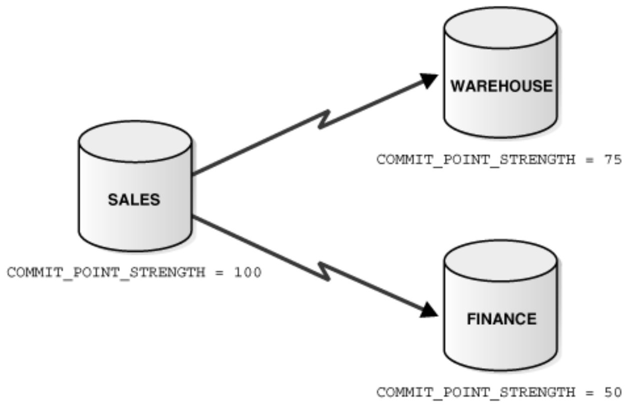
Unit 8
Transaction and Concurrency Control

8.1. Introduction to Transaction

- Transaction is a logical unit of work that represents the real-world events.
- A transaction is also defined as any one execution of a user program in a Database Management System (DBMS).
- The transaction management is the ability of a database management system to manage the different transactions that occur within it.
- Concurrency control is the activity of coordinating the actions of transactions that operate, simultaneously or in parallel to access the shared data.

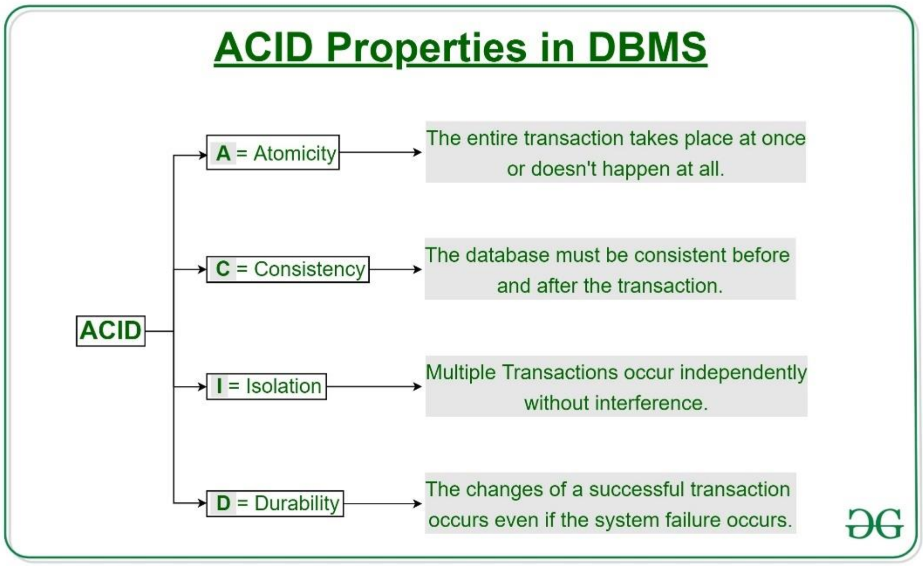

WAREHOUSE
COMMIT_POINT_STRENGTH = 75
SALES
COMMIT_POINT_STRENGTH = 100
FINANCE
COMMIT_POINT_STRENGTH = 50

- ***A transaction can be defined as a unit or part of any program at the time of its execution. During transactions data items can be read or updated or both.***

ACID Properties of Transaction

https://github.com/sanjeevlcc/notes_2081/blob/main/DBMS_BIM_BSCIT_BCA/LABS_ON_AIR/12_LAB%2011%20commit%20Rollback%20delete%20table%20data%20.txt

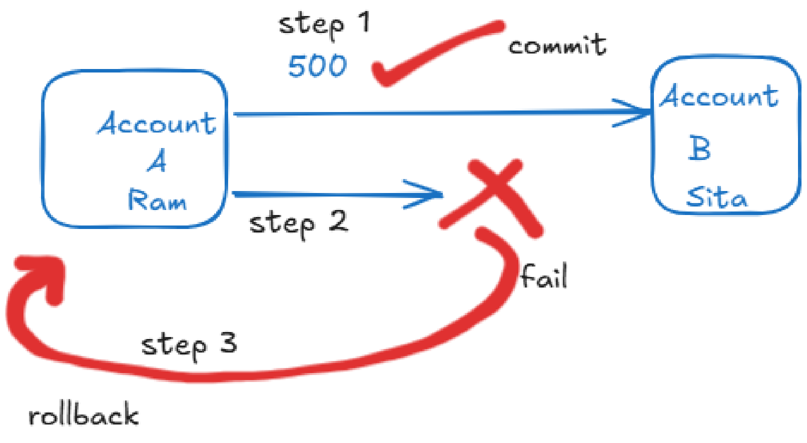
- A **transaction** is a single logical unit of work that accesses and possibly modifies the contents of a database.
- Transactions access data using read-and-write operations. To maintain consistency in a database, before and after the transaction, certain properties are followed.
- These are called **ACID** properties.


ACID Properties in DBMS
ACID
A = Atomicity
The entire transaction takes place at once or doesn't happen at all.
C = Consistency
The database must be consistent before and after the transaction.
I = Isolation
Multiple Transactions occur independently without interference.
D = Durability
The changes of a successful transaction occurs even if the system failure occurs.

1. Atomicity

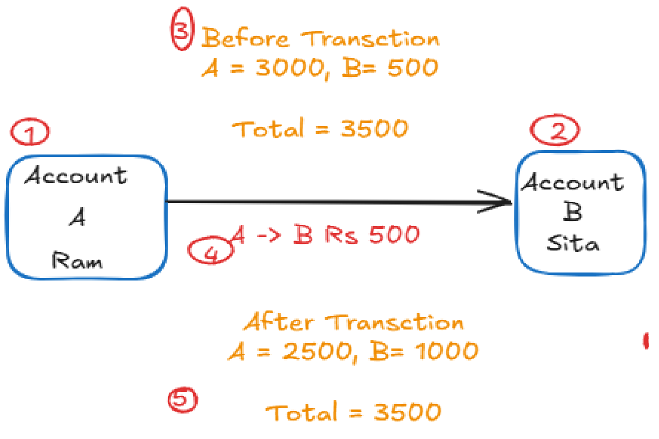
- **Definition**: A transaction is an indivisible unit of work that either completes fully or doesn't happen at all. If any part of the transaction fails, the entire transaction is rolled back.

- **Example**: Imagine transferring ₹500 from Account A to Account B.
    - **Step 1**: Debit ₹500 from Account A.
    - **Step 2**: Credit ₹500 to Account B. If Step 2 fails (e.g., due to a system crash), Step 1 should be rolled back to maintain the system's original state.



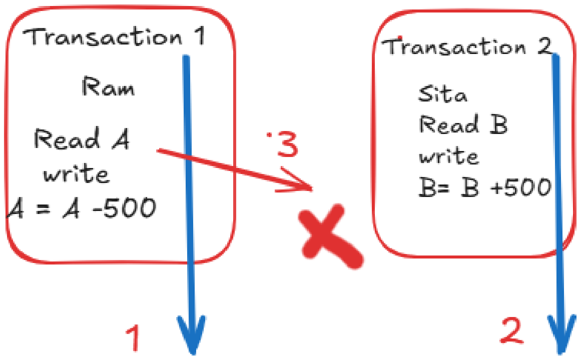
2. Consistency

- **Definition**: A transaction must ensure that the database moves from one valid state to another, maintaining all integrity constraints.
- **Example**: In the same transfer example, if the total balance of Account A and Account B before the transaction is ₹3500, the sum must still be ₹3500 after the transaction. Any violation of constraints (e.g., overdrawing Account A beyond its limit) would prevent the transaction from committing.



3. Isolation

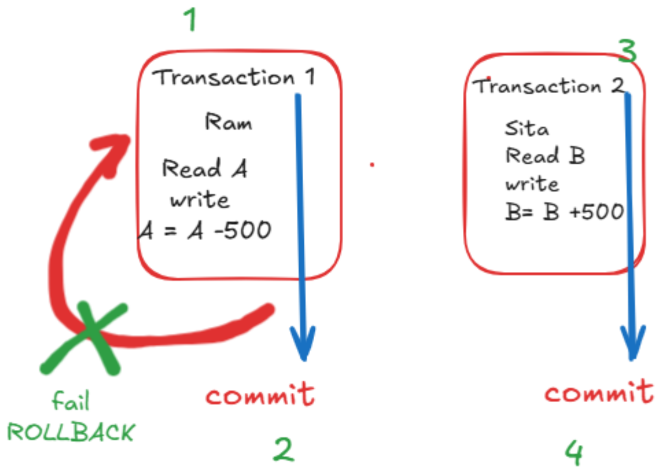
- **Definition**: Transactions should be executed in isolation, meaning the intermediate states of a transaction are not visible to other transactions until they are complete.
- **Example**: Suppose two transactions are occurring simultaneously:



- Transaction 1: Transfers ₹500 from Account A to Account B.
- Transaction 2: Reads the balance of Account B. Transaction 2 should not see the intermediate state of Account B while Transaction 1 is still in progress.

