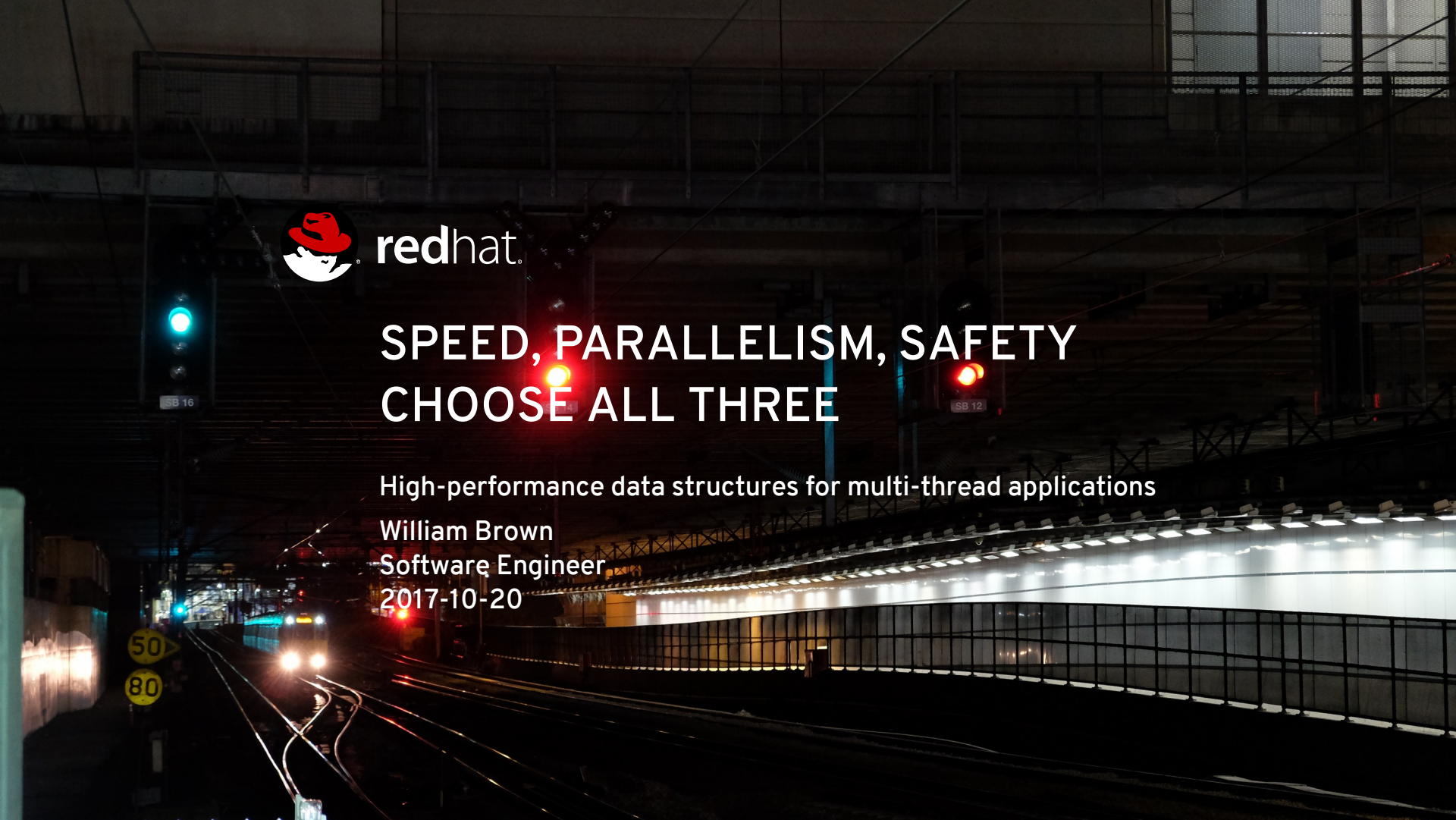
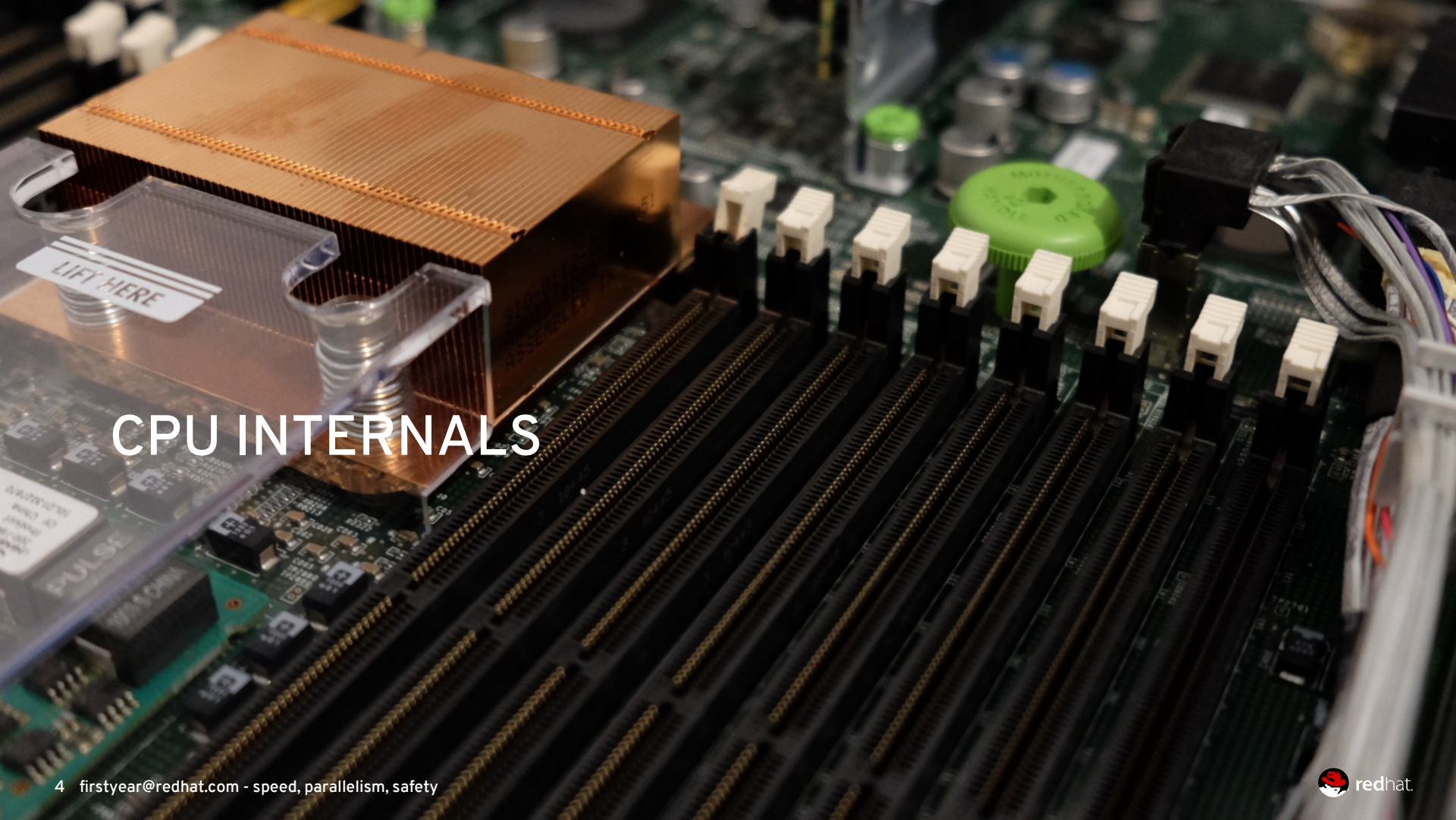

redhat.
SPEED, PARALLELISM, SAFETY
CHOOSE ALL THREE
High-performance data structures for multi-thread applications
William Brown
Software Engineer
2017-10-20
SB 16
SB 12
50
80

ldapwhoami -D uid=william,o=389ds

- Red Hat Asia Pacific (Australia)
- 389 Directory Server team
- Save questions for the end
  - Unless I am speaking too fast

