CSE 4305
Computer Organization and Architecture

Cache Memory

Course Teacher: Md. Hamjajul Ashmafee

Lecturer, CSE, IUT

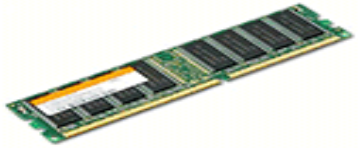
Email: ashmafee@iut-dhaka.edu

What is memory???

Computer Memory System

- **Storage** of the computer
- It has **wide range** of type, technology, performance, cost
- **No single technology** is optimal satisfying all requirements
- So a **hierarchy of memory** subsystems is **maintained** based on different features


Hard Disk


RAM


ROM


CD/DVD


Floppy

Memory Card


Pen Drive

Tape

Classification of Memory based on their key Characteristics

Location
- Internal (e.g., processor registers, cache, main memory)
- 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

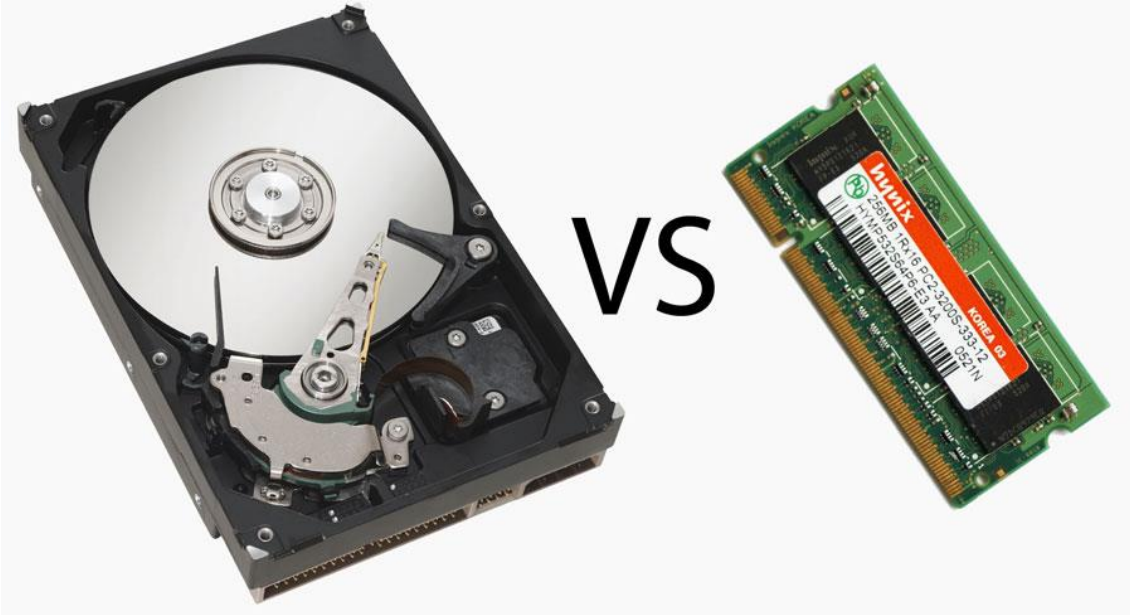
Physical Type
- Semiconductor
- Magnetic
- Optical
- Magneto-optical

Physical Characteristics
- Volatile/nonvolatile
- Erasable/nonerasable

Organization
- Memory modules

:: Location

- **Internal** – main memory, registers, cache – accessed by processor
- **External** – disk, tape – not directly accessed by processor rather via I/O controller – known as secondary/ auxiliary memory.



:: Capacity

- **Internal memory:** expressed in terms of **bits or bytes or words**
- **External memory:** expressed in **bytes** (MB, GB, TB, …)

:: Unit of Transfer

- **Word transfer:** transfer is equal to the number of **electrical lines** connected with memory module – word length
- **Block transfer:** data often transferred in much **larger units** than a word referring as **blocks**

:: Method of Access

- **Sequential Access:** in a specific linear sequence – magnetic tape
- **Direct Access:** directly reached based on locality – magnetic disk
- **Random Access:** any location can be selected at random and directly addressed and accessed – register, RAM
- **Associative Access:** based on comparison of desired location – cache memory

:: Performance

- Not directly specified
- Categorized based on the parameters:
  - **Access time (latency)** – from the instant of addressing to the instant of available – to read/ write
  - **Memory Cycle Time** –access time plus time until next access
  - **Transfer rate** – rate at which data is transferred

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

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)

:: Physical Characteristics

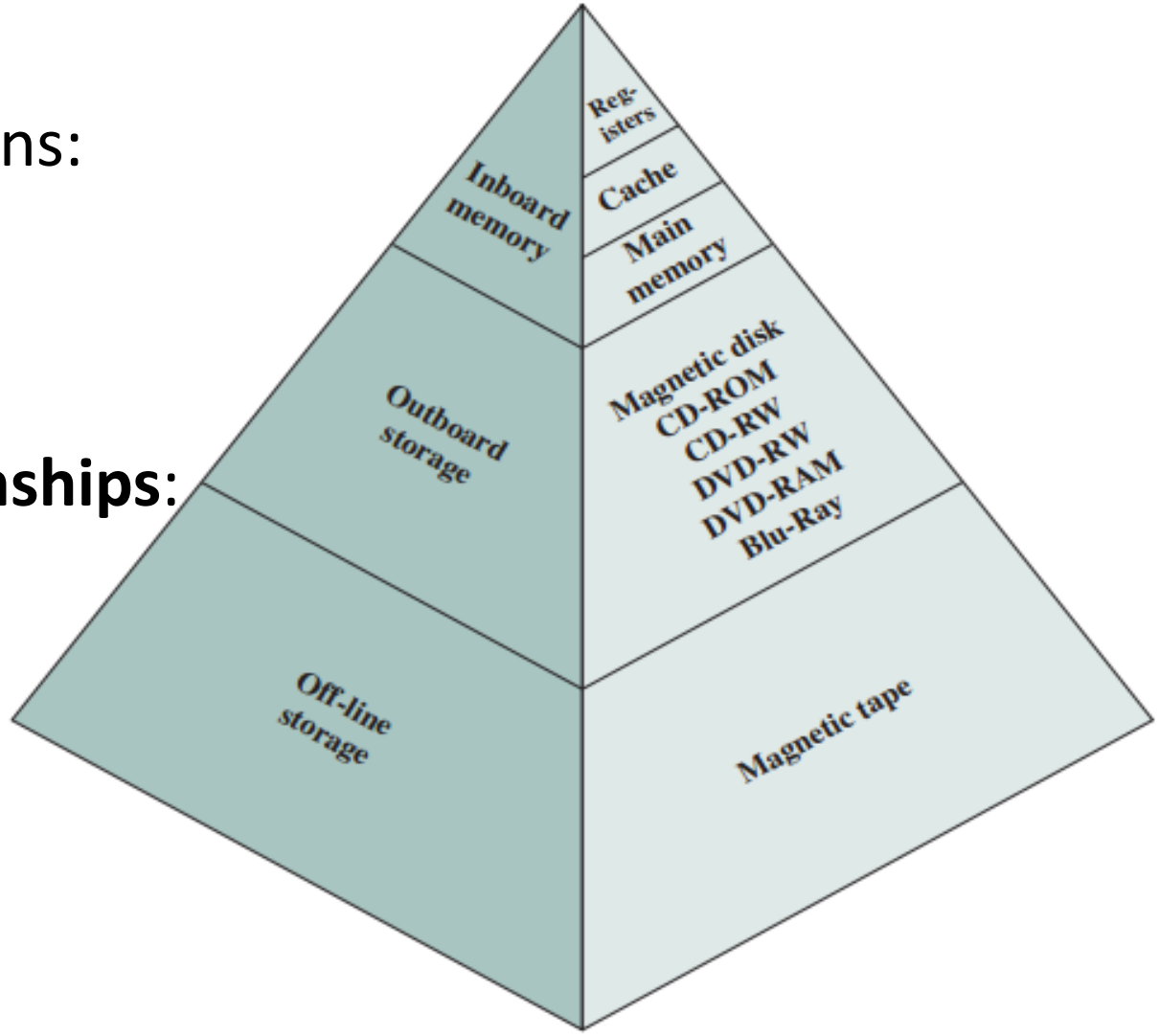
- **Volatile memory** - information decays naturally or is lost when electrical power is switched off
- **Nonvolatile memory** - information once recorded remains without deterioration until deliberately changed; no electrical power is needed to retain information.

