CS 61C: Great Ideas in Computer Architecture

Lecture 16: *Caches, Part 3*

Instructor: Sagar Karandikar

sagark@eecs.berkeley.edu

http://inst.eecs.berkeley.edu/~cs61c

Berkeley EECS

ELECTRICAL ENGINEERING & COMPUTER SCIENCES

You Are Here!

Software | *Hardware*

- Parallel Requests
  Assigned to computer
  e.g., Search "Katz"
- Parallel Threads
  Assigned to core
  e.g., Lookup, Ads
- Parallel Instructions
  >1 instruction @ one time
  e.g., 5 pipelined instructions
- Parallel Data
  >1 data item @ one time
  e.g., Add of 4 pairs of words
- Hardware descriptions
  All gates @ one time
- Programming Languages

Harness Parallelism & Achieve High Performance

Warehouse Scale Computer

Smart Phone

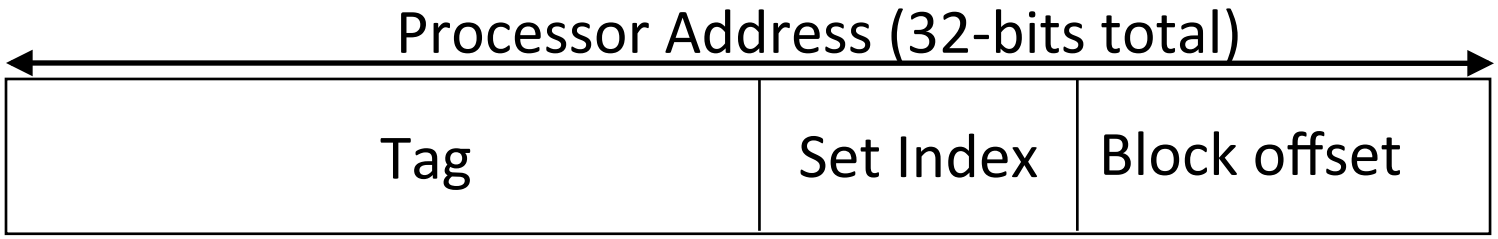
Today's Lecture

Caches Review

- Processor issues addresses, cache breaks them into Tag/Index/Offset
- Direct-Mapped vs. Set-Associative vs. Fully Associative
- AMAT = Hit Time + Miss Rate * Miss Penalty
- 3 Cs of cache misses: Compulsory, Capacity, Conflict
- Effect of cache parameters on performance

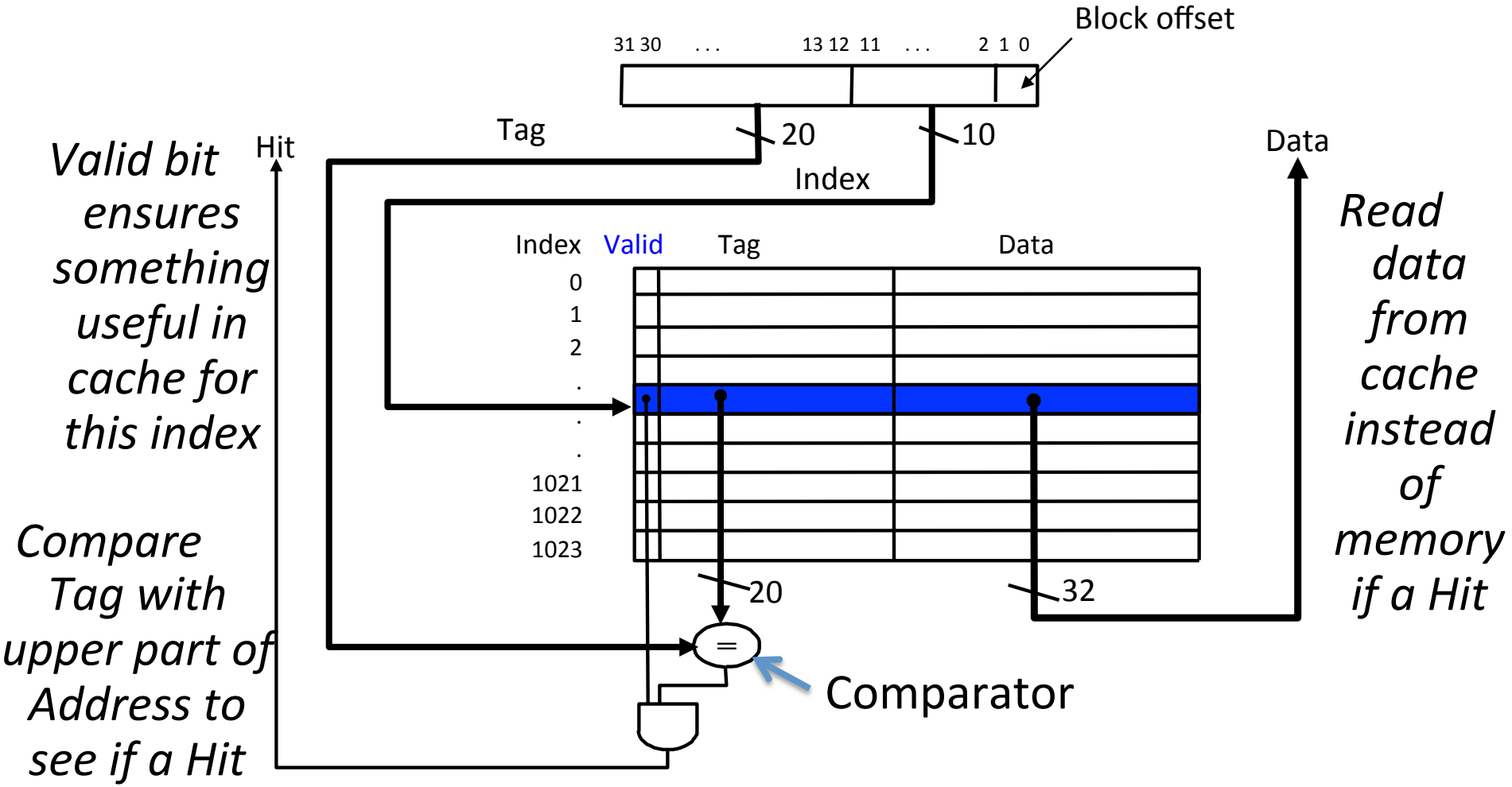
Review: Processor Address Fields used by Cache Controller

- Block Offset: Byte address within block
  - #bits = log_2(block size in bytes)
- Index: Selects which index (set) we want
  - # bits = log_2(number of sets)
- Tag: Remaining portion of processor address
  - processor address size - offset bits - index bits



