Microsoft

The Bw-Tree: A B-tree for New Hardware Platforms

Justin Levandoski
Researcher
Microsoft Research

Joint work with David Lomet and Sudipta Sengupta

Microsoft Research
Faculty Summit

An Alternate Title

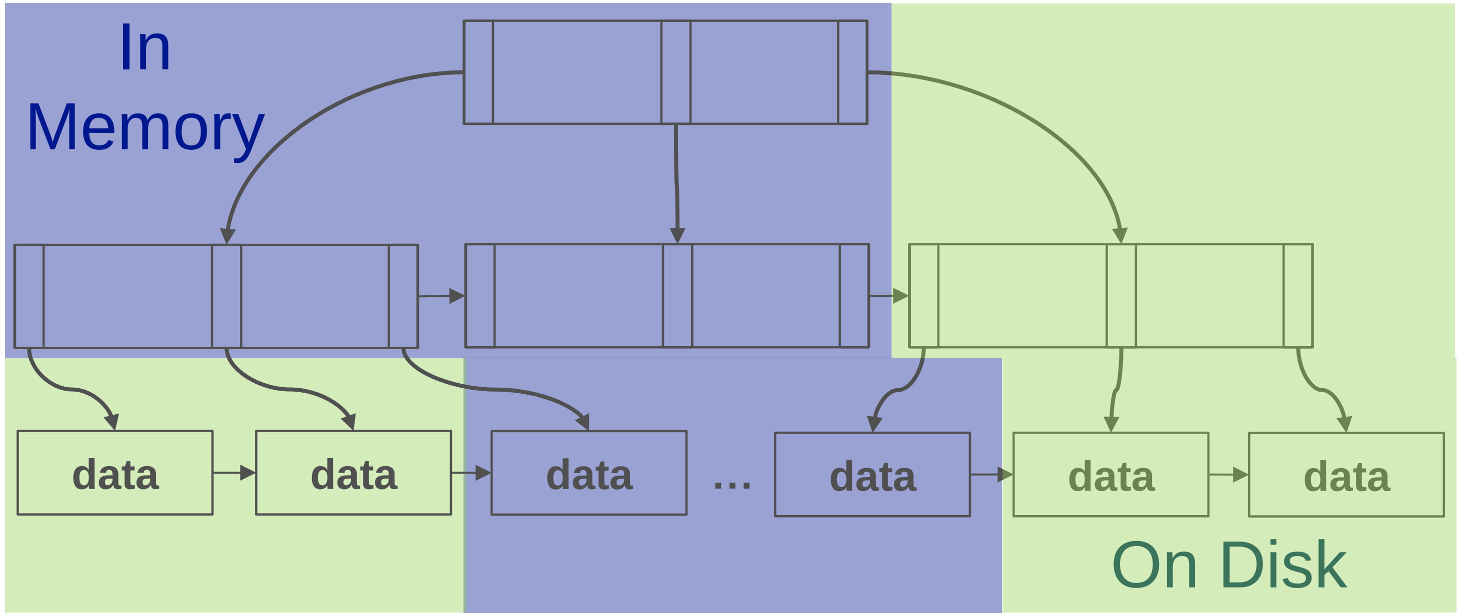
"The BW-Tree: A Latch-free, Log-structured B-tree for Multi-core Machines with Large Main Memories and Flash Storage"

BW = "Buzz Word"

The Buzz Words (1)

B-tree

Key-ordered access to records
Efficient point and range lookups
Self balancing
B-link variant (side pointers at each level)



The Buzz Words (2)

