Practical Data Compression for Modern Memory Hierarchies

Thesis Oral

Gennady Pekhimenko

Committee:
Todd Mowry (Co-chair)
Onur Mutlu (Co-chair)
Kayvon Fatahalian
David Wood, University of Wisconsin-Madison
Doug Burger, Microsoft
Michael Kozuch, Intel

Presented in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Performance and Energy Efficiency

Energy efficiency

Applications today are *data-intensive*

MEMCACHED

amazon DynamoDB

Memory Caching

Databases

Graphics

Computation vs. Communication

Modern memory systems are *bandwidth constrained*

Data movement is very costly

- Integer operation: **~1 pJ**
- Floating operation: **~20 pJ**
- Low-power memory access: **~1200 pJ**

Implications

- **½ bandwidth** of modern mobile phone memory **exceeds power budget**
- Transfer **less** or keep data **near processing units**

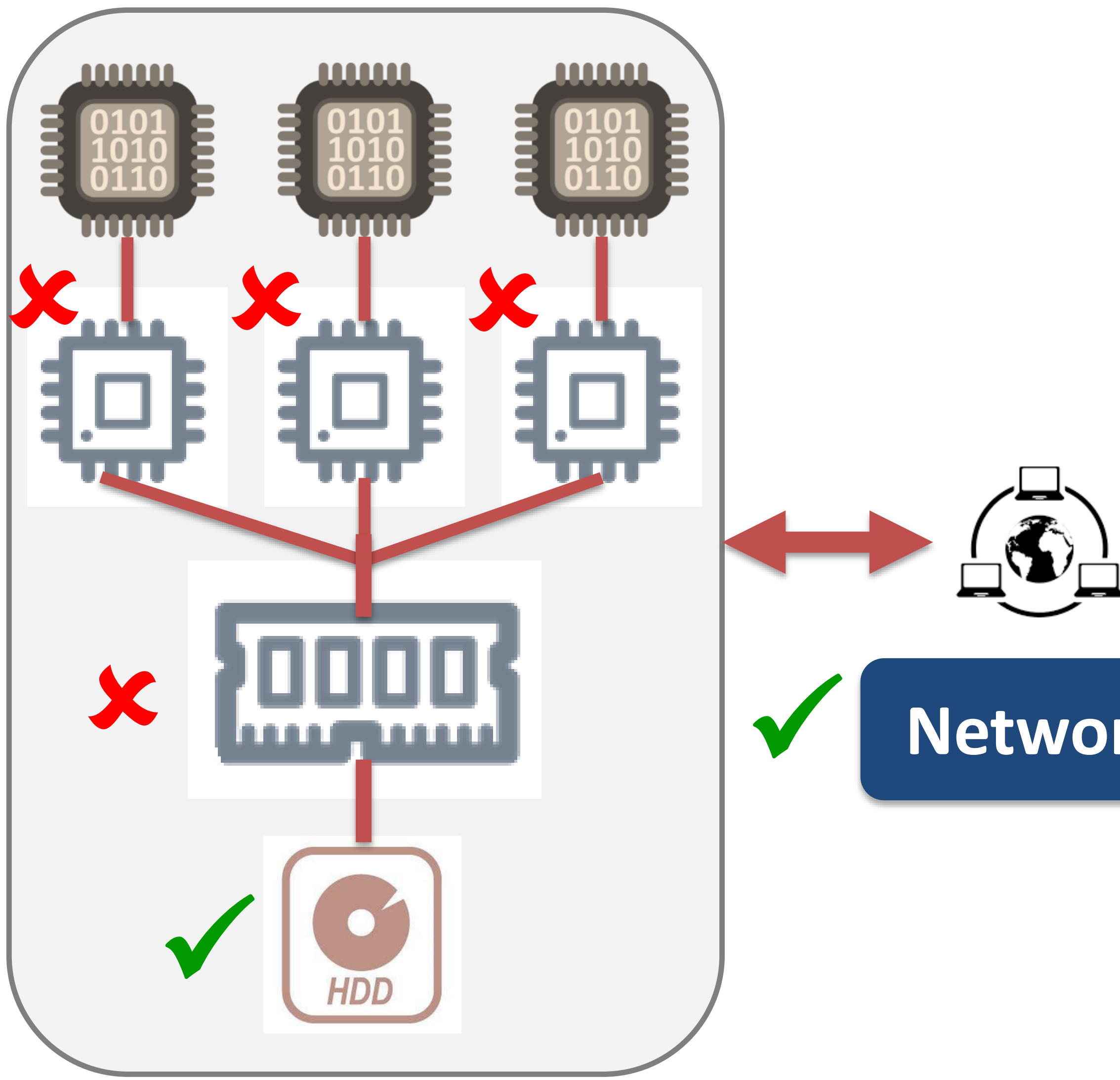
Data Compression across the System

Processor

Cache

Memory

Disk



Network

Software vs. Hardware Compression

	Software	vs.	Hardware
Layer	Disk		Cache/Memory
Latency	**milliseconds**		**nanoseconds**
Algorithms	Dictionary-based		Arithmetic

Existing **dictionary-based** algorithms are **too slow** for main memory hierarchies

Key Challenges for Compression in Memory Hierarchy

- Fast Access Latency
- Practical Implementation and Low Cost
- High Compression Ratio

Thesis Statement

It is possible to develop a new set of designs for data compression within modern memory hierarchies that is:

- ✓**Fast**
- ✓**Simple**
- ✓**Effective**

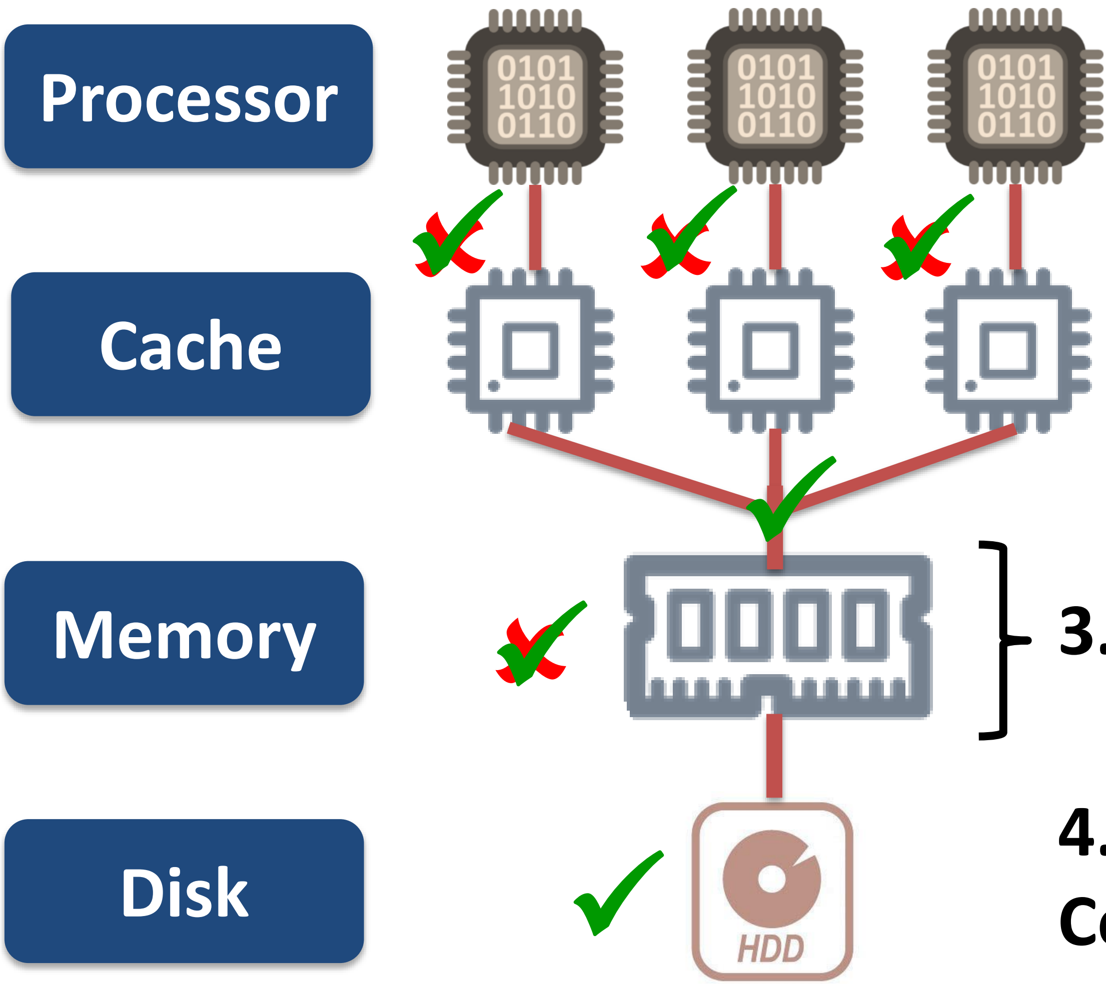
in saving storage space and consumed bandwidth

so that the resulting improvements in performance, cost, and energy efficiency will make it attractive to implement in future systems

Contributions of This Dissertation

- **Base-Delta-Immediate (BDI)** Compression algorithm with low latency and high compression ratio
- **Compression-Aware Management Policies (CAMP)** that incorporate compressed block size into cache management decisions
- **Linearly Compressed Pages (LCP)** framework for efficient main memory compression
- **Toggle-Aware Bandwidth** compression mechanisms for energy-efficient bandwidth compression

