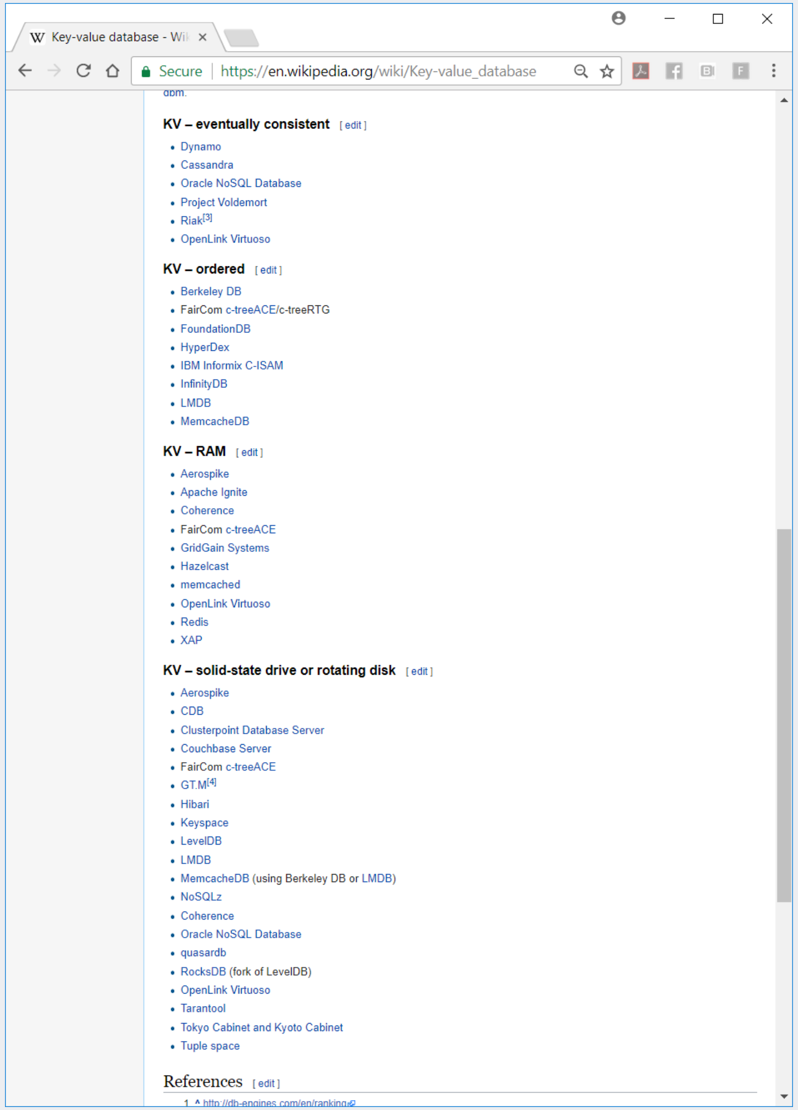
Workload Diversity with RocksDB

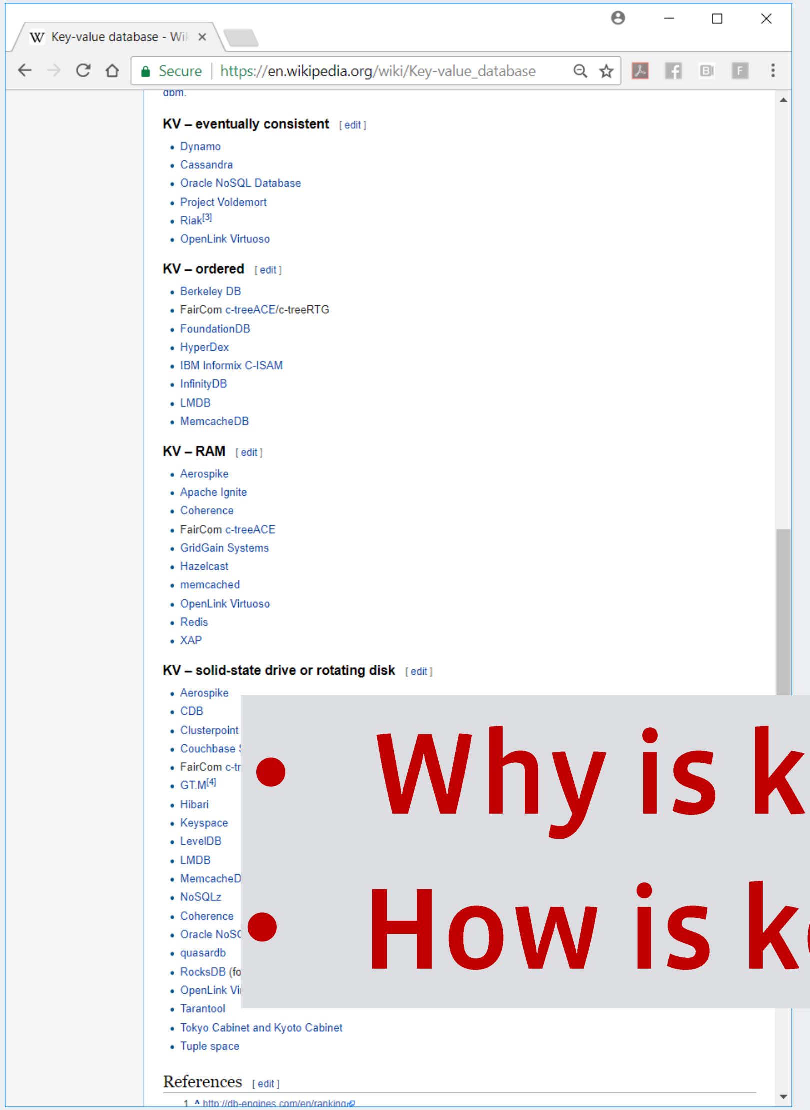
Siying Dong, Database Engineering Team, Facebook

