Serializability

Serializability: It ensures that concurrent transactions yield results that are consistent with some serial execution i.e the final state of the database after executing a set of transactions concurrently should be the same as if the transactions had been executed one after another in some order.

In case of concurrent schedule consistentcy issue may arise because of non-serial execution and we do serializability there, serial schedules are aready serial

**** Serializability only tells if schedule is serializable or not. It does not serialize it.**

- **There are two types of serializability:**

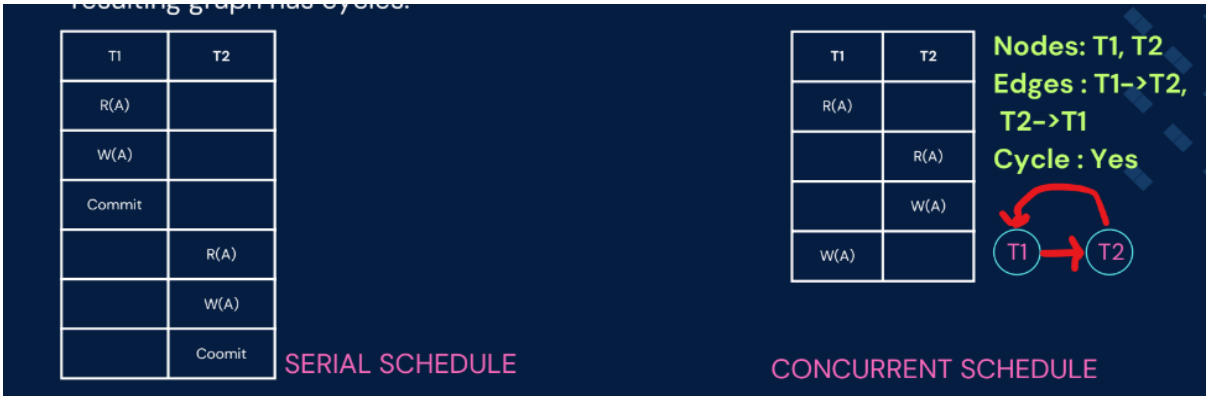
1. Conflict Serializability
2. View Serializability

- **To check if a concurrent schedule is serializable or not:** First check conflict serializability by checking if it has a cycle in conflict-graph or not, if no, then **conflict-serializable**, if yes then check for view serializability by checking its view equivalent follows some condition or not, if follows then **view-serializable**, otherwise not serializable.

- **<u>Conflict Cycle</u>**

  Transactions are represented by Nodes and Conflicts are represented by Edges.

  - Serial Schedule does not have any conflict cycle as there is no conflict so, no edges are there.
  - Concurent Schedule may or may not have a cycle in conflict graph.



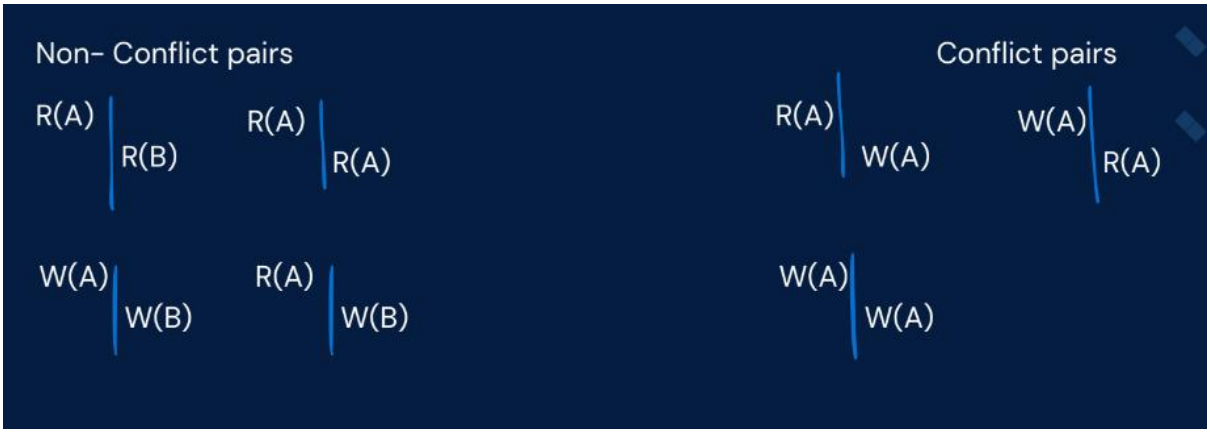
Conflict Serializability

If the schedule can be rearranged (without violating any dependencies) to form a serial schedule, then it is **conflict serializable.**

