Chapter 5A

Large and Fast: Exploiting Memory Hierarchy

Memory Technology

- Static RAM (SRAM)
  - Fast, expensive
- Dynamic RAM (DRAM)
  - In between
- Magnetic disk
  - Slow, inexpensive
- Ideal memory
  - Access time of SRAM
  - Capacity and cost/GB of disk

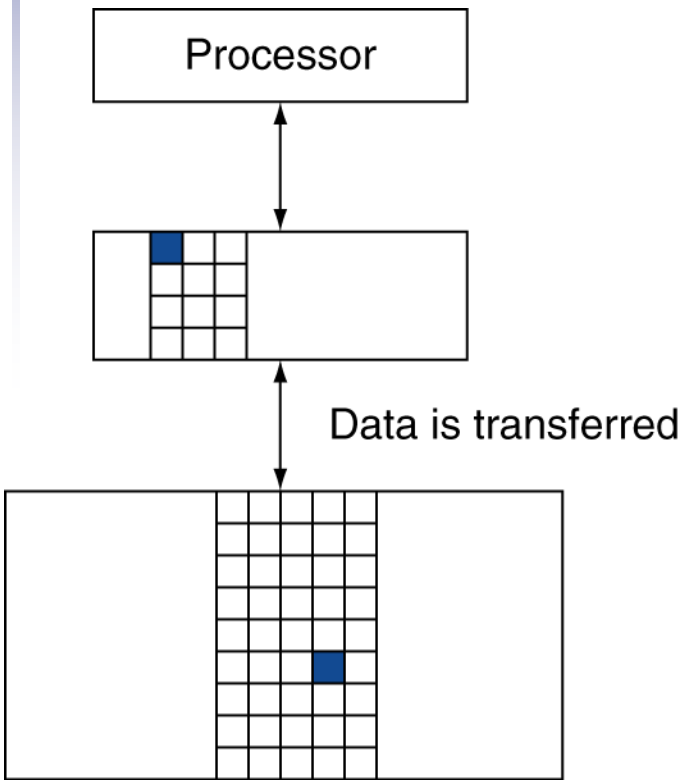
Principle of Locality

- Programs access a small proportion of their address space at any time
- Temporal locality
  - Items accessed recently are likely to be accessed again soon
  - e.g., instructions in a loop, induction variables
- Spatial locality
  - Items near those accessed recently are likely to be accessed soon
  - E.g., sequential instruction access, array data

Taking Advantage of Locality

- Memory hierarchy
- Store everything on disk
- Copy recently accessed (and nearby) items from disk to smaller DRAM memory
  - Main memory
- Copy more recently accessed (and nearby) items from DRAM to smaller SRAM memory
  - Cache memory attached to CPU

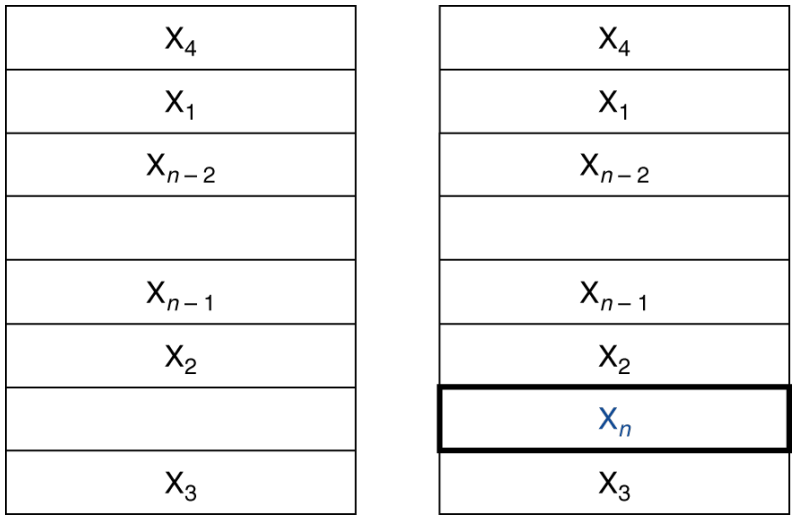
Memory Hierarchy Levels



- Block (aka line): unit of copying
  - May be multiple words
- If accessed data is present in upper level
  - Hit: access satisfied by upper level
    - Hit ratio: hits/accesses
- If accessed data is absent
  - Miss: block copied from lower level
    - Time taken: miss penalty
    - Miss ratio: misses/accesses
      = 1 – hit ratio
  - Then accessed data supplied from upper level

Cache Memory

- Cache memory
  - The level of the memory hierarchy closest to the CPU
- Given accesses X_1, …, X_{n-1}, X_n

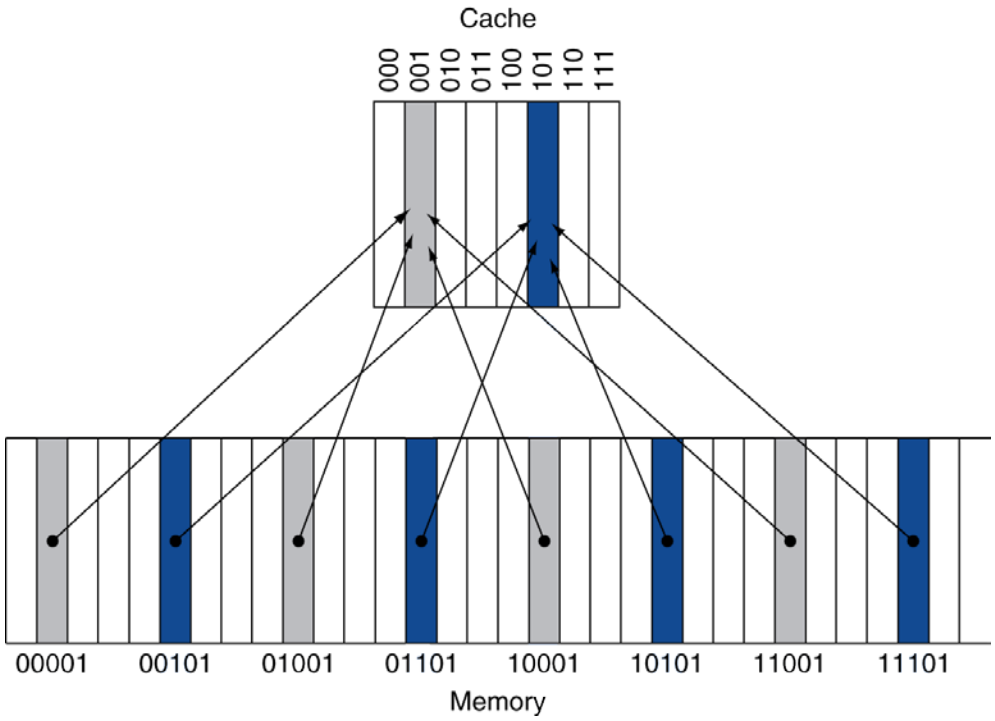


a. Before the reference to X_n b. After the reference to X_n

- How do we know if the data is present?
- Where do we look?

Direct Mapped Cache

- Location determined by address
- Direct mapped: only one choice
  - (Block address) modulo (#Blocks in cache)



- #Blocks is a power of 2
- Use low-order address bits

Tags and Valid Bits

- How do we know which particular block is stored in a cache location?
  - Store block address as well as the data
  - Actually, only need the high-order bits
  - Called the tag
- What if there is no data in a location?
  - Valid bit: 1 = present, 0 = not present
  - Initially 0

Cache Example

- 8-blocks, 1 word/block, direct mapped
- Initial state

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Miss	110

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	**Y**	**10**	**Mem[10110]**
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
26	11 010	Miss	010

Index	V	Tag	Data
000	N		
001	N		
010	**Y**	**11**	**Mem[11010]**
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Hit	110
26	11 010	Hit	010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
16	10 000	Miss	000
3	00 011	Miss	011
16	10 000	Hit	000