Oct 9, 2017

Key-Value Stores are Popular



Key-Value Stores are Popular



- **Why is key-value store popular?**
- **How is key-value store used?**

RocksDB

Key-Value Storage on Log-Structure Merge-Tree

RocksDB is versatile

RocksDB Application Diversity

- Inside Facebook:
  - **MyRocks**: MySQL Engine
  - **ZippyDB**: distributed Key-value store
  - **Laser**: data publishing service
  - **Dragon**: distributed graph query engine
  - **LogDevice**: distributed data store for logs
  - **Stylus**: stream processing framework

Workload Diversity

RocksDB Workload Diversity

1. Document Store
2. Social Graph Edges
3. Time-Ordered Events
4. Counter Service
5. Storage For Logs
6. Cache

RocksDB Workload Diversity

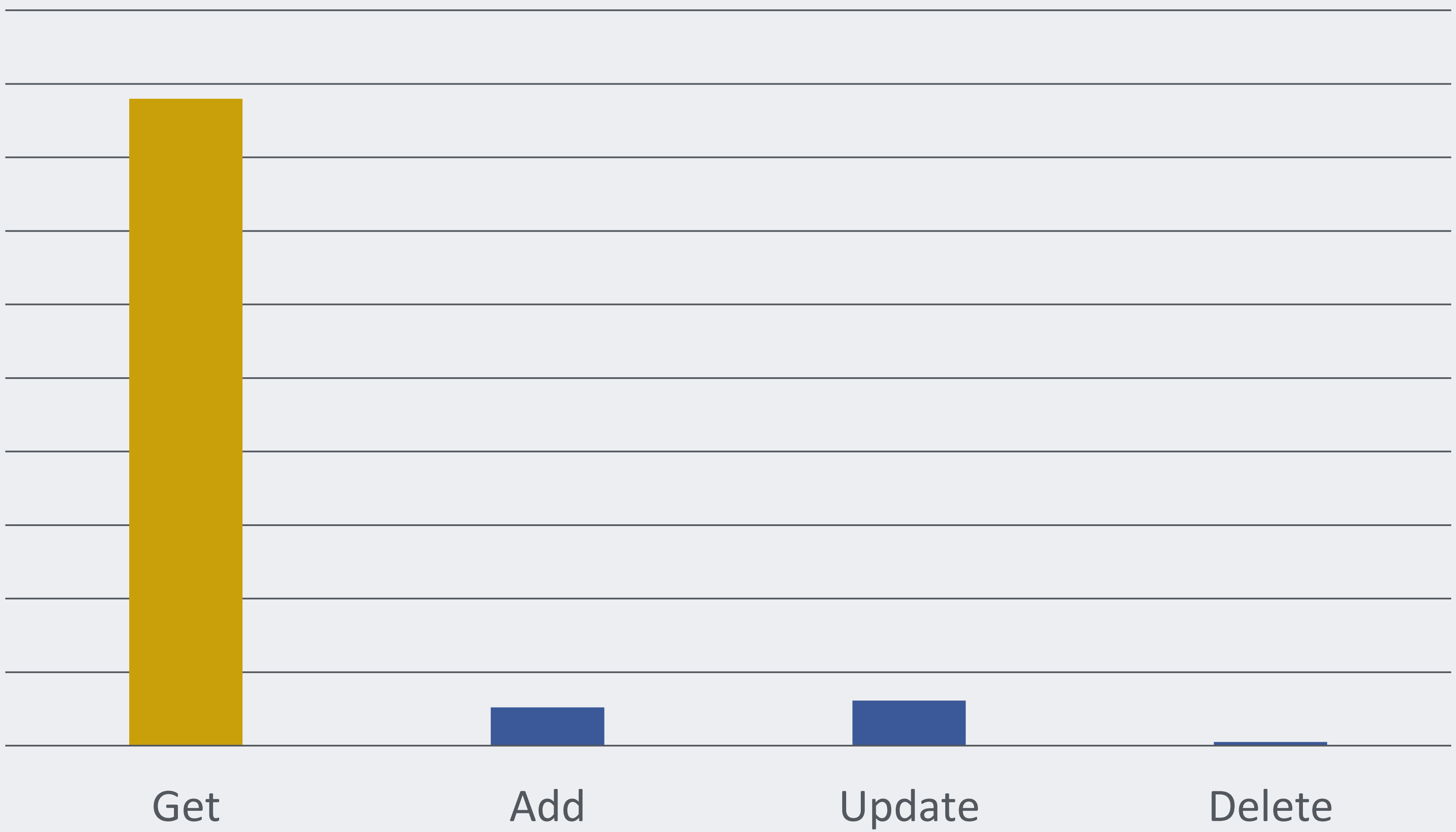
1	Document Store
2	Social Graph Edges
3	Time-Ordered Events
4	Counter Service
5	Storage For Logs
6	Cache

Document Store Example: Tao object

- TAO: Facebook's Distributed Data Store for the Social Graph
- Sample object:

```
'timestamp' => 1505514049,
 'author' => 100008855205466,
 'message_type' => 308,
 'body' => 'Performance results for load, create index and table scan for ......',
 'gid' => 530977083778847,
 'lang' => 'en',
 'creator_id' => 100008855205466,
 'detected_dialect' => 'en_XX',
 'detected_dialect_confidence' => 100,
 'modified_time' => 1505514292,
......
```

Tao Object: Operation Ratio



Tao Object: Object Size

	Number of Bytes
Mean	168
Median	78
P75	246
P90	441
P95	733
P99	1688

Document encoding

- Customized Format
- C++ struct
- Thrift (RPC Protocol) struct
- JSON/BSON

Workload

- Point lookup only
- Balance Read/write.

