Computer Organization and Architecture

CH 04: CACHE MEMORY

Content

CH 04
Computer Memory System Overview
Cache Memory Principles
Elements of Cache Design
~~Pentium 4 Cache Organization~~
~~ARM Cache Organization~~

Computer Memory System Overview

- Key Characteristics of Computer Memory Systems

Location
- Internal (e.g. processor registers, main memory, cache)
- External (e.g. optical disks, magnetic disks, tapes)

Capacity
- Number of words
- Number of bytes

Unit of Transfer
- Word
- Block

Access Method
- Sequential
- Direct
- Random
- Associative

Performance
- Access time
- Cycle time
- Transfer rate

Physical Type
- Semiconductor
- Magnetic
- Optical
- Magneto-optical

Physical Characteristics
- Volatile/nonvolatile
- Erasable/nonerasable

Organization
- Memory modules

Computer Memory System Overview

- Transfer rate: rate at which data is transferred into/out of a memory unit.
- For random-access memory

$$\mathrm{T} = \frac{1}{cycle\ time}$$

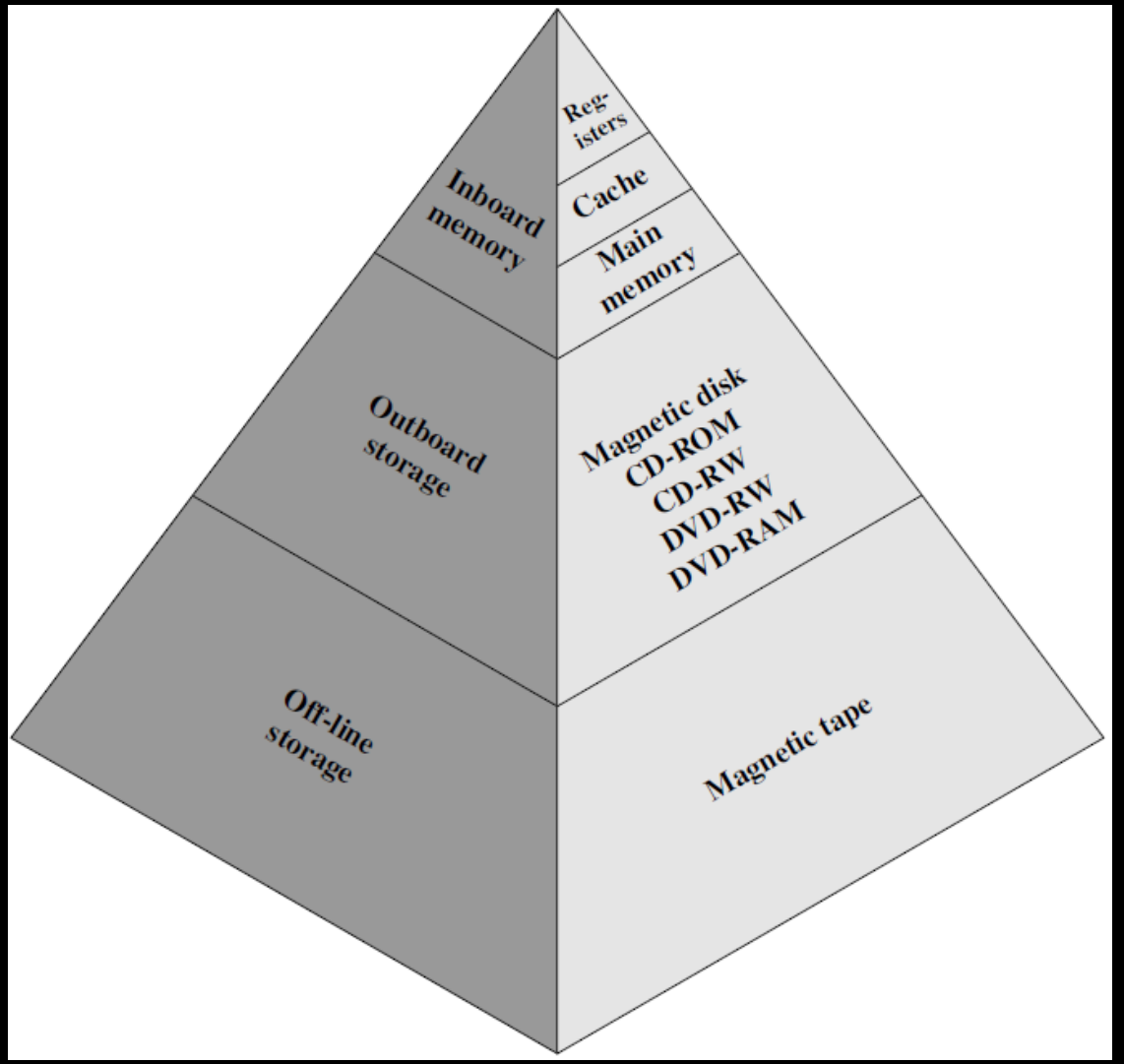
- For non-random-access memory

$$T_N = T_A + \frac{n}{R}$$

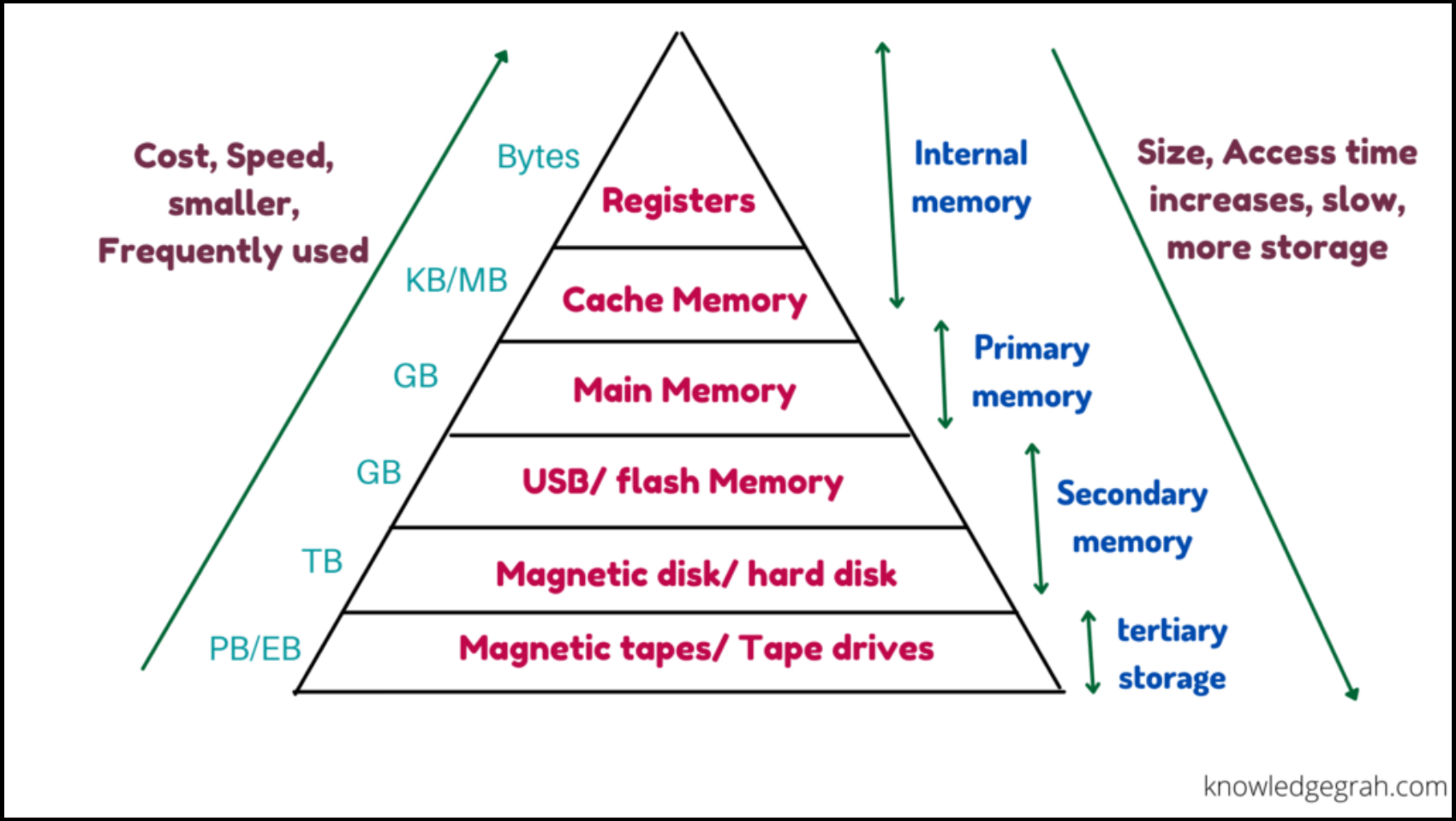
  - T_N Average time to read or write N bits
  - T_A Average access time
  - n Number of bits
  - R Transfer rate, in bits per second (bps)

Computer Memory System Overview

- The Memory Hierarchy is the organization of computers memory into a hierarchy from highest level to lowest level.
- As we go down the hierarchy:
  - Decreasing cost
  - Increasing capacity
  - Increasing access time
  - Decreasing frequency of access of the memory by the processor



Computer Memory System Overview

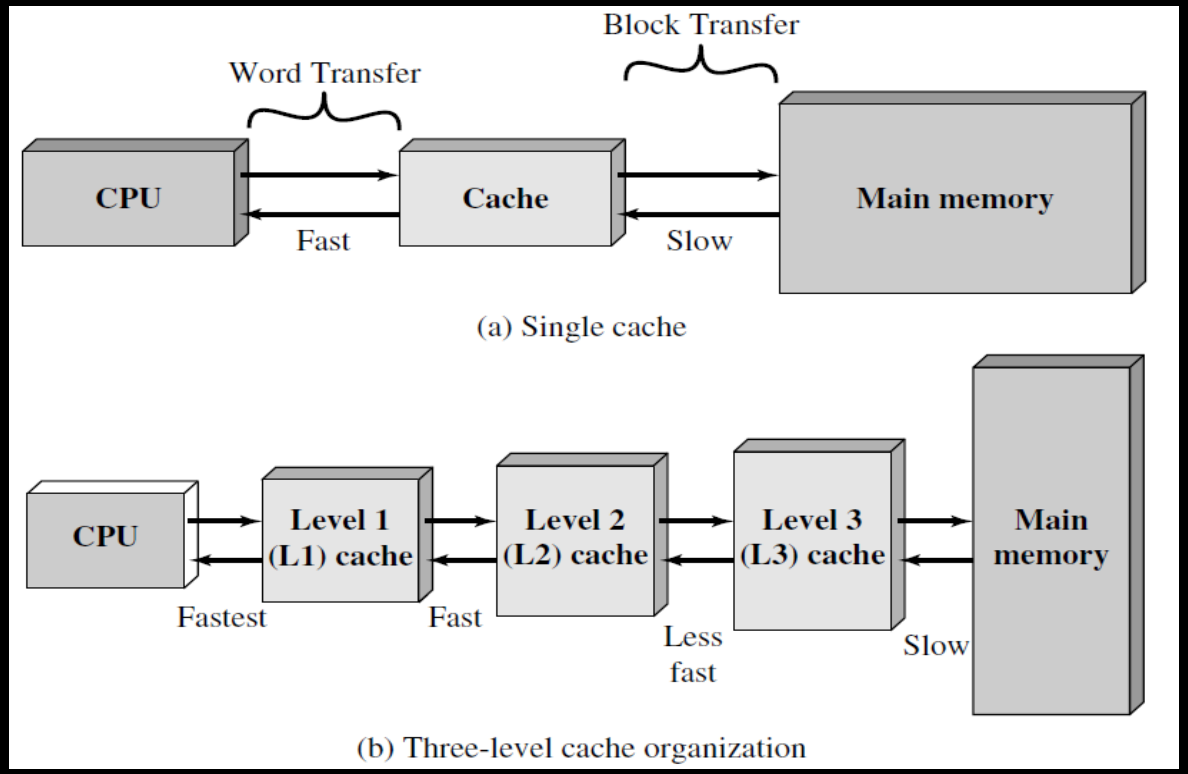


Content

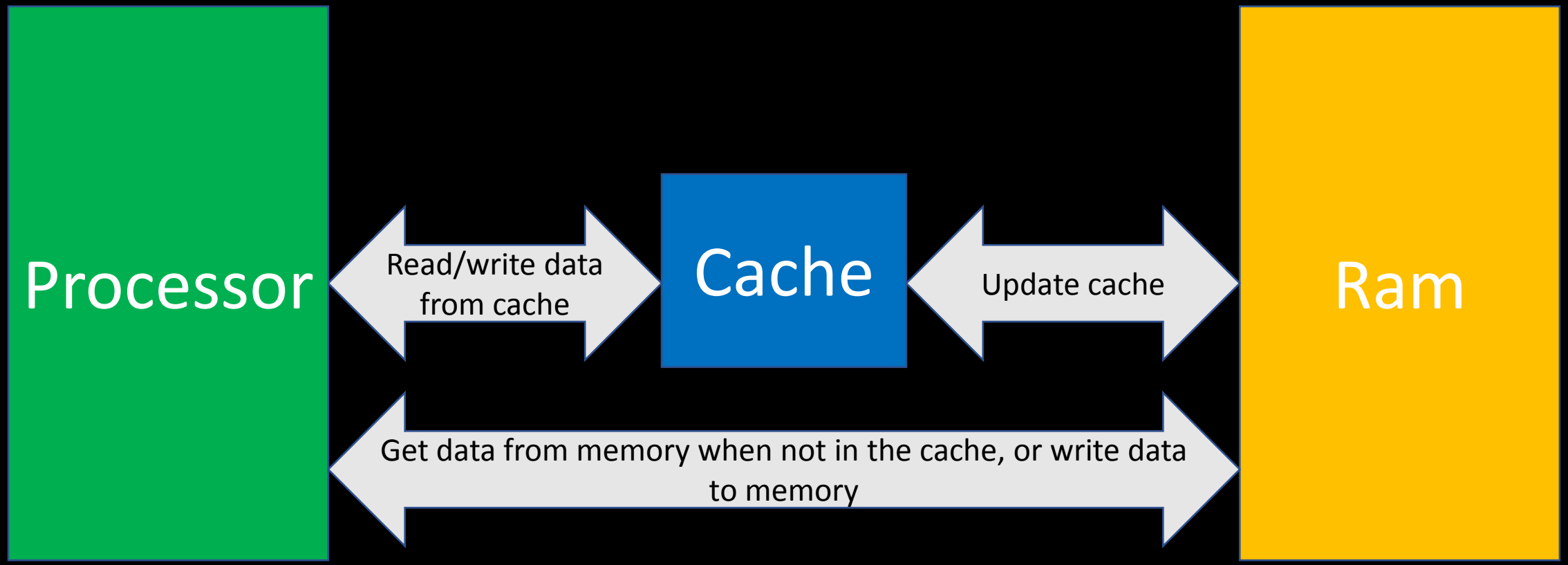
CH 04
Computer Memory System Overview
Cache Memory Principles
Elements of Cache Design
~~Pentium 4 Cache Organization~~
~~ARM Cache Organization~~

Cache Memory Principles

- Cache memory is a small-sized, volatile memory provides high-speed data access to a processor.
  - The cache contains a copy of portions of main memory.



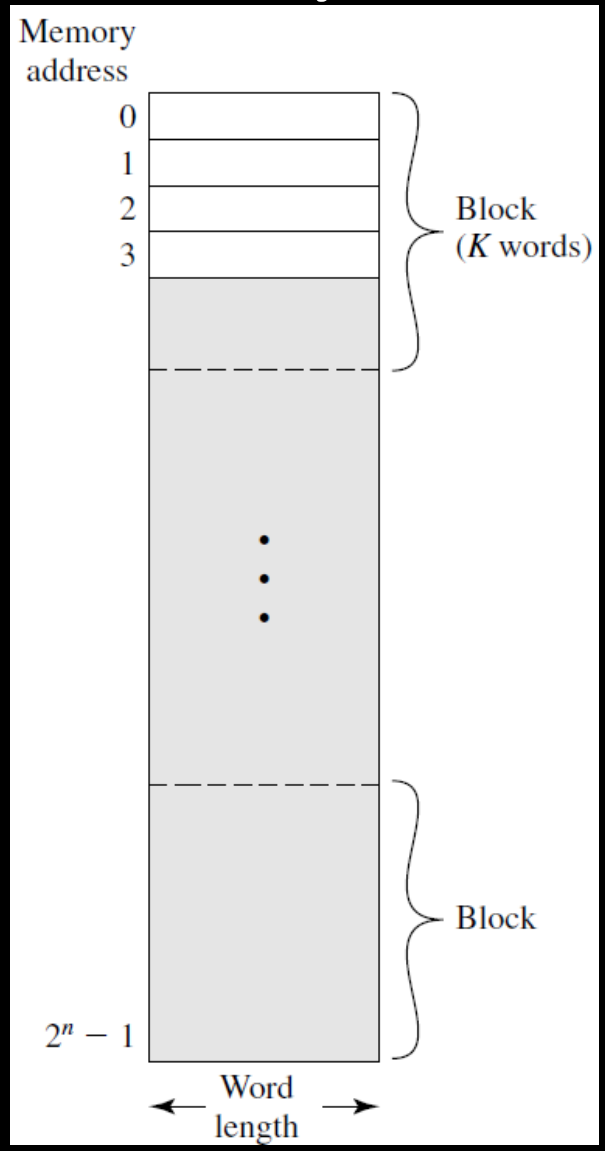
Cache Memory Principles



Cache Memory Principles

Main memory

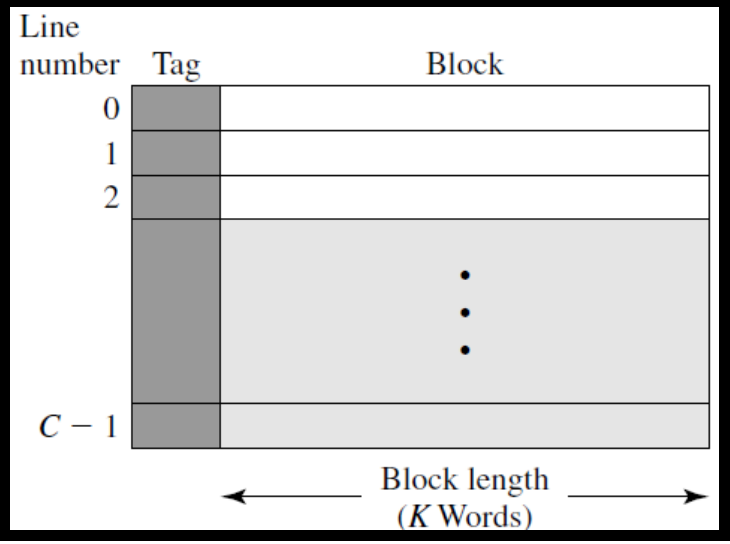
- Consists of 2^n words.
  - Each word has n bits.
- Divided into blocks.
  - Each block has K words.
  - There is $M = 2^n/K$ blocks

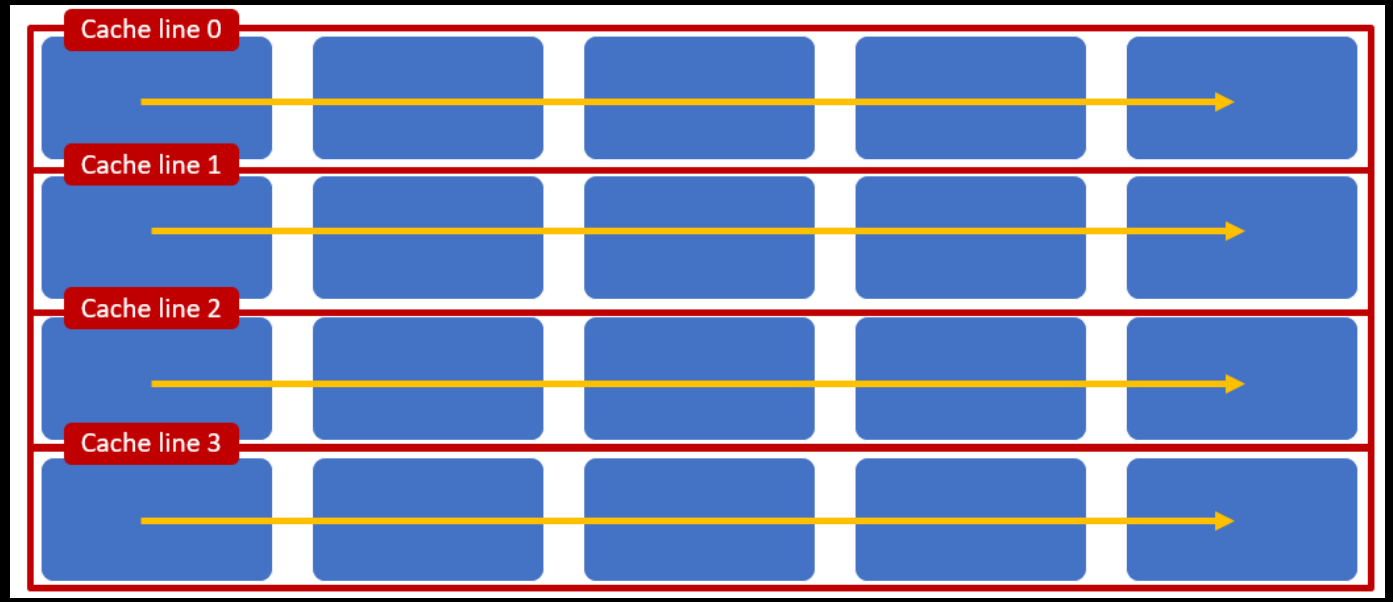


Cache Memory Principles

Cache

- Consists of m lines.
- Each line contains
  - K words,
  - a tag of a few bits,
  - control bits, tells if the line has been modified or not.



