

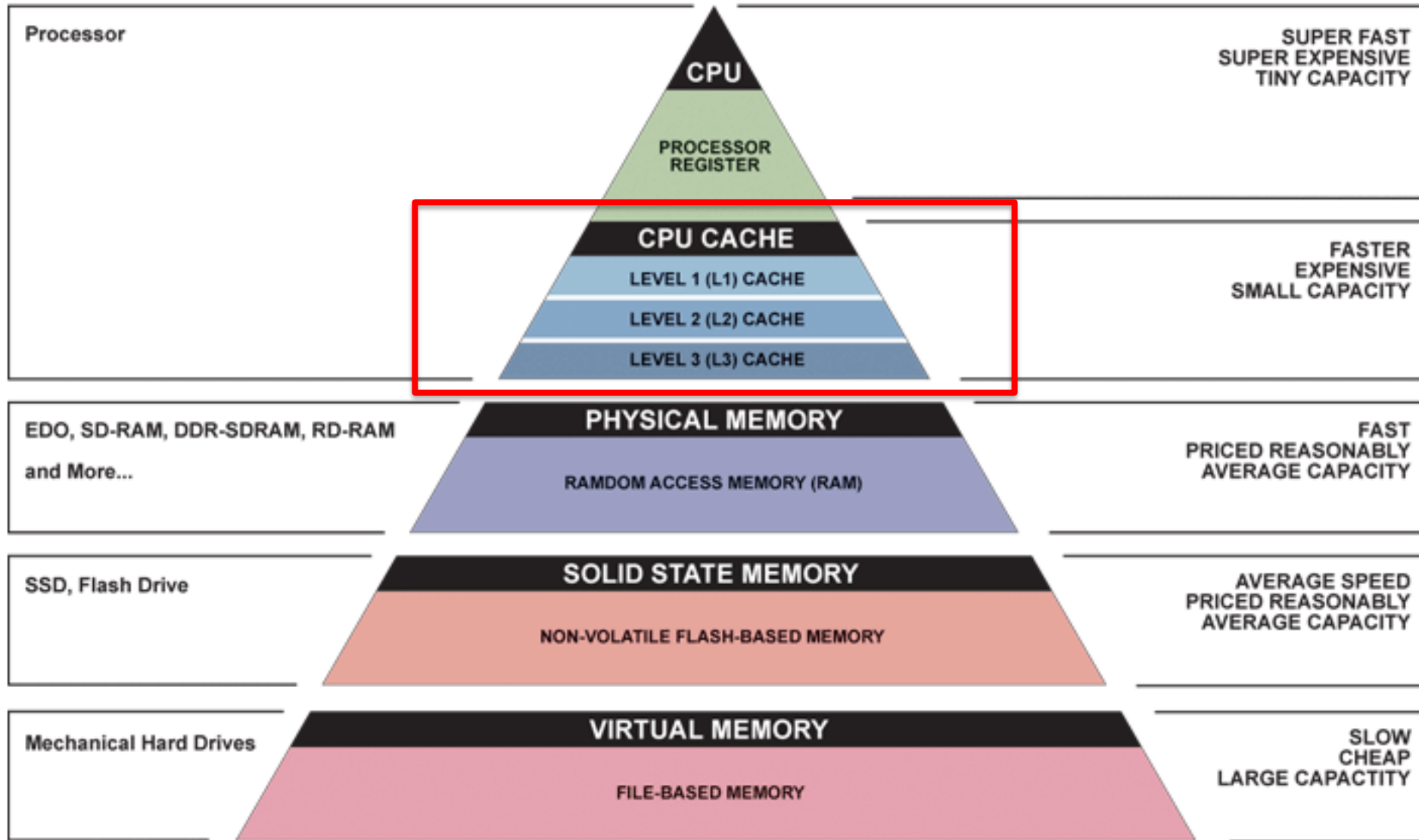
Great Ideas in Computer Architecture

Multilevel Caches, Cache Questions

Instructor: Sean Farhat



Great Idea #3: Principle of Locality/ Memory Hierarchy



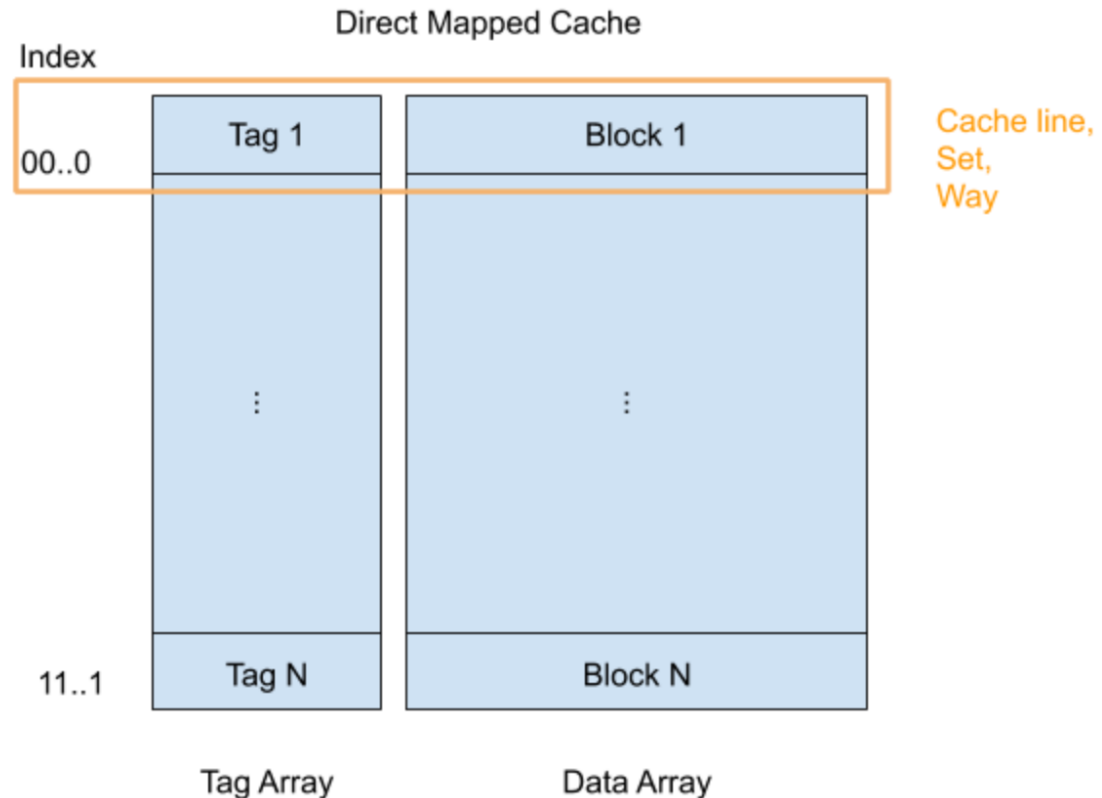
Review of Last Lecture

Direct-Mapped Caches:

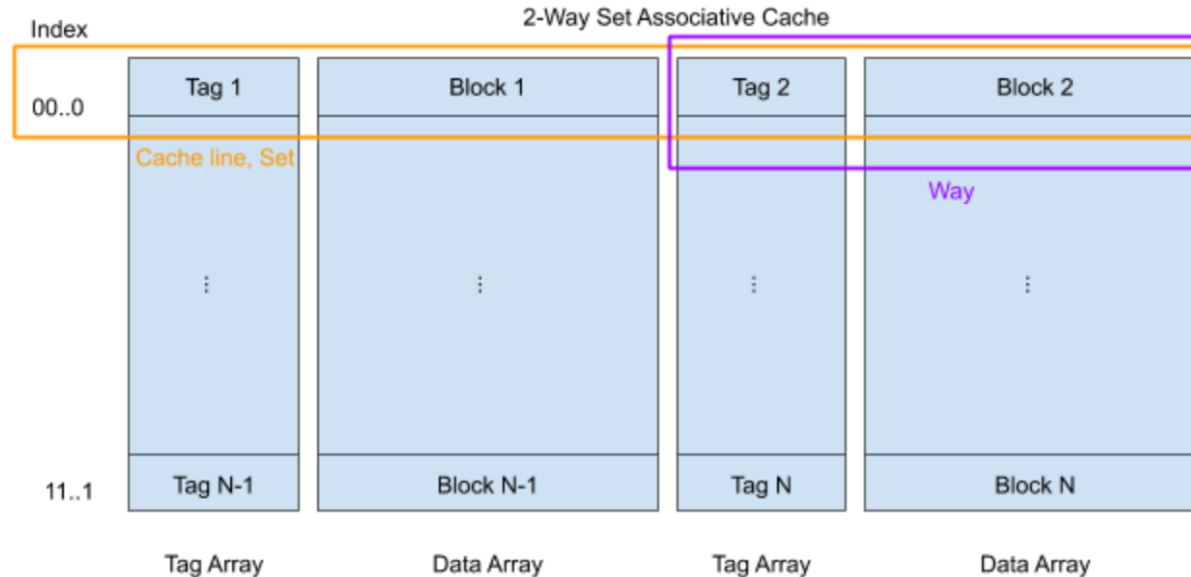
Use “hash function” to determine location for block

