SILT

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

Introduction

Motivation

- Problem: Read Amplification

Motivation

- Problem: Read Amplification
- Solution(?): Keep index in DRAM

Motivation

- Problem: Read Amplification
- Solution(?): Keep index in DRAM
- New Problem: Expense and lower capacity of DRAM

Goals

Goals

- Low # of flash reads per GET

Goals

- Low # of flash reads per GET
- Sequential instead of Random Writes

Goals

- Low # of flash reads per GET
- Sequential instead of Random Writes
- Memory efficient indexing

Goals

- Low # of flash reads per GET
- Sequential instead of Random Writes
- Memory efficient indexing
- Computation efficient indexing

Goals

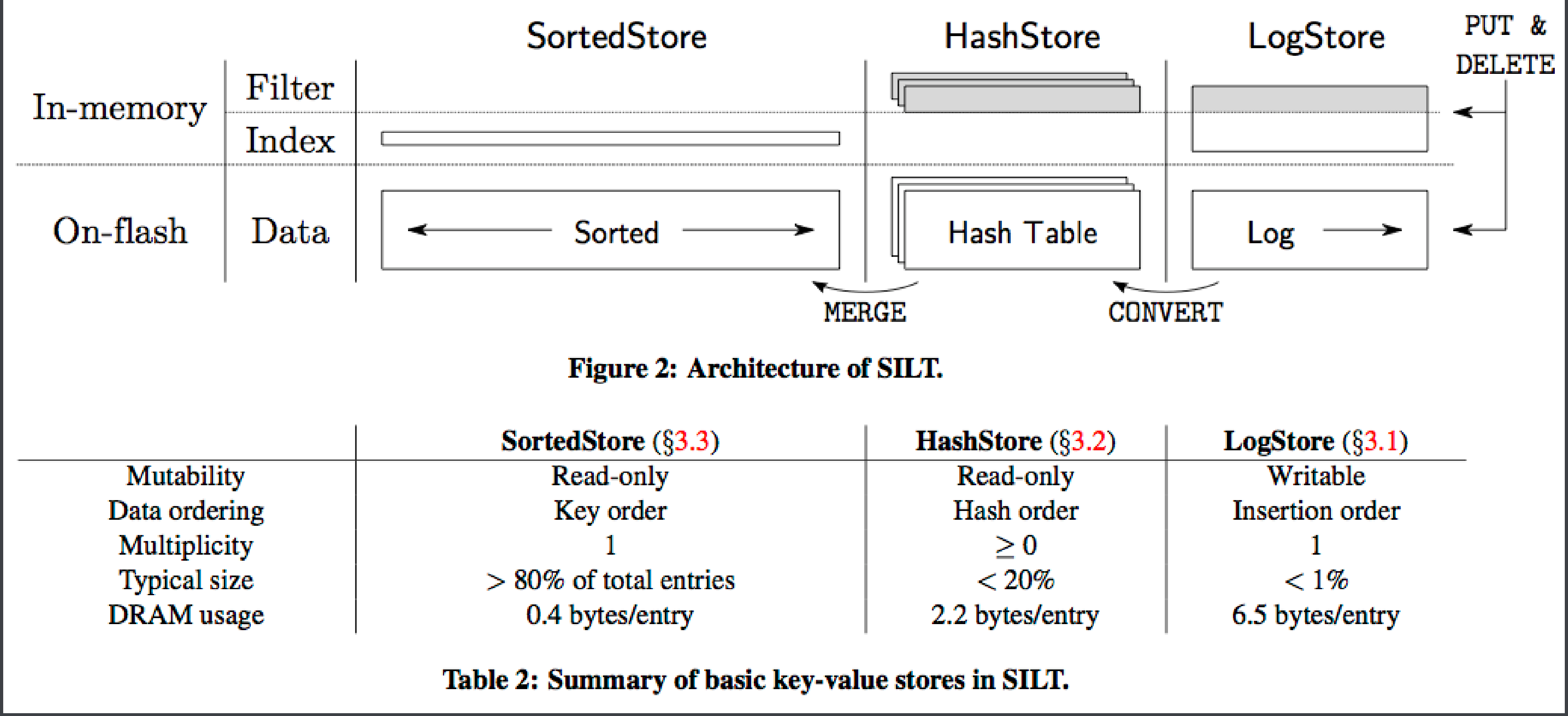
- Low # of flash reads per GET
- Sequential instead of Random Writes
- Memory efficient indexing
- Computation efficient indexing
- Effective use of flash space

Key Ideas

- Use three different KV stores
- Start in write-optimized store, move to memory-efficient store over time
- One flash read per lookup

Implementation

Overview



SortedStore
HashStore
LogStore
PUT & DELETE
In-memory
Filter
Index
On-flash
Data
Sorted
Hash Table
Log
MERGE
CONVERT

Figure 2: Architecture of SILT.

