ZipCache: A DRAM/SSD Cache with Built-in Transparent Compression

Rui Xie[1], Linsen Ma[1], Alex Zhong[2], Feng Chen[3], Tong Zhang[1]

Rensselaer Polytechnic Institute[1]

Harker School[2]

Louisiana State University[3]

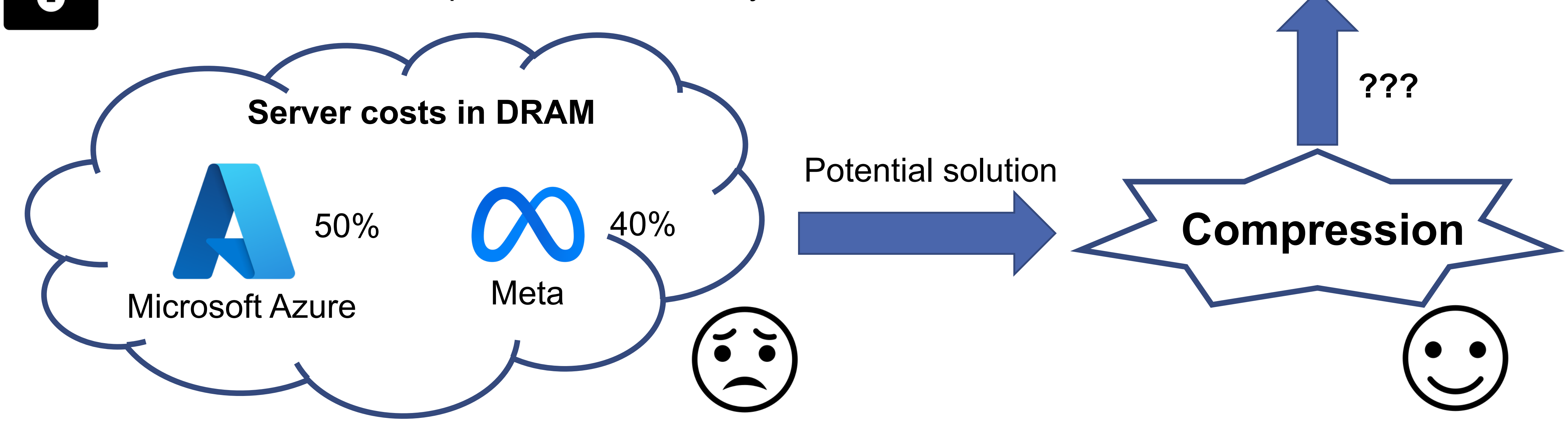
10/01/2024

Motivation

- Growing demand for larger KV caches
- Hardware constraints in expanding cache capacity
- Underutilization of compression in current systems

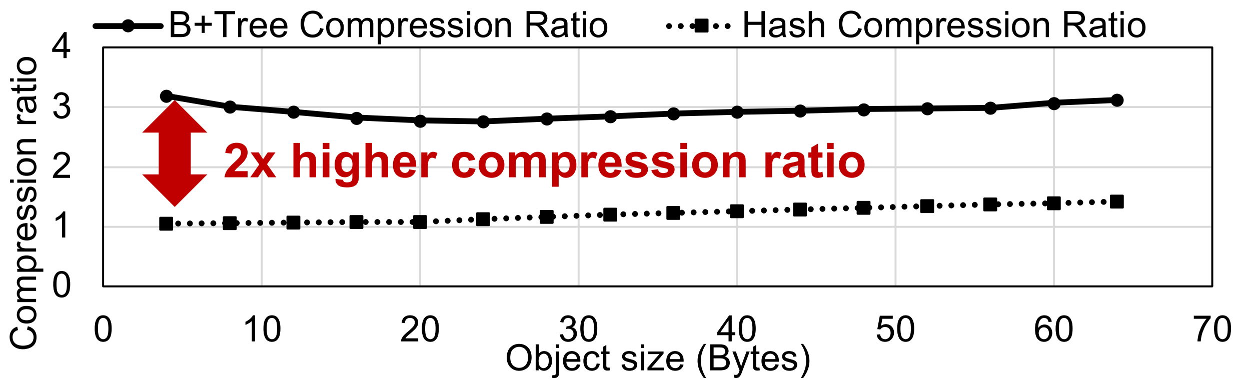
Key Challenges:

1. Hash index causes random data placement
2. Read & Write amplification
3. Inefficient decompression
4. Heavy computational overhead



