Advanced Parallel Architecture

Lesson 4

Annalisa Massini - 2014/2015

Modules and connections

Components and connections

- A computer consists of a set of components or modules of three basic types (processor, memory, I/O) that communicate with each other
- In effect, a computer is a network of basic modules
- Thus, there must be paths for connecting the modules

- The collection of paths connecting the various modules is called the ***interconnection structure***

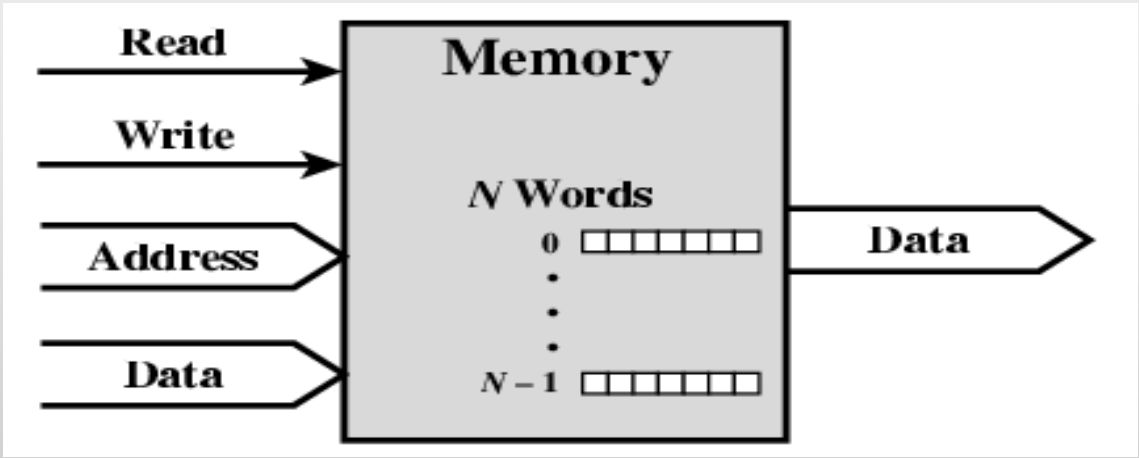
Components and connections

- The design of this structure depends on the exchanges that must be made among modules

- Different type of connection for different type of unit
  - Memory
  - Input/Output
  - CPU

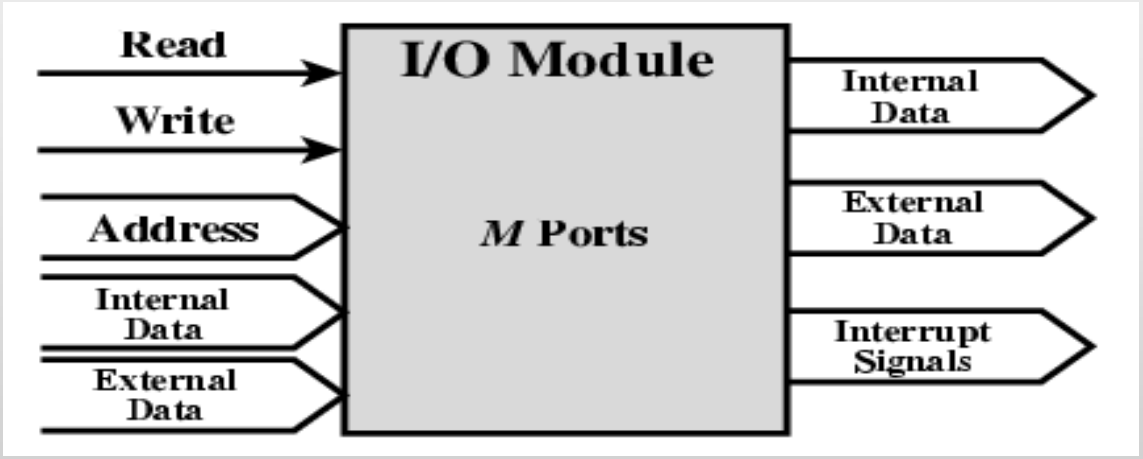
Memory Connection

- A memory module will consist of N words of equal length
- Each word is assigned a unique address (0, 1, . . . ,N – 1)
- Receives
  - data
  - addresses (of locations)
  - control signals (Read, Write, Timing)
- Sends
  - data



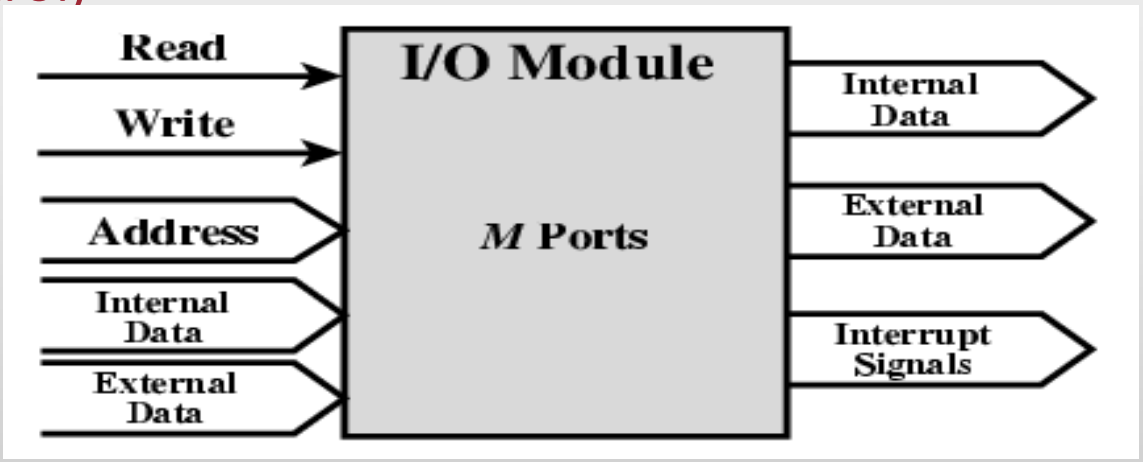
Input/Output Connection

- From an internal (to the computer system) point of view, **I/O is functionally similar to memory**
- An I/O module may control more than one external device by means of an interface (*port*) associated to an address (e.g., 0, 1, . . . ,M– 1)



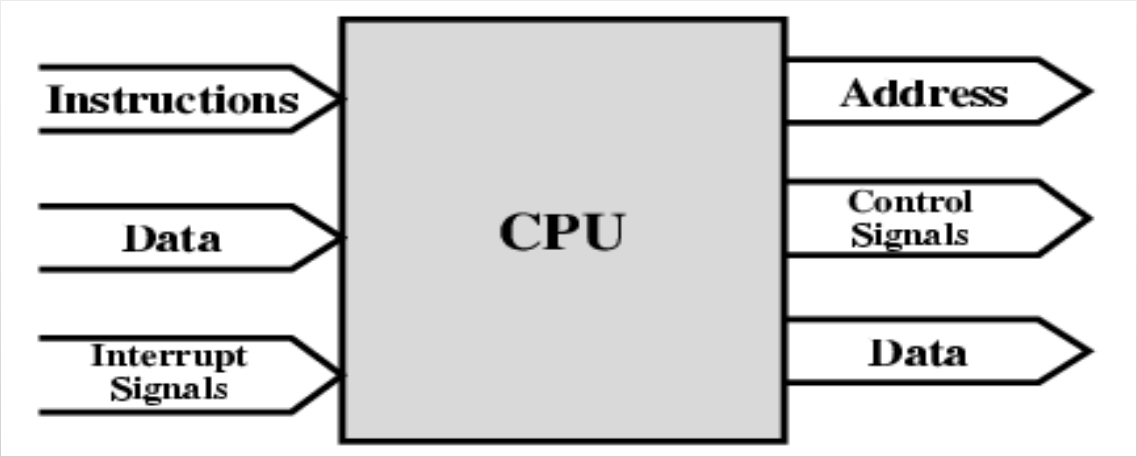
Input/Output Connection

- Receives
  - data from peripheral and from computer
  - control signals (read/write)
  - addresses from computer (e.g. port number to identify peripheral)
- Sends
  - data to peripheral and to computer
  - interrupt signals (control)



CPU Connection

- Receives
  - instruction
  - Data
  - interrupts
- Sends
  - data (after processing)
  - control signals to other units



Components and connections

- Hence, the interconnection structure must support the following types of transfers:
  - **Memory to processor:** The processor reads an instruction or a unit of data from memory
  - **Processor to memory:** The processor writes a unit of data to memory
  - **I/O to processor:**The processor reads data from an I/O device via an I/O module
  - **Processor to I/O:** The processor sends data to the I/O device
  - **I/O to or from memory:** For these two cases, an I/O module is allowed to exchange data directly with memory, without going through the processor, using direct memory access (DMA)

Bus

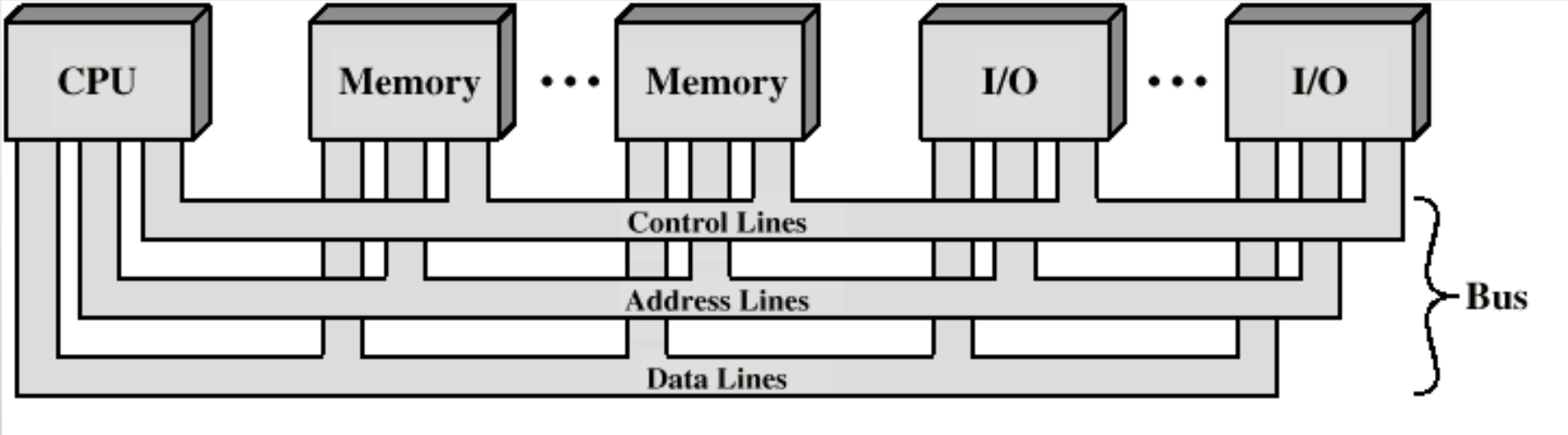
Buses

- There are a number of possible interconnection systems
- Single and multiple BUS structures are most common

- A bus is a communication pathway connecting two or more devices
- Usually broadcast
- Often grouped
  - A number of channels in one bus
  - e.g. 32 bit data bus is 32 separate single bit channels
- Power lines may not be shown

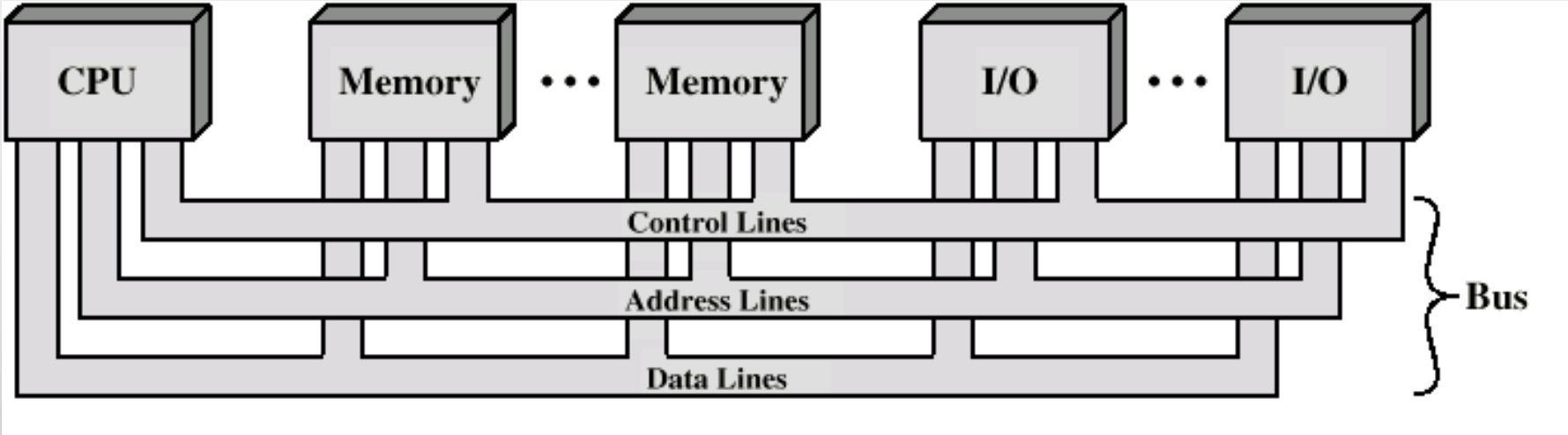
Bus structure

- **Data Bus** - Carries data
  - there is no difference between “data” and “instruction”
  - Width is a key determinant of performance (8, 16, 32, 64 bit)



