Compress Objects, Not Cache Lines: An Object-Based Compressed Memory Hierarchy

Paper link:
https://bit.ly/zippads
Lightning video link:
https://bit.ly/zippads-lightning

Po-An Tsai and Daniel Sanchez
{poantsai, sanchez}@csail.mit.edu

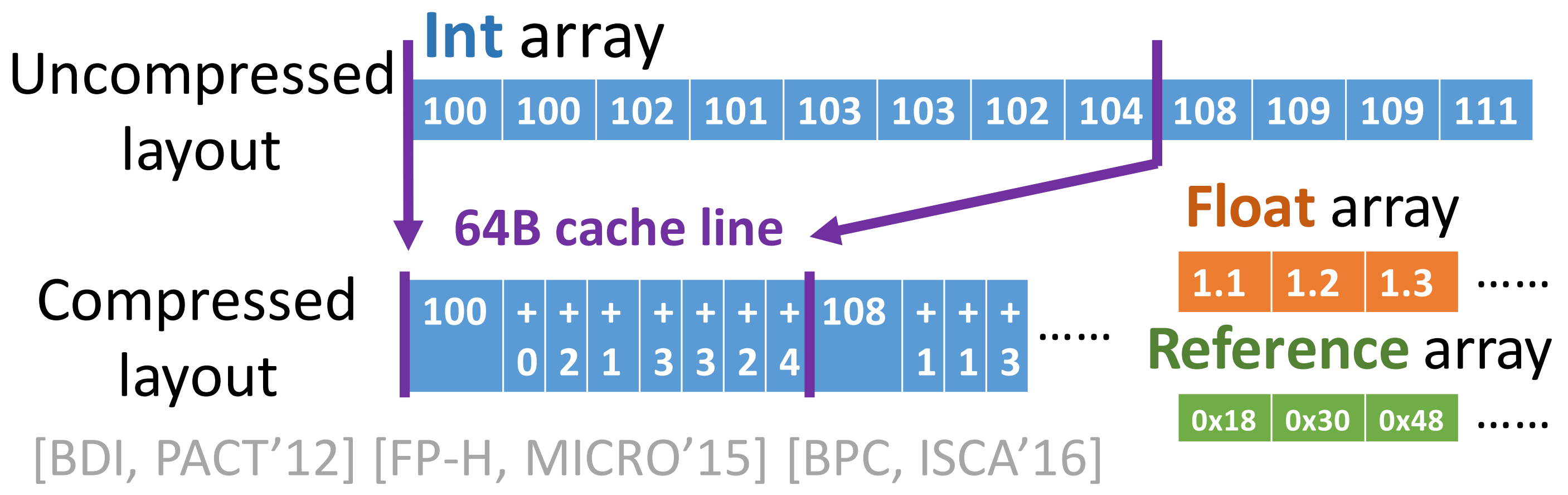
Background and Motivation

1. Compressed memory hierarchies require uncompressed-to-compressed address translation

Core issues loads/stores to *uncompressed address*

