COMPUTER SCIENCE

Database Management System

Transaction & Concurrency Control

Lock Based Protocol- (Part-02)

Lecture_9

Vijay Agarwal sir

TOPICS TO BE COVERED

01 Lock Based Protocol

02 Time Stamp Protocol

Transaction Concept

ACID Properties.

Schedule

- Serial Schedule
- Non serial Schedule

Serializable Schedule

- Conflict Serializable
- View Serializable.

- Serializablity
  - Conflict Serializable
  - View Serializable
- Recoverablity
  - Recoverable
  - Cascadeless
  - Strict Recoverable

SCHEDULE

Q.

Consider the following database schedule with two transactions, T_1 and T_2.

$S = r_2(X); r_1(X); r_2(Y); w_1(X); r_1(Y); w_2(X); a_1; a_2$

where $r_i(Z)$ denotes a read operation by transaction T_i on a variable Z, $w_i(Z)$ denotes a write operation by T_i on a variable Z and a_i denotes an abort by transaction T_i

Which one of the following statements about the above schedule is TRUE?

[MCQ:2016–2M]

(A) S is non-recoverable

(B) S is recoverable, but has a cascading abort

(C) S does not have a cascading abort

(D) S is strict

Q. Let S be the following schedule of operations of three transactions T_1, T_2 and T_3 in a relational database system:

$$R_2(Y), R_1(X), R_3(Z), R_1(Y), W_1(X), R_2(Z), W_2(Y), R_3(X), W_3(Z)$$

Consider the statements P and Q below:

P: S is conflict-serializable.

Q: If T_3 commits before T_1 finishes, then S is recoverable.

Which one of the following choices is correct?

[MCQ: 2021-2M]

(A) Both P and Q are true.

(B) P is true and Q is false.

(C) P is false and Q is true.

(D) Both P and Q are false.

Q.

Consider a simple checkpointing protocol and the following set of operations in the log.

(start, T4); (write, T4, y, 2, 3); (start, Tl);

(commit, T4); (write, T1, z, 5, 7);

(checkpoint);

(start, T2); (write, T2, x, 1, 9); (commit, T2);

(start, T3); (write, T3, z, 7, 2);

If a crash happens now and the system tries to recover using both undo and redo operations, what are the contents of the undo list and the redo list

[MCQ: 2015–2M]

