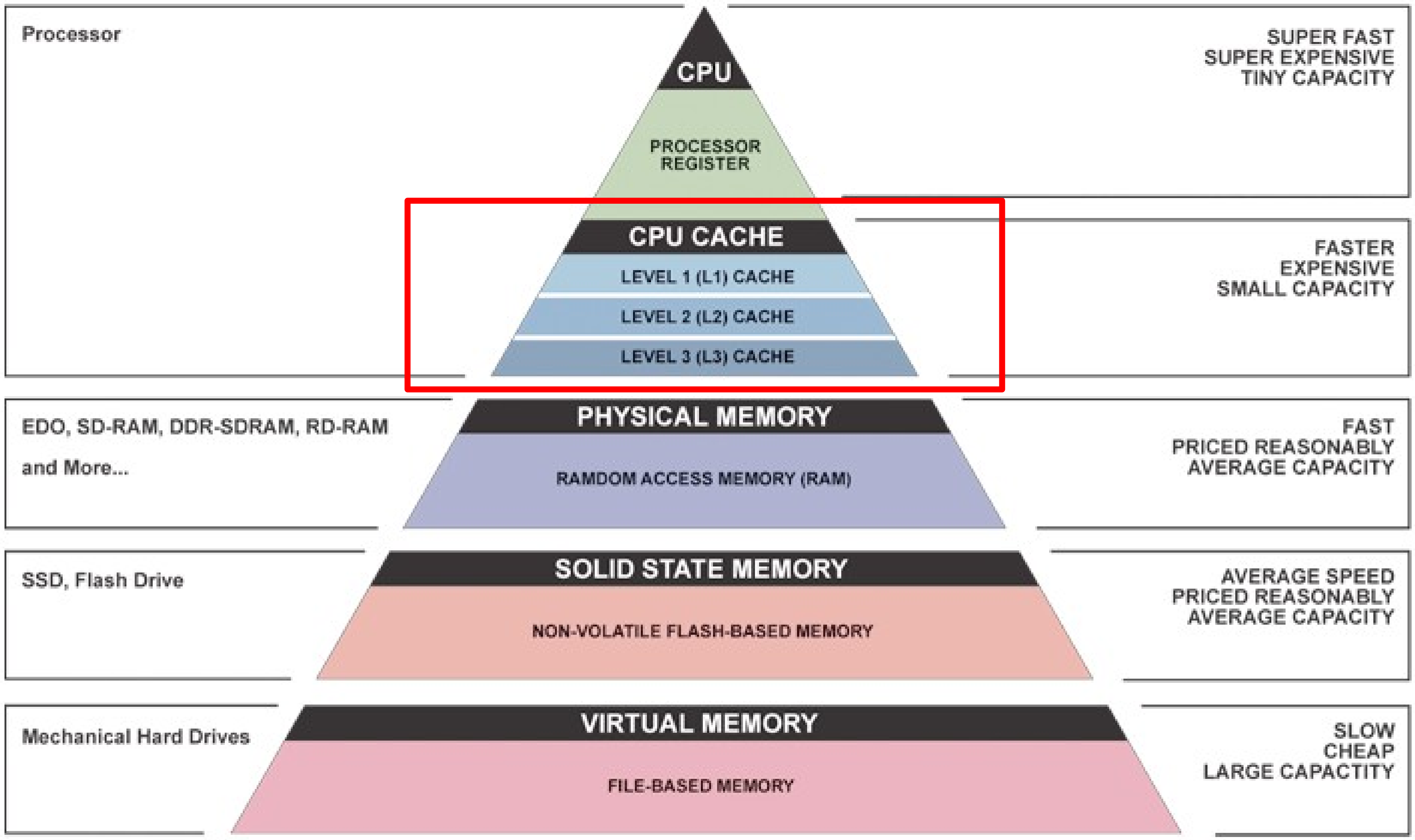
CS 61C: Great Ideas in Computer Architecture

Direct-Mapped Caches, Set Associative Caches, Cache Performance

Instructor: Alan Christopher

Great Idea #3: Principle of Locality/ Memory Hierarchy



Extended Review of Last Lecture

- Why have caches?
  - Intermediate level between CPU and memory
  - In-between in *size*, *cost*, and *speed*
- Memory (hierarchy, organization, structures) set up to exploit *temporal* and *spatial locality*
  - *Temporal:* If accessed, will access again soon
  - *Spatial:* If accessed, will access others around it
- Caches hold a subset of memory (in *blocks*)
  - We are studying how they are designed for fast and efficient operation (lookup, access, storage)

Extended Review of Last Lecture

- Fully Associative Caches:
  - Every block can go in any slot
    - Use random or LRU replacement policy when cache full
  - Memory address breakdown (on request)
    - Tag field is identifier (which block is currently in slot)
    - Offset field indexes into block
  - *Each* cache slot holds block data, tag, valid bit, and dirty bit (dirty bit is only for *write-back*)
    - The whole cache maintains LRU bits

Extended Review of Last Lecture

- Cache read and write policies:
  - Affect consistency of data between cache and memory
  - *Write-back* vs. *write-through*
  - *Write allocate* vs. *no-write allocate*
- On memory access (read or write):
  - Look at ALL cache slots in parallel
  - If Valid bit is 0, then ignore
  - If Valid bit is 1 and Tag matches, then use that data
- write, set Dirty bit if write-back

Extended Review of Last Lecture

256 B address space

cache size (C)

block size (K)

- Fully associative cache layout
  - 8-bit address space, 32-byte cache with 8-byte blocks
  - LRU replacement (5 bits), write-back and write allocate
  - Offset – 3 bits, Tag – 5 bits

Need dirty bit

Offset

Slot	V	D	Tag	000	001	010	011	100	101	110	111
0	X	X	XXXXX	0x??	0x??	0x??	0x??	0x??	0x??	0x??	0x??
1	X	X	XXXXX	0x??	0x??	0x??	0x??	0x??	0x??	0x??	0x??
2	X	X	XXXXX	0x??	0x??	0x??	0x??	0x??	0x??	0x??	0x??
3	X	X	XXXXX	0x??	0x??	0x??	0x??	0x??	0x??	0x??	0x??

