CENG 3420
Computer Organization and Design

Lecture 08: Memory - I

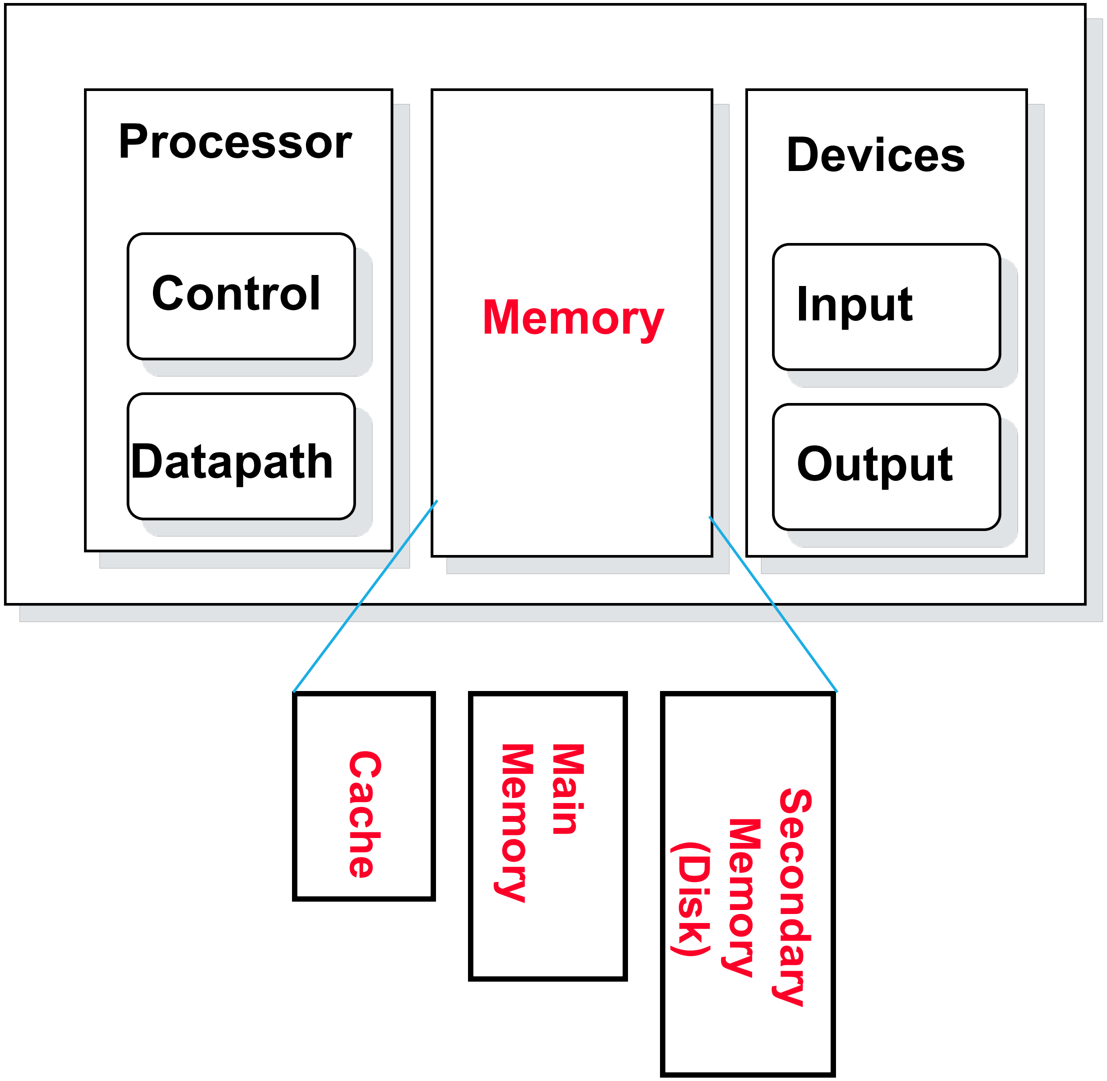
Bei Yu

香港中文大學
The Chinese University of Hong Kong

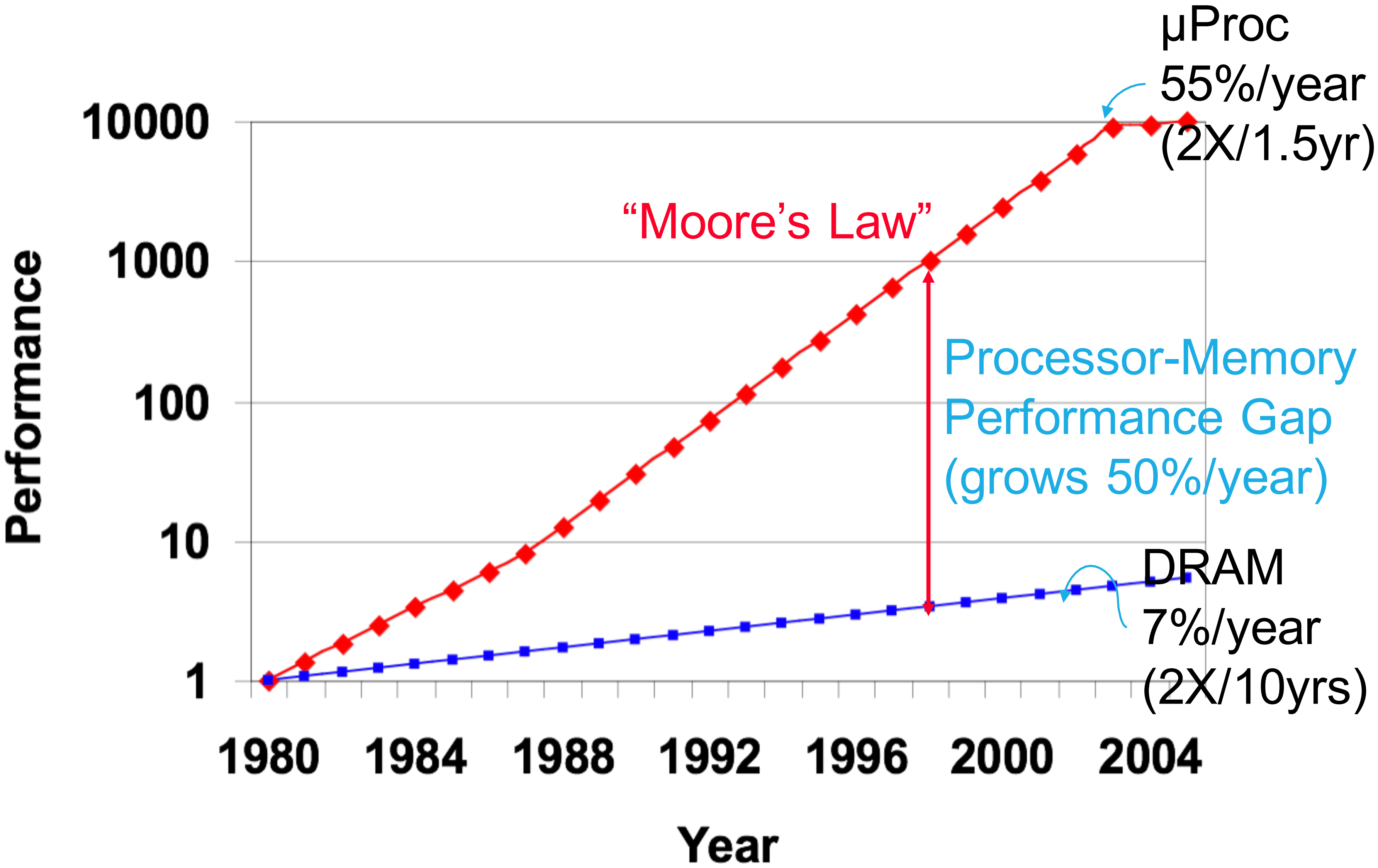
Outline

- ❑ Why Memory Hierarchy
- ❑ How Memory Hierarchy?
  - SRAM (Cache) & DRAM (main memory)
  - Memory System
- ❑ Cache Basics
- ❑ Cache Performance
- ❑ Reduce Cache Miss Rates
- ❑ Summary

Review: Major Components of a Computer



Processor-Memory Performance Gap



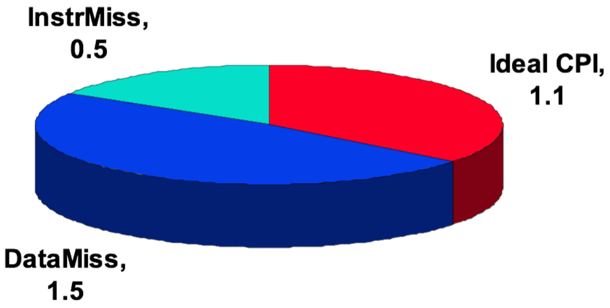
µProc
55%/year
(2X/1.5yr)

"Moore's Law"

Processor-Memory
Performance Gap
(grows 50%/year)

DRAM
7%/year
(2X/10yrs)

Memory Performance Impact on Performance



- Suppose a processor executes at
  - ideal CPI = 1.1
  - 50% arith/logic, 30% ld/st, 20% control

  and that 10% of data memory operations miss with a 50 cycle miss penalty

- EX: calculate practical CPI:
- CPI = ideal CPI + average stalls per instruction

Memory Hierarchy

- Fact: Large memories are slow and fast memories are small

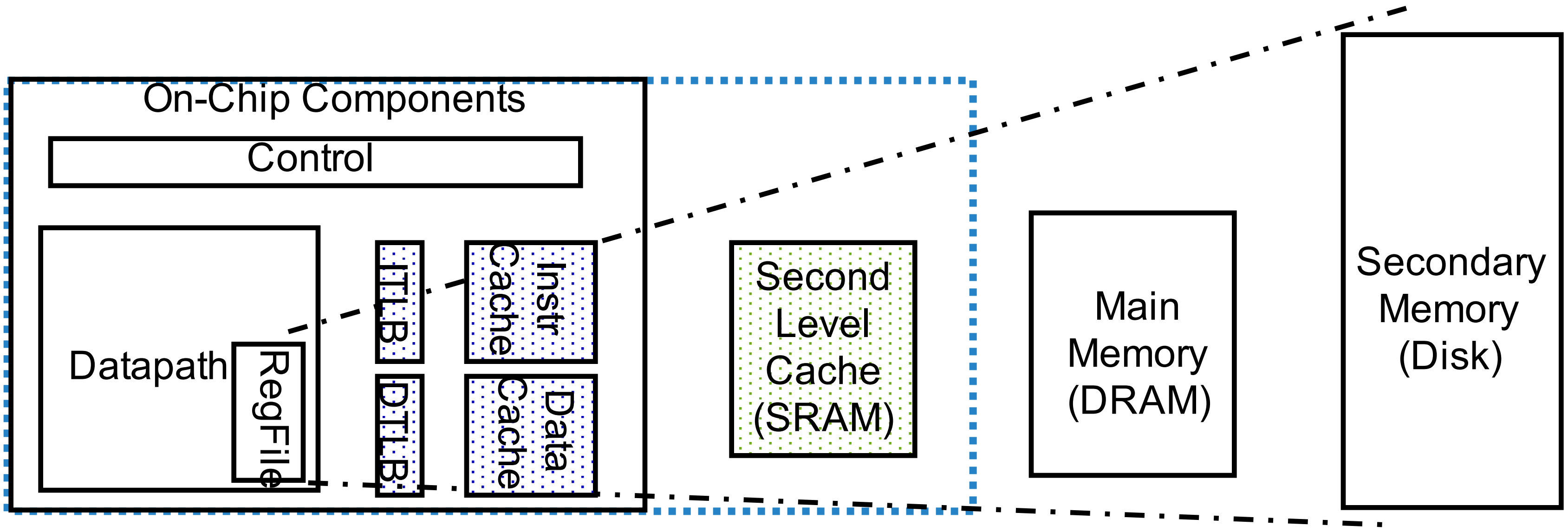
- How do we create a memory that gives the illusion of being large, cheap and fast (most of the time)?
  - With hierarchy
  - With parallelism

Outline

- ❑ Why Memory Hierarchy
- ❑ How Memory Hierarchy?
  - ● SRAM (Cache) & DRAM (main memory)
  - ● Memory System
- ❑ Cache Basics
- ❑ Cache Performance
- ❑ Reduce Cache Miss Rates
- ❑ Summary

A Typical Memory Hierarchy

- Take advantage of the principle of locality to present the user with as much memory as is available in the *cheapest* technology at the speed offered by the *fastest* technology



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

The Memory Hierarchy: Why Does it Work?

- Temporal Locality (locality in time)
  - If a memory location is referenced then it will tend to be referenced again soon

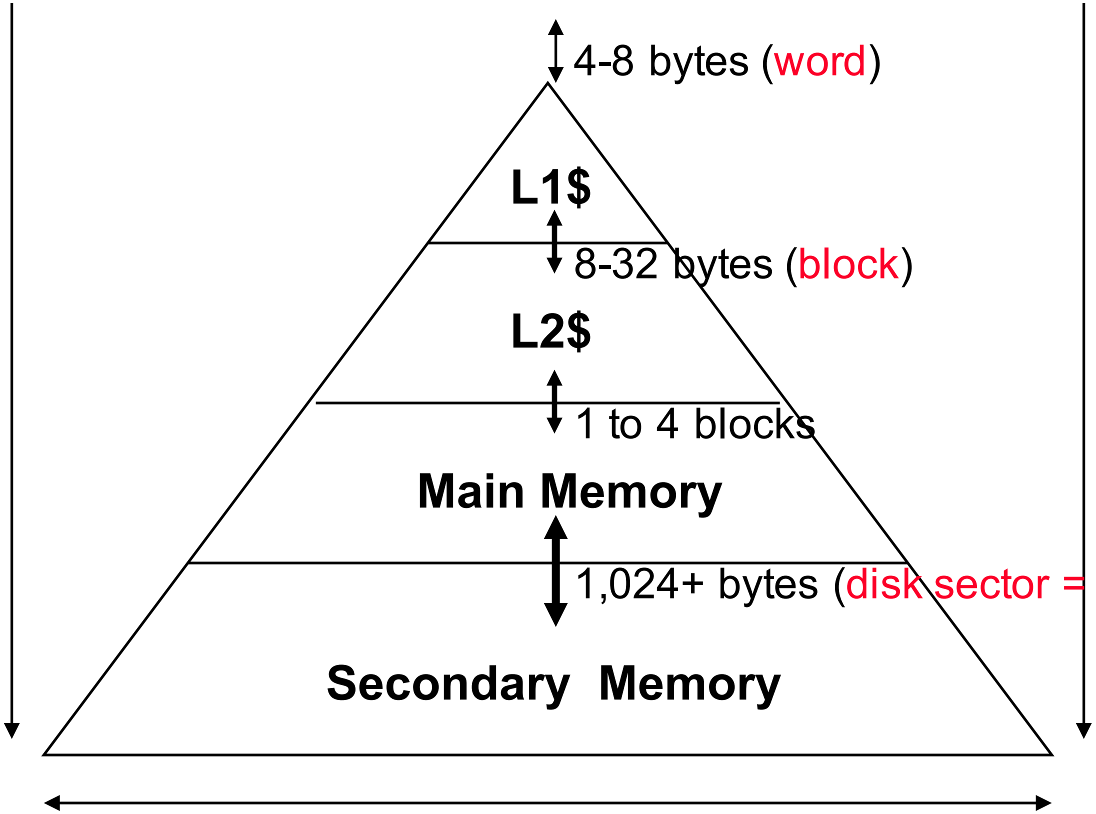
  ⇒ Keep most recently accessed data items closer to the processor

- Spatial Locality (locality in space)
  - If a memory location is referenced, the locations with nearby addresses will tend to be referenced soon

  ⇒ Move blocks consisting of contiguous words closer to the processor

Characteristics of the Memory Hierarchy

Increasing distance from the processor in access time



Processor

4-8 bytes (word)

L1$

8-32 bytes (block)

L2$

1 to 4 blocks

Main Memory

1,024+ bytes (disk sector = page)

Secondary Memory

(Relative) size of the memory at each level

Inclusive– what is in L1$ is a subset of what is in L2$ is a subset of what is in MM that is a subset of is in SM

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

How is the Hierarchy Managed?