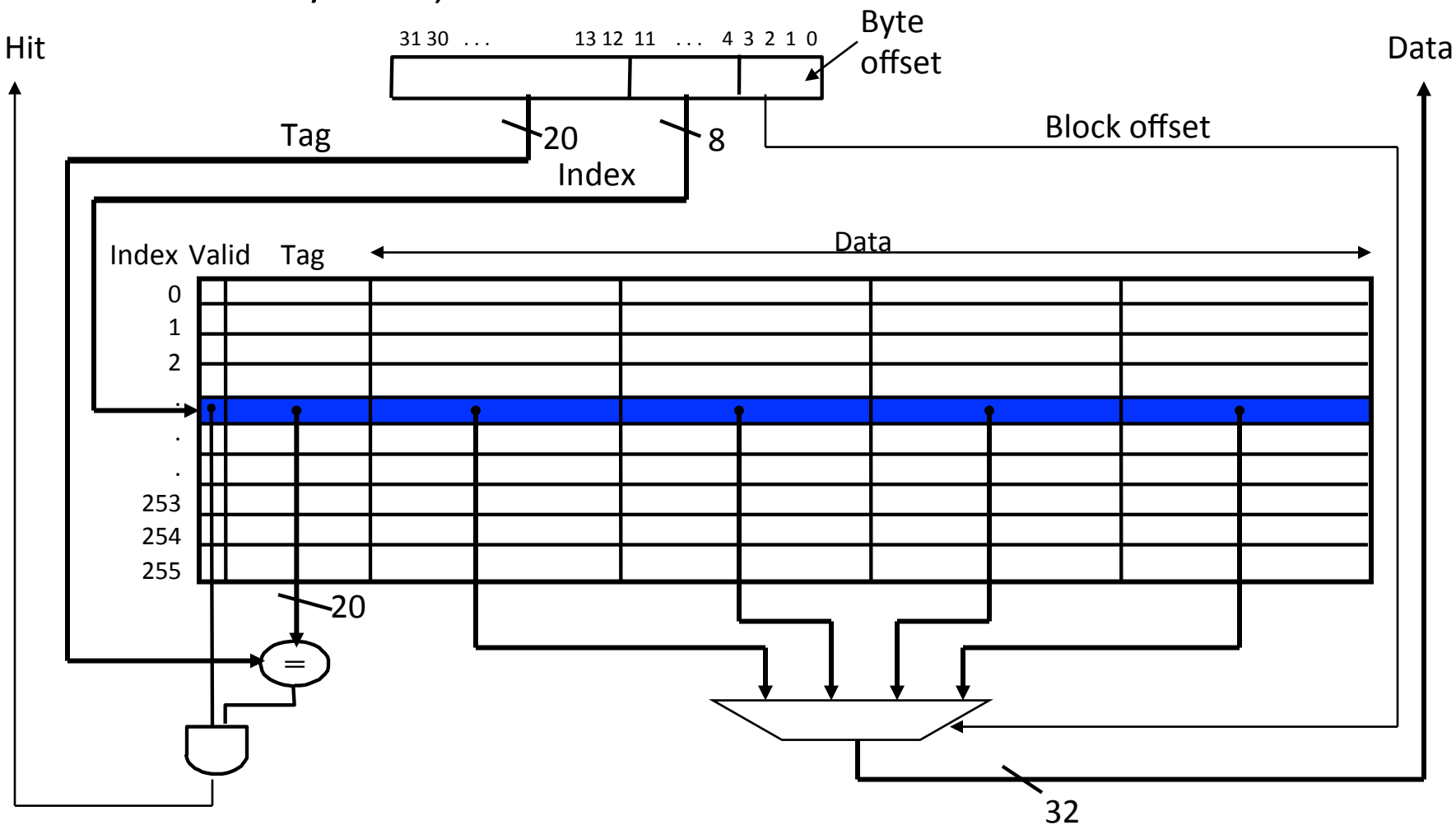
Review: Direct Mapped Cache

- One word blocks, cache size = 1Ki words (or 4KiB)



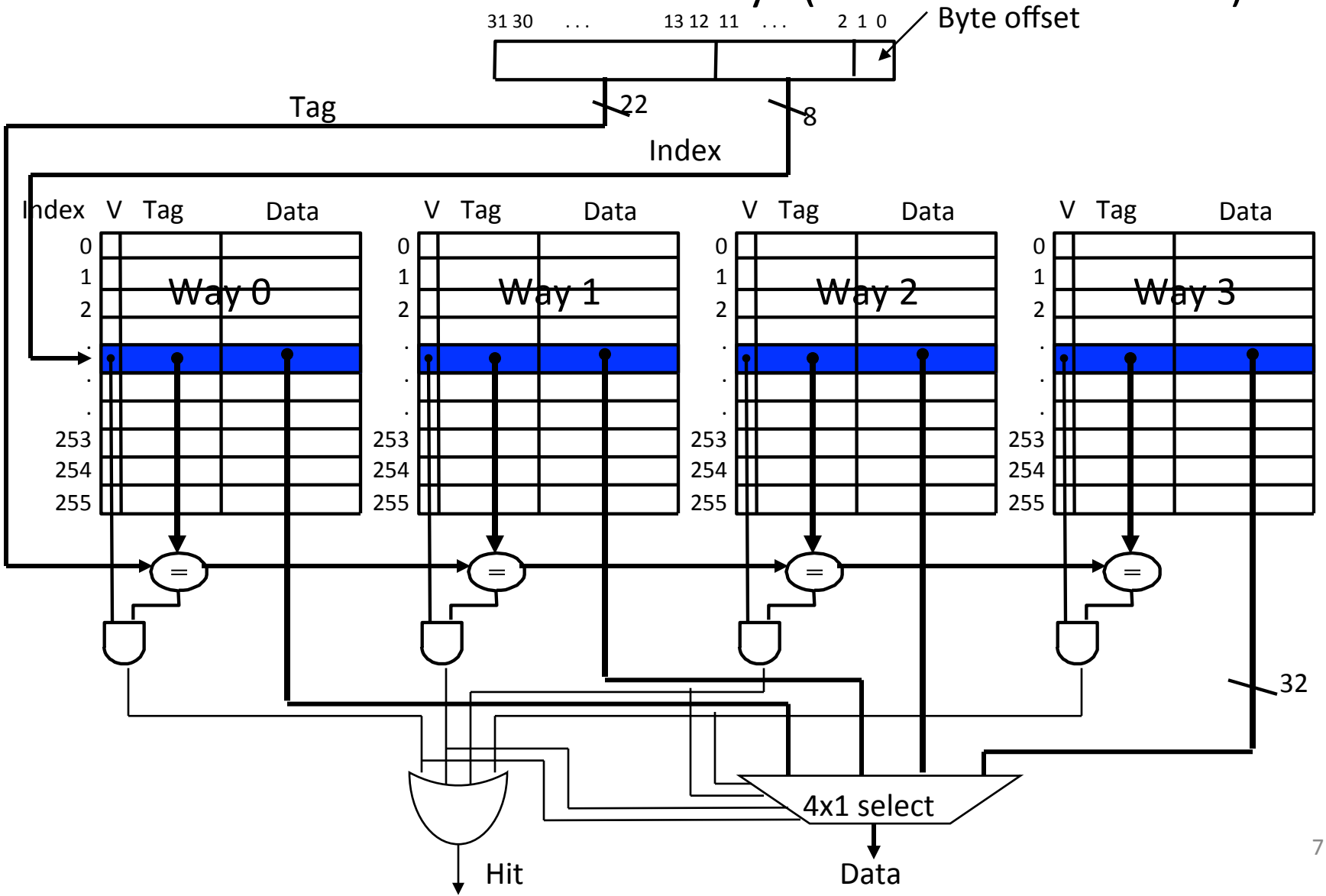
Review: Multiword-Block Direct-Mapped Cache

