05 Transactions and Concurrency Control

1. (b)

S_1:

T_1	T_2	T_3
	$w(a)$	
$w(a)$		
		$w(c)$
$R(a)$		
		$R(b)$
	$w(b)$	
		$R(c)$

Precedence Graph

$$\begin{tikzcd} T_1 & T_2 \arrow[l] \\ T_3 \arrow[ur] & \end{tikzcd}$$

Serializability order:
$T_3 \to T_2 \to T_1$

S_2:

T_1	T_2	T_3
		$R(a)$
$R(b)$		
		$R(a)$
	$W(b)$	
$R(a)$		
	$R(a)$	
	$W(c)$	
		$R(c)$

Precedence Graph

$$\begin{tikzcd} T_1 \arrow[r] & T_2 \arrow[dl] \\ T_3 & \end{tikzcd}$$

Serializability order:
$T_1 \to T_2 \to T_3$

Since, we need to make the precedence graph as: $T_3 \to T_2 \to T_1$

So, if we swap ($R_1(b)$ and $W_2(b)$), there will be a dependency between T_2 to T_1 not T_1 to T_2.

Similarly if we swap ($W_2(c)$ and $R_3(c)$), there will be a dependency between T_3 to T_2 not T_2 to T_3.

Both the swaps will result in $T_3 \to T_2 \to T_1$.

2. (b)

T_1	T_2	T_3
$r(a)$		
	$r(c)$	
$r(c)$		
		$r(c)$
		$r(b)$
$w(a)$		
	$r(a)$	
commit		
		$w(b)$
		commit
	$r(b)$	
	$w(b)$	
	$w(c)$	
	commit	

$$\begin{tikzcd} T_1 \arrow[r] & T_2 \\ T_3 \arrow[ur] & \end{tikzcd}$$

Above precedence graph shows schedule is conflict serializable.

Since all the three transactions are writing the values after reading them, hence there is no Blind write in any of the three transactions.

Since transaction T_2 is reading the value of 'a' that is written by an uncommitted transaction T_1, hence there is a dirty read. Since, there is a dirty read, hence the schedule is cascadeless.

Since, there is no such dependency, such that a transaction reads the data item value written by an uncommitted transaction and commits. Hence, the schedule is recoverable.

3. (6)

$$\underbrace{\boxed{W_1(A)\, r_2(A)}\ \boxed{W_3(A)\, r_4(A)}\ \boxed{W_5(A)\, r_6(A)}}\ \boxed{W_7(A)\, r_8(A)}$$

These three blocks can execute in any order.

4. (0)

Blind-write: Write operation on a data item without having a read operation on the same data item before.

The following are 5 blind-writes: $W_1(A)$, $W_2(A)$, $W_1(D)$, $W_2(D)$, $W_3(E)$

$\therefore \qquad X = 5$

The precedency graph for above schedule S_1 is shown below:

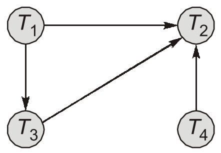
$$\begin{tikzcd} T_1 \arrow[r] & T_2 \arrow[l, bend right] \\ & T_3 \end{tikzcd}$$

Since cycle exist in the graph, the number of conflict serializable schedules are zero i.e. $Y = 0$.

$\therefore \qquad X \cdot Y = 0$

5. (c)

The precedence graph of the given schedule is

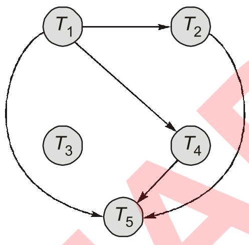


Therefore schedule is equivalent to
T_1, T_4, T_3, T_2 and T_1, T_3, T_4, T_2

6. (b)

In data base until check point not come data is not saved permanently, when checkpoint is comes all database until checkpoint all data stored permanently.
After checkpoint process which are committed are redo and which are not committed are undo:
So, undo is to be transaction T_3 redo is to be transaction T_4.

7. (a)



# of serial schedules conflict equal to schedule (S) is # of topological orders

$$T_1 \begin{matrix} T_2 - T_4 - T_5 \\ T_4 - T_2 - T_5 \end{matrix} \Bigg\} \text{ 2 sequences for } T_1\, T_2\, T_4\, T_5$$

T_3 can be any where in both sequences. Total 10 topological order.