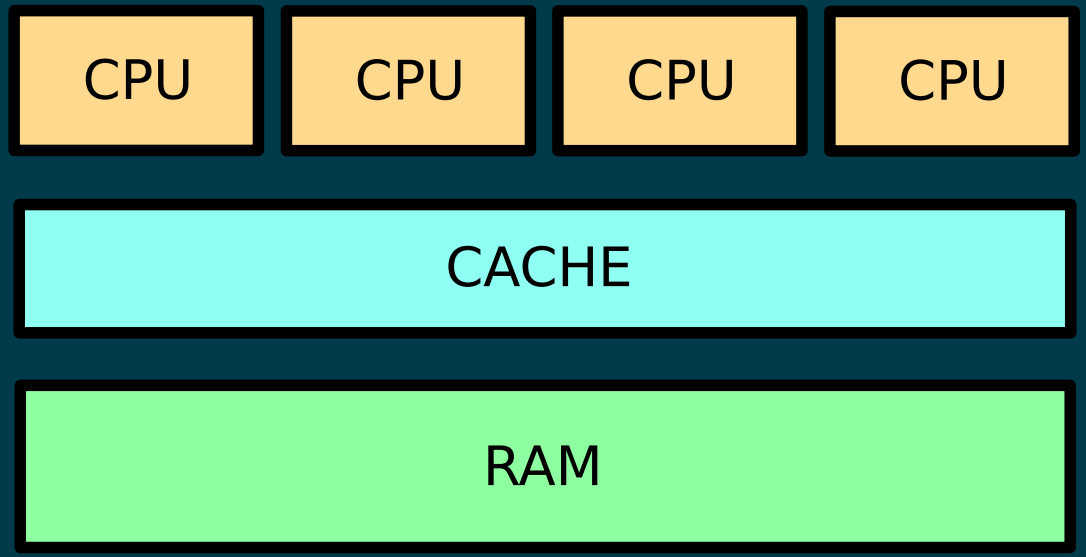
PROJECT STATUS



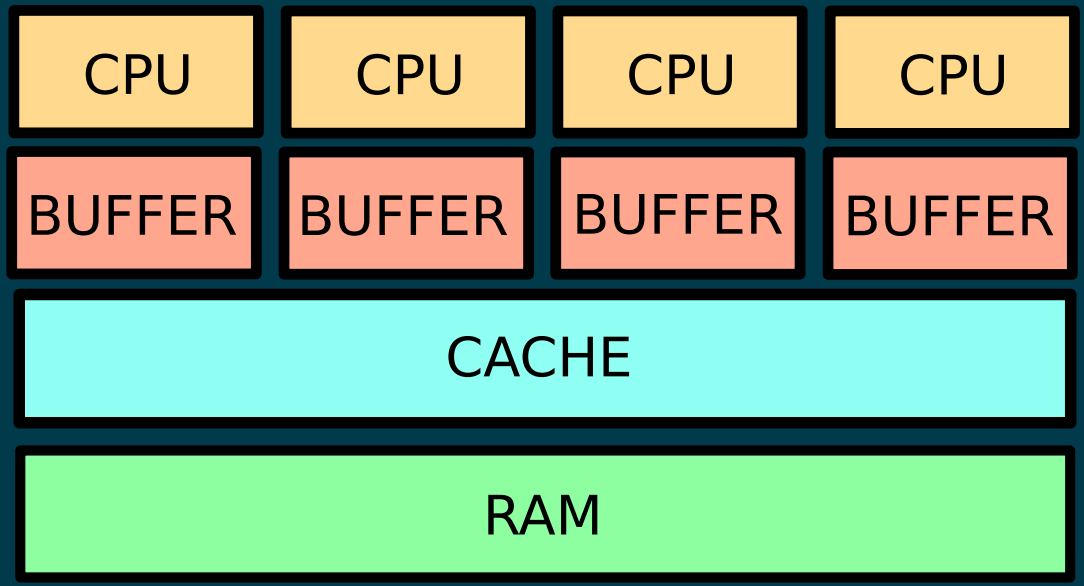
CPU INTERNALS

How we are told it works ...

Very simple model of a cpu.



How it really works.



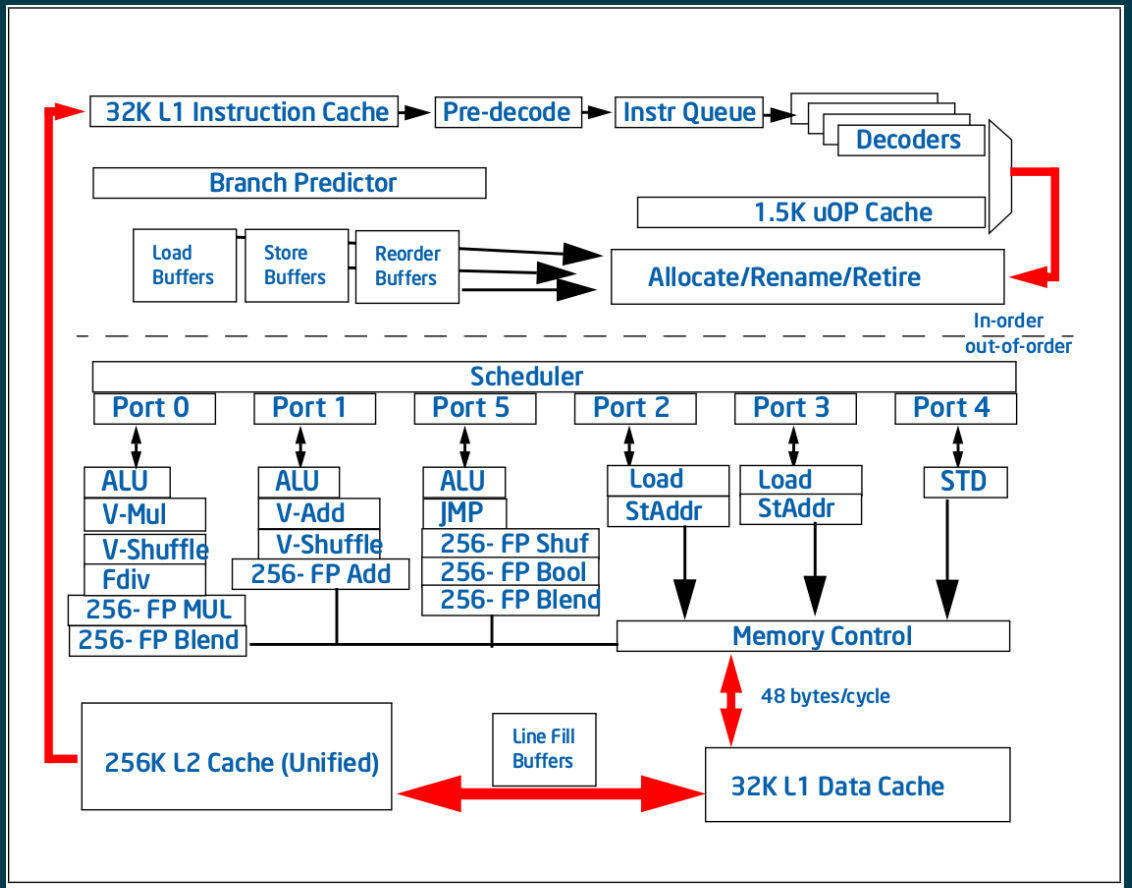


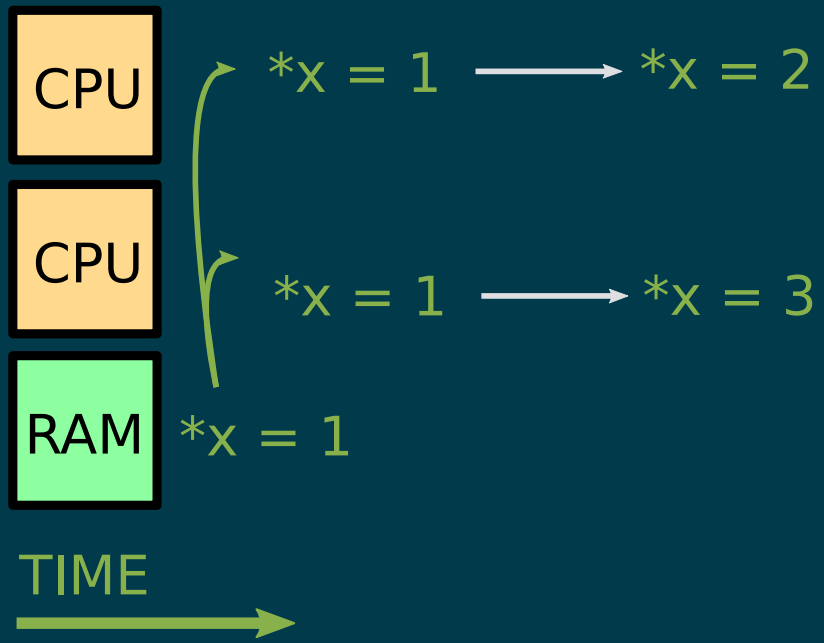
Figure 2-1. Intel microarchitecture code name Sandy Bridge Pipeline Functionality

What does this mean?