- Four words/block, cache size = 1Ki words



Review: Four-Way Set-Associative Cache

- 2^8 = 256 sets each with four ways (each with one block)



Review: Cache Measurement Terms

- Hit rate: fraction of accesses that hit in the cache
- Miss rate: 1 – Hit rate
- Miss penalty: time to replace a block from lower level in memory hierarchy to cache
- Hit time: time to access cache memory (including tag comparison)

These are controlled by the myriad cache parameters we can set…

Primary Cache Parameters

- Block size (aka line size)
  - how many bytes of data in each cache entry?
- Associativity
  - how many ways in each set?
  - Direct-mapped => Associativity = 1
  - Set-associative => 1 < Associativity < #Entries
  - Fully associative => Associativity = #Entries
- Capacity (bytes) = Total #Entries * Block size
- #Entries = #Sets * Associativity

Other Cache Parameters

- Write Policy
- Replacement policy

Write Policy Choices

- Cache hit:
  - **write through**: writes both cache & memory on every access
    - Generally higher memory traffic but simpler pipeline & cache design
  - **write back**: writes cache only, memory `written only when dirty entry evicted
    - A dirty bit per line reduces write-back traffic
    - Must handle 0, 1, or 2 accesses to memory for each load/store
- Cache miss:
  - **no write allocate**: only write to main memory
  - **write allocate** (aka fetch on write): fetch into cache
- Common combinations:
  - write through and no write allocate
  - write back with write allocate

Replacement Policy

In an associative cache, which line from a set should be evicted when the set becomes full?

- Random
- Least-Recently Used (LRU)
  - LRU cache state must be updated on every access
  - True implementation only feasible for small sets (2-way)
  - Pseudo-LRU binary tree often used for 4-8 way
- First-In, First-Out (FIFO) a.k.a. Round-Robin
  - Used in highly associative caches
- Not-Most-Recently Used (NMRU)
  - FIFO with exception for most-recently used line or lines

This is a second-order effect. Why?

Replacement only happens on misses

Sources of Cache Misses (3 C's)

- *Compulsory* (cold start, first reference):
  - 1st access to a block in memory, "cold", fact of life, not a lot you can do about it.
    - If running billions of instructions, compulsory misses are insignificant
- *Capacity*:
  - Cache cannot contain all blocks accessed by the program
    - Misses that would not occur with infinite cache
- *Conflict* (collision):
  - Multiple memory locations mapped to same cache set
    - Misses that would not occur with ideal fully associative cache

Rules for Determining Miss Type for a Given Access Pattern in 61C

1) Compulsory: A miss is compulsory if and only if it results from accessing the data for the first time. If you have ever brought the data you are accessing into the cache before, it's not a compulsory miss, otherwise it is.

2) Conflict: A conflict miss is a miss that's not compulsory and that would have been avoided if the cache was fully associative. Imagine you had a cache with the same parameters but fully-associative (with LRU). If that would've avoided the miss, it's a conflict miss.

3) Capacity: This is a miss that would not have happened with an infinitely large cache. If your miss is not a compulsory or conflict miss, it's a capacity miss.

Impact of Cache Parameters on Performance

- AMAT = Hit Time + Miss Rate * Miss Penalty
  - Note, we assume always first search cache, so must incorporate hit time for both hits and misses!
- For misses, characterize by 3Cs

Administrivia

- Project 3-1 Out
  - Last week, we built a CPU together, this week, you start building your own!
- HW4 Out - Caches
- Guerrilla Section on Pipelining, Caches on Thursday, 5-7pm, Woz

