LARGE CACHE DESIGN

Mahdi Nazm Bojnordi

Assistant Professor

School of Computing

University of Utah

THE UNIVERSITY OF UTAH

Overview

- Upcoming deadline
  - Feb. 3rd: project group formation
- This lecture
  - Gated Vdd/ cache decay, drowsy caches
  - Compiler optimizations
  - Cache replacement policies
  - Cache partitioning
  - Highly associative caches

Main Consumers of CPU Resources?

☐ A significant portion of the processor die is occupied by on-chip caches

Example: FX Processors

☐ Main problems in caches

  ❏ Power consumption

    ■ Power on many transistors

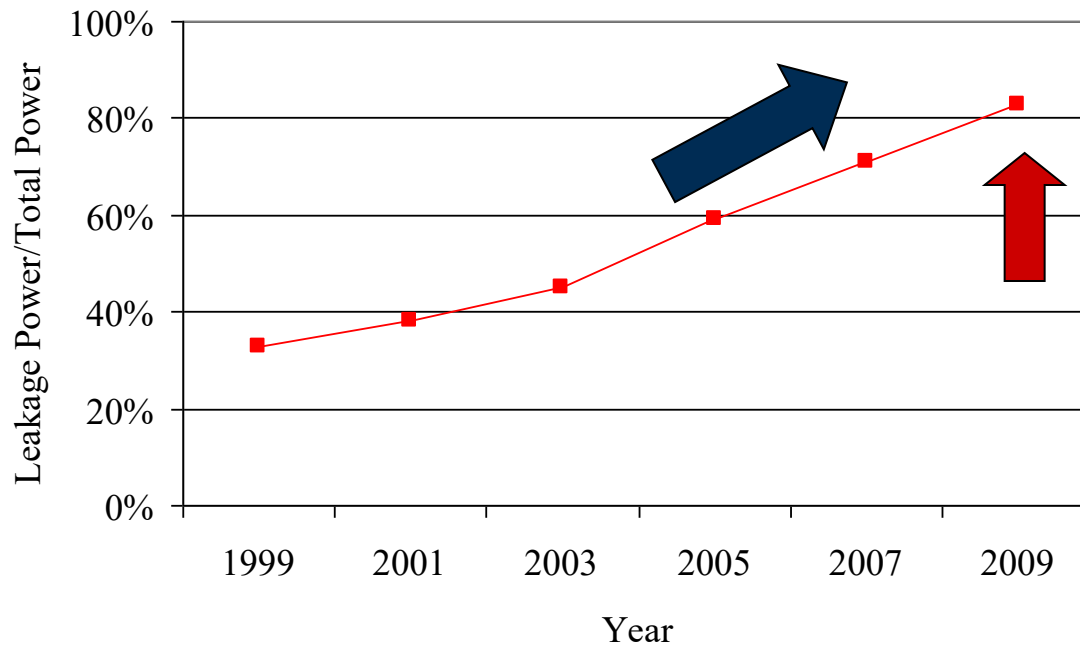 ❏ Reliability

    ■ Increased defect rate and errors
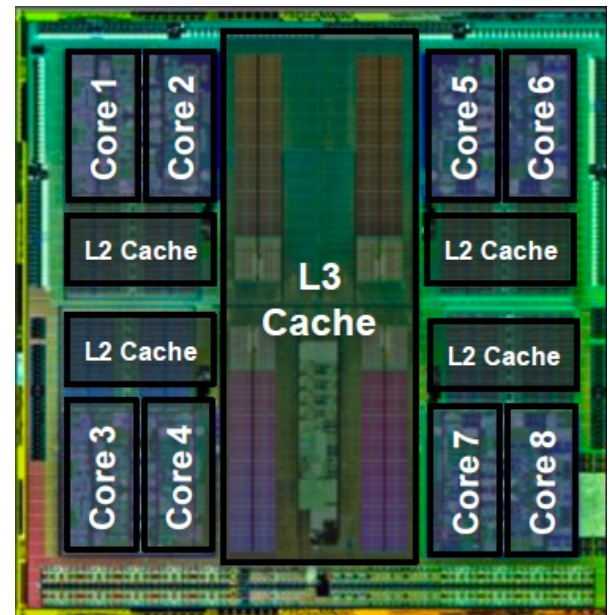
[source: AMD]

Leakage Power

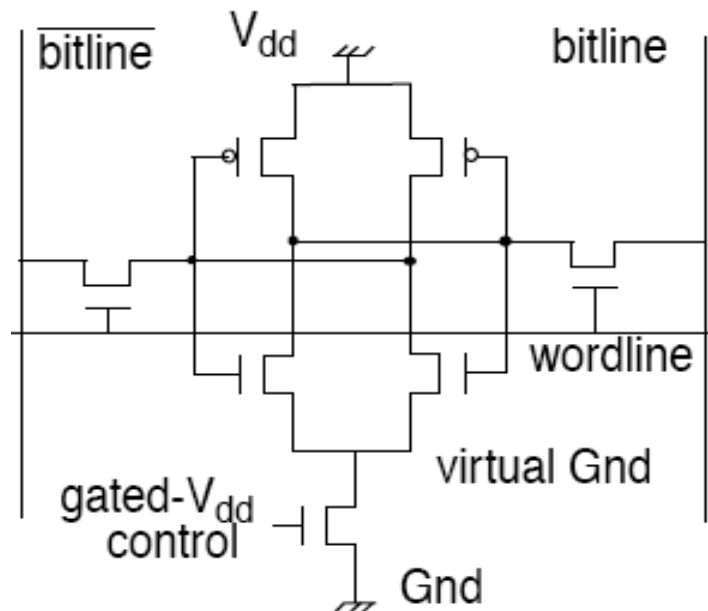☐ dominant source for power consumption as technology scales down

$$P_{leakage} = V \times I_{Leakage}$$
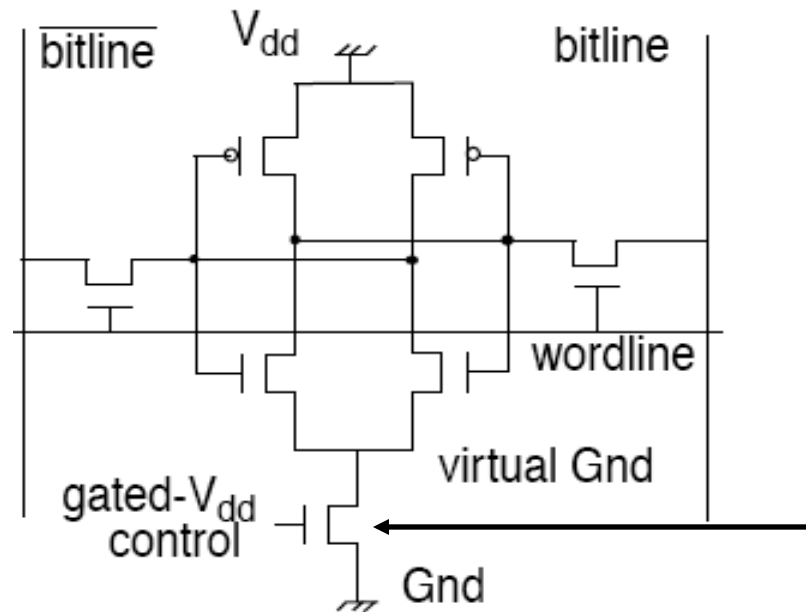

[source of data: ITRS]

