COMPUTER SCIENCE

Database Management System

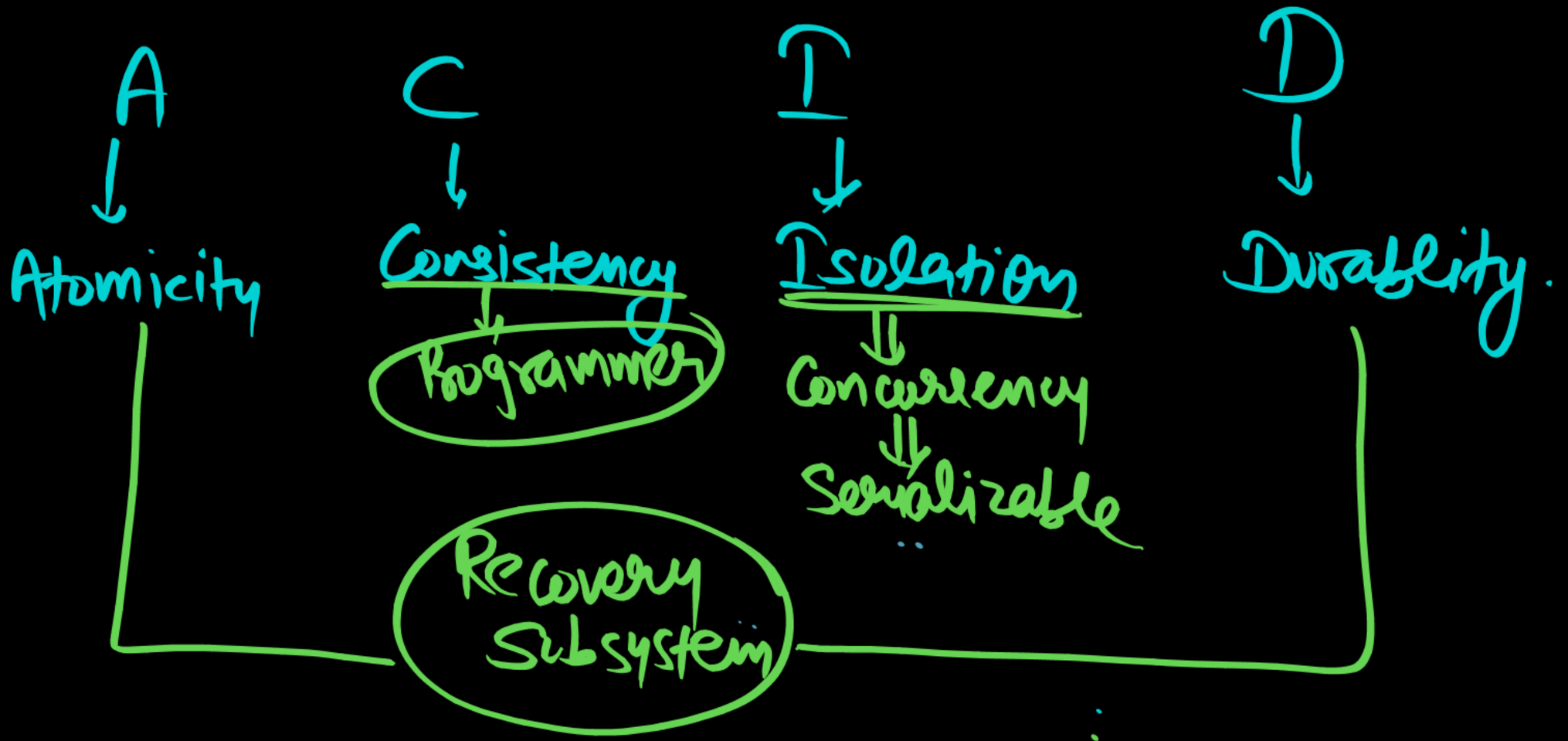
Transaction & Concurrency Control

Lecture_10

Vijay Agarwal sir

01 Lock Based Protocol

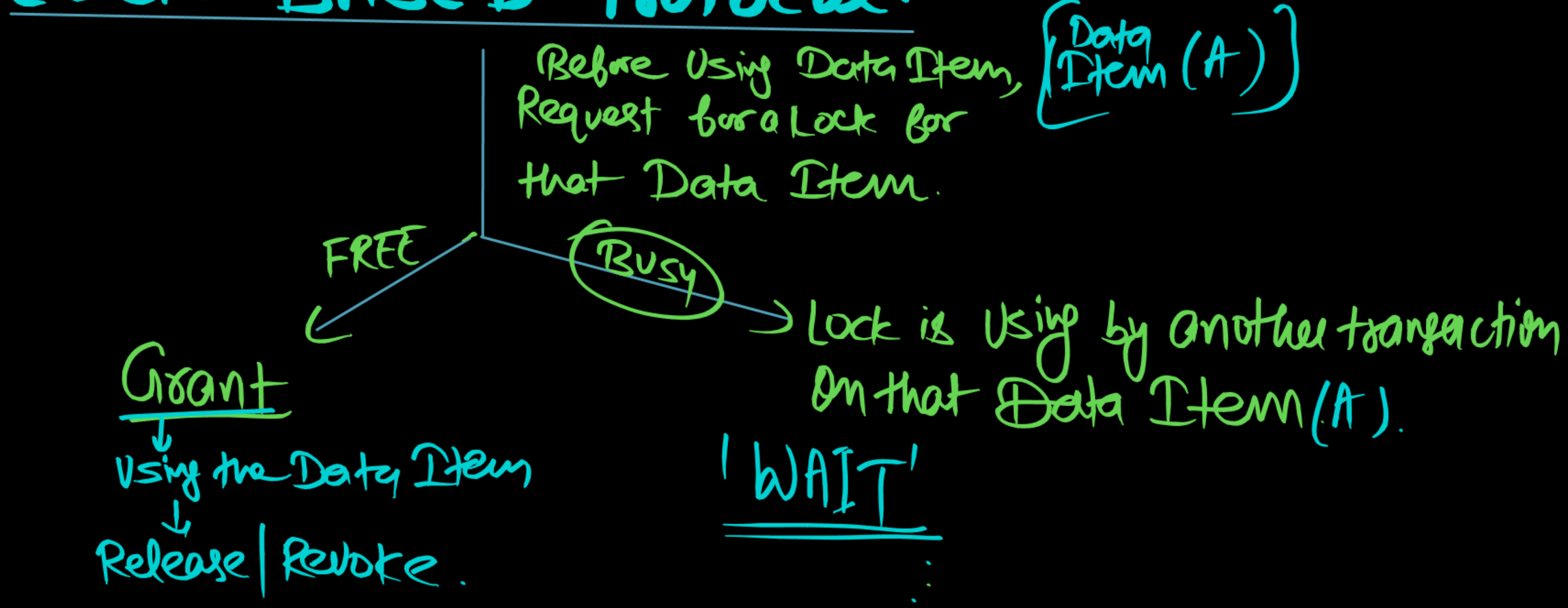
Transaction

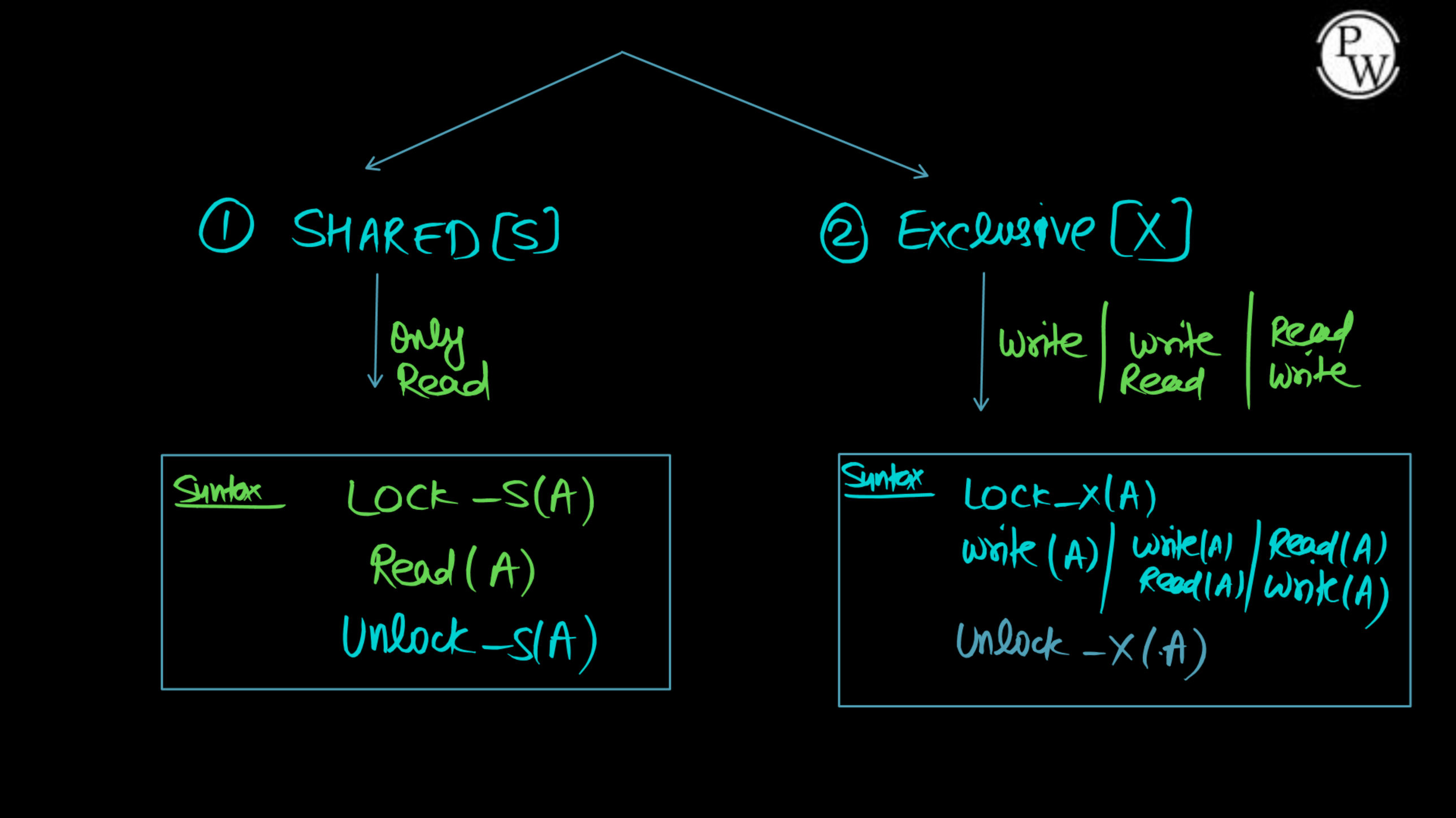



Serializability
1. Conflict Serializable
2. View Serializable
Recoverability
Recoverable schedule
Cascadeless
Strict schedule.

Implementation of Concurrency Control Scheme

LOCK BASED Protocal:




① SHARED [S]
Only Read
② Exclusive [X]
Write | Write Read | Read Write
Syntax
Lock-S(A)
Read(A)
Unlock-S(A)
Syntax
Lock-X(A)
write(A) | write(A) Read(A) | Read(A) Write(A)
Unlock-X(A)

R : Read
W : Write

S : → Only Read

X → Write

Tj

DATA Item (A)	S	X
S	YES	NO
X	NO	NO

Ti

Ti	Tj
S(A)	S(A) ✓
S(A)	X(A) X
X(A)	S(A) X
X(A)	X(A) X

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.

