

# THE ZCACHE: DECOUPLING WAYS AND ASSOCIATIVITY

Daniel Sanchez and Christos Kozyrakis  
Stanford University

MICRO-43, December 6<sup>th</sup> 2010

# Executive Summary

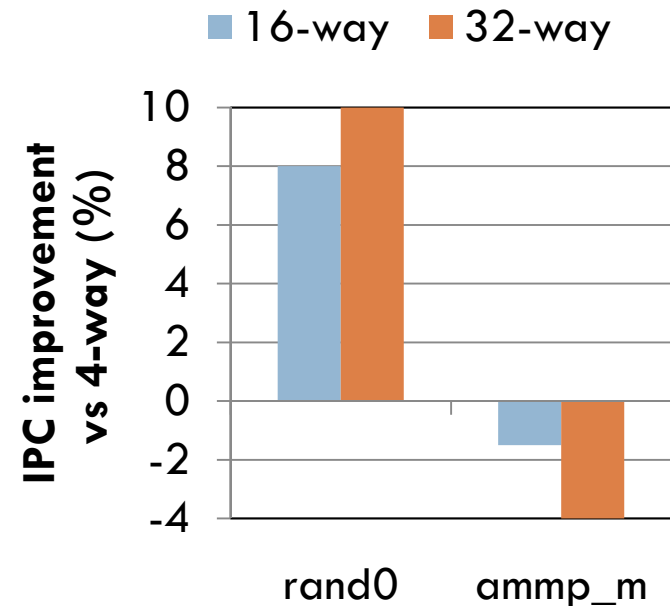
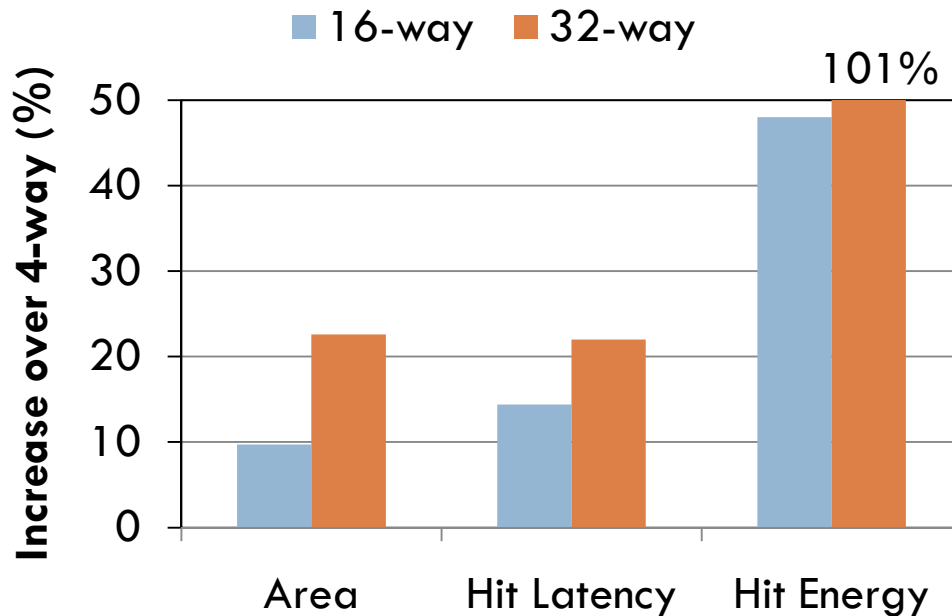
- Mitigating the memory wall requires large, highly associative caches
  - Last-level caches take ~50% chip area, have 24-32 ways in latest CMPs
  - More ways → large energy, latency and area overheads
  
- ZCache: A highly associative cache with a low number of ways
  - Improves associativity by increasing number of replacement candidates
  - Retains low energy/hit, latency and area of caches with few ways
  - Based on skew-associative caches and cuckoo hashing
  
- Analytical framework explains why zcache works
  - Associativity depends on number of **replacement candidates**, not ways or locations a block can be in

# Outline

- Introduction
- ZCache
- Analytical Framework
- Evaluation

# Introduction

- Uses of high associativity:
  - ▣ Improve performance by reducing conflict misses
  - ▣ Partitioning, pinning, storing speculative data ( e.g. TM, TLS)
- Increasing number of ways affects area, delay, energy



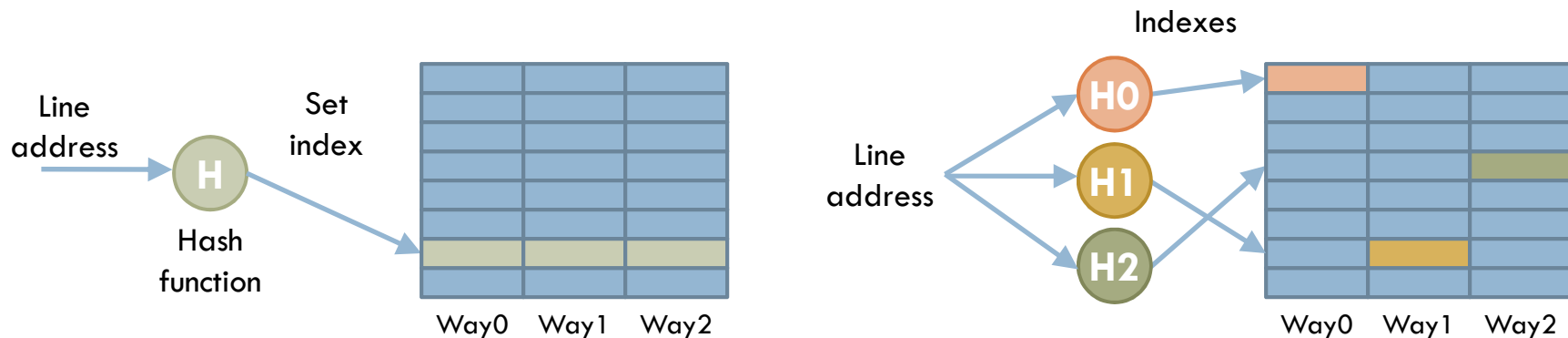
# Techniques for high associativity (1 / 2):

