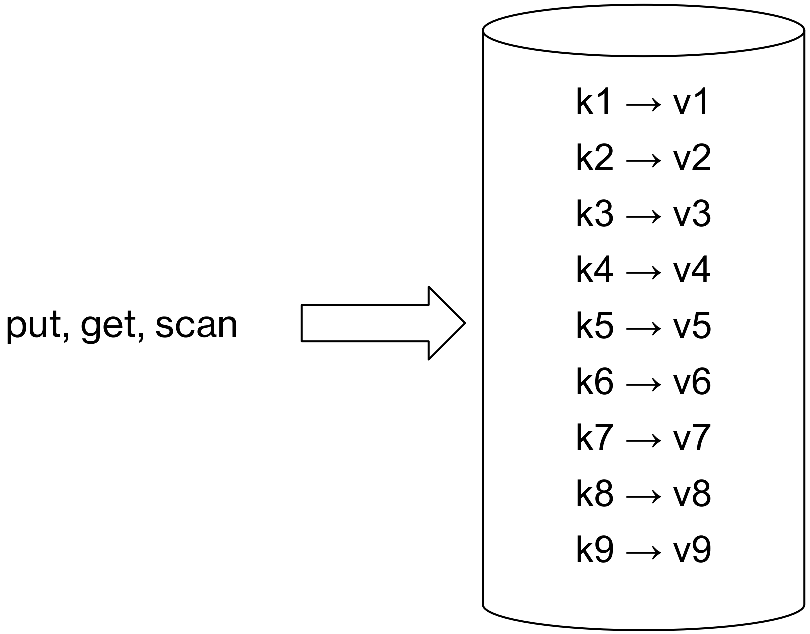
EvenDB: Optimizing Key-Value Storage for Spatial Locality

Eran Gilad, Edward Bortnikov, Anastasia Braginsky, Yonatan Gottesman, Eshcar Hillel (Yahoo Research), Idit Keidar (Technion), Nurit Moscovici (Outbrain), Rana Shahout (Technion)