4. Durability

- **Definition**: Once a transaction is committed, its changes must persist, even in the event of a system failure.
- **Example**: After the transfer of ₹500 from Account A to Account B is committed, the changes are permanently saved. Even if the system crashes right after the commit, the updated balances in both accounts should be preserved.

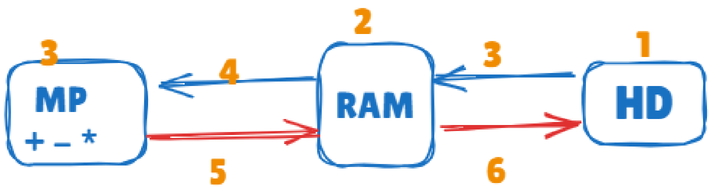


Summary Table:

Property	**Definition**	**Example**
Atomicity	All or nothing.	Money transfer must debit & credit both.
Consistency	Database remains in a valid state.	Total balance remains unchanged.
Isolation	Transactions don't interfere with each other.	One transfer doesn't affect another read.
Durability	Committed changes are permanent.	Balance updates persist after a crash.

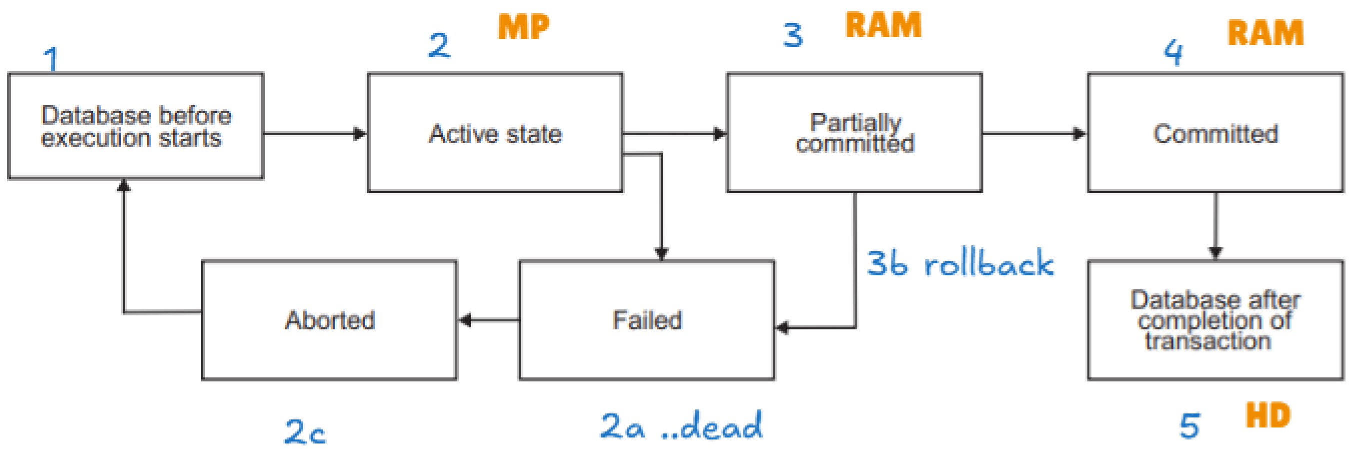
Transaction States

A database transaction goes through multiple states during its lifecycle. These states represent the progress of a transaction from initiation to completion or failure. Below are the key **Transaction States** with explanations:



➢ **Active state** : It is the initial state of transaction. During execution of statements, a transaction is in active state.

- **Partially committed** : A transaction is in partially committed state, when all the statements within transaction are executed but transaction is not committed.
- **Failed** : In any case, if transaction cannot be proceeded further then transaction is in failed state.
- **Committed :** After successful completion of transaction, it is in committed state.



8.2. Concept of Serializability

- A **schedule** is a sequence of operations (read, write, commit, abort) performed by a set of transactions in a database.
- Schedules can be classified into two types:
  - **Serial**
  - **Parallel (Concurrent)**.

1. Serial Schedule

- **Definition**: In a serial schedule, transactions are executed one after the other without overlapping. There is no interleaving of operations from different transactions.

- **Key Point**: Ensures consistency but might be less efficient due to the lack of concurrency.

Example of Serial Schedule

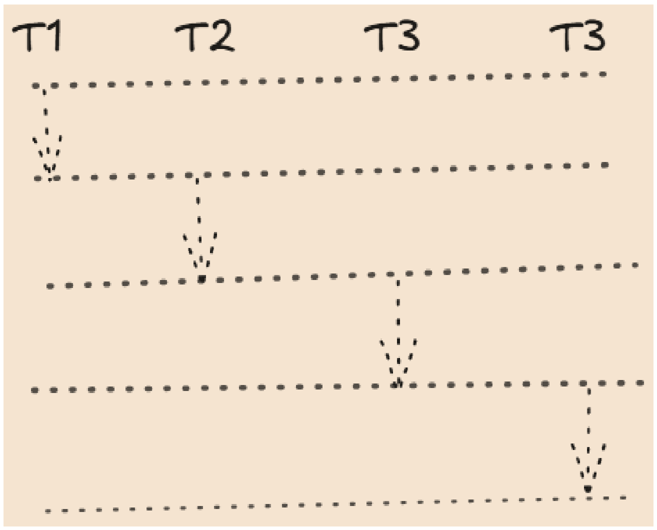
Transactions T1 and T2:

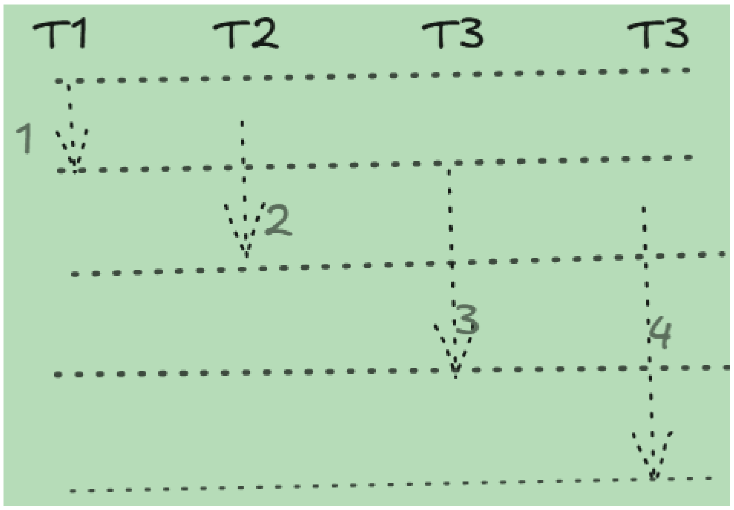
Initial Values: A=100 B=200

Transaction T1: A=A+50	**Transaction T2**: B=B×2
Serial Schedule: Execute T1 first, then T2	
T1: Read A=100, Update A=150, Write A=150	**T2:** Read B=200, Update B=400, Write B=400
Result: A=150, B=400	

2. Parallel (Concurrent) Schedule

- **Definition**: In a parallel schedule, operations from multiple transactions are interleaved. This improves efficiency but must ensure consistency and isolation (using techniques like locking or timestamp ordering).
- **Key Point**: Allows concurrency while maintaining database integrity.



