Computer Architecture Chapter 5

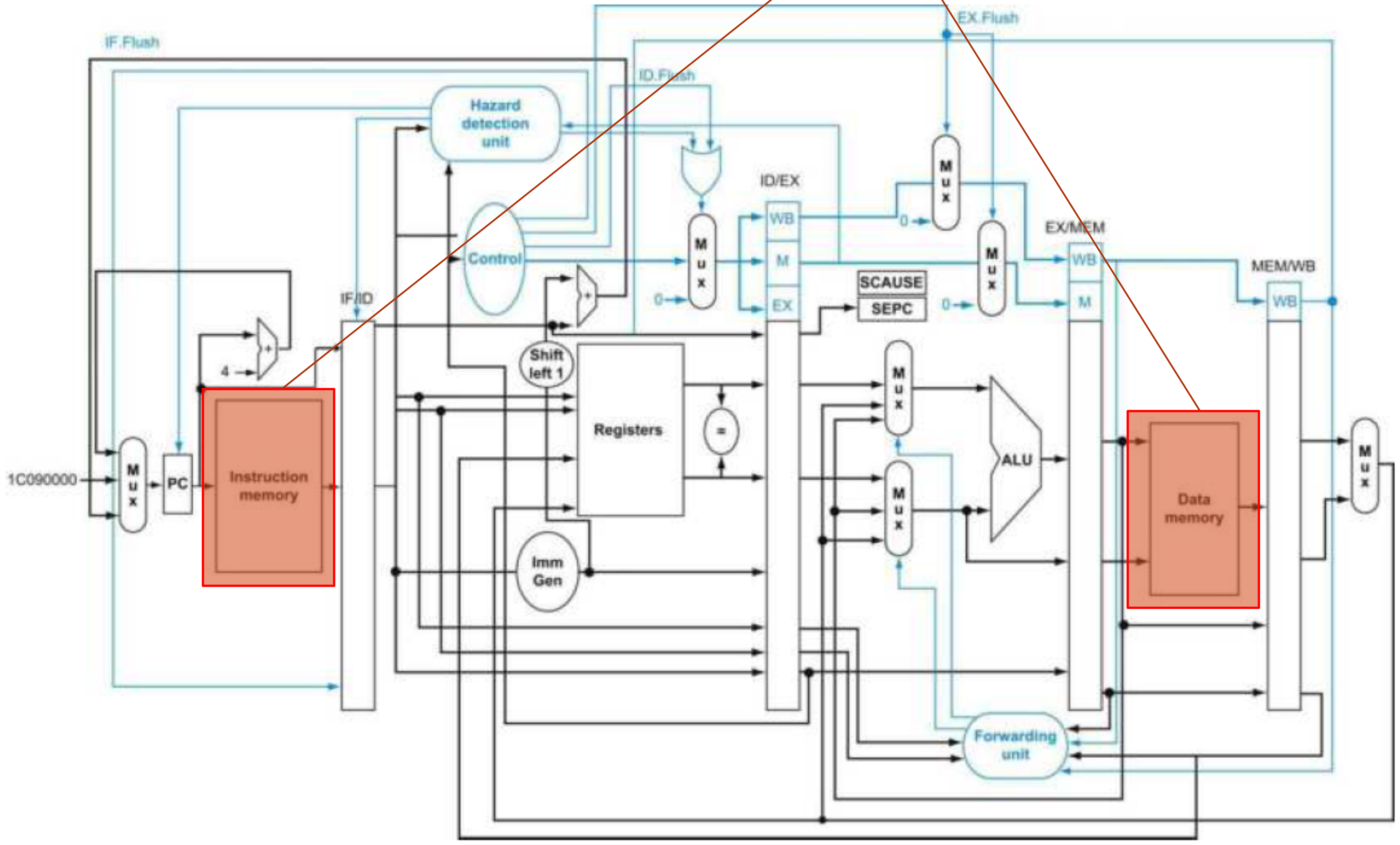
Large and Fast: Exploiting Memory Hierarchy

Joonho Kong
School of EE, KNU

Introduction

A big picture

Our focus in this Chapter



Introduction

Why devising efficient and fast memory architecture is important?

- Instruction memory: accessed every cycle
- Data memory: accessed every load/store (typically accounts for 25~30% of the program)

Devising efficient and fast memory architecture

- Large memory can store a huge amount of data
- Large memory is typically slow (takes more time to find data)
- Slow memory is typically fast
- Slow memory can only store the limited amount of data

- If you are a memory system designer, how do you devise memory system architecture?
- Large memory? or Small memory?

Introduction

Locality

1. Temporal locality

- If an item is referenced, it tends to be referenced again soon
- Let's assume that there is a "for" loop like: for(int i= 0; i< 100; i++), the variable "i" will be referenced by 100 times
- Most programs contain loops
  → showing large temporal locality

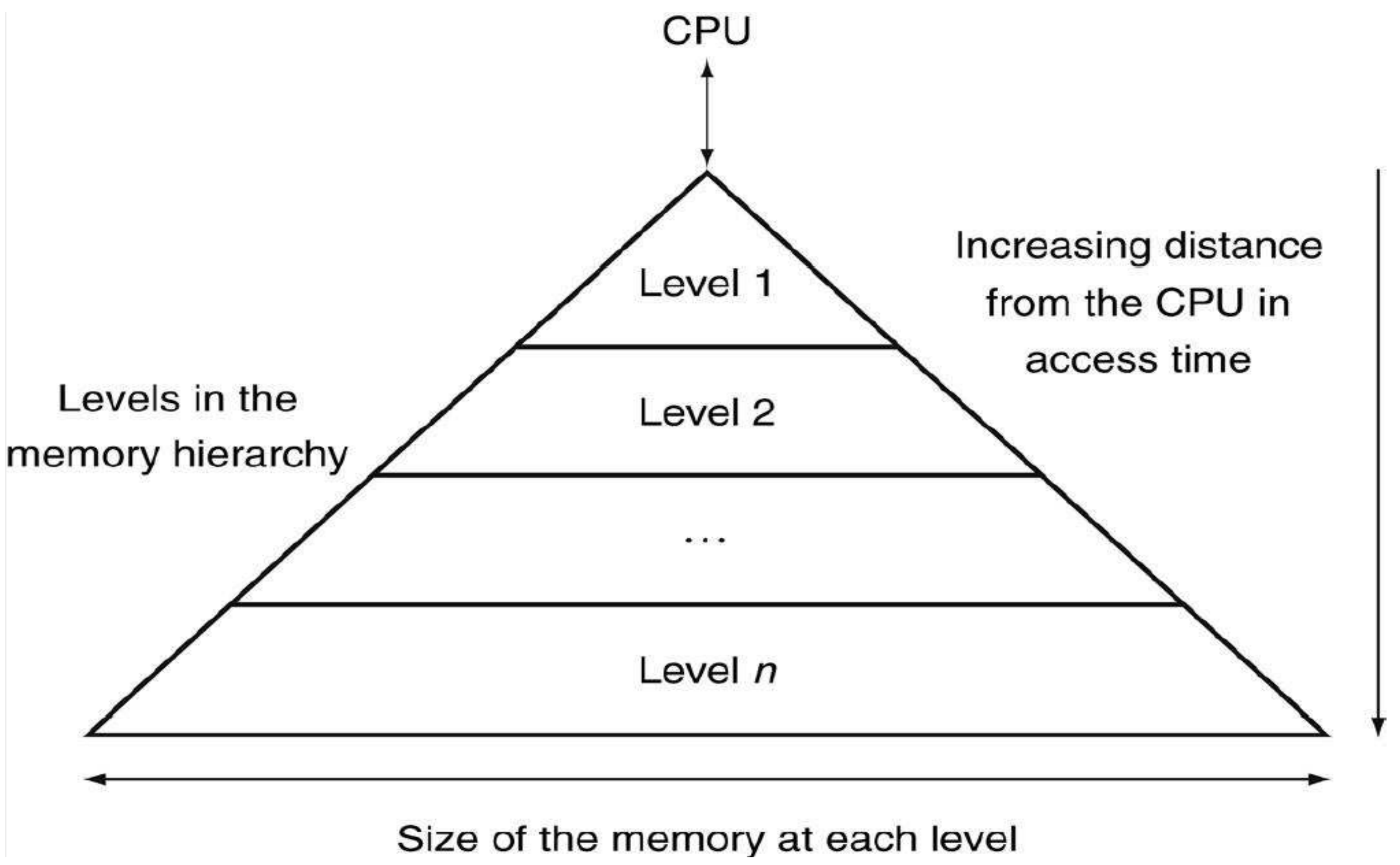
2. Spatial locality

- If an item is referenced, items located near the item tend to be referenced soon
- The array, pre-fetching, and sequential processing is example of spatial locality
  → showing high spatial locality

