CPU Caches

Part 2

Berkeley EECS
ELECTRICAL ENGINEERING & COMPUTER SCIENCES

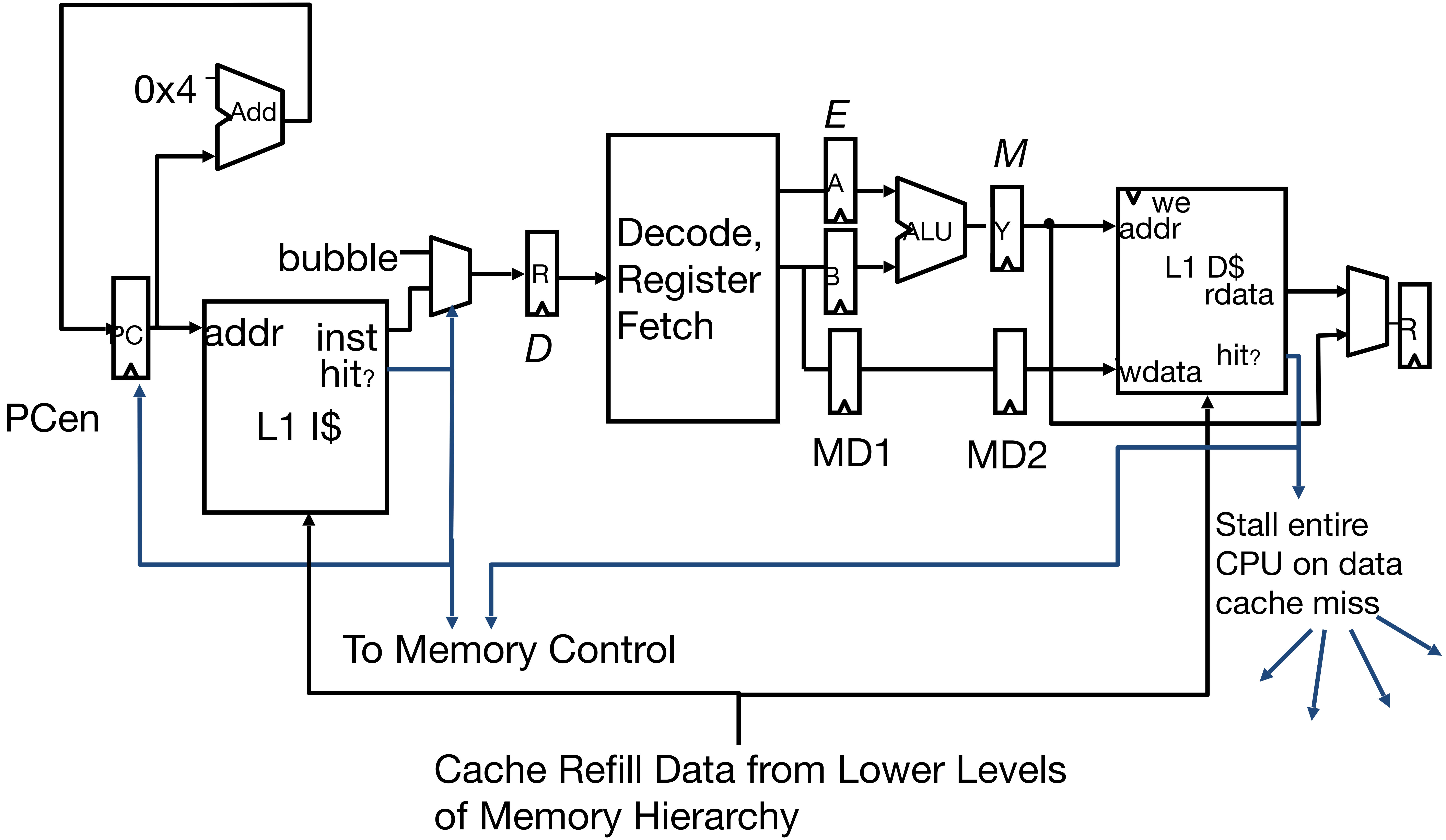
Administrivia

- Project 3A is due Today
- Midterm Exam Next Wednesday (3/17)
  - We have a Piazza post for study resources
  - Dry Run Delayed
- Exam Review Sessions (More Info on Piazza)
  - Course Staff: 3/13 — 3:00-5:30pm Pacific
  - CSM: 3/14 — 11:00am-1:00pm Pacific (Problem-based)
  - HKN: 3/13 — 11:00am-12:30pm Pacific (Conceptual)

Reminder: Cache Terms

- ***Cache***:
  - A small and fast memory used to increase the performance of accessing a big and slow memory
  - Uses ***temporal locality:*** The tendency to reuse data in the same space over time
  - Uses ***spacial locality:*** The tendency to use data at addresses near
- Cache ***hit***: The address being fetched is in the cache
- Cache ***miss:*** The address being fetched is not in the cache
- ***Valid bit***: Is a particular entry valid
- Cache ***flush***: Invalidate all entries

CPU-Cache Interaction (5-stage pipeline)



More Terms

- Cache ***level***:
  - The order in the memory hierarchy: L1$ is closest to the processor
  - L1 caches may only hold data (Data-cache, D$) or instructions (Instruction Cache, I$)
    - Most L2+ caches are "unified", can hold both instructions and data
- Cache ***capacity***:
  - The total # of bytes in the cache
- Cache ***line*** or cache ***block***:
  - A single entry in the cache
- Cache ***block size***:
  - The number of bytes in each cache line

Even More Terms

- ***Number of cache lines***:
  - Cache capacity / block size:
- Cache ***associativity***:
  - The number of possible cache lines a given address may exist in.
  - Also the number of comparison operations needed to check for an element in the cache
  - ***Direct mapped***: A data element can only be in one possible location (N=1)
  - ***N-way set associative***: A data element can be in one of N possible positions
  - ***Fully associative***: A data element can be at any location in the cache.
    - Associativity == # of lines
- Total # of cache lines == capacity of cache/line size
- Total # of lines in a set == # of cache lines / associativity

