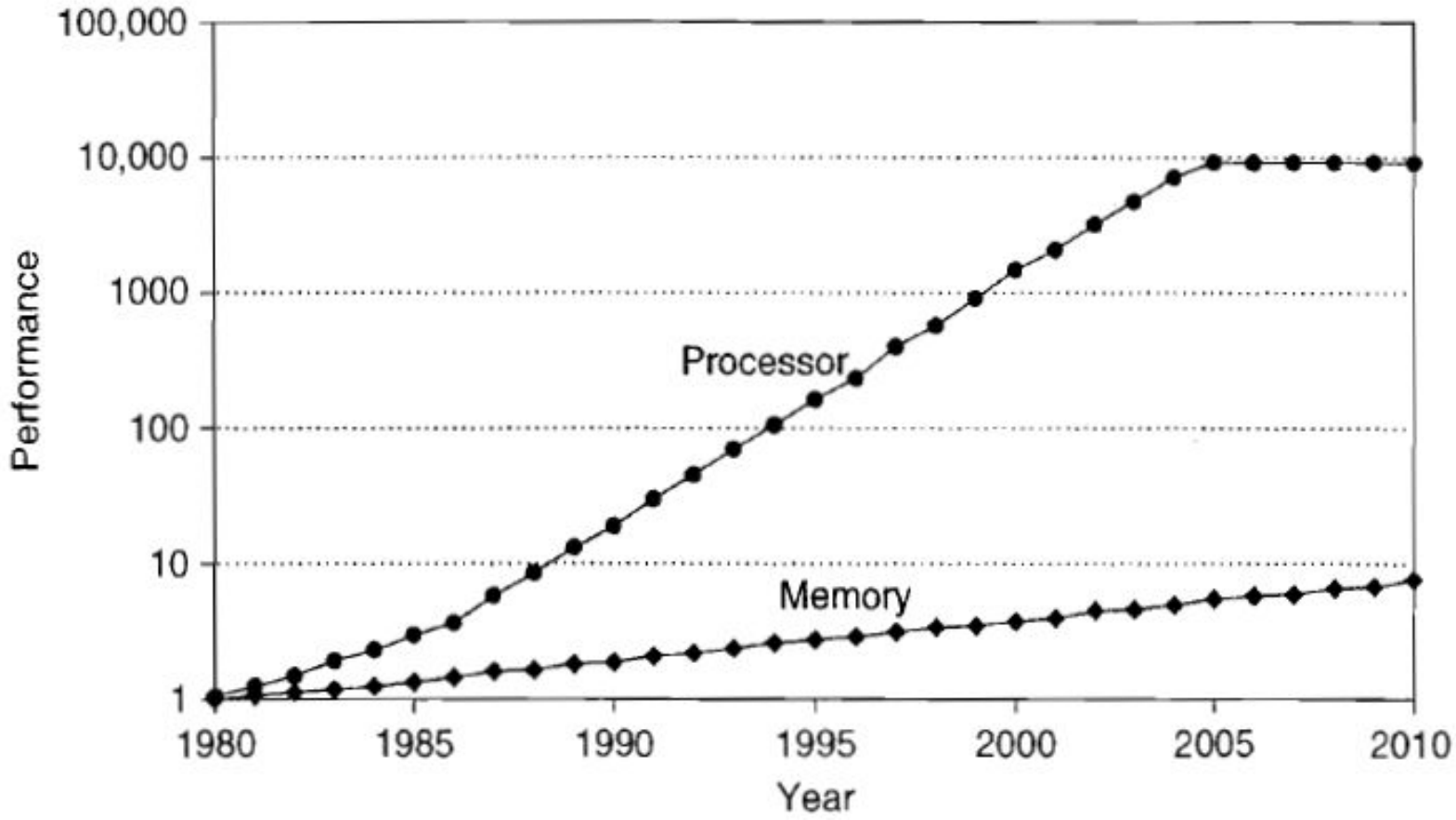
2

Memory Hierarchy Design

Source: Book1, Book5

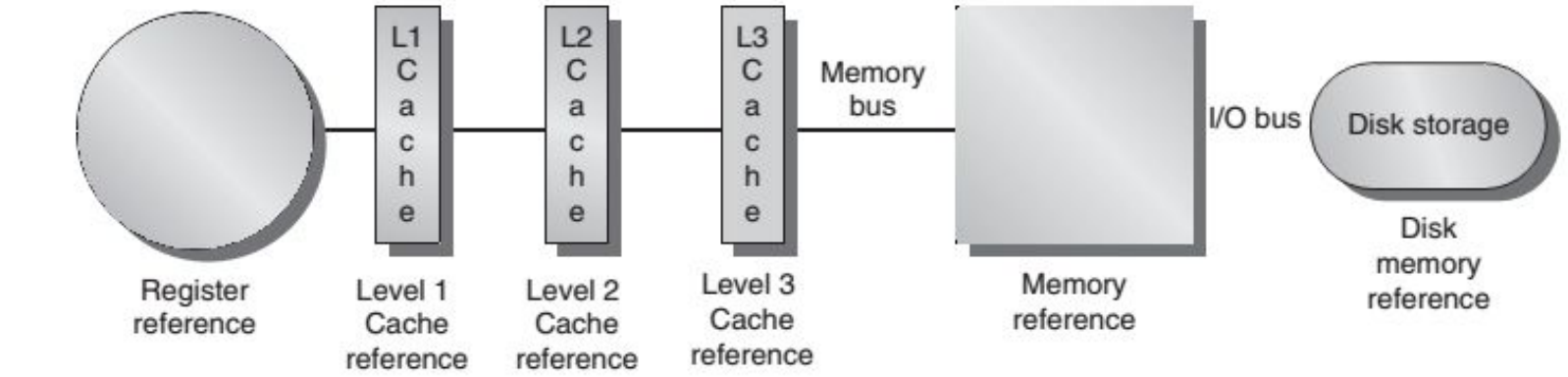
- **Source1:** Book1 (CH#2)
- **Source2:** Book 5 (CH#4): *Computer Organization and Architecture, Designing for Performance by* William Stallings (Chapter Uploaded on SLATE).