## ~~Increase number of locations~~

- Allow multiple locations per way
  - ▣ Column-associative caches [Agarwal93], set-balancing cache [Rolan09], ...
  - ▣ Hit latency ↑, hit energy ↑
- Use a victim cache
  - ▣ VC [Jouppi90], Scavenger [Basu07], ...
  - ▣ Area ↑, hit latency ↑, hit energy ↑
- Use indirection in the tag array
  - ▣ IIC [Hallnor00], V-Way cache [Qureshi05]
  - ▣ Area ↑, hit latency ↑, hit energy ↑

# Techniques for high associativity (2/2): Better hashing

- Use a hash function to index the cache
  - ▣ Simple hashing significantly reduces conflicts [Karbutli04]
- Skew-associative caches [Seznec93]
  - ▣ Index each way using a different hash function
  - ▣ A line conflicts with a different set of lines on each way, reducing conflict misses
  - ▣ No sets, cannot use replacement policy that relies on set ordering

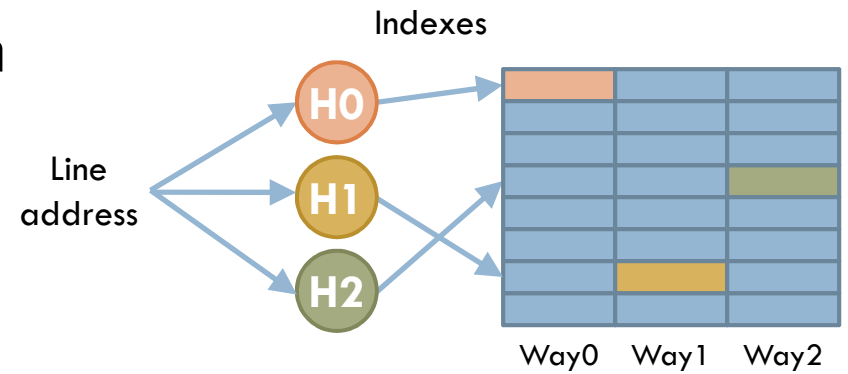


# Outline

- Introduction
- ZCache
- Analytical Framework
- Evaluation

# The ZCache Design

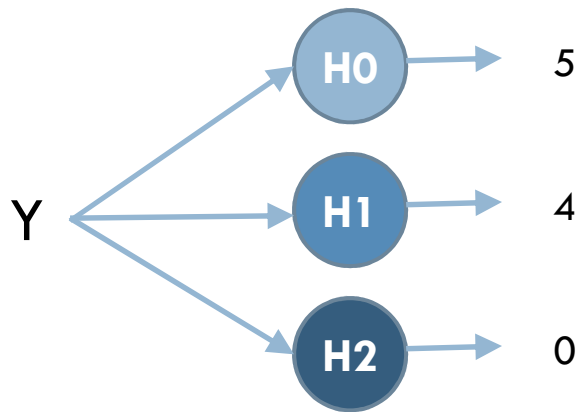
- Lookups and hits happen as in a skew-associative cache



- Misses exploit the multiple hash functions to obtain an arbitrarily large number of replacement candidates
  - ▣ Phase 1: Walk the tag array, get best candidate
  - ▣ Phase 2: Move a few lines to fit everything
  - ▣ This happens **infrequently** (on misses) and **off the critical path**
  - ▣ Draws on prior research in cuckoo hashing



# ZCache Replacement

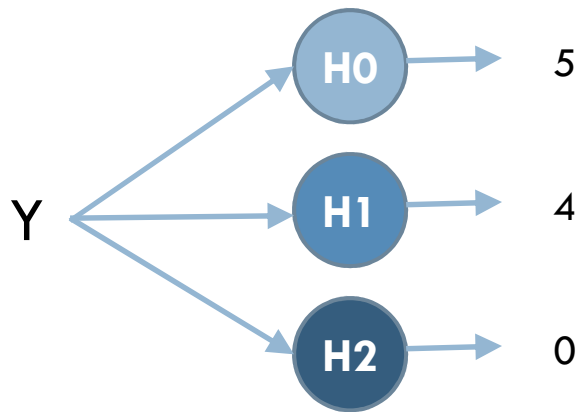


Way 0	Way 1	Way 2	
U	V	M	0
F	C	X	1
P	K	H	2
B	E	R	3
N	D	J	4
A	Z	Q	5
G	T	I	6
L	O	S	7



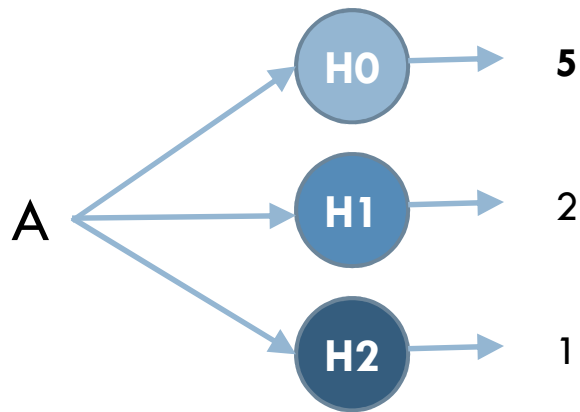
- Start replacement process while fetching Y

