

---

# Chapter 6: Memory

Ngo Lam Trung, Pham Ngoc Hung

[with materials from *Computer Organization and Design*, MK  
and M.J. Irwin's presentation, PSU 2008]

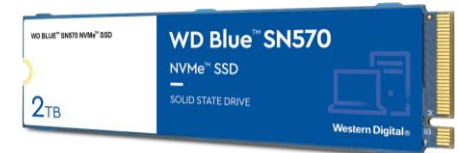
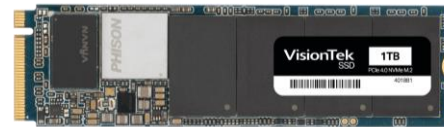
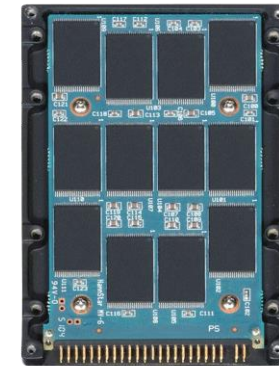
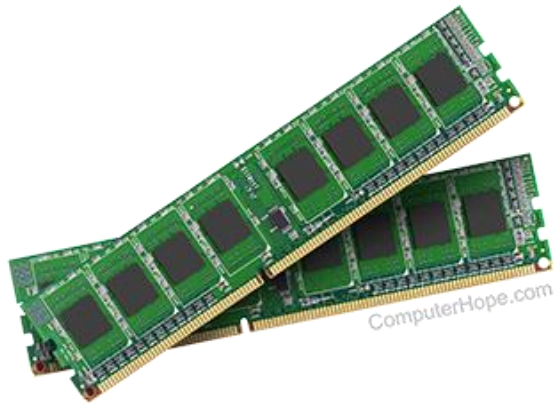
# Content

---

- ❑ Memory hierarchy
- ❑ Principal of locality
- ❑ Cache
- ❑ Virtual memory

# Memory

- ❑ Memory: where data are stored.



*Why is memory critical to performance?*

# Memory technology (2012)

---

- ❑ Static RAM (SRAM)

- ❑ 0.5ns – 2.5ns, \$500 – \$1000 per GB

- ❑ Dynamic RAM (DRAM)

- ❑ 50ns – 70ns, \$10 – \$20 per GB

- ❑ Flash memory

- ❑ 5,000ns – 50,000ns, \$0.75 – \$1 per GB

- ❑ Magnetic memory

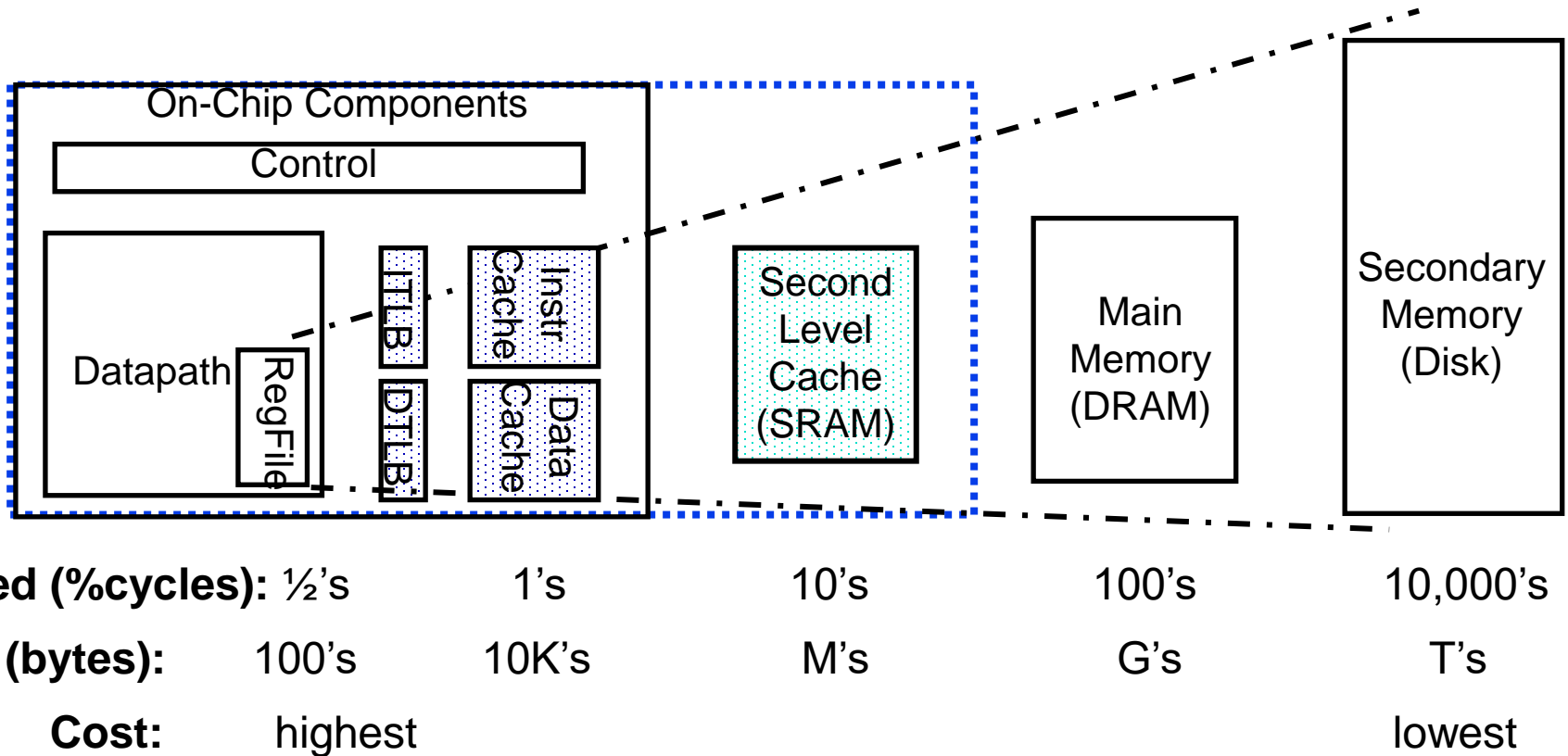
- ❑ 5,000,000ns – 20,000,000ns, \$0.05 – \$0.1 per GB

- ❑ Fact:

- ❑ Large memories are slow

- ❑ Fast memories are small (and expensive)

# A Typical Memory Hierarchy



- ❑ How to get an ideal memory
  - ❑ As fast as SRAM
  - ❑ As cheap as disk?

# The Memory Hierarchy: Locality Principal

## ❑ C program

```
int x[1000], temp;
for (i = 0; i < 999; i++)
    for (j = i+1; j < 1000; j++)
        if (x[i] < x[j])
        {
            temp = x[i];
            x[i] = x[j];
            x[j] = temp;
        }
```

Data memory at location of temp and x are accessed multiple times

Instruction memory at location of the two **for** loops are used repeatedly

# The Memory Hierarchy: Locality Principal

## ❑ Temporal Locality (locality in time)

- ❑ If a memory location is referenced then it will tend to be referenced again soon

