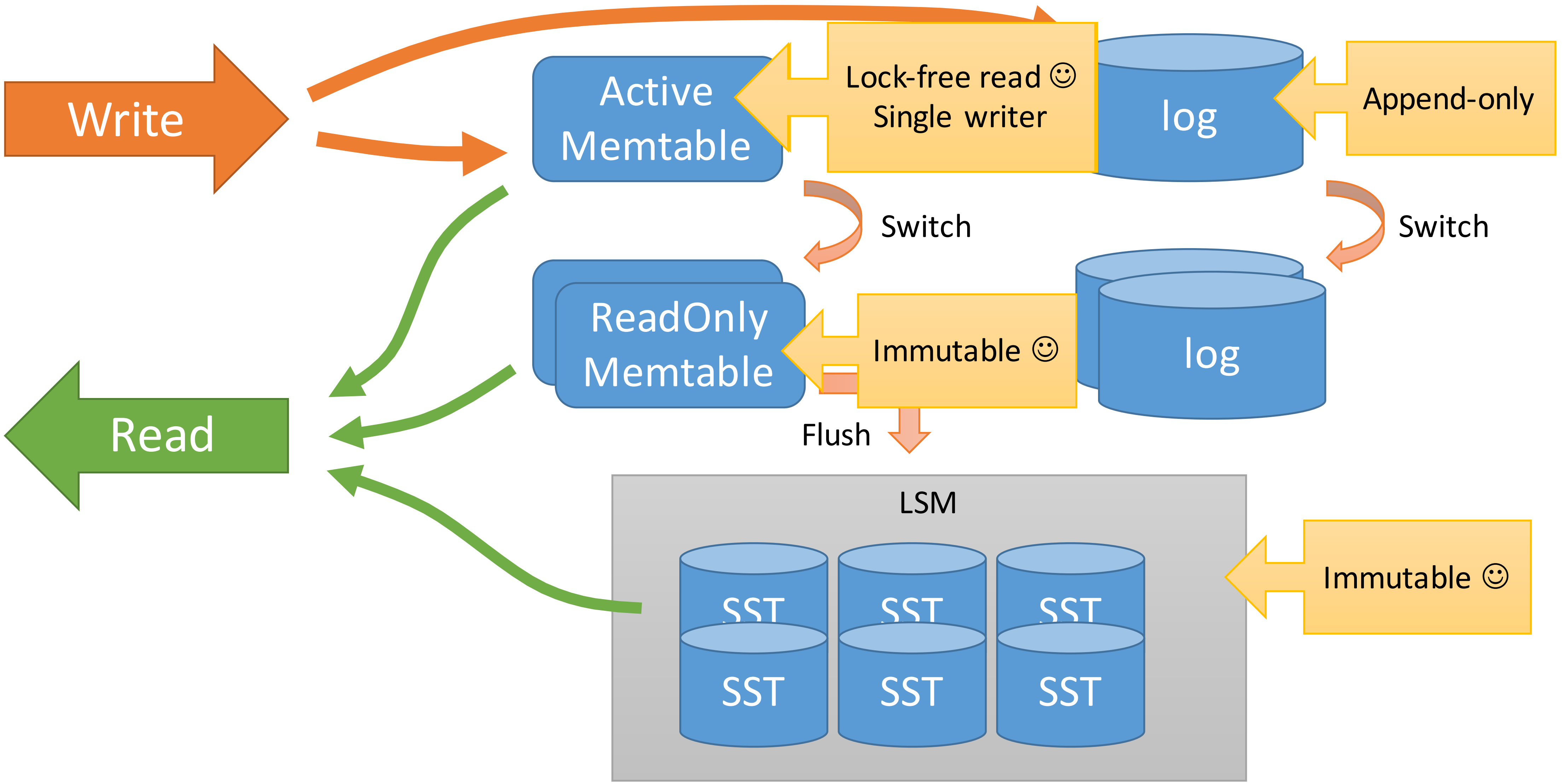
Improving RocksDB's Write Scalability

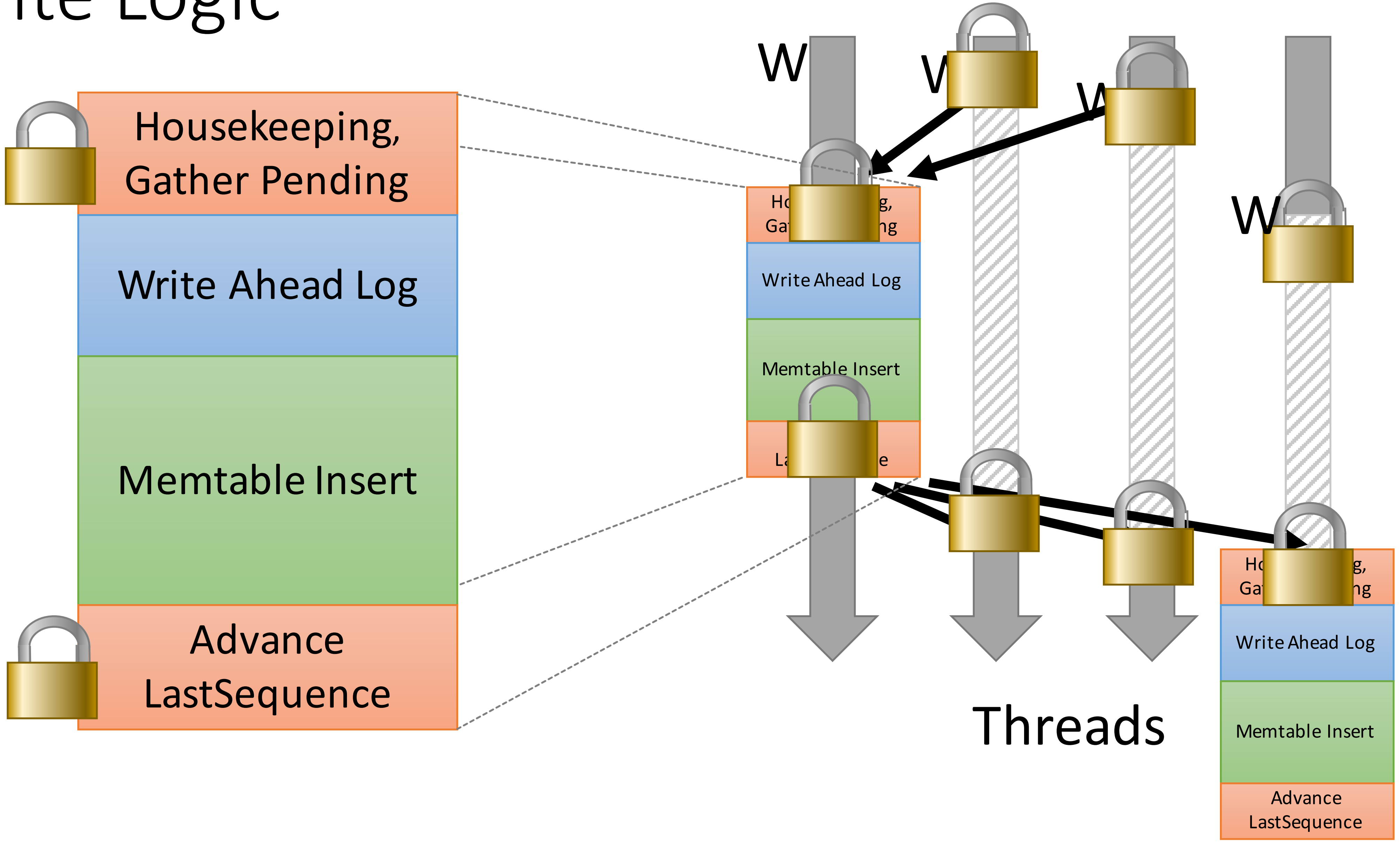
Nathan Bronson – Facebook

15 June 2016

RocksDB Architecture



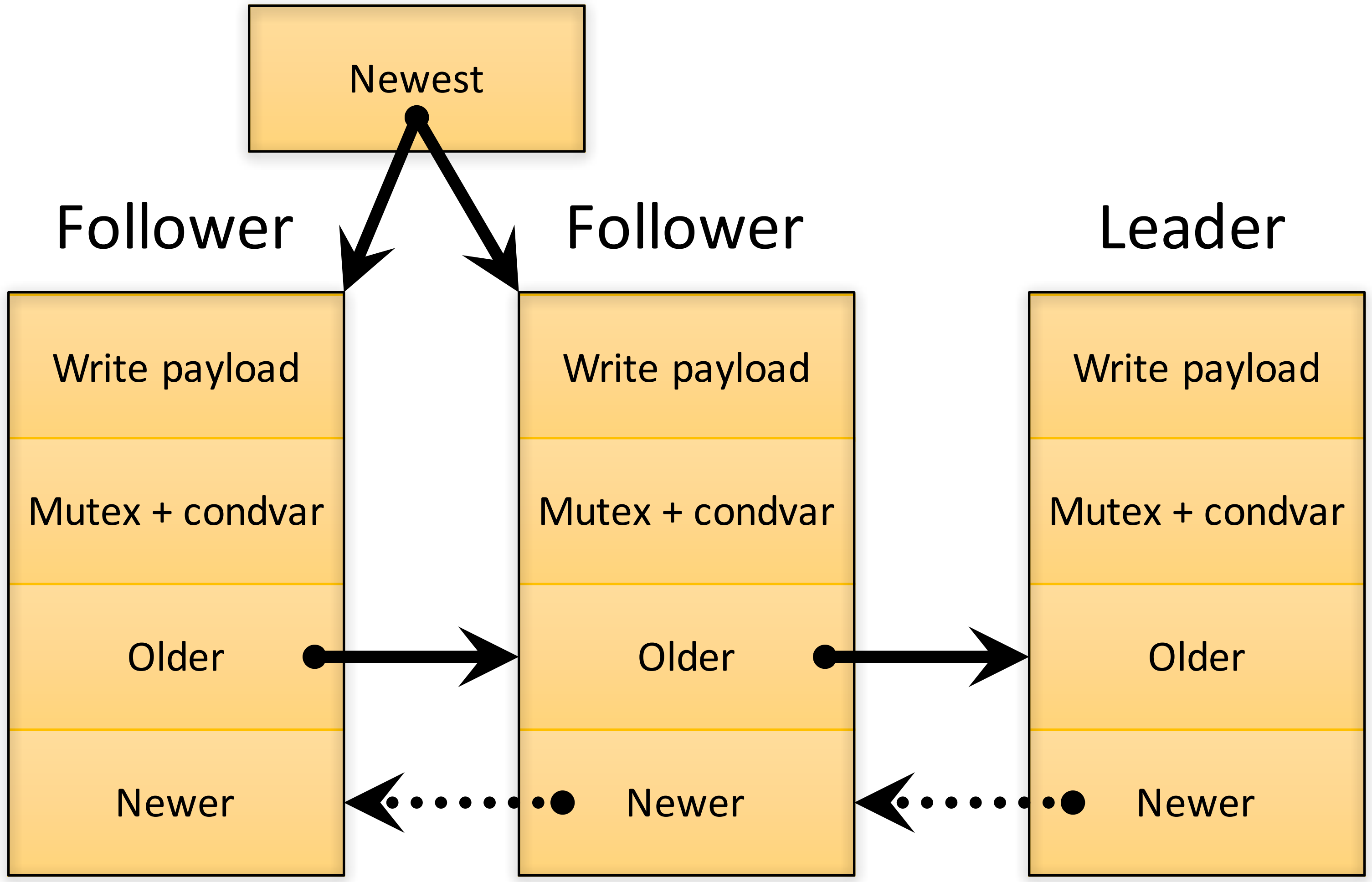
Write Logic


Housekeeping, Gather Pending
Write Ahead Log
Memtable Insert
Advance LastSequence
W
W
W
W
Write Ahead Log
Memtable Insert