Processor-Memory Performance Gap

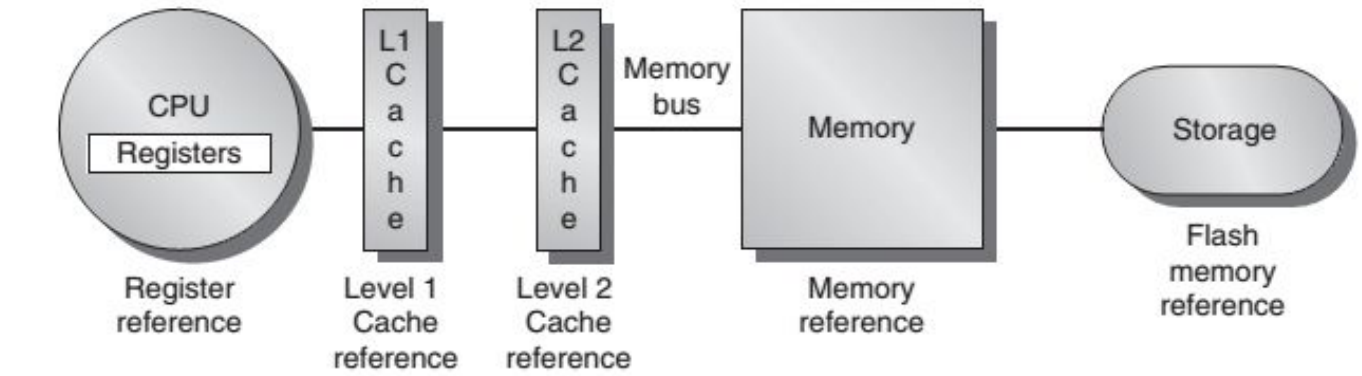


2.1 Introduction

- The ***principle of locality***: most programs do not access all code or data uniformly.
- This principle, plus the guideline that for a given implementation technology and power budget smaller hardware can be made faster, led to hierarchies based on memories of different speeds and sizes.
- Since fast memory is expensive, a memory hierarchy is organized into several levels—each smaller, faster, and more expensive per byte than the next lower level, which is farther from the processor.



(a) Memory hierarchy for server



(b) Memory hierarchy for a personal mobile device

Caches

- A **CPU cache** is a hardware cache used by the central processing unit (CPU) of a computer to reduce the average cost (time or energy) to access data from the main memory.

- A **cache** is a smaller, faster memory, closer to a processor core, which stores copies of the data from frequently used main memory locations.

- Most CPUs have different independent caches, including instruction and data caches, where the data cache is usually organized as a hierarchy of more cache levels (L1, L2, etc.).

https://en.wikipedia.org/wiki/CPU_cache

Primary Cache Vs Secondary Cache

- Cache memory is an extremely fast memory type that acts as a buffer between RAM and the CPU.
  - It holds frequently requested data and instructions so that they are immediately available to the CPU when needed.
- Cache memory is located in two general locations: inside the processor (internal cache) and on the motherboard (external cache):

- Internal cache: also known as **primary cache**, internal cache is located inside the CPU chip.

- External cache: also known as **secondary cache**, external cache is located on the motherboard outside the CPU. This is the cache referred to on PC specifications.

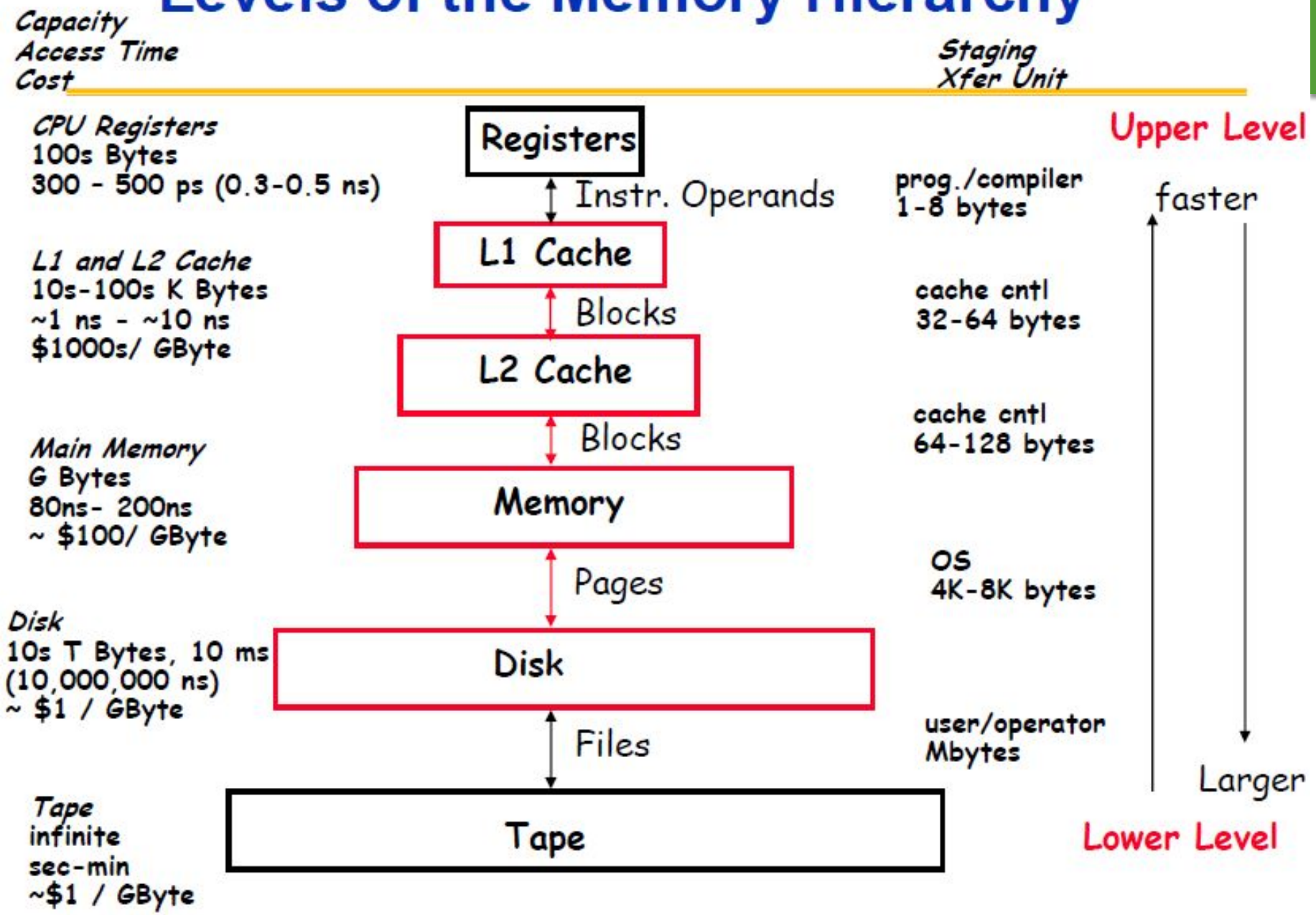
https://www.sqa.org.uk/e-learning/HardOSSupp01CD/page_19.htm