Introduction

Memory Hierarchy

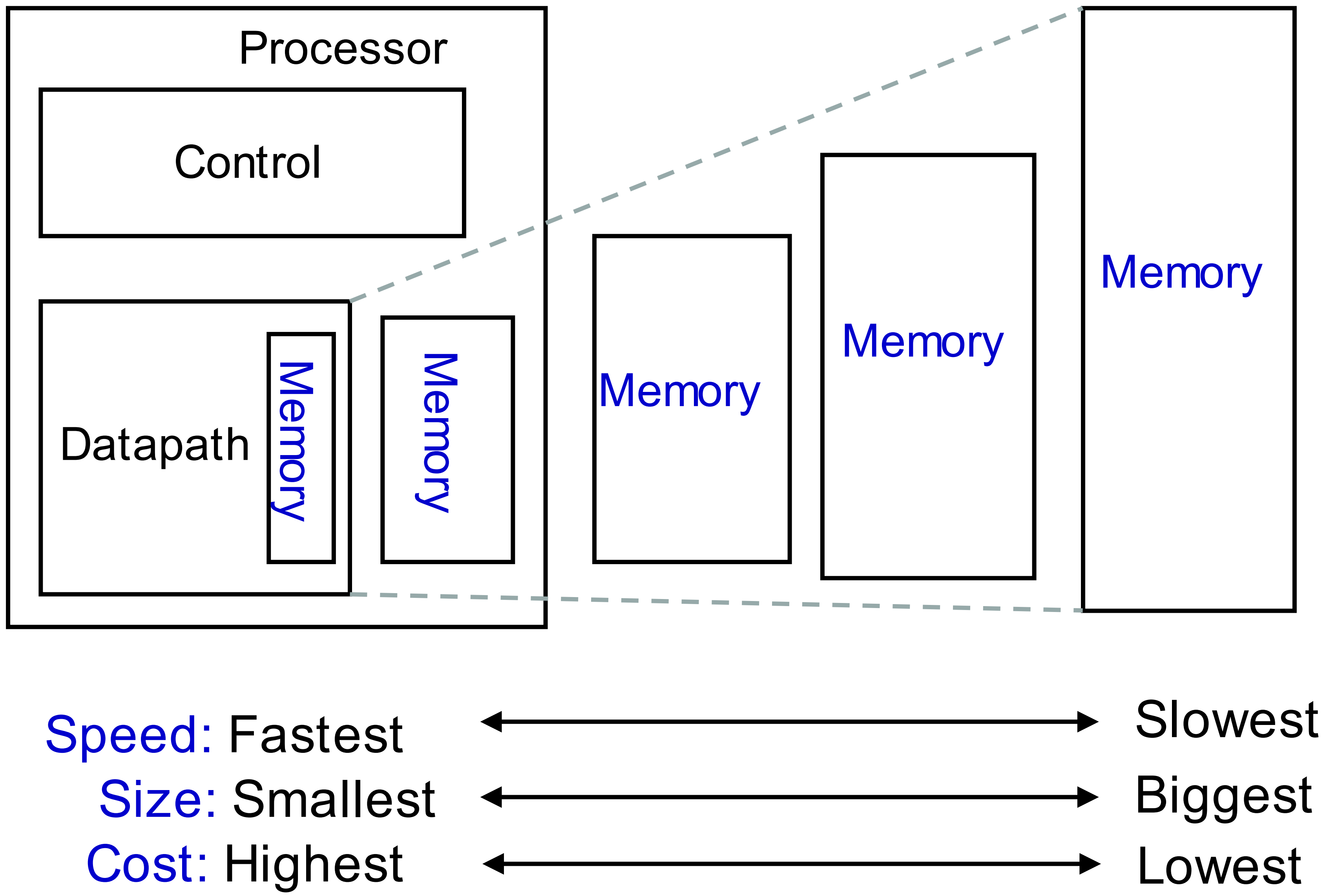
- A memory hierarchy consists of multiple levels of memory
- Each level of memory hierarchy has different size and speed
- Data is delivered through this memory hierarchy
- We place the frequently used data to the memory close to the CPU


CPU
Level 1
Level 2
...
Level n
Levels in the memory hierarchy
Increasing distance from the CPU in access time
Size of the memory at each level

Introduction

Memory Hierarchy

- Register – Cache – Memory – Permanent storage



The Basics of Caches

Basic working mechanism of the Cache

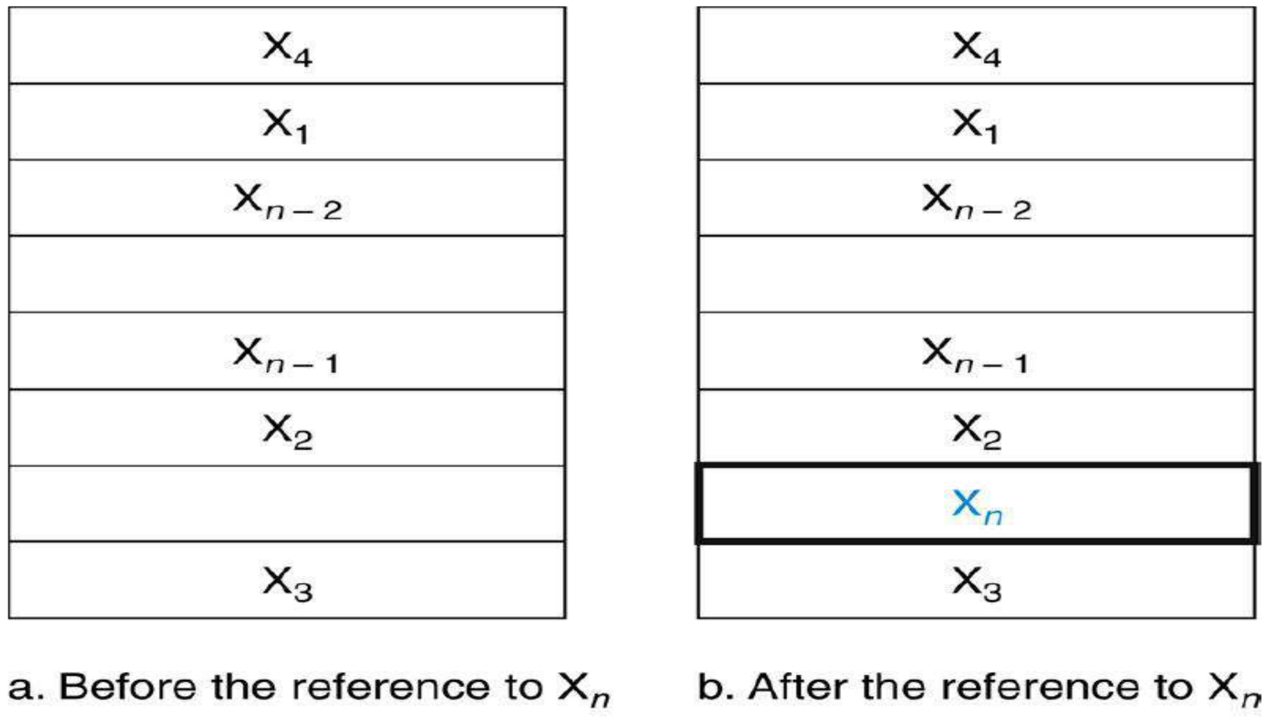
- A cache is an interim storage component which functions as a buffer for larger, slower storage components
- Take advantage of principle of locality
  - Provide cost efficiency
  - Provide the comparable speed of the fastest memory
- Computer systems often use multiple-levels of caches
- Caches provide **an ease of programmability to software programmers**
  - Programmer does not need to explicitly manage caches
  - Caches automatically manage data

The Basics of Caches

Basic working mechanism of the Cache

When the data block is not in the cache

- The processor searches the cache memory for data “Xn”
- The processor will access to a certain address of the main memory
- The processor brings corresponding data and copies it into cache memory
- In the next time we reference the same data, the processor finds the data from caches, not from main memory to get data “Xn”
- **Caches have much lower latency compared to the main memory**



