Distributed Transaction Management

Transaction Concepts and Models

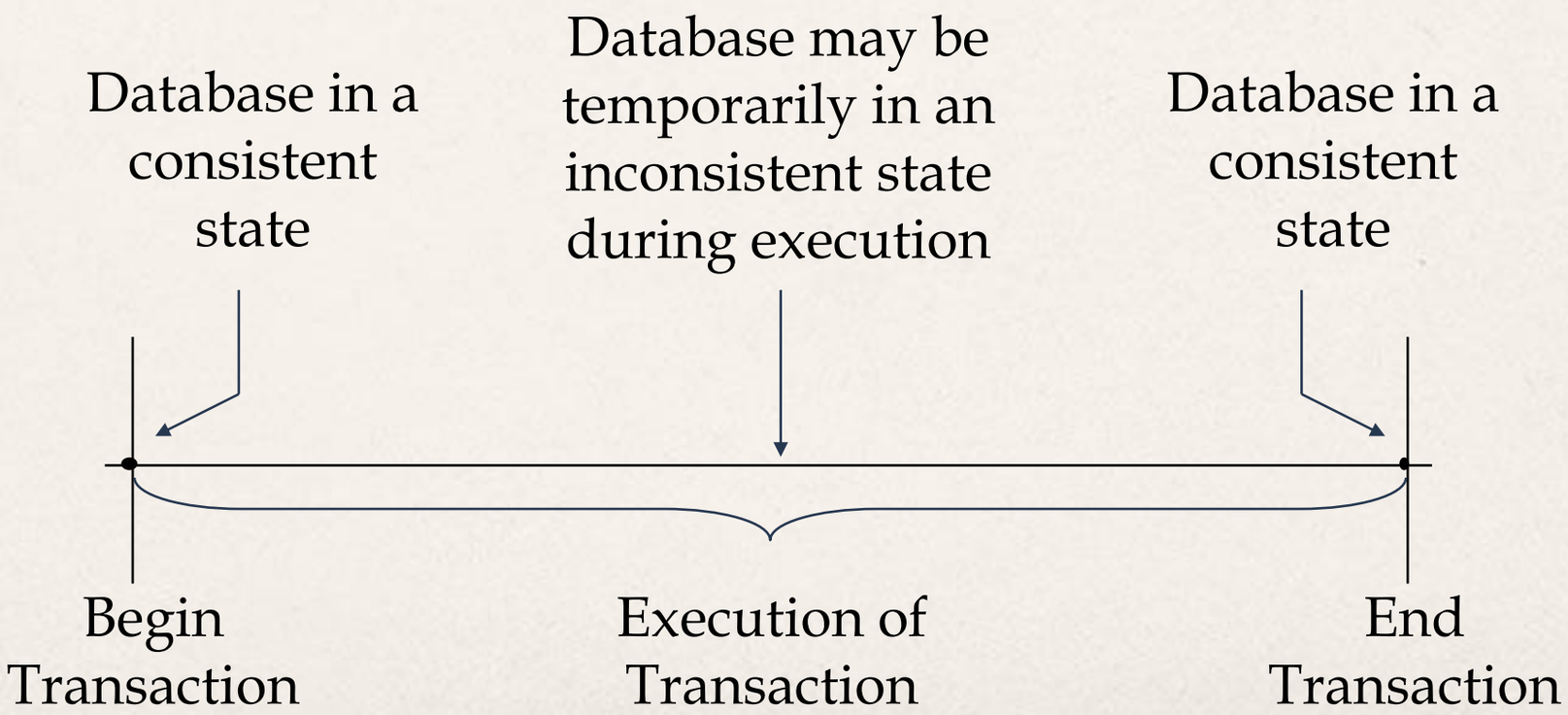
Contents

- Definition of a Transaction
  - Termination Conditions
  - Characterization and Formalization
- Properties of Transactions
  - ACID: Atomicity, Consistency, Isolation, Durability
- Architecture Revisited

Transaction

A transaction is a collection of actions that make consistent transformations of system states while **preserving system consistency**.

