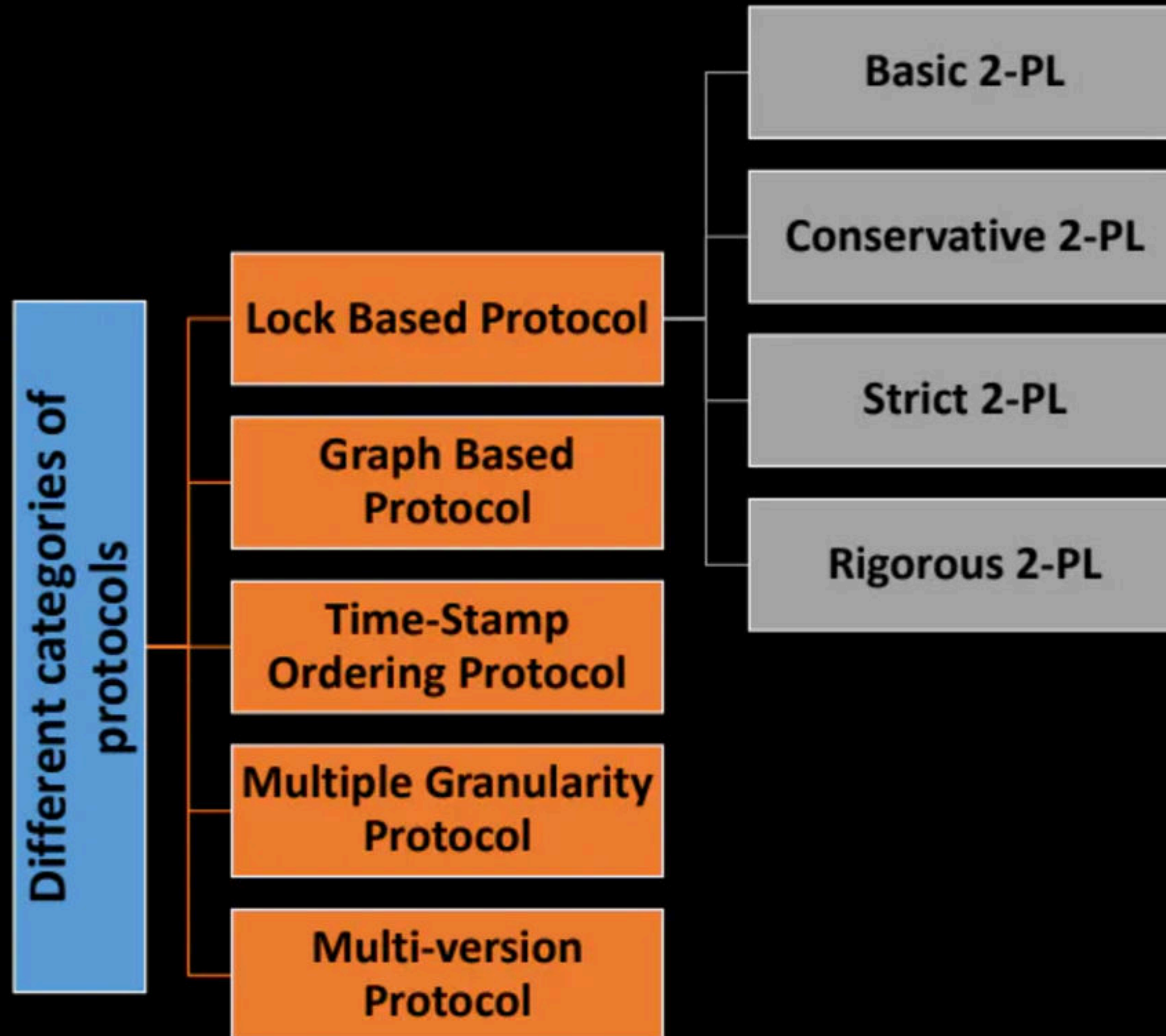


Concurrency Control protocols



Lock Based Protocol

Lock Based Protocol

A lock is a variable associated with a data item that describes a status of data item with respect to possible operation that can be applied to it.

Basically Two common locks which are used:

1.Shared Lock (S):

also known as Read-only lock.

It can be shared between transactions because while holding this lock the transaction does not have the permission to update data on the data item.

If we have data item A then S-lock is requested using Lock_R(A) instruction.

1.Exclusive Lock (X):

Data item can be both read as well as written.

This is Exclusive and cannot be held simultaneously on the same data item.

If we have data item A then X-lock is requested using Lock_W(A) instruction.