EURO/SYS'20

Key-value stores

- **key -> value mapping**



put, get, scan

k1 → v1

k2 → v2

k3 → v3

k4 → v4

k5 → v5

k6 → v6

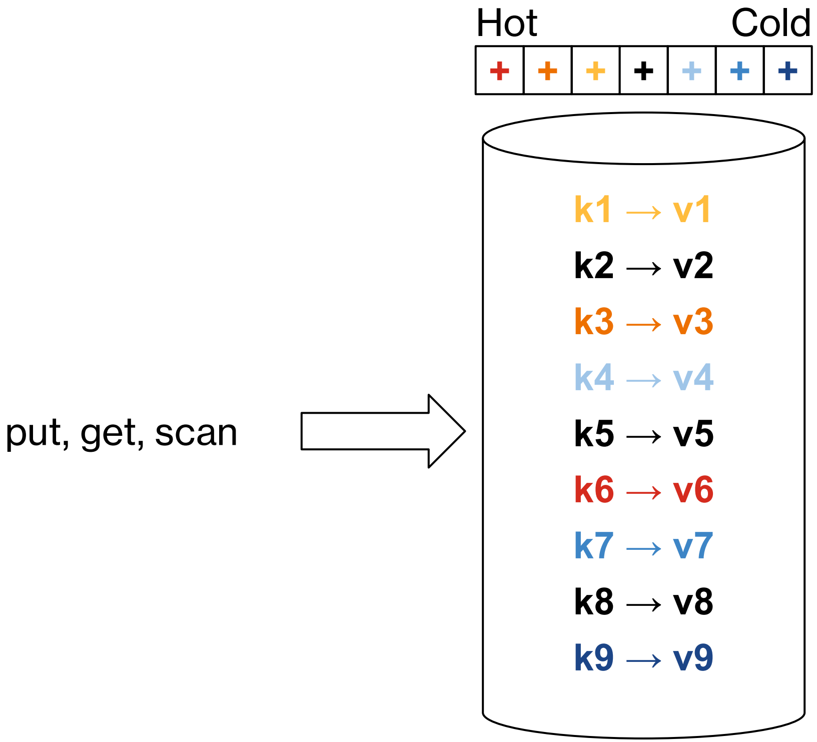
k7 → v7

k8 → v8

k9 → v9

Key-value stores

- key -> value mapping
- **skewed workload: some *keys* are hotter**



Hot Cold

+ + + + + + +

put, get, scan

k1 → v1

k2 → v2

k3 → v3

k4 → v4

k5 → v5

k6 → v6

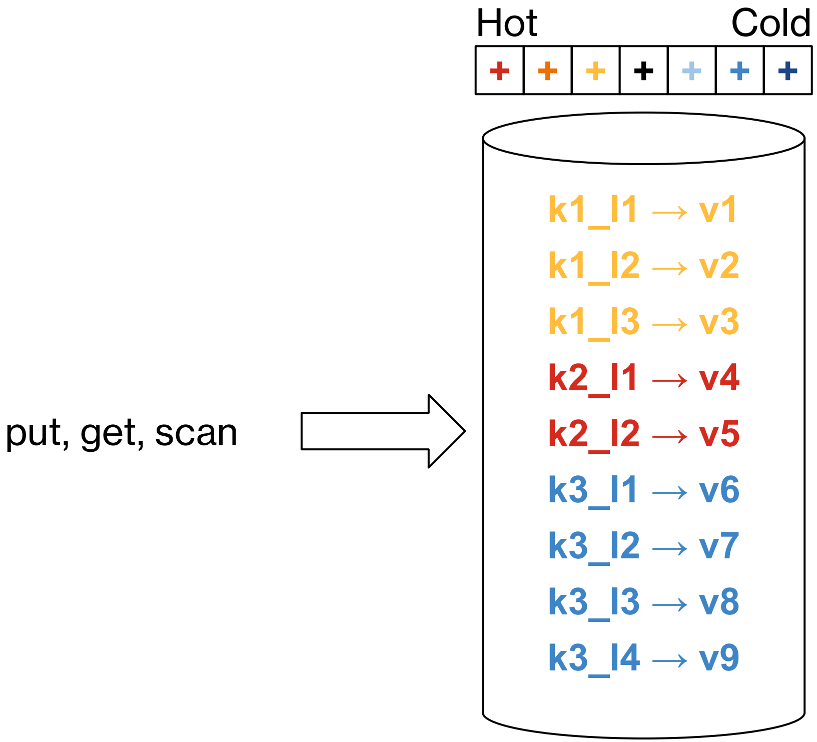
k7 → v7

k8 → v8

k9 → v9

Key-value stores

- key -> value mapping
- skewed workload: some *keys* are hotter
- **spatial locality: some *ranges* are hotter**
  - e.g., complex keys