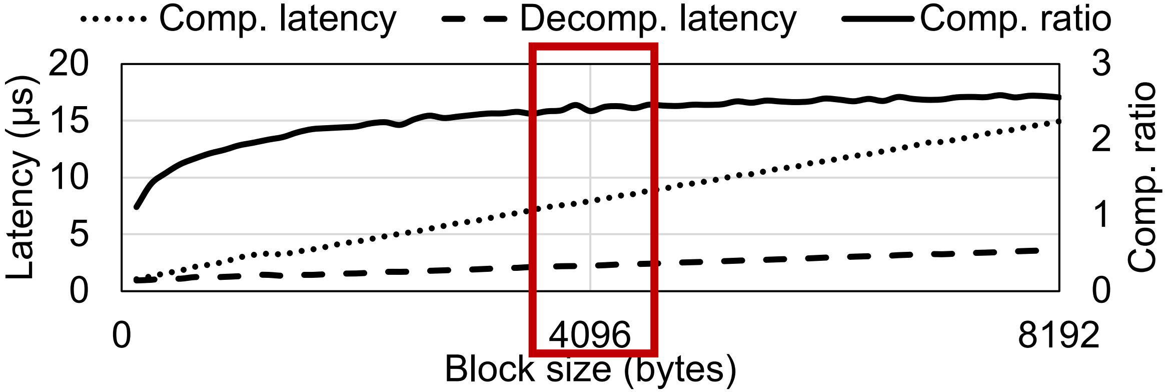
B+ tree vs Hash index

- 2x higher compression ratio than hash index
- Overall data access latency is comparable since decompression latency dominated



