NVMKV: A Scalable and Lightweight Flash Aware Key-Value Store

Leonardo Mármol‡, Swaminathan Sundararaman†, Nisha Talagala†, Raju Rangaswami‡*

† *SanDisk* ‡ *Florida International University* **work done at SanDisk*

FIU FLORIDA INTERNATIONAL UNIVERSITY

SanDisk

Introducing KV Stores

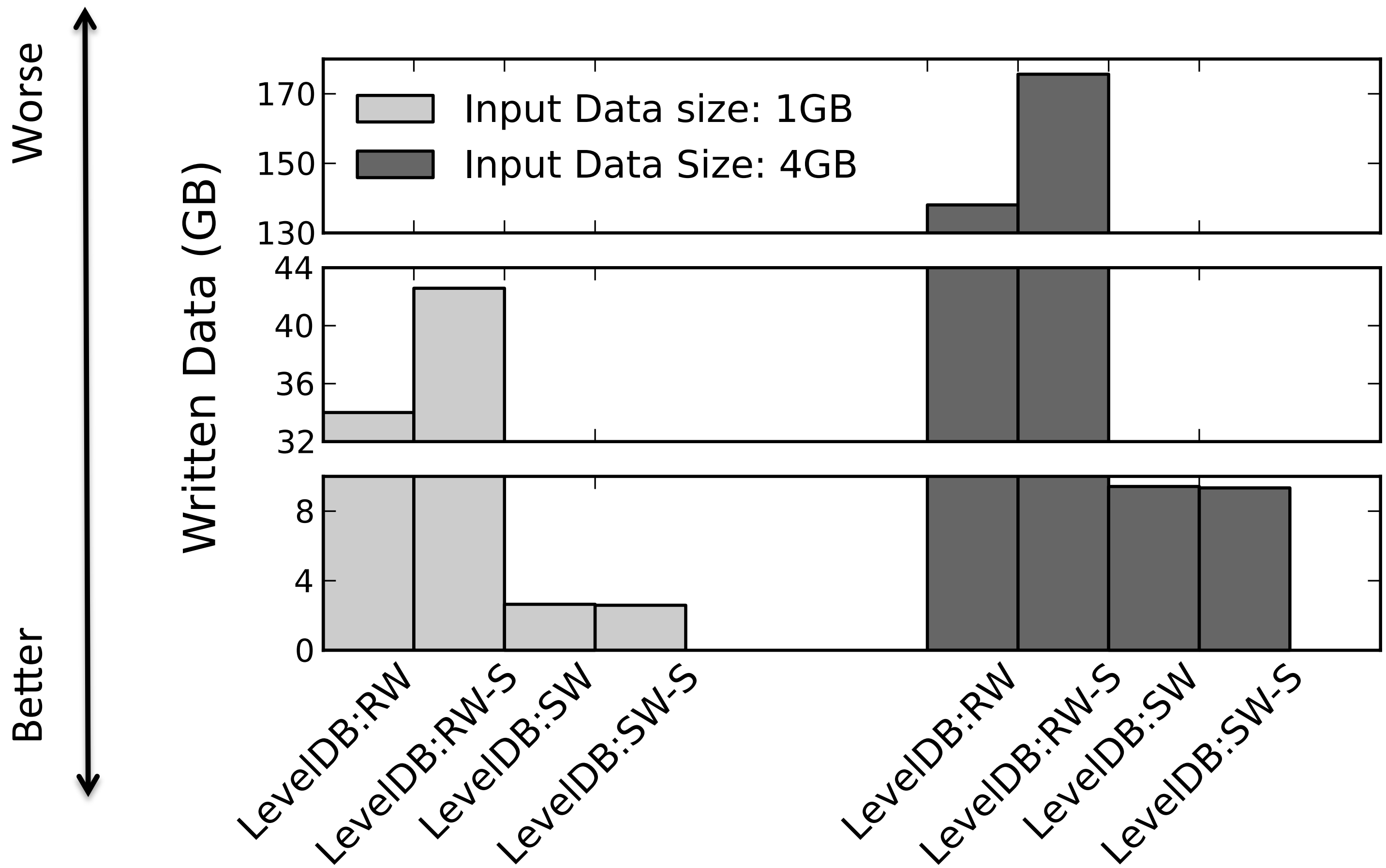
- Preferred data management solution for Internet services
- Provide simple interface
  - get, put, delete
- Provide weaker consistency model
  - Compared to RDMS
  - Tradeoff between performance and consistency

Limitations of existing solutions*

- Use log-structured/out-of-place updates
  - Better performance on hard-disk and older SSD's
- Require compaction/garbage collection
- Creates auxiliary write amplification
  - Performance penalty
  - Reduces the life of NAND flash