Lock Compatibility Matrix –

	Shared	Exclusive
Shared	Yes	No
Exclusive	No	No

Problems with simple locking

Example:

T1: A is trying to transfer an amount of ₹50 to B

T2: Display A+B

T1	T2
L_S(A)	
READ(A)	
U_S(A)	
L_X(A)	
A=A-50	
WRITE(A)	
U_X(A)	
	A+B 250

A	B
100	200
50	250

INCONSISTENT

Another scenario:

T1	T2
L_X(A)	
W(A)	
U_X(A)	
	L_S(A)
	R(A)
	U_S(A)
L_X(A)	
W(A)	
U_X(A)	

This as you may notice imposes a **Deadlock** as none can proceed with their execution.

So, Implementing this lock system without any restrictions that is Simple Lock based protocol has its own disadvantages, they **does not guarantee Serializability**.

Two Phase Locking(2-PL) comes in the picture, 2-PL ensures serializability. Now, let's check it !

Two Phase Locking –

A transaction is said to follow Two Phase Locking protocol if Locking and Unlocking can be done in two phases.

Growing Phase:

New locks on data items may be acquired but none can be released.

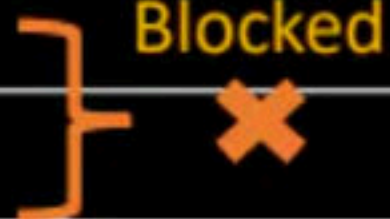
Shrinking Phase:

Existing locks may be released but no new locks can be acquired.

In the following example, $W(A)$ and $W(b)$ will not be allowed since T_i has exclusive lock

T_i	T_j
$L_X(A)$	
$W(A)$	
	$W(A)$
	$W(B)$
$L_X(B)$	
$W(B)$	

Blocked



So, What is the order in which serializability is obtained?

For that, there is something called as Lock Point !

What is **LOCK POINT** ?

It is the Point at which the growing phase ends,
i.e. when transaction takes the final lock it needs to carry on its work.

In the following example, $W(A)$ and $W(B)$ will not be allowed since T_i has exclusive lock



T_i	T_j
$L_X(A)$	
$W(A)$	
	$W(A)$
	$W(B)$
$L_X(B)$	
$W(B)$	
LOCK POINT	

The table illustrates a sequence of operations for two transactions, T_i and T_j . T_i holds an exclusive lock on A ($L_X(A)$) and then writes to A ($W(A)$). T_j attempts to write to A ($W(A)$) and then B ($W(B)$). A bracket groups $W(A)$ and $W(B)$ in T_j 's column, with a red 'X' and the word 'Blocked' indicating that these operations are disallowed because T_i holds an exclusive lock on A . The 'LOCK POINT' is marked at the end of T_i 's operations.

So, What is the order in which serializability is obtained?

