Jai Shree Ram

INDEX

NAME: Anurag Singhal SUB: DBMS STD: ___ SEC: ___ ROLL NO: ___

(82) Conflict Serializability, Precedence Graph :-

Q:-

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

— Timeline.

→ First, we have to make precedence graph → (mean, just graph with edges & vertices).

Vertex → No. of Transac's (T_1, T_2 & T_3).

Now

* Edges:

Check the conflict pairs in other transactions and draw Edges.

Like * T_1 → (check in T_2 & T_3)

* Conflict pairs :-

R-W
W-R
W-W

of same variable. Let, (x).

* Start :-

Now, Start with first oper^n. i.e,

In T_1 → R(x)

& check the conflict pair of R(x) in other Transac's i.e. (T_2 & T_3).

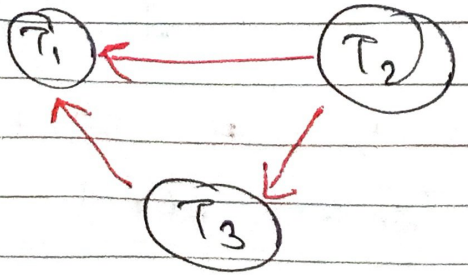
→ Conflict pair (C.P.) of R(x) → W(x)
&
now, Similarly,
conflict pair of W(x) → R(x), W(x).

→ Now, do check for every opera's in all transac's & after checking wrt those opera^. & If you find any conflict pair, then draw edge acc. to vertex (Transac^).

→

T_1	T_2	T_3	C.P. we find →
~~R(x)~~			
		~~R(y)~~	R(x) - W(x) ($T_3 \to T_1$)
		~~R(x)~~	R(y) - W(y) ($T_2 \to T_3$)
	~~R(y)~~		R(z) - W(z) ($T_2 \to T_1$).
	~~R(z)~~		W(z) - R(z) ($T_2 \to T_1$)
		~~W(y)~~	W(z) - W(z) ($T_2 \to T_1$)
	~~W(z)~~		
~~R(z)~~			
~~W(x)~~			
~~W(z)~~			

Precedence Graph:-



Date / / Page No.

* Now, Come on Graph :-
check that if there is any loop / cycle in the Graph.

How to check cycle :- अगर हम एक किसी भी Node से चलें, तो क्या हम घूम के वापस आ सकते हैं। If yes → loop / Cycle Exists.

(It is not must that the cycle must covers all the nodes. May be possible, it comes just after visiting one node). → also a cycle.

(* In our graph, there is no loop / cycle.).

* No loop / Cycle → Conflict Serializable Schedule
If loop / Cycle Exists → Not C.S.S.

* Conflict Serializable → Serializable → Consistent
(Serial Schedule)

* Now, how to make Serializable ! →

$T_1 \rightarrow T_2 \rightarrow T_3$
$T_1 \rightarrow T_3 \rightarrow T_2$
$T_2 \rightarrow T_1 \rightarrow T_3$
$T_2 \rightarrow T_3 \rightarrow T_1$
$T_3 \rightarrow T_1 \rightarrow T_2$
$T_3 \rightarrow T_2 \rightarrow T_1$

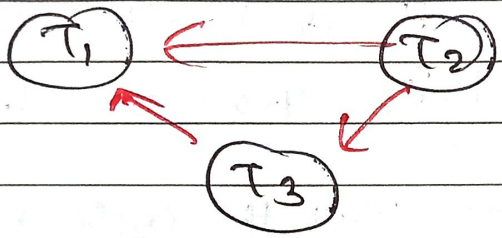
6 Cases → Now, which Case ?

Again, see the graph & check

Indegree = 0

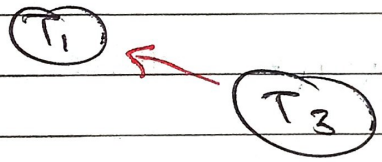
Indegree = 0, means a vertex with '0' indegree.

Mean, जहां i vertex पर कोई भी Edge (arrow) नहीं आ रहा हो।



T_2 has, Indegree = 0,

Now, remove T_2.



∴ $T_2 \rightarrow T_3 \rightarrow T_1$

T_1	T_2	T_3
	0	
		0
0		

---

83. View Serializability :→

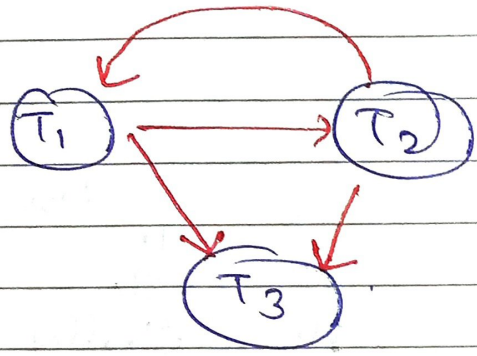
Q:- Check whether schedule is conflict Serializable or not?