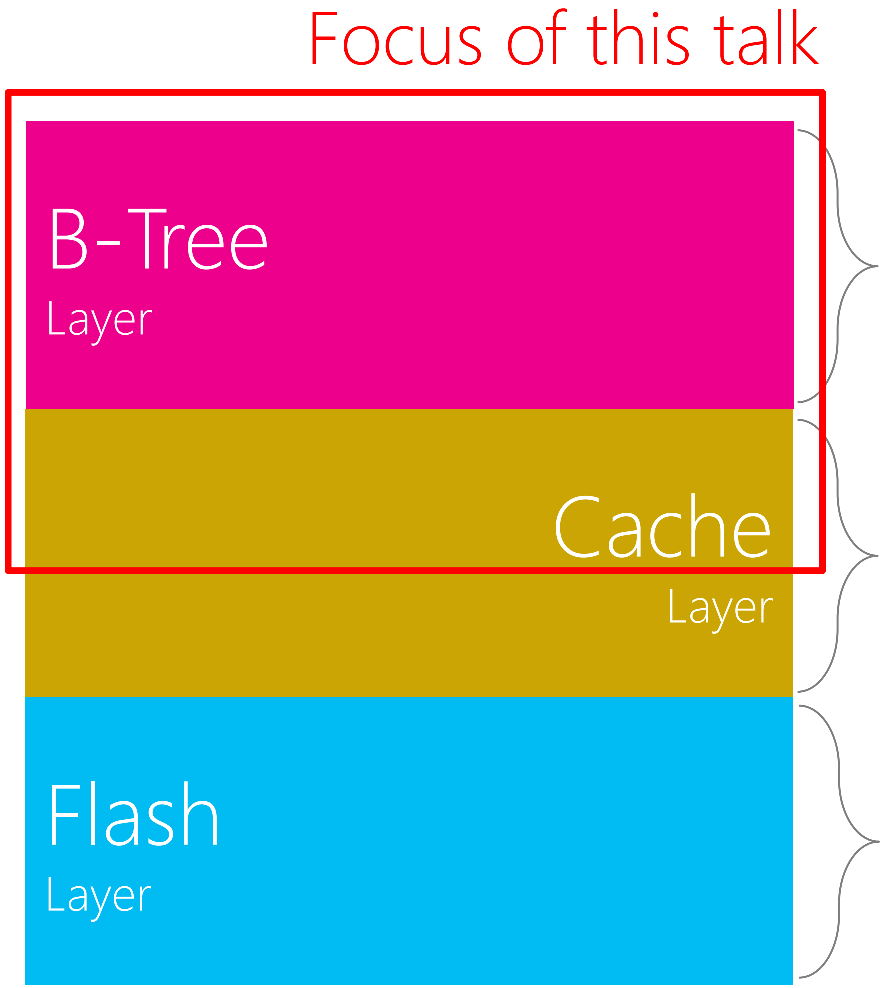
Multi-core + large main memories

- Latch (lock) free
  - Worker threads do not set latches for any reason
  - Threads never block
  - No data partitioning
- No updates in place
  - “Delta” updates
  - Reduces cache invalidation

Flash storage

- Good at random reads and sequential reads/writes
- Bad at random writes
- Use flash as append log
- Implements novel log-structured storage layer over flash

Bw-Tree Architecture



- Create-Read-Update-Delete ("CRUD") API
- B-tree search/update logic
- In-memory pages only

- Logical page abstraction for B-tree layer
- Brings pages from flash to RAM as necessary

- Sequential writes to log-structured storage
- Flash garbage collection

Multiple Deployment Scenarios

In-memory latch-free B-tree (Hekaton)

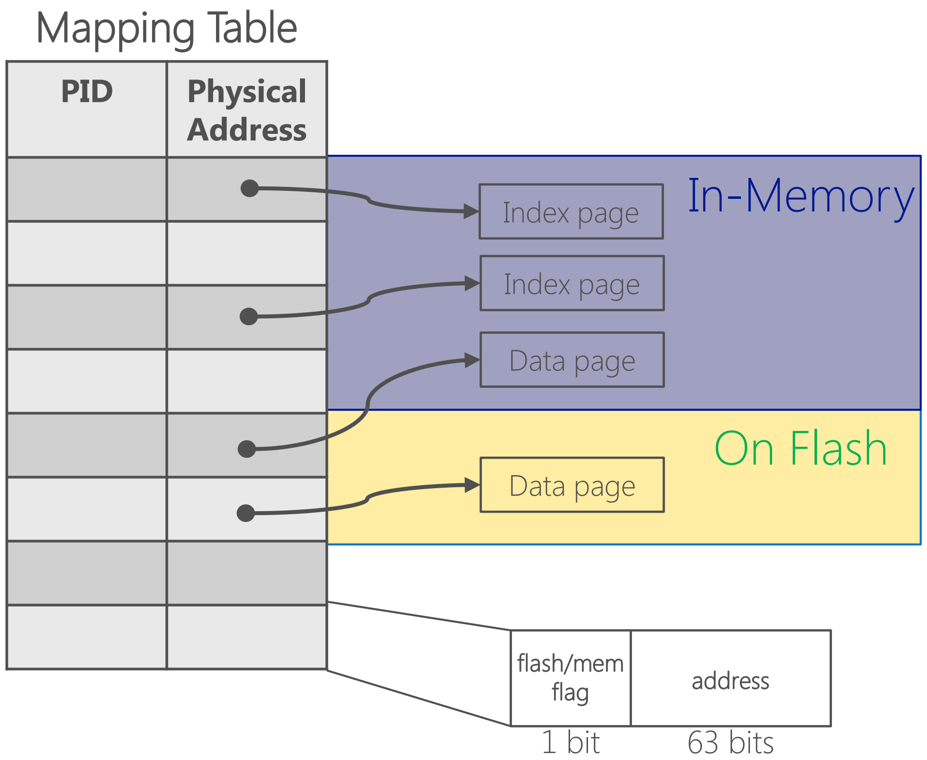
High-performance standalone atomic record store (e.g., LevelDB)