Practical Data Compression in Memory



1. Cache Compression

2. Compression and Cache Replacement

PACT 2012

HPCA 2015

3. Memory Compression

MICRO 2013

4. Bandwidth Compression

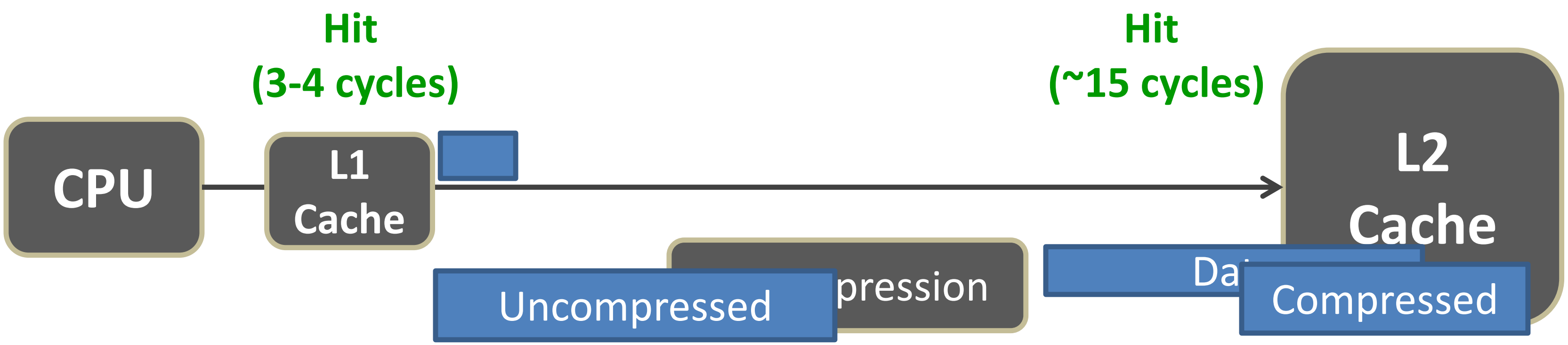
HPCA 2016

PACT
2012


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

1. Cache Compression

Background on Cache Compression



- Key requirement:
  - **Low decompression latency**

Key Data Patterns in Real Applications

Zero Values: initialization, sparse matrices, NULL pointers

0x00000000	0x00000000	0x00000000	0x00000000	…

Repeated Values: common initial values, adjacent pixels

0x*000000***C0**	0x*000000***C0**	0x*000000***C0**	0x*000000***C0**	…

Narrow Values: small values stored in a big data type

0x*000000***C0**	0x*000000***C8**	0x*000000***D0**	0x*000000***D8**	…

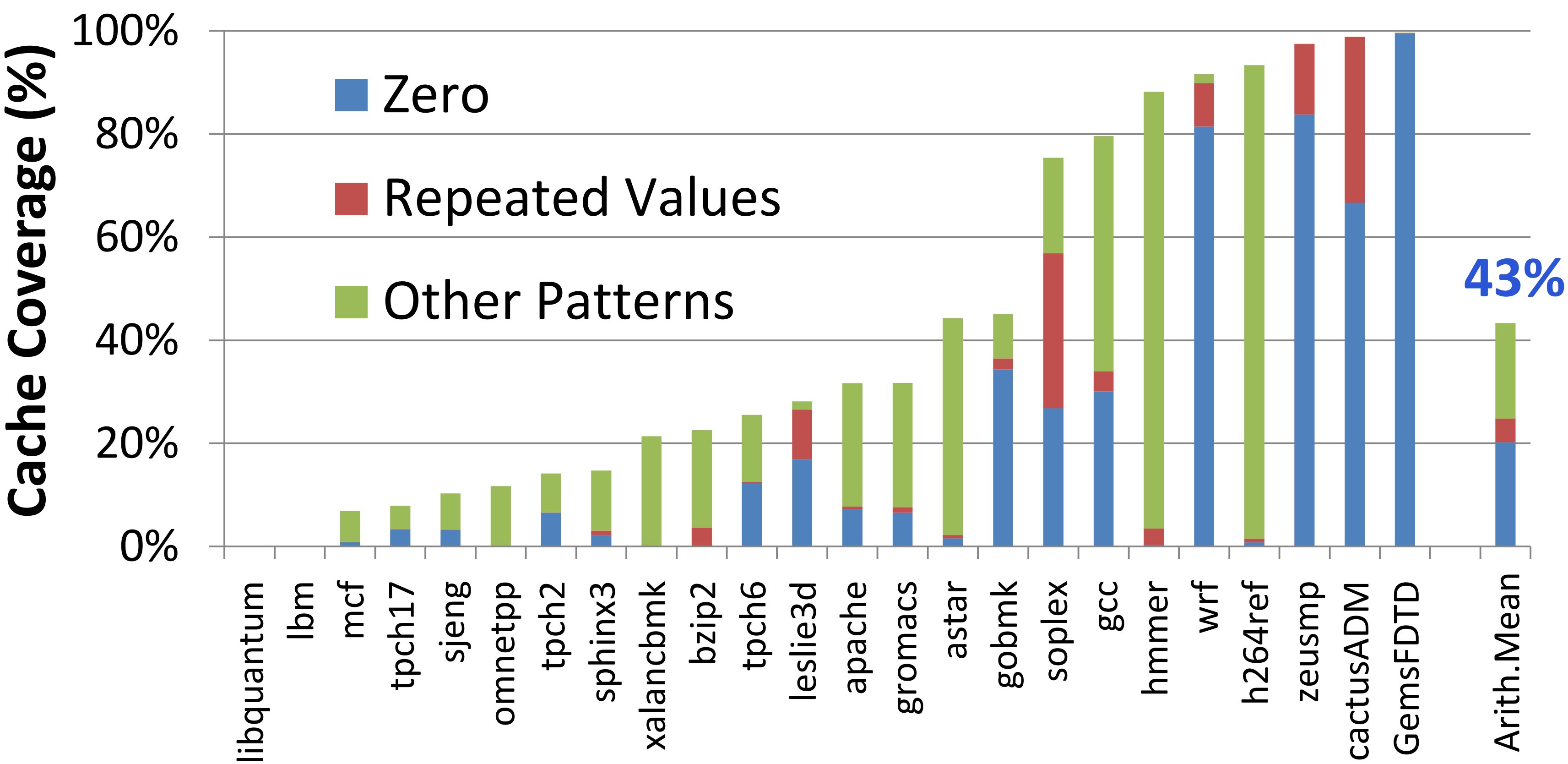
Other Patterns: pointers to the same memory region

0x*C04039***C0**	0x*C04039***C8**	0x*C04039***D0**	0x*C04039***D8**	…

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

Zero Values: initialization, sparse matrices, NULL pointers

0x00000000	0x00000000	0x00000000	0x00000000	…

Repeated Values: common initial values, adjacent pixels

0x*000000***C0**	0x*000000***C0**	0x*000000***C0**	0x*000000***C0**	…

Narrow Values: small values stored in a big data type

0x*000000***C0**	0x*000000***C8**	0x*000000***D0**	0x*000000***D8**	…

Other Patterns: pointers to the same memory region

0x*C04039***C0**	0x*C04039***C8**	0x*C04039***D0**	0x*C04039***D8**	…

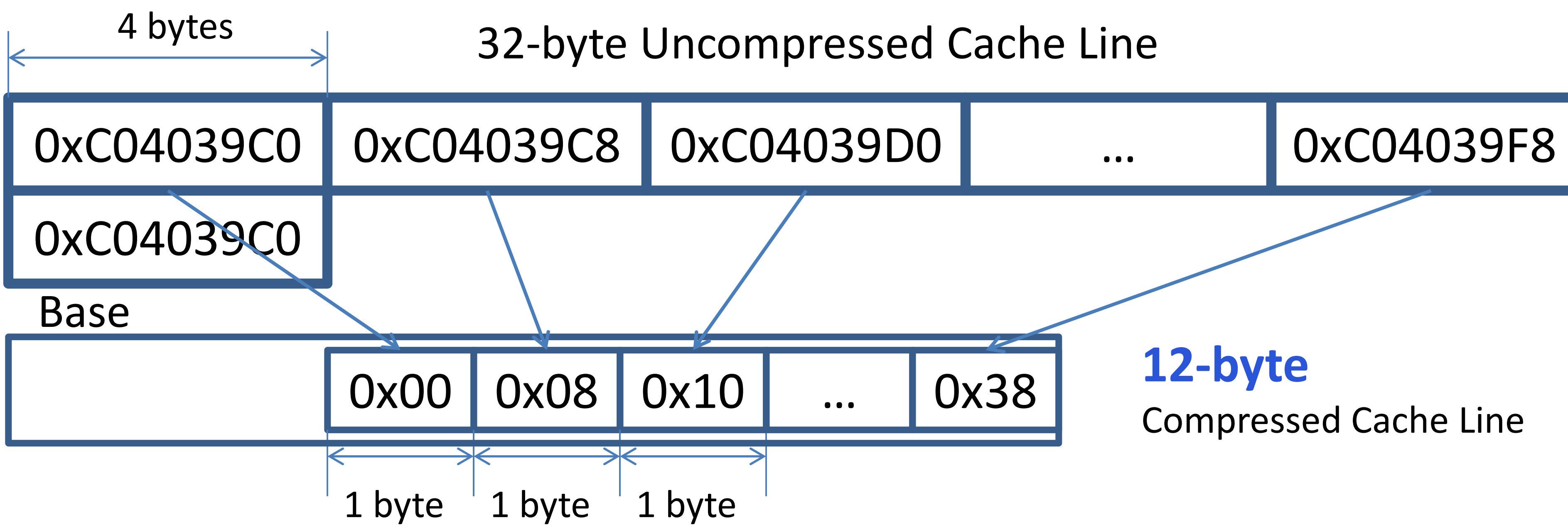
Key Data Patterns in Real Applications

Low Dynamic Range:

Differences between values are significantly smaller than the values themselves

- **Low Latency Decompressor**
- **Low Cost and Complexity Compressor**
- **Compressed Cache Organization**

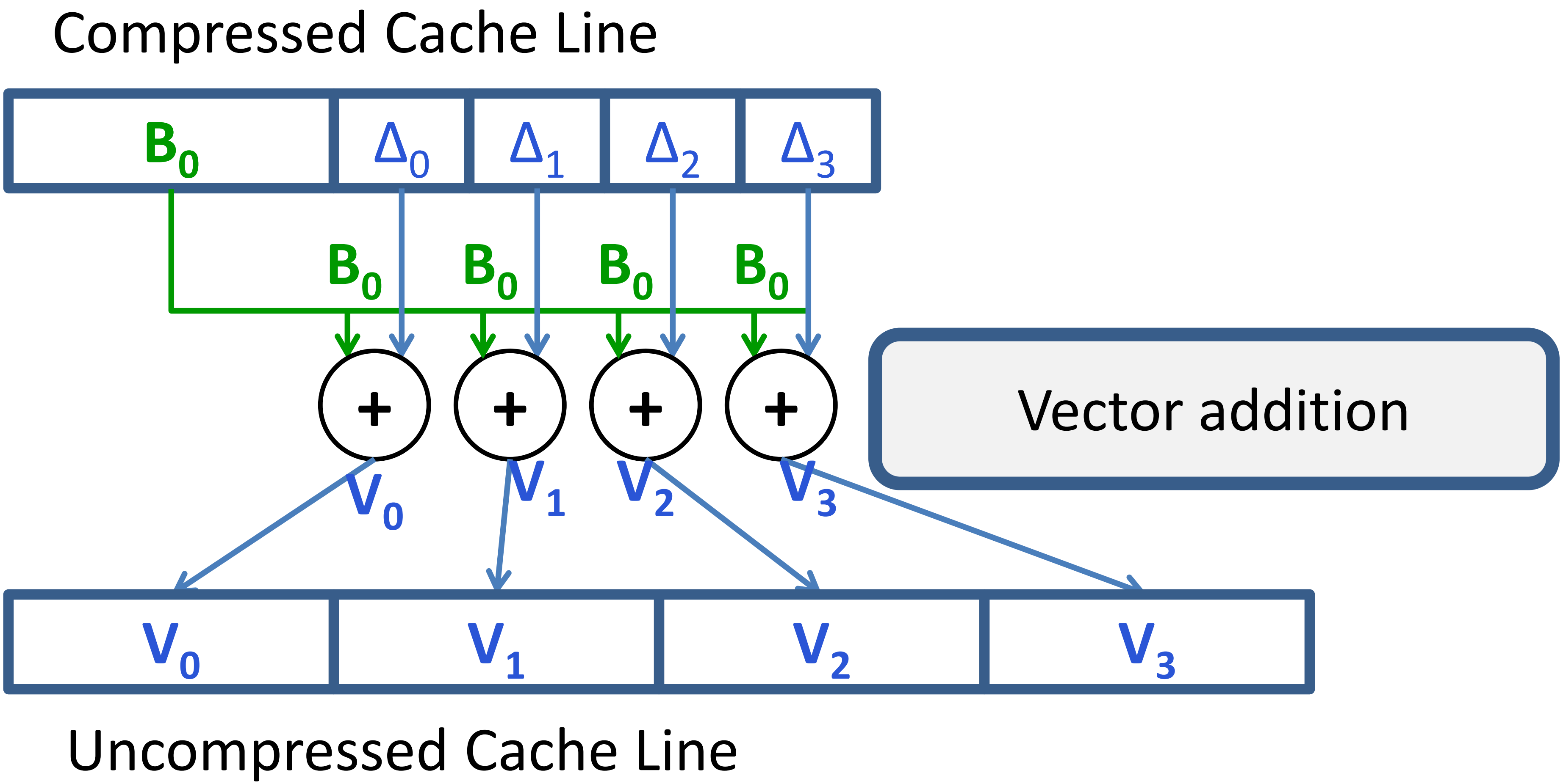
Key Idea: Base+Delta (B+Δ) Encoding



20 bytes saved

✓ **Effective:** good compression ratio

B+Δ Decompressor Design



✓ **Fast Decompression: 1-2 cycles**

Can We Get Higher Compression Ratio?