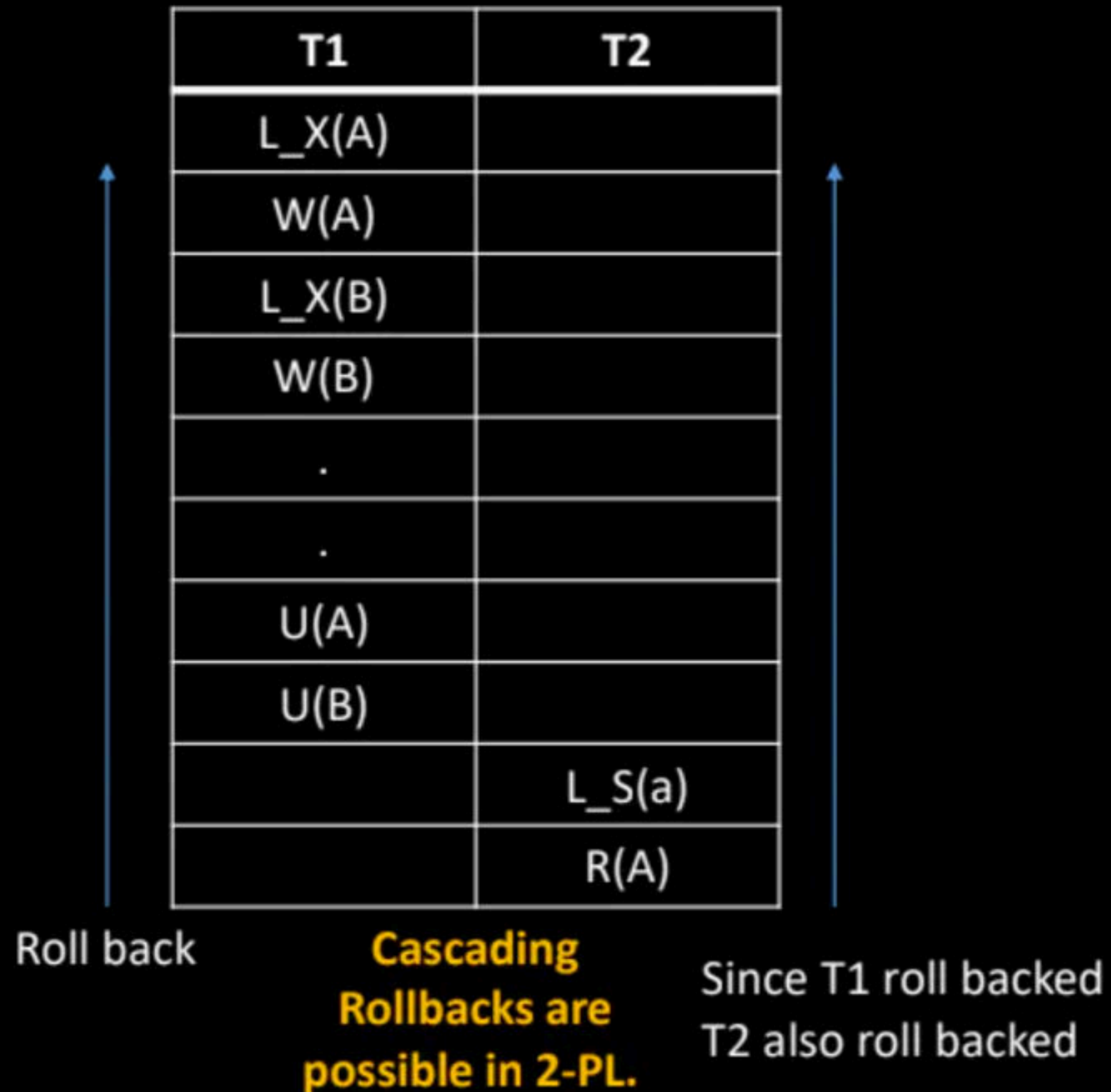
For that, there is something called as Lock Point !

What is **LOCK POINT** ?

It is the Point at which the growing phase ends,
i.e. when transaction takes the final lock it needs to carry on its work.

But there are still some drawbacks of 2-PL.

- Cascading Rollback is possible under 2-PL.
- Deadlocks and Starvation is possible.



T1	T2
L_X(A)	
	L_X(B)
L_X(B)	
	L_X(A)

Deadlock

Drawing the precedence graph, you may detect the loop. So Deadlock is also possible in 2-PL.

Strict 2-PL

This requires that in addition to the lock being 2-Phase all Exclusive(X) Locks held by the transaction be released until after the Transaction Commits.

Following Strict 2-PL ensures that our schedule is:

- Recoverable
- Cascadeless

Hence it gives us freedom from Cascading Abort which was still there in Basic 2-PL and moreover guarantee Strict Schedules but still Deadlocks are possible!

Rigorous 2-PL –

This requires that in addition to the lock being 2-Phase all Exclusive(X) and Shared(S) Locks held by the transaction be released until after the Transaction Commits.

Following Rigorous 2-PL ensures that our schedule is:

