

---

# CENG 3420

## Computer Organization and Design

### Lecture 08: Cache Review

Bei Yu

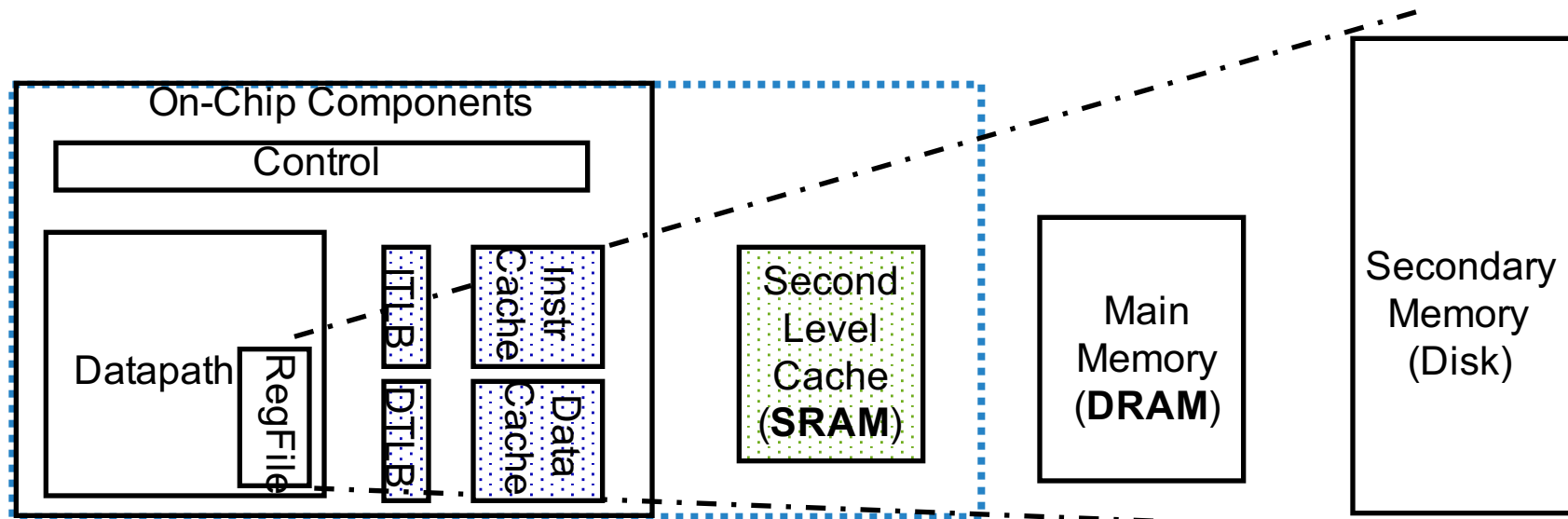


香港中文大學

The Chinese University of Hong Kong

# 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



<b>Speed (%cycles):</b>	1/2's	1's	10's	100's	10,000's
<b>Size (bytes):</b>	100's	10K's	M's	G's	T's
<b>Cost:</b>	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

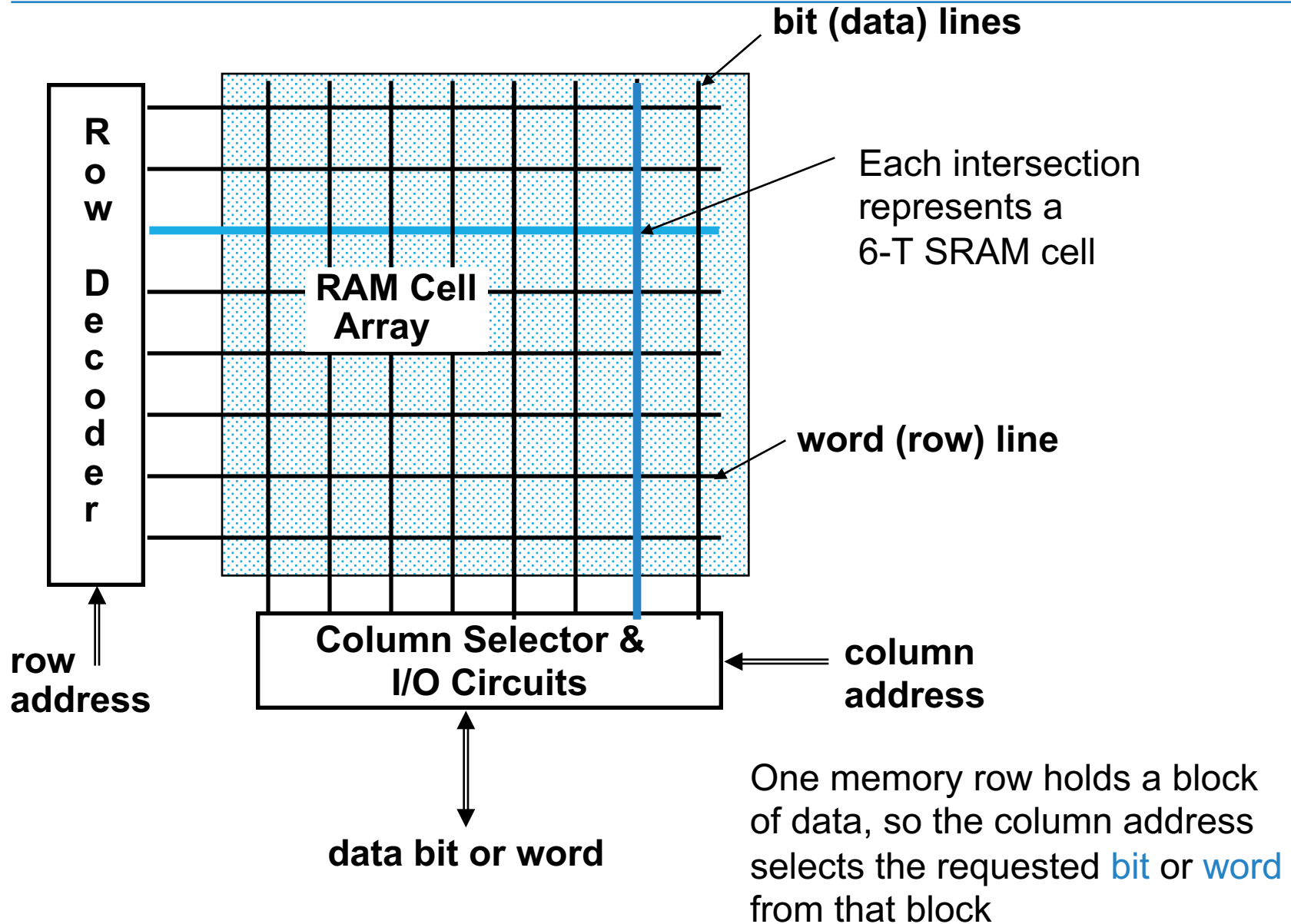
🦋 Keep **most recently accessed** data items closer to the processor

## □ Spatial Locality (locality in space)

- If a memory location is referenced, the locations with nearby addresses will tend to be referenced soon

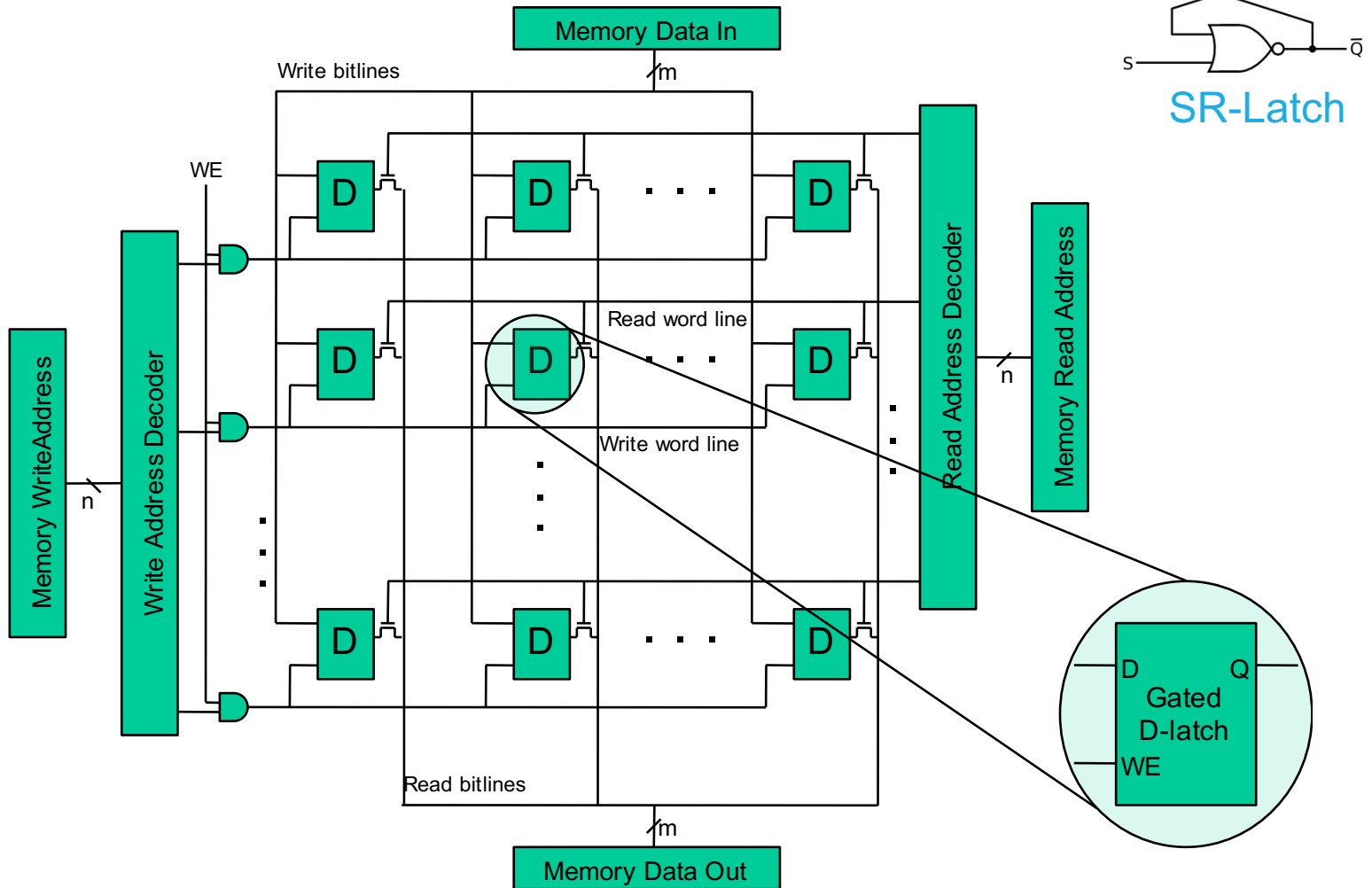
🦋 Move blocks consisting of **contiguous words** closer to the processor