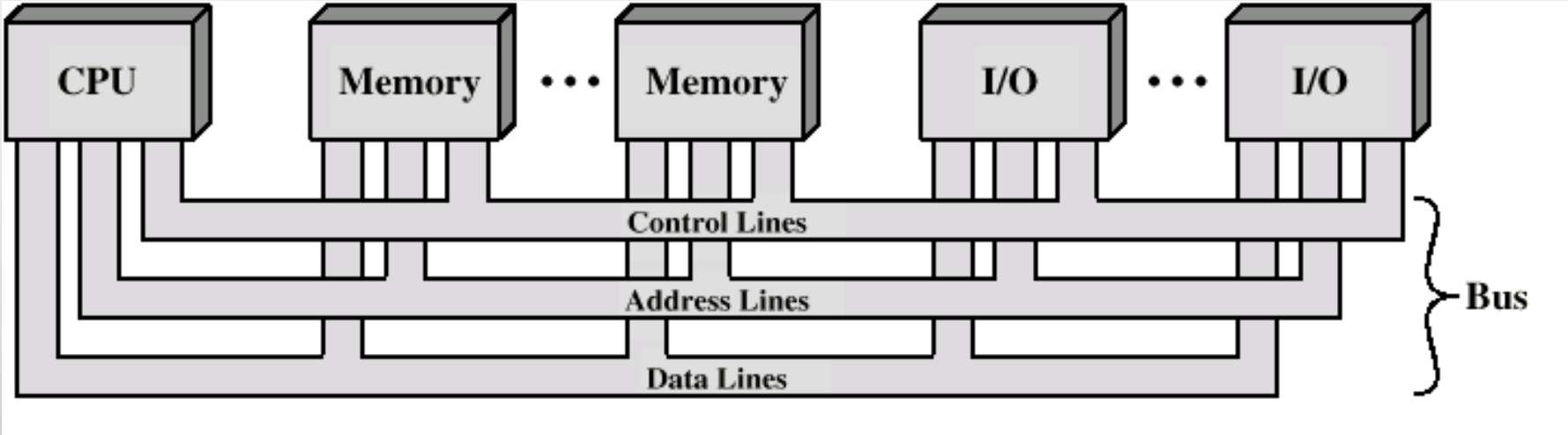
Bus structure

- **Address Bus** - Identify the source or destination of data
  - e.g. CPU needs to read an instruction (data) from a given location in memory
  - Bus width determines maximum memory capacity of system

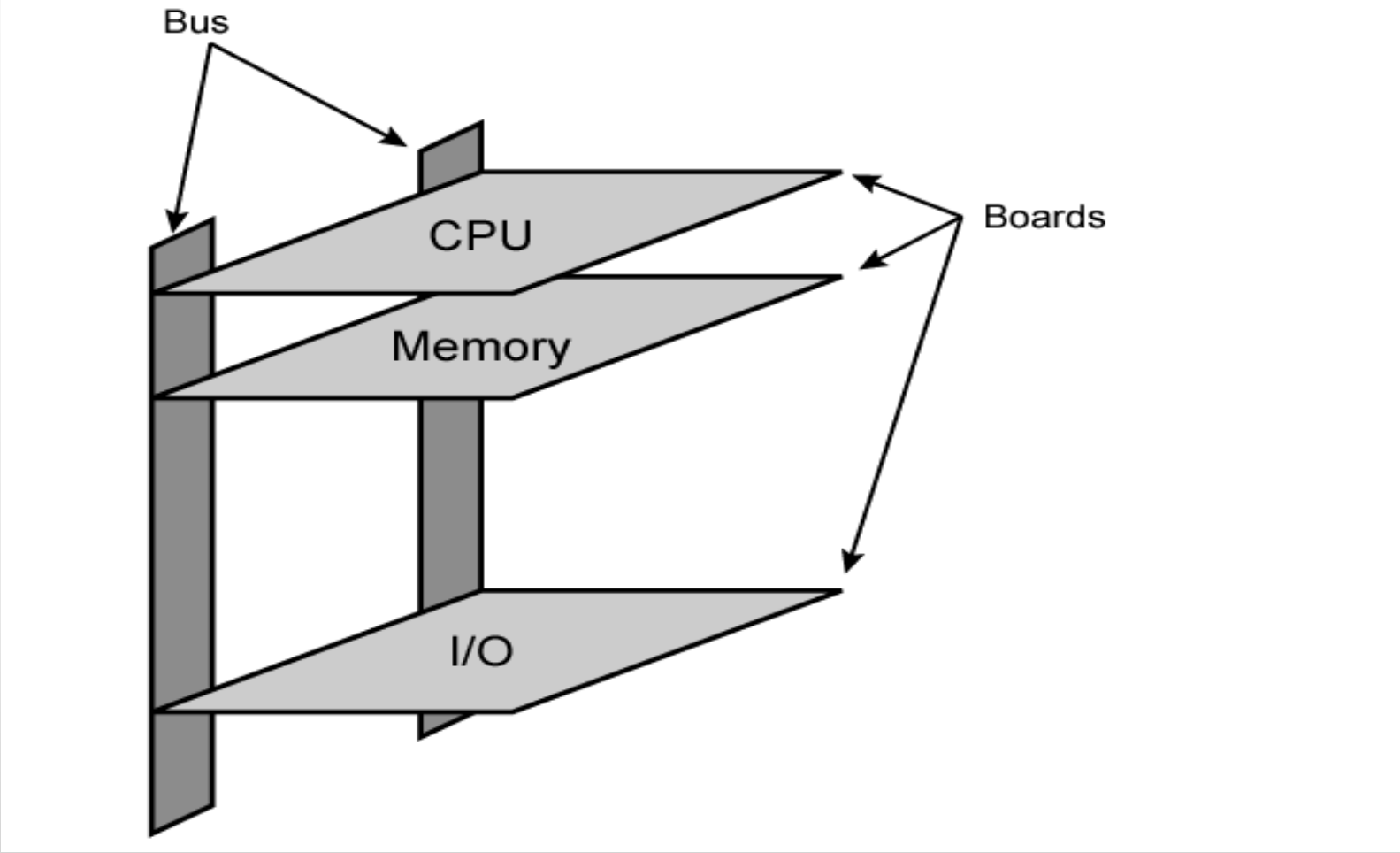


Bus structure

- **Control Bus** - Control and timing information
  - Memory read/write signal
  - Interrupt request
  - Clock signals

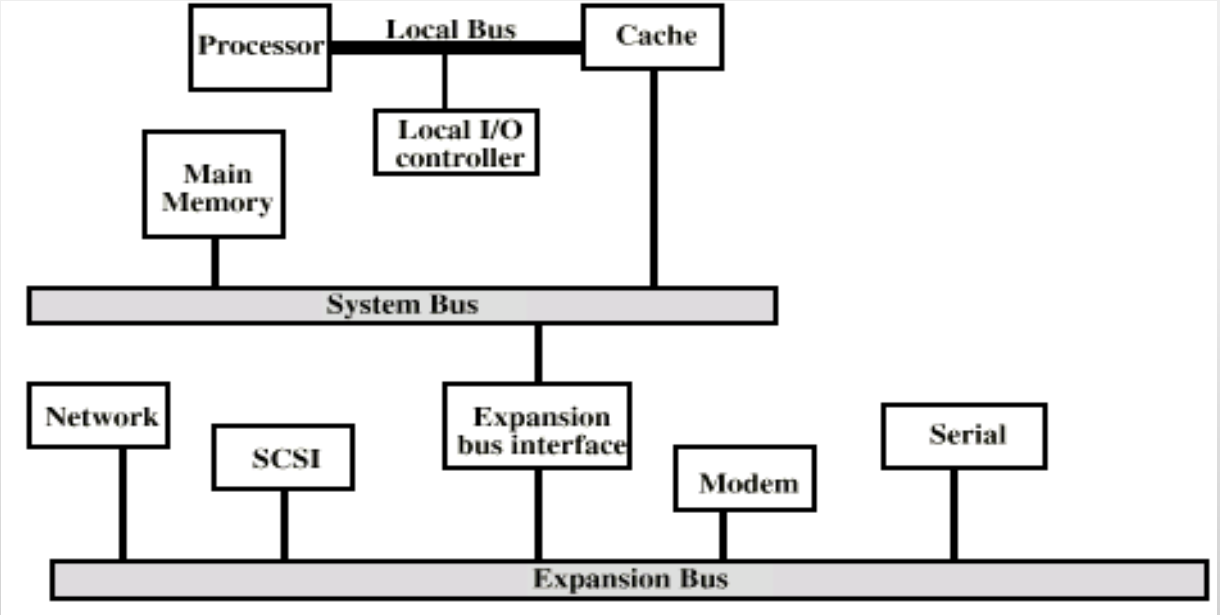


Physical Realization of Bus Architecture

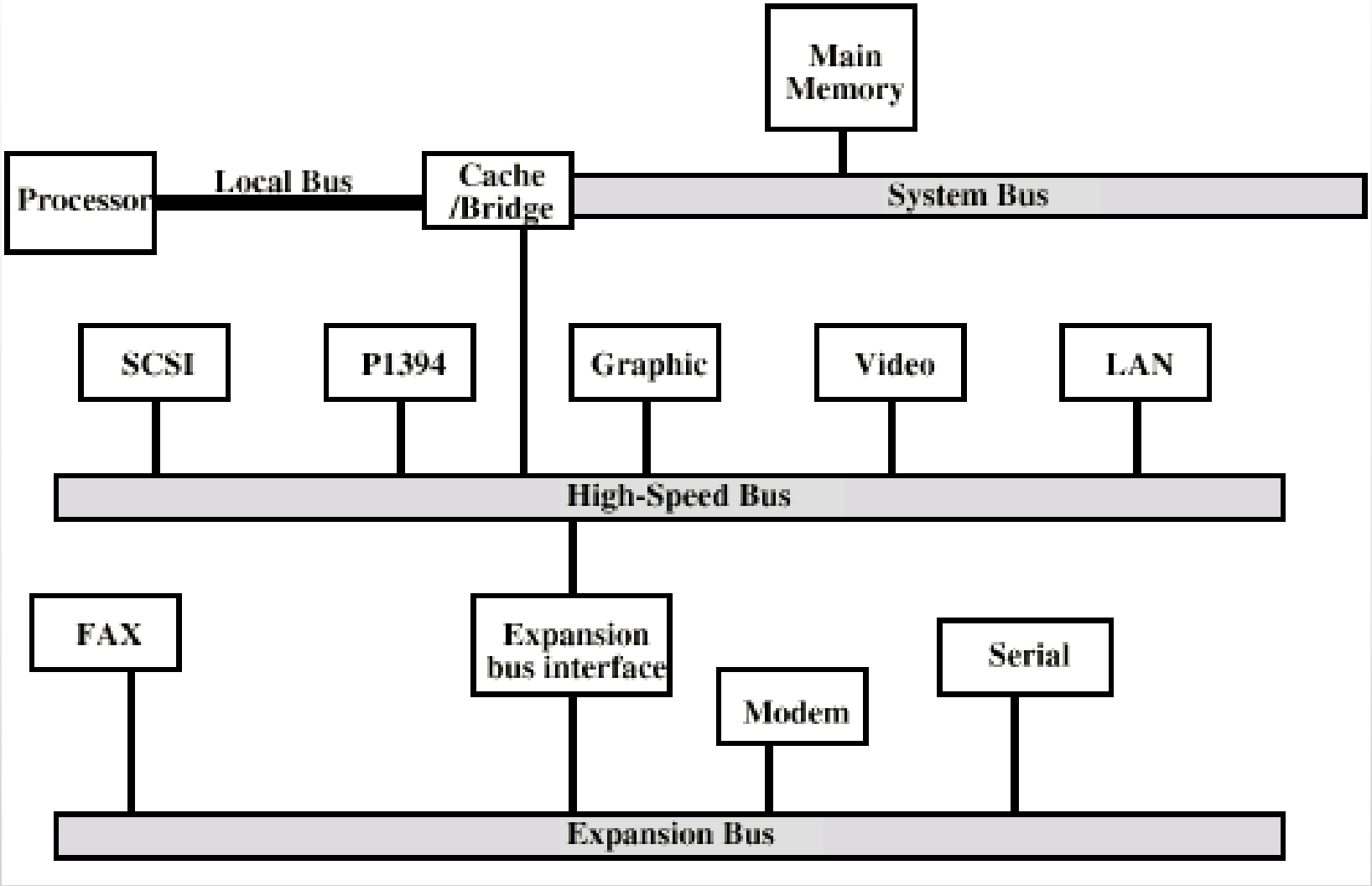


Single Bus Problems

- Lots of devices on one bus leads to:
  - Propagation delays
    - Co-ordination of bus use can affect performance (long data paths)
    - If aggregate data transfer approaches bus capacity
- Most systems use multiple buses to overcome these problems



