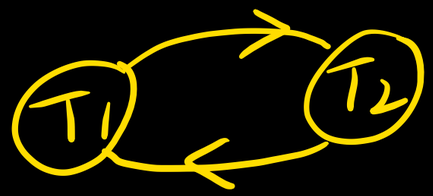
S: $r_2(A)\ r_2(B)\ w_2(A)\ w_1(A)\ r_3(A)\ w_1(B)\ w_2(B)\ w_3(B)$

Conflict serializable:



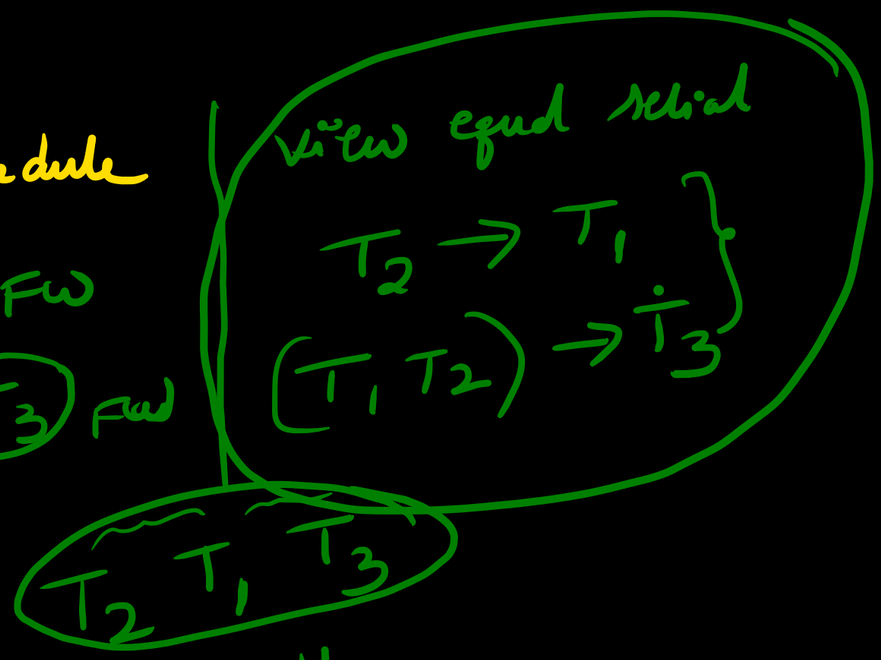
not CS.

View serializable:-

Given Schedule

Final writes: A: $T_2\ (T_1)$ FW

B: $T_1\ T_2\ (T_3)$ FW



But we have to test other conditions UR IR

S: $r_2(A)$ $r_2(B)$ $w_2(A)$ $w_1(A)$ $r_3(A)$ $w_1(B)$ $w_2(B)$ $w_3(B)$

Initial reads:

	initial read	writes
A	T_2	T_2 T_1
B	T_2	T_1 T_2 T_3

Updated reads:

A

$w_1(A) \rightarrow r_3(A)$

other writes in

T_2

$w_1(A)$ $w_2(A)$ $r_3(A)$

view equal serial

$T_2 \rightarrow T_1$

$T_2 \rightarrow T_1, T_3$

T_2

$T_1 \rightarrow T_3$

T_2 T_1 T_3

T_1 T_3 T_2

write down all possible serial schedules and then apply the rules we got: (if you can check the options, which one is obeying the rules.

$T_1\ T_2\ T_3$

$T_1\ T_3\ T_2$

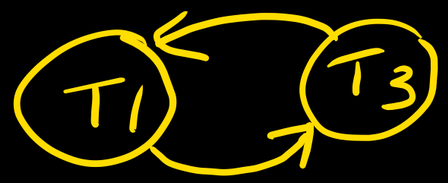
$T_2\ T_1\ T_3$ ✓ View serial equal schedule.

$T_2\ T_3\ T_1$

$T_3\ T_1\ T_2$

$T_3\ T_2\ T_1$

S_1 : $r_1(A)$ $r_2(A)$ $r_3(A)$ $w_1(A)$ $w_2(A)$ $w_3(A)$



not CS.

T_1 T_2 T_3

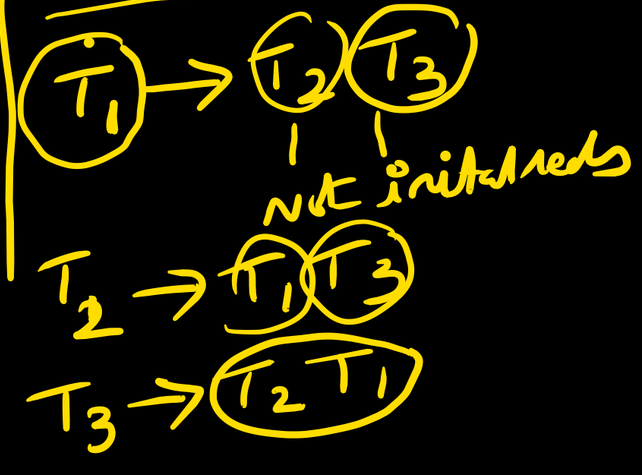
$r_1(A)$ $w_1(A)$ $r_2(A)$ $w_2(A)$

Initial read

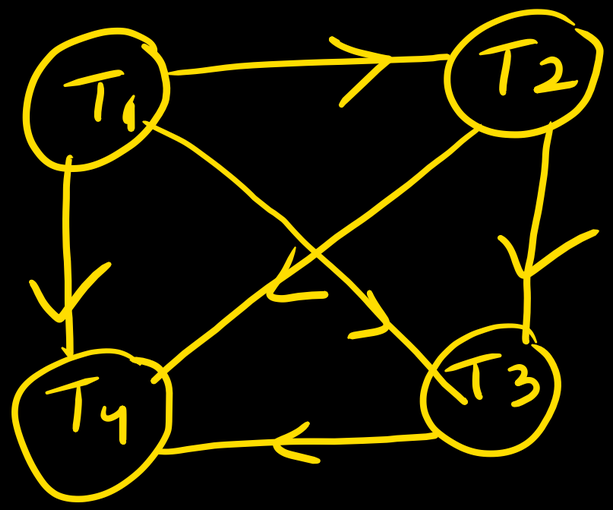
data item	IR	write
A	T_1 T_2 T_3	T_1 T_2 T_3

Not view serial.

view serial equal



S: $r_1(A)\ r_2(A)\ r_3(A)\ r_4(A)\ w_1(B)\ w_2(B)\ w_3(B)\ w_4(B)\ w_1(C)\ w_3(C)\ r_4(C)$



acyclic. So Conflict serializable

$T_1\ T_2\ T_3\ T_4$

is the conflict equivalent serial schedule

S: $r_1(A)\ r_2(A)\ r_3(A)\ r_4(A)\ w_1(B)\ w_2(B)\ w_3(B)\ w_4(B)\ w_1(C)\ w_3(C)\ r_4(C)$

Given schedule

View equal serial

$T_1 T_2 T_3 T_4$
$T_1 T_3 T_2 T_4$

IR A: IR write

$T_1 T_2 T_3 T_4$

UR: $w_3(C) \xrightarrow{T_3} R_4(C) \xrightarrow{T_4}$
another write on C
in $w_1(C)$ ✓