Write Ahead Log
Memtable Insert
Advance LastSequence
Threads

Lock-free Write Group Construction

- Join by CAS-ing head
- Reverse links set later

- Follower never takes global mutex ☺
- Leader takes it once
- Group chosen after housekeeping work ☺



Concurrent Memtable Insertion?

- Guy et al., at Yahoo showed excellent scalability and perf, but …
  - New memtable type, slower for sequential use cases
  - New write path code, different throttling and compaction logic
  - Serializable but not linearizable, no read-your-writes guarantee
  - Long path to maturity

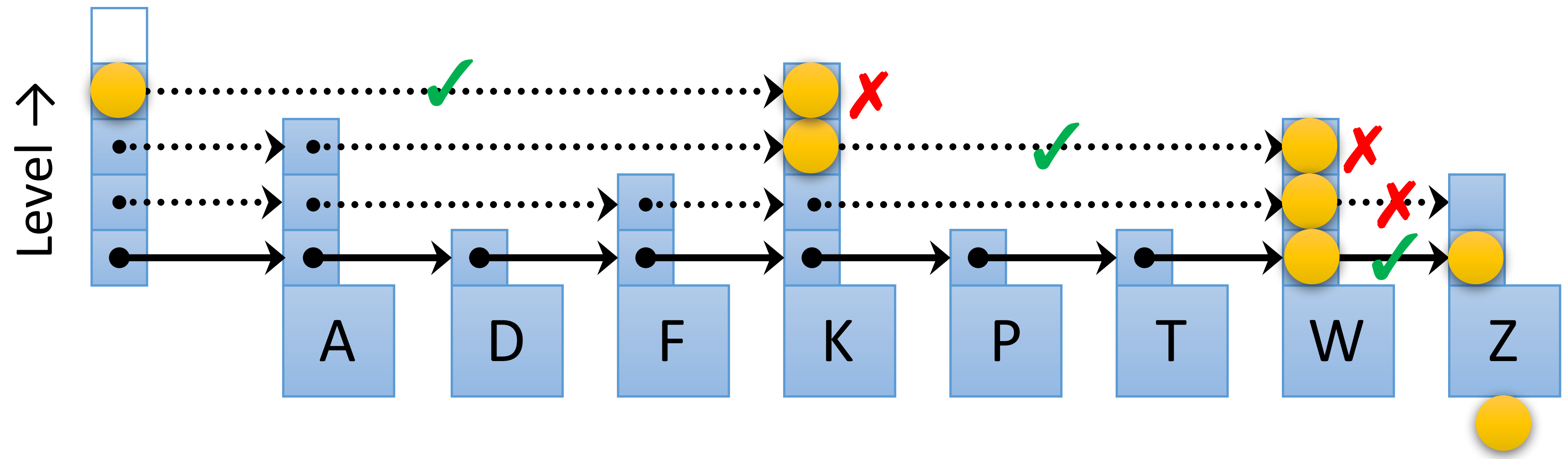
How much of the benefit can we capture without a new write path and without sacrificing linearizability?

