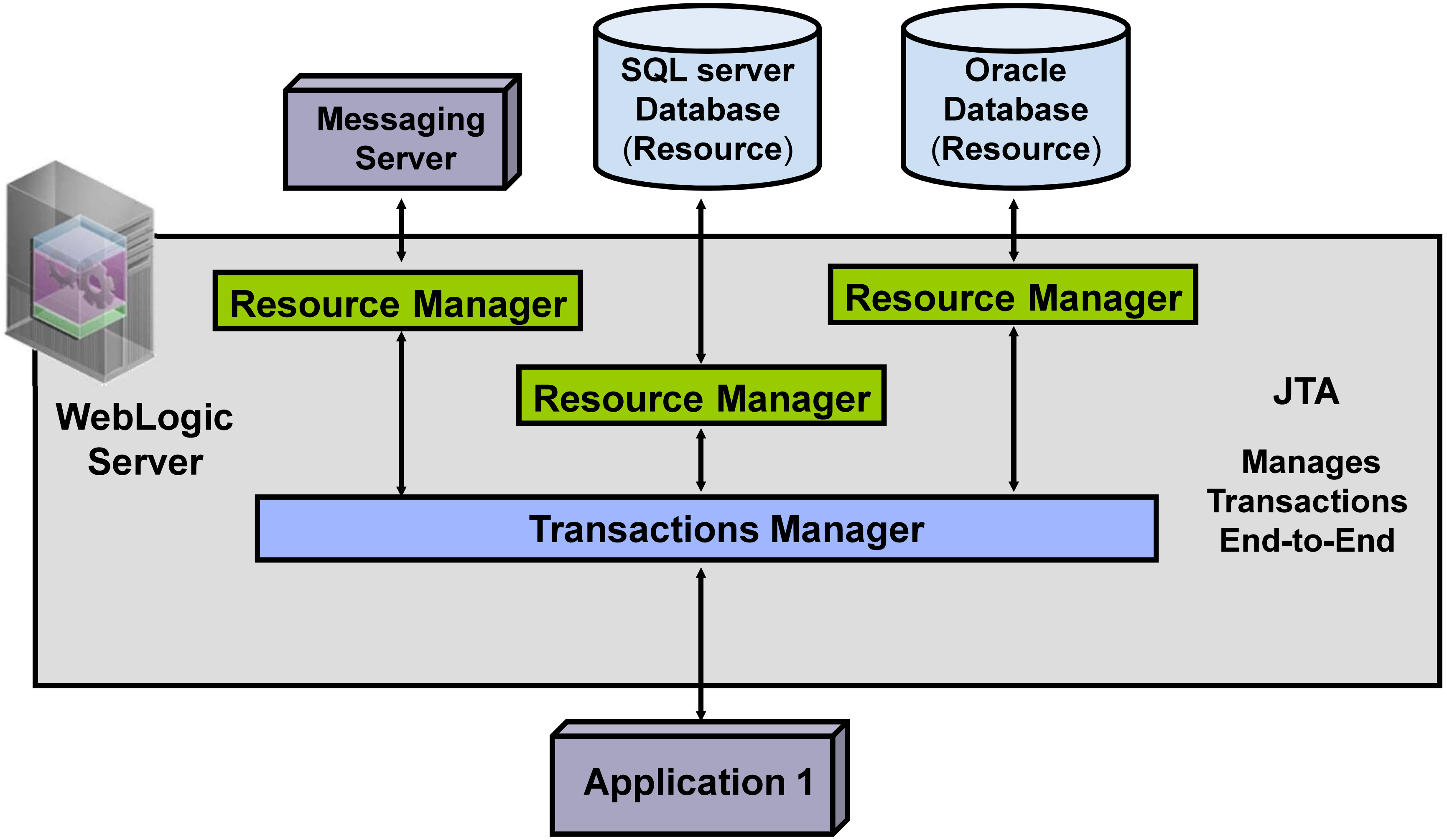
- Successful transfer (A)
- Unsuccessful transfer (accounts are left in an inconsistent state) (B)



Types of Transactions

- A *local transaction* deals with a single resource manager. It uses the non-Extended Architecture (non-XA) interface between WebLogic Server and resource managers.
- A *distributed transaction* coordinates or spans multiple resource managers.
- A *Global transaction* can deal with multiple resource managers. It uses the Extended Architecture (XA) interface between WebLogic Server and resource managers.
- You need to create non-XA or XA resources for local transactions. However, for global transactions, you need to create only XA resources.