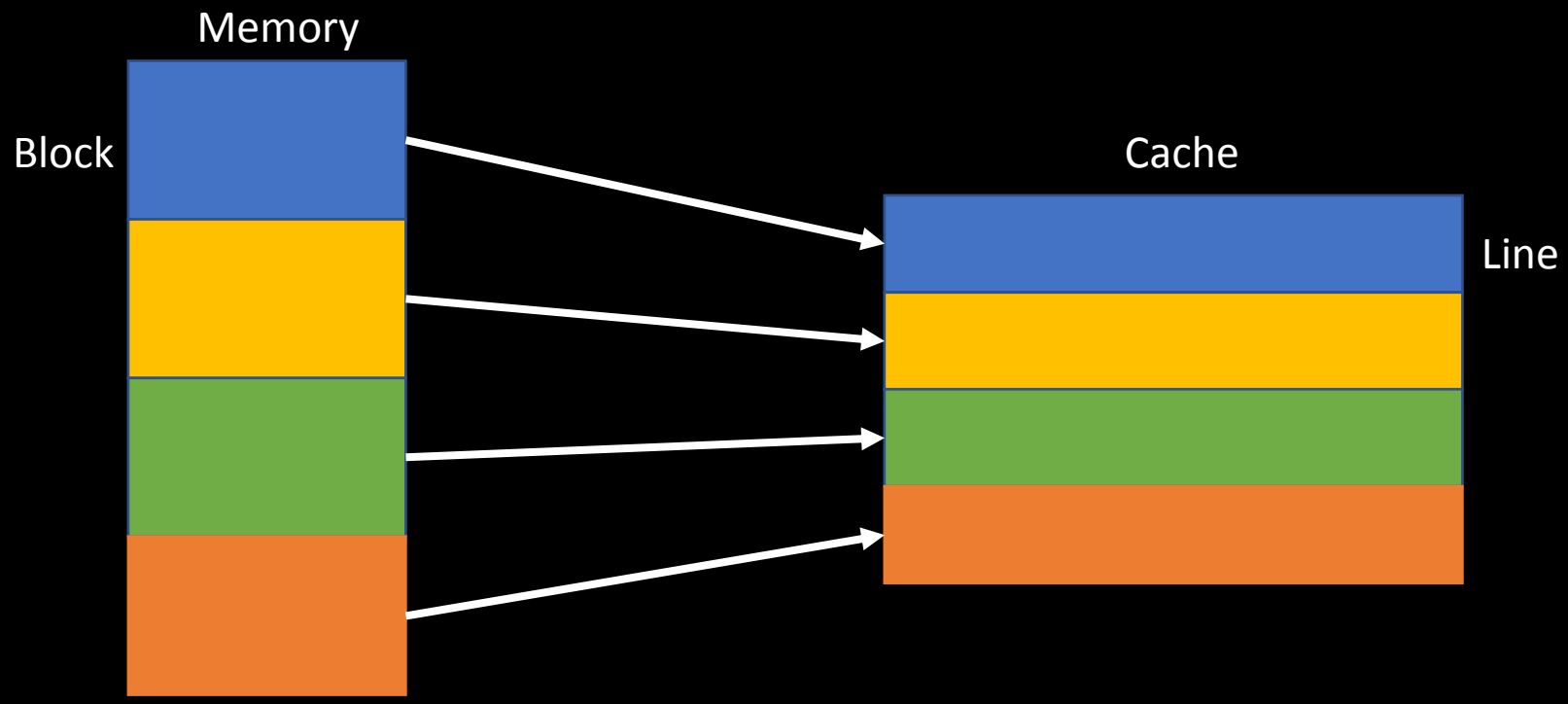


Cache Memory Principles

- The number of lines is less than the number of main memory blocks: $m < M$.

Cache Memory Principles

- The number of lines is less than the number of main memory blocks: m < M.
- When reading a block of memory, the block is copied into a line of the cache.
  - Each line includes a **tag** that identifies which particular block is currently being stored.



Cache Memory Principles

- Cache read operation.

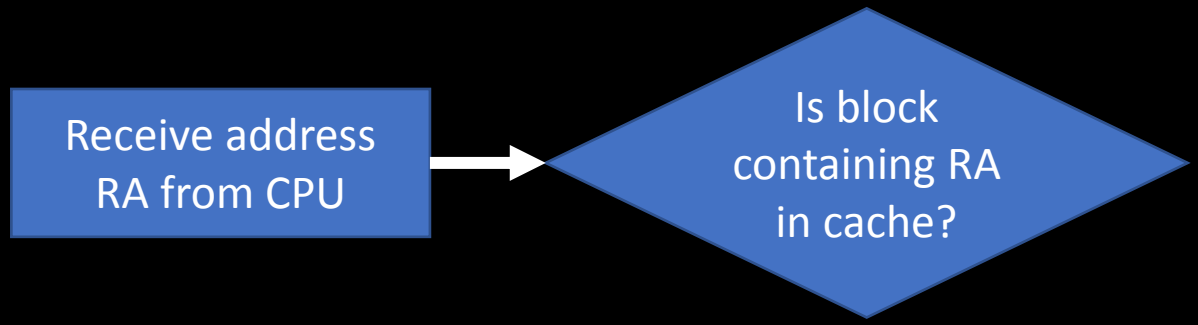
Cache Memory Principles

- Cache read operation.
  RA is read address of a word to be read

Receive address
RA from CPU

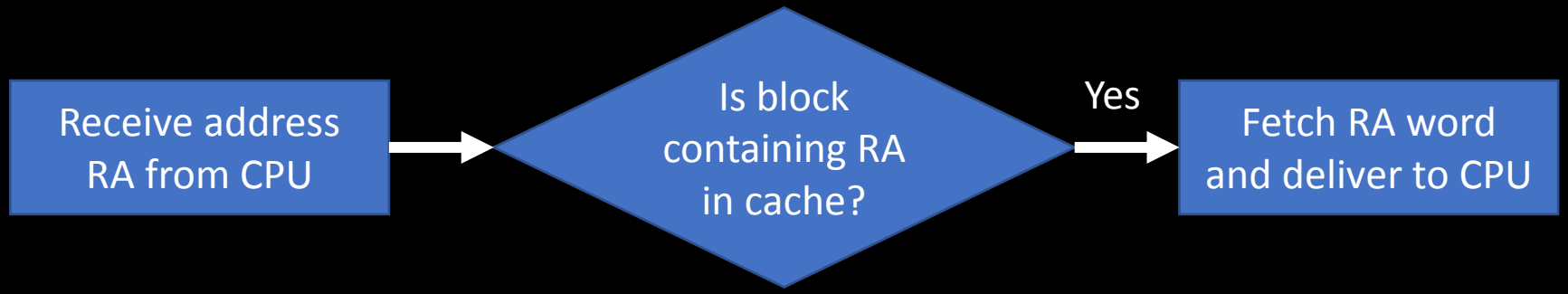
Cache Memory Principles

- Cache read operation.



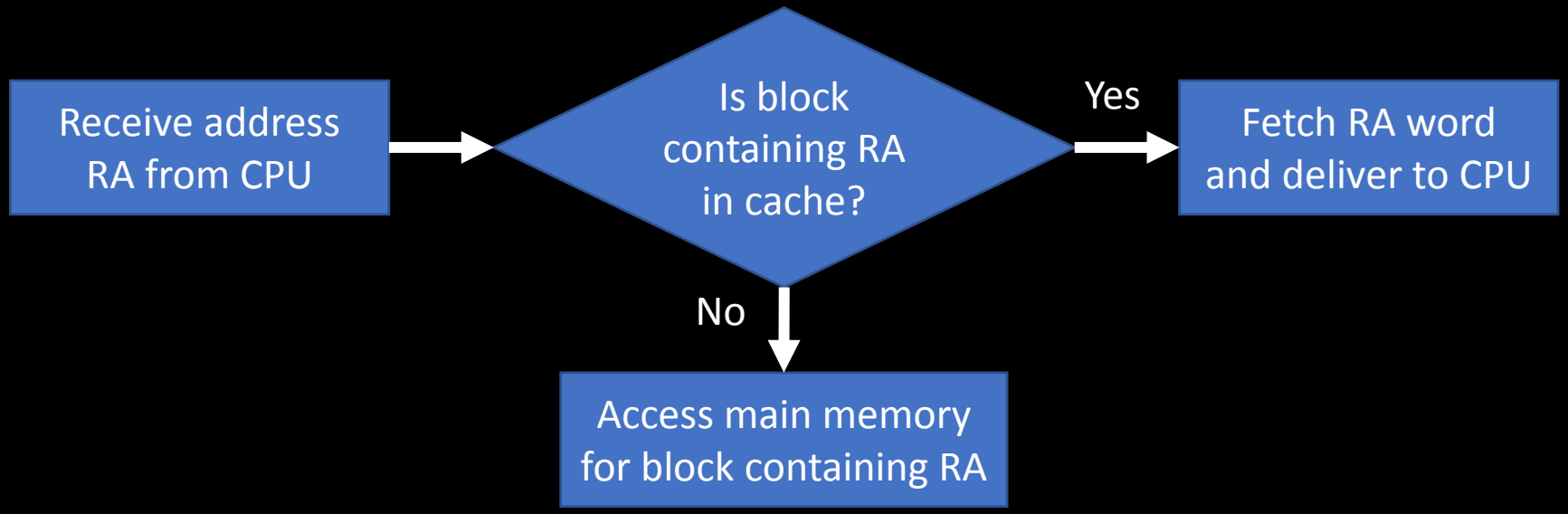
Cache Memory Principles

- Cache read operation.



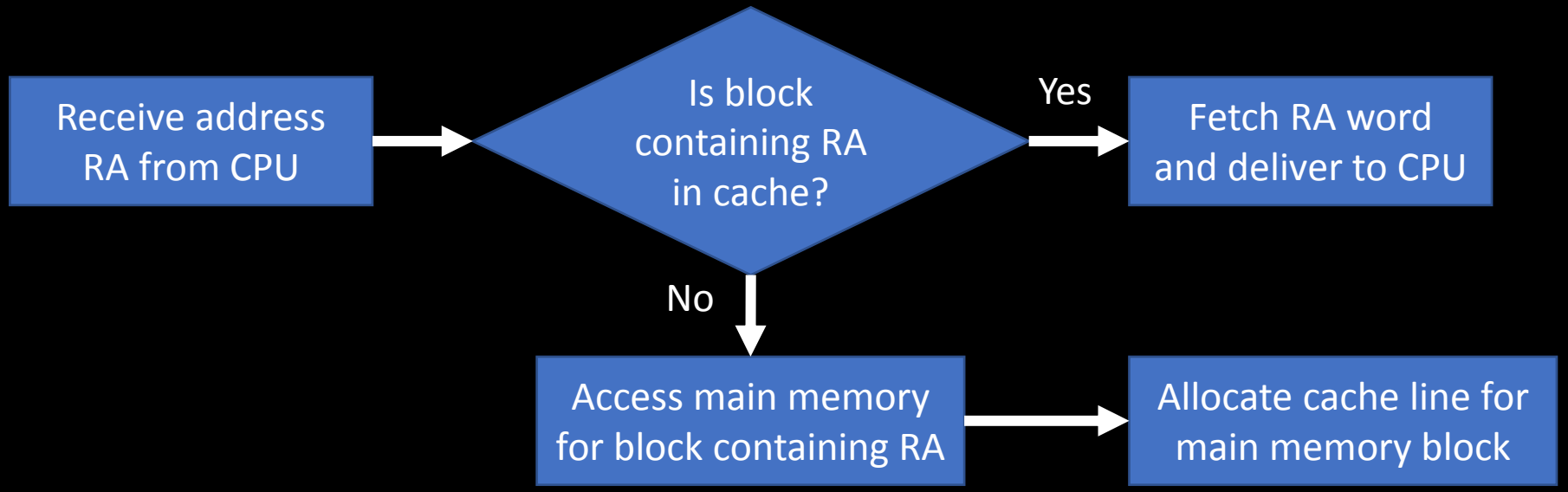
Cache Memory Principles

- Cache read operation.



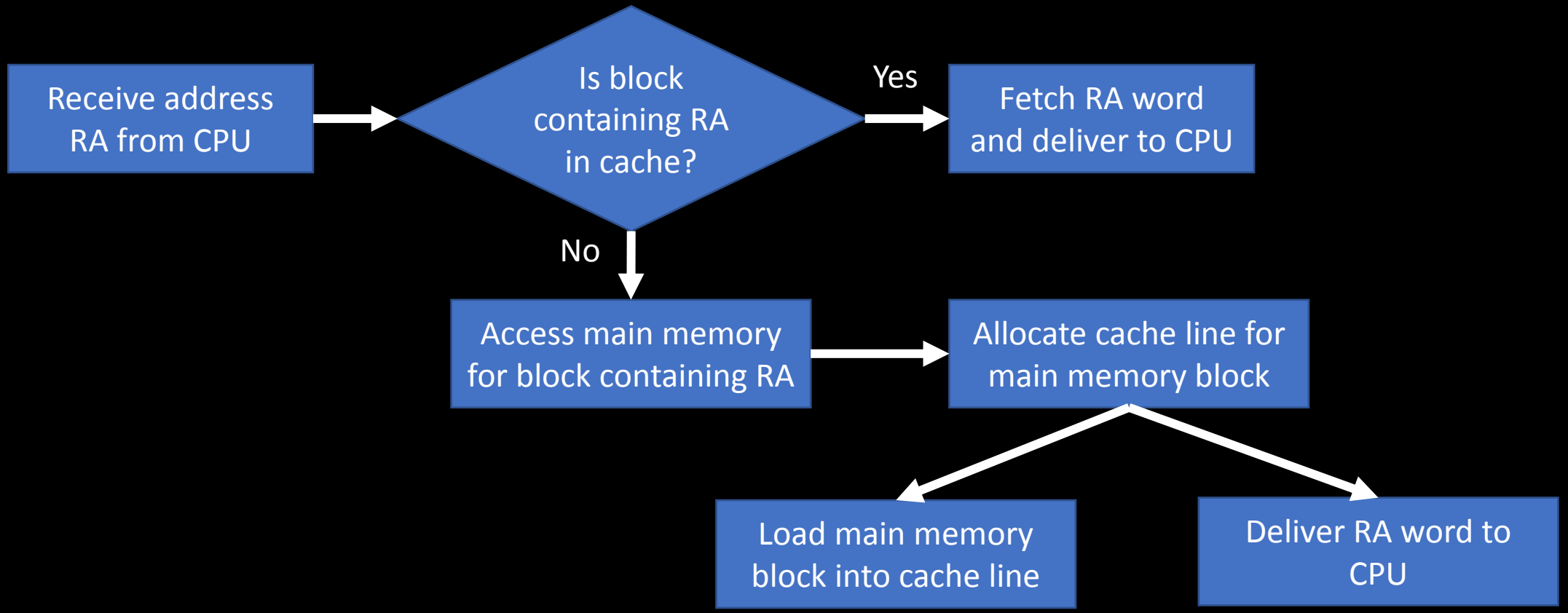
Cache Memory Principles

- Cache read operation.



Cache Memory Principles

- Cache read operation.



Content

CH 04
Computer Memory System Overview
Cache Memory Principles
Elements of Cache Design
~~Pentium 4 Cache Organization~~
~~ARM Cache Organization~~

Elements of Cache Design

- Mapping Function: tells how the cache is organized and how a memory block is mapped into the cache.

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

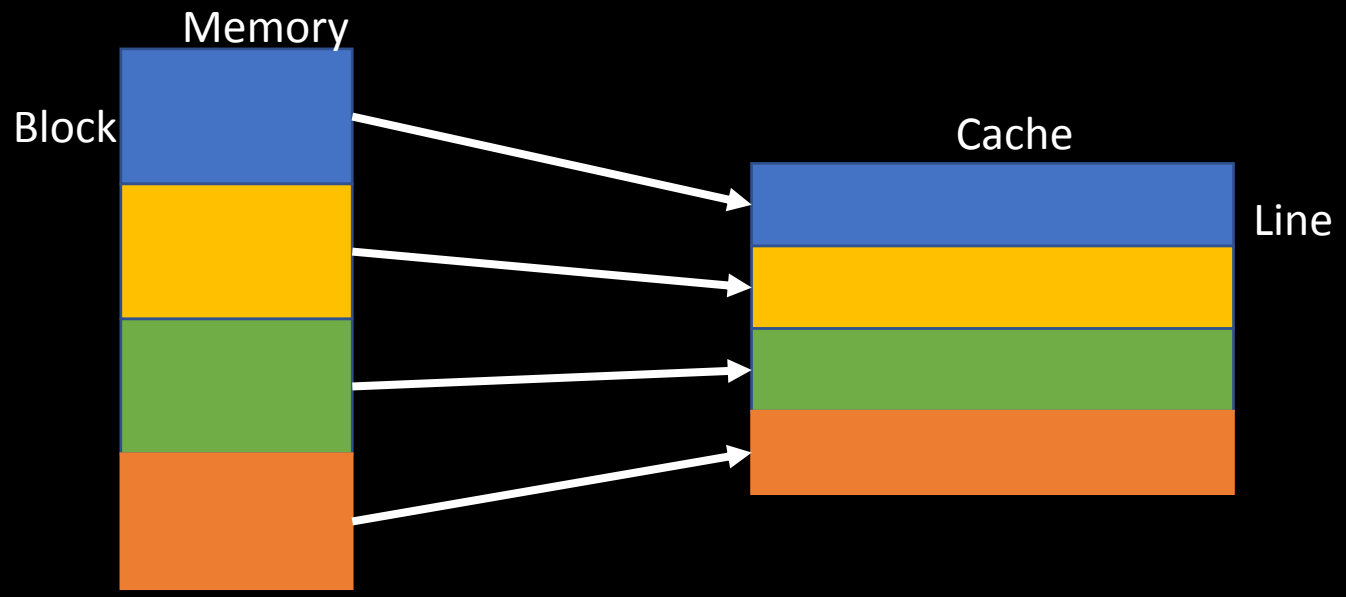
Elements of Cache Design – Direct Mapping

- Maps each block of main memory into only one possible cache line.
- The mapping is expressed as

$$i = j \; mod \; m$$

where

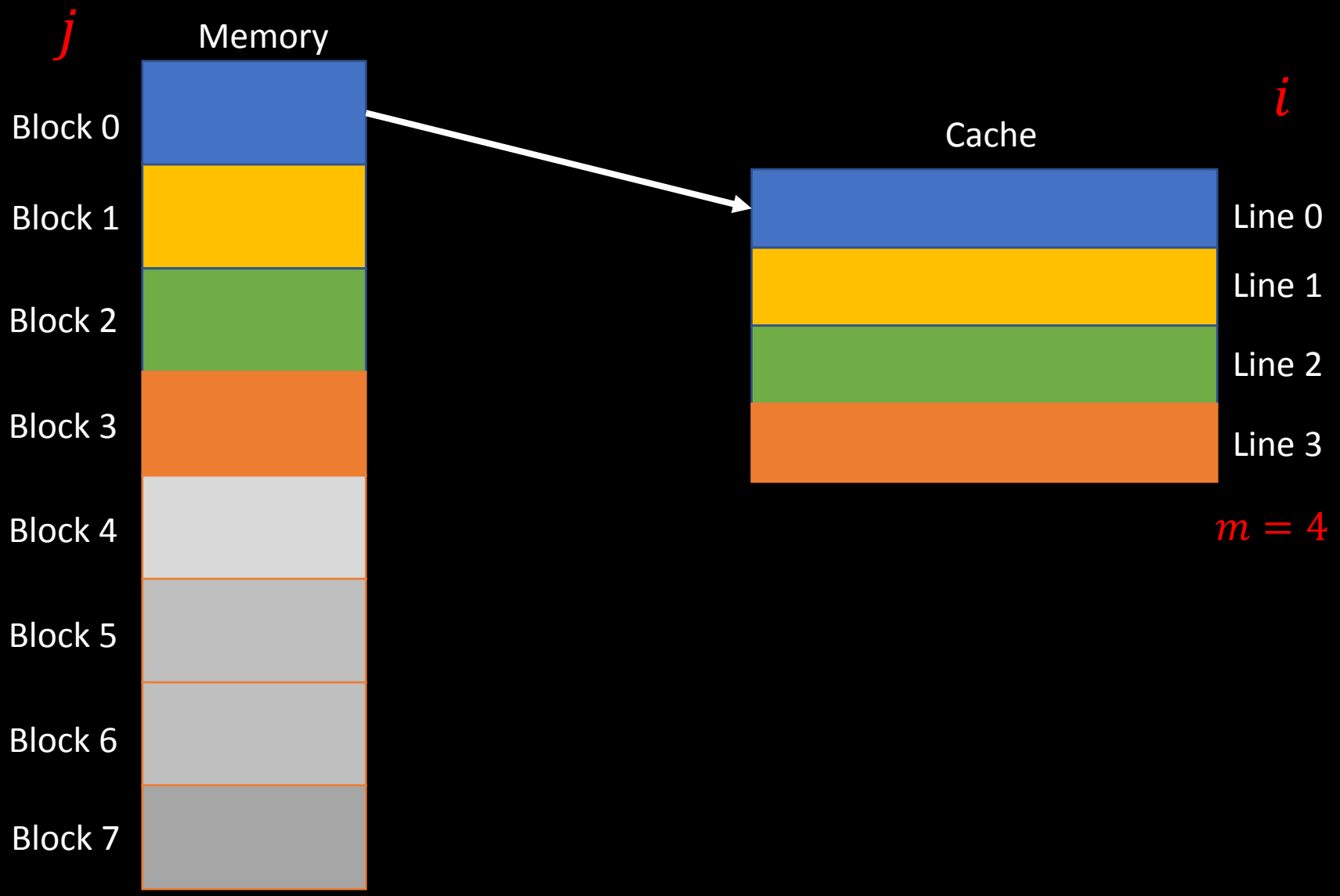
- i cache line number
- j main memory block number
- m number of lines in the cache