The Basics of Caches

Cache terminology

Block
- The minimum unit of data size which the caches deal with

Hit: If the data (block) is in this level of the memory hierarchy
- Hit rate = cache hit / memory accesses
- Hit time = the time taken to access the cache

Miss: If the data (block) is not in this level of the memory hierarchy
- Miss rate = cache miss / memory accesses
- Miss rate = 1 – hit rate
- Miss penalty = (time to replace a block from lower level) + (time to deliver this block from the lower level memories)

The Basics of Caches

Handling Cache Misses

For an instruction miss

1. Send the original PC value (PC-4) to the memory and stall the pipeline (IF)
2. Instruct main memory (or lower-level cache) to perform a read and wait for the memory (or lower-level cache) to complete the access
3. Write the result into the appropriate cache entry
4. Restart the instruction (typically simultaneously performed with 3)

For a data miss

1. Stall the pipeline (IF/ID/EX/MEM)
2. Instruct main memory (or lower-level cache) to perform a read and wait for the memory (or lower-level cache) to complete the access
3. Return the result from the memory unit and allow the pipeline to continue

The Basics of Caches

Cache arrangement

- Caches can only contain a subset of the data in the main memory
- How can we map the data to the cache memory entry?
- Several different approaches

Direct Mapped

Memory addresses map to a particular location in the cache

Fully Associative

Data can be placed anywhere in the cache

N-way Set Associative

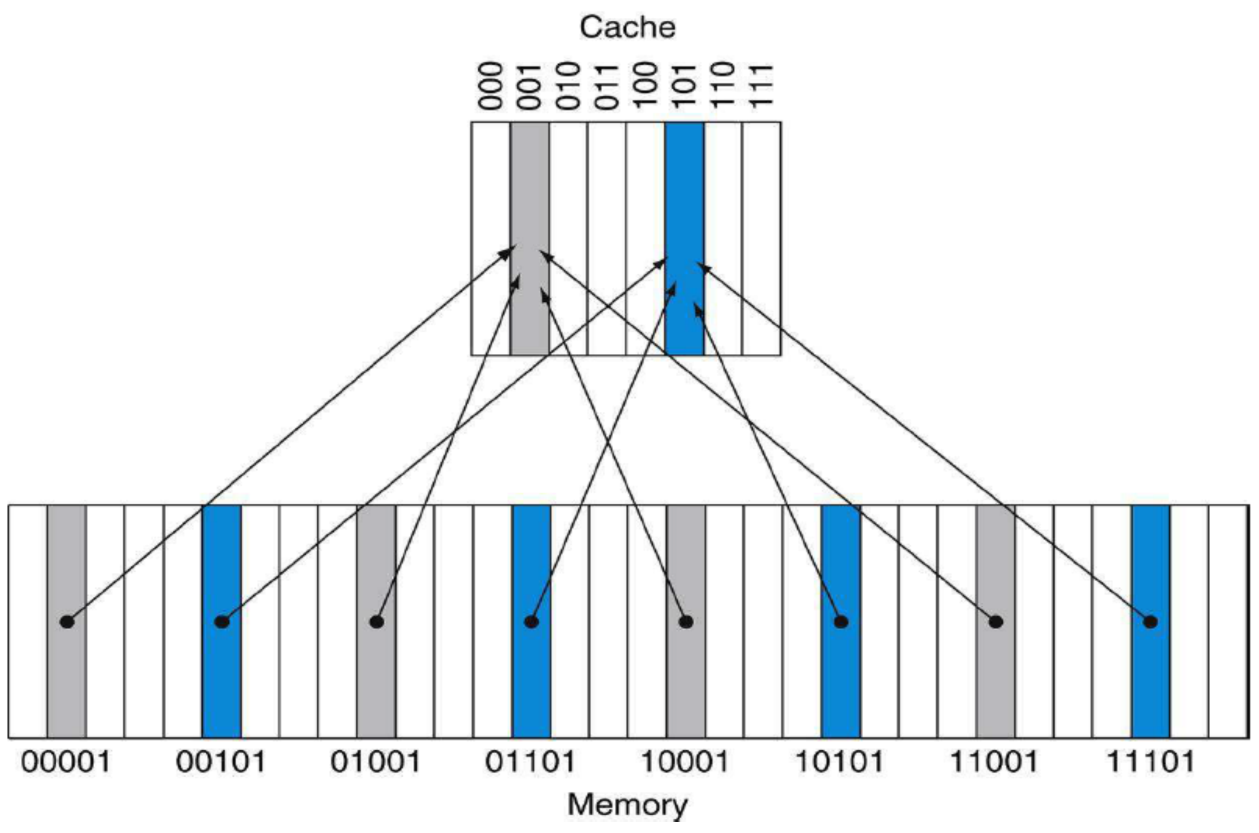
Data can be placed in a limited number (N) of places in the cache depending on the memory address

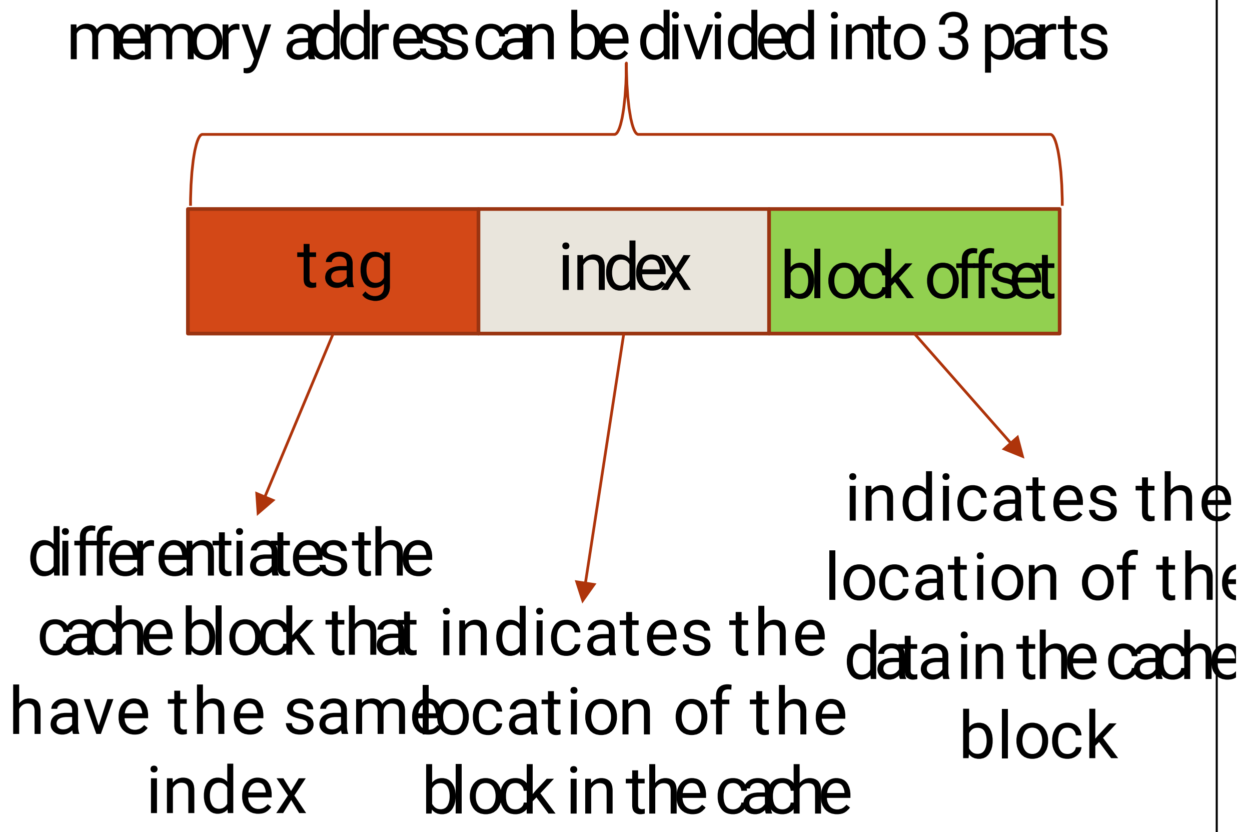
The Basics of Caches

Direct-Mapped Cache

- Each memory location is mapped to exactly one location in the cache
- There is only one place to put the item

Q) How can we find the data we are looking for in the cache?
→ we can search for the data using index and tag bits





memory address can be divided into 3 parts
tag
index
block offset
differentiates the cache block that have the same index
indicates the location of the block in the cache
indicates the location of the data in the cache block

