LET'S START WITH DBMS :)

Serializability and its types

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

LET'S START WITH DBMS :)

Consider nodes as transactions, and edges represent conflicts and detect if the resulting graph has cycles.

T1	T2
R(A)	
W(A)	
Commit	
	R(A)
	W(A)
	Coomit

SERIAL SCHEDULE

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

Nodes: T1, T2
Edges : T1->T2, T2->T1
Cycle : Yes

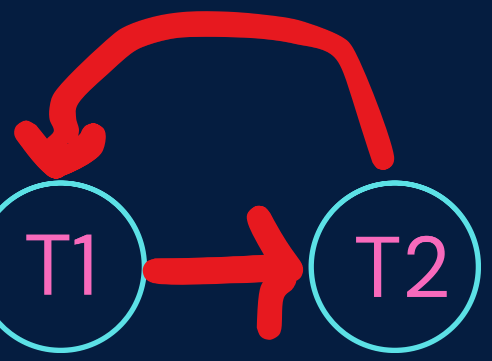
$$\begin{tikzcd} T1 \arrow[r] & T2 \arrow[l, bend right] \end{tikzcd}$$

CONCURRENT SCHEDULE

A concurrent schedule does not always have a cycle.
A concurrent schedule can be conflict-serializable, meaning that it is equivalent to some serial schedule of transactions and its conflict graph does not have any cycles.

LET'S START WITH DBMS :)

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



CONCURRENT SCHEDULE

Nodes: T1, T2
Edges : T1->T2, T2->T1
Cycle : Yes

Now, since a cycle is detected we need to serialize them
So, we use the serializibilty here

- **Conflict-Serializability->** to detect the cycle using conflict graph
- **View-Serializability->** to check if schedule is serializable after a cycle is detected.

Why serializing them?

- To avoid consistentcy issue which may arise because of non-serial execution

LET'S START WITH DBMS :)

Serializability and its types

Types of Serializability

- Conflict-Serializability
- View-Serializability