Administrivia

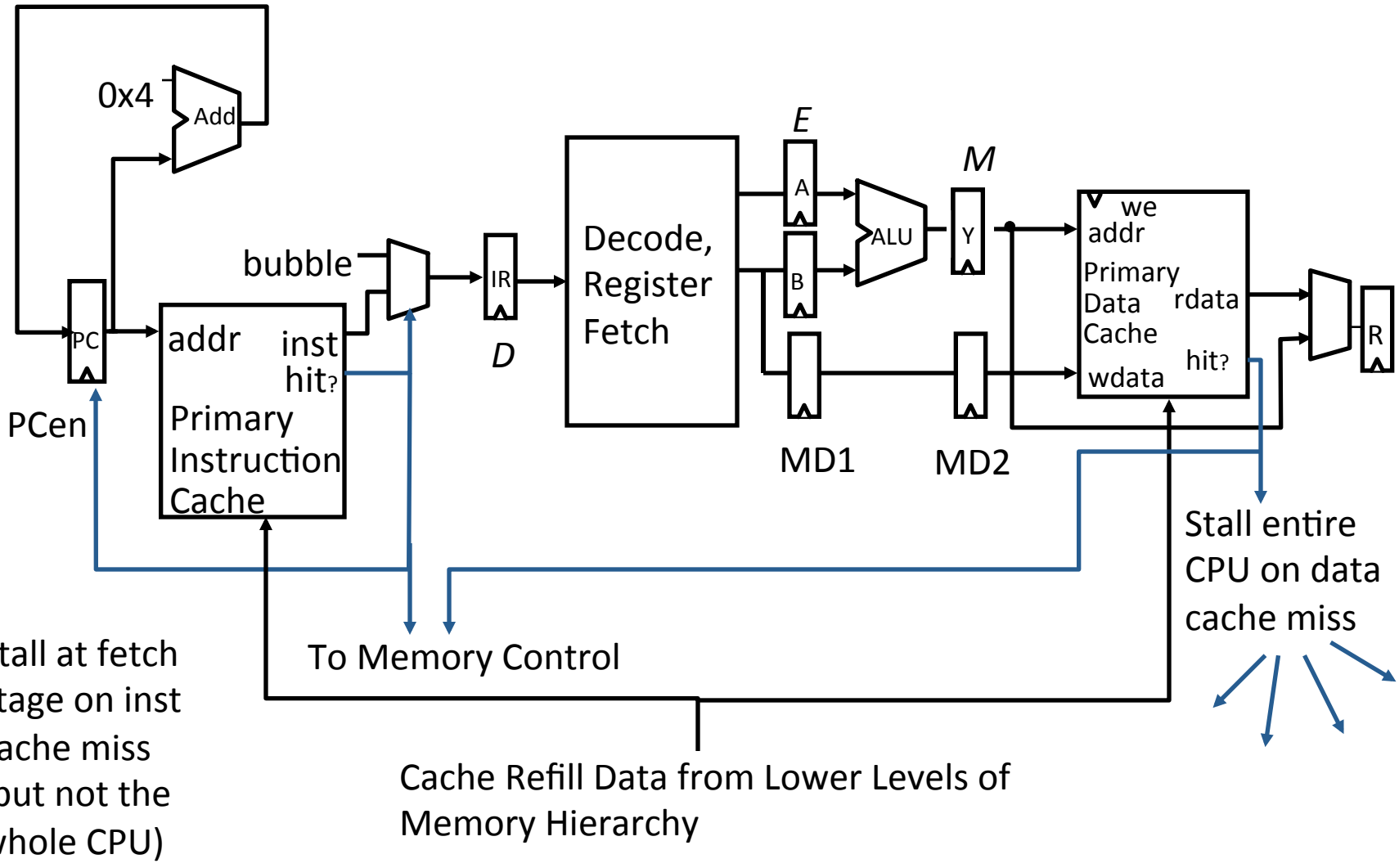
- Midterm 2 is one week from tomorrow
  - In this room, at this time
  - Two double-sided 8.5”x11” handwritten cheatsheets
  - We’ll provide a MIPS green sheet
  - No electronics
  - Covers up to and including **tomorrow’s lecture (07/21)***
  - Review session is Friday, 7/24 from 1-4pm in HP Aud.

* This is one less lecture than initially indicated

Break

CPU-Cache Interaction

(5-stage pipeline)



Increasing Block Size?

- Hit time as block size increases?
  - Hit time unchanged, but might be slight hit-time reduction as number of tags is reduced, so faster to access memory holding tags
- Miss rate as block size increases?
  - Goes down at first due to spatial locality, then increases due to increased conflict misses due to fewer blocks in cache
- Miss penalty as block size increases?
  - Rises with longer block size, but with fixed constant initial latency that is amortized over whole block

Increasing Associativity?

- Hit time as associativity increases?
  - Increases, with large step from direct-mapped to >=2 ways, as now need to mux correct way to processor
  - Smaller increases in hit time for further increases in associativity
- Miss rate as associativity increases?
  - Goes down due to reduced conflict misses, but most gain is from 1->2->4-way with limited benefit from higher associativities
- Miss penalty as associativity increases?
  - Unchanged, replacement policy runs in parallel with fetching missing line from memory

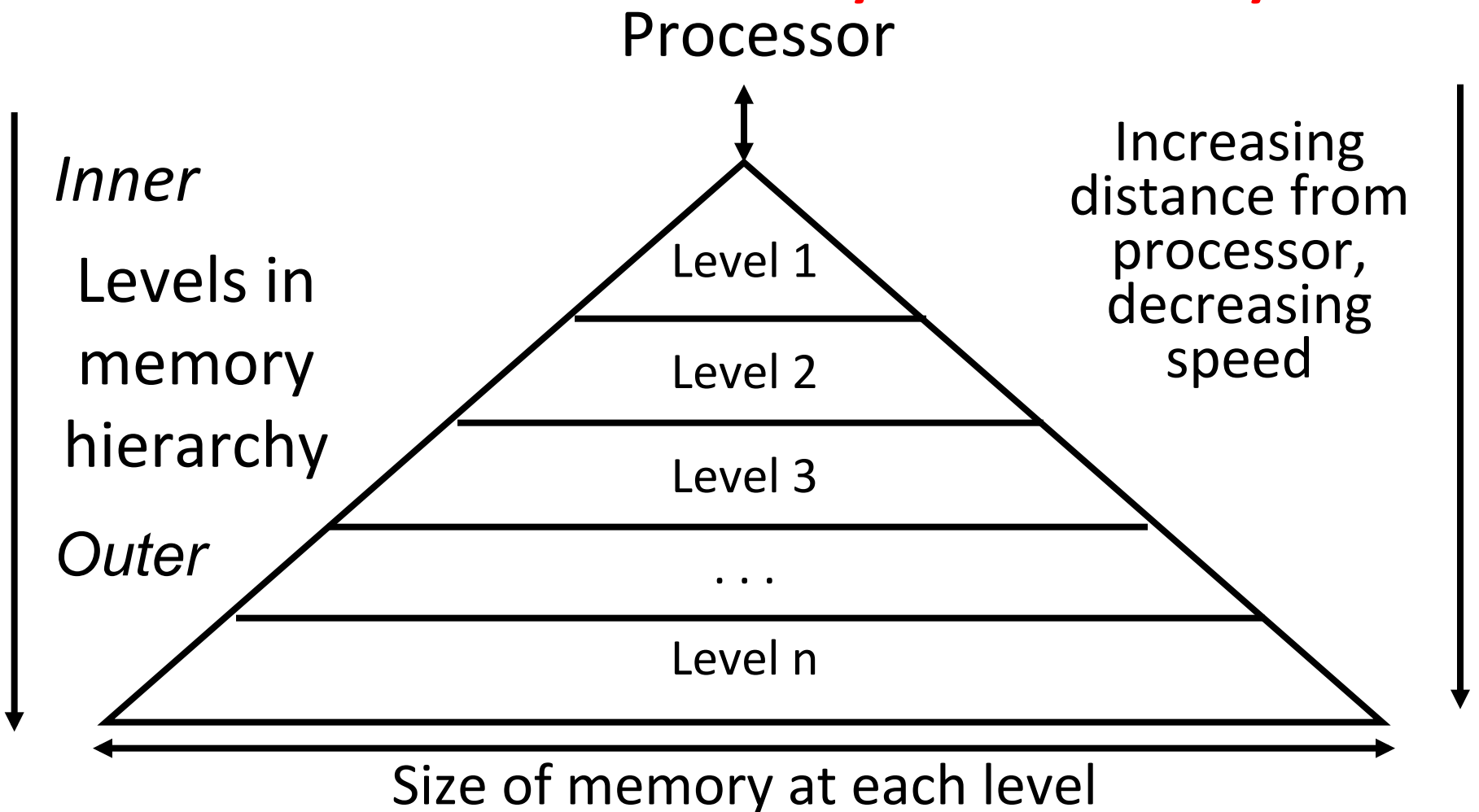
Increasing #Entries? (Capacity)

- Hit time as #entries increases?
  - Increases, since reading tags and data from larger memory structures
- Miss rate as #entries increases?
  - Goes down due to reduced capacity and conflict misses
  - *Architects rule of thumb: miss rate drops ~2x for every ~4x increase in capacity (only a gross approximation)*
- Miss penalty as #entries increases?
  - Unchanged

How to Reduce Miss Penalty?

