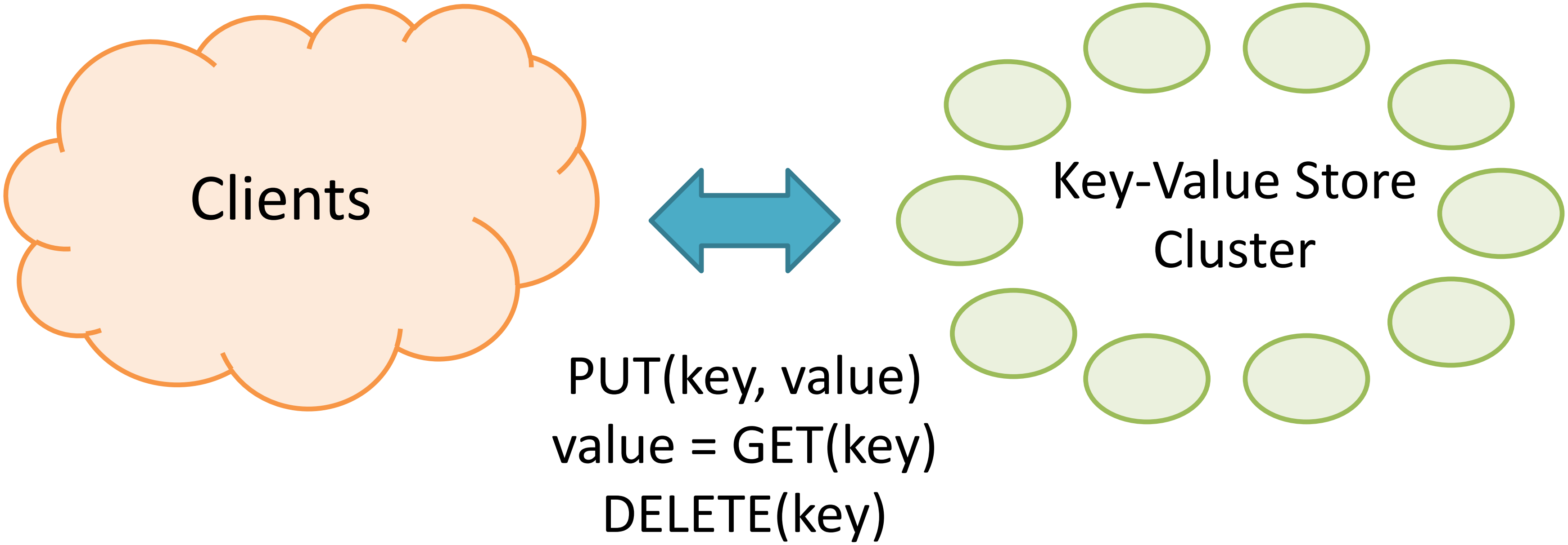
SILT: A Memory-Efficient, High-Performance Key-Value Store

Hyeontaek Lim, Bin Fan, David G. Andersen
Michael Kaminsky†

Carnegie Mellon University
†Intel Labs

2011-12-08

Key-Value Store



- E-commerce (Amazon)
- Web server acceleration (Memcached)
- Data deduplication indexes
- Photo storage (Facebook)

Flash-Based Key-Value Stores

- Many key-value store projects have examined **flash memory** as main storage
  - Faster than disk, cheaper than DRAM
  - 5+ papers in well-known systems/DB/networking conferences in the past 3 years
  - Several commercial products

- Frequently appearing design question: **reducing DRAM consumption** (why?)

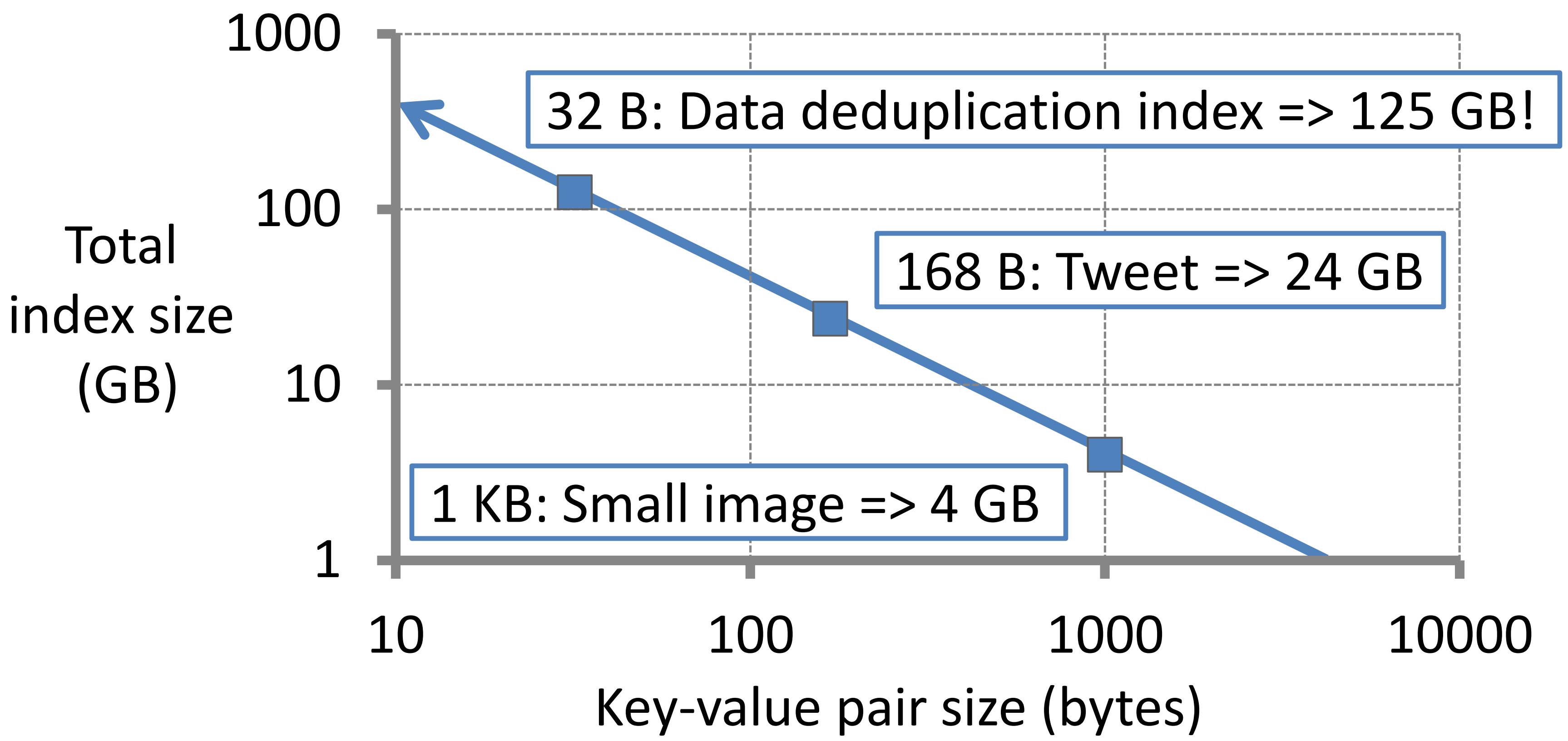
Need for In-Memory Indexing

- Random flash reads/sec = 48,000 (1 SATA SSD)
  - Fast… enough?
  - E.g., Memcached (DRAM-only key-value store) achieves over 300,000 queries per second

