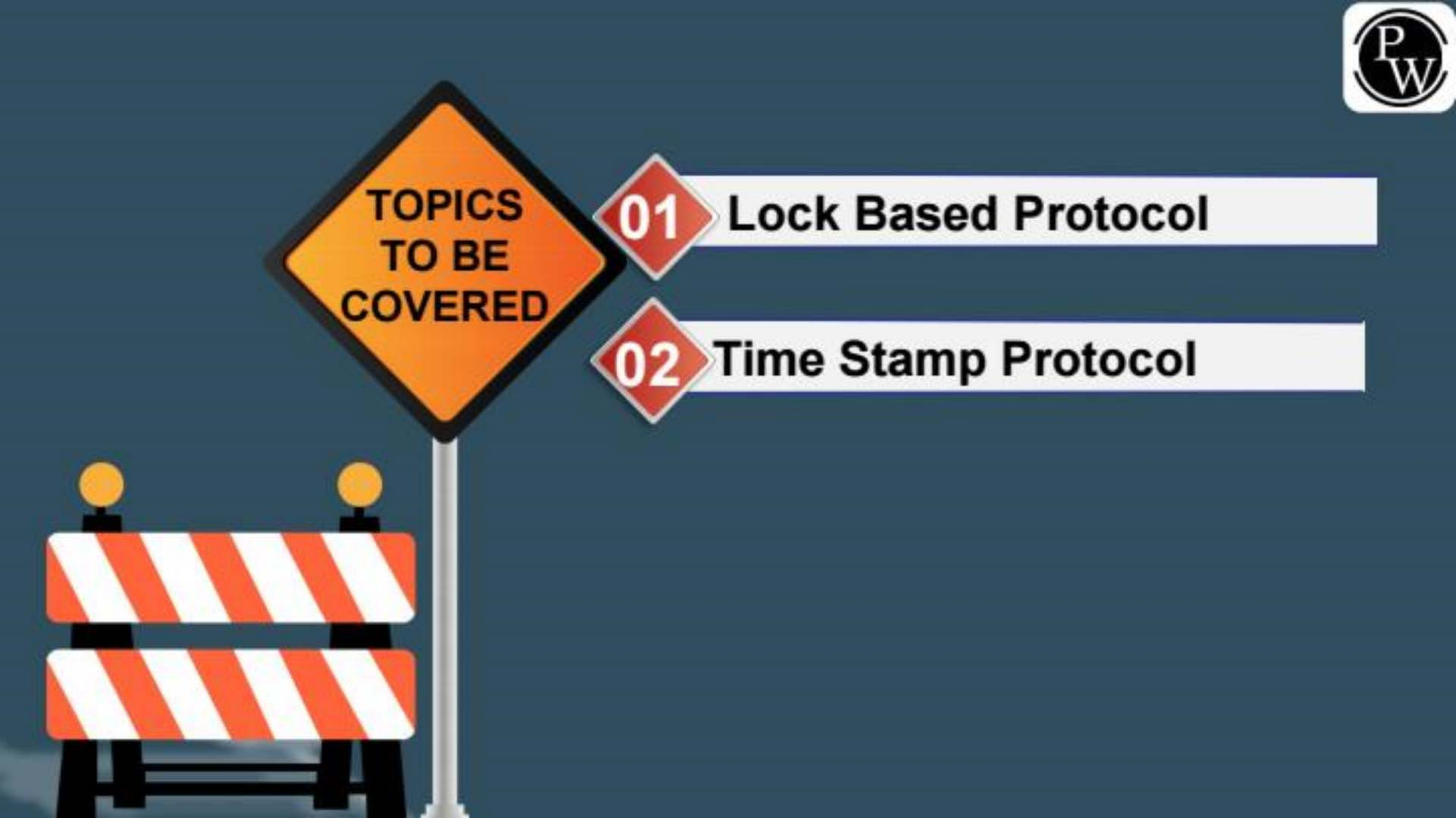
COMPUTER SCIENCE

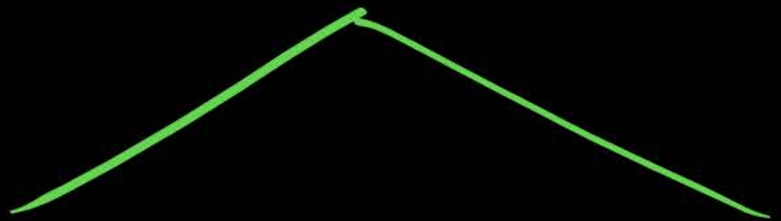
Database Management System

Transaction & Concurrency Control

Lecture_7

Vijay Agarwal sir


TOPICS
TO BE
COVERED
01
Lock Based Protocol
02
Time Stamp Protocol



Serializability Recoverability

Problem due to Concurrent Execution

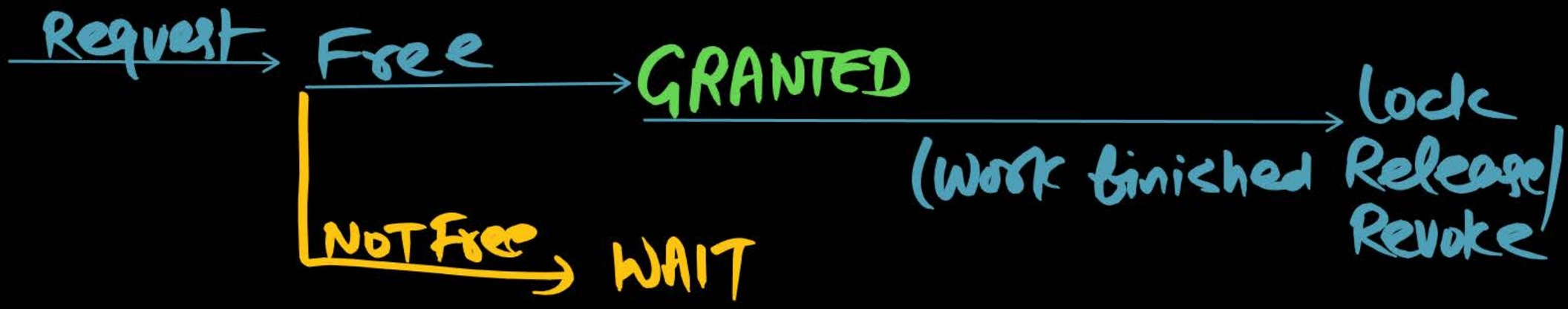
Topological Sorting

Finding Number of Conflict serializable schedule.

Implementation of Concurrency Control.

LOCK BASED Protocol.

LOCK BASED Protocol



Procedure

Before Using Any Data Item, Transaction should Request for a Lock,
If Lock is Free (Lock Not taken by any other Transaction on that Data Item)
then it will be granted, otherwise Transaction has to Wait (if Busy)

① SHARED LOCK [S] [Only Read]

② Exclusive Lock [X] [write(A) / write(A) Read(A) / Read(A) write(A)]

SHARED LOCK [S]

```
Lock_S(A)
  Read(A)
Unlock_S(A)
```

Exclusive Lock [X]

```
Lock_X(A)
write(A) (or) Read(A) write(A) (or) write(A) Read(A)
Unlock_X(A)
```

