Hwk10

1.

These 7 schedules are conflict serializable!

T1	T2
READ(X)	
WRITE(X)	
	READ(X)
	WRITE(X)
READ(Y)	
WRITE(Y)	

T1	T2
READ(X)	
WRITE(X)	
	READ(X)
READ(Y)	
	WRITE(X)
WRITE(Y)	

T1	T2
READ(X)	
WRITE(X)	
	READ(X)
READ(Y)	

WRITE(Y)	
	WRITE(X)

T1	T2
READ(X)	
WRITE(X)	
READ(Y)	
	READ(X)
	WRITE(X)
WRITE(Y)	

T1	T2
READ(X)	
WRITE(X)	
READ(Y)	
	READ(X)
WRITE(Y)	
	WRITE(X)

T1	T2
READ(X)	
WRITE(X)	
READ(Y)	

WRITE(Y)	
	READ(X)
	WRITE(X)

T1	T2
	READ(X)
	WRITE(X)
READ(X)	
WRITE(X)	
READ(Y)	
WRITE(Y)	

These 8 schedules are NOT conflict serializable!

T1	T2
	READ(X)
READ(X)	
	WRITE(X)
WRITE(X)	
READ(Y)	
WRITE(Y)	

T1	T2
	READ(X)
READ(X)	

WRITE(X)	
	WRITE(X)
READ(Y)	
WRITE(Y)	

T1	T2
	READ(X)
READ(X)	
WRITE(X)	
READ(Y)	
	WRITE(X)
WRITE(Y)	

T1	T2
	READ(X)
READ(X)	
WRITE(X)	
READ(Y)	
WRITE(Y)	
	WRITE(X)

T1	T2
READ(X)	

	READ(X)
	WRITE(X)
WRITE(X)	
READ(Y)	
WRITE(Y)	

T1	T2
READ(X)	
	READ(X)
WRITE(X)	
	WRITE(X)
READ(Y)	
WRITE(Y)	

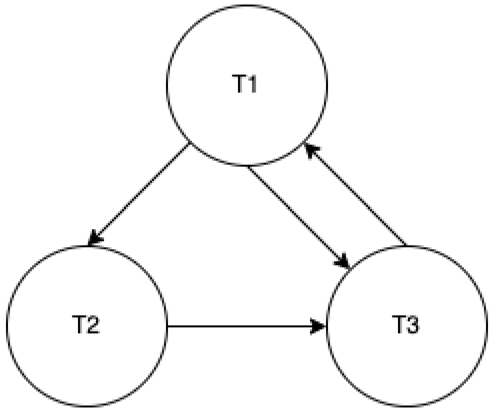
T1	T2
READ(X)	
	READ(X)
WRITE(X)	
READ(Y)	
	WRITE(X)
WRITE(Y)	

T1	T2

READ(X)	
	READ(X)
WRITE(X)	
READ(Y)	
WRITE(Y)	
	WRITE(X)

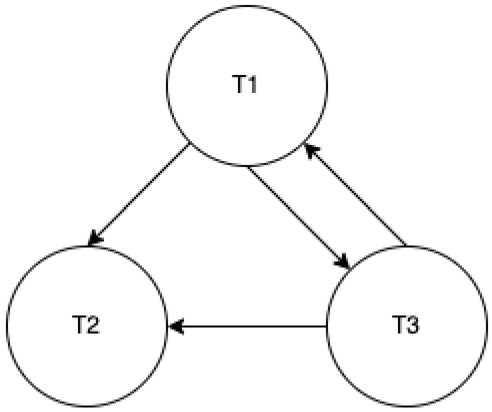
2.3.

Schedule 1:



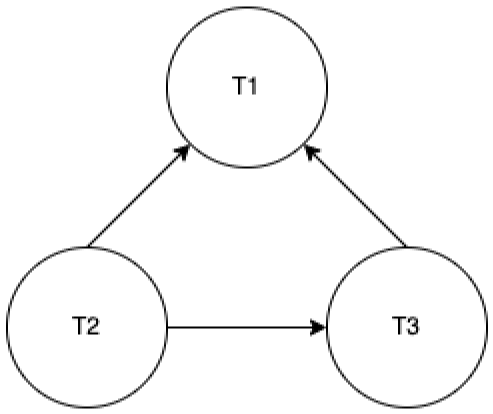
NOT conflict serializable, because there is a cycle in the precedence graph.