FW B: $T_1 T_2 T_3 (T_4)$ FW
C: $T_1 (T_3)$ FW

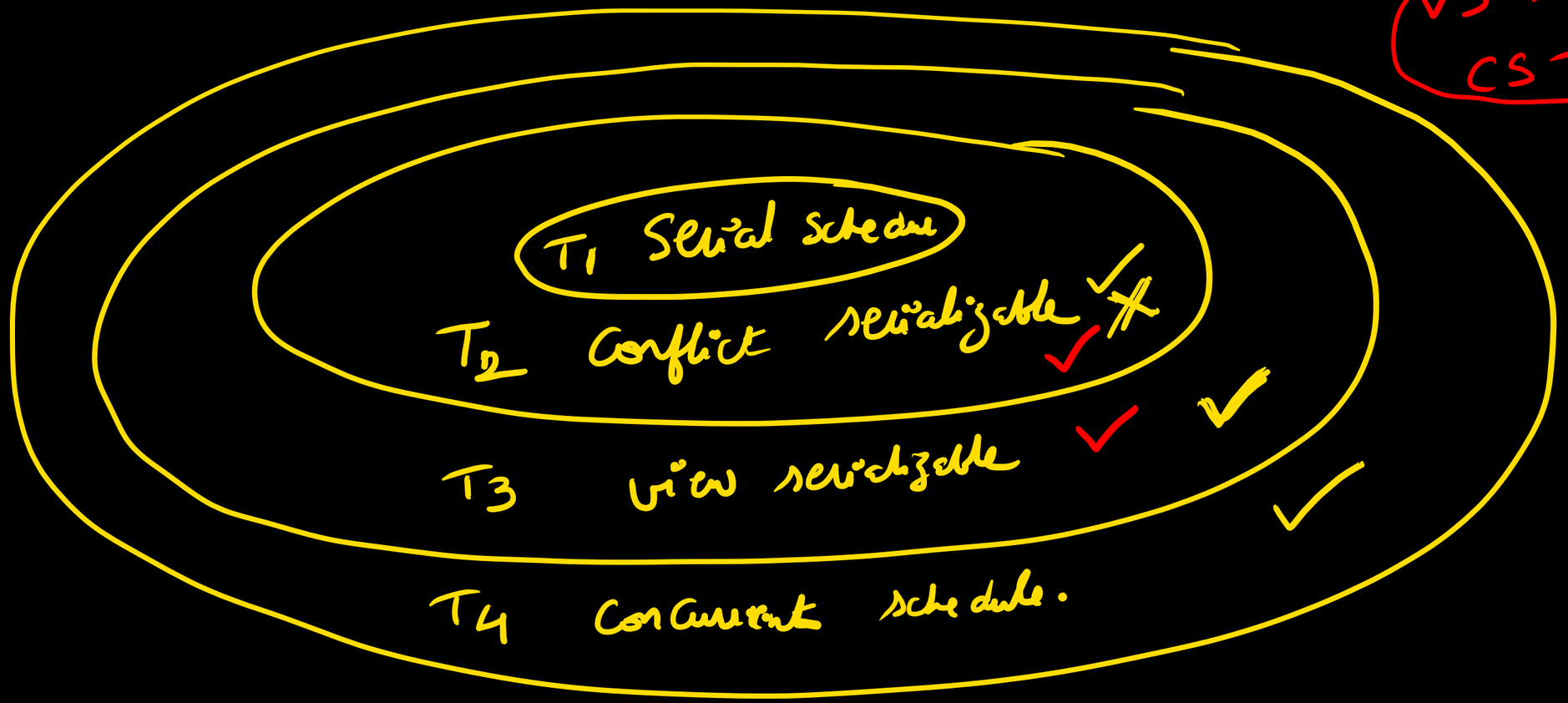
NO restrictions

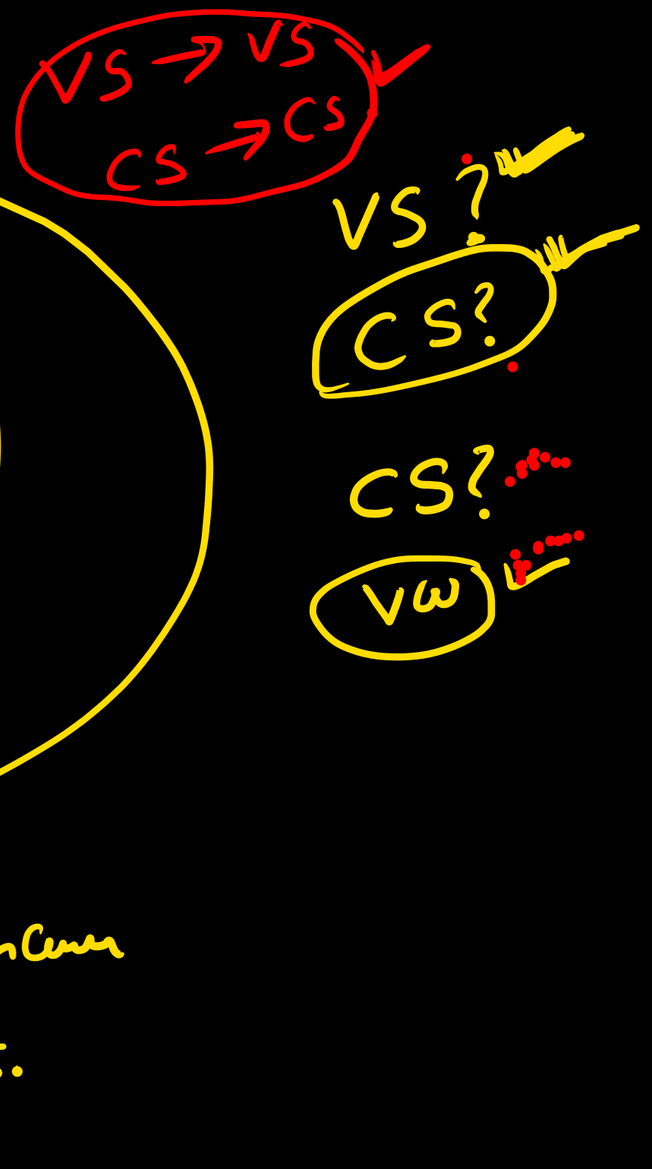
$T_1 \times$
$T_3 \rightarrow T_4$ ✓

$(T_1 T_2 T_3) \rightarrow T_4$ ✓
$T_1 \rightarrow T_3$ ✓

$T_1\ T_3\ T_4$
T_2 T_2 T_2
T_1 T_3 T_4

$T_2 T_1 T_3 T_4$





T1 in all

T2 in Conflict, view, Concurr

T3 view, Concurr

T4 Concurrent.

Testing conflict ~~Serializable~~ serializable is a polynomial time problem $O(n^2)$

Testing view serializable is NPC which means (exponential time)

problem.

view serializable

~~decid~~ difficult

Gate simple questions will be given

NO question in gate

Classification of Schedules based on Recoverability:

Till now we classified questions based on Serializability

- Conflict
- View

Concurrent execution of transactions may lead to

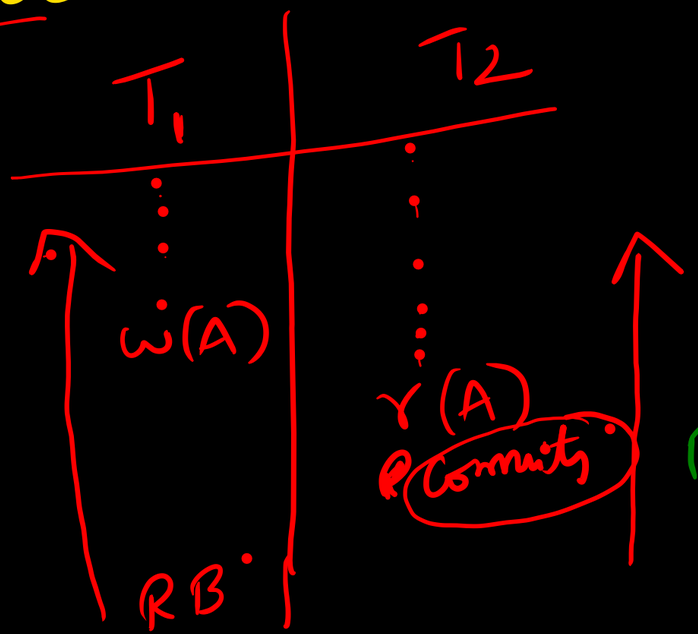
(i) irrecoverable problem.

(ii) Cascading roll back problem.

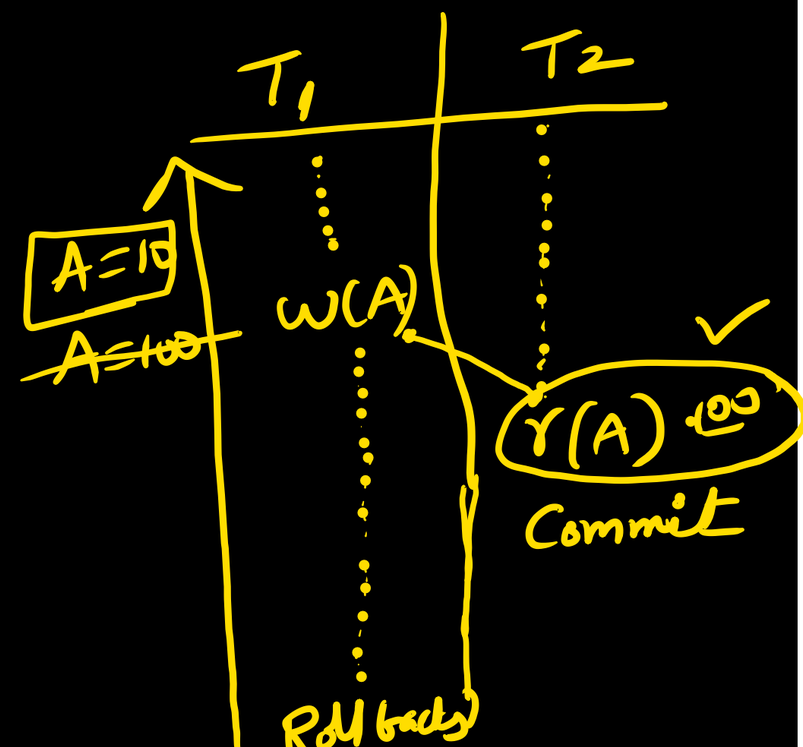
(iii) lost update problem.

Irrecoverable problem :-

When it is required to roll back a committed transaction, then it is called irrecoverable schedule.



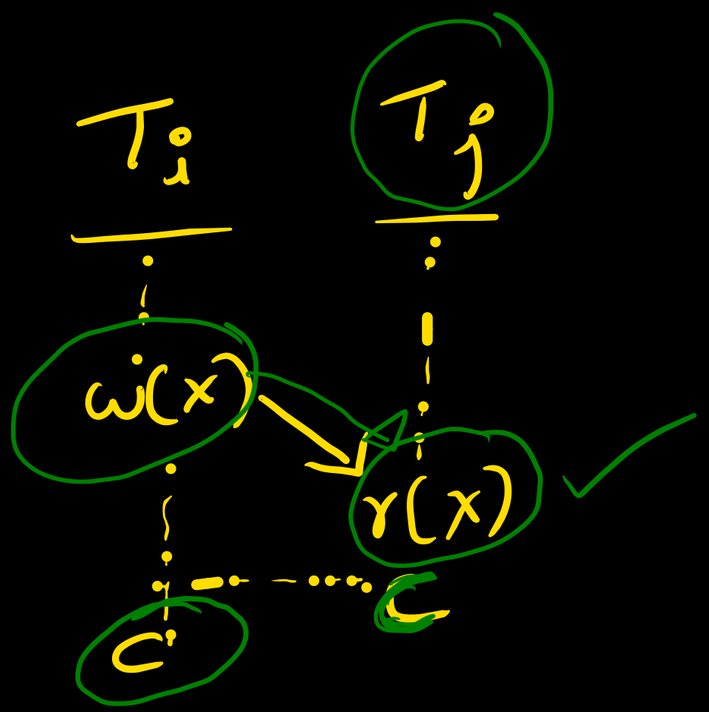
We cannot roll back a committed transaction



