SDC 19
September 23-26, 2019
Santa Clara, CA

LSM-based Storage Techniques - Strengths and Trade-offs

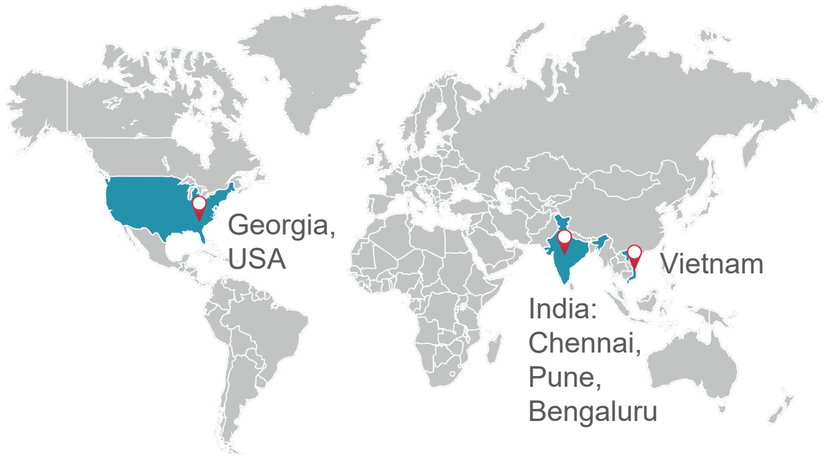
Rohan Puri, Shriram Pore
MSys Technologies, LLC

About MSys Technologies

Our WW Strength

800

And growing



Product Engineering Services

- Storage and Networking Engineering
- VMware Ecosystem Integration
- UX/UI, Enterprise Mobility
- Rapid Application Development
- AI, ML and Cognitive Services
- Fintech and Loyalty
- Contingent Hiring

Technology CoEs

- Storage CoE
- DevOps CoE
- QA Automation CoE
- Big data and Predictive Analytics CoE
- Digital Testing CoE
- Cloud CoE
- Open Source CoE

Outsourcing Partners to

Hewlett Packard Enterprise, CHEF, rubrik, Google, IBM

Key Alliances

CLOUD NATIVE COMPUTING FOUNDATION, amazon web services, kubernetes CERTIFIED SERVICE PROVIDER, docker, CHEF, IBM Watson

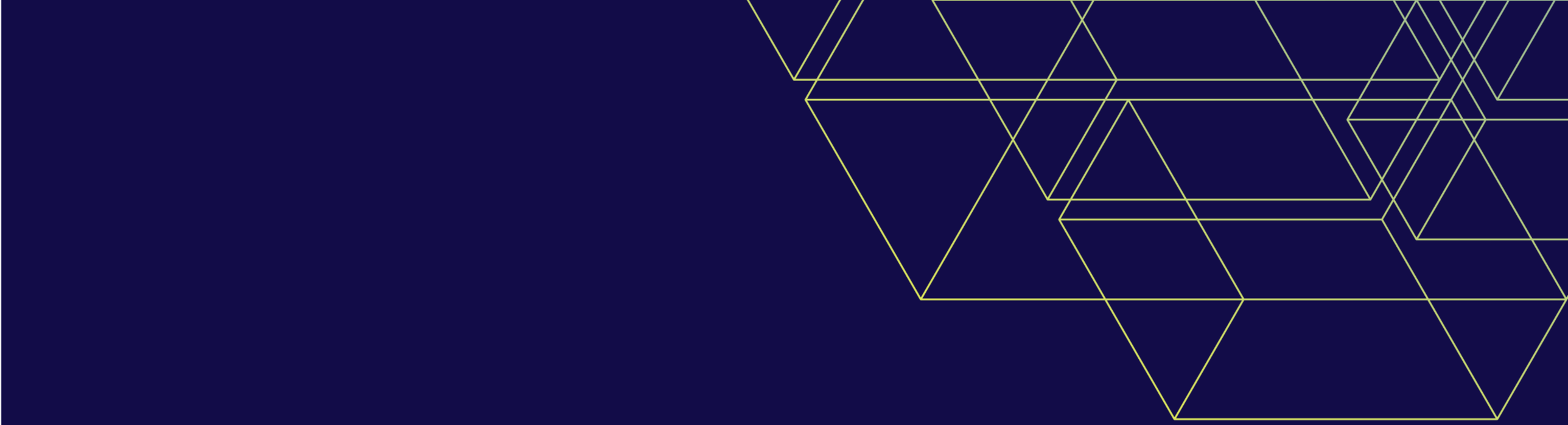
Agenda

- LSM-Tree Introduction
- Rum Conjecture
- LSM-Tree Basics
- LSM-Tree Improvements
- LSM-based systems
- Summary

Introduction

Introduction

- Used as storage layers of NoSQL DBs
  - Hbase, Cassandra, LevelDB, RocksDB, AsterixDB
- Writes
  - Buffer, Flush, Merge
- Advantages
  - Write performance
  - Space utilization
  - Tunability
  - Simplifies concurrency control & Recovery



Rum Conjecture

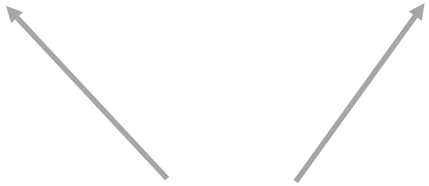
Rum Conjecture

Read Optimized

(Point/Tree Indexes)

Update optimized

(Differential)

