

---

# Introduction to transaction management

Dario Della Monica

These slides are a modified version of the slides provided with the book  
Özsu and Valduriez, *Principles of Distributed Database Systems* (3rd Ed.), 2011

The original version of the slides is available at: [extras.springer.com](http://extras.springer.com)

# Outline (distributed DB)

---

- Introduction (Ch. 1) \*
- Distributed Database Design (Ch. 3) \*
- Distributed Query Processing (Ch. 6-8) \*
- Distributed Transaction Management (Ch. 10-12) \*
  - Introduction to transaction management (Ch. 10) \*
  - Distributed Concurrency Control (Ch. 11) \*
  - Distributed DBMS Reliability (Ch. 12) \*

---

\* Özsu and Valduriez, *Principles of Distributed Database Systems* (3rd Ed.), 2011

# Outline (today)

---

- Introduction to transaction management (Ch. 10) <sup>\*</sup>
  - Definitions of transaction
  - Properties of Transactions (ACID)
    - ◆ Atomicity
    - ◆ Consistency
    - ◆ Isolation
    - ◆ Durability
  - Workflow of transaction management

---

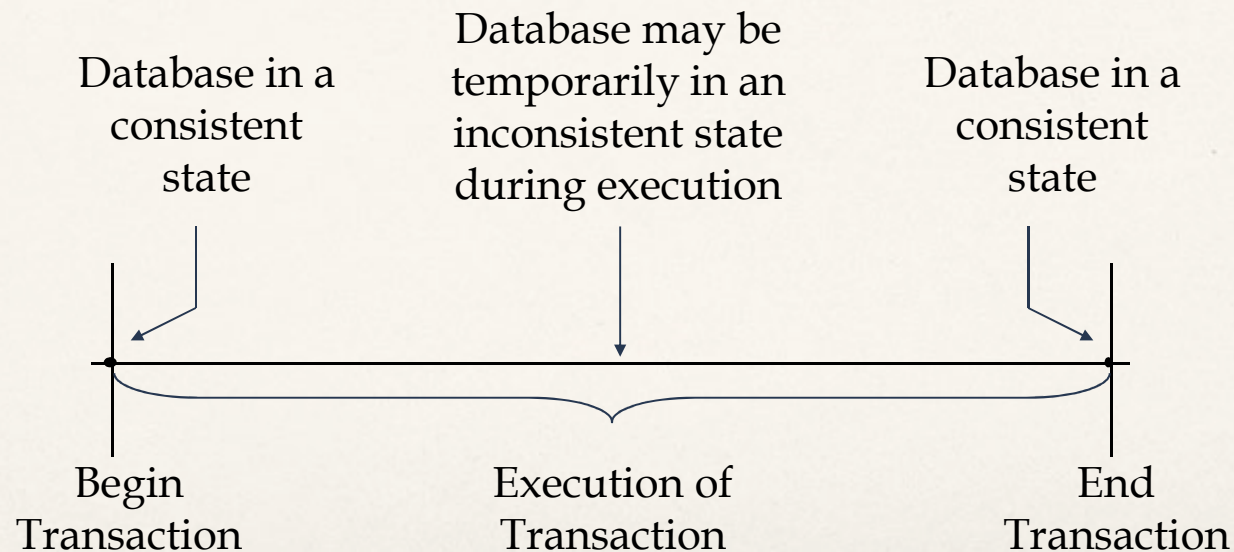
<sup>\*</sup> Özsu and Valduriez, *Principles of Distributed Database Systems* (3rd Ed.), 2011



# Transactions

A transaction is a collection of actions that make transformations of system states while preserving system consistency (from consistent state to another consistent state)

- concurrency: expected behavior when 2 queries modify the DB simultaneously
- Integrity: integrity constraints (e.g., primary/foreign keys), replicated copies have same values
- failure: restart or abort on failure while updating



# Alternative definitions

---

- One way to see transactions: we often treat a transaction as a program, that is, a sequence of DB operations, Write (W) and Read (R), interleaved with computation steps (e.g.,  $x := x+1$ ) and flow control instructions (if.then-else instructions), and delimited by Begin (B) and Commit (C)/ Abort (A)
- Another way to see then: a transaction is just a single execution the program with no selection instructions and computation steps