L1, L2, L3 Caches

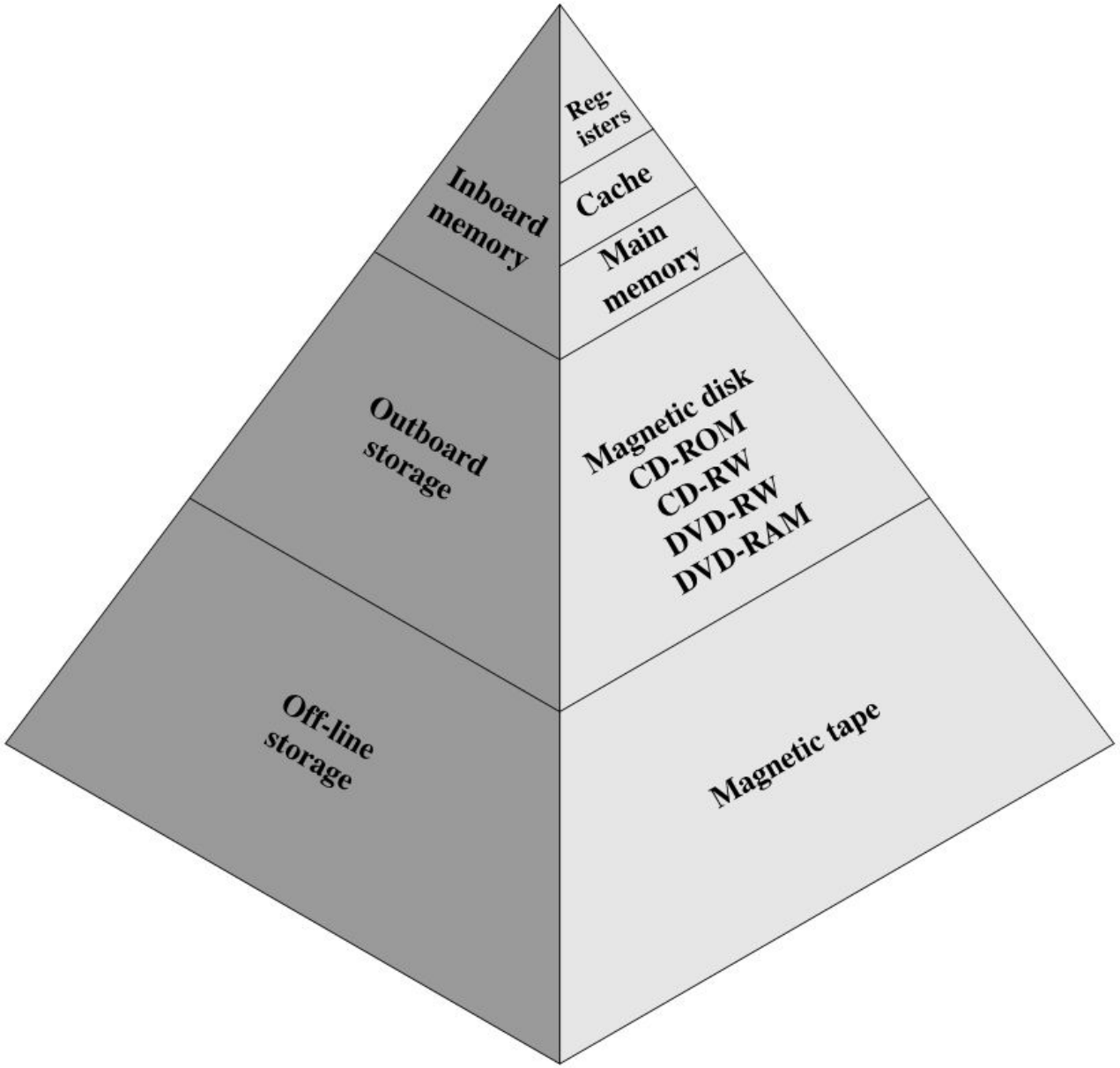
- Level 1 (**L1**) cache: L1 cache is placed internally on the processor chip and is, of course, the cache memory closest to the CPU.
- Level 2 (**L2**) cache: L2 cache, on older systems, is normally placed on the motherboard close to the CPU. Manufacturers today have both L1 and L2 cache installed on the CPU.
- Level 3 (**L3**) cache: L3 cache on new systems relates to the cache on the motherboard.

https://www.sqa.org.uk/e-learning/HardOSSupp01CD/page_19.htm

Levels of the Memory Hierarchy



http://rickvanderzwet.nl/trac/personal/browser/liacs/ca/slides/lecture01.pdf?rev=3