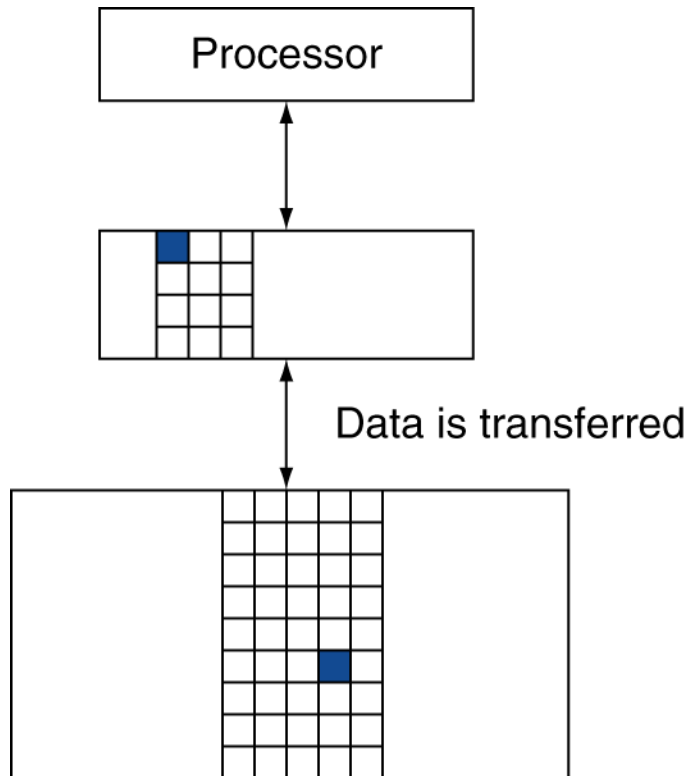
⇒ Keep **most recently accessed** data items closer to the processor

## ❑ 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

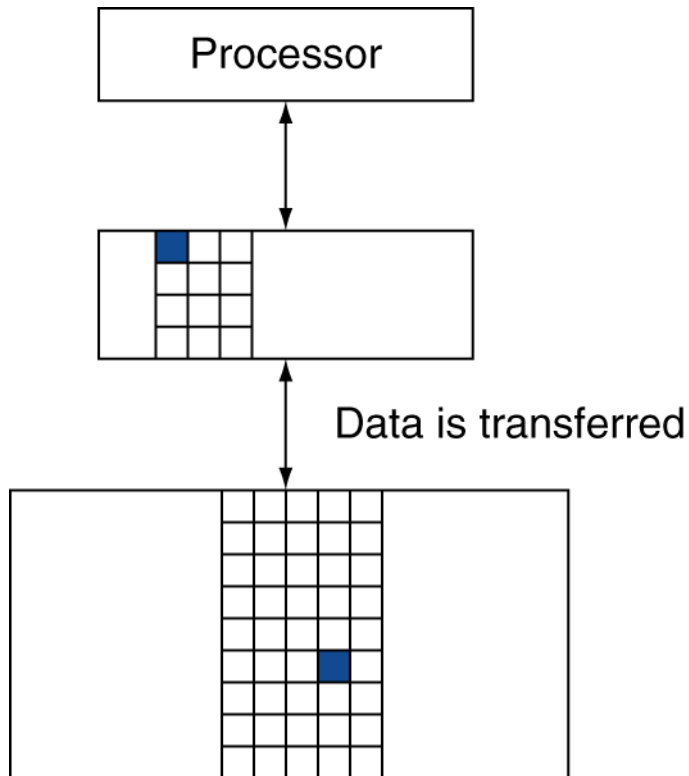
# Hierarchical memory access



- ❑ Data are stored in multiple levels.
  - ❑ High level: fast but small
  - ❑ Low level: slow but large
- ❑ Data are transferred in units of block (of multiple words) between levels, through the hierarchy.
- ❑ Frequently used data are stored closer to processor.



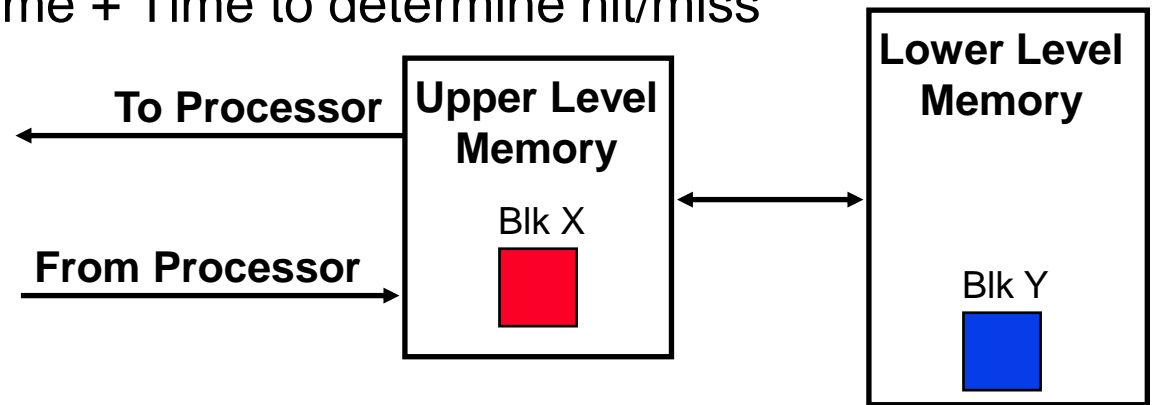
# Hierarchical memory access



- ❑ Associative data access:
  - ❑ Processor access data in lower level
  - ❑ Data transfer from lower level to processor via upper level(s)
- ❑ 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  
 $= 1 - \text{hit ratio}$
  - ❑ Then accessed data supplied from upper level

# The Memory Hierarchy: Terminology

- ❑ **Hit**: data is in some block in the upper level (**Blk X**)
  - ❑ **Hit Rate**: fraction of memory accesses found in upper level
  - ❑ **Hit Time**: Time to access the upper level which consists of
    - RAM access time + Time to determine hit/miss

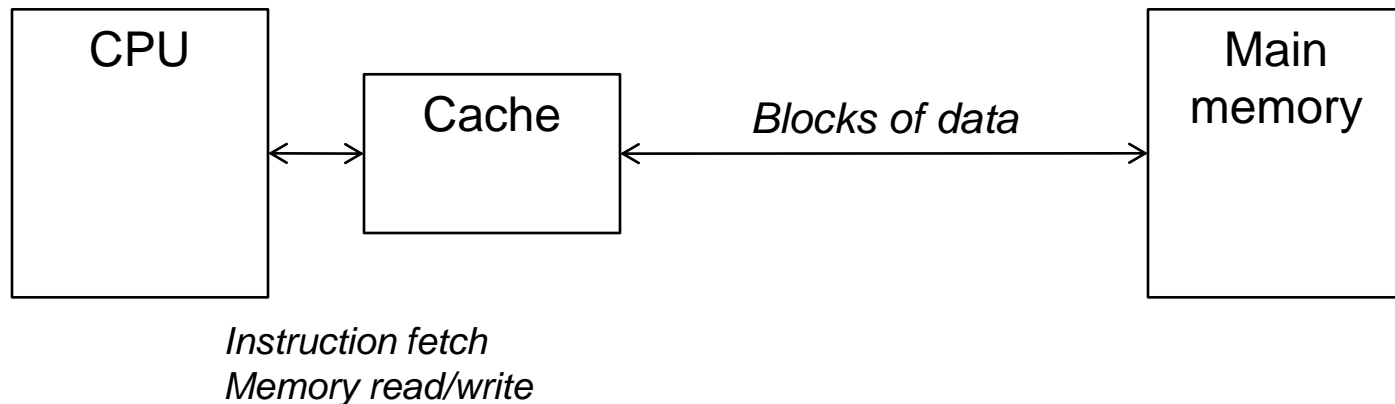


- ❑ **Miss**: data is not in the upper level so needs to be retrieve from a block in the lower level (**Blk Y**)
  - ❑ **Miss Rate** =  $1 - (\text{Hit Rate})$
  - ❑ **Miss Penalty**: Time to bring in a block from the lower level and replace a block in the upper level with it + Time to deliver the block to the processor
  - ❑ **Hit Time**  $\ll$  **Miss Penalty**

