Optimizing Flash-based Key-value Cache Systems

Zhaoyan Shen[†], Feng Chen[‡], Yichen Jia[‡], Zili Shao[†]

[†]Department of Computing, Hong Kong Polytechnic University

[‡]Computer Science & Engineering, Louisiana State University

The 8th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage’16)

Key-value Information

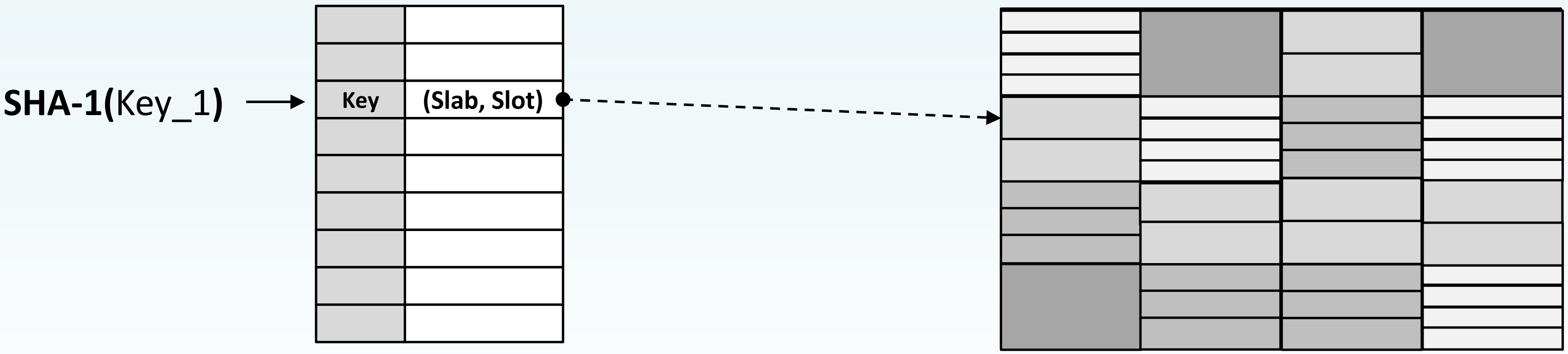
- Key-value access is dominant in web services
  - Many apps simply store and retrieve key-value pairs
  - **Key-value cache** is the first line of defense
    - Benefits: Improve throughput, reduce latency, reduce server load
  - In-memory KV cache is popular (Memcache)
    - High speed but has cost, power, capacity problem

Flash based Key-value Cache

Key-value cache

\+

Flash SSD



Hash Index (Memory)

Key-value Slabs (Flash LBA)

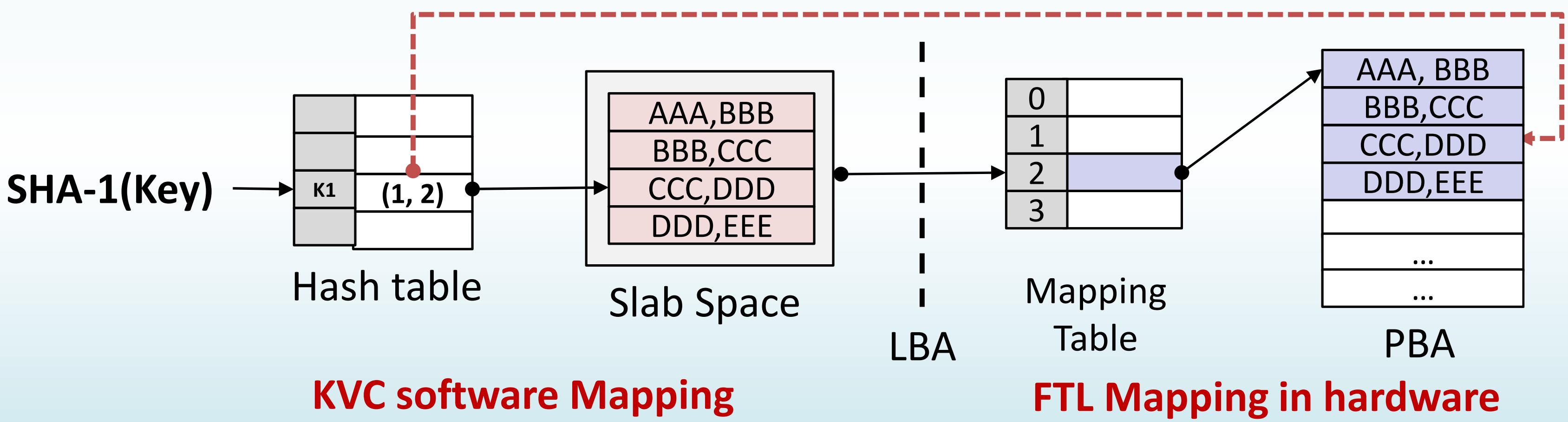
- In-memory hash map to track key-to-value mapping
- Slabs are used in a log-structured way
- Updated value item written to a new location and old values recycled later

