Lecture 15

Set-associative cache
Cache performance

Adapted from Computer Organization and Design, 4th edition, Patterson and Hennessy

The Memory Hierarchy: Terminology

- Block (or line): the minimum unit of information that is present (or not) in a cache
- Hit Rate: the fraction of memory accesses found in a level of the memory hierarchy
  - Hit Time: Time to access that level which consists of

    Time to access the block + Time to determine hit/miss
- Miss Rate: the fraction of memory accesses *not* found in a level of the memory hierarchy ⇒ 1 - (Hit Rate)
  - Miss Penalty: Time to replace a block in that level with the corresponding block from a lower level which consists of

    Time to access the block in the lower level + Time to transmit that block to the level that experienced the miss + Time to insert the block in that level + Time to pass the block to the requestor

Hit Time << Miss Penalty

Handling Cache Misses (Single Word Blocks)

- Read misses (I$ and D$)
  - stall the pipeline, fetch the block from the next level in the memory hierarchy, install it in the cache and send the requested word to the processor, then let the pipeline resume
- Write misses (D$ only)
  1. stall the pipeline, fetch the block from next level in the memory hierarchy, install it in the cache (which may involve having to evict a dirty block if using a write-back cache), write the word from the processor to the cache, then let the pipeline resume

  or

  2. Write allocate – just write the word into the cache updating both the tag and data, no need to check for cache hit, no need to stall

  or

  3. No-write allocate – skip the cache write (but must invalidate that cache block since it will now hold stale data) and just write the word to the write buffer (and eventually to the next memory level), no need to stall if the write buffer isn’t full

Multiword Block Considerations

- Read misses (I$ and D$)
  - Processed the same as for single word blocks – a miss returns the entire block from memory
  - Miss penalty grows as block size grows
    - Early restart – processor resumes execution as soon as the requested word of the block is returned
    - Requested word first – requested word is transferred from the memory to the cache (and processor) first
  - Nonblocking cache – allows the processor to continue to access the cache while the cache is handling an earlier miss
- Write misses (D$)
  - If using write allocate must *first* fetch the block from memory and then write the word to the block (or could end up with a “garbled” block in the cache (e.g., for 4 word blocks, a new tag, one word of data from the new block, and three words of data from the old block)

Handling Cache Hits

- Read hits (I$ and D$)
  - this is what we want!
- Write hits (D$ only)
  - require the cache and memory to be consistent
    - always write the data into both the cache block and the next level in the memory hierarchy (write-through)
    - writes run at the speed of the next level in the memory hierarchy – so slow! – or can use a write buffer and stall only if the write buffer is full
  - allow cache and memory to be inconsistent
    - write the data only into the cache block (write-back the cache block to the next level in the memory hierarchy when that cache block is “evicted”)
    - need a dirty bit for each data cache block to tell if it needs to be written back to memory when it is evicted – can use a write buffer to help “buffer” write-backs of dirty blocks

Reducing Cache Miss Rates #1

1. Allow more flexible block placement

- In a direct mapped cache a memory block maps to exactly one cache block
- At the other extreme, could allow a memory block to be mapped to *any* cache block – fully associative cache

- A compromise is to divide the cache into sets each of which consists of n "ways" (n-way set associative). A memory block maps to a unique set (specified by the index field) and can be placed in any way of that set (so there are n choices)

