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

Leonardo Mármol‡, Swaminathan Sundararaman†,
Nisha Talagala†, Raju Rangaswami‡, Sushma Devendrappa∗,
Bharath Ramsundar∗, Sriram Ganesan∗

† *FusionIO* ‡ *Florida International University* ∗*work done at FusionIO*

FLORIDA INTERNATIONAL UNIVERSITY

FUSION-IO

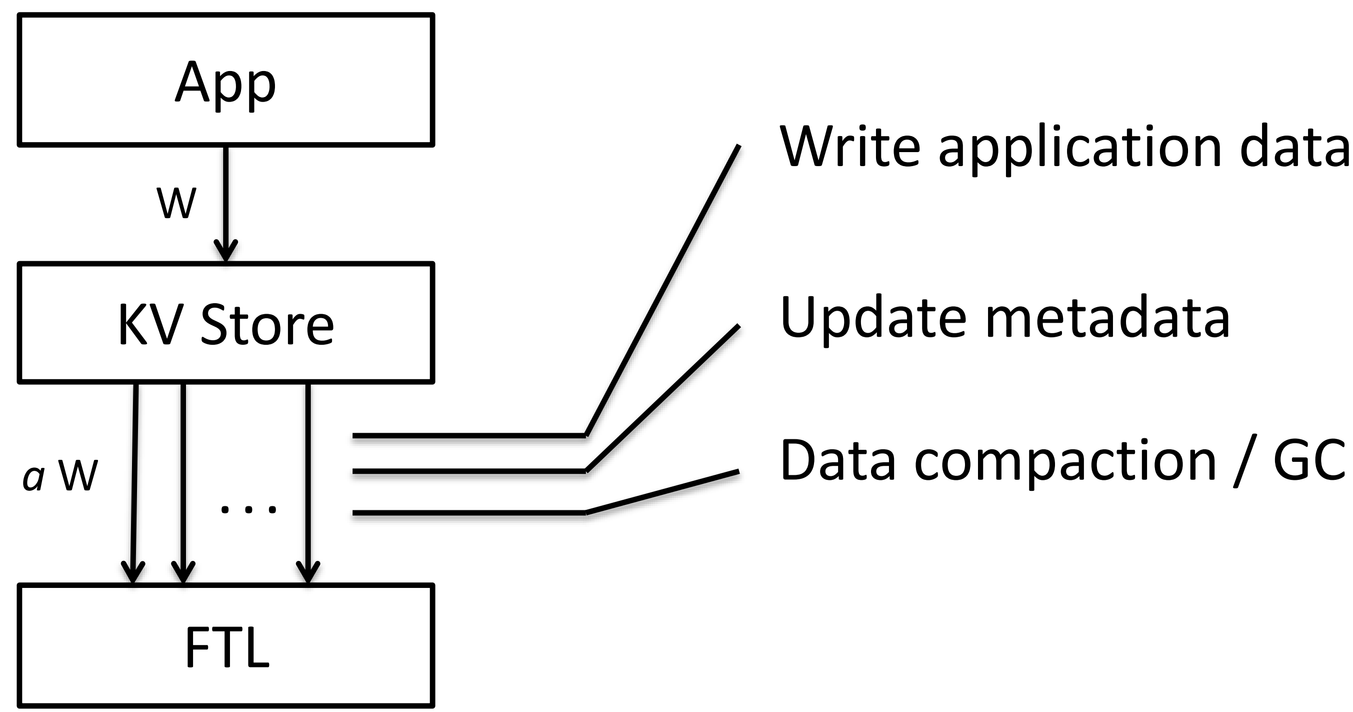
Introducing KV Stores

- Preferred data management solution for Internet services
- Provide simple interface
  - get, put, delete
- Provide weaker consistency model
  - compared to RDMS