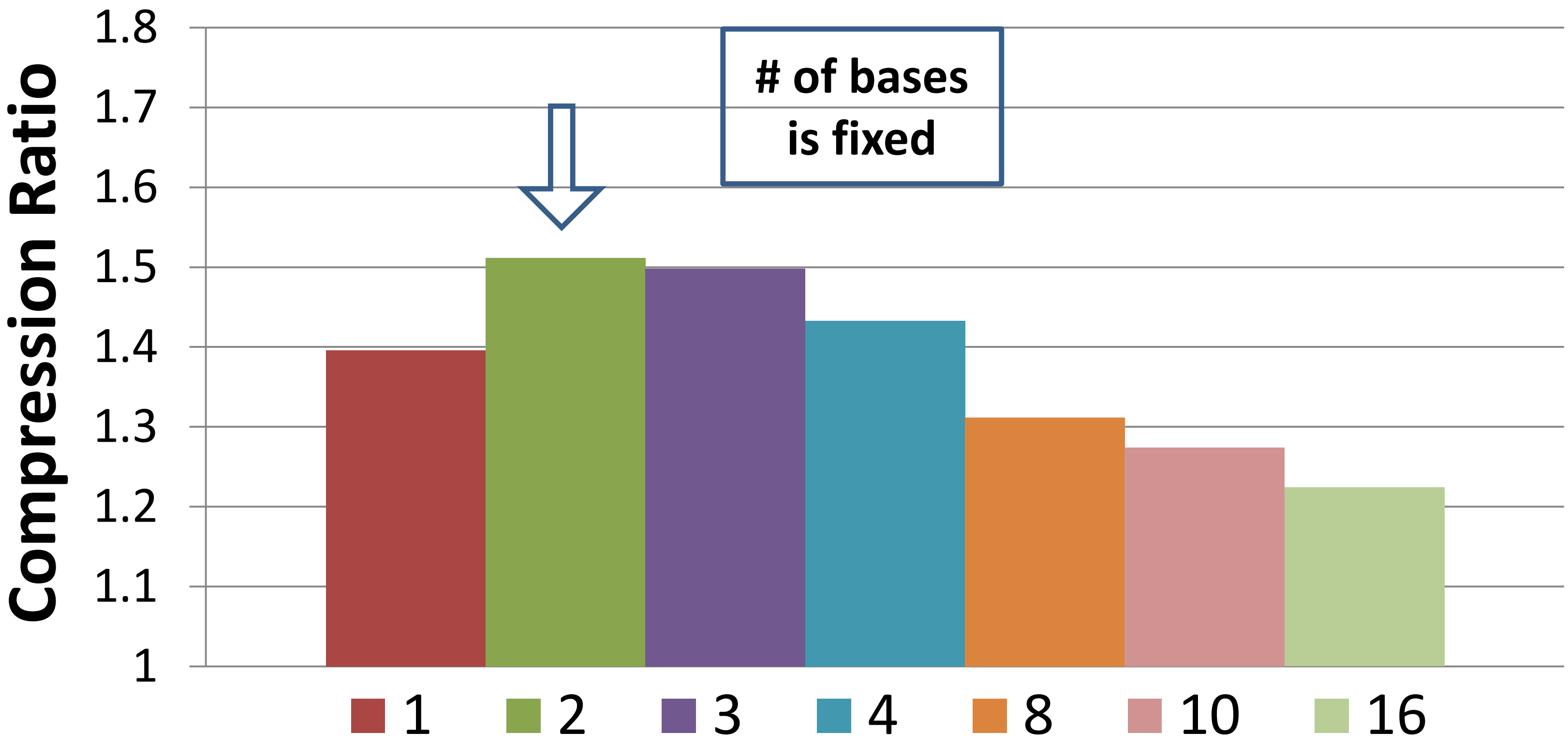
- Uncompressible cache line (with a single base):

0x09A40178	0x0000000B	0x09A4A838	0x0000000B	…

```
struct A {
  int* next;
  int count;};
```

- **Key idea -** use more bases
  - **More cache lines can be compressed**
  - **Unclear how to find these bases efficiently**
  - **Higher overhead (due to additional bases)**

B+Δ with Multiple Arbitrary Bases



✓ **2 bases – empirically** the best option

How to Find Two Bases Efficiently?

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

   ✓ **Base+Delta part**

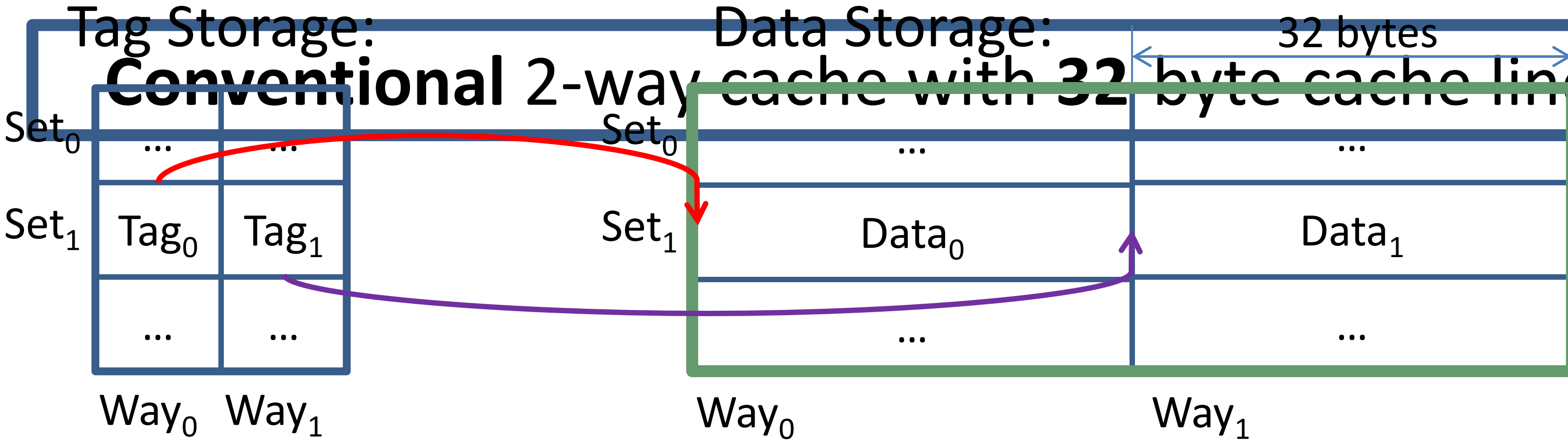
2. **Second base -** implicit base of **0**

   ✓ **Immediate part**

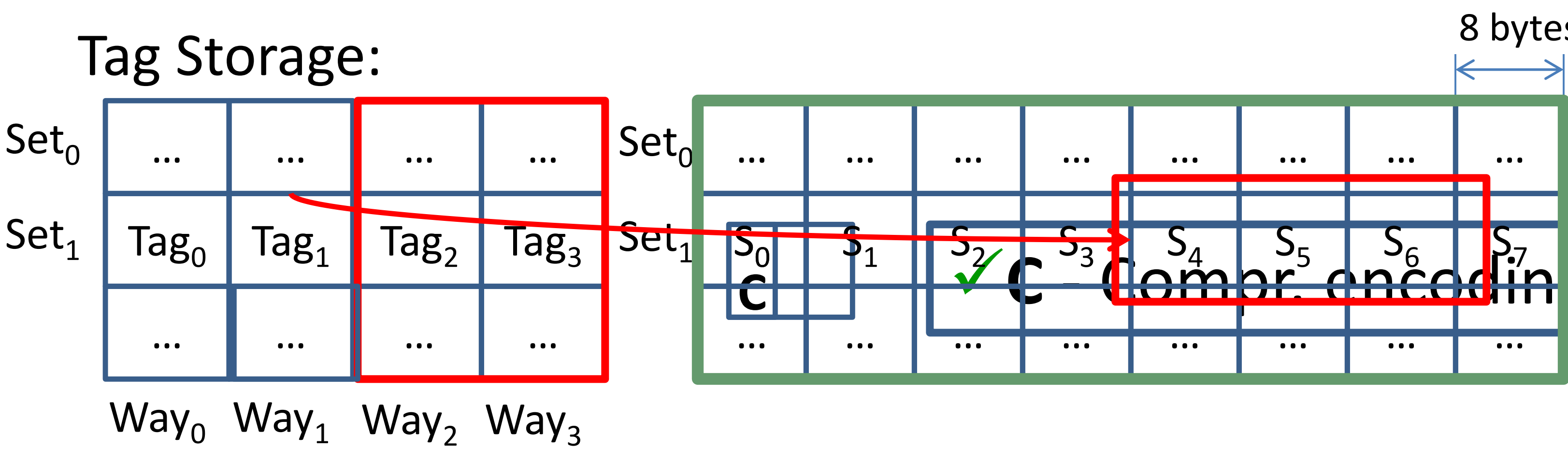
Base-Delta-Immediate (BΔI) Compression

BΔI Cache Organization

Conventional 2-way cache with **32** byte cache lines



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



✓ **C** - Compr. encoding bits

✓ Twice as many tags

✓ Tags map to multiple adjacent segments

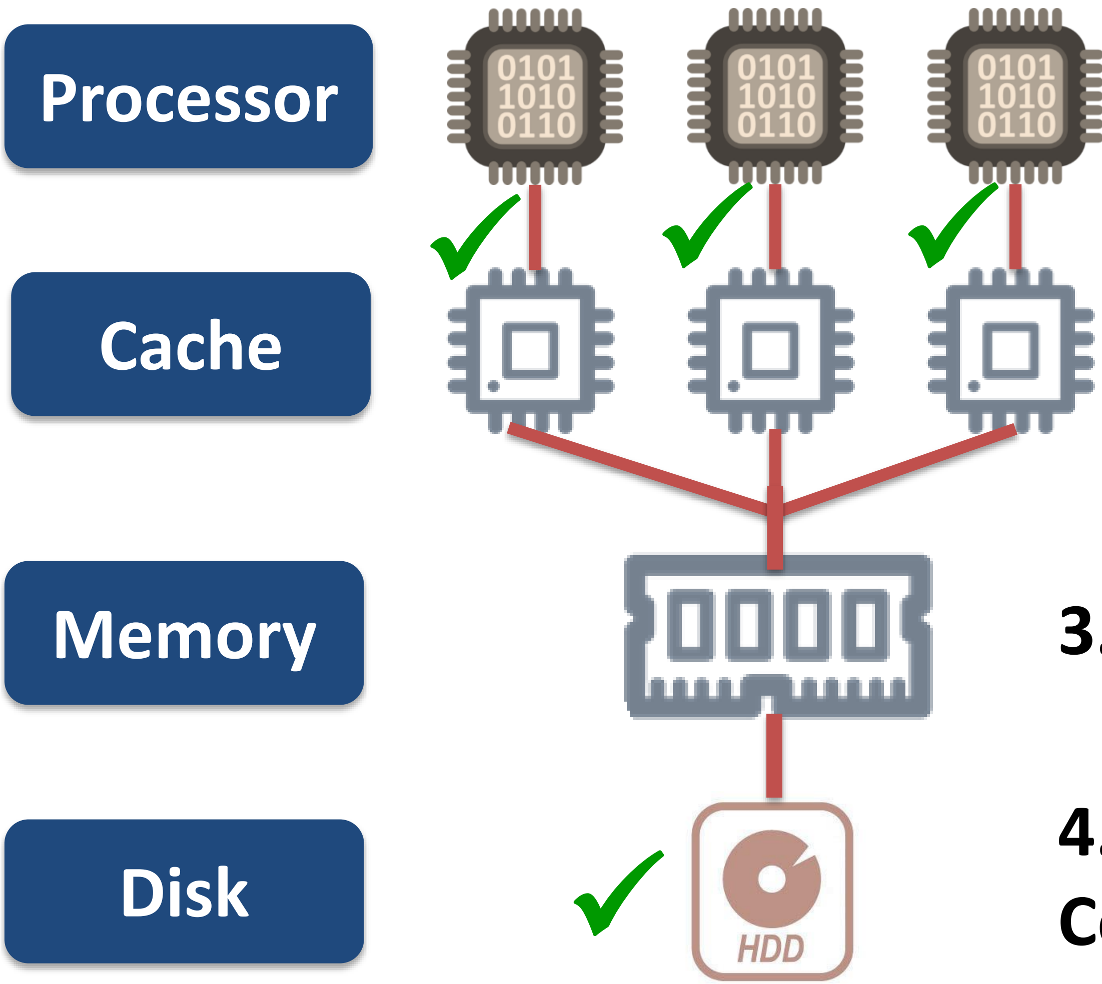
Methodology

- **Simulator**
  - x86 event-driven simulator (MemSim *[Seshadri+, PACT'12]*)
- **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
  - BDI (1-cycle decompression)
  - 4GHz, x86 in-order core, cache size (**1MB - 16MB**)