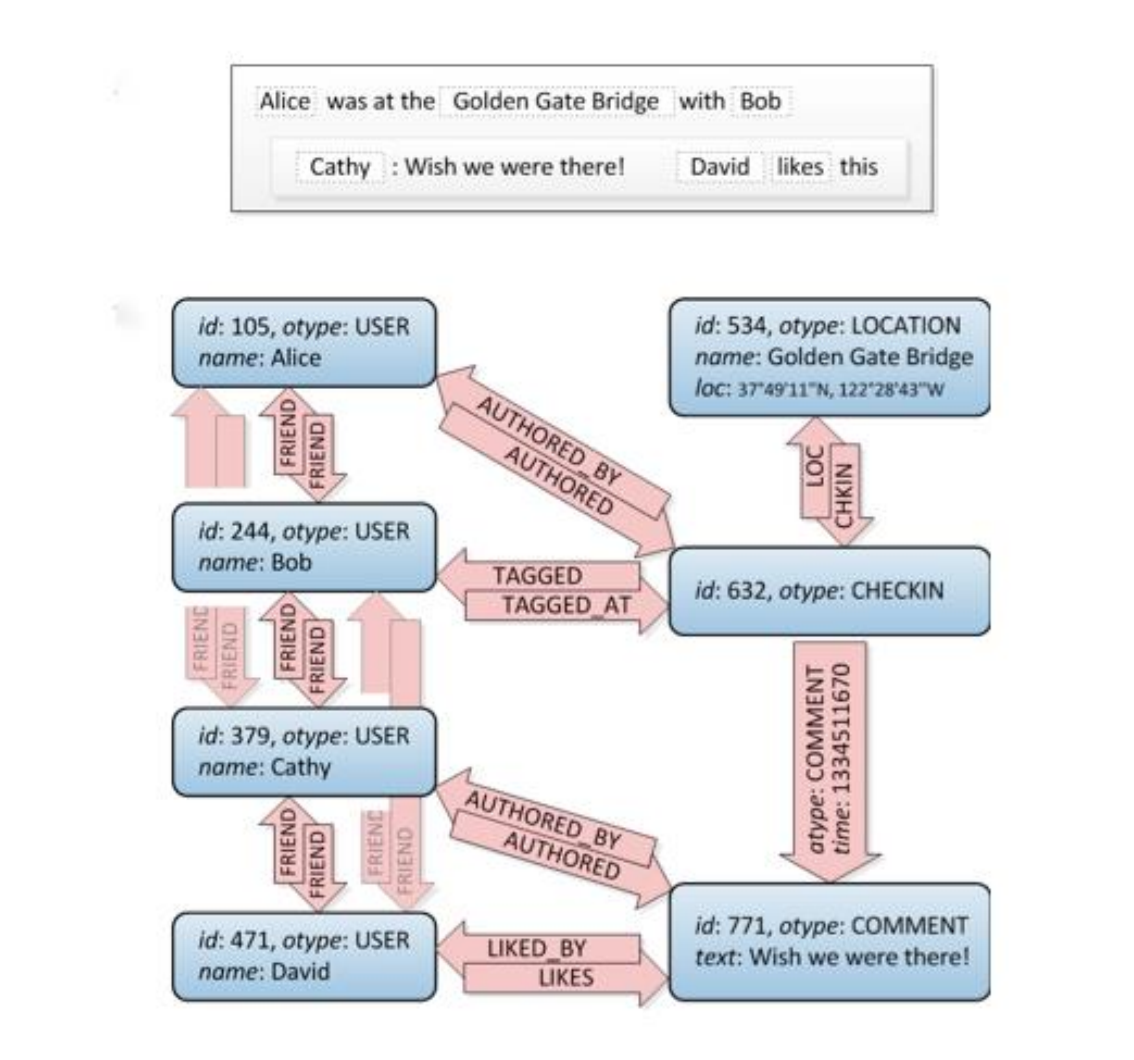
Update a small subset of attributes?

- Read-modify-write: read optimized
- Blindly write delta and merge in read: write optimized

RocksDB Workload Diversity

1. Document Store
2. Social Graph Edges
3. Time-Ordered Events
4. Counter Service
5. Storage For Logs
6. Cache

Example: Tao "associations"