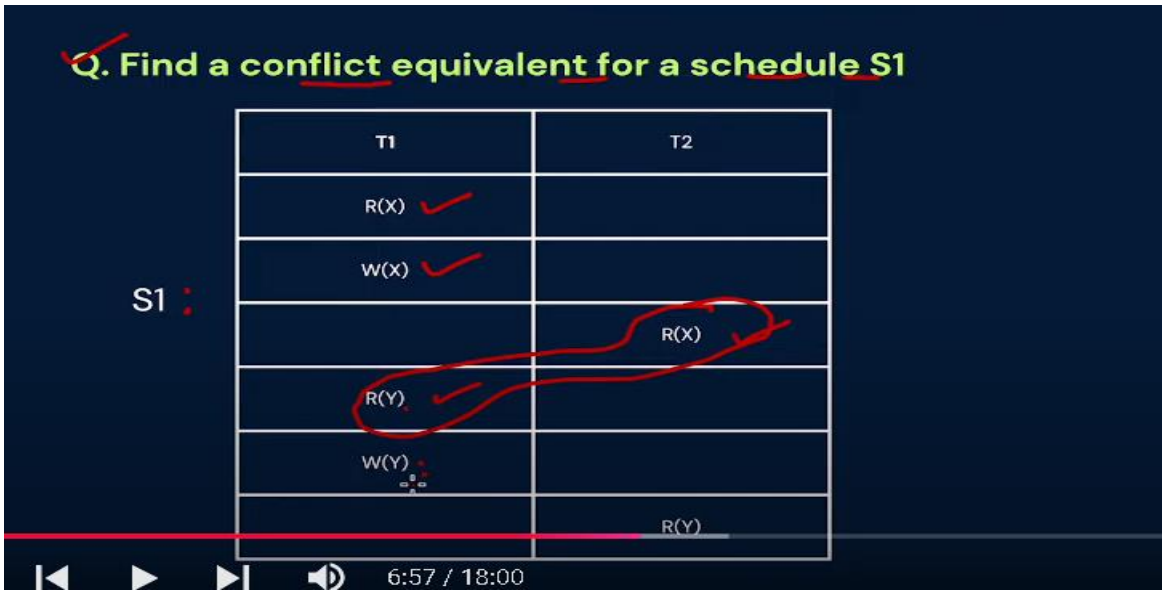
There are two methods to find if a schedule is conflict-serializable:

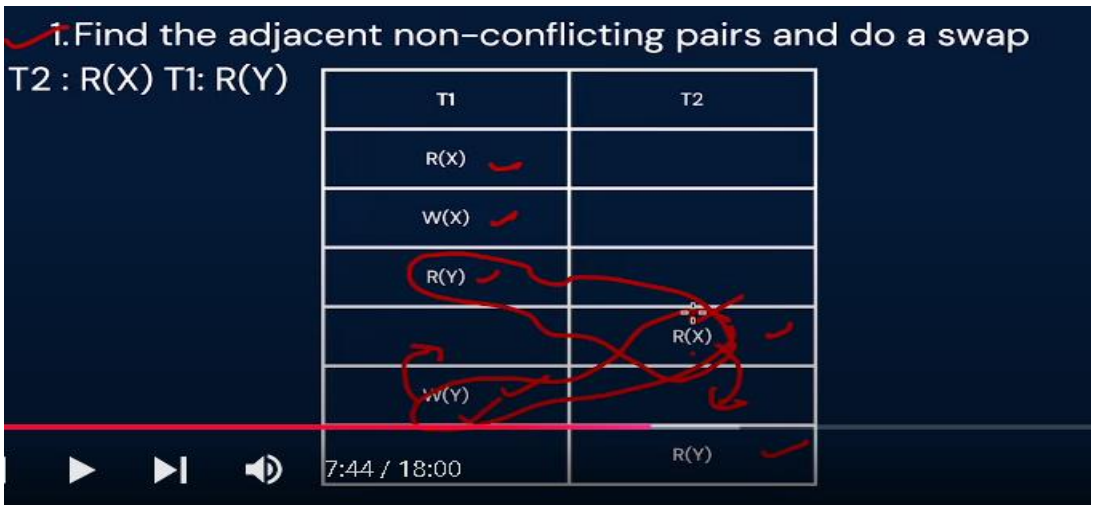
1. **Conflict Equivalent:** If a schedule S1 is formed after swapping adjacent non-conflicting operations/pairs in a given schedule S, then S1 and S are conflict equivalent. If the conflict equivalent S1 has no conflicting adjacent pairs after swapping all adjacent non-conflicting operations/pairs, then its Serializable.

   Conflict pairs: Read-Write, Write-Read, Write-Write (perfomed on the same data item)
   Non-conflict pairs: Read-Read (perfomed on the same data item), Read-Read (perfomed on the different data item), Write-Write (perfomed on the different data item)