:: Organization

- Also Not directly specified
- Memory cell's specific technology – **IC types and their relative position**

Memory Hierarchy

- **Design Constraints** based on 3 questions:
  - How much?
  - How fast?
  - How expensive?
- Modern technologies hold the **relationships**:
  - Faster access time, greater cost per bit;
  - Greater capacity, smaller cost per bit;
  - Greater capacity, slower access time.



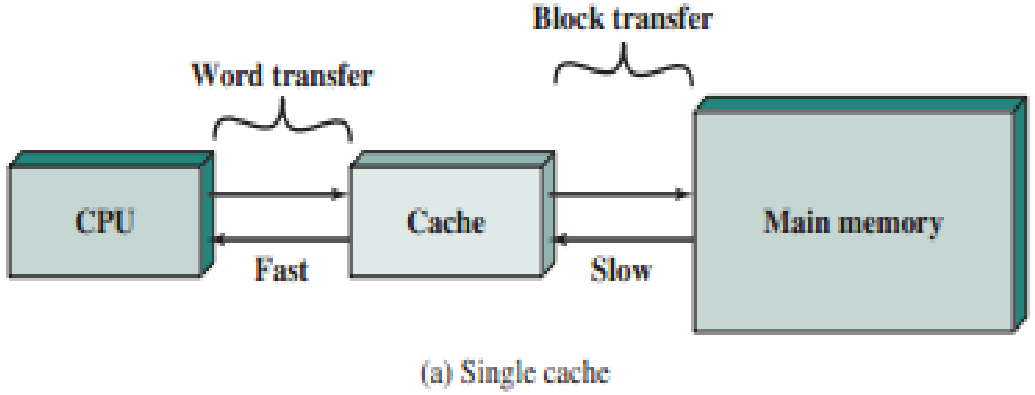
Memory covered here:

- Cache Memory
- Internal Memory
- External Memory

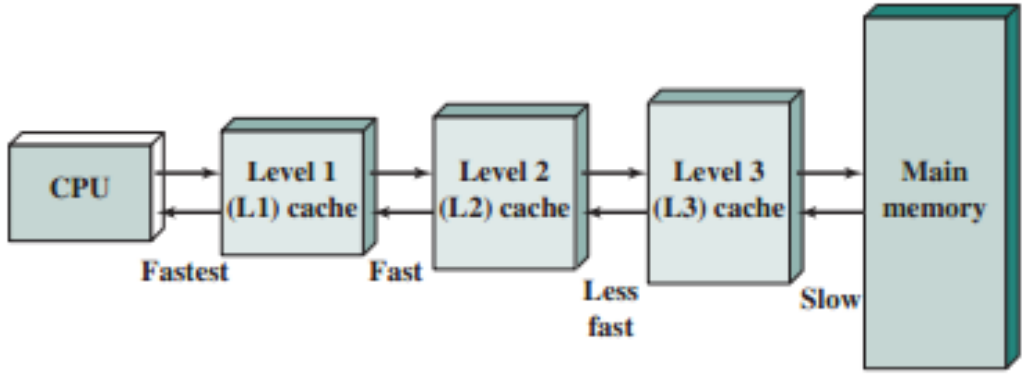
Cache Memory

Cache Memory Principles

- Cache memory is designed **to combine** the **memory access time** of **expensive, high-speed memory** combined with the **large memory size** of **less expensive, lower-speed memory**.
- The cache contains a **copy** of portions of main memory.



(a) Single cache

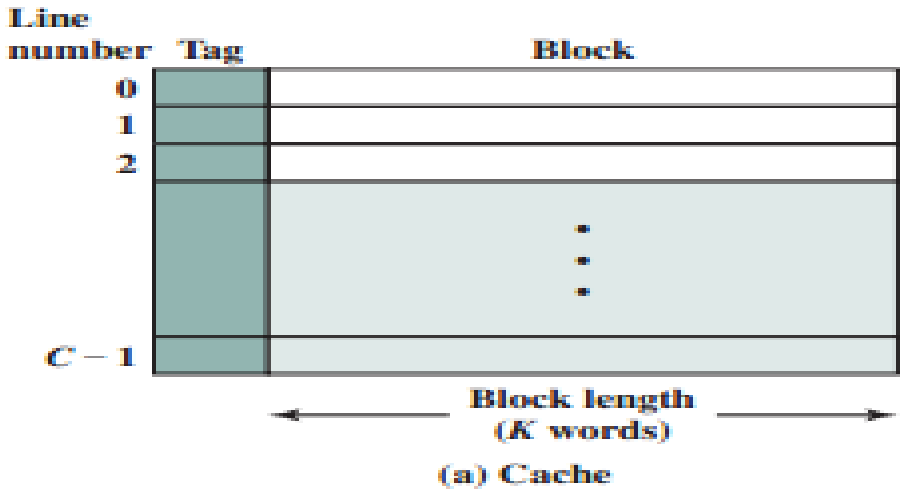


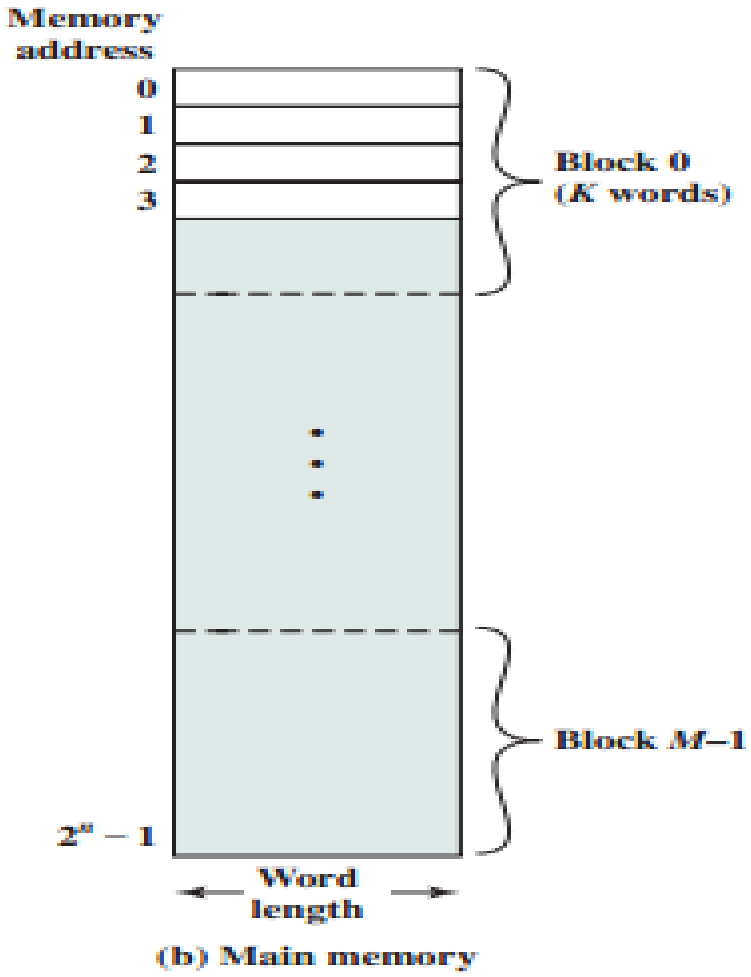
(b) Three-level cache organization

Cache Memory Principles…

- 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. (**Locality of reference**)
- **Locality of reference** - when a block of data is fetched into the cache to satisfy **a single memory reference**, it is **likely** that there will be future references to that same memory location or to other words in the block

