CS186 Discussion #9

(Transactions & Concurrency Control)

NOTE: This discussion uses animations, which won't show up on the PDF version.

Transactions

- Sequence of reads and writes:
  - `bank_account := bank_account - 500`
  - `wallet := wallet + 500`
- Either all instructions execute or none
- No other instruction sees only a part of this execution

ACID

- Atomicity: either none or all instructions executed
  - Enforced through logging
- Consistency: database remains in consistent state afterwards
  - Enforced through integrity constraints
- Isolation: runs as if it is only transaction
  - Enforced through serializability and locks
- Durability: committed changes are never lost
  - Enforced through logging

Isolation

- Could just execute 1 transaction at a time
  - “Serial schedule”
  - Slow

- Maximize parallelism while maintaining sense of isolation
  - Concurrency control

Serializability

- Serial schedule: run 1 transaction at a time
- Equivalent schedules: schedules with same transactions, same final state
- Serializable schedule: schedule equivalent to a serial schedule

T1	wallet1 := wallet1 - 200	acct := acct + 200		
T2			acct := acct - 500	wallet2 := wallet2 + 500

T1	wallet1 := wallet1 - 200	acct := acct + 200		
T2			acct := acct - 500	wallet2 := wallet2 + 500

Serial

T1	wallet1 := wallet1 - 200			acct := acct + 200
T2		acct := acct - 500	wallet2 := wallet2 + 500	

T1	wallet1 := wallet1 - 200			acct := acct + 200
T2		acct := acct - 500	wallet2 := wallet2 + 500	

Serializable

T1	wallet1 := wallet1 - 200			acct := acct + 200
T2		acct := acct - 500	wallet2 := wallet2 + 500	

T1	R(w1)	W(w1)					R(acct)	W(acct)
T2			R(acct)	W(acct)	R(w2)	W(w2)		

We simplify to this format, since we are more concerned with *when* reads and writes are happening.

T1	R(w1)	W(w1)	R(acct)					W(acct)
T2				R(acct)	W(acct)	R(w2)	W(w2)	

T1	R(w1)	W(w1)	R(acct)					W(acct)
T2				R(acct)	W(acct)	R(w2)	W(w2)	

Not serializable

Conflicts

- Two operations in different transactions on the same object where at least one is a write

T1	R(w1)	W(w1)	R(acct)					W(acct)
T2				R(acct)	W(acct)	R(w2)	W(w2)	

3 conflicts

Conflict Serializable

- Conflict serializable: equivalent to some serial schedule with the conflicts in the same order.

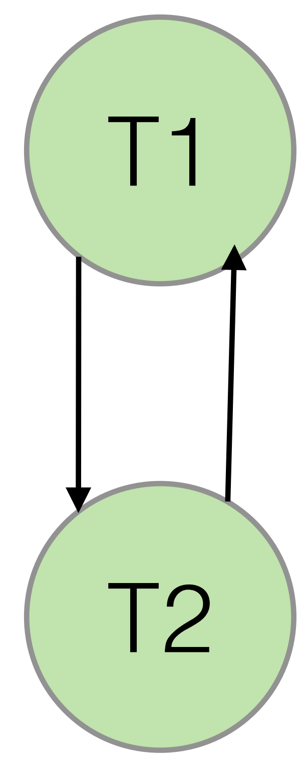
T1	R(w1)	W(w1)					R(act)	W(act)
T2			R(act)	W(act)	R(w2)	W(w2)		

T1					R(w1)	W(w1)	R(act)	W(act)
T2	R(act)	W(act)	R(w2)	W(w2)				

Dependency Graph

- One node per transaction
- Edge from T_i to T_j if operation O_i conflicts with O_j and O_i appears earlier in the schedule

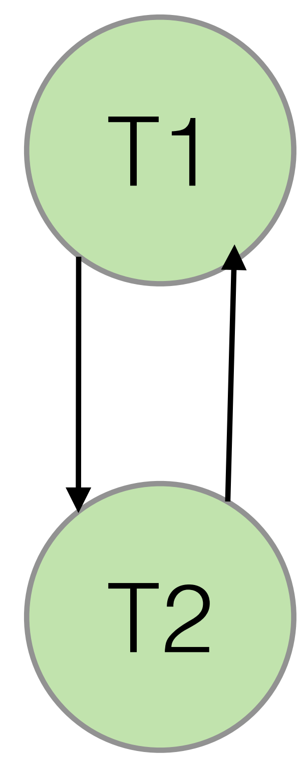
T1	R(1)	W(1)	R(a)					W(a)
T2				R(a)	W(a)	R(2)	W(2)	