- registers ↔ memory
  - by compiler (programmer?)
- cache ↔ main memory
  - by the cache controller hardware
- main memory ↔ disks
  - by the operating system (virtual memory)
  - virtual to physical address mapping assisted by the hardware (TLB)
  - by the programmer (files)

Outline

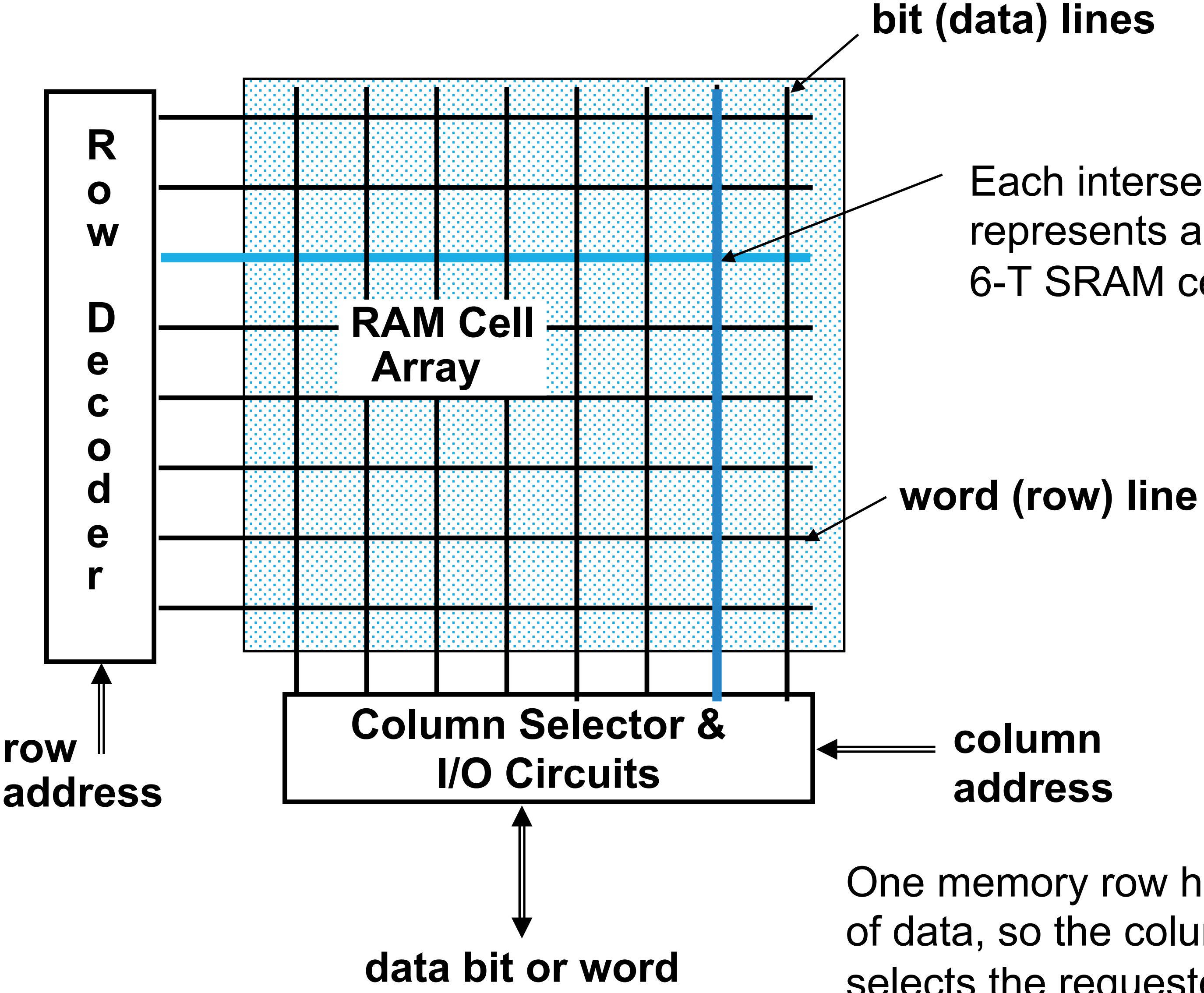
- Why Memory Hierarchy
- How Memory Hierarchy?
  - SRAM (Cache) & DRAM (main memory)
  - Memory System
- Cache Basics
- Cache Performance
- Reduce Cache Miss Rates
- Summary

Memory Hierarchy Technologies

- Caches use *SRAM* for speed and technology compatibility
  - Fast (typical access times of 0.5 to 2.5 nsec)
  - Low density (6 transistor cells), higher power, expensive ($2000 to $5000 per GB in 2008)
  - Static: content will last "forever" (as long as power is left on)

- Main memory uses *DRAM* for size (density)
  - High density (1 transistor cells), lower power, cheaper ($20 to $75 per GB in 2008)
  - Dynamic: needs to be "refreshed" regularly (~ every 8 ms)
    - consumes1% to 2% of the active cycles of the DRAM
  - Addresses divided into 2 halves (row and column)
    - *RAS* or *Row Access Strobe* triggering the row decoder
    - *CAS* or *Column Access Strobe* triggering the column selector

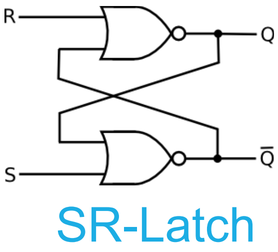
Classical SRAM Organization

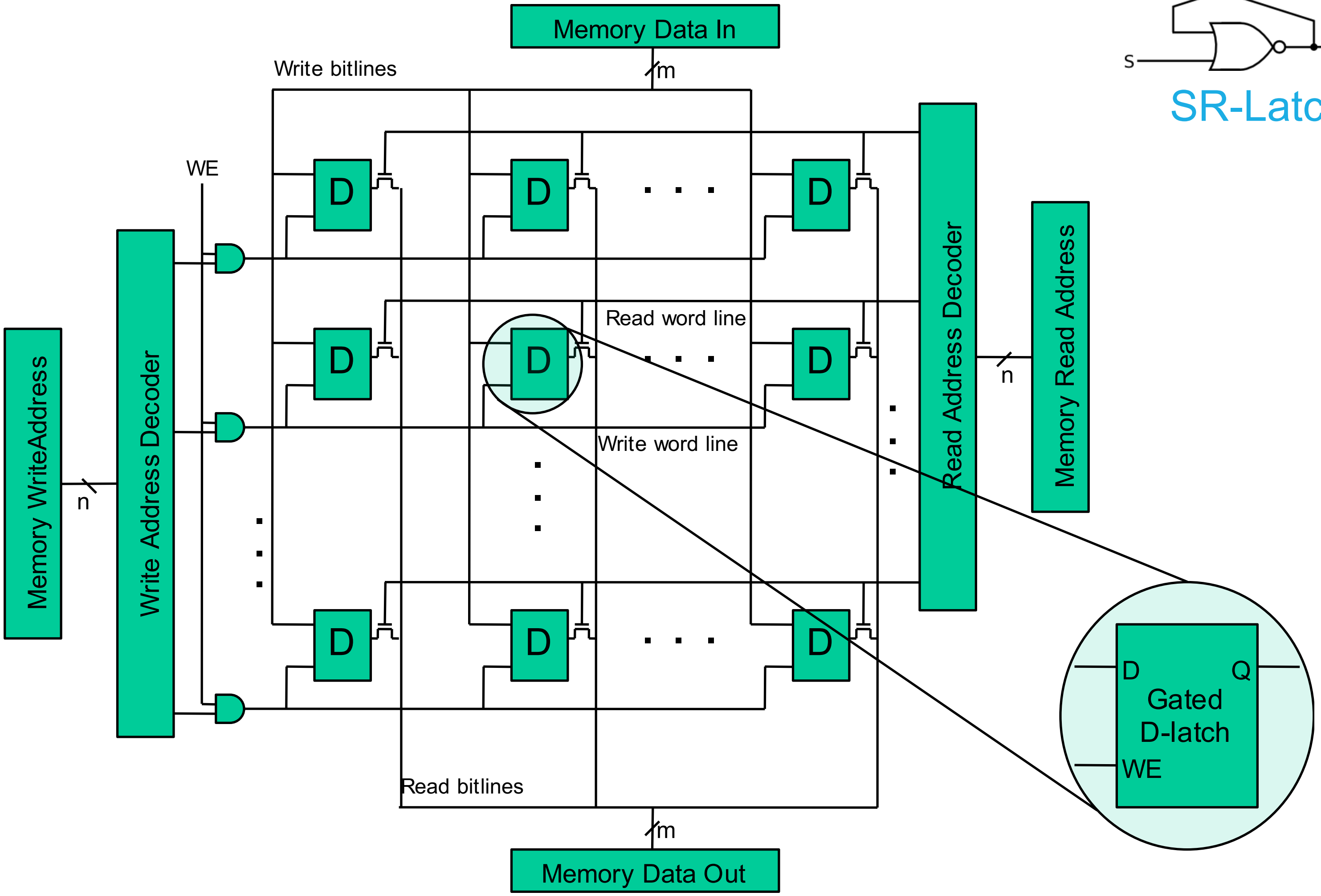


One memory row holds a block of data, so the column address selects the requested bit or word from that block

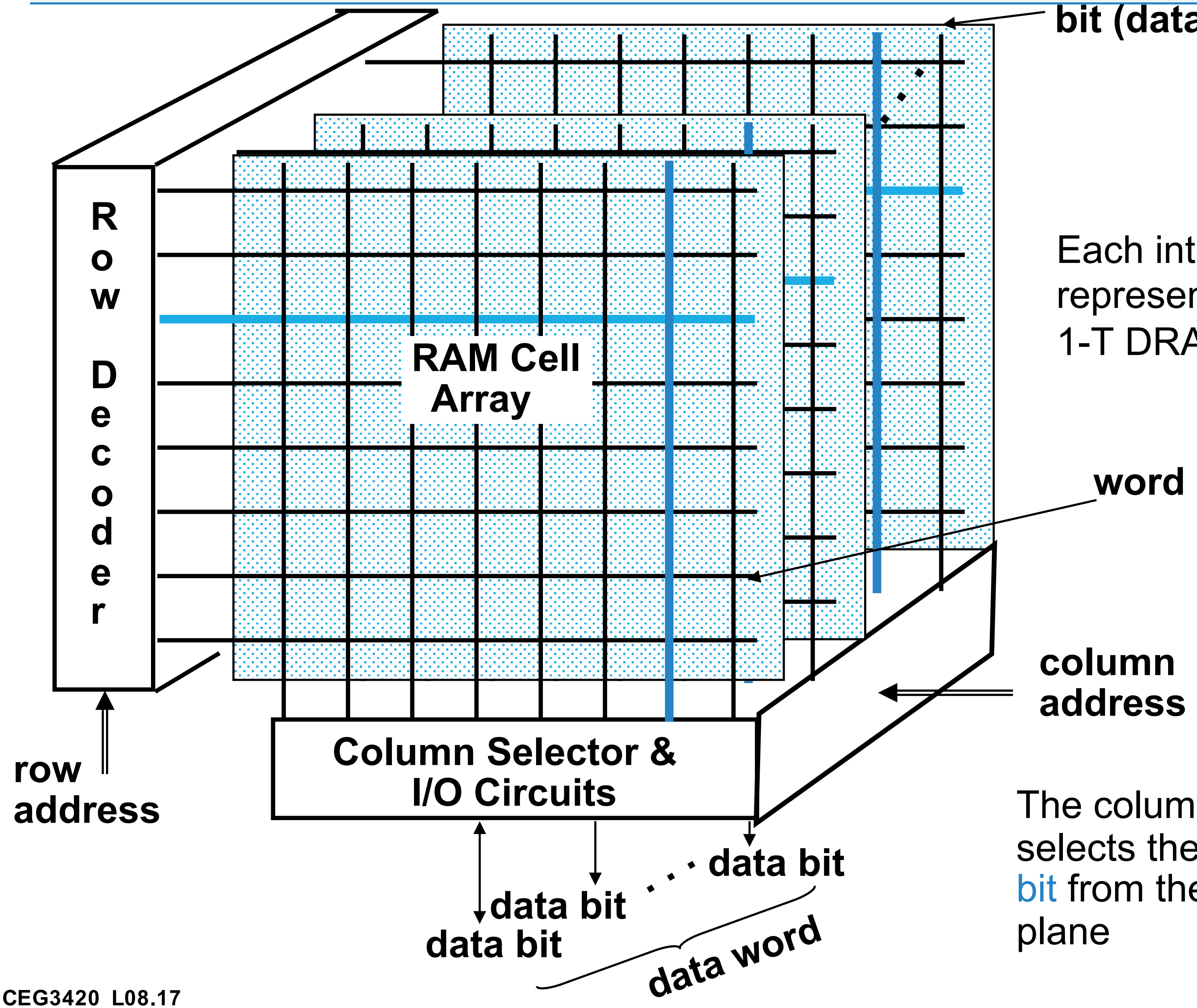
Classical SRAM Organization

- Latch based memory





Classical DRAM Organization

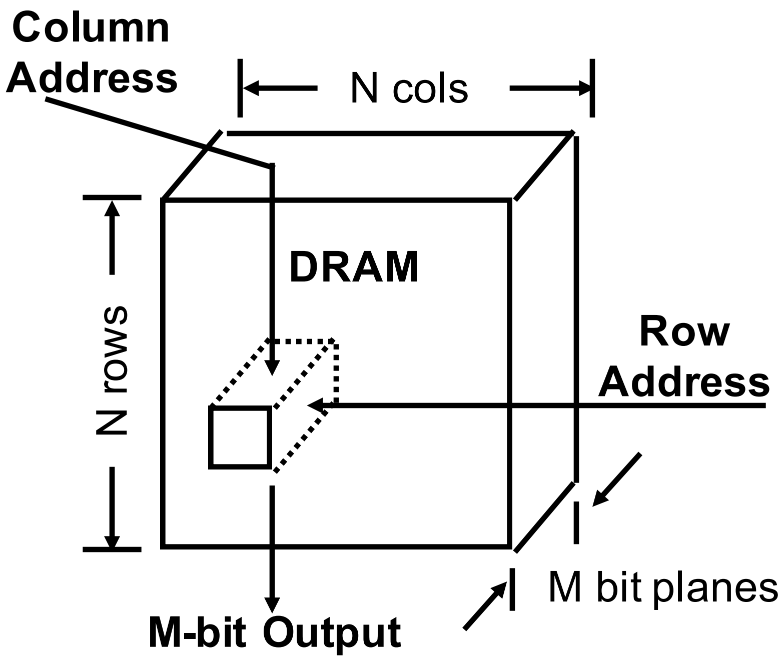


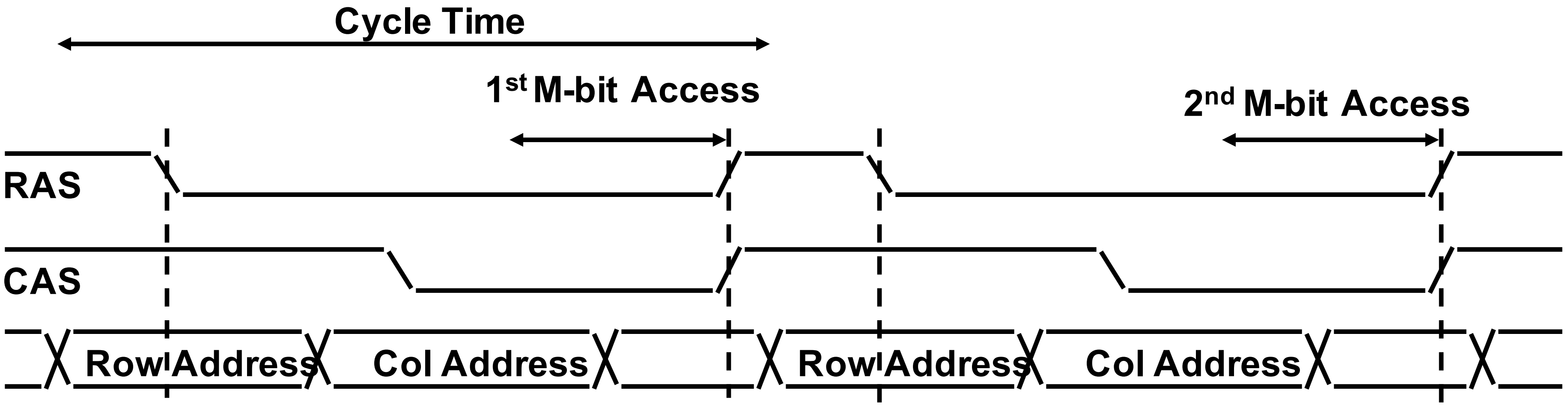
Each intersection represents a 1-T DRAM cell