Elements of Cache Design – Direct Mapping

$$i = j \; mod \; m$$

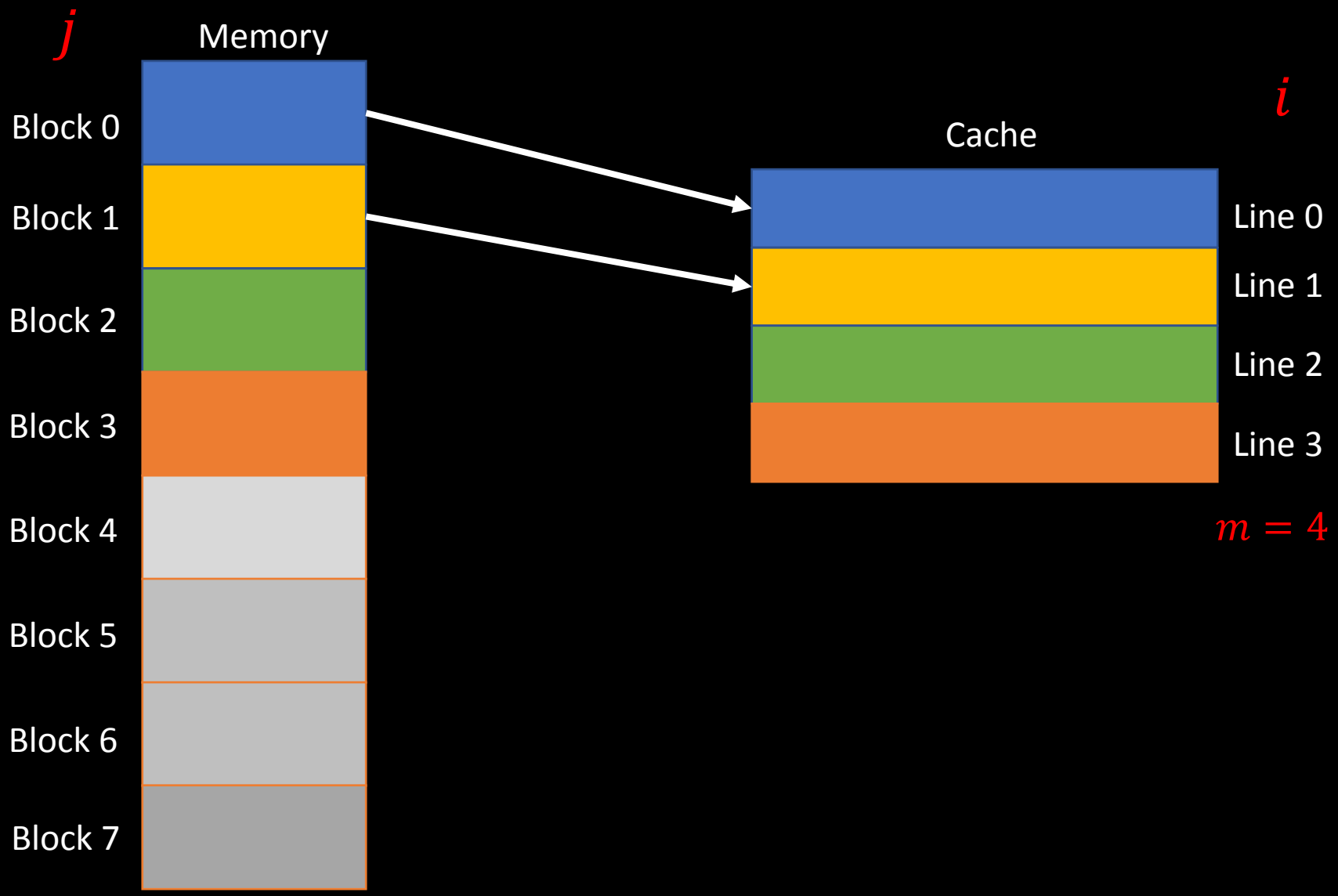
$$0 \; \% \; 4 = 0$$



Elements of Cache Design – Direct Mapping

$$i = j \; mod \; m$$

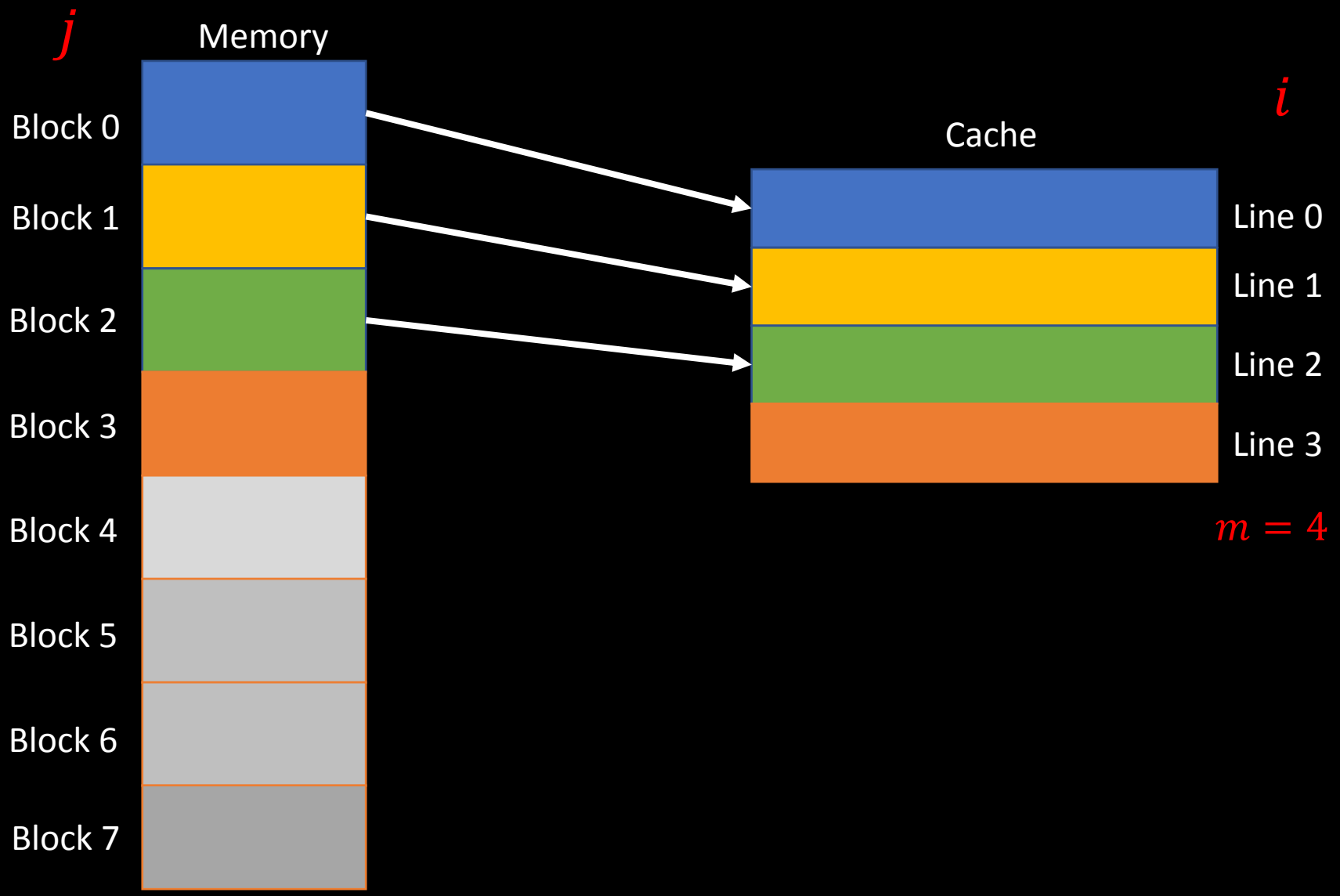
$$1 \; \% \; 4 = 1$$



Elements of Cache Design – Direct Mapping

$$i = j \; mod \; m$$

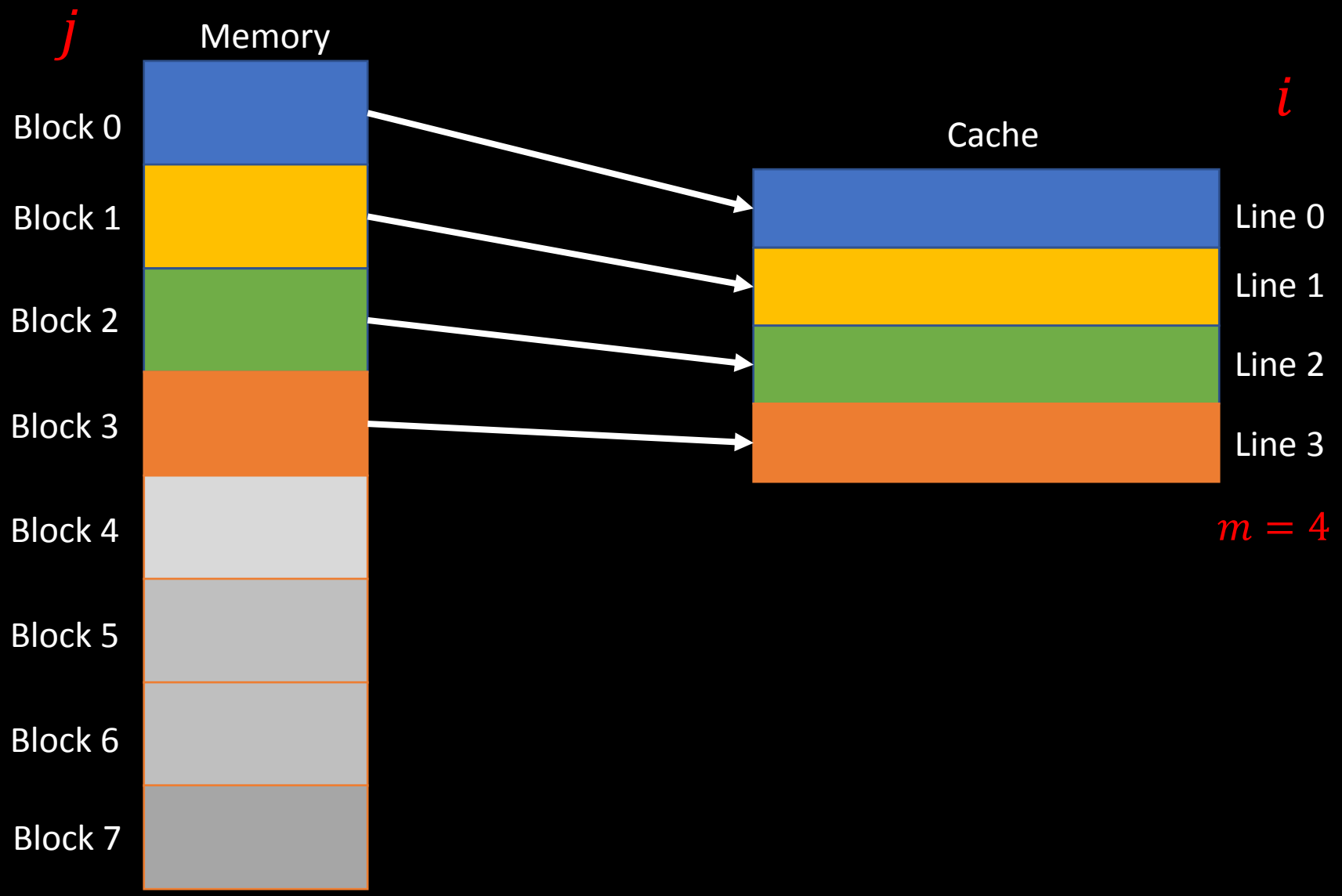
$$2 \; \% \; 4 = 2$$



Elements of Cache Design – Direct Mapping

$i = j \; mod \; m$

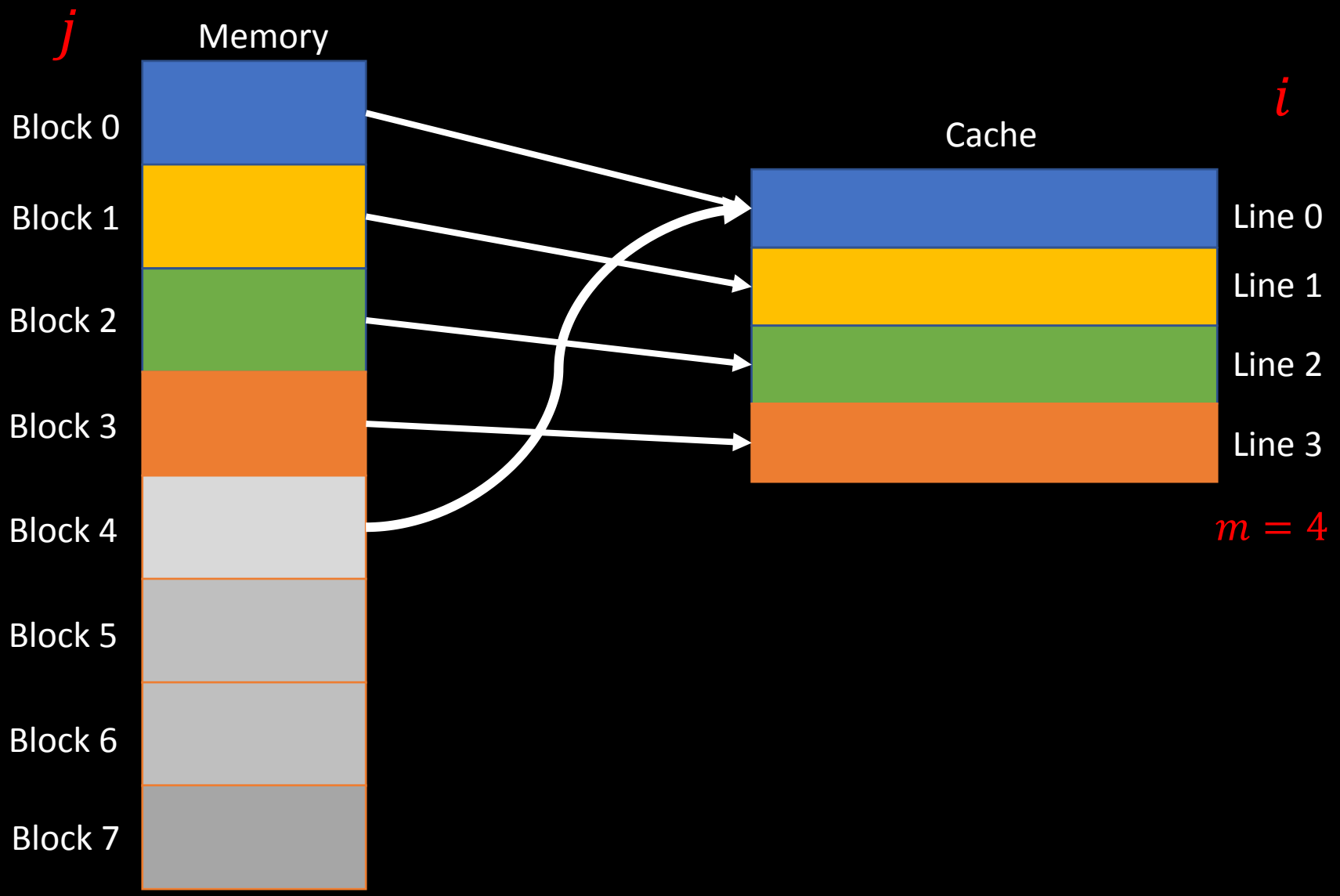
$3 \; \% \; 4 = 3$



Elements of Cache Design – Direct Mapping

$$i = j \; mod \; m$$

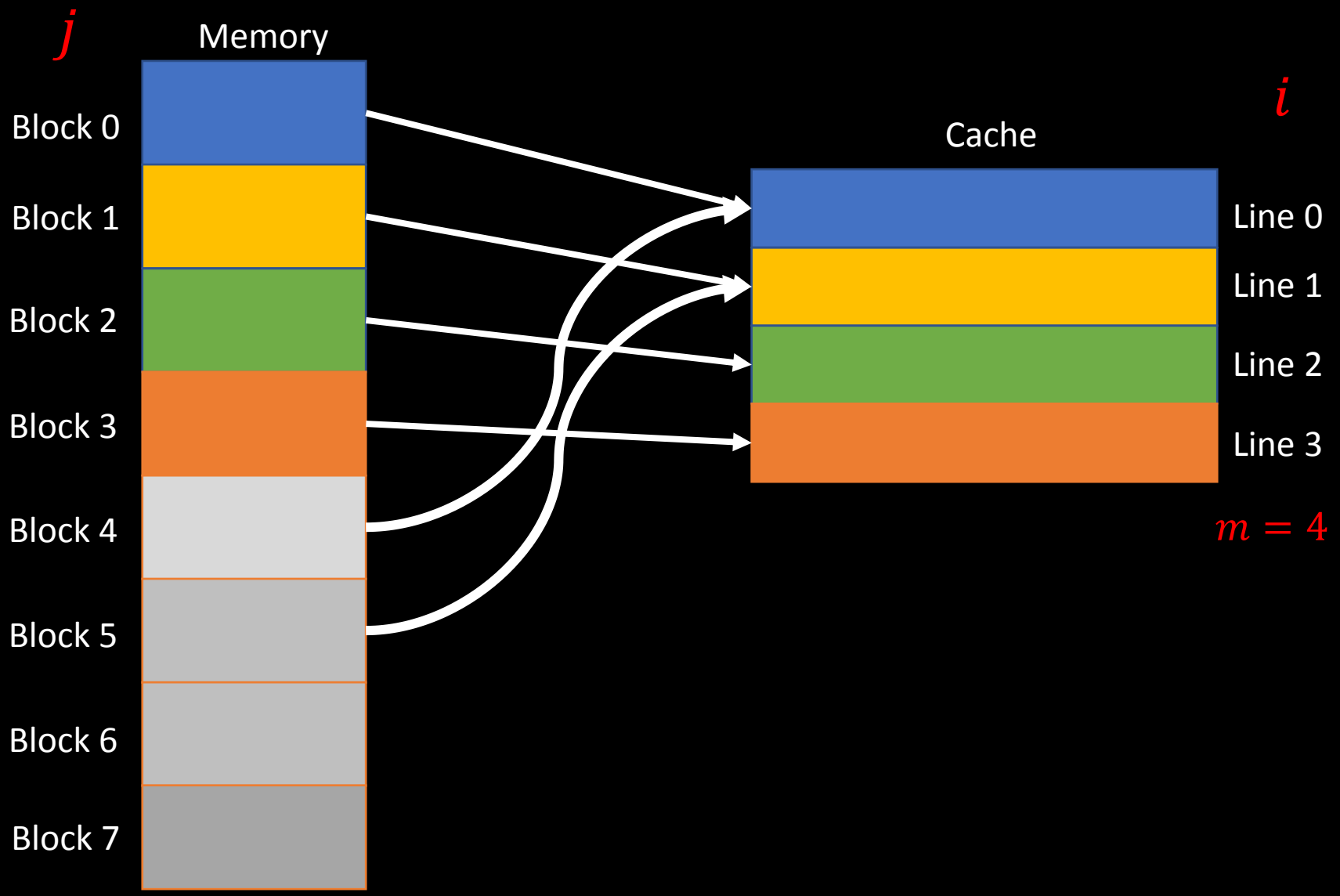
$$4 \; \% \; 4 = 0$$



Elements of Cache Design – Direct Mapping

$$i = j \; mod \; m$$

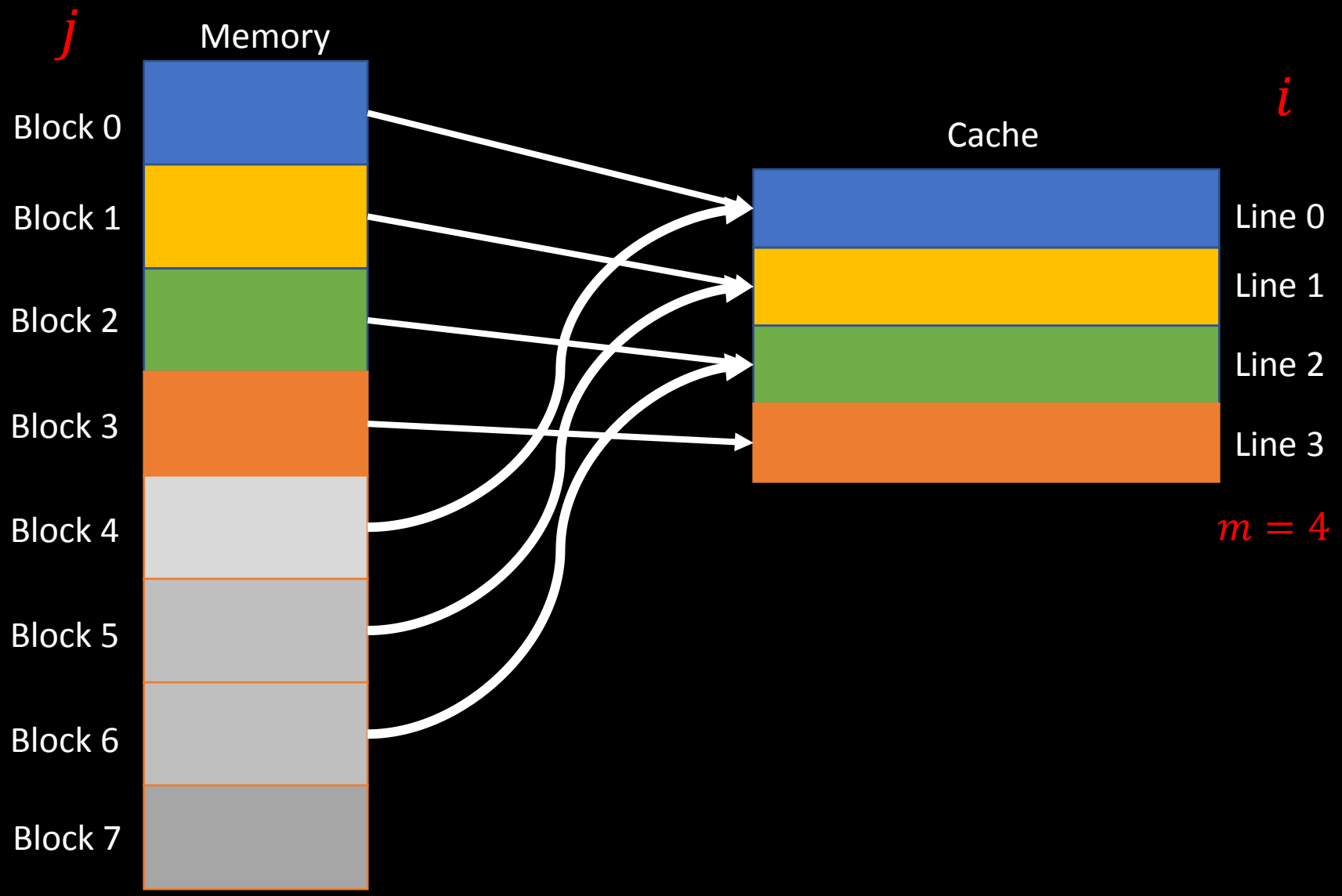
$$5 \; \% \; 4 = 1$$



Elements of Cache Design – Direct Mapping

$i = j \; mod \; m$

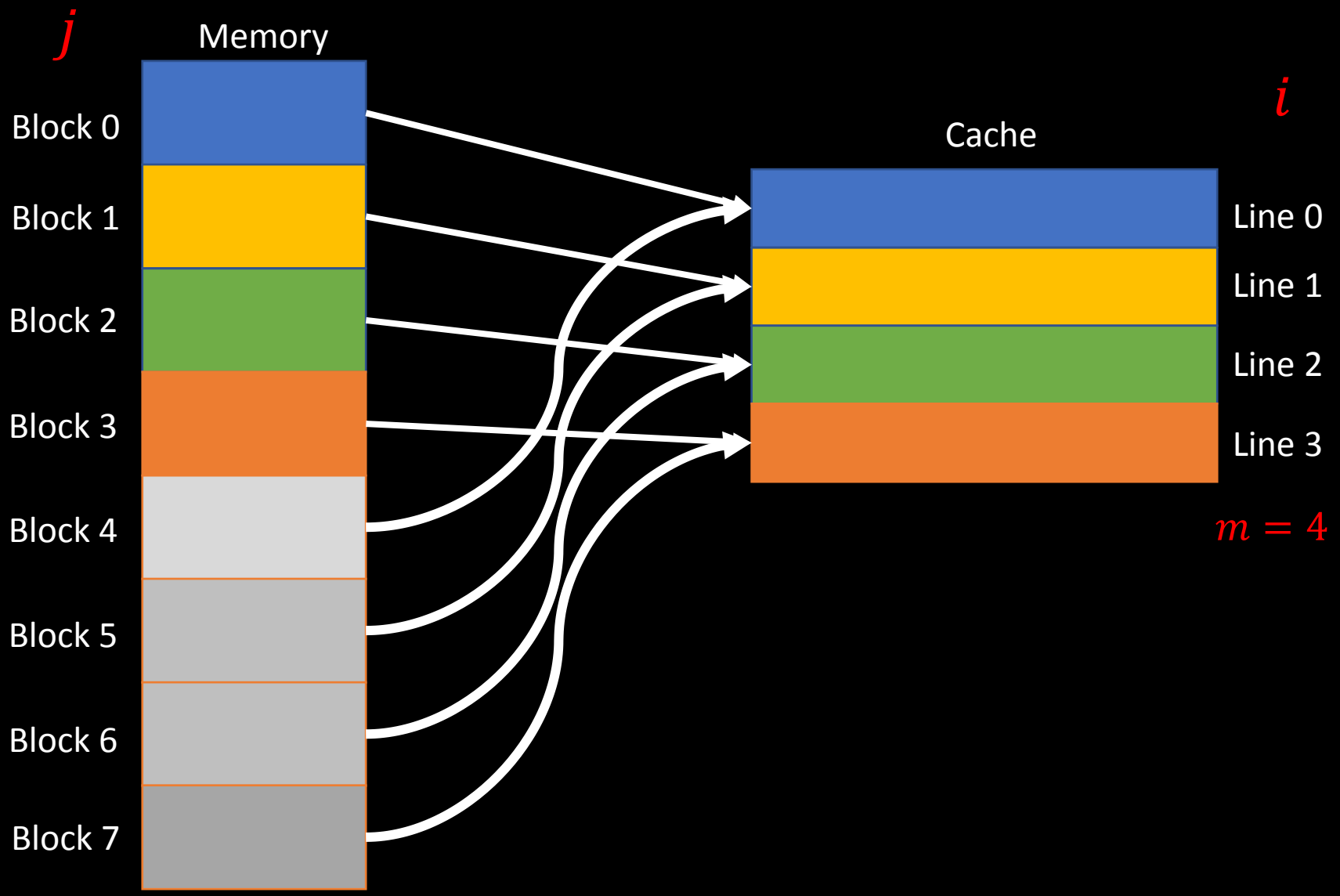
$6 \; \% \; 4 = 2$



Elements of Cache Design – Direct Mapping

$$i = j \; mod \; m$$

$$7 \; \% \; 4 = 3$$



Elements of Cache Design – Direct Mapping

- So, if we have a memory consists of 2^s blocks, the mapping of the memory blocks to cache lines is as follows:
  - Every cache line can store many blocks, but not at the same time.

Cache line	Main memory blocks assigned
0	$0, m, 2m, \ldots, 2^s - m$
1	$1, m + 1, 2m + 1, \ldots, 2^s - m + 1$
⋮	⋮
$m - 1$	$m - 1, 2m - 1, 3m - 1, \ldots, 2^s - 1$

Elements of Cache Design – Direct Mapping

The organization of direct cache mapping is as follows:

- We have a memory with address length = $(s + w)$ bits.

Elements of Cache Design – Direct Mapping

The organization of direct cache mapping is as follows:

- We have a memory with address length = $(s + w)$ bits.
  - The number of addressable units is 2^{s+w} words or bytes.

Elements of Cache Design – Direct Mapping

The organization of direct cache mapping is as follows:

- We have a memory with address length = $(s + w)$ bits.
  - The number of addressable units is 2^{s+w} words or bytes.
- Block size = line size = 2^{w} words or bytes.

Elements of Cache Design – Direct Mapping

The organization of direct cache mapping is as follows:

- We have a memory with address length = $(s + w)$ bits.
