

Computer Organization and Architecture

William Stallings

8th Edition

Chapter 4

Cache Memory

Memory subsystem

- Typical computer system is equipped with a hierarchy of memory subsystems, some internal to the system (directly accessible by the processor) and some external (accessible by the processor via an I/O module).

Characteristics

- Location
- Capacity
- Unit of transfer
- Access method
- Performance
- Physical type
- Physical characteristics
- Organisation

Table 4.1 Key Characteristics of Computer Memory Systems

Location	Performance
Internal (e.g. processor registers, main memory, cache)	Access time
External (e.g. optical disks, magnetic disks, tapes)	Cycle time
	Transfer rate
Capacity	Physical Type
Number of words	Semiconductor
Number of bytes	Magnetic
	Optical
Unit of Transfer	Magneto-optical
Word	Physical Characteristics
Block	Volatile/nonvolatile
	Erasable/nonerasable
Access Method	Organization
Sequential	Memory modules
Direct	
Random	
Associative	

Location

- Memory is internal and external to the computer.
- **Internal memory:**
 1. Internal memory is often equated with main memory.
 2. The processor requires its own local memory, in the form of registers.
 3. Cache is another form of internal memory.
- **External memory**
 - External memory consists of peripheral storage devices, such as disk and tape, that are accessible to the processor via I/O controllers.

Capacity

- Word size
 - The natural unit of organisation.
- Number of Bytes
- For **internal memory**, this is typically expressed in terms of bytes (1 byte 8 bits) or words. Common word lengths are 8, 16, and 32 bits.
- **External memory** capacity is typically expressed in terms of bytes.

Unit of Transfer

- For internal memory, the unit of transfer is equal to the number of electrical lines into and out of the memory module.
- This may be equal to the word length, but is often larger, such as 64, 128, or 256 bits.
- There is a three related concepts for internal memory:

1. Word

- It is the “natural” unit of organization of memory.
- The size of the word is typically equal to the number of bits used to represent an integer and to the instruction length.

2. Addressable units

- In some systems, the addressable unit is the word.
- The relationship between the length in bits A of an address and the number N of addressable units is:

$$2^A = N$$

3. Unit of transfer

- For main memory, this is the number of bits read out of or written into memory at a time.
- For external memory, data are often transferred in much larger units than a word, and these are referred to as blocks.

Unit of Transfer

- Internal
 - Usually governed by data bus width.
- External
 - Usually a block which is much larger than a word.
- Addressable unit
 - Smallest location which can be uniquely addressed.
 - Word internally.

Method of Accessing

1. Sequential access

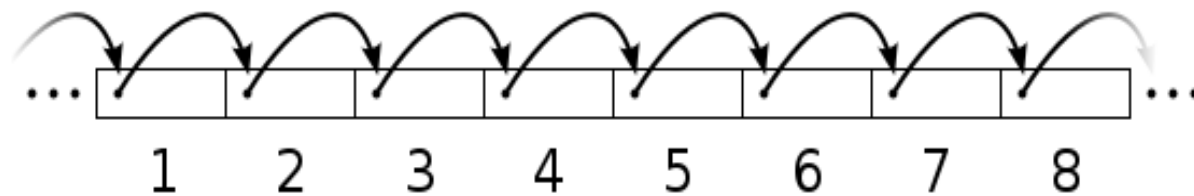
- Memory is organized into units of data, called records.
- Access must be made in a specific linear sequence.
- Start at the beginning and read through in order.
- Stored addressing information is used to separate records and assist in the retrieval process.
- A shared read–write mechanism is used, and this must be moved from its current location to the desired location, passing and rejecting each intermediate record.
- Access time depends on location of data and previous location.
- e.g. Tape.

Method of Accessing

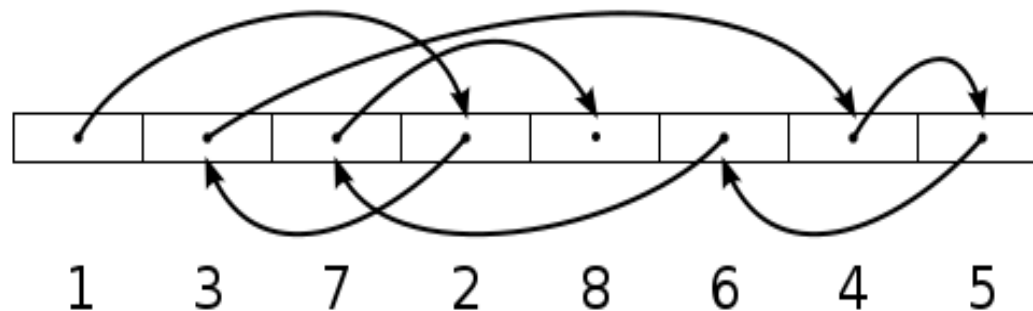
2. Direct access

- Direct access involves a shared read–write mechanism.
- Individual blocks or records have a unique address based on physical location.
- Access is by jumping to vicinity plus sequential search.
- Access time depends on location and previous location.
- e.g. Disk.

Sequential access



Random access



Method of Accessing

3. Random access

- Individual addresses identify locations exactly.
- Access time is independent of location or previous access.
- The time to access a given location is independent of the sequence of prior accesses and is constant.
- Any location can be selected at random and directly addressed and accessed.
- Main memory and some cache systems are random access.
- e.g. RAM.

Method of Accessing

4. Associative

- This is a random access type of memory that enables one to make a comparison of desired bit locations within a word for a specified match, and to do this for all words simultaneously.
- A word is retrieved based on a portion of its contents rather than its address.
- Data is located by a comparison with contents of a portion of the store.
- Access time is independent of location or previous access.
- e.g. Cache.

Performance

- The two most important characteristics of memory are capacity and performance.
- Three performance parameters are used:
 - Access time (latency).
 - Memory cycle time.
 - Transfer rate.

Performance

- Access time
 - Time between presenting the address and getting the valid data.
- Memory Cycle time
 - Time may be required for the memory to “recover” before next access.
 - Cycle time is access + recovery.
- Transfer Rate
 - Rate at which data can be moved.

1. Access time (latency)

▫ For random-access memory:

- It is the time it takes to perform a read or write operation.
- Also, it is the time from the instant that an address is presented to the memory to the instant that data have been stored or made available for use.

▫ For non-random-access memory:

- Access time is the time it takes to position the read–write mechanism at the desired location.

2. Memory cycle time

- It is applied to random-access memory and consists of the access time plus any additional time required before a second access can commence.
- Note that memory cycle time is concerned with the system bus, not the processor.

3. Transfer rate

- It is the rate at which data can be transferred into or out of a memory unit.
- For random-access memory, it is equal to $1/(\text{cycle time})$.
- For non-random-access memory, the following relationship holds:

$$T_N = T_A + n/R$$

where

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)

Physical Types

- The most common today are semiconductor memory, magnetic surface memory (used for disk and tape), and optical and magneto-optical.

Physical Types

- Semiconductor
 - RAM
- Magnetic
 - Disk & Tape
- Optical
 - CD & DVD
- Others
 - Bubble
 - Hologram

Physical Characteristics

- Decay
- Volatility
- Erasable
- Power consumption

Physical Characteristics

- In a volatile memory
 - Information decays naturally or is lost when electrical power is switched off.
- In a nonvolatile memory
 - Information once recorded remains without deterioration until deliberately changed; no electrical power is needed to retain information.
- Magnetic-surface memories are nonvolatile.
- Semiconductor memory may be either volatile or nonvolatile.

Physical Characteristics

- Nonerasable memory
 - It cannot be altered, except by destroying the storage unit.
- Semiconductor memory of this type is known as read-only memory (ROM).
- A practical nonerasable memory must also be nonvolatile.

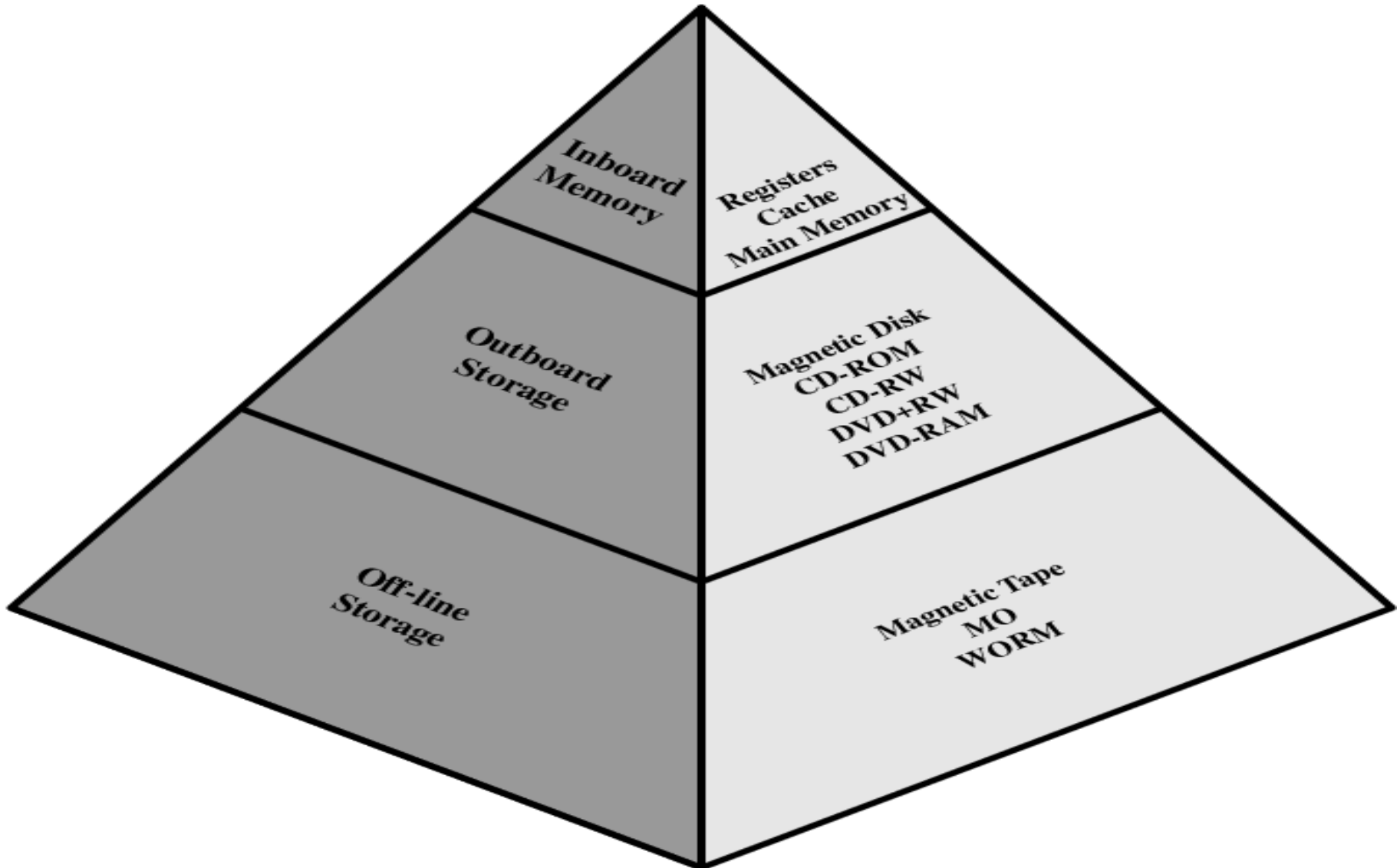
Organisation

- Physical arrangement of bits into words.
- Not always obvious.
- e.g. Interleaved.

Memory Hierarchy

- Registers
 - In CPU.
- Internal or Main memory
 - May include one or more levels of cache.
 - “RAM”.
- External memory
 - Backing store.

Memory Hierarchy - Diagram



The Memory Hierachy

- The design constraints on a computer's memory can be summed up by three questions:
 - How much? How fast? How expensive?
 - How much?
 - **Capacity.**
 - How fast?
 - **Time is money.**
 - How expensive?
 - **Cost.**
- To achieve greatest performance, the memory must be able to keep up with the processor.

- A variety of technologies are used to implement memory systems, and across this spectrum of technologies, the following relationships hold:
 - Faster access time, greater cost per bit.
 - Greater capacity, smaller cost per bit.
 - Greater capacity, slower access time.
- As one goes down the hierarchy such as in figure 4.1, the following occur:
 - a. Decreasing cost per bit.
 - b. Increasing capacity.
 - c. Increasing access time.
 - d. Decreasing frequency of access of the memory by the processor.

Hierarchy List

- Registers
- L1 Cache
- L2 Cache
- Main memory
- Disk cache
- Disk
- Optical
- Tape

So you want fast?

- It is possible to build a computer which uses only static RAM (see later).
- This would be very fast.
- This would need no cache.
 - How can you cache cache?
- This would cost a very large amount.

- Thus, smaller, more expensive, faster memories are supplemented by larger, cheaper, slower memories. The key to the success of this organization is item (d): decreasing frequency of access.
- If the accessed word is found in the faster memory, that is defined as a **hit**.
- A **miss** occurs if the accessed word is not found in the faster memory.

- **Locality of Reference**

- During the course of the execution of a program, memory references tend to cluster.
- e.g. Loops.

- The use of two levels of memory to reduce average access time works in principle, but only if conditions (a) through (d) apply.

- The fastest, smallest, and most expensive type of memory consists of the registers internal to the processor.
- Skipping down two levels, main memory is the principal internal memory system of the computer. Each location in main memory has a unique address.
- Main memory is usually extended with a higher-speed, smaller cache.
- The cache is not usually visible to the programmer or, indeed, to the processor. It is a device for staging the movement of data between main memory and processor registers to improve performance.

- The three forms of memory just described are, typically, volatile and employ semiconductor technology.
- The semiconductor memory comes in a variety of types, which differ in speed and cost.
- External, nonvolatile memory is also referred to as **secondary memory** or **auxiliary memory**. These are used to store program and data files and are usually visible to the programmer only in terms of files and records, as opposed to individual bytes or words. Disk is also used to provide an extension to main memory known as **virtual memory**.