A. Undo: T3, T1; Redo: T2

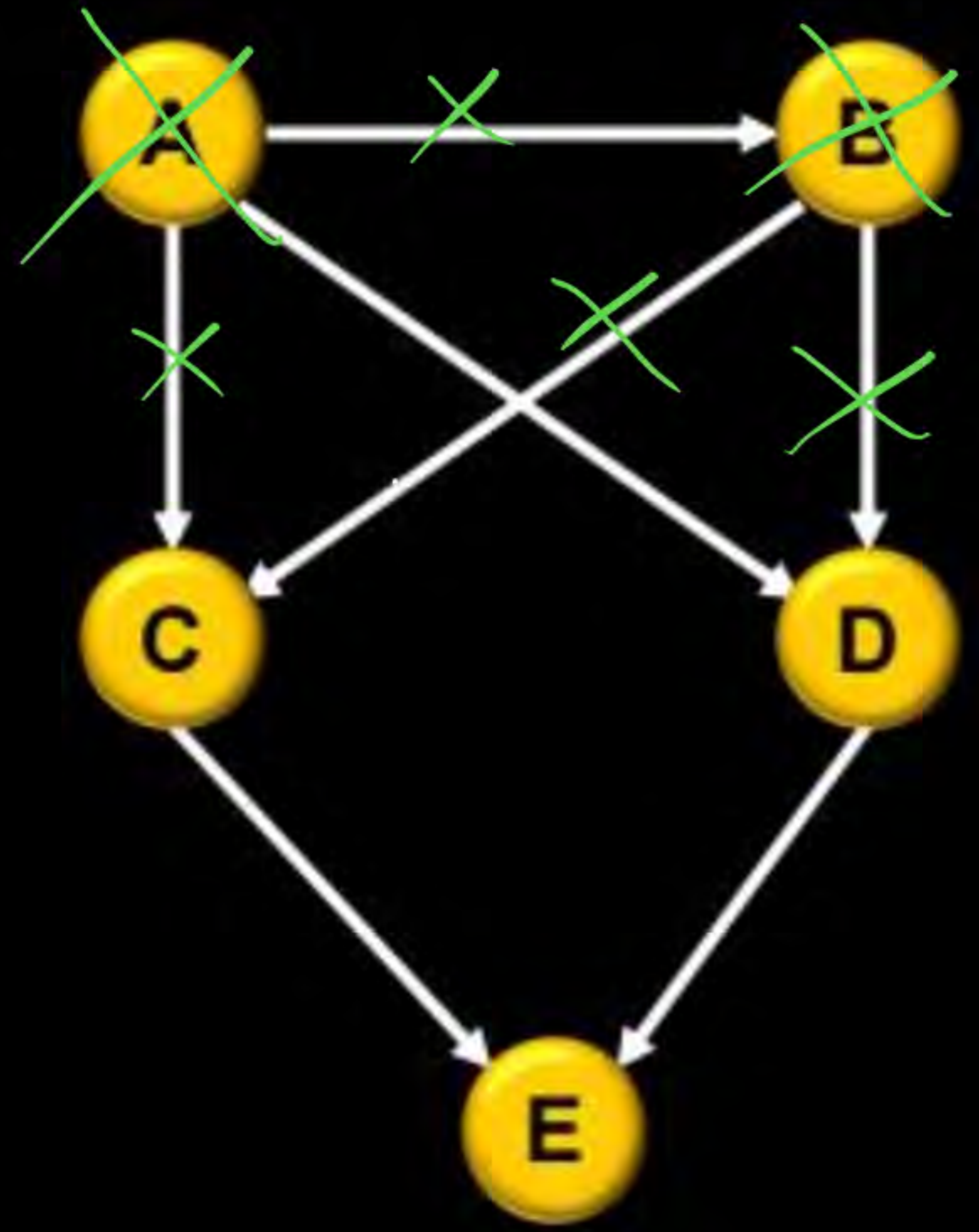
B. Undo: T3, T1; Redo: T2, T4

C. Undo: none; Redo: T2, T4, T3, Tl

D. Undo: T3, Tl, T4; Redo: T2

Topological Sorting

Q.1



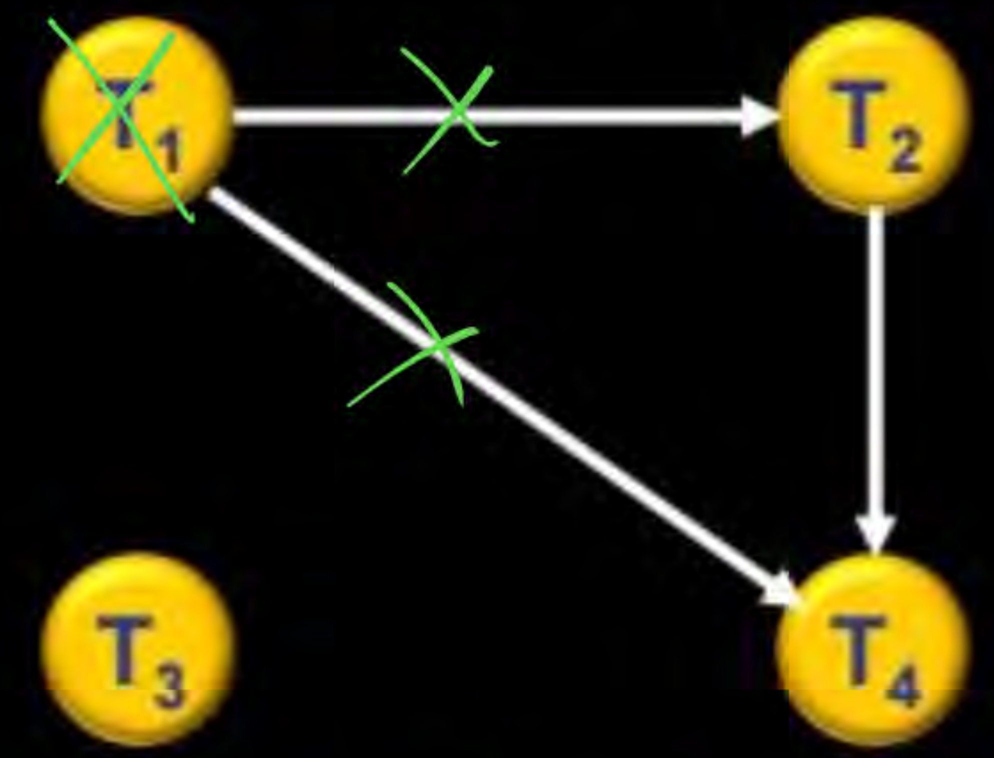
Ans1 2.

A B C D E

A B D C E

Topological Sorting

Q.2



T_3 Can be Placed Any where

Ans 2

4 Ans

T_3 T_3 T_3 T_3

↑ T_1 ↑ T_2 ↑ T_4 ↑

T_3 T_1 T_2 T_4

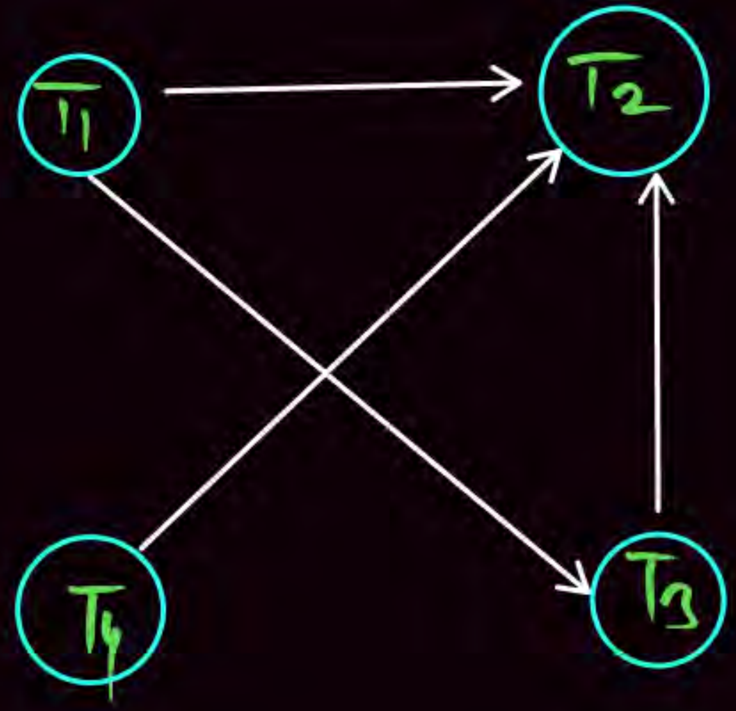
T_1 T_3 T_2 T_4

T_1 T_2 T_3 T_4

T_1 T_2 T_4 T_3

Q. 3

T_1	T_2	T_3	T_4
			R(A)
	R(A)		
		R(A)	
W(B)			
	W(A)		
		R(B)	
	W(B)		



3 serializability order.

