PARUL INSTITUTEOF ENGINEERING & TECHNOLOGY

FACULTY OF ENGINEERING & TECHNOLOGY

PARUL UNIVERSITY

Database Management System

Unit 6 : Transaction

Dr. Vishwanath

Parul® University

NAAC A++ ACCREDITED UNIVERSITY

DIGITAL LEARNING CONTENT

Outline

- Transaction
- Database Recovery
- Concurrency Control
- Deadlock

Transaction

Transaction in DBMS refers to the sequence of operations we perform on the database that is treated as a single unit of work.

- Two main issues to deal with:
  - Failures of various kinds, such as hardware failures and system crashes
  - Concurrent execution of multiple transactions

Transaction

Lets consider a standard example of an ATM

- Transaction is started
- Insert your ATM card.
- Select the language according to your preference.
- Select the option of savings account.
- Enter the amount you need to withdraw.
- Enter your pin secretly.
- Wait for the processing time.
- Collect the cash from the machine.
- Transaction completed.

Transaction

In DBMS transaction, there are mainly three main operations:

- Read
- Write
- Commit

Transaction

Suppose, we have to transfer Rs. 100 from account A to account B.

Initially account has Rs. 600 and B has Rs. 900.

```
R(A) -- 600      // Accessed from the RAM.
A = A-100        // Deducting Rs 100 from A.
W(A)--500        // Updated in RAM.
R(B) -- 900      // Accessed from RAM.
B=B+100          // 100₹ is added to B's Account.
W(B) --1000      // Updated in RAM.
commit           // The data in RAM is taken back to Hard Disk.
```

Transaction

In order to maintain consistency in a database, before and after the transaction, certain properties are followed.

These are called **ACID** properties.

Atomicity

It means, either the entire transaction takes place at once or doesn't happen at all.

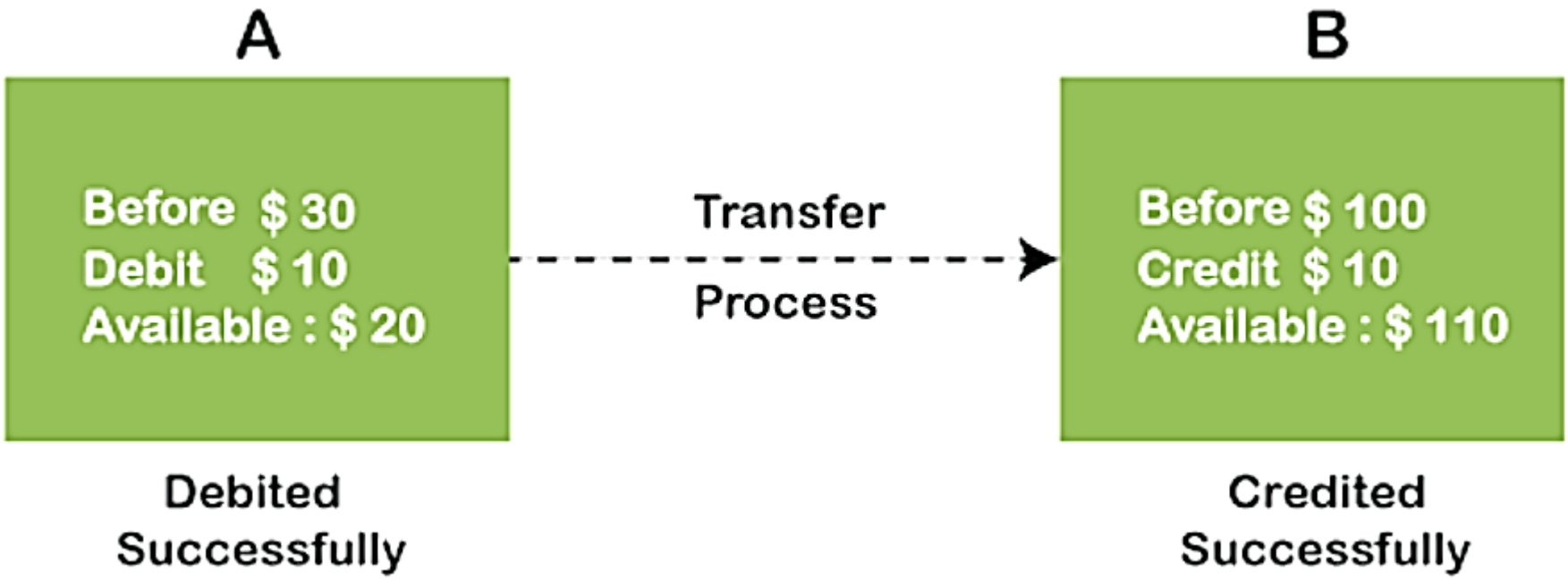
It involves the following two operations.

— **Abort** : If a transaction aborts, changes made to the database are not visible.

— **Commit** : If a transaction commits, changes made are visible.

Atomicity

If account A having $30 in his account from which he wishes to send $10 to account B.



Consistency

It refers to the correctness of a database.

This means that integrity constraints must be maintained so that the database is consistent before and after the transaction.

The total amount before and after the transaction must be maintained.
Total **before T** occurs = **30 +100 = 130** .
Total **after T occurs** = **20 + 110 = 130** .
Therefore, the database is **consistent** .

Isolation

This property ensures that multiple transactions can occur concurrently without leading to the inconsistency of the database state.

It means, if two operations are being performed on two different databases, they may not affect the value of one another.