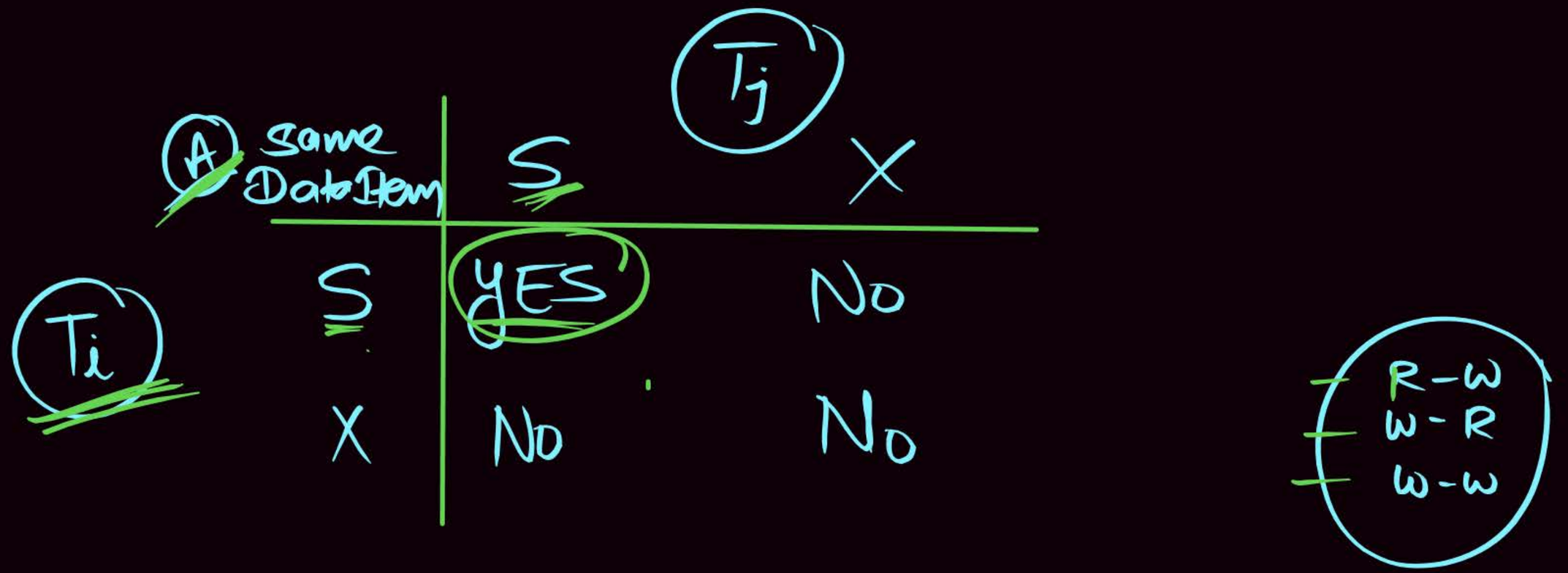
Lock-Based Protocols

- A lock is a mechanism to control concurrent access to a data item
- Data items can be locked in two modes:

  1. exclusive (X) mode. Data item can be both reads as well as written. X-lock is requested using **lock-X** instruction.

  2. Shared (S) mode. Data item can only be read. S-lock is requested using **lock-S** instruction.

- Lock requests are made to concurrency-control manager. Transaction can proceed only after request is granted.

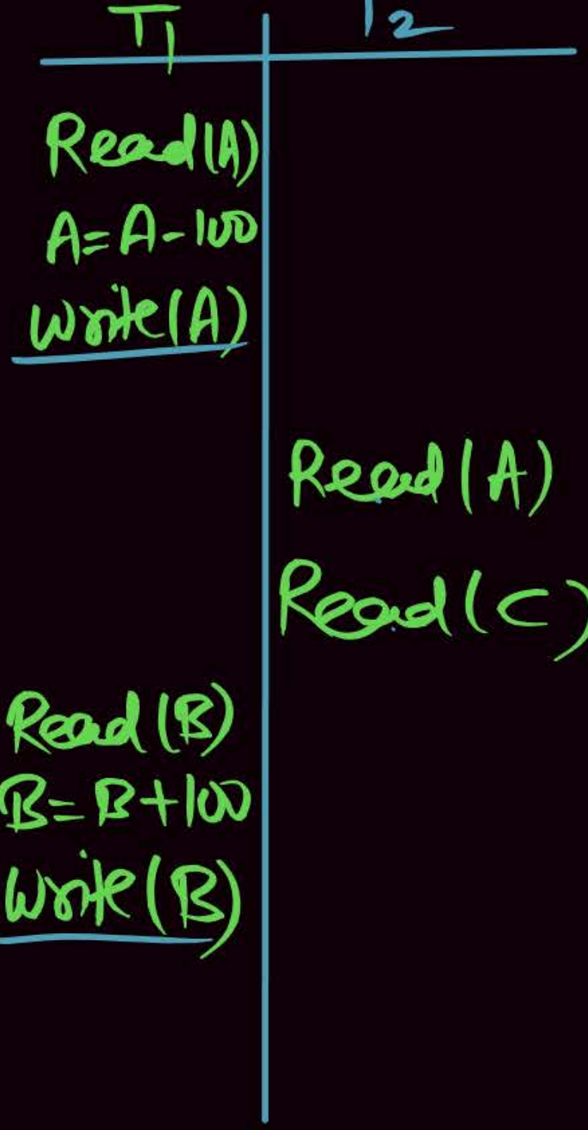

Tj
A
same DataItem
S
X
Ti
S
YES
No
X
No
No
R-W
W-R
W-W

Lock-Based Protocols (Cont.)

- Lock-compatibility matrix

DATA ITEM (A)	S	X
S	true	false
X	false	false

- A transaction may be garneted a lock on an item if the requested lock is compatible with lock 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 loc on the item.


T1
T2
Read(A)
A=A-100
Write(A)
Read(A)
Read(C)
Read(B)
B=B+100
Write(B)


T1
T2
Lock Manager
Lock-X(A)
Read(A)
A=A-100
Write(A)
Unlock-X(A)
GRANT-X(A, T1)
Release/
Revoke-X(A, T1)
Lock-S(A)
Read(A)
Unlock-S(A)
GRANT-S(A, T2)
Revoke-S(A, T2)
Lock-S(C)
Read(C)
Unlock-S(C)
GRANT-S(C, T2)
Revoke-S(C, T2)
Lock-X(B)
Read(B)
B=B+100
Write(B)
Unlock-X(B)
GRANT-X(B, T1)
Revoke-X(B, T1)

Schedule with Lock Grants

- A locking protocol is a set of rules followed by all transactions while requesting and releasing locks.
- Locking protocols enforce serializability by restricting the set of possible schedules.

T_1	T_2	Concurrency-control manager
lock-X(B)		
		grant-X(B, T_1)
read(B)		
B:=B – 50		
write(B)		
unlock(B)		Revoke-X(B, T_1)
	lock-S(A, T_2)	
		grant-S(A, T_2)
	read(A)	
	unlock(A)	
	lock-S(B)	
		grant-S(B, T_2)
	read(B)	
	unlock(B)	
	display(A+B)	
lock-X(A)		
		grant-X(A, T_1)
read(A)		
A:=A + 50		
write(A)		
unlock(A)		

Two Phase locking Protocol [2PL]

Lock & Unlock Request done in Two Phase

1. Growning Phase (Acquire Locks)
2. Shrinking Phase (Release lock)

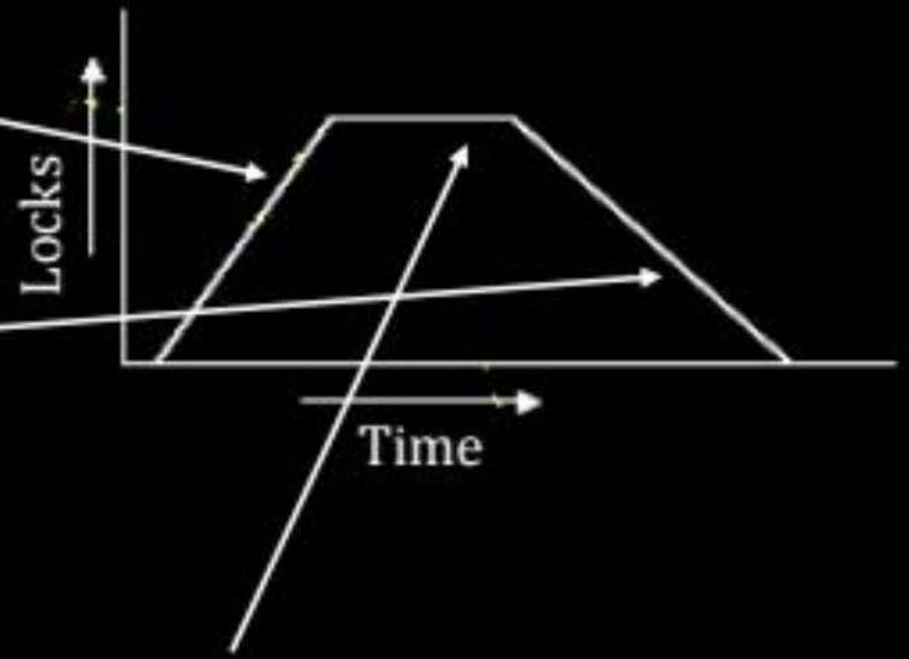
Note: Each Transaction first finish its Growing Phase then start Shrinking Phase.

