LAU
الجامعة اللبنانية الأميركية
Lebanese American University

- The Memory Hierarchy
- Cache Memory

Chapters 4 & 5

Based on:
William Stallings
Computer Organization and Architecture, 11th Global Edition

Key Characteristics of Computer Memory Systems

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

- CPU
  - Processor requires its own local memory, in the form of registers
- Internal
  - Often equated with main memory
  - Cache is another form of internal memory
- External
  - Consists of peripheral storage devices, that are accessible to the processor via I/O controllers

Capacity

- Word size
  - Natural unit of organization
  - Common word lengths: 8, 16, and 32 bits
- Number of words
  - Or Bytes
- External memory capacity is typically expressed in terms of bytes

Unit of Transfer

- For internal memory
  - Usually governed by bus data width
  - May be equal to the word length, but is often larger

- For external memory
  - Usually a block which is much larger than a word

- Addressable unit
  - Smallest location which can be uniquely addressed
  - In some systems, the word
  - Relationship between the length in bits A of an address and the number N of addressable units is $2^A = N$

Access Methods

- **Sequential**
  - Memory is organized into units of data, called records
  - Start at the beginning and read through in order
  - Access time is variable, depends on location of data and previous location
  - e.g. tape
- **Random**
  - Individual addresses identify locations exactly
  - Access time is independent of location or previous access
  - e.g. RAM
- **Direct**
  - Individual blocks have 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
- **Associative**
  - A word is retrieved based on a portion of its contents rather than its address
  - Access time is independent of location or previous access
  - e.g. cache

Performance

- **Access time** (latency)
  - 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
    - Access time plus any additional time required before second access can commence
  - Cycle time is access + recovery
  - Concerned with the system bus, not the processor

- **Transfer rate**
  - Rate at which data can be transferred into or out of a memory unit
  - In terms of bits/second
  - For random-access memory it is equal to 1/(cycle time)

Physical Types

The most common forms are:

- Semiconductor memory
  - RAM
- Magnetic surface memory
  - Disk and tape
- Optical
  - CD and DVD

Physical Characteristics

Several physical characteristics of data storage are important:

- Volatile memory
  - Information decays naturally or is lost when electrical power is switched off
- Nonvolatile memory
  - Once recorded, information 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
- Nonerasable memory
  - Cannot be altered, except by destroying the storage unit
  - Semiconductor memory of this type is known as read-only memory (ROM)

Organization

- Physical arrangement of bits into words
- Not always obvious

Memory Hierarchy

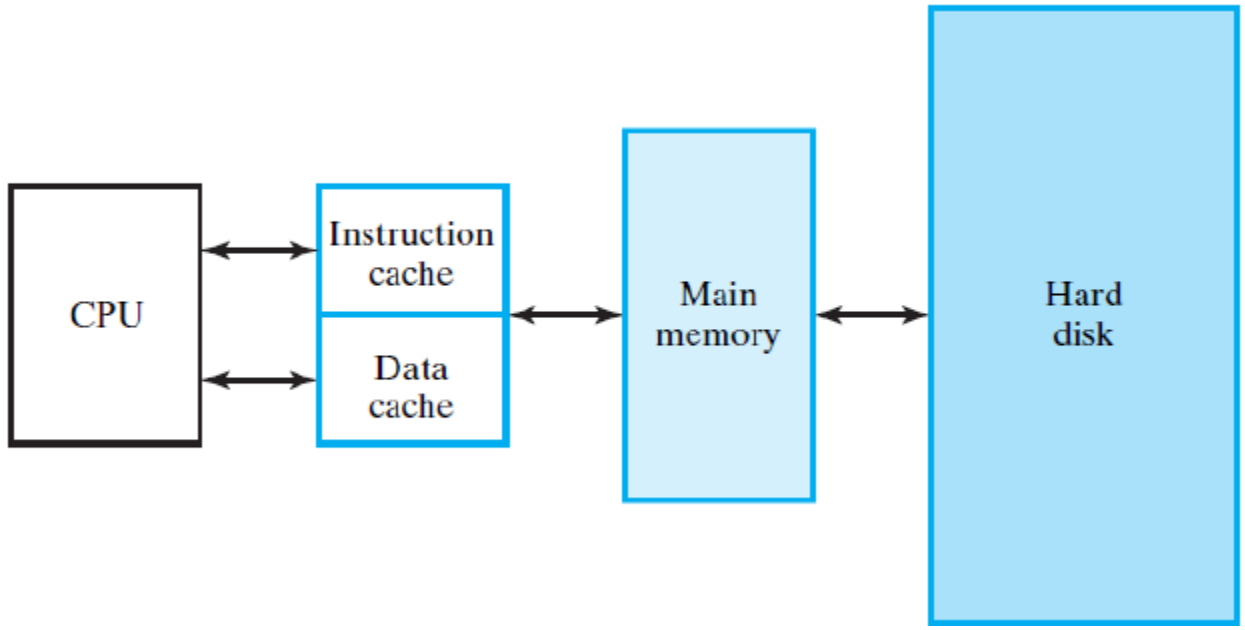
- Design constraints on a computer's memory can be summed up by three questions:
  - How much, how fast, how expensive