Gated Vdd

□ Dynamically resize the cache (number of sets)
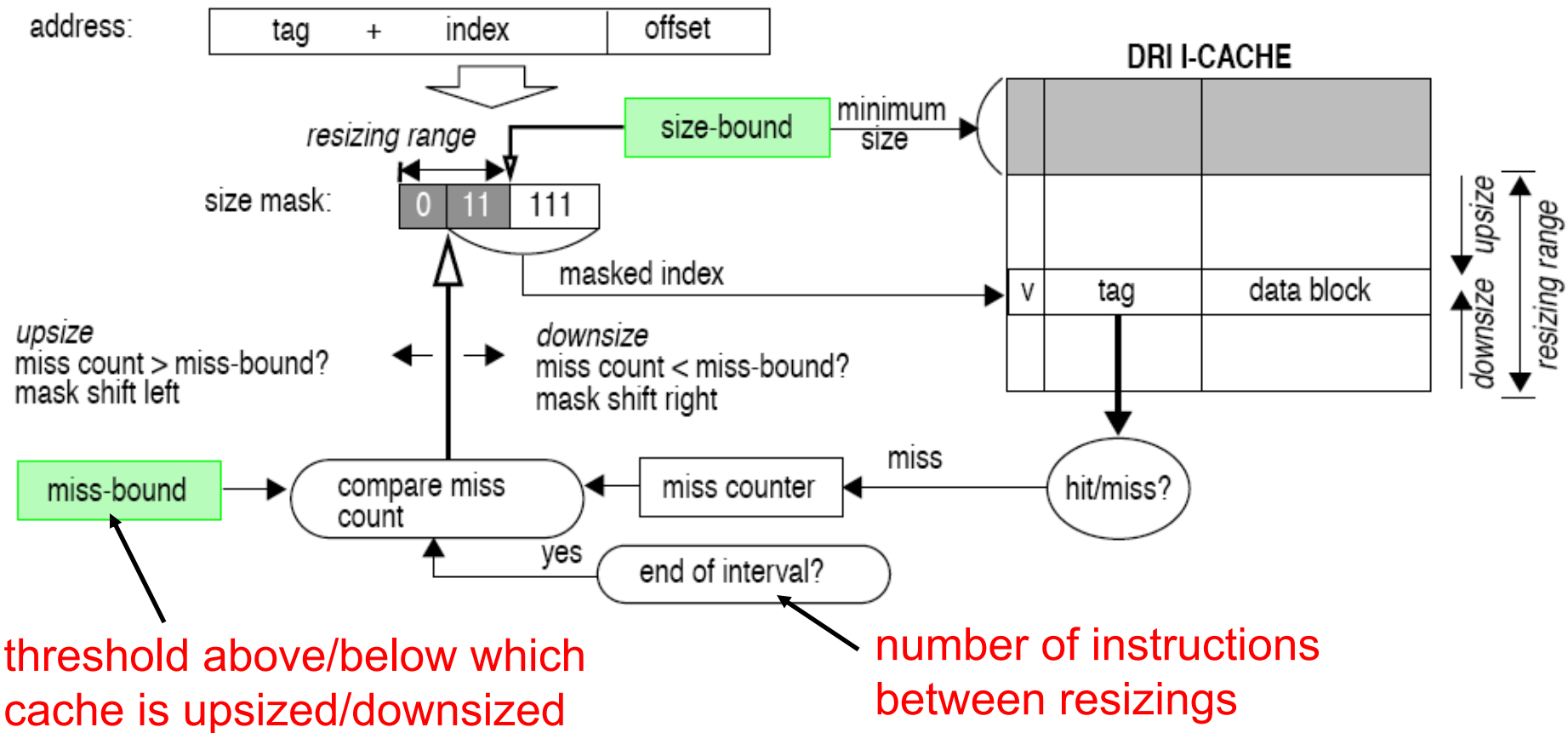

[Powell00]

Gated Vdd

- □ Dynamically resize the cache (number of sets)

- □ Sets are disabled by gating the path between Vdd and ground ("stacking effect")
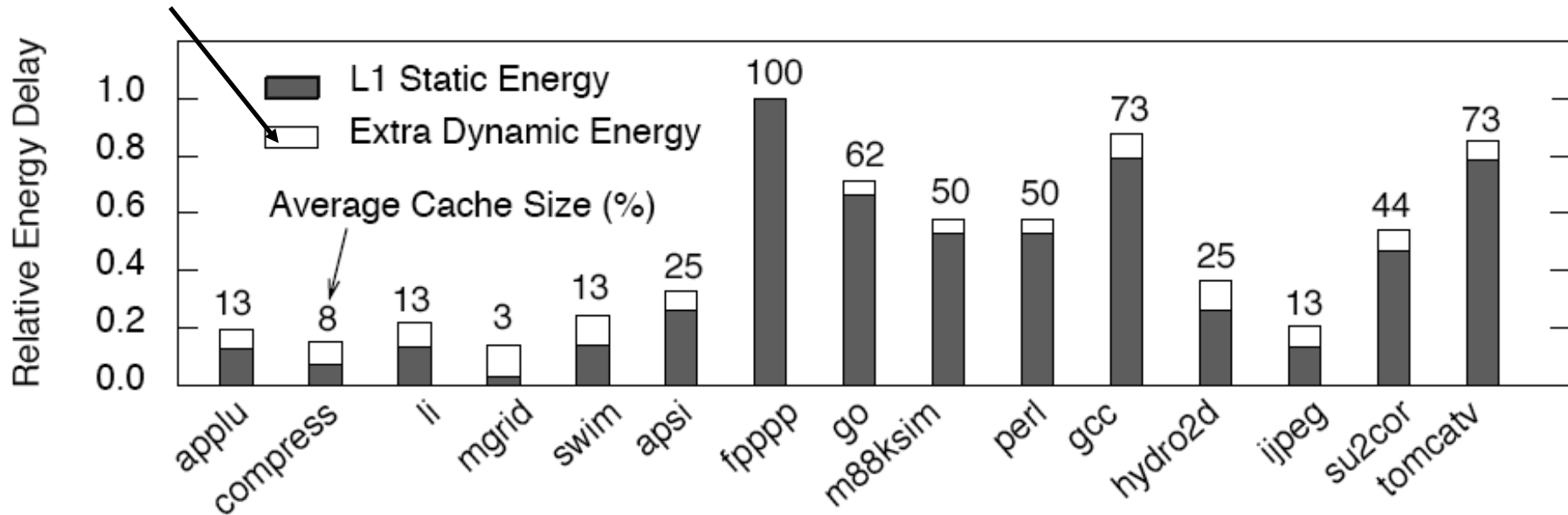

other possibilities, e.g., virtual Vdd (see paper)

shared among cells in same row (5% total area cost)
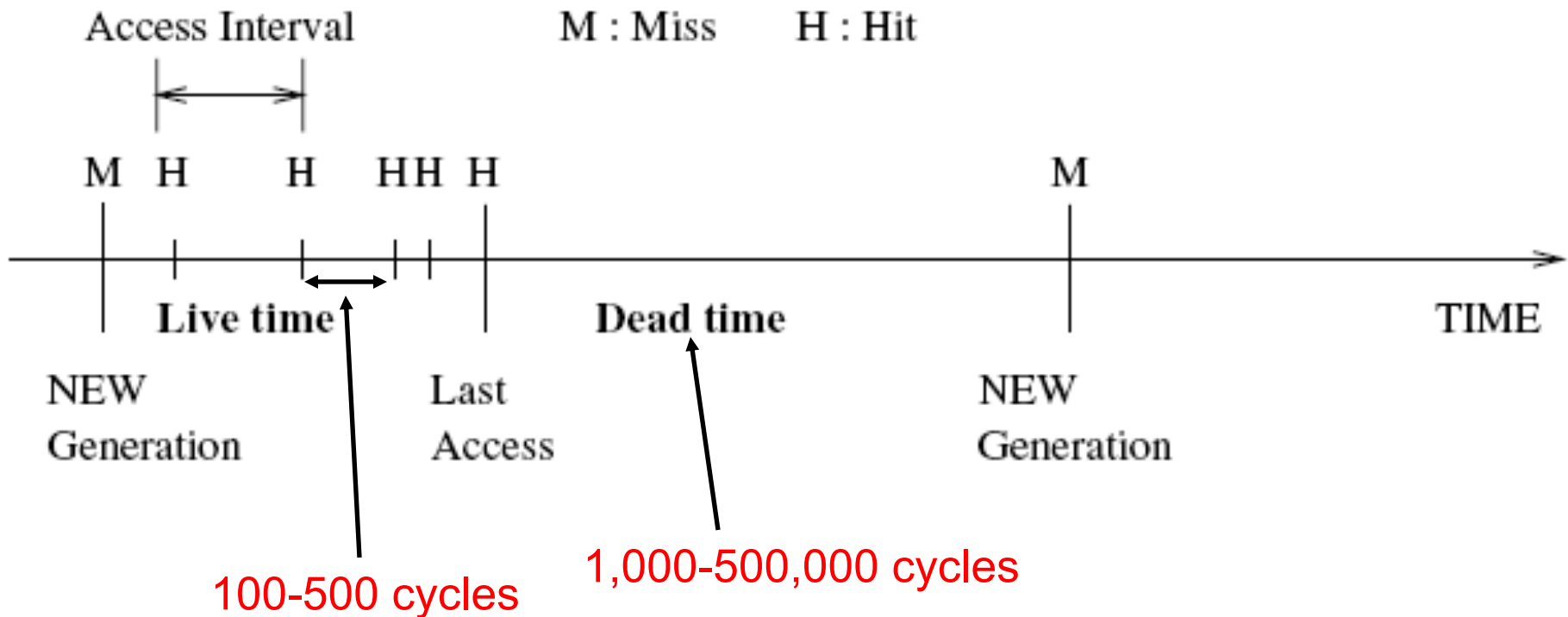
[Powell00]

Gated Vdd Microarchitecture

address:

| tag | + | index | offset |

DRI I-CACHE

resizing range

size-bound

minimum size

size mask: | 0 | 11 | 111 |

masked index

| v | tag | data block |

upsize
miss count > miss-bound?
mask shift left

downsize
miss count < miss-bound?
mask shift right

miss-bound

compare miss count

miss counter

miss

hit/miss?

yes

end of interval?

threshold above/below which cache is upsized/downsized

number of instructions between resizings

[Powell00]

Gated-Vdd I$ Effectiveness

due to additional misses

High mis-predication costs!