put, get, scan

Hot Cold

+	+	+	+	+	+	+

k1_l1 → v1

k1_l2 → v2

k1_l3 → v3

k2_l1 → v4

k2_l2 → v5

k3_l1 → v6

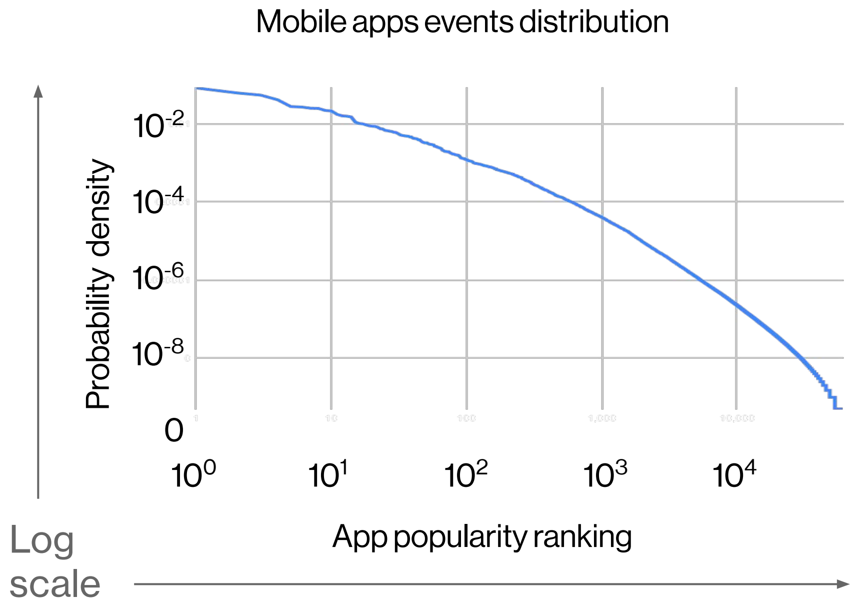
k3_l2 → v7

k3_l3 → v8

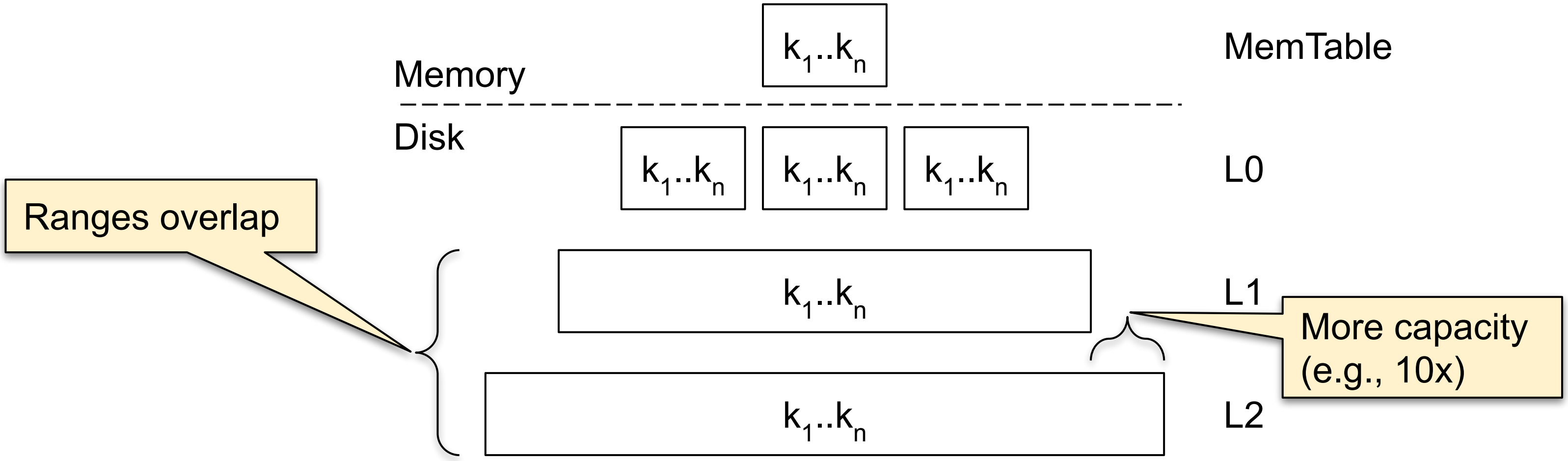
k3_l4 → v9

Key-value stores

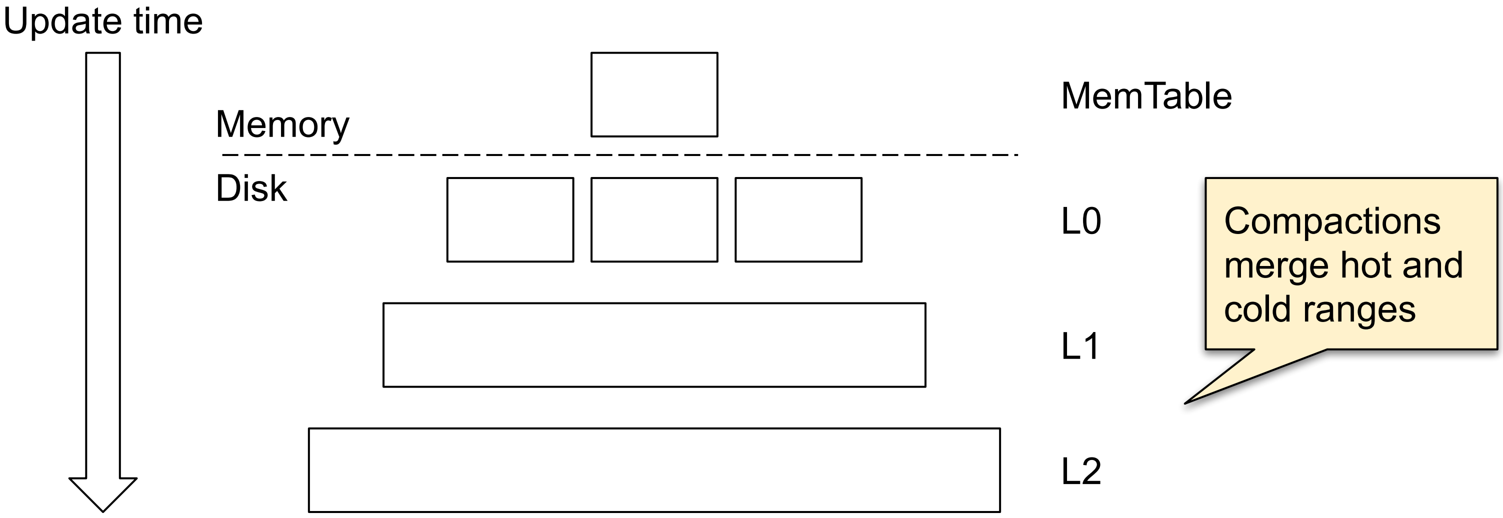
- key -> value mapping
- skewed workload: some *keys* are hotter
- spatial locality: some *ranges* are hotter
  - e.g., complex keys
- **Sample production trace:**
  - appname_timestamp
  - 1% of apps ⇒ 1% key prefixes ⇒ 94% of events



