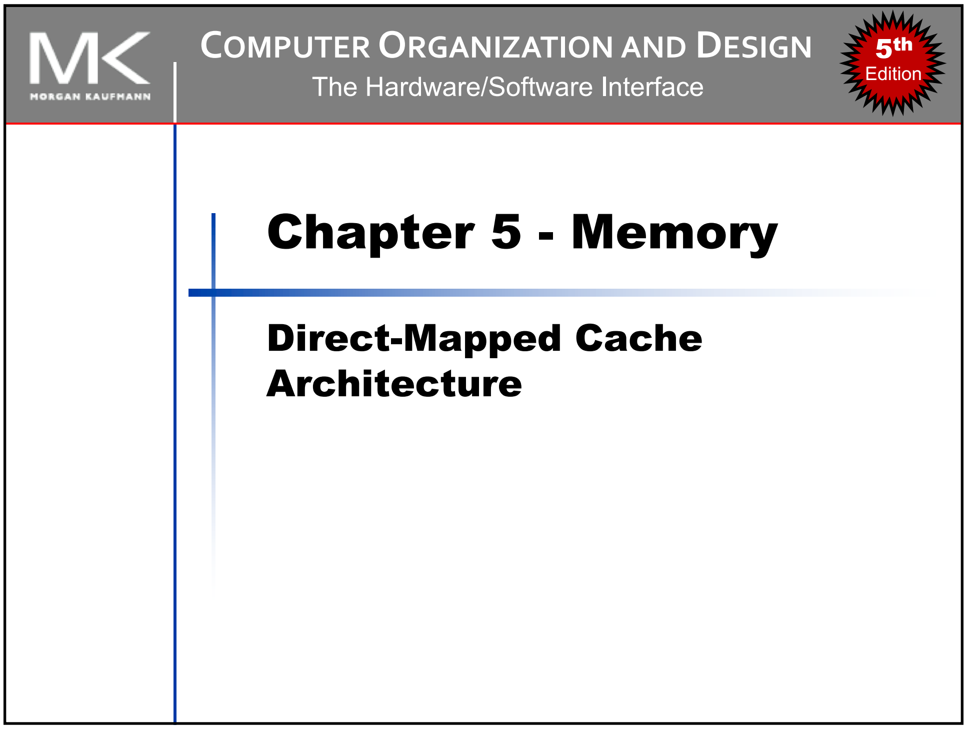
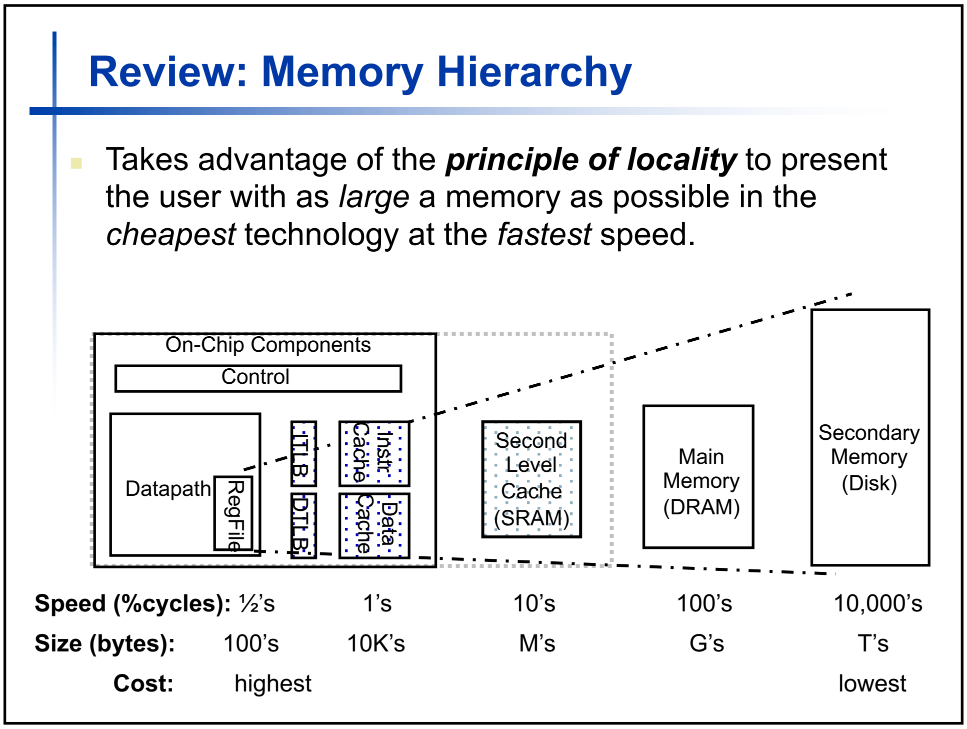


COMPUTER ORGANIZATION AND DESIGN
The Hardware/Software Interface
5th Edition

Chapter 5 - Memory

Direct-Mapped Cache Architecture



Review: Memory Hierarchy

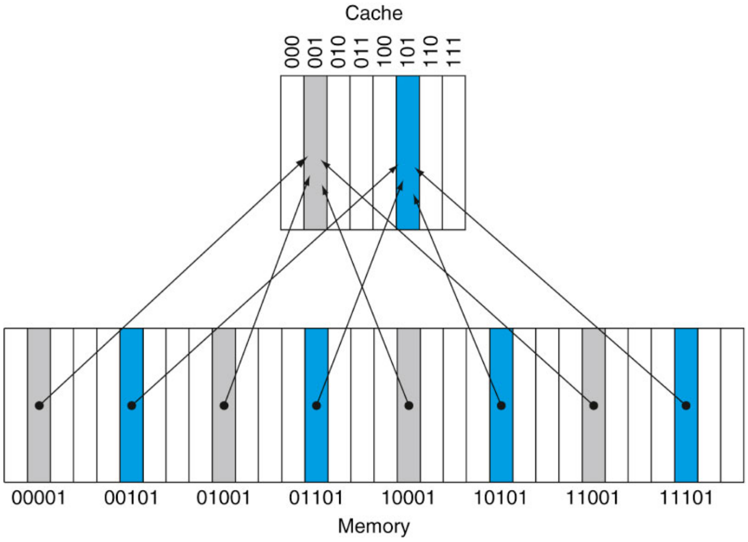
- Takes advantage of the ***principle of locality*** to present the user with as *large* a memory as possible in the *cheapest* technology at the *fastest* speed.

