Northeastern University

CS 5200 Homework 10 (Chapter 12 Connolly & Begg)
Each question is worth 10 points each.

1. Given the following 2 different transactions. List all potential schedules for T1 and T2 and determine which schedules are conflict serializable and which are not. (10 POINTS)

Transaction 1	Transaction 2
READ(X)	READ(X)
X = X -N;	X = X +M
WRITE(X)	WRITE(X)
READ(Y)	
Y = Y + N	
WRITE(Y)	

Answers 1:

Calculating all the potential solutions for the T1 and T2 schedules
1 mean for Transaction 1 , 2 mean for Transaction 2

T1: r1(X); w1(X); r1(Y): r2(Y);

T2: r2(X); w2(X);

In the above case
Transactions are two (2) , First transaction has 4 operations , Second transaction has 2 operations
N0 of possible outcomes a of the schedule is (4+2)!/(4!*2!)
=6 x 5 x 4 x 3 x 2 x 1/(4 x 3 x 2 x 1 x 2 x 1)
=720/48
=15
So totally 15 possible schedules are there

Below are the detailed information about the serializability of the 15

Conflict serializable :

S1:r1(X);w1(X);r1(Y);w1(Y);r2(X);w2(X);

S2:r1(X);w1(X);r1(Y);r2(X);w1(Y);w2(X);

S3:r1(X);w1(X);r1(Y);r2(X);w2(X);w1(Y);

S4:r1(X);w1(X);r2(X);r1(Y);w1(Y);w2(X);

S5:r1(X);w1(X);r2(X);r1(Y);w2(X);w1(Y);

S6:r1(X);w1(X);r2(X);w2(X);r1(Y);w1(Y);

S15:r2(X);w2(X);r1(X);w1(X);r1(Y);w1(Y);

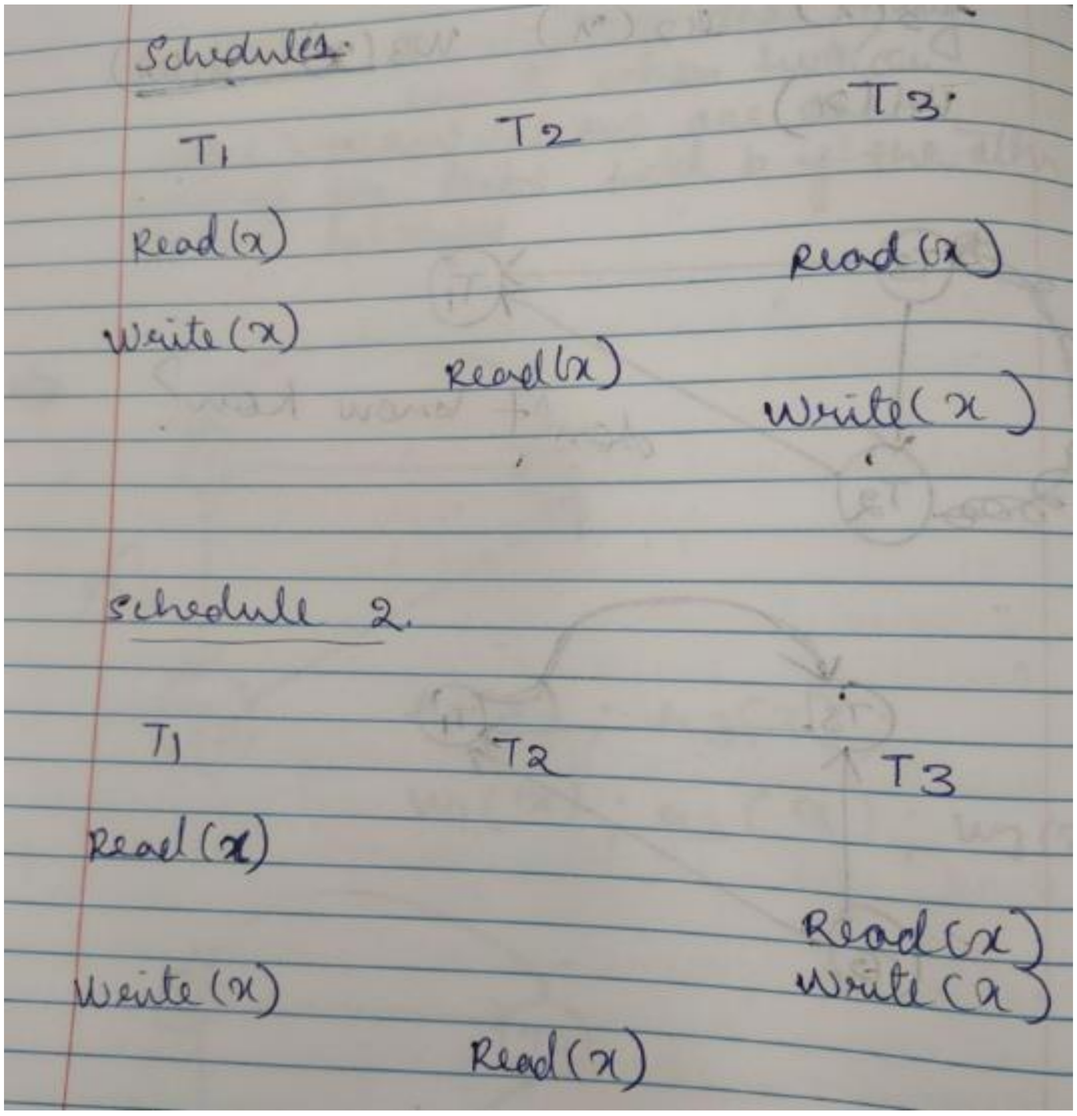
Not conflict serializable :