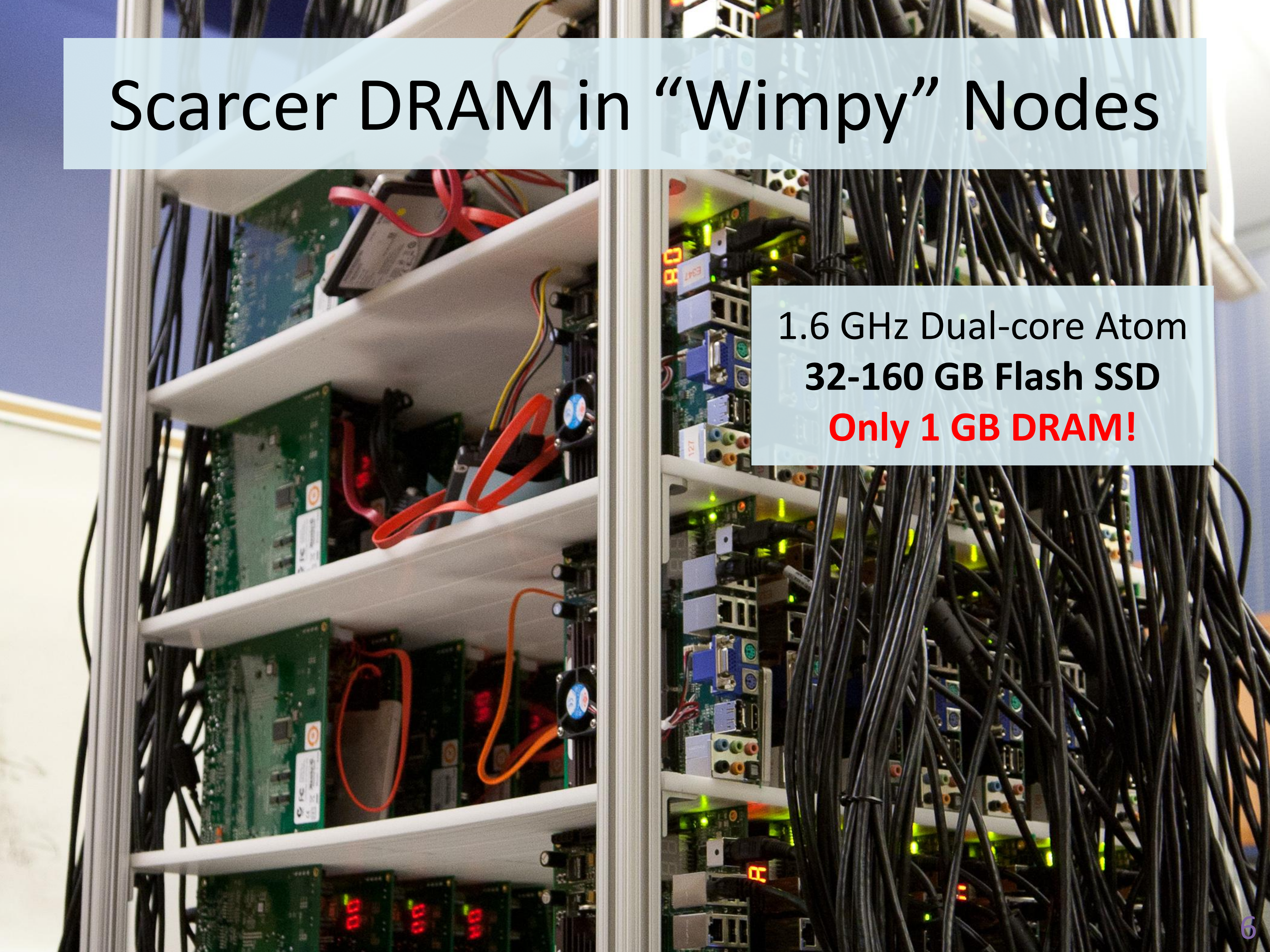
- **DRAM** is used to **index** (locate) items on flash for efficient flash I/O

Small Data, Large Index?

- 1 TB of data to store on flash
- 4 bytes of DRAM per key-value pair (previous state-of-the-art)
- ➢ Total index size = 1 TB / (key-value pair size) x 4 bytes



Scarcer DRAM in “Wimpy” Nodes



1.6 GHz Dual-core Atom

32-160 GB Flash SSD

Only 1 GB DRAM!

Three Metrics to Minimize

Memory overhead = Index size per entry

- Ideally 0 bytes/entry (no memory overhead)

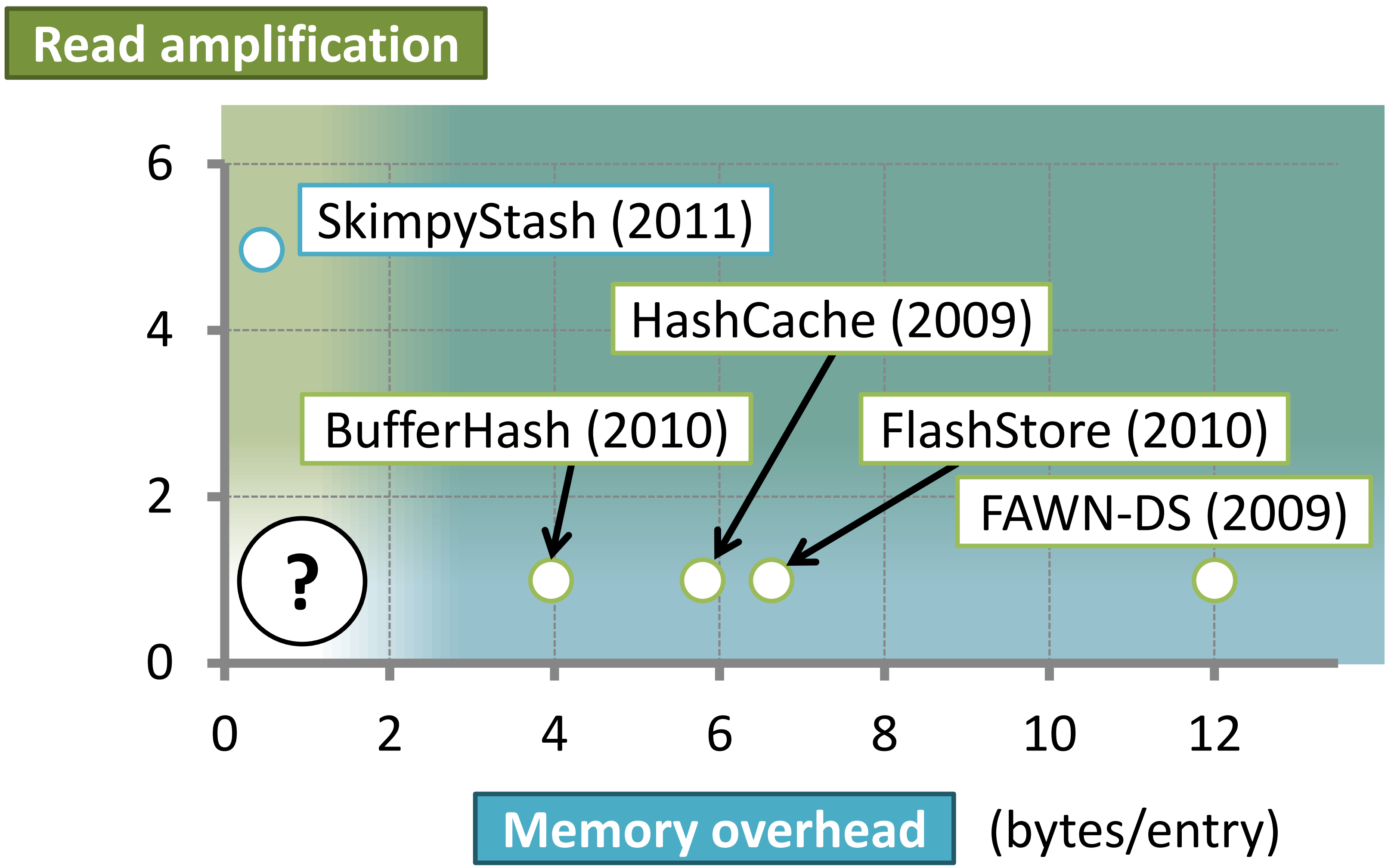
Read amplification = Flash reads per query

