Lec 17 Two Phase Locking Concurrency Control

We need to ensure correctness so schedule is coming in

Lock Management

Lock manager — like a hash table but for DB objects and their locks

Ex. see slide 4

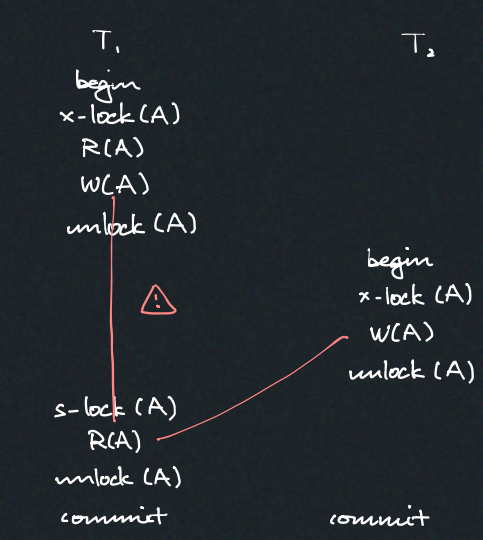
Execution engine decides which part calls the lock manager

Lock Types

- Protects DB content throughout transaction
- Has many modes (shared, exclusive, update, intention, ...)
- Detect & resolve deadlocks

Basic modes : shared, exclusive

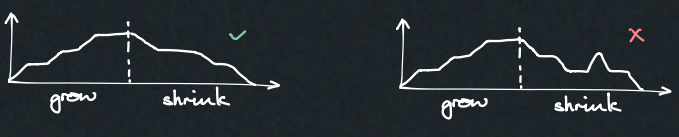
Ex.



Two Phase (2PL)

- Growing phase — acquire / upgrade locks
- Shrinking phase — only release lock

Result : not conflict serialisable if above violated

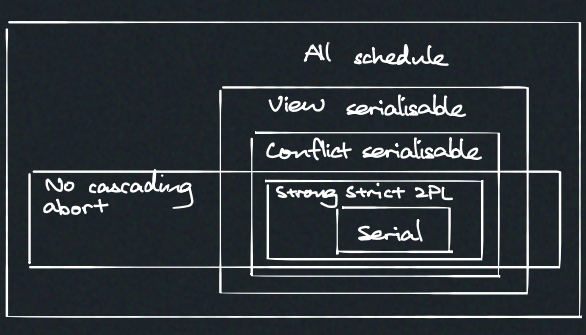


A bit more problem : ABORT Ex. slide 16

↑ they can cascade to many threads

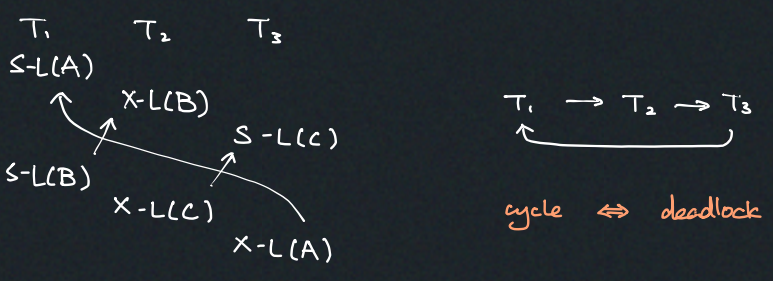
▷ Strong Strict 2PL
└ only unlock when transaction is done

+ Conflict serialisable
+ No cascading abort



2PL deadlocks

▷ Detection — make wait-for graph



Cycle detection is hard, but len-2 cycle detection is not
↳ DBs try len-2 detection first

Run detection periodically or when something seems stuck

* Victim Selection

→ By age
→ By progress
→ By # things locked
→ By # txns that will be killed as result

Regardless which, try prevent starvation

* Rollback Length

→ Complete rollback
→ Unroll to saveponct (specified by application code)

▷ Prevention — priority based on timestamp
aggressively kill things

→ Wait die — older transaction waits, kill newer one
→ Wound wait — newer transaction waits, kill older one

Observations

Locks are expensive — consider locking granularity
table / col / tuple / ... level

→ Allowing locking at any level in the DB hierarchy

DB — Table — Page — Tuple — Attribute
└ common

Intentional Lock

Stating intention to do something
└ intend to do something to somewhere down below

- Intention Shared (IS)
- Intention Exclusive (IX)
- Shared + IX (SIX)

Compatibility

	IS	IX	S	SIX	X
IS	✓	✓	✓	✓	×
IX	✓	✓	×	×	×
S	✓	×	✓	×	×
SIX	✓	×	×	×	×
X	×	×	×	×	×

Lock escalation

When someone needs to change their lock type