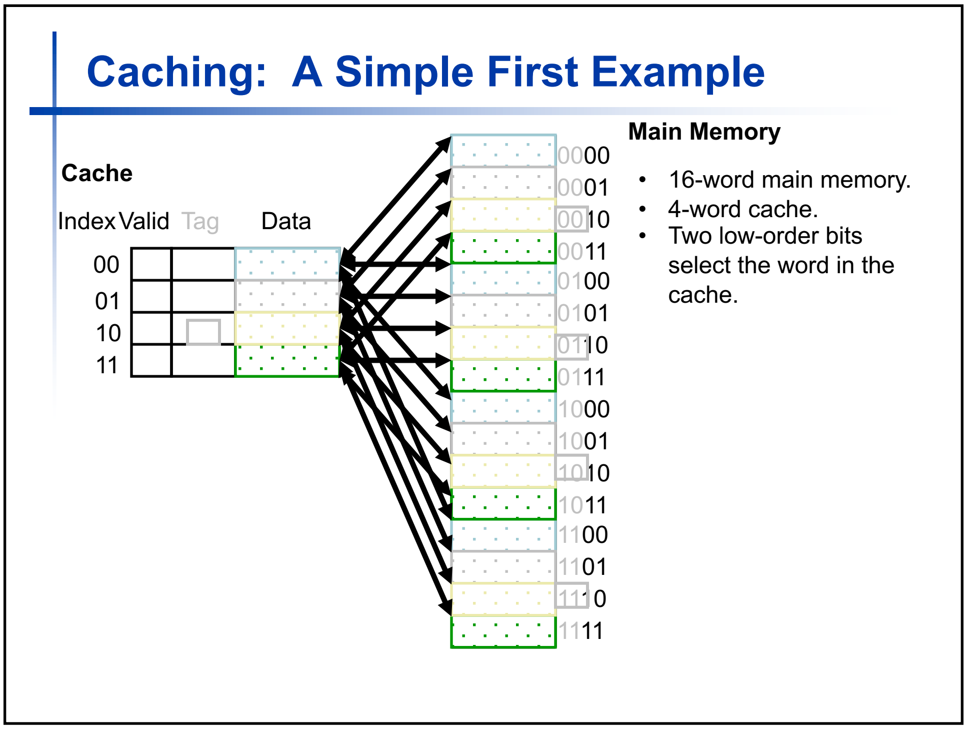
Speed (%cycles):	½'s	1's	10's	100's	10,000's
Size (bytes):	100's	10K's	M's	G's	T's
Cost:	highest				lowest

Cache Basics Review

- Two questions to answer in hardware:
  - Q1: How do we know if a data item is in the cache?
  - Q2: If it is in the cache, how do we find it?
- Direct mapped method
  - Each memory block is mapped to exactly one block in the cache:
    - Lots of lower level blocks must ***share*** blocks in the cache.
  - How do we know which particular block is stored in a cache location?
    - Store a portion of the block address as well as the data.
    - The higher-order address bits stored with the data are called the ***tag.***
  - What if there is no data in a location?
    - Valid bit: 1 = present, 0 = not present
    - Initially set to 0.

Direct-Mapped Cache With 8 Entries

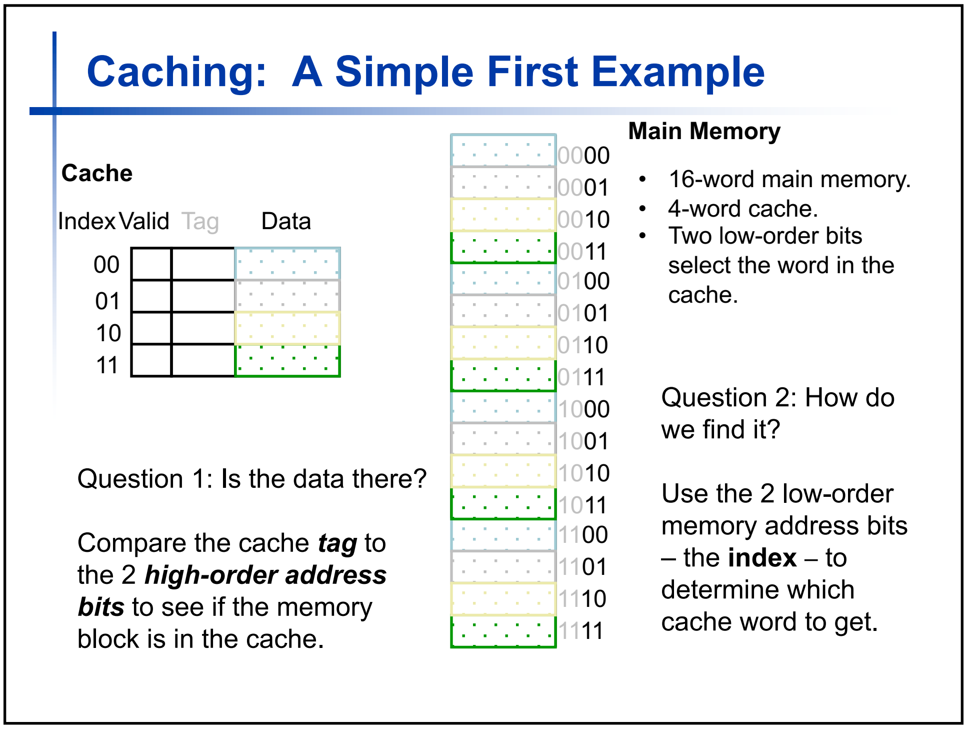




Caching: A Simple First Example

Main Memory

- 16-word main memory.
- 4-word cache.
- Two low-order bits select the word in the cache.