Dependency Graph

- Theorem: Schedule is conflict serializable iff its dependency graph is acyclic

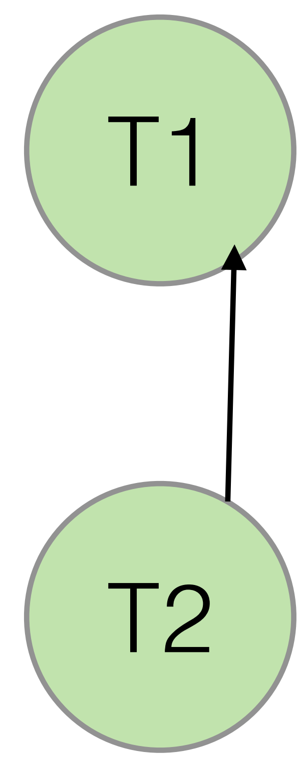
T1	R(1)	W(1)	R(a)					W(a)
T2				R(a)	W(a)	R(2)	W(2)	



Dependency Graph

- Theorem: Schedule is conflict serializable iff its dependency graph is acyclic

T1	R(1)	W(1)					R(a)	W(a)
T2			R(a)	W(a)	R(2)	W(2)		



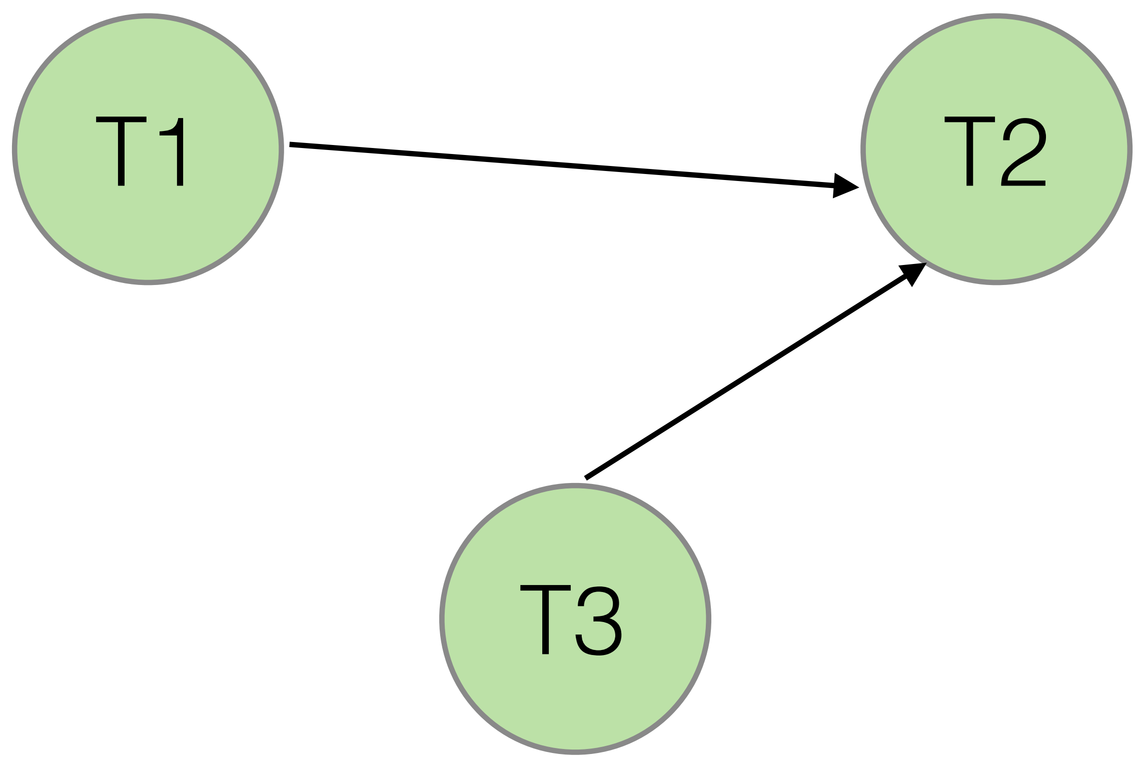
Serializability vs. Conflict Serializability