Structure of Cache – Main Memory System

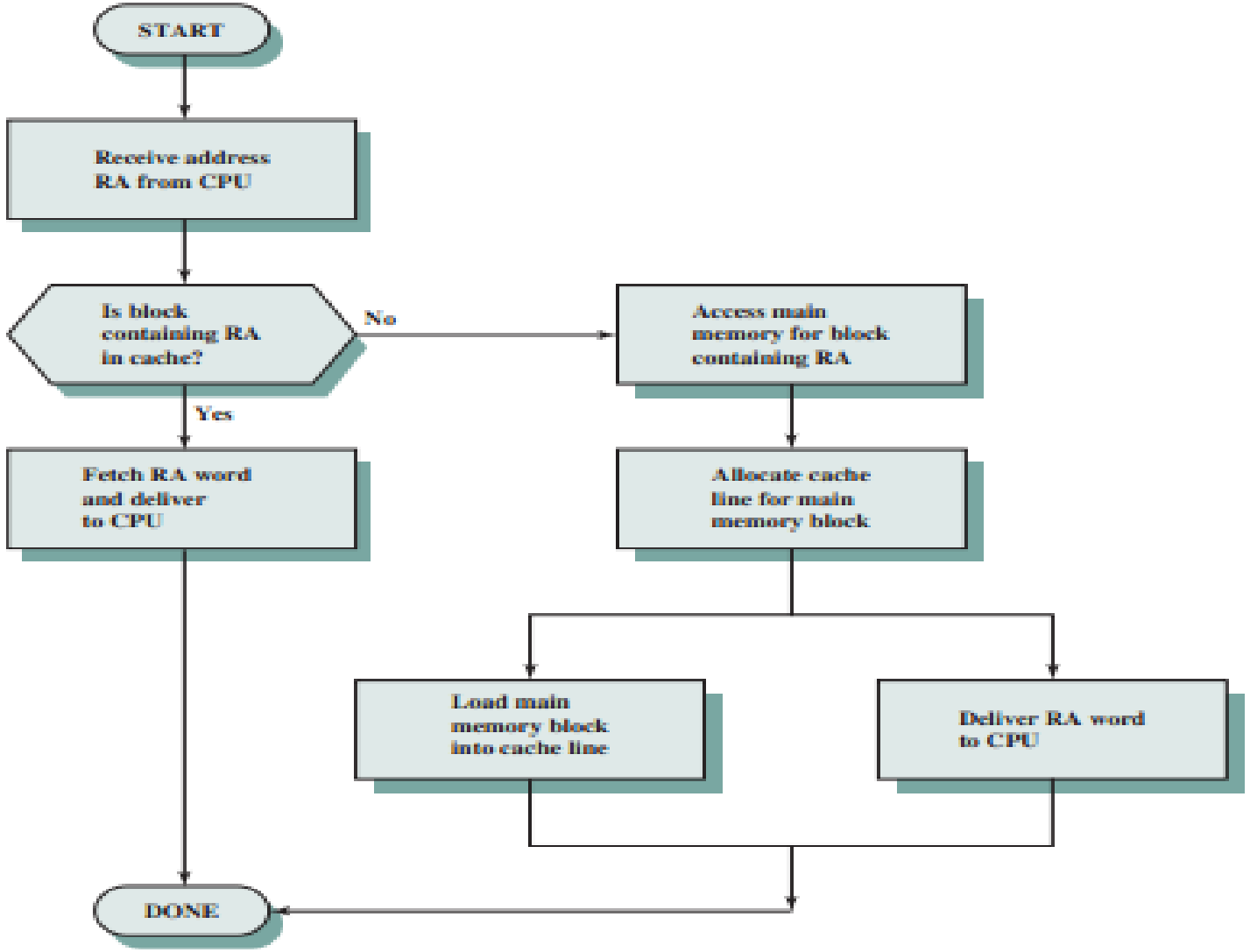




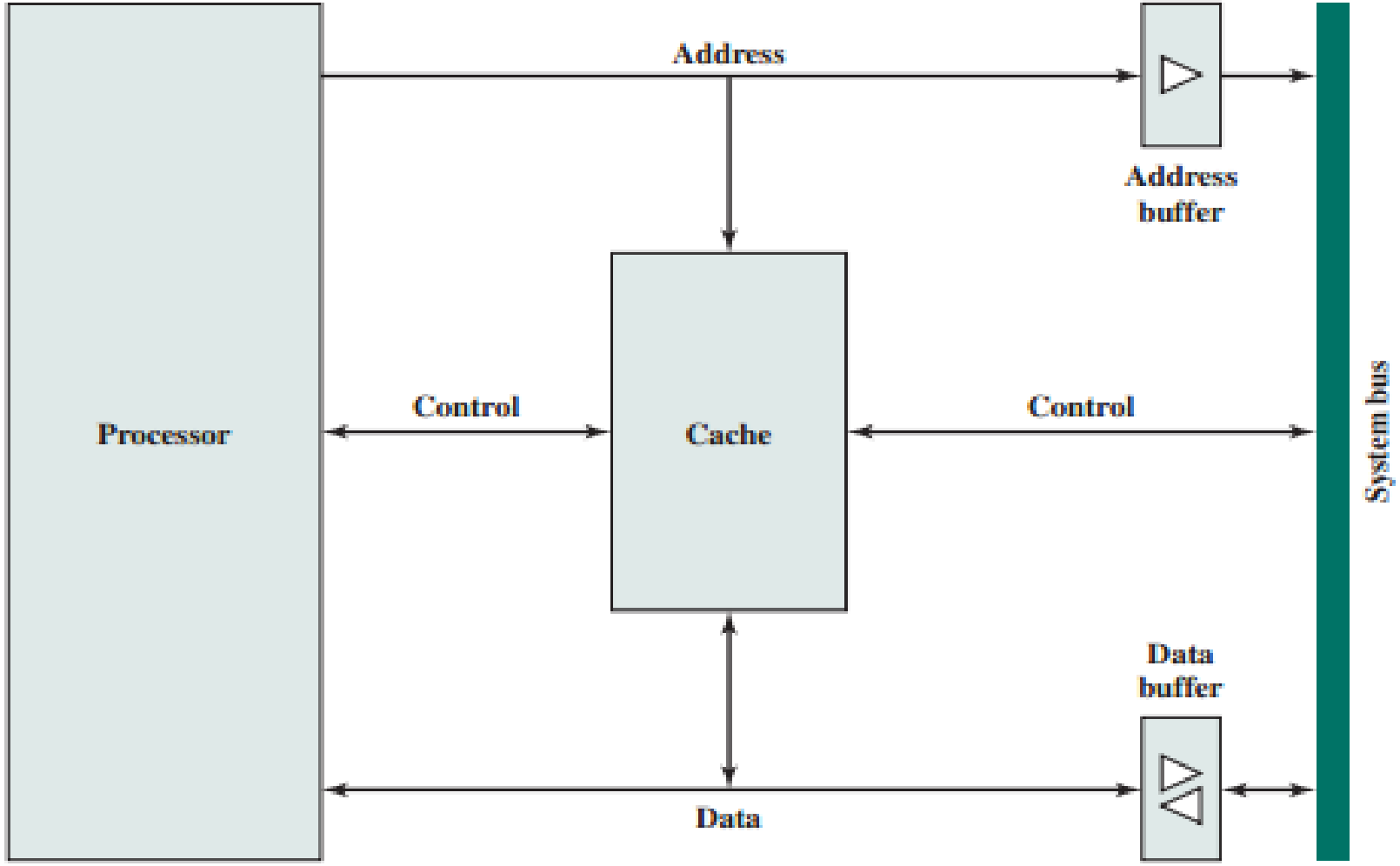
- Memory Address
- Block
- Line
- Tag

We can not utilize the whole space of any memory device, can we?