# Classical SRAM Organization

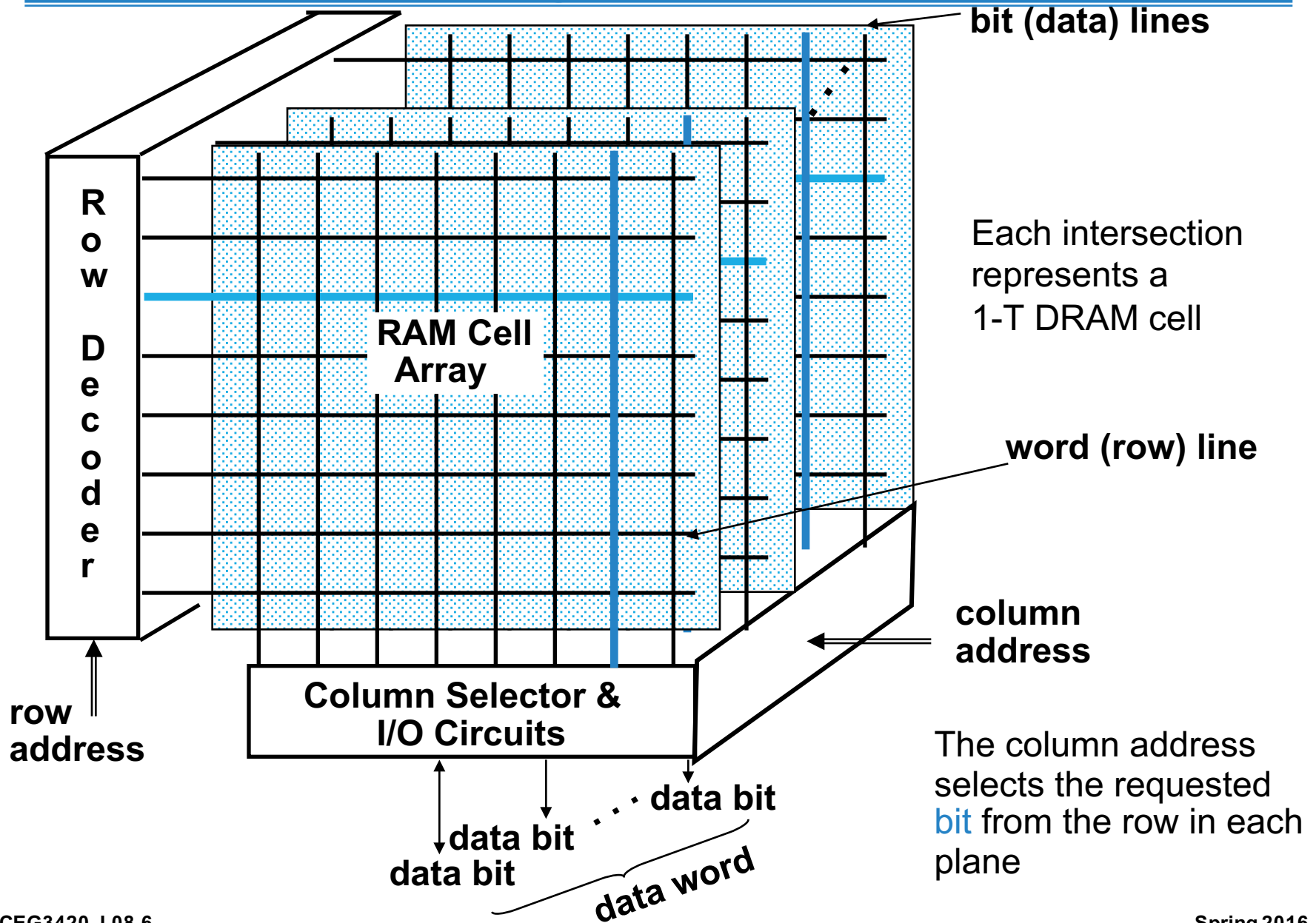


# Classical SRAM Organization

## □ Latch based memory



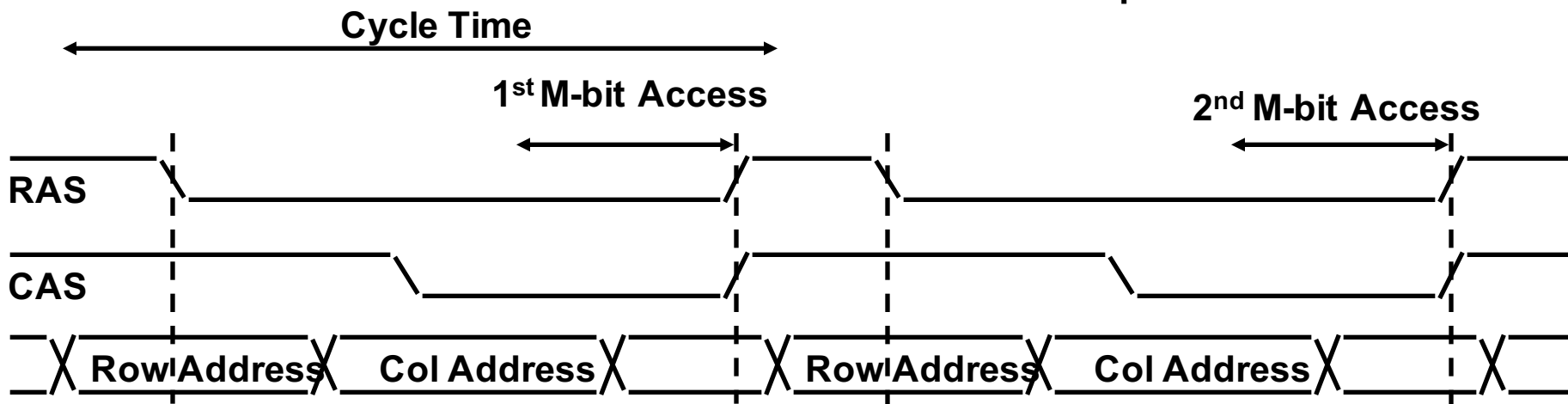
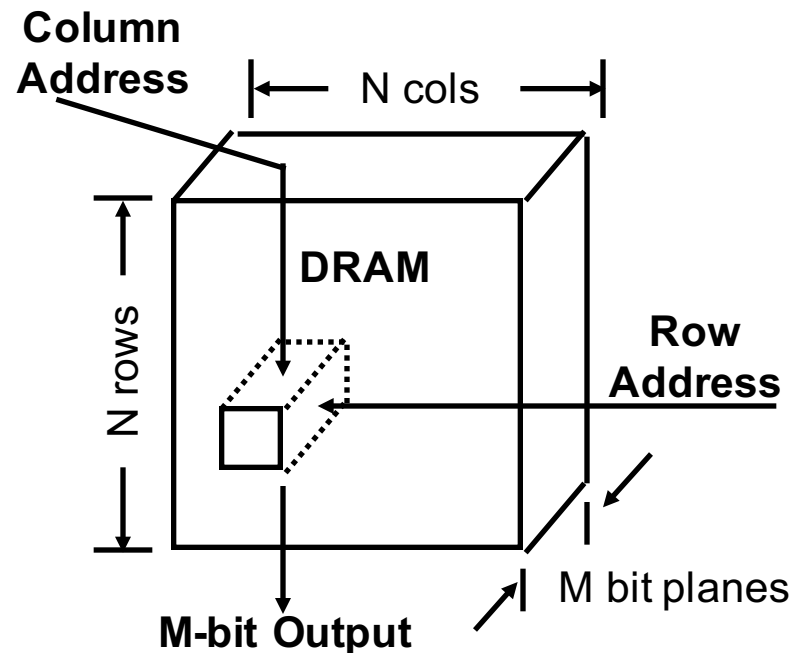
# Classical DRAM Organization



