Exploiting Compressed Block Size as an Indicator of Future Reuse

Gennady Pekhimenko,
Tyler Huberty, Rui Cai,
Onur Mutlu, Todd C. Mowry

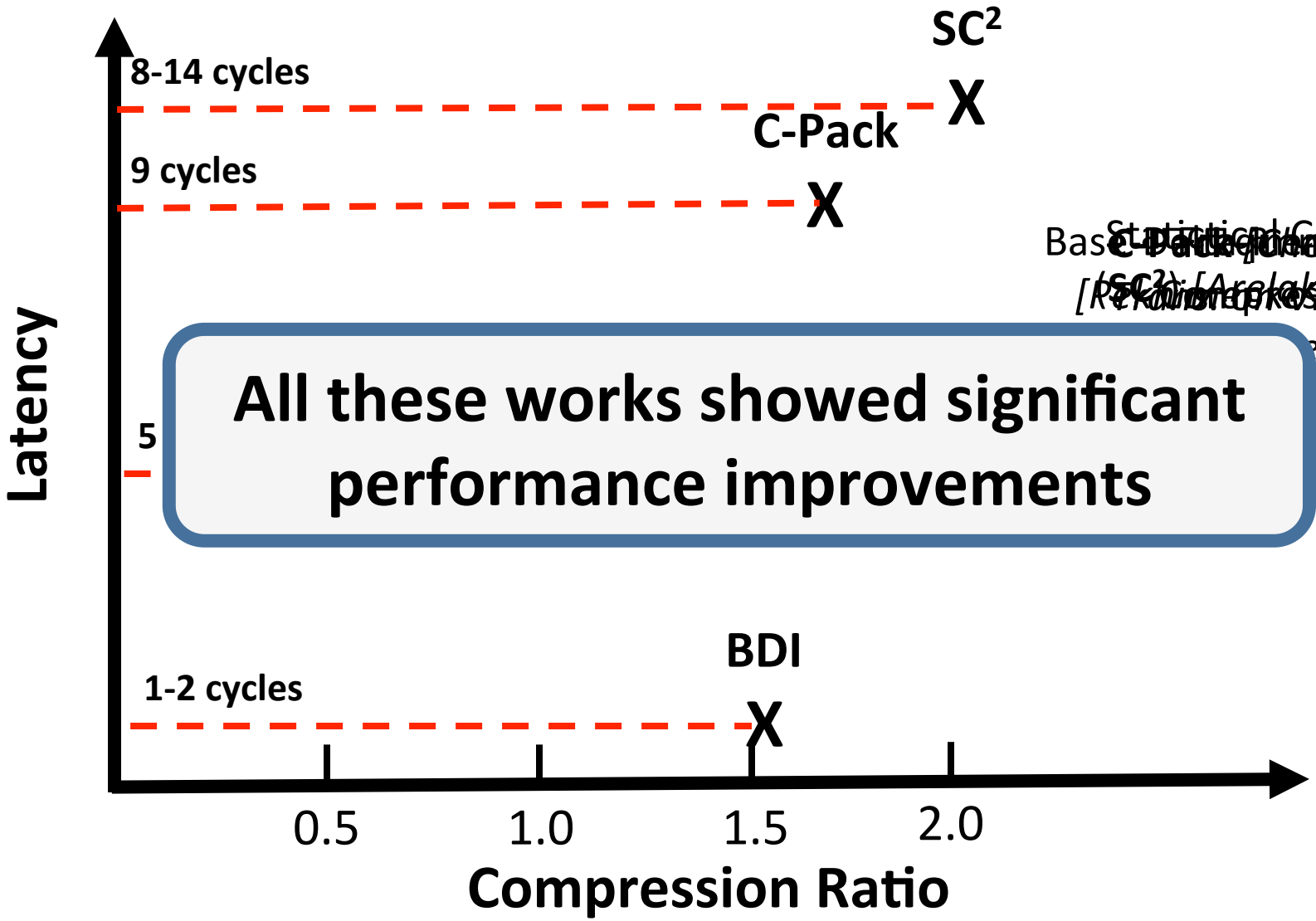
Phillip B. Gibbons,
Michael A. Kozuch

SAFARI Carnegie Mellon University intel

Executive Summary

- In a compressed cache, **compressed block size** is an additional dimension
- **Problem**: How can we maximize cache performance utilizing both block reuse and compressed size?
- **Observations**:
  - Importance of the cache block depends on its compressed size
  - Block size is indicative of reuse behavior in some applications
- **Key Idea**: Use compressed block size in making cache replacement and insertion decisions
- **Results:**
  - Higher performance (**9%/11%** on average for 1/2-core)
  - Lower energy (**12%** on average)