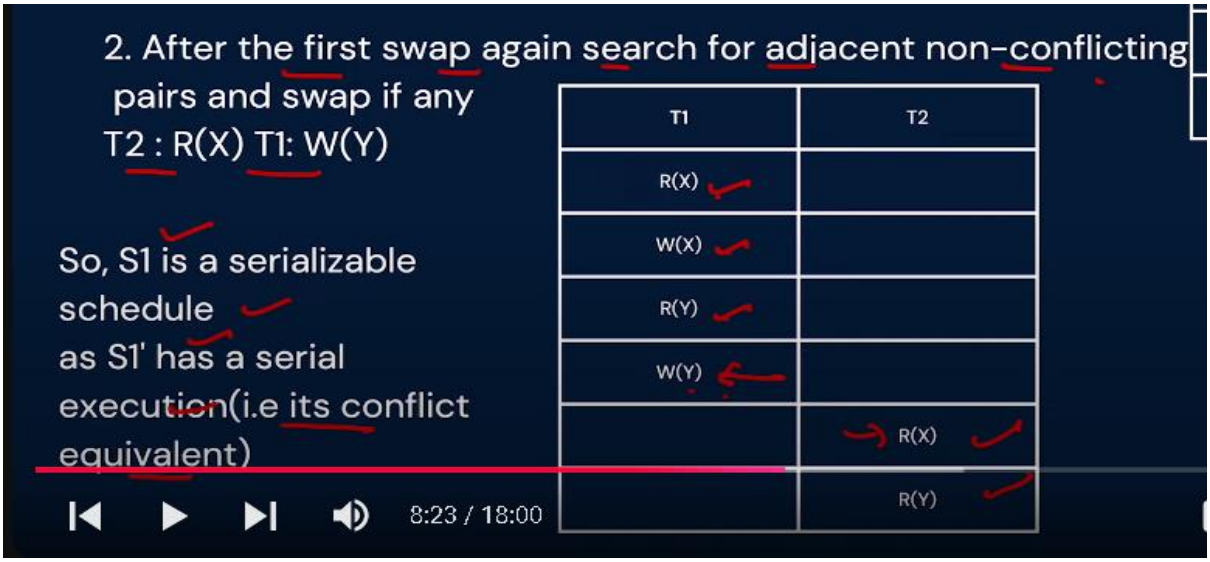


Now, why are we only swapping the non-conflict pairs and not the conflict ones?
So if we swap the conflict pairs, the order of exceution if it was
T1 : R(A)
T2: W(A)
the results values may change as first we were reading A and then writing/modifying it, but now it will be writing A and then reading the modified value so the result might change if we change the order of execution.





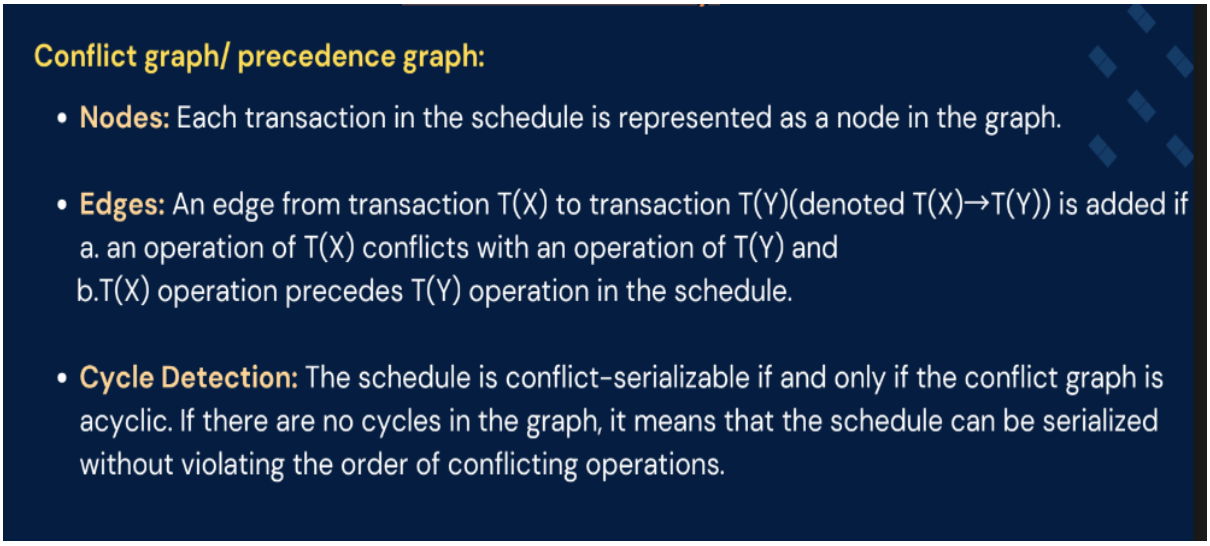
(do not swap already swapped pair, pair should have adjacent operations in the timeline)