8. (a)

Schedule (S)		(Equal serial)
Final write:	T_1 $\boxed{T_5}$	$T_1 \longrightarrow T_5$
Initial read	—	—
Updated read	$W_1(B) \rightarrow R_2(B)$ other write $W_5(B)$	$T_1 \xrightarrow[]{\overset{\text{X}}{(T_5)}} T_2$
	$W_1(B) \rightarrow R_4(B)$ other write $W_5(B)$	$T_1 \xrightarrow[]{\overset{\text{X}}{(T_5)}} T_4$
	$W_2(C) \rightarrow r_5(C)$	$T_2 \longrightarrow T_5$
	$W_4(E) \rightarrow r_5(E)$	$T_4 \longrightarrow T_5$

$$\underbrace{T_4 \quad ✓ \quad T_1 \times T_2 ✓ T_5 \times}$$

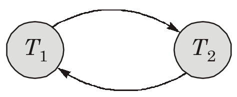
$\left.\begin{matrix} T_1 T_4 T_2 T_5 \\ T_1 T_2 T_4 T_5 \end{matrix}\right\}$ now for both ordering T_3 can be place anywhere so 10 view equal schedule.

9. (2)

- Only serial schedules $T_1 \rightarrow T_2$, $T_2 \rightarrow T_1$ are conflict serializable.
- None of non serial schedules are conflict serializable.

10. (2)

S_1:



Non serial schedule between T_1 and T_2 will not a conflict serializable schedule.

S_2:

T_1	T_2
Shared lock $S(x)$, $S(y)$, $S(z)$ $R_1(x)$ $R_1(y)$ $R_1(z)$ Exclusive lock not allowed on (y)	Shared lock $S(y)$, $S(x)$, $S(z)$ $R_1(y)$ $R_1(x)$ $R_1(z)$ Exclusive lock not allowed on (x)

Not allowed under 2PL protocol.

11. (b)

T_1	T_2	T_3
r_1(X)		
	w_2(X)	
	Commit	
w_1(X)		
		w_3(X)
Commit		Commit

Schedule is view serializable $T_1 \rightarrow T_2 \rightarrow T_3$.
The schedule is not strict because T_1 writer on (X) before T_3 so commit operation should occure before w_3(X) operation.

12. (12)

Conflict-equivalent to $T_1 \rightarrow T_2$:

T_1	T_2
R(A) W(A)	
	R(B) W(B)

Remaining 4 transactions can be arranged in any possible order.

Number of possibilities: $\dfrac{4!}{2! \times 2!} = 6$

Conflict-equivalent to $T_2 \rightarrow T_1$:

T_1	T_2
	R(A) W(A)
R(B) W(B)	

Remaining 4 transactions can be arranged in any possible order.

Number of possibilities: $\dfrac{4!}{2! \times 2!} = 6$

Total number of possibilities 6 + 6 = 12.

13. (30)

- Final write:

  On B: T_3 So, $\left\}\begin{pmatrix} T_1 \to T_2 \to T_3 \\ T_2 \to T_1 \to T_3 \end{pmatrix}\right.$
  On A: No one

- No write read pair present
- Initial read
  On A: T_1, T_2, T_3
  On B: No one

So, number of schedule possible: $T_2 \to T_1 \to T_3$ and $T_1 \to T_2 \to T_3$

Now check view equivalent to S:

1. $T_1 \to T_2 \to T_3$

(i)
$$\begin{tikzcd} & w_1(B) \arrow[dl] \arrow[dr] & \\ r_2(A) & & w_2(B)\ r_3(A) \end{tikzcd}$$
$w_3(B)$ = Here $r_1(A)$ will come in 3 ways

(ii)
$$\begin{tikzcd} w_1 \arrow[d] \arrow[dr] \arrow[drr] & & \\ r_2(A) & r_3(A) & w_2(B) \end{tikzcd}$$
$w_3(B)$ = Here $r_1(A)$ will come in 6 ways

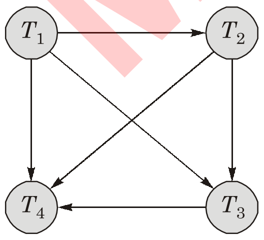
(iii)
$$\begin{tikzcd} w_1(B) \arrow[d] \arrow[dr] \arrow[drr] & & \\ r_3(A) & r_2(A) & w_2(B) \end{tikzcd}$$
$w_3(B)$ = Here $r_1(A)$ will come in 6 ways

Total ways = 3 + 6 + 6
= 15 ways

Similarly for $T_2 \to T_1 \to T_3$ 15 ways are possible.
So total number of view equivalent schedules are 15 + 15 = 30.