Comparison Summary

	Prior Work	vs. BΔI
Comp. Ratio	**1.51**	**1.53**
Decompression	**5**-9 cycles	**1-2** cycles
Compression	**3-10+ cycles**	**1-9 cycles**

Average performance of a **twice larger** cache



1. Cache Compression

2. Compression and Cache Replacement

3. Memory Compression

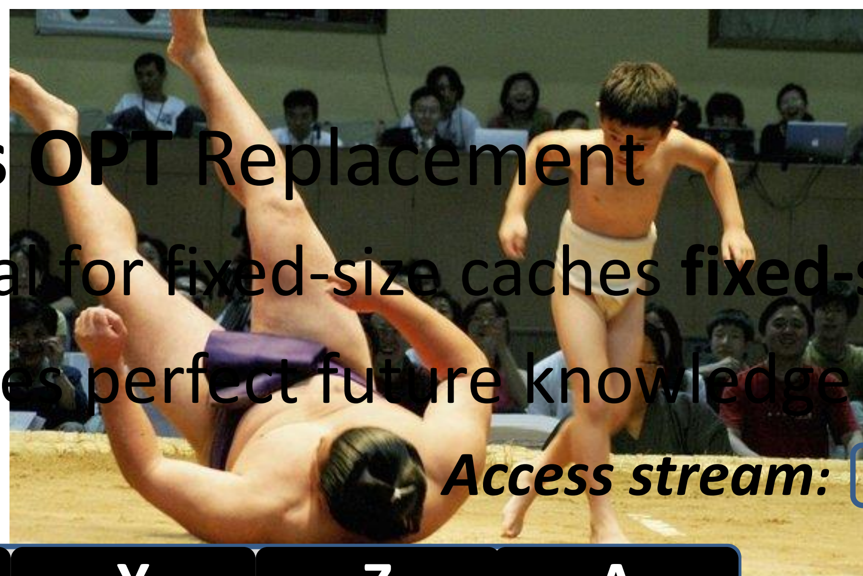
4. Bandwidth Compression

HPCA
2015

2. Compression and Cache Replacement

Cache Management Background

- Not only about **size**
  - Cache **management policies** are important
  - **Insertion, promotion and eviction**



- **Belady's OPT** Replacement
  - Optimal for fixed-size caches **fixed-size** blocks
  - Assumes perfect future knowledge

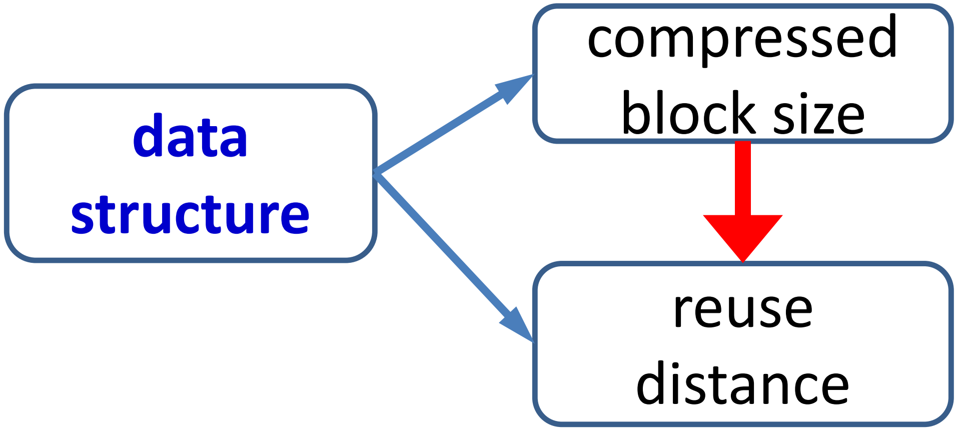
Access stream: **X A Y B C**

Cache:

X	Y	Z	A

Block Size Can Indicate Reuse

- Sometimes there is a **relation** between the **compressed block size** and **reuse** distance



- This relation can be **detected** through the compressed block size
- **Minimal** overhead to track this relation (compressed block information is a part of design)

Code Example to Support Intuition

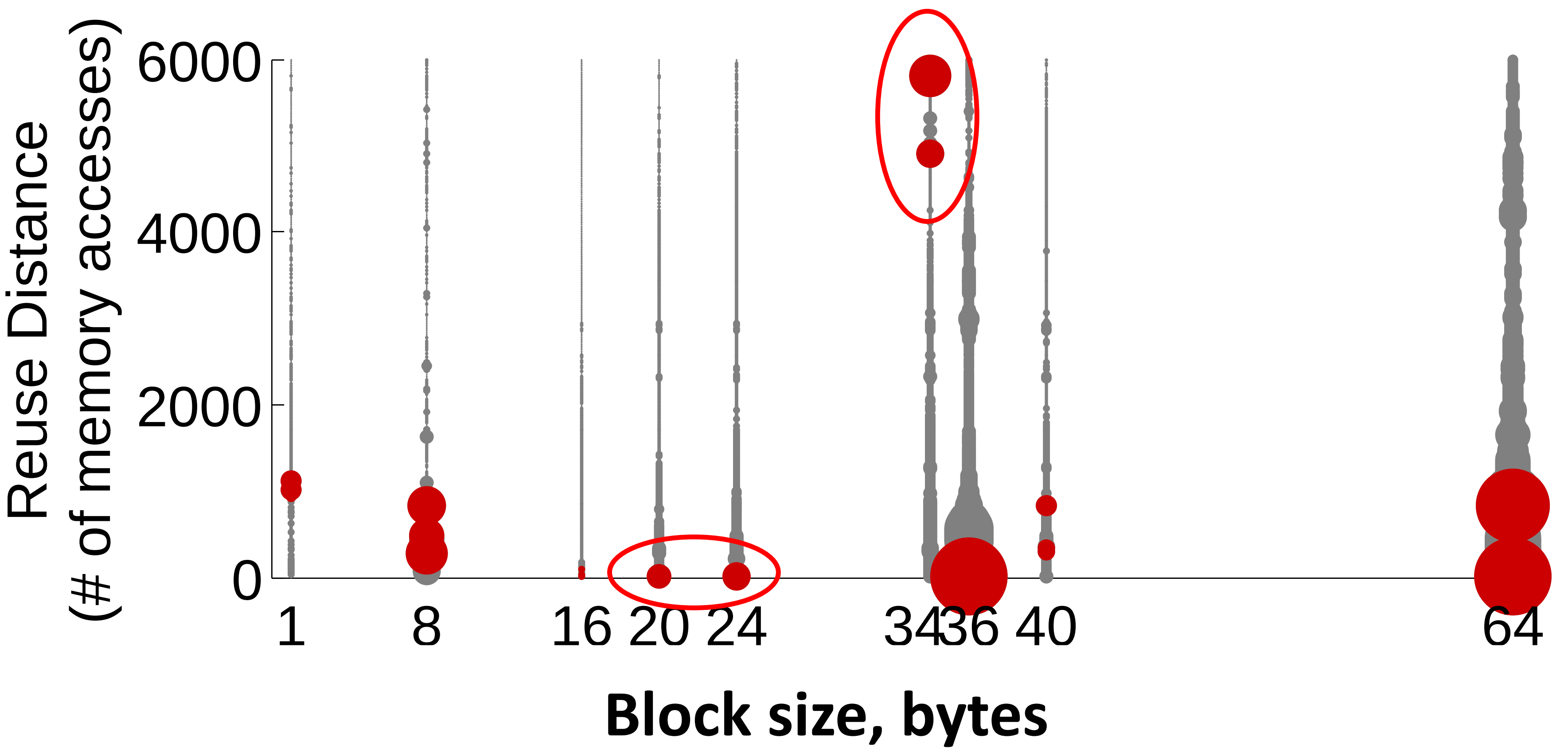
```
int A[N];              // small indices: compressible
double B[16];          // FP coefficients: incompressible
for (int i=0; i<N; i++) {
    int idx = A[i];
    for (int j=0; j<N; j++) {
      sum += B[(idx+j)%16];
    }
}
```

A[i]: long reuse, compressible

B[(idx+j)%16]: short reuse, incompressible

Compressed size can be an indicator of reuse distance

Block Size Can Indicate Reuse



Different sizes have different dominant reuse distances

Compression-Aware Management Policies (CAMP)

CAMP

SIP:
Size-based Insertion Policy

MVE:
Minimal-Value Eviction

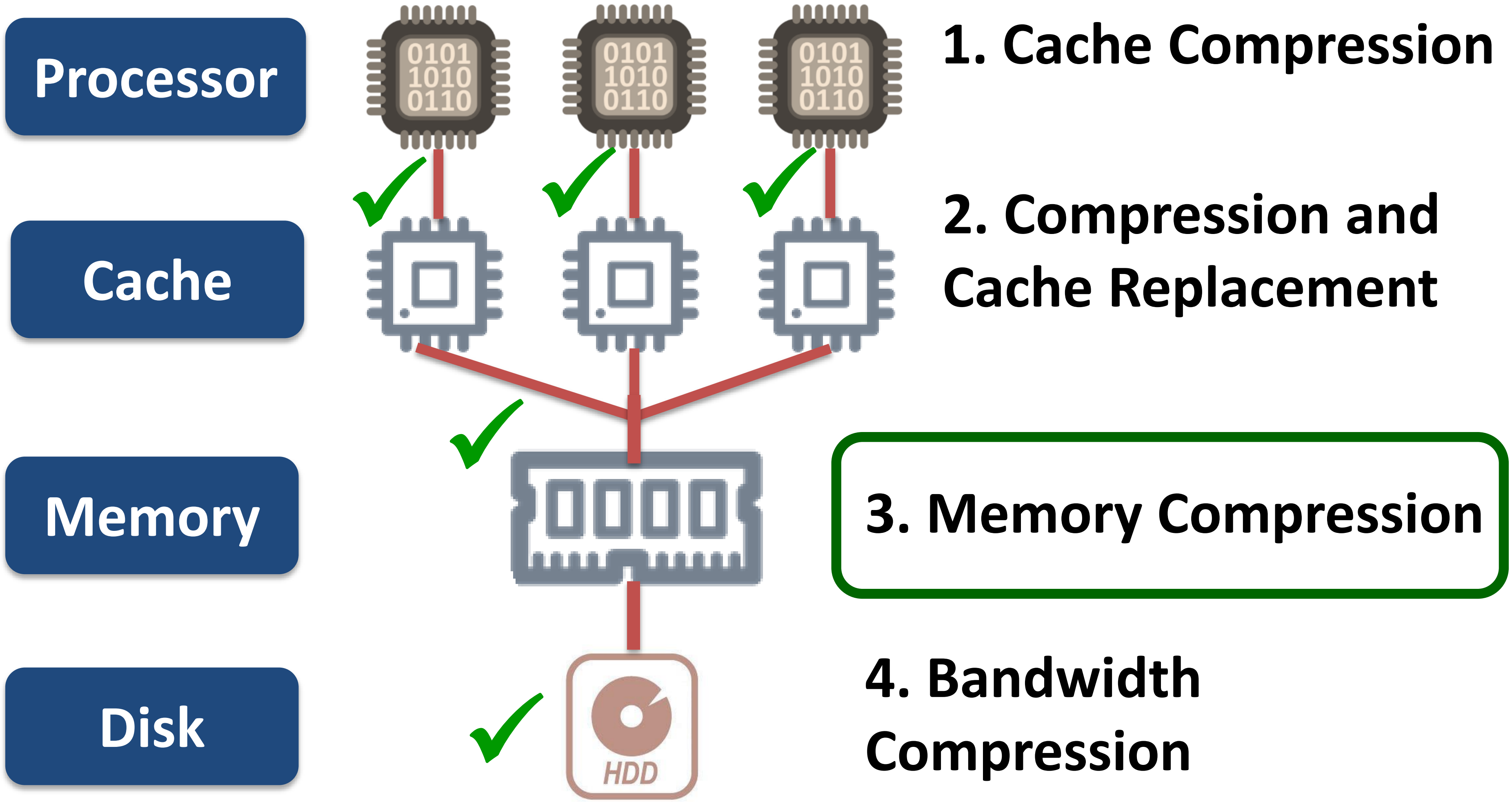
Value = **Probability of reuse** / **Compressed block size**

data structure

compressed block size

reuse distance

The benefits are on-par with cache compression - **2X additional** increase in capacity