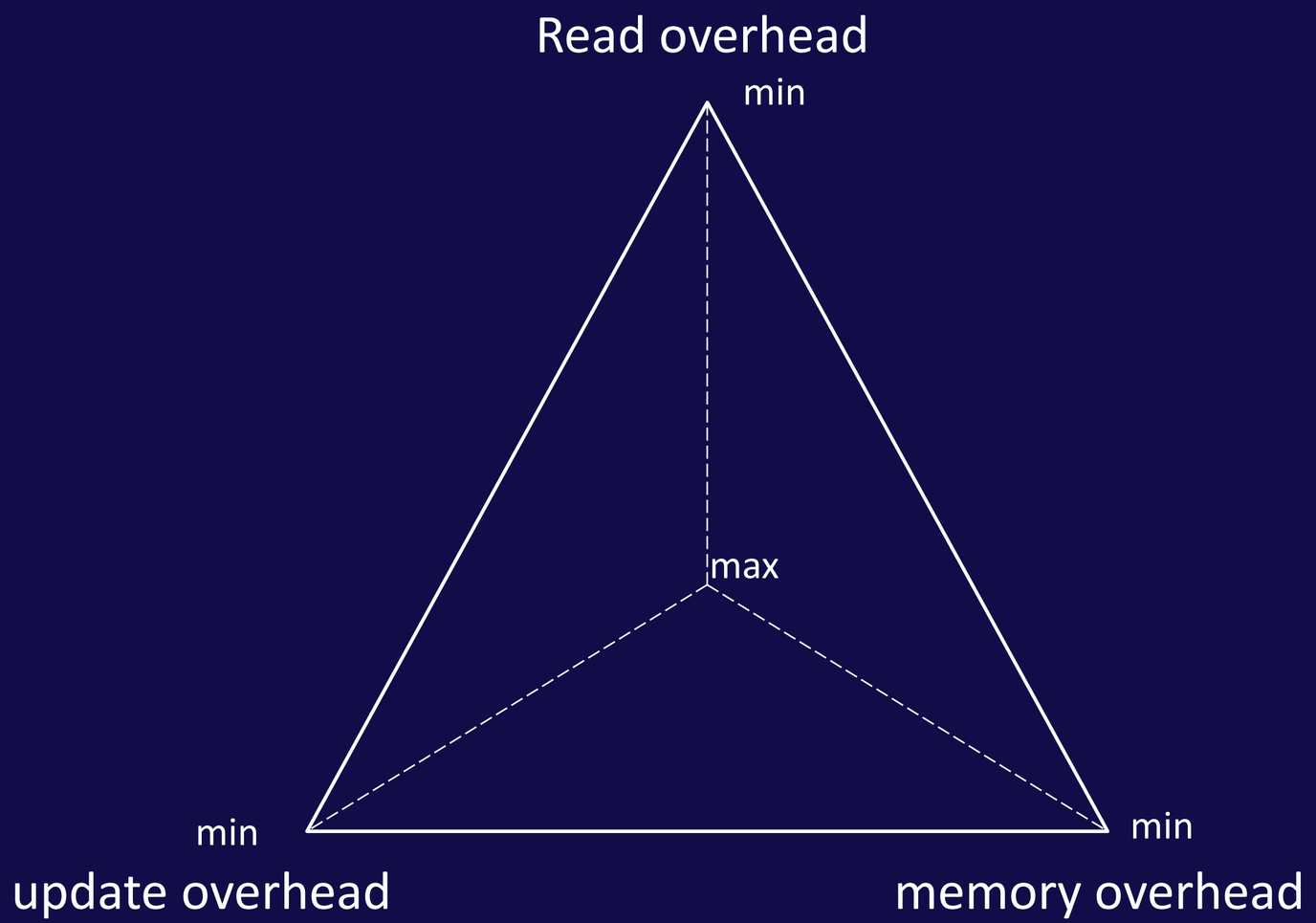


Access Method

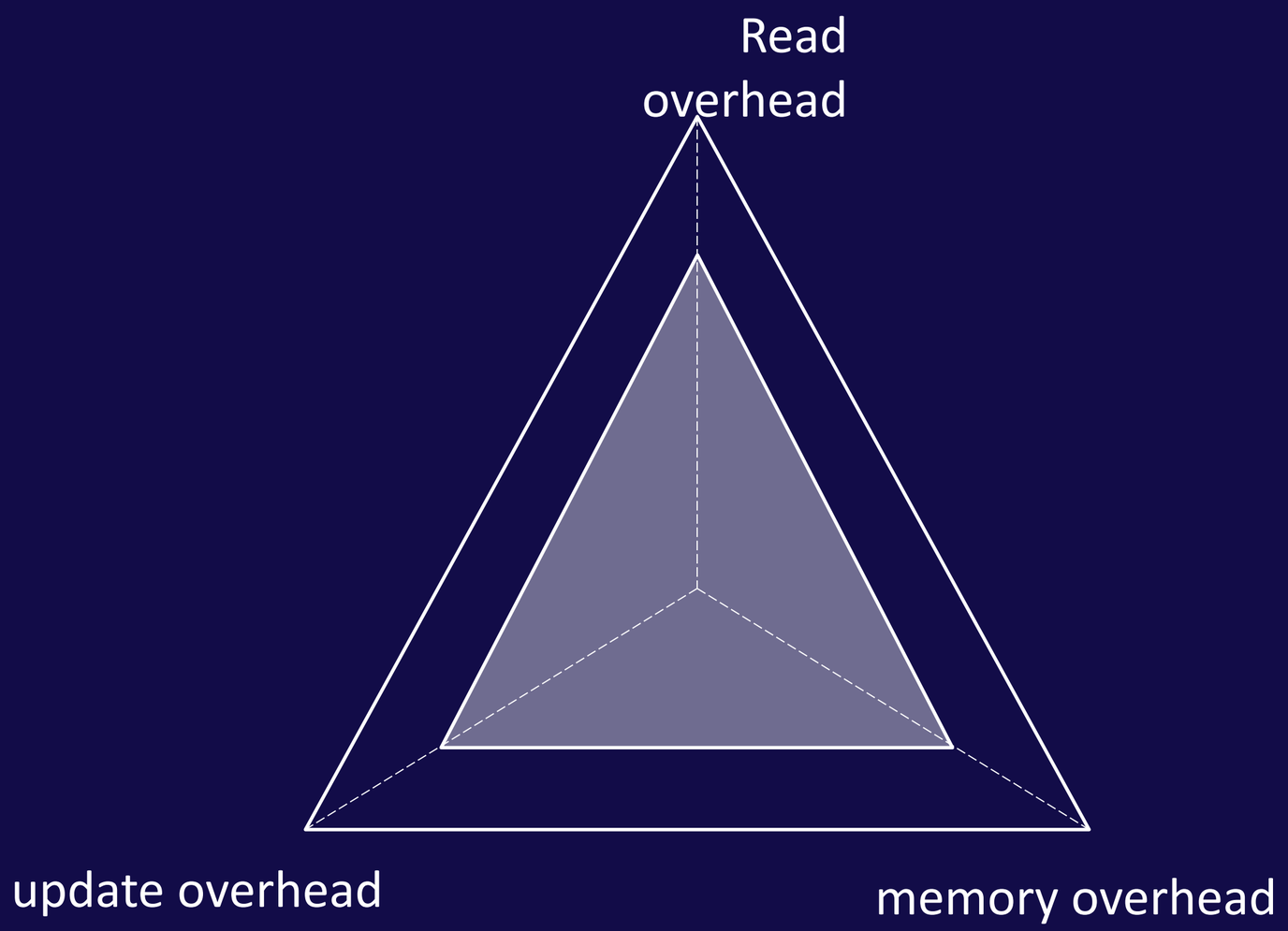
Memory optimized

(Compressible/Approximate)