# Classical DRAM Operation

## □ DRAM Organization:

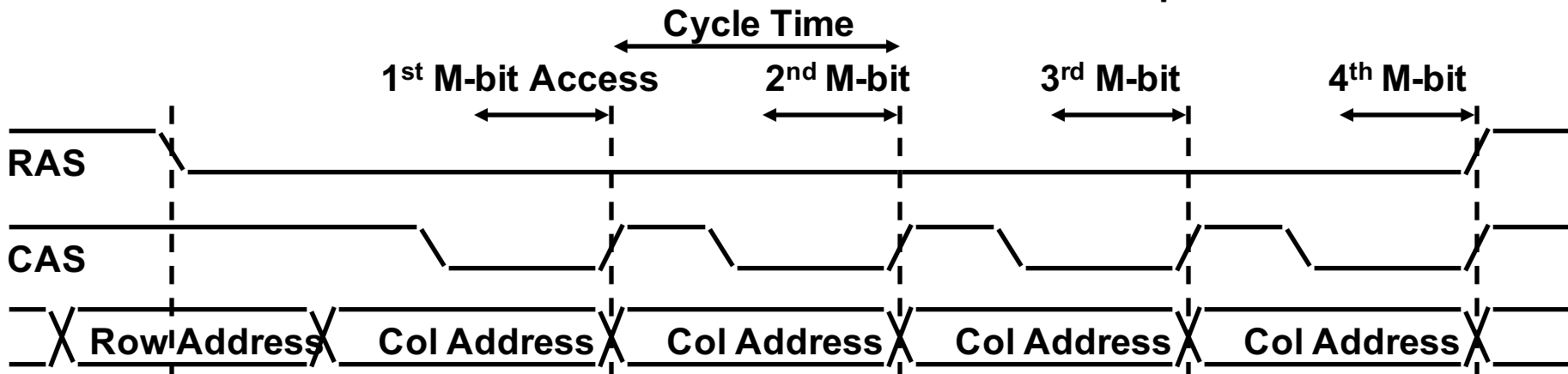
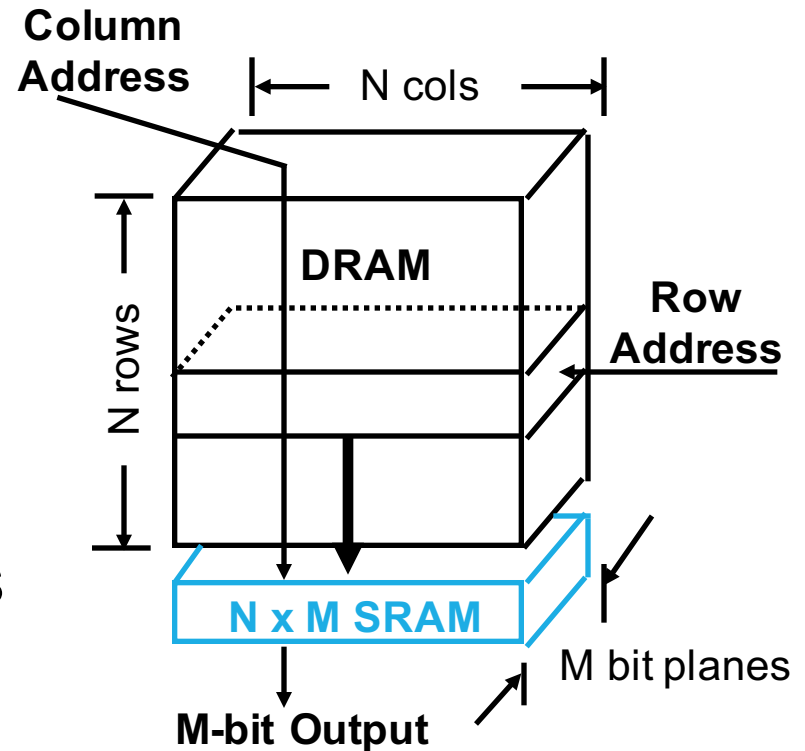
- N rows x N column x M-bit
- Read or Write M-bit at a time
- Each M-bit access requires a RAS / CAS cycle



# Page Mode DRAM Operation

## Page Mode DRAM

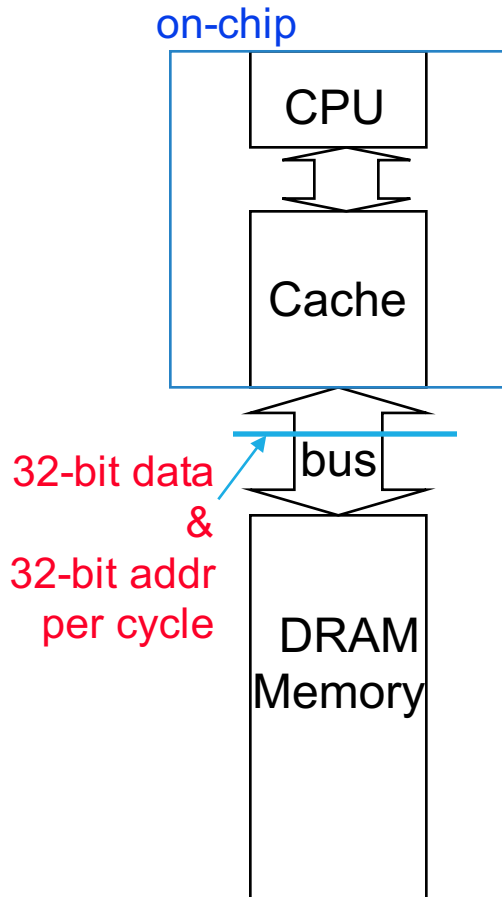
- N x M SRAM to save a row
- After a row is read into the SRAM “register”
  - Only CAS is needed to access other M-bit words on that row
  - RAS remains asserted while CAS is toggled





# 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)

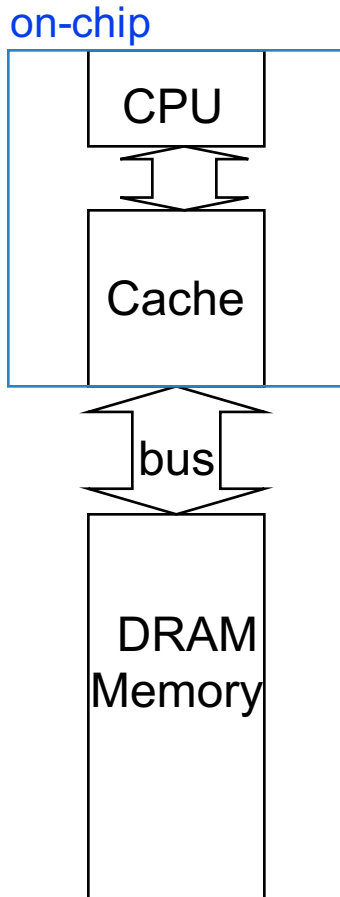
- Assume

1. 1 clock cycle to send the addr
2. 15 clock cycles to get the 1<sup>st</sup> word in the block from DRAM, 5 clock cycles for 2<sup>nd</sup>, 3<sup>rd</sup>, 4<sup>th</sup> 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