LRU
XXXXX

- Each slot has 71 bits; cache has 4*71+5 = 289 bits

Agenda

- Direct-Mapped Caches
- Administrivia
- Set Associative Caches
- Cache Performance

Direct-Mapped Caches (1/3)

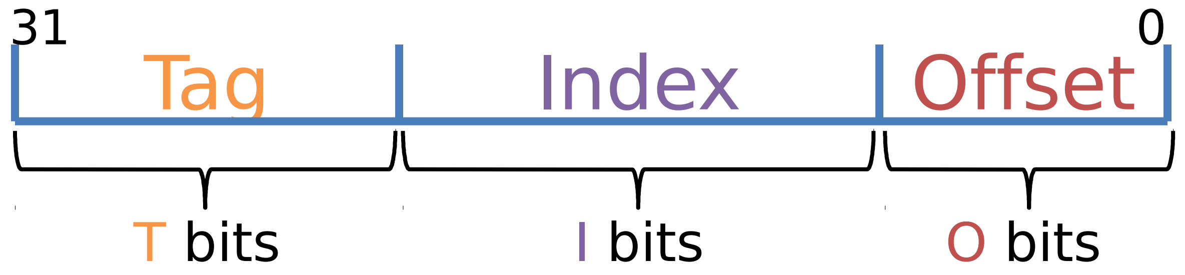
- Each memory block is mapped to *exactly one slot* in the cache (*direct-mapped*)
  - Every block has only one "home"
  - Use hash function to determine which slot
- Comparison with fully associative
  - Check just one slot for a block (faster!)
  - No replacement policy necessary
  - Access pattern may leave empty slots in cache

Direct-Mapped Caches (2/3)

- Offset field remains the same as before
- **Recall:** blocks consist of adjacent bytes
  - Do we want adjacent blocks to map to same slot?
  - Index field: Apply hash function to block address to determine *which slot* the block goes in
    - (*block* address) modulo (# of *blocks* in the cache)
- Tag field maintains same function (identifier), but is now shorter

TIO Address Breakdown

- Memory address fields:



- Meaning of the field sizes:
  - O bits ↔ 2^{O} bytes/block = 2^{O-2} words/block
  - I bits ↔ 2^{I} slots in cache = cache size / block size
  - T bits = A – I – O, where A = # of address bits (A = 32 here)

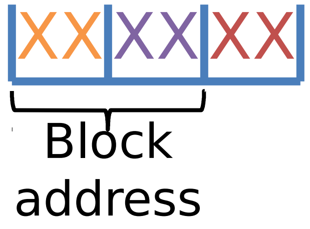
Direct-Mapped Caches (3/3)

- What's actually in the cache?
  - Block of data ($8 \times K = 8 \times 2^{O}$ bits)
  - Tag field of address as identifier (T bits)
  - Valid bit (1 bit)
  - Dirty bit (1 bit if write-back)
  - No replacement management bits!
- Total bits in cache = # slots $\times$ ($8\times K + T + 1 + 1$)
  $= 2^{I} \times (8\times 2^{O} + T + 1 + 1)$ bits

DM Cache Example (1/5)

- Cache parameters:
  - Direct-mapped, address space of 64B, block size of 1 word, cache size of 4 words, write-through

Memory Addresses:



- TIO Breakdown:
  - 1 word = 4 bytes, so O = $\log_2(4)$ = 2
  - Cache size / block size = 4, so I = $\log_2(4)$ = 2
  - A = $\log_2(64)$ = 6 bits, so T = 6 – 2 – 2 = 2
- Bits in cache = $2^2 \times (8\times2^2 + 2 + 1) = 140$ bits