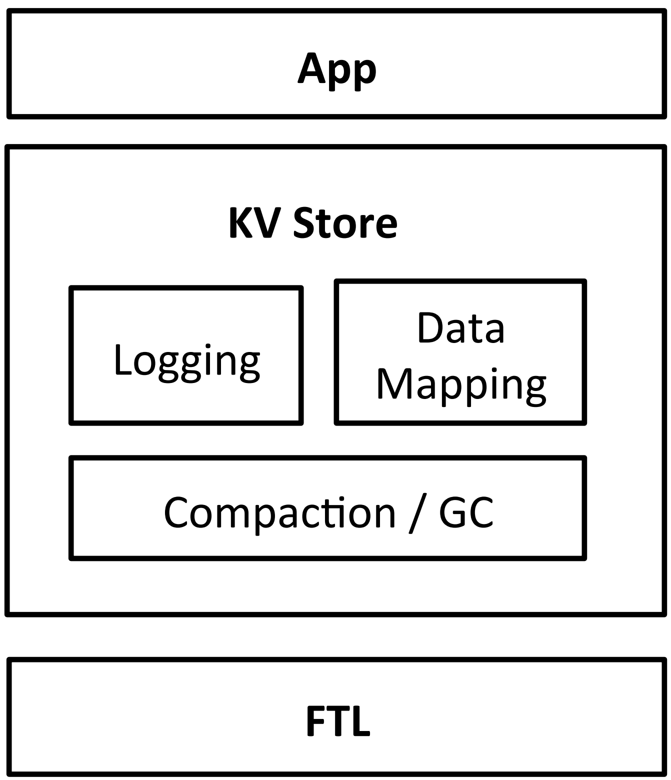
*Existing Key-Value Store optimized for FLASH storage

How bad is the situation?

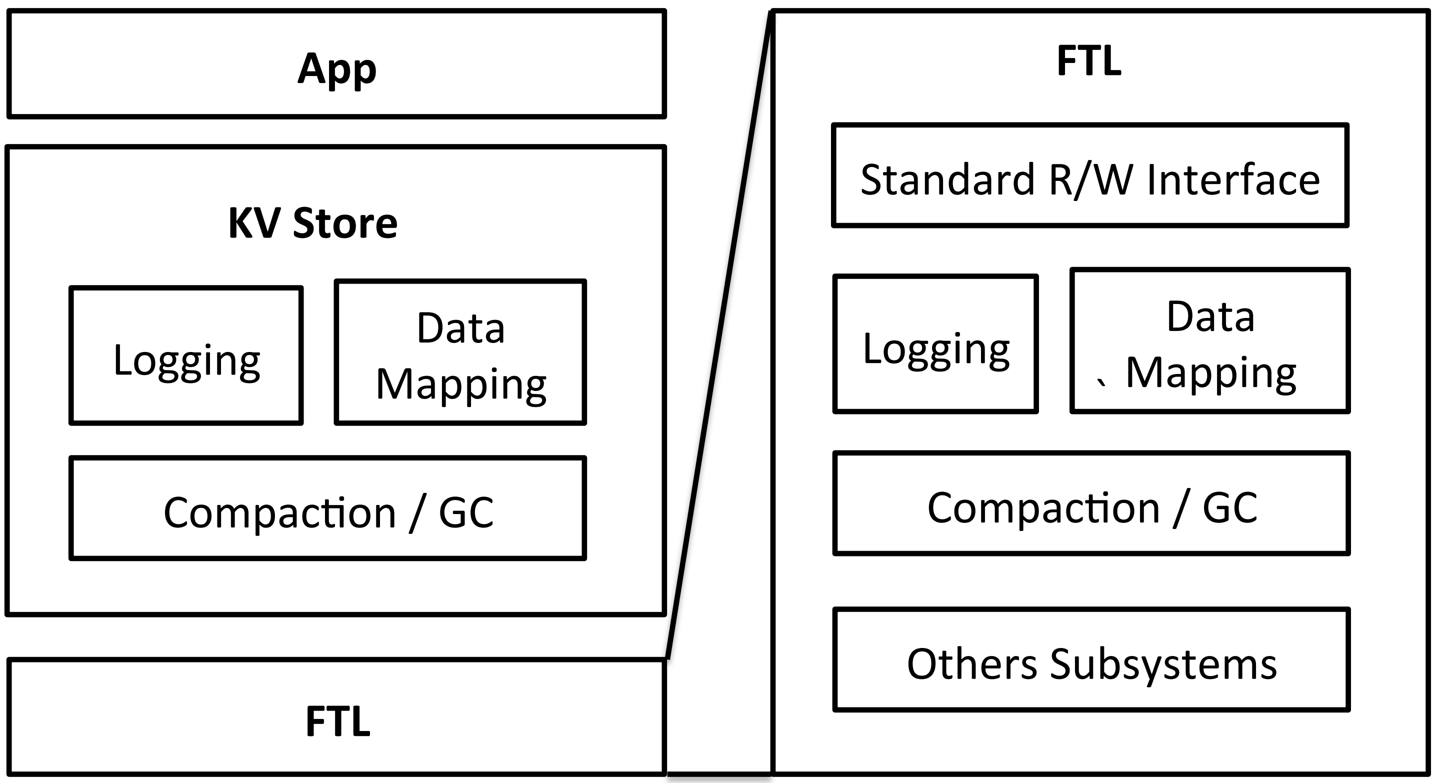


- Auxiliary write amplification varying from 2.5x to 43x
- “Don’t Stack your Log on my Log” by Yang et al. (INFLOW’14)