Roll back cannot happen after commit

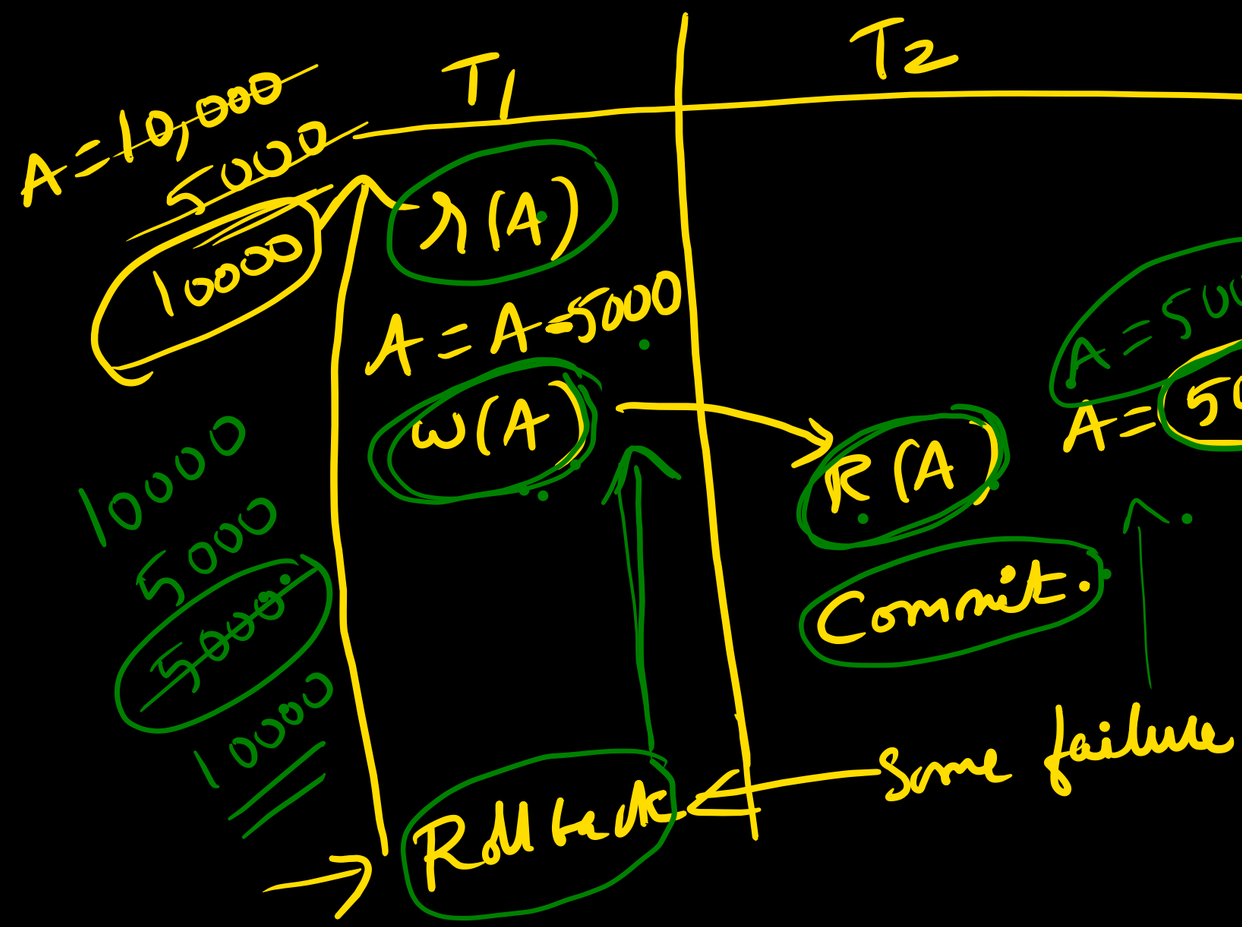
Formal definition:

A Schedule (S) is ~~[illegible]~~ Irrecoverable iff transaction T_j reads data item 'X' from T_i and commit of T_j happens after commit/Roll back of T_i.



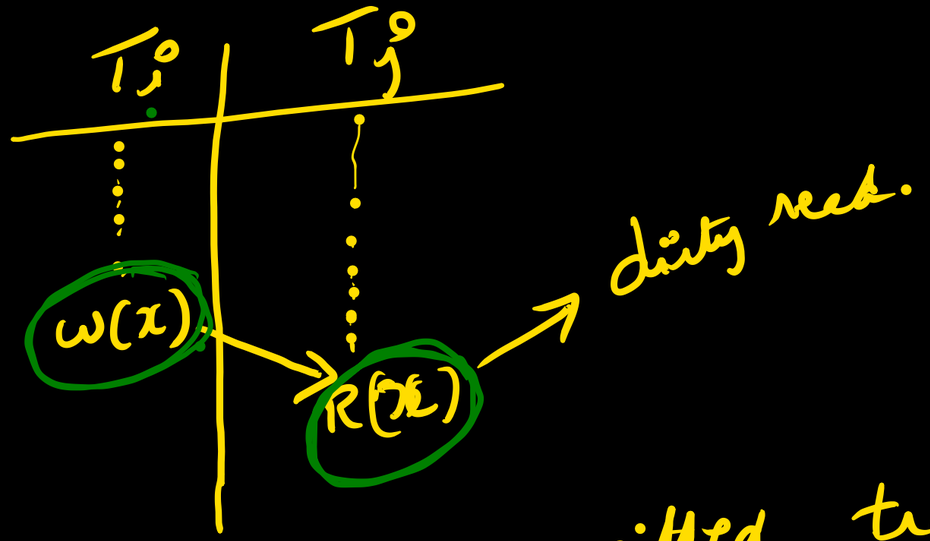
Ex: T_1 : withdraw 5000 from A $[r_1(A)\ A=A-5000\ w_1(A)\ C_1]$

T_2 : check bal of A $[r_2(A)\ C_2]$

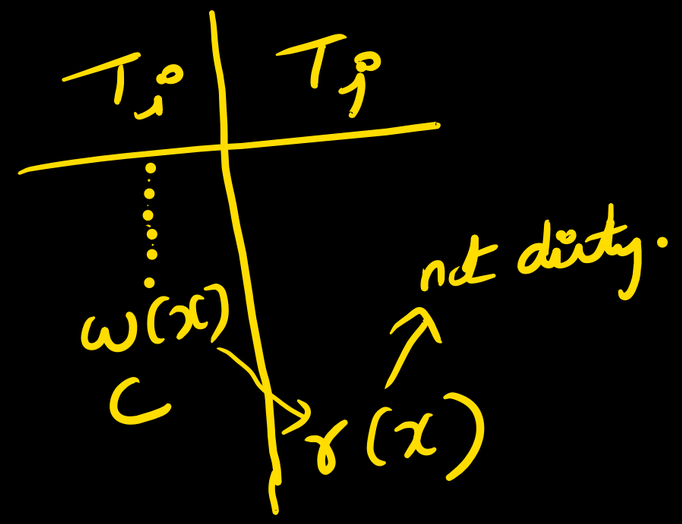


This the problem if schedule is irrecoverable.

uncommitted read & dirty read:-



Reading from uncommitted transaction is a dirty read.



IR

Not dirty

T_1	T_2	T_3
$r_1(A)$	$w_2(B)$	
	C_2	
		$R_3(B)$ → not dirty
$W(A)$		
	$R(A)$ → dirty ✓	

Dirty & uncommitted read :-

Transaction T_j reads 'x' which is updated by uncommitted transaction T_i, such a read is called uncommitted & dirty read

T_i	T_j
$W(x)$ →	$R(x)$

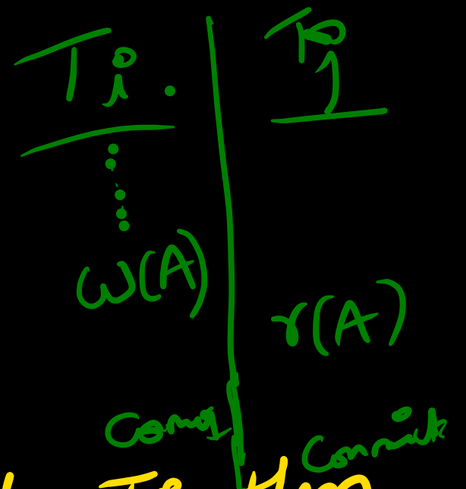
Recoverable schedule: ✓

Schedule 'S' is recoverable iff

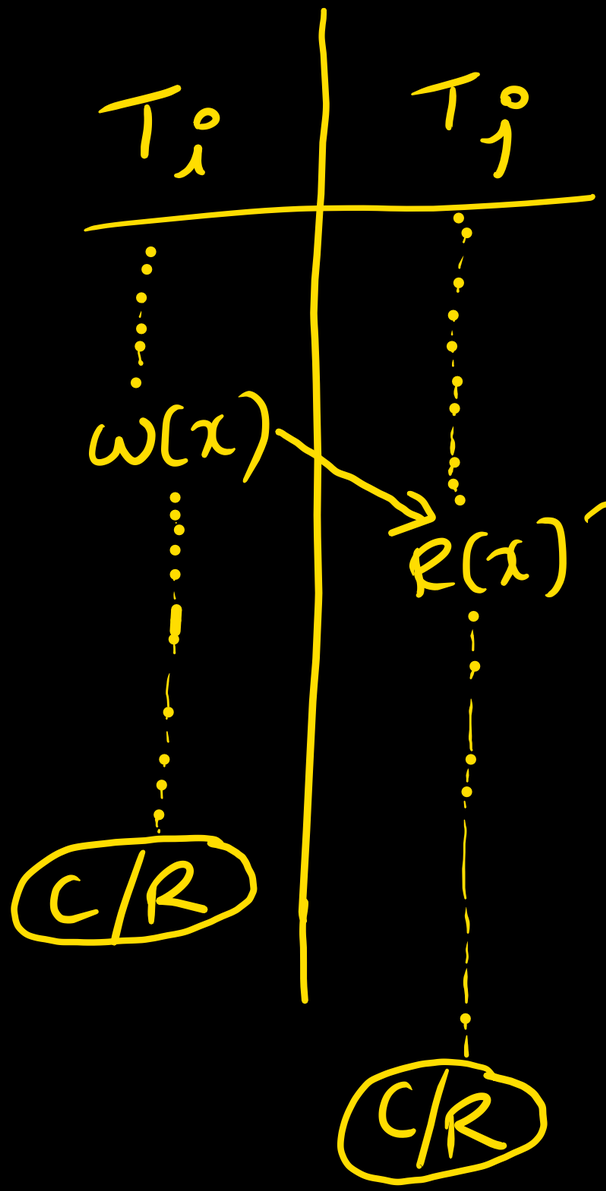
① ✓ no dirty reads in schedule S.