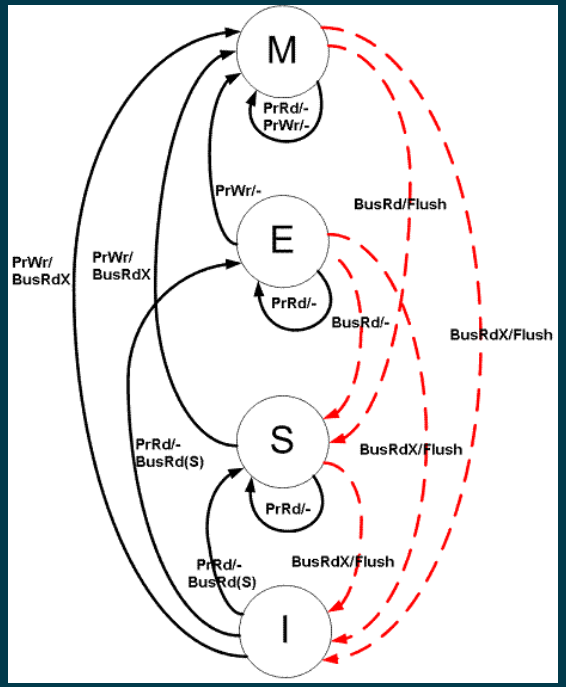
"Any writes in a cpu may never be seen by any other cpu, and writes performed by any other cpu may not be visible to this cpu"

HP bl460c

Threads over time



Cache coherency



- Modified
- Exclusive
- Shared
- Invalid

We can assume once a value is in cache, it is consistent.

Buffers and Invalidation Queues

Store barrier:

All stores before this point must be complete before a store after this point

Load barrier:

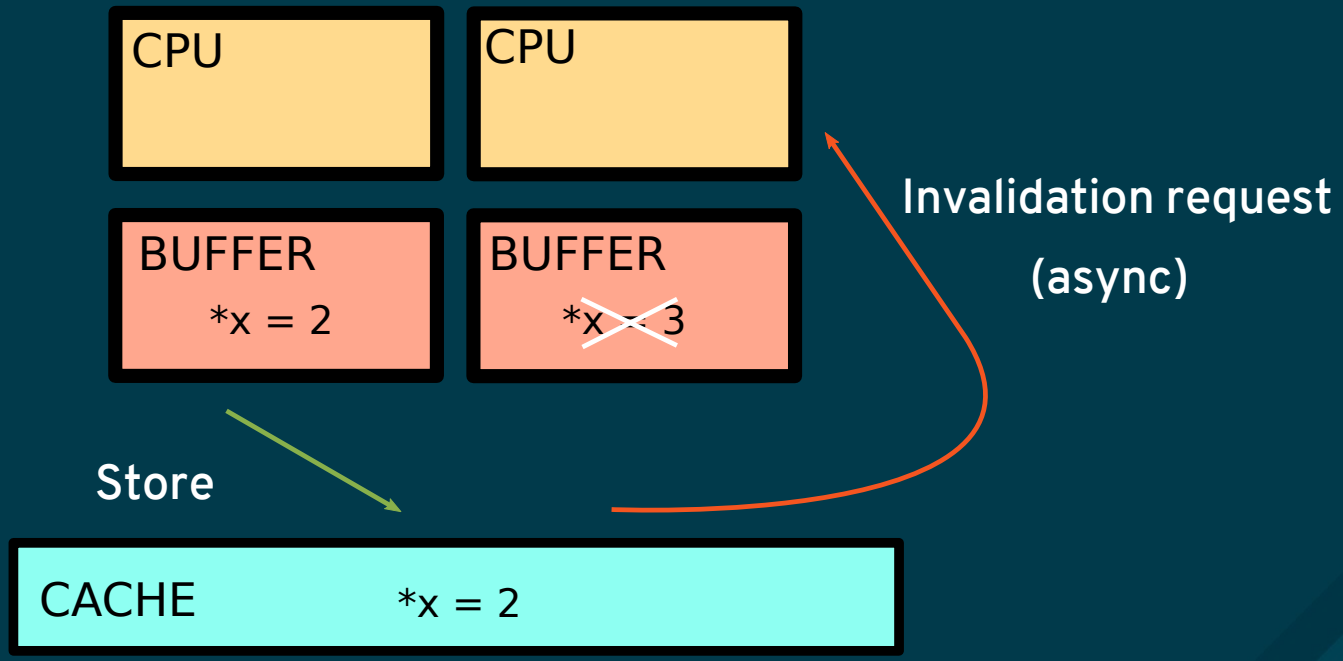
All invalidation requests must be satisfied before the next load

Store barriers

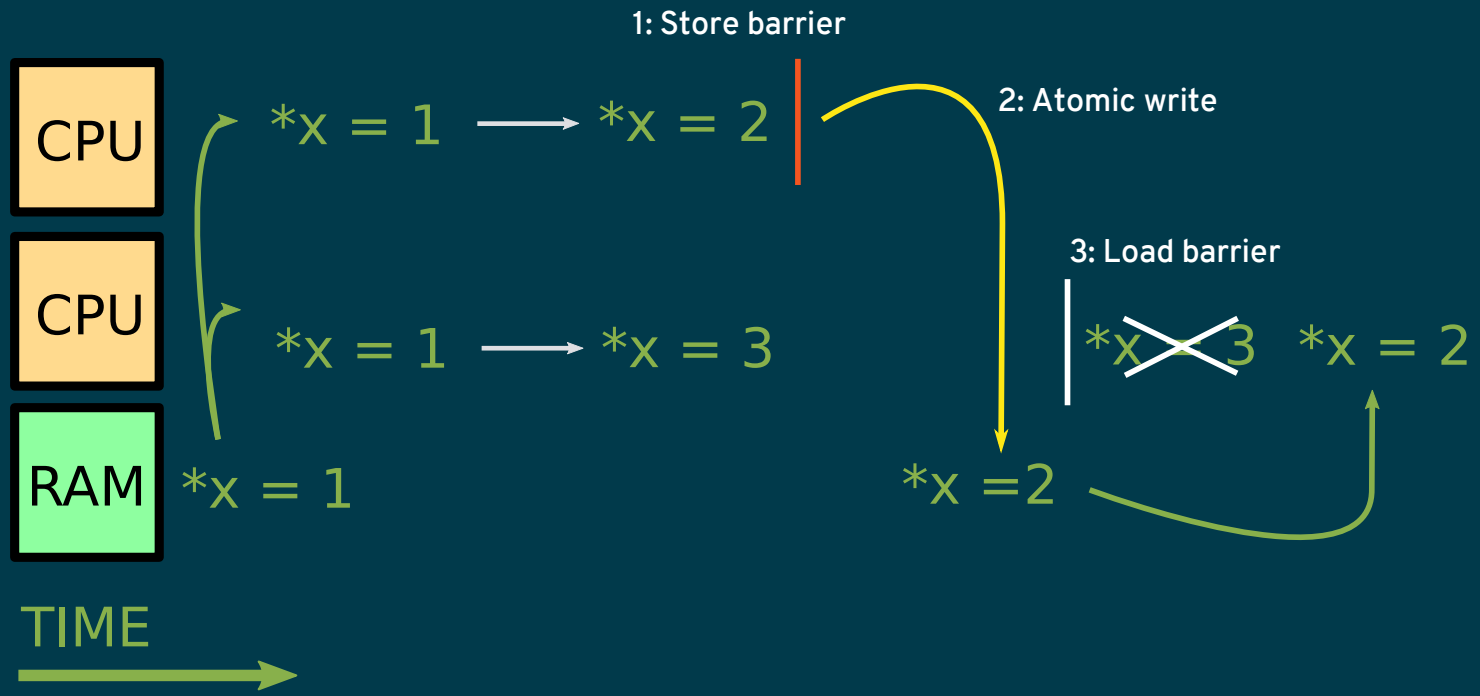
The value of X must be stored to cache before the value of Y

```
*x = 2
// store barrier
*y = 3
```

