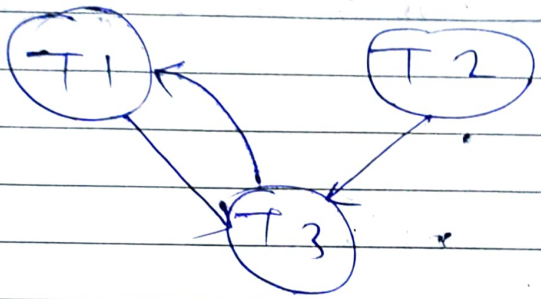
Tutorial 13

Nitin Chaudhary, F7, 9921103163

Ans (1)

(a)

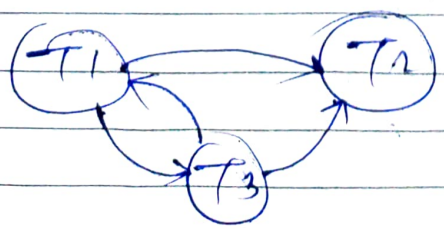
T1	T2	T3
r1(X)		
		r3(X)
w1(X)		
	r2(X)	
		w3(X)



As we have loop. So, this is not conflict serializable.

(b)

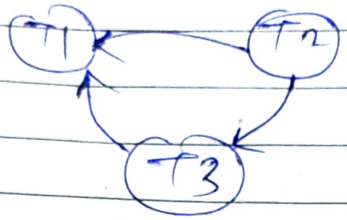
T1	T2	T3
r1(X)		
		r3(X)
		w3(X)
w1(X)		
	r2(X)	



As we have loop, so this is not conflict serializable

(c)

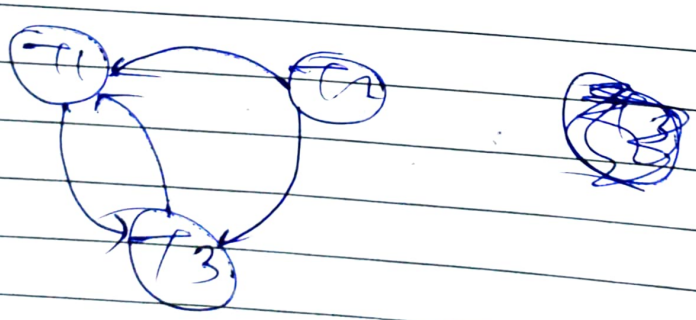
T_1	T_2	T_3
		$r_3(x)$
	$r_2(x)$	
		$w_3(x)$
$r_1(x)$		
$w_1(x)$		



As we do not have any continuous closed directed loop So this is conflict serializable schedule.

(d)

