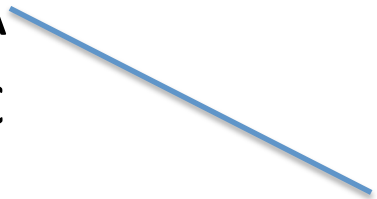
6.033 Lecture 18: Multisite Atomicity

4/14/2014

Sam Madden

Transaction Schedules

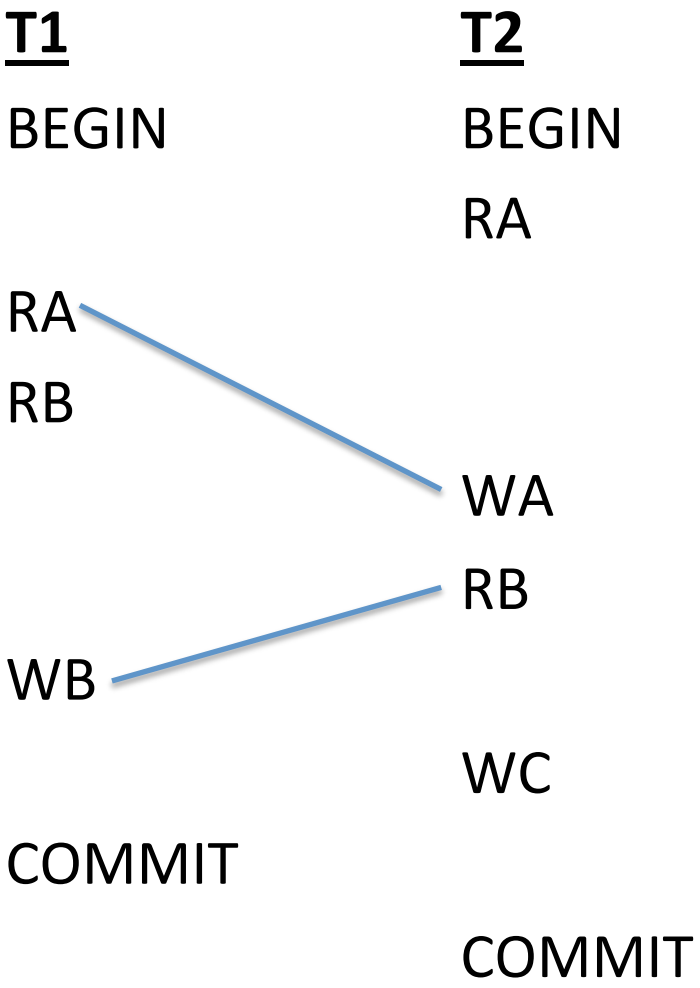
T1	**T2**
BEGIN	BEGIN
	RA
RA	
RC	
	WA
WC	
COMMIT	
	COMMIT



This schedule is **serializable**, because state of database is equivalent to running T1 then T2.

It is also **conflict serializable**, because for all conflicts between T1 and T2, conflicting operation occurs first in T1

Transaction Schedules



This schedule is not **serializable**.

T1 doesn't see T2's WA, T2 doesn't see T1's WB

Not **conflict serializable**, because T1's RA precedes T2's WA, but T2's RB precedes T1's WB

Locking Protocol w/ Release

```
Read(T, var):
    if var.lock not held by T:
        acquire(T, var.lock)
    return var.value

Write(T, var, newval)
    if var.lock not held by T:
        acquire(T, var.lock)
    var.val = newval //write log record

Commit(T):
    write commit record for T
    release all locks for T
```

Two-Phase Locking

- Phase 1: Acquire locks before accessing an object

Lock point – after all locks are acquired, transaction will never wait, can start releasing locks

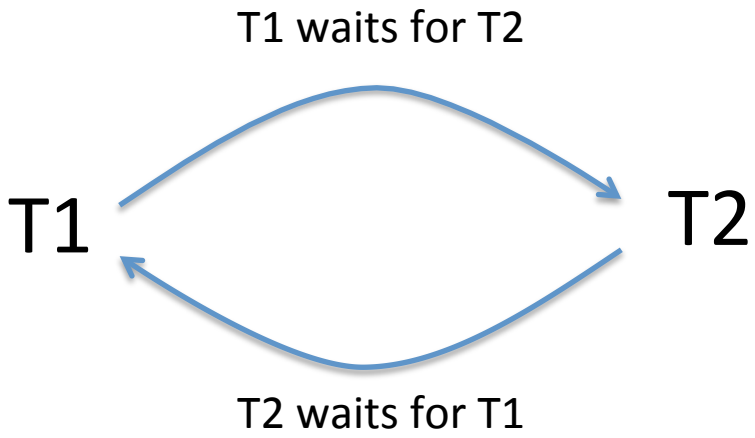
- Phase 2: Release locks on items when done with them

Strict two-phase locking holds write locks til end of transaction to prevent cascading aborts

Both variants provide serializability

Deadlock Detection

- Deadlocks can arise when transactions are waiting for each other



A cycle in the “waits-for” graph indicates deadlock

Locking w/ Reader-Writer Locks

```
Read(T, var):
    if var.lock not held by T:
        acquire_reader(T, var.lock)
            # block if any writers
    return var.value

Write(T, var, newval):
    if var.lock not held as writer by T:
        acquire_writer(T, var.lock)
            # block if any readers or writers
    var.value = newval //and write log record
```

Read committed

Table of doctors w/ names and whether on call

T1	T2
begin	
	begin
	select count(*) from doctors where oncall=true
update doctors set oncall=true where name = 'bob’	
commit	
	select count(*) from doctors where oncall=true

- W/ serializable, T1 will wait for T2
- W/ read committed, T2 will release read lock after select, which will allow T1 to run; T2 will see T1’s update (but do we care)?

2 Generals

2:102:15

attack at 2:15

ok, yes

commit!

ack

really commit!

ok, yes

In lossy network, no protocol can achieve agreement in a fixed number rounds.