	SortedStore (§3.3)	**HashStore** (§3.2)	**LogStore** (§3.1)
Mutability	Read-only	Read-only	Writable
Data ordering	Key order	Hash order	Insertion order
Multiplicity	1	≥ 0	1
Typical size	> 80% of total entries	< 20%	< 1%
DRAM usage	0.4 bytes/entry	2.2 bytes/entry	6.5 bytes/entry

Table 2: Summary of basic key-value stores in SILT.

Log Store

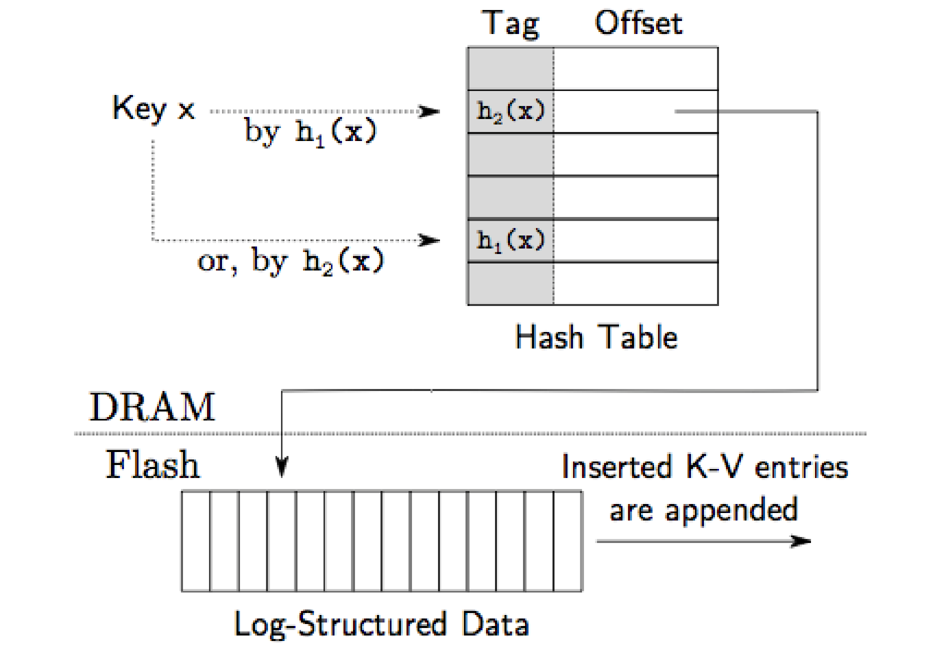


Figure 3: Design of LogStore: an in-memory cuckoo hash table (index and filter) and an on-flash data log.

- Cuckoo hash table index + flash log
- Ops written to log sequentially
- Convert to HashStore when index is full