- Data are stored more permanently on external mass storage devices, of which the most common are hard disk and removable media, such as removable magnetic disk, tape, and optical storage.
- Other forms of secondary memory include optical and magneto-optical disks.
- A portion of main memory can be used as a buffer to hold data temporarily that is to be read out to disk. Such a technique, sometimes referred to as a **disk cache**.

- To **improves performance** in two ways:

1. Disk writes are clustered. Instead of many small transfers of data, we have a few large transfers of data. This improves disk performance and minimizes processor involvement.
2. Some data destined for write-out may be referenced by a program before the next dump to disk. In that case, the data are retrieved rapidly from the software cache rather than slowly from the disk.

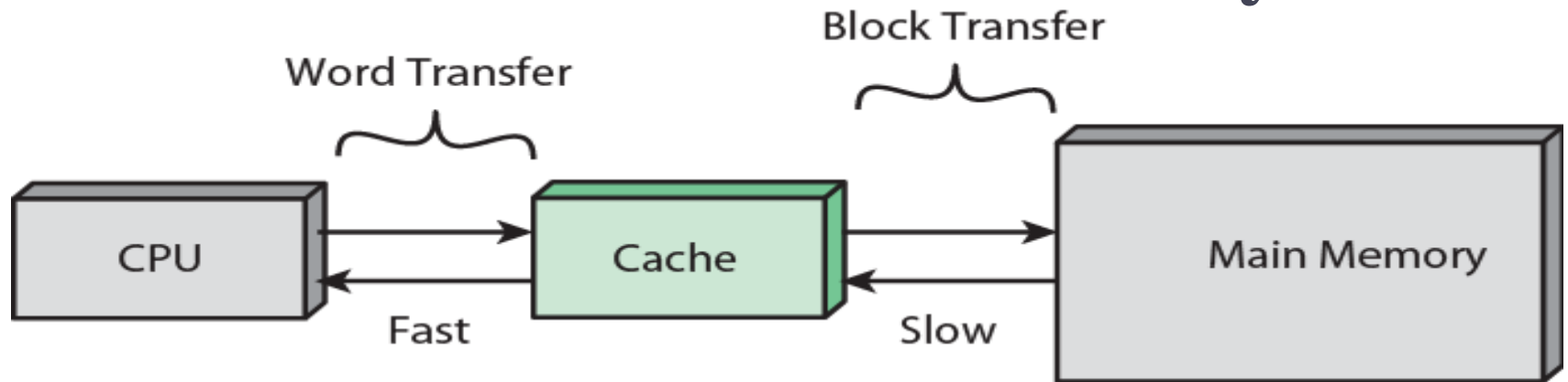
CACHE MEMORY PRINCIPLES

- Small amount of fast memory.
- Sits between normal main memory and CPU.
- May be located on CPU chip or module.

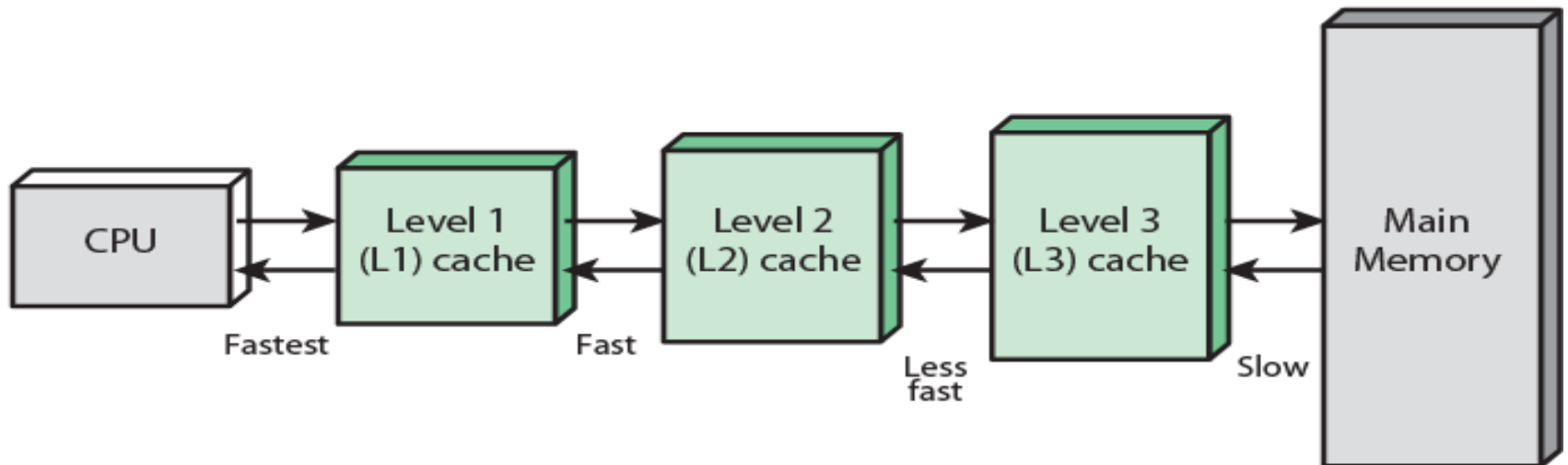
- Cache memory is intended to give memory speed approaching that of the fastest memories available, and at the same time provide a large memory size at the price of less expensive types of semiconductor memories.
- In next figure, there is a relatively large and slow main memory together with a smaller, faster cache memory.
- The cache contains a copy of portions of main memory.

- When the processor attempts to read a word of memory, a check is made to determine if the word is in the cache.
- If so, the word is delivered to the processor.
- If not, a block of main memory, consisting of some fixed number of words, is read into the cache and then the word is delivered to the processor.
- There is multiple levels of cache. The L2 cache is slower and typically larger than the L1 cache, and the L3 cache is slower and typically larger than the L2 cache.

Cache and Main Memory

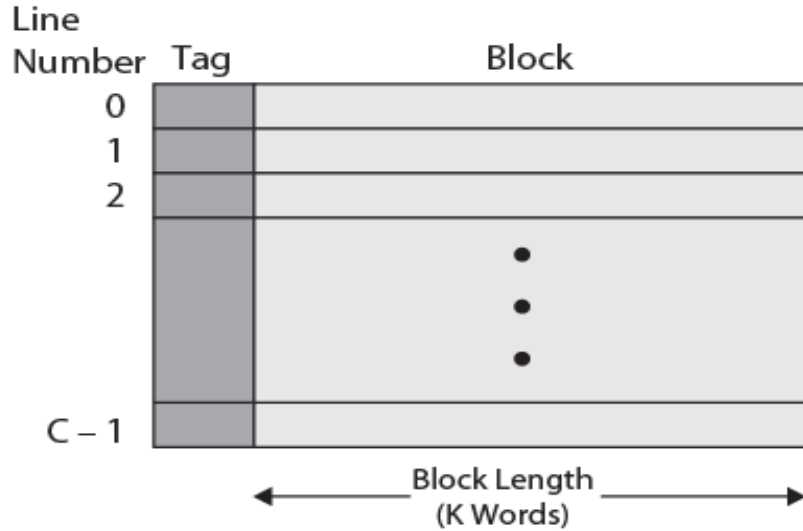


(a) Single cache

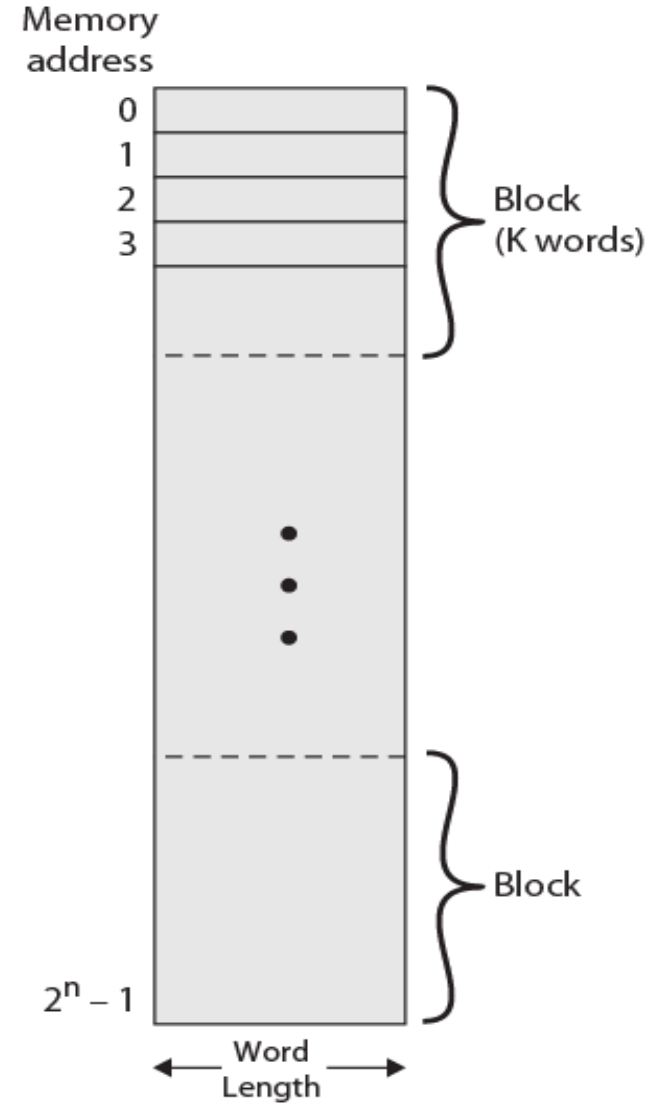


(b) Three-level cache organization

Cache/Main Memory Structure



(a) Cache



(b) Main memory

- **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.
- There are $M=2^n/K$ blocks in main memory.

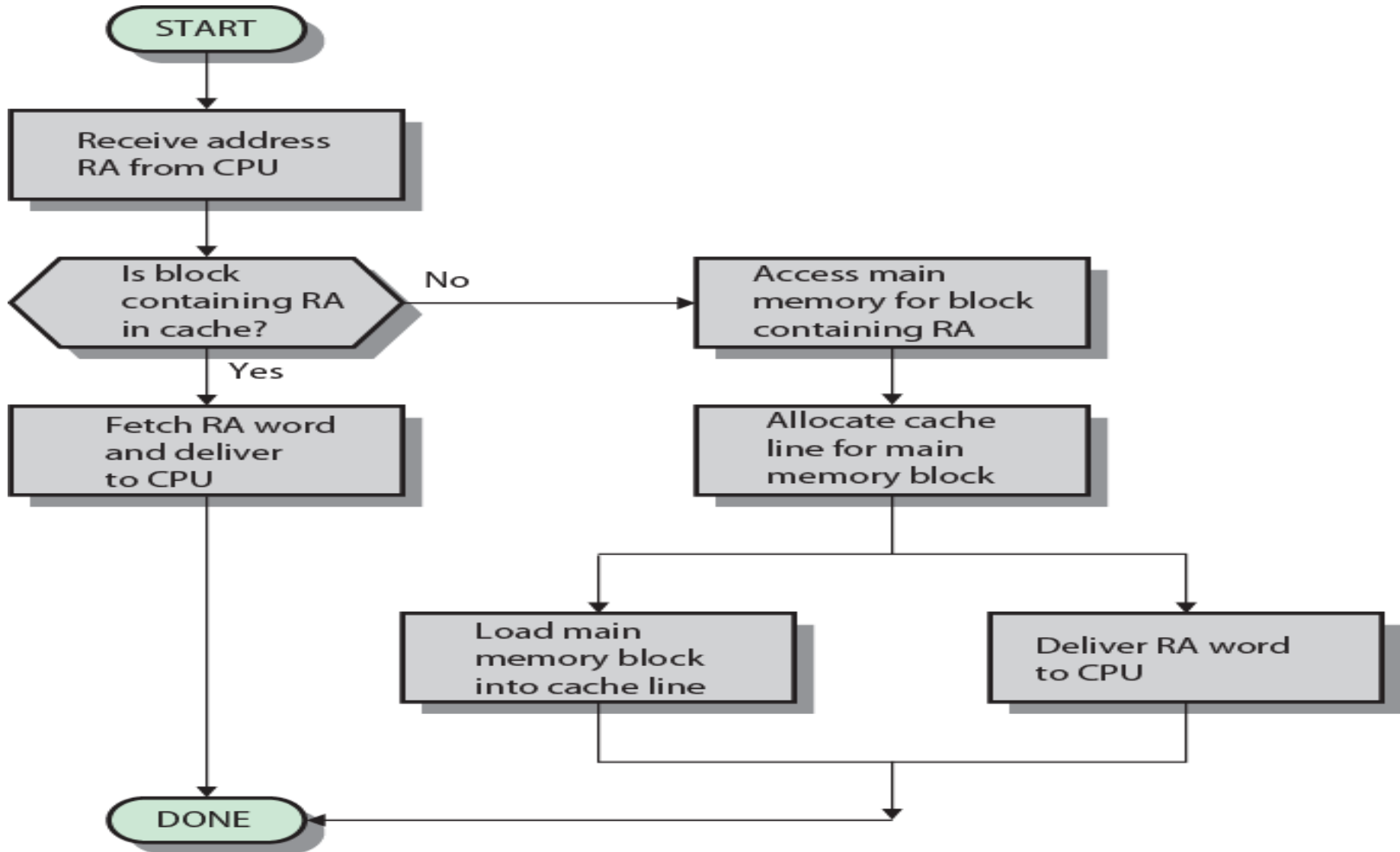
- The **cache** consists of m blocks, called **lines**.
- Each line contains K words, plus a tag of a few bits. Each line also includes control bits (not shown), such as a bit to indicate whether the line has been modified since being loaded into the cache.
- The length of a line, not including tag and control bits, is the **line size**.
- The line size may be as small as 32 bits, with each “word” being a single byte; in this case the line size is 4 bytes.

- The number of lines is considerably less than the number of main memory blocks ($m \ll M$).
- If a word in a block of memory is read, that block is transferred to one of the lines of 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.

