On Building Integrated and Distributed Database Systems

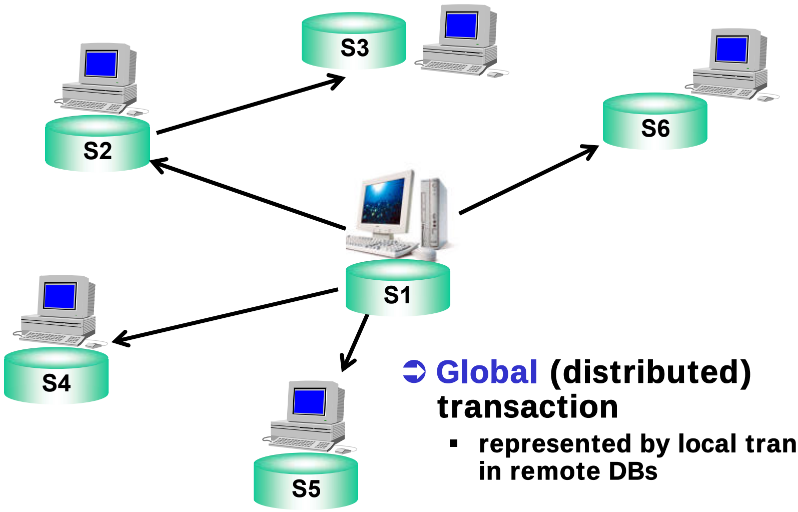
Transaction Commit in DDBS

Robert Wrembel
Poznań University of Technology
Institute of Computing Science
Poznań, Poland
`Robert.Wrembel@cs.put.poznan.pl`
`www.cs.put.poznan.pl/rwrembel`

Managing distributed transactions

- 2PC protocol
- 3PC protocol
- 2PC in Oracle implementation
- 2PC in heterogeneous DB systems

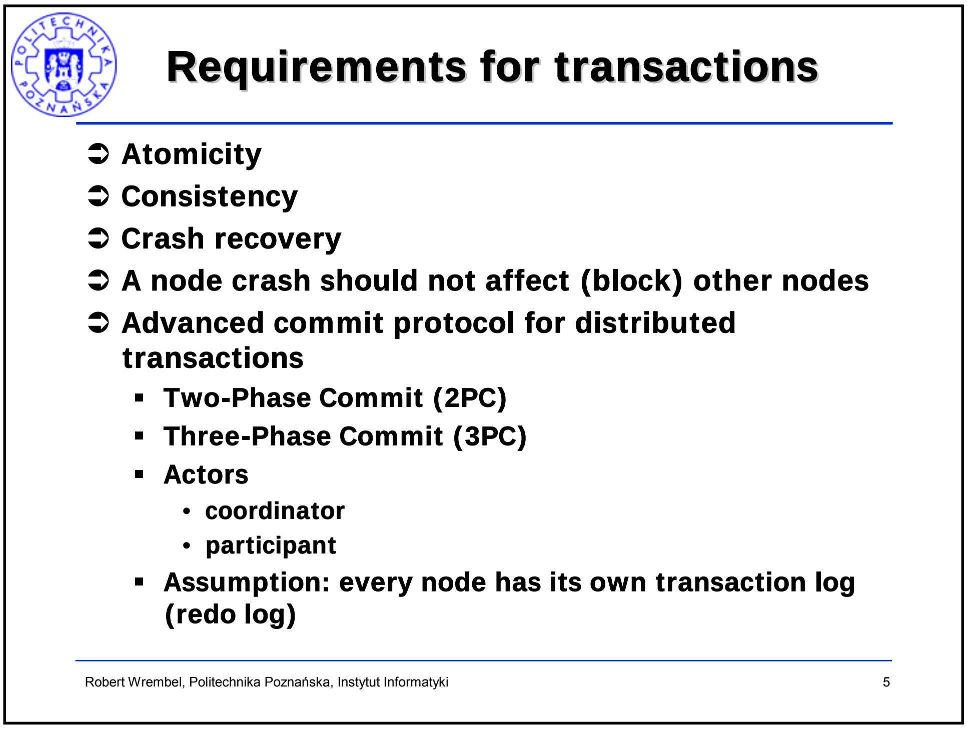
Distributed transaction



- ➲ **Global** (distributed) transaction
    - represented by local transactions in remote DBs

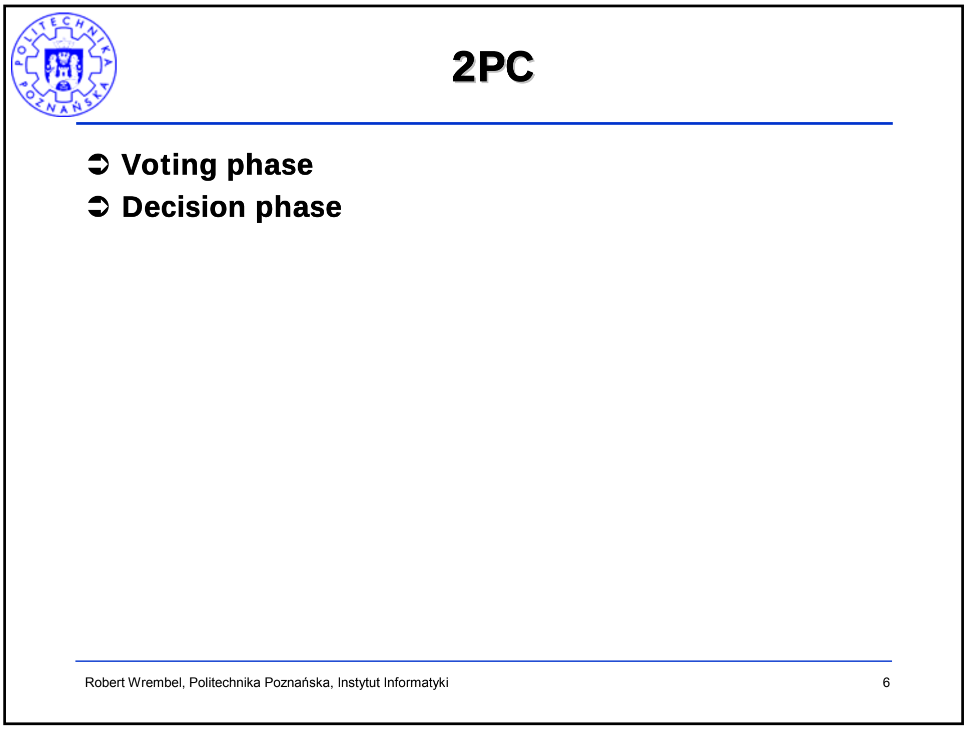
Crash in DDBS environment

- ➲ Message loss
- ➲ Medium crash
- ➲ Node crash
- ➲ Connection loss between site S1 and S2
    - S2 crashed
    - communication medium crashed
    - S2 is overloaded and cannot respond



Requirements for transactions

- Atomicity
- Consistency
- Crash recovery
- A node crash should not affect (block) other nodes
- Advanced commit protocol for distributed transactions
  - Two-Phase Commit (2PC)
  - Three-Phase Commit (3PC)
  - Actors
    - coordinator
    - participant
  - Assumption: every node has its own transaction log (redo log)



2PC

- Voting phase
- Decision phase

2PC - coordinator

- Voting phase
  1. Write to the log (on disk) begin_commit; send message PREPARE to all participants; wait for responses in a given time (timeout)
- Decision phase
  2. If message READY_COMMIT was received, register the node that responded; if all responded READY_COMMIT write commit to the log; send GLOBAL_COMMIT to all participants; wait for response (timeout)
  3. If all confirmed their commits, then write end_of_transaction to the log; if there exists a node that did not confirmed then resend GLOBAL_COMMIT to the node

2PC - coordinator

- Decision phase
  4. If at least one participant responded ABORT then write abort to the log; send message GLOBAL_ABORT to all participants; wait for response in a given time (timeout)