- Recoverable
- Cascadeless

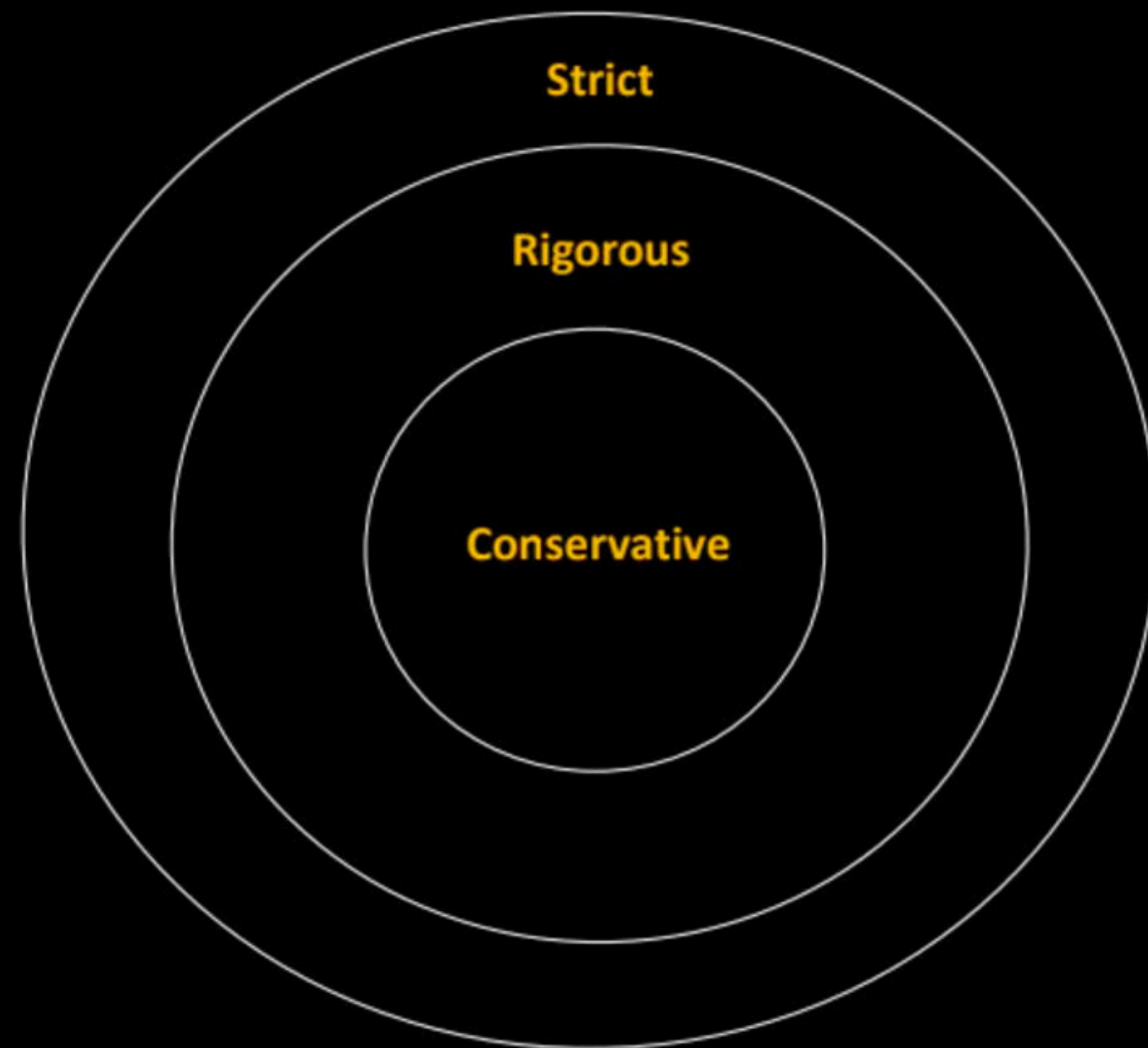
Hence it gives us freedom from Cascading Abort which was still there in Basic 2-PL and moreover guarantee Strict Schedules but still Deadlocks are possible!

Note the difference between Strict 2-PL and Rigorous 2-PL is that Rigorous is more restrictive, it requires both Exclusive and Shared locks to be held until after the Transaction commits and this is what makes the implementation of Rigorous 2-PL more easy.

Conservative 2-PL

This protocol requires the transaction to lock all the items it access before the Transaction begins execution by pre-declaring its read-set and write-set. If any of the pre-declared items needed cannot be locked, the transaction does not lock any of the items, instead it waits until all the items are available for locking.

Conservative 2-PL is Deadlock free and but it does not ensure Strict schedule.



Q.) Which of the following schedules(Transactions) are strict 2PL?

LOCK_S(A)
R(A)
LOCK_X(B)
R(B)
UNLOCK(A)
W(B)
UNLOCK(B)

LOCK_S(A)
R(A)
LOCK_X(B)
UNLOCK(A)
R(B)
W(B)
COMMIT
UNLOCK(B)

LOCK_S(A)
R(A)
LOCK_X(B)
R(B)
W(B)
COMMIT
UNLOCK(A)
UNLOCK(B)

Q.) Which of the following schedules(Transactions) are strict 2PL?

