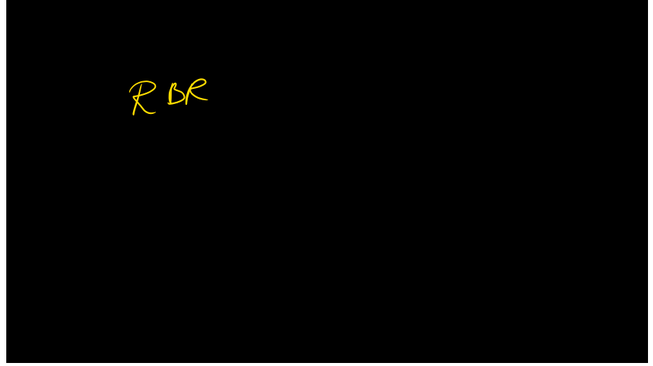
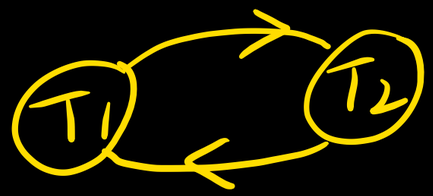

R BR

Re

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:



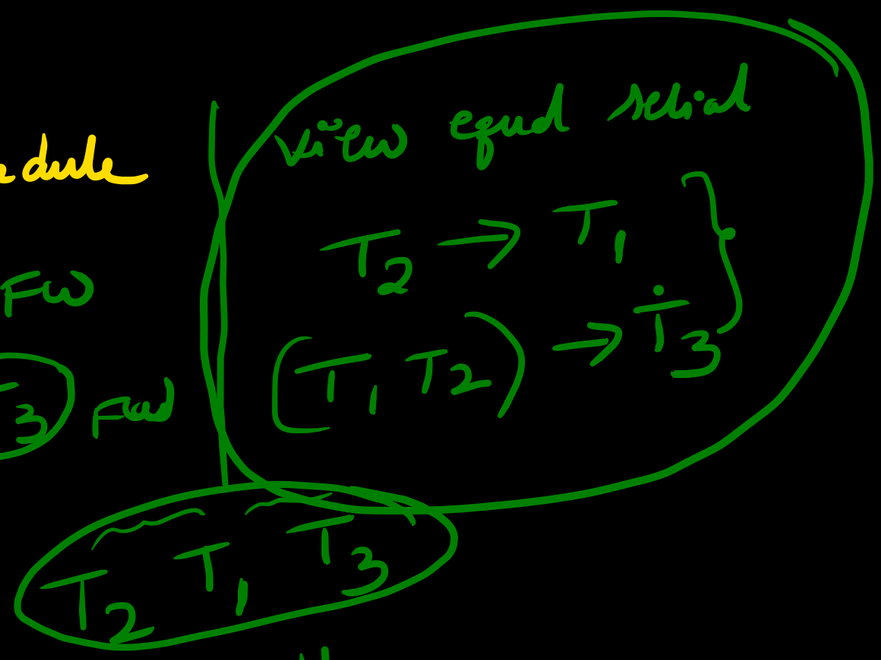
not CS.

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.

$T_1\ T_2\ T_3$

$T_1\ T_3\ T_2$

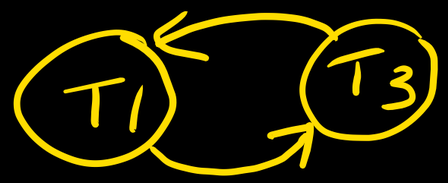
$T_2\ T_1\ T_3$ ✓ View serial equal schedule.