FB page => admin

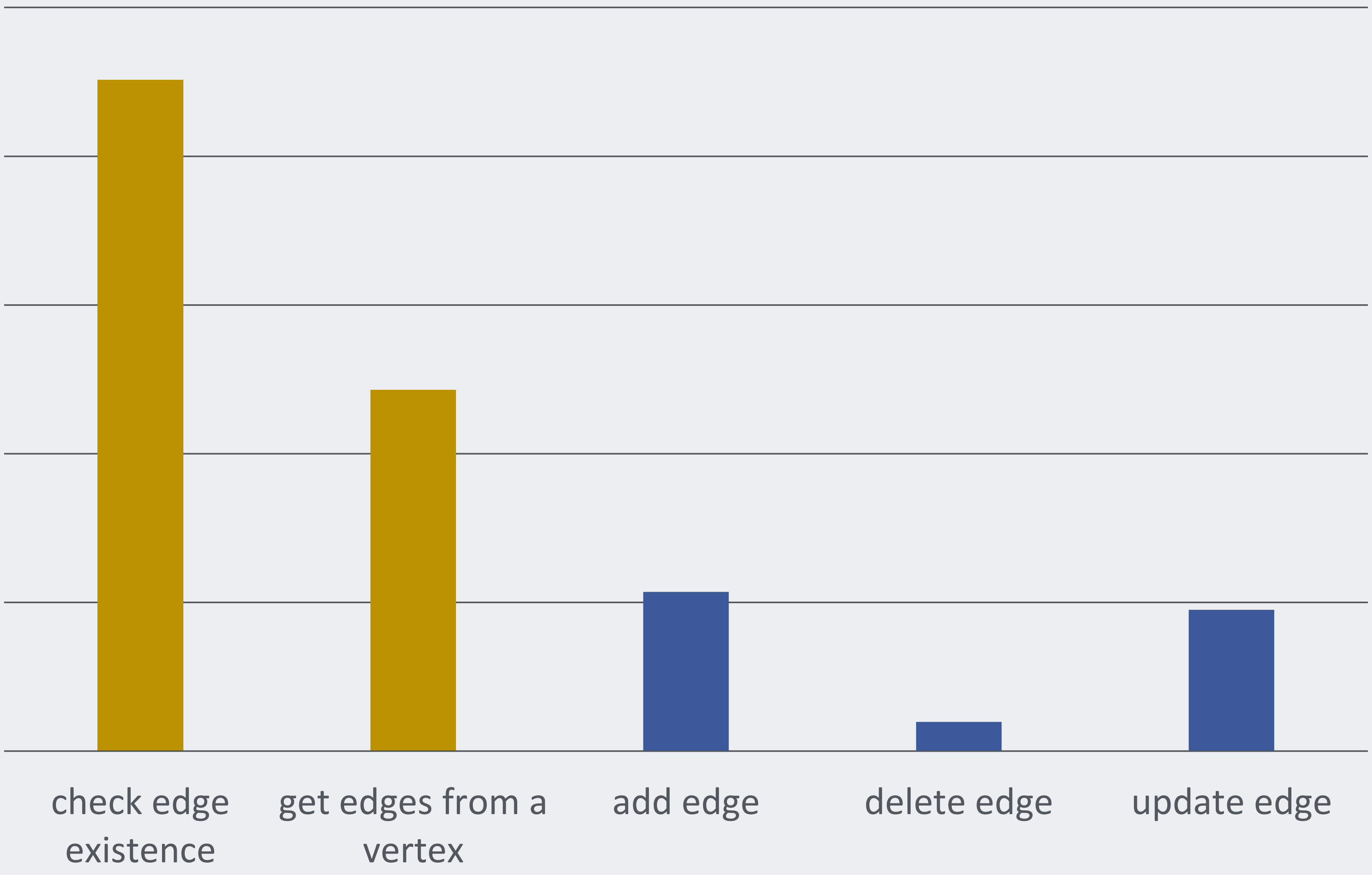
- Key: [*page_id*, *user_id*]
- Value: [*update_time*, *is_deleted*...]
- Common Query: find admins of a page of *page_id*:
  - Scan keys in the range prefixed with *page_id*

Number of Edges Per Vertex

	Number of Edges
Mean	3.85
Median	1
P75	1
P95	3
P99	22

Optimization: bloom filter using range prefix

Tao associations: Operation Ratio



Ranked Comments of a post

- Key: [*post_id*, *language_id*, *ranking_score*, *comment_id*]
- Value: (empty)
- Common Query: find top ranked comments of *post_id*:
  - Scan the first N keys between

    [*post_id*, *viewer_language*, *min_score*] to [... *max_score*]

RocksDB Workload Diversity

1. Document Store
2. Social Graph Edges
3. Time-Ordered Events
4. Counter Service
5. Storage For Logs
6. Cache

Example: Recent Time-Ordered Events

- Key: [*user_id*, *event_timestamp*] with *timestamp* in reverse order
- Value: some metadata of the event.
- Read Query: range scan between [*user_id*, *max_ts*] *to* [*user_id*, *min_ts*] limit N

Workload of the example

- Write/Read 63:1
- Average keys per range query: ~1,100
- Average value size per key: 230 bytes

Retire Old Time-Ordered Events

- Time-Ordered Events should retire if
  - It is too old
  - Too many Time-Ordered Events from a user
- Solution: compaction filter

