Distributed Data – HW2

Tomer Grinberg & Hadar Sugarman

1. *The claim is True*

Assume by contradiction that there exists a time t where $n \geq 2$ *transactions are deadlocked.*
First note that all transactions have a unique timestamp, ts*, hence there is a total order on the* ts'*. denote the transaction with minimal ts among the dealocked transactions as* T_i*, i. e for all* $j \neq i : i, j \in \{k \mid T_k \text{ is deadlocked}\}, ts(T_i) < ts(T_j)$*.*
By definition of deadlock, T_i *is waiting for a lock on some item* x*, which is locked by some other transaction* T_j*. By definition of the locking procedure,* T_j *is forced to release the lock in order to pass it to* T_i*, since* $ts(T_i) < ts(T_j)$*. Since* $ts(T_i)$ *is minimal, no other transaction can take its lock. by 2PLP protocol, once* T_i *is finished with* x*, it'll release the lock voluntarily and won'tbe able to request the lock again. notice this holds for all resources available, and so* T_i *is able to complete all its actions. in contradiction to the fact that* T_i *is deadlocked.*
$\Rightarrow$ *protocol 2PLP prevents deadlocks.* ■

2. *The claim is False*

first define cascading abort:
A situation in which the abort of one transaction forces the abort of another transaction to prevent the second transaction from reading invalid (uncommitted)data.
Consider the following example, with 2 transactions T_1 *and* T_2 *and some item* x*.*

	T_1	T_2
t_1	*Start*	
t_2		*Start*
t_3		*request lock* $L(x)$
t_4		$Write(x)$
t_5	*request lock* $L(x)$	
t_6		*give up lock* $L(x)$
t_7	$read(x)$	
t_8	*doing something*	*abort for some reason*
t_9	*abort*	

in the given example, $ts(T_1) < ts(T_2)$ *so when* T_1 *requests the lock on* x *at* t_3*, by protocol strict2PLP,* T_2 *is forced to release the lock, then* T_1 *reads* x.
Then at t_8 T_2 *has to abort which forces* T_1*to abort since the changes* T_2 *made to* x *no longer hold true.*
$\Rightarrow$ *strict2PLP does not prevent cascading abort*■