Figure 5.8 is from "Computer Organization and Design"

The Basics of Caches

Valid bit

- When computer is turned on, the cache memory is initially empty or filled with invalid data (such as random values)
- We need to distinguish those initial random values (garbage) and valid data
- To figure out whether the data is valid or not, a valid bit is added for each cache block

If valid bit is set (valid bit = 1),

- Processors do equality test of tag field

If valid bit is not set (valid bit = 0), this is a cache miss

- Processors ignore the equality test of the tag field, and then access to lower memory to get the data
- Then, corresponding data will be copied into cache memory

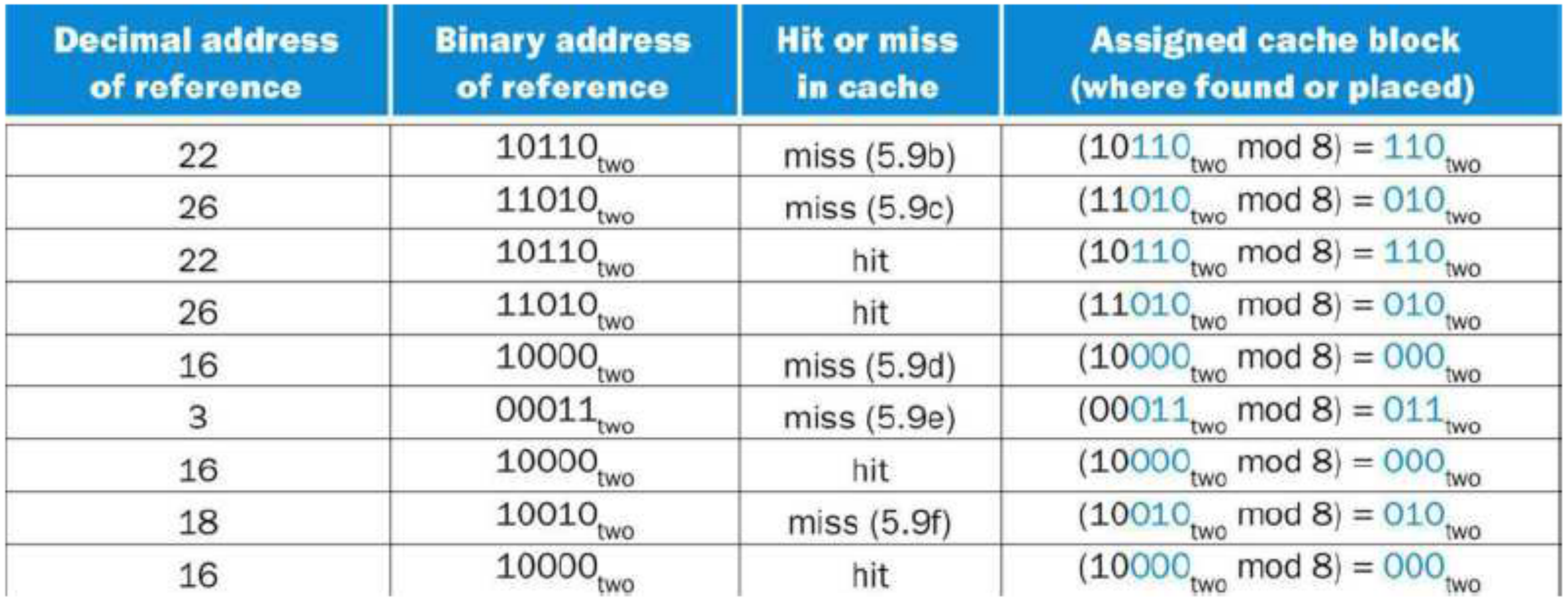
Each cache block has a valid bit

Figure 5.7 is from “Computer Organization and Design”

The Basics of Caches

Accessing a Cache

Q) How will the cache memory be filled after going through an access sequence shown below?



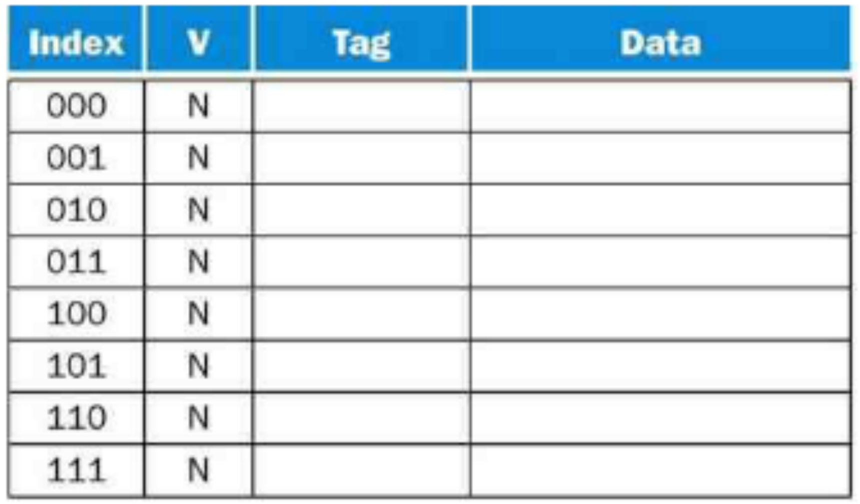
Decimal address of reference	Binary address of reference	Hit or miss in cache	Assigned cache block (where found or placed)
22	10110_{two}	miss (5.9b)	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	miss (5.9c)	$(11010_{two} \bmod 8) = 010_{two}$
22	10110_{two}	hit	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	hit	$(11010_{two} \bmod 8) = 010_{two}$
16	10000_{two}	miss (5.9d)	$(10000_{two} \bmod 8) = 000_{two}$
3	00011_{two}	miss (5.9e)	$(00011_{two} \bmod 8) = 011_{two}$
16	10000_{two}	hit	$(10000_{two} \bmod 8) = 000_{two}$
18	10010_{two}	miss (5.9f)	$(10010_{two} \bmod 8) = 010_{two}$
16	10000_{two}	hit	$(10000_{two} \bmod 8) = 000_{two}$

Figure 5.9 is from "Computer Organization and Design"

The Basics of Caches

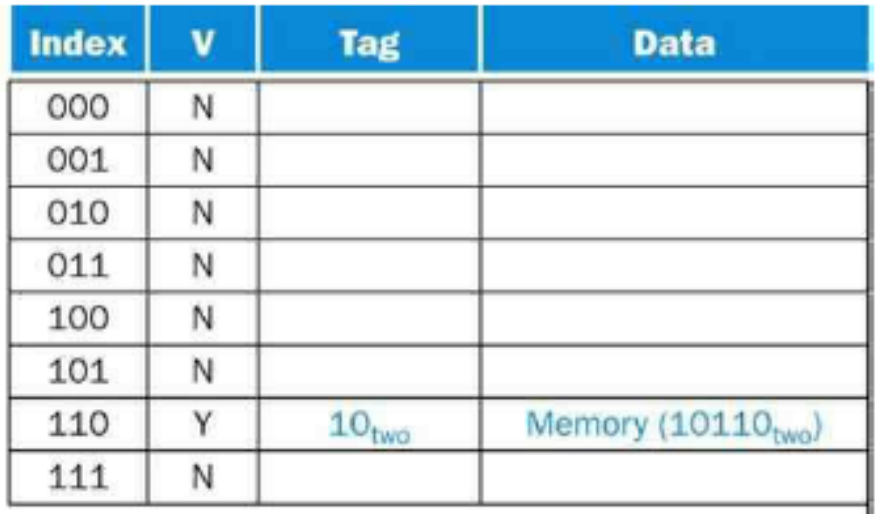
Accessing a Cache

1. The initial state of the cache after power-on



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

2. Access to the address of 22 (10110_{two}) → Miss



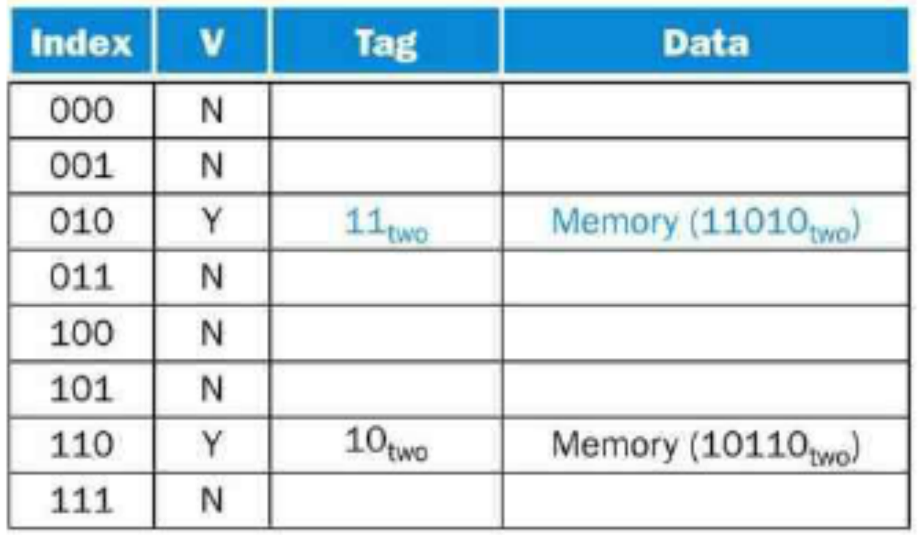
Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