$T_2\ T_3\ T_1$

$T_3\ T_1\ T_2$

$T_3\ T_2\ T_1$

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

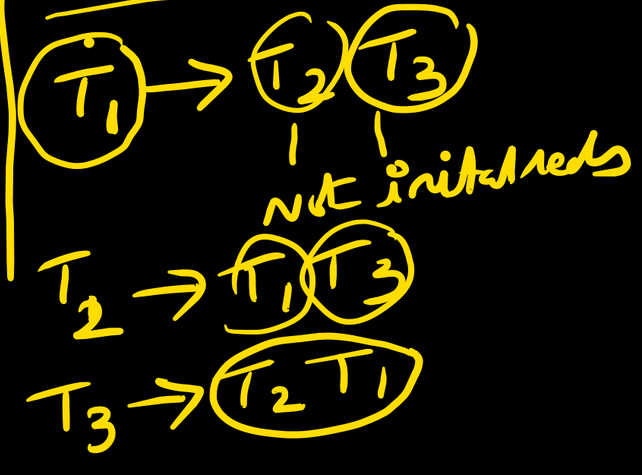
$r_1(A)$ $w_1(A)$ $r_2(A)$ $w_2(A)$

Initial read

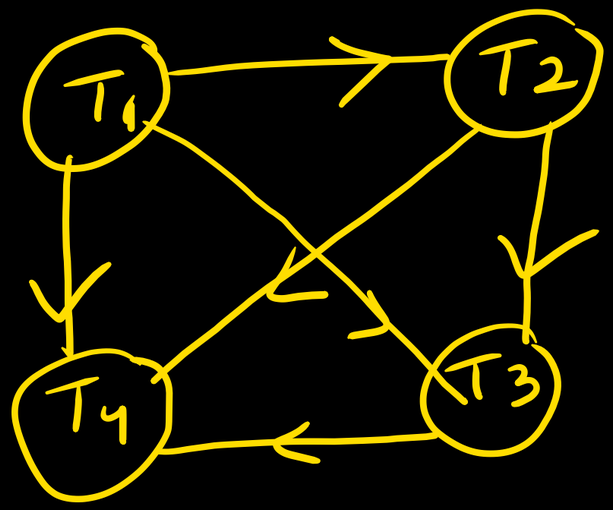
data item	IR	write
A	T_1 T_2 T_3	T_1 T_2 T_3