- Limits **query throughput**
- Ideally 1 (no wasted flash reads)

Write amplification = Flash writes per entry

- Limits **insert throughput**
- Also reduces **flash life expectancy**
  - Must be small enough for flash to last a few years

Landscape: Where We Were



Seesaw Game?

How can we improve?

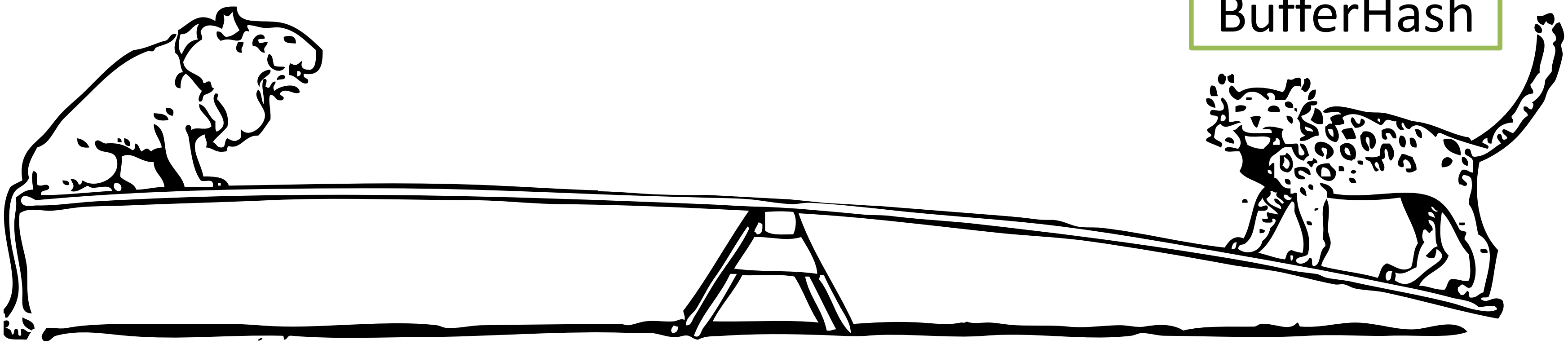
SkimpyStash

FAWN-DS

FlashStore

HashCache

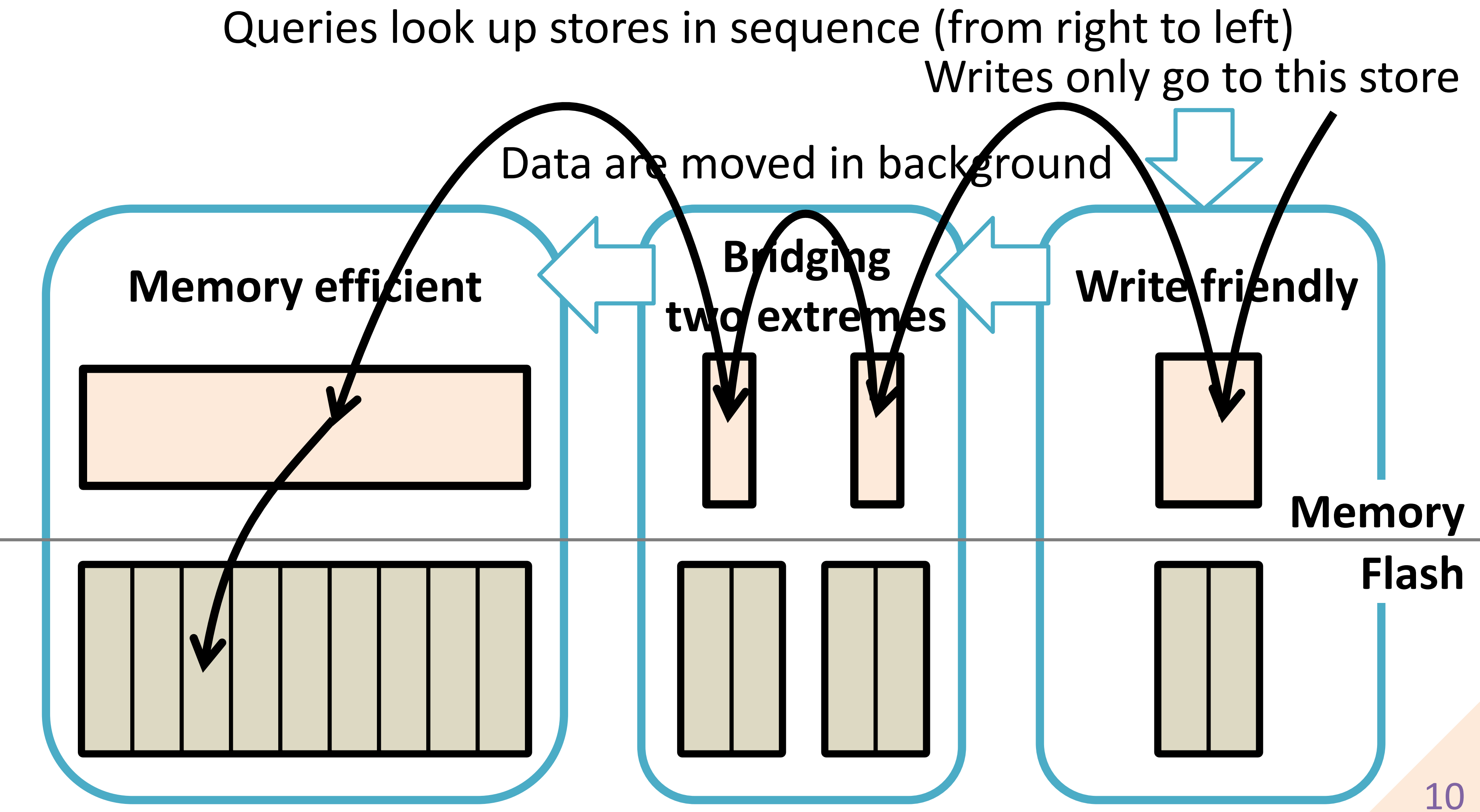
BufferHash



Memory efficiency

