Enhancements to RocksDB

Supporting a 1PB In-Memory Workload

Haobo Xu
Database Engineering@Facebook

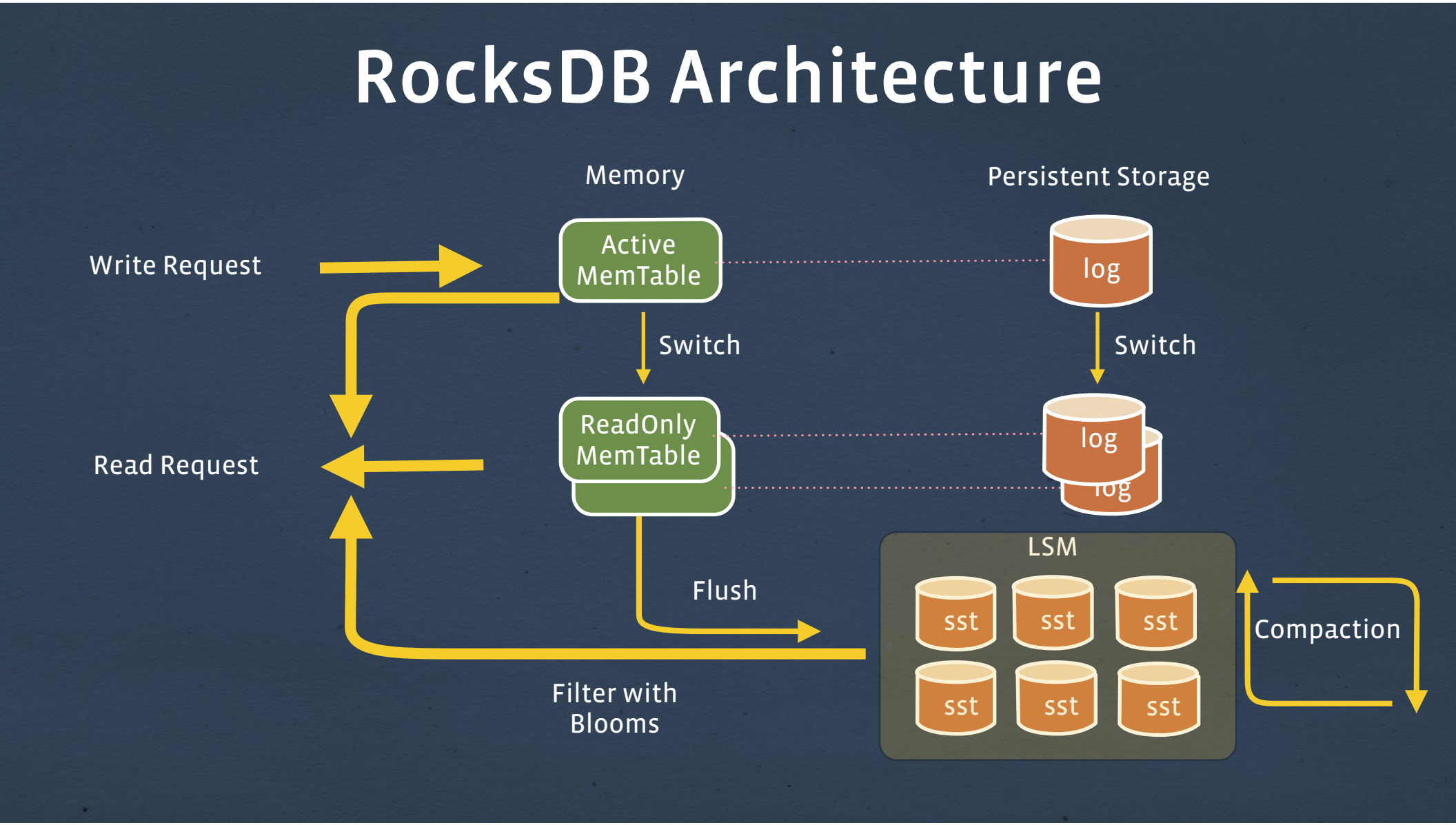
Agenda

- Quick intro to RocksDB
- A case study: What it takes to make RocksDB work for in-memory workload
- Take away

RocksDB API

- Keys and values are arbitrary byte arrays.
- Data are stored sorted by key. Client tells us how to sort via comparator.
- Update Operations: Put/Delete/Merge
- Queries: Get/Iterator
- Embedded Library