Potential for Data Compression



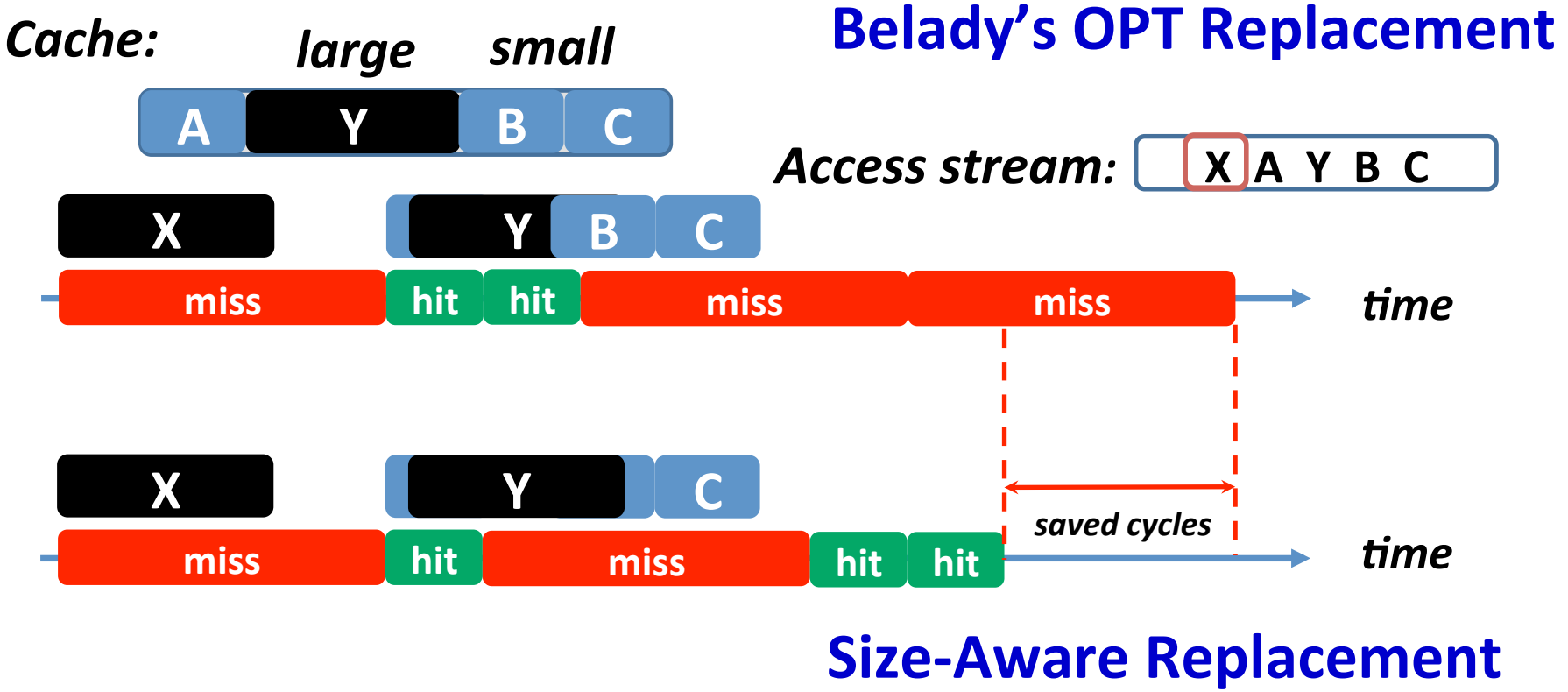
Size-Aware Cache Management

1. Compressed block size matters

2. Compressed block size varies

3. Compressed block size can indicate reuse

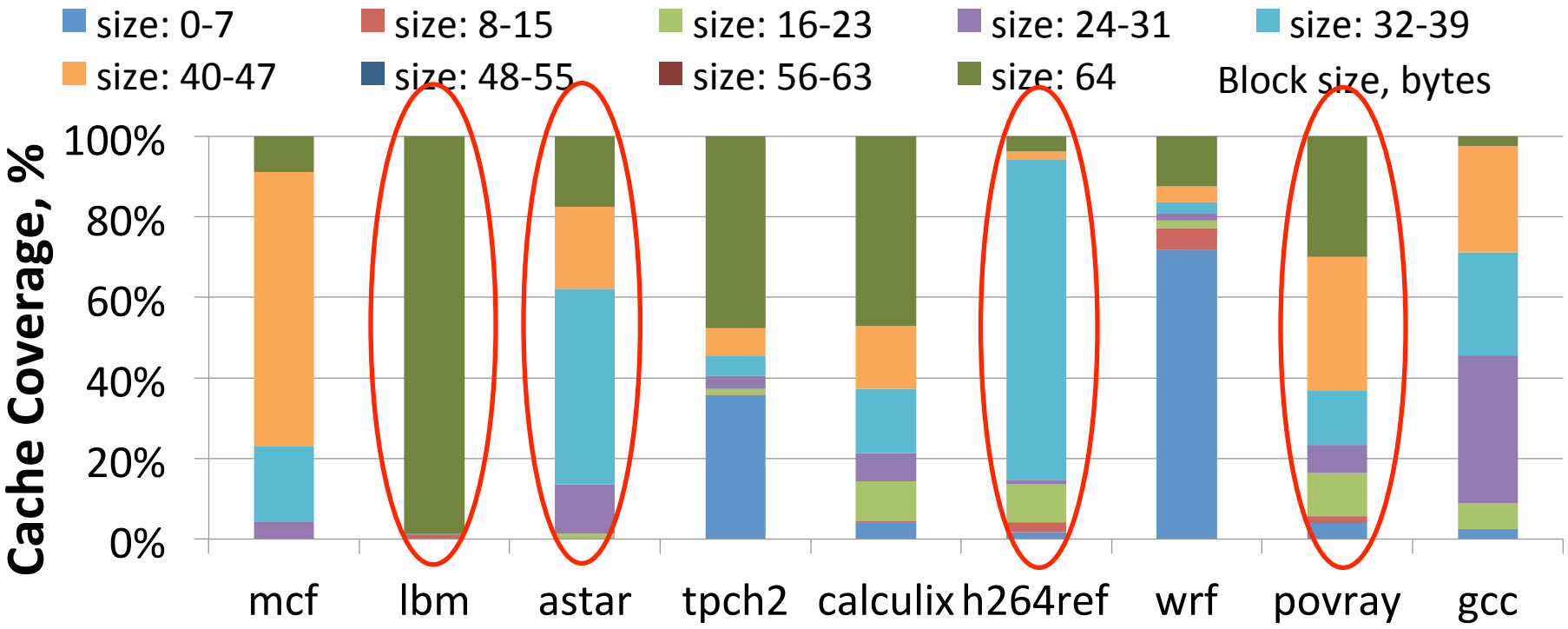
#1: Compressed Block Size Matters



Size-aware policies could yield fewer misses

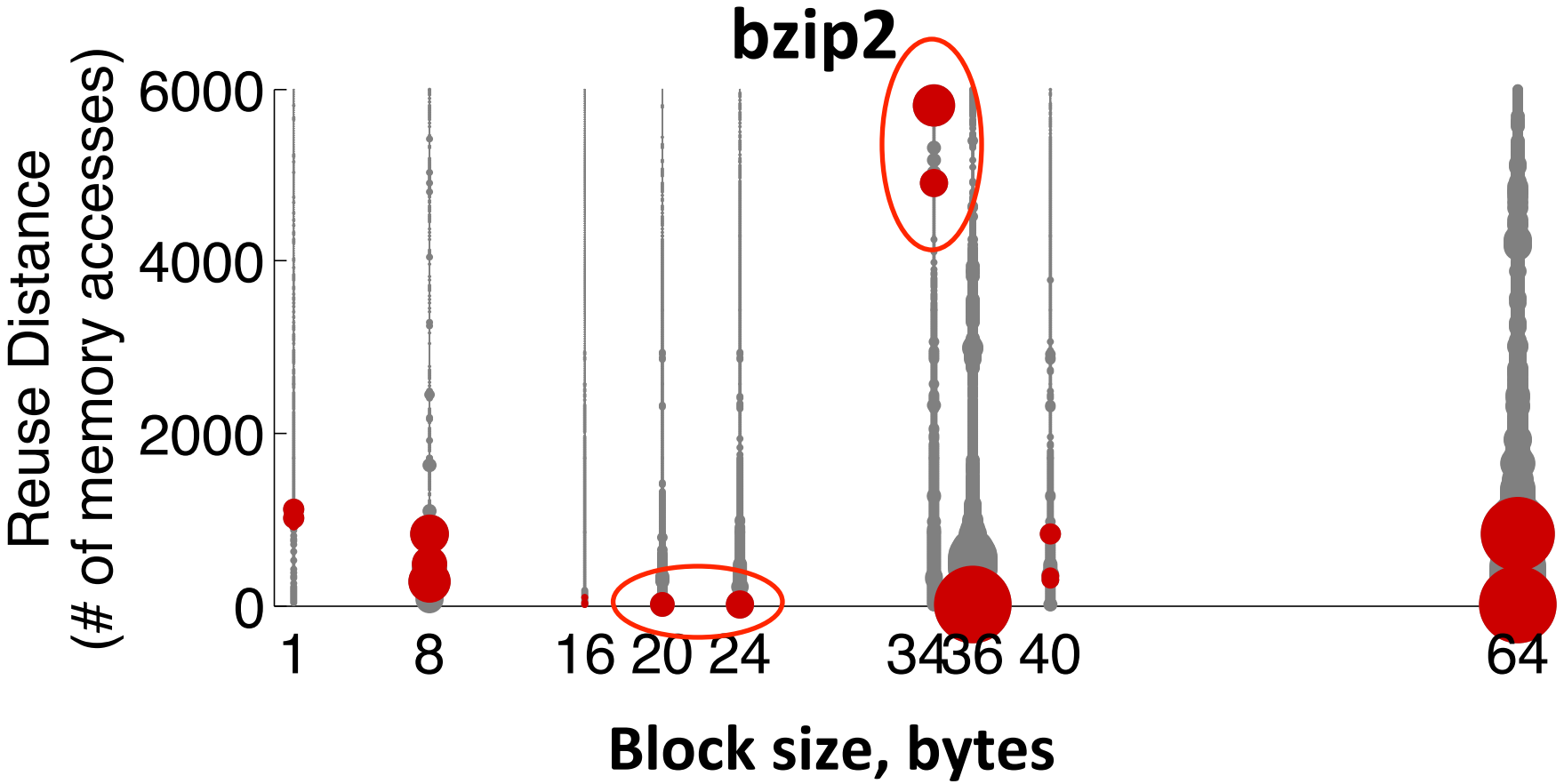
#2: Compressed Block Size Varies

BDI compression algorithm *[Pekhimenko+, PACT'12]*



Compressed block size varies within and between applications

#3: Block Size Can Indicate Reuse



Different sizes have different dominant reuse distances

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];      <-- long reuse, compressible
    for (int j=0; j<N; j++) {
        sum += B[(idx+j)%16];
    }
}
```

short reuse, incompressible

Compressed size can be an indicator of reuse distance

Size-Aware Cache Management

1. Compressed block size matters

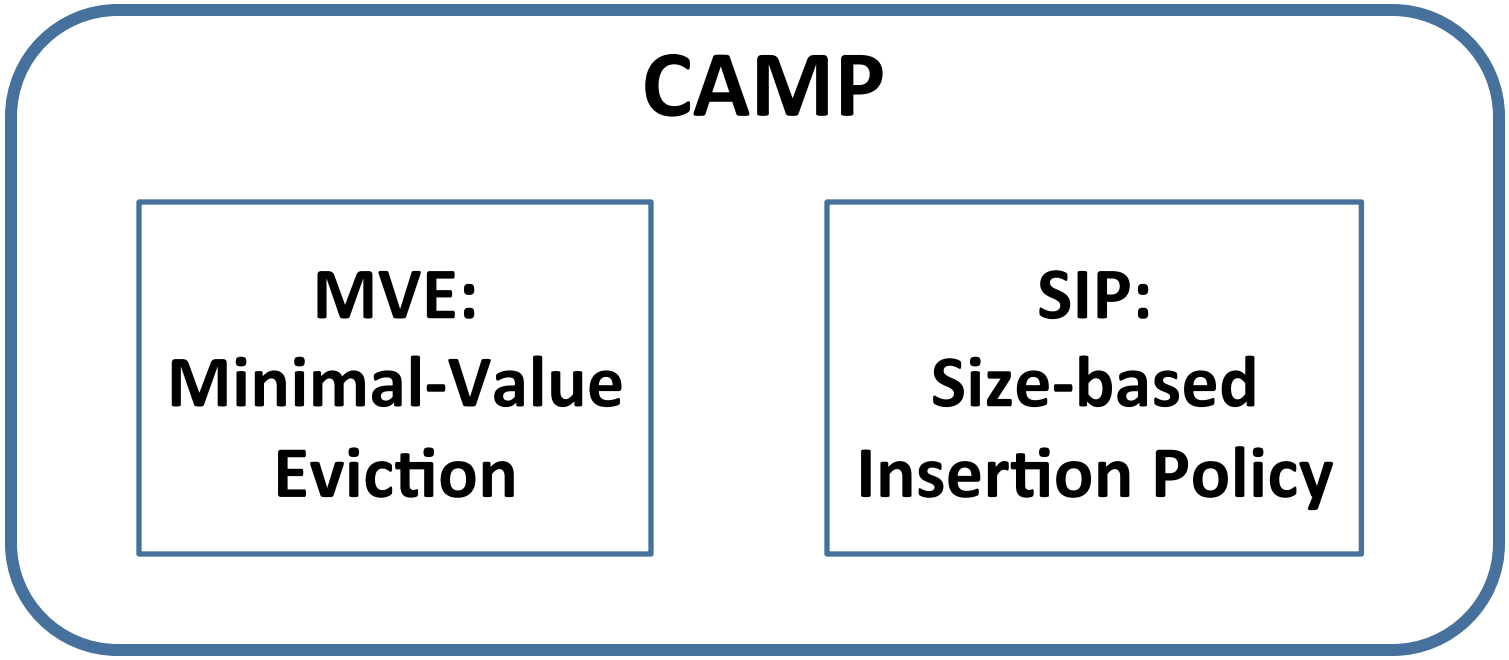
2. Compressed block size varies

3. Compressed block size can indicate reuse

Outline

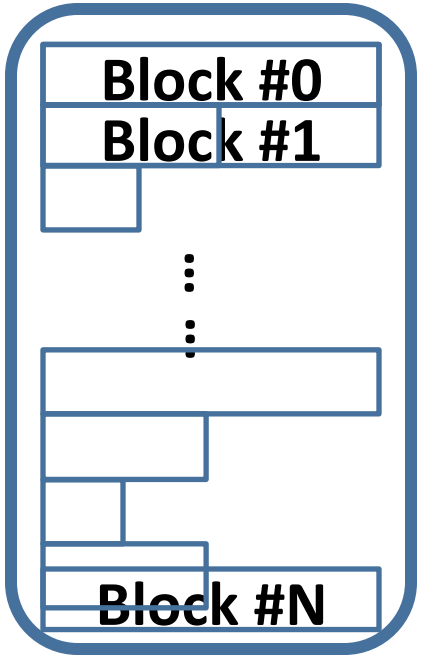
- Motivation
- Key Observations
- Key Ideas:
  - Minimal-Value Eviction (MVE)
  - Size-based Insertion Policy (SIP)
- Evaluation
- Conclusion

Compression-Aware Management Policies (CAMP)



Minimal-Value Eviction (MVE): Observations

Set:



Highest value
Highest priority: MRU

Importance of the block depends on both the likelihood of reference and the **compressed block size**

Lowest value
Lowest priority: LRU