Invalidation requests

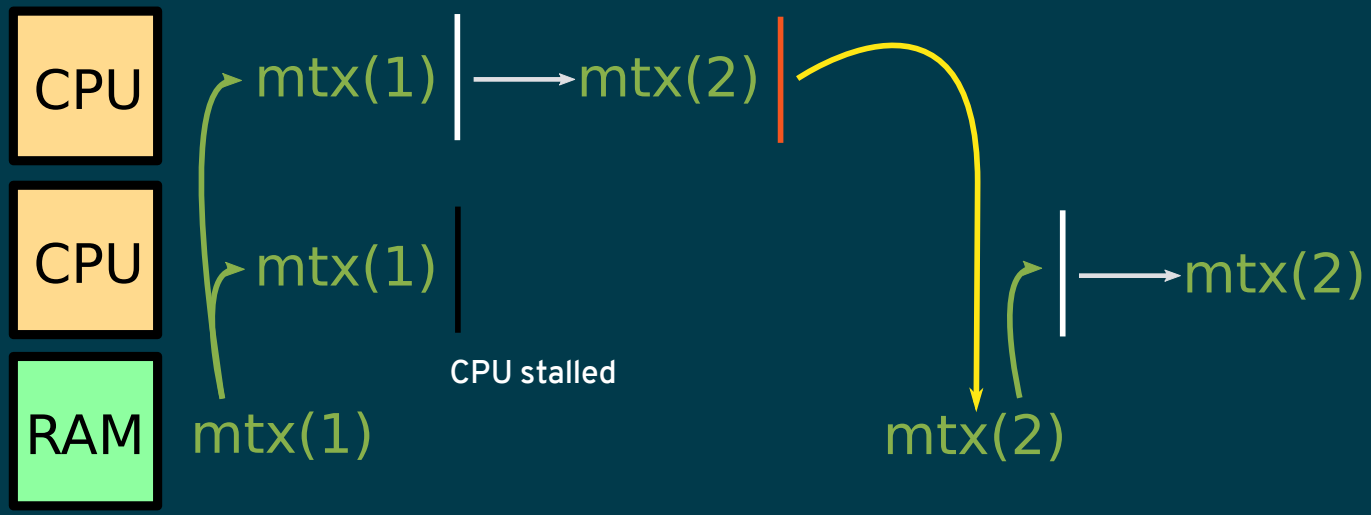


Threads with barriers

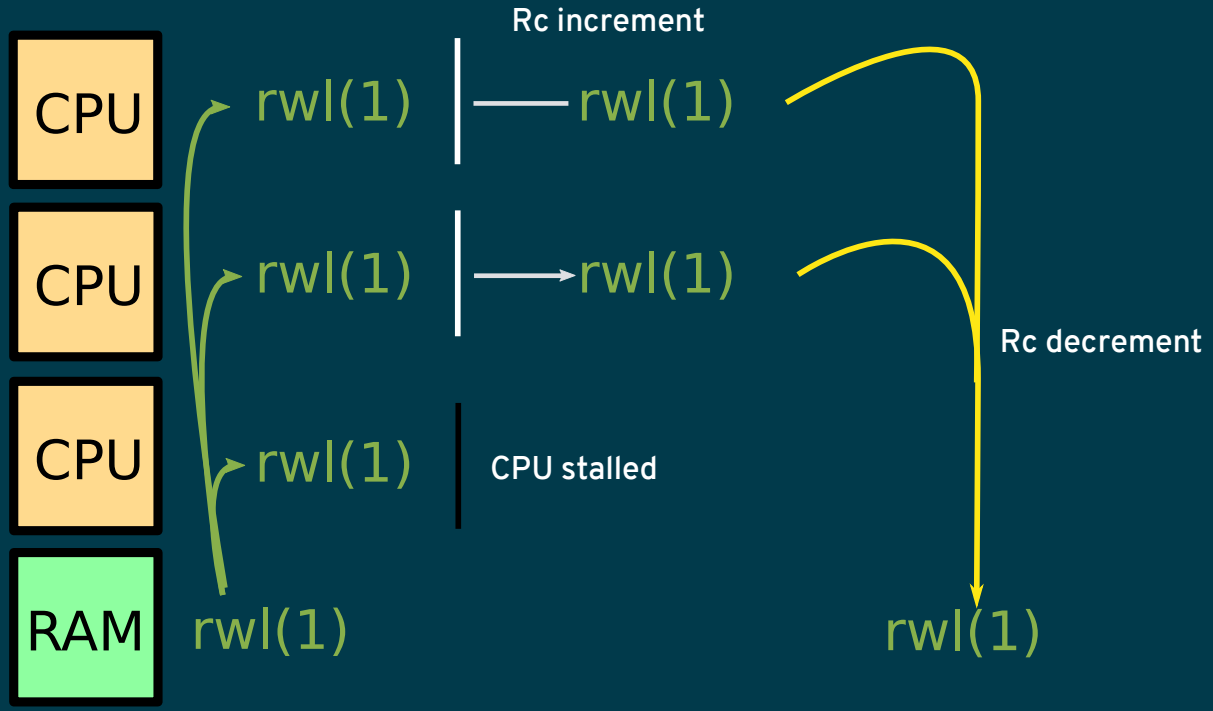


MUTEXES

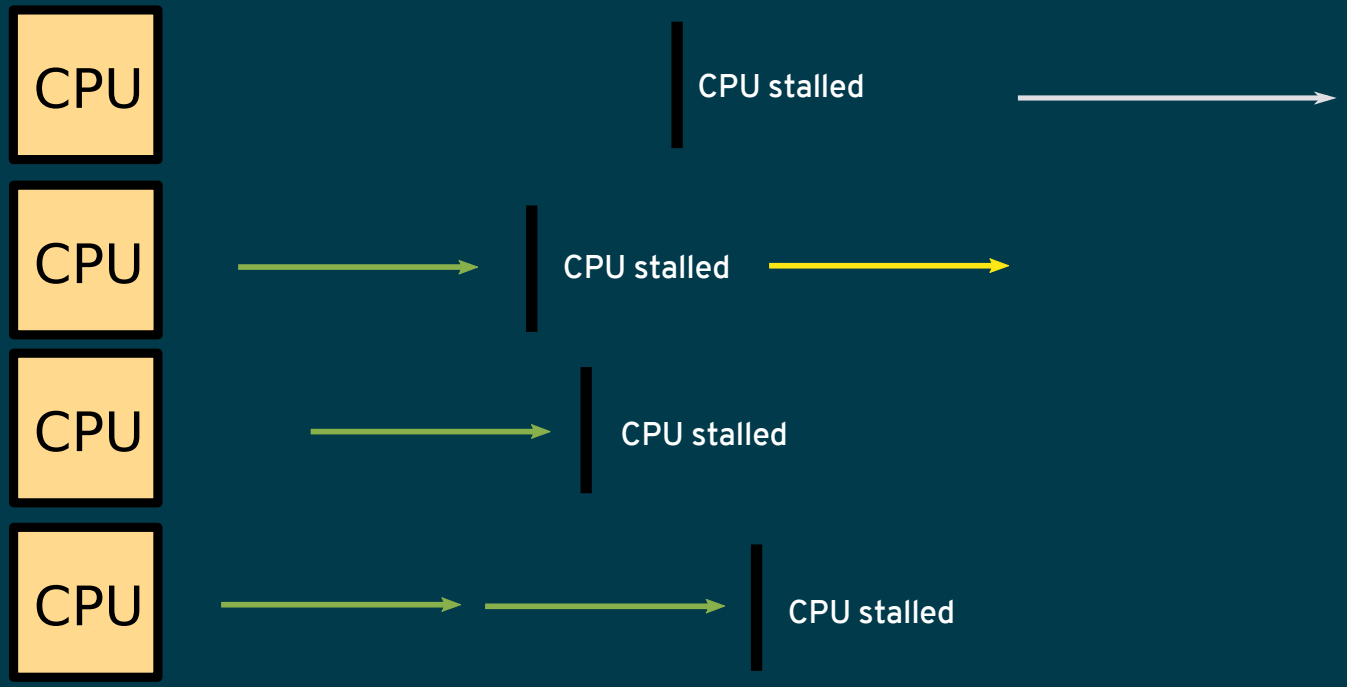
Threads with mutexes



Threads with rwlock



Rwlock behaviour



Summary of CPU internals / mutexes

- Writes are expensive for invalidation of cache lines
- CPU's may not be consistent
- Programs should match CPU behaviour
  - Single thread writes to locations
  - Parallel reads are faster (shared)
- Mutexes make writes safe
  - But they penalise mixed writes / read
- RWLock parallelises read
  - But penalised on writes
- Both behaviours don't match CPU expectations