  - The number of addressable units is 2^{s+w} words or bytes.
- Block size = line size = 2^w words or bytes.
  - Number of blocks in main memory is $\frac{2^{s+w}}{2^w} = 2^s$.

Elements of Cache Design – Direct Mapping

The organization of direct cache mapping is as follows:

- We have a memory with address length = $(s + w)$ bits.
  - The number of addressable units is 2^{s+w} words or bytes.
- Block size = line size = 2^w words or bytes.
  - Number of blocks in main memory is $\frac{2^{s+w}}{2^w} = 2^s$.
- Number of lines in the cache = $m = 2^r$.

Elements of Cache Design – Direct Mapping

The organization of direct cache mapping is as follows:

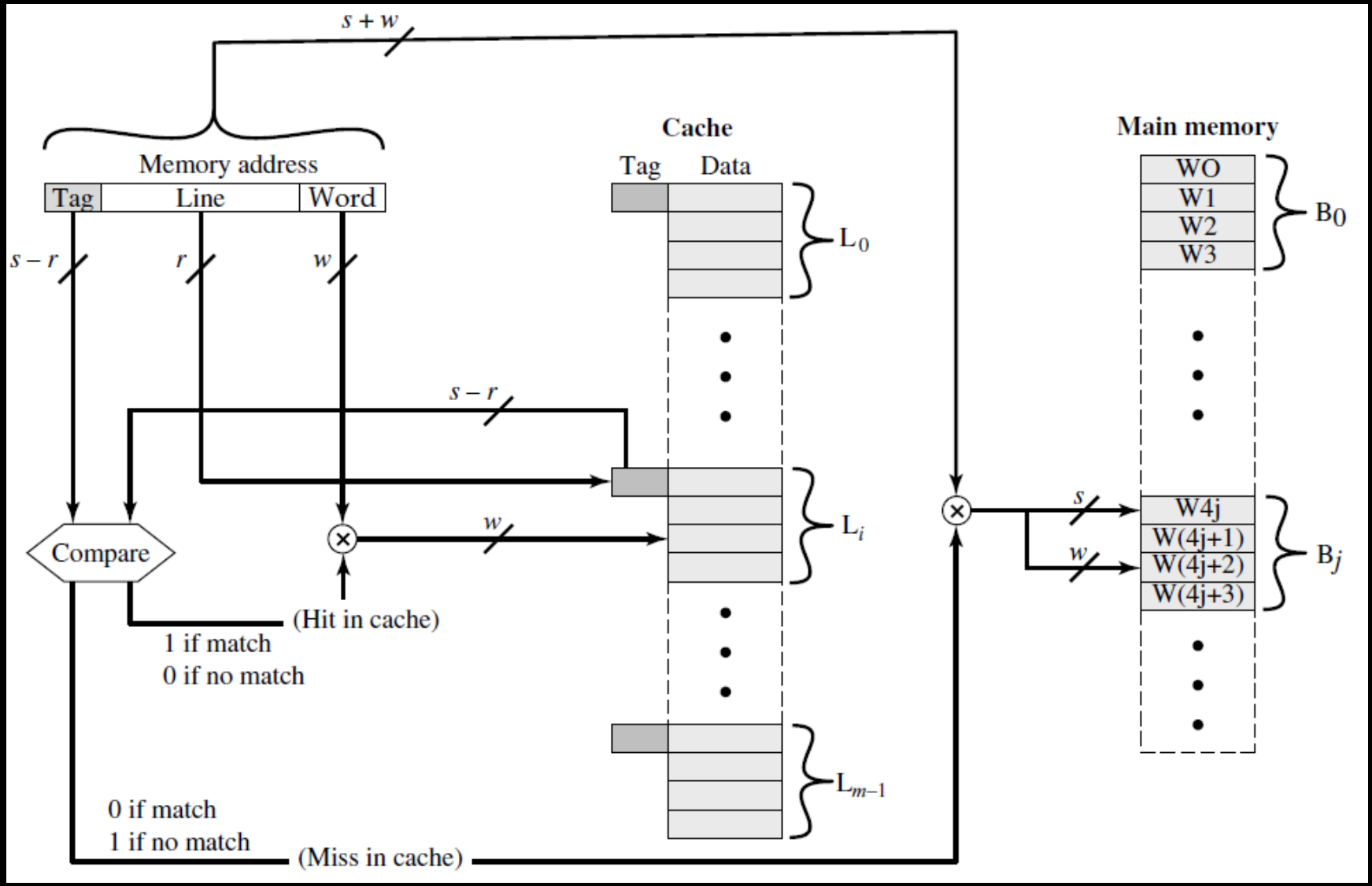
- We have a memory with address length = $(s + w)$ bits.
  - The number of addressable units is 2^{s+w} words or bytes.
- Block size = line size = 2^w words or bytes.
  - Number of blocks in main memory is $\frac{2^{s+w}}{2^w} = 2^s$.
- Number of lines in the cache = $m = 2^r$.
- Size of the cache = 2^{r+w} words or bytes.

Elements of Cache Design – Direct Mapping

The organization of direct cache mapping is as follows:

- We have a memory with address length = $(s + w)$ bits.
  - The number of addressable units is 2^{s+w} words or bytes.
- Block size = line size = 2^w words or bytes.
  - Number of blocks in main memory is $\frac{2^{s+w}}{2^w} = 2^s$.
- Number of lines in the cache = $m = 2^r$.
- Size of the cache = 2^{r+w} words or bytes.
- Size of the tag = $s - r$ bits.

Elements of Cache Design – Direct Mapping

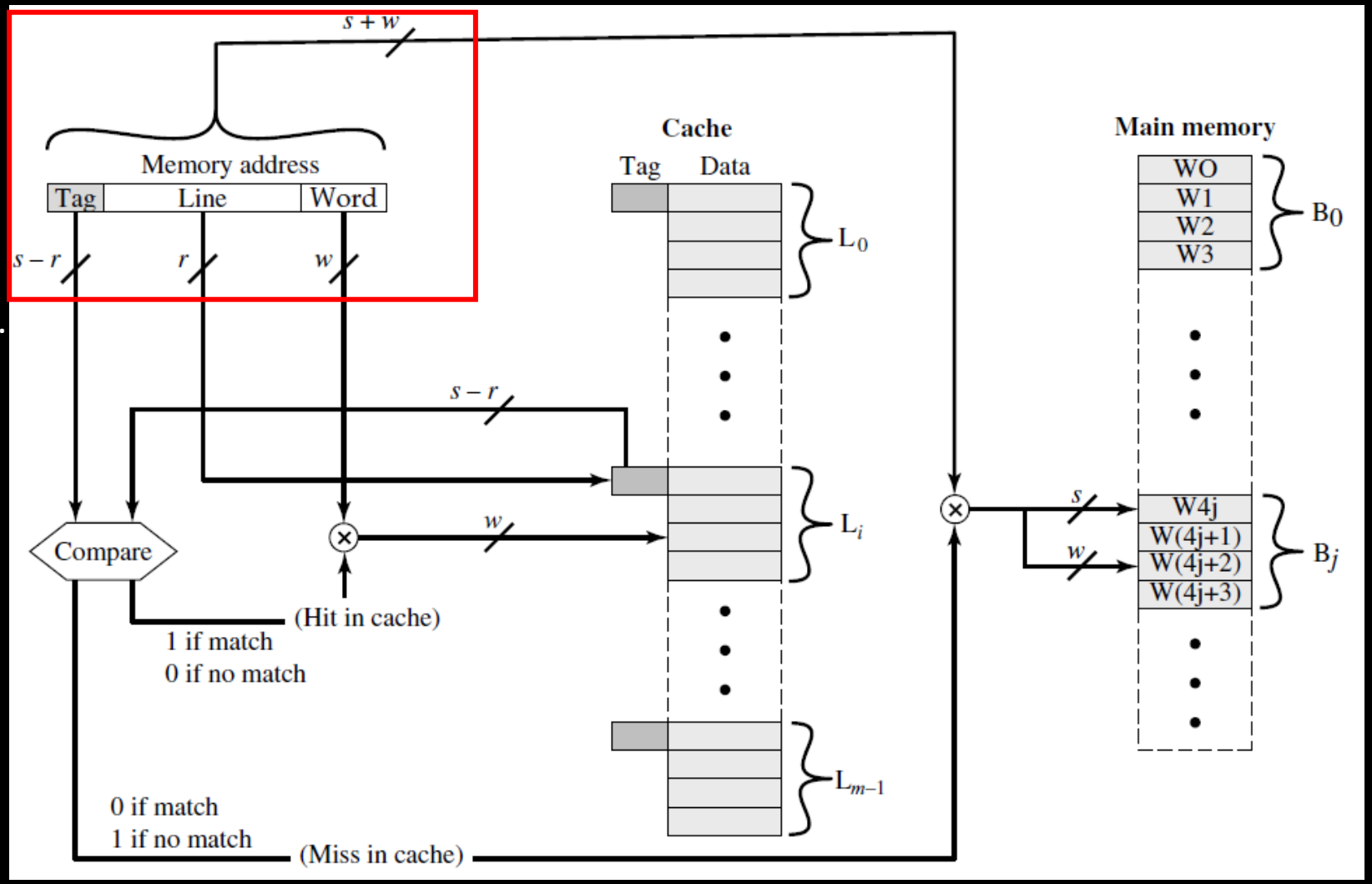


Elements of Cache Design – Direct Mapping

The memory address is divided into three parts:

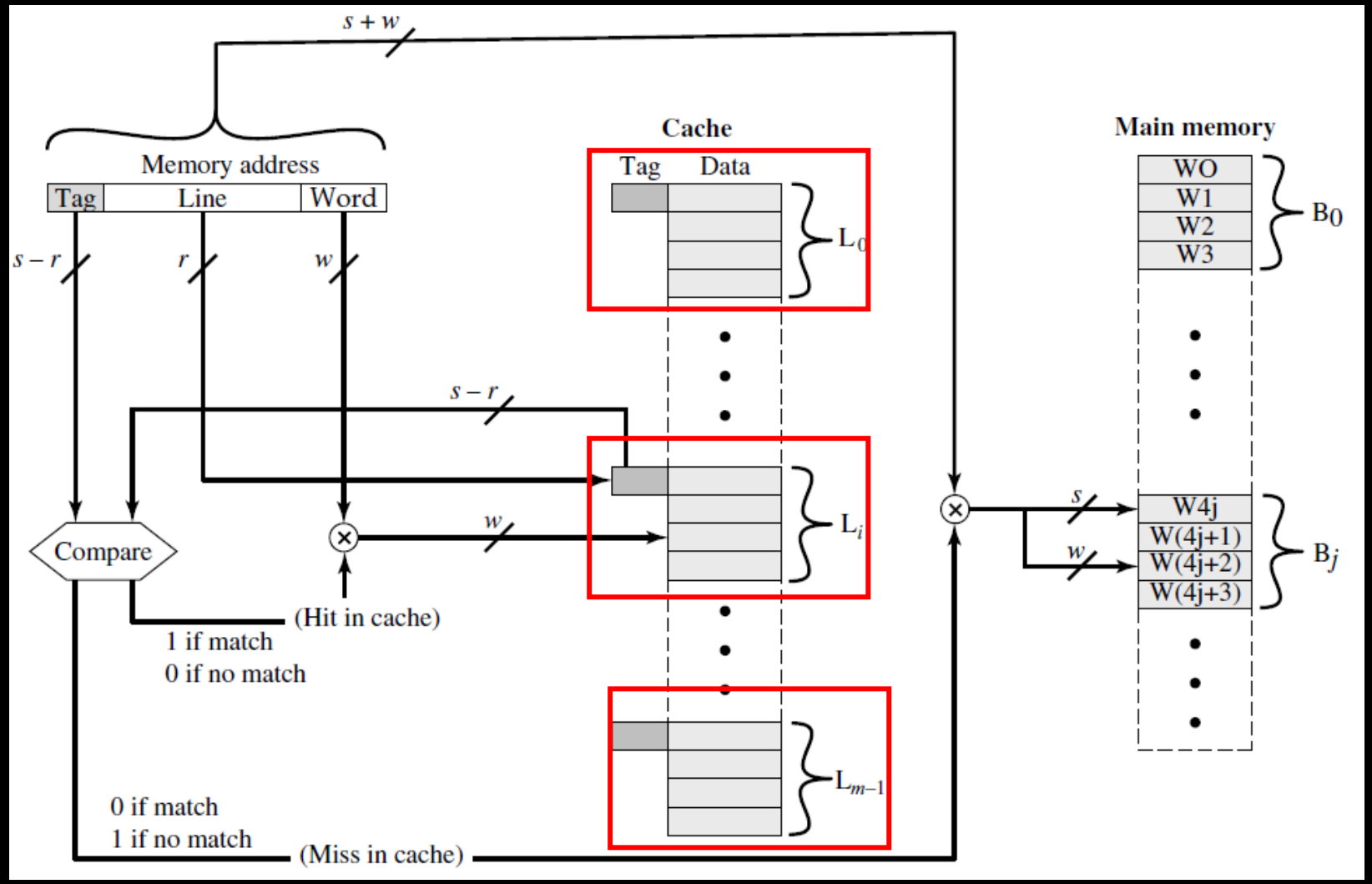
1. **Tag**, identifies which block is currently being stored. Length = s – r bits
2. **Line**, identifies one of the $m = 2^r$ lines of the cache. Length = r bits.
3. **Word**, identifies a unique word or byte within a block of main memory. Length = w bits.

The size of the whole address is $s + w$ bits.



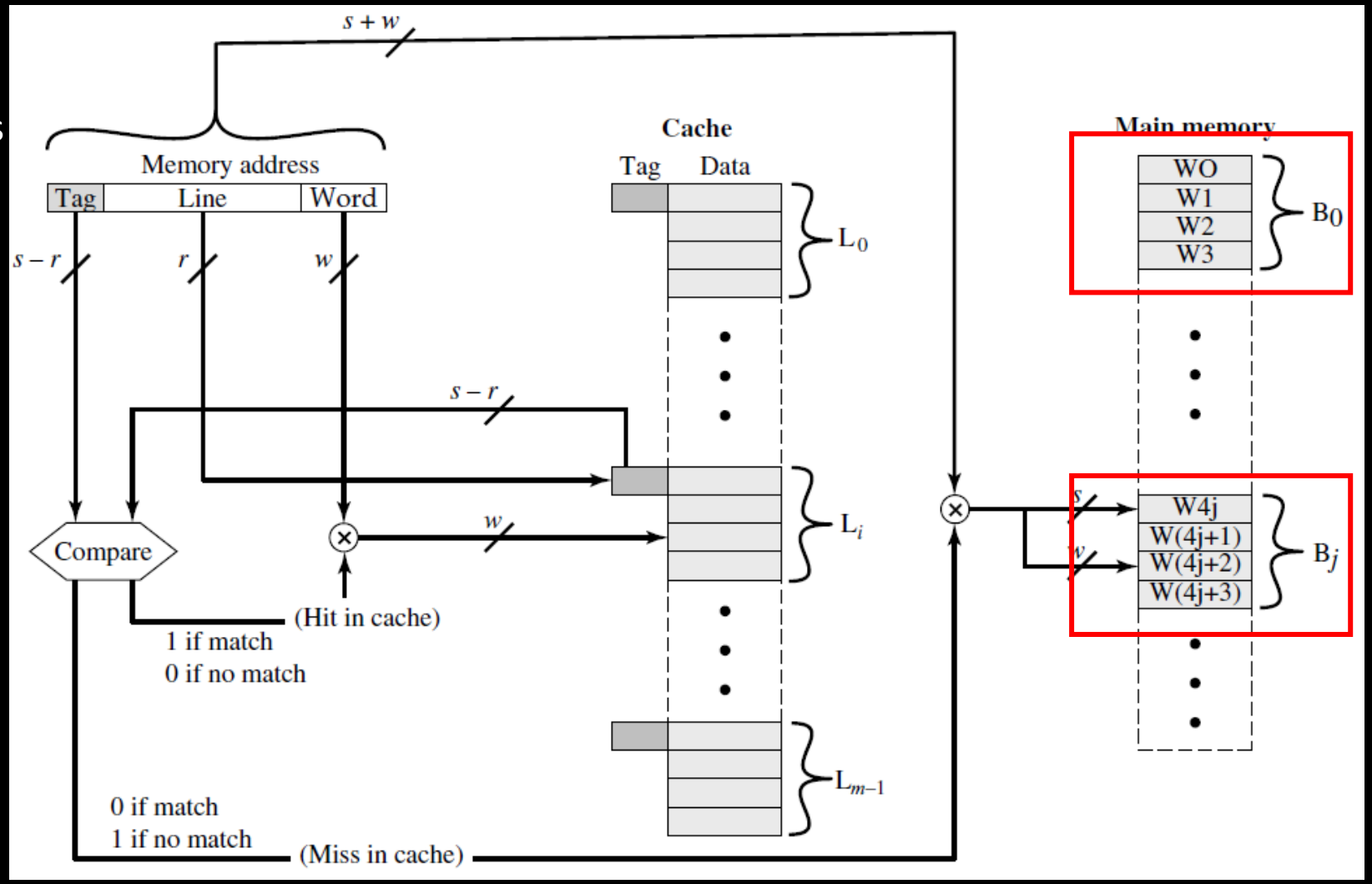
Elements of Cache Design – Direct Mapping

The cache is divided into lines.
Each line is identified by a tag.
Each line has the same number of words or bytes as the block of the memory.