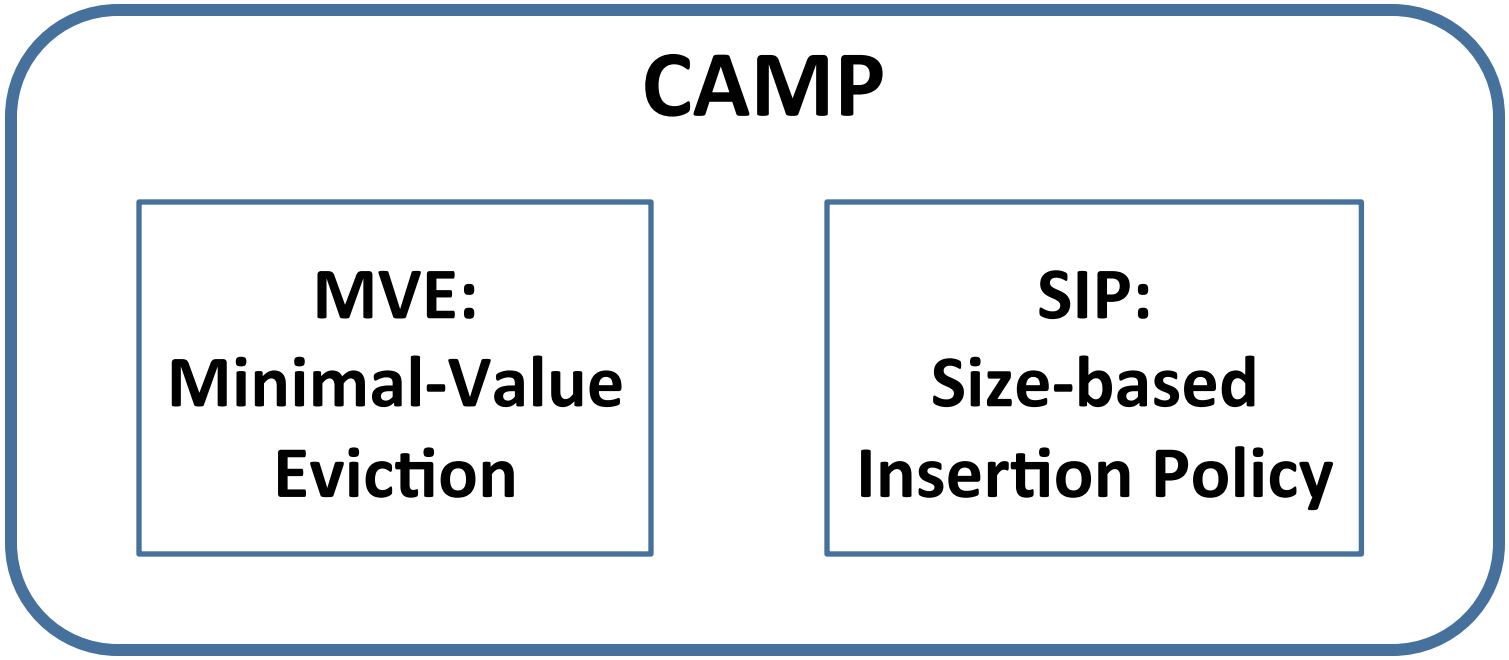
Minimal-Value Eviction (MVE): Key Idea

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

Probability of reuse:

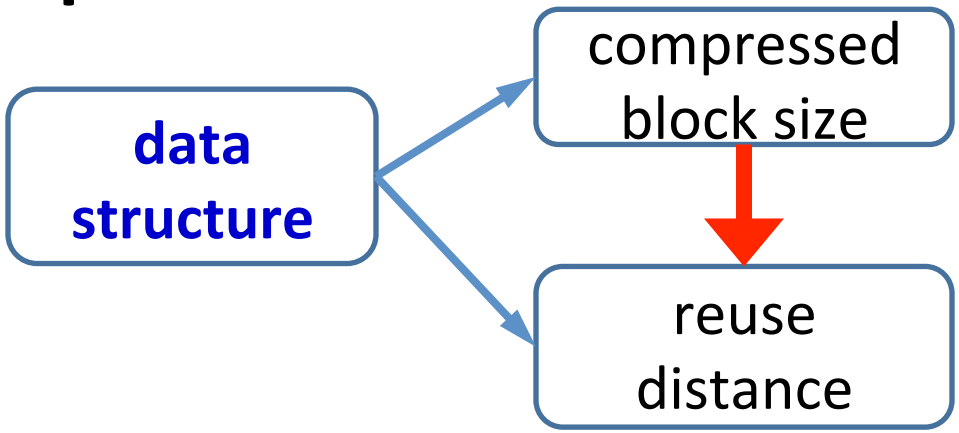
- Can be determined in many different ways
- Our implementation is based on re-reference interval prediction value (RRIP *[Jaleel+, ISCA'10]*)

Compression-Aware Management Policies (CAMP)



Size-based Insertion Policy (SIP): Observations

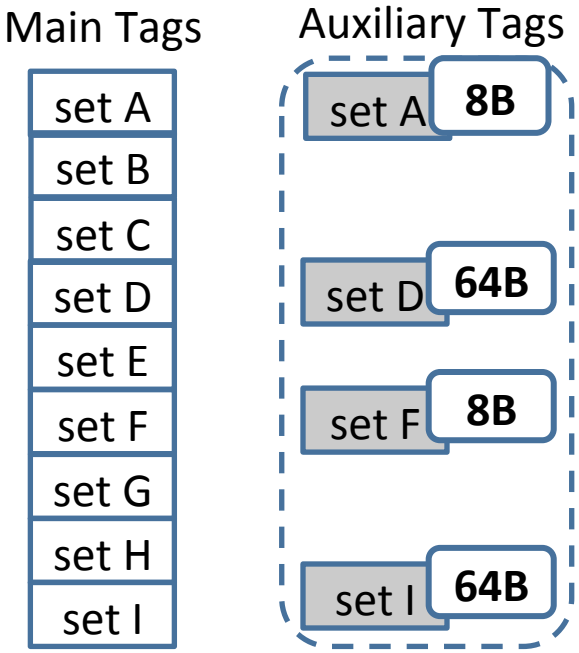
- Sometimes there is a **relation** between the **compressed block size** and **reuse** distance

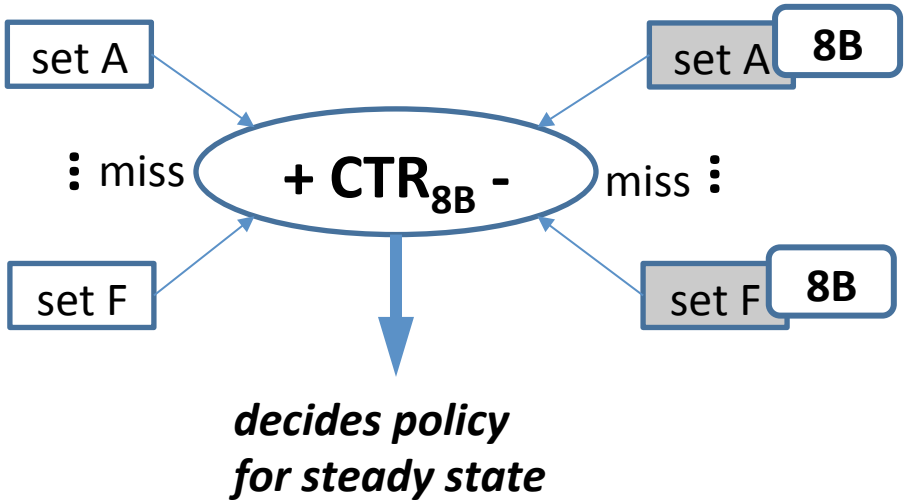


- This relation can be **detected** through the compressed block size
- **Minimal** overhead to track this relation (compressed block information is a part of design)

Size-based Insertion Policy (SIP): Key Idea

- Insert blocks of a certain size with higher priority if this improves cache miss rate
- Use dynamic set sampling *[Qureshi, ISCA'06]* to detect which size to insert with higher priority





Outline

- Motivation
- Key Observations
- Key Ideas:
  - Minimal-Value Eviction (MVE)
  - Size-based Insertion Policy (SIP)
- Evaluation
- Conclusion

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) *[Pekhimenko+, PACT'12]*
  - 4GHz, x86 in-order core, cache size (**1MB - 16MB**)

Evaluated Cache Management Policies

Design	Description
LRU	Baseline LRU policy - size-unaware
RRIP	Re-reference Interval Prediction *[Jaleel+, ISCA'10]* - size-unaware

Evaluated Cache Management Policies

