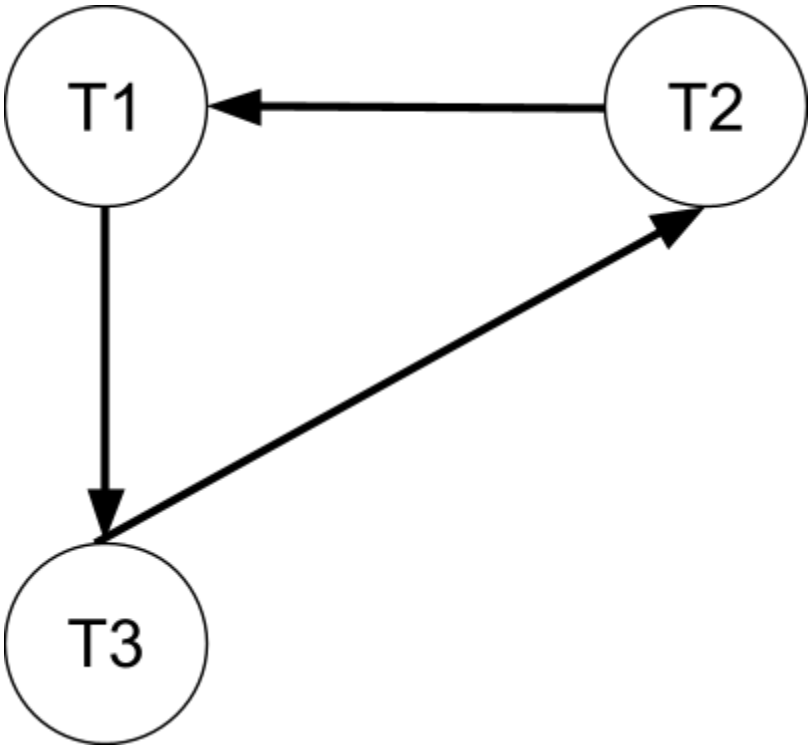
Problem Set 3, Part I

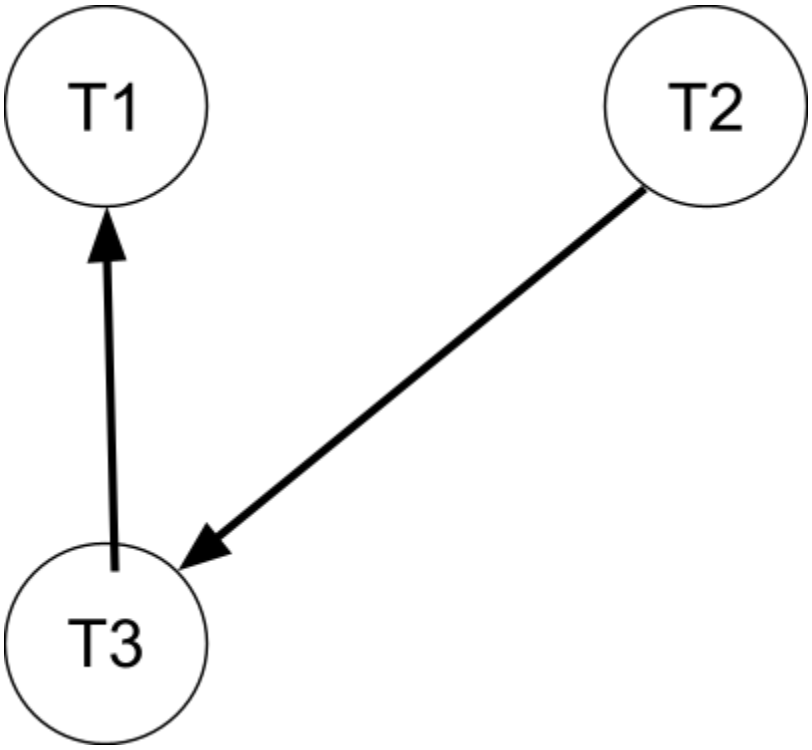
Problem 1: Conflict serializability

schedule 1
precedence graph:



conflict serializable (yes or no)? No
equivalent serial schedule or explanation: Since have a cycle going from 3 -> 2-> 1, we cannot have a conflict serializable schedule, thus we cannot have a serial schedule either. We cannot replace this schedule with an equivalent serial one since there is no way we can perform operations from a single schedule followed by another one. Say we replaced the current schedule with T3, T2, T1, we can clearly see that T3's read of C will read the wrong value here because **T1 will write to C after this read.** There is no way to change this because if we tried a different iteration, say, T2, T1, T3, once again because T2 writes to A and T3 is supposed to ready A before the write, we will get unexpected behavior.

schedule 2
precedence graph:



conflict serializable (yes or no)? Yes
equivalent serial schedule or explanation: The equivalent schedule is T2, T3, T1. That is, we can treat T2, T3, T1 as a serial schedule without having any conflicts.