Elements of Cache Design – Direct Mapping

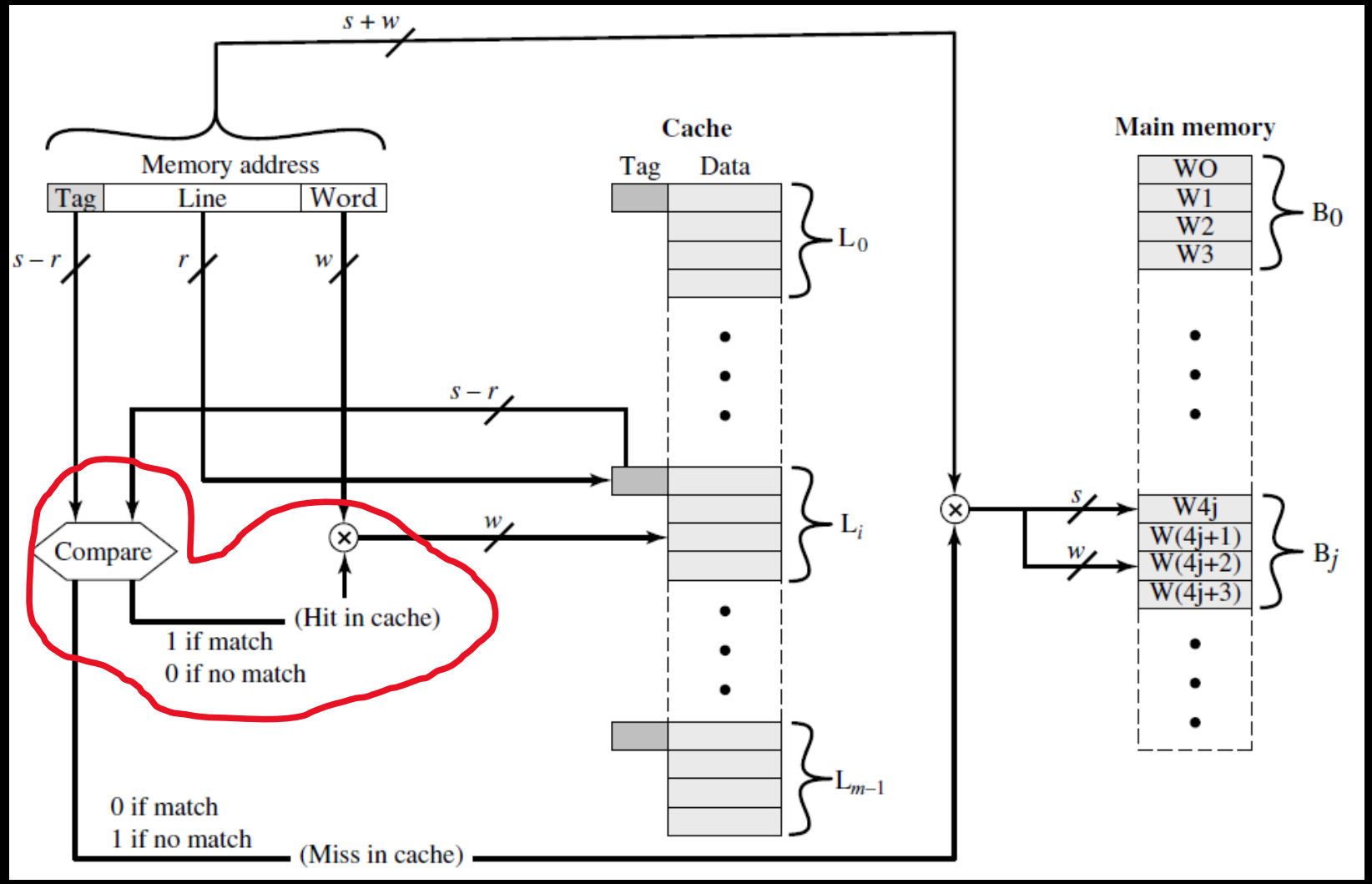
The memory is divided into blocks.
Each block holds the same number of words or bytes as the cache line.



Elements of Cache Design – Direct Mapping

The tag of the memory address is compared with tag of the cache.

If both tags are matched, we fetch the word specified by w at line r.
(This cache Hit)

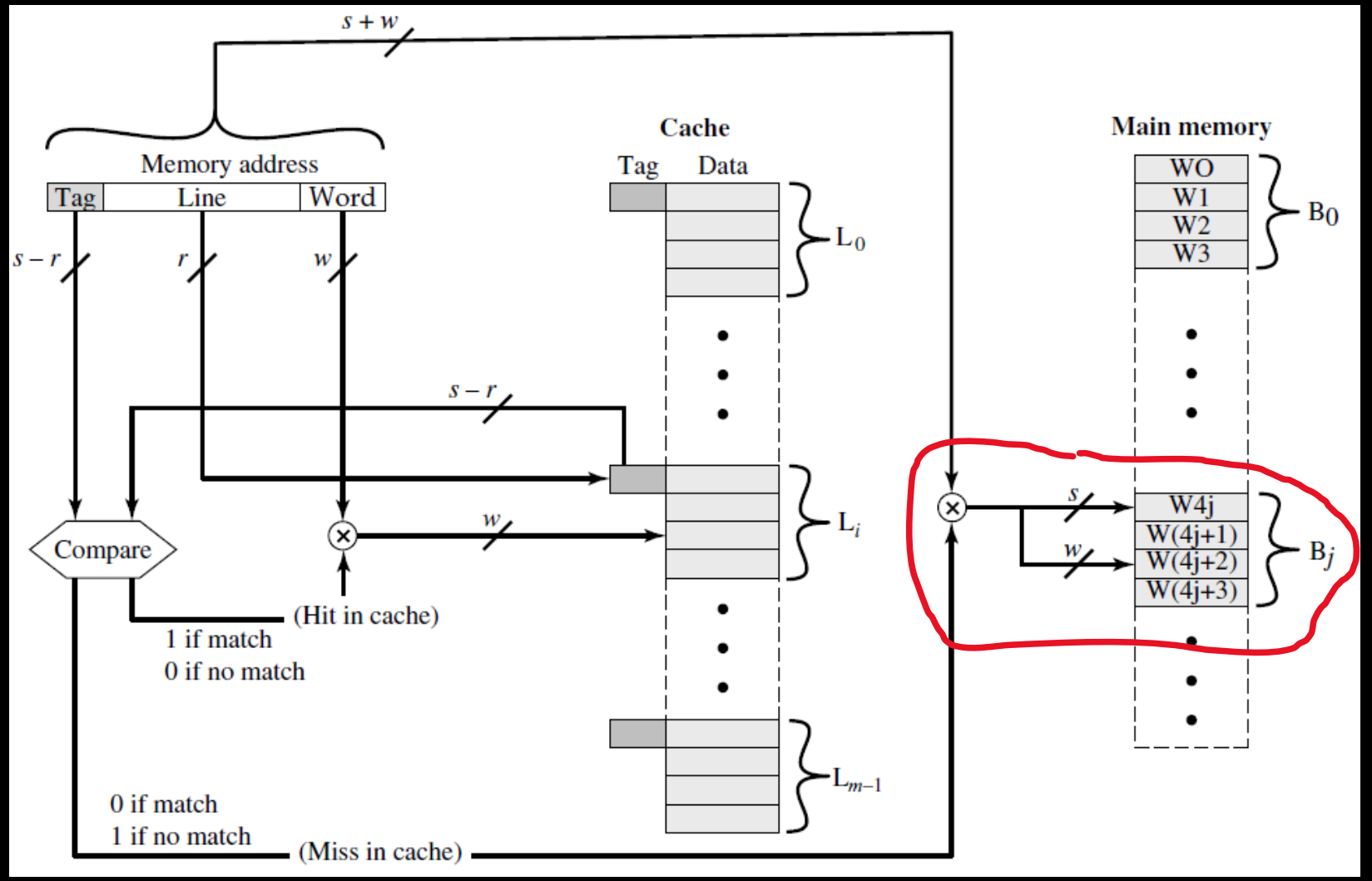


Elements of Cache Design – Direct Mapping

The tag of the memory address is compared with tag of the cache.

If both tags are matched, we fetch the word specified by w at line r.
(This cache Hit)

If the tags are not matched, we fetch the block specified by the $tag + line$ from memory, and fetch word specified by w.
(This cache Miss)



Elements of Cache Design – Direct Mapping

The examples supposes the following organization:

- The cache can hold 64 KBytes.
- Data are transferred between main memory and the cache in blocks of 4 bytes each.
- This means that the cache is organized as $16K = 2^{14}$ lines of 4 bytes each.
- The main memory consists of 16 Mbytes, with each byte directly addressable by a 24-bit address ($2^{24} = 16M$). Thus, for mapping purposes, we can consider main memory to consist of 4M blocks of 4 bytes each.

Elements of Cache Design – Direct Mapping

Example:

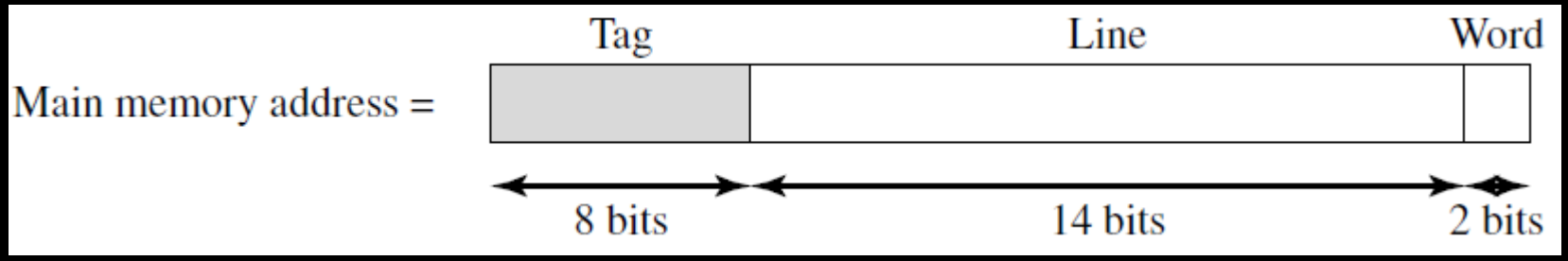
Example 4.2a Figure 4.10 shows our example system using direct mapping.[5] In the example, $m = 16\text{K} = 2^{14}$ and $i = j$ modulo 2^{14}. The mapping becomes

Cache Line	Starting Memory Address of Block
0	000000, 010000, . . . , FF0000
1	000004, 010004, . . . , FF0004
⋮	⋮
$2^{14} - 1$	00FFFC, 01FFFC, . . . , FFFFFC

Elements of Cache Design – Direct Mapping

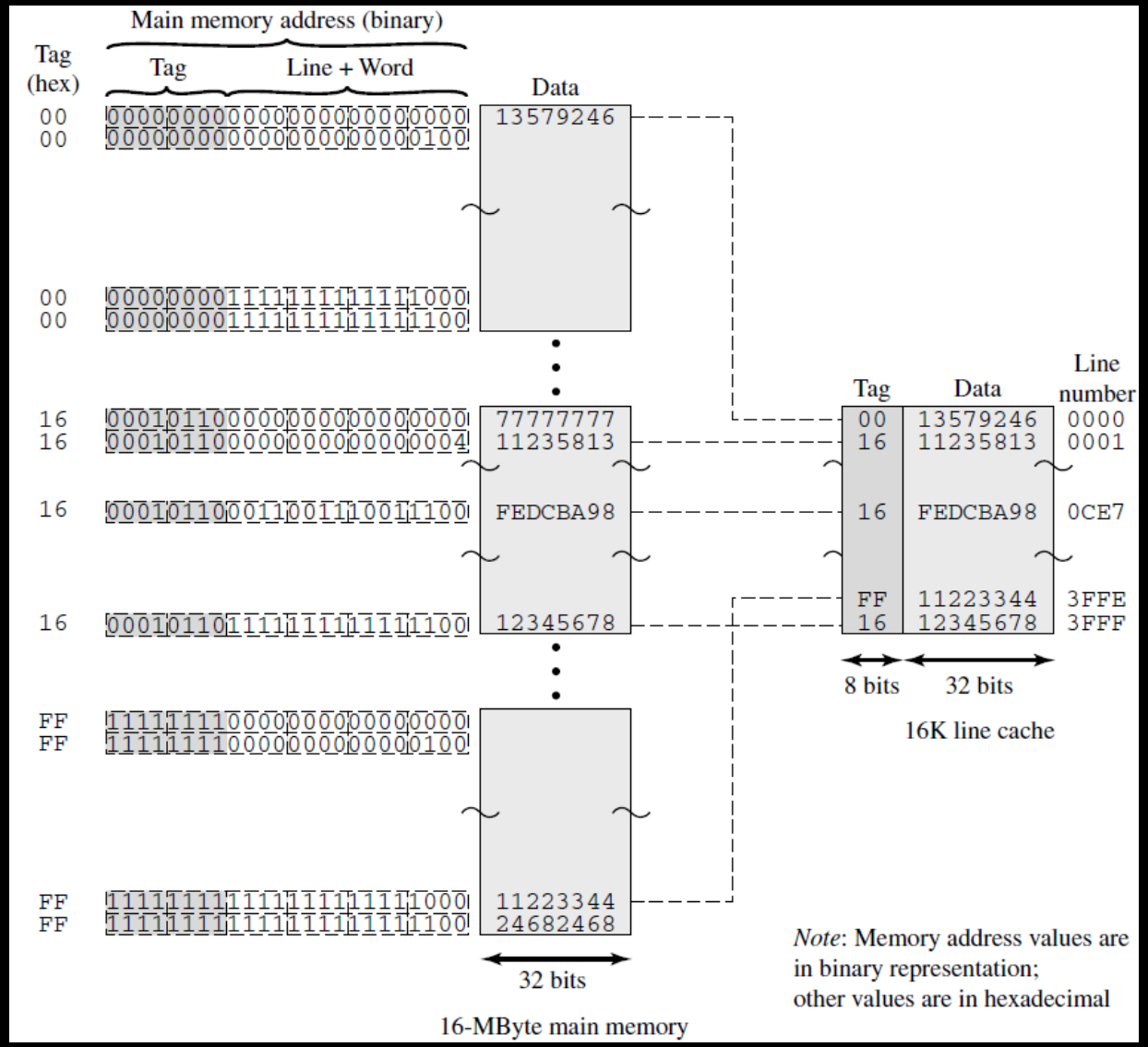
The memory address will be divided as follows:

- We have $m = 16K = 2^{14}$, then $r = 14$ bits.
- The size of each block is $4\ bytes$, thus each byte in each block is addressable by 2 bits. Thus, $w = 2$ bits.
- The address of the memory is 24 bits, then the remaining bits specifies the tag. Thus, the tag is $24 - (14 + 2) = 8$ bits.



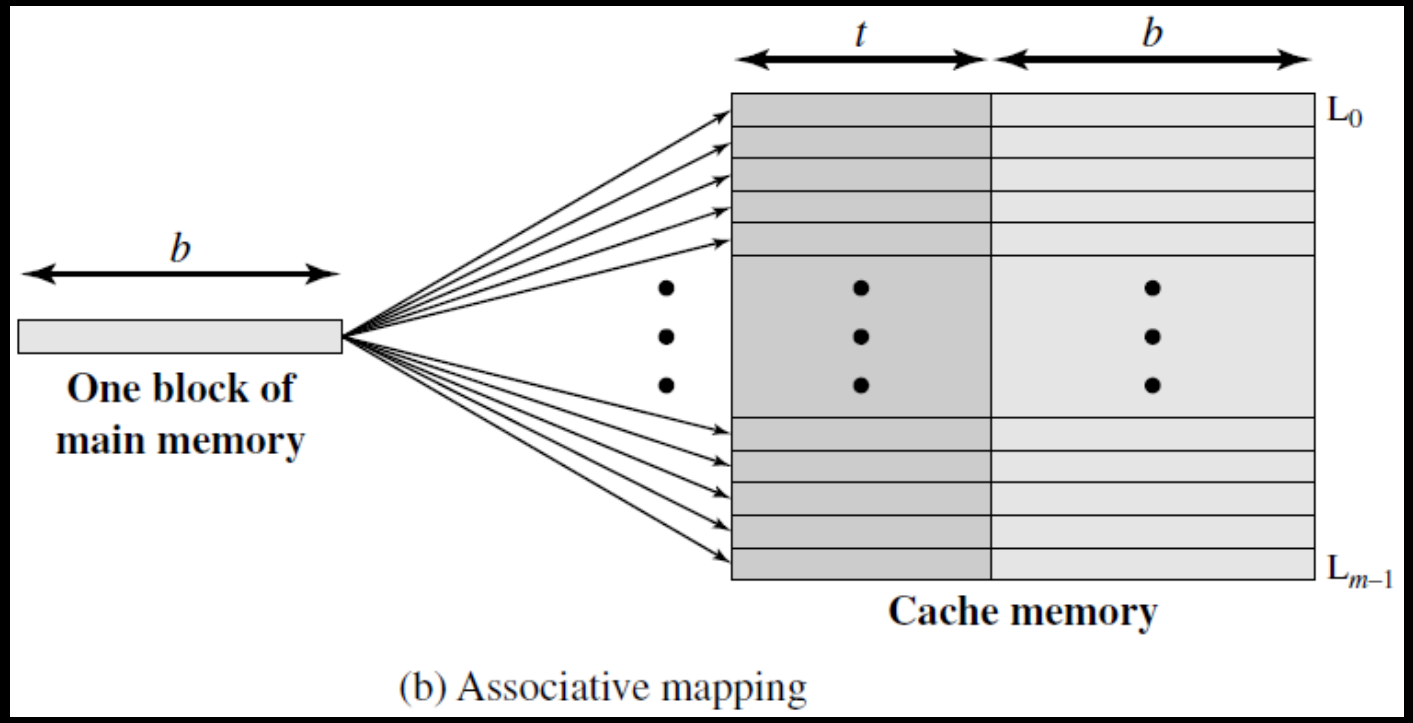
Elements of Cache Design – Direct Mapping

- The data length = $4\ bytes * 8\ = 32\ bits$.
- The cache system is presented with a 24-bit address.
- The 14-bit line number is used as an index into the cache to access a particular line.
- If the 8-bit tag number matches the tag number currently stored in that line, then the 2-bit word number is used to select one of the 4 bytes in that line.
- Otherwise, the 22-bit tag-plus-line field is used to fetch a block from main memory.
- The actual address that is used for the fetch is the 22-bit tag-plus-line concatenated with two 0 bits, so that 4 bytes are fetched starting on a block boundary.



Elements of Cache Design – Associative Mapping

- Permits each main memory block to be loaded into any line of the cache.
  - The cache control logic interprets a memory address simply as a **Tag** and a **Word** field.
  - The **Tag** field uniquely identifies a block of main memory.



(b) Associative mapping

Elements of Cache Design – Associative Mapping

The organization of associative cache mapping is as follows:

- We have a memory with address length = $(s + w)$ bits.
  - The number of addressable units is 2^{s+w} words or bytes.
- Block size = line size = 2^w words or bytes.
  - Number of blocks in main memory is $\frac{2^{s+w}}{2^w} = 2^s$.

Elements of Cache Design – Associative Mapping

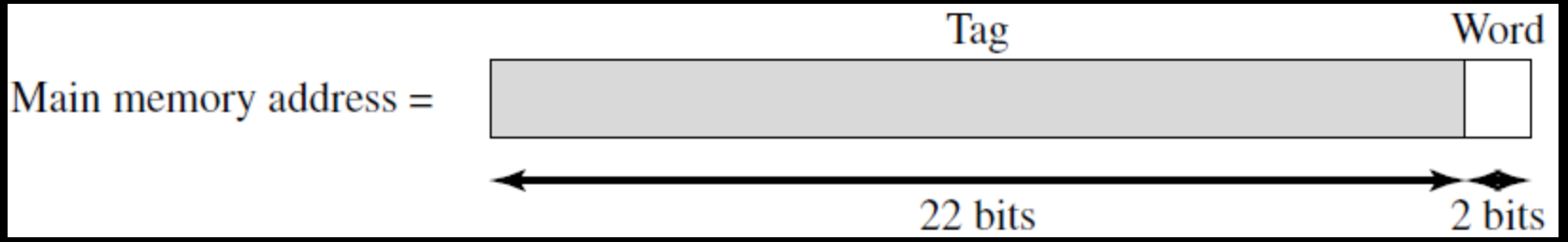
The organization of associative cache mapping is as follows:

- We have a memory with address length = $(s + w)$ bits.
  - The number of addressable units is 2^{s+w} words or bytes.
- Block size = line size = 2^w words or bytes.
  - Number of blocks in main memory is $\frac{2^{s+w}}{2^w} = 2^s$.
- Number of lines in cache = undetermined
- Size of tag = s bits.