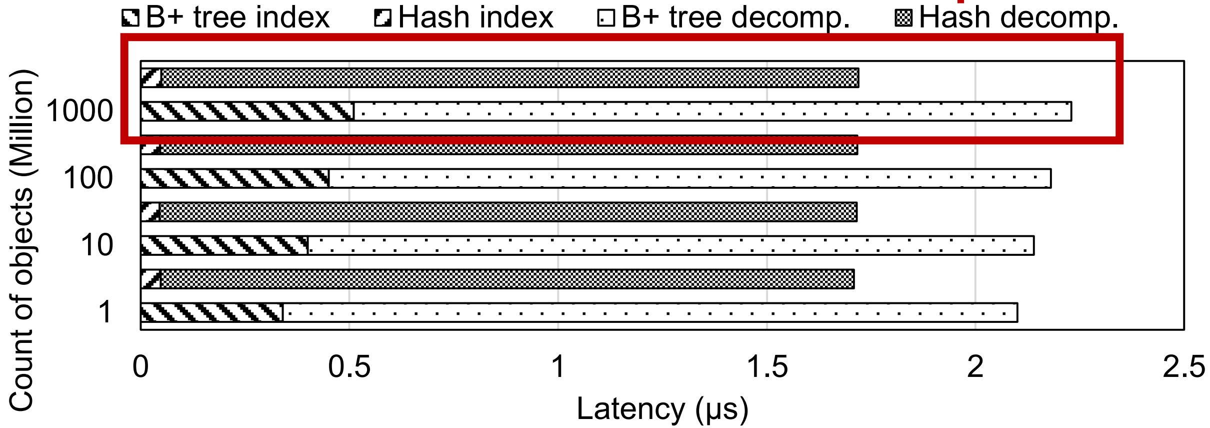
2x higher compression ratio

(a) Compression ratio comparison



We choose 4KB as a block

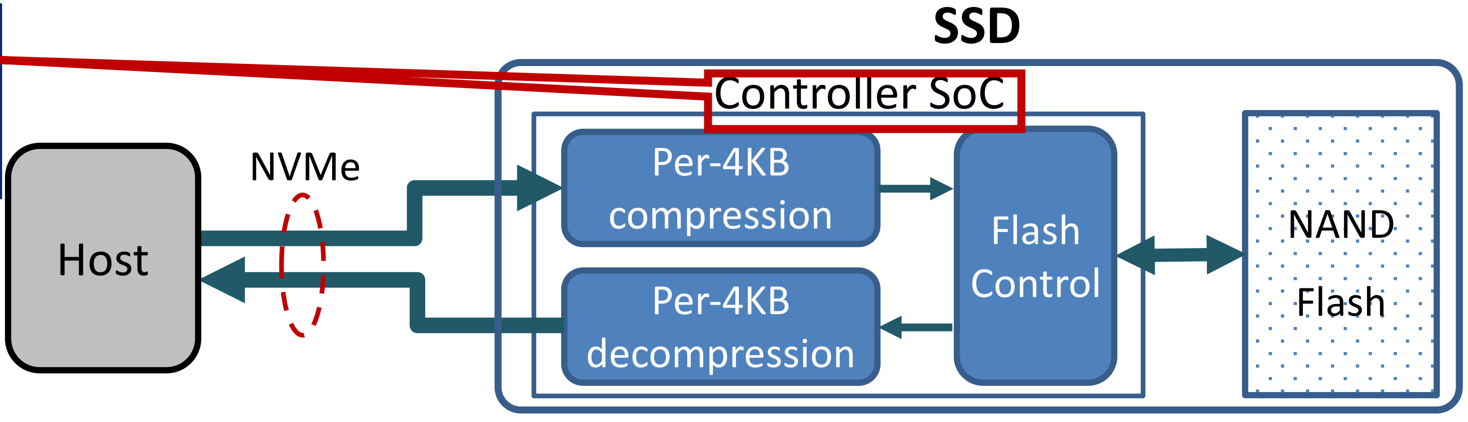
Comparable latency



(b) Latency comparison

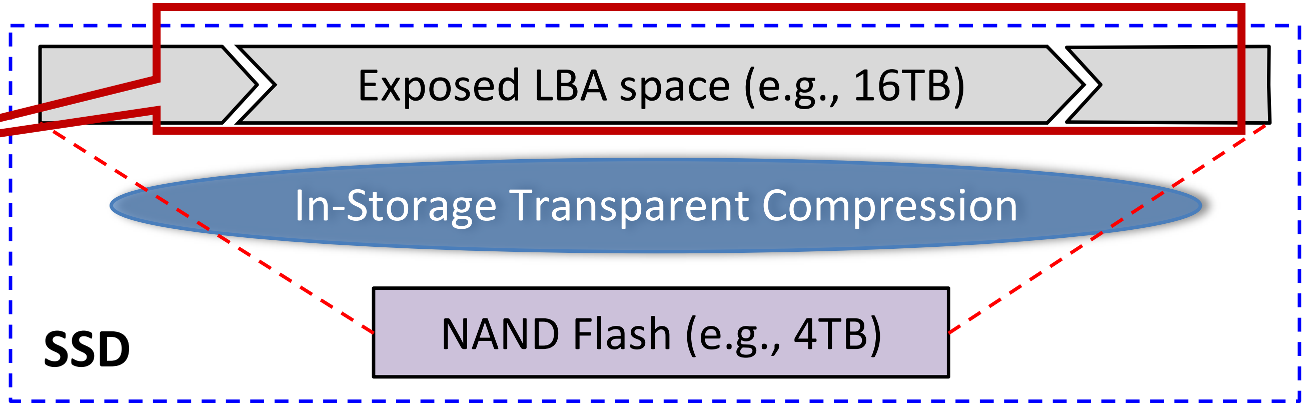
SSD: In-Storage Transparent Compression

- Hardware accelerated compression in SSDs
- SSD controller de/compression at I/O path
- No host CPU intervention



(a)

Larger logical storage space


Exposed LBA space (e.g., 16TB)
In-Storage Transparent Compression
NAND Flash (e.g., 4TB)
SSD

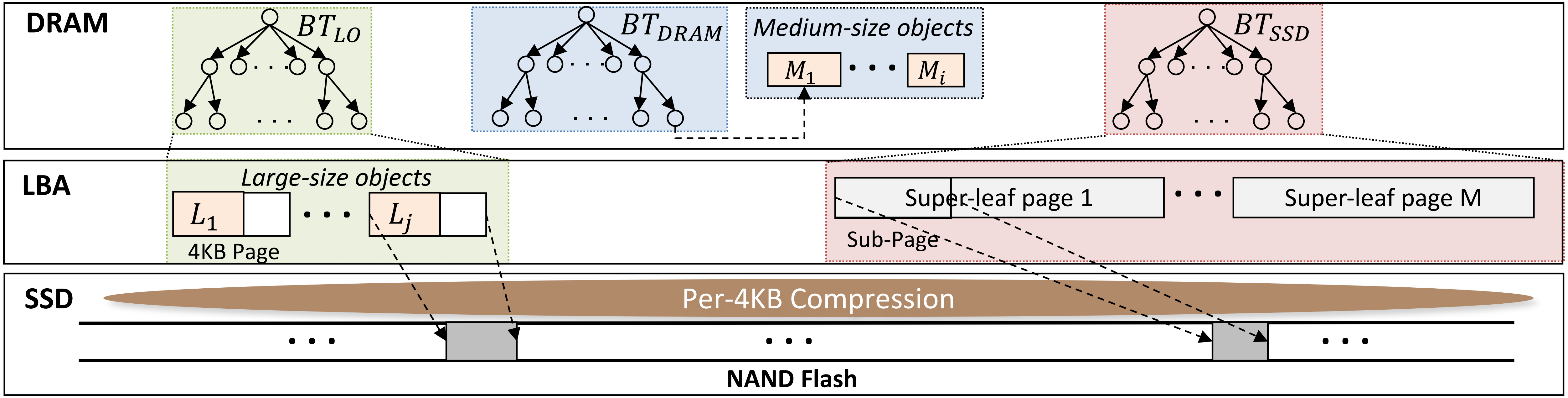
(b)