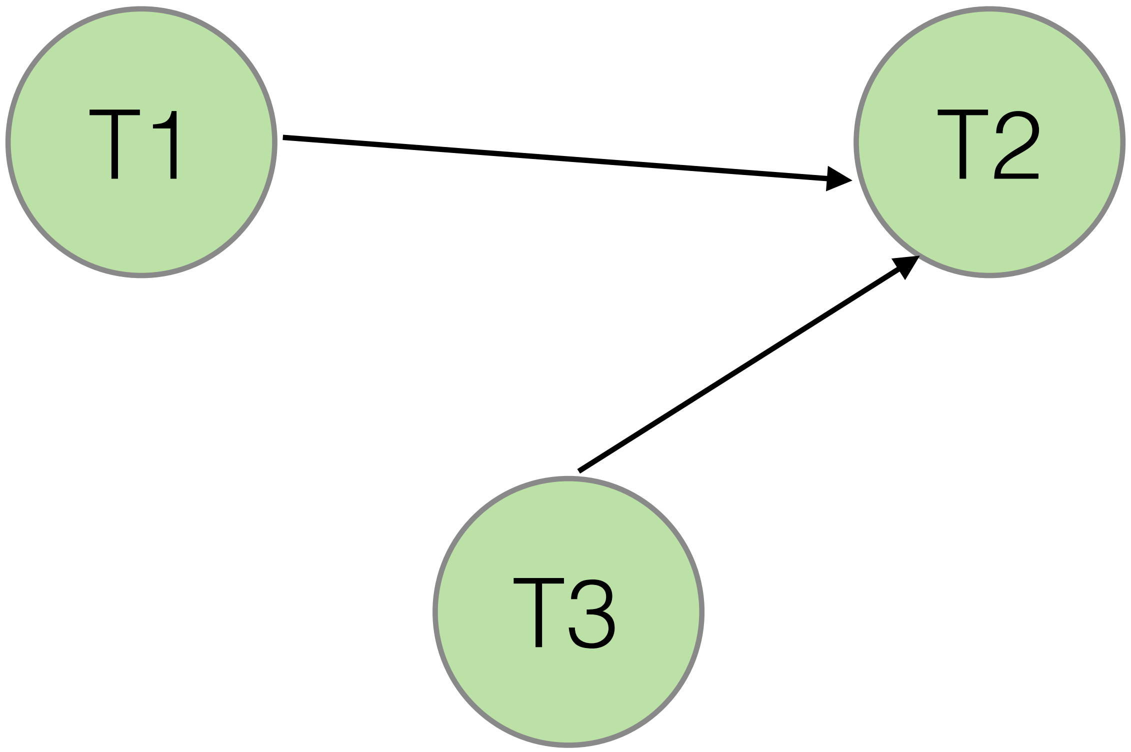
- It is much harder to verify serializability
- Any conflict serializable schedule is serializable
- Some serializable schedules are not conflict serializable

Worksheet Page 1

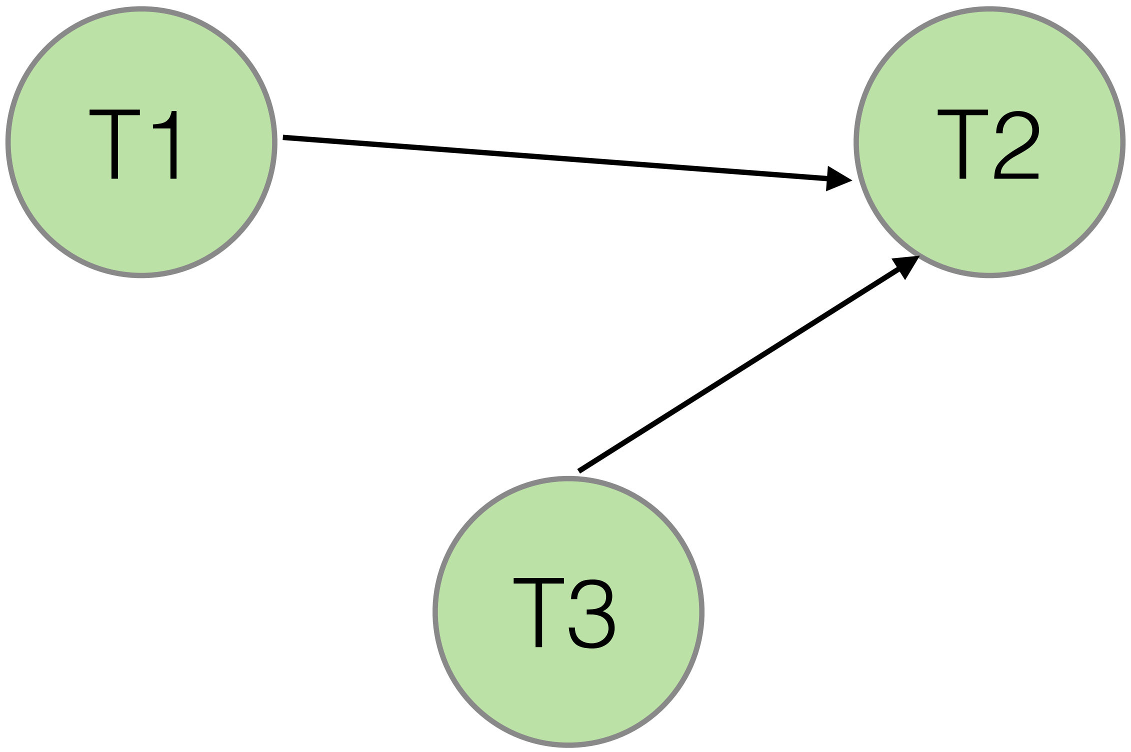
T1		R(A)	W(A)	R(B)					
T2					W(B)	R(C)	W(C)	W(A)	
T3	R(C)								W(D)