Data component (DC) in a decoupled "Deuteronomy" style transactional system

http://research.microsoft.com/deuteronomy/

Bw-Tree Latch Freedom

Mapping Table and Logical Pages

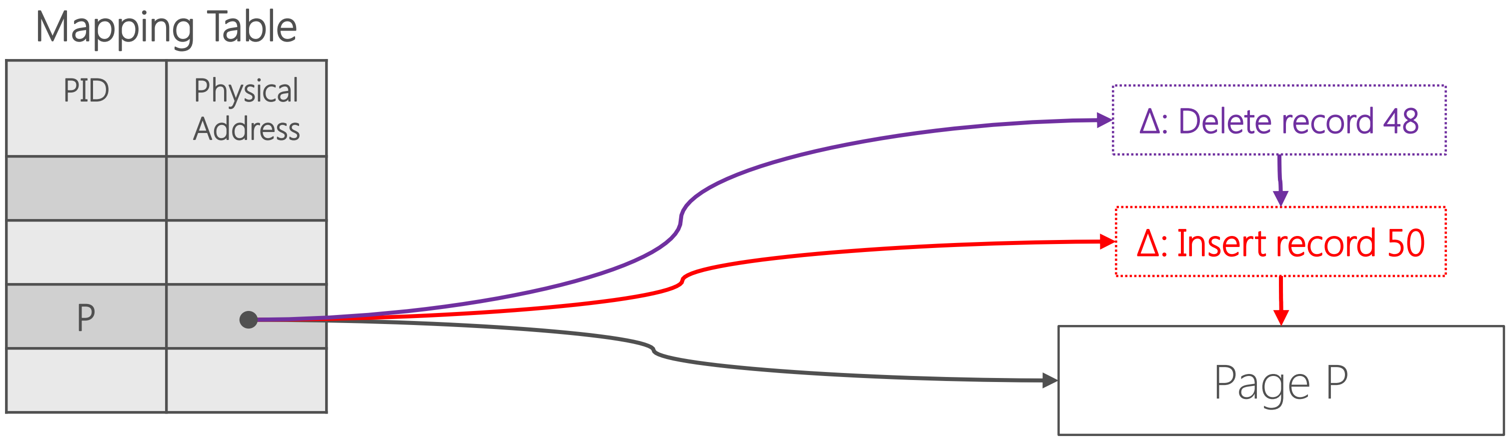


Mapping table translates logical page id to physical location

Important for latch-free behavior and log structuring

Isolates updates to a single page

