

# **Introduction to Computer Architecture**

## **Chapter 5**

### **Large and Fast: Exploiting Memory Hierarchy**

**Hyungmin Cho**

Department of Computer Science and Engineering  
Sungkyunkwan University

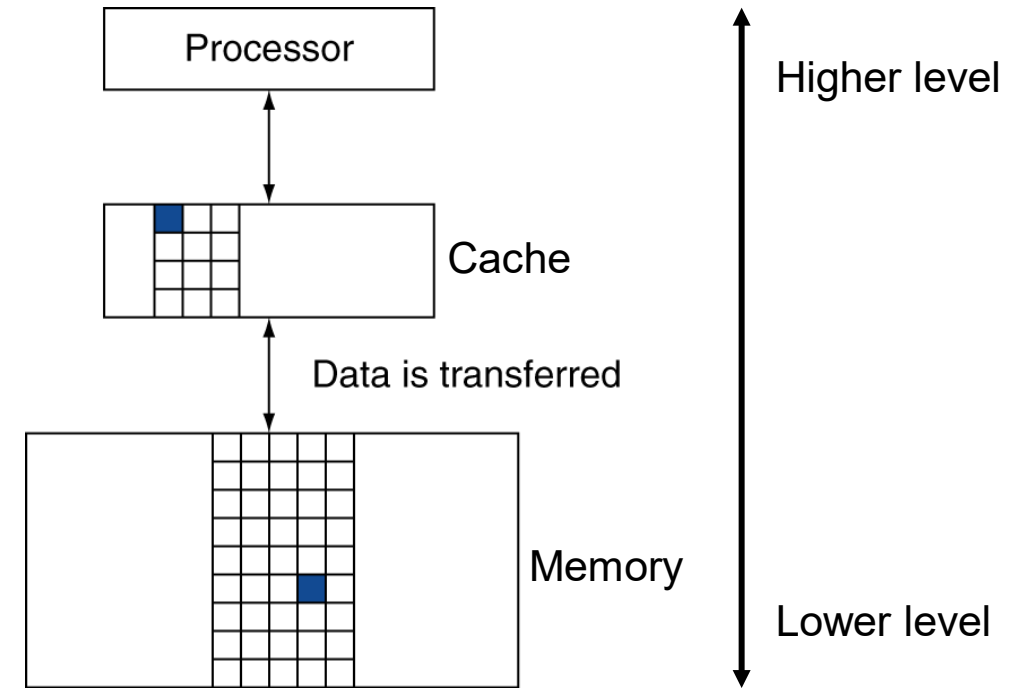
# Memory Technology

---

- Access latency
  - ❖ Static RAM (SRAM)
    - 0.5ns – 2.5ns (16KB~64KB)
    - 5ns – 10ns (1MB~4MB)
  - ❖ Dynamic RAM (DRAM)
    - 50ns – 70ns (GBs)
  - ❖ Flash storage
    - 5 – 50 $\mu$ s (TBs)
  - ❖ Magnetic disk
    - 5ms – 20ms (10+ TBs)

# Memory Hierarchy Levels

- **Cache:** small, fast memory near the core
- **Block (a.k.a. line):** unit of copying
  - ❖ May be multiple words
- If accessed data is present in cache: **Hit**
  - ❖ Hit ratio:  $\#hits / \#accesses$
- If accessed data is not present: **Miss**
  - ❖ **Miss:** block copied from lower level (**fill**)
    - Time taken: miss penalty
    - Miss ratio:  $\#misses / \#accesses = 1 - \text{hit ratio}$



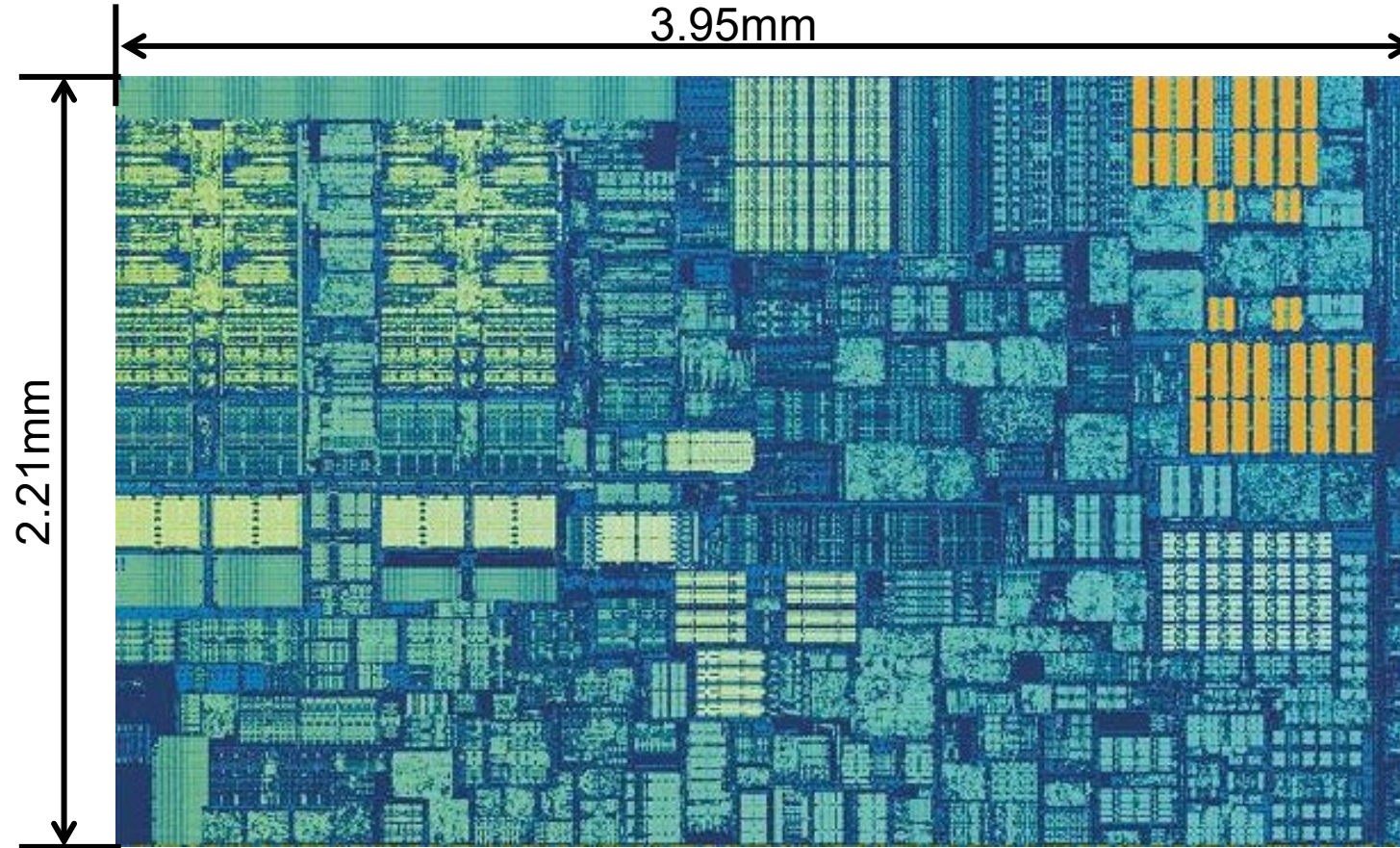
# Principle of Locality

---

- Programs access only a small portion of the memory at a given time
- **Temporal locality**
  - ❖ Items accessed recently are likely to be accessed again soon
  - ❖ e.g., instructions in a loop, index variables
- **Spatial locality**
  - ❖ Items near the recently accessed ones are likely to be accessed soon
  - ❖ e.g., sequential instruction access, array data

