what is Co-currency Control ?

→ refers to the techniques and mechanism used to manage and co-ordinate access to shared resources when multiple transaction are executing co-currently

Goal

(1) Lock based Co-currency control → transaction operates correctly

(2) 2PL
2-phase Locking → consistently
→ efficiently

(3) Multiversion cocurrency control

(4) Timestamp ordering

Lock Based Concurrency

→ Controling co-current access ; transaction must acquire appropriate locks before accessing / modifying a data item.

T_2

lock-S(A) : None can use A untill it unlocked
read(A)
unlock(A)
Lock-S(B)
read(B)
Unlock(B)

two type

Exclusive Lock

→ X mode
→ instruction :- lock-X
→ write Lock
→ when transaction want to modify data
→ one transaction at a time

shared Lock

→ S mode
→ instruction : Lock-S
→ Read Lock
→ read on accessing data
→ multiple transaction at a time

2PL - Phase Locking

→ is a cocurrency control that ensure serializability and prevent conflicts between co-current transaction

→ divided in two phase

Growing phase

→ transaction acquires locks on data

→ can obtain more locks without releasing Lock

obtain Lock ✓
release Lock X

Phase-1

Lock-S
Lock-X
convert Lock S → Lock X
(upgrade)

shrinking phase

→ transaction release the lock

→ cannot acquire any more lock

release Lock ✓
obtain Lock X

Phase-2

~~Loc~~
unlock(S)
unlock(X)

convert Lock X → Lock S
(downgrade)

Renar

ALTE

T_1	T_2	
(1) lock-S(X)	Lock-X(X)	growing
(2) Lock-S(Y)	Lock-X(Y)	
		shrinking
unlock(X)	unlock(X)	
unlock(Y)	unLock(Y)	