- **concurrency transparency** — concurrency control
- **failure transparency** — reliability



Transaction Example
A Simple SQL Query

Begin_transaction Salary_Update

begin

EXEC SQL UPDATE PAY

SET SAL = SAL*1.1

end

Example Database

Consider an airline reservation example with the relations:

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

CUST(CNAME, ADDR, BAL)

FC(FNO, DATE, CNAME,SPECIAL)

Transaction Example
SQL Notation

```
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
```

Termination Conditions

```
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
```

Characterization

- Read set (RS)
  - The set of data items that are read by a transaction
- Write set (WS)
  - The set of data items whose values are changed by this transaction
- Base set (BS)
  - RS ∪ WS

- RS[Reservation] = {FLIGHT.STSOLD, FLIGHT.CAP, FLIGHT.FNO, FLIGHT.DATE}
- WS[Reservation] = {FLIGHT.STSOLD, FC.FNO, FC.DATE, FC.CNAME, FC.SPECIAL}

Formalization

Let

- $O_{ij}(x)$ be some operation O_j of transaction T_i operating on entity x, where $O_j \in$ {read,write} and O_j is atomic
- $OS_i = \cup_j O_{ij}$
- $N_i \in$ {abort,commit}

Transaction T_i is a partial order $T_i = \{\sum_i , \prec_i\}$ where

1. $\sum_i = OS_i \cup \{N_i\}$
2. For any two operations O_{ij} , $O_{ik} \in OS_i$, if $O_{ij} = R(x)$ and $O_{ik} = W(x)$ for any data item x, then either $O_{ij} \prec_i O_{ik}$ or $O_{ik} \prec_i O_{ij}$
3. For all $O_{ij} \in OS_i$, $O_{ij} \prec_i N_i$

Example

Consider a transaction T:

Read(x)

Read(y)

$x \leftarrow x + y$

Write(x)