Delta Updates

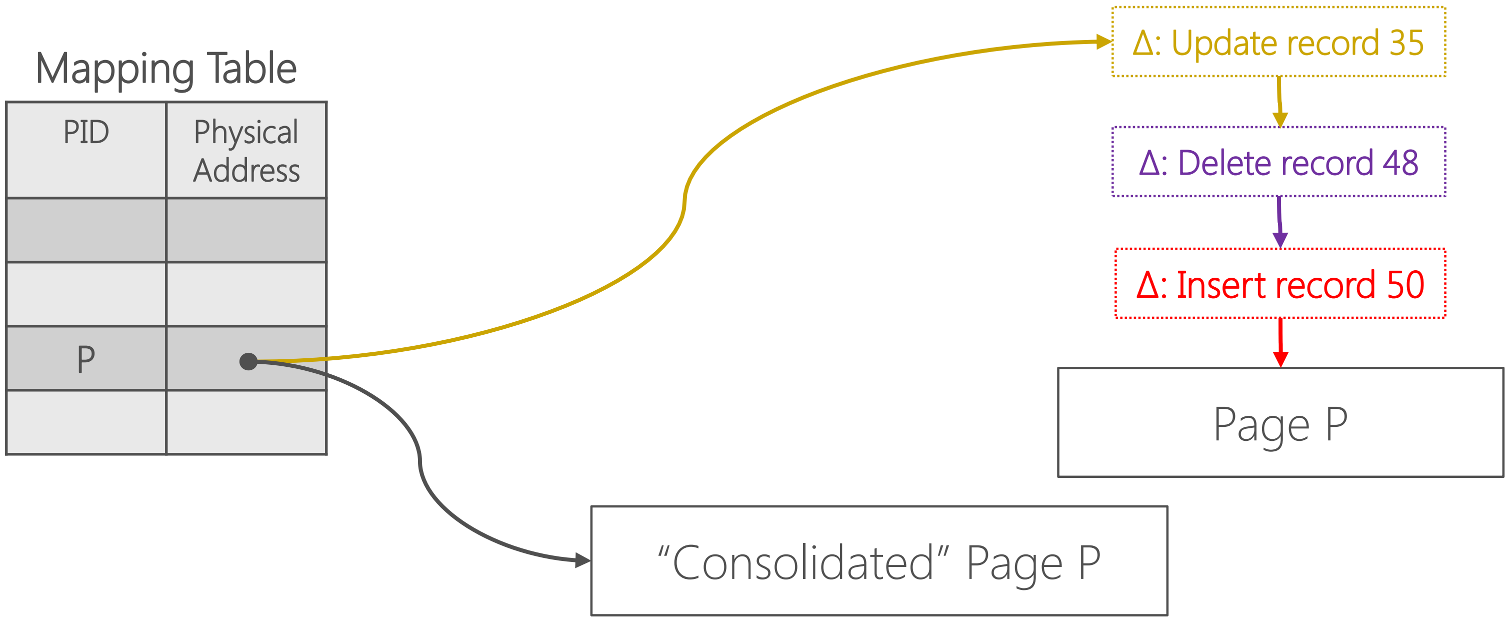


Each update to a page produces a new address (the delta)

Delta physically points to existing "root" of the page

Install delta address in the mapping table using compare and swap

Delta consolidation



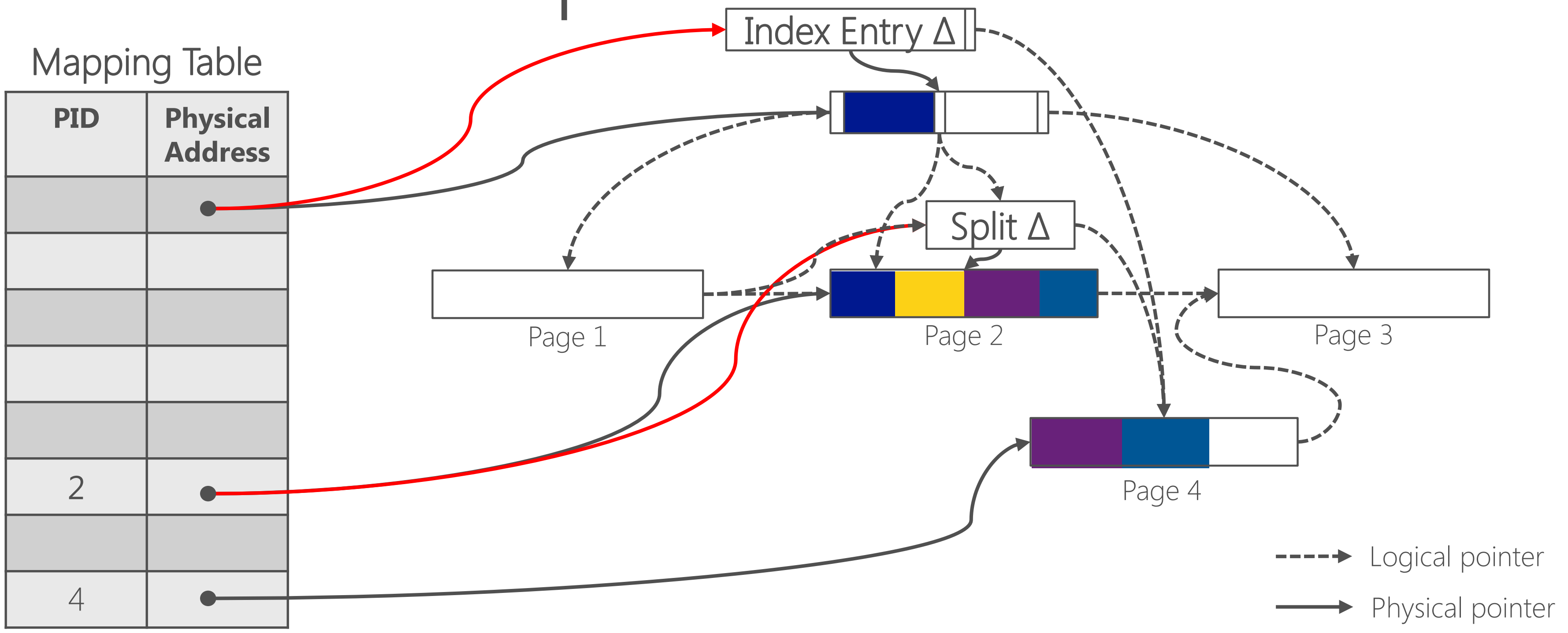
Long delta chains degrade search performance