# ZCache Replacement



Way 0	Way 1	Way 2	
U	V	M	0
F	C	X	1
P	K	H	2
B	E	R	3
N	D	J	4
A	Z	Q	5
G	T	I	6
L	O	S	7

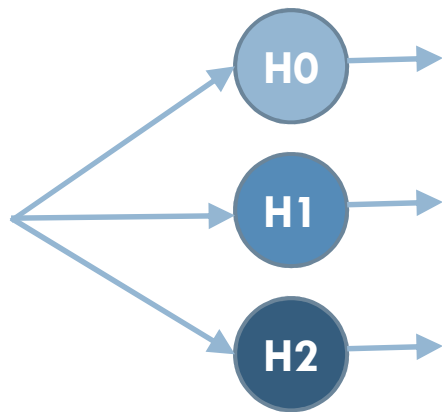
# ZCache Replacement



Way 0	Way 1	Way 2	
U	V	M	0
F	C	X	1
P	K	H	2
B	E	R	3
N	D	J	4
A	Z	Q	5
G	T	I	6
L	O	S	7

- Instead of evicting A, can **move** it and evict K or X

# ZCache Replacement

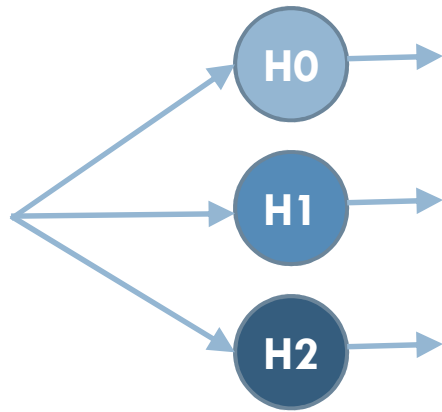


	Way 0	Way 1	Way 2	
	U	V	M	0
	F	C	X	1
	P	K	H	2
	B	E	R	3
	N	D	J	4
	A	Z	Q	5
	G	T	I	6
	L	O	S	7

1<sup>st</sup>-level candidates

Addr	Y	A	D	M						
H0	5	5	3	2						
H1	4	2	4	5						
H2	0	1	7	0						

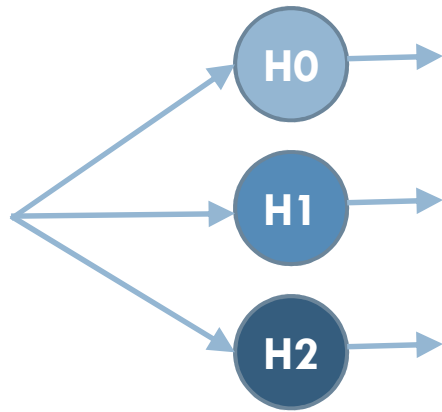
# ZCache Replacement



	Way 0	Way 1	Way 2	
	U	V	M	0
	F	C	X	1
	P	K	H	2
	B	E	R	3
	N	D	J	4
	A	Z	Q	5
	G	T	I	6
	L	O	S	7

Addr	1 <sup>st</sup> -level candidates				2 <sup>nd</sup> -level candidates					
	Y	A	D	M	B	K	X	P	Z	S
H0	5	5	3	2						
H1	4	2	4	5						
H2	0	1	7	0						