Commit

Then

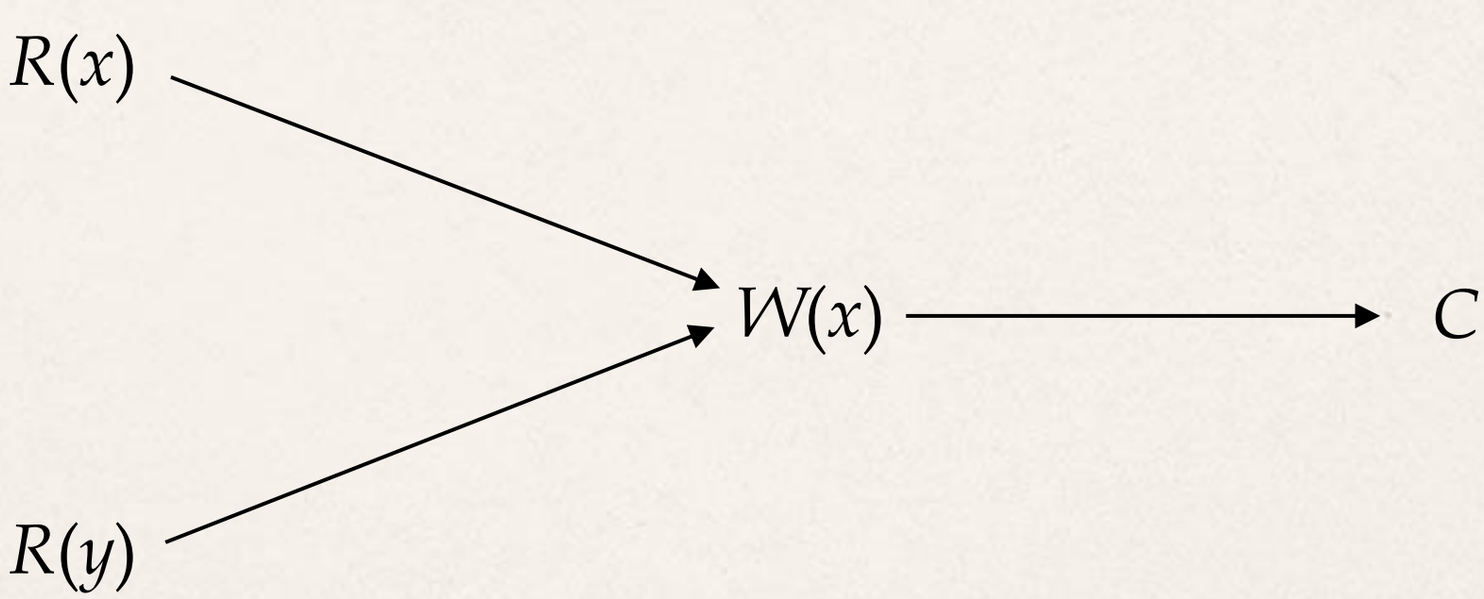
$\Sigma = \{R(x), R(y), W(x), C\}$

$\prec = \{(R(x), W(x)), (R(y), W(x)), (W(x), C), (R(x), C), (R(y), C)\}$

DAG Representation

Assume

$\prec = \{(R(x),W(x)), (R(y),W(x)), (W(x), C), (R(x), C), (R(y), C)\}$



Properties of Transactions

ATOMICITY

- all or nothing

CONSISTENCY

- no violation of integrity constraints

ISOLATION

- concurrent changes invisible

DURABILITY

- committed updates persist

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

Isolation

- Concurrent changes invisible (incomplete results)
  - An executing transaction cannot reveal its results to other concurrent transactions before its commitment.
  - Necessary to avoid cascading aborts.
- Serializability
  - If several transactions are executed concurrently, the results must be the same as if they were executed serially in some order (correctness criterion).

- Concurrent changes invisible $\Rightarrow$ serializability

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:

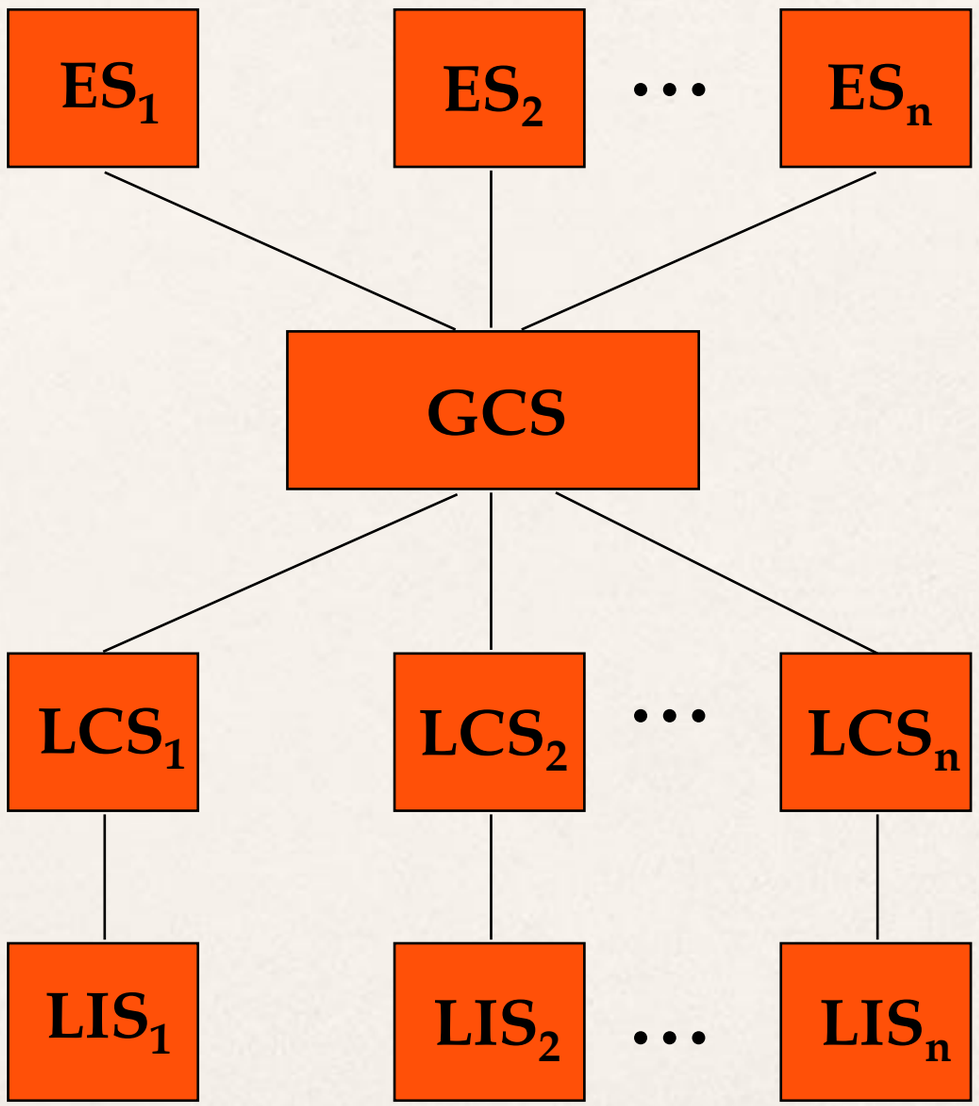
T_1:	Read(x)	
T_1:	$x \leftarrow x$+1	
T_1:	Write(x)	
T_1:	Commit	
T_2:	Read(x)	
T_2:	$x \leftarrow x$+1	
T_2:	Write(x)	
T_2:	Commit	