DATA STRUCTURES

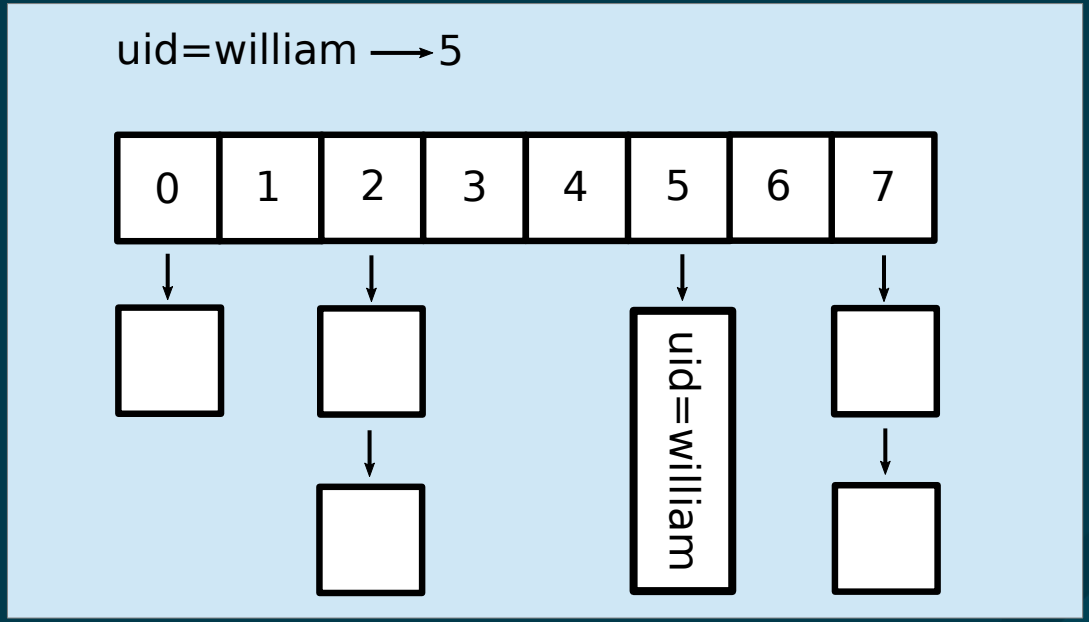
Hash maps

Pros:

- Simple structure
- Fast lookup (theoretical O(1))

Cons:

- Requires read-write lock for access
- Unsorted
- Relies on good hash function



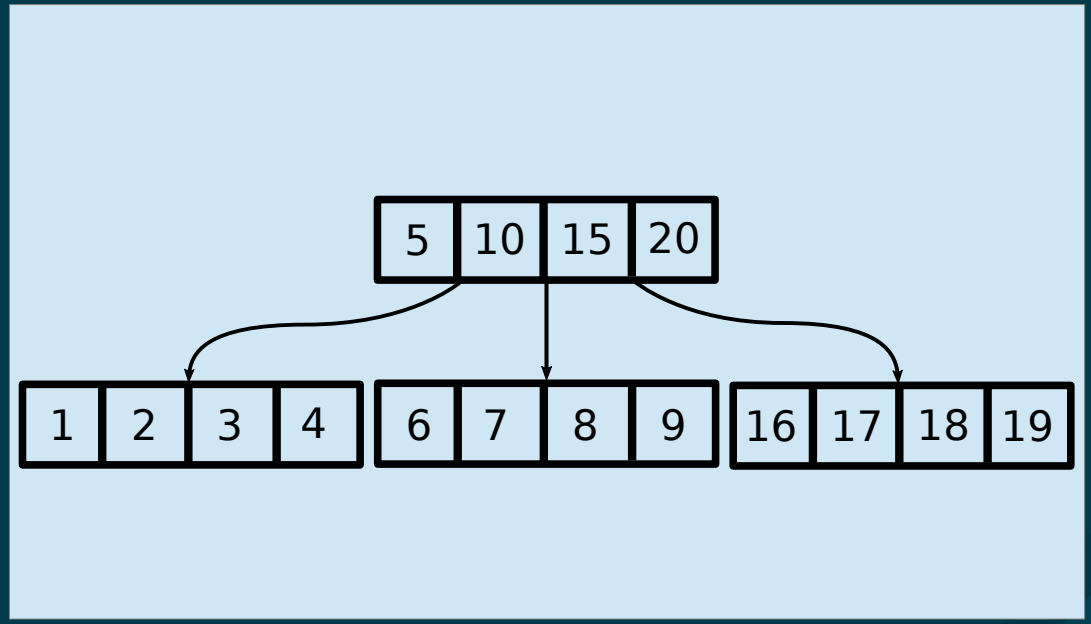
B-Trees

Pros:

- Sorted
- Cache friendly

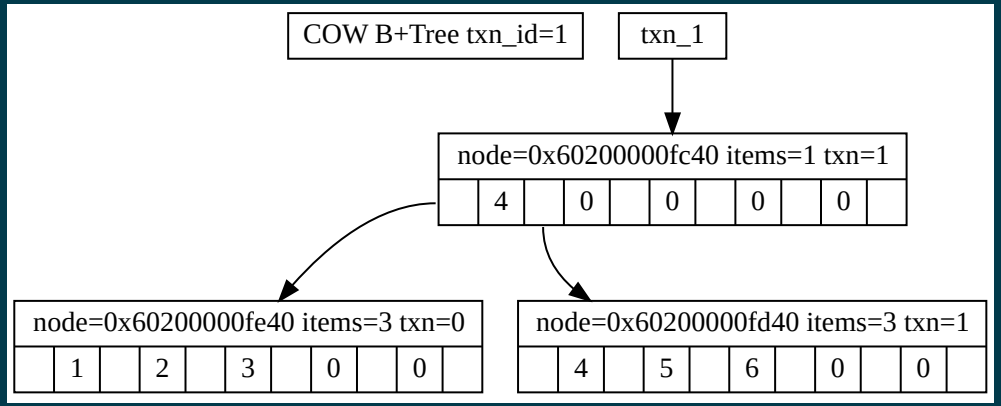
Cons:

- Requires read-write lock for access
- Slower than hmap (O(log n))
- Key comparisons can be expensive on char *