→ first make Precedence graph.

Loop → No Answerable. then, view serializable Answers.

S

T_1	T_2	T_3
~~R(A)~~		
	~~W(A)~~	
~~W(A)~~		
		W(A)



∴ Here, we get a loop.

So,

It is Non Conflict Serializable.

↳ Here, we can't tell that it Serializable or not.

↳ Now,

(*) To check that, we use View Serializability.

→ Now, we arrange this Table,

T_1	T_2	T_3
R(A)		
	W(A)	
W(A)		
		W(A)

We change the position

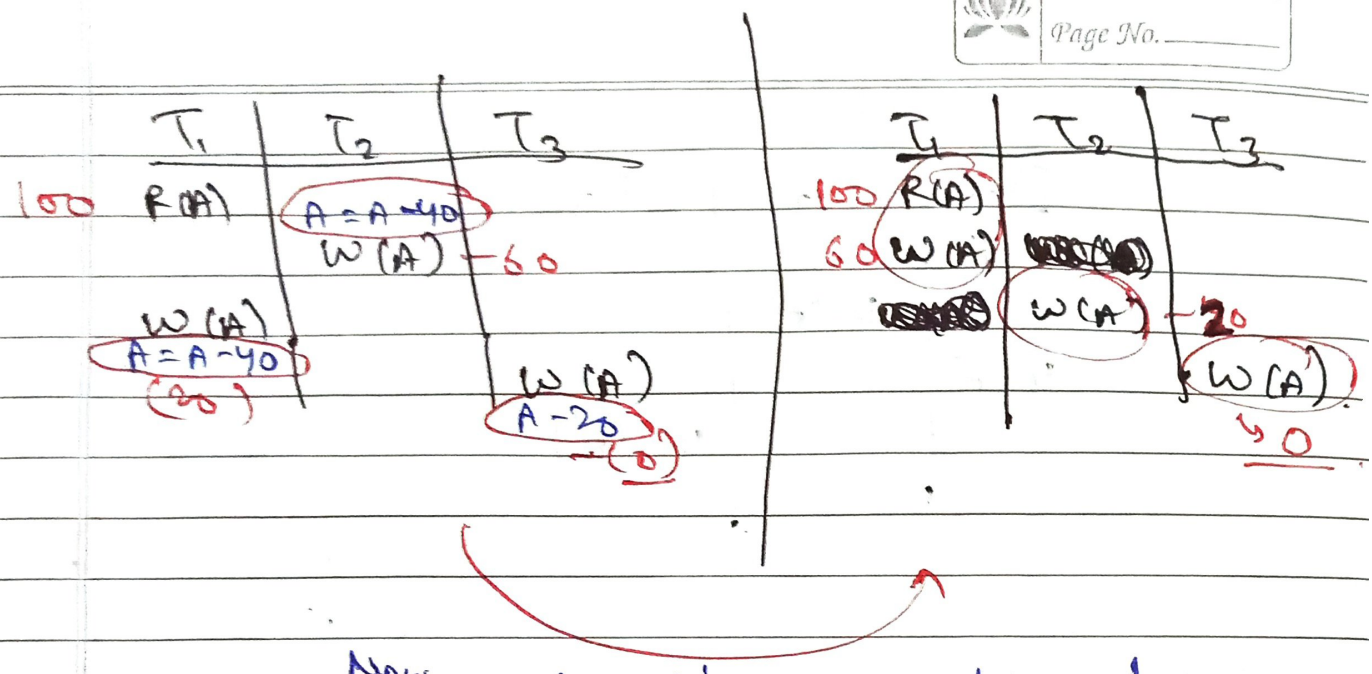
T_1	T_2	T_3
R(A)		
W(A)		
	W(A)	
		W(A).

Now this is a serial schedule. $T_1 \rightarrow T_2 \rightarrow T_3$.

But,

We have to check whether they match each other or not.

with A = 100.



Now, apply this same here.

Here, finally A value = 0

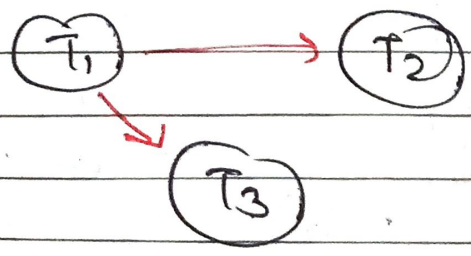
Here also finally A is '0'

Hence, Both are Equivalent.