Parts of the address

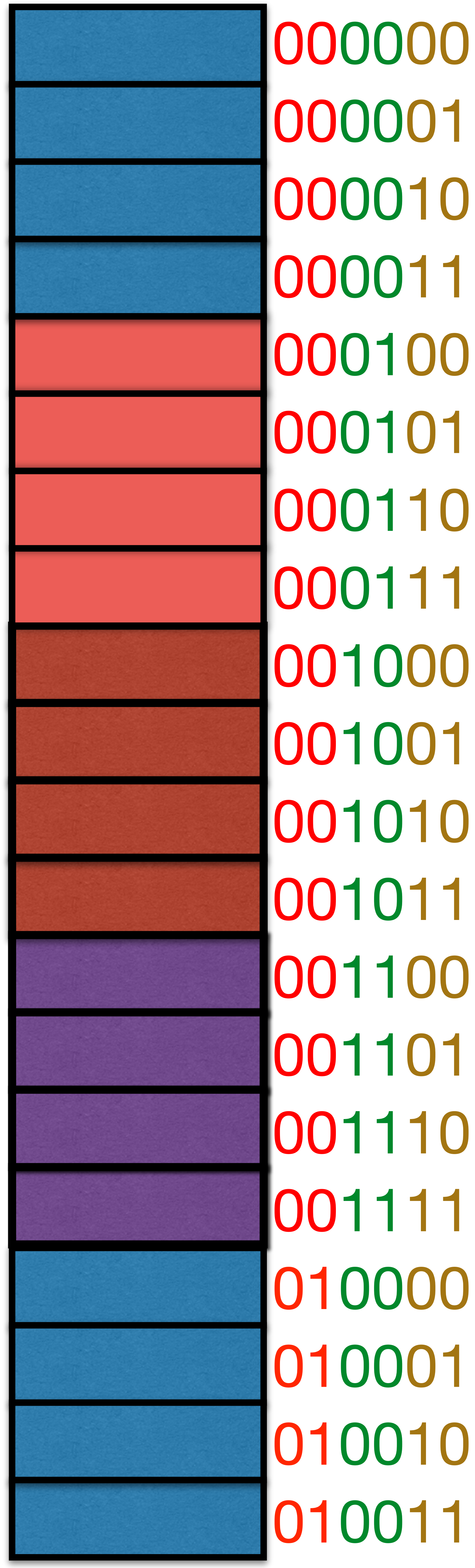
- Address is divided into |***TAG***|***INDEX***|***OFFSET***|
- ***Offset***:
  - The lowest bits of the memory address which say where data exists within the cache line.
  - It is $\log_2$(line/black size)
  - So for a cache with 64B blocks it is 6 bits
- ***Index***:
  - The portion of the address which says where in the cache an address may be stored
  - Takes $\log_2$(# of cache lines / associativity) bits
  - So for a 4 way associative cache with 512 lines it is 7 bits
- ***Tag***: The portion of the address which must be stored in the cache to check if a location matches
  - # of bits of address - (# of bits for index + # of bits for offset)
  - So with 64b addresses it is 51b...

Caching Example

Cache

Index	V	**Tag**	Data
00			
01			
10			
11			

Main Memory



4-byte blocks:
Two low order bits define the **offset** in the block

4 blocks in cache:
Middle two bits define **index** — where in the cache do we place this data?

6-bit addresses:
Remaining top two bits stored as a **tag** — what do we currently have in each block?

Still More Terms

- ***Eviction***:
  - The process of removing an entry from the cache
- ***Write Back***:
  - A cache which only writes data up the hierarchy when a cache line is evicted
  - Instead set a ***dirty bit*** on cache entries
    - All i7 caches are ***write back***
- ***Write Through***:
  - A cache which always writes to memory
- ***Write Allocate***:
  - If writing to memory ***not in the cache*** fetch it first
    - i7 L2 is Write Allocate
- ***No Write Allocate***:
  - Just write to memory without a fetch
    - i7 L1 is no write allocate

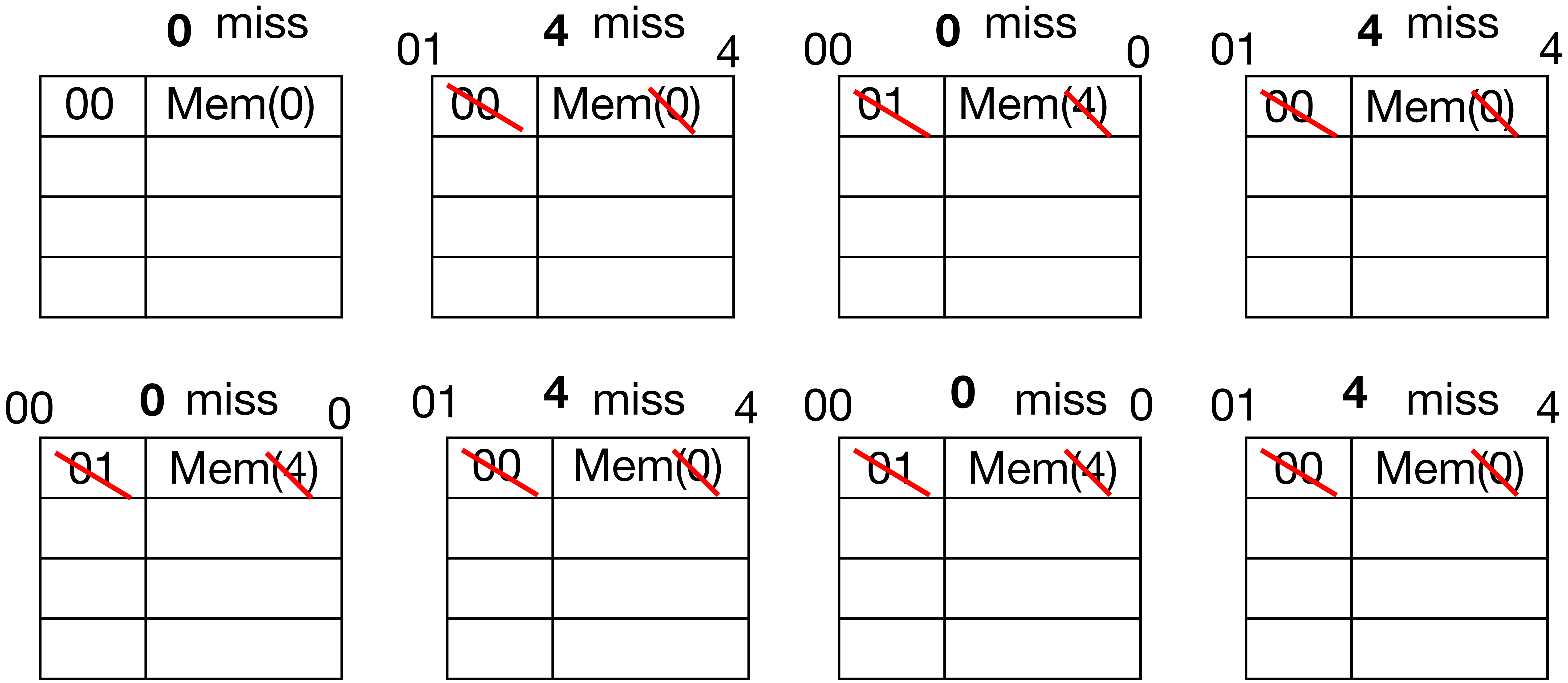
Cache Performance

- ***Hit Time***:
  - Amount of time to return data in a given cache: depends on the cache
  - i7 L1 hit time: 4 clock cycles
- ***Miss Penalty***:
  - Amount of ***additional*** time to return an element if its not in the cache: depends on the cache
- ***Miss Rate***:
  - Fraction of a ***particular program's*** memory requests which miss in the cache
- Average Memory Access Time (***AMAT***):
  - Hit time + Miss Rate * Miss Penalty

