# Practical Data Compression for Memory Hierarchies and Applications

**Gennady Pekhimenko**

**Assistant Professor**

**Computer Systems and Networking Group (CSNG)**

**EcoSystem Group**

UNIVERSITY OF
TORONTO

# Performance and Energy Efficiency

*Energy efficiency*

Applications today are *data-intensive*

MEMCACHED

**Memory Caching**

amazon DynamoDB

**Databases**

**Graphics**

# Computation vs. Communication

**Modern memory systems are *bandwidth constrained***

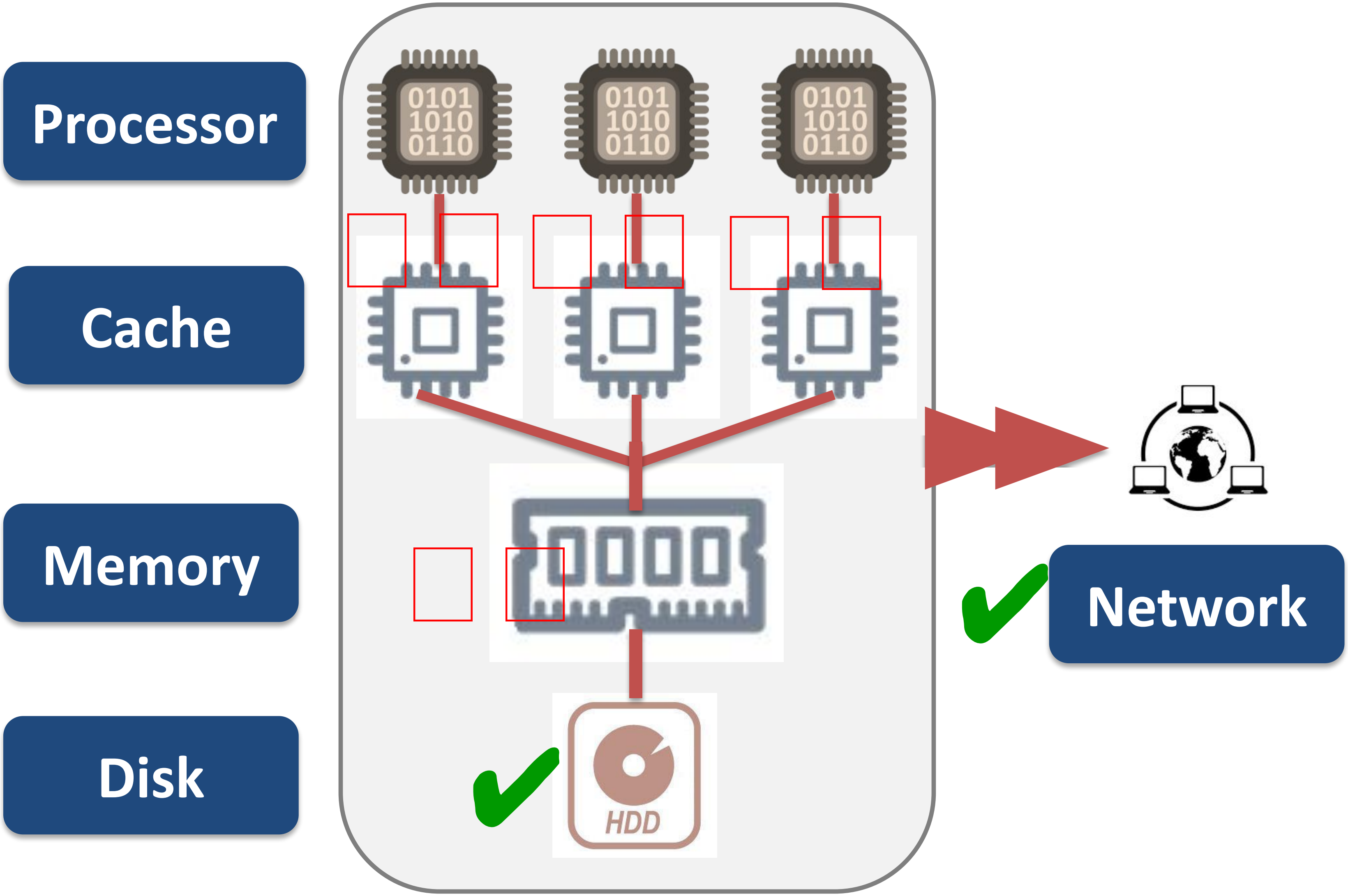
**Data movement is very costly**

- 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

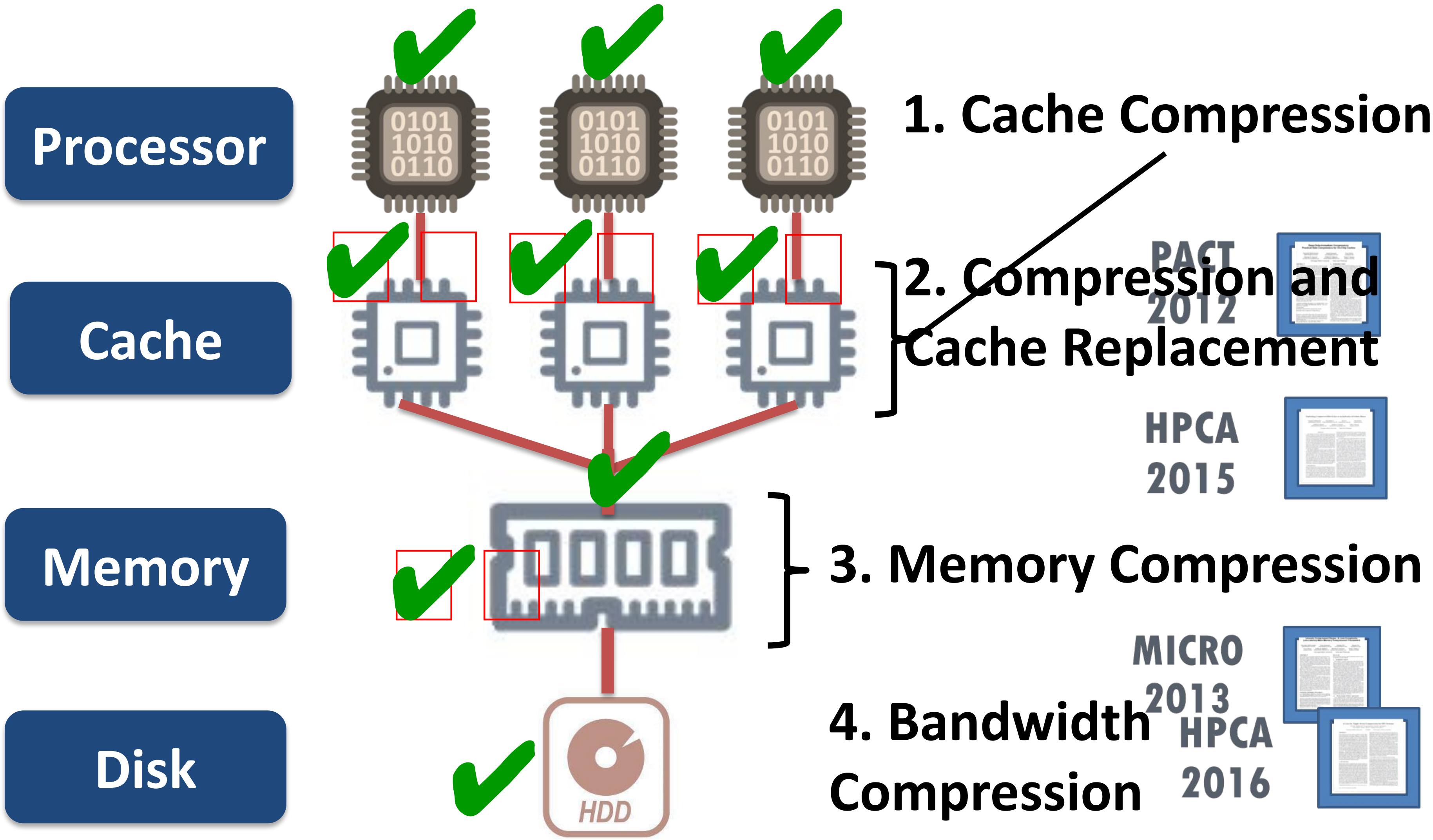
| | **Software** | **vs.** **Hardware** |
|---|---|---|
| **Layer** | Disk | Cache/Memory |
| **Latency** | **milliseconds** | **nanoseconds** |
| **Algorithms** | Dictionary-based | Arithmetic |