$\langle T_1\ T_4\ T_3\ T_2 \rangle$

$\langle T_1\ T_3\ T_4\ T_2 \rangle$

$\langle T_4\ T_1\ T_3\ T_2 \rangle$

(ii) alternate Approach

Indegree = 0

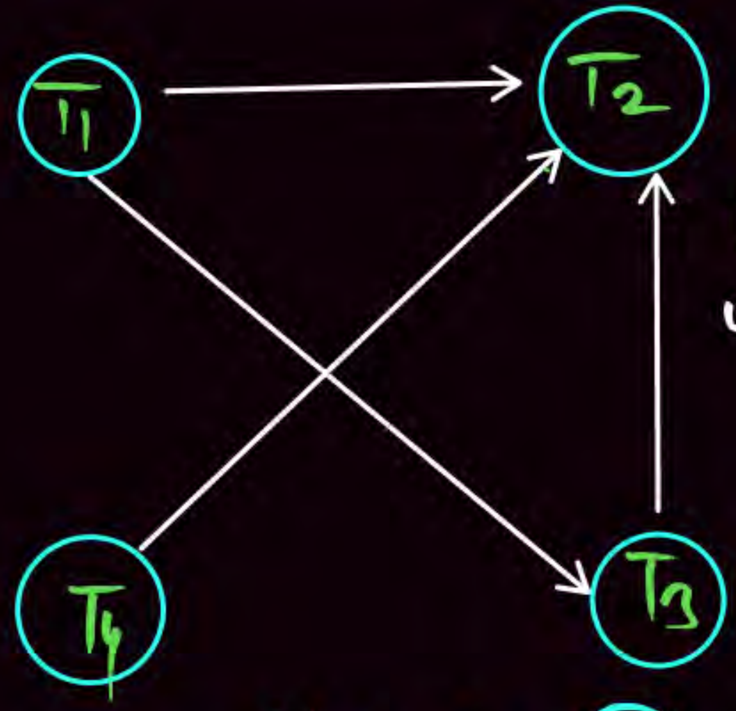
T1 & T4

Q.3

< T1 T4 T3 T2 >

< T4 T1 T3 T2 >

< T1 T3 T4 T2 >



Now For T1

(i) First start from T4

$T_3 \rightarrow T_2$

Case I

Case II

T4

T4

T4

_ T3 _ T2 _

(ii) then T1

T1 (1)

T1 (2)

_ T4 _ T3 _ T2 _

T1

T3 _ T4 _ T2 _

(3)