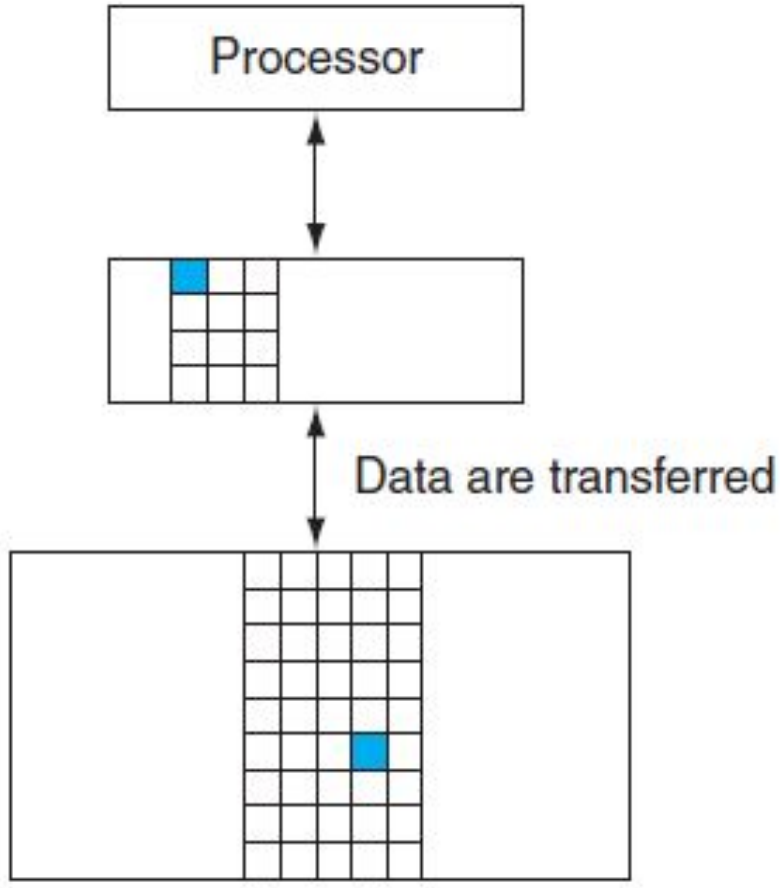

Reg-
isters
Cache
Main
memory
Inboard
memory
Outboard
storage
Magnetic disk
CD-ROM
CD-RW
DVD-RW
DVD-RAM
Off-line
storage
Magnetic tape

Data Transfer b/w Two Adjacent Levels

block The minimum unit of information that can be either present or not present in the two-level hierarchy.

hit rate The fraction of memory accesses found in a cache.

miss rate The fraction of memory accesses not found in a level of the memory hierarchy.



Hit Time and Miss Penalty

- Since performance is the major reason for having a memory hierarchy, the time to service hits and misses is important.

hit time The time required to access a level of the memory hierarchy, including the time needed to determine whether the access is a hit or a miss.

miss penalty The time required to fetch a block into a level of the memory hierarchy from the lower level, including the time to access the block, transmit it from one level to the other, and insert it in the level that experienced the miss.

The Basics of Caches

- ***Cache: a safe place for hiding or storing things.***
- Caches first appeared in research computers in the early 1960s and in production computers later in that same decade
- Every general-purpose computer built today, from servers to low-power embedded processors, includes caches

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

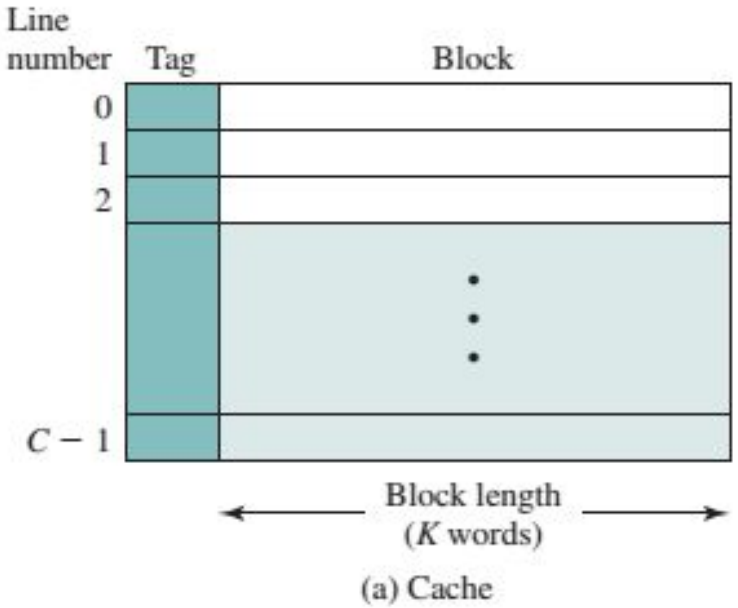
Number of Caches
- Single or two level
- Unified or split

Logical Vs Physical Caches

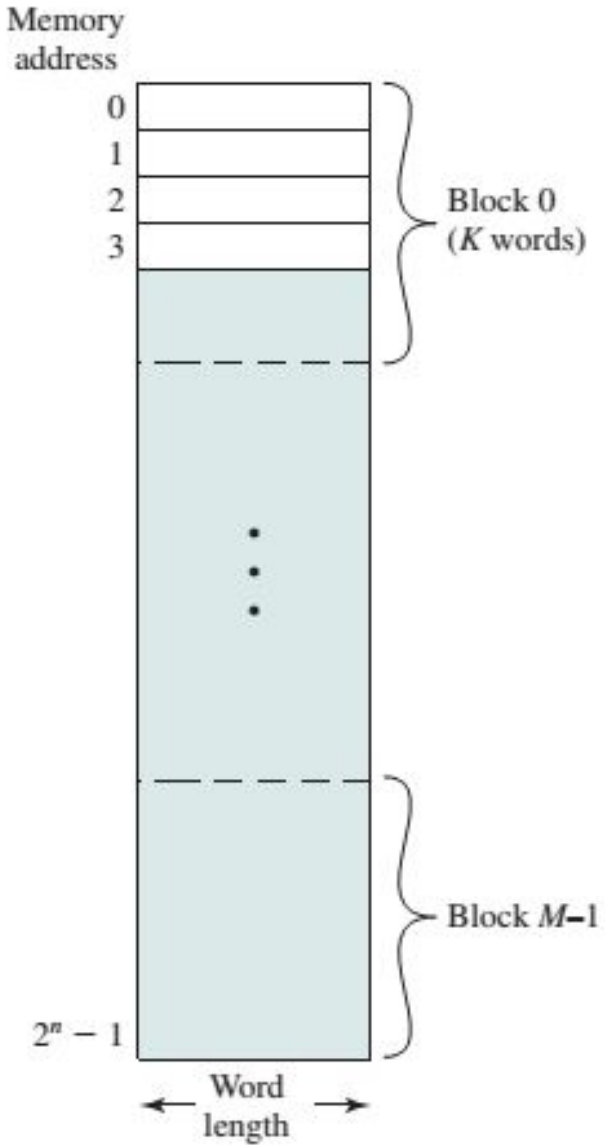
- A **logical cache**, also known as a **virtual cache**, stores data using **virtual addresses**.
- A physical cache stores data using main memory **physical addresses**.