S7:r1(X);r2(X);w1(X);r1(Y);w1(Y);w2(X);
S8:r1(X);r2(X);w1(X);r1(Y);w2(X);w1(Y);
S9:r1(X);r2(X);w1(X);w2(X);r1(Y);w1(Y);
S10:r1(X);r2(X);w2(X);w1(X);r1(Y);w1(Y);
S11:r2(X);r1(X);w1(X);r1(Y);w1(Y);w2(X);
S12:r2(X);r1(X);w1(X);r1(Y);w2(X);w1(Y);
S13:r2(X);r1(X);w1(X);w2(X);r1(Y);w1(Y);
S14:r2(X);r1(X);w2(X);w1(X);r1(Y);w1(Y);

2.Which of the following schedules is conflict serializable? For each serializable schedule determine the equivalent serial schedule. (10 POINTS)

Schedule 1	Schedule 2	Schedule 3
T1 Read(X)	T1 Read(X)	T3 Read(X)
T3 Read(X)	T3 Read(X)	T2 Read(X)
T1 Write(X)	T3 Write(X)	T3 Write(X)
T2 Read(X)	T1 Write(X)	T1 Read(X)
T3 Write(X)	T2 Read(X)	T1 Write(X)

Answers 2:
The above three schedules can be described in the following ways:

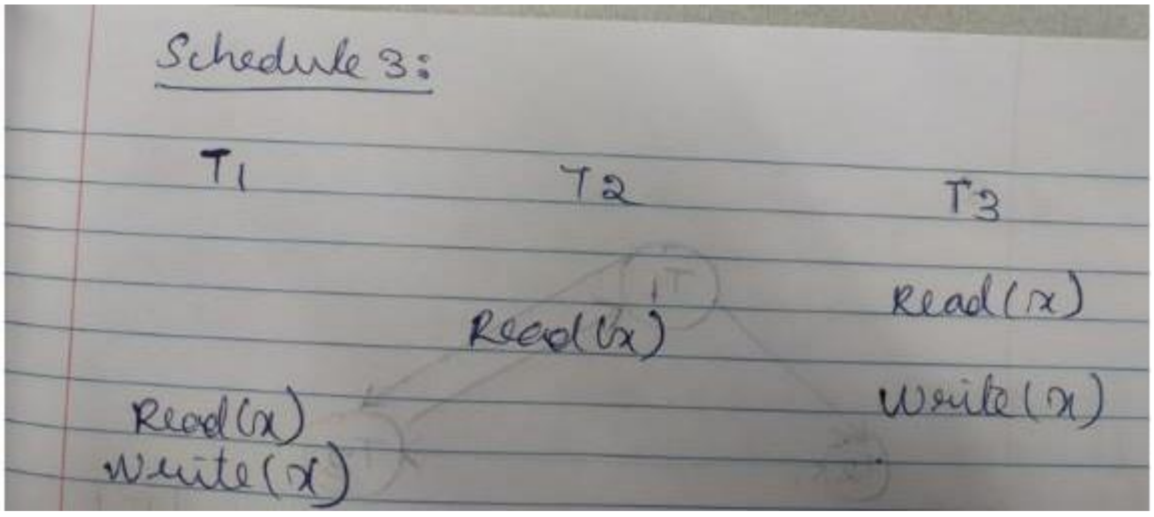


Schedules:

T_1	T_2	T_3
Read(x)		
		Read(x)
Write(x)		
	Read(x)	
		Write(x)

Schedule 2.