LSM-trees

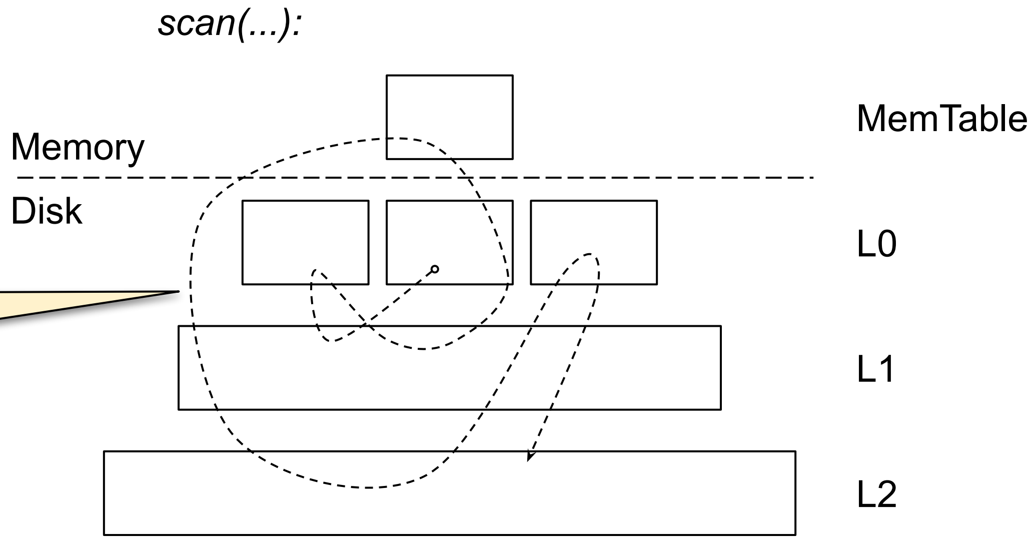

Memory
Disk
$k_1..k_n$
MemTable
$k_1..k_n$ $k_1..k_n$ $k_1..k_n$
L0
$k_1..k_n$
L1
$k_1..k_n$
L2
Ranges overlap
More capacity (e.g., 10x)

LSM-trees are designed for temporal locality



Compactions merge hot and cold ranges

LSM-trees are less suited for spatial locality



Ranges are fragmented

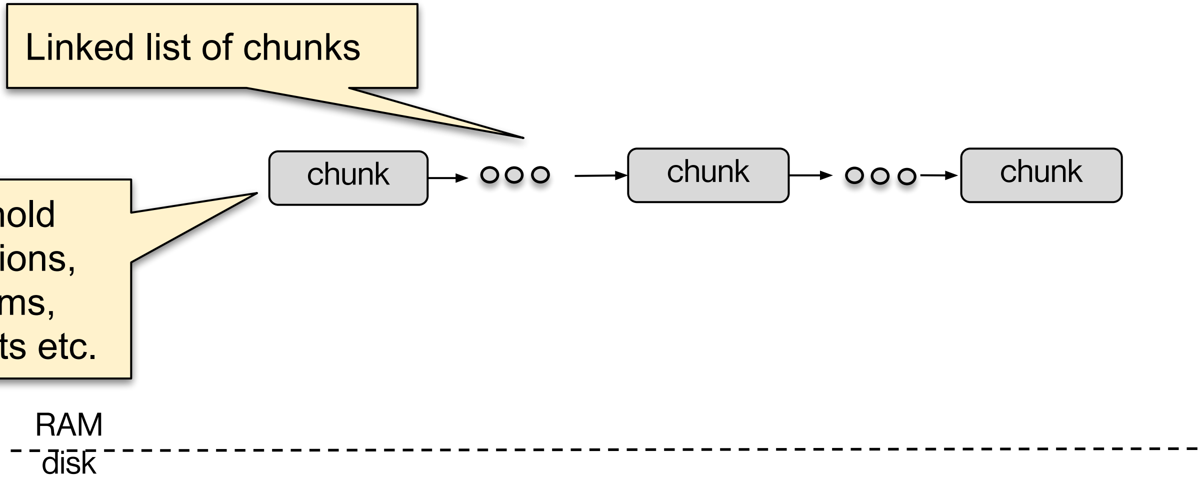
EvenDB

- Ordered key-value store
- Optimized for spatial locality
- Low write amplification
- Persistent, fast recovery
- Atomic operations, including scan

Chunk-based organization

- **Dynamically partitioned key space into *chunks***
  - Much smaller than shards
  - Much larger than blocks
- **Chunks are the basic unit for**
  - Disk I/O
  - Compaction
  - Memory caching
  - Concurrency control