Hash Store

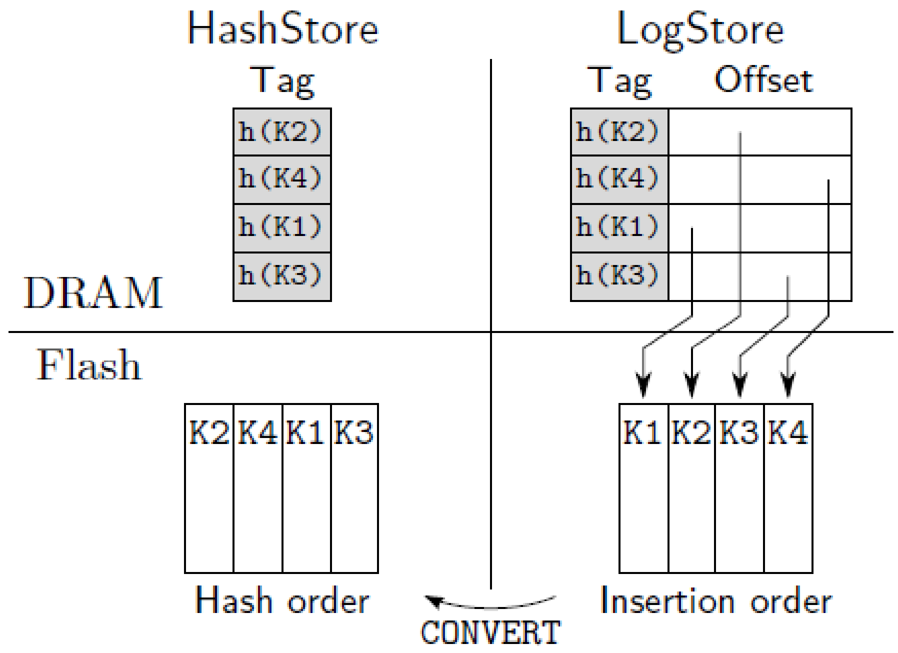
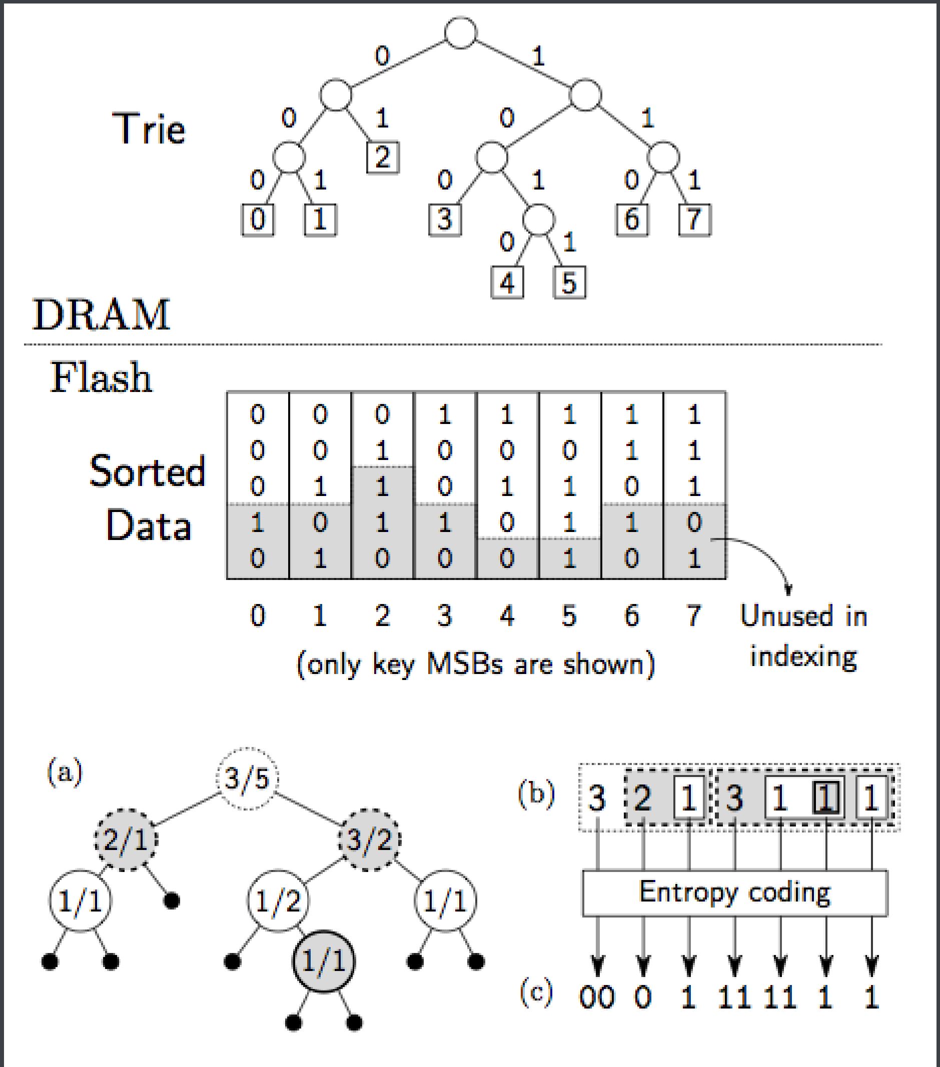


Figure 4: Convert a LogStore to a HashStore. Four keys K1, K2, K3, and K4 are inserted to the LogStore, so the layout of the log file is the insert order; the in-memory index keeps the offset of each key on flash. In HashStore, the on-flash data forms a hash table where keys are in the same order as the in-memory filter.