The column address selects the requested bit from the row in each plane

Classical DRAM Operation

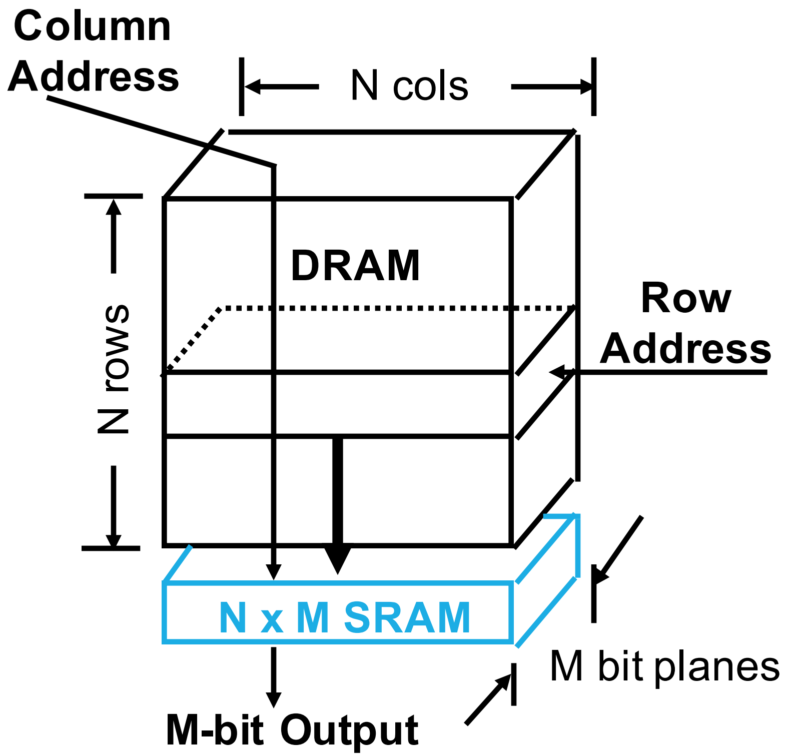
- DRAM Organization:
  - N rows x N column x M-bit
  - Read or Write M-bit at a time
  - Each M-bit access requires a RAS / CAS cycle

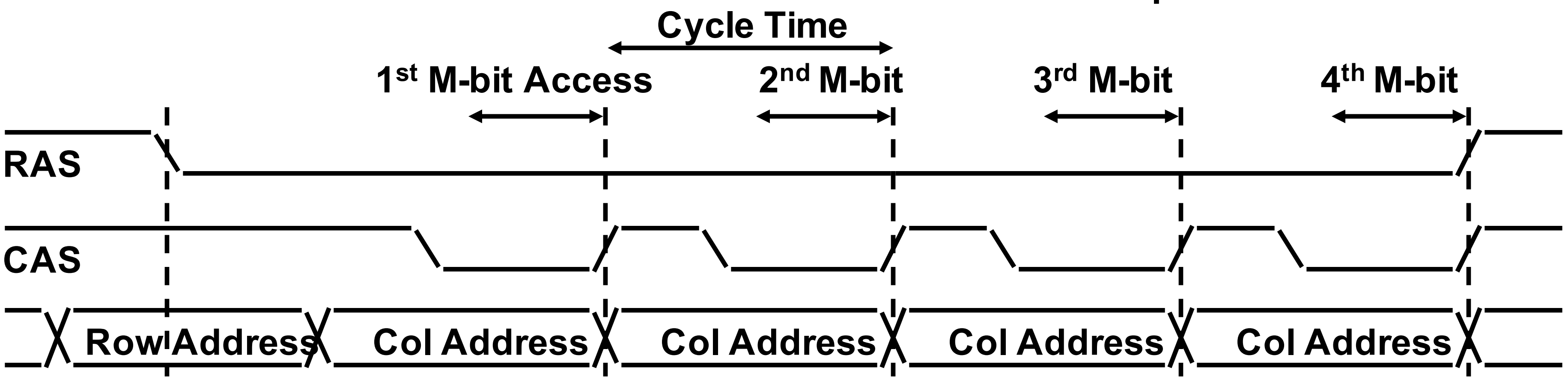




Page Mode DRAM Operation

- Page Mode DRAM
  - N x M SRAM to save a row
- After a row is read into the SRAM "register"
  - Only CAS is needed to access other M-bit words on that row
  - RAS remains asserted while CAS is toggled





DRAM Latency & Bandwidth Milestones (optional)

	DRAM	Page DRAM	FastPage DRAM	FastPage DRAM	Synch DRAM	DDR SDRAM
Module Width	16b	16b	32b	64b	64b	64b
Year	1980	1983	1986	1993	1997	2000
Mb/chip	0.06	0.25	1	16	64	256
Die size (mm^2)	35	45	70	130	170	204
Pins/chip	16	16	18	20	54	66
BWidth (MB/s)	13	40	160	267	640	1600
Latency (nsec)	225	170	125	75	62	52

Patterson, CACM Vol 47, #10, 2004

- In the time that the memory to processor bandwidth doubles the memory latency improves by a factor of only 1.2 to 1.4

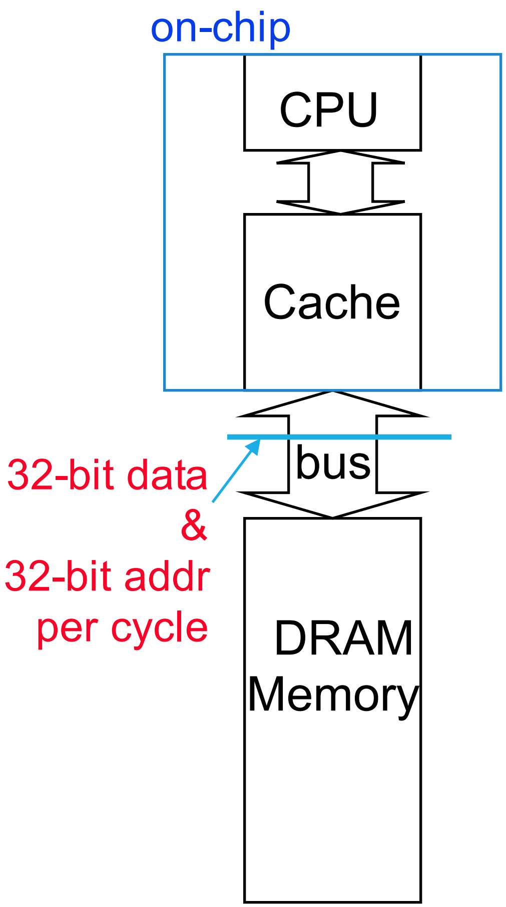
- To deliver such high bandwidth, the internal DRAM has to be organized as interleaved memory banks

Outline

- ❑ Why Memory Hierarchy
- ❑ How Memory Hierarchy?
  - ● SRAM (Cache) & DRAM (main memory)
  - ● Memory System
- ❑ Cache Basics
- ❑ Cache Performance
- ❑ Reduce Cache Miss Rates
- ❑ Summary

Memory Systems that Support Caches

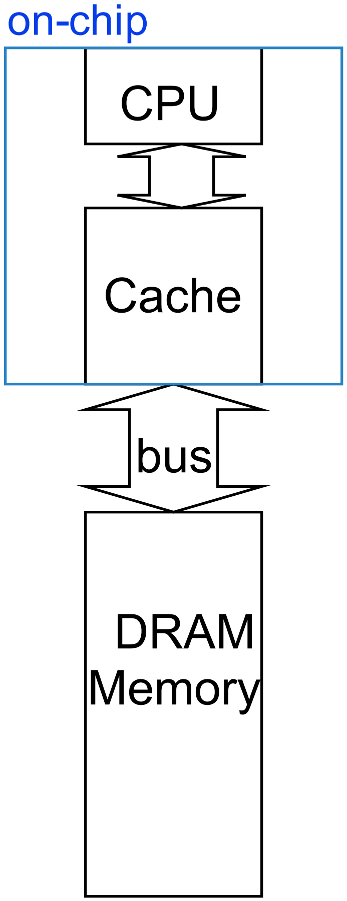
- The off-chip interconnect and memory architecture can affect overall system performance in dramatic ways



One word wide organization (one word wide bus and one word wide memory)

- Assume
  1. 1 clock cycle to send the addr
  2. 15 clock cycles to get the 1st word in the block from DRAM, 5 clock cycles for 2nd, 3rd, 4th words (column access time)
  3. 1 clock cycle to return a word of data
- Memory-Bus to Cache bandwidth
  - number of bytes accessed from memory and transferred to cache/CPU per clock cycle

One Word Wide Bus, One Word Blocks



- If the block size is one word, then for a memory access due to a cache miss, the pipeline will have to stall for the number of cycles required to return one data word from memory

  cycle to send address

  cycles to read DRAM

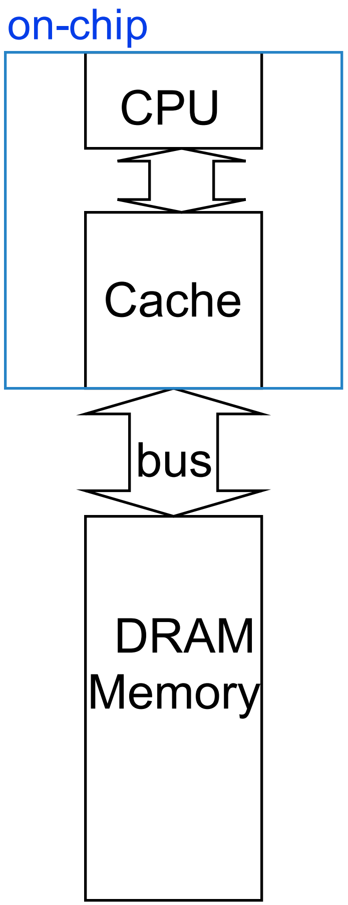
  cycle to return data

  —— total clock cycles miss penalty

- Number of bytes transferred per clock cycle (bandwidth) for a single miss is

  bytes per memory bus clock cycle

One Word Wide Bus, One Word Blocks



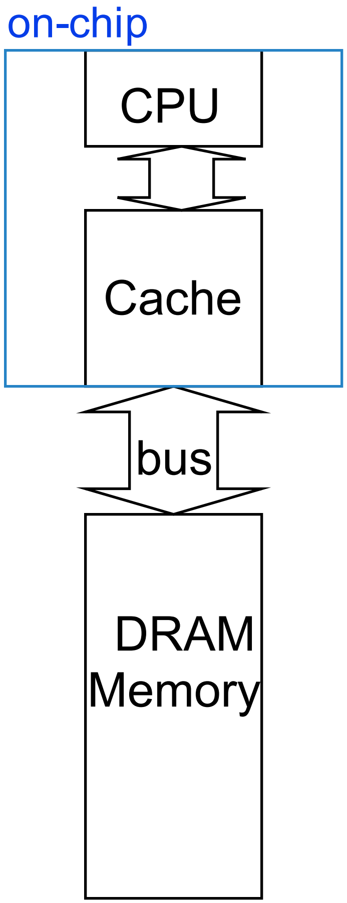
- If the block size is one word, then for a memory access due to a cache miss, the pipeline will have to stall for the number of cycles required to return one data word from memory

1	cycle to send address
15	cycles to read DRAM
1	cycle to return data
17	total clock cycles miss penalty

- Number of bytes transferred per clock cycle (bandwidth) for a single miss is

  4/17 = 0.235 bytes per memory bus clock cycle

One Word Wide Bus, Four Word Blocks



- What if the block size is four words and each word is in a different DRAM row?

cycle to send 1st address

cycles to read DRAM

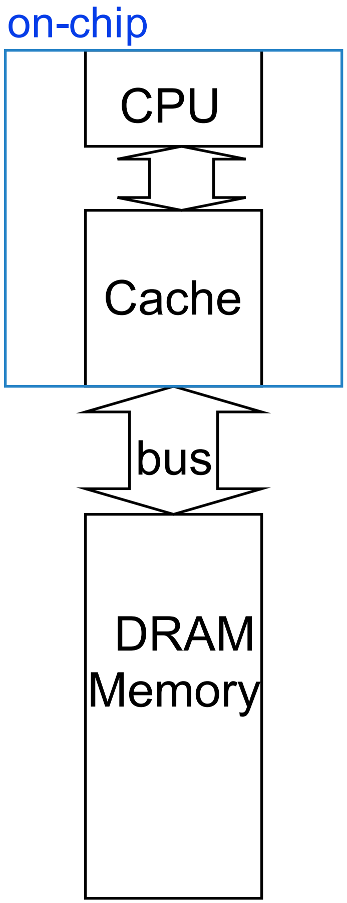
cycles to return last data word

total clock cycles miss penalty

- Number of bytes transferred per clock cycle (bandwidth) for a single miss is

bytes per clock

One Word Wide Bus, Four Word Blocks



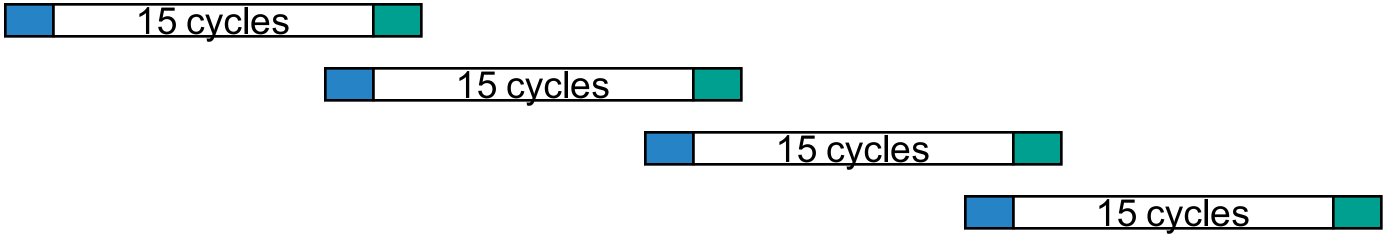
- What if the block size is four words and each word is in a different DRAM row?