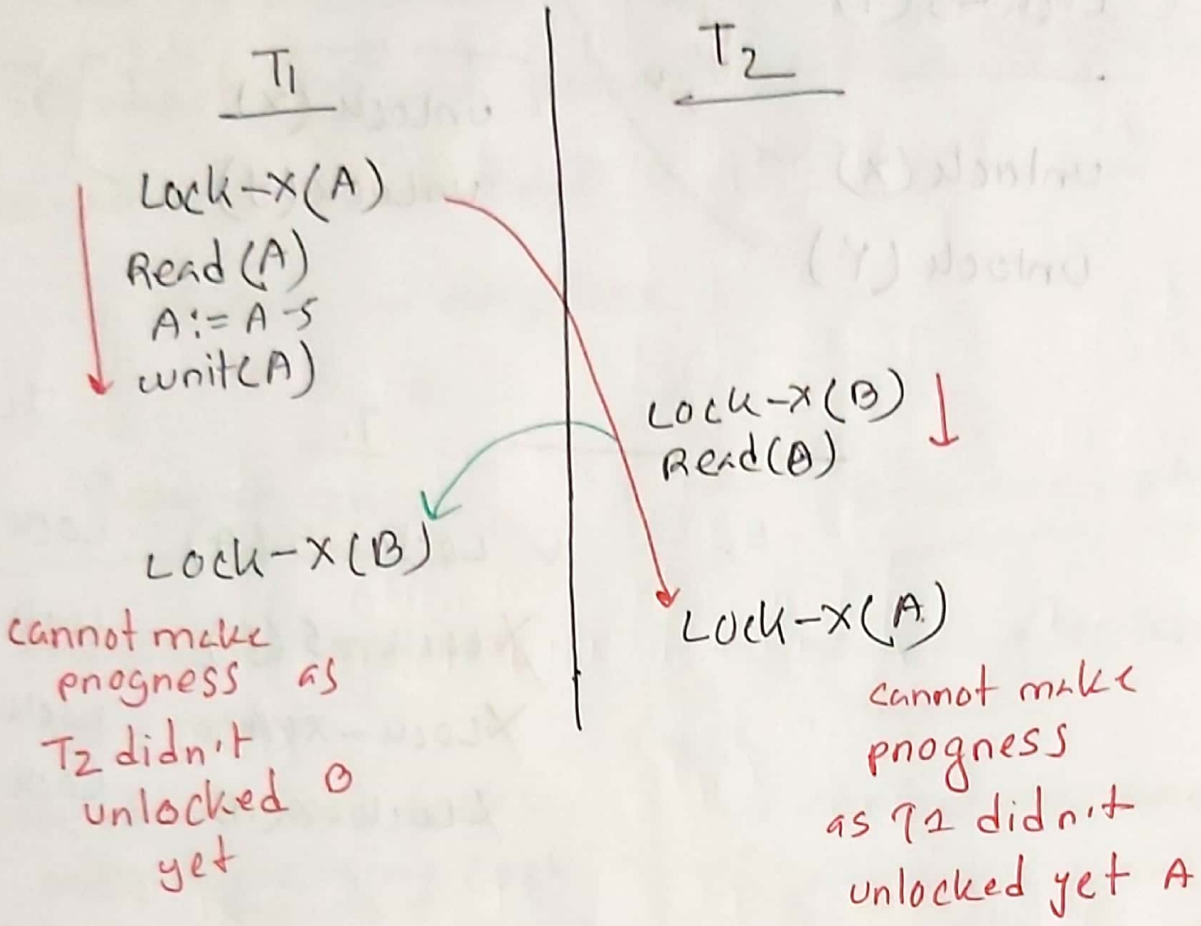
now can use

T_1	T_2
✓ Lock-S(A)	Lock-S(A)
✗ Lock-S(A)	Lock-X(A)
✗ Lock-X(A)	Lock-S(A)
✗ Lock-X(A)	Lock-X(A)

Drawbacks

- may lead to irrecoverability
- Cascading Rollback
- Deadlocks

Deadlock



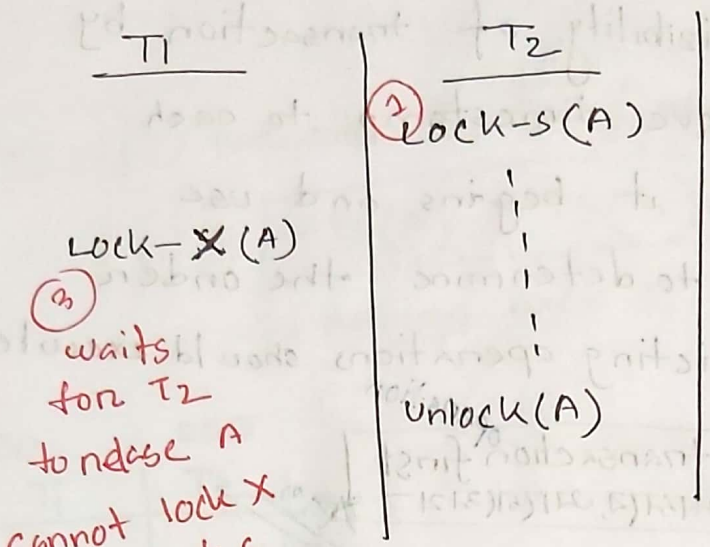
T_1 waiting for T_2 to release B

T_2 waiting for T_1 to release A

This situation = Deadlock

a system is deadlock if there is a set of transaction such that every transaction in the set is waiting for another transaction in the set

Starvation



→ help removing cascading Rollback

Strict 2PL — unlock after commit done

exclusive locks should hold until comit/abort

Ridgorous 2PL — same

shared/exclusive lock hold until comit

Timestamp Ordering

→ ensure serializibility of transaction by assigning an unique timestamp to each transactions when it begins and use these timestamps to determine the order in which conflicting operations should executed

Olden transaction operation first

R(A) R(A)

T_2	T_3	T_4	
100	200	300	→ timestamp
olden		youngest	

(1) $TS(T_i)$ = timestamp of T_i

$TS(T_2) = 100$, $TS(T_3) = 200$

(2) RTS = Readtimestamp = Last or Latest transaction timestamp than Read successfully