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

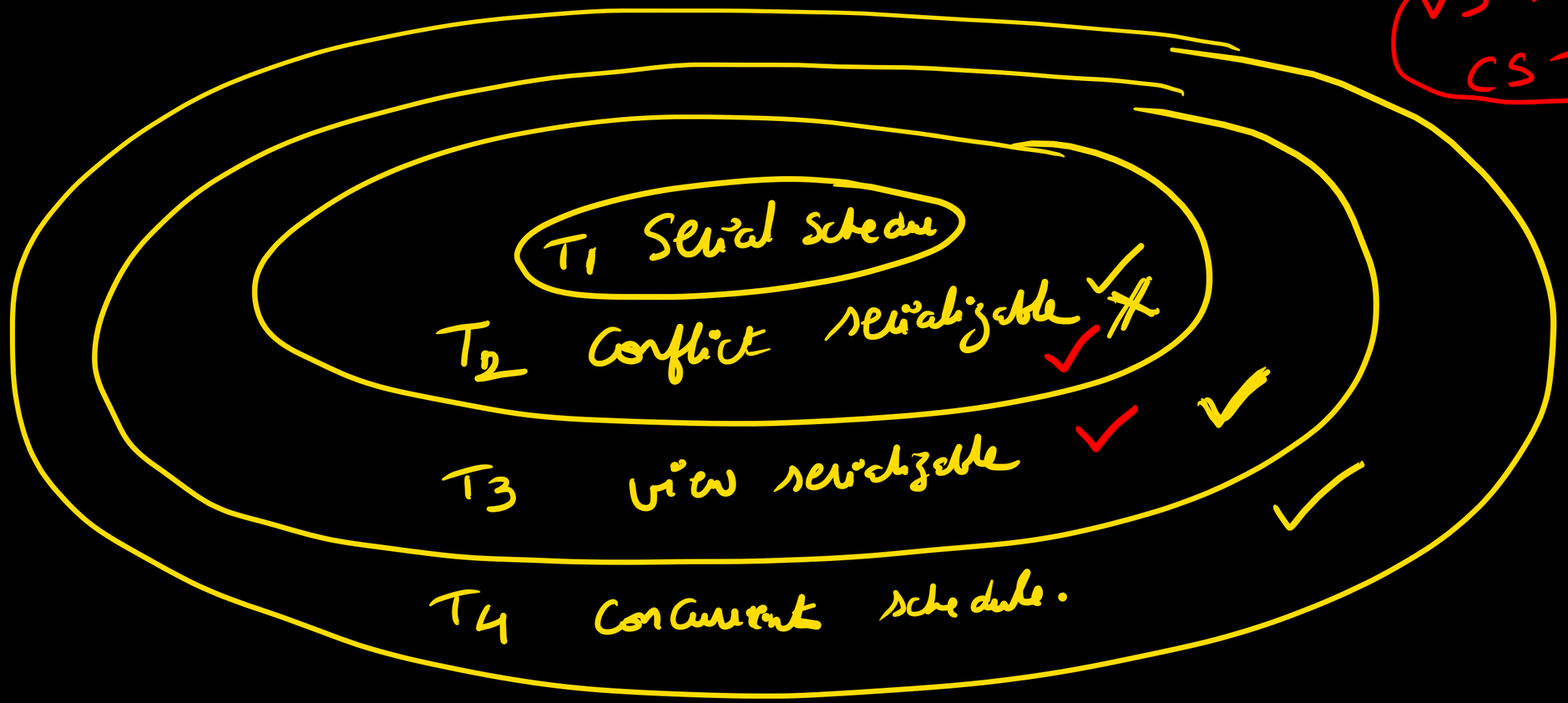
NO restrictions

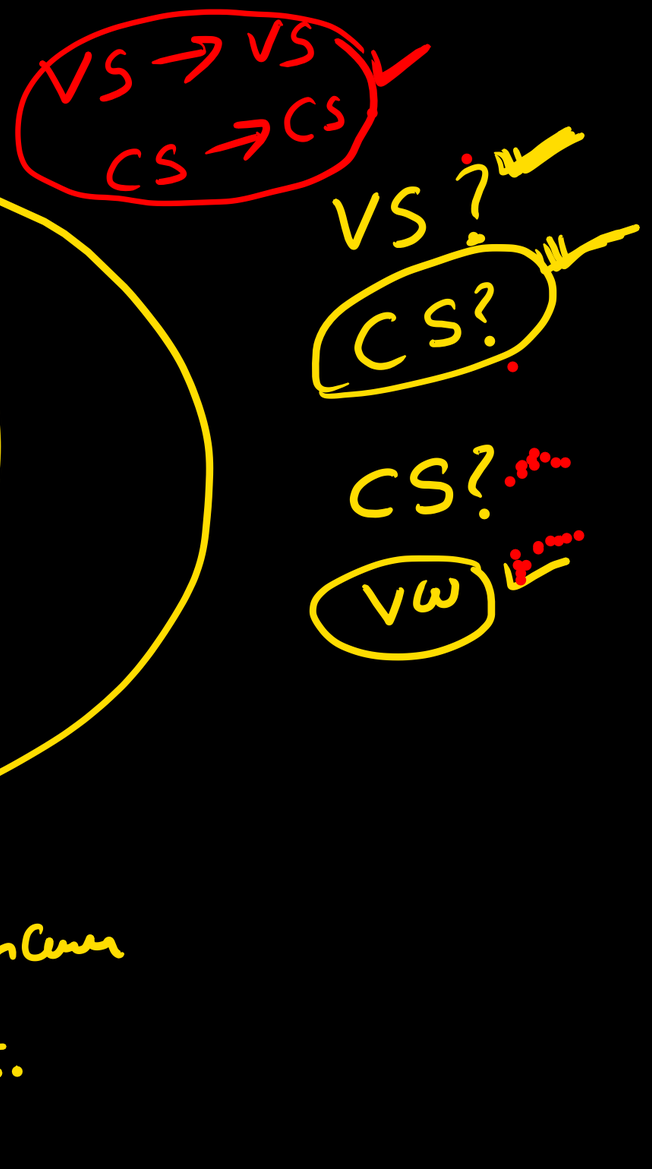
$T_1 \times$
$T_3 \rightarrow T_4$ ✓

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

$T_1\ T_3\ T_4$
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.

