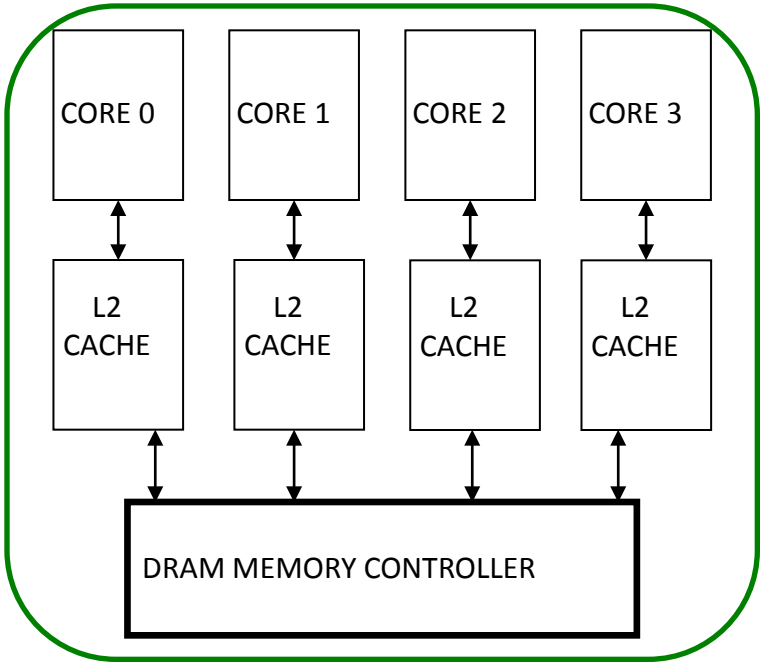
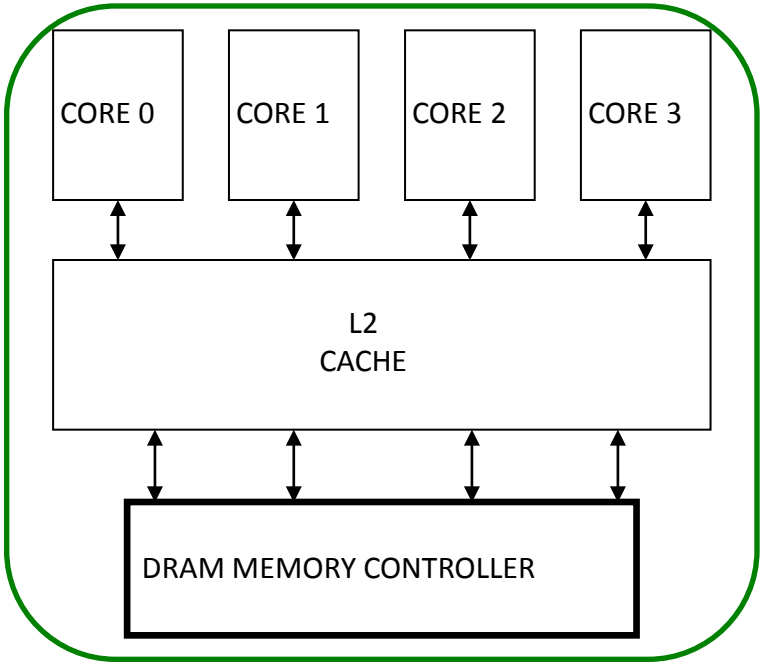
18-742
Parallel Computer Architecture
Lecture 11: Caching in Multi-Core Systems

Prof. Onur Mutlu and Gennady Pekhimenko
Carnegie Mellon University
Fall 2012, 10/01/2012

Review: Multi-core Issues in Caching

- How does the cache hierarchy change in a multi-core system?
- Private cache: Cache belongs to one core
- Shared cache: Cache is shared by multiple cores





Outline

- Multi-cores and Caching: Review
- Utility-based partitioning
- Cache compression
  - Frequent value
  - Frequent pattern
  - Base-Delta-Immediate
- Main memory compression
  - IBM MXT
  - Linearly Compressed Pages (LCP)

Review: Shared Caches Between Cores

- Advantages:
  - Dynamic partitioning of available cache space
    - No fragmentation due to static partitioning
  - Easier to maintain coherence
  - Shared data and locks do not ping pong between caches

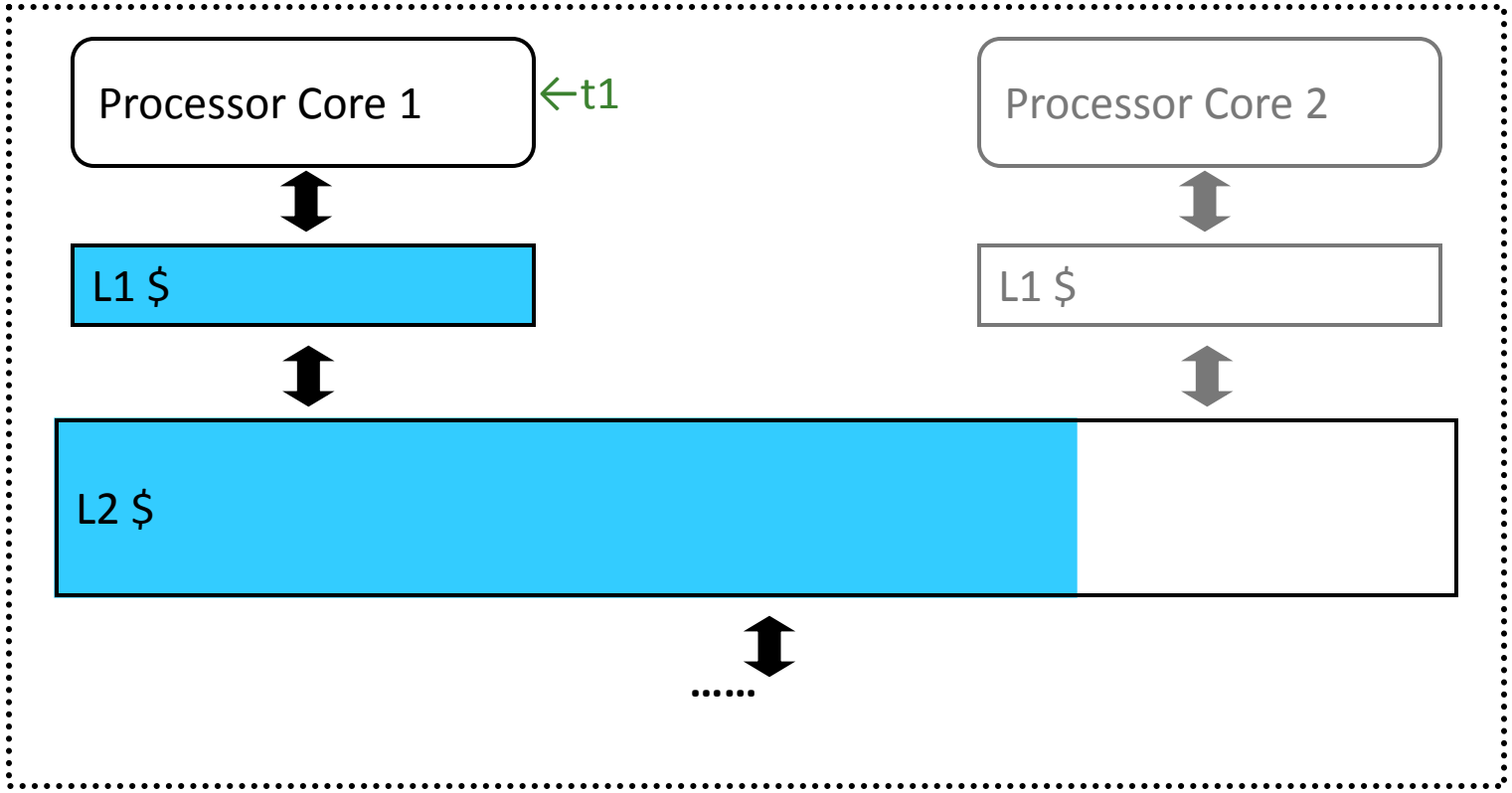
- Disadvantages
  - Cores incur conflict misses due to other cores' accesses
    - Misses due to inter-core interference
    - Some cores can destroy the hit rate of other cores
      - What kind of access patterns could cause this?
  - Guaranteeing a minimum level of service (or fairness) to each core is harder (how much space, how much bandwidth?)
  - High bandwidth harder to obtain (N cores → N ports?)

Shared Caches: How to Share?

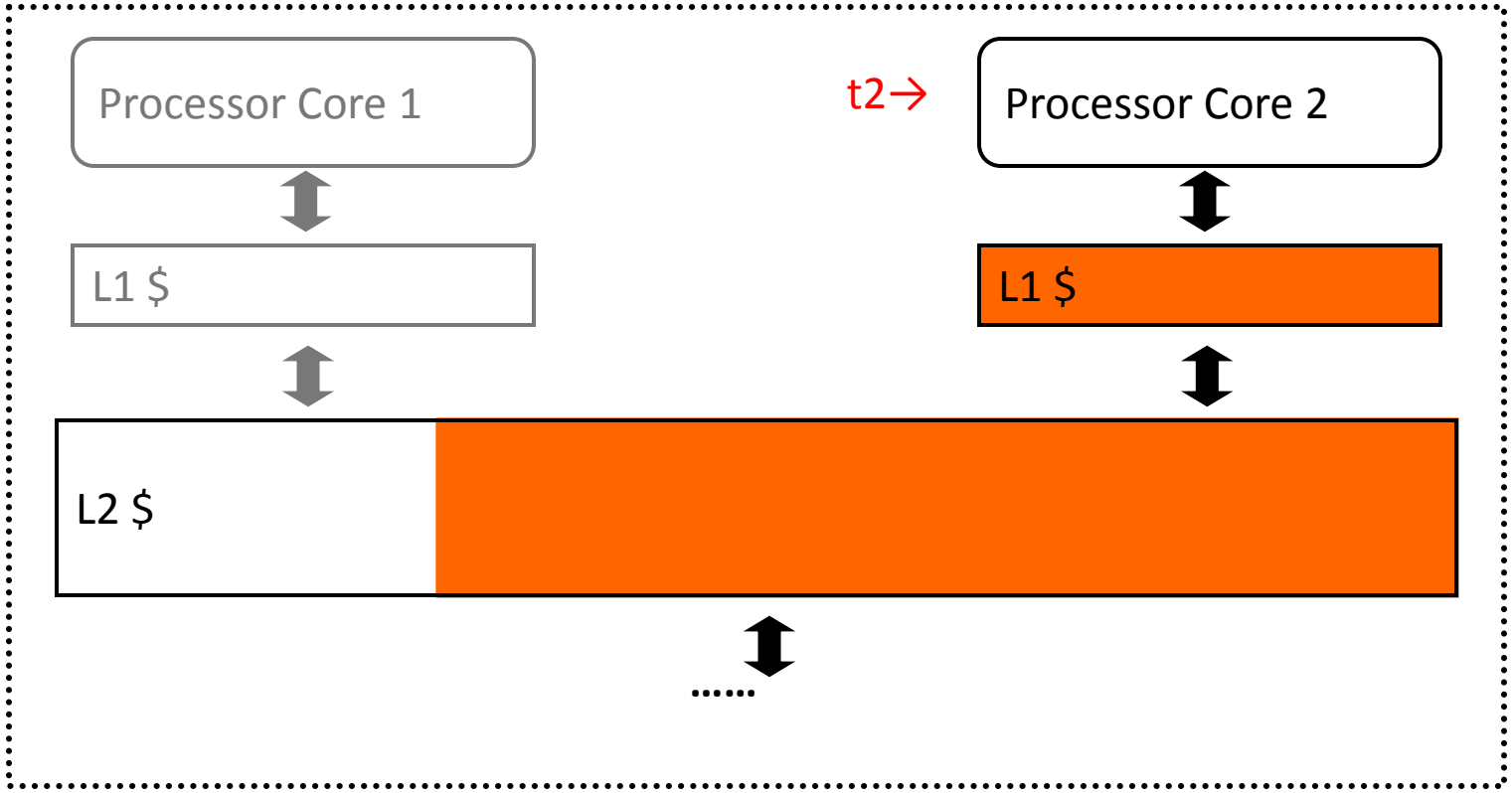
- Free-for-all sharing
  - Placement/replacement policies are the same as a single core system (usually LRU or pseudo-LRU)
  - Not thread/application aware
  - An incoming block evicts a block regardless of which threads the blocks belong to

- Problems
  - A cache-unfriendly application can destroy the performance of a cache friendly application
  - Not all applications benefit equally from the same amount of cache: free-for-all might prioritize those that do not benefit
  - Reduced performance, reduced fairness