- Each block maps deterministically into a single row; which row == determined by index bit(s)!

index bits = $\log_2(\text{cache size} / \text{block size})$



Review of Last Lecture



N-way Set Associative Caches:

Split slots into sets of size N, map into set, which is a row containing multiple blocks

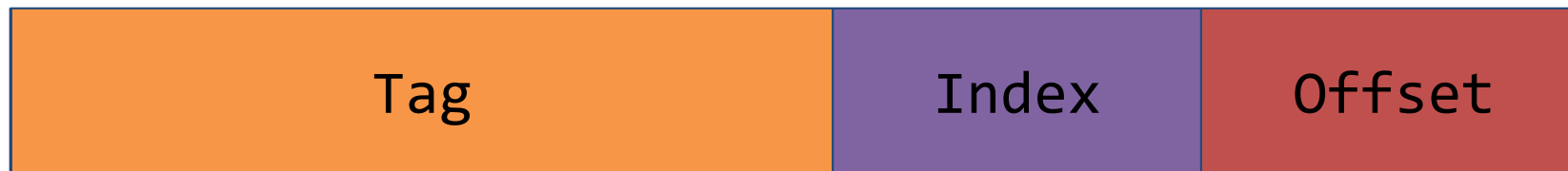
- Set (row) chosen by index, block (spot WITHIN row) chosen by replacement policy

$$\# \text{ index bits} = \log_2(\text{cache size} / (N * \text{block size}))$$

Review of Last Lecture

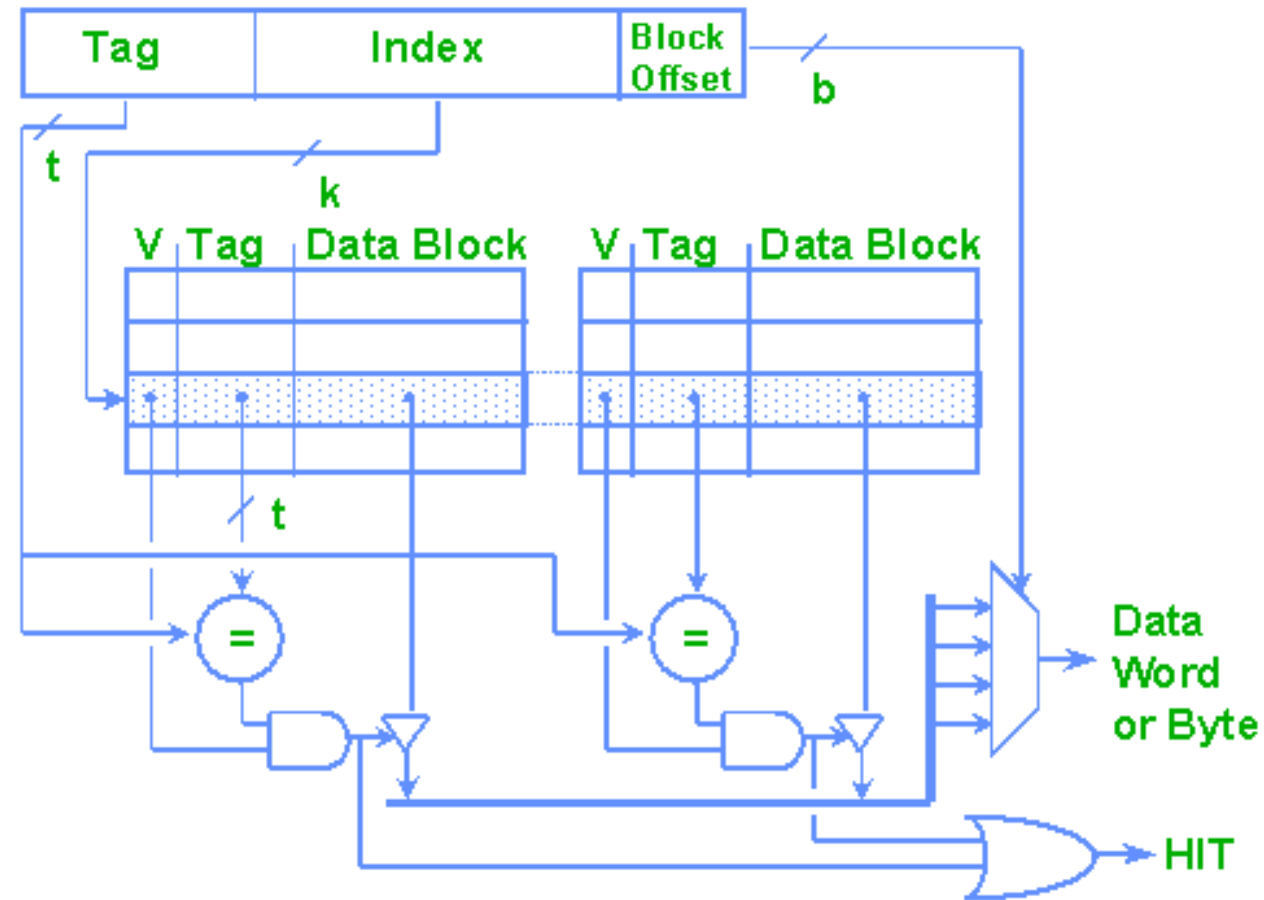
TIO breakdown of memory address

- **Index** field is result of hash function (which *set, row*)
 - $\log_2(\text{cache size} / (N * \text{block size}))$
- **Tag** field is identifier (which block is currently in slot)
 - $\text{addr} - \text{index} - \text{offset}$
- **Offset** field indexes into block
 - $\log_2(\text{block size})$



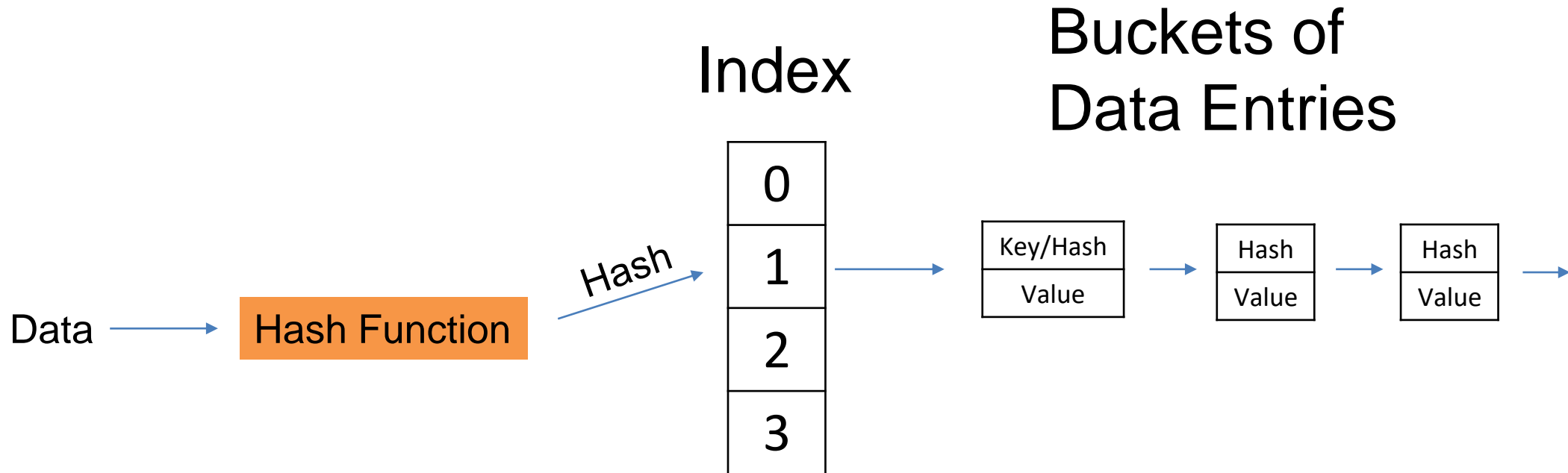
2-Way Set-Associative Cache

Credit: MIT



13

An Alternate Perspective: A Hash Table

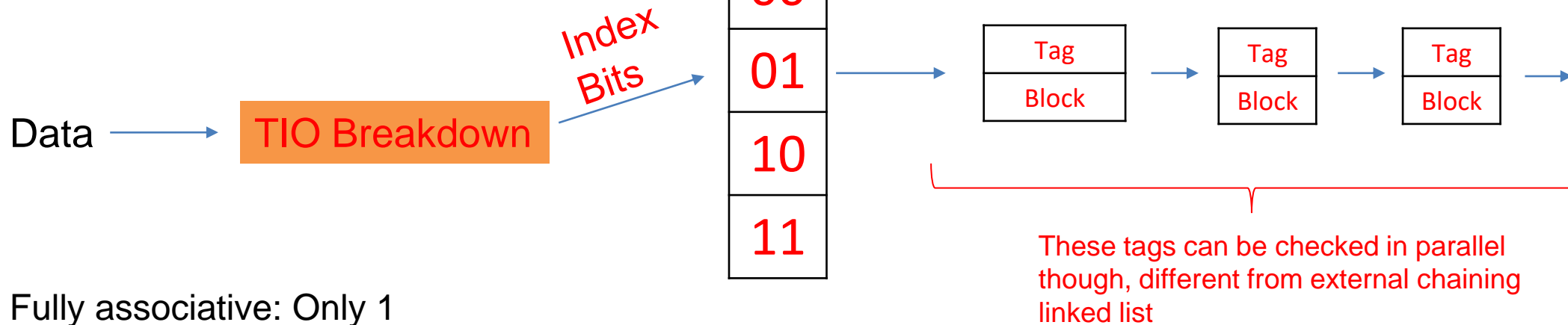


An Alternate Perspective: A Hash Table

Direct mapped cache : Each bucket only has one entry

N-way set associative: N entries in each bucket

Fully associative: Only 1 index/bucket, which holds all entries



Question

- How many total bits are maintained in the following cache?
- 4-way SA cache, random replacement
- Cache size 1 KiB, Block size 16 B
- Write-back
- 16-bit address space


