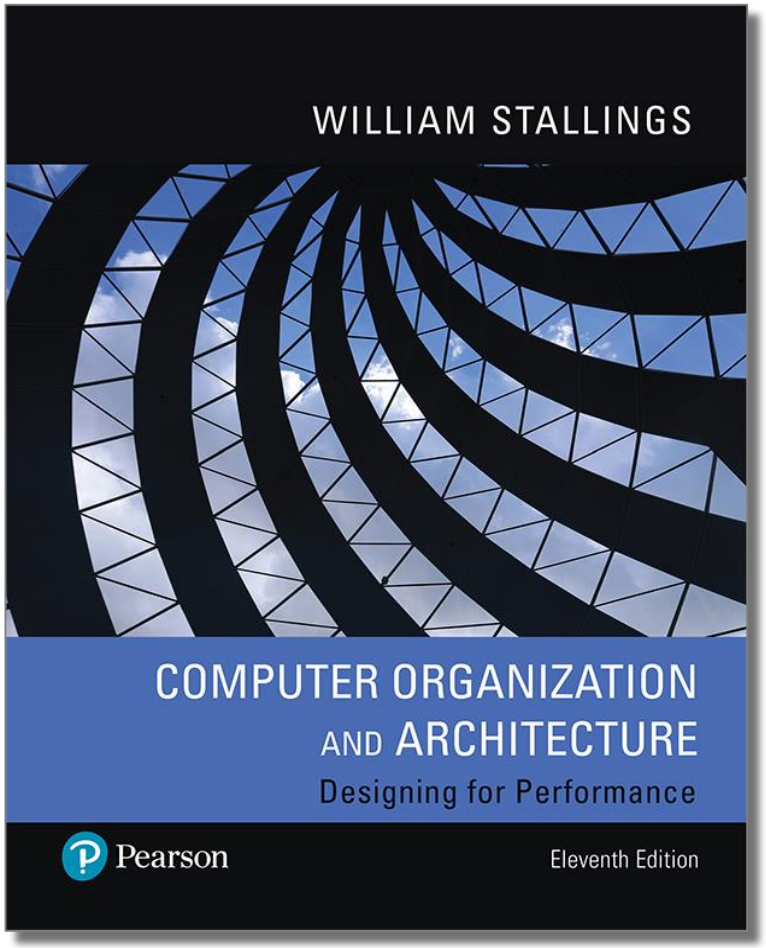
Computer Organization and Architecture

Designing for Performance

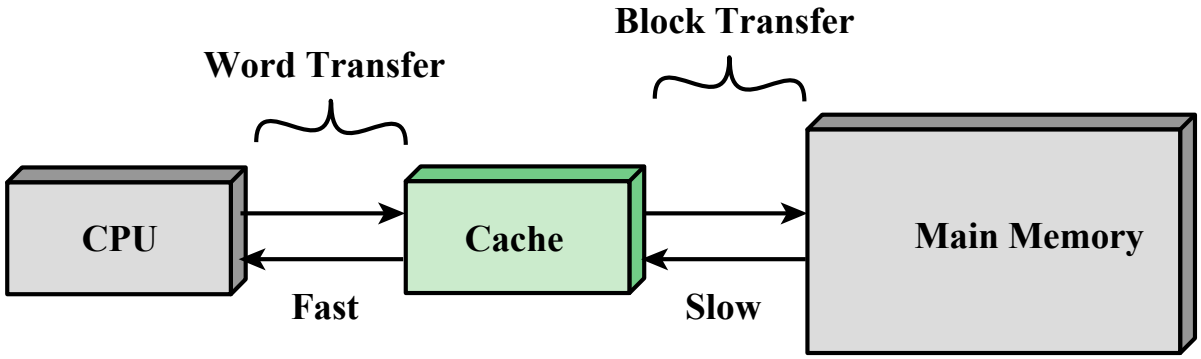
11th Edition



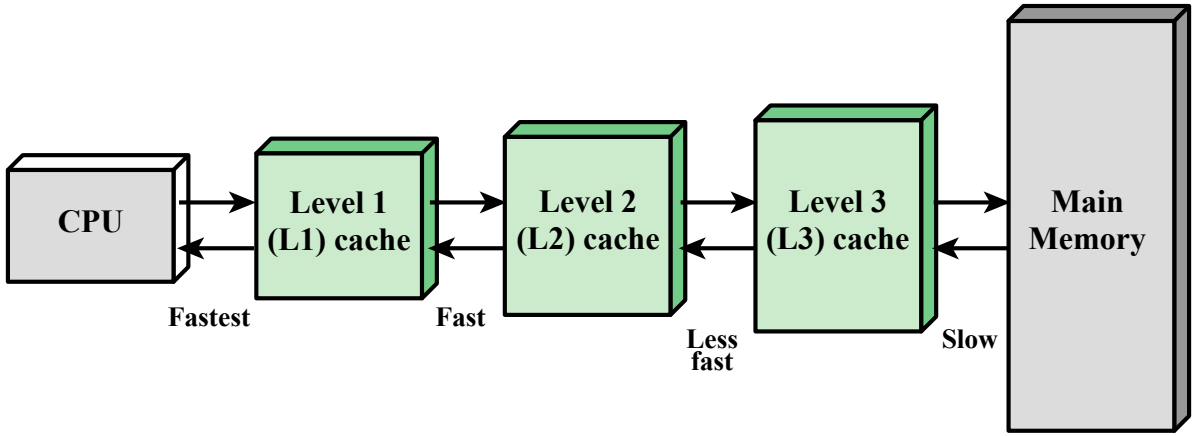
Chapter 5

Cache Memory

Figure 5.1
Cache and Main Memory



(a) Single cache



(b) Three-level cache organization

Cache Memory Principles

- Block
  - The minimum unit of transfer between cache and main memory
- Frame
  - To distinguish between the data transferred and the chunk of physical memory, the term frame, or block frame, is sometimes used with reference to caches
- Line
  - A portion of cache memory capable of holding one block, so-called because it is usually drawn as a horizontal object
- Tag
  - A portion of a cache line that is used for addressing purposes (which block?)
- Line size
  - The number of data bytes, or block size, contained in a line

Figure 5.2
Cache/Main Memory Structure

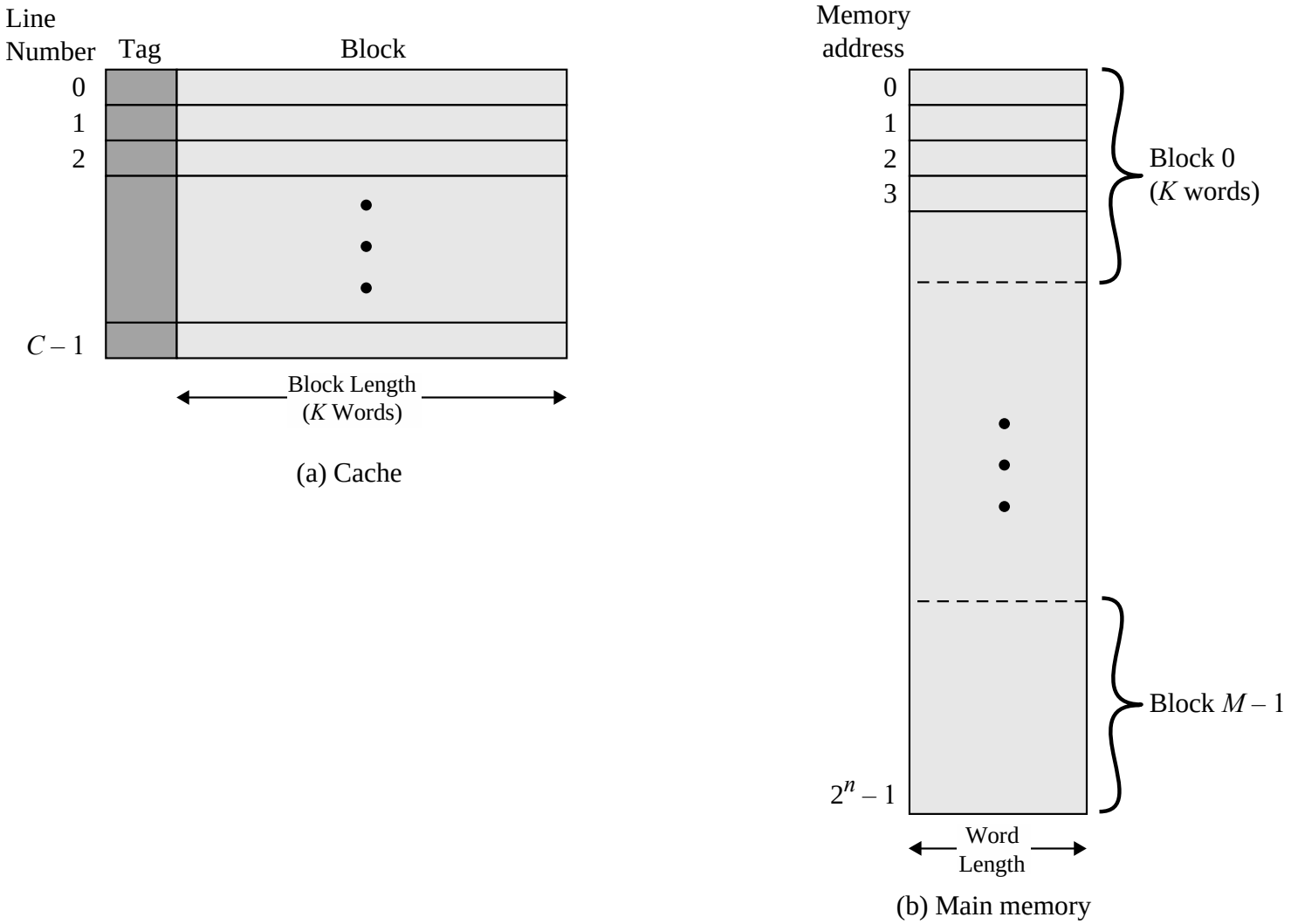
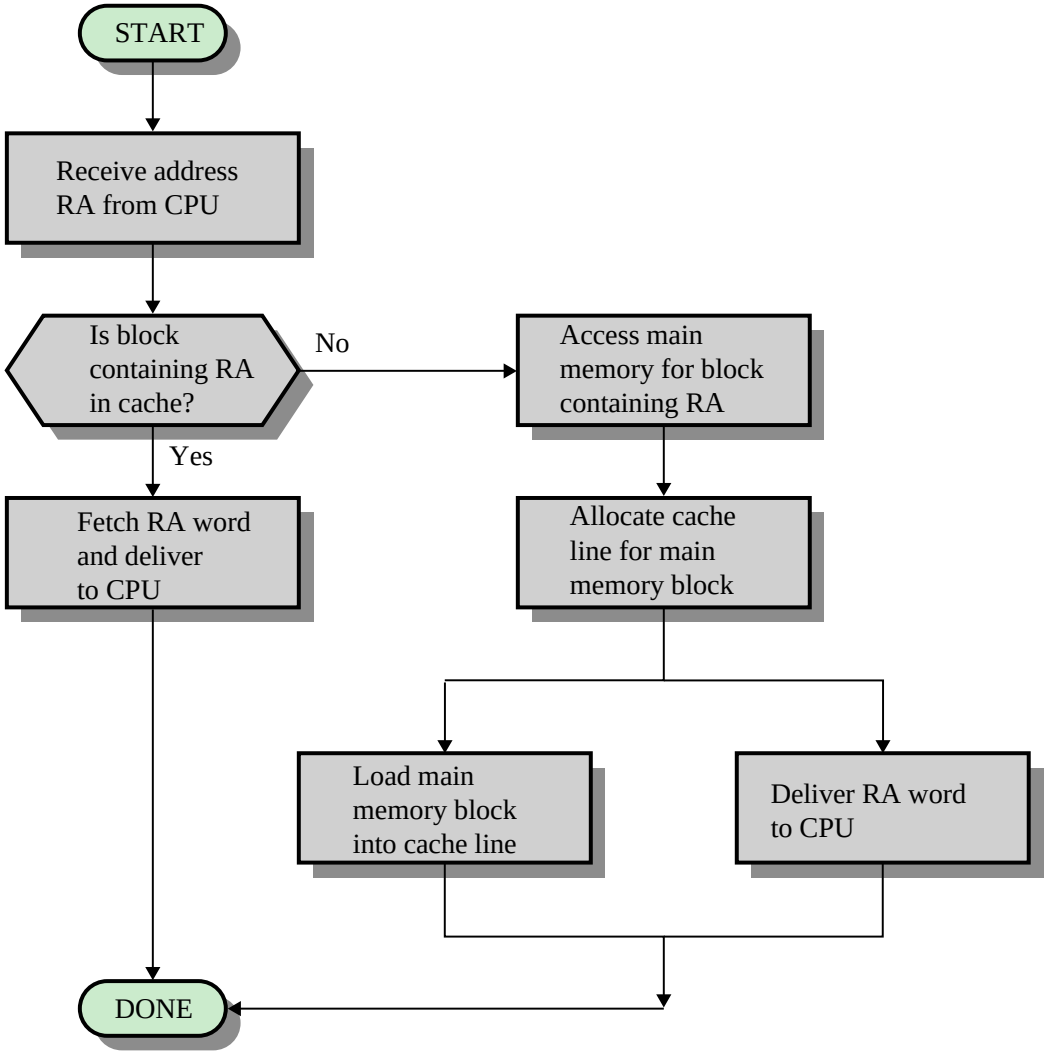


