Limitations of 2PL

2PL may have deadlocks.

T_1 : $r_1(A)$ $r_1(B)$

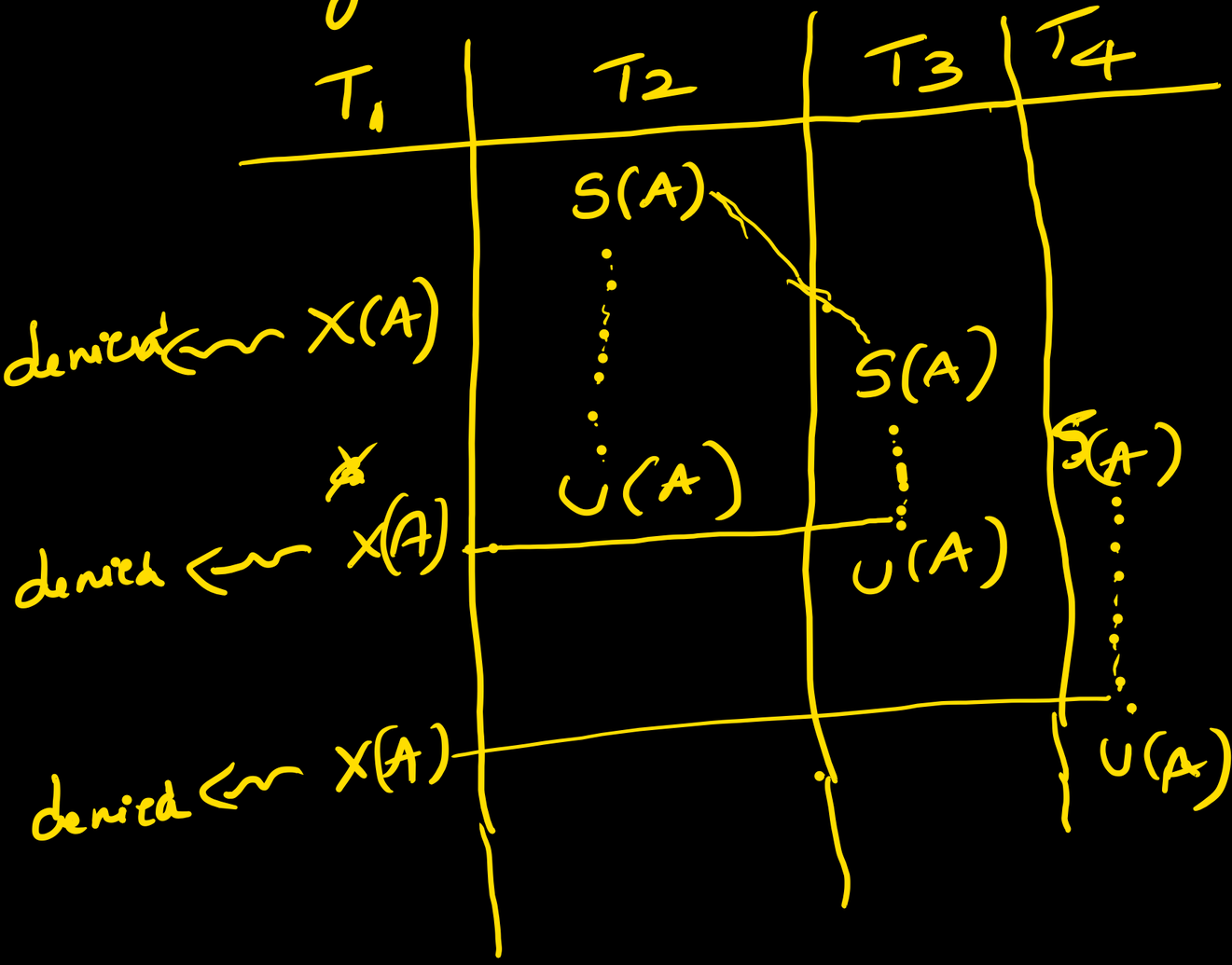
T_2 : $w_2(B)$ $w_2(A)$

Some schedule:

T_1	T_2
$S(A)$	
$r_1(A)$	
	$X(B)$ ←
	$w_2(B)$
denied ← $S(B)$ b/c T_2 has $X(B)$	
$r_1(B)$	
	$X(A)$ ← denied because T_1 has lock on A.
	$w_2(A)$

Deadlock

2) 2PL may lead to starvation



∴ T_1 is starving.

