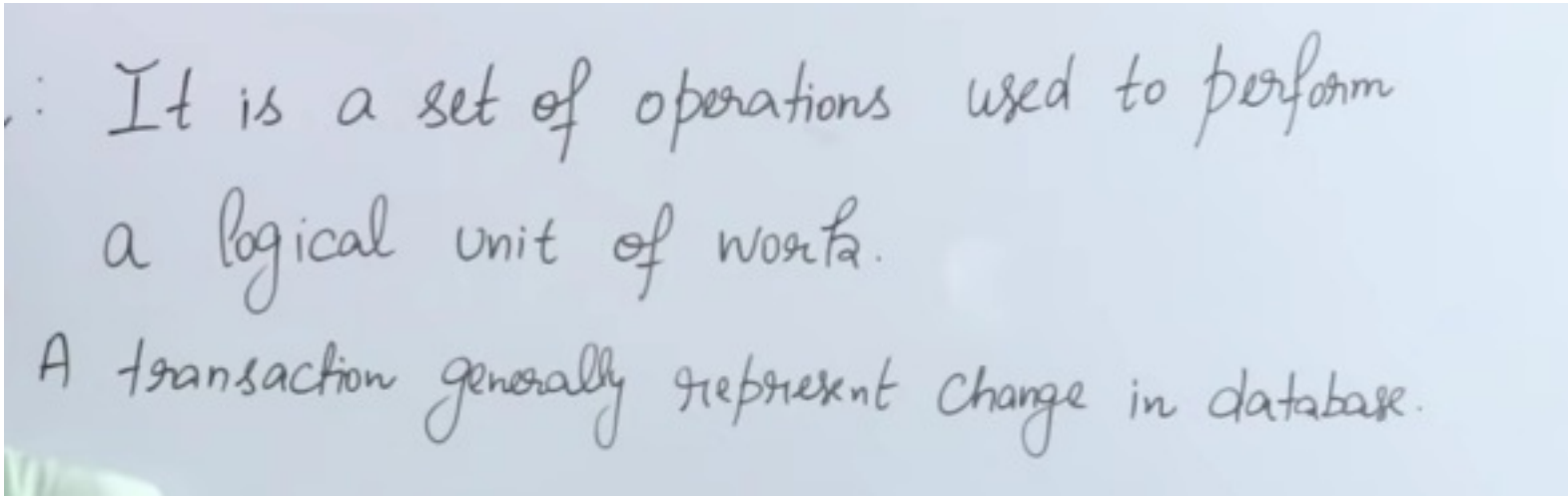
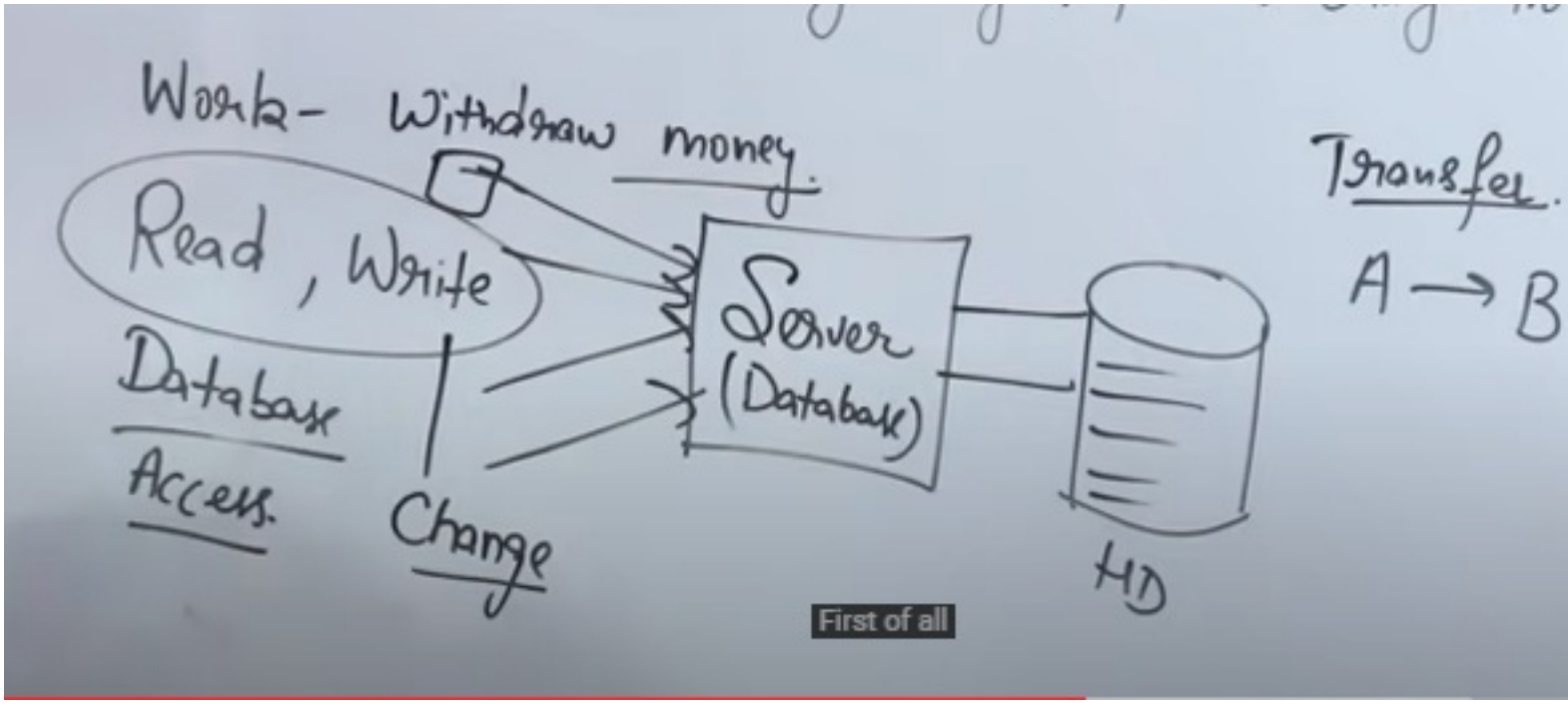
DBMS NOTES – II

TRANSACTION CONCURRENCY

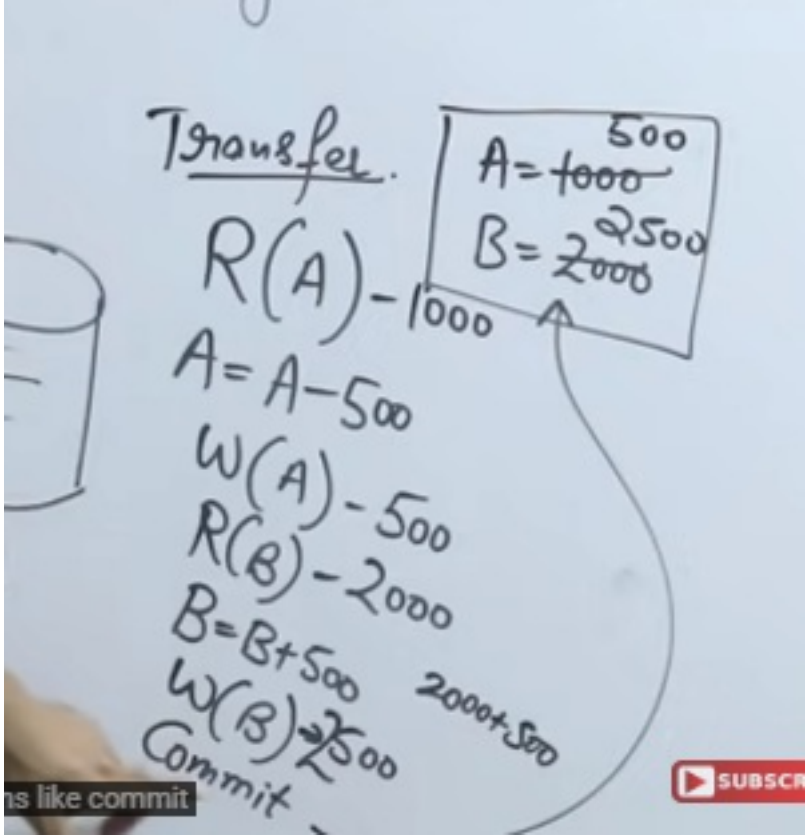




Lets say I want to transfer money from a to b

So first of all I would have to read the data of a

So following processes will occur in the RAM :



Reading hogi pehle a k account k baare mei

Then updation hoga if 500 has to be transferred

Commit is used to save all the changes in data in the database

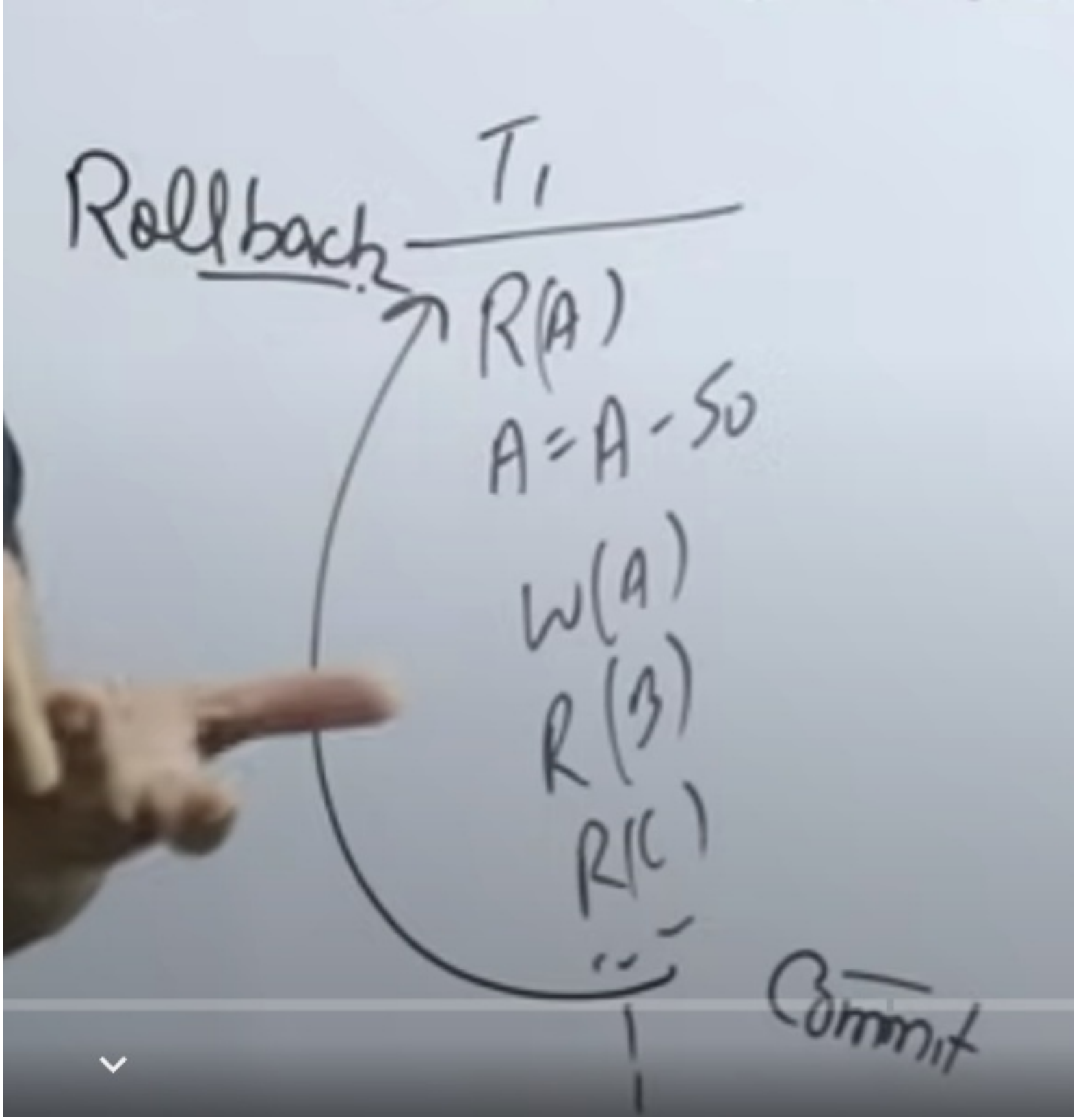
ACID properties

Atomicity Consistency Isolation Durability

- **Conceptual hai**
- **ATOMICITY**

Either all the operations will be executed Or none

Eg :

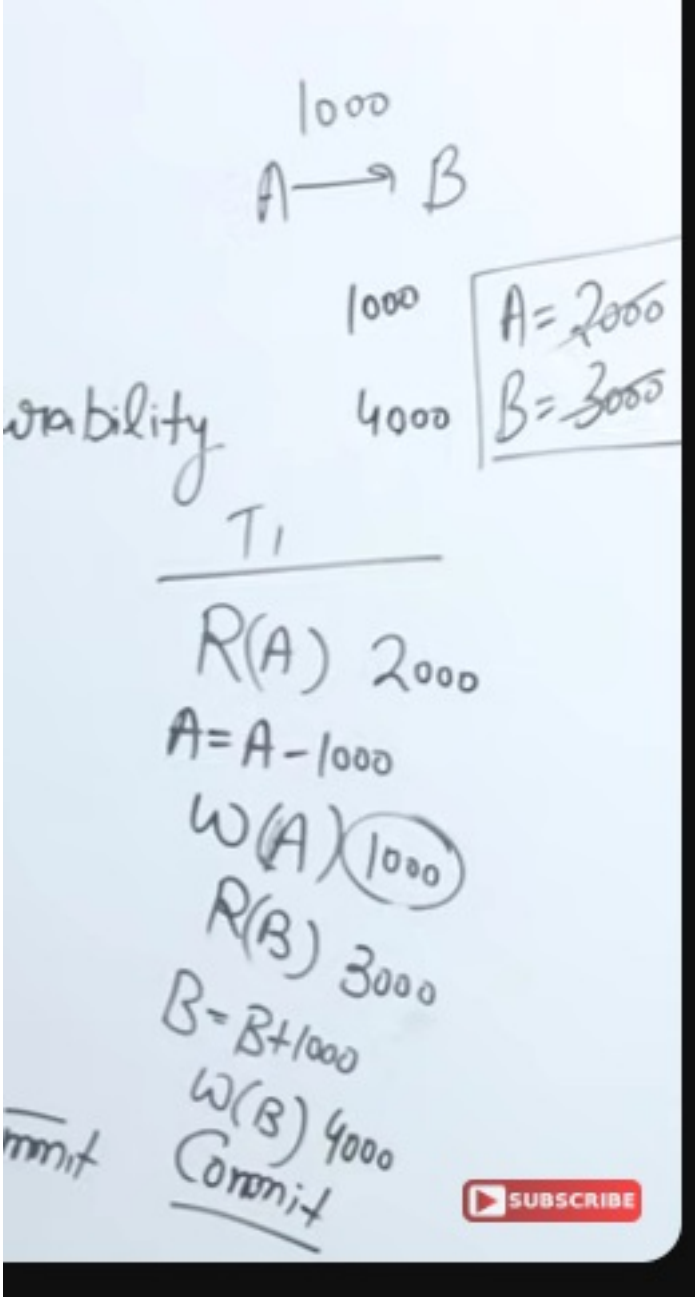


In T1 transaction if before commit an error occurs then the transaction ‘rolls back’ such that no operation is executed

A failed transaction can never be resumed it can only restart.

- **CONSISTENCY**

Before transaction starts and after it is completed , the sum of total money should be same.

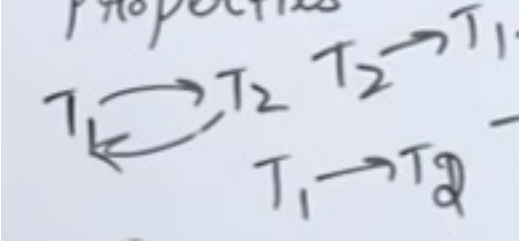
Eg : 

So basically before and after the transaction the total money remained same i.e 5000 (1000+4000 = 2000+3000)

- **ISOLATION (imp)**

Parallel schedule ko serial schedule mei convert krna chaahte

Serial schedule is always consistent isliye it is preferred.



will talk about it more

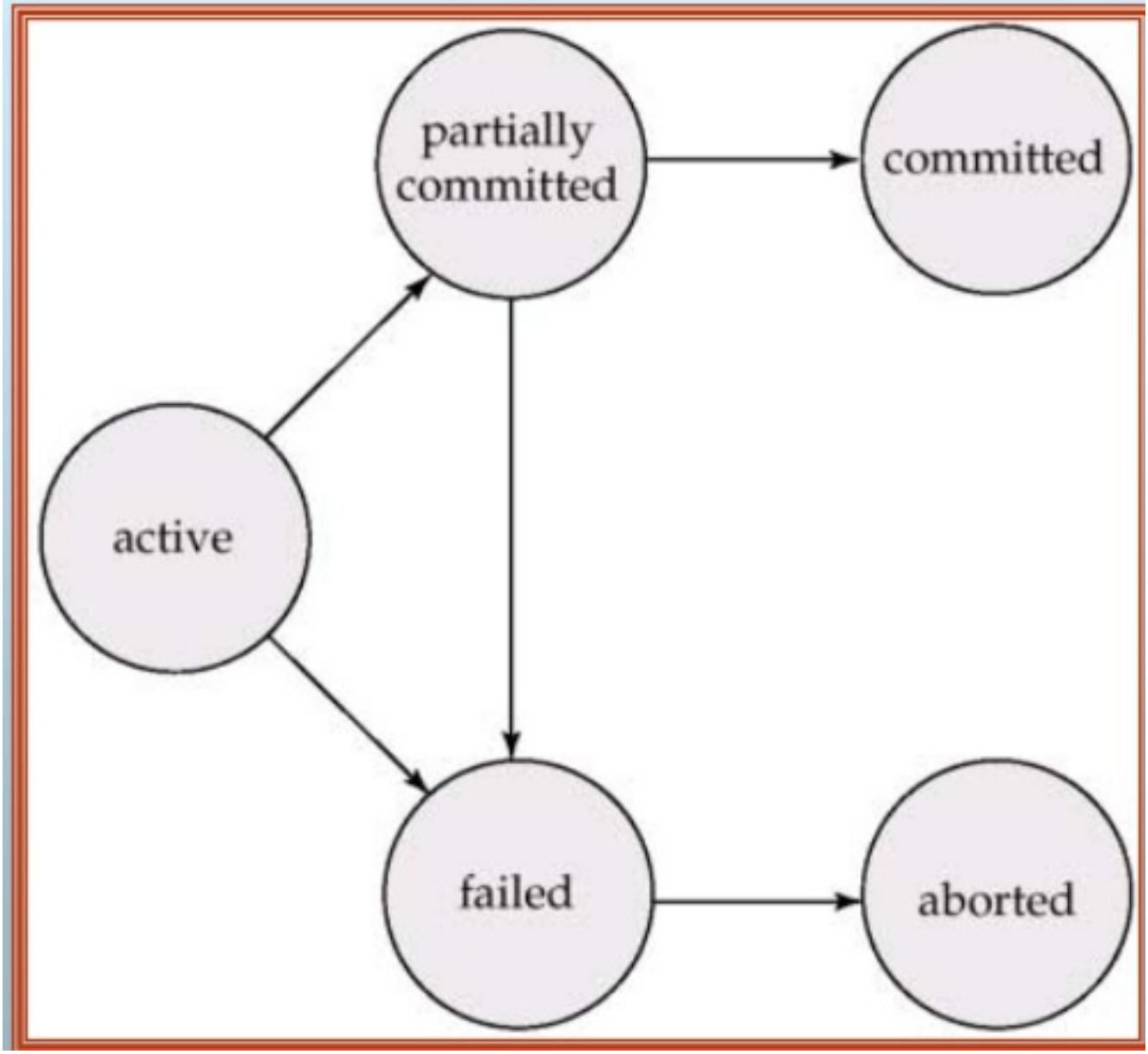
- **DURABILITY**

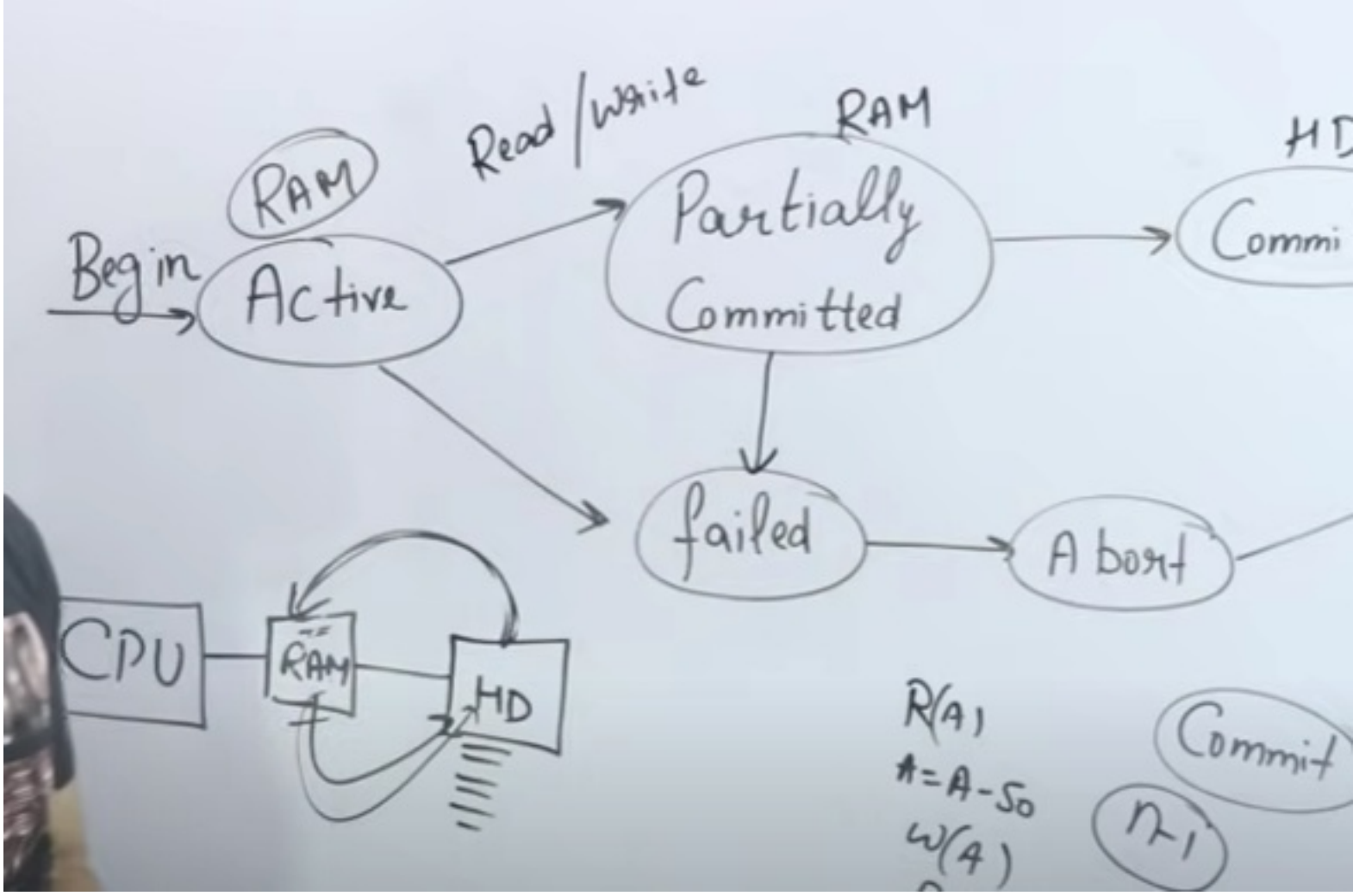
All changes should be permanent

Isliye commit krne k baad hard disk mei save hota hai data jo ki permanently rehta.