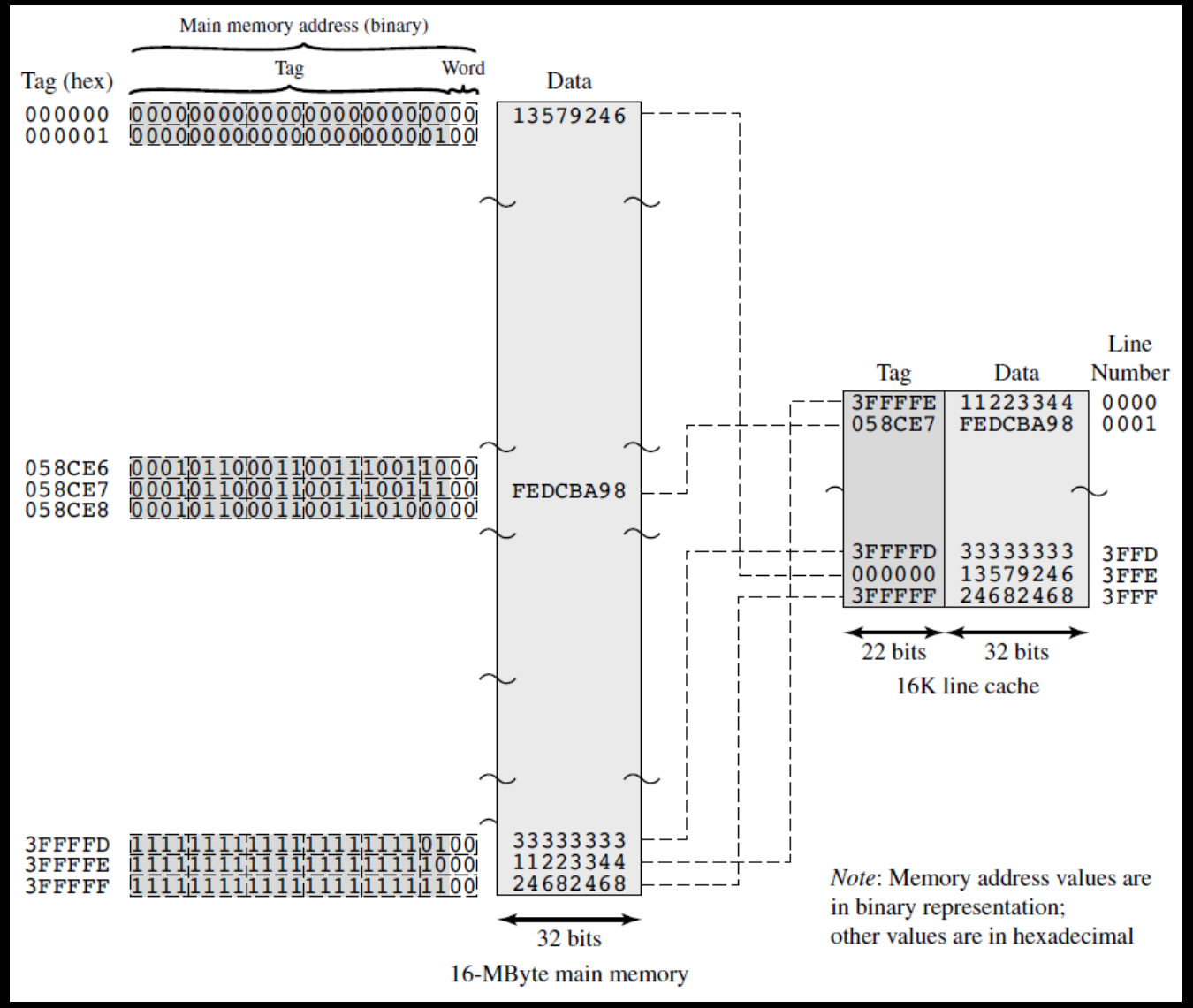
Elements of Cache Design – Associative Mapping

Example 4.2b Figure 4.12 shows our example using associative mapping. A main memory address consists of a 22-bit tag and a 2-bit byte number. The 22-bit tag must be stored with the 32-bit block of data for each line in the cache. Note that it is the leftmost (most significant) 22 bits of the address that form the tag. Thus, the 24-bit hexadecimal address 16339C has the 22-bit tag 058CE7. This is easily seen in binary notation:

memory address	0001	0110	0011	0011	1001	1100	(binary)
	1	6	3	3	9	C	(hex)
tag (leftmost 22 bits)	00	0101	1000	1100	1110	0111	(binary)
	0	5	8	C	E	7	(hex)



Elements of Cache Design – Associative Mapping



Elements of Cache Design – Set Associative Mapping

- The cache consists of a number sets, each of which consists of a number of lines.
- The relationships are

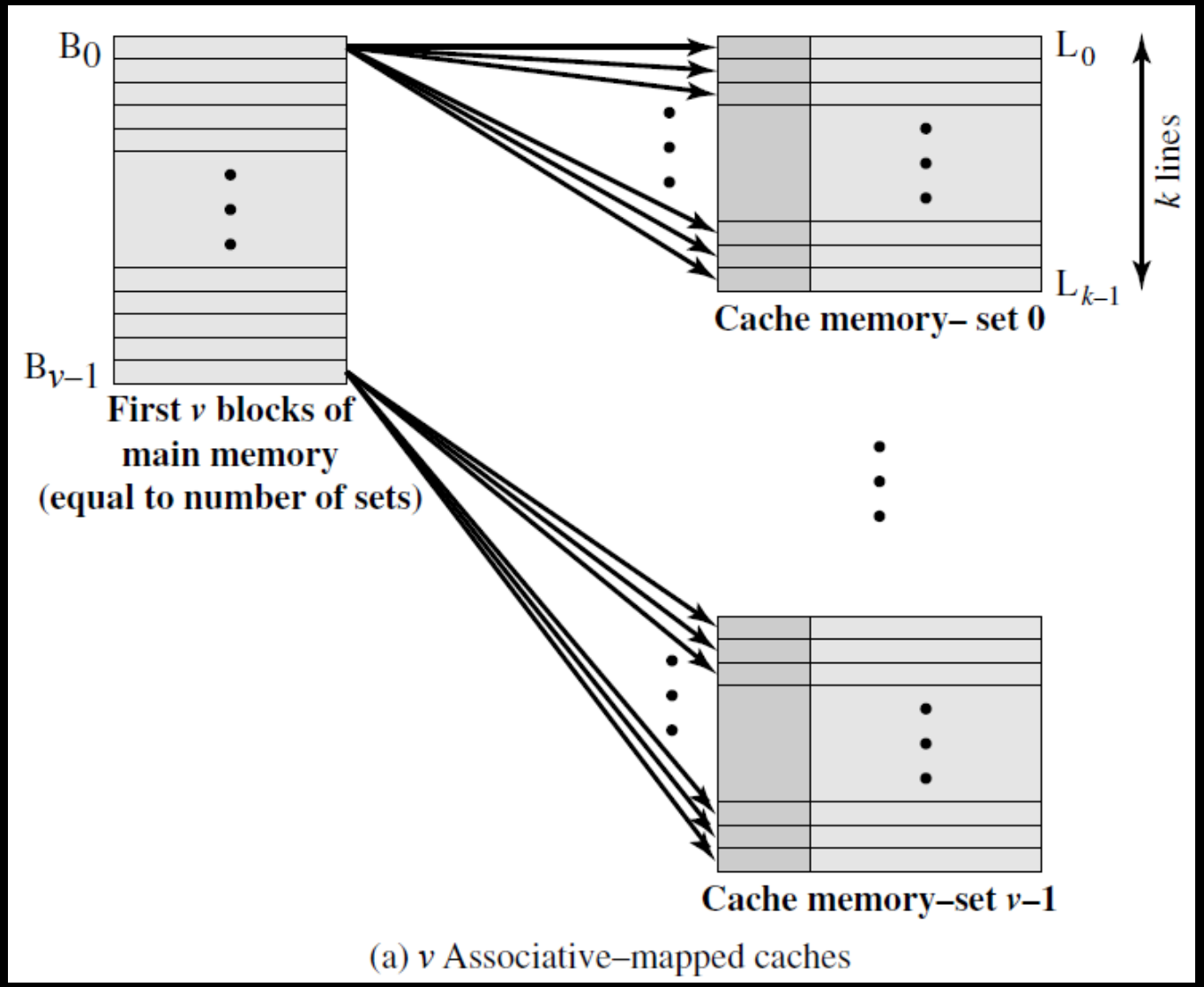
$$m = v * k$$
$$i = j \; mod \; v$$

where

- i = cache set number
- j = main memory block number
- m = number of lines in the cache
- v = number of sets
- k = number of lines in each set

Elements of Cache Design – Set Associative Mapping

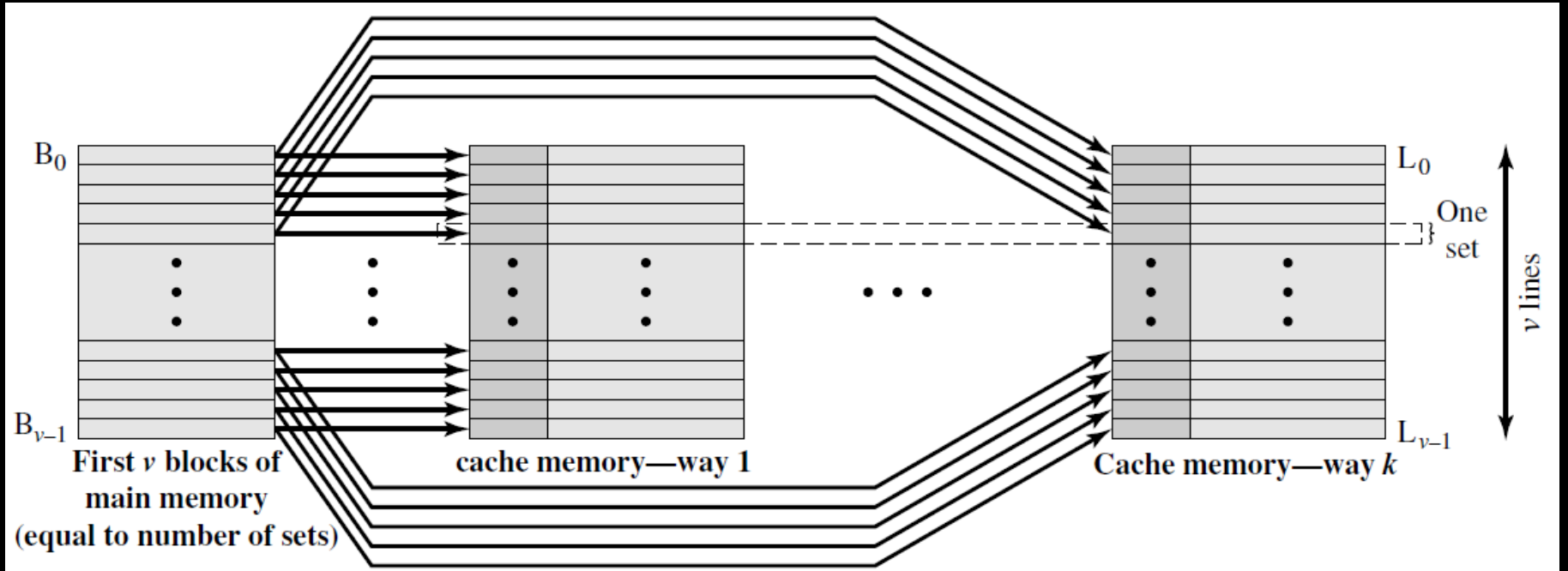
- This is referred to as k-way set-associative mapping.
  - Block B_j can be mapped into any of the lines of set j.



(a) v Associative–mapped caches

Elements of Cache Design – Set Associative Mapping

- The set-associative cache can be implemented as k –direct mapping caches.
  - Each direct-mapped cache is referred to as a **way**, consisting of lines.
  - The first lines of main memory are direct mapped into the lines of each way.



(b) k Direct–mapped caches

Elements of Cache Design – Set Associative Mapping

- The cache control logic interprets a memory address as three fields: **Tag**, **Set**, and **Word**.

Elements of Cache Design – Set Associative Mapping

The organization of associative cache mapping is as follows:

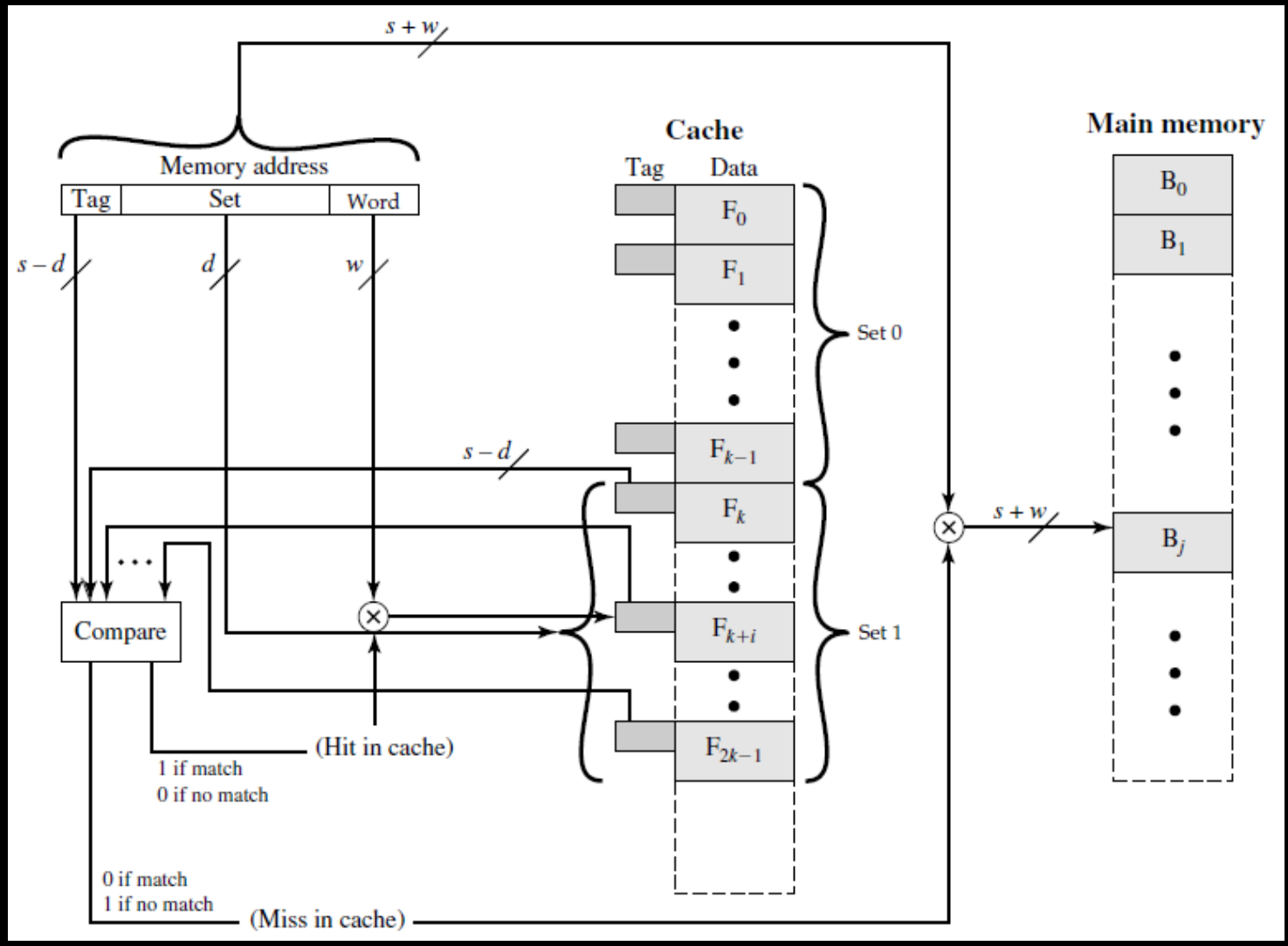
- We have a memory with address length = $(s + w)$ bits.
  - The number of addressable units is 2^{s+w} words or bytes.
- Block size = line size = 2^{w} words or bytes.
  - Number of blocks in main memory is $\frac{2^{s+w}}{2^{w}} = 2^{s}$.

Elements of Cache Design – Set Associative Mapping

The organization of associative cache mapping is as follows:

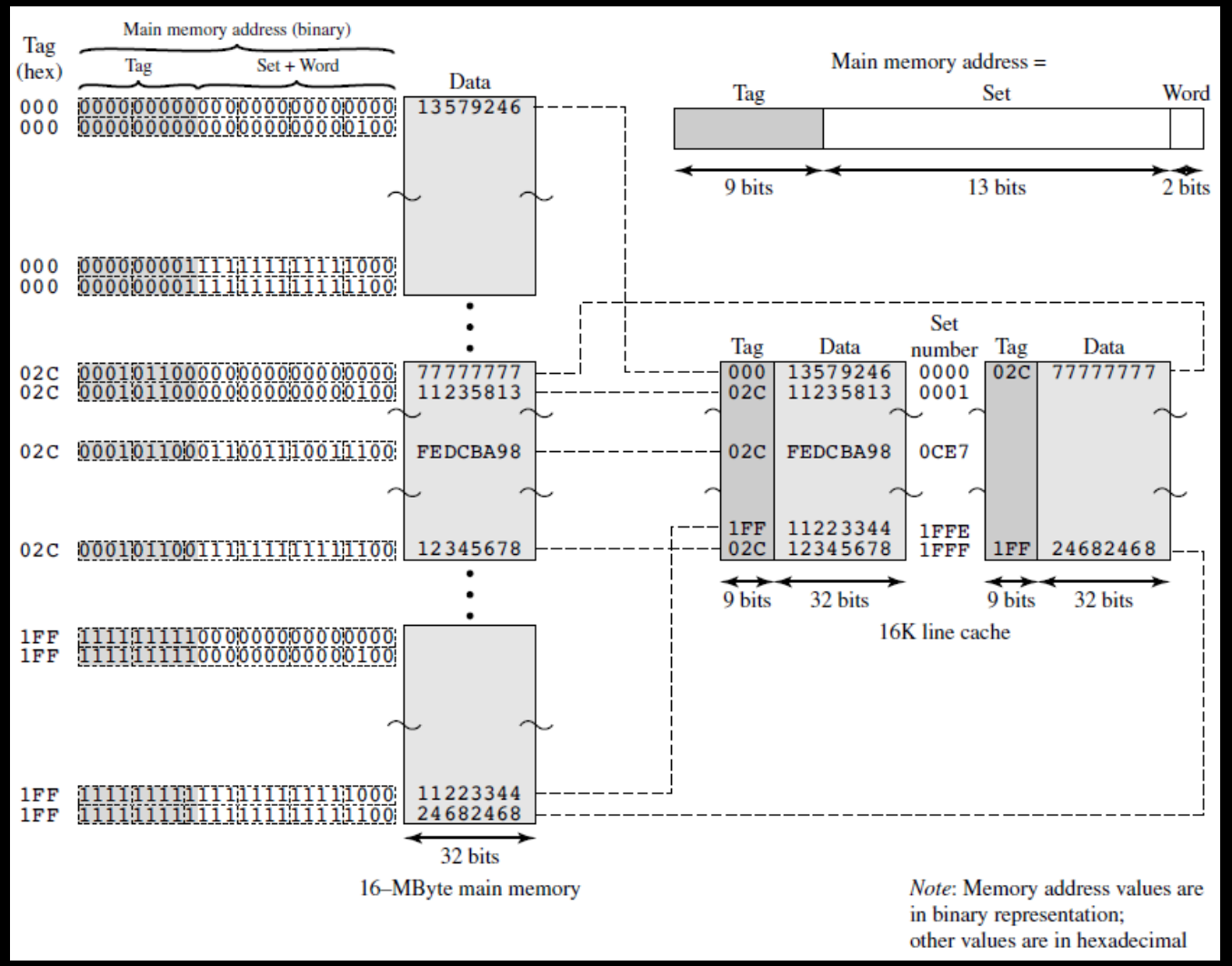
- We have a memory with address length = $(s + w)$ bits.
  - The number of addressable units is 2^{s+w} words or bytes.
- Block size = line size = 2^w words or bytes.
  - Number of blocks in main memory is $\frac{2^{s+w}}{2^w} = 2^s$.
- Number of lines = k.
- Number of sets = $v = 2^d$.
- Number of lines in cache = $m = kv = k * 2^d$
- Size of cache = $k * 2^{d+w}$ words or bytes
- Size of tag = $(s - d)$ bits

Elements of Cache Design – Set Associative Mapping



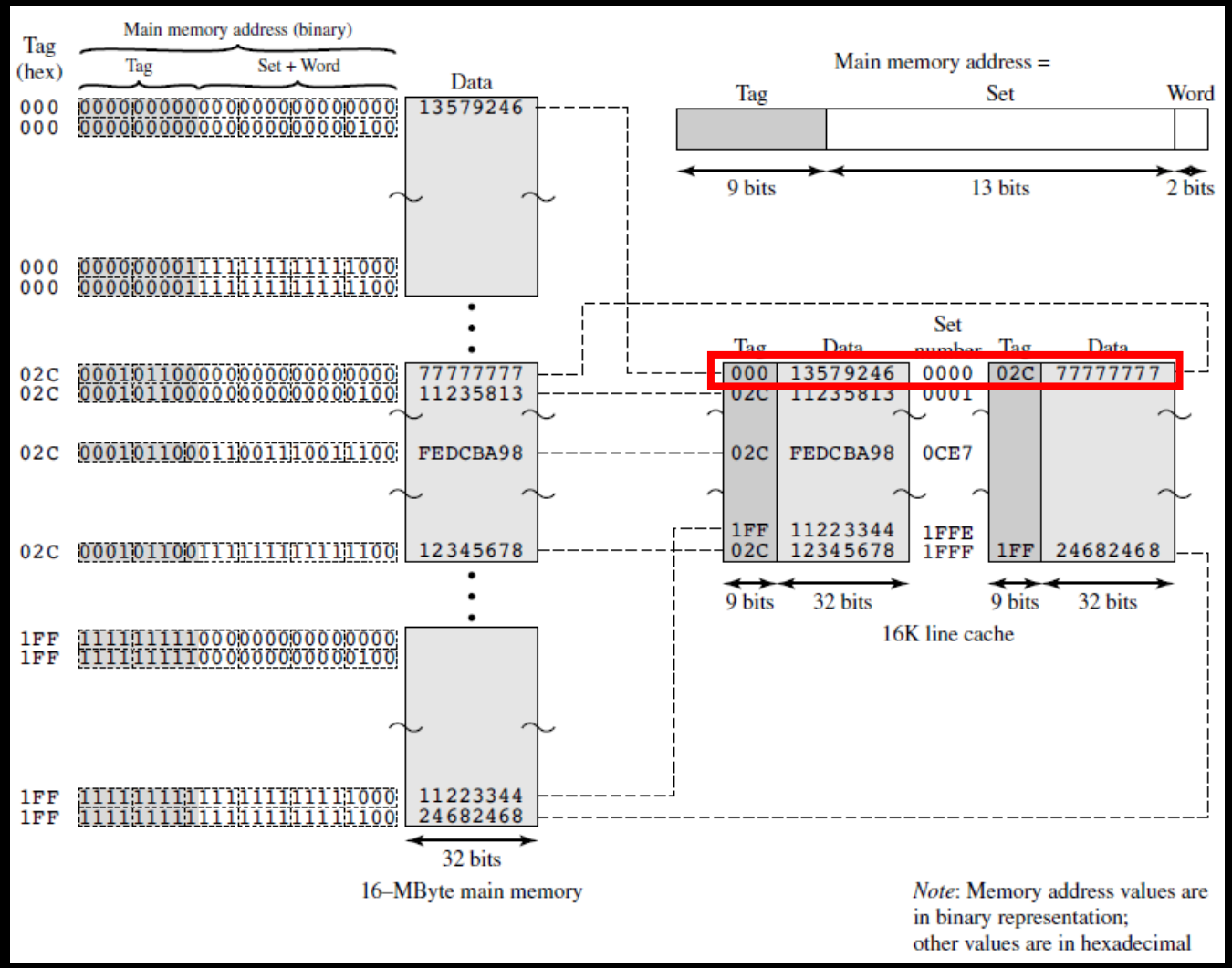
Elements of Cache Design – Set Associative Mapping

- **Example 4.2c** Figure 4.15 shows our example using set-associative mapping with two lines in each set, referred to as two-way set-associative. The 13-bit set number identifies a unique set of two lines within the cache. It also gives the number of the block in main memory, modulo 2^{13}.
  This determines the mapping of blocks into lines. Thus, blocks 000000, 008000, FF8000 of main memory map into cache set 0. Any of those blocks can be loaded into either of the two lines in the set. Note that no two blocks that map into the same cache set have the same tag number.
  For a read operation, the 13-bit set number is used to determine which set of two lines is to be examined. Both lines in the set are examined for a match with the tag number of the address to be accessed.