# Cache

---

- ❑ The memory hierarchy between the processor and main memory
  - ❑ CPU fetch instructions and data from cache, if found (**cache hit**) → fast access.
  - ❑ If not found (**cache miss**) → load a block from main memory into cache, then access in cache → slower access time (miss penalty)



# Cache Basics

---

- ❑ CPU needs to access a data item in memory

➔ Two questions to answer (in hardware):

- ❑ Q1: How do CPU know if the data item is in the cache?
- ❑ Q2: If it is, how does CPU 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):  
 **$(\text{block address}) \bmod (\# \text{ of blocks in the cache})$**
- ❑ The **tag** field: associated with each cache block that contains the address information (the upper portion of the address) required to identify the block (to answer Q1)
- ❑ The valid bit: if there is data in the block or not

# 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

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

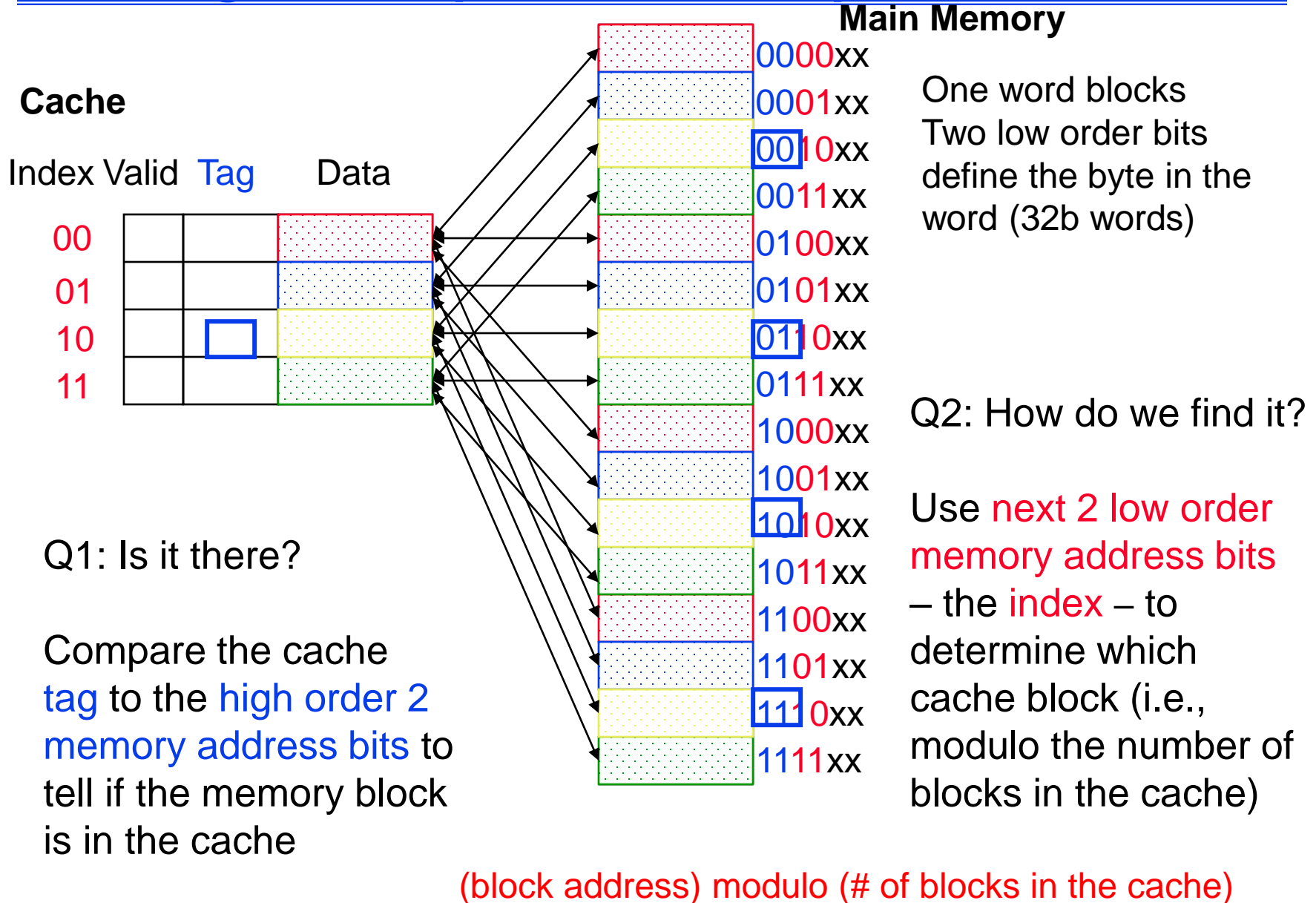
One word blocks  
Two low order bits define the byte in the word (32b words)

Q2: How does CPU 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



# 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

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

15

❑ 8 requests, 6 misses

❑ What if we repeatedly request 1,000,000 times



# Cache performance

---

- ❑ Given a MIPS CPU running a program with the miss rate of instruction cache is 2% and the miss rate of data cache is 4%. The processor has CPI of 2 without any memory stalls and the miss penalty is 100 cycles for all misses
- ❑ Determine how much faster that processor would run with a perfect cache that never missed. Assume the frequency of all loads and stores is 36%.
- ❑ Solution:

# Cache performance

---

- ❑ Given a MIPS CPU running a program with the miss rate of instruction cache is 2% and the miss rate of data cache is 4%. The processor has CPI of 2 without any memory stalls and the miss penalty is 100 cycles for all misses.
- ❑ Determine how much faster that processor would run with a perfect cache that never missed. Assume the frequency of all loads and stores is 36%.
- ❑ Solution:
- ❑ Given instruction count  $I$

$$\text{Instruction miss cycles} = I * 2\% * 100 = 2.00 * I$$

$$\text{Data miss cycles} = I * 36\% * 4\% * 100 = 1.44 * I$$

- ❑ Total mem-stall cycles:  $2.00 I + 1.44 I = 3.44 I$ .

$$\frac{\text{CPU time with stalls}}{\text{CPU time with perfect cache}} = \frac{I \times \text{CPI}_{\text{stall}} \times \text{Clock cycle}}{I \times \text{CPI}_{\text{perfect}} \times \text{Clock cycle}} = \frac{\text{CPI}_{\text{stall}}}{\text{CPI}_{\text{perfect}}} = \frac{5.44}{2} = 2.72$$