Cache Read Operation



Typical Cache Organization



Elements of Cache Design

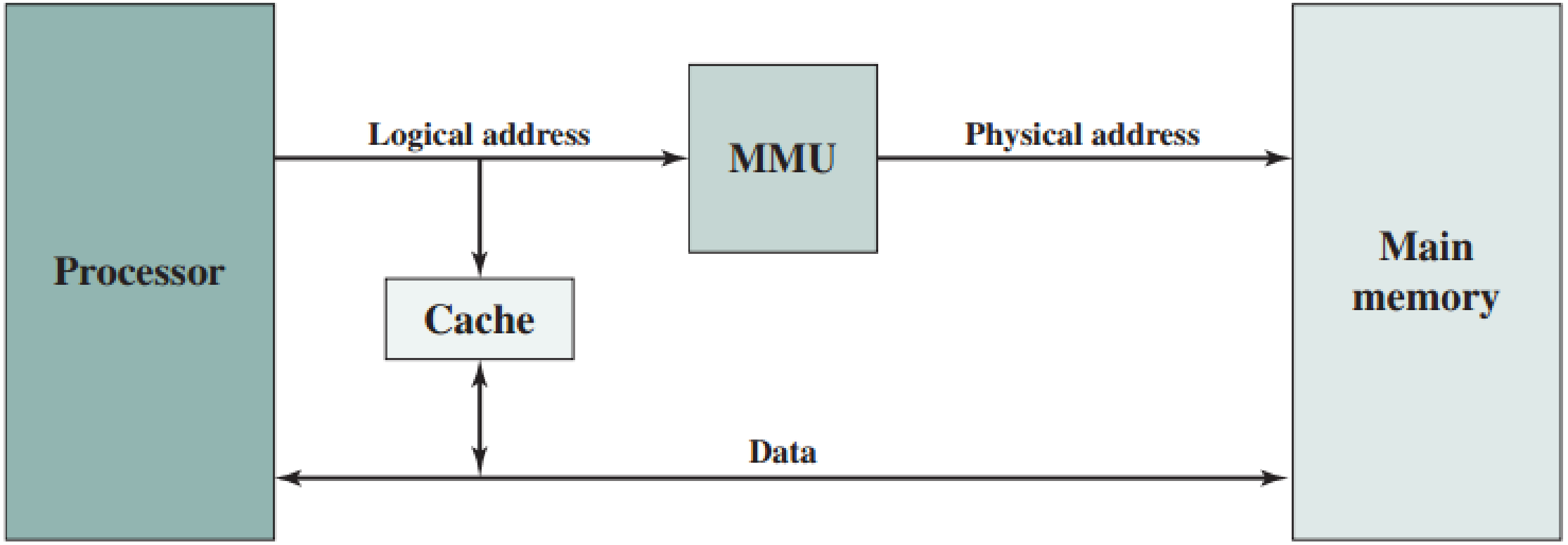
- Parameters considered in Cache design:

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

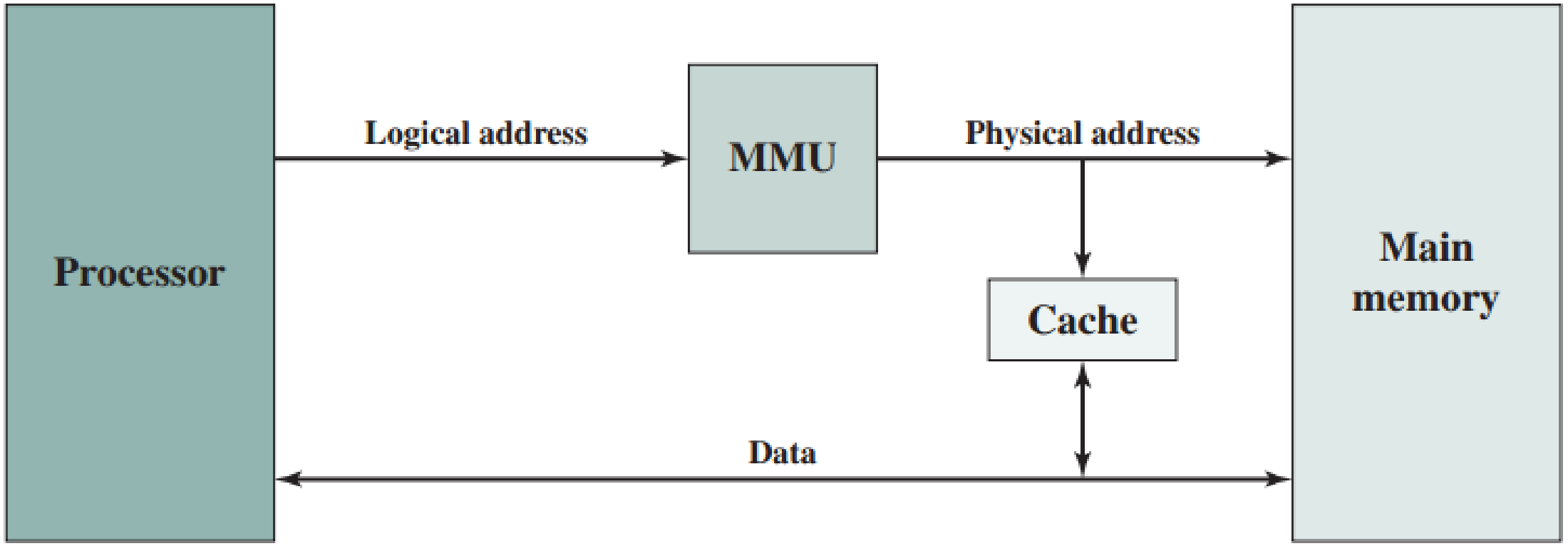
:: Cache Address

- As **virtual memory** is supported, it allows **a program** to address memory from **logical** point of view or **physical** point of view (as usual)
  - **Virtual Memory**: an imaginary **memory space** used to enlarge the address space **virtually** **as if** it can accommodate whole program in main memory
- When virtual memory is **used**, it produces logical (virtual) address – **MMU** (Memory Management Unit) is used to translate logical address to physical address.
- If the cache placed between **processor and MMU**, it stores data using **logical address** so that processor can directly access the cache
- **Otherwise**, processor needs MMU to access the cache

:: Cache Address: Logical Cache



:: Cache Address: Physical Cache



:: Cache Size

- Already discussed in Memory Hierarchy & Cache Memory Principle

Processor	Type	Year of Introduction	L1 Cache[a]	L2 Cache	L3 Cache
IBM 360/85	Mainframe	1968	16–32 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	128–256 kB	—	—
Intel 80486	PC	1989	8 kB	—	—
Pentium	PC	1993	8 kB/8 kB	256–512 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 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
Itanium 2	PC/server	2002	32 kB	256 kB	6 MB
IBM POWER5	High-end server	2003	64 kB	1.9 MB	36 MB

:: Mapping Function

- **Fewer cache lines** than **main memory blocks** - an **algorithm** is needed for mapping main memory blocks into cache lines:
  - Direct Mapping
  - Associative Mapping
  - Set-associative Mapping

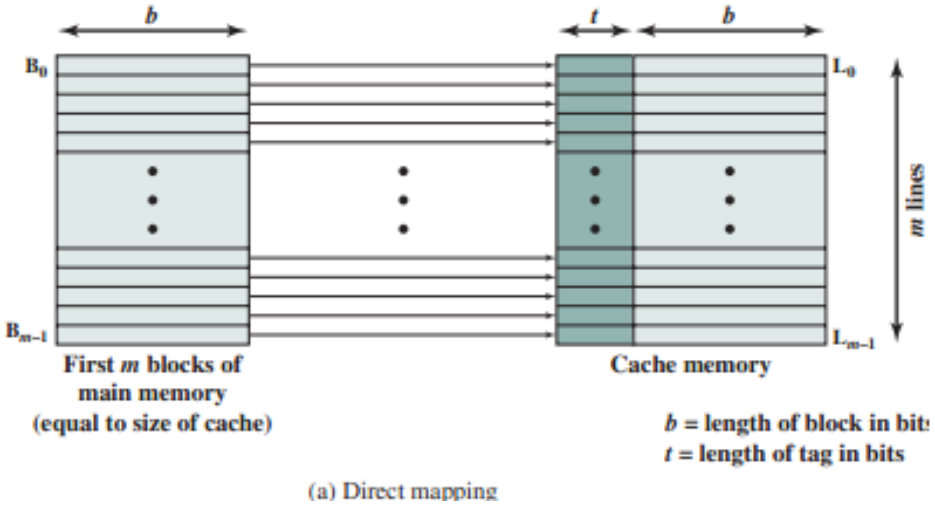
:: Mapping Function: Direct Mapping

- **Simplest** technique
- Maps **each block of main memory** into **only one possible cache line**
- Mapping expressed as