They are View Equivalent (≡).

Note:- Bcz, finally output is given by A of T_3. So, if adjust the pos" of A. of T_1 & T_2. So, there is no problem.

Now,



(By : Reading of only T_1)

↳ Hence, $T_1 \longrightarrow T_2 \longrightarrow T_3$ (By Table).

Note:- Conflict Serializable tell that it is not Serializable (no ans.). But, View Serializable checks it & tell that it is Serializable.

(parallel schedule)

84. Concurrency Control Protocols :→ (C.C.P.)

(Shared - Exclusive Locking Protocol)

→ C.C.P. is that how to make them Serializable, or how to make them recoverable Schedule are concurrent, but how we can make them serializable & recoverable, this comes under Concurrency Control Protocol (C.C.P.).

→ We achieve this, By using Locking Protocols.

(#) 'Shared - Exclusive locking' :→

We use 2 locks here,

→ Shared lock (S) → if trans. locked data item in Shared mode, then allowed to Read only.

U → means unlock.

T_1	
S(A)	→ Shared lock.
R(A)	→ only Read
U(A)	→ Unlock.

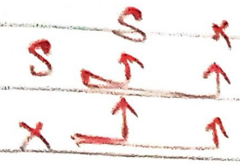
→ Exclusive lock (X) → if trans. locked data item in Exclusive mode then allowed to Read & write both.

T_2	
X(A)	→ Exclusive lock.
R(A)	
W(A)	
U(A).	→ unlock.

* Compatibility Table :- →

→ Request

grant ↓	S	X
S	Yes	No
X	No	No



Explo":-

S has only R(A)

X has both R(A), W(A)

⇒ S - S (No conflict) bcz R(A)-R(A).

⇒ S - X (Conflict).

R - R

R(A)	R(A)
	W(A)

⇒ X - S (Conflict)

R(A)	
W(A)	
	R(A)

⇒ X - X

R(A)	
W(A)	
	R(A)
	W(A)

→ conflict.

* Problems in S/X Locking :- →

1.) May not sufficient to produce only Serializable Schedule.

2.) May not free from Irrecoverability.

3.) May not free from deadlock.

4.) May not free from starvation.

(85.) Drawbacks in Shared | Exclusive :→
S/X locking protocol

⇒ locking gives us Serializable Schedule.
→ Consistent

1.) May not sufficient to produce only Serializable Schedule.
→ हो भी सकती, नहीं भी।

Ex.→

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

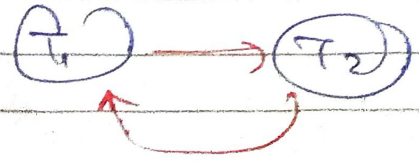
concurrent (parallel) →

try to Convert to → $T_1 \to T_2$ (or) $T_2 \to T_1$

T_1	T_2
X(A)	
R(A)	
W(A)	
U(A)	S(A)
	R(A)
X(B)	U(A)
R(B)	
W(B)	
U(B)	

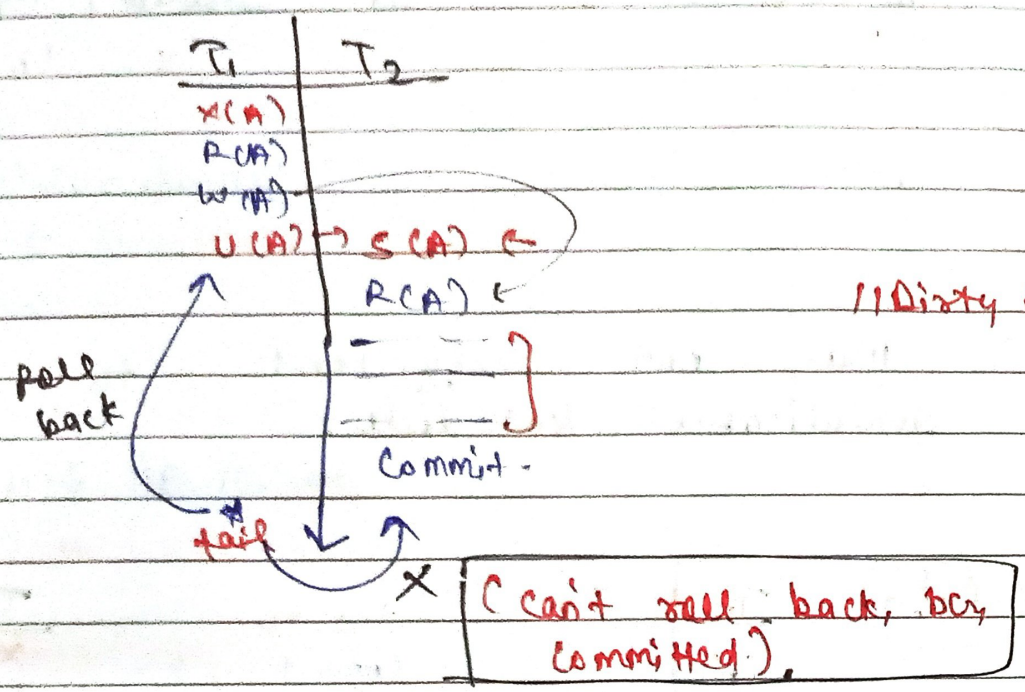
Note:→ Until, we do unlock, U(A) in T_1, we don't be able to put Shared lock S(A) on T_2. bcz by Compatibility Table. [(X-S) → No], so first we unlock X(A). & then, use S(A) on T_2 data.