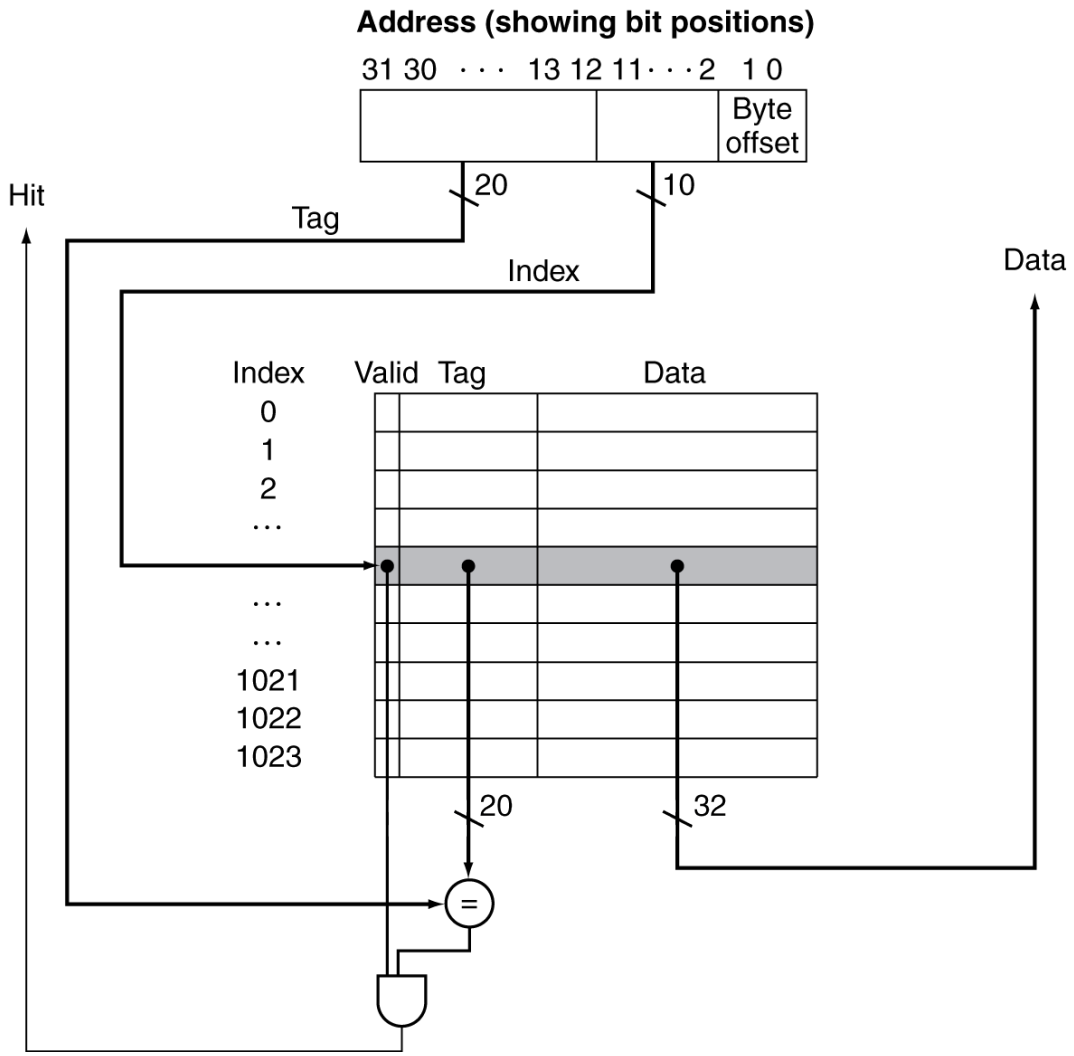
Index	V	Tag	Data
000	**Y**	**10**	**Mem[10000]**
001	N		
010	Y	11	Mem[11010]
011	**Y**	**00**	**Mem[00011]**
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	**Y**	**10 ← 11**	**Mem[10010] ← Mem[11010]**
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

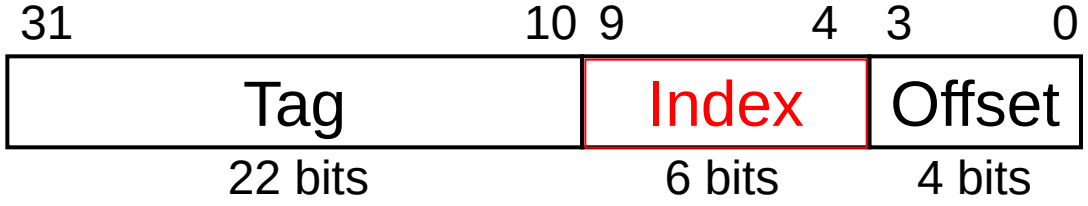
Direct mapped Cache



Assumption:
1 block = 1 word

Example: Larger Block Size

- 64 blocks, 16 bytes/block
  - To what block number does address 1200 map?
  - 0000 0000 0000 0000 0000 0100 1011 0000
- Block address = $\lfloor 1200/16 \rfloor = 75$
  - 0000 0000 0000 0000 0000 0100 1011 0000
- Block number = 75 modulo 64 = 11
  - 0000 0000 0000 0000 0000 0100 1011 0000



Block Size Considerations

- Larger blocks should reduce miss rate
  - Due to spatial locality
- But in a fixed-sized cache
  - Larger blocks ⇒ fewer entries
  - More competition ⇒ increased miss rate
- Larger miss penalty
  - Can override benefit of reduced miss rate
- Hiding transfer time ⇒ reducing effective miss penalty
  - early restart : as soon as available
  - critical-word-first

Miss rate versus Block Size

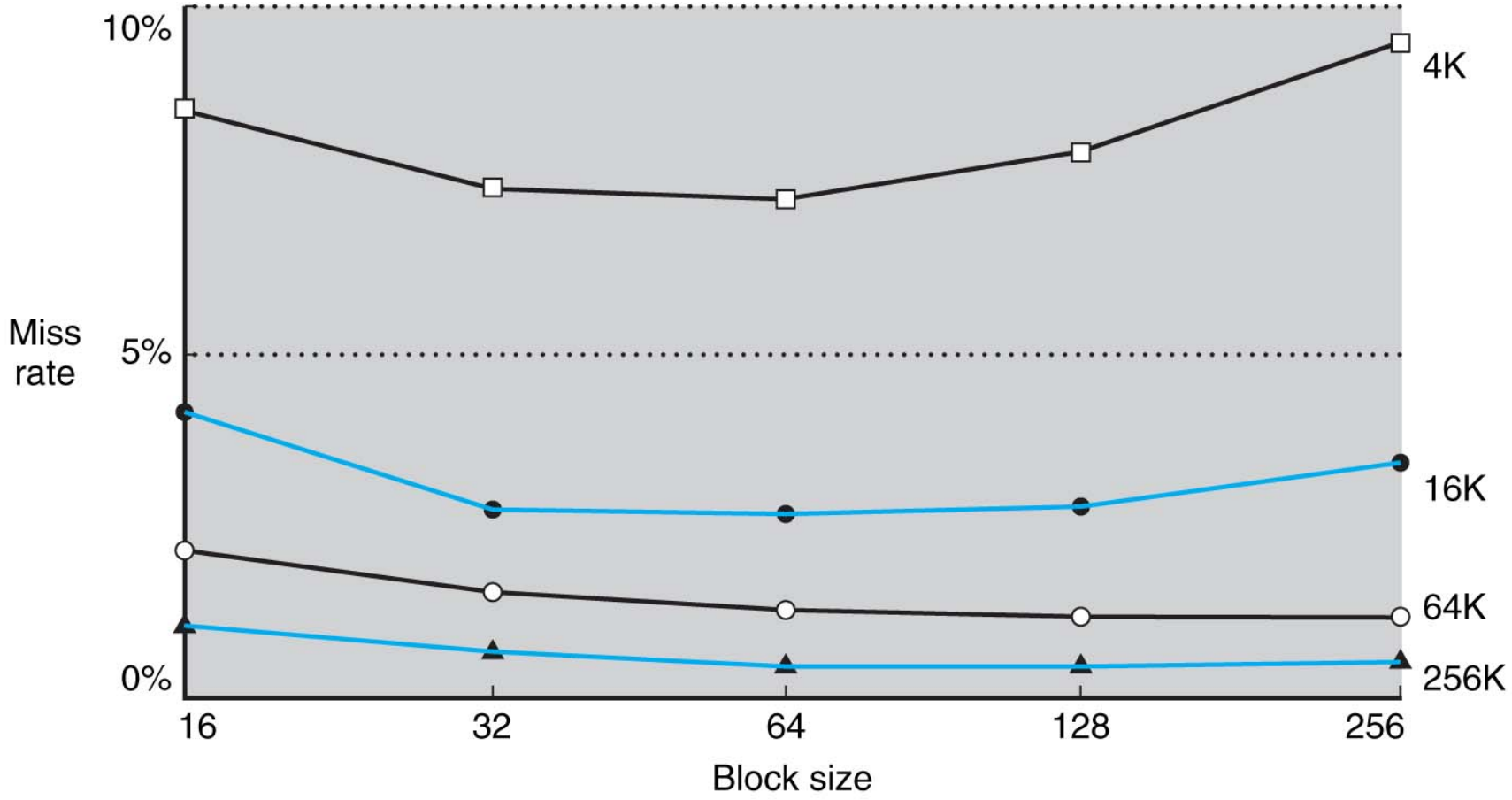


FIGURE 5.8 Miss rate versus block size. Note that the miss rate actually goes up if the block size is too large relative to the cache size. Each line represents a cache of different size. (This figure is independent of associativity, discussed soon.) Unfortunately, SPEC2000 traces would take too long if block size were included, so this data is based on SPEC92.

Instruction Cache Misses

- On cache hit, CPU proceeds normally
- On instruction cache miss
  - Stall the CPU pipeline
  - Invalidate instruction register
  - Send address (PC-4) to the memory
  - Fetch a corresponding block from memory
  - Write the cache entry
    - Put the data from memory in the data portion
    - Write its tag
    - Turn valid bit on
  - Restart Instruction fetch

Writes into data cache

- The cache and memory can be inconsistent
- How can we keep them consistent?
- The simplest way is to write the data into both the memory and the cache: write through
- But makes writes take longer
  - If base CPI = 1, 10% of instructions are stores, write to memory takes 10 cycles, Effective CPI = 1 + 0.1×10 = 2
    - assuming wait until completion of memory write (write stall)
- A solution: write buffer
  - Write the data into the cache and into the write buffer
  - Write buffer: holds data waiting to be written to memory
  - CPU continues immediately
    - Only stalls on write if write buffer is already full

Write-Back (copy back)

- Alternative: On data-write hit, just update the data is written only to the block in cache
  - Keep track of whether each block is dirty (updated)
- When a dirty block is replaced (on a miss)
  - Write it back to the lower-level memory
  - Can use a write buffer to allow replacing block to be read first before reading it from the lower level memory
- Write-back schemes can improve performance, especially when processors generates writes as fast or faster than the writes can be handled by main memory

Write Allocation

- What should happen on a write miss?
- Write miss for a write-through cache
  - No-write allocate: modified only in memory
    - Since some programs often write a whole block before reading it (e.g., initialization)
- Write miss for a write-back cache
  - Write allocate: fetch the block, followed by write hit action. Write misses act like read misses
- Write allocation policy: some computer allow it to be changed on a per page basis

Write in a write-through cache

- It can write the data into the cache and read a tag
- What if the tag mismatches? It is a miss.
- Overwriting the block is not catastrophic
  - Only need to be invalidated

Write in a write-back cache

- If we have a cache miss and the data in the cache is dirty, we must first the block back to memory.
  - Overwriting is not possible
- If we simply overwrite the block before we knew whether the store is hit or not (as we could for a write-through cache), we would destroy the contents of the blocks before being backed up in memory.
- Therefore, stores either require two cycles or require a write buffer to hold that data.