Figure 5.3
Cache Read Operation



Cache Read Operation

- If a cache miss occurs, two things must be accomplished:
  1. the block containing the word must be loaded in to the cache,
  2. and the word must be delivered to the processor.
- When a block is brought into a cache in the event of a miss, the block is generally not transferred in a single event.
- Typically, the transfer size between cache and main memory is less than the line size, with 128 bytes being a typical line size and a cache main memory transfer size of 64 bits (2 bytes).
- To improve performance, the **critical word first** technique is commonly used. When there is a cache miss, the hardware requests the missed word first from memory and sends it to the processor as soon as it arrives. This enables the processor to continue execution while filling the rest of the words in the block.

Elements of Cache Design

Table 5.1
Elements of Cache Design

Cache Addresses
- Logical
- Physical

Cache Size

Mapping Function
- Direct
- Associative
- Set associative

Replacement Algorithm
- Least recently used (LRU)
- First in first out (FIFO)
- Least frequently used (LFU)
- Random

Write Policy
- Write through
- Write back

Line Size

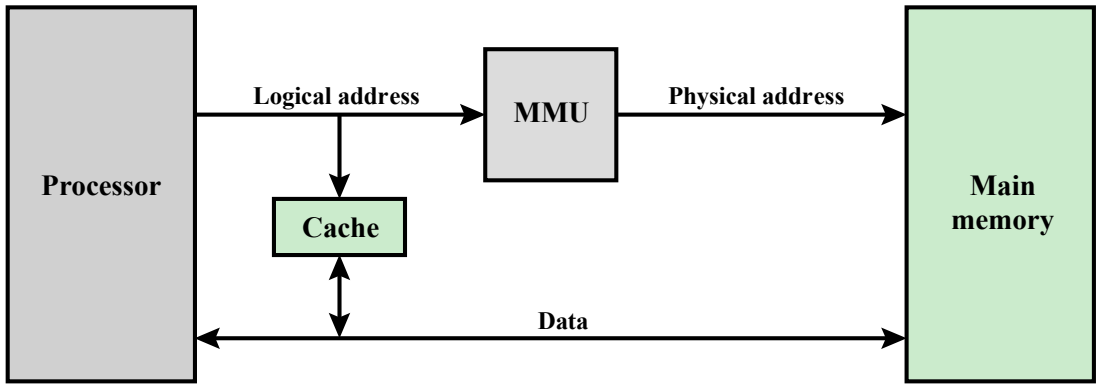
Number of Caches
- Single or two level
- Unified or split

Cache Addresses

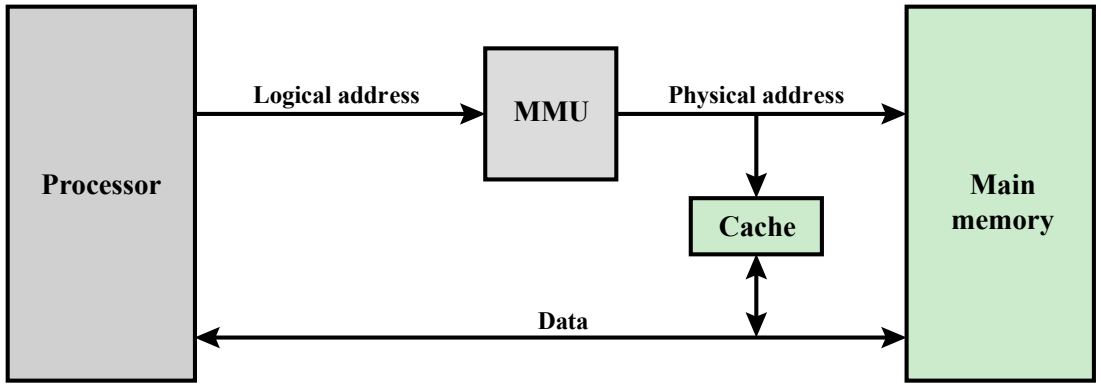
Virtual Memory

- Virtual memory
  - Facility that allows programs to address memory from a logical point of view, without regard to the amount of main memory physically available
  - When used, the address fields of machine instructions contain virtual addresses
  - For reads to and writes from main memory, a hardware memory management unit (MMU) translates each virtual address into a physical address in main memory

Figure 5.5
Logical and Physical Caches



(a) Logical Cache



(b) Physical Cache

Cache Addresses – Virtual Memory

- When virtual addresses are used, the system designer may choose to place the cache between the processor and the MMU or between the MMU and main memory (Figure 5.5).
  - A logical cache, also known as a **virtual cache**, stores data using virtual addresses. The processor accesses the cache directly, without going through the MMU.
  - A physical cache stores data using main memory physical addresses.

Cache Addresses – Virtual Memory

- Advantage of the logical cache
  - cache access speed is faster than for a physical cache, because the cache can respond before the MMU performs an address translation.
- Disadvantage: most virtual memory systems supply each application with the same virtual memory address space.
  - Each application sees a virtual memory that starts at address 0.
  - The cache memory must therefore be completely flushed with each application context switch, or extra bits must be added to each line of the cache to identify which virtual address space this address refers to.

Table 5.1
Elements of Cache Design

Cache Addresses	**Write Policy**
Logical	Write through
Physical	Write back
Cache Size	**Line Size**
Mapping Function	**Number of Caches**
Direct	Single or two level
Associative	Unified or split
Set associative	
Replacement Algorithm	
Least recently used (LRU)	
First in first out (FIFO)	
Least frequently used (LFU)	
Random	

Cache Size

- Preferable for the size of the cache to be:
  - Small enough so that the overall average cost per bit is close to that of main memory alone
  - Large enough so that the overall average access time is close to that of the cache alone
- Motivations for minimizing cache size:
  - The larger the cache, the larger the number of gates involved in addressing the cache resulting in large caches being slightly slower than small ones
  - The available chip and board area also limits cache size
- Because the performance of the cache is very sensitive to the nature of the workload, it is impossible to arrive at a single "optimum" cache size

Table 5.2
Cache Sizes of Some Processors

Processor	Type	Year of Introduction	L1 Cache[a]	L2 cache	L3 Cache
IBM 360/85	Mainframe	1968	16 to 32 kB	–	–
PDP-11/70	Minicomputer	1968	1 kB	–	–
IBM 3033	Mainframe	1968	64 kB	–	–
IBM 3090	Mainframe	1968	128 to 256 kB	–	–
Intel 80486	PC	1968	8 kB	–	–
Pentium	PC	1968	8 kB/8 kB	256 to 512 kB	–
PowerPC 620	PC	1968	32 kB/32 kB	–	–
IBM S/390 G6	Mainframe	1968	256 kB	8 MB	–
Pentium 4	PC/server	1968	8 kB/8 kB	256 kB	–
Itanium	PC/server	1968	16 kB/16 kB	96 kB	4 MB
Itanium 2	PC/server	1968	32 kB	256 kB	6 MB
IBM POWER5	High-end server	1968	64 kB	1.9 MB	36 MB
CRAY XD-1	Supercomputer	1968	64 kB/64 kB	1 MB	–
IBM POWER6	PC/server	1968	64 kB/64 kB	4 MB	32 MB
IBM z10	Mainframe	1968	64 kB/128 kB	3 MB	24-48 MB
Intel Core i7 EE 990	Workstaton/ Server	1968	6 × 32 kB/32 kB	6 × 1.5 MB	12 MB
IBM zEnterprise 196	Mainframe/ Server	1968	24 × 64 kB/128 kB	24 × 1.5 MB	24 MB L3 192 MB L4
IBM z13	Mainframe/ server	1968	24 × 96 kB/128 kB	24 × 2 MB/2 MB	64 MB L3 480 MB L4
Intel Core i0-7900X	Workstation/ server	1968	8 × 32 kB/32 kB	8 × 1 MB	14 MB

[a] Two values separated by a slash refer to instruction and data caches.

(Table can be found on page 145 in the textbook.)

