



Distributed Transactions Overview



P. O. Box 80049
Austin, TX 78708
Fax: +1 (801) 383-6152

Outline

At the end of this presentation, you will understand:

- How to define a transaction
- How transactions are coordinated across multiple data sources
- How Enterprise Java Beans use distributed transactions
- The different transaction policies, or attributes, used in EJB
- The different transaction isolation levels used by EJB data sources

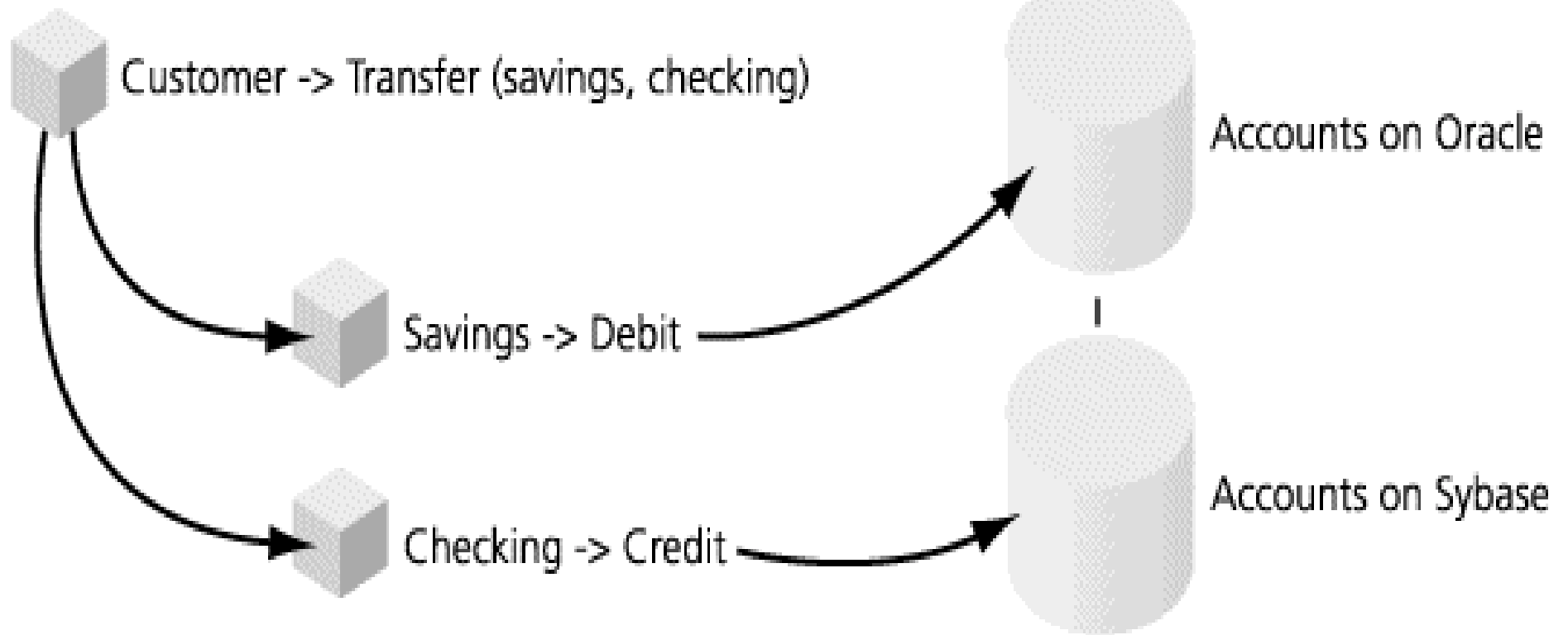
Transactions and ACID

- ▼ A Transaction is a unit of work that has **ACID** properties:
 - **Atomicity:** Changes are either all committed or rolled back
 - **Consistency:** Changes to the state of the resource move from one valid state to another
 - **Isolation:** Changes to the shared resources do not become visible to other clients until the transaction commits
 - **Durability:** Changes to the shared resources survive subsequent system or media failures

Where are we?

- How to define a transaction
- *How transactions are coordinated across multiple data sources*
- How Enterprise Java Beans use distributed transactions
- The different transaction policies, or attributes, used by Enterprise Java Beans
- The different transaction isolation levels used by Enterprise Java Bean data sources

An Example Transaction



A single transaction

What is Two Phase Commit (2PC)?

- ▼ When a transaction manager commits a transaction it is performed in two phases:
 - Phase I
 - Transaction manager (TM) issues prepare requests to each participating resource manager (RM) to query their ability to commit their work
 - A positive reply from an RM means that it is prepared
 - A negative reply initiates a rollback operation by the TM
 - Phase II
 - The TM directs all participating RMs to commit their work on behalf of the global transaction if phase I was successful
 - Otherwise, the TM issues a rollback directive to all participating RMs
- ▼ After phase II completes, TM conveys the outcome to the client

What is the X/Open DTP Model (XA)?

- ▼ X/Open Distributed Transaction Processing (DTP) model is a software architecture that allows:
 - Multiple application programs to share resources provided by multiple resource managers
 - Their work to be coordinated into global transactions
- ▼ Most RDBMS's support the XA protocol