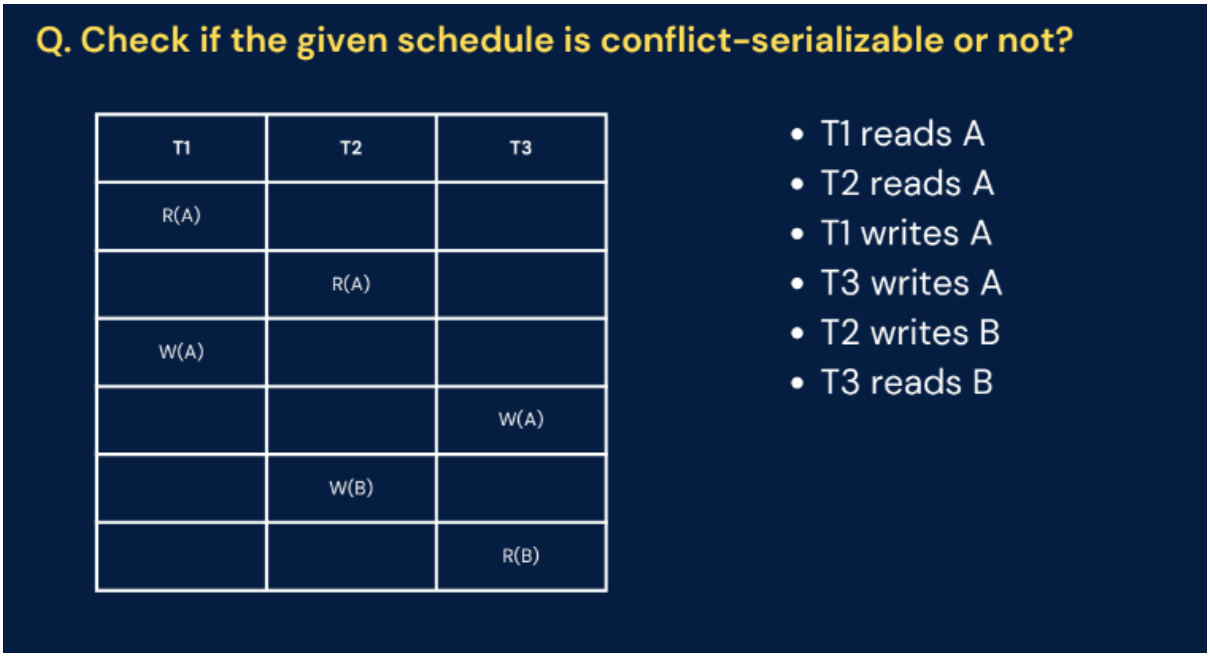


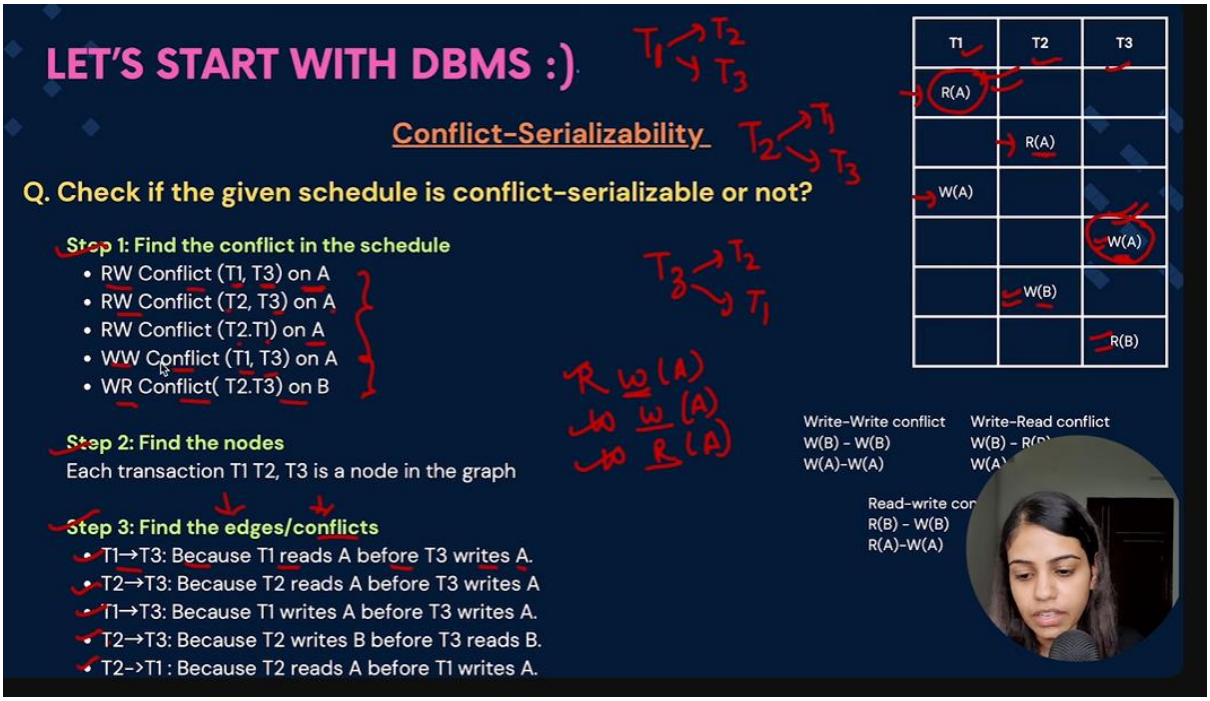
As, there is no more conflicting pairs (adjacent operations), It is serializable.

We do not use this method as frequent as for long transactions, it will take time.