① Gowning phase : In Growning Phase, Transaction May Obtain the Lock But Must Not Release Any Lock.

② shrinking Phase : In Shrinking Phase, Transaction Release the Lock But Must Not ask (Request) for Any New Lock.

The Two-Phase Locking Protocol

- A protocol which ensures conflict-serializable schedules.
- Phase 1: Growing Phase
  - Transaction may obtain locks
  - Transaction may not release lock
- Phase 2: Shrinking Phase
  - Transaction may release locks
  - Transaction may not obtain locks



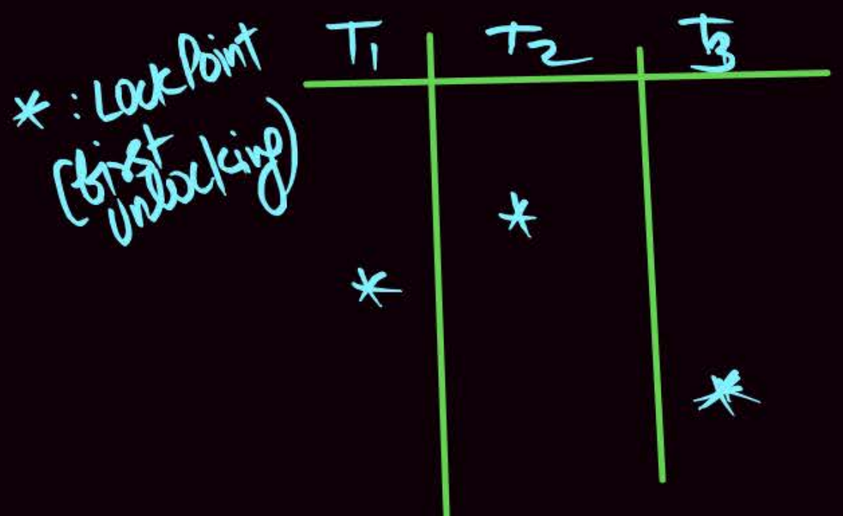
- The protocol assures serializability. It can be proved that transactions can be serialized in the order of their lock points (i.e., the point where a transaction acquired its final lock).

→ Serializability order.

LOCK POINT : Position of last lock operation (or) First Unlock operation.

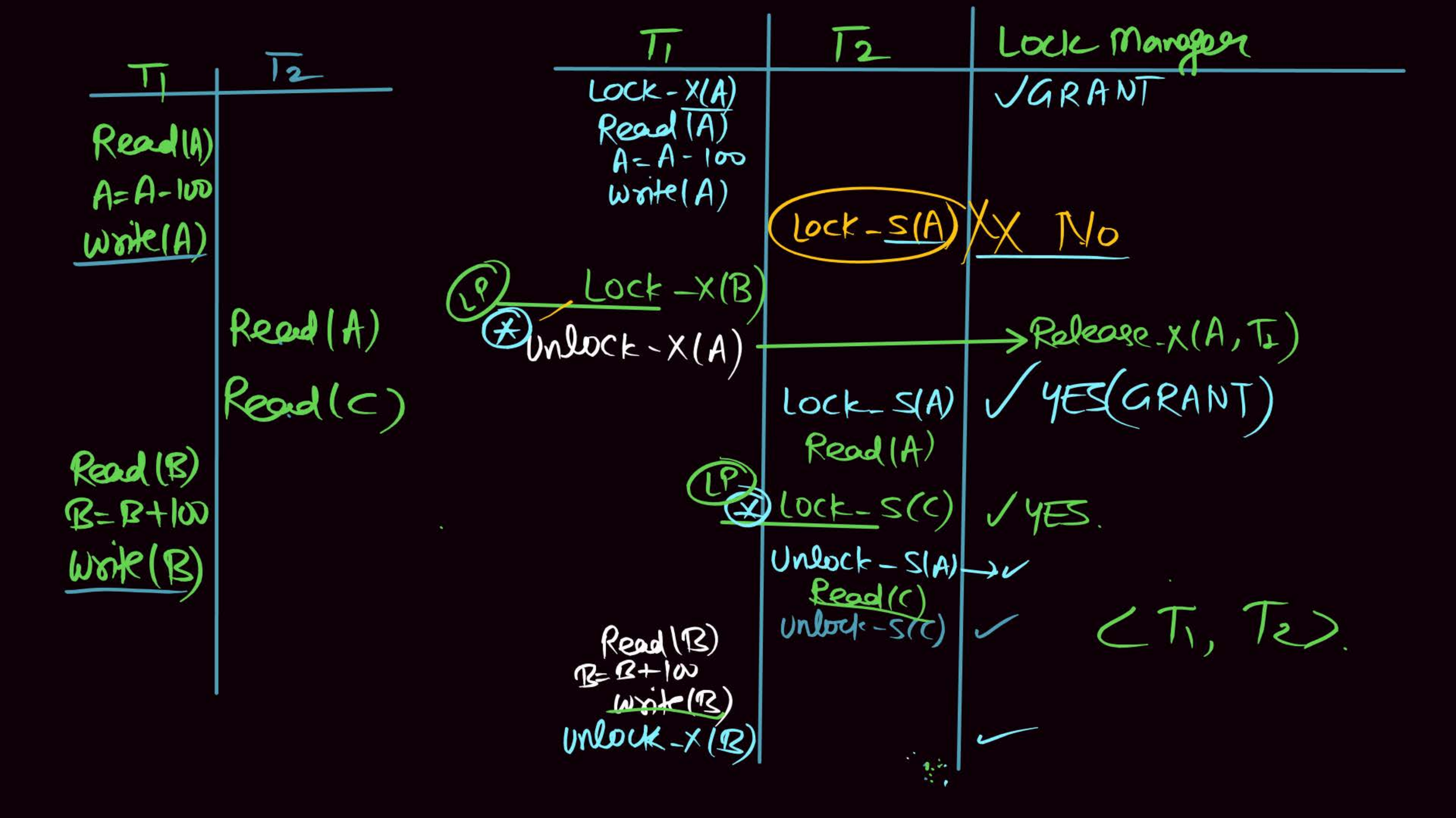
(OR)

Position from where Shrinking Phase of the transaction start.



Serializability order.
$\langle T_2, T_1, T_3 \rangle$

∴


T1
T2
Read(A)
A=A-100
Write(A)
Read(A)
Read(C)
Read(B)
B=B+100
Write(B)
T1
T2
Lock Manager
Lock-X(A)
Read(A)
A=A-100
Write(A)
✓GRANT
Lock-S(A)
XX No
LP
X
Lock-X(B)
Unlock-X(A)
Release-X(A, T1)
Lock-S(A)
✓ YES(GRANT)
Read(A)
LP
X
Lock-S(C)
✓ YES.
Unlock-S(A)
✓
Read(C)
Unlock-S(C)
✓
< T1, T2 >.
Read(B)
B=B+100
Write(B)
Unlock-X(B)
✓

Important Point about 2PL.

① 2PL Ensure Conflict Serializability (Serializability)

If a Schedule is allowed (followed) by 2PL then its Ensure Conflict Serializable Schedule (Serializability).

② Serializability Order determined by Lock Point

Important Point about 2PL.

③ 2PL Not Ensure Recoverablity.

Irrecoverable Schedule followed by 2PL.

