Lec 9 Concurrency Control

Want: multiple threads using same data structs safely

Concurrency Control

Protocol DBMS use to ensure correctness

- Logical correctness : thread being able to see data it's supposed to
- Physical correctness : internal repr is sound

Usually use latches

Latches

Recall

Lock	vs	Latches
transaction level		thread / worker level
allows rollback		no rollback
protect DB conten		protect data struct
long term		shorter term updates
for entire transaction		during critical section
detect & resolve		avoid deadlocks
kept by lock manager		kept in data struct

Latch modes
- Read (R) - Multiple threads can have it
  - Can acquire if another thread also reading
- Write (W) - One thread only
  - Cannot acquire if any other thread in read / write

	R	W
R	✓	×
W	×	×

Impl goals
- Fast
- If thread waiting for too long, deschedule it
  - └ Latch may need own queue to track waits

Impls
→ Test-and-Set Spinlock
  - Efficient (single instruction latch / unlatch)
  - Doesn't scale, not cache nor OS friendly

non-uniform memory
↓

```
std::atomic<bool> latch; ← can be in another CPU's DRAM :(
while ( latch.test_and_set (..)) { ... }
                                  ↑
                              busy loop :(
```

→ OS Mutex (bad idea)
  - Not scalable

```
std::mutex m;
m.lock();
...
m.unlock();
```

  - Problem: usually in userspace, but if one thread in write and the other not, we get into expensive OS space
    - └ that's why DBs tries to do things in user space

→ RWLocks

```
std::shared_mutex
```

  - Need mechanism to prevent starvation

→ Adaptive Spinlock
→ Queue-based Spinlock

Hardware level

Compare & swap — atomic instruction at asm level

```
               compare with this
                    ↓
compare_and_swap (&M, 20, 30)
                   ↑       ↑
               address    if equal set to this
```

Hash Table Latching

Note for resize just use global latch (easy way)
We worry about other access

→ Page / Block level latches
  - Each block has RW latch

→ Slot latches
  - Can use single-mode latch per slot

} tradeoff

B-Tree Latching

Latch some subtree instead of root!

→ Latch coupling / crabbing
  Allow : - Latch parent
  - Latch child
  - Unlatch parent if safe

  Safe mode : will not split or merge
  viz. not full for insert
  not underflow for delete

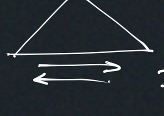
Find : get R latch on child
unlatch parent
keep going down

Ins/del : get W latch if needed
chech child safety
if safe unlatch ancestors

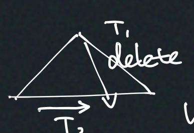
Bottleneck : root latched often
but root write very rare

→ Better latching alg : optimistic tree descend with R latch
if tail retry with W latch

Edge case, horizontal leaf traversal



If both read, okay



Uh oh. T_2 can kill itself and restart
(Else don't know how long to wait)