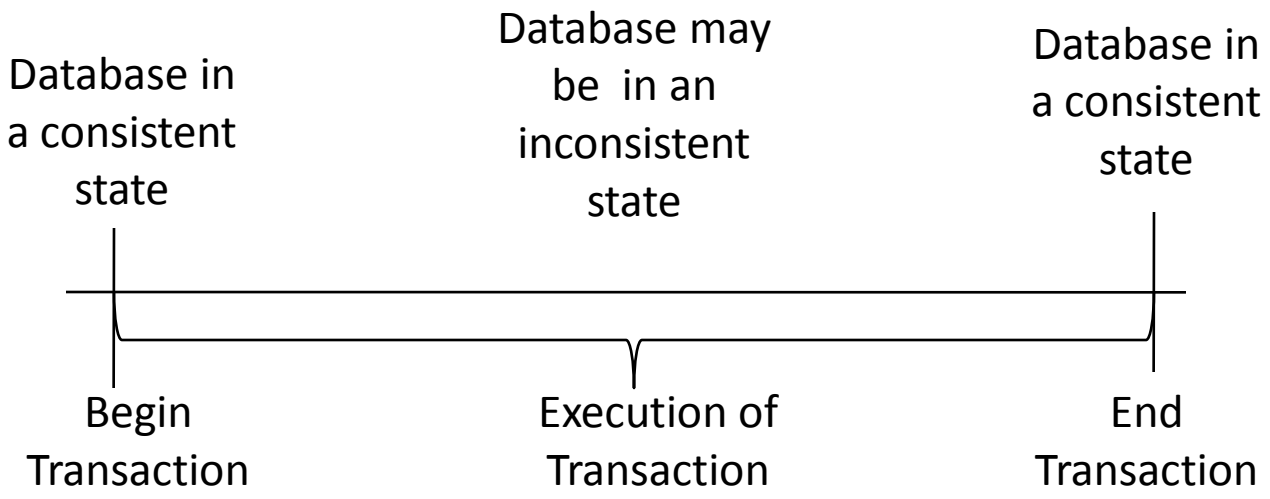
DISTRIBUTED TRANSACTION MANAGEMENT

UNIT-2

SUB-UNIT(2.2)

Transaction

- A transaction is a collection of actions that make consistent transformations of system states while preserving system consistency.
- It explains
  - ➢Concurrency transparency
  - ➢Failure transparency



Transaction Example (Simple SQL Query)

Transaction BUDGET_UPDATE

begin

EXEC SQL UPDATE PROJ

SET BUDGET = BUDGET∗1.1

WHERE PNAME = “CAD/CAM”

end

Transaction Example (Simple SQL Query)

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

```
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 ofTransactions

```
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 UPDATEFLIGHT
                        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 of Transaction

```
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.
```

Properties of Transaction

- **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**.

Consistency

- The internal consistency is maintained.
- A transaction which executes *alone against a consistent database leaves it in a consistent state.*
- Since transactions are correct programs they do not violate database integrity constraints.

Degree of consistency

- **Degree 0**
- **Degree 1**
- **Degree 2**
- **Degree 3**

Degree of Consistency

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(End of transaction).*

Degree of Consistency

Degree 2

- Transaction *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 doesn't commit any writes before EOT.*
- *T doesn't read dirty data from other transactions.*
- Other transactions do not get dirty by 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.

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.