T_1	T_2
R(A)	
W(A)	
	R(A)
	Commit
Commit	

Irrecoverable Schedule

T_1	T_2
X(A)	
R(A)	
W(A)	
Unlock-X(A)	
	S(A)
	R(A)
	Unlock-S(A)
	Commit
Commit	

Irrecoverable Schedule
But allowed (followed) by 2PL.

Important Point about 2PL.

(4) 2PL Suffers from (May Not Free From) Deadlock.

(1)

T_1	T_2
W(A)	
	R(B)
W(B)	
	R(A)

T_1	T_2
X(A)	
W(A)	
	S(B)
Denied by T_2 ↳ X(B)	R(B)
	S(A) ← T_1 Denied by T_1
Deadlock	

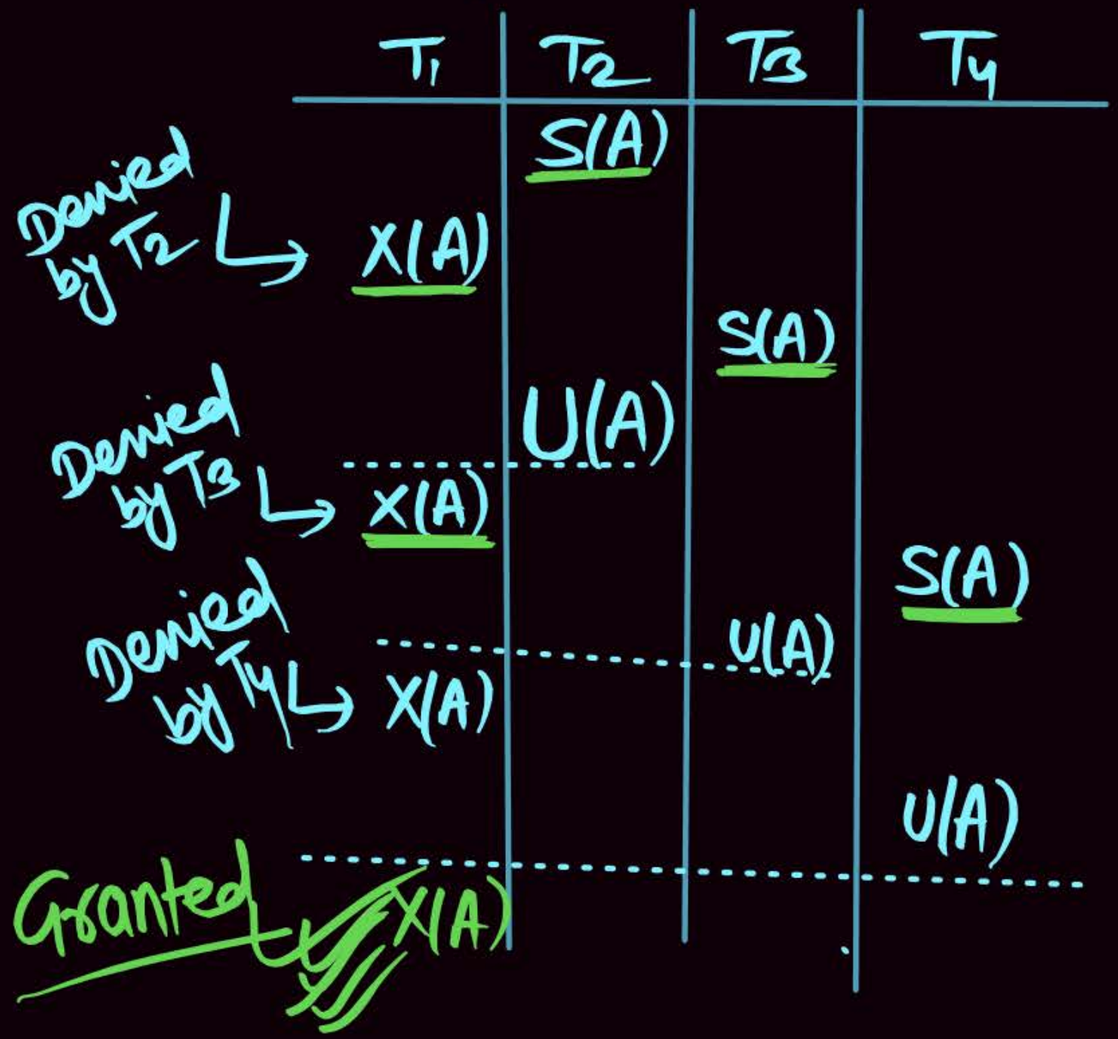
(2)

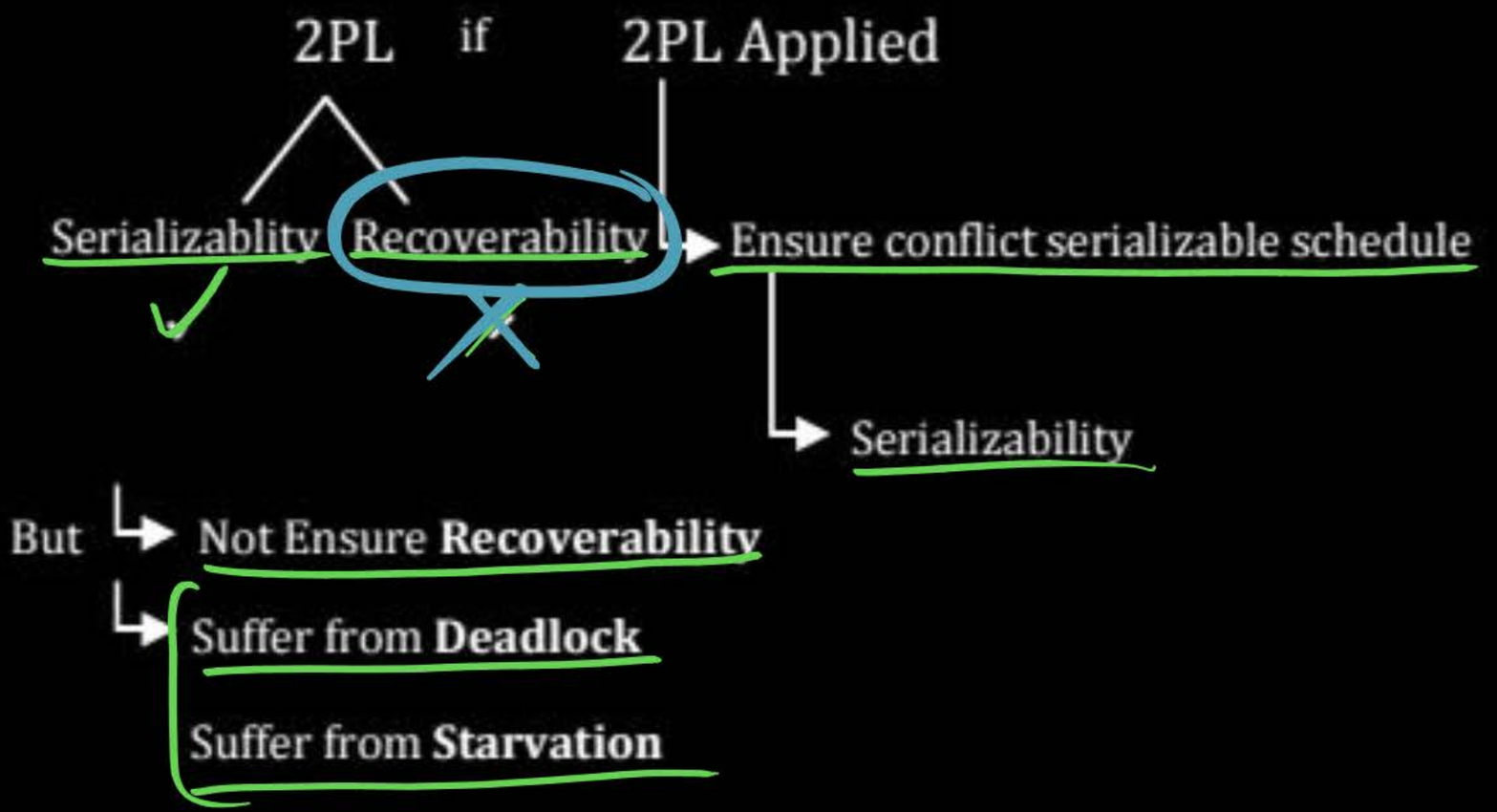
T_1	T_2
R(A)	
	W(B)
R(B)	
	W(A)

T_1	T_2
S(A)	
R(A)	
	X(B)
	W(B)
Denied by T_2 ↳ S(B)	
X(A) ← T_1 Denied by T_1	
Deadlock	

Important Point about 2PL.