Caching: A Simple First Example

Main Memory

- 16-word main memory.
- 4-word cache.
- Two low-order bits select the word in the cache.

Question 1: Is the data there?

Compare the cache ***tag*** to the 2 ***high-order address bits*** to see if the memory block is in the cache.

Question 2: How do we find it?

Use the 2 low-order memory address bits – the **index** – to determine which cache word to get.

Direct Mapped Cache Example

- Start with an empty cache - all blocks initially **not valid**.
- Consider the following string of address requests:

0000 0001 0010 0011 0100 0011 0100 1111

0 miss

00	Mem(0)

1 miss

00	Mem(0)
00	Mem(1)

2 miss

00	Mem(0)
00	Mem(1)
00	Mem(2)

3 miss

00	Mem(0)
00	Mem(1)
00	Mem(2)
00	Mem(3)

4 miss

01 ~~00~~	Mem(~~0~~) 4
00	Mem(1)
00	Mem(2)
00	Mem(3)

3 hit

01	Mem(4)
00	Mem(1)
00	Mem(2)
00	Mem(3)

4 hit

01	Mem(4)
00	Mem(1)
00	Mem(2)
00	Mem(3)

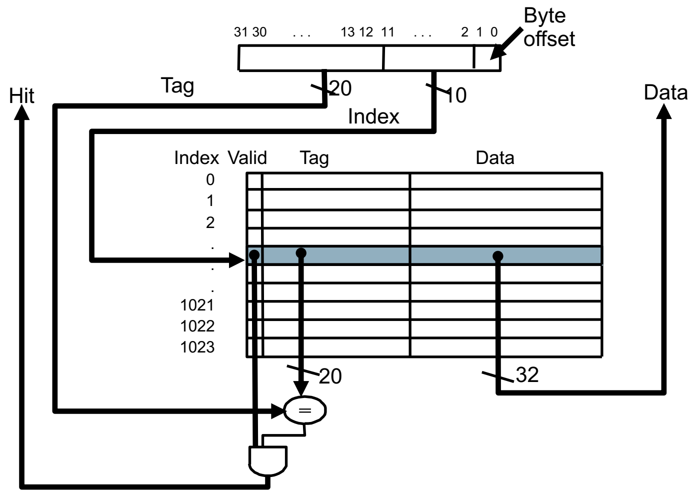
15 miss

01	Mem(4)
00	Mem(1)
00	Mem(2)
11 ~~00~~	Mem(~~3~~) 15

- 8 requests, 6 misses

MIPS Direct Mapped Cache Example

- One word blocks, cache size = 1K words (or 4KB)



Taking Advantage of Spatial Locality

- Start with an empty cache - all blocks initially **not valid**.
- Let cache block hold more than one word.

0000 0001 0010 0011 0100 0011 0100 1111

0 miss

00	Mem(1)	Mem(0)

1 hit

00	Mem(1)	Mem(0)

2 miss

00	Mem(1)	Mem(0)
00	Mem(3)	Mem(2)

3 hit

00	Mem(1)	Mem(0)
00	Mem(3)	Mem(2)

4 miss

01 ~~00~~	~~Mem(1)~~ 5	~~Mem(0)~~ 4
00	Mem(3)	Mem(2)