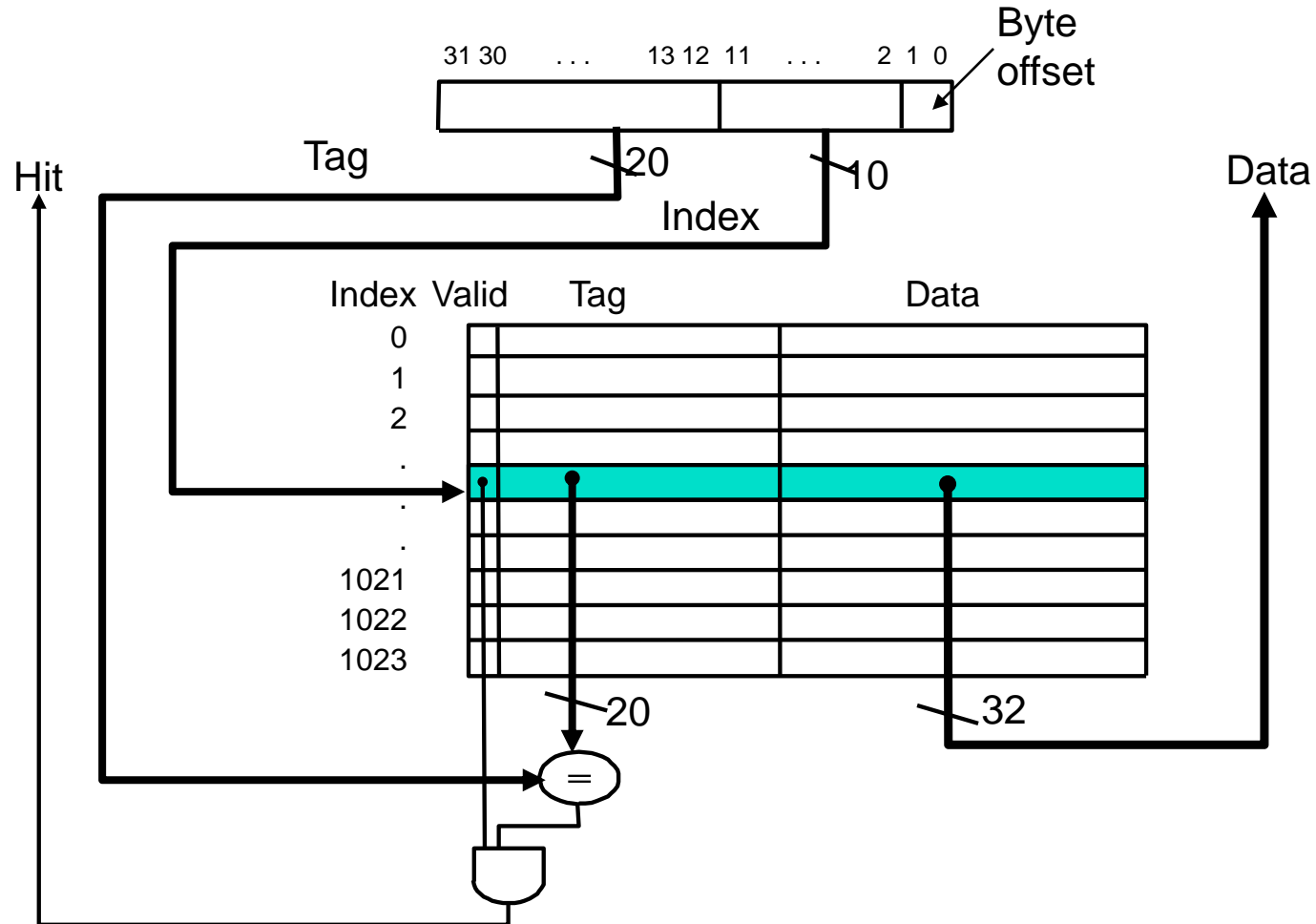
# Cache performance

---

- ❑ Given a MIPS CPU running a program with the miss rate of instruction cache is 2% and the miss rate of data cache is 4%. The processor has CPI of 2 without any memory stalls and the miss penalty is 100 cycles for all misses
- ❑ Determine how much faster that processor would run with a perfect cache that never missed. Assume the frequency of all loads and stores is 36%.
- ❑ What is the speed up if the CPU now has faster CPI of 1 (instead of 2)?

# MIPS Direct Mapped Cache Example

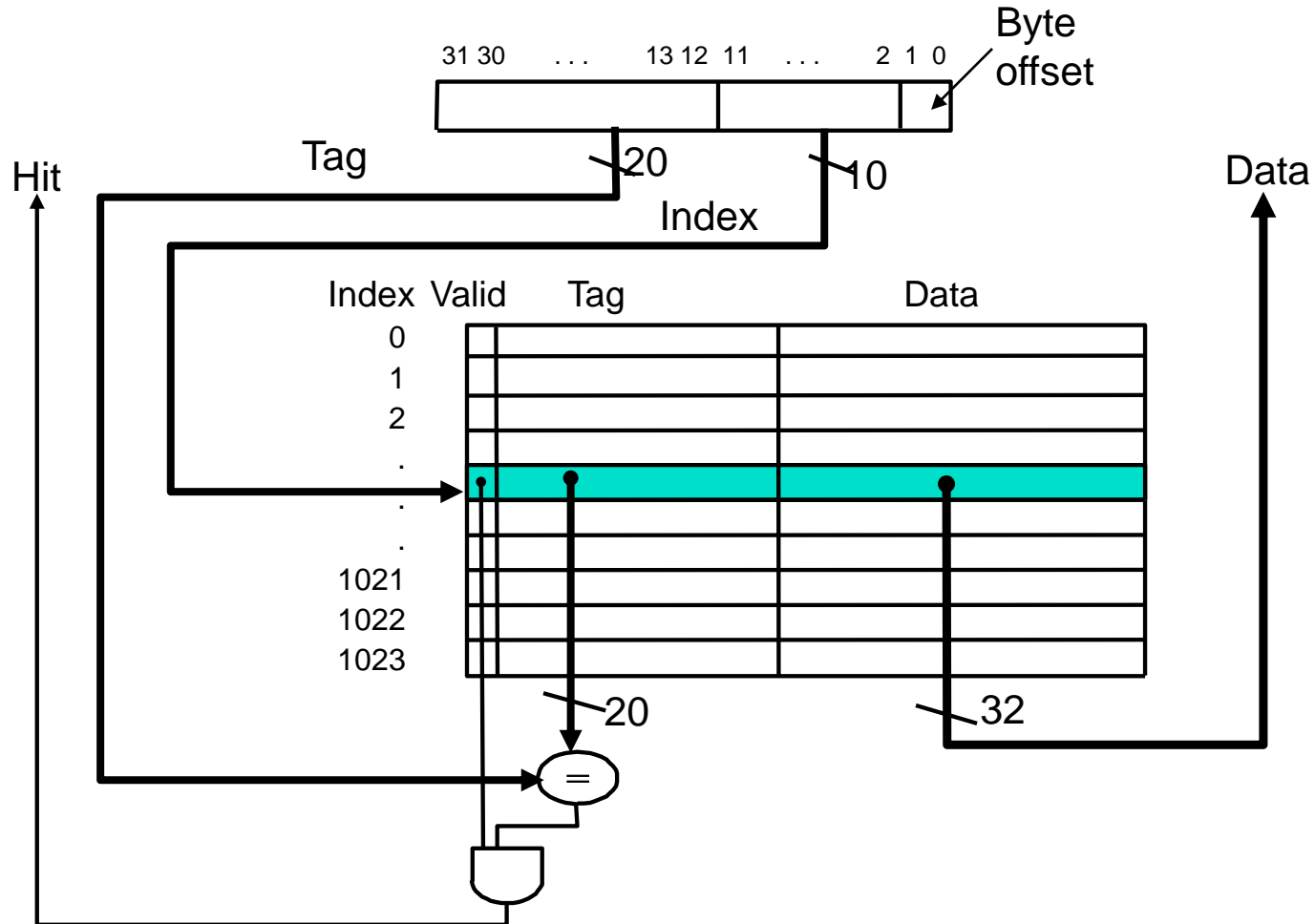
- ❑ One-word blocks, cache size = 1K words (or 4KB)