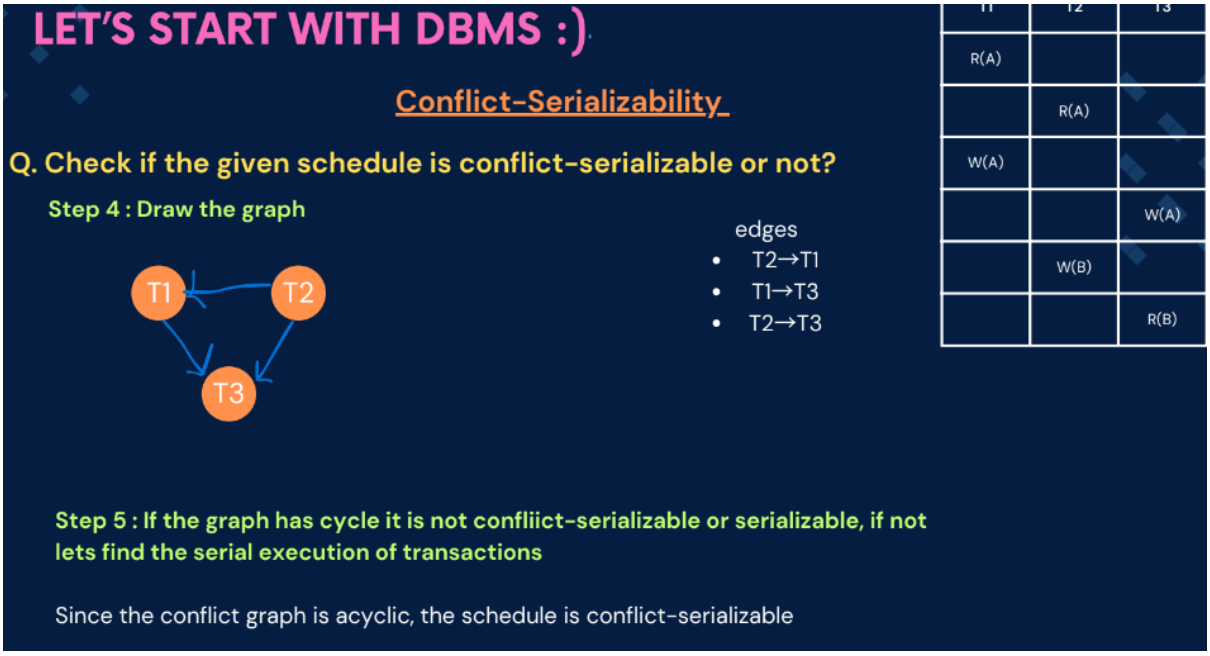
2. **Conflict graph/precedence graph:** A conflict graph, or precedence graph, is a directed graph used to determine conflict serializability. The nodes represent transactions, and the edges represent conflicts between transactions.