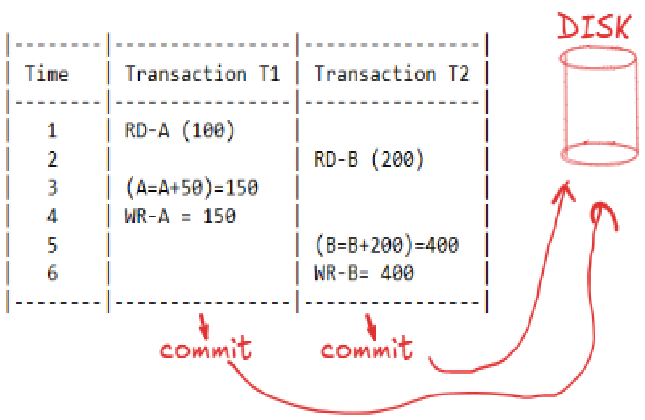


Example of Parallel Schedule

Transactions T1 and T2 (same as above):

Parallel Schedule: Interleaved operations:

1. T1: Read A=100
2. T2: Read B=200
3. T1: Update A=150
4. T1: Write A=150



```
|--------|-----------------|-----------------|
| Time   | Transaction T1  | Transaction T2  |
|--------|-----------------|-----------------|
|   1    | RD-A (100)      |                 |
|   2    |                 | RD-B (200)      |
|   3    | (A=A+50)=150    |                 |
|   4    | WR-A = 150      |                 |
|   5    |                 | (B=B+200)=400   |
|   6    |                 | WR-B= 400       |
|--------|-----------------|-----------------|
```

DISK
commit
commit

5. T2: Update B=400
6. T2: Write B=400

Result:

A=150 B=400

Key Differences

Feature	Serial Schedule	Parallel Schedule
Execution	One transaction at a time	Interleaved execution
Efficiency	Slower due to lack of concurrency	Faster due to concurrent processing
Consistency	Easy to maintain	Requires proper mechanisms (e.g., locks)
Example Use Case	Small-scale systems or critical operations	High-performance systems needing concurrency

8.3. Concurrent execution:

Aspect	**Serial Schedule**	**Parallel (Concurrent) Schedule**
Execution	Transactions execute one after another sequentially.	Transactions execute with interleaving operations.
Performance	Slower due to no concurrency.	Faster as multiple transactions run concurrently.
Complexity	Simple to implement and manage.	Complex due to the need for concurrency control mechanisms like locking or timestamp ordering.
Concurrency	No concurrency; only one transaction runs at a time.	Allows concurrency, enabling multiple transactions to run simultaneously.
Consistency	Guarantees consistency easily.	Requires mechanisms to ensure consistency, such as conflict resolution and isolation levels.
Isolation	Full isolation; no interference between transactions.	Needs additional measures (e.g., locking, schedules) to maintain isolation.

Resource Utilization	Low, as resources are used by one transaction at a time.	High, as multiple transactions utilize resources simultaneously.
Use Case	Suitable for small-scale or highly critical systems where accuracy is paramount.	Suitable for large-scale, high-performance systems needing faster processing.
Risk of Deadlock	None, as there's no concurrency.	Possible if concurrency control mechanisms are not properly implemented.
Example	Banking operations with high priority and security.	Real-time systems like online booking or e-commerce.

Concurrent Execution

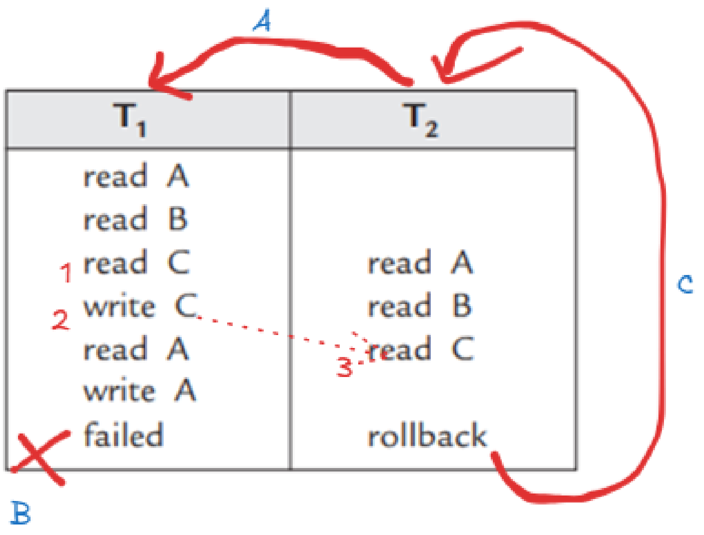
If more than one transaction are executed at the same time then they are said to be executed concurrently.

Recoverability

To maintain atomicity of database, undo effects of any transaction has to be performed in case of failure of that transaction. If undo effects successfully then that database maintains recoverability. This process is known as Rollback.

Cascading Rollback

- ➢ If any transaction Ti is dependent upon Tj and Tj is failed due to any reason then rollback Ti .
- ➢ This is known as cascading rollback.
- ➢ Consider, Here *T2 is dependent upon T1* because T2 reads value of C which is updated by T1.



T_1	T_2
read A	
read B	
read C	read A
write C	read B
read A	read C
write A	
failed	rollback

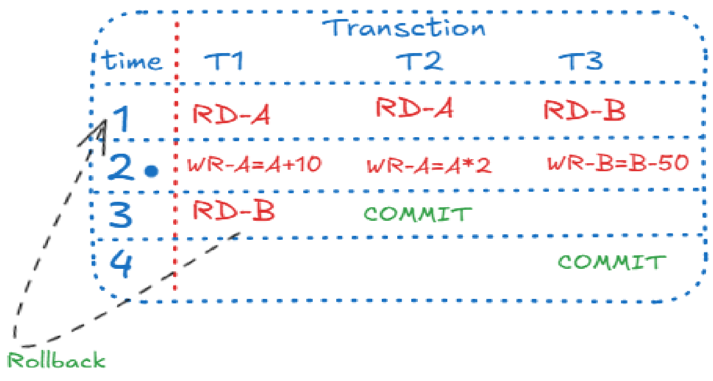
If T1 fails then rollback T1 and also T2.

Types of Problems in Concurrency

Concurrency in database systems can lead to several problems if not managed properly.

Common issues include:

1. *Lost Update*
2. *Dirty Read*
3. *Uncommitted Dependency (Cascading Rollback)*
4. *Non-Repeatable Read*
5. *Phantom Read*


Transction
Rollback

time	T1	T2	T3
1	RD-A	RD-A	RD-B
2	WR-A=A+10	WR-A=A*2	WR-B=B-50
3	RD-B	COMMIT	
4			COMMIT

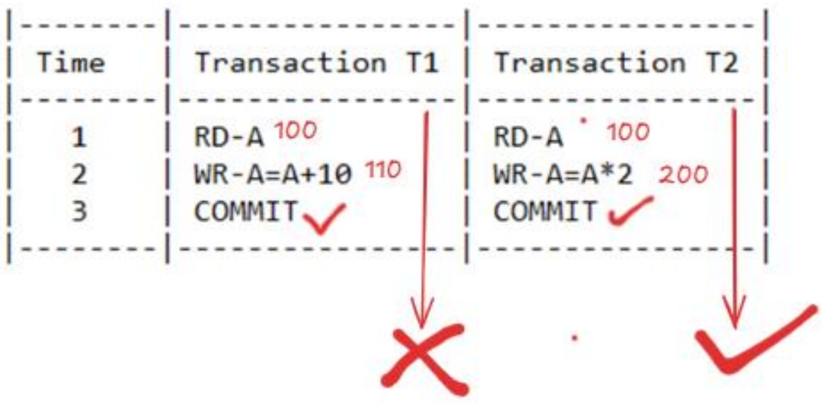
Time	Transaction	Operation	Description
T1.1	T1	READ(A)	T1 reads the value of A.
T2.1	T2	READ(A)	T2 reads the value of A concurrently.
T1.2	T1	WRITE(A = A + 10)	T1 updates A by adding 10.
T2.2	T2	WRITE(A = A * 2)	T2 updates A by multiplying by 2, overwriting T1's update.
T3.1	T3	READ(B)	T3 reads the value of B.
T3.2	T3	WRITE(B = B - 50)	T3 updates B by subtracting 50.
T1.3	T1	READ(B)	T1 reads the uncommitted value of B from T3.
T3.3	T3	ROLLBACK	T3 rolls back, invalidating T1's read.
T2.3	T2	COMMIT	T2 commits its changes to A.
T1.4	T1	COMMIT	T1 commits its changes to B.

1. Lost Update

- **Definition**: Two transactions update the same data simultaneously, and one update overwrites the other, leading to the loss of an intermediate update.

Example:

- Initial value of A=100.
- **T1** reads A, adds 10 (A=110), and writes it.
- Simultaneously, **T2** reads A, multiplies it by 2 (A=200), and writes it.
- **Result**: A=200 (Update from **T1** is lost).



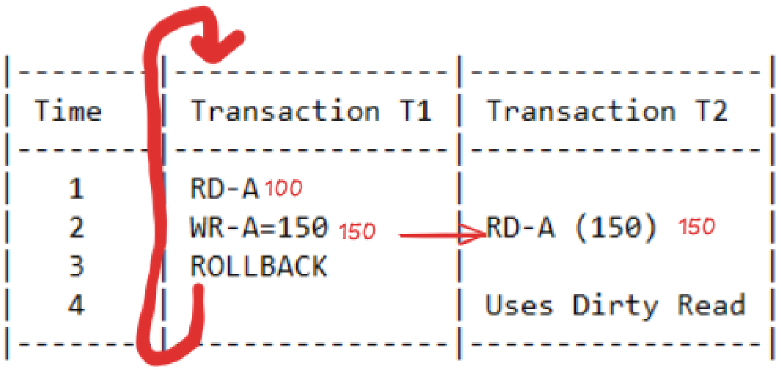
Time	Transaction T1	Transaction T2
1	RD-A 100	RD-A 100
2	WR-A=A+10 110	WR-A=A*2 200
3	COMMIT ✓	COMMIT ✓
	✗	✓

2. Dirty Read

- **Definition**: A transaction reads uncommitted changes made by another transaction, which can cause inconsistencies if the other transaction is rolled back.