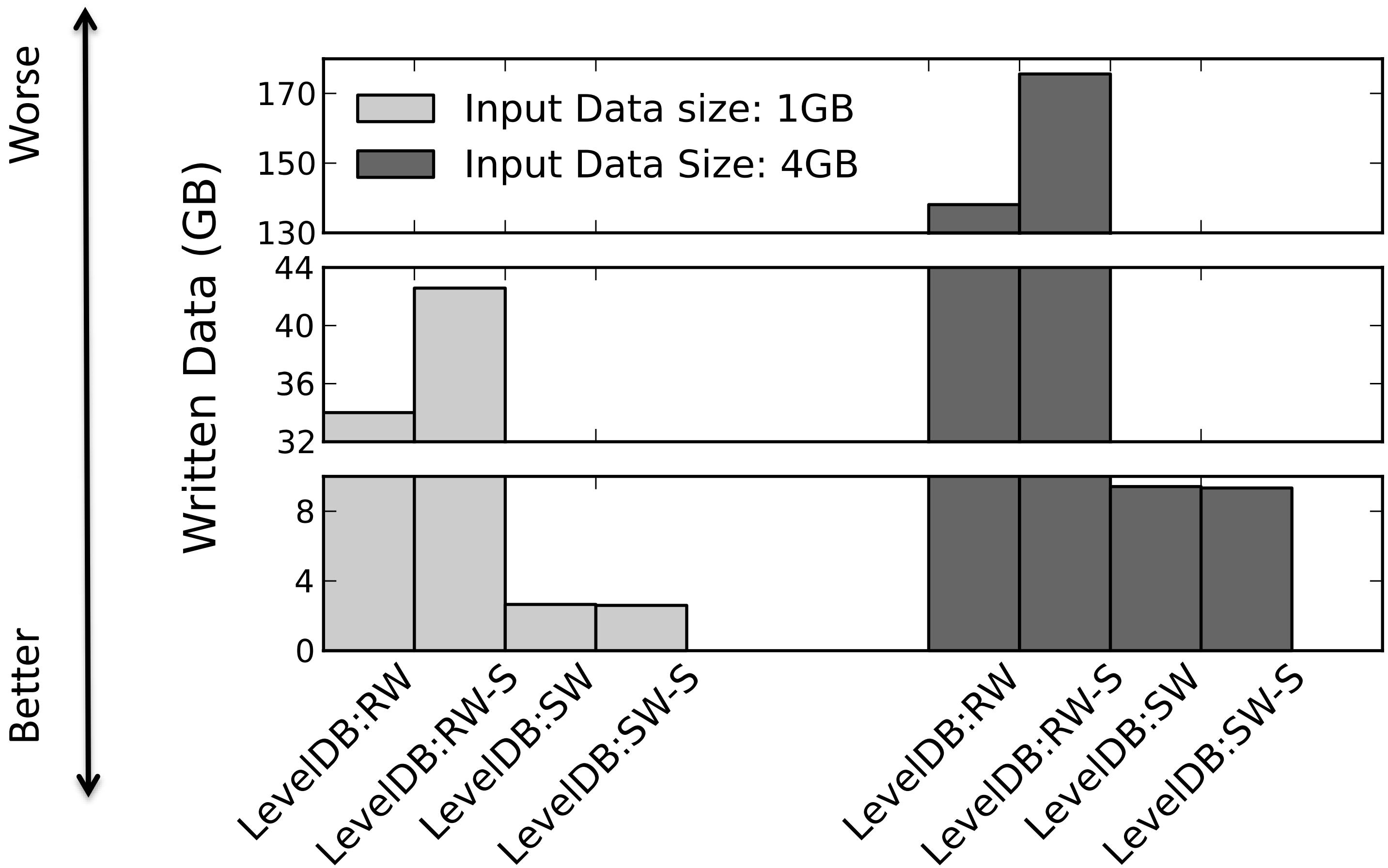
Limitations of existing solutions

- 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

Auxiliary Write Amplification

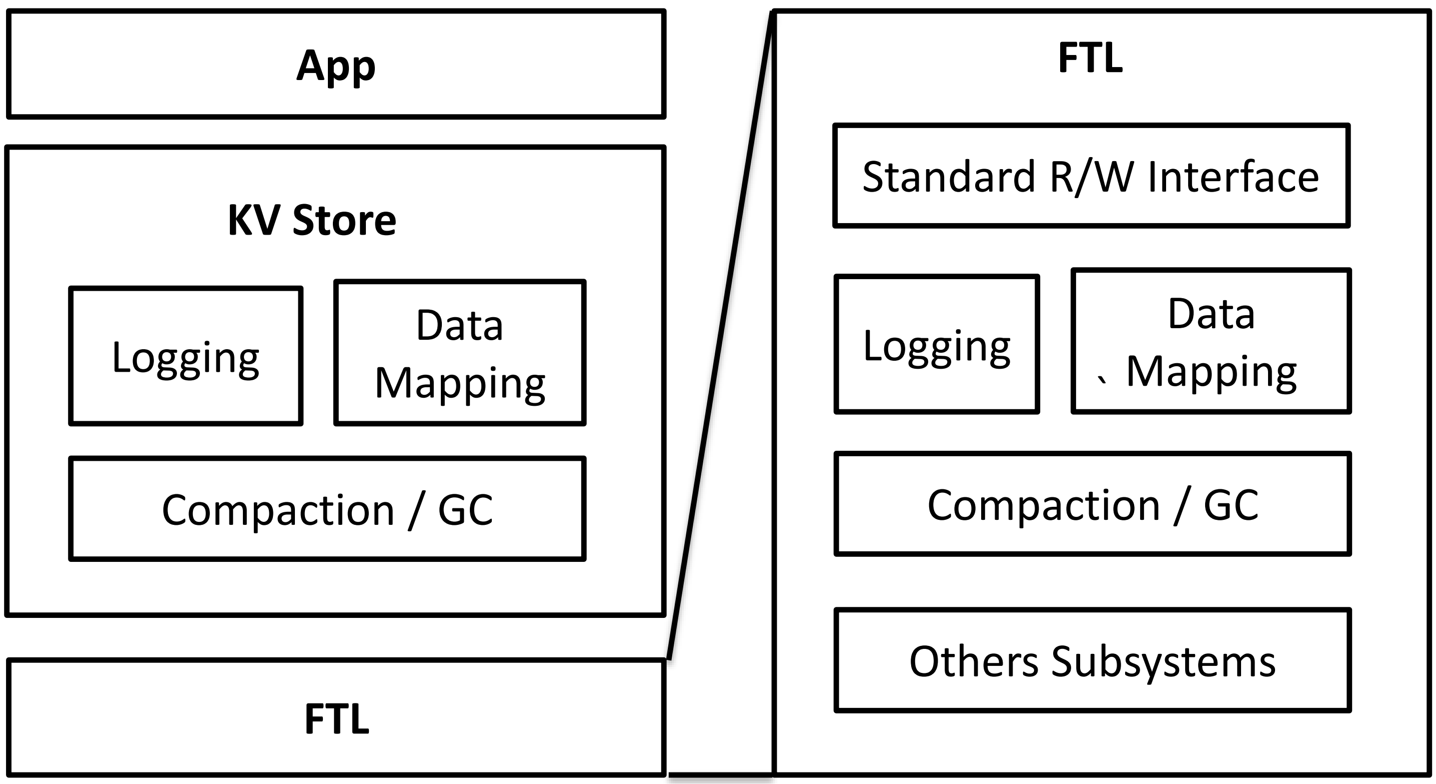


How bad is the situation?

