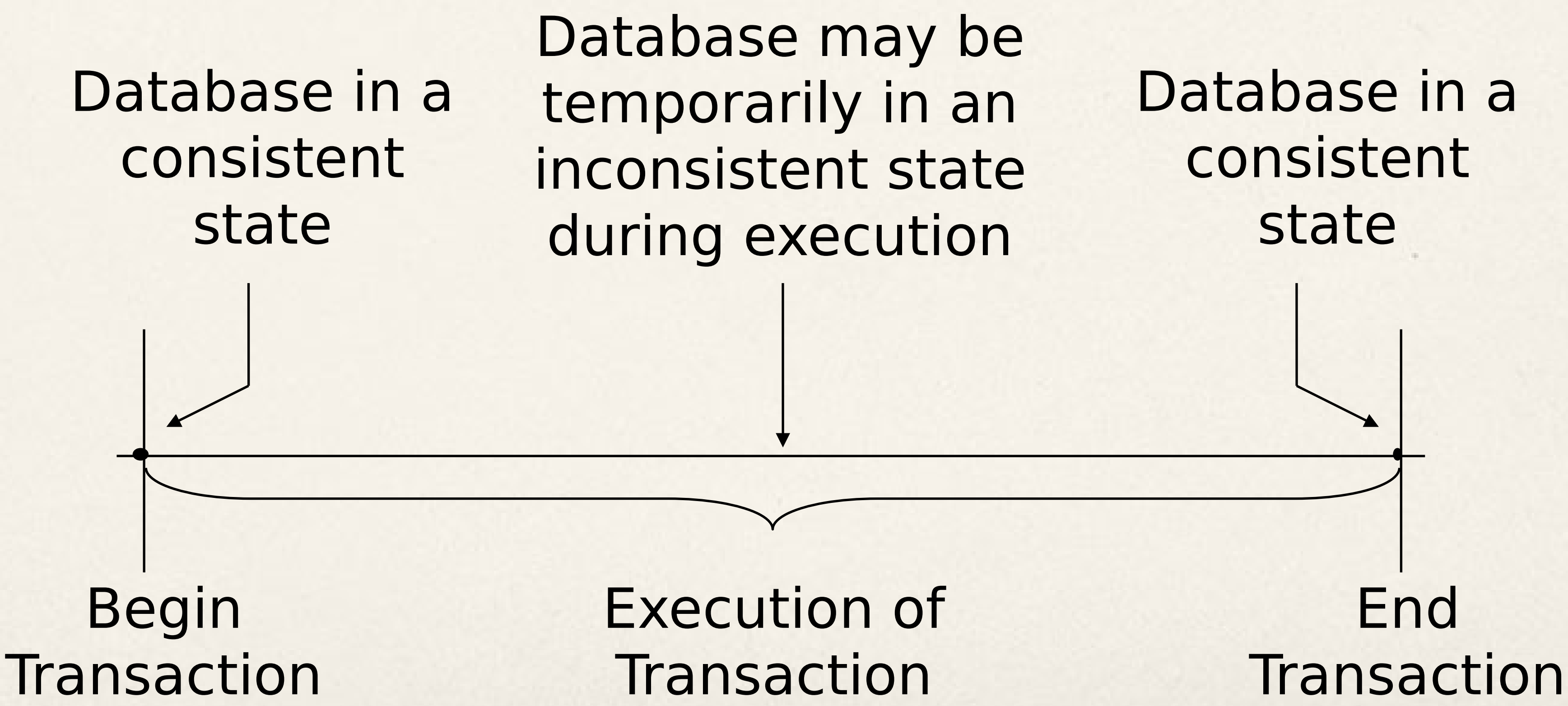
Outline

- Introduction
- Background
- Distributed Database Design
- Database Integration
- Semantic Data Control
- Distributed Query Processing
- Multidatabase Query Processing
- Distributed Transaction Management
  - Transaction Concepts and Models
  - Distributed Concurrency Control
  - Distributed Reliability
- Data Replication
- Parallel Database Systems
- Distributed Object DBMS
- Peer-to-Peer Data Management
- Web Data Management
- Current Issues

Transaction

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

- concurrency transparency
- failure transparency



© M. T. Özsu & P. Valduriez

Transaction Example – A Simple SQL Query

```
Transaction   BUDGET_UPDATE
begin
EXEC SQL UPDATE PROJ
        SET   BUDGET = BUDGET∗1.1
        WHERE PNAME = "CAD/CAM"
end.
```