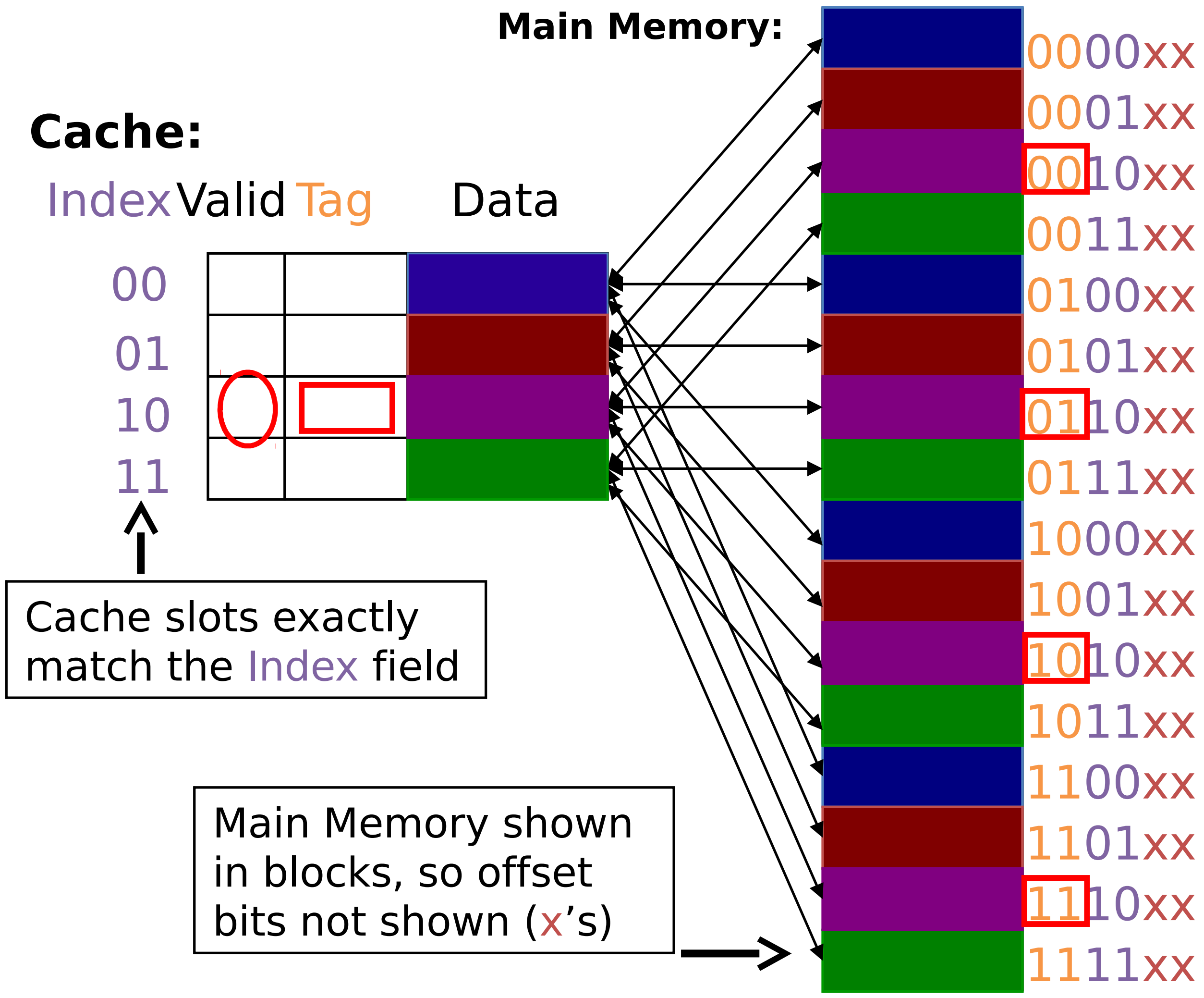
DM Cache Example (2/5)

- Cache parameters:
  - Direct-mapped, address space of 64B, block size of 1 word, cache size of 4 words, write-through
  - Offset – 2 bits, Index – 2 bits, Tag – 2 bits

				Offset		
	V	Tag	00	01	10	11
00	X	XX	0x??	0x??	0x??	0x??
Index 01	X	XX	0x??	0x??	0x??	0x??
10	X	XX	0x??	0x??	0x??	0x??
11	X	XX	0x??	0x??	0x??	0x??

- 35 bits per index/slot, 140 bits to implement

DM Cache Example (3/5)



Cache slots exactly match the Index field

Main Memory shown in blocks, so offset bits not shown (x's)

Which blocks map to each row of the cache? (see colors)

On a memory request: (let's say 001011_2)

1) Take Index field (10)

2) Check if Valid bit is true in that row of cache

3) If valid, then check if Tag matches

DM Cache Example (4/5)

- Consider the sequence of memory address accesses

Starting with a cold cache: 0 2 4 8 20 16 0 2

0 miss

00	1	00	M[0]	M[1]	M[2]	M[3]
01	0	00	0x??	0x??	0x??	0x??
10	0	00	0x??	0x??	0x??	0x??
11	0	00	0x??	0x??	0x??	0x??

2 hit

00	1	00	M[0]	M[1]	M[2]	M[3]
01	0	00	0x??	0x??	0x??	0x??
10	0	00	0x??	0x??	0x??	0x??
11	0	00	0x??	0x??	0x??	0x??

4 miss

00	1	00	M[0]	M[1]	M[2]	M[3]
01	1	00	M[4]	M[5]	M[6]	M[7]
10	0	00	0x??	0x??	0x??	0x??
11	0	00	0x??	0x??	0x??	0x??

8 miss

00	1	00	M[0]	M[1]	M[2]	M[3]
01	1	00	M[4]	M[5]	M[6]	M[7]
10	1	00	M[8]	M[9]	M[10]	M[11]
11	0	00	0x??	0x??	0x??	0x??

DM Cache Example (5/5)

- Consider the sequence of memory address accesses

Starting with a cold cache: 0 2 4 8 20 16 0 2

M H M M

20 miss

00	1	00	M[0]	M[1]	M[2]	M[3]
01	1	~~00~~	~~M[4]~~	~~M[5]~~	~~M[6]~~	~~M[7]~~
10	1	00	M[8]	M[9]	M[10]	M[11]
11	0	00	0x??	0x??	0x??	0x??

16 miss

00	1	~~00~~	~~M[0]~~	~~M[1]~~	~~M[2]~~	~~M[3]~~
01	1	01	M[20]	M[21]	M[22]	M[23]
10	1	00	M[8]	M[9]	M[10]	M[11]
11	0	00	0x??	0x??	0x??	0x??

0 miss

00	1	~~01~~	~~M[16]~~	~~M[17]~~	~~M[18]~~	~~M[19]~~
01	1	01	M[20]	M[21]	M[22]	M[23]
10	1	00	M[8]	M[9]	M[10]	M[11]
11	0	00	0x??	0x??	0x??	0x??