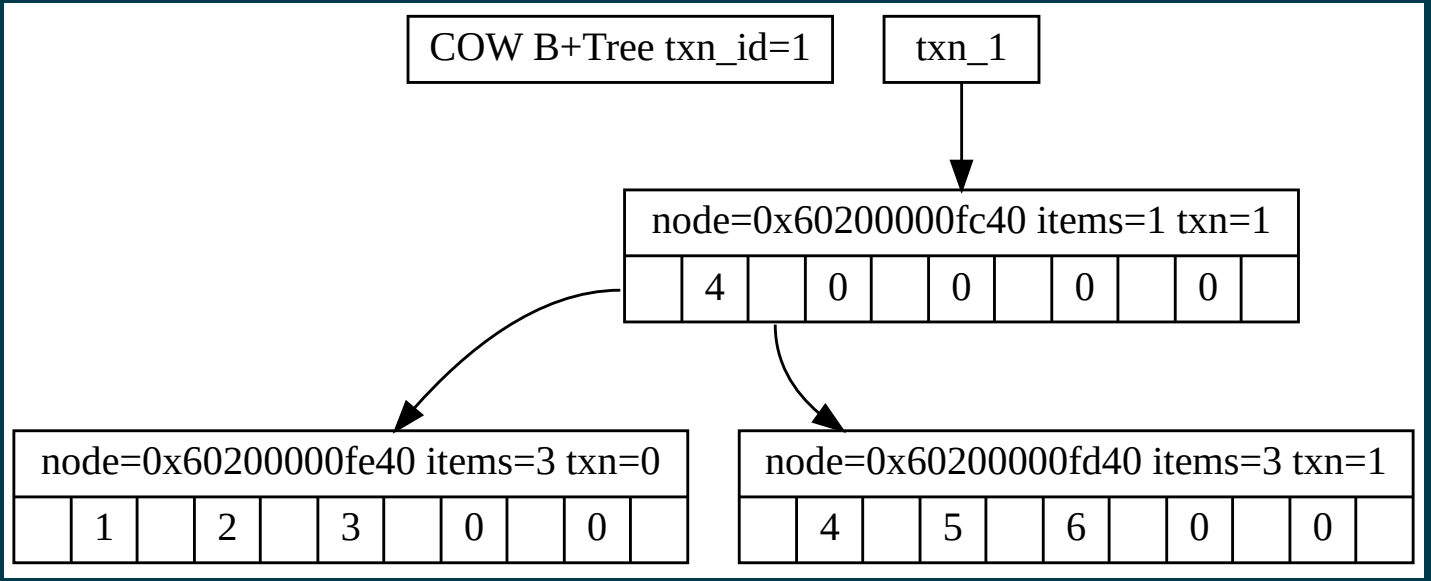


Copy on write B-Trees

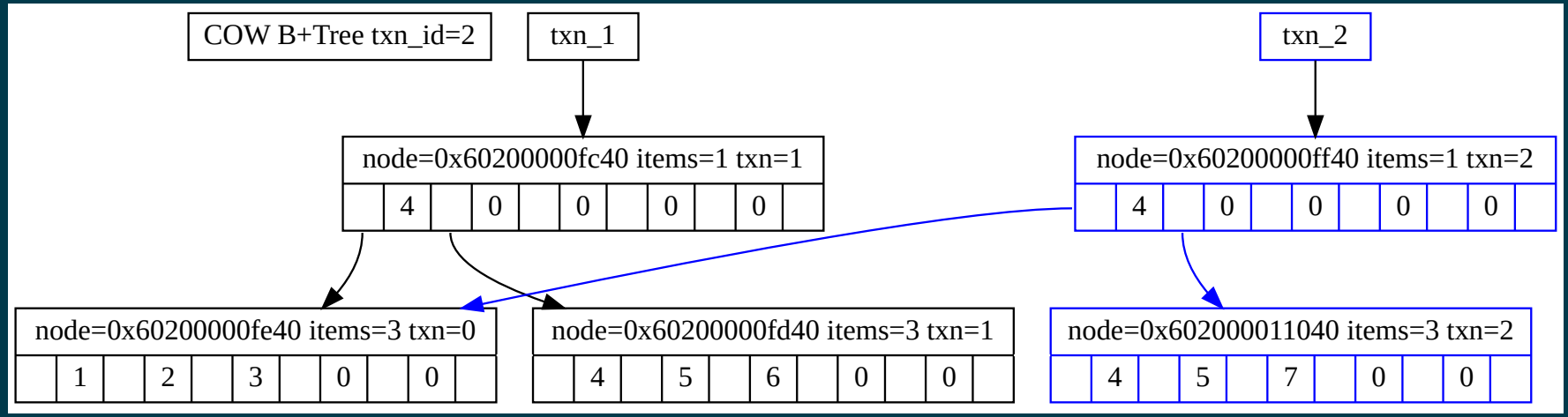
- Copies before write
- Single writer
- Multiple readers
- Built with mutex + read-write lock + atomics