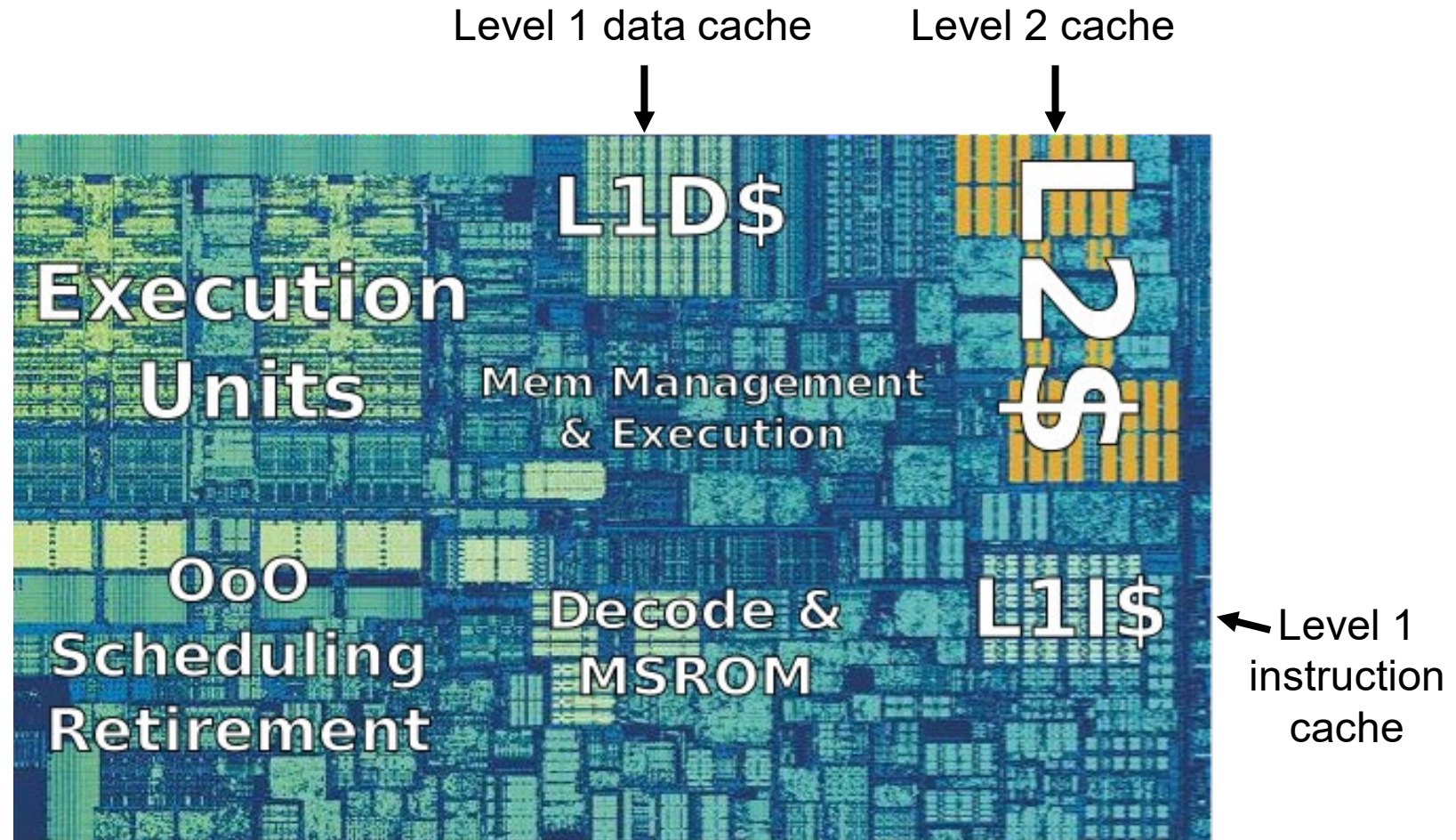
# CPU Die Image

- Intel “Skylake” core





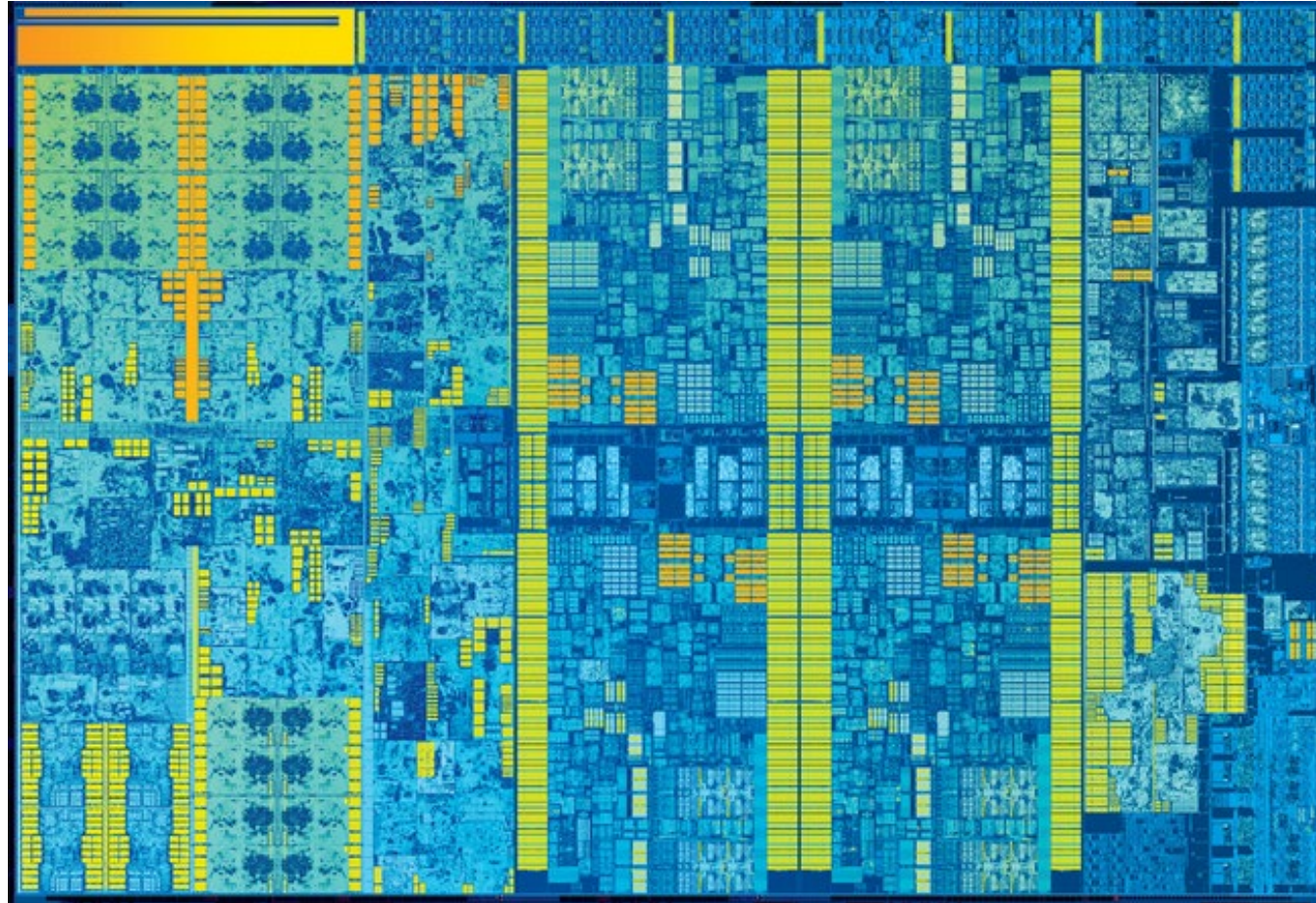
# CPU Die Image



# CPU Die Image

---

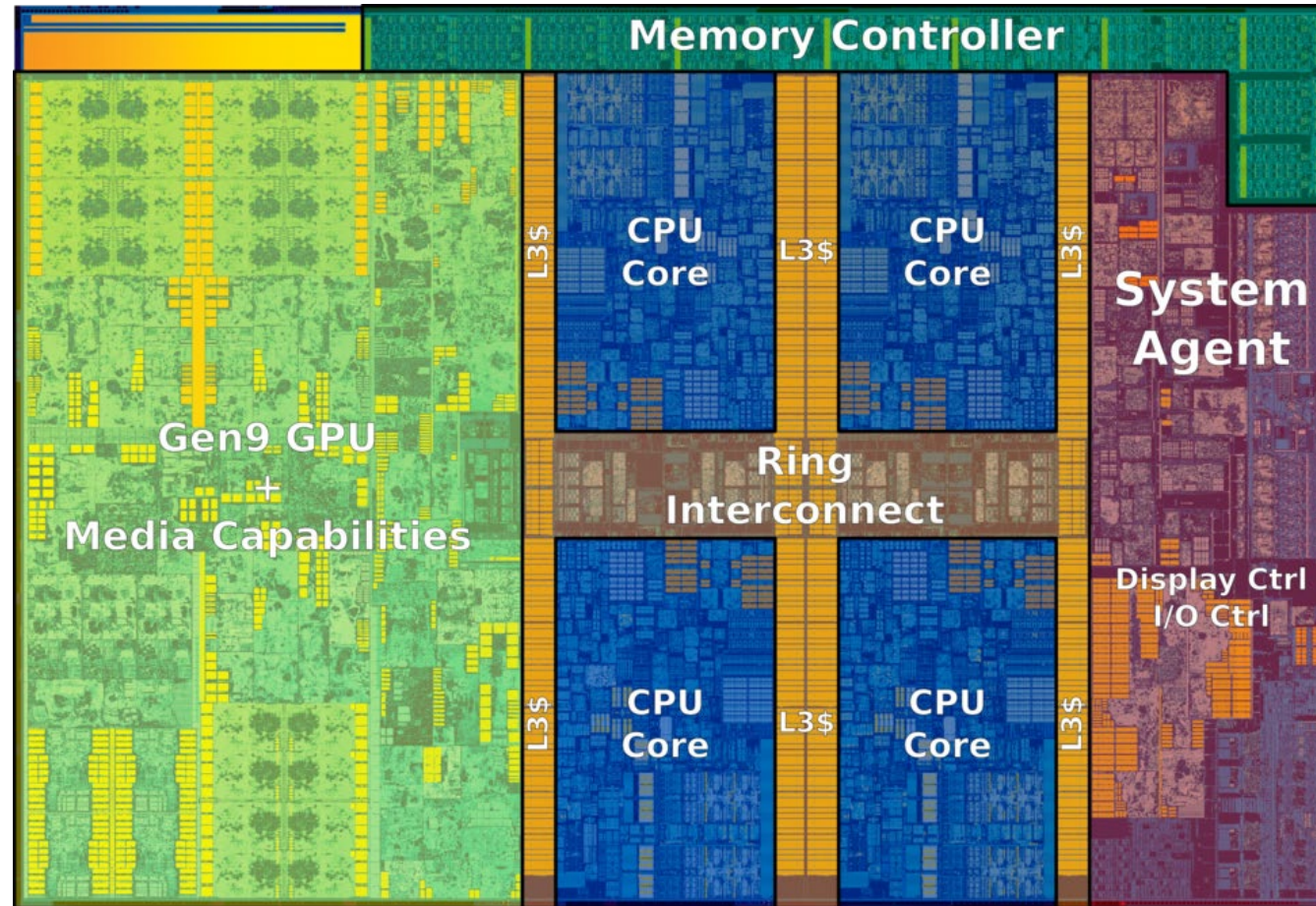
- Intel “Skylake” quad-core chip





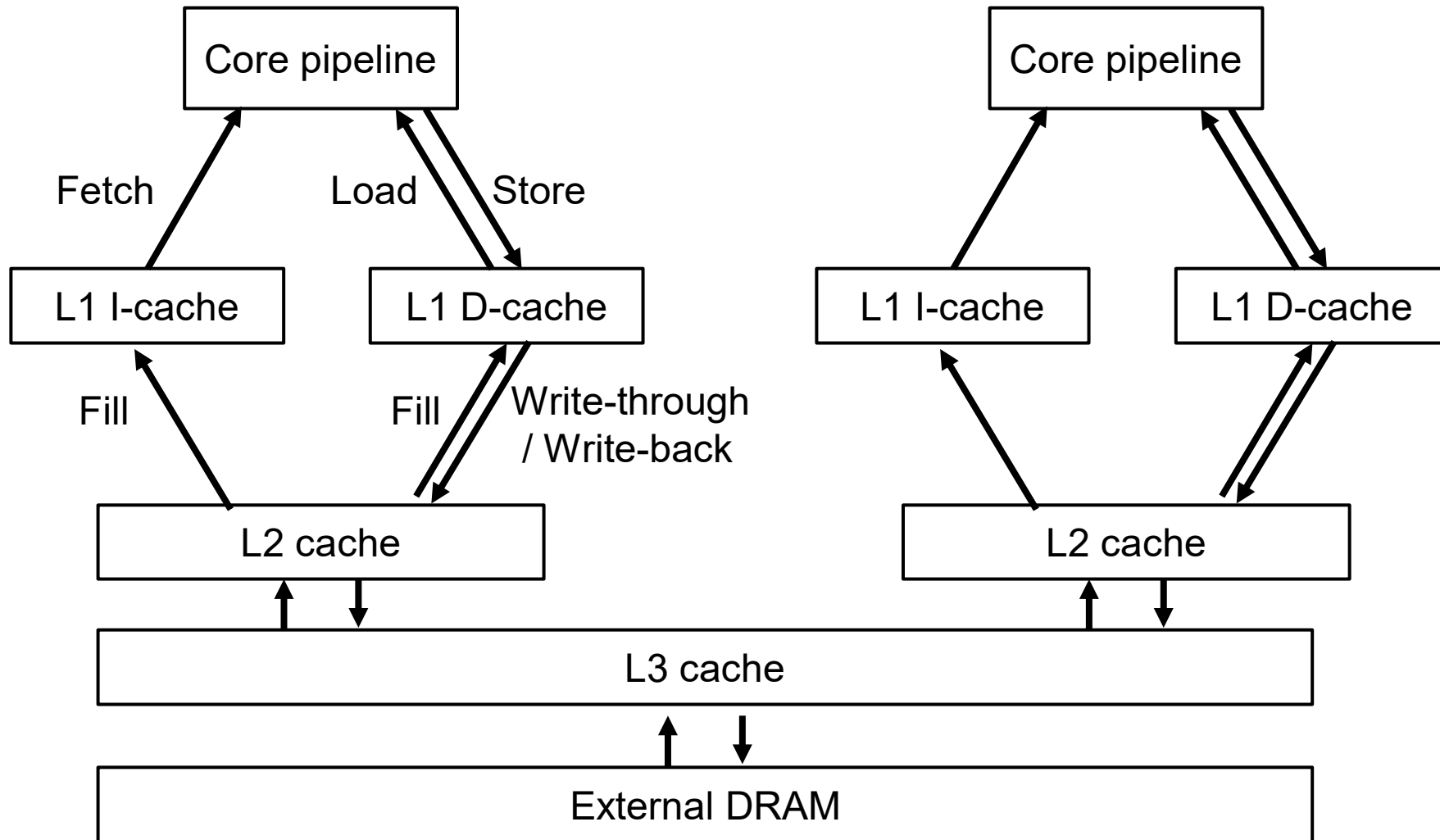
# CPU Die Image

- Intel “Skylake” quad-core CPU