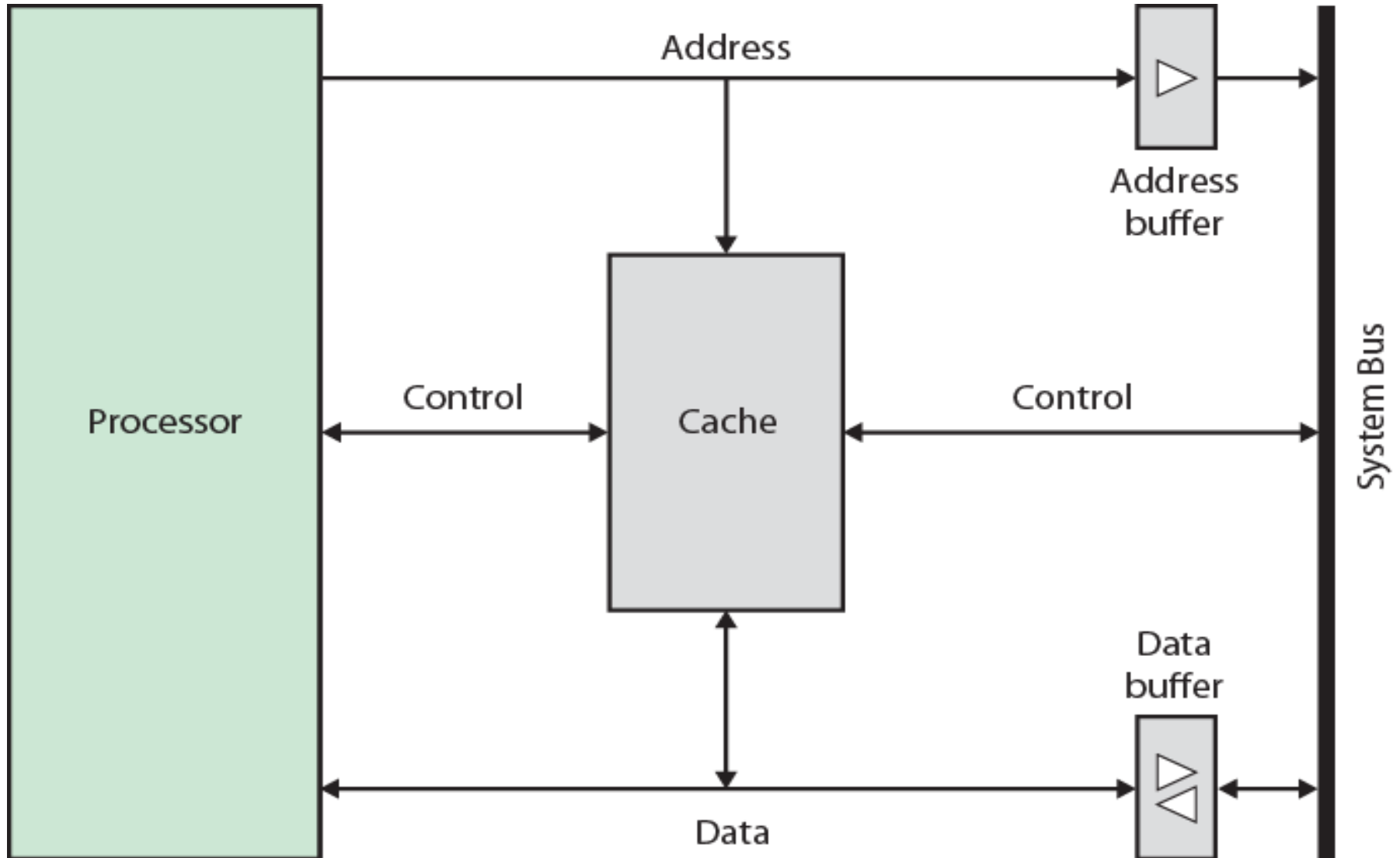
Cache operation – overview

- CPU requests contents of memory location.
- Check cache for this data.
- If present, get from cache (fast).
- If not present, read required block from main memory to cache.
- Then deliver from cache to CPU.
- Cache includes tags to identify which block of main memory is in each cache slot.
- **RA** is referred to the read address of a word to be read.

Cache Read Operation - Flowchart



Typical Cache Organization



- The processor generates the read address (**RA**) of a word to be read.
- If the word is contained in the cache, it is delivered to the processor.
- Otherwise, the block containing that word is loaded into the cache, and the word is delivered to the processor.
- The cache connects to the processor via data, control, and address lines. The data and address lines also attach to data and address buffers, which attach to a system bus from which main memory is reached.
- When a cache hit occurs, the data and address buffers are disabled and communication is only between processor and cache, with no system bus traffic.

- When a cache miss occurs, the desired address is loaded onto the system bus and the data are returned through the data buffer to both the cache and the processor.
- For a cache miss, the desired word is first read into the cache and then transferred from cache to processor.

Table 4.2 Elements of Cache Design

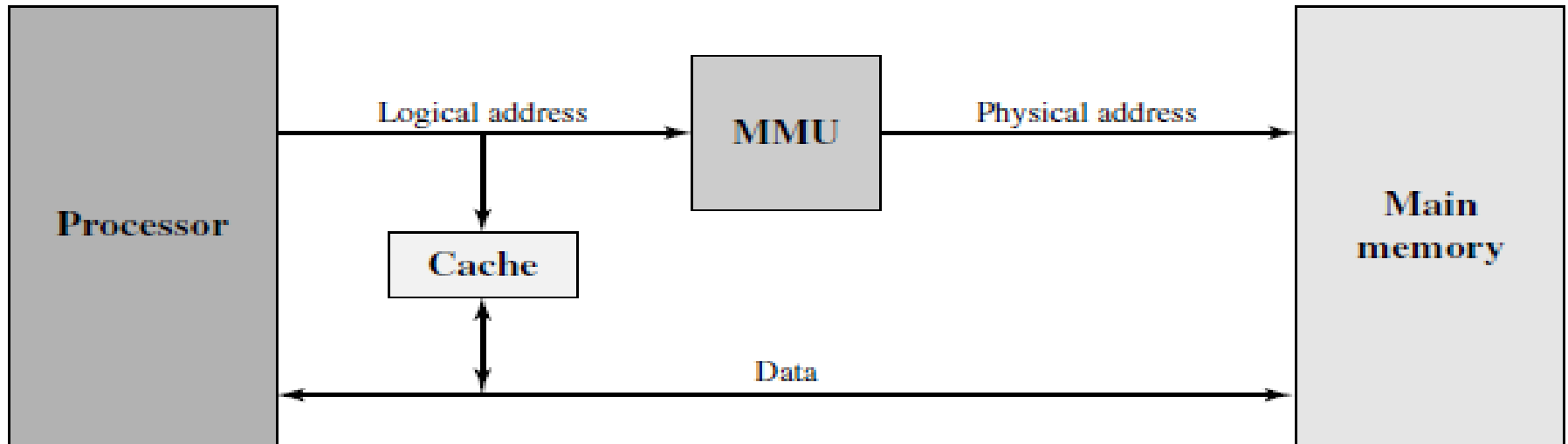
Cache Addresses	Write Policy
Logical	Write through
Physical	Write back
Cache Size	Write once
Mapping Function	Line Size
Direct	Number of caches
Associative	Single or two level
Set Associative	Unified or split
Replacement Algorithm	
Least recently used (LRU)	
First in first out (FIFO)	
Least frequently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

Elements of Cache Design

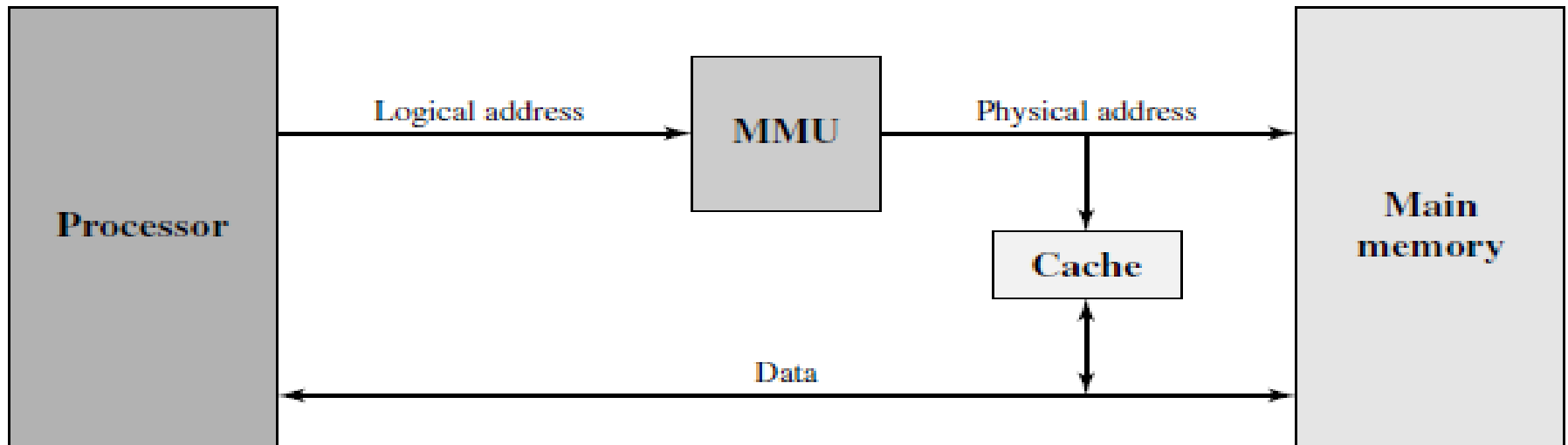
- Addressing
- Size
- Mapping Function
- Replacement Algorithm
- Write Policy
- Block Size
- Number of Caches

Cache Addresses

- Virtual memory is a facility that allows programs to address memory from a logical point of view, without regard to the amount of main memory physically available.
- When virtual memory is 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.
- 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.



(a) Logical cache



(b) Physical cache

Figure 4.7 Logical and Physical Caches

- 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.

Advantage/ disadvantage of logical cache (Virtual cache)

- **Advantage:**

- The logical cache is that cache access speed is faster than for a physical cache, because the cache can respond before the MMU performs an address translation.

Advantage/ disadvantage of logical cache (Virtual cache)

- **Disadvantage:**

- Most virtual memory systems supply each application with the same virtual memory address space.
- That is, each application sees a virtual memory that starts at address 0.
- Thus, the same virtual address in two different applications refers to two different physical addresses.
- 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.

Cache Size

- We would like 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, and large enough so that the overall average access time is close to that of the cache alone.
- There are several other motivations for minimizing cache size:
 - The larger the cache, the larger the number of gates involved in addressing the cache. The result is that large caches tend to be slightly slower than small ones even when built with the same integrated circuit technology and put in the same place on chip and circuit board.

Cache Size

- There are several other motivations for minimizing cache size:
 - 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.

Comparison of Cache Sizes

Processor	Type	Year of Introduction	L1 cache	L2 cache	L3 cache
IBM 360/85	Mainframe	1968	16 to 32 KB	—	—
PDP-11/70	Minicomputer	1975	1 KB	—	—
VAX 11/780	Minicomputer	1978	16 KB	—	—
IBM 3033	Mainframe	1978	64 KB	—	—
IBM 3090	Mainframe	1985	128 to 256 KB	—	—
Intel 80486	PC	1989	8 KB	—	—
Pentium	PC	1993	8 KB/8 KB	256 to 512 KB	—
PowerPC 601	PC	1993	32 KB	—	—
PowerPC 620	PC	1996	32 KB/32 KB	—	—
PowerPC G4	PC/server	1999	32 KB/32 KB	256 KB to 1 MB	2 MB
IBM S/390 G4	Mainframe	1997	32 KB	256 KB	2 MB
IBM S/390 G6	Mainframe	1999	256 KB	8 MB	—
Pentium 4	PC/server	2000	8 KB/8 KB	256 KB	—
IBM SP	High-end server/ supercomputer	2000	64 KB/32 KB	8 MB	—
CRAY MTA ^b	Supercomputer	2000	8 KB	2 MB	—
Itanium	PC/server	2001	16 KB/16 KB	96 KB	4 MB
SGI Origin 2001	High-end server	2001	32 KB/32 KB	4 MB	—
Itanium 2	PC/server	2002	32 KB	256 KB	6 MB
IBM POWER5	High-end server	2003	64 KB	1.9 MB	36 MB
CRAY XD-1	Supercomputer	2004	64 KB/64 KB	1MB	—

Mapping Function

- Because there are fewer cache lines than main memory blocks, an algorithm is needed for mapping main memory blocks into cache lines.
- Further, a means is needed for determining which main memory block currently occupies a cache line.
- The choice of the mapping function dictates how the cache is organized. Three techniques can be used: **direct, associative, and set associative.**

Mapping Function

- **Example 4.2** For all three cases, the example includes the following elements:
- The cache can hold 64 KBytes.
- Data are transferred between main memory and the cache in blocks of 4 bytes each.
- 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.

Mapping Function

- **Therefore, for example 4.2:**
- Cache of 64kByte.
- Cache block of 4 bytes.
 - i.e. cache is 16k (2^{14}) lines of 4 bytes.
- 16MBytes main memory.
- 24 bit address.
 - ($2^{24}=16M$).

Direct Mapping

- It is the simplest technique which maps each block of main memory into only one possible cache line.
- The mapping is expressed as:

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

where

i: cache line number

j: main memory block number

m: number of lines in the cache

- Each block of main memory maps into one unique line of the cache. 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 L_0 of cache, block B_{m+1} maps into line L_1 , and so on.

Direct Mapping

- The mapping function is easily implemented using the main memory address.
- Each block of main memory maps to only one cache line
 - i.e. if a block is in cache, it must be in one specific place.
- Memory address is viewed as two parts:
 - Least Significant w bits is to identify a unique word.
 - Most Significant (MSBs) s bits is to specify one memory block. The MSBs are split into a cache line field r and a tag of $s-r$ (most significant).