$RTS(A)$ =

Renar

ALTE

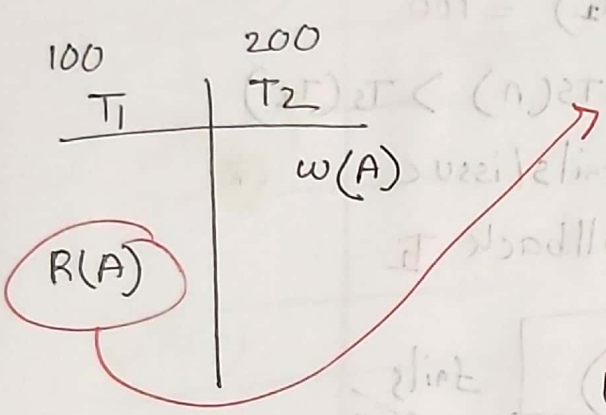
(3) WTS = write stamp = Latest/Last write timestamp

WTS(A) =

Rule-1 (1) Transaction T_i fails/issue a Read(A) operation

if $WTS(A) > TS(T_i)$ fails

∴ Rolling back T_i



Here T_2 w(A) comes first

$TS(T_1) = 100$

$WTS(A) = 200$ (write by T_2 ~~first~~ Last)

∴ $WTS(A) > TS(T_1)$

R(A) will fail/issue.

∴ Solv:- ~~A(R(A) wait untill w(A)~~

T_i will Roll back

(1) if $WTS(A) \leq TS(T_i)$ pass

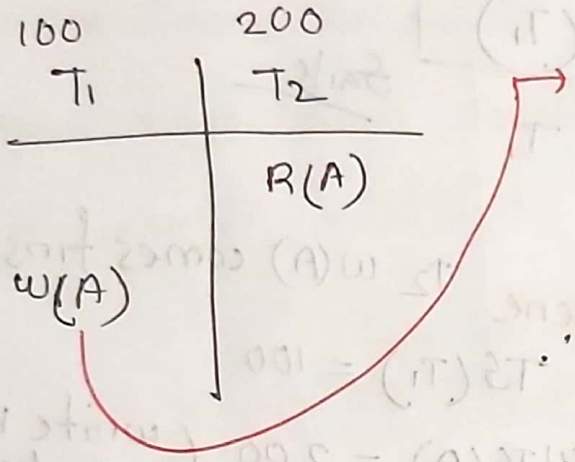
∴ allows read operation

∴ Set $RTS(A) = \max(RTS(A), TS(T_i))$

Rule-2 Transaction T_i fails/issue at write(A) operation

(i) if $RTS(A) > TS(T_i)$ fails

Rolling back T_i



Here, T_2 R(A) comes first

$RTS(A) = 200$

$TS(T_2) = 100$

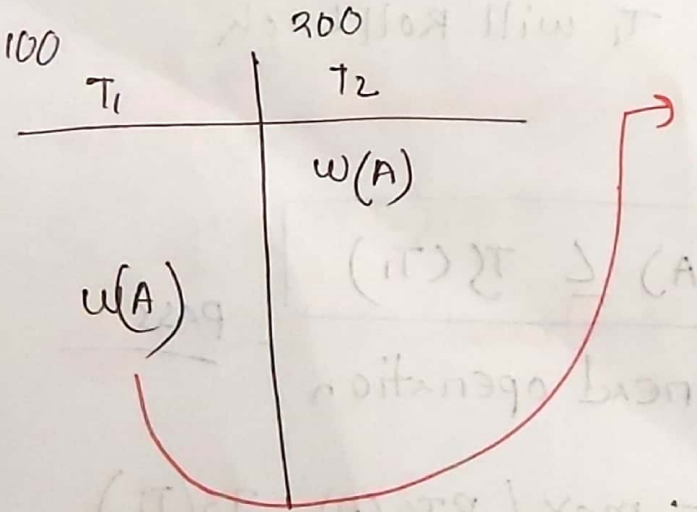
$\therefore RTS(A) > TS(T_2)$

$\therefore$ fails/issue

$\therefore$ Rollback T_2

(ii) $WTS(A) > TS(T_i)$ fails

Rolling back



Here T_2 W(A) comes first

$WTS(A) = 200$

$TS(T_1) = 100$

$WTS > TS(T_2)$

$\therefore$ fails

$\therefore$ Roling back T_1

(iii) else any

allows operations (Read, write)

and set WTS(A) = TS(Ti)