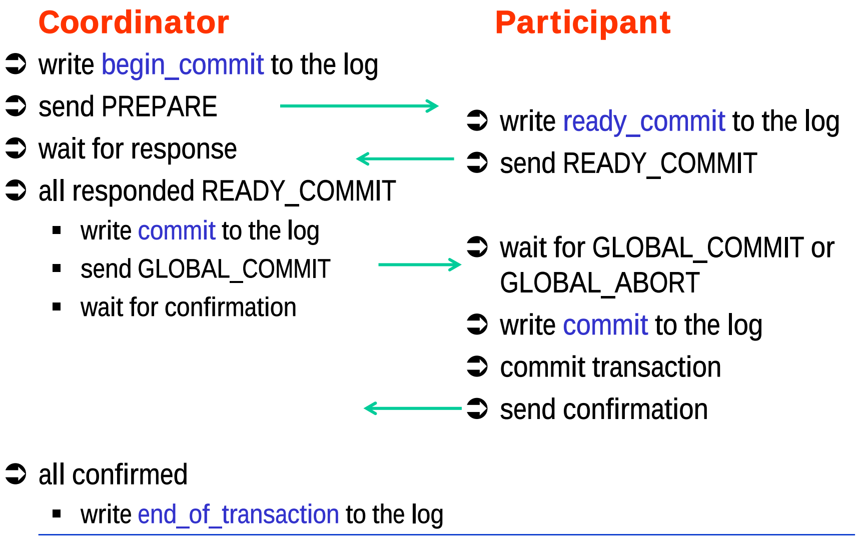
2PC - participant

- **Decision phase**
  1. **Receiving message PREPARE**
     - **write ready_commit to the log; write all DB buffers to disk; send READY_COMMIT to the coordinator**

     or

     - **write abort to the log; rollback transaction; send ABORT to the coordinator**
     - **wait for a message from the coordinator in a given time (timeout)**
  2. **If message GLOBAL_COMMIT received, then write commit to the log; commit transaction and release resources; send confirmation to the coordinator**
  3. **If message GLOBAL_ABORT received, then write abort to the log; rollback transaction; release resources; send confirmation to the coordinator**

2PC - summary



Coordinator

- write begin_commit to the log
- send PREPARE
- wait for response
- all responded READY_COMMIT
  - write commit to the log
  - send GLOBAL_COMMIT
  - wait for confirmation
- all confirmed
  - write end_of_transaction to the log

Participant

- write ready_commit to the log
- send READY_COMMIT
- wait for GLOBAL_COMMIT or GLOBAL_ABORT
- write commit to the log
- commit transaction
- send confirmation

2PC - summary

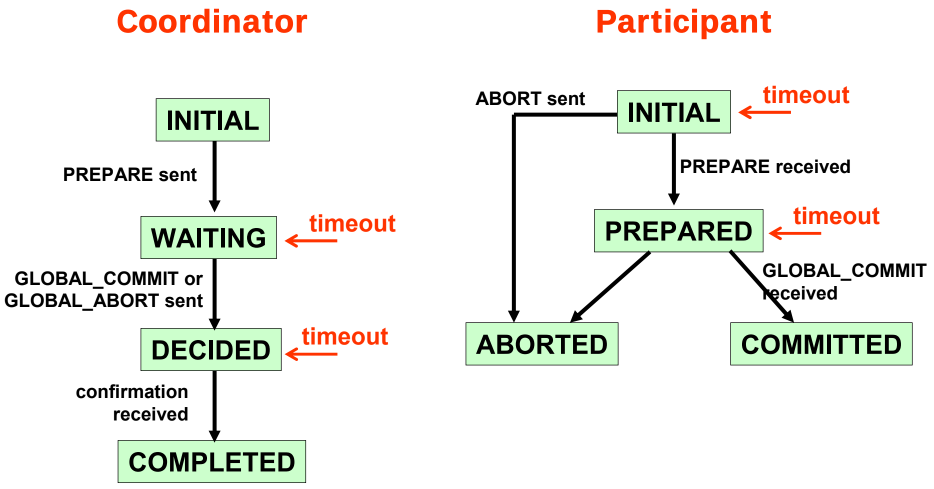
Coordinator

- write begin_commit to the log
- send PREPARE
- wait for response
- at least 1 responded ABORT
  - write abort to the log
  - send GLOBAL_ABORT
  - wait for confirmation
- all confirmed
  - write end_of_transaction to the log

Participant

- write abort to the log
- send ABORT
- rollback transaction

2PC - waiting

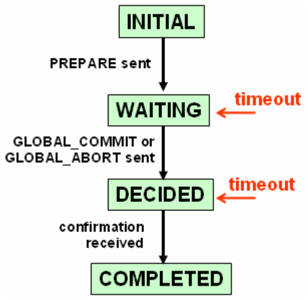


Reacting on timeout

- ➲ If a message does not arrive within a given time ⇨ timeout
- ➲ Timeout is managed by the termination protocol

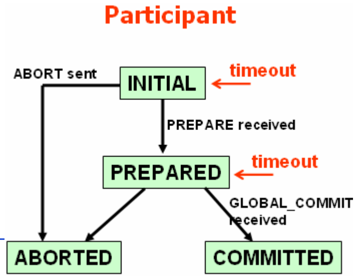
Termination protocol - coordinator

- ➲ In the WAITING state
  - commit decision is impossible
  - rollback decision is possible
- ➲ In the DECIDED state
  - resend decision to all the participants that have not confirmed



Termination protocol - participant

- In the INITIAL state
  - rollback transaction
  - if a participant received PREPARE after rollback, then it responds ABORT
- In the PREPARED state
  - participant is ready to commit ⇨ cannot rollback and cannot commit ⇨ the participant is locking data
  - minimizing the probability of locking ⇨ apply the cooperative termination protocol



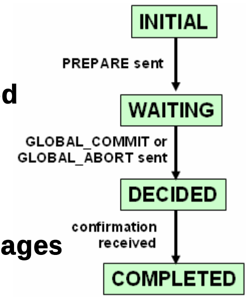
Cooperative termination protocol

- If the coordinator crashes ⇨ participants elect a new coordinator
  - a new coordinator finishes a distributed transaction
  - all participants must "know each other"
  - in every message from the coordinator the full list of participants is enclosed

Reacting on crash

- Node crash
- After repairing the node, recovery procedure is applied ⇨ recovery protocol

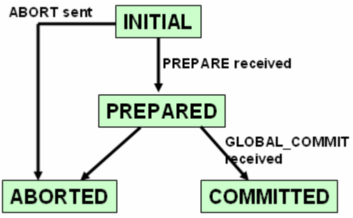
Recovery protocol - coordinator



- In the INITIAL state
  - if the commit procedure has not been started ⇨ start it
- In the WAITING state
  - wait for missing READY_COMMIT or ABORT
  - the next step depends on the received messages
- In the DECIDED state
  - decision has already been taken and message sent
  - if the coordinator received all the confirmations ⇨ end of work
  - is some messages are missing ⇨ apply the termination protocol in the DECIDED state

Recovery protocol - participant

- ➲ In the INITIAL state
  - rollback the transaction since the coordinator could only decide to abort
- ➲ In the PREPARED state
  - before crash the participant sent message ⇨ recovery by the termination protocol in the PREPARED state
- ➲ In the ABORTED/COMMITTED state
  - transaction was finished before crash ⇨ no action



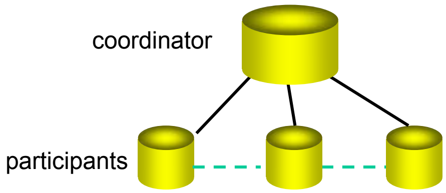
Communication topologies for 2PC

- ➲ Communication topology represents a "path" of exchanging messages between a coordinator and participants
- ➲ Centralized
- ➲ Linear
- ➲ Decentralized/distributed

Communication topologies for 2PC

➲ Centralized

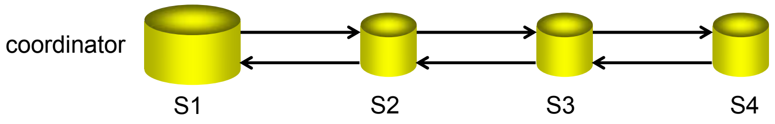
- the coordinator knows addresses of all participants
- electing a new coordinator ⇨ the address list is enclosed in messages
- there must exist a mean of communication between participants