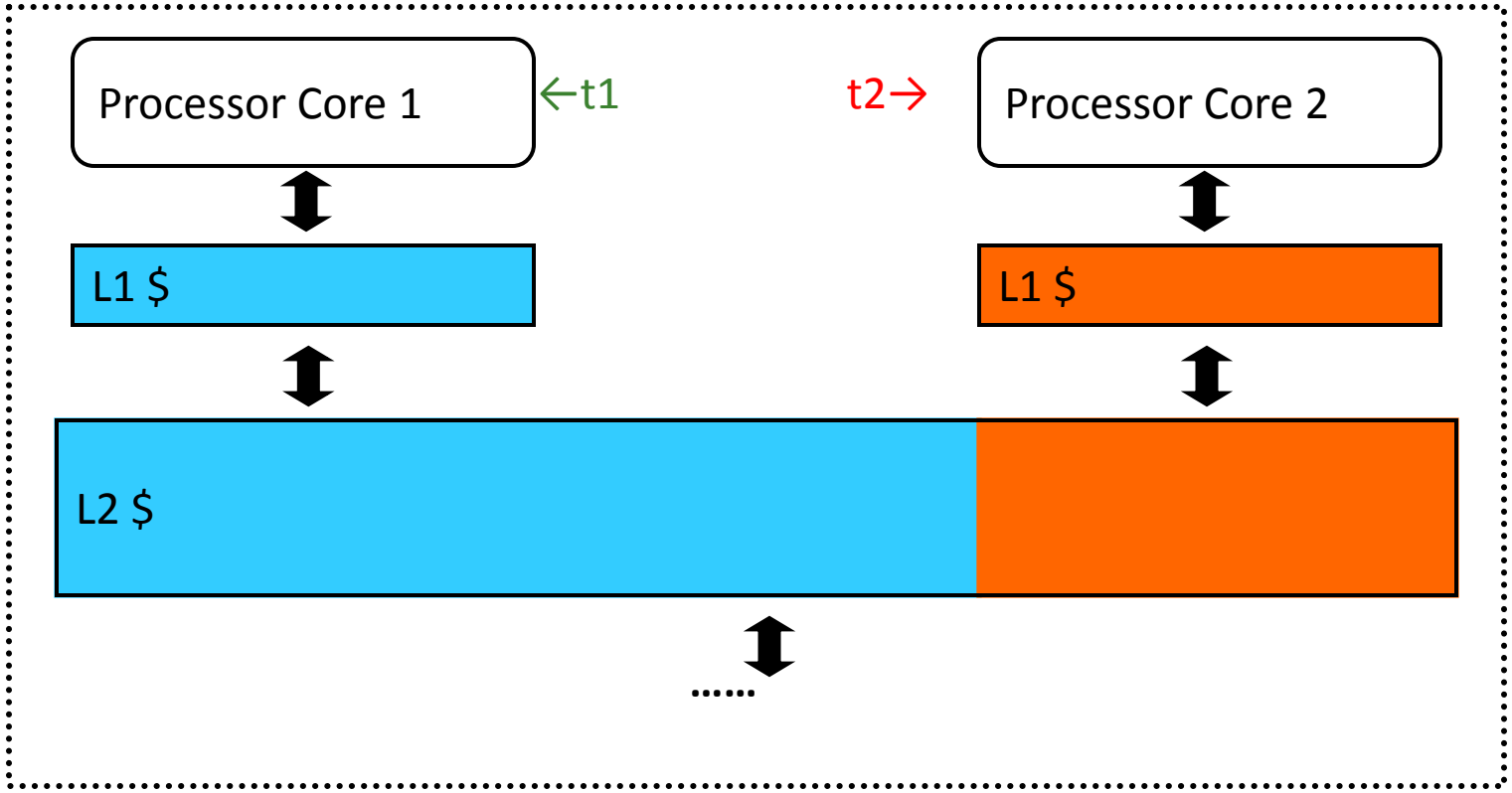
Problem with Shared Caches



Problem with Shared Caches



Problem with Shared Caches



t2's throughput is significantly reduced due to unfair cache sharing.

Controlled Cache Sharing

- Utility based cache partitioning
  - Qureshi and Patt, "Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches," MICRO 2006.
  - Suh et al., "A New Memory Monitoring Scheme for Memory-Aware Scheduling and Partitioning," HPCA 2002.

- Fair cache partitioning
  - Kim et al., "Fair Cache Sharing and Partitioning in a Chip Multiprocessor Architecture," PACT 2004.

- Shared/private mixed cache mechanisms
  - Qureshi, "Adaptive Spill-Receive for Robust High-Performance Caching in CMPs," HPCA 2009.

Utility Based Shared Cache Partitioning

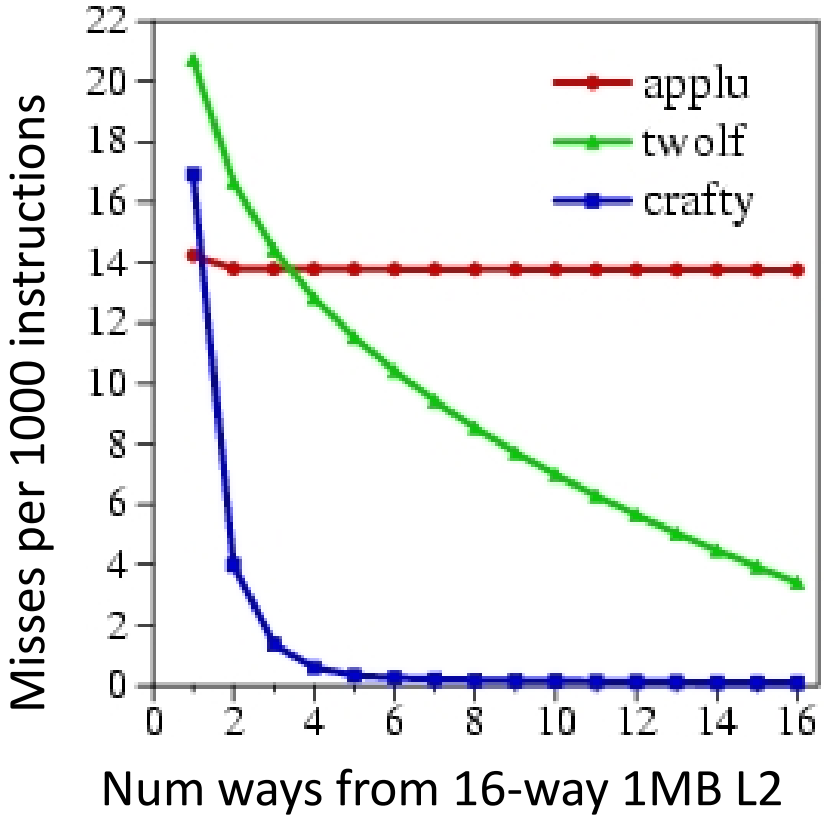
- Goal: Maximize system throughput
- Observation: Not all threads/applications benefit equally from caching → simple LRU replacement not good for system throughput
- Idea: Allocate more cache space to applications that obtain the most benefit from more space

- The high-level idea can be applied to other shared resources as well.

- Qureshi and Patt, "Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches," MICRO 2006.
- Suh et al., "A New Memory Monitoring Scheme for Memory-Aware Scheduling and Partitioning," HPCA 2002.

Utility Based Cache Partitioning (I)

Utility U_a^b = Misses with a ways – Misses with b ways

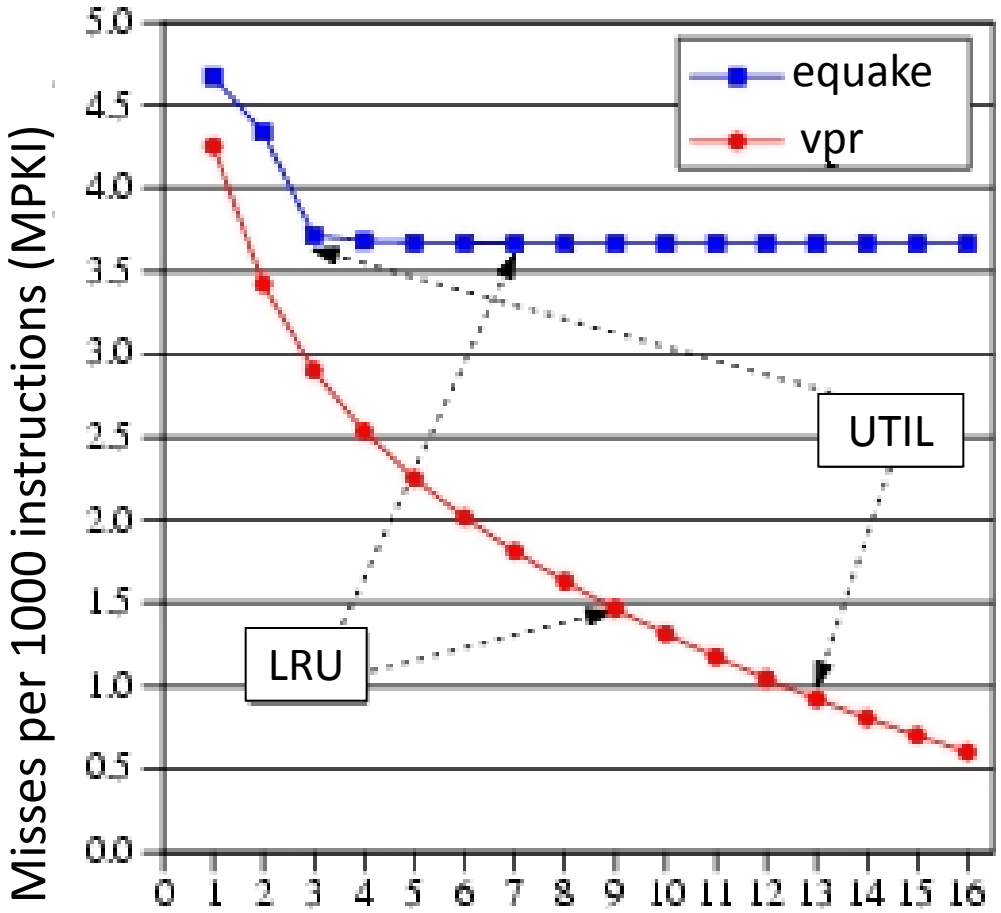


Low Utility

High Utility

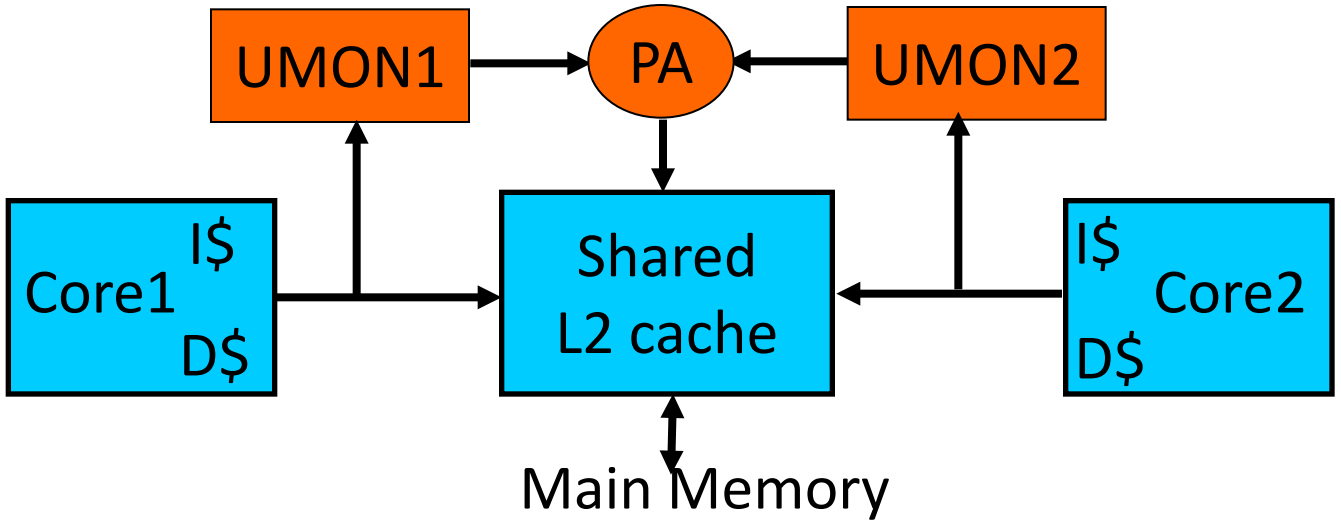
Saturating Utility

Utility Based Cache Partitioning (II)



Idea: Give more cache to the application that benefits more from cache

Utility Based Cache Partitioning (III)



Three components:

- ❑ Utility Monitors (UMON) per core
- ❑ Partitioning Algorithm (PA)
- ❑ Replacement support to enforce partitions

Cache Capacity

- How to get more cache without making it physically larger?

- **Idea**: Data compression for on chip-caches

Base-Delta-Immediate Compression: Practical Data Compression for On-Chip Caches

Gennady Pekhimenko
Vivek Seshadri
Onur Mutlu , Todd C. Mowry

Phillip B. Gibbons*
Michael A. Kozuch*

Carnegie Mellon

intel *

Executive Summary

- Off-chip memory latency is high
  - Large caches can help, **but** at significant cost
- Compressing data in cache enables larger cache at low cost
- **Problem**: Decompression is on the execution critical path
- **Goal**: Design a new compression scheme that has

1. low decompression latency, 2. low cost, 3. high compression ratio

- **Observation:** Many cache lines have low dynamic range data
- **Key Idea**: Encode cachelines as a base **+** multiple differences
- **Solution**: Base-Delta-Immediate compression with low decompression latency and high compression ratio
  - Outperforms three state-of-the-art compression mechanisms

Motivation for Cache Compression

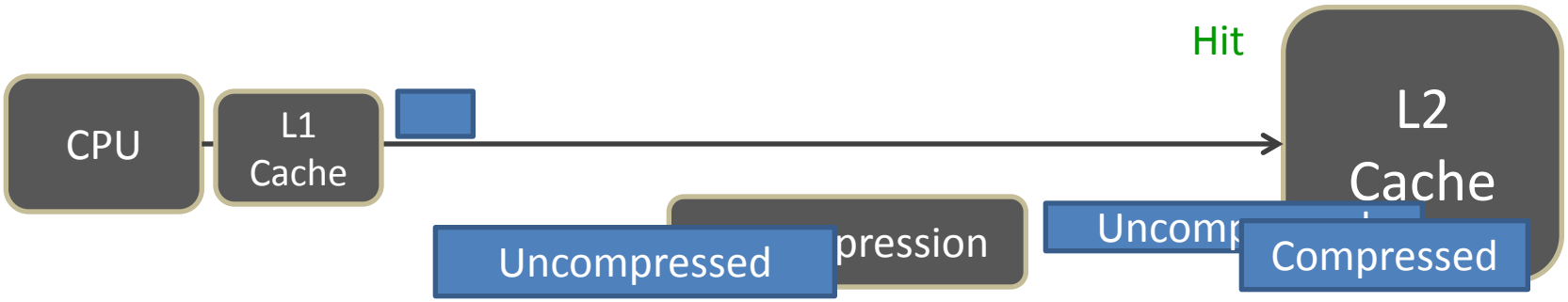
Significant redundancy in data:

0x**00000000**	0x**0000000**B	0x**00000000**3	0x**00000000**4	…

How can we exploit this redundancy?