Example: Direct-Mapped Cache with 4 Single-Word Blocks, Worst-Case Reference String

- Consider the main memory address reference string of word numbers: 0 4 0 4 0 4 0 4

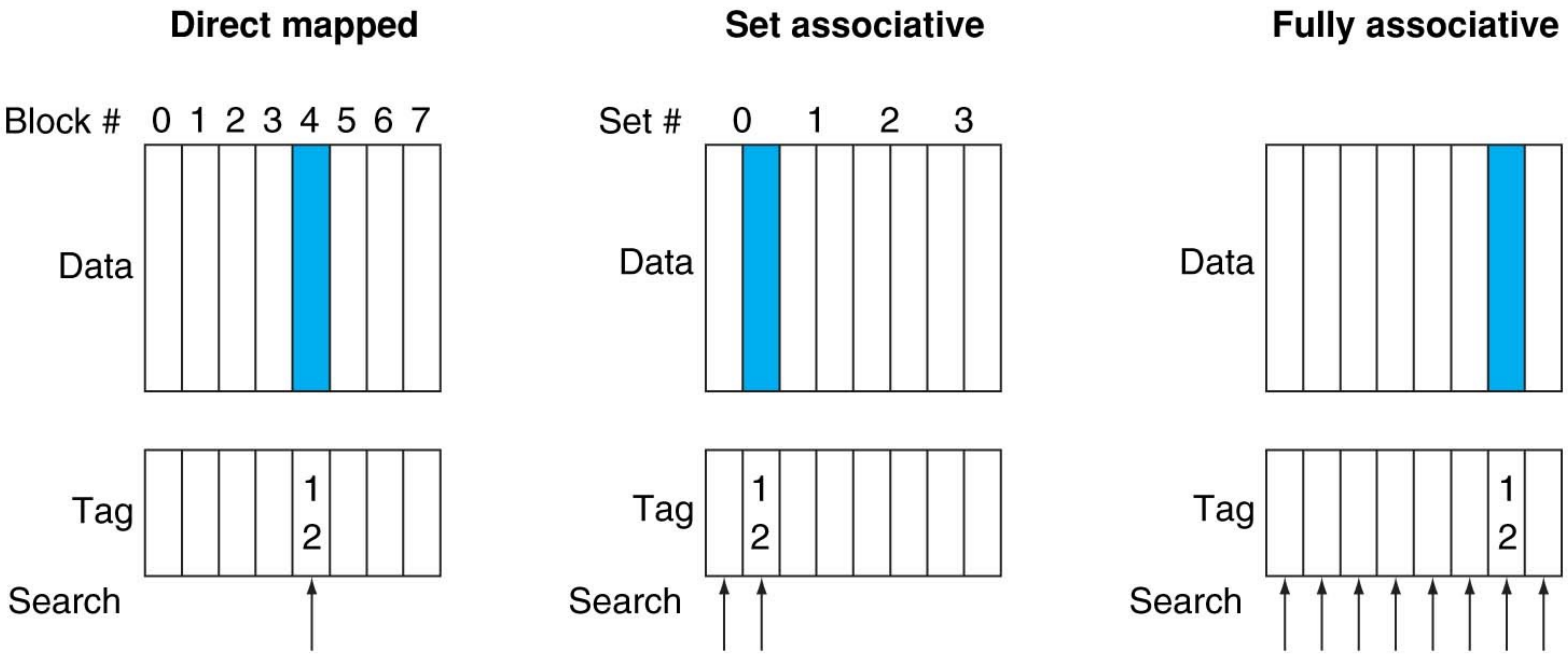
Start with an empty cache - all blocks initially marked as not valid



- 8 requests, 8 misses

Ping-pong effect due to conflict misses - two memory locations that map into the same cache block

Alternative Block Placement Schemes



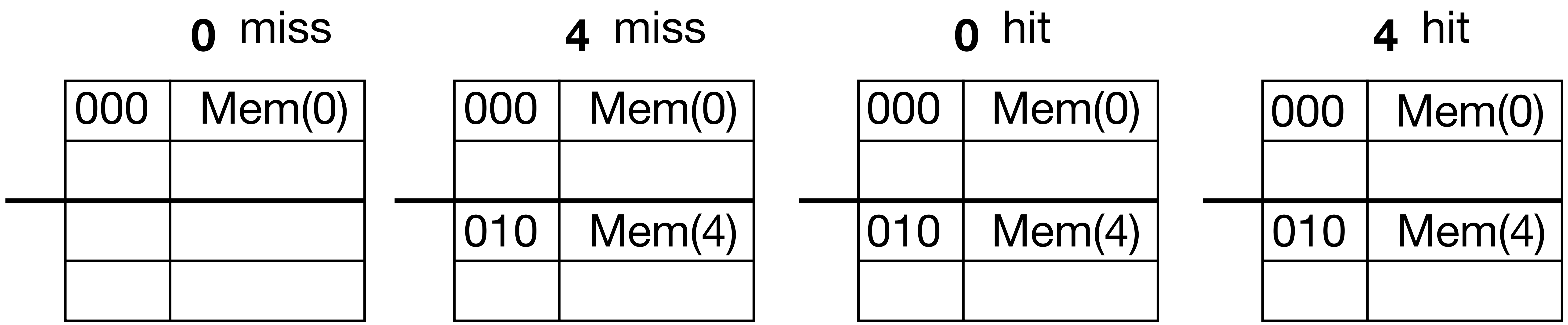
- DM placement: mem block 12 in 8 block cache: only one cache block where mem block 12 can be found—(12 modulo 8) = 4
- SA placement: four sets x 2-ways (8 cache blocks), memory block 12 in set (12 mod 4) = 0; either element of the set
- FA placement: mem block 12 can appear in any cache blocks