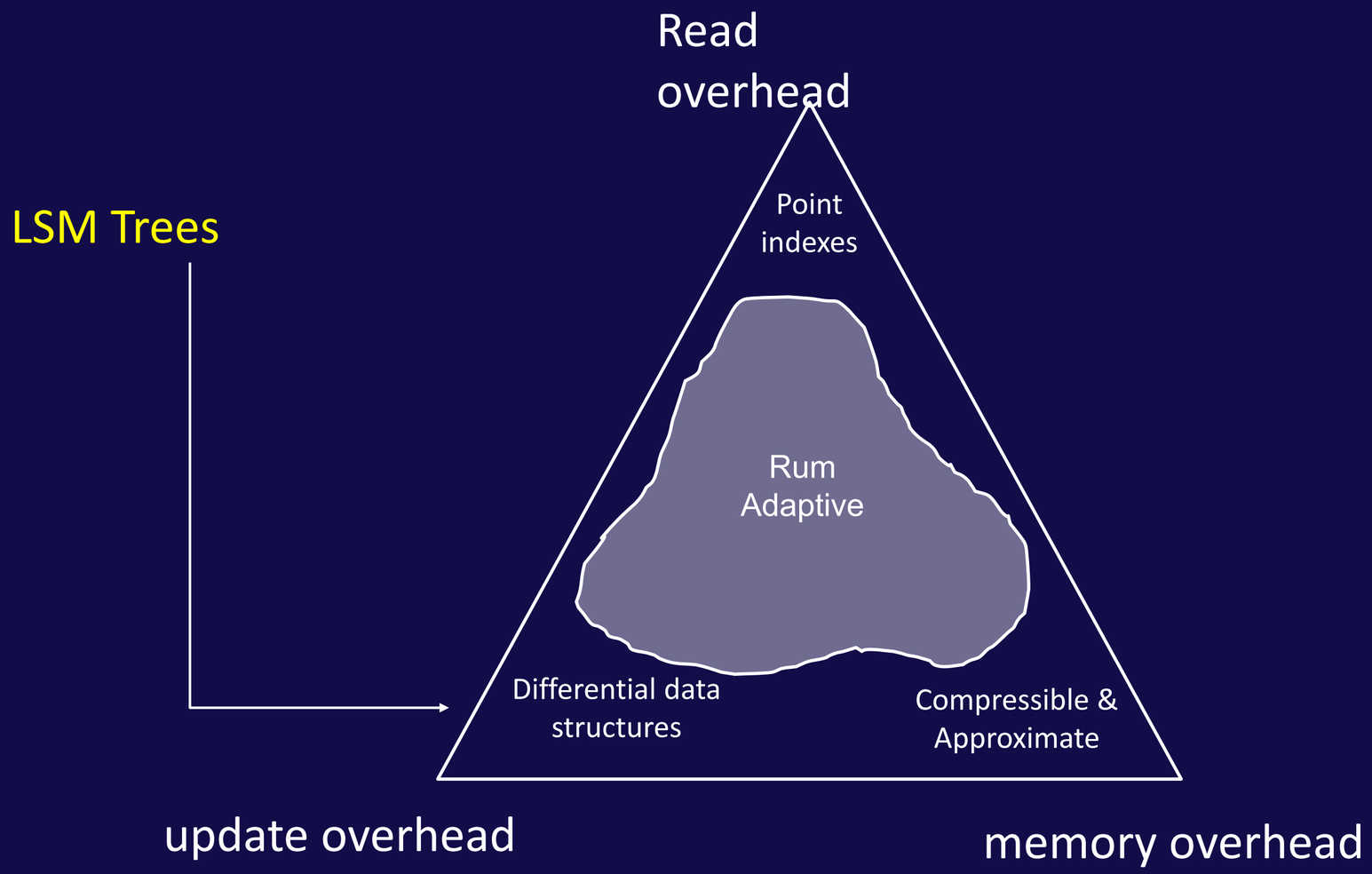
RUM Space

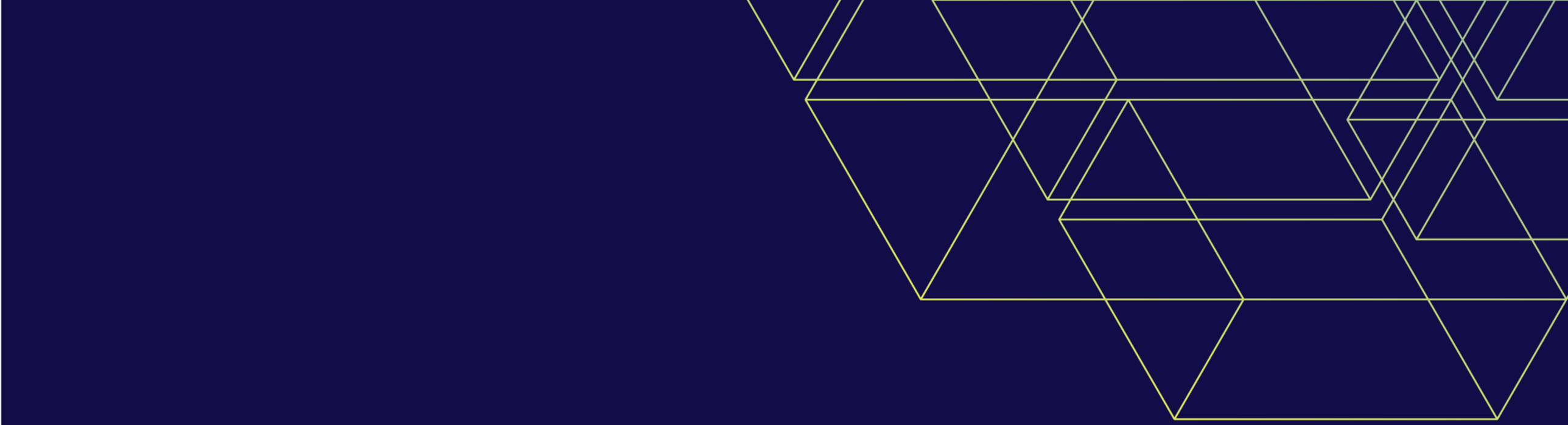


Ideal Solution



RUM Space





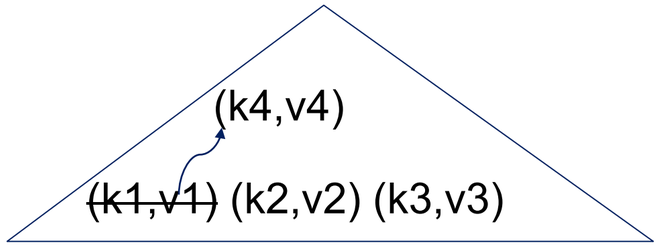
LSM Tree Basics

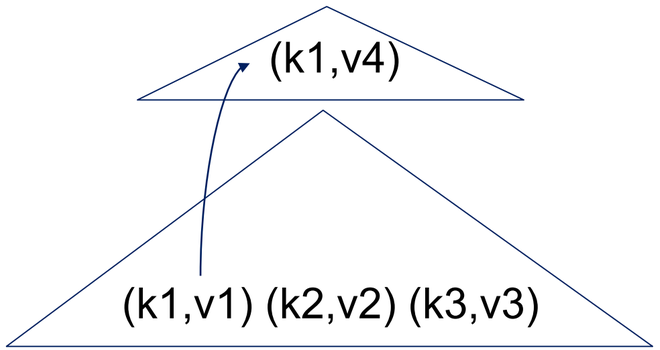
LSM Tree Basics

- History
- Today's LSM Trees
- Well-known optimizations
- Concurrency control & recovery
- Cost analysis

History

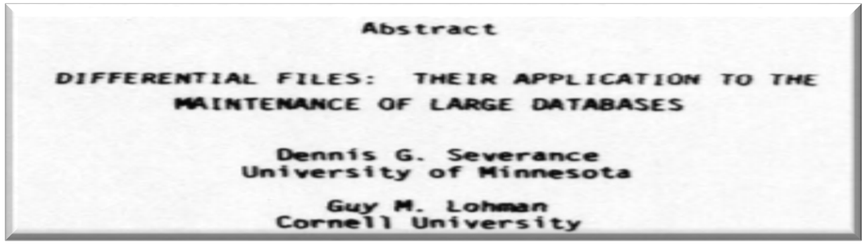
In-place Update	Out-of-place Update
Overwrite	Write to new locations
Improves read	Improves write
Only one copy of data	Multiple copies of data
Recovery complicated	Recovery simplified
Space amplification high	Space amplification low
No gc	Need gc





History

- Sequential out of place update mechanism is not new
- Differential files [1976]
  - Updates applied to diff file
  - Periodic merges with main file



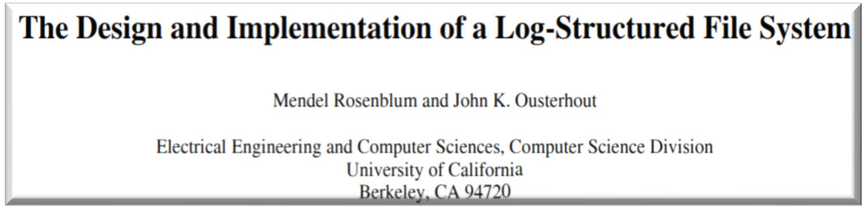
Abstract

DIFFERENTIAL FILES: THEIR APPLICATION TO THE MAINTENANCE OF LARGE DATABASES

Dennis G. Severance
University of Minnesota

Guy M. Lohman
Cornell University

- Postgres log-structured db [1980s]
  - Append writes -> sequential log
    - Achieve fast recovery, time-travel queries
- Log-structured File systems [1991]



The Design and Implementation of a Log-Structured File System

Mendel Rosenblum and John K. Ousterhout

Electrical Engineering and Computer Sciences, Computer Science Division
University of California
Berkeley, CA 94720

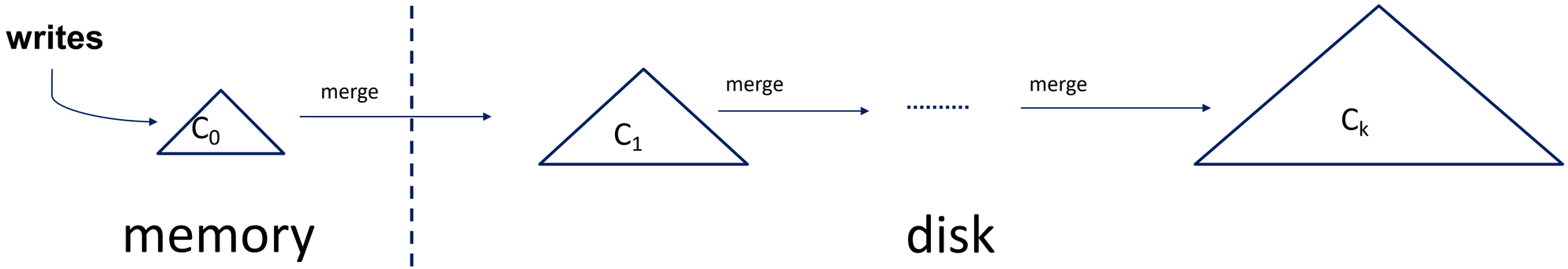
History

- Problems
  - Low query perf, since related logs scattered
  - Low space utilization
  - Hard to tune
- LSM-tree [1996]
  - Includes merge in structure itself, to address above issues
  - High write performance
  - Bounded query performance
  - Bounded space utilization


The Log-Structured Merge-Tree (LSM-Tree)

Patrick O'Neil[1], Edward Cheng[2]
Dieter Gawlick[3], Elizabeth O'Neil[1]
To be published: Acta Informatica