T1

(iii) alternate Approach

T_1 & T_4 Indegree = 0

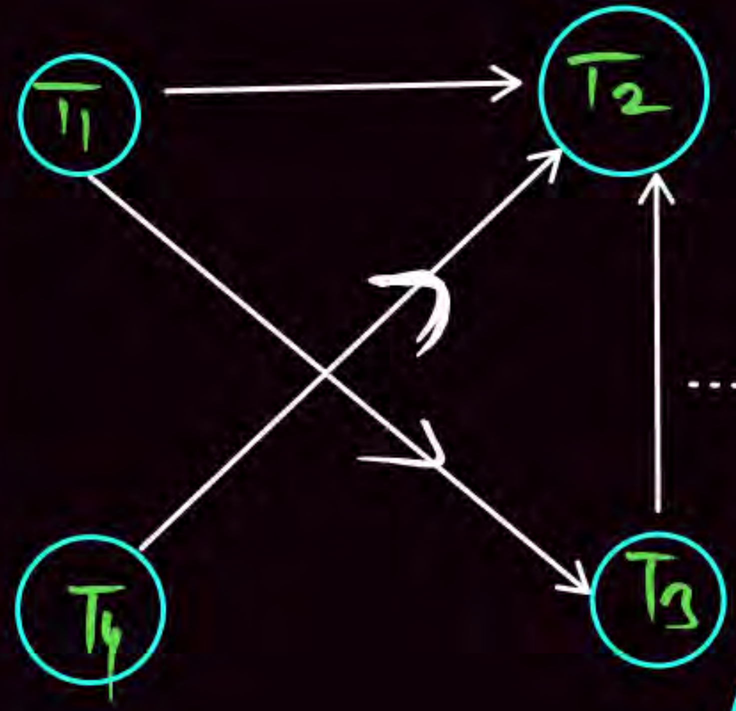
Q. 3

< T_4 T_1 T_3 T_2 >

< T_1 T_4 T_3 T_2 >

< T_1 T_3 T_4 T_2 >

Ans



T_1 & T_4 (i) Start from T_1 first

T_1

① T_1 T_1

T_3 T_2

① T_1 ② T_3 ③ T_2 ④

Now for T_4

T_4 T_1 T_4 T_3 T_4 T_2 T_4

Q.4

T_1 $T_2 \rightarrow T_3$ T_4

How many conflict serializable schedule

Here T_1 & T_4 having Indegree = 0

First T_1

T_1 ↑ T_2 ↑ T_3 ↑

T_1 ↑ T_2 ↑ T_3 ↑

T_1 ↑ T_2 ↑ T_3 ↑

Ans: 12

now for T_4 T_4

CASE I : ↑ $\boxed{T_1}$ ↑ T_2 ↑ T_3 ↑ → 4

CASE II : ↑ T_2 ↑ $\boxed{T_1}$ T_4 T_3 T_4 → 4

CASE III : ↑ T_2 ↑ T_3 ↑ $\boxed{T_1}$ ↑ → 4

12 Ans

CASE I ⇒ 4 CASE

T_4	$\boxed{T_1}$	T_2	T_3
$\boxed{T_1}$	T_4	T_2	T_3
$\boxed{T_1}$	T_2	T_4	T_3
$\boxed{T_1}$	T_2	T_3	T_4

CASE II ⇒ 4 CASE

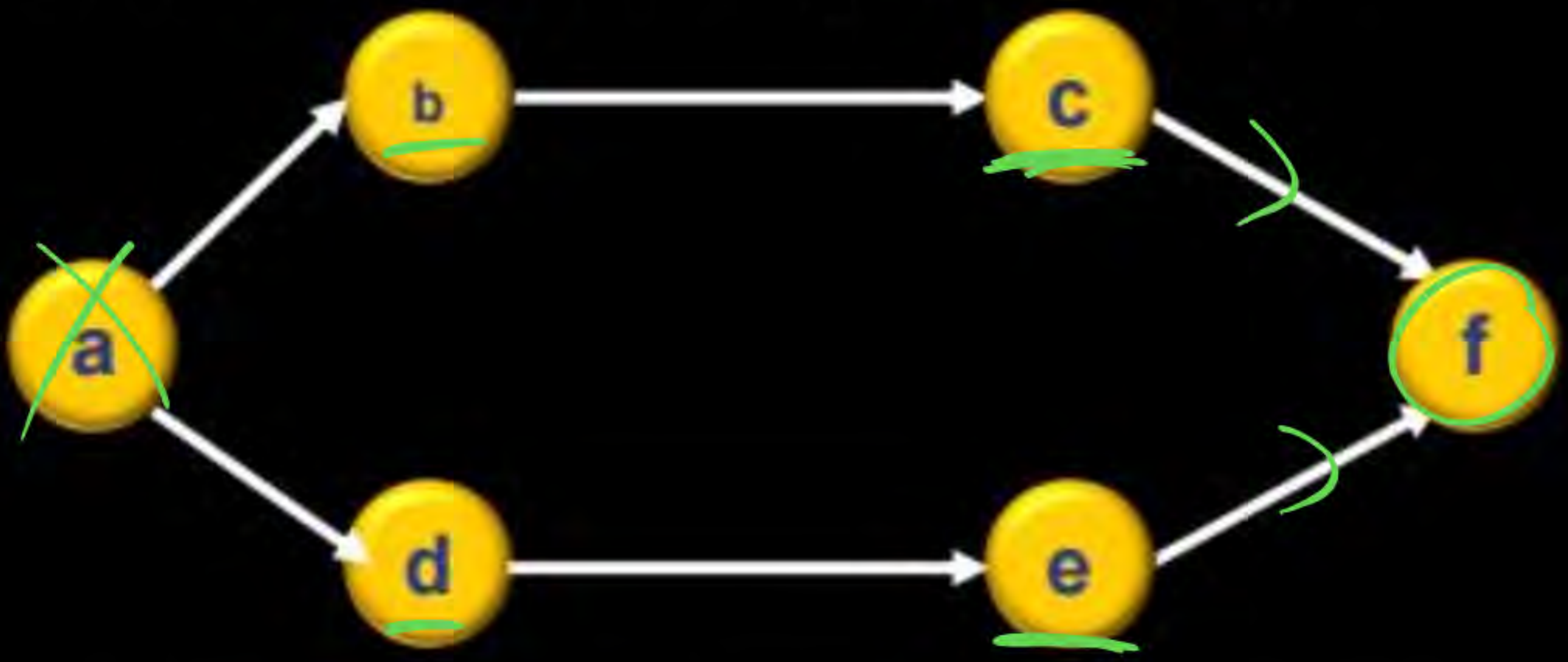
T_4	T_2	$\boxed{T_1}$	T_3
T_2	T_4	$\boxed{T_1}$	T_3
T_2	$\boxed{T_1}$	T_4	T_3
T_2	$\boxed{T_1}$	T_3	T_4

CASE III = 4 CASE

T_4	T_2	T_3	$\boxed{T_1}$
T_2	T_4	T_3	$\boxed{T_1}$
T_2	T_3	T_4	$\boxed{T_1}$
T_2	T_3	$\boxed{T_1}$	T_4