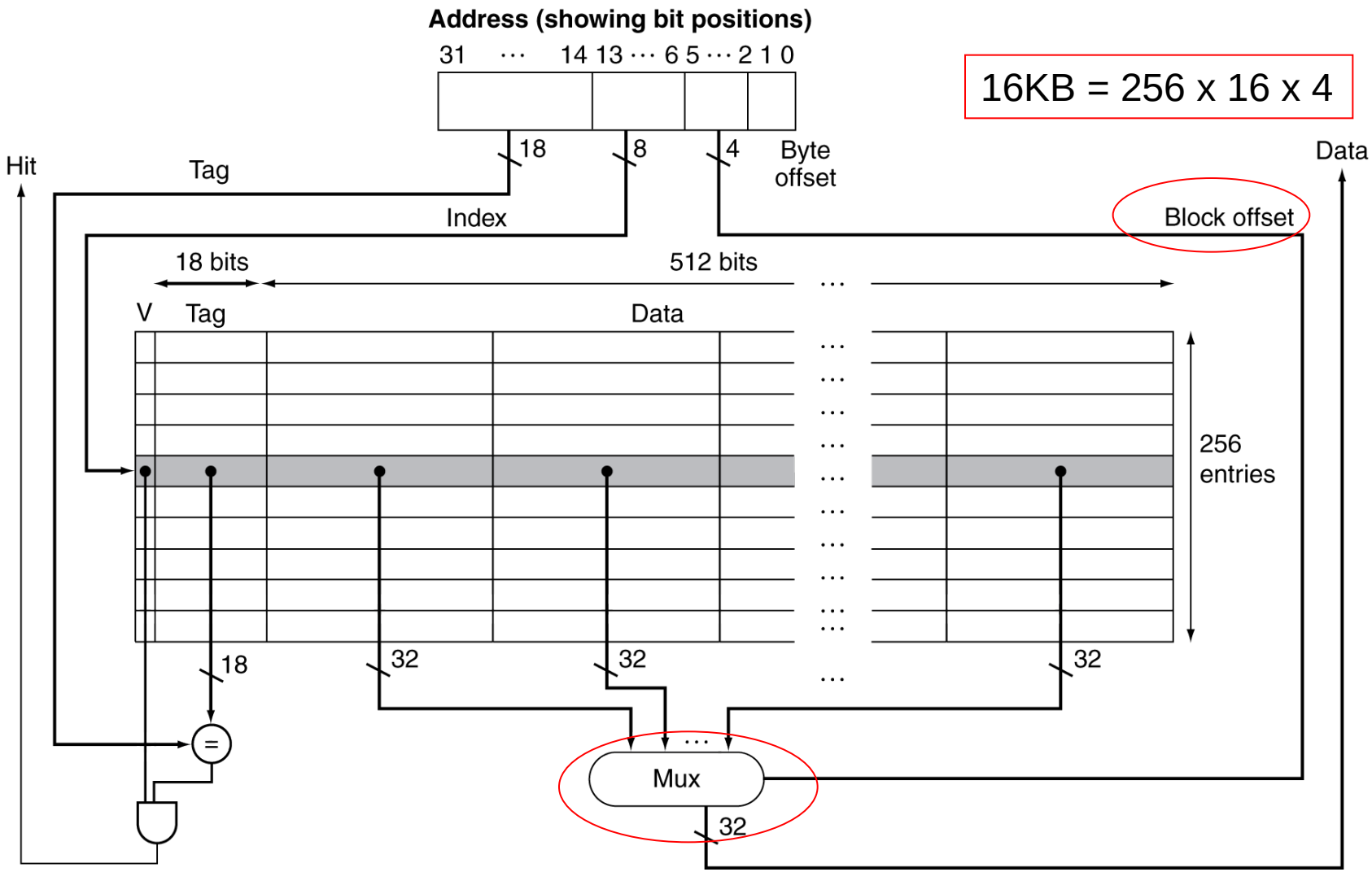
Write in a write-back cache

- Two-cycle write
  - A cycle to check for a hit
  - A cycle to actually perform the write if it is hit
- Write in the store buffer
  - Check for a hit and write the data in the store buffer
  - Assuming a cache hit, the new data is written from the store buffer into the cache on the next unused cache access cycle.
- Many write-back caches also include write buffers that are used to reduce the miss penalty when a miss replaces a modified block
  - How is the miss penalty reduced?

Example: Intrinsity FastMATH

- Embedded MIPS processor
  - 12-stage pipeline
  - Instruction and data access on each cycle
- Split cache: separate I-cache and D-cache
  - Each 16KB: 256 blocks × 16 words/block
  - D-cache: write-through or write-back
  - What is a combined (unified) cache?
- SPEC2000 miss rates
  - I-cache: 0.4%
  - D-cache: 11.4%
  - Weighted average: 3.24%

Example: Intrinsity FastMATH



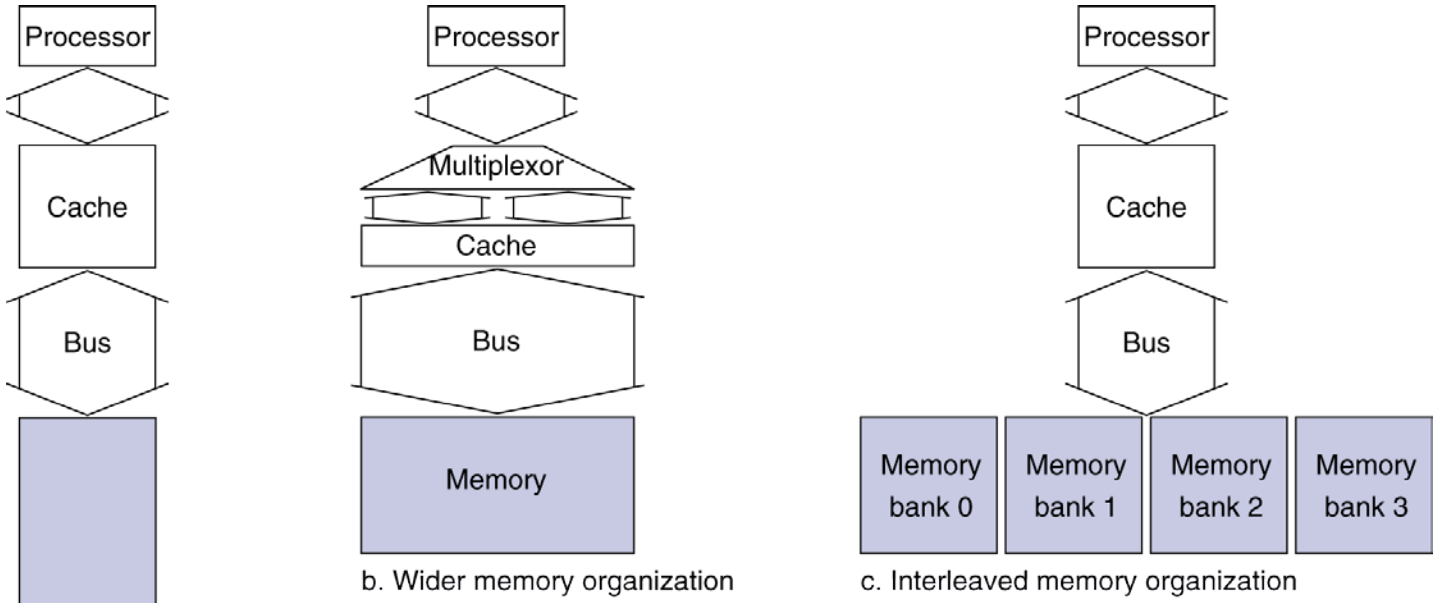
Unified cache

- A combined cache will usually have a better hit rate if the total size is equal to the sum of the two split caches
  - 3.18%
- Nonetheless, many processors use a split instruction and data cache to increase cache bandwidth.
- This observation cautions us that we cannot use miss rate as the sole measure of cache performance.
  - Memory bandwidth

Main Memory Supporting Caches

- Use DRAMs for main memory
  - Fixed width (e.g., 1 word = 4 bytes)
  - Connected with CPU by fixed-width clocked bus
    - Bus clock is typically slower than CPU clock
- Example cache block read
  - 1 bus cycle for address transfer
  - 15 bus cycles per DRAM access
  - 1 bus cycle per data transfer
- For 4-word block, 1-word-wide DRAM
  - Miss penalty = 1 + 4×15 + 4×1 = 65 bus cycles
  - Bandwidth = 16 bytes / 65 cycles = 0.25 B/cycle