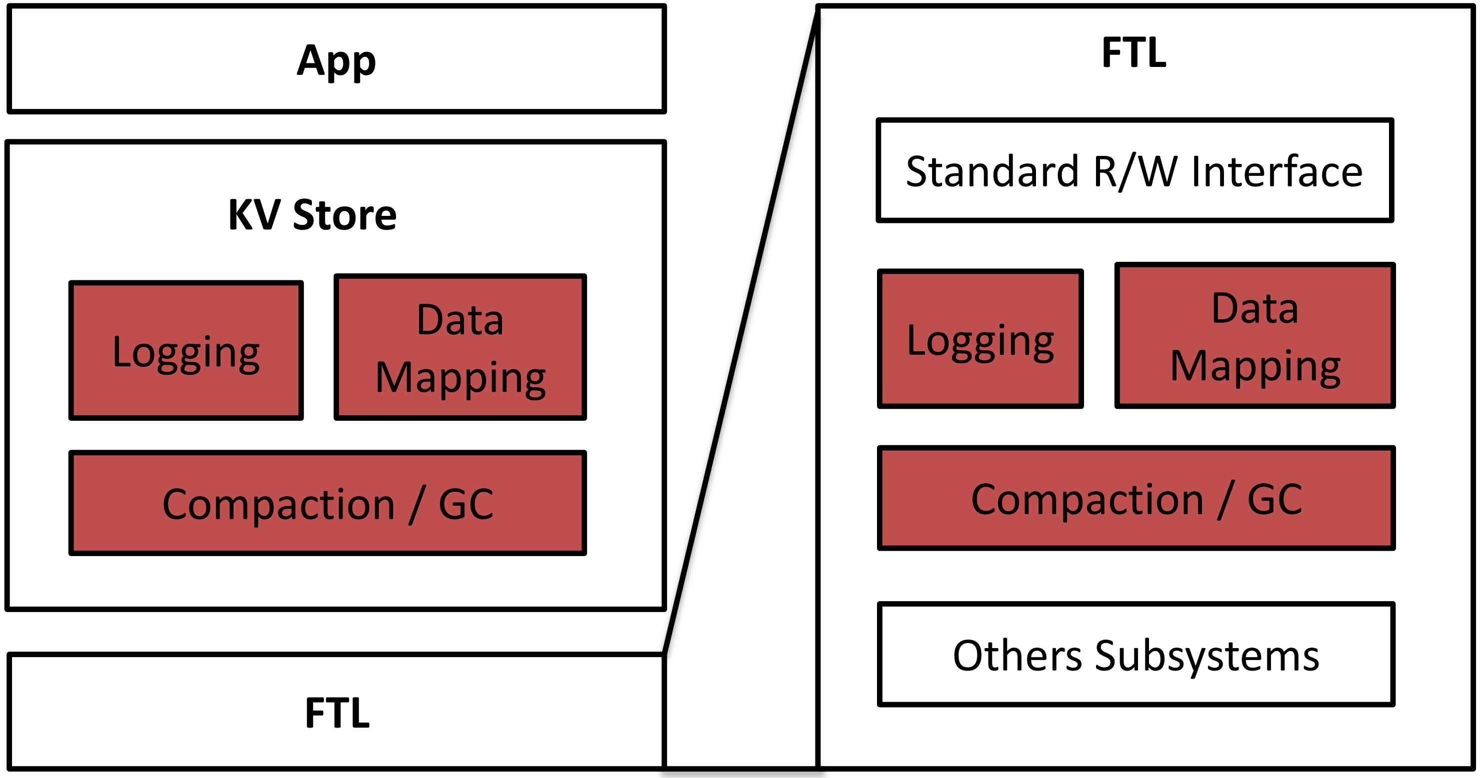


- Auxiliary write amplification varying from 2.5x to 43x

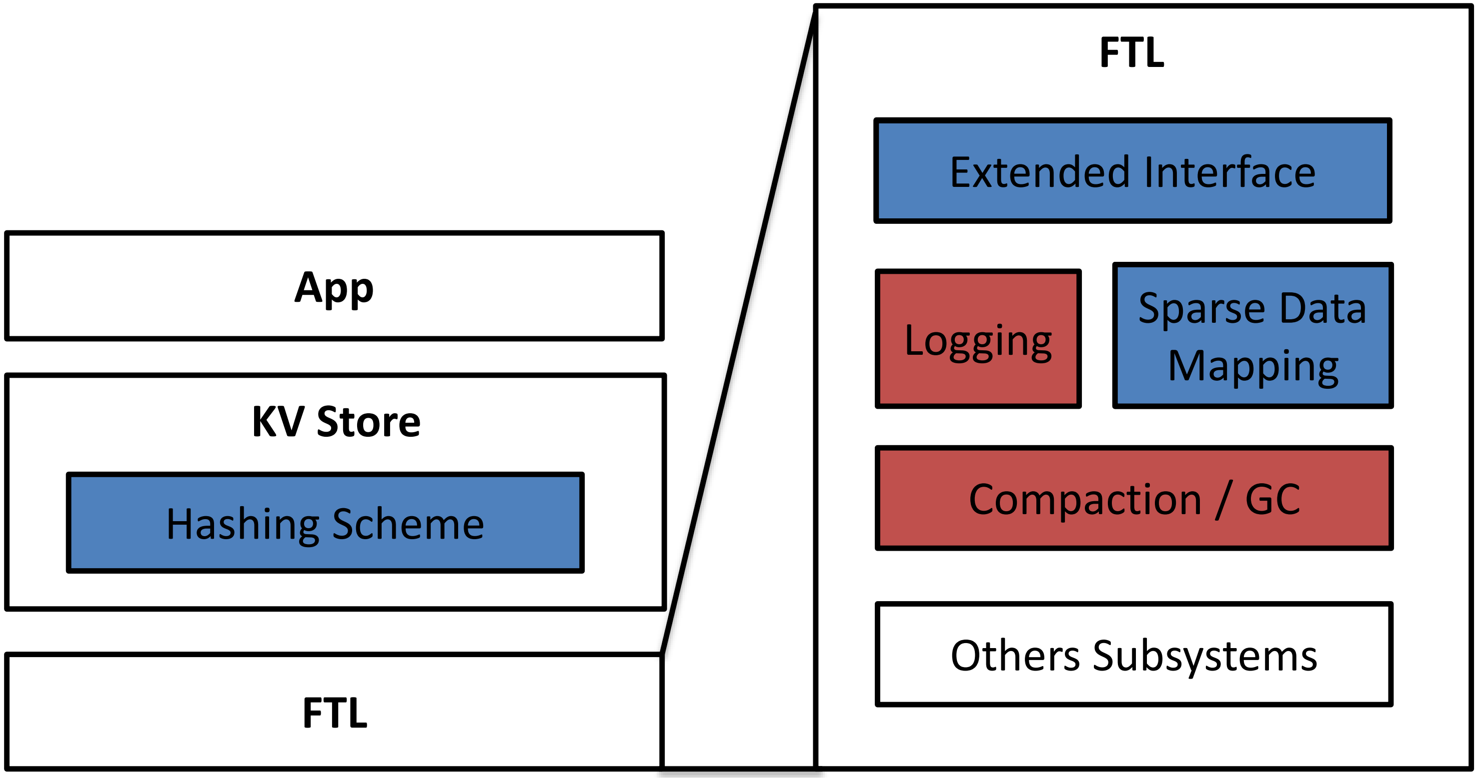
Existing KV Store Designs



FTL Cooperative KV Store Design



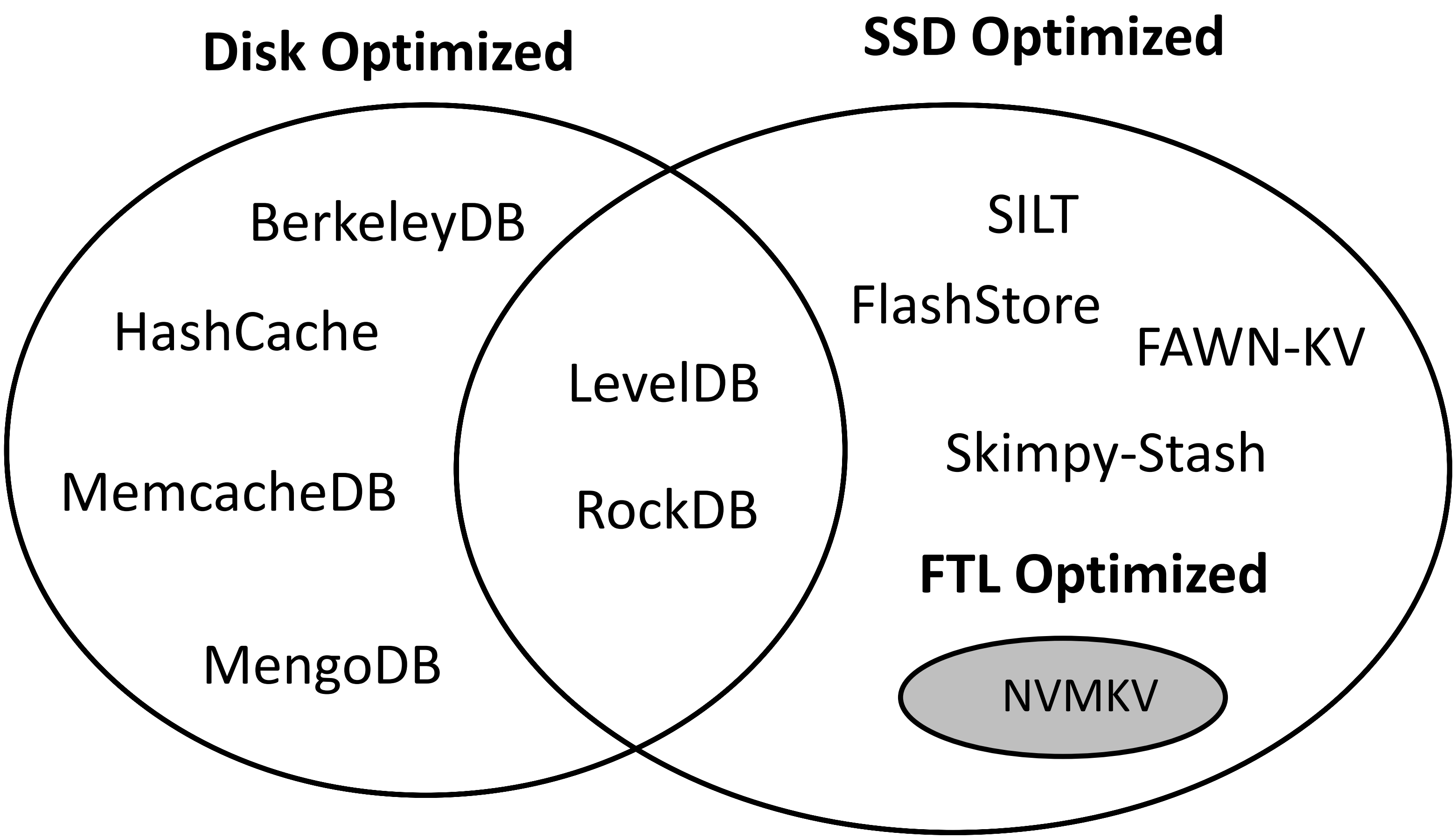