2) Here, we don't get Serializable Schedule even after appling S/X locking. As u can see in Table.



2.) May not free from irrecoverability.

Ex:-



(can't roll back, bcz, committed).

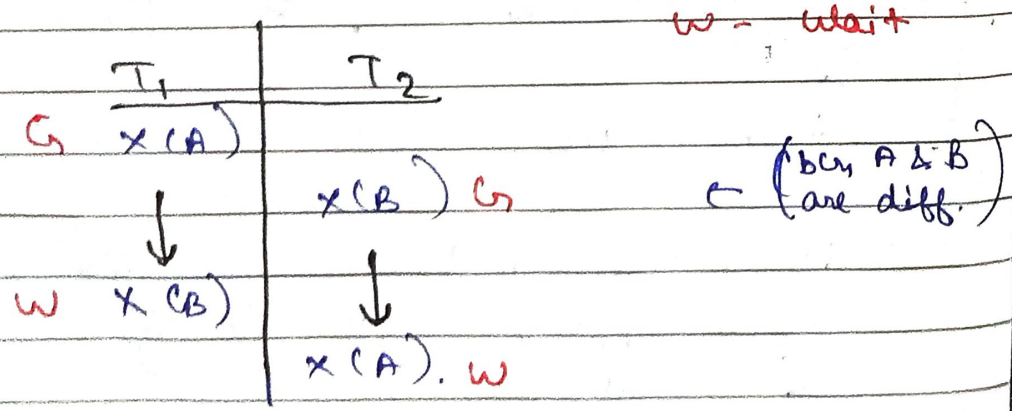
∴, It can't recover here.

3.) May not free from deadlock.

⇒ Deadlock :- When 2 person wait for resources, & they both are waiting in an infinite loop, so, when we wait infinitely, it is called Deadlock State.

G - Grant
W - Wait

Ex :-



2) Here, both are in waiting bcz, They are not unlock after use. So,

→ T_1 also waits that when T_2 unlocks, he can use on X(B).

Same

→ T_2 also waits that when T_1 unlocks, he can use on X(A).

So, Both are waiting in an Infinite loop.

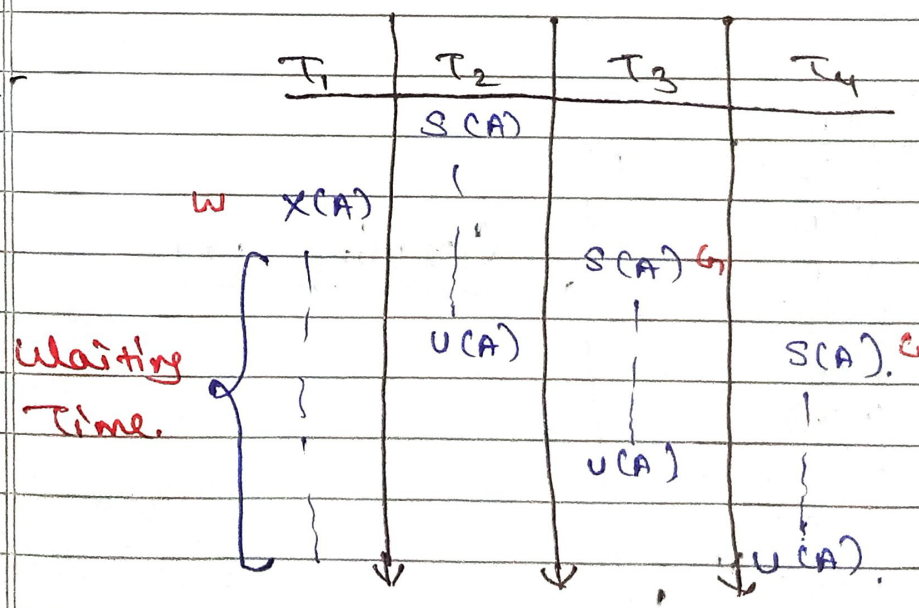
4.) May not free from starvation.