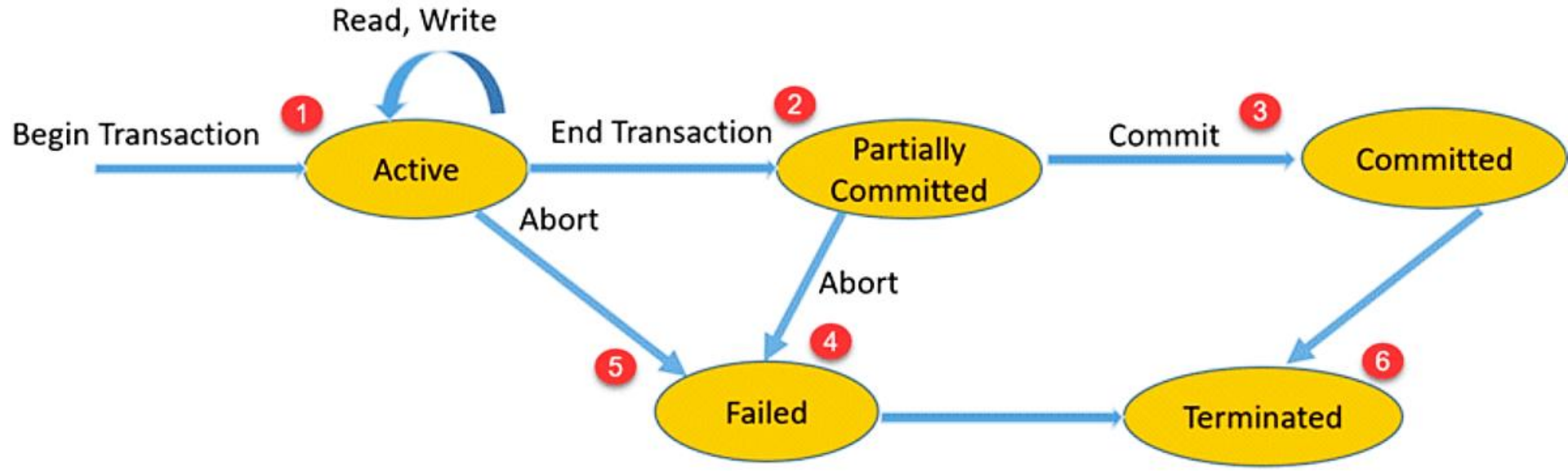
Durability

Durability ensures that the data after the successful execution of the operation becomes permanently saved in the database.

The durability of the data should be so perfect that even if the system fails or leads to a crash, the database still survives.

- ✓ Therefore, the ACID property of DBMS plays a vital role in maintaining the consistency and availability of data in the database.

Transaction Life Cycle



State	Transaction type
Active State	A transaction enters into an active state when the execution process begins. During this state read or write operations can be performed.

Transaction Life Cycle

State	Transaction type
Partially Committed	A transaction goes into the partially committed state after the end of a transaction.
Committed State	When the transaction is committed to state, it has already completed its execution successfully. Moreover, all of its changes are recorded to the database permanently.
Failed State	A transaction considered failed, when any one of the checks fails or if the transaction is aborted while it is in the active state.
Terminated State	State of transaction reaches terminated state, when certain transactions which are leaving the system can't be restarted.

Transaction

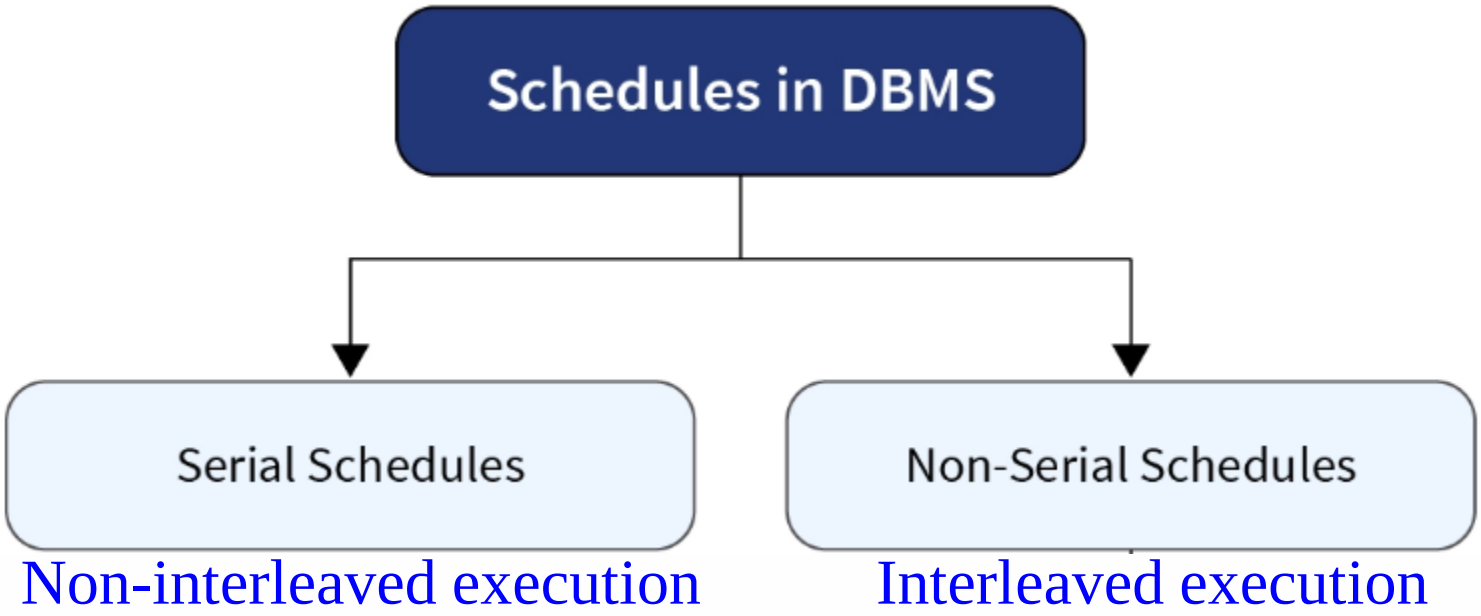
- ✓ Transactions serve as the backbone of data management in DBMS, providing a framework for reliable and consistent database operations.
- ✓ Maintaining ACID properties ensures that transactions execute accurately, remain isolated from each other, and persist despite system failures.
- ✓ Examples: online banking transactions, e-commerce platforms, and airline reservation systems.

Scheduling

When multiple transactions are running concurrently, then a sequence is needed in which the operations are to be performed because at a time, only one operation can be performed on the database.

This sequence of operations is known as **Schedule**, and this process is known as **Scheduling**.

Scheduling


Schedules in DBMS
Serial Schedules
Non-Serial Schedules
Non-interleaved execution
Interleaved execution

Serial Scheduling

Here, all the transactions are executed serially one after the other.

In serial Schedule, a transaction does not start execution until the currently running transaction finishes execution.

A serial schedule always gives the correct result.

Transaction T1	Transaction T2
R(A)	
W(A)	
R(B)	
W(B)	
commit	
	R(A)
	W(B)
	commit

Non-Serial Scheduling

Here, multiple transactions execute concurrently but in non-serial manner .

In the Non-Serial Schedule, the other transaction proceeds without the completion of the previous transaction.

A serial schedule always gives the correct result.