(A) $2^6 \times (2^7 + 2^3 + 2^1) = 8.625 \text{ Kib}$

(B) $2^4 \times (2^7 + 2^3 + 2^0) = 2.140625 \text{ Kib}$

(C) $2^4 \times (2^7 + 2^3 + 2^1) = 2.15625 \text{ Kib}$

(D) $2^4 \times (2^7 + 6 + 2^1) = 2.125 \text{ Kib}$

Question

- How many total bits are maintained in the following cache?
- 4-way SA cache, random replacement
- Cache size 1 KiB, Block size 16 B
- Write-back  dirty bit
- 16-bit address space

(A) $2^6 \times (2^7 + 2^3 + 2^1) = 8.625 \text{ Kib}$

(B) $2^4 \times (2^7 + 2^3 + 2^0) = 2.140625 \text{ Kib}$

(C) $2^4 \times (2^7 + 2^3 + 2^1) = 2.15625 \text{ Kib}$

(D) $2^4 \times (2^7 + 6 + 2^1) = 2.125 \text{ Kib}$

slots?

bits per slot?

- data?
- tag?
- valid?
- dirty?

Question

- How many total bits are maintained in the following cache?

- 4-way SA cache, random replacement

- Cache size 1 KiB, Block size 16 B

- Write-back

- 16-bit address space

TIO

O: $\log_2(16) = 4$

I: $2^6 \text{ slots} / 4 \text{ ways} = 2^4 \text{ sets so } I = 4$

T: $16 - 4 - 4 = 8 = 2^3 \text{ bits}$

Valid and dirty:

(A) $2^6 \times (2^7 + 2^3 + 2^1) = 8.625 \text{ Kib}$

2

In each slot:

(B) $2^4 \times (2^7 + 2^3 + 2^0) = 2.140625 \text{ Kib}$

Data: $16\text{B} * 8$
bits per byte =
 $128 \text{ bits} = 2^7$

(C) $2^4 \times (2^7 + 2^3 + 2^1) = 2.15625 \text{ Kib}$

(D) $2^4 \times (2^7 + 6 + 2^1) = 2.125 \text{ Kib}$

slots?

$2^{10}\text{B} \div 2^4\text{B}$
 $= 2^6 \text{ slots}$

Agenda

- **AMAT**
- Multilevel Caches
- Improving Cache Performance
- Anatomy of a Cache Question
- Example Cache Questions
- Bonus: Contemporary Cache Specs

Measuring Memory Accesses

- In a machine without caches, the time it takes to access memory is easy to measure! If going to memory takes 1000 cycles, every access is 1000 cycles
- What happens when we introduce caches?
 - Every access first checks the cache
 - Some hit (are found) and so we don't need to check memory at all!
 - Some miss and have to go to memory
 - Can represent “average” memory access time

Average Memory Access Time

- *Average Memory Access Time* (AMAT): average time to access memory considering both hits and misses

$$\text{AMAT} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$$

(abbreviated $\text{AMAT} = \text{HT} + \text{MR} \times \text{MP}$)

- **Hit time:** time to check if the data we're looking for is in the cache (also called "tag check!")
- **Miss rate:** percentage of accesses of a program that are *not* in the cache when we look for them
- **Miss penalty:** cost incurred by going to memory

AMAT Derivation

Consider a 2 level hierarchy which has a cache and main memory:

$$\begin{aligned}\text{AMAT} &= (\text{Hit rate} * \text{Time to go to cache}) + (\text{Miss rate} * \text{Time to go to memory}) \\ &= (\text{Hit rate} * \text{Hit time}) + (\text{Miss rate} * (\text{Miss penalty} + \text{Hit time})) \\ &= (\text{Hit rate} * \text{Hit time}) + (\text{Miss rate} * \text{Hit time}) + (\text{Miss rate} * \text{Hit penalty}) \\ &= \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}\end{aligned}$$

AMAT Goals!

- **Goal 1:** Examine how changing the different cache parameters affects our AMAT
 - Hit time, miss rate, miss penalty
- **Goal 2:** Examine how to optimize your code for better cache performance (Project 4)
 - Cache blocking, access patterns, etc.

Hit Time

- Time taken to search the cache for the address/information you want!
- Contributing factors:
 - How many comparisons do you have to do?
 - Cache size decreases → hit time decreases
 - Generally, as associativity increases, so does hit time
 - Why? Increase in associativity == decrease in number of sets, even without a change in blocks! We have to look in more places for the same data
 - Can do tag comparisons in parallel, but requires more hardware (costly) that is more complex (slower)

Miss Rate

- The fraction of accesses your program makes which do *not* hit in the cache; the fraction of accesses which require you to go to memory
- Contributing factors:
 - Access pattern in your program
 - Goal: apply $f(x)$ to all vals in array
 - Good: apply $f(x)$ to blocks of the array at a time
 - Bad: pick elements at random, or between blocks
 - Block/cache size, type
 - Larger blocks, larger caches, more associativity can hold more data, keep data around longer
 - Reducing misses (Three C's!)