14. (c)

Number of conflict equal serial schedule.



1 topological order $T_1 : T_2 : T_3 : T_4$
[1 serial schedule equal to conflict equal to S]
Number of view equal serial schedule

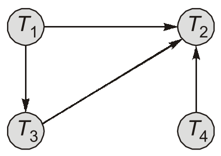
$$\underbrace{[T_1 T_2 T_3]}_{3! = 6} \to T_4$$
$= 6 - 1 = 5$

15. (b)

In data base until check point not come data is not saved permanently, when checkpoint is comes all data till checkpoint get stored permanently.
After checkpoint process which are committed are redo and which are not committed are undo:
So, undo is to be transaction T_3 redo is to be transaction T_4.

16. (c)

The precedence graph of the given schedule is



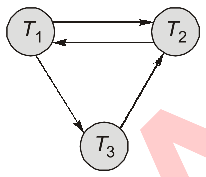
Therefore schedule is equivalent to (T_1, T_4, T_3, T_2), (T_1, T_3, T_4, T_2) and (T_4, T_3, T_1, T_2).

17. (d)

The only constraint on a serial order so far is that T_1 must precede T_2, because T_1 reads A before T_2 writes A. The only way we could get a cycle in the precedence graph is if blank space required T_2 to precede T_1. A read of anything by T_1 will not introduce any constraints, but $R_2(C)$ will cause an arc from T_2 and T_1 and lead to a cycle.

18. (c)

S is not serializable. The precedence graph of S is



Since there is a cycle in its precedence graph, therefore, it is not conflict serializable. S is not a serial schedule.

19. (b)

Since, there is a cycle in the precedence graph, so it is not conflict serializable. But there is a presence of blind write ($W_2(A)$), therefore we need to check for view serializability.

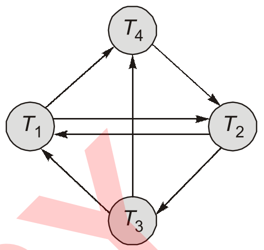
Initial reads: Initial reads of data item B, done by transaction T_2 and A is done by transaction T_1, but T_2 has already written A before it was read therefore, $T_2 \to T_1$.

Final writes: The last writer of data item A and B are T_2 and T_3 respectively.

Based on the above inference, the above schedule is view serializable and view equivalent order of transaction will be $T_2 \to T_1 \to T_3$.

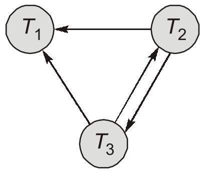
20. (0)

For S_1, the precedence graph will be



Since there is a cycle in the graph, therefore, it is not conflict serializable.

For S_2, the precedence graph will be



Since, there is a cycle in the precedence graph, therefore, it is not conflict serializable.

Therefore, the answer will be 0.

21. (a)

Statement-II is false because they are independent.

22. (a)

For a conflict serializable schedule, the linear ordering of transactions corresponds to topological sorting of the serialization graph. Since there can be multiple such orderings, multiple equivalent serial schedule may exist.

23. (54)

$$\text{Total number of schedules possible} = \frac{(4+4)!}{4!\,4!}$$

$= 70$

Following two conflict actions are possible:

T_1	T_2
$r_1(y)$	
	$w_2(y)$
$w_1(y)$	

	T_1	T_2	
$\frac{4!}{2!\,1!} = 12$	$r_1(x)$		
	$w_1(x)$	$r_2(y)$	
	$r_1(y)$		
		$\boxed{w_2(y)}$	Assume
$\frac{2!}{1!\,2!} = 1$	$w_1(y)$	$r_2(z)$	
		$w_2(z)$	

Number of permutation = 12 × 1 = 12

T_1	T_2
	$r_2(y)$
$w_1(y)$	
	$w_2(y)$

	T_1	T_2	
	$r_1(x)$		$\frac{4!}{3!\,1!} = 4$
	$w_1(x)$	$r_2(y)$	
	$r_1(y)$		
$\rightarrow$	$\boxed{w_1(y)}$		
		$w_2(y)$	$\frac{3!}{3!} = 1$
		$r_2(z)$	
		$w_2(z)$	

Number of permutation = 4 × 1 = 4
Total number of conflict serial schedules possible = 70 – 12 – 4 = 54.