[Powell00]

Cache Decay

- Exploits generational behavior of cache contents

Access Interval   M : Miss   H : Hit

M H   H   HH H   M

Live time   Dead time   TIME

NEW   Last   NEW
Generation   Access   Generation

100-500 cycles

1,000-500,000 cycles

[Kaxiras01]

Cache Decay

☐ Fraction of time cache lines that are "dead"
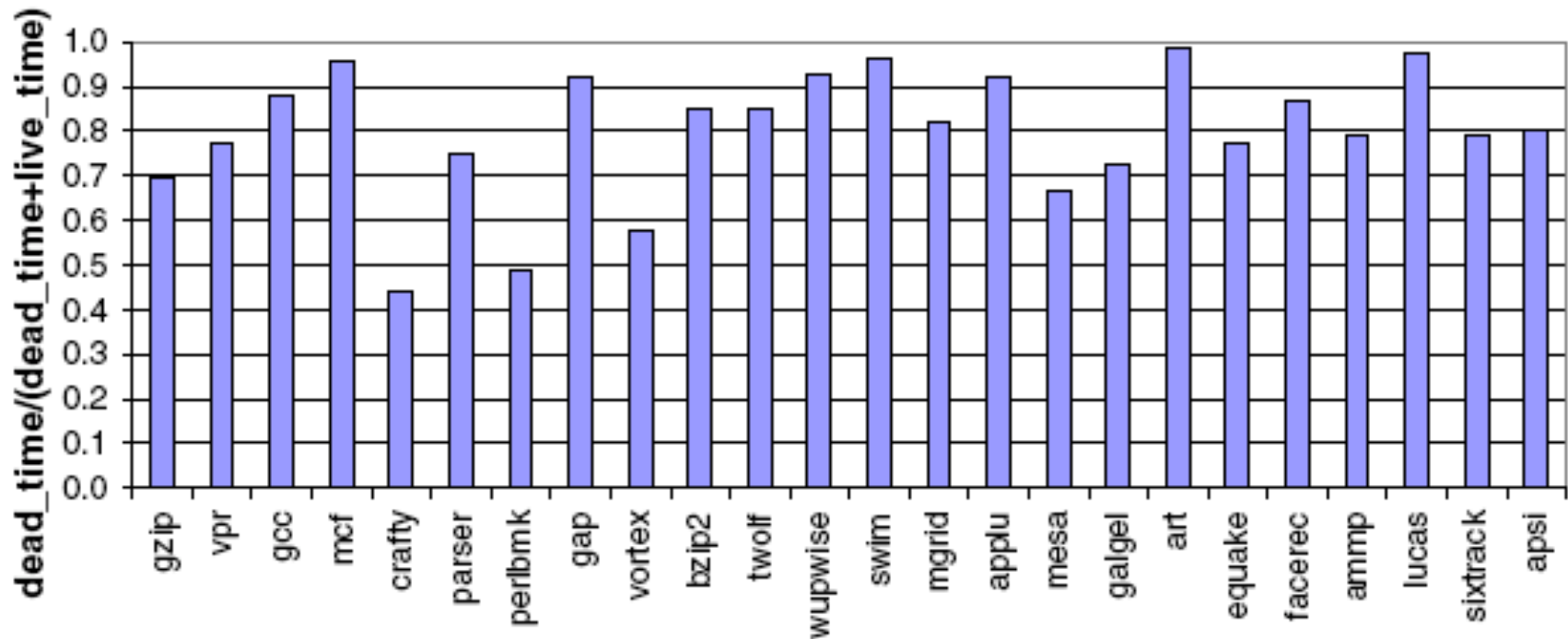
32KB L1 D-cache

[Kaxiras01]

Cache Decay Implementation

High mis-predication costs!

State Diagram for 2-bit (S1,S0), saturating, Gray-code counter with two inputs (WRD, T)

[Kaxiras01]
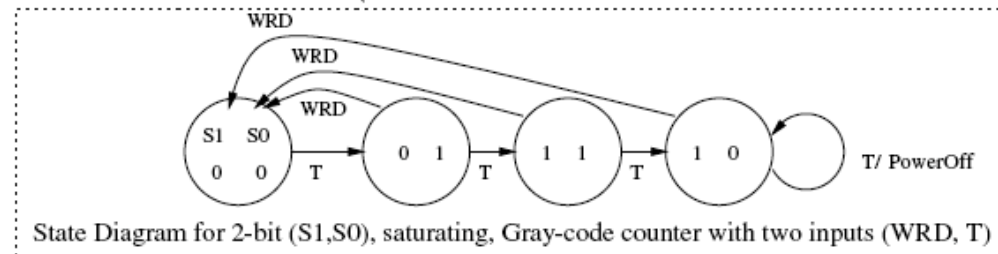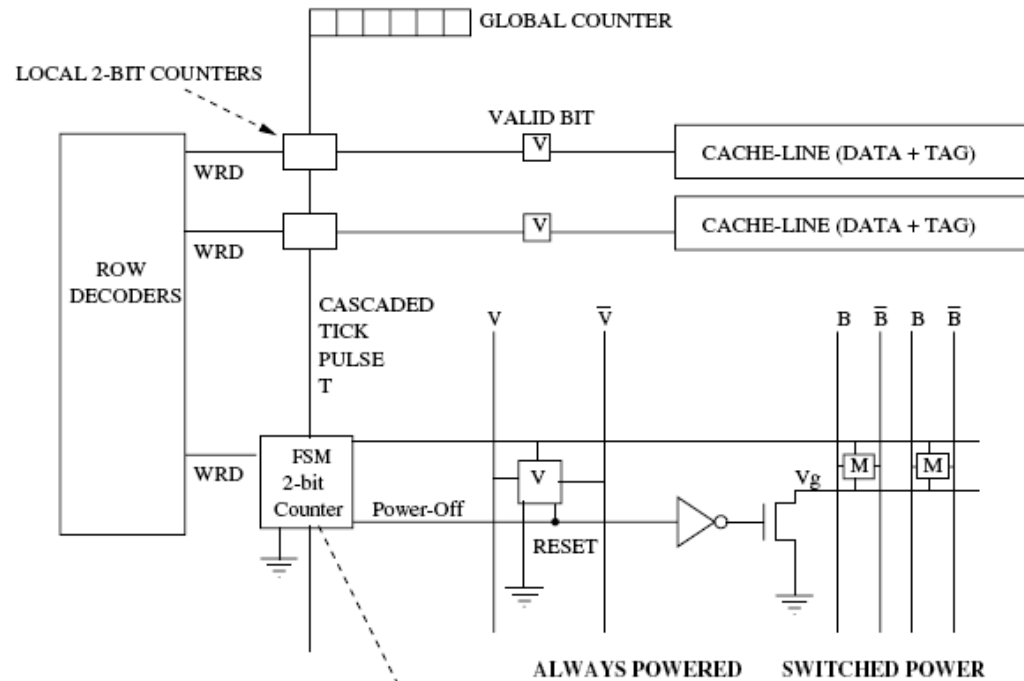
Drowsy Caches

- Gated-Vdd cells lose their state
  - Instructions/data must be refetched
  - Dirty data must be first written back

