LET'S START WITH DBMS :)

Concurrency control mechanisms

Some common concurrency control mechanisms are :

- **Lock-Based Protocols**(2PL, Strict 2PL, Rigorous 2PL , Conservative 2PL)
- **Timestamp-Based Protocols** (Basic Timestamp Ordering (TO))
- **Optimistic Concurrency Control** (OCC)
- **Multiversion Concurrency Control** (MVCC)

LET'S START WITH DBMS :)

Concurrency control mechanisms

T1	T2
X(A)	
R(A)	
W(A)	
U(A)	
	S(B)
	R(B)
	U(B)

- **Lock-Based Protocols**

Types of Locks

a. **Binary Locks**: A simple mechanism where a data item can be either locked (in use) or unlocked.If a thread tries to acquire the lock when it's already locked, it must wait until the lock is released by the thread currently holding it.

b. **Shared and Exclusive Locks**

- **Shared Lock (S-lock):** Allows multiple transactions to read a data item simultaneously but prevents any of them from modifying it. Multiple transactions can hold a shared lock on the same data item at the same time.

- **Exclusive Lock (X-lock):** Allows a transaction to both read and modify a data item. When an exclusive lock is held by a transaction, no other transaction can read or modify the data item.

LET'S START WITH DBMS :)

Concurrency control mechanisms

Note : When a transaction acquires a shared lock on a data item, other transactions can also acquire shared locks on that same item, enabling concurrent reads. However, no transaction can acquire an exclusive lock on that item as long as one or more shared locks are held.

When a transaction acquires an exclusive lock on a data item, it has full control over that item, meaning it can both read and modify it. No other transaction can acquire a lock on the same data item until the exclusive lock is released.

Shared lock -> any no of transactions
Exclusive lock -> only one transaction should hold it at one time

While shared and exclusive locks are vital for maintaining data integrity and consistency in concurrent environments, they can introduce significant challenges in terms of performance, deadlocks, reduced concurrency, and system complexity.

LET'S START WITH DBMS :)

Concurrency control mechanisms

Drawbacks of shared-exclusive locks

Performance issues : Managing locks requires additional CPU and memory resources. The process of acquiring, releasing, and managing locks can introduce significant overhead

Concurrency issues : Exclusive locks prevent other transactions from accessing locked data, which can significantly reduce concurrency.

Starvation: Some transactions may be delayed if higher-priority transactions consistently acquire locks before them, leading to starvation where a transaction never gets to proceed.

Deadlocks : Shared and exclusive locks can lead to deadlocks, where two or more transactions hold locks that the other transactions need.

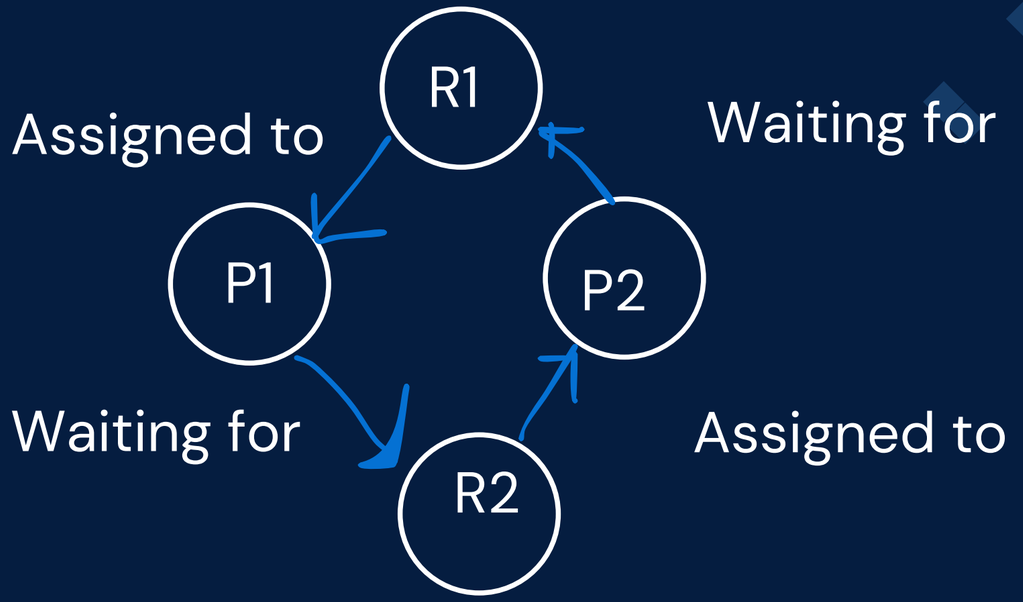
Irrecoverable : If Transaction B commits after the lock is release based on a modified value in transaction A which fails after sometime.

LET'S START WITH DBMS :)

Concurrency control mechanisms

Deadlock : It is a situtaion when 2 or more transactions wait for one another to give up the locks.

T1	T2
X(A)	
R(A)	
W(A)	
	X(B)
	R(B)
	W(B)
X(A)	
R(A)	
W(A)	
	X(B)
	R(B)
	W(B)



LET'S START WITH DBMS :)

Concurrency control mechanisms

Two-Phase Locking (2PL) : This protocol ensures serializability by dividing the execution of a transaction into two distinct phases

- **Growing Phase:** A transaction can acquire locks but cannot release any. This phase continues until the transaction has obtained all the locks it needs.