24. (b)

The consistency property ensures that the database remains in a consistent state before the start of the transaction and after the transaction is over. Here sum of the accounts x and y should remain same before and after execution of the given transactions which refers to the consistency of the sum.

25. (c)

The schedule S_1 is conflict serializable while schedule S_2 is not conflict serializable due to occurrence of cycle in between transaction T_1 and T_2. Therefore, statement (1) and (2) are correct. Since, schedule S_2 is not conflict serializable, therefore, it cannot have conflict equivalent serial schedule. Thus, the statement given in option (c) is not correct.
Hence, the correct option is (c).

26. (c)

Deletion of $R_2(B)$ at time stamp 5:
It will lead to a non conflict serializable schedule as $T_1 \rightarrow T_2$ and $T_2 \rightarrow T_1$ (due to $W_2(B) \rightarrow R_1(B)$).

But it will be cascadeless as there will be no uncommitted reads if we remove $R_1(B)$ at timestamp 5.

27. (0)

I. The schedule S is having cycle in this precedence graph. Hence, it is not conflict serializable but it is view serializable.
II. It is allowed in Thomas write rule as it will ignore the $W_2(A)$ (as at $W_2(A)$ current timestamp will be 1 and WTS(A) is 2. Since 2 > 1, TWR will ignore this but it will not ignore $W_3(B)$ and $R_1(B)$. Hence, it is not allowed.

28. (33)

If there are n transaction with n_1, n_2 .... n_n operations respectively then number of concurrent schedules

$$= \frac{(n_1 + n_2 + n_3 + ... n_n)!}{n_1! \times n_2! \; .... \times n_n!}$$

Number of concurrent schedule = $\frac{(3+4)!}{3! \times 4!} = 35$

Number of serial schedule = 2! = 2
Number of non serial schedules = 35 – 2 = 33

29. (c)

T_1	T_2	T_3
$r(A)$		
	$r(A)$	
$r(A)$ commit		
		$w(A)$
	commit	
		$r(A)$ commit

While preparing the precedence graph, we only consider the committed transactions.

Hence
$$\begin{tikzcd} T_1 \arrow[dr] & \\ & T_3 \end{tikzcd}$$

Hence, the schedule is conflict serializable. Since it is conflict serializable, hence view serializable too.

30. (2)

In S_2 : $w_3(y)$ is first and $r_2(y)$ appears second
Hence, c_2 should appear after c_3.
In recoverable [If T_j is reads a value written by T_j, the T_i must commit after T_j commits]
Schedules S_1 and S_3 are recoverable.

31. (d)

T_1	T_2	T_3
$r_1(A)$		
	$w_2(A)$	
	Commit2	
$w_1(A)$		
		$w_3(A)$
		Commit3
Commit1		

View serializable order = $T_1 \to T_2 \to T_3$

So, schedule is serializable schedule but not strict recoverable schedule, since T_1 ($w_1(A)$) to T_3 ($w_3(A)$) is voilated strict recoverable schedule because of commit order.

32. (a)

In S_1, every transaction commits right after it writes some items. There is no write to or read from an item before the last transaction that wrote that item has committed. So S_1 is strict. And cascadeless too.
In S_2, T_2 reads item Y from T_3 but T_2 commits before T_3 commits. So S_2 is non-recoverable. S_3 is not strict because T_2 writes Y before T_3 commits. But S_3 is cascadeless because there is no transaction reads items that were written by an uncommitted transaction.

33. (d)

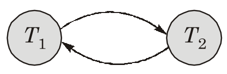
S_1 : All strict schedule may or may not serial i.e.