It may happen any amount of time.

DBMS — data item = critical section.

OS

3) 2PL restriction not sufficient for avoiding irrecoverable problem, Cascading RB problem and lost update problem.

Ex: S: $w_1(A)$ $r_2(A)$ $w_1(B)$ $w_2(B)$ C_2 C_1

T_1	T_2
$w_1(A)$	$r_2(A)$ ⇝ dirty read
$w_1(B)$	$w_2(B)$
	C_2 → T_2 commits before T_1
C_1	So irrecoverable schedule.

Even though it is not recoverable, it is allowed in 2PL.

Next slide → 2PL execution

S: $w_1(A)$ $r_2(A)$ $w_1(B)$ $w_2(B)$ C_2 C_1

T_1	T_2
$X(A)$	
$w_1(A)$	
$\rightarrow X(B)$	
$U(A)$	$S(A)$ ·
	$r_2(A)$
$\Rightarrow w_1(B)$	
$U(B)$	$X(B)$
	$w_2(B)$ $\leftarrow$
	$U(A)$
	$U(B)$
C_1	C_2

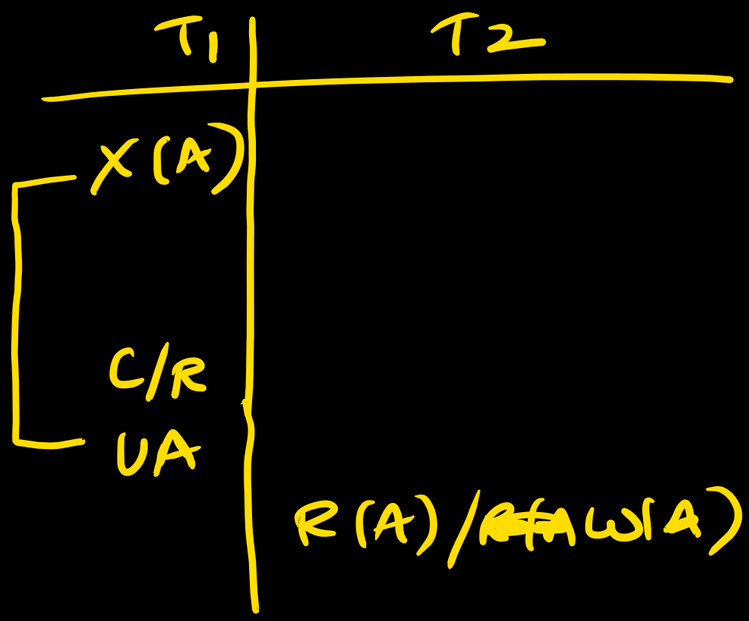
even irrecovable schedules are allowed in 2PL. also Cascading RB and lost updates are also possible in 2PL