Original Design

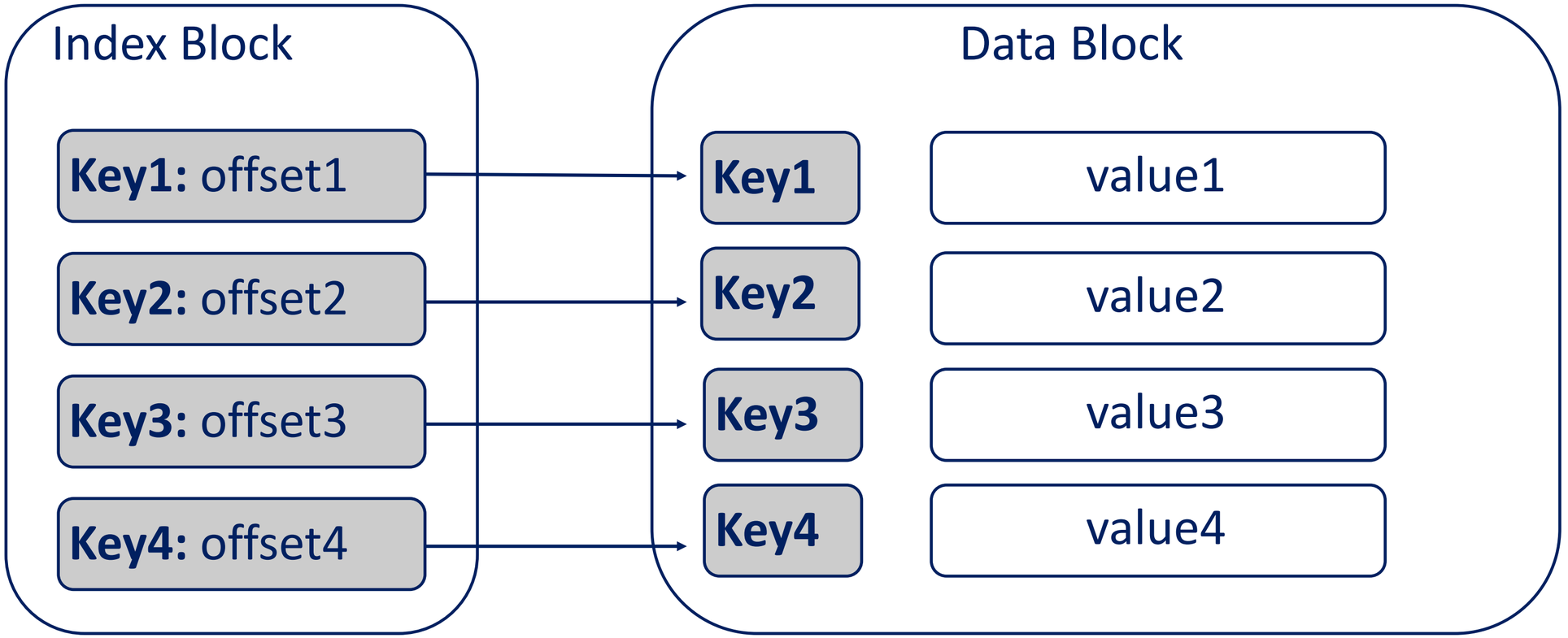


- Sequence of components C_0 to C_k
- Each component B+-tree
- C_i full -> rolling merge -> C_i to C_{i+1}
- Known as leveling merge policy
- Size ratio $T_i = C_i+1 / C_i$
- All T_is same -> optimizes write perf

Today's LSM

- Updates are out of place
- Writes - append to memory component
- Insert/update operations adds new entry
- Deletion adds anti-matter entry
- Exploit immutability of disk components(runs) for concurrency & recovery
- Multiple disk components merged into new one, without modifying existing ones. [Different than rolling merge]
- Component can be implemented using any index structure
- Memory Component -> B+tree or skip-list
- Disk Component -> B+tree or SSTable (sorted string table)

SSTable

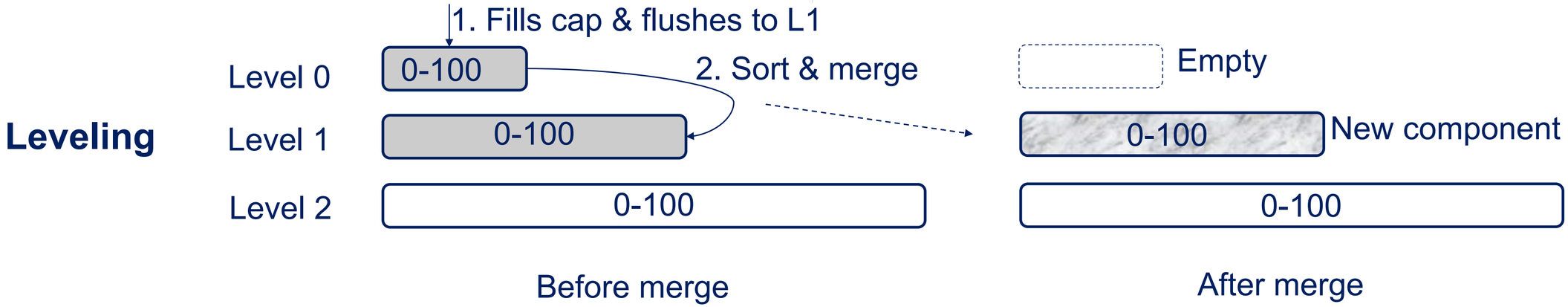


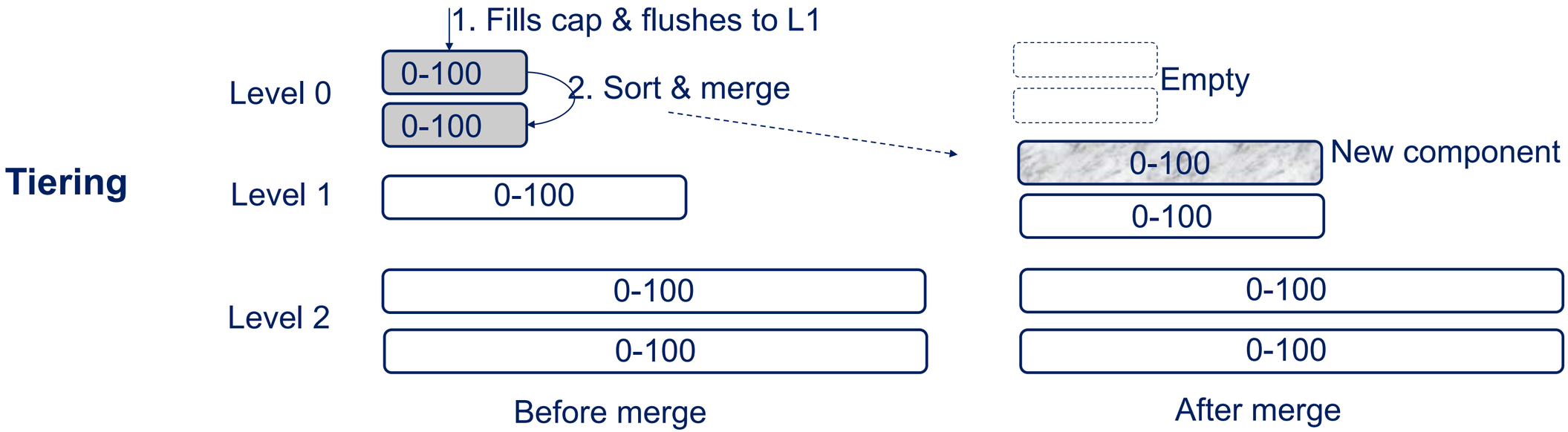
https://medium.com/databasss/on-disk-io-part-3-lsm-trees-8b2da218496f

Query

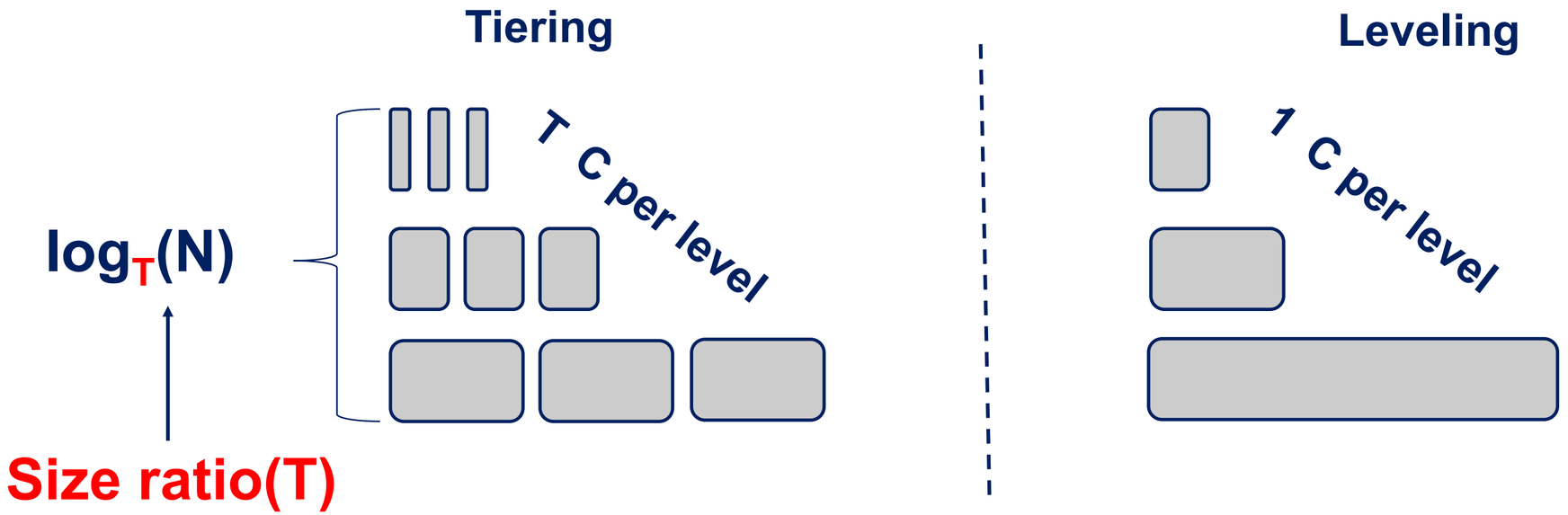
- Search multiple components to perform reconciliation
- Point lookup query
  - Search from newest to oldest component
  - Stop after first match found
- Range query
  - Search all components at same time
  - Feed search results to priority queue
  - Priority queue does reconciliation
- Query Perf ↓ when disk components ↑
- ↓ disk components by gradual merge