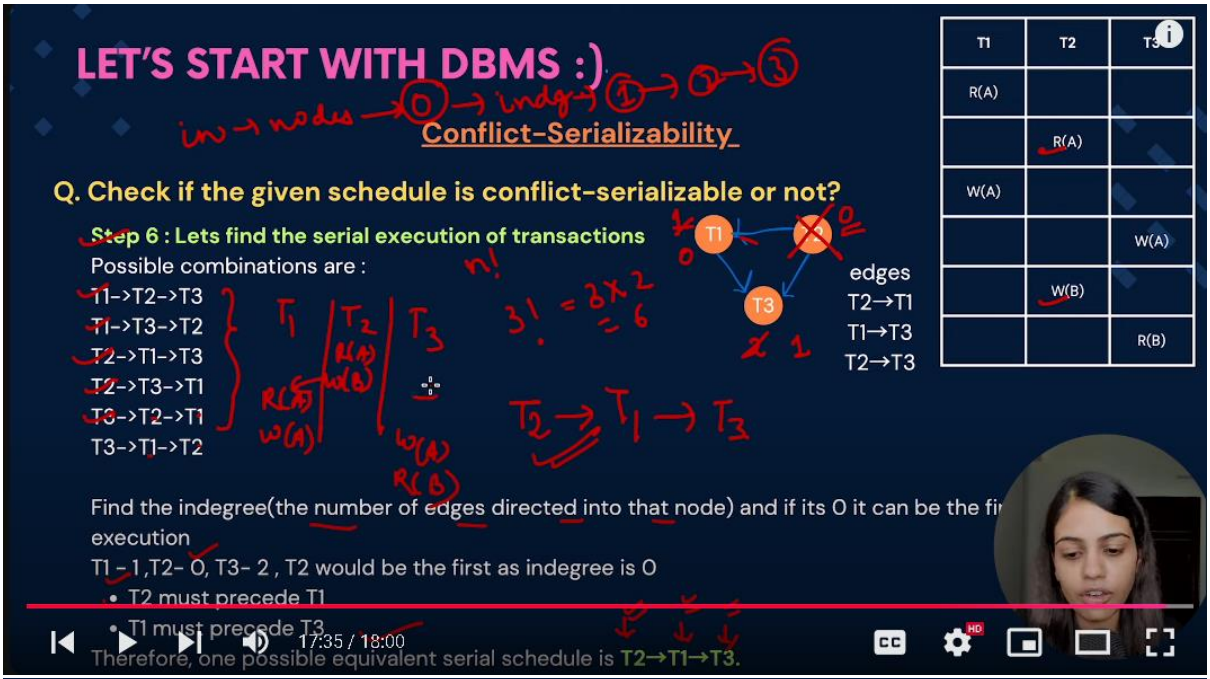






For T1, check T2 and T3. For T2 check T1 and T3 and so on for T3. For R, check W (read-write conflict), For W check W (write-write conflict) and R (write-read conflict) (on same data). ie. For R(A) in T1, check if there is any W(A) (read-write conflict) in T2 and T3 (may be non-adjecent)





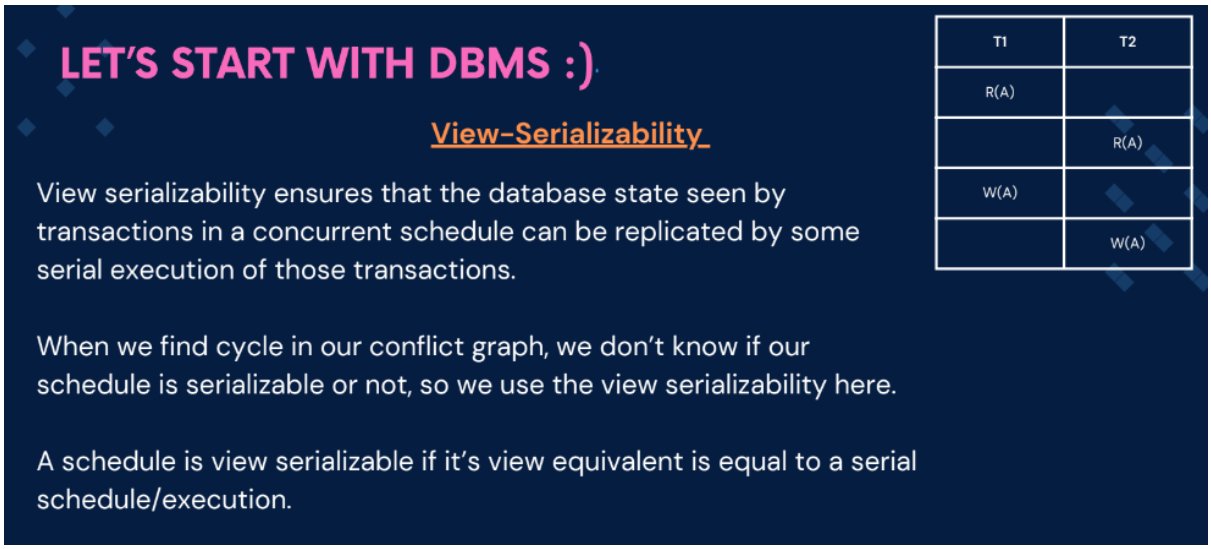
Find the indegree(the number of edges directed into that node) and if its 0 it can be the first in serial execution
T1 - 1 ,T2- 0, T3- 2 , T2 would be the first as indegree is 0
- T2 must precede T1
- T1 must precede T3

Therefore, one possible equivalent serial schedule is T2→T1→T3.

As T2 is removed from cycle (as it had 0 indegree means no conflict), indegree(T1) = 0 and indegree(T2) =1.

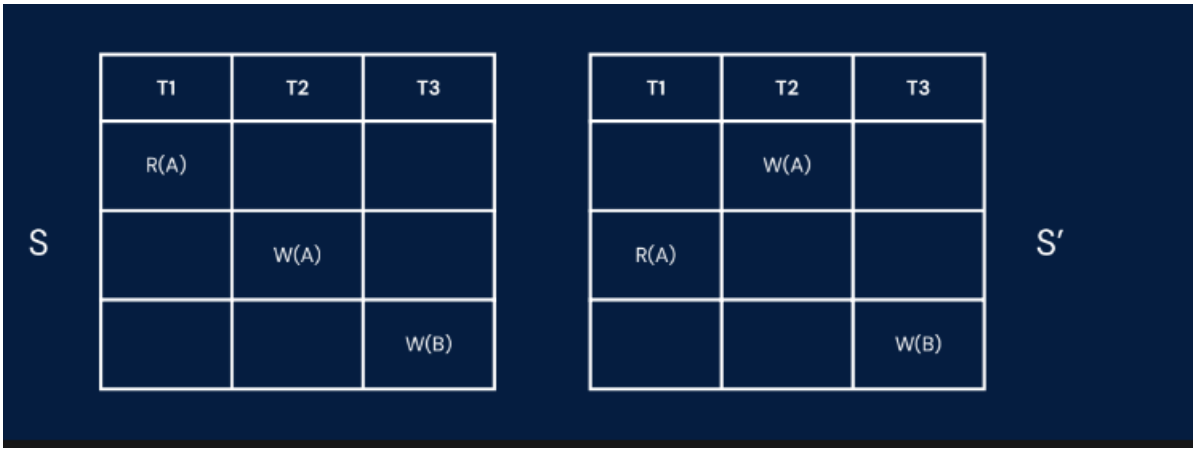
Here the drawn timeline is the conflict equivalent of the schedule.