- By dynamically scaling Vdd, cell is put into a drowsy state where it retains its value
  - Leakage drops superlinearly with reduced Vdd ("DIBL" effect)
  - Cell can be fully restored in a few cycles
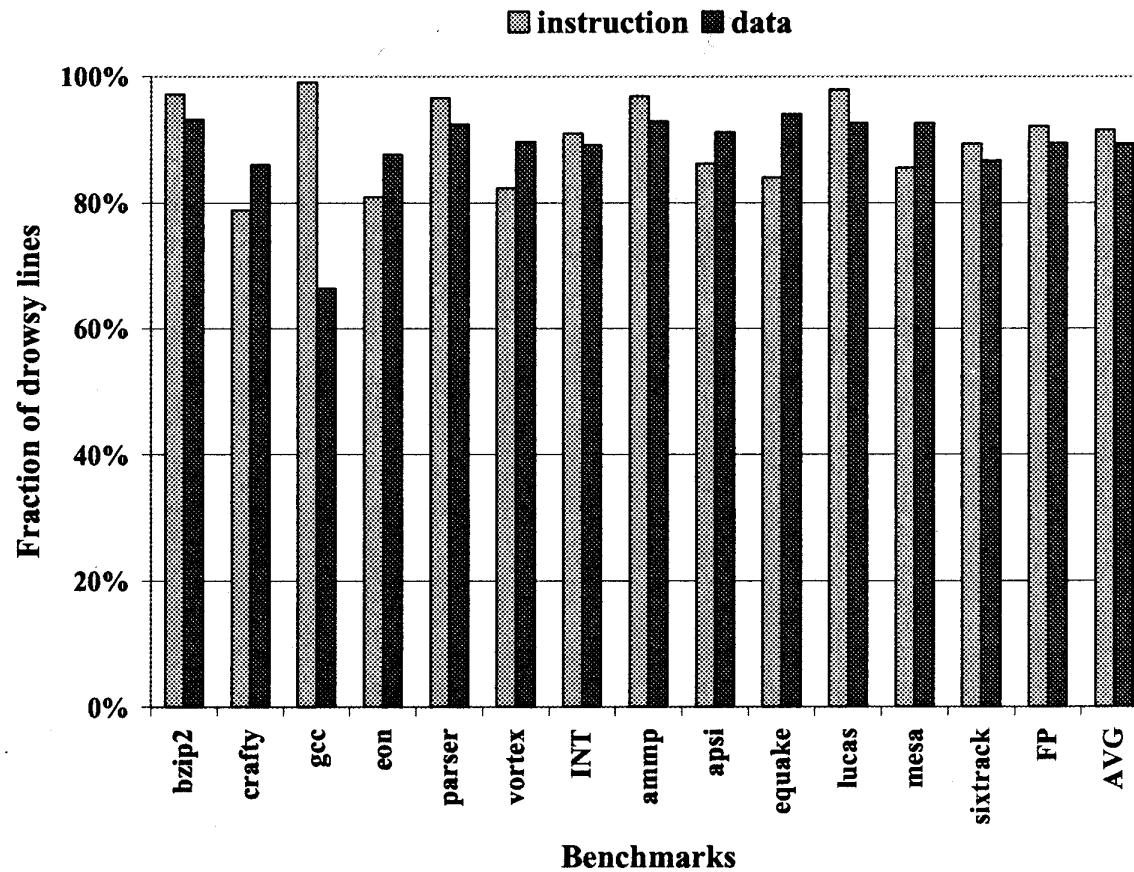 - Much lower misprediction cost than gated-Vdd, but noise susceptibility and less reduction in leakage

Drowsy Cache Organization
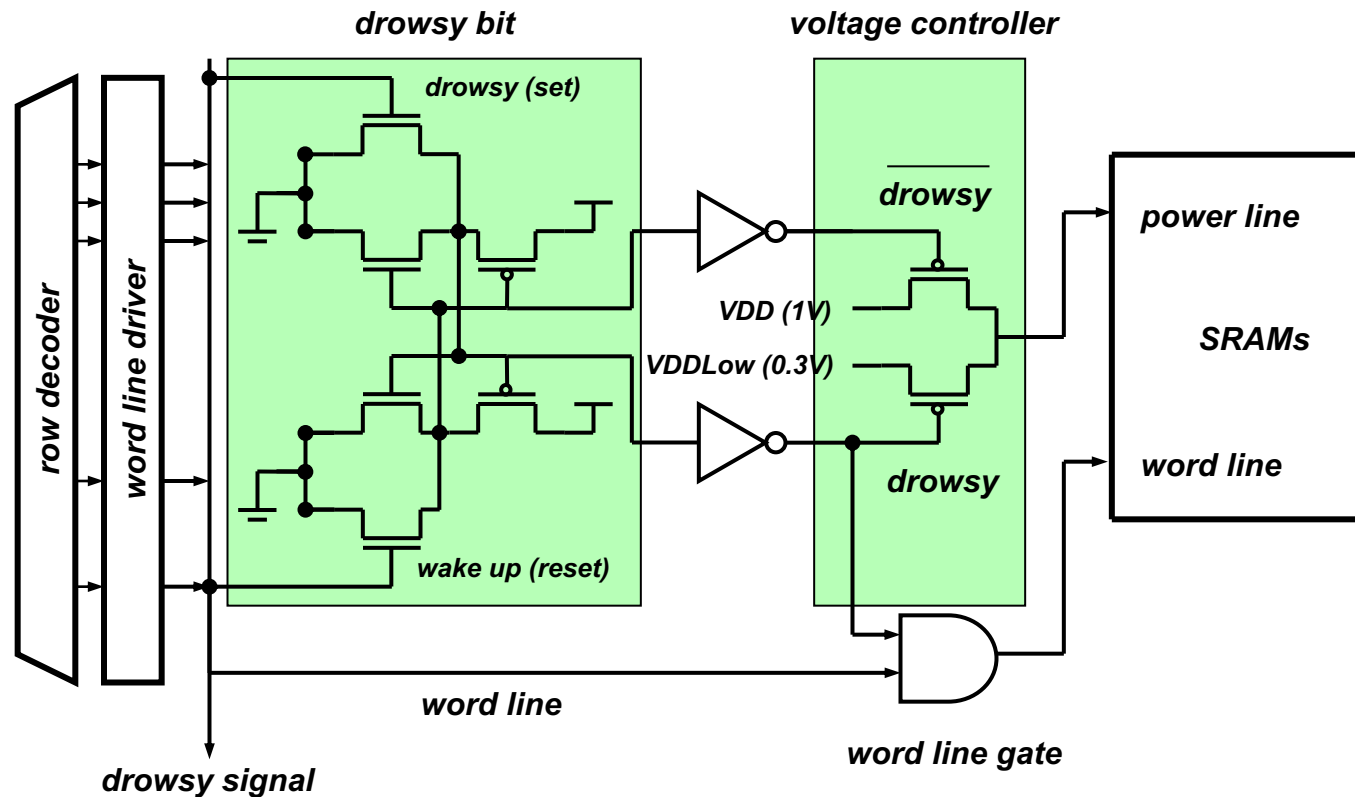
Keeps the contents (no data loss) *[Kim04]*

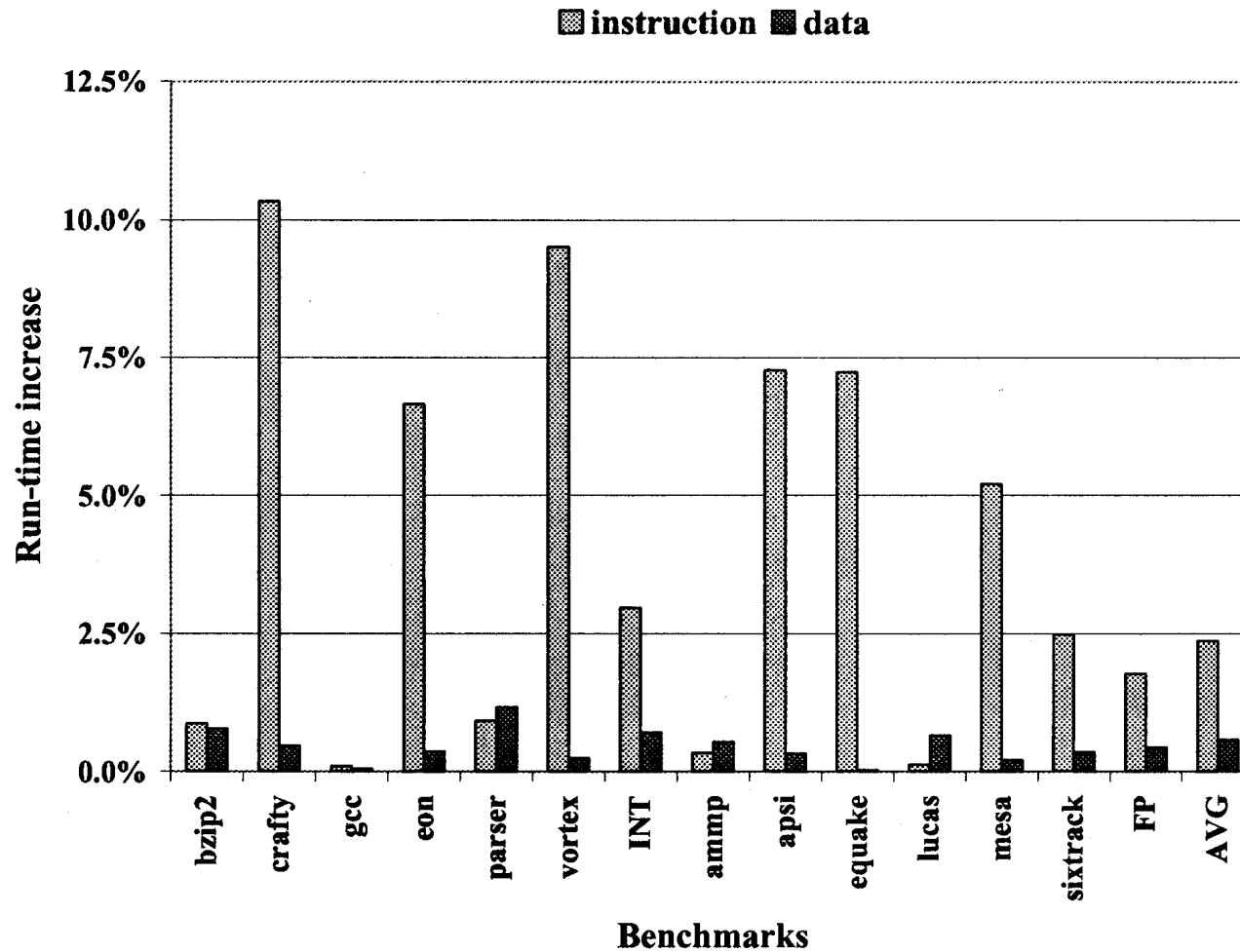# Drowsy Cache Effectivenes

32KB L1 caches   4K cycle drowsy period

[Kim04]