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





Conflict serializable

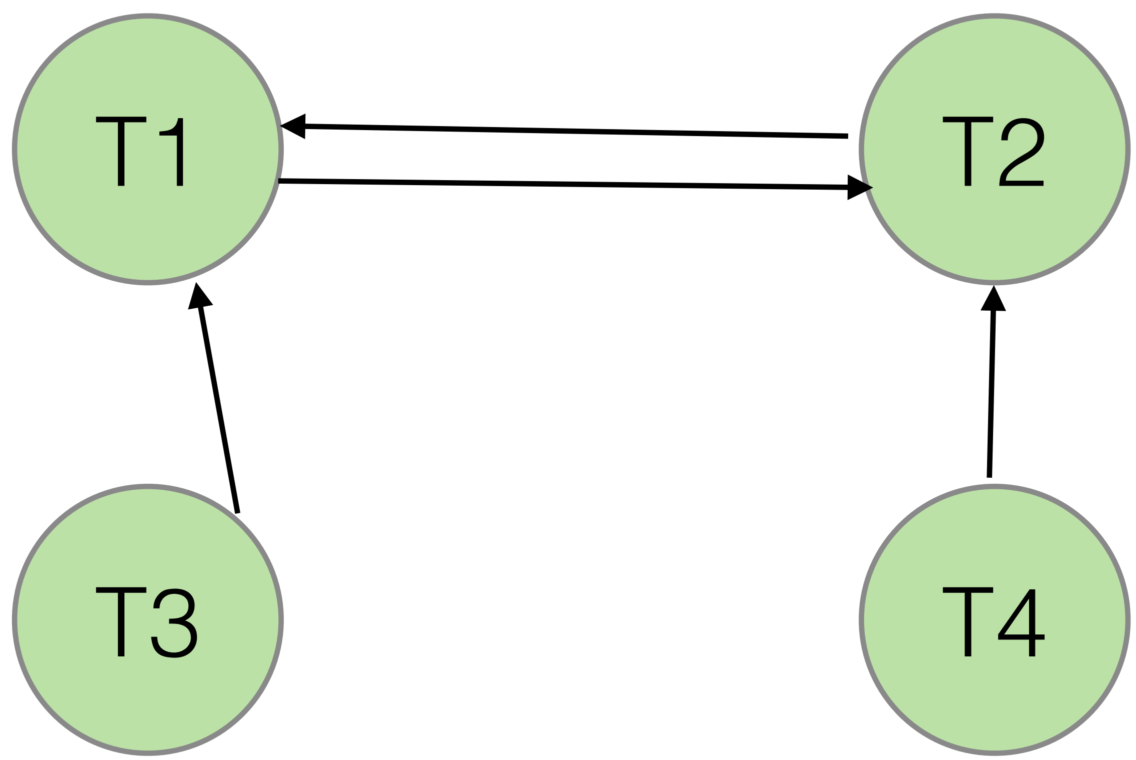


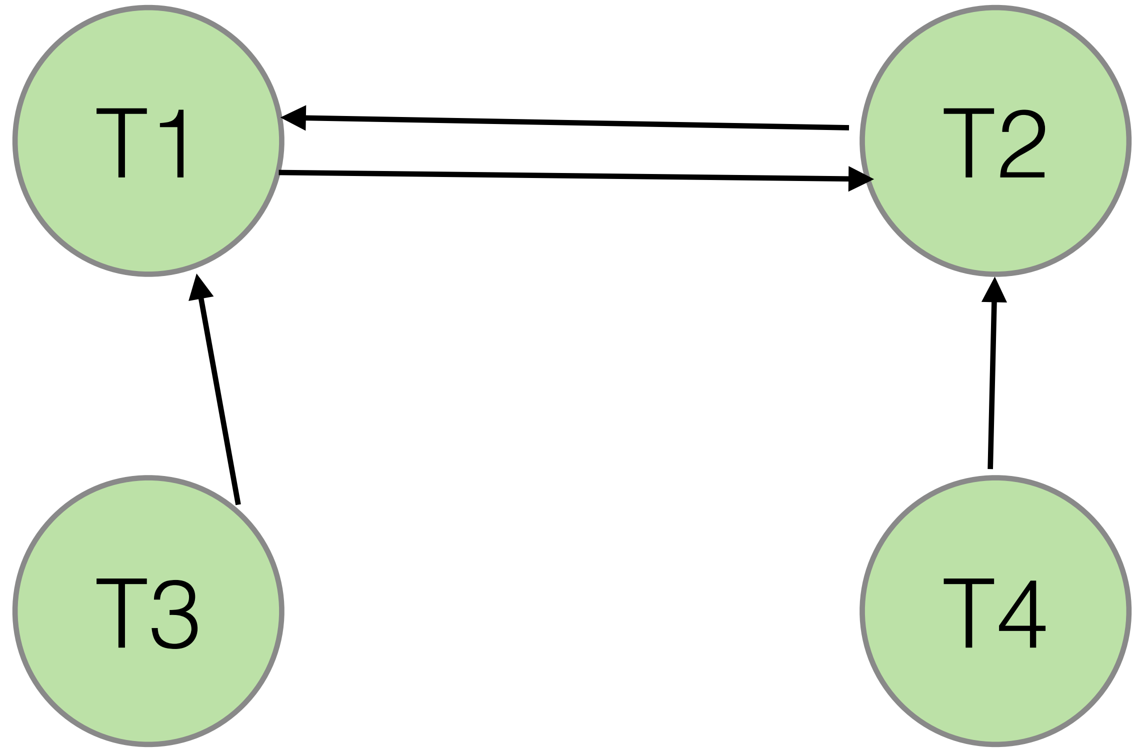
Conflict serializable

Equivalent serial schedules:
T3, T1, T2
T1, T3, T2

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

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





Not conflict serializable,
because cycle in precedence graph.

Locks

- Use locks to control access to objects
- Shared lock: multiple transactions can have shared lock on same item (e.g., reading)
- Exclusive lock: only one lock on item (e.g., writing)

Two-Phase Locking (2PL)

- Rule: Once you release a lock, you may never acquire a new lock.

- Ensures conflict serializability

Lock_X(w1)

Lock_X(acct)

Unlock(acct)
Unlock(w1)

T1	R(w1)	W(w1)					R(acct)	W(acct)
T2			R(acct)	W(acct)	R(w2)	W(w2)		

Lock_X(acct)

Lock_X(w2)

Unlock(acct)
Unlock(w2)

Strict 2PL

Cascading abort

T1	R(A)	W(A)			Abort
T2			R(A)	W(A)	

Strict two-phase locking: All locks held by transaction are only released at the end of the transaction.

Strict 2PL

Cascading abort

T1	R(A)	W(A)	Abort		
T2				R(A)	W(A)

Strict two-phase locking: All locks held by transaction are only released at the end of the transaction.