Example:

- Initial value of A=100
- **T1** updates A=150 but hasn't committed yet.
- **T2** reads A=150 before **T1** commits. Later, **T1** rolls back, and A returns to 100.
- **Result**: **T2** uses an incorrect value.



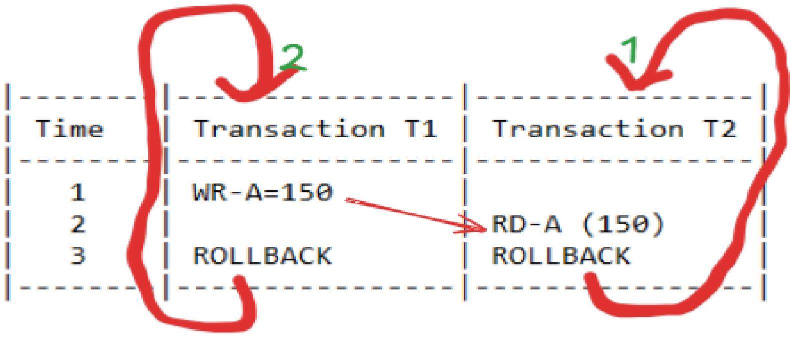
Time	Transaction T1	Transaction T2
1	RD-A 100	
2	WR-A=150 150	RD-A (150) 150
3	ROLLBACK	
4		Uses Dirty Read

3. Uncommitted Dependency (Cascading Rollback)

- **Definition**: A transaction depends on uncommitted changes from another transaction. If the latter rolls back, it forces the dependent transaction to roll back too.

Example:

- **T1** updates A=150 but hasn't committed yet.
- **T2** reads A=150 and uses it to update B=B+A.
- If **T1** rolls back, **T2** must also roll back to maintain consistency.



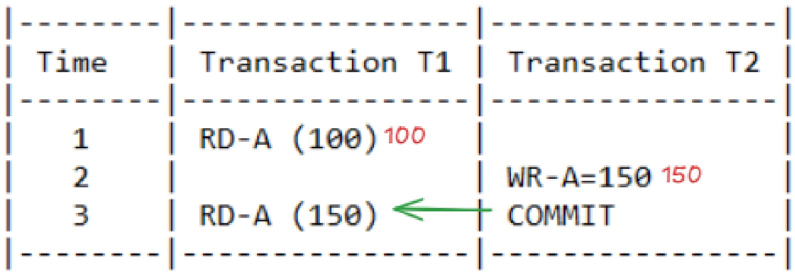
Time	Transaction T1	Transaction T2
1	WR-A=150	
2		RD-A (150)
3	ROLLBACK	ROLLBACK

4. Non-Repeatable Read

- **Definition**: A transaction reads the same data twice and finds different values because another transaction modified it in the meantime.

Example:

- Initial value of A=100.
- **T1** reads A=100
- **T2** updates A=150 and commits.
- **T1** reads A again and finds A=150A (different from its first read).



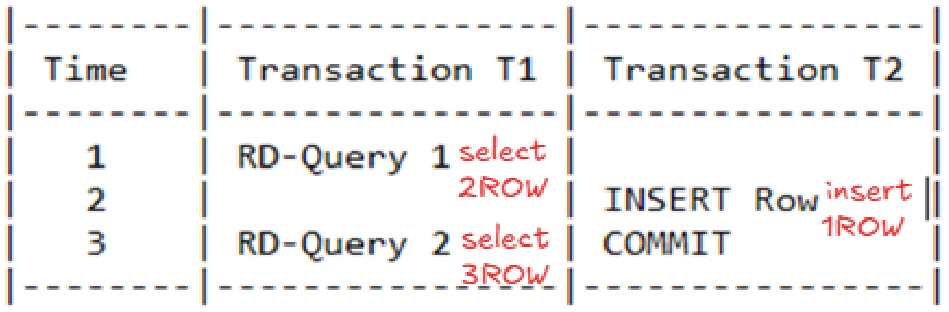
Time	Transaction T1	Transaction T2
1	RD-A (100) 100	
2		WR-A=150 150
3	RD-A (150)	COMMIT

5. Phantom Read

- **Definition**: A transaction retrieves a different set of rows in two queries due to changes made by another transaction (e.g., inserts or deletes).

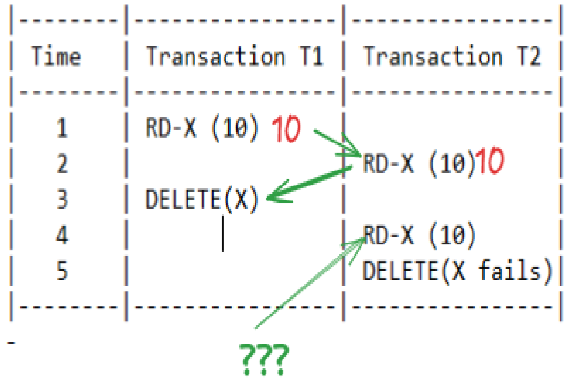
Example 1:

- **T1** reads rows where salary > 50,000 and finds two rows.
- **T2** inserts a new row with salary > 50,000 and commits.
- **T1** re-executes the same query and finds three rows.



```
|--------|----------------|----------------|
| Time   | Transaction T1 | Transaction T2 |
|--------|----------------|----------------|
|   1    | RD-Query 1 select |             |
|   2    |           2ROW | INSERT Row insert |
|   3    | RD-Query 2 select | COMMIT   1ROW |
|--------|-----------3ROW-|----------------|
```

Example 2:	
Transaction T1:	**Transaction T2**
Reads X=10. Deletes X.	Reads X=10 after T1 has deleted X. Tries to delete X, but X no longer exists (phantom behavior).
This inconsistency arises because the same data X behaves differently between queries by **T1** and **T2**.	



```
|-------|---------------|---------------|
| Time  | Transaction T1 | Transaction T2 |
|-------|---------------|---------------|
|  1    | RD-X (10) 10  |               |
|  2    |               | RD-X (10)10   |
|  3    | DELETE(X)     |               |
|  4    |               | RD-X (10)     |
|  5    |               | DELETE(X fails)|
|-------|---------------|---------------|
```

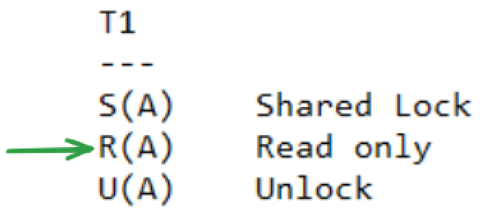
8.4. Lock based Concurrency Control

➢ Lock-based concurrency control is a mechanism used in databases to ensure the consistency and integrity of data during concurrent transactions.

➢ Locks are used to restrict access to data items when a transaction is reading or writing them, thereby preventing conflicts and ensuring isolation.

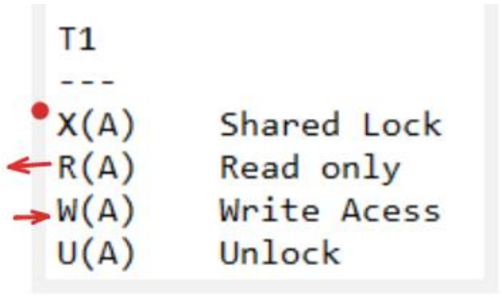
Types of Locks

1. **Shared Lock (S-lock)**:
   - Allows multiple transactions to **read** a data item but prevents any transaction from modifying it.
   - Example: Transaction T1 can read data A while Transaction T2 also reads A, but neither can write to X.



```
T1
---
S(A)     Shared Lock
R(A)     Read only
U(A)     Unlock
```

2. **Exclusive Lock (X-lock)**:
   - Only one transaction can hold an exclusive lock on a data item, allowing it to **read and write** the data.
   - Example: If Transaction T1 has an exclusive lock on A, no other transaction can read or write A until the lock is released.



```
T1
---
X(A)     Shared Lock
R(A)     Read only
W(A)     Write Acess
U(A)     Unlock
```

Locking Protocols

1. **Two-Phase Locking Protocol (2PL)**:
   - Ensures serializability by dividing the transaction's execution into two phases:
     - **Growing Phase**: A transaction acquires all the locks it needs.
     - **Shrinking Phase**: A transaction releases locks and cannot acquire new ones.
   - **Strict 2PL**: All exclusive locks are held until the transaction commits or aborts.
   - Example:

     T1: S-lock(X) -> S-lock(Y) -> X-lock(Z) -> COMMIT -> Release all locks

2. **Deadlock Avoidance in Locking**:
   - Deadlock occurs when multiple transactions hold locks on resources and wait for each other to release locks.
   - Deadlock prevention techniques include:
     - **Wait-Die**: Older transactions wait; younger transactions are aborted.
     - **Wound-Wait**: Older transactions preempt younger transactions by forcing them to abort.