Topological Sorting

Q. Consider the following directed graph:



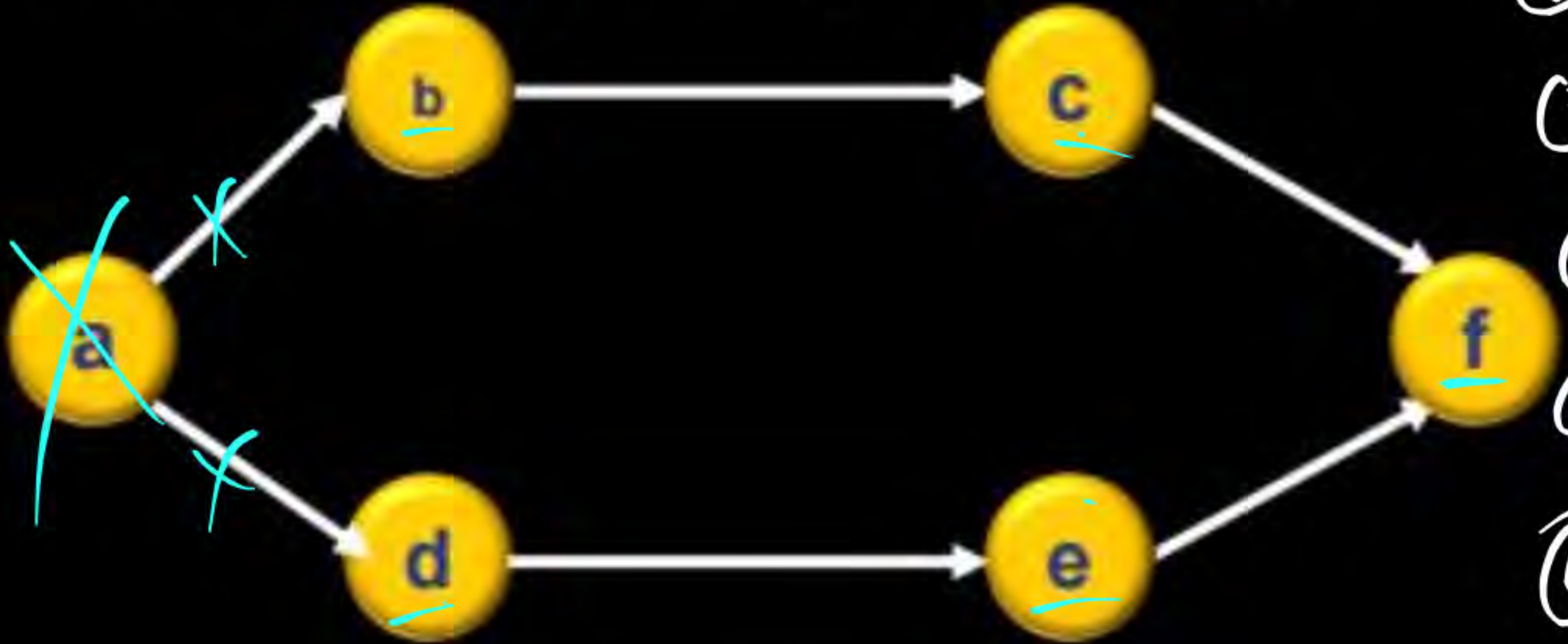
a bc de f
a de bc f
a b d c e f
a b d e c f
a d b e c f
a d b c e f

The number of different topological ordering of the vertices of the graph is ___6___.

[MCQ: 2016]

Topological Sorting

Q. Consider the following directed graph:



1. a b c d e f
2. a d e b c f
3. a b d c e f
4. a b d e c f
5. a d b e c f
6. a d b c e f

The number of different topological ordering of the vertices of the graph is ___6___.

[MCQ: 2016]

P_1 $\qquad$ P_2

L_1 $\qquad$ L_3

L_2 $\qquad$ L_4

$L_1\ L_2\ L_3\ L_4$

$L_3\ L_4\ L_1\ L_2$

$L_1\ L_3\ L_2\ L_4$ or $L_1\ L_3\ L_4\ L_2$

$L_3\ L_1\ L_4\ L_2$ or $L_3\ L_1\ L_2\ L_4$

Implementation of Concurrency Control.

LOCK BASED Protocol.

Account of B.

Concurrently

Net Banking

ATM

CHEQUE BOOK.

UPI

Before Using Any Data Item, Transaction Request for a lock.

- If lock is Free (Not taken by any other transaction on that Data Item) then Lock is granted, otherwise Transaction has to WAIT.
- Once transaction taken the lock then Perform its operation (Task) After Performing the operation, Transaction Release the lock (on) from that Data Item.

① SHARED LOCK [S LOCK]

⇓

Only Read.

Syntax : LOCK_S(A)

Read(A)

Unlock_S(A)

② Exclusive Lock [X Lock]

⇓

Write or Read & Write

Syntax: Lock_X(A)

Write(A) or Read(A)
Write(A)

Unlock_X(A)

Lock-Based Protocols

- A lock is a mechanism to control concurrent access to a data item
- Data items can be locked in two modes:

  1. exclusive (X) mode. Data item can be both reads as well as written. X-lock is requested using **lock-X** instruction.
  2. Shared (S) mode. Data item can only be read. S-lock is requested using **lock-S** instruction.

- Lock requests are made to concurrency-control manager. Transaction can proceed only after request is granted.

Same Data Item (A)	T_j: S	X
T_i: S	✓ YES	No
X	No	No.

S : SHARED

X : Exclusive.

R - W
W - R
W - W.

1

T_1	T_2	T_3	T_4
S(A)			
R(A)			
	S(A)		
	R(A)		
		S(A)	
		R(A)	
			S(A)
			R(A)

S-S : YES

2

T_1	T_2
S(A)	
R(A)	
	X(A) → Not allowed

3

T_1	T_2
X(A)	
W(A)	
	S(A) ⇒ Not granted bcz on A X lock taken by T_1 transaction

4

T_1	T_2
X(A)	
W(A)	
	X(A) → Not granted.