Cache Organization

- Main memory consists of up to 2^n addressable words, with each word having a unique n-bit address.
  - For mapping purposes, this memory is considered to consist of a number of **fixed-length blocks** of K words each.
  - That is, there are $M = 2^n/K$ blocks in main memory.
- The cache consists of m blocks, called **lines**.
  - In referring to the basic unit of the cache, the term *line* is used, rather than the term block.
- Each line contains K words plus a tag of a few bits. Each line also includes control bits.



(a) Cache



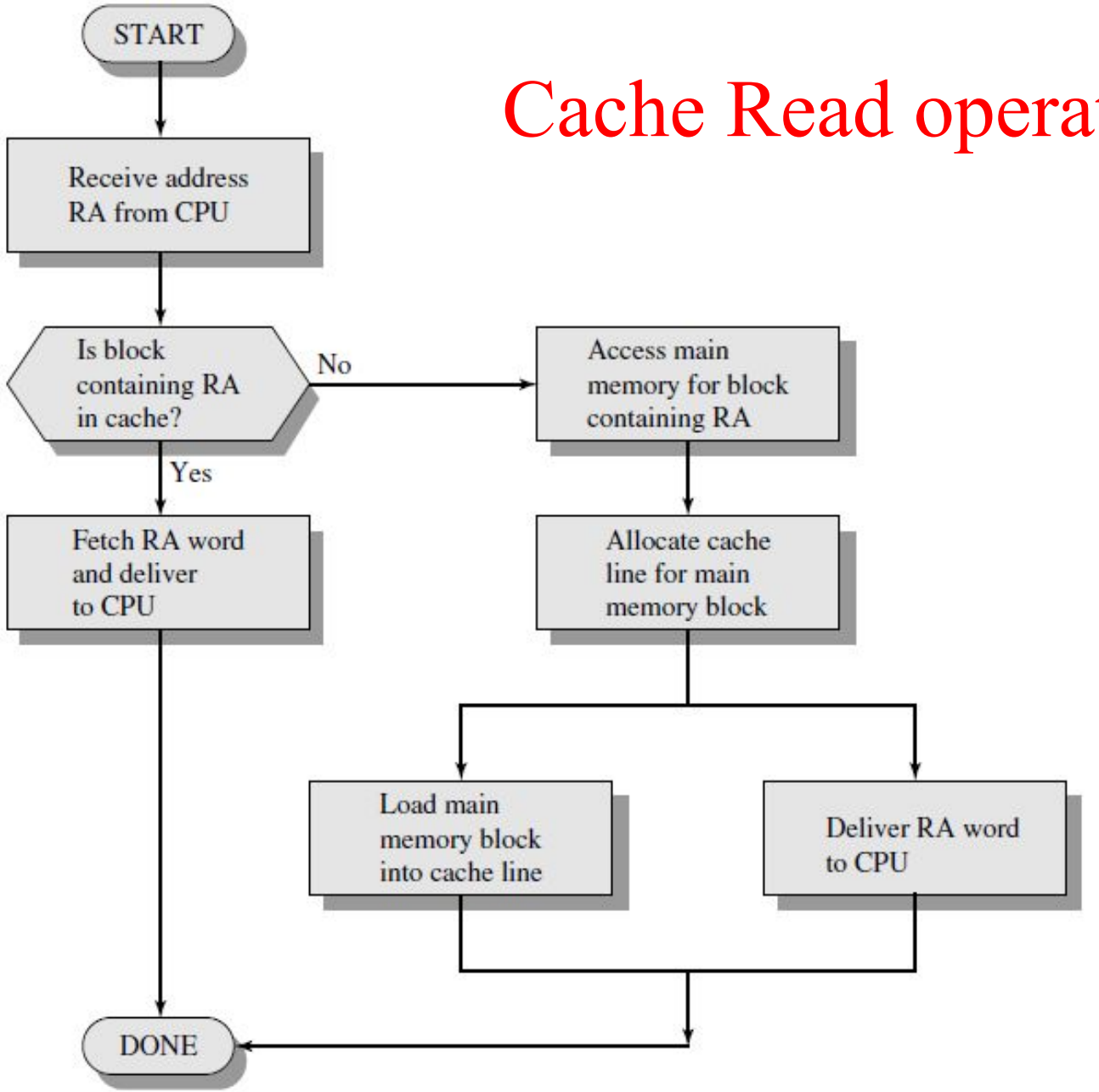
(b) Main memory

- The length of a line, not including tag and control bits, is the **line size**.
  - Line size may be as less as 32 bits (a word).

- The number of lines is considerably less than the number of main memory blocks ($m << M$).
  - At any time, some subset of the blocks of memory resides in lines in the cache.