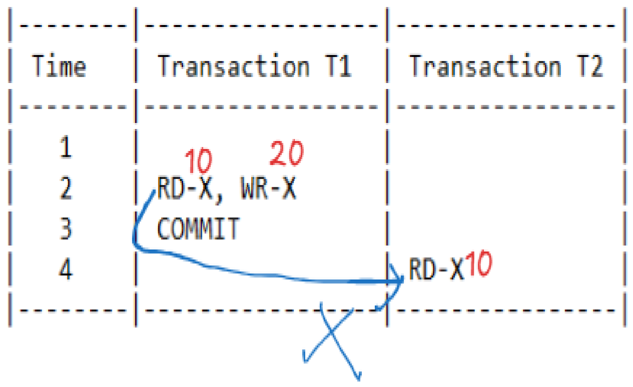
Example of Lock-Based Concurrency Control

Scenario:

- **T1**: Reads and then writes X.
- **T2**: Reads X.

Without Locking:

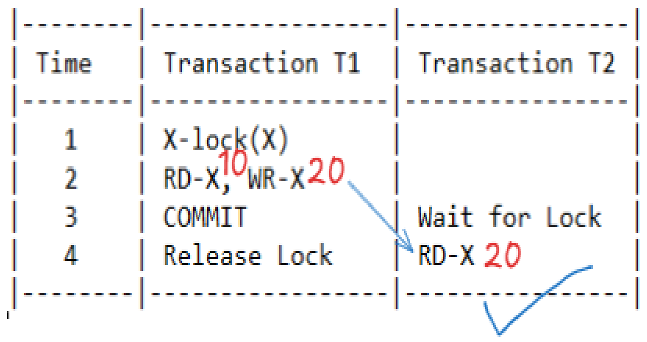
- T1 reads X=10, updates X=20, but before committing, T2 reads X=10, leading to inconsistency.



Time	Transaction T1	Transaction T2
1		
2	$\overset{10}{\text{RD-X}}$, $\overset{20}{\text{WR-X}}$	
3	COMMIT	
4		RD-X10

With Locking:

1. T1 acquires an **X-lock** on X.
2. T2 waits for the lock to be released before reading X.
3. T1 commits, releases the lock, and T2 proceeds, ensuring consistency.



Time	Transaction T1	Transaction T2
1	X-lock(X)	
2	$\overset{10}{\text{RD-X}}$, WR-X20	
3	COMMIT	Wait for Lock
4	Release Lock	RD-X 20

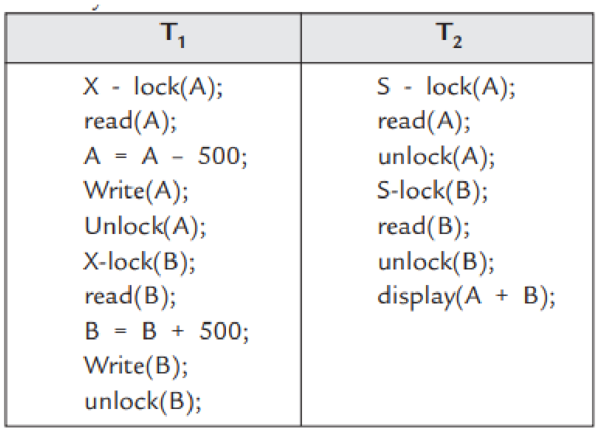
Advantages of Lock-Based Concurrency Control

1. Prevents conflicts between concurrent transactions.
2. Ensures data consistency and isolation.

Disadvantages

1. Can lead to **deadlocks**.
2. Reduces performance due to increased transaction wait time.

Example: Transactions T1 and T2 for banking example

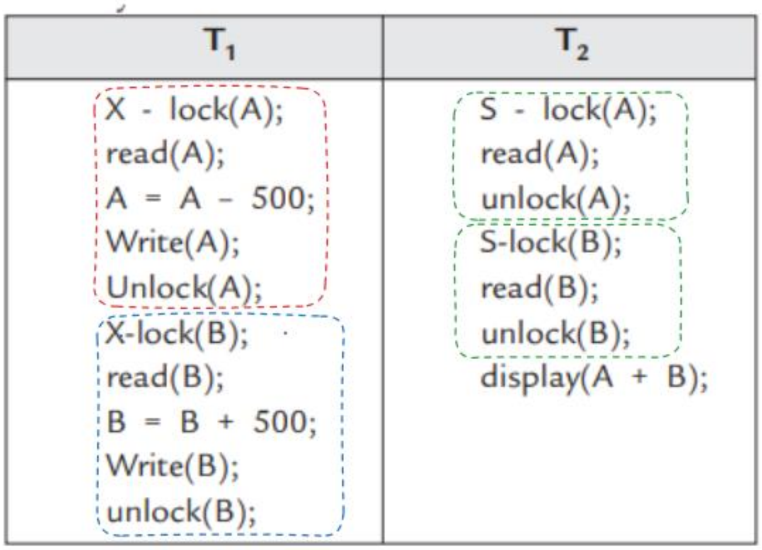


T₁	T₂

X - lock(A);
read(A);
A = A - 500;
Write(A);
Unlock(A);
X-lock(B);
read(B);
B = B + 500;
Write(B);
unlock(B);
	S - lock(A);
	read(A);
	unlock(A);
	S-lock(B);
	read(B);
	unlock(B);
	display(A + B);

Consider the banking system in which you have to transfer Rs 500 from account A to account B. Transaction T1 in transfer of amount and T2 shows sum of accounts A and B. Initially account A has Rs 1000 and B has Rs 300.

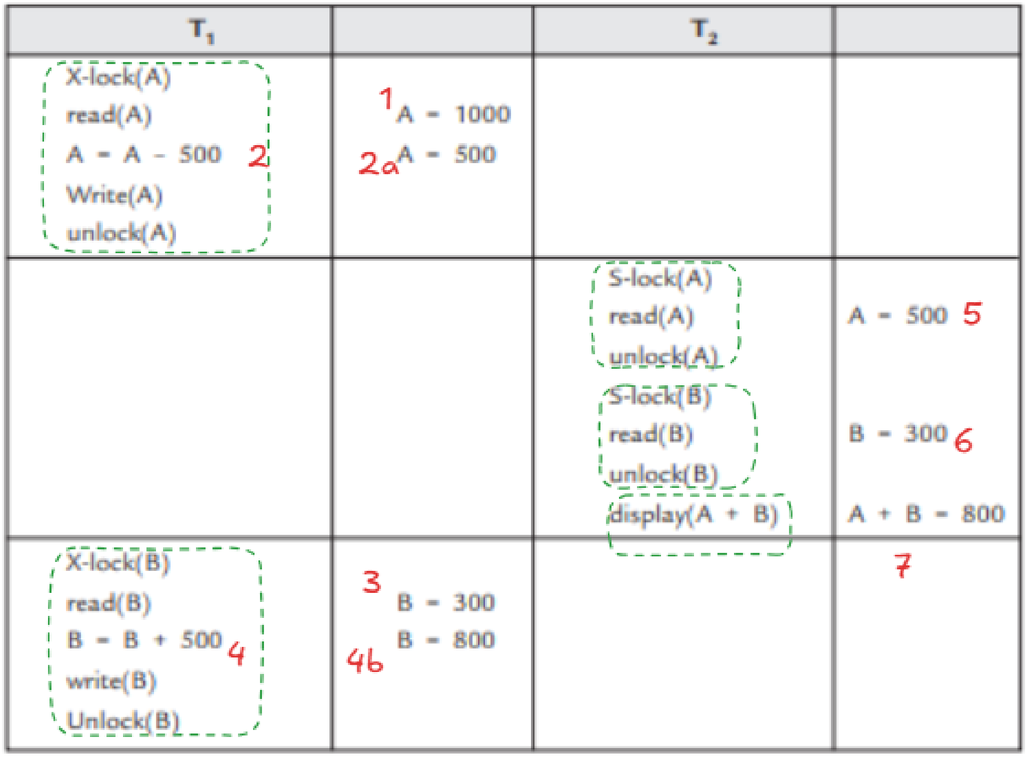
Solution:



T₁	T₂

X - lock(A);
read(A);
A = A - 500;
Write(A);
Unlock(A);
X-lock(B);
read(B);
B = B + 500;
Write(B);
unlock(B);
	S - lock(A);
	read(A);
	unlock(A);
	S-lock(B);
	read(B);
	unlock(B);
	display(A + B);

Schedule for transaction T1 and T2 in case locked data items are unlocked immediately after its final access.