1 cycle to send 1st address

4 x 15 = 60 cycles to read DRAM

1 cycles to return last data word

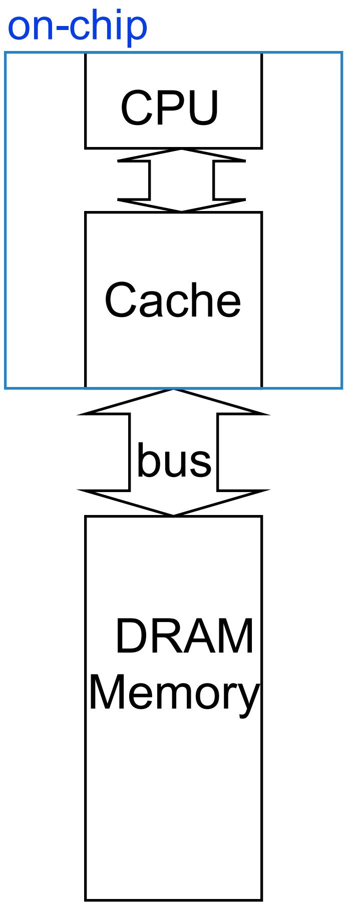
62 total clock cycles miss penalty



- Number of bytes transferred per clock cycle (bandwidth) for a single miss is

(4 x 4)/62 = 0.258 bytes per clock

One Word Wide Bus, Four Word Blocks



- What if the block size is four words and all words are in the same DRAM row?

cycle to send 1st address

cycles to read DRAM

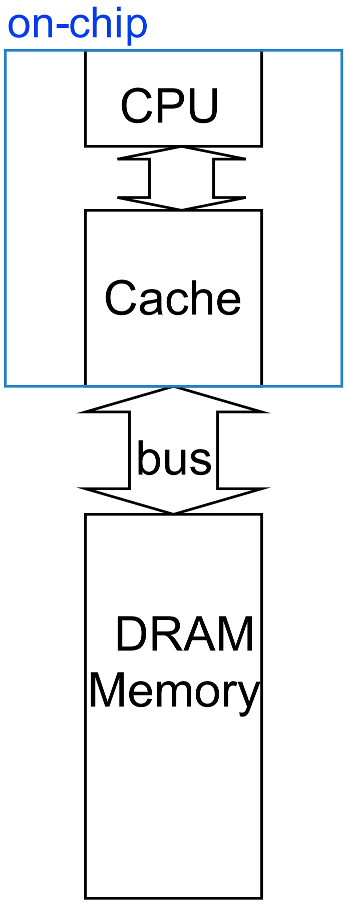
cycles to return last data word

total clock cycles miss penalty

- Number of bytes transferred per clock cycle (bandwidth) for a single miss is

bytes per clock

One Word Wide Bus, Four Word Blocks



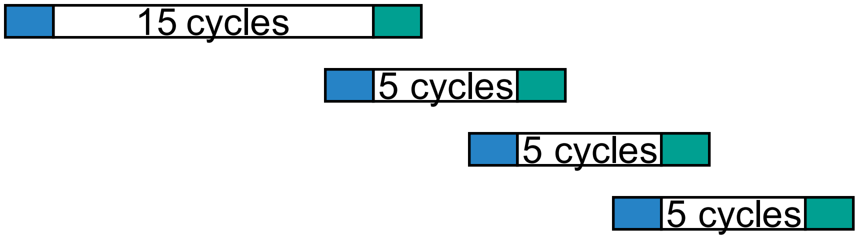
- What if the block size is four words and all words are in the same DRAM row?

1 cycle to send 1st address

15 + 3*5 = 30 cycles to read DRAM

1 cycles to return last data word

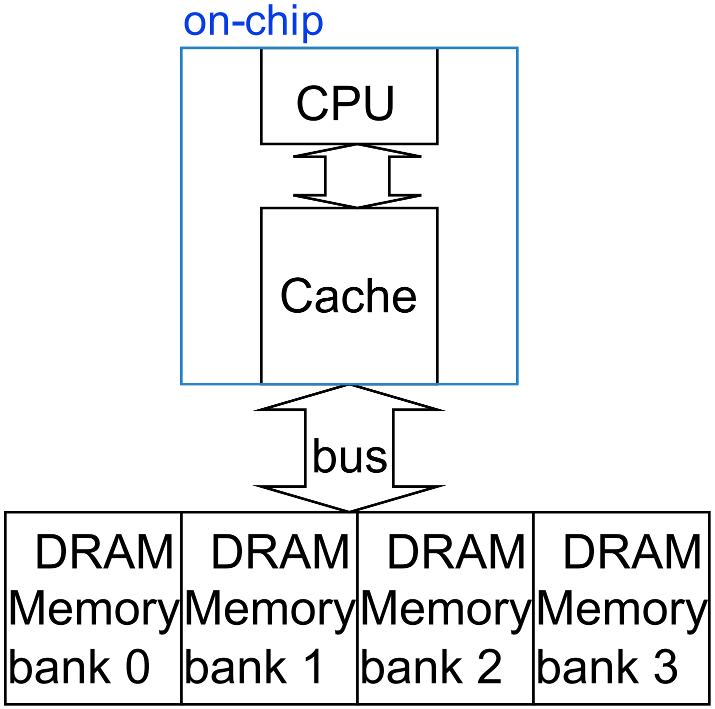
32 total clock cycles miss penalty



- Number of bytes transferred per clock cycle (bandwidth) for a single miss is

(4 x 4)/32 = 0.5 bytes per clock

Interleaved Memory, One Word Wide Bus



- For a block size of four words

cycle to send 1st address

cycles to read DRAM banks

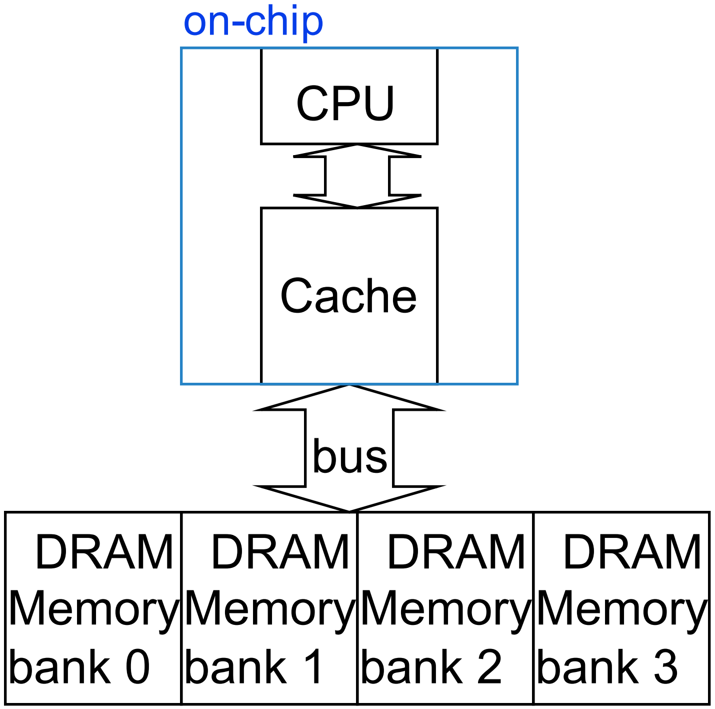
____ cycles to return last data word

total clock cycles miss penalty

- Number of bytes transferred per clock cycle (bandwidth) for a single miss is

bytes per clock

Interleaved Memory, One Word Wide Bus



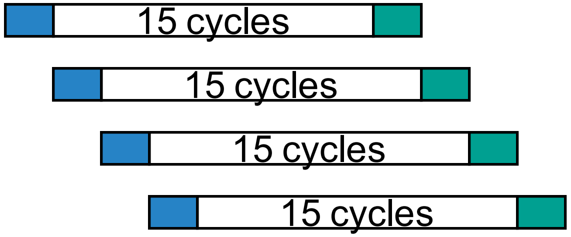
- For a block size of four words

1 cycle to send 1st address

15 + 3 = 18 cycles to read DRAM banks

1 cycles to return last data word

20 total clock cycles miss penalty



- Number of bytes transferred per clock cycle (bandwidth) for a single miss is

(4 x 4)/20 = 0.8 bytes per clock

Memory System Summary

- Its important to match the cache characteristics
  - caches access one block at a time (usually more than one word)
- with the DRAM characteristics
  - use DRAMs that support fast multiple word accesses, preferably ones that match the block size of the cache
- with the memory-bus characteristics
  - make sure the memory-bus can support the DRAM access rates and patterns
  - with the goal of increasing the Memory-Bus to Cache bandwidth

Outline

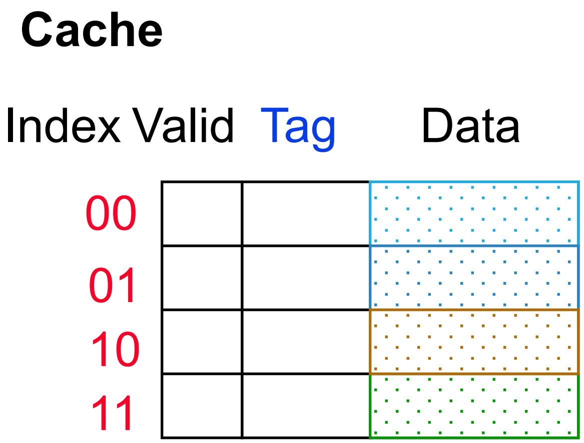
- Why Memory Hierarchy
- How Memory Hierarchy?
  - SRAM (Cache) & DRAM (main memory)
  - Memory System
- Cache Basics
- Cache Performance
- Reduce Cache Miss Rates
- Summary

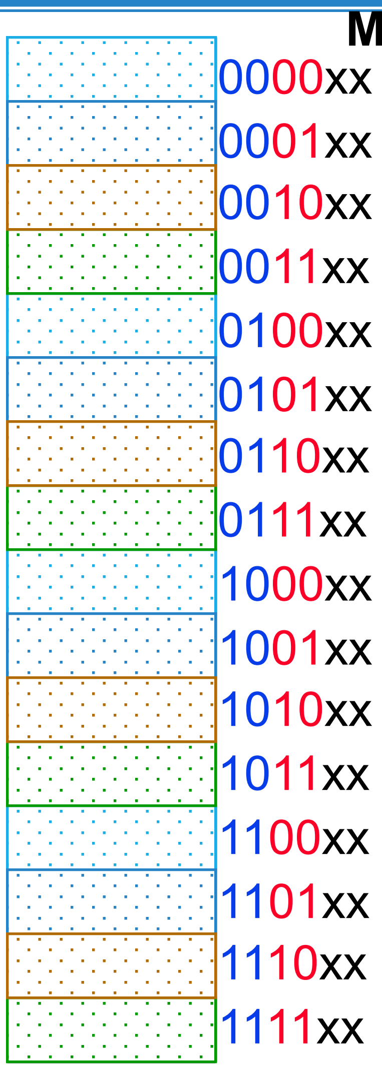
Cache Basics

- Two questions to answer (in hardware):
  - Q1: How do we know if a data item is in the cache?
  - Q2: If it is, how do we find it?

- Direct mapped
  - Each memory block is mapped to exactly one block in the cache
    - lots of lower level blocks must share blocks in the cache
  - Address mapping (to answer Q2):
  - Have a tag associated with each cache block that contains the address information (the upper portion of the address) required to identify the block (to answer Q1)

Caching: A Simple First Example





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

Q1: Is it there?

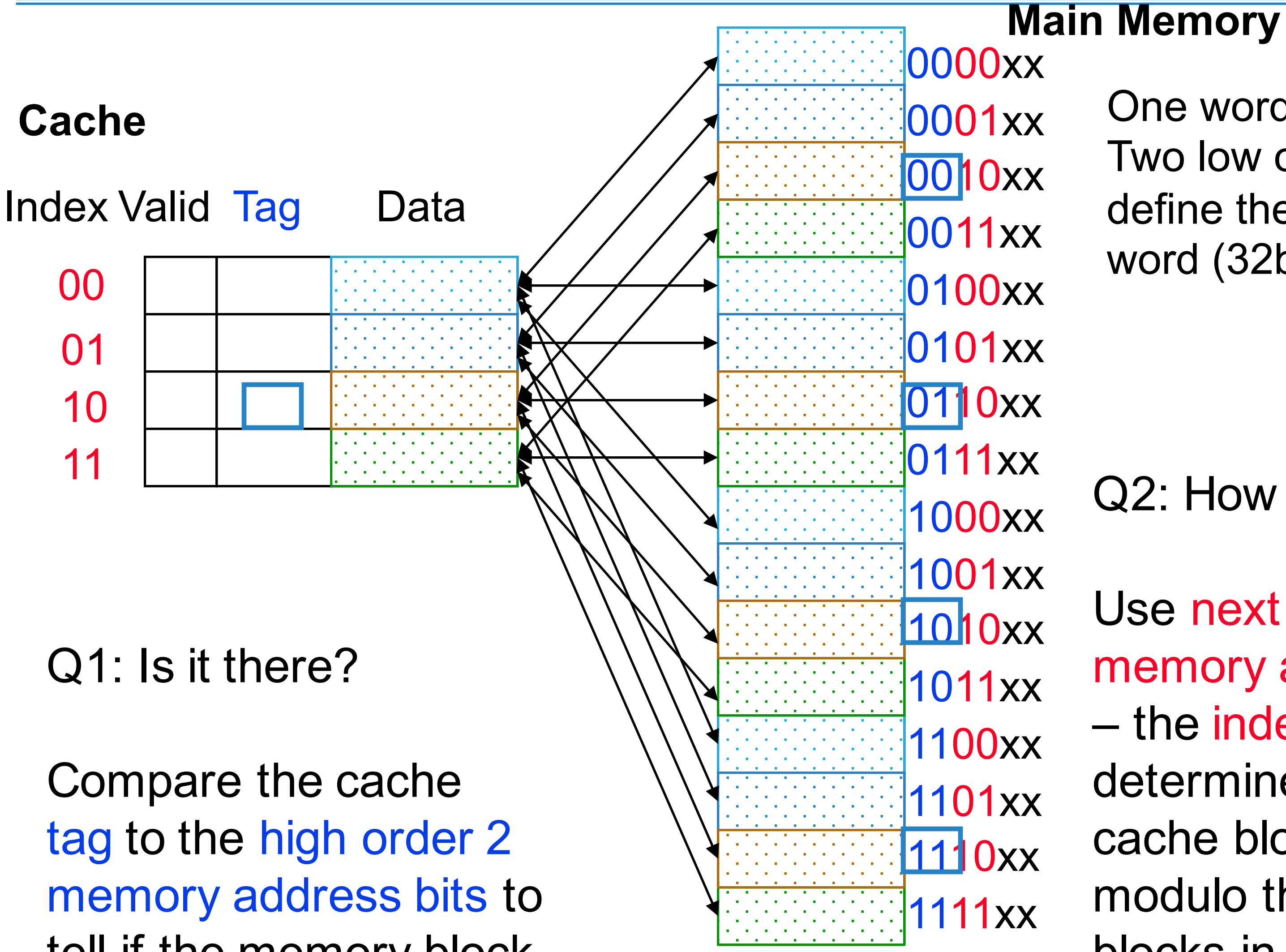
Compare the cache tag to the high order 2 memory address bits to tell if the memory block is in the cache

Q2: How do we find it?

Use next 2 low order memory address bits – the index – to determine which cache block (i.e., modulo the number of blocks in the cache)

(block address) modulo (# of blocks in the cache)

Caching: A Simple First Example



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

Q2: How do we find it?

Use next 2 low order memory address bits – the index – to determine which cache block (i.e., modulo the number of blocks in the cache)

Q1: Is it there?