Merge



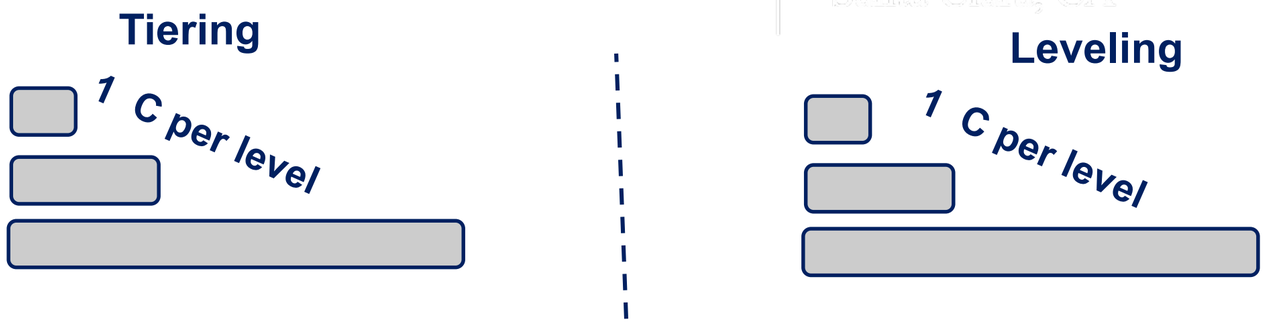


Merge

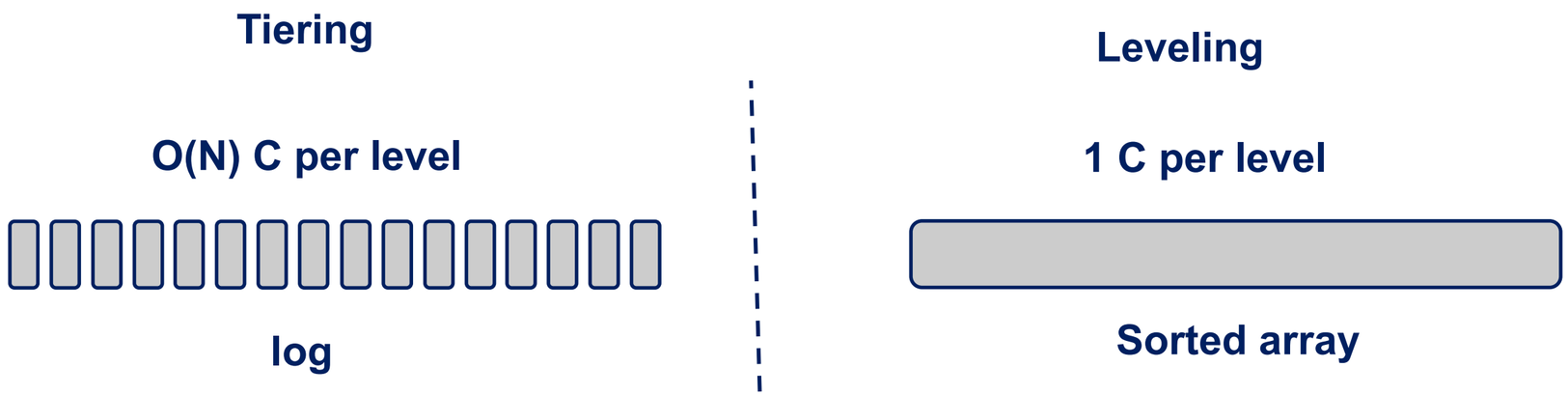


C - Component

Merge



Size ratio(T=2)

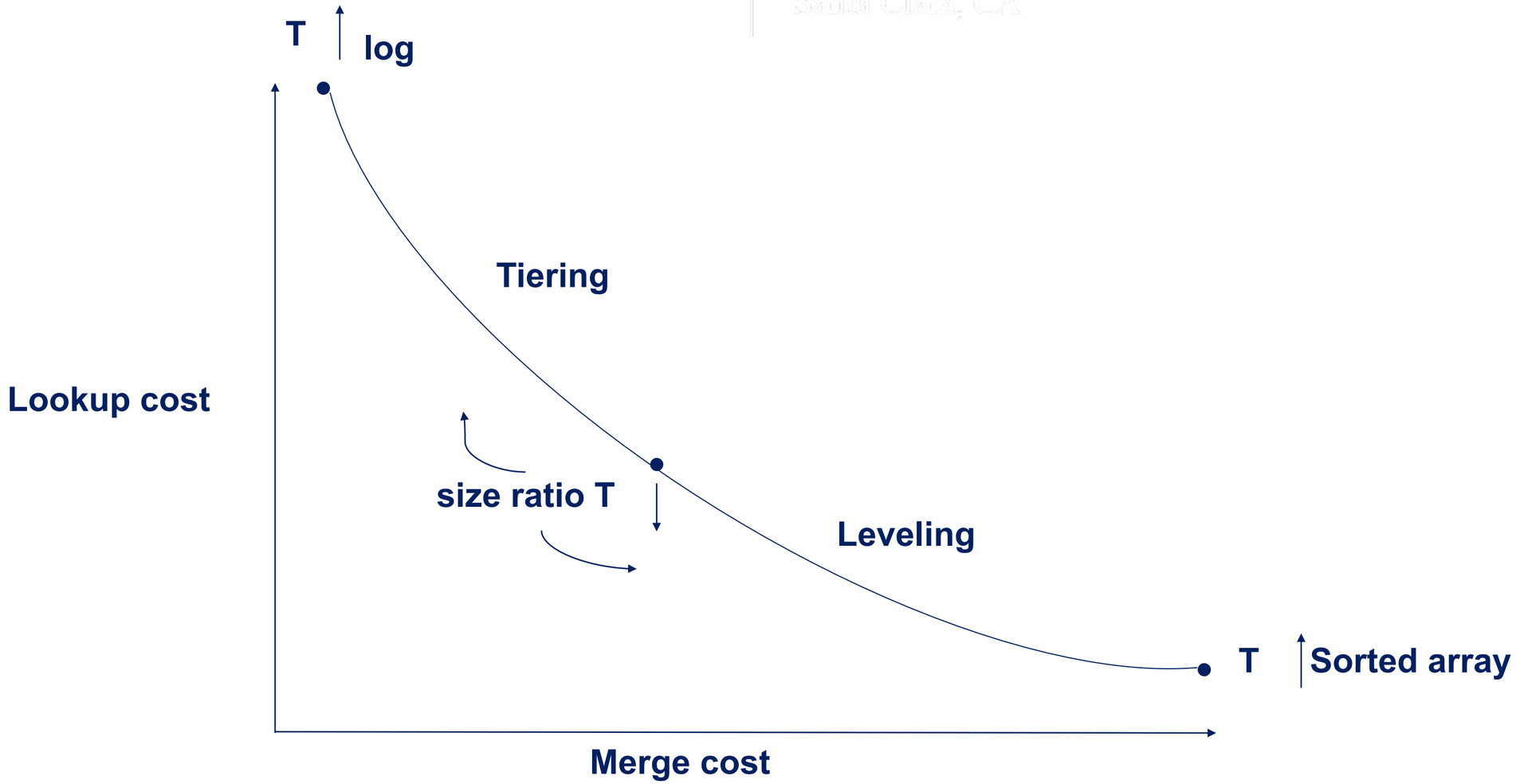


Size ratio(T=Very Large)