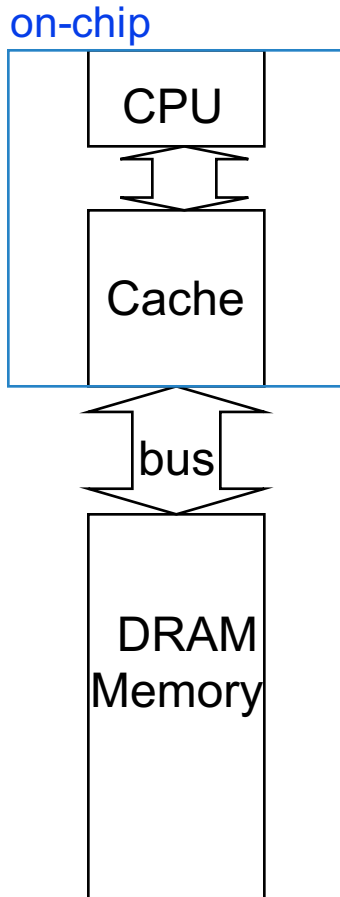
cycle to return data

— 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

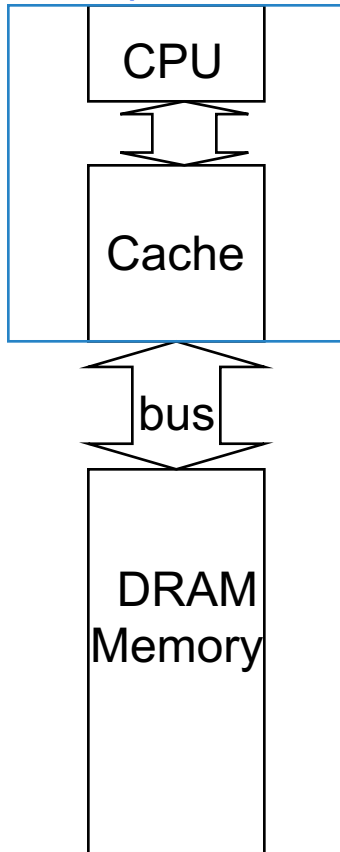
1  
17 total clock cycles miss penalty

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

$$4/17 = 0.235 \text{ bytes per memory bus clock cycle}$$

# One Word Wide Bus, Four Word Blocks

on-chip



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

cycle to send 1<sup>st</sup> address

cycles to read DRAM

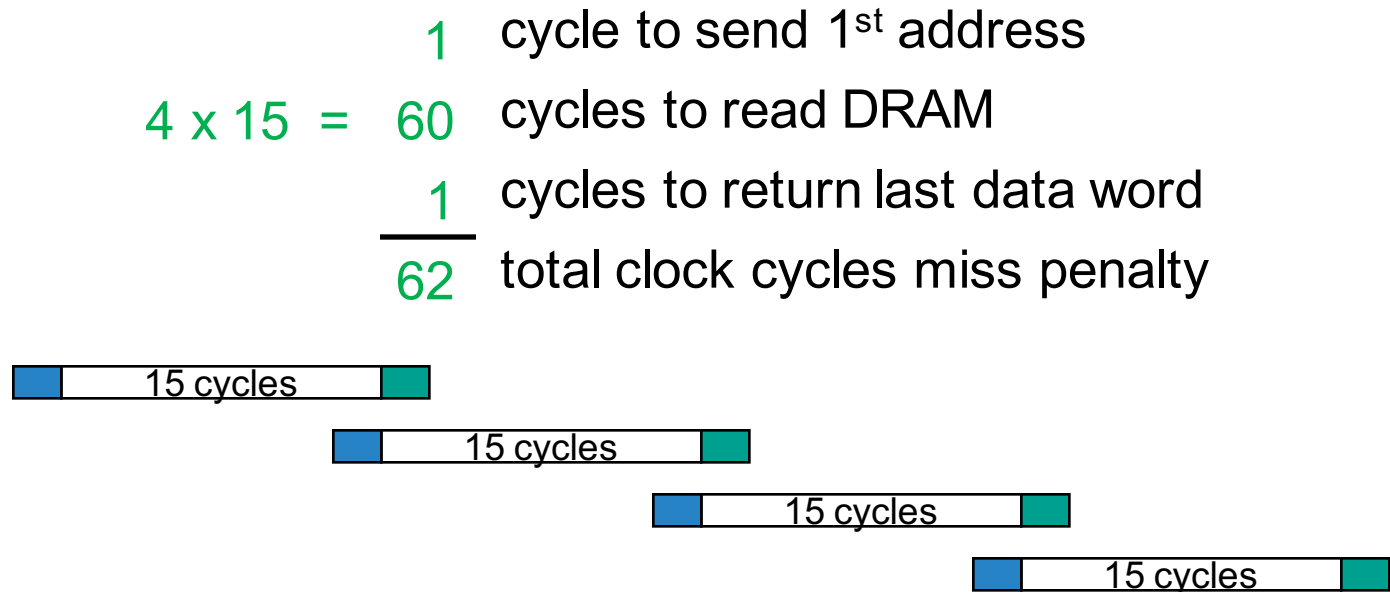
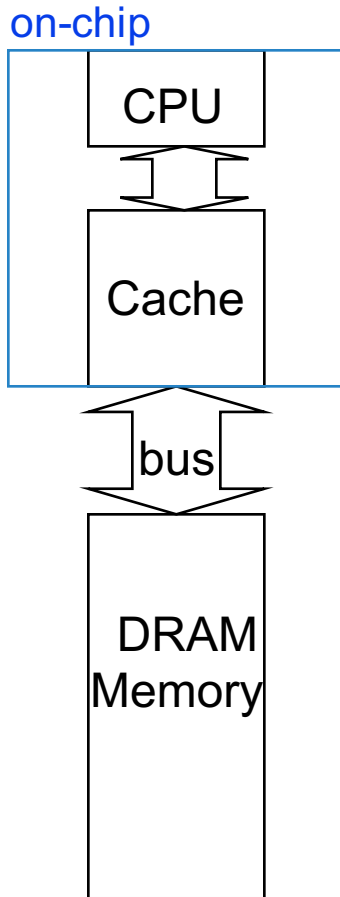
cycles to return last data word

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

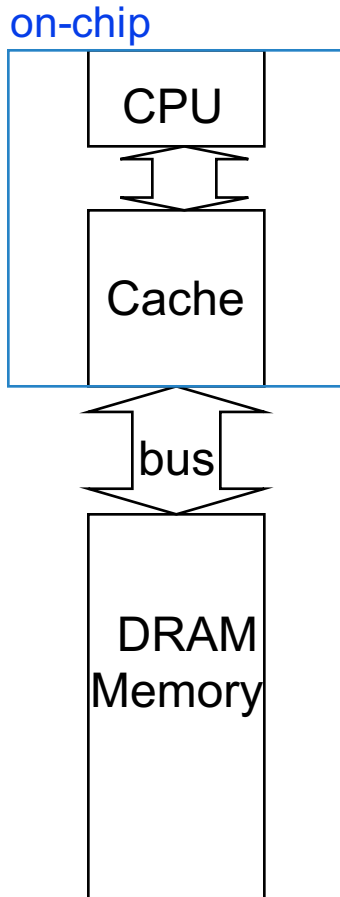
- What if the block size is four words and each word is in a different DRAM row?



- Number of bytes transferred per clock cycle (bandwidth) for a single miss is  $(4 \times 4)/62 = 0.258$  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 1<sup>st</sup> address

cycles to read DRAM

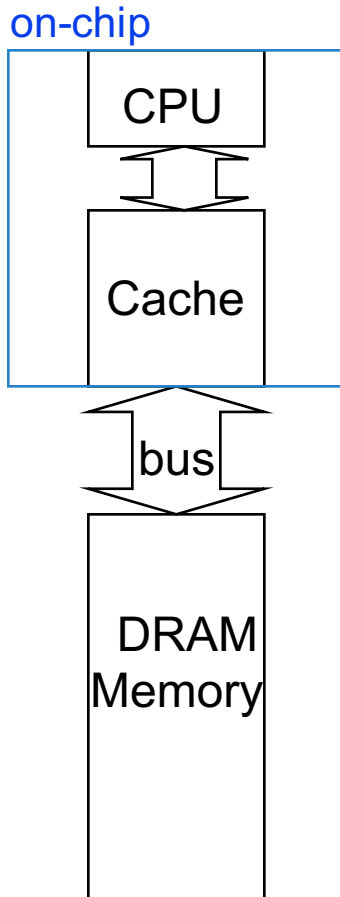
cycles to return last data word

— total clock cycles miss penalty

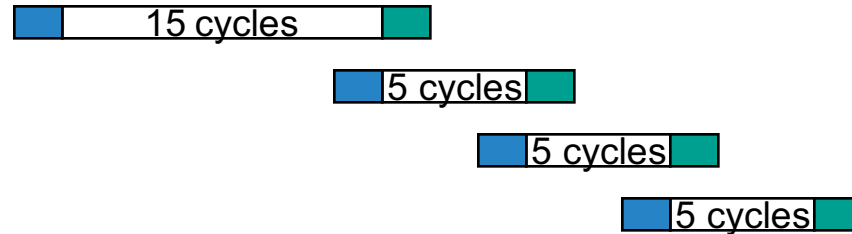
- Number of bytes transferred per clock cycle (bandwidth) for a single miss is  
bytes per clock

# One Word Wide Bus, Four Word Blocks

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



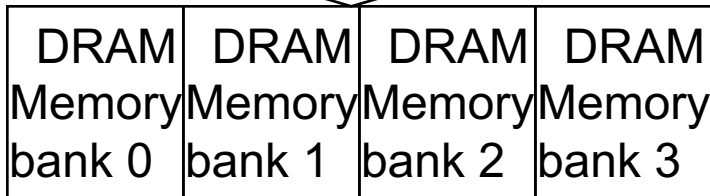
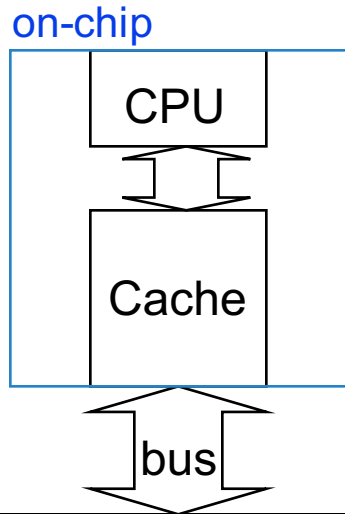
$$\begin{array}{r}
 1 \text{ cycle to send 1}^{\text{st}} \text{ address} \\
 15 + 3 \cdot 5 = 30 \text{ cycles to read DRAM} \\
 1 \text{ cycles to return last data word} \\
 \hline
 32 \text{ total clock cycles miss penalty}
 \end{array}$$



- Number of bytes transferred per clock cycle (bandwidth) for a single miss is  $(4 \times 4) / 32 = 0.5$  bytes per clock

# Interleaved Memory, One Word Wide Bus

- For a block size of four words



cycle to send 1<sup>st</sup> address  
 cycles to read DRAM banks  
 \_\_\_\_\_ cycles to return last data word  
 total clock cycles miss penalty