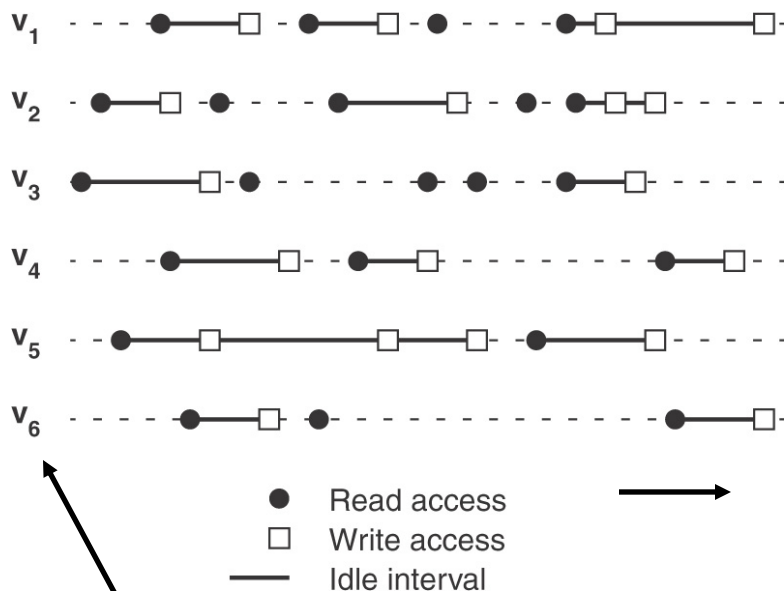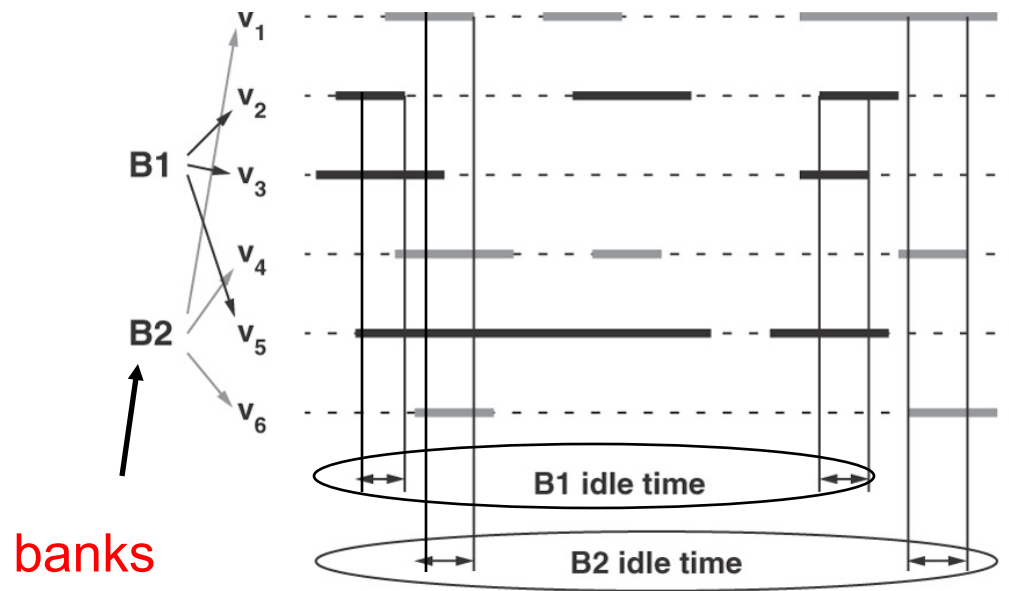# Drowsy Cache Performance Cost

[Kim04]

Software Techniques

Compiler-Directed Data Partitioning

☐ Multiple D-cache banks, each with sleep mode

☐ Lifetime analysis used to assign commonly idle data to the same bank

variables

banks

Compiler Optimizations

□ Loop Interchange

◻ Swap nested loops to access memory in sequential order

```
/* Before */                          /* After */
for (j = 0; j < 100; j = j+1)         for (i = 0; i < 5000; i = i+1)
    for (i = 0; i < 5000; i = i+1)        for (j = 0; j < 100; j = j+1)
        x[i][j] = 2 * x[i][j];                x[i][j] = 2 * x[i][j];
```
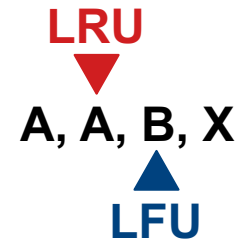
□ Blocking

◻ Instead of accessing entire rows or columns, subdivide matrices into blocks

◻ Requires more memory accesses but improves locality of accesses
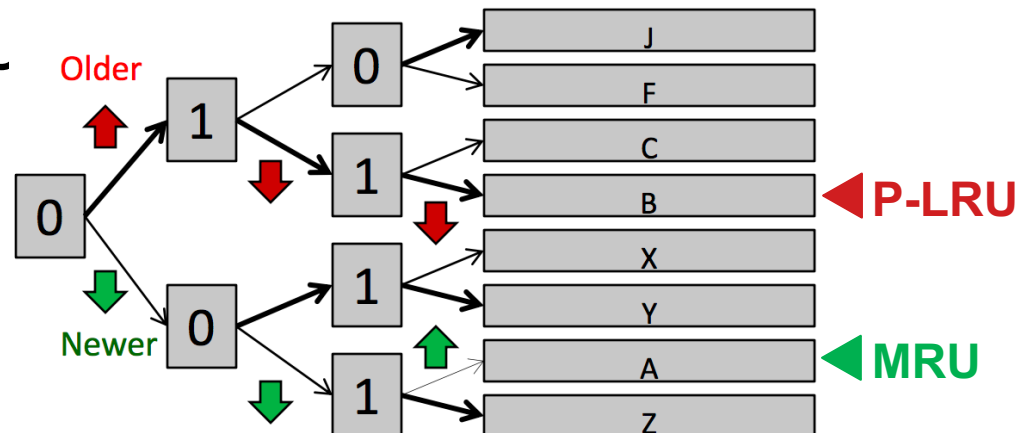
Replacement Policies

Basic Replacement Policies

- Least Recently Used (LRU)

- Least Frequently Used (LFU)

- Not Recently Used (NRU)

  - every block has a bit that is reset to 0 upon touch

  - a block with its bit set to 1 is evicted

  - if no block has a 1, make every bit 1

- Practical pseudo-LRU

LRU

A, A, B, X

LFU

Older

Newer

J

F

C

B   **◀ P-LRU**

X

Y

A   **◀ MRU**

Z

Common Issues with Basic Policies

□ Low hit rate due to cache pollution

  ◘ streaming (no reuse)

    ▪ A-B-C-D-E-F-G-H-I-…

  ◘ thrashing (distant reuse)

    ▪ A-B-C-A-B-C-A-B-C-…

□ A large fraction of the cache is useless – blocks that have serviced their last hit and are on the slow walk from MRU to LRU

Basic Cache Policies

- Insertion
  - Where is incoming line placed in replacement list?
- Promotion
  - When a block is touched, it can be promoted up the priority list in one of many ways
- Victim selection
  - Which line to replace for incoming line? (not necessarily the tail of the list)

Simple changes to these policies can greatly improve cache performance for memory-intensive workloads
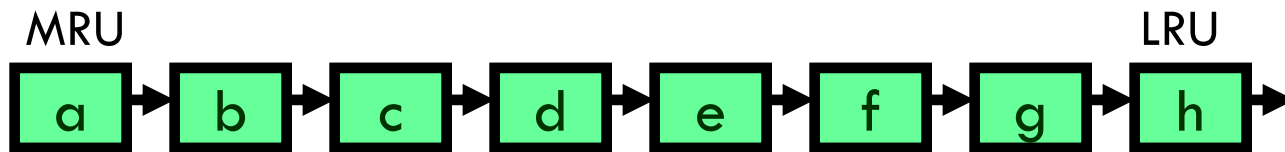
Inefficiency of Basic Policies

