Caffeine

Design of a Modern Cache

https://github.com/ben-manes/caffeine

Cache

Caffeine is a high performance, near optimal cache

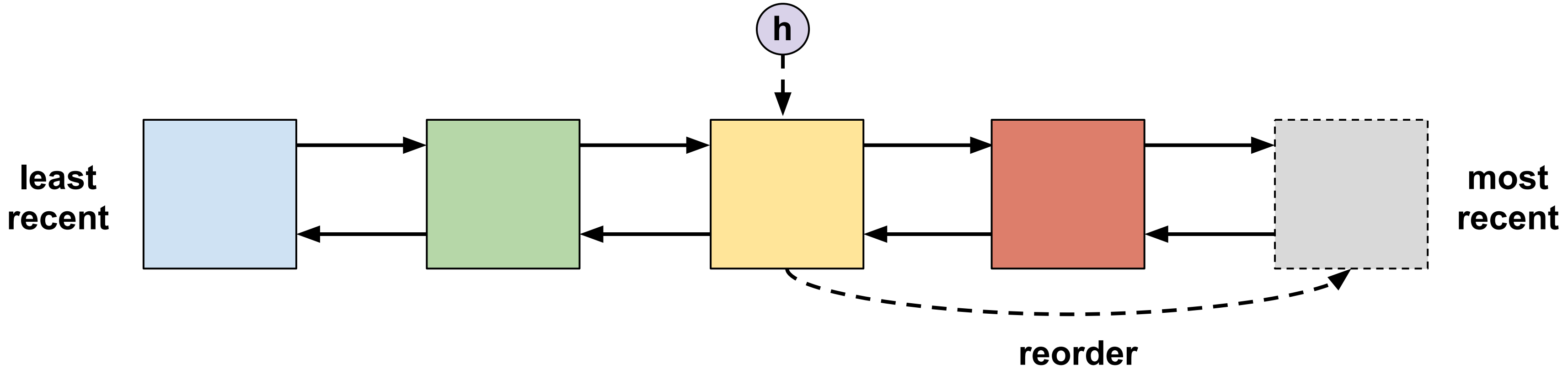
- Provides the familiar Guava Cache API
- Lots of features (memoization, maximum size, expiration, …)

```
LoadingCache<Key, Graph> graphs = Caffeine.newBuilder()
    .maximumSize(10_000)
    .expireAfterWrite(5, TimeUnit.MINUTES)
    .refreshAfterWrite(1, TimeUnit.MINUTES)
    .build(key -> createExpensiveGraph(key));
```

Let’s focus on the data structures

Least Recently Used

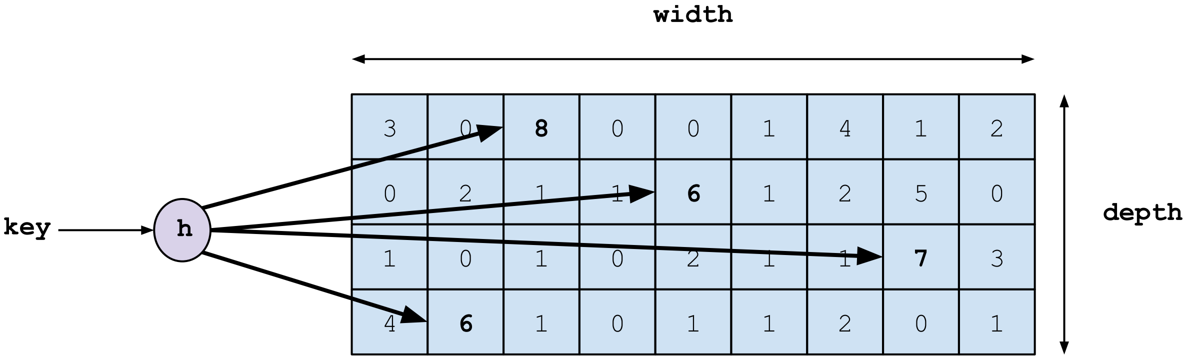
- Prioritizes eviction based on access order to prefer the most recent
- Suffers from cache pollution (“one hit wonders”)
- Retains a shallow history (working set only)
- O(1) time for add, remove, update
- Is not scan resistant