High performance

“SILT” Preview: (1) Three Stores with (2) New Index Data Structures

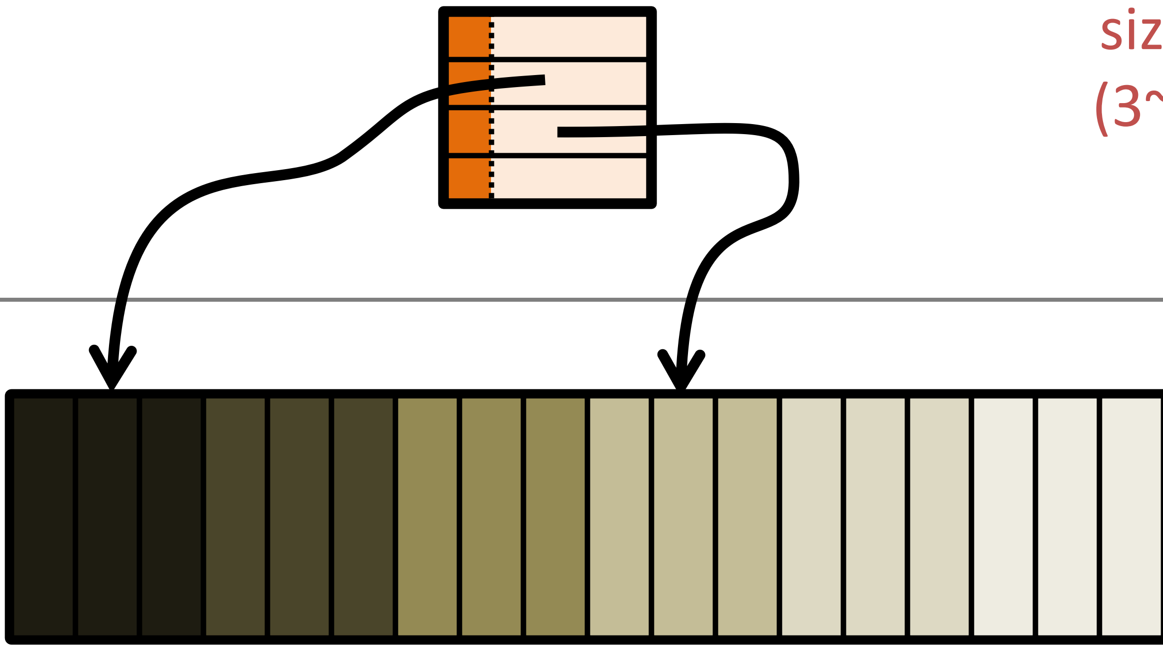


LogStore: No Control over Data Layout

~~Naive Hashtable (48+ B/entry)~~

SILT Log Index (6.5+ B/entry)

Still need pointers:
size ≥ log N bits/entry
(3~4 bytes in practice)



Memory

Flash

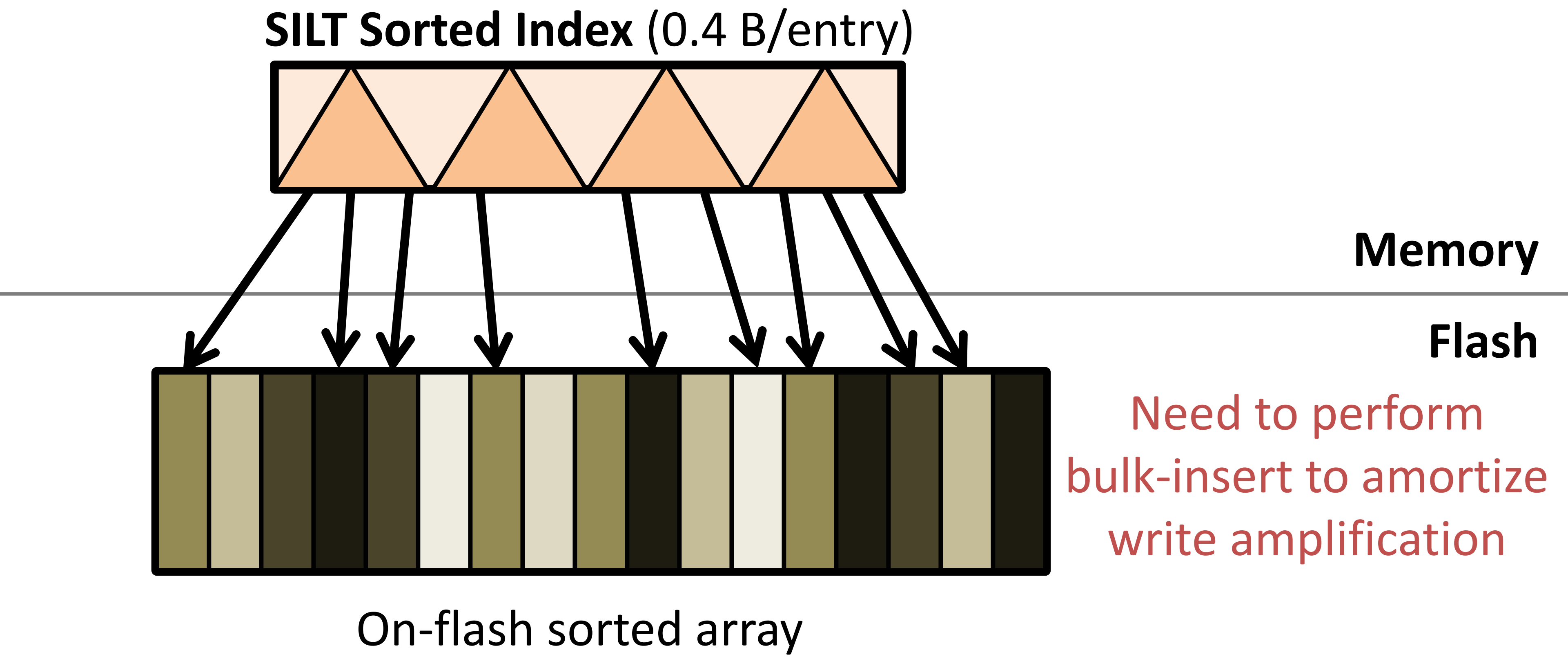
Inserted entries are appended

(sequential writes only)

(Older) On-flash log (Newer)