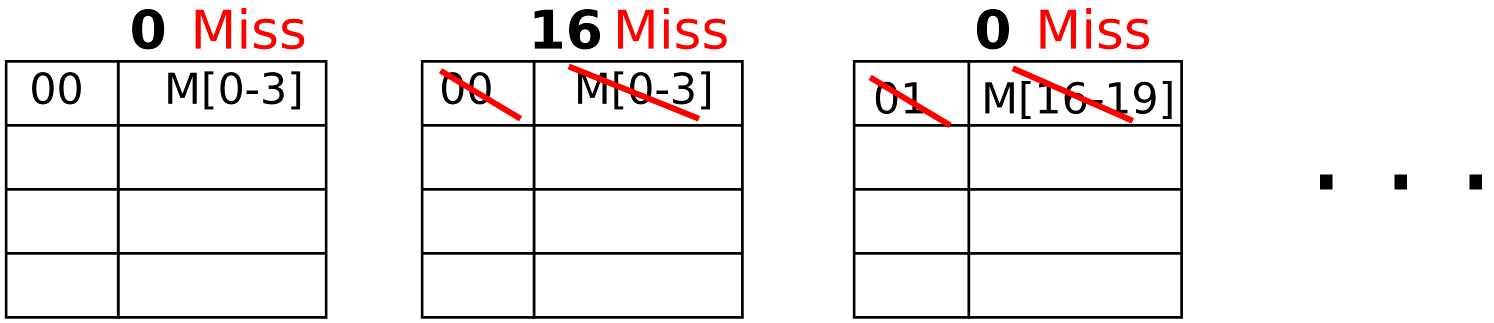
2 hit

00	1	00	M[0]	M[1]	M[2]	M[3]
01	1	01	M[20]	M[21]	M[22]	M[23]
10	1	00	M[8]	M[9]	M[10]	M[11]
11	0	00	0x??	0x??	0x??	0x??

- 8 requests, 6 misses – last slot was never used!

Worst-Case for Direct-Mapped

- Cold DM $ that holds 4 1-word blocks
- Consider the memory accesses: 0, 16, 0, 16,...



- HR of 0%
  - Ping pong effect: alternating requests that map into the same cache slot
- Does fully associative have this problem?

Comparison So Far

- Fully associative
  - Block can go into *any* slot
  - Must check ALL cache slots on request (“slow”)
  - TO breakdown (i.e. I = 0 bits)
  - “Worst case” still fills cache (more efficient)
- Direct-mapped
  - Block goes into *one specific* slot (set by Index field)
  - Only check ONE cache slot on request (“fast”)
  - TIO breakdown
  - “Worst case” may only use 1 slot (less efficient)

Agenda

- Direct-Mapped Caches
- Administrivia
- Set Associative Caches
- Cache Performance

Administrivia

- Proj1 still due Sunday
  - My OH tomorrow “go until they finish (sorta)”
    - As long as there's a student with pertinent questions I'll hang around.
    - I won't stay later than 7pm (go to Andrew's office hours at that point)
- HW4 will be released Friday, due next Sunday

Agenda

- Direct-Mapped Caches
- Administrivia
- Set Associative Caches
- Cache Performance

Set Associative Caches

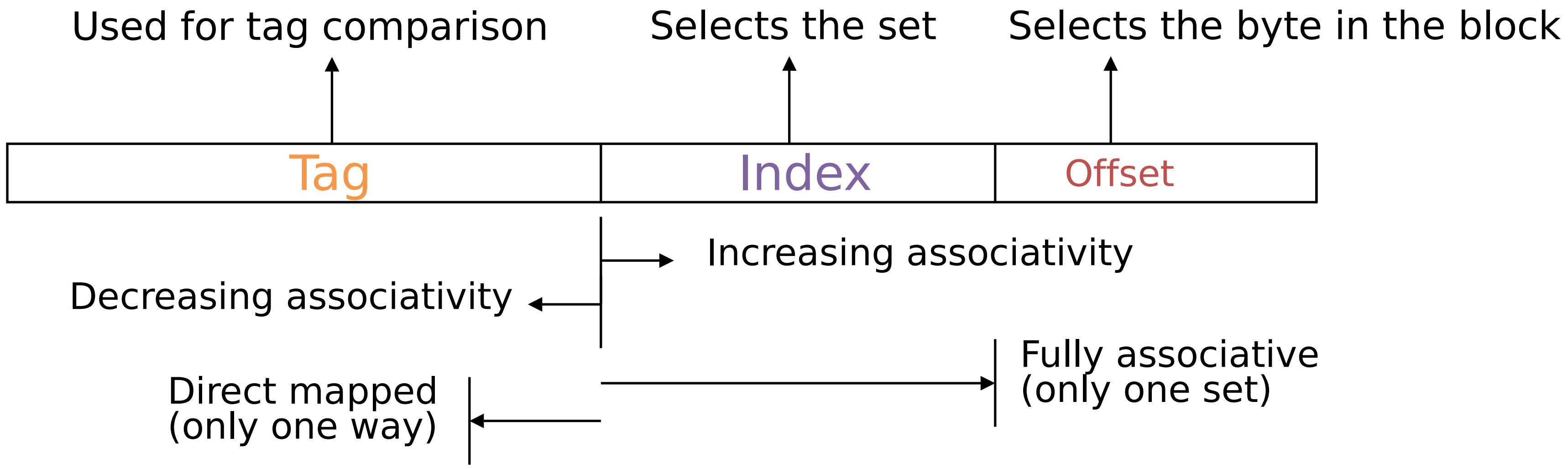
- Compromise!
  - More flexible than DM, more structured than FA
- *N-way set-associative:* Divide $ into sets, each of which consists of N slots
  - Memory block maps to a set determined by Index field and is placed in any of the N slots of that set
  - Call N the *associativity*
  - New hash function:
    (block address) modulo (# *sets* in the cache)
  - Replacement policy applies to every *set*