Example: 4-Word 2-Way SA $
Same Reference String

- Consider the main memory address reference string

  0 4 0 4 0 4 0 4

  Start with an empty cache - all blocks initially marked as not valid

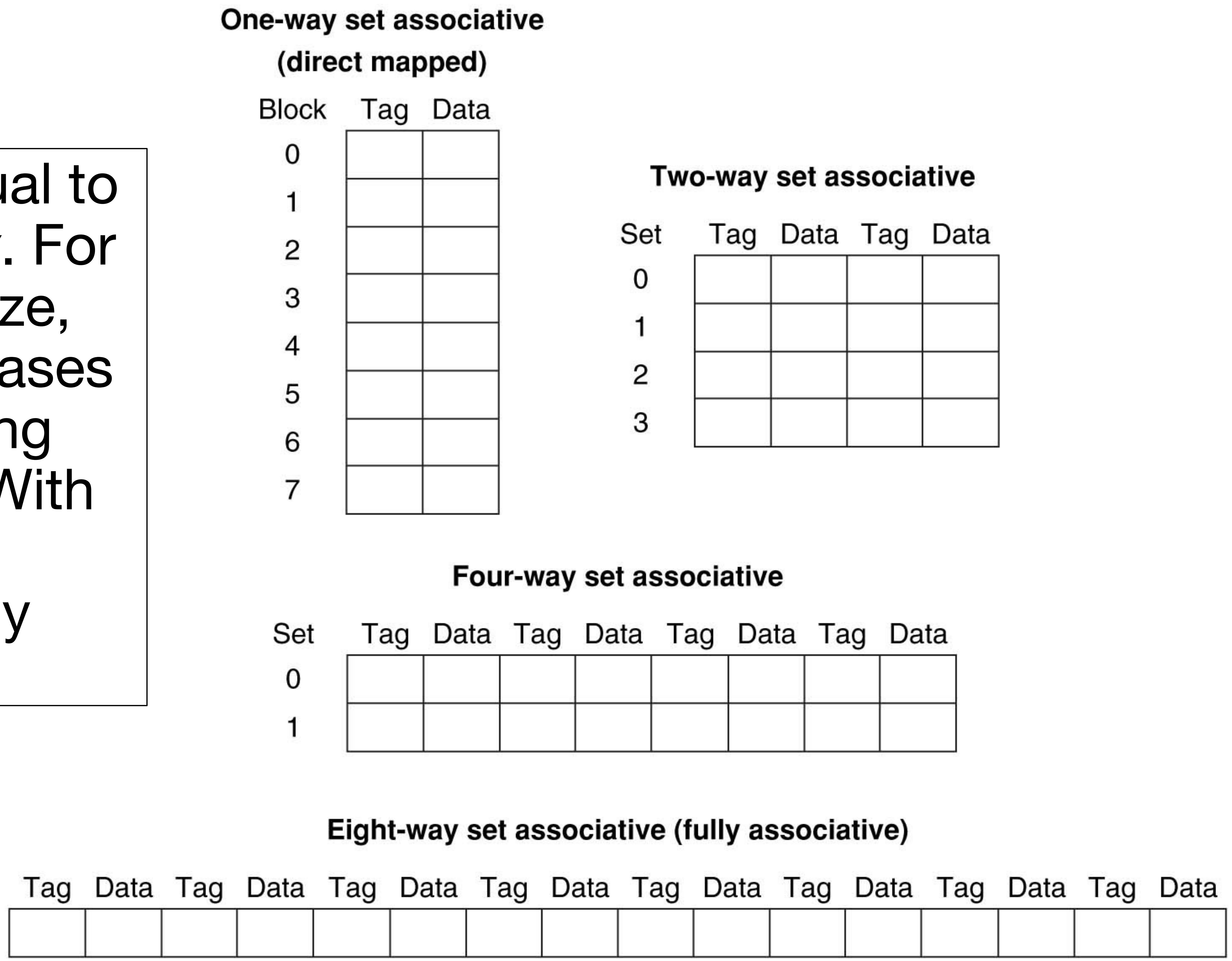


  - 8 requests, 2 misses

- Solves the ping-pong effect in a direct-mapped cache due to conflict misses since now two memory locations that map into the same cache set can co-exist!

Different Organizations of an Eight-Block Cache

Total size of $ in blocks is equal to *number of sets × associativity*. For fixed $ size and fixed block size, increasing associativity decreases number of sets while increasing number of elements per set. With eight blocks, an 8-way set-associative $ is same as a fully associative $.



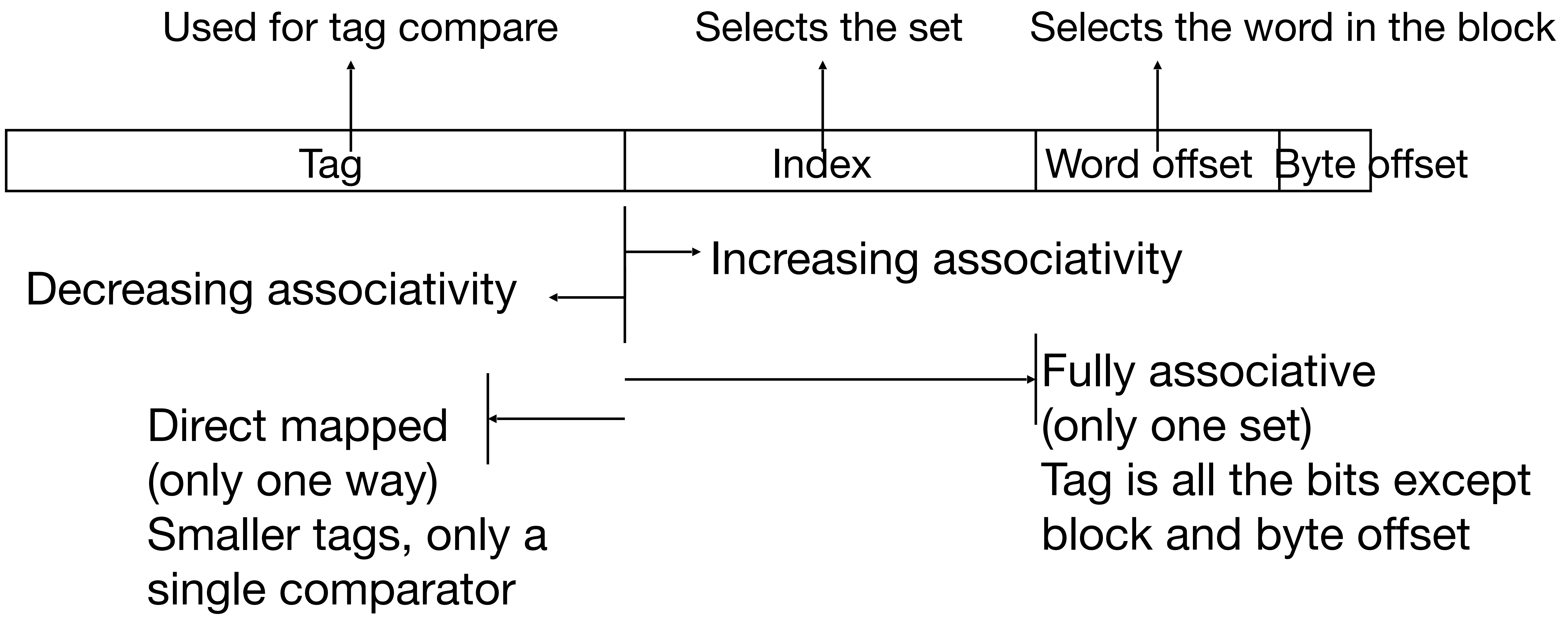
Range of Set-Associative Caches

- For a fixed-size cache and fixed block size, each increase by a factor of two in associativity doubles the number of blocks per set (i.e., the number or ways) and halves the number of sets – decreases the size of the index by 1 bit and increases the size of the tag by 1 bit

