190031187

11. T_1 : $r_1(x)$; ~~r_1~~(2) ; $w_1(x)$

T_2 : $r_2(2)$; r2(Y) ; w2(2) ; w2(Y) ;

T_3 : r3(X) ; r3(Y) ; W3 (Y) ;

S_1 : $r_1(x)$; r2(2) ; r1(2) ; r3(X) ; r3(Y) ; w1(x) ;
W3(Y) ; r2(Y) ; W2(2) ; W2(Y) ;

S2 r1(x) ; r2(2) ; r3(x) ; r1(2) ; r2(Y) ;
r3(Y) ; w1(x) ; w2(2) ; w3(Y) ; w2(Y) ;

T_1	T_2	T_3
r1(x)	r2(2)	r3(X)
r1(2)	r2(Y)	r3(Y)
W1(x)	W2(2)	W3(Y)
	W2(Y)	

schedule S1

T_1	T_2	T_3
r1(x)		
	r2(2)	
r1(2)		r3(X)
		r3(Y)
w1(x)		
		W3(Y)
	r2(Y)	
	W2(2)	
	W2(Y)	

Possible conflicts occur when T1 writes to x whe T3 is still reading x. However T3 does not write to x so this is ok.

T3 when reads and writes to y before T2 reads and writes to y so this is ok as well since T2 [redacted] reads and writes to z. it is also ok. that T1 reads z but does not write. This schedule is serializable because there are no cycles.

Schedule : S2

T1	T2	T3
r1(x)		
	r2(z)	
		r3(x)
r1(z)		
	r2(Y)	
		r3(Y)
w1(x)		
	w2(z)	
	w3(Y)	
	w2(Y)	

This [redacted] schedule is non-serializable and contains a major conflict. Both T2, T3 are accessing Y when T3 writes to it.

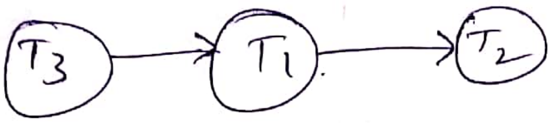
∴ when T2 writes to Y, the transaction for T3 is lost and overridden.

Schedule S1

It is a serializable schedule because

- T_1 only reads X ($r_1(X)$) which is not modified either by T_2 or T_3
- T_3 reads X ($r_3(X)$) before T_1 modifies it ($W_1(x)$), T_2 reads Y ($r_2(Y)$) and writes it ($W_2(Y)$) only after T_3 has written to it ($W_3(Y)$)

Thus serializable graph is



Schedule : S2

It is not a serializable schedule because

- T_2 reads Y ($r_2(Y)$) which is then read and modified by T_3 ($W_3(Y)$)
- T_3 reads Y ($r_3(Y)$) which then modified before T_2 modifies Y ($W_2(Y)$)

In the above order T_3 interferes in the execution of T_2 which makes the schedule non-serializable