3 hit

01	Mem(5)	Mem(4)
00	Mem(3)	Mem(2)

4 hit

01	Mem(5)	Mem(4)
00	Mem(3)	Mem(2)

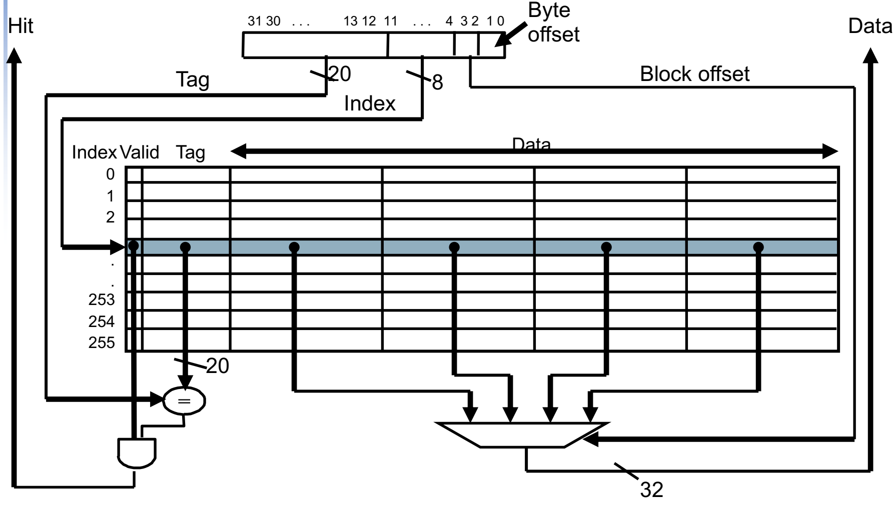
15 miss

01	Mem(5)	Mem(4)
11 ~~00~~	~~Mem(3)~~ 15	~~Mem(2)~~ 14

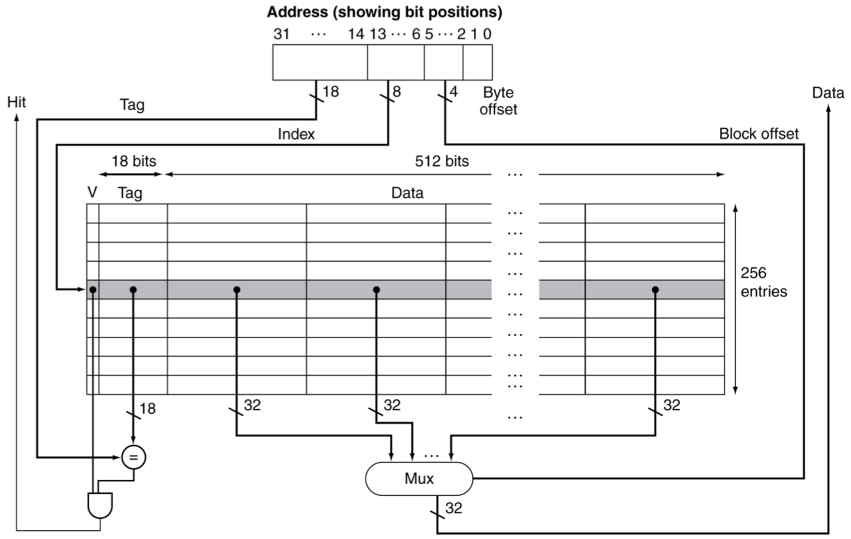
- 8 requests, 4 misses

Multiword Block Direct Mapped Cache

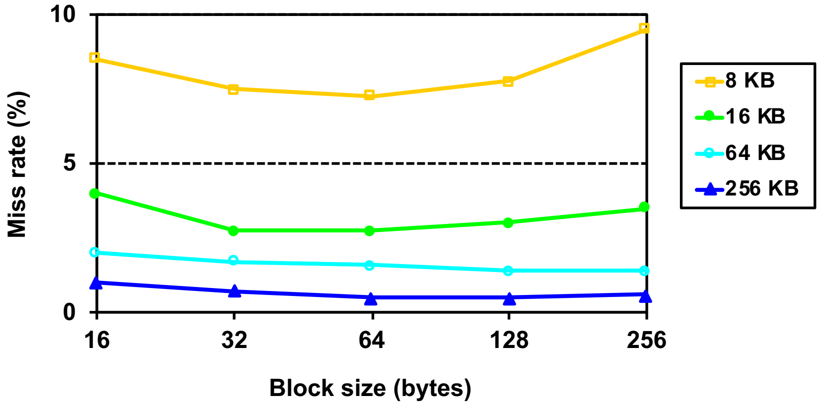
- Four words/block, cache size = 1K words