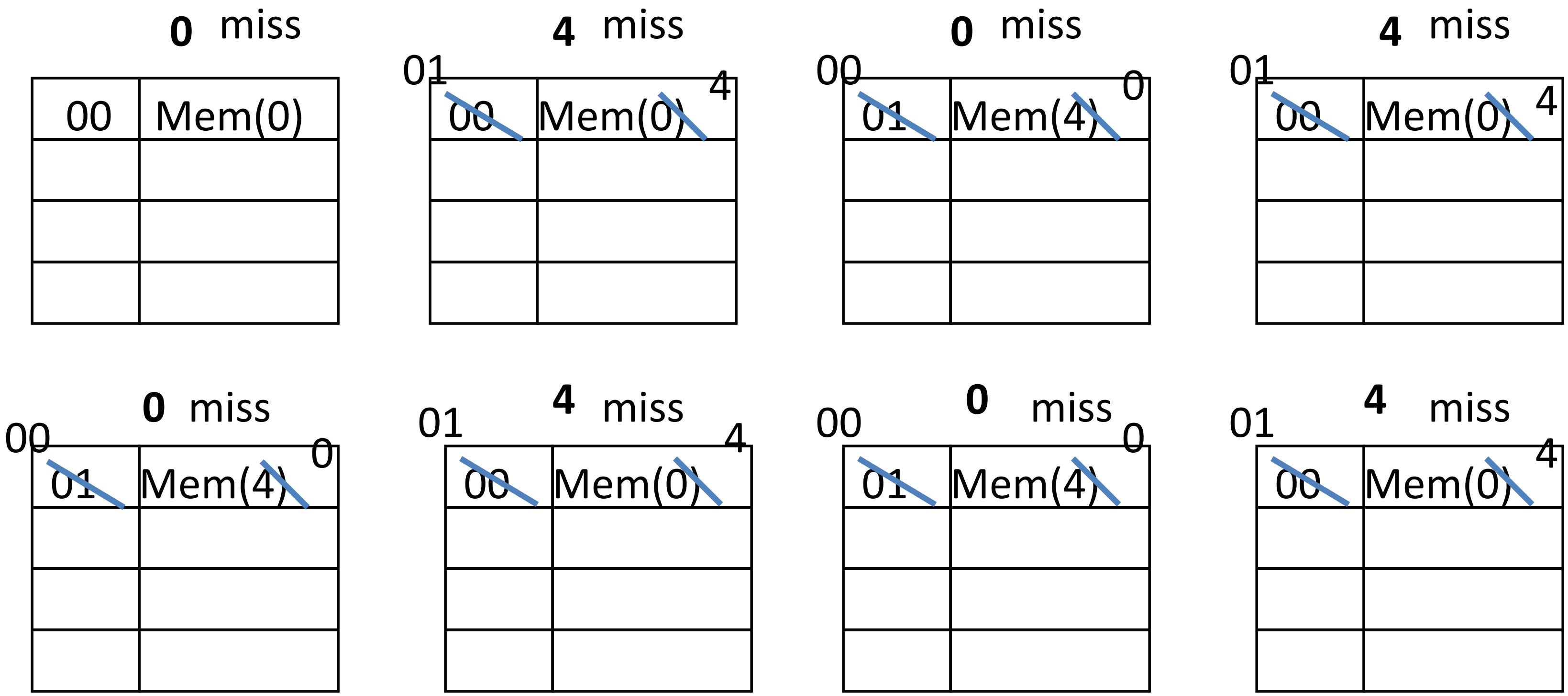
(block address) modulo (# sets in the cache)