Communication topologies for 2PC

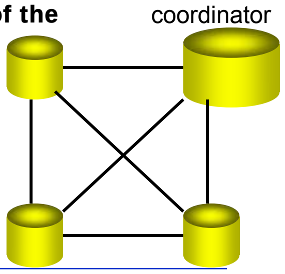
➲ Linear

- nodes are numbered (1 - coordinator, ...)
- node numbers represent the order of sending messages
- the voting phase: coordinator → participant
- the decision phase: participant → coordinator
- each participant adds to the received message it response ⇨ the message sent back includes a global decision



Communication topologies for 2PC

- Decentralized / distributed
  - the coordinator sends PREPARE to all participants
  - a participant sends its messages to all nodes
  - a participant waits with its decision for responses from all the other nodes ⇨ no need of a decision phase
    - a participant may take its decision independently on the others as it knows the global "state" of the transaction



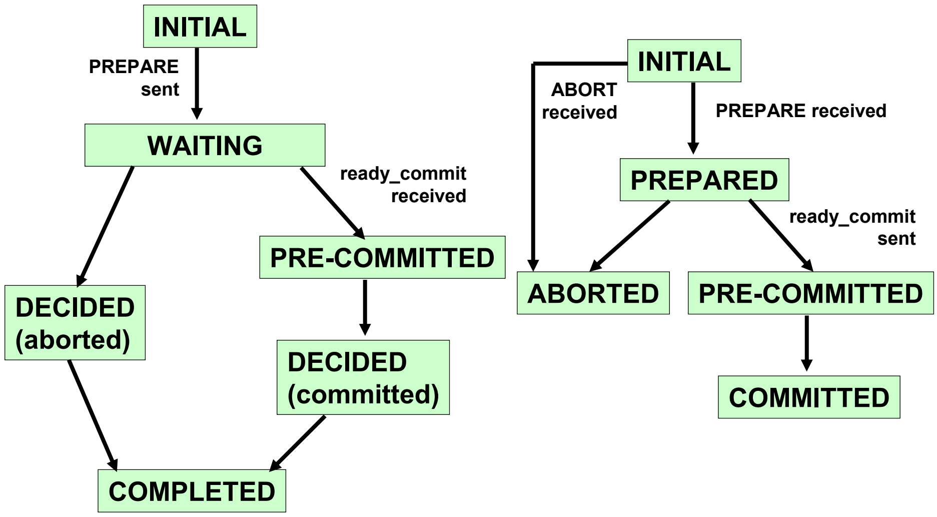
2PC summary

- Blocking protocol
  - a participant sends ready_commit but does not receive a message from the coordinator ⇨ locking data
  - probability of such scenario in practice is low ⇨ 2PC is implemented in practice

3PC

- Nonblocking protocol
  - at least 1 node must be on-line
- Minimizing time of uncertainty of a participant that responded ready_commit and is waiting for GLOBAL_COMMIT or GLOBAL_ABORT
- Coordinator
  - after receiving ready_commit from all the nodes it sends message PRE-COMMIT to all the participants
    - on receiving PRE_COMMIT a participant knows the global decision
  - after receiving the confirmation of PRE-COMMIT from all the participants, the coordinator sends COMMIT
  - ABORT is proceeded identically as 2PC

3PC



INITIAL

PREPARE sent

WAITING

ready_commit received

PRE-COMMITTED

DECIDED (aborted)

DECIDED (committed)

COMPLETED

INITIAL

ABORT received

PREPARE received

PREPARED

ready_commit sent

ABORTED

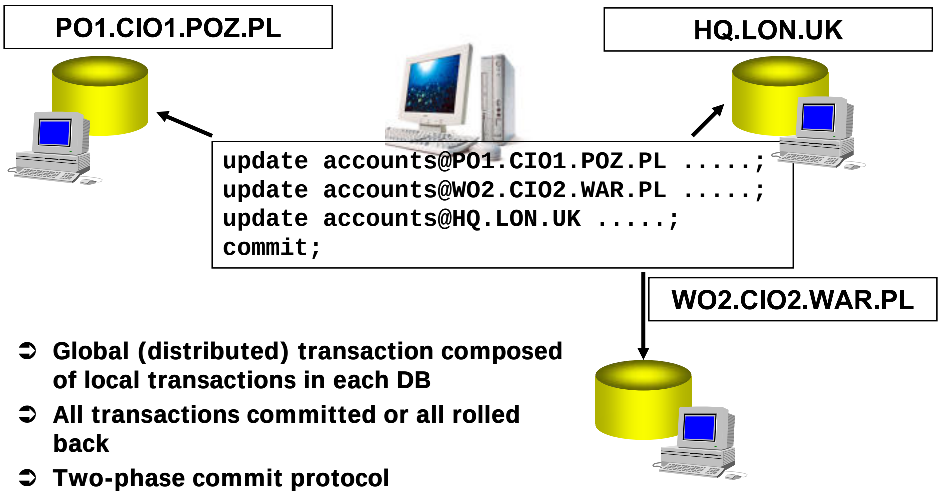
PRE-COMMITTED

COMMITTED

3PC

- A participant knows a global COMMIT decision before commit takes place
- Crash or timeout in the PRE-COMMITTED state does not require message exchange with the coordinator

2PC in Oracle



```
update accounts@PO1.CIO1.POZ.PL .....;
update accounts@WO2.CIO2.WAR.PL .....;
update accounts@HQ.LON.UK .....;
commit;
```

- Global (distributed) transaction composed of local transactions in each DB
- All transactions committed or all rolled back
- Two-phase commit protocol

Actors

- ➲ **Global coordinator (GC)**
  - **DB initiating a distributed transaction**
- ➲ **Participant**
  - **DB with a local transaction**
- ➲ **Commit point site (CPS)**
  - **initiates commit or rollback as instructed by GC**
  - **commits as the first one**
  - **selected by a DBA**
    - **instance config. parameter COMMIT_POINT_STRENGTH**
      - value 0-255
      - represents the importance of a DB
      - represents the quality (reliability) of a node
  - **a node having the highest value of COMMIT_POINT_STRENGTH becomes GC**

Actors

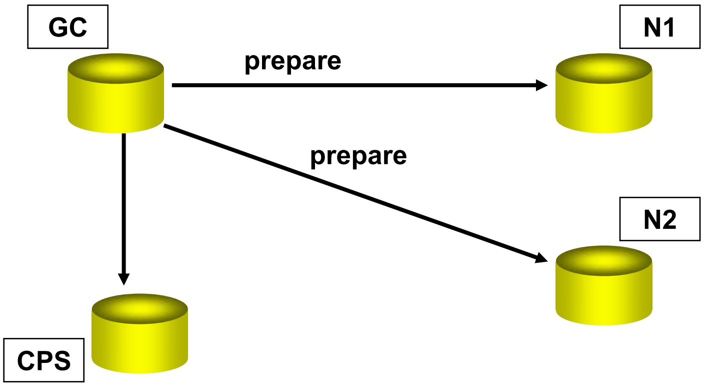
- ➲ **Commit point site**
  - **stores commit status of a distributed transaction**
  - **a distributed transaction is considered as committed if CPS has already committed it (even if other nodes haven't committed yet)**

2PC

- Prepare
- Commit
- Forget

2PC - prepare phase (GC)

- GC selects CPS
- GC sends PREPARE messages to participants



2PC - prepare phase (participant)

- Receives PREPARE message from GC
- If the participant is a local coordinator it propagates the message to its subordinate nodes
- If the participant haven't modified data ⇨ reply with READ-ONLY
- Write all buffers to the redo log
- IF a node is a local coordinator then receive PREPARE messages from its subordinate nodes and then reply PREPARED to GC
  - ELSE
    - rollback its local transaction
    - send ABORT to GC

2PC - commit phase

- GC receives confirmations from the participants
    - PREPARED
    - READ-ONLY (no updates)
    - ABORT (unable to prepare to commit)
- If all responded PREPARED ⇨ GC sends commit to CPS
    - CPS commits and sends a conformation to GC
    - upon receiving the confirmation from CPS, GC sends the COMMIT message to all the participants