T_1	T_2
W(A)	
	W(B)
W(C)	
	W(D)
C_1	
	C_2

Schedule is strict schedule but not serial.

S_2 : All recoverable schedule need not be conflict serializable.

T_1	T_2
R(A)	
	W(A)
	R(B)
W(B)	
C_1	
	C_2



Since cycle exist, so cannot be conflict serializable.

S_3: All strict schedule need not be conflict serializable.

T_1	T_2
R(A)	
	W(A)
	W(B)
	C_2
R(B)	
C_1	

Schedule is strict schedule but not conflict serializable.

S_4: All conflict serializable schedules may not free from cascading rollbacks i.e.

T_1	T_2	T_3
		W(A)
R(A)		
W(A)		
	R(A)	
	W(A)	
C_1		
	C_2	
		C_3

- Schedule is conflict serializable i.e.

$$\begin{tikzcd} T_1 \arrow[rr, bend left] & & T_2 \\ & T_3 \arrow[ul, bend left] \arrow[ur, bend right] & \end{tikzcd}$$

no cycle exist.

- Schedule have cascading abort.

So, none of these statement is true.

34. (b)

T_1	T_2	T_3
	$S(A)$	
	$R_2(A)$	
$X(B)$		
$W_1(B)$		
$X(C)$		
$W(C)$		
$S(A)$		
$U(A)$		
		$S(B)$
		$R_3(B)$
	$S(B)$	
	$R_2(B)$	
$S(B)$		
$R(A)$		
$U(B)$		
C_1		
	$R_2(C)$	
	$U(A)U(B)$	
	C_2	
		$X(A)$
		$W_3(A)$
		$U(B)U(A)$
		C_3

(*i*) Conflict serializable

$$\begin{tikzcd} T_1 \arrow[rr] \arrow[dr] & & T_2 \arrow[dl] \\ & T_5 & \end{tikzcd}$$

(*ii*) Allowed by 2PL.
(*iii*) Not strict recoverable.
(*iv*) No allowed by strict 2PL.

35. (d)

Transaction of T_1 and T_2 has the written value of data item *B* and *C* respectively but that value is later read by T_3, therefore, T_3 must be rolled back. Transaction T_3 wrote *D*, but no transaction has read *D*, so no further rollbacks are needed.

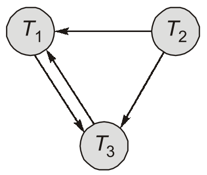
36. (a)

Since the schedule S does not have any read operation, therefore this schedule is recoverable schedule.

Yes, the given schedule is conflict serializable and cascadeless as there are no dirty reads.

37. (a)

I. Precedence graph of S



Since it results in a cycle due to uncommitted read of $R_1(Y)$ of value written by $W_3(Y)$, therefore, it is not conflict serializable.

II. T_2 reads *X* from T_1 which is not yet written, same for T_1. So it does not make any difference T_1, T_2 and T_3 reads *X* which is not yet amended. Therefore, it is recoverable and cascadeless.

38. (a)

When a transaction is writing a data item after reading it from an uncommitted transaction is nothing but a blind write. If transaction writing data commits then the transaction which read the data might get phantom tuple. Thus, this operation will not create any irrecoverable error. Therefore this statement is not true.

39. (a)

T_3 reads item *X* after the T_1 writes *X*. Thus, first T_3 will be rolled back followed by T_2 because later T_2 reads item *X* which T_3 has written.

40. (d)

I. This schedule is not a strict schedule due to W-W conflict ($W_3(Y) \rightarrow W_2(Y)$) but it is a cascadeless as there are no dirty reads. Therefore this statement is false.

II. This schedule is cascadeless as there is no presence of uncommitted read but it is not conflict serializable because there is the presence of cycle in its precedence graph.

41. (b)

In the transaction T_2, $R_2(a)$ is after $W_1(a)$ and $R_1(b)$ is after $W_1(b)$, so the schedule will remain non-recoverable even after changing the order of commits. Hence, the second statement is false.