Another Reference String Mapping

- Consider the main memory word reference string

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

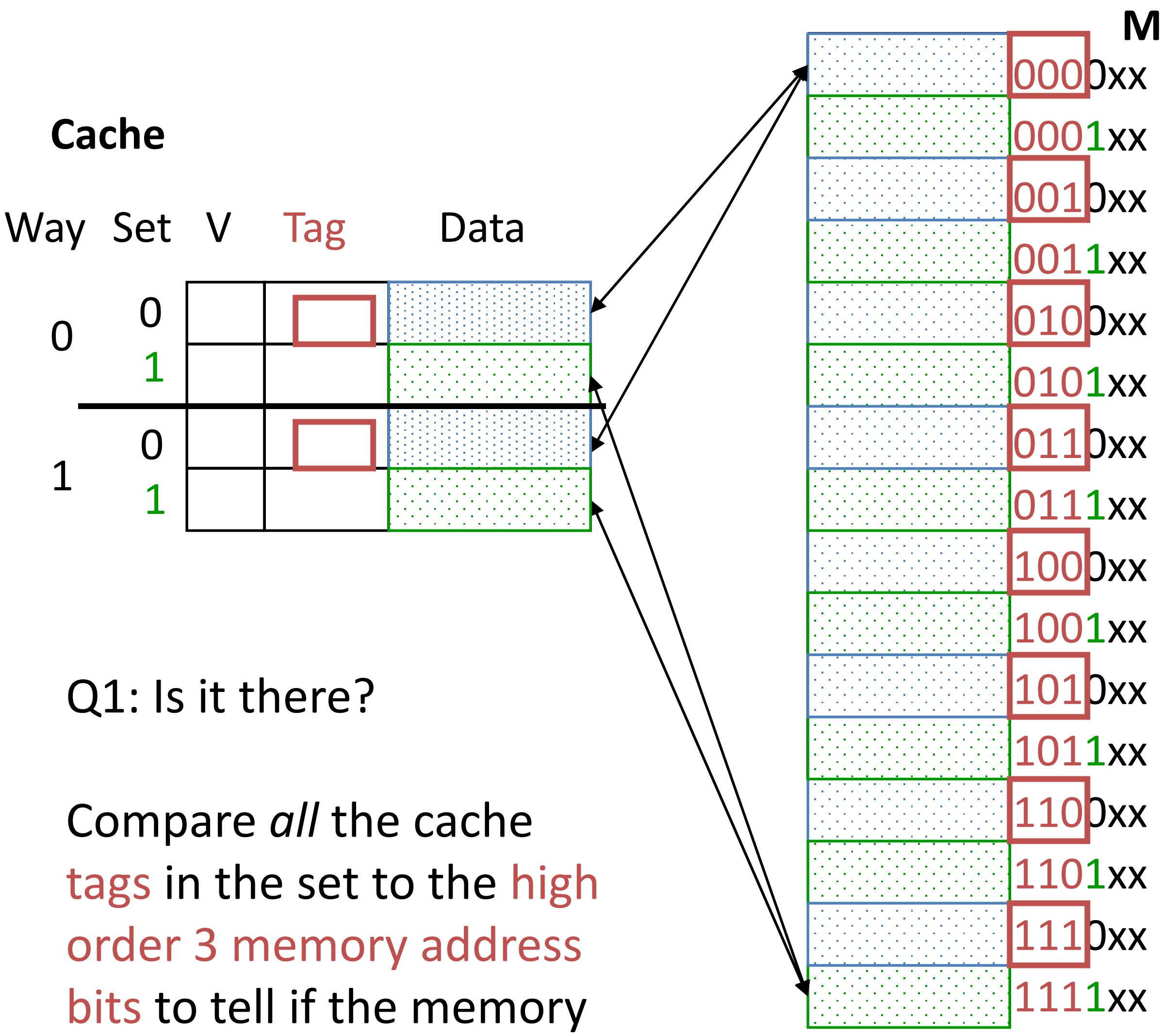
0 4 0 4 0 4 0 4



- 8 requests, 8 misses

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

Set Associative Cache Example



Cache

Way Set V Tag Data

0 0
1

1 0
1

Main Memory

0000xx
0001xx
0010xx
0011xx
0100xx
0101xx
0110xx
0111xx
1000xx
1001xx
1010xx
1011xx
1100xx
1101xx
1110xx
1111xx

One word blocks
Two low order bits
define the byte in the
word (32b words)

Q1: Is it there?

Compare *all* the cache tags in the set to the high order 3 memory address bits to tell if the memory block is in the cache

Q2: How do we find it?

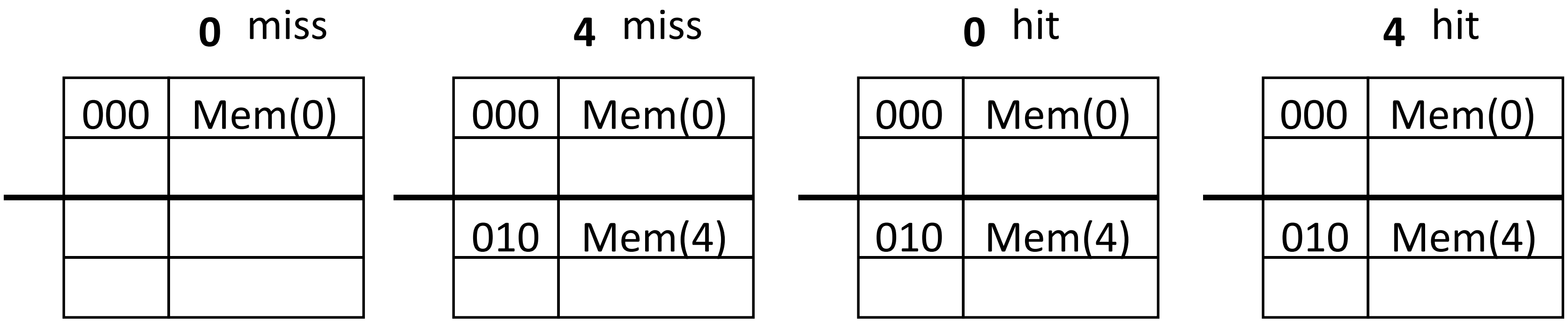
Use next 1 low order memory address bit to determine which cache set (i.e., modulo the number of sets in the cache)