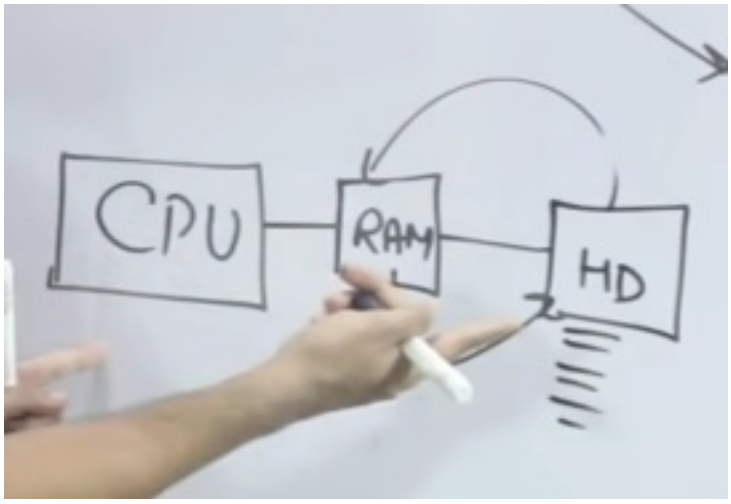
TRANSACTION STATES

active , partially committed , failed , committed , aborted


partially committed
committed
active
failed
aborted



- When we start executing a particular transaction it gets in the active state and goes to ram



- Partially Committed means all the operations have been performed just one is left i.e is COMMIT. RAM mei cheezein horha hoti
- Committed tab hota jab sab execute hojata sab including commit and the data is stored in hard disk
- Terminated : all the shared resources are freed
- Failed : beech mei fail hogya
- Abort : Kill / Restart

SCHEDULE

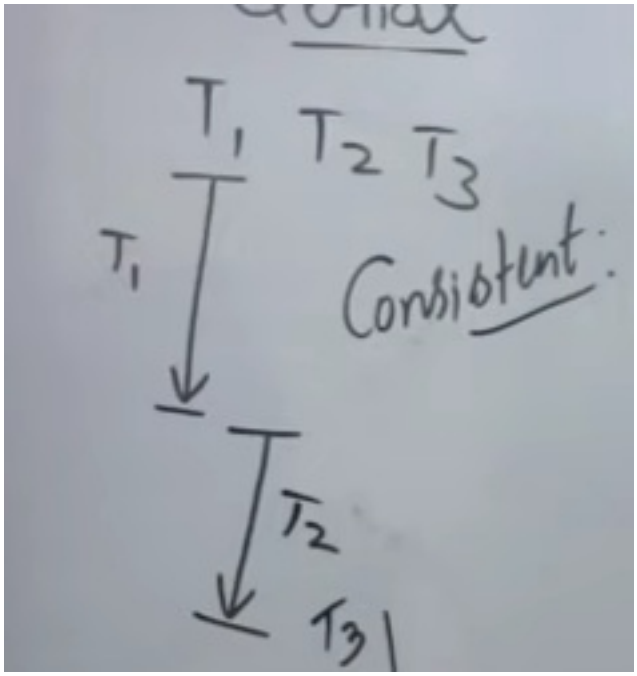
serial VS parallel

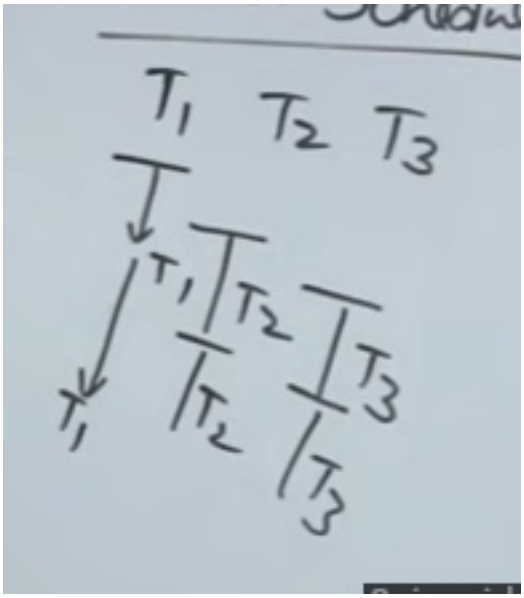
it is chronological execution of multiple transactions.

Serial Parallel

One transaction is fully committed then only Multiple transactions can start together and

another one begins get parallely executed.





Waiting time hojaata kaafi. Throughput increases.

Throughput decreases (number of transactions Nowadays used very much

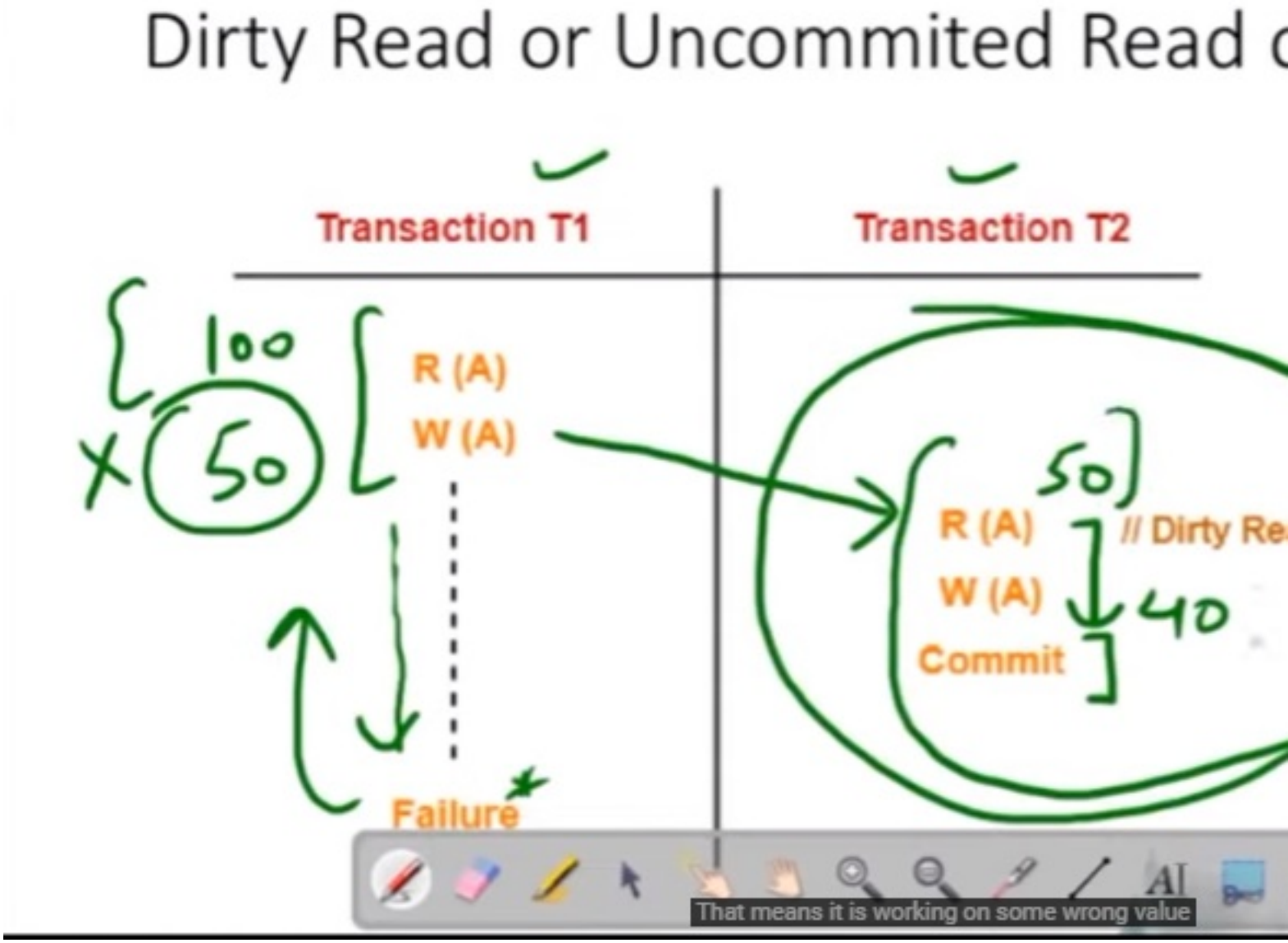
Executed per unit time)

CONCURRENT EXECUTION

CONCURRENCY : when multiple transactions run at a single time (parallel scheduling).

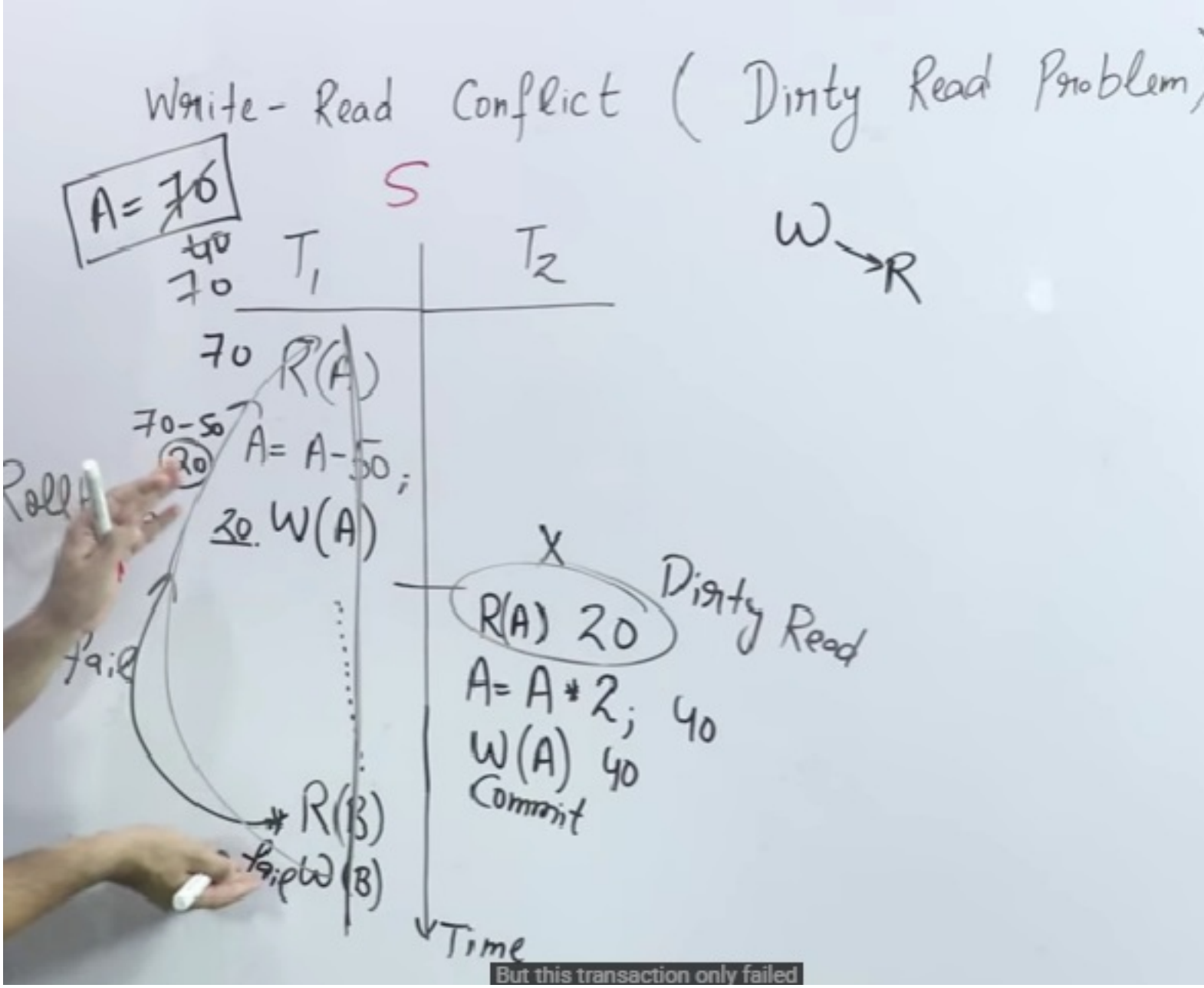
Various problems occur due to concurrency :

- Dirty read
- Incorrect summary
- Lost update
- Unrepeatable read
- Phantom read

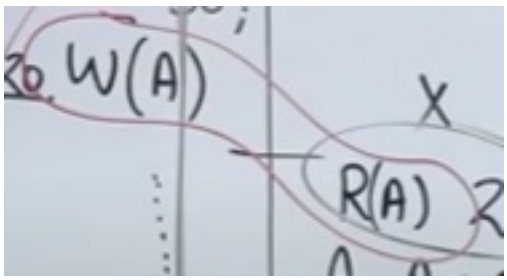


Important sirf write/read hi hai

Write-read conflict(dirty read problem)

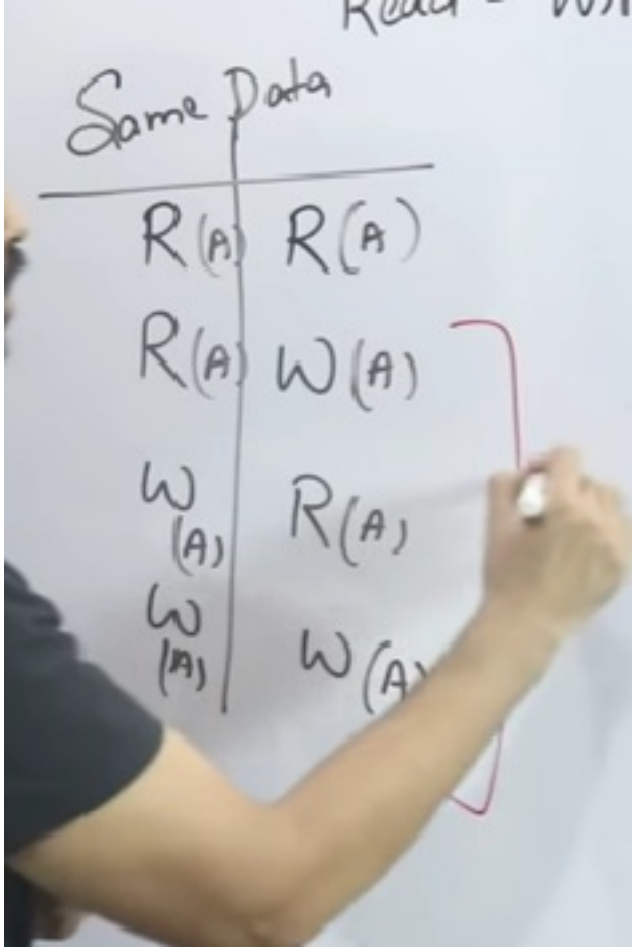


The main problem was wite k baad read aana

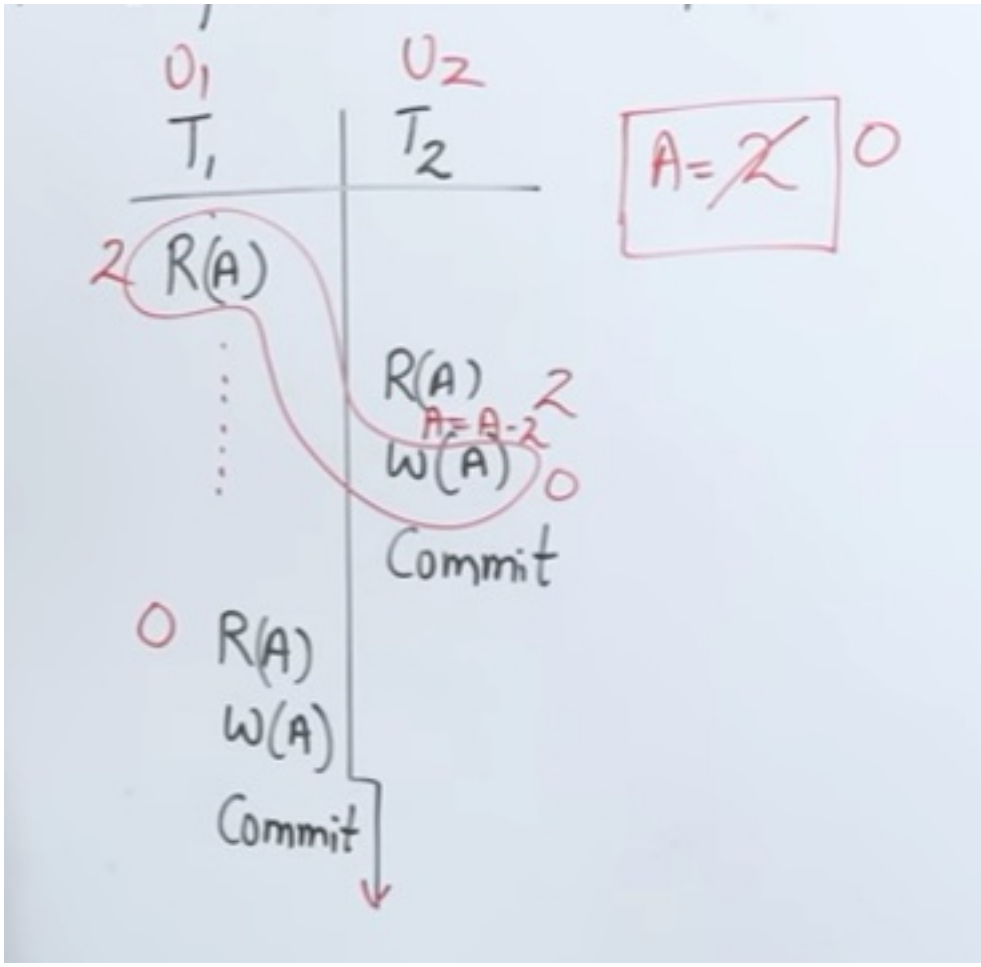


This concept will be used in serializability

Read -Write conflict(unrepeatable read)