Table 5.1
Elements of Cache Design

Cache Addresses
- Logical
- Physical

Cache Size

Mapping Function
- Direct
- Associative
- Set associative

Replacement Algorithm
- Least recently used (LRU)
- First in first out (FIFO)
- Least frequently used (LFU)
- Random

Write Policy
- Write through
- Write back

Line Size

Number of Caches
- Single or two level
- Unified or split

Cache Access

Recall - There are fewer cache lines than main memory blocks

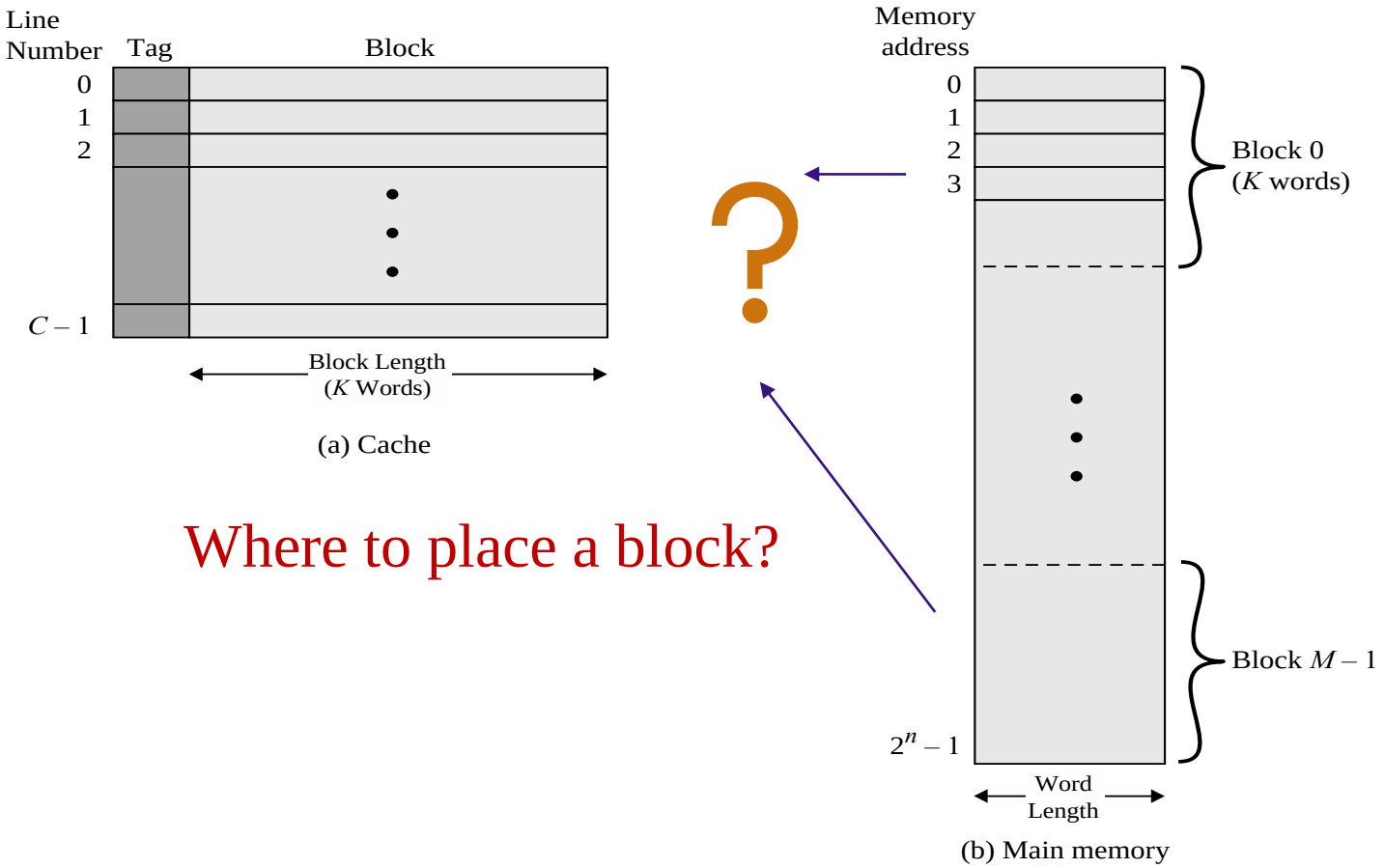
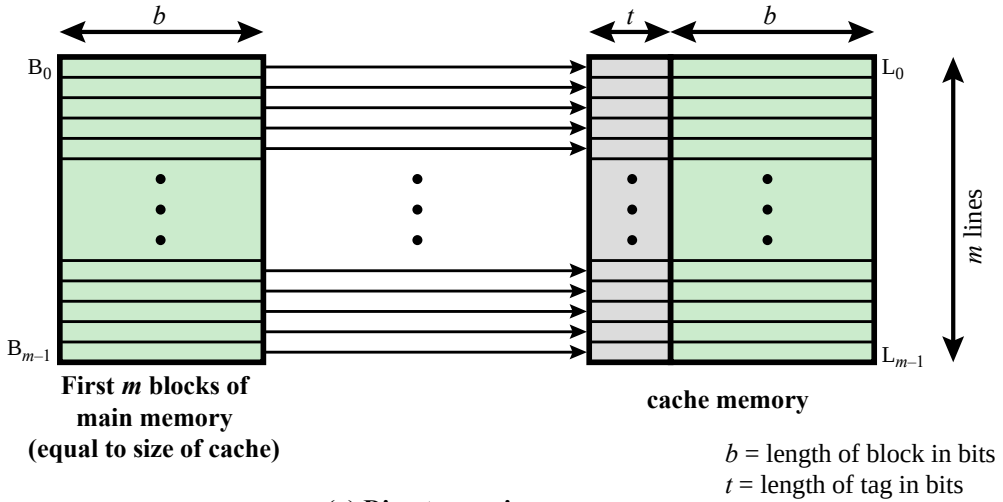


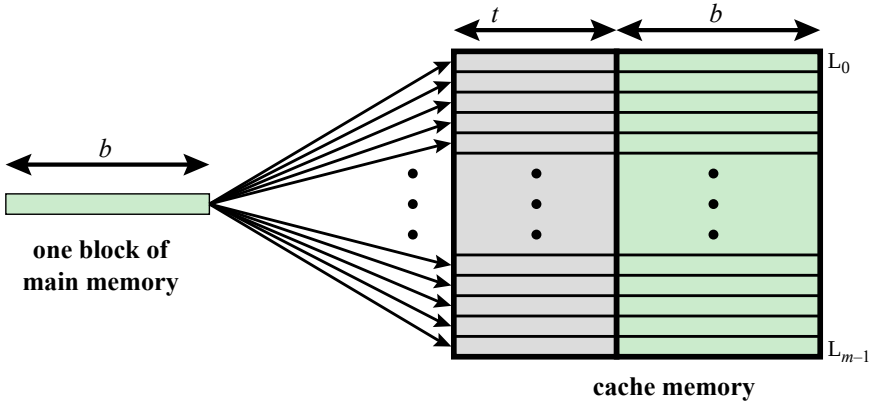
Table 5.3
Cache Access Methods

Method	Organization	Mapping of Main Memory Blocks to Cache	Access using Main Memory Address
Direct Mapped	Sequence of *m* lines	Each block of main memory maps to one unique line of cache.	*Line* portion of address used to access cache line; *Tag* portion used to check for hit on that line.
Fully Associative	Sequence of *m* lines	Each block of main memory can map to any line of cache.	*Tag* portion of address used to check every line for hit on that line.
Set Associative	Sequence of *m* lines organized as *v* sets of *k* lines each ($m = v \times k$)	Each block of main memory maps to one unique cache set.	*Line* portion of address used to access cache set; *Tag* portion used to check every line in that set for hit on that line.

Figure 5.6
Mapping from Main Memory to Cache: Direct and Associative



(a) Direct mapping



(b) Associative mapping

Direct Mapping

- Main Memory Address ➔ mapped ➔ Cache Access

- Each main memory block is mapped to one cache block ONLY.

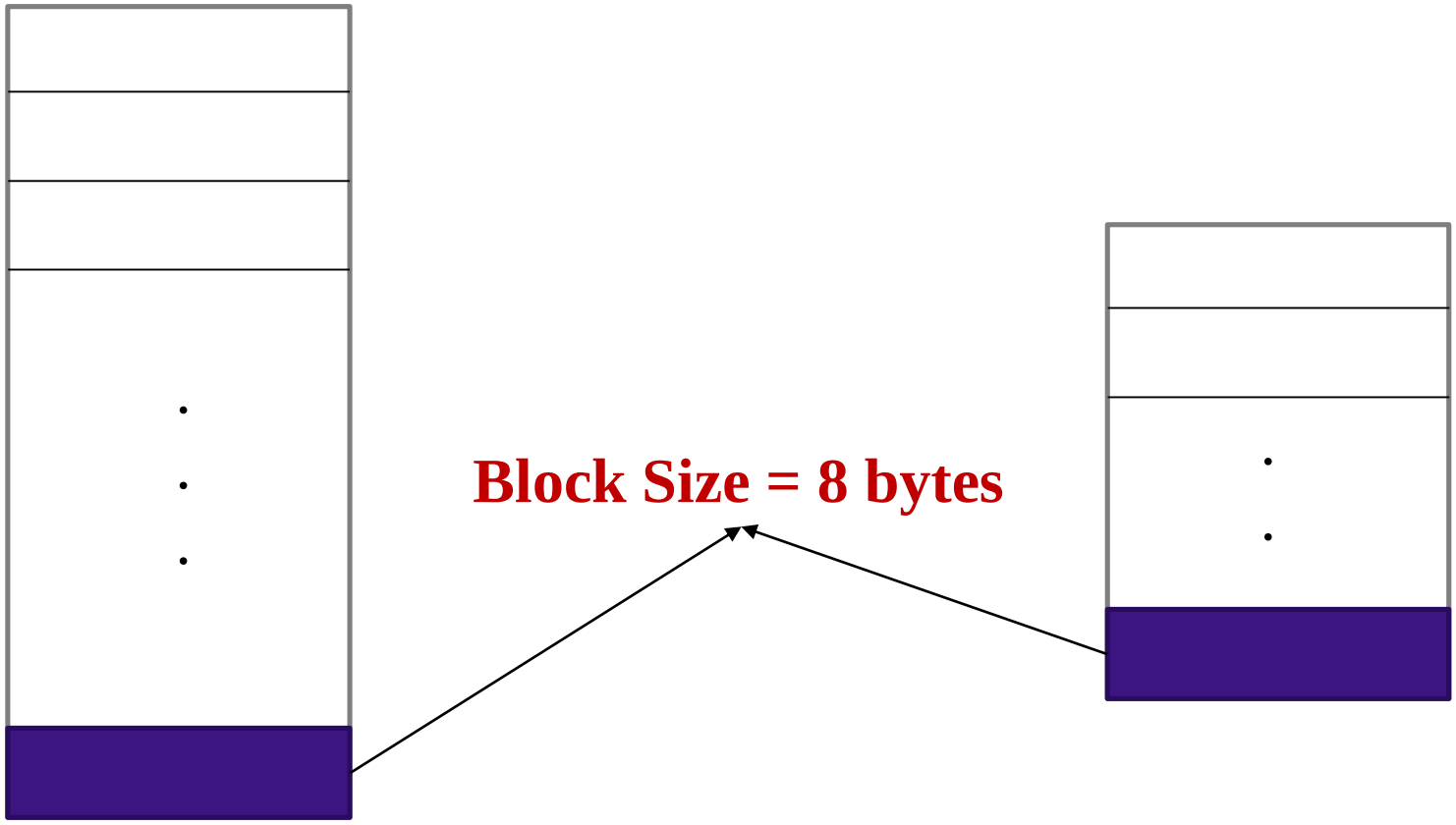
- Multiple main memory blocks can be mapped to the same cache block (remember fewer cache blocks means no way and no point of 1-1 mapping)