Another Reference String Mapping

- Consider the main memory word reference string

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

0 4 0 4 0 4 0 4



0 miss

000	Mem(0)

4 miss

000	Mem(0)
010	Mem(4)

0 hit

000	Mem(0)
010	Mem(4)

4 hit

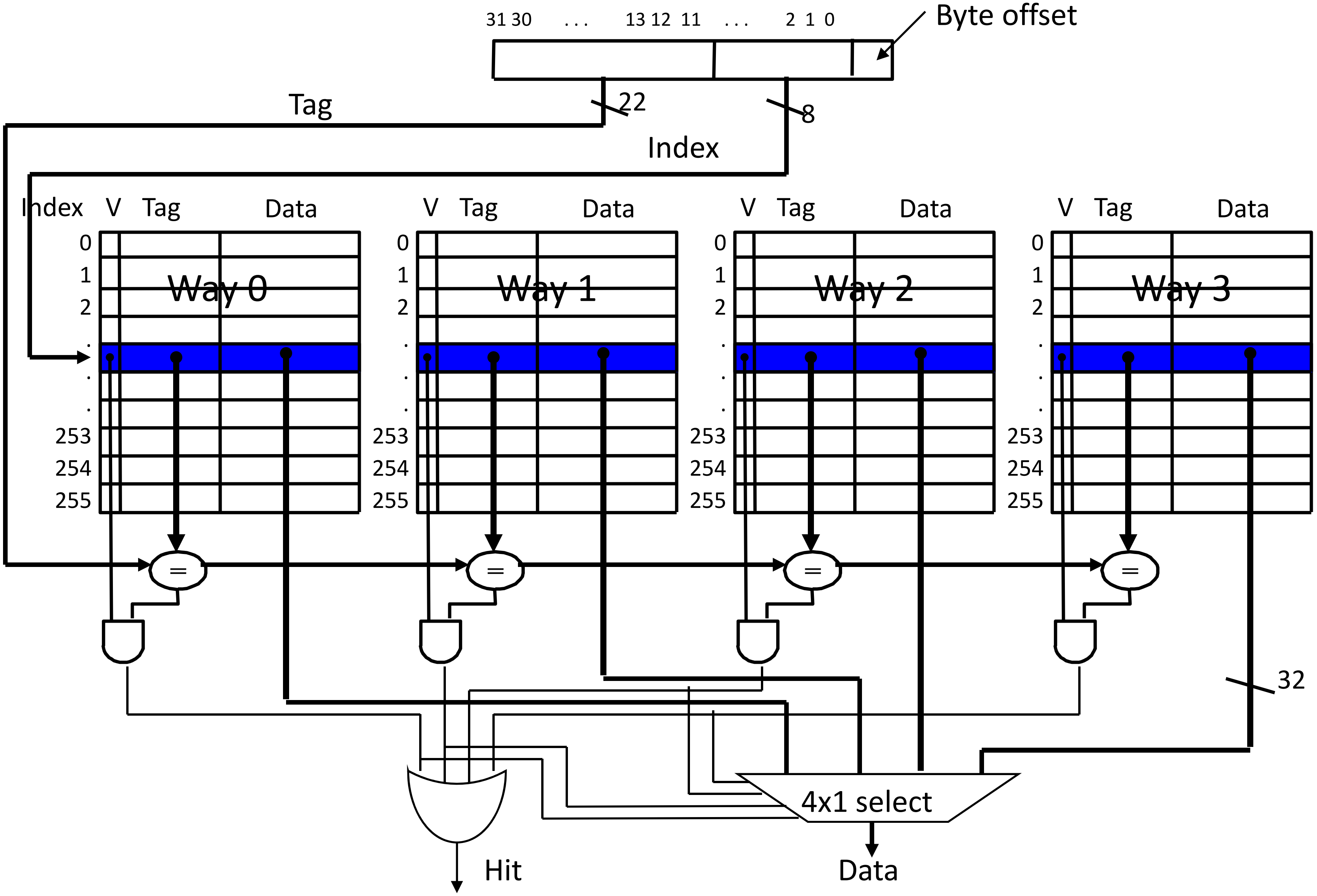
000	Mem(0)
010	Mem(4)