Deadlocks

- Dealing with deadlocks:
  - Prevention - stop from occurring
  - Detection - stop while occurring
  - In practice: Timeouts

Time	Transaction 1	Transaction 2
1	Lock_X(A) (granted)	
2		Lock_X(B) (granted)
3	Lock_X(B) (waiting)	
4		Lock_X(A) (waiting)
5	…	…

Deadlock Prevention

- Disallow deadlocks from ever occurring
- Use timestamps of transactions to determine which to abort
- If you restart, you restart with original timestamp

Deadlock Prevention

- Two transactions T_{old} and T_{young}
- Wait-Die
  - If T_{old} waiting for a lock from T_{young}, just waits
  - If T_{young} waiting for lock from $T_{old,}$ aborts himself
- Wound-Wait
  - If T_{old} waiting for a lock from $T_{young,}$ aborts T_{young}
  - If T_{young} waiting for lock from T_{old}, just waits

Deadlock Detection

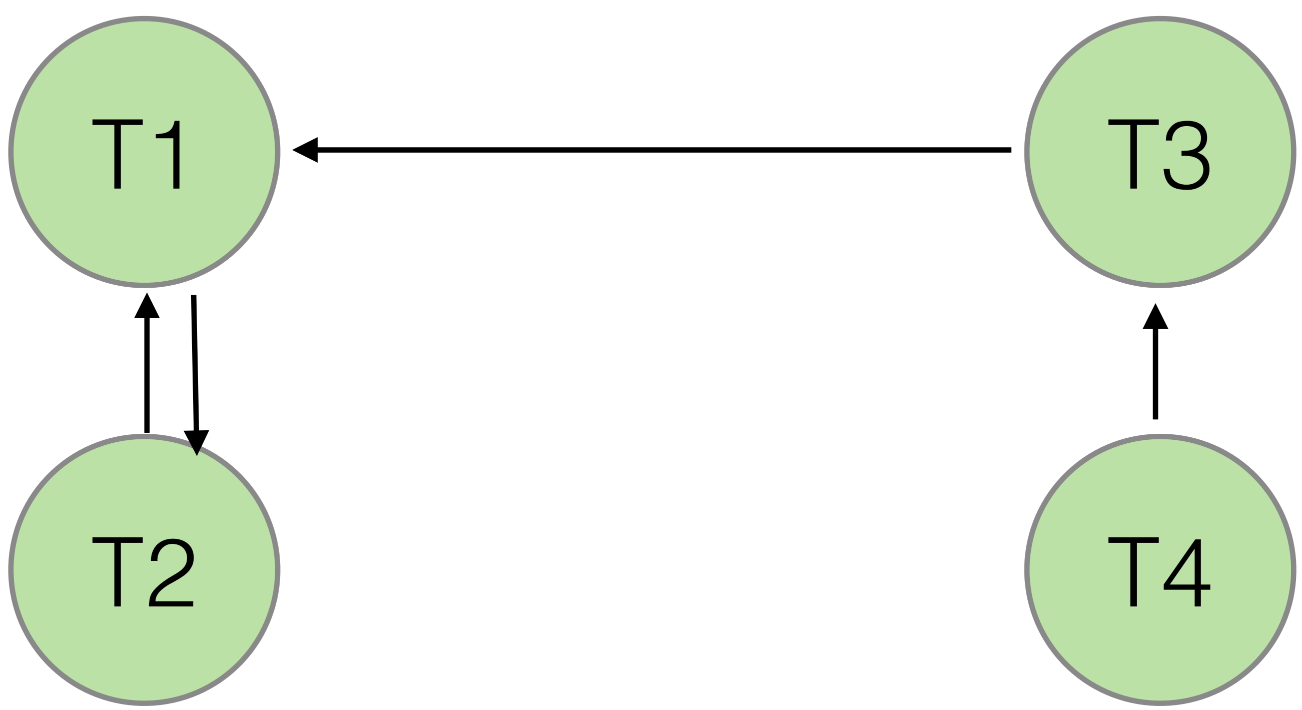
- Create waits-for graph of all transactions
- If cycle exists, abort one of transactions in cycle

Deadlock Detection

T1	X(A)			S(B)			
T2		X(B)	S(A)				
T3					X(C)	S(A)	
T4							S(C)

Deadlock Detection

T1	X(A)			S(B)			
T2		X(B)	S(A)				
T3					X(C)	S(A)	
T4							S(C)



Worksheet Page 2

(B=3, F=300)

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

(B=3, F=300)

Lock_X(B)	
Read(B) 3	Lock_S(F)
B = B*10	Read(F)
Write(B)	Unlock(F)
Lock_X(F)	Lock_S(B)
F = B*100	
Write(F)	
Unlock(B)	
Unlock(F)	
	Read(B)
	Print(F+B)
	Unlock(B)

(B=3, F=300)