Processor
Cache
Memory
Disk
0101
1010
0110
HDD
1. Cache Compression
2. Compression and Cache Replacement
3. Memory Compression
4. Bandwidth Compression

MICRO
2013


Linearly Compressed Pages: A Low-Complexity, Low-Latency Main Memory Compression Framework

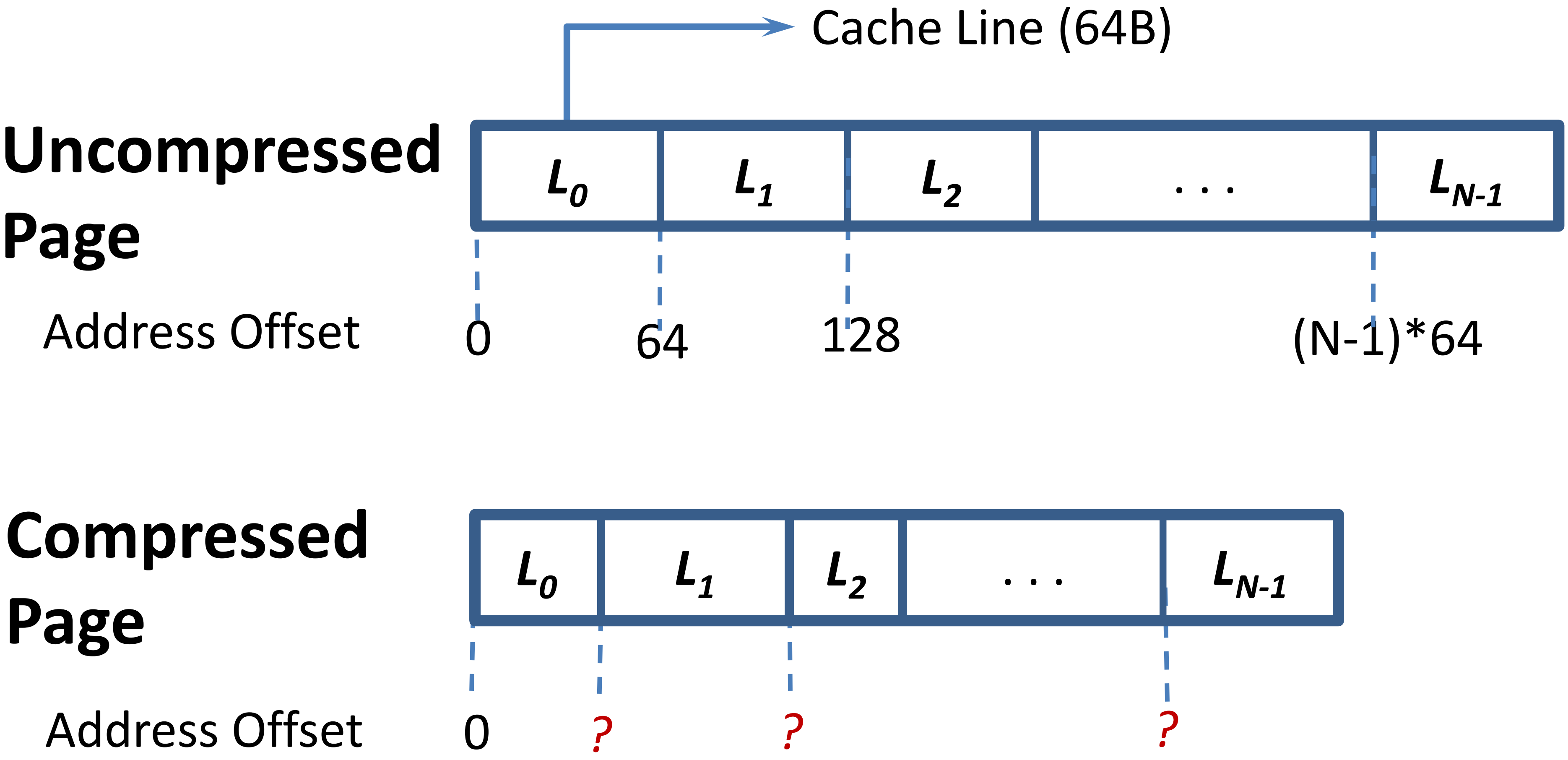
3. Main Memory Compression

Challenges in Main Memory Compression

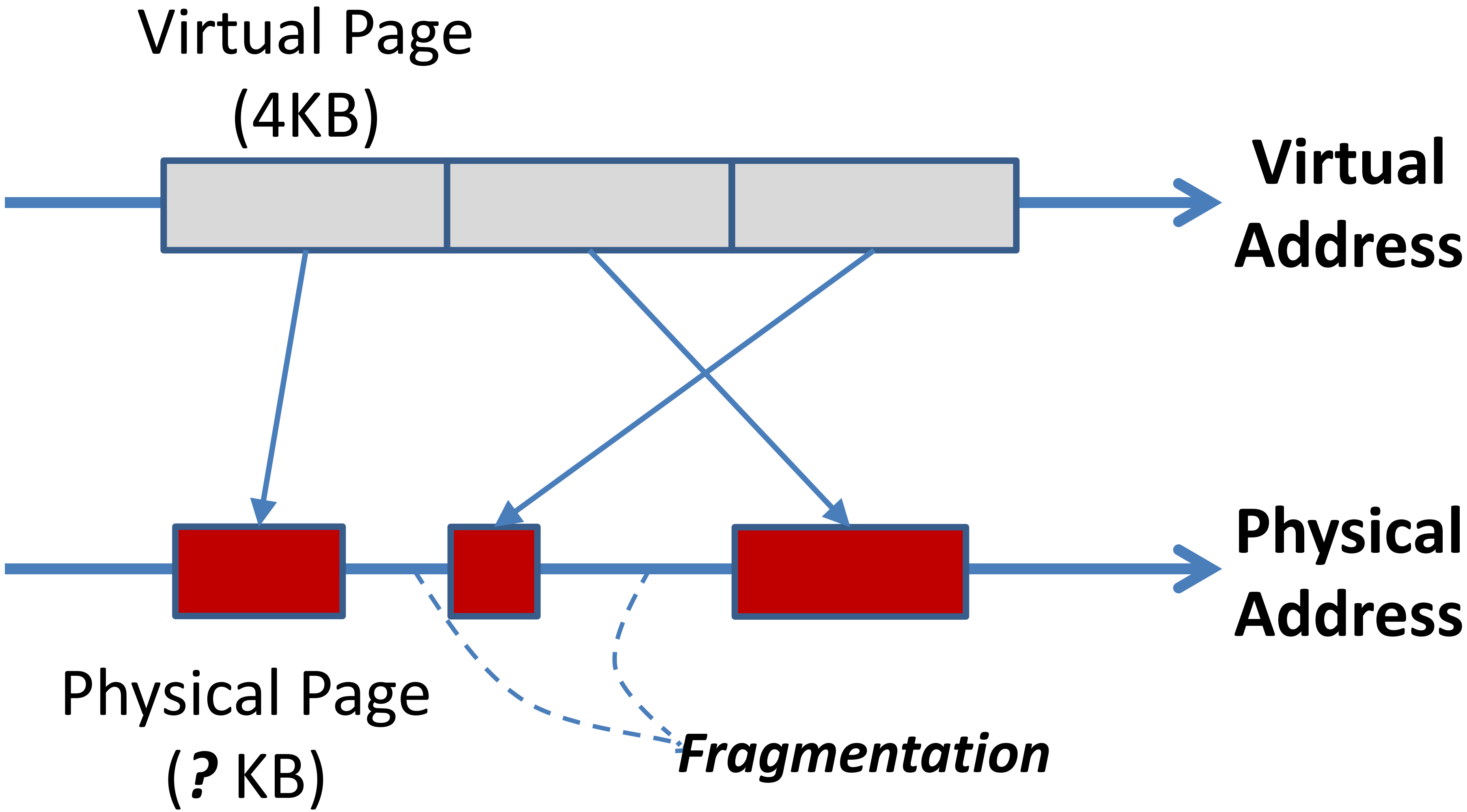
1. Address Computation

2. Mapping and Fragmentation

Address Computation



Mapping and Fragmentation



Shortcomings of Prior Work

Compression Mechanisms	**Compression Ratio**	**Address Comp. Latency**	**Decompression Latency**	**Complexity and Cost**
IBM MXT *[IBM J.R.D. '01]*	✓	✘	✘ **64 cycles**	✘

Shortcomings of Prior Work

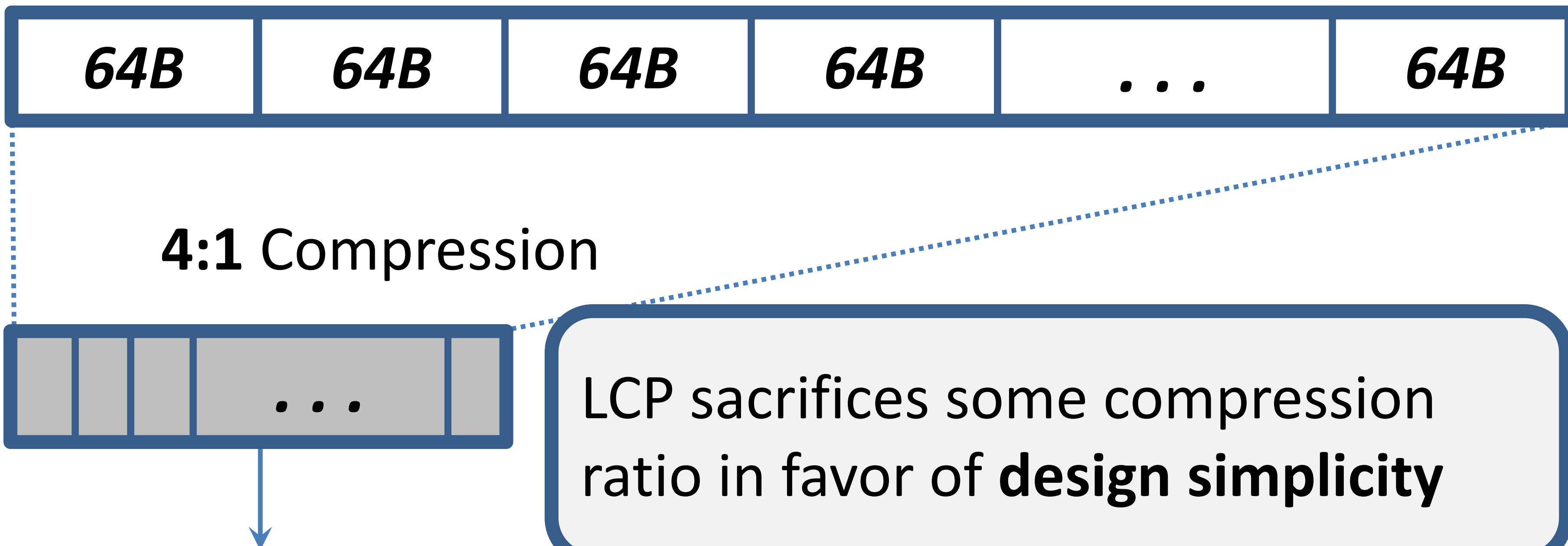
Compression Mechanisms	Compression Ratio	Address Comp. Latency	Decompression Latency	Complexity and Cost
IBM MXT *[IBM J.R.D. '01]*	✓	✗	✗	✗
Robust Main Memory Compression *[ISCA'05]*	✓	✗	✓ **5 cycles**	✗

