Lec 18 Timestamp Ordering Concurrency Control

2 phase locking
timestamp ordering
optimistic concurrency control
] pessimistic

T/O Concurrency Control

Each txn gets unique timestamp, serialise by this order
└ can just be a counter

Basic Timestamp Ordering (T/O) Protocol

Objects tagged with read & write timestamps

X R-TS(X) W-TS(X)

Read: can't read sth written in the future
- If future txn wrote to it:
  abort the reading thread and restart with new TS
- Else:
  allow read, update R-TS, and make local copy to allow repeated reads ← MVCC stuff

Write: can't write correctly if read or written in future
- If bad write:
  abort and restart self
- Else:
  allow write, update W-TS, make local copy to allow subsequent reads

Ex. slide 9

Other rules
+ Thomas write rule — Allow blind write view serialisability
  If TS(Ti) < W-TS(X), still allow write

Observation:
- Read, write both need to write things
- Long-running txn often blocked by short, new ones
+ No deadlock protection needed

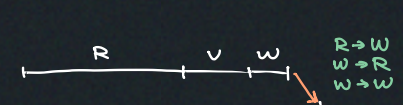
Optimistic Con. Ctrl. (OCC)

Good for when txns touch few things

Like git: make safe copy, journal actions, commit back
└ "validation"

1. Read Phase — all reads & local writes, work in local workspace
   track all reads & writes
2. Validation — any conflict with other txns? (restart if failed) ← declare timestamp at this point
3. Write Phase — apply local changes to DB

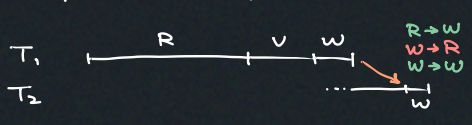
Cases

1 

Ok

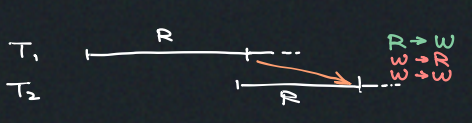
these set checks only cheap if OLTP

2 T1 finishes W before T2 starts W



Only ok if
WriteSet(T1) ∩ ReadSet(T2) = ∅

3 T1 finishes R before T2 finishes R



Only ok if
WriteSet(T1) ∩ ReadSet(T2) = ∅
WriteSet(T1) ∩ WriteSet(T2) = ∅

But which other txns to check against? (validation scope)

- Backward — Check against those that committed
- Forward — Check against those that could commit in future

Write phase:

- Serial commit — one txn write at a time
- Parallel commit — use latches to write but ensure ordering

Performance
+ Fast when few things to write
+ No dealing with deadlock
- Have to abort if validation fails
  └ so bad for long txns
+ Abort very simple — just not commit

Phantom Problem slide 34

Record level locking, but some other thread inserts

T1	T2
begin	begin
count → 99	
	insert
count → 100 ∴	← phantom inserted
	commit
commit	

Solutions:
- Redo scan to see if we see same record ids
  - Expensive
  + Ok if not scanning many things
- Predicate locking
  └ identify how things different txns look at intersect based on predicates
- Index locking — if predicate on index
  - Key-val locks
    └ Use virtual lock for non-existent key
  - Gap locks
    └ lock range of non-exist key btwn two existing keys
  - Key-range locks
    └ lock entire range regardless
  - Hierarchical locking
    └ different lock type for different levels, if range not known yet