C - Component

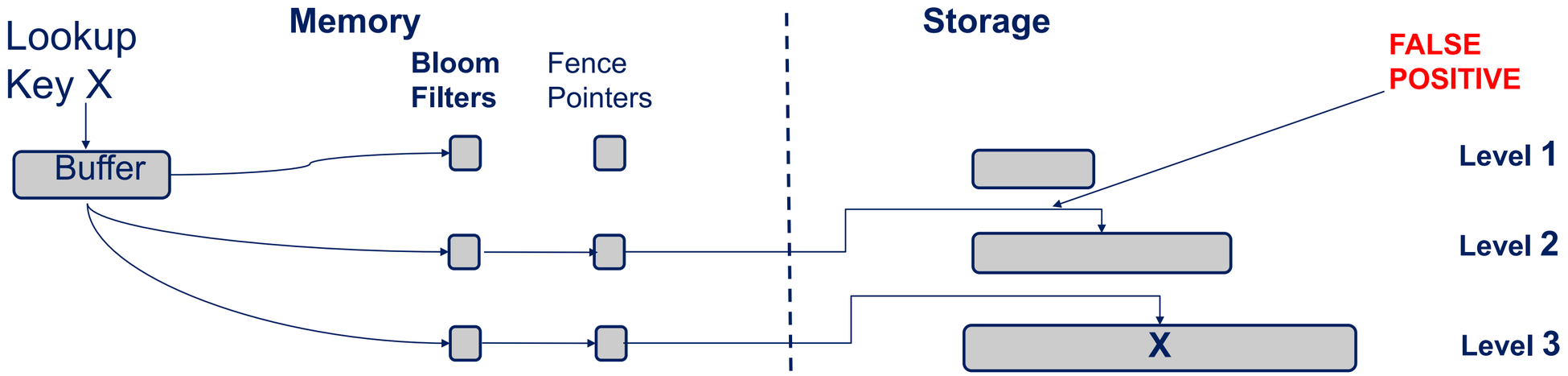
Ref: Niv Dayan

Merge



Well Known Optimizations - Bloom Filters

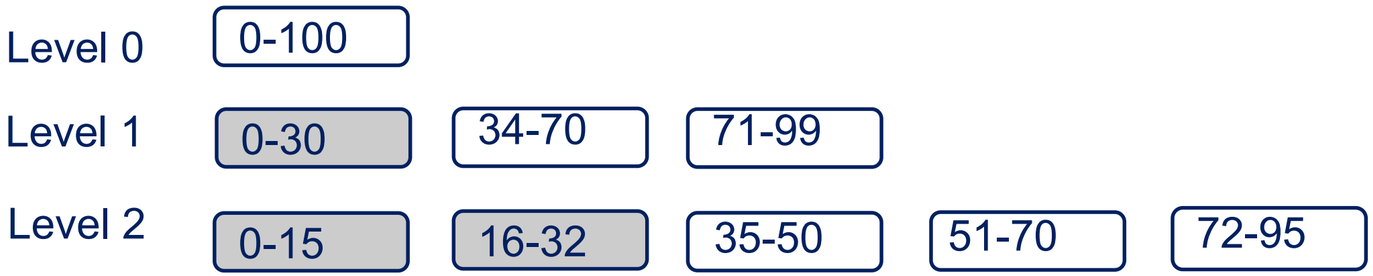
- Answer set membership queries
- False negative ? Never, False positive ? Can
- In practice around 10bits/key -> 1% false positive rate
- False positive impact ? No correctness issue, waste extra IO searching non-existent key
- Point lookup queries benefited
- Not usable for range queries



Well known Optimizations - Partitioning

- Range partition disk components into multiple small partitions
- SSTable used to denote such partition (levelDB)
- Breaks large C merge into smaller ones.
  - Bounds processing time of each merge &
  - Temporary disk space
- Orthogonal to merge policies