ZipCache Overview

A set of pre-defined objects thresholds:

Type	**Size range**	**Where to store**
Tiny-size	< 128B	DRAM and SSD
Medium-size	128B – 2KB	DRAM and SSD
Large-size	> 2KB	SSD

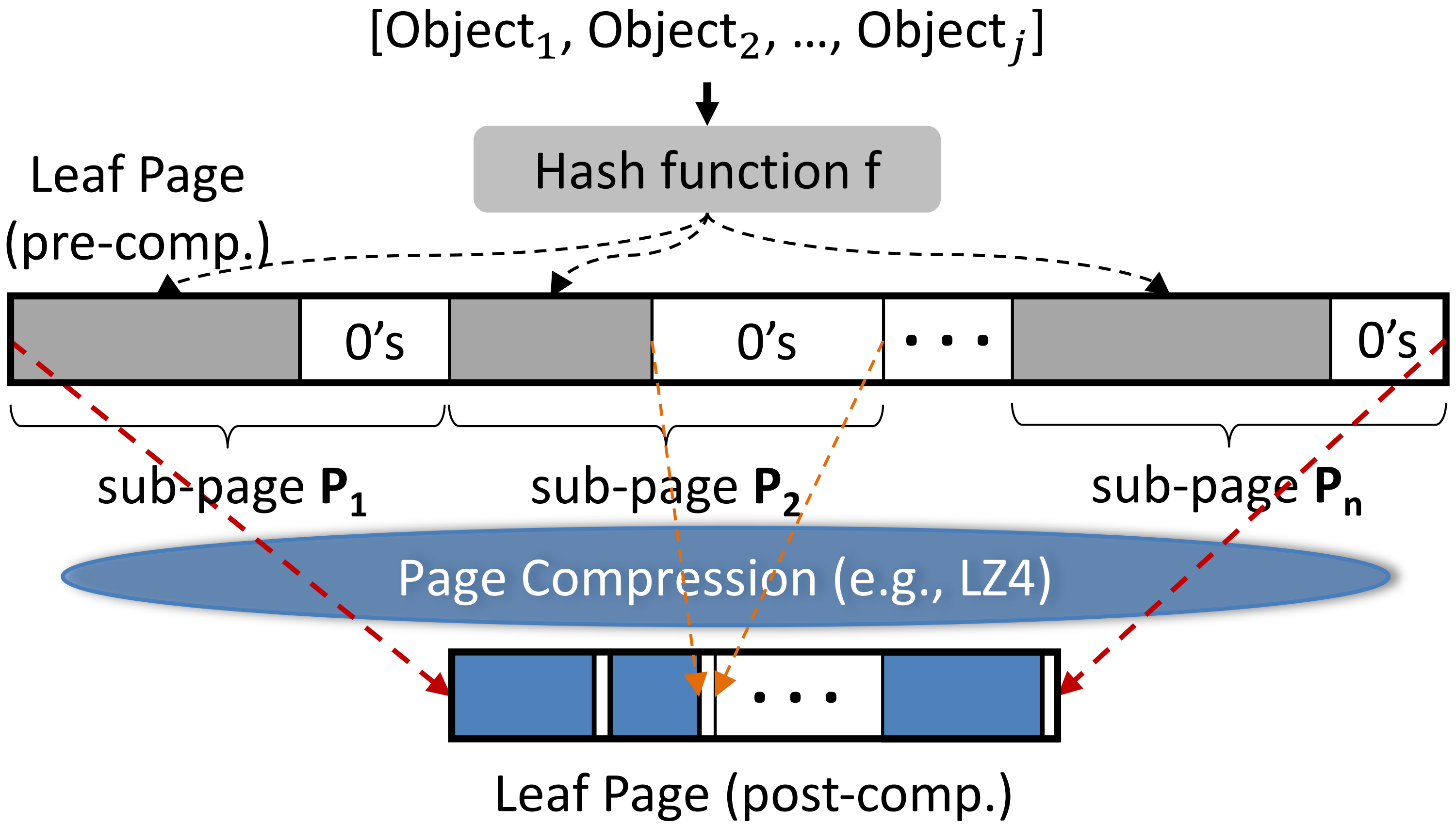
- BT_{DRAM} for DRAM cache
- BT_{SSD} for SSD cache
- BT_{LO} for indexing large-size objects



DRAM Cache Tier

Key features:

1. **Decompression early termination**
2. **Adaptive compression bypassing**
3. **Per-page write buffering**