My RocksDB Hack-a-month

- What I expected to be hard
  - Concurrent lock-free skip list
- What actually took the time
  - Lock-free write grouping
  - Moving to a thread-local random number generator (RNG)
  - Concurrent allocation of memtable memory
  - Lots of thread safety gaps in statistics and control logic
  - Sequential optimizations discovered along the way
  - Optimizing fine-grained inter-thread coordination

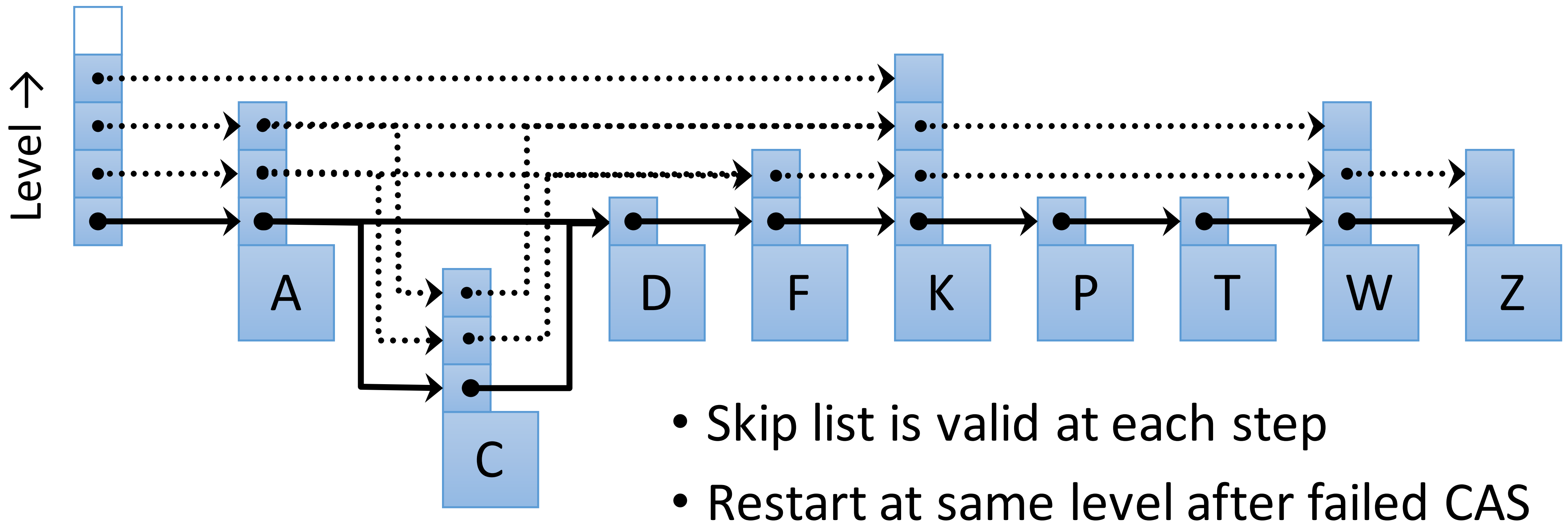
How to Search a Skip List

FindGE(“Y”)



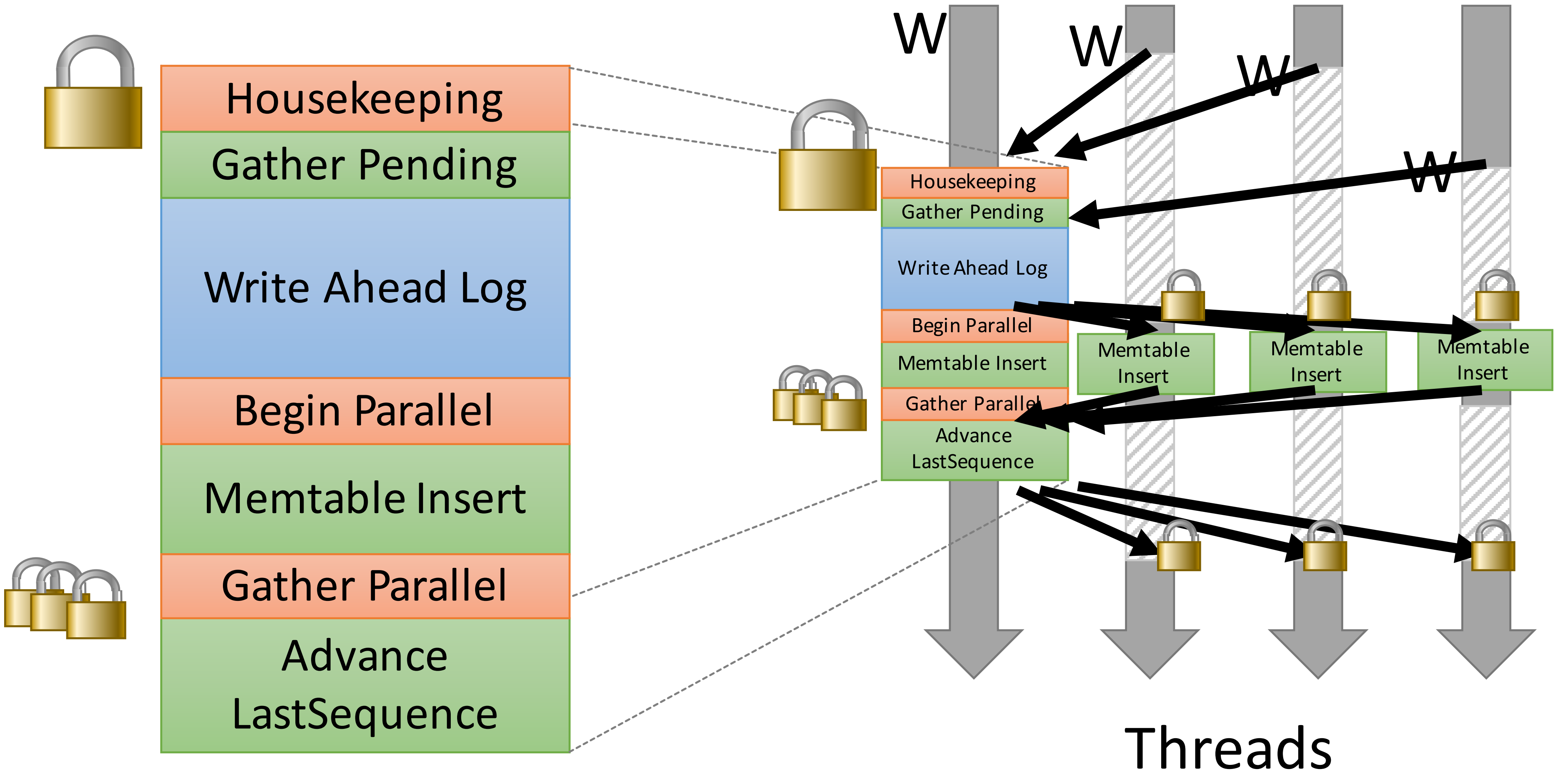
- Level 0 linked list has every element – encodes presence in list
- Level *i+1* has about ¼ of level *i* – allows $O(\log_4 n)$ search
- No rebalancing – node height chosen randomly during insertion