Transaction T1	Transaction T2
R1(A)	
W1(A)	
	R2(A)
	W2(A)
R1(B)	
W1(B)	
	R2(B)
	W2(B)

Serializability

In DBMS, all the transaction should be arranged in a particular order, even if all the transaction is concurrent.

If all the transaction is not serializable, then it produces the incorrect result.

Types of Serializability:

1. Conflict serializability
2. View serializability

Conflict Serializability

A schedule is called conflict serializability if after swapping of non-conflicting operations, it can transform into a serial schedule.

The schedule will be a conflict serializable if it is conflict equivalent to a serial schedule.

The two operations become conflicting if all conditions satisfy:

1. Both belong to separate transactions.
2. They have the same data item.
3. They contain at least one write operation.

Conflict Serializability

Example:

Swapping is possible only if S1 and S2 are logically equal.

1. T1: Read(A) T2: Read(A)

T1	T2
Read(A)	
	Read(A)

Schedule S1

Swapped

T1	T2
	Read(A)
Read(A)	

Schedule S2

Here, S1 = S2. That means it is non-conflict.

2. T1: Read(A) T2: Write(A)

T1	T2
Read(A)	
	Write(A)

Schedule S1

Swapped

T1	T2
	Write(A)
Read(A)	

Schedule S2

Here, S1 ≠ S2. That means it is conflict.

View Serializability

Here, each transaction should produce some result and these results are the output of proper sequential execution of the data item

- Unlike conflict serialized, the view serializability focuses on preventing inconsistency in the database.

Condition:

1. Schedules must have same set of transaction.
2. schedules should not have the same type of read or write operation.
3. both schedules should not have the same conflict; Order of execution of the same data item

View Serializability

Example:

T1	T2
Read(A)	
	Write(A)

Schedule S1

T1	T2
	Write(A)
Read(A)	

Schedule S2

Above two schedules are view equivalent because Initial read operation in S1 is done by T1 and in S2 it is also done by T1.

Serializability

Benefits of Serializability:

1. Predictable execution
2. Easier to Reason about & Debug
3. Reduced Costs
4. Increased Performance

DBMS transactions must follow the ACID properties to be considered serializable.

Two-phase commit protocol

The two-phase commit protocol is a set of actions used to make sure that an application program makes all changes to the collection of resources.

This protocol verifies the **all-or-no changes,** even if the application program, the system or a resource manager fails.

The protocol involves two phases:

1. One-phase Commit,
2. Two-phase Commit

One-Phase Commit

Phase 1 — Prepare Phase

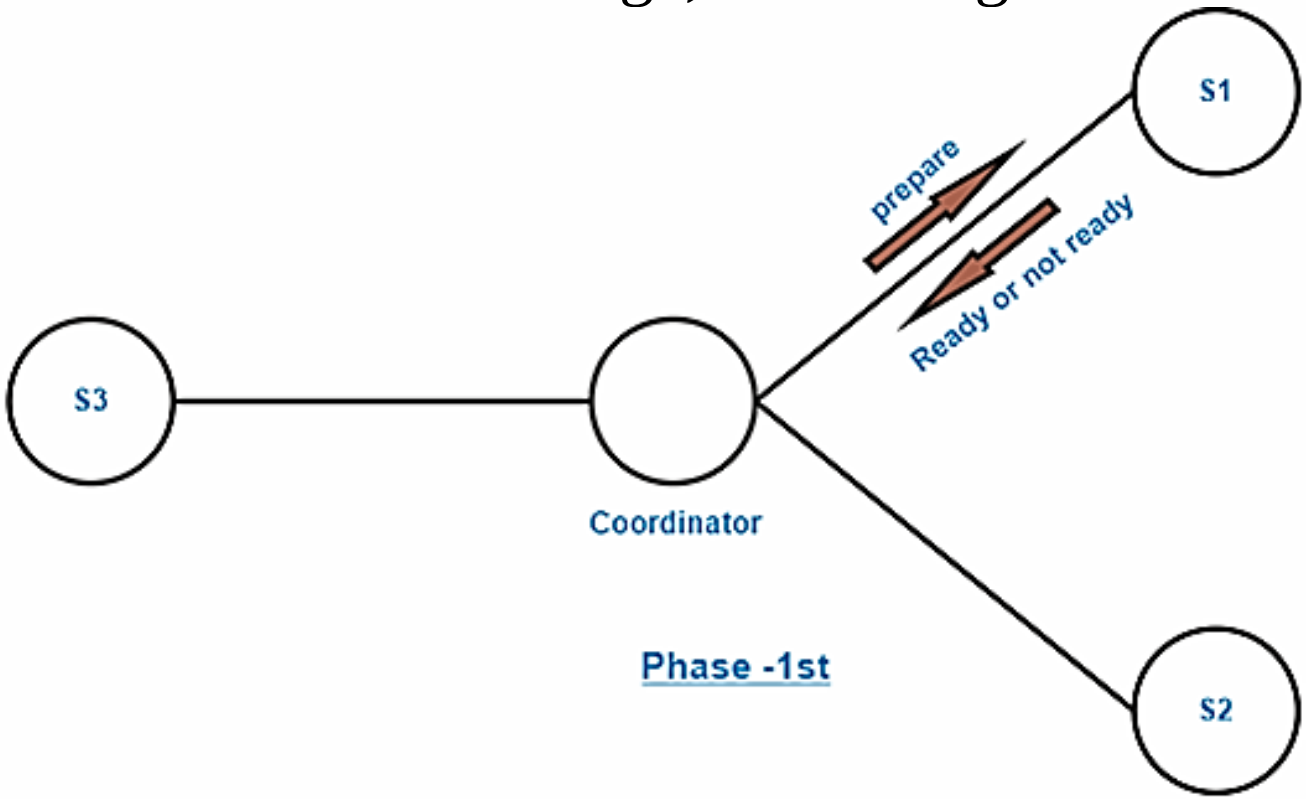
Consider the scenario where the transaction is being carried out at a controlling site and several slave sites.

These are the steps followed in the one-phase commit protocol:

1. The coordinator node sends a prepare message to all participating nodes, asking them if they are ready to commit the transaction.