- **Cache compression** helps
- Provides effect of a larger cache without making it physically larger

Background on Cache Compression



- Key requirements:
  - **Fast** (low decompression latency)
  - **Simple** (avoid complex hardware changes)
  - **Effective** (good compression ratio)

Zero Value Compression

- Advantages:
  - Low decompression latency
  - Low complexity

- Disadvantages:
  - Low average compression ratio

Shortcomings of Prior Work

Compression Mechanisms	Decompression Latency	Complexity	Compression Ratio
Zero	✓	✓	✗

Frequent Value Compression

- **Idea**: encode cache lines based on frequently occurring values
- Advantages:
  - Good compression ratio
- Disadvantages:
  - Needs profiling
  - High decompression latency
  - High complexity

Shortcomings of Prior Work

Compression Mechanisms	Decompression Latency	Complexity	Compression Ratio
Zero	✓	✓	✗
Frequent Value	✗	✗	✓

Frequent Pattern Compression

- **Idea**: encode cache lines based on frequently occurring patterns, e.g., half word is zero
- Advantages:
  - Good compression ratio
- Disadvantages:
  - High decompression latency (5-8 cycles)
  - High complexity (for some designs)

Shortcomings of Prior Work

Compression Mechanisms	Decompression Latency	Complexity	Compression Ratio
Zero	✓	✓	✗
Frequent Value	✗	✗	✓
Frequent Pattern	✗	✗/✓	✓

Shortcomings of Prior Work

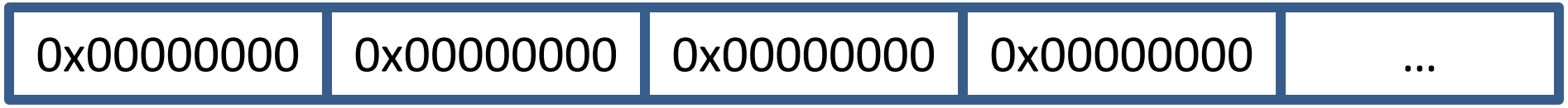
Compression Mechanisms	Decompression Latency	Complexity	Compression Ratio
Zero	✓	✓	✗
Frequent Value	✗	✗	✓
Frequent Pattern	✗	✗ / ✓	✓
Our proposal: BΔI	✓	✓	✓

Outline

- Motivation & Background
- Key Idea & Our Mechanism
- Evaluation
- Conclusion

Key Data Patterns in Real Applications

Zero Values: initialization, sparse matrices, NULL pointers



Repeated Values: common initial values, adjacent pixels

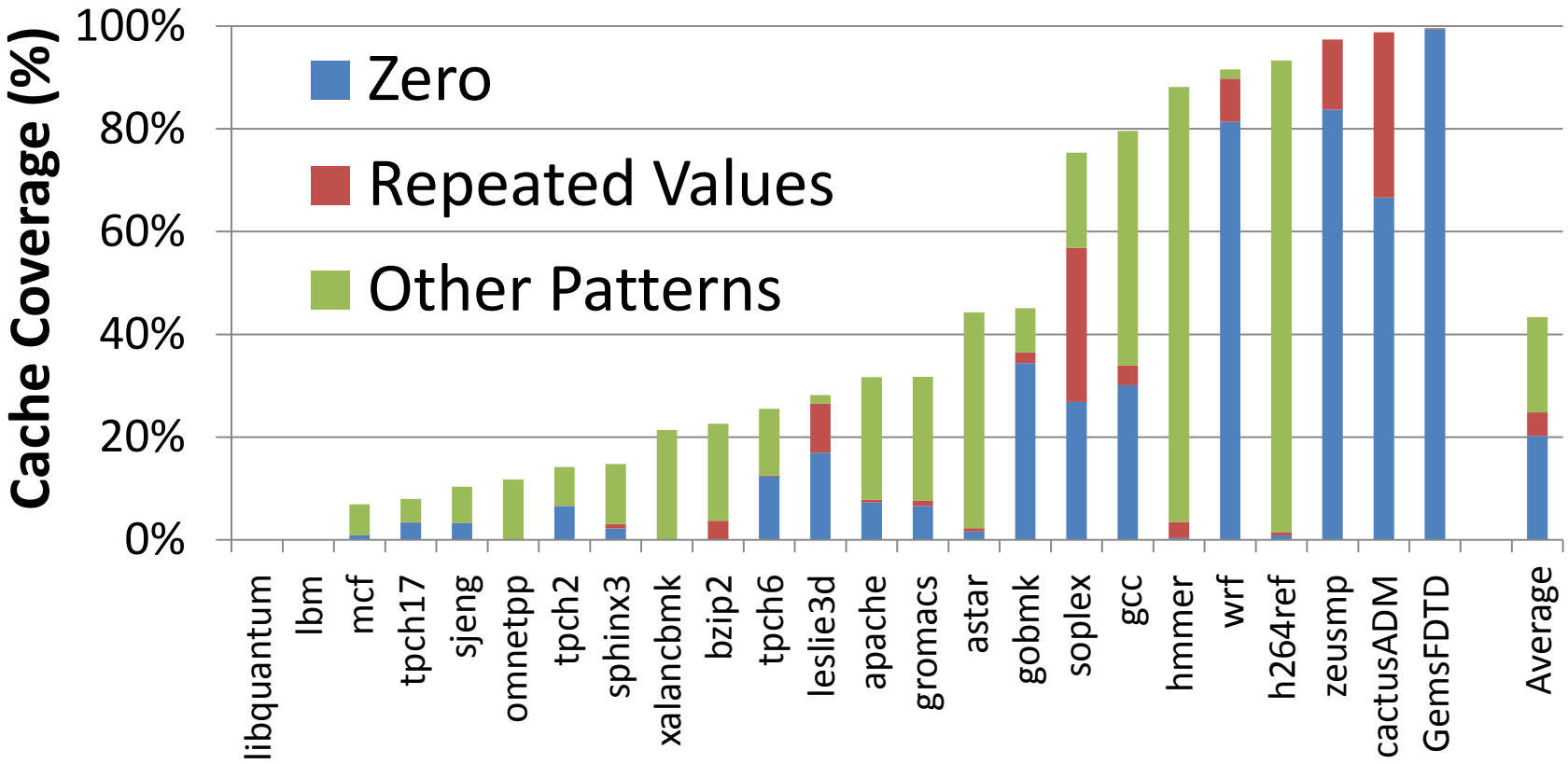
Narrow Values: small values stored in a big data type

Other Patterns: pointers to the same memory region

How Common Are These Patterns?

SPEC2006, databases, web workloads, 2MB L2 cache

"Other Patterns" include Narrow Values



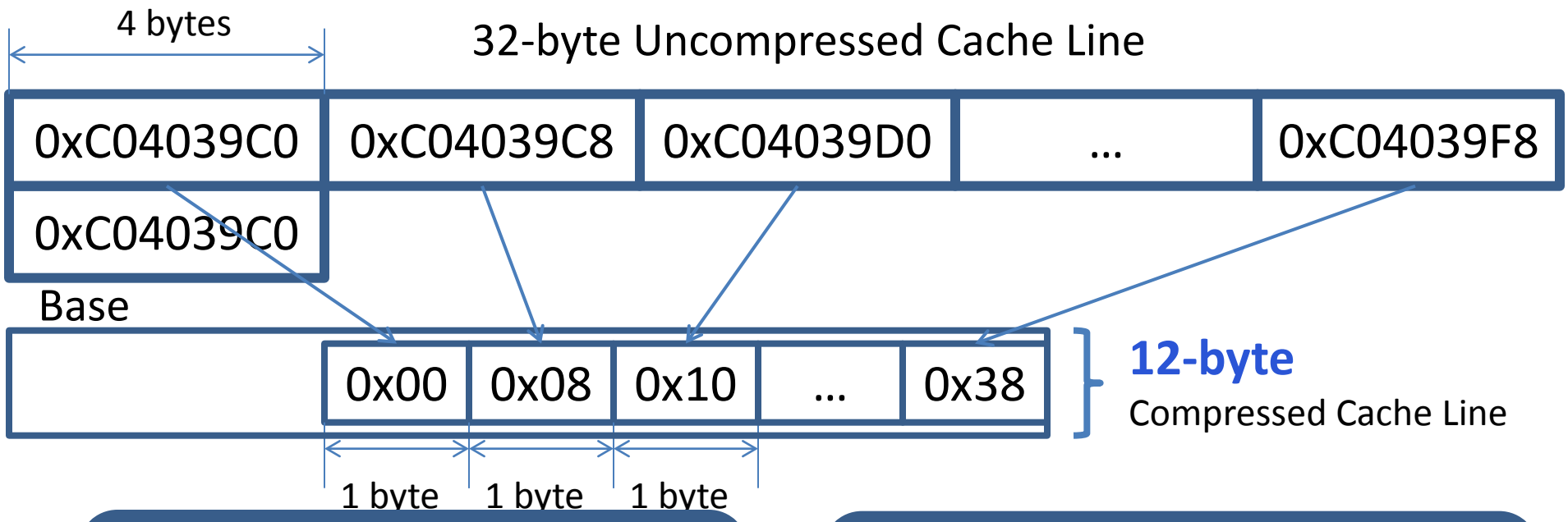
43% of the cache lines belong to key patterns

Key Data Patterns in Real Applications

Low Dynamic Range:

Differences between values are significantly smaller than the values themselves

Key Idea: Base+Delta (B+Δ) Encoding

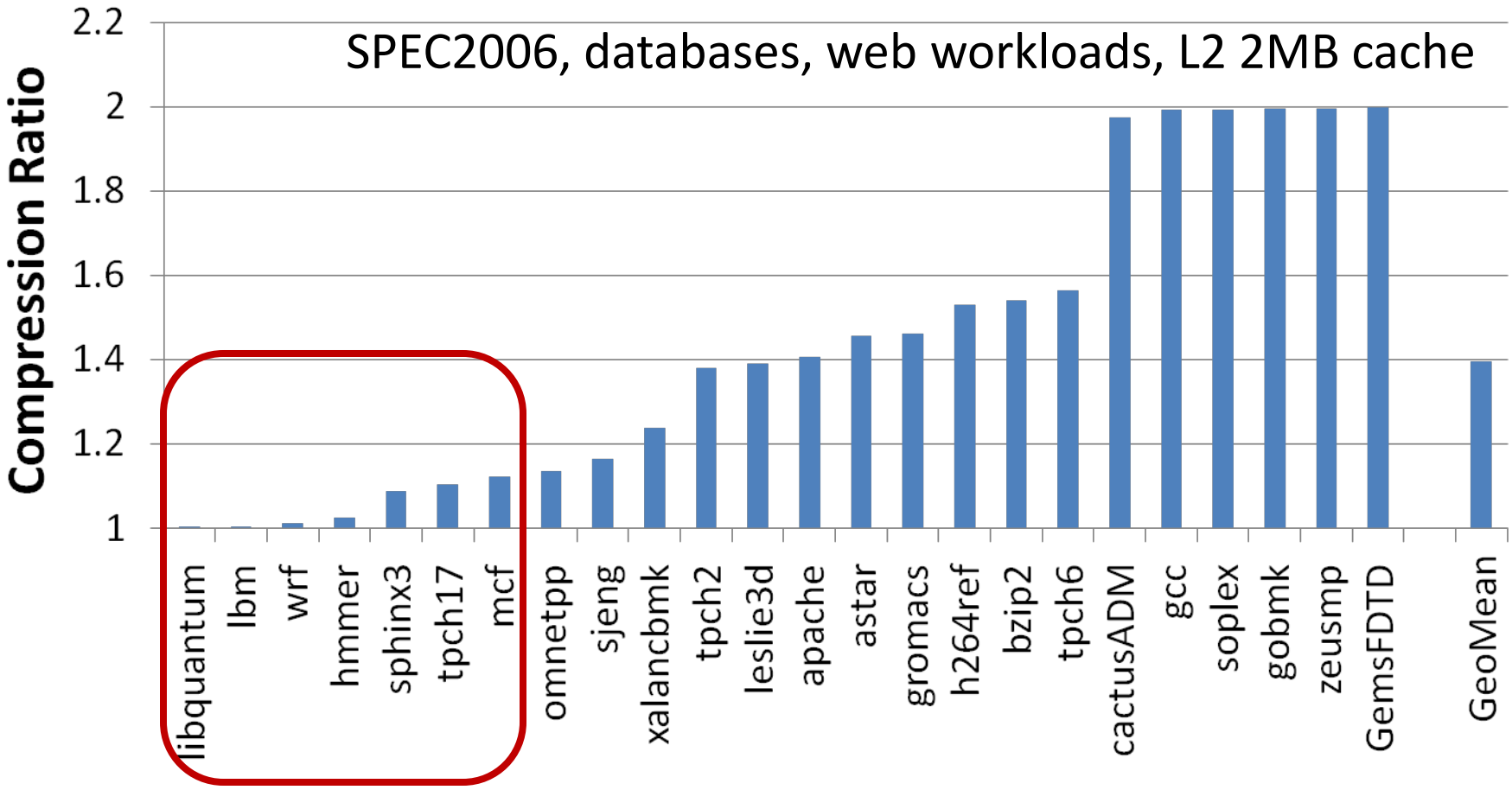


✓ **Fast Decompression:** vector addition

✓ **Simple Hardware:** arithmetic and comparison

✓ **Effective:** good compression ratio

B+Δ: Compression Ratio



Good average compression ratio (**1.40**)