- Because there are more blocks than lines, an individual line cannot be uniquely and permanently dedicated to a particular block. Thus, each line includes a **tag** that identifies which particular block is currently being stored.
  - The tag is usually a portion of the main memory address

Cache Read operation



Mapping Functions

- Cache has fewer lines than main memory blocks. We need
  - An algorithm for mapping main memory blocks into cache lines.
  - A method for determining which main memory block currently occupies a cache line (the tags).
  - The choice of the mapping function dictates how the cache is organized. Three techniques can be used:

1. **Direct Mapping**
2. **Associative Mapping**
3. **Set associative Mapping**

1. Direct Mapping

- The simplest technique, known as **direct** mapping, maps each block of main memory into only one possible cache line.
- The mapping is expressed as:

  i* = *j* modulo *m

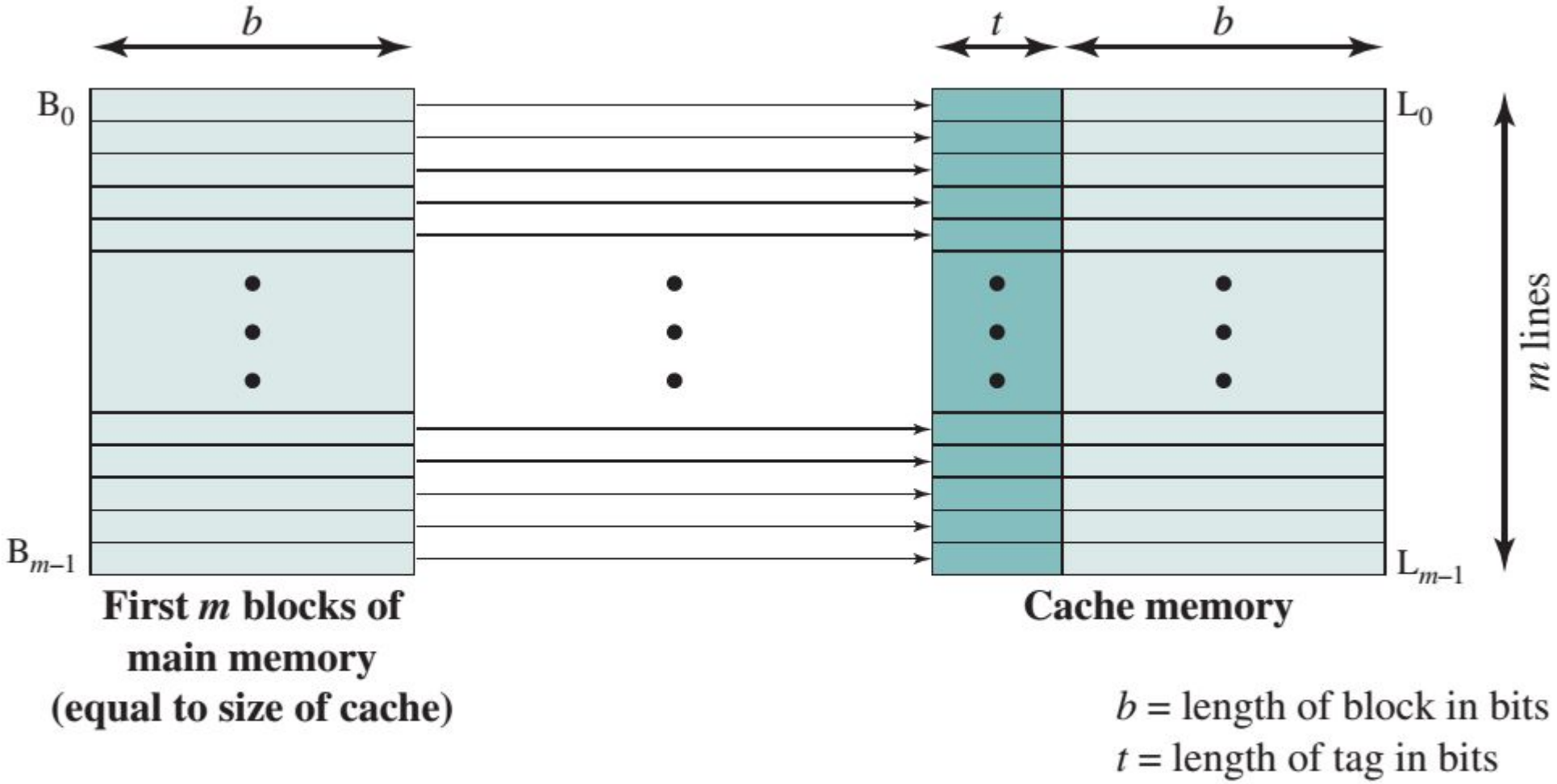
  - *Where*

    i = cache line number

    j = main memory block number

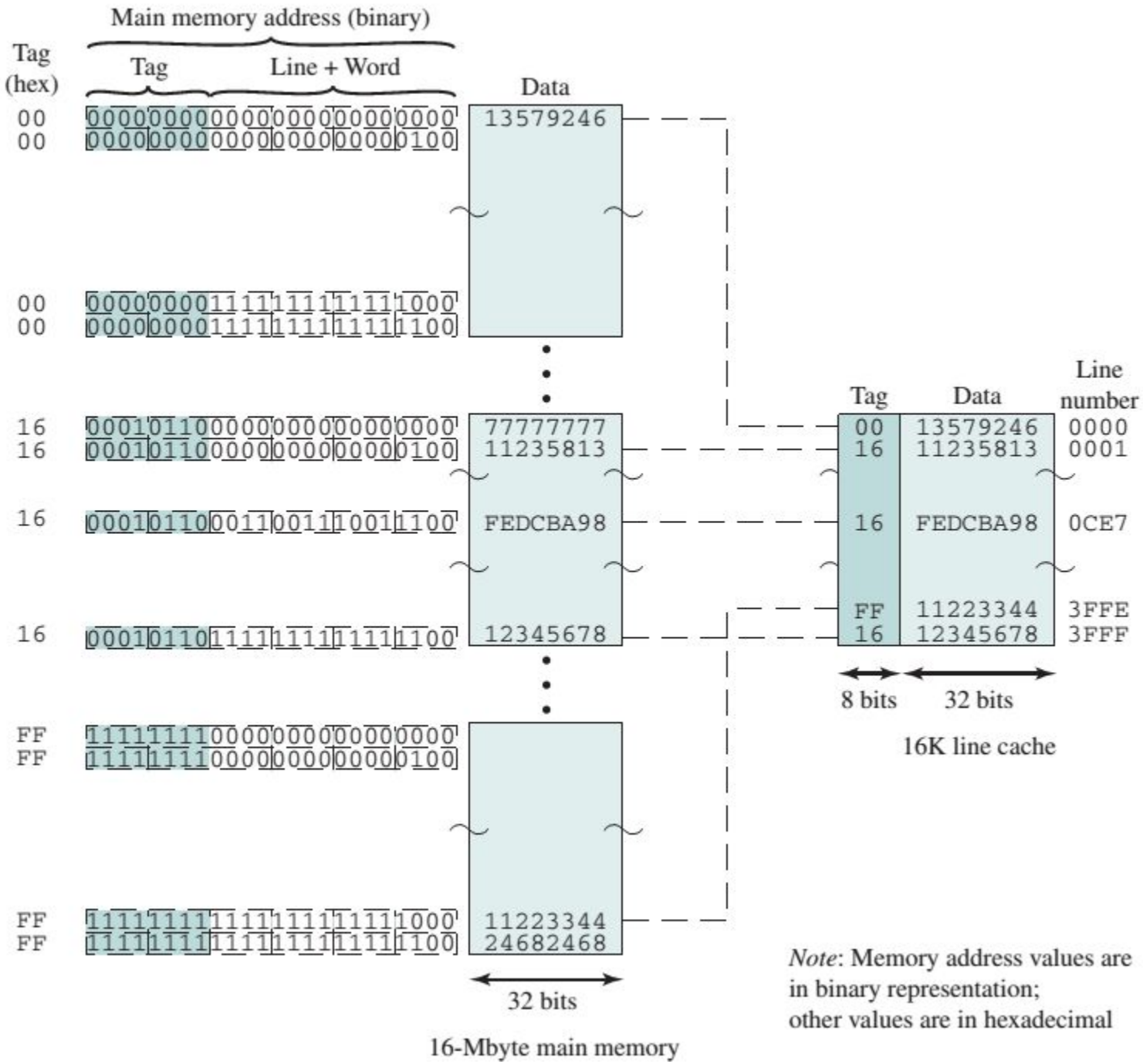
    m = number of lines in the cache

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



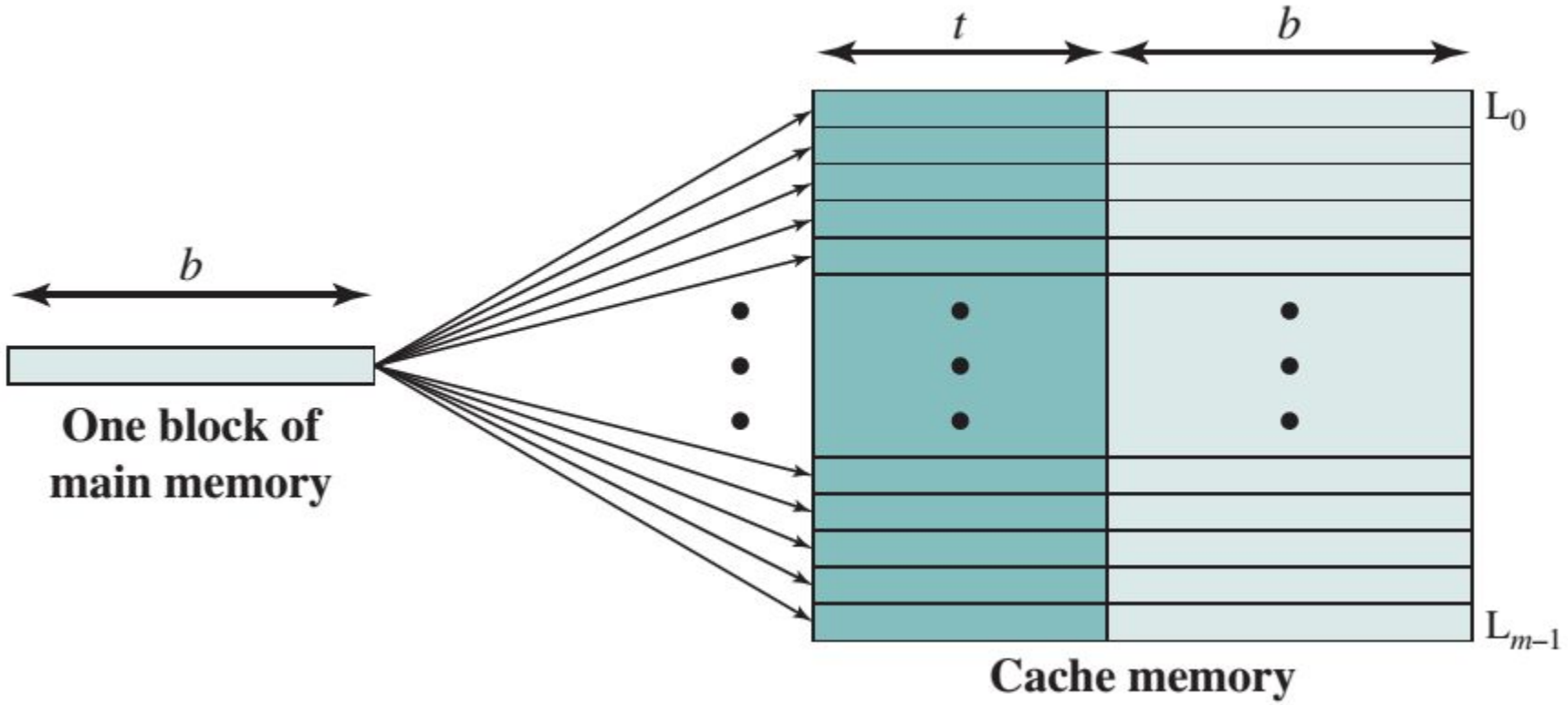
(a) Direct mapping

- The next blocks of main memory map into the cache in the same fashion; that is, block B*m* of main memory maps into line L0 of cache, block B $_{m+1}$ maps into line L1, and so on.