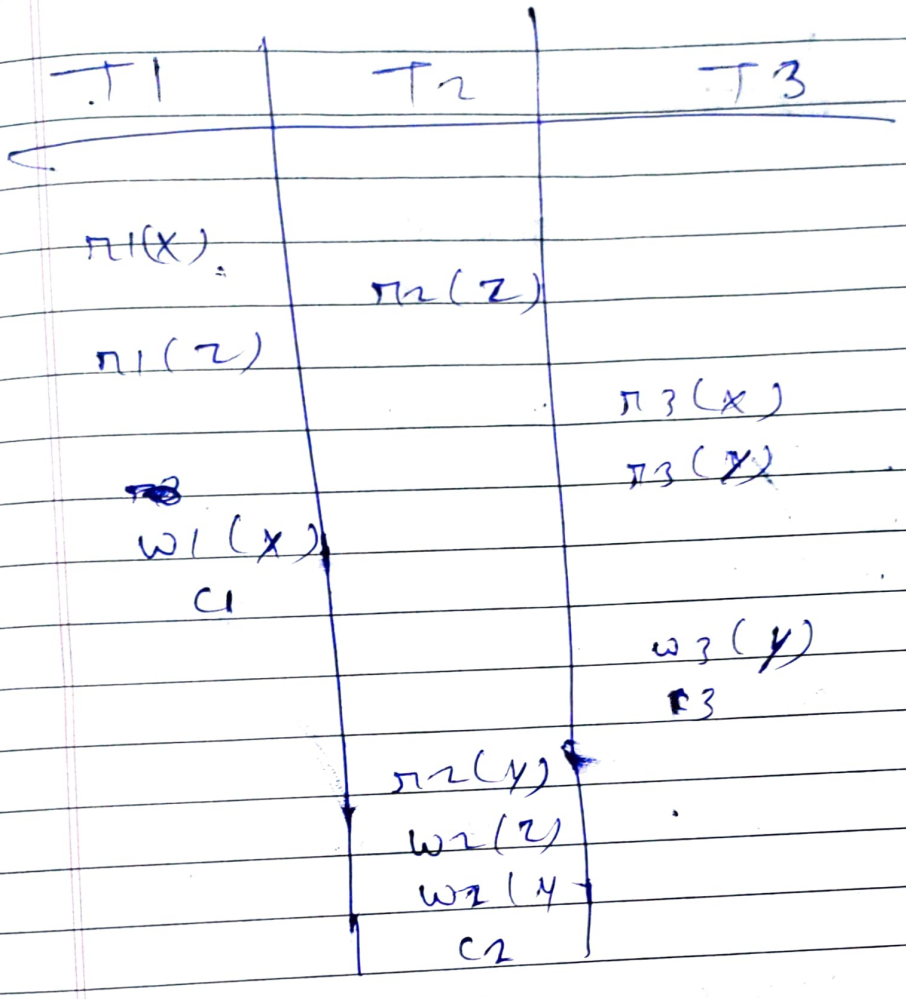
T_1	T_2	T_3
		$r_3(x)$
	$r_2(x)$	
$r_1(x)$		
		$w_3(x)$
$w_1(x)$		



As we have loop. So this is not Conflict serializable.

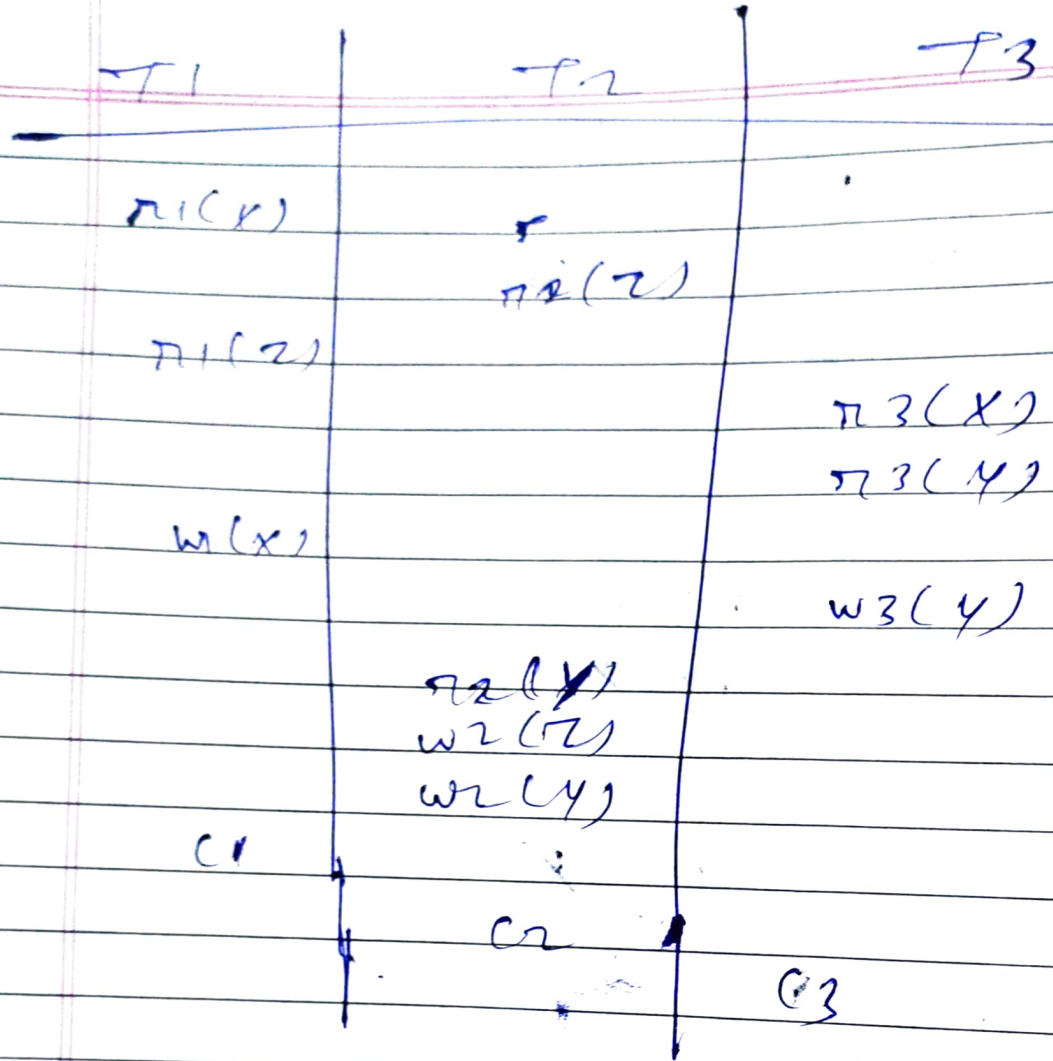
(2)