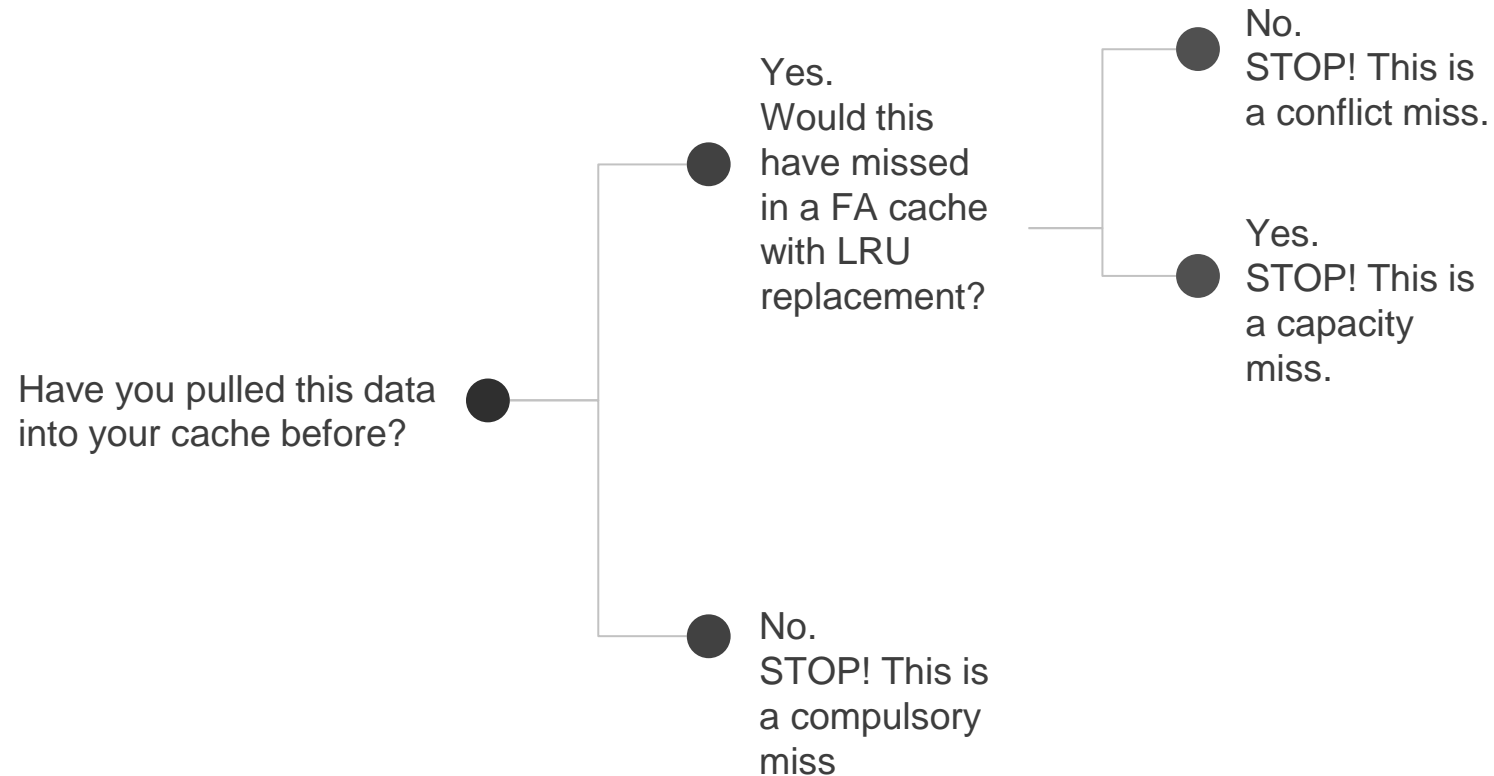
Sources of Cache Misses: The 3Cs

- **Compulsory:** (Many names: cold start, process migration (switching processes), 1st reference)
 - First access to block impossible to avoid;
Effect is small for long running programs
- **Capacity:**
 - Cache cannot contain all blocks accessed by the program, so full associativity won't hold all blocks
- **Conflict:** (collision)
 - Multiple memory locations mapped to the same cache location, so theres a lack of associativity

The 3Cs: Design Solutions

- Compulsory:
 - Increase block size (increases MP; too large blocks could increase MR)
- Capacity:
 - Increase cache size (may increase HT)
- Conflict:
 - Increase associativity (to fully associative) (may increase HT)

Which type of miss is it?

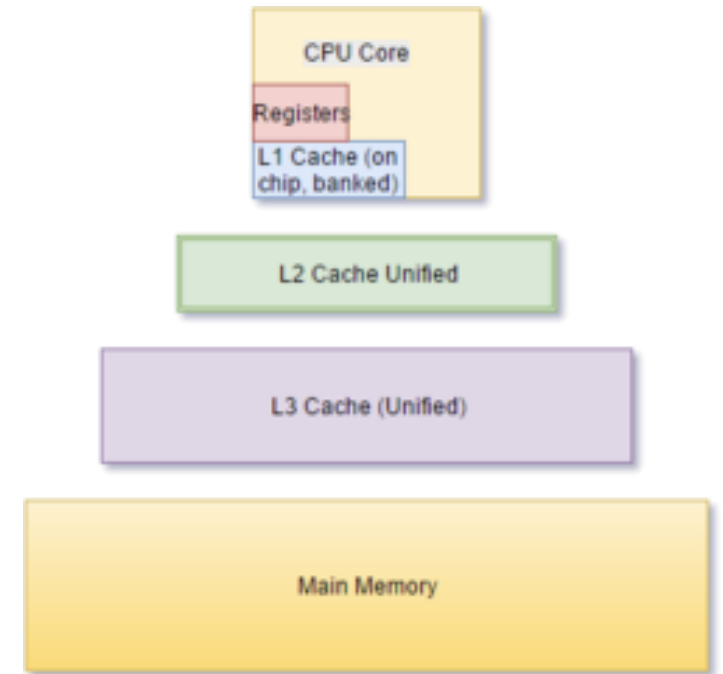


Conflict/Capacity Misses in FA Caches

- It is equivalent to say that in a fully associative cache:
 - Conflict misses simply do not exist
 - Don't make any sense
 - Are just the same as capacity misses
- Why? Conflict misses should be able to be avoided by increasing associativity, but in a fully associative cache, we can't do that anymore

Miss Penalty

- “Cost” (usually in cycles, or fractions of a second) of a miss
 - Often, this is going to memory, but sometimes it is the cost of going to another cache!
- Contributing factors:
 - How “big” is your memory hierarchy?
 - More steps == more penalty for things that miss at every level, but less things to incur the full penalty
 - Smaller block size → lower MP



Question

Processor specs: 200 ps clock, miss penalty of 50 clock cycles, miss rate of 0.02 misses/instruction, and hit time of 1 clock cycle

– Which of the following improvements will decrease (better) our AMAT the most?

A) 190 ps clock

B) Change miss penalty to 40 cycles

C) Change miss rate to 0.015 per instruction

AMAT Example

- **Processor specs:** 200 ps clock, MP of 50 clock cycles, MR of 0.02 misses/instruction, and HT of 1 clock cycle

AMAT = ???

$$1 + 0.02 \times 50 = 2 \text{ clock cycles} = 400 \text{ ps}$$

- Which improvement would be best?
 - 190 ps clock
 - $2 \text{ cycles} = 2 * 190 = 380$
 - MP of 40 clock cycles
 - $1 + 0.02 * 40 = 360$
 - MR of 0.015 misses/instruction
 - $1 + .015 * 50 = 350$

Review

- Cache performance measured using AMAT
 - Parameters that matter:
 - Hit Time (HT)
 - Miss Rate (MR)
 - Miss Penalty (MP)
 - $AMAT = Hit\ Time + Miss\ Rate \times Miss\ Penalty$
- The 3 Cs of cache misses and their fixes
 - Compulsory: Increase block size
 - Capacity: Increase cache size
 - Conflict: Make the cache fully associative

Agenda

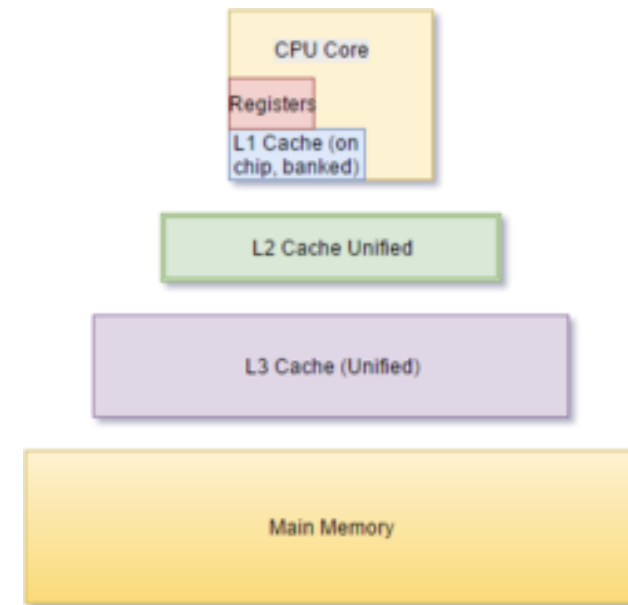
- AMAT
- **Multilevel Caches**
- Improving Cache Performance
- Anatomy of a Cache Question
- Example Cache Questions
- Bonus: Contemporary Cache Specs

Minimising AMAT

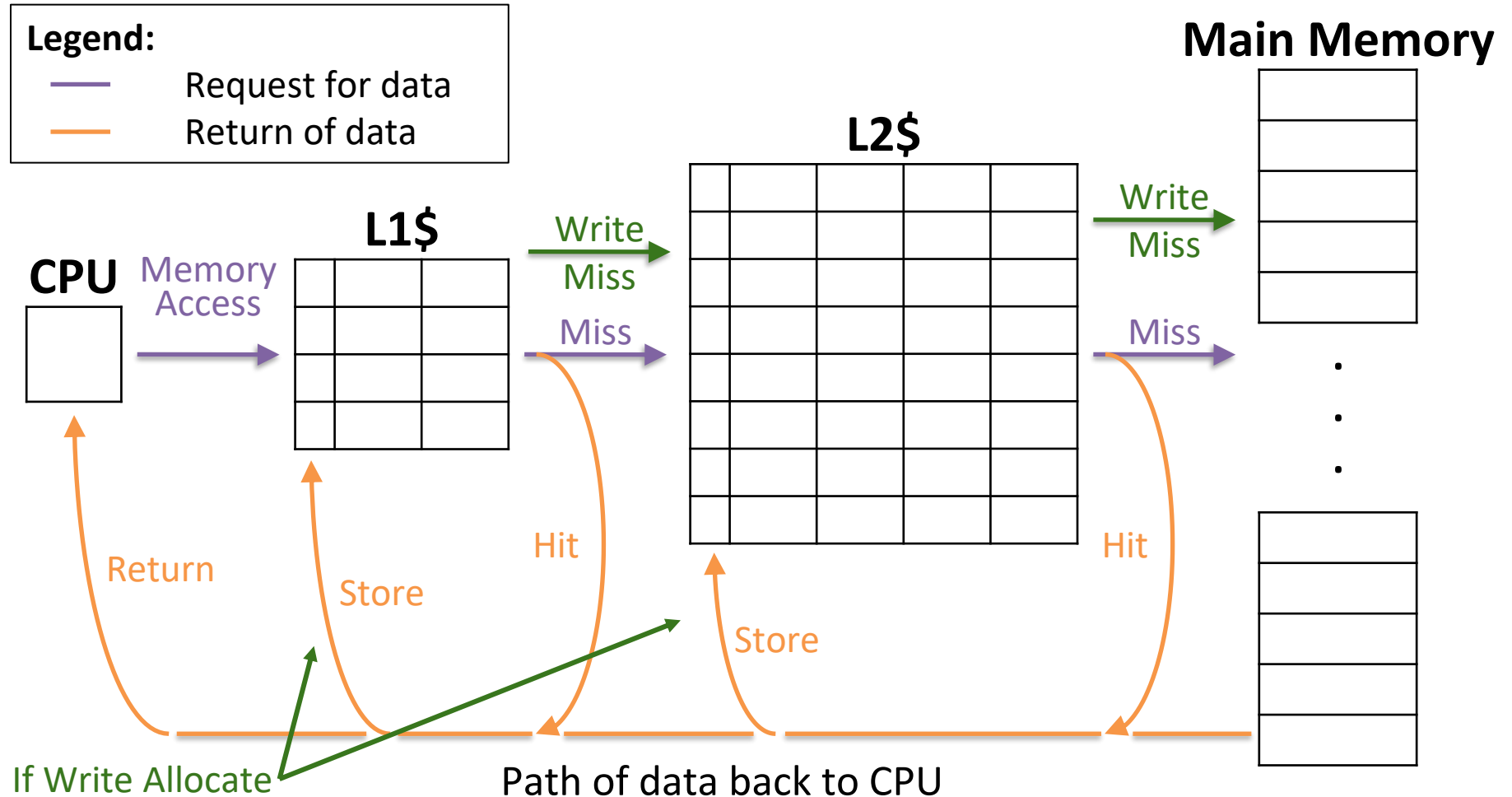
- To make AMAT small, we want to have a small hit time, low miss rate, and/or small miss penalty. It is hard to do all these at once!
 - Direct mapped caches have small hit time
 - Fully-Associative caches have a low miss rate
 - Small blocks help with lowering miss penalty, so do many opportunities to store data
- Can we combine all these things into one structure to achieve our goal?