Lock-Based Protocols (Cont.)

- Lock-compatibility matrix

T_1 T_2 T_3 . . . T_n

A SHARED LOCK

T_j

T_i

DATA Item (A)	S	X
S	true ✓	false ✗
X	false ✗	false ✗

- A transaction may be garneted a lock on an item if the requested lock is compatible with lock already held on the item by other transactions
- Note Any number of transactions can hold shared locks on an item
- Note But if any transaction holds an exclusive on the item no other transaction may hold any loc on the item.

T_1	T_2
Read(A)	
A = A-100	
Write(A)	
	Read(A)
	Read(B)
Read(B)	
B = B+100	
Write(B)	

T_1	T_2	Lock manager
Lock-X(A)		Grant-X(A, T_1)
Read(A)		
A = A-100		
Write(A)		Release or
Unlock-X(A)		Revoke-X(A, T_1)
	Lock-S(A)	Grant-S(A, T_2)
	Read(A)	
	Unlock-S(A)	Revoke-S(A, T_2)
	Lock-S(B)	Grant-S(B, T_2)
	Read(B)	
	Unlock-S(B)	Revoke-S(B, T_2)
Lock-X(B)		Grant-X(B, T_1)
Read(B)		
B = B+100		
Write(B)		
Unlock-X(B)		Revoke-X(B, T_2).

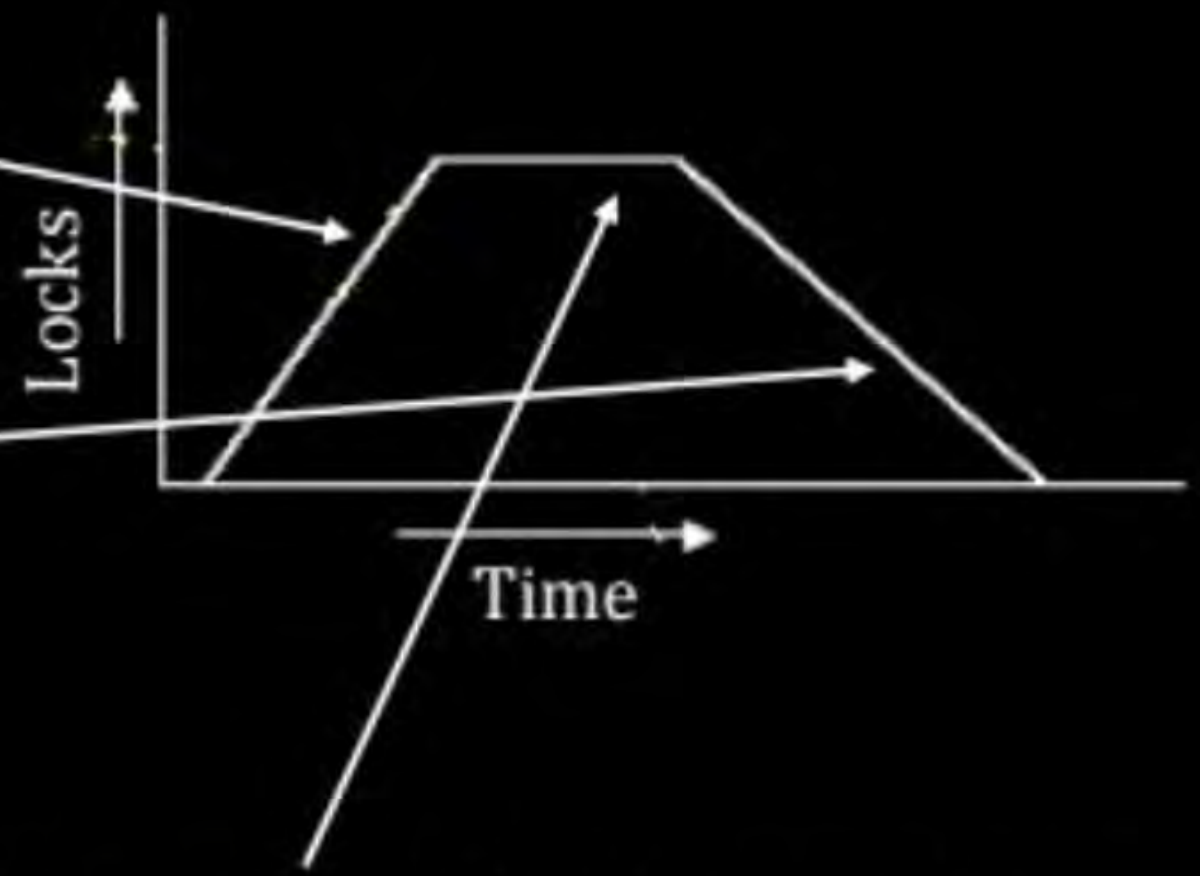
Schedule with Lock Grants

- A locking protocol is a set of rules followed by all transactions while requesting and releasing locks.
- Locking protocols enforce serializability by restricting the set of possible schedules.