Schedule 2:



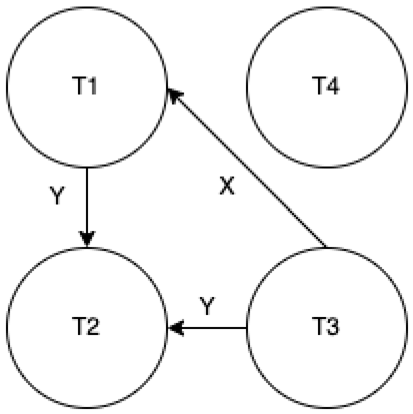
NOT conflict serializable, because there is a cycle in the precedence graph.

Schedule 3:



Conflict serializable.
Equivalent serial schedule:
T2 READ(X)
T3 READ(X)
T3 WRITE(X)
T1 READ(X)
T1 WRITE(X)

4.



5.

Time	T1	T2
t1	begin_transaction	
t2	write_lock(x)	begin_transaction
t3	write(x)	write_lock(y)
t4	write_lock(y)	write(y)

t5	WAIT…	write_lock(x)
t6	WAIT…	WAIT…

At t2, T1 acquire the write lock of x;
At t3, T2 acquire the write lock of y;
At t4, T1 tries to acquire write lock of y which is owned by T2, T1 has to wait;
At t5, T2 tries to acquire write lock of x which is owned by T1, T2 has to wait;
A t5, T1 and T2 are both waiting, which is the deadlock.

6.
The basic timestamping ordering (BTO) algorithm is to assign a unique timestamp to each transaction, with older transactions having smaller timestamp and younger transactions having bigger timestamp. The conflict is resolved by rolling back or restarting the older transactions because older transactions get less priority in the conflict. It ensures that the timestamp for the transactions determines the order that the conflicting operations are completed (conflict serializable).

Thomas' write rule (TW) algorithm provides an optimization for dealing with stale updates of a data object. TW identifies when stale updates can be safely ignored and not completed.

The main difference is in the write(x) function. In BTO, if ts(T) < write_timestamp(x), we will roll back the transaction and restart it using a later timestamp. However, in TW and the same situation, we will ignore the write based on the ignore obsolete write Thomas' write rule.

7.

TIME	
1	ts(B) = 21; read_timestamp(Z) = 21
2	read_timestamp(Y) = 21
3	write_timestamp(Y) = 21
4	ts(C) = 22; read_timestamp(Y) = 22

5	read_timestamp(Z) = 22
6	ts(A) = 23; read_timestamp(X) = 23
7	write_timestamp(X) = 23
8	write_timestamp(Y) = 22
9	write_timestamp(Z) = 22
10	**ts(B) < write_timestamp(X); Transaction B must be aborted and restarted with ts(B) = 24;** read_timestamp(X) = 24
11	read_timestamp(Y) = 23
12	write_timestamp(Y) = 23
13	write_timestamp(X) = 24

8.
According to the log:
Changes made by T1 have been written to secondary storage.
T2 and T3 are still active at time of crash, so they need to be **rollbacked** and the operations need to be **undone**.
T4 committed since the checkpoint, so its operations need to be **redone**.
There is no cascading rollbacks because although T2 reads D, which is previously written by T4, T4 committits before T2 reads D. So, it is a cascadeless schedule.

9.
Cascading rollback is when a single transaction failure leads to a series of transactions being restarted. We can still recover if the other transactions that read the uncommitted data have not yet committed.

Example of a schedule with a cascade rollrock:

TIME	Transaction 1	Transaction 2	Transaction 3
t1	write(x)		

t2	write(y)	read(x)	
t3		x = x + 1	read(y)
t4	**Rollback**		
t5		write(x)	y = y + 1
t6		Commit	write(y)
t7			Commit

T1 rollbacks at t4, so the values of x and y are not correct in T2 and T3. We need to remove the results of T1 and restart T2, T3.

REFERENCE

The T6_L6_Timestamp.pdf and T6_L2_Safe_Schedules-1.pdf slides on Canvas Files