Main memory address (binary)
Tag
(hex)
Tag
Line + Word
Data
00
00
13579246
00
00
16
16
77777777
11235813
16
FEDCBA98
16
12345678
FF
FF
FF
FF
11223344
24682468
32 bits
16-Mbyte main memory
Tag
Data
Line
number
00
16
13579246
11235813
0000
0001
16
FEDCBA98
0CE7
FF
16
11223344
12345678
3FFE
3FFF
8 bits
32 bits
16K line cache
Note: Memory address values are
in binary representation;
other values are in hexadecimal

2. Associative Mapping

It maps each main memory block into any line of the cache



Tag is used to identify a specific block of cache

3. Set Associative Mapping

- In this case, the cache consists of a number sets, each of which consists of a number of lines, The relationships are:

$$m = v \times k$$

$$i = j \text{ modulo } v$$

- Where

  i = cache set number

  j = main memory block number

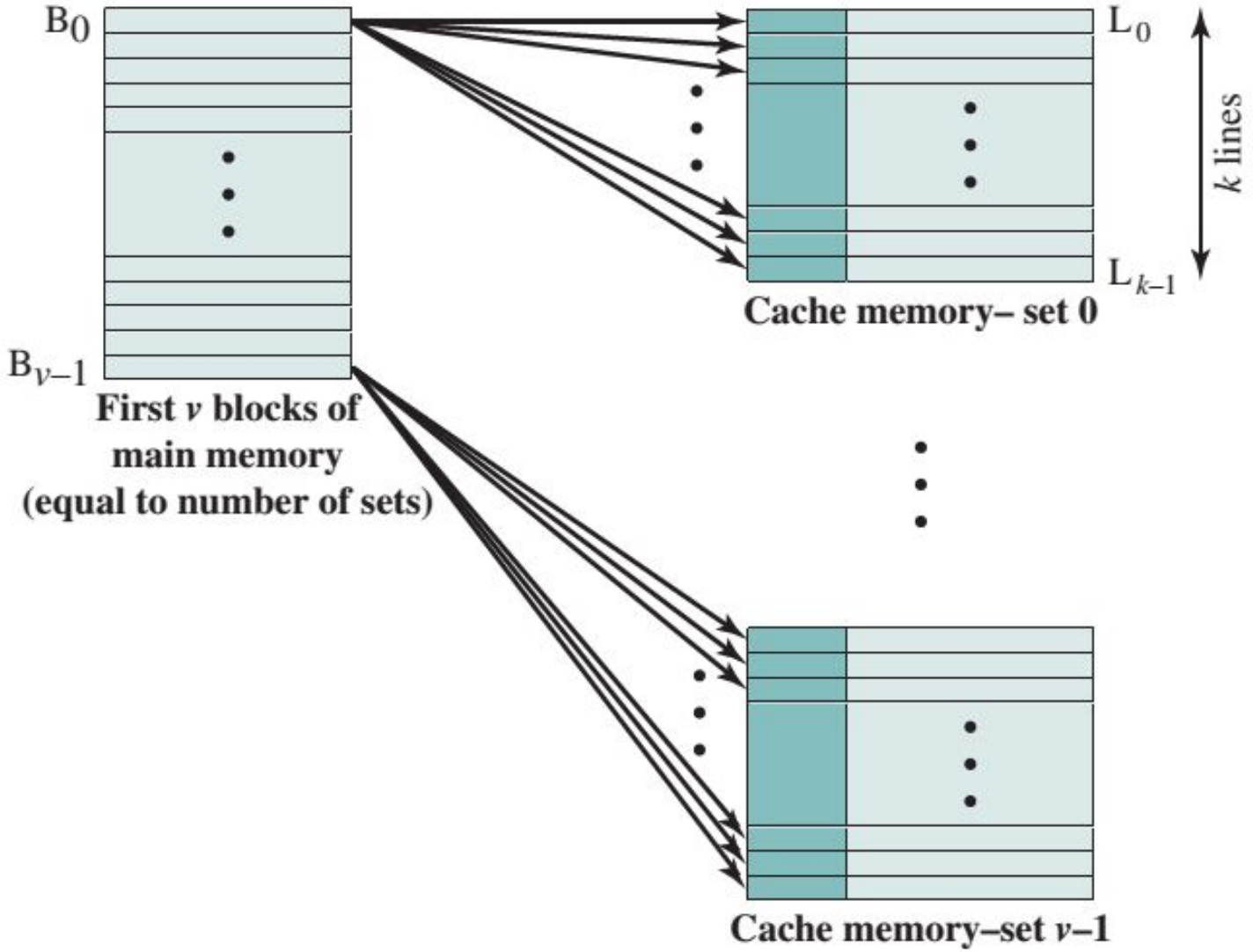
  m = number of lines in the cache

  v = number of sets

  k = number of lines in each set

3. Set Associative Mapping

- This is referred to as *k*-way set-associative mapping.
- With set-associative mapping, block B_j can be mapped into any of the lines of set *j*.
- For set-associative mapping, each word maps into all the cache lines in a specific set, so that main memory block B0 maps into set 0, and so on.


B0
Bv–1
First v blocks of
main memory
(equal to number of sets)
L0
Lk–1
k lines
Cache memory– set 0
Cache memory–set v–1

- Memory is accessed as words; where **word** is smallest accessible memory.
- **Blocks** may take multiple words, memory is accessed as blocks.
- Size of a **line** is equal to that of a block:
  - e.g. On a given system memory size is 64 words (**2^6 : 6 bits long address)**
  - Each block consists of 4 words (**2^2** : 2 bits long addresses); there are 16 blocks on the memory.
  - a cache of sized **4-lines** can contain **4** blocks of memory (**2** bits long line numbers) it can take 4 blocks; 16 words may be there in the given cache.
  - Because there are 4 words in each block the block offsets will be 2 bits long.

- CPU generates the **address** of required memory word, for our example it will be a 6-bit long address.

- Taking that address the word is first found in cache, if not found(**cache miss**), then main memory is searched for the word.

- **Address Seen by Memory**: The address is divided into two parts: the least two bits will be **block offset,** other 4-bits will be **block number.**
  - E.g. 001011 (0010: block number, 11: block offset).

- Direct mapping maps the blocks in their respective line numbers, or

  line number (i) = block number(j) % total lines in cache (m)

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