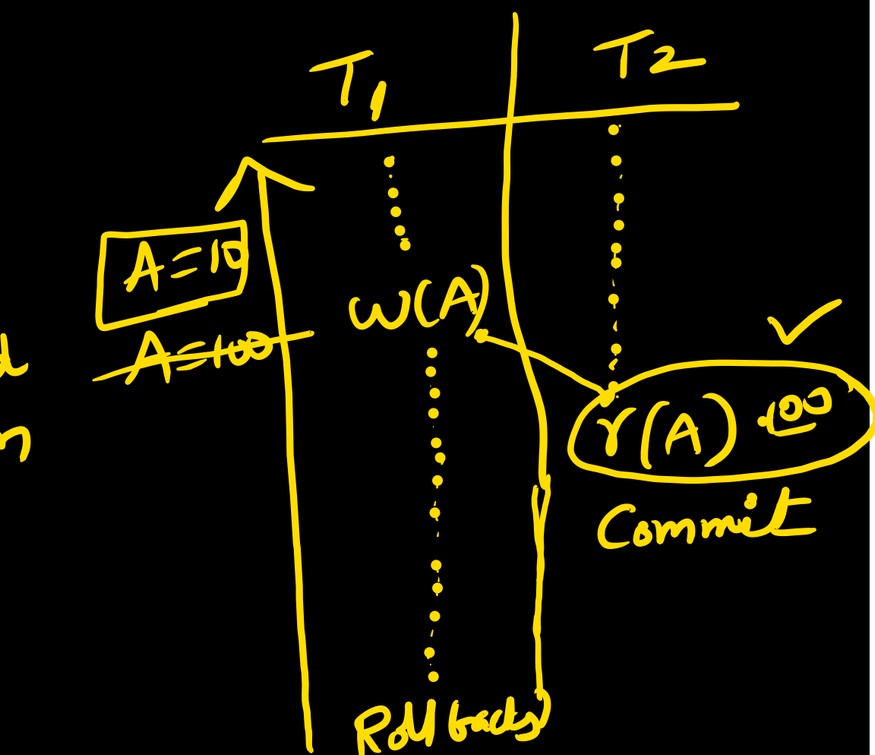
(ii) Cascading roll back problem.

(iii) lost update problem.

Irrecoverable problem :-

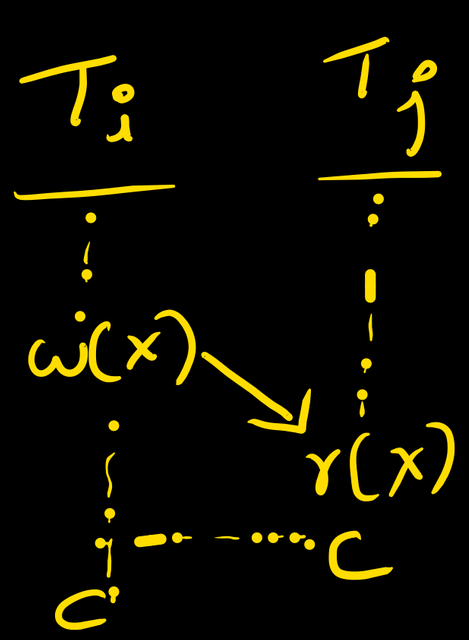
When it is required to roll back a committed transaction, then it is called irrecoverable schedule.