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

# MIPS Direct Mapped Cache Example

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



*Calculate the total size of this cache in Kilobits*

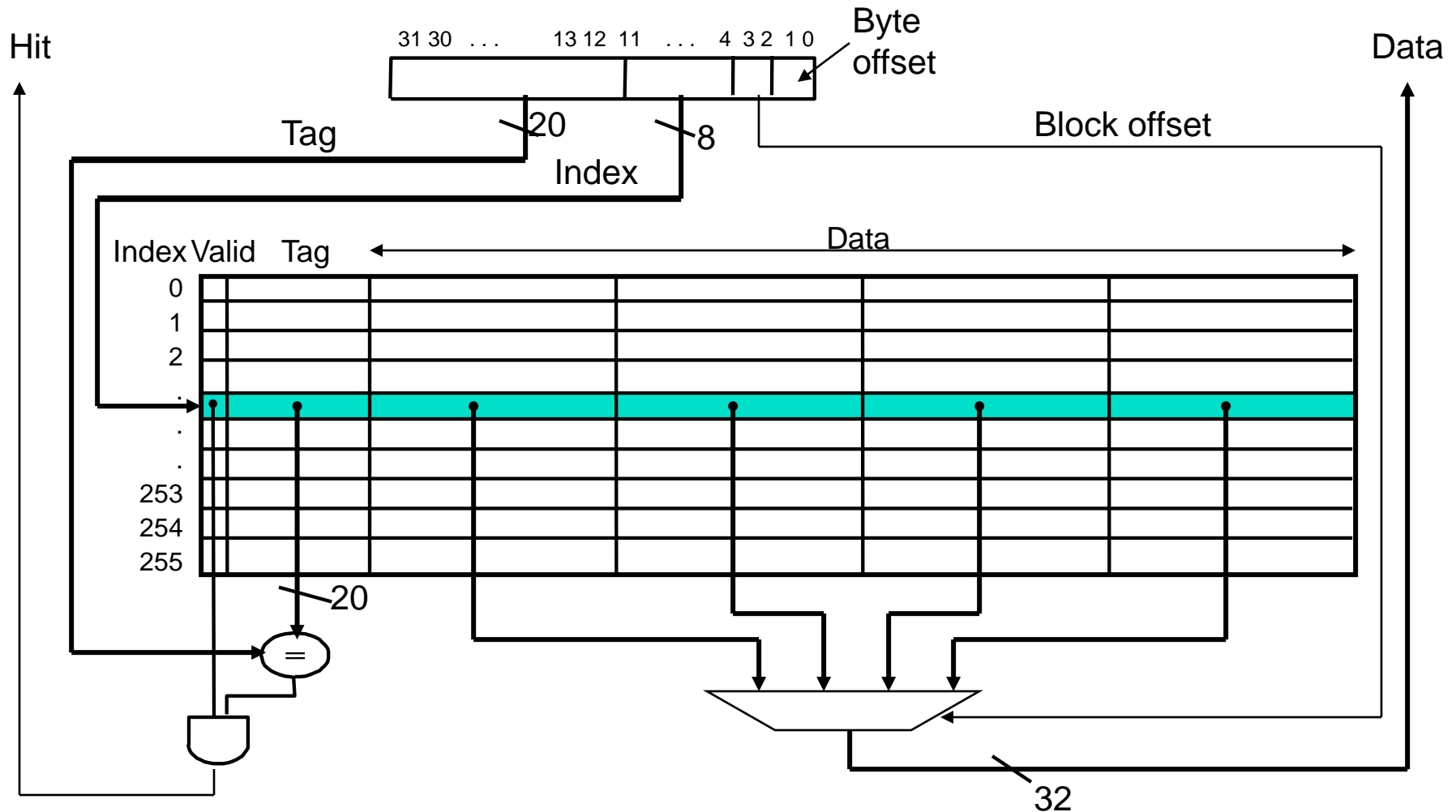
## Exercise

---

- ❑ How many total bits are required for a direct-mapped cache with 16 KiB of data and 1-word blocks, assuming a 32-bit address?

# 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**

<del>00</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**

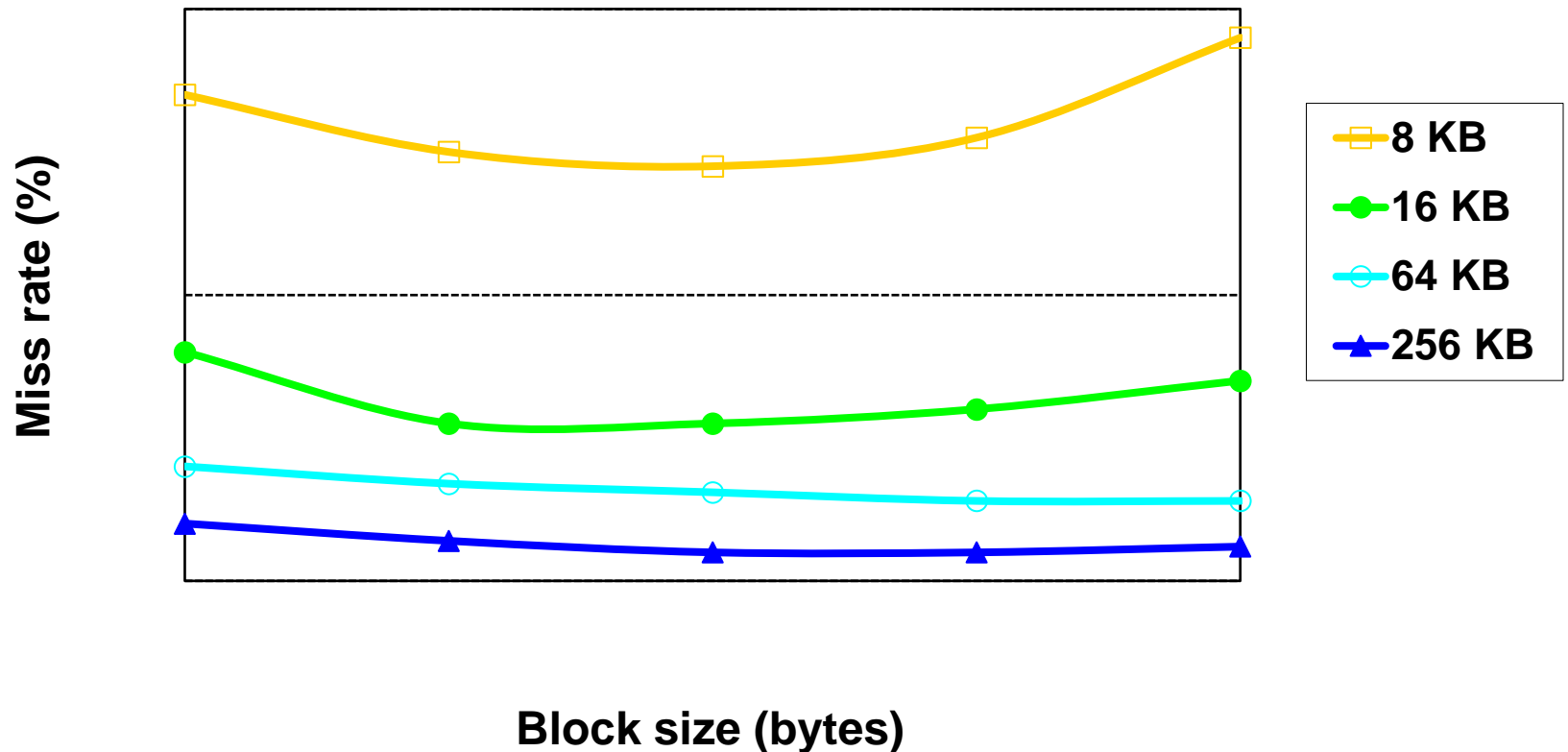
01	Mem(5)	Mem(4)
00	Mem(3)	Mem(2)

**15 miss**

<del>01</del>	<del>Mem(5)</del>	<del>Mem(4)</del>
<del>00</del>	<del>Mem(3)</del>	<del>Mem(2)</del>

- 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?
- ❑ The total number of bits in a direct-mapped cache is then
$$2^n \times (\text{block size} + \text{tag field size} + \text{valid field size})$$
- ❑ How many total bits are required for a direct mapped cache with 16KB of data and 4-word blocks assuming a 32-bit address?

## Exercise

---

- ❑ How many total bits are required for a direct-mapped cache with 16 KiB 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

# Sources of Cache Misses

---

## ❑ Compulsory (cold start, first reference):

- ❑ First access to a block.
- ❑ We cannot do much on this.
- ❑ Solution: increase block size (but also increases miss penalty).