Lock-Based Protocols (Cont.)

- Lock-compatibility matrix

Tj

Ti

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.

T_1	T_2
Read(A)	
A=A-100	
Write(A)	
	Read(A)
	Read(C)
Read(B)	
B=B+100	
Write(B)	

T_1	T_2	Lock manager
Lock_X(A)		GRANT_X(A, T_1)
Read(A)		
A=A-100		
Write(A)		Release/
Unlock_X(A)		Revoke_X(A, T_1)
	Lock_S(A)	GRANT_S(A, T_2)
	Read(A)	
	Unlock_S(A)	Revoke_S(A, T_2)
	Lock_S(C)	GRANT_S(C, T_2)
	Read(C)	
	Unlock_S(C)	Revoke_S(C, T_2)
Lock_X(B)		GRANT_X(B, T_1)
R(B)		
B=B+100		
Write(B)		Revoke_X(B, T_1)
Unlock_X(B)		

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)	revoke_S(A, T_2)
	lock-S(B)	
		grant-S(B, T_2)
	read(B)	
	unlock(B)	
	display(A+B)	revoke_S(B, T_2)
lock-X(A)		
		grant-X(A, T_1)
read(A)		
A:=A + 50		revoke_X(A, T_1)
write(A)		
unlock(A)		

Two Phase locking Protocol (2PL)

Lock & Unlock Request done in the 2 Phase.