RocksDB Architecture



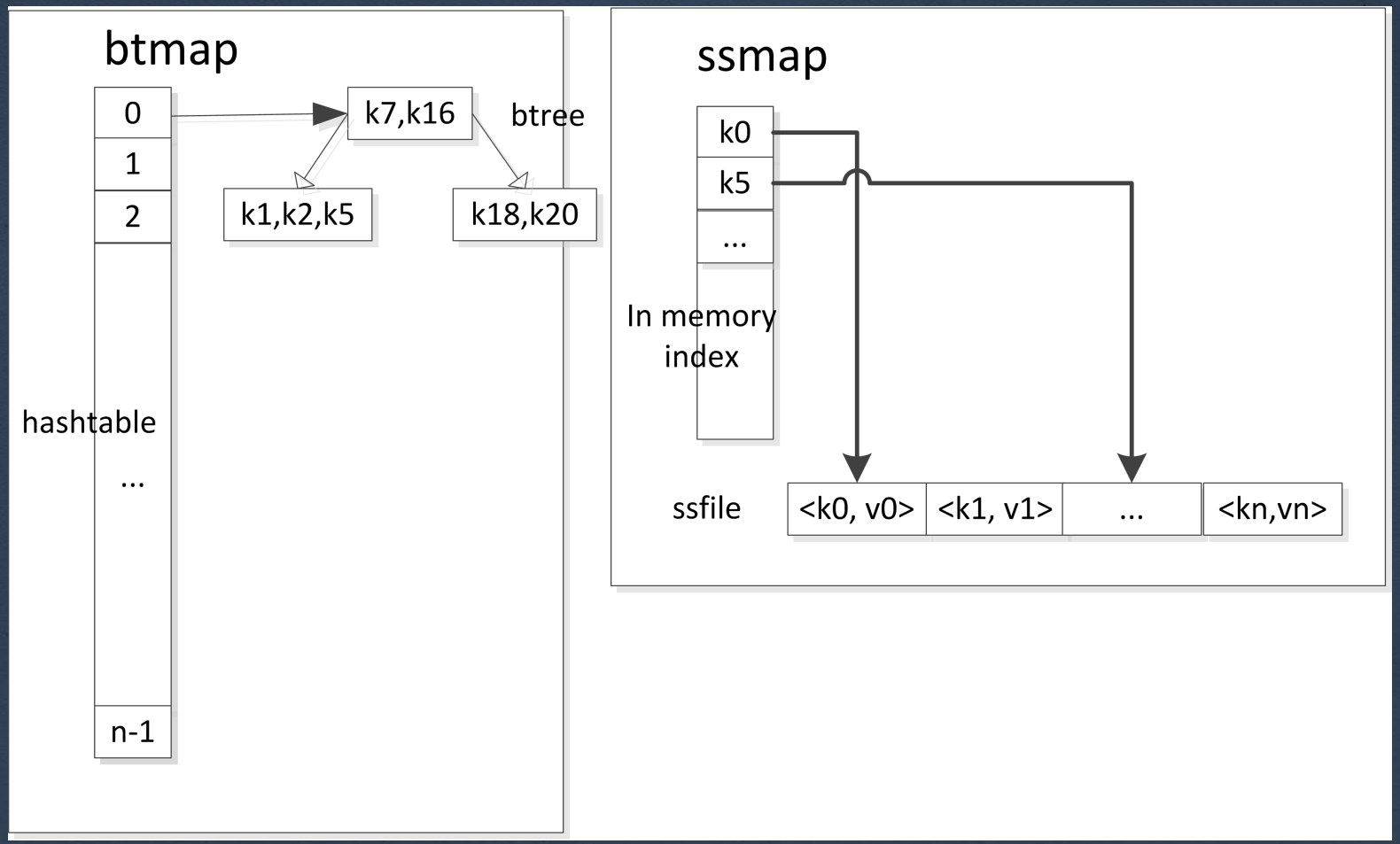
RocksDB -- The Trade Off

- Three Amplifications
  - Write Amplification (WAF)
  - Read Amplification (RAF)
  - Space Amplification (SAF)
- Compaction is the tuning knob
  - Adds to WAF
  - Reduces RAF
  - Reduces SAF
- Find the right balance for your workload is the key to success