(5) 2PL Suffer from Starvation.




2PL if 2PL Applied
Serializablity
Recoverability
Ensure conflict serializable schedule
Serializability
But
Not Ensure Recoverability
Suffer from Deadlock
Suffer from Starvation

STRICT 2PL : 2PL + ALL Exclusive (X Lock) Lock taken by the transaction

Must be Held Untill Commit/Rollback

(OR) X Lock Release After Commit/ROLLBACK

Strict Recoverable

- Conflict Serializable
- Recoverable Schedule
- Cascadeless (No Cascading Rollback)
- Strict Recoverable

Suffer from
- Deadlock
- Starvation

Rigorious 2PL : (2PL) + All Locks [Shared [S] & Exclusive [X]

Must be Hold by the transaction

Untill Commit / Rollback

(OR)

2PL + ALL lock [S & X] Release After Commit

Rigorious 2PL

↳ suffer from Deadlock

↳ Starvation.

The Two-Phase Locking Protocol (Cont.)

- Two-phase locking does not ensure freedom from deadlocks
- Extensions to basic two-phase locking needed to ensure recoverability of freedom from cascading roll-back
  - Strict two-phase locking: a transaction must hold all its exclusive locks till it commits/aborts.
    - Ensures recoverability and avoids casecading roll-backs
  - Rigorous two-phase locking: a transaction must hold all locks till commit/abort.
    - Transactions can be serialized in the order in which they commit.
- Most databases implement rigorous two-phase locking, but refer to if as simply two-phase locking

Conservative 2PL :· Each transaction Acquire ALL the Lock in the beginning (at the Start of Execution & Release After Commit.

Ensure

- Conflict Serializability
- Recoverability
- Cascadeless
- No Cascading Rollback
- Strict Recoverable
- No Deadlock (Freedom From Deadlock)

But suffer from Starvation.

T_1	T_1	T_1	T_1	T_1
1	2	3	4	5
Lock-S(A)	Lock-S(A)	Lock-S(A)	Lock-S(A)	Lock-S(A)
R(A)	R(A)	R(A)	Lock-X(B)	R(A)
Lock-X(B)	Lock-X(B)	Lock-X(B)	R(A)	Unlock(A)
R(B)	Unlock(A)	R(B)	R(B)	Lock-X(B)
Unlock(A)	R(B)	W(B)	W(B)	R(B)
W(B)	W(B)	Commit	Commit	W(B)
Unlock(B)	Commit	Unlock(A)	Unlock(A)	Unlock(B)
Commit	Unlock(B)	Unlock(B)	Unlock(B)	Commit

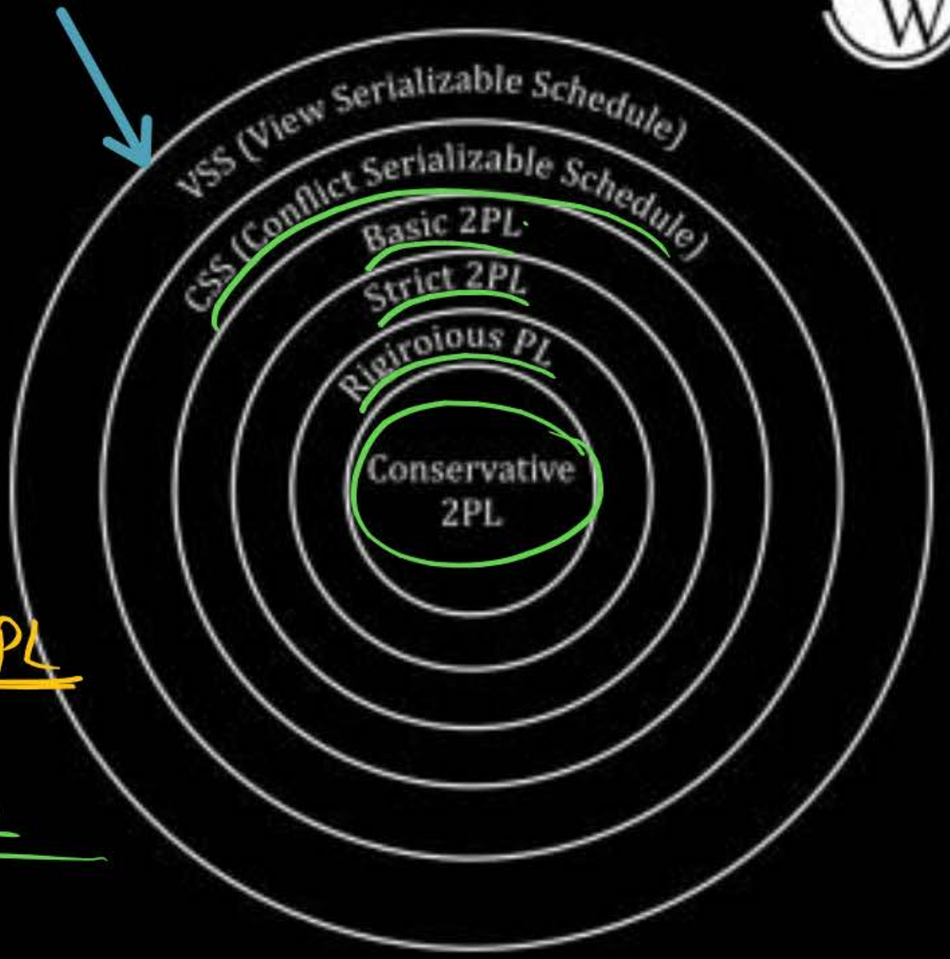
Basic 2PL

Strict 2PL (Basic 2PL)

Rigorious 2PL Strict & Basic 2PL

Conservative 2PL Rigorious, Strict & Basic 2PL

Not 2PL



T_1	T_2
R(A)	
	R(A)
R(B)	
W(B)	

TIMESTAMP BASED CONCURRENCY CONTROL

Timestamp-Based Protocols

- Each transaction T_i is issued a timestamp $TS(T_i)$ when it enters the system.
  - Each transaction has a unique timestamp
  - Newer transaction have timestamp strictly greater than earlier ones
  - Timestamp could be based on a logical counter
    - Real time may not be unique
    - Can use (wall-clock time, logical counter) to ensure
- Timestamp-based protocols manage concurrent execution such that **time-stamp order = serializability order**
- Several alternative protocols based on timestamps

Timestamp-Based Protocols

The **timestamp ordering (TSQ) protocol**

- Maintains for each data Q two timestamp values:
  - **W-timestamp**(Q) is the largest time-stamp of any transaction that executed write (Q) successfully.
  - **R-timestamp** (Q) is the largest time-stamp of any transaction that executed **read** (Q) successfully.
- Imposes rules on read and write operations to ensure that
  - any conflicting operations are executed in timestamp order
  - out of order operations cause transaction rollback

I: T_i – Read(Q) (Transaction T_i Issue R(Q) Operation)

(i) If TS (T_i) < WTS (Q): Read operation Reject & T_i Rollback.

(ii) If TS (T_i) ≥ WTS(Q): Read operation is allowed

and Set Read – TS(Q) = max[RTS(Q), TS (T_i)]

II: T_i – Write(Q) (Transaction T_i Issue Write(Q) Operation)

(i) If TS (T_i) < RTS (Q): Write operation Reject & T_i Rollback.

(ii) If TS (T_i) < WTS(Q): Write operation Reject & T_i Rollback.

(iii) Otherwise execute write (Q) operation

Set Read WTS(Q) = TS (T_i)

Timestamp-Based Protocols (Cont.)