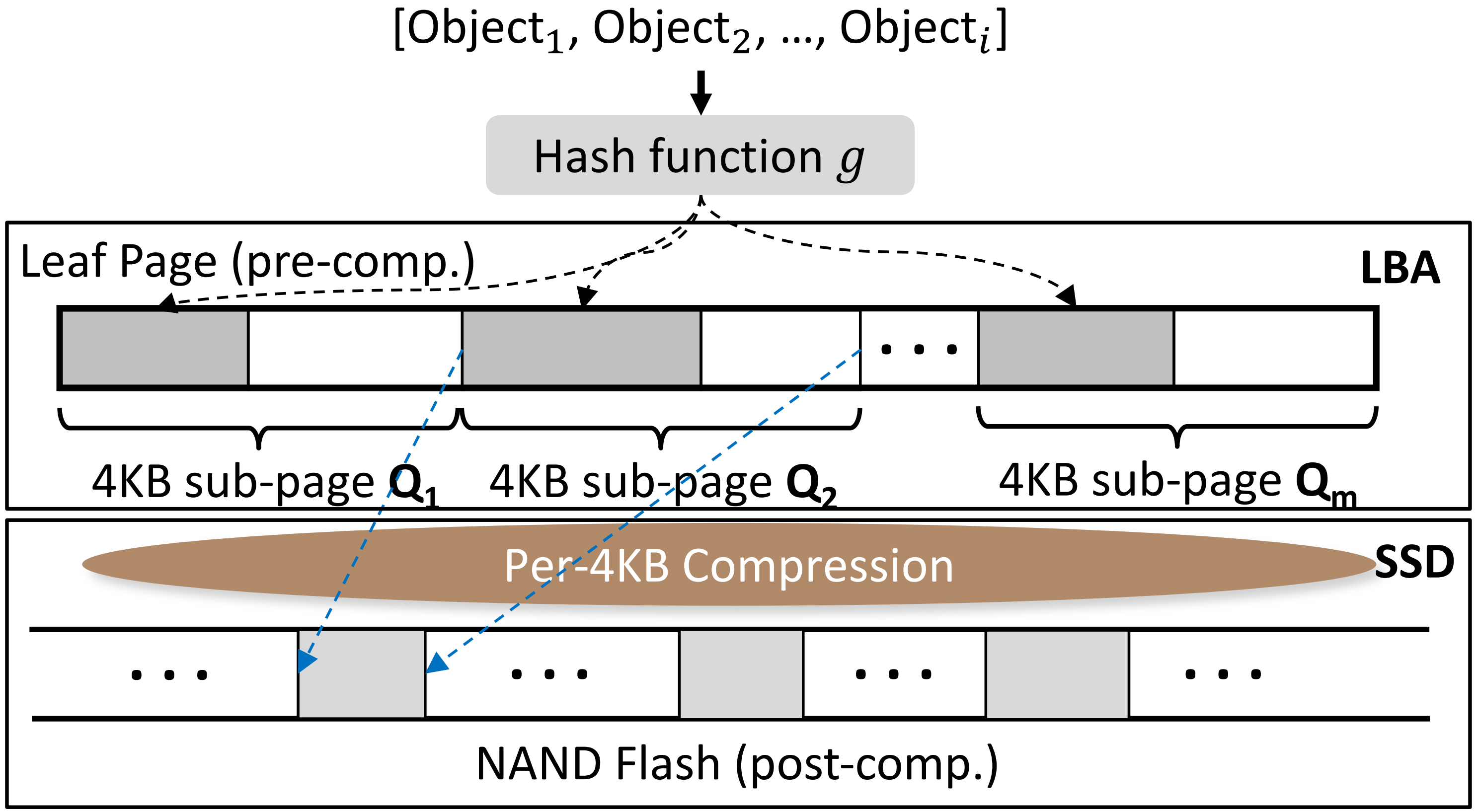
Performance benefits:

1. **Reduced latency** by minimizing decompression time
2. **Higher DRAM hit ratio**, improving cache performance

SSD Cache Tier

Key features:

1. **Intra-page object hashing**
2. **Page-based DRAM-to-SSD eviction**
3. **Sub-page under-filling**