Shortcomings of Prior Work

Compression Mechanisms	Compression Ratio	Address Comp. Latency	Decompression Latency	Complexity and Cost
IBM MXT *[IBM J.R.D. ’01]*	✔	✘	✘	✘
Robust Main Memory Compression *[ISCA’05]*	✔	✘	✔	✘
Linearly Compressed Pages: Our Proposal	✔	✔	✔	✔

Linearly Compressed Pages (LCP): Key Idea

Uncompressed Page (4KB: 64***64B**)



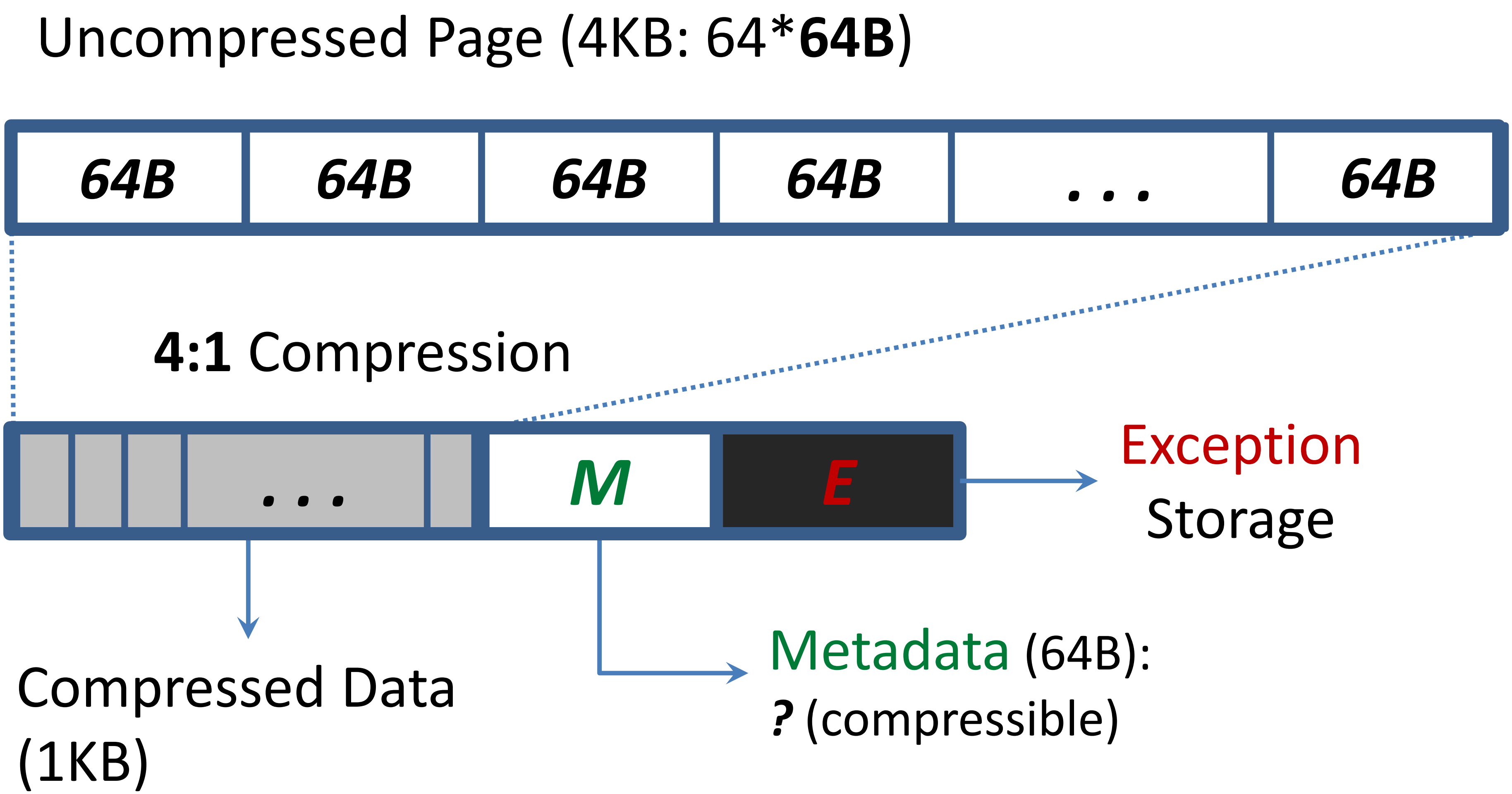
4:1 Compression

Compressed Data (1KB)

LCP sacrifices some compression ratio in favor of **design simplicity**

✓ LCP effectively solves challenge 1: **address computation**

LCP: Key Idea (2)

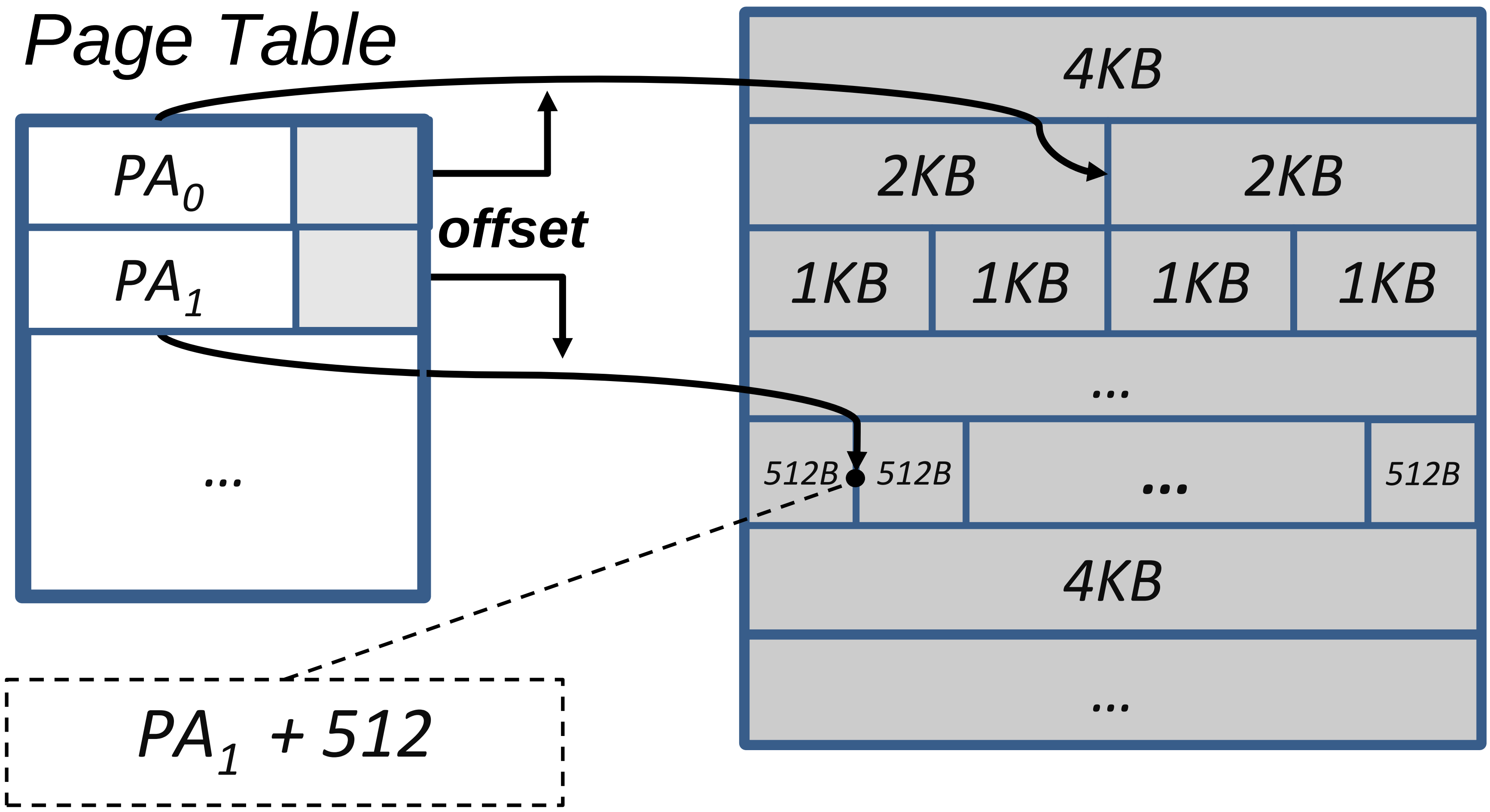


LCP Framework Overview

- Page Table entry extension
  - compression type and size

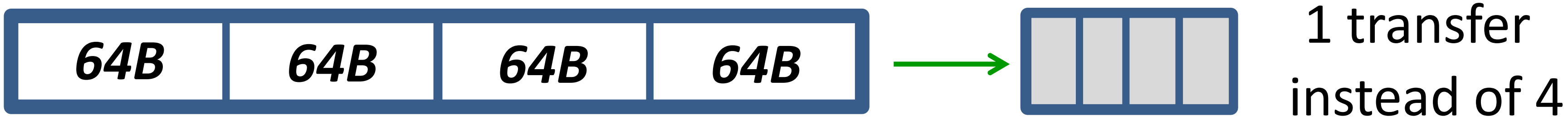
- **OS support** for multiple page sizes
  - **4** memory pools (512B, 1KB, 2KB, 4KB)
- Handling **uncompressible** data
- Hardware support
  - memory controller logic
  - **metadata (MD) cache**