- DRAM tags preserve order on flash
- Lower memory overhead than LS
- Merge multiple into SortedStore at once

Sorted Store

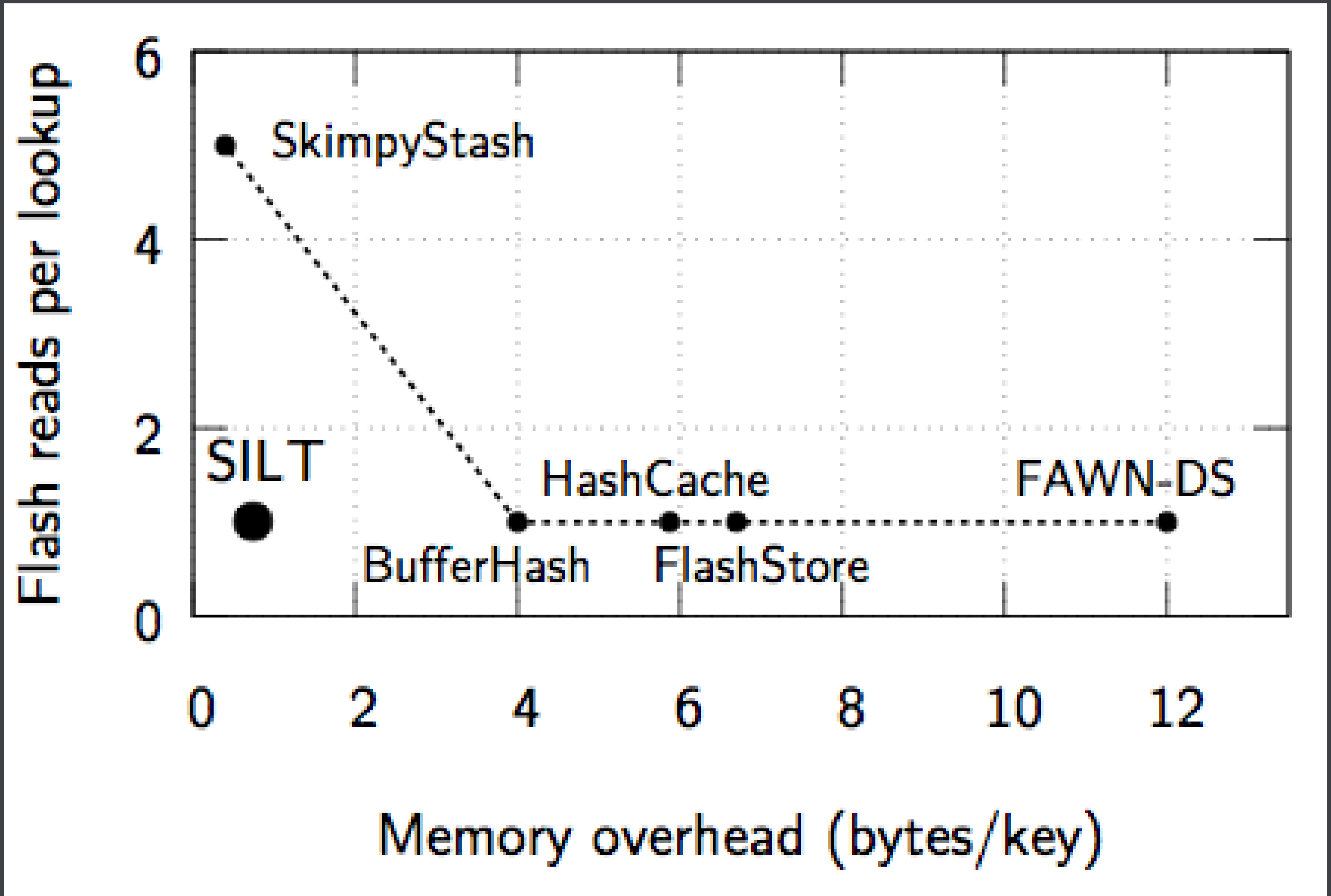


0	1	2	3	4	5	6	7
0	0	0	1	1	1	1	1
0	0	1	0	0	0	1	1
0	1	1	0	1	1	0	1
1	0	1	1	0	1	1	0
0	1	0	0	0	1	0	1