Critical Issues

- Redundant mapping
- Double garbage collection
- Over-over-provisioning

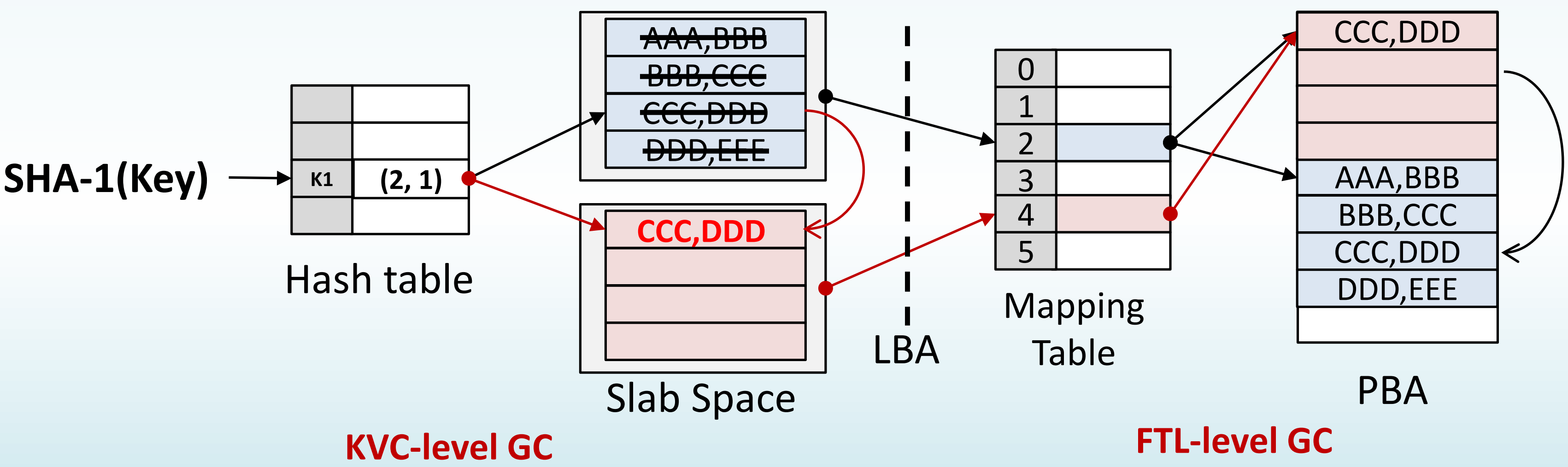
Critical Issue 1: Redundant Mapping

- **Redundant mapping** at application- and FTL-level
  - KVC: An in-memory hash table (Key → Slab, Offset)
  - FTL: An on-device page mapping table (LBA → PBA)
- **Problems**
  - Two mapping structures (unnecessarily) co-exist at two levels
  - A significant waste of on-device DRAM space (e.g., 1GB for 1TB)
    - The on-device DRAM buffer is costly, unreliable, and could be used for buffering.



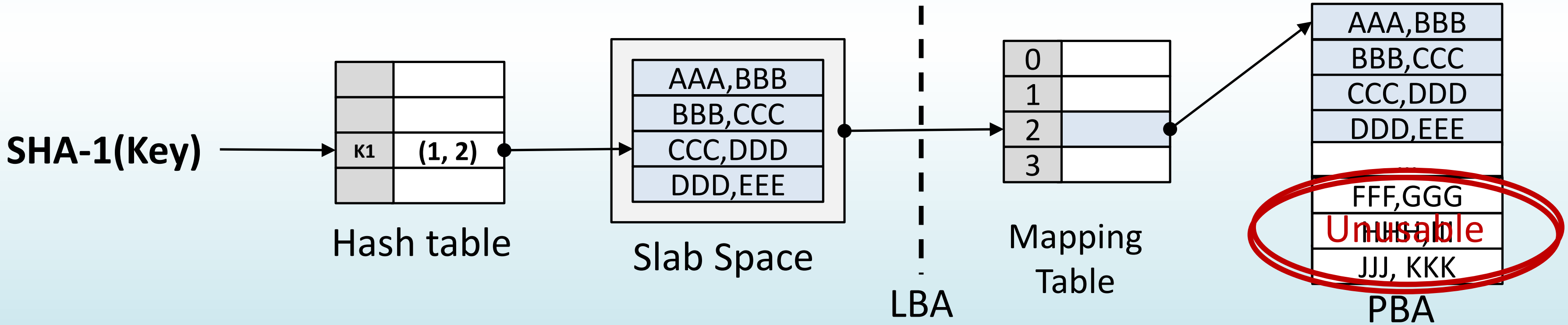
Critical Issue 2: Double Garbage Collection

- **Garbage collection** (GC) at app- and FTL- levels
  - KVC: Recycle deleted or changed key-value items
  - FTL: Recycle trimmed or changed pages
- **Problems**
  - Semantic validity of a key-value entry is not seen at FTL
  - Redundant data copy operation



Critical Issue 3: Over-over-provisioning

- **Over-provisioning** at FTL-level
  - FTL has a portion (20-30%) of flash as Over-Provisioning Space (OPS)
  - OPS space is invisible and **unusable** to the host applications
- **Problems**
  - OPS is reserved for dealing the worst-case situation, as a storage
  - Over-over provisioning for Key-value caches
    - Key-value caches are dominated by read (GET) traffic, not writes
  - Key-value cache hit ratio is highly sensitive to usable cache size
    - If 20-30% space can be released, the cache hit ratio can be greatly improved