42. (d)

- Since the schedules are non-serial, hence, every interleaving between T_1 and T_2 will led to a dependency between then in the graph. Hence none of the schedule will be conflict serializable.
-

T_1	T_2

S(z), S(x), S(y)
r(x)
r(z)
r(y)
Exclusives lock on z not possible
w(z)
	S(z), S(x), S(y)
	r(z)
	r(x)
	r(y)
	Exclusives lock on x not possible
	w(x)

- No non-serial schedule between T_1 and T_2 is allowed under 2PL protocol. Because, both the transaction are reading on the data item, that is written by other transaction as a last operation.
- Even if we try for upgradation of the locks, that is also not possible because, other transaction has already locked that data item is shared mode.

43. (c)

Check for S_1:

T_1	T_2	T_3
X(A)		
r(A)		
w(A)		
	X(B)	
	w(B)	
		X(C)
		w(C)
		u(C)
S(D)		
r(D)		
u(D)		
u(A)		
	X(D)	
	w(D)	
	u(D)	
	u(B)	

So S_1 is allowed under 2PL.

Check for S_2:

Making precedence graph:

$$\begin{tikzcd}
T_1 \arrow[r] & T_2 \\
T_4 \arrow[r] & T_3 \arrow[u] \arrow[ul]
\end{tikzcd}$$

No cycle in precedence graph.

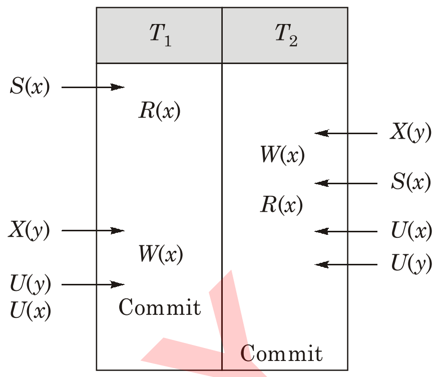
S_2 is allowed under 2PL.

44. (d)

- 2PL guarantee schedule will be conflict serializable but reverse is false.
- Strict 2PL schedule guarantee to prevent cascading aborts but not Basic 2PL, so false.
- Under multi granularity locking Locks are acquired top down, but released bottom up, so false.
- Wound wait and wait die algorithms are pessimistic deadlock avoidance algorithms and can cause more transaction aborts than needed, which is true.

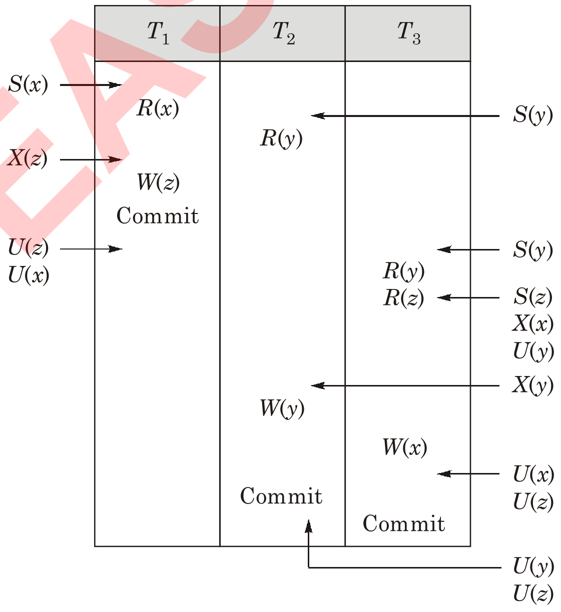
45. (a)

- Check S_1:



Schedule S_1 is allowed under 2PL.

- Check S_2:



Schedule S_2 is allowed under 2PL.

46. (d)

The given locking protocol follows the properties of strict 2PL which is conflict serializable, recoverable and avoid cascading rollbacks.

47. (d)

2PL ensures conflict serializability but may lead to deadlock.
Time-stamp concurrency control algorithm is a non-lock concurrency control method. In time-stamp based method, deadlock cannot occur as no transaction ever waits

48. (b)

First, all six shared-lock request are granted. When T_1 request an exclusive lock on A, it gets it, because it is the only transaction holding a lock on A. T_1 completes and releases its locks. Thus, when T_2 asks for an exclusive lock on B, it is the only transaction still holding a shared lock on B, so that lock too may be granted, and T_2 completes. After T_2 releases its locks, T_3 is able to upgrade its shared lock on C to exclusive, and it too proceeds. The actions are thus executed in exactly the same order as they are requested i.e., $T_1 \rightarrow T_2 \rightarrow T_3$, thus statement I is false. Since, the actions are executed in the same order of their request, therefore, there are no delays by the scheduler. Thus, statement II is true.