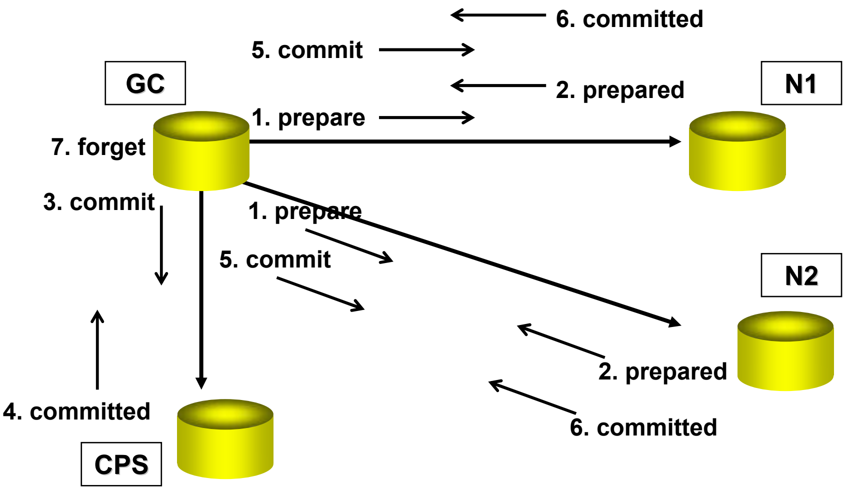
2PC - commit phase

- If at least one participant responded ABORT ⇨ GC sends the ROLLBACK message to CPS
    - CPS rolls back the transaction and sends a confirmation to GC
    - GC sends the ROLLBACK message to all the participants

2PC - commit phase (participant)

- Receives from GC the COMMIT message
- Commits its local transaction
- Releases locks
- Saves the commit record to the redo log

2PC - summary



Limiting the number of dist. trans.

- Instance configuration parameter DISTRIBUTED_TRANSACTIONS
- DISTRIBUTED_TRANSACTIONS=0
  - no distributed transactions
  - the background RECO process does not start

Crashes

- ➲ In the COMMIT (ROLLBACK) phase a network or node crashes
  - not all nodes committed (rolled back)
  - not all nodes confirmed their successful operations
  - distributed transaction enters the "in-doubt" state
- ➲ Automatic recovery of a distributed transaction (by RECO) in the "in-doubt" state after repairing the system
  - either all DBs commit or all roll back

Locking

- ➲ Distributed transaction in the "in-doubt" state locks data
- ➲ Other transaction requesting the locked data will receive
  - ORA-01591: lock held by in-doubt distributed transaction <id>
  - a command requesting the lock may be re-executed later

Locking

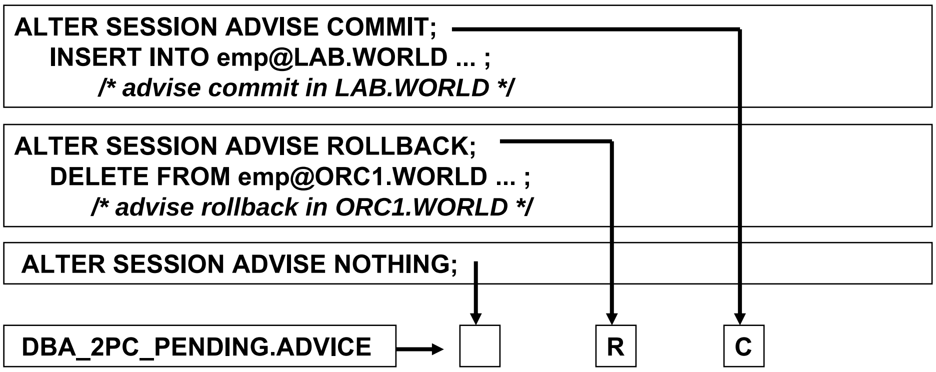
- ➲ Distributed transaction requests a lock on data in a remote node but the data have already been locked
  - waiting time for lock release ⇨ instance configuration parameter DISTRIBUTED_LOCK_TIMEOUT [sec]
    - 1- infinite waiting time; default 60
  - after timeout a command is automatically rolled back and can be re-executed later
    - ORA-02049: time-out distributed transaction waiting for lock

Locking

- ➲ Locks must be immediately released
- ➲ Repairing the system may take long time
- ➲ Manual intervention of a DBA

Manual commit / rollback

- ➲ Figuring out whether a local transaction (part of a distributed one) must be rolled back or committed ⇨ system view SYS.DBA_2PC_PENDING



ALTER SESSION ADVISE COMMIT;
INSERT INTO emp@LAB.WORLD ... ;
/ advise commit in LAB.WORLD */*

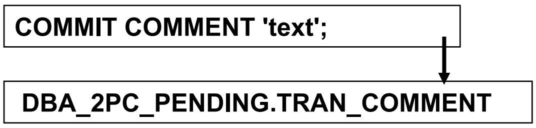
ALTER SESSION ADVISE ROLLBACK;
DELETE FROM emp@ORC1.WORLD ... ;
/ advise rollback in ORC1.WORLD */*

ALTER SESSION ADVISE NOTHING;

DBA_2PC_PENDING.ADVICE

Manual commit / rollback

- ➲ Commenting transactions



COMMIT COMMENT 'text';

DBA_2PC_PENDING.TRAN_COMMENT

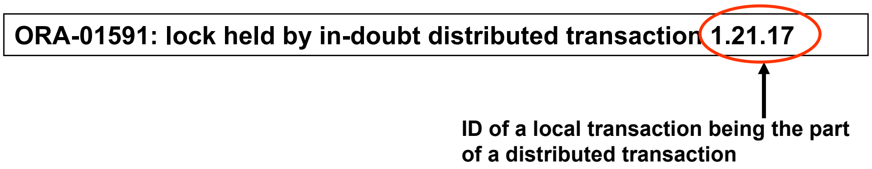
- ➲ Commentary
  - may describe type of an application
  - max. 50 characters

set transaction name 'money transter';

```
SQL> select name, status from v$transaction;
NAME                 STATUS
-------------------- ----------------
money transger       ACTIVE
```

Manual commit / rollback

- **Example scenario**: a user executing local transaction receives an error



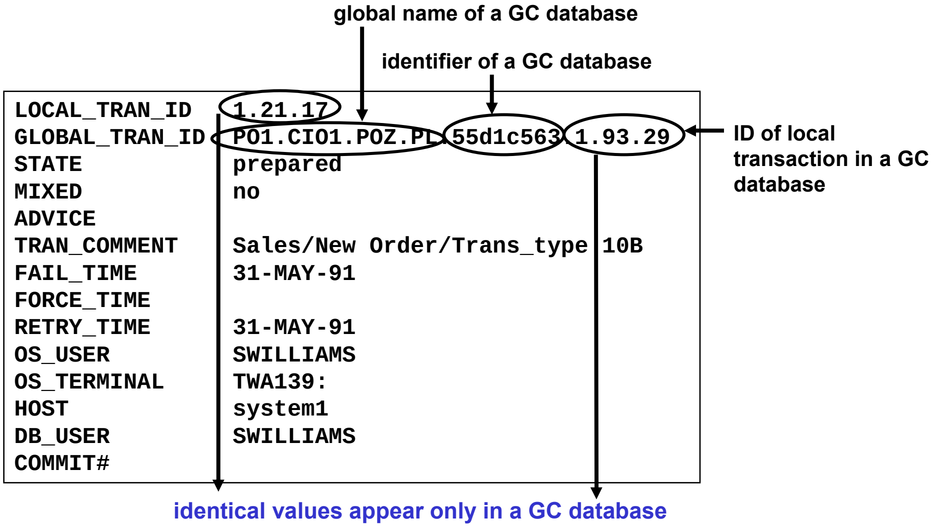
ORA-01591: lock held by in-doubt distributed transaction 1.21.17

ID of a local transaction being the part of a distributed transaction

- Accessing DBA_2PC_PENDING in a local DB

```
SELECT * FROM sys.dba_2pc_pending
WHERE local_tran_id = '1.21.17';
```

Manual commit / rollback



global name of a GC database

identifier of a GC database

```
LOCAL_TRAN_ID   1.21.17
GLOBAL_TRAN_ID  P01.CI01.POZ.PL.55d1c563.1.93.29
STATE           prepared
MIXED           no
ADVICE
TRAN_COMMENT    Sales/New Order/Trans_type 10B
FAIL_TIME       31-MAY-91
FORCE_TIME
RETRY_TIME      31-MAY-91
OS_USER         SWILLIAMS
OS_TERMINAL     TWA139:
HOST            system1
DB_USER         SWILLIAMS
COMMIT#
```