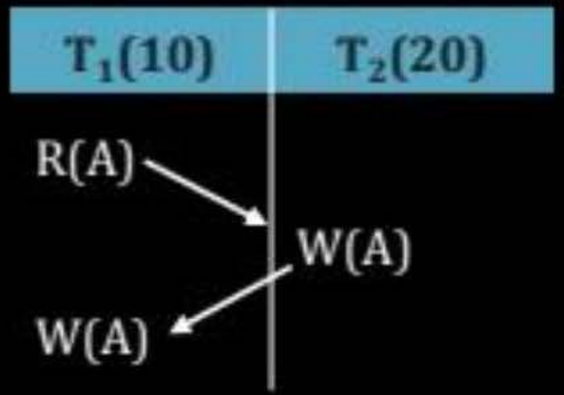
- Suppose a transaction T_i issues a **read** (Q)
  1. If $TS(T_i) \leq$ **W**-timestamp (Q), then T_i needs to read a value of Q that was already overwritten.
     - Hence, the **read** operation is rejected, and T_i is rolled back.
  2. If $TS(T_i) \geq$ **W**-timestamp (Q), then the **read** operation is executed, and R-timestamp(Q) is set to.

     max(R-timestamp (Q), TS (T_i)).

Timestamp-Based Protocols (Cont.)

- Suppose a transaction T_i issues **write**(Q)
  1. If $TS(T_i) < $ **R**-timestamp(Q), then the value of Q that T_i is producing was needed previously, and the system assumed that the value would never be produced.
     - Hence, the **write** operation is rejected, and T_i is rolled back.
  2. If $TS(T_i) <$ **W**-timestamp (Q), then T_i is attempting to write an obsolete value of Q.
     - Hence, this **write** operation is rejected, and T_i is rolled back.
  3. Otherwise, the **write** operation is executed, and W-timestamp(Q) is set to $TS(T_i)$.

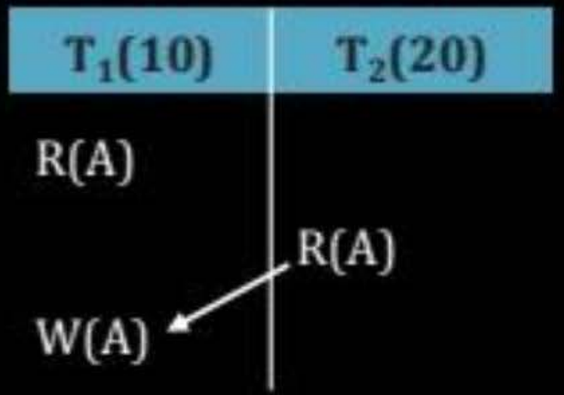
(1)



$TS(T_1) < TS(T_2)$

$T_1 \rightarrow T_2$

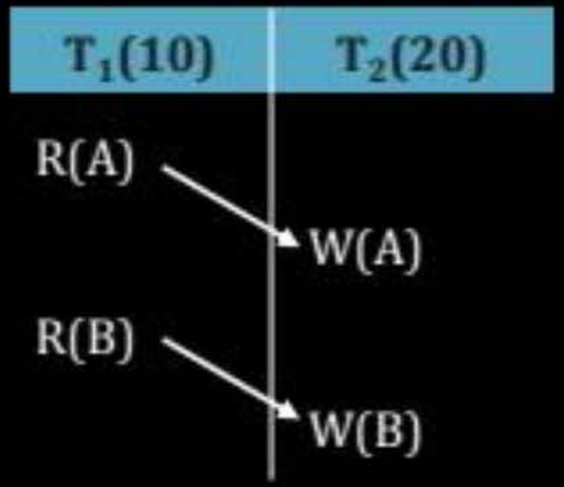
(2)



$TS(T_1) < TS(T_2)$

$T_1 \rightarrow T_2$

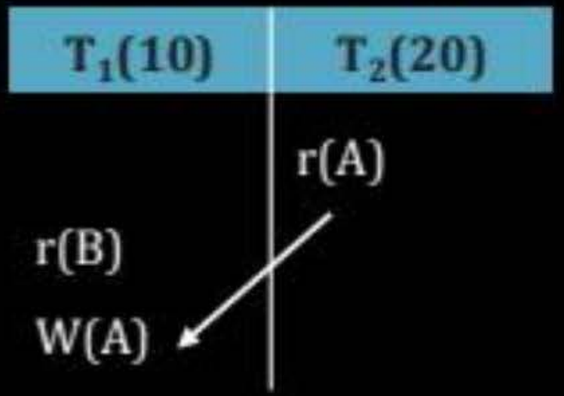
(3)



$TS(T_1) < TS(T_2)$

$T_1 \rightarrow T_2$

(4)



$TS(T_1) < TS(T_2)$

$T_1 \rightarrow T_2$

Thomas' Write Rule

- Modified version of the timestamp-ordering protocol in which obsolete **write** operations may be ignored under certain circumstances.
- When T_i attempts to write data item Q, if $TS(T_i) < W\text{-timestamp}(Q)$, then T_i is attempting to write an obsolete value of {Q}.
  - Rather than rolling back T_i as the timestamp ordering protocol would have don, this {**write**} operation can be ignored.
- Otherwise this protocol is the same as the timestamp ordering protocol.
- Thomas' Write Rule allows greater potential concurrency.
  - Allows some view-serializable schedules that are not conflict-serializable.

Thomas Write Rule (View Serializability)

1. TS (T_i) < RTS(Q) : Rollback
2. TS(T_i) < WTS(Q) : Write operation is Ignored and No Roll back

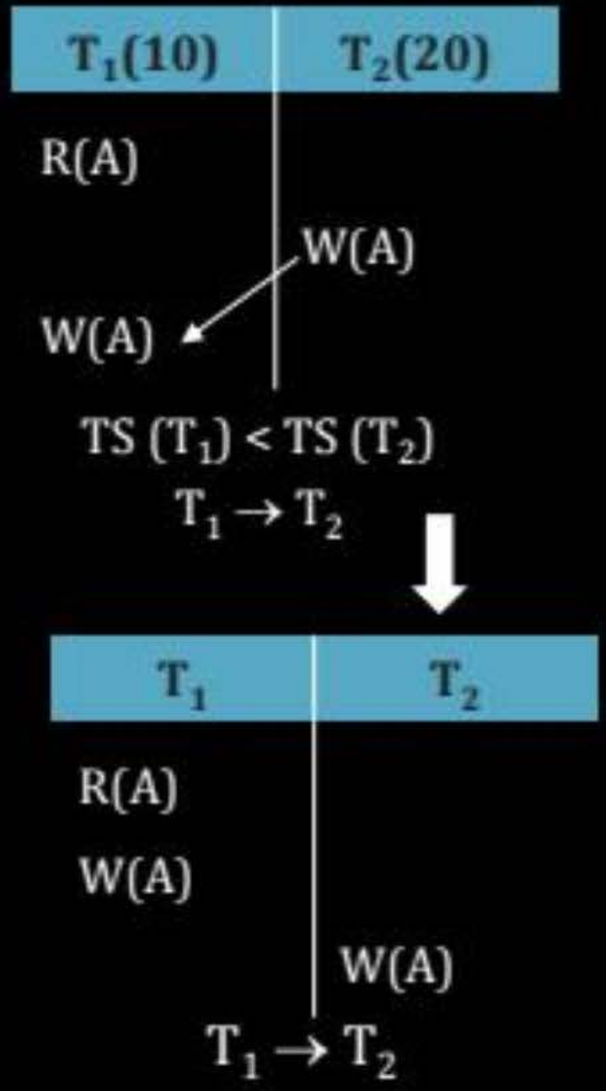
Same as TSP

Time Stamp Protocol: Ensure serializability deadlock free but starvation possible

Deadlock Prevention Algorithm

(1) Wait-Die (2) Wound-wait

Older Younger



Q.

In which one of the following Lock Scheme Deadlock cannot occur?