- Could there be locality on misses from a cache?
- Use multiple cache levels!
- With Moore's Law, more room on die for bigger L1 caches and for second-level (L2) cache
- And in some cases even an L3 cache!
- IBM mainframes have ~1GB L4 cache off-chip.

Review: Memory Hierarchy



As we move to outer levels the latency goes up and price per bit goes down.

Local vs. Global Miss Rates

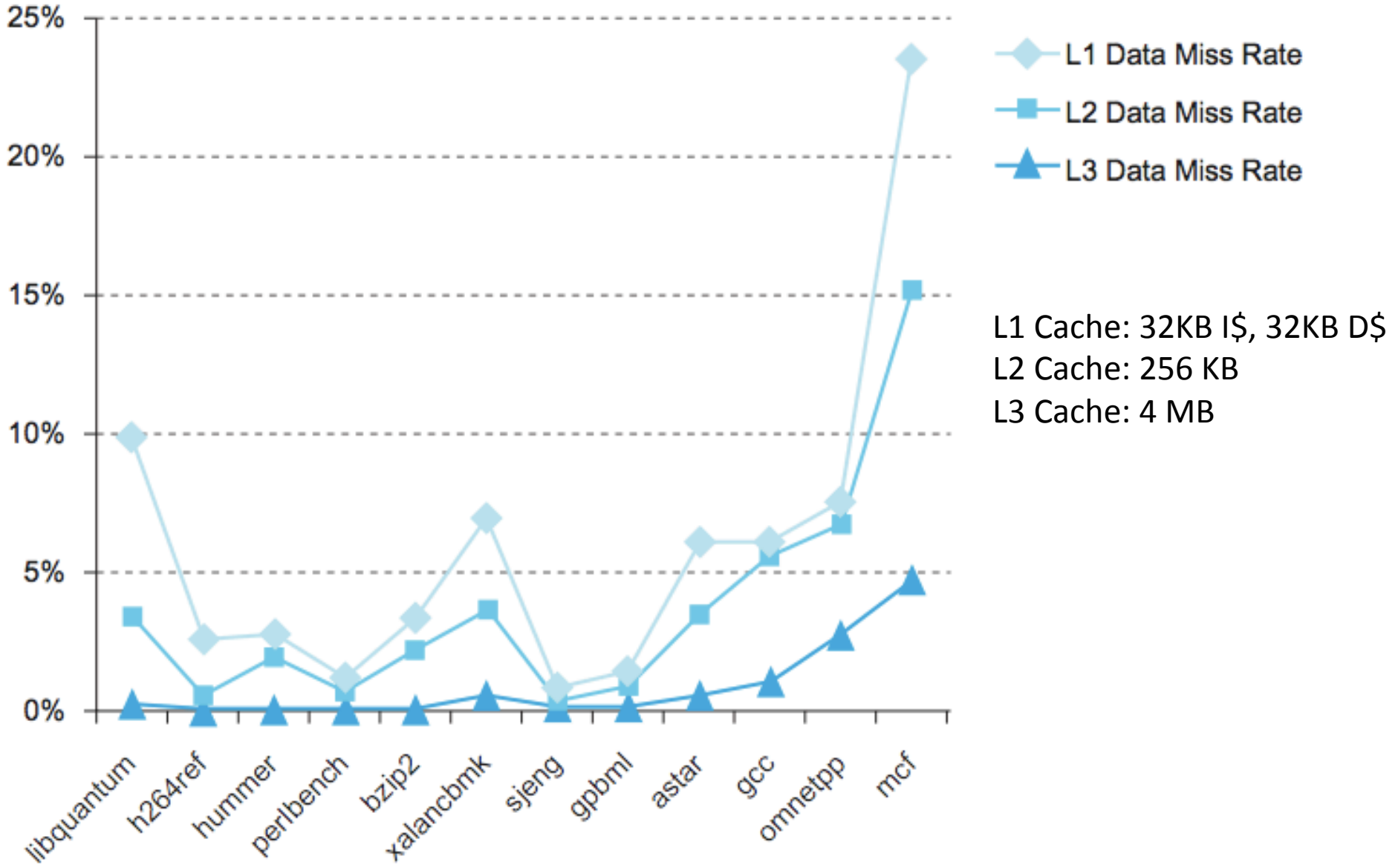
- *Local miss rate* – the fraction of references to one level of a cache that miss

  Local Miss rate L2$ = # $L2 Misses / # L1$ Misses

- *Global miss rate* – the fraction of references that miss in all levels of a multilevel cache

  L2$ local miss rate >> than the global miss rate

  L2$ Global Miss rate = L2$ Misses / Total Accesses



L1 Cache: 32KB I$, 32KB D$
L2 Cache: 256 KB
L3 Cache: 4 MB

FIGURE 5.47 The L1, L2, and L3 data cache miss rates for the Intel Core i7 920 running the full integer SPECCPU2006 benchmarks.

Local vs. Global Miss Rates

- *Local miss rate* – the fraction of references to one level of a cache that miss
  - Local Miss rate L2$ = $L2 Misses / L1$ Misses
- *Global miss rate* – the fraction of references that miss in all levels of a multilevel cache

  L2$ Global Miss rate = L2$ Misses / Total Accesses

  = L2$ Misses / L1$ Misses x L1$ Misses / Total Accesses

  = Local Miss rate L2$ x Local Miss rate L1$

- AMAT = Time for a hit + Miss rate x Miss penalty
- AMAT = Time for a L1$ hit + (local) Miss rate L1$ x (Time for a L2$ hit + (local) Miss rate L2$ x L2$ Miss penalty)