- DRAM trie index + sorted KVs on flash
- Index compressed with entropy coding
- Stores ~80% of data

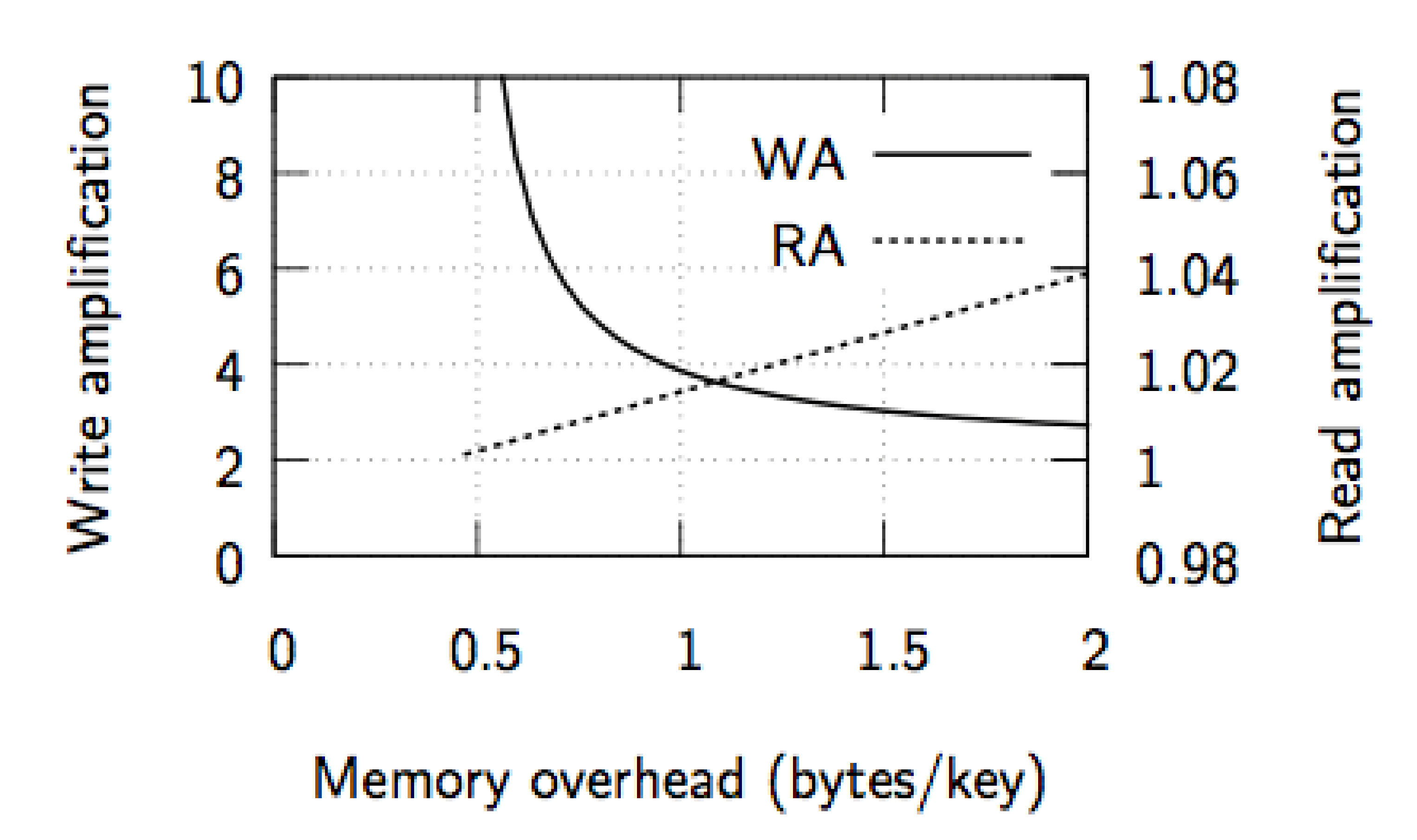
Analysis

Evaluation



SILT vs. other KV stores

Tradeoffs



Improving one metric will cost you in another

Takeaways

- Composition of existing systems can make a better system
- Avoiding amplification is hard
- System designers need to choose which metric to focus on

Thank You!