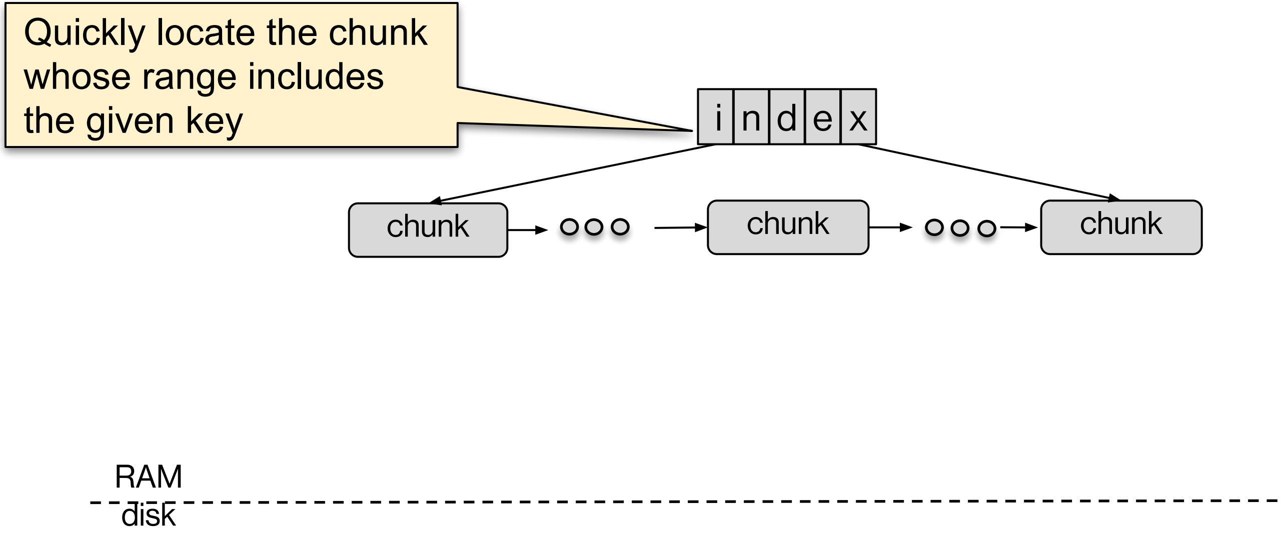
Chunks metadata



Linked list of chunks

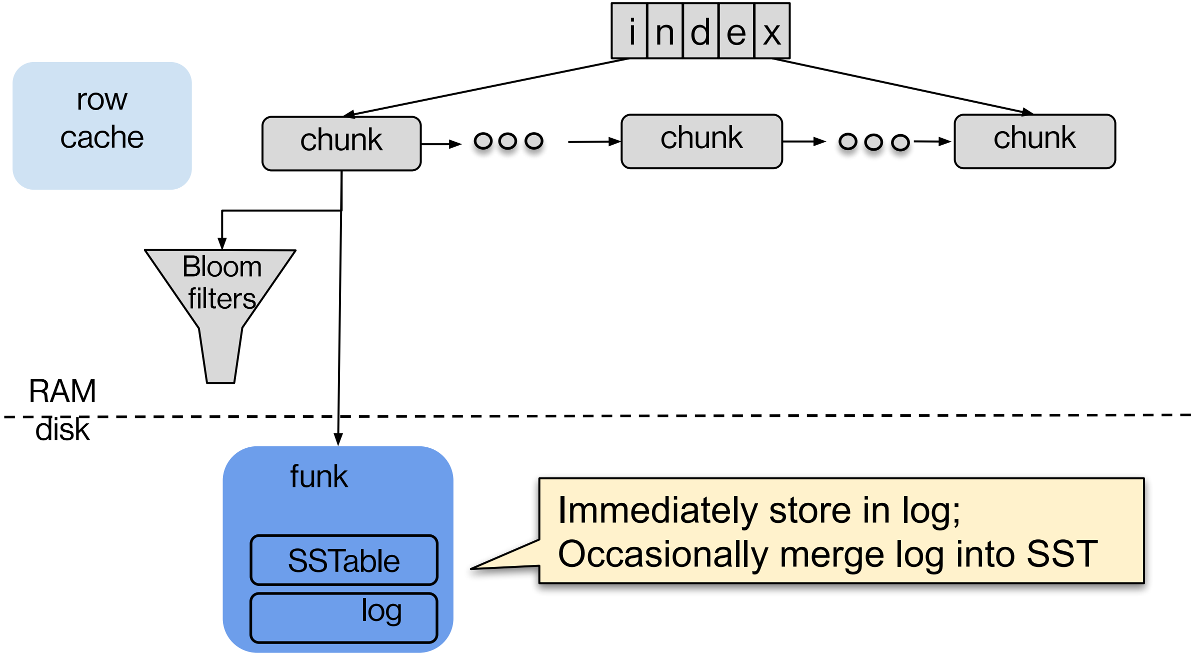
Chunk objects hold metadata - versions, sync. mechanisms, file handles, stats etc.

