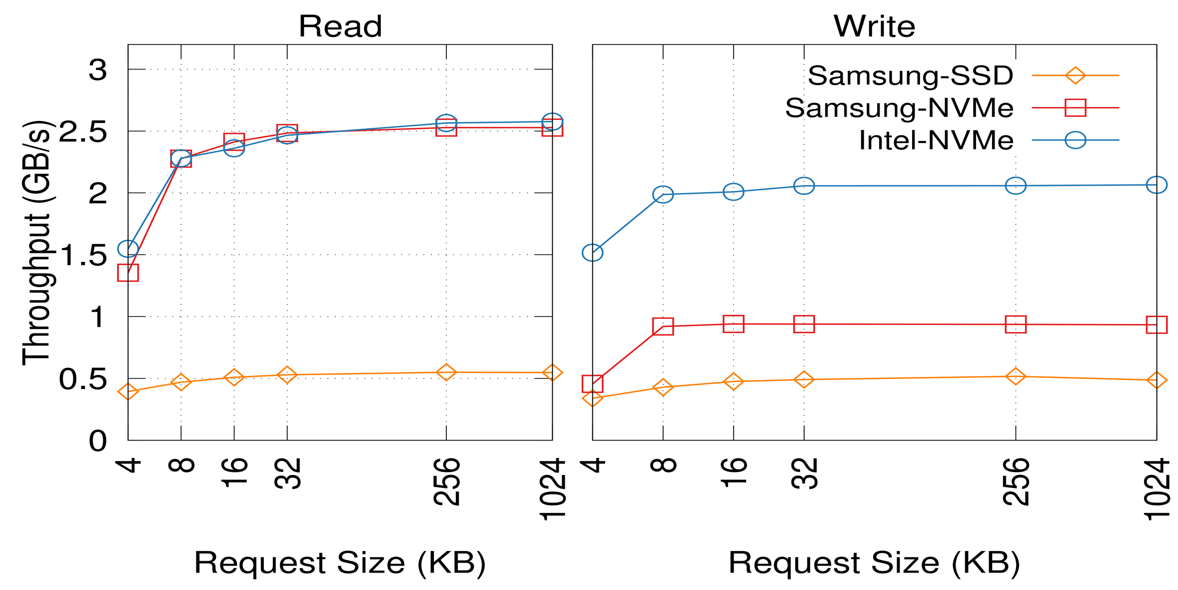
An Efficient Memory-Mapped Key-Value Store for Flash Storage

Anastasios Papagiannis, **Giorgos Saloustros**, Pilar González-Férez, Angelos Bilas
{apapag, **gesalous**, pilar, bilas}@ics.forth.gr
Foundation for Research and Technology - Hellas (FORTH), Greece
University of Crete, Greece University of Murcia, Spain