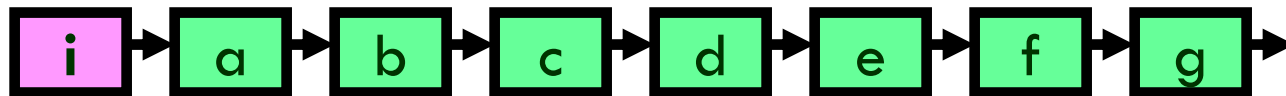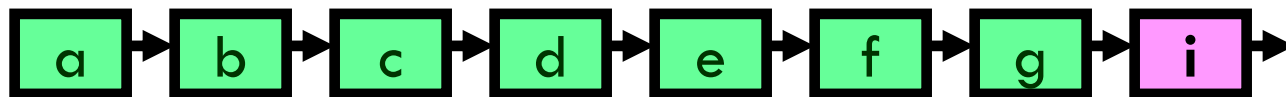☐ About 60% of the cache blocks may be dead on arrival (DoA)

[Qureshi'07]
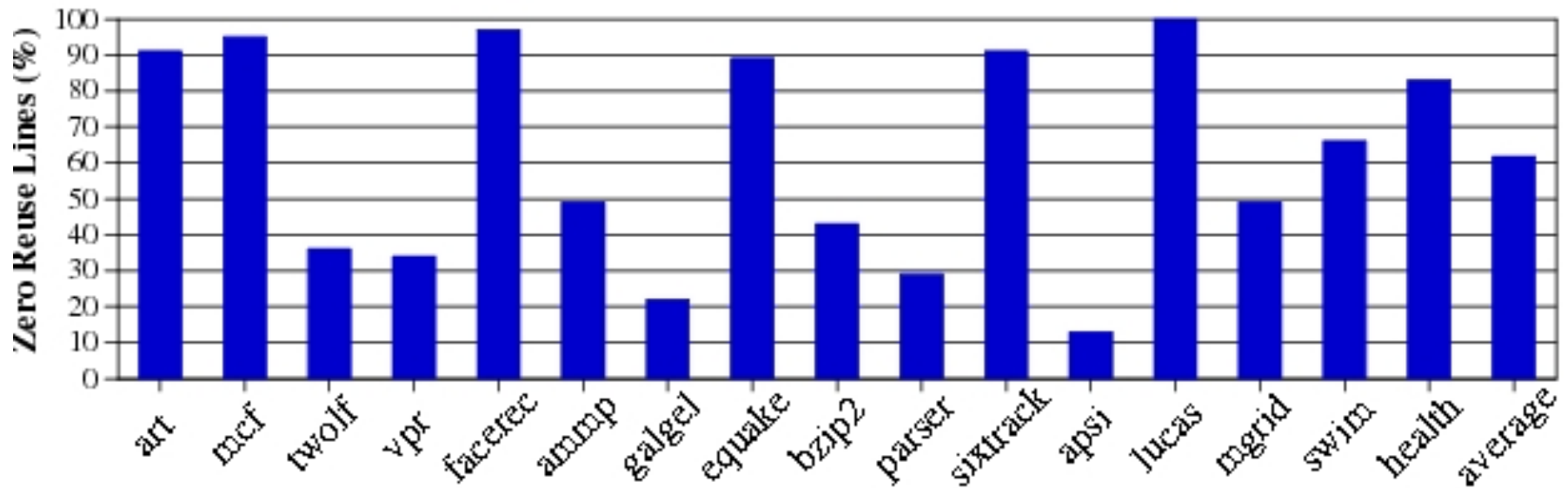
Adaptive Insertion Policies

☐ MIP: MRU insertion policy (baseline)

☐ LIP: LRU insertion policy

MRU   LRU

| a | b | c | d | e | f | g | h |

Traditional LRU places 'i' in MRU position.

| i | a | b | c | d | e | f | g |

LIP places 'i' in LRU position; with the first touch it becomes MRU.

| a | b | c | d | e | f | g | i |

[Qureshi'07]

Adaptive Insertion Policies

- LIP does not age older blocks
  - A, A, B, C, B, C, B, C, …

LRU   **MRU**
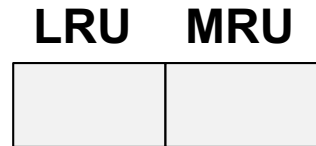
- BIP: Bimodal Insertion Policy
  - Let ε = Bimodal throttle parameter

```
if  ( rand() < ε )
      Insert at MRU position;
else
      Insert at LRU position;
```

[Qureshi'07]

Adaptive Insertion Policies

- There are two types of workloads: LRU-friendly or BIP-friendly
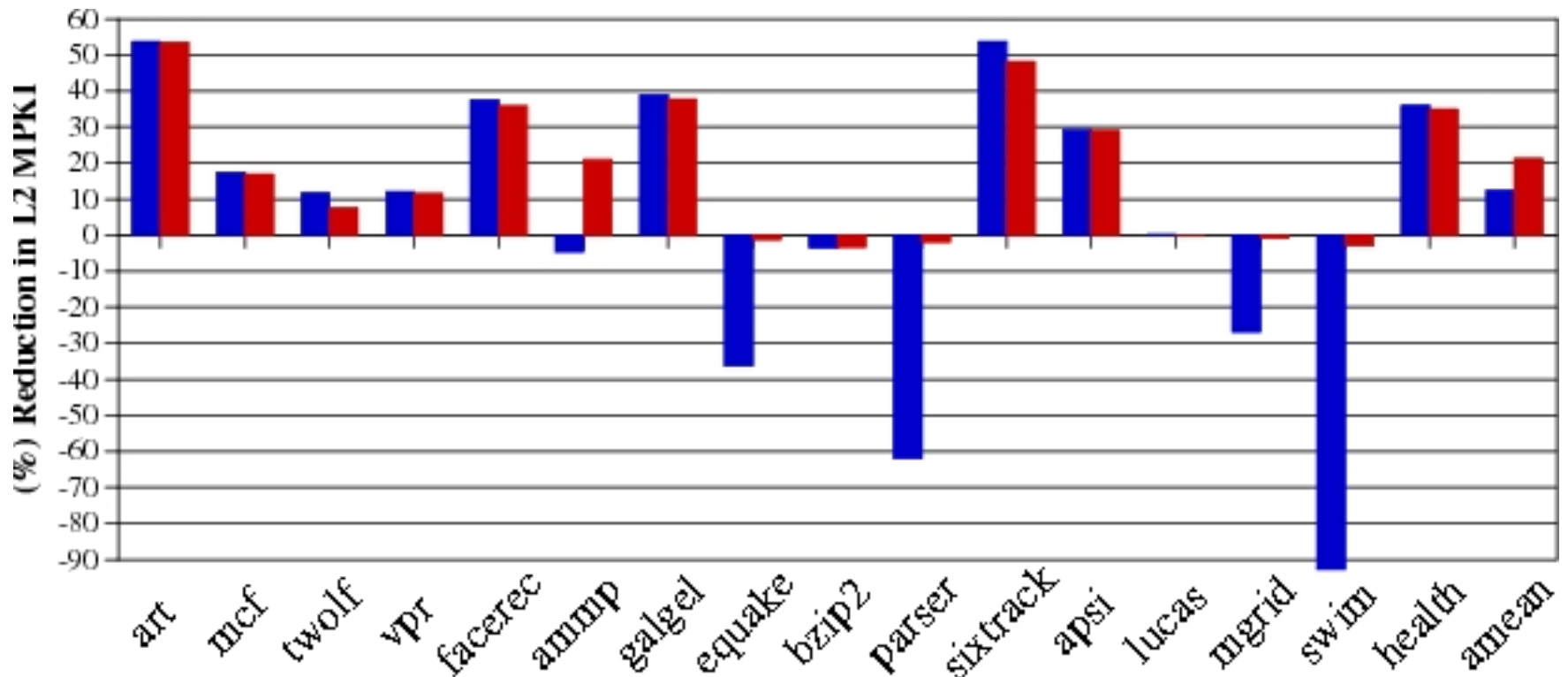
- DIP: Dynamic Insertion Policy
  - Set Dueling

Read the paper for more details.

LRU-sets	miss	+
BIP-sets		−
	miss	
Follower Sets		

n-bit cntr

MSB = 0?