Multiple Cache Levels

- With advancing technology, have more room on chip for bigger L1 caches and for L2 (and in some cases even L3) cache
 - Higher numbered caches are lower-level (closer to memory)
- Multilevel caching is a way to reduce miss penalty; picking the right kinds of caches can help reduce other AMAT factors
 - So what does this look like?



Multilevel Cache Diagram



Design Considerations

- L1\$ focuses on *low hit time* (fast access)
 - minimize HT to achieve shorter clock cycle
 - L1 MP significantly reduced by presence of L2\$, so can be smaller/faster even with higher MR
 - e.g. smaller \$ (fewer rows)
- L2\$, L3\$ focus on *low miss rate*
 - As much as possible avoid reaching to main memory (heavy penalty)
 - e.g. larger \$ with larger block sizes (same # rows)

Multilevel Cache AMAT

- $AMAT = L1\ HT + L1\ MR \times L1\ MP$
 - Now $L1\ MP$ depends on other cache levels
 - Must trace ENTIRE path to memory
- $L1\ MP = L2\ HT + L2\ MR \times L2\ MP$
 - If more levels, then continue this chain
(i.e. $MP_i = HT_{i+1} + MR_{i+1} \times MP_{i+1}$)
 - Final MP is main memory access time
- For two levels:
 $AMAT = L1\ HT + L1\ MR \times (L2\ HT + L2\ MR \times L2\ MP)$

Question

- **Processor specs:** 1 cycle L1 HT, 2% L1 MR,
5 cycle L2 HT, 5% L2 MR, 100 cycle main memory HT
- Calculate AMAT:
without \$L2, with \$L2
 - A) 3 1.3
 - B) 2 1.2
 - C) 2 1.1
 - D) 3 1.2

Question

- **Processor specs:** 1 cycle L1 HT, 2% L1 MR, 5 cycle L2 HT, 5% L2 MR, 100 cycle main memory HT
- Calculate AMAT:
 - Without L2\$:
$$AMAT_1 = \underline{1 + 0.02 \times 100} = 3$$
 - With L2\$:
$$AMAT_2 = \underline{1 + 0.02 \times (5 + 0.05 \times 100)} = 1.2$$

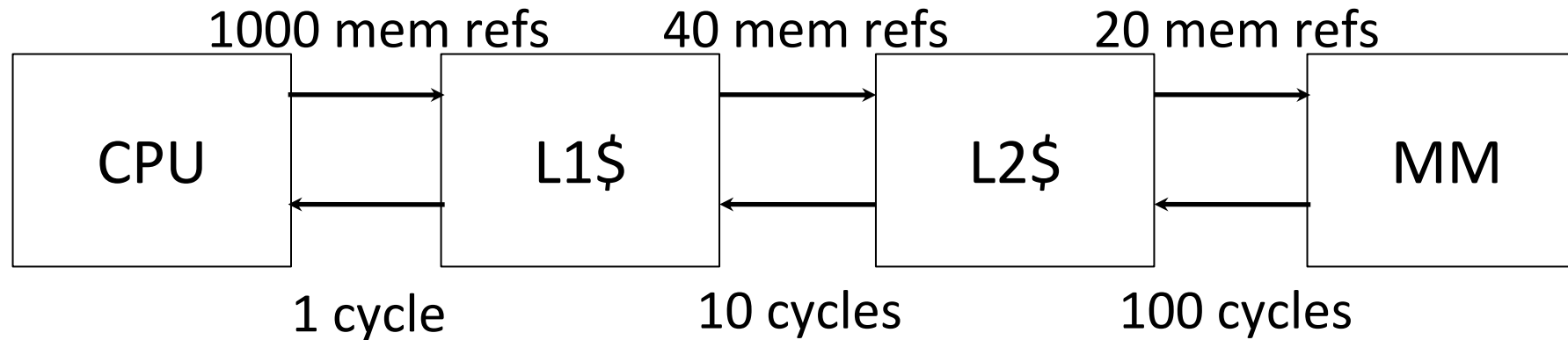
Local vs. Global Miss Rates

- *Local miss rate*: Fraction of references to one level of a cache that miss
 - e.g. L2\$ local MR = L2\$ misses/L1\$ misses
 - Specific to level of caching (as used in AMAT)
- *Global miss rate*: Fraction of all references that miss in all levels of a multilevel cache
 - Property of the overall memory hierarchy
 - Global MR is the **product of all local MRs**
 - Start at $\text{Global MR} = L_n \text{ misses} / L_{n-1} \text{ accesses}$ all multiplied together
 - So by definition, *global MR* \leq *any local MR*

Global Miss Rates

- We may also refer to the global miss rate of a particular level of cache
 - For example Global MR L2
 - This means the fraction of total accesses that miss at L1 and L2
- As a result we can sometimes talk about global miss rates without necessarily involving every level of cache

Local and Global Miss Rates



- For every 1000 CPU-to-memory references
 - 40 will miss in L1\$; what is the local MR? 0.04
 - 20 will miss in L2\$; what is the local MR? 0.5
 - Overall global miss rate? 0.02

Rewriting Performance

- For a two level cache, we know:

$$MR_{\text{global}} = L1\ MR \times L2\ MR$$

- AMAT:

$$- AMAT = L1\ HT + L1\ MR \times (L2\ HT + L2\ MR \times L2\ MP)$$

$$= L1\ HT + L1\ MR \times L2\ HT + MR_{\text{global}} \times L2\ MP$$

- *Aside*: Sometimes might have to convert between global and local MR

$$- L2\ Global\ MR = L2\ Local\ MR \times L1\ MR$$

$$- L2\ Local\ MR = L2\ Global\ MR \div L1\ MR$$

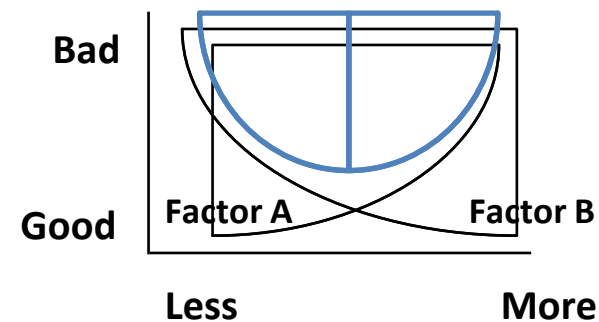
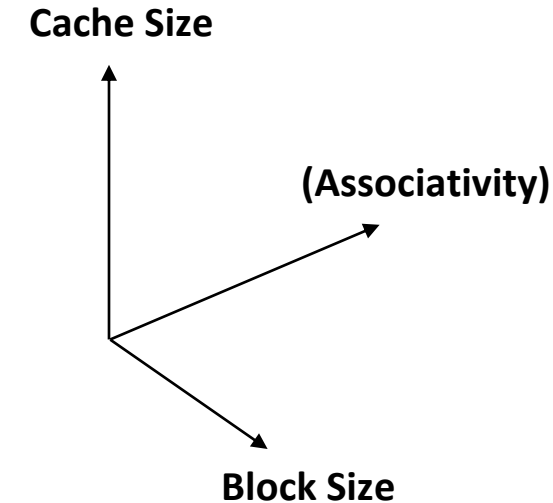
Agenda