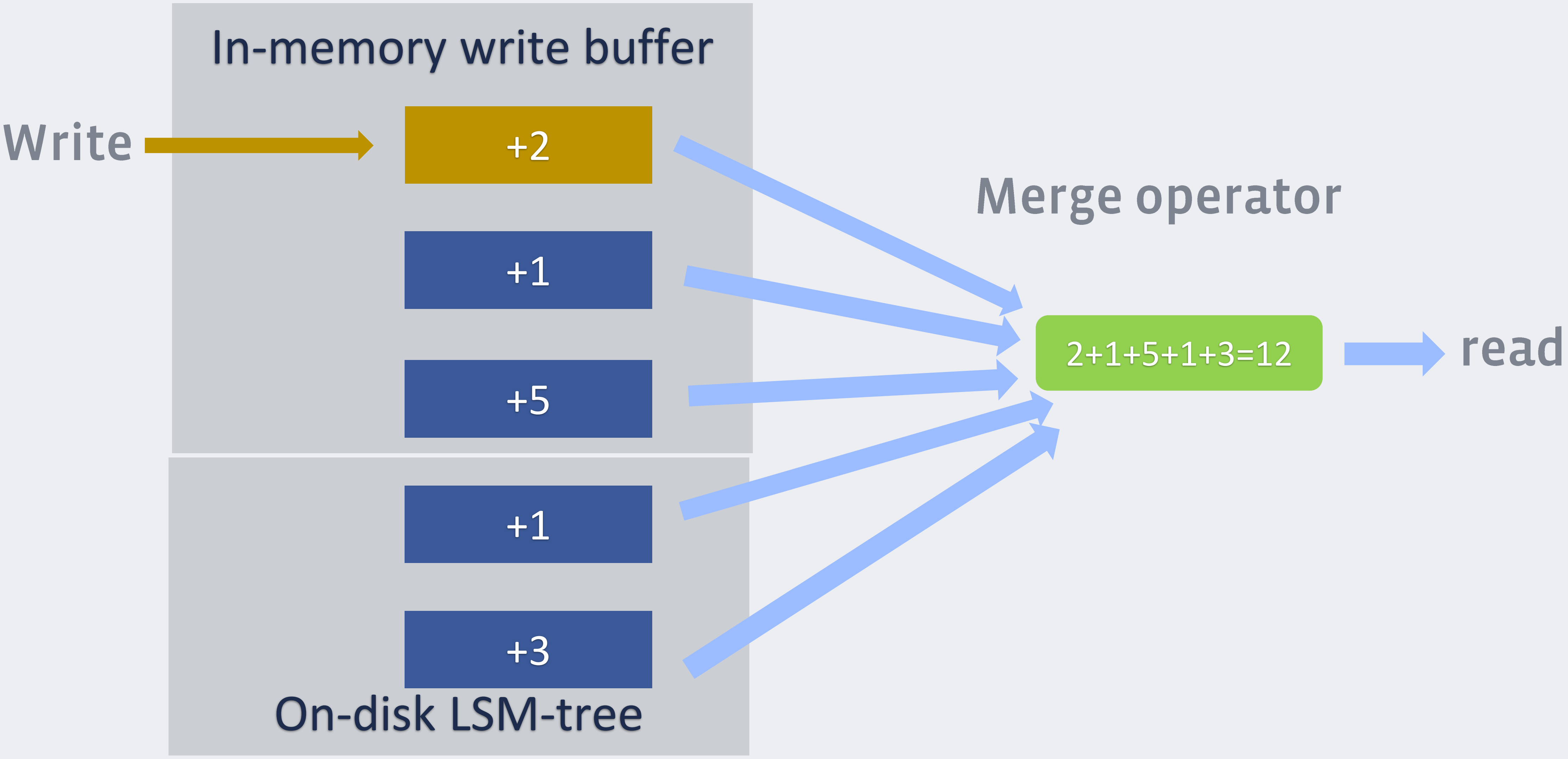
RocksDB Workload Diversity

1	Document Store
2	Social Graph Edges
3	Time-Ordered Events
4	Counter Service
5	Storage For Logs
6	Cache

Counter Service

- Key: counter_id
- Value: the counter. Common counter types:
  - Plain count
  - Unique count (using hyperloglog)
  - Auto-decaying Counter Service
- Usually update heavily

Delta Updates (merge operator)



Workload in one use case

- 180K key updates + 19K keys read per second per host
- Average key-update size 26 bytes
- How many delta entries to merge when read:
  - Median: 7
  - P90: 801

RocksDB Workload Diversity

1	Document Store
2	Social Graph Edges
3	Time-Ordered Events
4	Counter Service
5	Storage For Logs
6	Cache

Storage For Logs

- Key: [*logging_id*, *seq_id*] where *seq_id* is incremental
- Value: message contents
- Common Query: range scan from [*logging_id*, *last_seen_seq_id*] for all keys of *logging_id*.
- Write-heavy, most reads are against recent updates.
- Older data is deleted.

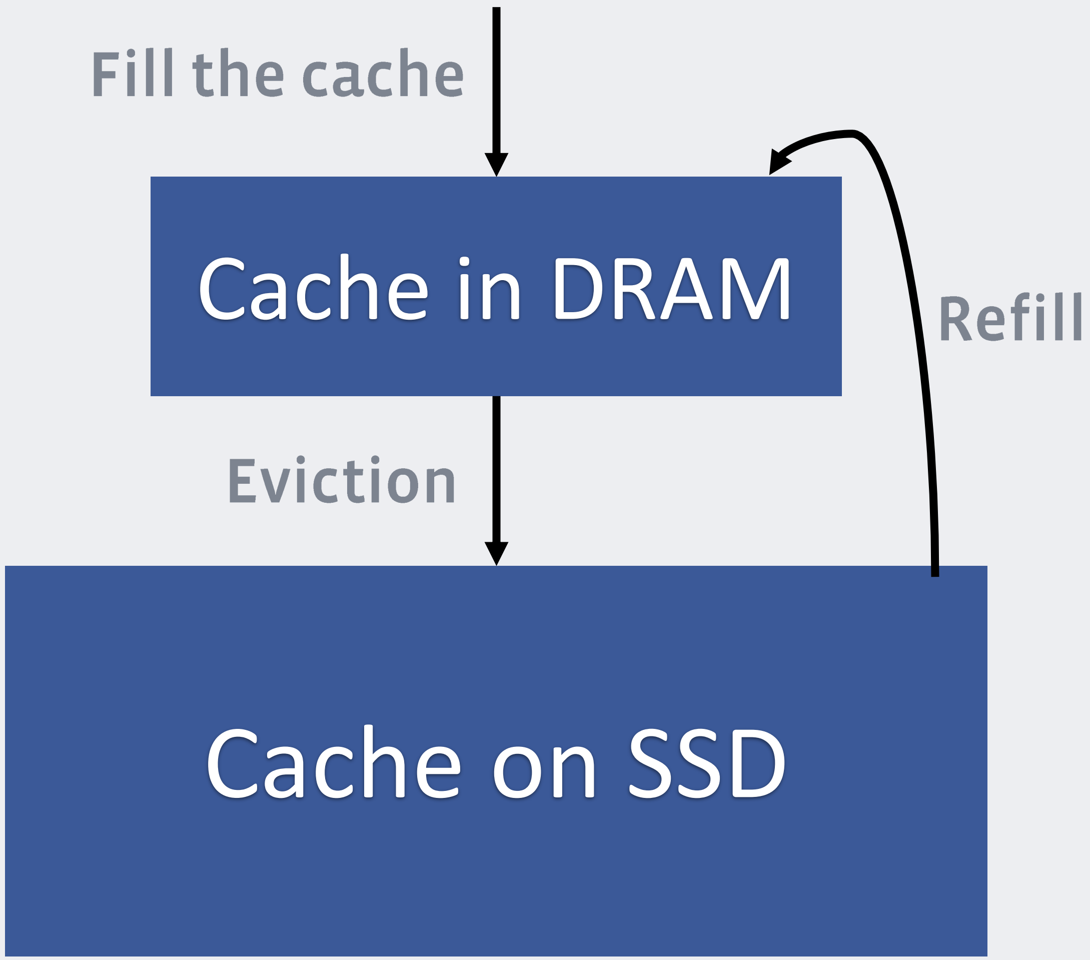
Example: A LogDevice Use Case

- For a host:
  - 260K keys inserted per second
  - 350K keys read per second
  - Every read gets 1.13 keys
  - Average log size 80 bytes