Memory overhead	Write amplification
6.5+ bytes/entry	1

SortedStore: Space-Optimized Layout

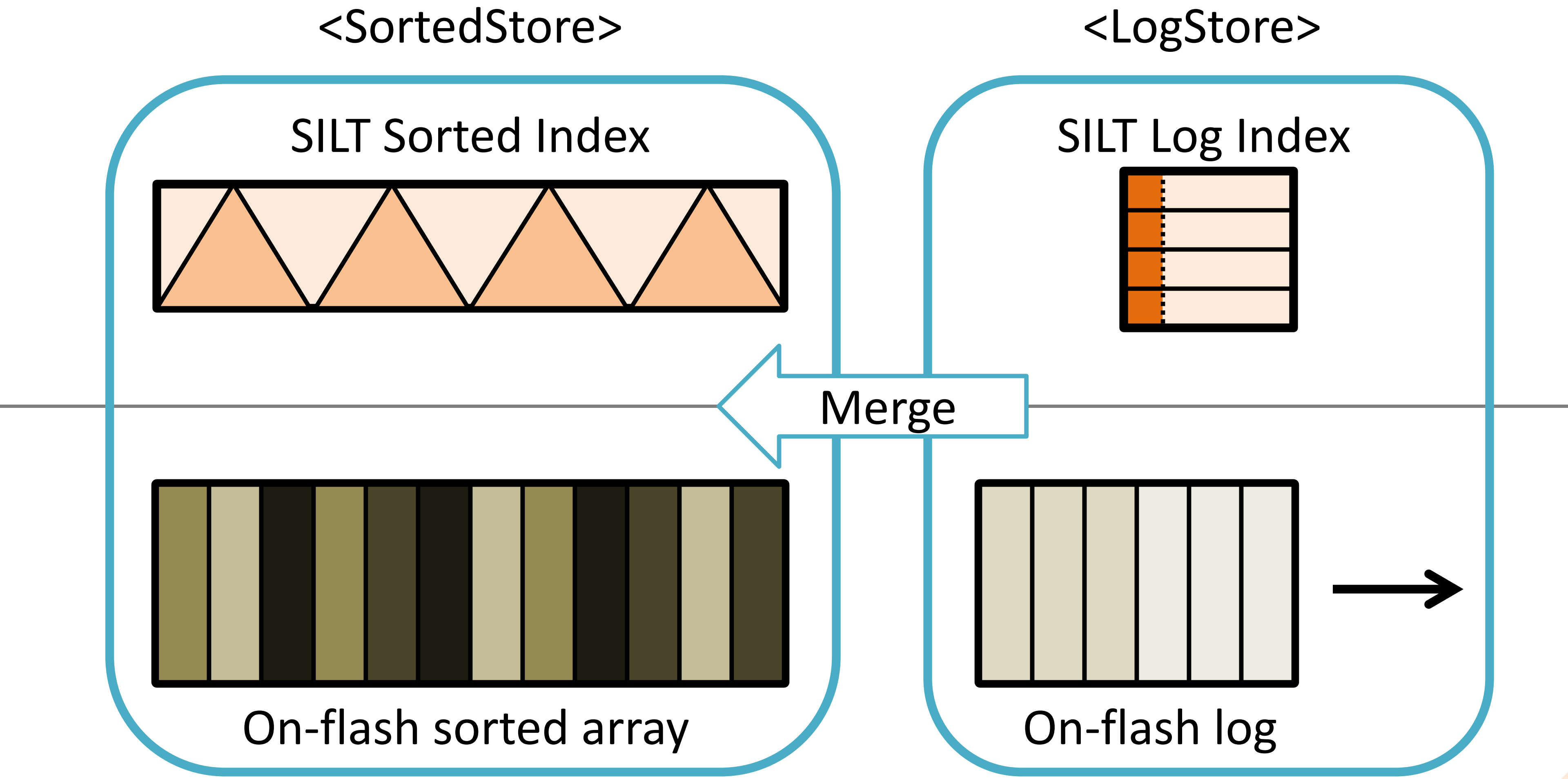


Need to perform bulk-insert to amortize write amplification

Memory overhead
0.4 bytes/entry

Write amplification
High

Combining SortedStore and LogStore



Merge: (1) Sort log (2) Sequentially merge two sorted data

Achieving both Low Memory Overhead and Low Write Amplification

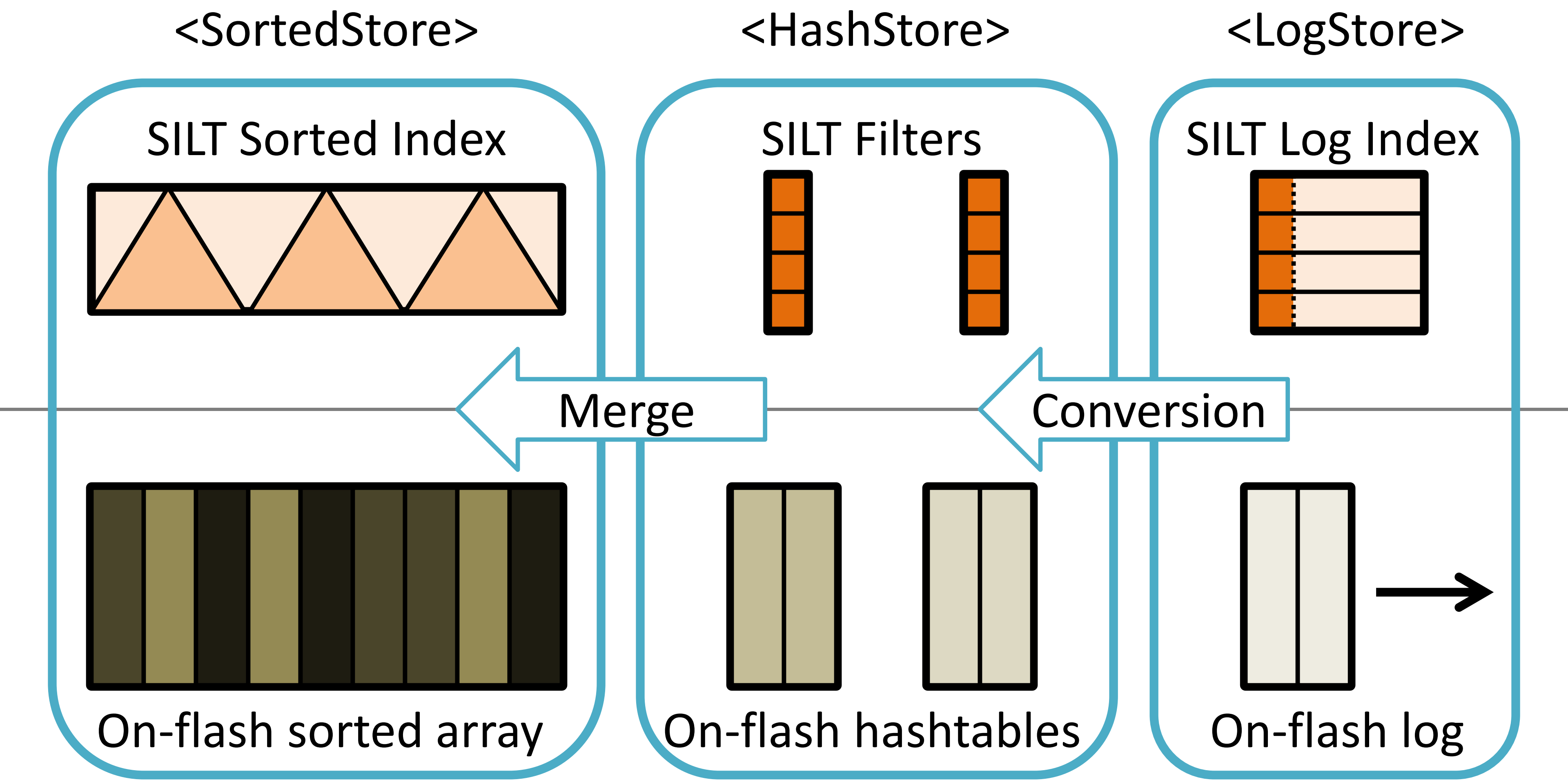
SortedStore
- Low memory overhead
- High write amplification