LOCK_S(A)
R(A)
LOCK_X(B)
R(B)
UNLOCK(A)
W(B)
UNLOCK(B)

Not Strict

LOCK_S(A)
R(A)
LOCK_X(B)
UNLOCK(A)
R(B)
W(B)
COMMIT
UNLOCK(B)

Strict

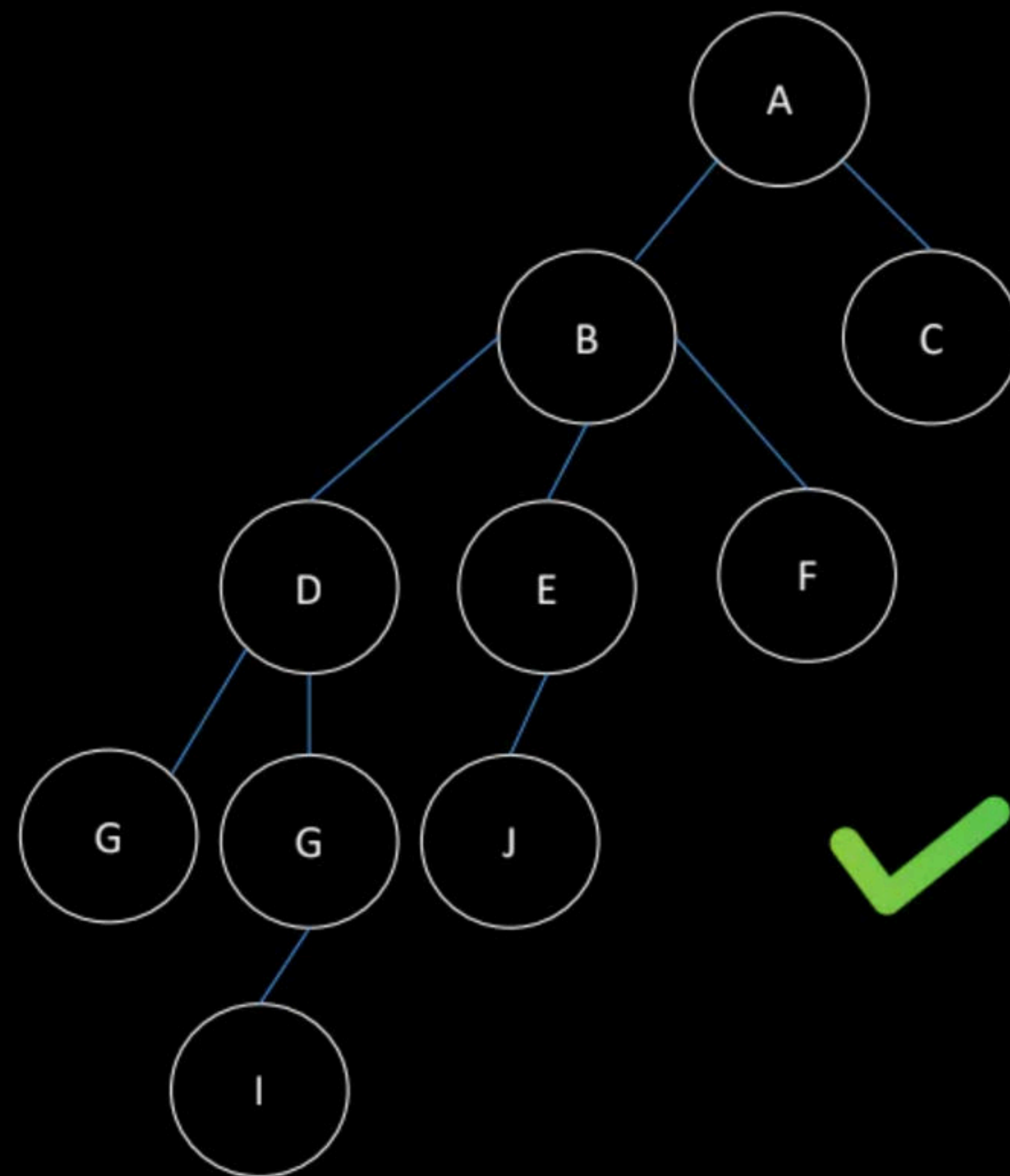
LOCK_S(A)
R(A)
LOCK_X(B)
R(B)
W(B)
COMMIT
UNLOCK(A)
UNLOCK(B)