Concurrent Insertion

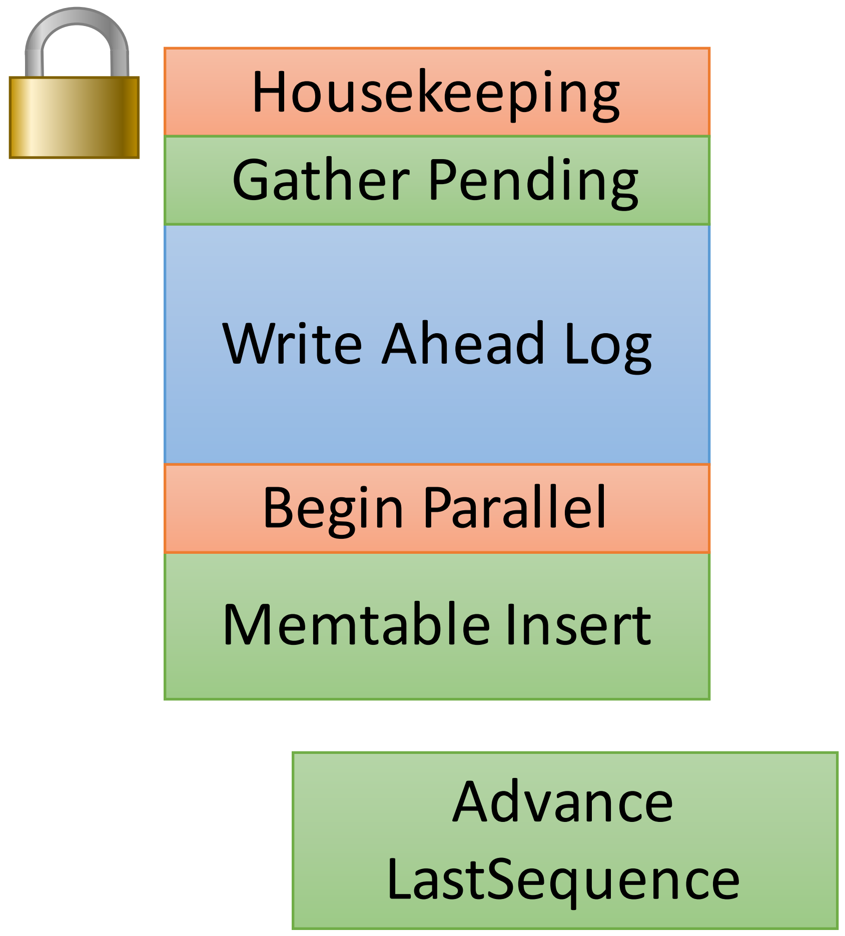


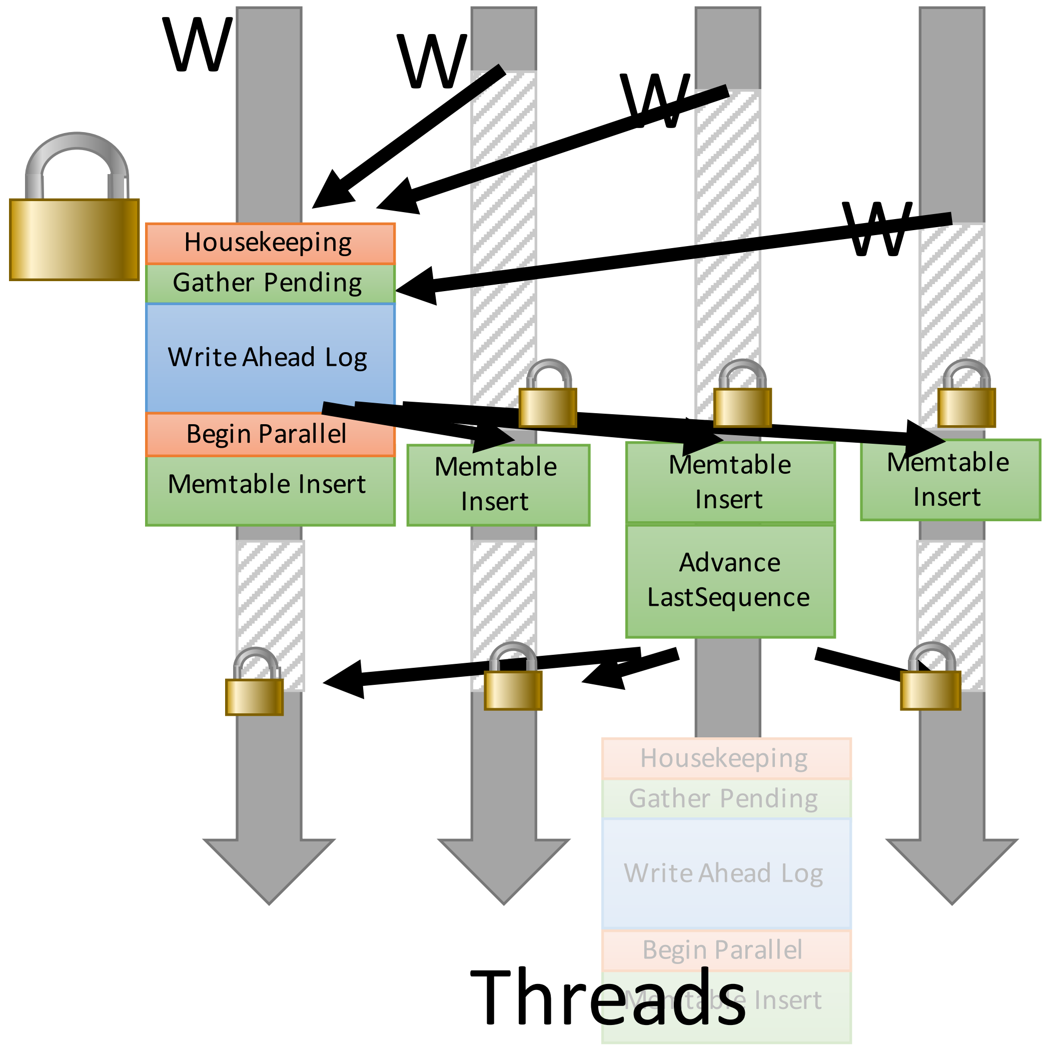
- Skip list is valid at each step
- Restart at same level after failed CAS
- Deletion is harder, but not needed

Concurrent Memtable Write



Concurrent Write: Early Exit