FTL Cooperative KV Store Design



New Approach

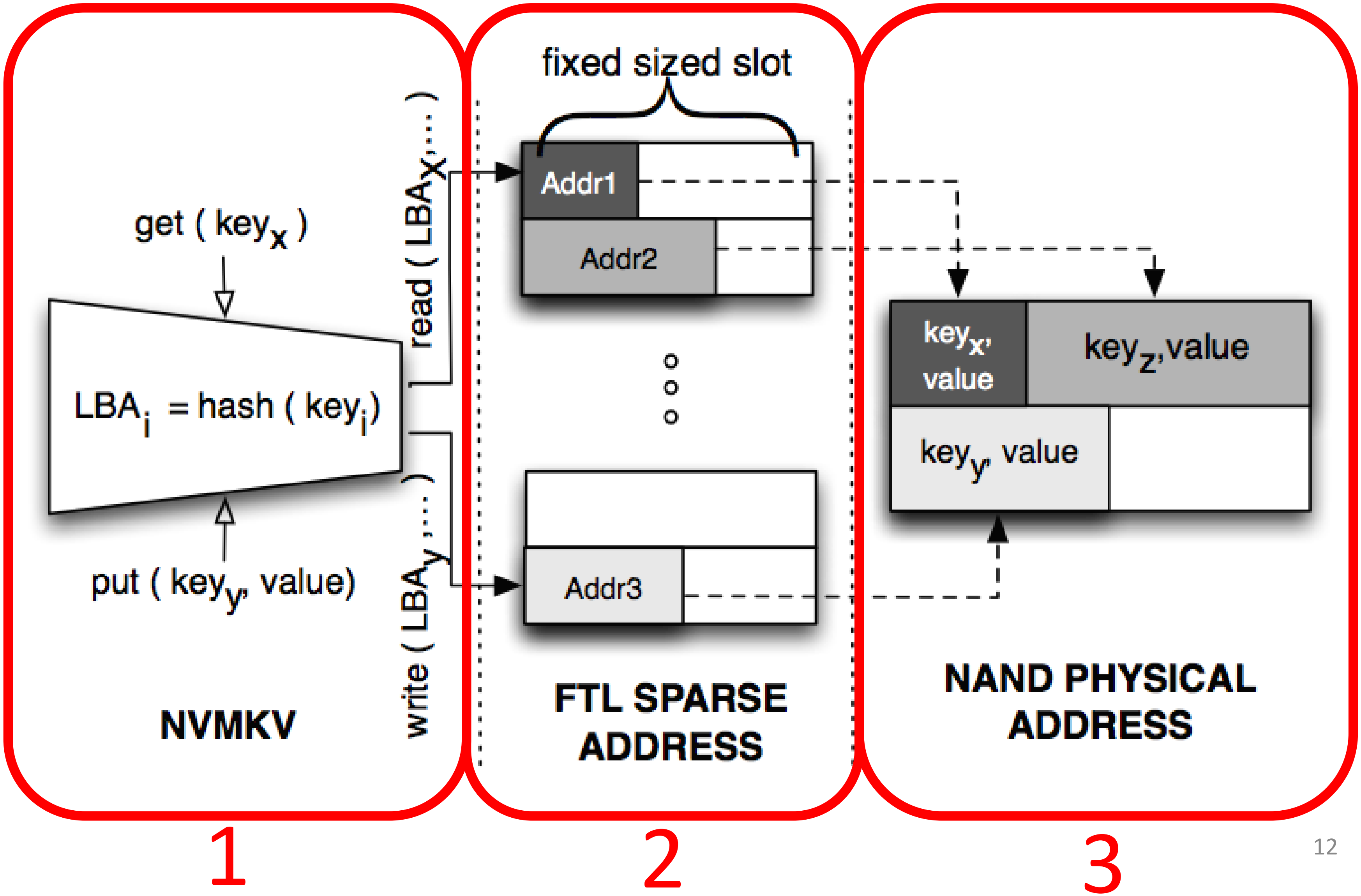
- Cooperative design with FTL
  - Minimize auxiliary write amplification
  - Maximize application level performance
  - Leverage FTL for atomicity and durability
    - Using the extended interface, updates are done atomically
  - Use constant amount of metadata
    - Independent of the number of KV pairs
  - Provide close to raw device performance
  - Provide greater I/O parallelism
    - Implementation with fewer locks