Increasing Memory Bandwidth



a. One-word-wide memory organization

b. Wider memory organization

c. Interleaved memory organization

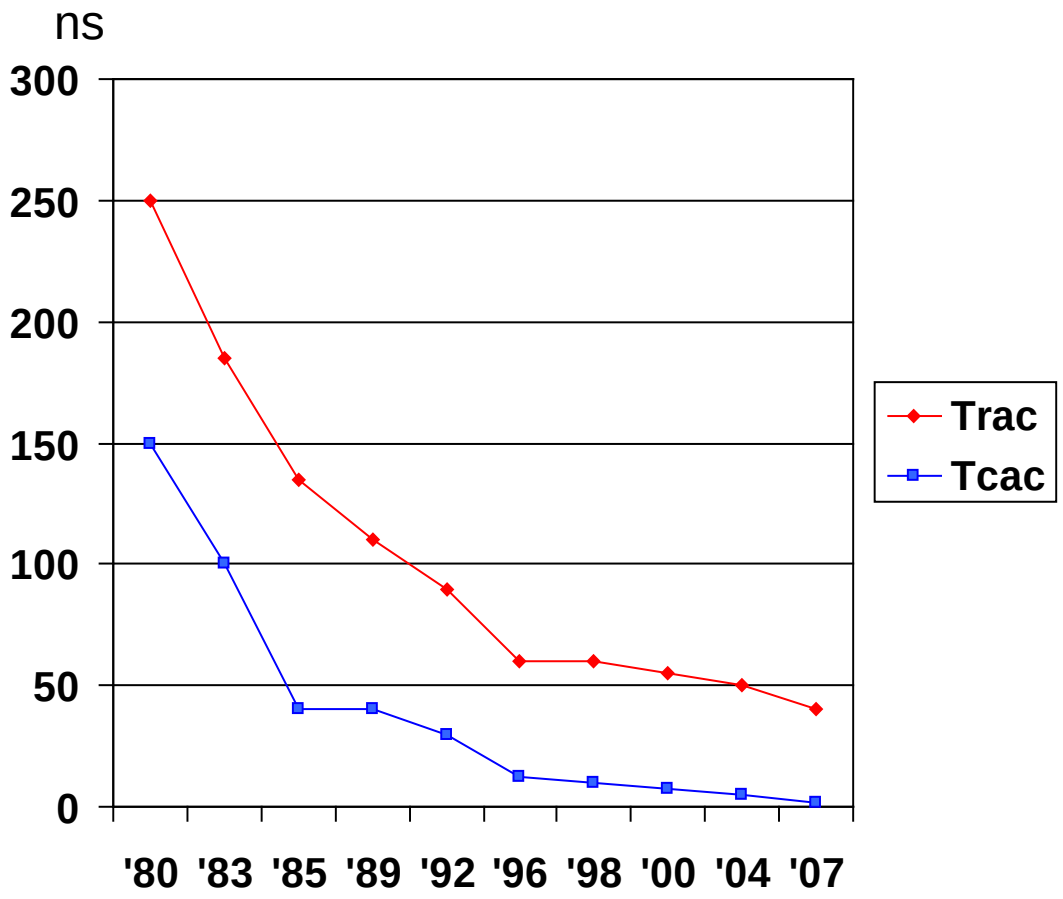
- 4-word wide memory
  - Miss penalty = 1 + 15 + 1 = 17 bus cycles
  - Bandwidth = 16 bytes / 17 cycles = 0.94 B/cycle
- 4-bank interleaved memory
  - Miss penalty = 1 + 15 + 4×1 = 20 bus cycles
  - Bandwidth = 16 bytes / 20 cycles = 0.8 B/cycle

Advanced DRAM Organization

- Bits in a DRAM are organized as a rectangular array
  - DRAM accesses an entire row
  - Burst mode: supply successive words from a row with reduced latency
- Double data rate (DDR) DRAM
  - Transfer on rising and falling clock edges
- Quad data rate (QDR) DRAM
  - Separate DDR inputs and outputs

DRAM Generations

Year	Capacity	$/GB
1980	64Kbit	$1500000
1983	256Kbit	$500000
1985	1Mbit	$200000
1989	4Mbit	$50000
1992	16Mbit	$15000
1996	64Mbit	$10000
1998	128Mbit	$4000
2000	256Mbit	$1000
2004	512Mbit	$250
2007	1Gbit	$50



Measuring Cache Performance

- Components of CPU time
  - Program execution cycles
    - Includes cache hit time
  - Memory stall cycles
    - Mainly from cache misses
- With simplifying assumptions:

$$\text{Memory stall cycles} = \frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty}$$

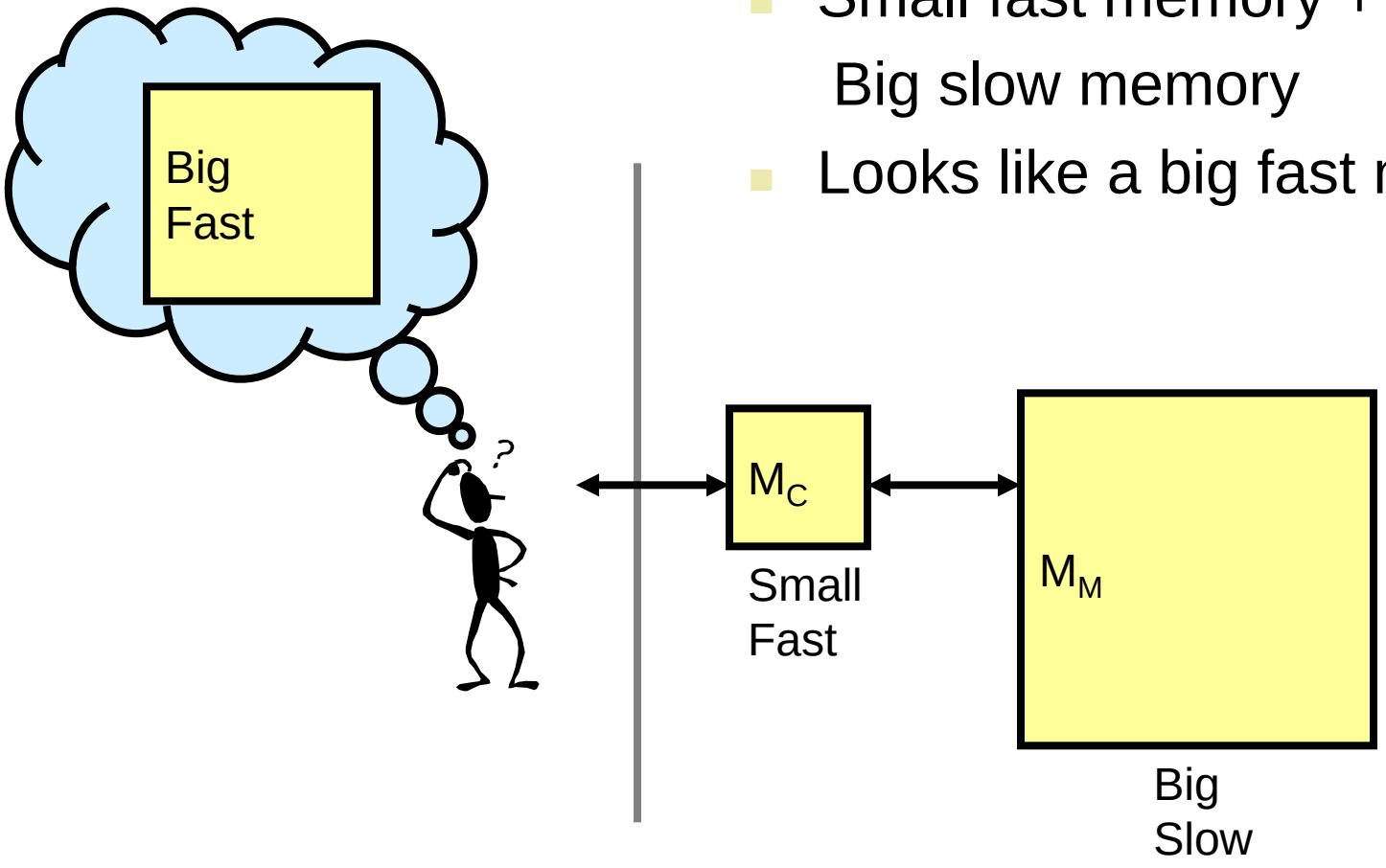
$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}$$

The Memory Bottleneck

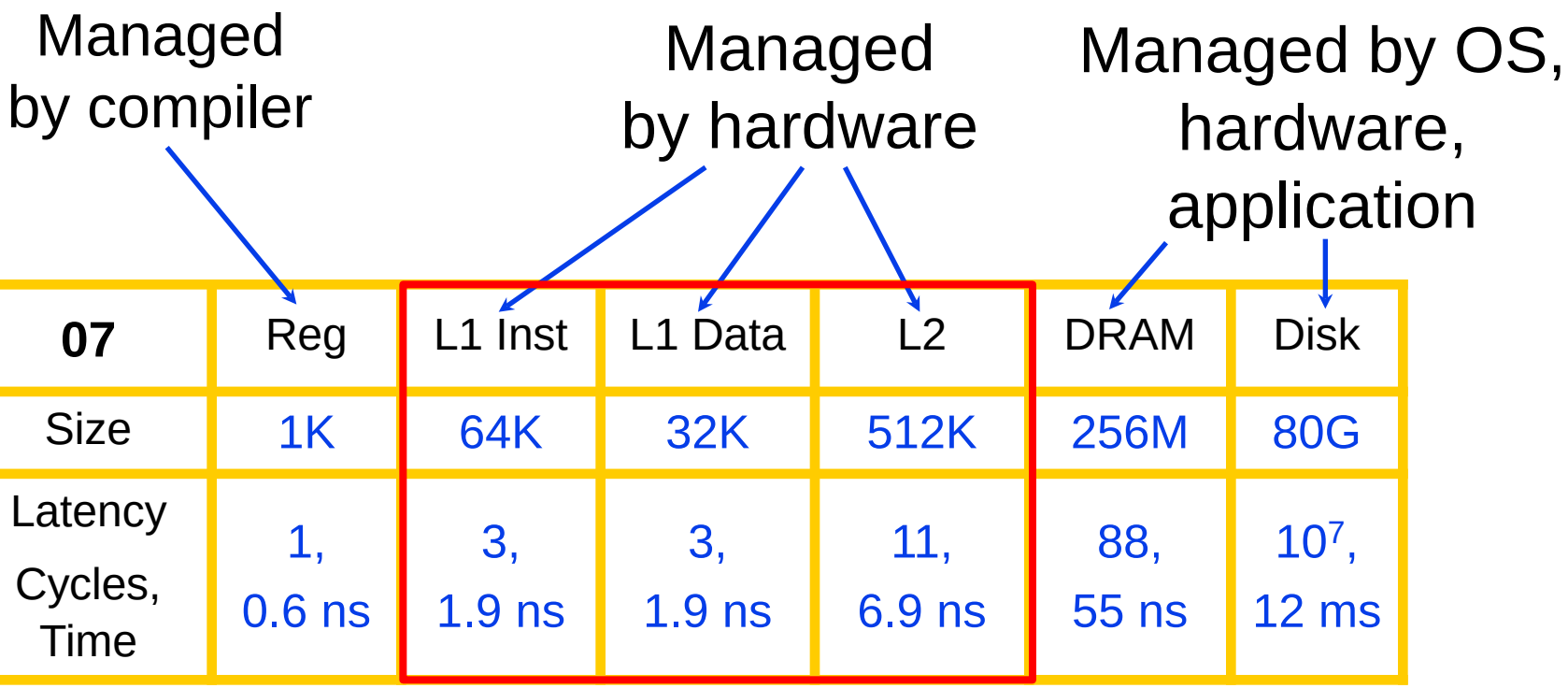
- Typical CPU clock rate
  - 1 GHz (1ns cycle time)
- Typical DRAM access time
  - 30ns (about 30 cycles)
- Typical main memory access
  - 100ns (100 cycles)
    - DRAM (30), precharge (10), chip crossings (30), overhead (30).
- Our pipeline designs assume 1 cycle access (1ns)
- Average instruction references
  - 1 instruction word
  - 0.3 data words
- This problem gets worse
  - CPUs get *faster*
  - Memories get *bigger*
- Memory delay is mostly communication time
  - reading/writing a bit is *fast*
  - it takes time to
    - select the right bit
    - route the data to/from the bit
- Big memories are *slow*
- Small memories can be made *fast*