In deadlock, waiting upto Infinite time.
But,
In starvation, waiting not upto the Infinite time.

a) Shared के बाद Shared ले सकते हैं without unlock. by compatibility Table.

Ex:-

S → Shared Lock
X → Exclusive "
U → Unlock.



So, here T_1 waits for X(A) until T_4 unlocks.
So, here starvation also. (Bcz Shared के बाद Exclusive, कितना भी unlock के grant नहीं होता।)

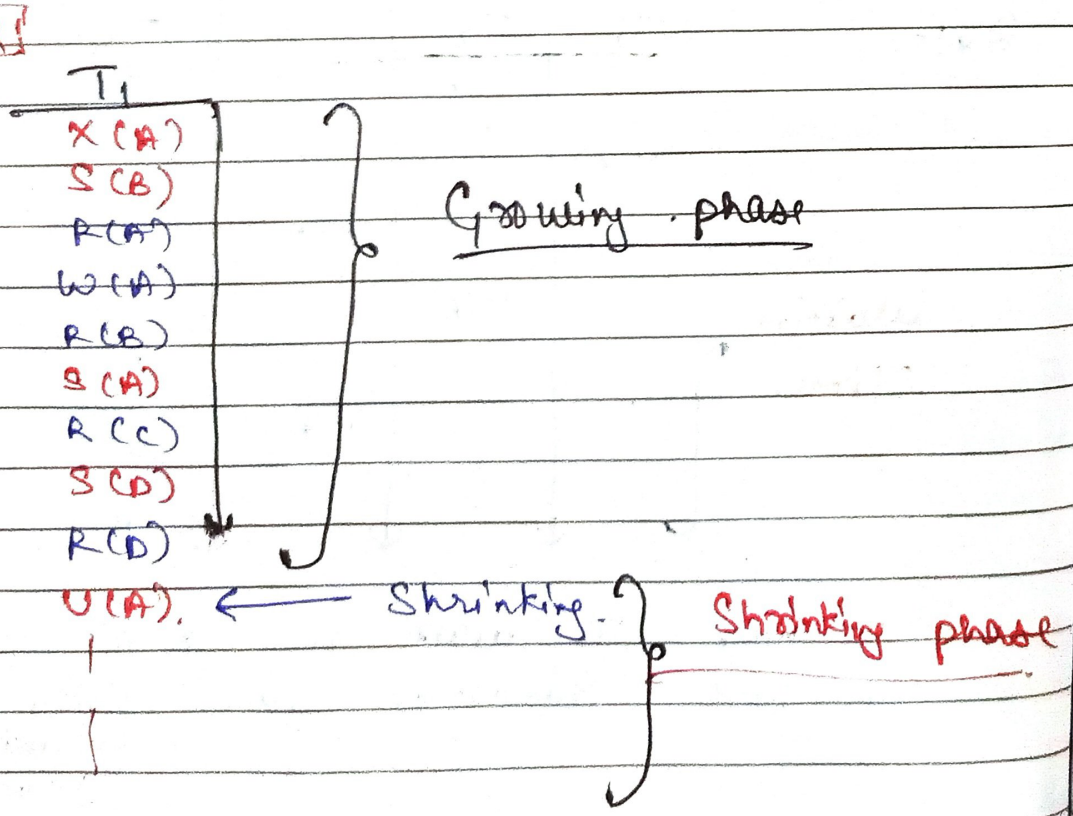
86. **Phase Locking (2PL) protocol in Transaction Concurrency Control :→**

⇒ 2PL (2 phase locking) is just the extension of simple Shared/Exclusive locking. We just do modifications in them.

(#) **2-Phase locking (2PL) :→**

→ Growing phase :→ locks are acquired & no locks are released.

→ Shrinking phase :→ locks are released & no locks are acquired.



Note! When Growing phase starts, then only apply locks.