Compare the cache tag to the high order 2 memory address bits to tell if the memory block is in the cache

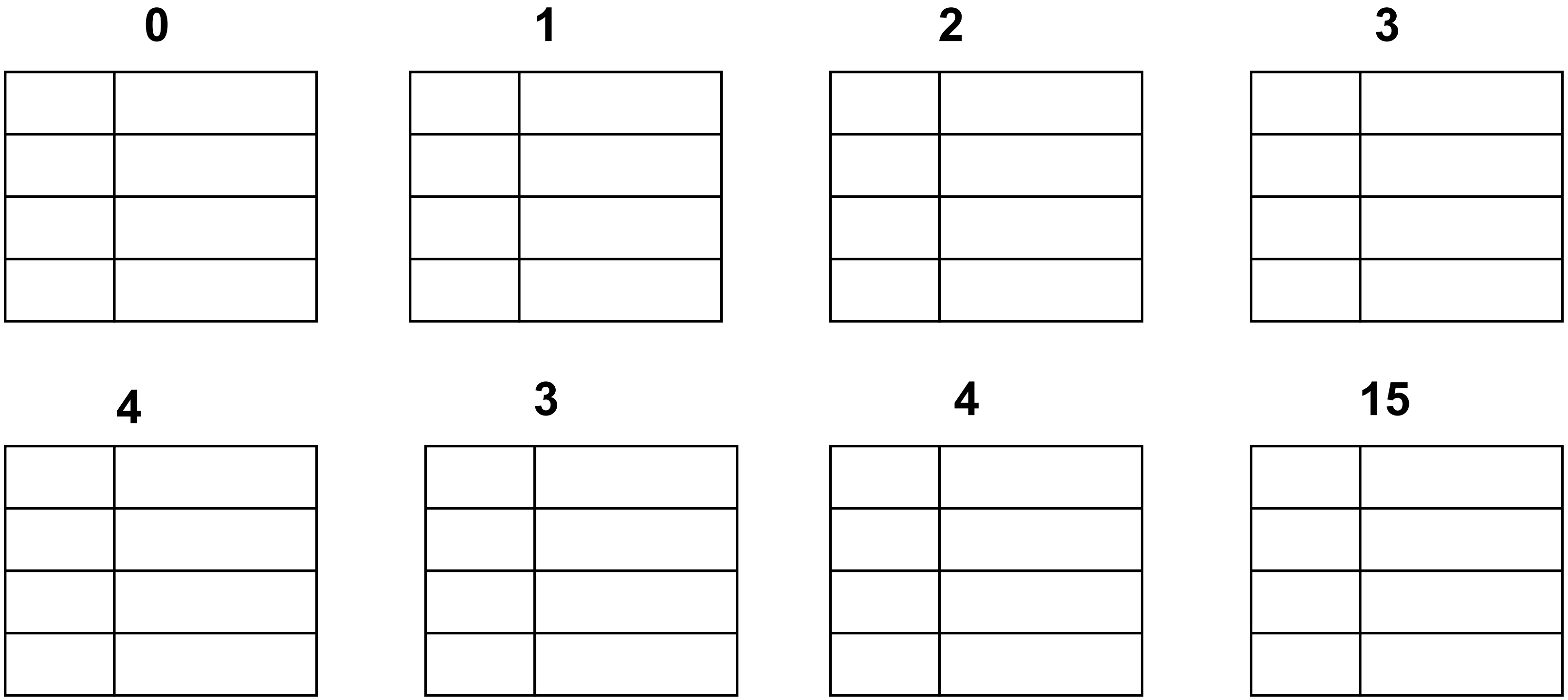
(block address) modulo (# of blocks in the cache)

Direct Mapped Cache

- ❑ Consider the main memory word reference string

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

0 1 2 3 4 3 4 15



Direct Mapped Cache

- ❑ Consider the main memory word reference string

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

0 1 2 3 4 3 4 15

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)

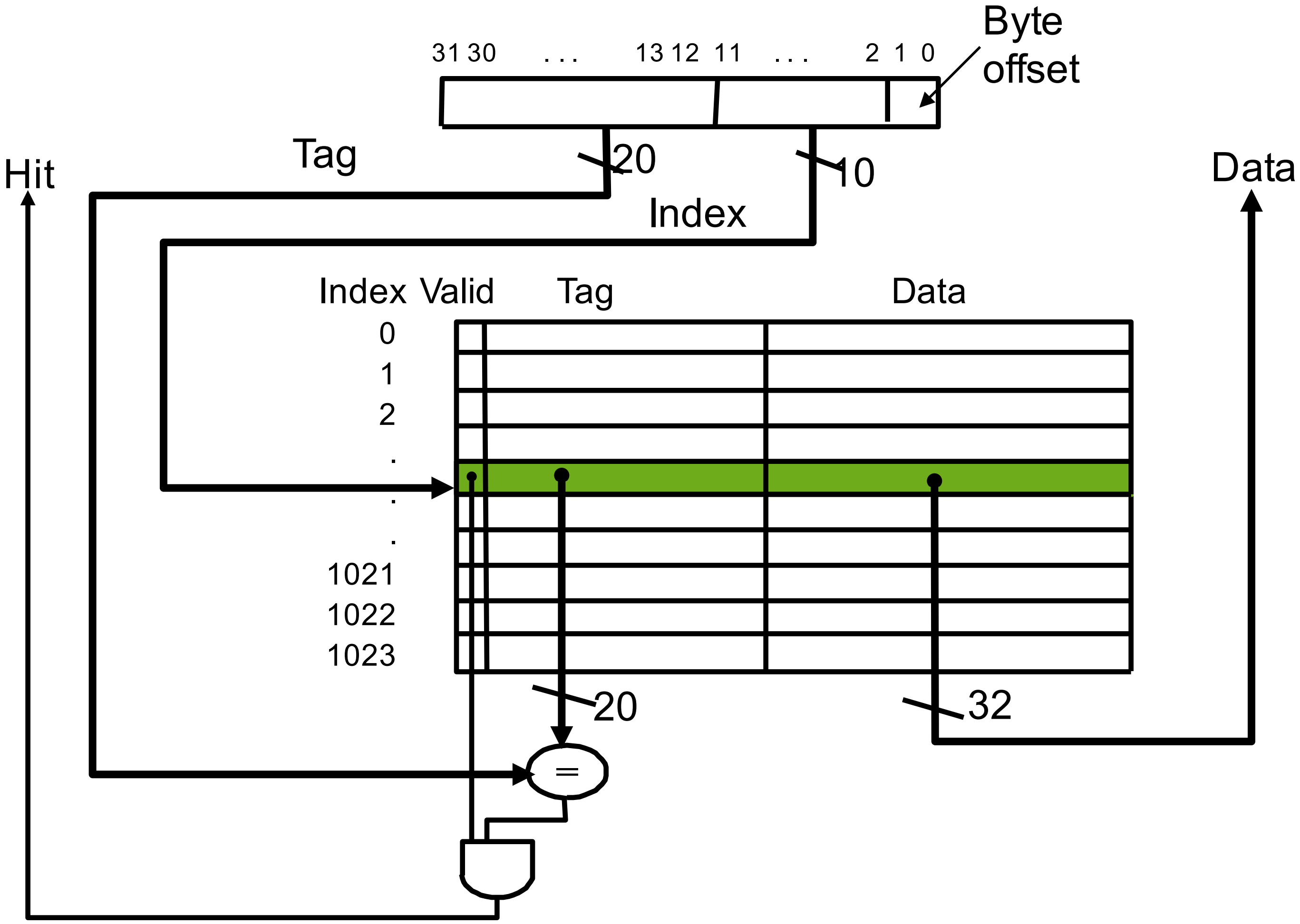
15 miss

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

- 8 requests, 6 misses

MIPS Direct Mapped Cache Example

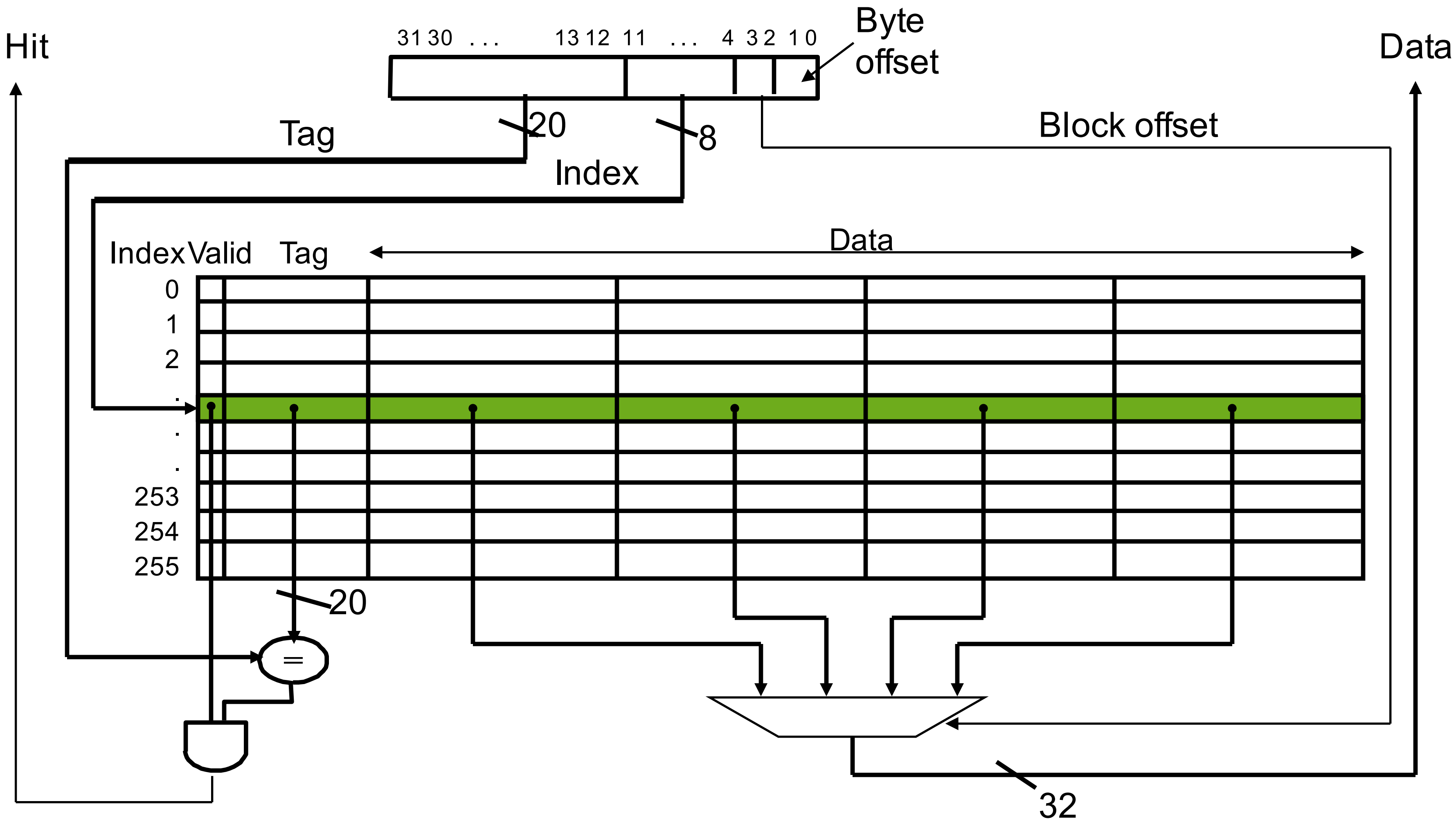
- One word blocks, cache size = 1K words (or 4KB)



What kind of locality are we taking advantage of?

Multiword Block Direct Mapped Cache

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



What kind of locality are we taking advantage of?

Taking Advantage of Spatial Locality

- Let cache block hold more than one word

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

0 1 2 3 4 3 4 15

0

1

2

3

4

3

4

15

Taking Advantage of Spatial Locality

- Let cache block hold more than one word

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

0 1 2 3 4 3 4 15

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

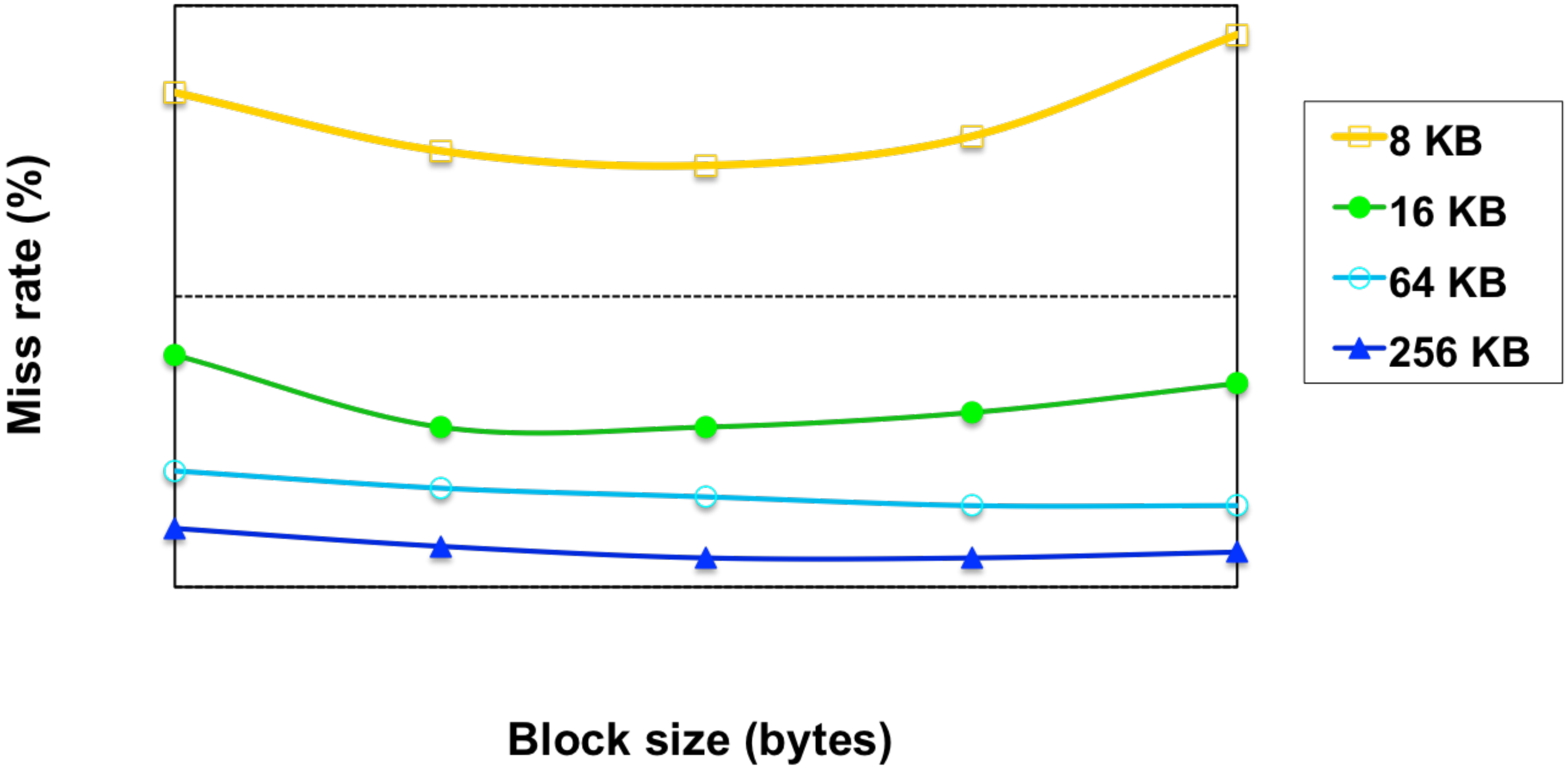
01	Mem(5)	Mem(4)
00	Mem(3)	Mem(2)

15 miss

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

- 8 requests, 4 misses

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 (increasing capacity misses)

Cache Field Sizes

- The number of bits in a cache includes both the storage for data and for the tags
  - 32-bit byte address
  - For a direct mapped cache with 2^n blocks, *n* bits are used for the index
  - For a block size of 2^m words (2^{m+2} bytes), *m* bits are used to address the word within the block and 2 bits are used to address the byte within the word
- What is the size of the tag field?

$$32 – (n + m +2)$$

- The total number of bits in a direct-mapped cache is then

$$2^n \text{ x (block size + tag field size + valid field size)}$$

EX: Bits in a Cache

- How many total bits are required for a direct mapped cache with 16KB of data and 4-word blocks assuming a 32-bit address?