(OR)

② If T_j reads x which is updated by T_i then Commit of T_j must happen only after Commit ✗ / Rollback of T_i



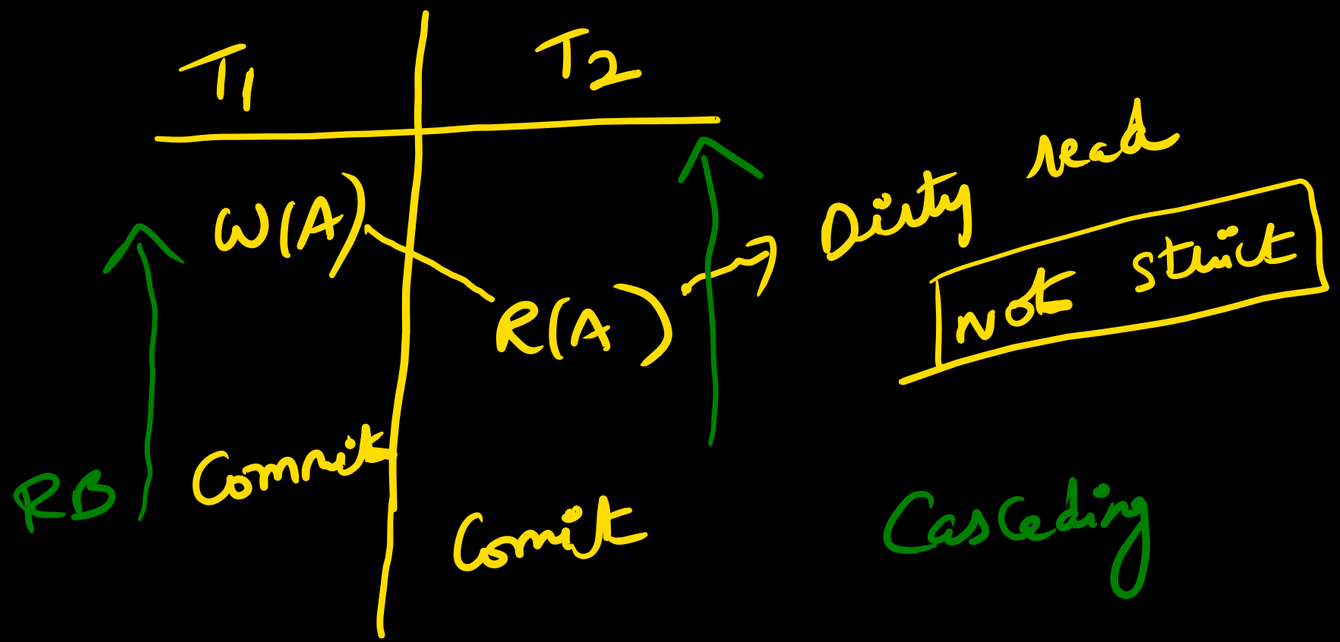
(v)



→ dirty read.

Even though this is dirty read, the schedule is recoverable.

Ex:



Recoverable

S:

T_1	T_2
R(A)	
	R(A)
W(A)	
	W(A)
C_1	
	C_2

Not dirty read

R(A) ⇝ Dirty read ✗

⇝ may be lost update.

Cascadeless

not stuck

NO dirty reads at all. So recoverable

3)

T_1	T_2
R(A)	
W(A)	
	R(A)
	R(B)
R(B)	
W(B)	
C1	
RB	
	C2

not dirty read

dirty

not dirty

dirty read so Cascading rollback.

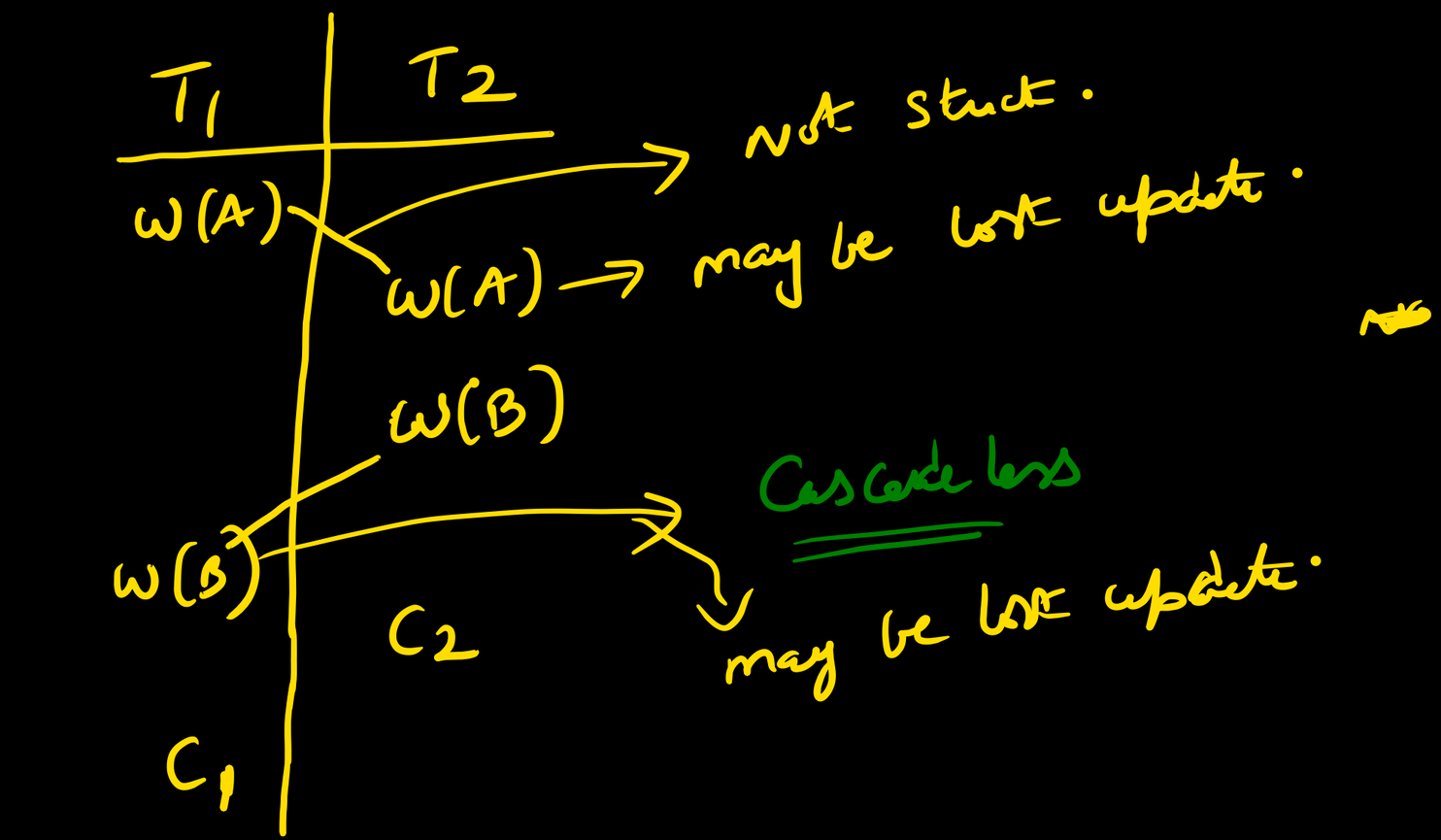
not strict.

not dirty

Cascading

Recoverable

4)



No dirty reads
so recoverable

5)

T_1	T_2	T_3
$R(A)$		
	$R(B)$	
	$W(A)$	
$W(A)$		
C_1		
	$R(C)$	
	$R(A)$	
	$W(B)$	
	C_2	
		$R(B)$
		C_3

$R(A)$ → not dirty

$R(B)$ → not dirty

$W(A)$ → may be lost update problem

$R(C)$ → not dirty

$R(A)$ → not dirty

$R(B)$ → not dirty

Not Strict

Cascadeless

Recoverable

No dirty reads

G)

T_1	T_2	T_3
$IR \rightsquigarrow r_1(A)$		
	$r_2(A)$	$\rightsquigarrow IR$
	$w_2(A)$	$\rightarrow r_3(A) \rightsquigarrow$ dirty $\rightarrow$ so not strict
dirty $\rightsquigarrow$	$w_2(B)$	
$r_1(B)$	C_2	
C_1		
		C_3

Cascading

not recoverable.