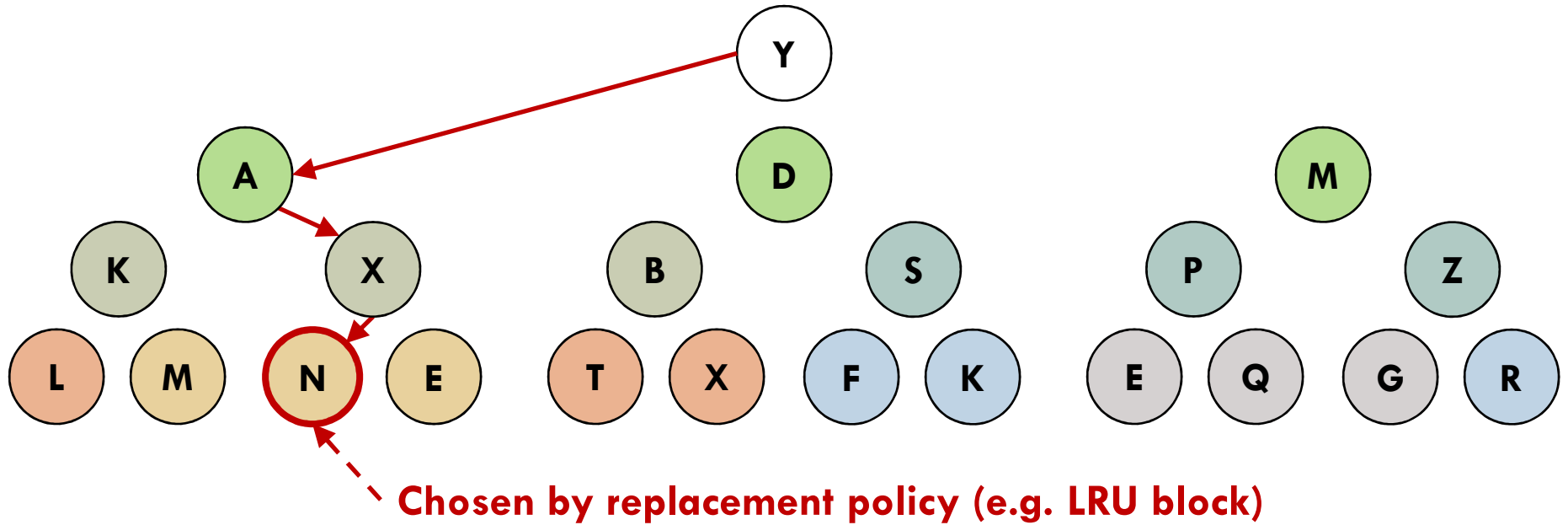
# ZCache Replacement



	Way 0	Way 1	Way 2	
	U	V	M	0
	F	C	X	1
	P	K	H	2
	B	E	R	3
	N	D	J	4
	A	Z	Q	5
	G	T	I	6
	L	O	S	7

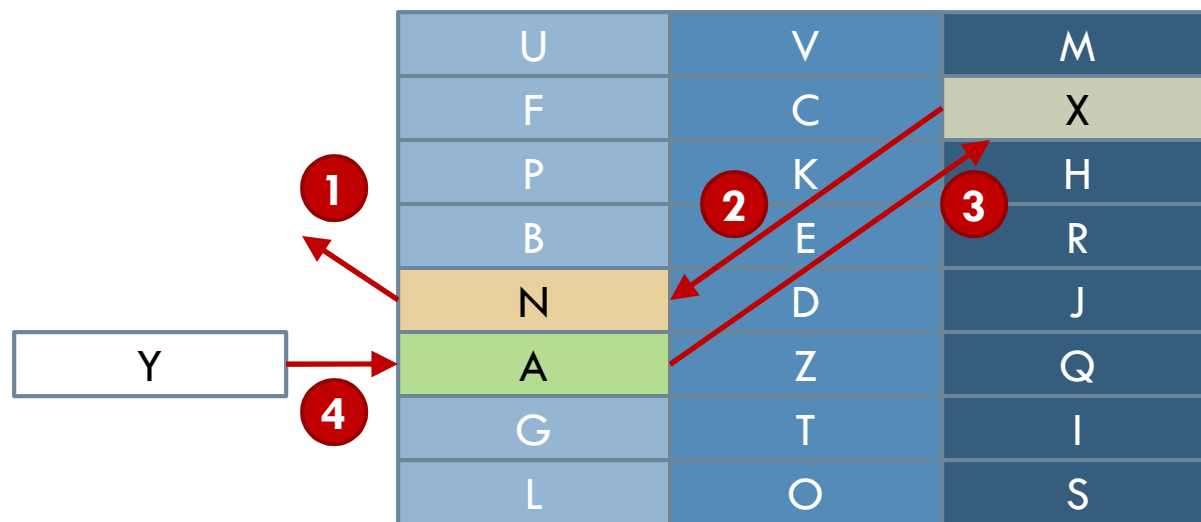
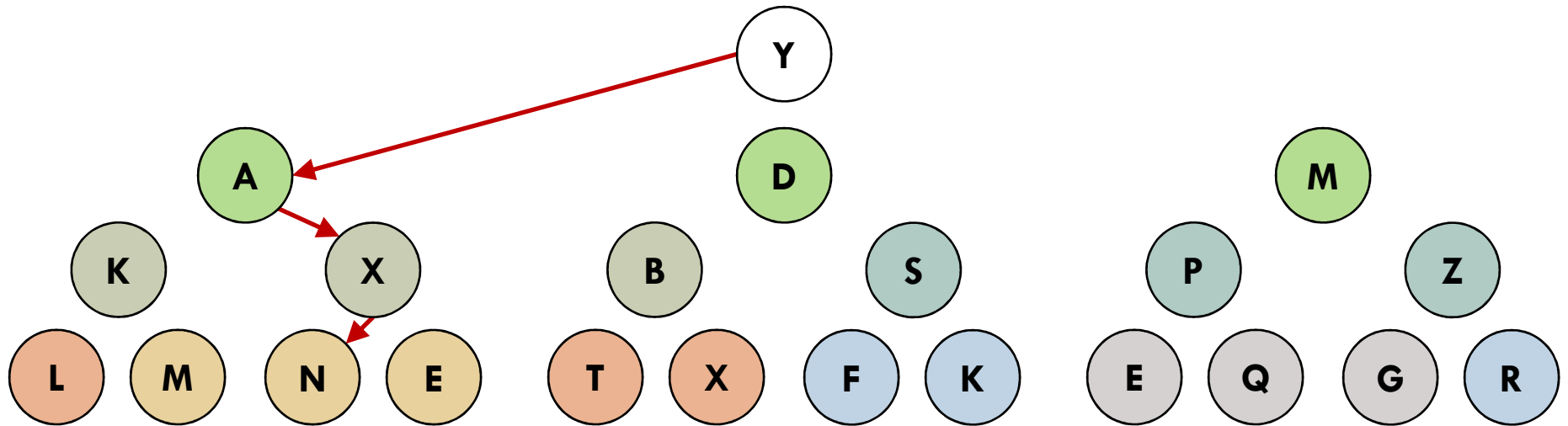
Addr	Y	A	D	M	B	K	X	P	Z	S
H0	5	5	3	2	3	7	4	2	6	1
H1	4	2	4	5	6	2	3	3	5	2
H2	0	1	7	0	1	0	1	5	3	7

# ZCache Replacement



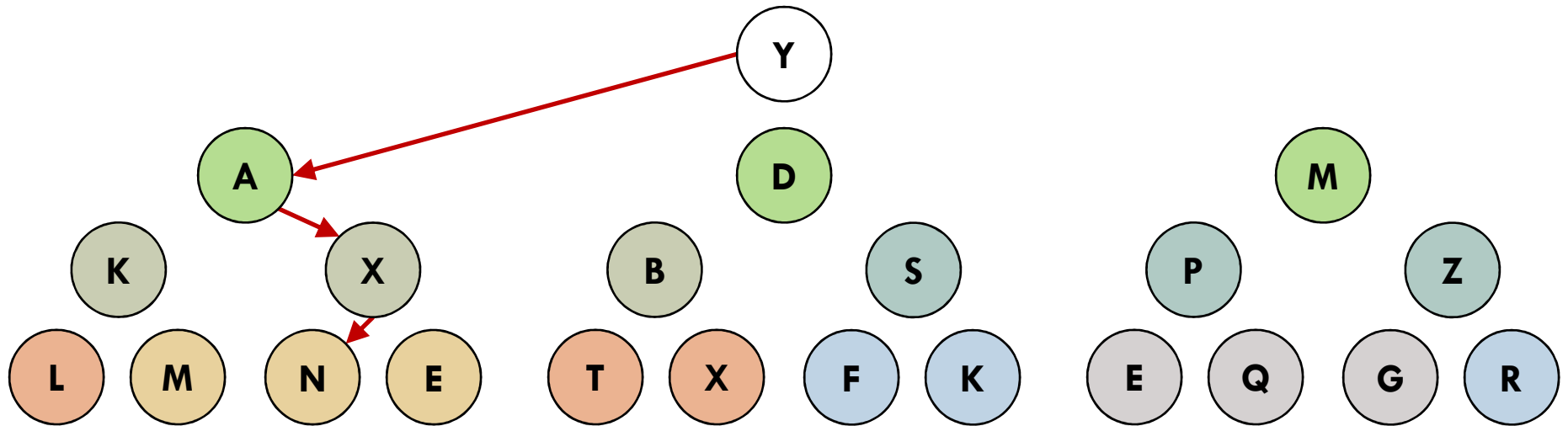
Addr	Y	A	D	M	B	K	X	P	Z	S
H0	5	5	3	2	3	7	4	2	6	1
H1	4	2	4	5	6	2	3	3	5	2
H2	0	1	7	0	1	0	1	5	3	7