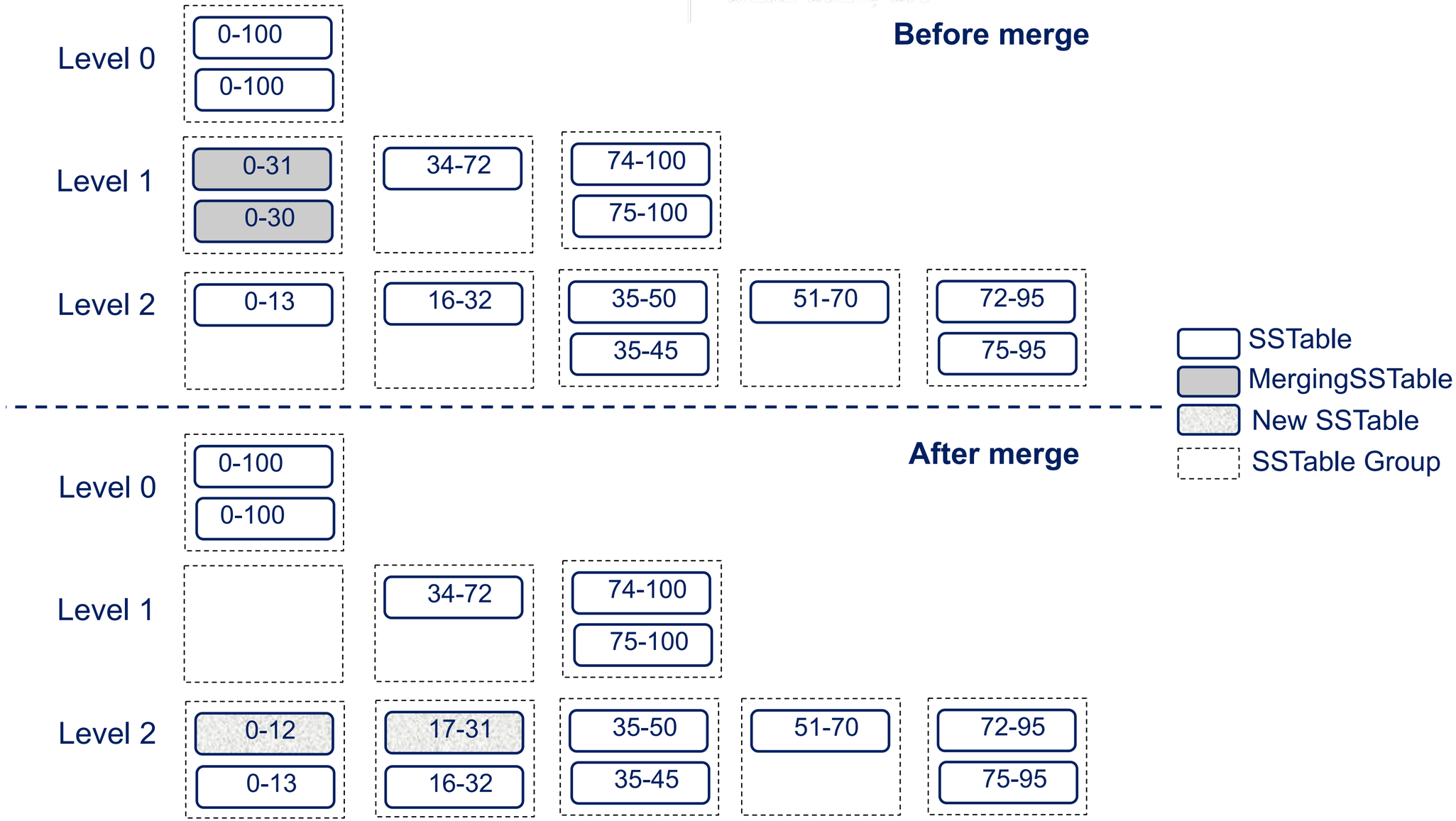
Partitioned Leveling Merge Policy



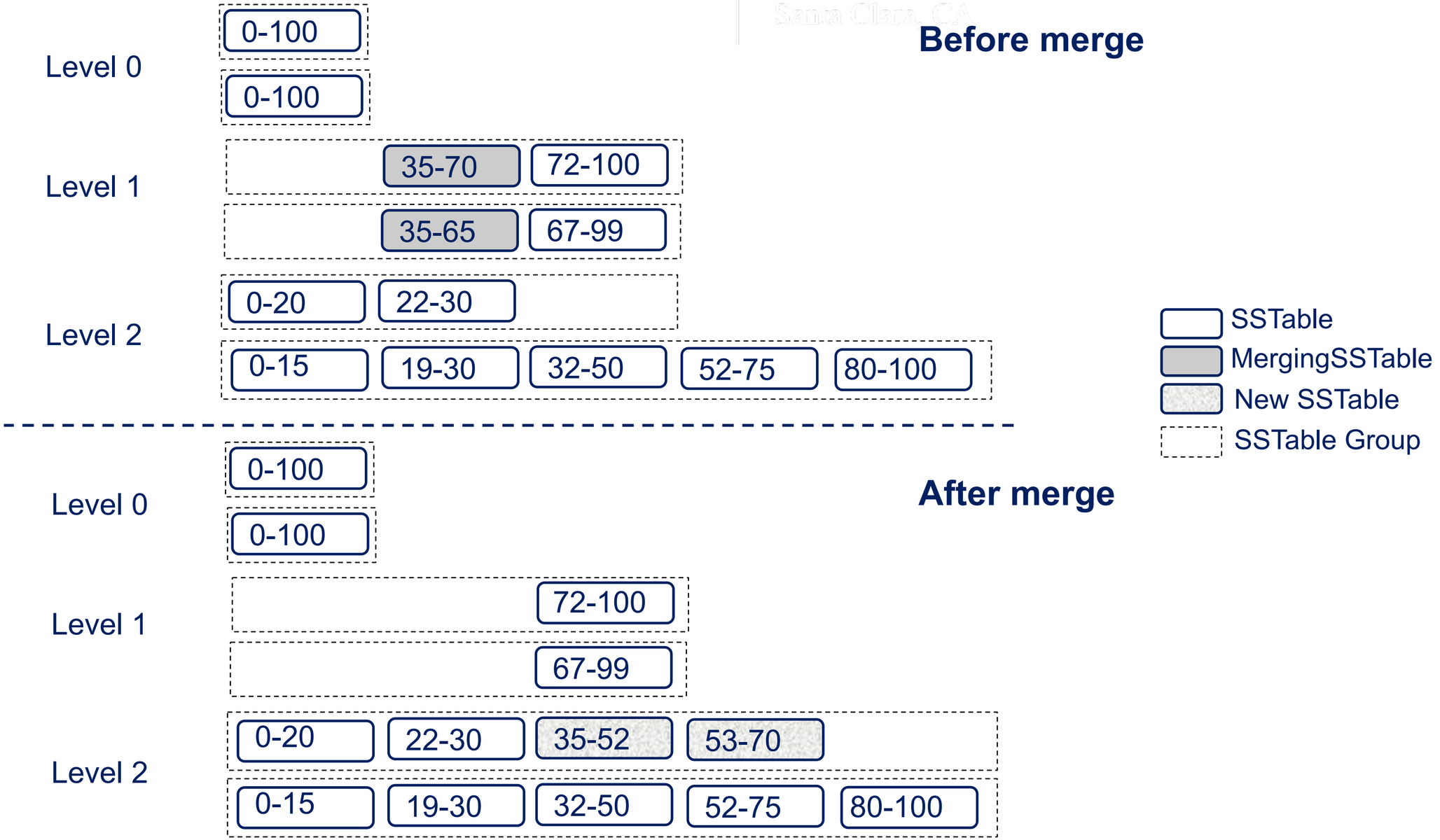
Before merge

After merge

Partitioned Tiering Vertical Grouping



Partitioned Tiering Horizontal Grouping



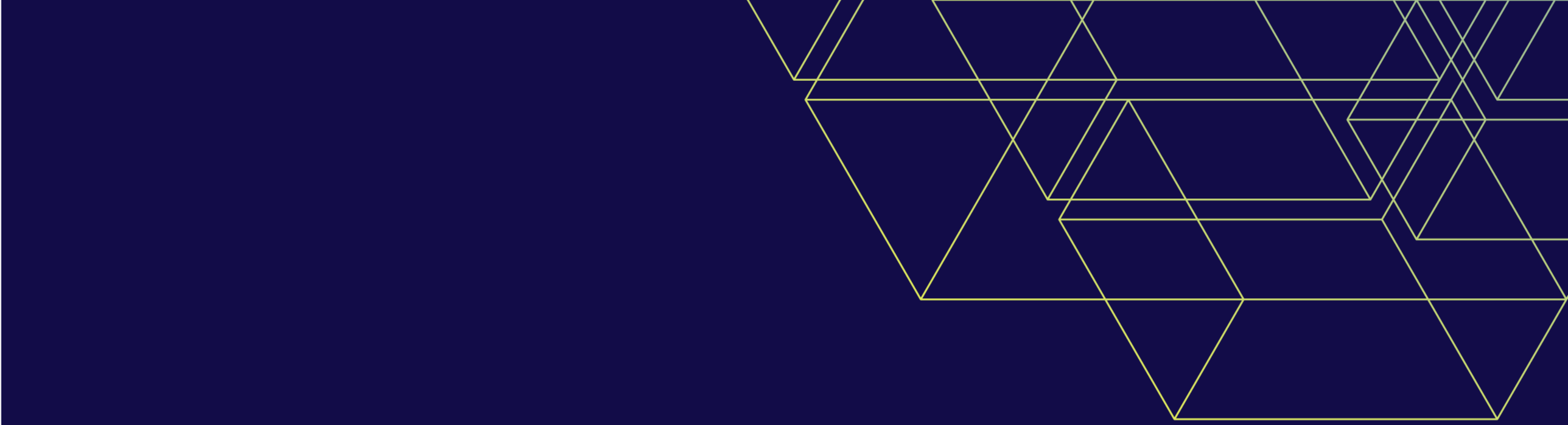
Concurrency Control & Recovery

- Writes appended to memory, WAL for durability.
- No-steal buffer mgmt policy
- Recovery
  - Transaction log replay
  - List of active components needs to be recovered
    - For un-partitioned :
      - Find all components with disjoint timestamps.
      - Overlapping timestamps components, comp with largest timestamp chosen & rest deleted, since they would have merged to form this selected component.
    - For partitioned (levelDB, RocksDB)
      - Timestamps approach doesn’t work
      - Maintain separate metadata log, stores all changes like add/delete SSTables.
      - Replay this log during recovery.

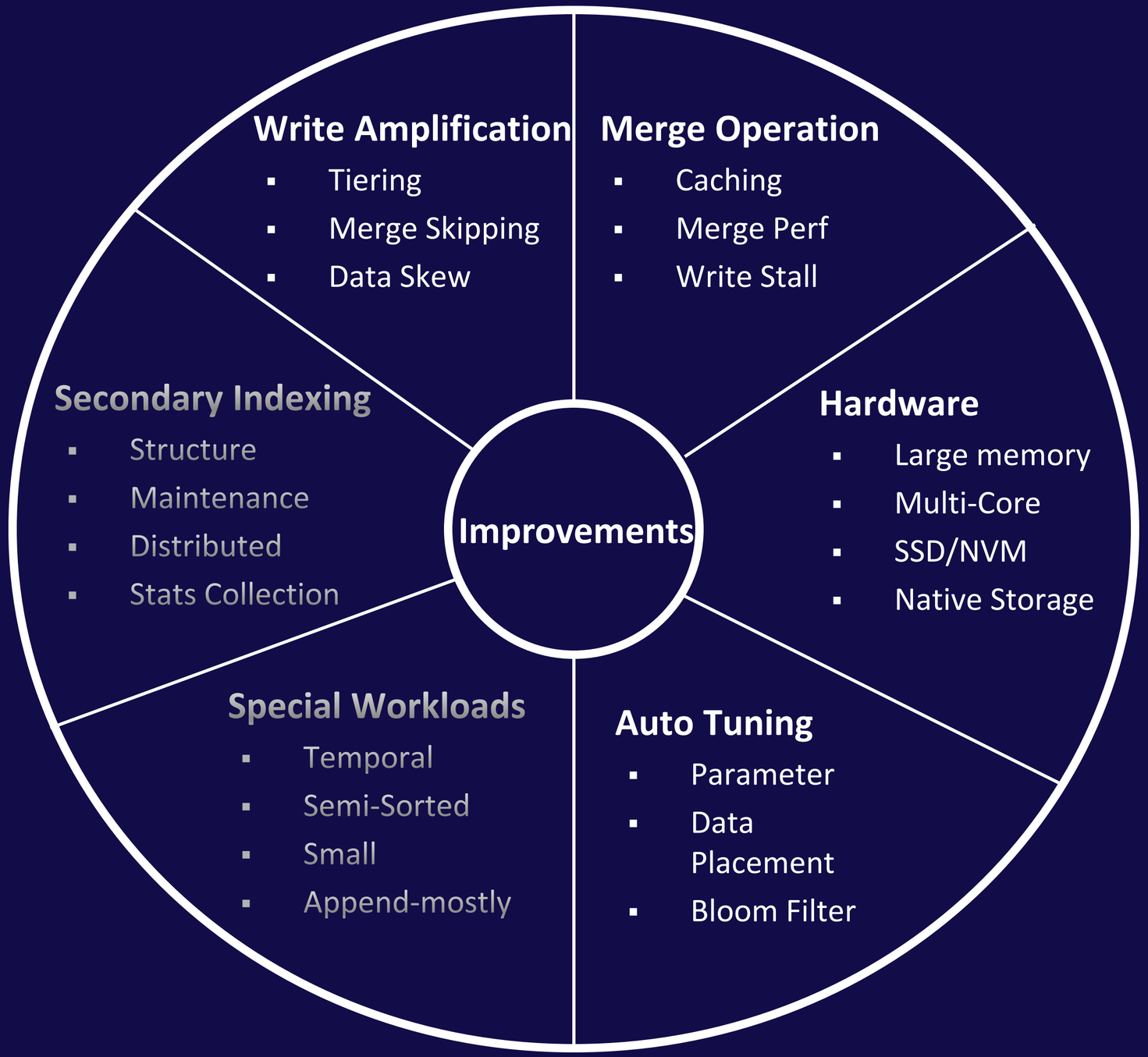
Cost Analysis

- Cost of writes/queries measured by counting disk IOs.
- Un-partitioned LSM-tree & worst case analysis.
- Size ratio T, no of levels L, B page size (no of entries each data page can store)
- P -> no of pages for memory component.
- Level i contains at most $T^i+1 * B * P$ entries
- s unique keys accessed by range query