Phase I (1) Growing Phase [Acquire Lock]

Phase II (2) Shrinking Phase [Release Lock]

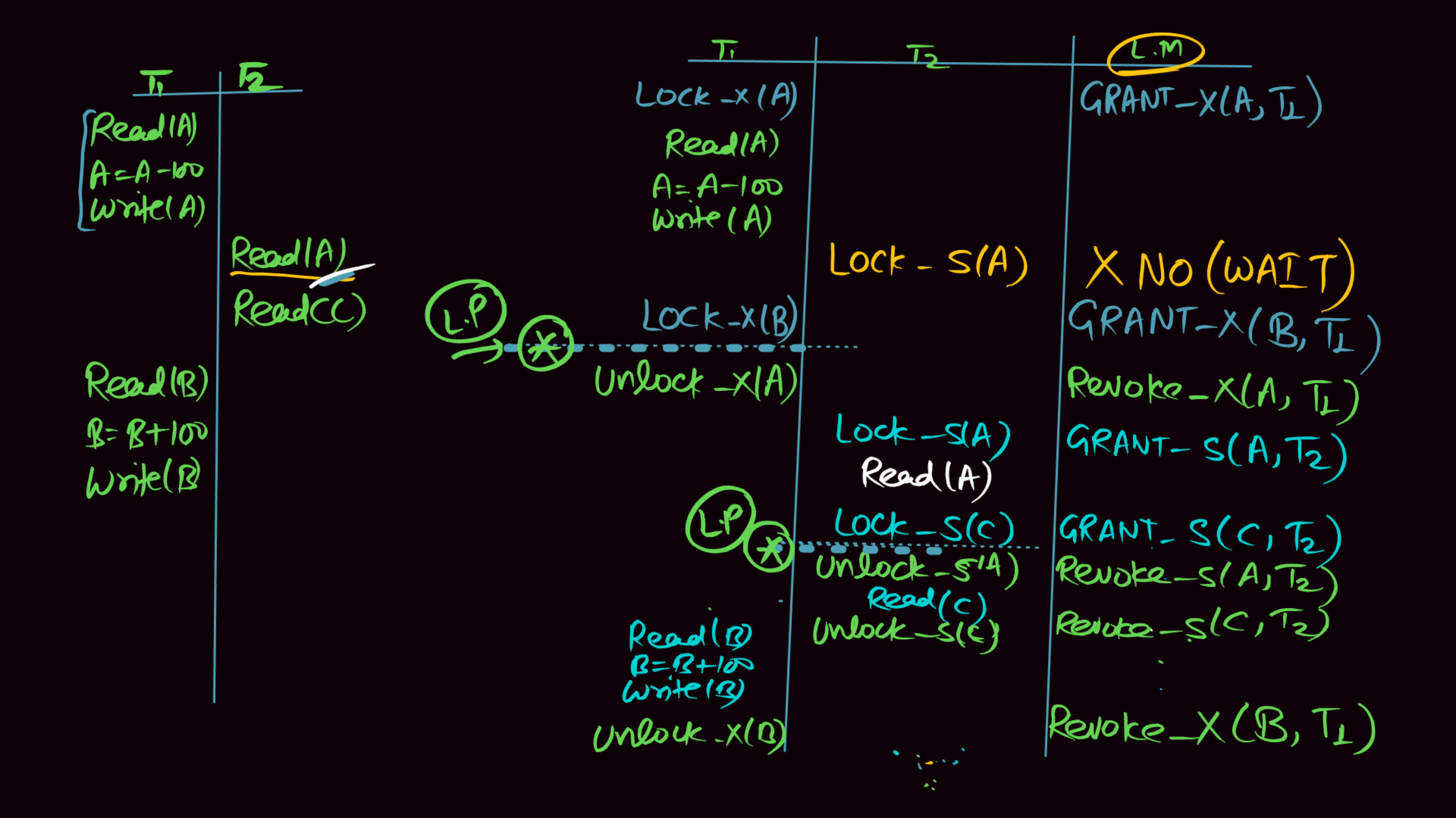
Note

Each Transaction first Finish its Growing Phase then Shrinking Phase.

Two Phase locking Protocol (2PL)

① Growing Phase : In the Growing Phase, Transaction May obtain the Locks. But Must Not Release Any Lock.

② Shrinking Phase : In the Shrinking Transaction May Release the Lock But Must Not ask for Any New Lock Request. ∴


T1
T2
Read(A)
A=A-100
Write(A)
Read(A)
Read(C)
Read(B)
B=B+100
Write(B)
T1
T2
L.M
Lock_X(A)
Read(A)
A=A-100
Write(A)
GRANT_X(A, T1)
Lock_S(A)
X NO (WAIT)
L.P
Lock_X(B)
GRANT_X(B, T1)
Unlock_X(A)
Revoke_X(A, T1)
Lock_S(A)
GRANT_S(A, T2)
Read(A)
L.P
Lock_S(C)
GRANT_S(C, T2)
Unlock_S(A)
Revoke_S(A, T2)
Read(C)
Unlock_S(C)
Revoke_S(C, T2)
Read(B)
B=B+100
Write(B)
Unlock_X(B)
Revoke_X(B, T1)

Important Point about 2PL.

① 2PL Ensure Conflict Serializable Schedule.
   → Serializablity.

② If a Schedule is followed by 2PL then Ensure Conflict Serializable Schedule.

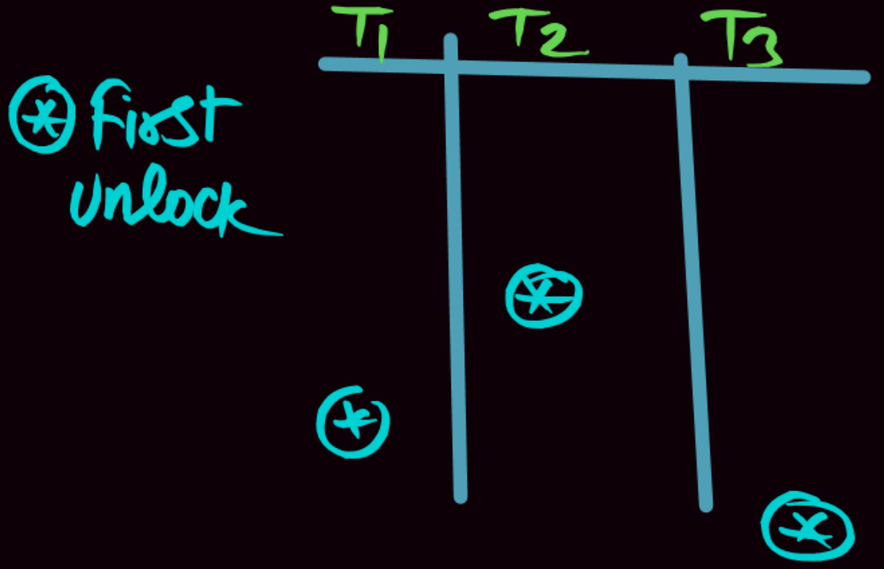
③ Equivalent Serial Schedule [Serializablity order] is determined by LOCK POINT.

