20.22

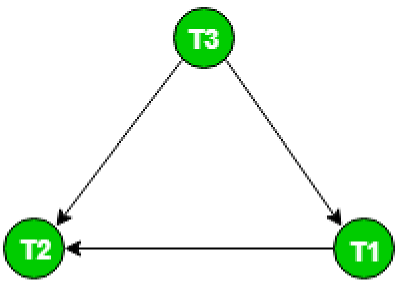
Which of the following schedules is (conflict) serializable? For each serializable schedule, determine the equivalent serial schedules.

a. $r_1(X); r_3(X); w_1(X); r_2(X); w_3(X);$
   Not serializable
b. $r_1(X); r_3(X); w_3(X); w_1(X); r_2(X);$
   Not serializable
c. $r_3(X); r_2(X); w_3(X); r_1(X); w_1(X);$
   $T_2 \rightarrow T_3 \rightarrow T_1$
d. $r_3(X); r_2(X); r_1(X); w_3(X); w_1(X);$
   Not serializable

20.23

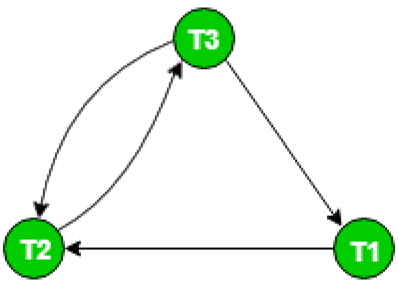
Consider the three transactions T_1, T_2, and T_3, and the schedules S_1 and S_2 given below. Draw the serializability (precedence) graphs for S_1 and S_2, and state whether each schedule is serializable or not. If a schedule is serializable, write down the equivalent serial schedule(s).

- T_1: $r_1(X); r_1(Z); w_1(X);$
- T_2: $r_2(Z); r_2(Y); w_2(Z); w_2(Y);$
- T_3: $r_3(X); r_3(Y); w_3(Y);$
- S_1: $r_1(X); r_2(Z); r_1(Z); r_3(X); r_3(Y); w_1(X); w_3(Y); r_2(Y); w_2(Z); w_2(Y);$



$T_3 \rightarrow T_1 \rightarrow T_2$

- S_2: $r_1(X); r_2(Z); r_3(X); r_1(Z); r_2(Y); r_3(Y); w_1(X); w_2(Z); w_3(Y); w_2(Y);$



21.25

Apply the timestamp ordering algorithm to the schedules in Figures 20.8(b) and (c), and determine whether the algorithm will allow the execution of the schedules.

(b)

T1	T2	T3	Conflict?	RTS/WTS Updates
	rZ		No	RTS(Z) = 2
	rY		No	RTS(Y) = 2
	wY		No	WTS(Y) = 2
		rY	3 > 2 No	RTS(Y) = 3
		rZ	3 > 0 No	RTS(Z) = 3
rX			No	RTS(X) = 1
wX			No	WTS(X) = 1
		wY	3 > 2 No	WTS(Y) = 3
		wZ	3 > 0 No	WTS(Z) = 3
	rX		2 > 1 No	RTS(X) = 2
rY			1 < 3 Yes	
wY			T1 aborted	
	wX		2 > 1 No	WTS(X) = 2

(c)

T1	T2	T3	Conflict?	RTS/WTS Updates
		rY	No	RTS(Y) = 3
		rZ	No	RTS(Z) = 3
rX			No	RTS(X) = 1
wX			No	WTS(X) = 1
		wY	3 > 0 No	WTS(Y) = 3
		wZ	3 > 0 No	WTS(Z) = 3
	rZ		2 < 3 Yes	
rY			1 < 3 Yes	
wY			T1 aborted	
	rY		T2 aborted	
	wY		T2 aborted	

T1	T2	T3	Conflict?	RTS/WTS Updates
	rX		T2 aborted	
	wX		T2 aborted	

21.27

Why is two-phase locking not used as a concurrency control method for indexes such as B^+-trees?

- It can lead to deadlocks.

22.21

Suppose that the system crashes before the [read_item, t_3, A] entry is written to the log in Figure 22.1(b). Will that make any difference in the recovery process?

No, it will still have to roll back the same transactions because T_3 still hasn't reached its commit point and T_2 still read the value of B written by T_3

22.22

Suppose that the system crashes before the [write_item, T_2, D, 25, 26] entry is written to the log in Figure 22.1(b). Will that make any difference in the recovery process?

T_2 and T_3 will still both be rolled back, but T_2 will roll back because it didn't reach its own commit point, not just because it reads a value written by T_3.

22.23

Figure 22.6 shows the log corresponding to a particular schedule at the point of a system crash for four transactions T_1, T_2, T_3, and T_4. Suppose that we use the immediate update protocol with checkpointing. Describe the recovery process from the system crash. Specify which transactions are rolled back, which operations in the log are redone and which (if any) are undone, and whether any cascading rollback takes place.

T_1 and T_4 both fully ran and committed, so they don't need to be rolled back at all. T_2 and T_3 both did not reach their commit points, so their transactions will be rolled back. However neither one of them read or wrote to any uncommited values, so their operations can all just be redone as normal.