Semantic Gap Problem

Key-value cache | **Flash SSD**

- Fine-grained GC
- Key-to-value mapping
- Validity of slab entries

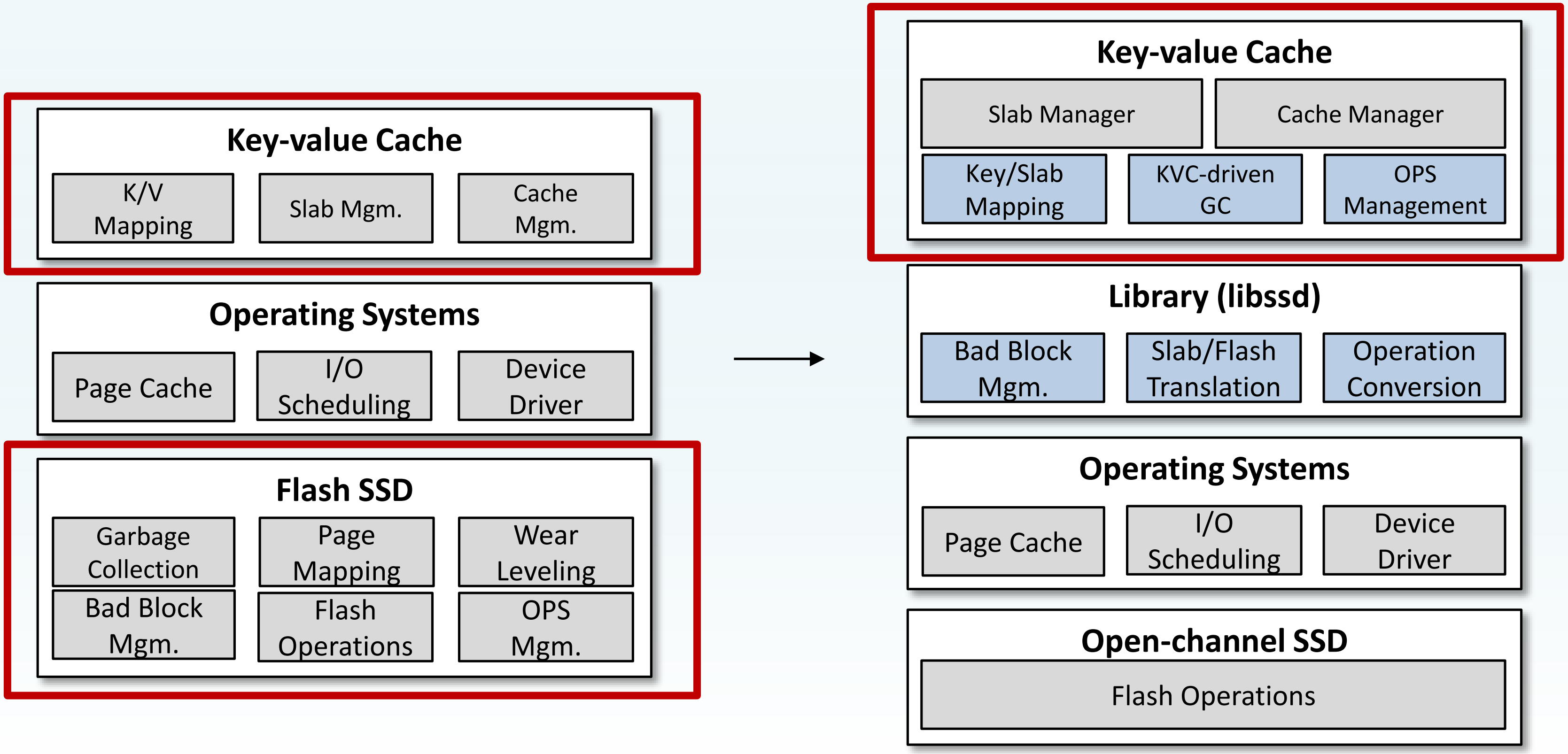
- Physical data layout on flash
- Direct flash memory control
- Proper mapping granularity

In the current SW/HW architecture, we can do little to address these three issues.

Optimization Goals

- Redundant mapping → **Single mapping**
- Double garbage collection → **App-driven GC**
- Over-over-provisioning → **Dynamic OPS**

Design Overview



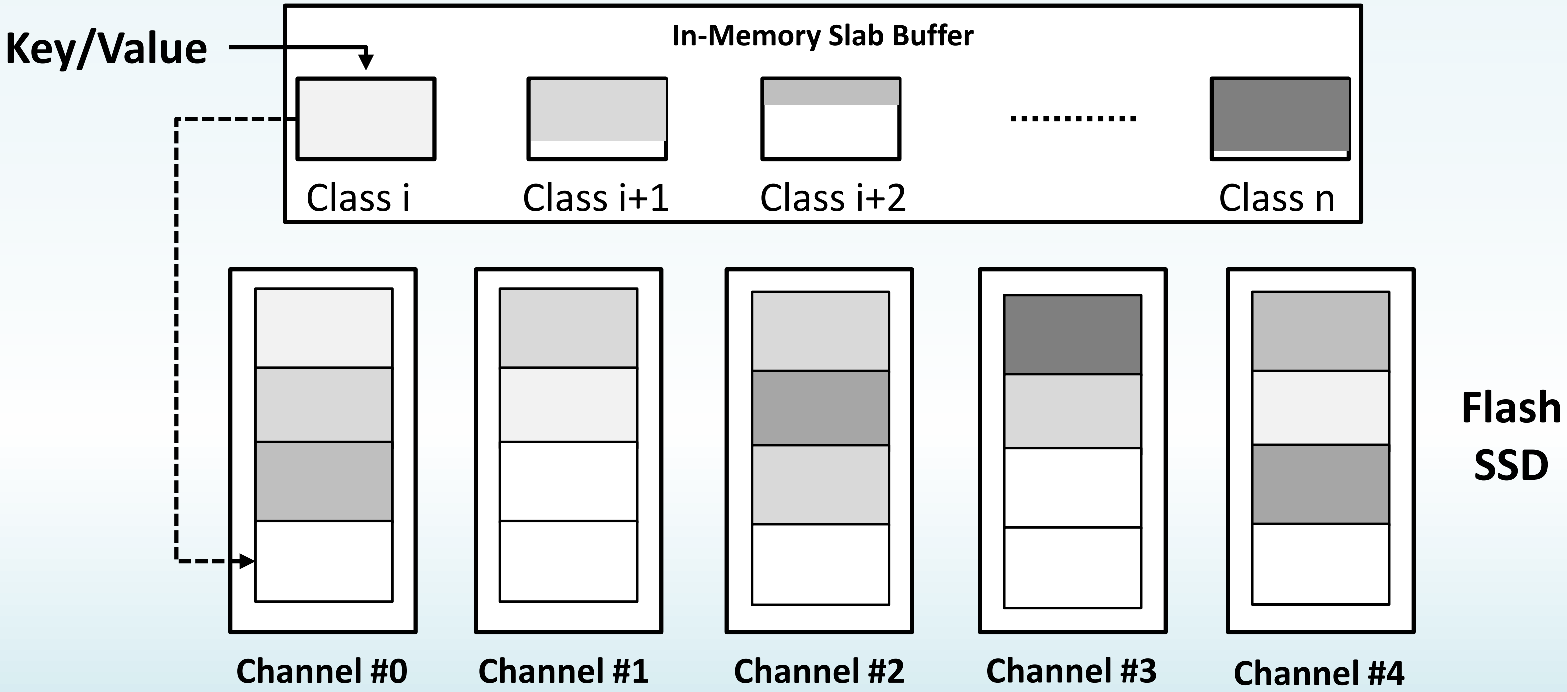
- An enhanced flash-aware key-value cache
- A thin intermediate library layer (`libssd`)
- A specialized flash memory PCI-E SSD hardware