High Performance Bus



Memory Hierarchy

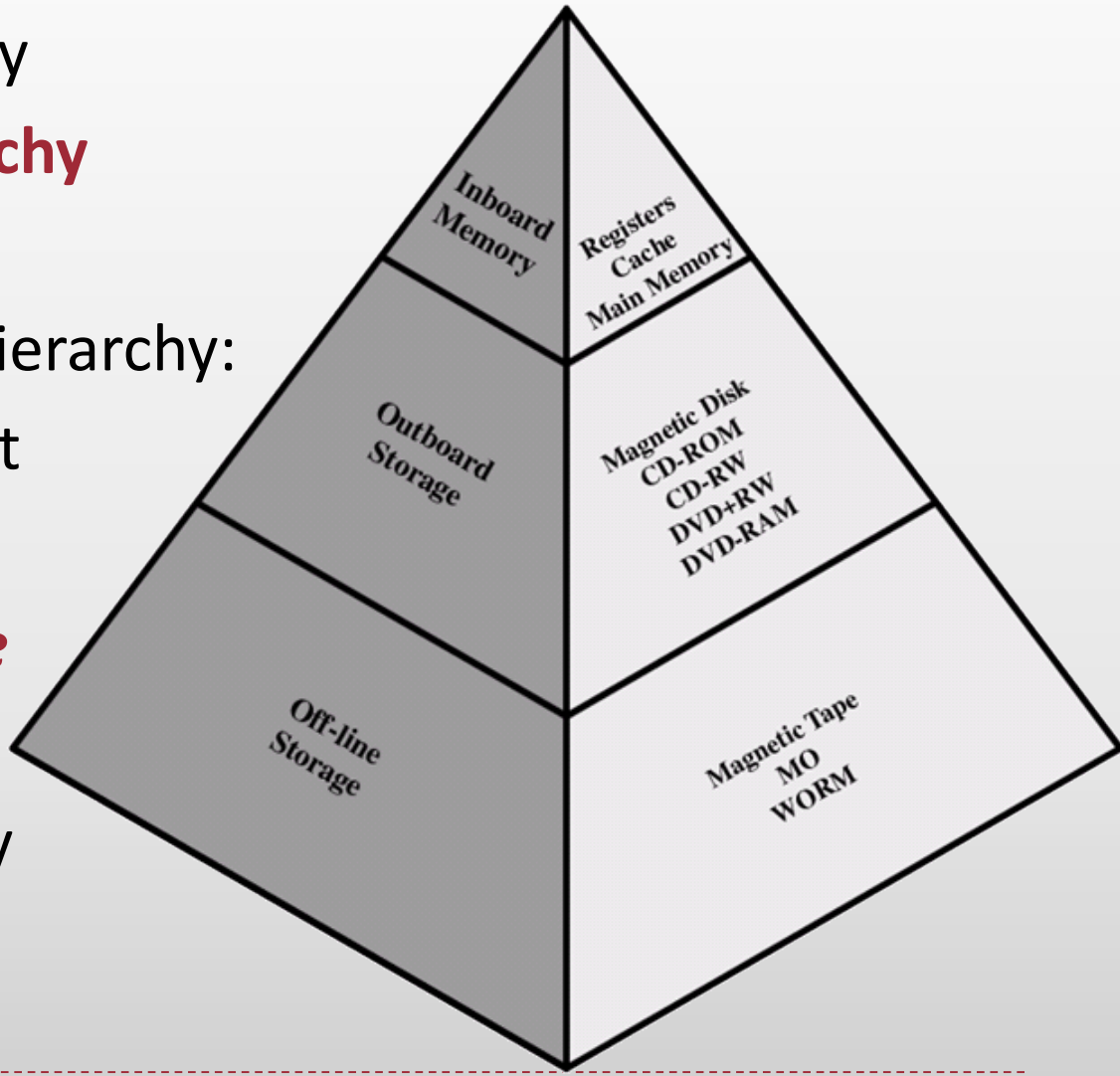
Memory Hierarchy

- the three key characteristics of memory:
  - Capacity
  - Access time
  - Cost
- A variety of technologies are used to implement memory systems.
- The following relationships hold:
  - Faster access time, greater cost per bit
  - Greater capacity, smaller cost per bit
  - Greater capacity, slower access time

Memory Hierarchy

The solution is to employ

a **memory hierarchy**



As one goes down the hierarchy:

- Decreasing ***cost*** per bit
- Increasing ***capacity***
- Increasing ***access time***
- Decreasing ***frequency***

of access of the memory

by the processor

Memory Hierarchy

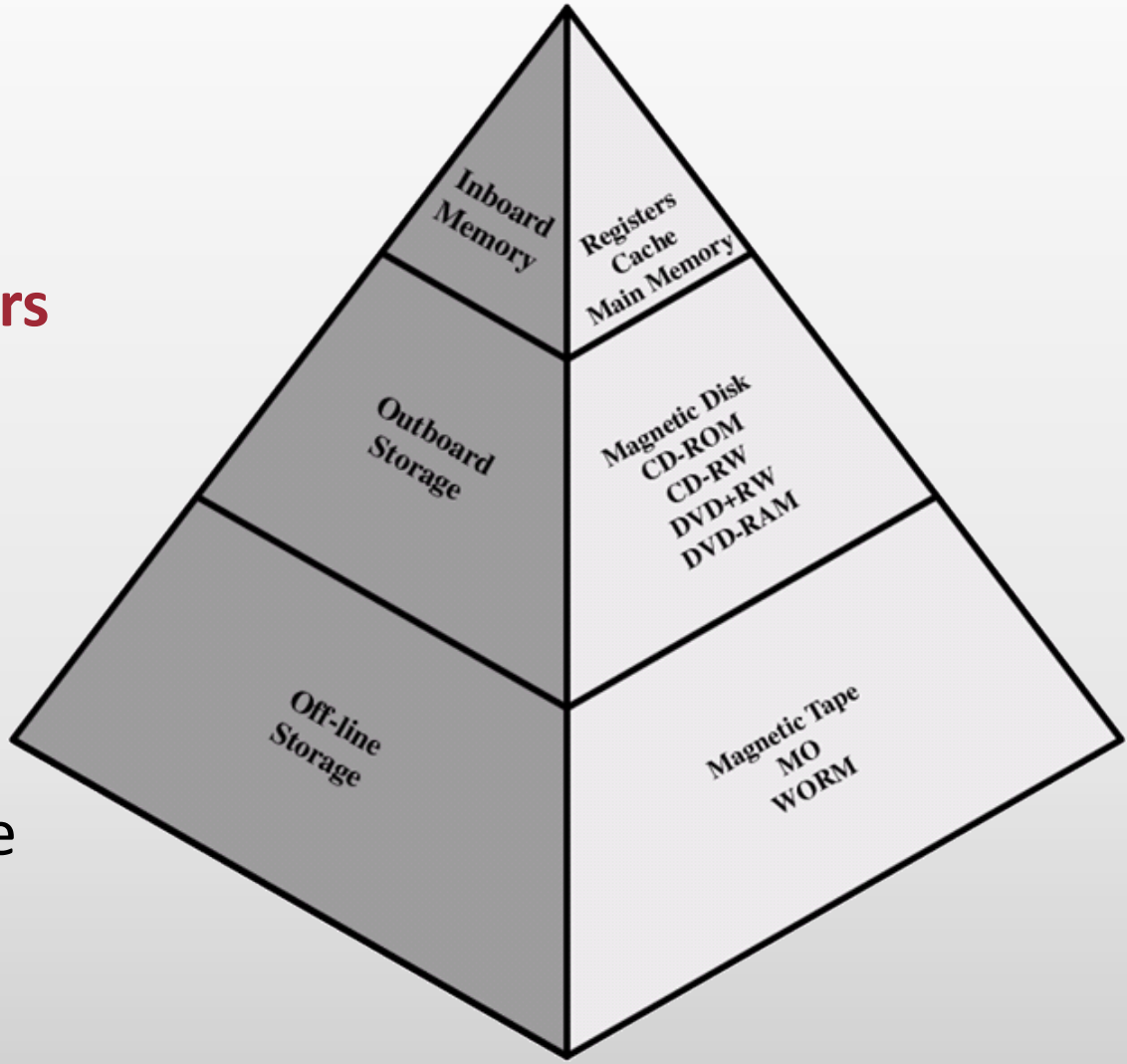
- During the execution of a program, memory references for instructions and data tend to cluster: **locality of reference**
- Programs typically contain a number of iterative loops and subroutines (repeated references to a small set of instructions)
- Similarly, operations on tables and arrays involve access to a clustered set of data words

- It is possible to **organize data across the hierarchy** such that the percentage of accesses to each successively lower level is substantially less than that of the level above

Memory Hierarchy

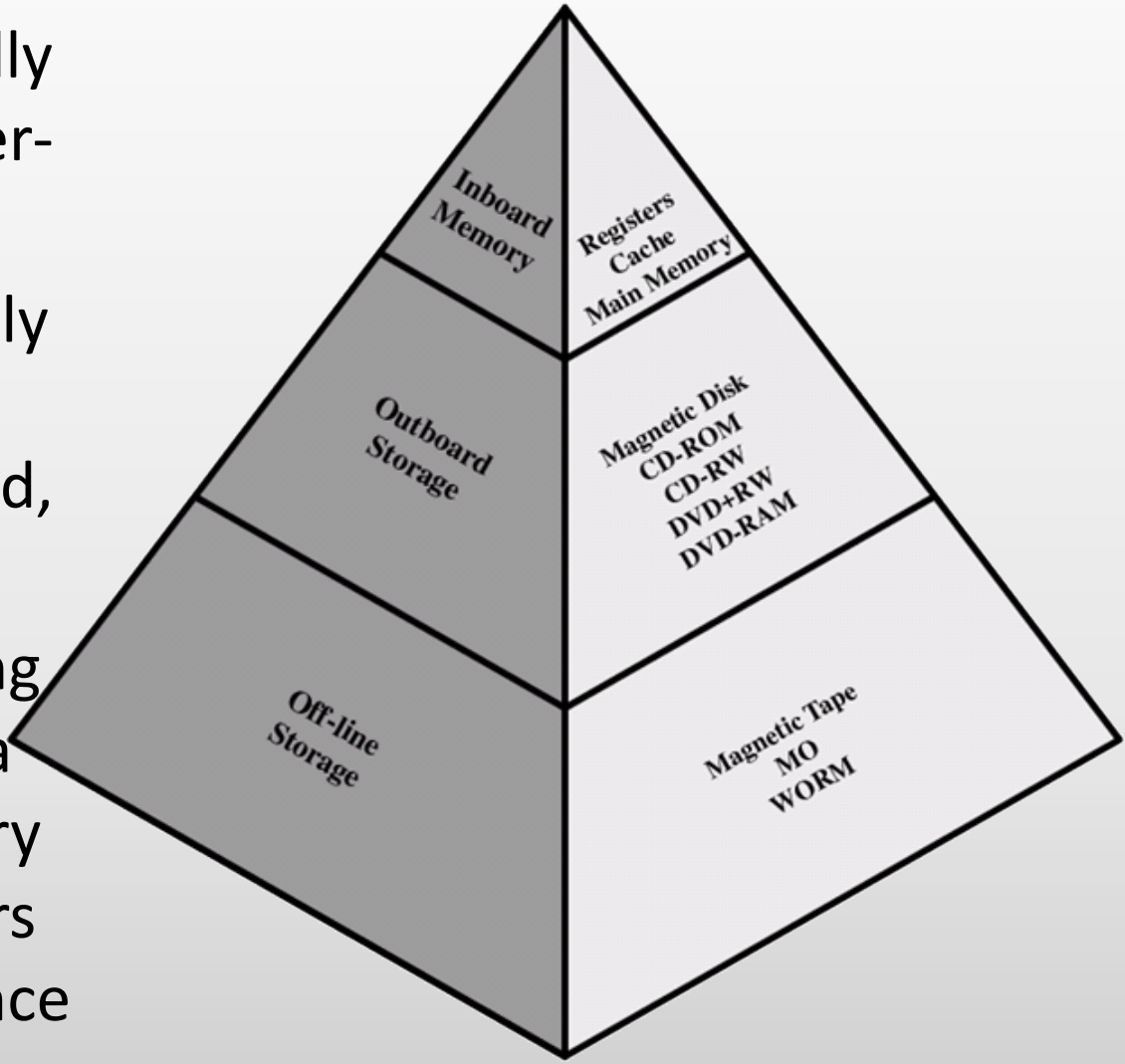
- The fastest, smallest, and most expensive type of memory consists of the **registers** internal to the processor