CountMin Sketch

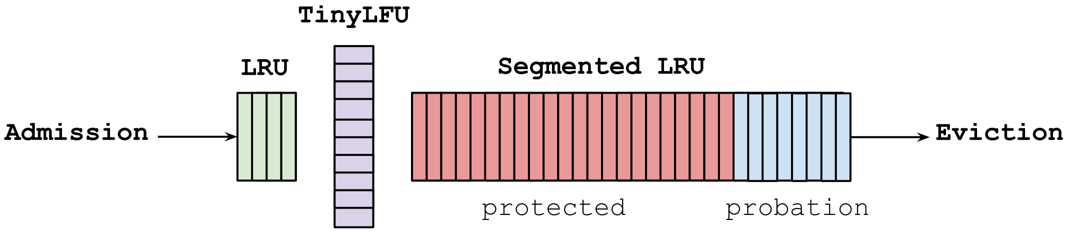
How to approximately find the top-K in a stream of events?

- Retain a Least Frequently Used cache of K entries
- Increment the hashed counters for each event
- Admit into the cache if *freq(event) > freq(victim)*

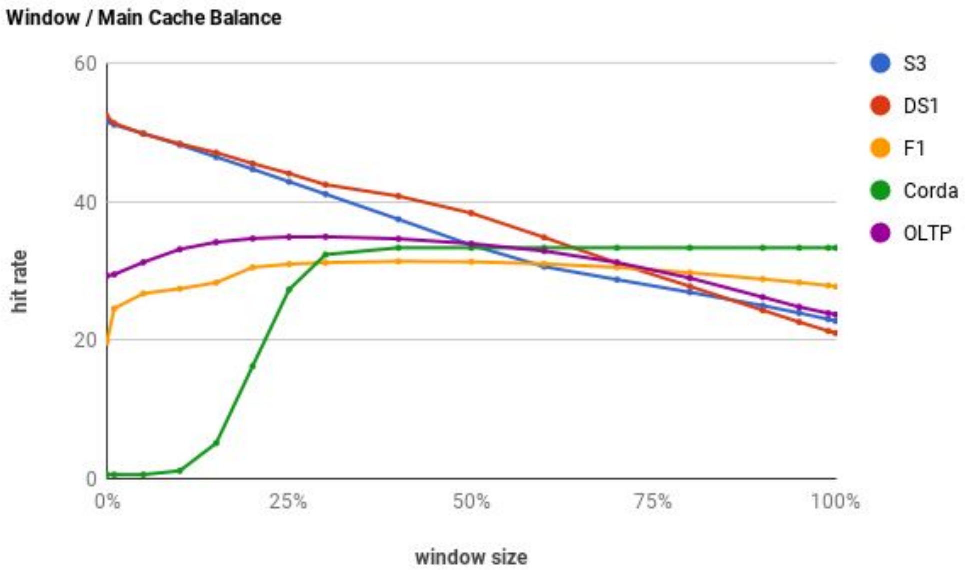


Window TinyLFU

- TinyLFU
  - Use small saturating counters (4-bit)
  - Reduce the number of counters by using a BloomFilter
  - Halve the counters every sample period (10x maximum size)
- Use an admission window to capture sparse bursts
- Use Segmented LRU to choose a better victim