We cannot roll back a committed transaction



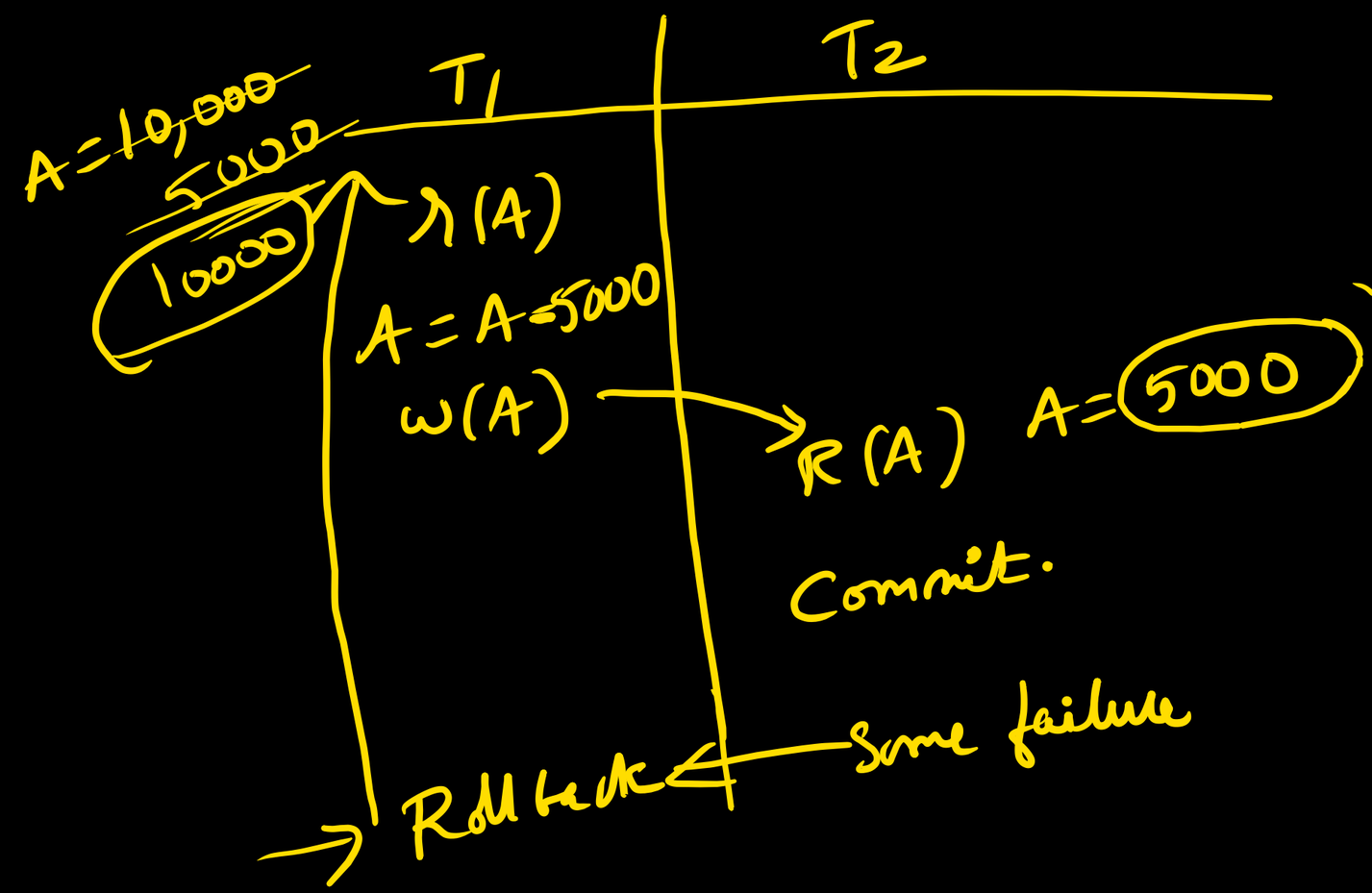
Formal definition:

A Schedule (S) is ~~irrecoverable~~ irrecoverable iff transaction T_j reads data item 'x' from T_i and commit of T_j happens before commit/Roll back of T_i.



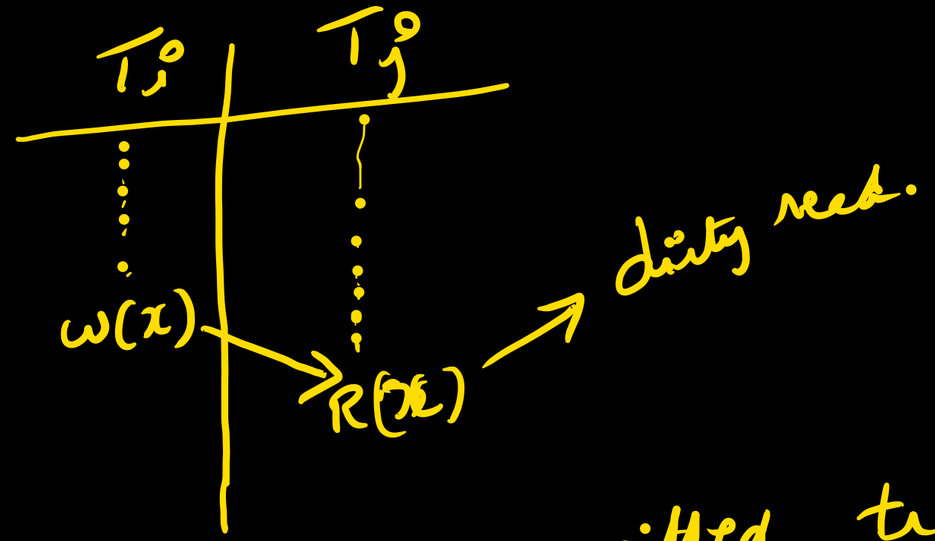
Ex: T_1 : withdraw 5000 from A $[r_1(A)\ A=A-5000\ w_1(A)\ C_1]$