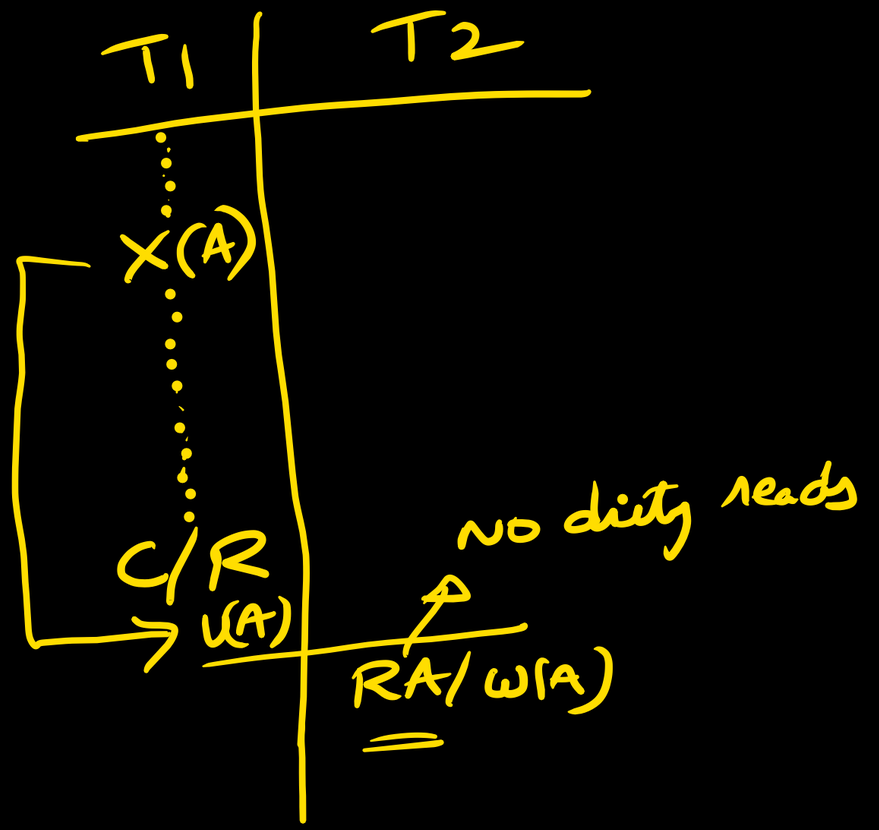
Strict 2PL protocol :-

(1) Basic 2PL : lock requests are not allowed in unlocking phase

and

(2) Strict recoverable property :



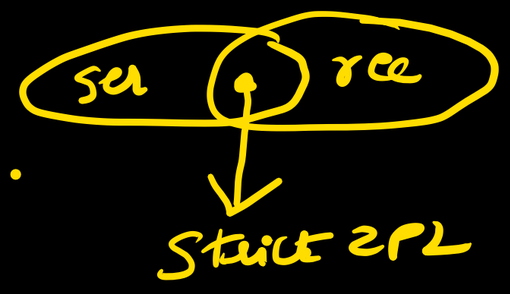


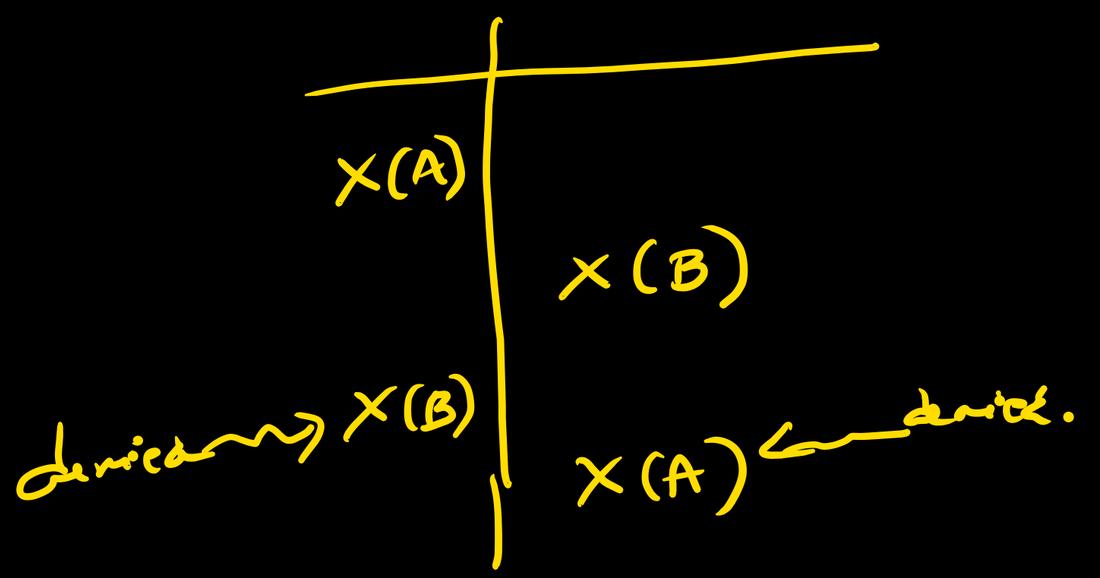
adv of Strict 2PL:

⇒ Guarantees Conflict serializable (2PL)
(equal to the serial schedule based on lock points)

⇒ Guarantees Strict recoverability

→ & Deadlocks and starvation may be possible.