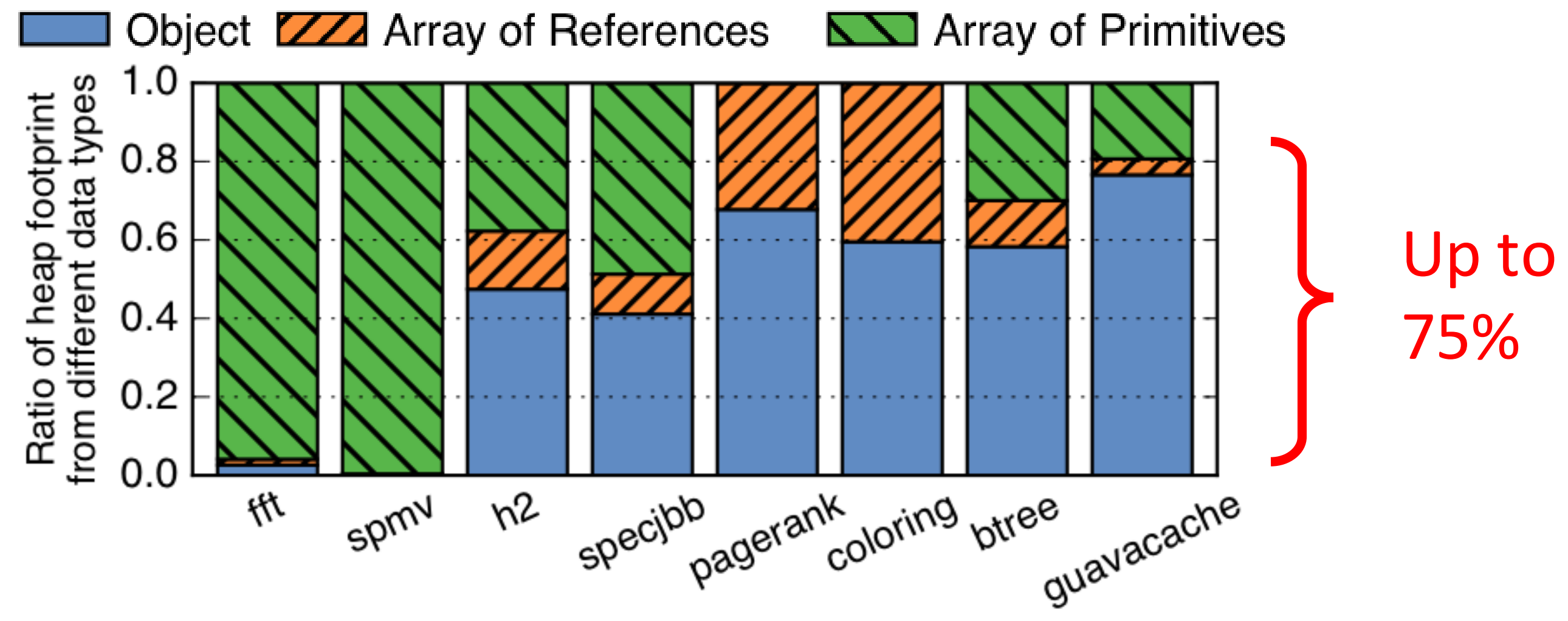
Data is stored in *compressed address*

2. Prior compression algorithms focus on compressing fixed-size cache lines and only work well for regular memory layout (e.g., arrays)



[BDI, PACT'12] [FP-H, MICRO'15] [BPC, ISCA'16]

3. Many programs mainly store objects in main memory and their layout is therefore irregular



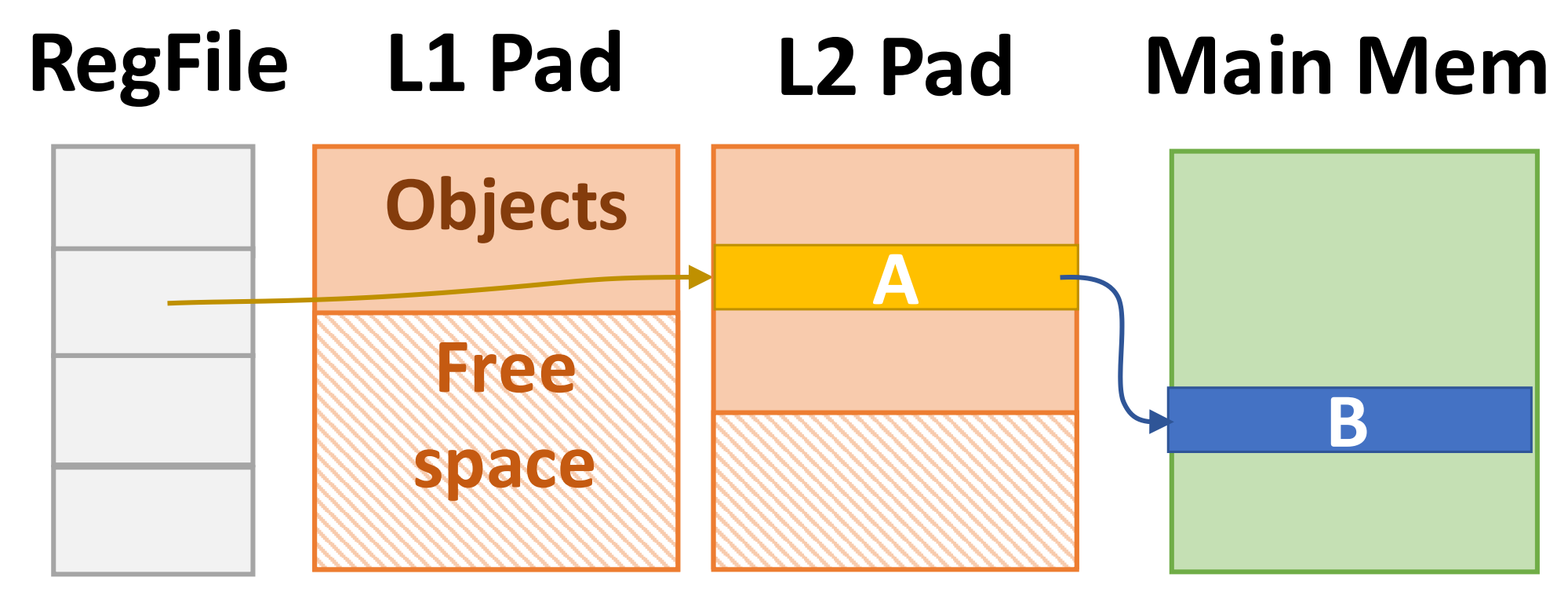
Objects, not cache lines, are the natural unit of compression!