Cache Memory

- Small fast memory +
  Big slow memory
- Looks like a big fast memory



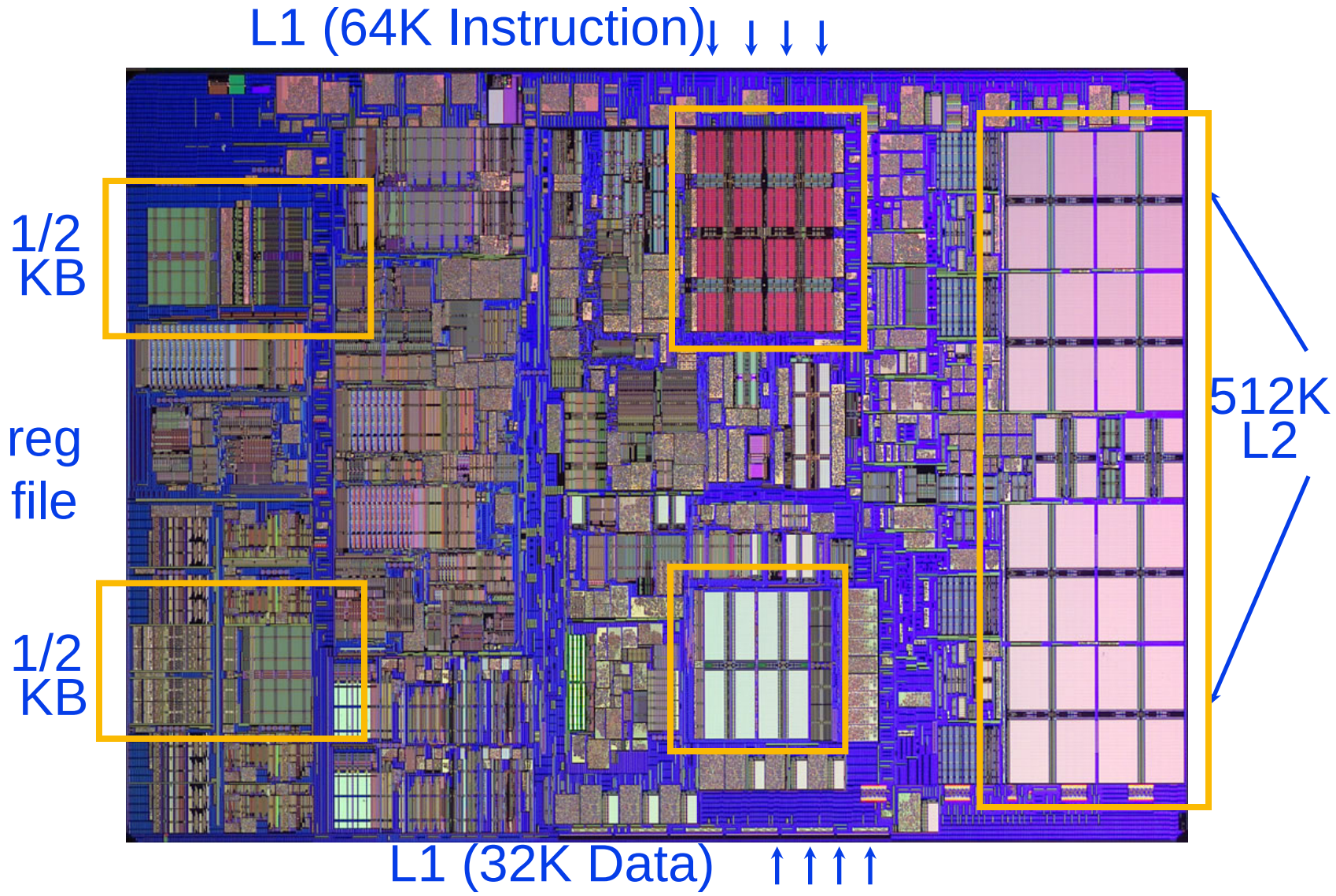
Memory Hierarchy: iMac G5



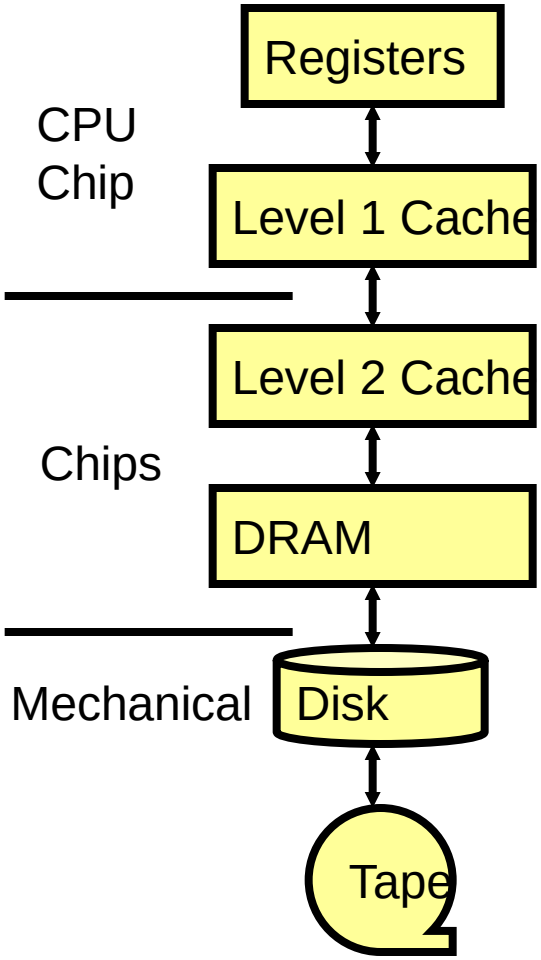
07	Reg	L1 Inst	L1 Data	L2	DRAM	Disk
Size	1K	64K	32K	512K	256M	80G
Latency Cycles, Time	1, 0.6 ns	3, 1.9 ns	3, 1.9 ns	11, 6.9 ns	88, 55 ns	10^7, 12 ms

Goal: Illusion of large, fast, cheap memory

iMac's G5: All caches on-chip



The Memory Hierarchy



Latency	Bandwidth	
1 cyc	3-10 words/cycle < 1KB	compiler managed
1-3cy	1-2 words/cycle 16KB -1MB	hardware managed
10-15cy	1-2 word/cycle 1MB - 12MB	hardware managed
50-300cy	0.5 words/cycle 64MB - 4GB	OS managed
10^6-10^7cy	0.01 words/cycle 40GB+	OS managed

Cache Performance Example

- Given
  - I-cache miss rate = 2%
  - D-cache miss rate = 4%
  - Miss penalty = 100 cycles
  - Base CPI (ideal cache) = 2
  - Load & stores are 36% of instructions
- Miss cycles per instruction
  - I-cache: $0.02 \times 100 = 2$
  - D-cache: $0.36 \times 0.04 \times 100 = 1.44$
- Actual CPI = 2 + 2 + 1.44 = 5.44
  - Ideal CPU is 5.44/2 =2.72 times faster

Average Access Time

- Hit time is also important for performance
- Average memory access time (AMAT)
  - AMAT = Hit time + Miss rate × 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) 1ns = 2ns
    - 2 cycles per instruction