# Transaction Example – A Simple SQL Query

---

**Transaction** BUDGET\_UPDATE

**begin**

```
EXEC SQL      UPDATE PROJ
              SET   BUDGET = BUDGET*1.1
              WHERE PNAME = "CAD/CAM"
```

**end.**



# Example Database

---

Consider an airline reservation example with the relations:

FLIGHT(FNO, DATE, SRC, DEST, STSOLD, CAP)

CUST(CNAME, ADDR)

FC(FNO, DATE, CNAME, SPECIAL)

# Example Transaction – A Simple Program

---

**Begin\_transaction** Reservation

**begin**

**input**(flight\_no, date, customer\_name);

**EXEC SQL UPDATE FLIGHT**

**SET STSOLD = STSOLD + 1**

**WHERE FNO = flight\_no AND DATE = date;**

**EXEC SQL INSERT**

**INTO FC(FNO, DATE, CNAME, SPECIAL);**

**VALUES (flight\_no, date, customer\_name, **null**);**

**output**("reservation completed")

**end . {Reservation}**



# Termination condition

---

- Commit (C) vs. Abort (A)
- Commit (C) denotes success
  - DB goes into a new state, visible to everybody
  - Cannot be undone
- Abort (A) happens on failure
  - Application logic reach a failure state (Abort keyword in the program)
    - ◆ Bad input, unfulfilled condition
    - ◆ Controlled through the program flow control (e.g., if-then-else)
    - ◆ E.g., a seat is reserved but payment does not go through
  - Deadlock (Abort command is sent by DBMS or OS)
  - Node/hardware failure
  - Abort causes **rollback** (restore the state before transaction started)

# Termination of Transactions

---

**Begin\_transaction** Reservation

**begin**

**input**(flight\_no, date, customer\_name);

**EXEC SQL**    **SELECT**            STSOLD,CAP  
                                 **INTO**                    temp1,temp2  
                                 **FROM**                    FLIGHT  
                                 **WHERE**                  FNO = flight\_no AND DATE = date;

**if** temp1 = temp2 **then**

**output**("no free seats");

**Abort**

**else**

**EXEC SQL** **UPDATE**            FLIGHT  
                                 **SET**                    STSOLD = STSOLD + 1  
                                 **WHERE**                  FNO = flight\_no AND DATE = date;

**EXEC SQL** **INSERT**  
                                 **INTO**                  FC(FNO, DATE, CNAME, SPECIAL);  
                                 **VALUES**                (flight\_no, date, customer\_name, **null**);

**Commit**

**output**("reservation completed")

**endif**

**end .** {Reservation}

# Properties of Transactions

---

---

## ATOMICITY

(Ch. 12) \*

- unit of operation, all or nothing/ Abort or Commit

## CONSISTENCY

(Ch. 11) \*

- ensures correctness (if DB is in a consistent state, so is after transaction execution, independently from failures or other issues)
  - ◆ no violation of integrity constraints
  - ◆ expected behavior in presence of concurrency

## ISOLATION

(Ch. 11) \*

- changes visible only after commit
- Intermediate changes invisible to other transactions ⇒ serializability

## DURABILITY

(Ch. 12) \*

- committed updates persist (permanent, cannot be undone)

---

\* Özsu and Valduriez, *Principles of Distributed Database Systems* (3rd Ed.), 2011



# Atomicity

---

- Either **all or none** of the transaction's operations are performed
- Atomicity requires that if a transaction is interrupted by a failure, its partial results must be **undone**
- The activity of preserving the transaction's atomicity in presence of transaction aborts due to input errors, system overloads, or deadlocks is called **transaction recovery**
- The activity of ensuring atomicity in the presence of system crashes is called **crash recovery**

# Consistency

---

- Internal consistency
  - A transaction which executes **alone** against a **consistent** database leaves it in a consistent state.
  - Transactions do not violate database integrity constraints
- Transactions are **correct** programs