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

# Practical Data Compression in Memory



**Processor**

**Cache**

**Memory**

**Disk**

1. Cache Compression

PACT
2012

2. Compression and
Cache Replacement

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

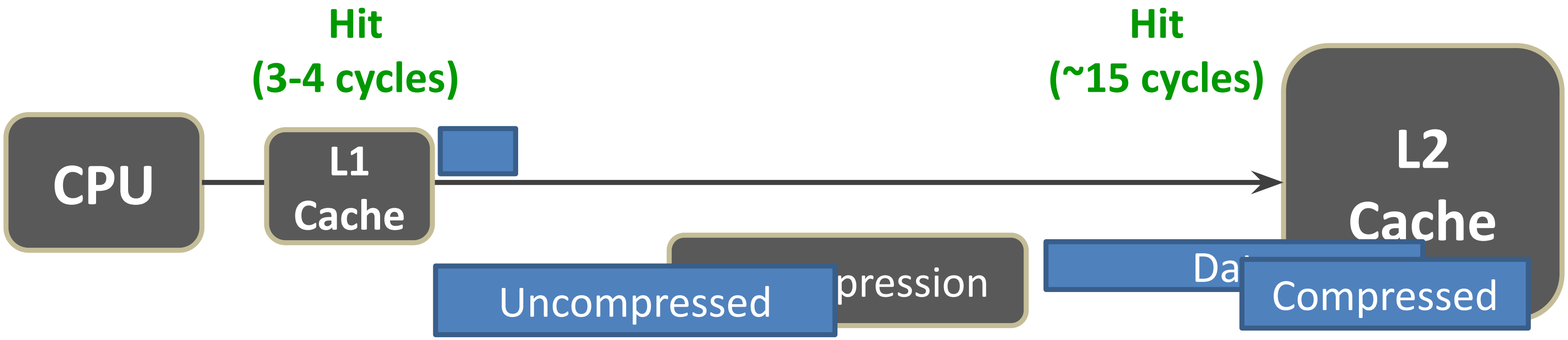
Gennady Pekhimenko† gpekhime@cs.cmu.edu
Vivek Seshadri† vseshadr@cs.cmu.edu
Onur Mutlu† onur@cmu.edu
Michael A. Kozuch‡ michael.a.kozuch@intel.com
Phillip B. Gibbons‡ phillip.b.gibbons@intel.com
Todd C. Mowry† tcm@cs.cmu.edu
†Carnegie Mellon University ‡Intel Labs Pittsburgh

ABSTRACT

1. INTRODUCTION

# 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

| | | | | |
|---|---|---|---|---|
| 0x000000**C0** | 0x000000**C0** | 0x000000**C0** | 0x000000**C0** | … |

**Narrow Values**: small values stored in a big data type

| | | | | |
|---|---|---|---|---|
| 0x000000**C0** | 0x000000**C8** | 0x000000**D0** | 0x000000**D8** | … |

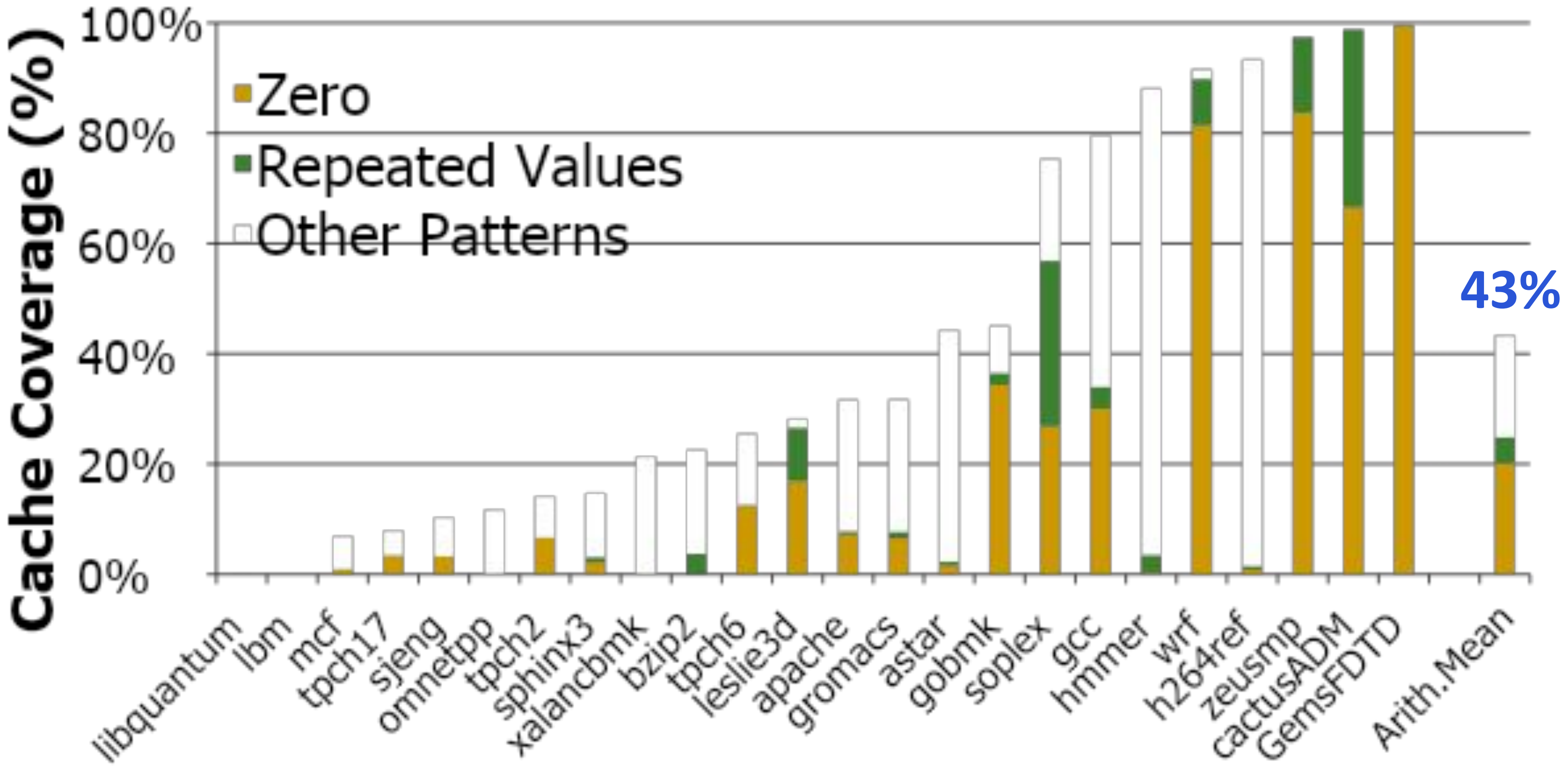
**Other Patterns**: pointers to the same memory region

| | | | | |
|---|---|---|---|---|
| 0xC04039**C0** | 0xC04039**C8** | 0xC04039**D0** | 0xC04039**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** | … |

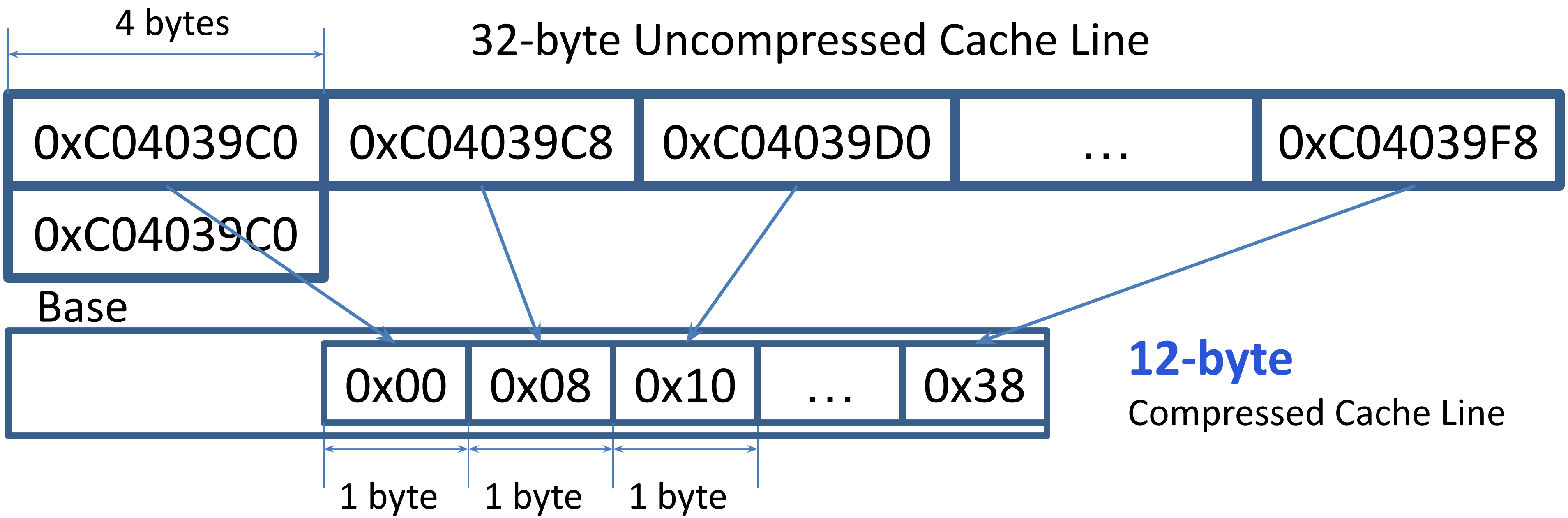
# Key Data Patterns in Real Applications