An Enhanced Flash-aware Key-value Cache

- Slab management
- Unified direct mapping
- Garbage collection
- OPS management

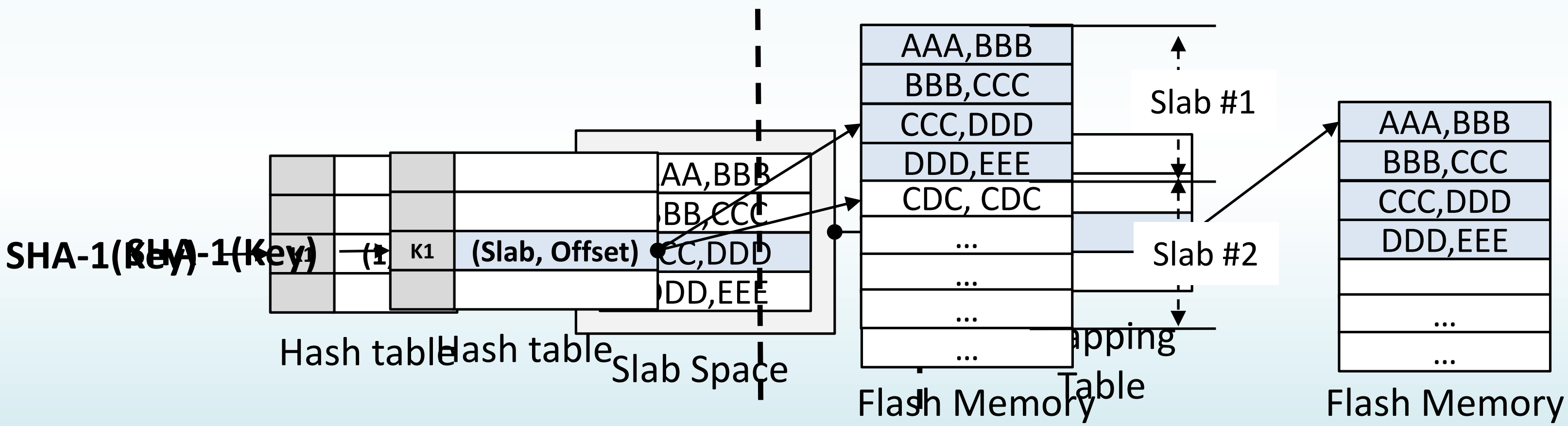
Slab Management

- Our choice – Directly use a flash block as a slab (8MB)
- **Slab buffer**: An in-memory slab maintained for each class
  - Parallelize and **asynchronize** the slab write I/Os to the flash memory
- Round-robin allocation of in-flash slab for load-balancing across channels



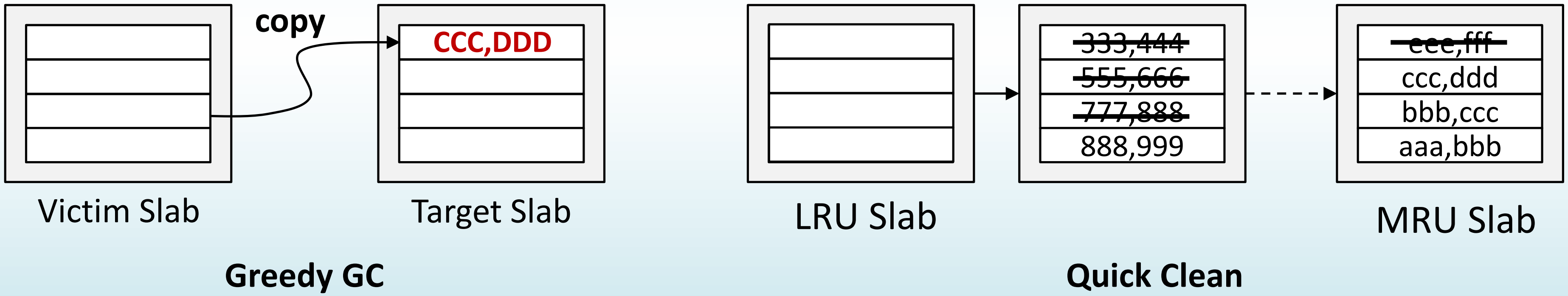
Unified Direct Mapping

- Collapse multiple levels of indirect mapping to only one
  - Prior mapping: Key→Slab→LBA→PBA
  - Current mapping: Key→Slab (i.e., Flash Block)
- **Benefits**
  - Removes the time overhead for lookup intermediate layers (Speed+)
  - Only one single must-have in-memory hash table is needed (Cost-)
  - On-device RAM space can be completely removed (or for other uses)