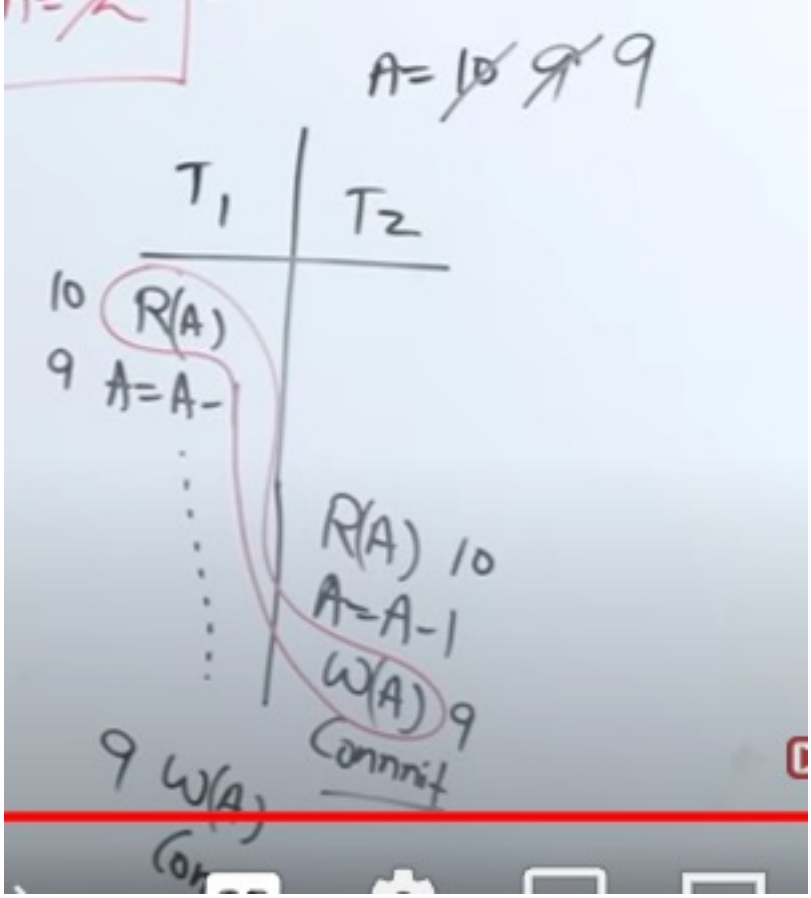


Problem occurs only in 3 cases when write is involved



Read – write conflict occurred and t1 read wrong value so it had to be aborted

Library example :

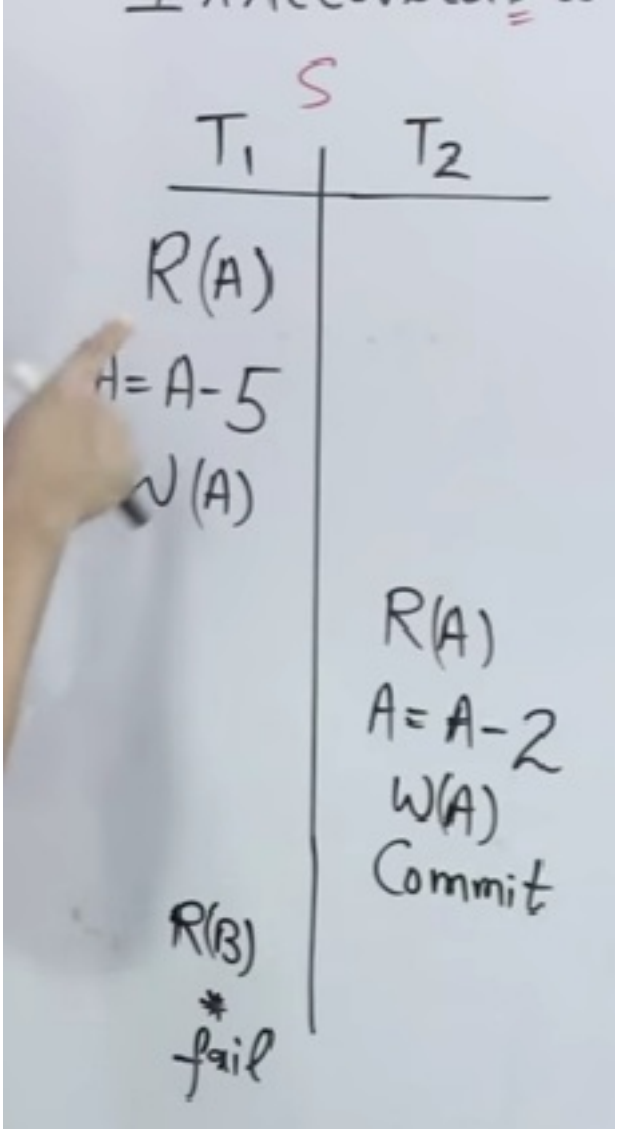


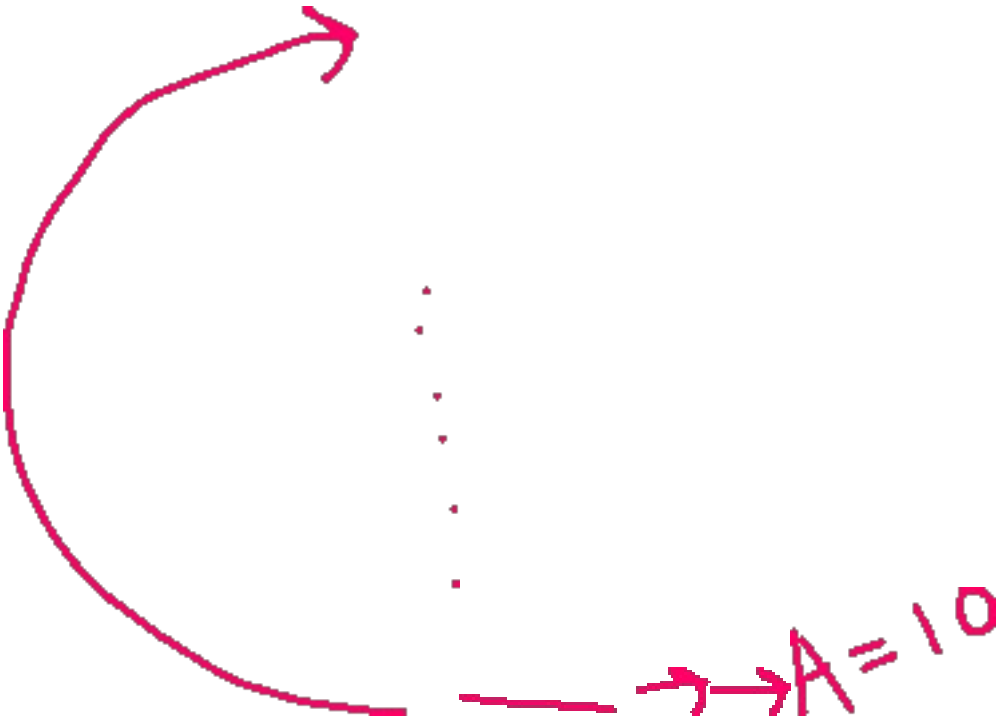
2 books got issued but database has 9 books instead of 8

irrecoverable VS recoverable schedules

while talking of schedules we talk about serializability and recoverability

IRRECOVERABLE SCHEDULE





Let initially value of A be 10

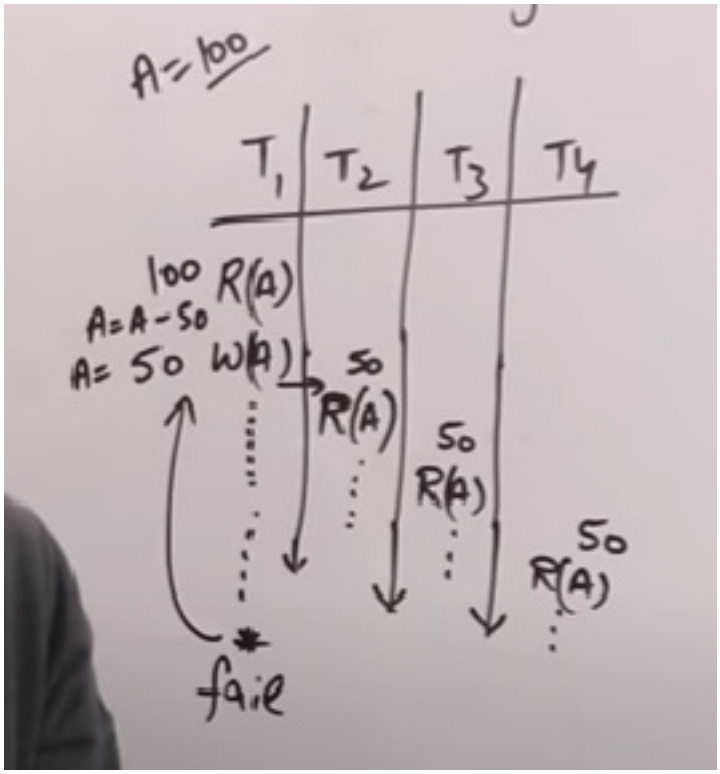
Jaise hi T1 transaction fails then as per atomicity property , T1 will rollback

After it is rolled back , A's value will be updated to 10 again

BUT NOW the changes made by T2 are lost forever , they cant be RECOVERED

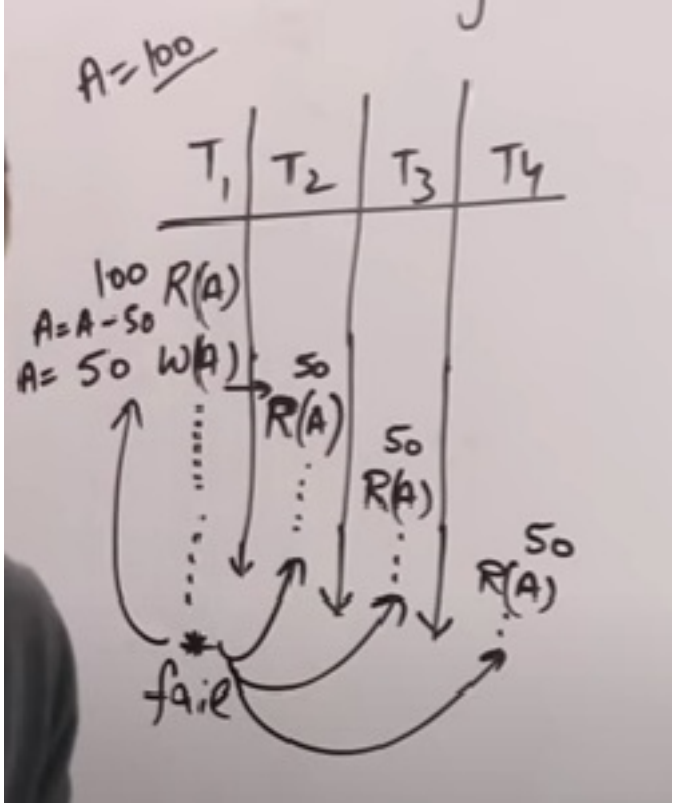
cascading schedule VS cascadeless schedules

cascading : due to occurrence of one event , multiple events are automatically occurring.



As T2,T3,T4 are working on the invalid value of A , they have to be aborted as well !

This is called cascading



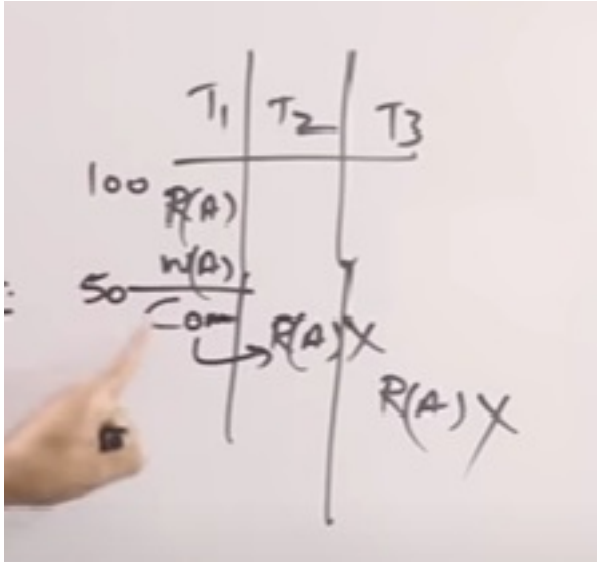
Ek t1 abort hua to usne baaki sbako bhi krdia

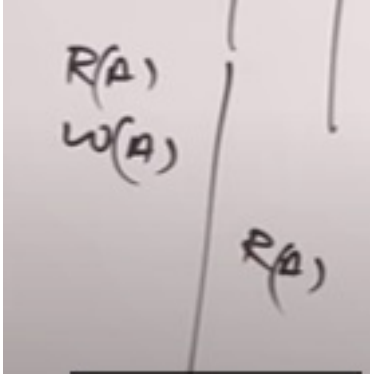
Disadvantage : cpu utilisation nhi kia proper, cpu cycle waste, degraded performance

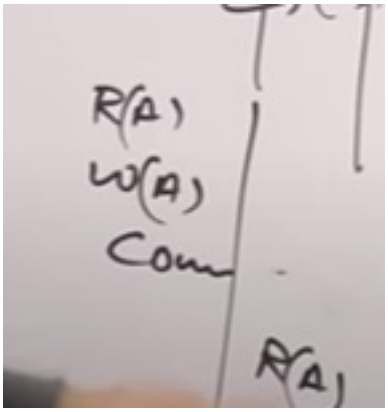
CASCADELESS SCHEDULE ?

How to remove cascading

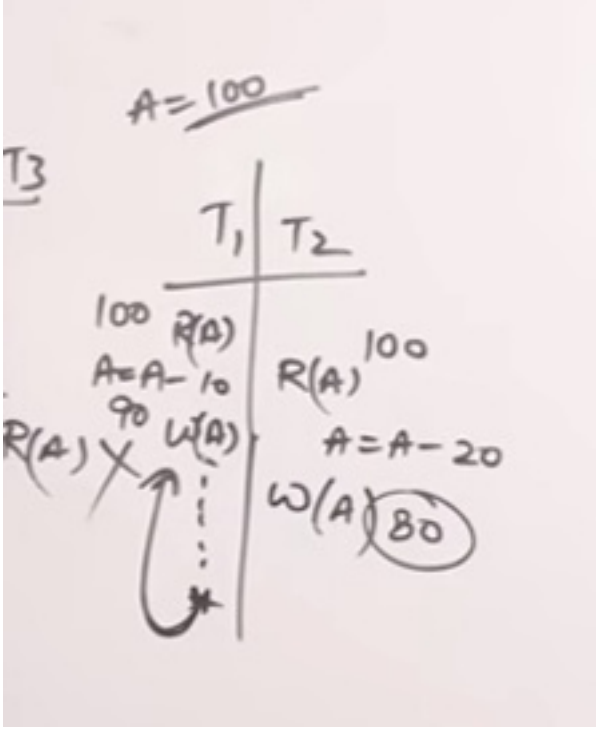
Until the T1 transaction gets committed or aborted , no read(a) would be performed in the subsequent transactions. They can only be read after T1 is committed.



Cascading : 

Cascadeless : 

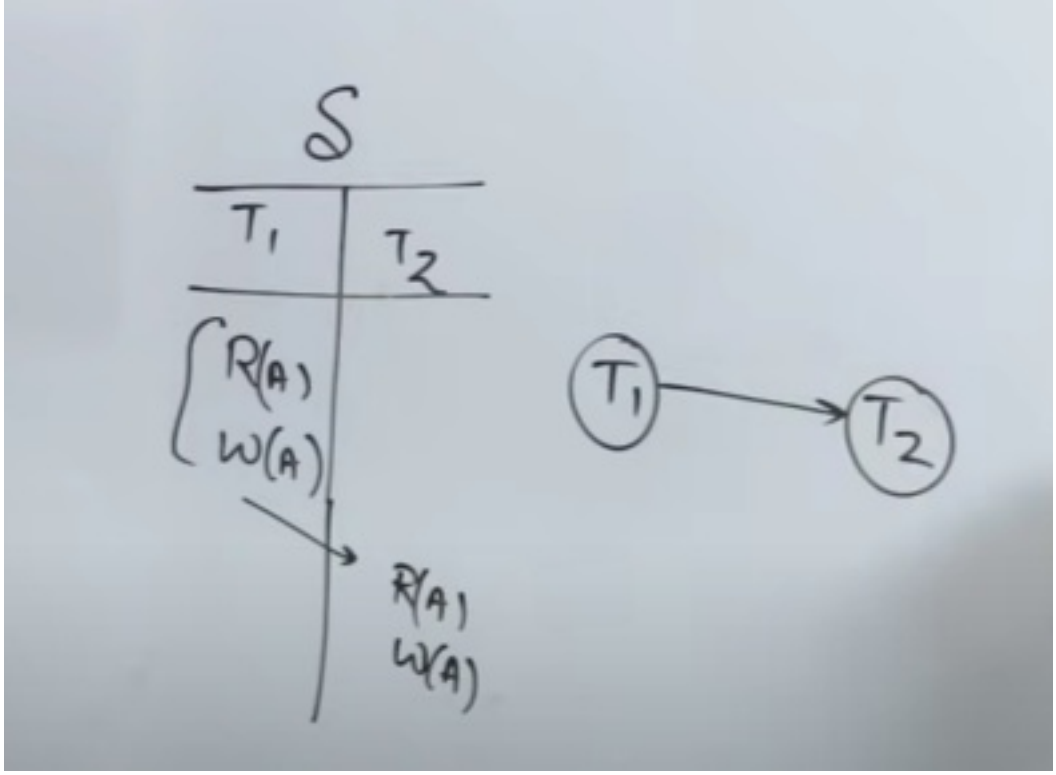
Cascadless mei write write problem aati hai. Cadscadeless sirf write k baad read aane pe reaad ko nhi chalata but write k baad write mei usme dikkat aati

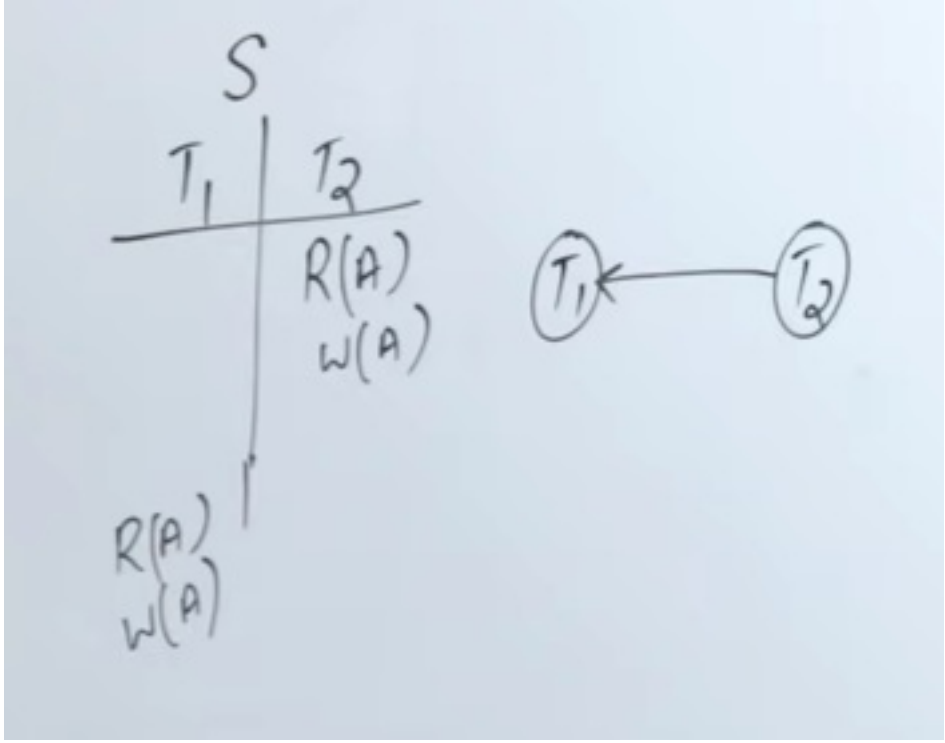


Write k baad wite strict recoverable batata hai

SERIALIZABILITY

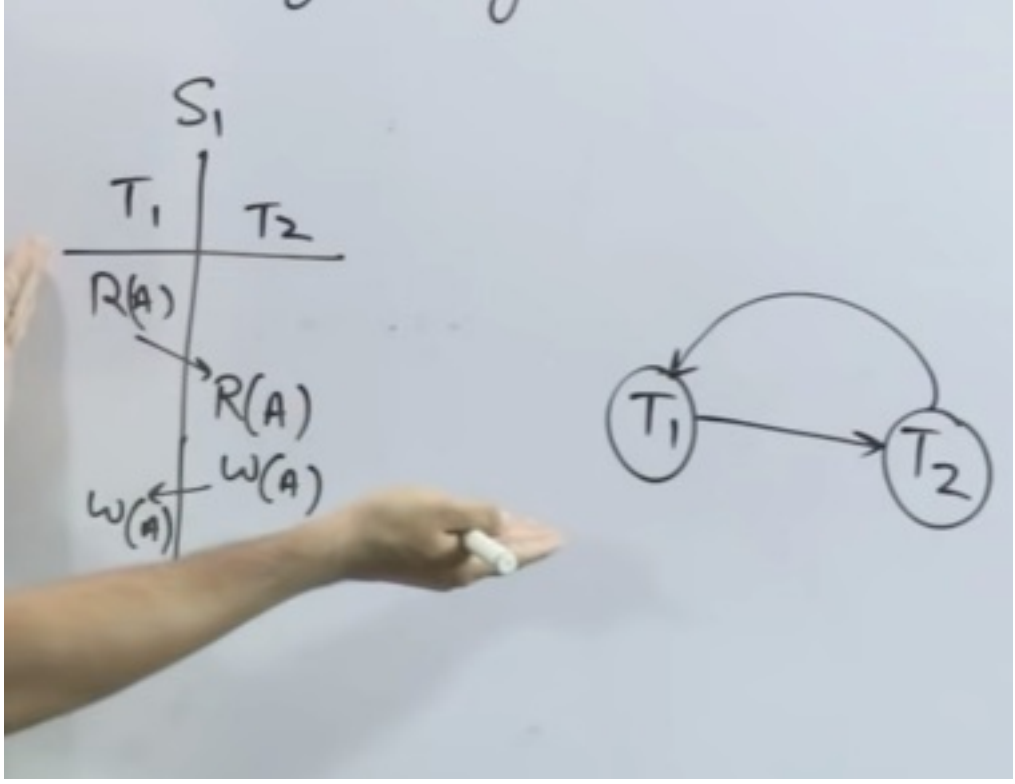
Can a schedule become serializable or not?