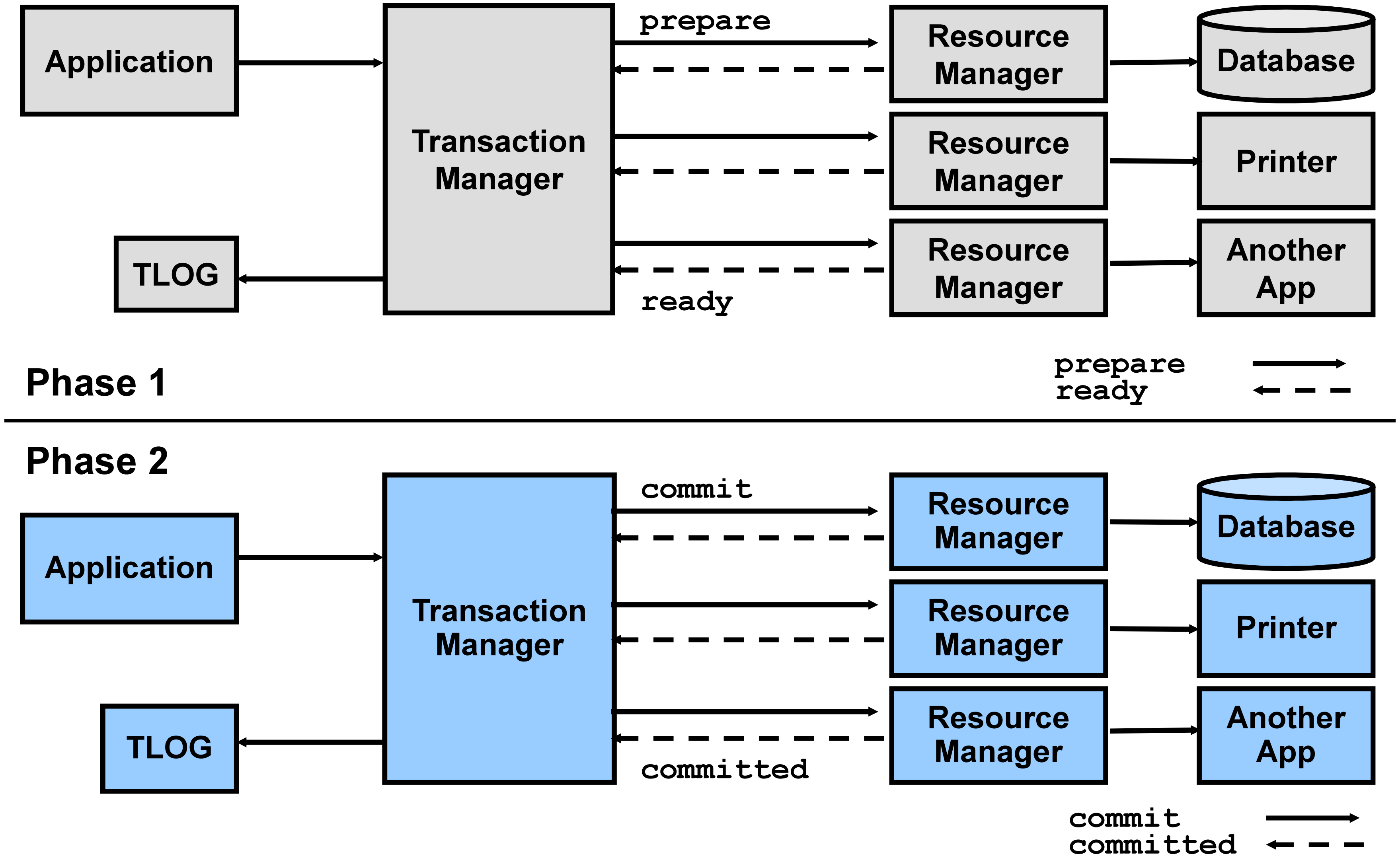
Transaction Management



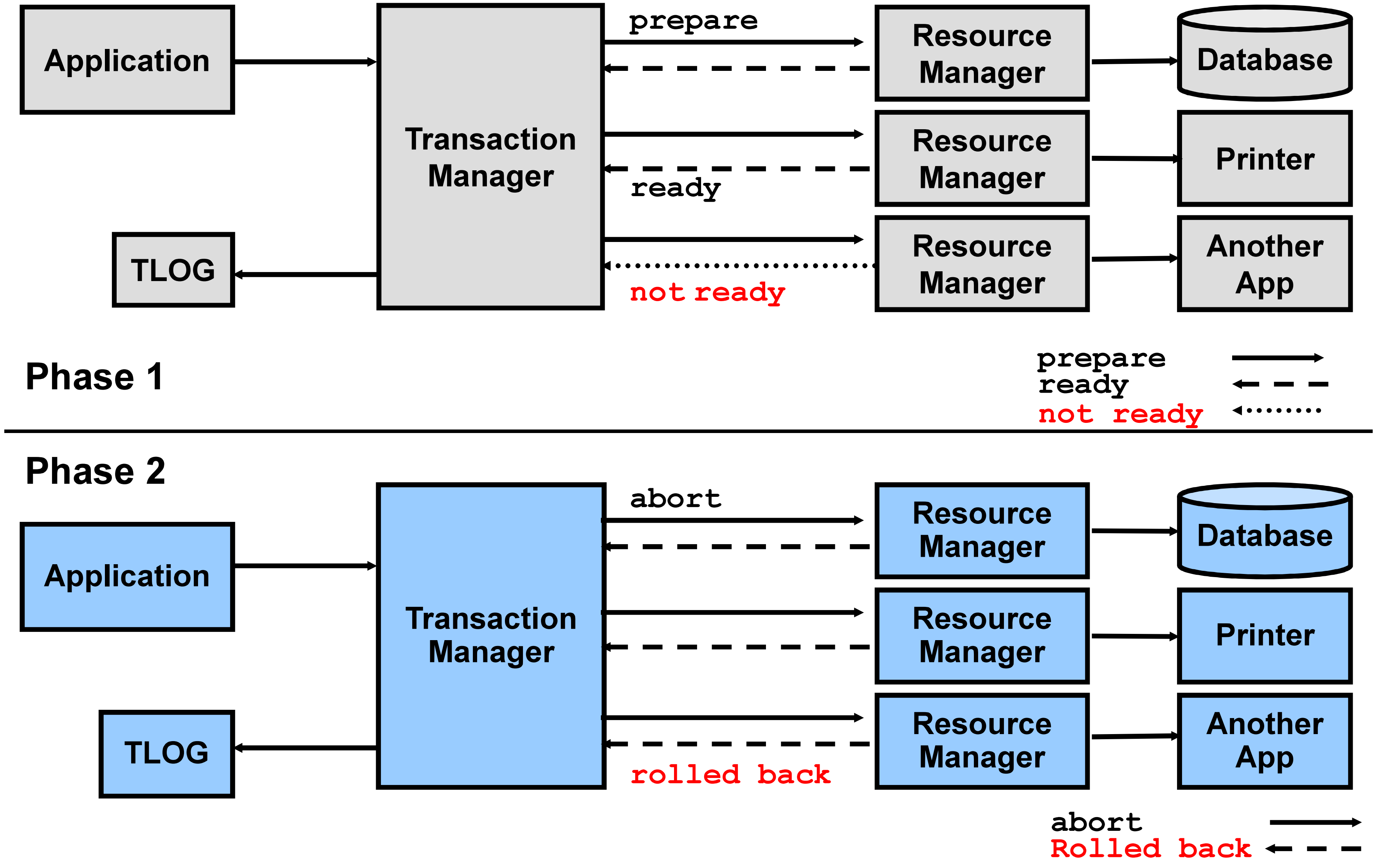
Two-Phase Commit Protocol

- The Two-Phase Commit (2PC) protocol uses two steps to commit changes within a distributed transaction.
  - Phase 1 asks RMs to prepare to make the changes
  - Phase 2 asks RMs to commit and make the changes permanent, or to roll back the entire transaction
- A global transaction ID (XID) is used to track all changes associated with a distributed transaction.

Successful Two-Phase Commit



Unsuccessful Two-Phase Commit



Quiz

The communication between the transaction manager and a specific resource manager is called:

1. A distributed transaction
2. A local transaction
3. A transaction branch

Java Transaction API (JTA)

- WLS uses JTA to implement and manage transactions.
- WLS JTA provides the following support:
  - Creates a unique transaction identifier (XID)
  - Supports an optional transaction name
  - Tracks objects involved in transactions
  - Notifies databases of transactions
  - Orchestrates 2PC using XA
  - Executes rollbacks
  - Executes automatic recovery procedures when failure
  - Manages time-outs

EJB Transaction Model

- Demarcating a transaction determines:
  - Who begins and ends a transaction
  - When each steps occurs
- A container-managed (declarative) transaction (CMT):
  - Is demarcated by the container at the method level
  - Is specified implicitly (by default) or declaratively through the use of annotations
- A bean-managed (explicit) transaction (BMT):
  - Is demarcated by the bean
  - Is specified programmatically in the bean through the JTA interface or the Java Database Connectivity (JDBC) interface

Managing Transactions with EJBs

- Bean-managed transactions:
  - Are performed programmatically using the `javax.transaction.UserTransaction` interface
  - Explicitly demarcate (start and end) transactions
  - Enable a transaction to span method calls
- Container-managed transactions:
  - Are specified declaratively using annotations or the XML deployment descriptor
  - Implicitly demarcate transaction boundaries at the start and end of each method call
  - Can use a session facade to enable a transaction to span multiple calls to the entities
- Manage queries and persistence of data by implementing Java Persistence API through `EntityManager`.