Effect of Associativity on TIO (1/2)

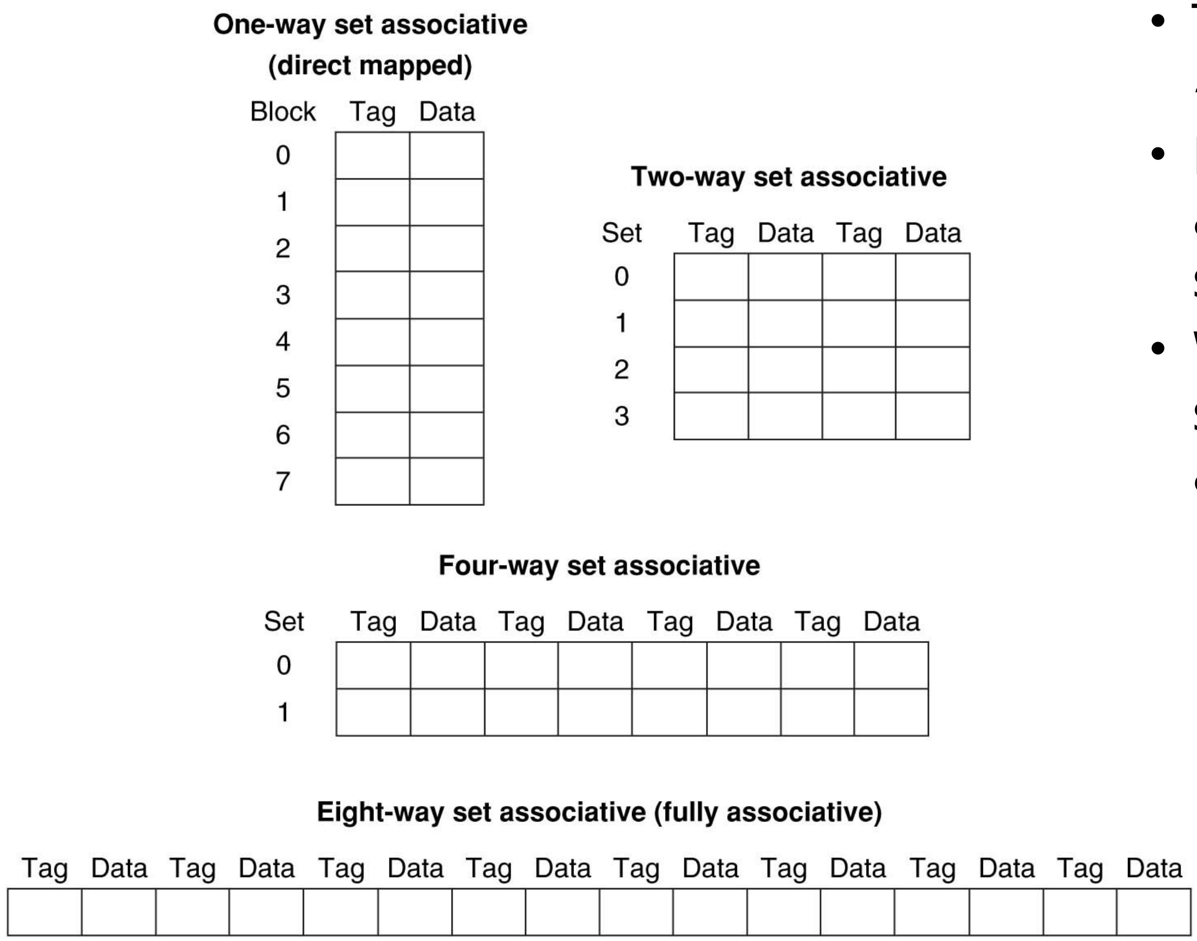
- Here we assume a cache of fixed size (C), fixed block size (K)
- **Offset:** Points to a byte in a block (same as before)
- **Index:** Instead of pointing to a *slot*, now points to a *set*, so $I = \log_2(C/K/N)$
  - Fully associative (1 set): 0 Index bits!
  - Direct-mapped (N = 1): max Index bits
  - Set associative: somewhere in-between
- **Tag:** Remaining identifier bits ($T = A - I - O$)

Effect of Associativity on TIO (2/2)

- For a fixed-size cache, each increase by a factor of two in associativity doubles the number of blocks per set (i.e. the number of slots) and halves the number of sets – decreasing the size of the Index by 1 bit and increasing the size of the Tag by 1 bit