Important Point about 2PL.

④ **LOCK POINT** : Position of LAST Lock (or) First Unlock of the Transaction.

(OR)

from where Shrinking Phase of the transaction Start.



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

∴

Important Point about 2PL.

(5) 2PL Not Ensure Recoverablity. (Irrecoverable Schedule followed by 2PL)

Irrecoverable Schedule but followed by 2PL.

Important Point about 2PL.

(1)

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

T_1	T_2
S(A)	
R(A)	
	X(B)
	W(B)
S(B) → Denied by T_2	
	X(A) ↩ Denied by T_1

Deadlock

(2)

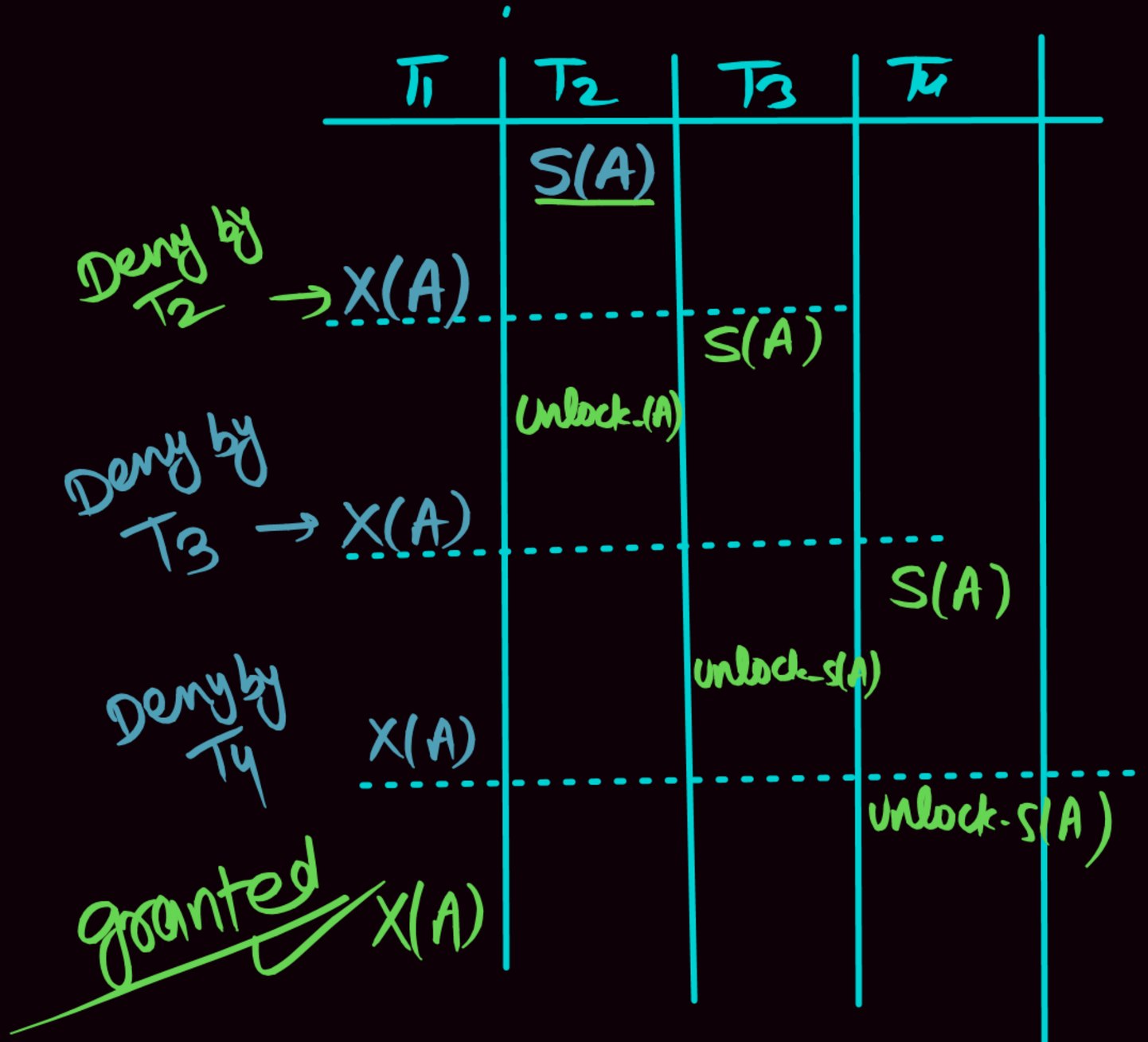
T_1	T_2
W(A)	
	R(B)
W(B)	
	R(A)

T_1	T_2
X(A)	
W(A)	
	S(B)
	R(B)
X(B) → Denied by T_2	
S(A) ↩ Denied by T_1	

Deadlock

⑥ 2PL Subber from Deadlock (Not Free From Deadlock)

⑦ 2PL Subber from Starvation.

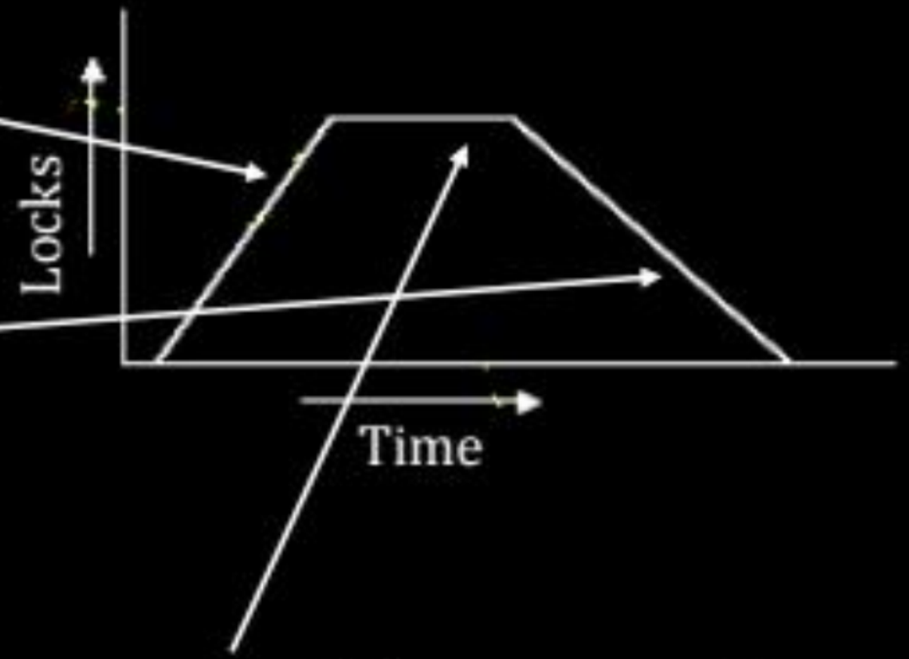


2PL Suffer from Starvation.