- 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!

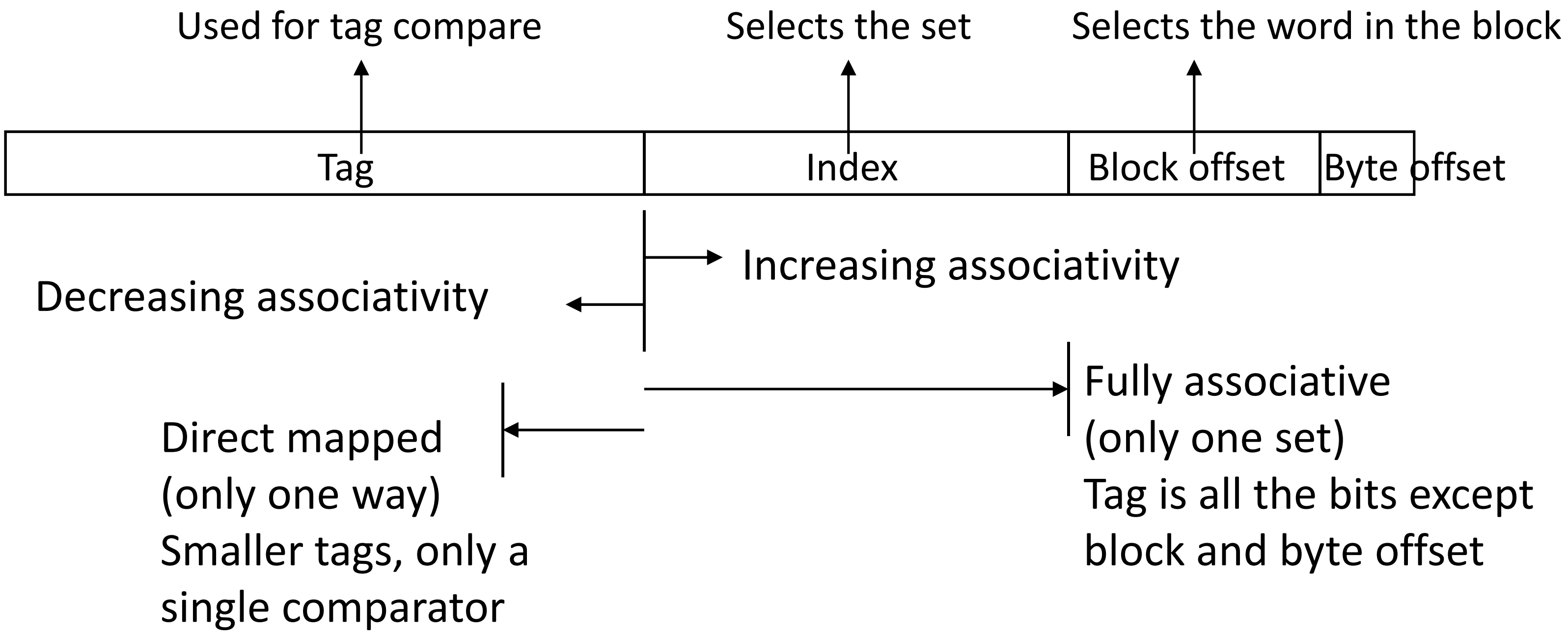
Four-Way Set Associative Cache

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



Range of Set Associative Caches

- 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 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