T_1	T_2	Concurrency-control manager
lock-X(B)		
		grant-X(B, T_1)
read(B) B:=B – 50 write(B) unlock(B)		
	lock-S(A, T_2)	
		grant-S(A, T_2)
	read(A) unlock(A) lock-S(B)	
		grant-S(B, T_2)
	read(B) unlock(B) display(A+B)	
lock-X(A)		
		grant-X(A, T_1)
read(A) A:=A + 50 write(A) unlock(A)		

The Two-Phase Locking Protocol

- A protocol which ensures conflict-serializable schedules.
- Phase 1: Growing Phase
    - Transaction may obtain locks
    - Transaction may not release lock
- Phase 2: Shrinking Phase
    - Transaction may release locks
    - Transaction may not obtain locks



- The protocol assures serializability. It can be proved that transactions can be serialized in the order of their lock points (i.e., the point where a transaction acquired its final lock).

2PL if 2PL Applied

Serializablity ✓ Recoverability ×

→ Ensure conflict serializable schedule

→ Serializability

But → Not Ensure **Recoverability**

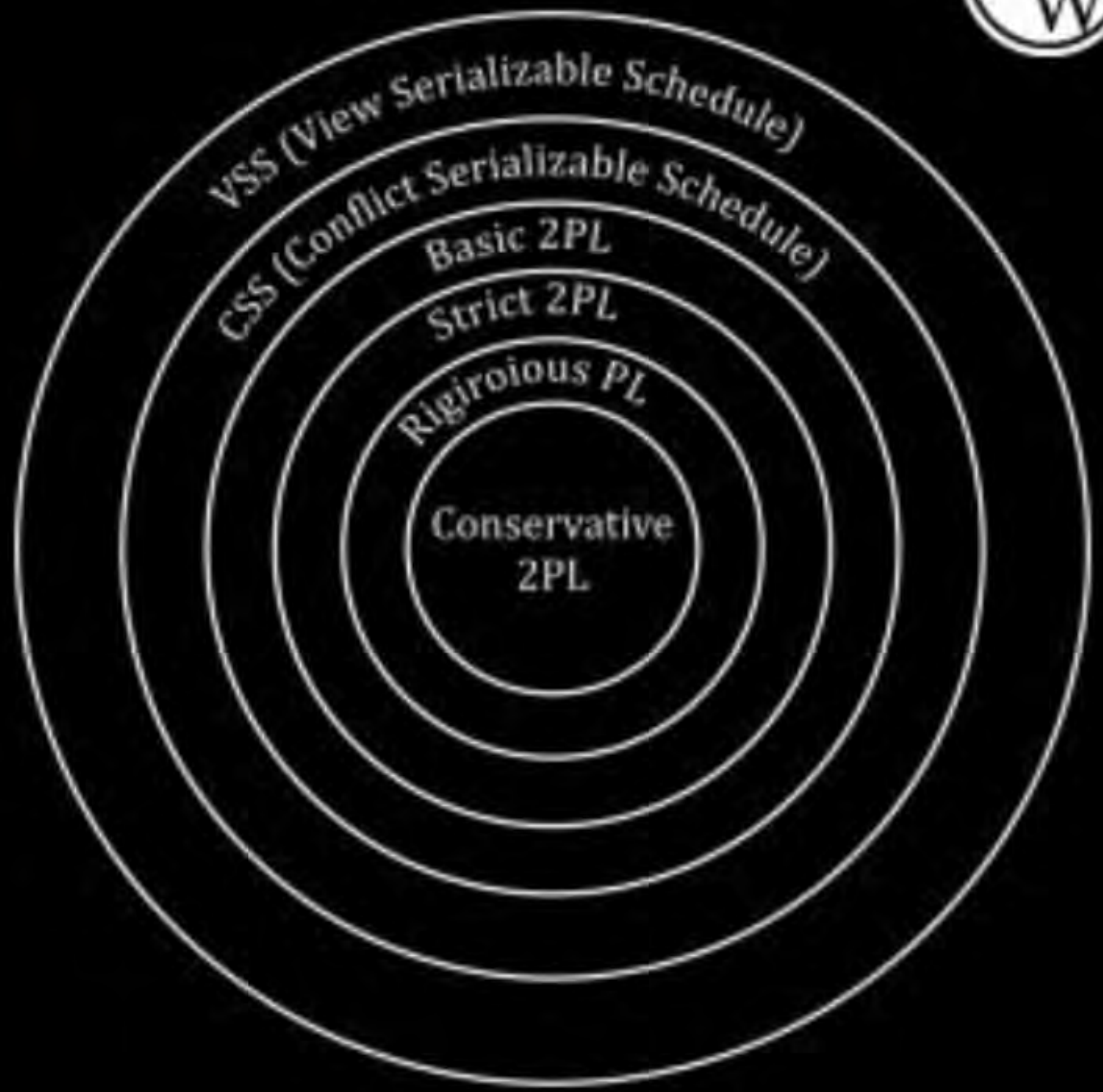
→ Suffer from **Deadlock**

Suffer from **Starvation**

The Two-Phase Locking Protocol (Cont.)