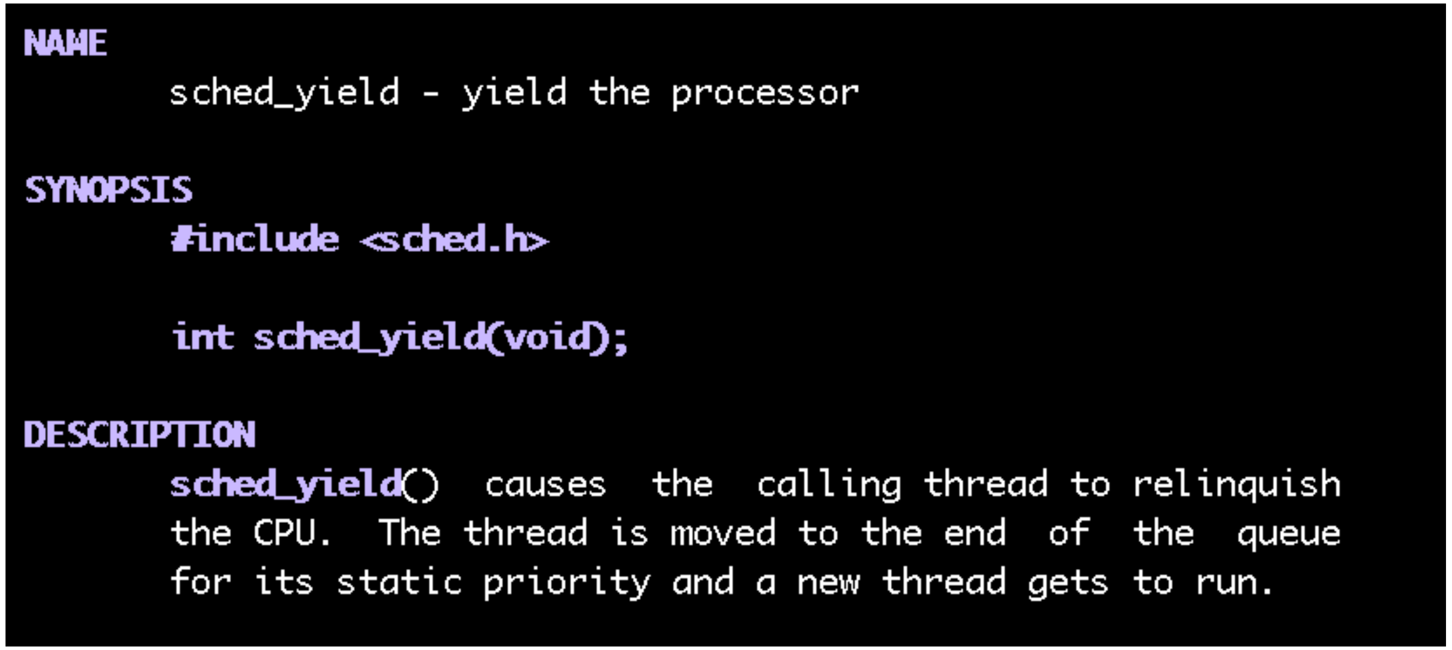
AwaitState’s spin/block tradeoff

```
while(!awoken) {
  if (good_chance_of_spin_success() &&
      os_runlist_has_little_work())
    selfishly_spin();
  else
    syscall(altruistically_suspend_thread);
}
```