Used for tag compare

Selects the set

Selects the word in the block

Tag

Index

Block offset

Byte offset

Decreasing associativity

Increasing associativity

Direct mapped
(only one way)
Smaller tags, only a single comparator

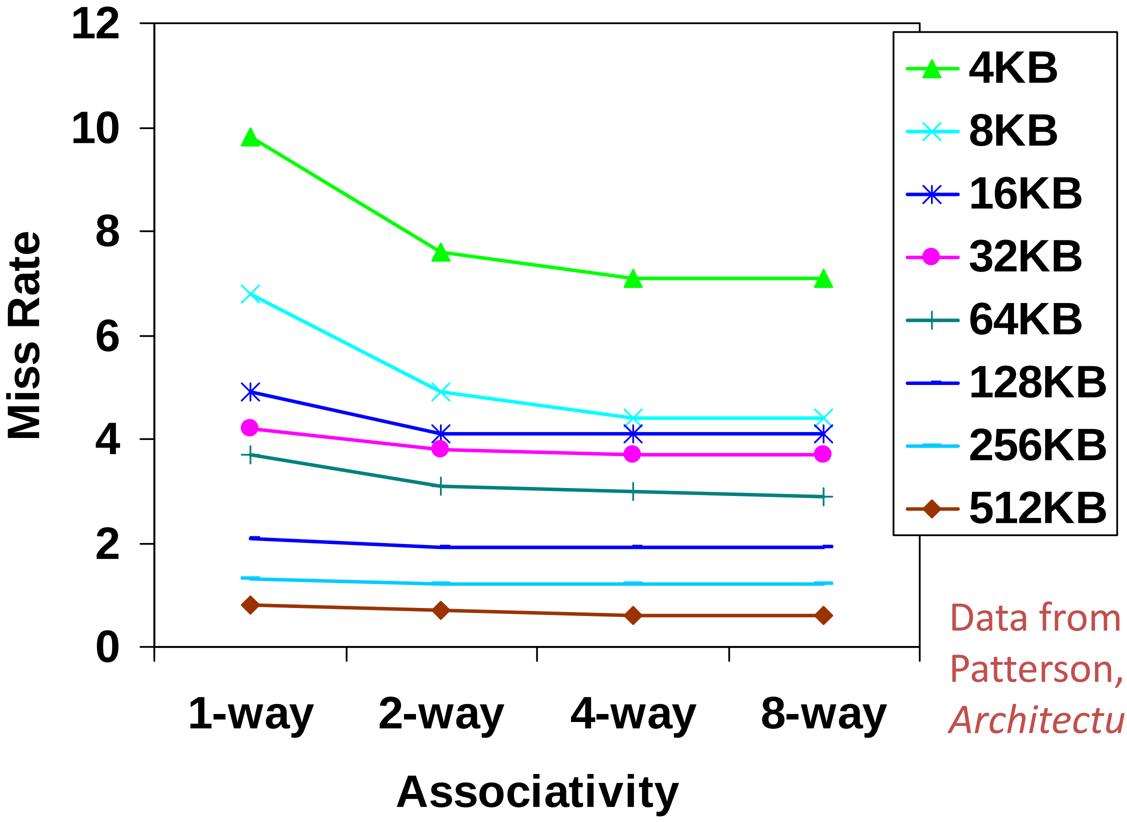
Fully associative
(only one set)
Tag is all the bits except block and byte offset

Costs of Set Associative Caches

- When a miss occurs, which way's block do we pick for replacement?
  - Least Recently Used (LRU): the block replaced is the one that has been unused for the longest time
    - Must have hardware to keep track of when each way's block was used relative to the other blocks in the set
    - For 2-way set associative, takes one bit per set → set the bit when a block is referenced (and reset the other way's bit)
- 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). In a direct mapped cache, the cache block is available before the Hit/Miss decision
    - So its not possible to just assume a hit and continue and recover later if it was a miss

Benefits of Set Associative Caches

- The choice of direct mapped or set associative depends on the cost of a miss versus the cost of implementation



Data from Hennessy & Patterson, *Computer Architecture*, 2003

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

Sources of Cache Misses

- Compulsory (cold start or process migration, first reference):
  - First access to a block, “cold” fact of life, not a whole lot you can do about it. If you are going to run “millions” of instruction, compulsory misses are insignificant
  - Solution: increase block size (increases miss penalty; very large blocks could increase miss rate)
- Capacity:
  - Cache cannot contain all blocks accessed by the program
  - Solution: increase cache size (may increase access time)
- Conflict (collision):
  - Multiple memory locations mapped to the same cache location
  - Solution 1: increase cache size
  - Solution 2: increase associativity (stay tuned) (may increase access time)