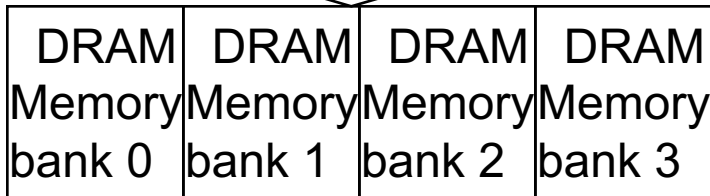
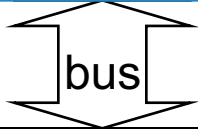
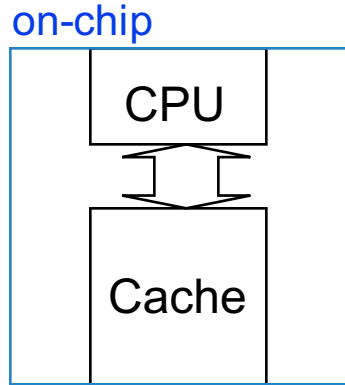
- Number of bytes transferred per clock cycle (bandwidth) for a single miss is

bytes per clock

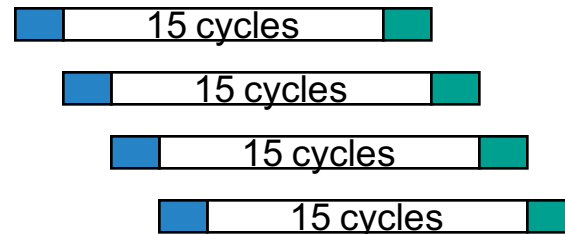


# Interleaved Memory, One Word Wide Bus

For a block size of four words



1 cycle to send 1<sup>st</sup> address  
 $15 + 3 = 18$  cycles to read DRAM banks  
 1 cycles to return last data word  
 20 total clock cycles miss penalty


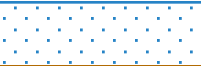




Number of bytes transferred per clock cycle (bandwidth) for a single miss is

$$(4 \times 4) / 20 = 0.8 \text{ bytes per clock}$$

# Caching: A Simple First Example

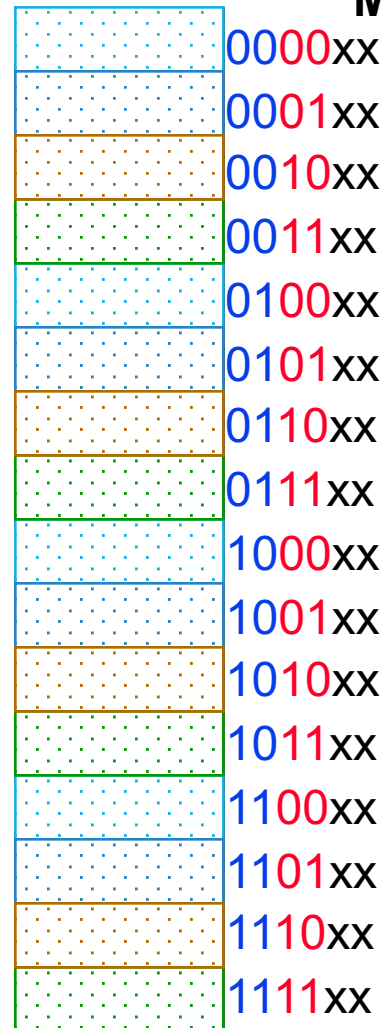
## Cache

Index	Valid	Tag	Data
00			
01			
10			
11			

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

## Main Memory



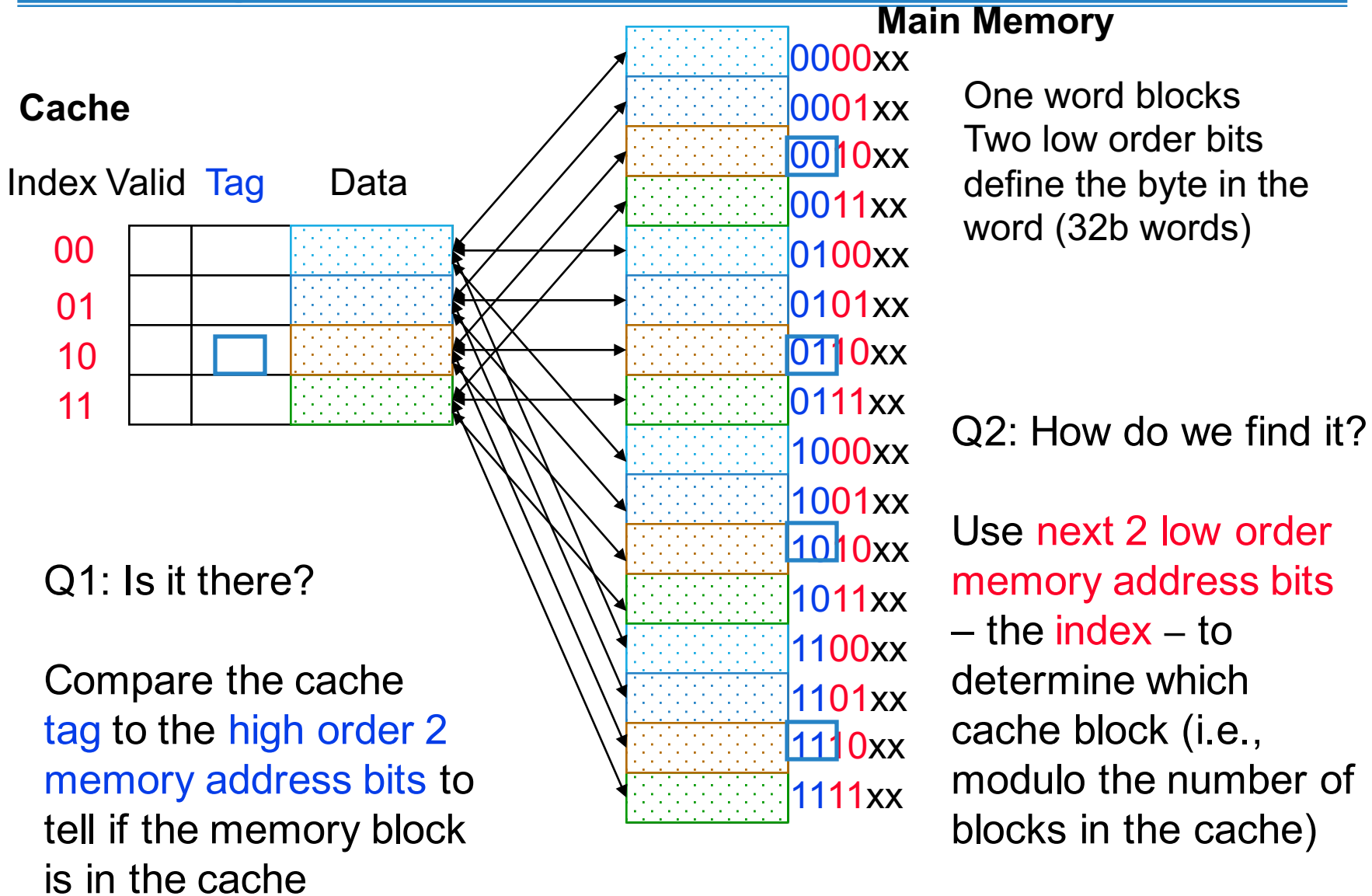
One word blocks  
Two low order bits define the byte in the word (32b words)

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)

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

# Caching: A Simple First Example



(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

**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

<del>00</del>	<del>Mem(0)</del>
00	Mem(1)
00	Mem(2)
00	Mem(3)

4

**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**

11

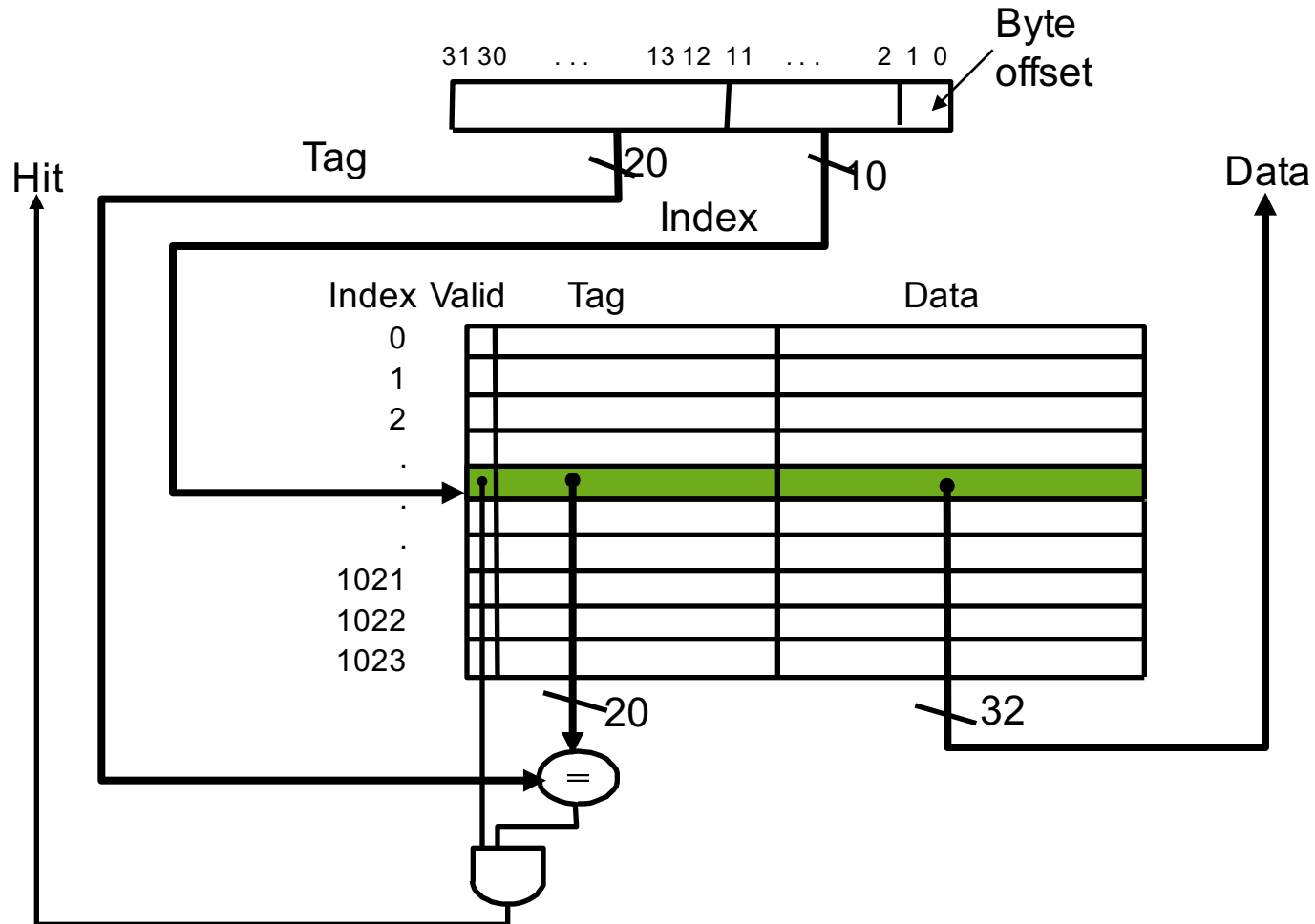
<del>01</del>	<del>Mem(4)</del>
00	Mem(1)
00	Mem(2)
<del>00</del>	<del>Mem(3)</del>