Chunks index



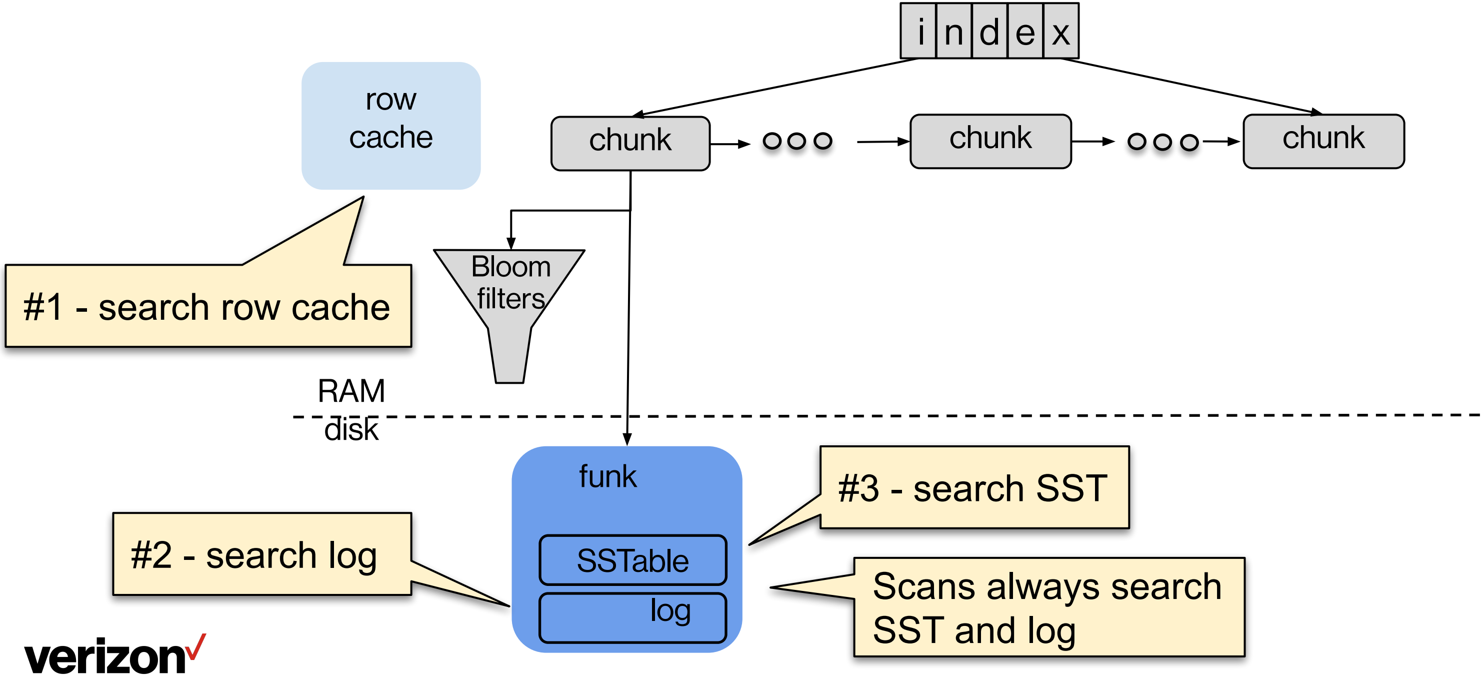
Quickly locate the chunk whose range includes the given key