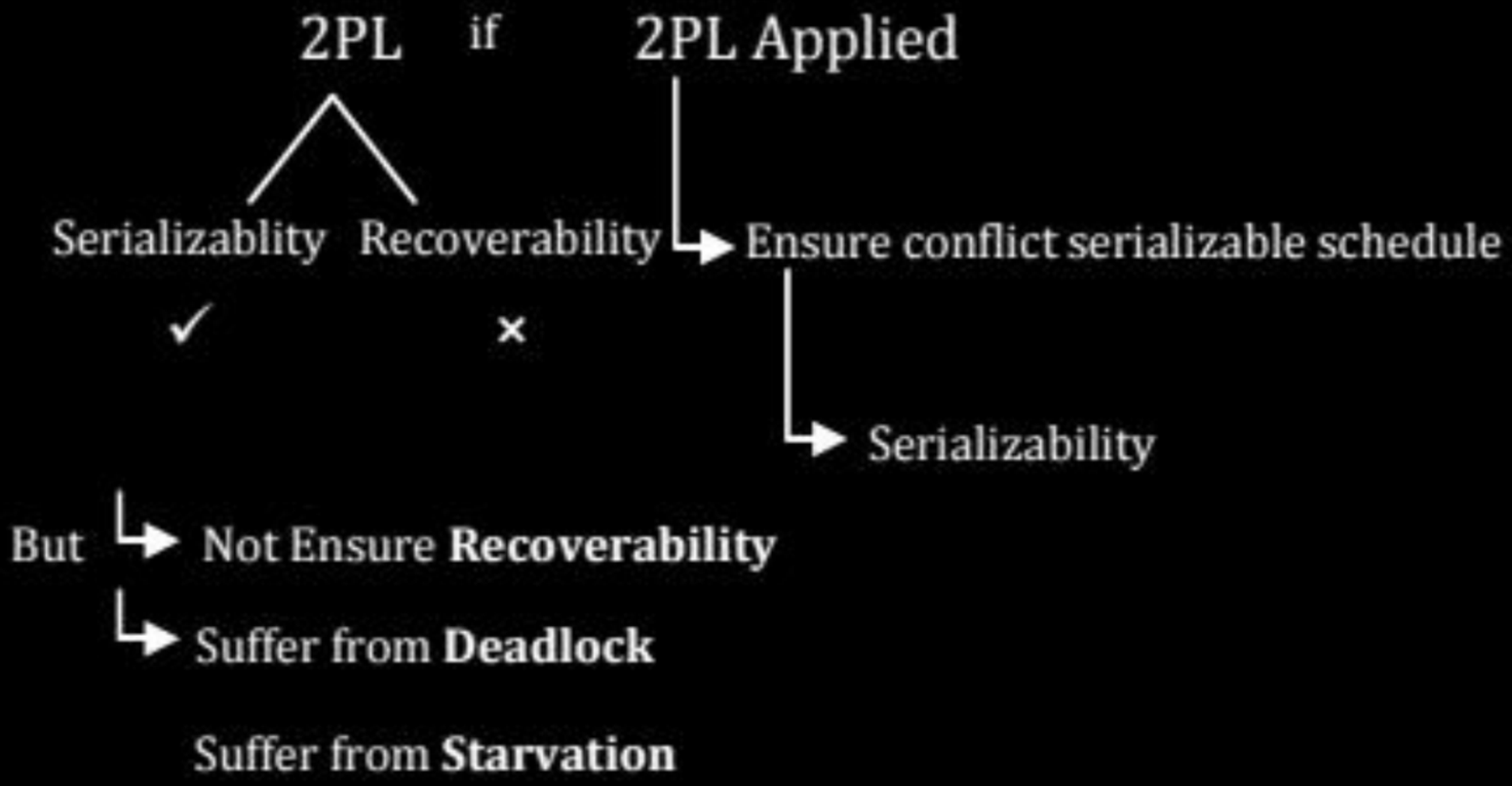
∴

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).


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 Lock taken by transaction must be Release After Commit (or) Rollback.

(OR)

2PL + All Exclusive Lock Taken by the Transaction Must be Held untill Commit/Rollback of the Transaction.

T_1	T_2
X(A)	
W(A)	
C/R	
Unlock-X(A)	
	S(A)/X(A)
	R(A)/W(A)

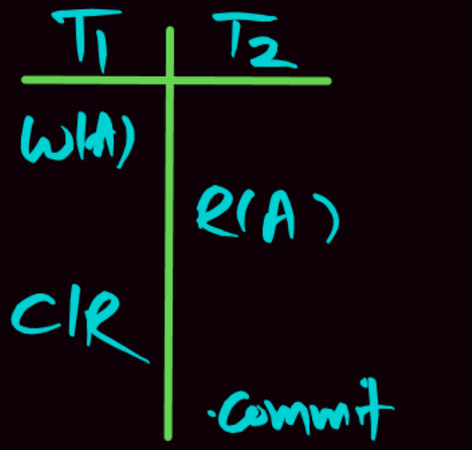
Strict Recoverable.