But some benchmarks have low compression ratio

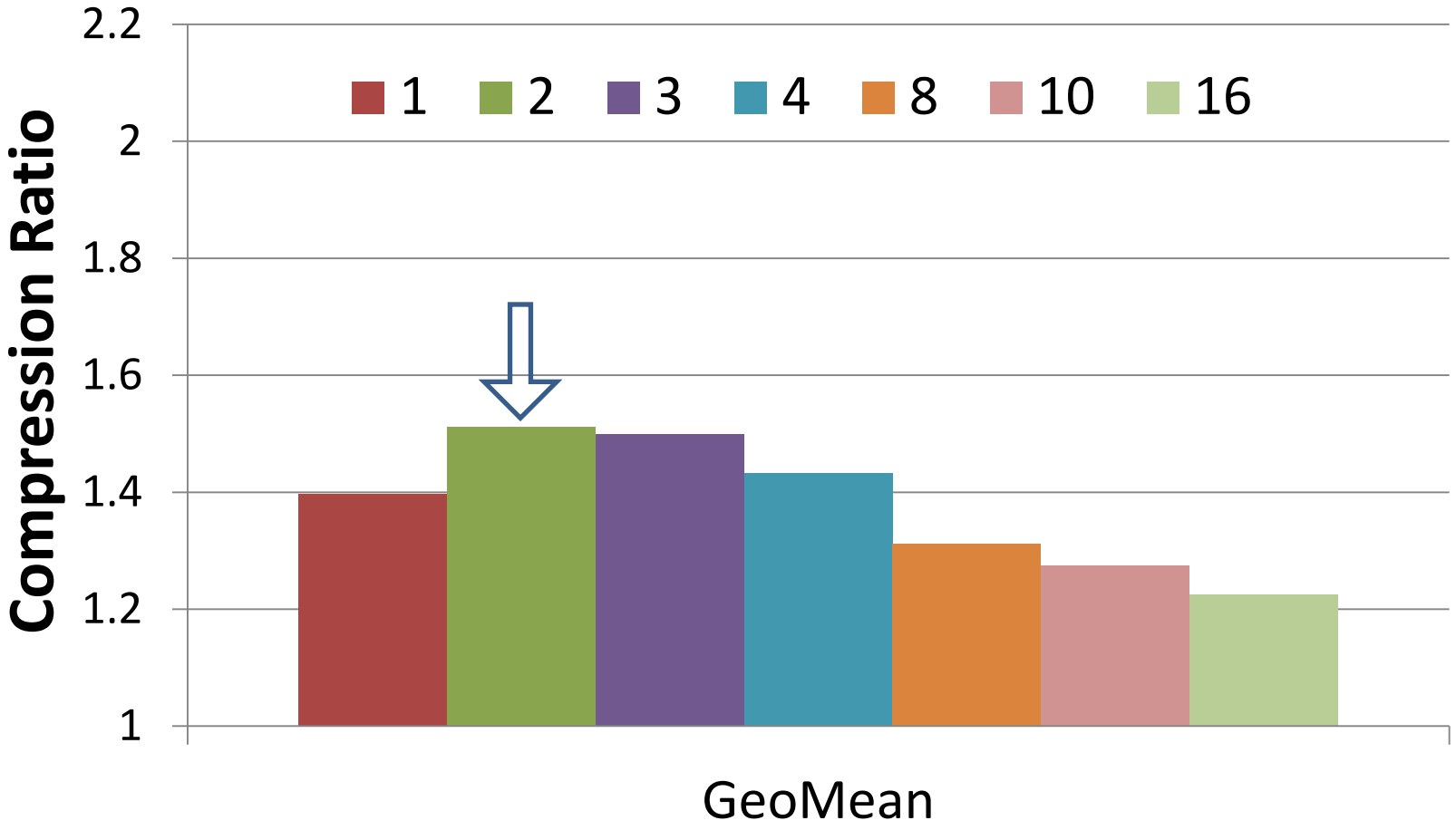
Can We Do Better?

- Uncompressible cache line (with a single base):

0x00000000	**0x09A4**0178	**0x0000000**B	**0x09A4**A838	…

- **Key idea:**
  Use more bases, e.g., two instead of one
- Pro:
  - More cache lines can be compressed
- Cons:
  - Unclear how to find these bases efficiently
  - Higher overhead (due to additional bases)

B+Δ with Multiple Arbitrary Bases



✓ **2 bases –** the best option based on evaluations

How to Find Two Bases Efficiently?

1. **First base - first element** in the cache line

   ✓ **Base+Delta part**

2. **Second base -** implicit base of **0**

   ✓ **Immediate part**

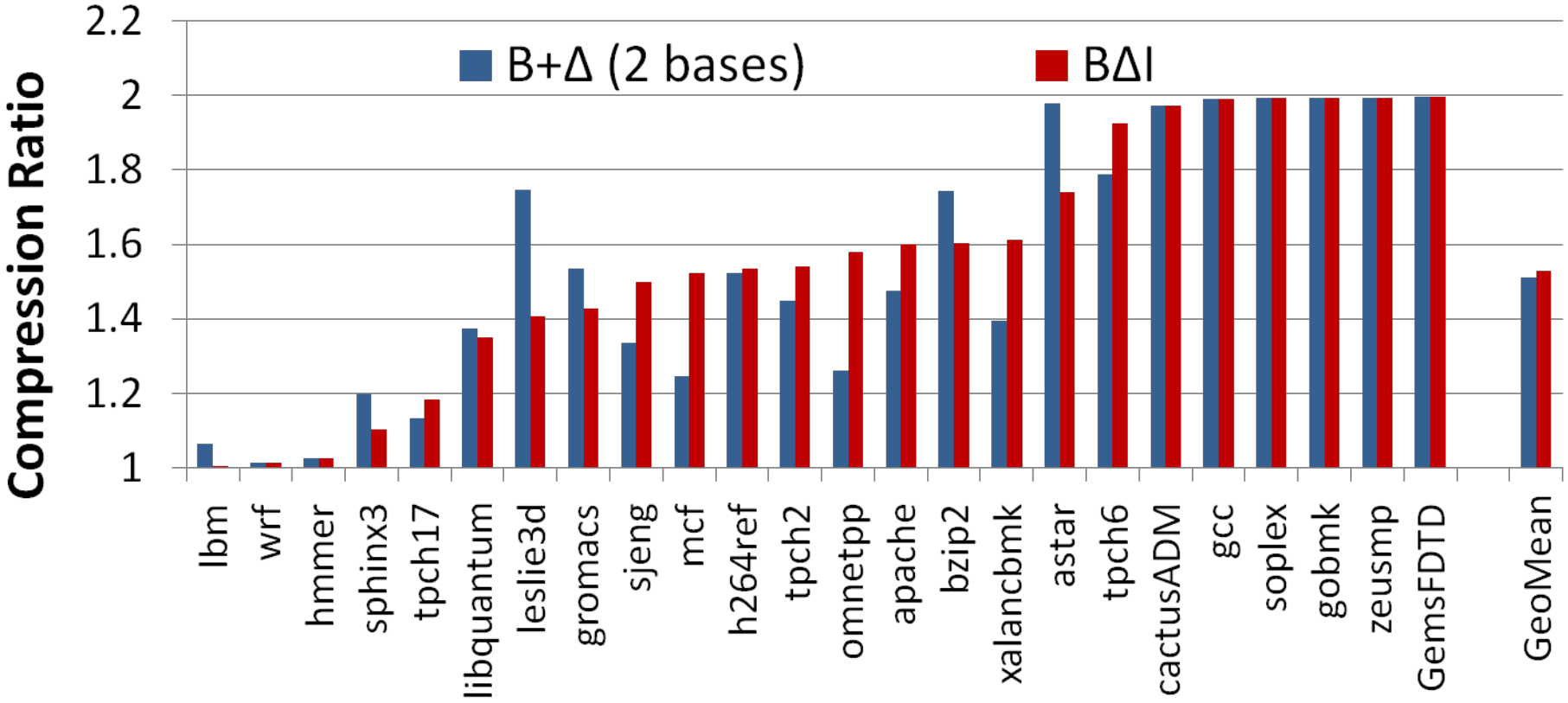
Advantages over 2 arbitrary bases:

– Better compression ratio

– Simpler compression logic

Base-Delta-Immediate (BΔI) Compression

B+Δ (with two arbitrary bases) vs. BΔI

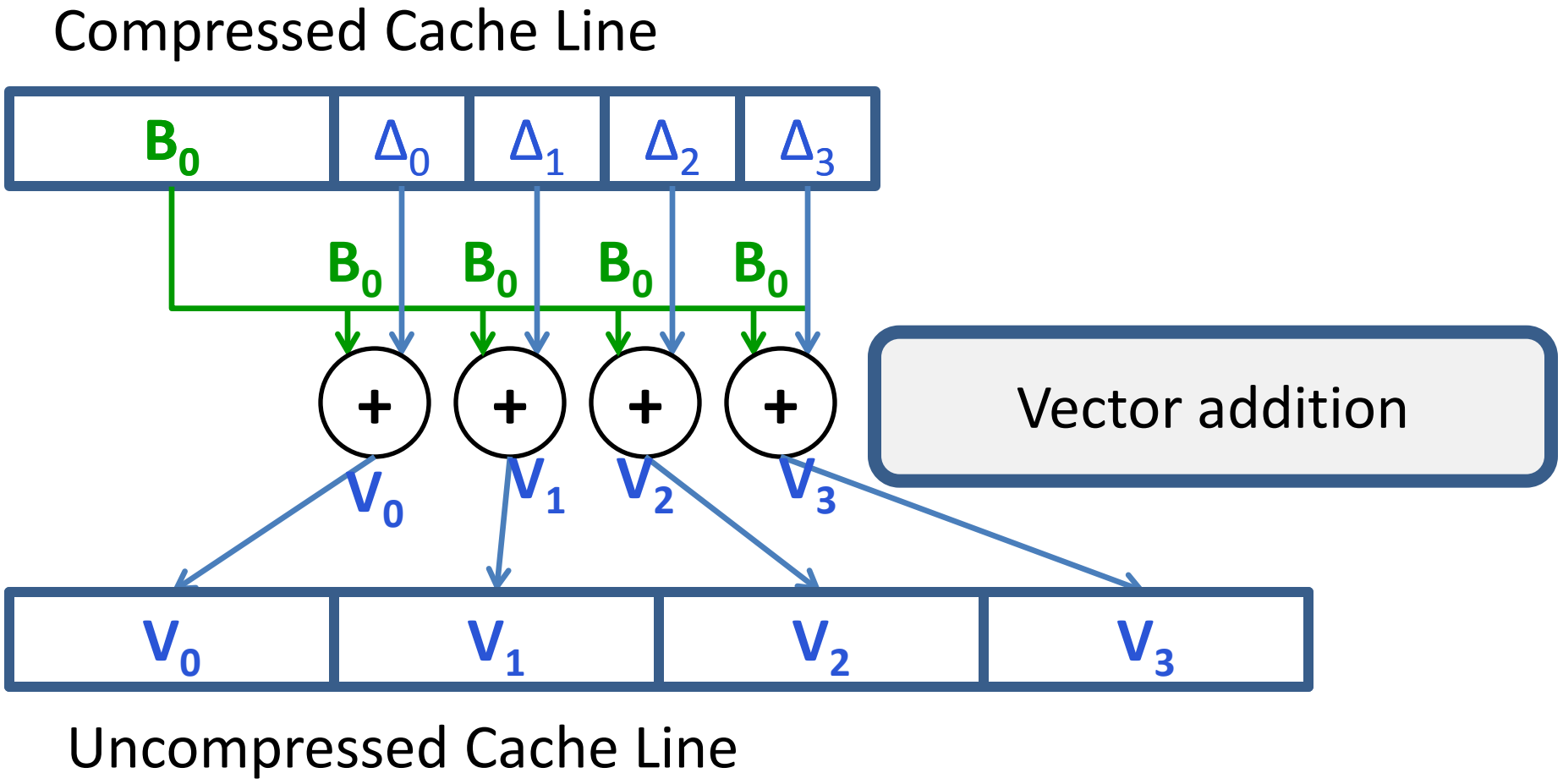


Average compression ratio is close, but **BΔI** is **simpler**

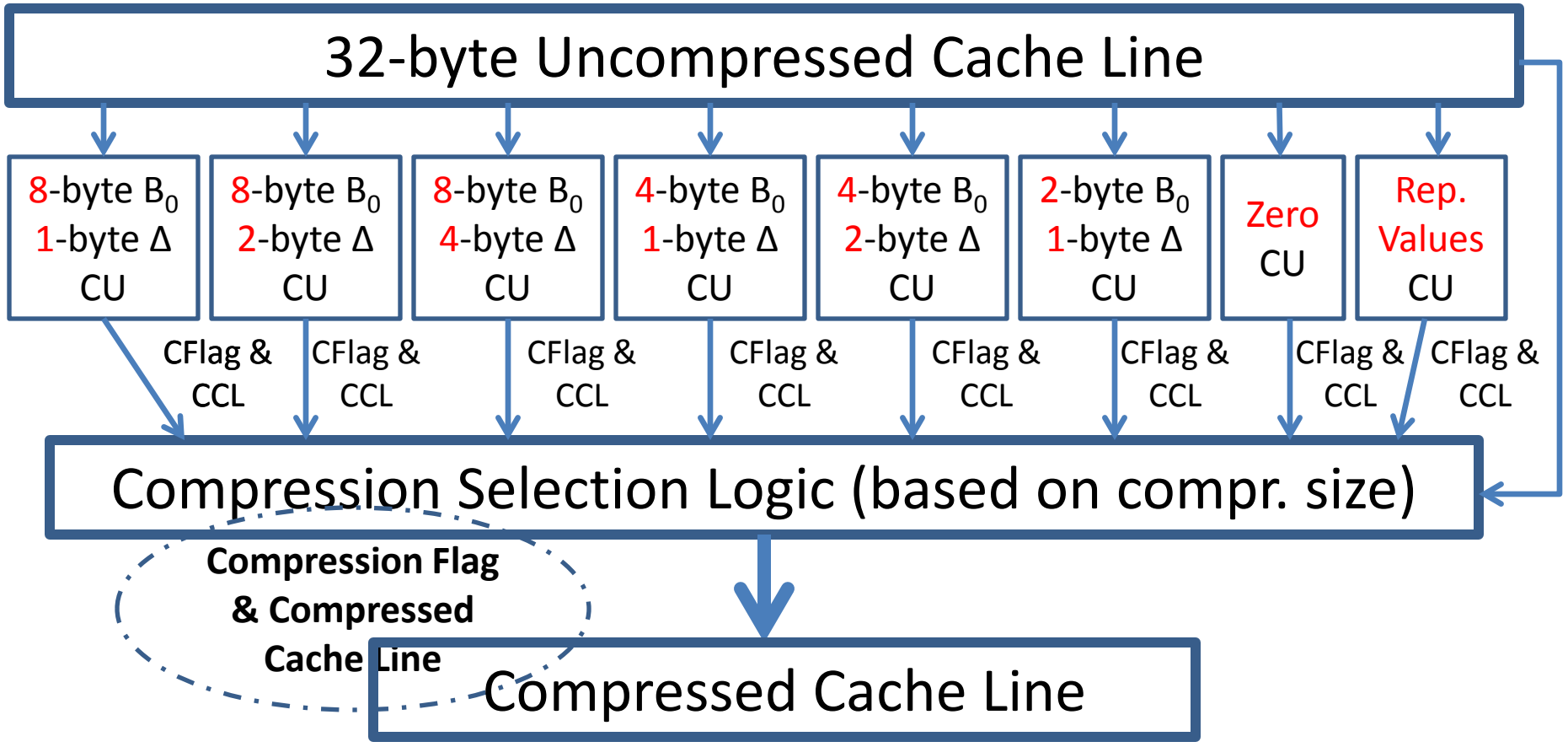
BΔI Implementation

- **Decompressor Design**
  - Low latency
- **Compressor Design**
  - Low cost and complexity
- **BΔI Cache Organization**
  - Modest complexity