# ZCache Replacement





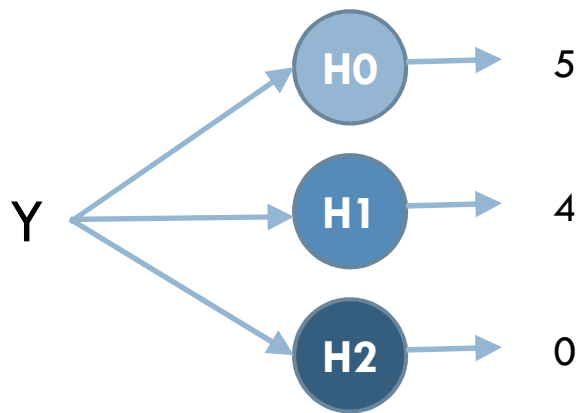
# ZCache Replacement



U	V	M
F	C	A
P	K	H
B	E	R
X	D	J
Y	Z	Q
G	T	I
L	O	S

# ZCache Replacement

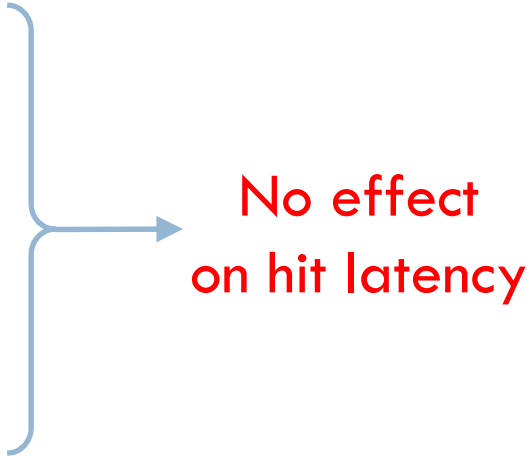
- Hits always take a single lookup



Way 0	Way 1	Way 2	
U	V	M	0
F	C	A	1
P	K	H	2
B	E	R	3
X	D	J	4
Y	Z	Q	5
G	T	I	6
L	O	S	7



# ZCache Implementation Overview

- Replacements take place:
    - ▣ Off the critical path
    - ▣ Concurrently with other operations
    - ▣ Walk accesses are pipelined
    - ▣ Do not saturate tag bandwidth in practice
  - Energy per miss mostly determined by walk
    - ▣ Similar to set-associative cache of same associativity
  - Cheap to implement
    - ▣ SRAM with 10s of bits to track candidates
    - ▣ Leverages existing MSHRs
  - See paper for more details
- 
- No effect on hit latency

# Number of Candidates

- An L-level walk on a W-way zcache gets R candidates:

$$R = W \cdot \sum_{n=0}^L (W - 1)^n$$

L \ W	2	3	4	8
0	2	3	4	8
1	4	9	16	64
2	6	21	52	456

- Few ways ( $W=4$ ) give many candidates with shallow walks
- Ratio of tag bandwidth vs bandwidth of next level limits number of candidates

# Outline

- Introduction
- ZCache
- Analytical Framework
- Evaluation

# An Analytical Associativity Framework

- Comparing associativity across cache designs is hard
  - ▣ Ways do not mean much
  - ▣ Conflict misses are workload and architecture-specific
- Goals
  - ▣ Find a general way to characterize associativity
  - ▣ Analyze what determines the performance of a zcache

# General Cache Model

- Cache array:
  - ▣ Holds tags and data
  - ▣ Implements associative lookup by address
  - ▣ On a replacement, gives list of replacement candidates
  - ▣ Model assumes nothing about array organization
- Replacement policy: Maintains a **global rank** of which cache blocks to replace
  - ▣ All policies conceptually do (LRU, LFU, OPT, ...)
  - ▣ Implementation does not need to

# Associativity Distribution

- Eviction priority: Rank of a block given by the replacement function, normalized to  $[0,1]$ 
  - ▣ Higher is better to evict
- Associativity distribution: Probability distribution of the eviction priorities of evicted blocks
  - ▣ Higher associativity  $\leftrightarrow$  distribution more skewed towards 1.0
  - ▣ Measures how well the array does, not the replacement policy
    - For good performance, replacement policy also needs to do a good job!



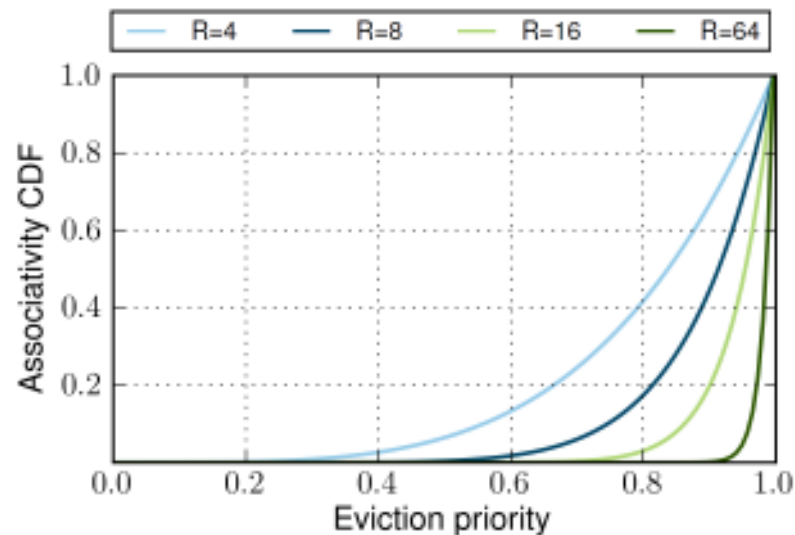
# Uniformity Assumption

- If the cache array gives  $R$  replacement candidates with uniformly distributed priorities,