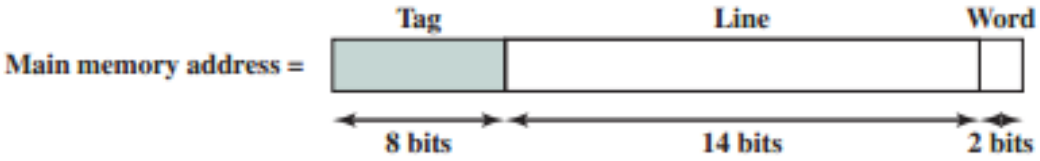
$$i = j \text{ modulo } m$$

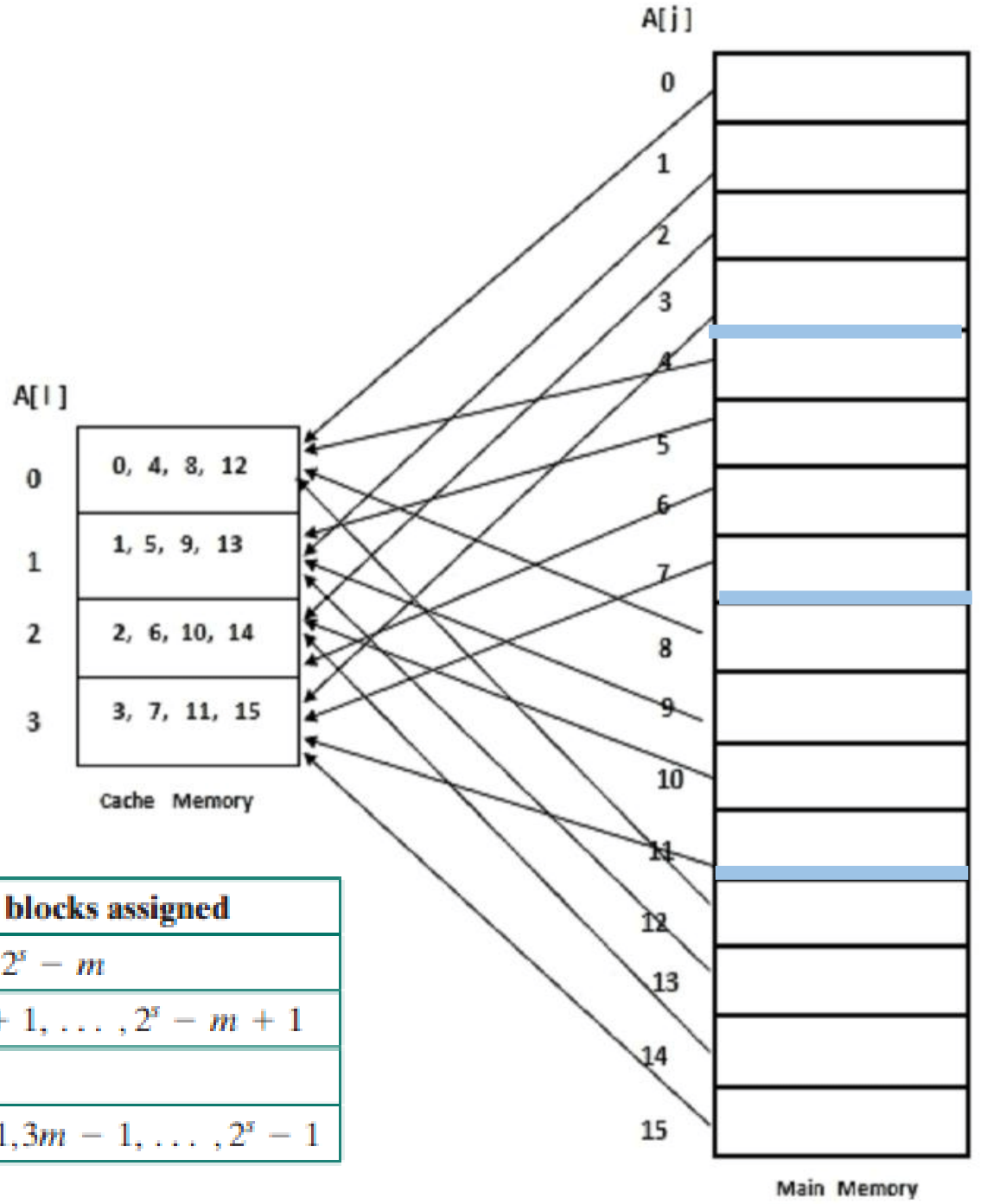
i = cache line number
j = main memory block number
m = number of lines in the cache