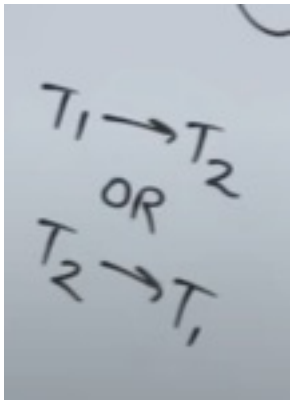
Both the above schedule are serial , so they don't cant be serialized ,obviously only parallel can be

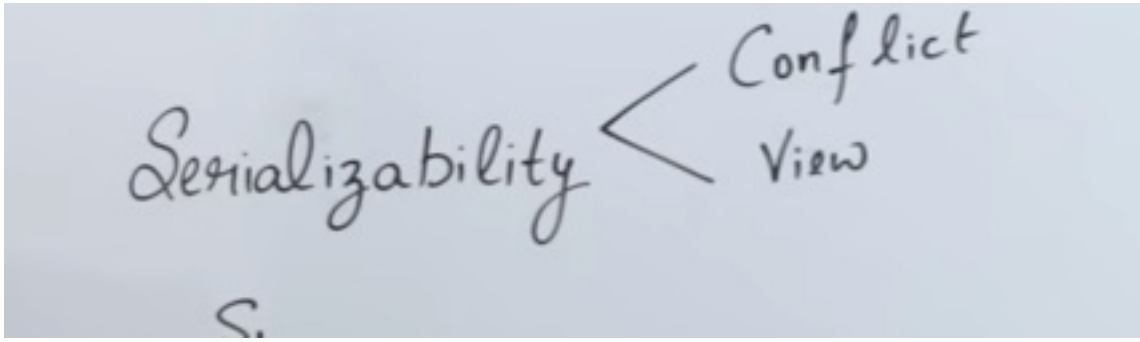
You just need to find a serial clone of a parallel schedule.



Parallel schedules have a loop

To make the above schedule serial we can either:





2 types of serializability : Conflict and View

Another eg :

S

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

$T_1 \rightarrow T_2 \rightarrow T_3$

$T_1 \rightarrow T_3 \rightarrow T_2$

$T_2 \rightarrow T_3 \rightarrow T_1$

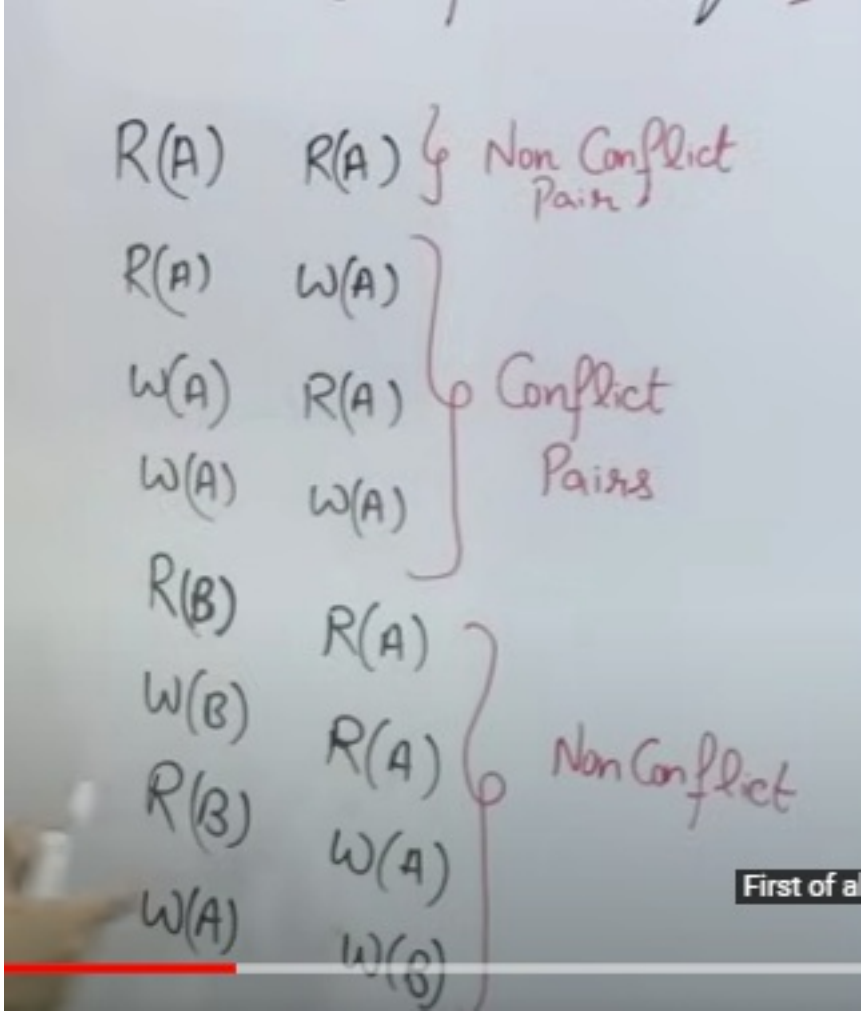
$T_2 \rightarrow T_1 \rightarrow T_3$

$T_3 \rightarrow T_1 \rightarrow T_2$

$T_3 \rightarrow T_2 \rightarrow T_1$

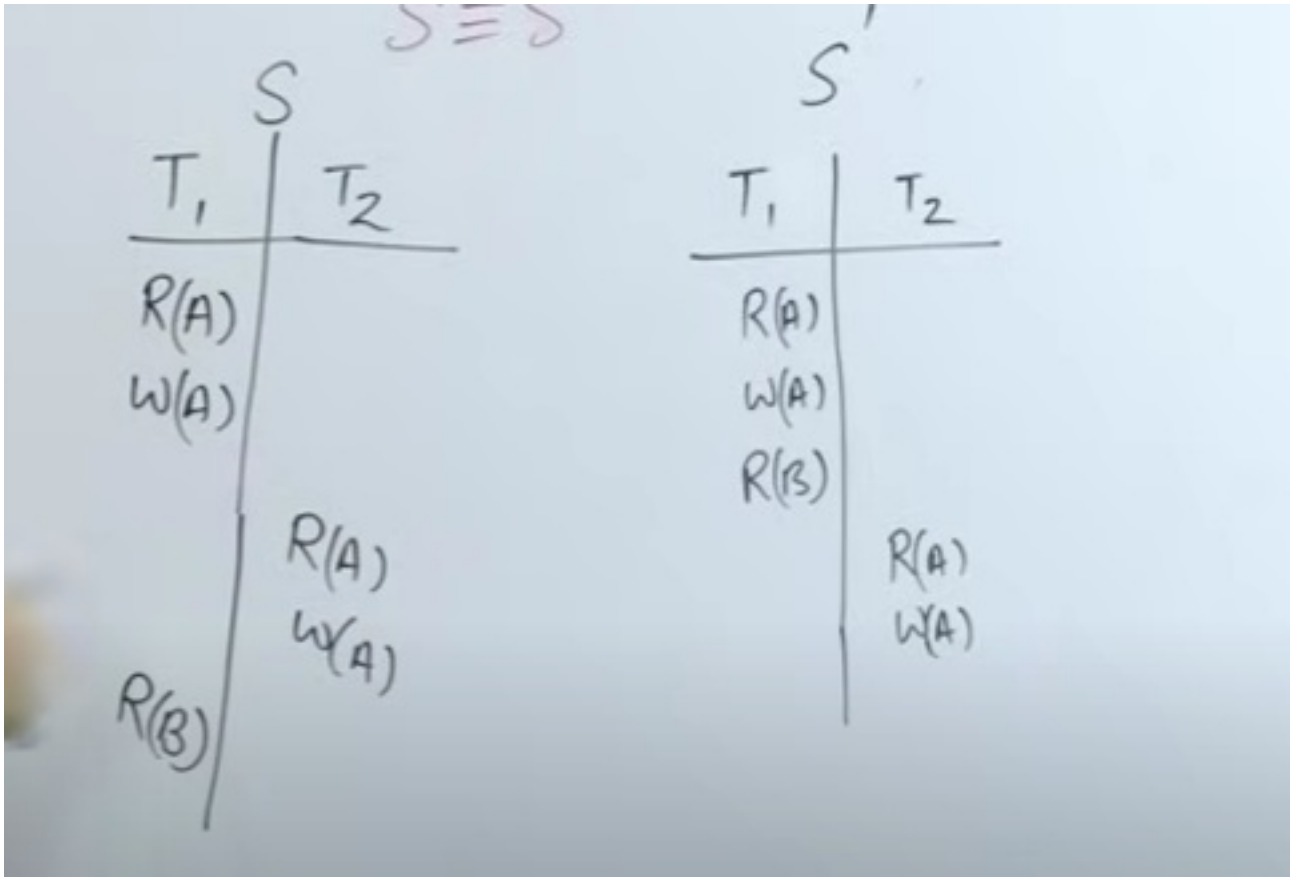
$\longrightarrow$ **conflict equivalent**

we would need to check whether 2 schedules are conflict equivalent or not



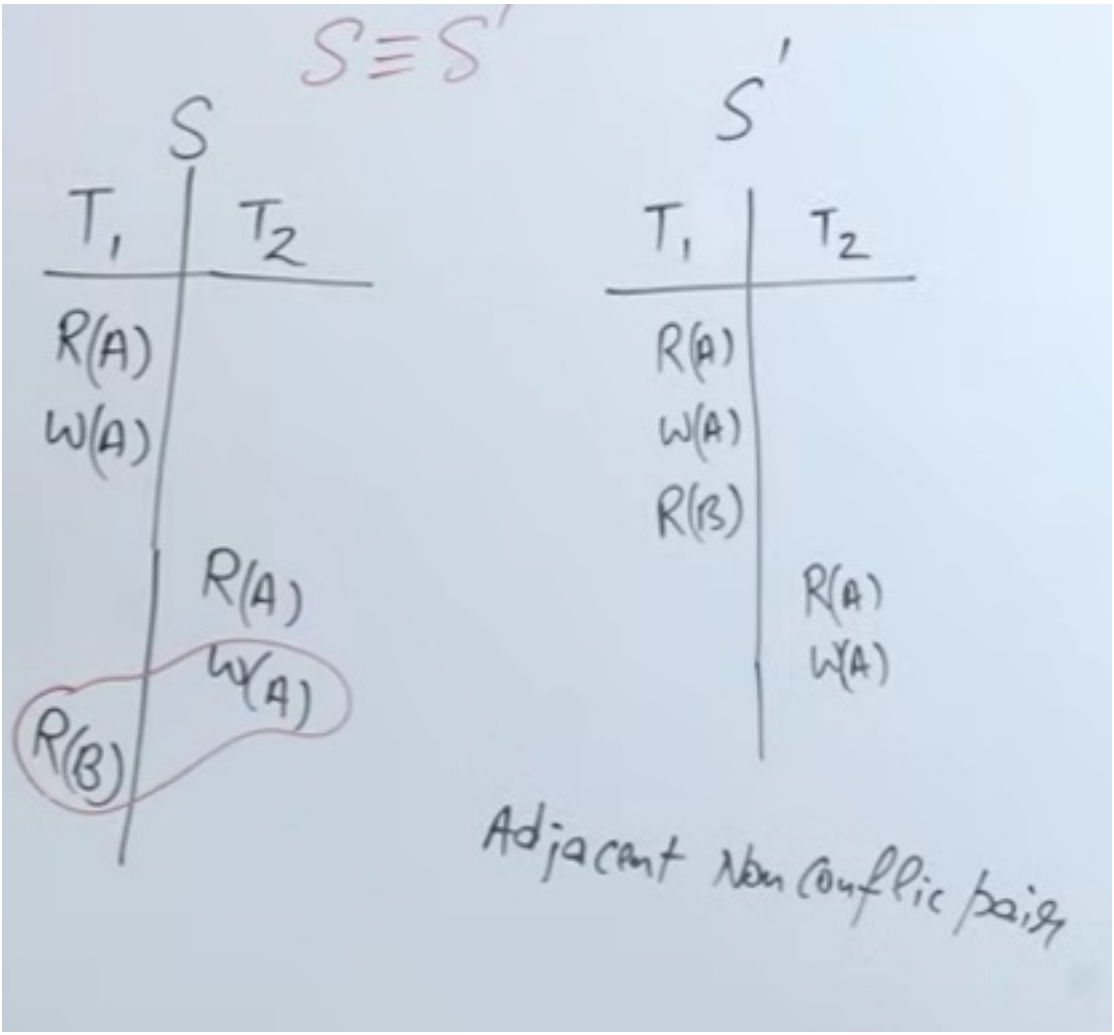
When variables are same and the operation includes read and write then only a conflict occurs.

Check conflict equivalence of s and s’

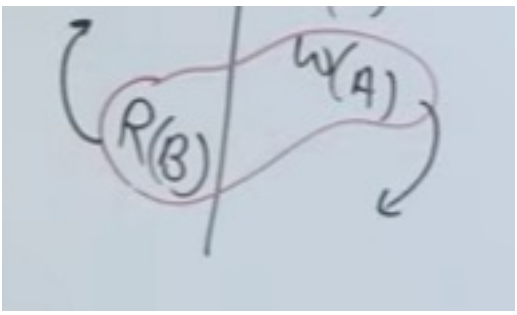


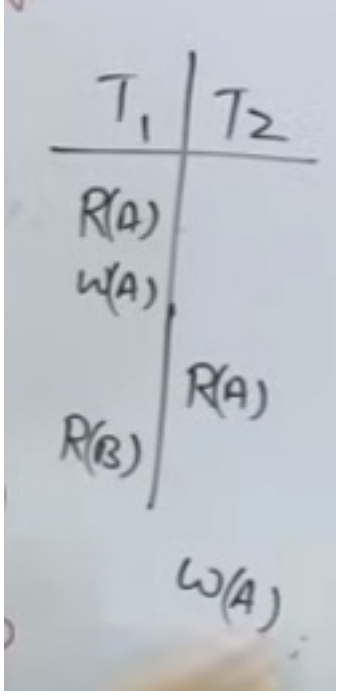
R(B) of T1 is displaced

We need to check for non conflict pair

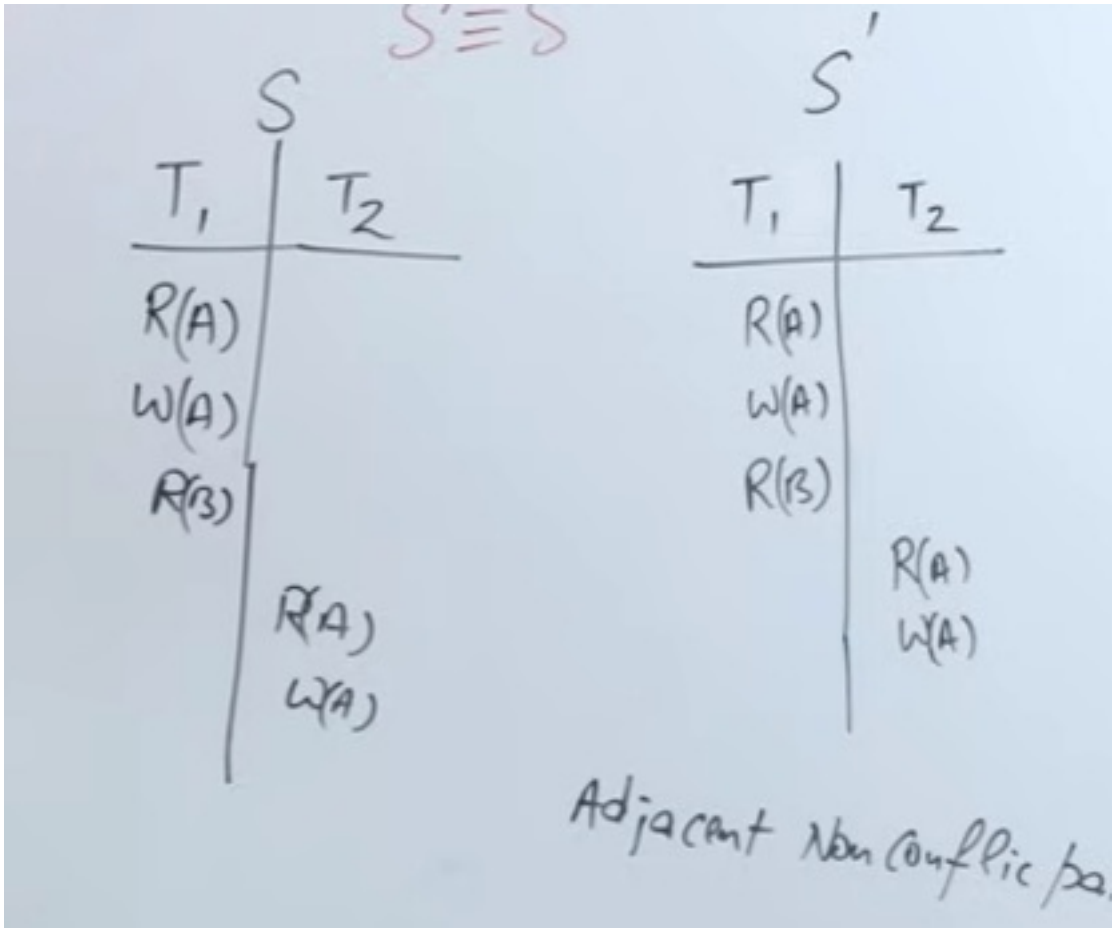


Swap the non conflict vaalon ko





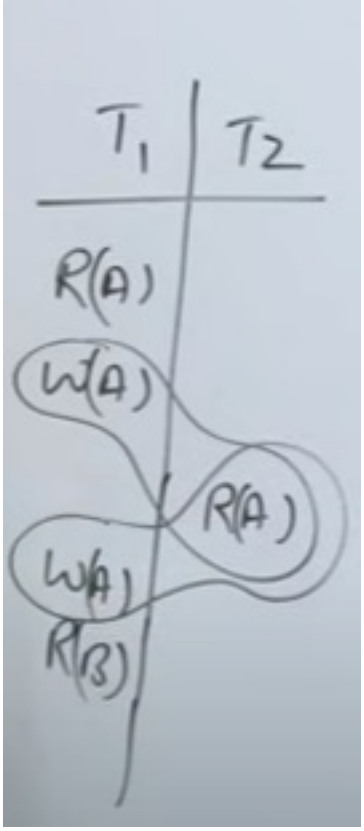
Again an adjacent non conflict pair detected , so swap again



Both the schedules became equivalent

Therefore they were equivalent

If we need to find equivalent for this :

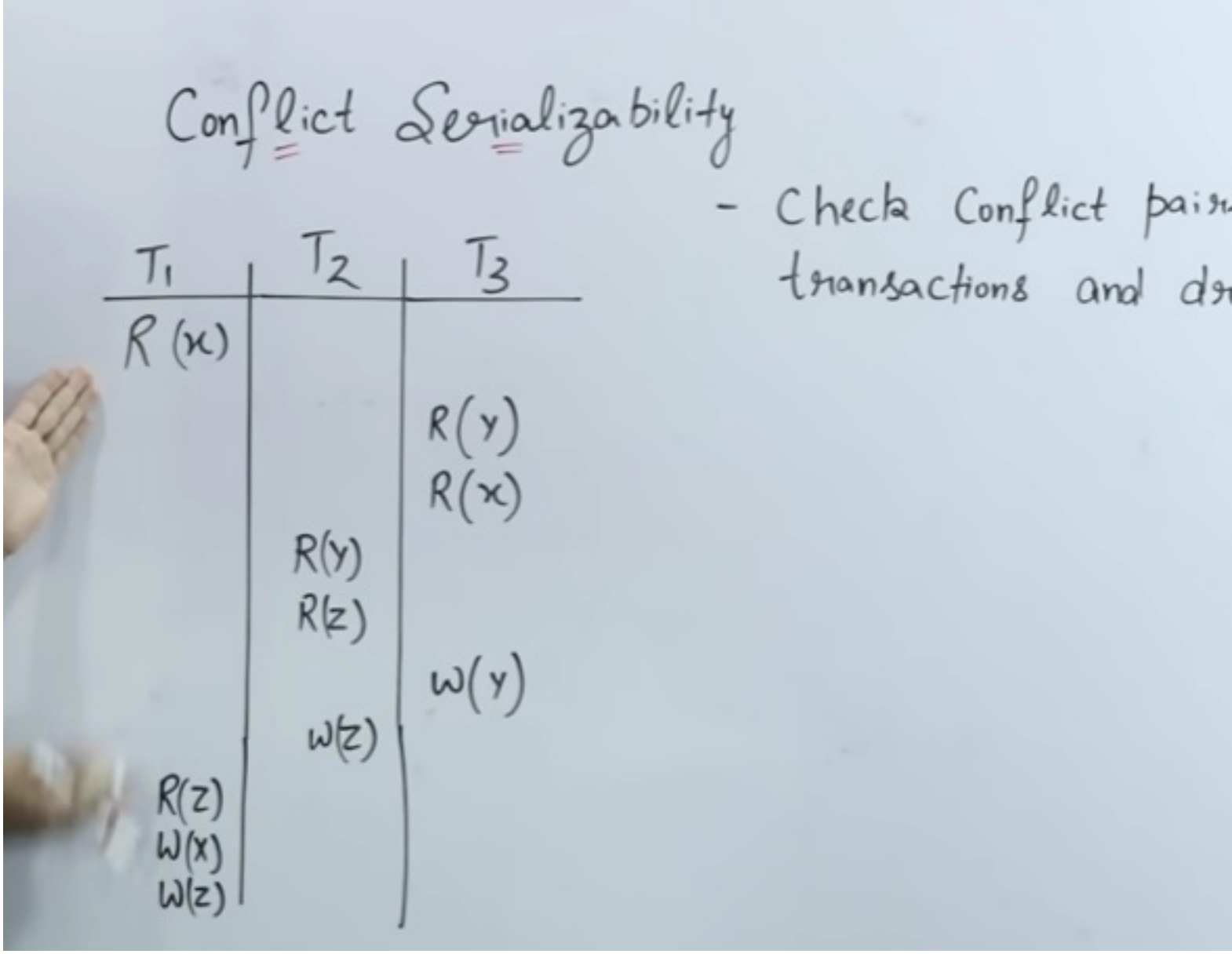


There are 2 adjacent conflict pairs , so no swapping can be done

If a schedule's conflict equivalent exists , then it will always be serializable

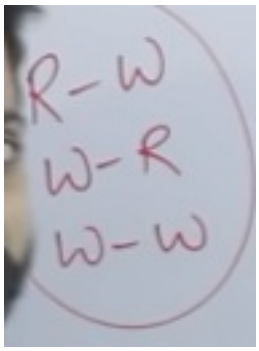
There are other better methods to find conflict equivalents.

how to find conflict equivalent?