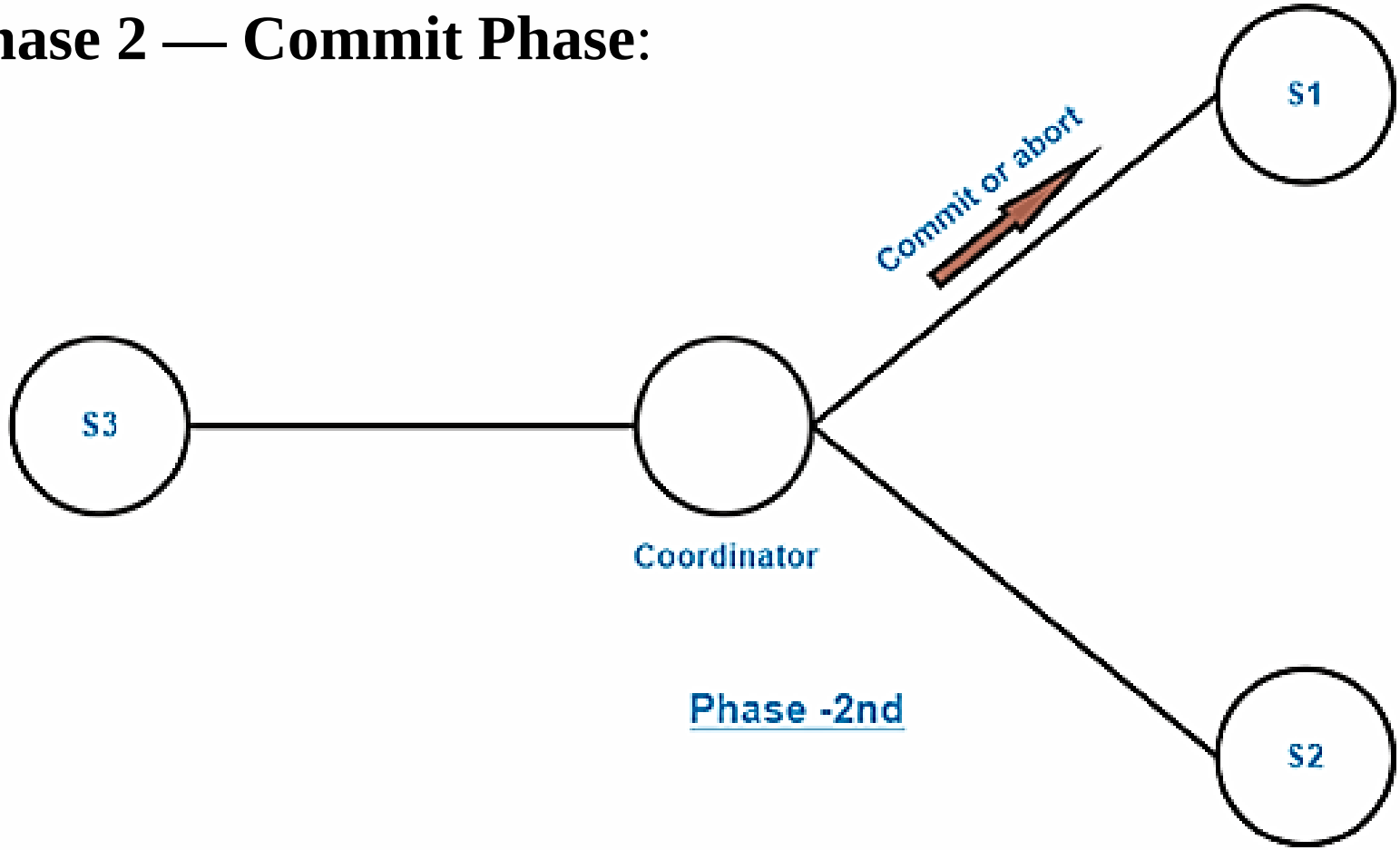
One-Phase Commit

2. Each participant acquires a "lock" on the resource/s and replies with either a Yes or No message, indicating whether they can commit.



Two-Phase Commit

Phase 2 — Commit Phase:



Two-Phase Commit

1. The coordinator decides whether to commit or abort the transaction based on the responses received in the Prepare phase.

2. If all participants have responded with a Yes message, the coordinator sends a commit message to all the participants.

3. If any participant has responded with a No message, the coordinator sends an abort message to all the participants, and the transaction is rolled back.

Two-Phase Commit

Advantages:

Consistency

Atomicity

Simplicity

Disadvantages:

Scalability

Single Point of Failure

Performance

Database Recovery

DBMS is a highly complex system with hundreds of transactions being executed every second.

The durability and robustness of a DBMS depends on its complex architecture and its underlying hardware and system software.

If it fails or crashes amid transactions, it is expected that the system would follow some sort of algorithm or techniques to recover lost data.

Database Recovery

To see where the problem has occurred, we generalize a failure into various categories, as follows:

- Transaction failure
- System Crash
- Disk Failure

Recovery and Atomicity:

- When a system crashes, it may have several transactions being executed and various files opened for them to modify the data items.

Database Recovery

Transactions are made of various operations, which are atomic in nature.

But according to ACID properties of DBMS, atomicity of transactions as a whole must be maintained, that is, either all the operations are executed or none.

Database Recovery

When a DBMS recovers from a crash, it should maintain the following −

- It should check the states of all the transactions, which were being executed.
- A transaction may be in the middle of some operation.
- It should check whether the transaction can be completed now or it needs to be rolled back.
- No transactions would be allowed to leave the DBMS in an inconsistent state.

Database Recovery

Techniques of database recovery:

- Maintaining the logs of each transaction, and writing them onto some stable storage before actually modifying the database.

- Maintaining shadow paging, where the changes are done on a volatile memory, and later, the actual database is updated.

Recovery: Log-based recovery

Log is a sequence of records, which maintains the records of actions performed by a transaction.

It is important that the logs are written prior to the actual modification and stored on a stable storage media

Log-based recovery works as follows −

• The log file is kept on a stable storage media.

• When a transaction enters the system and starts execution, it writes a log about it.

Recovery: Log-based recovery

Let's assume there is a transaction to modify the City of a student. The following logs are written for this transaction.

- When the transaction is initiated, then it writes 'start' log.

1. <Tn, Start>

- When the transaction modifies the City from 'Noida' to 'Bangalore', then another log is written to the file.

2. <Tn, City, 'Noida', 'Bangalore' >

Recovery: Log-based recovery

- When the transaction is finished, then it writes another log to indicate the end of the transaction.

3. <Tn, Commit>

When the system is crashed, then the system checks the log to find which transactions need to be undone and which need to be redone.