- **Main memory** is the principal internal memory system of the computer



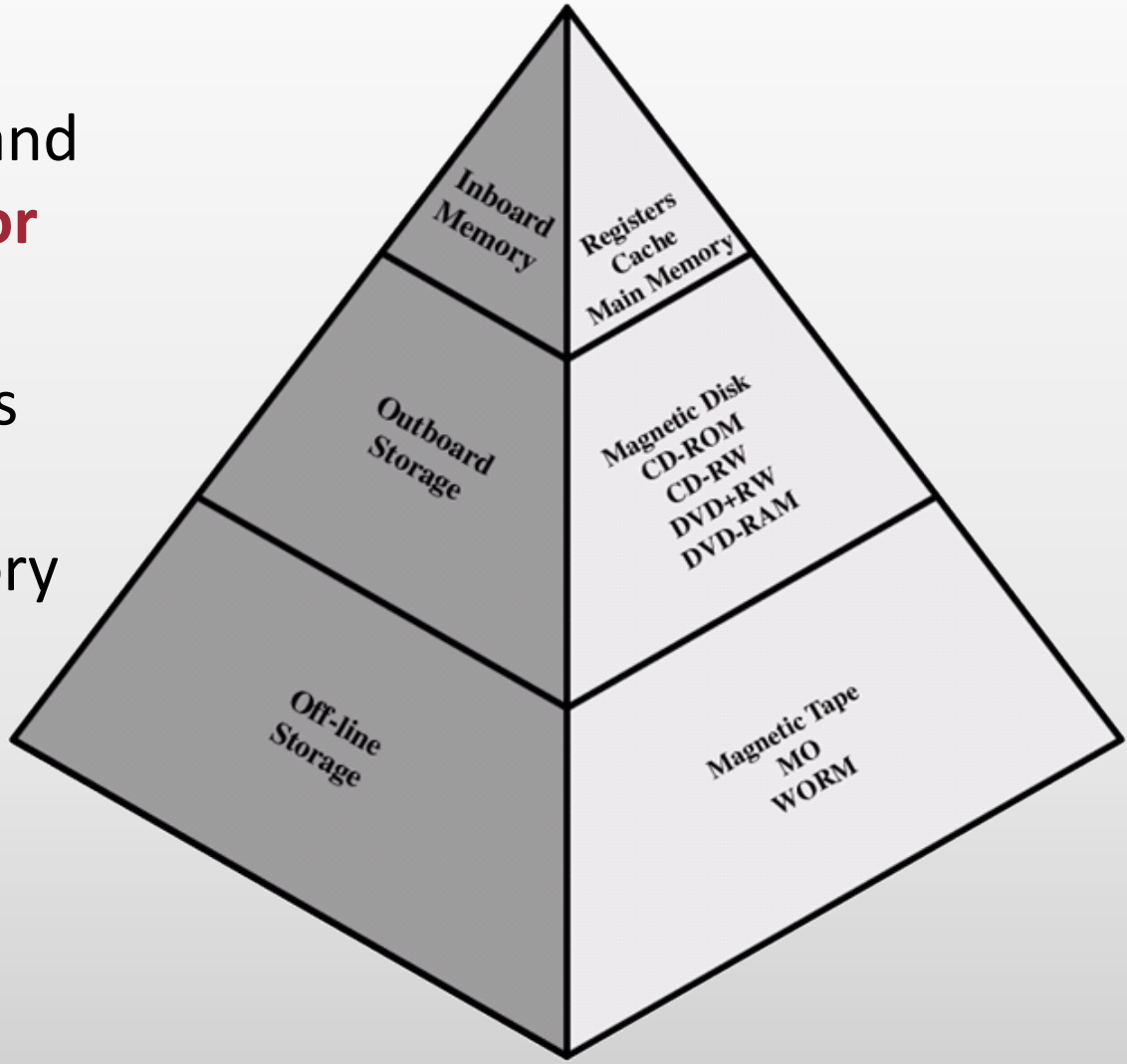
Memory Hierarchy

- 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



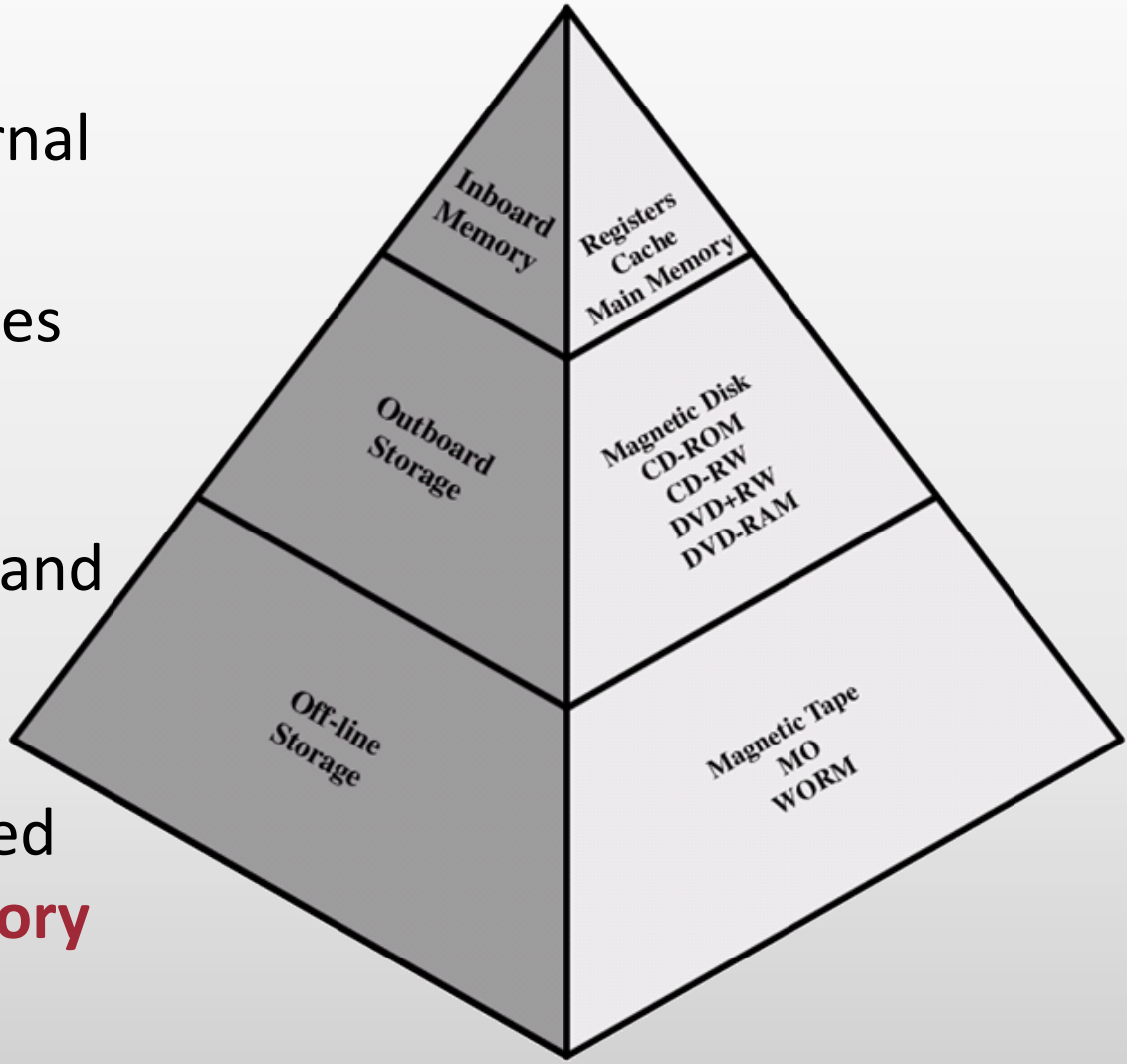
Memory Hierarchy

- These three forms of memory are **volatile** and employ **semiconductor technology**
- The use of three levels exploits the fact that semiconductor memory comes in a variety of types, which differ in **speed and cost**



Memory Hierarchy

- Data are stored more **permanently** on external mass storage devices
- The most common ones are hard disk and removable media (removable magnetic and optical storage)
- External, nonvolatile memory is also referred to as **secondary memory**
- or **auxiliary memory**



Characteristics of memory systems

- Location
  - Internal (e.g. processor registers, main memory, cache)
  - External (e.g. optical disks, magnetic disks, tapes)
- Capacity
  - Word size
  - Number of words, Number of bytes
- Unit of transfer
  - Word
  - Block

Characteristics

- Access method
  - Sequential (e.g. tape)
    - Start at the beginning and read through in order
    - Access time depends on location of data and previous location
  - Direct (e.g. disk)
    - Individual blocks have unique address
    - Access time depends on location and previous location
  - Random (e.g. RAM)
    - Individual addresses identify locations exactly
    - Access time is independent of location or previous access
  - Associative (e.g. Cache)
    - Data is located by a comparison with contents of a portion of the store
    - Access time is independent of location or previous access

Characteristics

- 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

Characteristics

- Physical type
  - Semiconductor (RAM)
  - Magnetic (Disk & Tape)
  - Optical (CD & DVD)
- Physical characteristics
  - Volatile/nonvolatile
  - Erasable/nonerasable
  - Power consumption
- Organization
  - Physical arrangement of bits into words
  - Memory modules

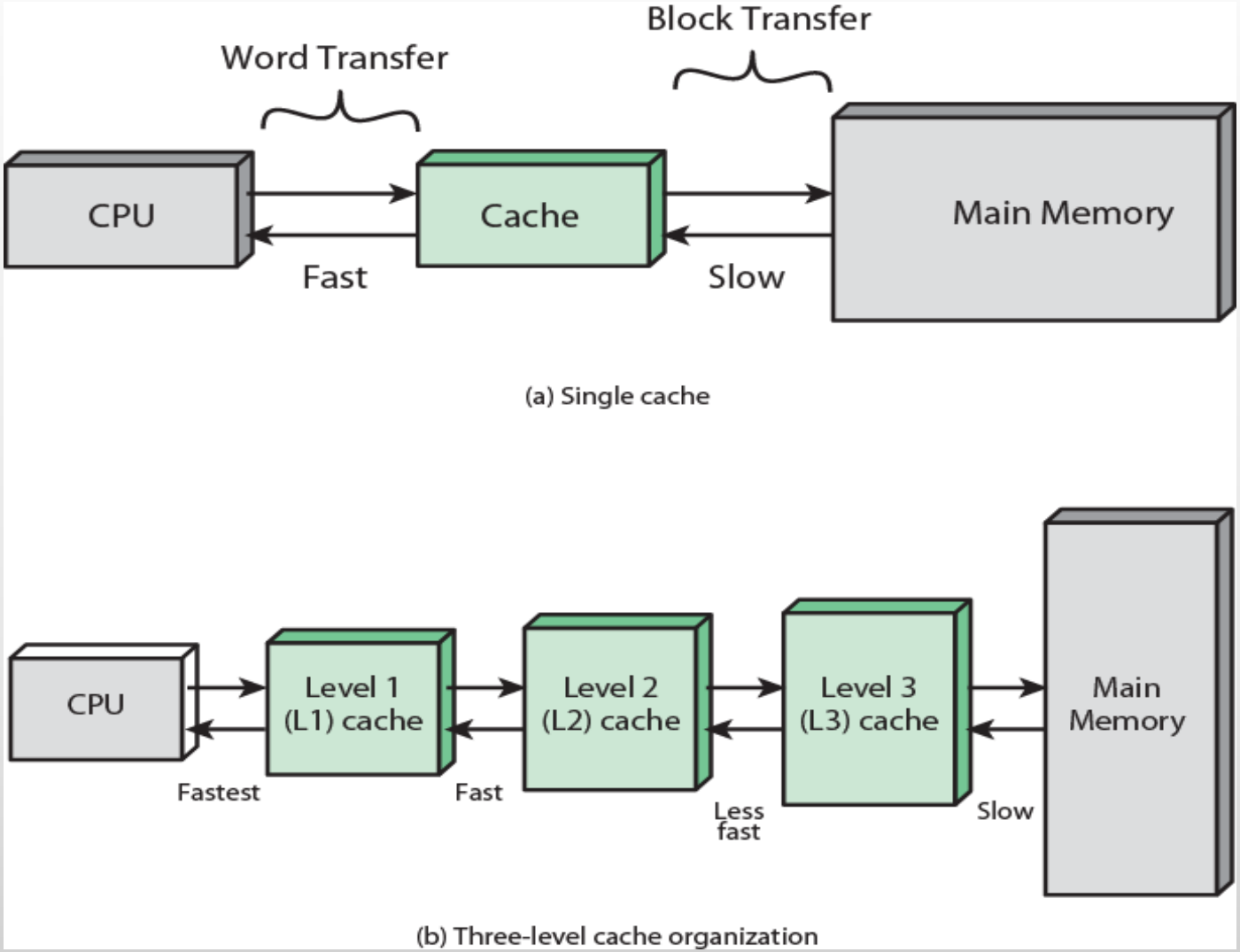
Hierarchy List

- Registers
- L1 Cache
- L2 Cache
- L3 Cache
- Main memory
- Disk cache
- Disk
- Optical
- Tape

Cache Memory

Cache and Main Memory

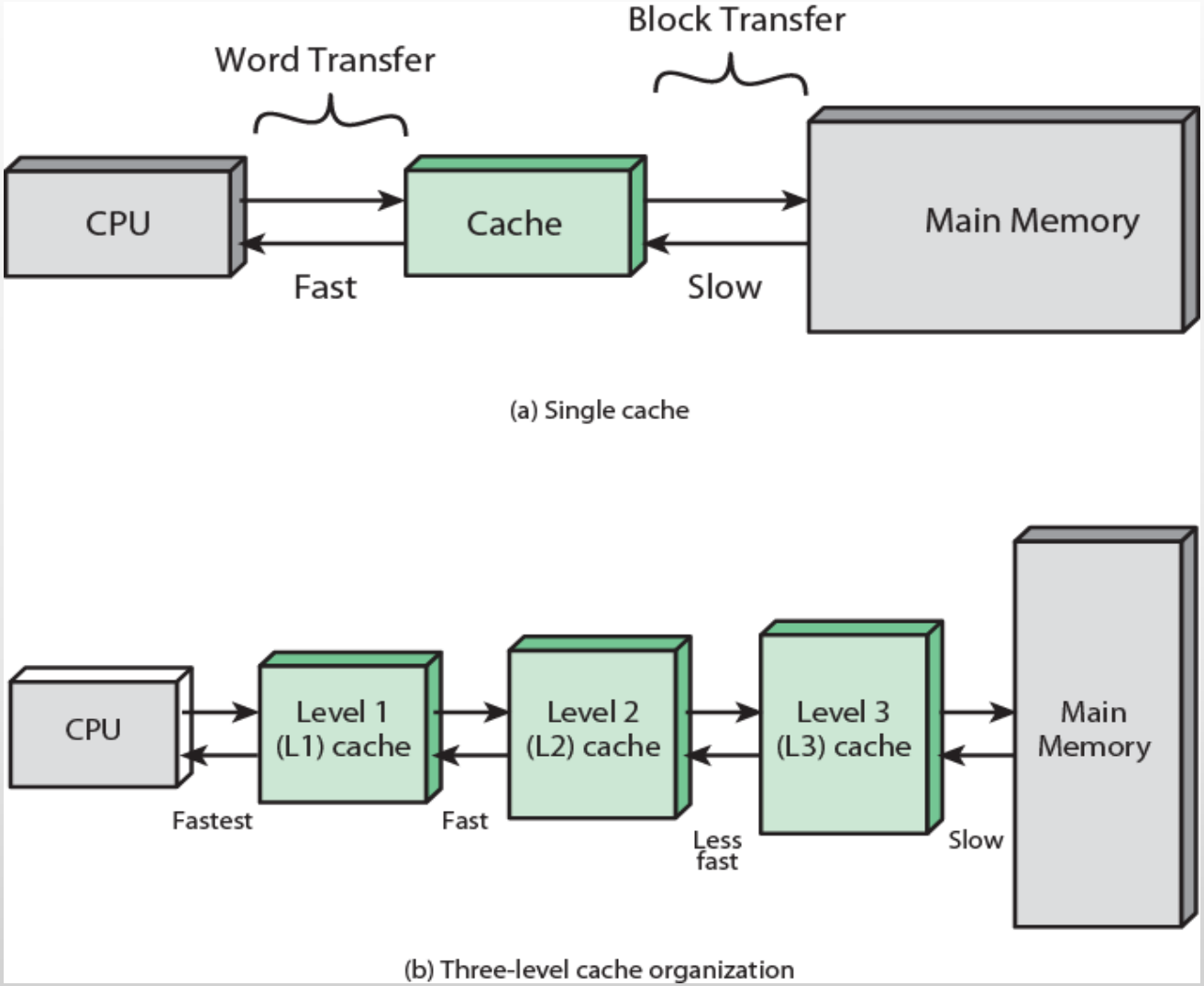
- A relatively *large* and *slow* **main memory** together with a smaller, faster **cache memory**
- The cache contains a copy of portions of main memory