Insight 1: Object-based applications always follow pointers to access objects

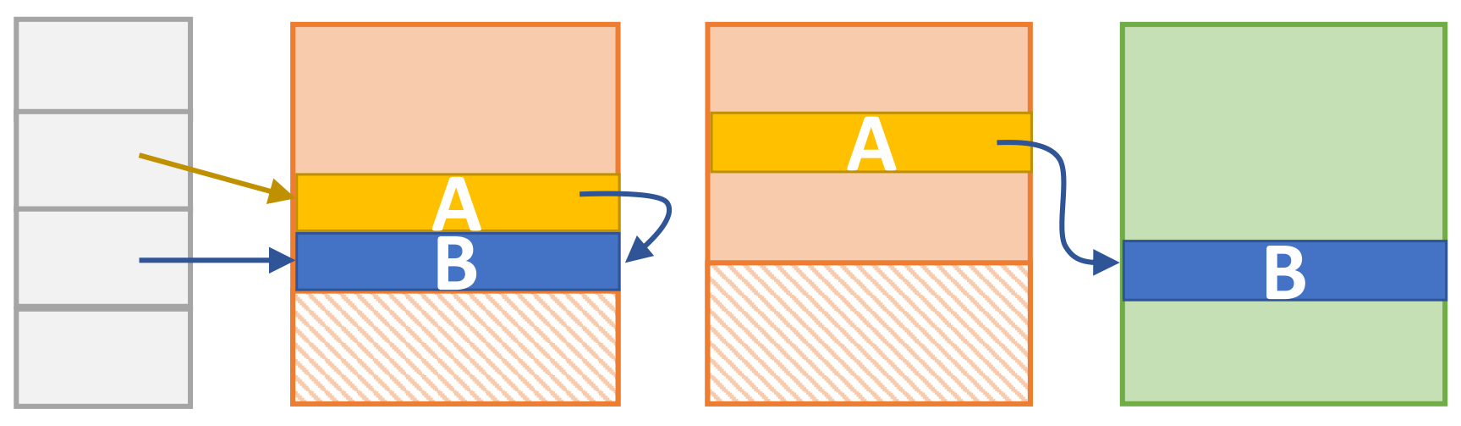
Insight 2: There is significant redundancy across objects of the same type

Baseline System: Hotpads, An Object-Based Memory Hierarchy [MICRO'18]