15

● 8 requests, 6 misses

# MIPS Direct Mapped Cache Example

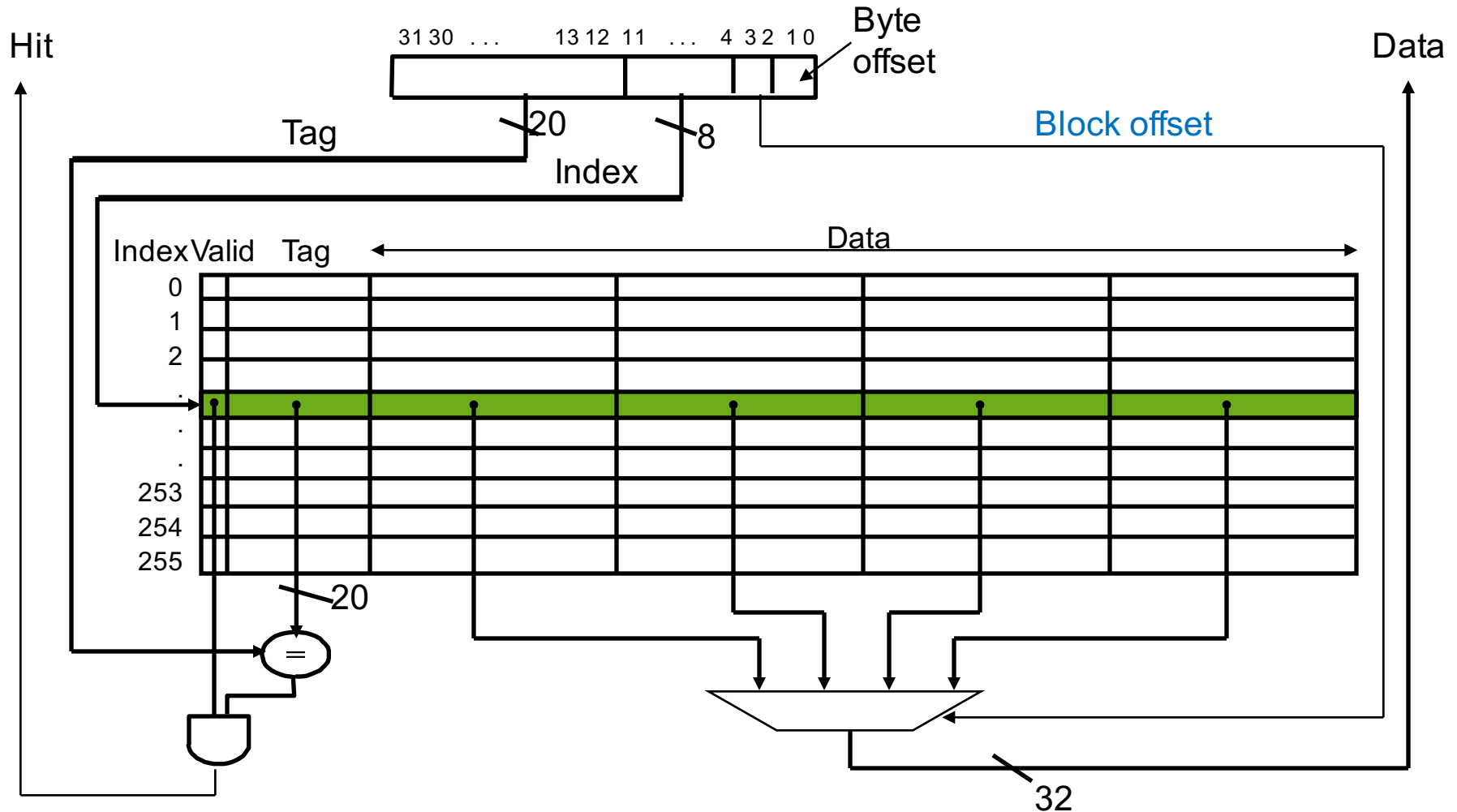
- One word blocks, cache size = 1K words (or 4KB)



*What kind of locality are we taking advantage of?*

# Multiword Block Direct Mapped Cache

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



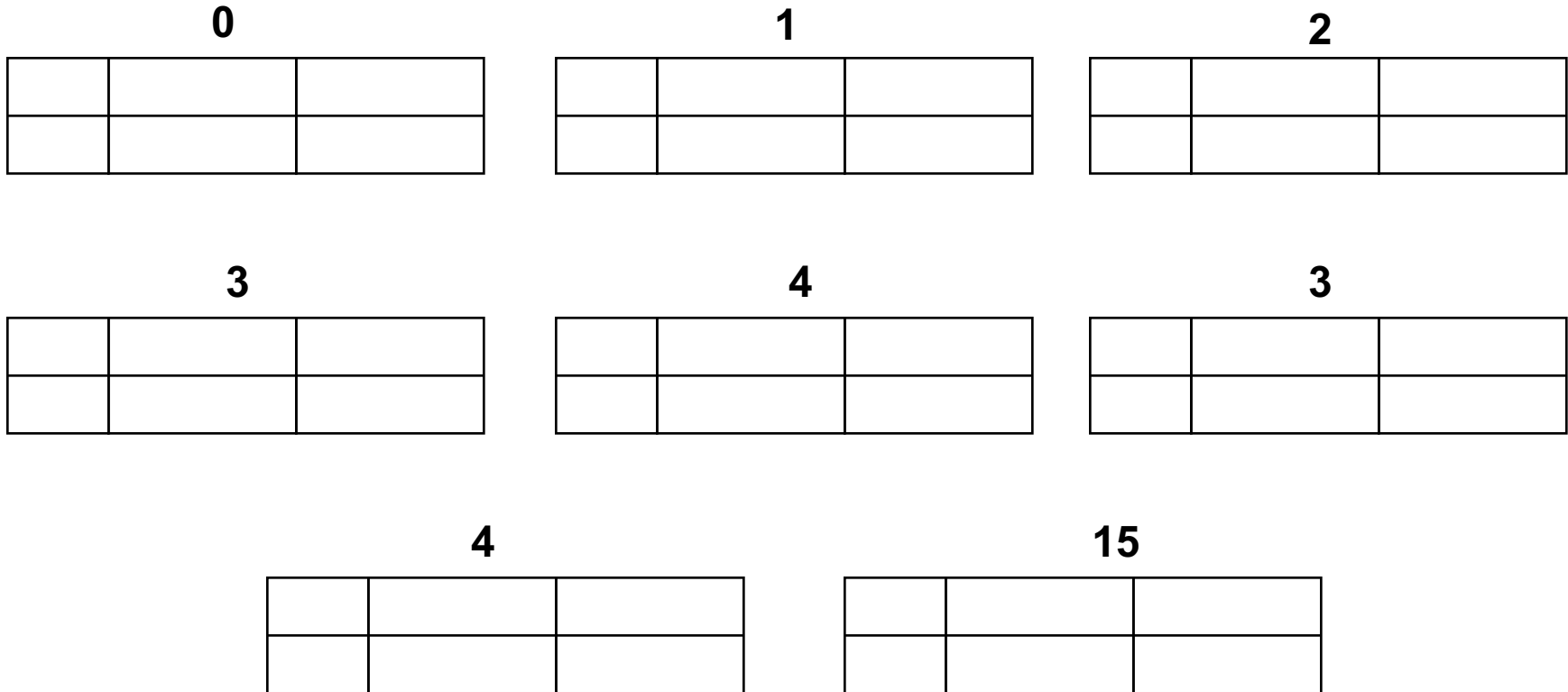
*What kind of locality are we taking advantage of?*