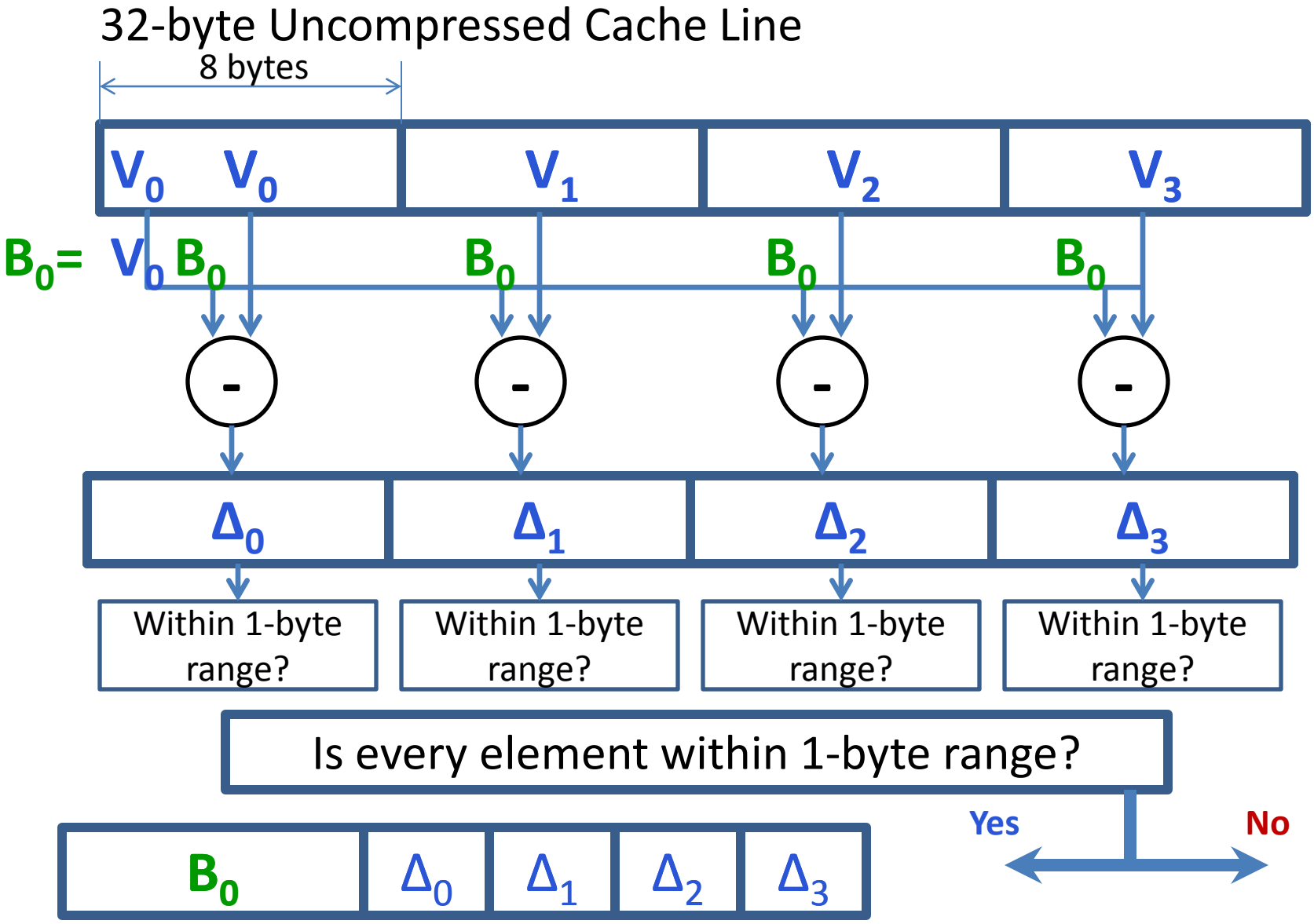
BΔI Decompressor Design



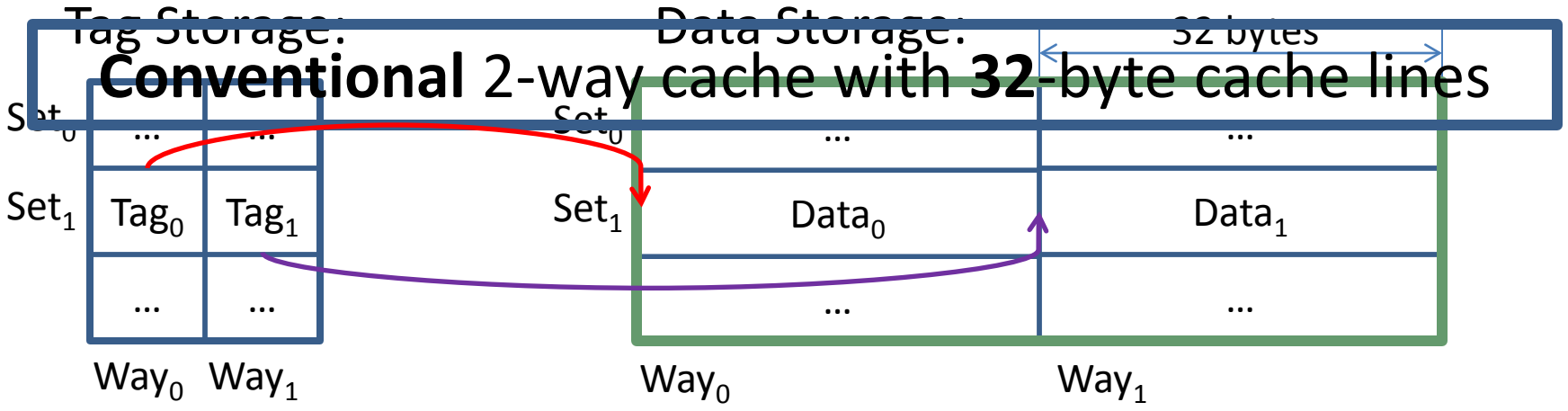
BΔI Compressor Design



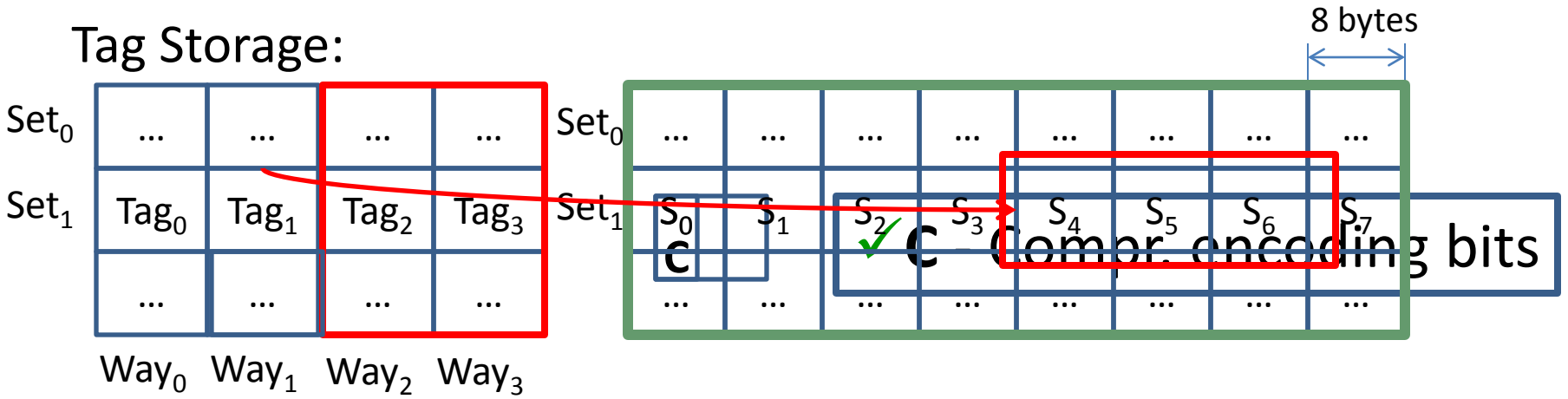
BΔI Compression Unit: 8-byte B_0 1-byte Δ



BΔI Cache Organization



BΔI: 4-way cache with **8**-byte segmented data



✓ Twice as many tags

✓ Tags map to multiple adjacent segments

2.3% overhead for 2 MB cache

Qualitative Comparison with Prior Work

- **Zero-based designs**
  - ZCA *[Dusser+, ICS'09]*: zero-content augmented cache
  - ZVC *[Islam+, PACT'09]*: zero-value cancelling
  - Limited applicability (only zero values)
- **FVC** *[Yang+, MICRO'00]*: frequent value compression
  - High decompression latency and complexity
- **Pattern-based compression designs**
  - FPC *[Alameldeen+, ISCA'04]*: frequent pattern compression
    - High decompression latency (5 cycles) and complexity
  - C-pack *[Chen+, T-VLSI Systems'10]*: practical implementation of FPC-like algorithm
    - High decompression latency (8 cycles)