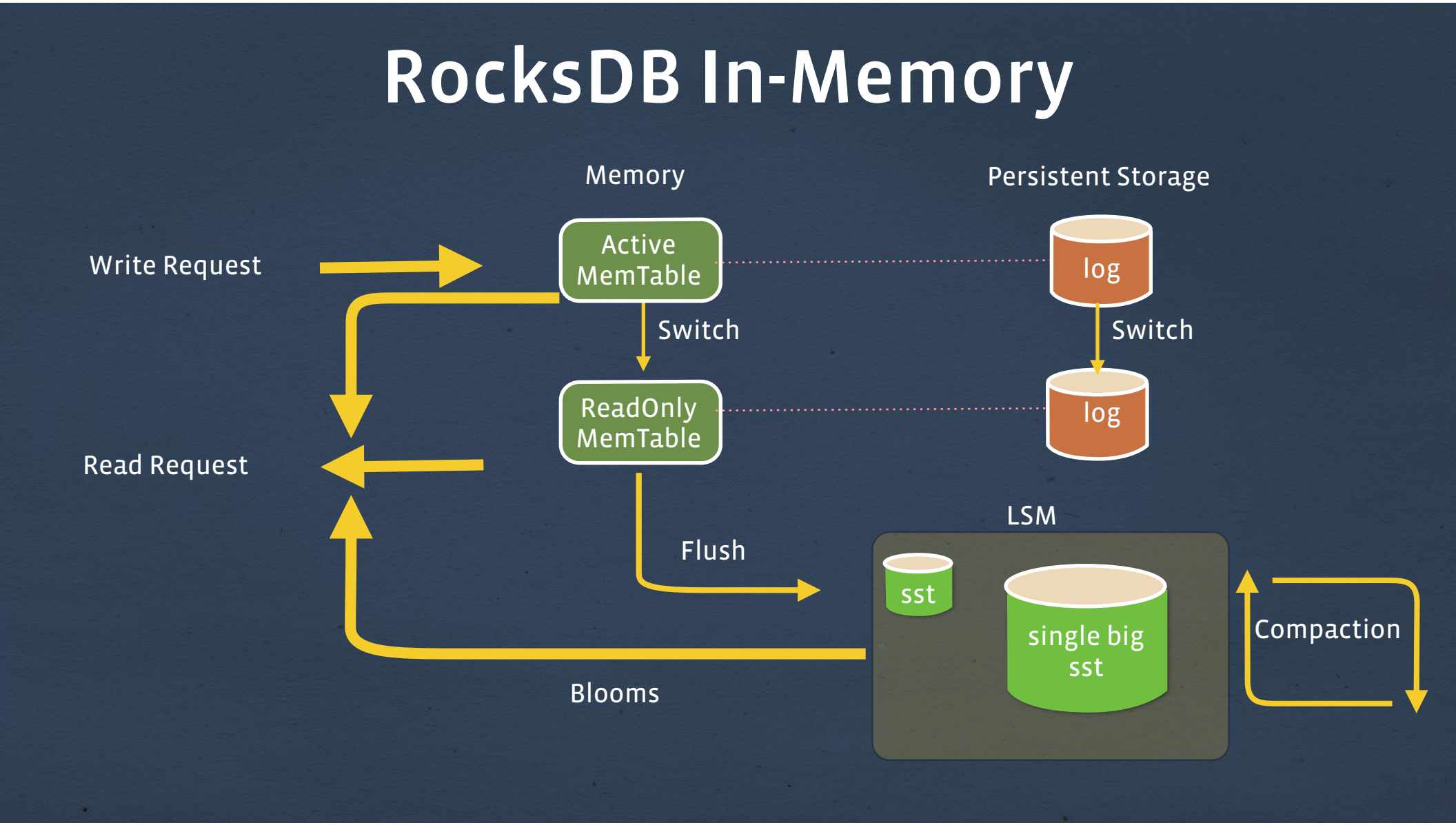
The New Challenge - In Memory

- Existing application service
  - Lots of servers, 1PB total RAM
  - Extreme low latency
- Existing in-memory key/value storage solution
  - very efficient
  - tightly coupled with application
  - no transaction log
- Can RocksDB help?