Outline

❑ Why Memory Hierarchy

❑ How Memory Hierarchy?

- SRAM (Cache) & DRAM (main memory)
- Memory System

❑ Cache Basics

❑ Cache Performance

❑ Reduce Cache Miss Rates

❑ Summary

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

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)

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

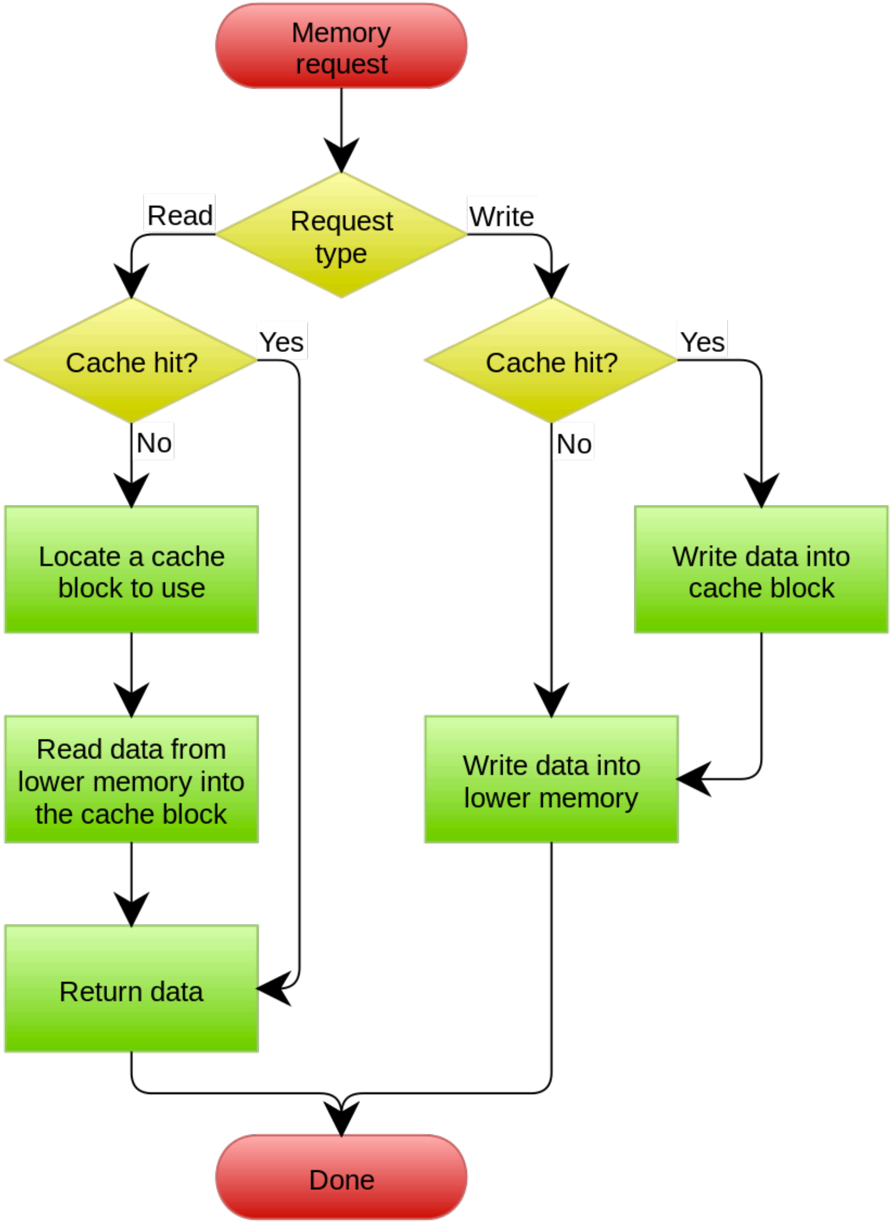
  Or (normally used in write-back caches)

  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

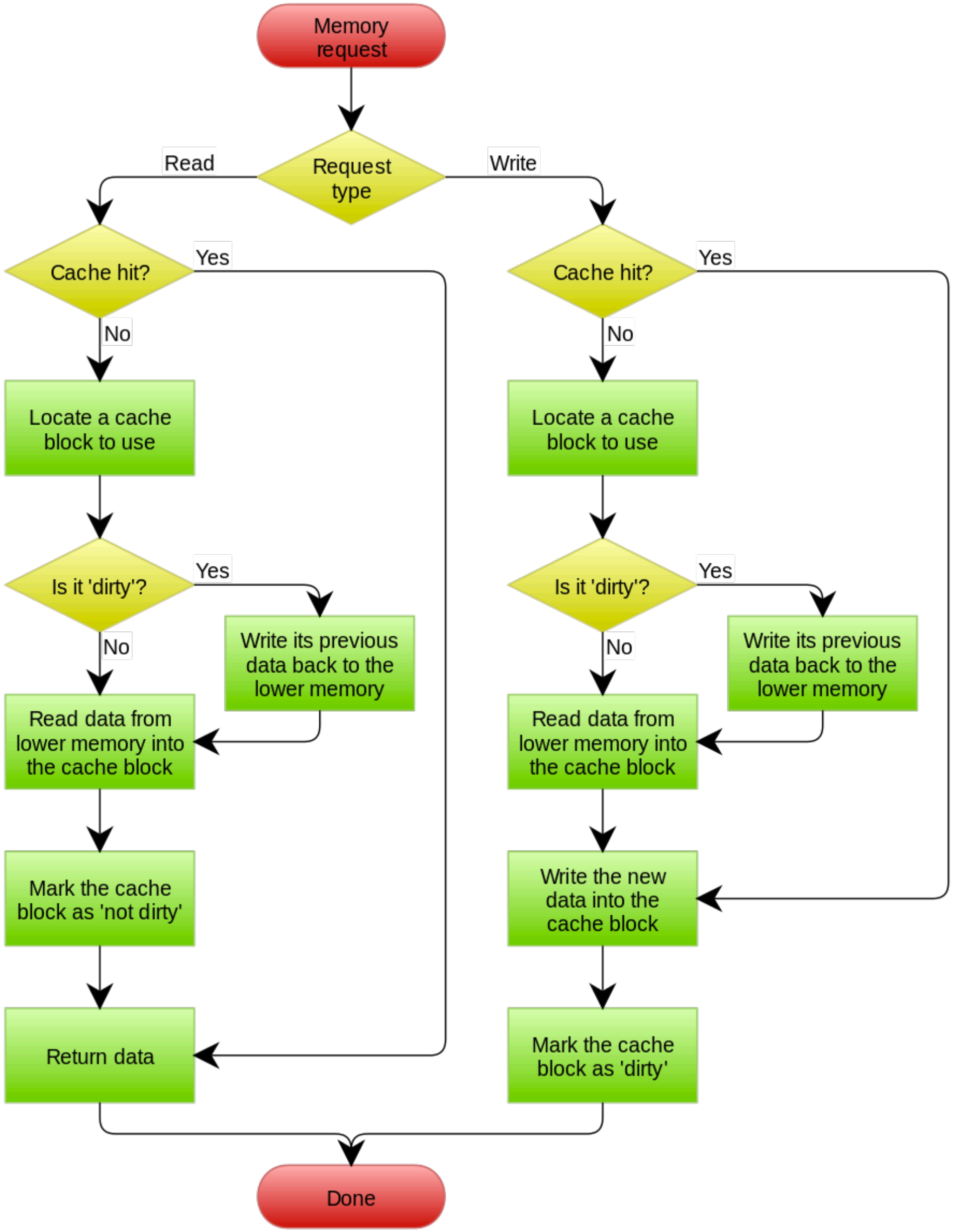
  Or (normally used in write-through caches with a write buffer)

  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

Write-Through Cache with No-Write Allocation


Memory request
Request type
Read
Write
Cache hit?
Yes
No
Cache hit?
Yes
No
Locate a cache block to use
Read data from lower memory into the cache block
Return data
Write data into cache block
Write data into lower memory
Done

Write-Back Cache with Write Allocation


Memory request
Request type
Read
Write
Cache hit?
Yes
No
Locate a cache block to use
Is it 'dirty'?
Yes
No
Write its previous data back to the lower memory
Read data from lower memory into the cache block
Mark the cache block as 'not dirty'
Return data
Cache hit?
Yes
No
Locate a cache block to use
Is it 'dirty'?
Yes
No
Write its previous data back to the lower memory
Read data from lower memory into the cache block
Write the new data into the cache block
Mark the cache block as 'dirty'
Done

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)

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

$$\text{Memory-stall cycles} = 2\% \times 100 + 36\% \times 4\% \times 100 = 3.44$$

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

  more than twice the $\text{CPI}_{\text{ideal}}$!

- What if the $\text{CPI}_{\text{ideal}}$ is reduced to 1?
- 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?

Outline

- ❑ Why Memory Hierarchy
- ❑ How Memory Hierarchy?
  - ● SRAM (Cache) & DRAM (main memory)
  - ● Memory System
- ❑ Cache Basics
- ❑ Cache Performance
- ❑ Reduce Cache Miss Rates
- ❑ Summary

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)

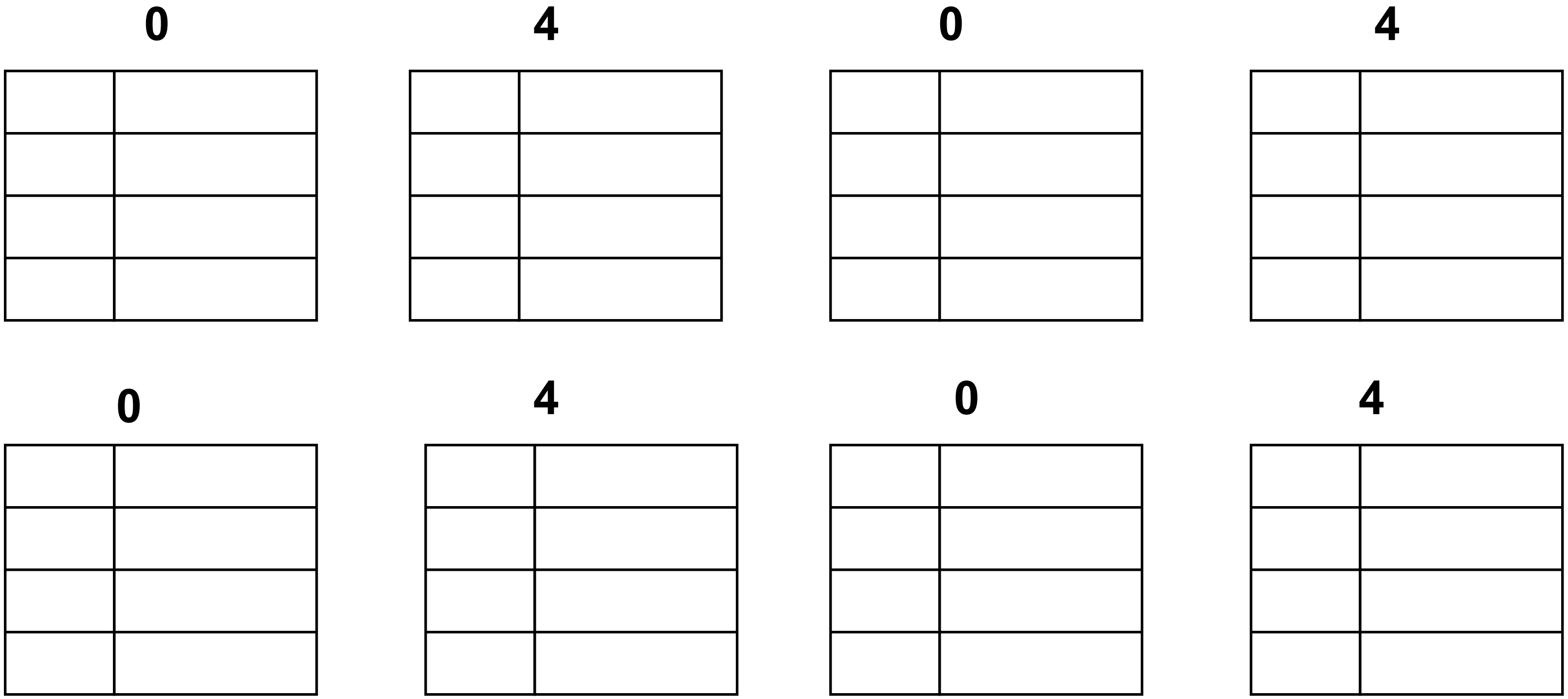
(block address) modulo (# sets in the cache)