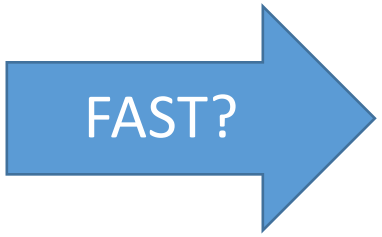
Spin?	Short wait	Long wait
Didn’t try (much)	**Bad**	**Good**
Successful	**Good**	**Selfish**
Unsuccessful	-	**Selfish**

“Soft yield” - ~~Dirty hack~~ Elegant heuristic

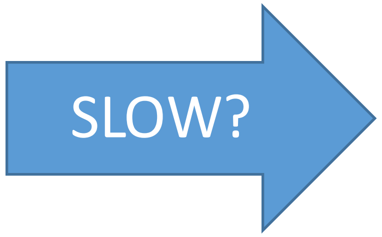
How do we query the OS runqueue in a portable fashion?



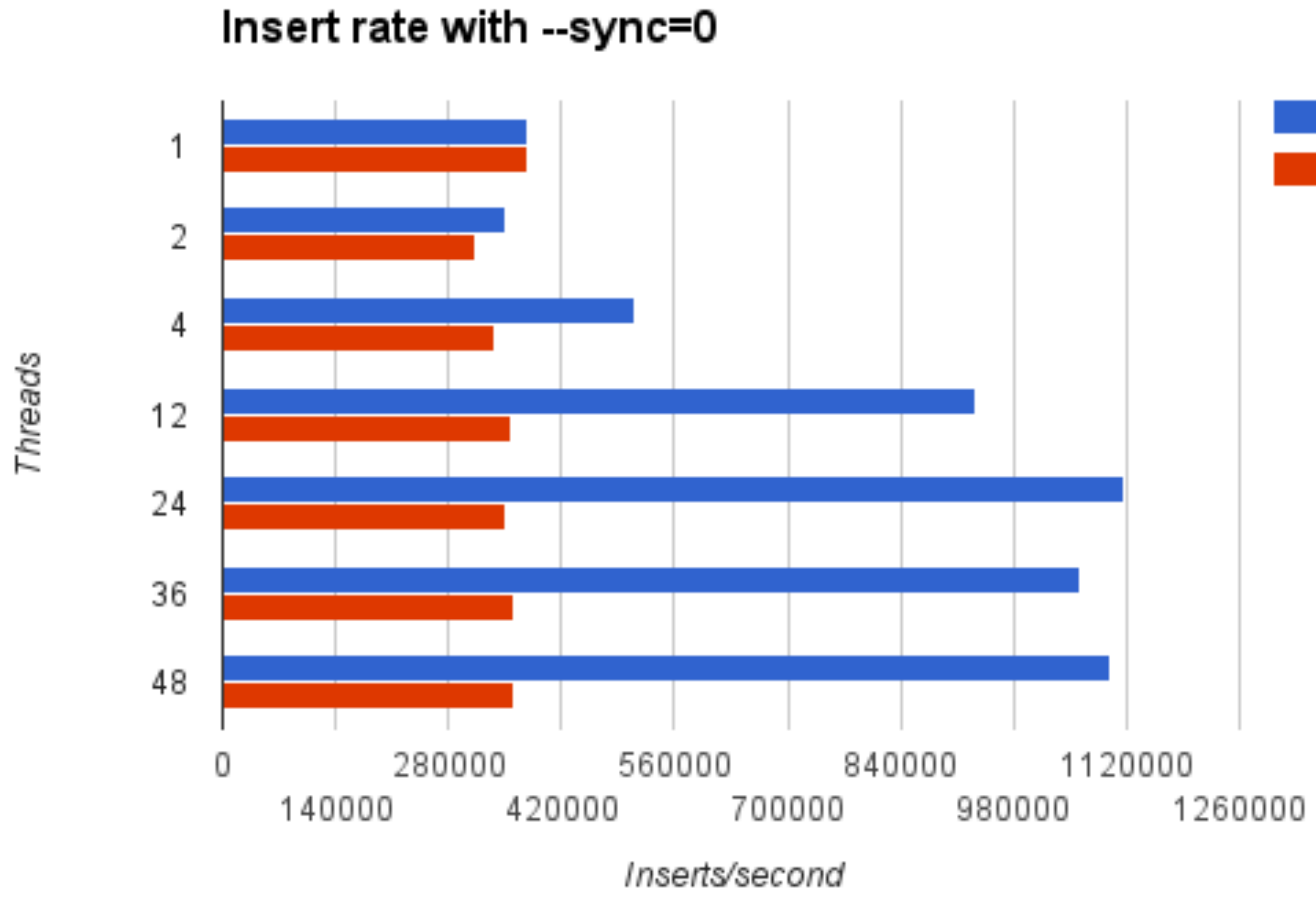
```
NAME
       sched_yield - yield the processor

SYNOPSIS
       #include <sched.h>

       int sched_yield(void);

DESCRIPTION
       sched_yield()  causes  the  calling thread to relinquish
       the CPU.  The thread is moved to the end  of  the  queue
       for its static priority and a new thread gets to run.
```