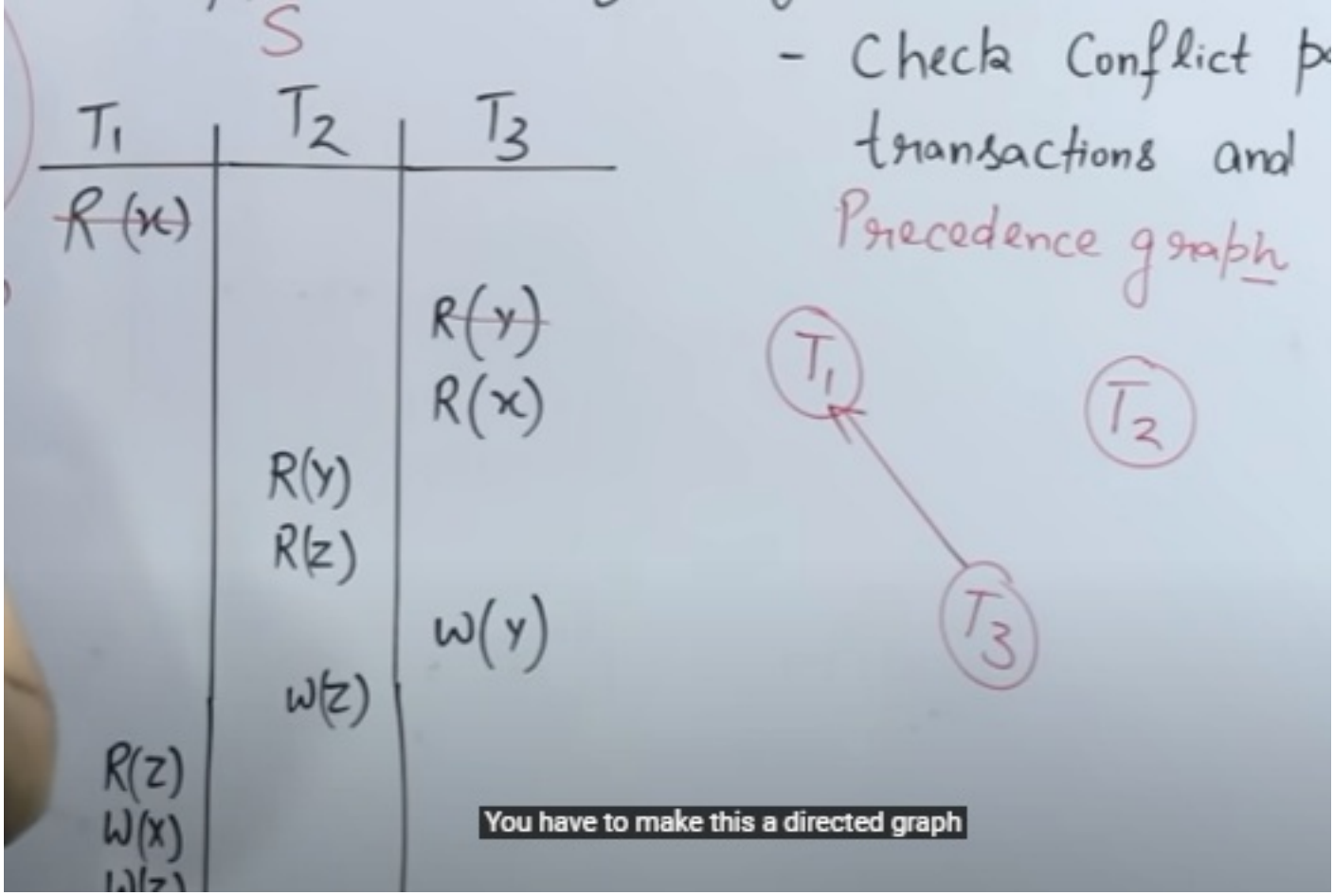


We need to draw a precedence graph.

- The number of vertices would be equal to the total transactions given.
- Check for the following conflict pairs and draw edges

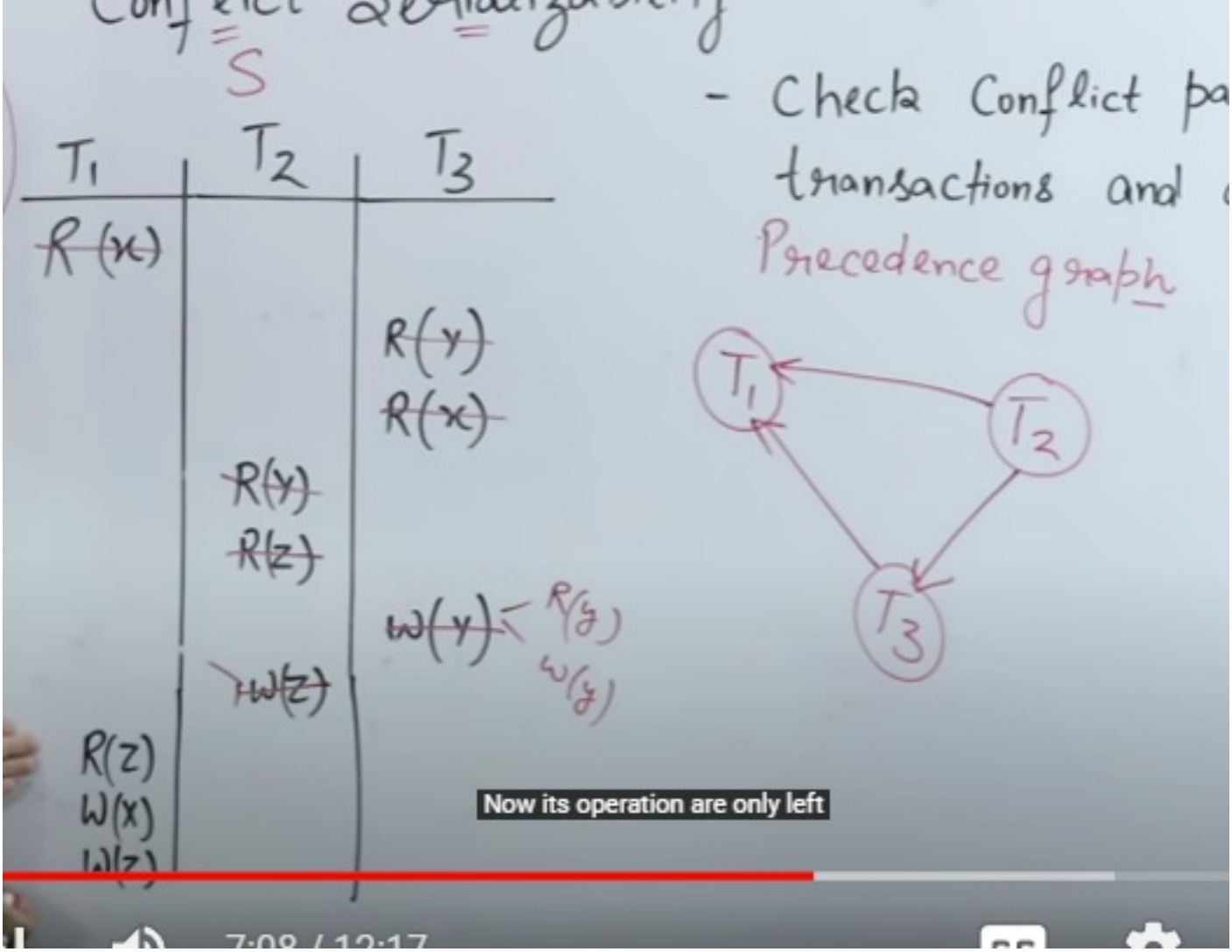


- Here the first conflict occurred here :

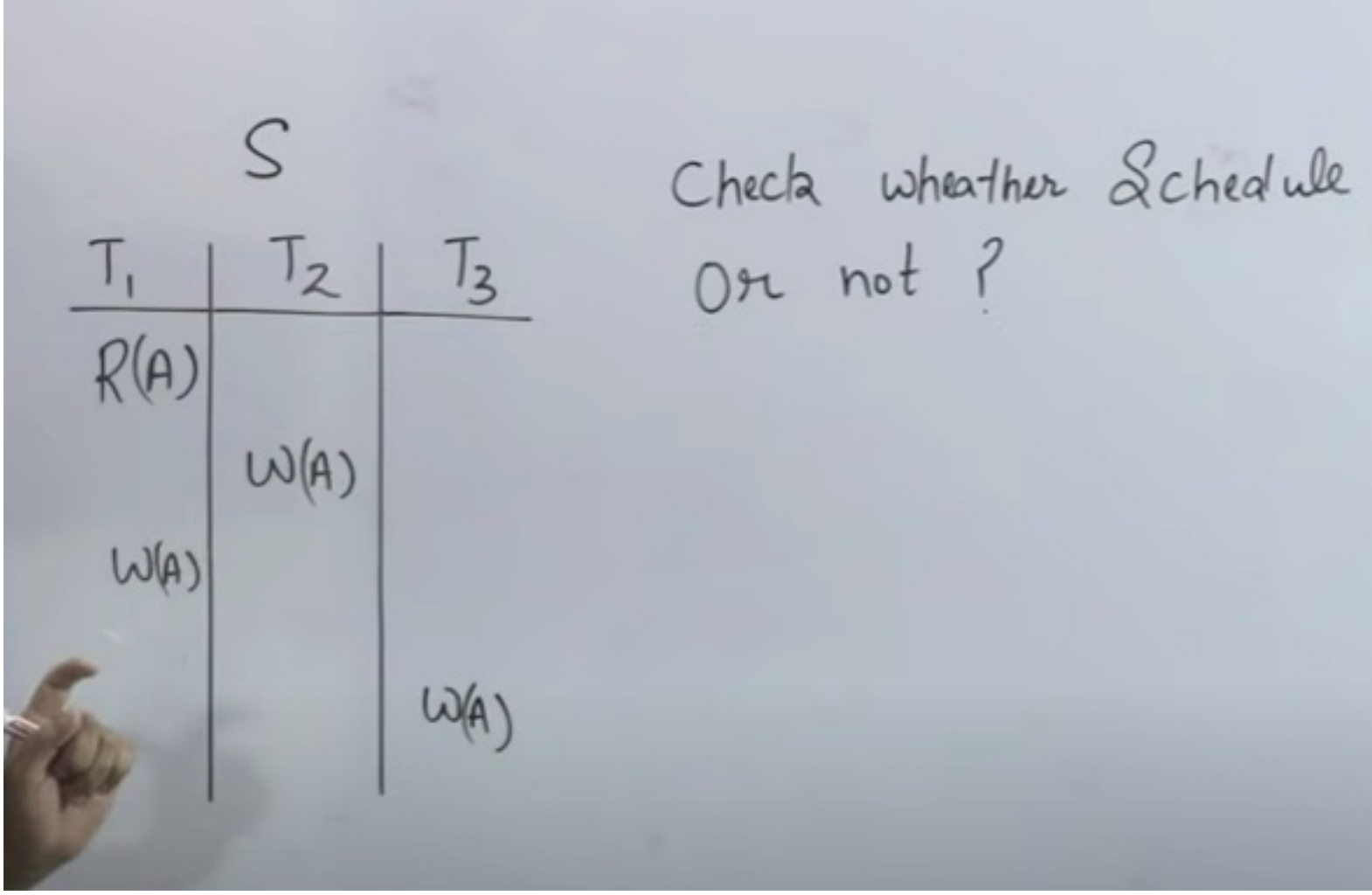


So we drew a directed edge from T3 to T1.

- Also, remember write ka conflict read aur write dono hota hai !
- This would be the resulting graph :

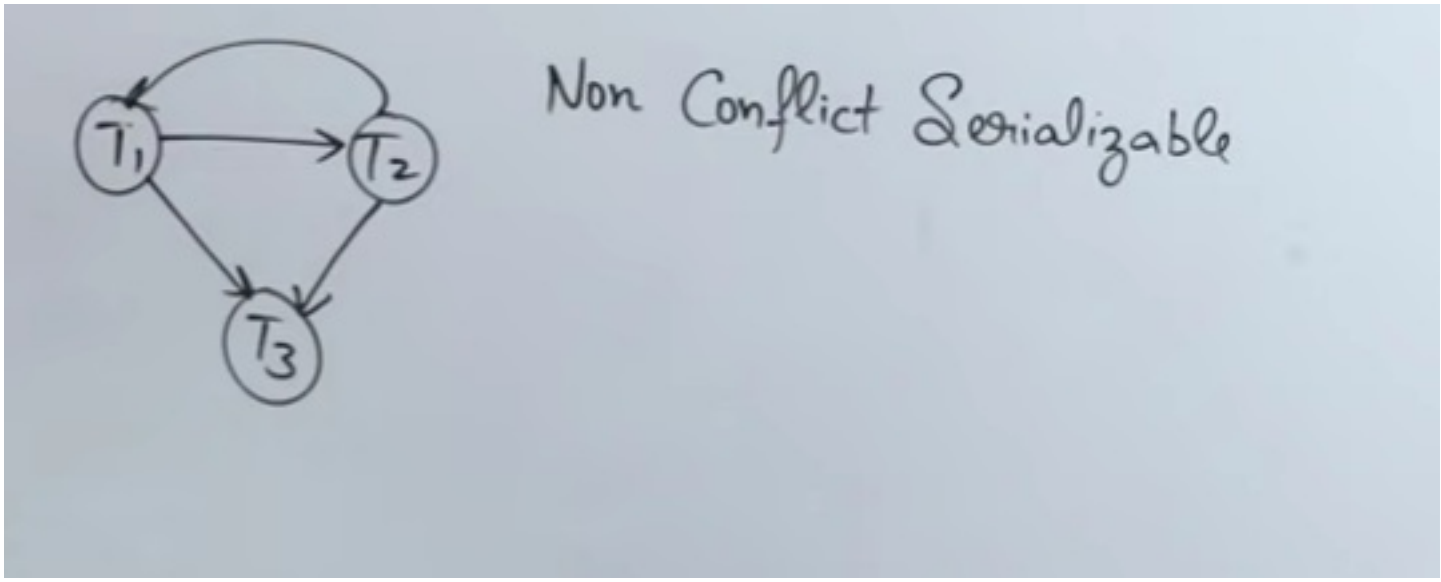


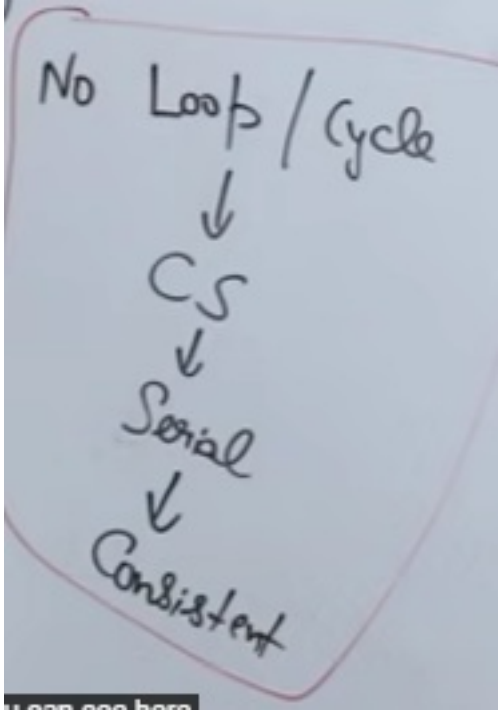
- Check for loop/ cycle -> if doesn't exist -> conflict serializable -> equivalent serial schedule exists -> consistent. If loop exists -> check for view serializability (later)
- What will be the serial schedule of the above parallel schedule?lets find it
- Find the vertex with indegree 0
- **T 2 -> T3 -> T1**
- Basically do topological sort by updating the indegrees and removing the vertex along with connected edges.



Ans :

No it is not. But we don't know yet whether it can be serialized or not. Non conflict serializable ka mtlb ye nhi hota ki serializable hi nhi ho ♀

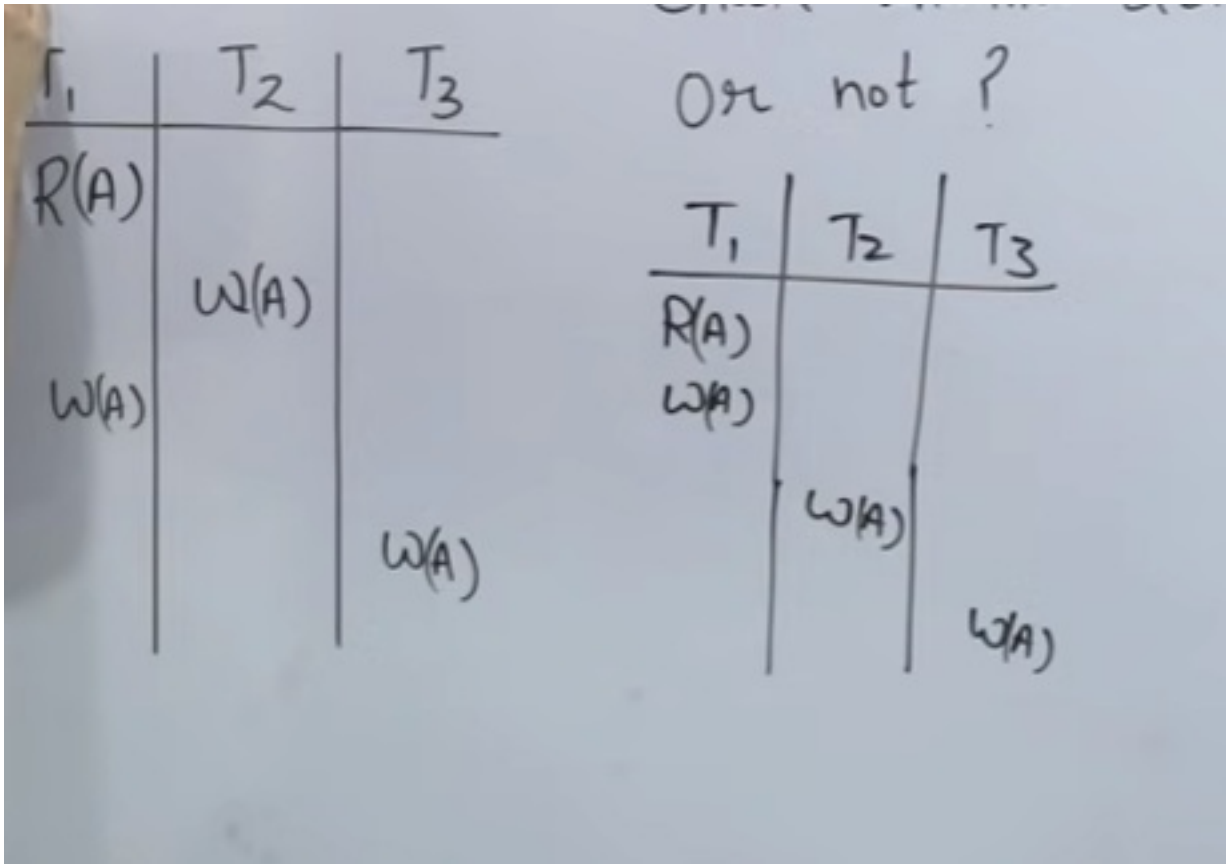




To check non conflict serializable schedule we use VIEW SERIALIZABLE method.