Characteristic	Intel Nehalem	AMD Opteron X4 (Barcelona)
L1 cache organization	Split instruction and data caches	Split instruction and data caches
L1 cache size	32 KB each for instructions/data per core	64 KB each for instructions/data per core
L1 block size	64 bytes	64 bytes
L1 write policy	Write-back, Write-allocate	Write-back, Write-allocate
L1 hit time (load-use)	Not Available	3 clock cycles
L2 cache organization	Unified (instruction and data) per core	Unified (instruction and data) per core
L2 cache size	256 KB (0.25 MB)	512 KB (0.5 MB)
L2 block size	64 bytes	64 bytes
L2 write policy	Write-back, Write-allocate	Write-back, Write-allocate
L2 hit time	Not Available	9 clock cycles
L3 cache organization	Unified (instruction and data)	Unified (instruction and data)
L3 cache size	8192 KB (8 MB), shared	2048 KB (2 MB), shared
L3 block size	64 bytes	64 bytes
L3 write policy	Write-back, Write-allocate	Write-back, Write-allocate
L3 hit time	Not Available	38 (?)clock cycles

On a 2012 Retina MBP

```
~ » sudo sysctl -a | grep cache
hw.cachelinesize = 64
hw.l1icachesize = 32768
hw.l1dcachesize = 32768
hw.l2cachesize = 262144
hw.l3cachesize = 6291456
kern.flush_cache_on_write: 0
kern.namecache_disabled: 0
vm.vm_page_filecache_min: 209715
vfs.generic.nfs.client.access_cache_timeout: 60
vfs.generic.nfs.server.reqcache_size: 64
net.inet.ip.rtmaxcache: 128
net.inet6.ip6.rtmaxcache: 128
hw.cacheconfig: 8 2 2 8 0 0 0 0 0 0
hw.cachesize: 17179869184 32768 262144 6291456 0 0 0 0 0
hw.cachelinesize: 64
hw.l1icachesize: 32768
```

On a hive machine, run:
`cat /sys/devices/system/cpu/cpu0/cache/index0/*`

A cool thing about Unix: you can poke around by reading files

CPI/Miss Rates/DRAM Access SpecInt2006

Name	CPI	Data Only L1 D cache misses/1000 instr	Data Only L2 D cache misses/1000 instr	Instructions and Data DRAM accesses/1000 instr
perl	0.75	3.5	1.1	1.3
bzip2	0.85	11.0	5.8	2.5
gcc	1.72	24.3	13.4	14.8
mcf	10.00	106.8	88.0	88.5
go	1.09	4.5	1.4	1.7
hmmer	0.80	4.4	2.5	0.6
sjeng	0.96	1.9	0.6	0.8
libquantum	1.61	33.0	33.1	47.7
h264avc	0.80	8.8	1.6	0.2
omnetpp	2.94	30.9	27.7	29.8
astar	1.79	16.3	9.2	8.2
xalancbmk	2.70	38.0	15.8	11.4
Median	1.35	13.6	7.5	5.4

Clickers/Peer Instruction

- Overall, what are L2 and L3 local miss rates?

CS61C In the News

- 2 "Privacy-respecting" Laptops on the frontpage of Hacker News yesterday:
  - The Gluglug Libreboot X200
  - Purism Librem 13/15
- Why does it matter?
  - You can run all the free software you want, but:
    - 1) Who knows what the hardware is really doing?
    - 2) 99.999% of your machines still have proprietary "blobs" of driver/firmware code
      - (some are even encrypted)

https://news.ycombinator.com/item?id=9912034
https://news.ycombinator.com/item?id=9912863

Break

Analyzing Code Performance on a Cache

- Let's say we're interested in analyzing program performance with respect to the cache:

```
#define SZ 2048 // 2^11
int foo() {
   int a[SZ];
   int sum = 0;
   for (int i = 0; i < SZ; i++) {
      sum += a[i];
   }
   return sum;
}
```

Analyzing Code Performance on a Cache

```
#define SZ 2048 // 2^11
int foo() {
  int a[SZ];
  int sum = 0;
  for (int i = 0; i < SZ; i++) {
    sum += a[i];
  }
  return sum;
}
```

1) Assume we only care about D$

2) Assume array is "block aligned"
i.e. First byte of a[0] is the first byte in a block

3) Assume local vars live in registers

So what *do* we care about?
The sequence of accesses to array `a`

Analyzing Code Performance on a Cache

Cache specs:
4 KiB Direct-Mapped
16 B Blocks

What's the miss rate?

```
#define SZ 2048 // 2^11
int foo() {
   int a[SZ];
   int sum = 0;
   for (int i = 0; i < SZ; i++) {
      sum += a[i];
   }
   return sum;
}
```

Important first questions:

1) What does our access pattern look like within a block?
2) How many cache blocks does our array take up? Does it fit entirely in the cache at once?

Analyzing Code Performance on a Cache

Cache specs:
4 KiB Direct-Mapped
16 B Blocks

What's the miss rate?

```
#define SZ 2048 // 2^11
int foo() {
   int a[SZ];
   int sum = 0;
   for (int i = 0; i < SZ; i++) {
      sum += a[i];
   }
   return sum;
}
```

1) What does our access pattern look like within a block?

We're iterating over 2048 ints, each of which are 4 bytes. A block in this cache holds 4 ints: (16 B/block)/(4 B/int) = 4 ints/block

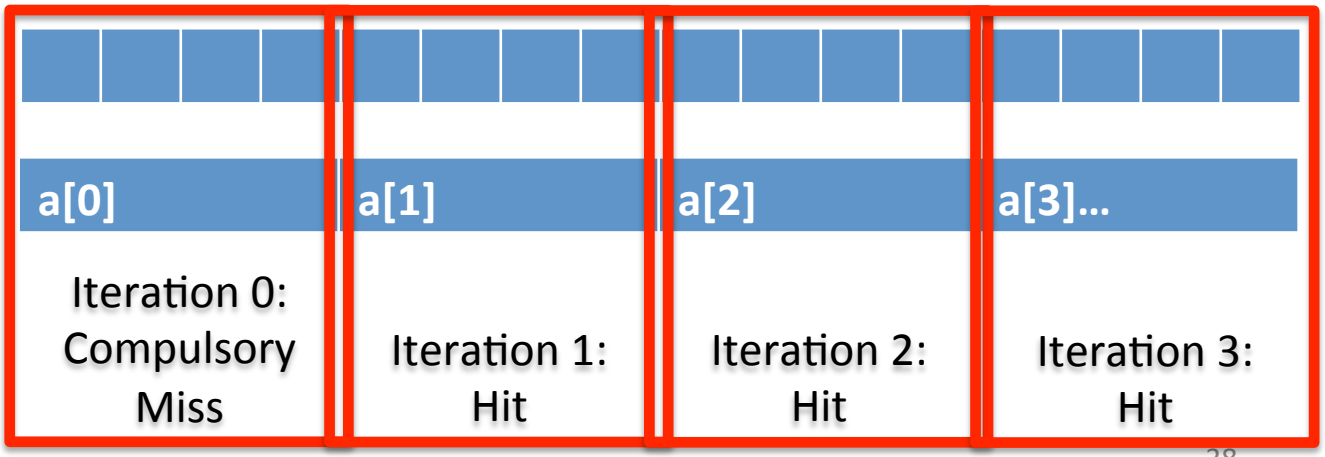
Analyzing Code Performance on a Cache

1) What does our access pattern look like within a block?