Physical Memory Layout



LCP Optimizations

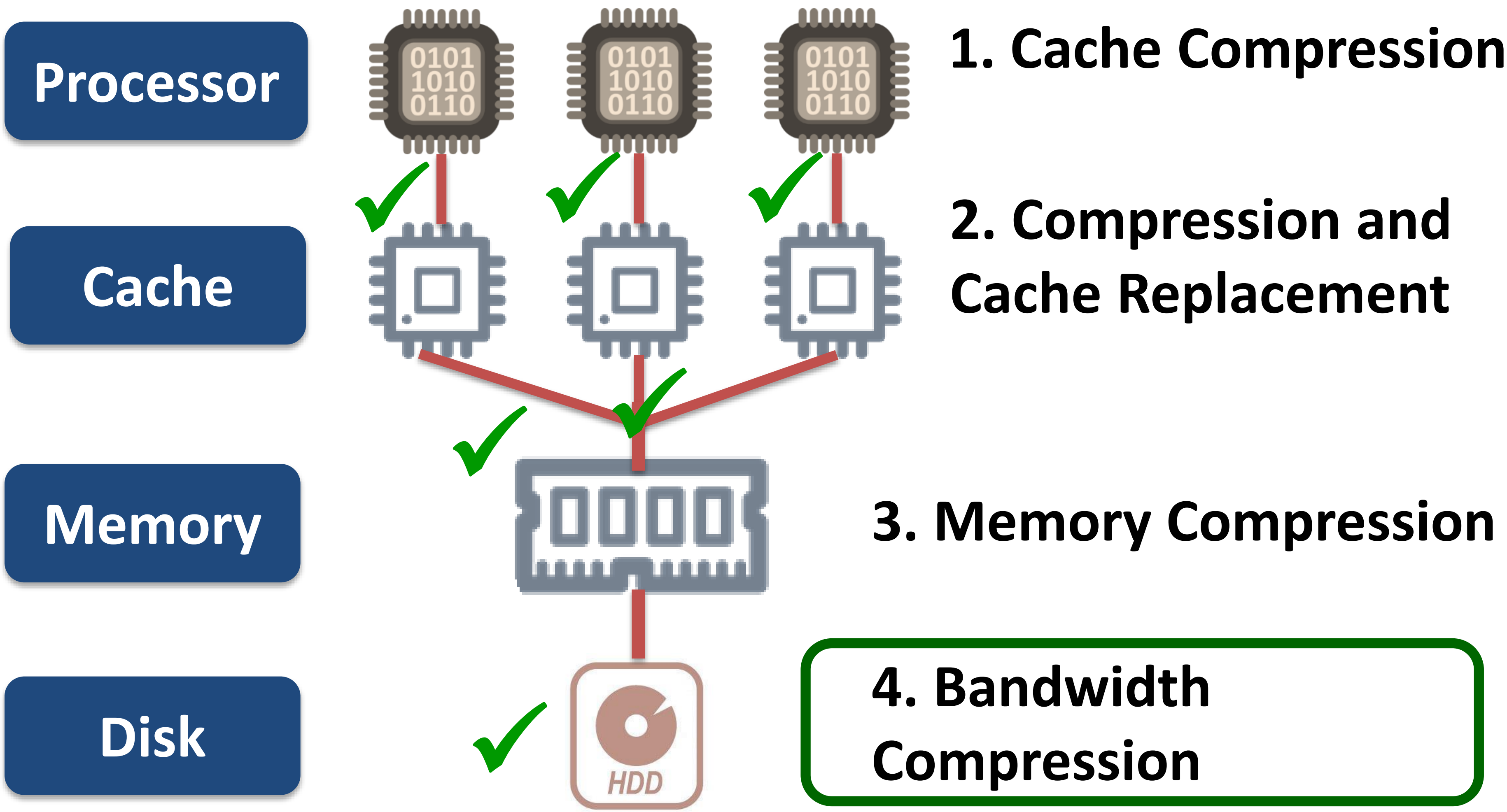
- **Metadata** cache
  - Avoids additional requests to metadata
- Memory bandwidth reduction:



- Zero pages and zero cache lines
  - Handled separately in TLB (1-bit) and in metadata (1-bit per cache line)

Summary of the Results

	Prior Work	vs.	LCP
Comp. Ratio	1.59		1.62
Performance	-4%		+14%
Energy Consumption	↑6%		↓5%


Processor
Cache
Memory
Disk
0101
1010
0110
HDD
1. Cache Compression
2. Compression and Cache Replacement
3. Memory Compression
4. Bandwidth Compression

HPCA 2016

CAL 2015


Linearly Compressed Pages: A Low-Complexity, Low-Latency Main Memory Compression Framework

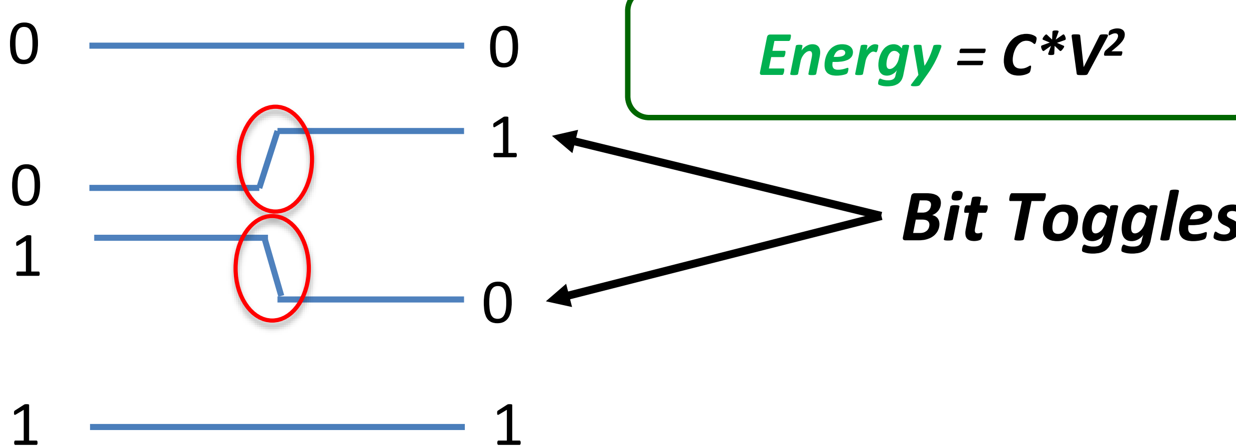
4. Energy-Efficient Bandwidth Compression

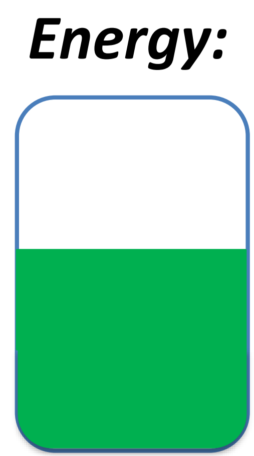
Energy Efficiency: Bit Toggles

How energy is spent in data transfers:

Previous data: 0011 **New data:** 0101

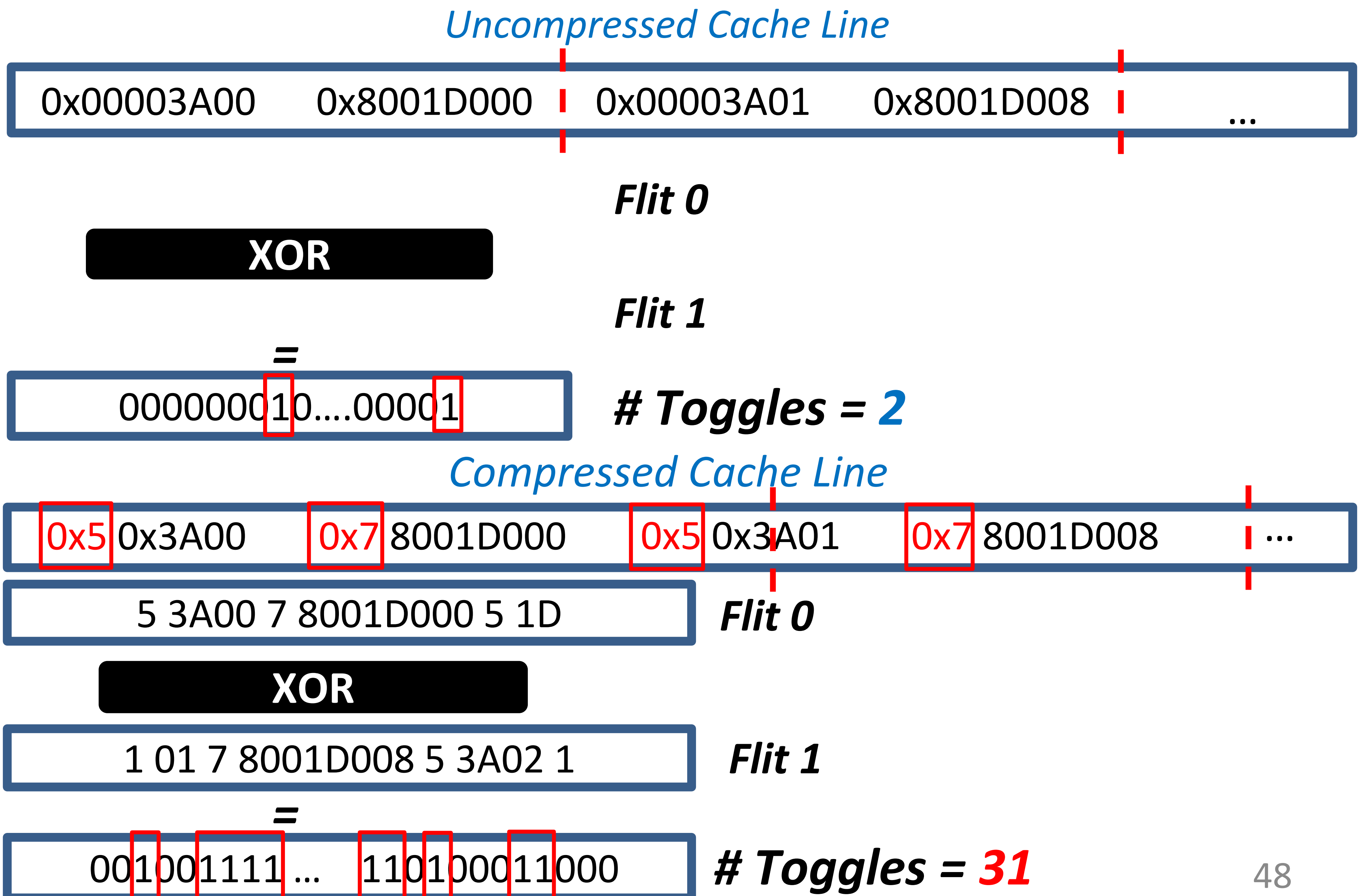
Energy* = *C*V²



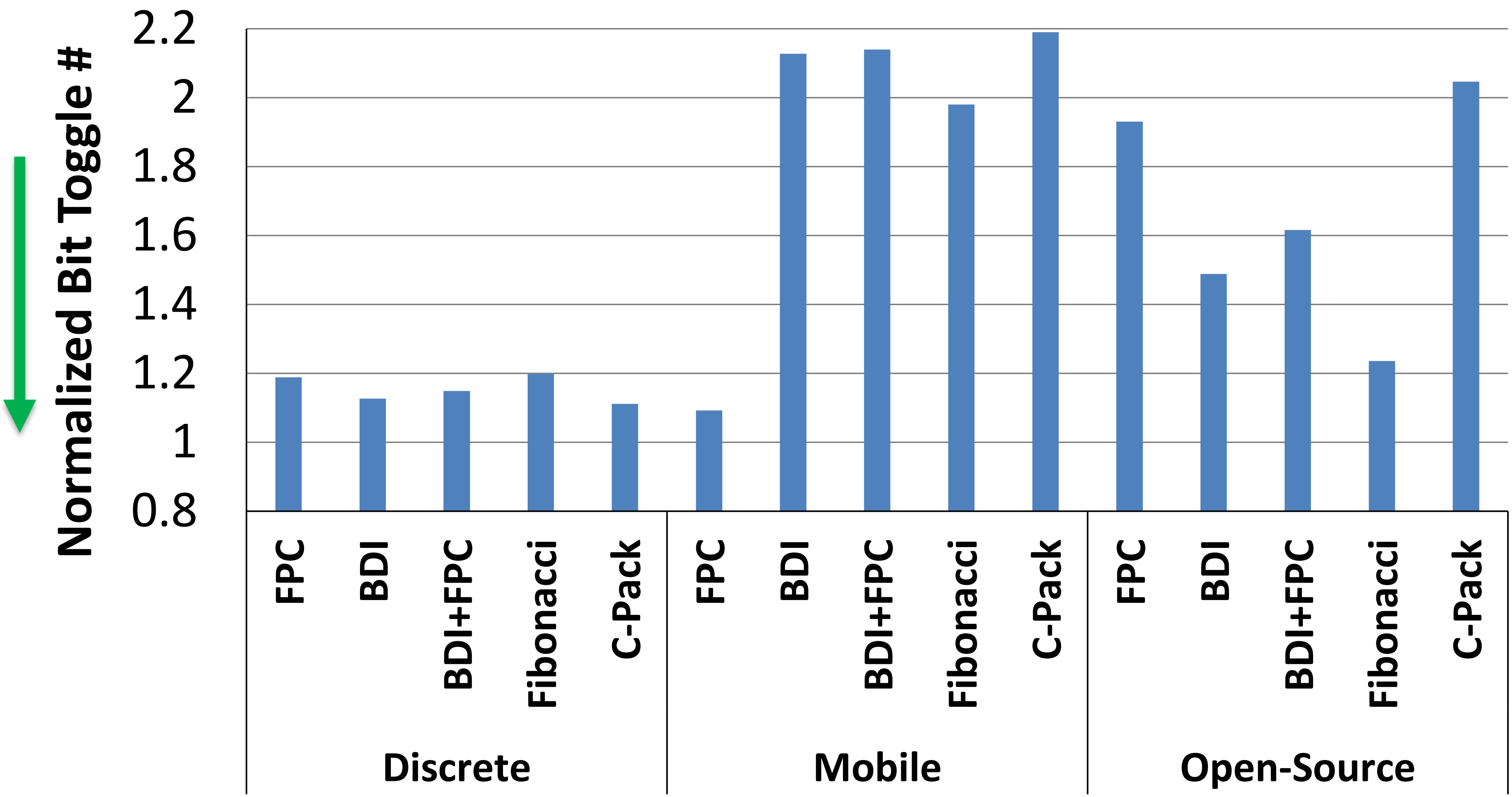


Energy of data transfers (e.g., NoC, DRAM) is proportional to the bit toggle count

Excessive Number of Bit Toggles



Effect of Compression on Bit Toggles



Compression significantly increases bit toggle count

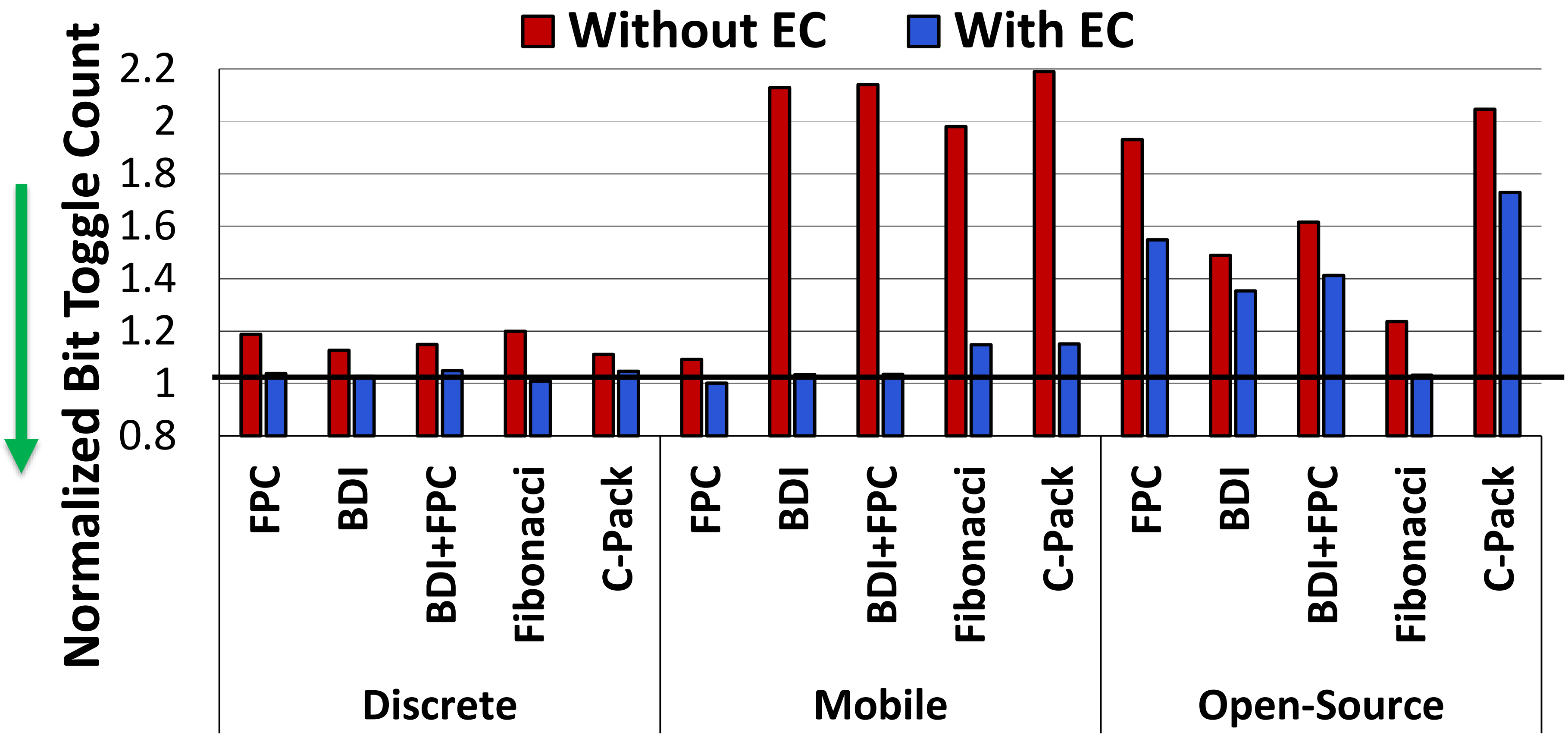
Energy Control

- ***Bit toggle count***: compressed vs. uncompressed
- Use a heuristic (*Energy X Delay* *or* *Energy X Delay*^{2} metric) to estimate the trade-off
- Take ***bandwidth utilization*** into account
- Throttle compression when it is **not** beneficial

Methodology

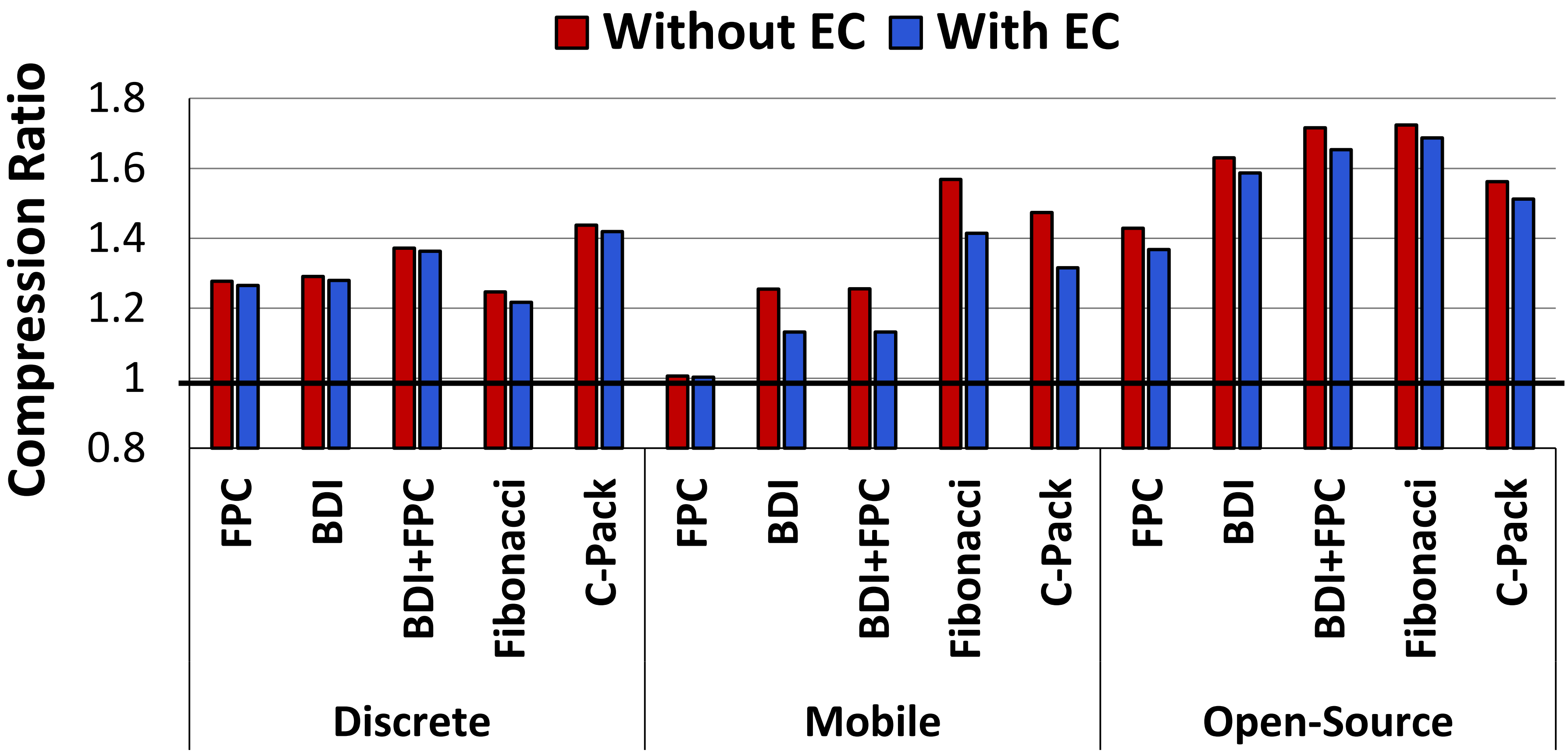
- **Simulator**: GPGPU-Sim 3.2.x and in-house simulator
- **Workloads**:
  - **NVIDIA** apps (discrete and mobile): **221 apps**
  - Open-source (Lonestar, Rodinia, MapReduce): **21 apps**
- **System parameters (Fermi)**:
  - 15 SMs, 32 threads/warp
  - 48 warps/SM, 32768 registers, 32KB Shared Memory
  - Core: 1.4GHz, GTO scheduler, 2 schedulers/SM
  - Memory: 177.4GB/s BW, GDDR5
  - Cache: L1 - 16KB; L2 - 768KB

Effect of EC on Bit Toggle Count



- ✓ EC significantly reduces the bit toggle count
- ✓ Works for different compression algorithms

Effect of EC on Compression Ratio



EC preserves most of the benefits of compression

Acknowledgments

- Todd Mowry and Onur Mutlu
- Phil Gibbons and Mike Kozuch
- Kayvon Fatahalian, David Wood, and Doug Burger
- SAFARI and LBA group members
- Collaborators at CMU, MSR, NVIDIA and GaTech
- CALCM and PDL
- Deb Cavlovich
- Family and friends

Conclusion

- Data stored in memory hierarchies has significant redundancy
  - Inefficient usage of existing limited resources
- Simple and efficient mechanisms for hardware-based data compression
  - On-chip caches
  - Main memory
  - On-chip/off-chip interconnects
- Our mechanisms improve performance, cost and energy efficiency

Practical Data Compression for Modern Memory Hierarchies

Thesis Oral

Gennady Pekhimenko

Committee:
Todd Mowry (Co-chair)
Onur Mutlu (Co-chair)
Kayvon Fatahalian
David Wood, University of Wisconsin-Madison
Doug Burger, Microsoft
Michael Kozuch, Intel

Presented in partial fulfillment of the requirements for the degree of Doctor of Philosophy