Delta consolidation creates new search-optimized page

Consolidation piggybacked onto regular operations

Old page state becomes garbage (protected by epochs)

Latch-Free Node Splits



B-link structure allows us to “half split” without latching

Splitting requires two atomic steps

1. Install split at child level by creating new page
2. Install index term for new page at the parent

Performance Highlights

Performance Highlights

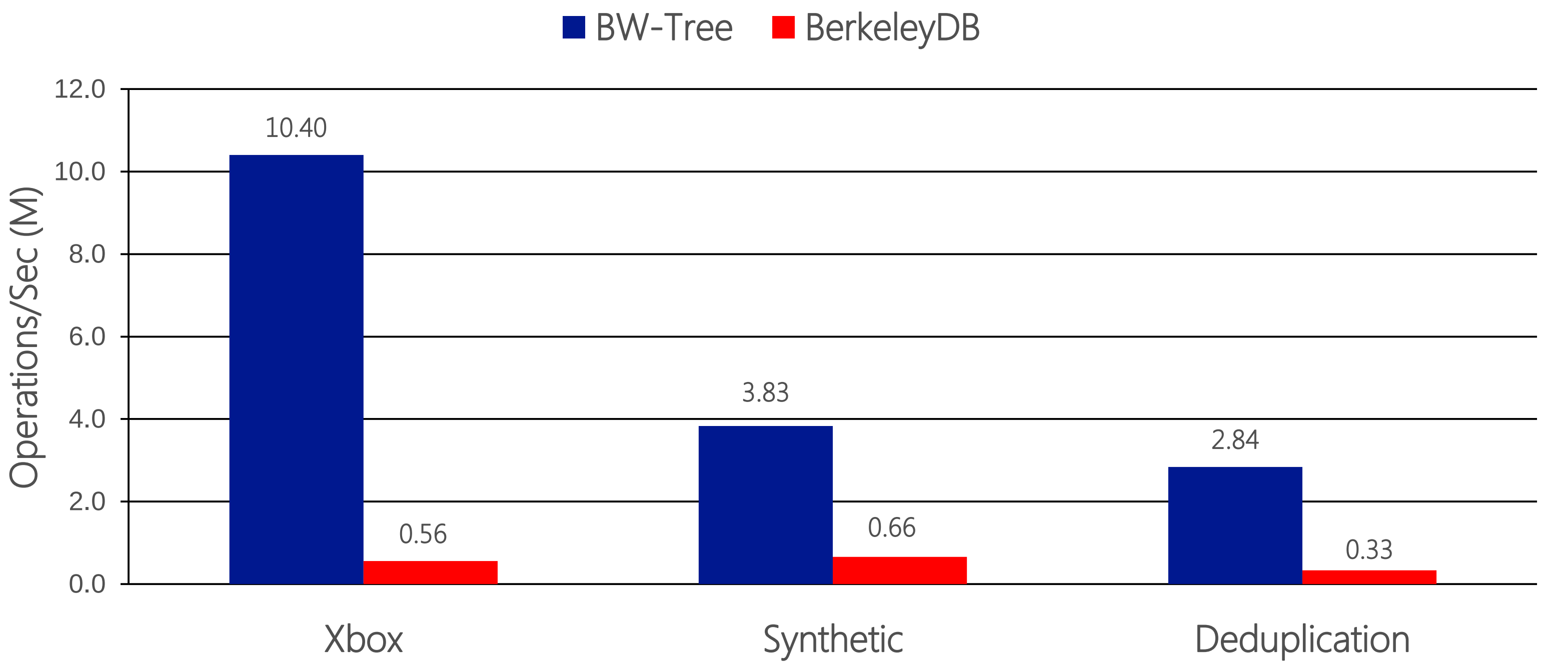
Experimented against

BerkeleyDB standalone B-tree (no transactions)
Latch-free skiplist