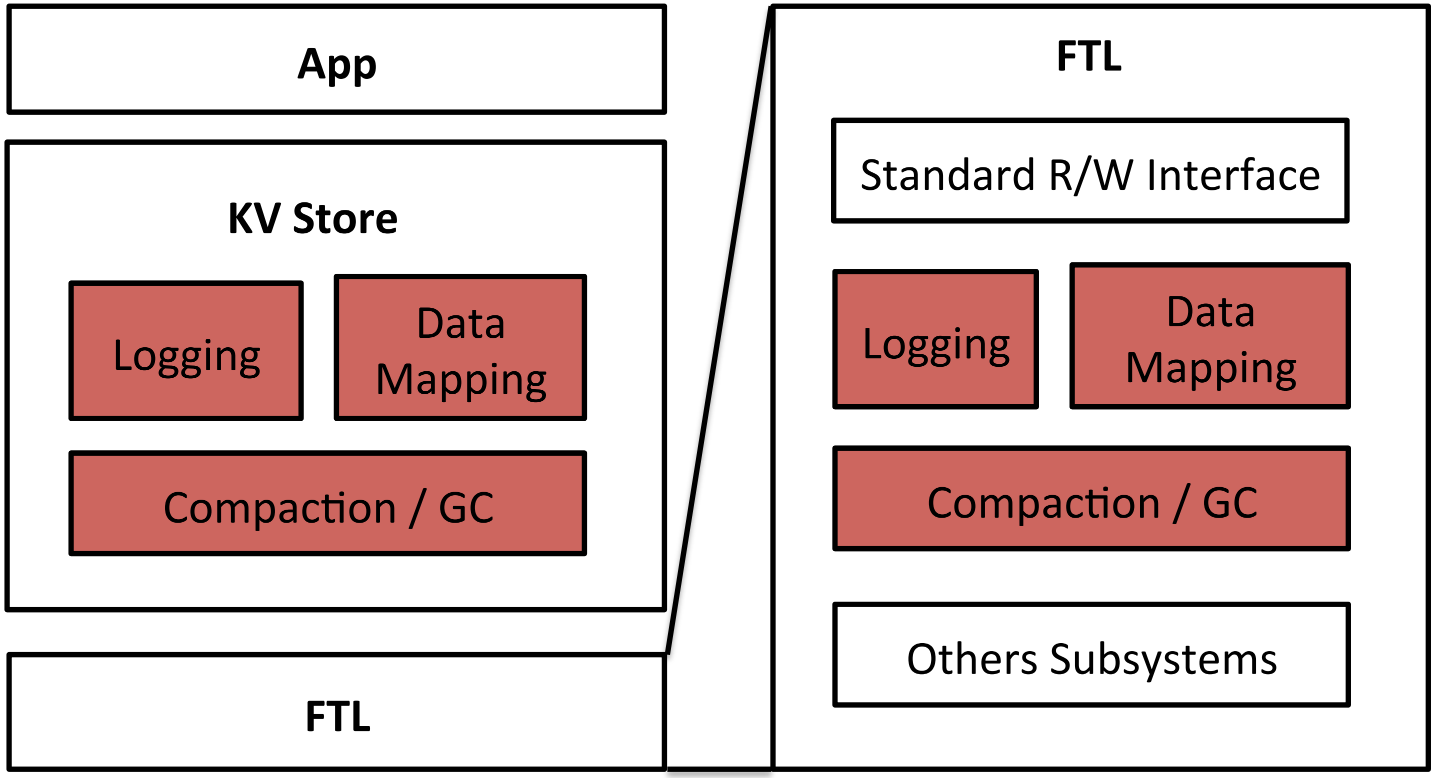
Existing KV Store Designs



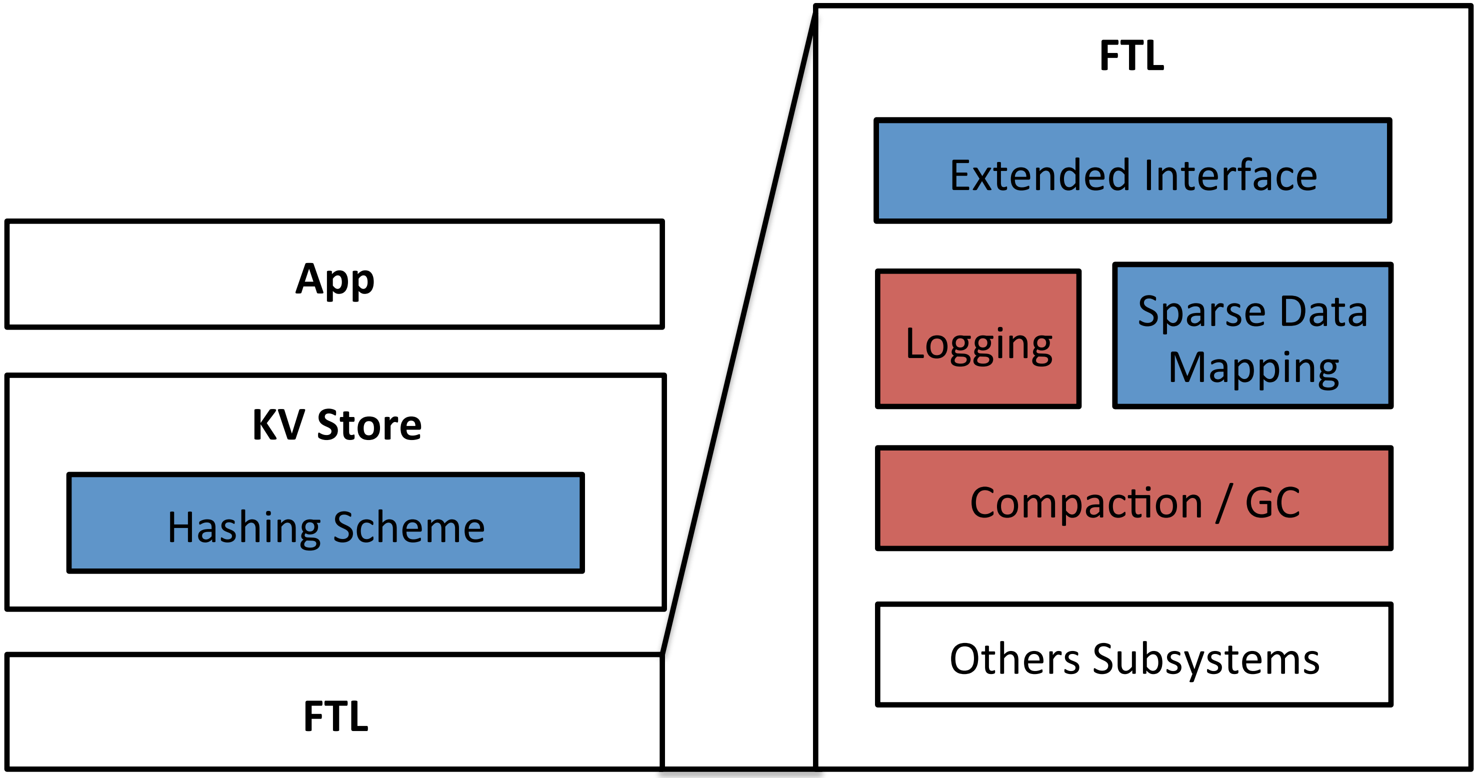
Existing KV Store Designs



Existing KV Store Designs

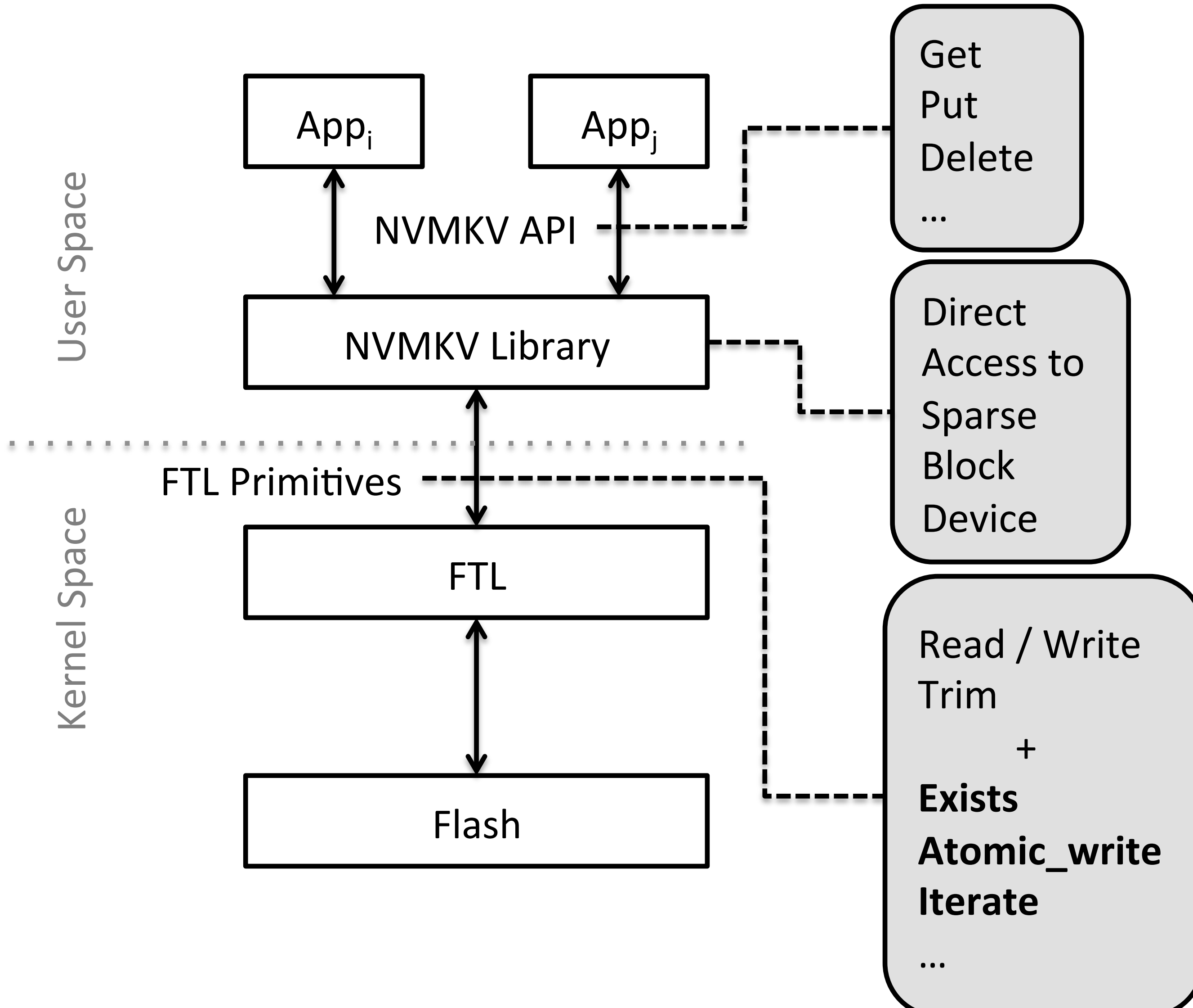


FTL Cooperative KV Store Design



Design

NVMKV System Architecture



