Administrivia

HW4 Wednesday
Project 2 due next Monday
Midterm 2 next Monday in class
Format
- cumulative
- question booklet + answer sheet
- 1 page cheat sheet

Conflict Serializability

def: possible to swap non-conflicting operations to derive a serial schedule.

$\forall$ conflicting operations O1 of T1, O2 of T2
- O1 always before O2 in the schedule or
- O2 always before O1 in the schedule

	1	2	3	4
T1:	R(A)	W(A)	R(B)	W(B)
	5	6	7	8
T2:	R(A)	W(A)	R(B)	W(B)

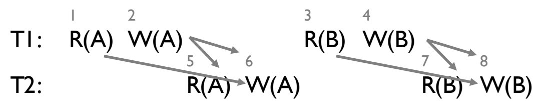
Logical

Conflicts
16, 25, 26, 38, 47, 48

Logical

	1	2	3	4
T1:	R(A)	W(A)	R(B)	W(B)
	5	6	7	8
T2:	R(A)	W(A)	R(B)	W(B)

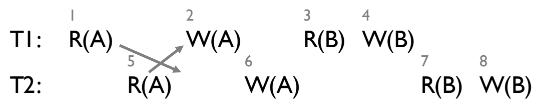
Serializable



Logical

	1	2	3	4
T1:	R(A)	W(A)	R(B)	W(B)
	5	6	7	8
T2:	R(A)	W(A)	R(B)	W(B)

Not Serializable



Conflict Serializability

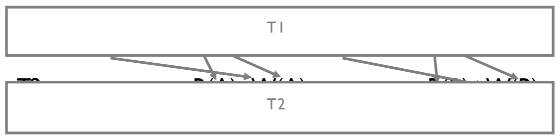
Transaction Precedence Graph
- Edge Ti → Tj if:
  1. Ti read/write A before Tj writes A or
  2. Ti writes some A before Tj reads A

If graph is acyclic (does not contain cycles) then conflict serializable!

Logical

	1	2	3	4
T1:	R(A)	W(A)	R(B)	W(B)
	5	6	7	8
T2:	R(A)	W(A)	R(B)	W(B)

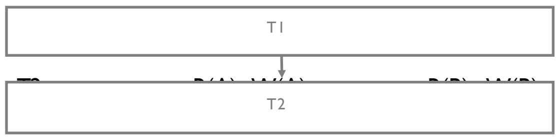
Serializable



Logical

	1	2	3	4
T1:	R(A)	W(A)	R(B)	W(B)
	5	6	7	8
T2:	R(A)	W(A)	R(B)	W(B)

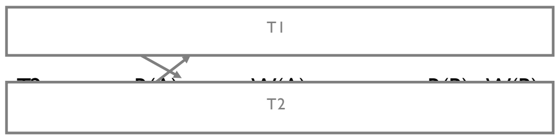
Serializable



Logical

	1	2	3	4
T1:	R(A)	W(A)	R(B)	W(B)
	5	6	7	8
T2:	R(A)	W(A)	R(B)	W(B)

Not Serializable



Fine, but what about COMMITing?

```
T1    R(A)  W(A)                     R(B) ABORT
T2                    R(A) COMMIT
```

Not recoverable

Promised T2 everything is OK. IT WAS A LIE.

```
T1    R(A) W(B)  W(A)                      ABORT
T2                          R(A) W(A)
```

Cascading Rollback.

T2 read uncommitted data → T1's abort undos T1's ops & T2's

Lock-based Concurrency Control

Must get a shared(read) or exclusive(write) lock BEFORE op

If other xact has lock, can get if lock table says so

YES

		T1	
	Allowed?	S	X
T2	S	Y	N
	X	N	N

Can this schedule happen?

```
T1    R(A)    W(A)                  R(B)  ABORT
T2                     R(A)  COMMIT
```

Lock-based Concurrency Control

Two-phase locking (2PL)

Growing phase	acquire locks
Shrinking phase:	release locks

```
                          shrink here
T1    R(A) W(B)  W(A)   ↙                ABORT
T2                          R(A) W(A)
```

Uh Oh, same problem

Lock-based Concurrency Control

Strict two-phase locking (Strict 2PL)

- Growing phase acquire locks
- Shrinking phase: release locks
- Hold onto locks until commit/abort

Why? Which problem does it prevent?

```
T1    R(A) W(B)   W(A)                     ABORT
T2                          R(A) W(A)
```

Guarantees serializable schedules! Avoids cascading rollbacks!

HW 4 due Wednesday
beginning of class, hard copy (2:30pm)

sample final solutions up

some corrections to practice problem solutions

extra OH Friday – 5pm?

Final next Wednesday in class

Review

Issues
- TR: dirty reads
- RW: unrepeatable reads
- WW: lost writes

Schedules
- Equivalence
- Serial
- Serializable

Serializability
- Conflict serializability
- how to detect

Conflict Serializable Issues
- Not recoverable
- Cascading Rollback

Strict 2 phase locking