RocksDB Workload Diversity

1. Document Store
2. Social Graph Edges
3. Time-Ordered Events
4. Counter Service
5. Storage For Logs
6. Cache

RocksDB as a Cache: an example



RocksDB as a Cache: an example

- Put()/Get() only.
- 3.4K write/s, 14K read/s per host
- Hit rate about 13%
- Write heavy, sometimes read heavy too.
- Evict old data in FIFO.

	Number of Bytes
Mean	422
Median	141
P95	340
P99	4000

RocksDB as a Cache: Some Tuning Experience

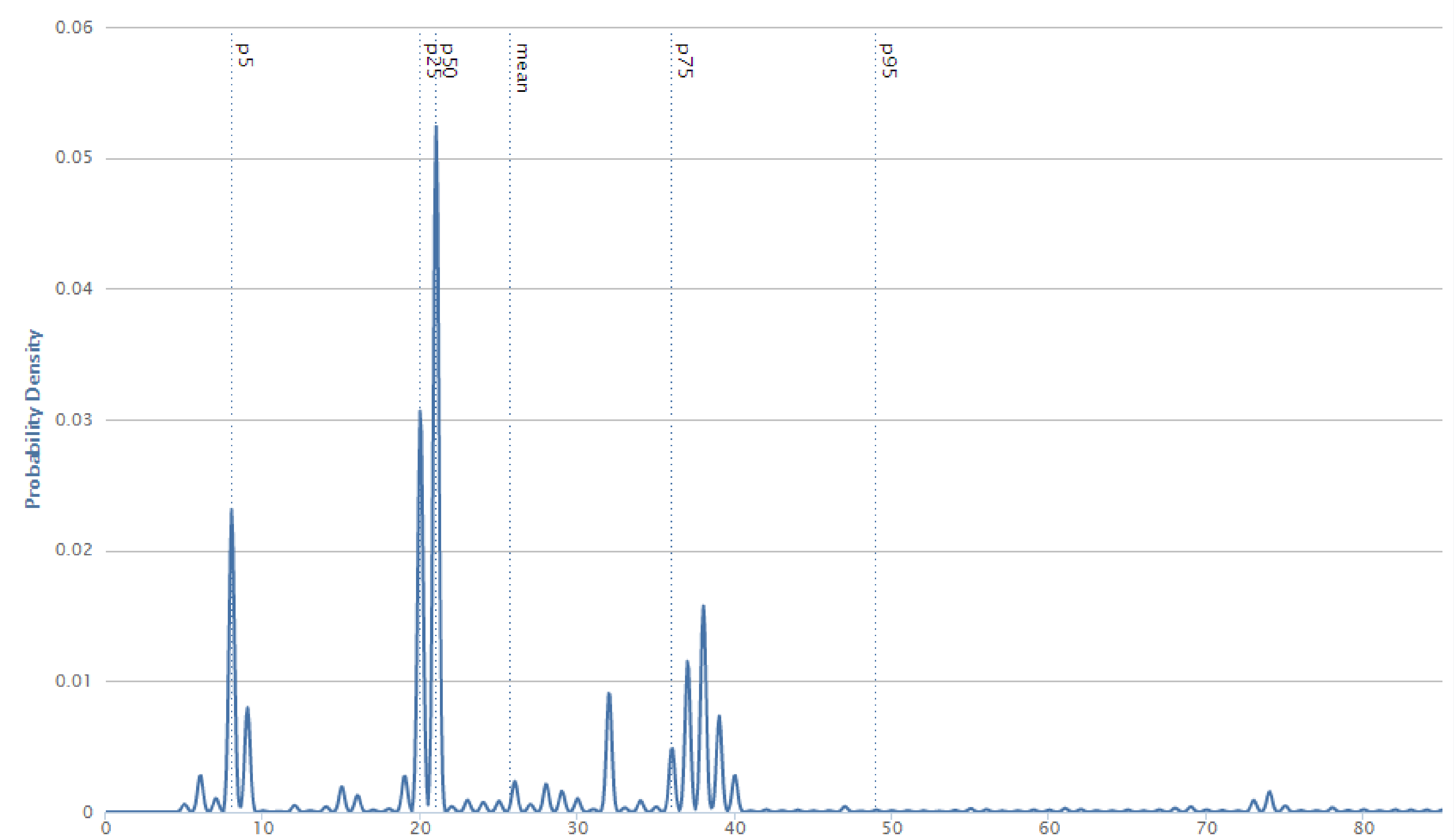
- Minimize compaction
- Be careful about bloom filter false positive rate
- Drop oldest data directly
- Key and value should be stored separately for large values.