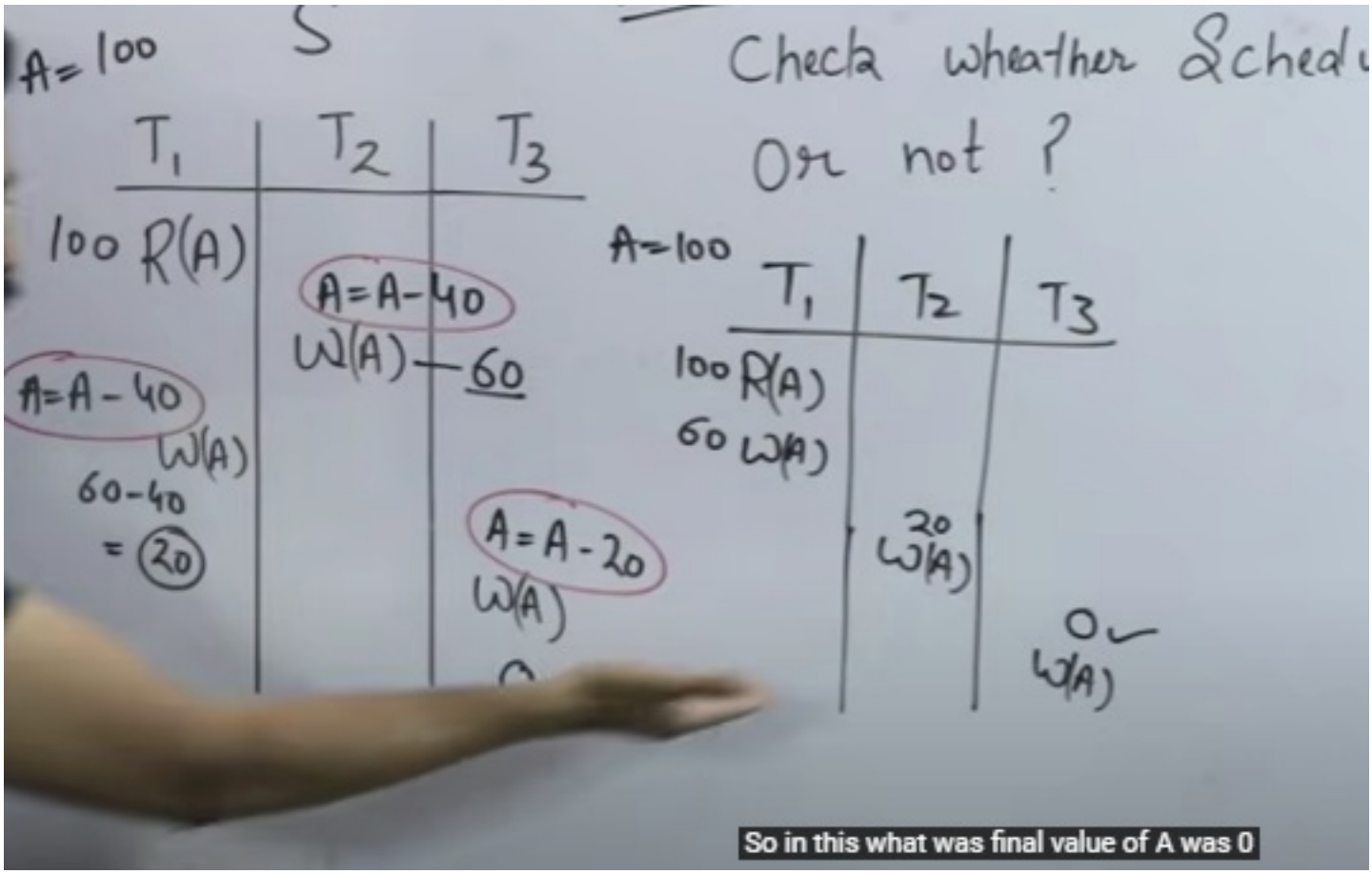
view serializable

do they match ? :



Lets find out

Lets assume A's value to be 100

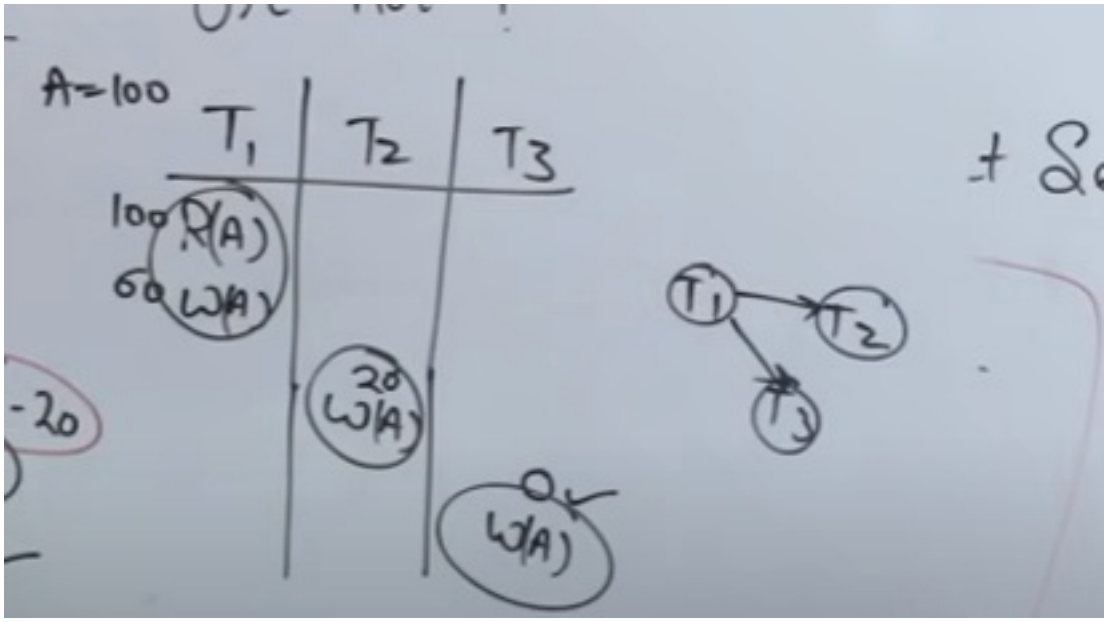


Both are equivalent as dono ka output match horha hai

But they are NOT CONFLICT EQUIVALENT

THEY ARE VIEW EQUIVALENT.

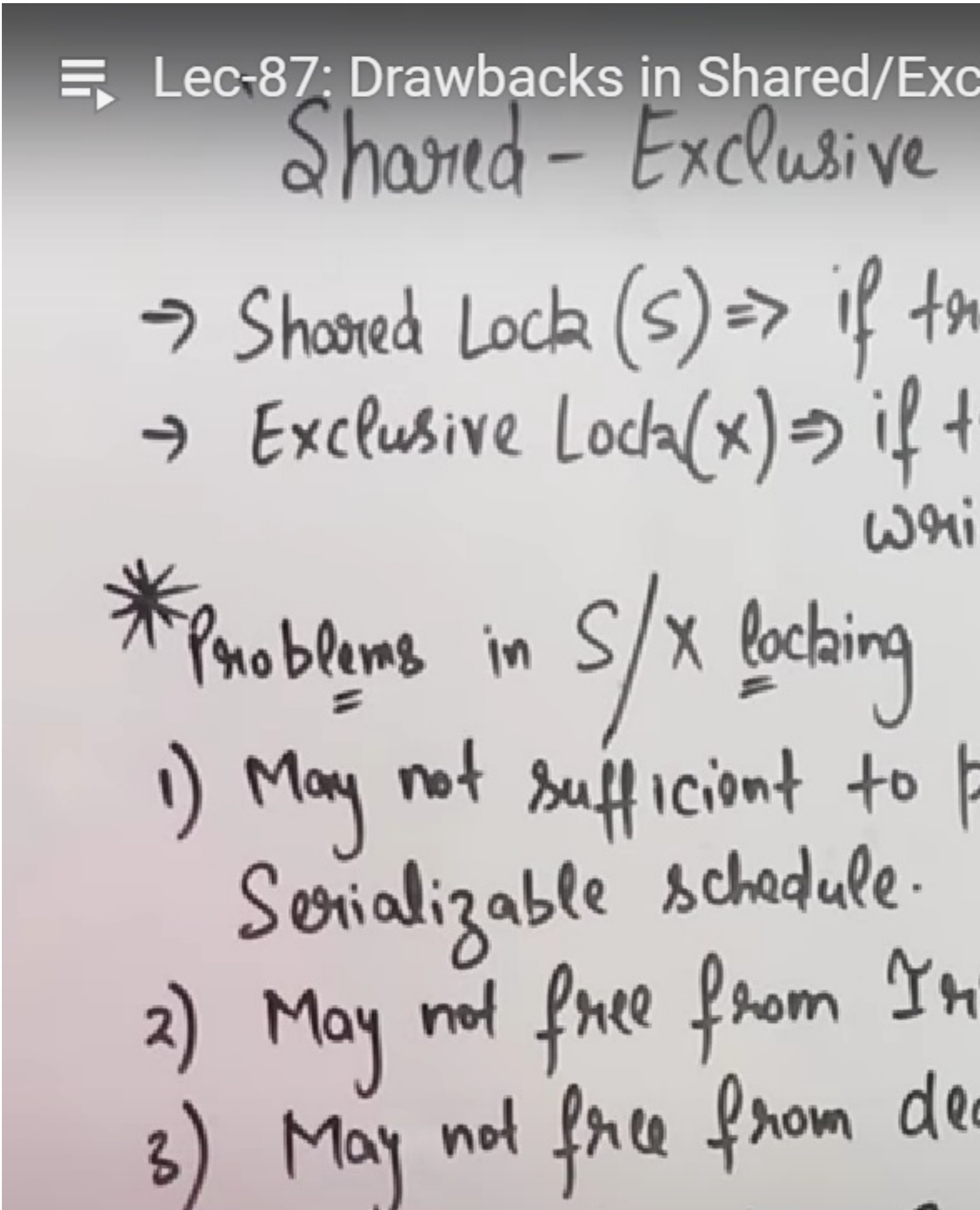
Precedence graph bnake bhi dekhlo

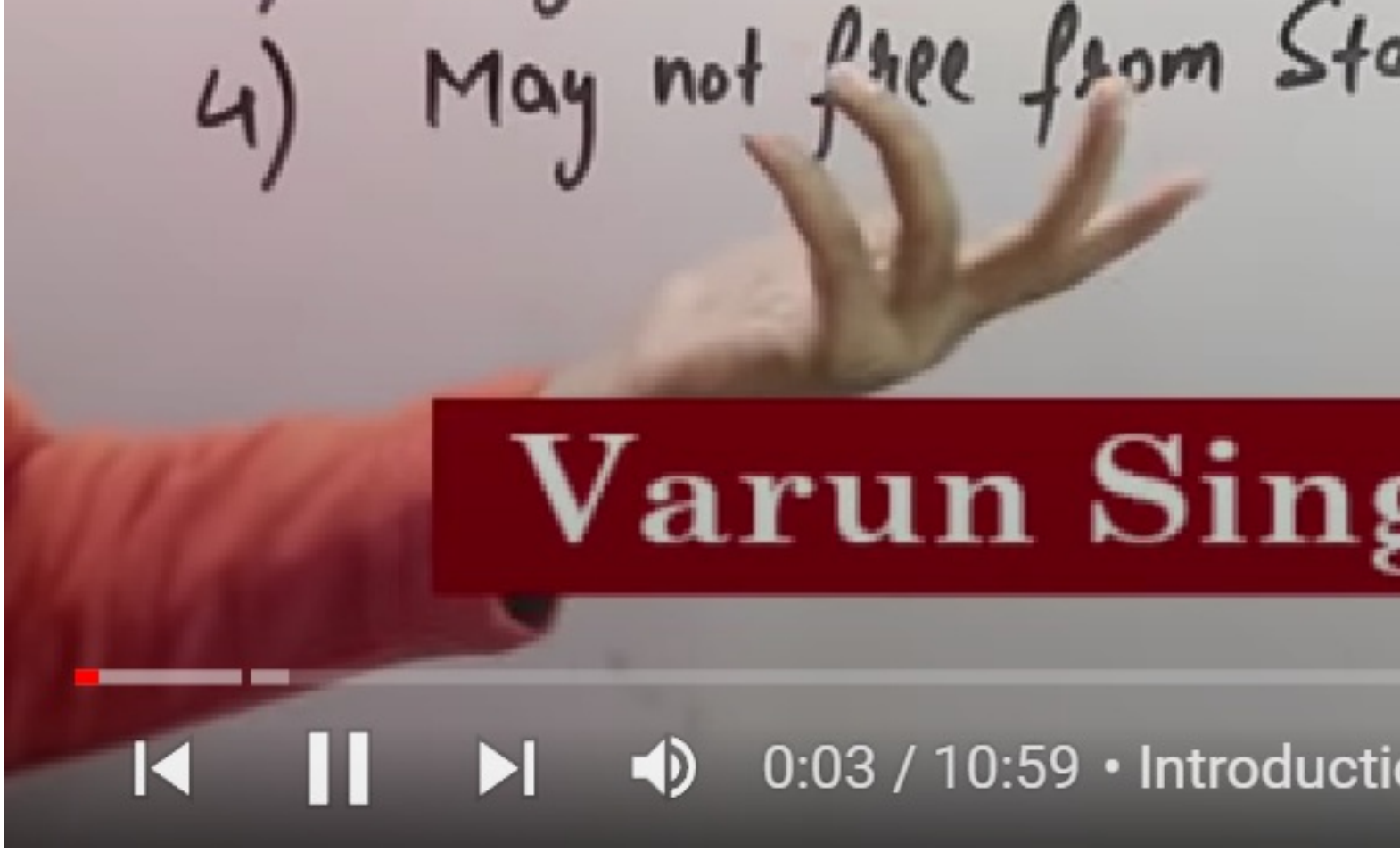


CONCURRENCY CONTROL PROTOCOL

IT AIMS TO SERIALIZE AND MAKE A TRANSACTION RECOVERABLE.

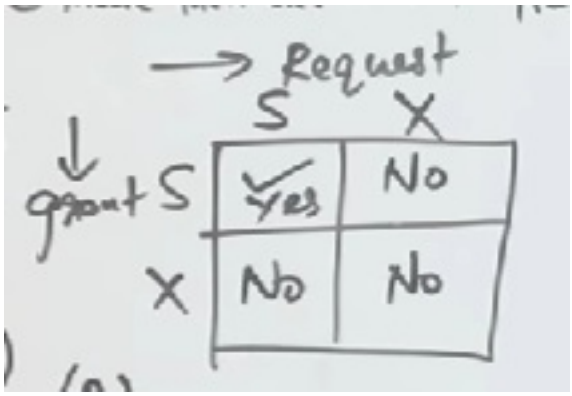
SHARED – EXCLUSIVE LOCKING.





COMPATIBILITY TABLE :

SHARED CAN BE GRANTED OVER SHARED



↓ grant \ → Request	S	X
S	✓ Yes	No
X	No	No

PROBLEMS WITH S/X LOCK

(A)

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

$T_1 \rightarrow T_2$

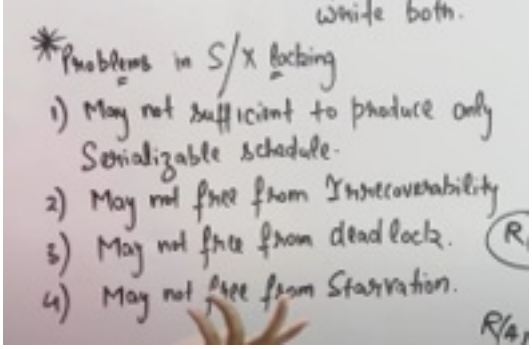
OR

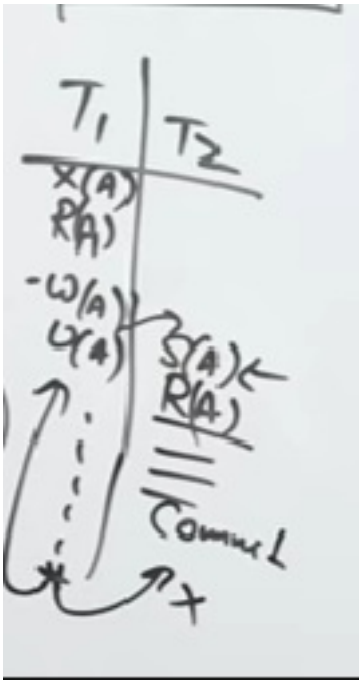
$T_2 \rightarrow T_1$

(A)

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

-> NOT SERIALIZED



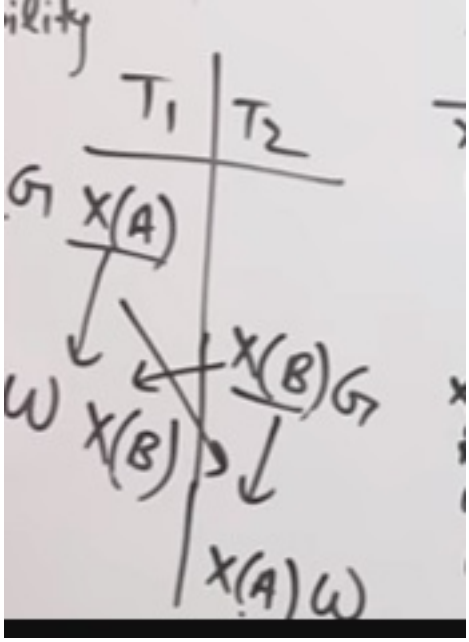


2ND POINT HOGYA PROVE.

(DIRTY READ HOGYA)

NOW LETS DISCUSS WHATS DEADLOCK

Do log resources ki wait krre aur dono infinite loop mei wait krre.

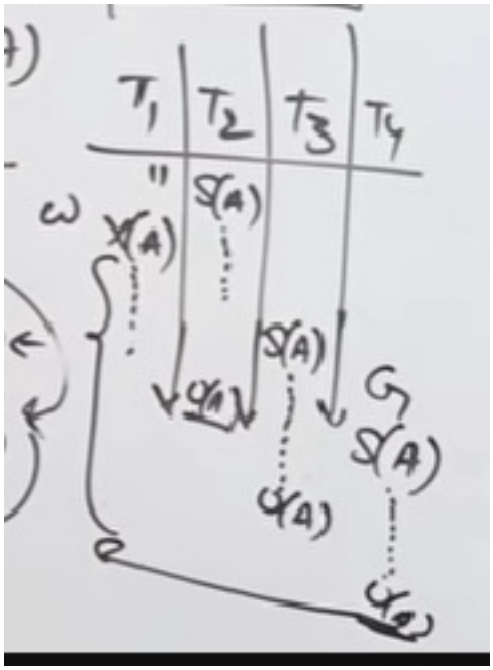


This is deadlock situation

STARVATION?

STARVATION IS LIKE DEADLOCK ONLY PR NOT FOR INFINITELY.

Since shared can be granted above a shared lock so T1 will have to wait till T4 ‘s shared lock has been unlocked.



Since we know when t1 will be granted the X(A) , it is called starvation.