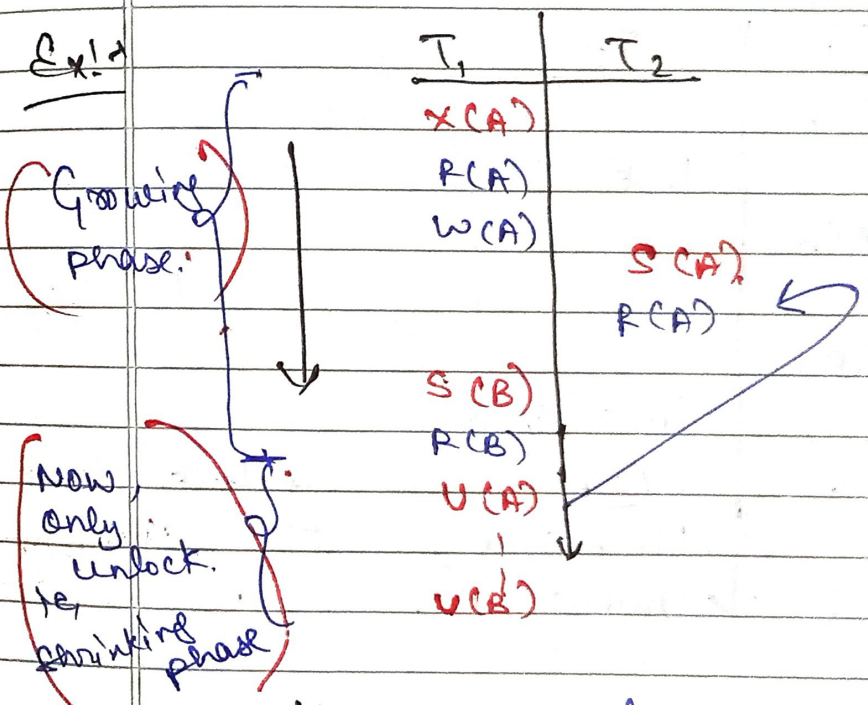
and,

When we first unlock, i.e, Shrinking phase starts, then we only unlock. (and not able to apply any lock).

⇒ We achieve Serializability by this, that we don't achieve in simple shared/exclusive protocol.

So,
we made (2-PL) protocol for this.

Ex!-



Hence, we first starts T_1, & if T_2 comes in b/w then we don't entertain him. First, we complete T_1 & then goes to T_2.

$T_1 \rightarrow T_2$,

Serializability Achieved.

↓

Consistency automatically achieved.

Note:- The transaction which follow (2-PL), then it is always serializable.

Ex:- यहाँ पर भी Shared के अलावा Shared तो दे ही सकते हैं। Means,

While T_1 is in growing phase by using S(A) [Shared lock], then at same time T_2 also starts its growing phase by using S(A). bcz by Compatibility Table.

Ex:-

S

T_1	T_2
Lock S(A)	
	Lock S(A)
Lock X(B)	
→ Unlock (A)	
Unlock (B)	
	Lock X (B)
	→ Lock point
	Unlock (A)
	Unlock (B)

Growing
Lock point
Shrinking
Growing
Lock point
Shrinking

Now,

* Serializability Schedule, How formed?

i.e., $T_1 \rightarrow T_2$ (OR)

$T_2 \rightarrow T_1$

So, This is done by (Lock Point).

Commit के बाद Roll back. नहीं कर सकते।

⇒ Lock Point :→ where trans. is taking the last lock. (or)
where trans is unlock first time.

⇒ जिसका Lock Point पहले आ जाएगा। वो trans. पहले आ जाएगी।

i.e., $T_1 \rightarrow T_2$

87. Drawbacks in (2-PL) protocol :→

→ Advantage: Always Ensures Serializability.

⇒ Drawbacks :→

(1.) May not free from Irrecoverability.