Cache and Main Memory

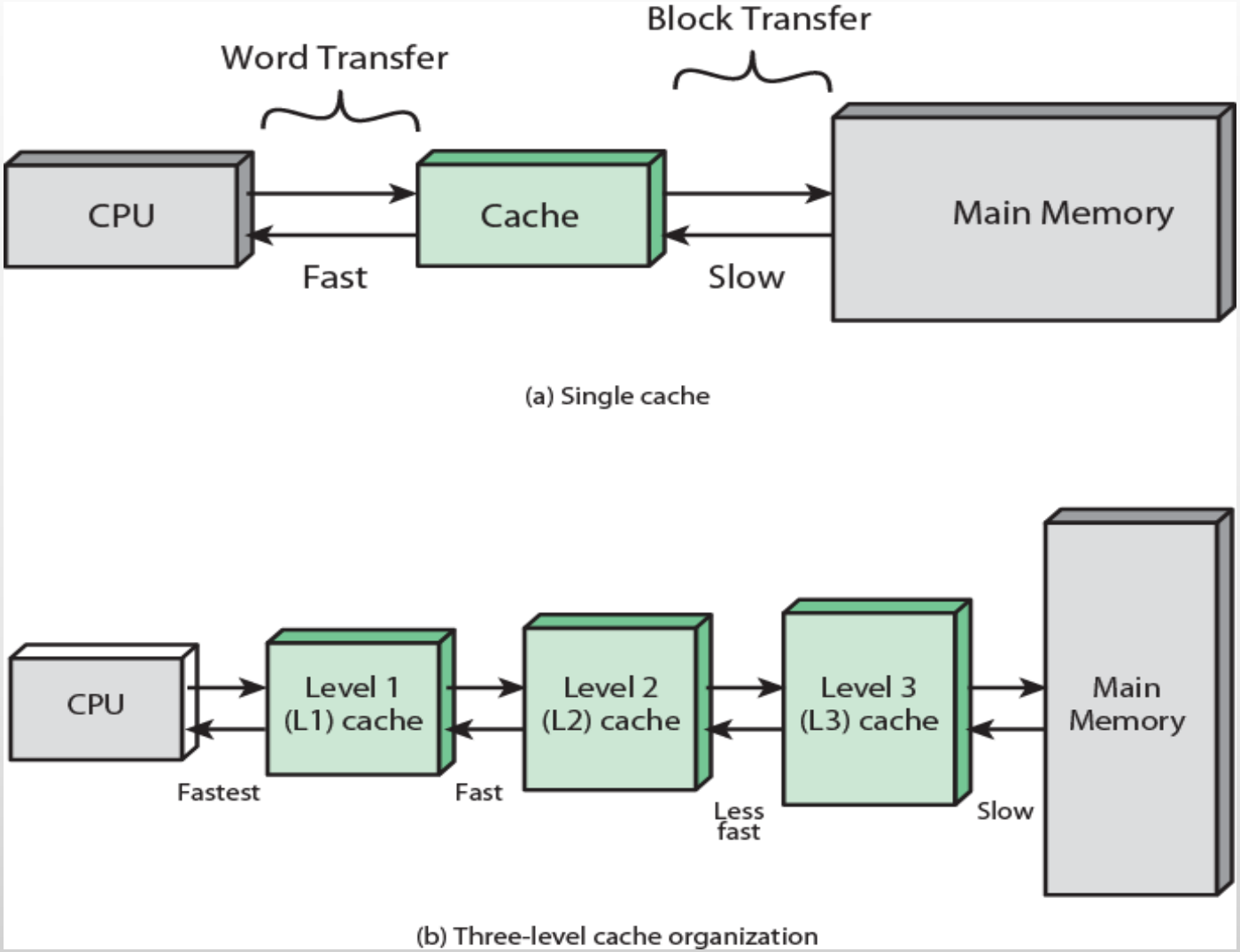
- ***Multiple levels of cache***

- The **L2 cache** is slower and typically larger than the **L1 cache**
- The **L3 cache** is slower and typically larger than the **L2 cache**



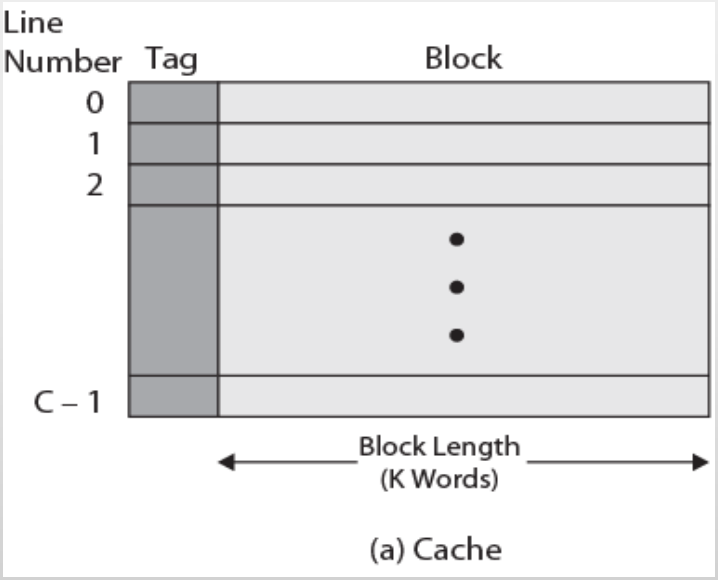
Cache and Main Memory

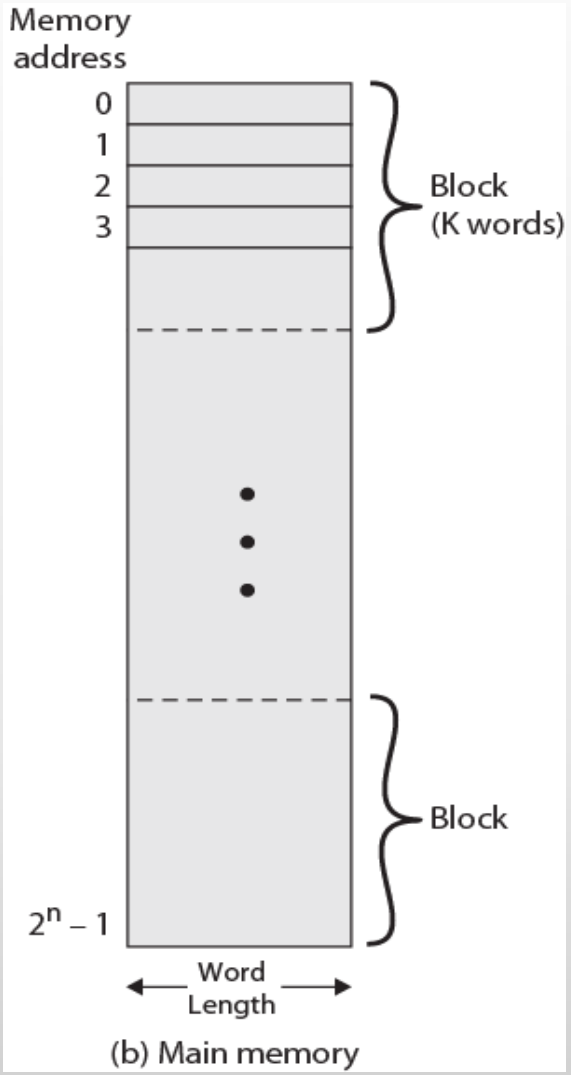
- When the processor attempts to read a word of memory, a **check** is made to determine **if the word is in the cache**



Cache/Main Memory Structure

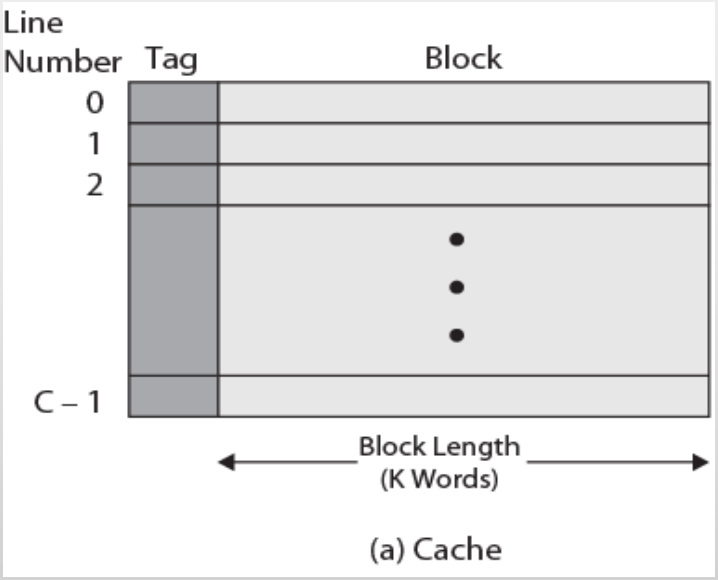
- Main memory - up to 2^n *addressable words*
- Each word - *unique n-bit address*
- Main memory is considered to consist of
- M blocks of K words each → **$M=2^n/K$ blocks**

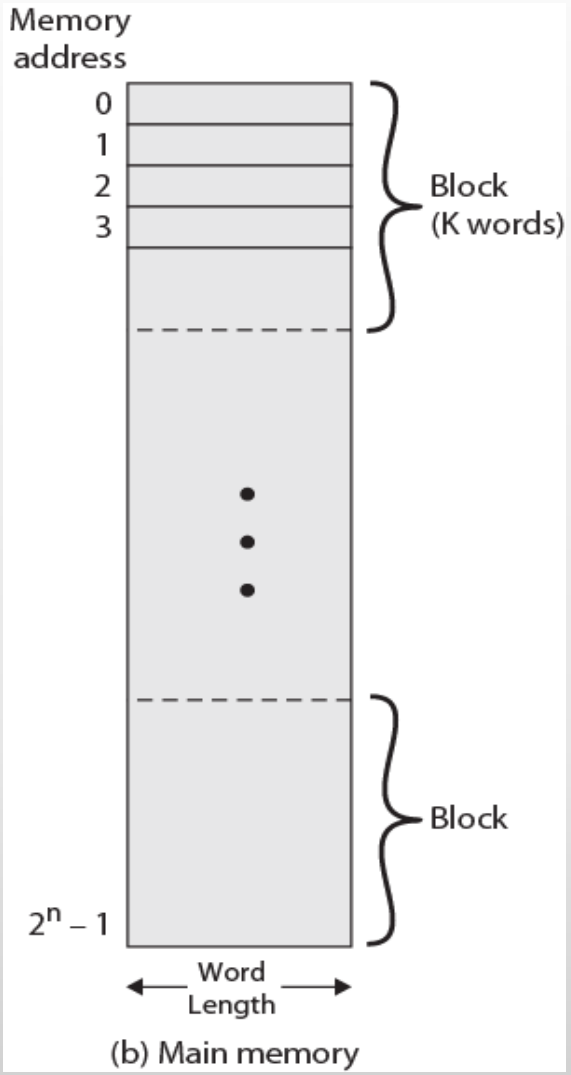




Cache/Main Memory Structure

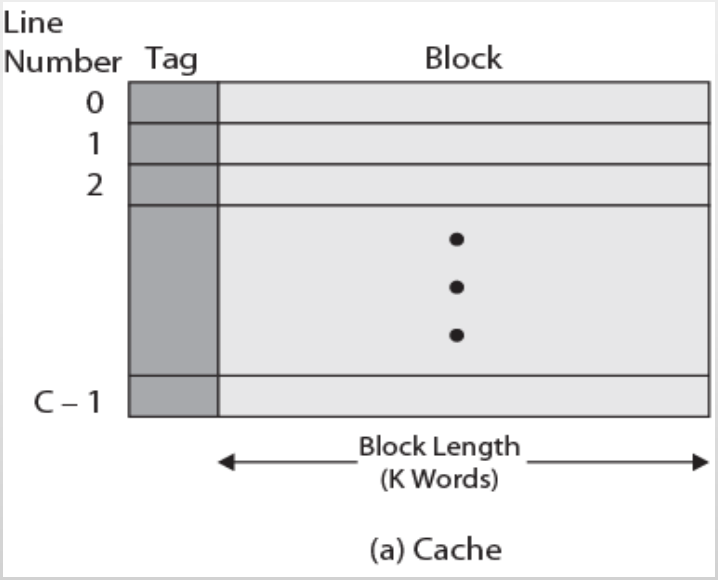
- The cache consists of **C lines**
- Each line contains **K words**, plus a **tag**
- Each **line of cache** corresponds to a **block in main memory**