T_i

- X(A)
- S(B)
- X(C)
- S(D)

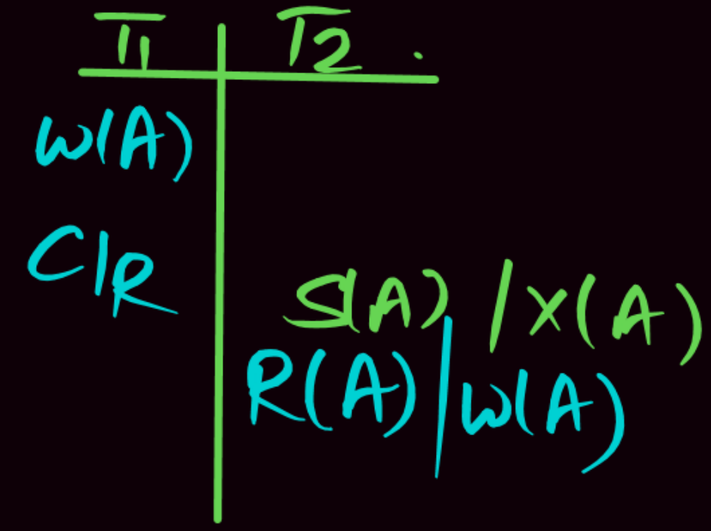
≡

Unlock-(B)
Unlock-(D)
C/R
- Unlock-X(A)
- Unlock-X(C)



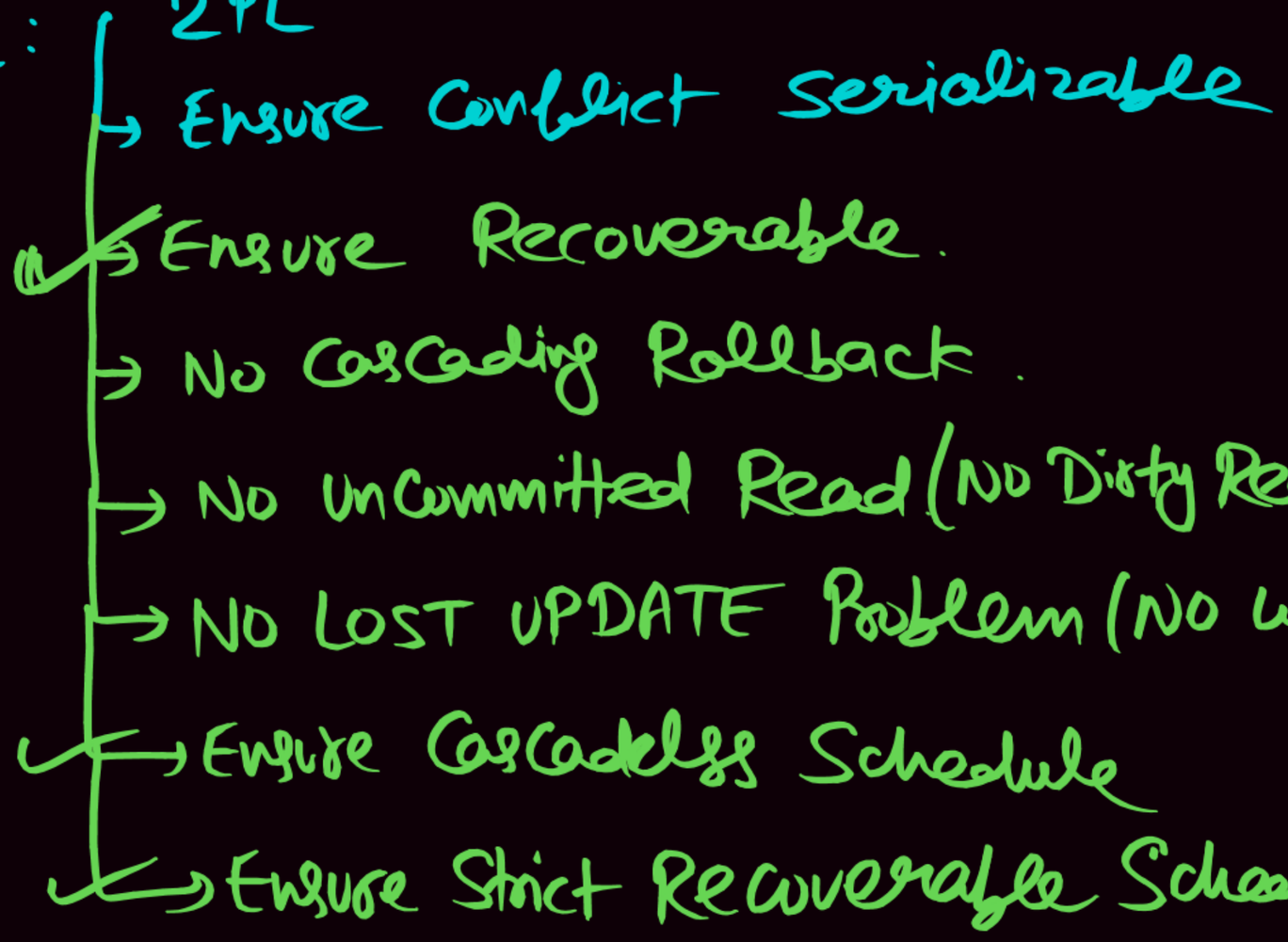
Recoverable

Cascadeless



Strict Recoverable

STRICT 2PL :



Suffers from

- Deadlock
- Starvation.

Rigorious 2PL :

2PL + ALL Locks [SHARED (S) & Exclusive (X)] taken by the transaction Must be Held by the transaction Untill Commit/Rollback.