Direct Mapping

Main memory (128 bytes)

Needs 7-bits address

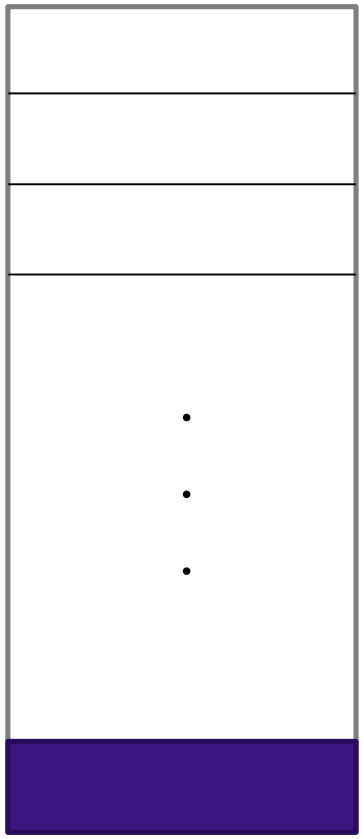
Cache memory (32 bytes)



Direct Mapping

Main memory (128 bytes)

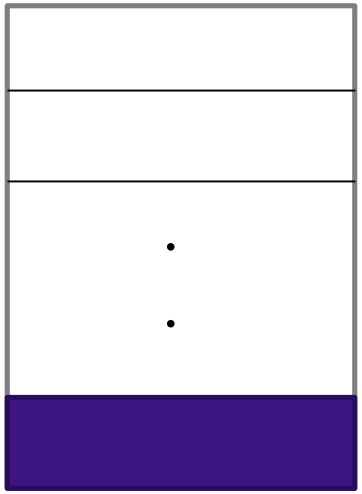
Needs 7-bits address



Cache memory (32 bytes)

Block Size = 8 bytes

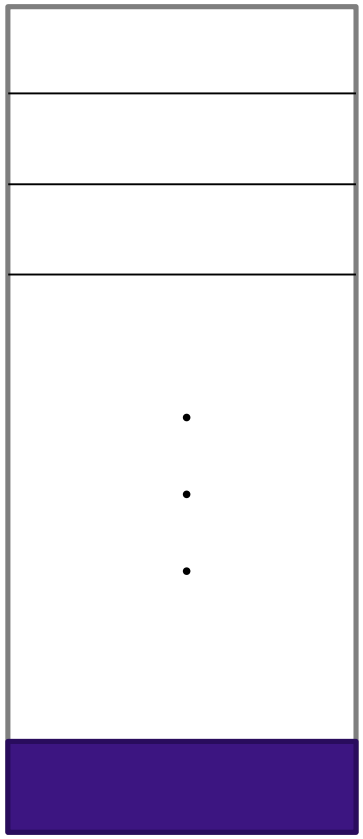
What is the Number of blocks in the main memory?



Direct Mapping

Main memory (128 bytes)

Needs 7-bits address

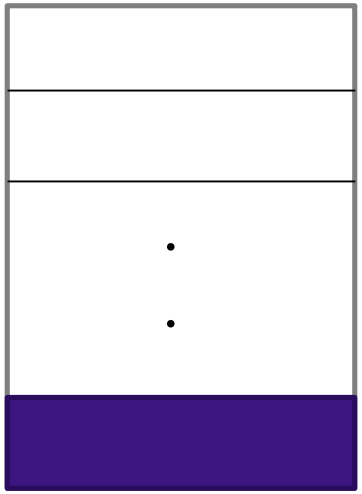


Cache memory (32 bytes)

Block Size = 8 bytes

What is the Number of blocks in the main memory?

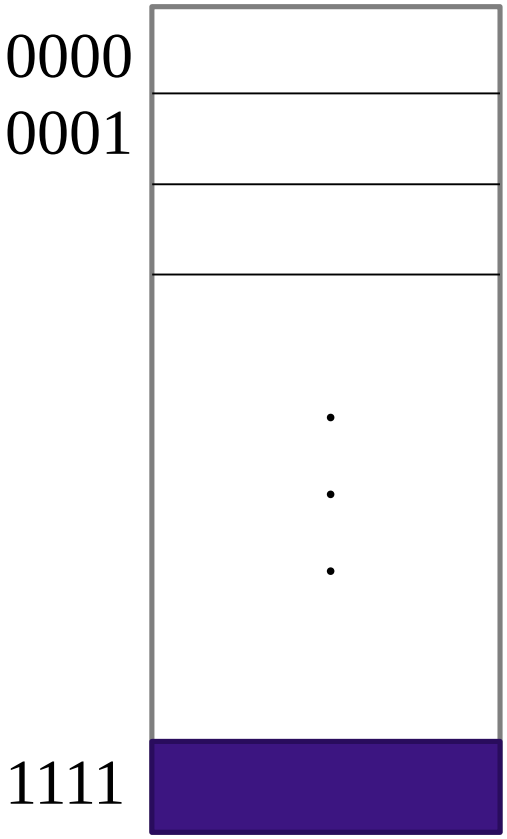
MBN = 128 / 8 = 16



Direct Mapping

Main memory (128 bytes)

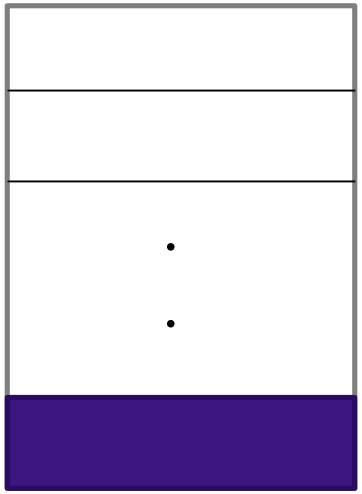
Needs 7-bits address



Block Size = 8 bytes

MBN = 128 / 8 = 16 (2^4)

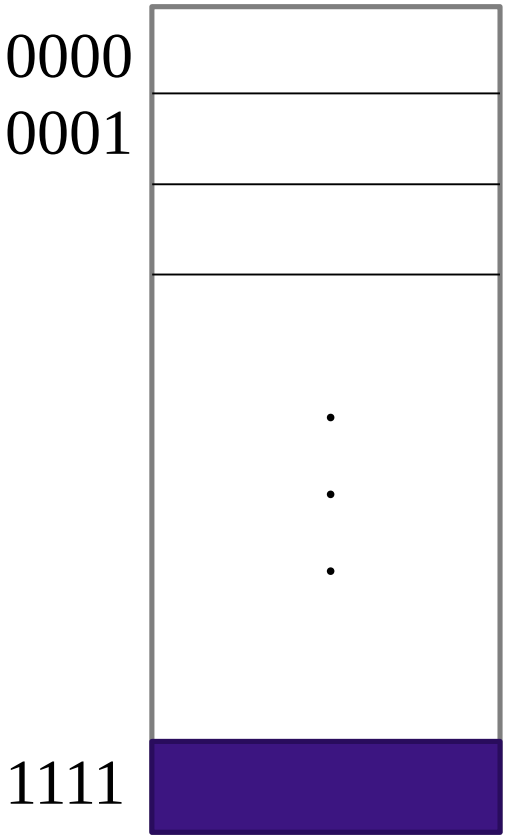
Cache memory (32 bytes)



Direct Mapping

Main memory (128 bytes)

Needs 7-bits address

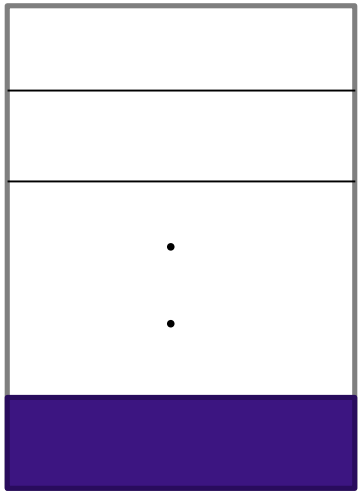


Cache memory (32 bytes)

Block Size = 8 bytes

MBN = 128 / 8 = 16 (2^4)

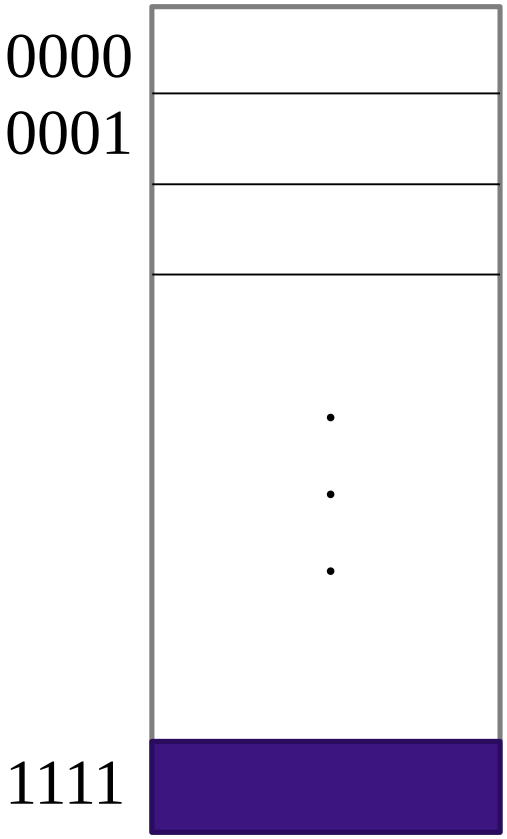
What is the number of cache blocks?