Memory Address Structure Direct Mapping

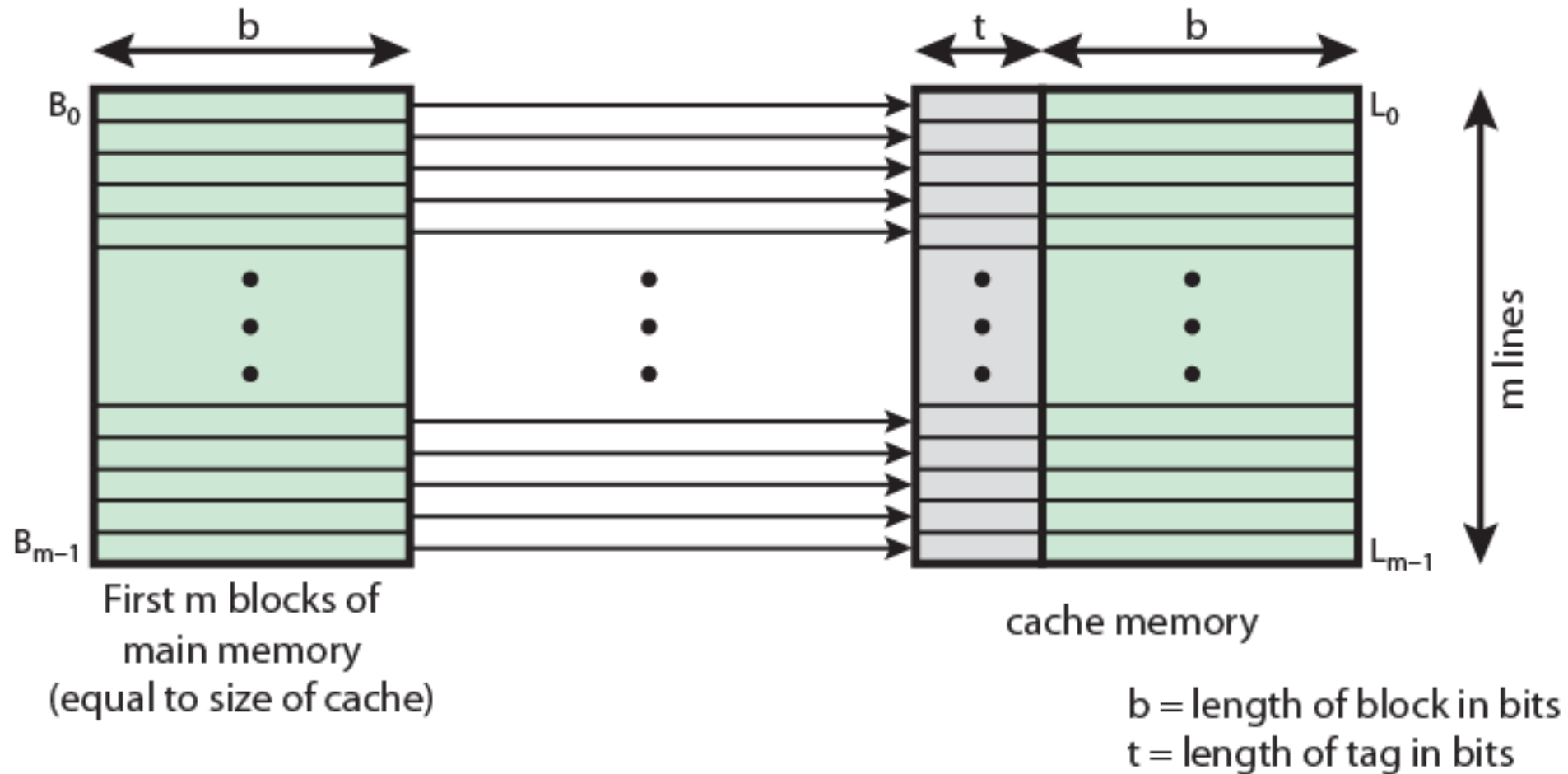
Tag $s-r$	Line or Slot r	Word w
8-bit	14-bit	2-bit

- 24 bit memory address.
- 2 bit word identifier (4 byte block).
- 22 bit block identifier.
 - 8 bit tag (=22-14).
 - 14 bit slot or line.
- No two blocks in the same line have the same Tag field.
- Check contents of cache by finding line and checking Tag.

Direct Mapping Summary

- To summarize it:
 - Address length = $(s + w)$ bits.
 - Number of addressable units = 2^{s+w} words or bytes.
 - Block size = line size = 2^w words or bytes.
 - Number of blocks in main memory = $2^{s+w} / 2^w = 2^s$
 - Number of lines in cache = $m = 2^r$
 - Size of cache = 2^{r+w} words or bytes.
 - Size of tag = $(s - r)$ bits.

Direct Mapping from Cache to Main Memory



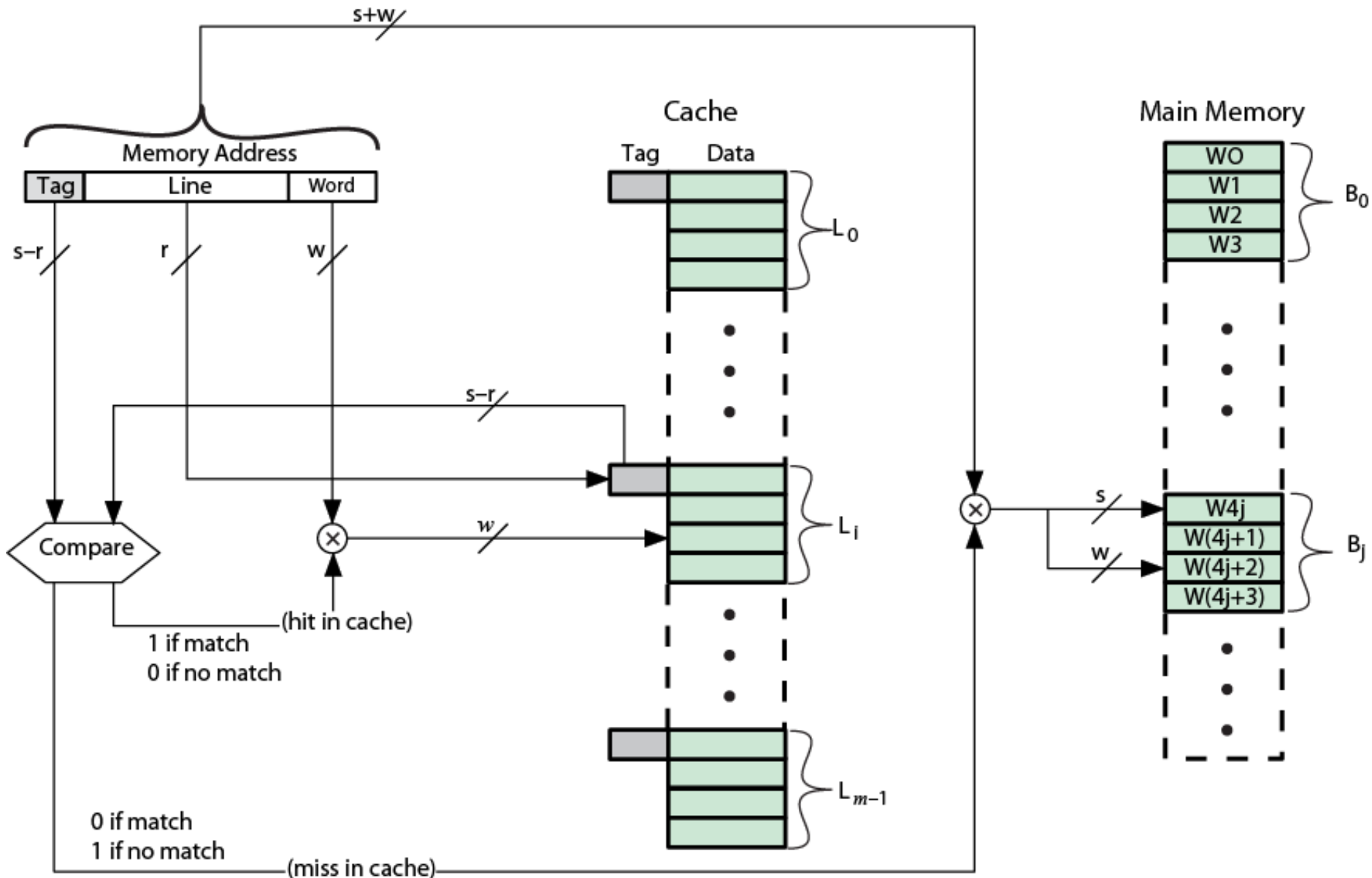
(a) Direct mapping

Direct Mapping Cache Line Table

Cache line	Main Memory blocks assigned
0	$0, m, 2m, 3m \dots 2s-m$
1	$1, m+1, 2m+1 \dots 2s-m+1$
...	
$m-1$	$m-1, 2m-1, 3m-1 \dots 2s-1$

- Thus, the use of a portion of the address as a line number provides a unique mapping of each block of main memory into the cache.
- When a block is actually read into its assigned line, it is necessary to tag the data to distinguish it from other blocks that can fit into that line. The most significant $s - r$ bits serve this purpose.

Direct Mapping Cache Organization



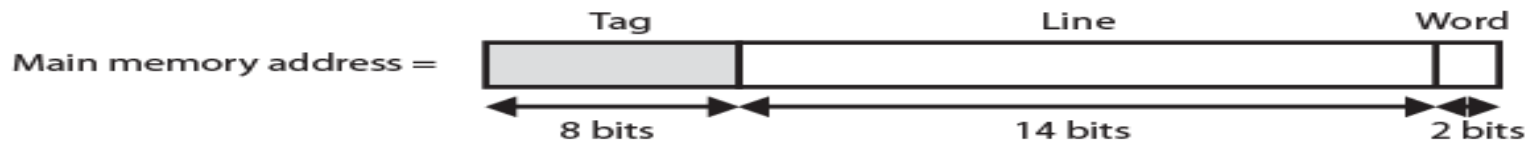
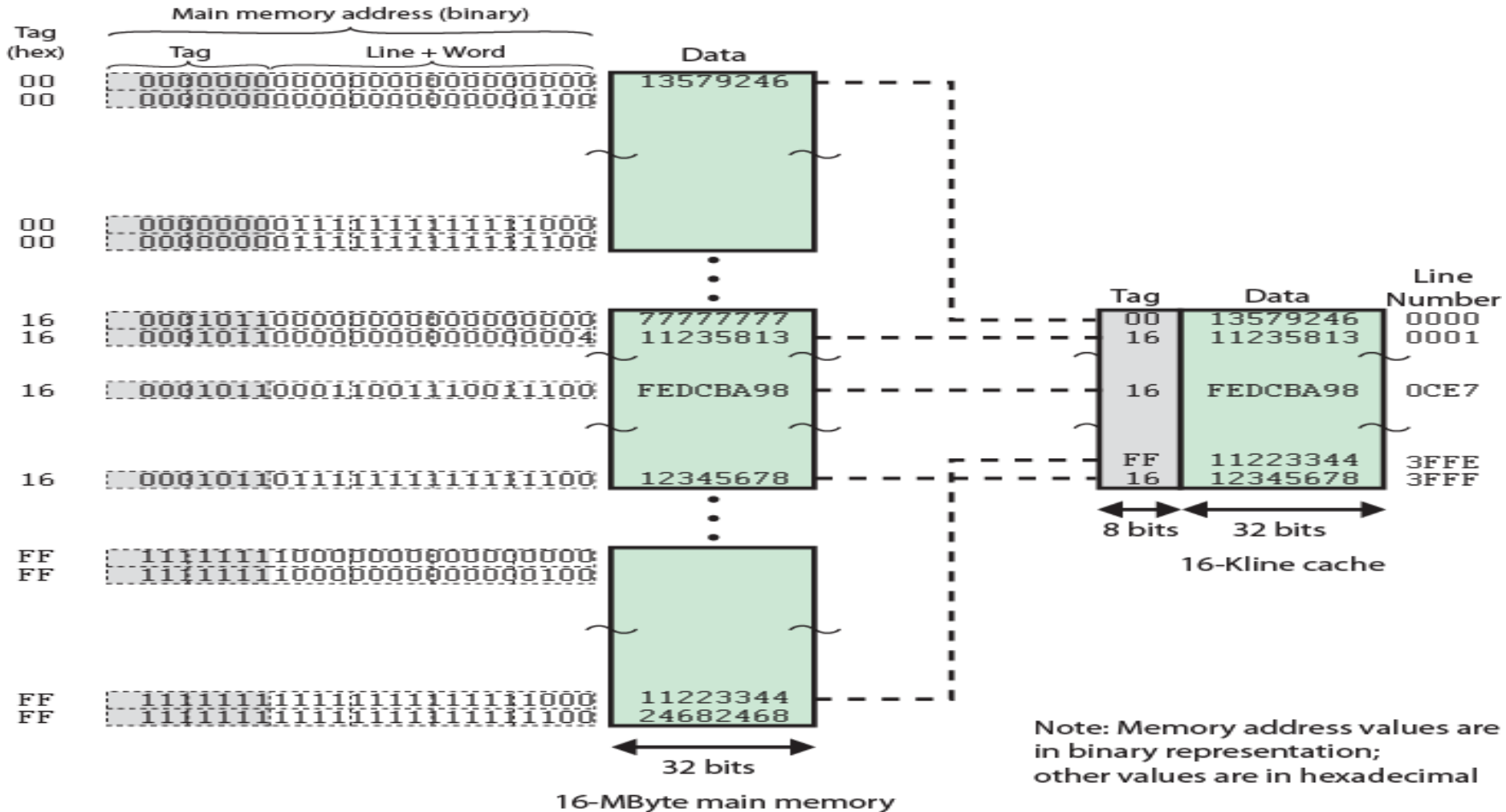
Example 4.2a Figure 4.10 shows our example system using direct mapping.⁵ In the example, $m = 16\text{K} = 2^{14}$ and $i = j$ modulo 2^{14} . The mapping becomes

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

Note that no two blocks that map into the same line number have the same tag number. Thus, blocks with starting addresses 000000, 010000, ..., FF0000 have tag numbers 00, 01, ..., FF, respectively.