© M. T. Özsu & P. Valduriez

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)

Example Transaction – SQL Version

```
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}
```


© M. T. Özsu & P. Valduriez

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}
```

Example Transaction – Reads & Writes

```
Begin_transaction Reservation
begin
input(flight_no, date, customer_name);
temp ← Read(flight_no(date).stsold);
if temp = flight(date).cap then
begin
output("no free seats");
Abort
end
else begin
Write(flight(date).stsold, temp + 1);
Write(flight(date).cname, customer_name);
Write(flight(date).special, null);
Commit;
output("reservation completed")
end

end. {Reservation}
```

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

$$RS[\text{Reservation}] = \{\text{FLIGHT.STSOLD, FLIGHT.CAP}\}$$

$$WS[\text{Reservation}] = \{\text{FLIGHT.STSOLD, FC.FNO, FC.DATE, FC.CNAME, FC.SPECIAL}\}$$

$$BS[\text{Reservation}] = \{\text{FLIGHT.STSOLD, FLIGHT.CAP, FC.FNO, FC.DATE, FC.CNAME, FC.SPECIAL}\}$$

© M. T. Özsu & P. Valduriez

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 $T_i = \{\Sigma_i, \prec_i\}$ where

❶ $\Sigma_i = OS_i \cup \{N_i\}$

❷ For any two operations O_{ij} , $O_{ik} \in OS_i$, if $O_{ij} = R(x)|W(x)$ and $O_{ik} = W(x)$ for any data item x, then either $O_{ij} \prec_i O_{ik}$ or $O_{ik} \prec_i O_{ij}$

❸ $O_{ij} \in OS_i, O_{ij} \prec_i N_i$

© M. T. Özsu & P. Valduriez

Example

Consider a transaction T:

Read(x)