# 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





# 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**

<del>01</del>	<del>Mem(1)</del>	<del>Mem(0)</del>
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**

<del>11</del>	Mem(5)	<del>Mem(4)</del>
<del>00</del>	<del>Mem(3)</del>	<del>Mem(2)</del>

- 8 requests, 4 misses

# Cache Field Sizes

---

- The number of bits in a cache includes both the storage for data and for the tags
  - 32-bit 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  
 $2^n \times (\text{block size} + \text{tag field size} + \text{valid field size})$

## 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?

# 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

# 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

# Measuring Cache Performance

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

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

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

$$\begin{aligned}\text{Read-stall cycles} &= \text{reads/program} \times \text{read miss rate} \\ &\quad \times \text{read miss penalty}\end{aligned}$$

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

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

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

# 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

0


4


0


4


0


4


0

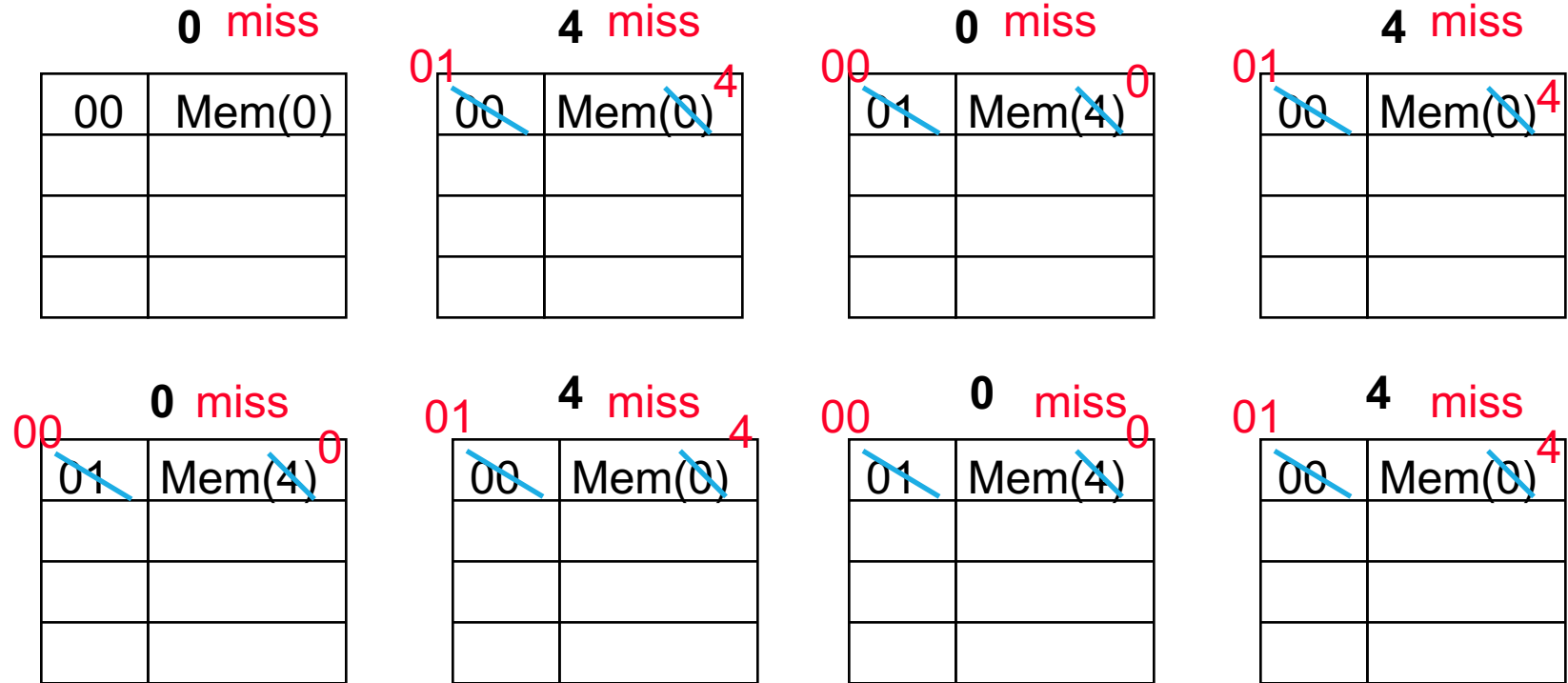

4




# Another Reference String Mapping

❑ Consider the main memory word reference string

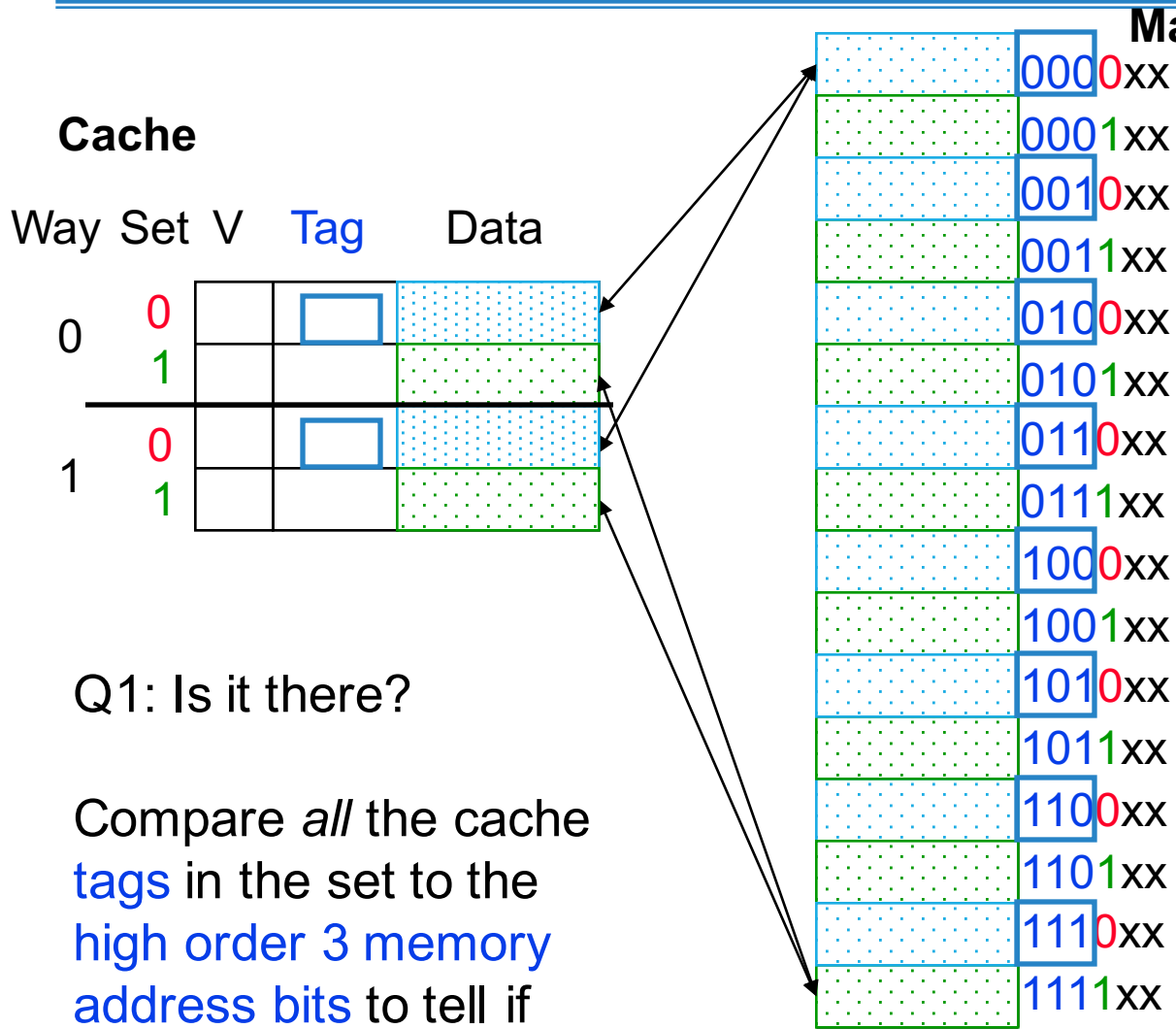
Start with an empty cache - all blocks initially marked as not valid  
 0 4 0 4 0 4 0 4



● 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

# EX: 2-way set associate

- Consider the main memory word reference string, how many misses?

Start with an empty cache

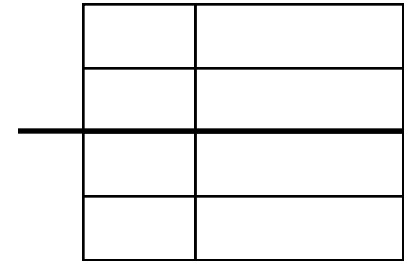
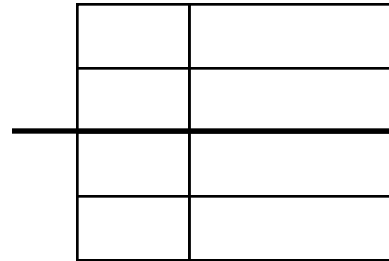
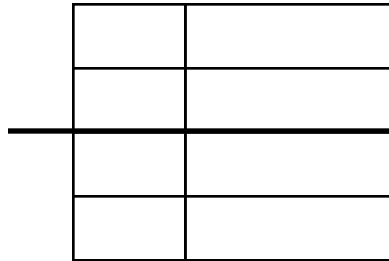
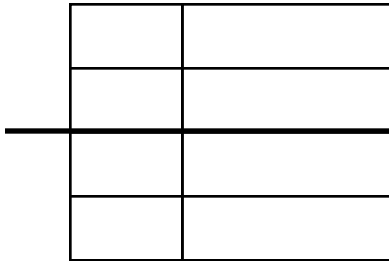
0 4 0 4 0 4 0 4

0

4

0

4



# EX: 2-way set associate

- Consider the main memory word reference string, how many misses?