T_1	T_2	T_3
Read(x)		
		Read(x)
		Write(x)
Write(x)		
	Read(x)	



Schedule 3:

T_1	T_2	T_3
		Read(x)
	Read(x)	
		Write(x)
Read(x)		
Write(x)		

Schedule 1: Precedence is there so; it is no conflict serializable
Schedule 2: Precedence is there so it’s no conflict serializable
Schedule 3: Precedence is not there so its conflict serializable.

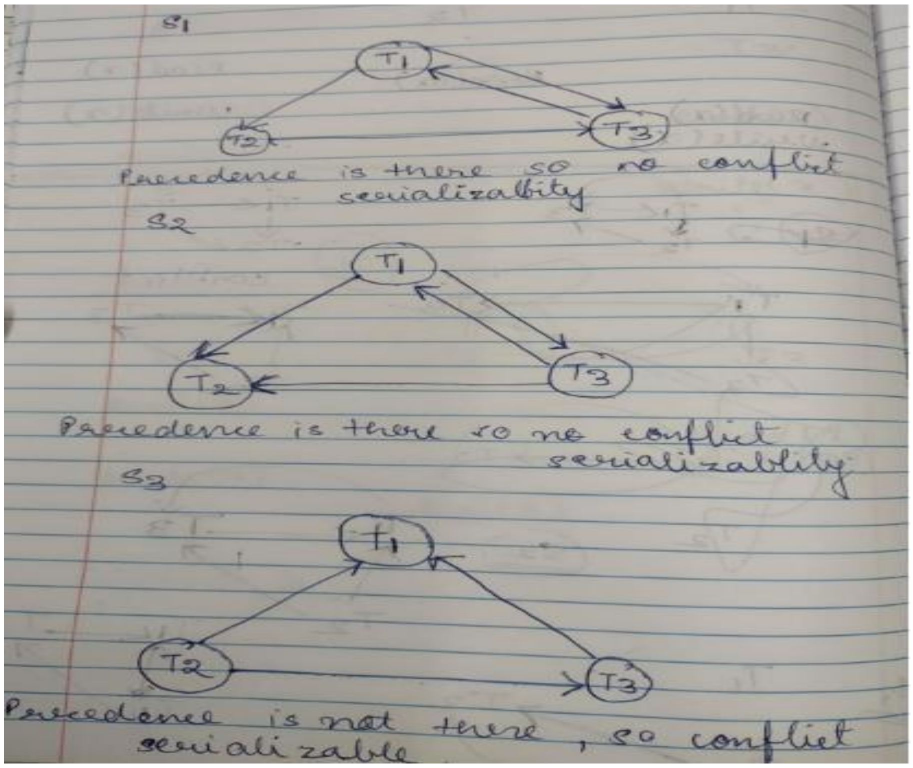
The equivalent serializability of the of the Schedule 3 is as given below:

$$\begin{tikzcd} T2 \arrow[r] & T3 \arrow[r] & T1 \end{tikzcd}$$ and is defined as below:

T1	T2	T3
	Read(x)	
		Read(x)
		Write(x)
Read(x)		
Write(x)		

3 .Draw the precedence graph for the 3 schedules in problem 2. (20 POINTS)

Answers 3: The precedence graph for the above problem is as given bel



Precedence is there so no conflict serializalbity

Precedence is there so no conflict serializablity

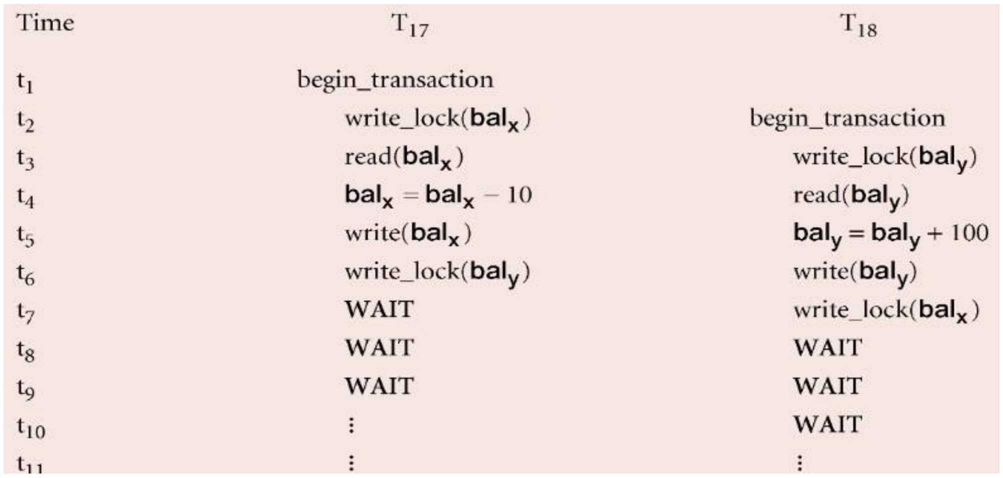
Precedence is not there, so conflict serializable