Tag	Index	Word offset	Byte offset

Range of Set-Associative Caches

- For a *fixed-size* cache and fixed block size, each increase by a factor of two in associativity doubles the number of blocks per set (i.e., the number or ways) and halves the number of sets – decreases the size of the index by 1 bit and increases the size of the tag by 1 bit



Total Cache Capacity =

Associativity × # of sets × block_size

Bytes = blocks/set × sets × Bytes/block

$$C = N \times S \times B$$

Tag	Index	Byte Offset

address_size = tag_size + index_size + offset_size

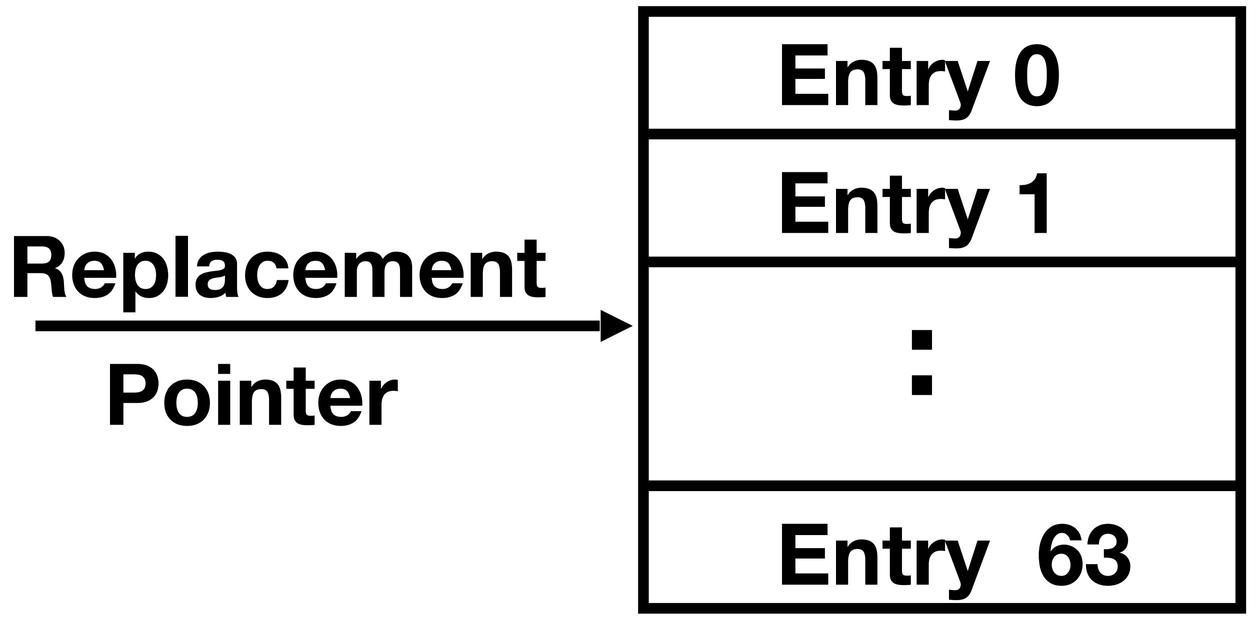
= tag_size + $\log_2(S)$ + $\log_2(B)$

Costs of Set-Associative Caches

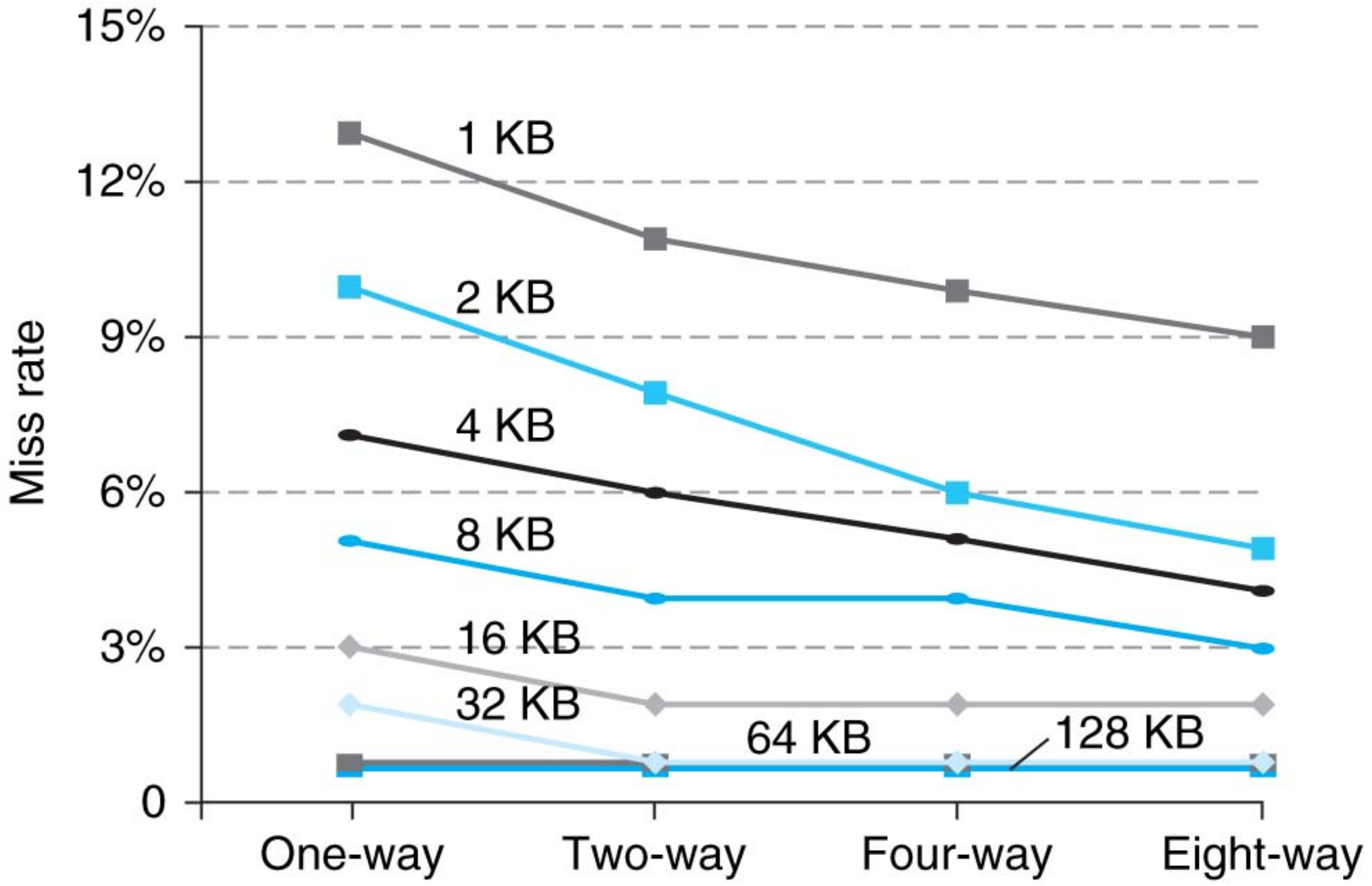
- N-way set-associative cache costs
  - N comparators (delay and area)
  - MUX delay (set selection) before data is available
  - Data available after set selection (and Hit/Miss decision). DM $: block is available before the Hit/Miss decision
    - In Set-Associative, not possible to just assume a hit and continue and recover later if it was a miss
- When miss occurs, which way's block selected for replacement?
  - ***Least Recently Used*** (LRU): one that has been unused the longest (principle of temporal locality)
    - Must track when each way's block was used relative to other blocks in the set
    - For 2-way SA $, one bit per set → set to 1 when a block is referenced; reset the other way's bit (i.e., "last used")