LogStore
- High memory overhead
- Low write amplification

Now we can achieve simultaneously:
Write amplification = 5.4 = 3 year flash life
Memory overhead = 1.3 B/entry

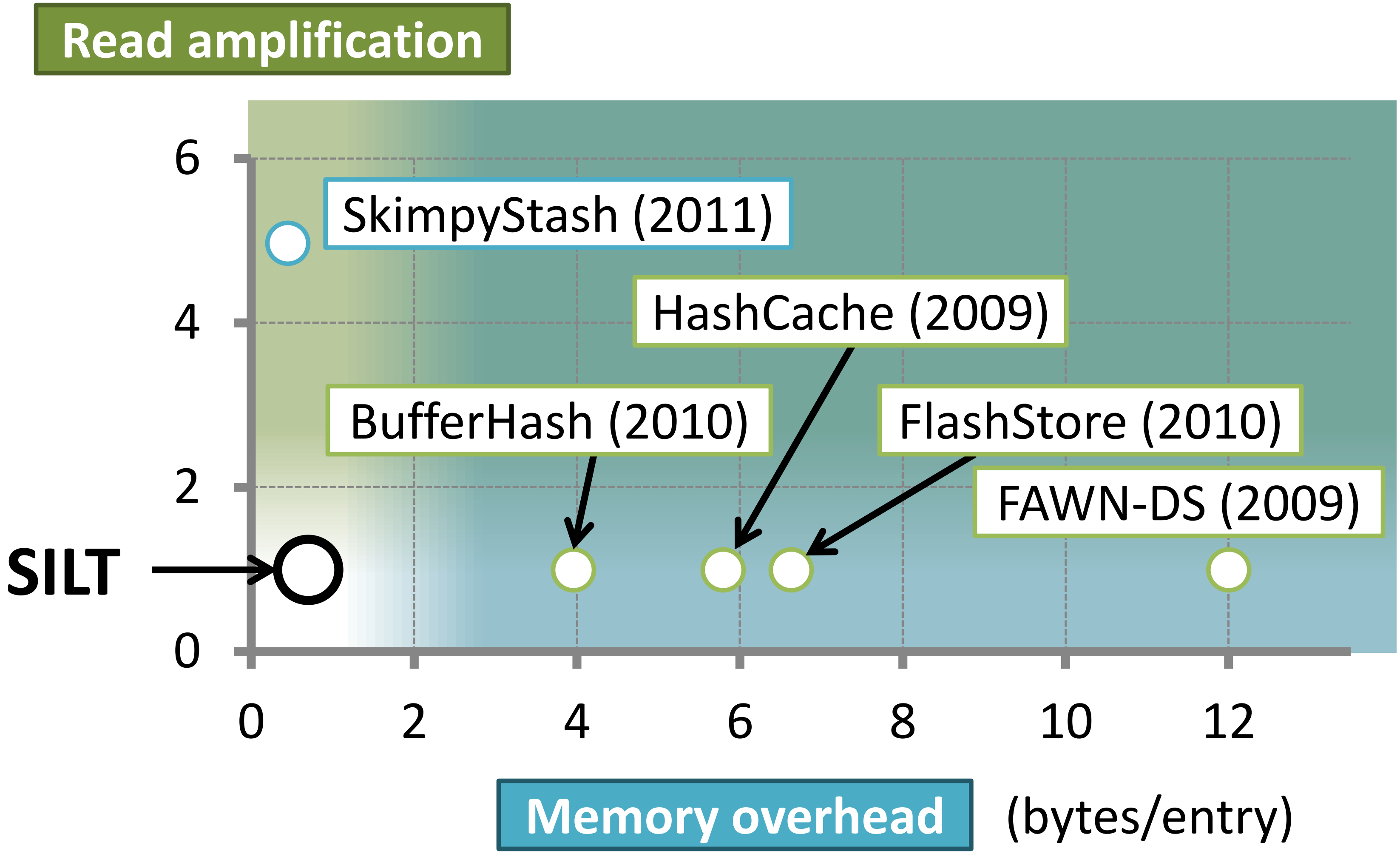
With “HashStores”, memory overhead = 0.7 B/entry!

SILT’s Design (Recap)



Memory overhead	Read amplification	Write amplification
0.7 bytes/entry ✓	**1.01** ✓	**5.4** ✓