NVMKV API

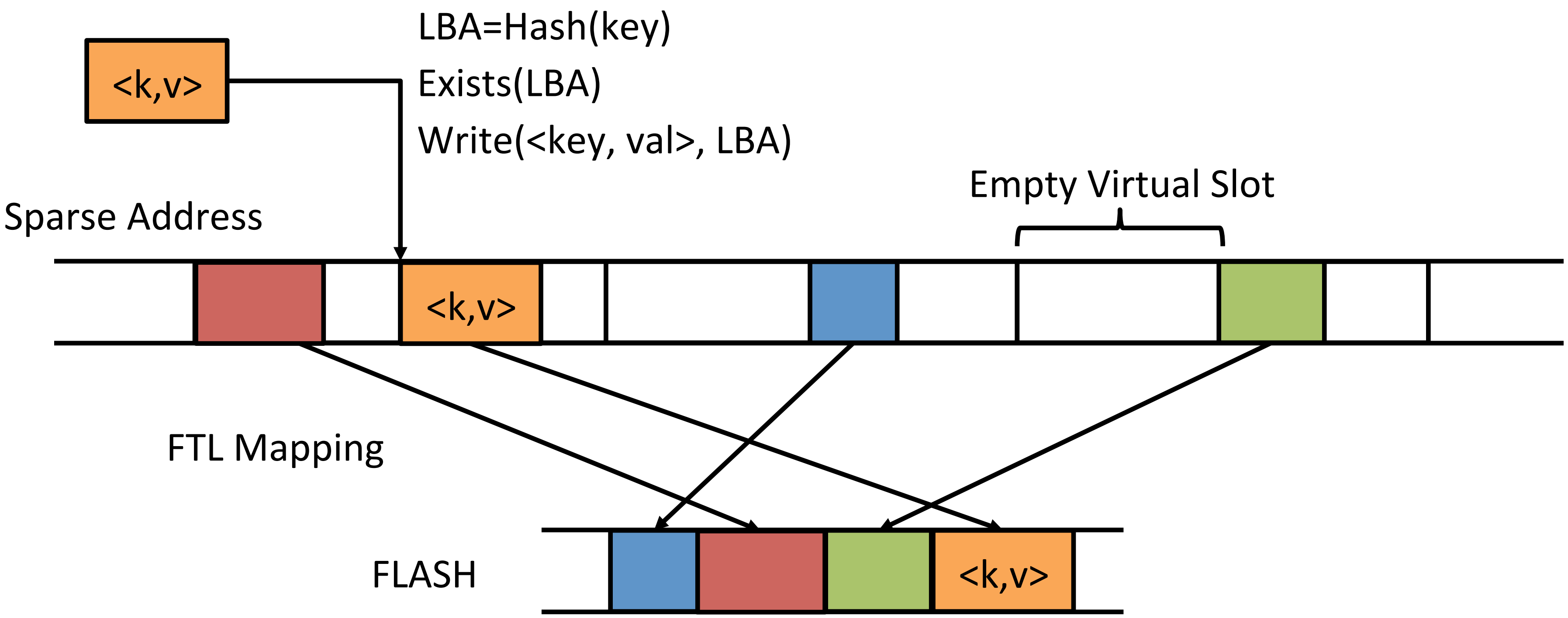
- Management
  - Open / Create
  - Close / Destroy
- Basic operations
  - Get, Put, Delete
- Batch operations
  - Batch-Get, Batch-Put
  - Batch-Delete
  - Delete-all

Code Fragment

```
fd = open(dev_name,…);
id = nvmkv_open(fd,…);
…
nvmkv_put(id,k1,v1,…);
…
nvmkv_exists(id,k2,…);
…
nvmkv_get(id,k2,…);
…
nvmkv_close(id);
close(fd)
```

- A full description of the API can be found in the paper
- “Beyond Block I/O: Rethinking Traditional Storage Primitives” by Ouyang et al.