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



4 bytes

32-byte Uncompressed Cache Line

| 0xC04039C0 | 0xC04039C8 | 0xC04039D0 | … | 0xC04039F8 |
|---|---|---|---|---|

0xC04039C0

Base

| 0x00 | 0x08 | 0x10 | … | 0x38 |
|---|---|---|---|---|

**12-byte**

Compressed Cache Line

1 byte

1 byte

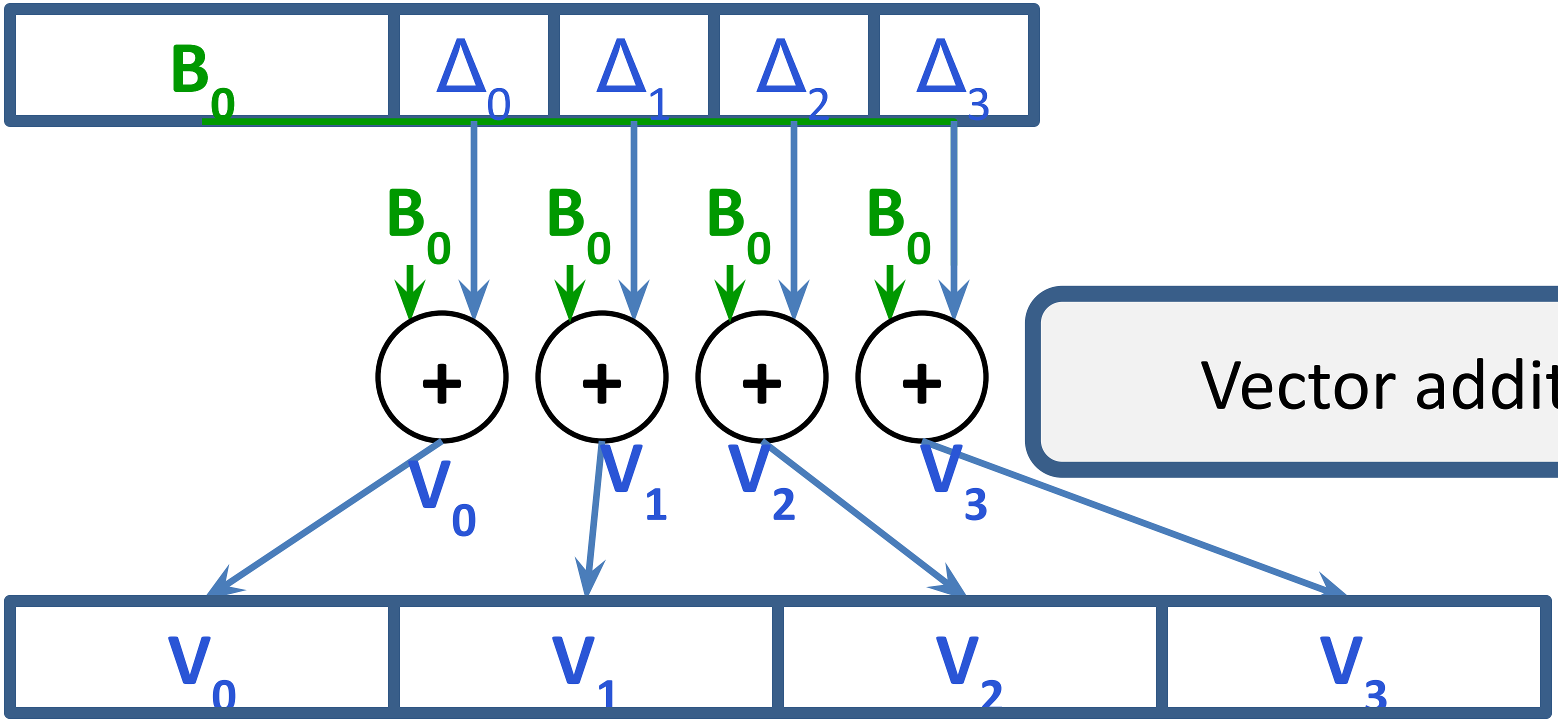
1 byte

**20 bytes saved**

✔ **Effective:** good compression ratio

# B+Δ Decompressor Design

Compressed Cache Line



Vector addition

Uncompressed Cache Line

✔ **Fast Decompression: 1-2 cycles**

# Can We Get Higher Compression Ratio?

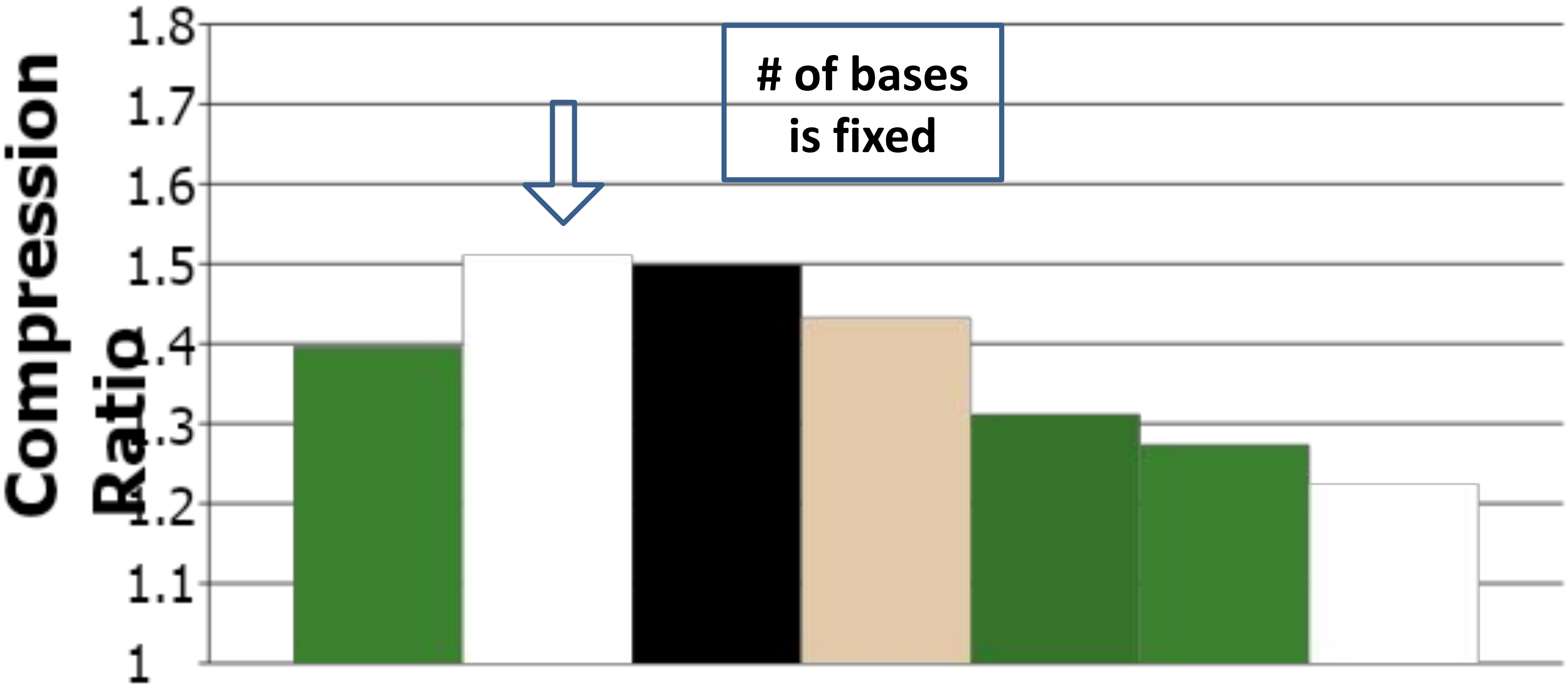
- Uncompressible cache line (with a single base):