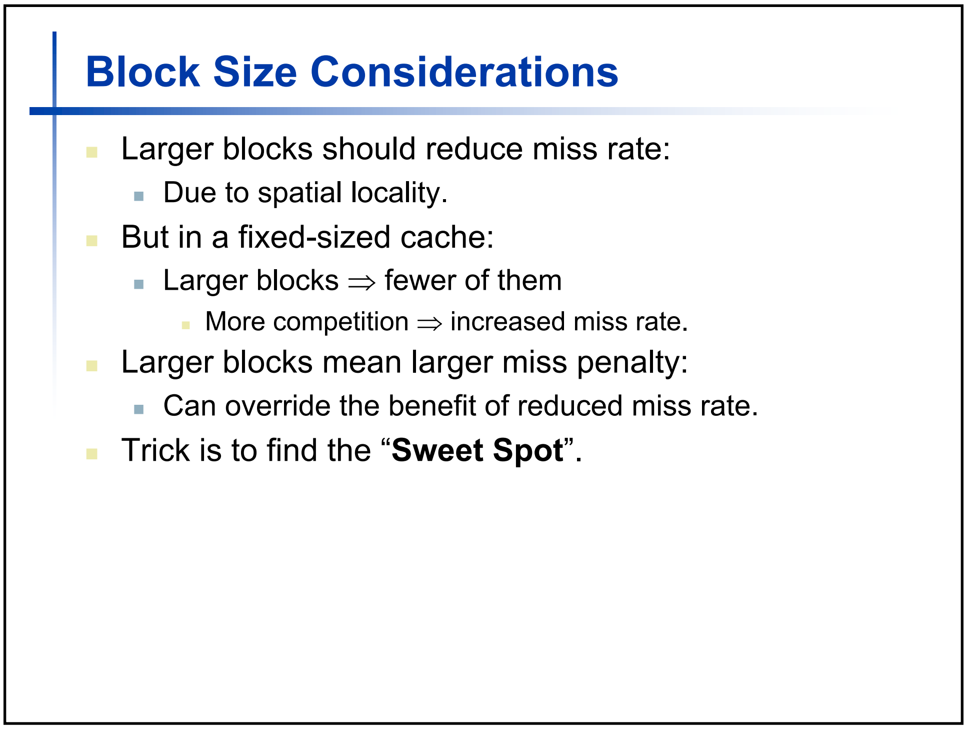
Example: Intrinsity FastMATH



Miss Rate vs Block Size vs Cache Size

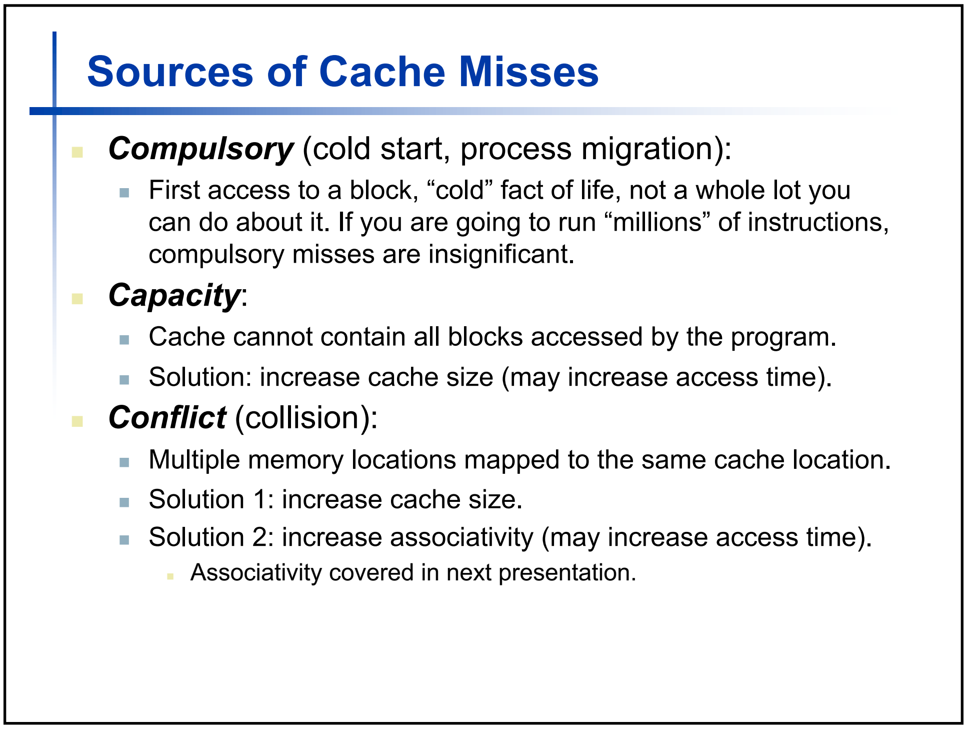


- Miss rate goes up if the block size becomes a significant fraction of the cache size because the number of blocks that can be held in the same size cache is smaller.



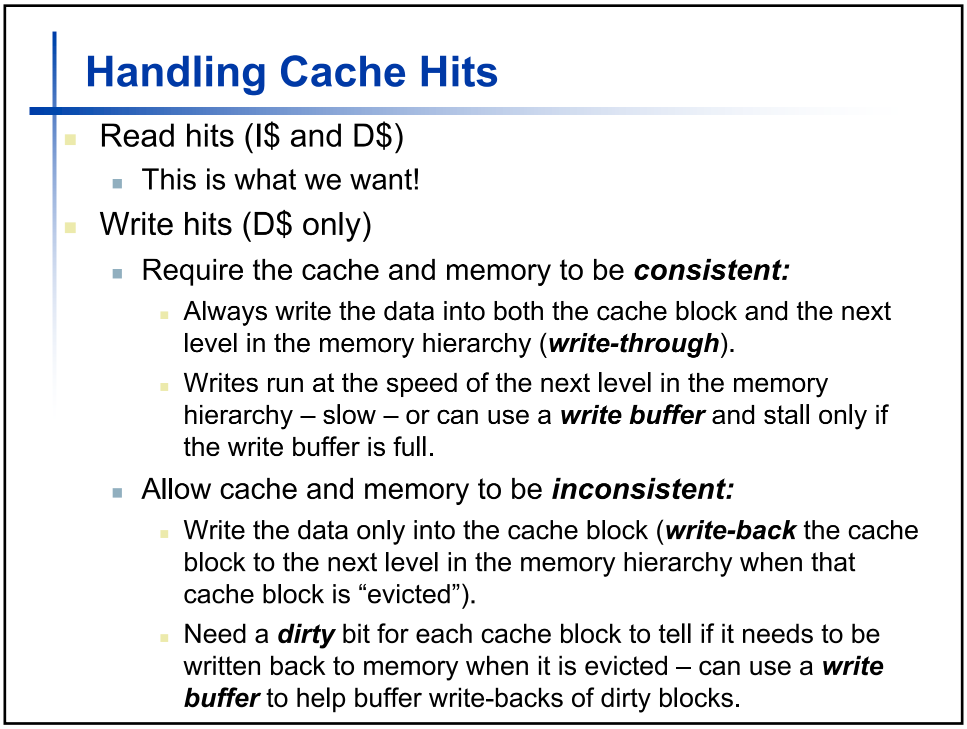
Block Size Considerations

- Larger blocks should reduce miss rate:
  - Due to spatial locality.
- But in a fixed-sized cache:
  - Larger blocks ⇒ fewer of them
    - More competition ⇒ increased miss rate.
- Larger blocks mean larger miss penalty:
  - Can override the benefit of reduced miss rate.
- Trick is to find the “**Sweet Spot**”.



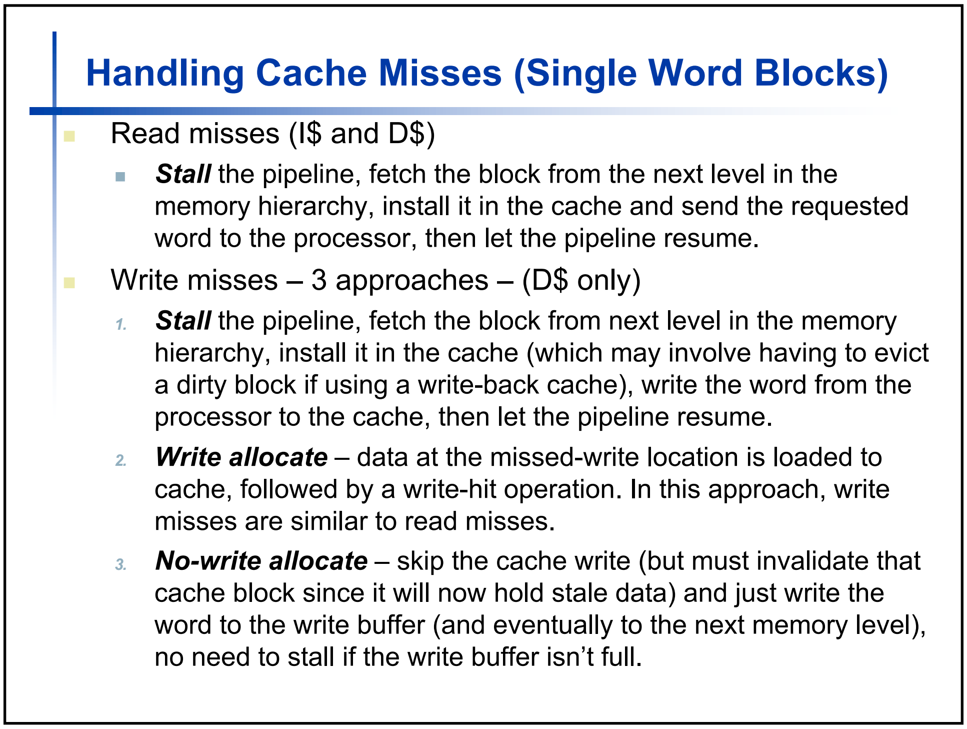
Sources of Cache Misses

- ***Compulsory*** (cold start, process migration):
  - 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 instructions, compulsory misses are insignificant.
- ***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 (may increase access time).
    - Associativity covered in next presentation.