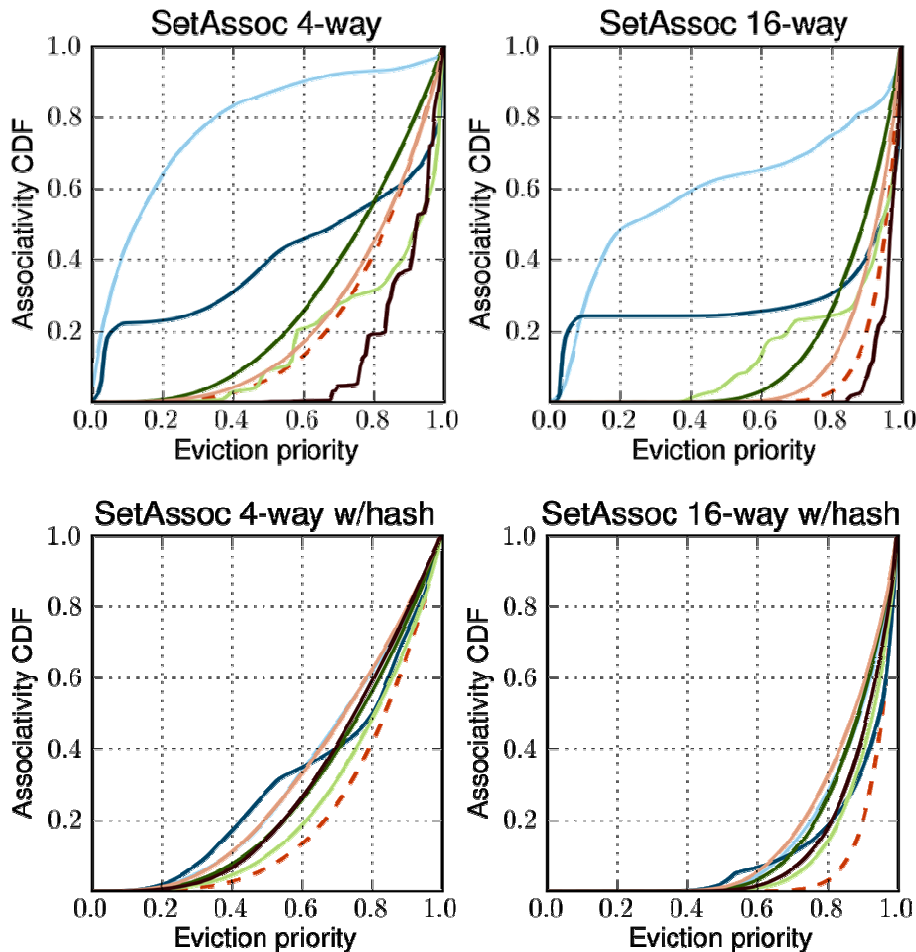
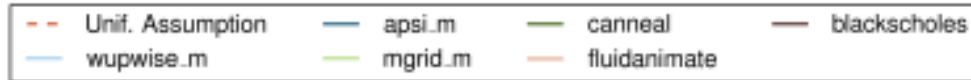
$$E_1, \dots, E_R \sim i.i.d. \ U[0,1]$$

$$A = \max \{E_1, \dots, E_R\}$$

$$F_A(x) = P(A \leq x) = x^R, x \in [0,1]$$



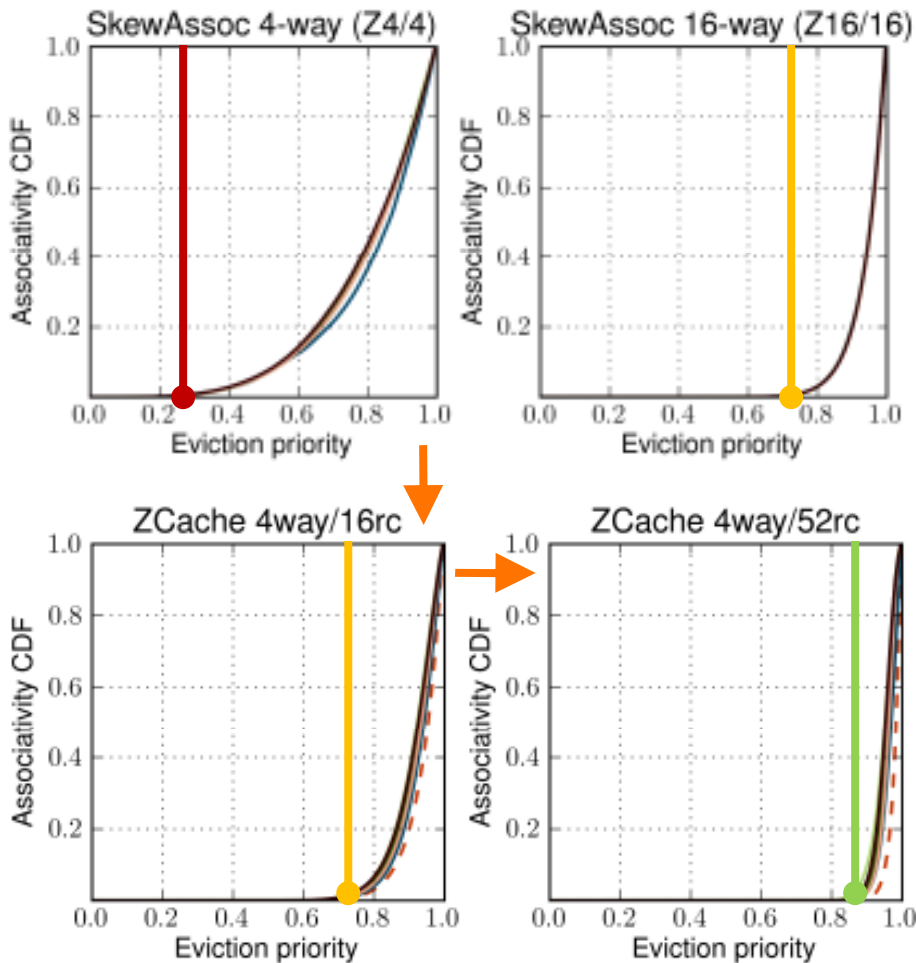
# Associativity Distributions in Practice



□ Set-associative caches do significantly worse than UA

□ Hashing ( $H_3$ ) improves associativity, but still sensibly worse than UA

# Associativity Distributions for ZCaches



- Skew-associative caches (1-level zcaches) are very close to UA
- Increasing candidates but not ways still yields distrib very close to UA