| | | | | |
|---|---|---|---|---|
| **0x09A4**0178 | **0x000000**00 | **0x09A4**A838 | **0x000000**0B | … |

```
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**

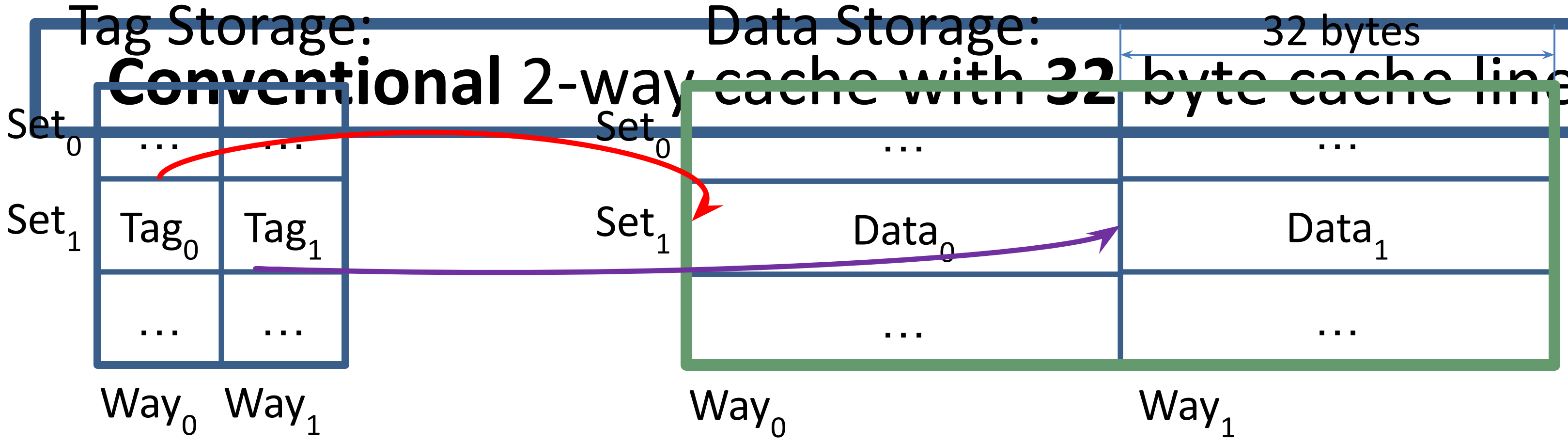
2. **Second base -** implicit base of **0**

   ✓ **Immediate part**

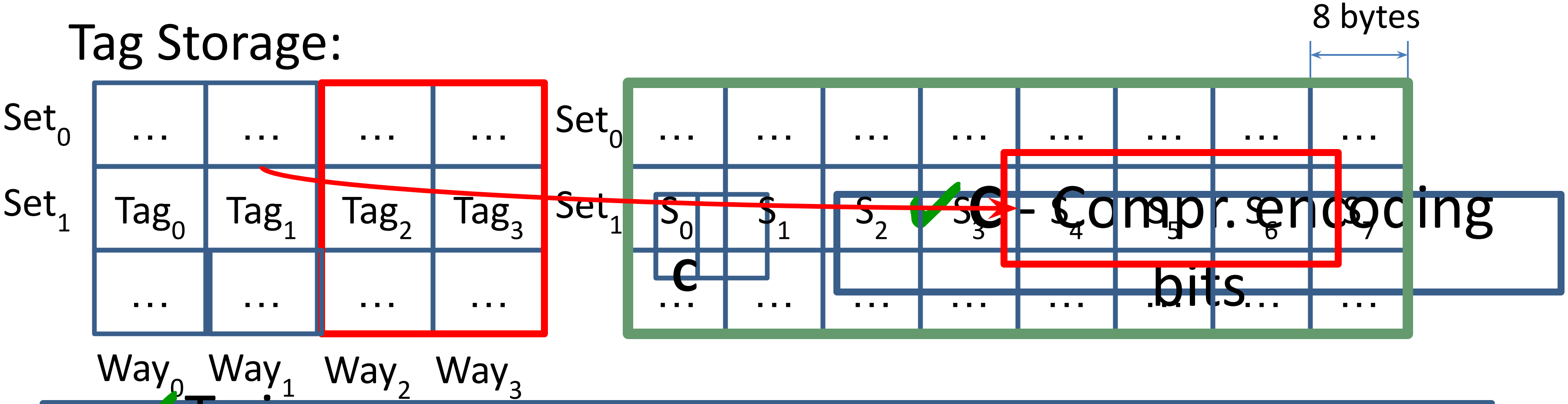
**Base-Delta-Immediate (BΔI) Compression**

# 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

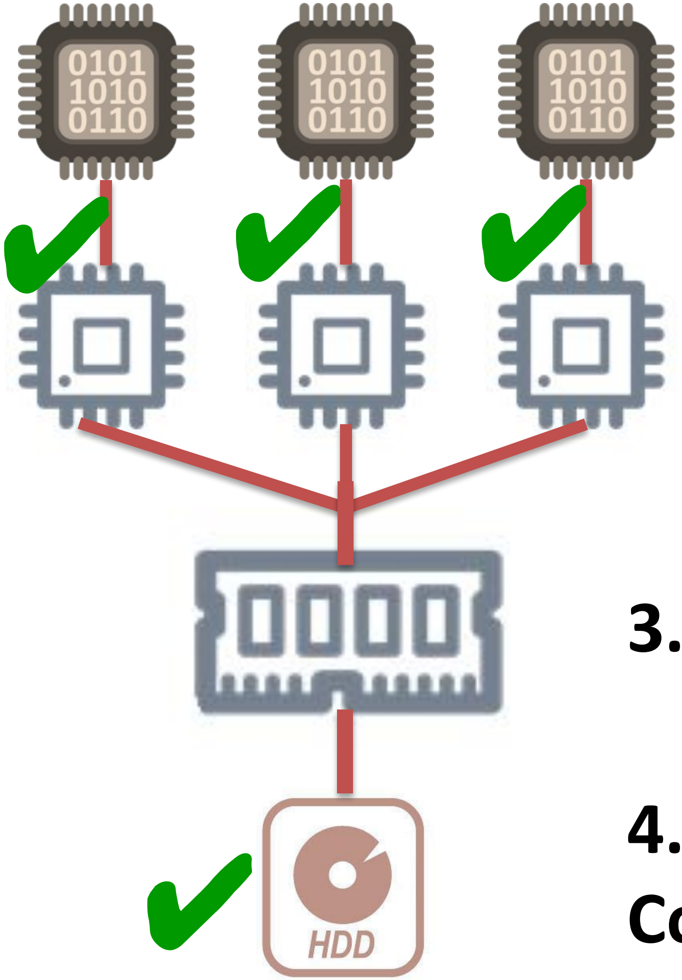
✓Tags map to multiple adjacent segments

2.3% overhead for 2 MB cache

# 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



**Processor**

**Cache**

**Memory**

**Disk**

1. Cache Compression
2. Compression and Cache Replacement
3. Memory Compression
4. Bandwidth Compression

**HPCA**
**2015**

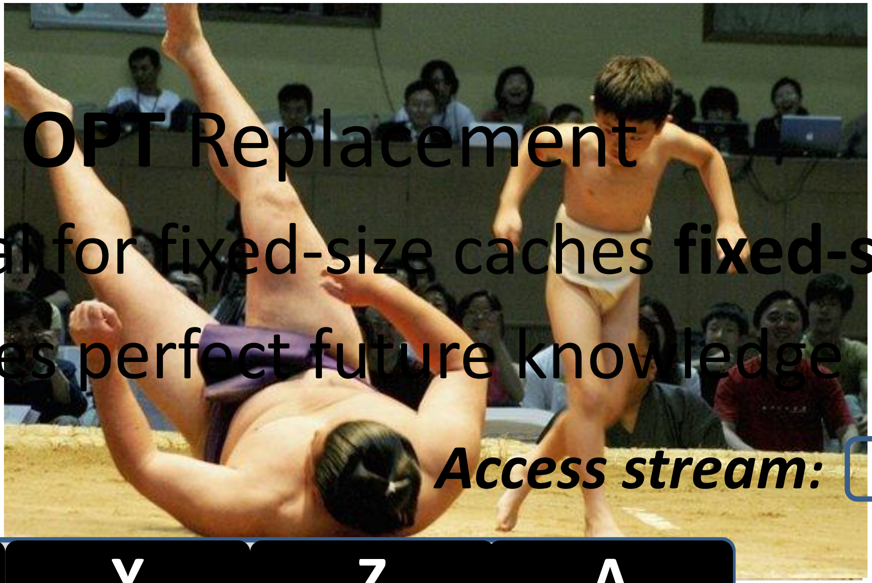
Exploiting Compressed Block Size as an Indicator of Future Reuse

Gennady Pekhimenko Tyler Huberty Rui Cai Onur Mutlu
Phillip B. Gibbons Michael A. Kozuch Todd C. Mowry
Carnegie Mellon University Intel Labs Pittsburgh

# 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

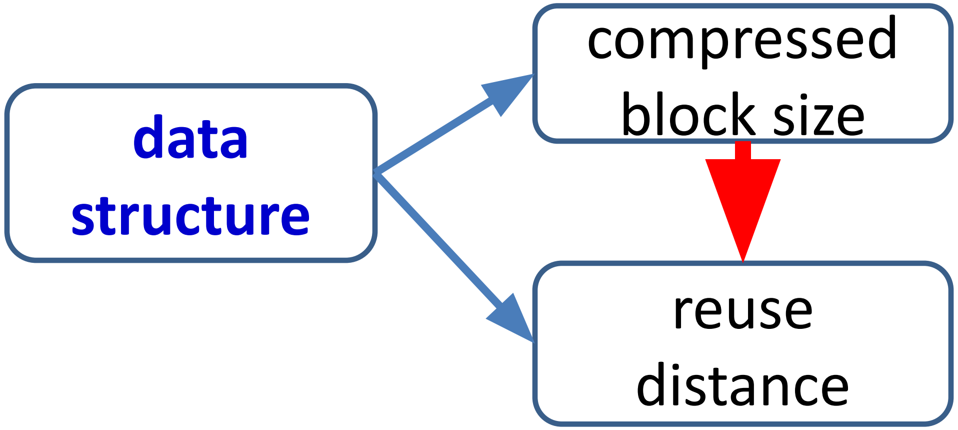
***Cache:***

| X | Y | Z | A |
|---|---|---|---|

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

# Block Size Can Indicate Reuse

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



**data structure**

compressed block size