- We're iterating over 2048 ints, each of which are 4 bytes. A block in this cache holds 4 ints: (16 B/block)/ (4 B/int) = 4 ints/block

Block as bytes:

Block as ints:



This repeats over and over!
25% miss rate

Analyzing Code Performance on a Cache #2

Cache specs:
4 KiB Direct-Mapped
16 B Blocks

What's the miss rate?

```
#define SZ 2048 // 2^11
int foo() {
   int a[SZ];
   int sum = 0;
   for (int i = 0; i < SZ; i+=2) sum += a[i];
   for (int i = 0; i < SZ; i+=2) sum += a[i];
   return sum;
}
```

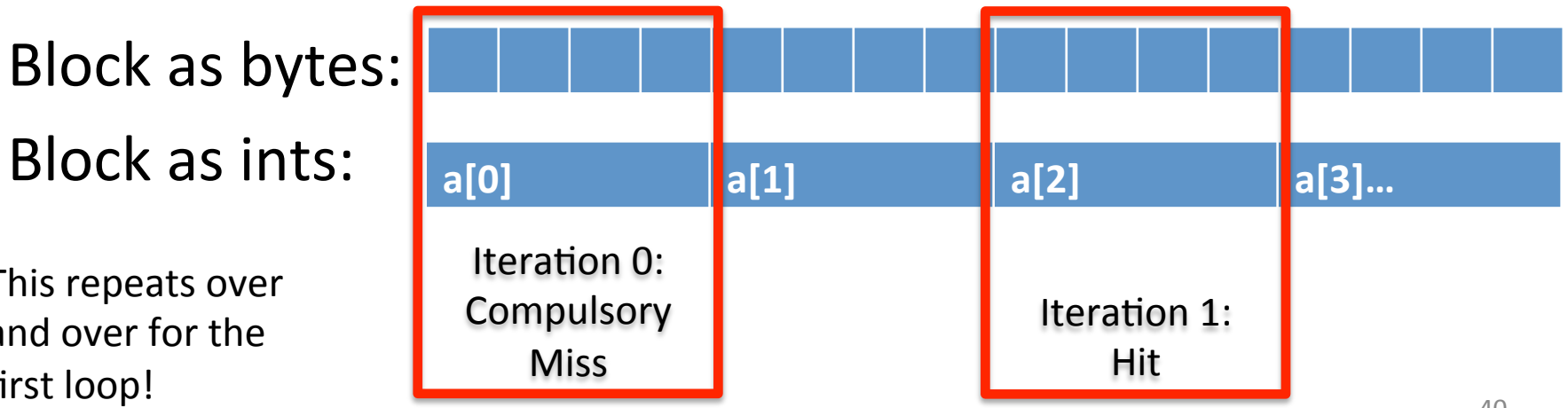
1) What does our access pattern look like within a block?

We're iterating over an array of 2048 ints (but only accessing 1024 of them), each of which are 4 bytes. A block in this cache holds 4 ints: (16 B/block)/(4 B/int) = 4 ints/block

Analyzing Code Performance on a Cache

1) What does our access pattern look like within a block? (let's stick with the first loop)

- We're iterating over ~~2048 ints~~ 1024 ints, each of which are 4 bytes. A block in this cache holds 4 ints: (16 B/block)/(4 B/int) = 4 ints/block



This repeats over and over for the first loop!
50% miss rate

Analyzing Code Performance on a Cache

But now, we have to worry about question 2:

2) How many cache blocks does our array take up?

The array contains 2048 ints. That's 2048*4 = 8192 B.
Cache blocks are 16 B each, so 8192/16 = 512 blocks.

Does it fit entirely in the cache at once?

No. Cache is only 4096 B, contains only 256 blocks.

Cache specs:
4 KiB Direct-Mapped
16 B Blocks

Analyzing Code Performance on a Cache #2

Cache specs:
4 KiB Direct-Mapped
16 B Blocks

What's the miss rate?

```
#define SZ 2048 // 2^11
int foo() {
   int a[SZ];
   int sum = 0;
   for (int i = 0; i < SZ; i+=2) sum += a[i];
   for (int i = 0; i < SZ; i+=2) sum += a[i];
   return sum;
}
```

So, to figure out what loop 2's performance is going to be like, we need to know what's in the cache at the end of loop 1…

Analyzing Code Performance on a Cache #2

Cache specs:
4 KiB Direct-Mapped
16 B Blocks

What's the miss rate?

```
#define SZ 2048 // 2^11
int foo() {
   int a[SZ];
   int sum = 0;
   for (int i = 0; i < SZ; i+=2) sum += a[i];
   // what's the state of the cache here?
}
```

Say for the sake of argument that our array starts at address 0. Then, halfway through the first loop (between i = 1022 and i = 1024), the first half of the array is in the cache.

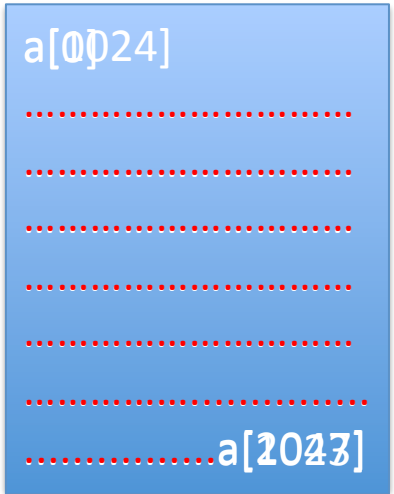
Analyzing Code Performance on a Cache #2

Cache specs:
4 KiB Direct-Mapped
16 B Blocks

What's the miss rate?

```
#define SZ 2048 // 2^11
int foo() {
   int a[SZ];
   int sum = 0;
   for (int i = 0; i < SZ; i+=2) sum += a[i];
   // what's the state of the cache here?
}
```

Now, when we execute the second half of the accesses in loop one, we replace everything already in the cache:



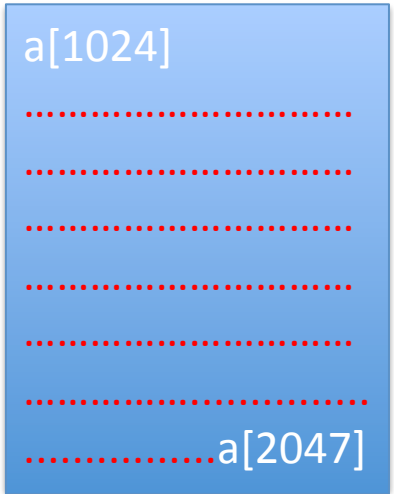
Analyzing Code Performance on a Cache #2

Cache specs:
4 KiB Direct-Mapped
16 B Blocks

What's the miss rate?

```
#define SZ 2048 // 2^11
int foo() {
   int a[SZ];
   int sum = 0;
   for (int i = 0; i < SZ; i+=2) sum += a[i];
   // what's the state of the cache here?
}
```

So, at the end of the first loop, the second half of the array is in the cache.