Cache Replacement Policies

- Random Replacement
  - Hardware randomly selects a cache evict
- Least-Recently Used
  - Hardware keeps track of access history
  - Replace the entry that has not been used for the longest time
  - For 2-way set-associative cache, need one bit for LRU replacement
- Example of a Simple “Pseudo” LRU Implementation
  - Assume 64 Fully Associative entries
  - Hardware replacement pointer points to one cache entry
  - Whenever access is made to the entry the pointer points to:
    - Move the pointer to the next entry
  - Otherwise: do not move the pointer
  - (example of “not-most-recently used” replacement policy)



Benefits of Set-Associative Caches



- Largest gains are in going from direct mapped to 2-way (20%+ reduction in miss rate)

Sources of Cache Misses (3 C's)

- *Compulsory* (cold start, first reference):
  - 1st access to a block, 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 of the same size***

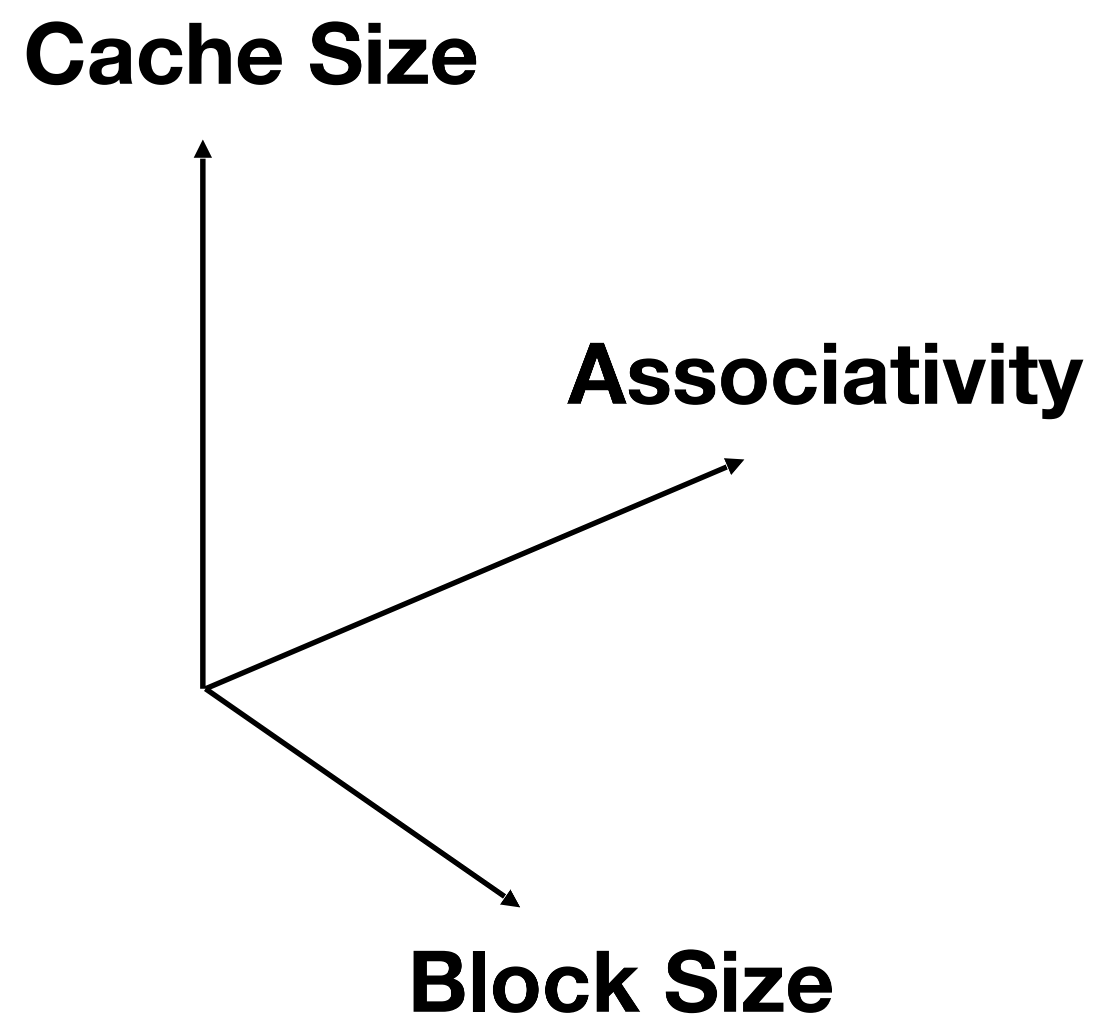
Improving Cache Performance
AMAT = ***Time for a hit*** + ***Miss rate*** x ***Miss penalty***