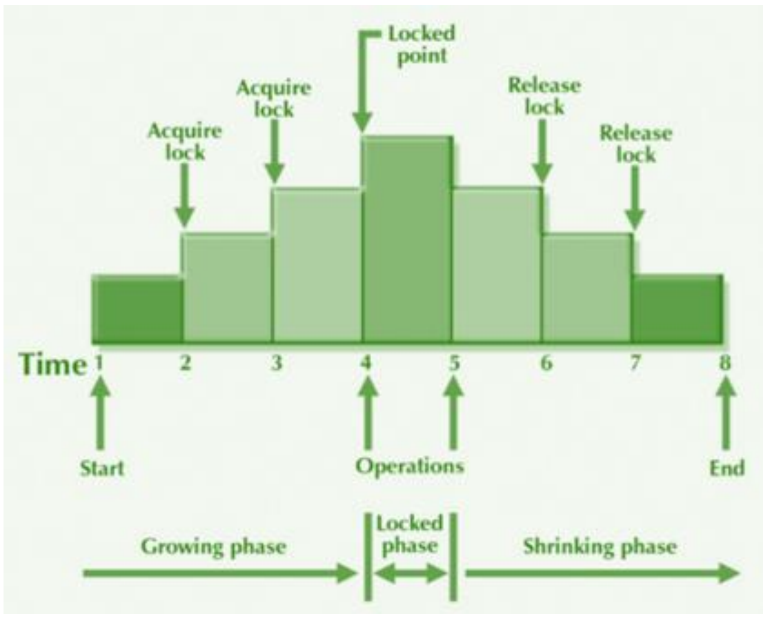
T₁		T₂	

X-lock(A)
read(A)
A = A - 500 2
Write(A)
unlock(A)
	1 A = 1000
	2a A = 500		

		S-lock(A)
		read(A)
		unlock(A)
		S-lock(B)
		read(B)
		unlock(B)
		display(A + B)
			A = 500 5
			B = 300 6
			A + B = 800

X-lock(B)
read(B)
B = B + 500 4
write(B)
Unlock(B)
	3 B = 300
	4b B = 800
			7

8.5. 2PL and Strict 2PL



- Locking protocols are methods to ensure **concurrent transaction execution** while maintaining data consistency and isolation in a database.
- These protocols specify when locks can be acquired and released.

- Below are three main locking protocols:

  i. ***Two-Phase Locking Protocol (2PL)***:
  ii. ***Strict Two-phase Locking Protocol***
  iii. ***Rigorous Two-phase Locking Protocol***

1. **Two-Phase Locking Protocol (2PL)**:

   The **Two-Phase Locking Protocol** ensures **serializability** by dividing a transaction's execution

   T1: S-lock(X) -> S-lock(Y) -> X-lock(Z) -> COMMIT -> Release all locks

   into two distinct phases:

   Growing Phase:

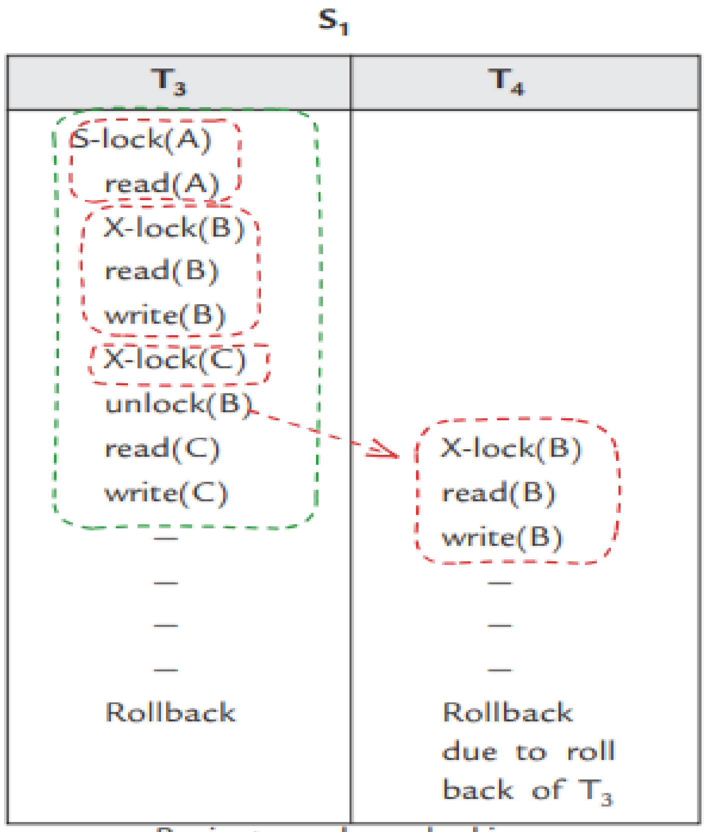
   - A transaction **acquires locks** (shared or exclusive).
   - No locks can be released during this phase.

   Shrinking Phase:

   - A transaction **releases locks** but cannot acquire any new locks.

   Key Characteristics:

   - Guarantees **serializability** of transactions.



T_3	T_4
S-lock(A)	
read(A)	
X-lock(B)	
read(B)	
write(B)	
X-lock(C)	
unlock(B)	
read(C)	X-lock(B)
write(C)	read(B)
—	write(B)
—	—
—	—
—	—
Rollback	Rollback due to roll back of T_3

Basic two-phase locking

- Does **not ensure recoverability** or prevent cascading rollbacks.

Advantages:

- It ensures serializability.

Disadvantages:

- It may cause deadlock.
- It may cause cascading rollback

2. Strict Two-Phase Locking Protocol (Strict 2PL)

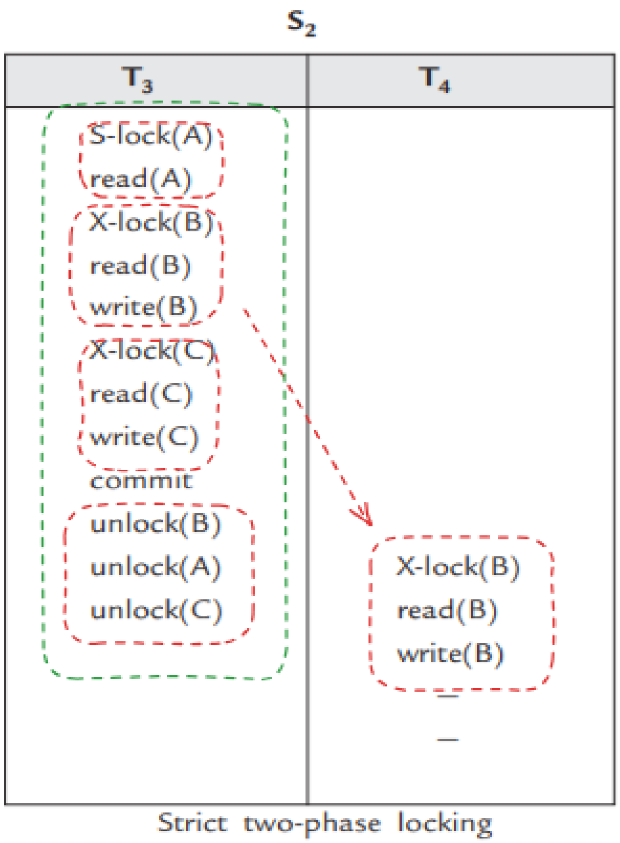
In **Strict 2PL**, the protocol extends 2PL by holding **exclusive locks (X-locks)** until the transaction **commits or aborts**.

Key Characteristics:

- Ensures **serializability** like 2PL.
- Guarantees **recoverability** and prevents **cascading rollbacks**.
- Frequently used in real-world database systems.