1. If the log contains the record <Tn, Start> and <Tn, Commit> then the **Transaction Tn** needs to be **done**.

Recovery: Log-based recovery

2. If log contains record<T_n, Start> but does not contain the record either <Tn, commit> or <Tn, abort>

then the **Transaction Tn** needs to be **undone**.

Database Recovery

Recovery with Concurrent Transactions:

When more than one transaction are being executed in parallel, the logs are interleaved.

At the time of recovery, it would become hard for the recovery system to backtrack all logs, and then start recovering.

To ease this situation, most modern DBMS use the concept of '**checkpoints**'.

Recovery: Checkpoints

Checkpoint

Keeping and maintaining logs in real time may fill out all the memory space available in the system and the log file may grow too big to be handled.

- It may be possible that Log Based Recovery consumes more time while recovering the data using Logs. Thus, to reduce the time consumption, we can use Checkpoints.

Recovery: Checkpoints

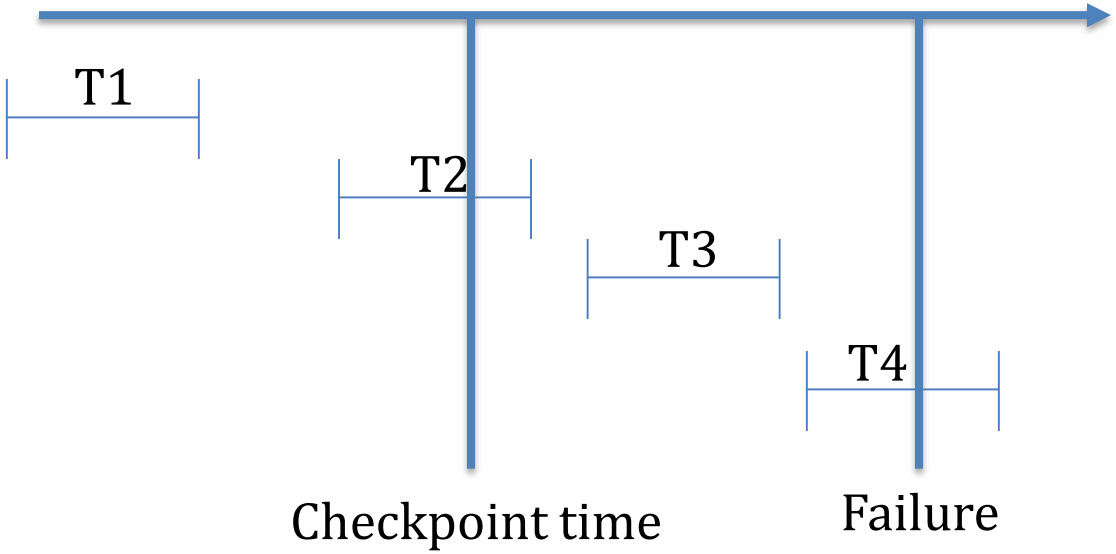
Checkpoint

Checkpoint declares a point before which the DBMS was in consistent state, and all the transactions were committed.

When a system with concurrent transactions crashes and recovers, it behaves in the following manner:

Recovery: Checkpoints

Checkpoint



Working of Checkpoints in case of Failure

- ✓ T1 will be ignored as it is committed before checkpoint.
- ✓ T2 and T3 will restart again as they were active even after checkpoint, but committed before Failure.
- ✓ T4 will be completely ignored as it was active even after the checkpoint and has not committed.

Recovery: Checkpoints

The recovery system reads the logs backwards from the end to the last checkpoint.

It maintains two lists, an undo-list and a redo-list.

If the recovery system sees a log with <Tn, Start> and <Tn, Commit> or just <Tn, Commit>, it puts the transaction in the **redo-list**.

If the recovery system sees a log with <Tn, Start> but no commit or abort log found, it puts the transaction in **undo-list**.

Recovery: Shadow Paging

Shadow paging is one of the techniques that is used to recover transaction/ database from failure.

- The database is fragmented into fixed sized blocks referred as **PAGES**. Each pages are stored in a **Page Table**.

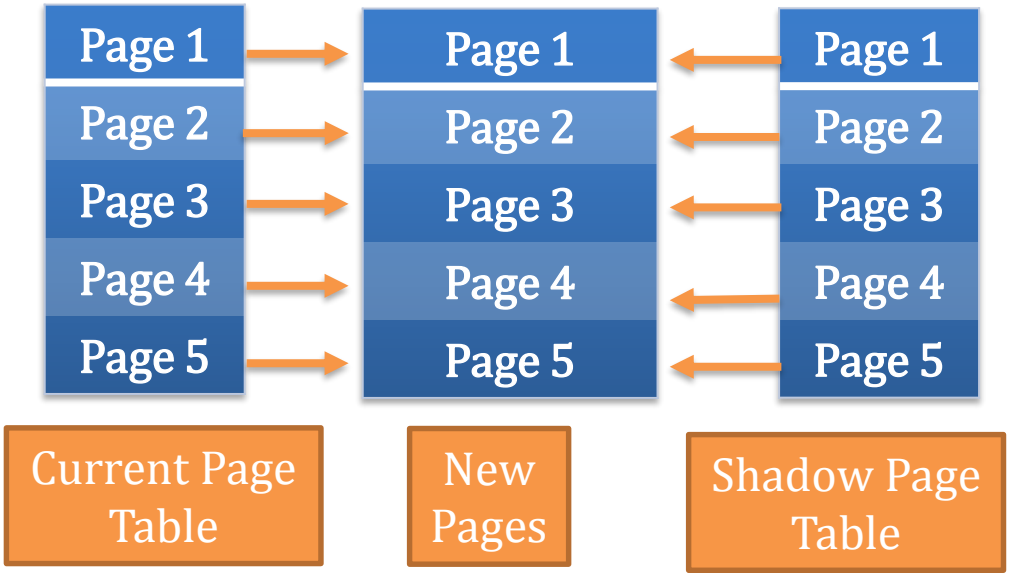
- Shadow Paging maintains 2 Page Tables:
  - ✓ Current Page Table
  - ✓ Shadow Page Table