Workloads

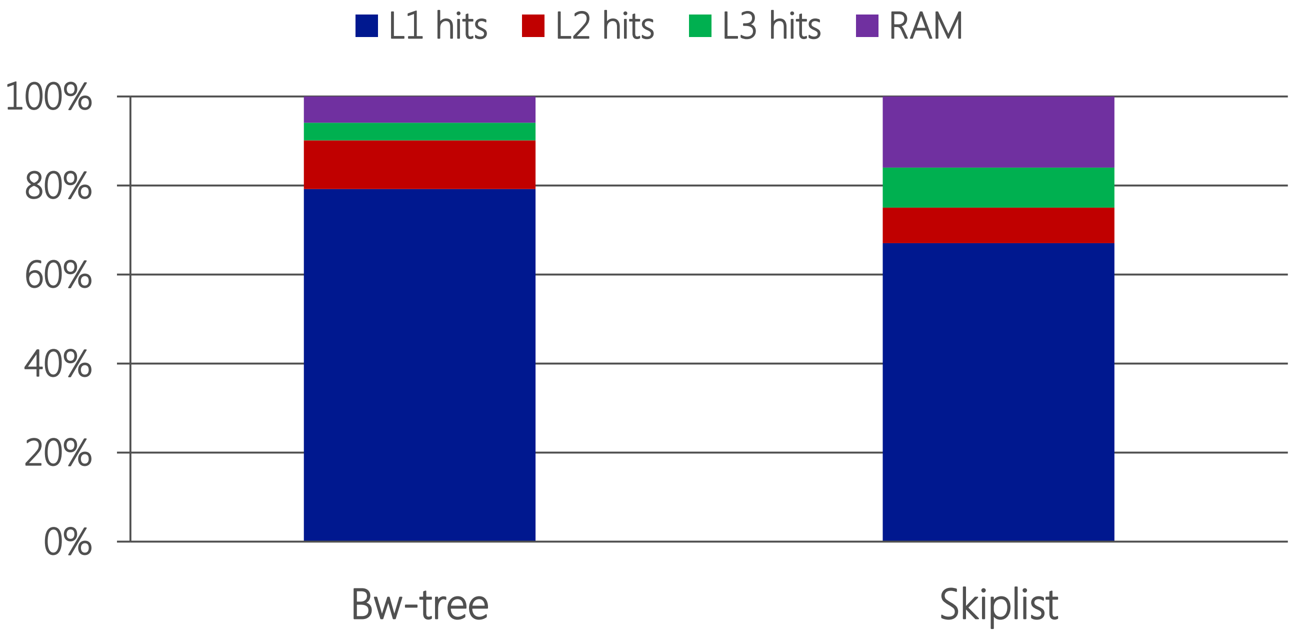
- XBOX
  - 27M get/set operations from XBox Live Primetime
  - 94 bytes keys, 1200 byte payloads, read/write ratio of 7:1
- Enterprise storage deduplication
  - 27M deduplication chunks from real enterprise trace
  - 20-byte keys (SHA-1 hash), 44-byte payload, read-write ratio of 2.2:1
- Synthetic
  - 42M operations with keys randomly generated
  - 8-byte keys, 8-byte payloads, read-write ratio of 5:1

vs BerkeleyDB



vs Skiplists

	Bw-Tree	Skiplist
Synthetic workload	3.83M Ops/Sec	1.02 M Ops/Sec



Microsoft