Read(y)
$x \leftarrow x + y$
Write(x)
Commit

Then

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

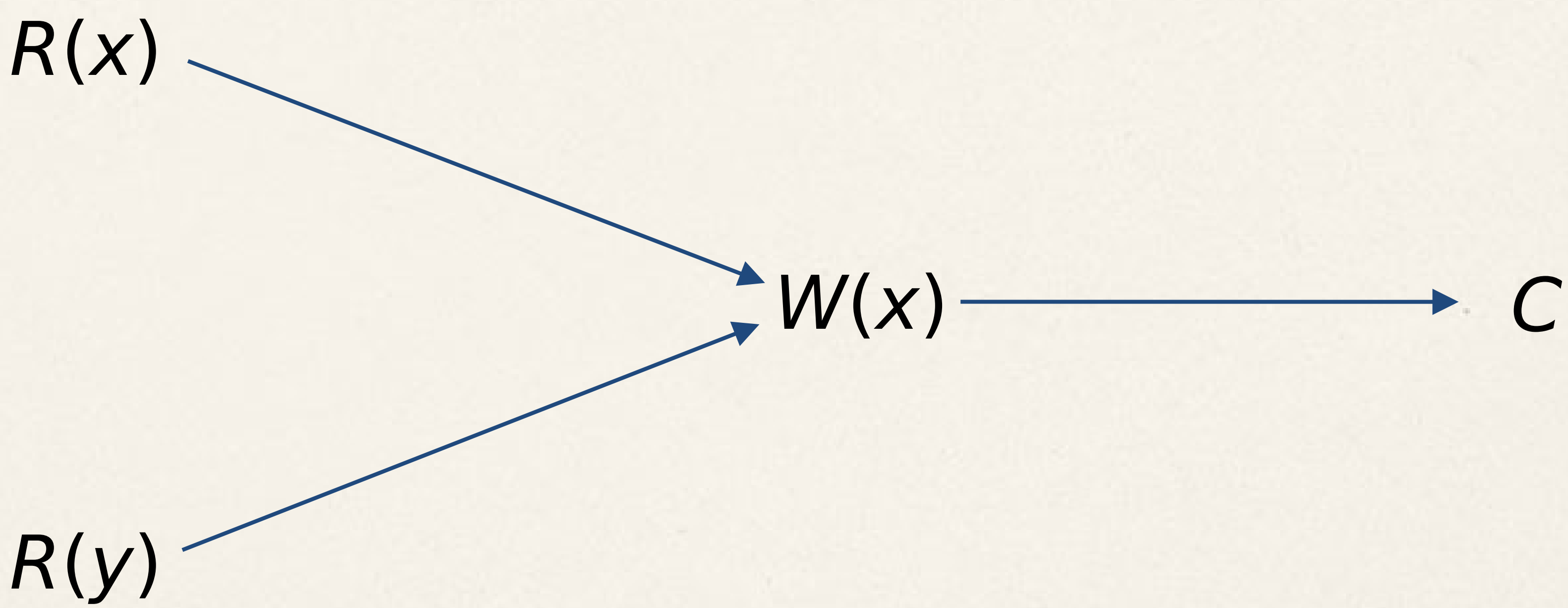
$\prec = \{(R(x), W(x)), (R(y), W(x)), (W(x), C), (R(x), C), (R(y), C)\}$

© M. T. Özsu & P. Valduriez

DAG Representation

Assume

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



Principles of Transactions

ATOMICITY

all or nothing

CONSISTENCY

no violation of integrity constraints

ISOLATION

concurrent changes invisible ⇒ serializable

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.

© M. T. Özsu & P. Valduriez

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

© M. T. Özsu & P. Valduriez

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.

© M. T. Özsu & P. Valduriez

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.

© M. T. Özsu & P. Valduriez

Isolation Example

- Consider the following two transactions:

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

- Possible execution sequences:

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

© M. T. Özsu & P. Valduriez

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.

$$\ldots, W_1(x), \ldots, R_2(x), \ldots, C_1(\text{or } A_1), \ldots, C_2(\text{or } A_2)$$

or

$$\ldots, W_1(x), \ldots, R_2(x), \ldots, C_2(\text{or } A_2), \ldots, C_1(\text{or } A_1)$$

© M. T. Özsu & P. Valduriez

SQL-92 Isolation Levels

Phenomena:

- 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.