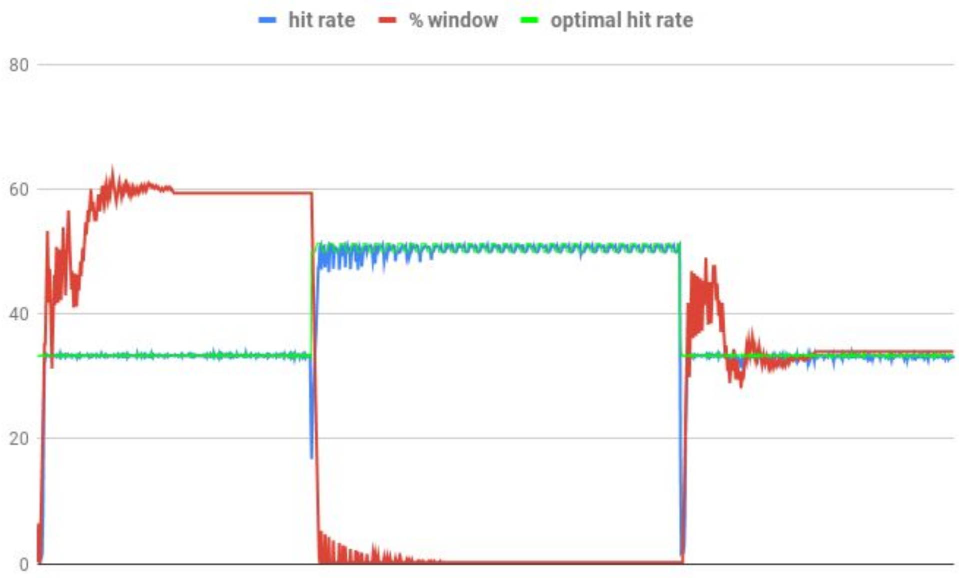


Adaptive Window (1)



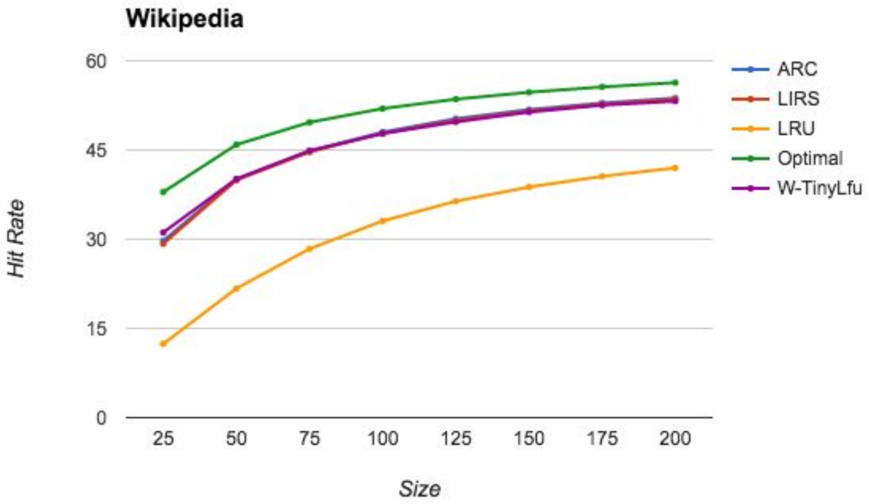
Optimize by Hill Climbing

Adaptive Window (2)

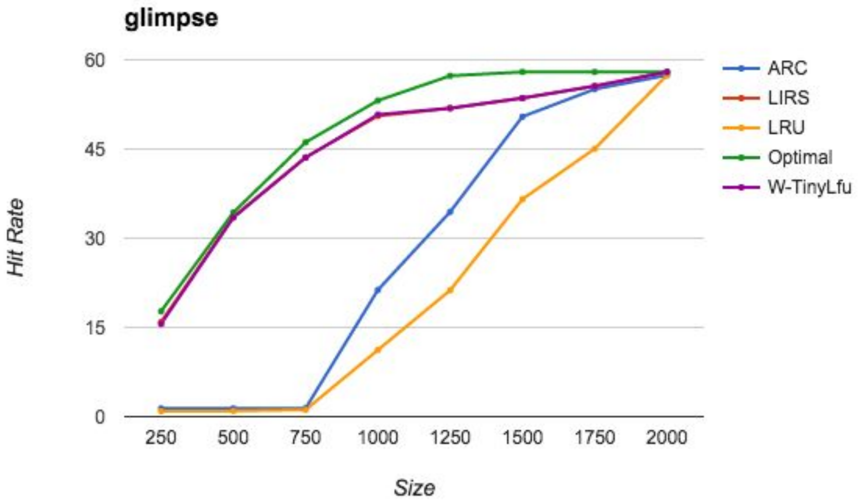


Recency ⇒ Frequency ⇒ Recency

Efficiency (1)