The Existing Solution



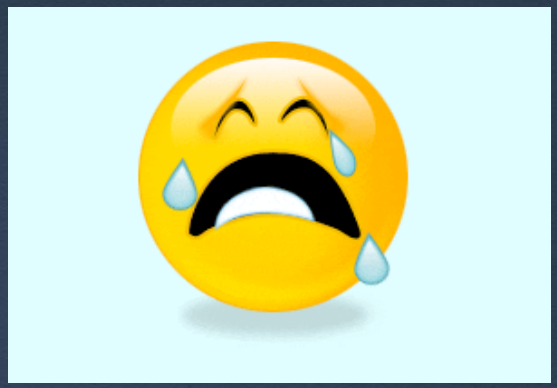
RocksDB In-Memory

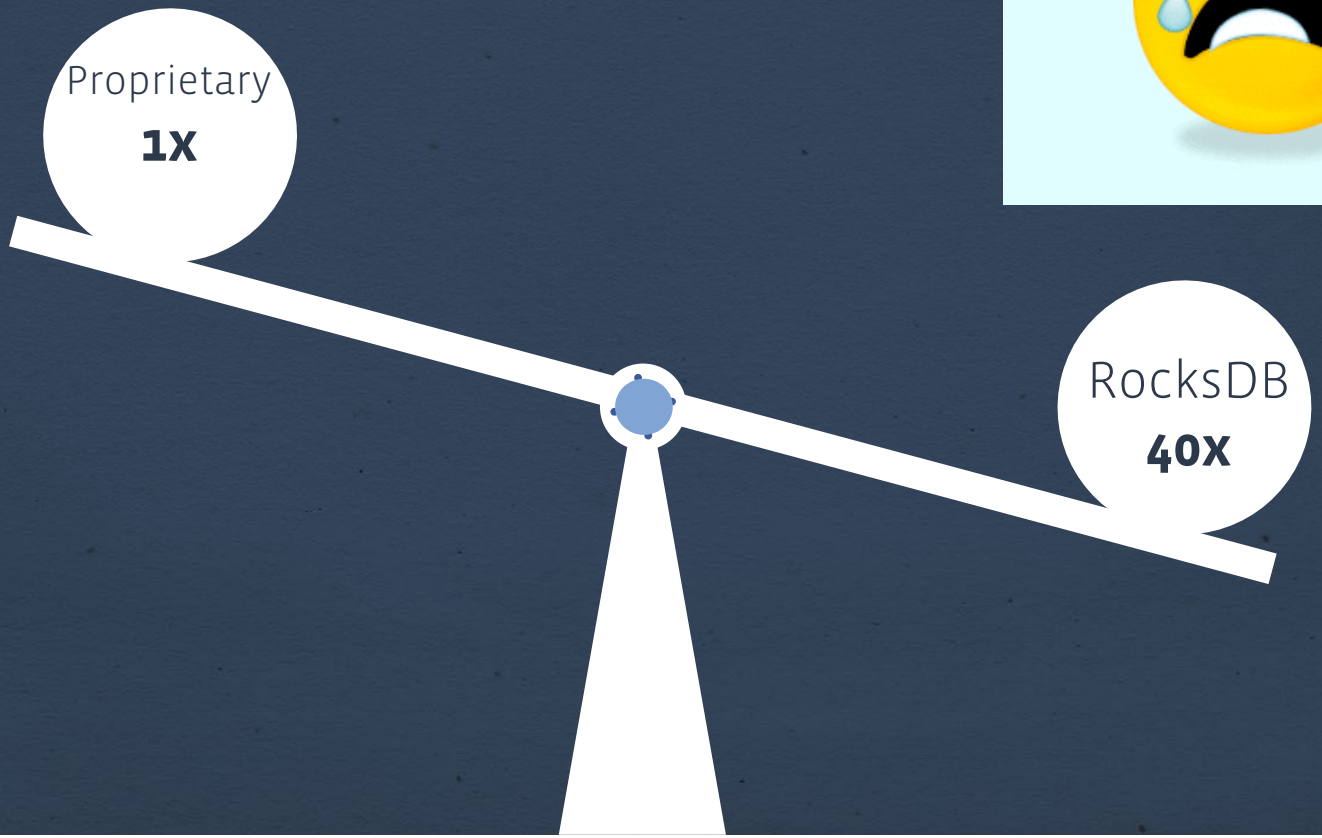


The Trade Off In-Memory

- Characteristic of Memory, compared to traditional storage
  - Low Latency
  - High Throughput
  - Limited space
- Was it a good trade?
  - Minimized Read Amplification