Rigorous 2PL:

Basic 2PL: lock requests not allowed in unlocking phase

and

All locks of trans(T) shared/exclusive lock must hold until commit/roll back

S(A)

R(A)

X(A)

W(A)

S(B)

R(B)

X(C)

W(C)

S(D)

R(D)

Commit

U(A)

U(B)

U(C)

U(D)

lock point

lock point

lock point

Rigorous 2PL:

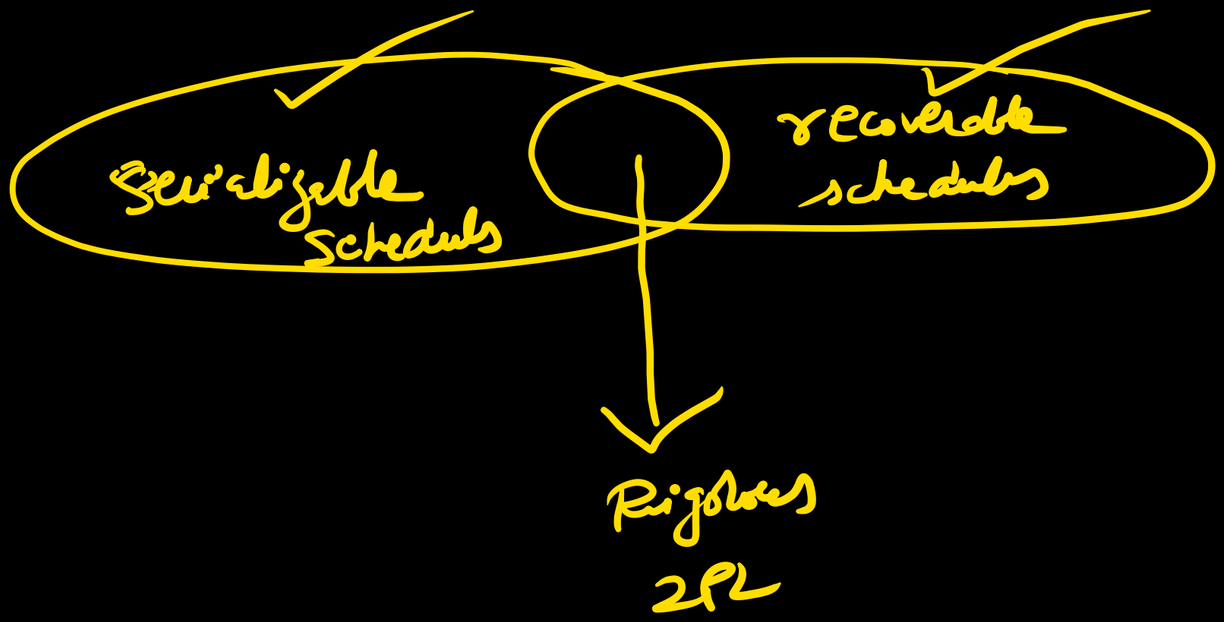
→ Guarantees of conflict serializability. Each serial schedule is in order of commit

T_1	T_2	T_3
C		
		C
	C	

Conflict equal to serial schedule
$T_1 T_3 T_2$

Rigorous 2PL

→ Guarantees strict recoverability.



Disadv of rigorous 2PL:

Deadlock and starvation are possible

Conservative 2PL protocol:-

(deadlock free protocol)

→ Transaction must check availability of all data items before r/w operations. If all data items are available then transaction

T_1 : $r_1(A)$ $w_1(B)$ $w_1(C)$

Locks: (S(A) X(B) X(C))

If all of these are availble, the only transaction will start

T_1
S(A)
X(B)
X(C)
$r_1(A)$
$w_1(B)$
$w_1(C)$
Commit
U(A)
U(B)
U(C)
Commit

~~Disadv of Conservative~~

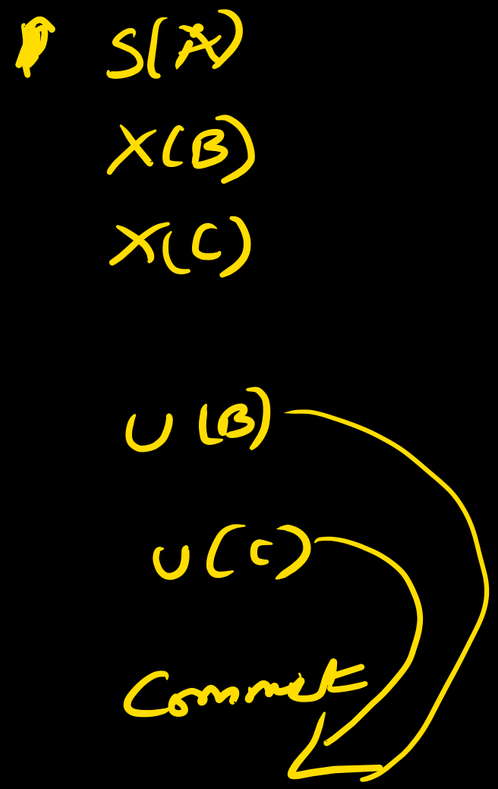
adv of Conservative:

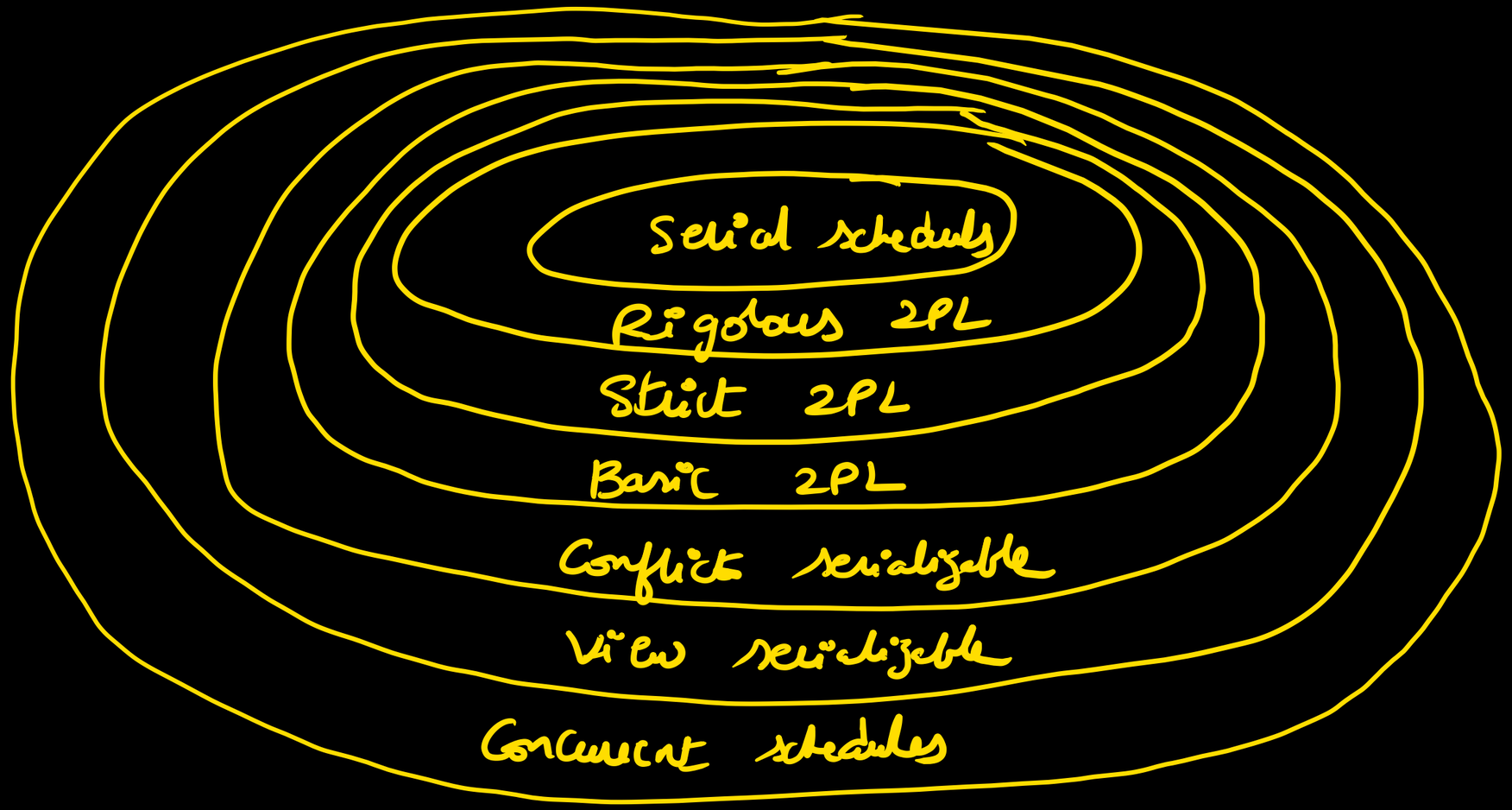
NO deadlocks.