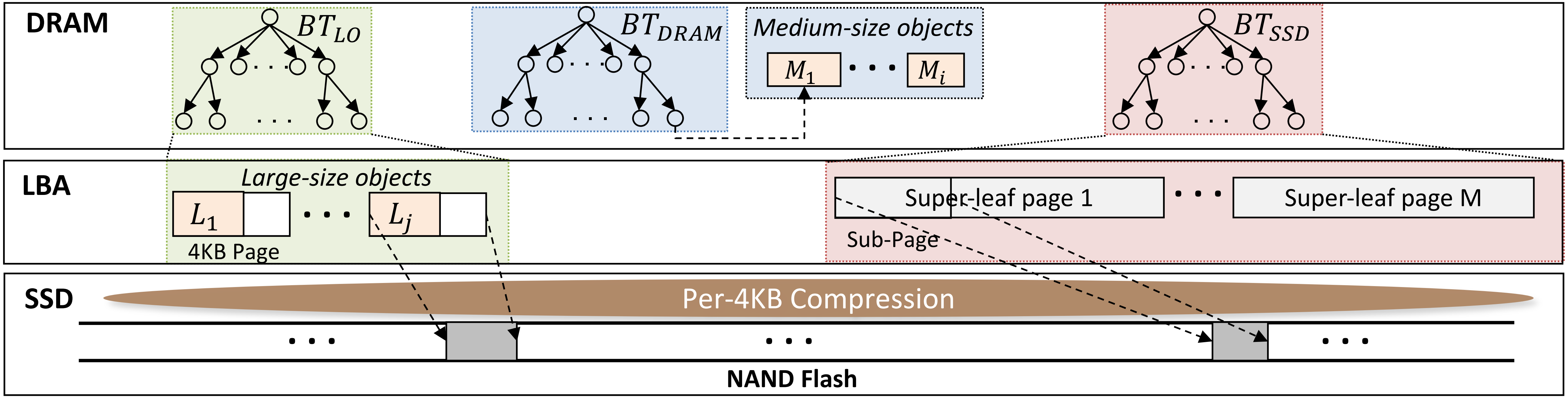


Performance benefits:

1. Up to 4x logical storage expansion
2. Reduced write amplification by up to 26.2x

Major Operations

- **GET:** Search through order $BT_{DRAM} \rightarrow BT_{SSD} \rightarrow BT_{LO}$
- **SCAN:** Range scans over 3 B+ trees
- **PUT:**
  - *tiny/medium* inserted to DRAM cache tier, and search BT_{LO} for possible deletion (*large* with same key)
  - *Large* written to SSD and pointer inserted to BT_{LO}, (*tombstone* for same key in DRAM cache tier)
- **DELETE:** insert *tombstone* to DRAM cache tier and search BT_{LO} for possible deletion



Performance result

Workload locality

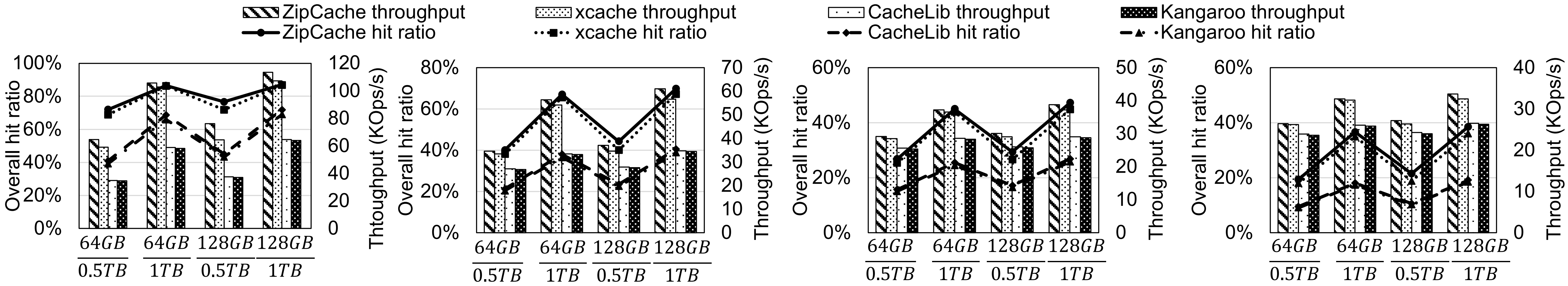
Strong	Moderate	Weak	Zero
80%→8%	80%→20%	80%→64%	*Random*

80% cache access requests hit 20% of all cache objects

Baseline:

- ✓ Xcache: SSD compression
- ✓ CacheLib: no compression
- ✓ Kangaroo: a variant of CacheLib for reducing SSD write amplification