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}

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

Transaction processing Issues

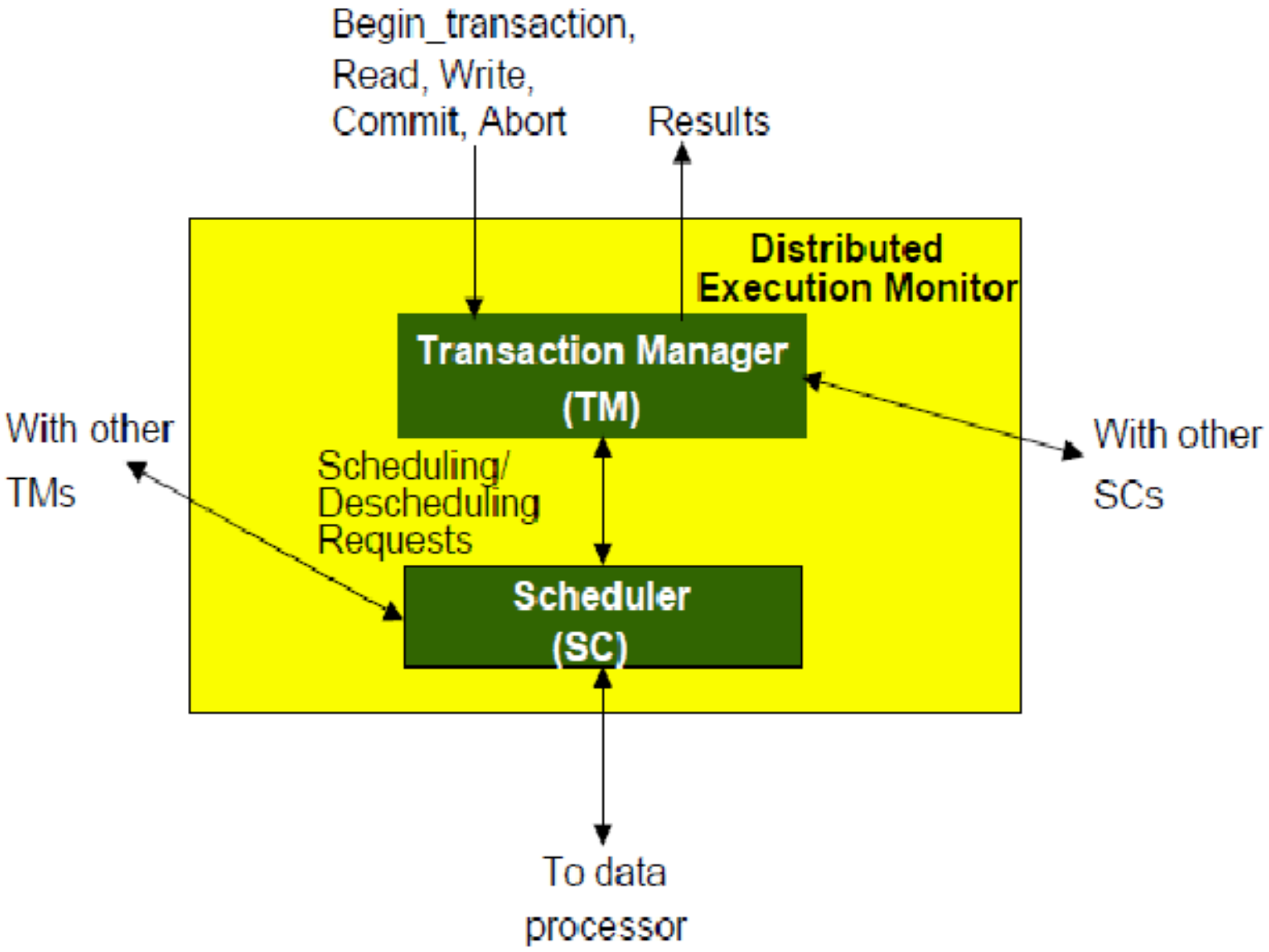
Concurrency control algorithms

- How to synchronize concurrent transaction executions (correctness criterion).
- Intra-transaction consistency, Isolation

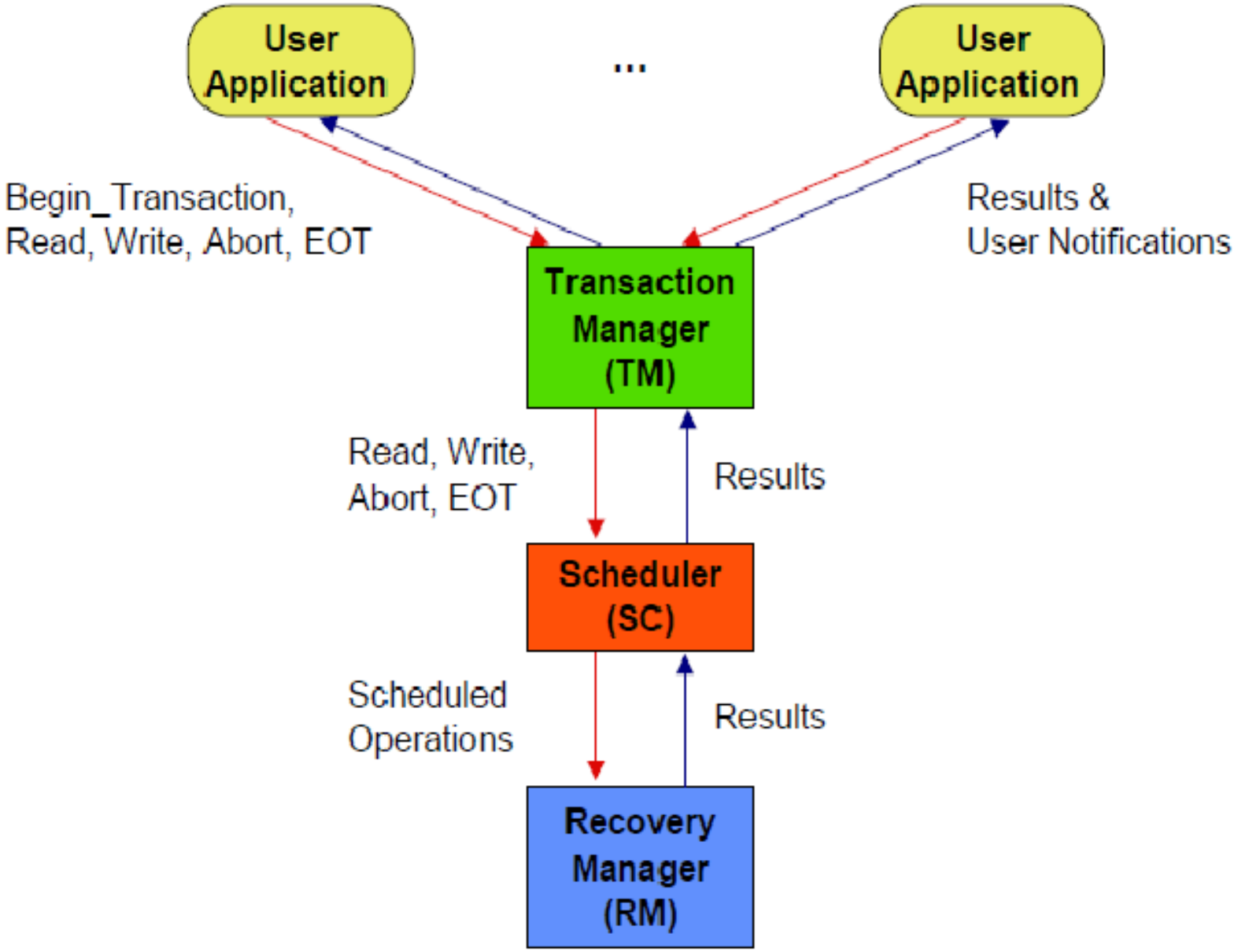
Replica control protocols

- How to control the mutual consistency of replicated data
- One copy equivalence and ROWA(Read over Write Aceess).

Transaction architecture



Centralized Transaction



Distributed Database Transaction