Direct Mapping

Main memory (128 bytes)

Needs 7-bits address



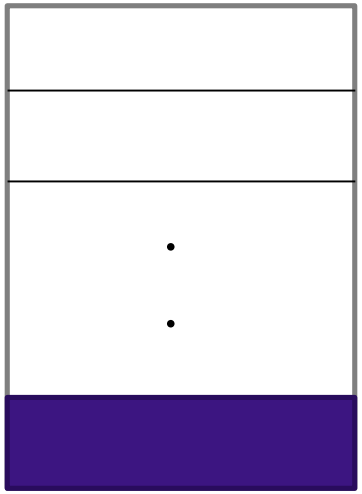
Block Size = 8 bytes

MBN = 128 / 8 = 16 (2^4)

What is the number of cache blocks?

CBN = 32 / 8 = 4 (2^2)

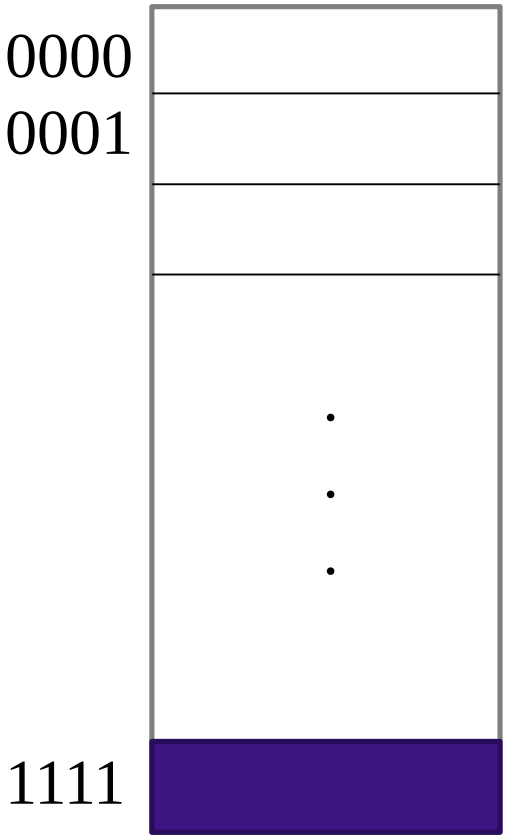
Cache memory (32 bytes)



Direct Mapping

Main memory (128 bytes)

Needs 7-bits address

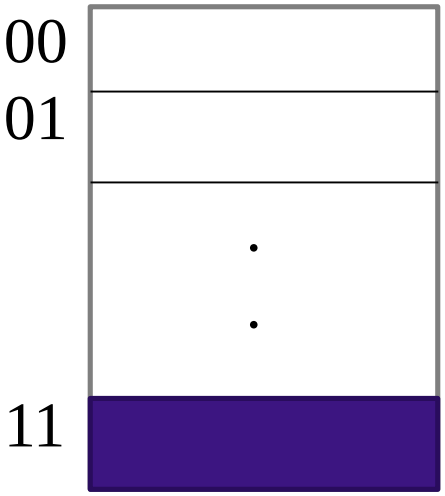


Block Size = 8 bytes

MBN = 128 / 8 = 16 (2^4)

CBN = 32 / 8 = 4 (2^2)

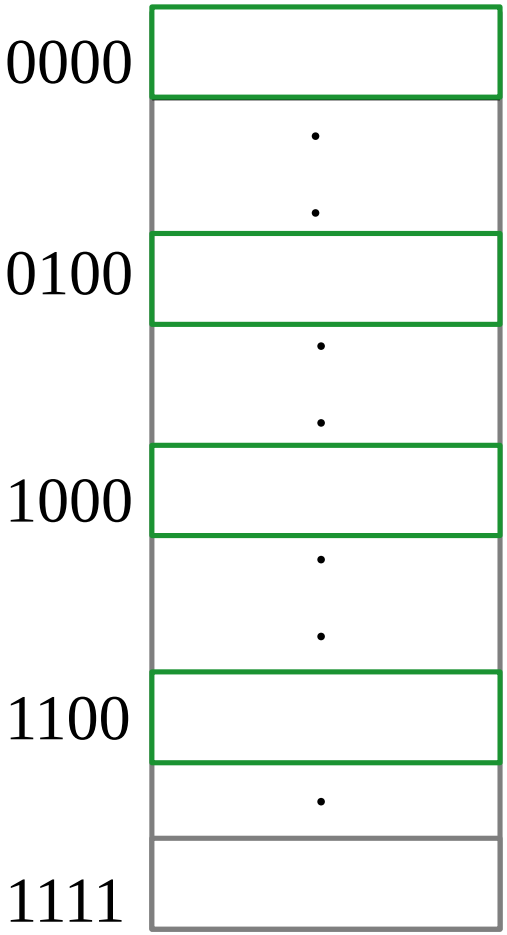
Cache memory (32 bytes)



Direct Mapping

Main memory (128 bytes)

Needs 7-bits address



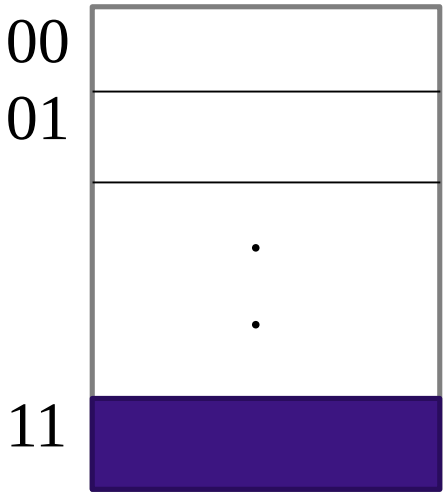
Block Size = 8 bytes

MBN = 128 / 8 = 16 (2^4)

CBN = 32 / 8 = 4 (2^2)

How mapping occurs?

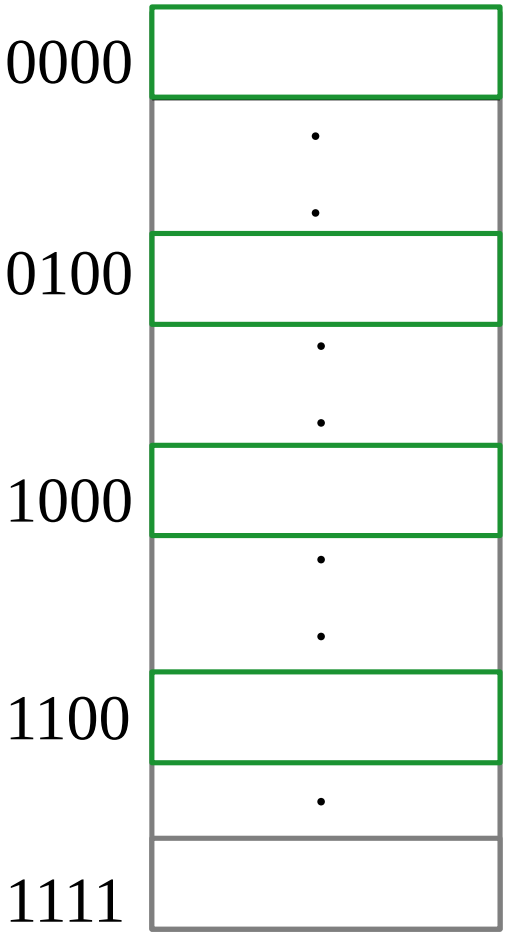
Cache memory (32 bytes)



Direct Mapping

Main memory (128 bytes)

Needs 7-bits address

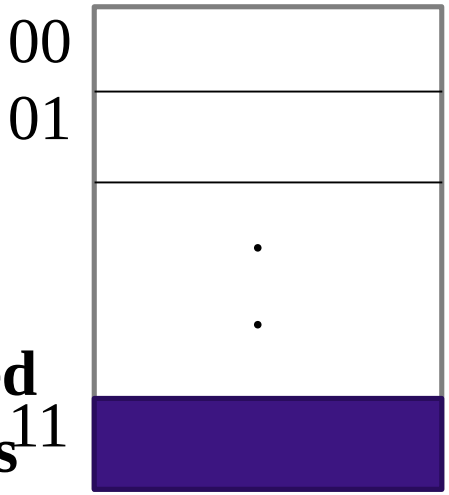


Cache memory (32 bytes)

Block Size = 8 bytes

MBN = 128 / 8 = 16 (2^4)

CBN = 32 / 8 = 4 (2^2)



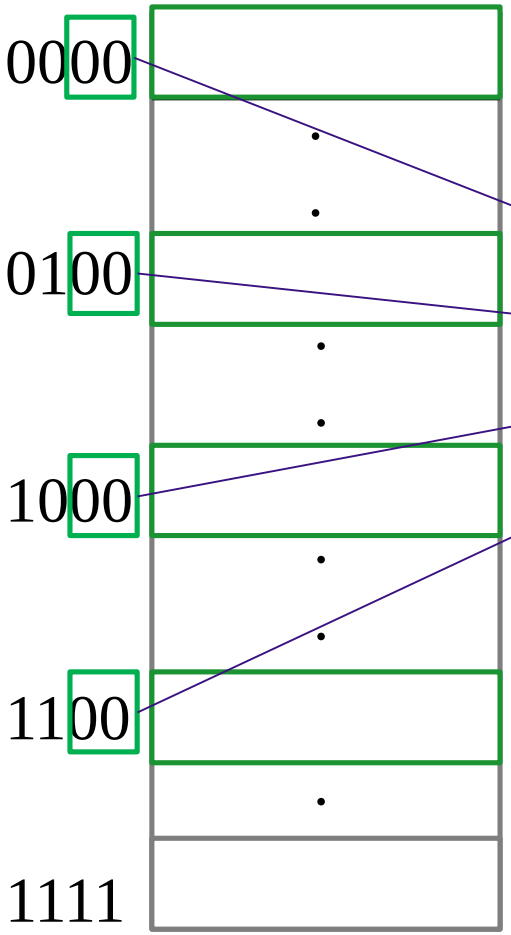
How mapping occurs?

Least significant MB address bits is mapped to the CB address bits

Direct Mapping

Main memory (128 bytes)

Needs 7-bits address

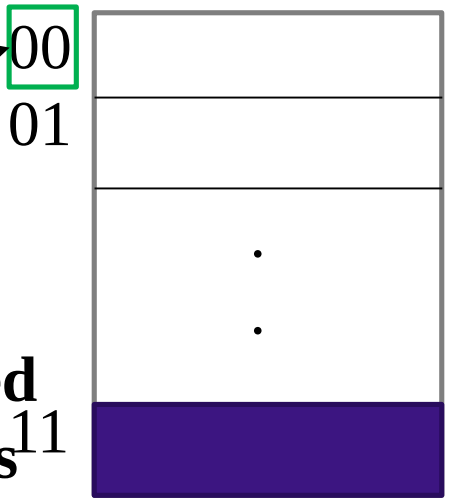


Cache memory (32 bytes)

Block Size = 8 bytes

MBN = 128 / 8 = 16 (2^4)

CBN = 32 / 8 = 4 (2^2)



How mapping occurs?