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

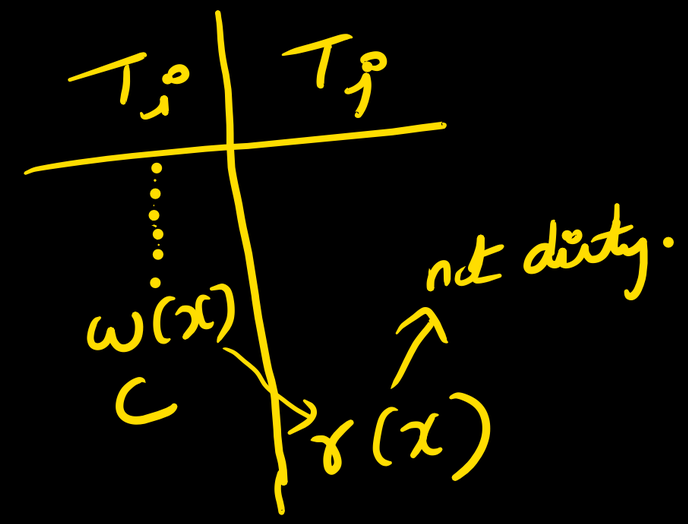


This the problem if schedule is irrecoverable.

uncommitted read & dirty read :-



Reading from uncommitted transaction is a dirty read.



T_1	T_2	T_3
$r_1(A)$		
	$w_2(B)$	
	C_2	
$w(A)$		$R_3(B)$
	$R(A)$	

Not dirty

$R_3(B)$ → not dirty

$R(A)$ → 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

Recoverable schedule:

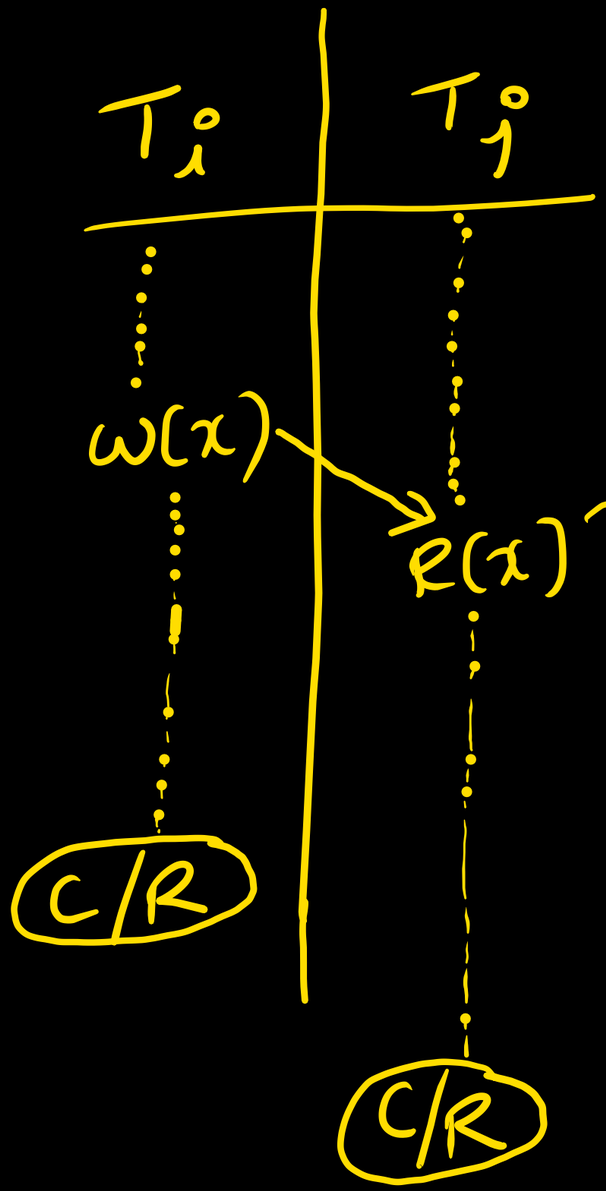
Schedule 'S' is recoverable iff

(1) no deity 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:

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

Dirty read

RB

Cascading

Recoverable

S:

T_1	T_2
R(A)	
	R(A) ⇝ Dirty read ✗
W(A)	
	W(A)
C_1	
	C_2

Not dirty read

Cascadeless

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

dirty

not dirty

not dirty

Cascading

Recoverable

4)

T_1	T_2
$w(A)$	
	$w(A)$
	$w(B)$
$w(B)$	
	C_2
C_1	

Cascadeless

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

Not dirty ← $R(A)$

$R(B)$ → Not dirty

$R(C)$ → Not dirty

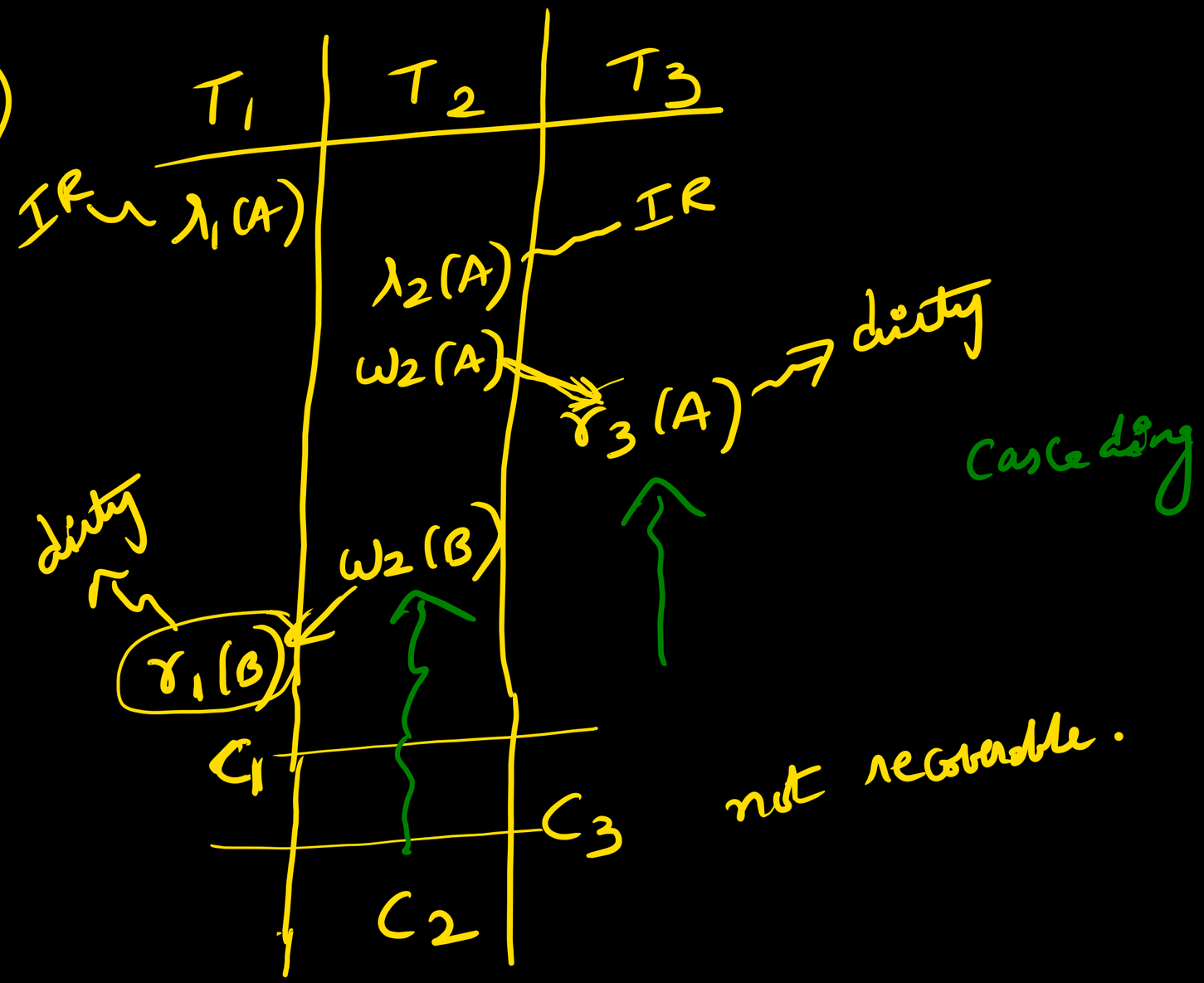
$R(A)$ → Not dirty

$R(B)$ → Not dirty

Cascadeless

Recoverable
No dirty reads

G)



not recoverable.