- AMAT
- Multilevel Caches
- Improving Cache Performance
- Anatomy of a Cache Question
- Example Cache Questions
- Bonus: Contemporary Cache Specs

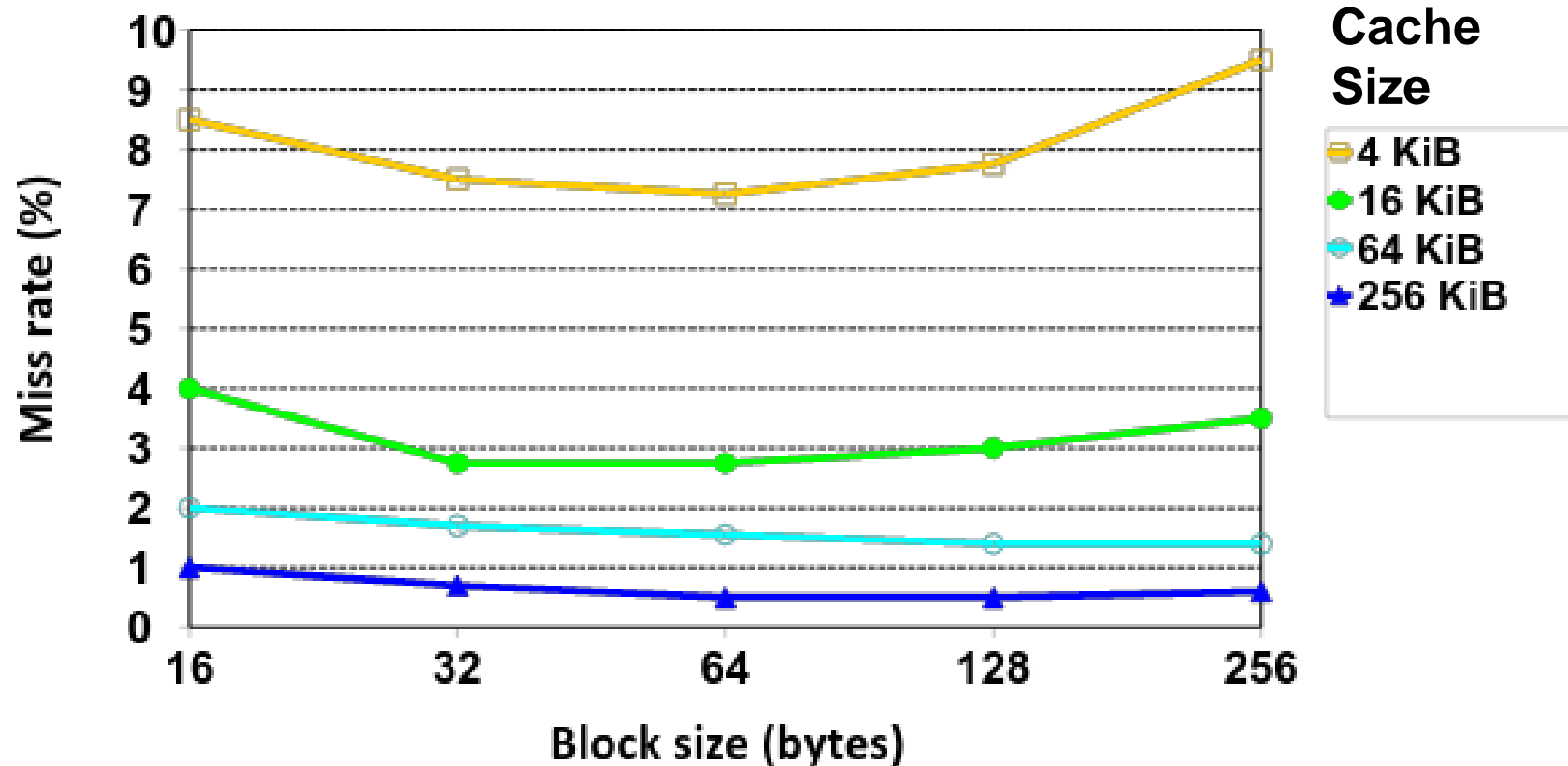
The Cache Design Space

Several interacting dimensions

- Cache parameters:
 - Cache size, Block size, Associativity
- Policy choices:
 - Write-through vs. write-back
 - Replacement policy
- Optimal choice is a compromise
 - Depends on access characteristics
 - Workload and use (I\$, D\$)
 - Depends on technology / cost
- Simplicity often wins

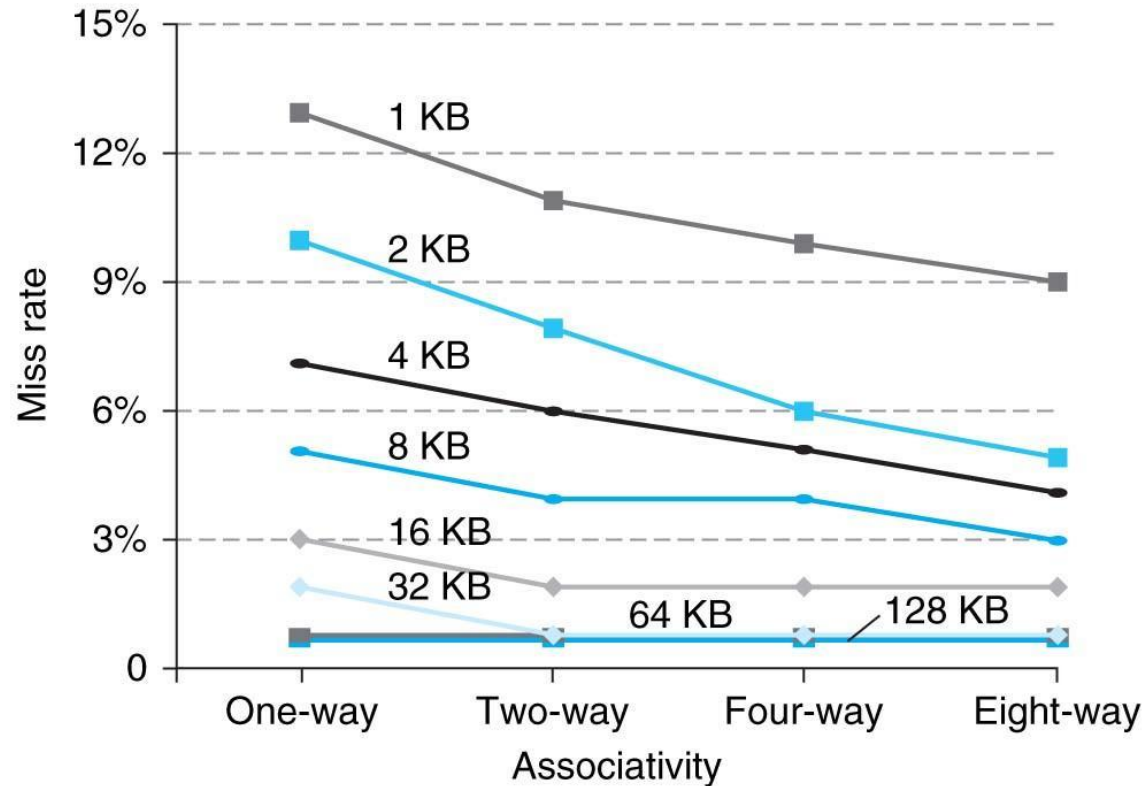


Effect of Block and Cache Sizes on Miss Rate



- Miss rate goes up if the block size becomes a significant fraction of the cache size because the number of blocks that can be held in the same size cache is smaller (increasing capacity misses)

Benefits of Set-Associative Caches



- Consider cost of a miss vs. cost of implementation
- Largest gains are in going from direct mapped to 2-way (20%+ reduction in miss rate)

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 cache associativity	4-way (I), 8-way (D) set associative	2-way set associative
L1 replacement	Approximated LRU replacement	LRU replacement
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 cache associativity	8-way set associative	16-way set associative
L2 replacement	Approximated LRU replacement	Approximated LRU replacement
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 cache associativity	16-way set associative	32-way set associative
L3 replacement	Not Available	Evict block shared by fewest cores
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

Agenda

- AMAT
- Multilevel Caches
- Improving Cache Performance
- **Anatomy of a Cache Question**
- **Example Cache Questions**
- Bonus: Contemporary Cache Specs

Anatomy of a Cache Question

- Cache questions come in a few flavors:
 - 1) TIO Breakdown
 - 2) For fixed cache parameters, analyze the performance of the given code/sequence
 - 3) For fixed cache parameters, find best/worst case scenarios
 - 4) For given code/sequence, how does changing your cache parameters affect performance?
 - 5) AMAT

The Cache

- What are the important cache parameters?
 - Must figure these out from problem description
 - Address size, cache size, block size, associativity, replacement policy
 - Solve for TIO breakdown, # of sets, set size
- Are there multiple levels?
 - Mostly applies to AMAT questions
- What starts in the cache?
 - Not always specified (best/worst case)

Code: Arrays

- Elements stored sequentially in memory
 - Ideal for spatial locality
 - Different arrays not necessarily next to each other
- Remember to account for data size!
 - char is 1 byte, int is 4 bytes
- Pay attention to access pattern
 - Touch *all* elements (e.g. shift, sum)
 - Touch *some* elements (e.g. histogram, stride)
 - How many times do we touch each element?
 - If looping, consider “per iteration” and generalize

Code: Linked Lists/Structs

- Nodes stored separately in memory
 - Addresses of nodes may be very different
 - Type and ordering of linking is important
- Remember to account for size/ordering of struct elements
- Pay attention to access pattern
 - Generally must start from “head”
 - How many struct elements are touched?
 - Again: can you generalise within a loop?