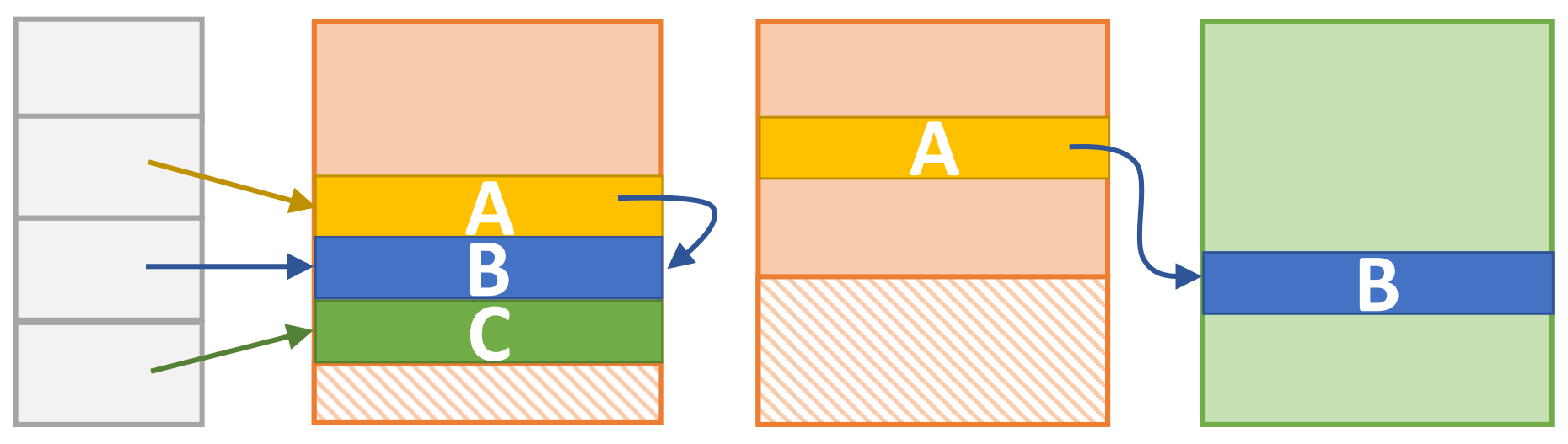
Example Hotpads hierarchy



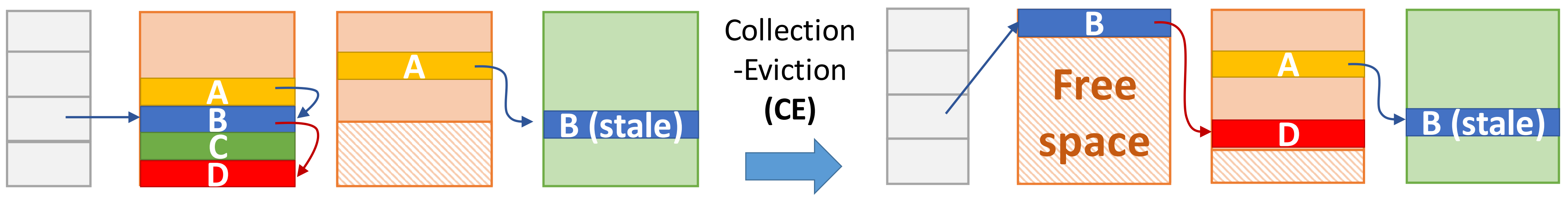
Feature 1.
Object-based data movement



Feature 2.
In-hierarchy object allocation

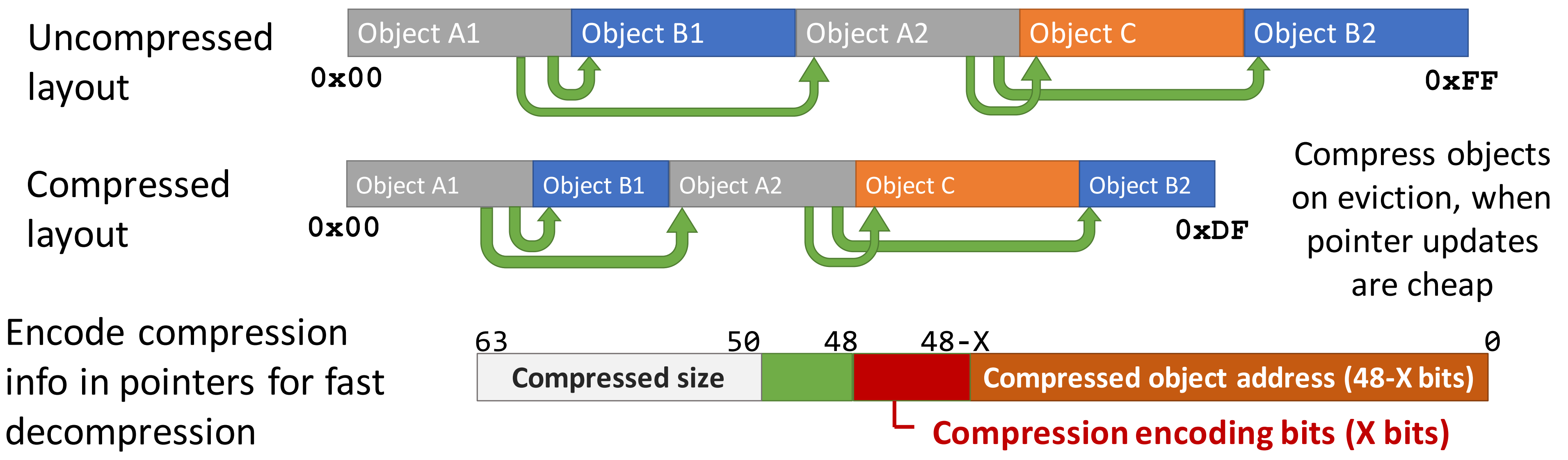


Feature 3.
Bulk **GC** and object **eviction** process that updates pointers to moved objects