Strict

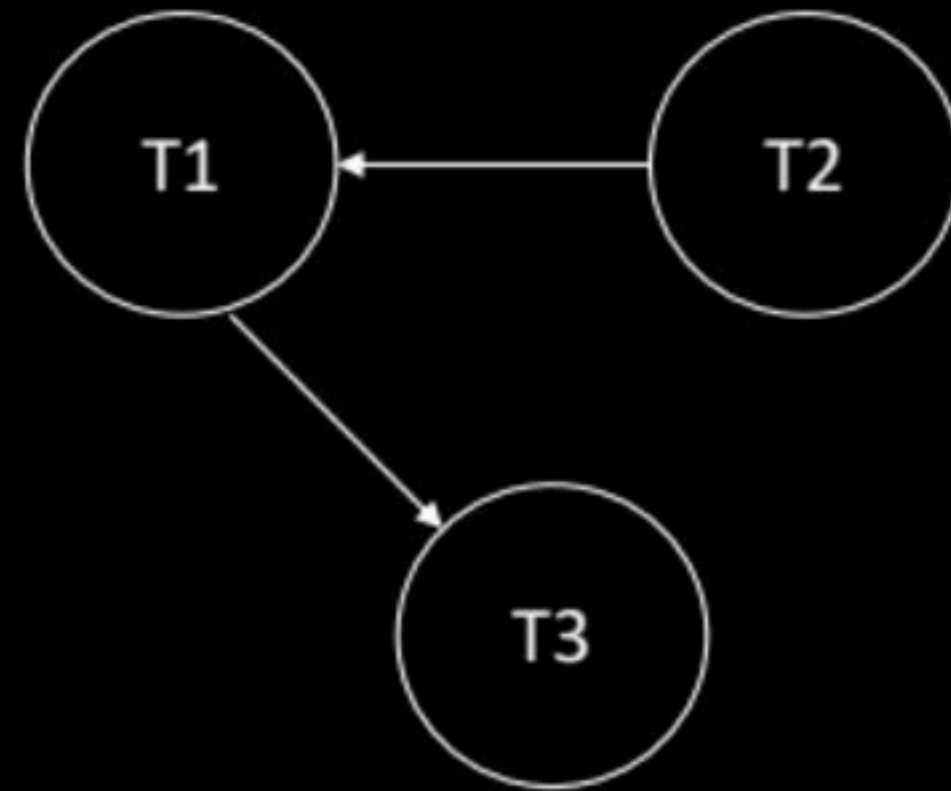
Graph based protocol

- Even though name is graph based protocol it is implemented as tree protocol.
- It is the directed acyclic graph known as “database graph”
- Advantage is we can completely avoid deadlocks.
- In tree protocol the only lock instruction allowed is “lock_x” and each transaction ‘Ti’ can lock a data item at most once
- and following below rules.
 - 1.)The first lock ‘Ti’ may be on any data item
 - 2.)Subsequently a data item can be locked by Ti only if the parent of the data item is currently locked by Ti
 - 3.)Data items may be unlocked at any time
 - 4.)A data item that has been locked and unlocked by Ti cannot subsequently be relocked by ‘Ti’

T1	T2	T3
LOCK_X(B)		
	LOCK_X(D)	
	LOCK_X(H)	
	UNLOCK_X(D)	
LOCK_X(E)		
LOCK_(D)		
UNLOCK(B)		
UNBLOCK(E)		
		LOCK_X(B)
		LOCK_X(E)
	UNLOCK(H)	
LOCK_X(G)		
UNLOCK(D)		
		UNLOCK(E)
		UNLOCK(B)



Precedence Graph



No cycle,
Conflict serializable

Advantage –

- Ensures Conflict Serializable Schedule.
- Ensures Deadlock Free Schedule
- Unlocking can be done anytime

With some advantages comes some Disadvantages also.

Disadvantage –

- ***Cascading Rollbacks*** is still a problem. We don't follow a rule of when Unlock operation may occur so this problem persists for this protocol. Overall this protocol is mostly known and used for its unique way of implementing Deadlock Freedom.

Time stamp based protocol

- Which requires that ordering among transactions is determined in advance based on their time stamps (i.e The time at which transaction enters into the system for execution)
- Each transaction is given unique fixed time stamp denoted by $TS(T_i)$
- If any transaction T_j enters after T_i , the relation between their time stamps will be $TS(T_i) < TS(T_j)$ which means that the resulting schedule must be equivalent to serial schedule $T_i \rightarrow T_j$

Time Stamp Ordering protocol

Time stamp ordering protocol ensures that any conflicting read and write operations are executed in time stamp order, if not such an operation is rejected and transaction will be rolled back. The rolled back transaction will be restarted with a new time stamp.

Suppose,
 $TS(T1) = 1$
 $TS(T2) = 2$

T1	T2
R(A)	
	W(A)
W(A)	

Time Stamp Ordering protocol

Time stamp ordering protocol ensures that any conflicting read and write operations are executed in time stamp order. If not such an operation is rejected and transaction will be rolled back. The rolled back transaction will be restarted with a new time stamp.