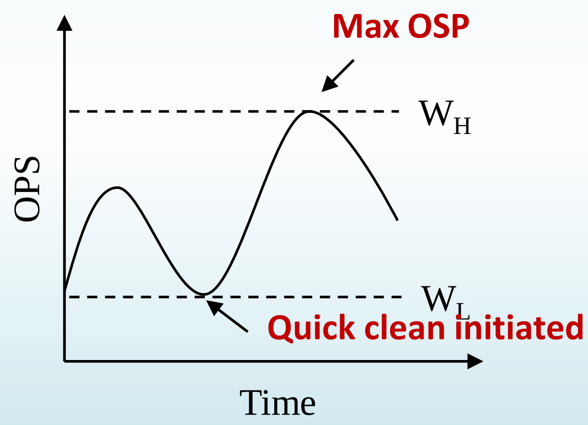
App-driven Garbage Collection

- One single GC is driven directly by key-value cache system
  - All slab writes are in units of blocks (no need for device-level GC)
  - GC is directly triggered and controlled by application-level KVC
- Two GC policies
  - **Greedy GC**: the **least occupied** slab is evicted to move minimum slots
  - **Quick clean**: the **LRU** slab is immediately dropped recycling valid slots
  - Adaptively used for different circumstances (busy or idle)



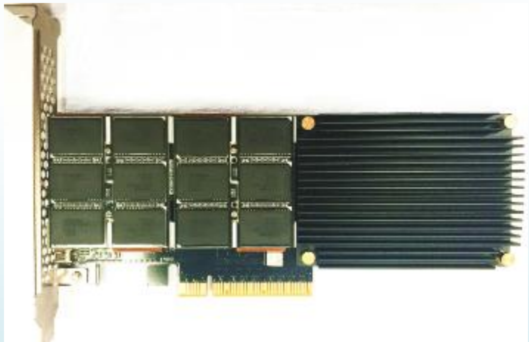
Over-Provisioning Space Management

- Dynamically tuning OPS space online
  - **Rationale** – KVC is typically read-intensive and OPS can be small
  - **Goal** – keep just enough OPS space (adaptive to intensity of writes)
- OPS management policies
  - **Heuristic method**: An OPS window (W_L and W_H) to estimate size
    - Low watermark hit – Trigger quick clean, raise the OPS window
    - High watermark hit – OPS is over-allocated, lower the OPS window