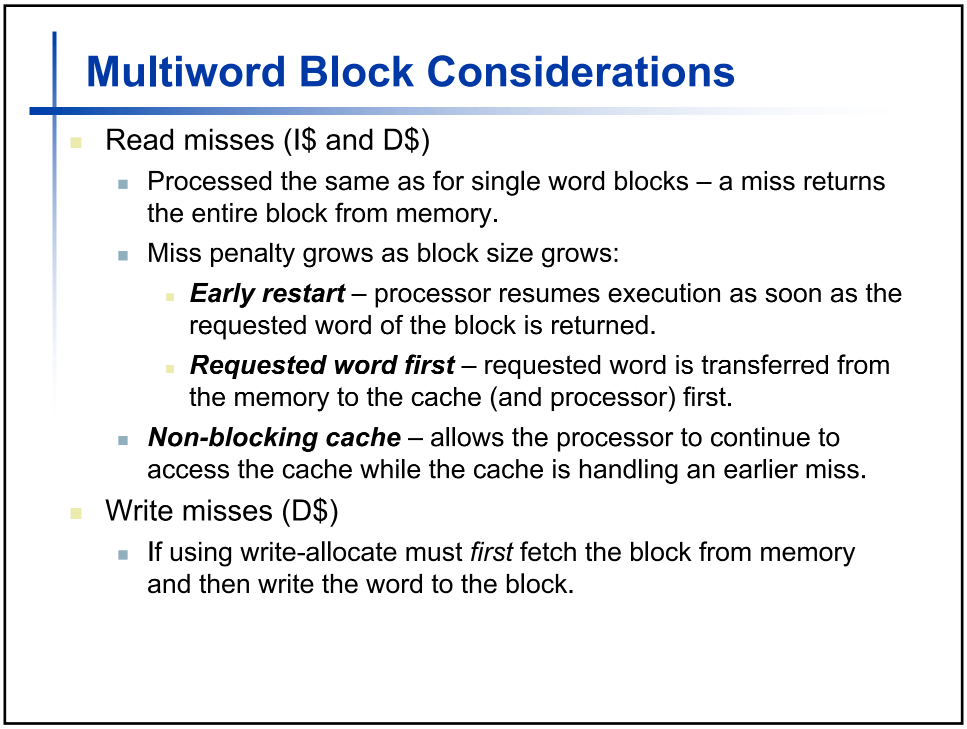
Handling Cache Hits

- 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 – 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 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.



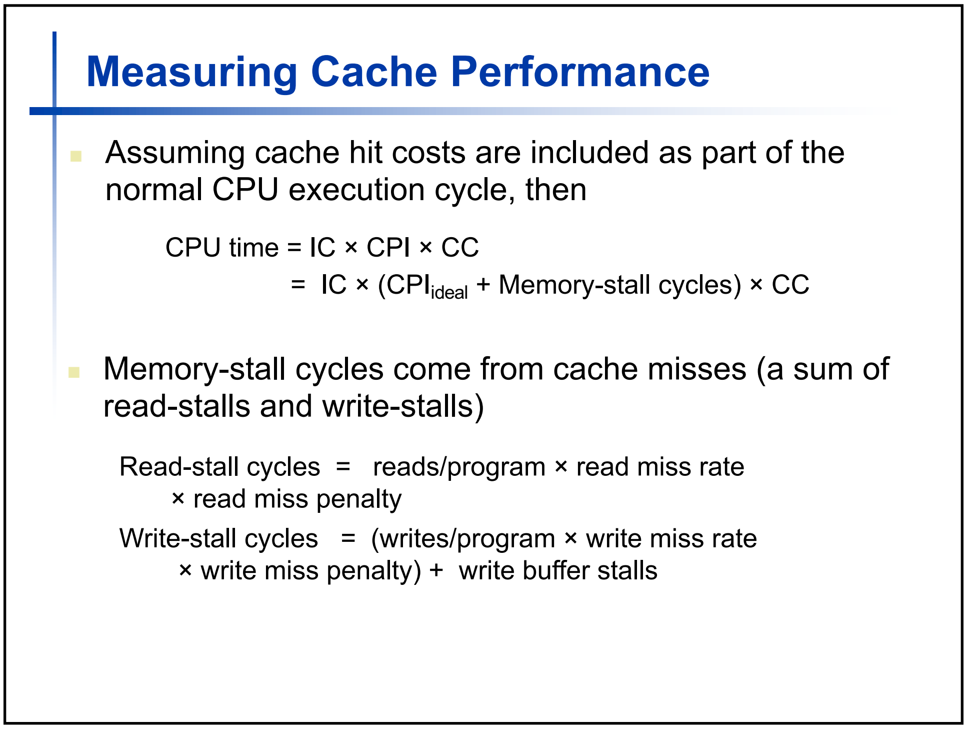
Handling Cache Misses (Single Word Blocks)

- Read misses (I$ and D$)
  - ***Stall*** the pipeline, fetch the block from the next level in the memory hierarchy, install it in the cache and send the requested word to the processor, then let the pipeline resume.
- Write misses – 3 approaches – (D$ 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.
  2. ***Write allocate*** – data at the missed-write location is loaded to cache, followed by a write-hit operation. In this approach, write misses are similar to read misses.
  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$)
  - Processed the same as for single word blocks – a miss returns the entire block from memory.
  - Miss penalty grows as block size grows:
    - ***Early restart*** – processor resumes execution as soon as the requested word of the block is returned.
    - ***Requested word first*** – requested word is transferred from the memory to the cache (and processor) first.
  - ***Non-blocking cache*** – allows the processor to continue to access the cache while the cache is handling an earlier miss.
- Write misses (D$)
  - If using write-allocate must *first* fetch the block from memory and then write the word to the block.