Benefits of Transactions: High Availability and Recovery

Transactions greatly simplify server recovery, thereby enabling availability

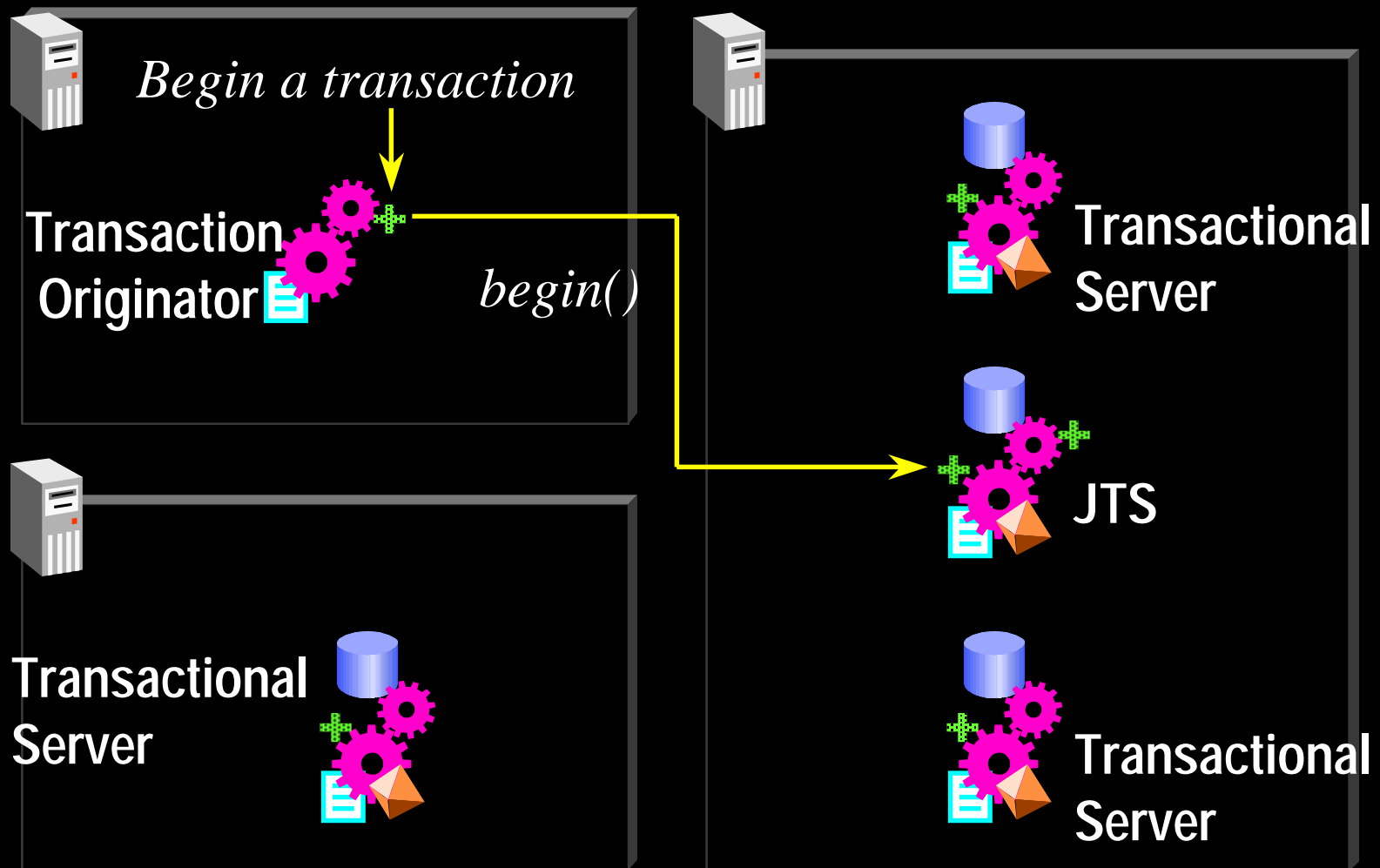
- ▼ Should a server fail, when it recovers:
 - Consistency ensures state from transactions active during failure is ignored
 - Durability ensures state from successfully completed transactions is maintained
- ▼ In other words, all work not check-pointed by a successful commit is discarded under error conditions
- ▼ Consequently, client code can be much simpler
- ▼ Highly available server code can also be simpler, if a service is managing the transactional resources

Java Transaction Service (JTS)

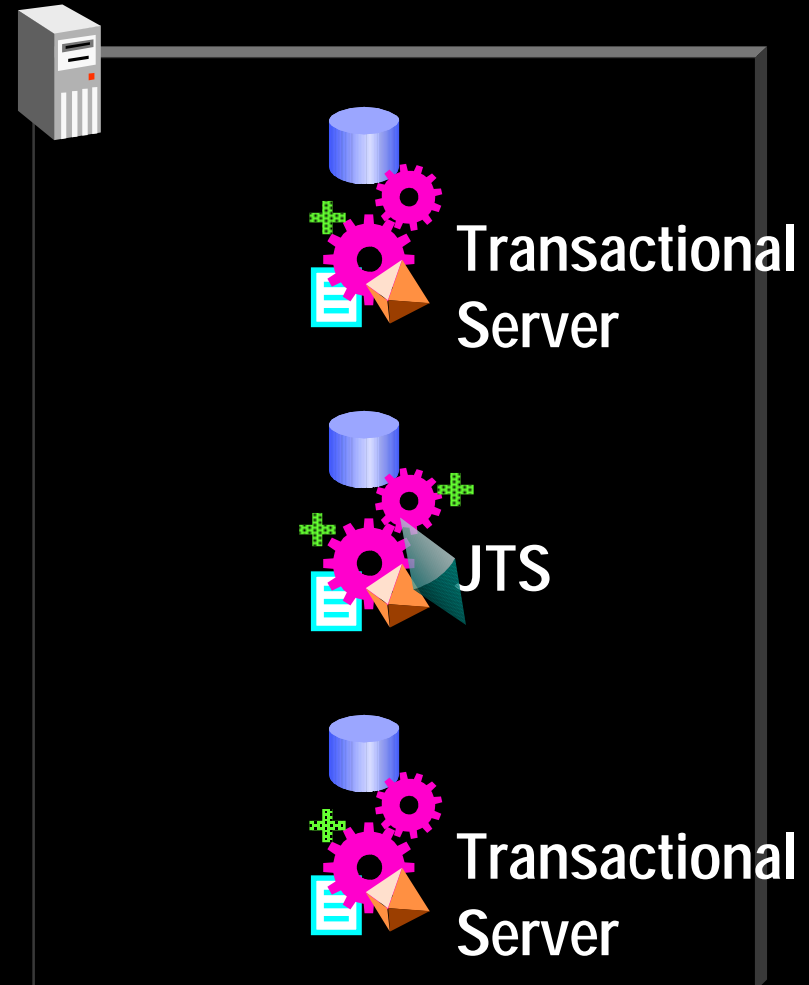
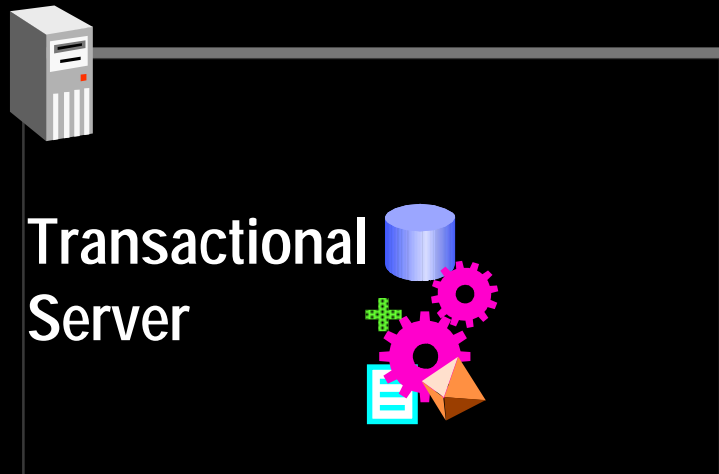
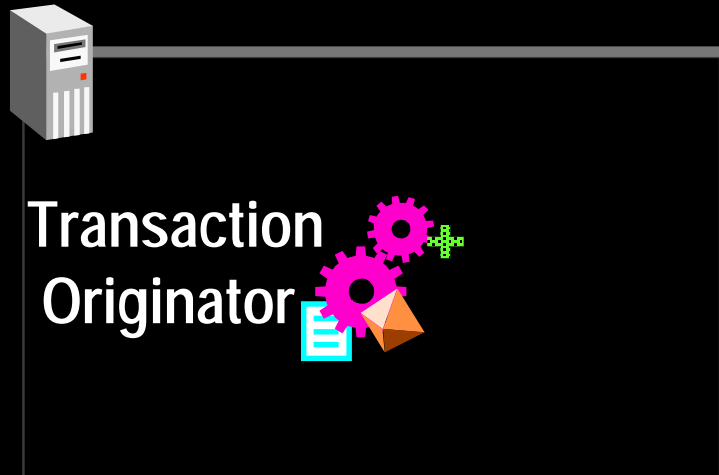
▼ The Java Transaction Service (JTS):