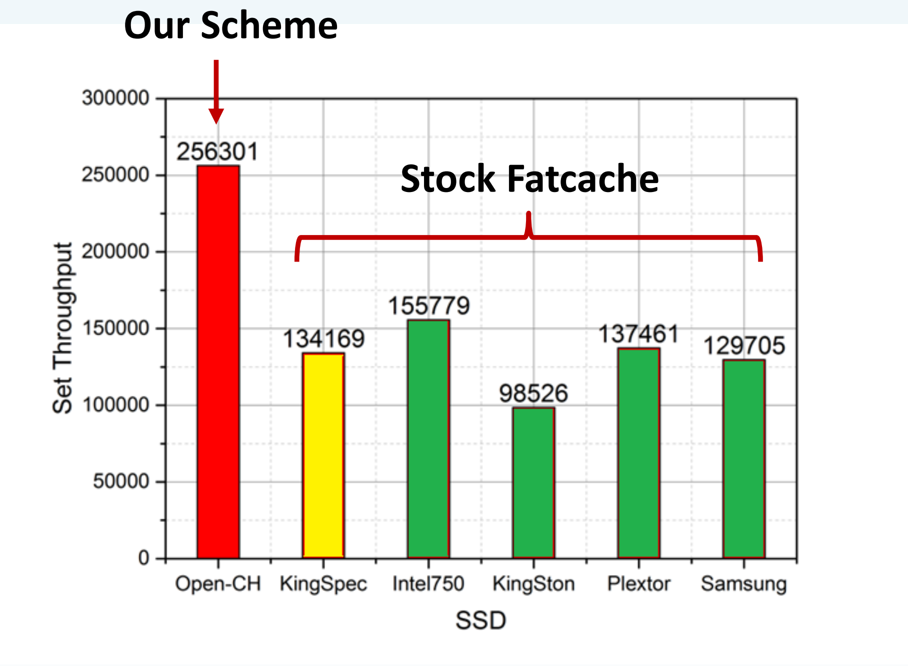


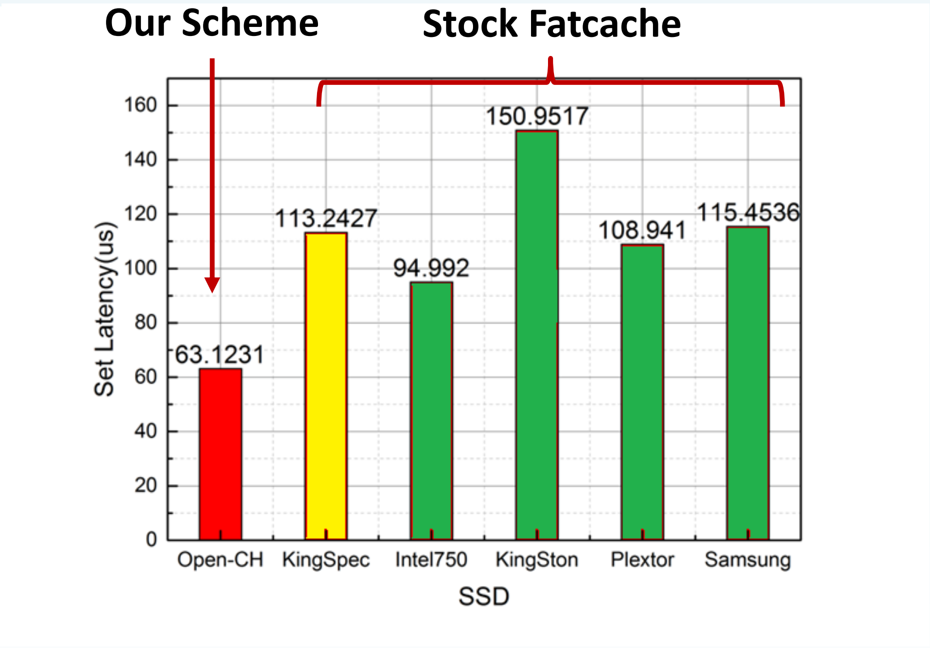
Preliminary Experiments

- **Implementation**
  - Key-value cache on Twitter's Fatcache to fit hardward
  - `Libssd` Library (621 lines of code in C)
- **Experimental Setup**
  - Intel Xeon E-1225, 32GB Memory, 1TB Disk, Open-Channel SSD
  - Ubuntu 14.04 LTS, Linux 3.17.8, Ext4 filesystem
- **Hardware: Open-channel SSD**
  - A PCI-E based with 12 channel, and 192 LUNs
  - Direct control to the device (via `ioctl` interface)



SET – Throughput and Latency





- **SET Workloads:** 40milliom requests of 1KB key/value pairs
- **Both set throughput/latency from our scheme are the best**

Conclusion

- KV stores become critical as they are one of the most important building blocks in modern web infrastructures and high-performance data-intensive applications.
- We build a highly efficient flash-based cache system which enables three benefits, namely a unified single-level direct mapping, a cache-driven fine-grained garbage collection, and an adaptive over-provisioning scheme
- We are implementing a prototype on the Open-Channel SSD hardware and our preliminary results show that it is highly promising

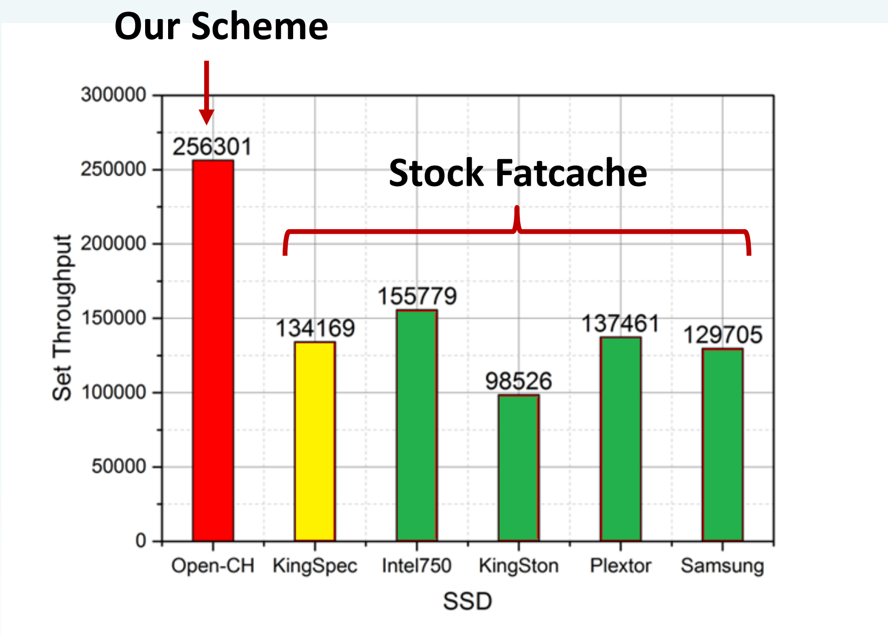
THE HONG KONG POLYTECHNIC UNIVERSITY
香港理工大學