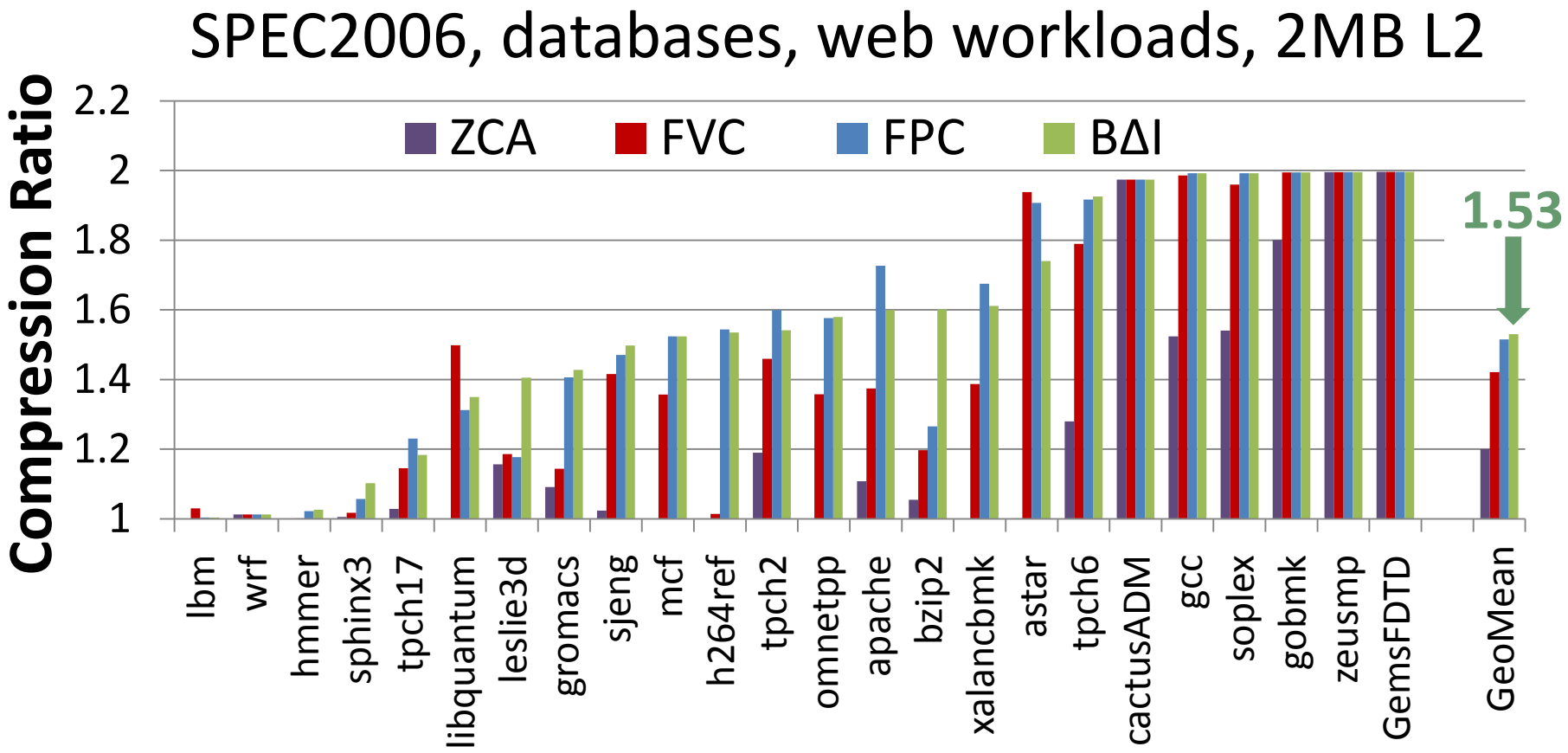
Outline

- Motivation & Background
- Key Idea & Our Mechanism
- Evaluation
- Conclusion

Methodology

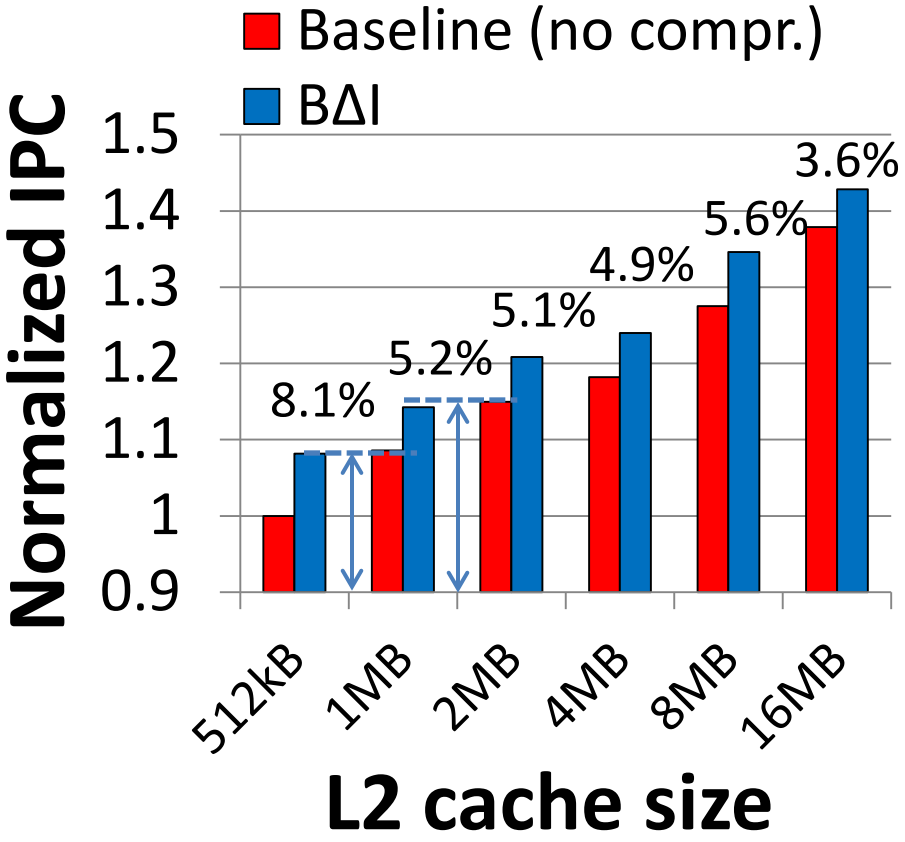
- **Simulator**
  - x86 event-driven simulator based on Simics *[Magnusson+, Computer'02]*
- **Workloads**
  - SPEC2006 benchmarks, TPC, Apache web server
  - 1 – 4 core simulations for 1 billion representative instructions
- **System Parameters**
  - L1/L2/L3 cache latencies from CACTI *[Thoziyoor+, ISCA'08]*
  - 4GHz, x86 in-order core, **512kB - 16MB** L2, simple memory model (**300**-cycle latency for row-misses)

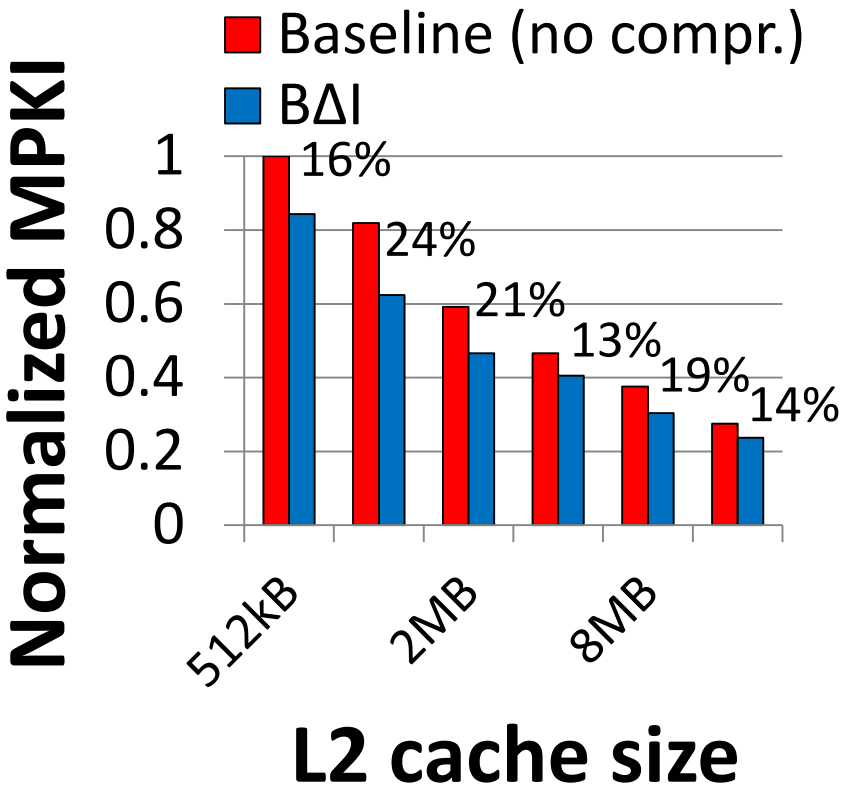
Compression Ratio: BΔI vs. Prior Work



BΔI achieves the highest compression ratio

Single-Core: IPC and MPKI

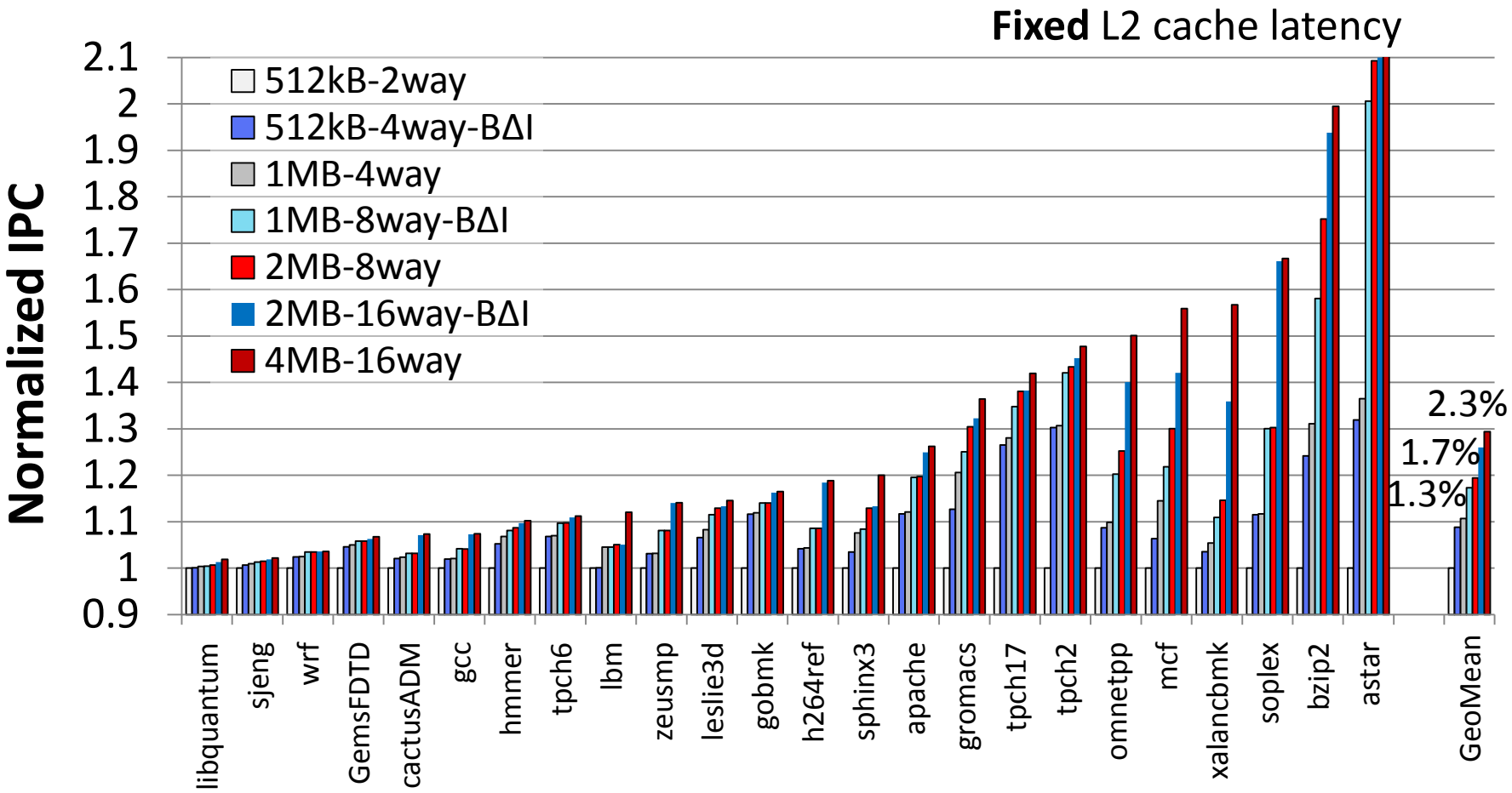




BΔI achieves the performance of a 2X-size cache

Performance improves due to the decrease in MPKI

Single-Core: Effect on Cache Capacity



BΔI achieves performance close to the upper bound

Multi-Core Workloads

- Application classification based on

  Compressibility: effective cache size increase

  (Low Compr. (***LC***) < 1.40, High Compr. (***HC***) >= 1.40)

  Sensitivity: performance gain with more cache

  (Low Sens. (***LS***) < 1.10, High Sens. (***HS***) >= 1.10; 512kB -> 2MB)

- Three classes of applications:
  - LCLS, HCLS, HCHS, **no LCHS** applications

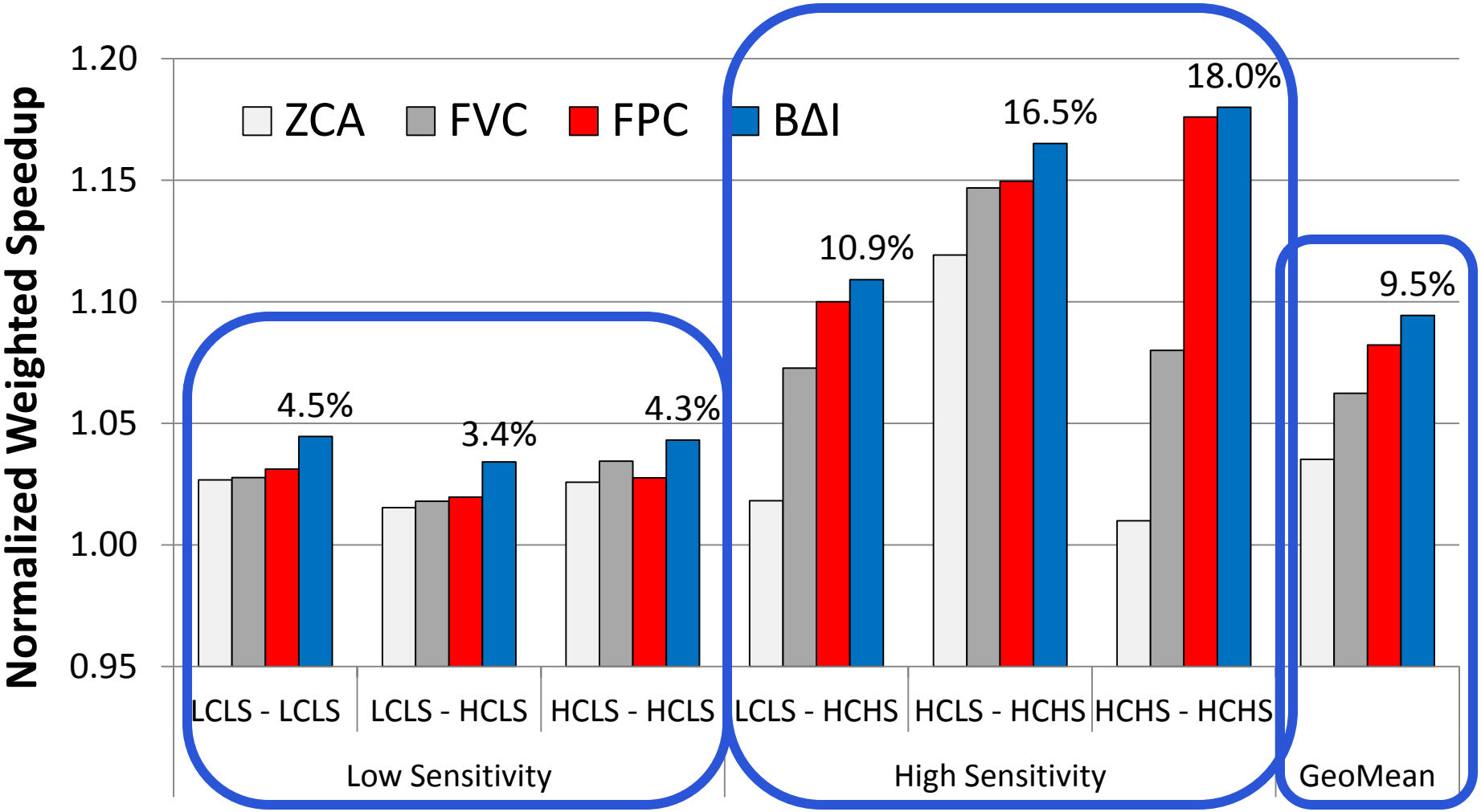
- For 2-core - **random** mixes of each possible class pairs (20 each, 120 total workloads)