Lock_X(B)	
Read(B) 3	Lock_S(F)
B = B*10 30	Read(F) 300
Write(B)	Unlock(F)
Lock_X(F)	Lock_S(B)
F = B*100	
Write(F)	
Unlock(B)	
Unlock(F)	
	Read(B)
	Print(F+B)
	Unlock(B)

(B=3, F=300)

Lock_X(B)	
Read(B) 3	Lock_S(F)
B = B*10 30	Read(F) 300
Write(B)	Unlock(F)
Lock_X(F)	Lock_S(B)
F = B*100 3000	
Write(F)	
Unlock(B)	
Unlock(F)	
	Read(B)
	Print(F+B)
	Unlock(B)

(B=3, F=300)

Lock_X(B)	
Read(B) 3	Lock_S(F)
B = B*10 30	Read(F) 300
Write(B)	Unlock(F)
Lock_X(F)	Lock_S(B)
F = B*100 3000	
Write(F)	
Unlock(B)	
Unlock(F)	
	Read(B) 30
	Print(F+B)
	Unlock(B)

(B=3, F=300)

Lock_X(B)	
Read(B) 3	Lock_S(F)
B = B*10 30	Read(F) 300
Write(B)	Unlock(F)
Lock_X(F)	Lock_S(B)
F = B*100 3000	
Write(F)	
Unlock(B)	
Unlock(F)	
	Read(B) 30
	Print(F+B) **300+30**
	Unlock(B)

(B=3, F=300)

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

lock acquired after another lock has been released!

Not 2PL or Strict 2PL!

2PL

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

2PL

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

Deadlock!

Strict 2PL

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

Strict 2PL

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

Deadlock!

Strict 2PL

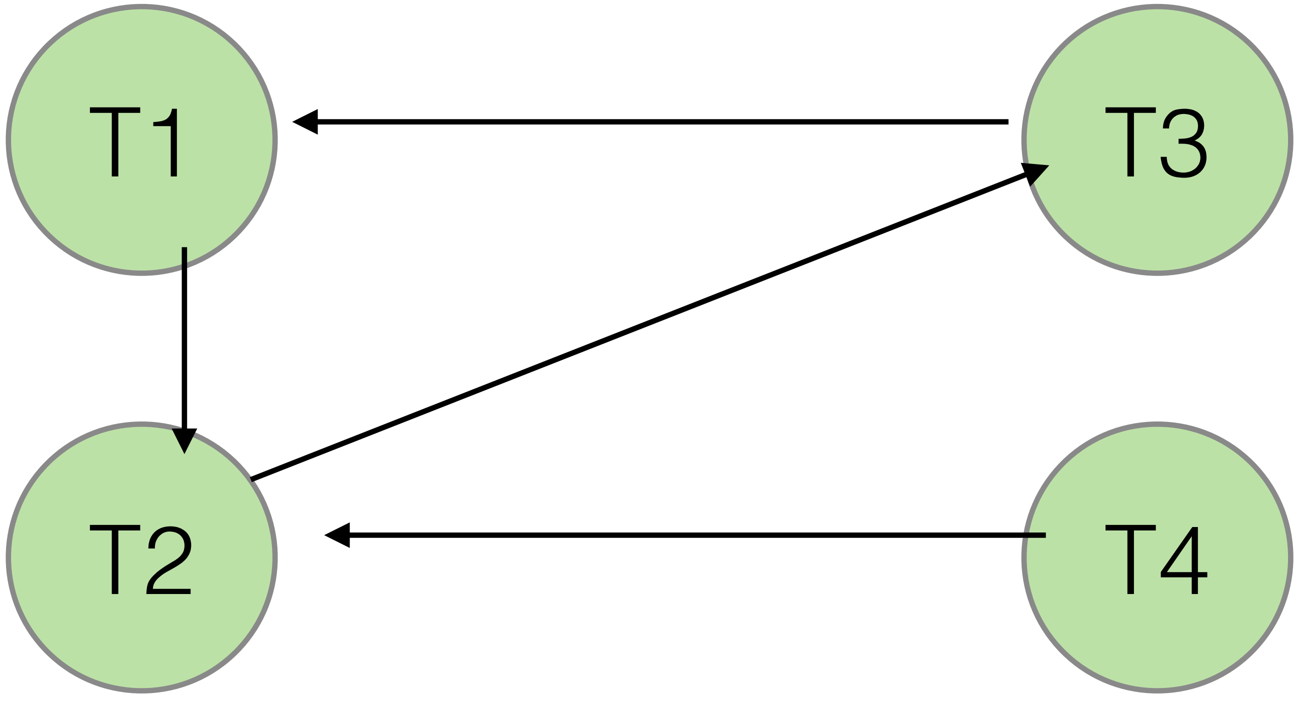
Lock_X(B)	
Read(B)	Lock_S(F)
B = B*10	Read(F)
Write(B)	Unlock(F)
Lock_X(F)	Lock_S(B)
F = B*100	
Write(F)	
Unlock(B)	
Unlock(F)	
	Read(B)
	Print(F+B)
	Unlock(B)