Design	Description
LRU	Baseline LRU policy - size-unaware
RRIP	Re-reference Interval Prediction *[Jaleel+, ISCA'10]* - size-unaware
ECM	Effective Capacity Maximizer *[Baek+, HPCA'13]* + size-aware

Evaluated Cache Management Policies

Design	Description
LRU	Baseline LRU policy - size-unaware
RRIP	Re-reference Interval Prediction *[Jaleel+, ISCA'10]* - size-unaware
ECM	Effective Capacity Maximizer *[Baek+, HPCA'13]* + size-aware
CAMP	Compression-Aware Management Policies (MVE + SIP)

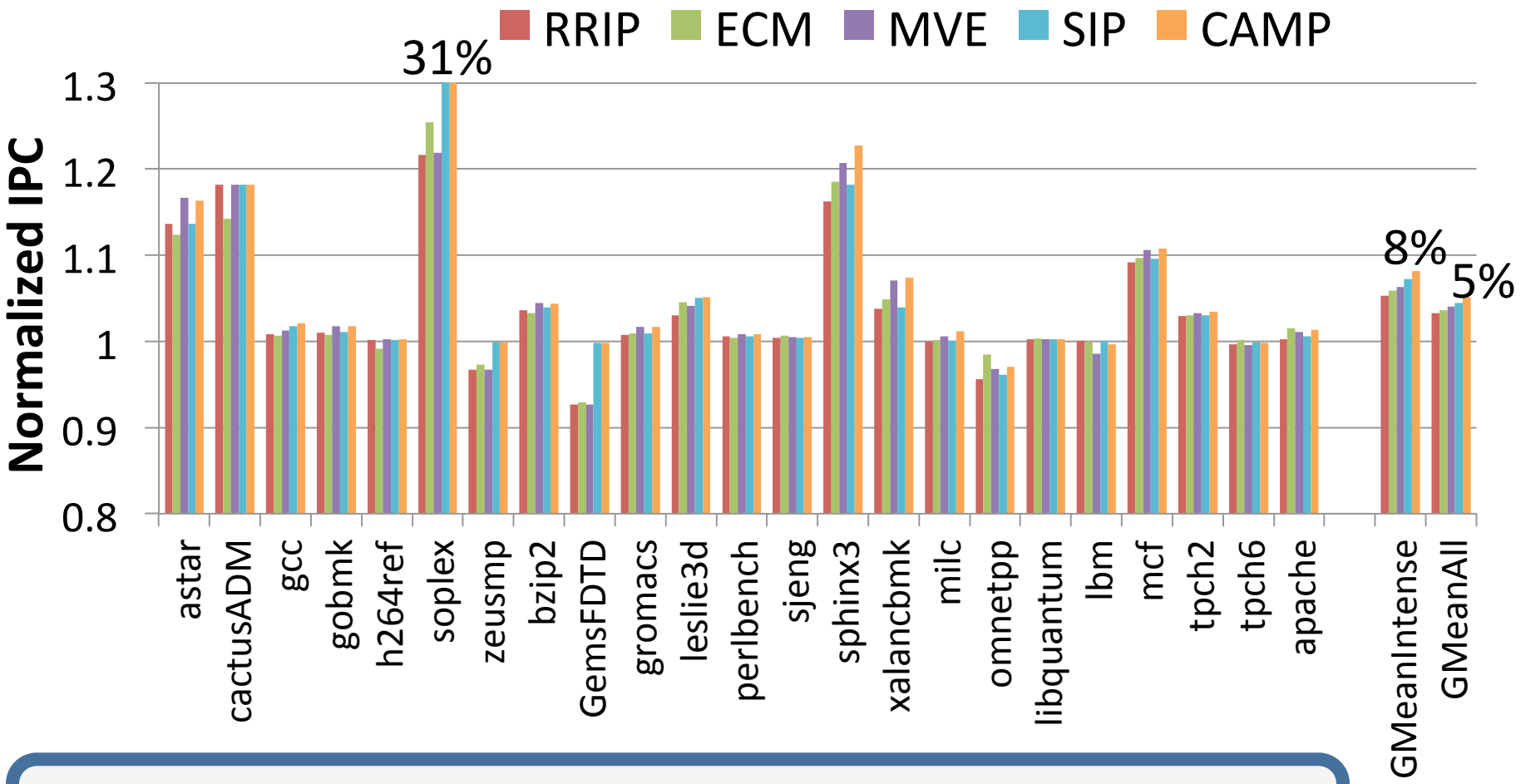
Size-Aware Replacement

- Effective Capacity Maximizer (ECM)
  [Baek+, HPCA'13]
  - Inserts "big" blocks with lower priority
  - Uses heuristic to define the threshold

- Shortcomings
  - Coarse-grained
  - No relation between block size and reuse
  - Not easily applicable to other cache organizations

CAMP Single-Core

SPEC2006, databases, web workloads, 2MB L2 cache



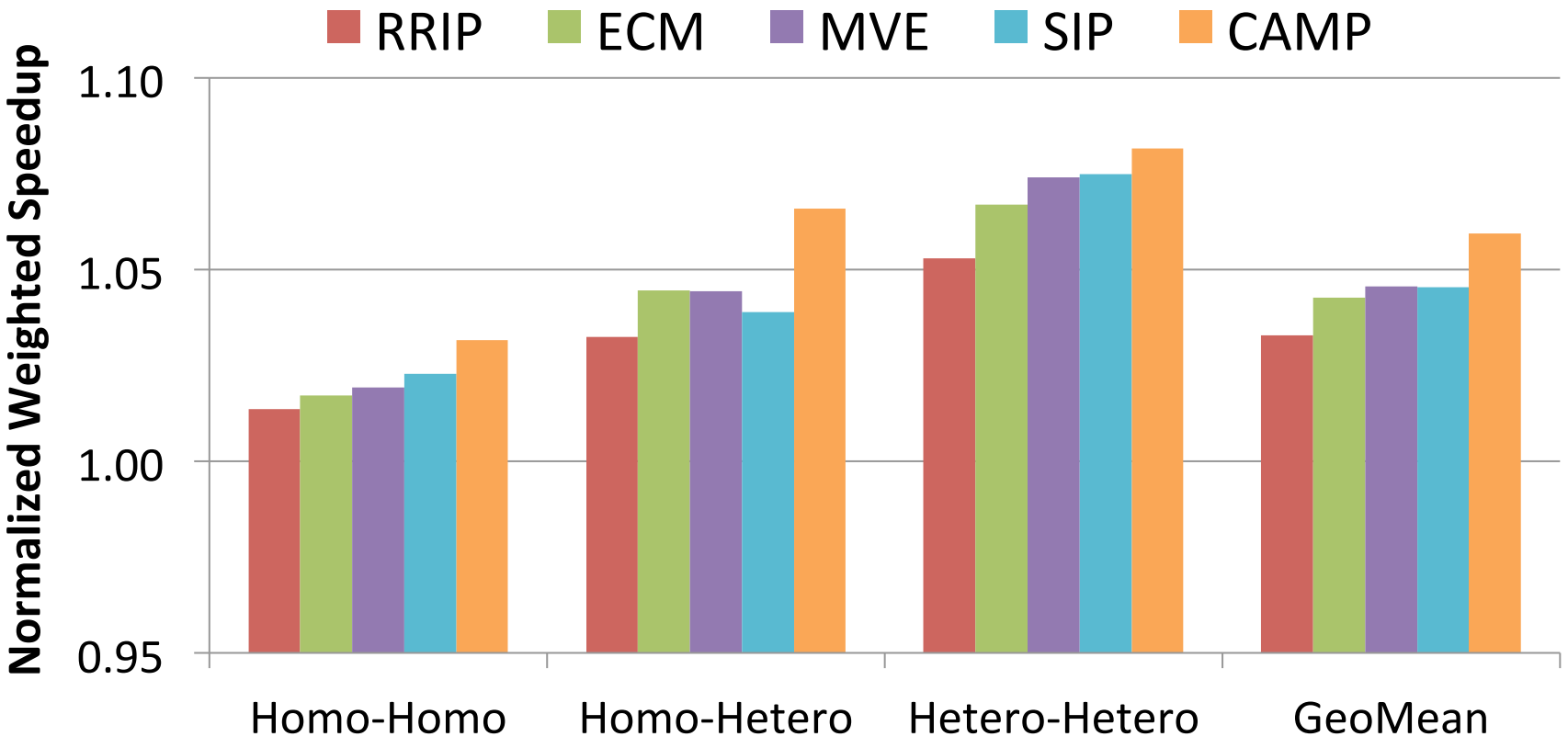
CAMP improves performance over LRU, RRIP, ECM

Multi-Core Results

- Classification based on the compressed block size distribution
  - **Homo**geneous (1 or 2 dominant block size(s))
  - **Hetero**geneous (more than 2 dominant sizes)

- We form three different 2-core workload groups (20 workloads in each):
  - Homo-Homo
  - Homo-Hetero
  - Hetero-Hetero

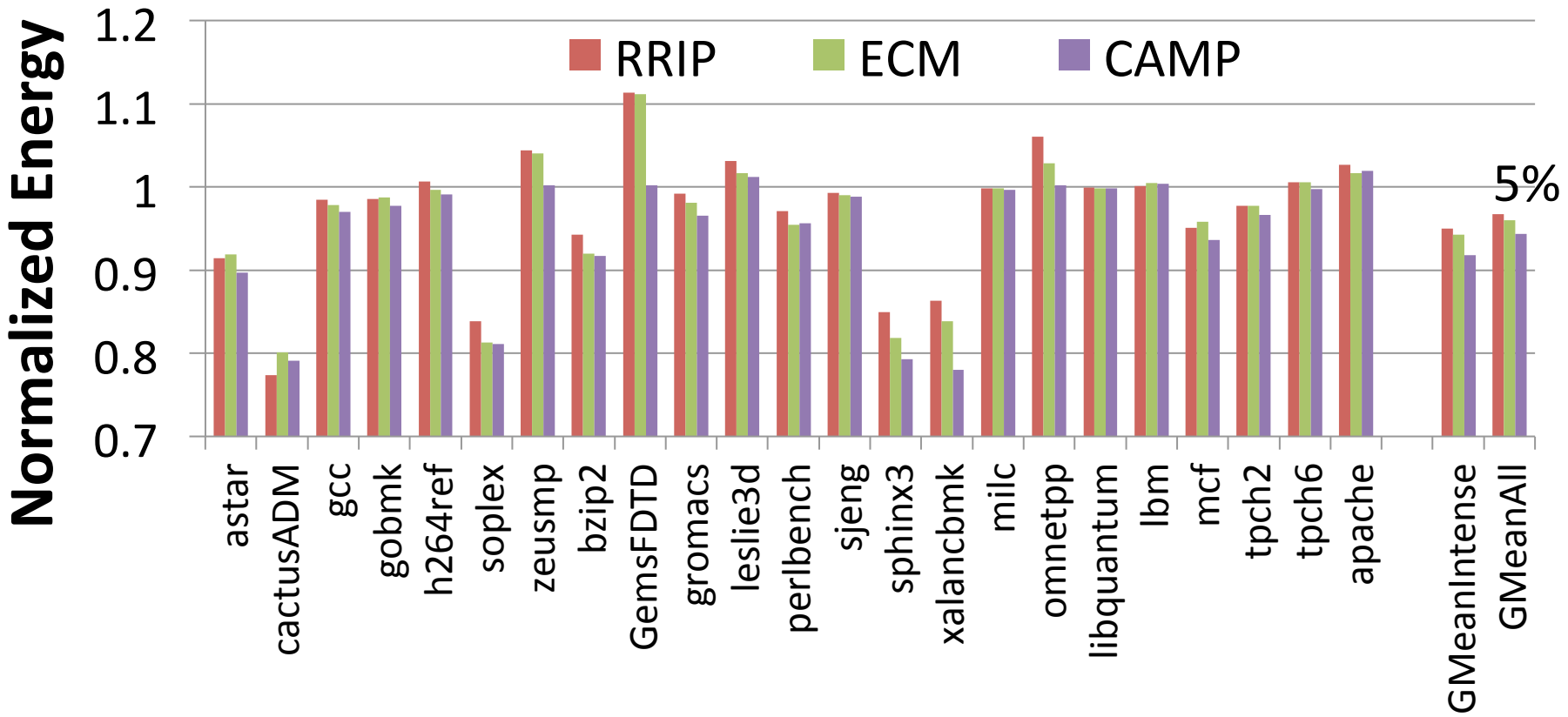
Multi-Core Results (2)



Higher benefits with higher heterogeneity in size

Effect on Memory Subsystem Energy

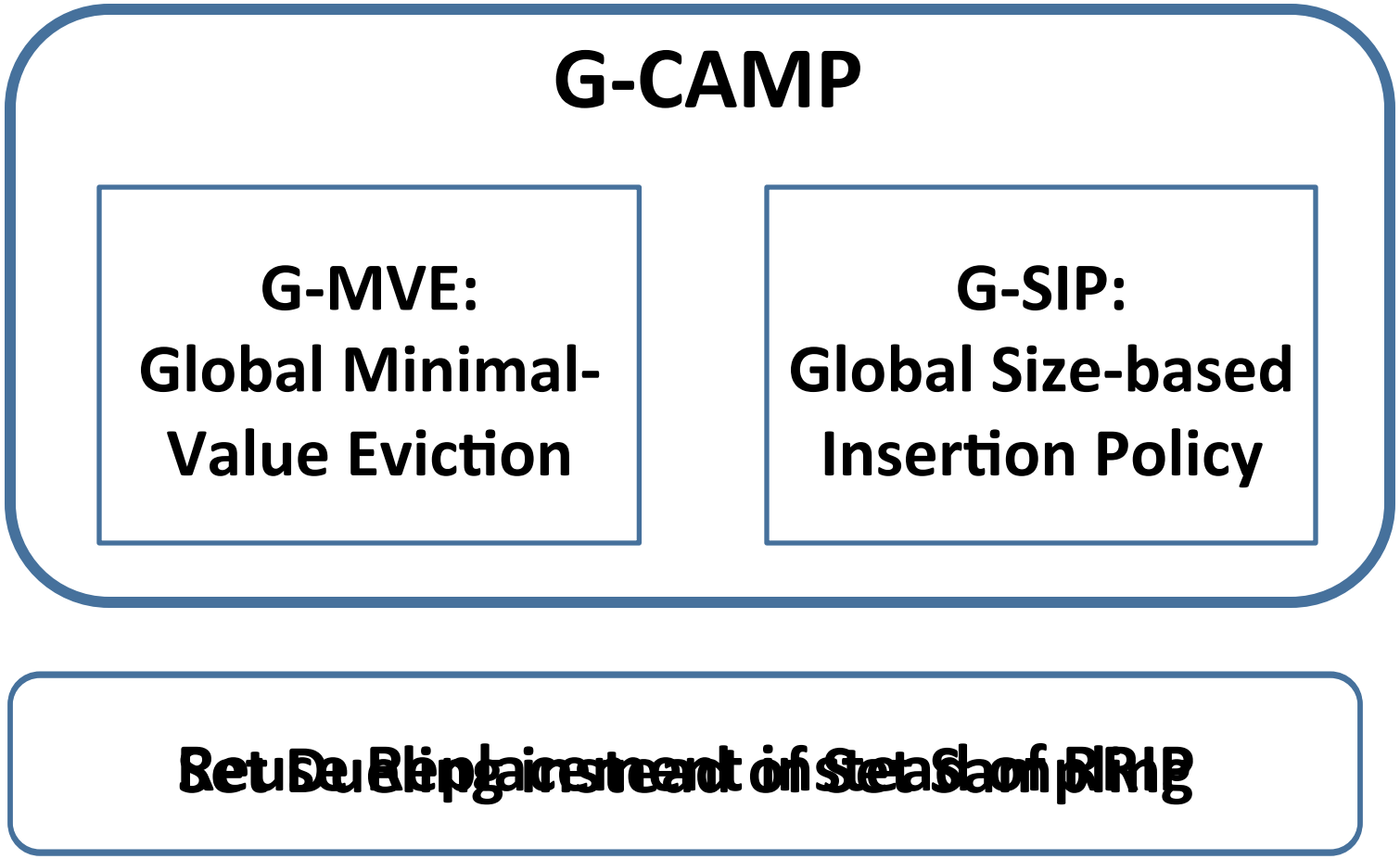
L1/L2 caches, DRAM, NoC, compression/decompression