Another Reference String Mapping

- Consider the main memory word reference string

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

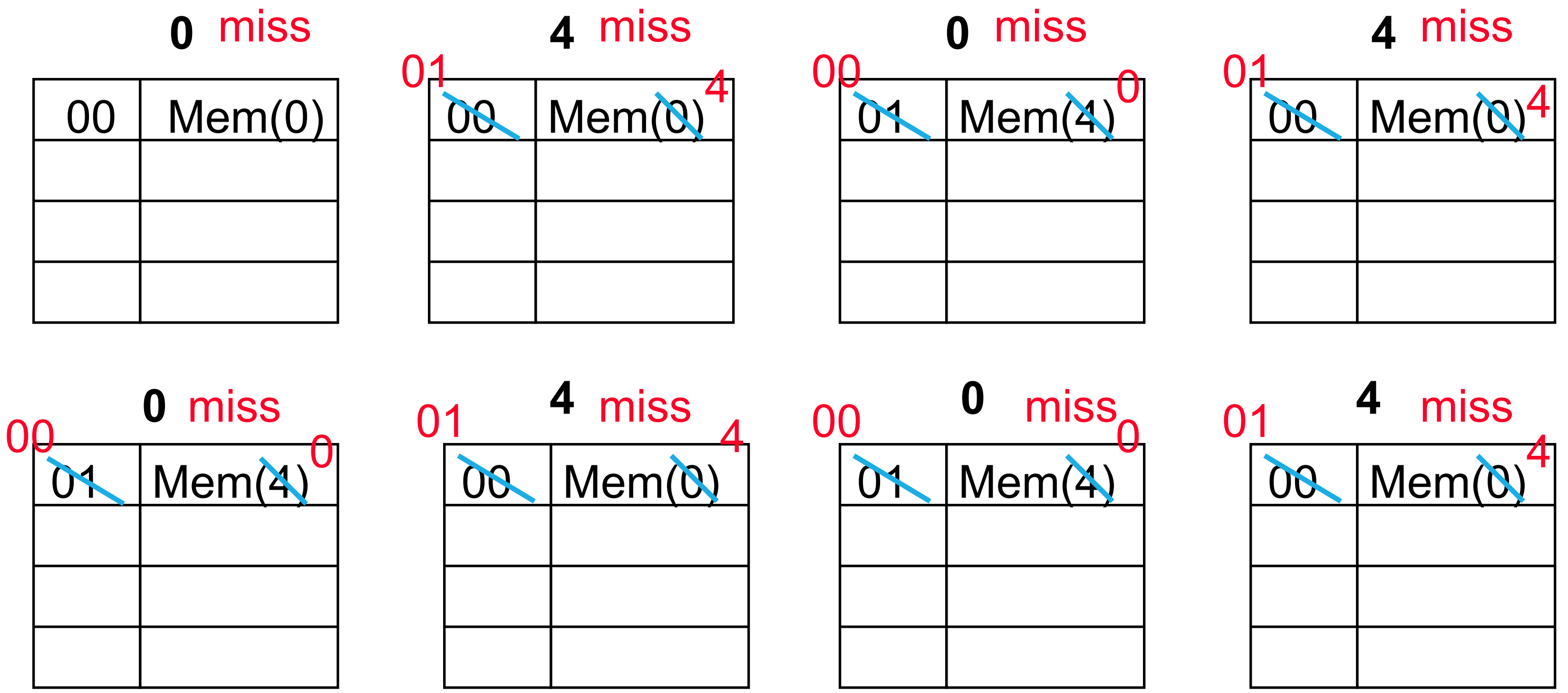
0 4 0 4 0 4 0 4



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

00	Mem(0)

4 miss

~~00~~ 01	Mem(~~0~~) 4

0 miss

~~01~~ 00	Mem(~~4~~) 0

4 miss

~~00~~ 01	Mem(~~0~~) 4

0 miss

~~01~~ 00	Mem(~~4~~) 0

4 miss

~~00~~ 01	Mem(~~0~~) 4

0 miss

~~01~~ 00	Mem(~~4~~) 0

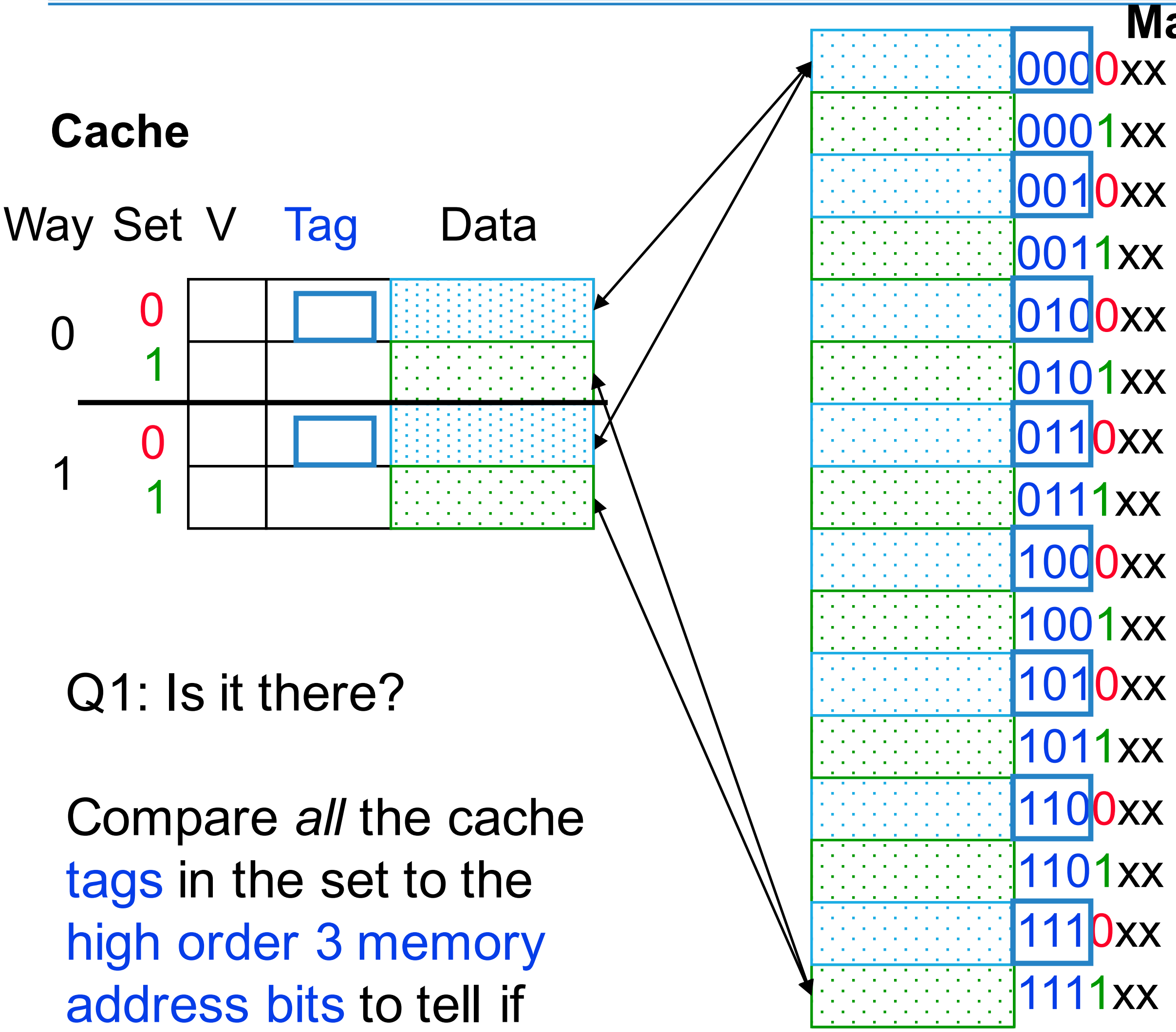
4 miss

~~00~~ 01	Mem(~~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



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

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)

Q1: Is it there?

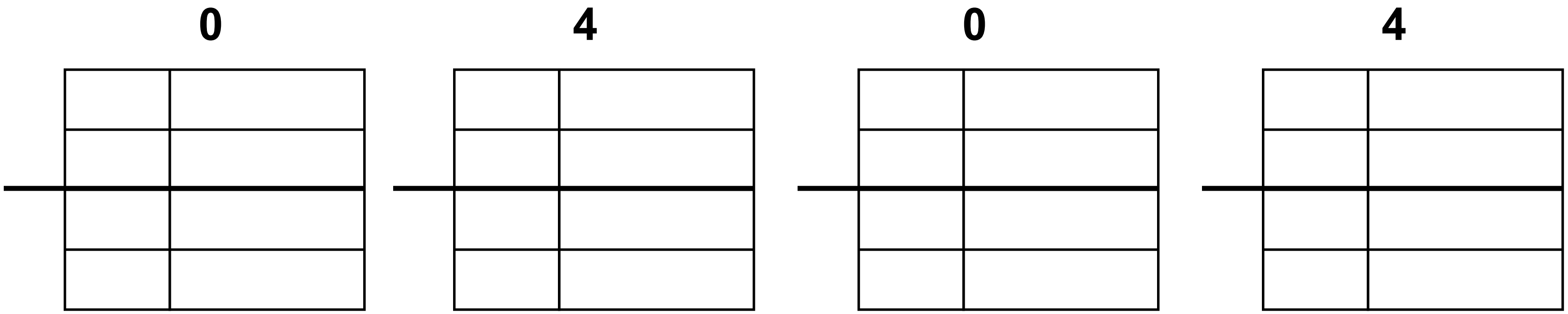
Compare *all* the cache tags in the set to the high order 3 memory address bits to tell if the memory block is in the cache

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

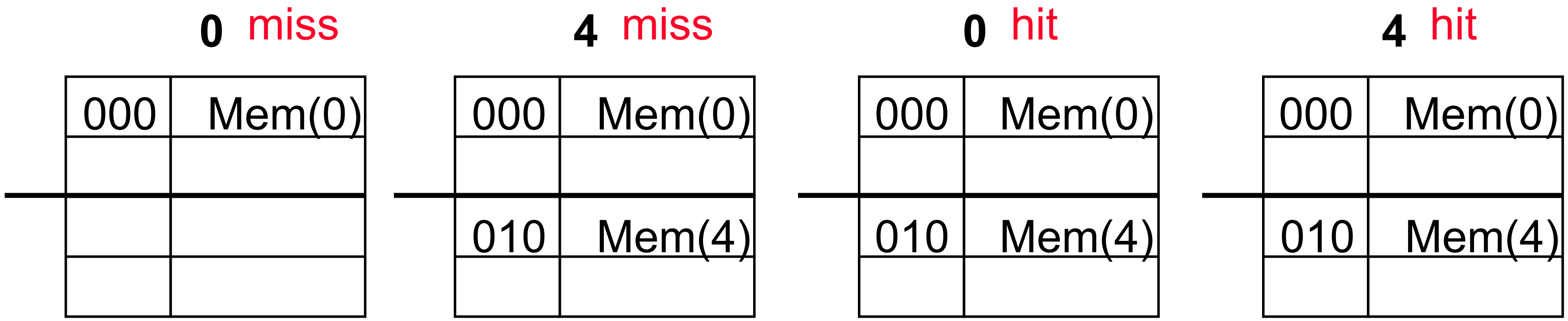


Another Reference String Mapping

- ❑ 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

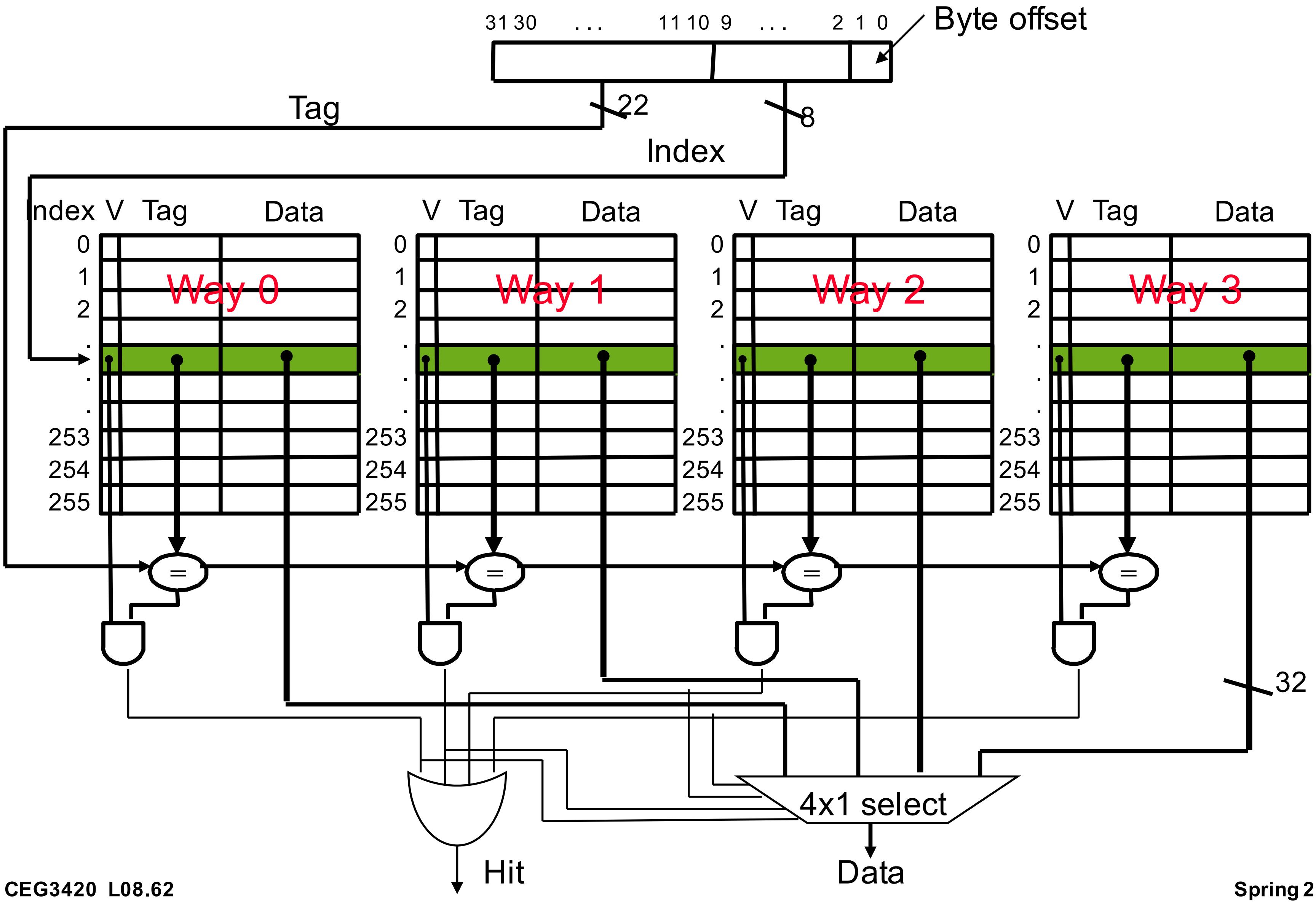
000	Mem(0)
010	Mem(4)

- ● 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!

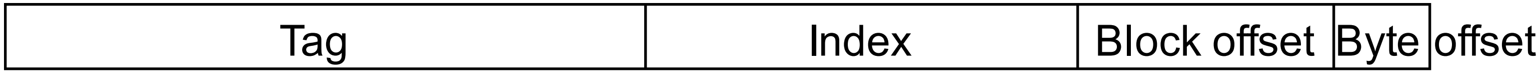
Four-Way Set Associative Cache

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



Range of Set Associative Caches

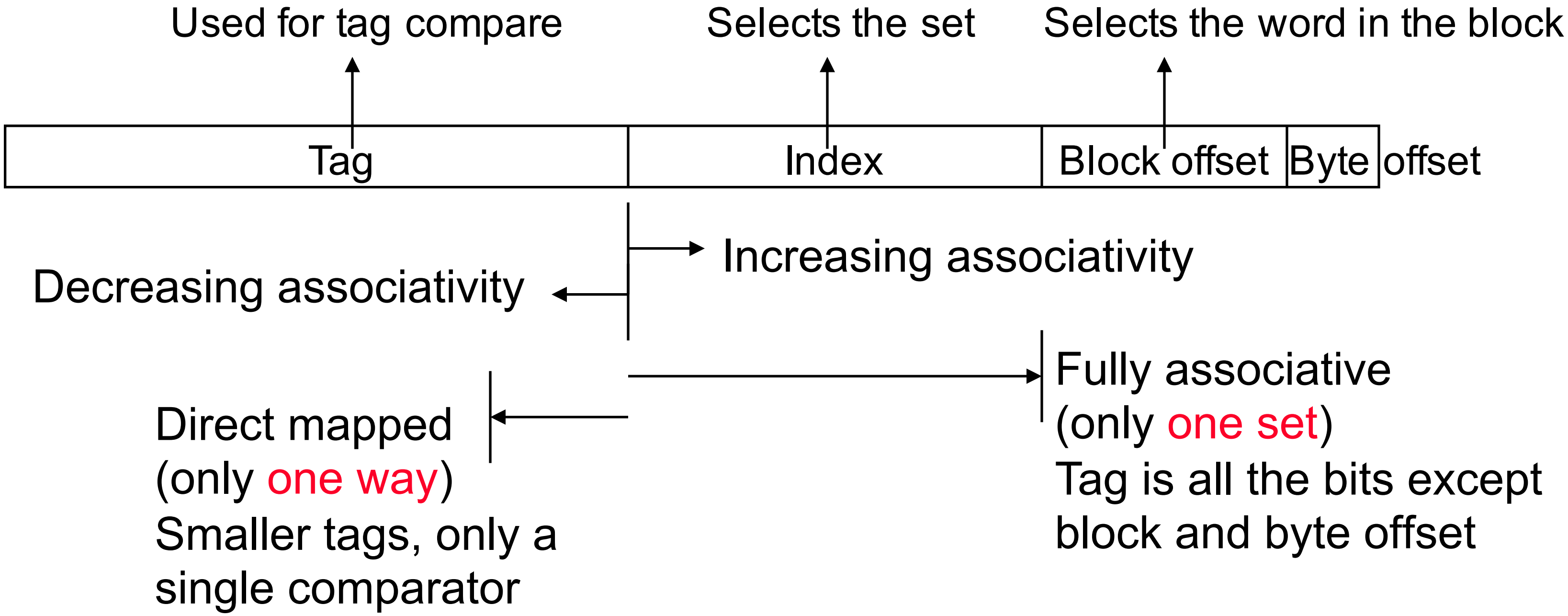
- For a fixed size cache, each increase by a factor of two in associativity doubles the number of blocks per set (i.e., the number or ways) and halves the number of sets – decreases the size of the index by 1 bit and increases the size of the tag by 1 bit