Zippads: An Object-Based Compressed Memory Hierarchy

Point directly to compressed objects to avoid translation

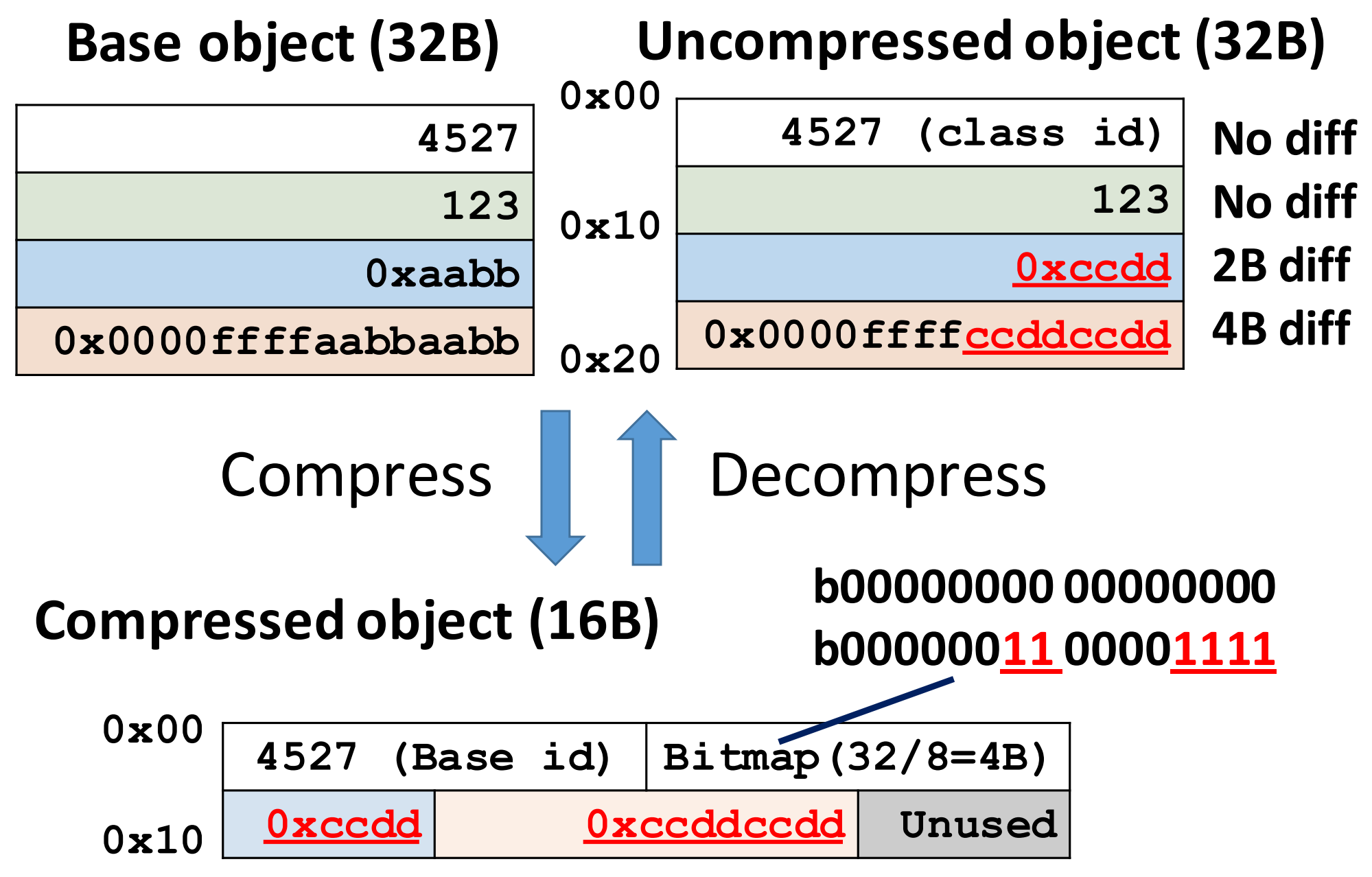


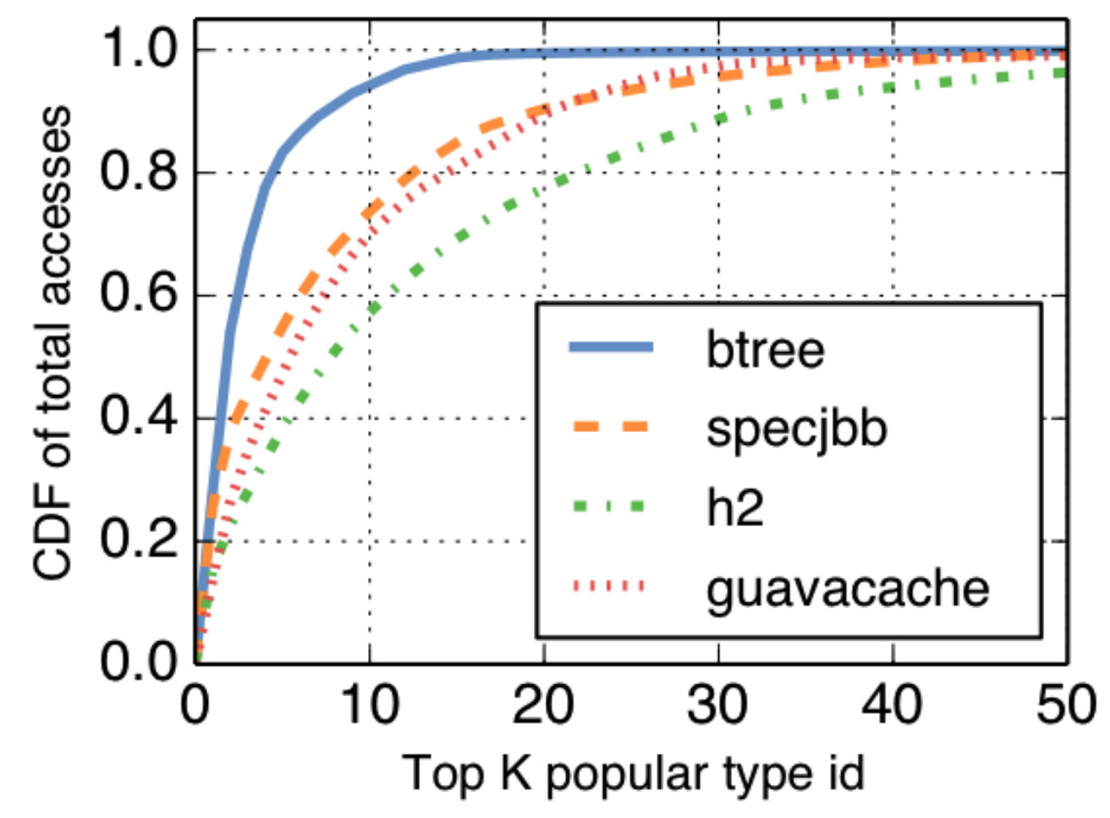
Compress objects on eviction, when pointer updates are cheap

Encode compression info in pointers for fast decompression

COCO: Cross-Object-Compression