RocksDB Workload Diversity

1	Document Store
2	Social Graph Edges
3	Time-Ordered Events
4	Counter Service
5	Storage For Logs
6	Cache

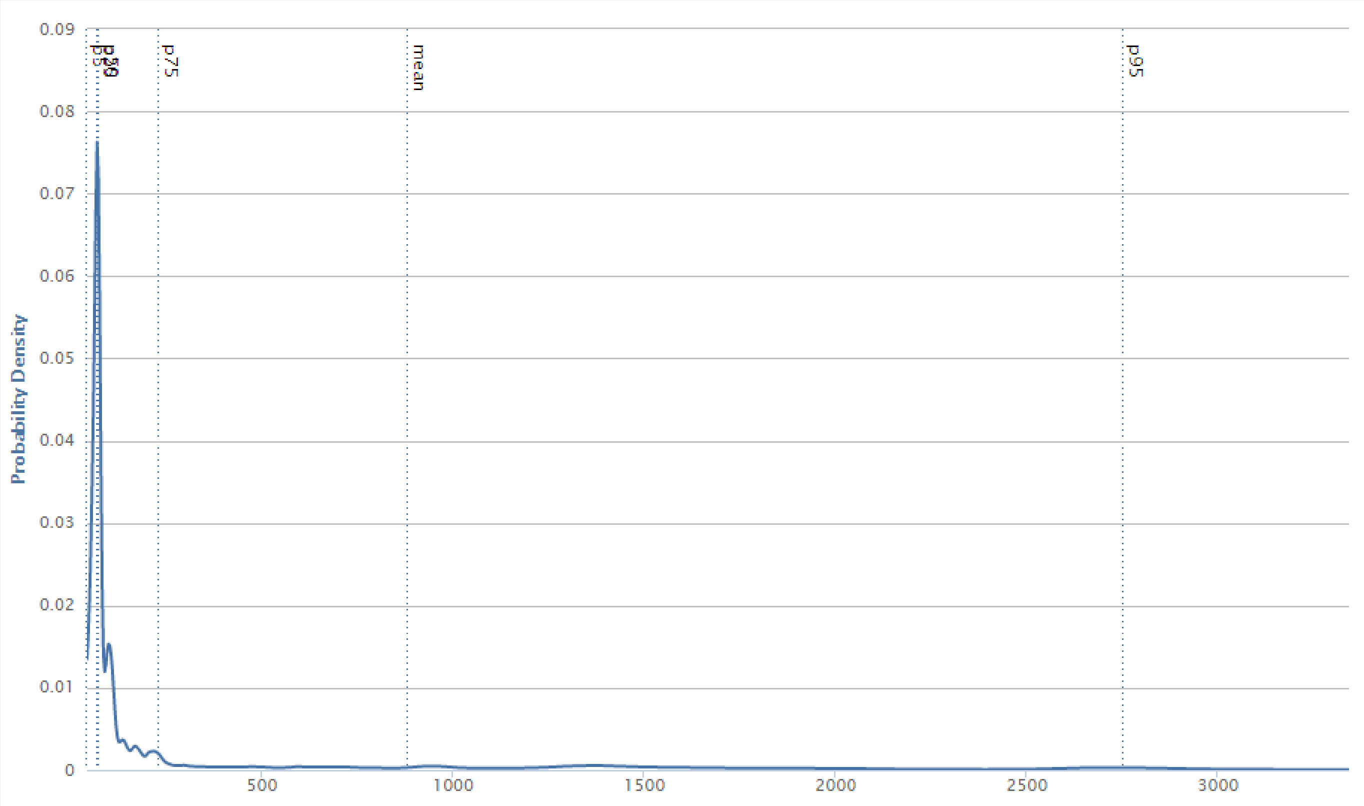
Common Workload Pattern

Key Length Distribution



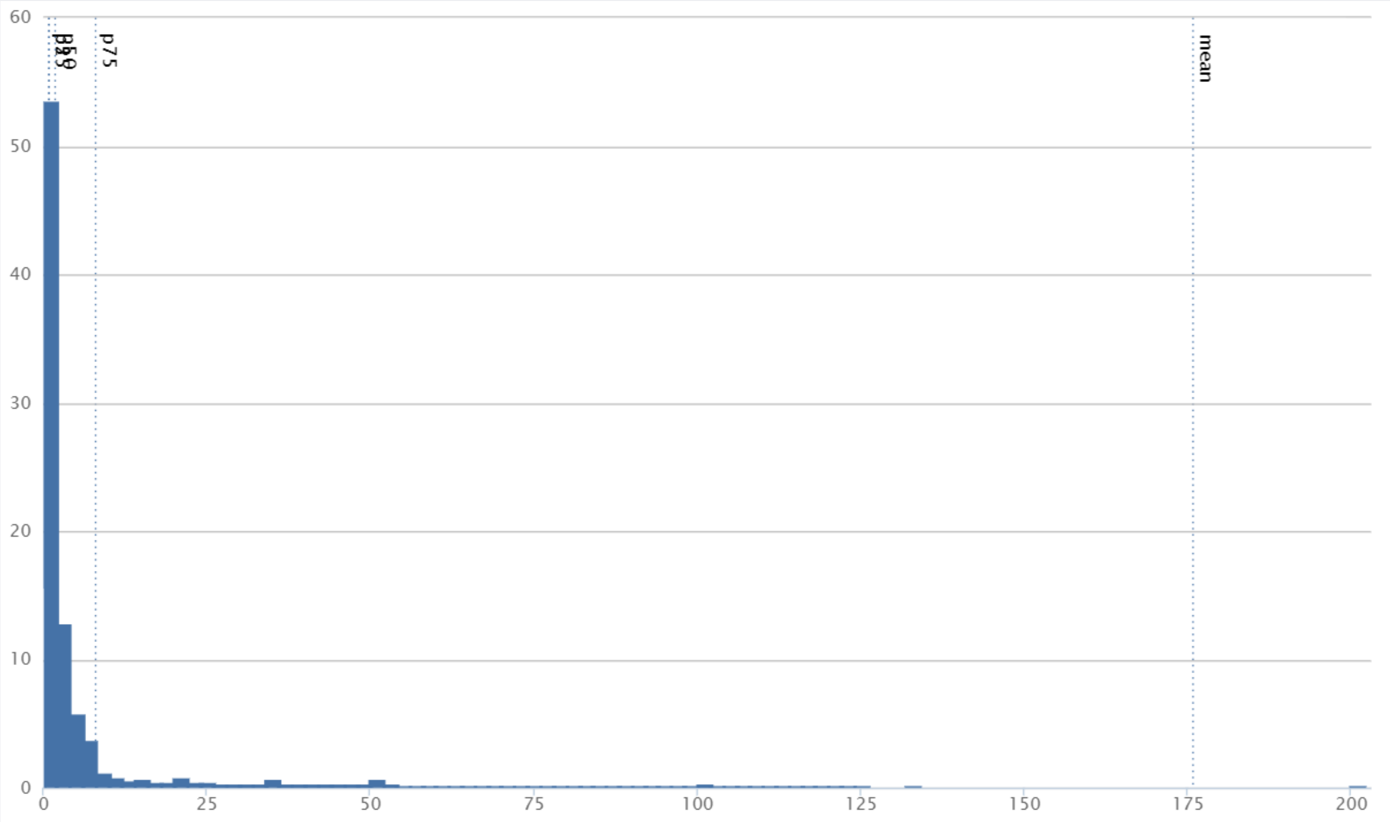
Key + Value Length Distribution

	K+V length (Bytes)
Mean	869
Median	59
P75	216
P90	1.68K
P95	2.75K
P99	12.2K