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	

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

Distributed Database Reference Architecture



External schemas

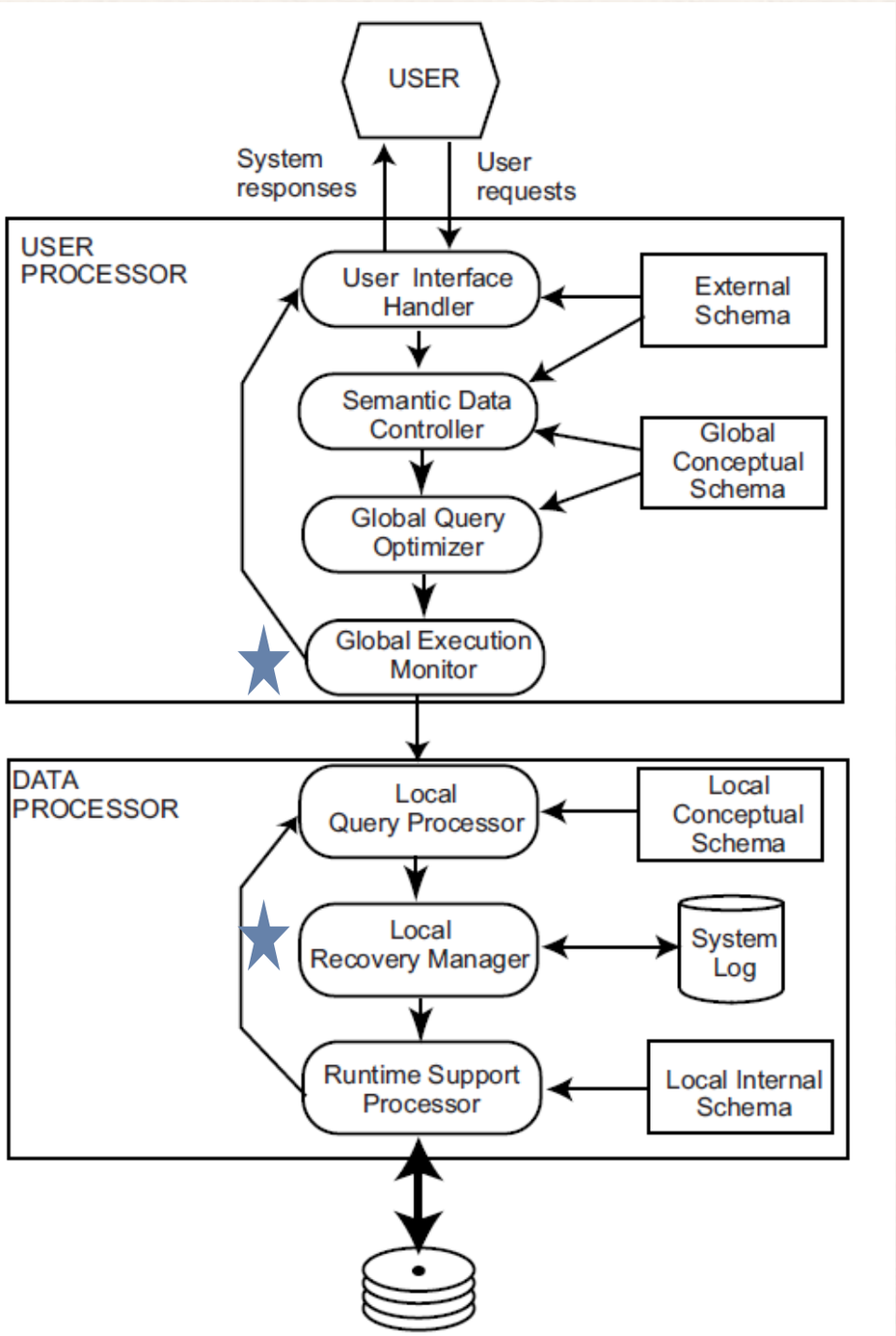
Global conceptual schema

Local conceptual schema

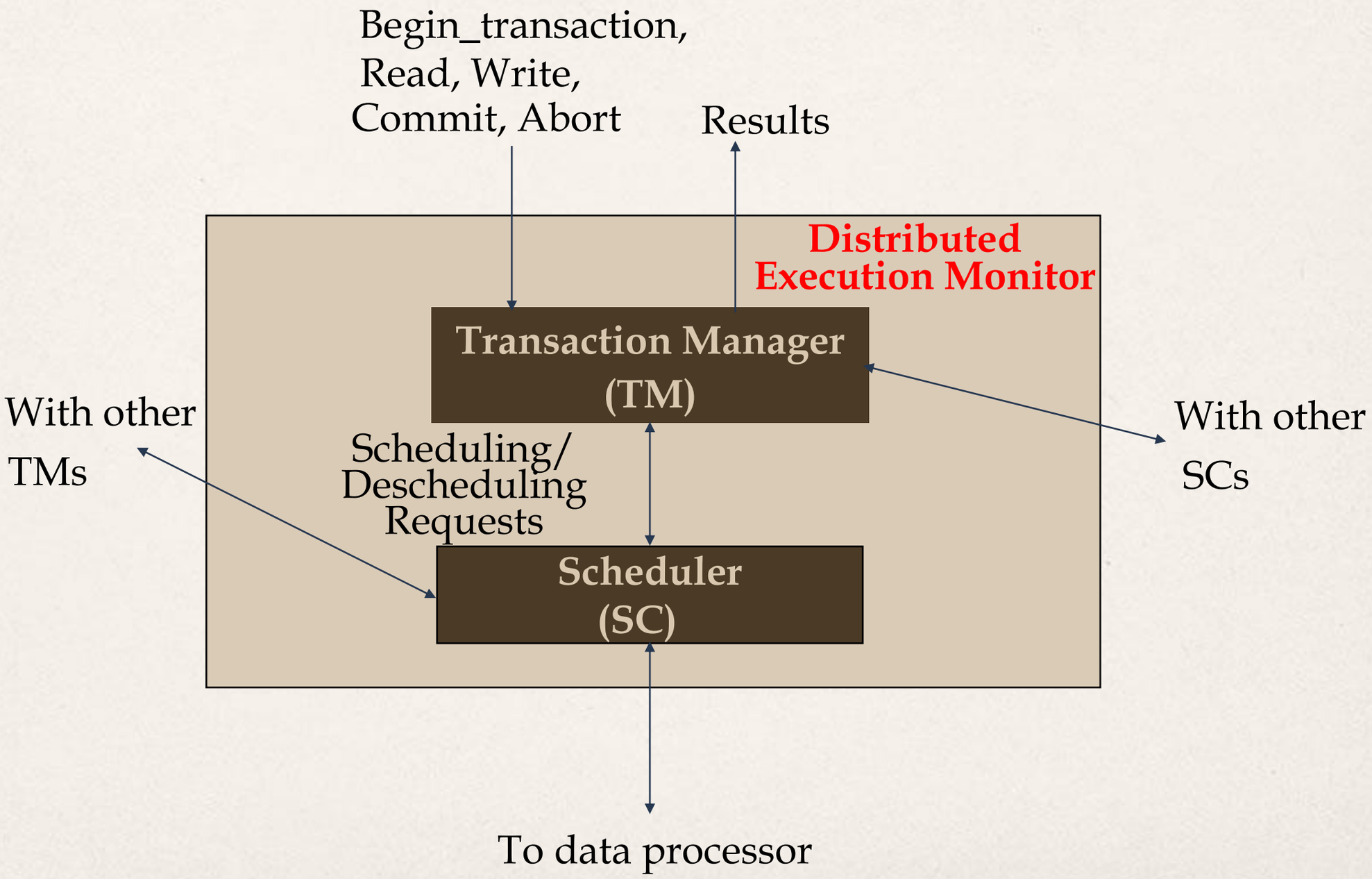
Local internal schema

Data organizational view

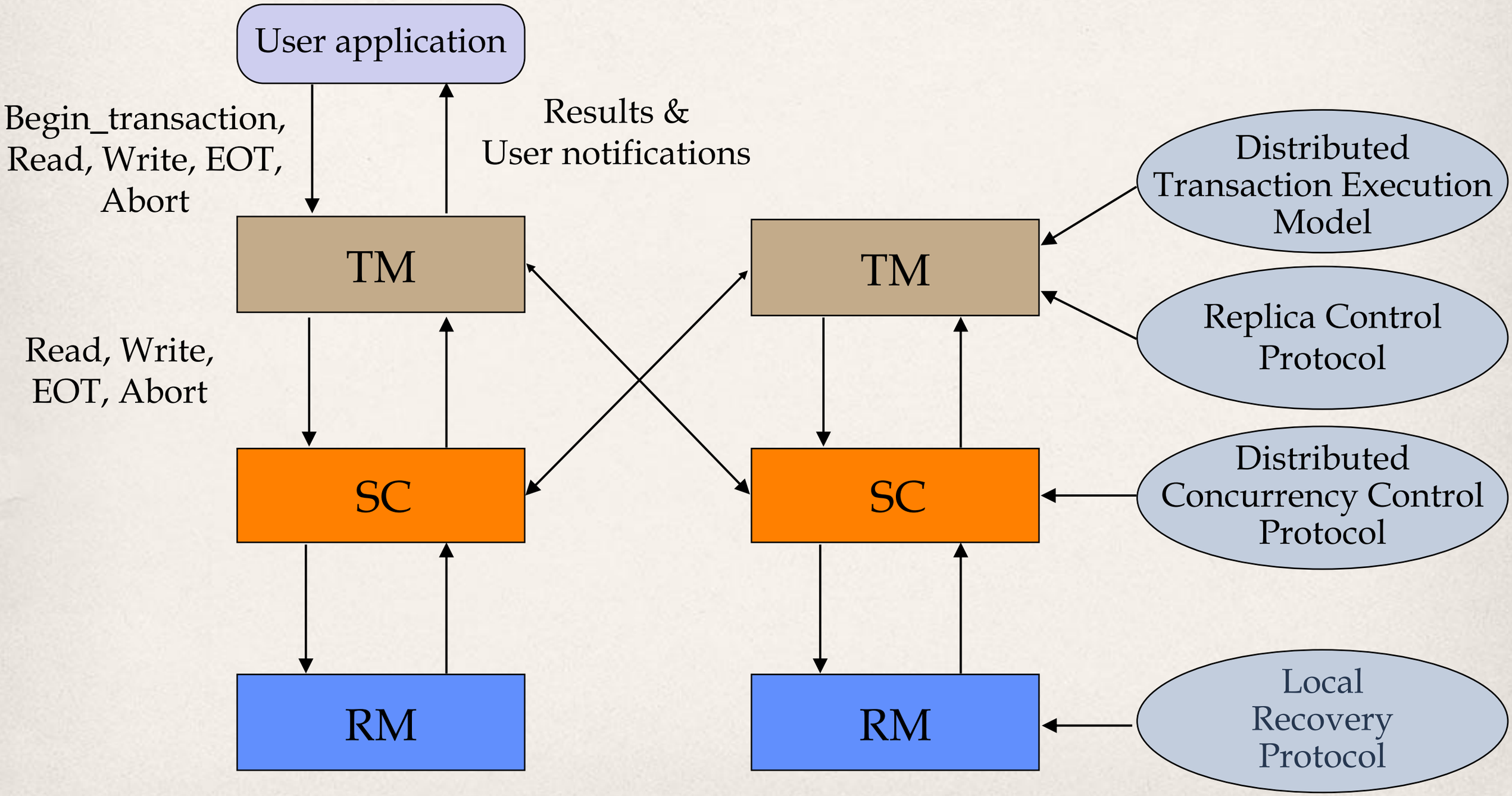
Components of a Distributed DBMS



Architecture Revisited



Distributed Transaction Execution



Distributed Transaction Management

Transaction Concepts and Models

Consistency Degrees

- 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.

✦ Dirty data refers to data values that have been updated by a transaction prior to its commitment

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 1