$$\ldots, R_1(x), \ldots, W_2(x), \ldots, C_1(\text{or } A_1), \ldots, C_2(\text{or } A_2)$$

or

$$\ldots, R_1(x), \ldots, W_2(x), \ldots, C_2(\text{or } A_2), \ldots, C_1(\text{or } A_1)$$

© M. T. Özsu & P. Valduriez

SQL-92 Isolation Levels

Phenomena:

- Phantom

  T_1 searches the database according to a predicate while T_2 inserts new tuples that satisfy the predicate.

$$\ldots, R_1(P), \ldots, W_2(y \text{ in } P), \ldots, C_1(\text{or } A_1), \ldots, C_2(\text{or } A_2)$$

or

$$\ldots, R_1(P), \ldots, W_2(y \text{ in } P), \ldots, C_2(\text{ or } A_2), \ldots, C_1(\text{or } A_1)$$

© M. T. Özsu & P. Valduriez

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.

© M. T. Özsu & P. Valduriez

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

© M. T. Özsu & P. Valduriez

Characterization of Transactions

- Based on
  - Application areas
    - Non-distributed vs. distributed
    - Compensating transactions
    - Heterogeneous transactions
  - Timing
    - On-line (short-life) vs batch (long-life)
  - Organization of read and write actions
    - Two-step
    - Restricted
    - Action model
  - Structure
    - Flat (or simple) transactions
    - Nested transactions
    - Workflows

General:

$$T_1 : \{R(x), R(y), W(y), R(z), W(x), W(z), W(w), C\}$$

Two-step:

$$T_2 : \{R(x), R(y), R(z), W(x), W(z), W(y), W(w), C\}$$

Restricted:

$$T_3 : \{R(x), R(y), W(y), R(z), W(x), W(z), R(w), W(w), C\}$$

$$T_4 : \{R(x), R(y), R(z), R(w), W(x), W(z), W(y), W(w), C\}$$