# Typical Cache Hierarchy



# Cache Memory

- Given accesses  $X_1, \dots, X_{n-1}, X_n$

$X_4$
$X_1$
$X_{n-2}$
$X_{n-1}$
$X_2$
$X_3$

a. Before the reference to  $X_n$

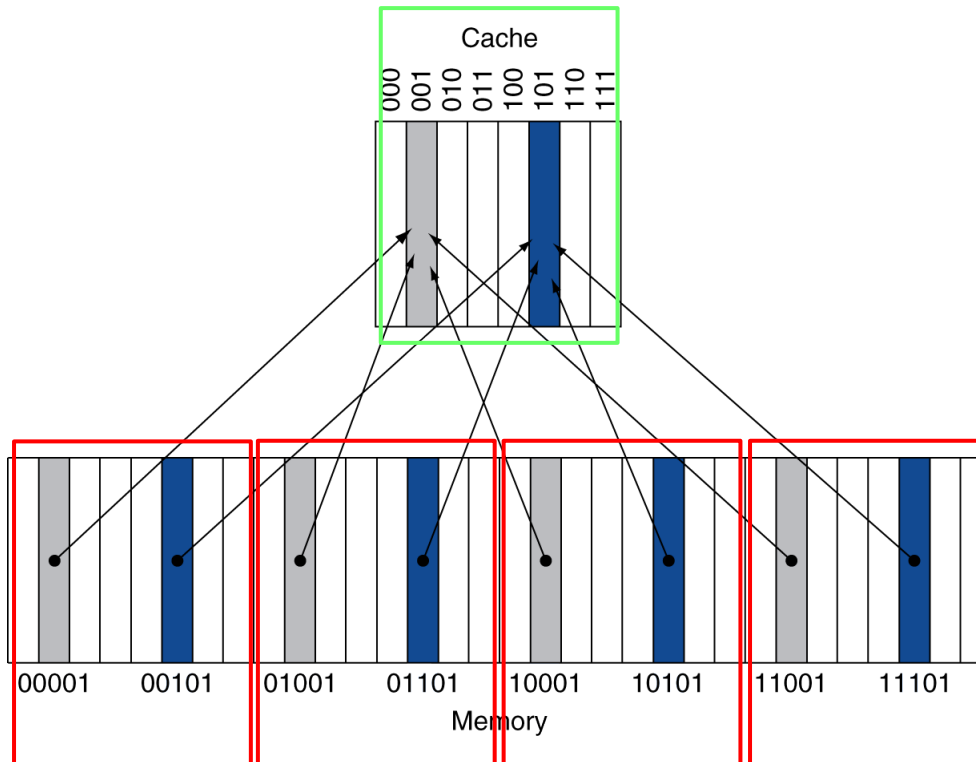
$X_4$
$X_1$
$X_{n-2}$
$X_{n-1}$
$X_2$
$X_n$
$X_3$

b. After the reference to  $X_n$

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