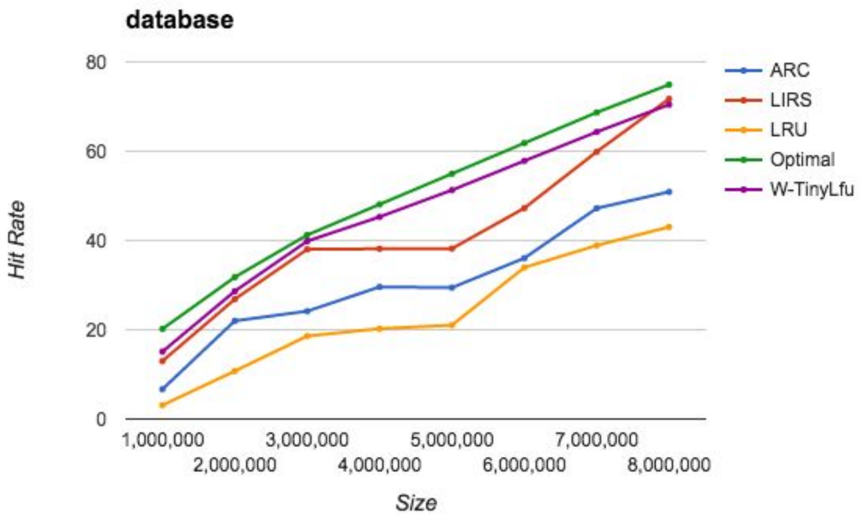


Website Popularity

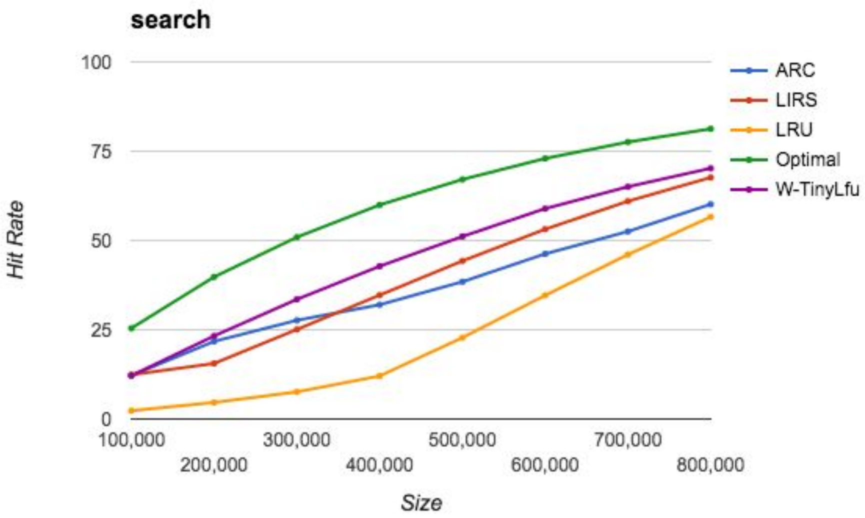


Multi-Pass Analysis

Efficiency (2)