Measuring Cache Performance

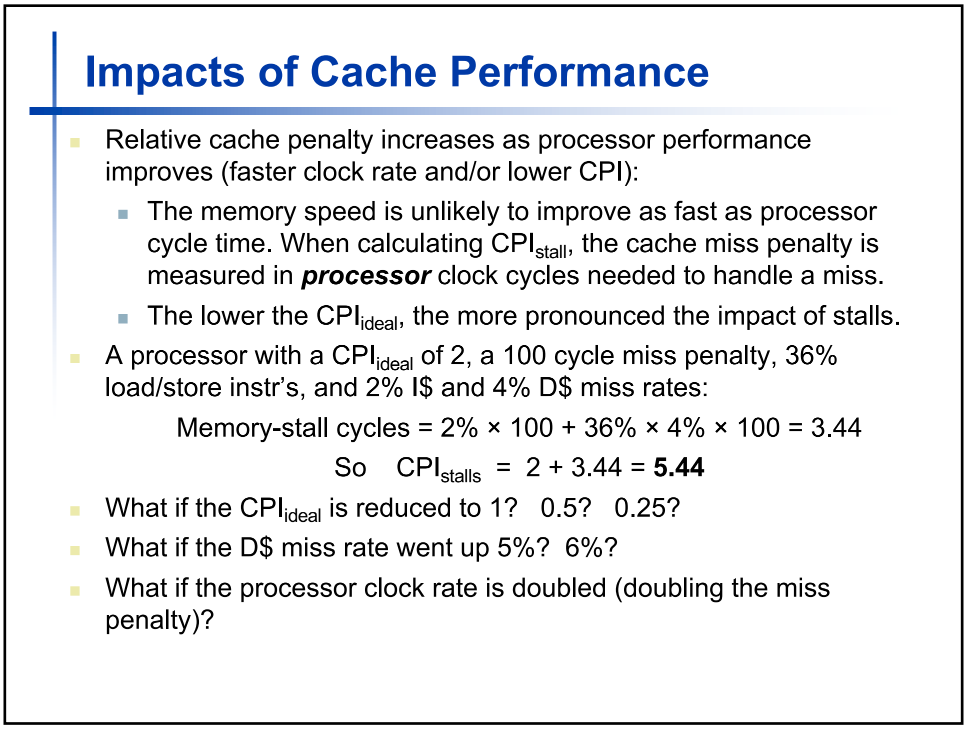
- Assuming cache hit costs are included as part of the normal CPU execution cycle, then

$$\begin{aligned} \text{CPU time} &= \text{IC} \times \text{CPI} \times \text{CC} \\ &= \text{IC} \times (\text{CPI}_{\text{ideal}} + \text{Memory-stall cycles}) \times \text{CC} \end{aligned}$$

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

Read-stall cycles = reads/program × read miss rate × read miss penalty

Write-stall cycles = (writes/program × write miss rate × write miss penalty) + write buffer stalls