4.Provide a schedule that exhibits the deadlock problem. Describe the issue. (10 POINTS) (reference from the text book)

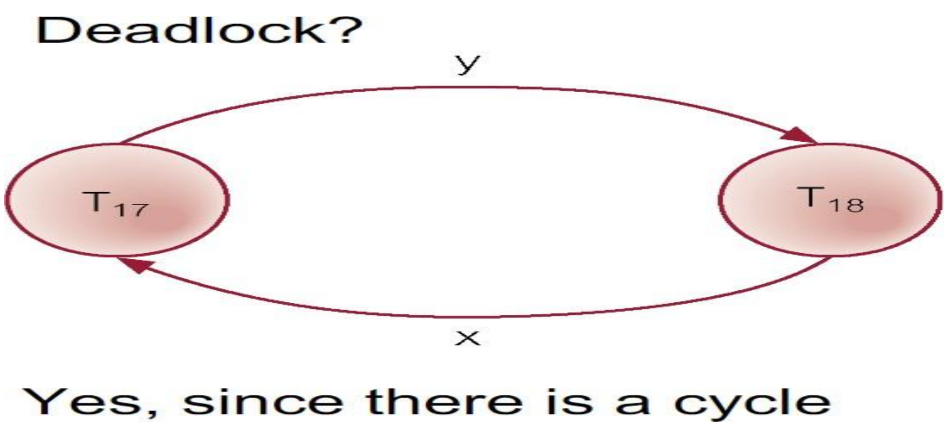
Answer 4: Deadlock is an impasse that may result when two (or more) transactions are each waiting for locks held by the other to be released.

E.g., 1., of the deadlock problem:

Below is the transaction table of the schedules. The issue is the transaction cycle gets repeated here. As the transaction T17 and T18 are waiting for each other to get their locks that are held by one another to get released. Hence, it's a deadlock problem.

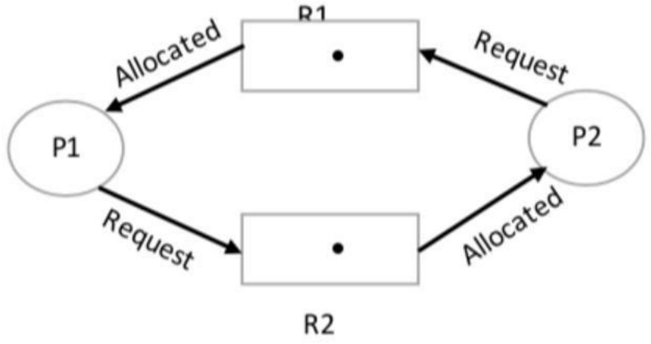


Time	T_{17}	T_{18}
t_1	begin_transaction	
t_2	write_lock($\mathbf{bal_x}$)	begin_transaction
t_3	read($\mathbf{bal_x}$)	write_lock($\mathbf{bal_y}$)
t_4	$\mathbf{bal_x} = \mathbf{bal_x} - 10$	read($\mathbf{bal_y}$)
t_5	write($\mathbf{bal_x}$)	$\mathbf{bal_y} = \mathbf{bal_y} + 100$
t_6	write_lock($\mathbf{bal_y}$)	write($\mathbf{bal_y}$)
t_7	WAIT	write_lock($\mathbf{bal_x}$)
t_8	WAIT	WAIT
t_9	WAIT	WAIT
t_{10}	⋮	WAIT
t_{11}	⋮	⋮



Example 2: When two trains approach each other at a crossing, both shall come to a full stop, and neither shall start up again until the other has gone.

.