Classes of Key-Value Stores



Design

Sparse Address Mapping

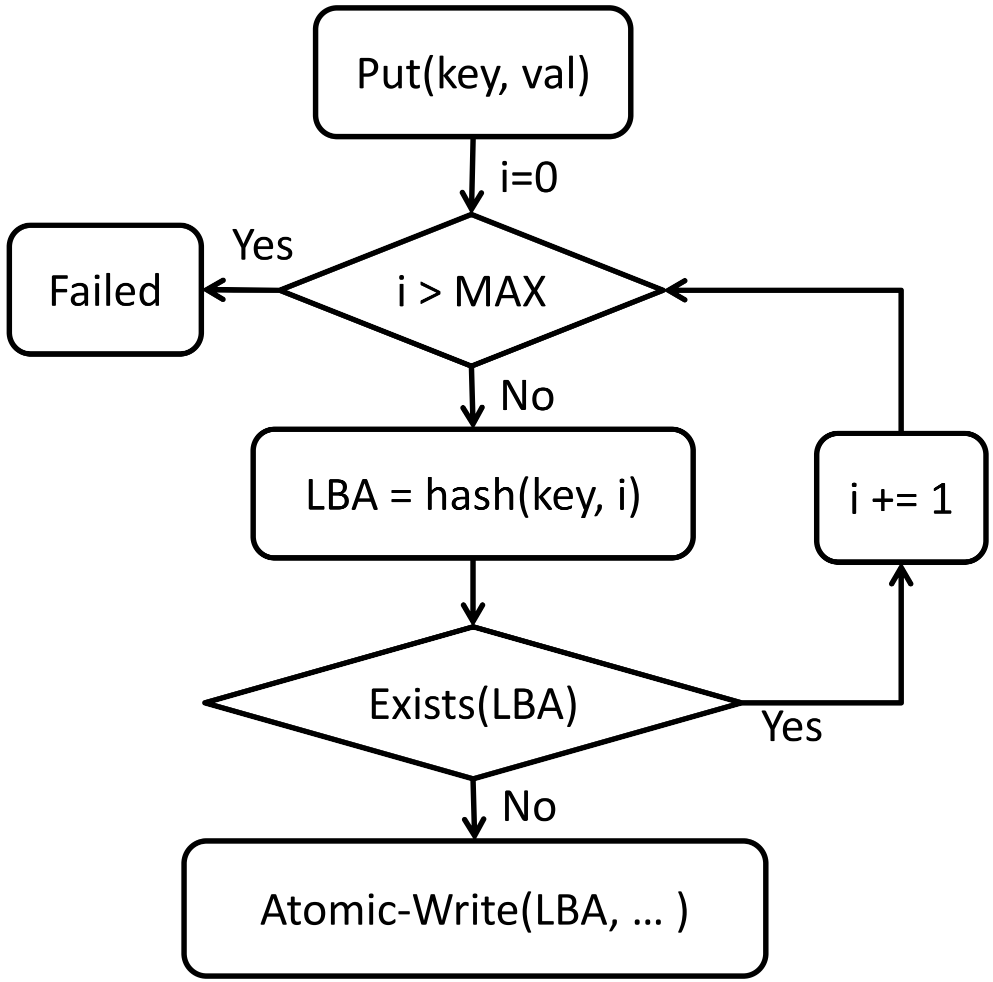


Extended FTL Interface

Primitive	Description
Exists	Query if an address is populated.
Atomic-Write	Write to an address range atomically.
Atomic-Trim	Delete an address range atomically.
Iterate	Return all populated addresses.

Many applications will benefit from having this extended interface available to them!

Hashing and Collision

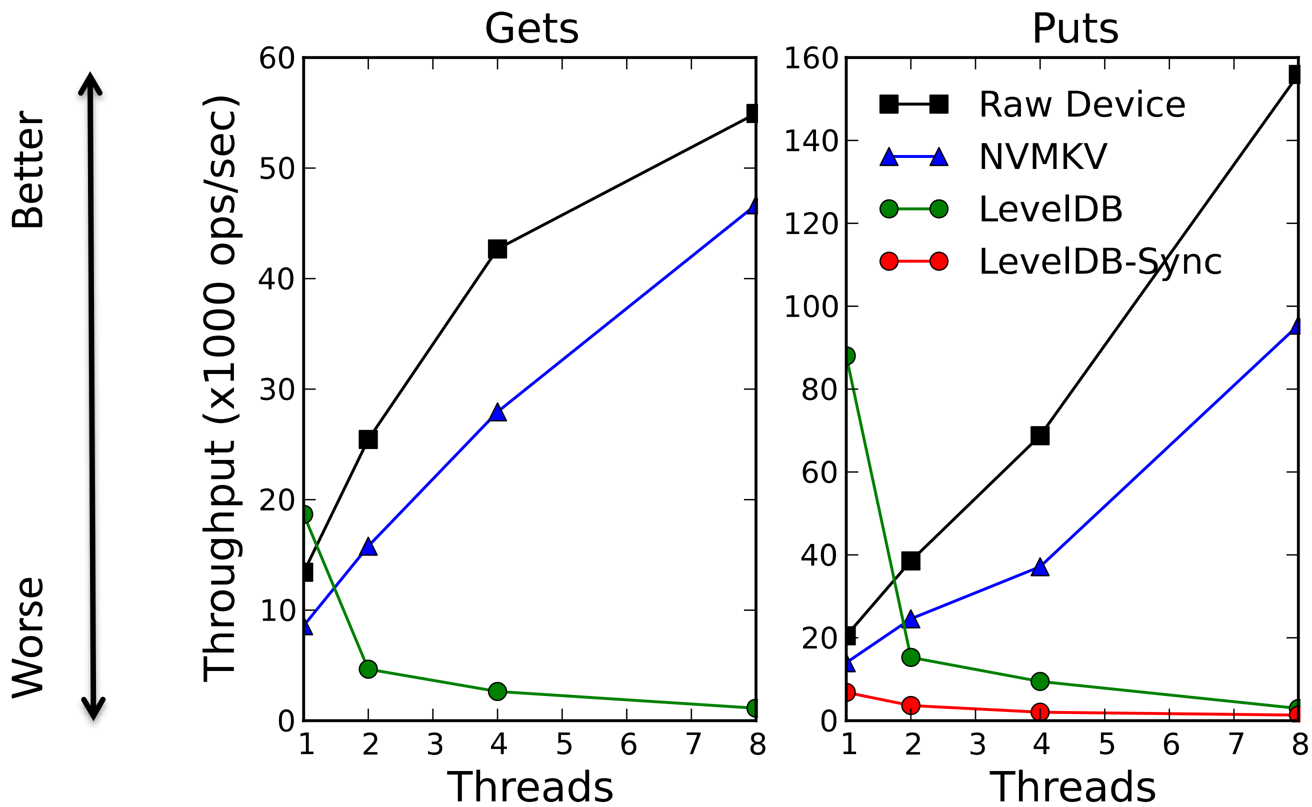


Hashing and Collision (Details)

- Keys are hashed into 48-bit addresses
  - Key Bit Range (KBR): maximum number of keys
  - Value Bit Range (VBR): maximum kv-pair size
  - KBR + VBR = 48 bits
- Collision are handle with Polynomial probing
  - Try 8 different locations before failing
- Case Study: {Device Size: 1TB, KBR: 36, VBR: 12}
  - 64 billion kv-pairs of 2MB of maximum size
  - Probability of put failing is $1/2^{40}$