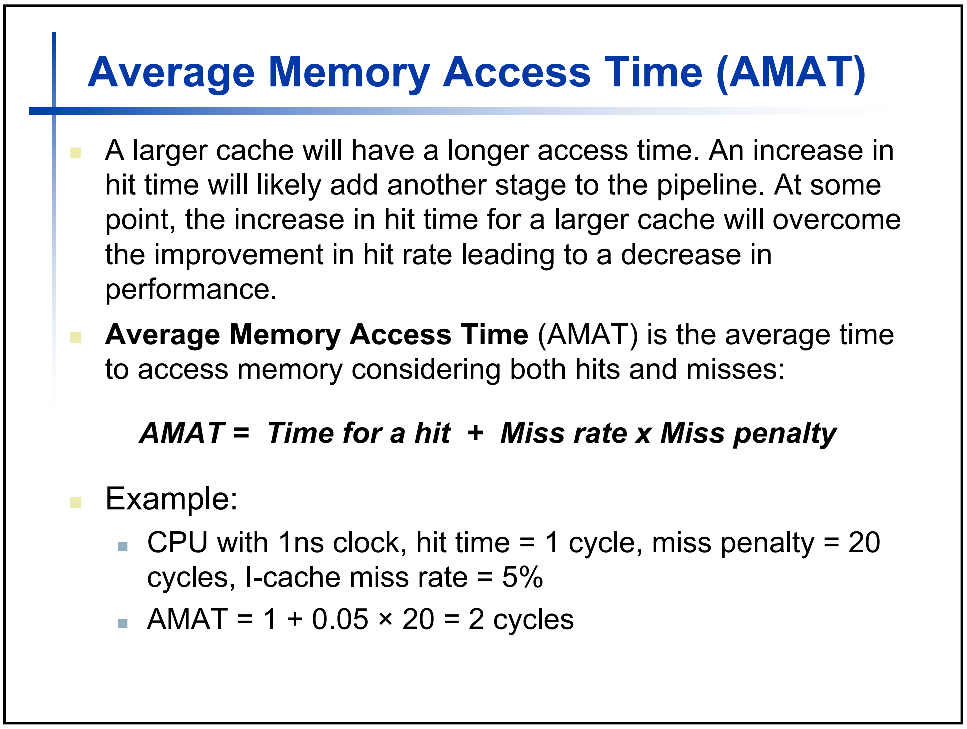


Impacts of Cache Performance

- 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

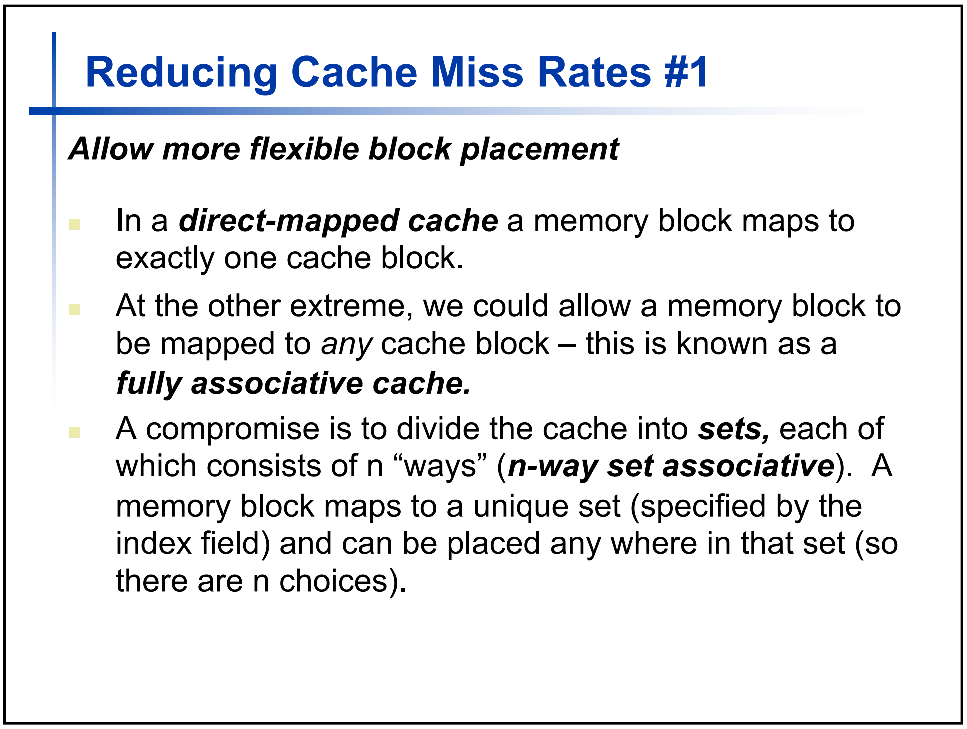
  So $\text{CPI}_{\text{stalls}}$ = 2 + 3.44 = **5.44**
- What if the $\text{CPI}_{\text{ideal}}$ is reduced to 1? 0.5? 0.25?
- What if the D$ miss rate went up 5%? 6%?
- What if the processor clock rate is doubled (doubling the miss penalty)?



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 time to access memory considering both hits and misses:

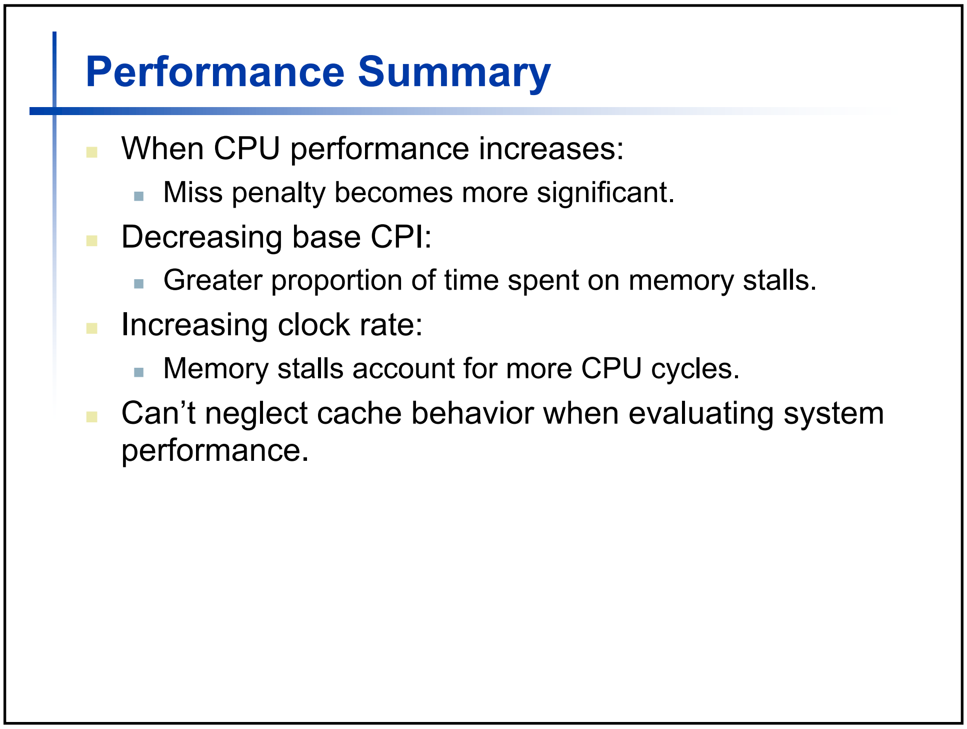
  AMAT = Time for a hit + Miss rate x Miss penalty
- Example:
  - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, I-cache miss rate = 5%
  - AMAT = 1 + 0.05 × 20 = 2 cycles



Reducing Cache Miss Rates #1

Allow more flexible block placement

- In a ***direct-mapped cache*** a memory block maps to exactly one cache block.
- At the other extreme, we could allow a memory block to be mapped to *any* cache block – this is known as a ***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 any where in that set (so there are n choices).



Performance Summary

- When CPU performance increases:
  - Miss penalty becomes more significant.
- Decreasing base CPI:
  - Greater proportion of time spent on memory stalls.
- Increasing clock rate:
  - Memory stalls account for more CPU cycles.
- Can’t neglect cache behavior when evaluating system performance.