# Analytical Framework: Conclusions

- In caches with good hashing, the number of replacement candidates  $R$  determines associativity
- ZCaches provide large number of candidates with few ways → **Decouple ways and associativity**

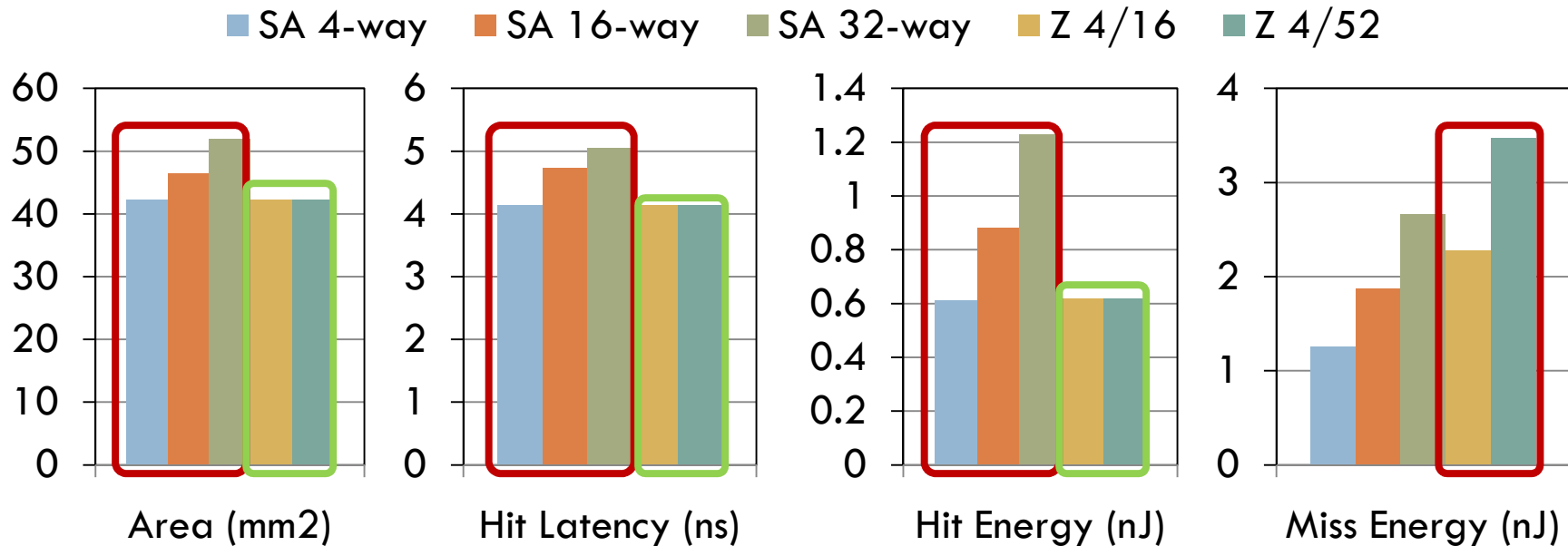
# Outline

- Introduction
- ZCache
- Analytical Framework
- **Evaluation**

# Methodology

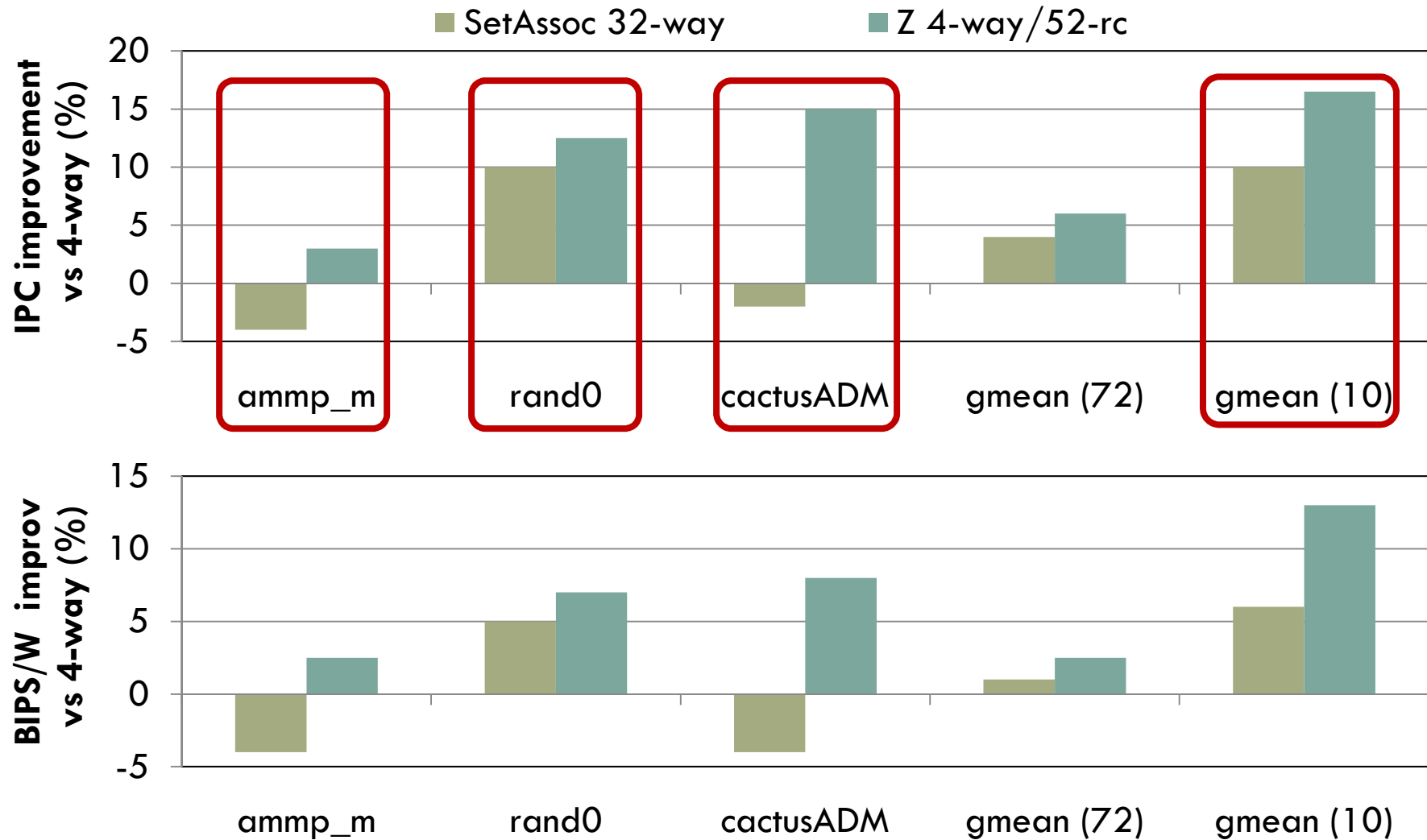
- Infrastructure:
  - CACTI-based models for cache cost estimates
  - McPAT for full-CMP area, power estimations
  - Microarchitectural simulation with Pin-based simulator
- Target system:
  - 32 in-order x86-64 cores (single-issue, 2GHz, 32KB I/D L1s)
  - Fully shared L2, 8MB, 8 1MB banks (set-assoc/zcache)
  - All L2 caches use hashing ( $H_3$ )
- 72 workloads:
  - Multithreaded: PARSEC, SPECOMP
  - Multiprogrammed: SPEC CPU2006
- See paper for more details