Figure 5.9 is from “Computer Organization and Design”

The Basics of Caches

Accessing a Cache

3. Access to the address 26 (11010_{two}) → Miss



Index	V	Tag	Data
000	N		
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

4. Access to the address of 22 (10110_{two}) → Hit

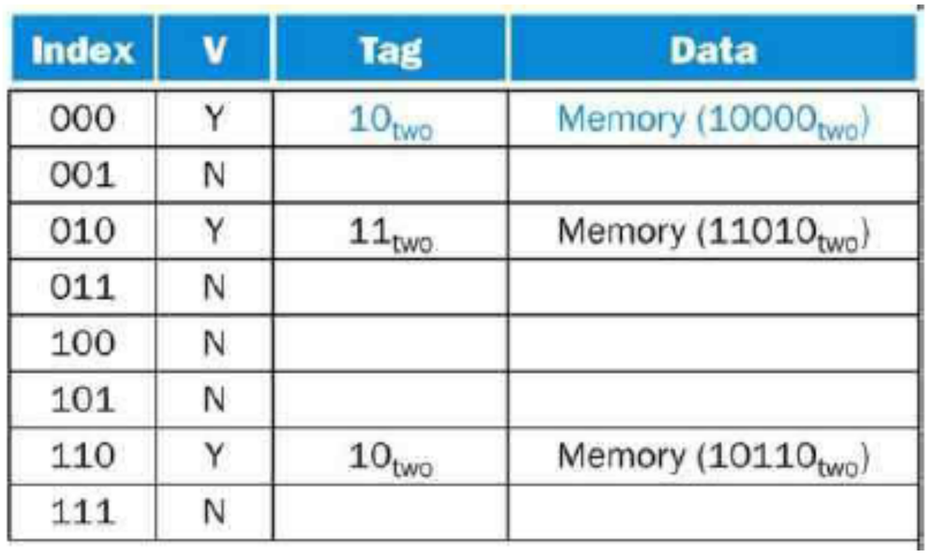
5. Access to the address 26 (11010_{two}) → Hit

Figure 5.9 is from "Computer Organization and Design"

The Basics of Caches

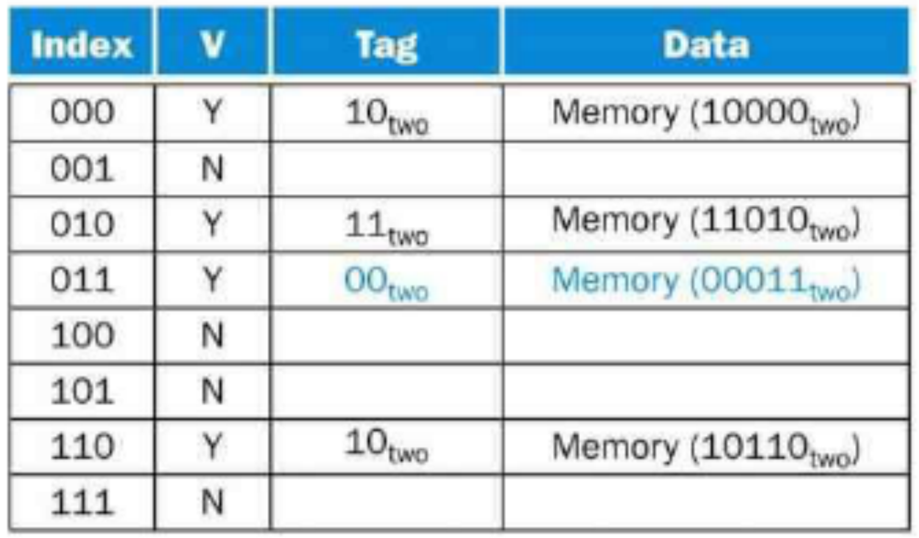
Accessing a Cache

6. Access to the address of 16 (10000_{two}) → Miss



Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

7. Access to the address of 3 (00011_{two}) → Miss



Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

Figure 5.9 is from “Computer Organization and Design”

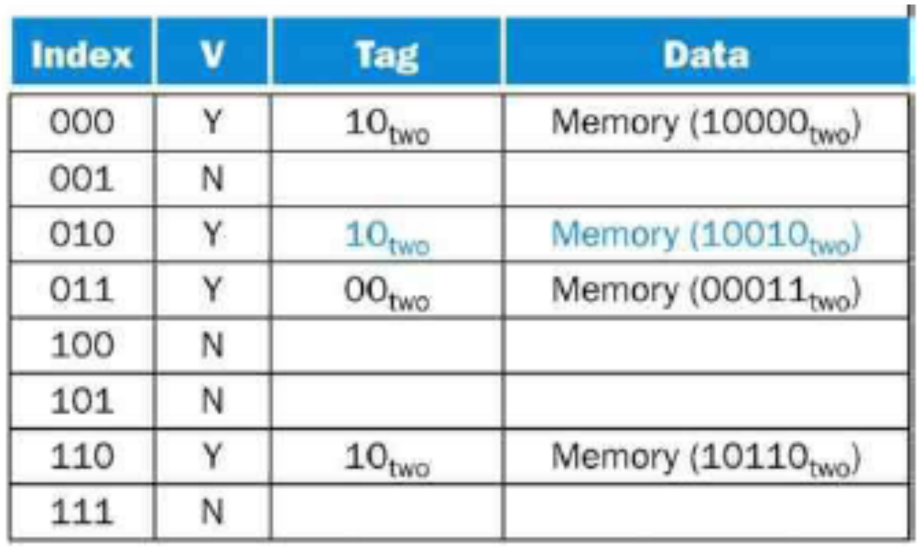
The Basics of Caches

Accessing a Cache

8. Access to the address of 16 (10000_{two}) → Hit

9. Access to the address of 18 (10010_{two}) → Miss

→ the corresponding cache entry is replaced from the data in address 26 (11010_{two}) to that in address 18 (10010_{two})



Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	10_{two}	Memory (10010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

10. Access to the address of 16 (10000_{two}) → Hit

Figure 5.9 is from “Computer Organization and Design”