CAMP reduces energy consumption in the memory subsystem

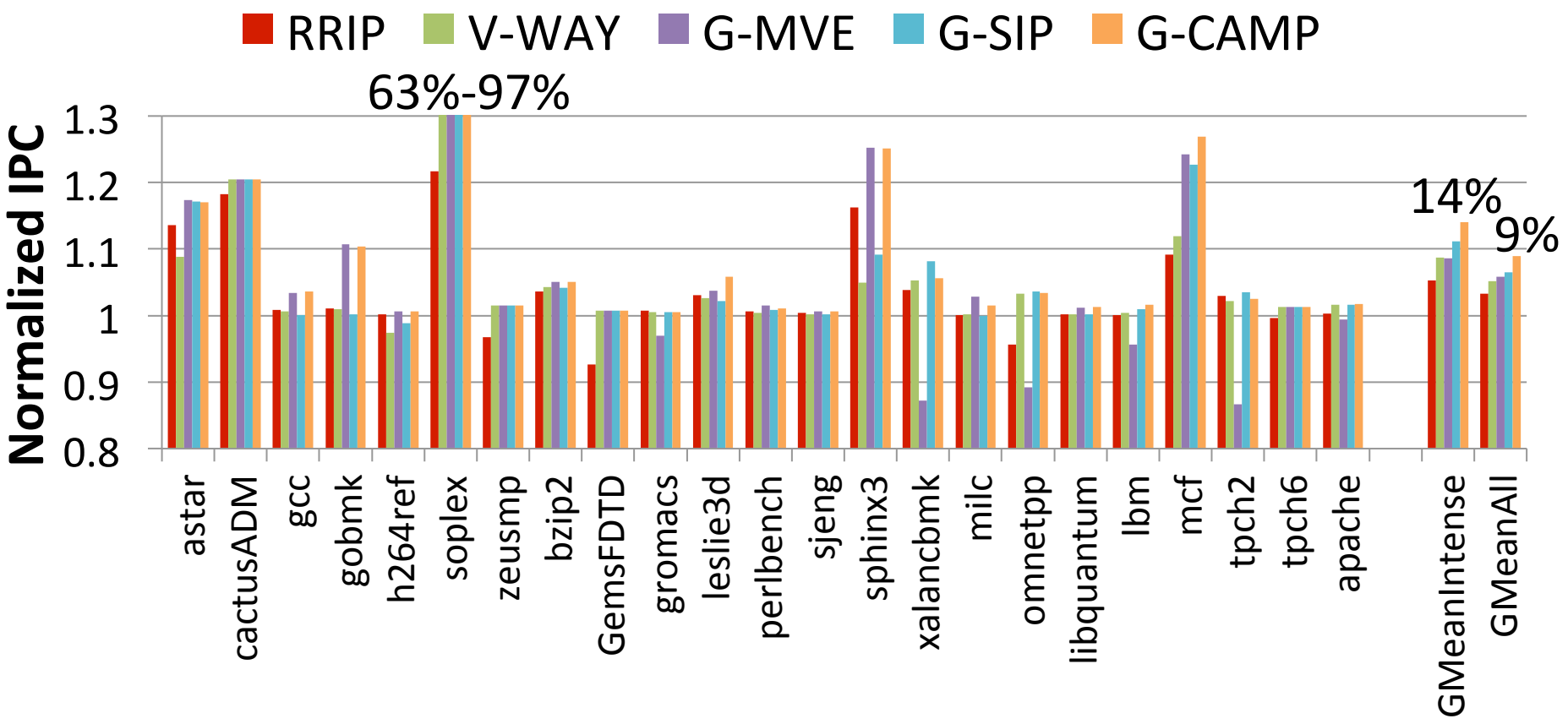
CAMP for Global Replacement

CAMP with V-Way *[Qureshi+, ISCA'05]* cache design



G-CAMP Single-Core

SPEC2006, databases, web workloads, 2MB L2 cache

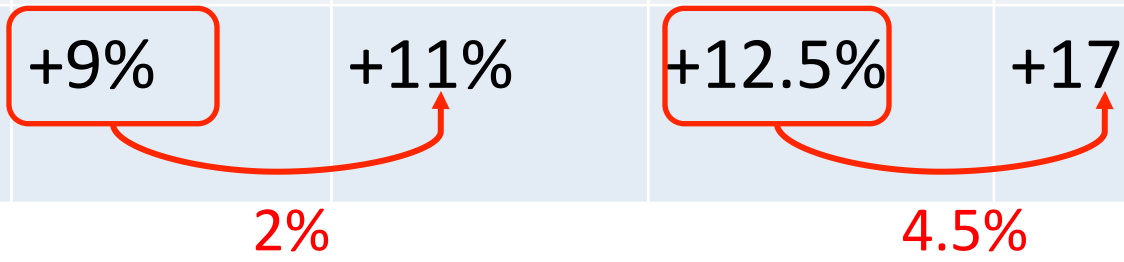


G-CAMP improves performance over LRU, RRIP, V-Way

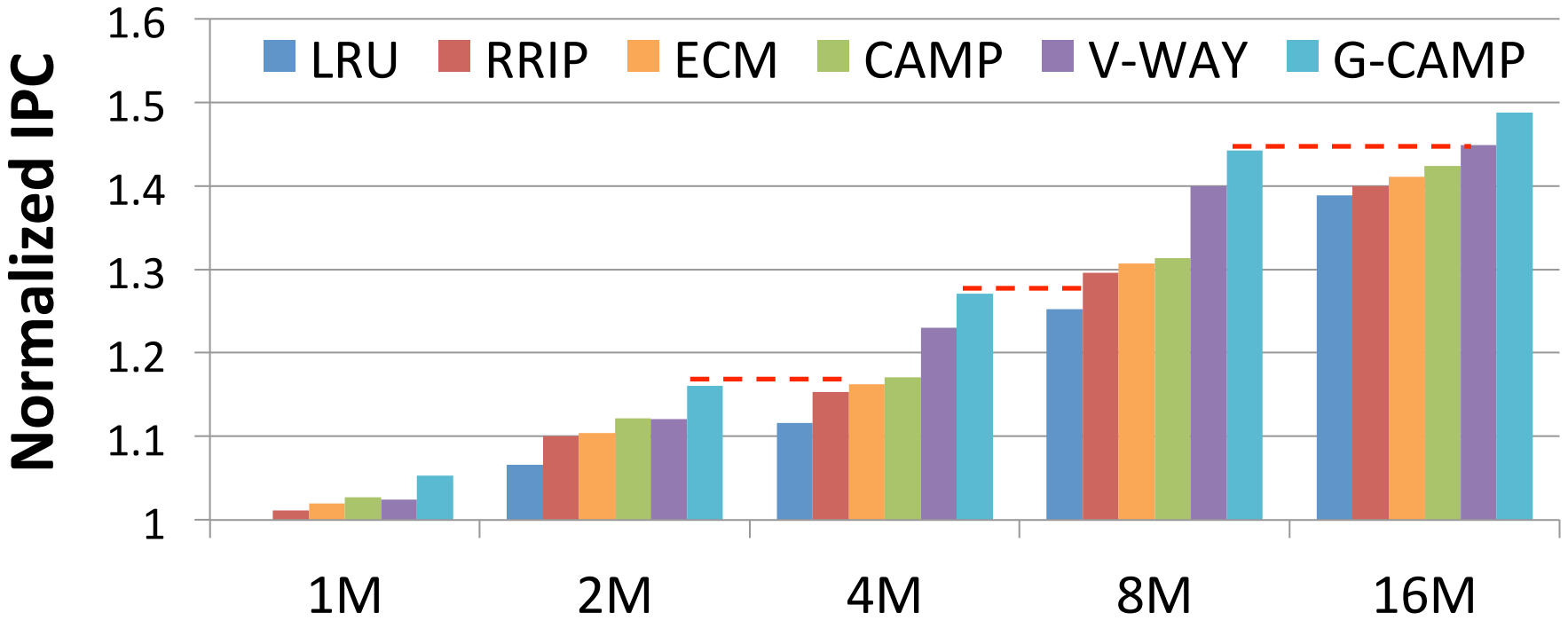
Storage Bit Count Overhead

Over **uncompressed** cache with LRU

Designs (with BDI)	LRU	CAMP	V-Way	G-CAMP
tag-entry	+14b	+14b	+15b	+19b
data-entry	+0b	+0b	+16b	+32b
total	+9%	+11%	+12.5%	+17%



Cache Size Sensitivity



G-CAMP outperforms LRU with **2X** cache sizes

Other Results and Analyses in the Paper

- Block size distribution for different applications
- Sensitivity to the compression algorithm
- Comparison with uncompressed cache
- Effect on cache capacity
- SIP vs. PC-based replacement policies
- More details on multi-core results

Conclusion

- In a compressed cache, **compressed block size** is an additional dimension
- **Problem**: How can we maximize cache performance utilizing both block reuse and compressed size?
- **Observations**:
  - Importance of the cache block depends on its compressed size
  - Block size is indicative of reuse behavior in some applications
- **Key Idea**: Use compressed block size in making cache replacement and insertion decisions
- **Two techniques**: Minimal-Value Eviction and Size-based Insertion Policy
- **Results:**
  - Higher performance (**9%/11%** on average for 1/2-core)