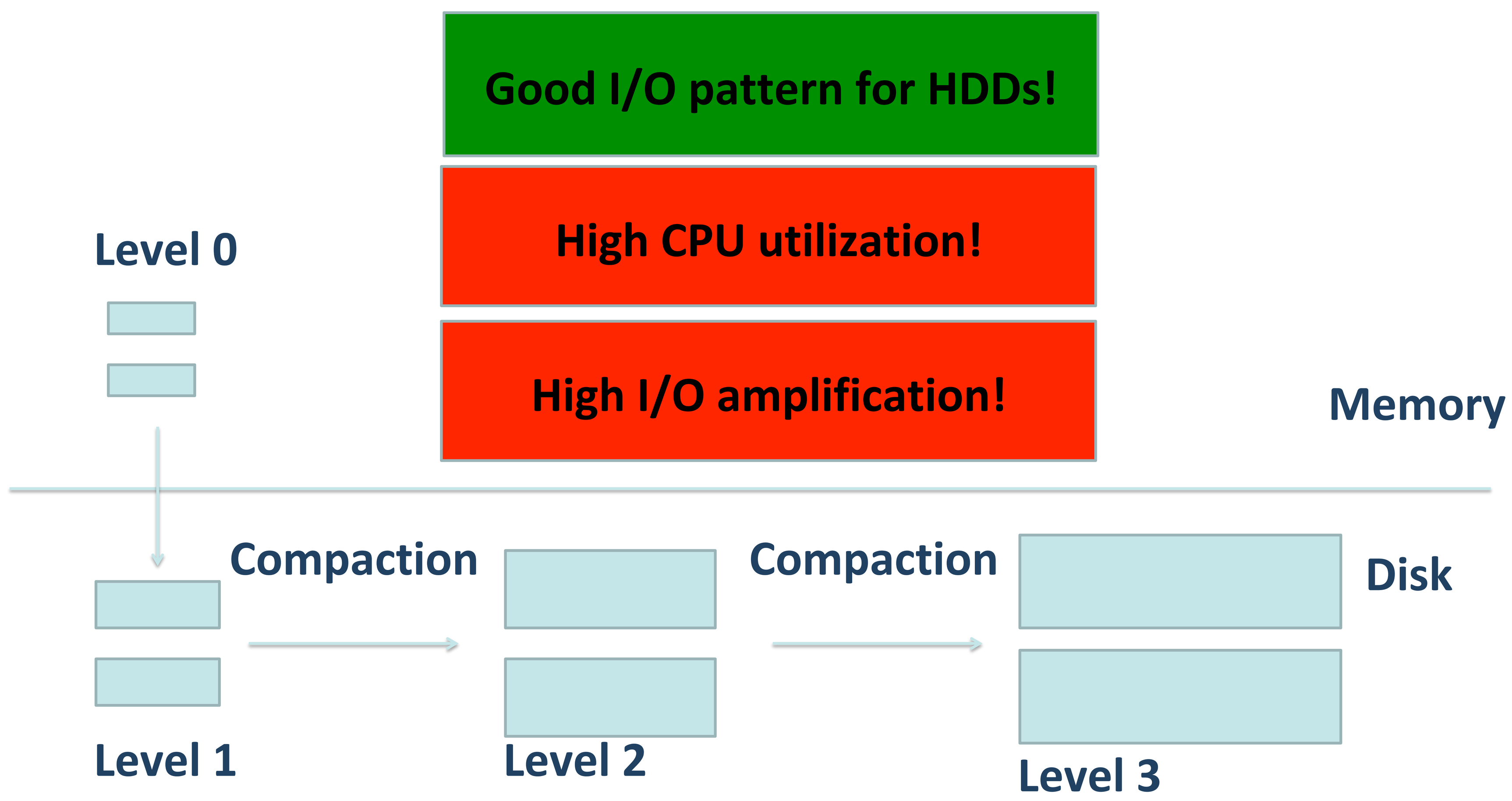
FORTH Institute of Computer Science

Key-Value Stores Today

- Key-value store: Dictionary that stores **arbitrary** key-value pairs
  - Used **extensively**: web indexing, social networks, data analytics
  - Supports inserts, lookups, scans and deletes
- Today key-value stores are **inefficient**
  - **Consume a lot of CPU cycles** - mainly designed for HDDs
- Our goal: **Improve efficiency of key-value stores**
  - Reconsider design of key-value stores for fast storage (SSDs)