# Direct Mapped Cache

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



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



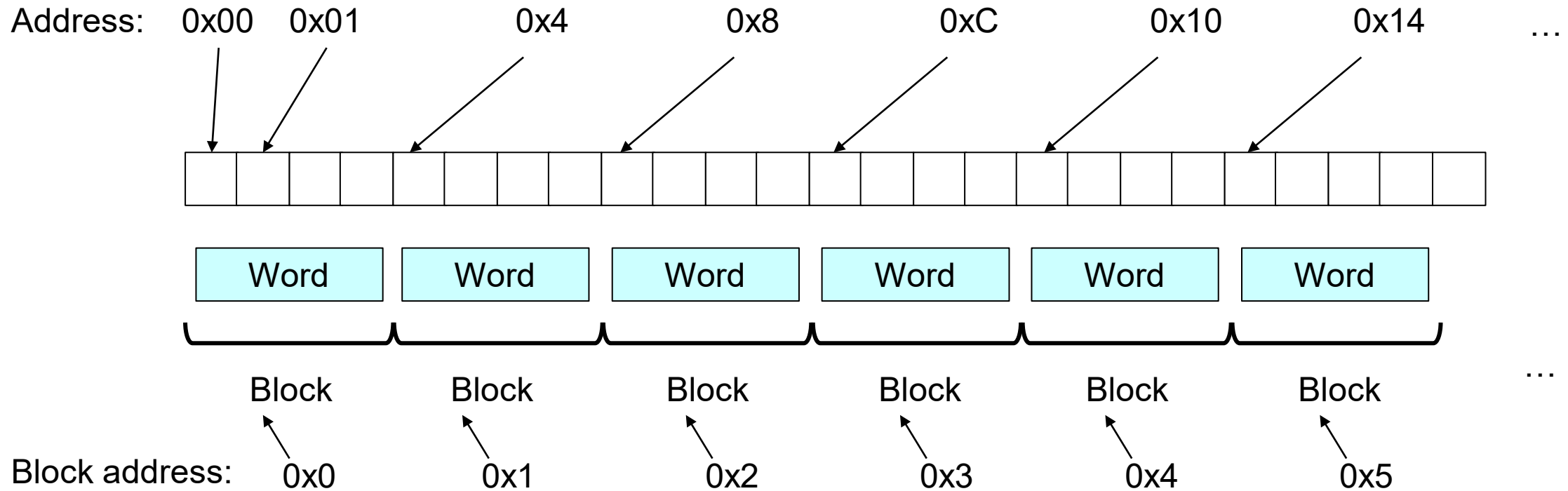
# Tags and Valid Bits

---

- How do we know if a block is in the cache?
  - ❖ Store block address as well as the data
  - ❖ 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 set to 0