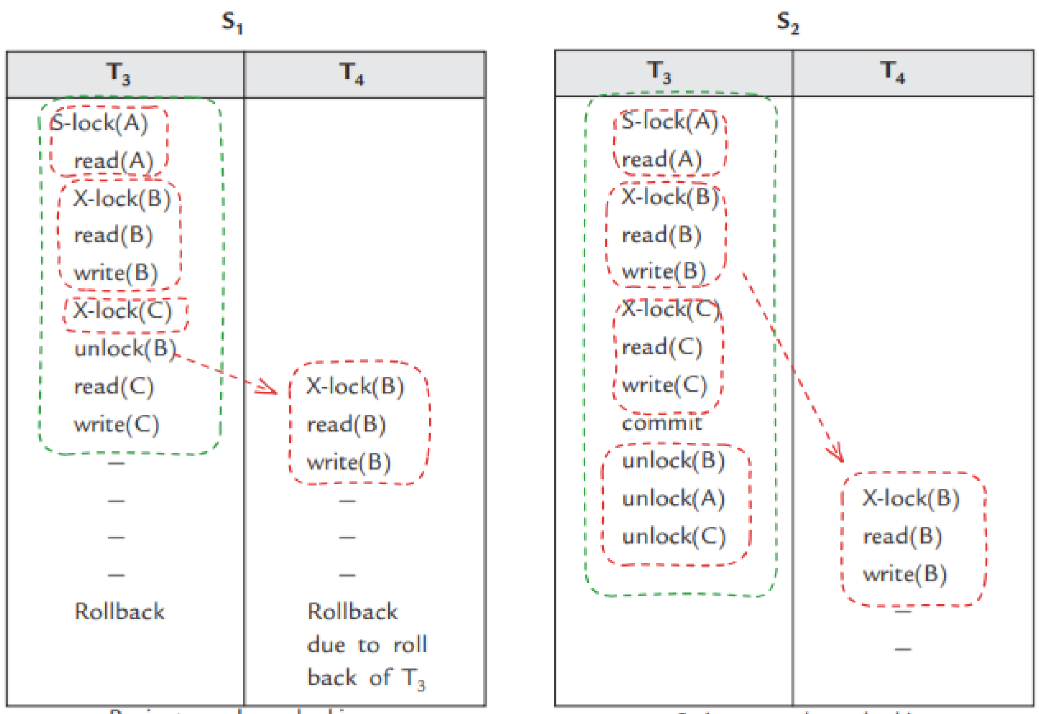
Advantages

There is no cascading rollback in strict two phase locking protocol



Example: Consider partial schedules ***S1 (in basic two-phase locking)*** and ***S2 (in strict two-phase locking)*** as shown in fig.

In S1 there is cascading rollback of T4 due to rollback of T3 but in S2, all exclusive lock are kept till end and avoid cascading rollback.



S_1

T_3	T_4
S-lock(A)	
read(A)	
X-lock(B)	
read(B)	
write(B)	
X-lock(C)	
unlock(B)	
read(C)	X-lock(B)
write(C)	read(B)
–	write(B)
–	–
–	–
–	–
Rollback	Rollback due to roll back of T_3

Basic two-phase locking
(a)

S_2

T_3	T_4
S-lock(A)	
read(A)	
X-lock(B)	
read(B)	
write(B)	
X-lock(C)	
read(C)	
write(C)	
commit	
unlock(B)	
unlock(A)	X-lock(B)
unlock(C)	read(B)
	write(B)
	–
	–

Strict two-phase locking
(b)

3. **Rigorous Two-Phase Locking Protocol**

In **Rigorous 2PL**, the protocol takes the strictness of Strict 2PL further:

- **All locks (both shared and exclusive)** are held until the transaction **commits or aborts**.
- In rigorous two phase locking protocol, all locks are kept until the transaction commits. It means both shared and exclusive locks cannot be released till the end of transaction.

Key Characteristics:

- Ensures **serializability**, **recoverability**, and **strict isolation**.
- Prevents **cascading rollbacks** and allows easier recovery in case of transaction failure.
- Higher level of locking leads to **longer wait times** and potential performance degradation.

Here, both shared and exclusive locks are retained until the transaction T1 commits.

Time	Transaction T1	Transaction T2
1	S-lock(X)	
2	RD-X	
3	X-lock(X)	
4	WR-X	Wait for Lock
5	COMMIT	RD-X
6	Release Locks	

Deadlock Avoidance in Locking:

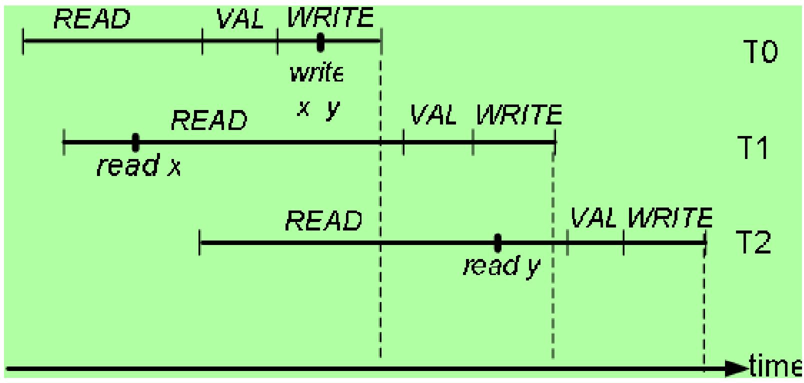
- Deadlock occurs when multiple transactions hold locks on resources and wait for each other to release locks.
- Deadlock prevention techniques include:
  - **Wait-Die**: Older transactions wait; younger transactions are aborted.
  - **Wound-Wait**: Older transactions preempt younger transactions by forcing them to abort.

8.6. Timestamp concept / Timestamp-based concurrency control

- In two-phase and graph based protocols, conflict transactions are found at run time. To order them in advance, use timestamp based protocols.

  NOTE: (Conflict transactions are those transactions that need same data items during execution).

- The **timestamp-based concurrency control** is a method in DBMS to manage concurrent transactions without using locks. It ensures **serializability** by assigning a **timestamp** to each transaction and enforcing a strict ordering based on those timestamps.



Key Concepts of Timestamps

1. **Timestamp (TS)**:
   - Each transaction Ti is assigned a unique timestamp TS(Ti) at the start.
   - Timestamps represent the logical order of transactions, ensuring older transactions execute first.
   - Smaller timestamps indicate older transactions, and larger timestamps indicate newer ones.
2. **Read and Write Timestamps**:
   - Each data item X maintains:

- **$R_TS(X)$:** The largest timestamp of any transaction that successfully read X.
- **$W_TS(X)$**: The largest timestamp of any transaction that successfully wrote X.

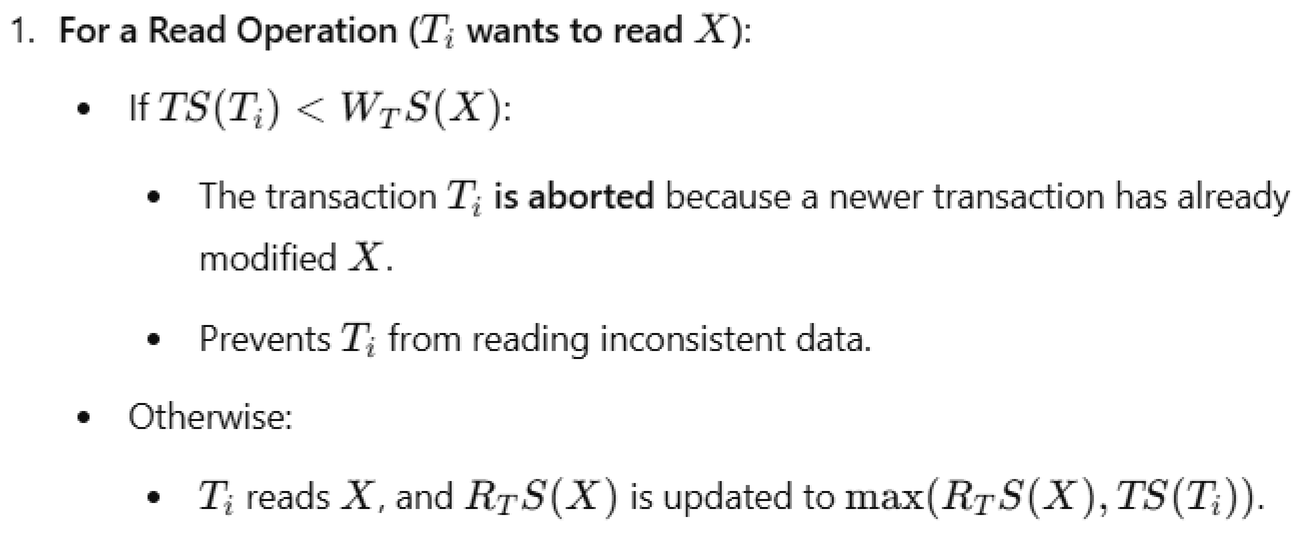
Advantages of Timestamp Protocol

1. **Deadlock-Free**:
   - No locks are used, so deadlocks cannot occur.
2. **Ensures Serializability**:
   - Transactions are ordered based on timestamps, guaranteeing serializable schedules.