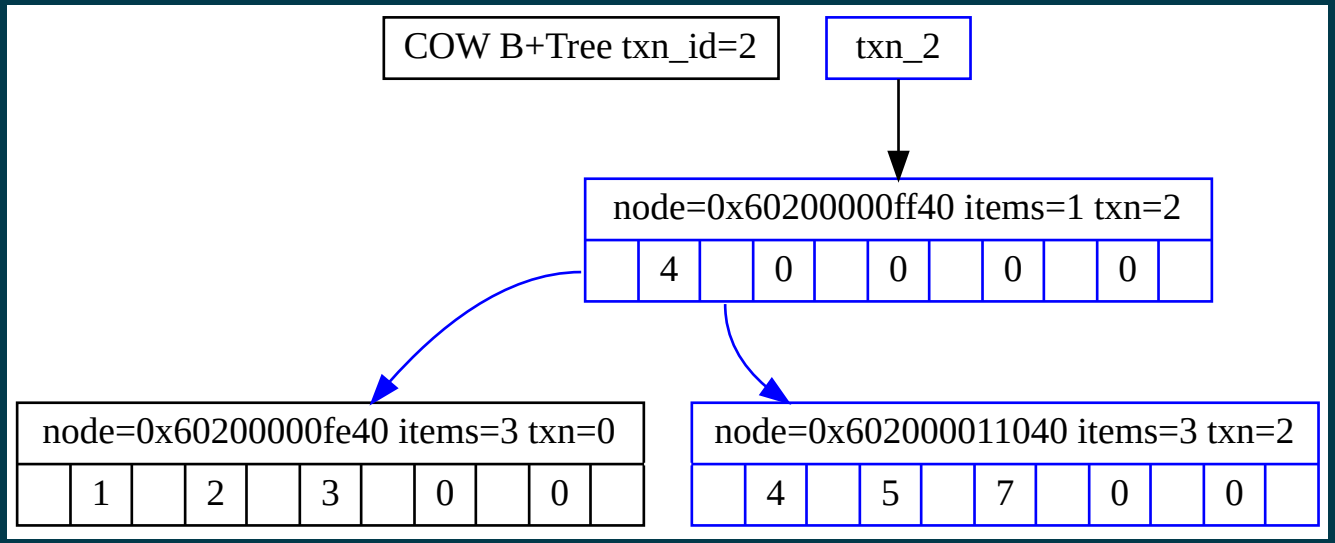
Copy on write B-Trees



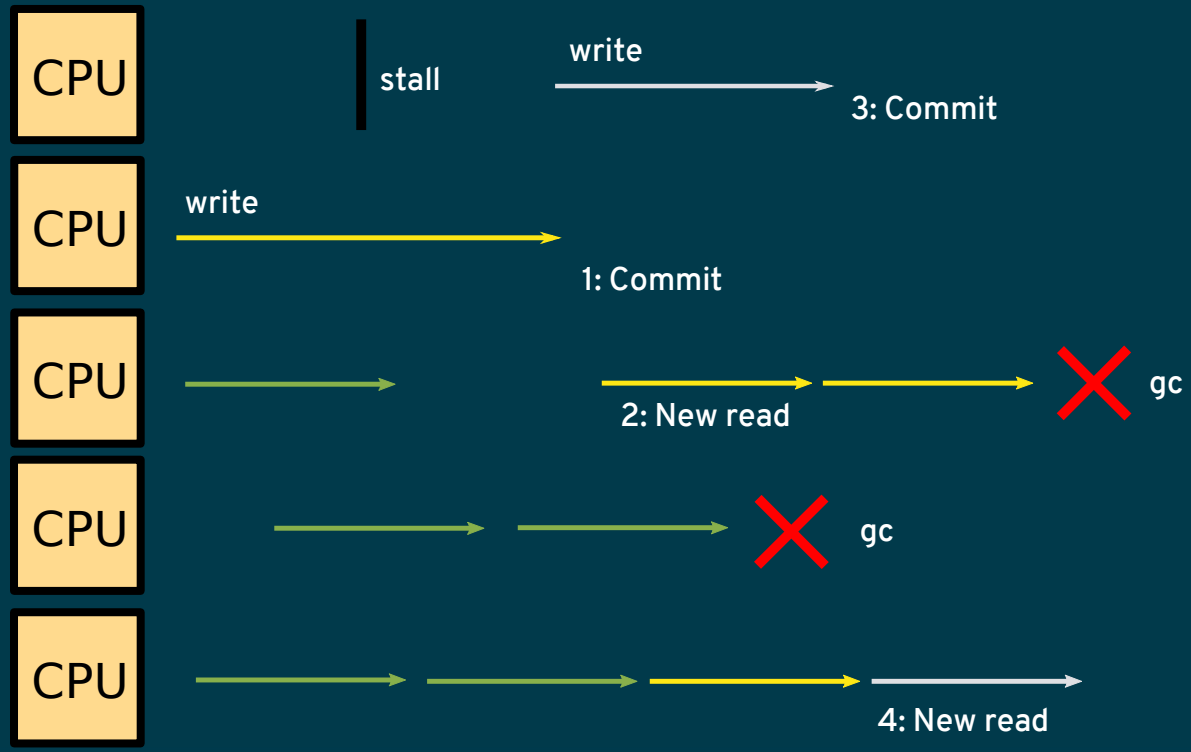
Copy on write B-Trees

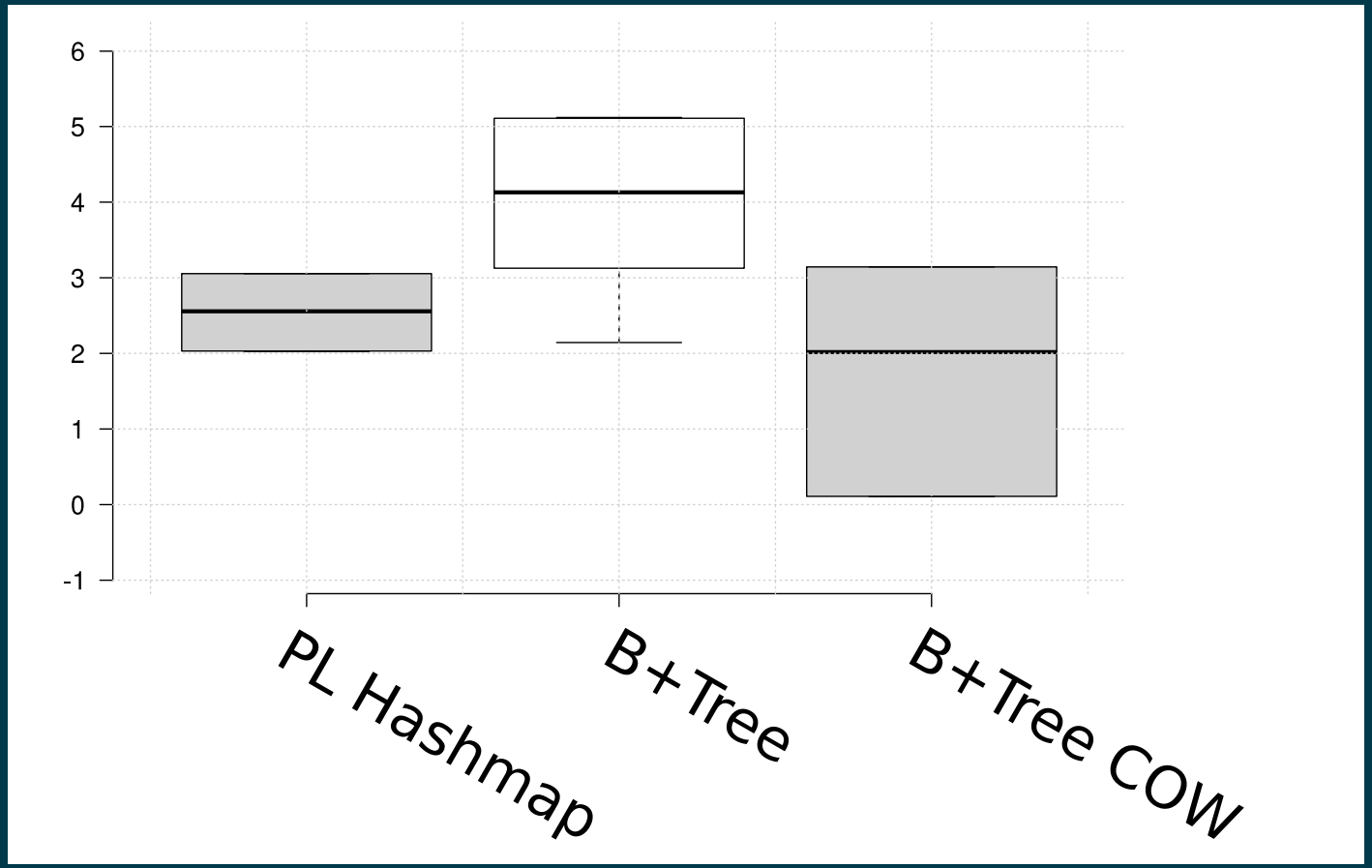


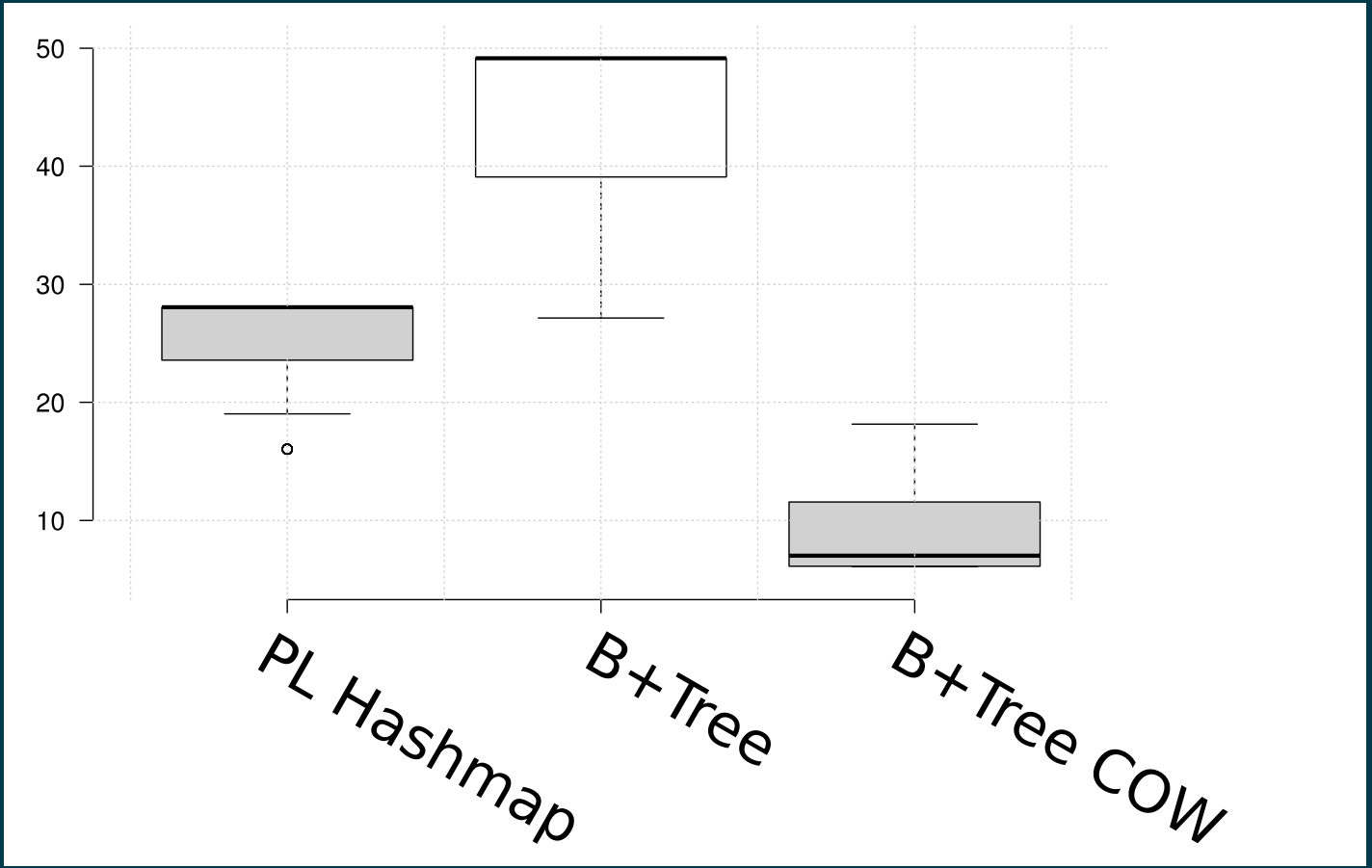
Copy on write B-Trees

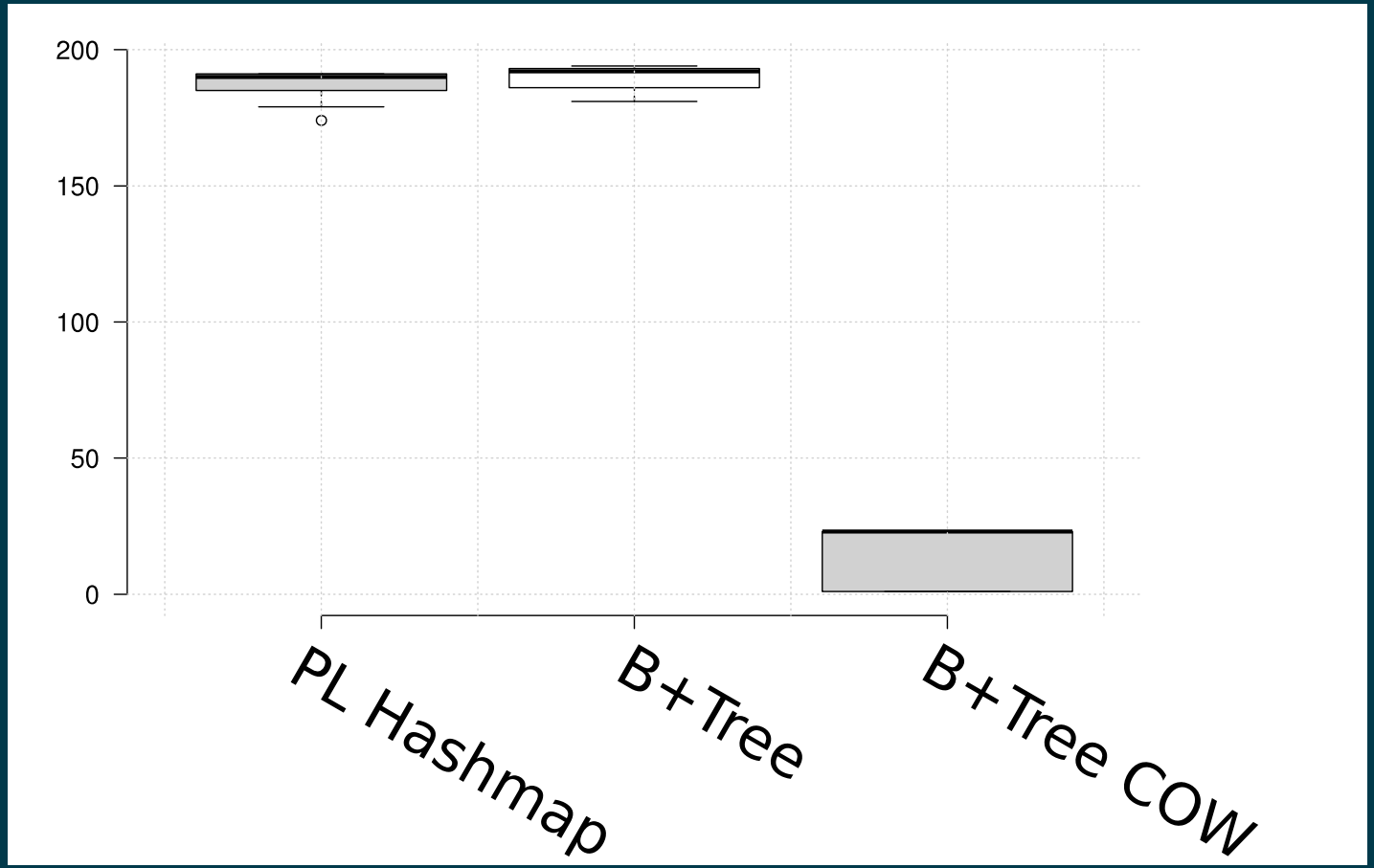


Transactions over time



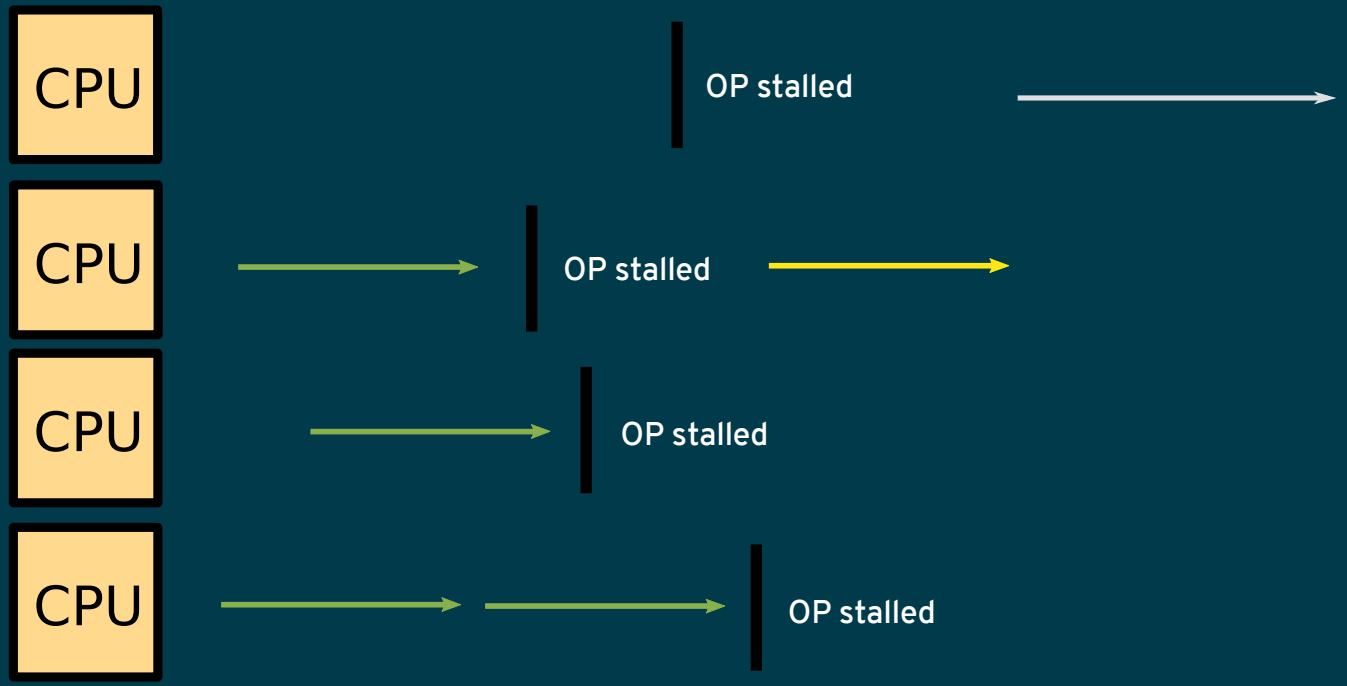






APPLICATION ARCHITECTURE

Current operation design



Transactional operation design

COW and Thread Safety

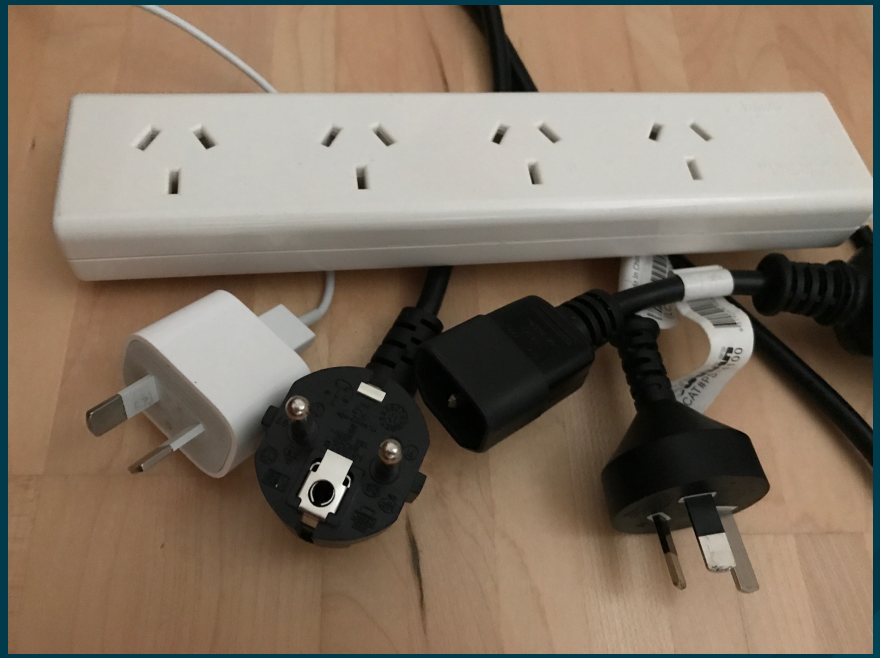
- Every operation starts a new read transaction
- Writes can have limited scopes
- Guarantees resources until operation complete

Dynamic plugins

- Dlopen handles in cow tree
- Plugin handles in cow tree (with context data)
