Week 9: ACID, Transactions and Concurrency Control

ACID – Atomicity, Consistency, Isolation, Durability
Isolation Example: Integrity constraint – Maintain A = B
(Source http://www.csc.liv.ac.uk/~valli/Comp302/Concurrency-addition.pdf)

T1:	T2:
Read(A) A ← A + 100 Write(A) Read(B) B ← B + 100 Write(B)	Read(A) A ← A* 2 Write(A) Read(B) B ← B*2 Write(B)

Serial Schedule

Read(A); A ← A + 100 Write(A) Read(B); B ← B + 100 Write(B)	
	Read(A); A ← A* 2 Write(A) Read(B); B ← B*2 Write(B)

Equivalent schedule that is not serial

Read(A); A ← A + 100 Write(A)	
	Read(A); A ← A* 2 Write(A)
Read(B); B ← B + 100 Write(B)	
	Read(B); B ← B*2 Write(B)

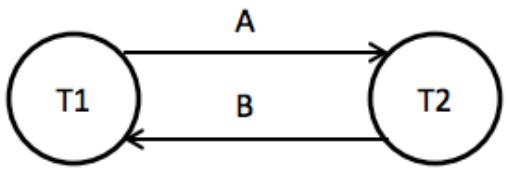
Conflict Serializability: Sufficient condition for serializability.
Two operations in different transactions conflict if they are operations to the same data item and one of them is a write. Types of conflict:

- WR: read uncommitted data
- RW: unrepeatable reads
- WW: overwriting uncommitted data

To check if S is conflict serializable: Construct a dependency graph - nodes are transactions, and edge from T1 to T2 iff operation O1 in T2 conflicts with O2 and T2 and O1 happens earlier than O2 in S. If the dependency graph is acyclic, the schedule is conflict serializable.

Example:

Read(A); A ← A + 100 Write(A)	
	Read(A); A ← A* 2 Write(A) Read(B); B ← B*2 Write(B)
Read(B); B ← B + 100 Write(B)	



Enforcing Conflict Serializability Using 2-phase locking

Lock types: shared (anyone with shared lock can read) and exclusive (can write)

Regular 2-PL:

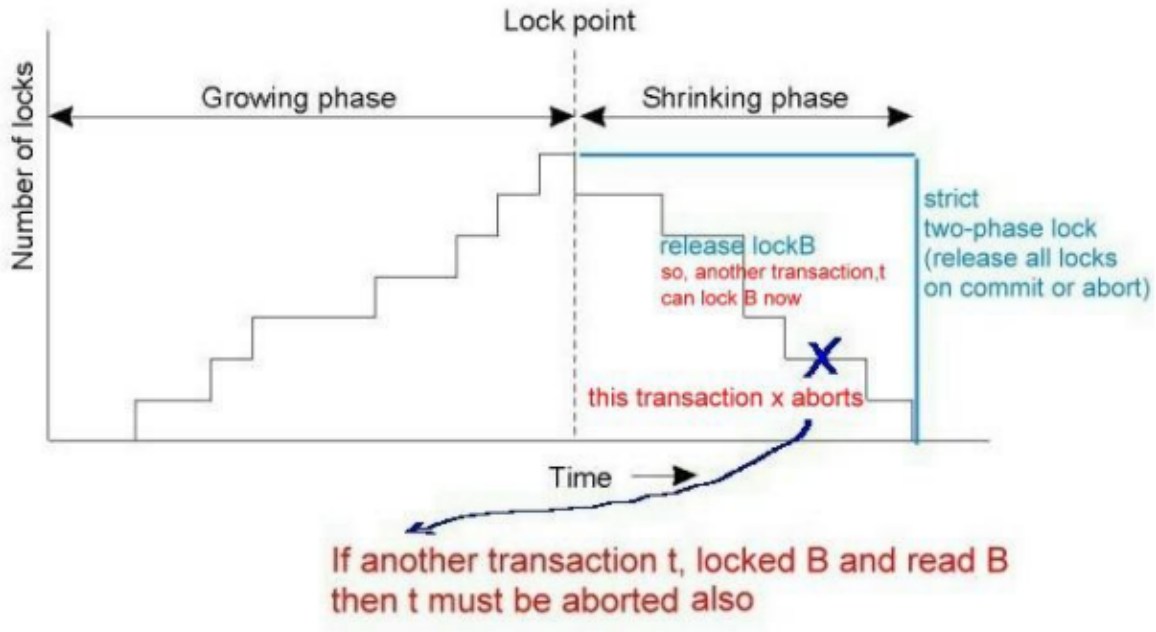
1) Each transaction must obtain
   - S (shared) or an X (exclusive) lock on object before reading
   - X (exclusive) lock on object before writing.
2) A transaction cannot request additional locks once it releases locks

	S	X
S	✓	×
X	×	×

Strict-2PL: Release all locks at end of transaction. That way, no one can will have uncommitted data (and have to abort if this transaction aborts!).

Problem with regular 2-PL: Cascading aborts

(Source: http://dslab.ee.ncku.edu.tw/~tsai/ds/ds-synchronization/2pl.jpg)



Deadlocks

Deadlock: Lack of progress, when two (or more) actions are waiting for the other to finish

Deadlock Avoidance: use priorities for transactions based on start timestamps

Wait-die: if T1 is older, wait for T2; else abort T1

Wound-wait: if T1 is older, T2 aborts; else T1 waits

Deadlock Detection:

Graph of actions that are waiting for each other. If there is a cycle, then you have deadlock.

1. Consider the following schedules:

T1		R(A)	W(A)	R(B)					
T2					W(B)	R(C)	W(C)	W(A)	
T3	R(C)								W(D)

(a) Draw the dependency graph (precedence graph) for the schedule.

(b) Is the schedule conflict serializable? If so, what are all the (conflict) equivalent serial schedules? If not, why not?

T1	R(A)		R(B)				W(A)	
T2		R(A)		R(B)				W(B)
T3					R(A)			
T4						R(B)		

(a) Draw the dependency graph (precedence graph) for the schedule.

(b) Is the schedule conflict serializable? If so, what are all the (conflict) equivalent serial schedules? If not, why not?

2) a. What will be printed in the following execution (B=3, F=300)?

Lock_X(B)	
Read(B)	
B = B*10	
Write(B)	Lock_S(B)
Lock_X(F)	
F = B*100	
Write(F)	
Unlock(B)	
Unlock(F)	
	Lock_S(F)
	Read(F)
	Read(B)
	Print(F+B)
	Unlock(B)
	Unlock(F)

b. Does the execution use: (a) 2PL or (b) Strict 2PL?