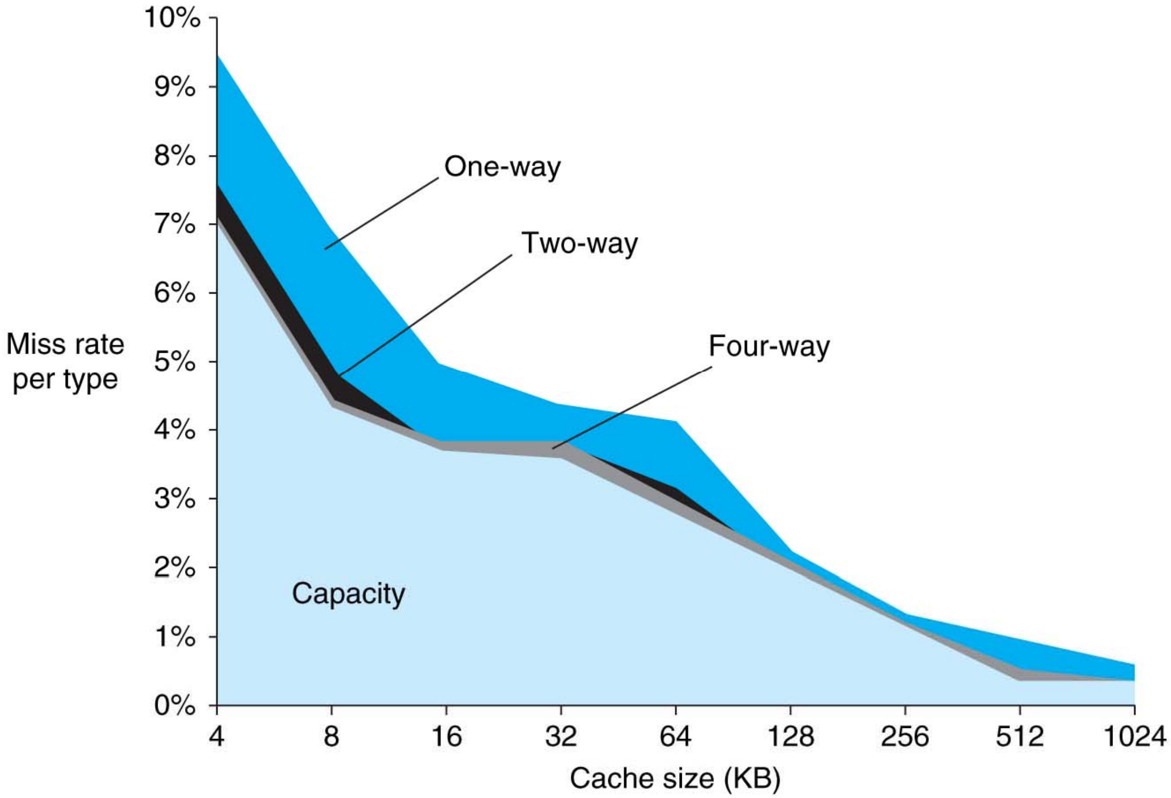


FIGURE 5.31 The miss rate can be broken into three sources of misses. This graph shows the total miss rate and its components for a range of cache sizes. This data is for the SPEC2000 integer and floating-point benchmarks and is from the same source as the data in Figure 5.30. The compulsory miss component is 0.006% and cannot be seen in this graph. The next component is the capacity miss rate, which depends on cache size. The conflict portion, which depends both on associativity and on cache size, is shown for a range of associativities from one-way to eight-way. In each case, the labeled section corresponds to the increase in the miss rate that occurs when the associativity is changed from the next higher degree to the labeled degree of associativity. For example, the section labeled *two-way* indicates the additional misses arising when the cache has associativity of two rather than four. Thus, the difference in the miss rate incurred by a direct-mapped cache versus a fully associative cache of the same size is given by the sum of the sections

marked *eight-way, four-way, two-way,* and *one-way*. The difference between eight-way and four-way is so small that it is difficult to see on this graph. Copyright © 2009 Elsevier, Inc. All rights reserved.