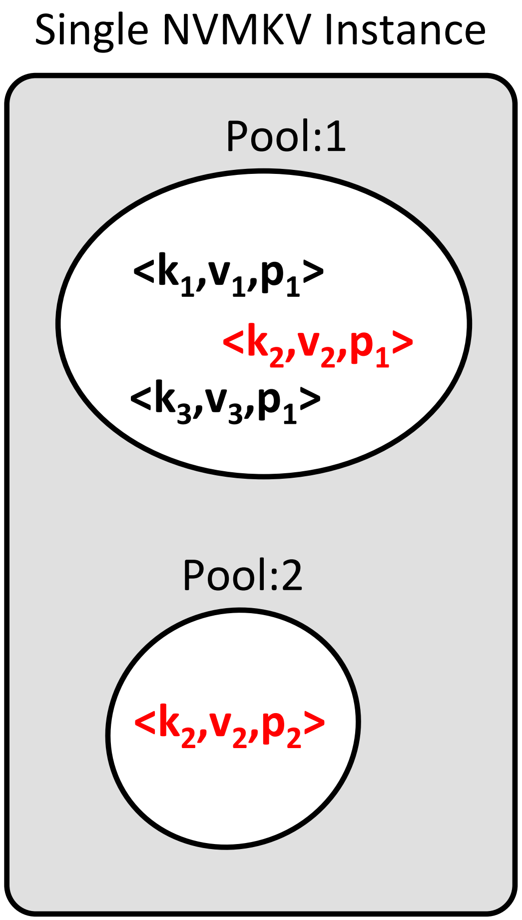
I/O on a Sparse Block Device



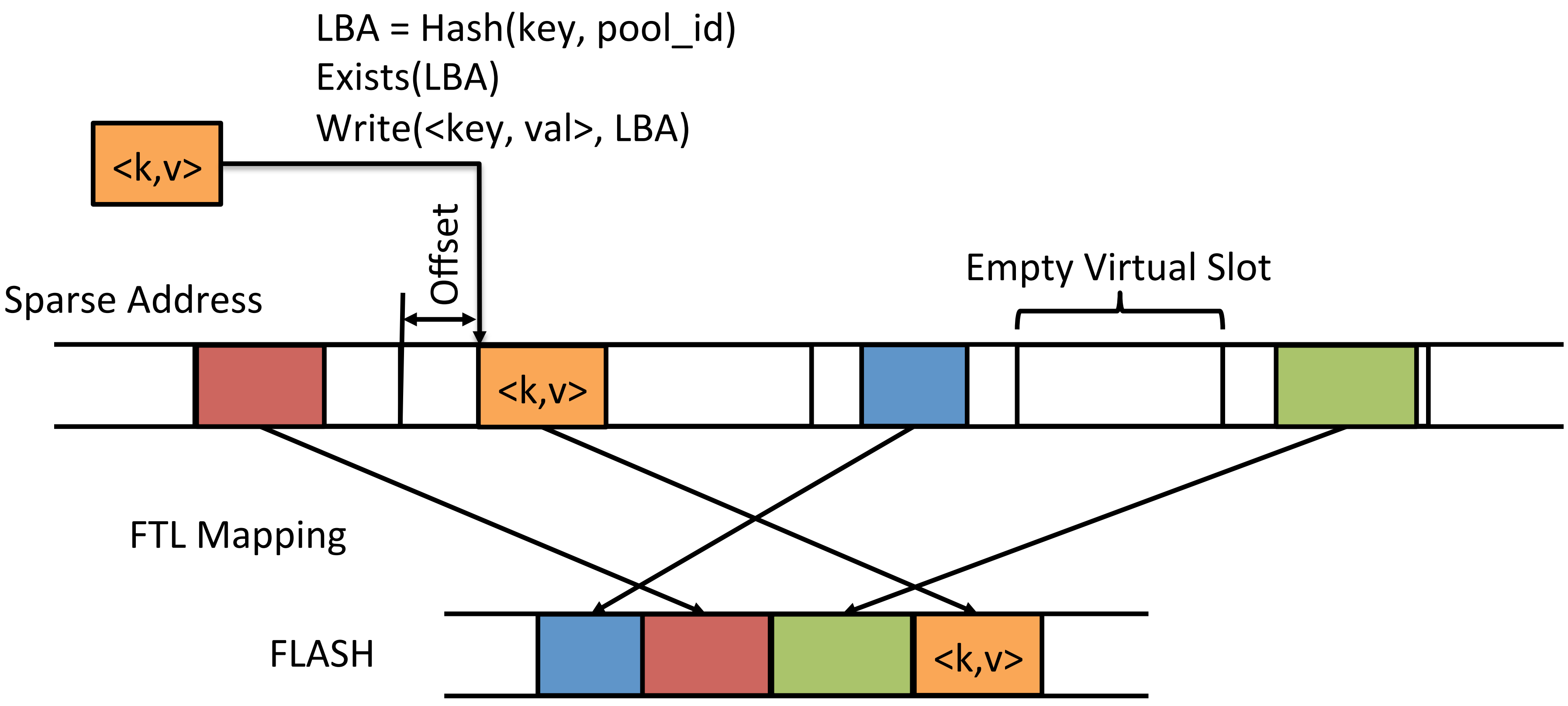
- Exists(LBA) does not perform I/O
- New LBA(s) are computed using polynomial probing
- The size of the virtual slots can be configured

Multiple NVMKV On Single Device

- Pool: Set of unique key-value pairs that shared the same pool id.
  - Part of hashing scheme
    - LBA = Hash(key, pool_id)
  - No in-memory metadata
  - No extra I/O operations
  - Now operations required pool_id, in addition to key & value