Suppose,
 $TS(T1) = 1$
 $TS(T2) = 2$

T1	T2
R(A)	
	W(A) ✓
W(A)	

Conflicting action

We need to decide whether to allow this operation or not

Now, Since, $TS(T1) < TS(T2)$
The operation **will be allowed**

Time Stamp Ordering protocol

Time stamp ordering protocol ensures that any conflicting read and write operations are executed in time stamp order. If not such an operation is rejected and transaction will be rolled back. The rolled back transaction will be restarted with a new time stamp.

Suppose,
 $TS(T1) = 1$
 $TS(T2) = 2$



Conflicting action

We need to decide whether to allow this operation or not

The operation will **not be allowed**,
Since greater transaction timestamp
Has already executed

Let's check one more example

Suppose,

$TS(T1) = 1$

$TS(T2) = 2$

T1	T2
R(A)	
	R(A)
W(A)	

ROLLBACK

REJECT

At some
point of time

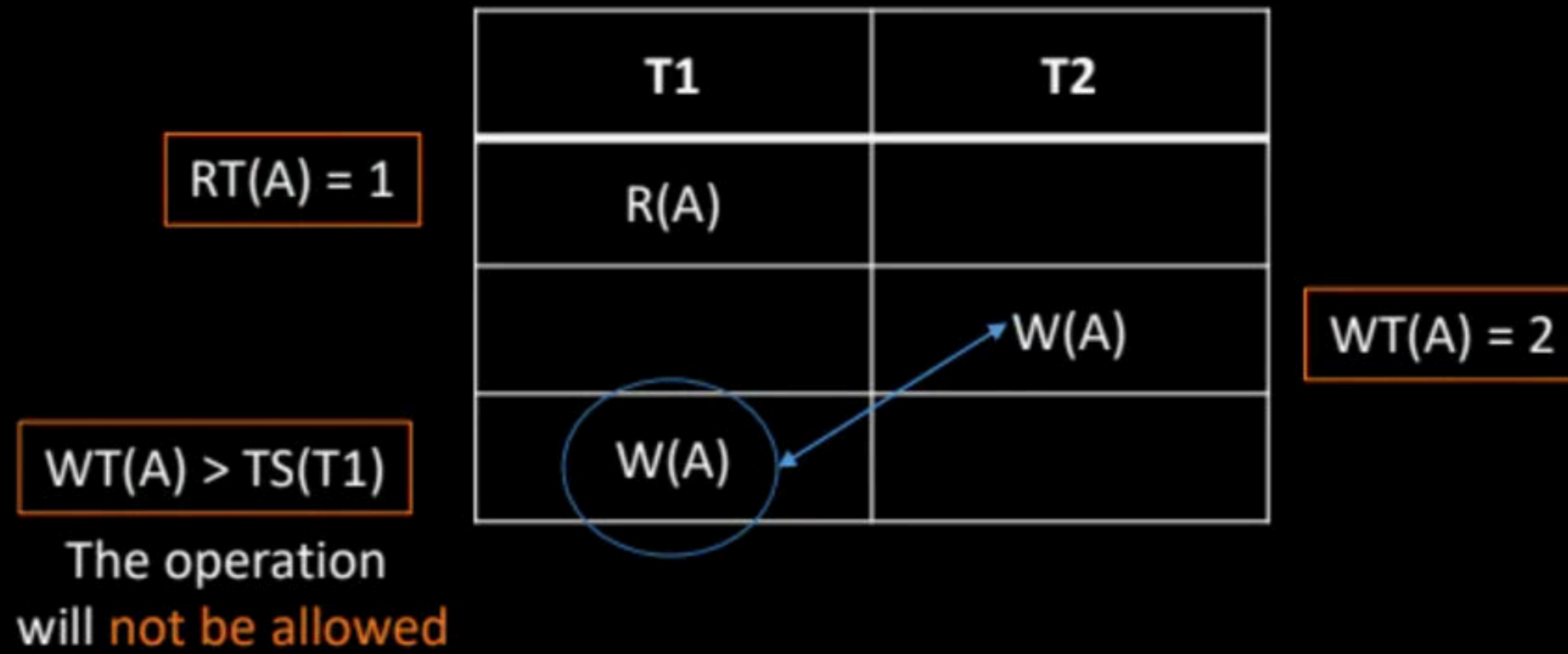
T2	T1
R(A)	
	R(A)
	W(A)

