Part 3: Two-Phase Locking (20 points)

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

a.) Now modify the above schedule by adding locks, which may block some transactions from doing their operations until the lock is released. You'll need to rewrite the above schedule in a table form. (The lecture slides show how to represent blocking in your schedules.)

Use two-phase locking (doesn't need to be "strict") in your modified schedule to ensure a conflict-serializable schedule for the transactions above.

Use the notation L(A) to indicate that the transaction acquires the lock on element A and U(A) to indicate that the transaction releases its lock on.

T1	T2	T3
L(A)		

L(B)		
R(A)		
W(A)		
U(A)		
		L(A)
		R(A)
		W(A)
	L(A) Blocked…	
R(B)		
		L(B) Blocked…
W(B)		
U(B)		
		…Granted L(B)
		U(A)
	…Granted L(A)	
	R(A)	
	L(B) Blocked…	
		R(B)
		W(B)
		U(B)
	…Granted L(B)	
	U(A)	
	R(B)	
	U(B)	
	commit	

commit		
		commit

b.) If 2PL ensures conflict-serializability, why do we need strict 2PL? Explain briefly.

Given that two-phase locking ensures conflict-serializability, the reason why we need strict two-phase locking is because it offers us something that two-phase locking cannot; recoverability. Without strict two-phase locking, unrecoverable schedules are possible, due to the fact that someone can rollback their transaction after you have already committed. However, under strict two-phase locking, all unlocks are done together with the COMMIT or ROLLBACK, thus it is impossible to have such an unrecoverable schedule as explained above. However, strict two-phase locking usually turns schedules into purely serial schedules, which is time consuming when compared to other techniques and/or isolation levels.