: Direct Mapping…



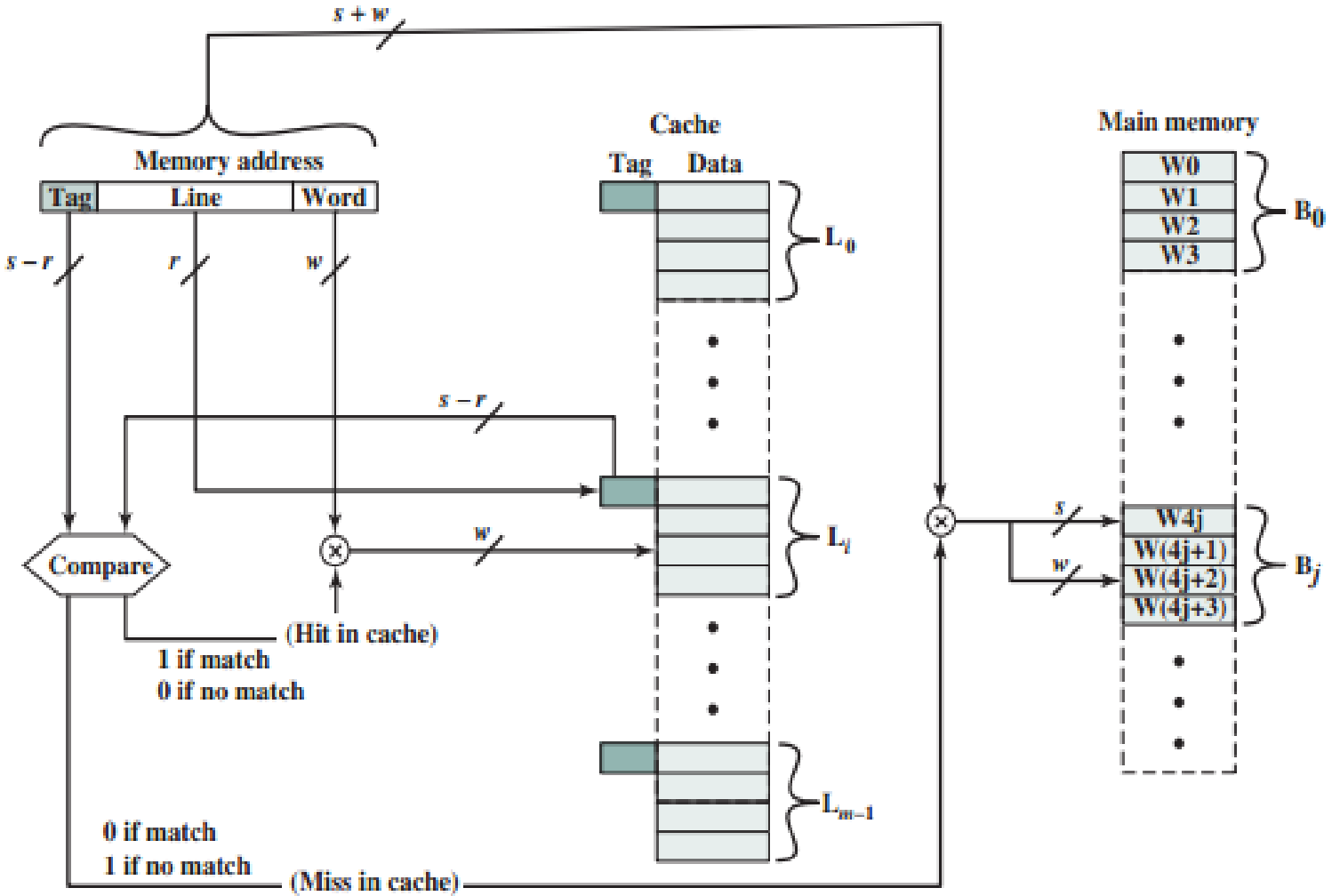
(a) Direct mapping





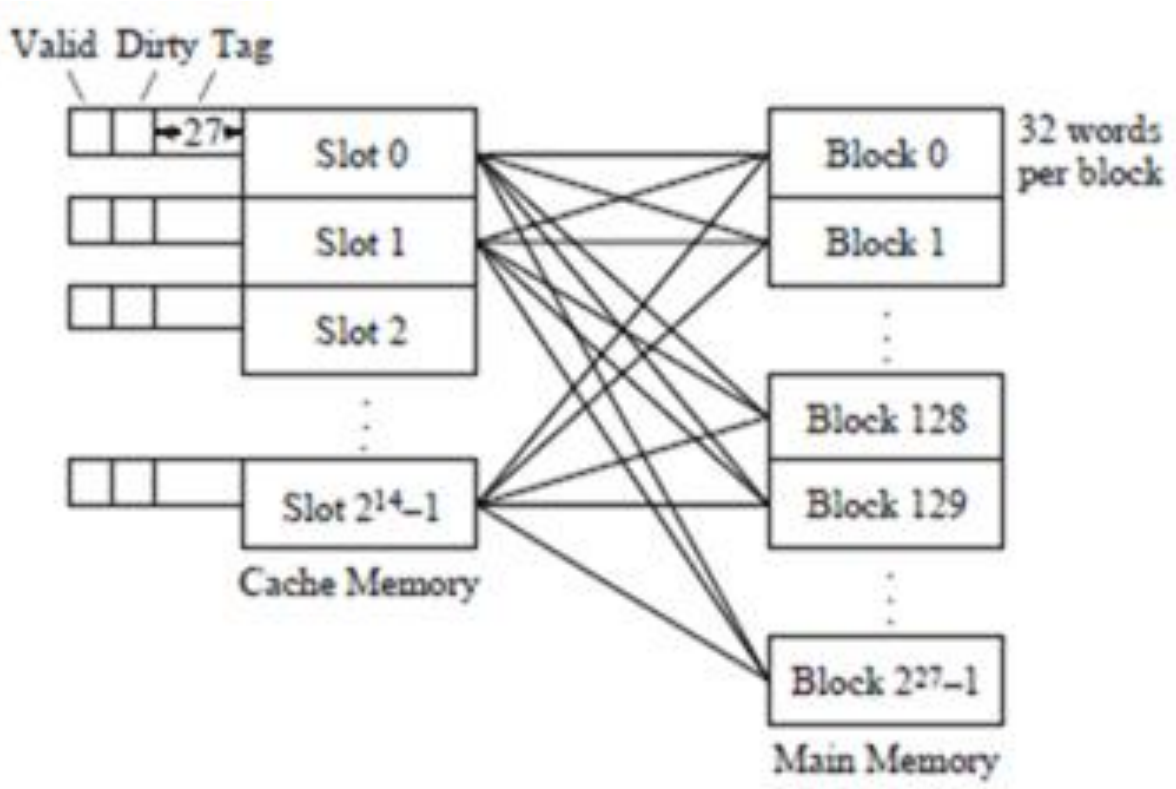
Cache line	Main memory blocks assigned
0	$0, m, 2m, \ldots, 2^s - m$
1	$1, m+1, 2m+1, \ldots, 2^s - m + 1$
⋮	⋮
$m-1$	$m-1, 2m-1, 3m-1, \ldots, 2^s - 1$

Direct Mapping…

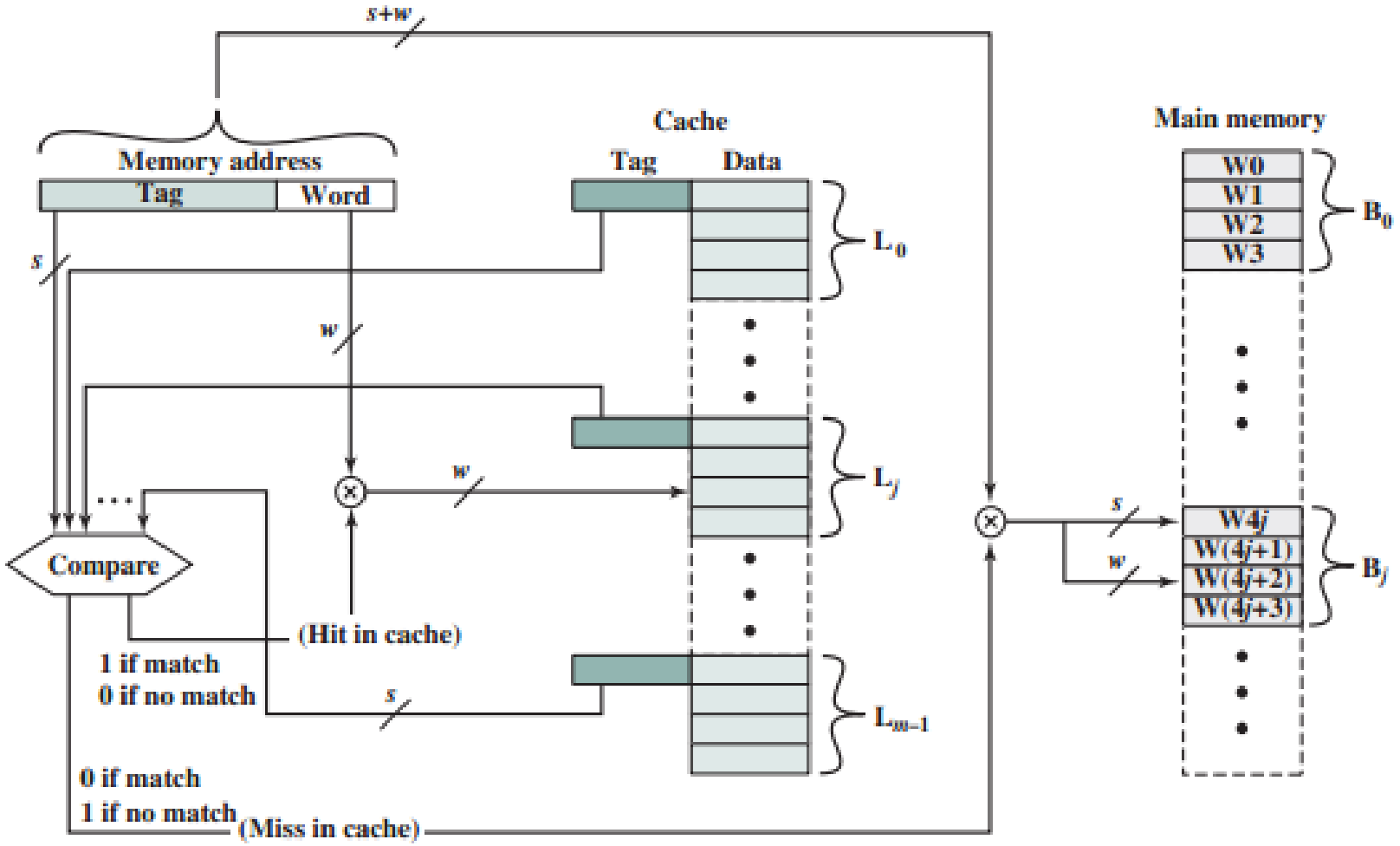


:: Mapping Function: Fully Associative Mapping

- Overcomes the **disadvantage** of direct mapping
- Permit each main memory block to be loaded into **any line of the cache**



: Fully Associative Mapping…



:: Mapping Function: Set Associative Mapping

- Compromise that exhibits the **strengths** of **both** the direct mapping and associative mapping reducing their disadvantages
- Cache consists of number **sets** (replace line from DM); each of them consisting of number of lines
- Relationship shown as:

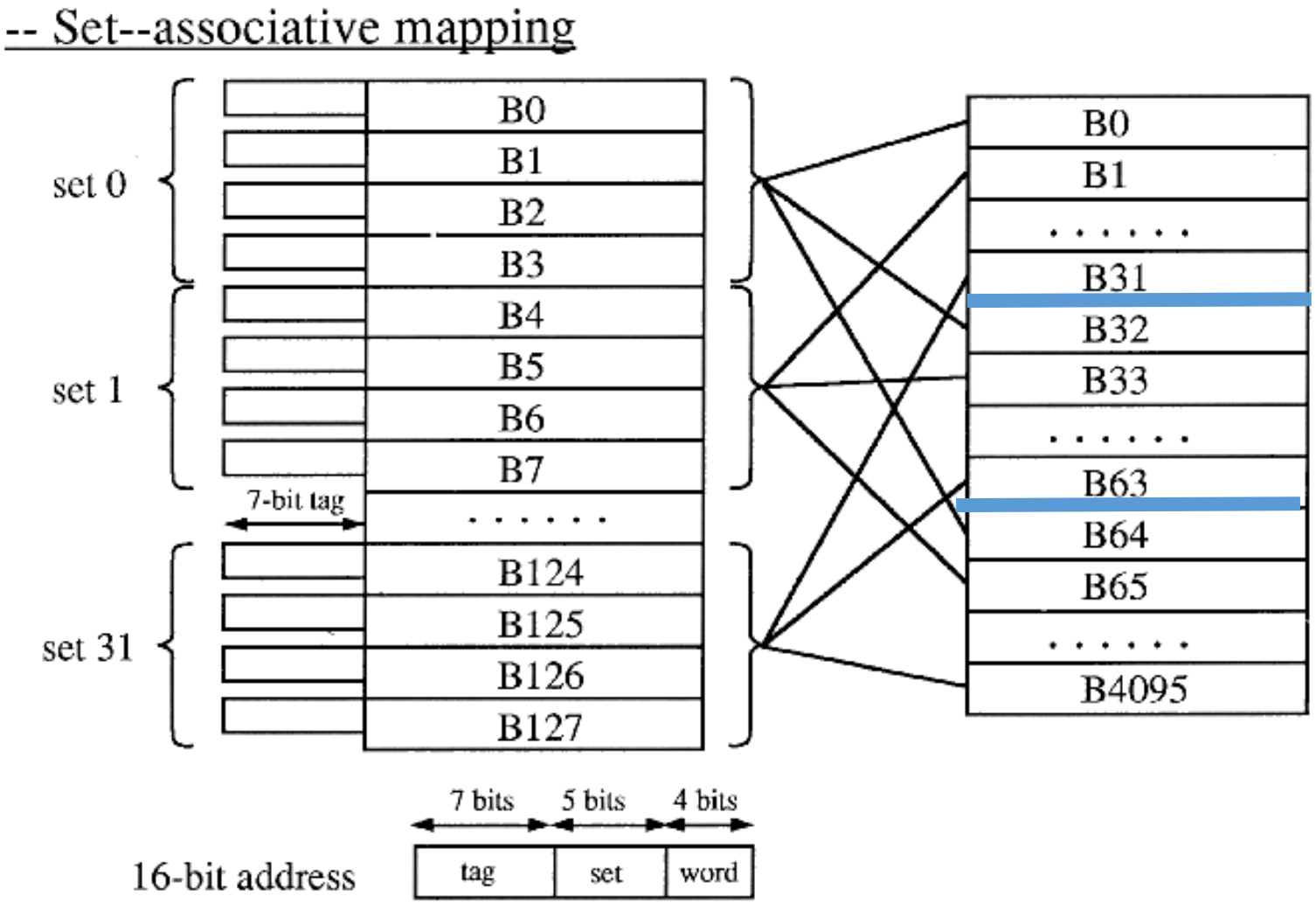
$$m = v \times k$$

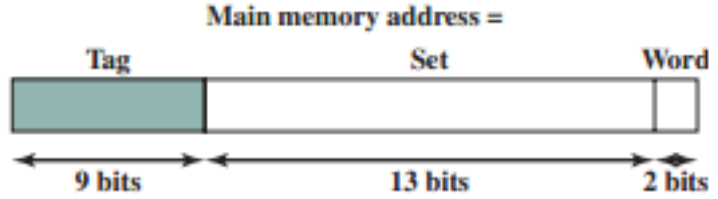
$$i = j \text{ modulo } v$$

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

Referred as k-way set associative mapping

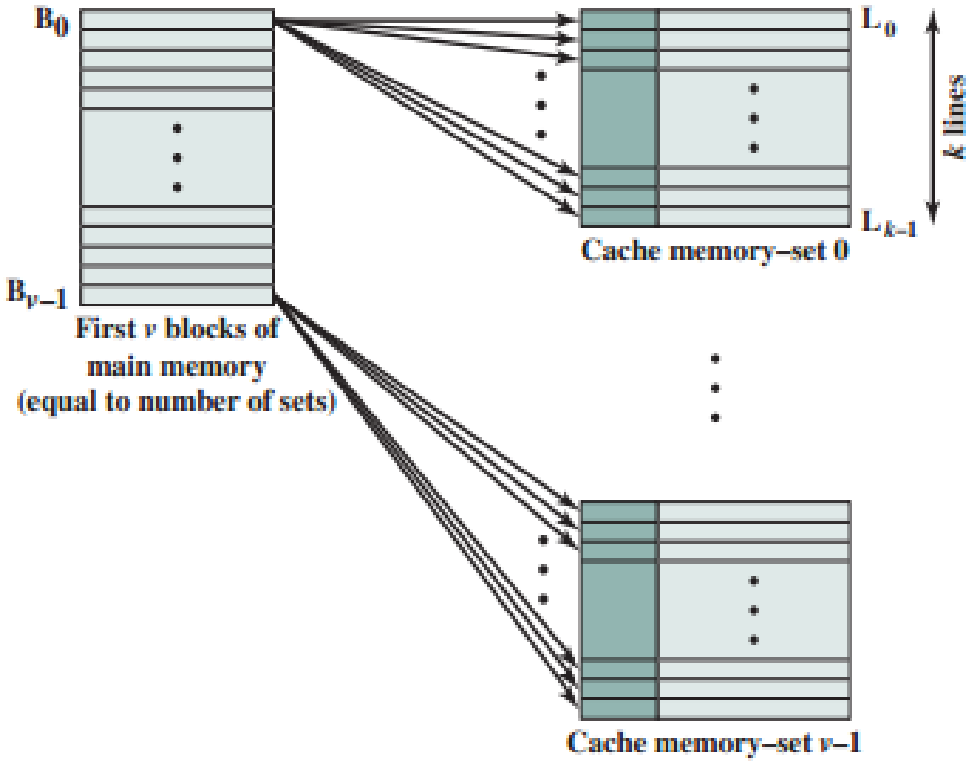
: Set Associative Mapping…





: Set Associative Mapping…

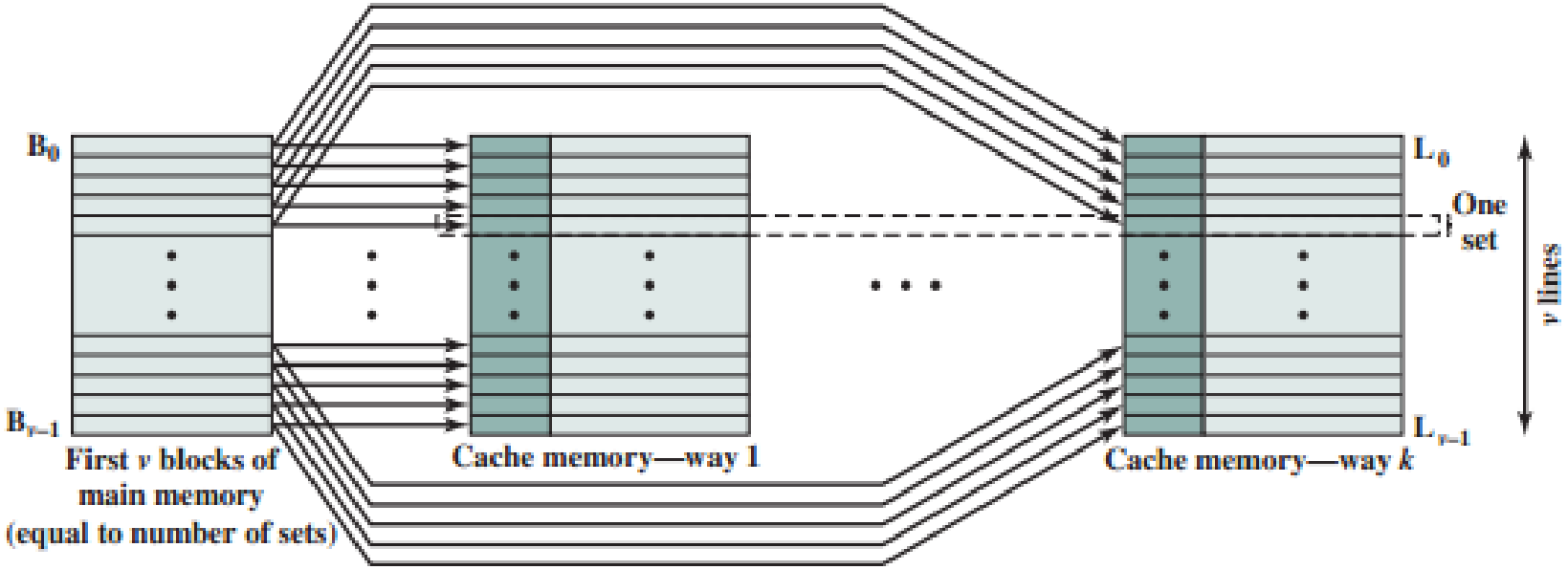
- Set associative cache physically implemented as v(=1) associative caches. (Assume, the cache is same size of a set, a block can occupy any line; max: v=1 and k=m)