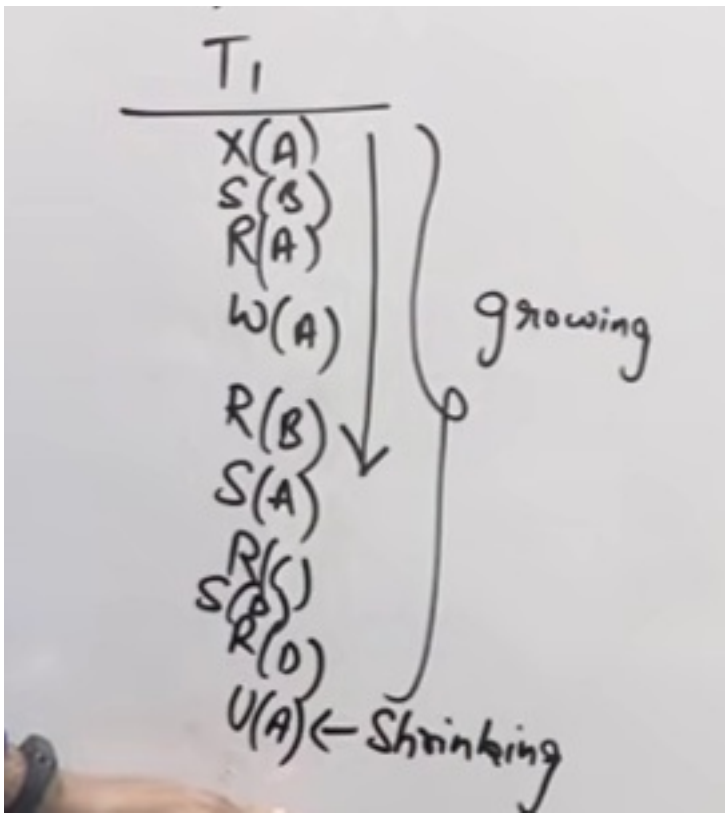
CONCURRENCY CONTROL PROTOCOL

2 PHASE LOCKING(2PL)

MODIFICATION OF S/X ONLI

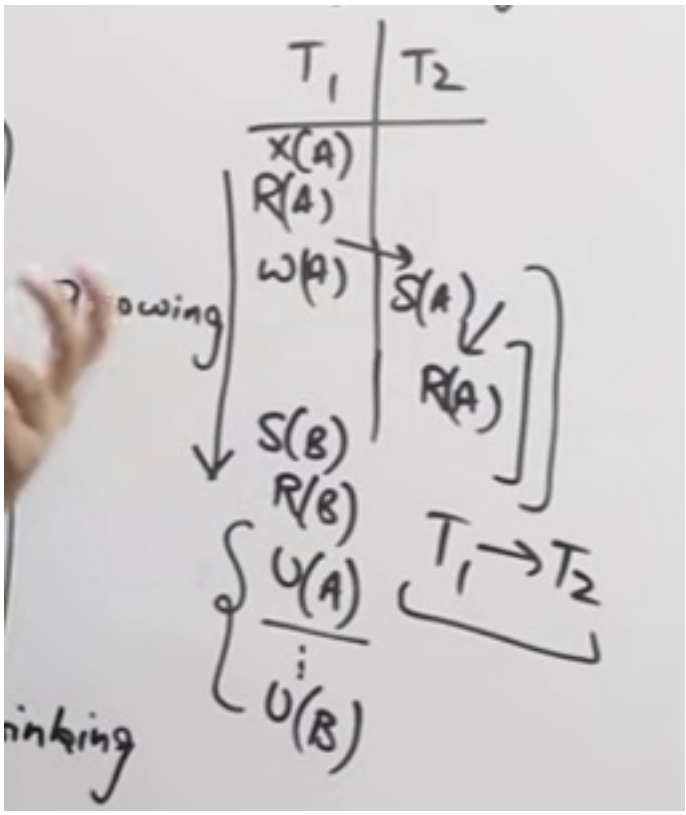
→ Growing phase: locks are acquired and no locks are

→ Shrinking phase: locks are released and no locks are

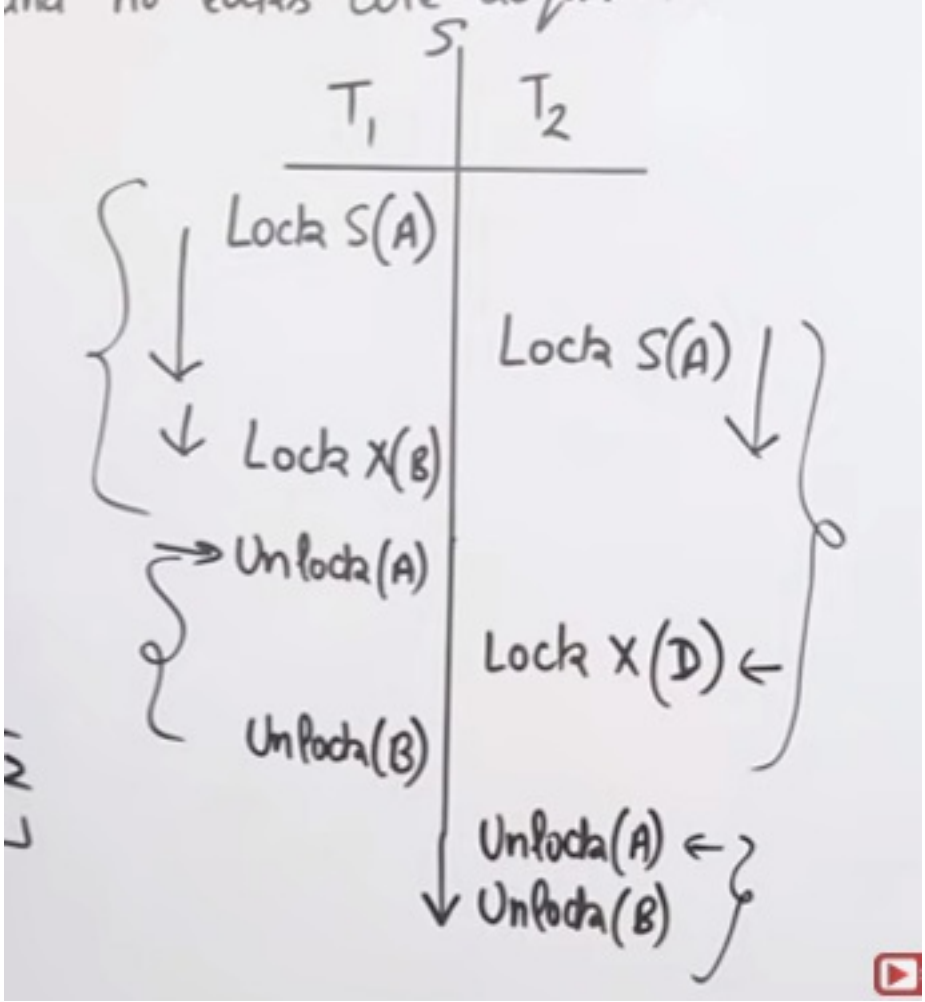


After the shrinking phase no locks will be acquired but only released

Through this, we can achieve serializability and hence consistency.

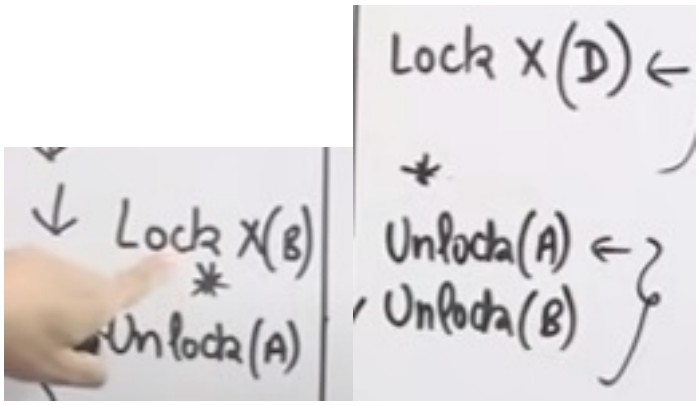


All the 2PL transactions are consistent.



To check what serializability is achieved we see where is the lock point.

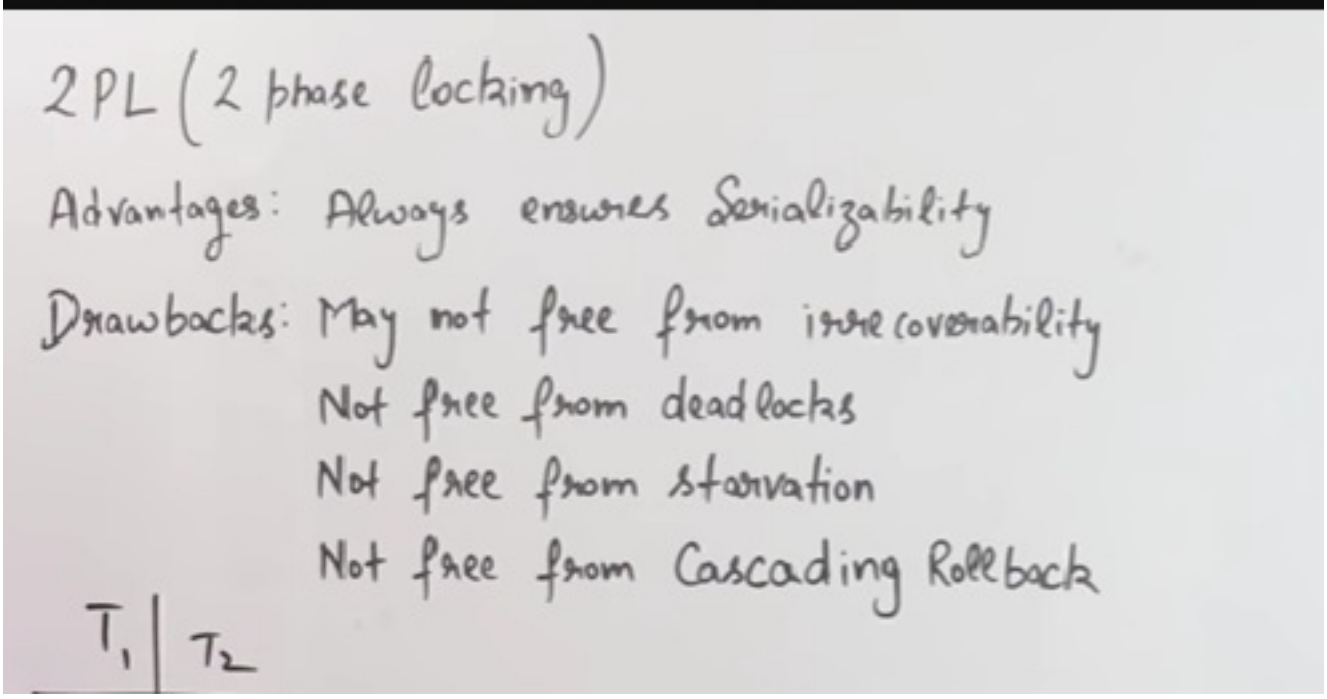
Lock point -> pehla unlock hota hai jaha pe

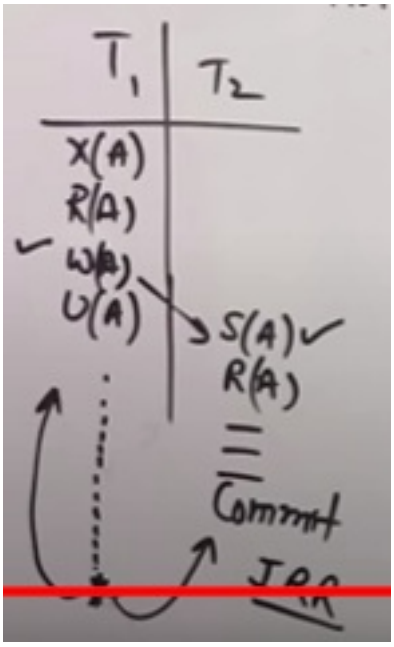


So t1 -> t2

Kyunki t1 ka lock point pehle aaraha.

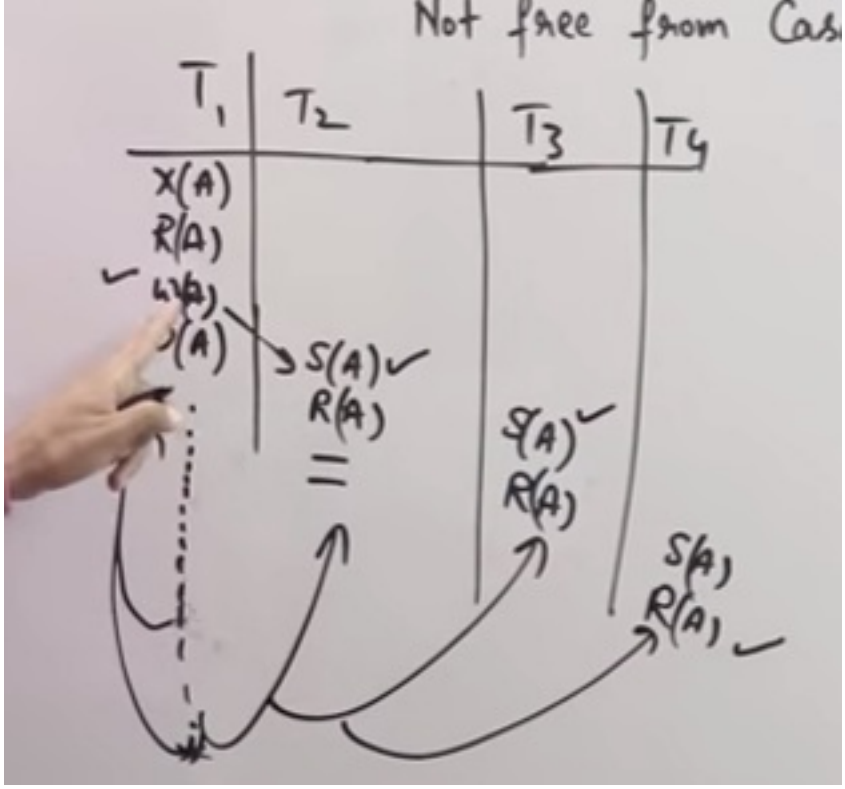
DRAWBACKS IN 2PL



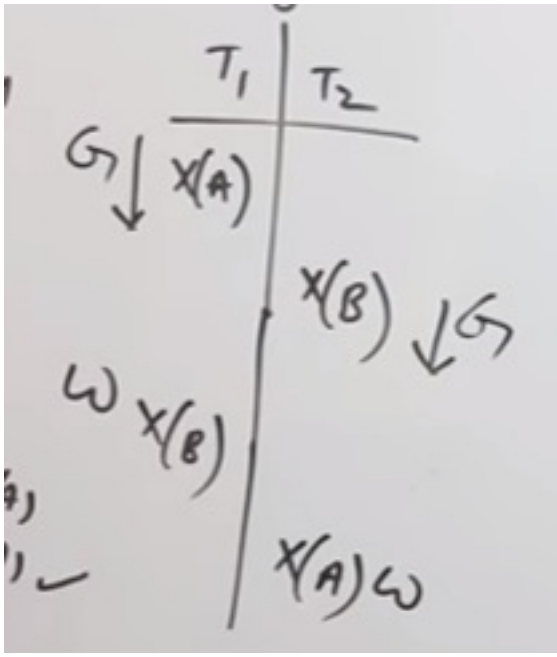


-> IRRECOVERABILITY

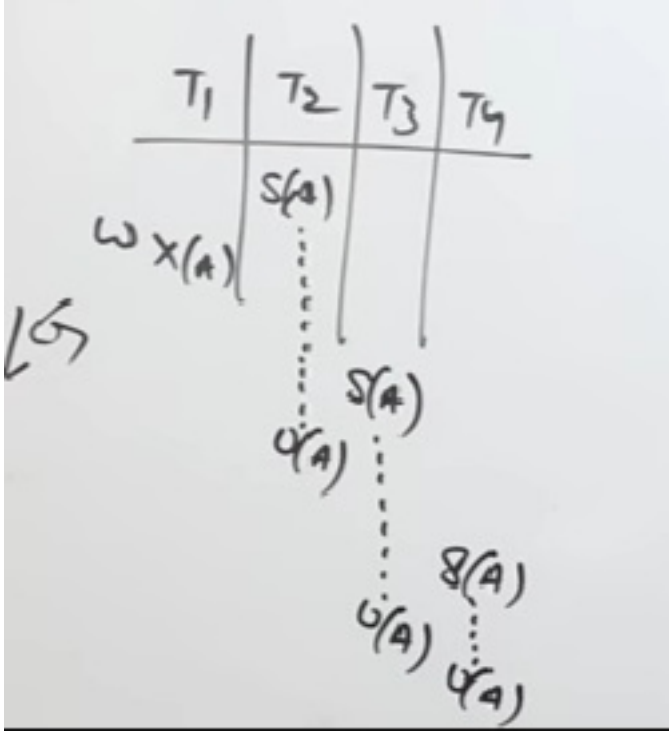
CASCADING ROLLBACK :



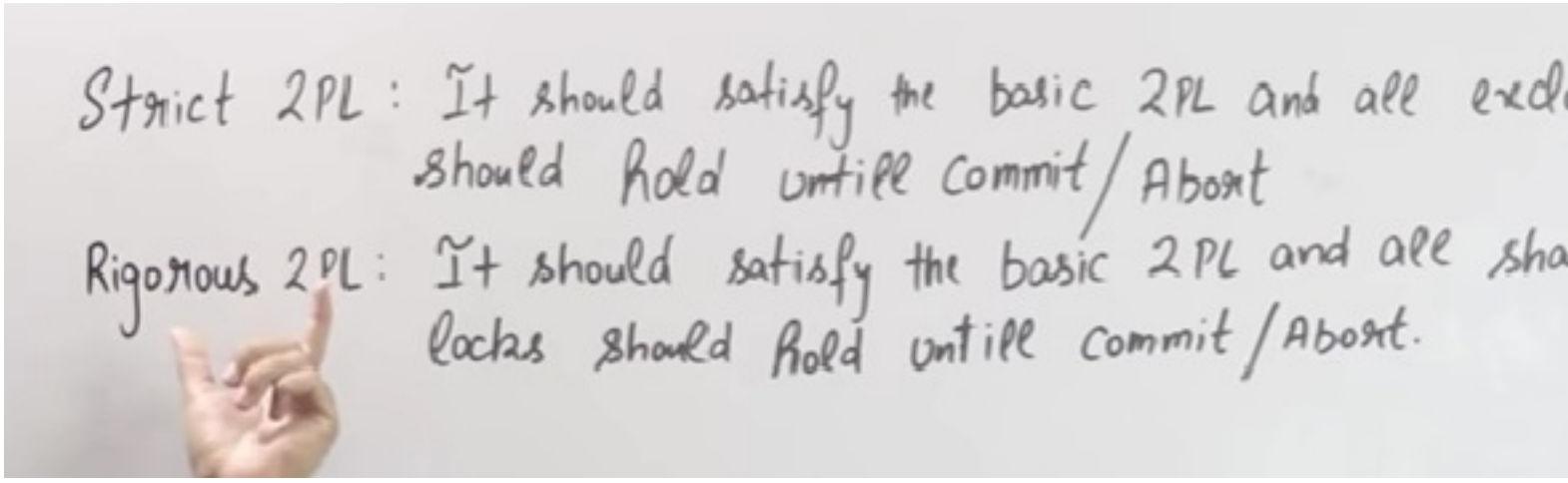
DEADLOCK :

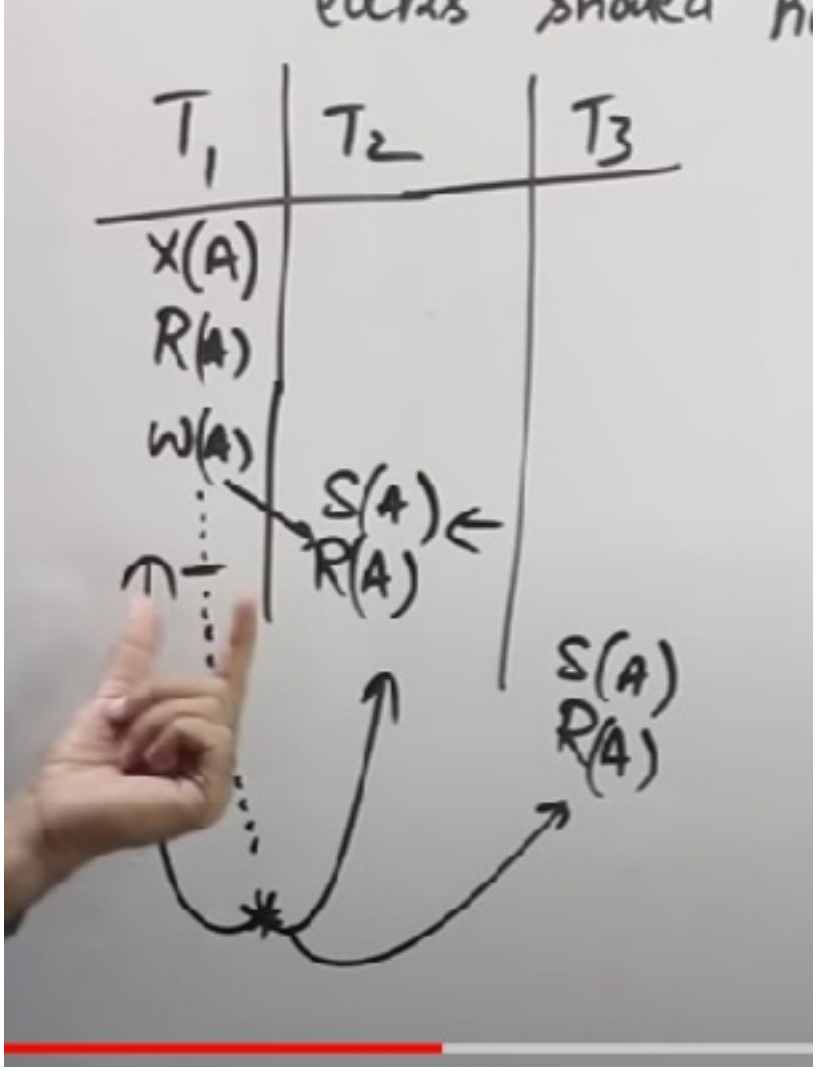


STARVATION:



EXTENSIONS OF 2PL :



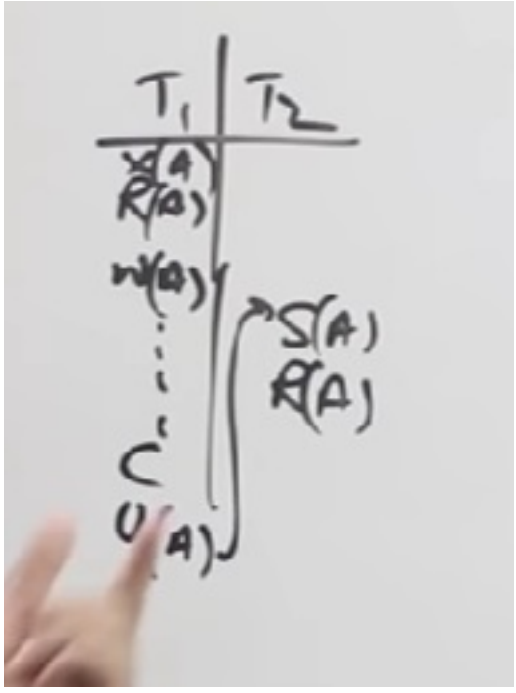


Yaha pe abhi humne strict 2PL nhi kra hai use

So cascading rollback hogya sab

But

Cascading rollback issue is resolved if we use strict 2PL

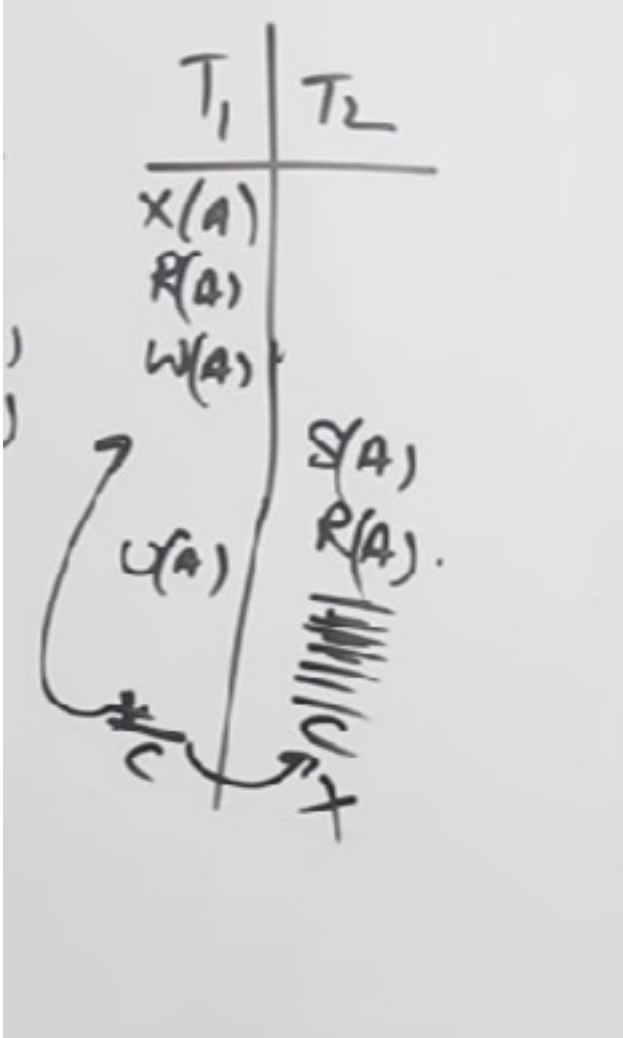


Here T1 will first get committed then only U(a) hoga and after that T2 mei R(a) will

be read from database only.