  - Minimized Space Amplification
  - Allowed Aggressive Compaction and Big Write Amplification

First Try



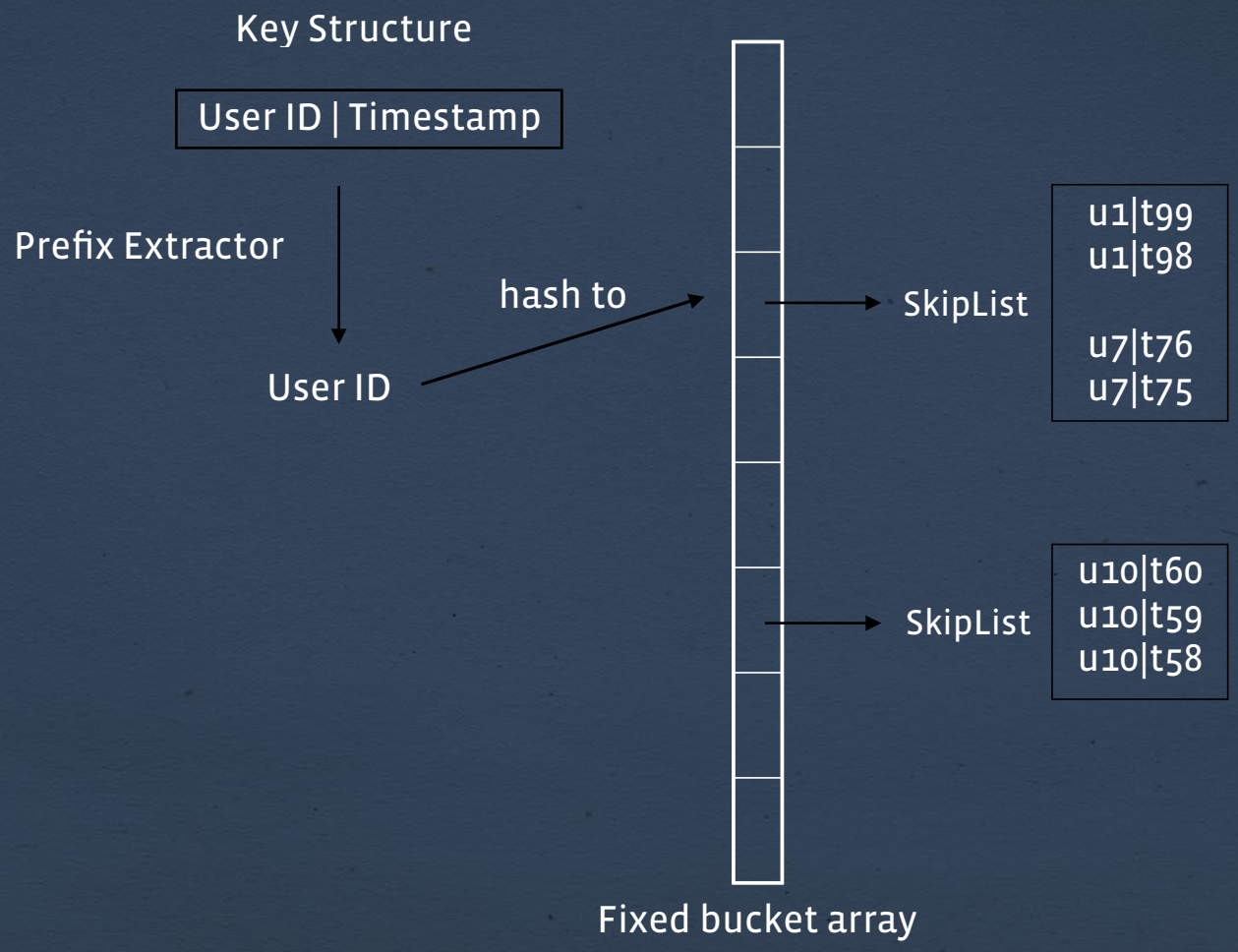


Know your workload

- We are storing user action history
  - user id | timestamp => action
  - Write: Bob Liked Page ABC at 4:30pm yesterday
  - Read: Bob's activities since yesterday