49. (c)

2PL protocol only ensures/guarantees serializability. Hence, a schedule having 2PL protocol could have both deadlock as well as irrecoverability. Therefore this statement is false.

50. (2)

Suppose the schedule starts with T_1 locking and reading A. If T_2 locks B before T_1 reaches its unlocking phase, then there is a deadlock and the schedule cannot complete. Thus, if T_1 performs an action first, it must perform all its actions before T_2 performs any. Likewise, if T_2 starts first, it must complete before T_1 starts or there is a deadlock. Thus, only the two serial schedules of these transactions are legal.

51. (a)

In conservative 2PL protocol, a transaction has to lock all the items before the transaction begins execution.
Advantages of conservative 2 PLP:
- No possibility of deadlock.
- Ensure serializability.

The given scenario (Step 1, Step 2 and Step 3) is conservative 2 PLP.

52. (a)

It would make no difference because the write protocol is such that the most recent transaction to write an item is also the one with the largest time-stamp to have done so in basic time-stamp ordering.
Hence both the definition will give same result in all the cases.

53. (c)

T_1	T_2	T_3
$r(x)$		
	$r(y)$	
		$r(y)$
$w(x)$		
		$w(x)$
	$r(z)$	
$w(x)$		

The above schedule is not allowed under any of the basic time-stamp protocol as well as Thomas write.

54. (d)

Schedule is not allowed under both basic time stamp and Thomas Write rule.

55. (a)

In wait-die scheme, when transaction T_i request a data items currently held by T_j, T_i is allowed to wait only if it has a time stamp smaller then that of T_j otherwise T_i is rolled back (die). Here process P is running so it has time stamp less than process Q now if process P need a resource held by process Q then process P has to wait.

56. (b)

T_1	T_2	T_3
$r(x)$		
	$r(y)$	
		$r(y)$
$w(x)$		
		$w(x)$
	$r(z)$	
$w(x)$		

The above schedule is not allowed under the basic time-stamp protocol but allowed under Thomas write.

57. (d)

Let us consider each data separately:

- On considering data item x : data item x is read by T_1 and finally written by transactions T_2, therefore $T_2 > T_1$.
- On considering data item y : data item y is read by T_3 and finally written by transaction T_2, therefore $T_2 > T_3$.
- On considering data item z : data item z is first read by T_3 and finally written by T_1, therefore $T_1 > T_3$.

From above analysis, we can say that time stamp of $T_3 < T_1 < T_2$. Thus, only option which satisfies is option (d).

58. (c)

Here, T_1 should happen before T_3 due to $W_1(C) > R_3(C)$ (i.e., $T_1 > T_3$)

T_3 should happen before T_2 due to $R_3(B) > W_2(B)$ (i.e., $T_3 > T_2$) but according to the given time stamp, T_2 is scheduled before T_3. Hence, it will roll back. Therefore, this schedule is not allowed under TSP. Since it is R-W conflict therefore, it is not even allowed under TWR.

59. (a)

In the question, it is given that 'unique time stamps' are assigned to the transactions. So, there will be no equal timestamps.

Lamport's logical clock:

In this timestamp are assigned in increasing order only.

According to the given algorithm,

$TS(T_2) < TS(T_1)$

then T_1 is killed,

else

T_2 will wait

So, in both the cases, it will be deadlock free and there will be no starvation.

60. (d)

	Irrecoverable	Recoverable	Deadlock	Deadlock free
Strict 2 PL	No	Yes	Yes	No
Time stamp ordering protocol	Yes	No	No	Yes
2 PL Protocol	Yes	No	Yes	No
Conservative 2 PL Protocol	No	Yes	No	Yes

61. (b)

It is allowed in TWR as it will ignore the $W_2(A)$ as at $W_2(A)$ current timestamp will be 1 and WTS(A) is 2, since 2 > 1, TWR will ignore this. For $W_1(A)$, it is according to the given timestamp only. Hence it will be allowed (3 > 2). Hence, the given schedule is allowed under TWR but not in TSP.

■■■■