Method 2: Check the View-Serializability of a Schedule

There's another method to check the view-serializability of a schedule. The first step of this method is the same as the previous method i.e. checking the conflict serializability of the schedule by creating the precedence graph.

Let us take an example of the schedule as follows to check the **View Serializability**

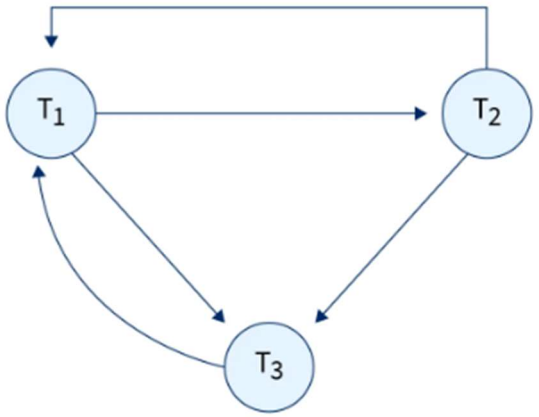
T1	T2	T3
R(X)		
	W(X)	
		R(X)
W(X)		
		W(X)

Giải:

Writing all the conflicting operations:

- R1(X), W2(X) (T1 → T2)
- R1(X), W3(X) (T1 → T3)
- W2(X), R3(X) (T2 → T3)
- W2(X), W1(X) (T2 → T1)
- W2(X), W3(X) (T2 → T3)
- R3(X), W1(X) (T3 → T1)
- W1(X), W3(X) (T1 → T3)

The precedence graph for the above schedule is as follows:



If the precedence graph doesn't contain a loop/cycle, then it is conflict serializable, and thus concludes that the given schedule is consistent. We are not required to perform the **View Serializability** test as a conflict serializable schedule is view-serializable as well.

As the above graph contains a loop/cycle, it does not conflict with serializable. Thus we need to perform the following steps –

Next, we will check for blind writes. If the blind writes don't exist, then the schedule is non-view Serializable. We can simply conclude that the given schedule is inconsistent.

If there exist any blind writes in the schedule, then it may or may not be view serializable.

Blind writes: If a write action is performed on a data item by a Transaction (updation), without performing the reading operation then it is known as blind write.

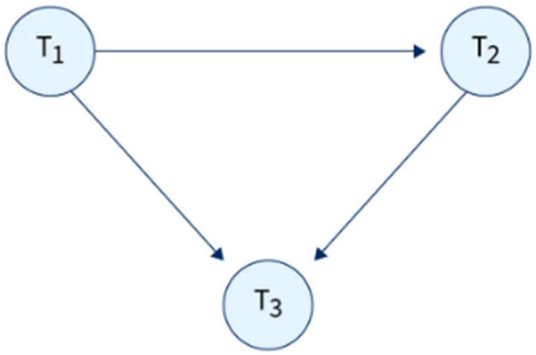
In the above example, transaction T2 contains a blind write, thus we are not sure if the given schedule is view-serializable or not.

Lastly, we will draw a **dependency graph** (đồ thị phụ thuộc) for the schedule. Note: The dependency graph is different from the precedence graph.

Steps for dependency graph:

- Firstly, T1 reads X, and T2 first updates X. So, T1 must execute before the T2 operation. (T1 → T2)
- T3 performs the final updation on X thus, T3 must execute after T1 and T2. (T1 → T3 and T2 → T3)
- T2 updates X before T3, so we get the dependency as (T2 → T3)

The dependency graph is formed as follows:



As no cycles exist in the dependency graph for the example we can say that the schedule is view-serializable.

If any cycle/ loop exists then it is not view-serializable and the schedule is inconsistent. If cycle//loop doesn't exist then it is view-serializable.

Conclusion

- View Serializability is a long process to check the consistency of the given schedule.

- If the given schedule is conflict serializable, then it is view-serializable as well. The opposite is **not** true.
- There are two methods to check if the given schedule is View Serializable, one is **using the serial schedule**, and another **uses blind writes and dependency graphs**.