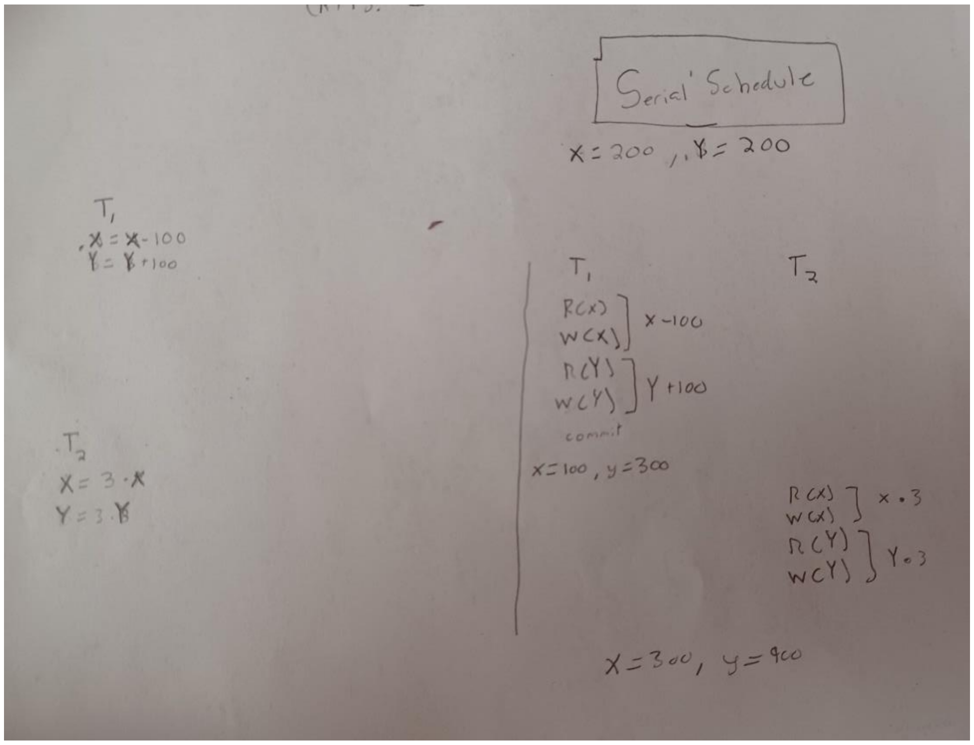
Author: Samuel Rettig

A)



In this serial schedule, T1 writes and reads without any issues. This results in X losing one hundred while Y gains 100. Similarly, when T2 rolls around it does not interfere with the T1 schedule, allowing for each to be multiplied by 3.

B)

T1	T3
R(X)	
W(X)	
	R(X)
	W(X)
	Commit
R(Y)	
W(Y)	
commit	

The Write-Read conflict happens in two places in this schedule. First is when T3 initially starts. The write to X just happened, however, has not been committed. This causes a dirty read on/for T3. Similarly, this happens when the schedule goes back to T1.

Shared Lock:

T1	T2	Description	Step
S(X)		Get a shared lock on X	1
R(X)			
X(X)		Get an exclusive lock on X	2
W(X)			
	S(X)	Request exclusive lock on X (T1 has it), cannot get	3
	~~R(X)~~	deferred	
	~~W(X)~~	deferred	
S(Y)		Get a shared lock on Y	4
R(Y)			
X(Y)		Get an exclusive lock on Y	5
W(Y)			
		Releases locks	
	S(X)	Get a shared lock on X	6
	R(X)		
	X(X)	Get an exclusive lock on X	7
	W(Y)		
		Release locks	

C)

T1	T2
	R(X)
	R(Y)
R(X)	
W(X)	
R(Y)	
	W(Y)
	W(X)
	commit
commit	

Here the Read-Write Conflict happens when R(Y) on T1 goes to W(Y) in T2. This happens because T1 is trying to read while T2 is trying to execute.

T1	T2	Description	Step
	S(X)	Get a shared lock on X	1
	R(X)		
	S(Y)	Get a shared lock on Y	2
	R(Y)		
R(X)		Request exclusive lock on X (T2 has it), cannot get. Thus it waits.	5
W(X)		Cannot write w/out reading	5
R(Y)		See above	7
	X(Y)	Get an exclusive lock on Y	3
	X(X)	Get an exclusive lock on X	4
		(After this T2 commits, releasing locks)	

Explanation: T2 via the locks on both X and Y commit first before T1 can make their own locks. Thus, the read-write error never occurs as the lock does not allow itself to be released until it is safe to do so.