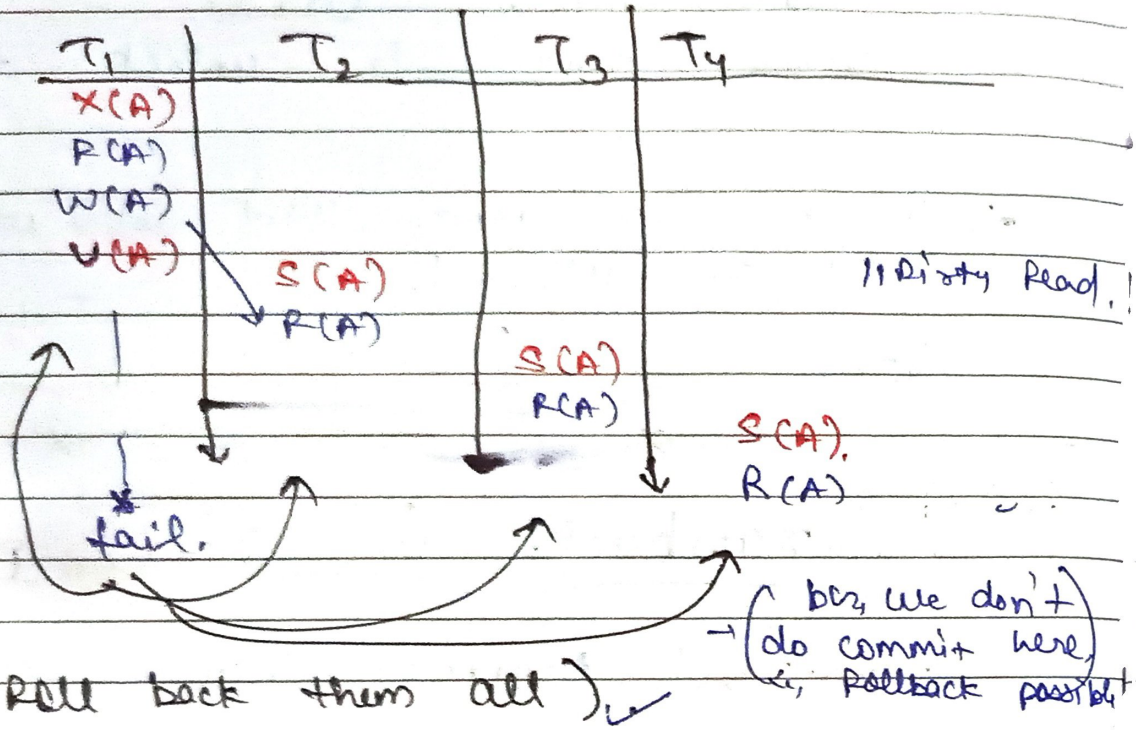
T_1	T_2
X(A)	
R(A)	
W(A)	
U(A)	
	R(A)
Roll back	Commit
fail	

→ (we can't roll back it, bcz it is already commited)

Hence,
It is Ex. of Irrecoverable Schedule.
We can't recover it (Roll back it).
(normal).

(2.) Not free from Cascading Rollback →

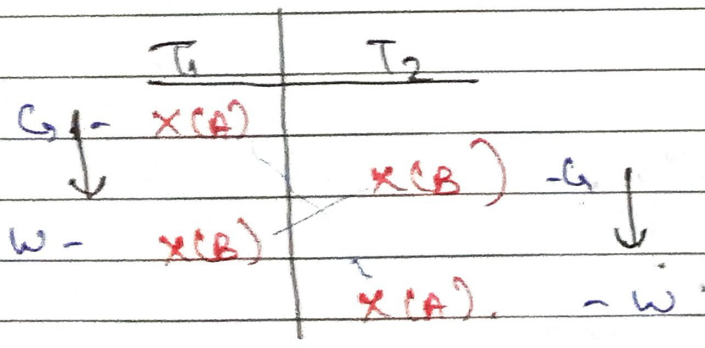
Ex:-



Bad performance & Cascading Rollback is possible here.

(3.) Not free from Deadlocks.

G - Grant
W - Wait



(किसी ने भी unlock नहीं किया अभी, ∴, दूसरी waiting में ही है)

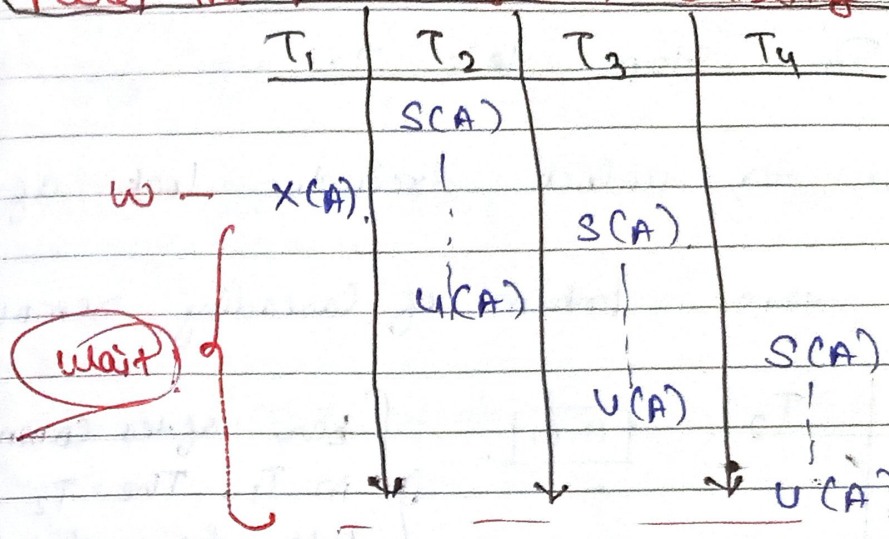
Infinite waiting here

(T1 & T2 both waiting here to complete their transac's).