Conflicting action

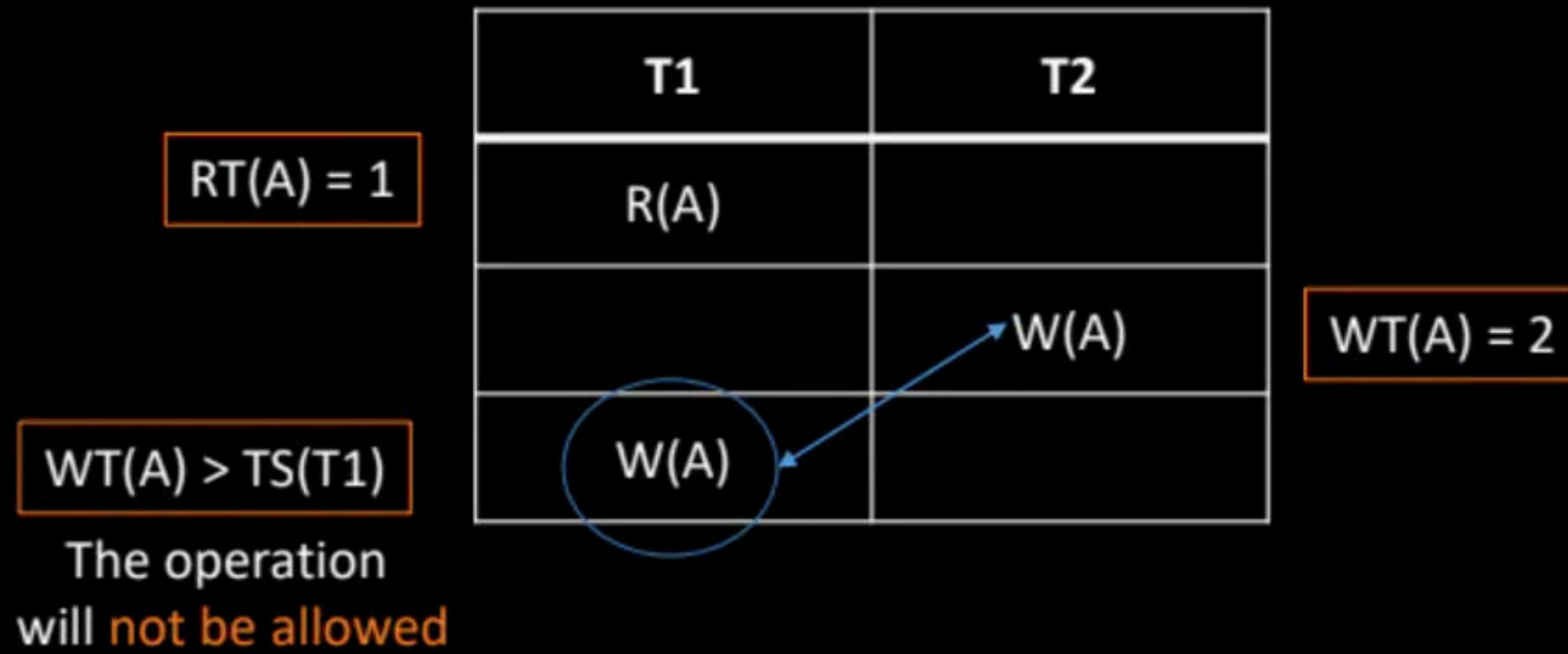
We need to decide whether to
allow this operation or not

The operation will **not be allowed**,
Since greater transaction timestamp
Has already executed

Actual implementation of this protocol is this way,



Actual implementation of this protocol is this way,



- Timestamp protocol ensures freedom from deadlock as no transaction ever waits.
- But the schedule may *not be cascade free*, and may not even be recoverable.

Timestamp Ordering Protocol states that if $R_i(X)$ and $W_j(X)$ are conflicting operations then $R_i(X)$ is processed before $W_j(X)$ if and only if $TS(T_i) < TS(T_j)$.

Whenever a schedule does not follow serializability order according to the Timestamp, user generally reject it and rollback the Transaction. Some operations on the other hand are harmless and can be allowed.

Thomas Write Rule allows such operations and is a modification on the Basic Timestamp Ordering protocol. **In Thomas Write Rule user ignore outdated writes.**

Example:

T1	T2
R(A)	
	W(A) ✓
W(A) ✓	

Conflicting action

The operation **will be allowed**

In Thomas Write Rule

By ignoring the previous write

$$TS(T1) < TS(T2)$$

	Not Allowed	Allowed
Time Stamp Ordering Protocol	R2(A) W1(A) W2(A) R1(A) W2(A) W1(A)	R2(A) R1(A)
Thomas Write Rule	R2(A) W1(A) W2(A) R1(A)	R2(A) R1(A) W2(A) W1(A)