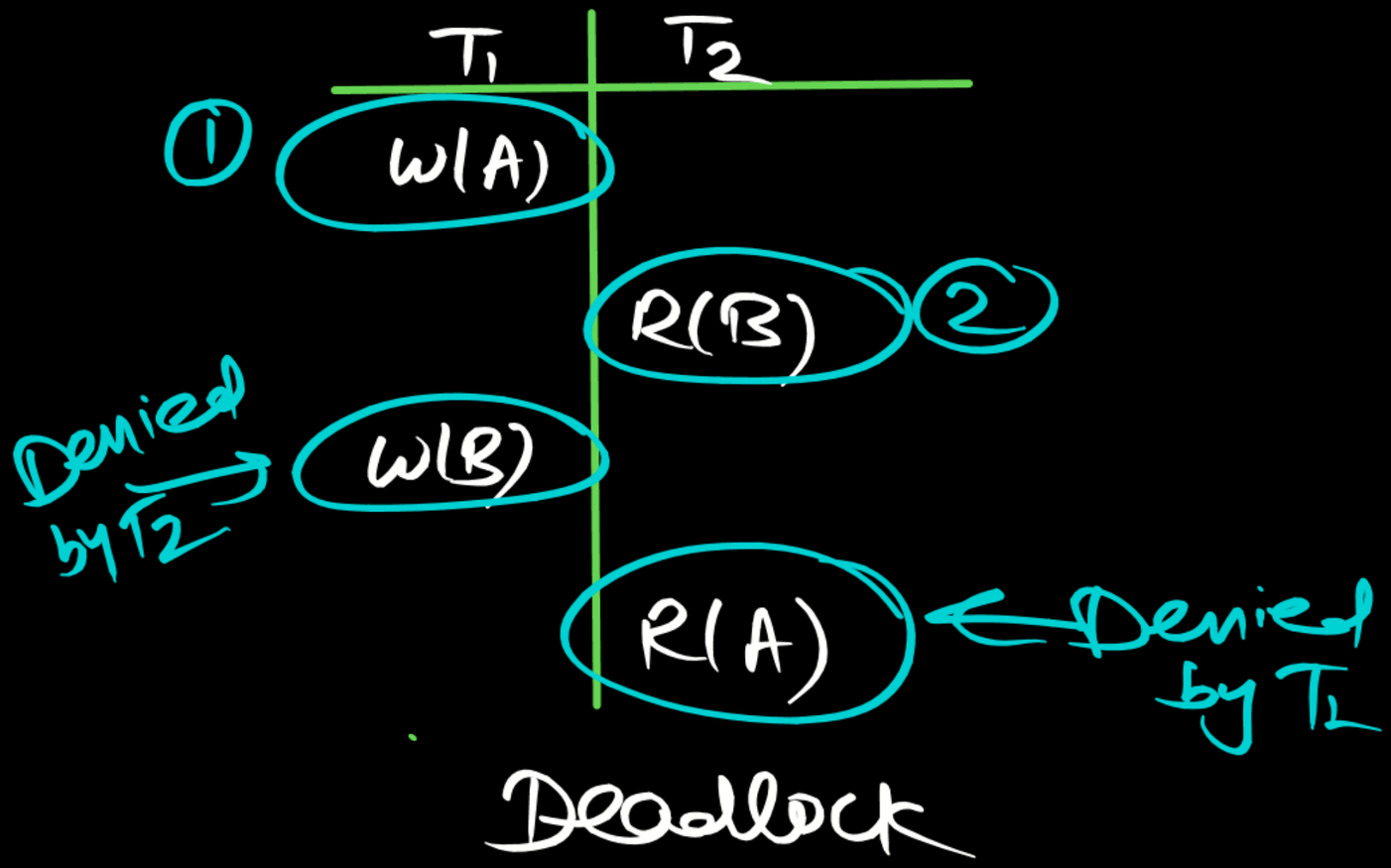
OR

Release After Commit / Rollback.

Ti
X(A)
S(B)
X(C)
S(D)
≡
C/R
U(A)
U(B)
U(C)
U(D)

Rigorious 2PL Suffers from:

1. Deadlock
2. Starvation.


T1
T2
1
W(A)
R(B)
2
Denied by T2
W(B)
R(A)
Denied by T1
Deadlock

Conservative 2PL : Each Transaction acquire the lock in the beggining of the Transaction (Before the Start) & Release all lock After Commit/Rollback.

Conservative 2PL
- → Ensure Conflict Serializable
- → Recoverable, Cascadeless, Strict Recoverable
- → No Cascading Rollback/Abort
- → No Uncommit Read / No Dirty Read
- → No Lost UPDATE Problem
- → NO DEADLOCK.

Suffer from 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
  - (2) Strict two-phase locking: a transaction must hold all its exclusive locks till it commits/aborts.
    - Ensures recoverability and avoids casecading roll-backs
  - (3) Rigorous two-phase locking: a transaction must hold all locks till commit/abort.
    - Transactions can be serialized in the order in which they commit.

(4) Conservative 2PL

- Most databases implement rigorous two-phase locking, but refer to if as simply two-phase locking

Conservative

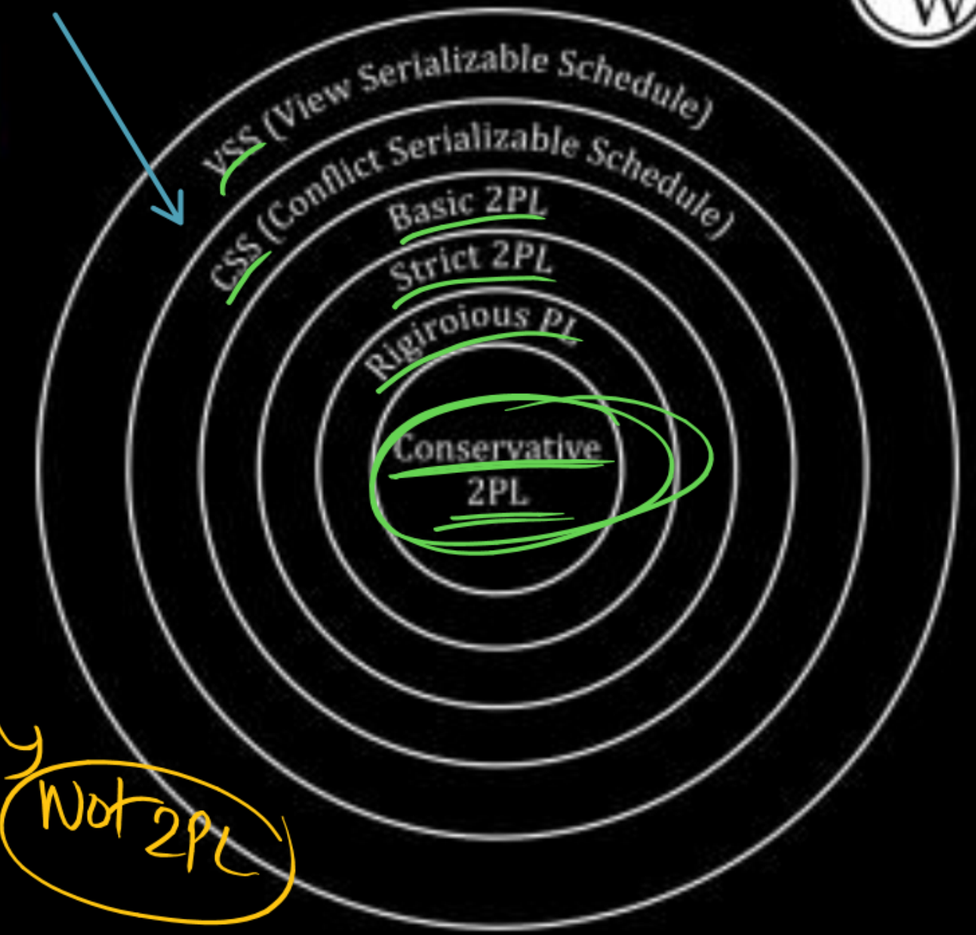
1	2	3	4	5
Lock-S(A)	Lock-S(A)	Lock-S(A)	Lock-S(A)	Lock-S(A)
R(A)	R(A)	R(A)	Lock-X(B)	R(A)
Lock-X(B)	Lock-X(B)	Lock-X(B)	R(A)	Unlock(A)
R(B)	Unlock(A)	R(B)	R(B)	Lock-X(B)
Unlock(A)	R(B)	W(B)	W(B)	R(B)
W(B)	W(B)	Commit	Commit	W(B)
Unlock(B)	Commit	Unlock(A)	Unlock(A)	Unlock(B)
Commit	Unlock(B)	Unlock(B)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

BASIC 2PL.