Merge Policy	Write	Point Lookup (Zero/Non-zero)	Short range query	Long range query	Space amplification
Leveling	O(T * L / B)	O(L * $e^{-M/n}$) / O(1)	O(L)	O(s / B)	O((T+1) / T)
Tiering	O(L / B)	O(T * L * $e^{-M/N}$) / O(1)	(T* L)	O(T * s / B)	O(T)



LSM Tree Improvements



Reducing write amplification - Tiering

- WBTree
- Light Weight Compaction (LWC)-Tree
- PebblesDB
- dCompaction
- All share similar high-level design based on partitioned tiering with vertical grouping
- Main difference is how workload balancing of SSTable groups is performed
  - WBTree - relies on hashing, so gives ups range query support
  - LWCTree – dynamically shrinks key ranges of dense SSTable groups
  - PebblesDB – relies on probabilistically selected guards.
  - dCompaction – no built in support for workload balancing.
- Skewed SSTable groups impact perf of these structures needs evaluation
- SirfDB -> Partitioned tiering design with horizontal grouping

Reducing write amplification

- Merge skipping
  - Skip-tree
    - Each entry merges from level 0 down to largest level.
    - Directly push some entries to higher level, by skipping some level-by-level merges
    - This will reduce total write cost.
    - Uses mutable buffers
    - Introduces non-trivial implementation complexity
- Exploit data skew
  - TRAID
    - Separate hot keys from cold keys, so that cold keys are flushed.
    - Hot keys old versions discarded without flushing
    - Hot keys not flushed, hence copied to transaction log
    - Optimization – use transaction log as disk component, index built on top
    - Range query perf impacted, since values not sorted in log

Optimize merge operations

- Improve merge perf
  - VT-Tree
    - Stitching op, no overlap while merging, resultant SSTable points to this page.
    - Avoids reading/copying it again.
    - Drawbacks
      - Causes fragmentation, since pages no longer continuous on disk.
      - To alleviate this, it stiches only when there are K (stitching threshold) continuous pages
      - Since keys in stiched pages are not scanned, bloom filters cannot be produced.
      - To address this VT-Tree uses quotient filters
  - Pipelined merge to utilize CPU & IO parallelism [Zhang]
    - Merge op phases
      - Read, merge-sort, write
    - Read, write IO-heavy, while merge-sort CPU heavy.

Optimize merge operations

- Reducing buffer cache misses
  - Merge ops can interfere with caching behavior of system
  - New component enable, can cause buffer cache misses since new component is not cached yet.
  - If all pages of new component caches during merging, it would evict lots of working pages, again causing buffer cache misses.
  - Log-structured buffer merge Tree.
- Minimize write stalls
  - Due to heavy background operations – flushes, merges
  - bLSM – spring-and-gear merge scheduler // unpartitioned level merge policy
    - Tolerate extra component at each level, so that merges at different levels can proceed in parallel.
    - Limits max write speed at memory component to eliminate large write stalls.
    - Drawbacks
      - Only designed for unpartitioned leveling merge policy
      - Bounds max latency of writing to memory component, queueing latency is ignored.

Hardware Opportunities

- Large memory
  - FloDB
  - Accordion
- Multi-core
  - cLSM
- SSD/NVM
  - FD-Tree similar to LSM, to reduce random writes on SSDs.
  - FD+-Tree improves merge process.
  - WiscKey
  - HashKV
  - SifrDB
  - NoveLSM
- Native Storage

Auto-tuning

- Reduces burden on end user.
- Parameter-tuning
  - Monkey
- Tuning merge policy
  - Dostoevsky
- Native Storage

LSM-based Systems

- LevelDB – open-sourced by google 2011
  - Simple KV interface puts, gets, scans.
  - Embedded storage engine for higher-level applns
  - Partitioned leveling merge policy
- RocksDB – fork of levelDB, by faceboo, 2012
  - Lots of features
  - Major motivation for fb, was good space utilization
  - Size ratio defaults to 10, leveling impl at 90% data at largest level
  - Improvements to partitioned leveling merge

Summary

- RUM conjecture
- Learnt about basics of LSM trees
- Various optimizations taxonomy
- LSM-based practical systems

References

- http://daslab.seas.harvard.edu/rum-conjecture/
- LSM-Tree [Patrick O'Neil]
- LSM-based Storage Techniques: A Survey [Chen Luo, Michael J. Carey]
- Niv Dayan talks
- https://medium.com/databasss
- Mark Callaghan talks/blogs
- http://www.benstopford.com/2015/02/14/log-structured-merge-trees/
- Special thanks to Chen Luo [for answering questions related to LSMs]
- https://blog.acolyer.org/2014/11/26/the-log-structured-merge-tree-lsm-tree/
- Special thanks to [for inspiration]
  - Dr. Vijay Gokhale
  - Prof. Remzi H. Arpaci-Dusseau
  - Prof. Andy Pavlo
  - Prof. Erez Zadok

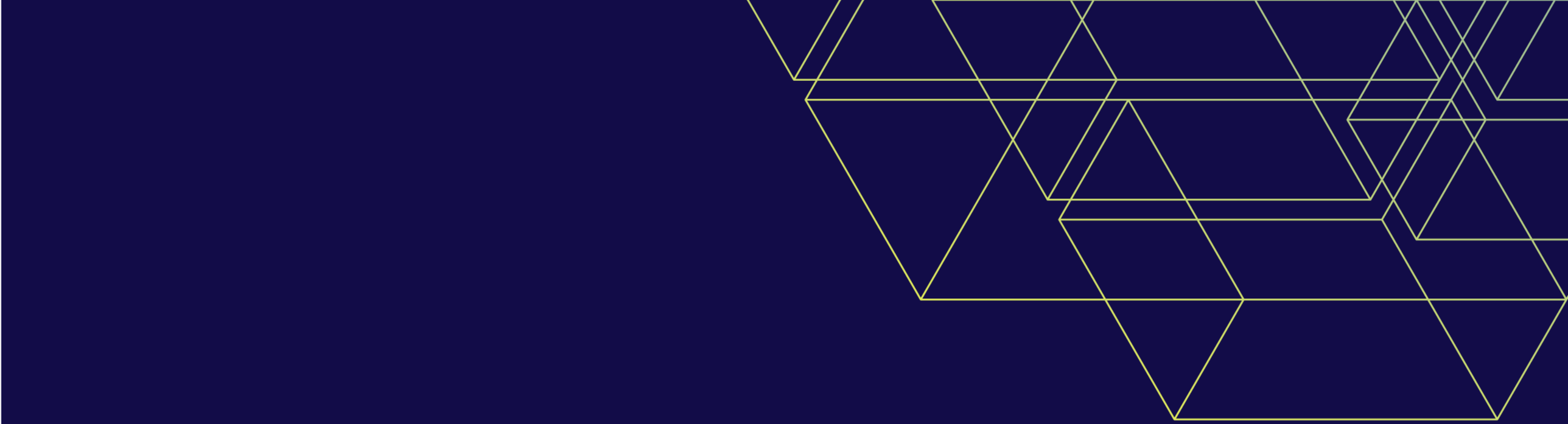
Shriram Pore

Shriram Pore is a Storage industry veteran holding around two decades of extensive experience. In the current role of Associate VP – Storage Engineering at MSys, Shriram is leading a strategic cross-company effort to build a Cloud Engineering CoE overseeing a range of roles including product engineering for storage, virtualization and in data center technologies.

Rohan Puri

Rohan holds over a decade of experience in developing storage software, mainly file systems. At Msys, Rohan leads the file systems development effort with ownership of modules like synchronous replication, transactional store, file checksums, software encryption, file systems metadata.

Questions?



Thank You