Exploit redundancy across objects by storing only the bytes that differ from a representative object





The popularity of object types is skewed
→ Store representative objects with a cache

Evaluation

Methodology:

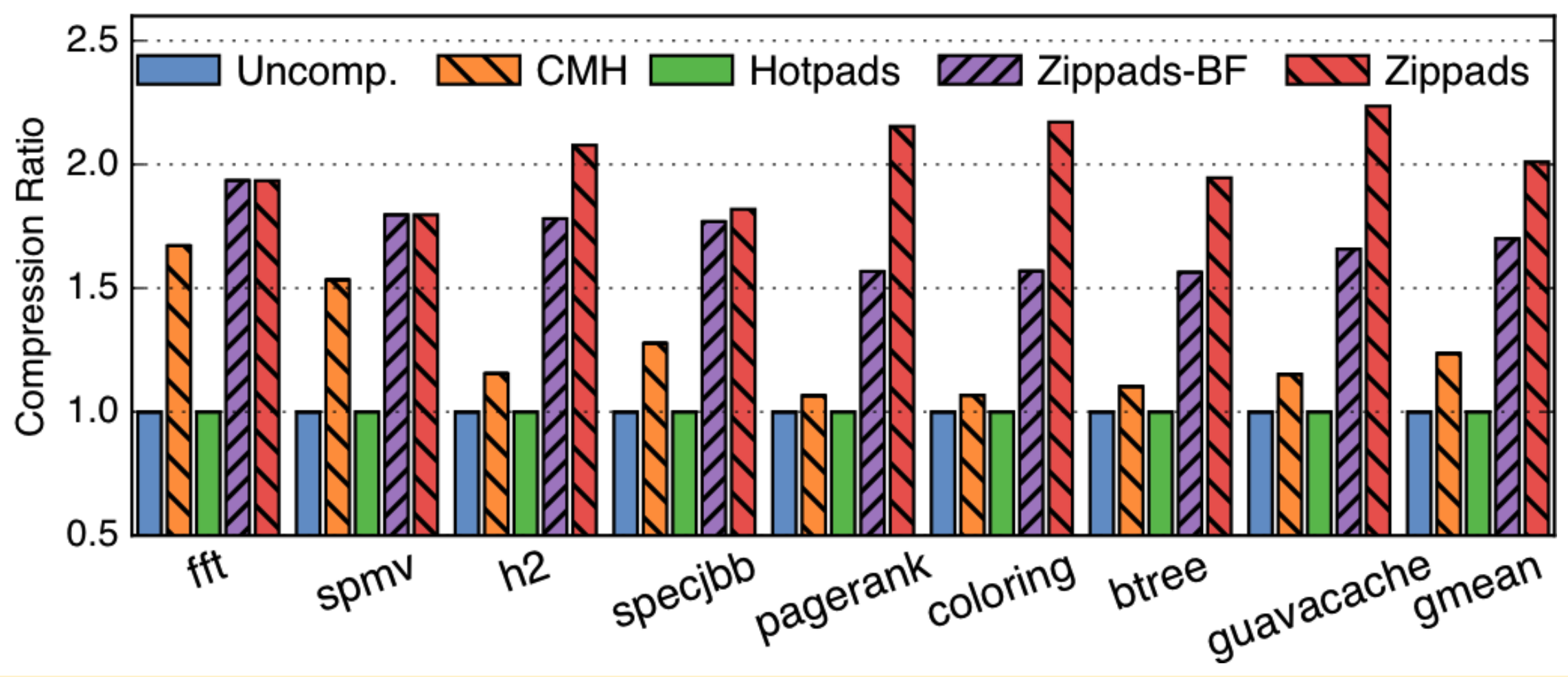
- Simulate Zippads using Maxsim (Zsim+Maxine JVM)
- **8 Java apps** from scientific, DB, graph analytics, KV store
- See our paper for **C/C++ apps** results