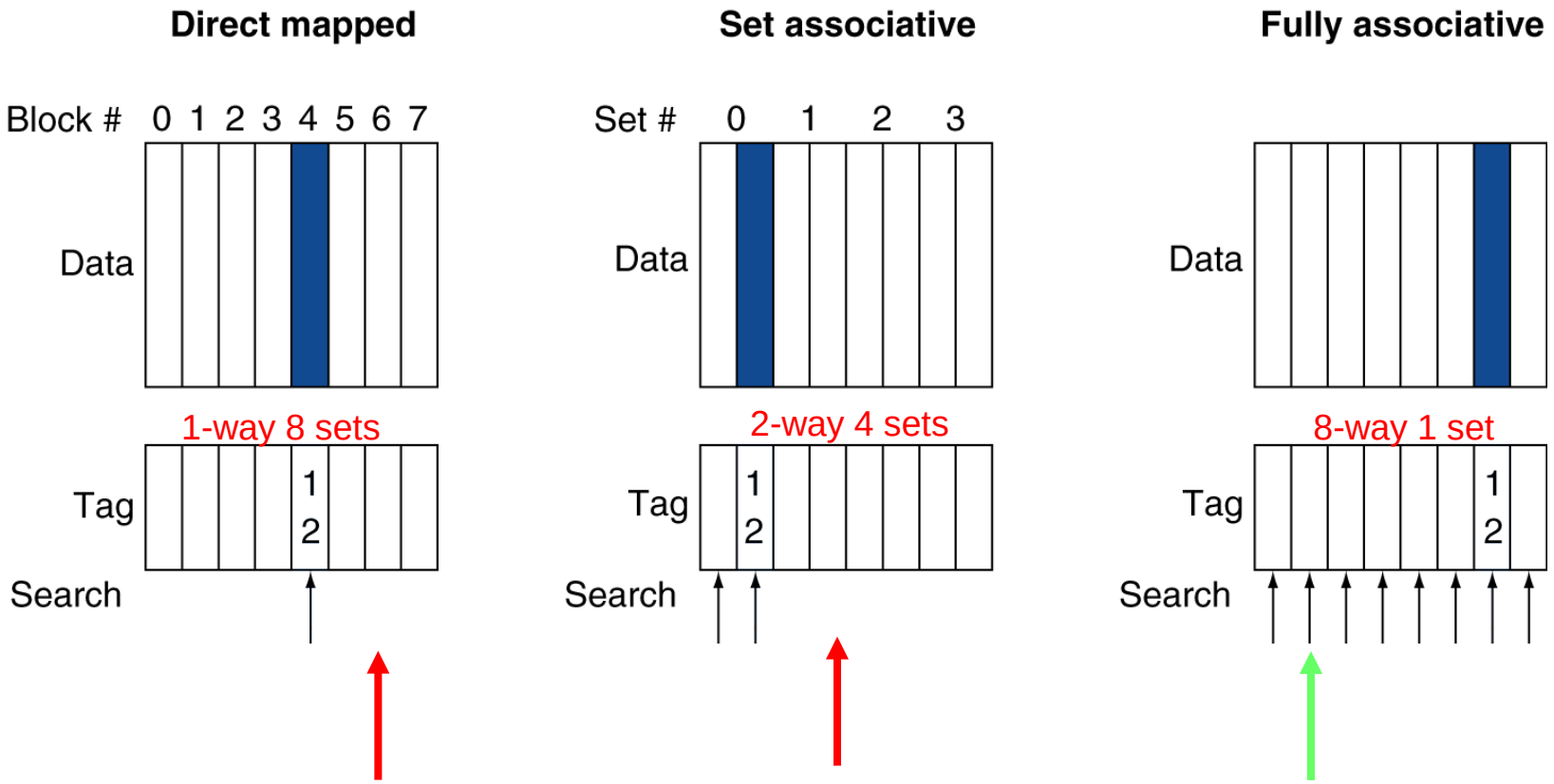
Performance Summary

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

Associative Caches

- Fully associative caches
  - Allow a given block to go in any cache entry
  - Requires all entries to be searched at once
  - Comparator per entry (expensive?!)
- *n*-way set-associative caches
  - Set selection: block number determines which set
  - Each set contains *n* entries → *n* comparators (more sets → less entries/set → less comparators)
  - Allow a given block to go in any entry of the selected set

Associative Cache Example

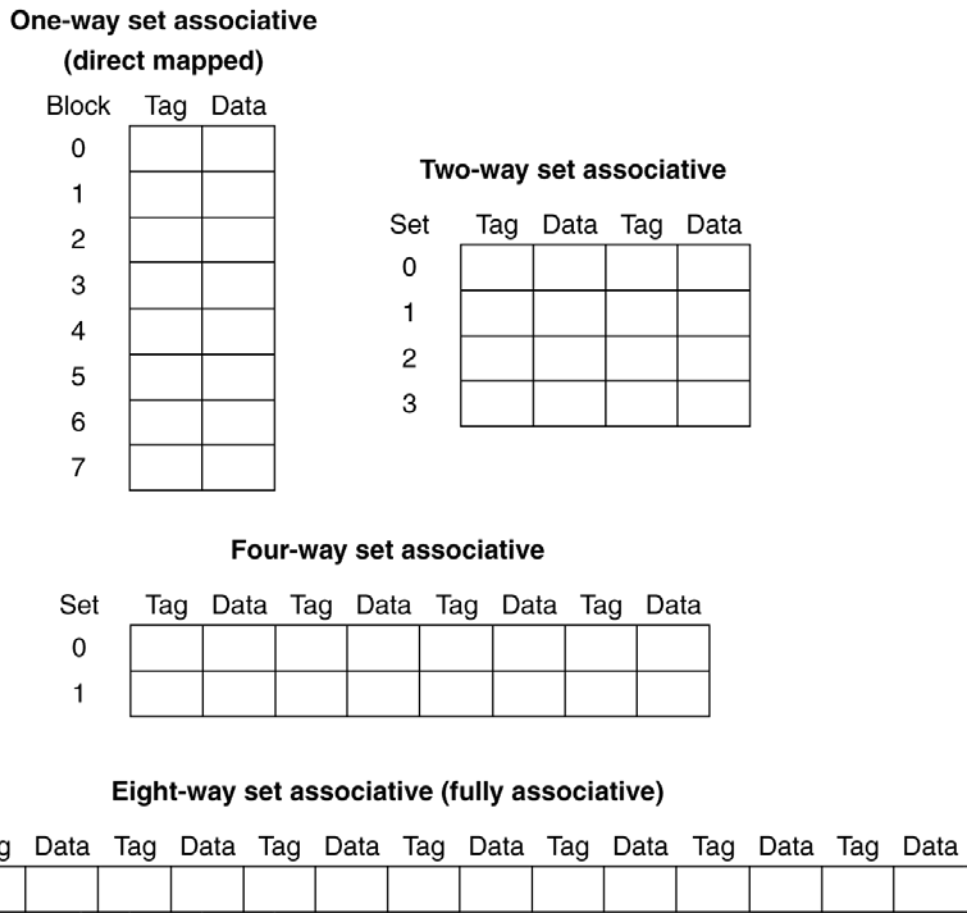


Selected Set = (Block number) modulo (#Sets in cache)

The tag of every element in the set must be compared with the tag of the input address.

Spectrum of Associativity

- For a cache with 8 entries



Associativity Example

- Compare 4-block caches (4 one-word blocks)
  - Direct mapped, 2-way set associative, fully associative
  - Block access sequence: 0, 8, 0, 6, 8
- Direct mapped

Block address	Cache index	Hit/miss	Cache content after access			
			0	1	2	3
0	0	miss	**Mem[0]**			
8	0	miss	**Mem[8]**			
0	0	miss	**Mem[0]**			
6	2	miss	Mem[0]		**Mem[6]**	
8	0	miss	**Mem[8]**		Mem[6]	

Associativity Example

- 2-way set-associative: (even: set 0, odd: set 1)

Block address	Cache index	Hit/miss	Cache content after access			
			Set 0		Set 1	
0	0	miss	**Mem[0]**			
8	0	miss	Mem[0]	**Mem[8]**		
0	0	hit	**Mem[0]**	Mem[8]		
6	0	miss	Mem[0]	**Mem[6]**		
8	0	miss	**Mem[8]**	Mem[6]		

Replacement policy: LRU (least recently used)

- Fully associative

Block address		Hit/miss	Cache content after access			
0		miss	**Mem[0]**			
8		miss	Mem[0]	**Mem[8]**		
0		hit	**Mem[0]**	Mem[8]		
6		miss	Mem[0]	Mem[8]	**Mem[6]**	
8		hit	Mem[0]	**Mem[8]**	Mem[6]	