- Allows multiple invocations on multiple distributed objects to be treated as one unit of work
- Supports industry-standard, 2-phase-commit protocols
- Specifies the relationship between a transaction manager and its participating resources
- Also specifies the responsibilities of these components throughout the critical completion phases of a transaction
- Can be built upon older TP monitors such as Tuxedo, Encina, etc.
- Applies transactional semantics to the distributed object world

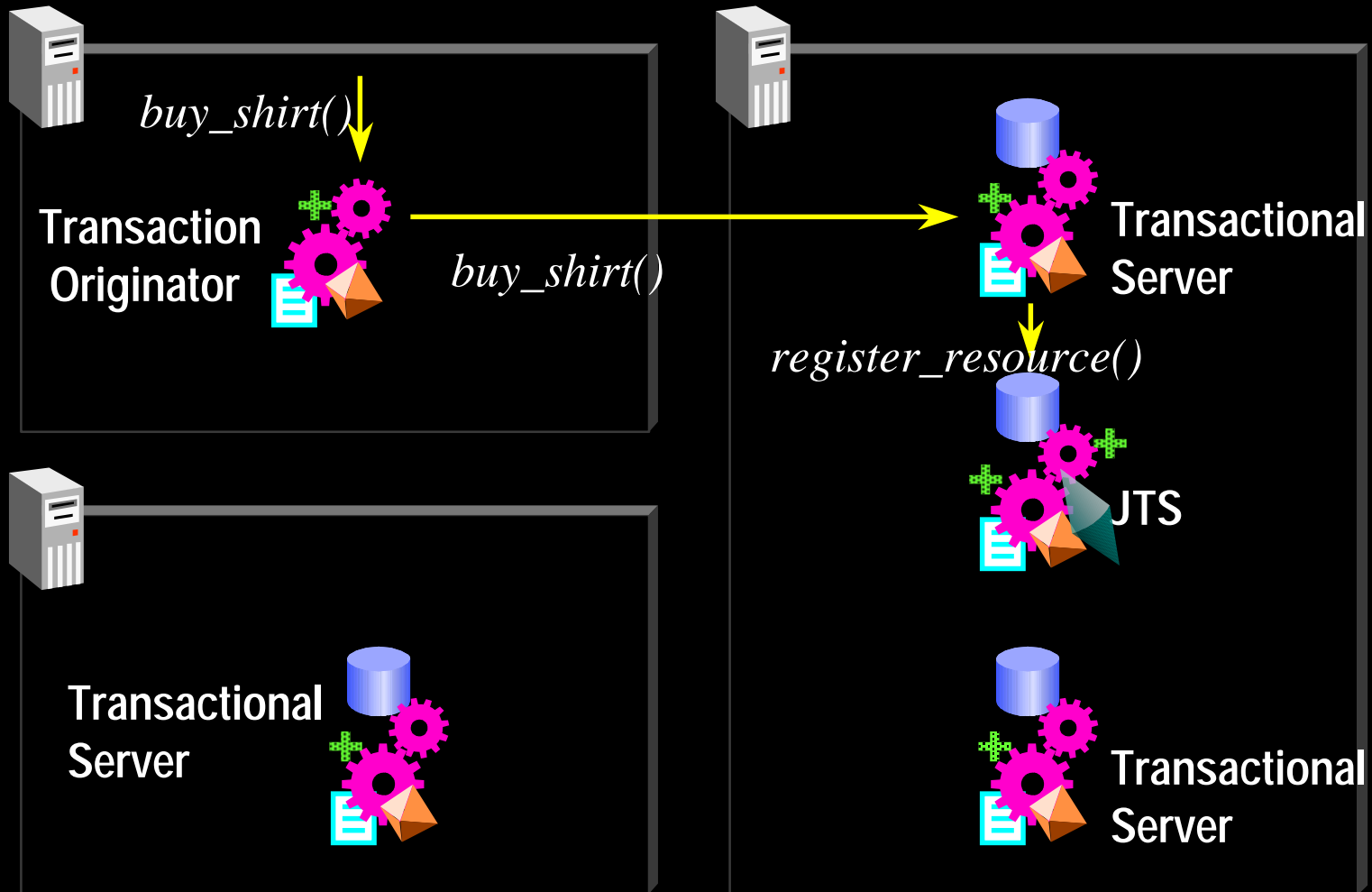
Scenario: JTS In Use



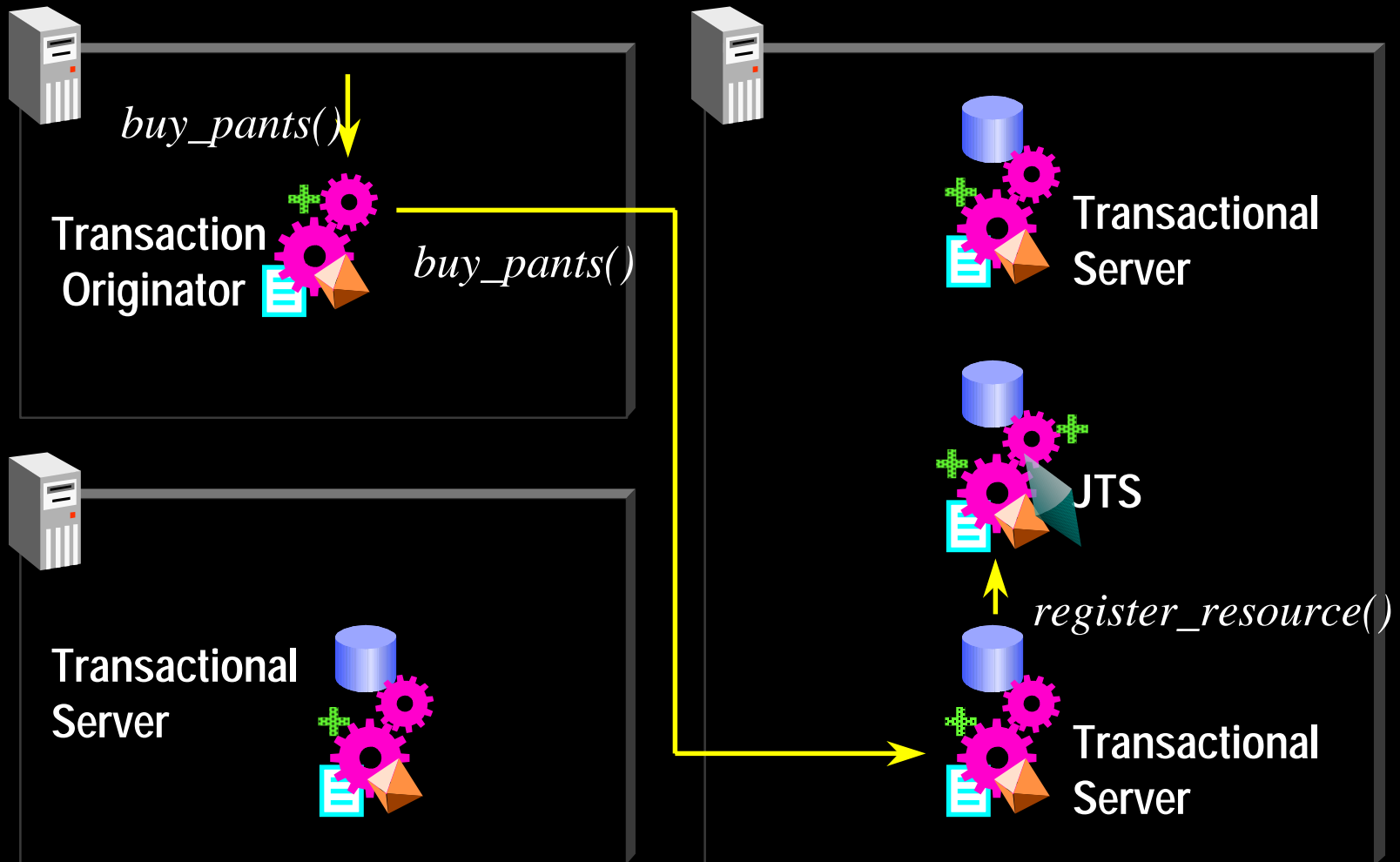
Transaction Context Created



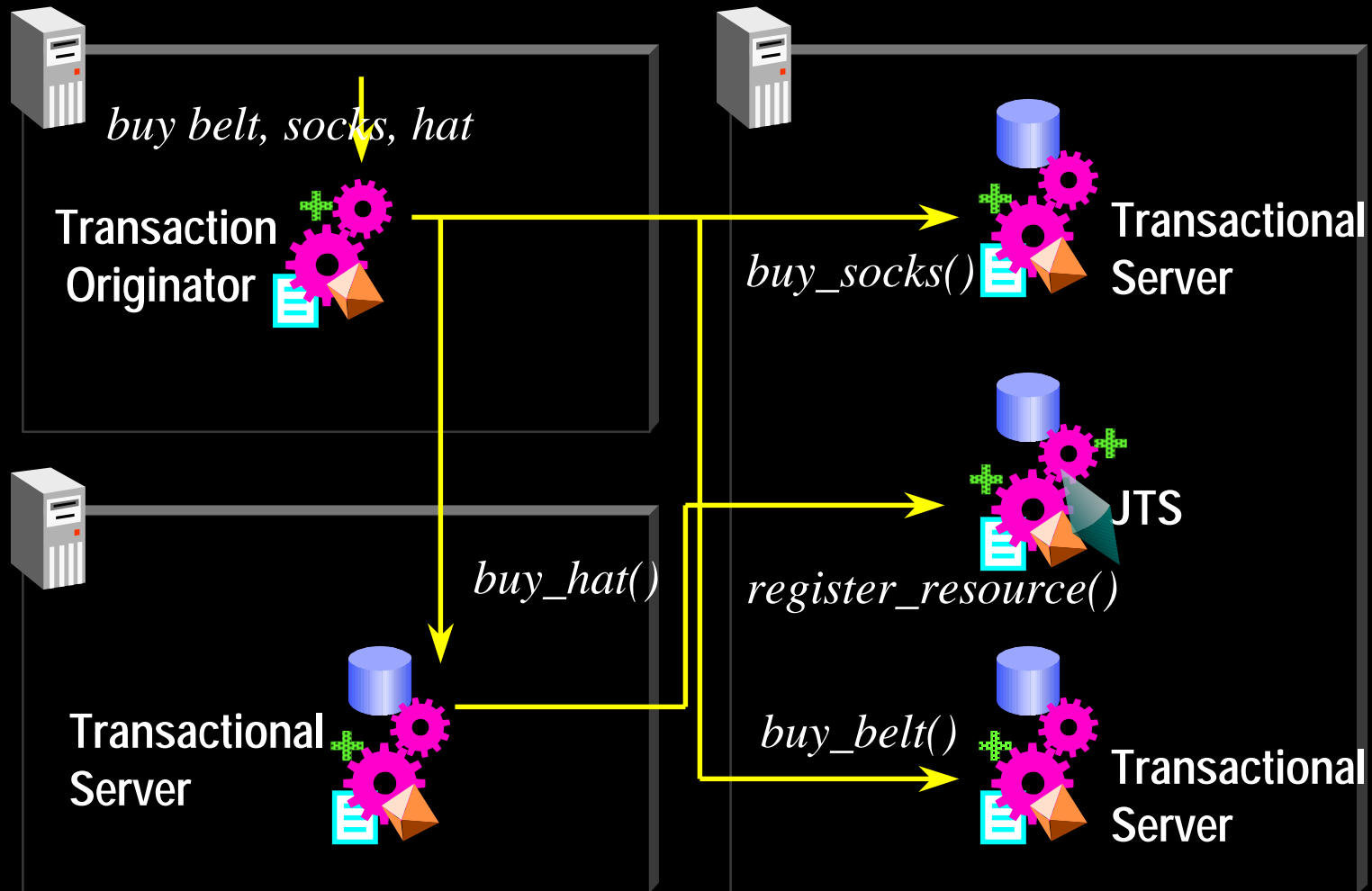
Transactional Operation



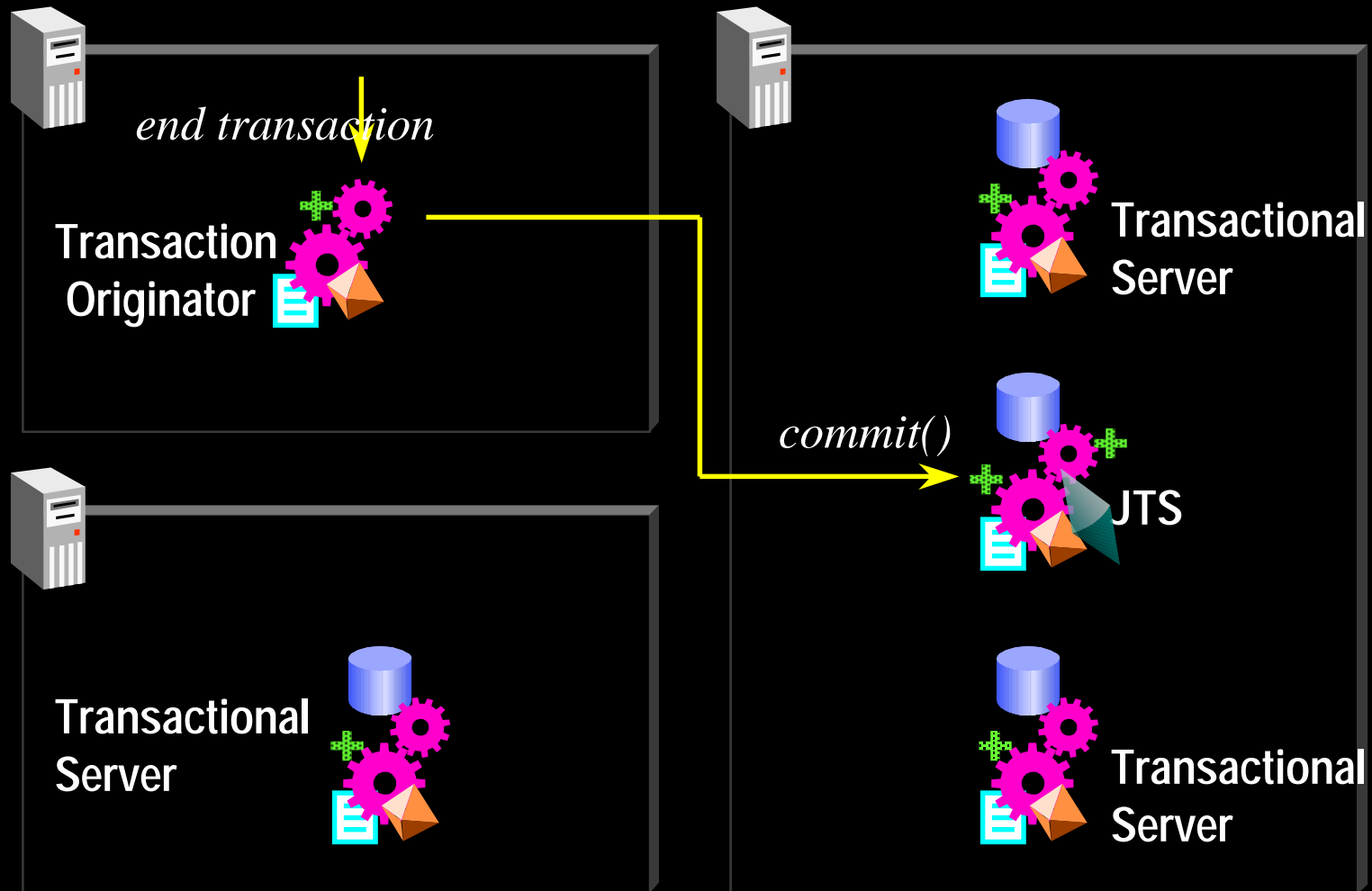
Transactional Operation