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

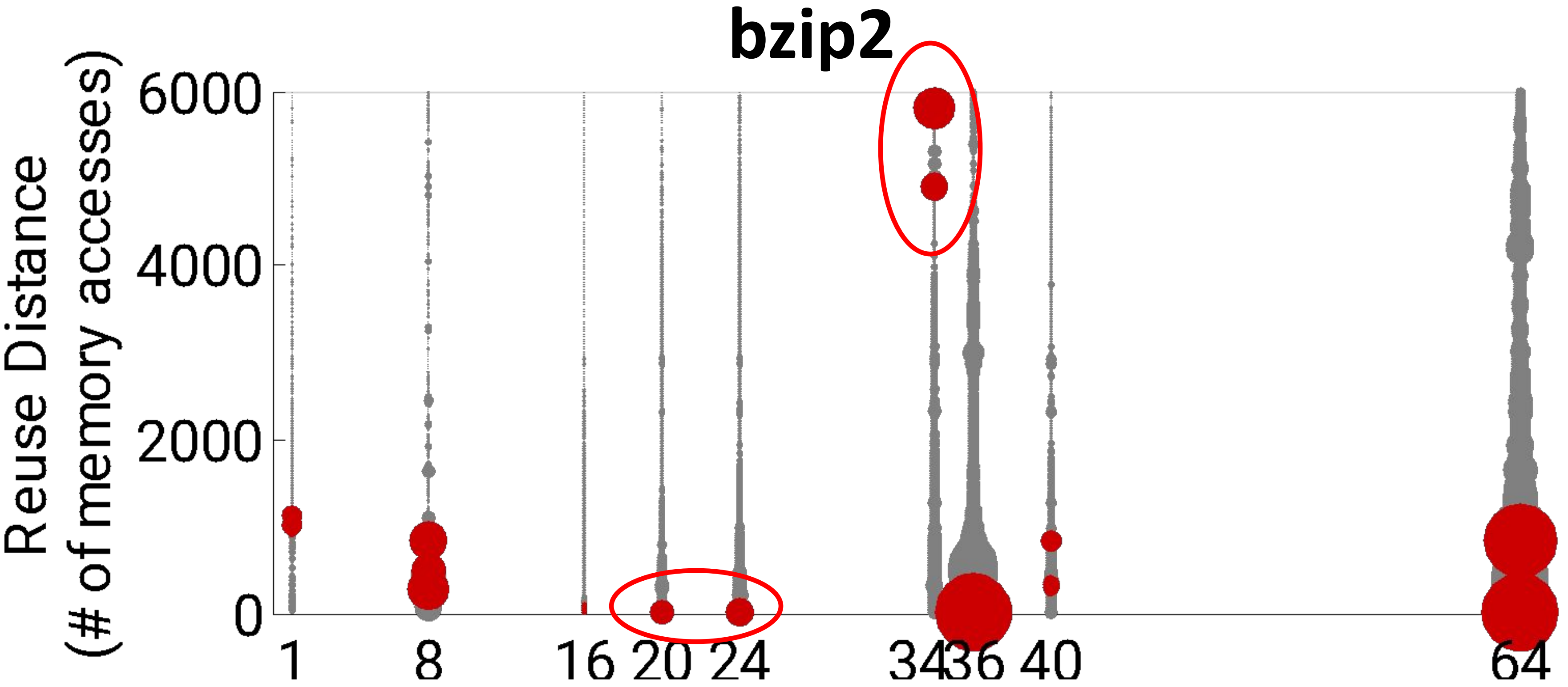
```
int A[N];                // small indices: compressible
double B[16];          // FP coefficients: incompressible
for (int i=0; i<N; i++) {
    int idx = A[i];
    for (int j=0; j<N; j++) {
      sum += B[(idx+j)%16];
    }
}
```

long reuse, compressible

short reuse, incompressible

Compressed size can be an indicator of reuse distance

# Block Size Can Indicate Reuse



Block size, bytes

Different sizes have different dominant reuse distances

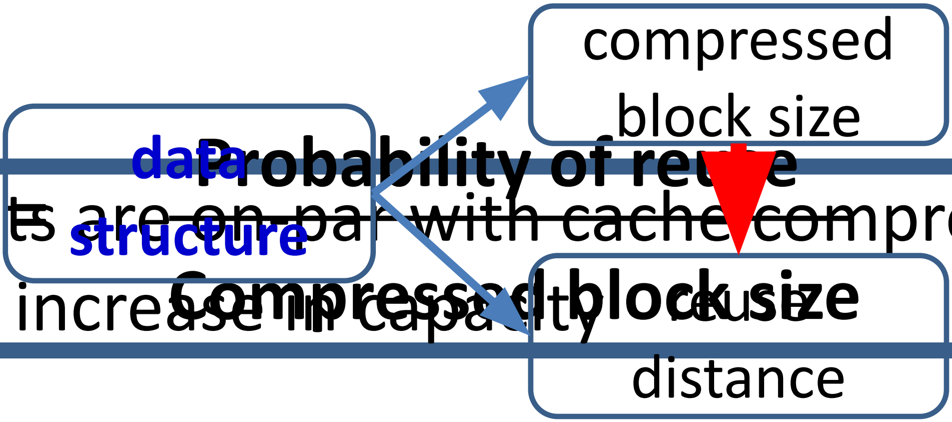
# Compression-Aware Management Policies (CAMP)

**CAMP**

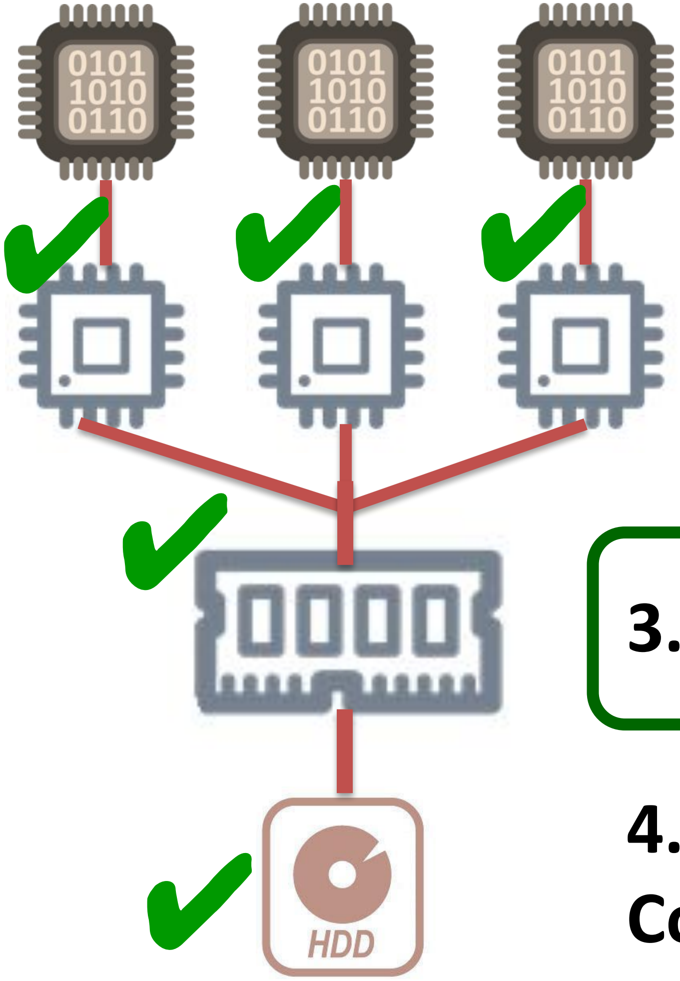
**SIP:**
**Size-based Insertion Policy**

**MVE:**
**Minimal-Value Eviction**

$$\textbf{Value} = \frac{\textbf{Probability of reuse}}{\textbf{Compressed block size}}$$



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



**Processor**

**Cache**

**Memory**

**Disk**

1. Cache Compression
2. Compression and Cache Replacement
3. Memory Compression
4. Bandwidth Compression

MICR
O
2013

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

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

Carnegie Mellon University, Intel Labs Pittsburgh

ABSTRACT

Keywords

1. INTRODUCTION

Categories and Subject Descriptors

1.1 Shortcomings of Prior Approaches