Strict 2PL

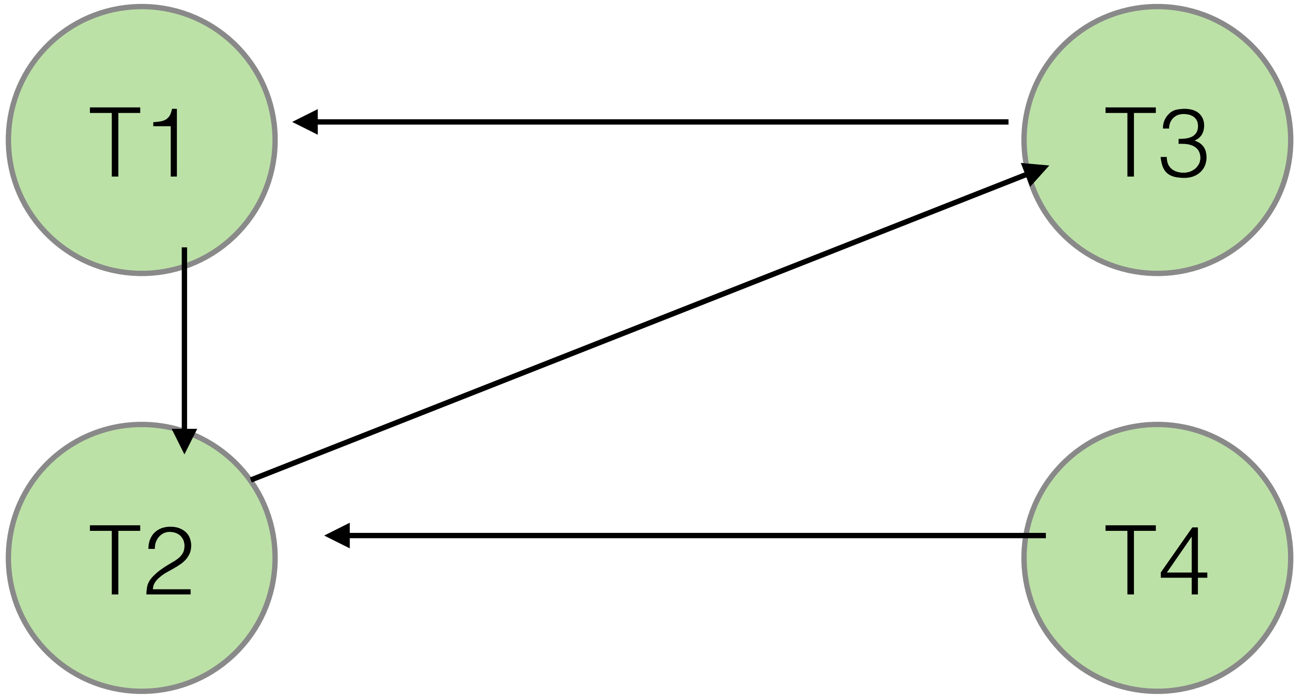
Lock_X(B)	
Read(B)	
B = B*10	
Write(B)	
Lock_X(F)	
F = B*100	Lock_S(F)
Write(F)	
Unlock(B)	
Unlock(F)	
	Read(F)
	Lock_S(B)
	Read(B)
	Print(F+B)
	Unlock(B)
	Unlock(F)

T1	S(A), S(D)		S(B)				
T2		X(B)			X(C)		
T3				S(D), S(C)			X(A)
T4						X(B)	

T1	S(A), S(D)		S(B)				
T2		X(B)			X(C)		
T3				S(D), S(C)			X(A)
T4						X(B)	



T1	S(A), S(D)		S(B)				
T2		X(B)			X(C)		
T3				S(D), S(C)			X(A)
T4						X(B)	



Deadlock possible: cycle in graph