Referring back to Figure 4.5, a read operation works as follows. 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.

Direct Mapping Example



Direct Mapping pros & cons

- **Advantages:**

- Simple.
- Inexpensive.

- **Disadvantage:**

- Fixed cache location for given block. If a program accesses 2 blocks that map to the same line repeatedly, cache misses are very high which is called *thrashing*.

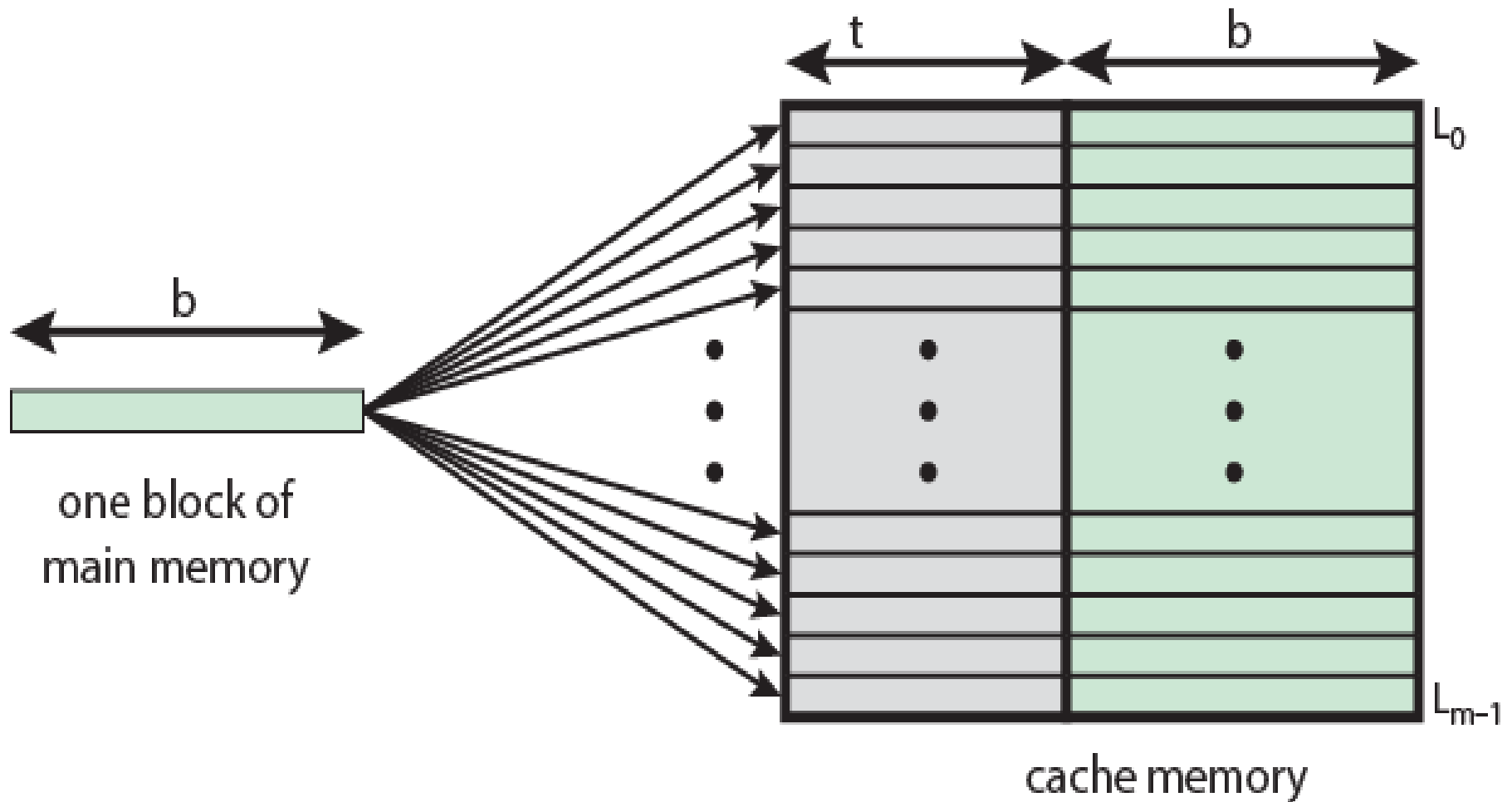
Victim Cache

- One approach to lower the miss penalty is to remember what was discarded in case it is needed again. Since the discarded data has already been fetched, it can be used again at a small cost.
- Such recycling is possible using a victim cache.
- Victim cache is an approach to reduce the conflict misses of direct mapped caches without affecting its fast access time.
- Victim cache is a fully associative cache, whose size is typically 4 to 16 cache lines, residing between a direct mapped L1 cache and the next level of memory

Associative Mapping

- A main memory block can load into any line of 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.
- Every line's tag is examined for a match.
- Cache searching gets expensive.

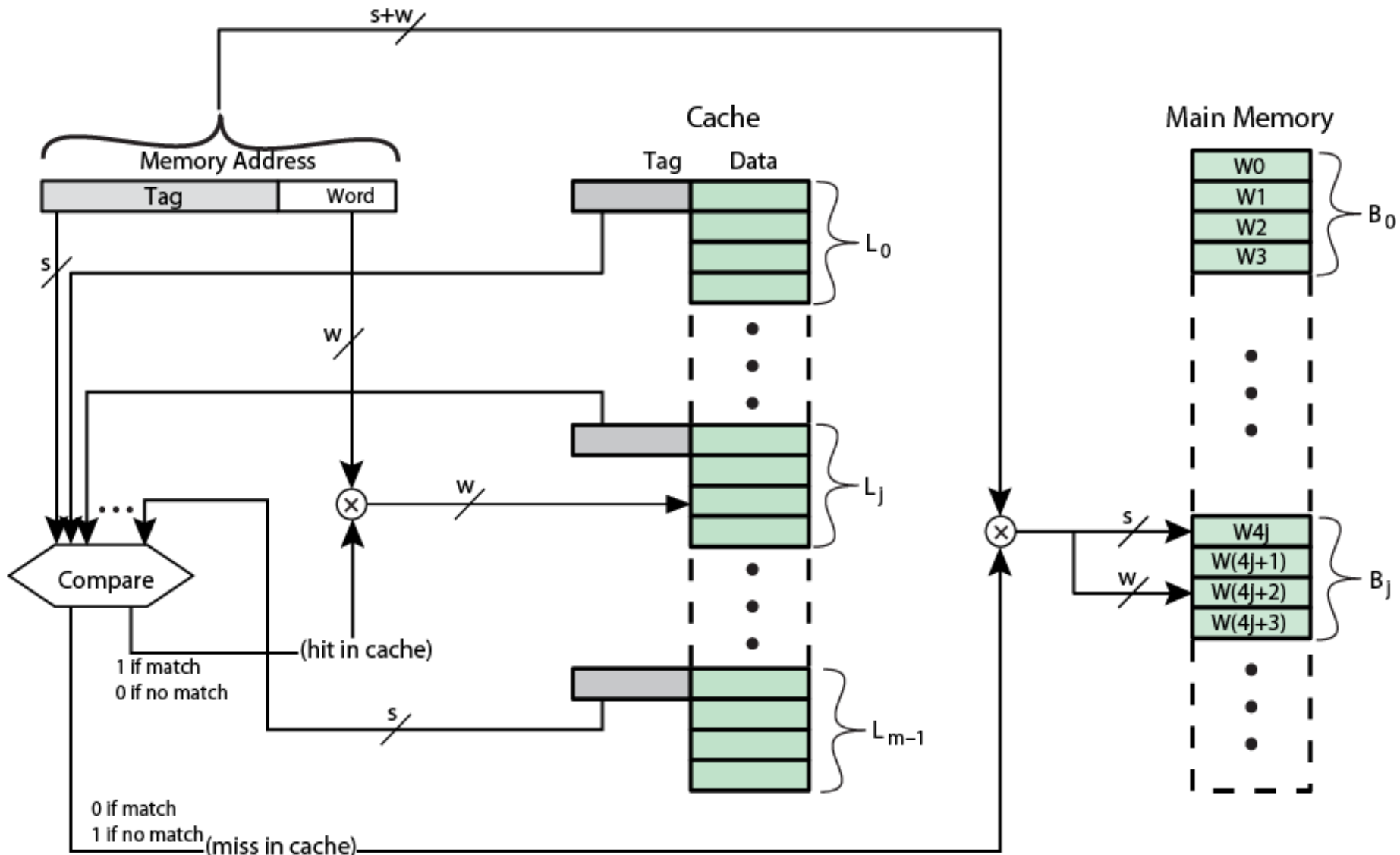
Associative Mapping from Cache to Main Memory



Associative Mapping Summary

- To summarize:
 - Address length = $(s + w)$ bits
 - Number of addressable units = 2^{s+w} words or bytes
 - Block size = line size = 2^w words or bytes
 - Number of blocks in main memory = $2^{s+w} / 2^w = 2^s$
 - Number of lines in cache = undetermined
 - Size of tag = s bits

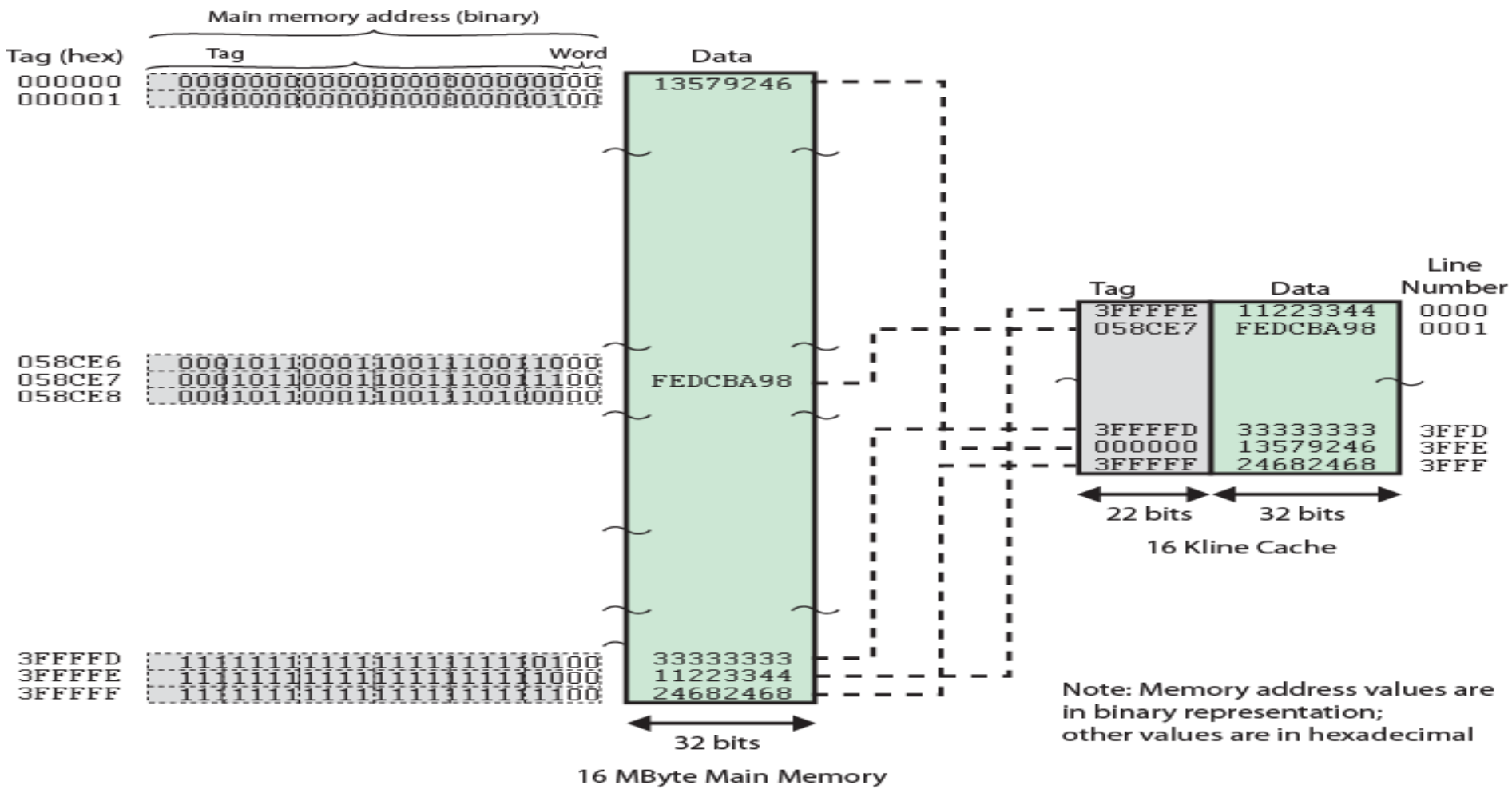
Fully Associative Cache Organization



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)

Associative Mapping Example



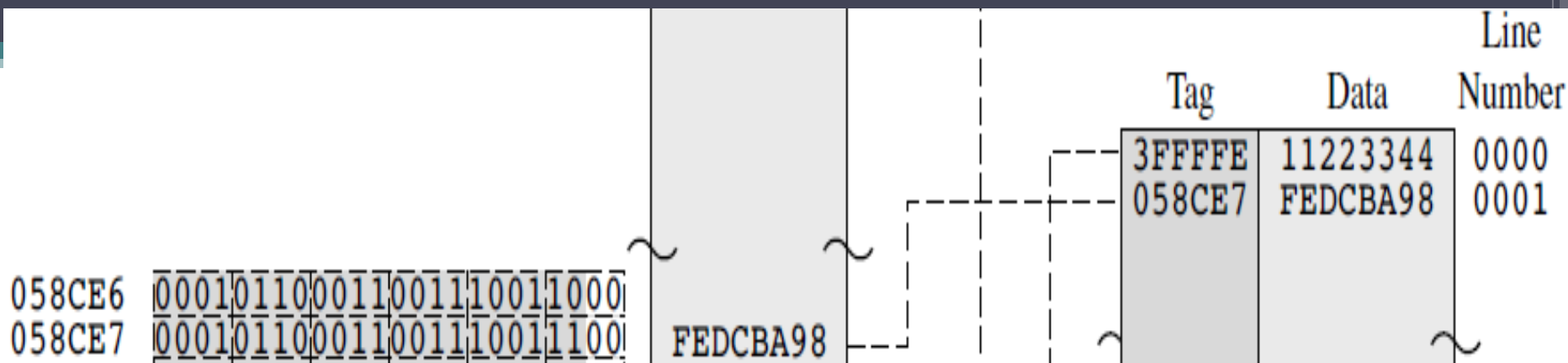
Associative Mapping Address Structure



- 22 bit tag stored with each 32 bit block of data of cache.
- Compare tag field with tag entry in cache to check for hit.
- Least significant 2 bits of address identify which 16 bit word is required from 32 bit data block.