ID of local transaction in a GC database

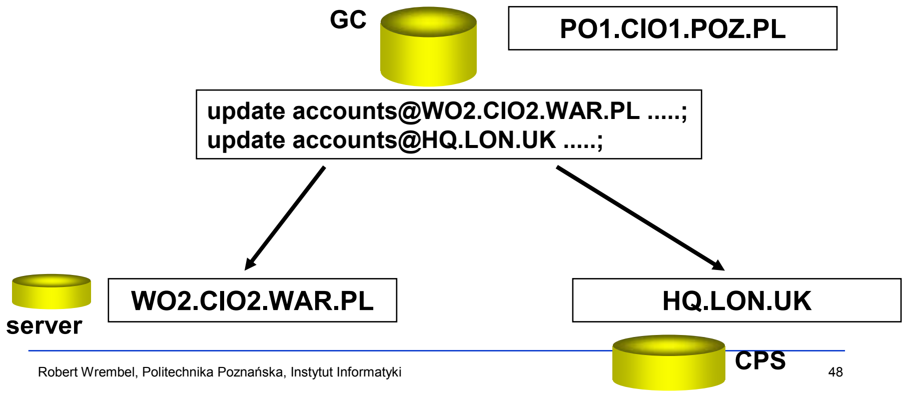
identical values appear only in a GC database

Manual commit / rollback

- The STATE column
  - collecting
    - only in a GC DB
    - collecting confirmation messages from participants
  - prepared
    - transaction prepared to commit
    - GC could receive confirmation message or not
  - committed
    - transaction in this node has been committed
  - forced commit
    - transaction committed manually
  - forced abort
    - transaction aborted manually

Manual commit / rollback

- Find CPS as it stores the information whether a distributed transaction was committed or not
- View SYS.DBA_2PC_NEIGHBORS



GC

PO1.CIO1.POZ.PL

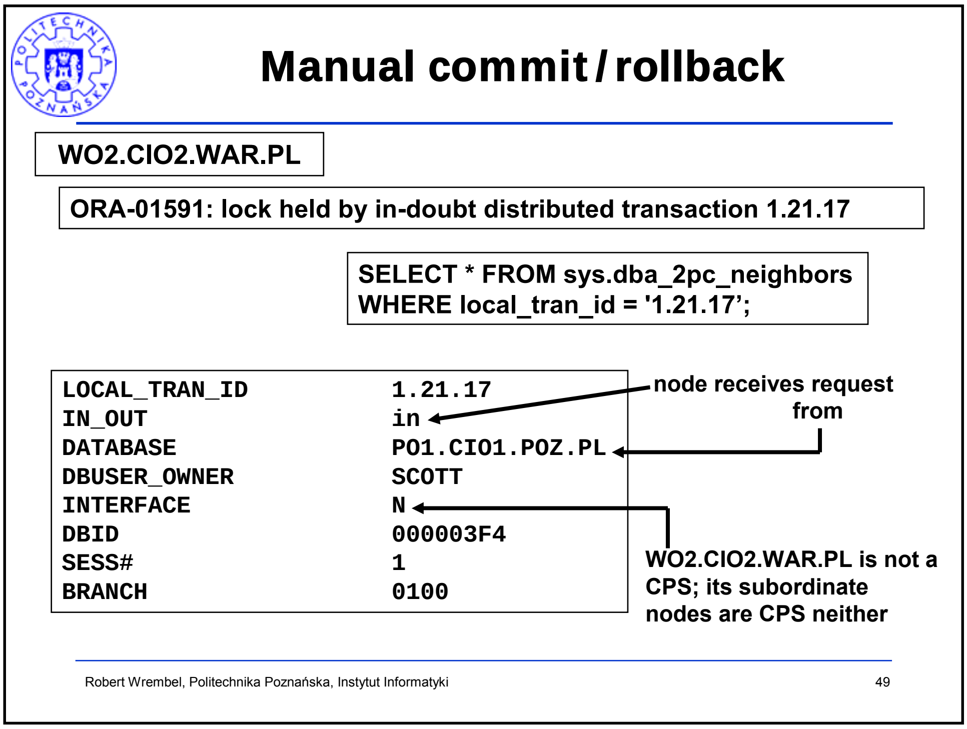
update accounts@WO2.CIO2.WAR.PL .....;
update accounts@HQ.LON.UK .....;

server

WO2.CIO2.WAR.PL

HQ.LON.UK

CPS



Manual commit / rollback

WO2.CIO2.WAR.PL

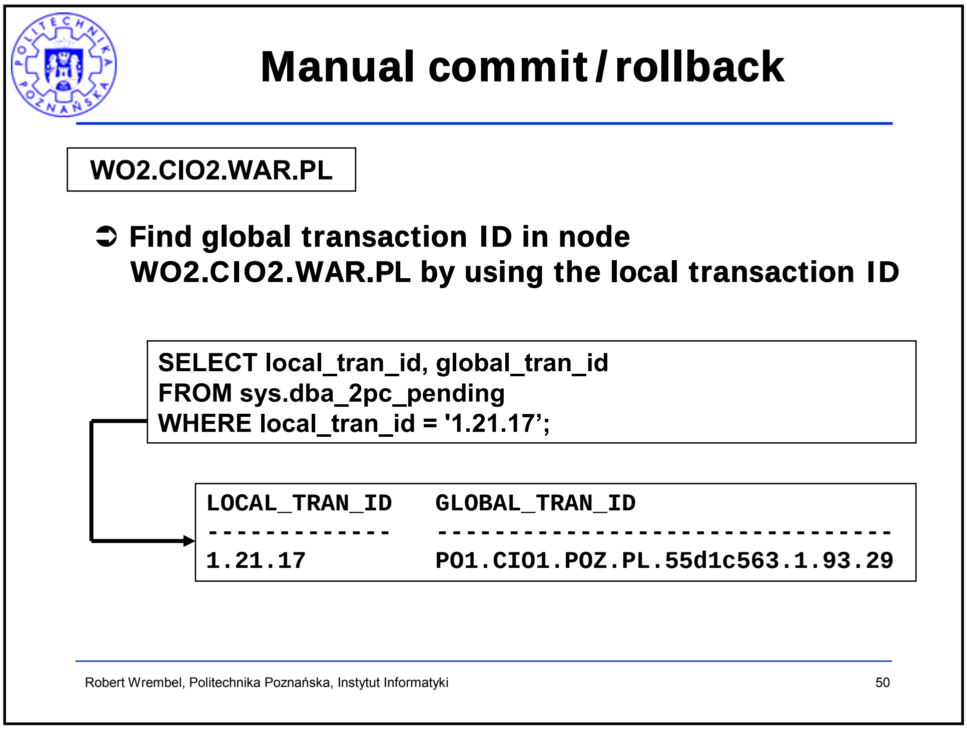
ORA-01591: lock held by in-doubt distributed transaction 1.21.17

```
SELECT * FROM sys.dba_2pc_neighbors
WHERE local_tran_id = '1.21.17';
```

```
LOCAL_TRAN_ID        1.21.17
IN_OUT               in
DATABASE             PO1.CIO1.POZ.PL
DBUSER_OWNER         SCOTT
INTERFACE            N
DBID                 000003F4
SESS#                1
BRANCH               0100
```