- Query pattern does not impose total ordering across user ids
- Profiling result agrees

Prefix Hashed SkipList Memtable

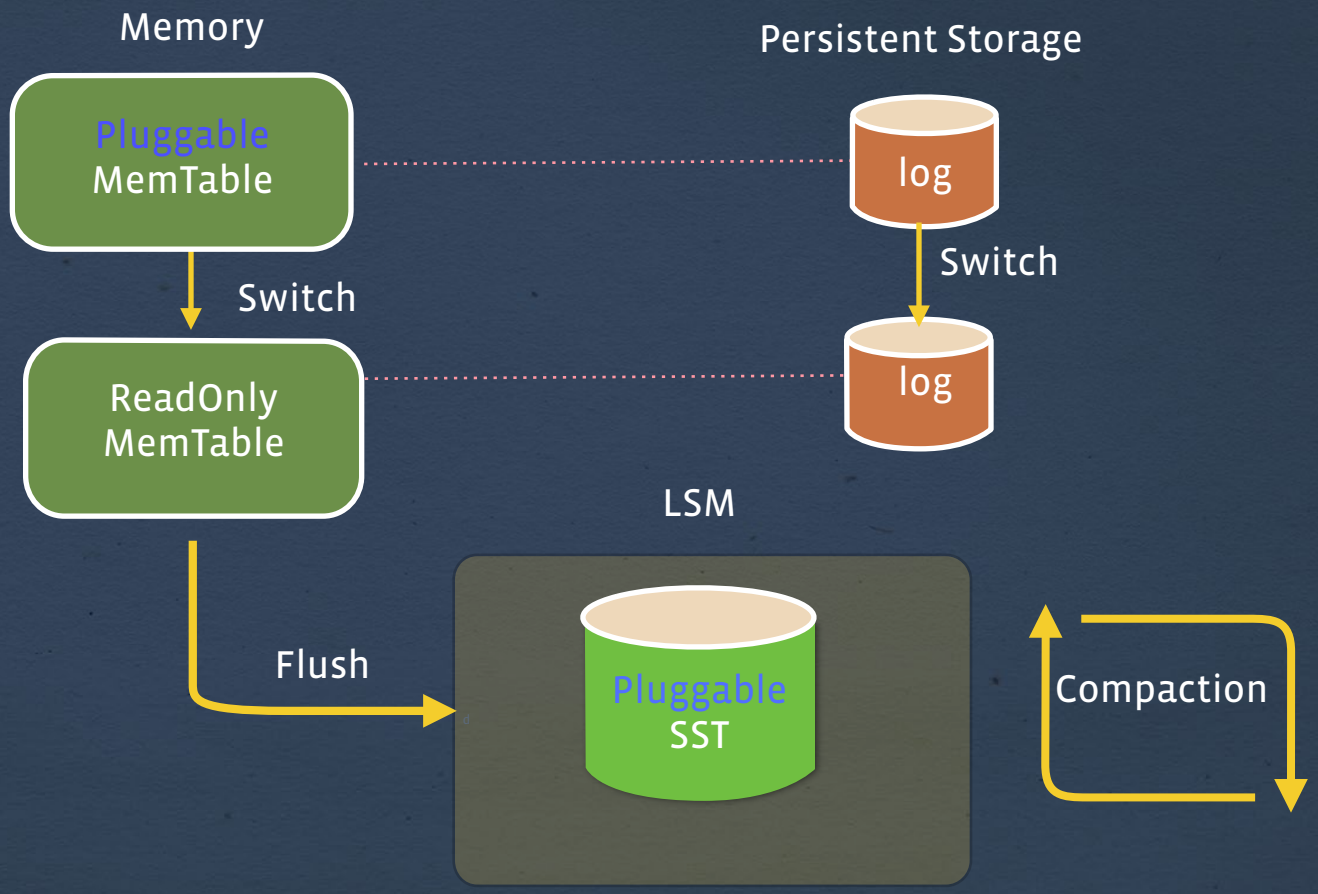


Byte Addressable Plain SST

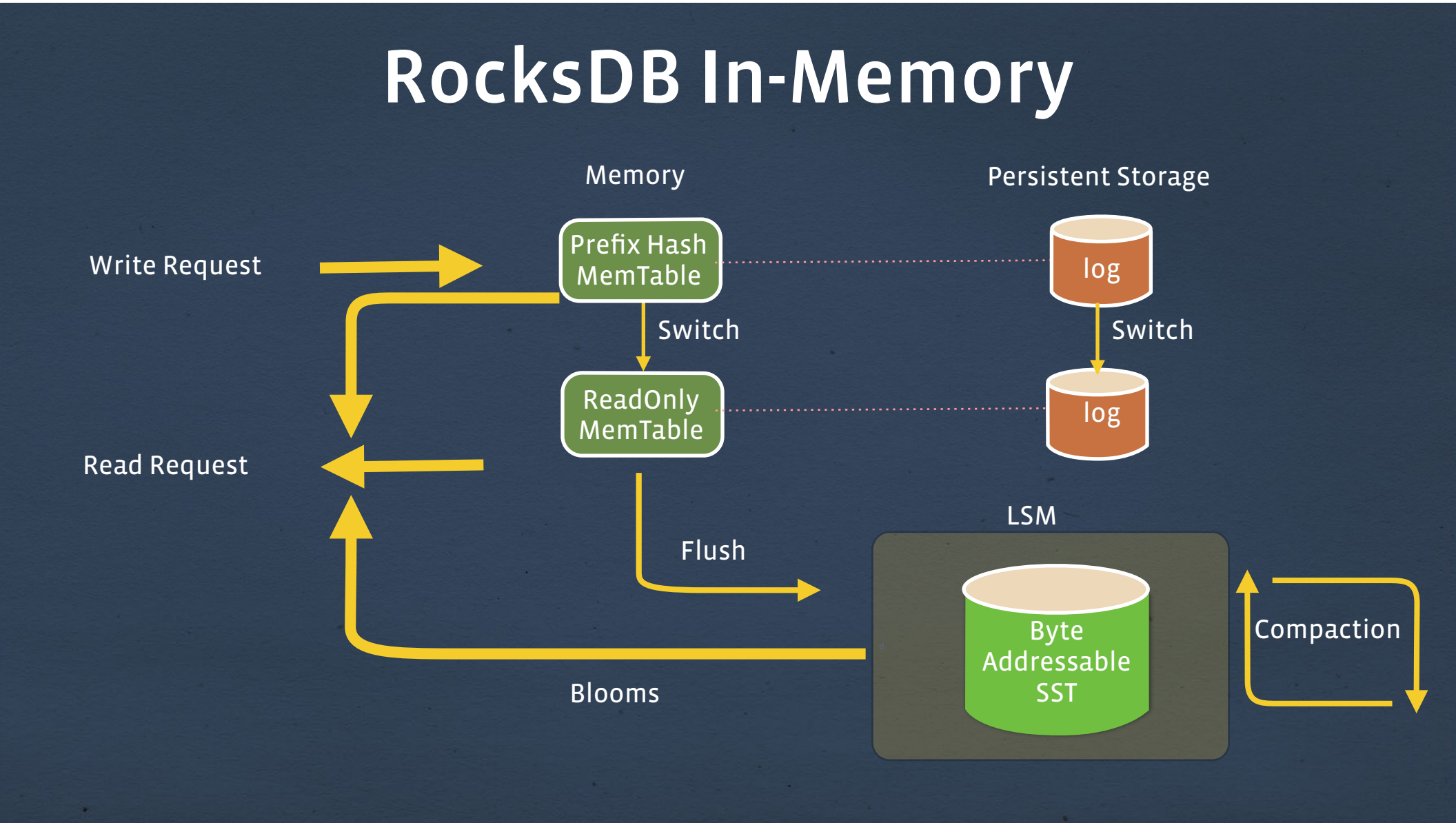
- Existing RocksDB SST optimized for block based storage
  - Files are partitioned to fix-sized blocks
  - Use block cache to reduce slow block transfer from device
  - Irrelevant in RAM

- Solution: A much simpler format that just stores sorted key/value pairs sequentially
  - no blocks, no caching
  - build efficient lookup index on load (prefix hash + binary search)