• e.g.

Address	Tag	Data	Cache line
FFFFFC	FFFFFC	24682468	3FFF



- Address = 0001 0110 0011 0011 1001 1100

1 6 3 3 9 C

- Tag = 0000 0101 1000 1100 1110 0111

0 5 8 C E 7

- Data = FEDCBA98

- Cache line = 0001

Associative Mapping Pros & Cons

- **Advantages:**
 - Flexibility as to which block to replace when a new block is read into the cache.
 - It is designed to maximize the hit ratio.
- **Disadvantage:**
 - The complex circuitry required to examine the tags of all cache lines in parallel.

Set Associative Mapping

- Cache is divided into a number of sets.
- Each set contains a number of lines.
- A given block maps to any line in a given set.
 - e.g. Block B can be in any line of set i.
- e.g. 2 lines per set.
 - 2 way associative mapping.
 - A given block can be in one of 2 lines in only one set.

Set Associative Mapping

- The relationships are:

$$m = v * k$$

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

where

i = cache set number

j = main memory block number

m = number of lines in the cache

v = number of sets

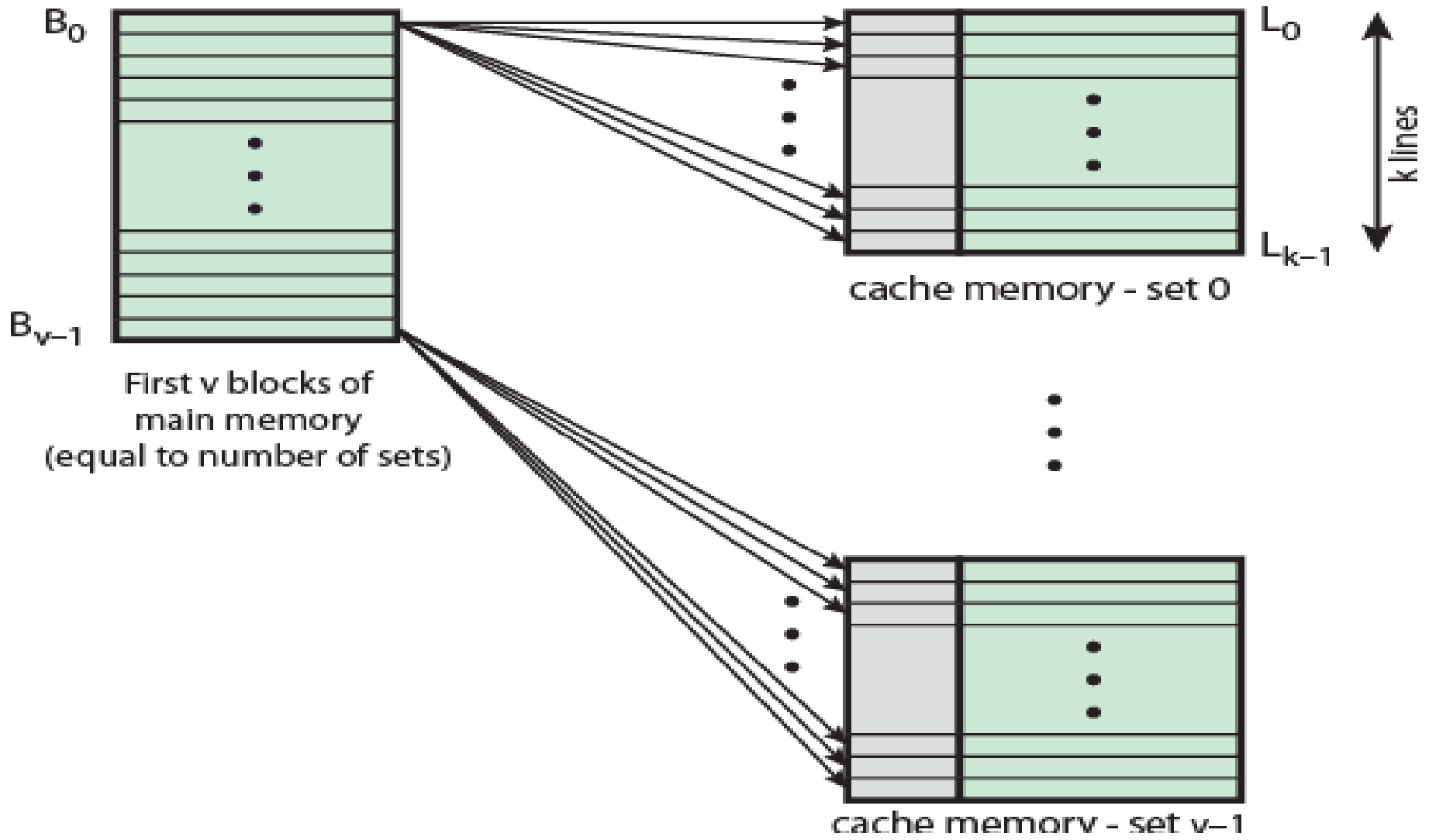
k = number of lines in each set

- This is referred to as k -way set-associative mapping.

v Associative-mapped caches

- The next figure illustrates this mapping for the first v blocks of main memory.
- For set-associative mapping, each word maps into all the cache lines in a specific set, so that main memory block B_0 maps into set 0, and so on.
- Thus, the set-associative cache can be physically implemented as v associative caches.

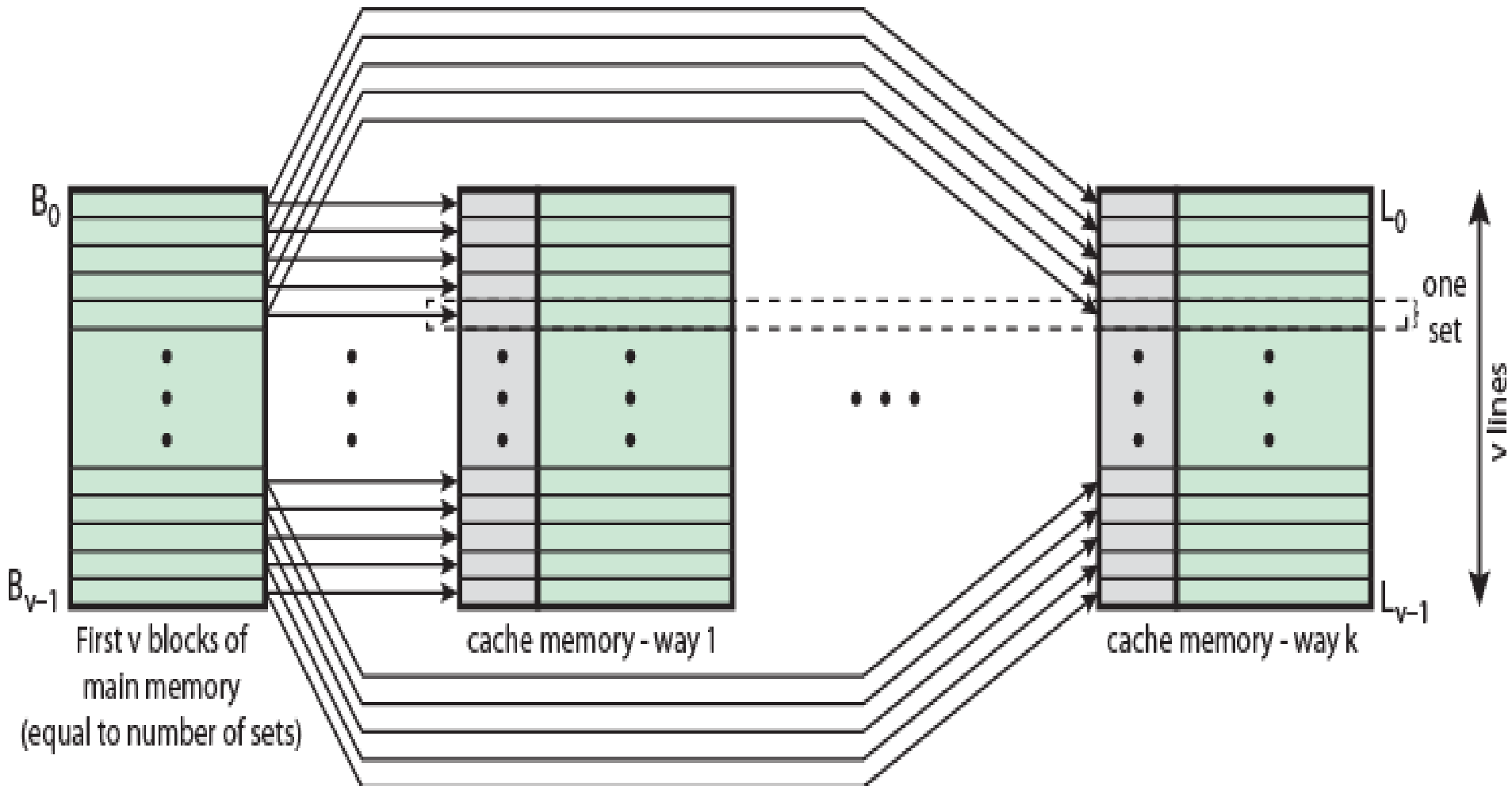
v Associative-mapped caches



k-way Associative-mapped caches or k Direct-mapped caches

- It is also possible to implement the set-associative cache as k direct mapping caches as next figure.
- Each direct-mapped cache is referred to as a way, consisting of v lines. The first v lines of main memory are direct mapped into the v lines of each way; the next group of v lines of main memory are similarly mapped, and so on.
- The direct-mapped implementation is typically used for small degrees of associativity (small values of k) while the associative-mapped implementation is typically used for higher degrees of associativity.

k-way Associative-mapped caches or k Direct-mapped caches

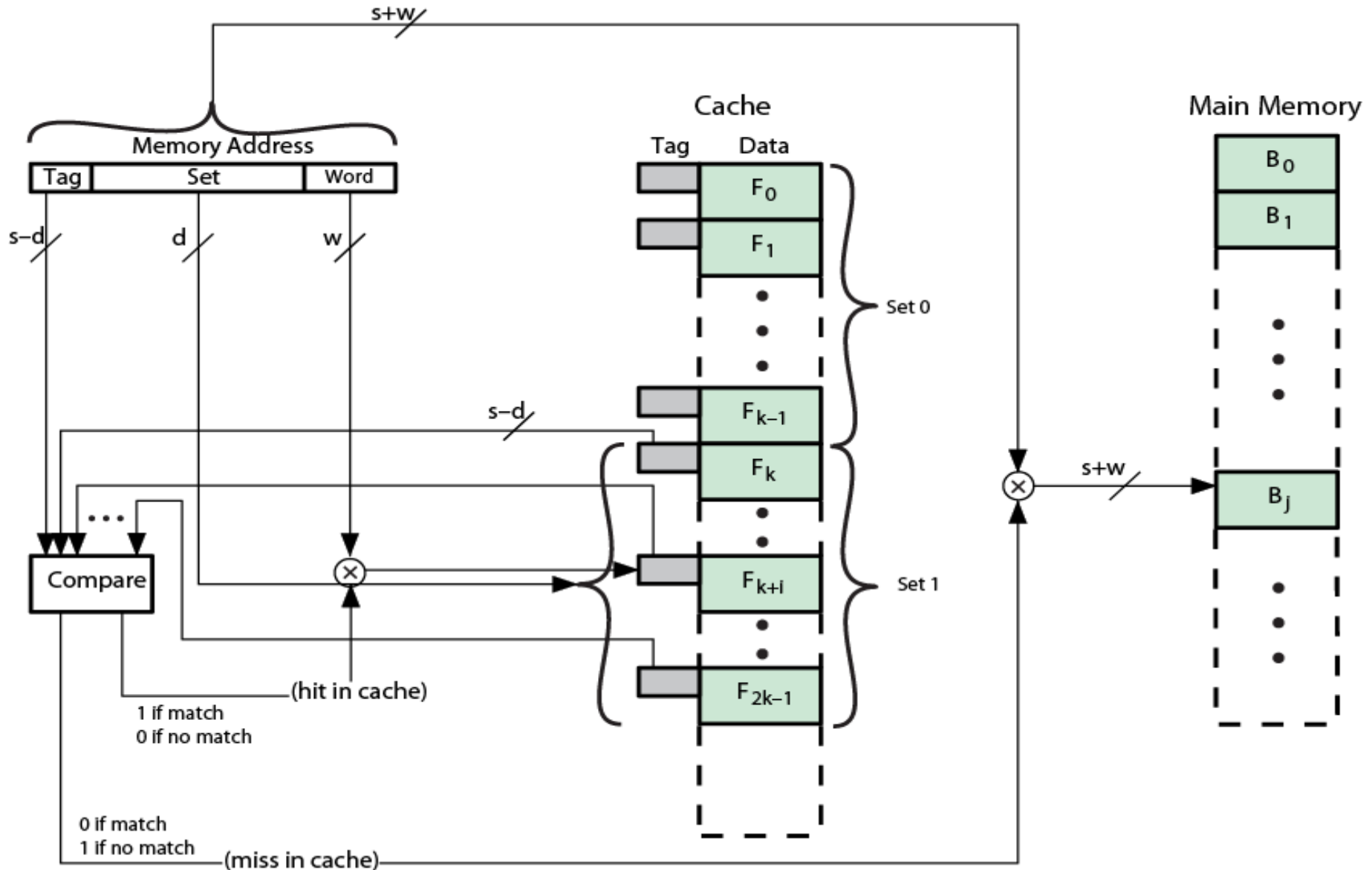


- The cache control logic interprets a memory address as three fields: Tag, Set, and Word.
- The d set bits specify one of $v = 2^d$ sets.
- The s bits of the Tag and Set fields specify one of the 2^s blocks of main memory.
- With fully associative mapping, the tag in a memory address is quite large and must be compared to the tag of every line in the cache. With k -way set-associative mapping, the tag in a memory address is much smaller and is only compared to the k tags within a single set.