Analyzing Code Performance on a Cache #2

Cache specs:
4 KiB Direct-Mapped
16 B Blocks

What's the miss rate?

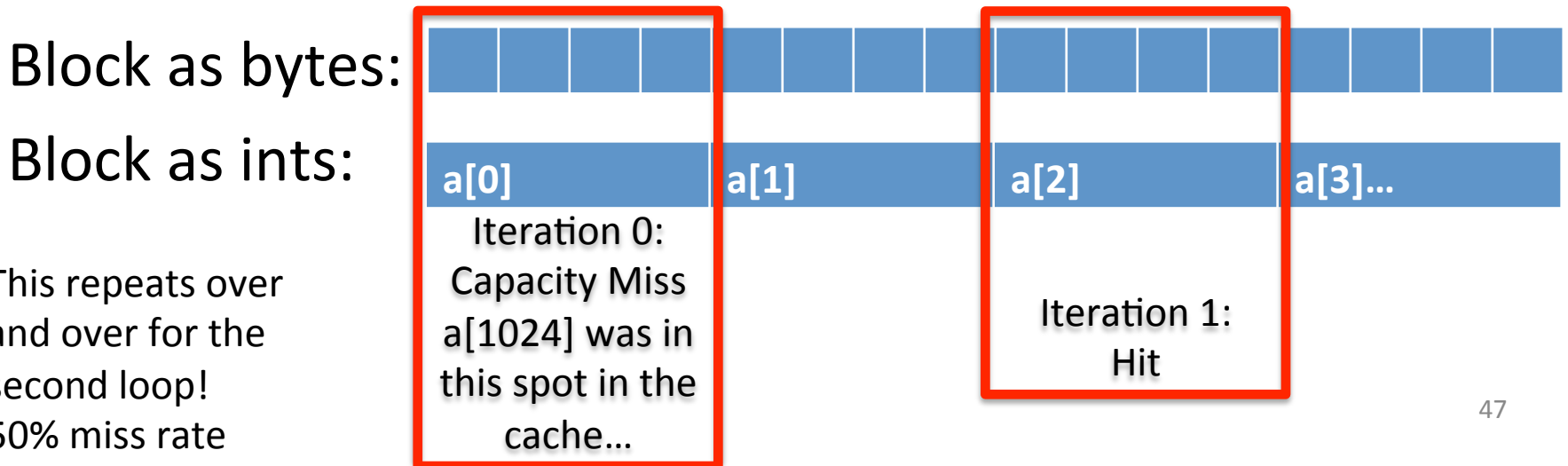
```
#define SZ 2048 // 2^11
int foo() {
   int a[SZ];
   int sum = 0;
   for (int i = 0; i < SZ; i+=2) sum += a[i];
   for (int i = 0; i < SZ; i+=2) sum += a[i];
   return sum;
}
```

Now, let's take a look at loop 2. Notice that it's going to start by accessing the first half of the array, none of which is in the cache.

Analyzing Code Performance on a Cache

1) What does our access pattern look like within a block? (now, for the second loop)

- We're iterating over ~~2048 ints~~ 1024 ints, each of which are 4 bytes. A block in this cache holds 4 ints: (16 B/block)/(4 B/int) = 4 ints/block



This repeats over and over for the second loop!
50% miss rate

Analyzing Code Performance on a Cache #2

Cache specs:
4 KiB Direct-Mapped
16 B Blocks

What's the miss rate?

```
#define SZ 2048 // 2^11
int foo() {
   int a[SZ];
   int sum = 0;
   for (int i = 0; i < SZ; i+=2) sum += a[i];
   for (int i = 0; i < SZ; i+=2) sum += a[i];
   return sum;
}
```

So, we got a 50% miss rate for loop #1 and a 50% miss rate for loop #2. Now, we compute a weighted average:

Total miss rate = weighted average of miss rates.
Half the accesses were in loop 1, the other half in loop 2:
Total miss rate = 0.5 * loop1 miss rate + 0.5 * loop2 miss rate
Total miss rate = 0.5 * 50% + 0.5 * 50% = 50%

Clicker Question p1

Cache specs:
1 KiB Direct-Mapped
16 B Blocks

```
#define SZ 2048 // 2^11
int foo() {
   int a[SZ];
   int sum = 0;
   for (int i = 0; i < SZ; i+=256) sum += a[i];
   for (int i = 0; i < SZ; i+=256) sum += a[i];
   return sum;
}
```

What's the miss rate for the first loop?

A. 0 % B. 25 % C. 50 % D. 75 % E. 100%

Clicker Question p2

Cache specs:
1 KiB Direct-Mapped
16 B Blocks

```
#define SZ 2048 // 2^11
int foo() {
   int a[SZ];
   int sum = 0;
   for (int i = 0; i < SZ; i+=256) sum += a[i];
   for (int i = 0; i < SZ; i+=256) sum += a[i];
   return sum;
}
```

What's the miss rate for the second loop?

A. 0 % B. 25 % C. 50 % D. 75 % E. 100%

Clicker Question p3

Cache specs:
1 KiB Direct-Mapped
16 B Blocks

```
#define SZ 2048 // 2^11
int foo() {
   int a[SZ];
   int sum = 0;
   for (int i = 0; i < SZ; i+=256) sum += a[i];
   for (int i = 0; i < SZ; i+=256) sum += a[i];
   return sum;
}
```

What types of misses do we get in the first/second loops?

A. Capacity/Capacity
B. Compulsory/Capacity
C. Compulsory/Conflict
D. Conflict/Capacity
E. Compulsory/Compulsory

In Conclusion, Huge Cache Design Space

- Several interacting dimensions
  - Cache size
  - Block size
  - Associativity
  - **Replacement policy**
  - **Write-through vs. write-back**
  - **Write-allocation**
- Optimal choice is a compromise
  - Depends on access characteristics
    - Workload
    - Use (I-cache, D-cache)
  - So we learned how to analyze code performance on a cache
- Simplicity often wins