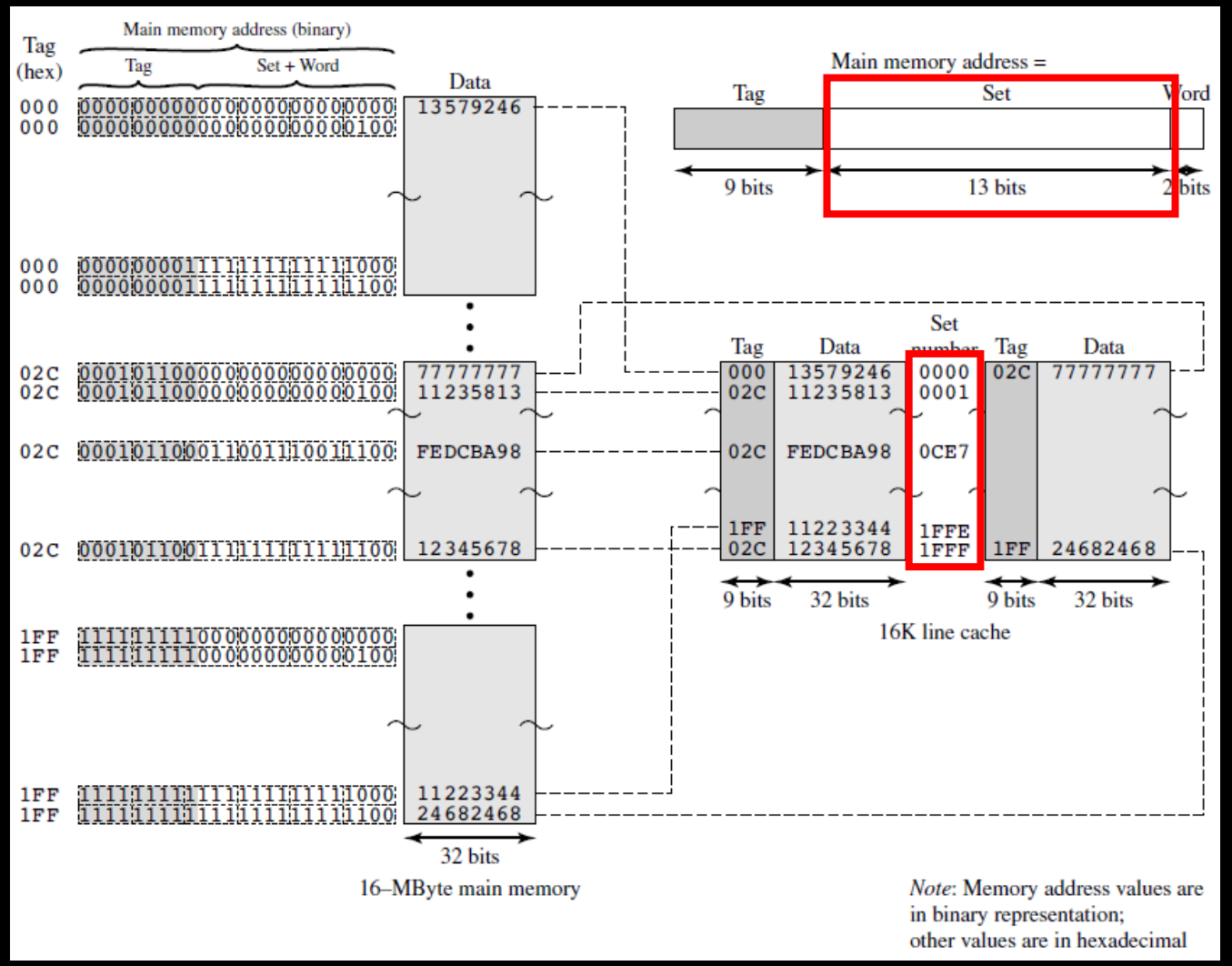
Elements of Cache Design – Set Associative Mapping

- **Example 4.2c** Figure 4.15 shows our example using set-associative mapping with two lines in each set, referred to as two-way set-associative. The 13-bit set number identifies a unique set of two lines within the cache. It also gives the number of the block in main memory, modulo 2^{13}. This determines the mapping of blocks into lines. Thus, blocks 000000, 008000, FF8000 of main memory map into cache set 0. Any of those blocks can be loaded into either of the two lines in the set. Note that no two blocks that map into the same cache set have the same tag number. For a read operation, the 13-bit set number is used to determine which set of two lines is to be examined. Both lines in the set are examined for a match with the tag number of the address to be accessed.



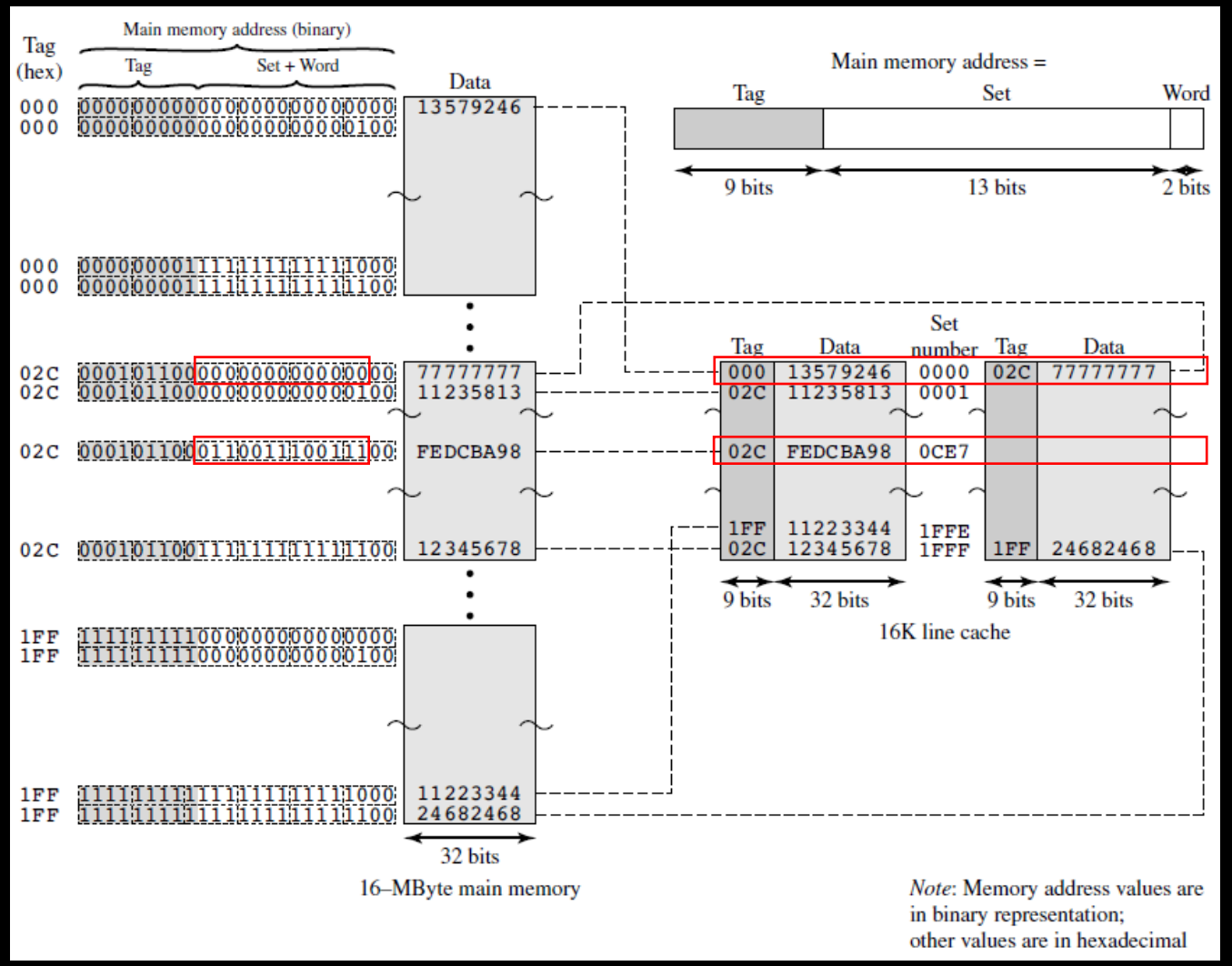
Elements of Cache Design – Set Associative Mapping

- **Example 4.2c** Figure 4.15 shows our example using set-associative mapping with two lines in each set, referred to as two-way set-associative. The 13-bit set number identifies a unique set of two lines within the cache. It also gives the number of the block in main memory, modulo 2^{13}.
  This determines the mapping of blocks into lines. Thus, blocks 000000, 008000, FF8000 of main memory map into cache set 0. Any of those blocks can be loaded into either of the two lines in the set. Note that no two blocks that map into the same cache set have the same tag number.
  For a read operation, the 13-bit set number is used to determine which set of two lines is to be examined. Both lines in the set are examined for a match with the tag number of the address to be accessed.



Elements of Cache Design – Set Associative Mapping

- **Example 4.2c** Figure 4.15 shows our example using set-associative mapping with two lines in each set, referred to as two-way set-associative. The 13-bit set number identifies a unique set of two lines within the cache. It also gives the number of the block in main memory, modulo 2^{13}. This determines the mapping of blocks into lines. Thus, blocks 000000, 008000, FF8000 of main memory map into cache set 0. Any of those blocks can be loaded into either of the two lines in the set. Note that no two blocks that map into the same cache set have the same tag number.
  For a read operation, the 13-bit set number is used to determine which set of two lines is to be examined. Both lines in the set are examined for a match with the tag number of the address to be accessed.



Exercises

4.1 A set-associative cache consists of 64 lines, or slots, divided into four-line sets. Main memory contains 4K blocks of 128 words each. Show the format of main memory addresses.

Exercises

We have:

- m (number of lines in the cache) = 64
- k (number of lines in each set) = 4
- Number of memory blocks = 4k
- Block size = 128

We need:

- Tag, Set, Word length in bits

Exercises

We have:

- m (number of lines in the cache) = 64
- k (number of lines in each set) = 4
- Number of memory blocks = 4k
- Block size = 128

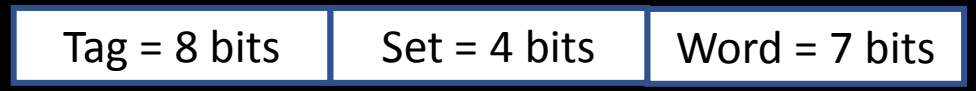
We need:

- Tag, Set, Word length in bits

$\therefore v$ (number of sets) $= 64 / 4 = 16 = 2^4$
$\therefore d$ (set bits) $= 4$ bits.

$\therefore$ memory size = $4 * 128 = 512KB = 2^{19}$
$\therefore$ address length $= 19$ bits
$\because$ each block has 128 words.
$\therefore w = 7$ bits $(2^7 = 128)$

$\therefore$ Tag $= 19 - (4 + 7) = 8$ bits.



Exercises

4.2 A two-way set-associative cache has lines of 16 bytes and a total size of 8 kbytes. The 64-Mbyte main memory is byte addressable. Show the format of main memory addresses.

Exercises

We have:

- Block size = line size = 16 bytes.
- Size of the cache = 8 Kbytes

- Size of the memory = 64 Mbytes.

We need:

Tag, Set, Word length in bits.

Exercises

We have:

- Block size = line size = 16 bytes.
- Size of the cache = 8 Kbytes.

$\therefore w = 4\ bits\ (16 = 2^4)$

$\therefore$ number of lines in the cache = $\frac{8KB}{16} = 512$ lines

$\because$ the cache is two-way … $\therefore$ number of sets = $\frac{512}{2} = 256$ set

each set has 2 lines.

$\therefore$ number of bits for each set = 8 bits $(256 = 2^8)$

- Size of the memory = 64 Mbytes.

$\therefore$ address length = $Log_2(64 * 1024 * 1024) = 26$

We need:

Tag, Set, Word length in bits.

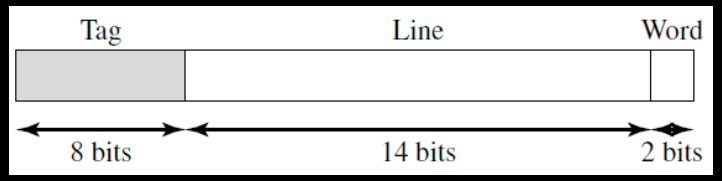
$\therefore$ tag length = $26 - (4 + 8) = 14$ bits

Tag = 14 bits	Set = 8 bits	Word = 4 bits

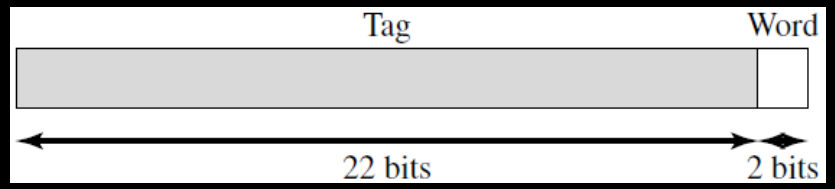
Exercises

4.3 For the hexadecimal main memory addresses 111111, 666666,BBBBBB, show the following information, in hexadecimal format:

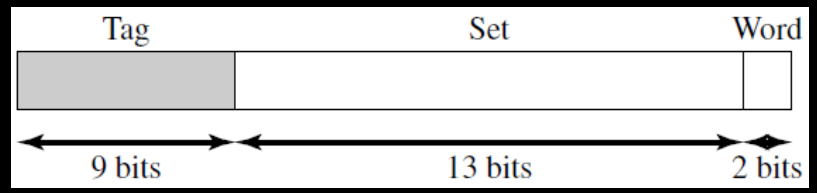
a. Tag, Line, and Word values for a direct-mapped cache, using the format of Figure 4.10



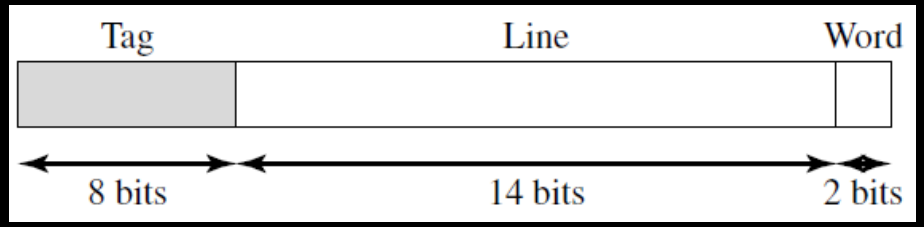
b. Tag and Word values for an associative cache, using the format of Figure 4.12



c. Tag, Set, and Word values for a two-way set-associative cache, using the format of Figure 4.15



Exercises



Convert the number into bits:

111111

0001 0001 0001 0001 000100 01

Tag	Line	Word
11	444	1

666666

0110 0110 0110 0110 011001 10

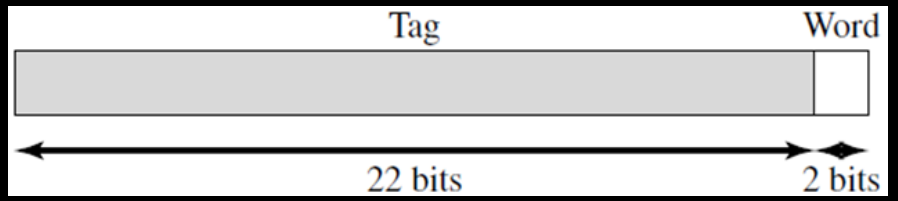
Tag	Line	Word
66	1999	2

BBBBBB

1011 1011 1011 1011 101110 11

Tag	Line	Word
BB	2EEE	3

Exercises



Convert the number into bits:

111111

0001 0001 0001 0001 000100 01

Tag: 44444

Word: 1

666666

0110 0110 0110 0110 011001 10

Tag: 199999

Word: 2

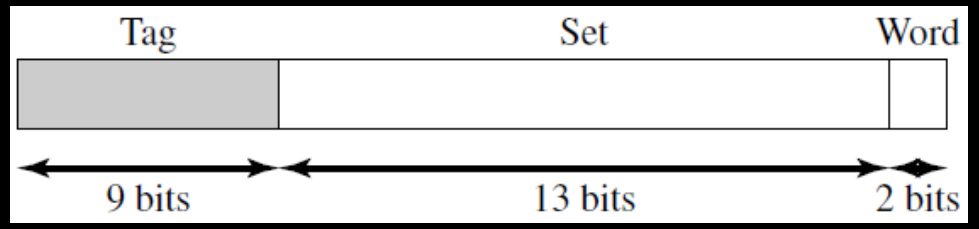
BBBBBB

1011 1011 1011 1011 101110 11

Tag: 2EEEEE

Word: 3

Exercises



Convert the number into bits:

111111

0001 0001 0001 0001 000100 01

Tag	Set	Word
22	444	1

666666

0110 0110 0110 0110 011001 10

Tag	Set	Word
CC	1999	2

BBBBBB

1011 1011 1011 1011 101110 11

Tag	Set	Word
177	EEE	3

Exercises

4.5 Consider a 32-bit microprocessor that has an on-chip 16-KByte four-way set-associative cache. Assume that the cache has a line size of four 32-bit words. Draw a block diagram of this cache showing its organization and how the different address fields are used to determine a cache hit/miss. Where in the cache is the word from memory location ABCDE8F8 mapped?

Exercises

32-bit microprocessor → the address length is 32 bit.

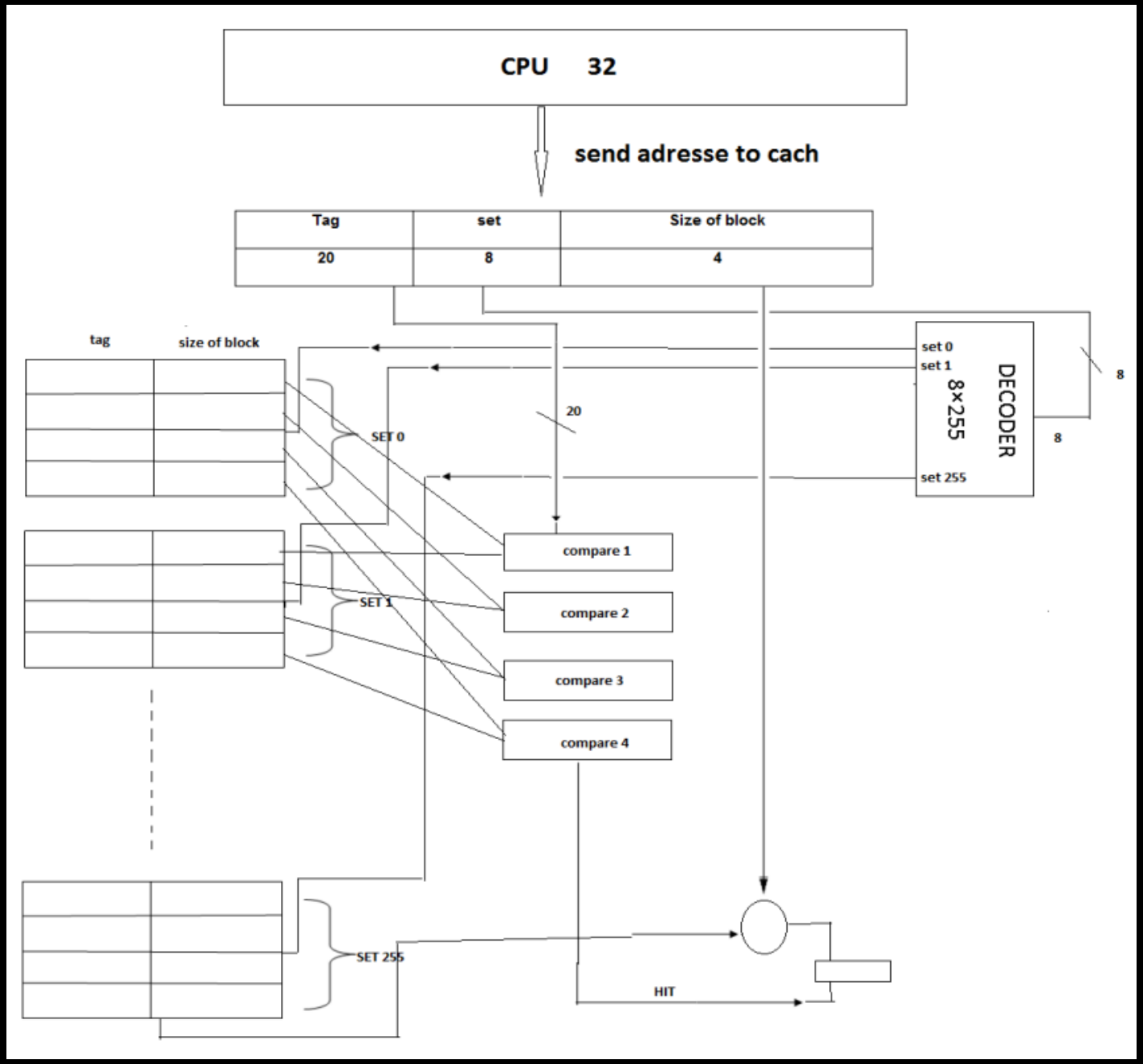
The cache is size is 16KB,
the line has 4 words, each word is 32 bits → the line size $= 4 * 32 =$
$128\ bits\ =\ 16\ bytes$ → number of lines in the cache $= \frac{16KB}{16B} = 1024$ lines.