- Guarantees plugin config and library until operation concludes

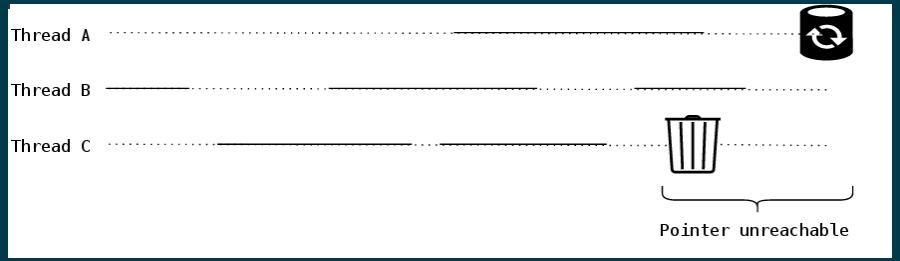
Connection handling

- New connections (accept) are in write transaction
- All operations take read txn to tree for access to conn data
- Closed connections are pruned once all former reads complete



Hazard pointers/Epoch

- Same effect as atomic RC
- Potentially faster
- Nicer semantics for a programmer



- https://ticki.github.io/blog/fearless-concurrency-with-hazard-pointers/

Rust

- Strict language
- Correct behaviour is often performant behaviour
- Has concurrency libraries (epoch)

Copy on Write Structs

- Server configuration
- Plugin contexts
- Much more ….

CONCLUSION

THANK YOU

firstyear@redhat.com