Access Patterns

- How many hits within a single block once it is loaded into cache?
- Will block still be in cache when you revisit its elements?
- Are there special/edge cases to consider?
 - Usually edge of block boundary or edge of cache size boundary
- Sometimes you can generalise a block access pattern for the entire code chunk!

Agenda

- AMAT
- Multilevel Caches
- Improving Cache Performance
- Anatomy of a Cache Question
- **Example Cache Questions**
- Bonus: Contemporary Cache Specs

Example 1 (Sp07 Final)

a) 1 GiB address space, 100 cycles to go to memory. Fill in the following table:

	L1	L2
Cache Size	32 KiB	512 KiB
Block Size	8 B	32 B
Associativity	4-way	Direct-mapped
Hit Time	1 cycle	33 cycles
Miss Rate	10%	2%
Write Policy	Write-through	Write-through
Replacement Policy	LRU	n/a
Tag	17	11
Index	10	14
Offset	3	5
AMAT	AMAT L1 = $1 + 0.1 * 35 = 4.5$	AMAT L2 = $33 + 0.02 * 100 = 35$

Example 1 (Sp07 Final)

Only use L1\$: **C** = 32 KiB, **K** = 8 B, **N** = 4, LRU, write-through
char A[] is block aligned and SIZE = 32 MiB

```
char *A = (char *) malloc (SIZE*sizeof(char));  
/* number of STRETCHes */  
for (i = 0; i < (SIZE/STRETCH); i++) {  
    /* go up to STRETCH */  
    for(j=0;j<STRETCH;j++)      sum  += A[i*STRETCH+j];  
    /* down from STRETCH */  
    for(j=STRETCH-1;j>=0;j--)  prod *= A[i*STRETCH+j];  
}
```

- 2nd inner for loop hits same indices as 1st inner for loop, but in reverse order
- Always traverse full SIZE, regardless of STRETCH

Example 1 (Sp07 Final)

Only use L1\$: **C** = 32 KiB, **K** = 8 B, **N** = 4, LRU, write-through

char A[] is block aligned and SIZE = 32 MiB

```
char *A = (char *) malloc (SIZE*sizeof(char));  
for (i = 0; i < (SIZE/STRETCH); i++) {  
    for(j=0;j<STRETCH;j++)      sum  += A[i*STRETCH+j];  
    for(j=STRETCH-1;j>=0;j--)  prod *= A[i*STRETCH+j];  
}
```

- b) As we double our STRETCH from 1 to 2 to 4 (...etc), we notice the number of cache misses doesn't change! What is the largest value of STRETCH *before* cache misses changes? (Use IEC)

32 KiB, when STRETCH exactly equals C

Example 1 (Sp07 Final)

Only use L1\$: **C** = 32 KiB, **K** = 8 B, **N** = 4, LRU, write-through

char A[] is block aligned and SIZE = 32 MiB

```
char *A = (char *) malloc (SIZE*sizeof(char));
for (i = 0; i < (SIZE/STRETCH); i++) {
    for (j=0; j<STRETCH; j++)      sum += A[i*STRETCH+j];
    for (j=STRETCH-1; j>=0; j--)  prod *= A[i*STRETCH+j];
}
```

c) If we double our STRETCH from (b), what is the ratio of cache *hits* to *misses*?

Now STRETCH = 64 KiB. Moving sequentially by byte, so each block for entire 1st inner loop has 1 miss and 7 hits (7:1). Upper half of STRETCH lives in cache, so first half of 2nd inner loop is 8 hits/block (8:0). Second half is as before (7:1).

Example 1 (Sp07 Final)

Only use L1\$: **C** = 32 KiB, **K** = 8 B, **N** = 4, LRU, write-through

char A[] is block aligned and SIZE = 32 MiB

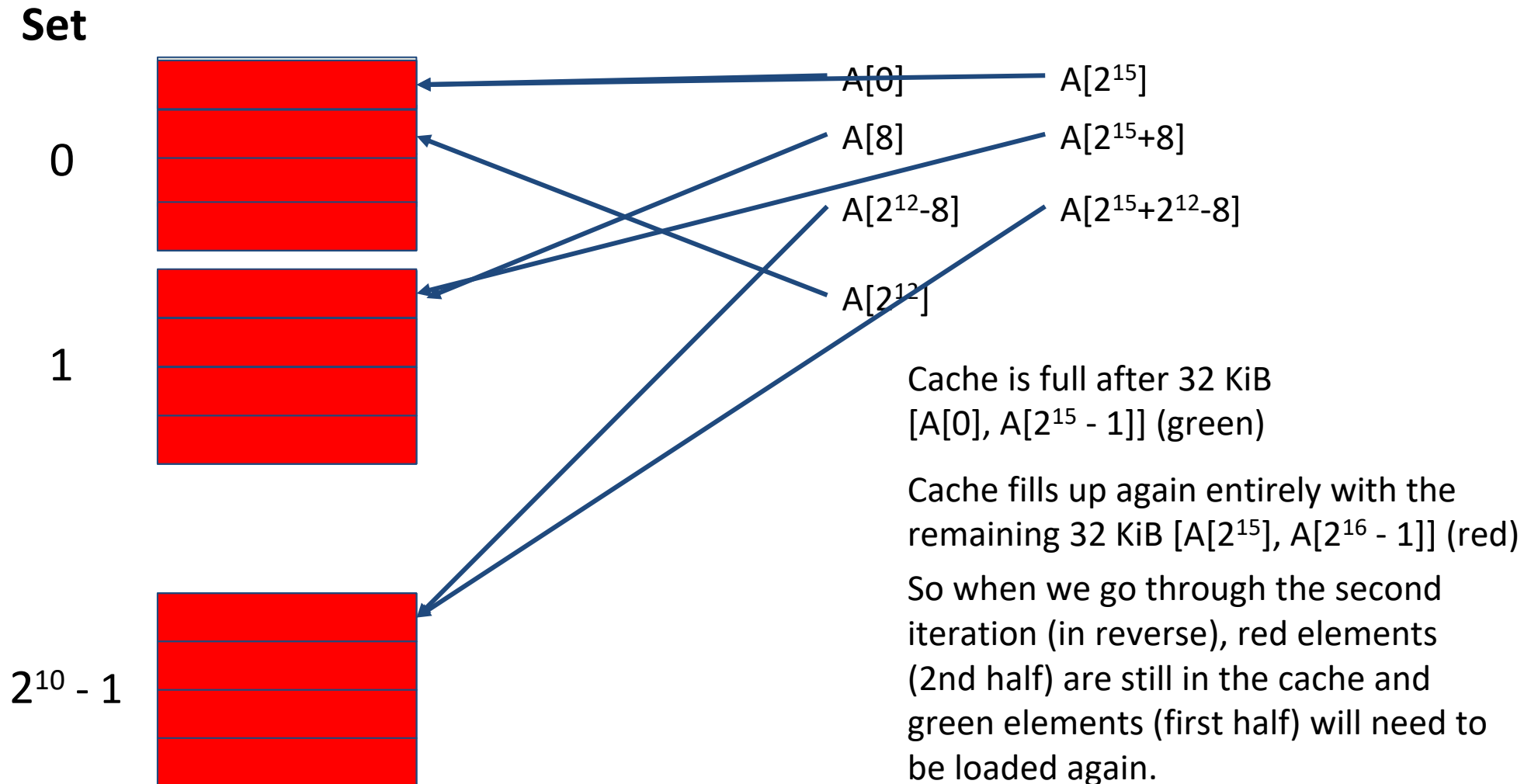
```
char *A = (char *) malloc (SIZE*sizeof(char));
for (i = 0; i < (SIZE/STRETCH); i++) {
    for (j=0; j<STRETCH; j++)      sum  += A[i*STRETCH+j];
    for (j=STRETCH-1; j>=0; j++)  prod += A[i*STRETCH+j];
}
```

c) If we double our STRETCH from (b), what is the ratio of cache *hits* to *misses*?