Landscape: Where We Are



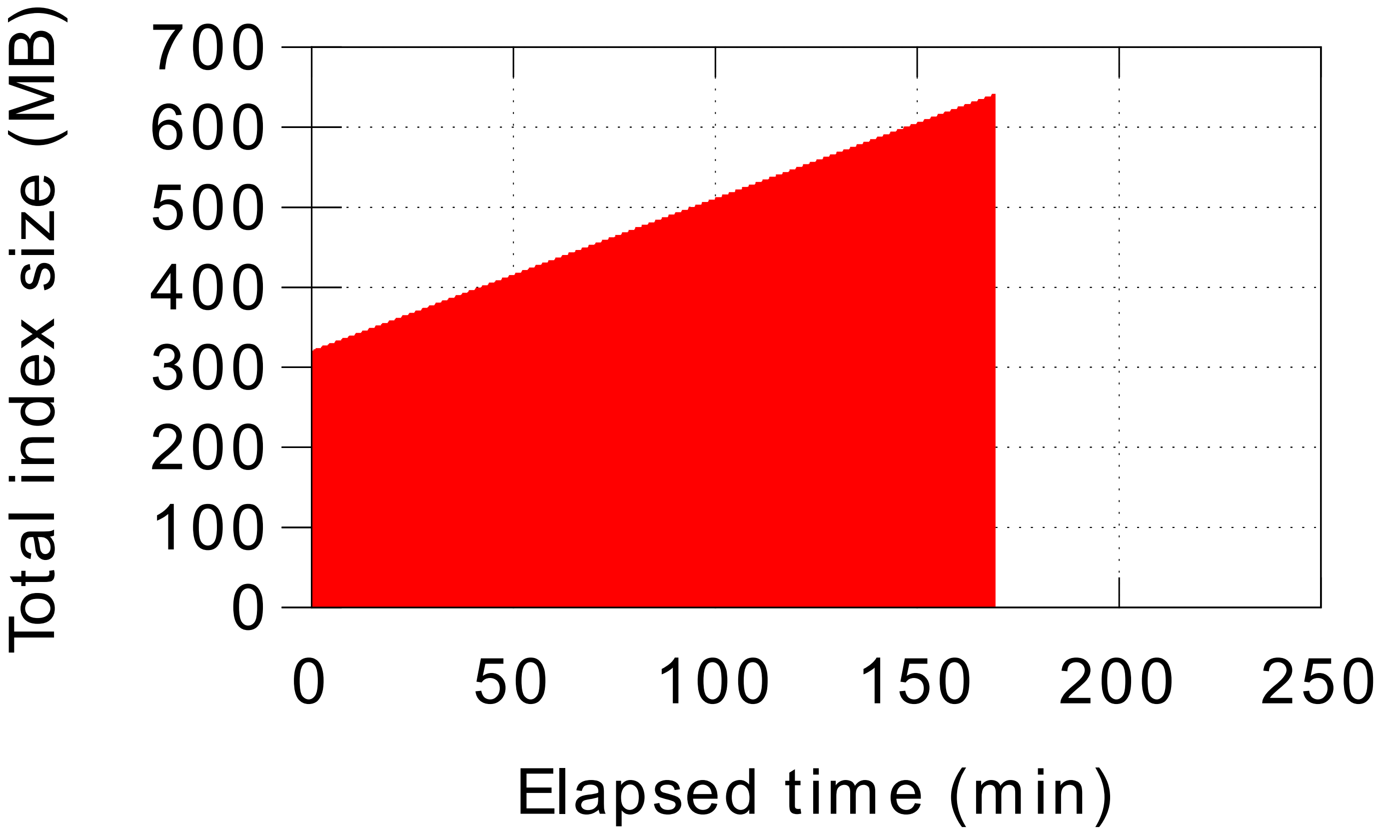
Evaluation

1. Various combinations of indexing schemes
2. Background operations (merge/conversion)

Experiment Setup	
CPU	2.80 GHz (4 cores)
Flash drive	SATA 256 GB (48 K random 1024-byte reads/sec)
Workload size	20-byte key, 1000-byte value, ≥ 50 M keys
Query pattern	Uniformly distributed (worst for SILT)

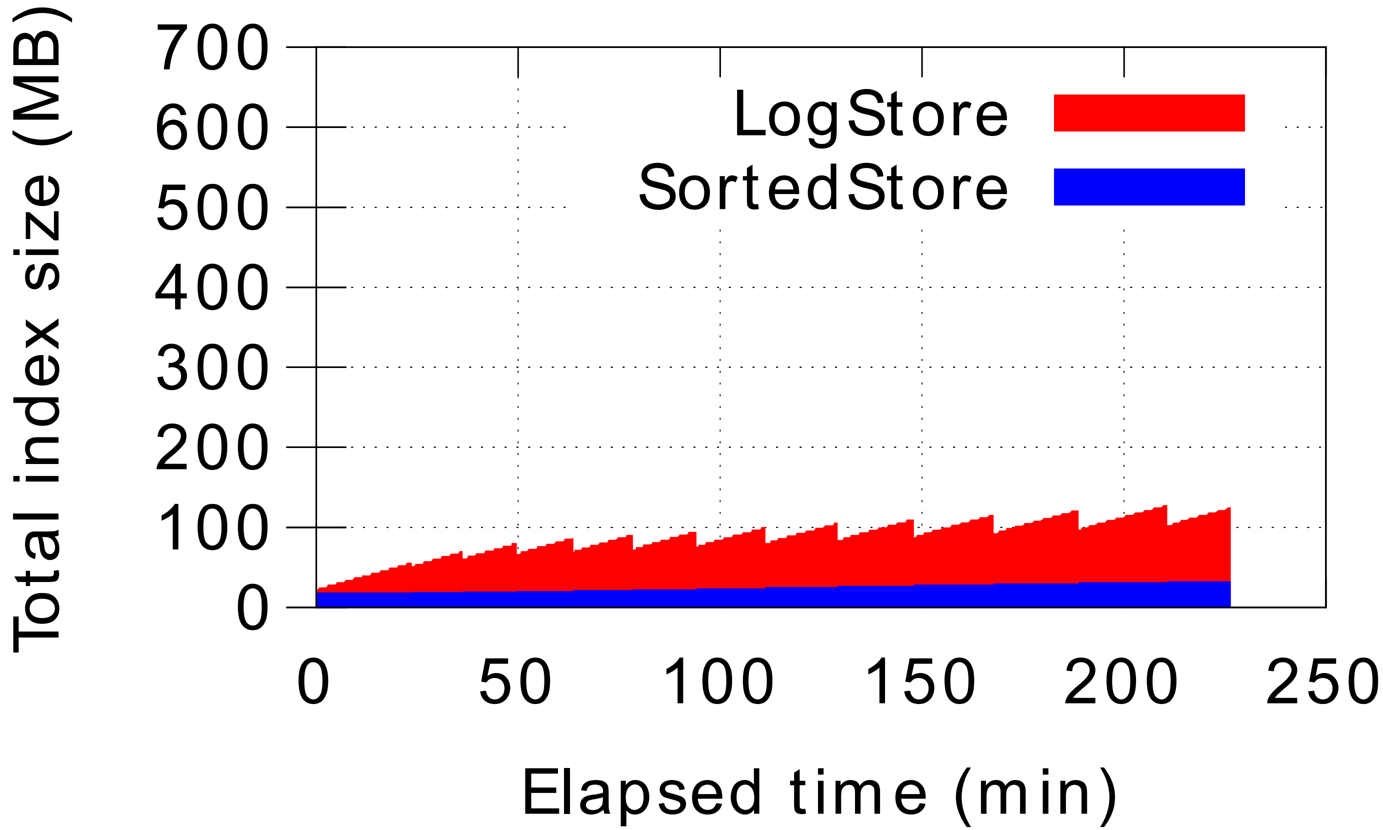
LogStore Alone: Too Much Memory

Workload: 90% GET (50-100 M keys) + 10% PUT (50 M keys)