Managing Transactions with EJBs

- The bean provider is not exposed to the complexity of distributed transactions.
- The Java EE container provides a transaction infrastructure.
- EJBs do not support a nested transaction model.

Types of Transaction Management

- Container-managed transactions:
  - No transactional management code in the bean
  - Chosen implicitly by default or explicitly by use of the `@TransactionManagement` `(TransactionManagementType.CONTAINER)` annotation
  - Available to entities, session beans, and message-driven beans
- Bean-managed transactions:
  - Bean implementation must demarcate the begin, commit, or rollback for the transaction.
  - `@TransactionManagement` `(TransactionManagementType.BEAN)` annotation
  - Available only to session beans and MDBs

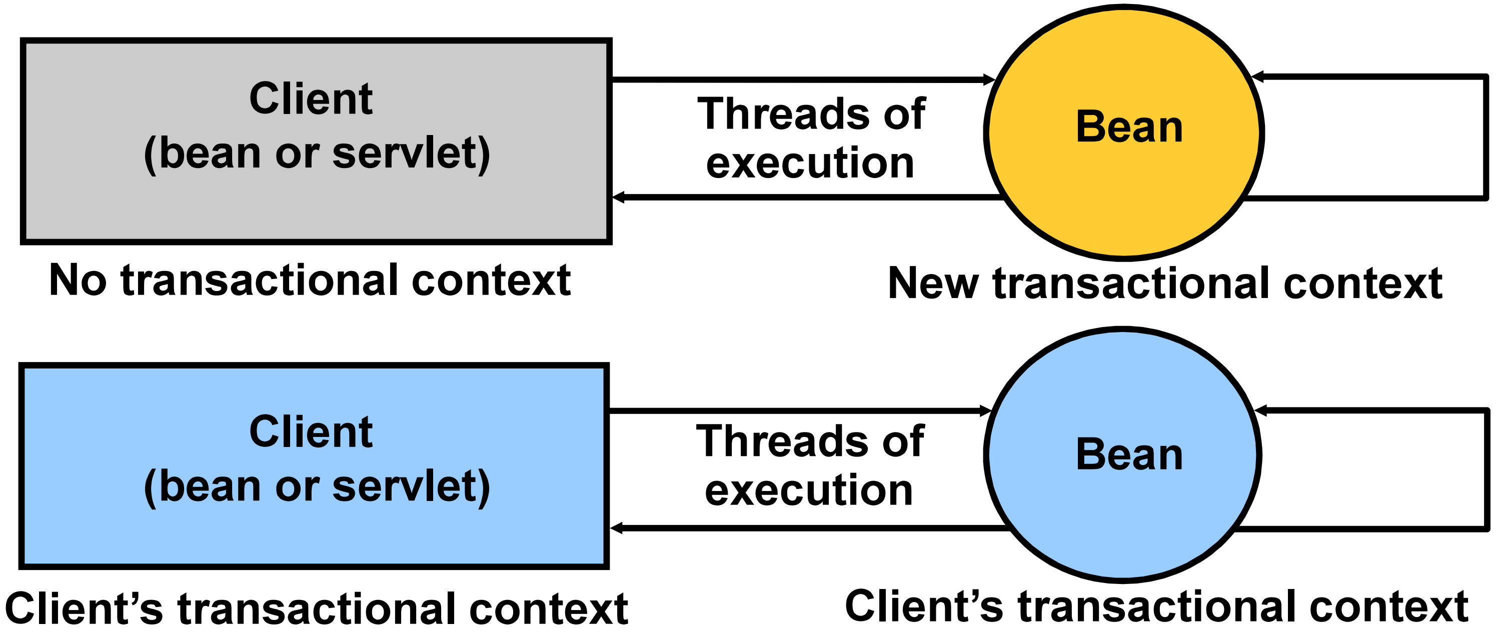
Container-Managed Transactions

- `@TransactionManagement(TransactionManagementType.CONTAINER)`
- Container-managed transactions can specify one of the following `@TransactionAttribute` annotations:
  - `REQUIRED` (default)
  - `SUPPORTS`
  - `MANDATORY`
  - `NEVER`
  - `REQUIRES_NEW`
  - `NOT_SUPPORTED`
- The transaction attribute can be specified at the:
  - Class level (where it applies to all business methods)
  - Method level (where it applies to a specific method)

Transaction Attribute: `REQUIRED`

A client has:

- No transaction: The bean starts a new one.
- A transaction: The bean uses it.