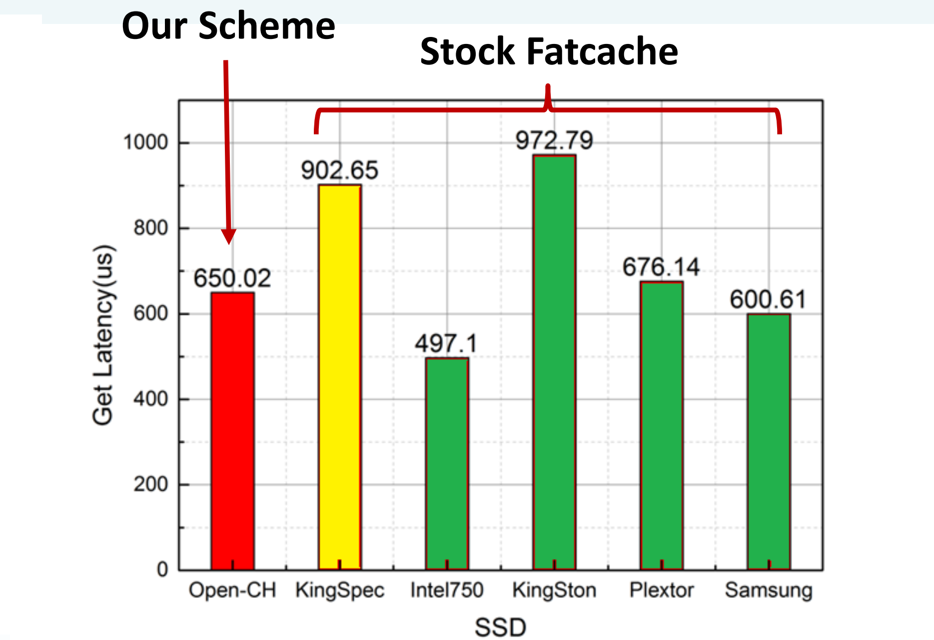
LSU
LOUISIANA STATE UNIVERSITY

Thank You !

Backup Slides

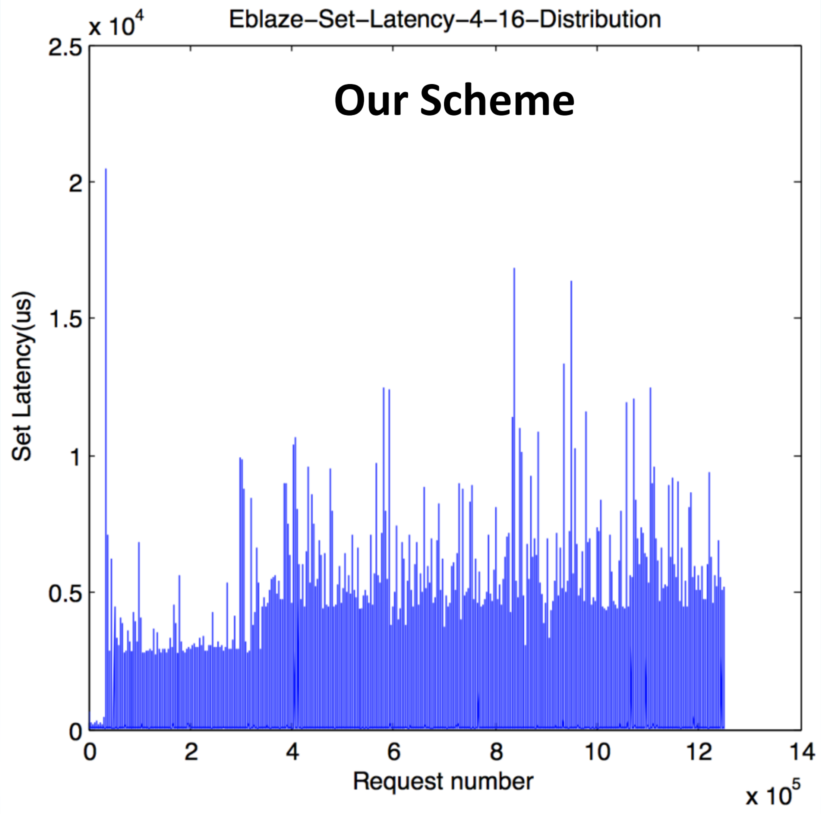
GET – Throughput and Latency

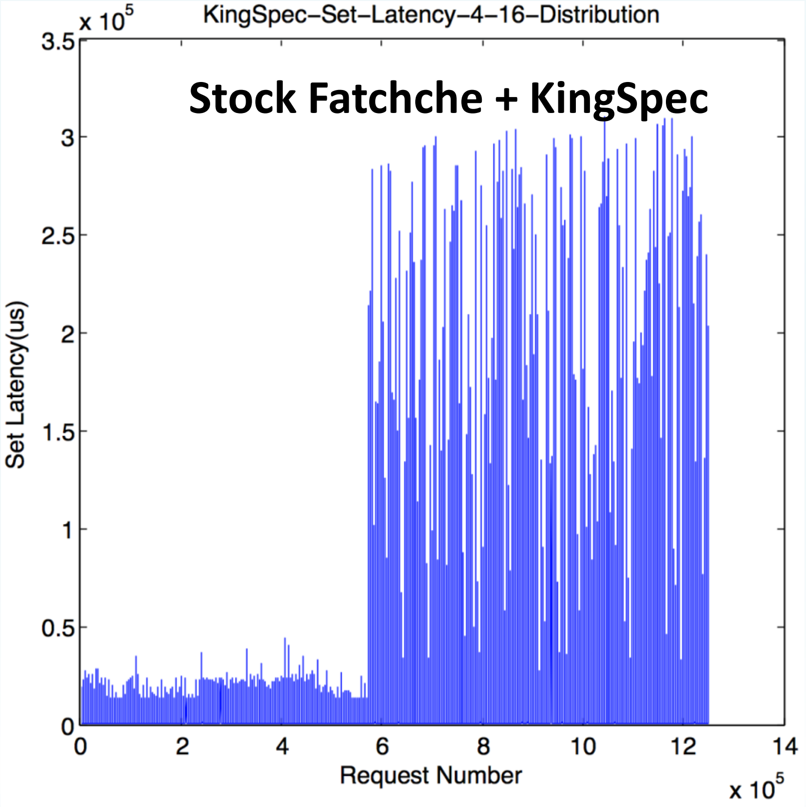




- GET performance is largely determined by the raw speed
- GET latencies are among the lowest in the set of SSDs