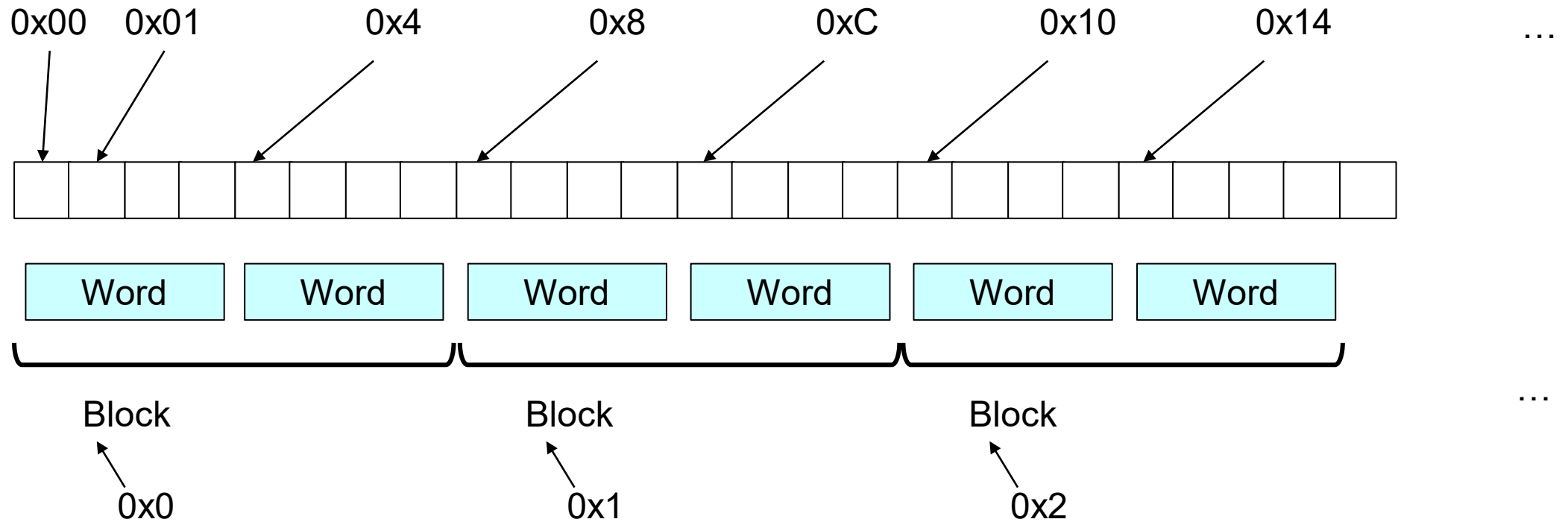
# Cache Example

- 1 word/block



# Cache Example

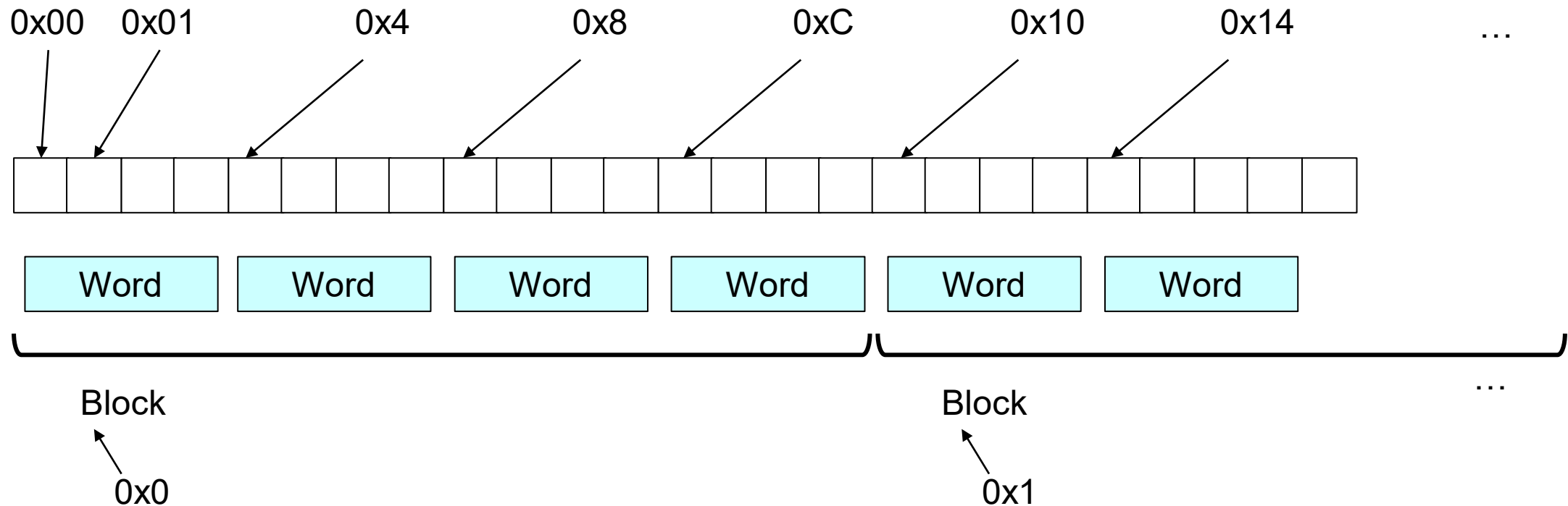
- Another example: 2 words/block





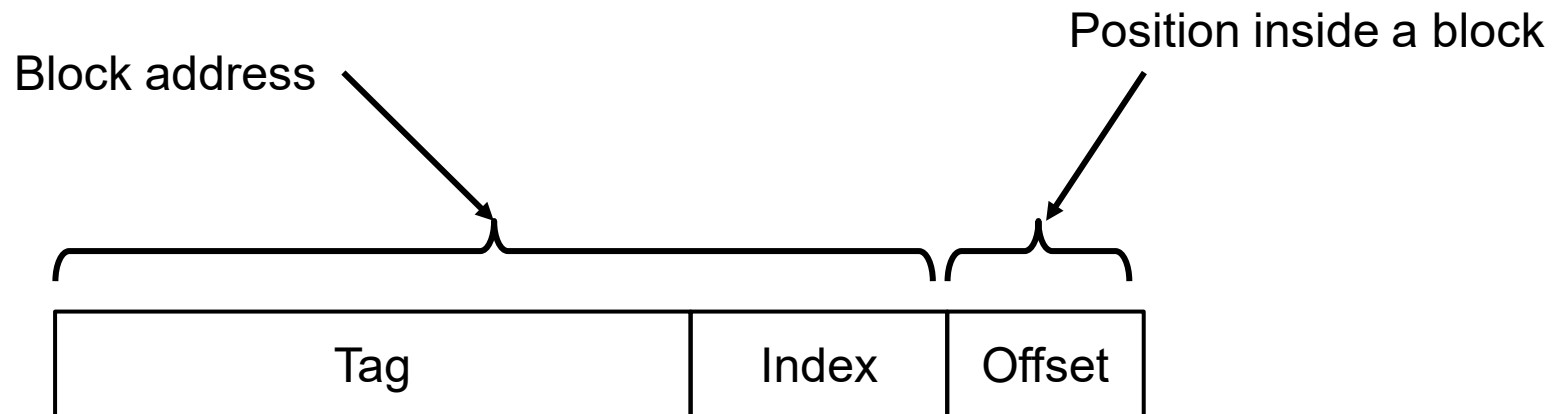
# Cache Example

- Another example: 4 words/block



# Address Decomposition

- **Offset:** Determined by the block size
  - ❖ e.g., 1 word/block: 2 bits, 8 words/block: 5 bits
- **Index:** Determined by the number of blocks in cache
  - ❖ e.g., 4KB cache, 4 words/block  
1 block is 16 bytes.  $4\text{KB} / 16\text{B} = 256$  blocks in cache  
 $\log_2(256) = 8 \rightarrow 8\text{-bit index}$
- **Tag:** The rest



# Cache Example

---

- Memory address: 8 bit
- 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

Load 1

Address	Block address	Hit/miss	Cache block
0x58	010 110	Miss	110

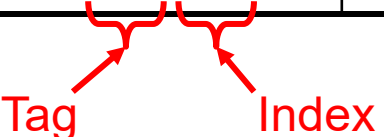


Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	<b>Y</b>	<b>010</b>	<b>Mem[0x58]</b>
111	N		

# Cache Example

Load 2

Address	Block address	Hit/miss	Cache block
0x68	011 010	Miss	010



Index	V	Tag	Data
000	N		
001	N		
010	<b>Y</b>	<b>011</b>	<b>Mem[0x68]</b>
011	N		
100	N		
101	N		
110	Y	010	Mem[0x58]
111	N		

# Cache Example

---

	Address	Block address	Hit/miss	Cache block
Load 3	0x58	010 110	Hit	110
Load 4	0x68	011 010	Hit	010

Index	V	Tag	Data
000	N		
001	N		
010	Y	011	Mem[0x68]
011	N		
100	N		
101	N		
110	Y	010	Mem[0x58]
111	N		