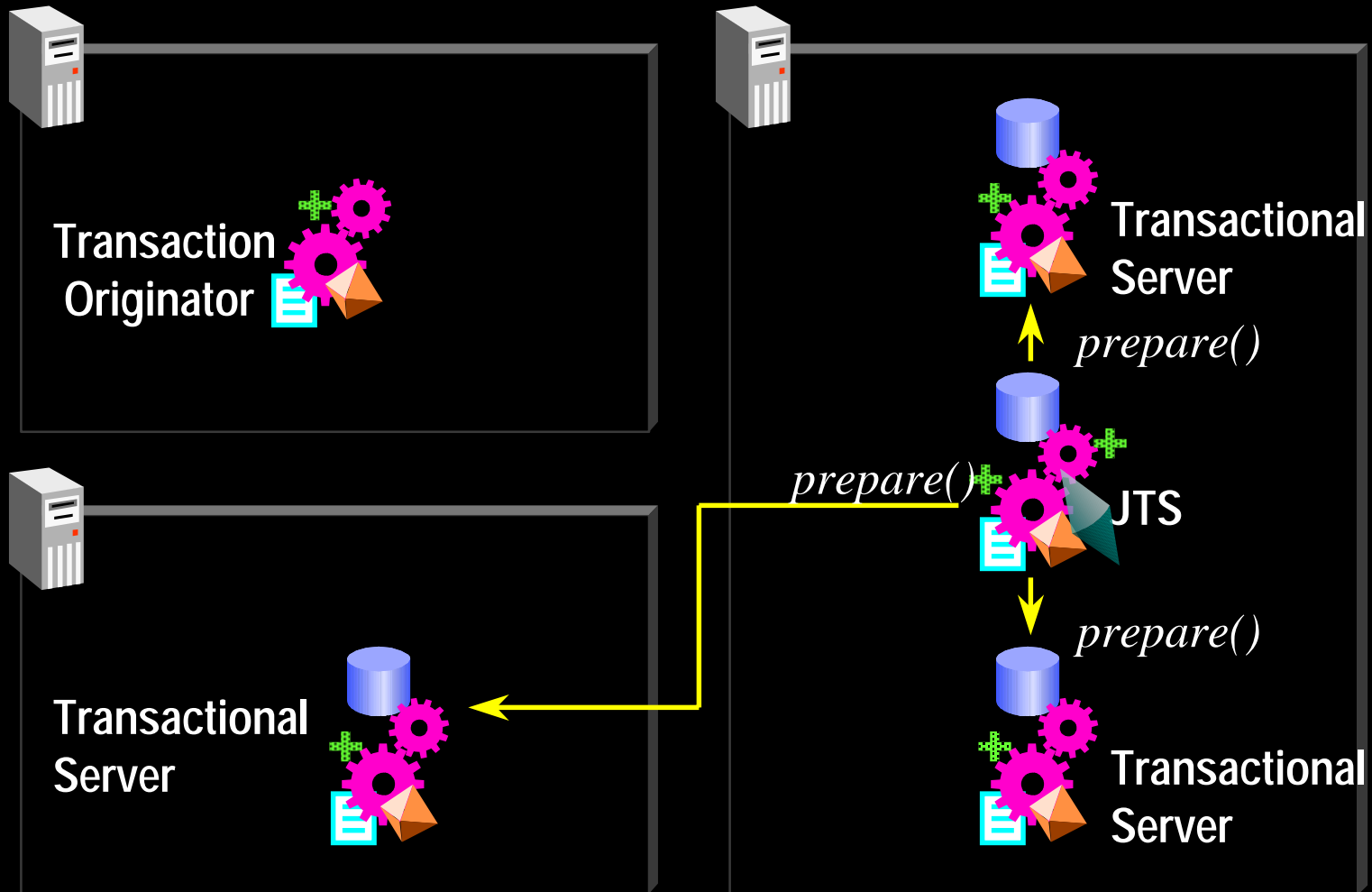
More Transactional Operations



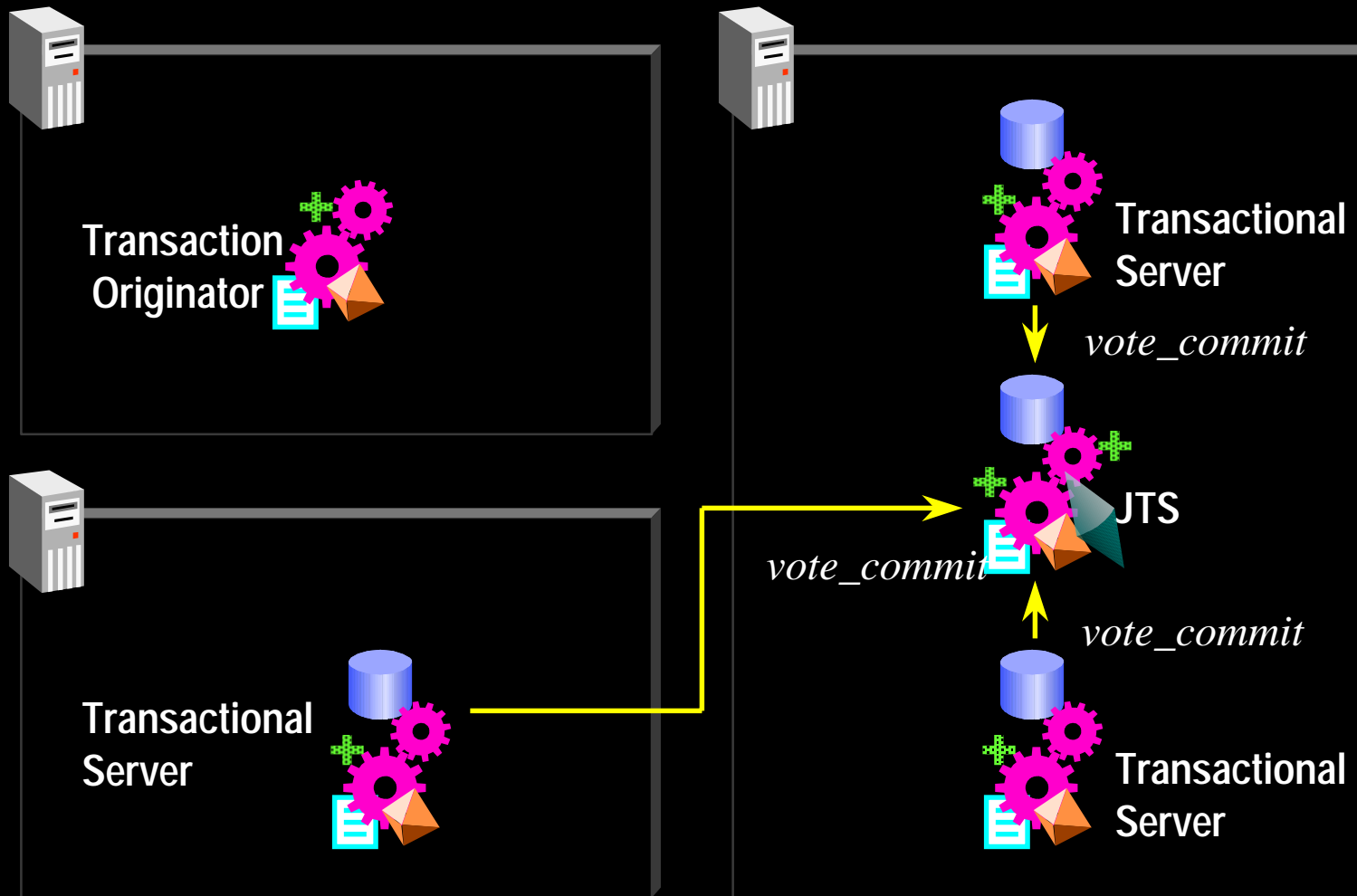
End Transaction



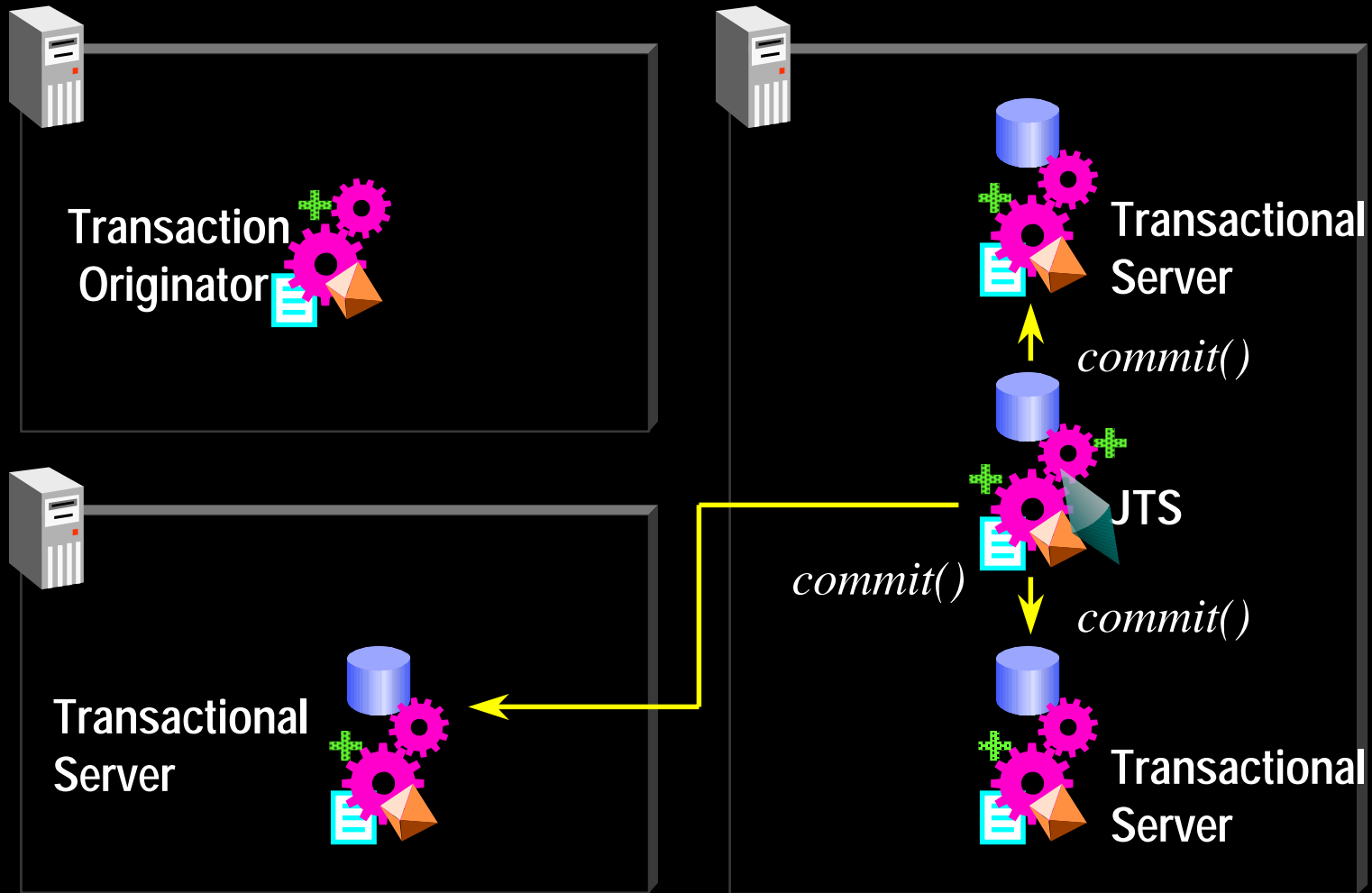
Completion Phase 1



Responses to Phase 1



Completion Phase 2



Where are we?