Considering the equal-sized chunks of half of each inner for loop, we have loop 1 1st (7:1), loop 1 2nd (7:1), loop 2 1st (8:0), and loop 2 2nd (7:1).

$$7+7+8+7:1+1+0+1 = \mathbf{29:3}$$

Example 1 (But visual)



Example 2 (Sp13 Final)

32-bit RISC-V, 4 GiB memory, single L1\$ of size **C** with block size **K** (**C** \geq **K** and C is a power of 2).

A, **B** are arrays in different places of memory of equal size **n** (power of 2 and a [natural #] multiple of **C**), block aligned.

```
// sizeof(uint8_t) = 1
SwapLeft(uint8_t *A, uint8_t *B, int n) {
    uint8_t tmp;
    for (int i = 0; i < n; i++) {
        tmp = A[i];
        A[i] = B[i];
        B[i] = tmp;
    }
}
```

Array data size is 1 byte

Do **n** times:

← Read A[i]

← Read B[i], Write A[i]

← Write B[i]

Example 2 (Sp13 Final)

32-bit RISC-V, 4 GiB memory, single L1\$ of size C with block size K ($C \geq K$ and C is a power of 2).

A, **B** are arrays in different places of memory of equal size n (power of 2 and a [natural #] multiple of C), block aligned.

a) If the cache is direct-mapped and the *best* hit:miss ratio is “H:1”, what is the block size in bytes?

Best case is $A[i]$ and $B[i]$ DON'T map to same slot.

Use every value of $i \in [0, n)$ only once.

Rd A, Rd B, Wr A, Wr B \rightarrow Miss, Miss, Hit, Hit (1st time)

\rightarrow Hit, Hit, Hit, Hit ($K-1$ times in block)

Per block:

$$4*(K-1)+2:2 = 4K-2:2 = 2K-1:1 = H:1 \rightarrow K = (H+1)/2$$

Example 2 (Sp13 Final)

32-bit RISC-V, 4 GiB memory, single L1\$ of size C with block size K ($C \geq K$ and C is a power of 2).

A, **B** are arrays in different places of memory of equal size n (power of 2 and a [natural #] multiple of C), block aligned.

b) What is the *worst* hit:miss ratio?

Worst case is $A[i]$ and $B[i]$ map to same slot (conflict).

Rd A, Rd B, Wr A, Wr B \rightarrow Miss, Miss, Miss, Miss (all times)

because blocks keep replacing each other

0:1 (or 0:<anything>)

Example 2 (Sp13 Final)

- c) Fill in code for SwapRight so that it does the same thing as SwapLeft but improves the (b) hit:miss ratio.

```
SwapRight(uint8_t *A, uint8_t *B, int n) {  
    uint8_t tmpA, tmpB;  
    for (int i = 0; i < n; i++) {  
        tmpA = A[i];           ← Read A[i]  
        tmpB = B[i];           ← Read B[i]  
        B[i] = tmpA;           ← Write B[i]  
        A[i] = tmpB;           ← Write A[i]  
    }  
}
```

Example 2 (Sp13 Final)

32-bit RISC-V, 4 GiB memory, single L1\$ of size C with block size K ($C \geq K$ and C is a power of 2).

A, **B** are arrays in different places of memory of equal size n (power of 2 and a [natural #] multiple of C), block aligned.

d) What is the *worst* hit:miss ratio for SwapRight?

Worst case is $A[i]$ and $B[i]$ map to same slot (conflict).

Rd A, Rd B, Wr B, Wr A \rightarrow Miss, Miss, Hit, Miss (1st time)

\rightarrow Hit, Miss, Hit, Miss ($K-1$ times)

Per block:

$$(K-1)*2+1:(K-1)*2+3 = \textcolor{red}{2K-1:2K+1}$$

Example 2 (Sp13 Final)

e) Change the cache to be **2-way set-associative**. Cache size **C**, block size **K**. What is the *worst* hit:miss ratio for SwapLeft with the following replacement policy?

- LRU and an empty cache

Even if $A[i]$ and $B[i]$ map to same set, they can both co-exist.

Rd A, Rd B, Wr A, Wr B \rightarrow Miss, Miss, Hit, Hit (1^{st} time)

\rightarrow Hit, Hit, Hit, Hit ($K-1$ times in block)

So **$2K-1:1$** (from part (a))

Summary

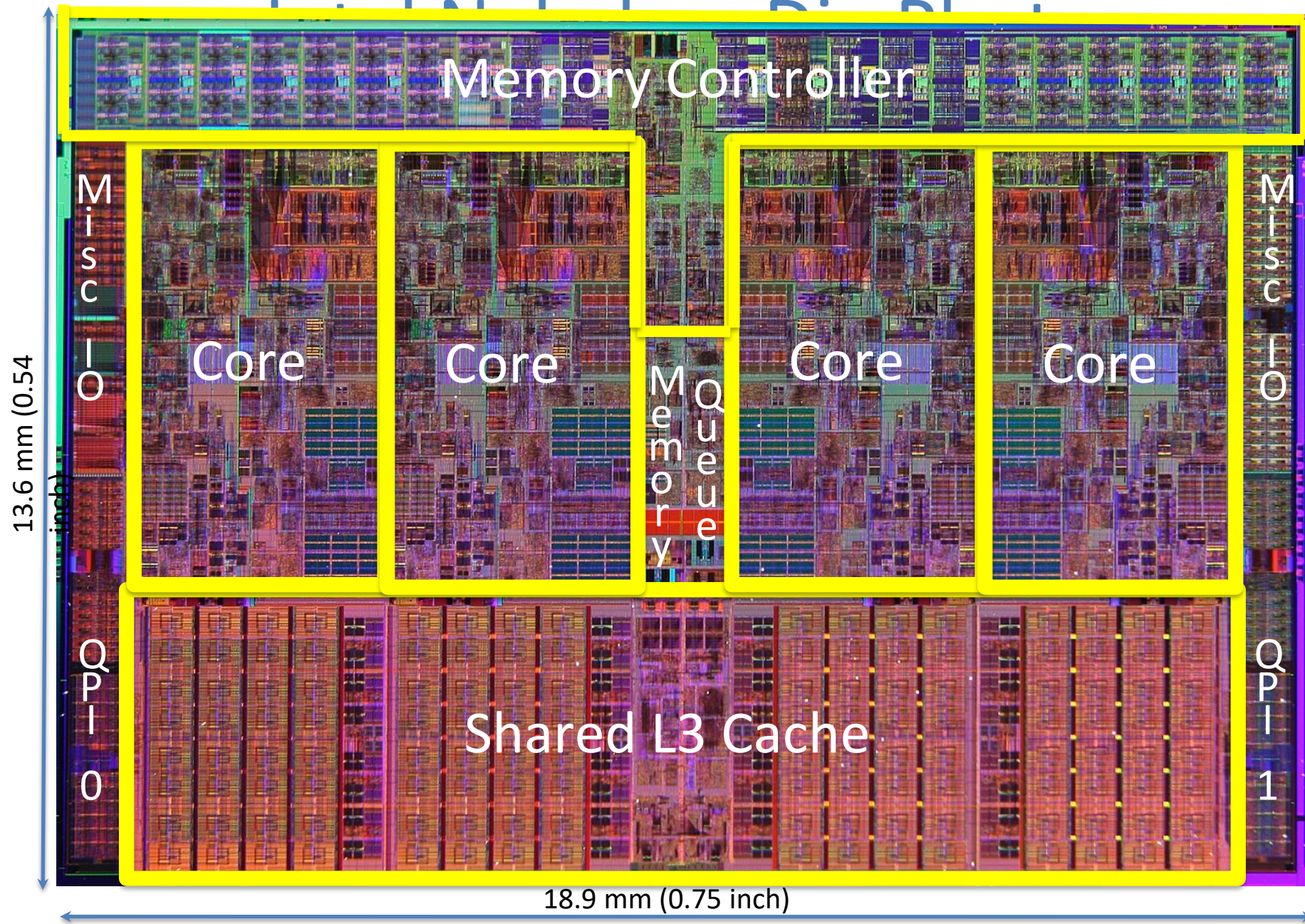
- Multilevel caches reduce *miss penalty*
 - Local vs. global miss rate
 - Optimize first level to be fast (low HT)
 - Optimize lower levels to not miss (minimize MP)
- Cache performance depends heavily on cache design (there are many choices)
 - Effects of parameters and policies
 - Cost vs. effectiveness
- Cache problems are hard!

BONUS SLIDES

You are NOT responsible for the material contained on the following slides, and we may not have enough time to get to them in lecture. They are good to look at if you have free time. They have been prepared in a way that should be easily readable.

Agenda

- Multilevel Caches
- Improving Cache Performance
- Anatomy of a Cache Question
- Example Cache Questions
- **Bonus: Contemporary Cache Specs**



Core Area Breakdown

32KiB I\$ per core
32KiB D\$ per core
512KiB L2\$ per core
Share one 8-MiB L3\$