  - *T* does not overwrite dirty data of other transactions
  - *T* does not commit any writes before EOT
- Degree 0
  - Transaction *T* does not overwrite dirty data of other transactions

SQL-92 Isolation Levels

- Defined on the basis of what ANSI call phenomena:
  - Dirty read
    - T_1 modifies x which is then read by T_2 before T_1 terminates. If T_1 aborts, T_2 has read a value which never exists in the database.
  - Non-repeatable (fuzzy) read
    - T_1 reads x; T_2 then modifies or deletes x and commits. If T_1 tries to read x again, it reads a different value or it 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)

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

Types of Transactions

- Based on
  - Timing
    - On-line (short-life) vs batch (long-life)
  - Organization of read and write actions
    - General
    - Two-step
    - Restricted
    - Action model
  - **Structure**
    - Flat (or simple) transactions
    - Nested transactions

Transaction Structure

- Flat transaction
  - Consists of a sequence of primitive operations embraced between **begin** and **end** markers.

Begin_transaction Reservation
…
end.

Transaction Structure

- Nested transaction
  - The operations of a transaction may themselves be transactions.

Begin_transaction Reservation
begin
  Begin_transaction Airline
    …
  end. {Airline}
  Begin_transaction Hotel
    …
  end. {Hotel}
end. {Reservation}

Transactions Provide…

- *Atomic* and *reliable* execution in the presence of failures
- *Correct* execution in the presence of multiple user accesses
- Correct management of *replicas* (if they support it)