Recovery: Shadow Paging

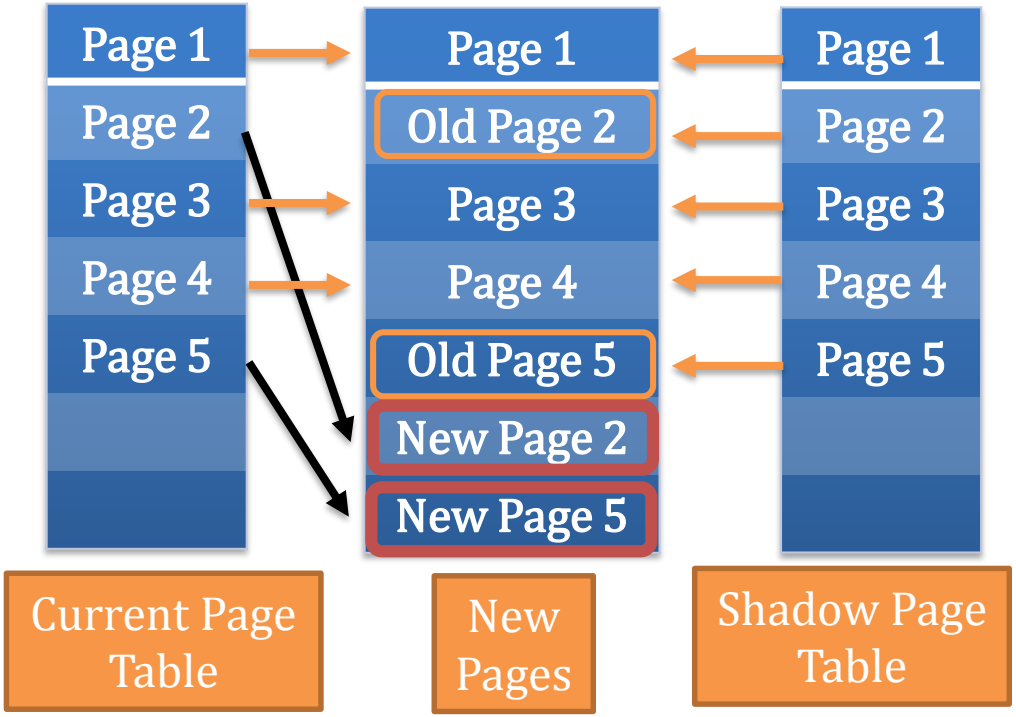
- When the transaction starts, both the tables are identical.
- During transaction, only Current Page Table changes, Shadow Page Table never changes.

- All the operations are performed on Current Page Table.
- When a database page is updated:
  - ✓ A New Copy of the page is created
  - ✓ Current Page Table points to the New Copy.
  - ✓ Then, the updates are performed.

Recovery: Shadow Paging



- ✓ When the transaction starts, both Current Page Table and Shadow Page Table points to Same Pages in New Pages



- ✓ When the transaction updates Page 2 and Page 5, then a new copy of the Page 2 and 5 is created in New Pages.
- ✓ Current Page Table points to New Copy and Shadow Page Table continues pointing Old Copy
- ✓ Thus, in case of Failure, Shadow Page Table can be used to recover the data as it keeps pointing Old Pages.

Recovery: Shadow Paging

The advantages of shadow paging are as follows −

- No need for log records.
- No undo/ Redo algorithm.
- Recovery is faster.

The disadvantages of shadow paging are as follows −

- Data is fragmented or scattered.
- Garbage collection problem.
- Concurrent transactions are difficult to execute.

Concurrency Control

Concurrency control is the procedure for managing simultaneous operations without conflicting with each other.

- Concurrency control is used to address conflicts which occur with a multi-user system who have access to perform READ and WRITE operation in database.
- If T1 conflicts with T2 over a data item A, then the existing concurrency control decides if T1 or T2 should get the A and if the other transaction is rolled-back or waits.
- Resolve read-write and write-write conflicts.
- Apply isolation through mutual exclusion between conflicting transactions.

Concurrency Control

- Concurrency control helps to ensure serializability.

- Allowing Concurrent execution of operations may lead to following problems:
  - ✓ **Lost Update:** if two transactions T1 and T2 both read the same data and then update it; in this case, first update will be overwritten by the second update.
  - ✓ **Dirty Read:** when one transaction update some item and then fails. Moreover, this updated item is accessed by another transaction before it is rolled back to its initial value.
  - ✓ **Incorrect Retrieval:** when one transaction accesses data to use it in other operation; but before it can use, another transaction updates that data and commits.

Concurrency Control: Lock Based Protocol

A lock is a variable associated with data item to control concurrent access to that data item.

- Data items can be locked in 2 ways:
  - ✓ **Exclusive (X) mode**: Data item can be both read as well as write. X-lock is requested using **lock-X** instruction.
  - ✓ **Shared (S) mode**: Data item can only be read. S-lock is requested using **lock-S** instruction.

Transaction can proceed only after request is granted.

Concurrency Control: Lock Based Protocol

- A transaction may be granted a lock on an item if the requested lock is compatible with locks already held on the item by other transactions
- Any number of transactions can hold shared locks on an item,
  - ✓ but if any transaction holds an exclusive on the item no other transaction may hold any lock on the item.
- If a lock cannot be granted, the requesting transaction is made to wait till all incompatible locks held by other transactions have been released.

Concurrency Control: Lock Based Protocol

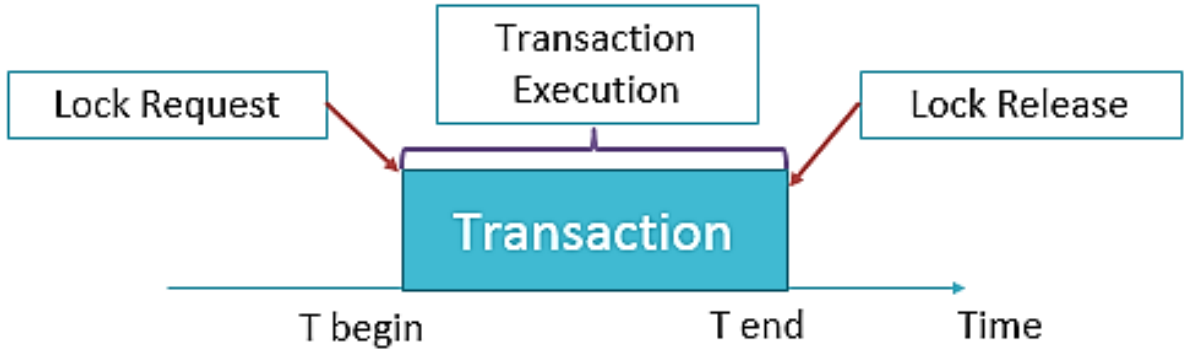
- **Example of Transactions using Locking.**

```
T2: lock-S(A);
    read (A);
    unlock(A);
    lock-S(B);
    read (B);
    unlock(B);
    display(A+B)
```

- A **locking protocol** is a set of rules followed by all transactions while requesting and releasing locks

Concurrency Control: Lock Based Protocol

- The execution phase of transaction can be described as following:
  - ✓ When execution of transaction starts, create a list of data items and type of lock, it needs and request for that lock.
  - ✓ When all the locks are granted, transaction continues execution of its operation.
  - ✓ As soon as the transaction releases its first lock, it cannot demand for any lock; but can only release the acquired locks.