View Serializability

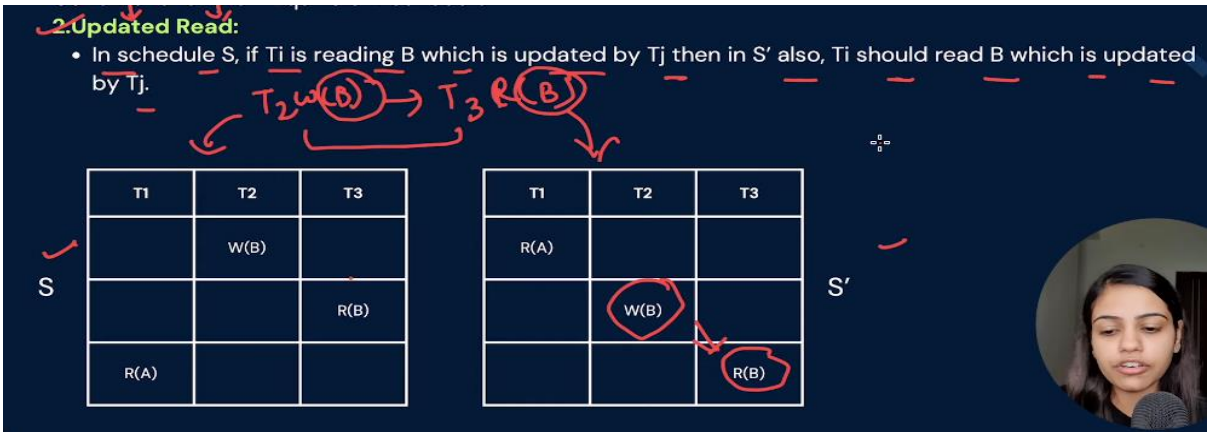


Conditions for a schedule to be view-equivalent:

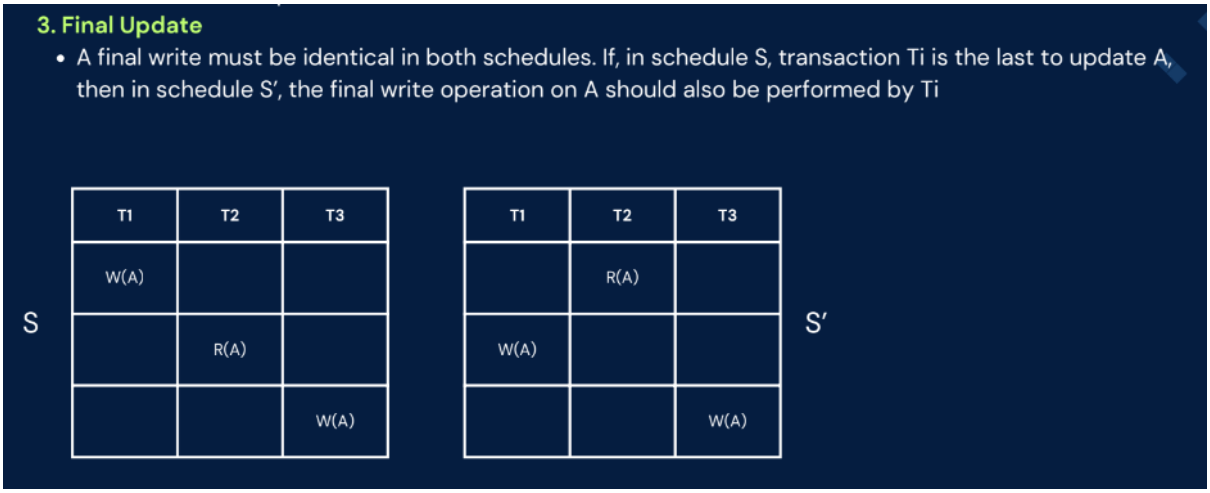
1. **Initial read:** (same for all data like A)
   If for data A, if T1 reads A at first (before any other transaction), then in the view equivalent, only T1 should read A at first.