YES
Use LRU

No
Use BIP
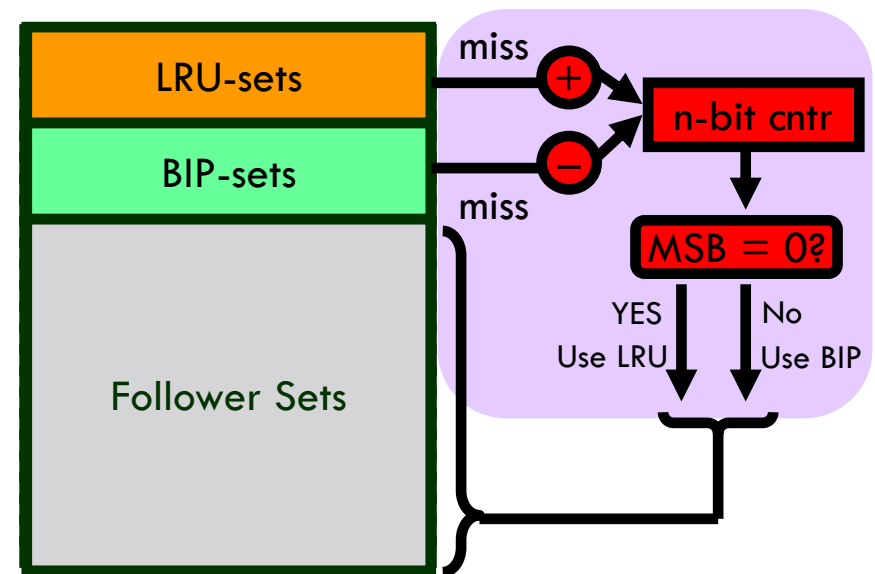
monitor ➜ choose ➜ apply
(using a single counter)

[Qureshi'07]

Adaptive Insertion Policies

☐ DIP reduces average MPKI by 21% and requires less than two bytes storage overhead


[Qureshi'07]

Re-Reference Interval Prediction

- Goal: high performing scan resistant policy
  - DIP is thrash-resistance
  - LFU is good for recurring scans
- Key idea: insert blocks near the end of the list than at the very end
- Implement with a multi-bit version of NRU
  - zero counter on touch, evict block with max counter, else increment every counter by one

Read the paper for more details.
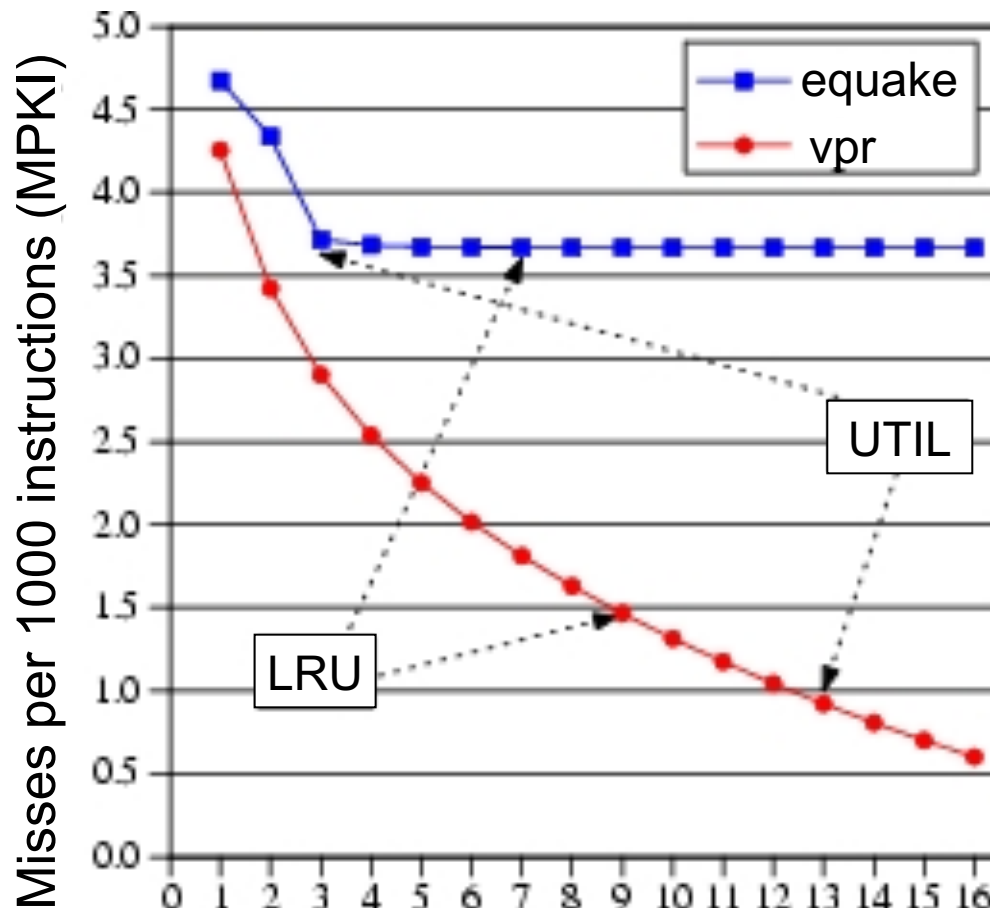
[Jaleel'10]

Shared Cache Problems

- A thread's performance may be significantly reduced due to an unfair cache sharing

- Question: how to control cache sharing?
  - Fair cache partitioning [Kim'04]
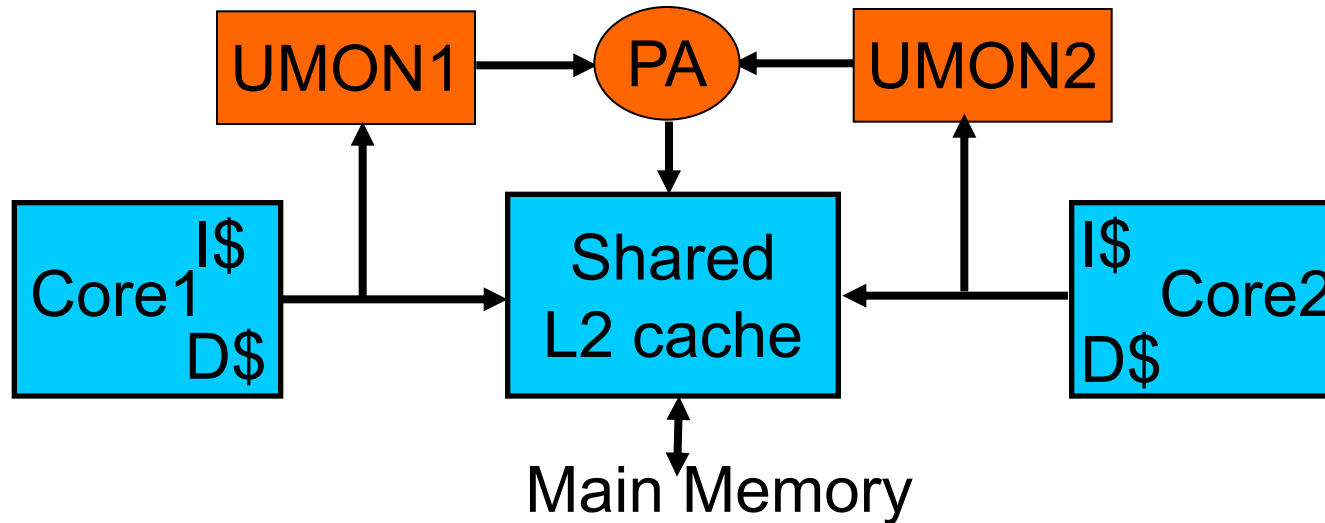 - Utility based cache partitioning [Qureshi'06]
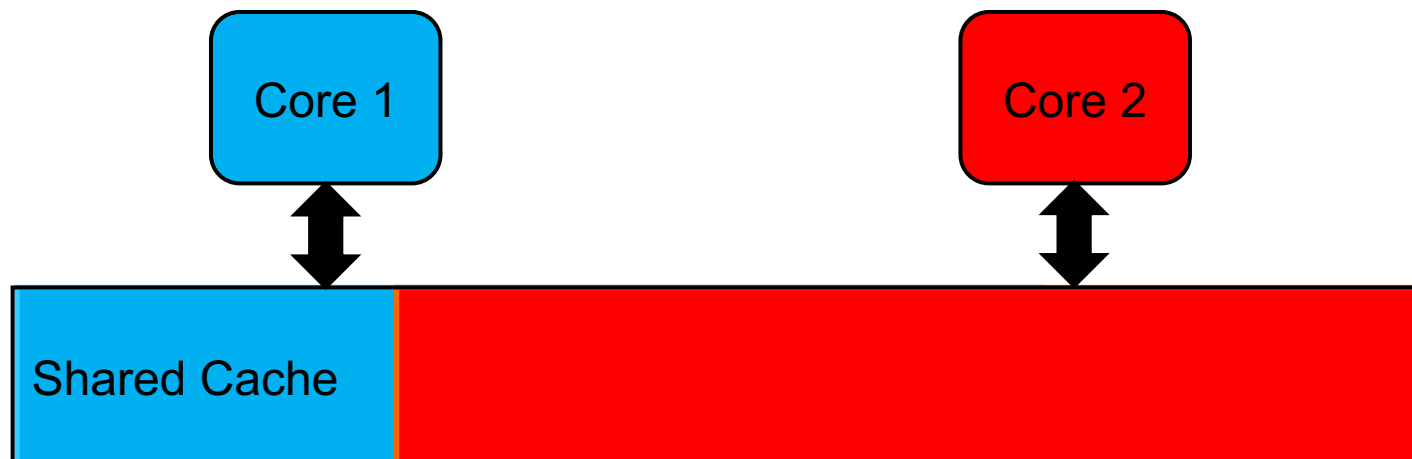
Utility Based Cache Partitioning