node receives request from

WO2.CIO2.WAR.PL is not a CPS; its subordinate nodes are CPS neither



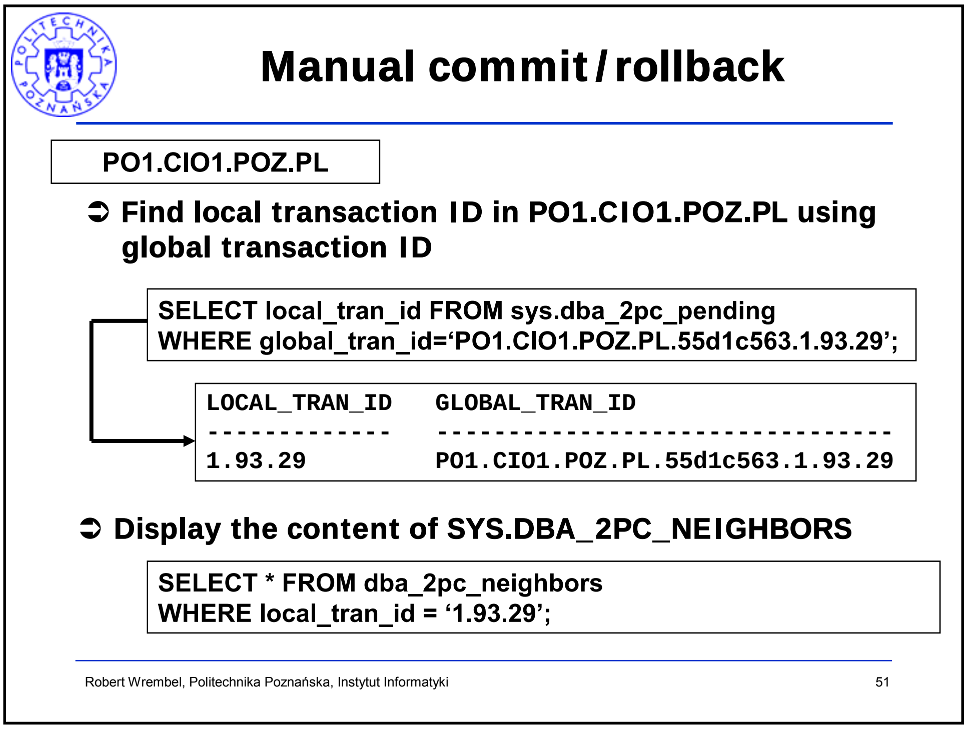
Manual commit / rollback

WO2.CIO2.WAR.PL

- ➲ Find global transaction ID in node WO2.CIO2.WAR.PL by using the local transaction ID

```
SELECT local_tran_id, global_tran_id
FROM sys.dba_2pc_pending
WHERE local_tran_id = '1.21.17';
```

```
LOCAL_TRAN_ID   GLOBAL_TRAN_ID
-------------   --------------------------------
1.21.17         PO1.CIO1.POZ.PL.55d1c563.1.93.29
```



Manual commit / rollback

PO1.CIO1.POZ.PL

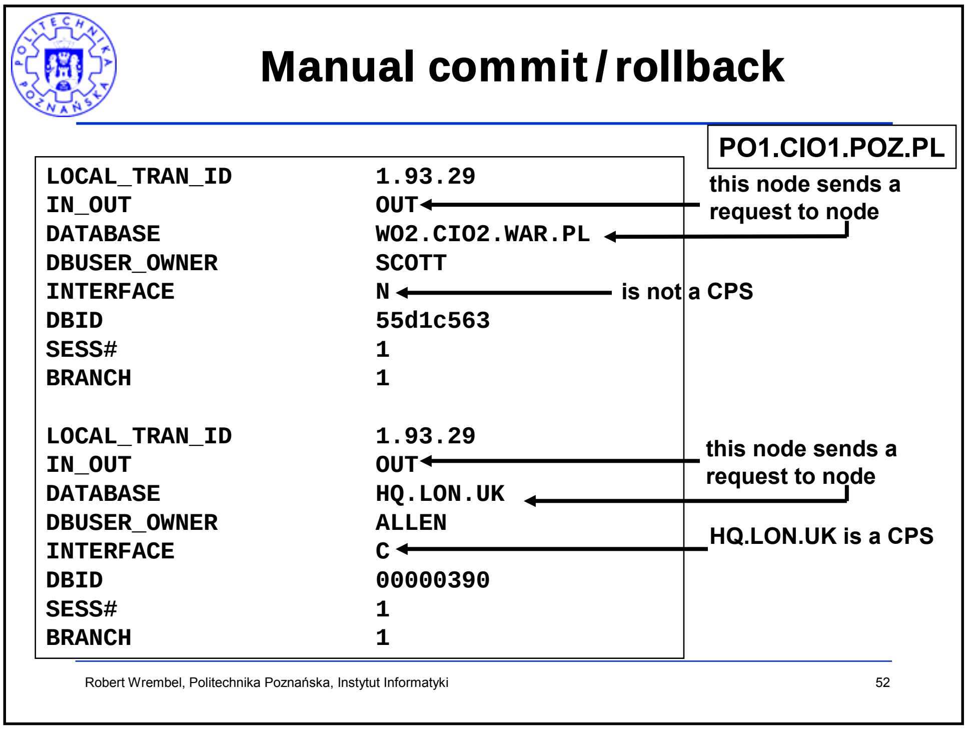
➲ Find local transaction ID in PO1.CIO1.POZ.PL using global transaction ID

```
SELECT local_tran_id FROM sys.dba_2pc_pending
WHERE global_tran_id='PO1.CIO1.POZ.PL.55d1c563.1.93.29';
```

```
LOCAL_TRAN_ID   GLOBAL_TRAN_ID
-------------   --------------------------------
1.93.29         PO1.CIO1.POZ.PL.55d1c563.1.93.29
```

➲ Display the content of SYS.DBA_2PC_NEIGHBORS

```
SELECT * FROM dba_2pc_neighbors
WHERE local_tran_id = '1.93.29';
```



Manual commit / rollback

PO1.CIO1.POZ.PL

```
LOCAL_TRAN_ID          1.93.29
IN_OUT                 OUT
DATABASE               WO2.CIO2.WAR.PL
DBUSER_OWNER           SCOTT
INTERFACE              N
DBID                   55d1c563
SESS#                  1
BRANCH                 1

LOCAL_TRAN_ID          1.93.29
IN_OUT                 OUT
DATABASE               HQ.LON.UK
DBUSER_OWNER           ALLEN
INTERFACE              C
DBID                   00000390
SESS#                  1
BRANCH                 1
```

this node sends a request to node

is not a CPS

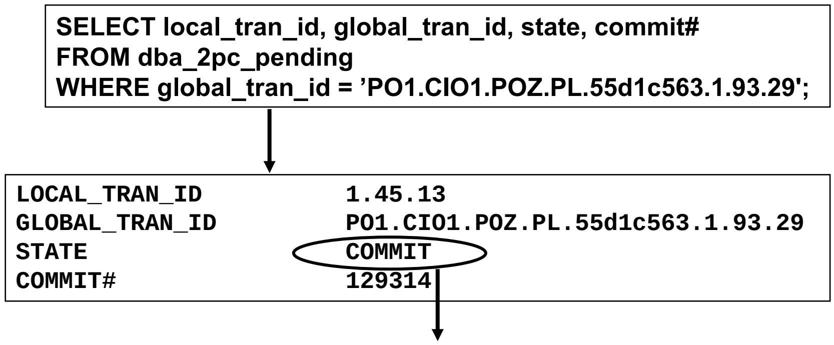
this node sends a request to node

HQ.LON.UK is a CPS

Manual commit / rollback

HQ.LON.UK

- ➲ Get the transaction state in the CPS

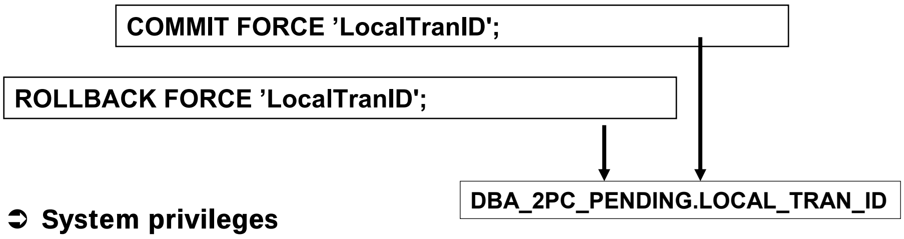


```
SELECT local_tran_id, global_tran_id, state, commit#
FROM dba_2pc_pending
WHERE global_tran_id = ’PO1.CIO1.POZ.PL.55d1c563.1.93.29';
```

```
LOCAL_TRAN_ID          1.45.13
GLOBAL_TRAN_ID         P01.CI01.POZ.PL.55d1c563.1.93.29
STATE                  COMMIT
COMMIT#                129314
```

- ➲ Global transaction has been committed

Manual commit / rollback



```
COMMIT FORCE ’LocalTranID';
```

```
ROLLBACK FORCE ’LocalTranID';
```