# Consistency Degrees

---

- Degree 0
  - Transaction  $T$  does not overwrite dirty data of other transactions
  - Dirty data refers to data values that have been updated by a transaction prior to its commitment
- Degree 1
  - $T$  does not overwrite dirty data of other transactions
  - $T$  does not commit any writes before EOT



# Consistency Degrees (cont'd)

---

- Degree 2
  - $T$  does not overwrite dirty data of other transactions
  - $T$  does not commit any writes before EOT
  - $T$  does not read dirty data from other transactions
- Degree 3
  - $T$  does not overwrite dirty data of other transactions
  - $T$  does not commit any writes before EOT
  - $T$  does not read dirty data from other transactions
  - Other transactions do not dirty any data read by  $T$  before  $T$  completes.

# Isolation

---

- Serializability
  - If several transactions are executed concurrently, the results must be the same as if they were executed serially in some order
- Incomplete results
  - An incomplete transaction cannot reveal its results to other transactions before its commitment
  - Necessary to avoid cascading aborts

# Isolation Example

---

- Consider the following two transactions:

$T_1$ :    Read( $x$ )  
           $x \leftarrow x+1$   
          Write( $x$ )  
          Commit

$T_2$ :    Read( $x$ )  
           $x \leftarrow x+1$   
          Write( $x$ )  
          Commit

- Possible execution sequences:

$T_1$ :    Read( $x$ )  
 $T_1$ :     $x \leftarrow x+1$   
 $T_1$ :    Write( $x$ )  
 $T_1$ :    Commit  
 $T_2$ :    Read( $x$ )  
 $T_2$ :     $x \leftarrow x+1$   
 $T_2$ :    Write( $x$ )  
 $T_2$ :    Commit

$T_1$ :    Read( $x$ )  
 $T_1$ :     $x \leftarrow x+1$   
 $T_2$ :    Read( $x$ )  
 $T_1$ :    Write( $x$ )  
 $T_2$ :     $x \leftarrow x+1$   
 $T_2$ :    Write( $x$ )  
 $T_1$ :    Commit  
 $T_2$ :    Commit



# SQL-92 Isolation Levels

---

## Phenomena:

- Dirty read
  - $T_1$  modifies  $x$  which is then read by  $T_2$  before  $T_1$  terminates;  $T_1$  aborts
    - ◆  $T_2$  has read value which never exists in the database
- Non-repeatable (fuzzy) read
  - $T_1$  reads  $x$ ;  $T_2$  then modifies or deletes  $x$  and commits.  $T_1$  tries to read  $x$  again but reads a different value or can't find it
- Phantom
  - $T_1$  searches the database according to a predicate while  $T_2$  inserts new tuples that satisfy the predicate

# SQL-92 Isolation Levels (cont'd)

---

- Read Uncommitted
  - For transactions operating at this level, all three phenomena are possible
- Read Committed
  - Fuzzy reads and phantoms are possible, but dirty reads are not
- Repeatable Read
  - Only phantoms possible
- Anomaly Serializable
  - None of the phenomena are possible