  - Lower energy (**12%** on average)

Exploiting Compressed Block Size as an Indicator of Future Reuse

Gennady Pekhimenko,
Tyler Huberty, Rui Cai,
Onur Mutlu, Todd C. Mowry

Phillip B. Gibbons,
Michael A. Kozuch

SAFARI Carnegie Mellon University intel

Backup Slides

Potential for Data Compression

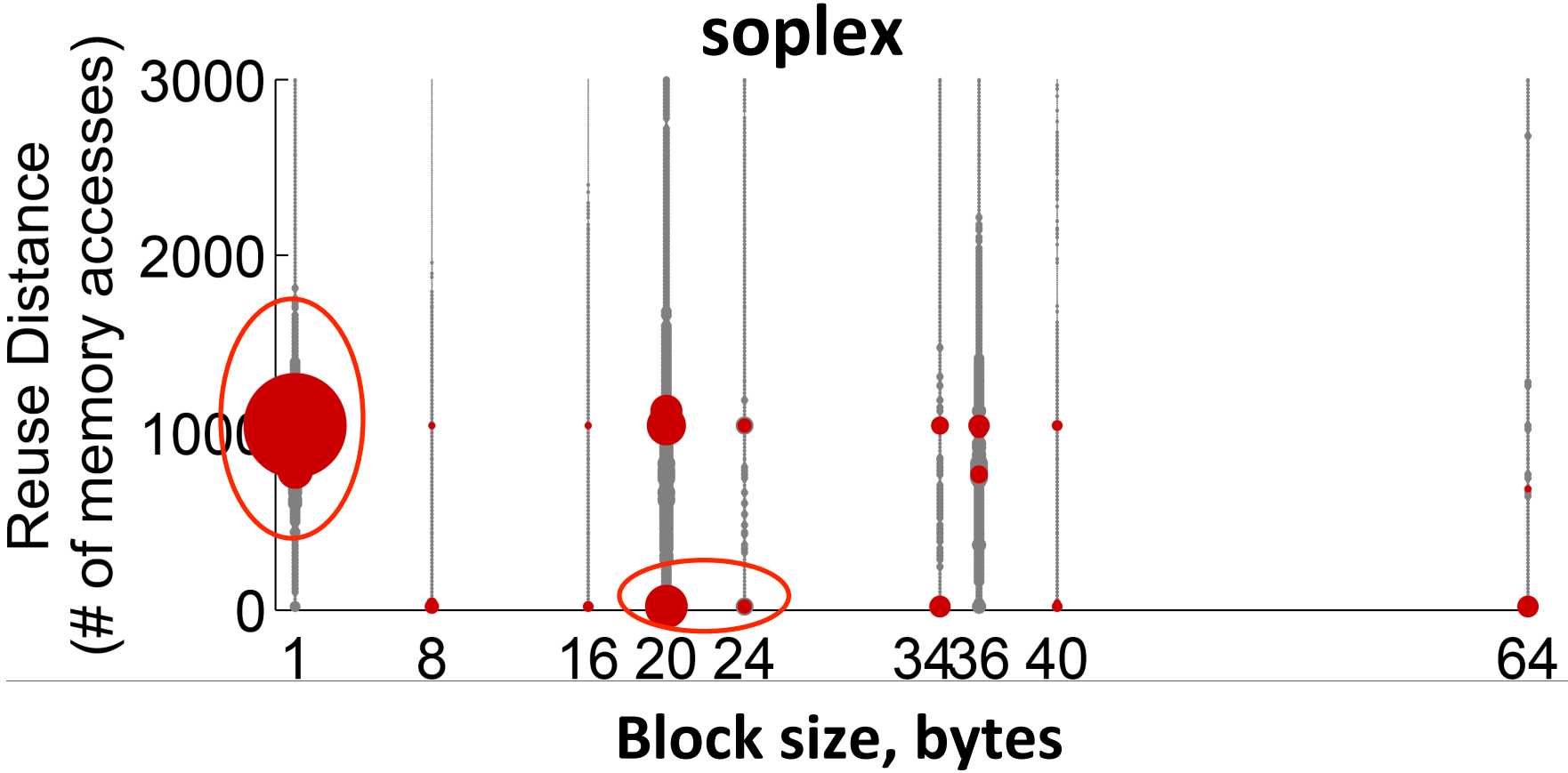
Compression algorithms for on-chip caches:

- Frequent Pattern Compression (FPC) *[Alameldeen+, ISCA'04]*
  - performance: **+15%,** comp. ratio: **1.25-1.75**
- C-Pack *[Chen+, Trans. on VLSI'10]*
  - comp. ratio: **1.64**
- Base-Delta-Immediate (BDI) *[Pekhimenko+, PACT'12]*
  - performance: **+11.2%,** comp. ratio: **1.53**
- Statistical Compression *[Arelakis+, ISCA'14]*
  - performance: **+11%,** comp. ratio: **2.1**

Potential for Data Compression (2)

- Better compressed cache management
  - Decoupled Compressed Cache (DCC) *[MICRO'13]*
  - Skewed Compressed Cache *[MICRO'14]*

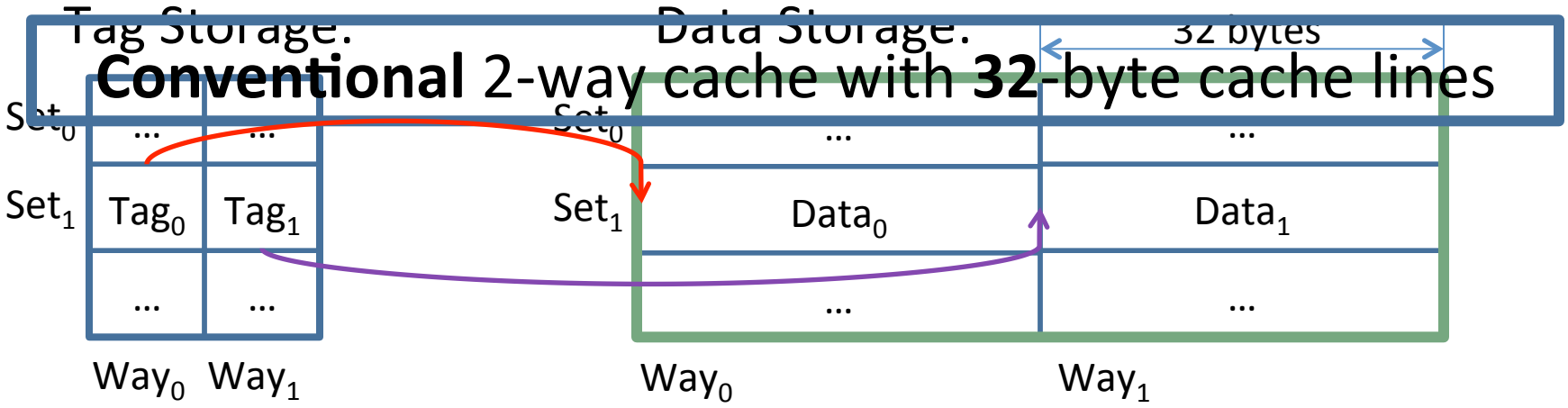
3: Block Size Can Indicate Reuse



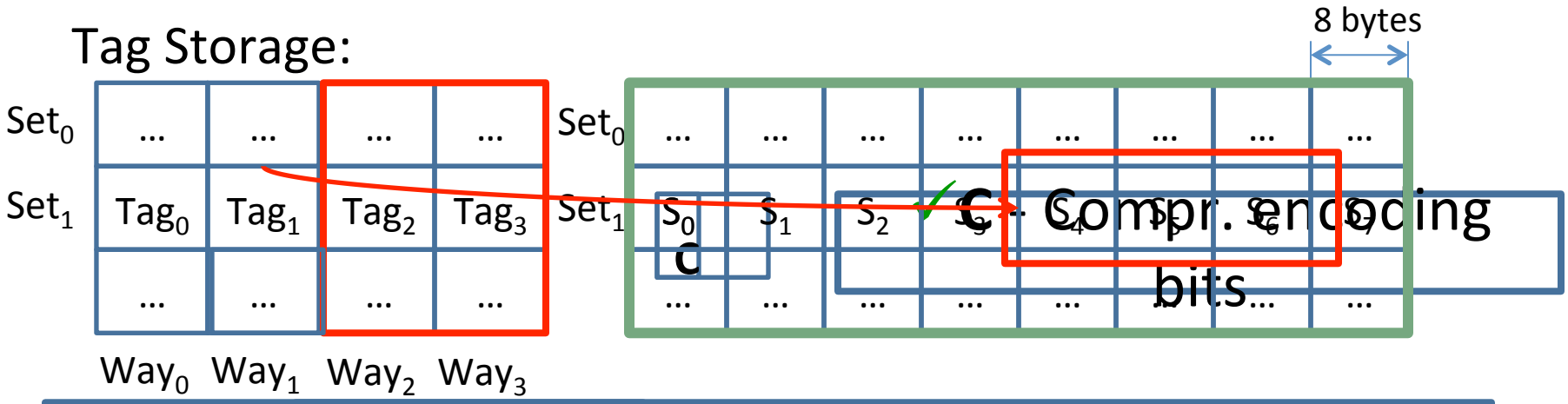
Different sizes have different dominant reuse distances

Conventional Cache Compression

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



Compressed 4-way cache with **8**-byte segmented data



✓Twice as many tags

✓Tags map to multiple adjacent segments

CAMP = MVE + SIP

Minimal-Value eviction (MVE)

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

- Probability of reuse based on RRIP *[ISCA'10]*

Size-based insertion policy (SIP)