& Conflict serializable

Dis of Conservative:-

1) less degree of Concurency
2) Starvation will occur
3) not strict ~~Acct~~ Recoverable as exclusive locks are rel released after Commit





If a schedule is conflict serializable, then it can be executed in Rigorous 2PL. may & may not

Time stamp ordering protocols:

Every transaction which enters the system will get a number called time stamp.

10	20	25	30	35
T_1	T_2	T_3	T_4	T_5

Older transactions have lower numbers and younger transactions will have higher numbers.

T_i^{10} T_j^{100}

T_i is older to T_j

T_j is younger to T_i

$TS(T_i) < TS(T_j)$

T_i is older

T_j is younger.

Some times, time stemps values can be used as trans-ID of priority to the transactions

Data items will also get ~~TS~~ TS.

(10) T_1	20 T_2
$RTS(A)=10 \leftarrow r(A)$	$w(A) \rightarrow wTS(A)=20$

Time stamp of data item (x)

Read_TS (x): Highest transaction TS value that has performed R(x).

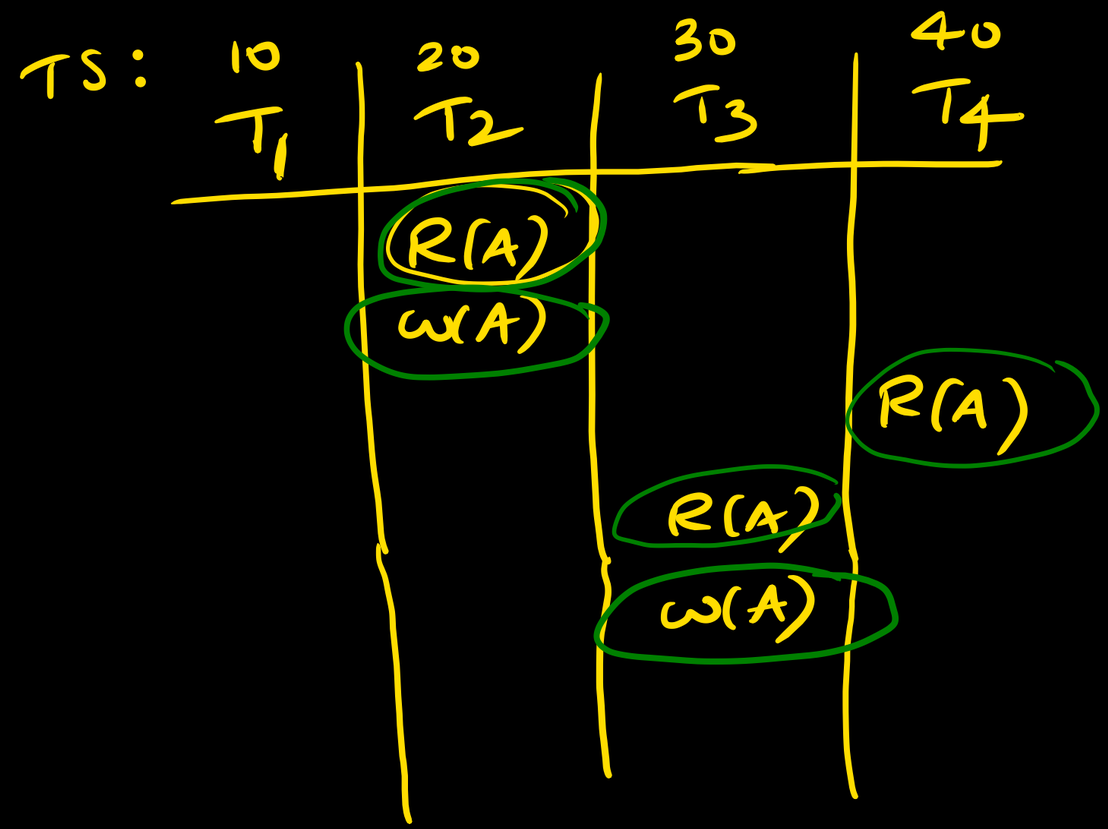
Write_TS(x): Highest transtion TS value that has performed w(x)

TS: 5 10 15 20 25

S:

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

	A	B
RTS	~~20~~	~~10~~ 15
WTS	5	


TS:
10
T1
20
T2
30
T3
40
T4
R(A)
W(A)
R(A)
R(A)
W(A)
RTS(A): ~~20~~ 40
WTS(A): ~~20~~ 30

Data item (A) transaction T

RTS (A) > TS(T)