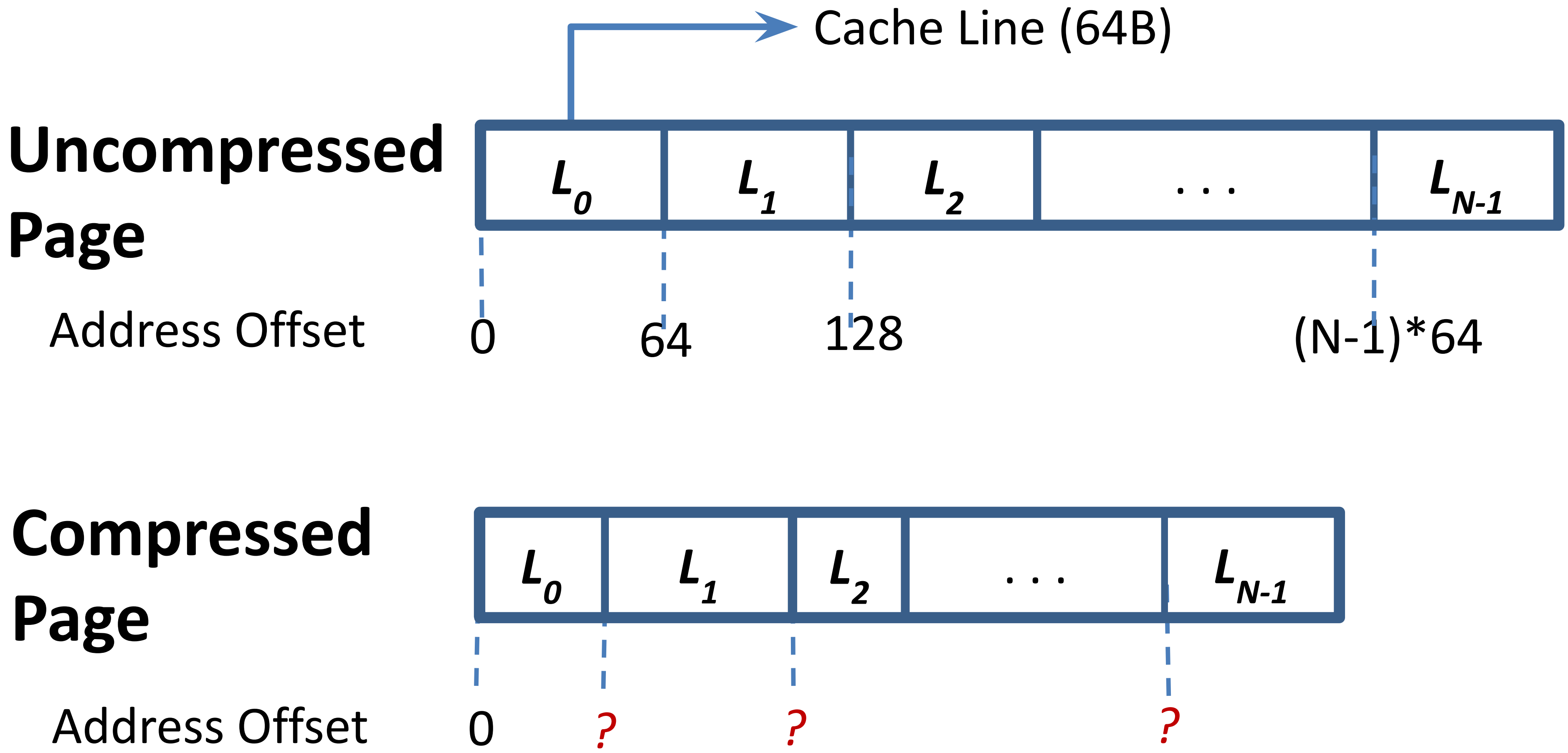
# 3. Main Memory Compression

# Challenges in Main Memory Compression

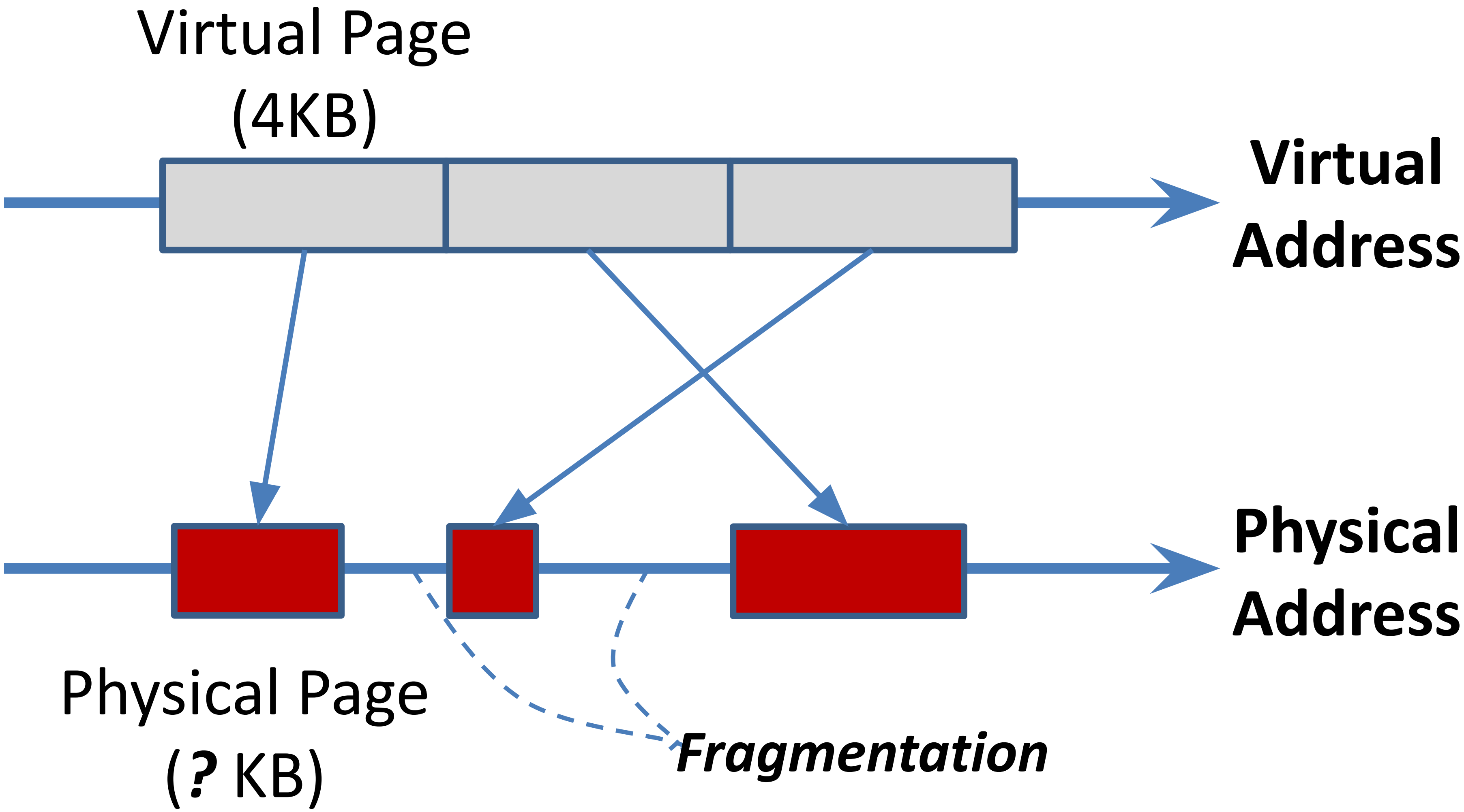
1. Address Computation

2. Mapping and Fragmentation

# Address Computation



# Mapping and Fragmentation



Virtual Page (4KB)

**Virtual Address**

**Physical Address**

Physical Page (***?*** KB)

***Fragmentation***

# Shortcomings of Prior Work

| Compression Mechanisms | Compression Ratio | Address Comp. Latency | Decompression Latency | Complexity and Cost |
|---|---|---|---|---|
| IBM MXT<br>*[IBM J.R.D. '01]* | ✓ | ☐ | **64 cycles**<br>☐ | ☐ |
| | | | | |
| | | | | |

# Shortcomings of Prior Work

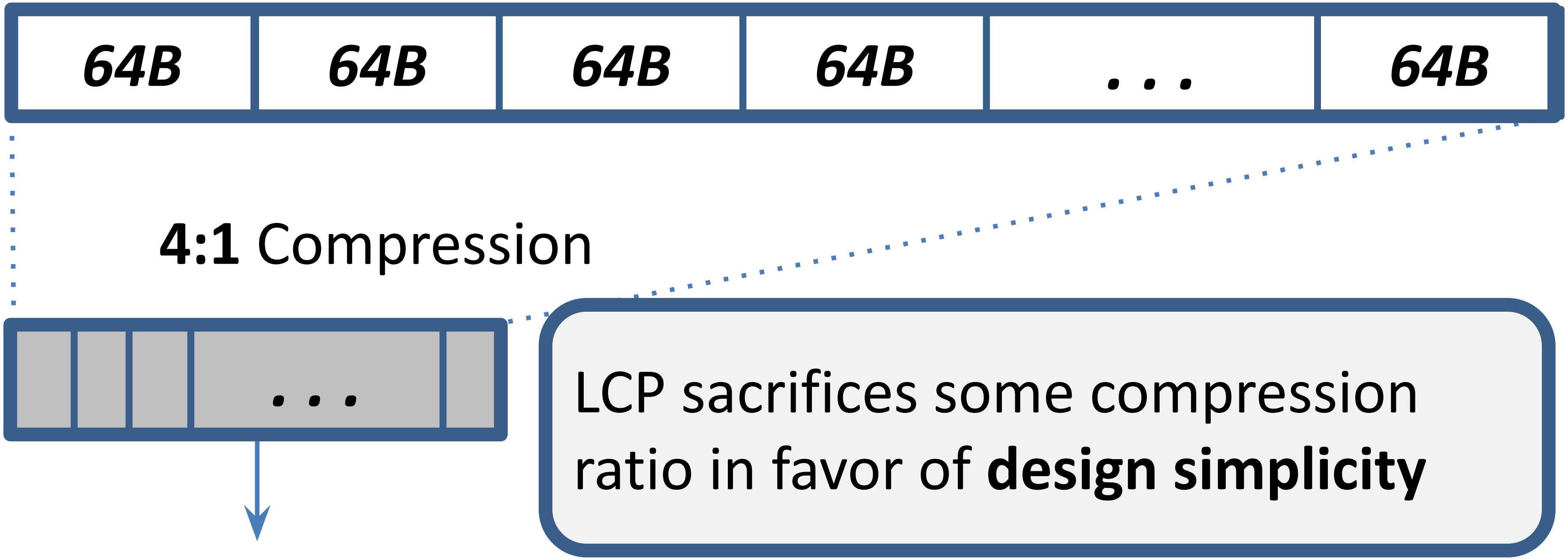
| Compression Mechanisms | Compression Ratio | Address Comp. Latency | Decompression Latency | Complexity and Cost |
|---|---|---|---|---|
| IBM MXT<br>*[IBM J.R.D. '01]* | ✓ | ☐ | ☐ | ☐ |
| Robust Main Memory Compression<br>*[ISCA'05]* | ✓ | ☐ | **5 cycles**<br>✓ | ☐ |
| | | | | |

# Shortcomings of Prior Work

| Compression Mechanisms | Compression Ratio | Address Comp. Latency | Decompression Latency | Complexity and Cost |
|---|---|---|---|---|
| IBM MXT<br>*[IBM J.R.D. '01]* | ✓ | ☐ | ☐ | ☐ |
| Robust Main Memory Compression<br>*[ISCA'05]* | ✓ | ☐ | ✓ | ☐ |
| **Linearly Compressed Pages: Our** | ✓ | ✓ | ✓ | ✓ |