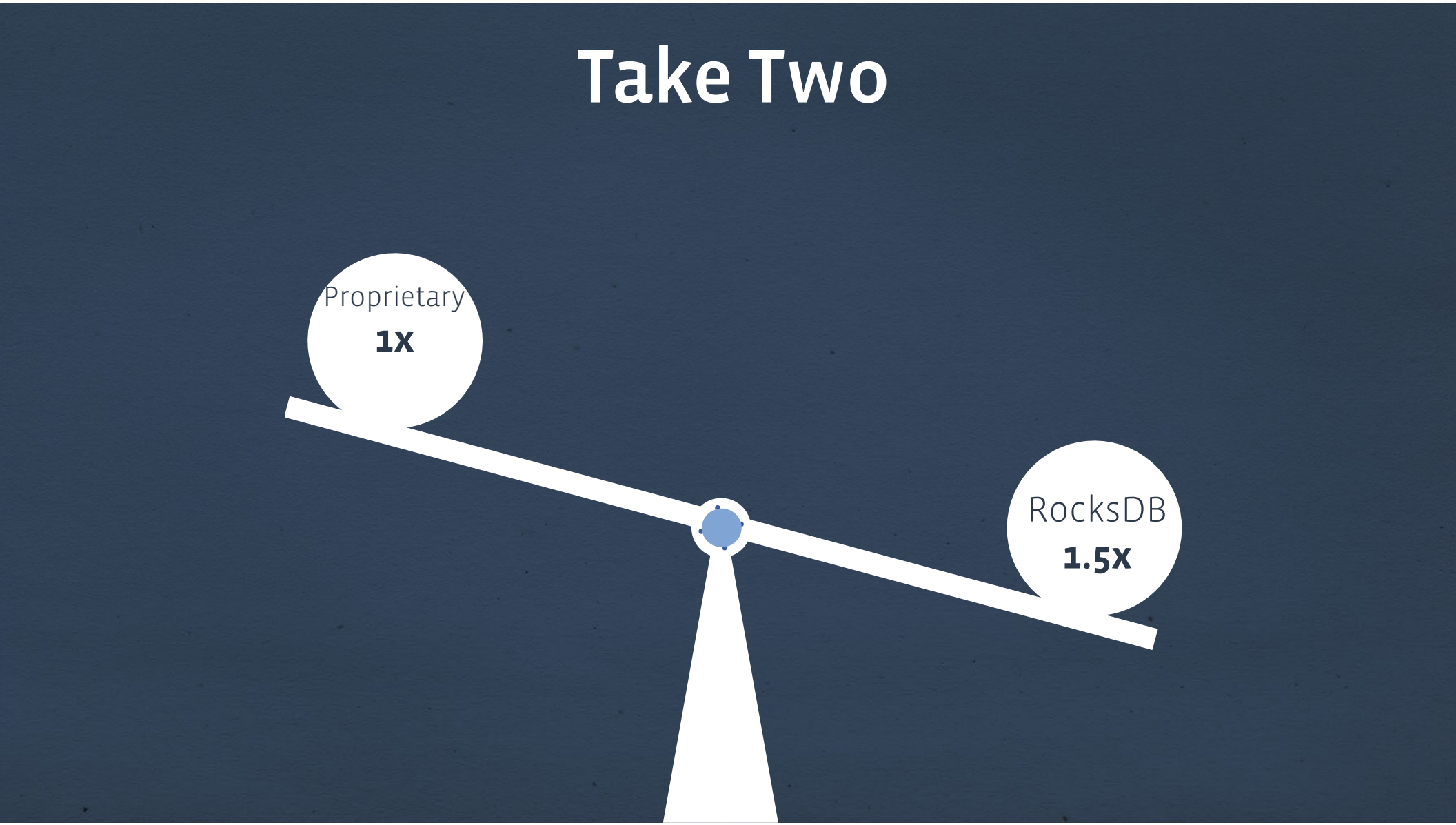
RocksDB Open & Pluggable



RocksDB In-Memory



Take Two

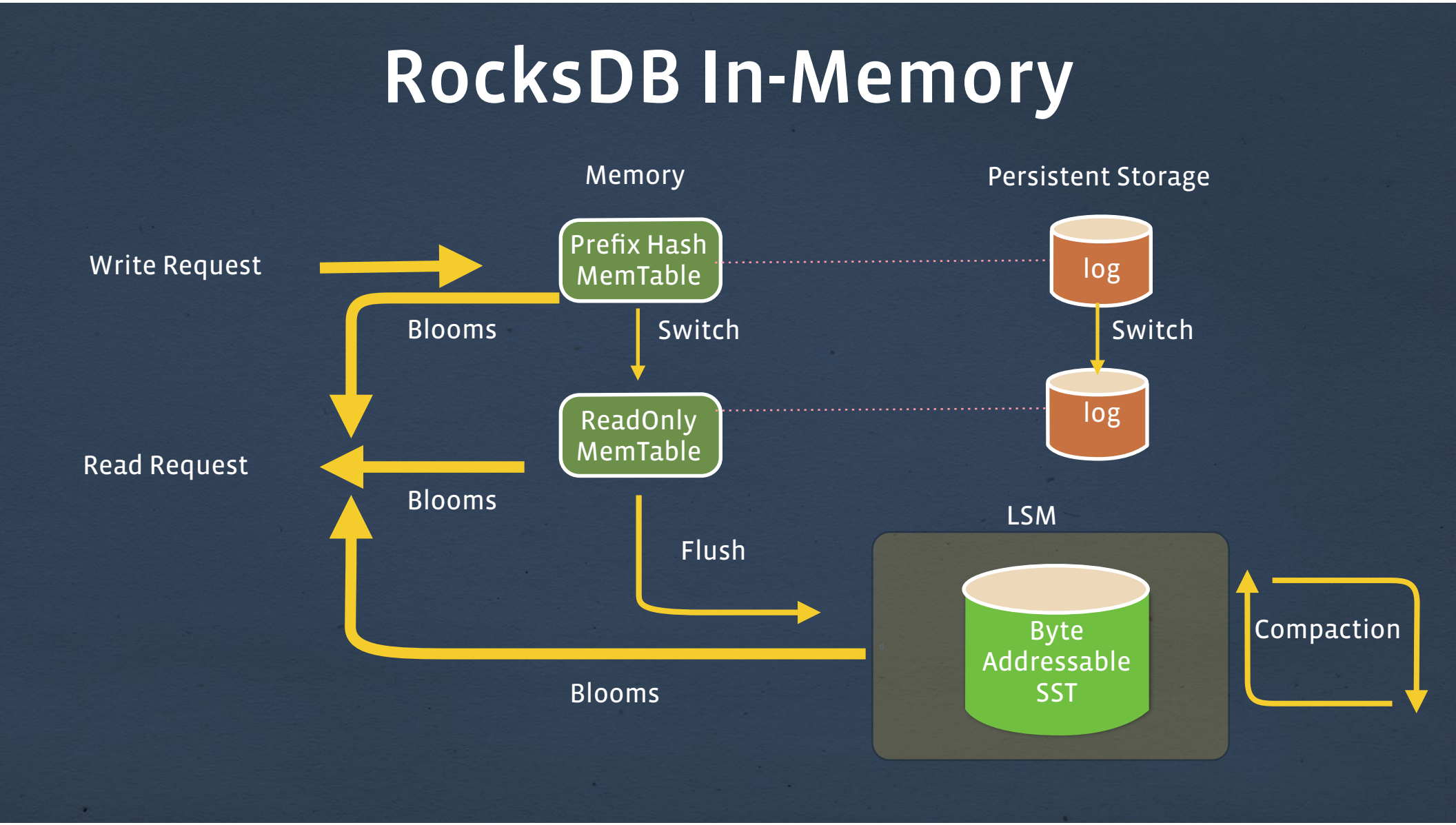


Further Optimize Not Found

- Only 20% of queries return any data.
- Less than 2% of queries hit anything in the memtable
- Problem: Get needs to go through the memtable lookups that eventually yield nothing.