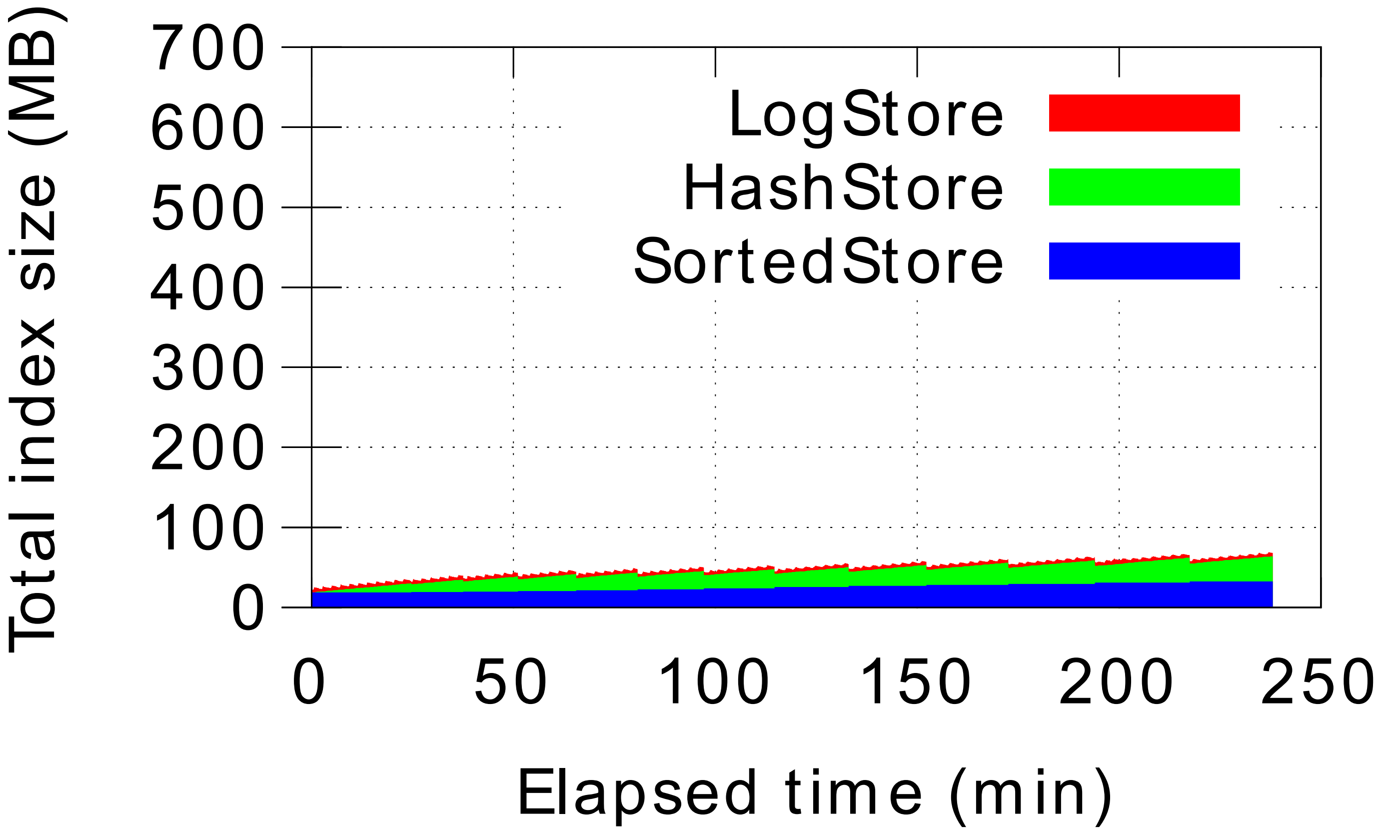
LogStore+SortedStore: Still Much Memory

Workload: 90% GET (50-100 M keys) + 10% PUT (50 M keys)



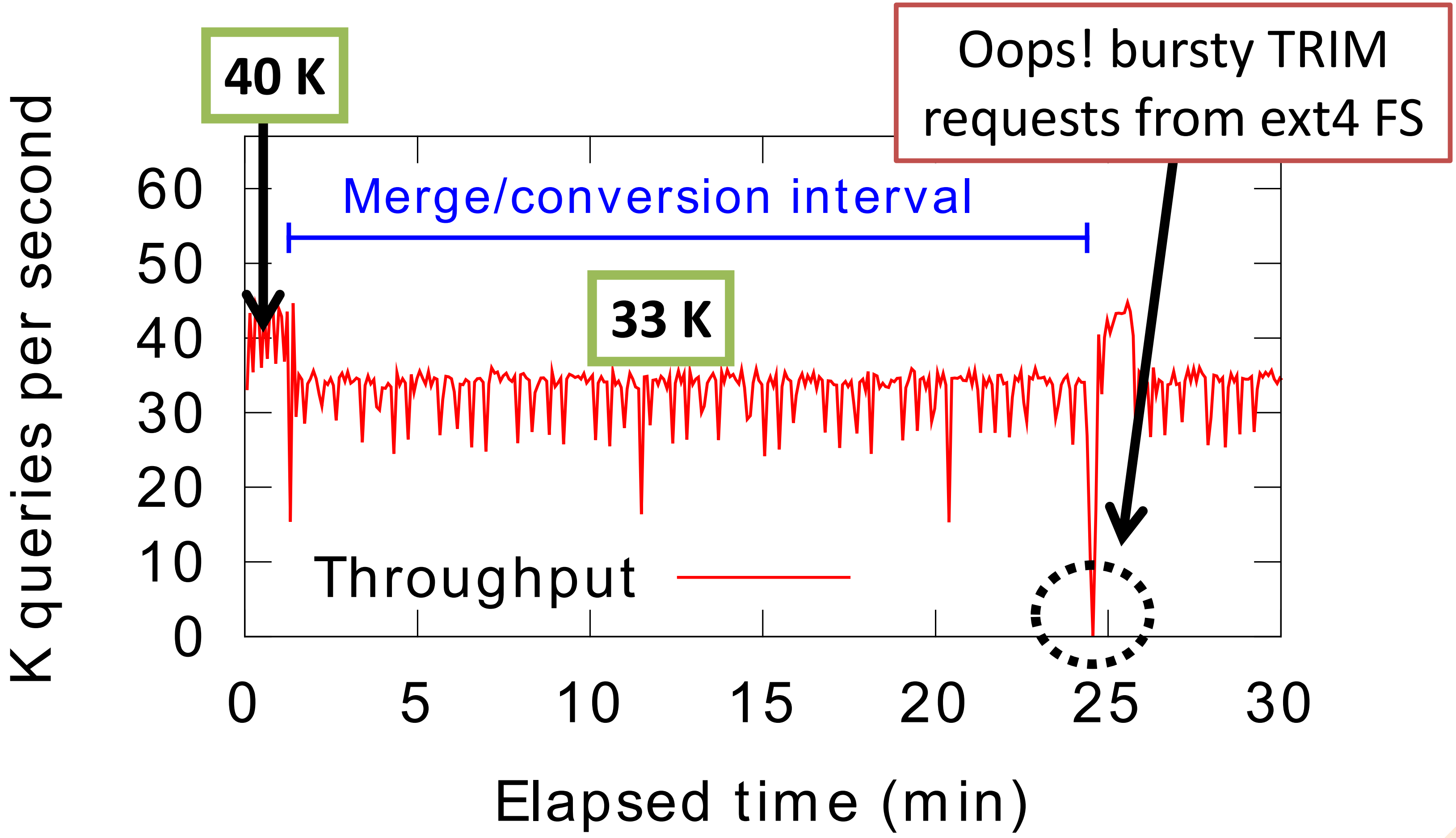
Full SILT: Very Memory Efficient

Workload: 90% GET (50-100 M keys) + 10% PUT (50 M keys)



Small Impact from Background Operations

Workload: 90% GET (100~ M keys) + 10% PUT



Conclusion

- SILT provides both **memory-efficient** and **high-performance** key-value store
  - New items are put into a write-friendly store; later migrated to a memory-efficient store
  - Two new compact index data structures

- Full source code is available
  - https://github.com/silt/silt