Compared schemes:

1. **Uncomp:** 3-level cache hierarchy without compression
2. **CMH:** State-of-the-art compressed memory hierarchy
3. **Hotpads**
4. **Zippads:** With and without **COCO**

Zippads significantly reduces memory footprint

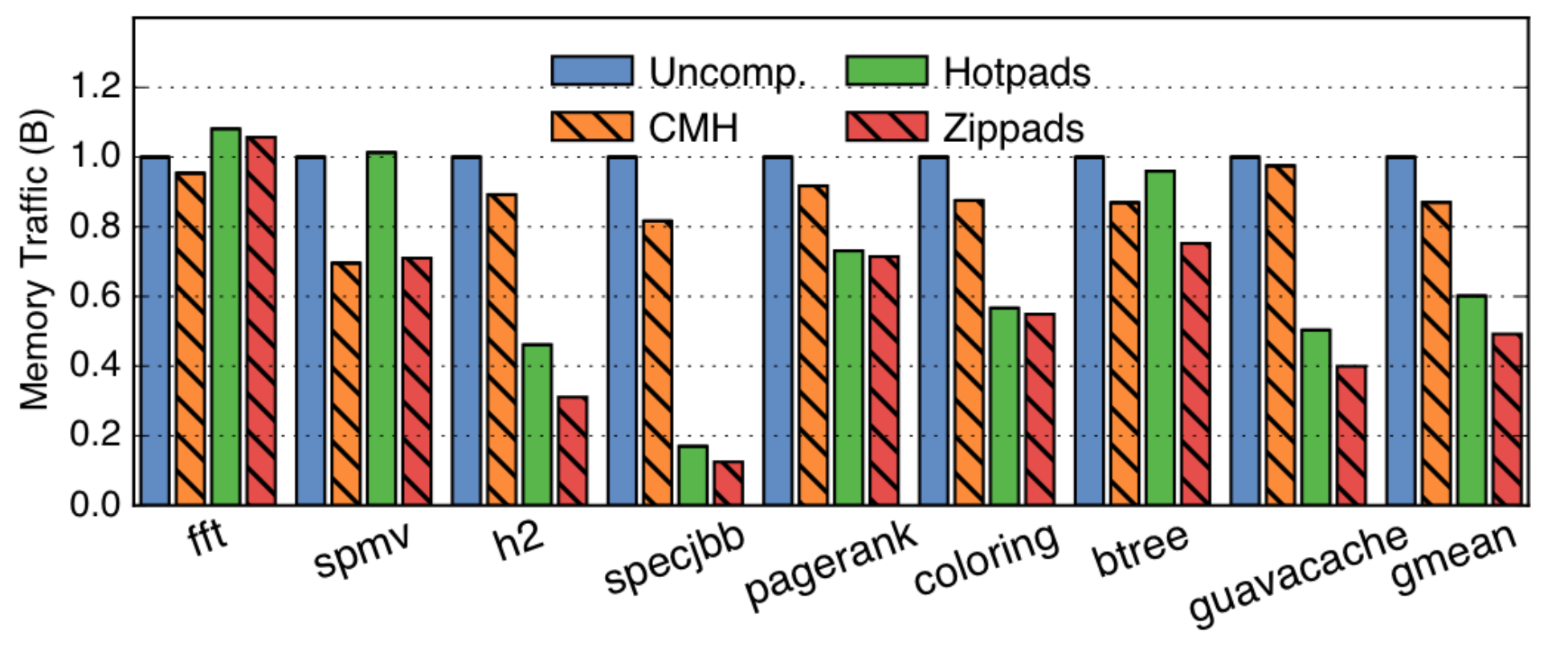
- **CMH** and **Zippads** compress well for array-heavy apps
- **Zippads** compresses much better for object-heavy apps