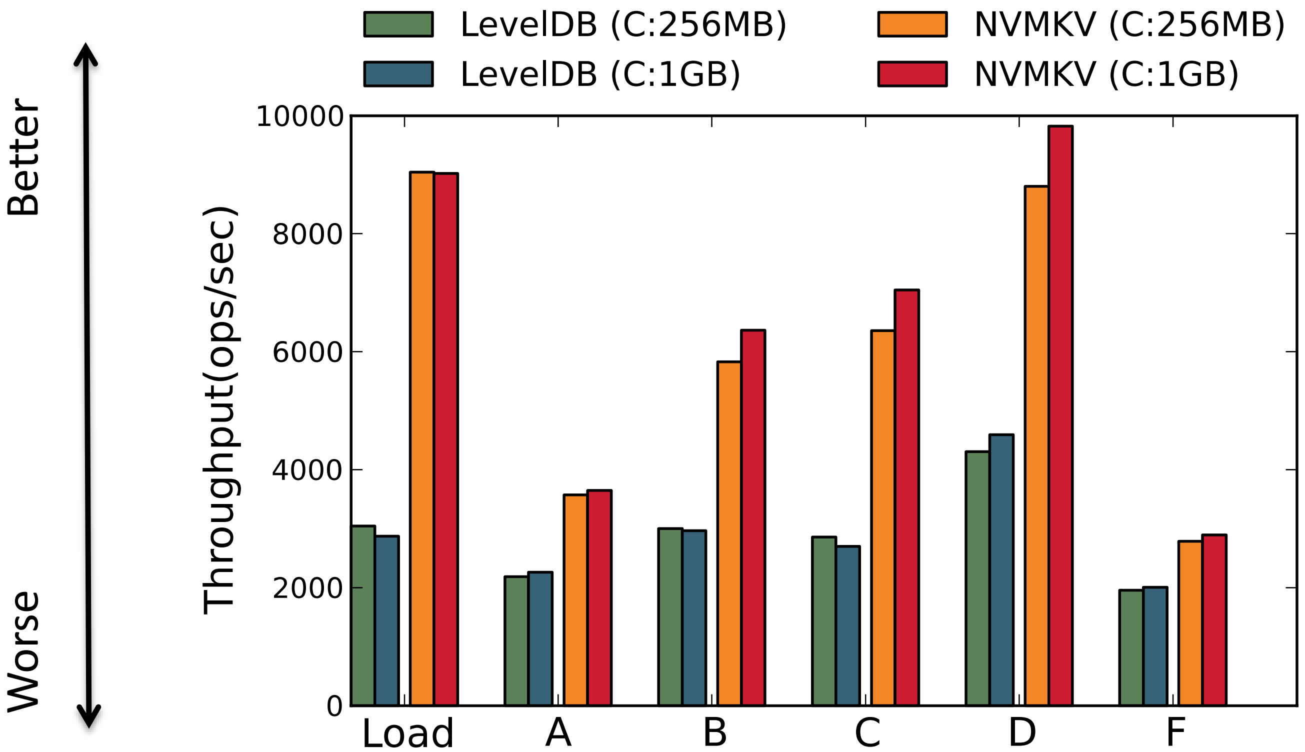
Evaluation Results

Microbenchmark Results



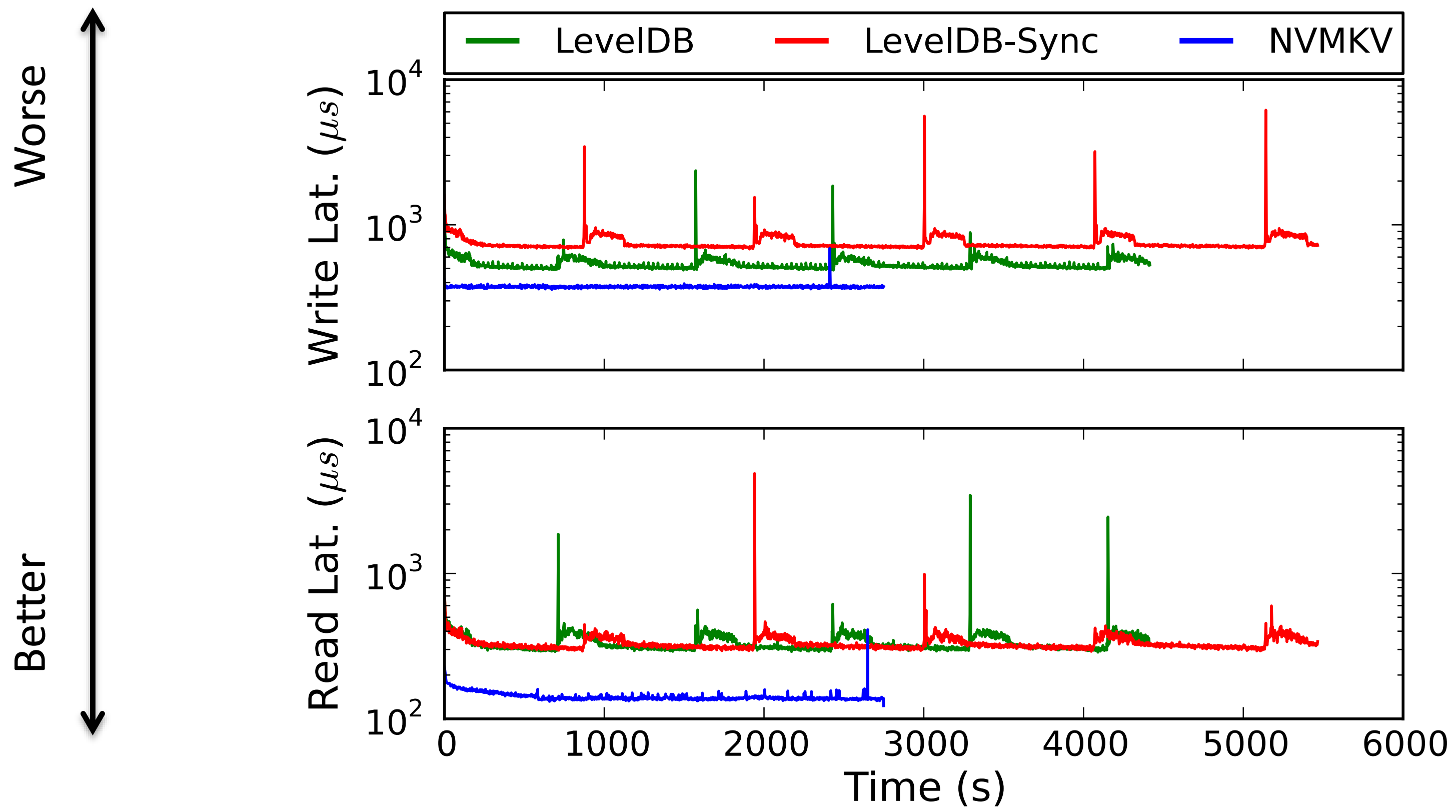
- NVMKV scales better than LevelDB as the number of threads increases
- NVMKV outperforms LevelDB even at low thread counts and without FS cache
- NVMKV provides durability and atomicity of put operations