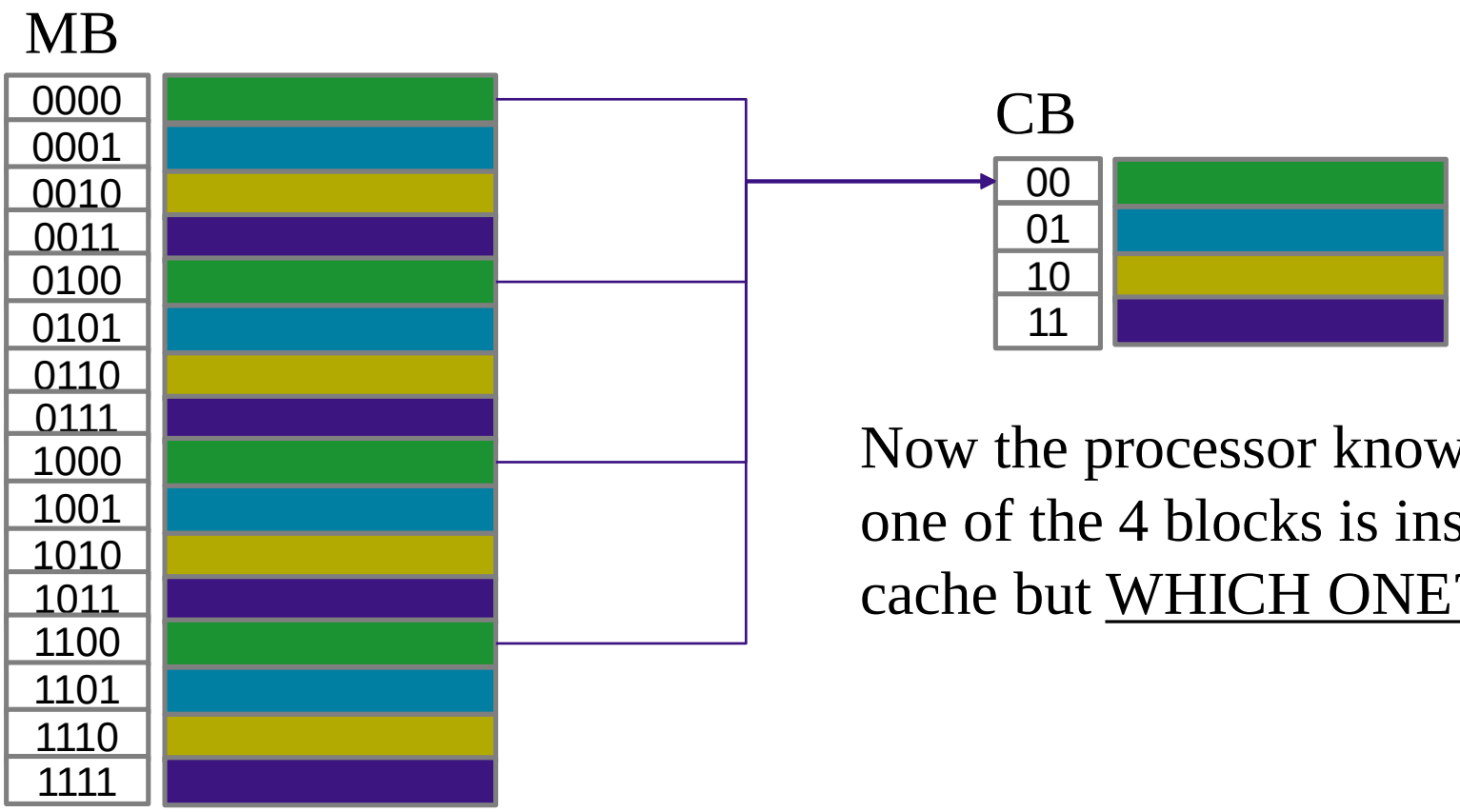
Least significant MB address bits is mapped to the CB address bits

Direct Mapping

Main memory (128 bytes)

Needs 7-bits address

Cache memory (32 bytes)



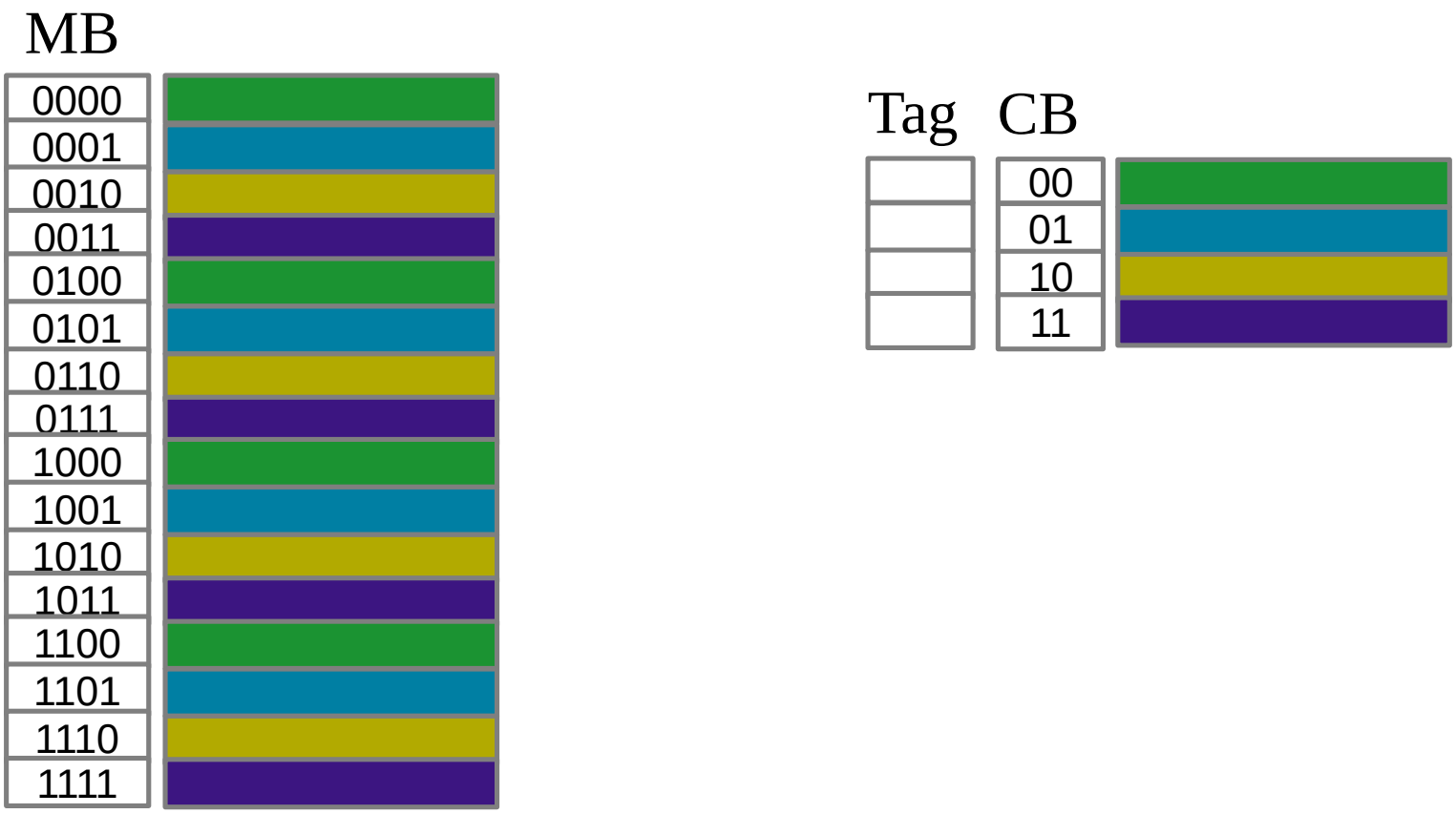
Now the processor knows that one of the 4 blocks is inside the cache but WHICH ONE?

Direct Mapping

Main memory (128 bytes)

Needs 7-bits address

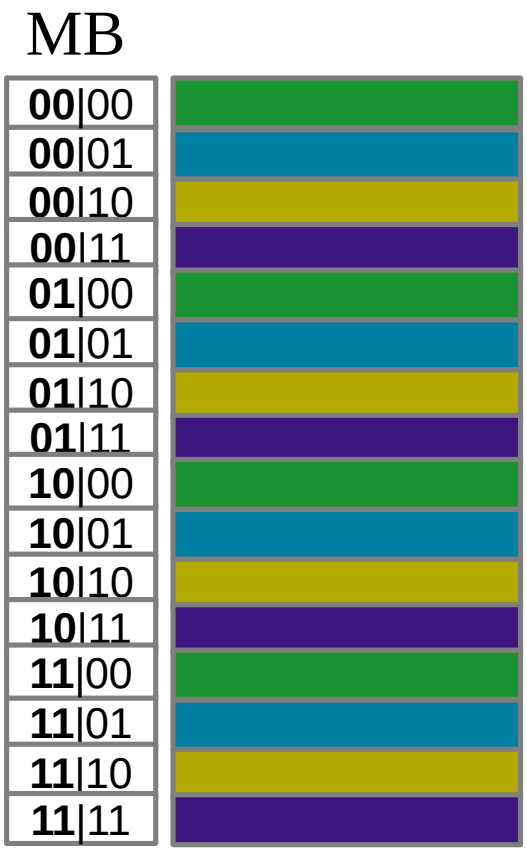
Cache memory (32 bytes)



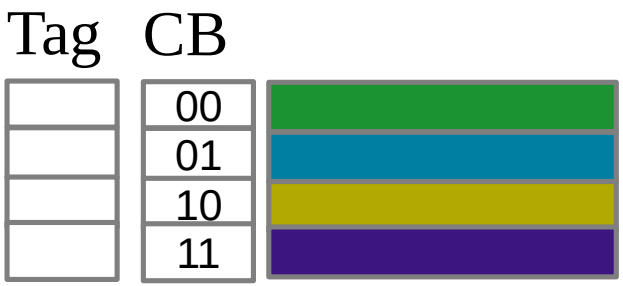
Direct Mapping

Main memory (128 bytes)

Needs 7-bits address



Cache memory (32 bytes)