Inserting KV-Pairs into Pools



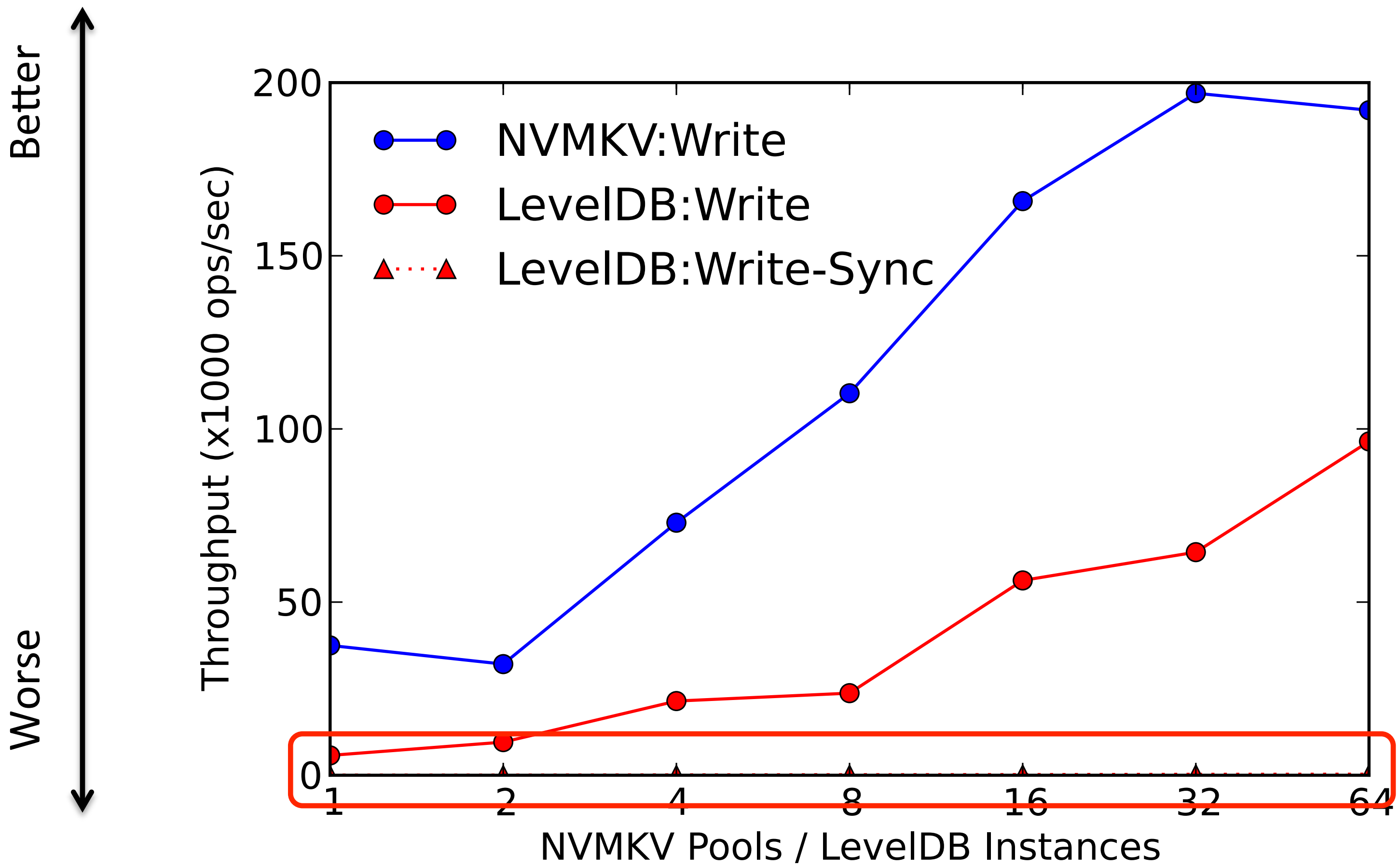
- Only one <key,val> per virtual slot
- Offset within the virtual slot determines the pool_id
- Offsets in virtual address does not creates wasted space in flash

Improving Performance by Caching

- **Read-only Cache**: Improves read performance.
- **Collision Cache**: Prevents multiple I/Os when cache collision occurs.
- For more details, please consult the paper.