DBA_2PC_PENDING.LOCAL_TRAN_ID

- ➲ System privileges
  - FORCE TRANSACTION, FORCE ANY TRANSACTION
- ➲ DISTRIBUTED_RECOVERY_CONNECTION_HOLD_TIME
  - Time (in seconds) during which a database link remains active if a distributed transaction cannot be finished
  - Default 200 sec

Crash tests

- Simulating crashes of a distributed transaction

```
COMMIT COMMENT ‘ORA-2PC-CRASH-TEST-n’;
```

1 : Crash commit point site after collect
2 : Crash non-commit point site after collect
3 : Crash before prepare (non-commit point site)
4 : Crash after prepare (non-commit point site)
5 : Crash commit point site before commit
6 : Crash commit point site after commit
7 : Crash non-commit point site before commit
8 : Crash non-commit point site after commit
9 : Crash commit point site before forget
10: Crash non-commit point site before forget

Turn off automatic recovery of distributed transactions in all DBs

```
ALTER SYSTEM DISABLE DISTRIBUTED RECOVERY;
```

Heterogeneous environments

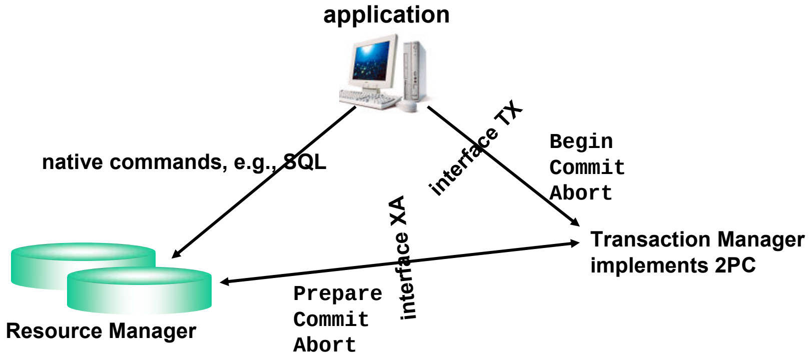
➲ Support for 2PC

➲ Different implementations of 2PC

➲ Heterogeneous systems must cooperate ⇨ communication interface must be standardized

- LU6.2 (IBM)
- X/Open Distributed Transaction Processing (OSI-TP)

X/Open DTP



- TM tasks
  - defining transaction scope (commands in the transaction)
  - managing transaction IDs
  - communicating with other TMs

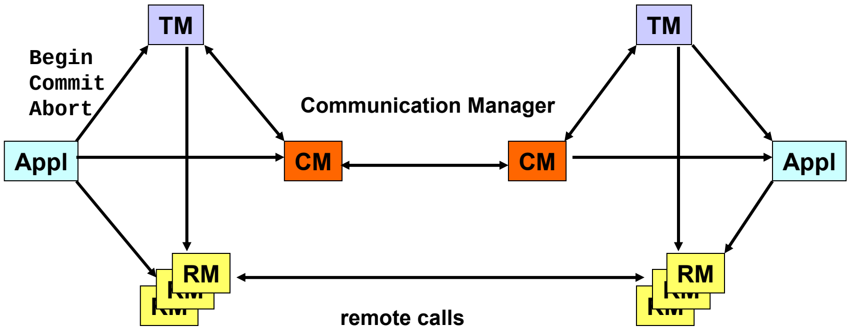
X/Open DTP

- **Application**
  - **requesting that TM starts a distributed transaction**
  - **sending commands to RM**
  - **requesting that TM finishes a distributed transaction**

X/Open DTP

- Interface TX: application → TM
  - tx_open, tx_close – opening closing a session
  - tx_begin – starting a transaction
  - tx_commit, tx_abort – commit, abort
- Interface XA: TM → RM
  - xa_open, xa_close – attaching, detaching from RM
  - xa_start, xa_end – starting, ending transaction of a given ID
  - xa_rollback – rolling back a transaction of a given ID
  - xa_prepare – preparing a transaction of a given ID
  - xa_commit – committing a transaction of a given ID
  - xa_forget – releasing resources

X/Open DTP



Begin
Commit
Abort

Communication Manager

remote calls

Transactions in JDBC

- JDBC 1.0 ⇨ distributed transactions not supported
  - connecting to a db by means of connect string
- JDBC 2.0 ⇨ distributed transactions supported
  - connecting to a db by means of data source definitions
  - DS definitions are available in a name service accessible via Java Naming and Directory Interface (JNDI)
- JDBC 3.0 ⇨ savepoints supported
- Typical Java database applications are based on EJB components
  - distributed transaction requires
    - Java application
    - application server
    - transaction manager
    - resource manager (database)
  - communication between components defined by the Java Transaction API (JTA) standard

Simple example

- JDBC 1.x
  - Connection object ⇨ responsible for executing a DB command
  - Driver Manager ⇨ opens connection with a DB using a registered driver
    - the driver is selected based on connect string to a database

```
// registering Oracle JDBC
DriverManager.registerDriver(new oracle.jdbc.OracleDriver());
// opening db connection by means of DriverManager
conn = DriverManager.getConnection("jdbc:oracle:thin:@dcs-rw:1521:ora", "scott","tiger");
conn.setAutoCommit(false);
// executing SQL command
Statement stmt = conn.createStatement();
stmt.executeUpdate("update emp set sal=sal*1.1 where deptno=10");
// other SQL commands
conn.commit();
```

by default autocommit is true

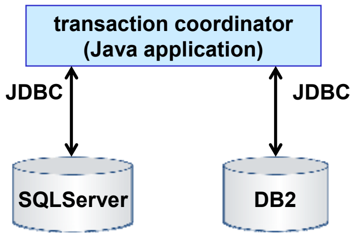
or conn.rollback();

Simple example

- ➲ Using data source definitions and JNDI

```
// get object representing JNDI context
Context cont = new InitialContext();
// begin transaction
UserTransaction trans = (UserTransaction) ctx.lookup("java:comp/env/UserTransaction");
// by means of a name get data source description
OracleDataSource dataSrc = (OracleDataSource)ctx.lookup("jdbc/Emp");
// get the connection object
Connection conn = dataSrc.getConnection();
// SQL commands
trans.commit();
```

Case study: SQLServer-DB2



- ➲ SQLServer 2008 R2 Express
- ➲ DB2 9.7 Express-C
- ➲ OS: Windows XP SP3
- ➲ JDBC type 4 (includes transaction support with the XA interface)

Transaction state (SQLServ)

- **sys.dm_tran_active_transactions.transaction_state**
  - **0: transaction has not been completely initialized yet**
  - **1: transaction has been initialized but has not been started**
  - **2: transaction is active**
  - **3: transaction has ended (used for read-only transactions)**
  - **4: commit process has been initiated on the distributed transaction (the distributed transaction is still active but further processing cannot take place)**
  - **5: transaction is in a prepared state and waiting resolution**
  - **6: transaction has been committed**
  - **7: transaction is being rolled back**
  - **8: transaction has been rolled back**
- **sys.dm_tran_active_transactions.dtc_state**
  - **1: ACTIVE**
  - **2: PREPARED**
  - **3: COMMITTED**
  - **4: ABORTED**
  - **5: RECOVERED**

Transaction state (DB2)

- **Graphical application: IBM DB2 →Monitoring Tools → Indoubt Transaction Manager**
- **`list indoubt transactions`**
  - Committed: transaction committed manually
  - Ended: transaction ended, possibly by timeout
  - Indoubt: waiting to be committed or rolled back
  - Missing commit acknowledgement: transaction waiting for commit message
  - Rolled back

Configuration

- **SQLServer**
  - **mixed (DB, OS) authentication mode**
  - **TCP/IP connections allowed (required for JDBC)**
  - **run MSDTC (Microsoft Distributed Transaction Cooridnator)**
  - **attach system library** sqljdbc_xa.dll → copy from directory xa32 (32-bit OS) or ia64 (64-bit OS) where JDBC was installed into SQLServer home directory/bin), run xa_install.sql as system administrator
  - grant role SqlJDBCXAUser to a user executing distributed transaction
- **DB2**
  - **standard configuration**

Implementation outline (DB2)