7)

IR	T_1	T_2
IR	$r_1(A)$	$r_2(A)$ → IR
		$w_2(A)$
		$w_2(B)$
	$r_1(B)$	

Wait Strict

DR

C_2

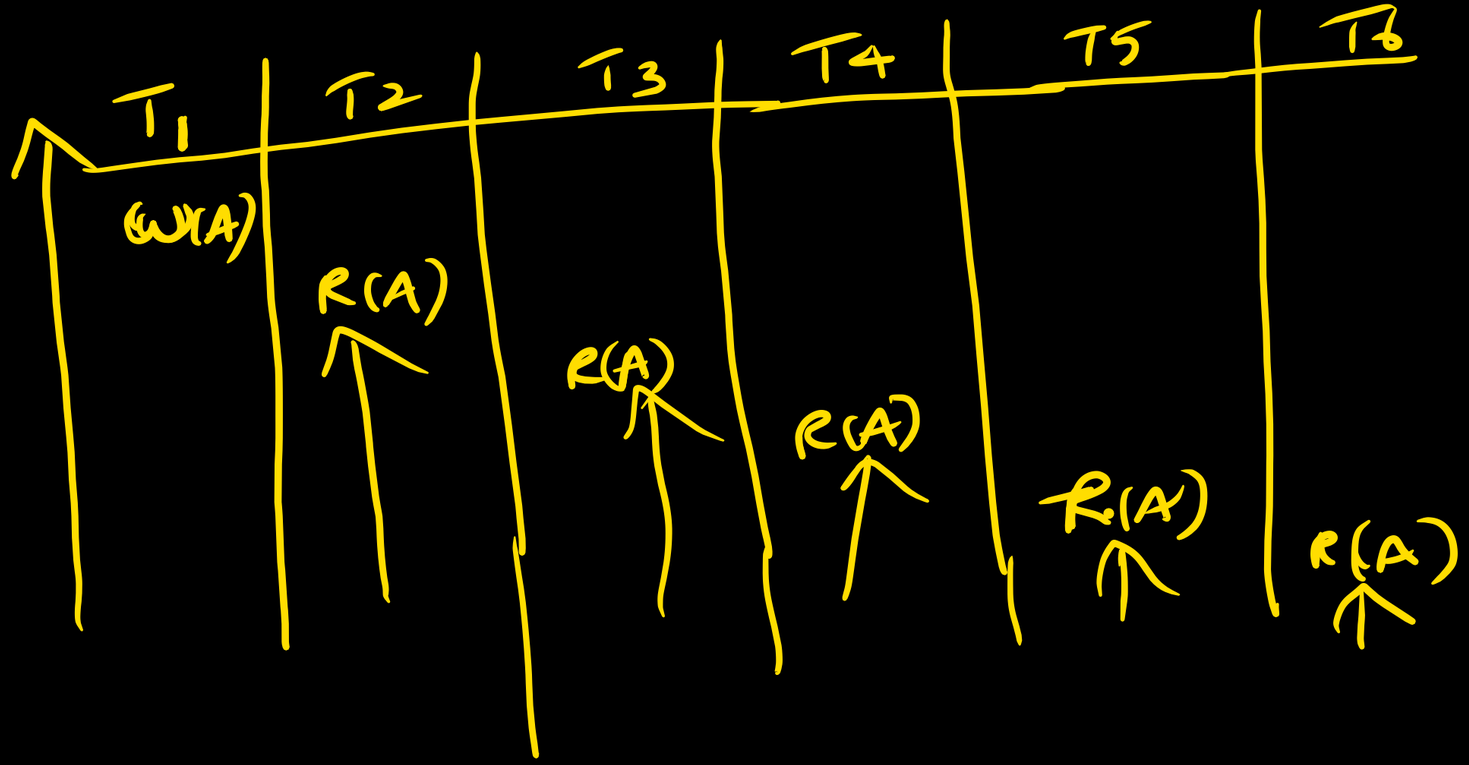
C_1

Cascading.

Recoverable

Cascading Roll back:

Because of failure of some transaction, we are forced to rollback some set of other transactions in schedule 'S'.



T_1	T_2	T_3	T_4	T_5	T_6
W(A)	R(A)				
	W(A)				
	W(B)	W(A)	R(B)	R(B)	
				W(B)	R(B)

$T_1 \rightarrow T_2 \rightarrow T_4$

$T_2 \rightarrow T_5 \rightarrow T_6$

Disadvantages of Cascading RB ✓

⇒ wastage of CPU time

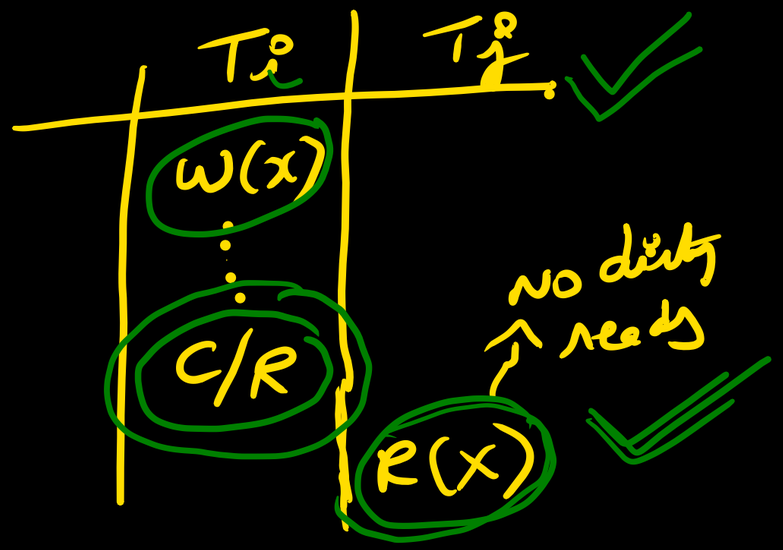
⇒ wastage of I/O access ~~time~~ cost ✓

⇒

⇒ Solution to Cascading roll backs is Cascadeless roll backs

Cascadeless Roll backs :-

Dirty reads not allowed



Lost update problem:-

Transaction T_j^0 writes 'X' which is already written by uncommitted transaction T_i^0