- Dynamically prioritizes blocks based on their compressed size (e.g., insert into MRU position for LRU policy)

Overhead Analysis

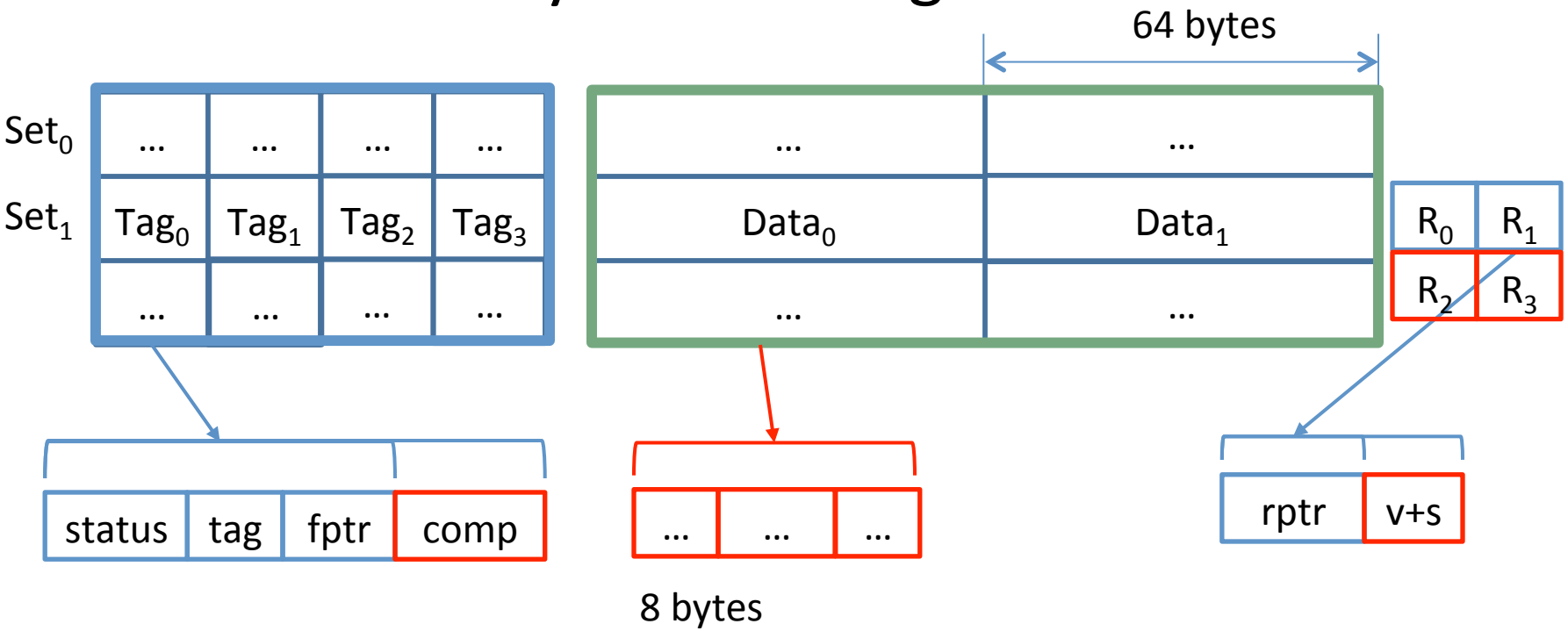
	Base	BDI	CAMP	V-Way	V-Way+C	G-CAMP
tag-entry(bits)	21	35([37])	35	36 [a]	40 [e]	40
data-entry(bits)	512	512	512	528 [b]	544 [f]	544
# tag entries	32768	65536	73728 [c]	65536	65536	65536
# data entries	32768	32768	32768	32768	32768	32768
tag-store (kB)	86	287	323	295	328	328
data-store (kB)	2097	2097	2097	2163	2228	2228
other	0	0	8*16 [d]	0	0	8*16
total (kB)	2183	**2384**	**2420**	2458	**2556**	**2556**

Table 1: Storage overhead of different mechanisms for a 2MB L2 cache. "V-Way+C" means V-Way with compression.

[a] +15 forward ptr; [b] +16 reverse ptr; [c] +1/8 set sampling in **SIP**; [d] CTR's in **SIP**; [e] +4 for comp. encoding; [f] +32 (2 reverse ptrs per data entry, 13 bits each, and 2 extended validity bits, 3 bits each)

CAMP and Global Replacement

CAMP with V-Way cache design



G-MVE

Two major changes:

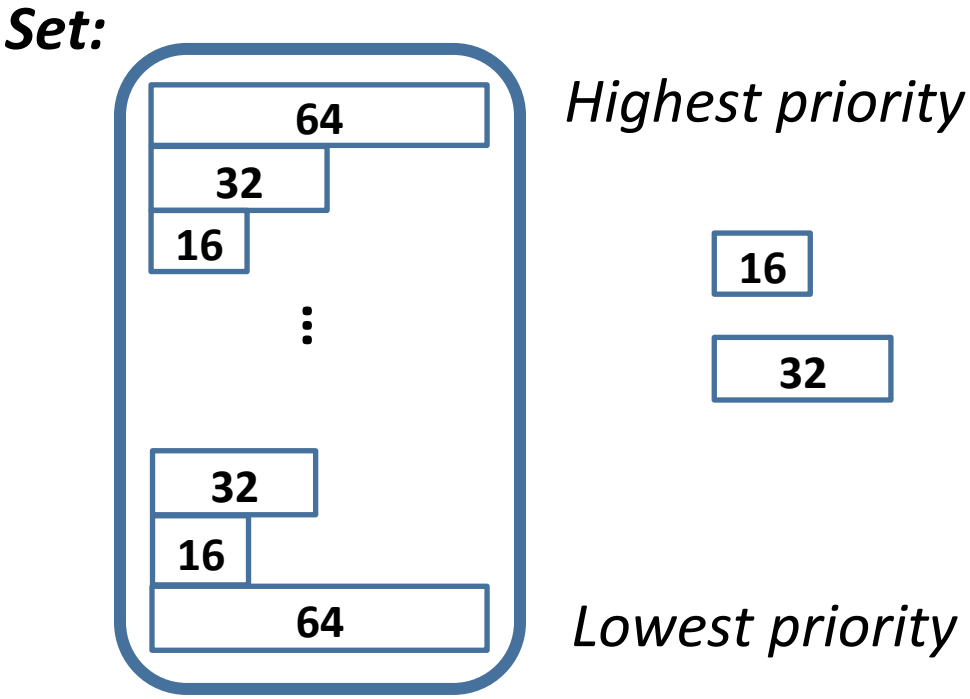
- **Probability of reuse** based on Reuse Replacement policy
  - On insertion, a block's counter is set to zero
  - On a hit, the block's counter is incremented by one indicating its reuse
- Global replacement using a pointer (PTR) to a reuse counter entry
  - Starting at the entry PTR points to, the reuse counters of 64 valid data entries are scanned, decrementing each non-zero counter

G-SIP

- Use set dueling instead of set sampling (SIP) to decide best insertion policy:
  - Divide data-store into 8 regions and block sizes into 8 ranges
  - During training, for each region, insert blocks of a different size range with higher priority
  - Count cache misses per region
- In steady state, insert blocks with sizes of top performing regions with higher priority in all regions

Size-based Insertion Policy (SIP): Key Idea

- Insert blocks of a certain size with higher priority if this improves cache miss rate



Evaluated Cache Management Policies

Design	Description
LRU	Baseline LRU policy
RRIP	Re-reference Interval Prediction *[Jaleel+, ISCA'10]*
ECM	Effective Capacity Maximizer *[Baek+, HPCA'13]*
V-Way	Variable-Way Cache *[ISCA'05]*
CAMP	Compression-Aware Management (Local)
G-CAMP	Compression-Aware Management (Global)

Performance and MPKI Correlation

Performance improvement / MPKI reduction

Mechanism	LRU	RRIP	ECM
CAMP	8.1%/-13.3%	2.7%/-5.6%	2.1%/-5.9%

CAMP performance improvement correlates with MPKI reduction

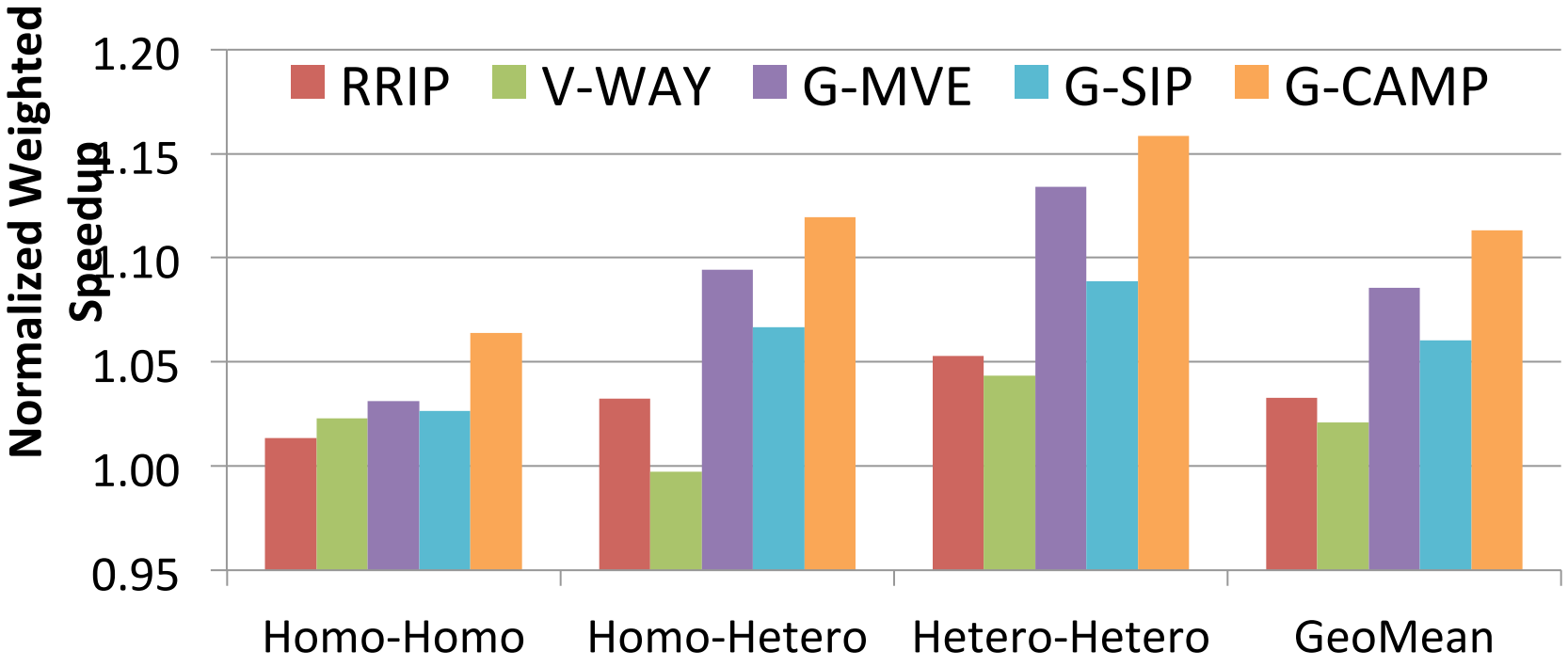
Performance and MPKI Correlation

Performance improvement / MPKI reduction

Mechanism	LRU	RRIP	ECM	V-Way
CAMP	8.1%/-13.3%	2.7%/-5.6%	2.1%/-5.9%	N/A
G-CAMP	14.0%/-21.9%	8.3%/-15.1%	7.7%/-15.3%	4.9%/-8.7%

CAMP performance improvement correlates with MPKI reduction

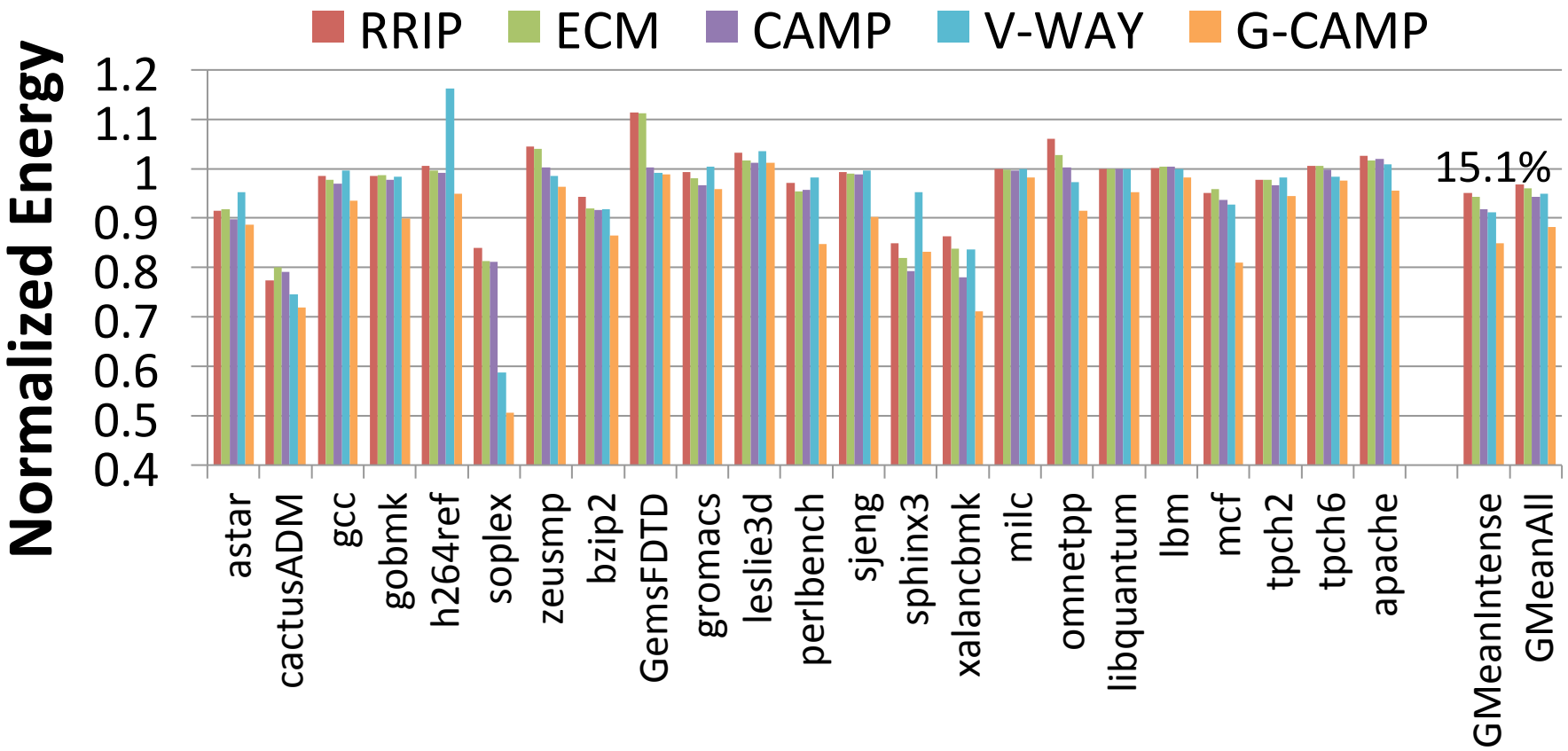
Multi-core Results (2)



G-CAMP outperforms LRU, RRIP and V-Way policies

Effect on Memory Subsystem Energy

L1/L2 caches, DRAM, NoC, compression/decompression



CAMP/G-CAMP reduces energy consumption in the memory subsystem