- **Create a connection and set up connection parameters**

```
Class.forName("com.ibm.db2.jcc.DB2Driver");
com.ibm.db2.jcc.DB2XADataSource db2XaDS=new com.ibm.db2.jcc.DB2XADataSource();
DB2XADataSource db2XaDS = new DB2XADataSource();
Properties db2Properties = new Properties();
db2XaDS.setServerName("10.0.1.100");
db2XaDS.setDatabaseName("SAMPLE");
db2XaDS.setDriverType(4);
db2XaDS.setPortNumber(50000);
```

- **Make the connection**

```
db2XAConn = db2XaDS.getDB2XAConnection("ist", "ist", db2Properties);
//get the object representing the connection (javax.sql.Connection)
db2Conn = db2XAConn.getConnection();
//get the object representing XA resource
db2XAResource = db2XAConn.getXAResource();
```

Implementation outline (DB2)

➲ Create transaction ID

- the same value is assigned to global transaction ID
- different values are assigned to local transaction IDs

```
xidDb2 = XidImpl.getUniqueXid(1,1);
xidSqlsrv = XidImpl.getUniqueXid(1,2);
```

➲ Start the transaction

```
db2XAResource.start(xidDb2, XAResource.TMNOFLAGS);
sqlsrvXAResource.start(xidSqlsrv, XAResource.TMNOFLAGS);
```

➲ Execute a statement and receive its results → implement interface java.sql.Statement

```
Statement sqlsrv2stmt = sqlsrvConn.createStatement();
sqlsrv2stmt.execute(sqlText);
```

Implementation outline (DB2)

➲ 2PC: sending the prepare message

```
db2XAResource.end(xidDb2, XAResource.TMSUCCESS);
sqlsrvXAResource.end(xidSqlsrv, XAResource.TMSUCCESS);
int prpDb2;
int prpSqlsrv;
try { prpDb2 = db2XAResource.prepare (xidDb2);
    } catch (XAException xae)
       {xae.printStackTrace();
        prpDb2 = -1;}
try { prpSqlsrv = sqlsrvXAResource.prepare (xidSqlsrv);
    } catch (XAException xae)
       {xae.printStackTrace();
        prpSqlsrv = -1;}
```

Implementation outline (DB2)

➲ Decide if global commit is possible

```
if(!((prpDb2 == XAResource.XA_OK) || (prpDb2 == XAResource.XA_RDONLY)))
  doCommit = false;
if(!((prpSqlsrv == XAResource.XA_OK) || (prpSqlsrv == XAResource.XA_RDONLY)))
  doCommit = false;
if (prpDb2 == XAResource.XA_OK)
  { if (doCommit) {//send global commit
                   db2XAResource.commit (xidDb2, false);}
    else {//send global rollback
          db2XAResource.rollback (xidDb2);}
  }
if (prpSqlsrv == XAResource.XA_OK)
  { if (doCommit) {//send global commit
                   sqlsrvXAResource.commit (xidSqlsrv, false);}
    else {//send global rollback
          sqlsrvXAResource.rollback (xidSqlsrv);}
  }
```

Implementation outline (DB2)

➲ Close the connections

```
db2Conn.close();
db2Conn = null;
sqlsrvConn.close();
sqlsrvConn = null;
db2XAConn.close();
db2XAConn = null;
sqlsrvXAConn.close();
sqlsrvXAConn = null;
```

Implementation outline (SQLServ)

```
SQLServerXADataSource msDs = new SQLServerXADataSource();
msDs.setUser("ist");
msDs.setPassword("ist");
msDs.setServerName("10.0.1.100");
msDs.setPortNumber(1433);
msDs.setDatabaseName("master");
XAConnection msXaCon = msDs.getXAConnection();
XAResource msXaRes = msXaCon.getXAResource();
```

Tests

- **Commit of a prepared transaction → success**
- **Rollback of a prepared transaction → success**
- **Global coordinator crash before prepare**
  - **DB2 → rollback**
  - **SQLServer → transaction is active (locked data) → manual kill**
    - find the transaction in sys.dm_tran_active_transactions. transaction_ouw
    - **`kill ouw`**
- **Global coordinator crash after prepare**
  - **DB2 → in doubt (locked data)**
    - **`db2 list indoubt transactions`**
    - **`db2 list indoubt transactions with prompting`** (prompt for rollback or commit)
  - **SQLServer → in doubt (locked data)**
    - use MSDTC (**`dcomcnfg`**)

Tests

- Node (SQLServer) crash before prepare
  - SQLServer transaction is rolled back
  - DB2 transaction is rolled back by the global coordinator

JTA

- Java application servers offer standard Java Transaction API (JTA) → a set of methods in JTA which "packages" your traditional JDBC calls into the Two-Phase-Commit protocol
- Annotation @TransactionManagement(TransactionManagementType.BEAN) must be present if we want to successfully obtain the UserTransaction object

```
@TransactionManagement(TransactionManagementType.BEAN)

Context context = new InitialContext();
EJBContext ejbContext = (EJBContext) context.lookup("java:comp/EJBContext");
UserTransaction userTransaction = ejbContext.getUserTransaction();
```

JTA

```
com.ibm.db2.jcc.DB2XADataSource db2DS = new DataSources.getDB2DataSource();
db2ds.setUser("ist");
db2ds.setPassword("ist");
db2ds.setServerName("192.168.1.3");
db2ds.setPortNumber(50001);
db2ds.setDriverType(4);
db2ds.setDatabaseName("example");
XAConnection db2xaCon = db2DS.getDB2XAConnection();
```

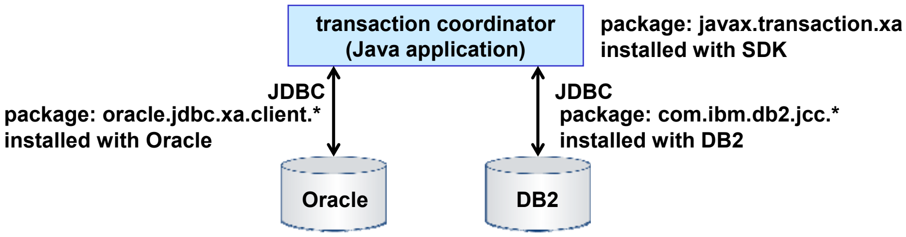
```
SQLServerXADataSource msDS = new DataSources.getMSDataSource();
msDs.setUser("ist");
msDs.setPassword("ist");
msDs.setServerName("192.168.1.4");
msDs.setPortNumber(1433);
msDs.setDatabaseName("master");
XAConnection msxaCon = msDS.getXAConnection();
```

JTA

```
userTransaction.begin();
 Statement statement1 = db2con.createStatement();
 statement1.executeUpdate("update emp set sal=sal*1.1 where ename='BLAKE'");
 Statement statement2 = mscon.createStatement();
 statement2.executeUpdate("update emp set sal=sal*1.2");
userTransaction.commit();
```

```
statement1.close();
statement2.close();
db2con.close();
mscon.close();
```

Case study: Oracle-DB2



transaction coordinator (Java application)

package: javax.transaction.xa installed with SDK

JDBC package: oracle.jdbc.xa.client.* installed with Oracle

JDBC package: com.ibm.db2.jcc.* installed with DB2

- Oracle11g R2
- IBM DB2 9.7 Express-C
- JDBC type 4
- Windows XP

Transaction state (Oracle)

- SYS.DBA_2PC_PENDING.STATE
  - collecting
  - prepared
  - committed
  - forced commit
  - forced rollback

Tests

- Global coordinator crash before prepare
  - DB2 → rollback
  - Oracle → rollback
- Global coordinator crash after prepare
  - DB2 → in doubt (locked data)
    - `db2 list indoubt transactions`
    - `db2 list indoubt transactions with prompting` (prompt for rollback or commit)
  - Oracle → in doubt (locked data)
    - SYS.DBA_2PC_PENDING.STATUS
    - `commit force 'local_tran_id';`
- Node (Oracle) crash before prepare
  - DB2 transaction is rolled back by the global coordinator
  - Oracle transaction is rolled back

Tests

- Node (DB2) crash before commit (the transaction is prepared)
  - DB2 → transaction status: Indoubt
- Node (Oracle) crash before commit (the transaction is prepared)
  - Oracle → prepared (locked data)