- **COCO** adds extra benefits
- **Zippads+COCO** improves over CMH by 63%

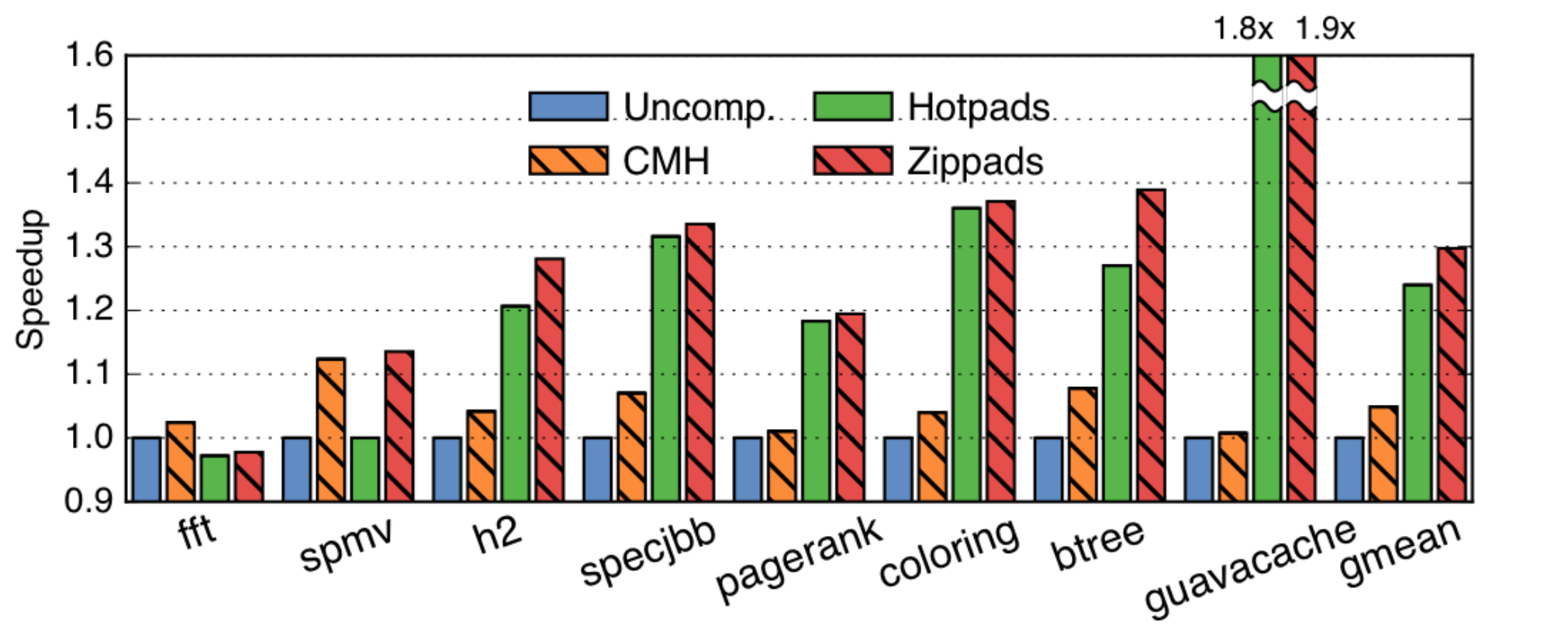
Zippads reduces memory traffic

- Achieves the lowest memory traffic (40% lower than CMH)
- Combines benefits of CMH and Hotpads



Zippads improves performance

- Outperforms CMH by 24% while reducing footprint much further



See our paper (https://bit.ly/zippads) for more features, details, and evaluation results!