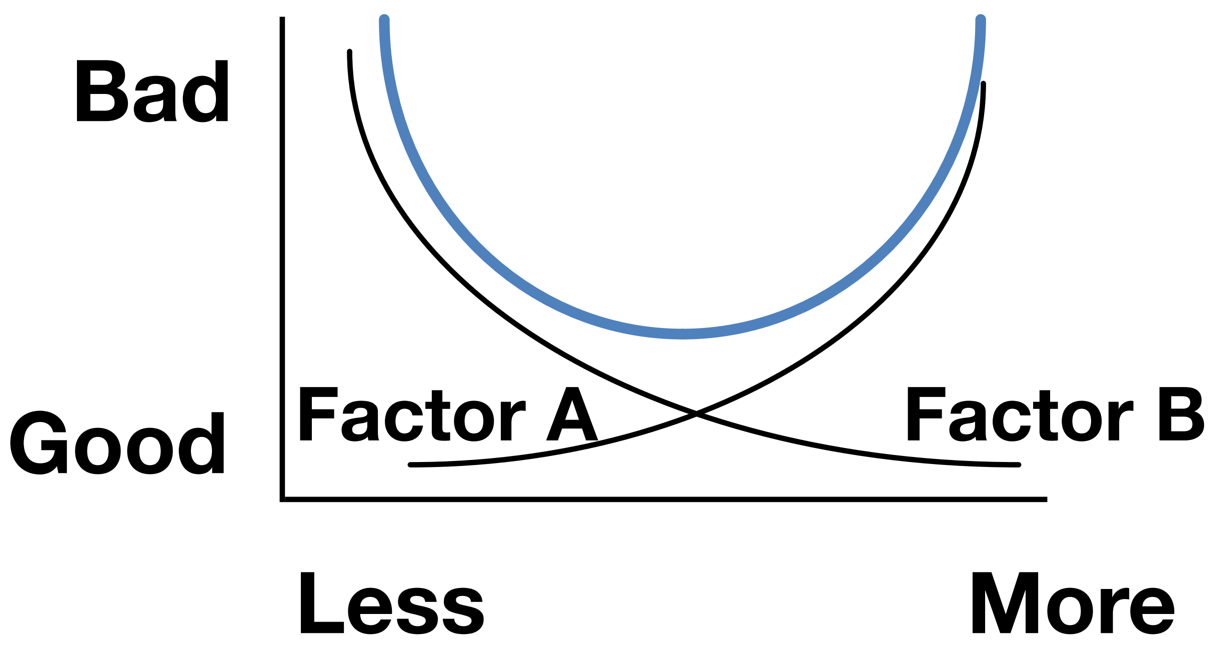
- Note: miss penalty is the ***additional*** time required for a cache miss
- Reduce the time to hit in the cache
  - E.g., Smaller cache
- Reduce the miss rate
  - E.g., Bigger cache
    Longer cache lines (somewhat: improves ability to exploit spacial locality at the cost of reducing the ability to exploit temporal locality)
  - E.g., Better programs!
- Reduce the miss penalty
  - E.g., Use multiple cache levels

Cache Design Space

Computer architects expend considerable effort optimizing organization of cache hierarchy – big impact on performance and power!

- 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)
  - Depends on technology / cost
- Simplicity often wins





Primary Cache Parameters

- Block 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***

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
    - Able to build reasonably efficient wide muxes
- 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?
  - Mostly unchanged, replacement policy runs in parallel with fetching missing line from memory

Increasing #Entries?

- 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
  - Architect's rule of thumb: miss rate drops ~2x for every ~4x increase in capacity (only a gross approximation)
- Miss penalty as #entries increases?
  - Unchanged

At some point, increase in hit time for a larger cache may overcome the improvement in hit rate, yielding a decrease in performance

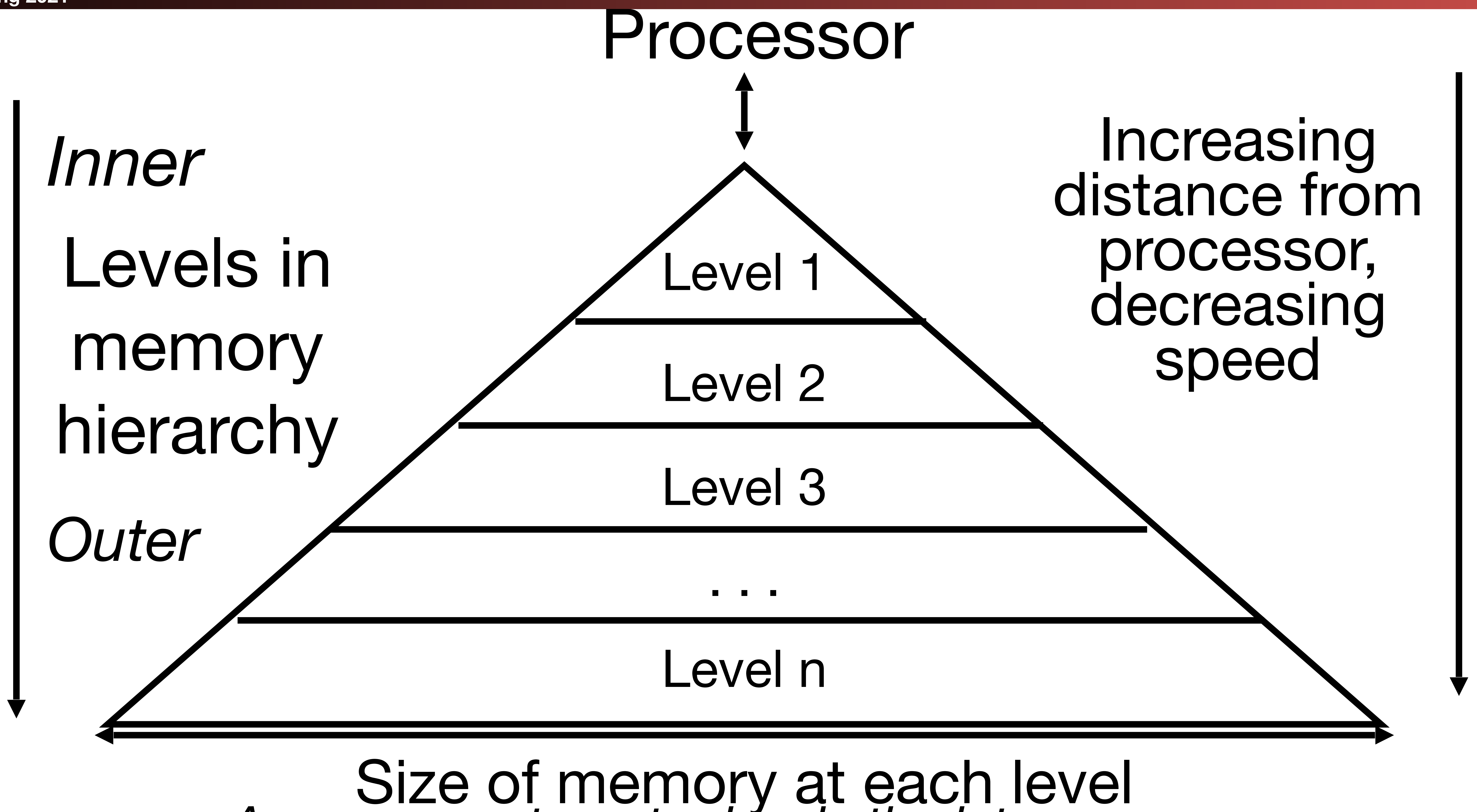
Increasing Block Size?