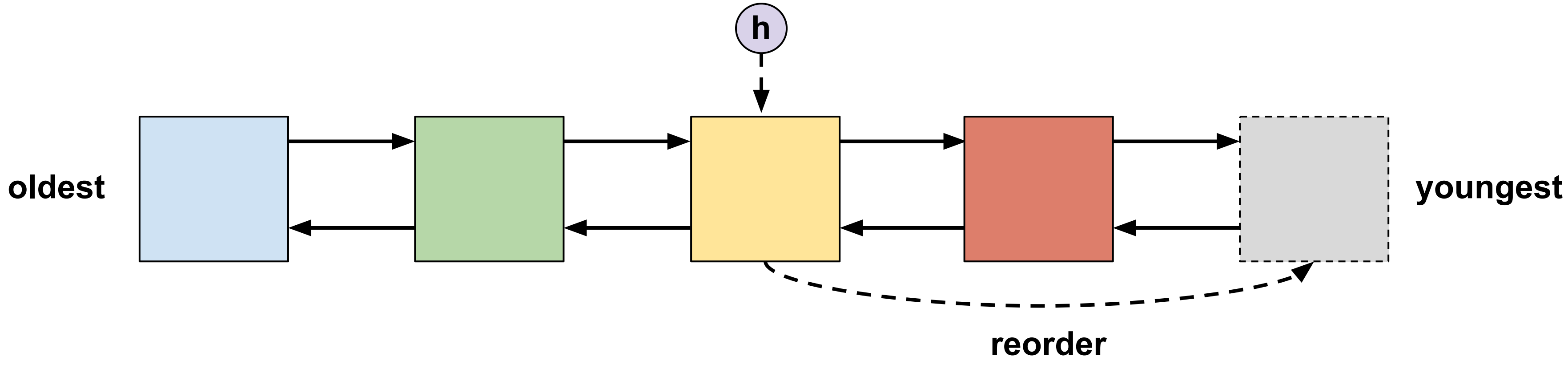
SQL Database



Document Search

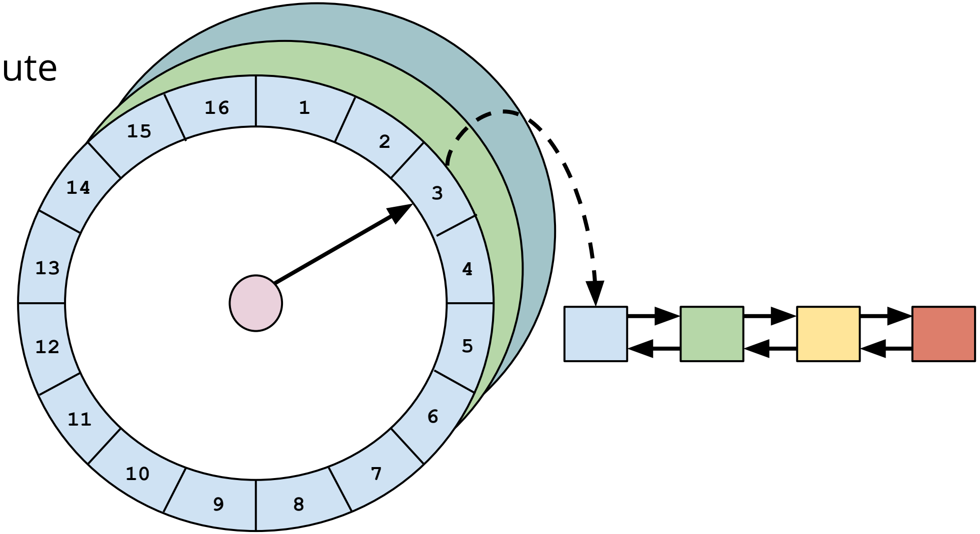
Fixed Expiration

- A time-bounded Least Recently Used policy
- Time-to-Idle reorders on every access
- Time-to-Live reorders on every write



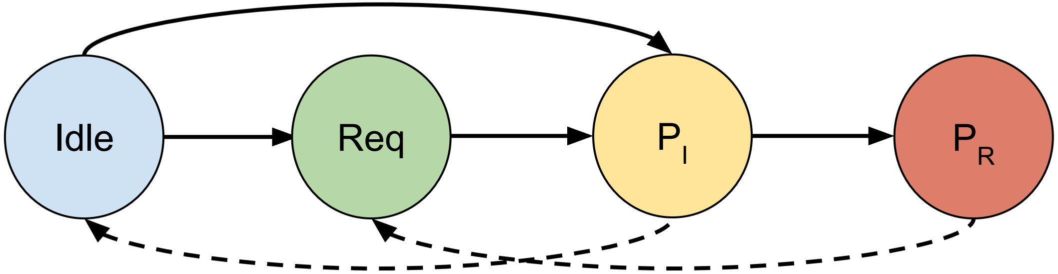
Variable Expiration

- Timer Wheels provide approximate timeouts by bucketing
  - Adding to a bucket requires hashing to the coarse resolution
  - A clock hand “ticks” as buckets expire and can be swept
- Hierarchical Timer Wheels
  - Cascade wheels from day, hour, minute
  - When advancing a larger wheel the timers are inserted into the smaller
- Optimize the cascade operation using bit manipulation tricks



Concurrency

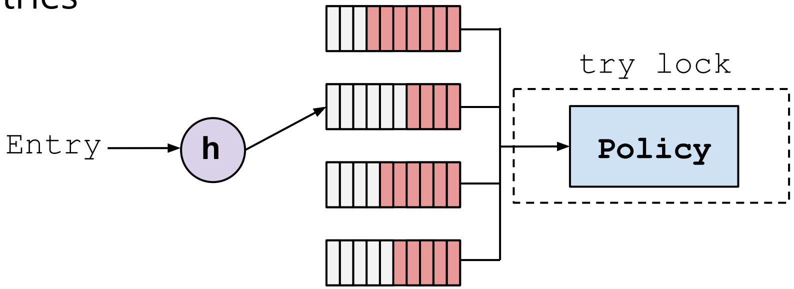
- Previously shown data structures are O(1), but not thread-safe
  - Naively locking or CAS'ing creates contention which causes slow down
  - Alternative, thread-safe data structures are O(lg n) or worse
- Write-Ahead Log (WAL), e.g. databases & file systems
  - Apply the operation immediately to the hash table
  - Record the change into a read or write buffer
  - Replay under lock in asynchronous batches