The Basics of Caches

Overall architecture of Direct-Mapped Cache

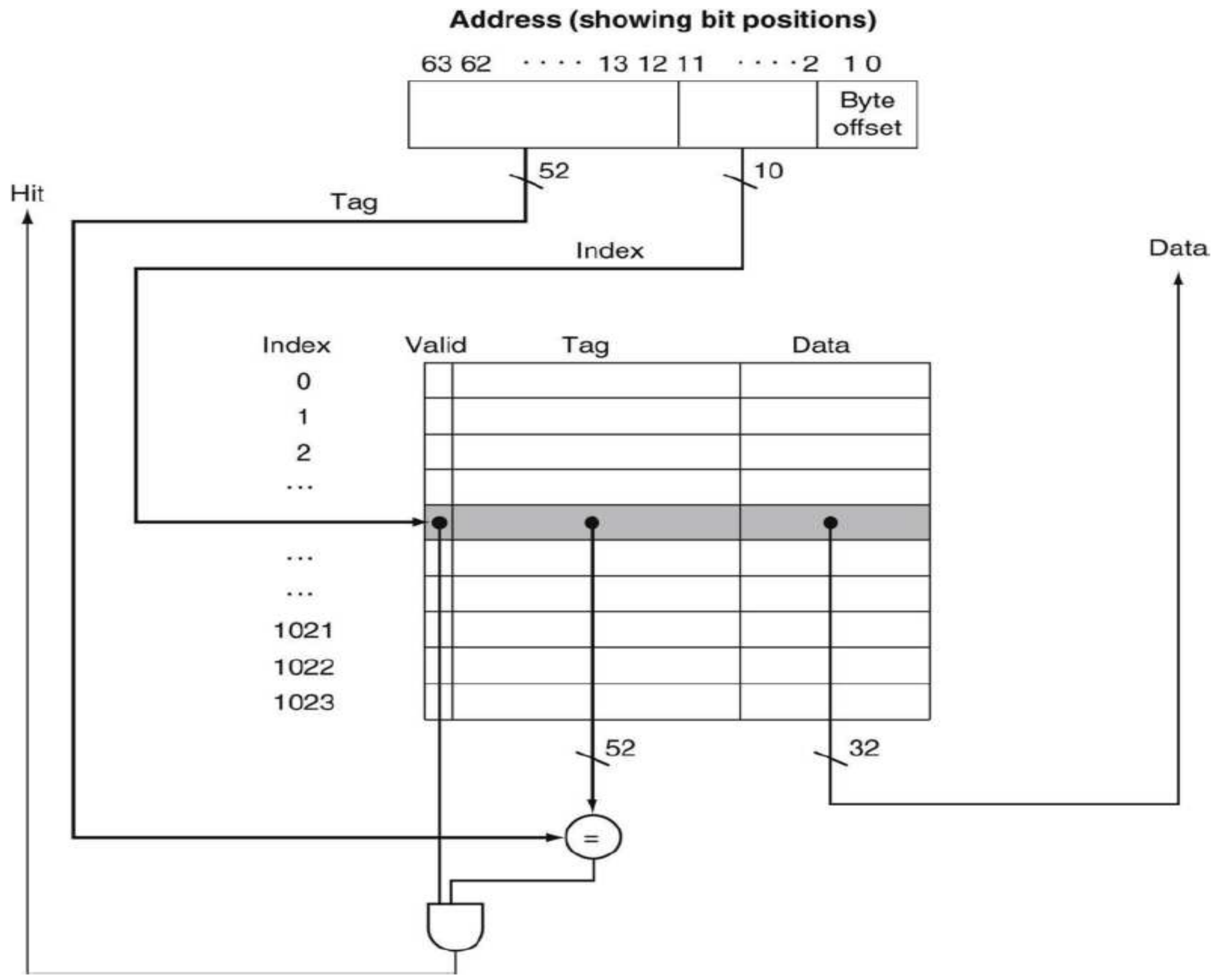


Figure 5.10 is from “Computer Organization and Design”

The Basics of Caches

Composition of memory address for Direct-Mapped Cache

- Index bits are to select the entry of the cache
- Tag bits are for equality test

→ When the valid bit is set and tag field is equal, the result is cache hit and corresponding data can be accessed from the cache

- The total cache size is 1024 (the number of entries) * 4 (word size) = 4KiB

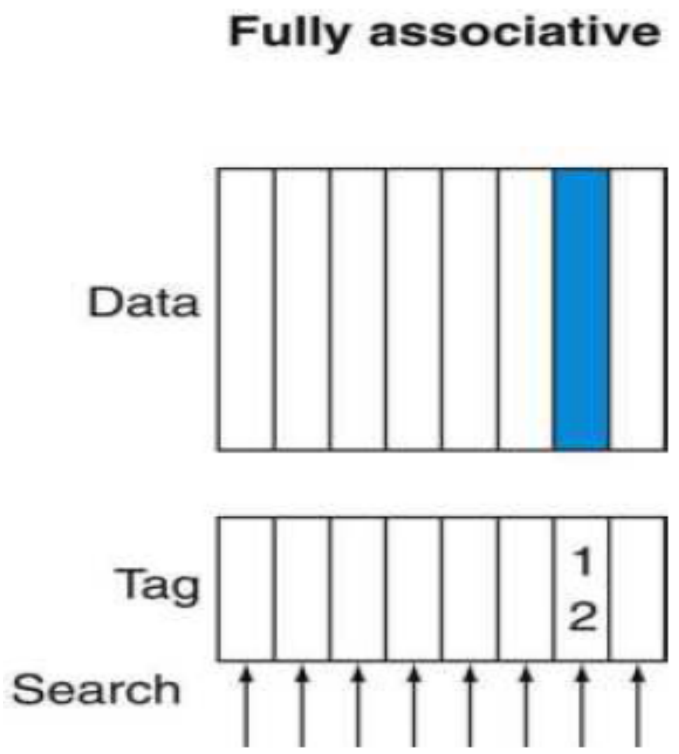
→ **It does not include the tag field and valid bit**

→ When calculating the cache size, we only count data storage (not counting the metadata)

Other Cache Organizations

Fully-associative Cache

- A block in memory can be associated with any entry in the cache
- To find a given block, all entries in the cache must be searched
- It can be done in parallel with CAM (content addressable memory) cells
- This scheme requires significant hardware cost
- This scheme is efficient in the cache what has small number of blocks



we must search every entry in the cache – must do tag matching for every entry

Figure 5.14 is from "Computer Organization and Design"

Other Cache Organizations

Set-associative Cache

- It is a hybrid scheme between the direct-mapped and fully-associative cache
- Each block in memory can be placed in a fixed number of cache entry
- Set-associative cache with "n" locations for a block is called "n-way set-associative" cache
- The direct-mapped cache can be regarded as "1-way set-associative cache"
- A block is mapped into a set, and then only the blocks in the set are searched

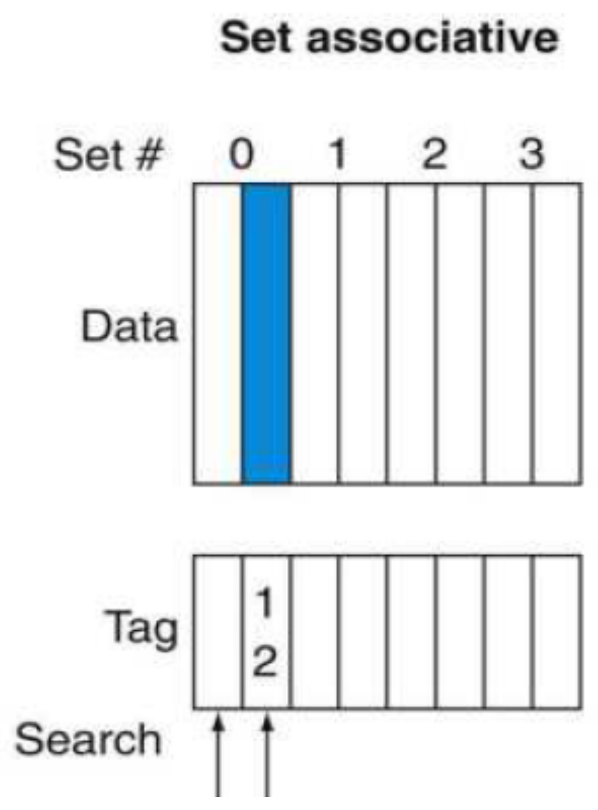


Figure 5.14 is from "Computer Organization and Design"