X, initially = ~~100~~ ~~200~~ ~~300~~ 100

Strict recoverable schedule :-

Cascadeless (no dirty reads) and no lost update problem

T_i	T_j
$w_i(x)$	
⋮	
C/R	
	$R(x)$

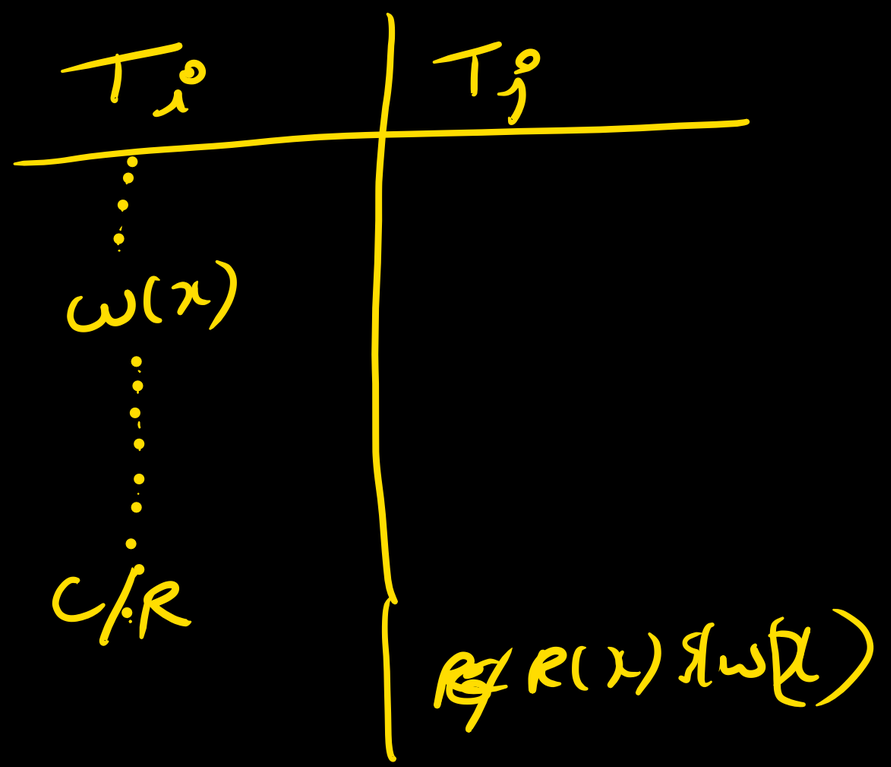
T_i	T_j
$w_i(x)$	
⋮	
C/R	
	$R(x)/w(x)$

T_i	T_j
$w(x)$	
⋮	
C/R	
$w(x)$	

No lost update
This update won't be lost

Combine

If T_i writes x, then $R(x)/W(x)$ of T_j must delayed until C/R of T_i



T_1	T_2
R(A)	
	R(A)
W(A)	
C_1	
	W(A)
	C_2

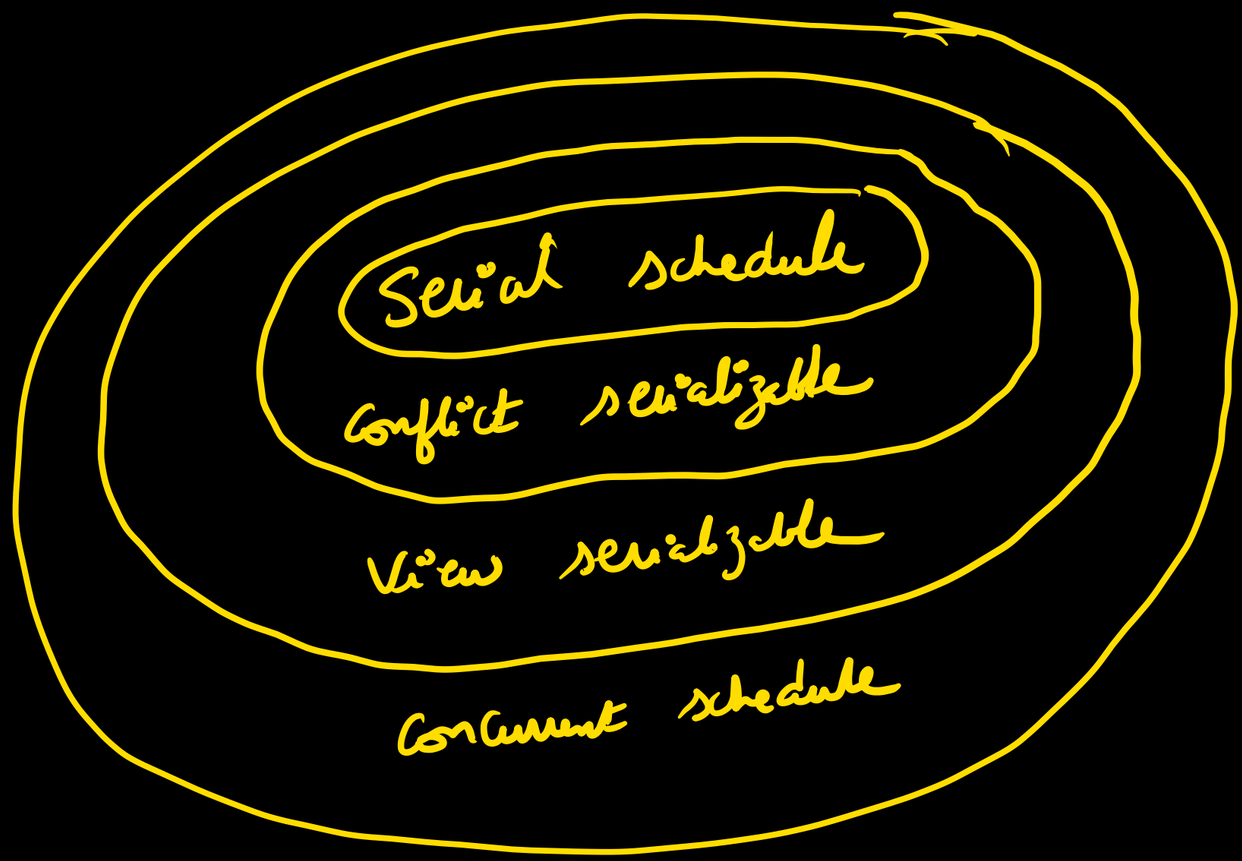
Strict.

Classification based on Recoverability:

If a schedule is Cascadeless roll back schedule then it is strict recoverable schedule Y/N (False.)

NO

Classification based on Serializability



Gate:

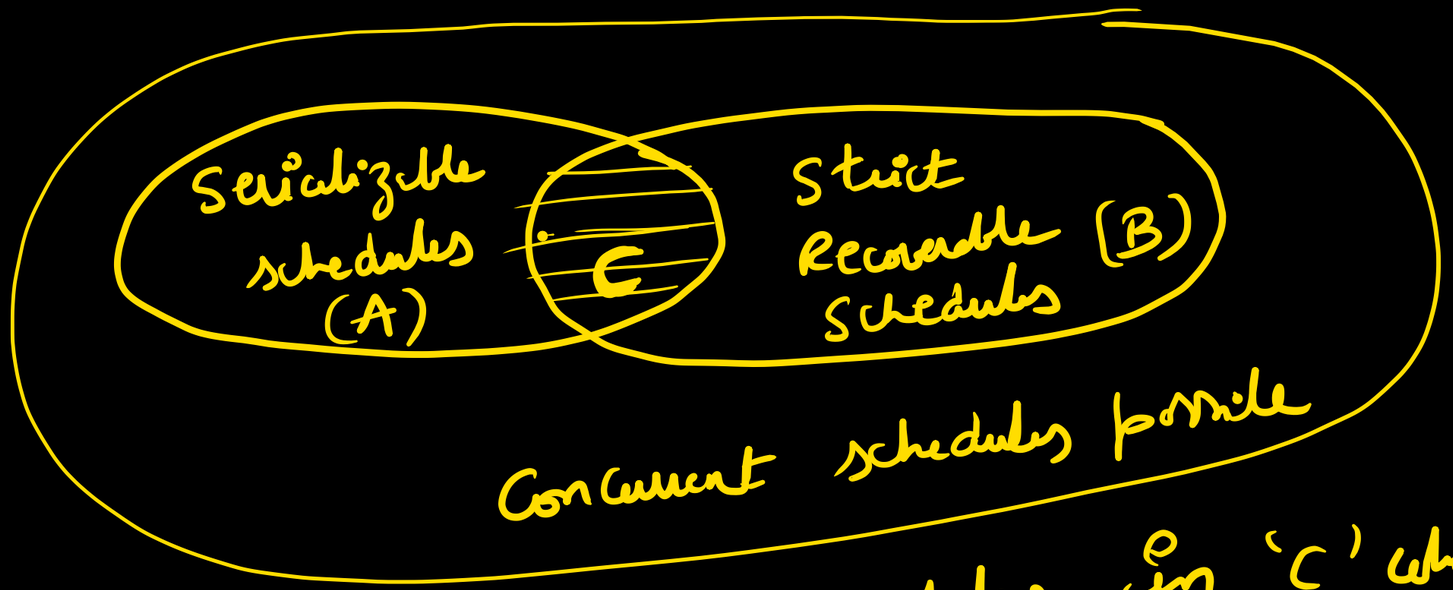
Every conflict serializable is view serializable.

True

Every view serializable is conflict serializable

NO. may or may not be.

Combine Serializability with recoverability:



Our aim is to get schedules in 'C' which are both Serable and recoverable. (Sweet spot)

These schedules in 'C' ensures integrity (correctness)

Every serable schedule is recoverable?

Y/N

NO . may or may not be.

---

there are some Serializable schedules which are recoverable.

T/F True

To get schedules in 'C' we used concurrency control protocols.

Concurrency control protocols can be two types

→ locking protocols

→ Time stamp ordering protocols.

The aim is to get ~~protocol~~ in 'C'

↓

Schedules

Concurrency control protocol:

Goal: Concurrency control protocol should not allow

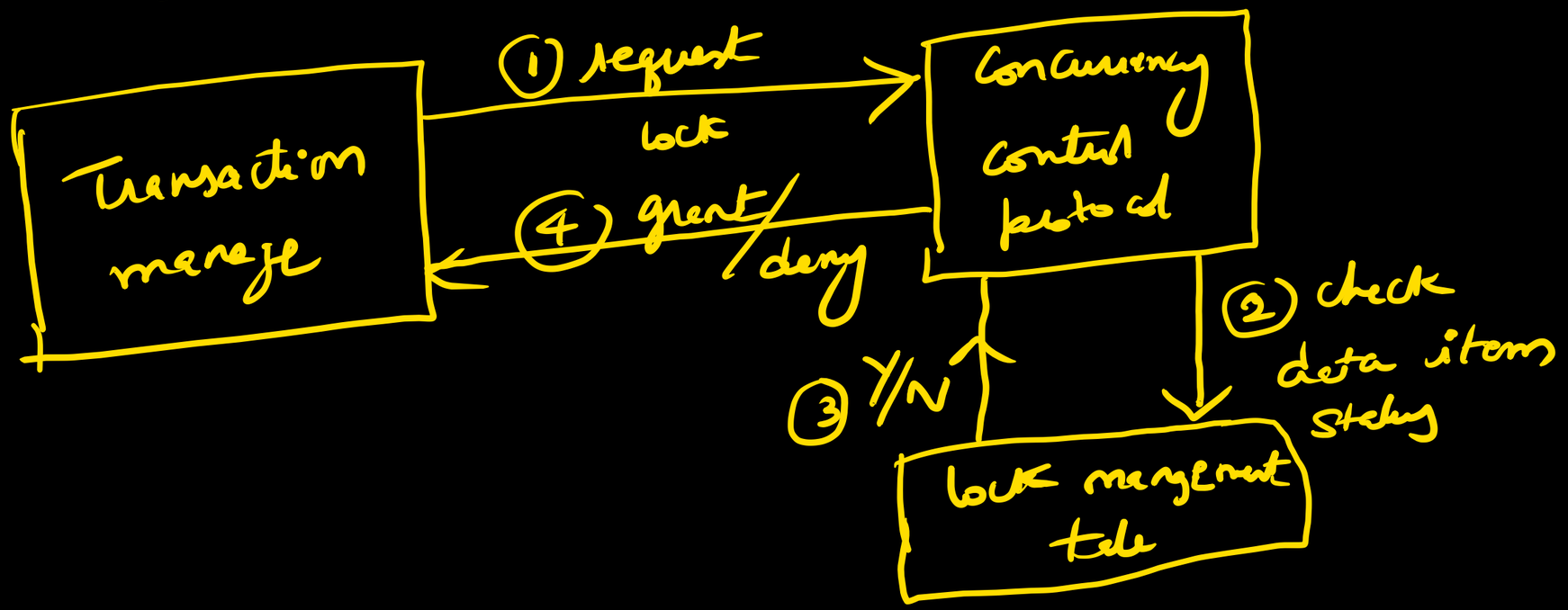
1) non serializable schedules

or

2) non strict recoverable schedules

Locking protocol:

lock: variable which is used to identify status of a data item.