Set Associative Mapping Summary

- Address length = $(s + w)$ bits.
- Number of addressable units = 2^{s+w} words or bytes.
- Block size = line size = 2^w words or bytes.
- Number of blocks in main memory = $2^{s+w} / 2^w = 2^s$.
- Number of lines in set = k .
- Number of sets = $v = 2^d$.
- Number of lines in cache = $m = k * v = k * 2^d$.
- Size of cache = $k * 2^{d+w}$ words or bytes.
- Size of tag = $(s - d)$ bits.

K-Way Set Associative Cache Organization



Set Associative Mapping Example

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.

Set Associative Mapping Address Structure

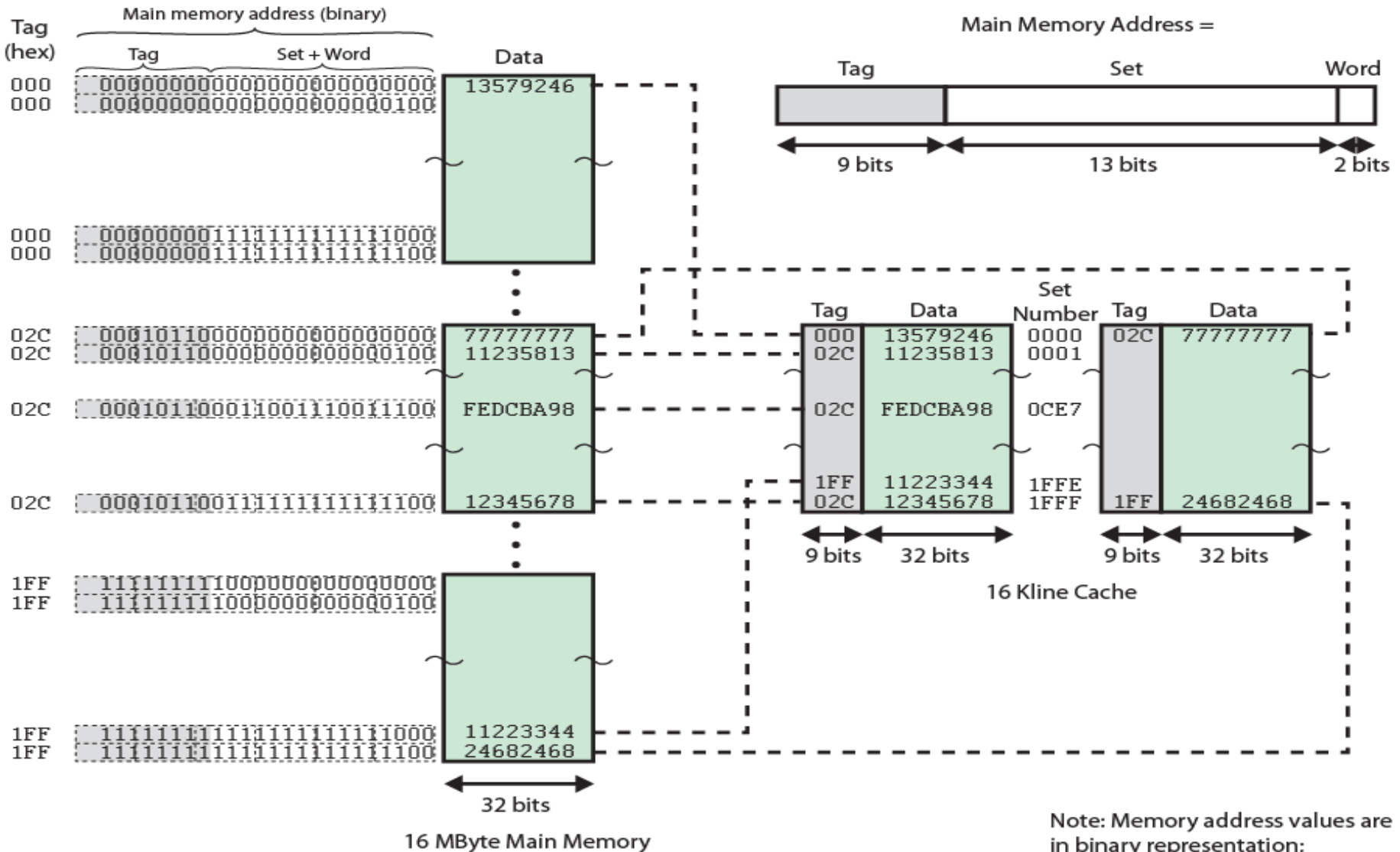
Tag 9 bit	Set 13 bit	Word 2 bit
------------------	-------------------	----------------------

- Use set field to determine cache set to look in.
- Compare tag field to see if we have a hit.

• e.g

Address	Tag	Data	Set number
1FF 7FFC	1FF	12345678	1FFF
001 7FFC	001	11223344	1FFF

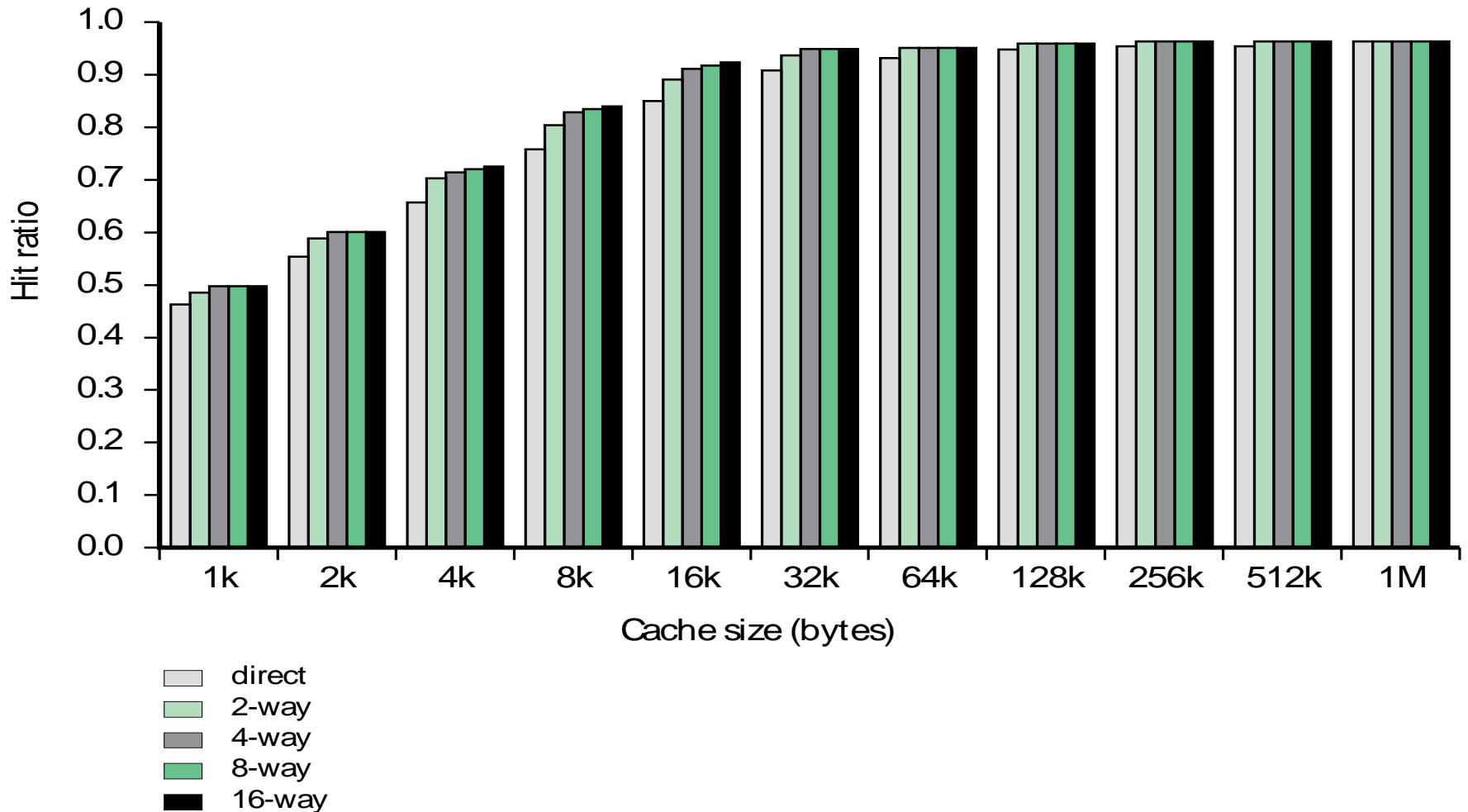
Two Way Set Associative Mapping Example



Direct and Set Associative Cache Performance Differences

- Significant up to at least 64kB for 2-way.
- Difference between 2-way and 4-way at 4kB much less than 4kB to 8kB.
- Cache complexity increases with associativity.
- Not justified against increasing cache to 8kB or 16kB.
- Above 32kB gives no improvement.
- (simulation results).

Figure 4.16 Varying Associativity over Cache Size



Replacement Algorithms

Direct mapping

- No choice.
- Each block only maps to one line.
- Replace that line.

Replacement Algorithms

Associative & Set Associative

- To achieve high speed, such an algorithm must be implemented in hardware.
- Once the cache has been filled, when a new block is brought into the cache, one of the existing blocks must be replaced.
- The four most popular replacement algorithms:
 1. Least Recently used (LRU).
 2. First in first out (FIFO).
 3. Least frequently used (LFU).
 4. Random.

Replacement Algorithms

Associative & Set Associative

1. Least Recently used (LRU)
 - Replace that block in the set that has been in the cache longest with no reference to it.
 - e.g. in 2 way set associative.
 - Which of the 2 block is LRU?
2. First in first out (FIFO)
 - Replace block that has been in cache longest.
3. Least frequently used (LFU)
 - Replace block which has had fewest hits.
4. Random
 - 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:
 1. If the old block in the cache has not been altered, then it may be overwritten with a new block without first writing out the old block.
 2. If at least one write operation has been performed on a word in that line of the cache, then main memory must be updated by writing the line of cache out to the block of memory before bringing in the new block. A

Write Policy – Some problems to contend with

1. More than one device may have access to main memory.

For example, an I/O module may be able to read-write directly to memory. If a word has been altered only in the cache, then the corresponding memory word is invalid.

Further, if the I/O device has altered main memory, then the cache word is invalid.

2. More complex problem occurs when multiple processors are attached to the same bus and each processor has its own local cache.

Then, if a word is altered in one cache, it could invalidate a word in other caches.

Write Policy – Some problems to contend with

3. If data in one cache are altered, this invalidates not only the corresponding word in main memory, but also that same word in other caches (if any other cache happens to have that same word).

Write through

- All writes go to main memory as well as cache for ensuring that main memory is always valid.
- Multiple CPUs can monitor main memory traffic to keep local (to CPU) cache up to date.
- Lots of traffic.
- Slows down writes.
- Remember bogus write through caches!

Write back

- To minimize memory writes.
- With 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.
- Other caches get out of sync.
- I/O must access main memory through cache.