S3



Strict → Not a strict schedule
Cascadless → Contain cascading rollbacks
Non Recoverable → Between T2 and T3 as T2 started before T3 but committed after T3.

Sq →



Not Strict ✗,

Contain cascading rollback,

Recoverable

S5

T_1	T_2	T_3
$r_1(x)$		
	$r_2(z)$	
		$r_3(x)$
$r_1(z)$		
	$r_2(y)$	
		$r_3(y)$
$w_1(x)$		
c_1		
	$w_2(z)$	
		$w_3(y)$
	$w_2(y)$	
		c_3
	c_2	

Not a strict schedule,
Contains cascading rollbacks,
Not recoverable as T_2 starts before T_3 but it commits after T_3

S1

T_1	T_2	T_3
$R(x)$		
	$W(x)$	
	$W(y)$	
		$W(y)$
$W(y)$		
c		
	c	
		c

S2

T_1	T_2	T_3
$R(x)$		
	$W(y)$	
	$W(x)$	
		$W(y)$
$W(y)$		
c		
	c	
		c

(1) Assume we use wait - Die policy.

Sequence S1: T1 acquired shared lock on X.
when T2 asks for exclusive lock on X, since T2 has a lower priority, it will be aborted.
T3 now gets exclusive lock on X.
when T1 also asks for exclusive lock on Y which is still held by T3, since T1 has higher priority, T1 will be blocked waiting.
T3 now finished write, commits and releases all the locks.
T1 wakes up, acquires the lock, proceeds and finishes.
T2 now can be restarted successfully.

Sequence S2: The sequence and consequence are the same with sequence S1, except T2 was able to advance a little more before it gets aborted.
T1 acquires shared lock on X.
T2 acquires exclusive - lock on Y.
T2 asks for an exclusive - lock on X, but since it has lower priority than T1, it will be aborted.
T3 acquires exclusive lock on Y.
T1 tries to acquire exclusive lock, but since it has higher priority than T3, it is allowed to wait.
T3 releases the lock and commits.
T1 acquire the lock on Y and proceeds.

Q: In deadlock detection, transactions are allowed to wait, they are not aborted until a deadlock has been detected. (Compared to prevention schema, some transactions may have been aborted prematurely)

Sequence S1: T1 gets a shared-lock on X.
T2 blocks, waiting for an exclusive-lock on X
T3 gets an exclusive-lock on Y;
T1 blocks waiting for an exclusive lock on Y;
T3 finishes, commits and releases locks,
T1 wakes up, gets an exclusive-lock on Y, finishes up and releases lock on X and Y;
T2 now gets both an exclusive-lock on X and Y, and proceeds to finish.
No deadlocks.

Sequence S2: There is a deadlock. T1 waits for T2, while T2 waits for T1. Deadlocks are resolved either using a timeout or by maintaining a waits-for-graph.

(3) Sequence S1: With Conservative and strict 2PL, the sequence is easy. T1 acquiring lock on both X and Y, commits releases locks, then T2 and then T3.

Sequence S2: Same as sequence S1.

Complete