Concurrency Control: Two Phase Lock Protocol

- The Two Phase Lock protocol has 2 phases:
  - ✓ **Growing Phase:**
    - transaction may obtain locks
    - transaction may not release locks
  - ✓ **Shrinking Phase:**
    - transaction may release locks
    - transaction may not obtain locks
- It ensures serialized transactions based on lock points (the point where a transaction acquired its final lock).

Concurrency Control: Two Phase Lock Protocol

- The Two Phase Lock Conversions are as follows:
  - ✓ **Growing Phase:**
    - transaction receive Lock - S
    - transaction receive Lock - X
    - transaction convert Lock - S to Lock - X
  - ✓ **Shrinking Phase:**
    - transaction releases Lock - S
    - transaction releases Lock - X
    - transaction convert Lock - X to Lock – S and release
- It ensures serialized transactions.

Concurrency Control: Two Phase Lock Protocol

- The Two Phase Lock can be of 2 types:

✓ **Strict two phase locking protocol:**
- A transaction may release all the shared locks after the Lock Point, but cannot release any of the exclusive locks until the transaction commits or aborts.

✓ **Rigorous two phase locking protocol:**
- A transaction is not allowed to release any lock (either shared or exclusive) until it commits.

Concurrency Control: Multiple Granularity

Granularity is the size of the data item allowed to lock.

Multiple Granularity means hierarchically breaking up the database into blocks that can be locked and can be tracked what needs to lock and in what way.

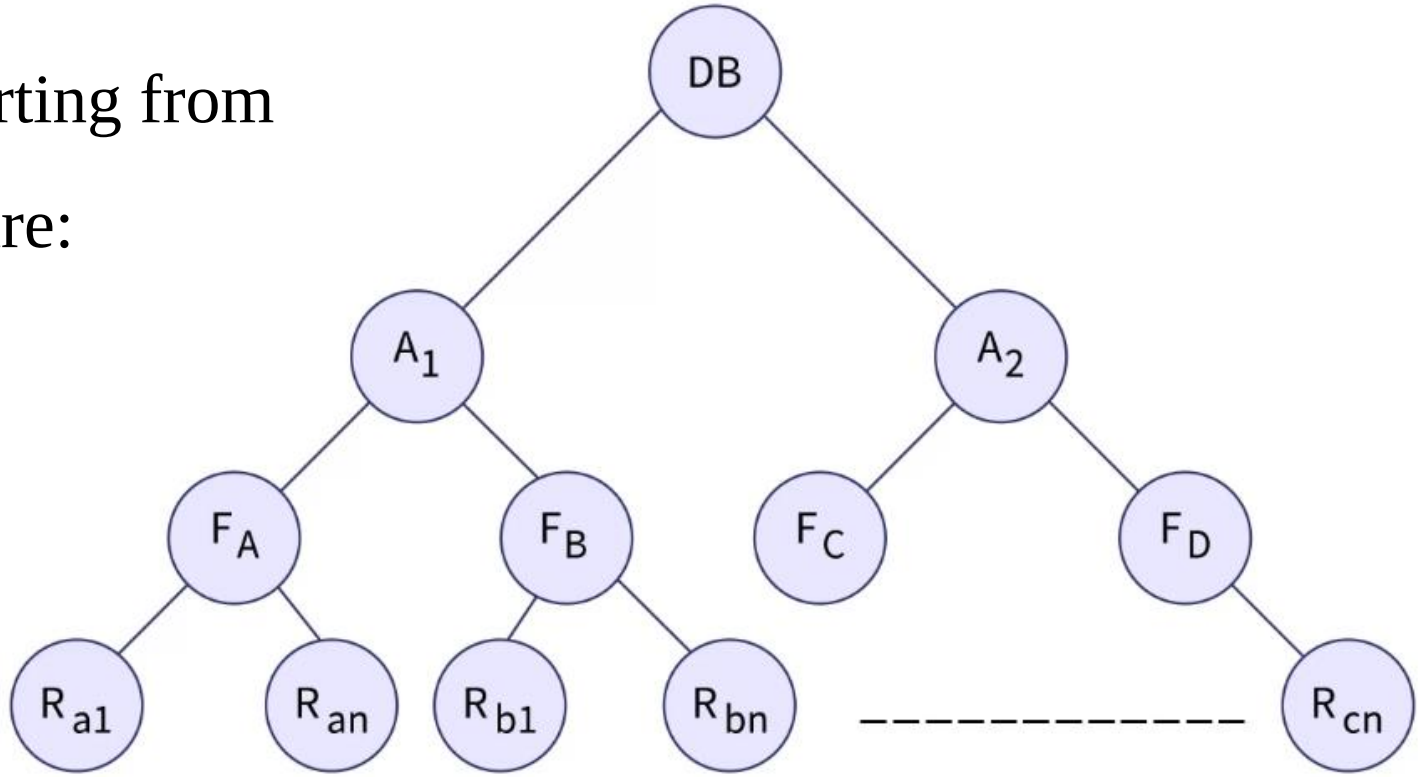
Such a hierarchy can be represented graphically as a tree.

Concurrency Control: Multiple Granularity

For example, consider the tree, which consists of four levels of nodes.

The levels starting from the top level are:

- database
- area
- file
- record



Concurrency Control: Intention Locking

It indicate an intent to modify a particular row.

Intent locks do not conflict with read locks, so acquiring an intent lock does not block other transactions from reading the same row.

However, intent locks do prevent other transactions from acquiring either an intent lock or a write lock on the same row, guaranteeing that the row cannot be changed by any other transaction before an update.

Concurrency Control: Time-Based Protocol

Each transaction is issued a timestamp when it enters the system.

- An old transaction T1 has time-stamp TS(T1) and new transaction T2 is assigned time-stamp TS(T2), then following condition satisfies: TS(T1) < TS(T2). For e.g. TS(T1) = 0002 and TS(T2) = 0005, then TS(T1) < TS(T2).