Experiment setting: 6TB working set size



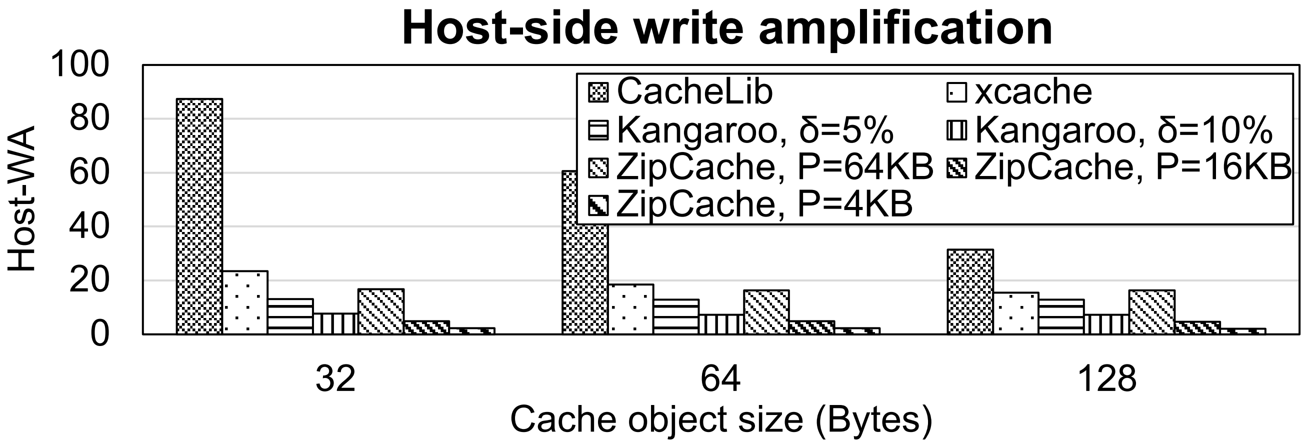
(a) Strong locality

(b) Moderate locality

(c) Weak locality

(d) Zero locality

SSD Write Amplification



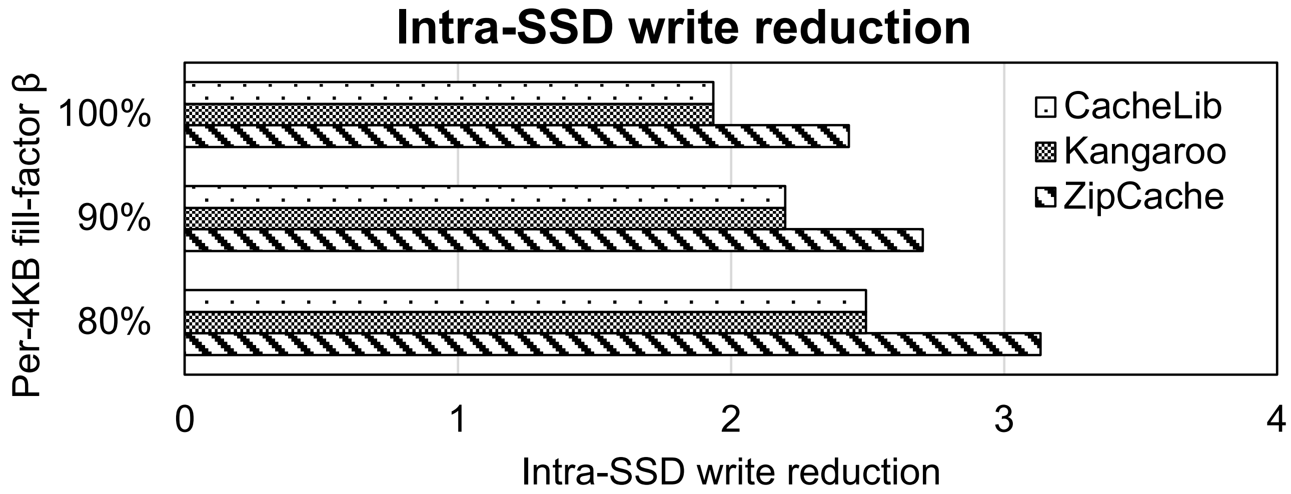
$$WA = \frac{WA_{host}}{WR_{NAND}}$$

Host-side write amplification

Intra-SSD compression

Baseline:

- ✓ CacheLib: Hash to 4KB SSD page
- ✓ Kangaroo: Apply write-ahead log to amortize WA
- ✓ Xcache: Log-structure merge tree