Client
(bean or servlet)

Threads of execution

Bean

No transactional context

New transactional context

Client
(bean or servlet)

Threads of execution

Bean

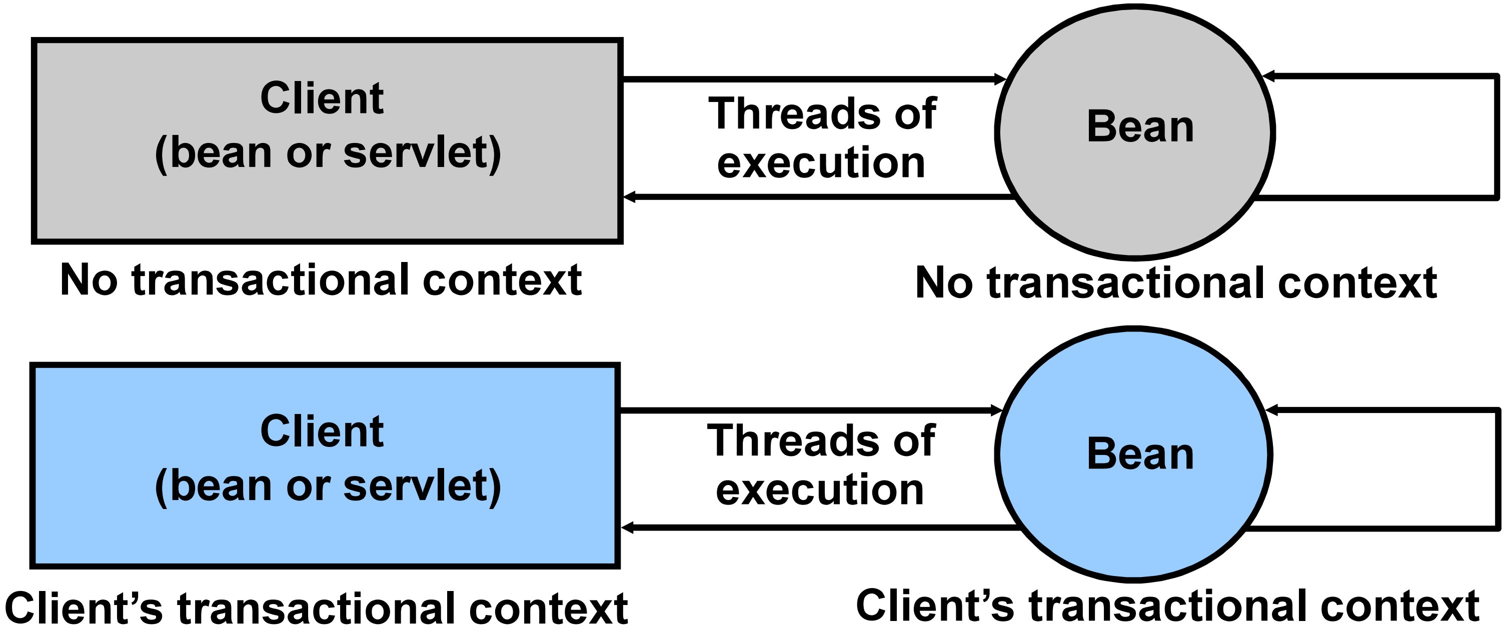
Client’s transactional context

Client’s transactional context

Transaction Attribute: SUPPORTS

A client has:

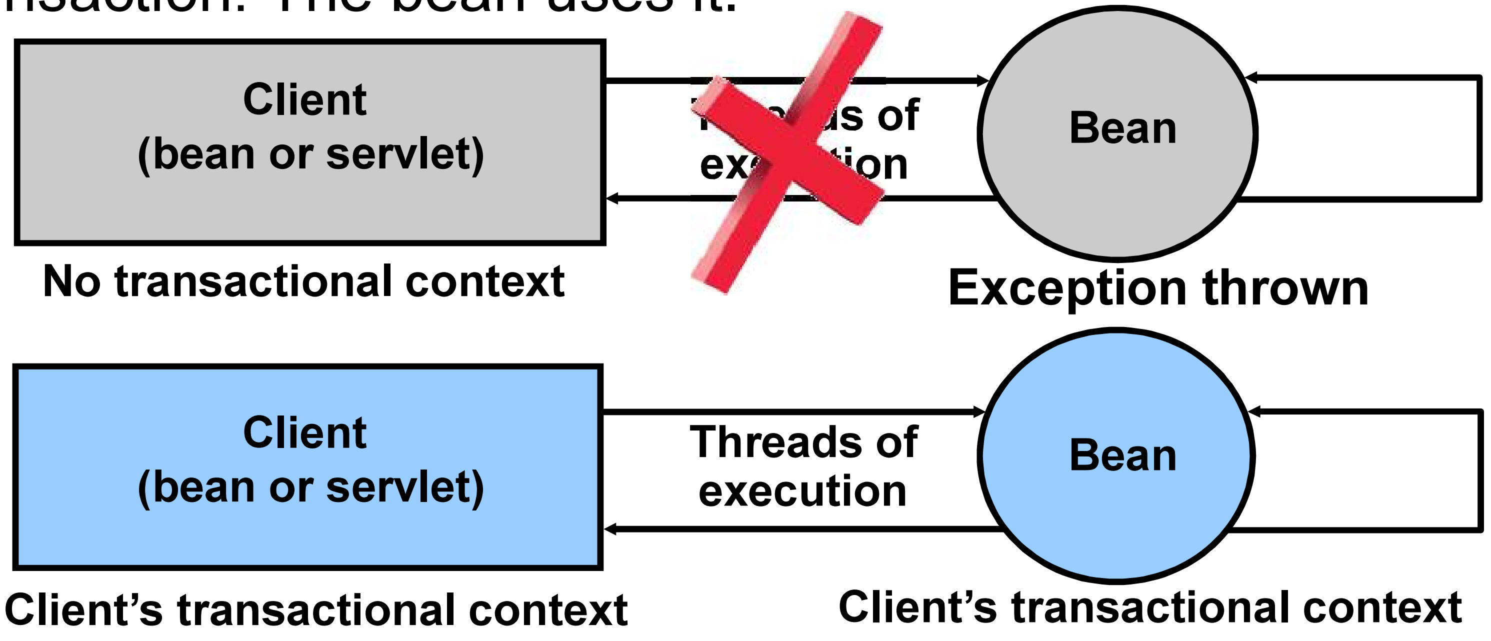
- No transaction: The bean does not start a new one.
- A transaction: The bean uses it.



Transaction Attribute: MANDATORY

A client has:

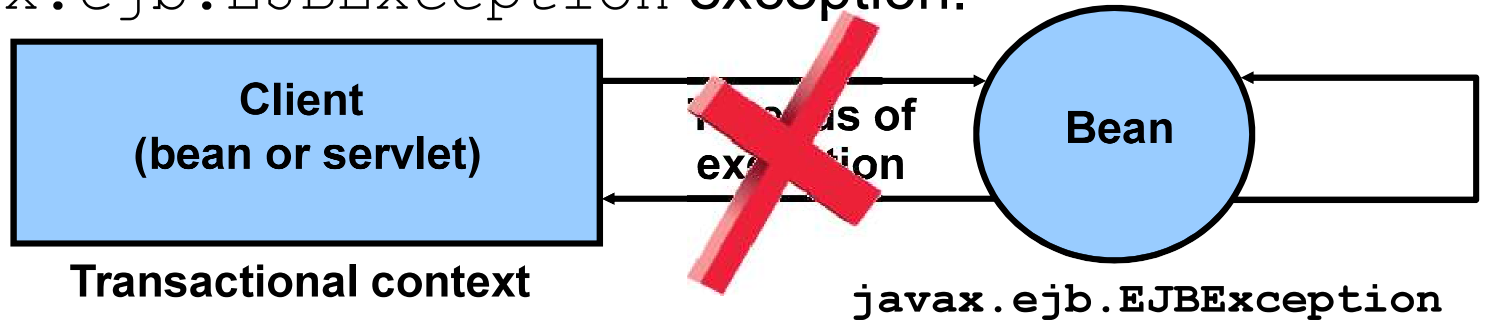
- No transaction: The bean throws the `javax.transaction.EJBTransactionRequiredException` exception.
- A transaction: The bean uses it.



Transaction Attribute: NEVER

A client has:

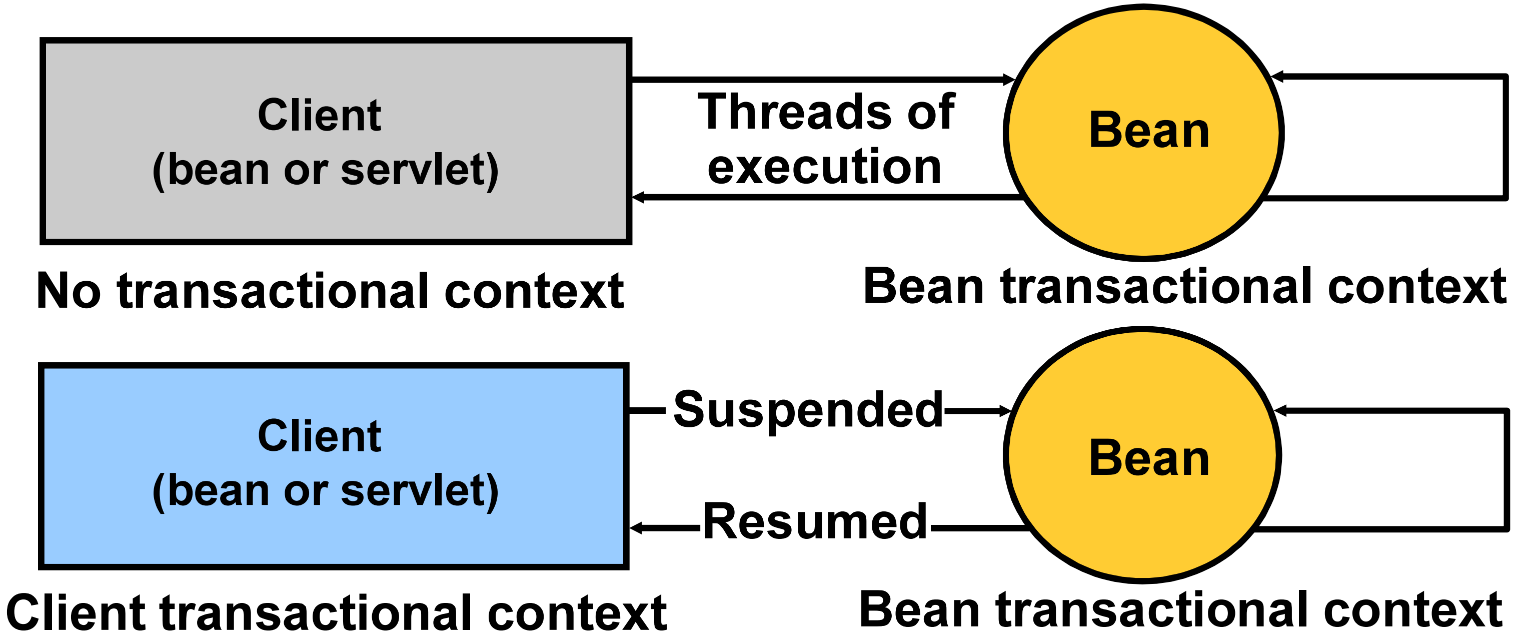
- No transaction: The container calls the method with no transactional context.
- A transaction: The container throws the `javax.ejb.EJBException` exception.



Transaction Attribute: REQUIRES_NEW

A client has:

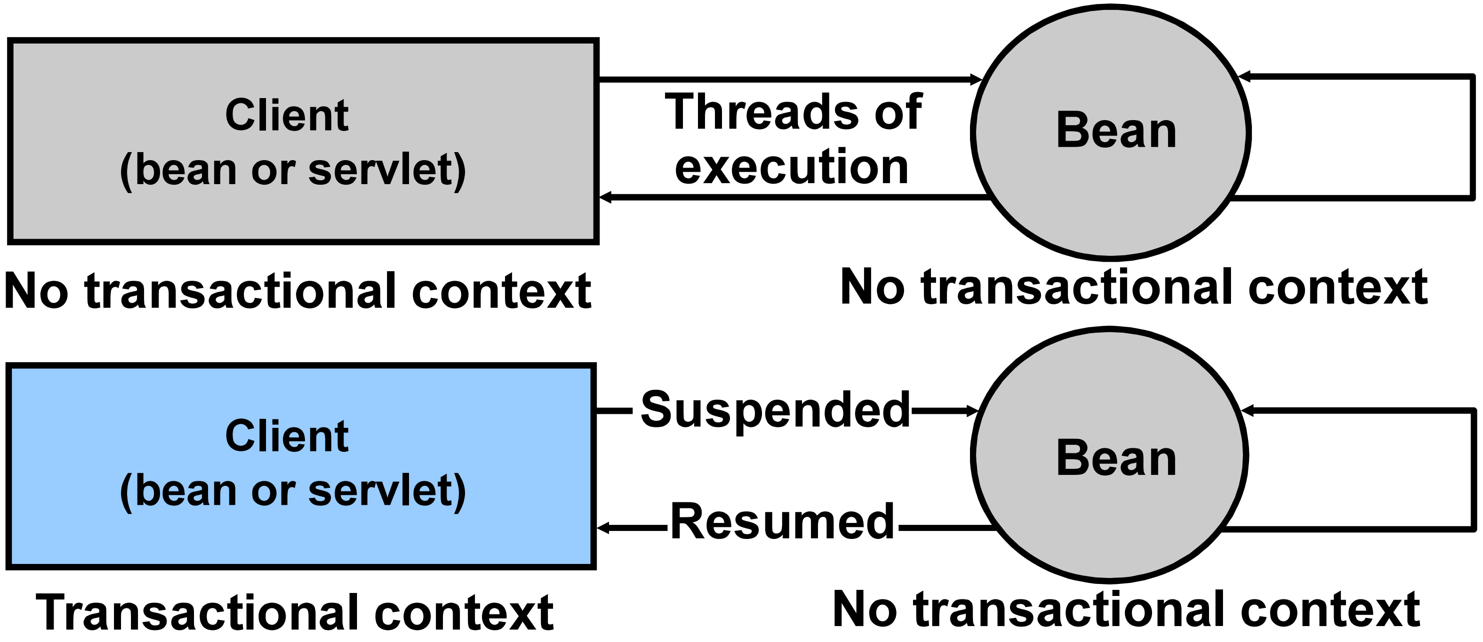
- No transaction: The bean starts a new one.
- A transaction: It is suspended; the bean starts a new one and commits it, and then reassociates the old one.