## ❑ 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)

# Handling Cache Misses (Single Word Blocks)

## ❑ Read misses (I\$ and D\$)

- ❑ **stall** the pipeline, fetch the block from the next level in the memory hierarchy, install it in the cache and send the requested word to the processor, then let the pipeline resume

## ❑ Write misses (D\$ only)

1. **stall** the pipeline, fetch the block from next level in the memory hierarchy, install it in the cache (which may involve having to evict a dirty block if using a write-back cache), write the word from the processor to the cache, then let the pipeline resume

or

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

or

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

# Multiword Block Considerations

---

## ❑ Read misses (I\$ and D\$)

- ❑ Processed the same as for single word blocks – a miss returns the entire block from memory
- ❑ Miss penalty grows as block size grows
  - **Early restart** – processor resumes execution as soon as the requested word of the block is returned
  - **Requested word first** – requested word is transferred from the memory to the cache (and processor) first
- ❑ **Nonblocking cache** – allows the processor to continue to access the cache while the cache is handling an earlier miss

## ❑ Write misses (D\$)

- ❑ If using write allocate must *first* fetch the block from memory and then write the word to the block (or could end up with a “garbled” block in the cache (e.g., for 4-word blocks, a new tag, one word of data from the new block, and three words of data from the old block))



# Reducing Cache Miss Rates #1

---

→ Allow more flexible block placement

- ❑ **Direct mapped cache:** a memory block maps to exactly one cache block
- ❑ **Fully associative cache** allow a memory block to be mapped to *any* cache block
- ❑ 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