Trans (T):

lock (A) ⇐ granted by Concurrency protocol

R(A)

W(A)

lock (B) ⇐ denied.

wait

lock (B) ⇐ granted

W(B)

unlock (A)

unlock B

Binary locking:

2 operations Binary { lock (data item)
unlock (data item)

Disadvantage of this

T_i	T_j
lock(A)	
R(A)	
	lock A ⟸ denied
	R(A)

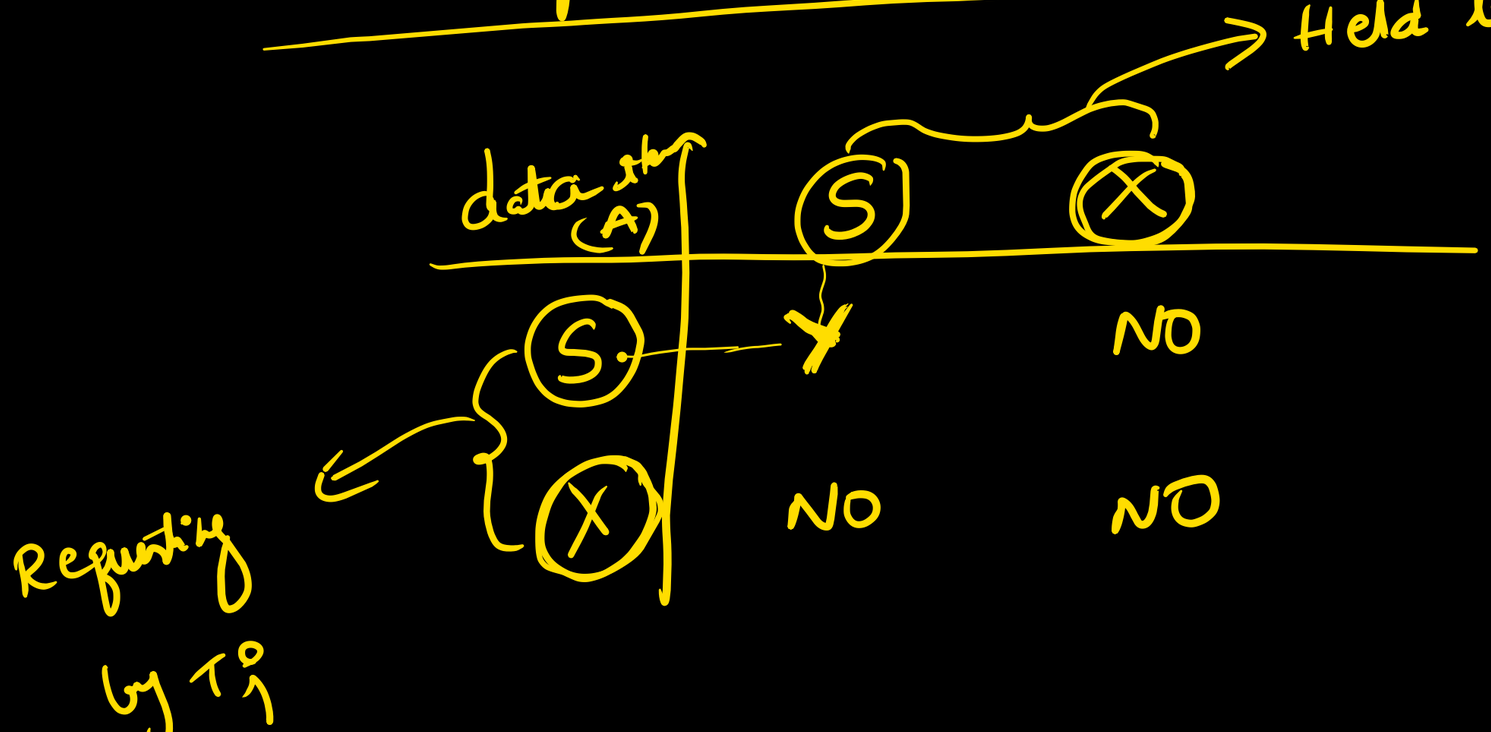
new locking system:

- Shared lock (S(A)) → Read only
- Exclusive lock (X(A)) → Read write lock
- unlock (A)

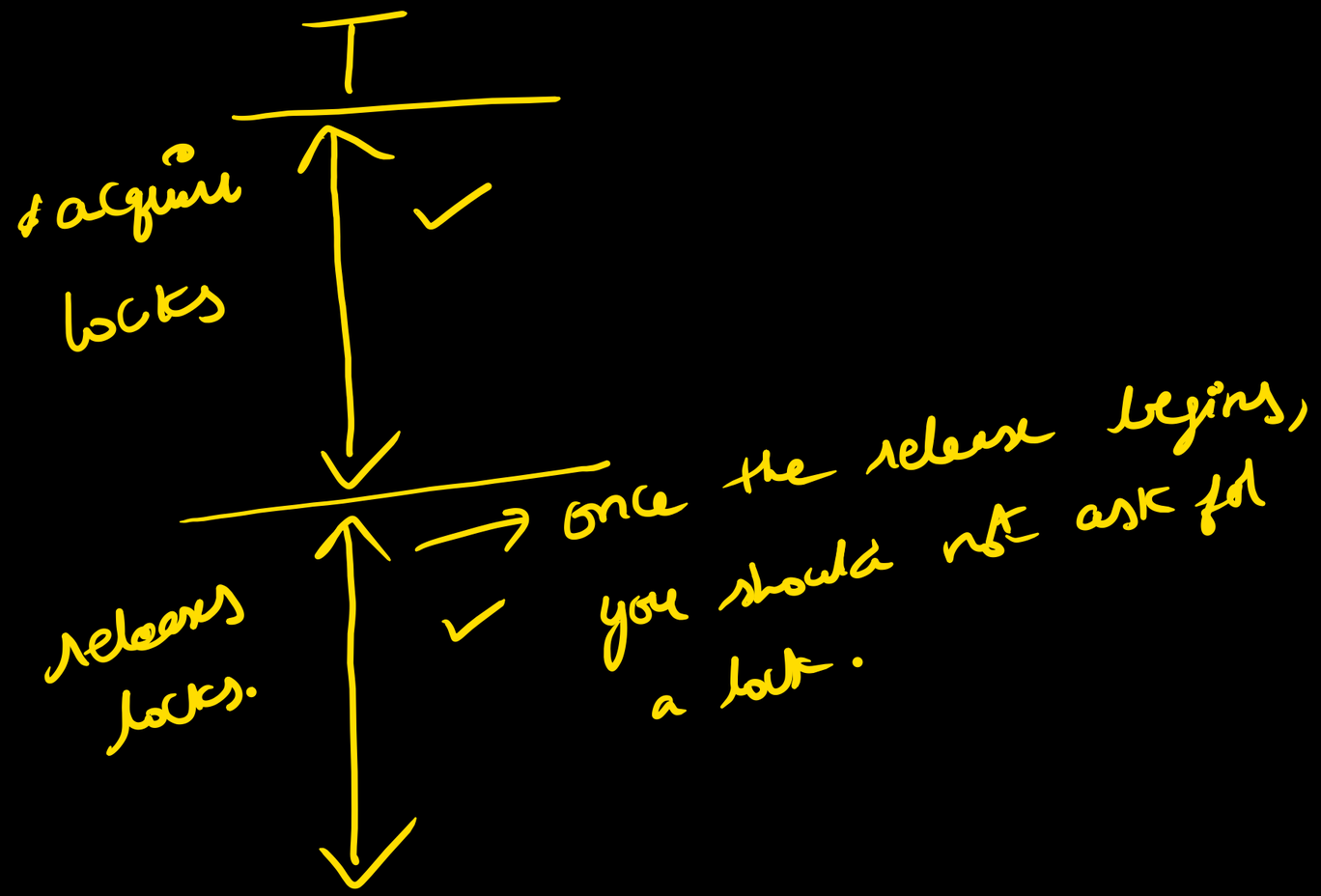
T_i	T_j
S(A)	
R(A)	
	S(A)
	R(A)

Shared lock can be shared by many transactions for read only access.

Lock Compatibility table:-



Two phase locking protocol:-



Locking and unlocking can happen in any order

Trans (T)

S(A)

X(B)

S(C)

X(D)

U(B)

U(D)

U(A)

U(C)