Problem 2: Two-phase locking and isolation

part	*yes or no?*	*explanation*
2.1	no	Unlocking before acquiring the lock for A and C would violate 2PL since the transaction would go into the lock release phase.
2.2	no	T1 would be an exclusive lock request to write to A however T2 has a shared lock for A thus without unlocking the shared lock, we cannot acquire one for writing.
2.3	no	Strict locking occurs when all transactions hold the exclusive lock until committing. Here we commit after releasing the lock.
2.4	yes	Since there is no dirty read in this schedule, we can ensure recoverability. [Note that we write to A in T1 AFTER reading A, so a failure would not affect recoverability here]
2.5	no	If T2 fails, then we would have to rollback T1 since T2 writes to B and T1 reads it later on.

Problem 3: Lock modes

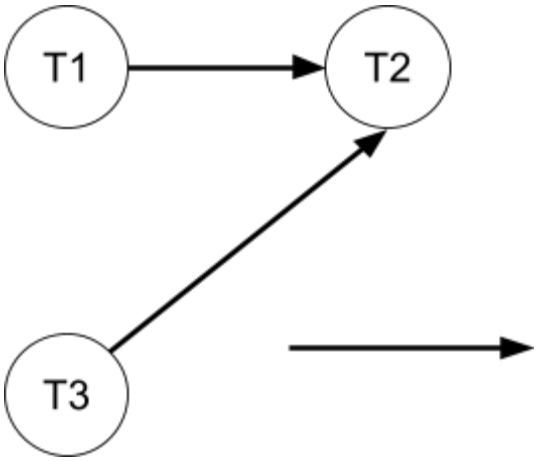
request	***granted or denied?***	***explanation***
xl2(B)	denied	Wait for T4 to release the shared lock
sl3(B)	granted	You can share a shared lock to read B since T4 has a shared lock to read T4
sl4(A)	denied	T3 has an update lock however we cannot share it
ul2(A)	denied	T2 cannot get an update lock since T3 has one
ul3(B)	granted	A share lock can share an update lock thus T4 allows T3 to get an update lock
ul4(C)	denied	T2 has an exclusive lock for C thus we cannot get an update lock

Problem 4: Deadlock detection

sequence 1: schedule

T1	T2	T3
sl(A); r1(A) xl(B); w1(B)		
	sl(B); Denied (wait for T1)	
		sl(C); r3(C)
xl(A); (upgrade from read) w1(A) Commit u(A) u(B)		
	Xl(c); denied (wait for T3)	
		Xl(A); W3(A) commit

waits-for graph (if deadlock occurs):



sequence 2: schedule

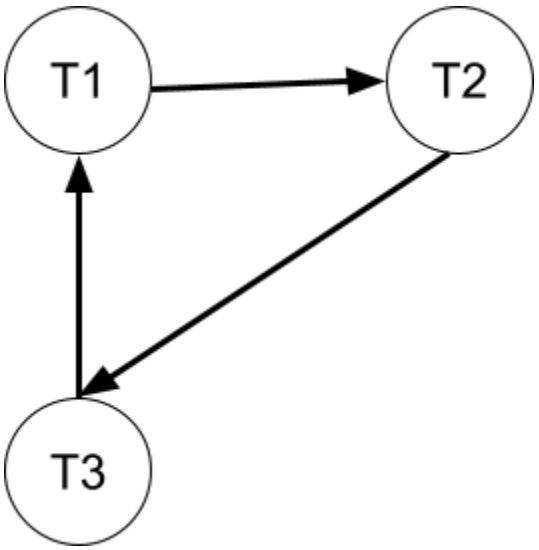
T1	T2	T3
		xl(A); W3(A)

sl(C); r1(C);
sl(A);
denied(wait
for T3)		

	sl(B); r2(B);
	xl(C);
	denied
	(wait for T1)	

		sl(C); r3(C) xl(B); denied (Wait for

waits-for graph (if deadlock occurs):



		T2)

Deadlock occurs