Other Cache Organizations

N-way Set-associative Cache for 8 Blocks

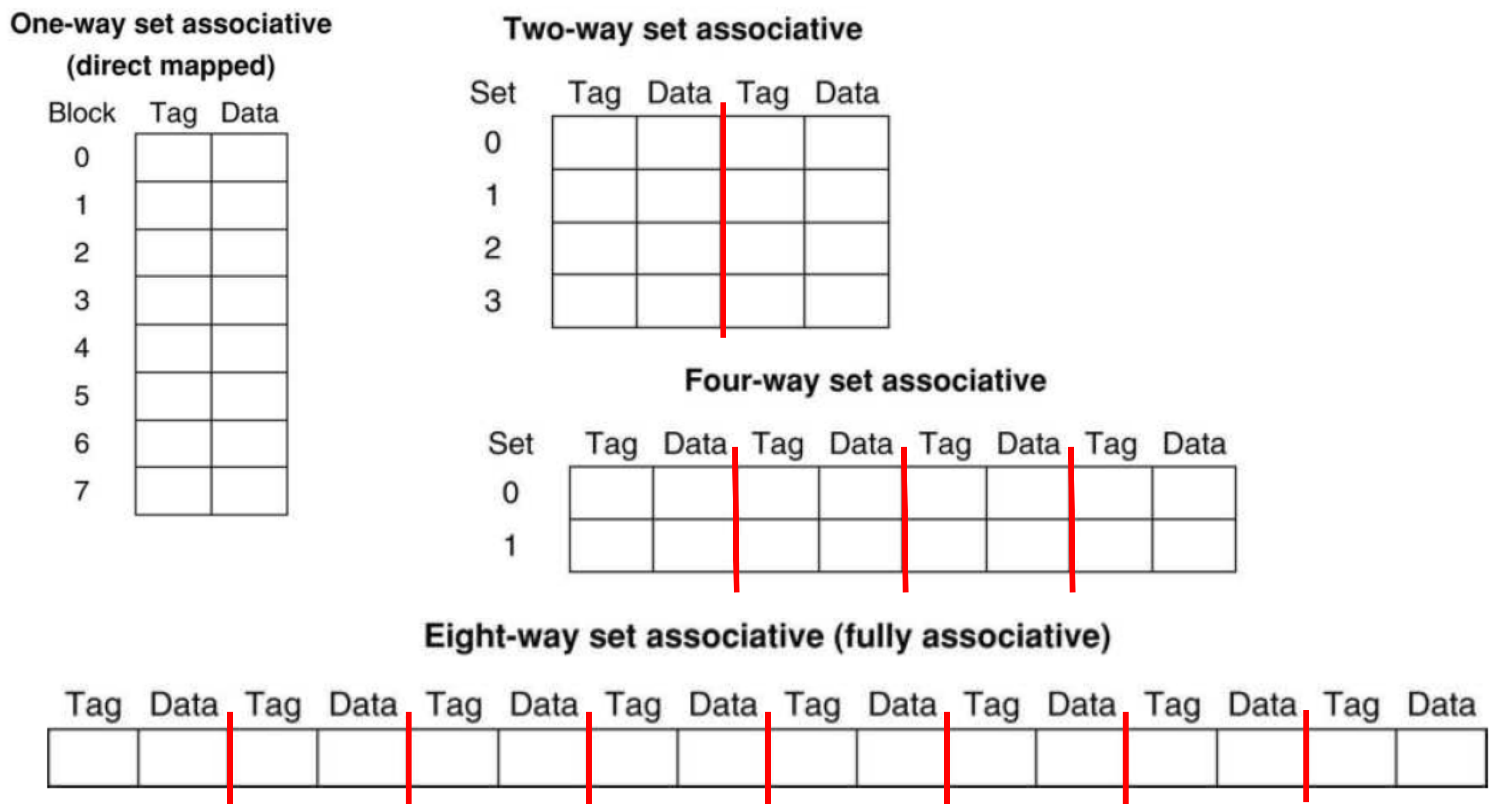


Figure 5.15 is from "Computer Organization and Design"

Other Cache Organizations

N-way Set-associative Cache

- It usually decreases the miss rate
- The main disadvantage is a potential increase in the hit time
- N-way set associative caches allow up to N conflicting references to be cached
  → N is the number of cache blocks in each set
  → N comparisons are needed to search all blocks in the set in parallel
  → When there is a cache miss and no available entry in the cache set, which block should be replaced? (this was easy for direct mapped caches – there's only one entry!)

- For fixed cache capacity, higher associativity leads to higher hit rates
  → Because more combinations of memory blocks can be present in the cache (i.e., more flexible management)
  → Set associativity enables better management, but at what cost?

Other Cache Organizations

4-way Set-associative Cache

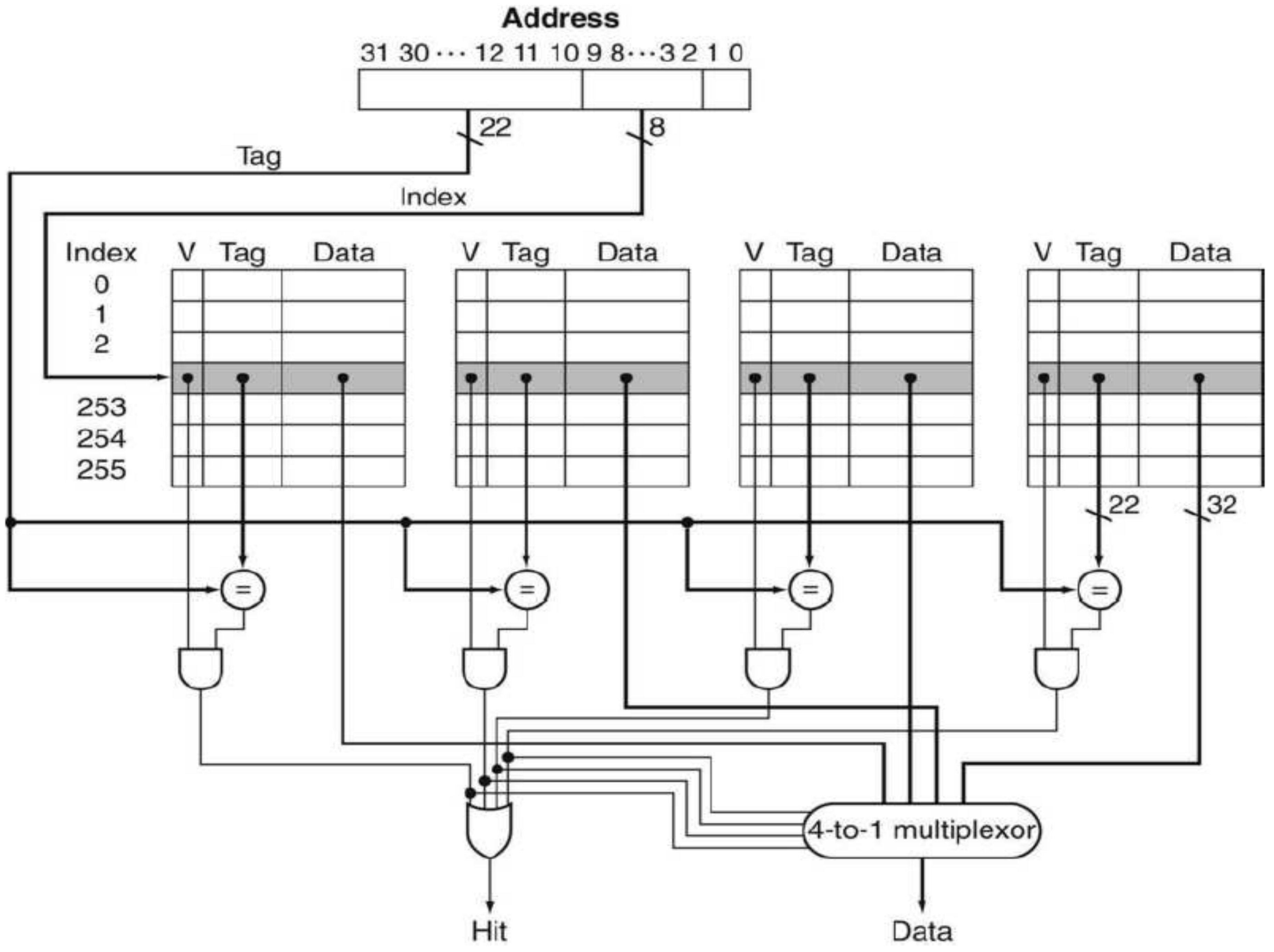


Figure 5.18 is from “Computer Organization and Design”

Cache Replacement Policy

Replacement policy - Least Recently Used (LRU)

When a miss is occurred and one block should be replaced, which block should be replaced?
- The most efficient and popular way to replace the block is LRU
- The LRU policy replace the block that has been unused for the longest time

accessing address	0	1	2	3	0	1	4	0	1	2	3	4
block 0												
block 1												
block 2												

An example of LRU policy in the case of fully associative caches

Impact of Cache Block Size

Large Block Size for Lower Miss Rate

- Larger blocks exploit spatial locality
- Increasing the block size can decrease the miss rate until the block size of 64B
- But increasing the block size leads to decrease the number of block in the cache
 → Less flexible, coarser-grained management, miss rate increase beyond the block size of 64B
- Miss penalty will increase as the block size increases
 → We need to pay more cycles to deliver the blocks