# Linearly Compressed Pages (LCP): Key Idea

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



**4:1** Compression

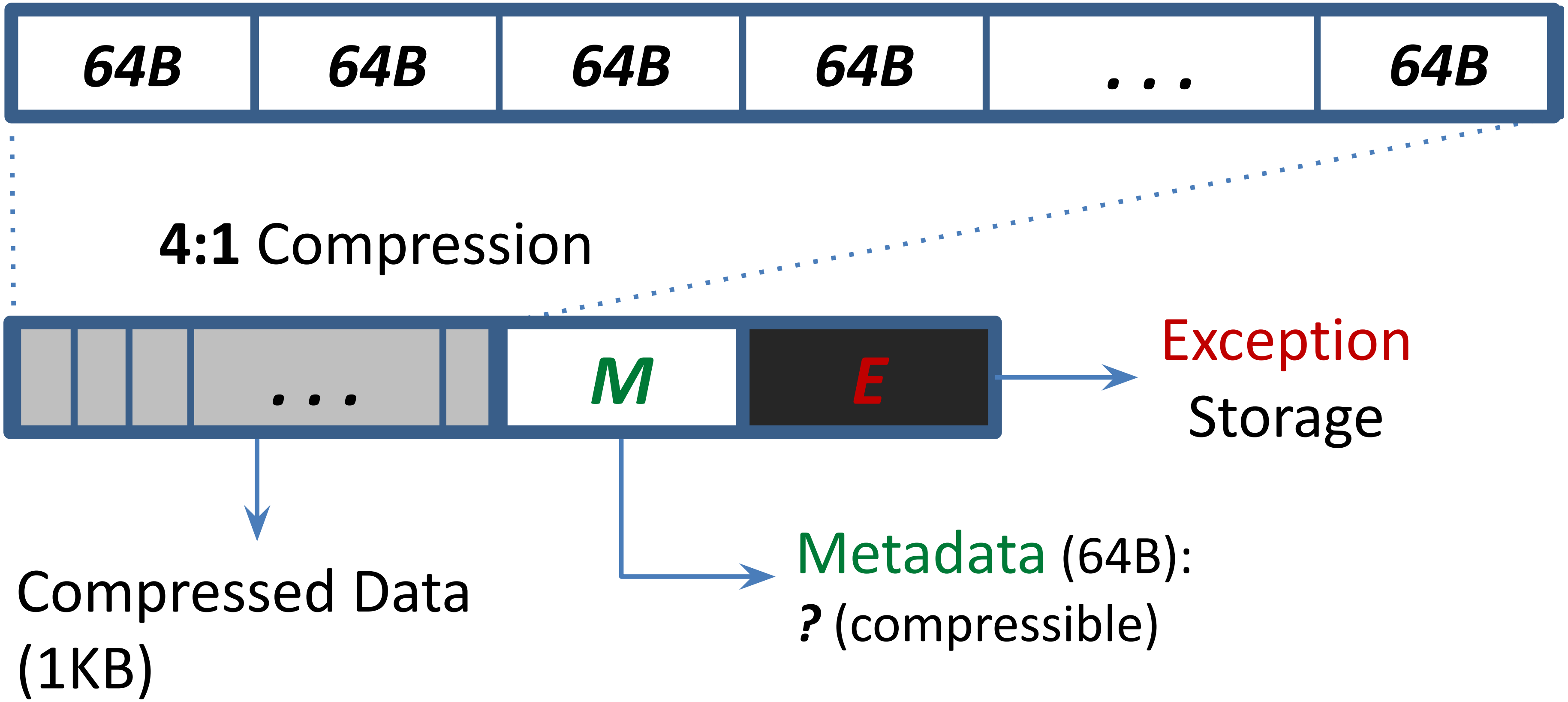
Compressed Data (1KB)

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

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

# LCP: Key Idea (2)

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



**4:1** Compression

Exception Storage

Compressed Data (1KB)

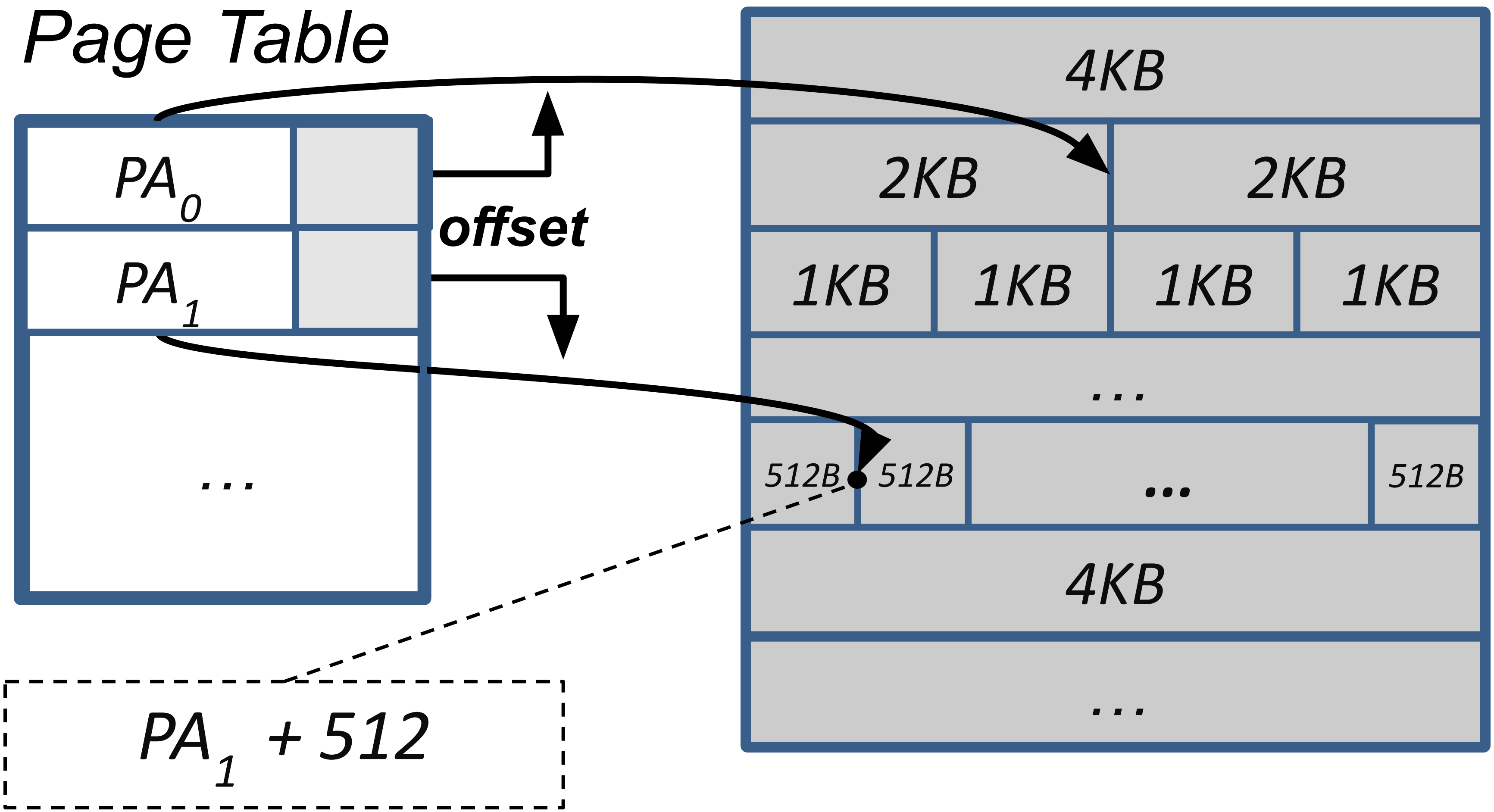
Metadata (64B):
***?*** (compressible)

# LCP Framework Overview

- Page Table entry extension
  - compression type and size

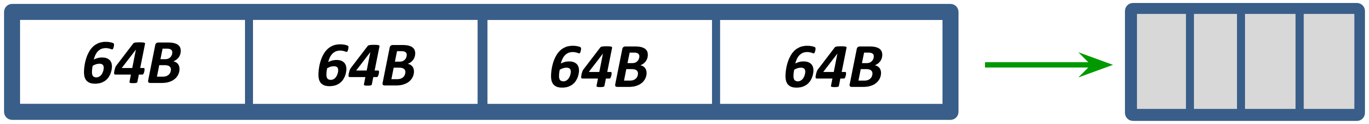
- **OS support** for multiple page sizes
  - **4** memory pools (512B, 1KB, 2KB, 4KB)
- Handling **uncompressible** data
- Hardware support
  - memory controller logic
  - **metadata (MD) cache**

# Physical Memory Layout



# LCP Optimizations

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



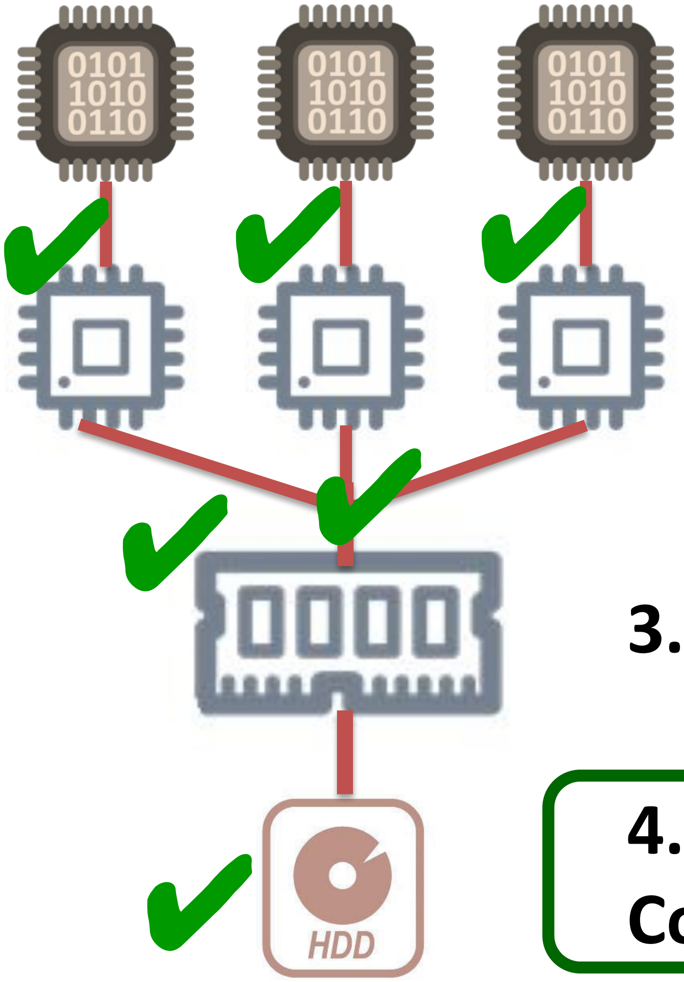
***64B*** ***64B*** ***64B*** ***64B***

1 transfer
instead of 4

- 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**

**1. Cache Compression**

**Cache**

**2. Compression and Cache Replacement**

**Memory**

**3. Memory Compression**

**Disk**

**4. Bandwidth Compression**

HPCA
2016


A Case for Toggle-Aware Compression for GPU Systems

CAL
2015


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