# Durability

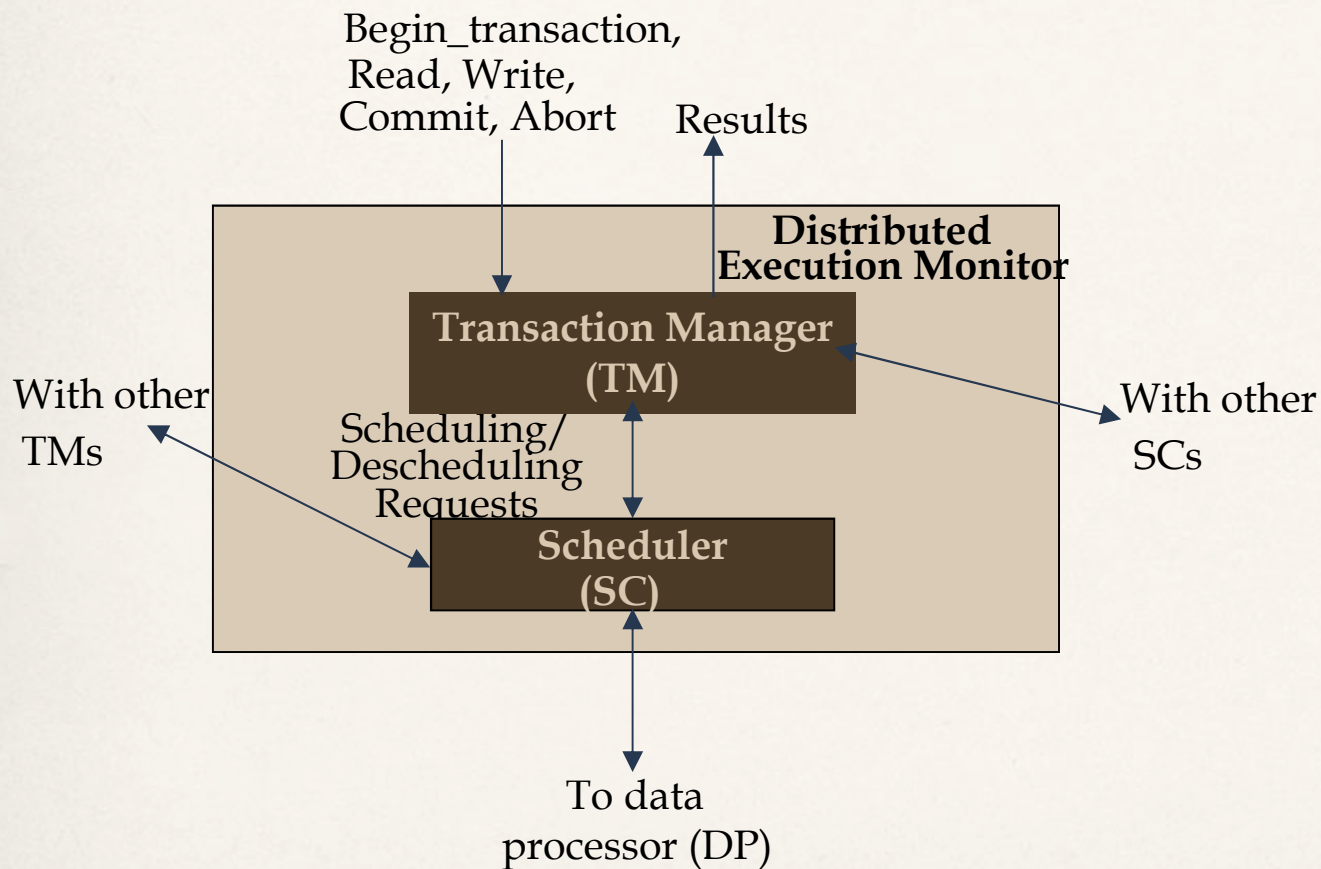
---

---

- Once a transaction commits, the system must guarantee that the results of its operations will never be lost, in spite of subsequent failures
- Database recovery



# Architecture



**TM:** coordinates requests (OP) of transaction operations by applications, sends requests to SC's at same and different sites