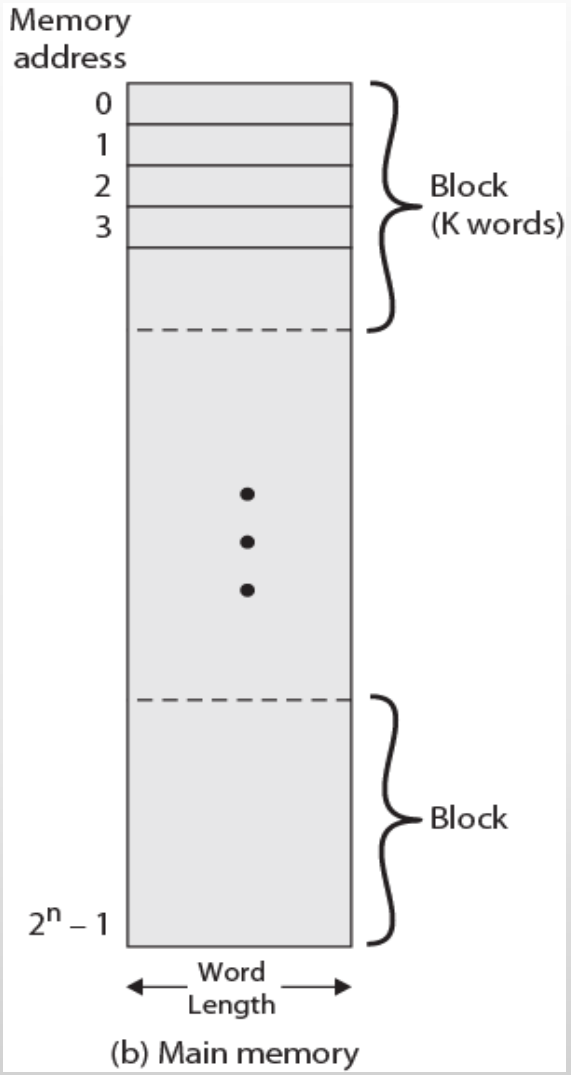




Cache/Main Memory Structure

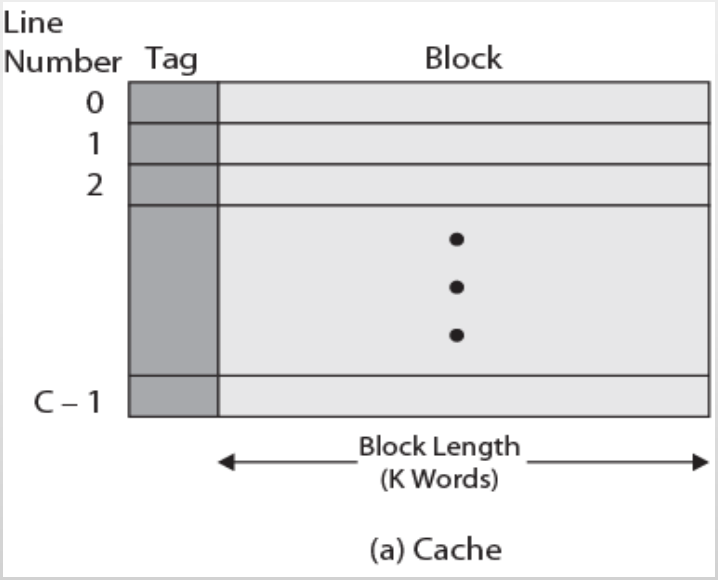
- The *number of lines* is *considerably less* than the *number of main memory blocks*
- At any time, some subset of the blocks of memory resides in lines in the cache

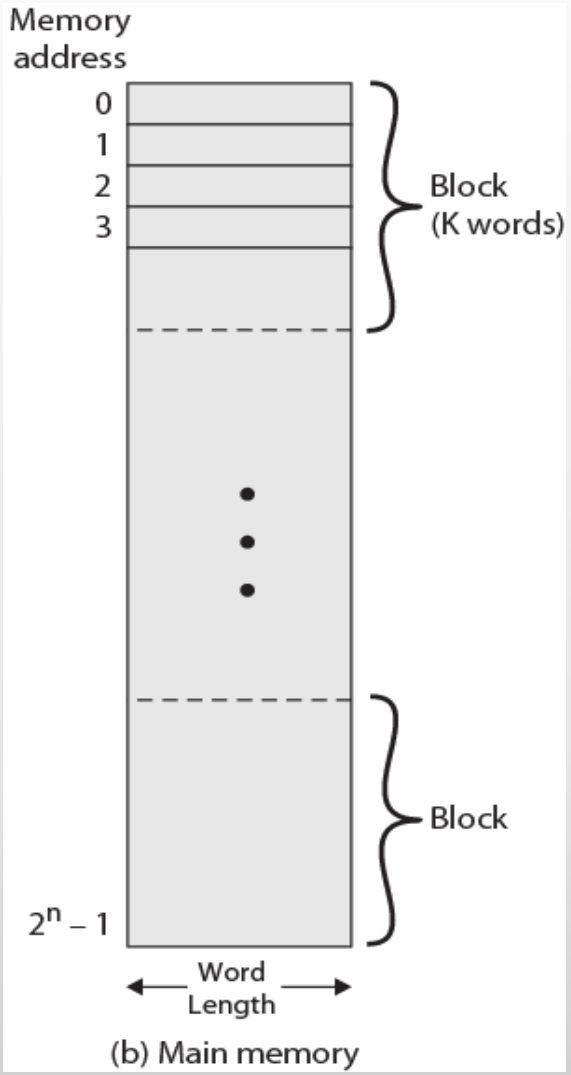




Cache/Main Memory Structure

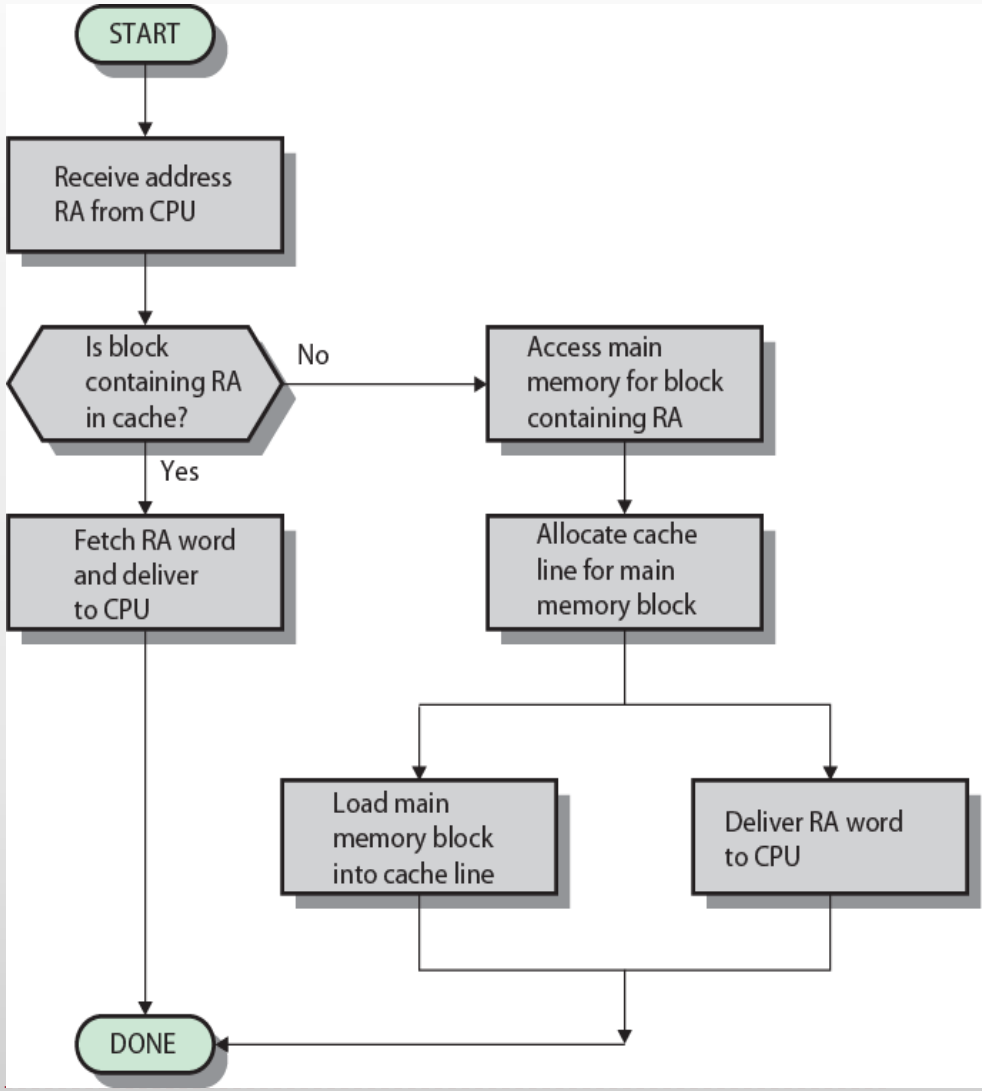
- If a word in a **block** of memory is **read**, that block is **transferred** to one of **cache** lines
- An individual **line cannot be** uniquely and permanently **dedicated to** a particular **block** → **tag** identifying the block is being stored





Cache – Read operation

- CPU requests contents of memory location
- Check cache for this data
- If present:
  - get from cache
  - else read required block from main memory to cache
- Then deliver to CPU

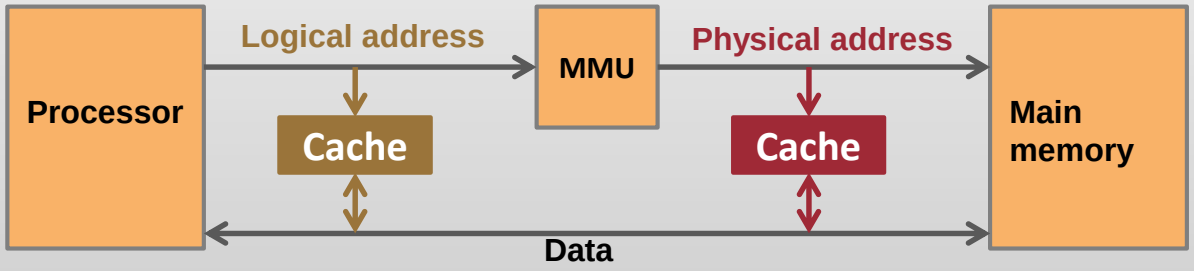


Cache Design

- Addressing
- 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
  - Write once
- Line(Block) Size
- Number of Caches
  - Levels
  - Unified or split

Cache Addresses

- Cache can be located
  - **Between processor and virtual MMU**
  - **Between MMU and main memory**
- **Logical cache** (virtual cache) stores data using virtual addresses
  - Processor accesses cache directly, not thorough physical cache
  - Cache access faster, before MMU address translation
  - Virtual addresses use same address space for different applications
- **Physical cache** stores data using main memory physical addresses



Mapping function

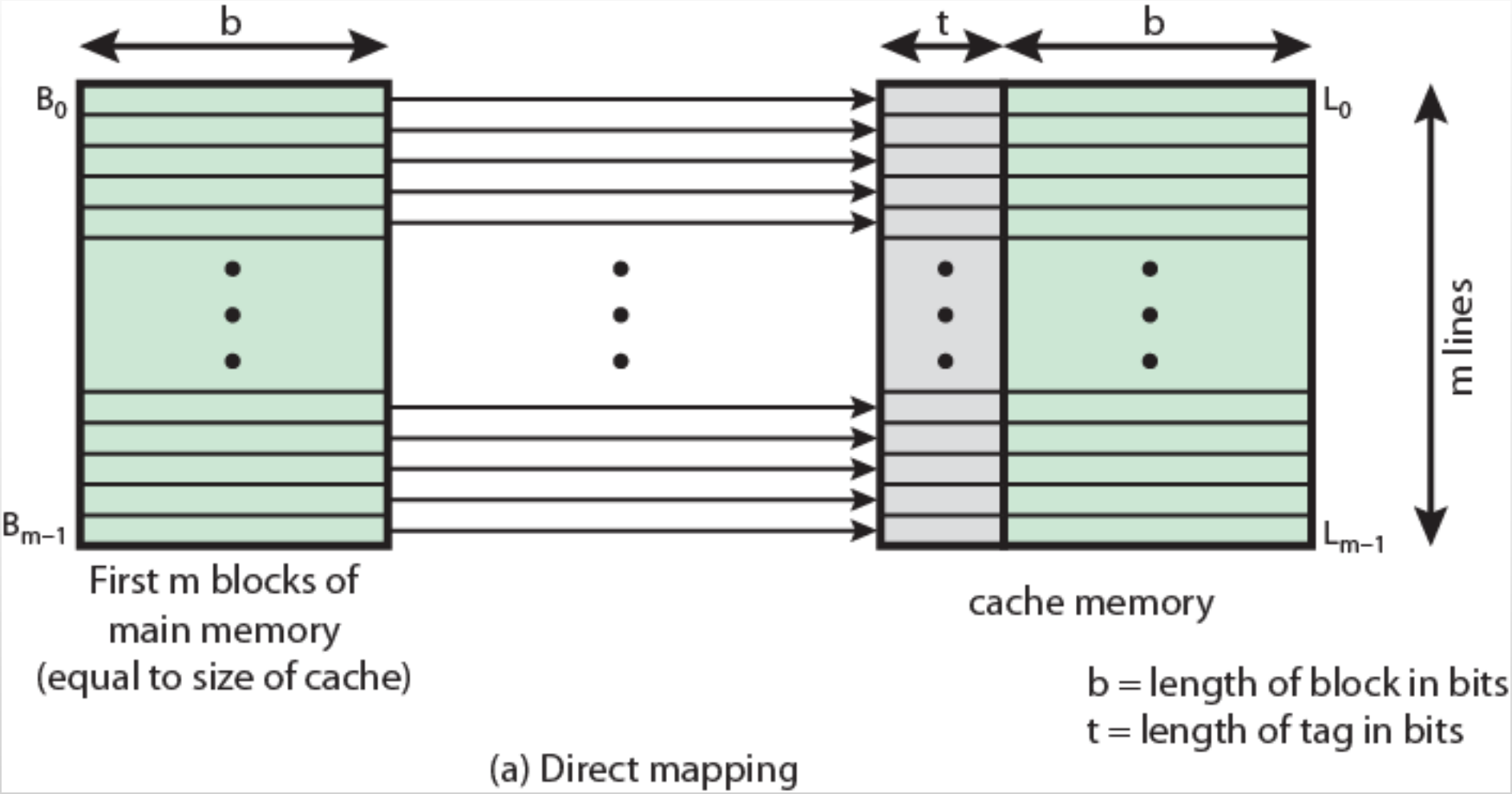
- Fewer cache lines than main memory blocks:
  - ***algorithm for mapping main memory blocks into cache lines***
  - ***means for determining which main memory block currently occupies a cache line***