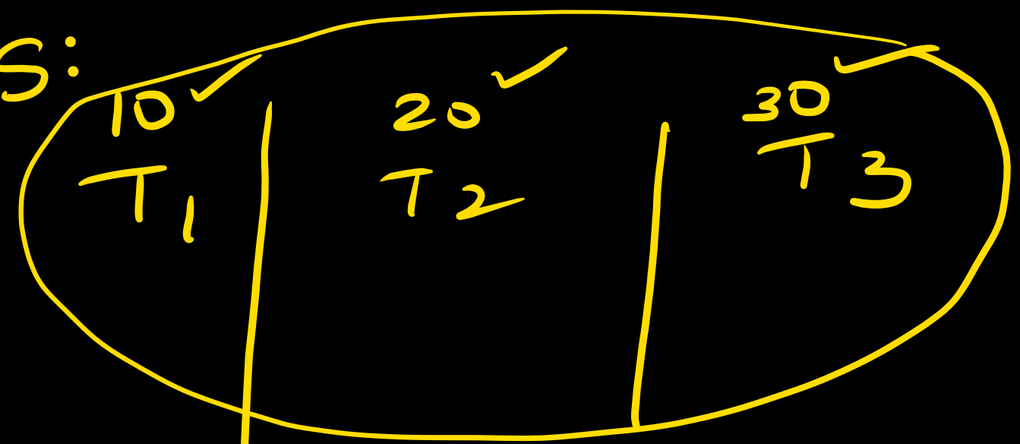
we can say there an younger transaction$_x$ than T has done R(A)

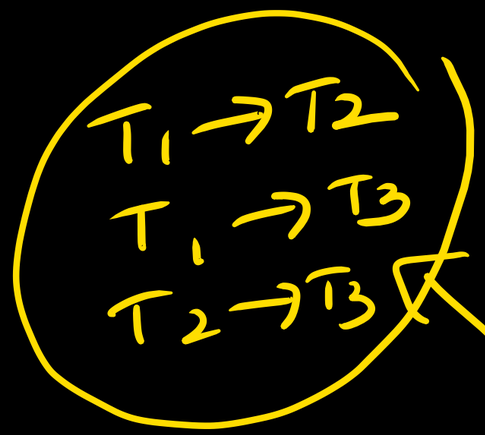
else NO younger transaction$_x$ than T has done R(A)

Basic Time stamp ordering protocol:-

Concurrent execution must be equal to if time stamp order Serial.

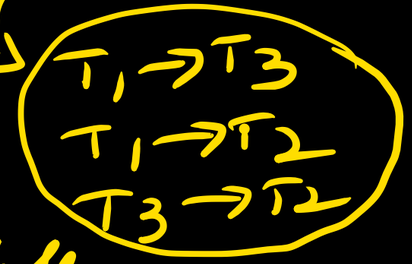
S:





→ Conflict serial schedule that 'S' is equal to.

10 30 20
T1 T2 T3



Possible rollbacks:

TS: 10 T_1	20 T_2
rollback	$W(A)$
$R(A)$	$T_2 \to T_1$

$T_1 \to T_2$

Protocol: Set of rules

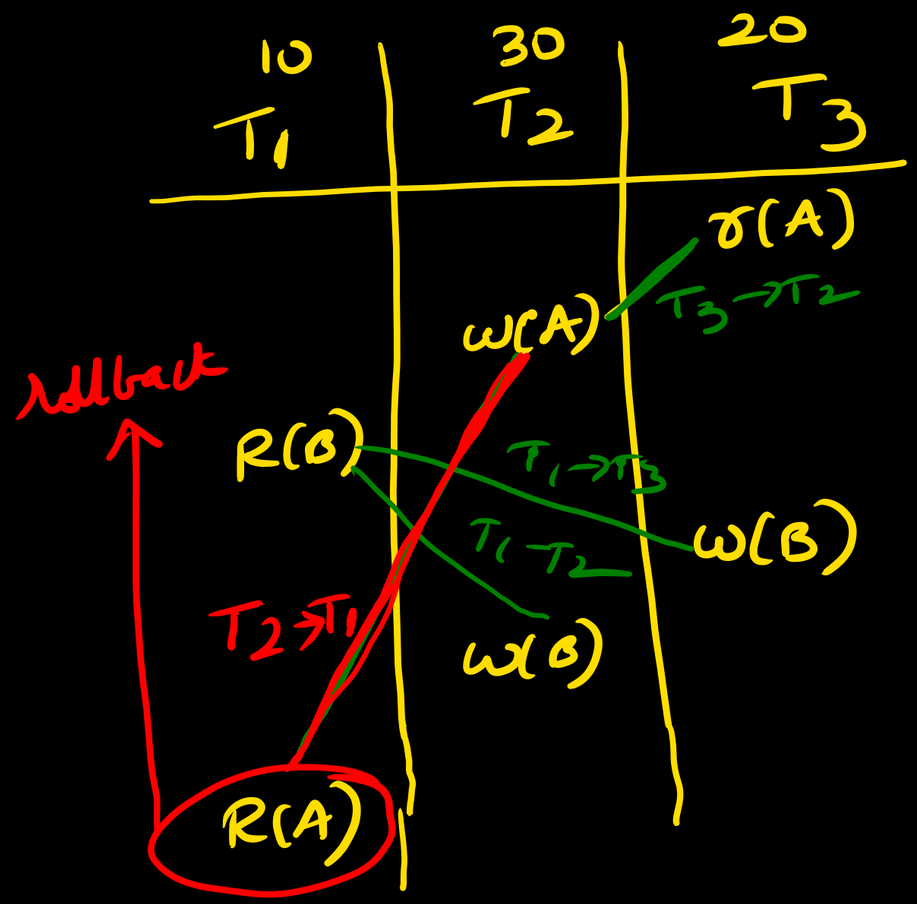
10 T_1	20 T_2
Rollback	$R(A)$
$W(A)$	$T_2 \to T_1$

$T_1 \to T_2$

T_1 10	T_2 20
rollback.	$W(A)$
$W(A)$	$T_2 \to T_1$

$T_1 \to T_2$

Will the following Schedule follow Basic TS Ordering protocol



$T_1 \rightarrow T_3 \rightarrow T_2$

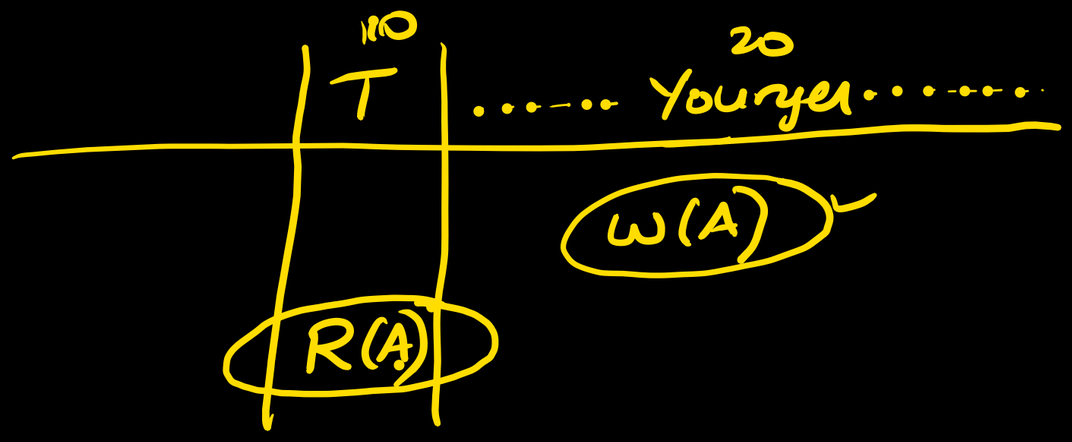
$T_1 \rightarrow T_3$

$T_1 \rightarrow T_2$

$T_3 \rightarrow T_2$

Basic TO protocol:

→ Transaction T issue a ~~write~~ Read (A):



If $WTS(A) > TS(T)$, means younger transaction has already written on (A), so rollback 'T'

else allow R(A) by 'T' and

$RTS(A) = TS(T)$.

2) Transaction T issues (write) w(A):

	T	..... Younger .....
		R(A)/w(A)
	w(A)	

{ if RTS(A) > TS(T) → roll back T
  else if wTS(A) > TS(T) → roll back T
  else allow w(A) by trans (T)
  = Set wTS(A) = TS(T) ✓

LK Examples → tomorrow

if these two conditions failed, it means that no younger transaction has done read/write w(A).

mon, Fri 6 pm - 8 pm

algo ds

gay mon, T, w, T, F, S, S — 815 10:15 practice.

Tues wed thus satur → digital (6:00-8:pm)