Start with an empty cache

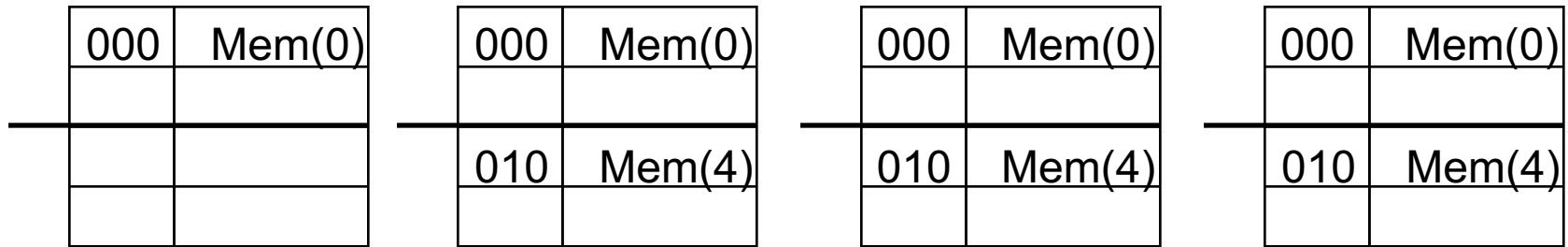
0 4 0 4 0 4 0 4

0 miss

4 miss

0 hit

4 hit

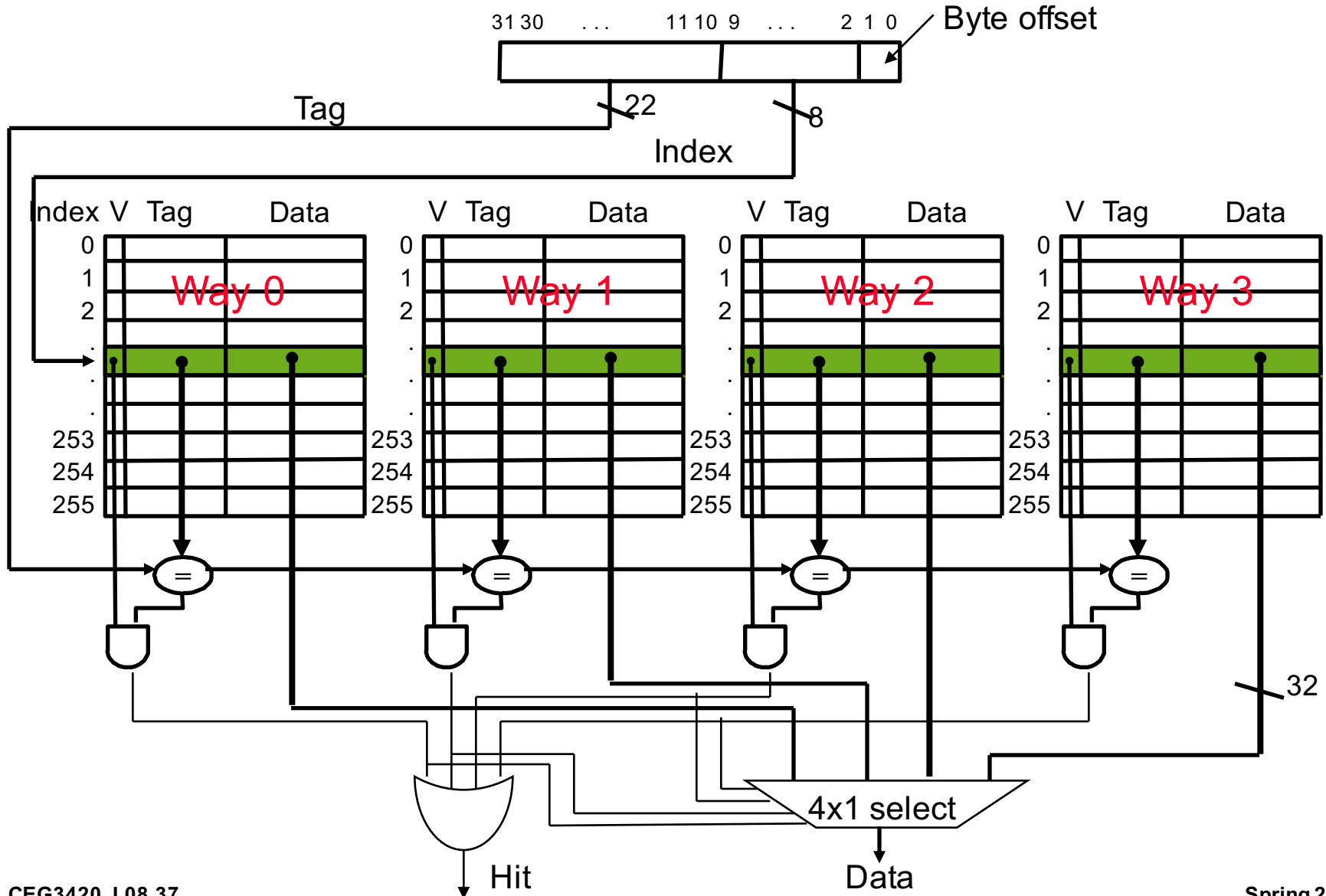


- 8 requests, 2 misses

- 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<sup>8</sup> = 256 sets each with four ways (each with one block)



# Range of Set Associative Caches

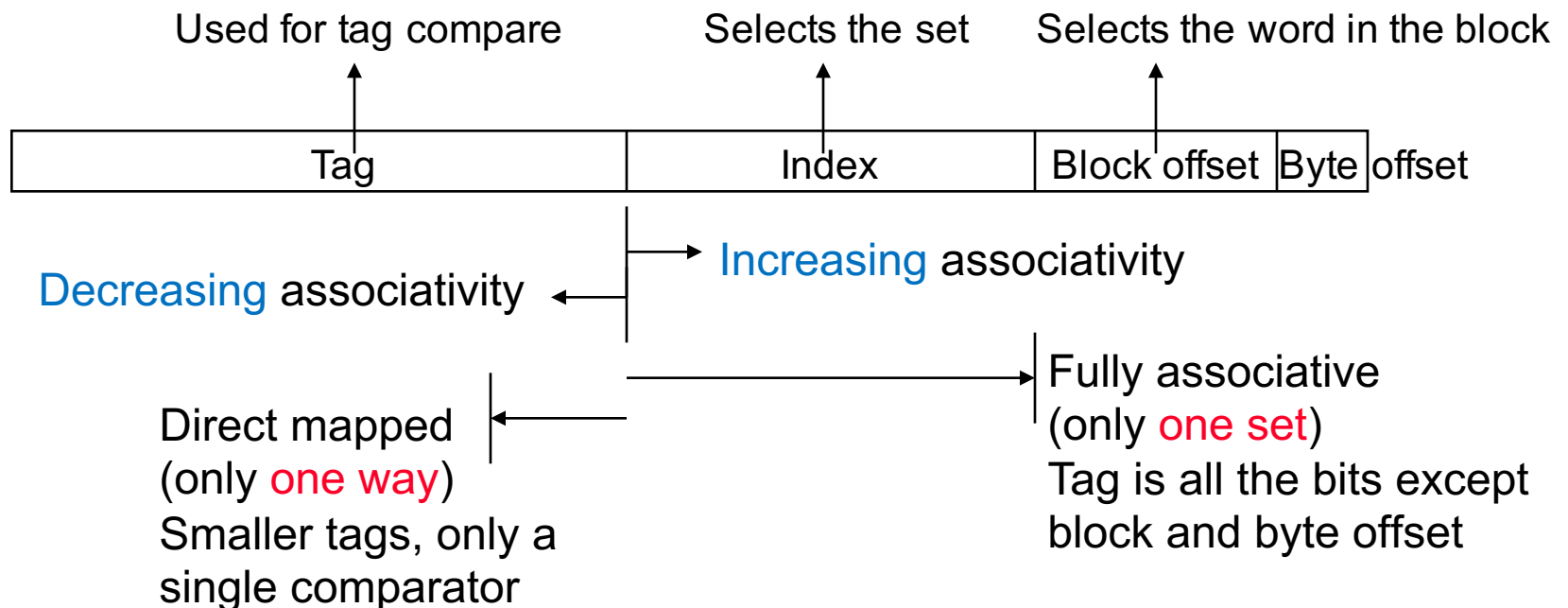
---

- 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 of 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



# 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 of 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



# 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



# 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,  $CPI_{ideal}$  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

$$\begin{aligned} CPI_{stalls} &= 2 + (.02 \times 25 + .005 \times 100) \\ &\quad + (.36 \times .04 \times 25 + .36 \times .005 \times 100) = 3.54 \\ &\quad \text{(as compared to 5.44 with no L2\$)} \end{aligned}$$