- Mapping function → cache organization
- Three techniques can be used:
  - Direct
  - Associative
  - Set associative

Direct Mapping

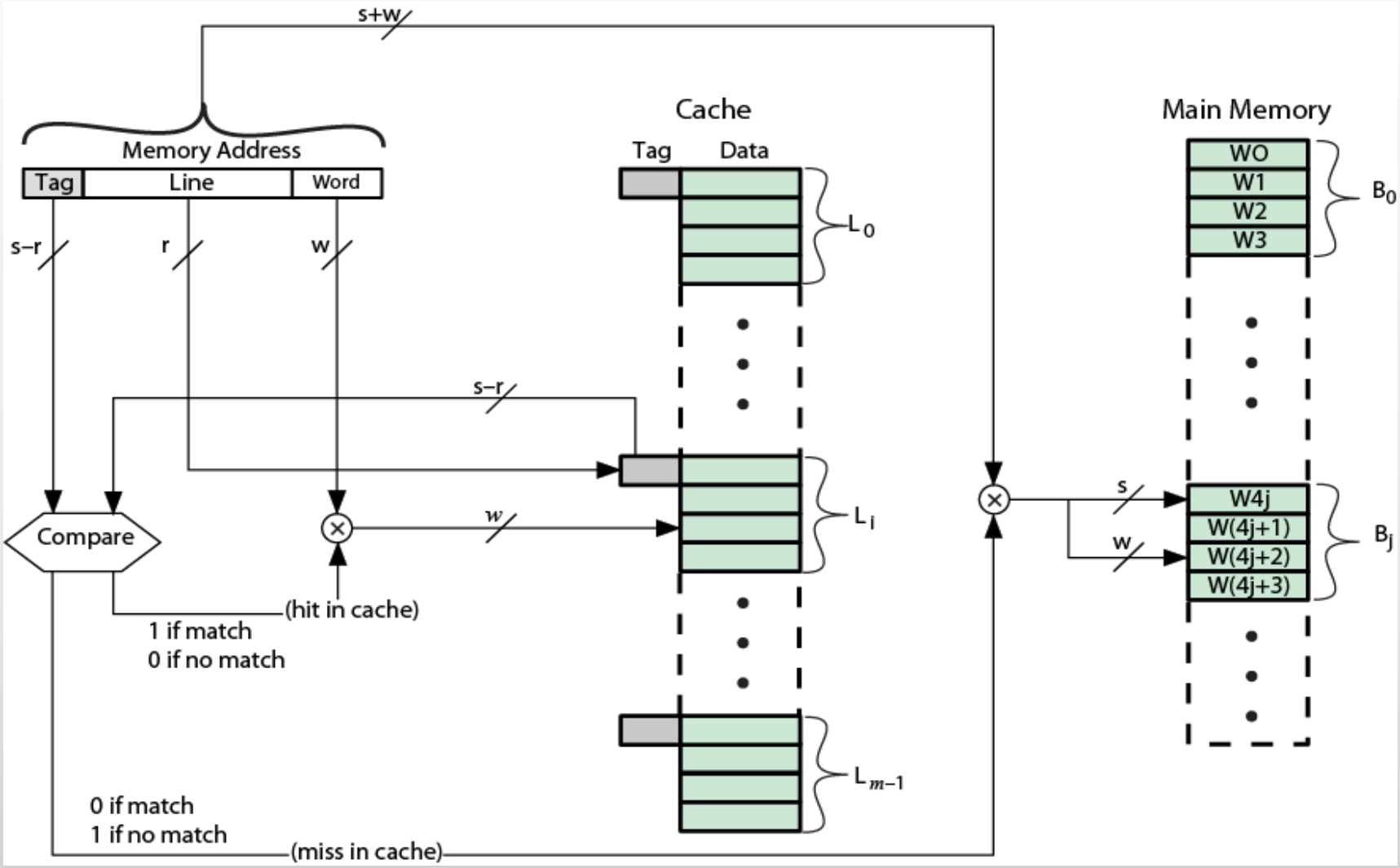
- Each block of main memory maps to only one cache line
  - **i.e. if a block is in cache, it must be in one specific place**
- Main memory address can be divided in two parts fields:
  - Least significant ***w* bits** identify unique word
  - Most significant ***s* bits** specify one of the 2^s memory blocks:
    - cache line - r bits
    - tag - (s-r) bits

Direct Mapping from Cache to Main Memory



(a) Direct mapping

Direct Mapping Cache Organization

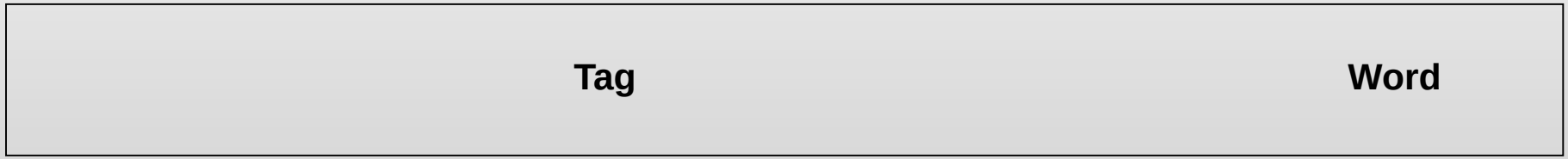


Direct Mapping pros & cons

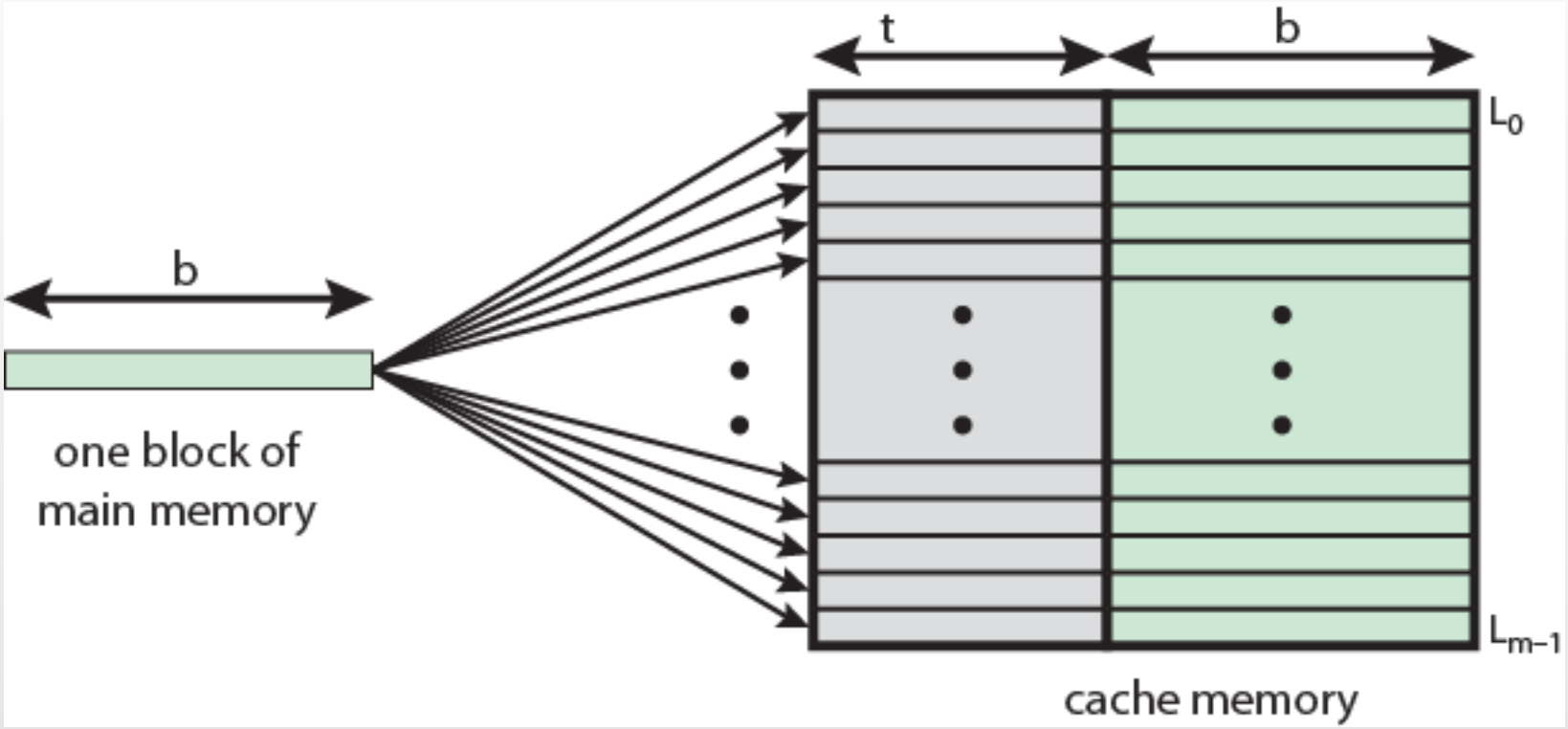
- Simple
- Inexpensive
- Fixed location for given block
  - If a program accesses 2 blocks that map to the same line repeatedly, cache misses are very high

Associative Mapping

- A main **memory block** can be loaded into **any line of cache**
- Memory address is interpreted as
  - **Tag** field
  - **Word** field
- Tag uniquely identifies block of memory
- No field in the address corresponds to the line number

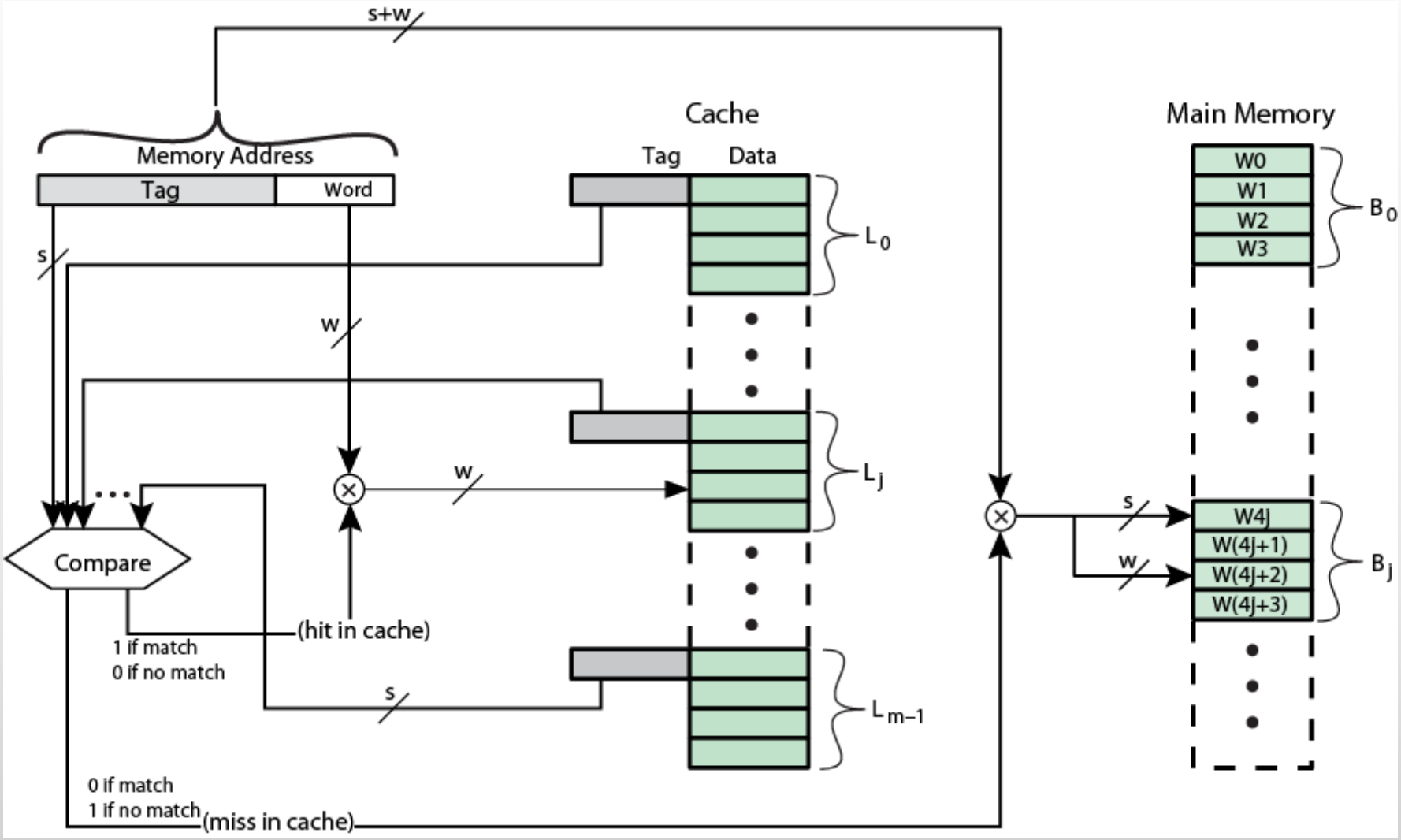


Associative Mapping from Cache to Main Memory



- To determine whether a block is in the cache, the cache **control logic must simultaneously examine every line's tag** for a match