Transaction Attribute: NOT_SUPPORTED

A client has:

- No transaction: The bean does not start a new one.
- A transaction: The bean suspends it. The transaction resumes when the client gains control.



Quiz

The transaction demarcation in EJB 3.0 is implemented through the use of the `@TransactionManagement` annotation. If this annotation is not present, the container defaults to bean-managed transaction (BMT).

1. True
2. False

CMT: `setRollbackOnly()`

- The `setRollbackOnly()` method can control the transaction state in the bean for a CMT.
- The `setRollbackOnly()` method marks the current transaction to rollback.
- If a transaction is marked for rollback, the container rolls back the transaction before returning to the caller.

Container-Managed Transaction: Example

```
@TransactionManagement(TransactionManagementType.CONTAINER)
@TransactionAttribute(TransactionAttributeType.REQUIRED)
@Stateful public CartBean implements Cart {

  public void initialize() {
     ...
  }

  @TransactionAttribute(TransactionAttributeType.MANDATORY)
  public void setTaxRate(float taxRate) {
     ...
  }

  @TransactionAttribute(TransactionAttributeType.MANDATORY)
  public void addItem(float taxRate) {
     ...
  }
...
```

1

2

Java Transaction API (JTA)

Java Transaction API (JTA) is:

- Java EE standard for implementing transactions
- Interface between the transaction manager and the components involved in the transaction (for example, beans, EJB containers, resource managers, and so on)
  - The JTA package provides an application interface for managing transactions (`UserTransaction`).
- Used for:
  - Enlisting resources: Single-phase or two-phase commit
  - Demarcating transactions

JTA: UserTransaction Interface

- Allows applications to explicitly manage transaction boundaries
- Encapsulates most of the functionality of a transaction manager

```
public interface javax.transaction.UserTransaction{
  public abstract void begin ();
  public abstract void commit ();
  public abstract int getStatus ();
  public abstract void rollback ();
  public abstract void setRollbackOnly ();
  public abstract void setTransactionTimeout(int secs);
}
```

Bean-Managed Transactions

- Are declared using `@TransactionManagement(TransactionManagementType.BEAN)`
- Are demarcated and managed by using the JTA `UserTransaction` interface to:
  - Initialize a transaction context by calling the `begin()` method
  - Terminate a transaction context by calling the `commit()`, or the `rollback()` method

Bean-Managed Transaction: Example

```
@TransactionManagement(TransactionManagementType.BEAN) 1
@Stateful
public CartBean implements Cart {
@Resource
private UserTransaction utx;                2

public void setTaxRate(float taxRate)  {
    utx.begin();  3
    try {

      ...         4

      utx.commit();          5
    } catch  (Exception ex)  {
      utx.rollback();        5
      ex.printStackTrace();
    }
...
```

BMT Demarcation: Restrictions

- Session beans and message-driven beans can have bean-managed transactions only if they specify the use of BMT.
- An instance that starts a transaction must complete the transaction before it starts a new transaction.
- A stateful session bean can commit a transaction before a business method ends.
- A stateless session bean must commit the transaction before the business method returns.
- A message-driven bean must commit the transaction before the `onMessage()` method returns.

Enlisting Database Resources

- The process of including SQL updates in a transaction is called "enlisting."
- JTA automatically enlists databases opened with a `DataSource` object in a global `UserTransaction` object.
- Starting from JDK 1.2, a `DataSource` published into a JNDI namespace is one way to make connections. Connections can also be made implicitly through an EJB 3.0 `EntityManager` instance.
- If your global transaction involves more than one database, you must configure a two-phase commit engine.

Summary

In this lesson, you should have learned how to:

- Choose between container-managed and bean-managed transactions
- Set the transaction attribute for container-managed transactions
- Create transaction demarcations

Practice: Overview

These practices cover the following topics:

- Adding a new product record by using explicit default container-managed persistent attributes
- Adding a new product by using bean-managed persistent techniques in a stateful session bean