Strict 2PL (Basic 2PL)

Rigorious 2PL Strict 2PL Basic 2PL)

Conservative 2PL (Rigorious 2PL Strict 2PL Basic 2PL



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.)

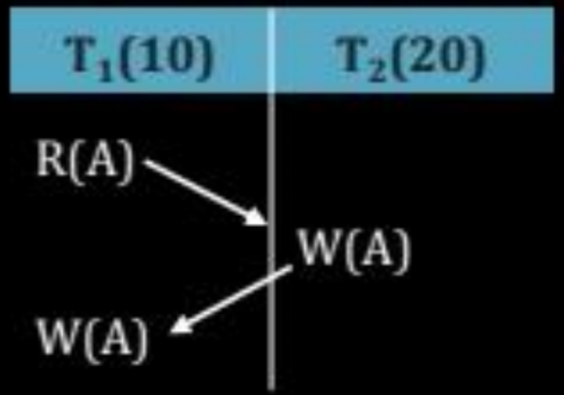
- Suppose a transaction T_i issues a **read** (Q)
  1. If $TS(T_i) \leq$ **W**-timestamp (Q), then T_i needs to read a value of Q that was already overwritten.
     - Hence, the **read** operation is rejected, and T_i is rolled back.
  2. If $TS(T_i) \geq$ **W**-timestamp (Q), then the **read** operation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

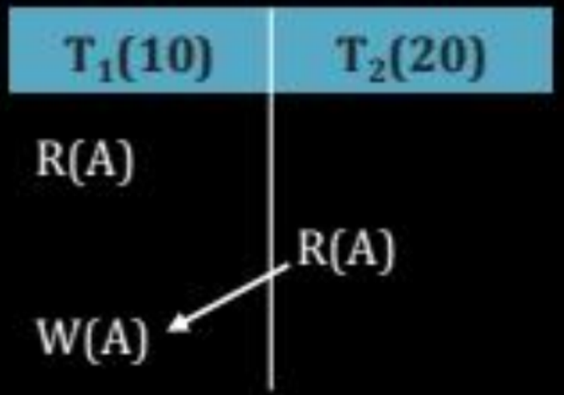
(1)



$TS(T_1) < TS(T_2)$

$T_1 \rightarrow T_2$

(2)



$TS(T_1) < TS(T_2)$

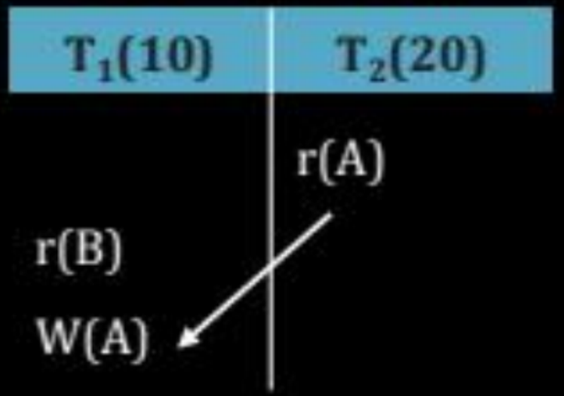
$T_1 \rightarrow T_2$

(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

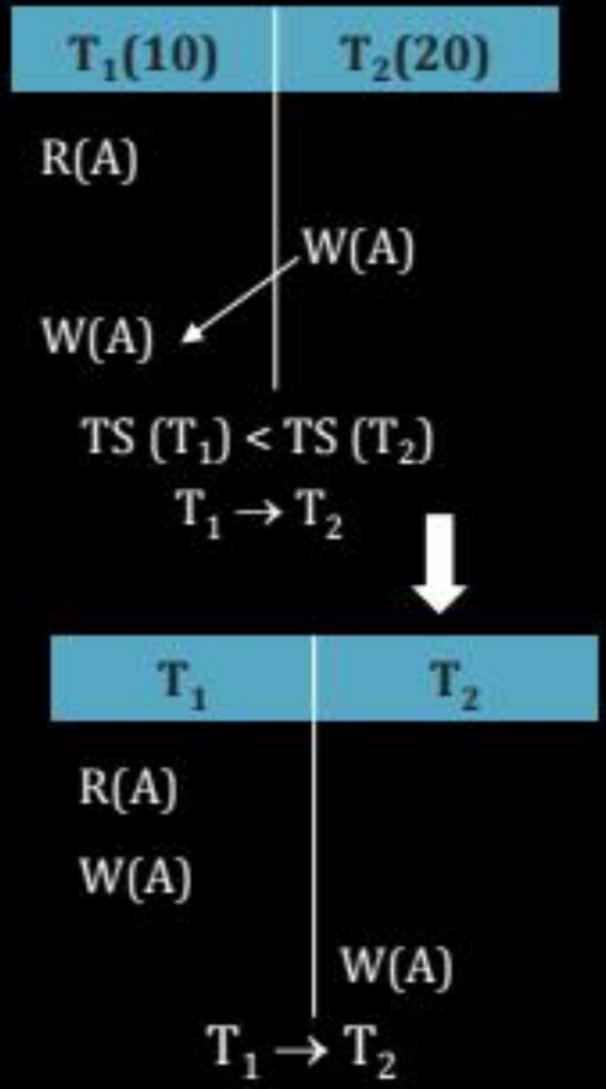
Same as TSP

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

Q.

Consider the following statements:

S_1: All strict recoverable schedule are serial.

S_2: All recoverable schedules are conflict serializable.

S_3: All strict schedules are conflict serializable.

S_4: All conflict serializable schedules are free from cascading rollbacks.

Which of the following is true?

(a) Only S_1 and S_4

(b) Only S_2, S_3 and S_4

(c) Only S_2 and S_4

(d) None of these

Q.

Consider the following transaction:

T_1: $R_1(x)$ $W_1(x)$ $R_1(y)$ $W_1(y)$

T_2: $W_2(y)$ $W_2(x)$

The number of non-serial schedules between T_1 and T_2 which are serializable?

(a) 2

(b) 13

(c) 15

(d) None of these

Any Doubt ?

THANK YOU!