- How to define a transaction
- How transactions are coordinated across multiple data sources
- *How Enterprise Java Beans use distributed transactions*
- The different transaction policies, or attributes, used by Enterprise Java Beans
- The different transaction isolation levels used by Enterprise Java Bean data sources

Enterprise Java Beans and Distributed Transactions

EJB depends on JTS for managing distributed transactions

- ▼ EJB clients can explicitly originate and complete transactions
 - i.e. demarcation code exists in the client at compile time
- ▼ A more flexible alternative is EJB container-managed transactions:
 - Containers implicitly originate and complete transactions
 - Transactional policies may be configured declaratively at deployment time in the deployment descriptor

Where are we?

- How to define a transaction
- How transactions are coordinated across multiple data sources
- How Enterprise Java Beans use distributed transactions
- *The different transaction policies, or attributes, used by Enterprise Java Beans*
- The different transaction isolation levels used by Enterprise Java Bean data sources

EJB Container Transaction Policies



NOT SUPPORTED

- Upon entering the method, any current transaction is suspended for the duration of the invocation

REQUIRED

- If there is no transaction active, then one will be started upon entering the method
- The transaction will be committed before leaving the method if the container started it

SUPPORTS

- If a transaction is active, work is done in the context of that transaction
- No new transactions are started

EJB Container Transaction Policies

...continued



REQUIRES NEW

- A new transaction will be started upon entering the method, even if there is a transaction active
- The transaction will be committed before leaving the method

MANDATORY

- Upon entering the method, an exception is thrown if there is no transaction active

NEVER

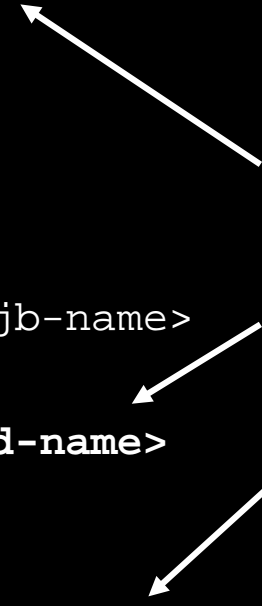
- This bean can never be involved in a transaction, an exception will be thrown if a transaction is active

BEAN MANAGED

- A Session Bean performs the transaction control itself. This option is NOT available for Entity Beans

Setting the Container Managed Transaction Policy

```
// File: ejb-jar.xml
...
<ejb-jar>
  <enterprise-beans>
    <session>
      ...
      <transaction-type>Container</transaction-type>
    </session>
  </enterprise-beans>
  ...
  <container-transaction>
    <method>
      <ejb-name>lab08.ShoppingCartHome</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>performPurchase</method-name>
      <method-params></method-params>
    </method>
    <trans-attribute>Required</trans-attribute>
  </container-transaction>
```



Setting the Container Managed Transaction Policy

```
// File: ejb-jar.xml
...
<ejb-jar>
  <enterprise-beans>
    <session>
      ...
      <transaction-type>Container</transaction-type>
    </session>
  </enterprise-beans>
  ...
  <container-transaction>
    <method>
      <ejb-name>lab08.ShoppingCartHome</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>performPurchase</method-name>
      <method-params></method-params>
    </method>
    <trans-attribute>Required</trans-attribute>
  </container-transaction>
```

Setting the Container Managed Transaction Policy

```
// File: ejb-jar.xml
...
<ejb-jar>
  <enterprise-beans>
    <session>
      ...
      <transaction-type>Container</transaction-type>
    </session>
  </enterprise-beans>
  ...
  <container-transaction>
    <method>
      <ejb-name>lab08.ShoppingCartHome</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>performPurchase</method-name>
      <method-params></method-params>
    </method>
    <trans-attribute>Required</trans-attribute>
  </container-transaction>
```

When the method performPurchase is called, the container will ensure the method will be invoked within a transaction.

Setting the Container Managed Transaction Policy ...*cont'd*

File: *ejb-jar.xml* ...continued

```
<container-transaction>
```

```
  <method>
```

```
    <ejb-name>lab08.ShoppingCartHome</ejb-name>
```

```
    <method-intf>Remote</method-intf>
```

```
    <method-name>*</method-name>
```

```
  </method>
```

```
  <trans-attribute>NotSupported</trans-attribute>
```

```
</container-transaction>
```

```
</assembly-descriptor>
```

```
</ejb-jar>
```

All methods will have the "NotSupported" transaction attribute applied to them.

Setting the Container Managed Transaction Policy ...*cont'd*

File: *ejb-jar.xml* ...continued

```
<container-transaction>
```

```
  <method>
```

```
    <ejb-name>lab08.ShoppingCartHome</ejb-name>
```

```
    <method-intf>Remote</method-intf>
```

```
    <method-name>*</method-name>
```

```
  </method>
```

```
  <trans-attribute>NotSupported</trans-attribute>
```

```
</container-transaction>
```

```
</assembly-descriptor>
```

```
</ejb-jar>
```

All methods will have the "NotSupported" transaction attribute applied to them.

Explicitly Starting Transactions

```
//File: clientTransaction.java

import javax.naming.InitialContext;
import javax.transaction.UserTransaction;

public class clientTransaction {
    public static void main (String [] argv){
        UserTransaction ut =null;
        InitialContext ctx =new InitialContext();
        ut
= (UserTransaction)ctx.lookup("java:comp/UserTransaction ");
        ut.begin();
        //do some transaction work
        ut.commit(); //or ut.rollback();
    }
}
```

javax.transaction.UserTransaction



```
public interface javax.transaction.UserTransaction
{
    public abstract void begin();
    public abstract void commit();
    public abstract int getStatus();

    //rolls back a transaction
    public abstract void rollback();

    //flags a transaction to be rolled back
    public abstract void setRollbackOnly();
    public abstract void setTransactionTimeout(int);
}

//Note that exceptions have been removed for clarity
```

Where are we?