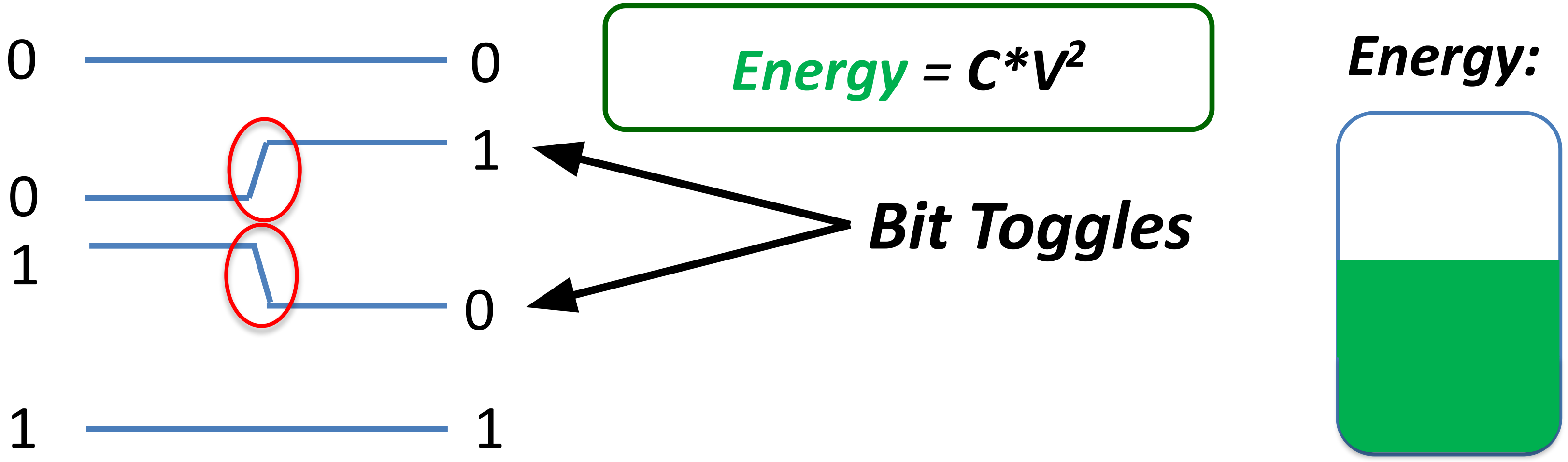
# 4. Energy-Efficient Bandwidth Compression

# Energy Efficiency: Bit Toggles

## How energy is spent in data transfers:

**Previous data:** 0011

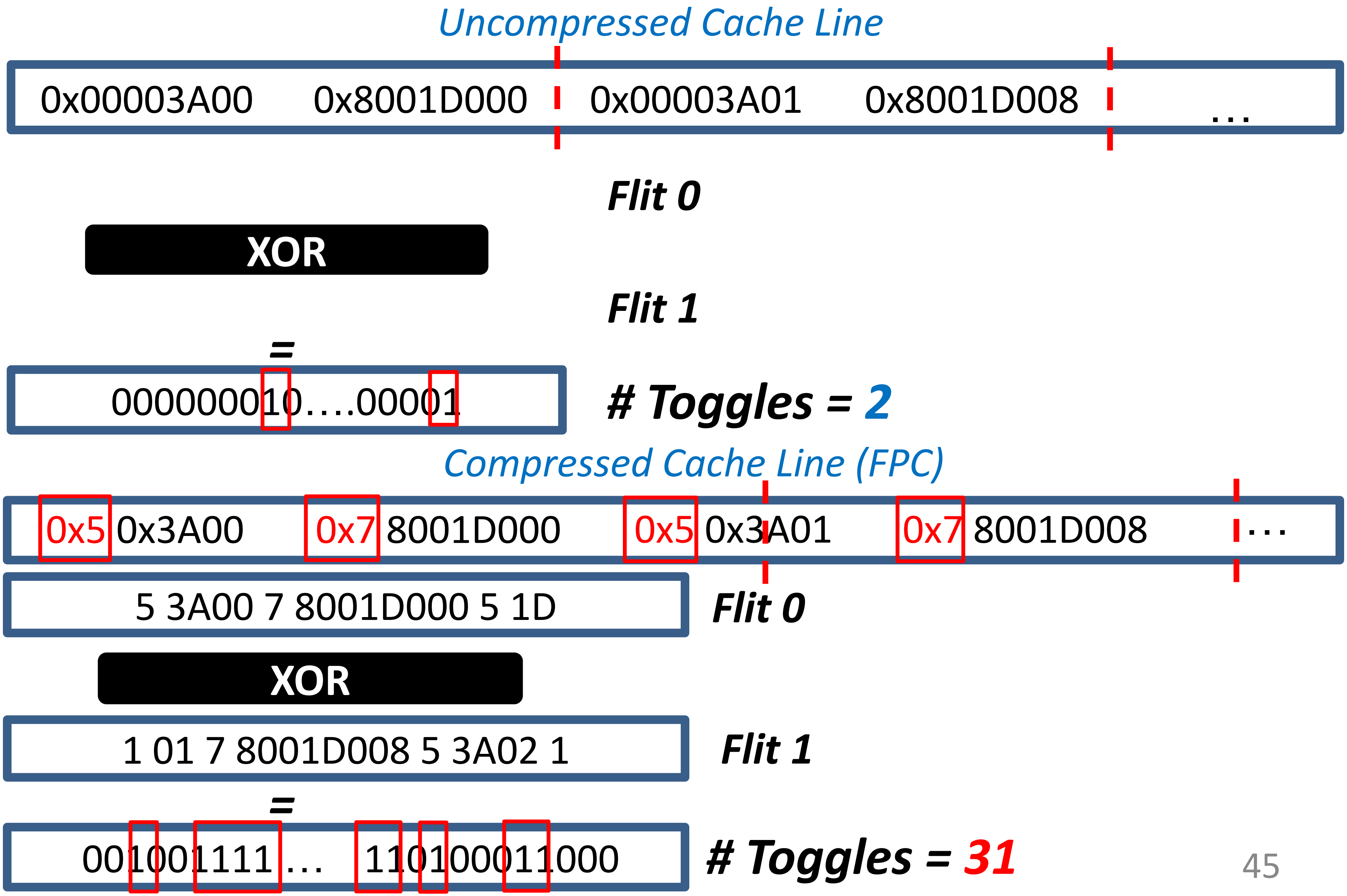
**New data:** 0101



***Energy** = $C*V^2$*

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

# Excessive Number of Bit Toggles



*Uncompressed Cache Line*

0x00003A00 0x8001D000 | 0x00003A01 0x8001D008 | …

***Flit 0***

**XOR**

***Flit 1***

**=**

000000010….00001

***# Toggles = 2***

*Compressed Cache Line (FPC)*

0x5 0x3A00 0x7 8001D000 0x5 0x3|A01 0x7 8001D008 | …

5 3A00 7 8001D000 5 1D

***Flit 0***

**XOR**

1 01 7 8001D008 5 3A02 1

***Flit 1***

**=**

001001111 … 110100011000

***# Toggles = 31***

# Toggle-Aware Data Compression

***Problem:***

- *1.53X* effective compression ratio
- *2.19X* increase in toggle count

***Goal:***

- **Trade-off** between toggle count and compression ratio

***Key Idea:***

- Bit toggle count: compressed vs. uncompressed
- Use a heuristic (*Energy X Delay* or *Energy X Delay*$^2$ metric) to estimate the trade-off
- Throttle compression to reach estimated trade-off

# Practical Data Compression for Memory Hierarchies and Applications

**Gennady Pekhimenko**

**Assistant Professor**

**Computer Systems and Networking Group (CSNG)**

**EcoSystem Group**

UNIVERSITY OF
TORONTO