[MCQ]

A. Basic 2PL

B. Strict 2PL

C. Conservative 2PL

D. Rigorous 2PL

Q.

Consider the following statement about lock–based protocol

(A) 2 PL (2phase locking) protocol Ensure view serializability

(B) 2PL ensure recoverability &No cascading rollback.

(C) Strict 2 PL ensure recoverability & no cascading rollback.

(D) Strict 2 PL avoids deadlock (not suffering from deadlock).

How many numbers of above statement are correct?

[MCQ]

A 1

B 2

C 3

D 4

Q.

[MSQ]

Consider the following Schedule:

$r_1(x)\ r_2(y)\ r_2(x)\ w_1(z)\ r_1(y)\ w_3(y)\ r_3(z)\ w_2(y)\ w_3(x)$

which of the following time stamp ordering Not allows to execute schedule using Thomas Write rule time stamp Ordering Protocol?

A $(T_1, T_2, T_3) = (20, 30, 10)$

B $(T_1, T_2, T_3) = (10, 20, 30)$

C $(T_1, T_2, T_3) = (10, 30, 20)$

D $(T_1, T_2, T_3) = (30, 20, 10)$

MCQ

Consider the following two statements about database transaction schedules:

I. Strict two-phase locking protocol generates conflict serializable schedules that are also recoverable.

II. Timestamp-ordering concurrency control protocol with Thomas Write Rule can generate view serializable schedules that are not conflict serializable.

Which of the above statements is/are True?

[GATE-2019-CS: 1M]

A. I only

B. II only

C. Both I and II

D. Neither I nor II

MCQ

Consider the following partial Schedule S involving two transactions T1 and T2. Only the read and the write operations have been shown. The read operation on data item P is denoted by read (P) and the write operation on data item P is denoted by write (P).

Suppose that the transaction T1 fails immediately after time instance 9. Which one of the following statements is correct?

Time	Transaction-id	
	T 1	T 2
1.	read(A)	
2.	write(A)	
3.		read(C)
4.		write(C)
5.		read(B)
6.		write(B)
7.		read(A)
8.		commit
9.	read(B)	

[GATE-2015-CS: 2M]

A. T2 must be aborted and then both T1 and T2 must be restarted to ensure transaction atomicity

B. Schedule S is non recoverable and cannot ensure transaction atomicity

C. One T2 must be aborted and then restarted to ensure transaction atomicity

D. Schedule S is recoverable and can ensure atomicity and nothing else needs to be done

MCQ

Which of the following concurrency control protocols ensure both conflict serializability and freedom from deadlock?

I. 2-phase locking

II. Time-stamp ordering

[GATE-2010-CS: 1M]

A I only

B II only

C Both I and II

D Neither I nor II

MCQ

Consider a simple checkpointing protocol and the following set of operations in the log.
(start, T4); (write, T4, y, 2, 3); (start, T1);
(commit, T4); (write, T1, z, 5, 7);
(checkpoint);
(start, T2); (write, T2, x, 1, 9); (commit, T2);
(start, T3); (write, T3, z, 7, 2);
If a crash happens now and the system tries to recover using both undo and redo operations, what are the contents of the undo list and the redo list? [GATE-2015-CS: 2M]

After checkPoint

Commit T_2 : Redo T_2

Undo : T_3 & T_1.

A. Undo: T3, T1; Redo: T2

B. Undo: T3, T1; Redo: T2, T4

C. Undo: none; Redo: T2, T4, T3, T1

D. Undo: T3, T1, T4; Redo: T2

Those Transaction Commit before Check Point Not Required Any Redo operation.

Those Transaction Commit After Check point Require Redo operation

Those Transaction Not Commit Require Undo operation.

If T_I Commit then Redo.

If Any Transaction that Not Commit Require UNDO operation.

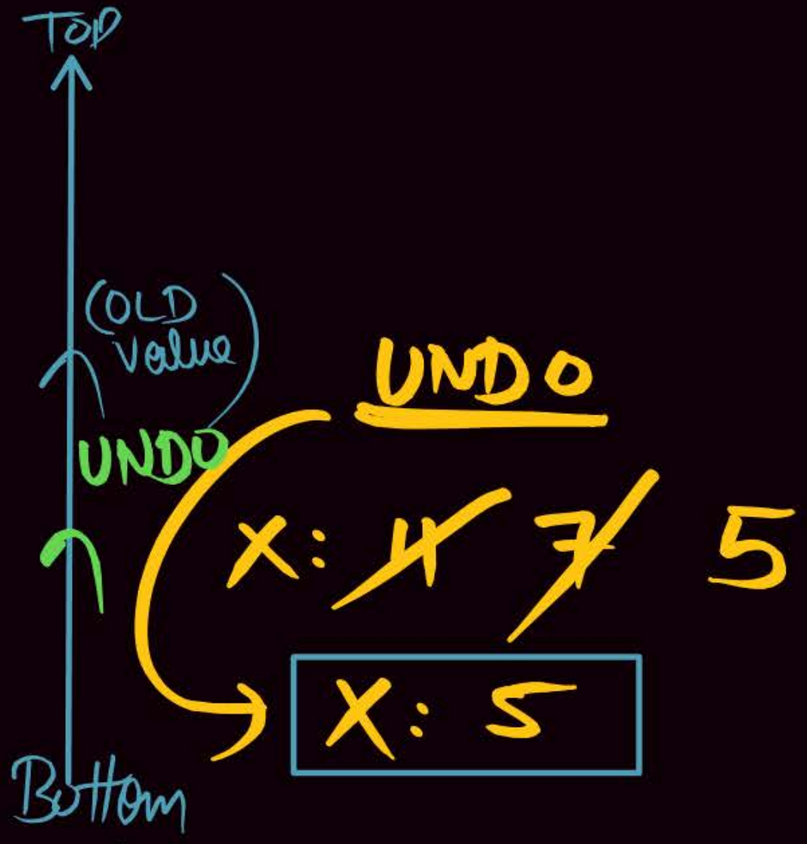
Log Based Recovery

Redo & Undo Concept

Transaction Number (operation , Data Item , (OV) OLD Value , (NU) New Value)

(X , 5 , 7)

(X , 7 , 11)



Redo

X: 5 7 11

X = 11

Q.

Consider the following database schedule with two transactions, T_1 and T_2.

$S = r_2(X); r_1(X); r_2(Y); w_1(X); r_1(Y); w_2(X); a_1; a_2$

where $r_i(Z)$ denotes a read operation by transaction T_i on a variable Z, $w_i(Z)$ denotes a write operation by T_i on a variable Z and a_i denotes an abort by transaction T_i

Which one of the following statements about the above schedule is TRUE?

[MCQ:2016–2M]