LevelDB Comparison using YCSB



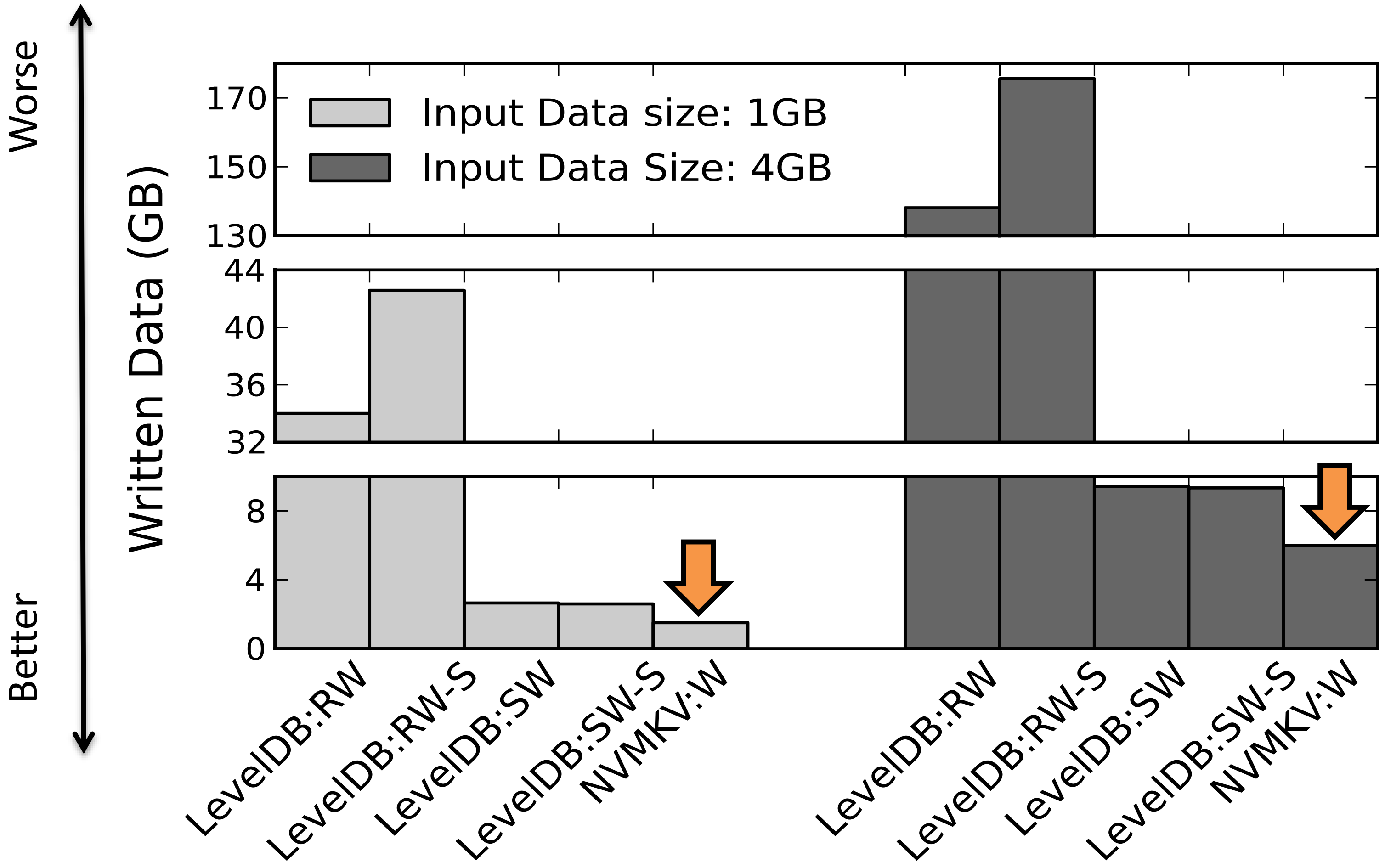
- NVMKV outperforms LevelDB in every workload using 1/4 of the cache size
- LevelDB uses both it's own cache and the file system cache
- Unlike LevelDB, NVMKV perform every update atomically and synchronous

LevelDB Comparison using YCSB



- NVMKV outperforms LevelDB in both sync and async configurations
- NVMKK provides a more stable and predictable performance

Auxiliary Write Amplification



Conclusions

Summary

- We propose a FTL cooperative design that allows for:
  - Simple KV-Store design and implementation
  - Constant amount of metadata
  - High performance / parallelism
  - Atomicity and durability of KV operations
  - Low write amplification

Fork me on GitHub

Thank you!

opennvm.github.io

FLORIDA INTERNATIONAL UNIVERSITY

FUSION-iO