Replay States
- Idle
- Required
- Processing to Idle
- Processing to Required

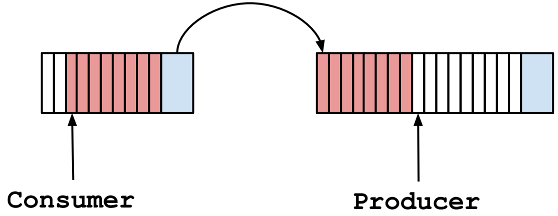
Read Buffer

- Striped ring buffers
  - Selects a buffer by the thread id, not the key's hash, to spread hot entries
  - Adds a new buffer when too many appends fail (CAS conflicts)
  - Based on Java's Striped64 (e.g. LongAdder)
- Lossy
  - May drop additions when full or exhausted retries
  - Due to popular entries being requested often
    the lost access history has little effect
- Triggers a maintenance cycle
  when a buffer is full

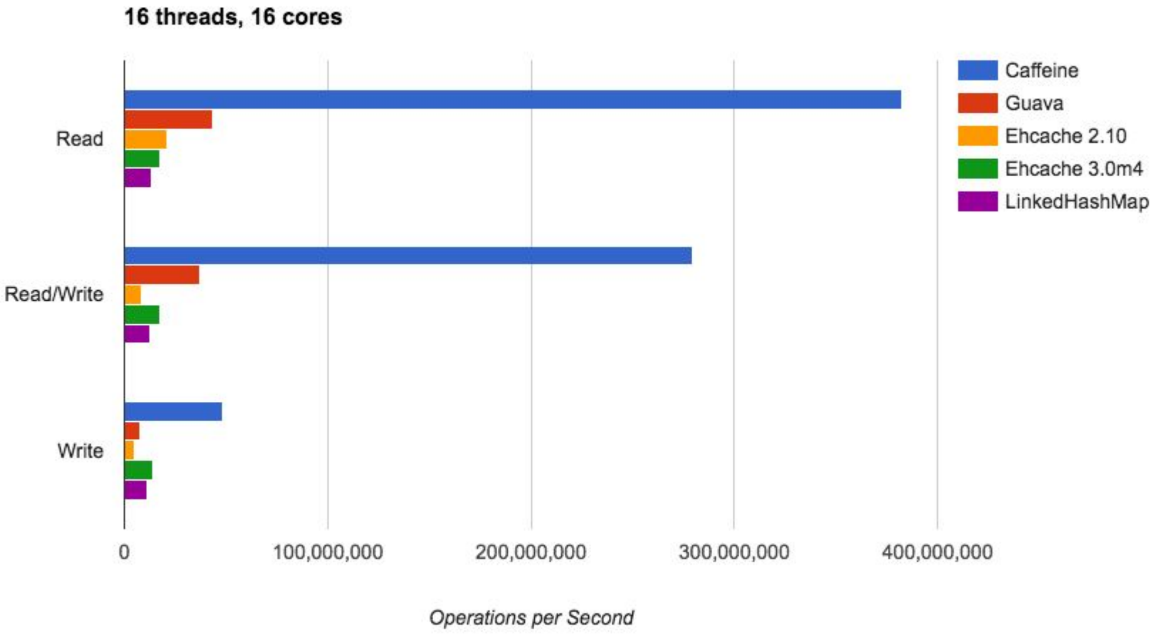


Write Buffer

- Bounded Ring Buffer
  - Grows from an initial size to a maximum, if required by the write rate
  - A forwarding pointer allows the consumer to discover the new array
  - JCTools’ MpscGrowableArrayQueue (embedded)
- Back pressure when full
  - Writers assist in performing the maintenance work
  - Rarely occurs as requires a write rate >> replay rate
- Triggers a maintenance cycle immediately



Performance



Last Words

- Decouple and break down problems to optimize them individually
- Combine simple data structures for efficient, powerful features
- Utilize these *performance mantras*
    - Don't do it
    - Do it, but don't do it again
    - Do it cheaper
    - Do it less
    - Do it later
    - Do it when they're not looking
    - Do it concurrently