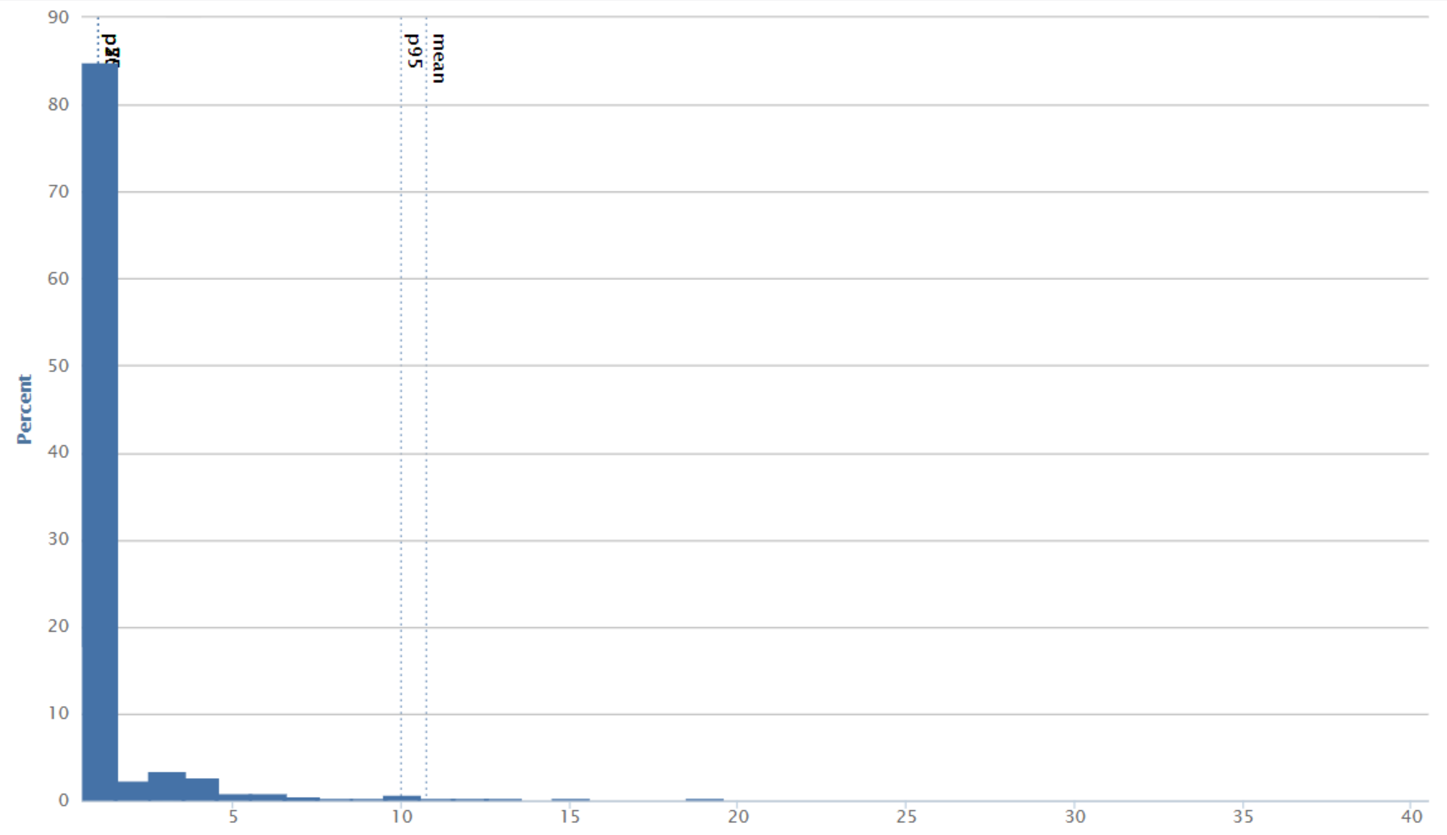
Number of Keys Per Range Query

	#keys updated per write
Mean	176
Median	2
P75	8
P95	732
P99	3.67K



Number of Keys Updated Per Writex

	#keys updated per write
Mean	10.8
Median	1
P95	10
P99	250



Take-Away

- RocksDB is versatile:
  - Diversified Applications
  - Diversified Workloads

Thank You!

RocksDB

Key-Value Storage on Log-Structure Merge-Tree