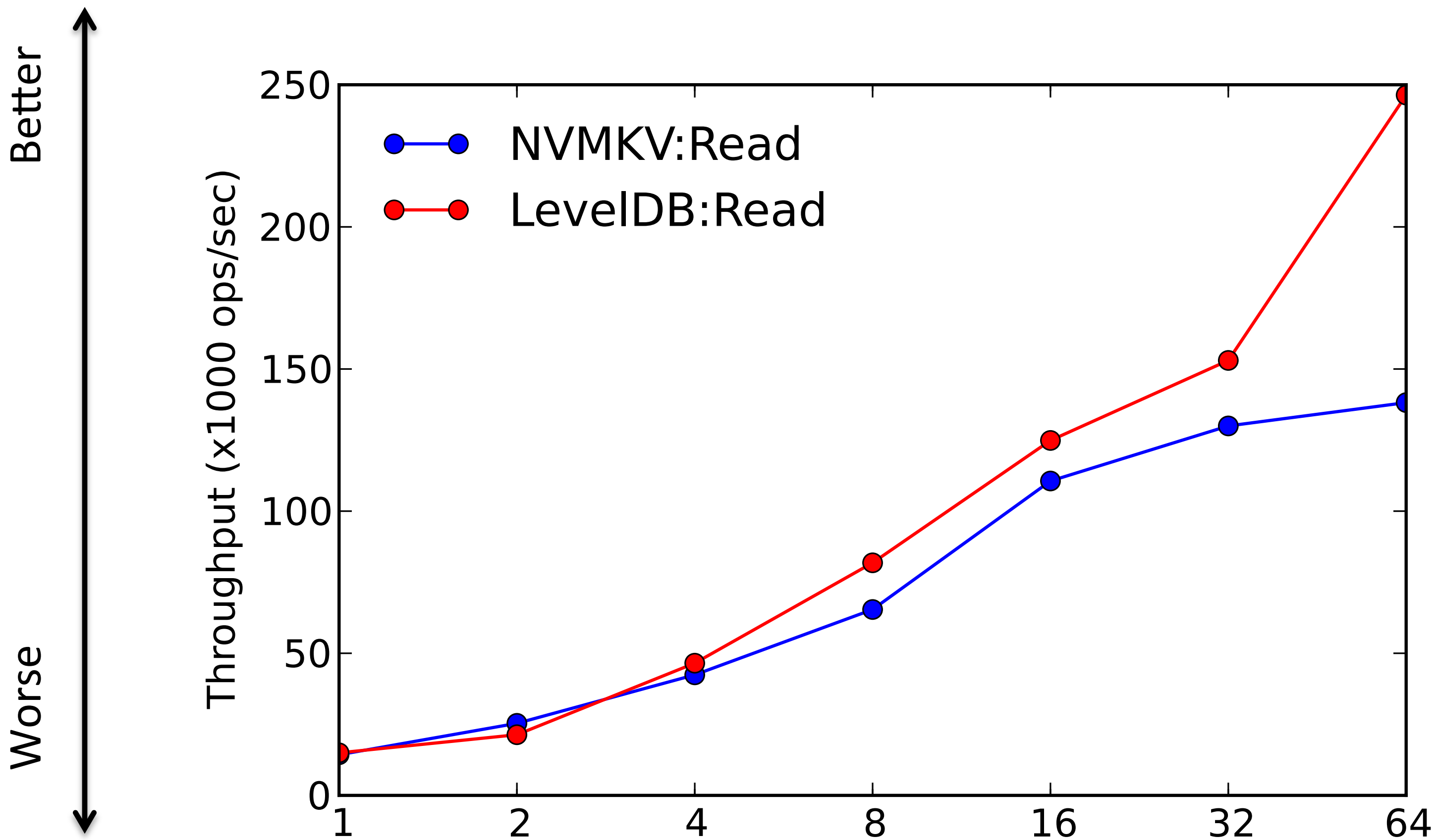
Evaluation Results

NVMKV Performance



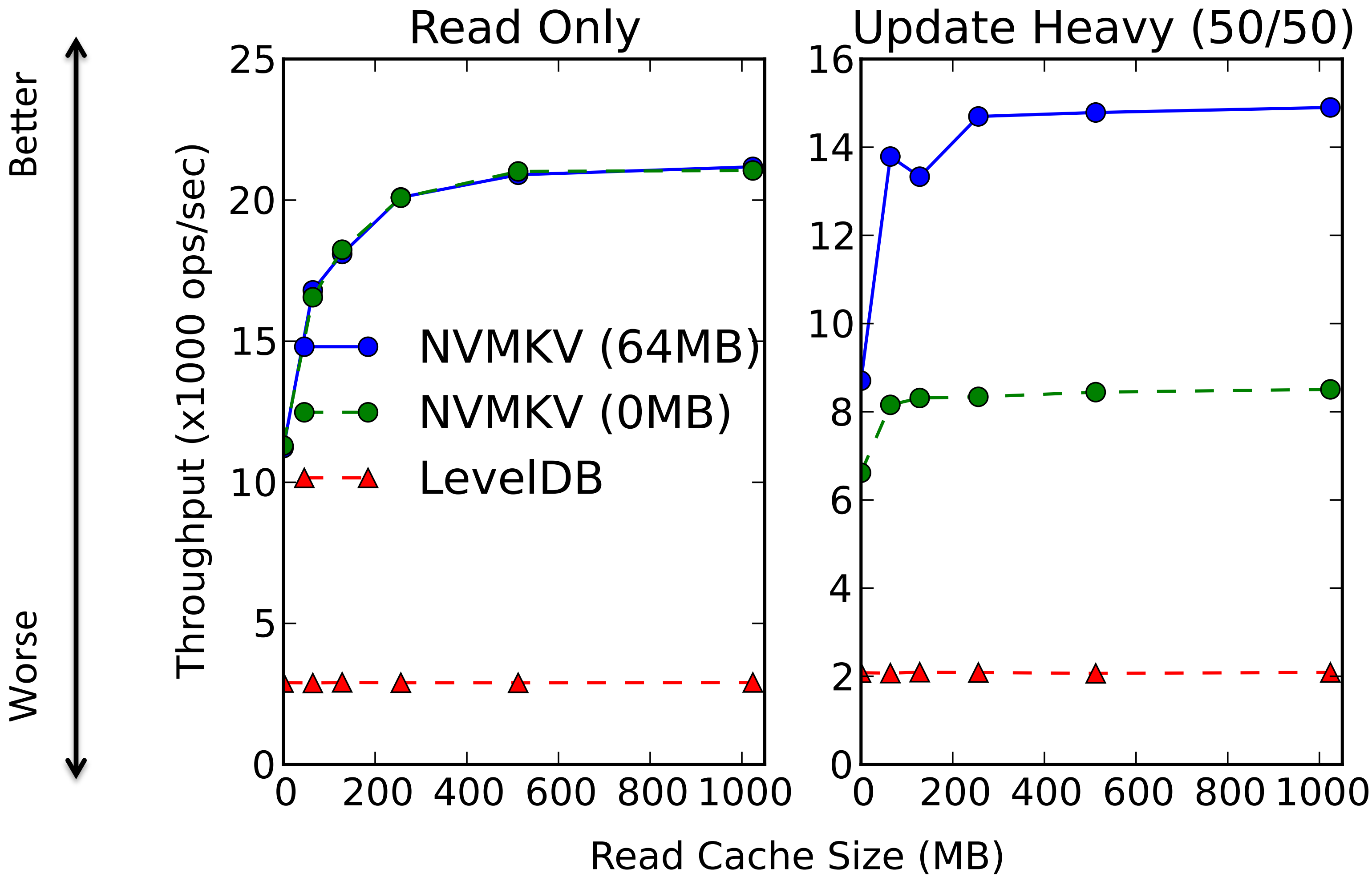
- NVMKV outperform LevelDB even when LevelDB uses async writes
- NVMKV writes are always synchronous (no buffering)

NVMKV Performance



- LevelDB benefits from both its own cache and the file system cache (warmed)
- NVMKV perform competitively up to 32 pools even without any caching

YCSB Benchmark



Summary

- We propose a FTL cooperative design that allows for:
  - Simple KV-Store design and implementation
  - Constant amount of metadata
  - Multiple KV-Stores sharing the same FLASH
  - High performance / parallelism
  - Atomicity and durability of KV operations
  - Low write amplification
    - Reduced write amplification by up to 29x compared to LevelDB

Fork me on GitHub

Thank you!

opennvm.github.io

FLORIDA INTERNATIONAL UNIVERSITY

SanDisk