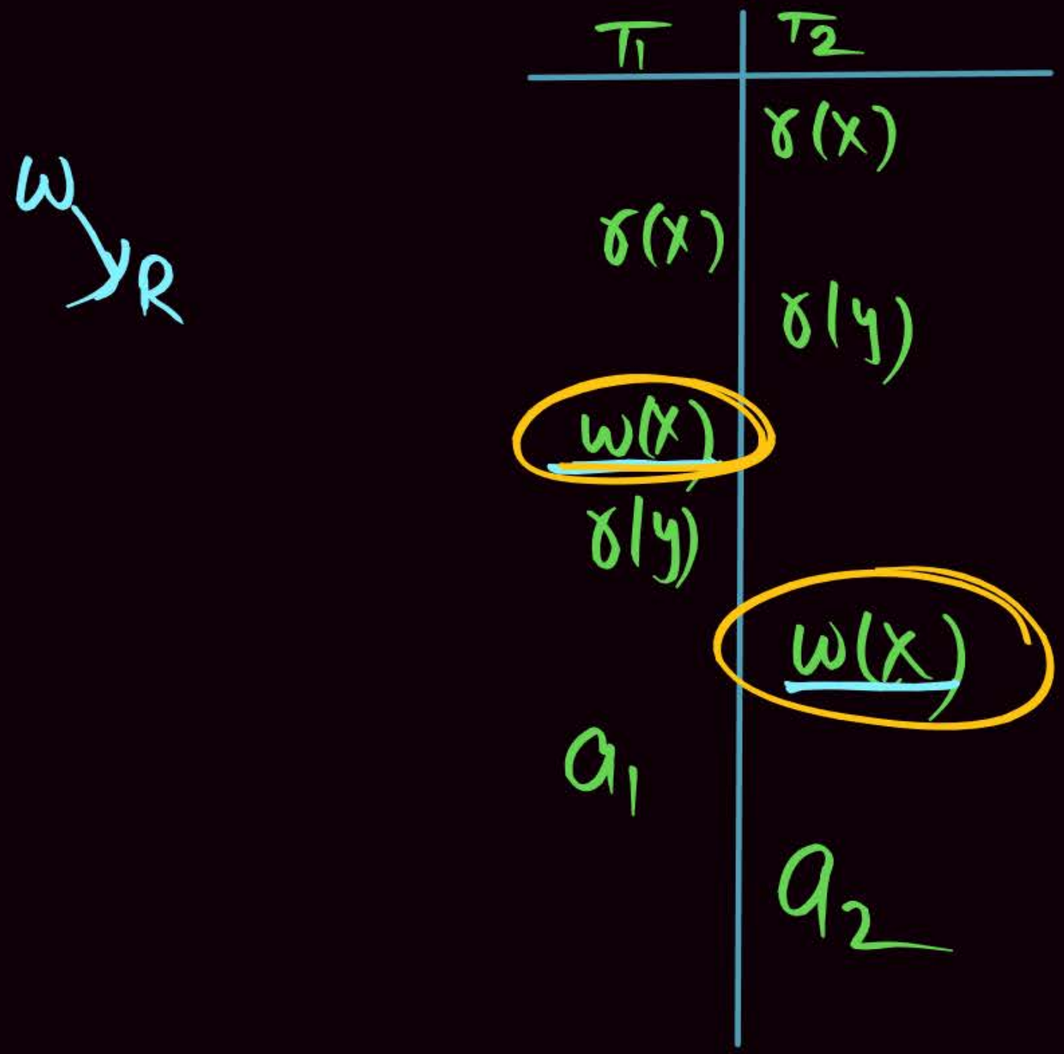
A. S is non-recoverable

B. S is recoverable, but has a cascading abort

C. S does not have a cascading abort

D. S is strict

$r_2(x)\ r_1(x)\ r_2(y)\ w_1(x)\ r_1(y)\ w_2(x)\ a_1\ a_2$



✓ Recoverable

✓ Cascadeless
(No Cascading Rollback)

✗ Strict Recoverable

Q.

Let S be the following schedule of operations of three transactions T_1, T_2 and T_3 in a relational database system:

$$R_2(Y), R_1(X), R_3(Z), R_1(Y), W_1(X), R_2(Z), W_2(Y), R_3(X), W_3(Z)$$

Consider the statements P and Q below:

P: S is conflict-serializable.

Q: If T_3 commits before T_1 finishes, then S is recoverable.

Which one of the following choices is correct?

[MCQ: 2021-2M]

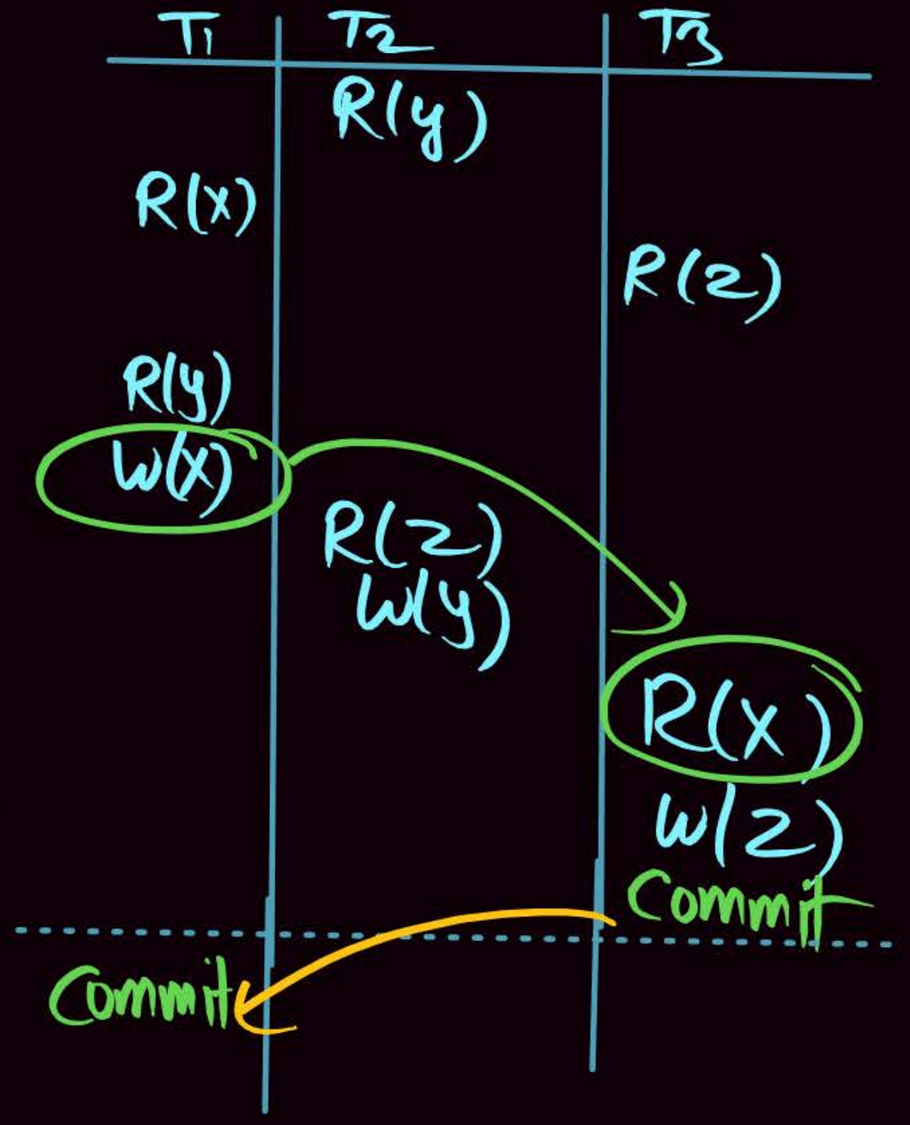
A Both P and Q are true.

B P is true and Q is false.

C P is false and Q is true.

D Both P and Q are false.

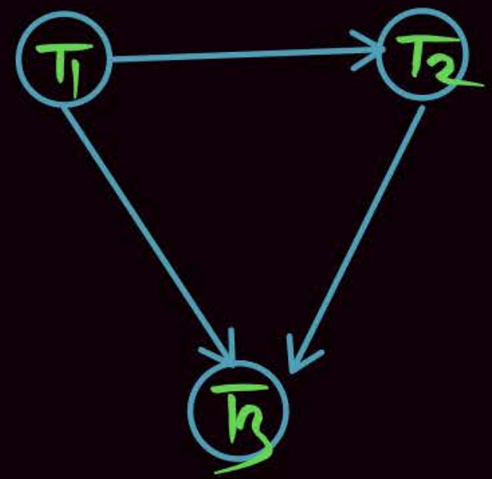
$R_2(y)\ R_1(x)\ R_3(z)\ R_1(y)\ W_1(x)\ R_2(z)\ W_2(y)\ R_3(x)\ W_3(z)$



Not Recoverable

$R_2(y)$ $R_1(x)$ $R_3(z)$ $R_1(y)$ $W_1(x)$ $R_2(z)$ $W_2(y)$ $R_3(x)$ $W_3(z)$

T_1	T_2	T_3
	$R(y)$	
$R(x)$		
		$R(z)$
$R(y)$		
$W(x)$		
	$R(z)$	
	$W(y)$	
		$R(x)$
		$W(z)$



$< T_1\ T_2\ T_3$

Conflict Serializable

Any Doubt ?

THANK YOU!