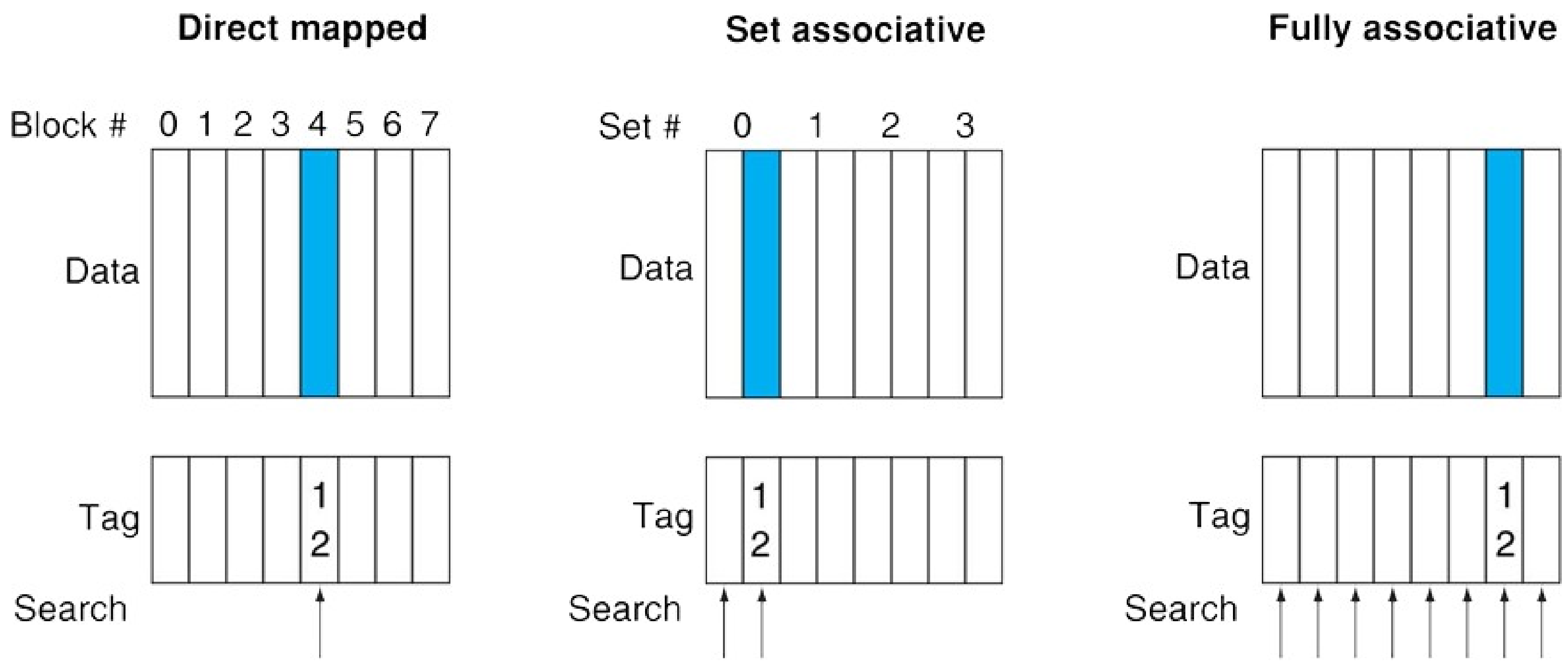
Example: Eight-Block Cache Configs


One-way set associative (direct mapped)
Block
Tag
Data
0
1
2
3
4
5
6
7
Two-way set associative
Set
Tag
Data
Tag
Data
0
1
2
3
Four-way set associative
Set
Tag Data Tag Data Tag Data Tag Data
0
1
Eight-way set associative (fully associative)
Tag Data Tag Data Tag Data Tag Data Tag Data Tag Data Tag Data Tag Data

- Total size of \$ = *# sets × associativity*
- For fixed \$ size, associativity ↑ means # sets ↓ and slots per set ↑
- With 8 blocks, an 8-way set associative \$ is same as a fully associative \$

Block Placement Schemes

- Place memory block 12 in a cache that holds 8 blocks

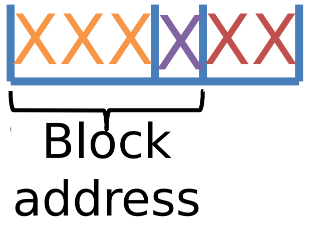


- **Fully associative:** Can go in *any* of the slots (all 1 set)
- **Direct-mapped:** Can only go in slot (12 mod 8) = 4
- **2-way set associative:** Can go in either slot of set (12 mod 4) = 0

SA Cache Example (1/5)

- Cache parameters:
  - 2-way set associative, 6-bit addresses, 1-word blocks, 4-word cache, write-through
- How many sets?
  - C/K/N = 4/1/2 = 2 sets
- TIO Breakdown:
  - O = $\log_2(4)$ = 2, I = $\log_2(2)$ = 1,
  - T = 6 – 1 – 2 = 3

Memory Addresses:



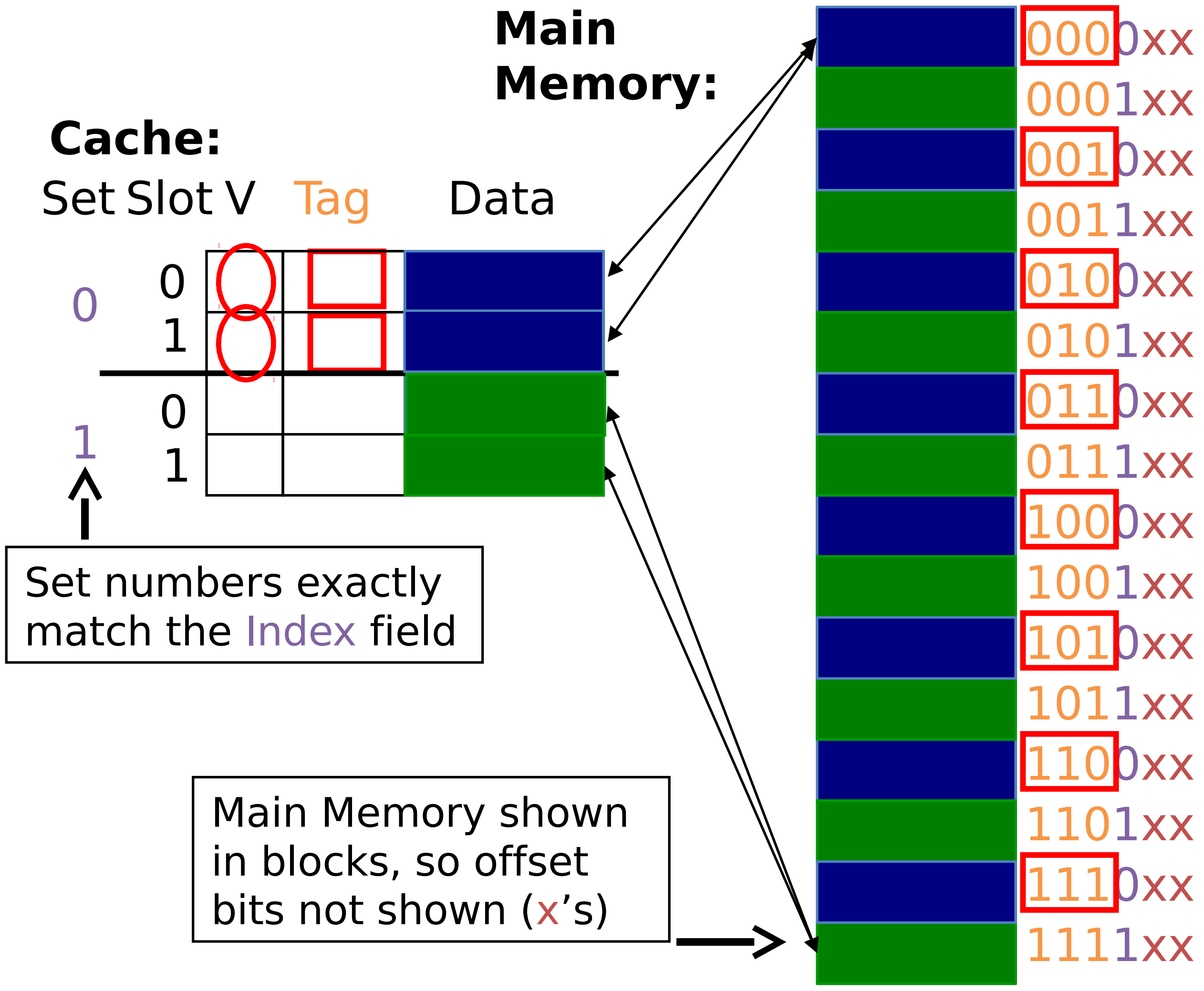
SA Cache Example (2/5)

- Cache parameters:
  - 2-way set associative, 6-bit addresses, 1-word blocks, 4-word cache, write-through
  - Offset – 2 bits, Index – 1 bit, Tag – 3 bits

Index				Offset				
		V	Tag	00	01	10	11	LRU
0	0	X	XXX	0x??	0x??	0x??	0x??	X
	1	X	XXX	0x??	0x??	0x??	0x??	
1	0	X	XXX	0x??	0x??	0x??	0x??	X
	1	X	XXX	0x??	0x??	0x??	0x??	

- 36 bits per slot, 36*2+1 = 73 bits per set, 2*73 = 146 bits to implement

SA Cache Example (3/5)



Set numbers exactly match the Index field

Main Memory shown in blocks, so offset bits not shown (x's)

Each block maps into one set (either slot) (see colors)

On a memory request:
(let's say 001011$_{two}$)

1) Take Index field (0)

2) For EACH slot in set, check valid bit, then compare Tag

SA Cache Example (4/5)

- Consider the sequence of memory address accesses

Starting with a cold cache: 0 2 4 8 20 16 0 2

0 miss

0	0	1	000	M[0]	M[1]	M[2]	M[3]
	1	0	000	0x??	0x??	0x??	0x??
1	0	0	000	0x??	0x??	0x??	0x??
	1	0	000	0x??	0x??	0x??	0x??

2 hit

0	0	1	000	M[0]	M[1]	M[2]	M[3]
	1	0	000	0x??	0x??	0x??	0x??
1	0	0	000	0x??	0x??	0x??	0x??
	1	0	000	0x??	0x??	0x??	0x??

4 miss

0	0	1	000	M[0]	M[1]	M[2]	M[3]
	1	0	000	0x??	0x??	0x??	0x??
1	0	1	000	M[4]	M[5]	M[6]	M[7]
	1	0	000	0x??	0x??	0x??	0x??

8 miss

0	0	1	000	M[0]	M[1]	M[2]	M[3]
	1	1	001	M[8]	M[9]	M[10]	M[11]
1	0	1	000	M[4]	M[5]	M[6]	M[7]
	1	0	000	0x??	0x??	0x??	0x??