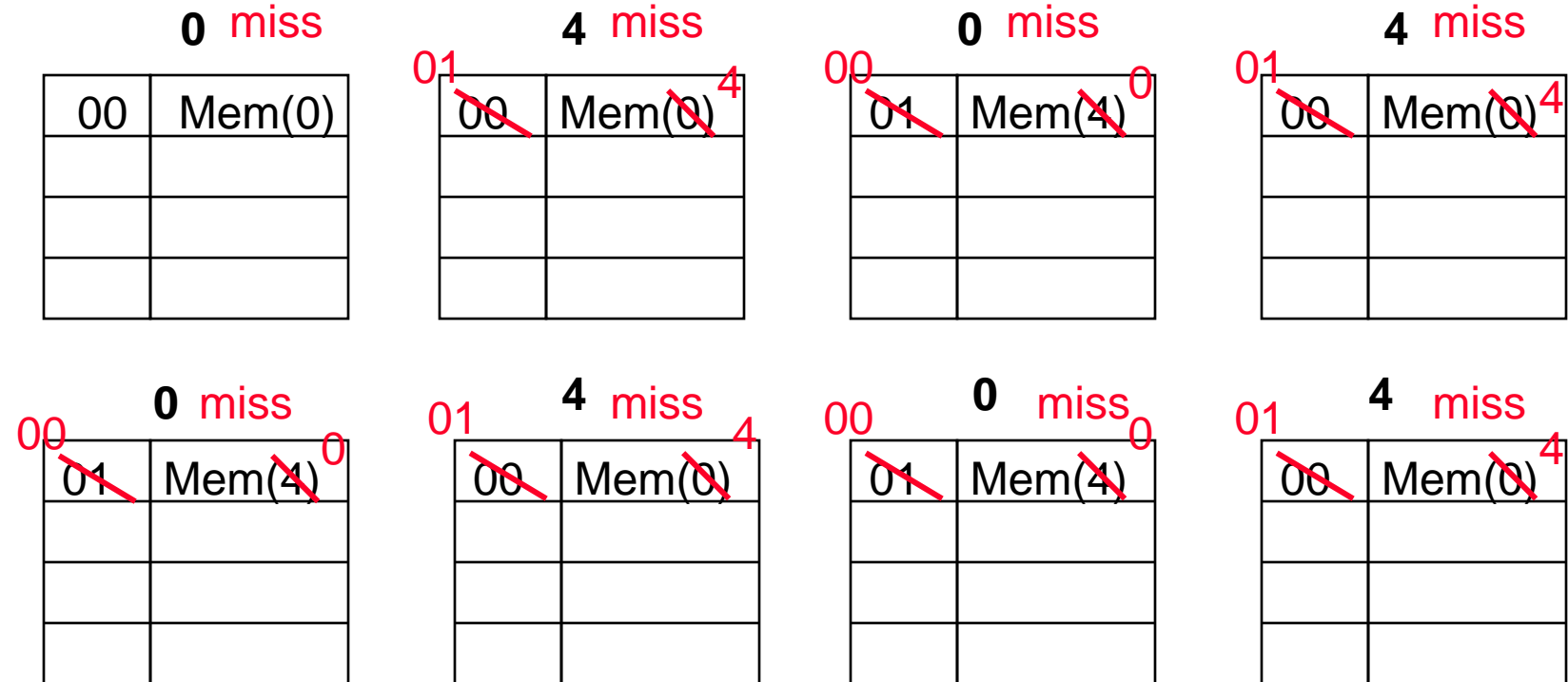

4


# Another Reference String Mapping

❑ Consider the main memory word reference string

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

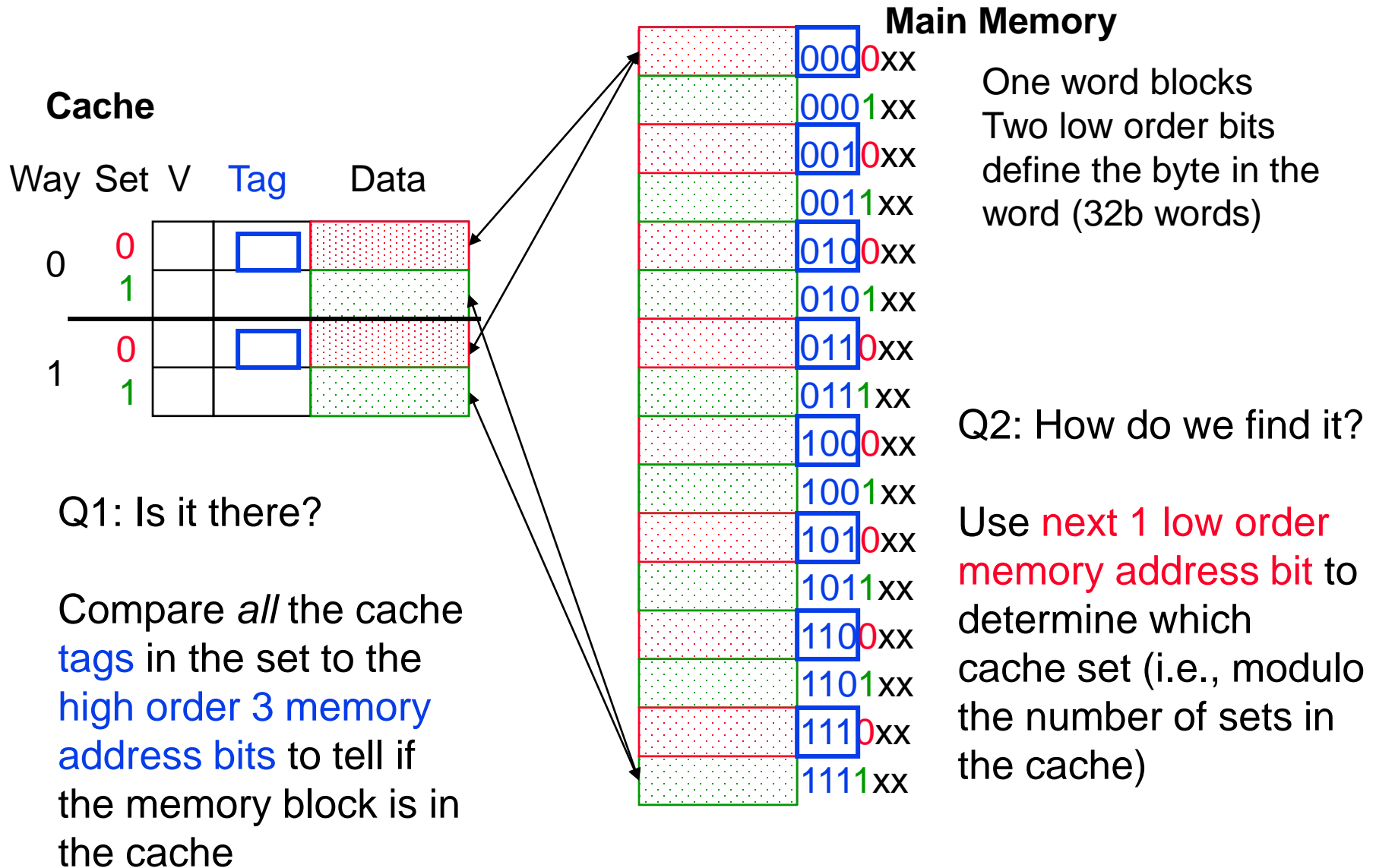
0 4 0 4 0 4 0 4



❑ 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



# 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

4 miss

0 hit

4 hit

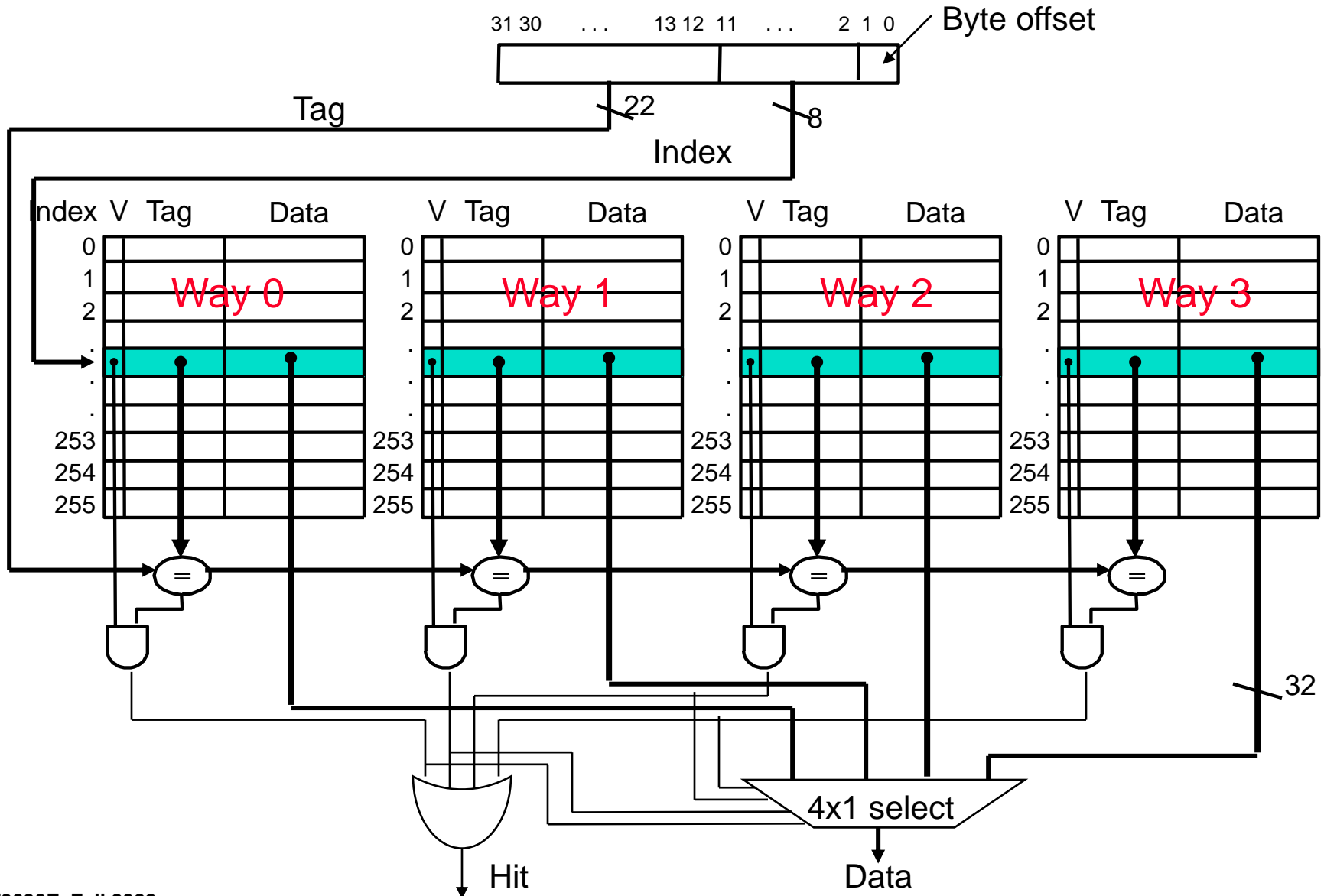
000	Mem(0)	000	Mem(0)	000	Mem(0)	000	Mem(0)
		010	Mem(4)	010	Mem(4)	010	Mem(4)

❑ 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!