The cache is 4-way → the number of sets $= \frac{1024}{4} = 256$ sets.

Number of bits for a set $= 8$ bits.

The line size $= 16$ bytes $= 2^4$ → number of bits for $w = 4$.

The tag length = 32 – (4 + 8) = 20 bits.


CPU 32
send adresse to cach
Tag
set
Size of block
20
8
4
tag
size of block
SET 0
SET 1
SET 255
20
8
8
set 0
set 1
set 255
8×255
DECODER
compare 1
compare 2
compare 3
compare 4
HIT

Exercises

Where in the cache is the word from memory location ABCDE8F8 mapped?

	Tag	Set	Word
Convert to binary: <1010101111001101111>	<10001111>	<1000>	
	ABCDE	8F	8

Exercises

4.8 Consider a machine with a byte addressable main memory of 2^{16} bytes and block size of 8 bytes. Assume that a direct mapped cache consisting of 32 lines is used with this machine.

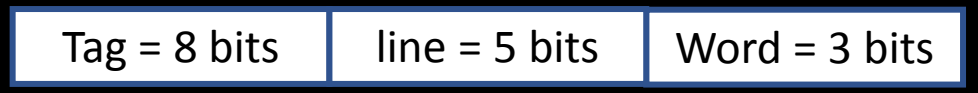
a. How is a 16-bit memory address divided into tag, line number, and byte number?

Exercises

4.8 Consider a machine with a byte addressable main memory of 2^{16} bytes and block size of 8 bytes. Assume that a direct mapped cache consisting of 32 lines is used with this machine.

a. How is a 16-bit memory address divided into tag, line number, and byte number?

Block size = 8 bytes = 2^3 → $w = 3$ bits.
The number of lines = 32 lines = 2^5 → line number length = 5 bits.
The tag length = 16 – (5 + 3) = 8 bits



Exercises

4.8 Consider a machine with a byte addressable main memory of 2^{16} bytes and block size of 8 bytes. Assume that a direct mapped cache consisting of 32 lines is used with this machine.

b. Into what line would bytes with each of the following addresses be stored?

0001 0001 0001 1011

1100 0011 0011 0100

1101 0000 0001 1101

1010 1010 1010 1010

Exercises

4.8 Consider a machine with a byte addressable main memory of 2^{16} bytes and block size of 8 bytes. Assume that a direct mapped cache consisting of 32 lines is used with this machine.

b. Into what line would bytes with each of the following addresses be stored?

Line length = 5 bit.
0001 0001 0001 1011 → Line 3
1100 0011 0011 0100 → Line 6
1101 0000 0001 1101 → Line 3
1010 1010 1010 1010 → Line 21

Exercises

4.8 Consider a machine with a byte addressable main memory of 2^{16} bytes and block size of 8 bytes. Assume that a direct mapped cache consisting of 32 lines is used with this machine.

c. Suppose the byte with address 0001 1010 0001 1010 is stored in the cache. What are the addresses of the other bytes stored along with it?

Exercises

4.8 Consider a machine with a byte addressable main memory of 2^{16} bytes and block size of 8 bytes. Assume that a direct mapped cache consisting of 32 lines is used with this machine.

c. Suppose the byte with address 0001 1010 0001 1010 is stored in the cache. What are the addresses of the other bytes stored along with it?

The first 3 bits identify the word number. So, the line starts with first 3 bits 0s and the ends with the first 3 bits 1s.
0001 1010 0001 1000 to *0001 1010 0001 1111*

Exercises

4.8 Consider a machine with a byte addressable main memory of 2^{16} bytes and block size of 8 bytes. Assume that a direct mapped cache consisting of 32 lines is used with this machine.

d. How many total bytes of memory can be stored in the cache?

Exercises

4.8 Consider a machine with a byte addressable main memory of 2^{16} bytes and block size of 8 bytes. Assume that a direct mapped cache consisting of 32 lines is used with this machine.

d. How many total bytes of memory can be stored in the cache?

The cache has 32 lines, each line can store 8 bytes (as block size), thus the cache size is 32 * 8 = 256 bytes.

Exercises

4.8 Consider a machine with a byte addressable main memory of 2^{16} bytes and block size of 8 bytes. Assume that a direct mapped cache consisting of 32 lines is used with this machine.

e. Why is the tag also stored in the cache?

Exercises

4.8 Consider a machine with a byte addressable main memory of 2^{16} bytes and block size of 8 bytes. Assume that a direct mapped cache consisting of 32 lines is used with this machine.

e. Why is the tag also stored in the cache?

Because a line in the cache can store more than one memory block, so the tag is required to distinguish between them.

Content

CH 04
Computer Memory System Overview
Cache Memory Principles
Elements of Cache Design
~~Pentium 4 Cache Organization~~
~~ARM Cache Organization~~

Elements of Cache Design

- Once the cache has been filled, when a new block is brought into the cache, one of the existing blocks must be replaced.
- For direct mapping, there is only one possible line for any block, and no choice is possible.
- For the associative and set associative techniques, a replacement algorithm is needed.

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

Replacement Algorithms – LRU

Least Recently Used (LRU)

- Replace the block in the set that has been in the cache longest with no reference to it.

Replacement Algorithms – LRU

- For two-way set associative.
  - Each line includes a USE bit.
  - When a line is referenced, its USE bit is set to 1 and the USE bit of the other line in that set is set to 0.
  - When a block is to be read into the set, the line whose USE bit is 0 is used.

STEP 1: initial

	USE
01234	0
56789	0

STEP 2: reference line 2

	USE
01234	0
56789	1

STEP 3: add new value 9999

	USE
9999	1
56789	0

Replacement Algorithms – LRU

- For a fully associative cache.
  - The cache maintains a separate list of indexes to all the lines in the cache.
  - When a line is referenced, it moves to the front of the list.
  - For replacement, the line at the back of the list is used.

STEP 1: initial

01234
56789
4562
7747

STEP 2: add new value 111

111
01234
56789
4562

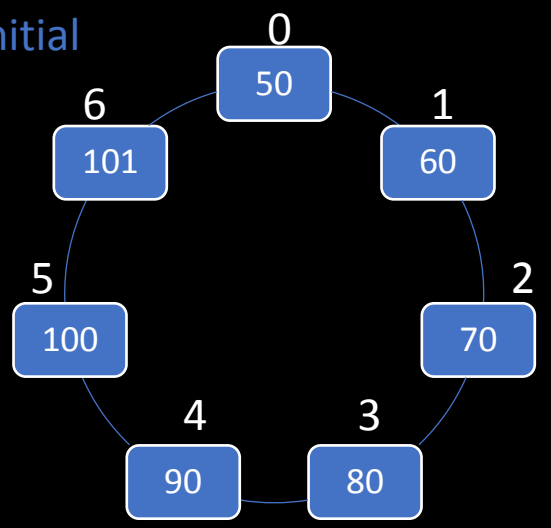
~~7747~~

Replacement Algorithms – FIFO

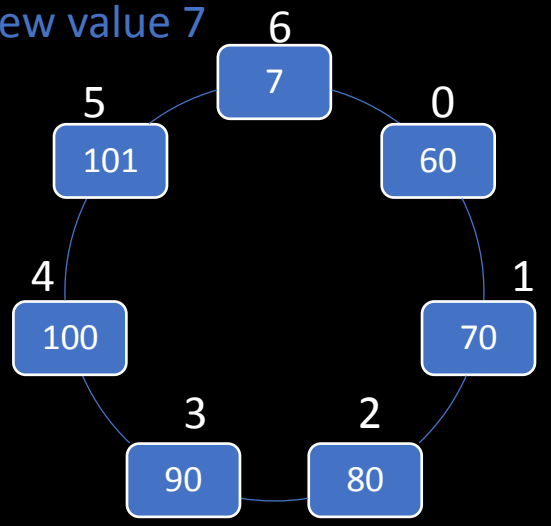
First-In-First-Out (FIFO):

- Replace that block in the set that has been in the cache longest.
- FIFO is implemented as a round-robin or circular buffer technique.

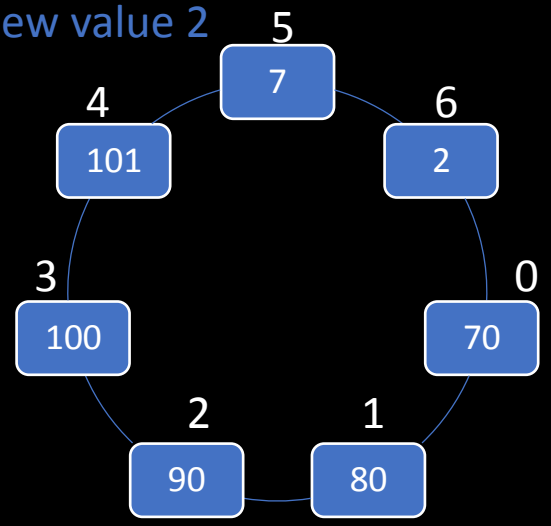
STEP 1: initial



STEP 2: add new value 7



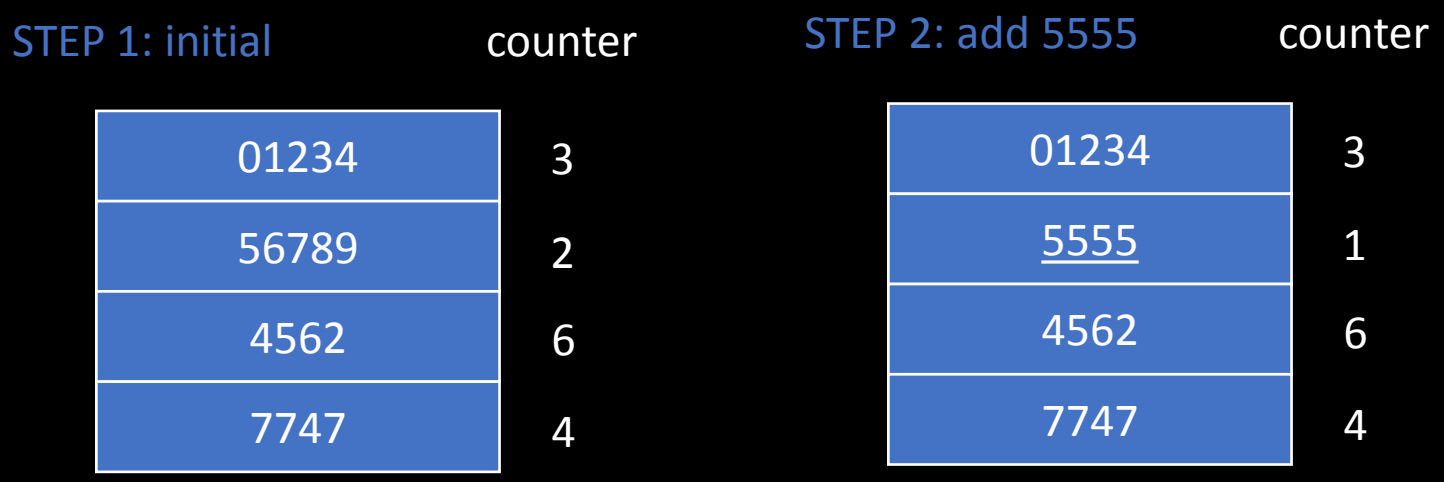
STEP 3: add new value 2



Replacement Algorithms – LFU

Least Frequently Used (LFU)

- Replace the block in the set that has experienced the fewest references.
- LFU could be implemented by associating a counter with each line.



Replacement Algorithms – Random

- A technique not based on usage is to pick a line at random from among the candidate lines.

Write Policy

When a block that is resident in the cache is to be replaced, there are two cases to consider.

- If the old block in the cache has not been modified, it can be replaced with the new block without writing it out to the memory.
- If at least on word of the block has been modified in the cache, it must be updated in the memory before replacing it in the cache.
- Write policies:
  - Write through
  - Write back

Write Policy

Write through

- All write operations are made to main memory as well as to the cache, ensuring that main memory is always valid.

Write back

- Updates are made only in the cache.
- When an update occurs, a *dirty bit*, or *use bit*, associated with the line is set.
- Then, when a block is replaced, it is written back to main memory if and only if the dirty bit is set.

Number of Caches

- Contemporary design of caches includes two levels of the caches:
  - L1 cache, this on-chip cache – embedded on the same chip of the processor.
  - L2 cache, this off-chip cache – not part of the processor's chip (external).

Why we use two levels of caches?

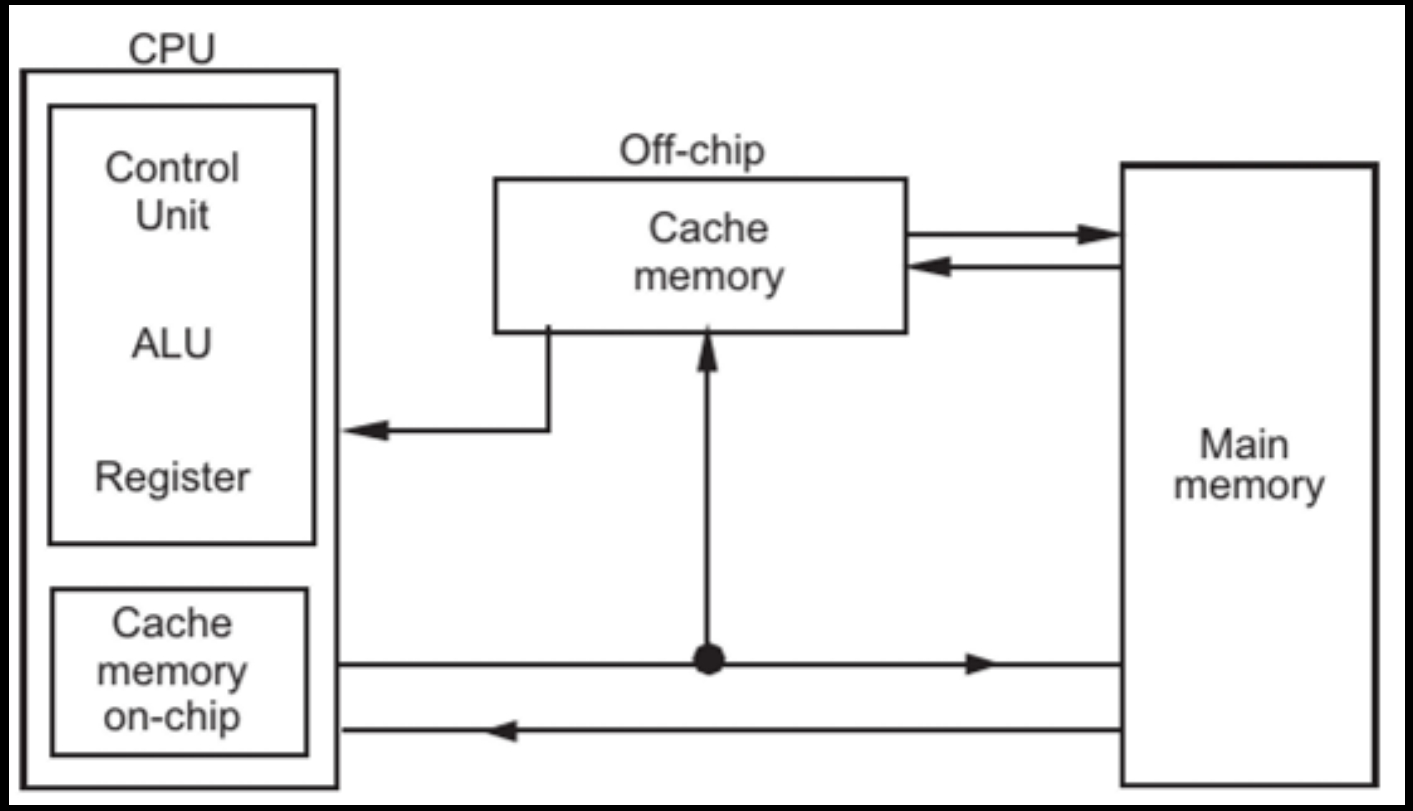
Number of Caches

- Contemporary design of caches includes two levels of the caches:
  - L1 cache, this on-chip cache – embedded on the same chip of the processor.
  - L2 cache, this off-chip cache – not part of the processor's chip (external).

Why we use two levels of caches?

- If there is no L2 cache and the processor makes an access request for a memory location not in the L1 cache, then the processor must access memory across the bus.
  - Due to the slow bus speed and slow memory access time, this results in poor performance.
- However, if an L2 cache is used, then the missing information can be quickly retrieved.

Number of Caches



Number of Caches

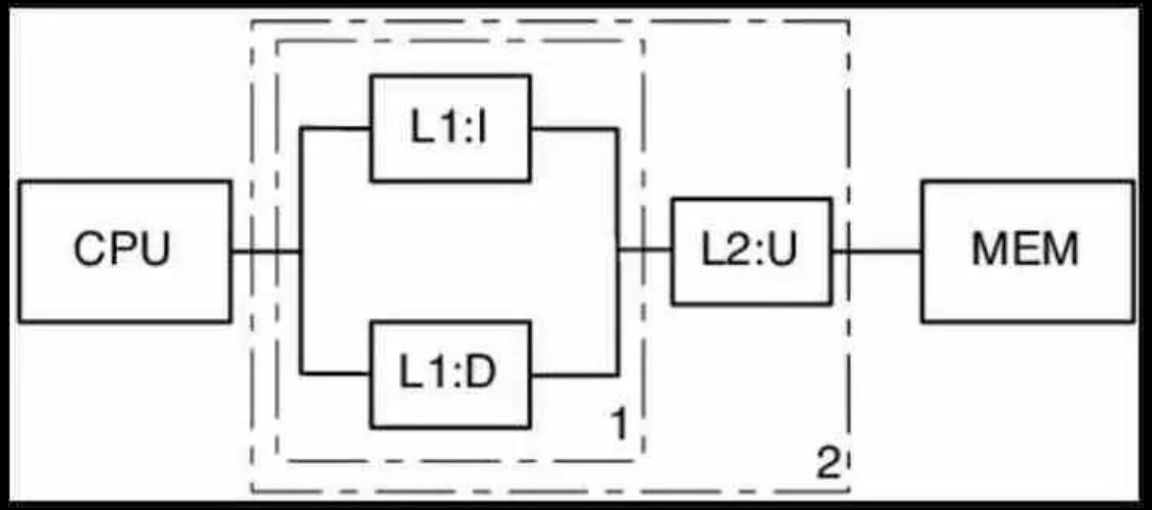
- A cache that stores both instructions and data is called a **unified cache**.
- An L1 cache can be separated into two parts, referred to as a **split cache**.
  - One cache for the instructions.
  - One cache for the data.
- When the processor fetches an instruction from main memory, it first consults the instruction L1 cache.
- When the processor fetches data from main memory, it first consults the data L1 cache.

Number of Caches

- What is the advantage of split cache?

Number of Caches

- What is the advantage of split cache?
- It eliminates contention for the cache between the instruction fetch/decode unit and the execution unit.
  - This enables the execution unit to fetch an instruction, at the same time fetching a data.



Exercises

4.21 Consider a single-level cache with an access time of 2.5 ns, a line size of 64 bytes, and a hit ratio of $H = 0.95$. Main memory uses a block transfer capability that has a first word (4 bytes) access time of 50 ns and an access time of 5 ns for each word thereafter.

a. What is the access time when there is a cache miss? Assume that the cache waits until the line has been fetched from main memory and then re-executes for a hit.

b. Suppose that increasing the line size to 128 bytes increases the H to 0.97. Does this reduce the average memory access time?

Exercises

When there is a cache miss, this means that that processor will take 2.5 ns for reading from the cache. If the data is not there, it will take 50 ns to fetch the first word from memory and 5 ns for fetching each of the remaining 16 word (line size = 64 bytes, each word = 4 bytes → each line has 64/4=16 words), plus another 2.5 for fetching the data from the cache again.

So, the access time is $T_{miss} = 2.5 + 50 + 5 * 16 + 2.5 = 130\ ns$

Exercises

After increasing the block size to 128, we get 128/4 = 32 words.

So, the access time for miss in the cache is

$$T_{miss} = 2.5 + 50 + 5 * 31 + 2.5 = 210\ ns$$

The average memory access time = Time on cache hit + Time on cache miss

$$= T_{hit} * hit\ ratio + T_{miss} * (1 - hit\ ratio)$$

Before increasing block size, $T_{avg} = 2.5\ * 0.95 + 130 * (1 - 0.95) = 8.875\ ns$

After increasing block size, $T_{avg} = 2.5\ * 0.97 + 210\ * (1 - 0.97) = 8.725\ ns$

Therefore, average memory access time is reduced.

Exercises

4.22 A computer has a cache, main memory, and a disk used for virtual memory. If a referenced word is in the cache, 20 ns are required to access it. If it is in main memory but not in the cache, 60 ns are needed to load it into the cache, and then the reference is started again. If the word is not in main memory, 12 ms are required to fetch the word from disk, followed by 60 ns to copy it to the cache, and then the reference is started again. The cache hit ratio is 0.9 and the main memory hit ratio is 0.6. What is the average time in nanoseconds required to access a referenced word on this system?

Exercises

Location of referenced word	Hit ratio	Time
In cache	0.9	20 ns
Not in cache, but in memory	(1-0.9) * 0.6	60 + 20 = 80 ns
Not in cache, not in memory	(1-0.9) * (1-0.6)	**12ms** + 60 + 20 = 12,000,80 ns

The average access time is

$$T_{avg} = 0.9 * 20 + (1 - 0.9) * 0.6 * 80 + (1 - 0.9) * (1 - 0.6) * 1200080 = 480026$$

TASK

CH 04
4.4
4.10