SA Cache Example (5/5)

- Consider the sequence of memory address accesses

Starting with a cold cache: 0 2 4 8 20 16 0 2

M H M M

20 miss

0	0	1	000	M[0]	M[1]	M[2]	M[3]
	1	1	001	M[8]	M[9]	M[10]	M[11]
1	0	1	000	M[4]	M[5]	M[6]	M[7]
	1	1	010	M[20]	M[21]	M[22]	M[23]

16 miss

0	0	1	~~000~~	~~M[0]~~	~~M[1]~~	~~M[2]~~	~~M[3]~~
	1	1	001	M[8]	M[9]	M[10]	M[11]
1	0	1	000	M[4]	M[5]	M[6]	M[7]
	1	1	010	M[20]	M[21]	M[22]	M[23]

0 miss

0	0	1	010	M[16]	M[17]	M[18]	M[19]
	1	1	~~001~~	~~M[8]~~	~~M[9]~~	~~M[10]~~	~~M[11]~~
1	0	1	000	M[4]	M[5]	M[6]	M[7]
	1	1	010	M[20]	M[21]	M[22]	M[23]

2 hit

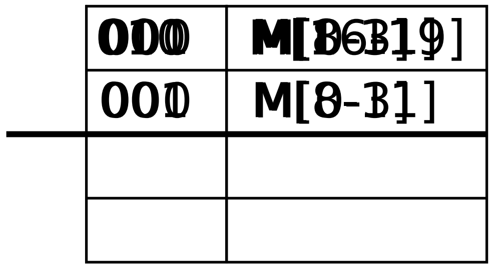
0	0	1	010	M[16]	M[17]	M[18]	M[19]
	1	1	000	M[0]	M[1]	M[2]	M[3]
1	0	1	000	M[4]	M[5]	M[6]	M[7]
	1	1	010	M[20]	M[21]	M[22]	M[23]

- 8 requests, 6 misses

Worst Case for Set Associative

- Worst case for DM was repeating pattern of 2 into same cache slot (HR = 0/n)
  - Set associative for N > 1: HR = (n-2)/n
- Worst case for N-way SA with LRU?
  - Repeating pattern of at least N+1 that maps into same set
  - Back to HR = 0: 0, 8, 16, 0, 8, …

M M M M M



Question: What is the TIO breakdown for the following cache?

- 32-bit address space
- 32 KiB 4-way set associative cache
- 8 word blocks

	T	I	O
(B)	21	8	3
(G)	19	8	5
(P)	19	10	3
(Y)	17	10	5

Technology Break

Agenda

- Direct-Mapped Caches
- Administrivia
- Set Associative Caches
- Cache Performance

Cache Performance

- Two things hurt the performance of a cache:
  - Miss rate and miss penalty
  - *Average Memory Access Time* (AMAT): average time to access memory considering both hits and misses

  AMAT = Hit time + Miss rate × Miss penalty

  (abbreviated AMAT = HT + MR × MP)

  - **Goal 1:** Examine how changing the different cache parameters affects our AMAT
  - **Goal 2:** Examine how to optimize your code for better cache performance

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 × 50 = 2 clock cycles = 400 ps

  - Which improvement would be best?
    - 190 ps clock 380 ps
    - MP of 40 clock cycles 360 ps
    - MR of 0.015 misses/instruction 350 ps

Cache Parameter Effects

- What is the potential impact of much larger cache on AMAT? (same block size)
  - Increase HR
  - Longer HT: smaller is faster
  - At some point, increase in hit time for a larger cache may overcome the improvement in hit rate, yielding a decrease in performance
- Effect on TIO? Bits in cache? Cost?

Effect of Cache Performance on CPI

- **Recall:** CPU Performance

  CPU Time = Instructions × CPI × Clock Cycle Time
  (IC) (CC)

  - Include memory accesses in CPI:

    $\text{CPI}_{\text{stall}} = \text{CPI}_{\text{base}}$ + Average Memory-stall Cycles

    CPU Time = IC × $\text{CPI}_{\text{stall}}$ × CC

    - Simplified model for memory-stall cycles:

$$\text{Memory-stall cycles} = \frac{\text{Accesses}}{\text{Instruction}} \times \text{MR} \times \text{MP}$$

CPI Example

- **Processor specs:** CPI_{base} of 2, a 100 cycle MP, 36% load/store instructions, and 2% I$ and 4% D$ MRs
  - How many times per instruction do we access the I$? The D$?
  - MP is assumed the same for both I$ and D$
  - Memory-stall cycles will be sum of stall cycles for both I$ and D$

CPI Example

- **Processor specs:** CPI_{base} of 2, a 100 cycle MP, 36% load/store instructions, and 2% I$ and 4% D$ MRs
  - Memory-stall cycles
    $$= (\underbrace{100\% \times 2\%}_{\text{I\$}} + \underbrace{36\% \times 4\%}_{\text{D\$}}) \times 100 = 3.44$$
  - $\text{CPI}_{\text{stall}} = 2 + 3.44 = 5.44$ (more than 2 x CPI_{base}!)

    What if the CPI_{base} is reduced to 1?
- What if the D$ miss rate went up by 1%?

Impacts of Cache Performance

$$CPI_{stall} = CPI_{base} + \text{Memory-stall Cycles}$$

- Relative penalty of cache performance increases as processor performance improves (faster clock rate and/or lower CPI_{base})
  - Relative contribution of CPI_{base} and memory-stall cycles to CPI_{stall}
  - Memory speed unlikely to improve as fast as processor cycle time
- What can we do to improve cache performance?

Sources of Cache Misses: The 3Cs

- Compulsory: (cold start or process migration, 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
- Conflict: (collision)
  - Multiple memory locations mapped to the same cache location

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 cache size
  - Increase associativity (may increase HT)

Summary

- Set associativity determines flexibility of block placement
  - Fully associative: blocks can go anywhere
  - Direct-mapped: blocks go in one specific location
  - N-way: cache split into sets, each of which have *n* slots to place memory blocks
- Cache Performance
  - AMAT = HT + MR × MP
  - CPU time = IC × $\text{CPI}_{\text{stall}}$ × CC
    = IC × (CPI_{base} + Memory-stall cycles) × CC