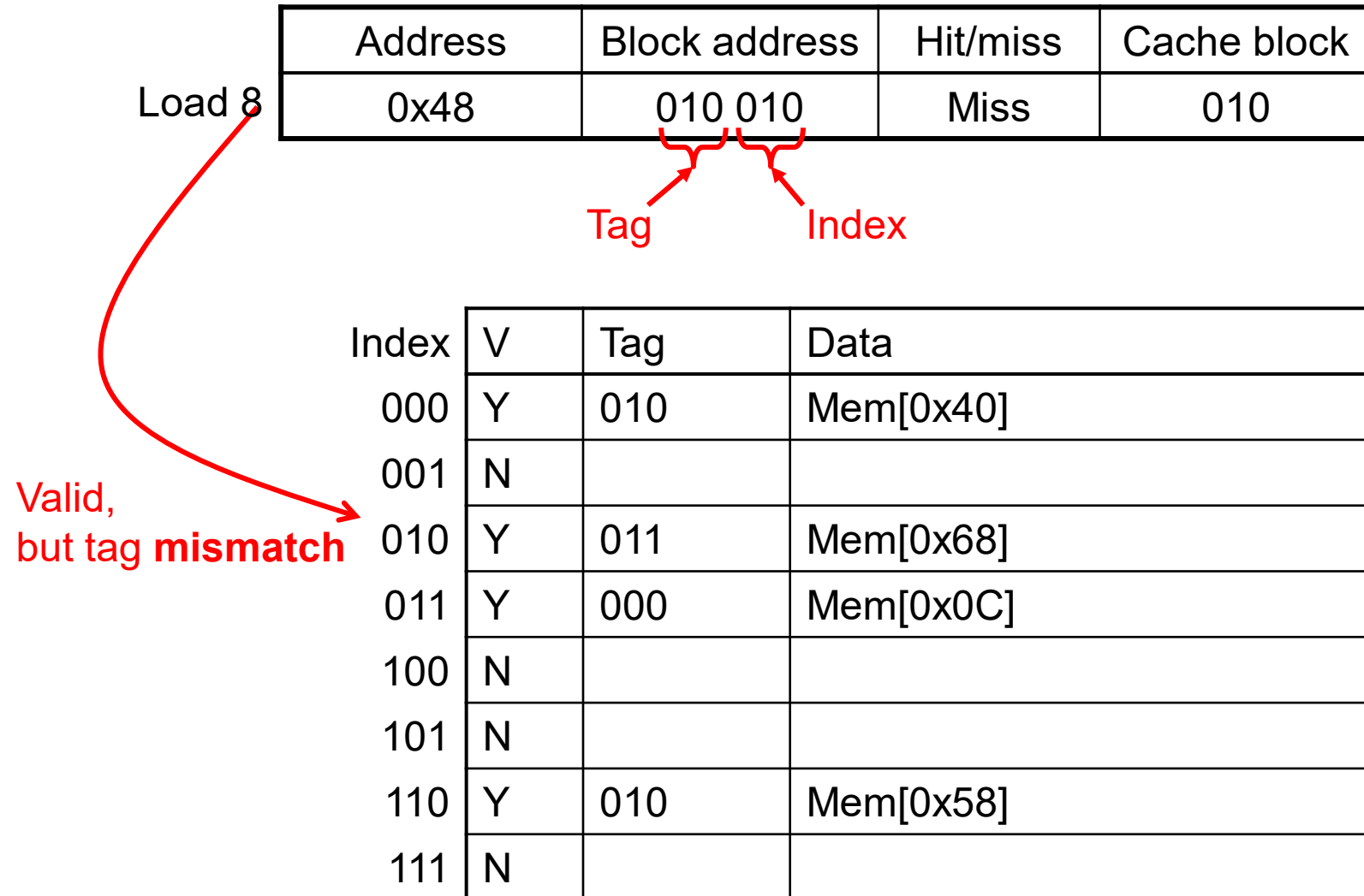


# Cache Example

	Address	Block address	Hit/miss	Cache block
Load 5	0x40	010 000	Miss	000
Load 6	0x0C	000 011	Miss	011
Load 7	0x40	010 000	Hit	000

Index	V	Tag	Data
000	Y	010	Mem[0x40]
001	N		
010	Y	011	Mem[0x68]
011	Y	000	Mem[0x0C]
100	N		
101	N		
110	Y	010	Mem[0x58]
111	N		

# Cache Example



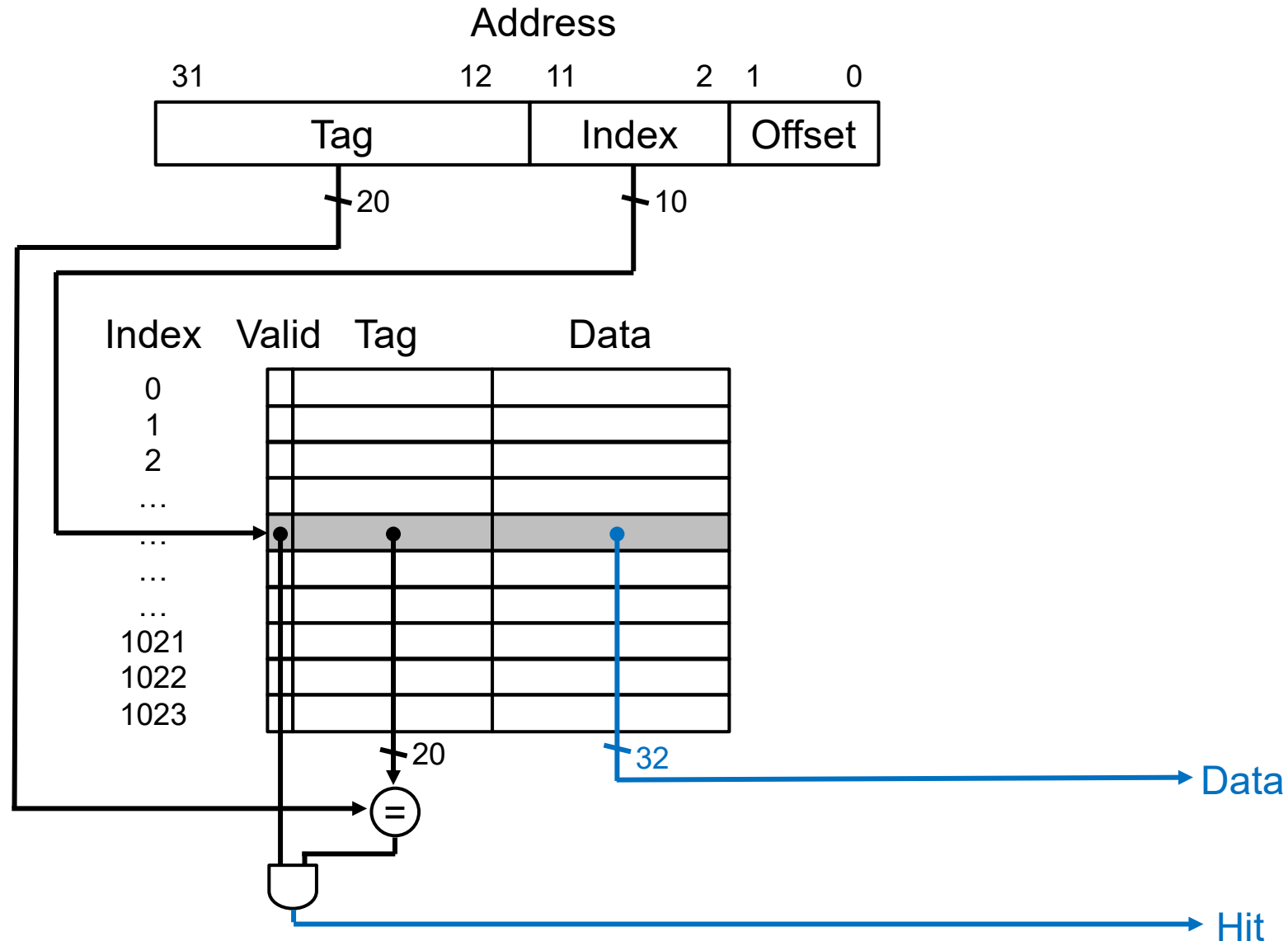
# Cache Example

Load 8	Address	Block address	Hit/miss	Cache block
	0x48	010 010	Miss	010

“Cache conflict”