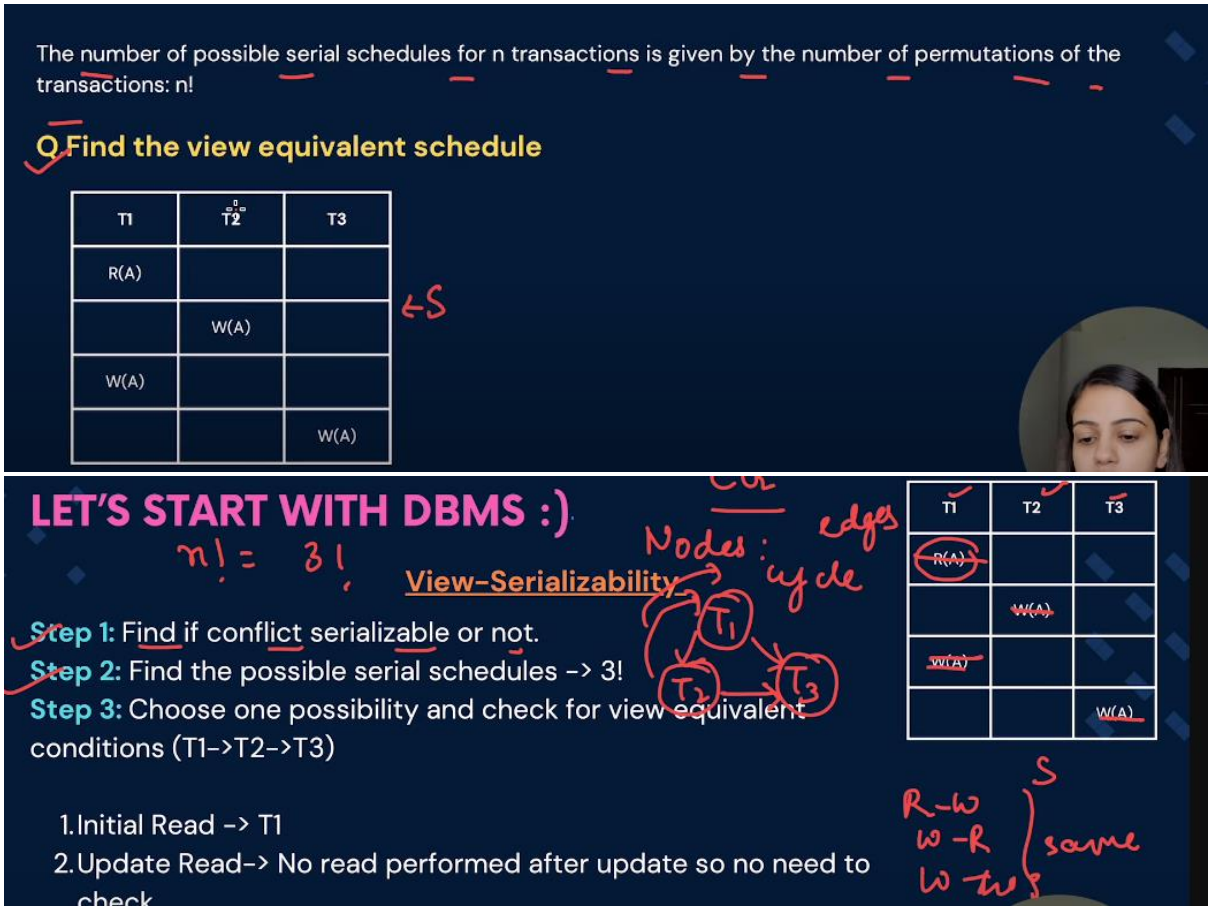
2. **Updated Read:** (so that no consistency issue)



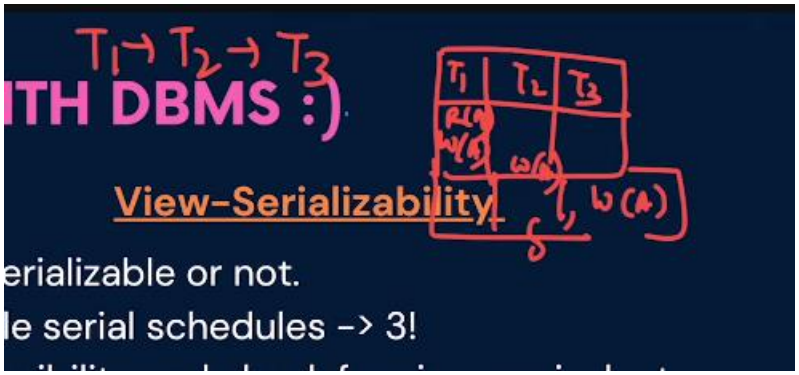
3. **Final Update:** (same for all data like A)



All three conditions should be fulfilled for the schedule with its view equivalent to be serializable.



As there is a cycle in the conflict graph in the image, it is not conflict-serializable.



As we have picked T1->T2->T3, we put all operation of T1, then put of T2 and then T3. And This will form S′ view equivalent.
If S' satifies all condition then, its serializable.

Only one data: A
Initial read on A is T1 in both S and S′
Final read on A is T3 in both S and S′

LET'S START WITH DBMS :)

View-Serializability

Step 1: Find if conflict serializable or not.
Step 2: Find the possible serial schedules -> 3!
Step 3: Choose one possibility and check for view equivalent conditions (T1->T2->T3)

1. Initial Read -> T1
2. Update Read-> No read performed after update so no need to check
3. Final Update-> T3

Step 4: If the view equivalent schedule matches the condition, it is view serializable.

T1	T2	T3
R(A)		
	W(A)	
W(A)		
		W(A)