**SC:** manages concurrent accesses to resources (DB entities)

**DP:** local DBMS module for data manipulation

# Transaction management protocol

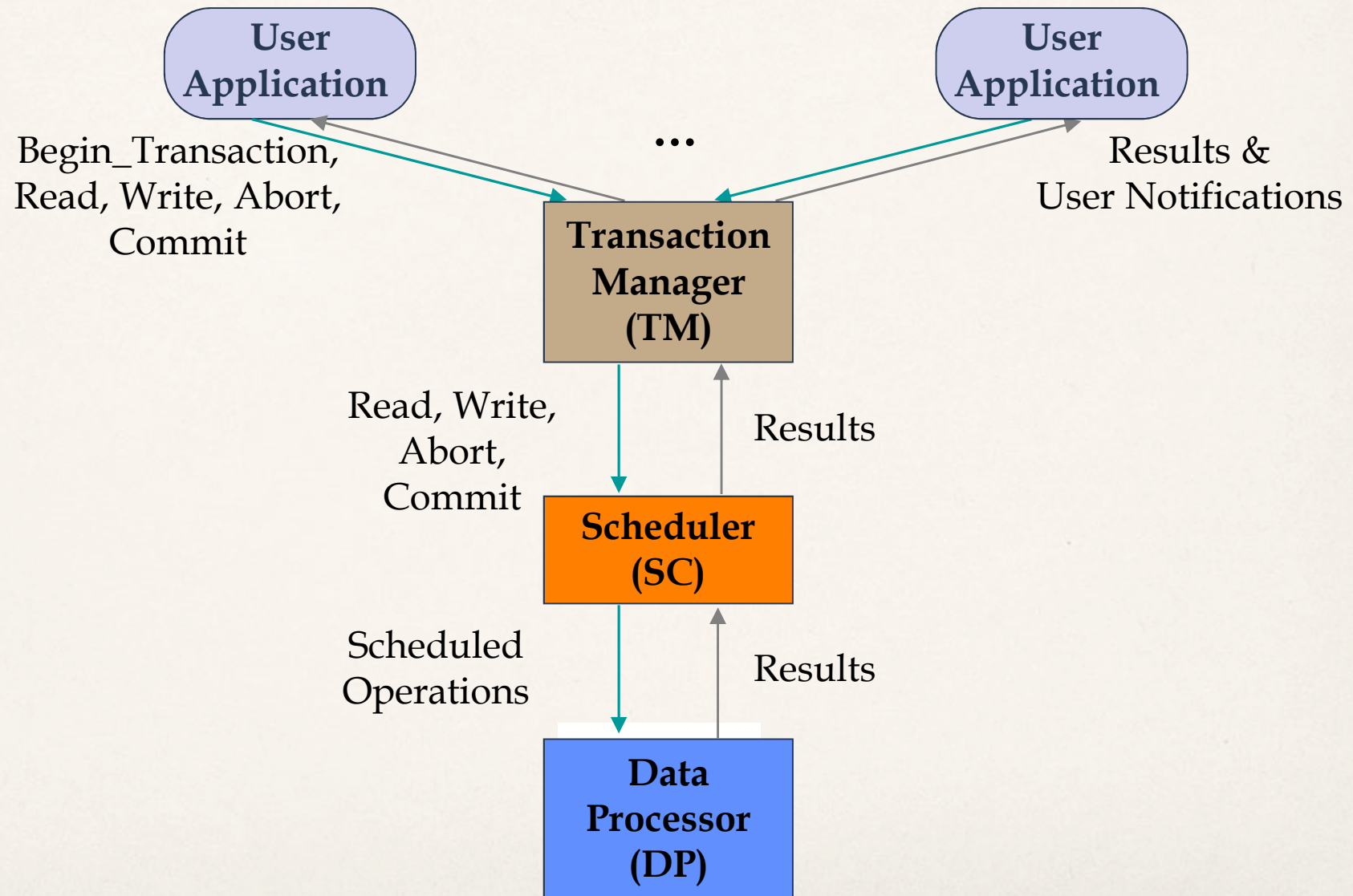
---

- Transactions originate at one site
- TM of that site will be the coordinator for that transaction
- Transaction operations (interface between TM and user/application)
  - { B, R, W, C, A }
  - B (Begin): TM does some bookkeeping (record transaction name, originating site, originating application, ...)
  - R (Read)/W (Write) – these have to do with concurrent access control (Consistency and Isolation) – Ch. 11\*:
    - ♦ TM asks local/remote DP to read/update **after** concurrent access controls is granted by local/remote SC that *guarantees mutual exclusion in accessing data and serializability* (isolation and thus consistency)
  - C (Commit) – this has to do with reliability (Atomicity and Durability) – Ch. 12\*:
    - ♦ TM coordinates all sites involved to make data permanently available
  - A (Abort) – this has to do with reliability (Atomicity and Durability) – Ch. 12\*:
    - ♦ TM coordinates rollback; no effect of transaction is visible to other transactions
- We ignore data replication. To extend our discussion see Ch. 13 (we do not cover that chapter)

---

\* Özsu and Valduriez, *Principles of Distributed Database Systems* (3rd Ed.), 2011

# Centralized Transaction Execution





# Distributed Transaction Execution