Disk storage - updates



Immediately store in log;
Occasionally merge log into SST

Disk storage - lookups



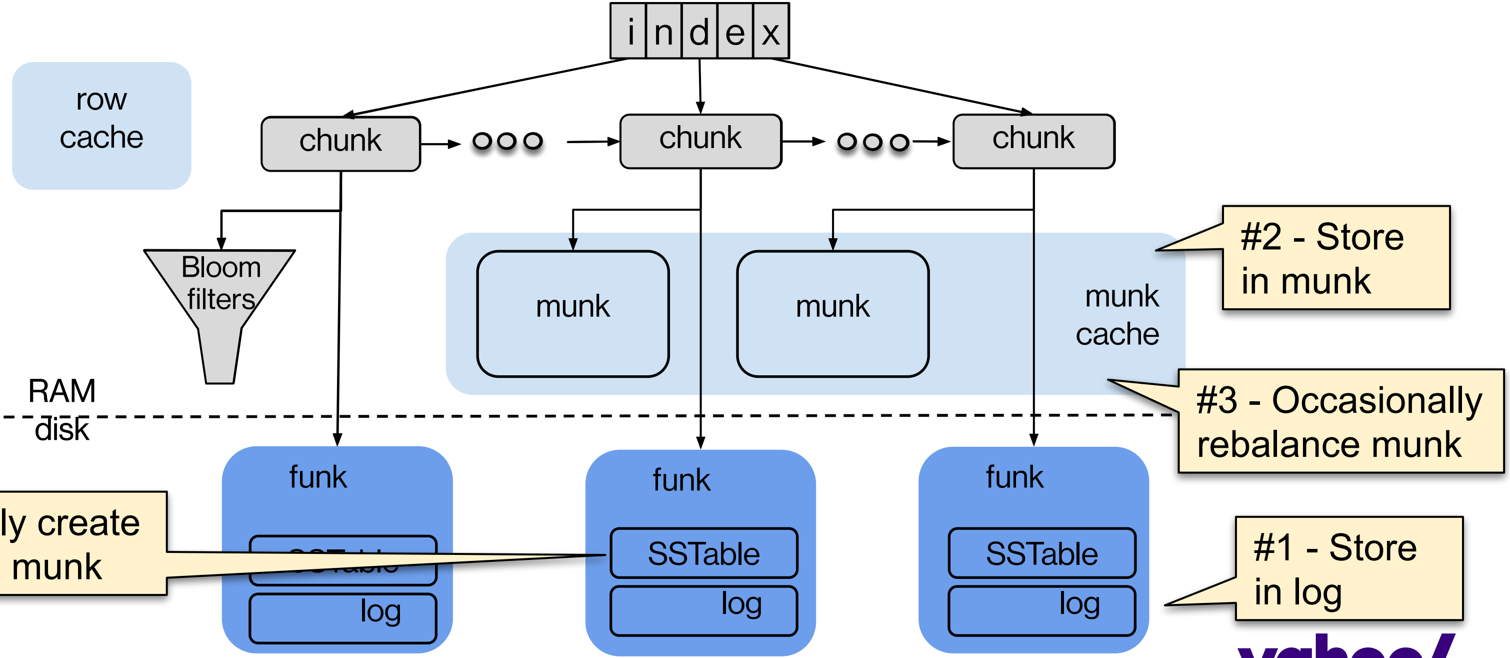
#1 - search row cache

#2 - search log

#3 - search SST

Scans always search SST and log

Memory cache - updates



#2 - Store in munk

#3 - Occasionally rebalance munk

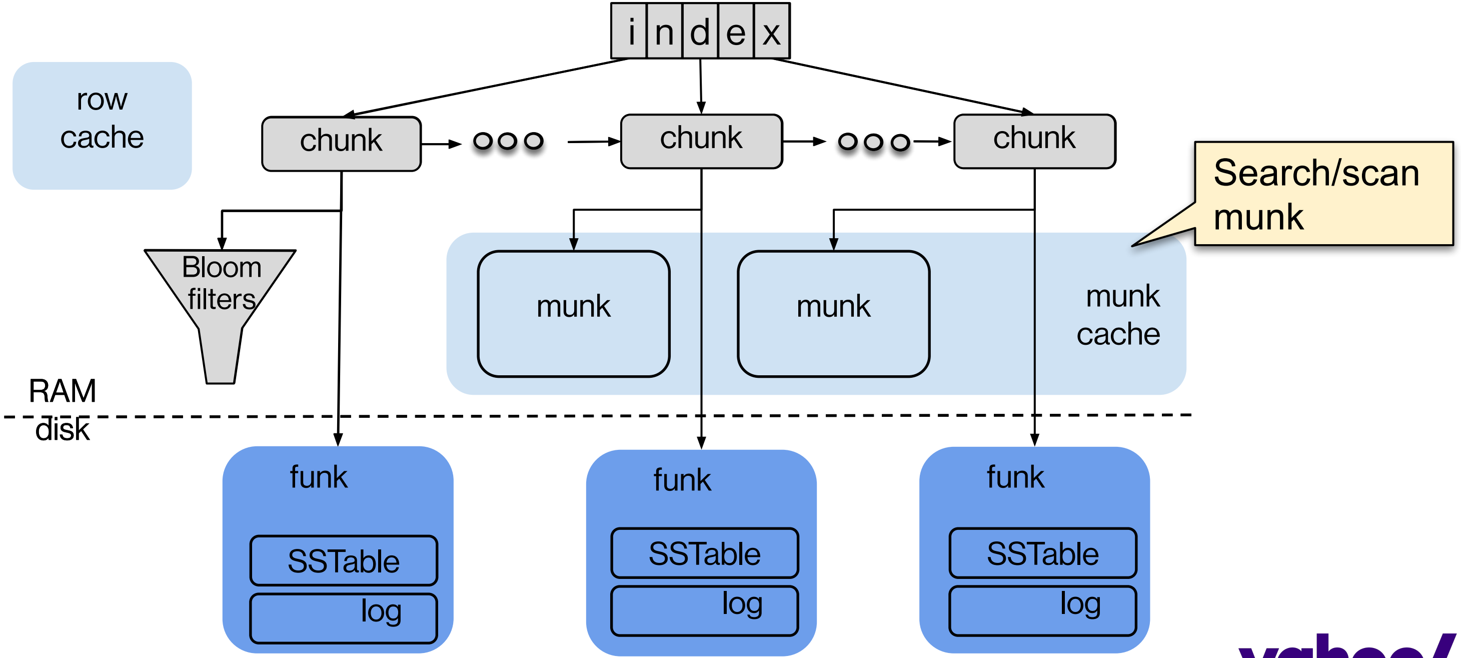
#1 - Store in log

#4 - Rarely create SST from munk

verizon media

yahoo! research

Memory cache - lookups

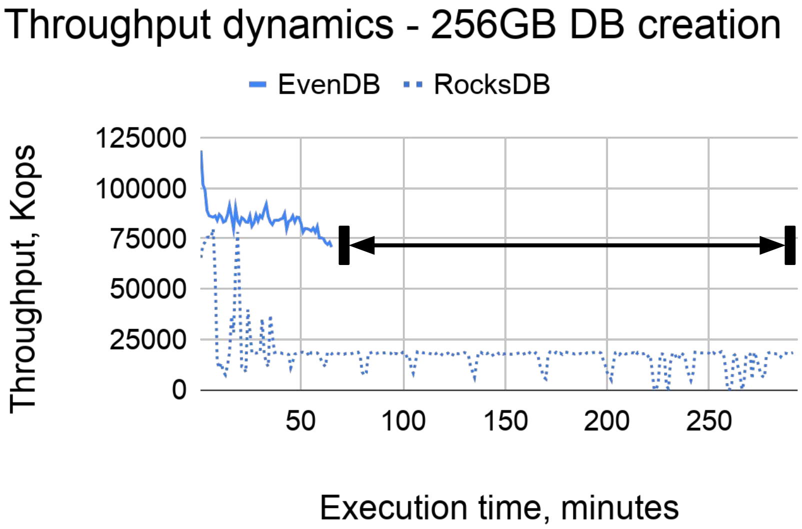


Evaluation

- **3 benchmark suites**
  - Traces from internal production system, 256GB DB - some presented next
  - Standard and extended YCSB benchmarks - results in paper