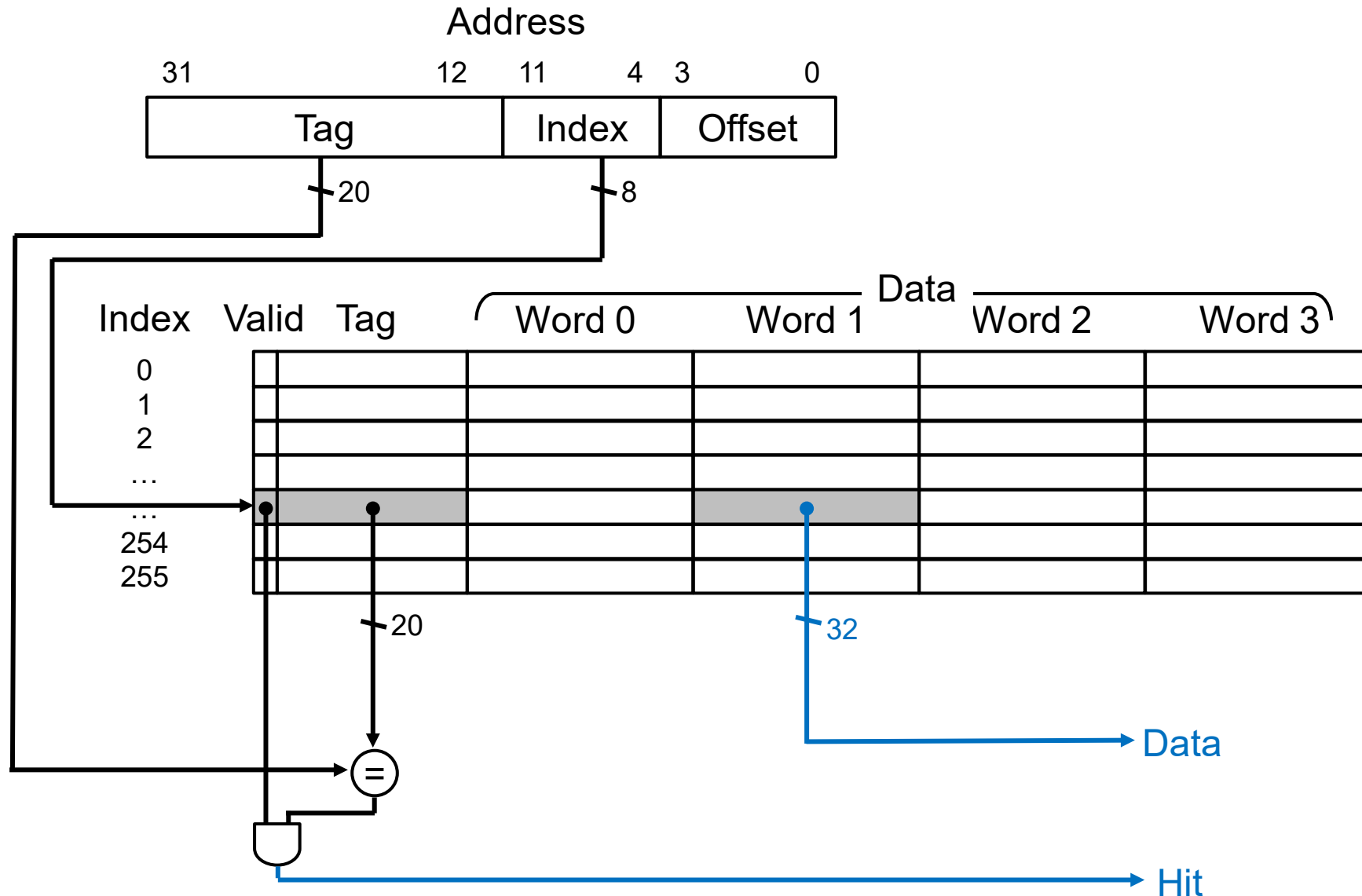
Replace	Index	V	Tag	Data
	000	Y	010	Mem[0x40]
	001	N		
	010	<b>Y</b>	<b>010</b>	<b>Mem[0x48]</b>
	011	Y	000	Mem[0x0C]
	100	N		
	101	N		
	110	Y	010	Mem[0x58]
	111	N		

# Address Subdivision (1)



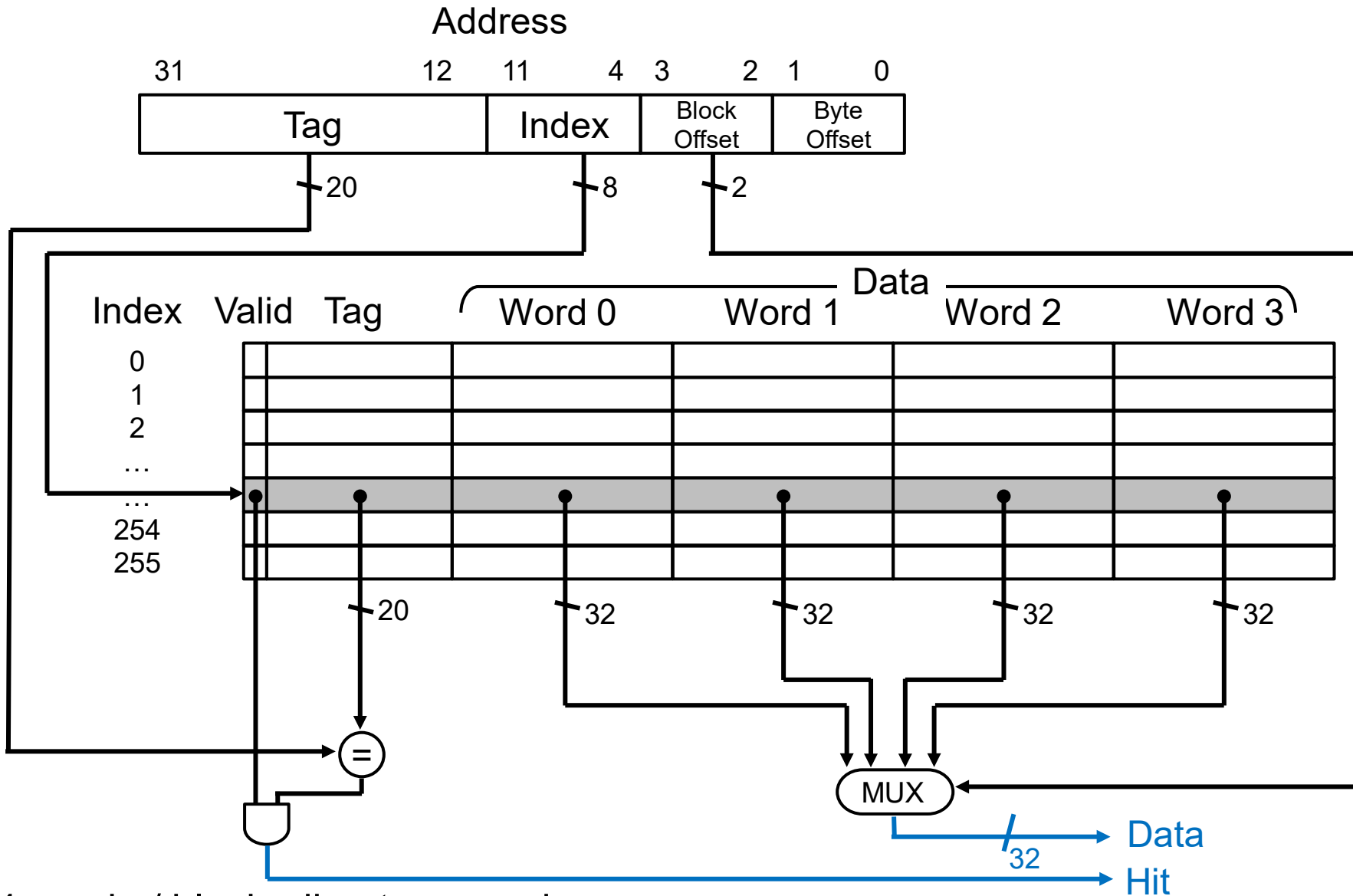
- 4KB cache, 1 word / block, direct-mapped