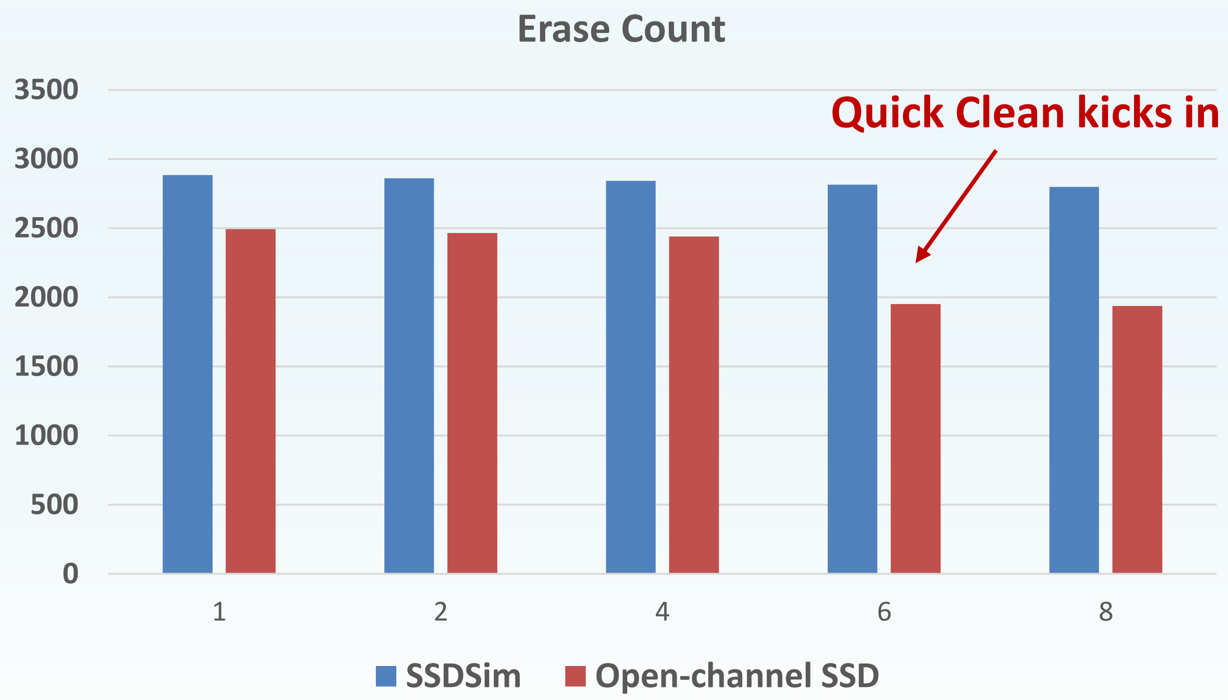
SET Latencies – A Zoom-in View





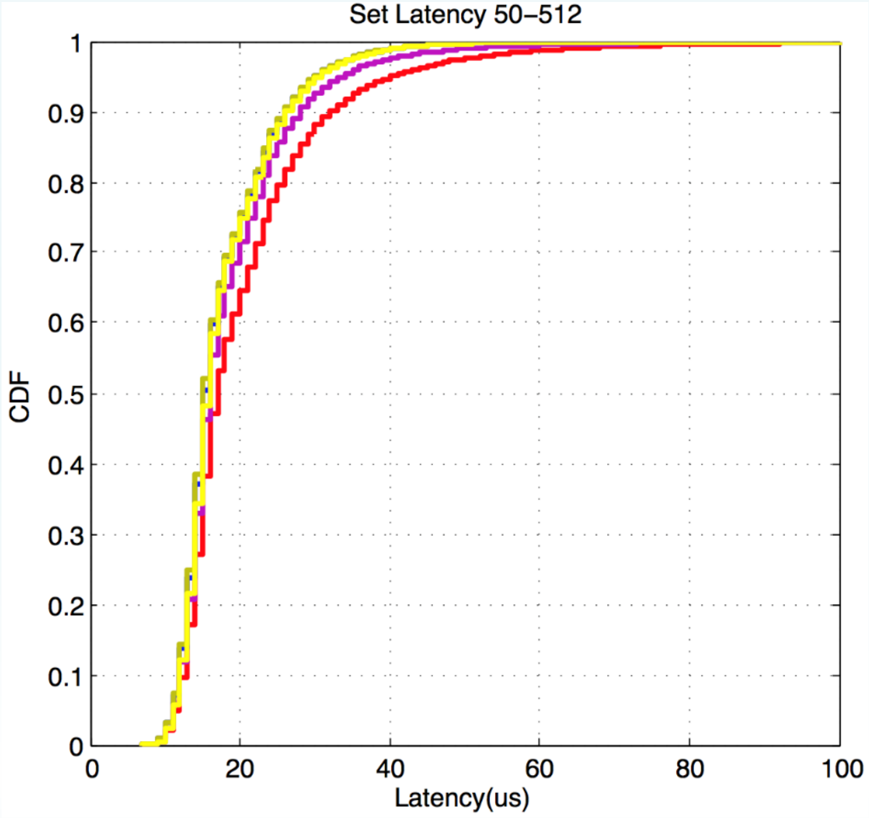
- The direct control successfully removes high cost GC effects
- Quick clean removes long I/Os under high write pressure

Block Erase Count



- Trace collected with running Fatcache on Samsung SSD
- Block trace is replayed on MSR's SSD simulator for erase count
- Our solution reduces erase count by 30%

Effect of the In-memory Slab Buffer



- 10x buffer size difference does not affect latency significantly
- A small (56MB) in-memory slab buffer is sufficient for use