Fully Associative Cache Organization



Associative Mapping Summary

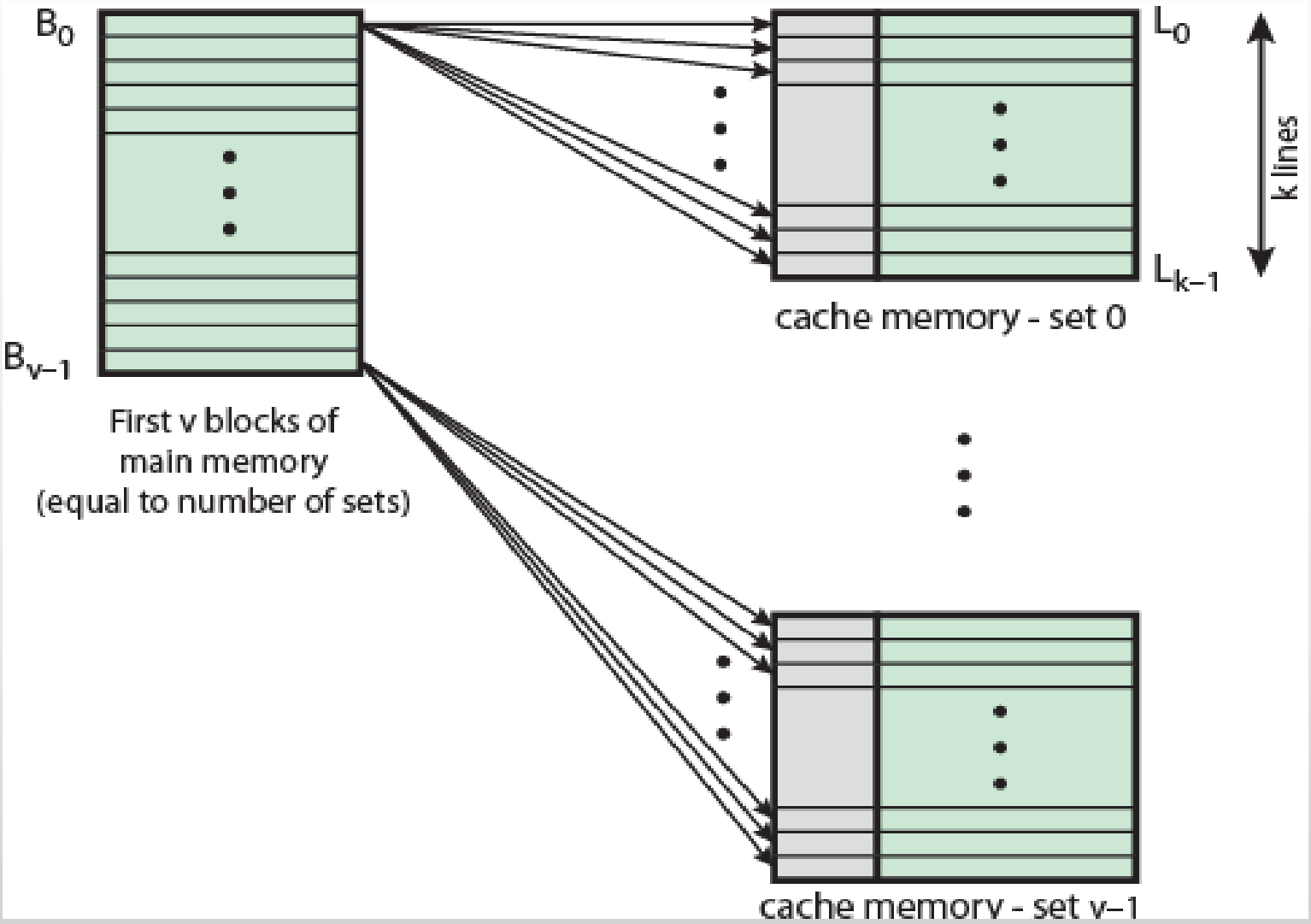
- Address length = (s + w) bits
- Number of addressable units = 2^{s+w} words or bytes
- Block size = line size = 2^{w} words or bytes
- Number of blocks in main memory = $2^{s+w}/2^{w} = 2^{s}$
- Number of lines in cache = undetermined
- Size of tag = s bits

Set Associative Mapping

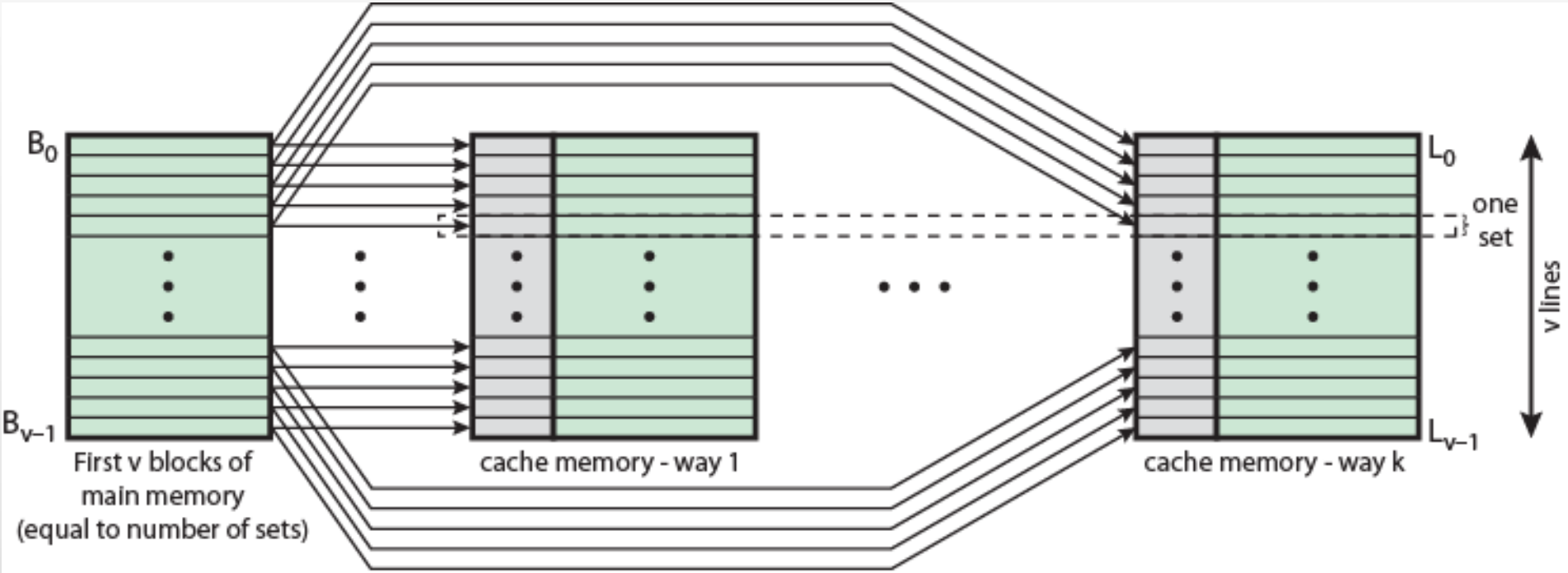
- Set-associative mapping exhibits the **strengths of both** the ***direct*** and **associative** approaches while reducing their disadvantages

- 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

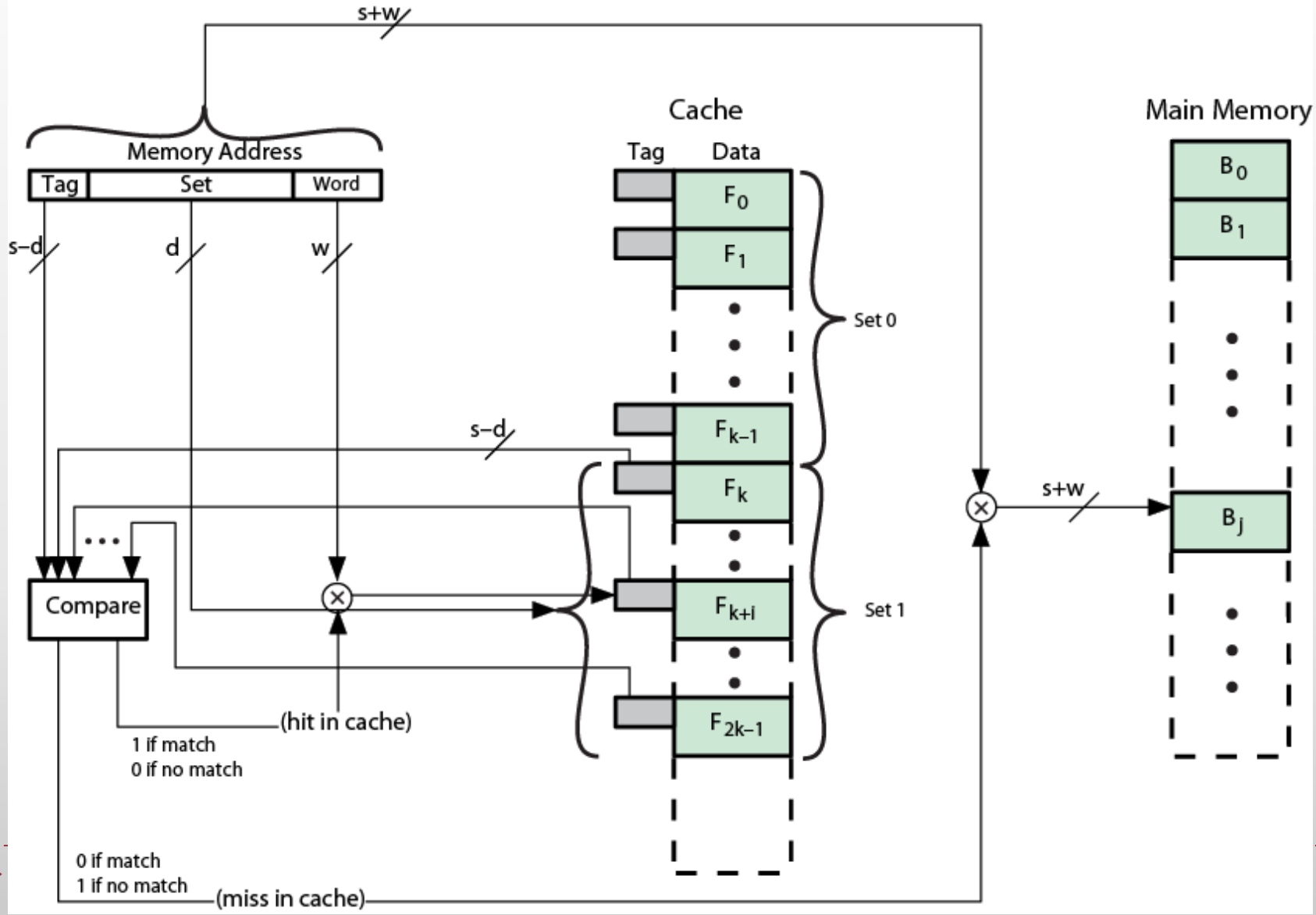
Mapping From Main Memory to Cache: v Associative



Mapping From Main Memory to Cache: k-way Associative



K-Way Set Associative Cache Organization



Set Associative Mapping Address Structure

- 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

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^{d}
- Number of lines in set = k
- Number of sets = v = 2^{d}
- Number of lines in cache = kv = k * 2d
- Size of tag = (s – d) bits

Replacement Algorithms

- Direct mapping
  - No choice
  - Each block only maps to one line
  - Replace that line
- Associative & Set Associative
  - Hardware implemented algorithm (speed)
  - Least Recently used (LRU)
    - but in 2 way set associative “Which of the 2 block is lru?”
  - First in first out (FIFO)
  - Least frequently used
    - replace block which has had fewest hits
  - Random

Write Policy

- **Write through**
  - All writes go to main memory as well as cache
  - Lots of traffic
  - Slows down writes
- **Write back**
  - Updates initially made in cache only and *update bit* is set
  - If block is to be replaced, write to main memory if update bit
  - Other caches get out of sync
  - I/O must access main memory through cache

Write Policy

- In a bus organization we can have:
  - **more than one device** (typically a processor) has a **cache**
  - **main memory is shared**, a new problem is introduced
- 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 word)

Line Size

- When a block of data is retrieved and placed in the cache
  - not only the desired word is retrieved
  - but also some number of adjacent words

- Increased block size will increase hit ratio
  - principle of locality
- Hit ratio will decreases as block becomes even bigger
  - Probability of using newly fetched information becomes less than probability of reusing replaced

Line Size

- 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 close to optimum
- For HPC systems, 64 and 128 byte most common

Multilevel Caches

- The use of **multiple caches** has become the **norm**

- High logic density enables **caches on chip**
  - Faster than bus access
  - When the requested instruction or data is found in the on-chip cache, the bus access is eliminated
  - Bus is free for other transfers

Multilevel Caches

- Common to use both on and off chip cache
  - L1 on chip, L2 off chip in static RAM
  - L2 access much faster than DRAM or ROM
  - L2 often uses separate data path
  - L2 may be on chip
  - Resulting in L3 cache

Unified versus Split Caches

- It is quite common to split the cache into two:
  - one dedicated to instructions
  - one dedicated to data
- These two caches both exist at the same level, typically as two L1 caches
  - When the processor attempts to fetch an **instruction** from main memory, it first consults the instruction L1 cache
  - when the processor attempts to fetch **data** from main memory, it first consults the data L1 cache

Unified versus Split Caches

- Advantages of unified cache
  - Higher hit rate
    - Balances load of instruction and data fetch
      - if many more instruction fetches are involved in the execution, then the cache will tend to fill up with instructions
      - if an execution pattern involves relatively more data fetches, the opposite will occur
    - Only one cache to design & implement
- Advantages of split cache
  - Eliminates cache contention between instruction fetch/decode unit and execution unit
    - Important in pipelining