Here the issue is if two or more processes are waiting for some events to happen, which never happens, then, hence the schedules are in deadlock state.
Here, P1 and P2 are two processes. R1 and R2 are two resources.
Note: R –> P means process (P) has taken the resource (R). For example, R2 –> P2 means resource R2 has been allocated to process P2.
P –> R means process (P) is requesting for resource (R). For example, P1 –> R2 means Process P1 is requesting for resource R2.
In this figure, there are two resources R1 and R2 and two processes P1 and P2.
Let us say R1 is allocated to P1 or P1 has taken R1. P1 is requesting for R2 but R2 is held by P2. Now, P2 is requesting for R1. So, we can say that it is kind of deadlock, or it is a deadlock situation.

Necessary conditions for deadlock:

We can detect a deadlock via a wait-for graph

Mutual Exclusion: A process should work in mutual exclusive way. It means a resource is non-sharable. At a time only one process can use the resource. If another process requests to use resource, the requesting process must be delayed until the resource has been released.
Hold and Wait: A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.
No Preemption: Once a process is holding a resource, then that resource cannot be taken away from the process until the process voluntarily releases it.
Circular Wait: A set of processes (P1, P2…..Pn) of waiting processes must exist such that P1 is waiting for resource that is held by P2, P2 is waiting for resource that is held by P3…Pn is waiting for a resource that is held by P1.

How to handle a deadlock:

1) We can prevent deadlocks by placing an ordering on the transactions and allowing the order to determine which transactions can wait (Wound-Wait) (Wait-Die).
2) Conservative two-phase locking (C2PL): A transaction must request all locks at the beginning of the transaction. If it does not receive all requested locks, it must wait until all locks can be granted.

3)Deadlock prevention or avoidance: Don't allow the system to get into a deadlock state.

4)Deadlock Detection and recovery: Let deadlock occur, then do preemption to handle it.

5)Ignore the problem all together: if deadlock occur once in a year or so, it may be better to let it happen and reboot the system. This is the approach that windows and Unix take.

5. Apply the basic timestamping algorithm to the following schedule. State if it can be performed as is or what transactions will need to be restarted given the basic timestamping ordering algorithm. (20 POINTS)

TIME	Transaction A	Transaction B	Transaction C
1		READ(Z)	
2		READ(Y)	
3		WRITE(Y)	
4			READ(Y)
5			READ(Z)
6	READ(X)		
7	WRITE(X)		
8			WRITE(Y)
9			WRITE(Z)
10		READ(X)	
11	READ(Y)		
12	WRITE(Y)		
13		WRITE(X)	

Answer 5: After applying the time stamping algorithm, at the 10 th step we need to restart it again bcz a conflict is faced at these points. Hence this will stop at 10 step and will need to call it again. Therefore, the Transaction A will work first, then transaction B and then transaction C will get executed. Hence we need to stop at 10th step, then transaction A gets executed after that , continuing transaction B gets executed and after that it rollback to the transaction C . This is how the operation of the Identified rollbacked transactions will take place.
Hence it can't be performed as it is, and it needs to restart at that point.

Read and write table :

	X	Y	Z
R	4	5	5
W	4	5	5

Time stamping Table:

	Ta	Tb	Tc
Ts	3	4	5

6.Below is a log corresponding to a particular schedule at the point of a system crash for 4 transactions T1, T2, T3, and T4. Suppose that we use the immediate update protocol with checkpointing. Describe the recovery process from the system crash. Specify which transactions are rolled back, which operations in the log are redone and which operations in the log are undone. State whether any cascading rollbacks take place. (20 POINTS)

Start TRANSACTION 1
T1 READ(A)
T1 READ(D)
T1 WRITE(D, 20, 25)
COMMIT T1
CHECKPOINT
Start TRANSACTION T2
T2 READ(B)
T2 WRITE(B 12, 18)
Start TRANSACTION T4
T4 READ(D)
T4 WRITE (D, 25, 15)
START TRANSACTION T3
T3 WRITE(C,30,40)

T4 READ(A)
T4 WRITE(A, 30, 20)
T4 COMMIT
T2 READ(D)
T2 WRITE(D,15,25)
SYSTEM CRASH

Answer 6 :

In this example:
Transactions T1, T4 are committed transactions Transactions T2, T3 are uncommitted transactions
In this protocol first log record is written on to the Log file and then modification will be done in that. Here within the protocol the immediate update protocol method is used for the checkpoints with checkpoint. If a system crash occurs, then committed transaction's operations are redone. Uncommitted transaction's operations are undone**.**