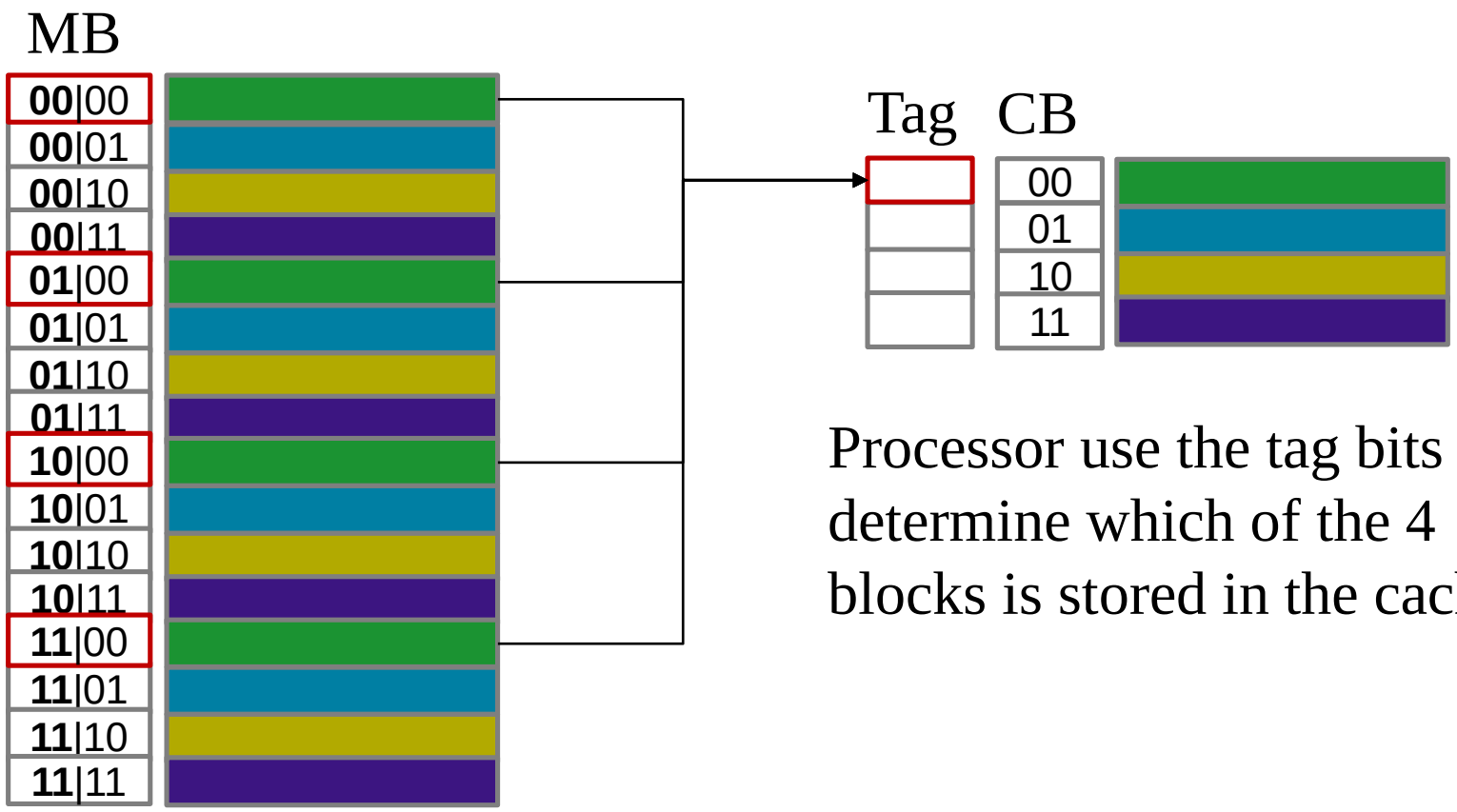


Direct Mapping

Main memory (128 bytes)

Needs 7-bits address

Cache memory (32 bytes)



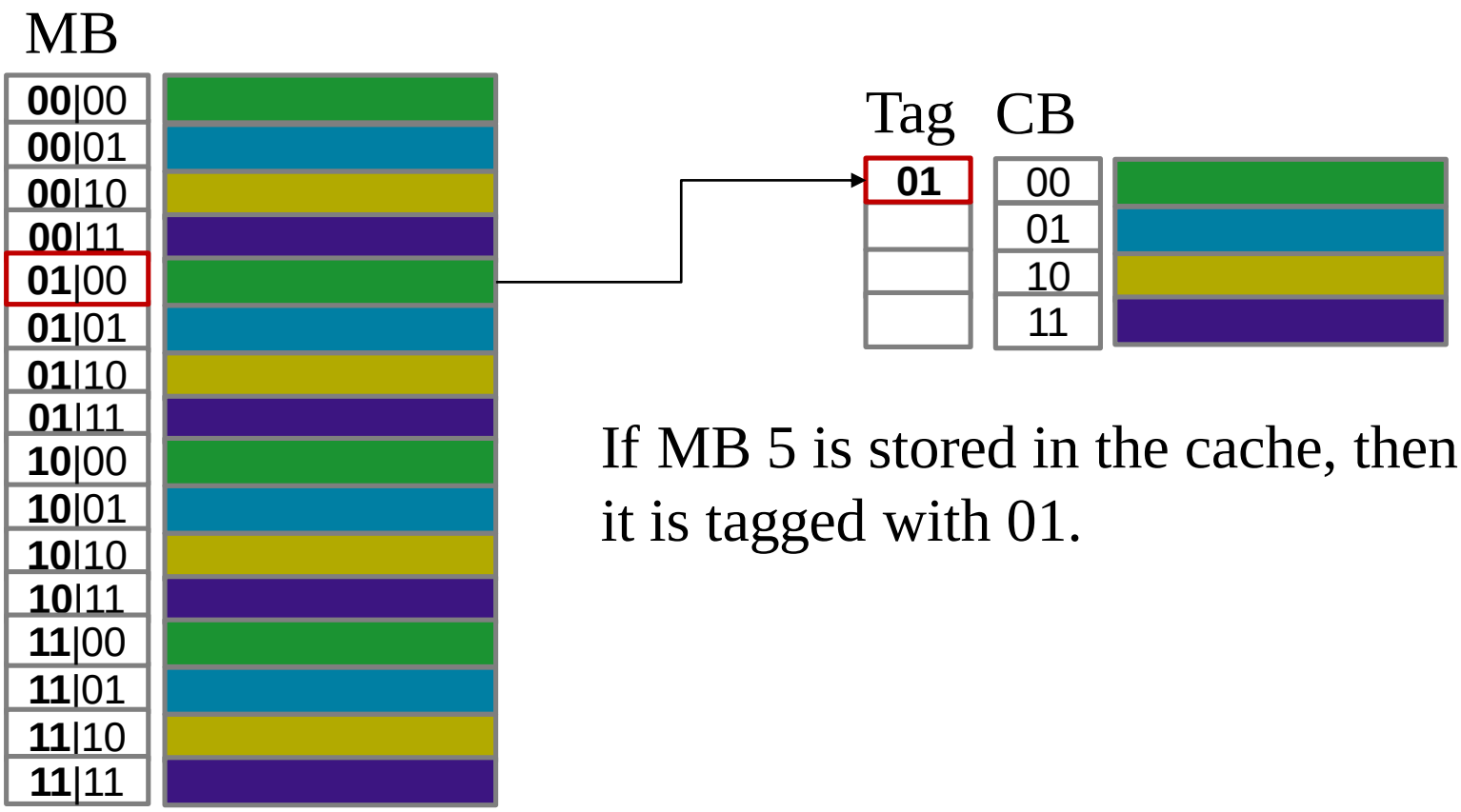
Processor use the tag bits to determine which of the 4 blocks is stored in the cache

Direct Mapping

Main memory (128 bytes)

Needs 7-bits address

Cache memory (32 bytes)



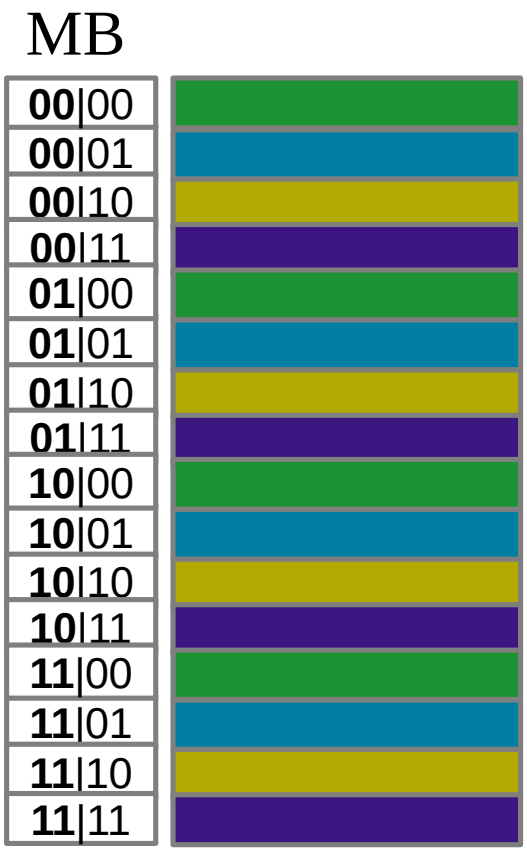
If MB 5 is stored in the cache, then it is tagged with 01.

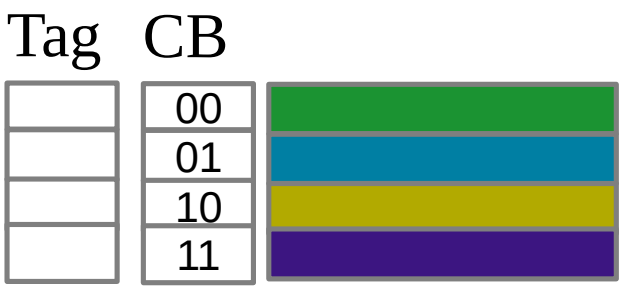
Direct Mapping

Main memory (128 bytes)

Needs 7-bits address

Cache memory (32 bytes)



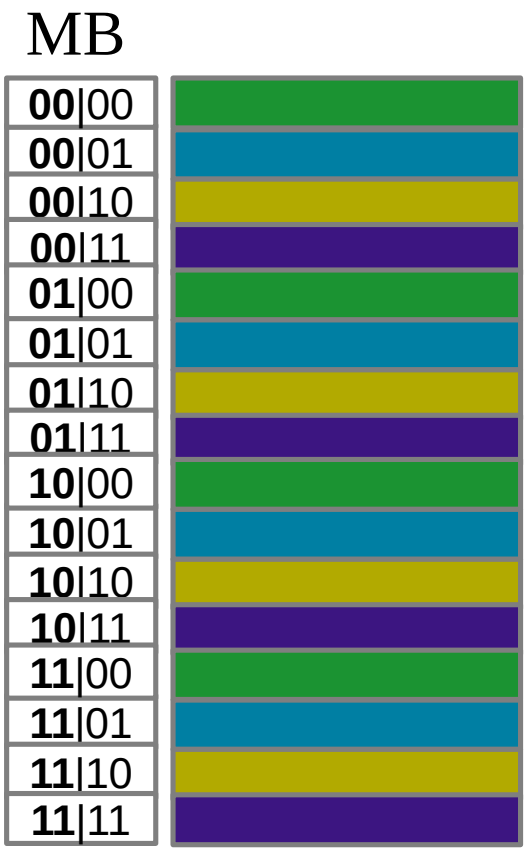


Remember main memory address is 7 bits. We used only 4 bits, what about the last 3?