- How to define a transaction
- How transactions are coordinated across multiple data sources
- How Enterprise Java Beans use distributed transactions
- The different transaction policies, or attributes, used by Enterprise Java Beans
- *The different transaction isolation levels used by Enterprise Java Bean data sources*

Transaction Isolation Concepts

There exist three types of transaction violations

- Dirty reads
- Non-repeatable reads
- Phantom reads

Transaction Isolation Levels

When deploying an EJB, you control the extent of transaction violations permitted by that beans' DataSource using the following isolation levels:

isolation levels:	Dirty reads permitted?	Non-repeatable reads permitted?	Phantom reads permitted?
Transaction Read Uncommitted	<i>yes</i>	<i>yes</i>	<i>yes</i>
Transaction Read Committed	<i>no</i>	<i>yes</i>	<i>yes</i>
Transaction Repeatable Read	<i>no</i>	<i>no</i>	<i>yes</i>
Transaction Serializable Read (most stringent)	<i>no</i>	<i>no</i>	<i>no</i>

Setting the Isolation Level

```
// File: CMP_Weblogic_CMP_RDBMS959981339550.xml
```

```
...
```

```
<weblogic-rdbms-bean>
```

```
...
```

```
<attribute-map>
```

```
...
```

```
</attribute-map>
```

```
<finder-list>
```

```
...
```

```
</finder-list>
```

```
<options>
```

```
...
```

```
  <transaction-  
isolation>TRANSACTION_SERIALIZABLE</transaction-isolation>
```

```
</options>
```

```
</weblogic-rdbms-bean>
```

The transaction isolation level is defined for the connection used by the EJB to communicate with the DB.



Setting the Isolation Level

```
// File: CMP_Weblogic_CMP_RDBMS959981339550.xml
```

```
...
```

```
<weblogic-rdbms-bean>
```

```
...
```

```
<attribute-map>
```

```
...
```

```
</attribute-map>
```

```
<finder-list>
```

```
...
```

```
</finder-list>
```

```
<options>
```

```
...
```

```
<transaction-  
isolation>TRANSACTION_SERIALIZABLE</transaction-isolation>
```

```
</options>
```

```
</weblogic-rdbms-bean>
```

The transaction isolation level is defined for the connection used by the EJB to communicate with the DB.

Do You Need a Distributed Transaction Service?

- ▼ Just because your application has transactions does not mean you need support for distributed transactions
- ▼ Because enterprise software architects have become accustomed to not having the luxury of distributed transactions, many have learned to architect around the need
 - Most enterprises do not use TP monitors
 - 90% of enterprises that use TP monitors use them as middleware, not distributed transaction coordinators, often due to the large performance penalty

You Need Distributed Transactions When:

- ▼ Your business objects use multiple or different kinds of databases and the transaction must span more than one db
- ▼ You have a single db, but your business objects are distributed
- ▼ You want to involve legacy systems, other transactional middleware, or queuing systems in your transactions
- ▼ You have none of the above needs right now, but want to allow for any of the above in the future, without making code changes

Simplify Transaction Model Whenever Possible



Short, single-phase, server-side transactions are best for performance and reliability.

Simplifying Transaction Models

How much ACID do you really need?

- Need to strike balance between ACID and *throughput + responsiveness*
- Often forces us to improve concurrency at the expense of currency
- For example, many banking systems use stale data (yesterday's balances) as their starting point... unauthorized overdrafts sometimes result, but other limits keep that under control
 - Real-time balances that reflect intra-day balances increase system complexity

Simplifying Transaction Models

- ▼ If Read operations greatly outnumber Write operations:
 - You can arrange to send transactions requiring Writes to a particular container
 - All modifications to bean state happen in this one container
 - Can combine with problem-domain-level partitioning

Summary

- ▼ How to define a transaction and its ACID properties
- ▼ How transactions are coordinated across multiple data sources
- ▼ The interfaces used by the Java Transaction Service (JTS)
- ▼ How Enterprise Java Beans use distributed transactions
- ▼ The different transaction policies that can be used by Enterprise Java Beans
- ▼ The different Transaction Isolation policies that can be use by Enterprise Java Beans
- ▼ Changing transaction properties
- ▼ Studying the amount of transaction distribution you really need

Appendix A: Sample ejb-jar.xml file



```
// File:  ejb-jar.xml

<?xml version="1.0"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD
Enterprise JavaBeans 1.1//EN"
'http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd'>

<ejb-jar>
  <enterprise-beans>
    <session>
      <ejb-name>lab08.ShoppingCartHome</ejb-name>
      <home>lab08_StatefulSession.ShoppingCartHome</home>
      <remote>lab08_StatefulSession.ShoppingCart</remote>
      <ejb-class>lab08_StatefulSession.ShoppingCartBean</ejb-
class>
      <session-type>Stateful</session-type>
      <transaction-type>Container</transaction-type>
    </session>
  </enterprise-beans>
```

Appendix A: Sample ejb-jar.xml file

...continued

File: *ejb-jar.xml ...continued*

```
<container-transaction>
  <method>
    <ejb-name>lab08.ShoppingCartHome</ejb-name>
    <method-intf>Remote</method-intf>
    <method-name>performPurchase</method-name>
    <method-params></method-params></method>
    <trans-attribute>Required</trans-attribute>
  </container-transaction>
<container-transaction>
  <method>
    <ejb-name>lab08.ShoppingCartHome</ejb-name>
    <method-intf>Remote</method-intf>
    <method-name>*</method-name></method>
    <trans-attribute>NotSupported</trans-attribute>
  </container-transaction>
</assembly-descriptor>
</ejb-jar>
```

Appendix B: Sample weblogic-ejb-jar.xml file



```
<?xml version="1.0"?>
<!DOCTYPE weblogic-ejb-jar PUBLIC "-//BEA Systems, Inc.//DTD WebLogic 5.1.0 EJB//EN"
'http://www.bea.com/servers/wls510/dtd/weblogic-ejb-jar.dtd'>
<weblogic-ejb-jar><weblogic-enterprise-bean>
  <ejb-name>lab08.ShoppingCartHome</ejb-name>
  <caching-descriptor>
    <max-beans-in-free-pool>100</max-beans-in-free-pool>
    <max-beans-in-cache>100</max-beans-in-cache>
    <idle-timeout-seconds>300</idle-timeout-seconds>
  </caching-descriptor>
  <jndi-name>lab08.ShoppingCartHome</jndi-name>
  <transaction-isolation>
    <isolation-level>TRANSACTION_SERIALIZABLE</isolation-level>
  <method>
    <ejb-name>lab08.ShoppingCartHome</ejb-name>
    <method-intf>Remote</method-intf>
    <method-name>*</method-name>
  </method>
  </transaction-isolation>
</weblogic-enterprise-bean>
<security-role-assignment>
  <role-name>system</role-name>
  <principal-name>system</principal-name>
</security-role-assignment>
<security-role-assignment>
  <role-name>guest</role-name>
  <principal-name>guest</principal-name>
</security-role-assignment></weblogic-ejb-jar>
```