Transactions which are committed	**Transactions which are to be rolled back**	**The operations which are to Redone**	**The Operation which are to be Undone**
T1,T4	**T2,T3**	**[write_item,T1,D,20,25]** **[write_item,T4,D,25,15]** **[write_item,T4,A,30,20]**	**[write_item,T2,B,12,18]** **[write_item,T3,C,30,40]** **[write_item,T2,D,15,25]**

the uncommitted transaction becomes independent. That's how the recovery of the system crash takes place

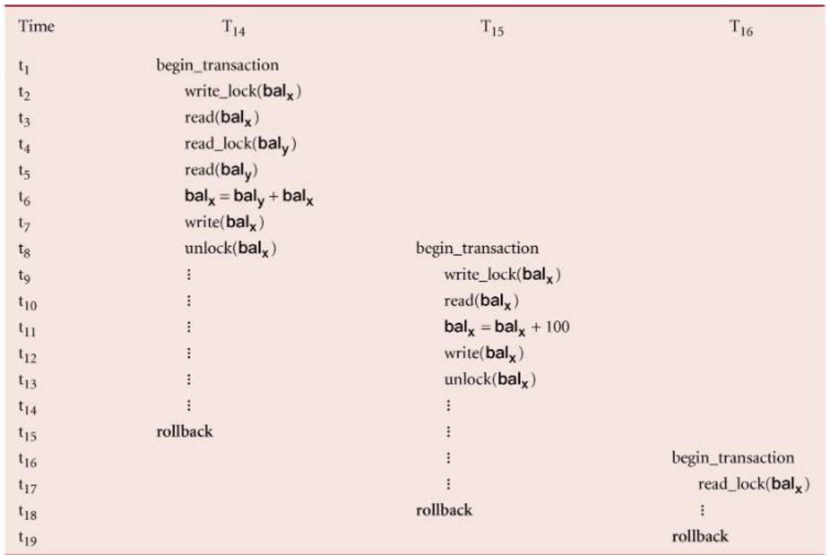
By Rollbacking of T3 no transaction get affected or changed. Transaction T2 using object D is modified by committed transaction of the T4, hence no cascading rollbacks possible here.

7.Describe what a cascading rollback is. (10 points) (Reference form the text book)

Answer 7 :

If every transaction in a schedule follows 2PL, schedule is
conflict serializable. However, problems can occur with interpretation of when
locks can be released. We may need to hold on to the locks longer to produce a recoverable schedule.
As shown in the below figure below are the example of the transaction T14 , T15 and T16 it
describes the rollback well **.**

2PL does not solve the problem



Time	T_{14}	T_{15}	T_{16}
t_1	begin_transaction		
t_2	write_lock($\mathbf{bal_x}$)		
t_3	read($\mathbf{bal_x}$)		
t_4	read_lock($\mathbf{bal_y}$)		
t_5	read($\mathbf{bal_y}$)		
t_6	$\mathbf{bal_x} = \mathbf{bal_y} + \mathbf{bal_x}$		
t_7	write($\mathbf{bal_x}$)		
t_8	unlock($\mathbf{bal_x}$)	begin_transaction	
t_9	⋮	write_lock($\mathbf{bal_x}$)	
t_{10}	⋮	read($\mathbf{bal_x}$)	
t_{11}	⋮	$\mathbf{bal_x} = \mathbf{bal_x} + 100$	
t_{12}	⋮	write($\mathbf{bal_x}$)	
t_{13}	⋮	unlock($\mathbf{bal_x}$)	
t_{14}	⋮	⋮	
t_{15}	rollback	⋮	
t_{16}		⋮	begin_transaction
t_{17}		⋮	read_lock($\mathbf{bal_x}$)
t_{18}		rollback	⋮
t_{19}			rollback

Transactions conform to 2PL.T14aborts. Since T15is dependent on T14, T15must also be rolled back. Since T16is dependent on T15, it too must be rolled back. This is called cascading rollback.

A recoverable schedule is classified as the cascading schedule. It commits operation of a particular transaction that performs read operation which is delayed until the uncommitted transaction either commits or roll backs.

A cascading rollback where if one transaction fails, then it will cause rollback of other dependent transactions. CPU time wastage is one of the disadvantages of the cascading rollback .

To prevent this with 2PL, leave release of all locks until end of transaction. This variation of 2PL is known as rigorous two-phase locking. Strict two-phase locking holds only the exclusive locks until the end of the transaction. The cascading rollback can be avoided using cascade less schedule and it also saves CPU time.