Action:

$$T_5 : \{[R(x), W(x)], [R(y), W(y)], [R(z), W(z)], [R(w), W(w)], C\}$$

© M. T. Özsu & P. Valduriez

Transaction Structure

- Flat transaction

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

Begin_transaction Reservation
…
end.

- Nested transaction

  The operations of a transaction may themselves be transactions.

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

© M. T. Özsu & P. Valduriez

Nested Transactions

- Have the same properties as their parents ☂ may themselves have other nested transactions.
- Introduces concurrency control and recovery concepts to within the transaction.
- Types
  - Closed nesting
    - Subtransactions begin after their parents and finish before them.
    - Commitment of a subtransaction is conditional upon the commitment of the parent (commitment through the root).
  - Open nesting
    - Subtransactions can execute and commit independently.
    - Compensation may be necessary.

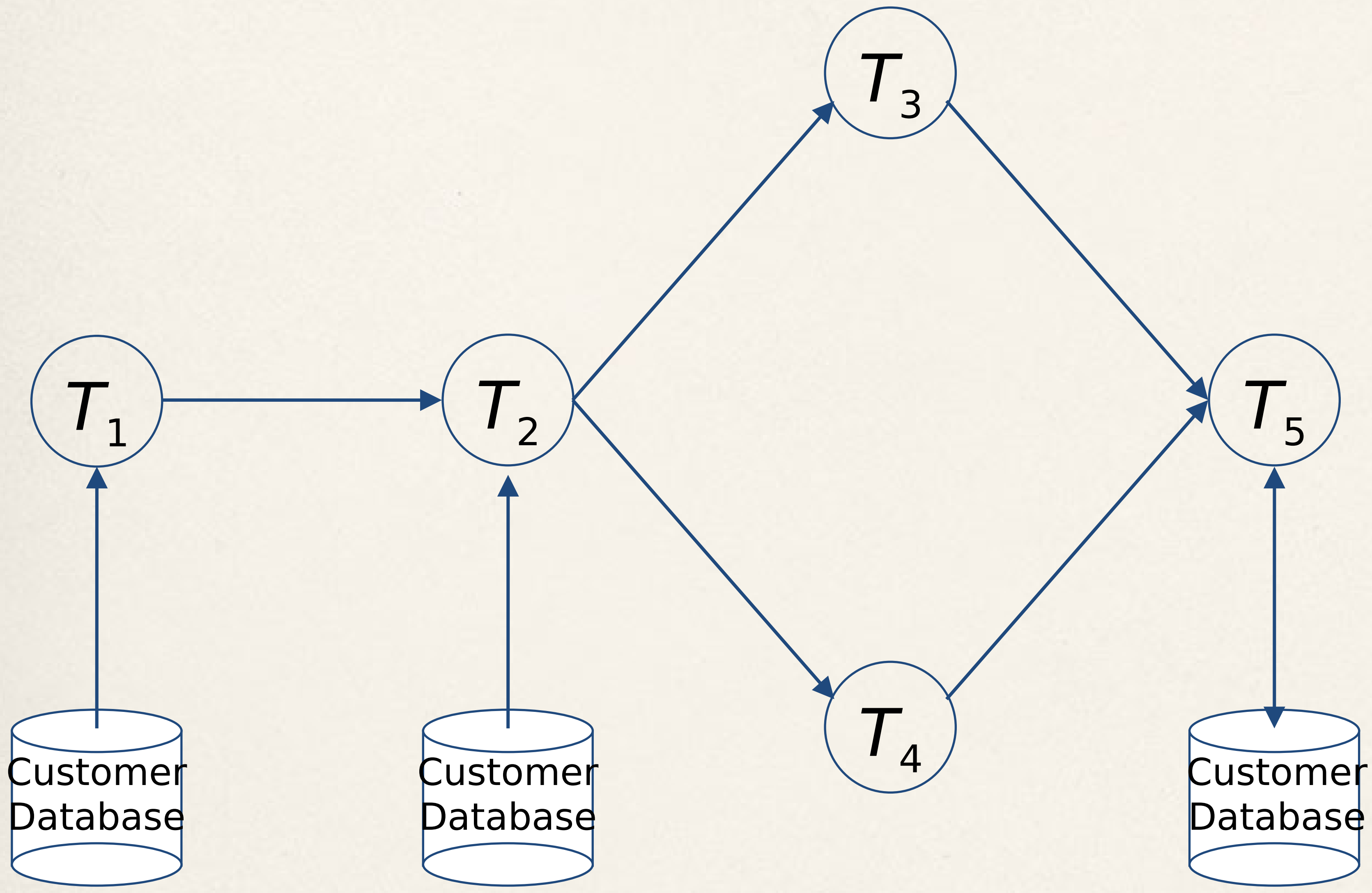
© M. T. Özsu & P. Valduriez

Workflows

- "A collection of tasks organized to accomplish some business process."
- Types
  - Human-oriented workflows
    - ✦ Involve humans in performing the tasks.
    - ✦ System support for collaboration and coordination; but no system-wide consistency definition
  - System-oriented workflows
    - ✦ Computation-intensive & specialized tasks that can be executed by a computer
    - ✦ System support for concurrency control and recovery, automatic task execution, notification, etc.