Disadvantages of Timestamp Protocol

1. **Abort Overhead**:
   - Transactions may abort frequently if they conflict with newer transactions.
2. **Starvation**:
   - Older transactions may be repeatedly aborted if newer transactions continue to access the same data.
3. **Storage Overhead**:
   - Maintaining **$R_TS(X)$** and **$W_TS(X)$** for all data items increases memory usage.

Basic Rules for Timestamp Protocol



1. **For a Read Operation (T_i wants to read X):**
   - If $TS(T_i) < W_TS(X)$:
     - The transaction T_i **is aborted** because a newer transaction has already modified X.
     - Prevents T_i from reading inconsistent data.
   - Otherwise:
     - T_i reads X, and $R_TS(X)$ is updated to $\max(R_TS(X), TS(T_i))$.

2. **For a Write Operation (T_i wants to write X):**
   - If $TS(T_i) < R_TS(X)$:
     - The transaction T_i **is aborted** because a newer transaction has already read X.
     - Prevents overwriting changes read by newer transactions.
   - If $TS(T_i) < W_TS(X)$:
     - The transaction T_i **is aborted** because a newer transaction has already modified X.
   - Otherwise:
     - T_i writes to X, and $W_TS(X)$ is updated to $TS(T_i)$.

Example

Scenario:

- Two transactions: $T1$ (older) and $T2$ (newer).
- Data item X has:
  - Initial $R_TS(X) = 0, W_TS(X) = 0$.

Actions:

1. $T1$ (TS=5) wants to **read** X:
   - $TS(T1) > W_TS(X)$ (5 > 0).
   - $T1$ reads X, $R_TS(X) = \max(0, 5) = 5$.
2. $T2$ (TS=10) wants to **write** X:
   - $TS(T2) > W_TS(X)$ (10 > 0).
   - $T2$ writes to X, $W_TS(X) = 10$.
3. $T1$ (TS=5) wants to **write** X:
   - $TS(T1) < W_TS(X)$ (5 < 10).
   - $T1$ **is aborted** because a newer transaction ($T2$) has already modified X.

```
|--------|-----------------|----------------|
| Time   | Transaction T1  | Transaction T2 |
|--------|-----------------|----------------|
|   1    | RD(X) (TS=5)    |                |
|        | R_TS(X) = 5     |                |
|   2    |                 | WR(X) (TS=10)  |
|        |                 | W_TS(X) = 10   |
|   3    | WR(X) (TS=5)    |                |
|        | ABORT (TS=5 < W_TS=10)           |
|--------|-----------------|----------------|
```

Q/A

Fill in the Blanks (20 Questions)

1. A transaction is a logical unit of _______ in a database system.
2. The ACID property that ensures all parts of a transaction are completed or none is _______.
3. _______ is the process of managing the simultaneous execution of transactions in a database.
4. A schedule in which transactions execute one after another without overlapping is called a _______ schedule.
5. The phase in 2PL where locks can only be acquired is called the _______ phase.
6. The database property that ensures data consistency across transactions is _______.
7. A transaction that reads uncommitted data from another transaction results in a _______.
8. Shared locks allow multiple transactions to _______ the same data.
9. Exclusive locks are required for transactions that need to _______ data.
10. The _______ protocol in concurrency control holds exclusive locks until commit or abort.
11. A _______ is the unique identifier assigned to each transaction to determine its order.
12. _______ occurs when multiple transactions wait on each other for locks in a cyclic manner.
13. In timestamp-based protocols, _______ ensures serializability by ordering transactions.
14. Cascading rollback occurs when a transaction rollback triggers _______ rollbacks.
15. The _______ phase in transaction states indicates execution of all operations but no commit yet.
16. The locking mechanism that avoids cascading rollbacks is _______ 2PL.
17. The property of a database system that ensures committed changes persist is _______.
18. Concurrent execution improves system performance but requires _______ control.
19. _______ reads occur when a transaction reads the same data twice but sees different results.
20. _______ reads result from new rows being added or deleted by other transactions between reads.

Multiple Choice Questions (MCQ) (20 Questions)

1. What does the "A" in ACID properties stand for?
   - a) Atomicity
   - b) Accessibility
   - c) Alterability
   - d) Accountability
2. Which property ensures that a transaction leaves the database in a consistent state?
   - a) Isolation
   - b) Durability
   - c) Consistency
   - d) Atomicity
3. What is the main purpose of concurrency control?
   - a) To optimize database queries
   - b) To ensure transactions execute serially
   - c) To manage simultaneous transaction execution
   - d) To back up the database
4. In a serial schedule:
   - a) Transactions execute concurrently.
   - b) Transactions execute one after the other.
   - c) Transactions use timestamps.
   - d) None of the above.
5. What type of lock allows only reading of a data item?
   - a) Exclusive lock
   - b) Shared lock
   - c) Timestamp lock
   - d) None of the above
6. Which phase of 2PL prevents acquiring new locks?
   - a) Growing phase
   - b) Shrinking phase
   - c) Locking phase
   - d) Unlocking phase

7. Deadlocks occur due to:
   - a) Shared locks
   - b) Cyclic waiting on resources
   - c) Timestamp conflicts
   - d) Serialization
8. What is the primary benefit of Strict 2PL?
   - a) No deadlocks
   - b) Prevents cascading rollbacks
   - c) Improves performance
   - d) Guarantees durability
9. Which timestamp-based property prevents a transaction from reading older data?
   - a) WTS
   - b) RTS
   - c) Commit timestamp
   - d) Rollback timestamp
10. What is the main drawback of locking mechanisms?
    - a) Complexity
    - b) Deadlocks
    - c) Isolation
    - d) Performance

Short Questions (20 Questions)

1. Define a transaction in a database system.
2. What are the four ACID properties?
3. Explain atomicity with an example.
4. How does consistency ensure database integrity?
5. What is the difference between serial and parallel schedules?
6. What are shared and exclusive locks?
7. Define cascading rollback with an example.
8. What are the two phases of 2PL?
9. How does Strict 2PL prevent cascading rollbacks?

10. What is a phantom read?
11. Define timestamp-based concurrency control.
12. How do RTS and WTS help maintain concurrency?
13. What is a dirty read?
14. Explain the concept of lost update.
15. How is recoverability ensured in transactions?
16. What causes deadlocks in transactions?
17. What is meant by the term "serializability"?
18. How does isolation contribute to transaction control?
19. Differentiate between non-repeatable read and phantom read.
20. Describe a real-world example where concurrency control is essential.

Comprehensive Questions (20 Questions)

1. Illustrate with an example the lifecycle of a transaction, including all transaction states.
2. Discuss the ACID properties of transactions with real-world examples.
3. Explain the significance of serializability in ensuring database consistency.
4. Compare and contrast serial and parallel schedules with examples.
5. Demonstrate how cascading rollback occurs using a scenario.
6. Describe the two-phase locking protocol (2PL) and its importance in concurrency control.
7. How does Strict 2PL ensure recoverability? Provide an example.
8. Discuss the timestamp-based concurrency control mechanism.
9. Analyze the pros and cons of locking mechanisms in databases.
10. Illustrate the impact of deadlocks and how they can be avoided.
11. Provide a detailed example of a phantom read.
12. Explain how lost updates occur and how they can be avoided.
13. What are the main challenges in concurrent execution of transactions?
14. Compare and contrast Strict 2PL and Rigorous 2PL.
15. How does concurrency control improve database performance?
16. Illustrate the concept of a dirty read with a step-by-step example.

17. What is meant by recoverability, and why is it critical in transactions?
18. Provide an example of a real-world application using timestamp ordering for concurrency.
19. Discuss the role of isolation levels in preventing concurrency-related problems.
20. Explain with a practical example how ACID properties are implemented in modern databases.

Answers

Fill in the Blanks Answers

1. *work*
2. *Atomicity*
3. *Concurrency control*
4. *serial*
5. *growing*
6. *consistency*
7. *dirty read*
8. *read*
9. *write*
10. *Strict 2PL*
11. *timestamp*
12. *deadlock*
13. *timestamp ordering*
14. *cascading*
15. *partially committed*
16. *Strict 2PL*
17. *durability*
18. *concurrency*
19. *Non-repeatable*
20. *Phantom*

MCQ Answers

1. *a) Atomicity*
2. *c) Consistency*
3. *c) To manage simultaneous transaction execution*

4. *b) Transactions execute one after the other.*
5. *b) Shared lock*
6. *b) Shrinking phase*
7. *b) Cyclic waiting on resources*
8. *b) Prevents cascading rollbacks*
9. *a) WTS*
10. *b) Deadlocks*