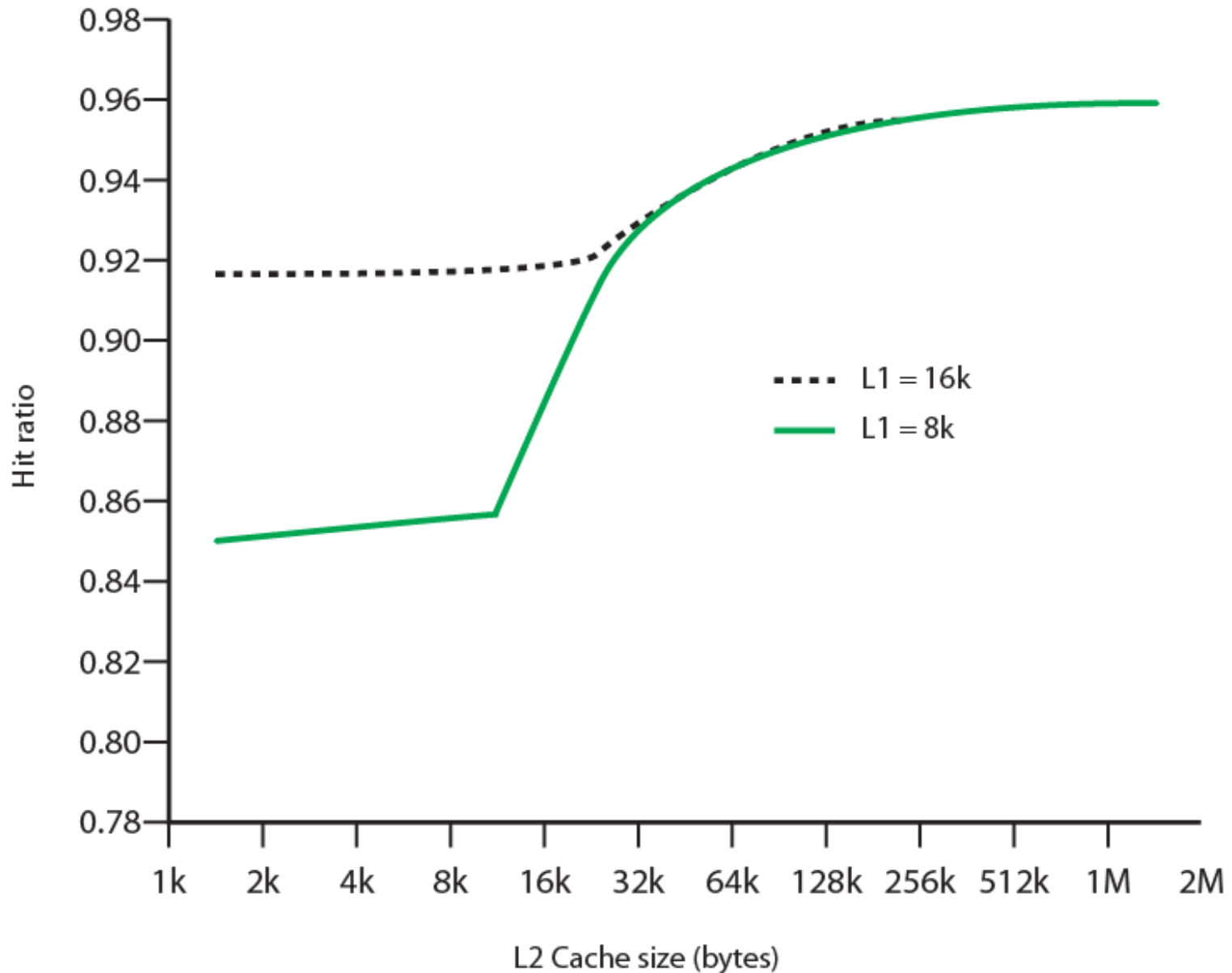
Line Size

- Retrieve not only desired word but a number of adjacent words as well.
- Increased block size will increase hit ratio at first.
 - the principle of locality.
- Hit ratio will decrease as block becomes even bigger.
 - Probability of using newly fetched information becomes less than probability of reusing the information to be replaced.
- Larger blocks
 - Reduce number of blocks that fit in cache.
 - Data overwritten shortly after being fetched.
 - Each additional word is less local so less likely to be needed.
- No definitive optimum value has been found.
- 8 to 64 bytes seems reasonable.

Multilevel Caches

- High logic density enables caches on chip.
 - Faster than bus access.
 - Frees bus for other transfers.
- Common to use both on and off chip cache.
 - L1 on chip, L2 off chip in static RAM (SRAM).
 - L2 access much faster than DRAM or ROM.
 - L2 often uses separate data path.
 - L2 may now be on chip.
 - Resulting in L3 cache.
 - Bus access or now on chip...

Hit Ratio (L1 & L2) For 8 kbytes and 16 kbyte L1



Unified v Split Caches

- One cache for data and instructions or two, one for data and one for instructions.
- Advantages of unified cache:
 - Higher hit rate.
 - Balances load of instruction and data fetch.
 - Only one cache to design & implement.
- Advantages of split cache:
 - Eliminates cache contention between instruction fetch/decode unit and execution unit.
 - Important in pipelining.



Self-Study
from page #158

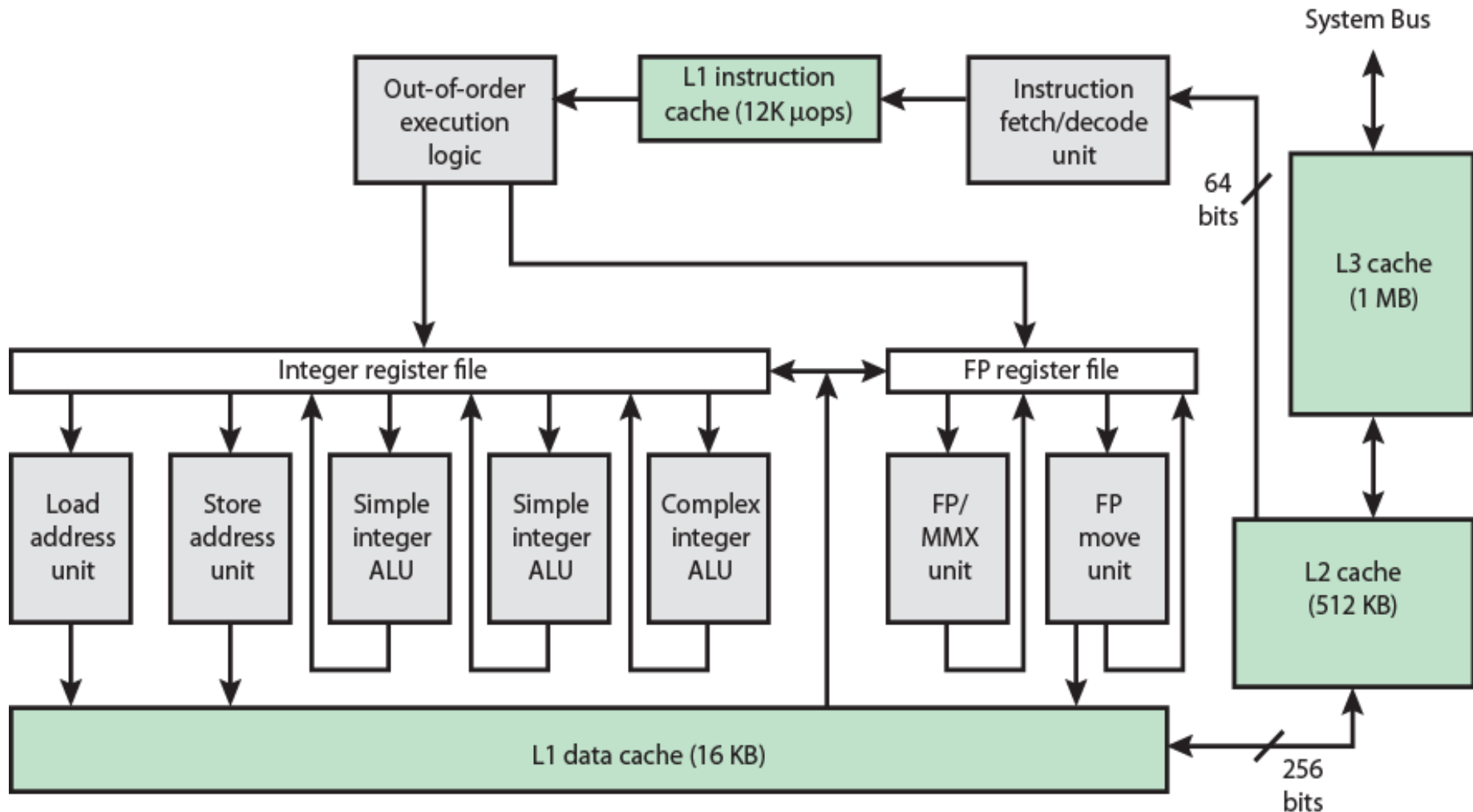
Pentium 4 Cache

- 80386 – no on chip cache
- 80486 – 8k using 16 byte lines and four way set associative organization
- Pentium (all versions) – two on chip L1 caches
 - Data & instructions
- Pentium III – L3 cache added off chip
- Pentium 4
 - L1 caches
 - 8k bytes
 - 64 byte lines
 - four way set associative
 - L2 cache
 - Feeding both L1 caches
 - 256k
 - 128 byte lines
 - 8 way set associative
 - L3 cache on chip

Intel Cache Evolution

Problem	Solution	Processor on which feature first appears
External memory slower than the system bus.	Add external cache using faster memory technology.	386
Increased processor speed results in external bus becoming a bottleneck for cache access.	Move external cache on-chip, operating at the same speed as the processor.	486
Internal cache is rather small, due to limited space on chip	Add external L2 cache using faster technology than main memory	486
Contention occurs when both the Instruction Prefetcher and the Execution Unit simultaneously require access to the cache. In that case, the Prefetcher is stalled while the Execution Unit's data access takes place.	Create separate data and instruction caches.	Pentium
Increased processor speed results in external bus becoming a bottleneck for L2 cache access.	Create separate back-side bus that runs at higher speed than the main (front-side) external bus. The BSB is dedicated to the L2 cache.	Pentium Pro
	Move L2 cache on to the processor chip.	Pentium II
Some applications deal with massive databases and must have rapid access to large amounts of data. The on-chip caches are too small.	Add external L3 cache.	Pentium III
	Move L3 cache on-chip.	Pentium 4

Pentium 4 Block Diagram



Pentium 4 Core Processor

- **Fetch/Decode Unit**
 - Fetches instructions from L2 cache
 - Decode into micro-ops
 - Store micro-ops in L1 cache
- **Out of order execution logic**
 - Schedules micro-ops
 - Based on data dependence and resources
 - May speculatively execute
- **Execution units**
 - Execute micro-ops
 - Data from L1 cache
 - Results in registers
- **Memory subsystem**
 - L2 cache and systems bus

Pentium 4 Design Reasoning

- Decodes instructions into RISC like micro-ops before L1 cache
- Micro-ops fixed length
 - Superscalar pipelining and scheduling
- Pentium instructions long & complex
- Performance improved by separating decoding from scheduling & pipelining
 - (More later – ch14)
- Data cache is write back
 - Can be configured to write through
- L1 cache controlled by 2 bits in register
 - CD = cache disable
 - NW = not write through
 - 2 instructions to invalidate (flush) cache and write back then invalidate
- L2 and L3 8-way set-associative
 - Line size 128 bytes

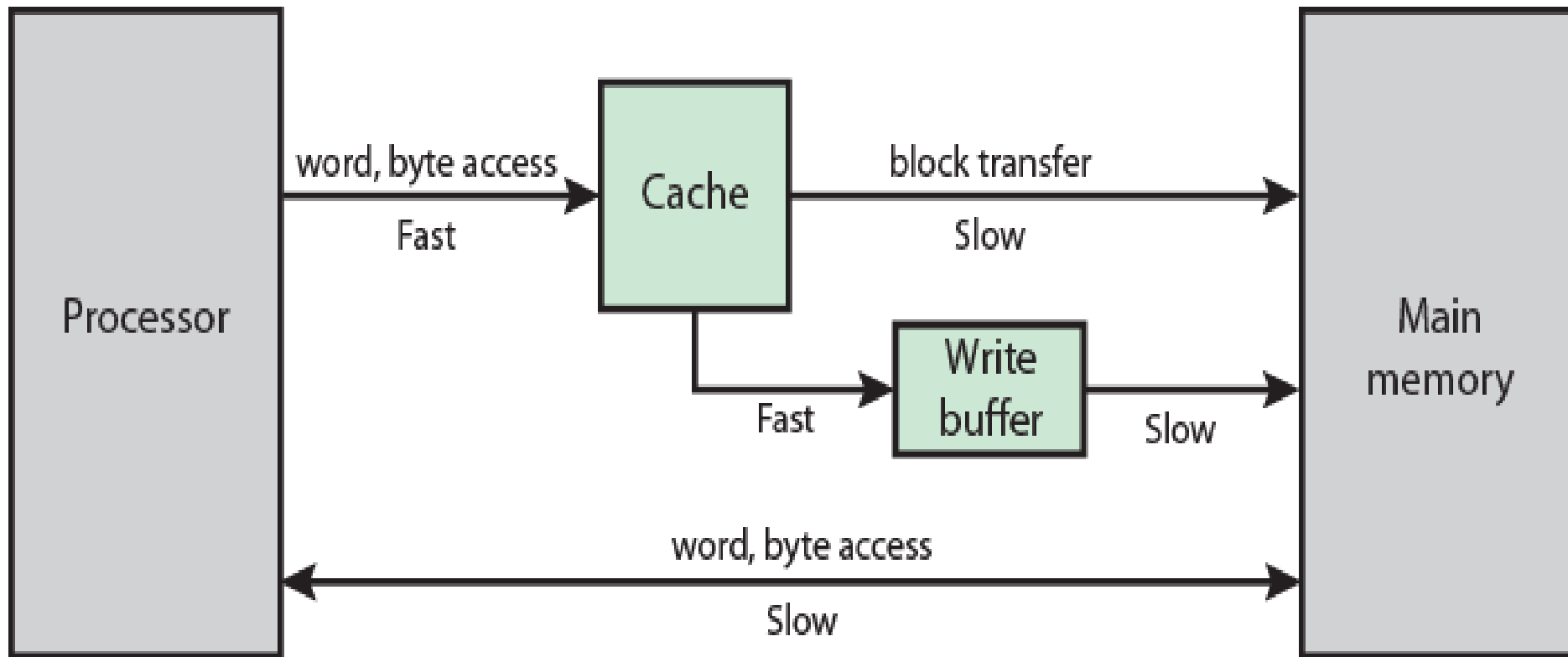
ARM Cache Features

Core	Cache Type	Cache Size (kB)	Cache Line Size (words)	Associativity	Location	Write Buffer Size (words)
ARM720T	Unified	8	4	4-way	Logical	8
ARM920T	Split	16/16 D/I	8	64-way	Logical	16
ARM926EJ-S	Split	4-128/4-128 D/I	8	4-way	Logical	16
ARM1022E	Split	16/16 D/I	8	64-way	Logical	16
ARM1026EJ-S	Split	4-128/4-128 D/I	8	4-way	Logical	8
Intel StrongARM	Split	16/16 D/I	4	32-way	Logical	32
Intel Xscale	Split	32/32 D/I	8	32-way	Logical	32
ARM1136-JF-S	Split	4-64/4-64 D/I	8	4-way	Physical	32

ARM Cache Organization

- **Small FIFO write buffer**
 - Enhances memory write performance
 - Between cache and main memory
 - Small c.f. cache
 - Data put in write buffer at processor clock speed
 - Processor continues execution
 - External write in parallel until empty
 - If buffer full, processor stalls
 - Data in write buffer not available until written
 - So keep buffer small

ARM Cache and Write Buffer Organization



Internet Sources

- Manufacturer sites.
 - Intel
 - ARM
- Search on cache.