7)

T_1	T_2
IR	
$\gamma_1(A)$	
	$\gamma_2(A)$ → IR
	$\omega_2(A)$
DR ← $\gamma_1(B)$	$\omega_2(B)$

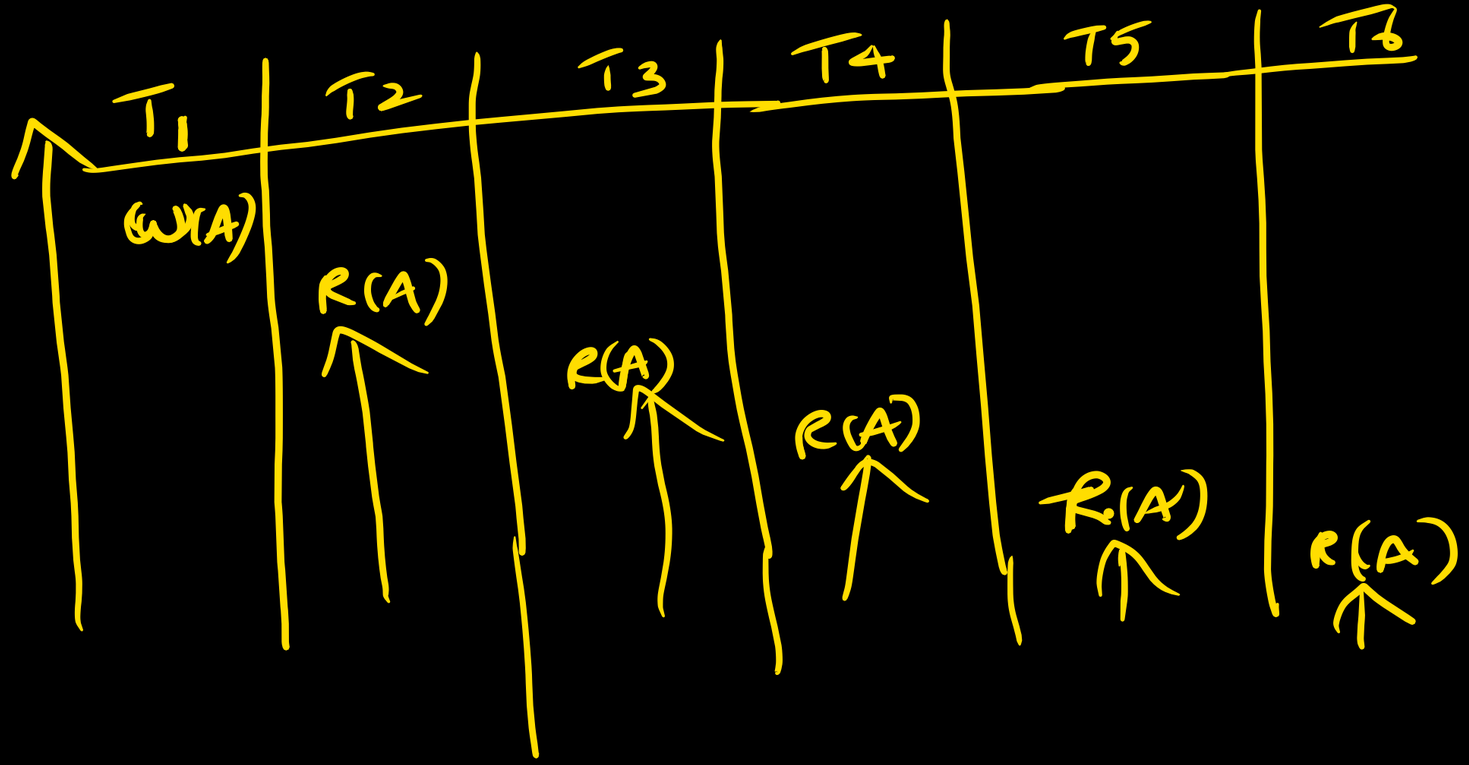
C_1 C_2

Recoverable

Cascading.

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 back :-

Dirty reads not allowed