  - Transactional workflows
    - ✦ In between the previous two; may involve humans, require access to heterogeneous, autonomous and/or distributed systems, and support selective use of ACID properties

© M. T. Özsu & P. Valduriez

Workflow Example



T_1: Customer request obtained

T_2: Airline reservation performed

T_3: Hotel reservation performed

T_4: Auto reservation performed

T_5: Bill generated

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)

© M. T. Özsu & P. Valduriez

Transaction Processing Issues

- Transaction structure (usually called transaction model)
  - Flat (simple), nested
- Internal database consistency
  - Semantic data control (integrity enforcement) algorithms
- Reliability protocols
  - Atomicity & Durability
  - Local recovery protocols
  - Global commit protocols

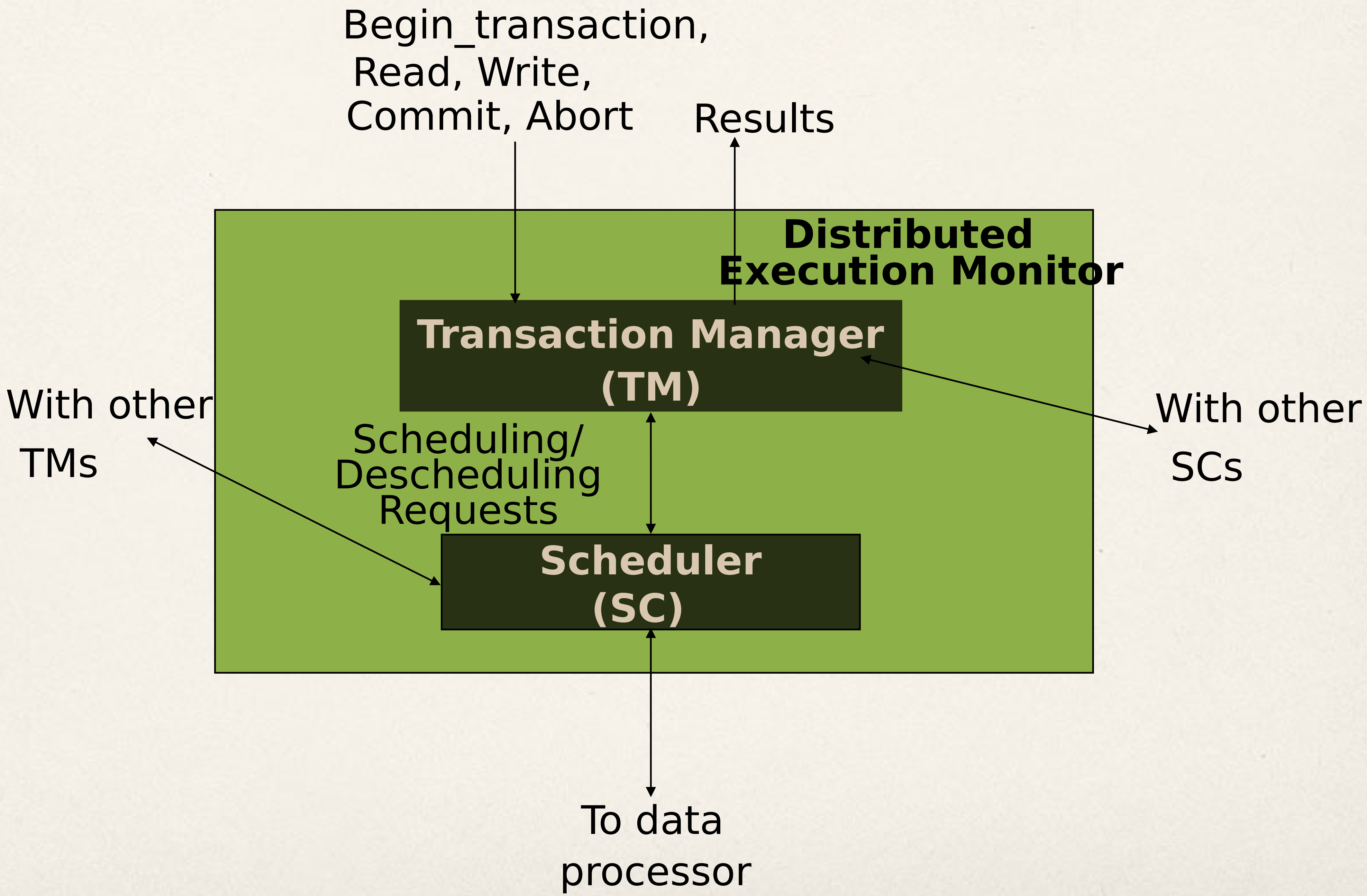
© M. T. Özsu & P. Valduriez

Transaction Processing Issues

- Concurrency control algorithms
  - How to synchronize concurrent transaction executions (correctness criterion)
  - Intra-transaction consistency, Isolation
- Replica control protocols
  - How to control the mutual consistency of replicated data
  - One copy equivalence and ROWA

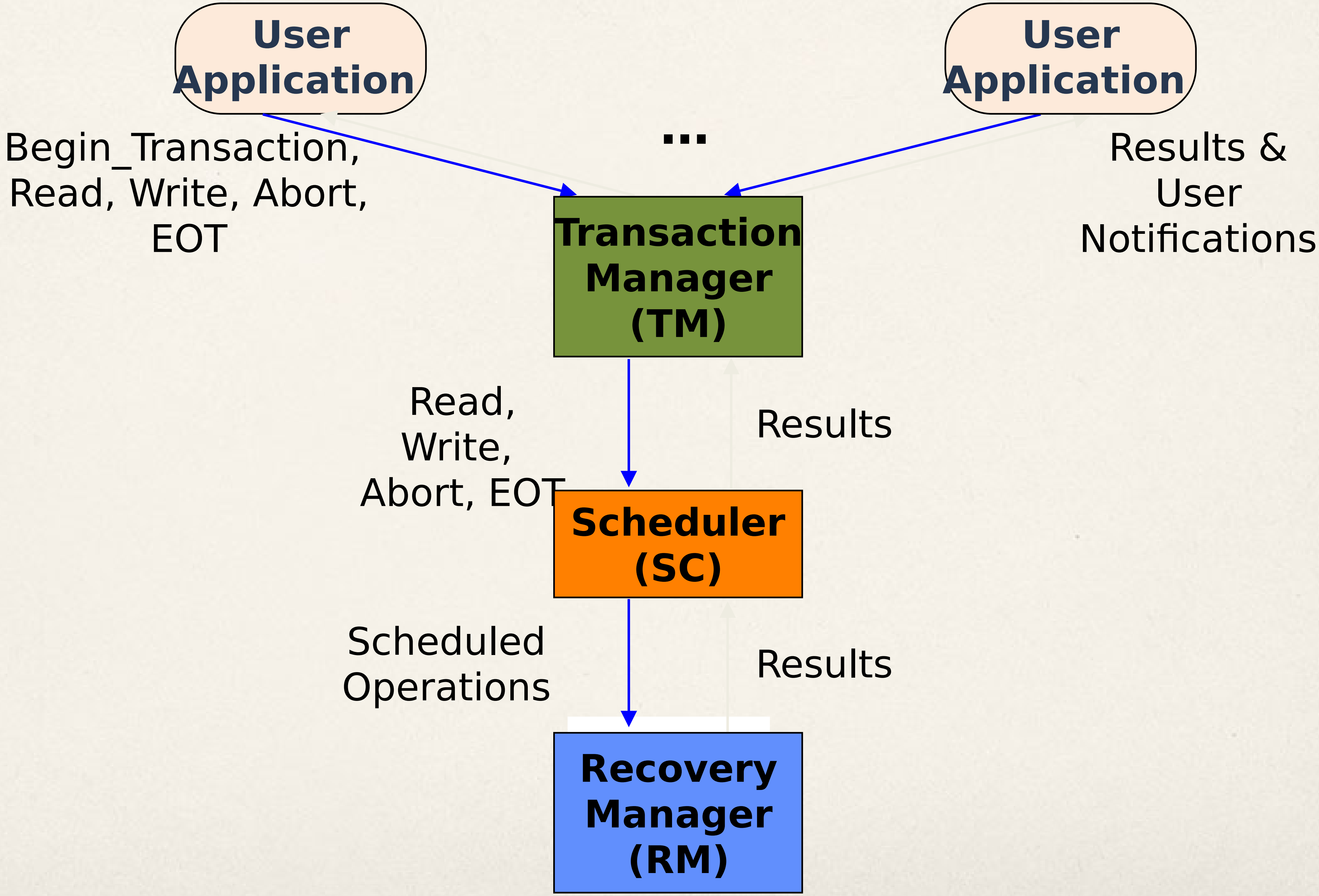
© M. T. Özsu & P. Valduriez

Architecture Revisited



© M. T. Özsu & P. Valduriez

Centralized Transaction Execution



© M. T. Özsu & P. Valduriez

Distributed Transaction Execution