- **State-of-the-art LSM: RocksDB**

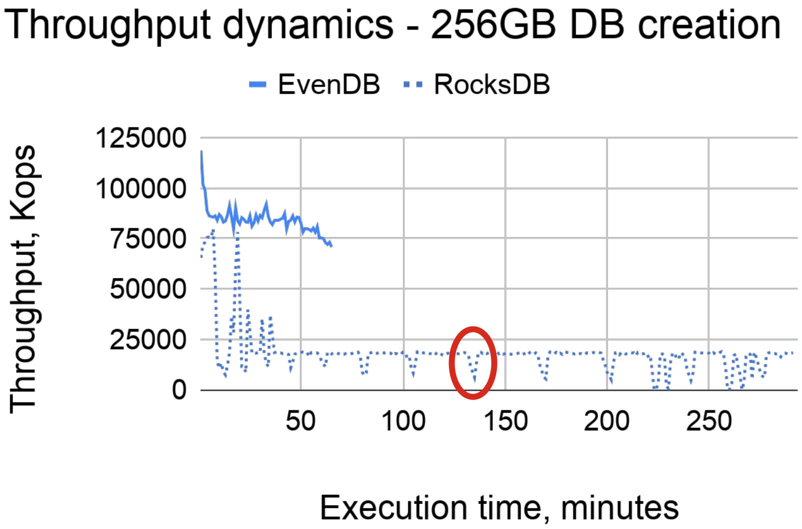
Real dataset ingestion

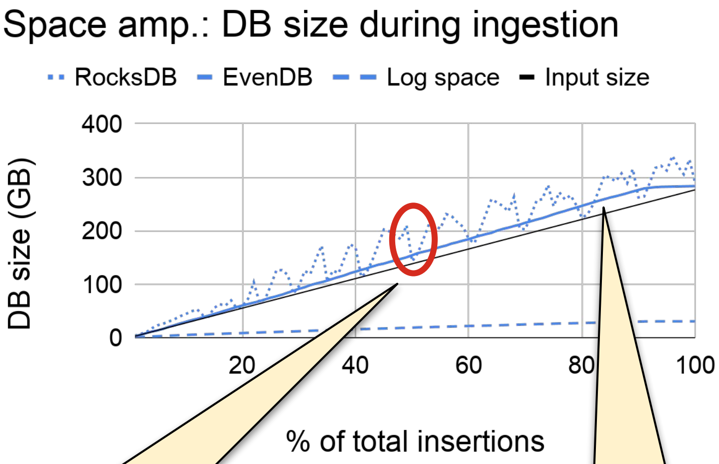


EvenDB 4.4x faster,

write amp. 4x lower (better)

Compactions impact

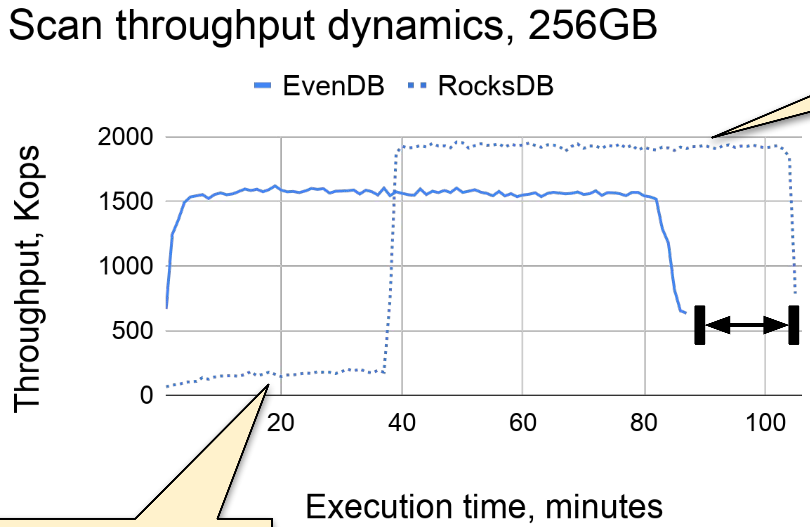




RocksDB throughput drops during compaction

EvenDB runs much smoother

Real dataset scans



RocksDB faster after storage optimized

~38 minutes stall after DB creation

EvenDB 1.2x faster than RocksDB

Summary

- **EvenDB introduces a novel key-value store architecture**
- **Chunk arrangement better suited for spatially-local workloads than LSM:**
  - Lower write amplification
  - Single level of storage, no overlapping
  - Memory serves reads and writes
- **EvenDB outperforms RocksDB when:**
  - Workload is spatially-local or most working set fits in RAM
  - In par otherwise
  - Demonstrated in real workload and synthetic YCSB benchmarks