- Transaction with older timestamp are given priority for execution.

Concurrency Control: Time-Based Protocol

- The protocol manages concurrent execution such that the **read** and **write** operations are executed in timestamp order.

- The protocol maintains for each data Q two timestamp values:

  - **W-timestamp(Q):** largest time-stamp of any transaction that executed write(Q) successfully.

  - **R-timestamp(Q):** largest time-stamp of any transaction that executed read(Q) successfully.

Concurrency Control: Time-Based Protocol

- Suppose a transaction T1 issues a **read(Q).**
  - ✓ If **TS(T1)** <= **W-timestamp(Q)**, then the read operation is rejected as **write** operation is in execution. For e.g. TS(T1) = 0002, W-timestamp(Q) = 0008.
  - ✓ If **TS(T1)** >= **W-timestamp(Q)**, then the read operation is executed as it is assumed that **write** operation is committed. For e.g. TS(T1) = 0005, W-timestamp(Q) = 0003

- Suppose a transaction T1 issues a **write(Q).**
  - ✓ If **TS(T1)** <= **R-timestamp(Q)**, then the write operation is rejected as **read** operation is in execution. For e.g. TS(T1) = 0002, R-timestamp(Q) = 0008.
  - ✓ If **TS(T1)** >= **R-timestamp(Q)**, then the write operation is executed as it is assumed that **read** operation is complete. For e.g. TS(T1) = 0005, R-timestamp(Q) = 0003

Deadlock

A deadlock is a situation in which one transaction is waiting for other transaction to release the resources.

- Let there be a set of transaction {T1, T2, T3} such that T1 is waiting for a data item that T2 holds, and T2 is waiting for a data item that T3 holds, T3 is waiting for a data item that T1 holds.
- Here, each transaction is waiting for some other. Hence, no transaction will work smoothly.
- The only solution to this situation is transaction being rolled back to its initial state every time and come again to check the availability of resources.

Parul® University

NAAC A++ ACCREDITED UNIVERSITY

Deadlock Detection

- A Deadlock Condition arise only if all these four conditions are true together.

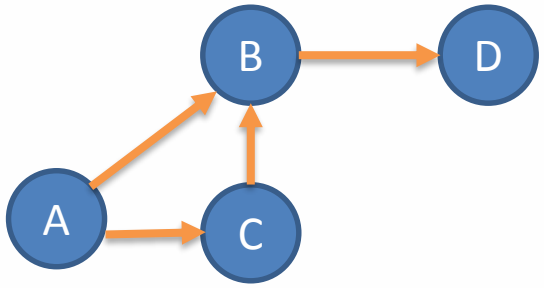
  - ✓ **Mutual Exclusion:** One Process to work at one time. If any other process requests the same resource, it will wait till the first release it.
  - ✓ **Hold and Wait:** One process holds one resource and waiting for other resource that is held by other process.
  - ✓ **Non-Preemption:** Process can release resources only when it has completed its task. No force to acquire resource
  - ✓ **Circular Wait:** One Process waiting for other, Other waiting for another and thus creates the cycle.

Parul® University
NAAC A++ ACCREDITED UNIVERSITY

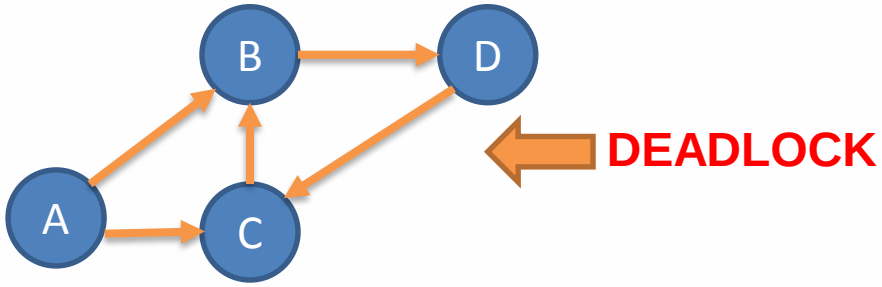
Deadlock Detection

- The most simpler way to detect deadlock is to use **Wait-for Graph.**
- Each Transaction is represented by a node. When a Transaction Ti is waiting for the resource held by Tj, a **directed edge** is drawn from Ti to Tj (Ti -----→ Tj).
- If the Wait-for graph has any cycle, then we can say there is a **Deadlock** and all the transactions in the cycle are also said to be deadlocked.

Deadlock Detection



- A is waiting for B and C
- C is waiting for B
- B is waiting for D
- Still, there is no cycle, hence it is not **Deadlock.**



- A is waiting for B and C
- B is waiting for D
- D is waiting for C
- C is waiting for B
- There is a cycle B – D – C - B, It is a **Deadlock.** for B, C, and D.

Deadlock Recovery

- The most simpler way to recover from deadlock is to **Roll Back** any of the transactions.
- The transaction which incurs minimum loss is rolled back and it is known as **Victim.**
- The decision to roll back a transaction is made on following criteria:
  - ✓ The transaction which have minimum locks
  - ✓ The transaction which has executed less work
  - ✓ The transaction which is very far from completion

Deadlock Prevention

- A Deadlock can be prevented using:

  - ✓ **Wait-Die Approach:**
    - If an older transaction is requesting a resource which is held by younger transaction, then older transaction waits.
    - If an younger transaction is requesting a resource which is held by older transaction, then younger transaction is killed and rolled back.

Wait-Die	
O needs a resource held by Y	O Waits
Y needs a resource held by O	Y Dies

Deadlock Prevention

✓ Wound-Wait Approach:

- If an older transaction is requesting a resource which is held by younger transaction, then older transaction forces younger transaction to kill the transaction and releases the resource.
- If an younger transaction is requesting a resource which is held by older transaction, then younger transaction waits till older transaction releases resource.

Wound-Wait	
O needs a resource held by Y	Y Hurts / Dies
Y needs a resource held by O	Y Waits

Deadlock Prevention

- A Deadlock can be prevented using:
  - ✓ **Timeout-Based Approach:**
    - A transaction waits for a lock only for a specified amount of time. After that, the transaction is rolled back.
    - So deadlock never occurs.

Parul® University

NAAC A++ ACCREDITED UNIVERSITY

DIGITAL LEARNING CONTENT

Thank You!!!

DIGITAL LEARNING CONTENT

Parul® University

www.paruluniversi