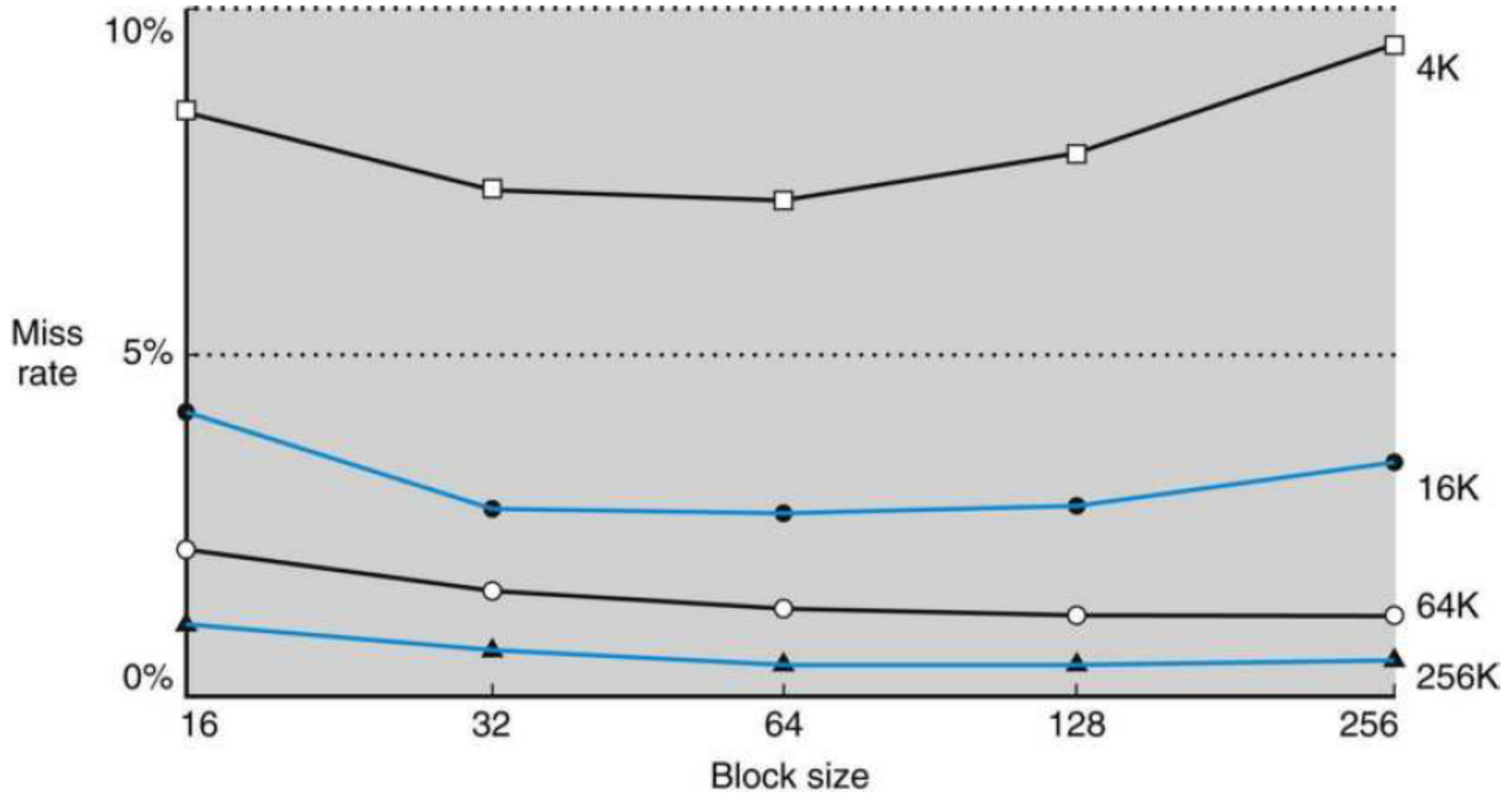


Figure 5.11 is from "Computer Organization and Design"

Cache Write Policies

Handling Writes

1. Write-through

- The simplest way to keep the data coherency between the main memory and the cache
- This scheme always writes the data into both the caches and the main memory (or for all levels of the caches and memory) when data is updated (modified)
- It would not provide good performance because every write accesses to main memory and lower-level caches (too frequent write requests to the memory and lower-levels of the caches)

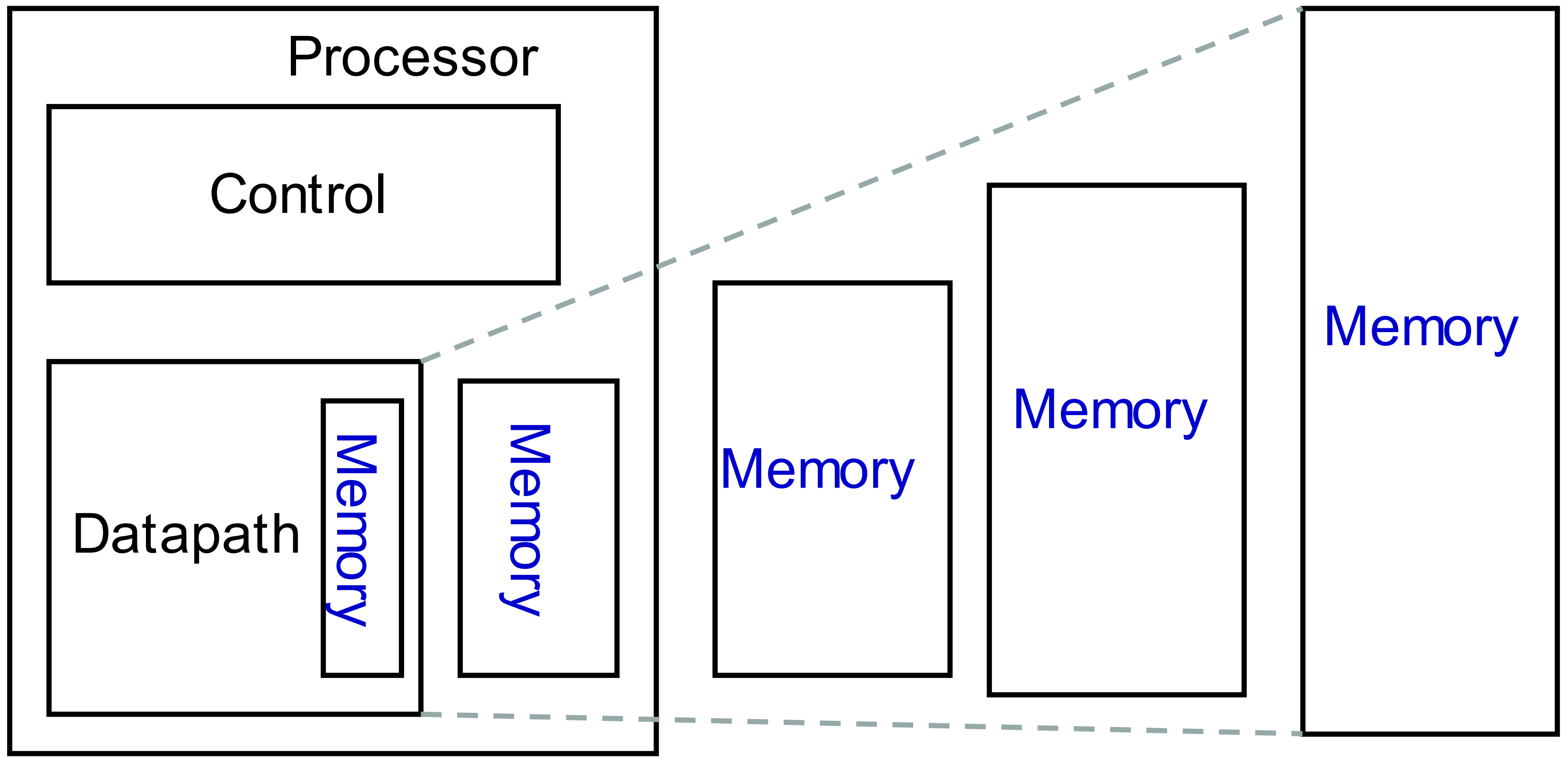
2. Write-back

- A data is updated only in the current cache
- The updated data will be written in the main memory (or lower-level caches) when the cache block is evicted
- This scheme can improve performance because it reduces the number of accesses to main memory
- Write-back scheme is more complicated to implement than write-through scheme

Multi-Level Caches

Reducing the Miss Penalty Using Multilevel Cache

- Most processors support an additional level(s) of caching
- Second-level of cache is accessed whenever a miss occurs in upper cache
- Access to second-level cache reduce the access time of main memory
- It reduces miss penalty


Processor
Control
Datapath
Memory
Memory
Memory
Memory
Memory