(a) v associative–mapped caches

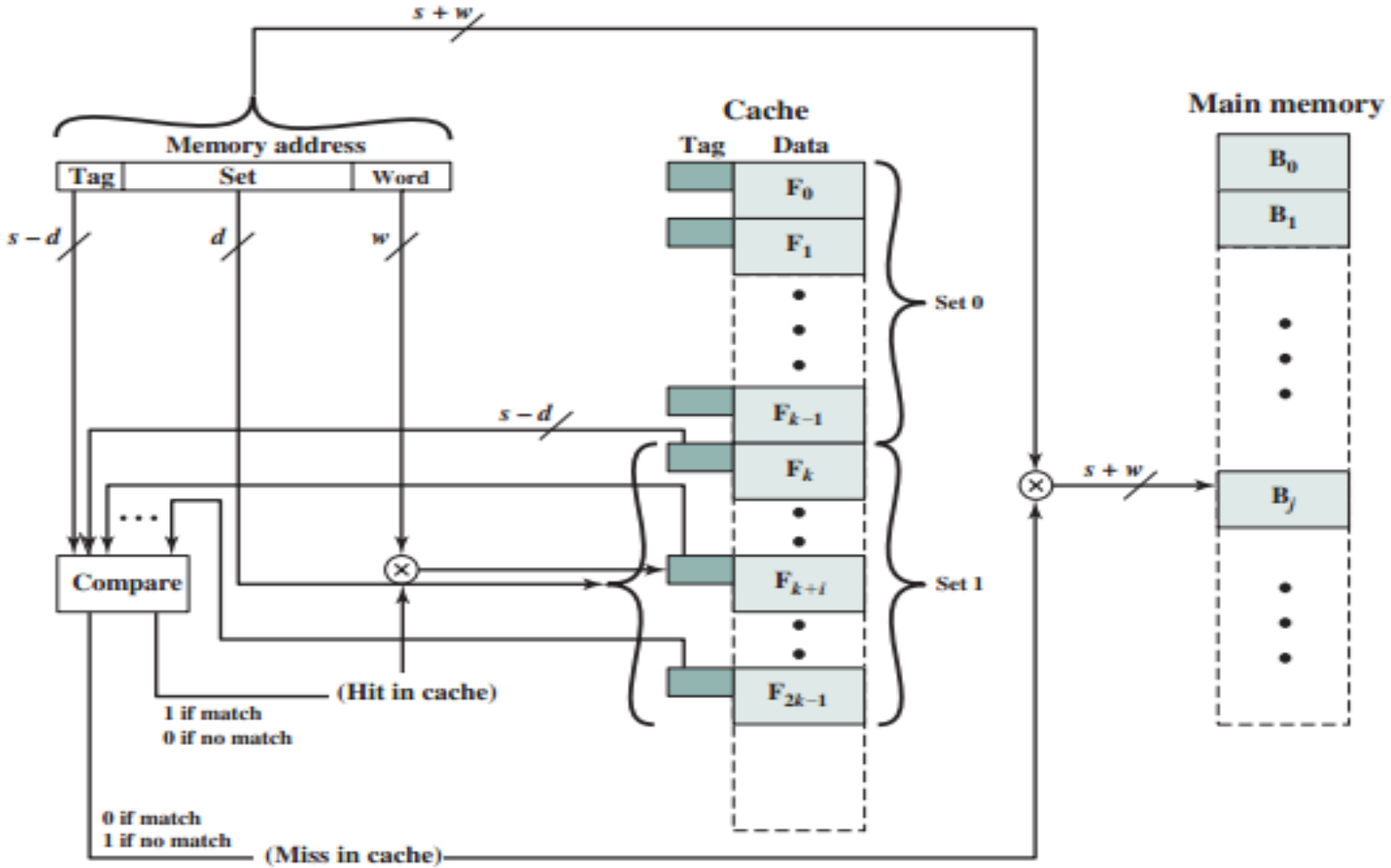
: Set Associative Mapping…

- Also set associative cache physically implemented as k(=1) direct mapping caches. (Assume, the set is same size of a line, a block can occupy a single fixed line; max: v=m and k=1)



(b) k direct–mapped caches

: Set Associative Mapping…



:: Replacement Algorithm

- **One cache is filled and another new block** is required to bring, one of the existing must be replaced
- In **direct mapping** only one choice is possible; but for **associative and set-associative mapping** choices are there – to choose a block among all replacement algorithm is needed.
  - **LRU (Least Recently Used)** – checking **USE bit**; recently used block set its USE bit=1, where others set it as 0
  - **FIFO (First In First Out)** – using circular buffer technique
  - **LFU (Least Frequently Used)** – using counter of each line
  - **Random** – slightly inferior/ lower to algorithms based on usage

:: Write Policy

- When the **resident block of the cache is replaced**, there are two cases:
  - **Old block is not altered** – fine; no update is required in main memory
  - **Old block is altered (even by a single bit)** – Oh My GOD!!! - whole corresponding main memory block must be updated.
- Two serious problems are there to consider:
  - If we **change a cache word**, corresponding **memory word is invalid**; or if we **change the memory word**, corresponding **cache word (if any) will be invalid**
  - If the **problem occurs with multiprocessor** and their local caches!!!

:: Write Policy

- Possible **techniques**:
  - **Write through**: all write operations are **made to main memory as well as to the cache** ensuring the main memory is always valid – generates **substantial memory traffic**
  - **Write back**: **Updates made only in the cache** rather than the main memory – **when a update is occurred in cache, a dirty/use bit is informed** – when that **block is replaced, the dirty/ use bit is checked**; **if it is 1**, also update the main memory **otherwise** leave as it is – **problem is when cache is updated, main memory is invalid**

Cache Coherency

If a multicore system is considered, though we use write through technique (where if any cache word is altered, immediately the memory will be updated to ensure the memory validation all the time), there is another issue that other caches may be invalid – remedy is cache coherency

- Active field of research (***)

Cache Coherency…

- Approaches of cache coherency:
  - **Bus watching with write through**: **Each cache controller monitor** the bus to check whether their shared data has been changed or not – it will invalidate the cache data if it finds memory data altered
  - **Hardware transparency**: **Additional h/w** that ensures that all caches are reflected if main memory is updated
  - **Non-cacheable memory**: **select a portion of the memory as non-cacheable** that caches are not able to fetch data from here – all the time they have to fetch data from memory or if there is any change in data, that memory portion will be directly updated – all vulnerable data will be stored here to avoid inconsistency

:: Line Size

- Line size should be considered to design cache memory because if we increase the line size, at first hit ratio will be increased (based on locality)– but after a certain amount of increase of line size, the hit ratio begins to be decreased.
- Other issues:
  - If we increase the line size, less amount of blocks will be fit in cache and as the number of blocks are very few, within very short time they will be overwritten again and again
  - As the line size is increased, the additional data with the required one will be less important

:: Number of Caches

- Though it is started with single cache, nowadays multiple caches are used following different design issues:
  - Multilevel Cache: on-chip cache & off-chip cache
  - Unified vs Split Cache: both data and instructions in single cache or split cache