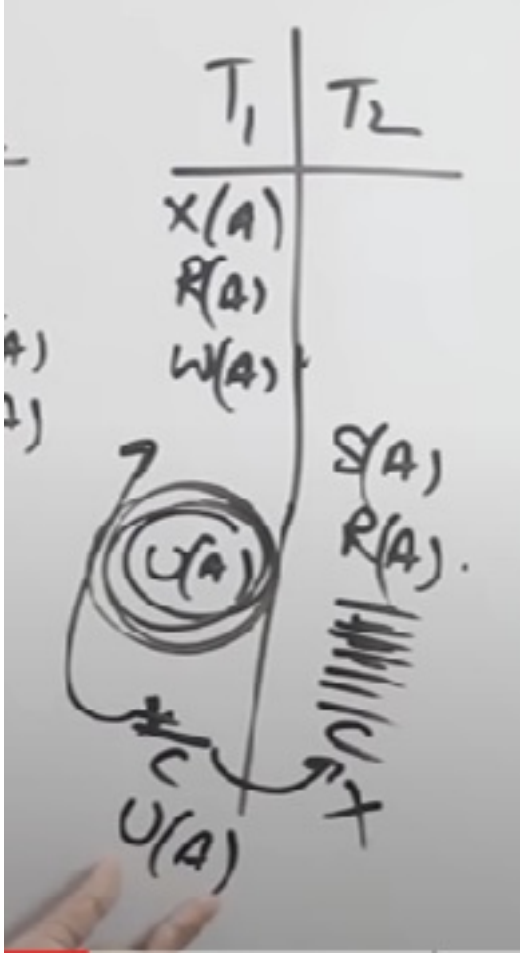
STRICT 2PL ALWAYS PRODUCES CASCADELESS

Lets talk about irrecoverability now :



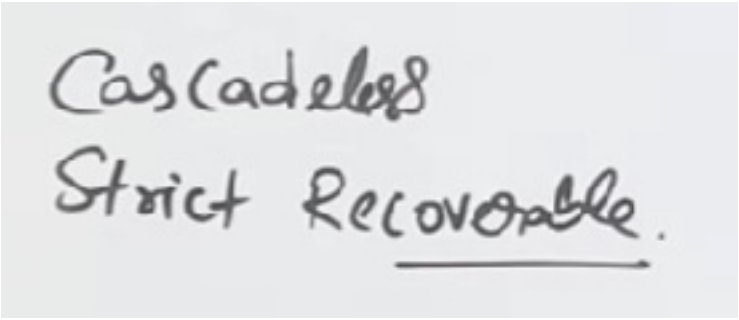
Yaha pe T2 commit hogya to rollback to ho ni skta , so T1 k roll back k kaaran irrecoverable hogya saara system

On applying strict 2 PL :



Unlock hua nhi to t2 ka S(A) grant nhi hua aur ussey pehle hi system abort krgya hehehe so recoverable bngya sab !

So strict 2pl se cascadeless k saath saath recoverable bhi bngya

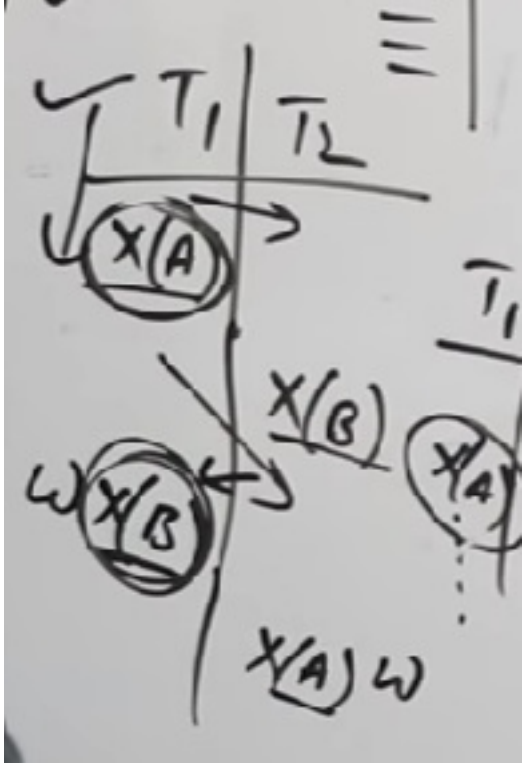


Rigorous 2 pl is stricter

Deadlock aur starvation nhi solve krte but ye

There exists something called CONSERVATIVE 2 PL as well

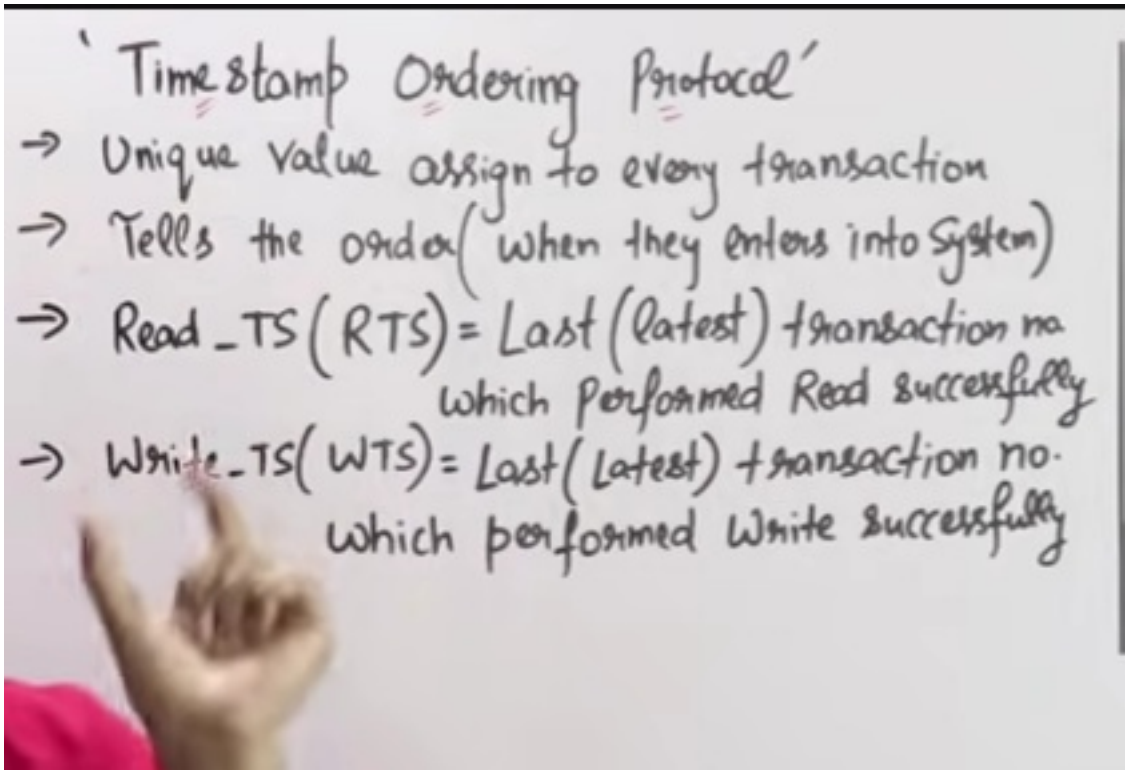
It says ki pehle transition ko saare locks grant krdo so it wont have to wait for other transitions and therefore deadlock problem solve hojayega

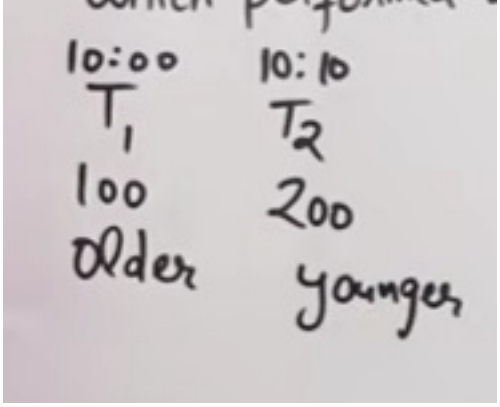


TIMESTAMP ORDERING PROTOCOL

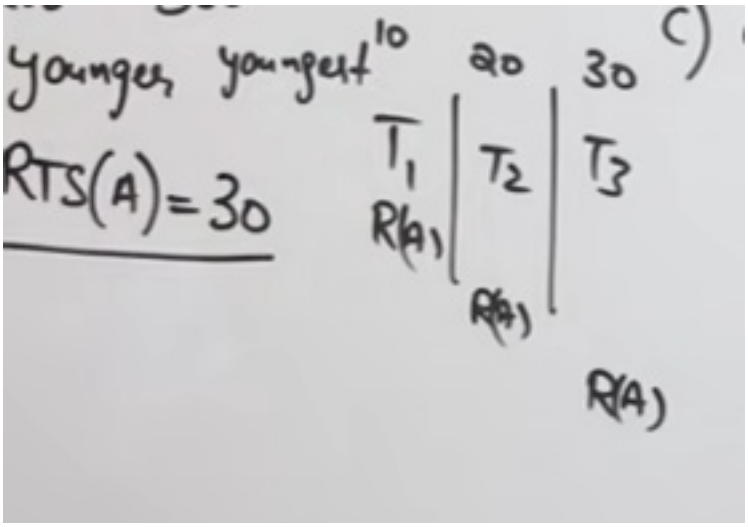
YE BHI EK CONCURRENCY CONTROL PROTOCOL HAI

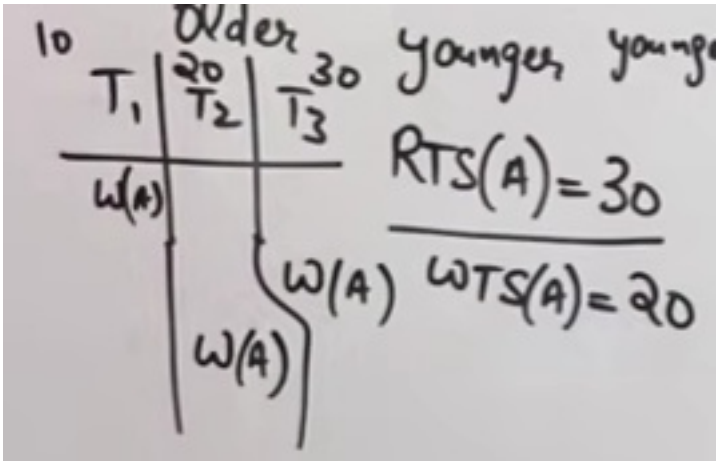
(isme don’t look at conflicts sab old young ka khel hai)

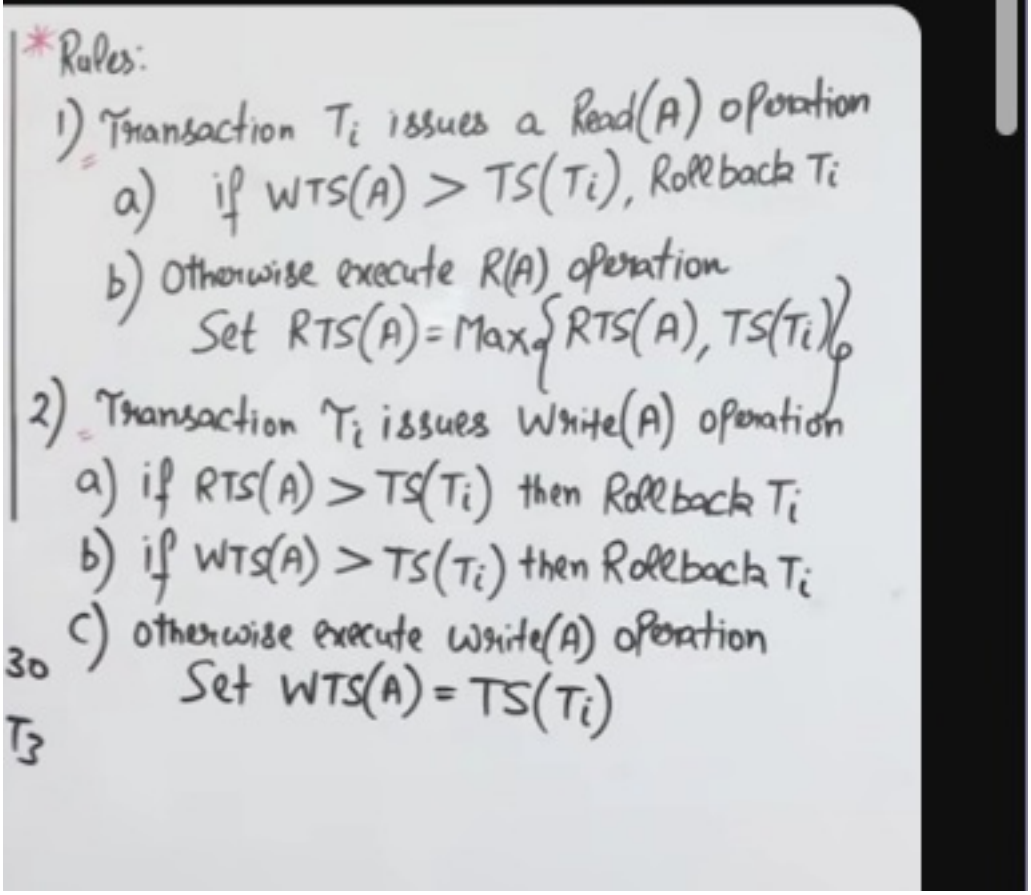




Dus baje agar ek transaction aaya usey maine timestamp dedia 100 ka aur later jo aaya usey 200 ka issey dono ki age maalum pd gyi

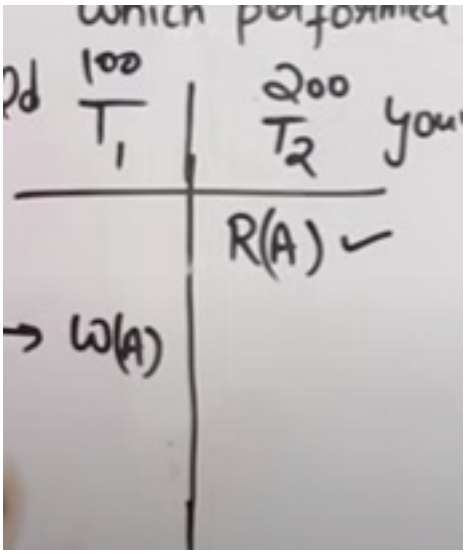






So pehle deal with the older one

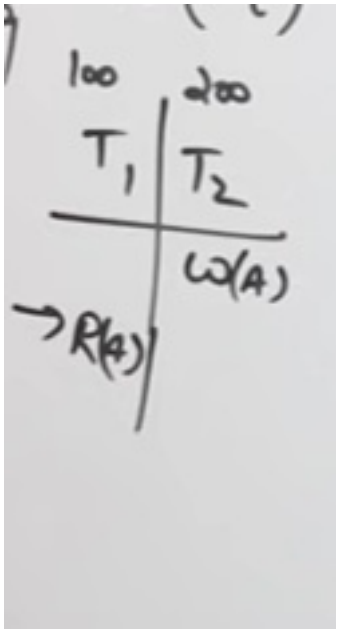
Neglect krdo conflict ko bs older vaale ko pehle execute krna hai no matter what

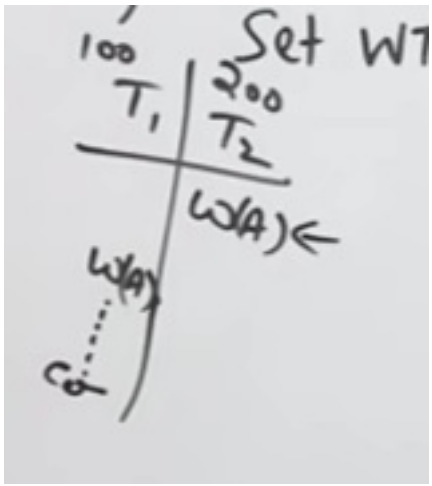


Rules state pehle aai hu transaction baad mei operation nhi krni chahiye !

So idhr T1 ko roll back krna pdega

Similarly





AWAYS TREAT OLDER ONE FIRST.

BASICALLY OLDER KUCH BHI YOUNGER SE BAADD MEI KRE TO USKE PROCESS KO ROLLBACK KRDO

NUMERICAL

T_1 (100)	T_2 (200)	T_3 (300)
R(A)		
	R(B)	
W(C)		
		R(B)
R(C)		
	W(B)	
		W(A)

	A	B	C
RTS	0	0	0
WTS	0	0	0

$0 > 100$

Otherwise execute R(A) operation

Set $RTS(A) = Max\{RTS(A), TS(T_i)\}$
0, 100

	A	B	C
RTS	~~0~~ 100	0	0
WTS	0	0	0

$0 > 200$

Set $RTS(A) = Max\{RTS(A), TS(T_i)\}$

	A	B	C
RTS	~~Ø~~ 100	~~Ø~~ 200	0
WTS	0	0	0

SIMILARLY DO FOR ALL ITHER OPERATIONS

'Timestamp Ordering Protocol'

(100) T_1 oldest	(200) T_2	T_3 (300) youngest
R(A)		
	R(B)	
→ W(C)		
		R(B) ←
→ R(C)		
	W(B)	
		W(A)

0 > 100
0 > 200
0 > 100
0 > 100
0 > 300
100 > 100
[300 > 200]

	A	B	C
RTS	~~Ø~~ 100	~~Ø~~ ~~200~~ 300	~~Ø~~ 100
WTS	0	0	~~Ø~~ 100

*Rules:

1) Transaction T_i
   a) if WTS(A
   b) Otherwise exec
   Set RTS(A
2) Transaction T_i
   a) if RTS(A) > T
   b) if WTS(A) > T
   c) otherwise execu
   Set WTS(A

T2 WILL NOW RESTART BAAD MEI SABSE

'Timestamp Ordering Protocol'

(100) oldest T_1	(200) T_2	T_3 (300) youngest
R(A)		
	R(B)	
→ W(C)		(1000) R(B) ←
→ R(C)	2000 W(B) ~~=~~ Rollback	
		W(A)

0 > 100
0 > 200
0 > 100
0 > 100
0 > 300
100 > 100
[300 > 200]

100 > 300
0 > 3

	A	B	C
	100	~~200~~ 300	
RTS	Ø	Ø	~~Ø~~ 100
WTS	0	0	~~Ø~~ 100

*Rules:

1) Tran
a)
b)

2) Tran
a) if
b) if
c) ot

3

300