☐ Key idea: give more cache to the application that benefits more from cache

[Qureshi'06]

Utility Based Cache Partitioning
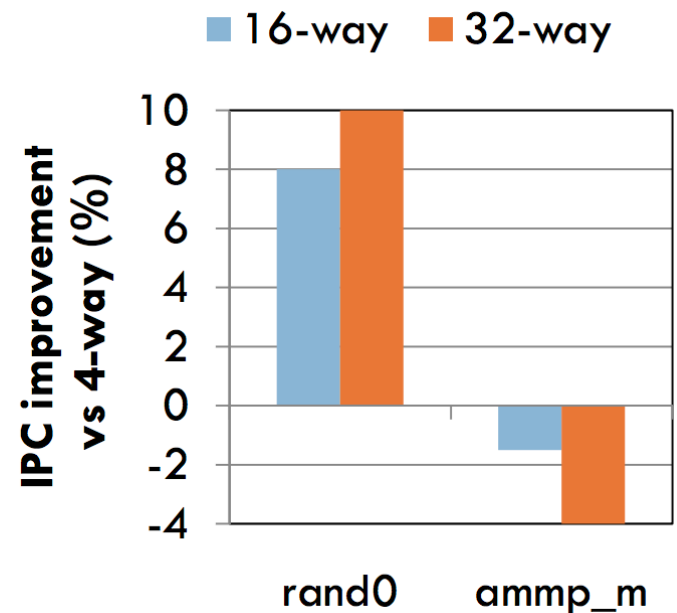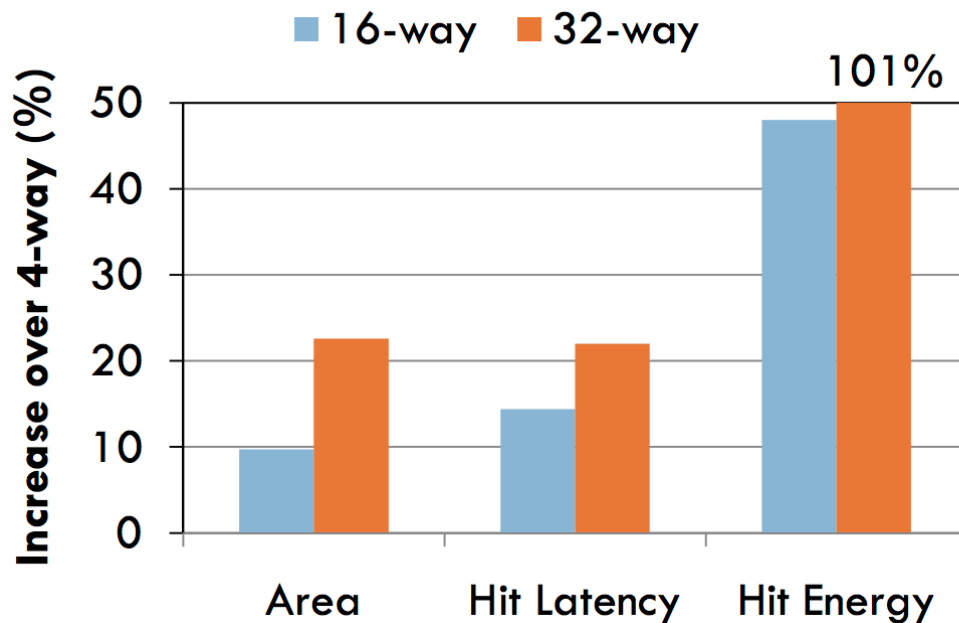

Three components:

❑ Utility Monitors (UMON) per core

❑ Partitioning Algorithm (PA)

❑ Replacement support to enforce partitions

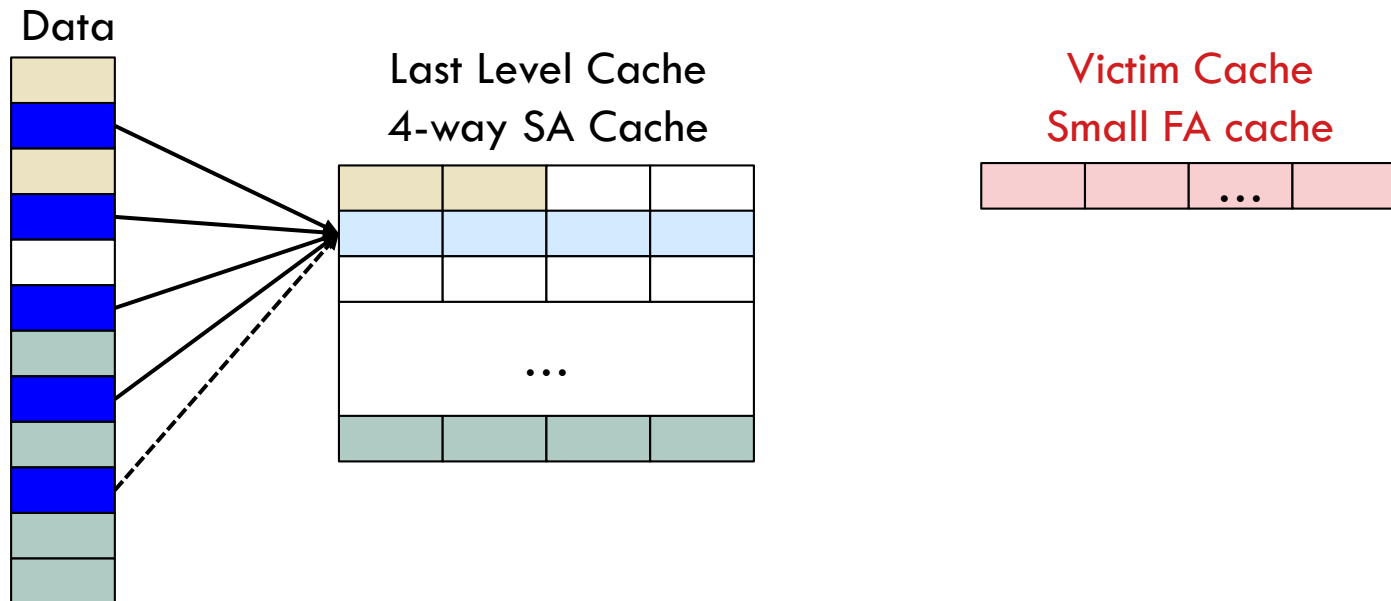*[Qureshi'06]*

Highly Associative Caches

☐ Last level caches have ~32 ways in multicores

  ☐ Increased energy, latency, and area overheads

[Sanchez'10]

Recall: Victim Caches

☐ Goal: to decrease conflict misses using a small FA cache

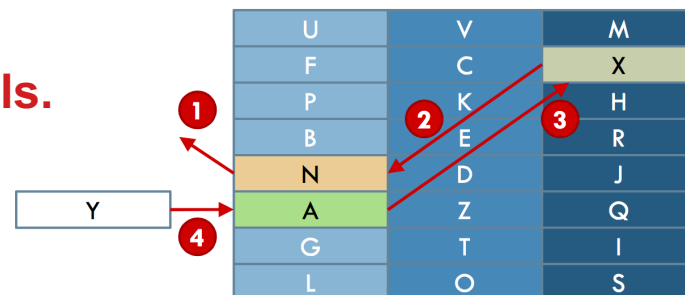**Can we reduce the hardware overheads?**

The ZCache

- Goal: design 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

- Skewed associative cache: each way has a different indexing function (in essence, W direct-mapped caches)

[Sanchez'10]

The ZCache

□ When block A is brought in, it could replace one of four (say) blocks B, C, D, E; but B could be made to reside in one of three other locations (currently occupied by F, G, H); and F could be moved to one of three other locations
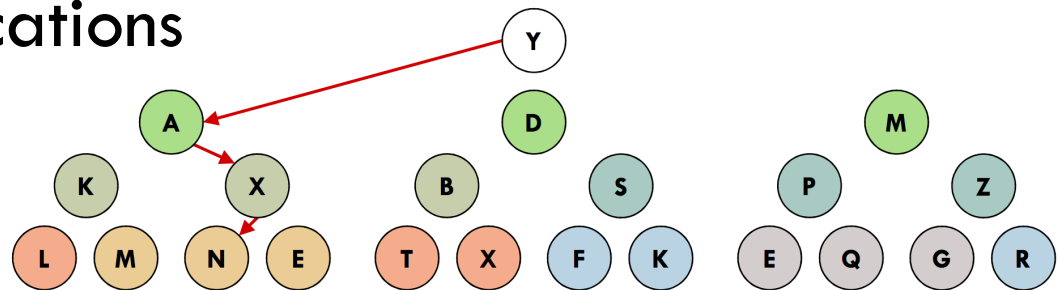
Read the paper for more details.

[Sanchez'10]