# Address Subdivision (2)



- 4KB cache, 4 words / block, direct-mapped

# Address Subdivision (3)



- 4KB cache, 4 words / block, direct-mapped



# Block Size Considerations

---

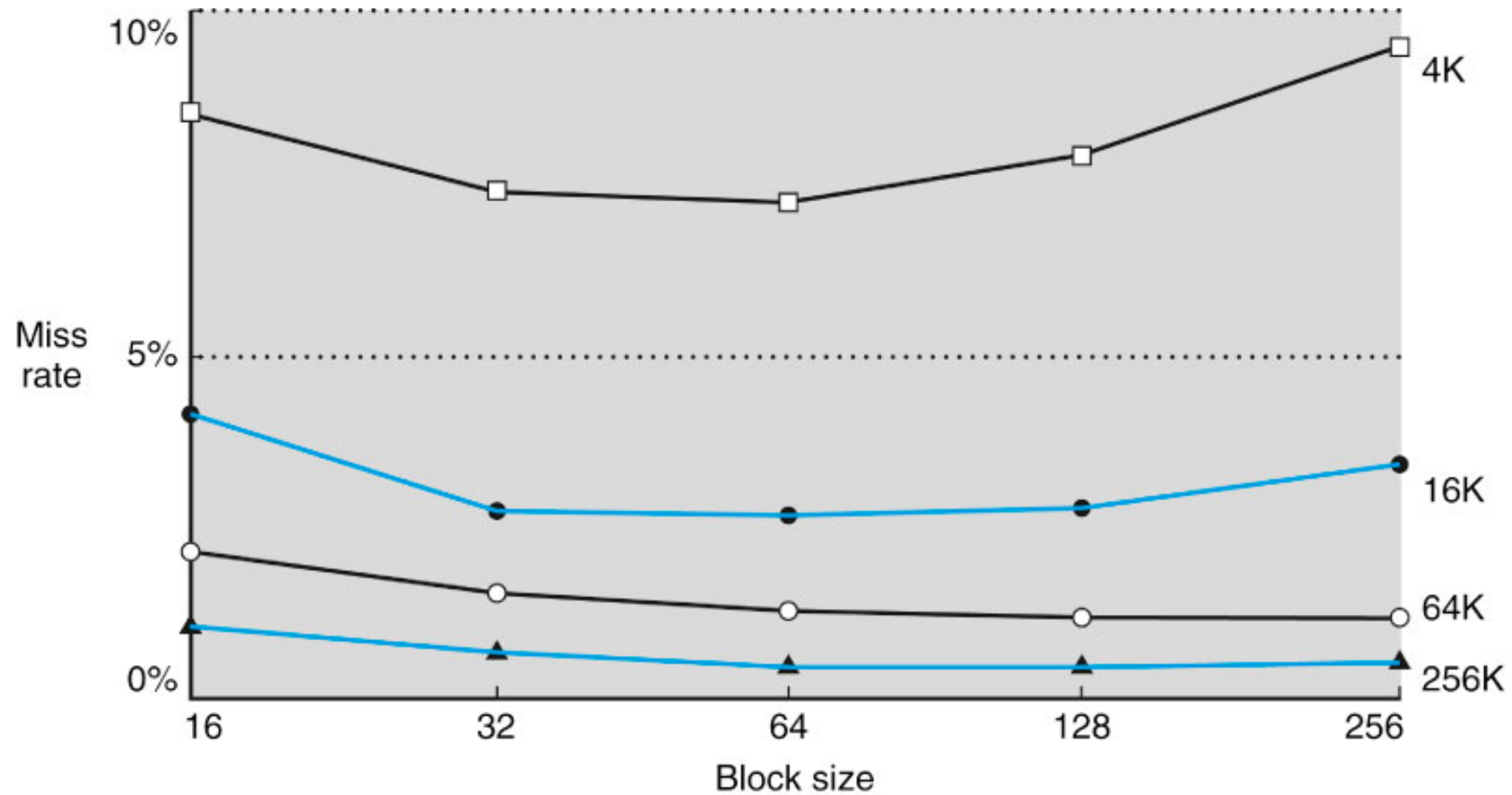
- Larger blocks should reduce miss rate
  - ❖ Spatial locality
- Example: 32bit address, 4 words / block
  - ❖ If there is a miss on address 0x80001234...
  - ❖ ... cache is filled with the following 4 words
    - 0x80001230
    - 0x80001234
    - 0x80001238
    - 0x8000123C

# Block Size Considerations

---

- ...but in a fixed-sized cache,
  - ❖ Larger blocks  $\Rightarrow$  smaller number of cache lines  $\Rightarrow$  more competition  
 $\Rightarrow$  more replacement (*pollution*)  $\Rightarrow$  **higher miss rate**

# Block Size Trade-off



# Cache Misses

---

- On cache hit, CPU proceeds normally
- On cache miss
  - ❖ Stall the CPU pipeline
  - ❖ Fetch block from next level of hierarchy
  - ❖ Instruction cache miss
    - Restart instruction fetch
  - ❖ Data cache miss
    - Complete data access

# Handling Write (Store) Operations

---

- Store hit:
  - ❖ Update cache?
  - ❖ Update memory?
- Store miss:
  - ❖ Fetch the line to the cache?

# Write-Through

---

- On write, also update memory immediately
- ...but writes take a long time
  - ❖ e.g., if base CPI = 1, 10% of instructions are stores, write to memory takes 100 cycles
    - Effective CPI =  $1 \times 100\% + 100 \times 10\% = 11$
- Solution: write buffer
  - ❖ Holds data waiting to be written to memory
  - ❖ CPU continues immediately
    - Stall on write if only the write buffer is full



# Write-Back

---

- On data-write hit, just update the block in cache
  - ❖ Keep track of whether each block is **dirty** (**D** bit)



- When a dirty block is replaced
  - ❖ Write it back to memory
  - ❖ This also uses the write buffer

# Handling Store Miss

---

## ■ No write allocate

- ❖ Just go to the memory and update memory only
- ❖ Cache is unchanged.

## ■ Write allocate

- ❖ Fetch the line from memory as if we have a load miss
- ❖ ... and then do the same thing as a store hit.

# Cache Write Scheme Summary - 1

---

- Write-through, write allocate
  - ❖ Write hit: update cache, update memory
  - ❖ Write miss: fetch the block, update cache, update memory
  
- Write-through, no write allocate
  - ❖ Write hit: update cache, update memory
  - ❖ Write miss: update memory

# Cache Write Scheme Summary - 2

---

- Write-back, write allocate
  - ❖ Write hit: update cache (set dirty bit)
  - ❖ Write miss: fetch the block, update cache (set dirty bit)
  - ❖ Dirty line replace: update memory
  
- Write-back, no write allocate
  - ❖ Write hit: update cache (set dirty bit)
  - ❖ Write miss: update memory
  - ❖ Dirty line replace: update memory

# Measuring Cache Performance

---

- Components of CPU time
  - ❖ Program execution cycles
    - Instruction cycles without any extra stalls from memory
  - ❖ Memory stall cycles
    - Cache miss penalties

- With simplifying assumptions:

Memory stall cycles

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

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

# Cache Performance Example

---

- Given
  - ❖ I-cache miss rate = 2%
  - ❖ D-cache miss rate = 4%
  - ❖ Miss penalty = 100 cycles for a load/store miss
  - ❖ Base CPI (ideal cache: all hits) = 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)
  - ❖  $AMAT = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$
- Example
  - ❖ CPU with 2ns clock, hit time = 1 cycle, miss penalty = 20 cycles, I-cache miss rate = 6%
  - ❖  $AMAT = 2ns \times 100\% + 40ns \times 6\% = 2ns + 2.4ns = 4.4ns$