- **Shrinking Phase:** After the transaction releases its first lock, it can no longer acquire any new locks. During this phase, the transaction releases all the locks it holds.

T1	T2
X(A)	
R(A)	
W(A)	
	R(A)
S(B)	
R(B)	
U(A)	
U(B)	

Any transaction which is following 2PL locking achieves serializability and consistency.

LET'S START WITH DBMS :)

Concurrency control mechanisms

Two-Phase Locking (2PL)

Advantages :

1. It guarantees that the schedule of transactions will be serializable, meaning the results of executing transactions concurrently will be the same as if they were executed in some serial order.
2. By ensuring that transactions are serializable, 2PL helps maintain data integrity and consistency, which is critical in environments where data accuracy is essential.

Disadvantages :

1. Deadlocks, starvation and cascading rollbacks
2. Transactions must wait for locks to be released by other transactions. This can lead to increased waiting times and lower system throughput.
3. In case of a system failure, recovering from a crash can be complex

LET'S START WITH DBMS :)

Concurrency control mechanisms

Strict Two-Phase Locking (Strict 2PL):

A stricter variant where exclusive locks are held until the transaction commits or aborts. This helps prevent cascading rollbacks (where one transaction's rollback causes other transactions to roll back).

T1	T2
X(A)	
R(A)	
W(A)	
	R(A)
Commit	
U(A)	

Advantages:

- Prevents Cascading Aborts
- Ensures Strict Serializability

Disadvantages:

- Since write locks are held until the end of the transaction, other transactions may be blocked for extended periods
- Transactions may experience longer wait times to acquire locks
- Deadlocks and starvation is there

LET'S START WITH DBMS :)

Concurrency control mechanisms

Rigorous Two-Phase Locking:

An even stricter version where all locks (both shared and exclusive) are held until the transaction commits. This guarantees strict serializability.

Advantages:

- Since all locks are held until the end of the transaction, the system can easily ensure that transactions are serializable and can be recovered
- Prevents Cascading Aborts and Dirty Reads
-

Disadvantages:

- Performance bottlenecks
- Increased Transaction Duration
- Deadlocks and starvation is there

LET'S START WITH DBMS :)

Concurrency control mechanisms

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

Conservative Two-Phase Locking:

Conservative Two-Phase Locking(Static Two-Phase Locking) is a variant of the standard 2PL protocol that aims to prevent deadlocks entirely by requiring a transaction to acquire all the locks it needs before it begins execution.

If the transaction is unable to acquire all the required locks (because some are already held by other transactions), it waits and retries. The transaction only starts execution once it has successfully acquired all the necessary locks.

Since a transaction never starts executing until it has all the locks it needs, deadlocks cannot occur because no transaction will ever hold some locks and wait for others

In this scenario, deadlocks cannot occur because neither T1 nor T2 starts execution until it has all the locks it needs.

LET'S START WITH DBMS :)

Concurrency control mechanisms

Timestamp-Based Protocols: It assign a unique timestamp

- Every transaction is assigned a unique timestamp when it enters the system. It is used to order the transactions based on their Timestamps.

- There are two important timestamps for each data item:
  - **Read Timestamp (RTS):** The last timestamp of any transaction that has successfully read the data item.

  - **Write Timestamp (WTS):** The last timestamp of any transaction that has successfully written the data item.

Transaction with smaller timestamp(Old) -> Transaction with larger timestamp(young)

LET'S START WITH DBMS :)

Concurrency control mechanisms

Timestamp-Based Protocols: It assign a unique timestamp

Rules : Consider if there are transactions performed on data item A.

R_TS(A) -> Read time-stamp of data-item A.

W_TS(A)-> Write time-stamp of data-item A.

1.Check the following condition whenever a transaction Ti issues a Read (X) operation:

- If W_TS(A) >TS(Ti) then the operation is rejected. (rollback Ti)
- If W_TS(A) <= TS(Ti) then the operation is executed. (set R_TS(A) as the max of (R_TS(A), TS(Ti)

2. Check the following condition whenever a transaction Ti issues a Write(X) operation:

- If TS(Ti) < R_TS(A) then the operation is rejected. (rollback Ti)
- If TS(Ti) < W_TS(A) then the operation is rejected and Ti is rolled back otherwise the operation is executed. Set W_TS(A)=TS(Ti)

LET'S START WITH DBMS :)

Concurrency control mechanisms

Timestamp-Based Protocols: It assign a unique timestamp

Let's assume there are two transactions T1 and T2. Suppose the transaction T1 has entered the system at 7:00PM and transaction T2 has entered the system at 7:05pm then T1 has the higher priority, so it executes first as it is entered the system first. T1->T2

If younger has done a Read/Write and older wants to do Read/Write, thats a rollback case.

T1	T2
R(A)	
	W(A)

T1	T2
W(A)	
	R(A)

T1	T2
W(A)	
	W(A)

T1	T2
	R(A)
W(A)	

T1	T2
	W(A)
W(A)	

T1	T2
	W(A)
R(A)	