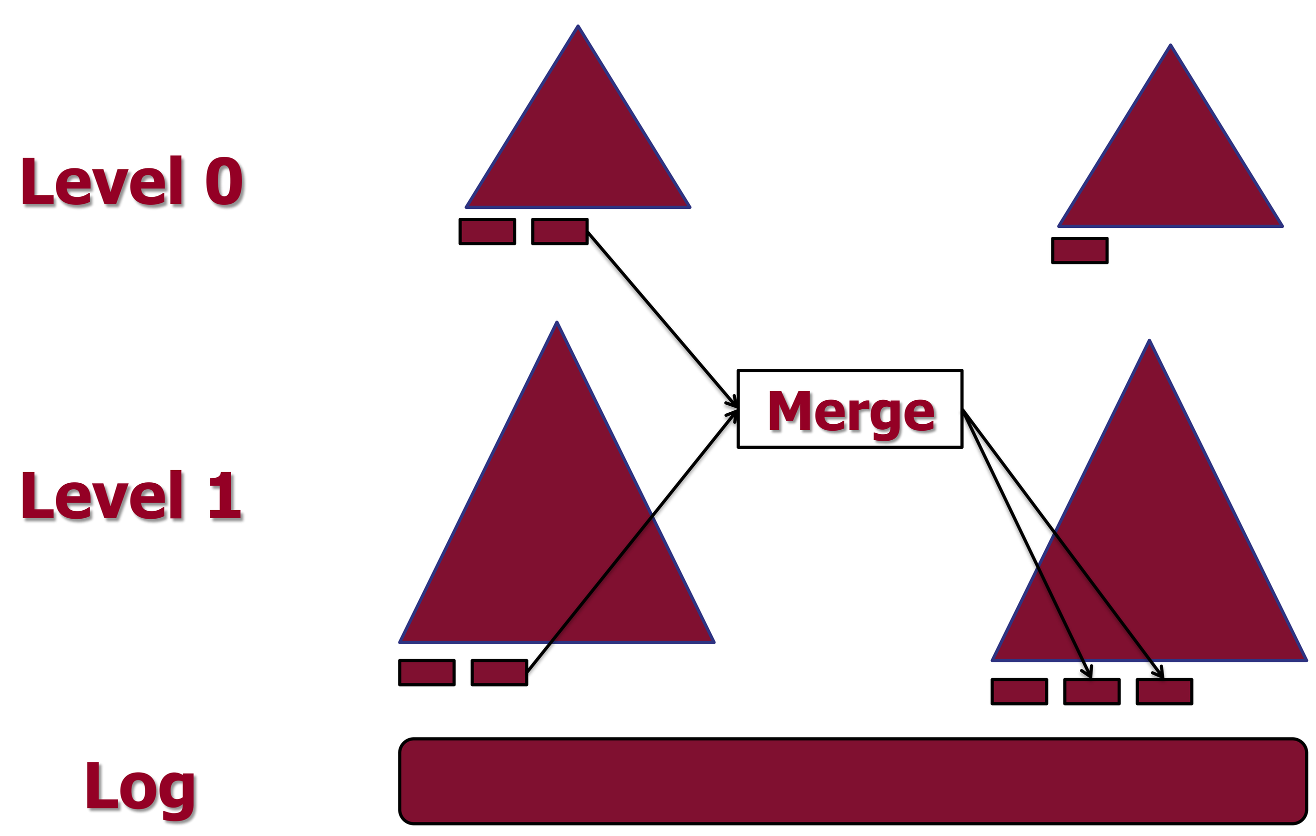
Issues with current approaches



- **Data caching and I/O**
  - Key-value stores require a user-space DRAM cache
  - Explicit I/O using read()/write()
  - Hits in DRAM require lookup – significant overhead
  - Both for index and data, in every traversal
  - Alternative: **mmap()**
- **Linux kernel mmap()**
  - Cannot control page eviction
  - System freezes with heavy workload – high tail latencies

Kreon Design

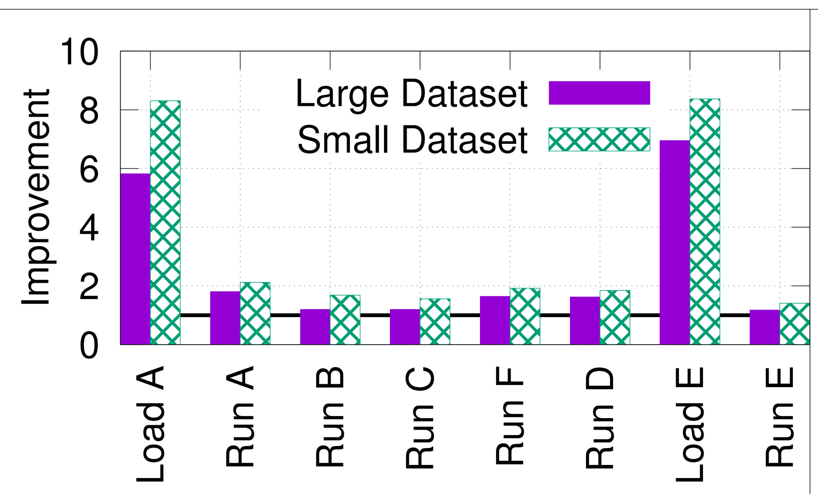
- **Spill vs. Compaction**
  - Produce **many small random read I/Os**
    - Does **not** affect performance in fast storage devices
  - Keep an **index per-level**
    - Does **not** require **sorting and merging**



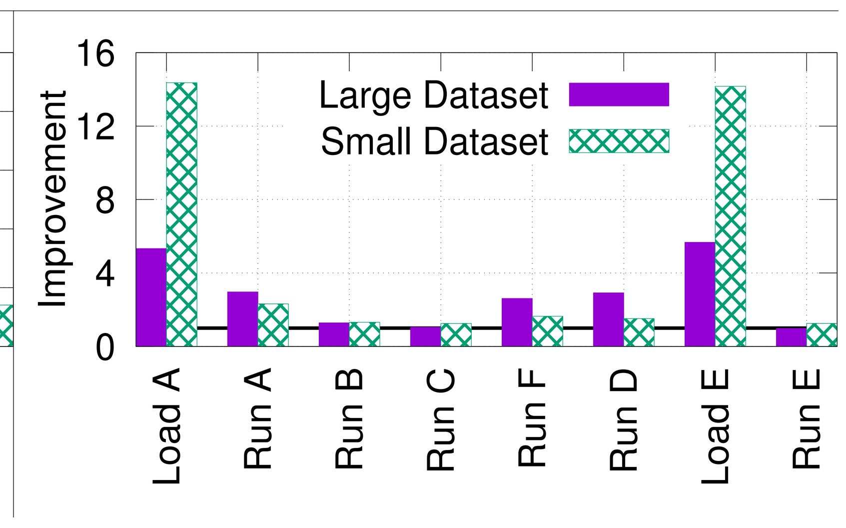
- **Kmmap vs. Linux kernel mmap()**
  - Choose what to evict using per-page priority
    - **Level-0: priority 0**
    - Level-1 (index): priority 1
    - Level-1 (leaves): priority 2
    - Append-only Log: priority 3
    - Evict pages with **high priority first**
  - Copy-On-Write allows us to optimize msync()
    - Keep **timestamp** when a page becomes dirty
    - On msync() write only pages that dirtied before this call
    - Allow to **dirty** pages **concurrently**

Experimental Analysis

- YCSB, Small (memory) and Large (device) Dataset
- **Efficiency:** Up to 8.3x less cycles/op compared to RocksDB.
- **Throughput:** Up to 5.3x more ops/sec compared to RocksDB.
- **I/O Amplification**: Up to 4.6x less MB written compared to RocksDB in write intensive workloads.



Efficiency



Throughput

With the support of the European Commission under the Horizon 2020 Framework Programme for Research and Innovation through the Vineyard (GA 687628) and ExaNeSt (GA 671553) projects.

Computer Architecture & VLSI Systems Laboratory (CARV)
http://www.ics.forth.gr/carv