Measuring Cache Performance

- Assuming cache hit costs are included as part of the normal CPU execution cycle, then

$$\text{CPU time} = \text{IC} \times \text{CPI} \times \text{CC}$$

$$= \text{IC} \times \underbrace{(\text{CPI}_{\text{ideal}} + \text{Memory-stall cycles})}_{\text{CPI}_{\text{stall}}} \times \text{CC}$$

- ❑ Memory-stall cycles come from cache misses (a sum of read-stalls and write-stalls)

$$\text{Read-stall cycles} = \text{reads/program} \times \text{read miss rate} \times \text{read miss penalty}$$

$$\text{Write-stall cycles} = (\text{writes/program} \times \text{write miss rate} \times \text{write miss penalty}) + \text{write buffer stalls}$$

- ❑ For write-through caches, we can simplify this to

$$\text{Memory-stall cycles} = \text{accesses/program} \times \text{miss rate} \times \text{miss penalty}$$

Impacts of Cache Performance

- Relative cache penalty increases as processor performance improves (faster clock rate and/or lower CPI)
  - The memory speed is unlikely to improve as fast as processor cycle time. When calculating $\text{CPI}_{\text{stall}}$, the cache miss penalty is measured in *processor* clock cycles needed to handle a miss
  - The lower the $\text{CPI}_{\text{ideal}}$, the more pronounced the impact of stalls
- A processor with a $\text{CPI}_{\text{ideal}}$ of 2, a 100 cycle miss penalty, 36% load/store instr's, and 2% I$ and 4% D$ miss rates

  Memory-stall cycles = 2% × 100 + 36% × 4% × 100 = 3.44

  So $\text{CPI}_{\text{stalls}}$ = 2 + 3.44 = **5.44**

  more than twice the $\text{CPI}_{\text{ideal}}$!
- What if the $\text{CPI}_{\text{ideal}}$ is reduced to 1? 0.5? 0.25?
- What if the D$ miss rate went up 1%? 2%?
- What if the processor clock rate is doubled (doubling the miss penalty)?

Average Memory Access Time (AMAT)

- A larger cache will have a longer access time. An increase in hit time will likely add another stage to the pipeline. At some point the increase in hit time for a larger cache will overcome the improvement in hit rate leading to a decrease in performance.
- Average Memory Access Time (AMAT) is the average to access memory considering both hits and misses

AMAT = Time for a hit + Miss rate x Miss penalty

- What is the AMAT for a processor with a 20 psec clock, a miss penalty of 50 clock cycles, a miss rate of 0.02 misses per instruction and a cache access time of 1 clock cycle?

Reducing Cache Miss Rates #2

2. Use multiple levels of caches

- With advancing technology have more than enough room on the die for bigger L1 caches *or* for a second level of caches – normally a unified L2 cache (i.e., it holds both instructions and data) and in some cases even a unified L3 cache
- For our example, $\text{CPI}_{\text{ideal}}$ of 2, 100 cycle miss penalty (to main memory) and a 25 cycle miss penalty (to UL2$), 36% load/stores, a 2% (4%) L1 I$ (D$) miss rate, add a 0.5% UL2$ miss rate

$$\text{CPI}_{\text{stalls}} = 2 + .02\times25 + .36\times.04\times25 + .005\times100 + .36\times.005\times100 = 3.54$$

(as compared to 5.44 with no L2$)

Multilevel Cache Design Considerations

- Design considerations for L1 and L2 caches are very different
  - Primary cache should focus on minimizing hit time in support of a shorter clock cycle
    - Smaller with smaller block sizes
  - Secondary cache(s) should focus on reducing miss rate to reduce the penalty of long main memory access times
    - Larger with larger block sizes
    - Higher levels of associativity
- The miss penalty of the L1 cache is significantly reduced by the presence of an L2 cache – so it can be smaller (i.e., faster) but have a higher miss rate
- For the L2 cache, hit time is less important than miss rate
  - The L2$ hit time determines L1$'s miss penalty
  - L2$ local miss rate >> than the global miss rate