locking phase

growing phase

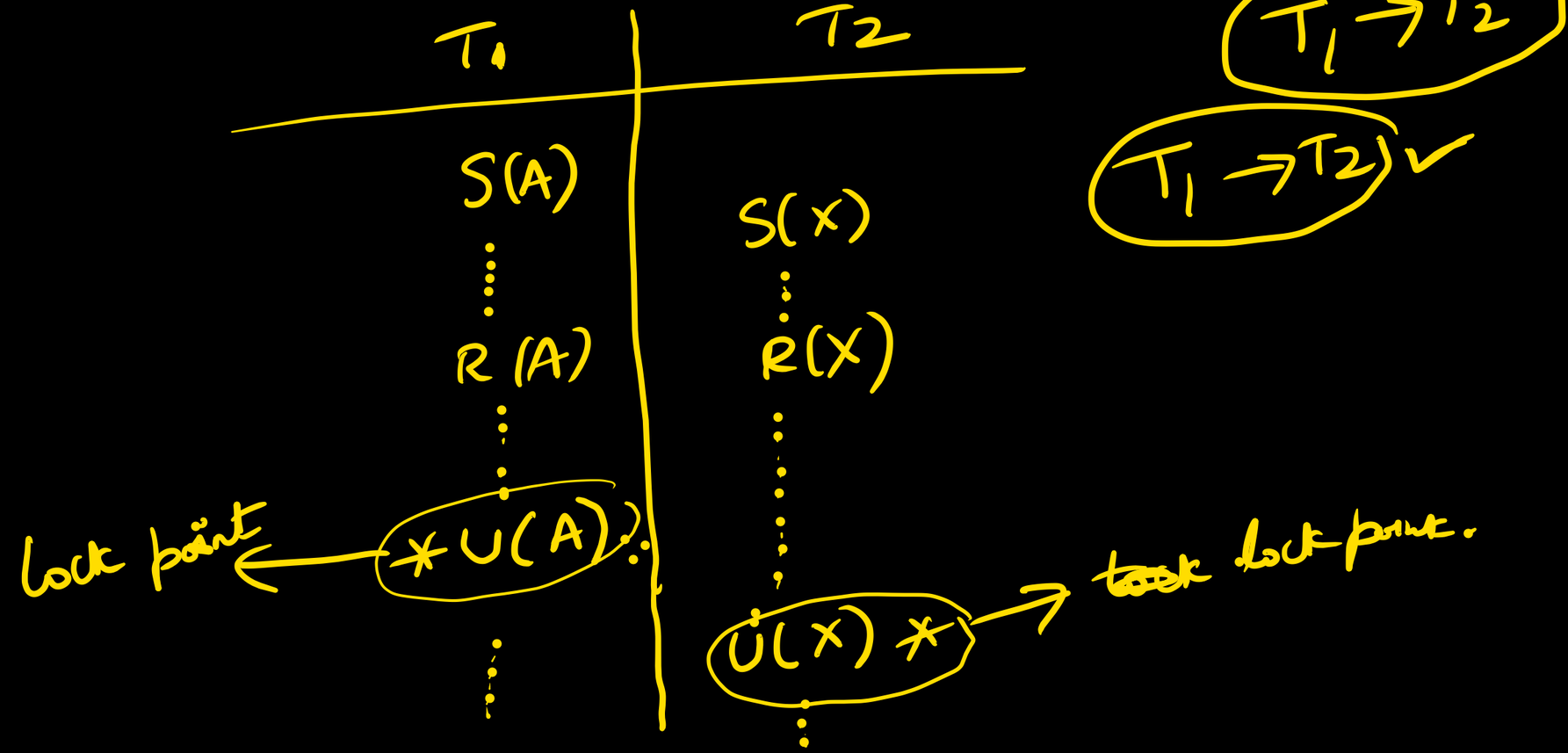
unlocking phase

shrinking phase

is called lock point of transaction

Transaction is not allowed to request locks in unlocking phase of Trans (T).

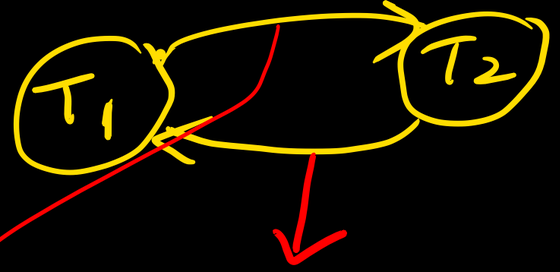
⇒ If $T_1 \rightarrow T_2$ conflict pair has to be executed by 2PL, then lock point of T_1 will be before lock point of T_2



→ If schedule (S) is not conflict serializable schedule (.cyclic precedence graph) then schedule cannot be created by 2PL.

T_1	T_2
*	
	*

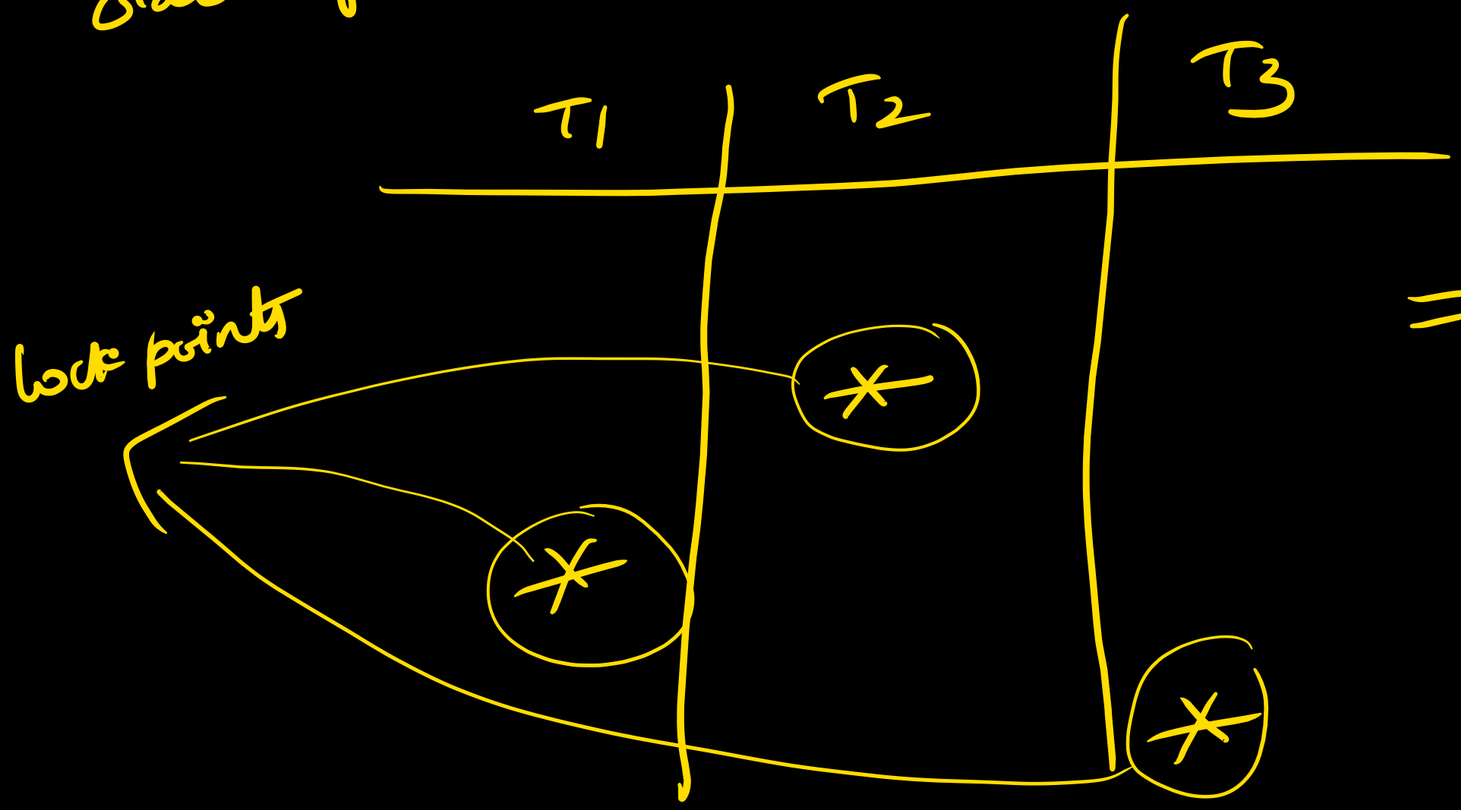
T_1	T_2
	*
*	



Lock point of T_2 should be before lock point of T_1.

Lock point of T_1 should be before Lock point of T_2.

→ If a schedule is executed in 2PL, then it is serializable in the order of lock point.



⇒ Conflict equal to serial schedule $T_2 T_1 T_3$

Doubt: What happens if we execute non conflict serializable schedule in 2PL?

S: $r_1(A)$ $w_2(A)$ $w_2(B)$ $r_1(B)$

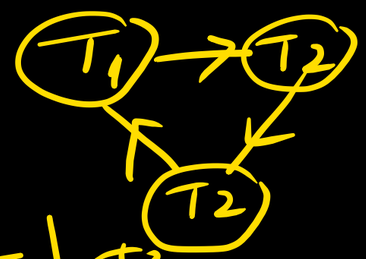
T_1	T_2
S(A)	
$r_1(A)$	
S(B)	
U(A)	
	X(BA)
	$w_2(A)$
	X(B)
	$w_2(B)$
$r_1(B)$	

Schedule will be stopped

denied (because S(B) is with T_1)

Ex: $r_1(A), w_2(A), w_2(B), r_3(B), r_3(C), w_1(C)$

T_1	T_2	T_3
S(A)		
$r_1(A)$		
X(C)		
U(A)	X(A)	
	$w_2(A)$	
	X(B)	
	$w_2(B)$	
	U(B)	
T_1 cannot acquire X(C) till T_2 finishes		S(B)
		$r_3(B)$
		S(C)
		$r_3(C)$
$w_1(C)$		
$w_1(C)$ ✓		



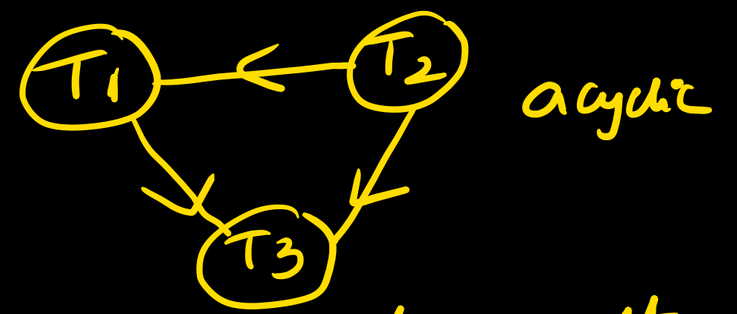
T_1	T_2	T_3
S(A)		
$r_1(A)$		
	X(A)	
	w_2	
		denied

denied

T_3 aborts & schedule will stop.

5) $r_1(A)\ r_3(A)\ r_2(A)\ w_2(B)\ w_1(B)\ w_3(B)$

T_1	T_2	T_3
S(A)		S(A)
$r_1(A)$		$r_3(A)$
	S(A)	
	$r_2(A)$	
	X(B)	
	$w_2(B)$	
	*U(A)	
	U(B)	
X(B)		
$w_1(B)$		
*U(A)		
U(B)		
		X(B)
		$w_3(B)$
		U(A) U(B) *



acyclic

Topological sort

$T_2\ T_1\ T_3$

lock point

$T_2 \rightarrow T_1 \rightarrow T_3$

lock point order = one of the topological order of precedence graph ✓

Every schedule allowed in 2PL is also Conflict serializable but not every Conflict serializable schedule is allowed by 2PL.