Multi-Core Workloads

Cat.	Name	C. Ratio	Sens.	Name	C. Ratio	Sens.	Name	C. Ratio	Sens.
LCLS	gromacs	1.43 / L	L	hmmer	1.03 / L	L	lbm	1.00 / L	L
	sphinx	1.10 / L	L	tpch17	1.18 / L	L	wrf	1.01 / L	L
HCLS	apache	1.60 / H	L	zeusmp	1.99 / H	L	gcc	1.99 / H	L
	sjeng	1.50 / H	L	tpch2	1.54 / H	L	tpch6	1.93 / H	L
HCHS	astar	1.74 / H	H	bzip2	1.60 / H	H	mcf	1.52 / H	H
	soplex	1.99 / H	H	h264ref	1.52 / H	H			

Cat.	Name	C. Ratio	Sens.	Name	C. Ratio	Sens.
LCLS	libquantum	1.25 / L	L	leslie3d	1.41 / L	L
HCLS	GemsFDTD	1.99 / H	L	gobmk	1.99 / H	L
	cactusADM	1.97 / H	L			
HCHS	xalancbmk	1.61 / H	H	omnetpp	1.58 / H	H

Multi-Core: Weighted Speedup



If at least one application is sensitive, then the performance improves

BΔI performance improvement is the highest (9.5%)

Other Results in Paper

- Sensitivity study of having **more** than 2X tags
  - Up to 1.98 average compression ratio
- Effect on **bandwidth** consumption
  - 2.31X decrease on average
- Detailed quantitative comparison with prior work
- **Cost analysis** of the proposed changes
  - 2.3% L2 cache area increase

Conclusion

- A new **Base-Delta-Immediate** compression mechanism
- Key insight: many cache lines can be efficiently represented using **base + delta encoding**
- Key properties:
  - **Low** latency decompression
  - **Simple** hardware implementation
  - **High compression ratio** with high coverage
- **Improves** *cache hit ratio* and *performance* of both single-core and multi-core workloads
  - Outperforms state-of-the-art cache compression techniques: FVC and FPC

Linearly Compressed Pages: A Main Memory Compression Framework with Low Complexity and Low Latency

Gennady Pekhimenko, Vivek Seshadri, Yoongu Kim, Hongyi Xin, Onur Mutlu, Phillip B. Gibbons*, Michael A. Kozuch*, Todd C. Mowry

Carnegie Mellon

intel

Executive Summary

- Main memory is a limited shared resource
- **Observation**: Significant data redundancy
- **Idea**: Compress data in main memory
- **Problem**: How to avoid latency increase?
- **Solution**: Linearly Compressed Pages (LCP): fixed-size cache line granularity compression
  1. Increases capacity (**69%** on average)
  2. Decreases bandwidth consumption (**46%**)
  3. Improves overall performance (**9.5%**)

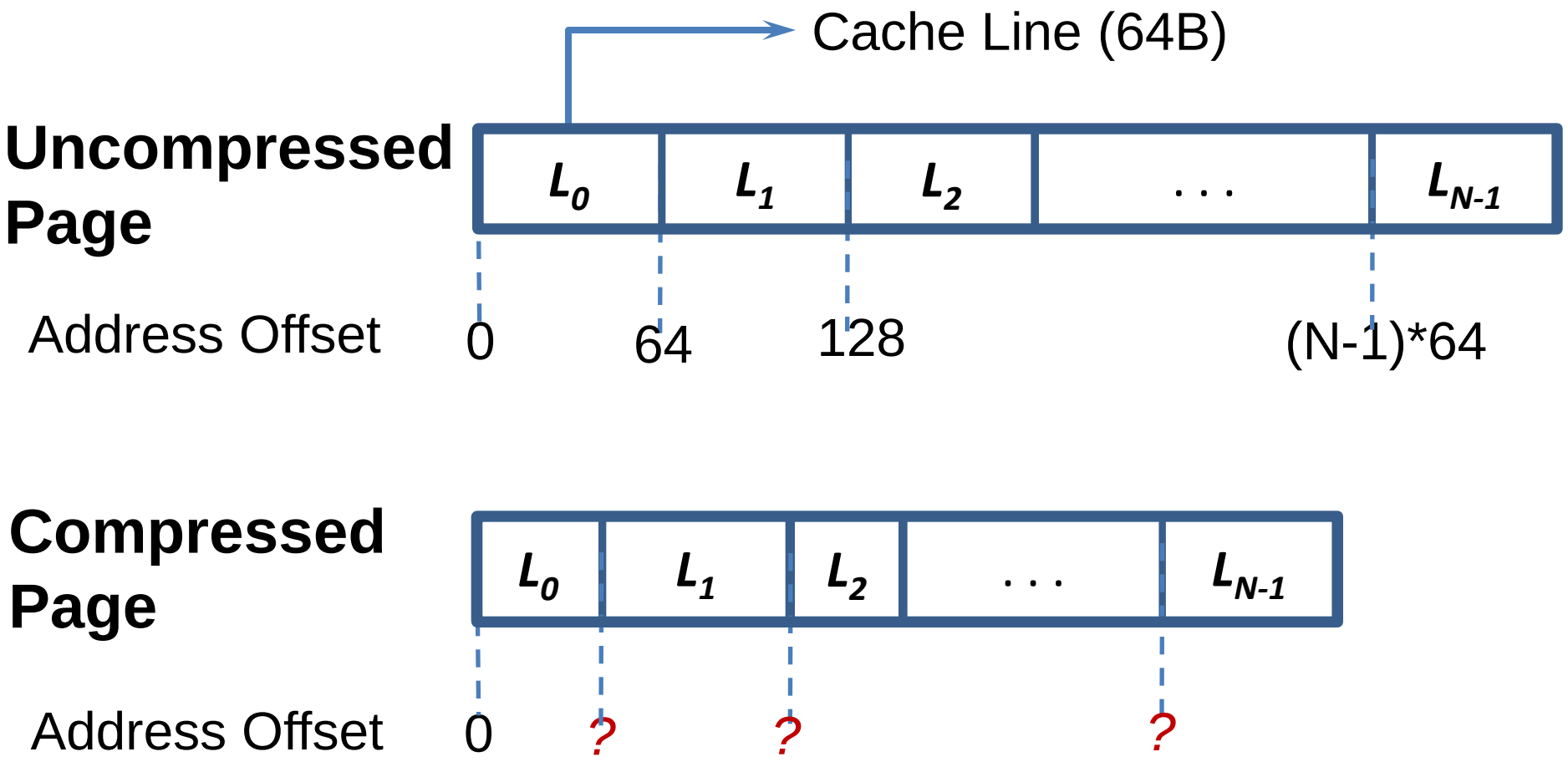
Challenges in Main Memory Compression

1. Address Computation

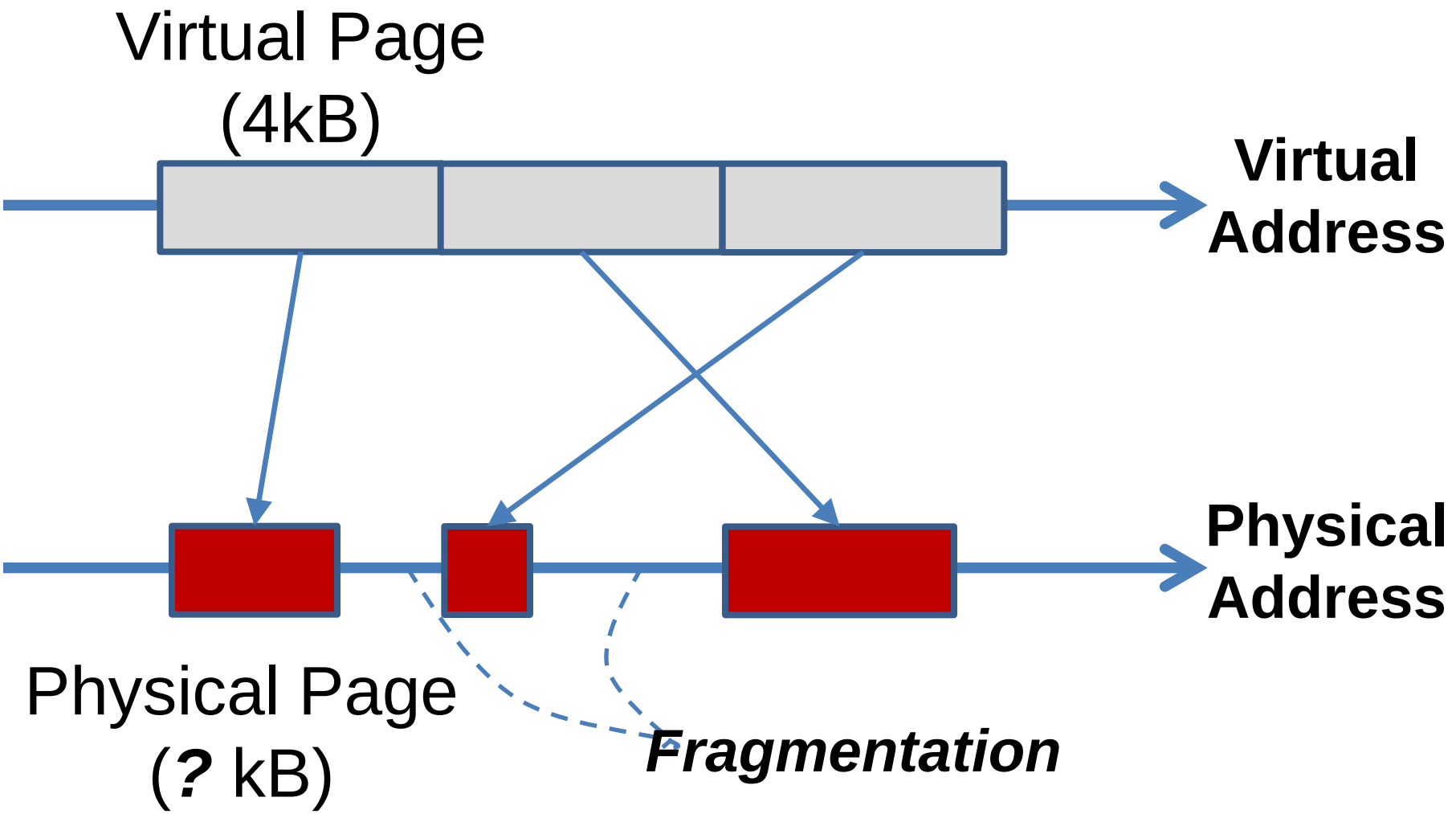
2. Mapping and Fragmentation

3. Physically Tagged Caches

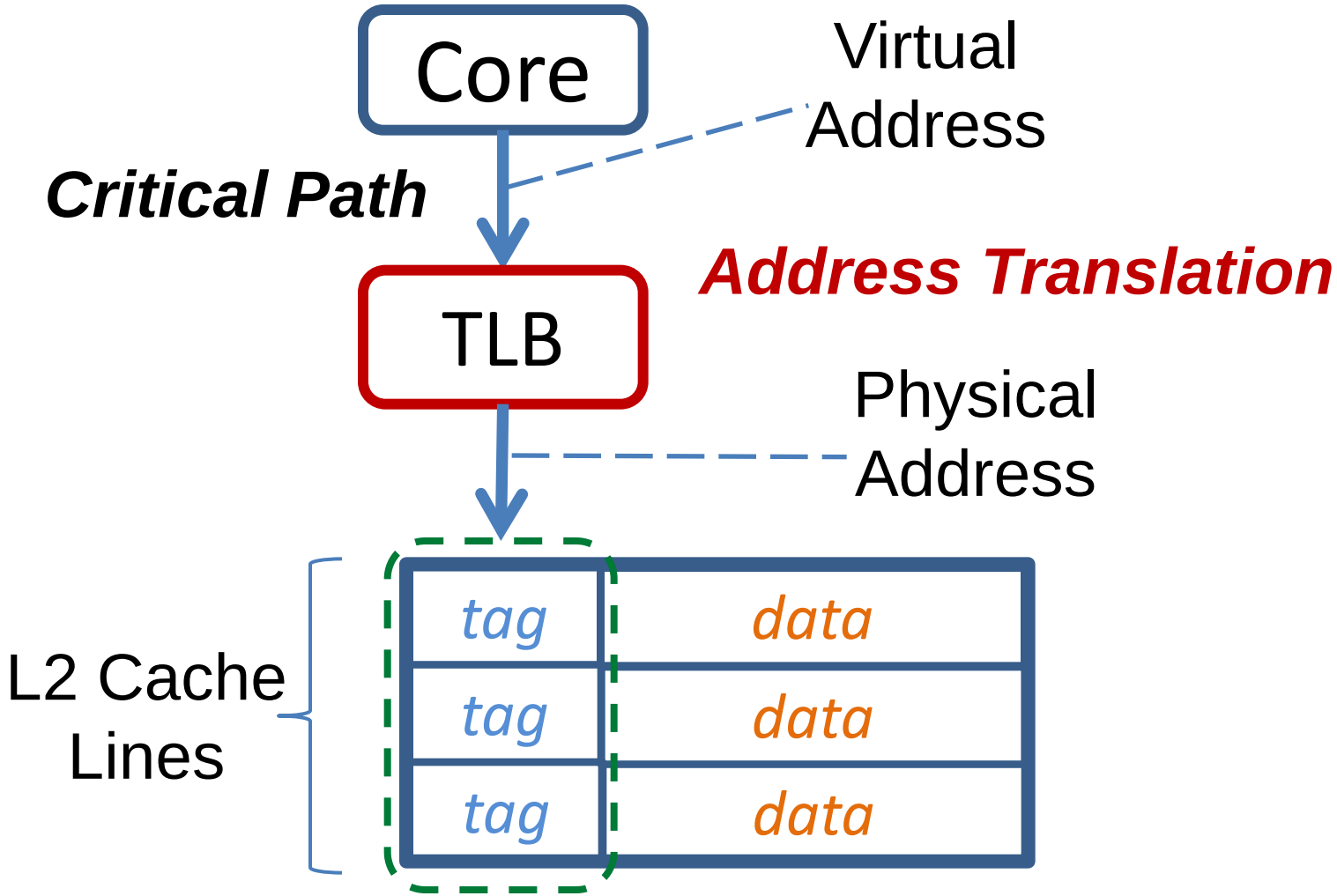
Address Computation



Mapping and Fragmentation



Physically Tagged Caches



Shortcomings of Prior Work

Compression Mechanisms	Access Latency	Decompression Latency	Complexity	Compression Ratio
IBM MXT *[IBM J.R.D. '01]*	✗	✗	✗	✓