How Much Associativity

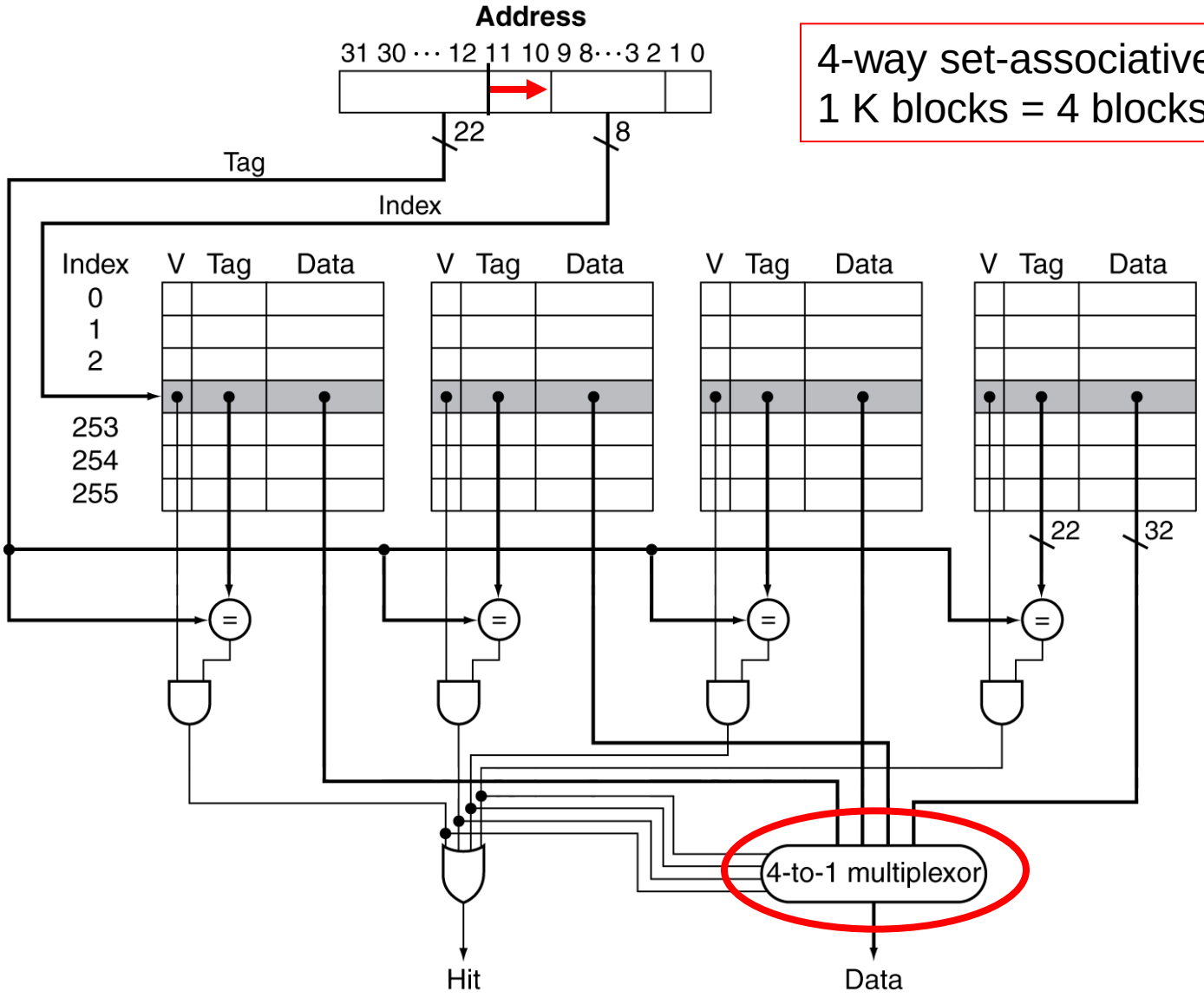
- Increased associativity decreases miss rate
  - But with diminishing returns
- Simulation of a system with 64KB D-cache, 16-word blocks, SPEC2000
  - 1-way: 10.3%
  - 2-way: 8.6%
  - 4-way: 8.3%
  - 8-way: 8.1%

Locating a block in the cache

- The index (n bits) is used to select the set containing the address of interest.
  - Direct mapped: index= n bits → Cache size: 2^n blocks
- If the total cache size is kept the same (2^n blocks), increasing associativity (2^k-way) reduces the number of sets (2^{n-k} sets).
  - A 2^k-way set-associative cache has an (n-k)-bit index to select one set among 2^{n-k} sets .
  - Fully associative (2^n-way) = 1 set = no index
  - The number of way = the number of blocks in a set

Associativity up → # of blocks in a set → # of sets down → index bits down → tag bits up

Set Associative Cache Organization



4-way set-associative cache
1 K blocks = 4 blocks/set x 256 sets

CAM

- Content addressable memory
  - Input: key (tag)
  - Output: address (index)
- SRAM
  - Input: address
  - Output: data
- CAM mean that cache designers can afford to implement much higher set associativity
  - than if they needed to build the hardware out of SRAMs and comparators

Replacement Policy

- Direct mapped: no choice
- Set associative
  - Prefer non-valid entry, if available
  - Otherwise, choose among entries in the set
- Least-recently used (LRU)
  - Choose the one unused for the longest time
    - Simple for 2-way, manageable for 4-way, too hard beyond that
- Random
  - Gives approximately the same performance as LRU for high associativity

Size of tags vs. set associativty

- Increasing associativity requires more comparators and more tag bits per cache block.
- Assuming a cache of 4K blocks, a 4-word block size, and a 32-bit address, find the total number of sets and the total number of tag bits for caches that are direct mapped, two-way, and four-way set associative, and fully associative.
- 4-word block: 16 bytes per block → (32-4)=28 bits
  - 28 bits are divided into index and tag

Size of tags vs. set associativty

- Direct-mapped: 4K blocks → 12-bit index (full index)
  - (28-12)=16-bit tag → 16 x 4K = 64K tag bits
  - A 16-bit comparator
- 2-way set-associative
  - (28-11)=17-bit tag → 17 x 2 x 2K = 68K tag bits
  - Two 17-bit comparators
- 4-way set-associative
  - (28-10)=18-bit tag → 18 x 4 x 1K = 72K tag bits
  - Four 18-bit comparators
- fully set-associative (no index)
  - (28-0)=28-bit tag → (16+12) x 4K x 1 = 112K tag bits
  - 4096 28-bit comparators

Multilevel Caches

- L1 (primary) cache attached to CPU
  - Small, but fast
- L2 cache services misses from primary cache
  - Larger, slower, but still faster than main memory
- Main memory services L2 cache misses
- Some high-end systems include L3 cache

Miss rate: global and local

- Global cache miss:
  - the fraction of references that missed in all cache levels
- Local cache miss:
  - the ratio of all misses in a cache divided by the number of accesses to it.

Multilevel I Cache Example

- Given
  - CPU base CPI = 1, clock rate = 4GHz
  - Miss rate/instruction = 2%
  - Main memory access time = 100ns
- With just primary cache (L1)
  - Miss penalty = 100ns/0.25ns = 400 cycles
  - Effective CPI = 1 + 0.02 × 400 = 9

Using a global miss rate

- Now add L-2 cache
  - Access time = 5ns
  - Global miss rate to main memory = 0.5%
  - Local miss rate = 0.5%/2% = 25%
- L1 miss with L2 hit
  - Penalty = 5ns/0.25ns = 20 cycles
- L1 miss with L2 miss
  - Extra penalty = 400 cycles
- CPI = 1 + 0.02 × 20 + 0.005 × 400 = 3.4
  - Performance improvement with L2 = 9/3.4 = 2.6

Using a local miss rate

- Hit : 75%, Miss: 25% in 2nd level cache
- CPI = 1 + 0.02 * 0.75 x 20

  + 0.02 * 0.25 x (20 + 400) = 3.4

Multilevel Caches

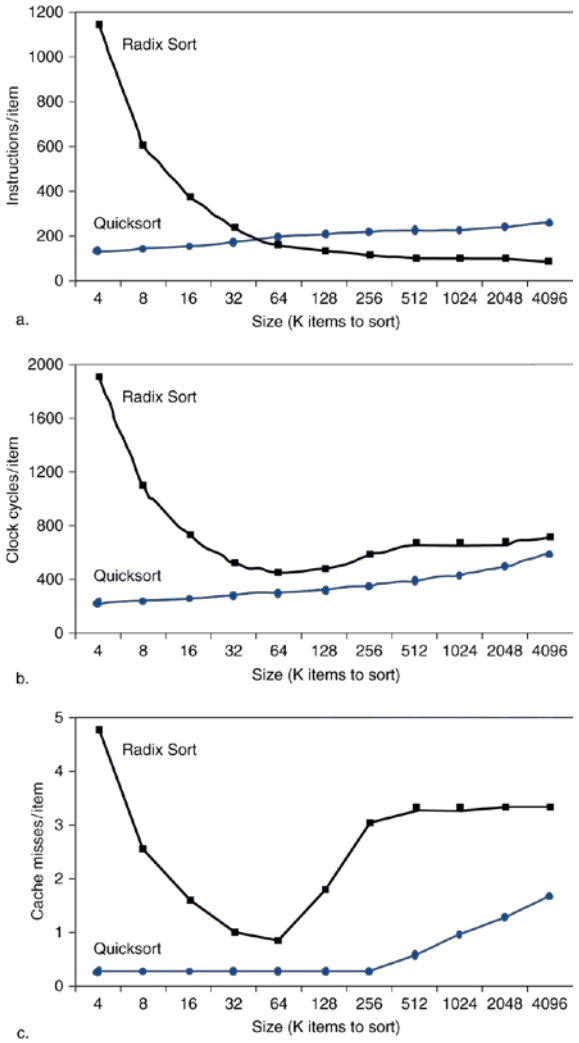
- L1 (primary) cache attached to CPU
  - Smaller, compared to a single-level cache
  - Smaller block size
  - Focus on minimizing hit time to yield a shorter clock cycle or fewer pipeline stages
- L2 cache services misses from L1 cache
  - Much larger than a single-level cache, since its access time is less critical
  - Larger block size
  - Focus on miss rate to reduce the penalty of long memory access times
  - Often uses higher associativity than the primary cache.