Tag | Index | Block offset | Byte offset

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

Increasing associativity

Decreasing associativity

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

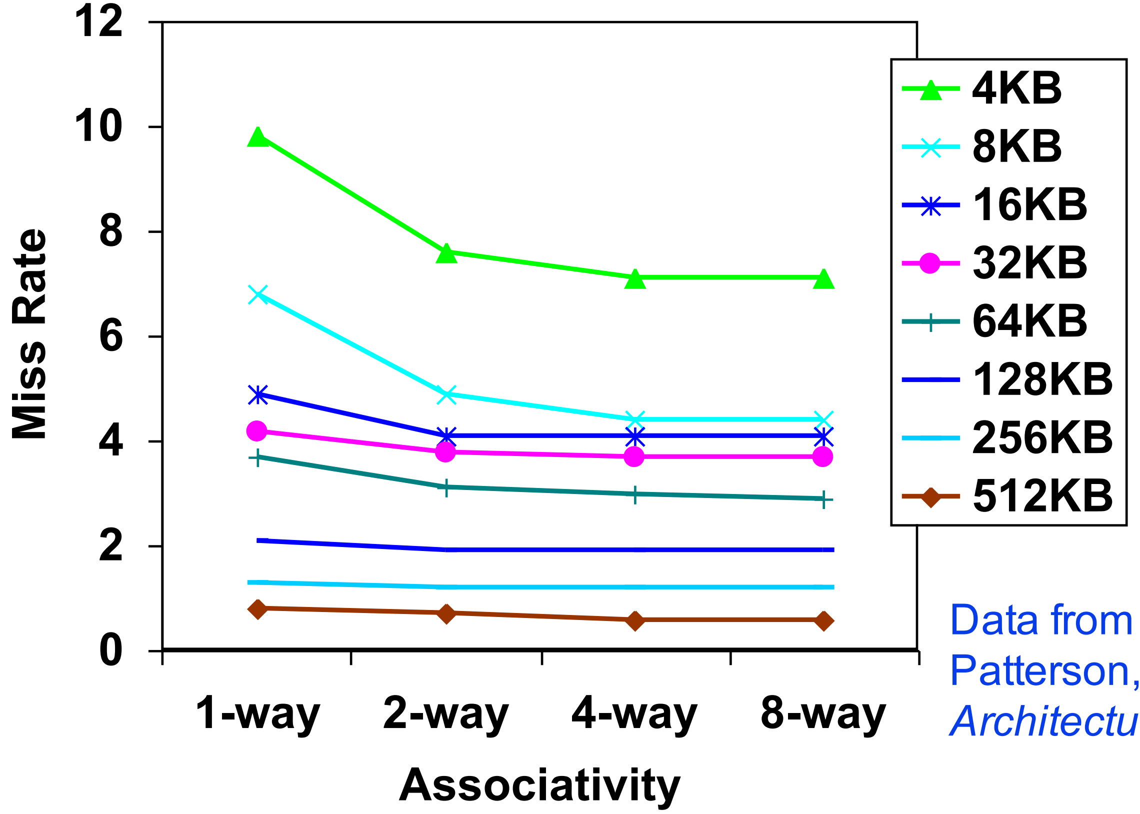
Direct mapped (only one way)
Smaller tags, only a single comparator

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)

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

$$\begin{aligned} \text{CPI}_{\text{stalls}} = 2 &+ (.02 \times 25 + .005 \times 100) \\ &+ (.36 \times .04 \times 25 + .36 \times .005 \times 100) = 3.54 \end{aligned}$$

(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

Two Machines' Cache Parameters

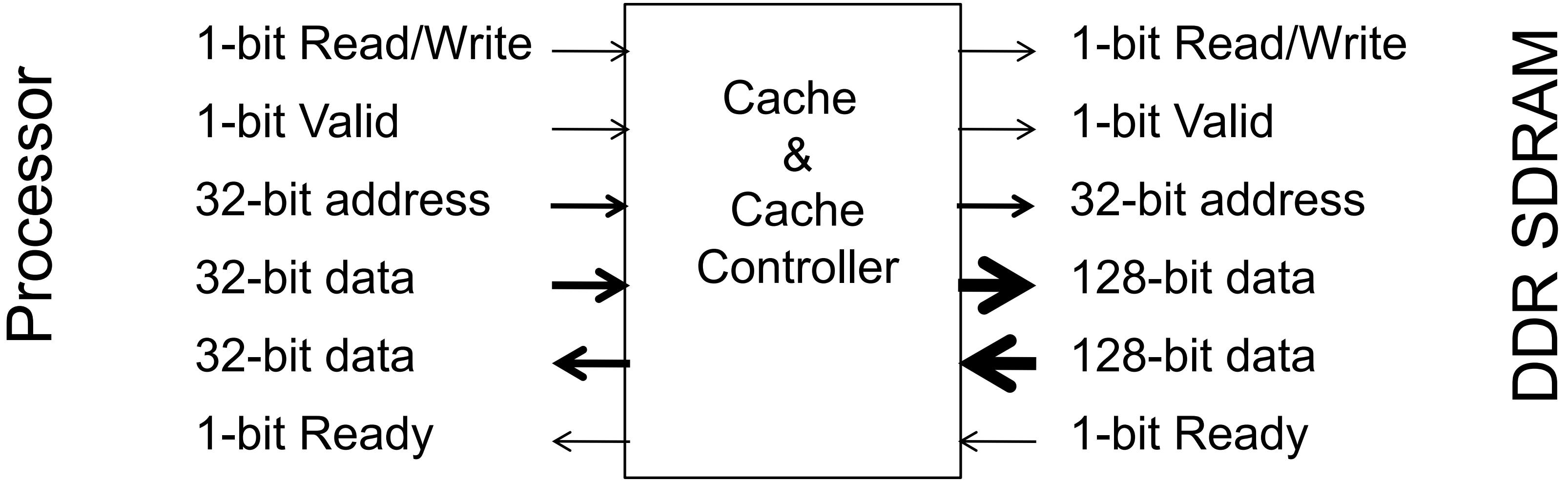
	Intel Nehalem	AMD Barcelona
L1 cache organization & size	Split I$ and D$; 32KB for each per core; 64B blocks	Split I$ and D$; 64KB for each per core; 64B blocks
L1 associativity	4-way (I), 8-way (D) set assoc.; ~LRU replacement	2-way set assoc.; LRU replacement
L1 write policy	write-back, write-allocate	write-back, write-allocate
L2 cache organization & size	Unified; 256kB (0.25MB) per core; 64B blocks	Unified; 512KB (0.5MB) per core; 64B blocks
L2 associativity	8-way set assoc.; ~LRU	16-way set assoc.; ~LRU
L2 write policy	write-back, write-allocate	write-back, write-allocate
L3 cache organization & size	Unified; 8192KB (8MB) shared by cores; 64B blocks	Unified; 2048KB (2MB) shared by cores; 64B blocks
L3 associativity	16-way set assoc.	32-way set assoc.; evict block shared by fewest cores
L3 write policy	write-back, write-allocate	write-back; write-allocate

Two Machines’ Cache Parameters

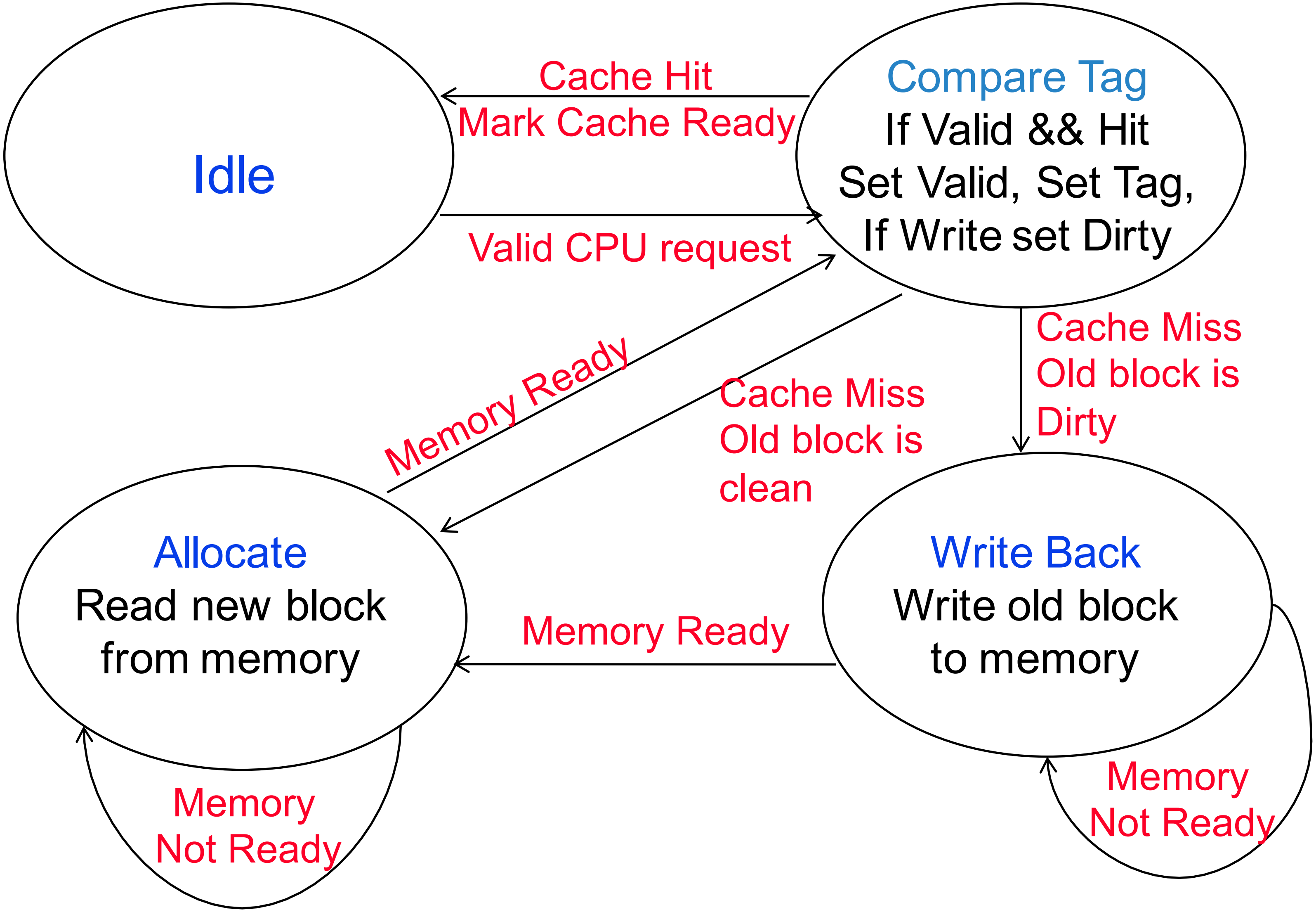
	Intel P4	AMD Opteron
L1 organization	Split I$ and D$	Split I$ and D$
L1 cache size	8KB for D$, 96KB for trace cache (~I$)	64KB for each of I$ and D$
L1 block size	64 bytes	64 bytes
L1 associativity	4-way set assoc.	2-way set assoc.
L1 replacement	~ LRU	LRU
L1 write policy	write-through	write-back
L2 organization	Unified	Unified
L2 cache size	512KB	1024KB (1MB)
L2 block size	128 bytes	64 bytes
L2 associativity	8-way set assoc.	16-way set assoc.
L2 replacement	~LRU	~LRU
L2 write policy	write-back	write-back

FSM Cache Controller

- Key characteristics for a simple L1 cache
  - Direct mapped
  - Write-back using write-allocate
  - Block size of 4 32-bit words (so 16B); Cache size of 16KB (so 1024 blocks)
  - 18-bit tags, 10-bit index, 2-bit block offset, 2-bit byte offset, dirty bit, valid bit, LRU bits (if set associative)



Four State Cache Controller



Outline

- ❑ Why Memory Hierarchy
- ❑ How Memory Hierarchy?
  - ● SRAM (Cache) & DRAM (main memory)
  - ● Memory System
- ❑ Cache Basics
- ❑ Cache Performance
- ❑ Reduce Cache Miss Rates
- ❑ Summary

Summary: Improving Cache Performance

0. Reduce the time to hit in the cache

- smaller cache
- direct mapped cache
- smaller blocks
- for writes
  - no write allocate – no “hit” on cache, just write to write buffer
  - write allocate – to avoid two cycles (first check for hit, then write) pipeline writes via a delayed write buffer to cache

1. Reduce the miss rate

- bigger cache
- more flexible placement (increase associativity)
- larger blocks (16 to 64 bytes typical)
- victim cache – small buffer holding most recently discarded blocks

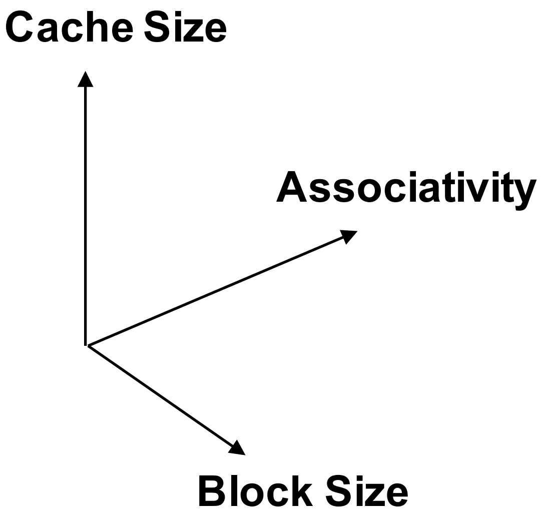
Summary: Improving Cache Performance

2. Reduce the miss penalty

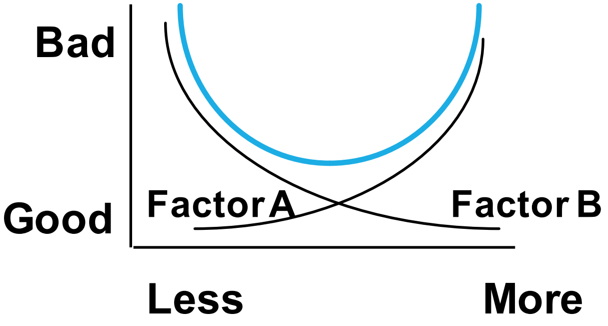
- smaller blocks
- use a write buffer to hold dirty blocks being replaced so don't have to wait for the write to complete before reading
- check write buffer (and/or victim cache) on read miss – may get lucky
- for large blocks fetch critical word first
- use multiple cache levels – L2 cache not tied to CPU clock rate
- faster backing store/improved memory bandwidth
  - wider buses
  - memory interleaving, DDR SDRAMs

Summary: The Cache Design Space

- Several interacting dimensions
  - cache size
  - block size
  - associativity
  - replacement policy
  - write-through vs write-back
  - write allocation



- The optimal choice is a compromise
  - depends on access characteristics
    - workload
    - use (I-cache, D-cache, TLB)
  - depends on technology / cost
- Simplicity often wins


Bad
Good
Factor A
Factor B
Less
More