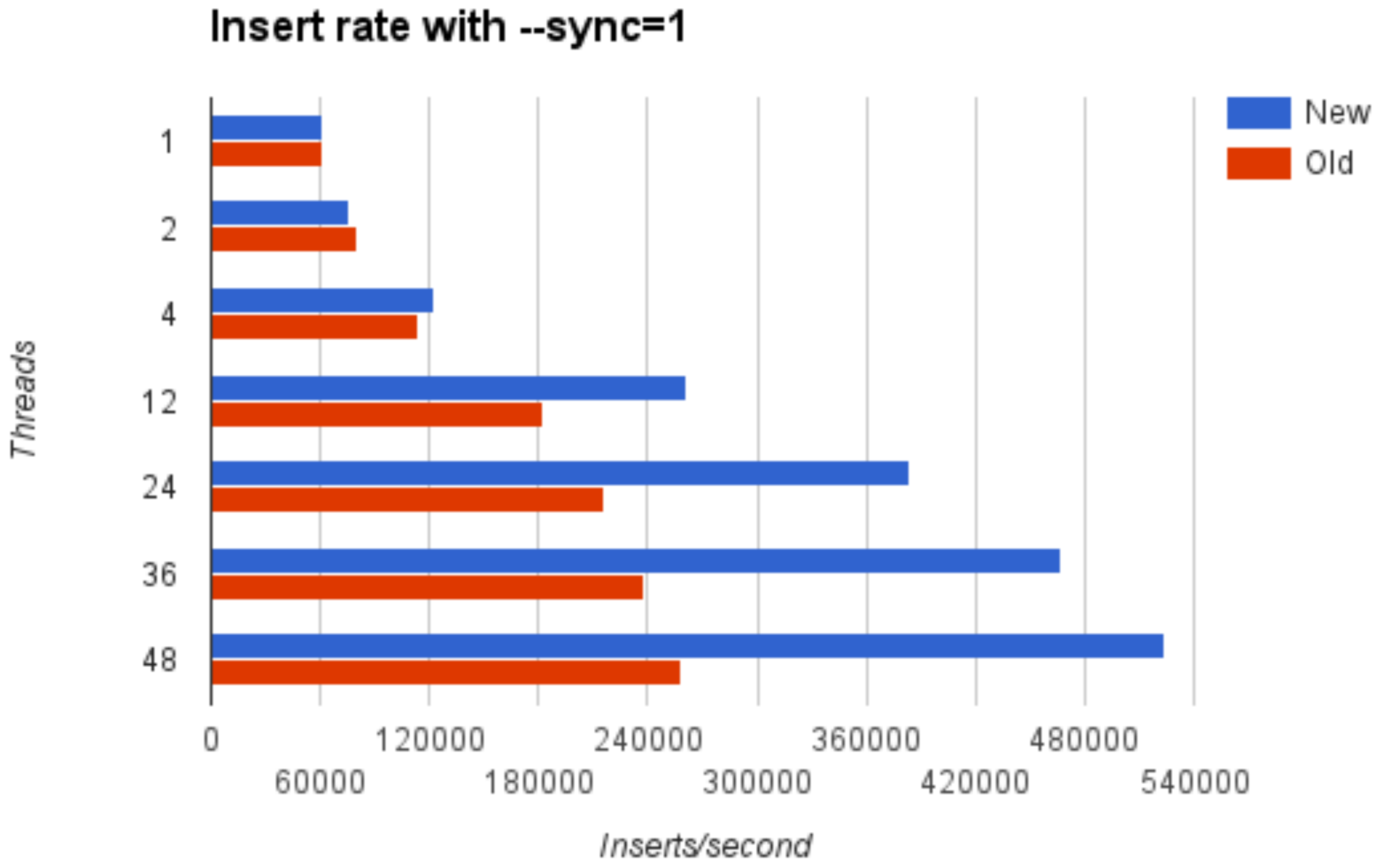
Spin more aggressively



Block right away



http://smalldatum.blogspot.com/2016/02/concurrent-inserts-and-rocksdb-memtable.html



http://smalldatum.blogspot.com/2016/02/concurrent-inserts-and-rocksdb-memtable.html

How to use it

Version >= 4.4

```
options.allow_concurrent_memtable_write = true;
options.enable_write_thread_adaptive_yield = true;
```