- 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

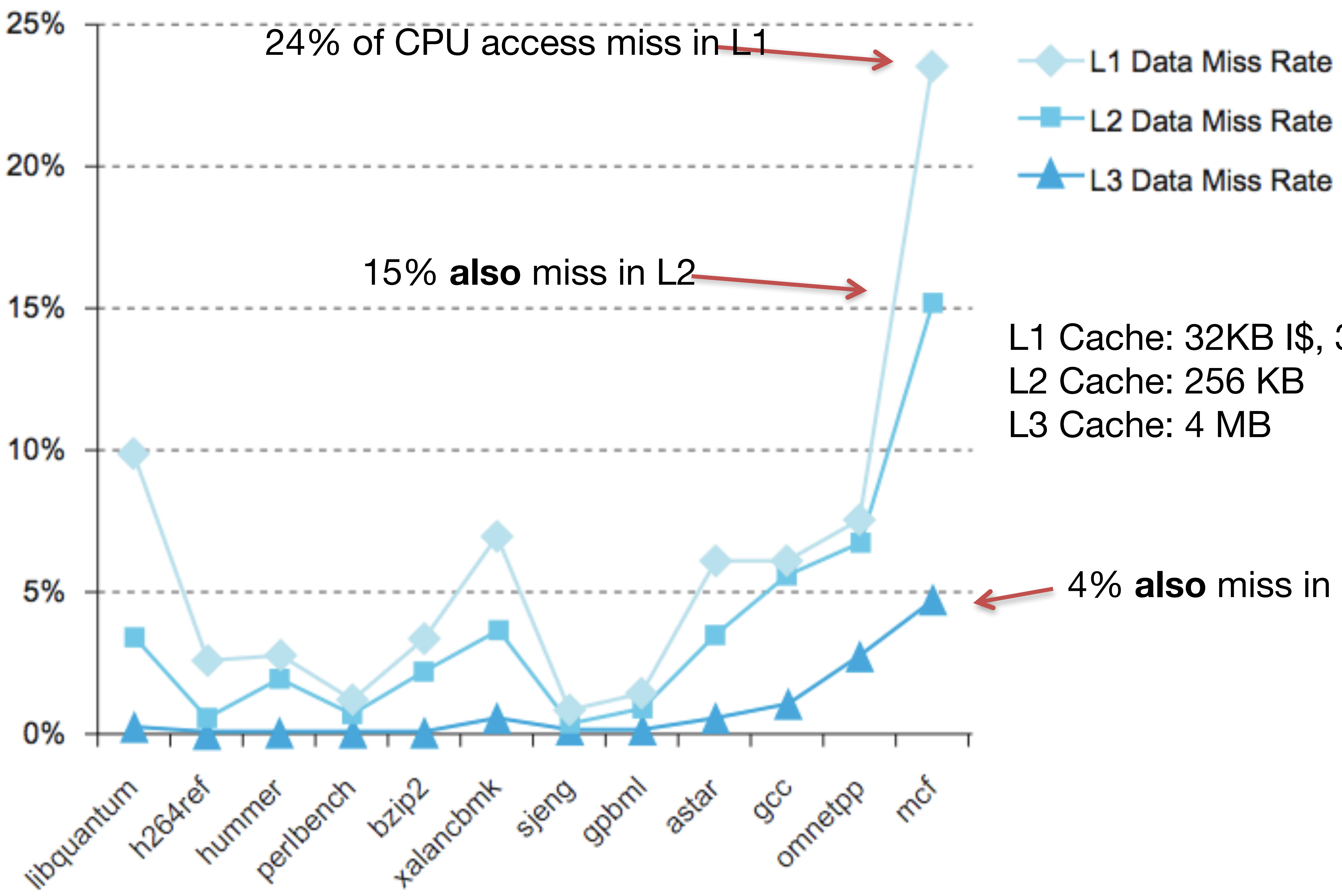
How to Reduce Miss Penalty?

- 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 now big L3 caches!
- 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.



24% of CPU access miss in L1

15% **also** miss in L2

4% **also** miss in L3

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

- ***Global*** miss rate – the fraction of references that miss some level of a multilevel cache
  - misses in this cache divided by the total number of memory accesses generated by the CPU
- ***Local*** miss rate – the fraction of references to one level of a cache that miss
  - Local Miss rate L2$ = L2$ Misses / L1$ Misses
= L2$ Misses / total_L2_accesses
  - L2$ local miss rate >> than the global miss rate

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
  - Global Miss rate = L2$ Misses / Total Accesses
    = (L2$ Misses / L1$ Misses) × (L1$ Misses / Total Accesses)
    = Local Miss rate L2$ × Local Miss rate L1$
- AMAT = Time for a hit + Miss rate × Miss penalty
  - For 2-level cache system:
    AMAT = Time for a L1$ hit + Miss rate L1$ ×
    (Time for a L2$ hit + (local) Miss rate L2$ × L2$ Miss penalty)

Real World Caches: Nehalem and Barcelona

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

A Modern x86's Organization: Intel Xeon E7 v3 (Haswell EX)

- A 18 core processor!
  - And you can get more cores today
- Each core can run two separate threads
  - Two ***separate*** program counters
  - Very pipelined: 14-19 stages (depending on actual instruction)
  - Very superscalar: Issuing up to 7 µOP per cycle between the two threads
  - Very out-of-order: 168 actual registers, 192 instruction window for reordering
- Addressing:
  - 64b addressing, 64B block size for all caches

The Caches

- Each core, 32kB 4-way associative L1 instruction cache, 64B block size
  - 4 cycles latency, pipelined
    - So the ***first 4 stages*** of the instruction pipeline!
  - 512 cache lines
  - Offset: lg(64) = 6b, Index: lg((512/4)) = 7b, Tag: 64-13 = 51b
- Each core, 32 kB L1, 8-way associative write-back data cache
  - 4 cycles latency, pipelined, write back but ***no write allocate***!
  - Pseudo-LRU replacement
- Each core, 256kB 8-way associative write-back L2 cache
  - 10 cycles latency, write back with write allocate
  - Pseudo-LRU
- Common cache, 45MB 16-way associative unified L3 cache (2.5MB per core)
  - Each core has its own section of cache, but all cores can read/write all entries
  - ***Almost***-random replacement

CPI/Miss Rates/DRAM Access SpecInt2006

Name	CPI	L1 D cache misses/1000 instr (Data Only)	L2 D cache misses/1000 instr (Data Only)	DRAM accesses/1000 instr (Instructions and Data)
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

In Conclusion, 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)
  - Depends on technology / cost
- Simplicity often wins