- Solution: Add a bloom filter to memtable!

RocksDB In-Memory



Current State

Proper Recovery

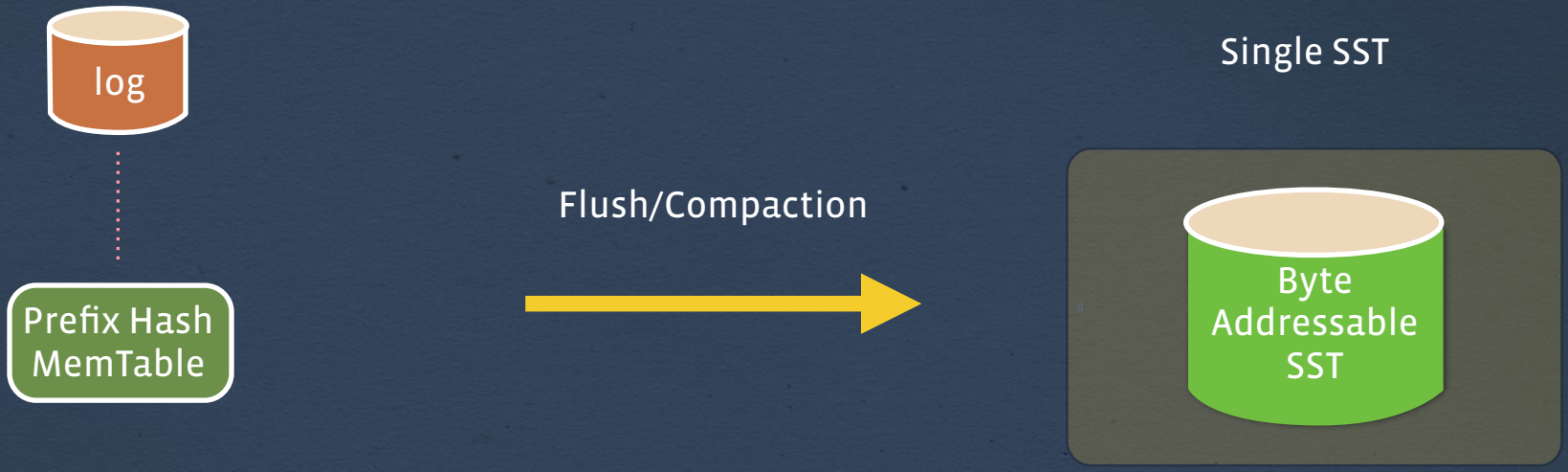
Proprietary
1X

RocksDB
1.1X

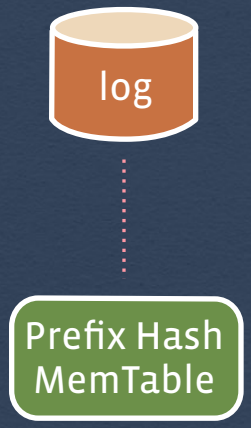
Clean API

Performance

Take Away

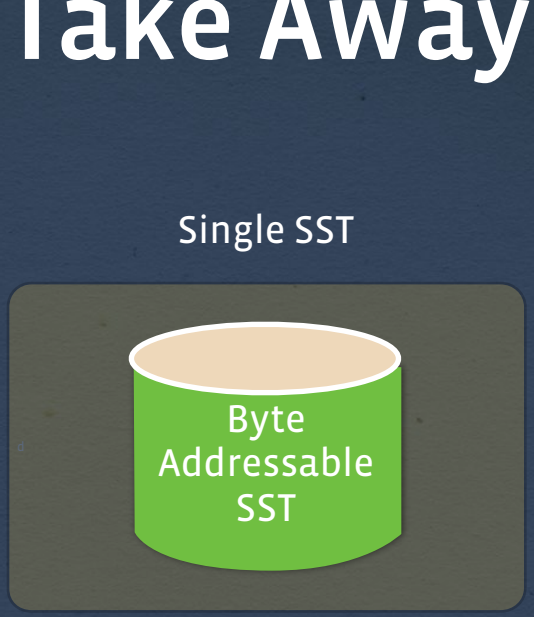

log
Prefix Hash MemTable
Flush/Compaction
Single SST
Byte Addressable SST

Take Away



- Concurrent Read/Write
- Be fast
- Be space efficient
- Is it possible?

Take Away



- Readonly, concurrency friendly
- key/value sorted and packed
- efficient lookup index on top
- Perfect data structure for efficient lookup/scan

Take Away

- Concurrent Read/Write
- Be fast
- Be space efficient
- Is it possible?

Flush/Compaction

- One time investment
- Be careful about investment expiration!

Single SST

- Readonly, concurrency friendly
- key/value sorted and packed
- efficient lookup index on top
- Perfect data structure for efficient lookup/scan