Shortcomings of Prior Work

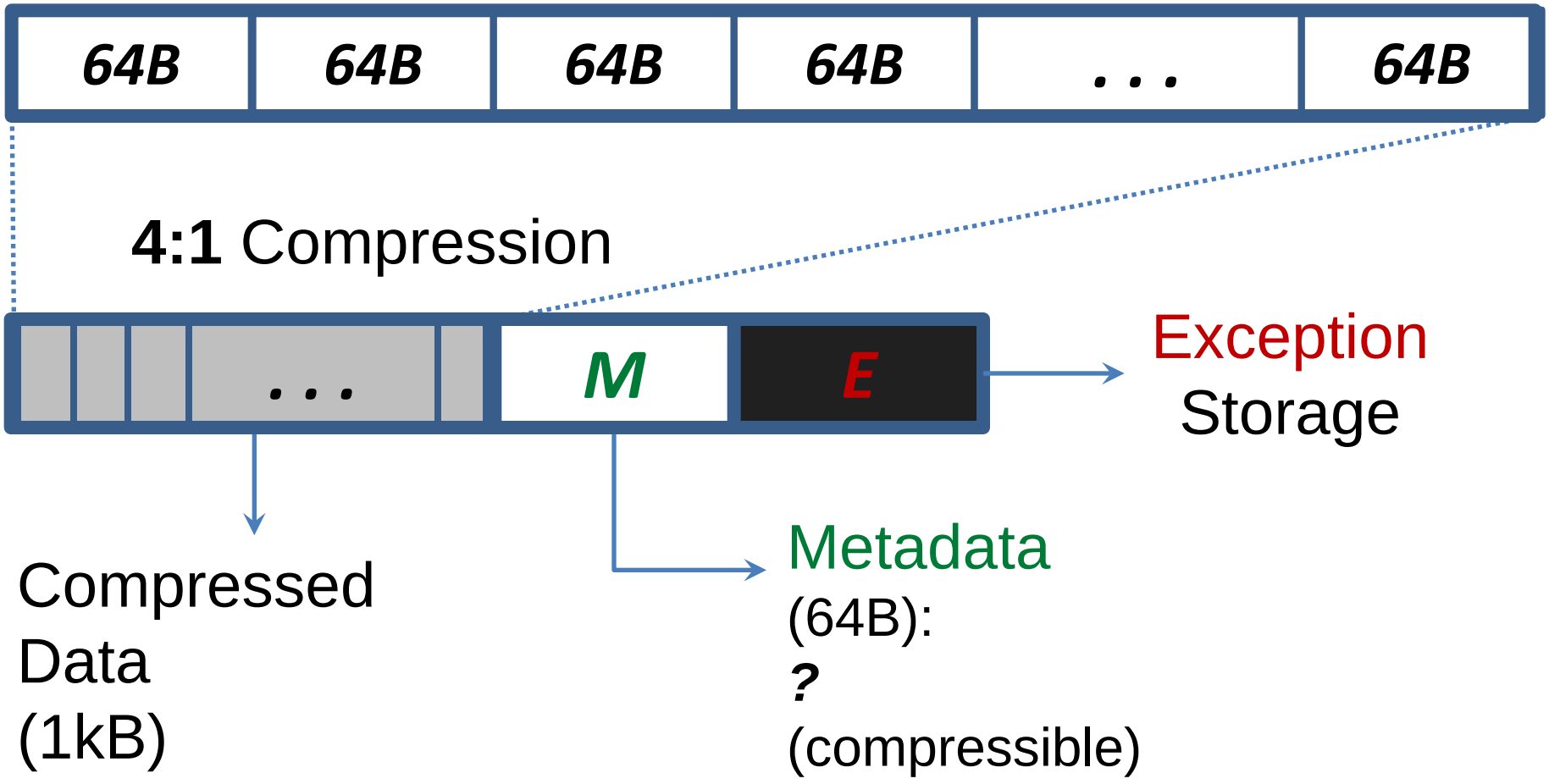
Compression Mechanisms	Access Latency	Decompression Latency	Complexity	Compression Ratio
IBM MXT *[IBM J.R.D. '01]*	✗	✗	✗	✓
Robust Main Memory Compression *[ISCA'05]*	✗	✓	✗	✓

Shortcomings of Prior Work

Compression Mechanisms	Access Latency	Decompression Latency	Complexity	Compression Ratio
IBM MXT *[IBM J.R.D. '01]*	✗	✗	✗	✓
Robust Main Memory Compression *[ISCA'05]*	✗	✓	✗	✓
LCP: Our Proposal	✓	✓	✓	✓

Linearly Compressed Pages (LCP): Key Idea

Uncompressed Page (4kB: 64*__64B__)



LCP Overview

- Page Table entry extension
  - compression type and size
  - extended physical base address
- Operating System management support
  - **4** memory pools (512B, 1kB, 2kB, 4kB)
- Changes to cache tagging logic
  - physical page base address + **cache line index** (within a page)
- Handling page overflows
- Compression algorithms: **BDI** *[PACT'12]* , **FPC** *[ISCA'04]*

LCP Optimizations

- **Metadata** cache
  - Avoids additional requests to metadata
- Memory bandwidth reduction:

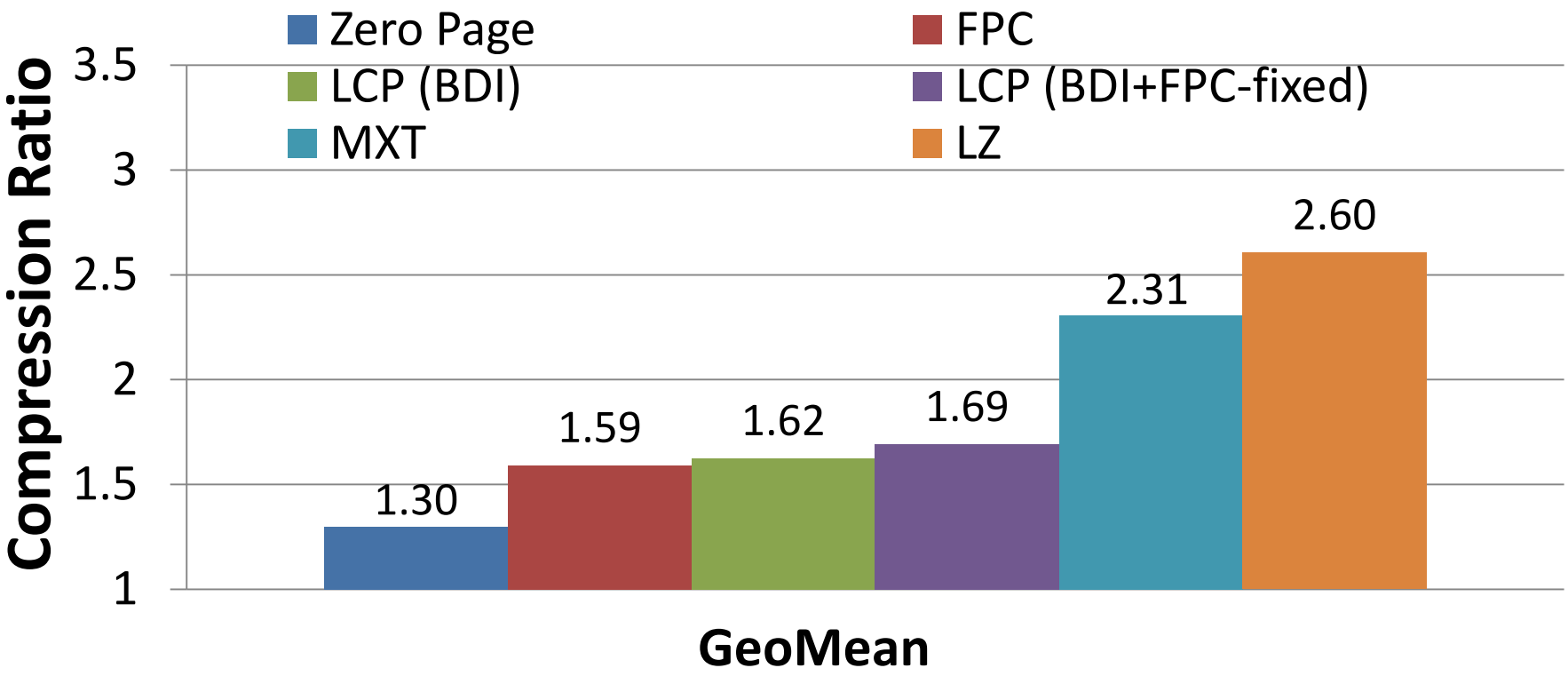
- Zero pages and zero cache lines
  - Handled separately in TLB (1-bit) and in metadata (1-bit per cache line)
- Integration with cache compression
  - BDI and FPC

Methodology

- **Simulator**
  - x86 event-driven simulators
    - Simics-based *[Magnusson+, Computer'02]* for CPU
    - Multi2Sim *[Ubal+, PACT'12]* for GPU
- **Workloads**
  - SPEC2006 benchmarks, TPC, Apache web server, GPGPU applications
- **System Parameters**
  - L1/L2/L3 cache latencies from CACTI *[Thoziyoor+, ISCA'08]*
  - 512kB - 16MB L2, simple memory model

Compression Ratio Comparison

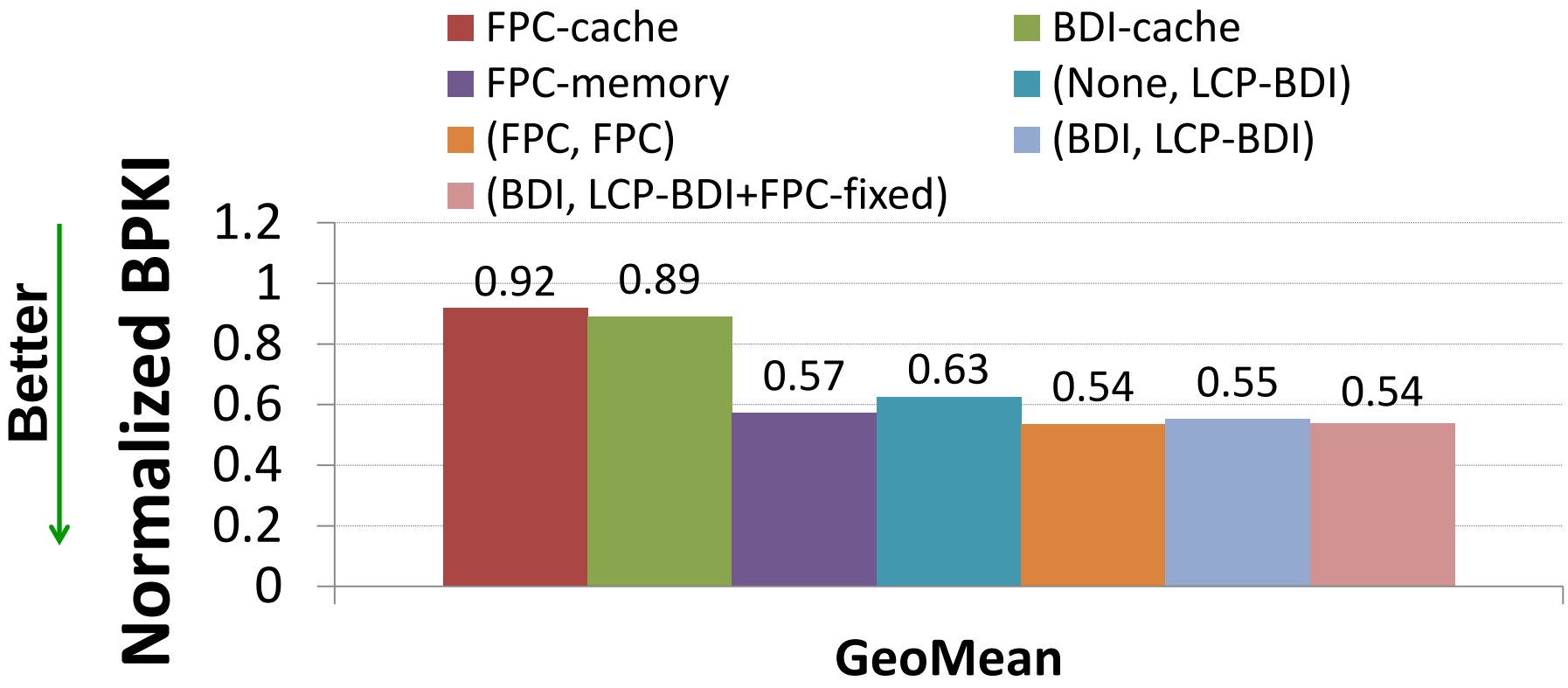
SPEC2006, databases, web workloads, 2MB L2 cache



LCP-based frameworks achieve competitive average compression ratios with prior work

Bandwidth Consumption Decrease

SPEC2006, databases, web workloads, 2MB L2 cache



LCP frameworks significantly reduce bandwidth (**46**%)

Performance Improvement

Cores	LCP-BDI	(BDI, LCP-BDI)	(BDI, LCP-BDI+FPC-fixed)
1	6.1%	**9.5%**	9.3%
2	13.9%	**23.7%**	23.6%
4	10.7%	**22.6%**	22.5%

LCP frameworks significantly improve performance

Conclusion

- A new main memory compression framework called **LCP(Linearly Compressed Pages)**
  - **Key idea: fixed size** for compressed cache lines within a page and **fixed compression algorithm** per page

- LCP evaluation:
  - Increases capacity (**69%** on average)
  - Decreases bandwidth consumption (**46%**)
  - Improves overall performance (**9.5%**)
  - Decreases energy of the off-chip bus (**37**%)