# Cache Costs



- Each design is optimized for area\*latency\*energy
- ZCaches:
  - ▣ Retain hit area, hit latency, hit energy of a 4-way SA cache
  - ▣ Energy per miss comparable to similarly-associative SA cache

# Performance and Energy-Efficiency



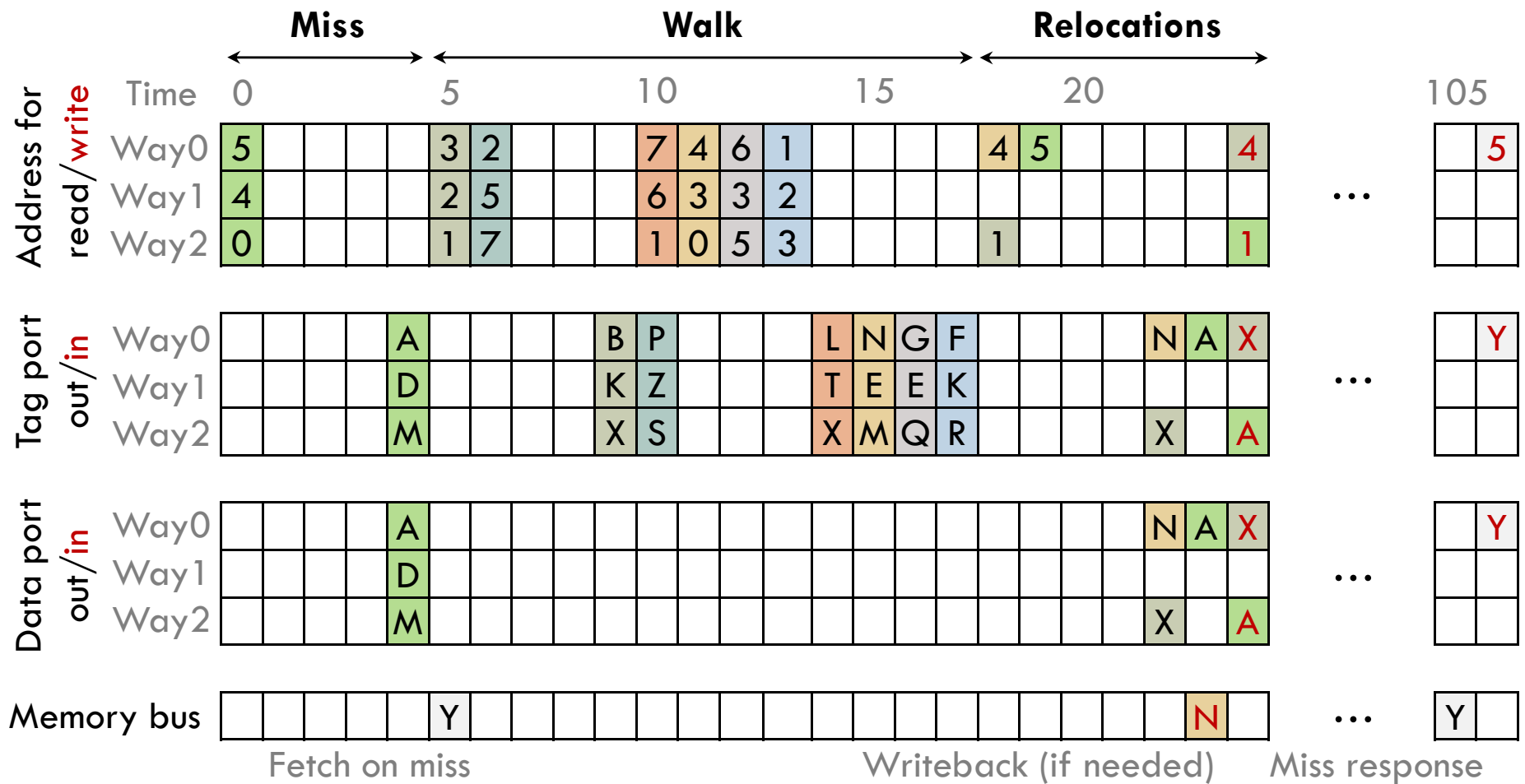


# Conclusions

- ZCaches enable efficient highly-associative caches
  - Low number of ways
  - Associativity gained **by increasing replacement candidates**
  - Costs of high associativity (energy, tag bandwidth) paid only on misses
- Analytical framework shows that **replacement candidates determine associativity**

THANK YOU FOR  
YOUR ATTENTION  
QUESTIONS?

# Backup: Replacement Timeline



# Backup: LRU with coarse-grain timestamps

36

- 8-bit timestamp per tag
- Tag each block with a global timestamp counter
- Increment timestamp every  $k=5\%$  accesses
  - ▣ Wraparounds are rare