- Two-phase locking does not ensure freedom from deadlocks
- Extensions to basic two-phase locking needed to ensure recoverability of freedom from cascading roll-back
  - **Strict two-phase locking:** a transaction must hold all its exclusive locks till it commits/aborts.
    - Ensures recoverability and avoids casecading roll-backs
  - **Rigorous two-phase locking:** a transaction must hold all locks till commit/abort.
    - Transactions can be serialized in the order in which they commit.
- Most databases implement rigorous two-phase locking, but refer to if as simply two-phase locking

1	2	3	4	5
Lock-S(A)	Lock-S(A)	Lock-S(A)	Lock-S(A)	Lock-S(A)
R(A)	R(A)	R(A)	Lock-X(B)	R(A)
Lock-X(B)	Lock-X(B)	Lock-X(B)	R(A)	Unlock(A)
R(B)	Unlock(A)	R(B)	R(B)	Lock-X(B)
Unlock(A)	R(B)	W(B)	W(B)	R(B)
W(B)	W(B)	Commit	Commit	W(B)
Unlock(B)	Commit	Unlock(A)	Unlock(A)	Unlock(B)
Commit	Unlock(B)	Unlock(B)	Unlock(B)	Commit



Timestamp-Based Protocols

- Each transaction T_i is issued a timestamp $TS(T_i)$ when it enters the system.
  - Each transaction has a unique timestamp
  - Newer transaction have timestamp strictly greater than earlier ones
  - Timestamp could be based on a logical counter
    - Real time may not be unique
    - Can use (wall-clock time, logical counter) to ensure
- Timestamp-based protocols manage concurrent execution such that **time-stamp order = serializability order**
- Several alternative protocols based on timestamps

Timestamp-Based Protocols

The **timestamp ordering (TSQ) protocol**

- Maintains for each data Q two timestamp values:
  - **W-timestamp**(Q) is the largest time-stamp of any transaction that executed write (Q) successfully.
  - **R-timestamp** (Q) is the largest time-stamp of any transaction that executed **read** (Q) successfully.
- Imposes rules on read and write operations to ensure that
  - any conflicting operations are executed in timestamp order
  - out of order operations cause transaction rollback

I: T_i – Read(Q) (Transaction T_i Issue R(Q) Operation)

(i) If TS (T_i) < WTS (Q): Read operation Reject & T_i Rollback.

(ii) If TS $(T_i) \geq$ WTS(Q): Read operation is allowed

and Set Read – TS(Q) = max[RTS(Q), TS (T_i)]

II: T_i – Write(Q) (Transaction T_i Issue Write(Q) Operation)

(i) If TS (T_i) < RTS (Q): Write operation Reject & T_i Rollback.

(ii) If TS (T_i) < WTS(Q): Write operation Reject & T_i Rollback.

(iii) Otherwise execute write (Q) operation

Set Read WTS(Q) = TS (T_i)

Timestamp-Based Protocols (Cont.)

- Suppose a transaction T_i issues a **read** (Q)
  1. If $TS(T_i) \leq$ **W**-timestamp (Q), then T_i needs to read a value of Q that was already overwritten.
     - Hence, the **read** operation is rejected, and T_i is rolled back.
  2. If $TS(T_i) \geq$ **W**-timestamp (Q), then the **read** operation is executed, and R-timestamp(Q) is set to.

     max(R-timestamp (Q), TS (T_i)).

Timestamp-Based Protocols (Cont.)

- Suppose a transaction T_i issues **write**(Q)

1. If $TS(T_i)$ < **R**-timestamp(Q), then the value of Q that T_i is producing was needed previously, and the system assumed that the value would never be produced.
   - Hence, the **write** operation is rejected, and T_i is rolled back.
2. If $TS(T_i)$ < **W**-timestamp (Q), then T_i is attempting to write an obsolete value of Q.
   - Hence, this **write** operation is rejected, and T_i is rolled back.
3. Otherwise, the **write** operation is executed, and W-timestamp(Q) is set to $TS(T_i)$.

(1)

$T_1(10)$	$T_2(20)$
R(A)	
	W(A)
W(A)	

$TS(T_1) < TS(T_2)$

$T_1 \rightarrow T_2$

(2)

$T_1(10)$	$T_2(20)$
R(A)	
	R(A)
W(A)	

$TS(T_1) < TS(T_2)$

$T_1 \rightarrow T_2$

(3)

$T_1(10)$	$T_2(20)$
R(A)	
	W(A)
R(B)	
	W(B)

$TS(T_1) < TS(T_2)$

$T_1 \rightarrow T_2$

(4)

$T_1(10)$	$T_2(20)$
	r(A)
r(B)	
W(A)	

$TS(T_1) < TS(T_2)$

$T_1 \rightarrow T_2$

Thomas' Write Rule

- Modified version of the timestamp-ordering protocol in which obsolete **write** operations may be ignored under certain circumstances.
- When T_i attempts to write data item Q, if $TS(T_i) < W\text{-timestamp}(Q)$, then T_i is attempting to write an obsolete value of {Q}.
  - Rather than rolling back T_i as the timestamp ordering protocol would have don, this {**write**} operation can be ignored.
- Otherwise this protocol is the same as the timestamp ordering protocol.
- Thomas' Write Rule allows greater potential concurrency.
  - Allows some view-serializable schedules that are not conflict-serializable.

Thomas Write Rule (View Serializability)

1. $TS(T_i) < RTS(Q)$: Rollback
2. $TS(T_i) < WTS(Q)$: Write operation is Ignored and No Roll back

Same as TSP

Time Stamp Protocol: Ensure serializability deadlock free but starvation possible

Deadlock Prevention Algorithm

(1) Wait-Die (2) Wound-wait

Older Younger

$T_1(10)$	$T_2(20)$
R(A)	
	W(A)
W(A)	

$TS(T_1) < TS(T_2)$

$T_1 \rightarrow T_2$

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

$T_1 \rightarrow T_2$

THANK YOU!