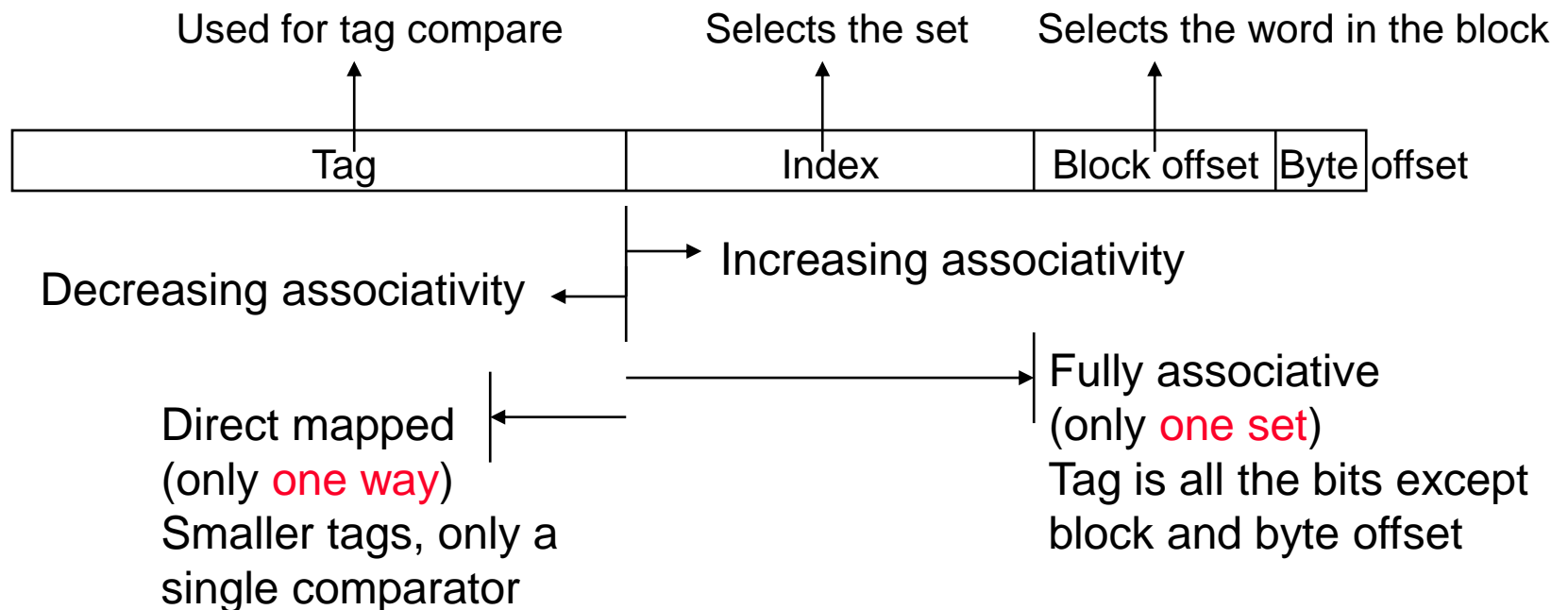
# Four-Way Set Associative Cache

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



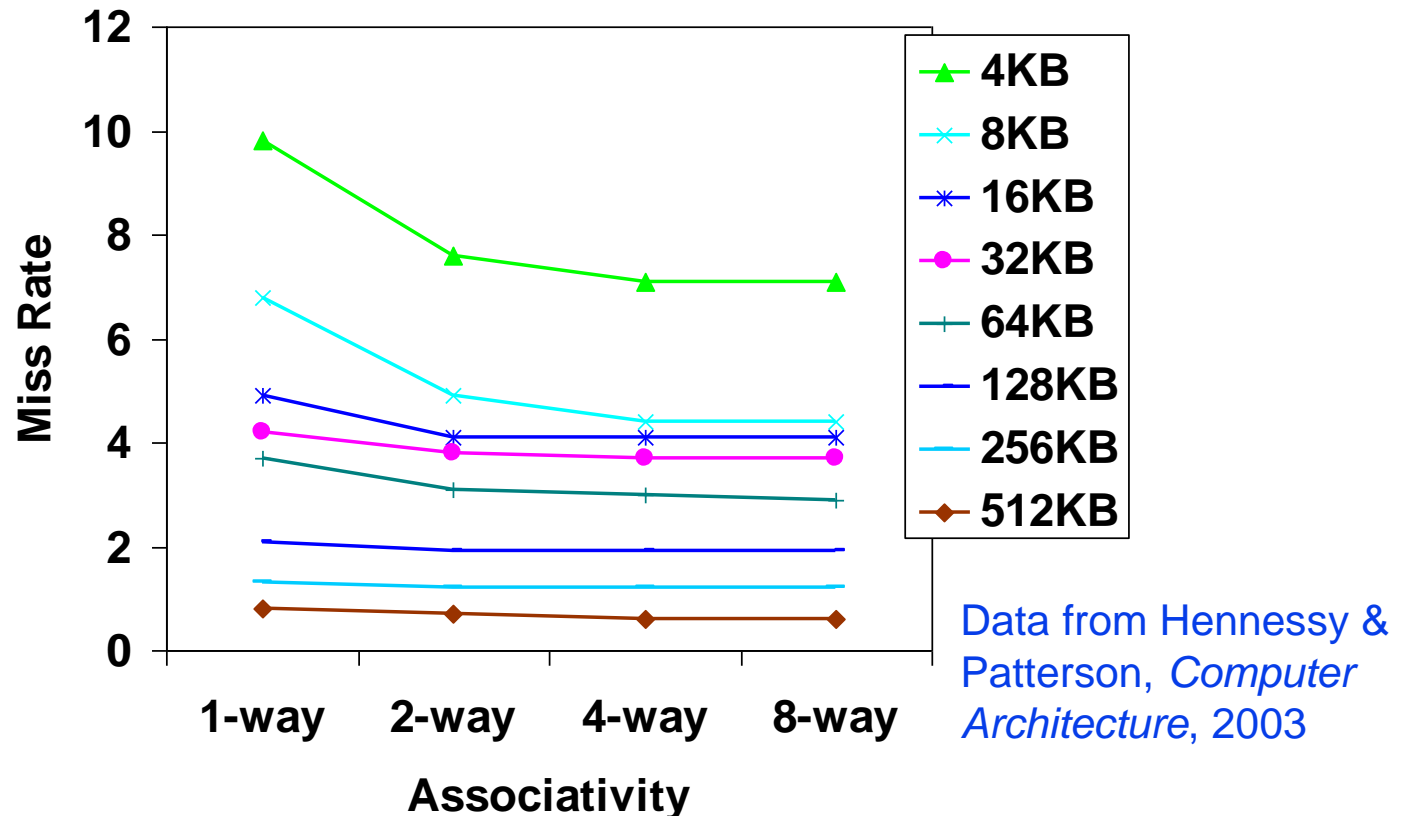
# Range of Set Associative Caches

- ❑ For a fixed size cache, increase of the number of blocks per set results in decrease of the number of sets



# 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



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



# Block replacement

---

- ❑ Cache miss: a new block is loaded to cache, it will replace an old block
- ➔ Which block should be replaced?
- ❑ Direct-mapped cache: exactly one choice
- ❑ Associative cache: one of multiple blocks in the set must be selected
  - ❑ ➔ LRU scheme: (least recently used) block that has been unused the longest time is selected for replacement.
  - ❑ Mechanism for relative last time used tracking is necessary.

# LRU block replacement

❑ Consider the main memory word reference string

Start with an empty cache - all blocks initially marked as not valid  
0 4 2 4 0 0 0 4

Last  
used

**0 miss**

x	000	Mem(0)

**4 miss**

	000	Mem(0)
x	010	Mem(4)

**2 miss**

x	001	Mem(2)
	010	Mem(4)

**4 hit**

	001	Mem(2)
x	010	Mem(4)

Last  
used

**0 miss**

x	000	Mem(0)
	010	Mem(4)

**0 hit**

x	000	Mem(0)
	010	Mem(4)

**0 hit**

x	000	Mem(0)
	010	Mem(4)

**4 hit**

	000	Mem(0)
x	010	Mem(4)

## Reducing Cache Miss Rates #2

---

### → Use multiple levels of caches

- ❑ Very costly in 1990s: US\$100000 or above
- ❑ Common in 2020s: ~US\$500 machines
- ❑ Normally a **unified** L2 cache (holding both instructions and data, for each core) and a unified L3 cache shared for all cores

# 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 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  $\gg$  than the global miss rate

## Example

---

- ❑ Given a processor with a base CPI of 1.0 and clock rate of 4 GHz. Main memory access time is 100 ns.
  - ❑ All data references are hit in primary cache (L1).
  - ❑ Instruction miss rate of 2% in primary cache (L1).
- ❑ A new L2 is added
  - ❑ Access time from L1 to L2 is 5 ns.
  - ❑ Instruction miss rate (to main memory) reduced to 0.5%.
- ❑ What is speed-up after adding the L2?

# Exercise

---

- ❑ Given a CPU with 32 bits address and the below word reference string.

3, 180, 43, 2, 191, 88, 190, 14, 181, 44, 186, 253

- ❑ Identify the binary address, tag field, block index field, and hit ratio in the following cases.
  - ❑ The CPU has direct-mapped cache of 16 one-word blocks.
  - ❑ The CPU has direct-mapped cache of 8 two-word blocks.

## Exercise

---

- ❑ Given a CPU with 32 bits address and the below word reference string.

3, 180, 43, 2, 191, 88, 190, 14, 181, 44, 186, 253

- ❑ The CPU has direct-mapped cache with a total of 8 data words. Miss penalty is 25 cycles.
- ❑ Which of the following designs is optimal given the above reference string?
  - ❑ 8x one-word blocks, access time of 2 cycles
  - ❑ 4x two-word blocks, access time of 3 cycles.
  - ❑ 2x four-word blocks, access time of 5 cycles.