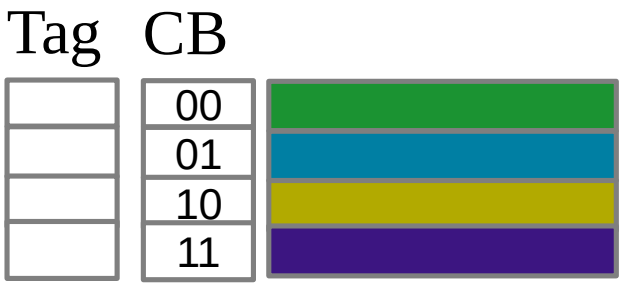
Direct Mapping

Main memory (128 bytes)

Needs 7-bits address

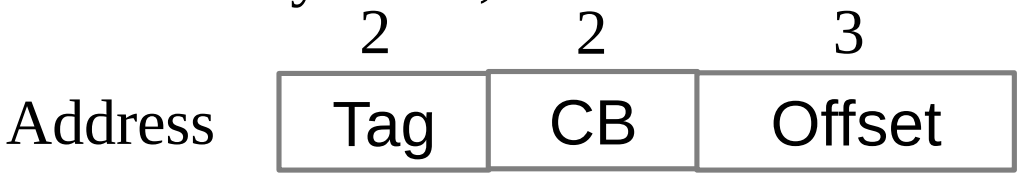


Cache memory (32 bytes)



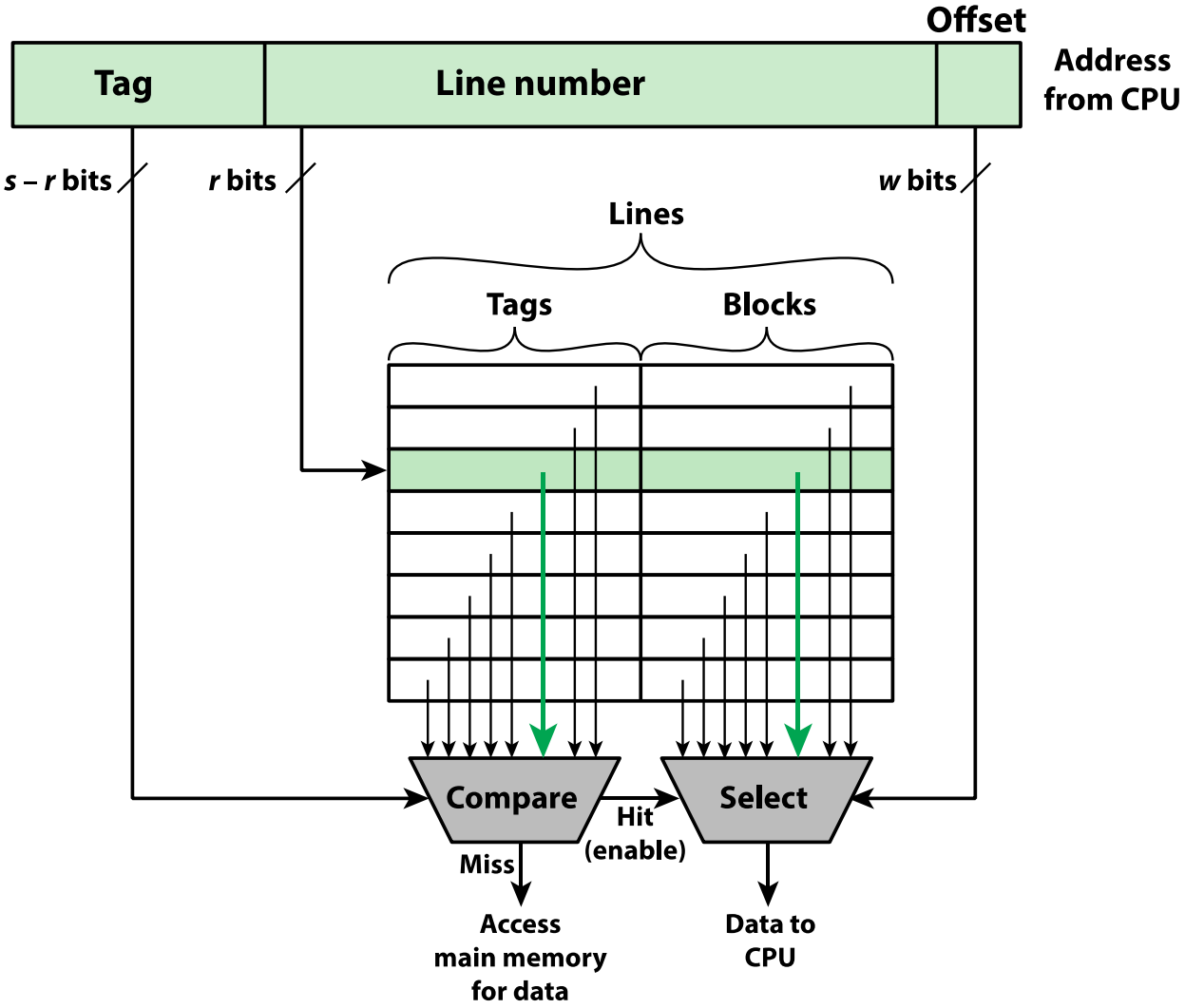
Remember main memory address is 7 bits.

We used only 4 bits, what about the last 3?



Processor uses the offset access a specific byte within the block!

Figure 5.7
Direct-Mapping Cache Organization



Example

- Consider a 4 block cache memory. Calculate the number of hits and misses based on the following main memory block number access order:
  - 5, 4, 6, 3, 2, 1, 4, 1, 5, 3, 0

Assume the cache is initially empty & the direct mapping is: *line number = block number % number of lines*

Example

- Consider a 4 block cache memory. Calculate the number of hits and misses based on the following main memory block number access order:
  - 5, 4, 6, 3, 2, 1, 4, 1, 5, 3, 0

Assume the cache is initially empty & the direct mapping is: *line number = block number % number of lines*

Miss: 0

Hit: 0

	Cache
00	
01	
10	
11	

Example

- Consider a 4 block cache memory. Calculate the number of hits and misses based on the following main memory block number access order:
  - **5**, 4, 6, 3, 2, 1, 4, 1, 5, 3, 0

Assume the cache is initially empty & the direct mapping is: *line number = block number % number of lines*

Miss: 1

Hit: 0

	Cache
00	
01	5
10	
11	

Example

- Consider a 4 block cache memory. Calculate the number of hits and misses based on the following main memory block number access order:
  - 5, **4**, 6, 3, 2, 1, 4, 1, 5, 3, 0

Assume the cache is initially empty & the direct mapping is: *line number = block number % number of lines*

Miss: 2

Hit: 0

	Cache
00	4
01	5
10	
11	

Example

- Consider a 4 block cache memory. Calculate the number of hits and misses based on the following main memory block number access order:
  - 5, 4, **6**, 3, 2, 1, 4, 1, 5, 3, 0

Assume the cache is initially empty & the direct mapping is: *line number = block number % number of lines*

Miss: 3

Hit: 0

	Cache
00	4
01	5
10	6
11	

Example

- Consider a 4 block cache memory. Calculate the number of hits and misses based on the following main memory block number access order:
  - 5, 4, 6, **3**, 2, 1, 4, 1, 5, 3, 0

Assume the cache is initially empty & the direct mapping is: *line number = block number % number of lines*

Miss: 4

Hit: 0

	Cache
00	4
01	5
10	6
11	3

Example

- Consider a 4 block cache memory. Calculate the number of hits and misses based on the following main memory block number access order:
  - 5, 4, 6, 3, **2**, 1, 4, 1, 5, 3, 0

Assume the cache is initially empty & the direct mapping is: *line number = block number % number of lines*

		Cache
Miss: 5	00	4
Hit: 0	01	5
	10	~~6~~ 2
	11	3

Example

- Consider a 4 block cache memory. Calculate the number of hits and misses based on the following main memory block number access order:
  - 5, 4, 6, 3, 2, **1**, 4, 1, 5, 3, 0

Assume the cache is initially empty & the direct mapping is: *line number = block number % number of lines*

Miss: 6

Hit: 0

	Cache
00	4
01	~~5~~ 1
10	~~6~~ 2
11	3

Example

- Consider a 4 block cache memory. Calculate the number of hits and misses based on the following main memory block number access order:
  - 5, 4, 6, 3, 2, 1, **4**, 1, 5, 3, 0

Assume the cache is initially empty & the direct mapping is: *line number = block number % number of lines*

Miss: 6

Hit: 1

	Cache
00	4
01	~~5~~ 1
10	~~6~~ 2
11	3

Example

- Consider a 4 block cache memory. Calculate the number of hits and misses based on the following main memory block number access order:
  - 5, 4, 6, 3, 2, 1, 4, **1**, 5, 3, 0

Assume the cache is initially empty & the direct mapping is: *line number = block number % number of lines*

Miss: 6

Hit: 2

	Cache
00	4
01	~~5~~ 1
10	~~6~~ 2
11	3

Example

- Consider a 4 block cache memory. Calculate the number of hits and misses based on the following main memory block number access order:
  - 5, 4, 6, 3, 2, 1, 4, 1, **5**, 3, 0

Assume the cache is initially empty & the direct mapping is: *line number = block number % number of lines*

Miss: 7

Hit: 2

	Cache
00	4
01	~~5~~ ~~1~~ 5
10	~~6~~ 2
11	3

Example

- Consider a 4 block cache memory. Calculate the number of hits and misses based on the following main memory block number access order:
  - 5, 4, 6, 3, 2, 1, 4, 1, 5, **3**, 0

Assume the cache is initially empty & the direct mapping is: *line number = block number % number of lines*

Miss: 7

Hit: 3

	Cache
00	4
01	~~5~~ ~~1~~ 5
10	~~6~~ 2
11	3

Example

- Consider a 4 block cache memory. Calculate the number of hits and misses based on the following main memory block number access order:
  - 5, 4, 6, 3, 2, 1, 4, 1, 5, 3, **0**

Assume the cache is initially empty & the direct mapping is: *line number = block number % number of lines*

Miss: 8

Hit: 3

	Cache
00	~~4~~ 0
01	~~5~~ ~~1~~ 5
10	~~6~~ 2
11	3

Copyright