(4.) Not free from Starvation.

(wait for limited time)

In 2PL
Here, the problem of Serializability removed.



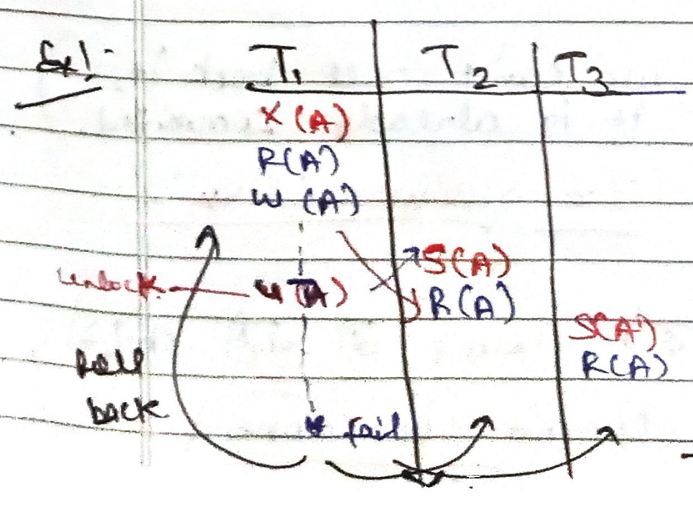
88. Strict 2PL, Rigorous 2PL & Conservative 2PL Schedule :→

→ These are the Extension (Advance) of 2PL :→ basic.

Strict 2PL :→
It should satisfy the basic 2PL and all exclusive locks should hold until commit/Abort.

Rigorous 2PL :→ It should satisfy the basic 2PL and all shared, exclusive locks should hold until commit/Abort.

Ex!-

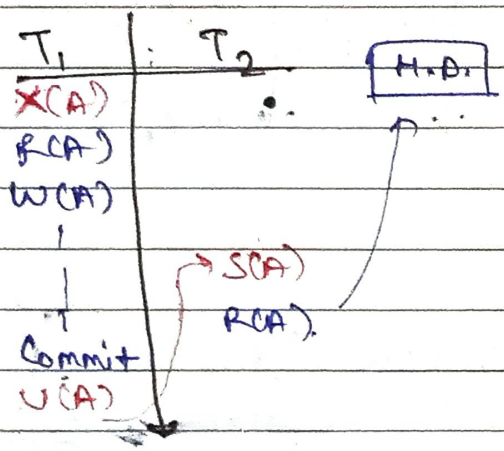


→ (also have to roll back T2 & T3 i.e. (Cascading roll backs)

So, To stop this →.

→ (We have to unlock Exclusive lock after Commit.
then, this problm of Cascading removed.

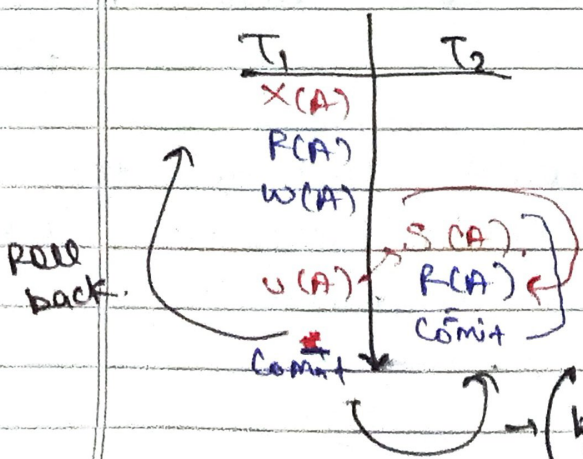
⇒



(bcz, after Commit in T_1. The T_2 takes data from database (H.D.) is not from W(A) of T_1 so, no dirty Read.

Hence, Here, Cascading Rollback is Removed,
∴ It will always produce Cascadeless.

⇒ Now, ∀ Irrecoverability :→



→ (Same data i.e, A)

→ (but we can't roll back it, bcz it is already committed.

∴, Irrecoverability.

→ Hence, हम इसको पहले unlock ही नहीं करेंगे।
We unlock it after commit is done.