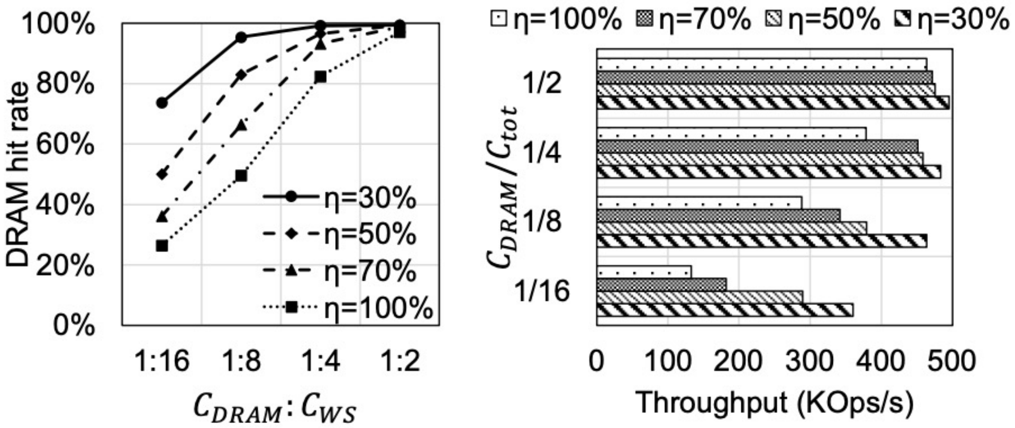


- ➢ ZipCache and Kangaroo have comparable host-side WA
- ➢ ZipCache achieves lower intra-ssd write reduction

Reduce WA up to 26.2x

Sensitivity Study

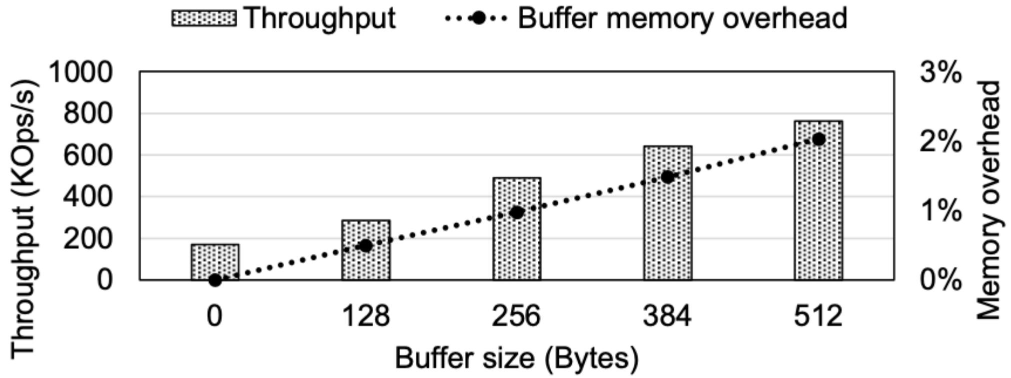
Compressibility



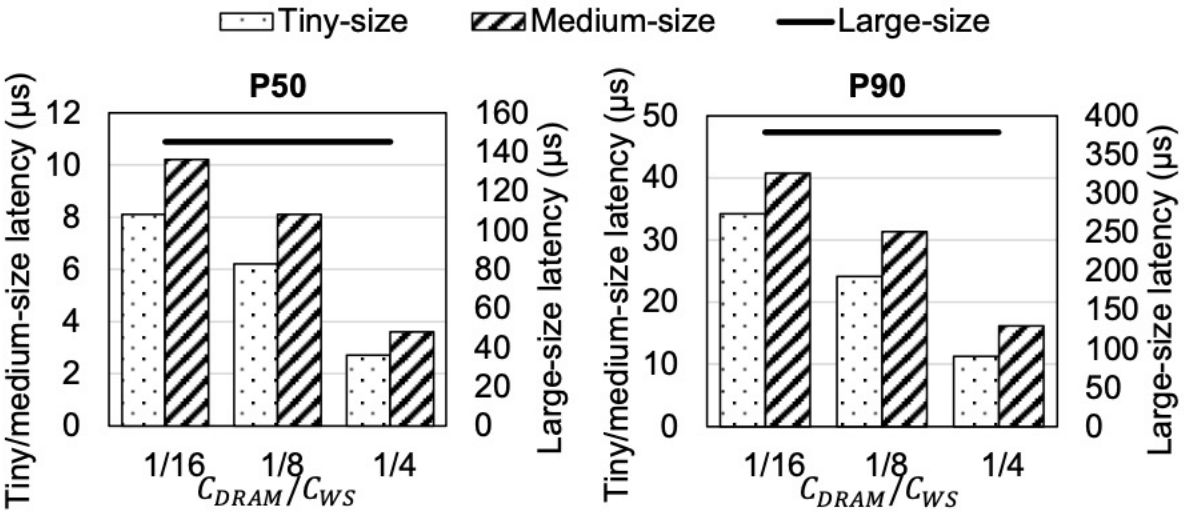
(a) DRAM tier hit ratio

(b) Throughput

Write buffer size



Cache object size (GET latency)



Thanks for listening!

- **ZipCache** integrates **compression** in key-value caches to improve performance
- Keys:
  - **B+ Tree indexing**, **transparent SSD compression**, and **early decompression termination**
- **Performance**:
  - Up to **72.4% higher throughput**, **42.4% lower latency**
  - Reduces **write amplification** by **26.2x**