Interactions with Advanced CPUs

- Out-of-order CPUs can execute instructions during cache miss
  - Pending store stays in load/store unit
  - Dependent instructions wait in reservation stations
    - Independent instructions continue
- Effect of miss depends on program data flow
  - Much harder to analyze
  - Use system simulation

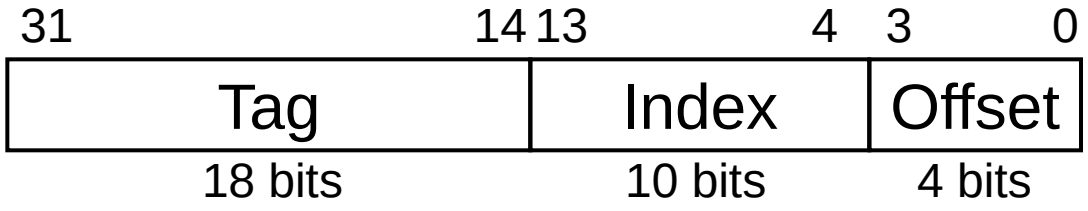
Interactions with Software

- Misses depend on memory access patterns
  - Algorithm behavior
  - Compiler optimization for memory access

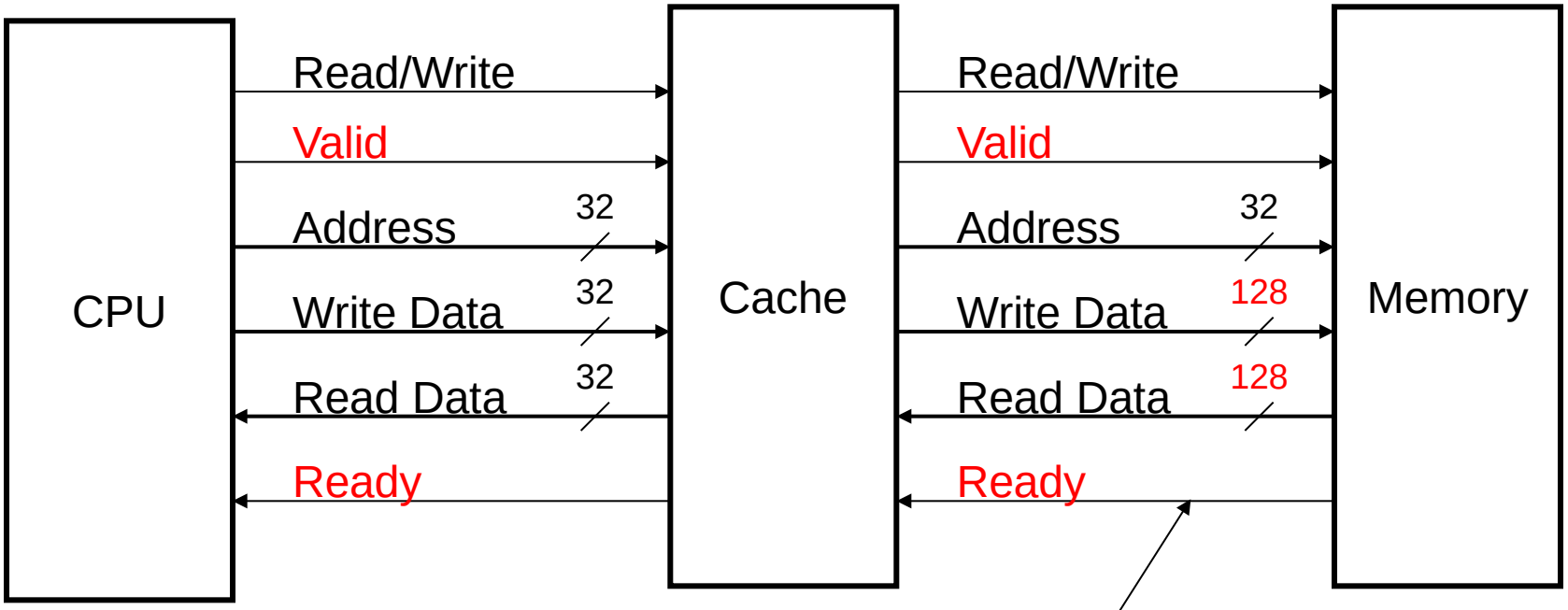


Cache Control

- Example cache characteristics
  - Direct-mapped, write-back, write allocate
  - Block size: 4 words (16 bytes)
  - Cache size: 16 KB (1024 blocks)
  - 32-bit byte addresses
  - Valid bit and dirty bit per block
  - Blocking cache
    - CPU waits until access is complete



Interface Signals



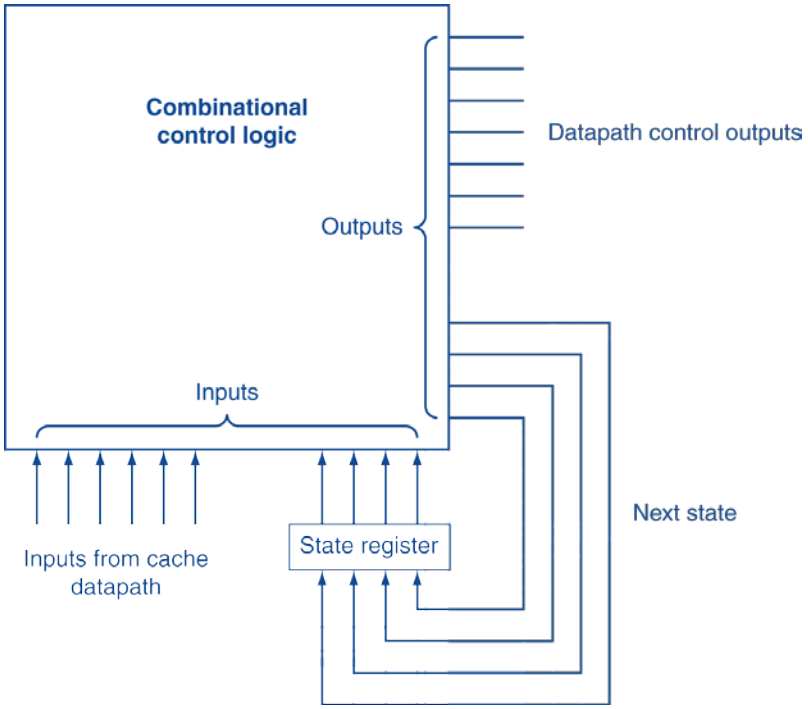
Valid: saying whether there is a cache operation or not
Ready: saying that the cache operation is complete

Multiple cycles per access

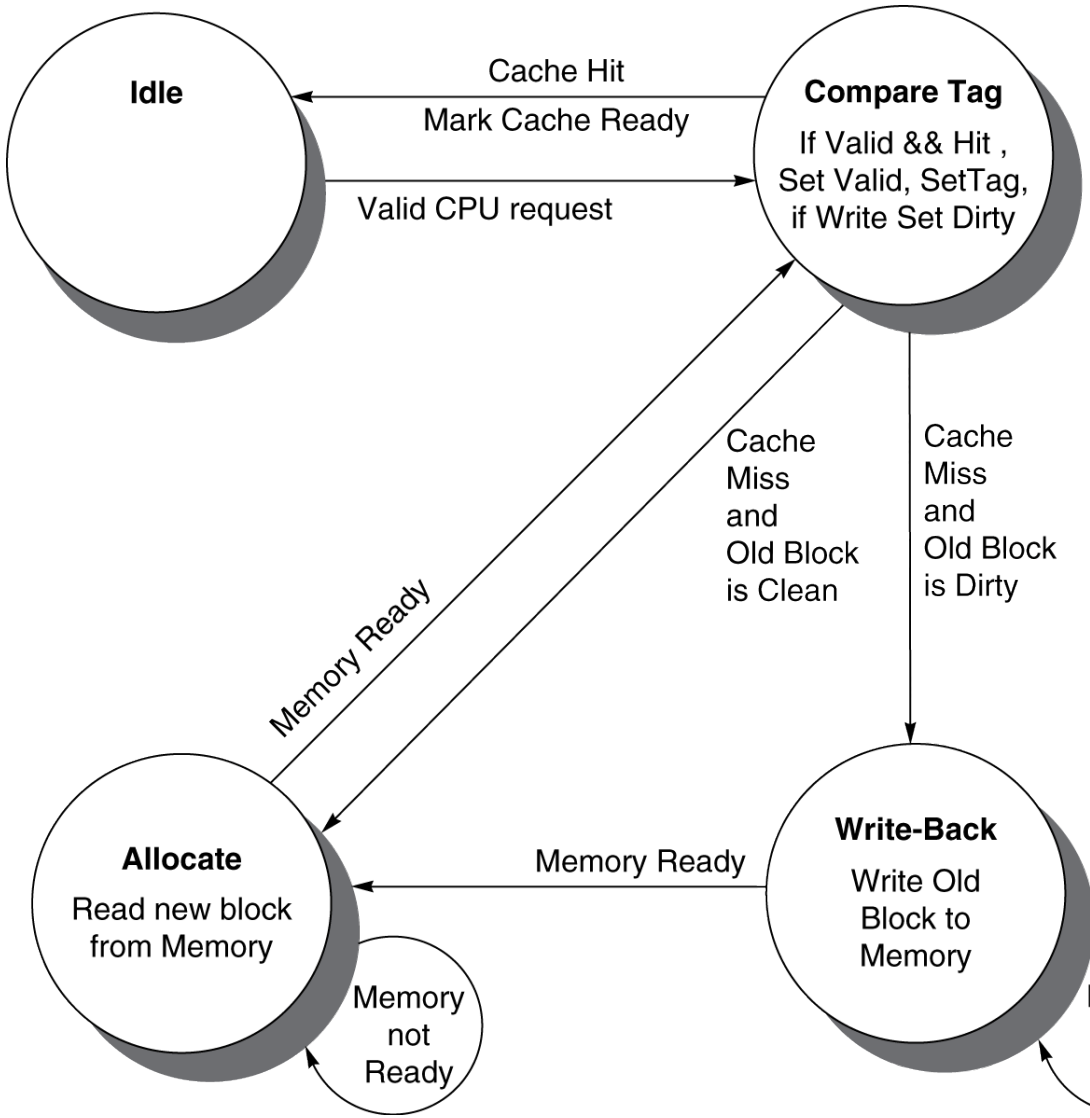
Valid: saying whether there is a memory operation or not
Ready: saying that the memory operation is complete

Cache controller: FSM

- Use an FSM to sequence control steps
- Set of states, transition on each clock edge
  - State values are binary encoded
  - Current state stored in a register
  - Next state = f_n (current state, current inputs)
- Control output signals = f_o (current state)



Cache Controller FSM



Both compare the tag and read/write the data in a cycle

Could partition into separate states to reduce clock cycle time

Homework: chapter 5

- Due before starting the midterm exam on Oct. 27.
- Exercise 5.2
- Exercise 5.3
- Exercise 5.5
- Exercise 5.8
- Exercise 5.12