- There is a trade-off among capacity, access time, and cost
  - Faster access time, greater cost per bit
  - Greater capacity, smaller cost per bit
  - Greater capacity, slower access time

- The way out of the memory dilemma is not to rely on a single memory component or technology, but to employ a **memory hierarchy**

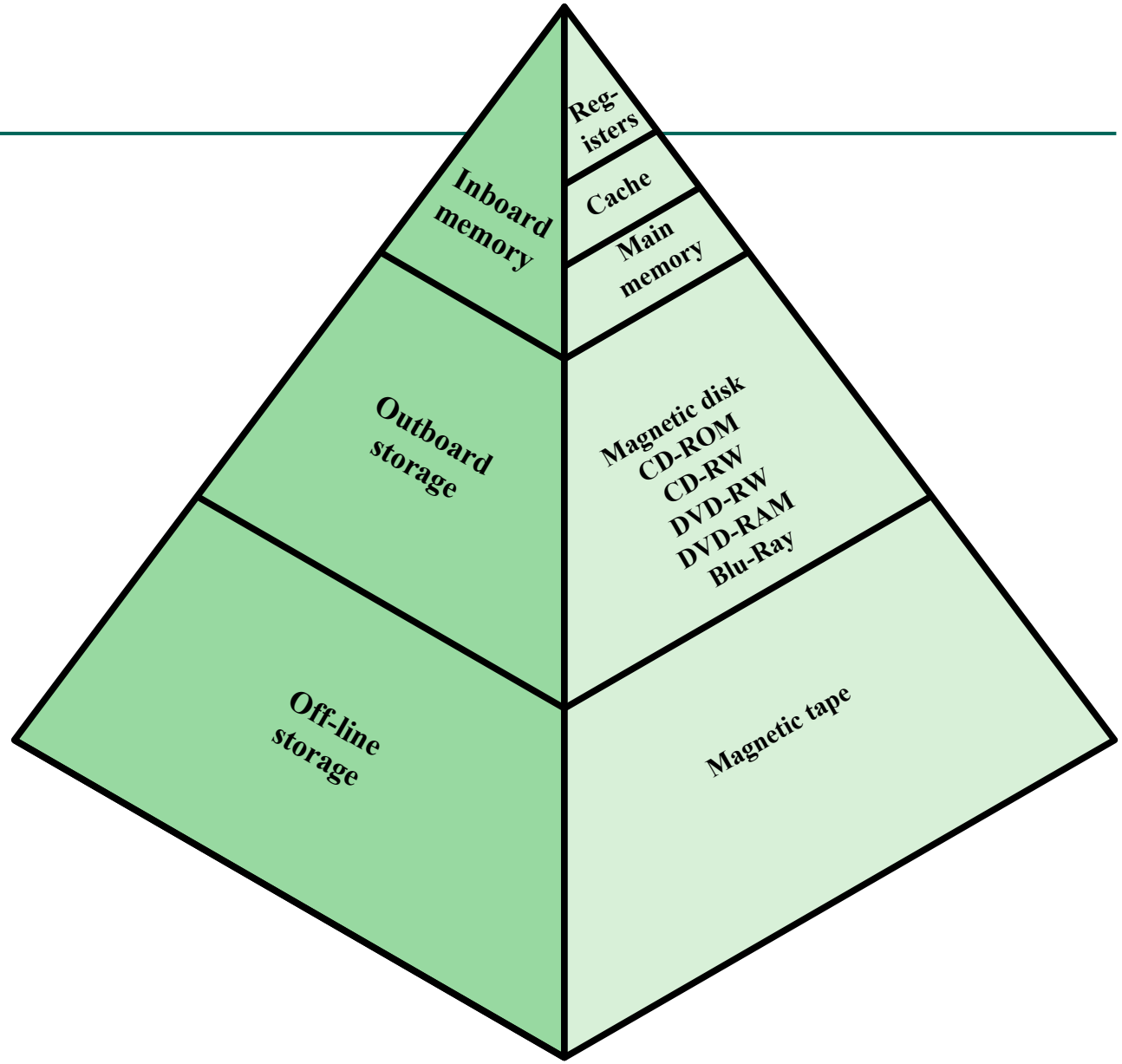
Memory Hierarchy

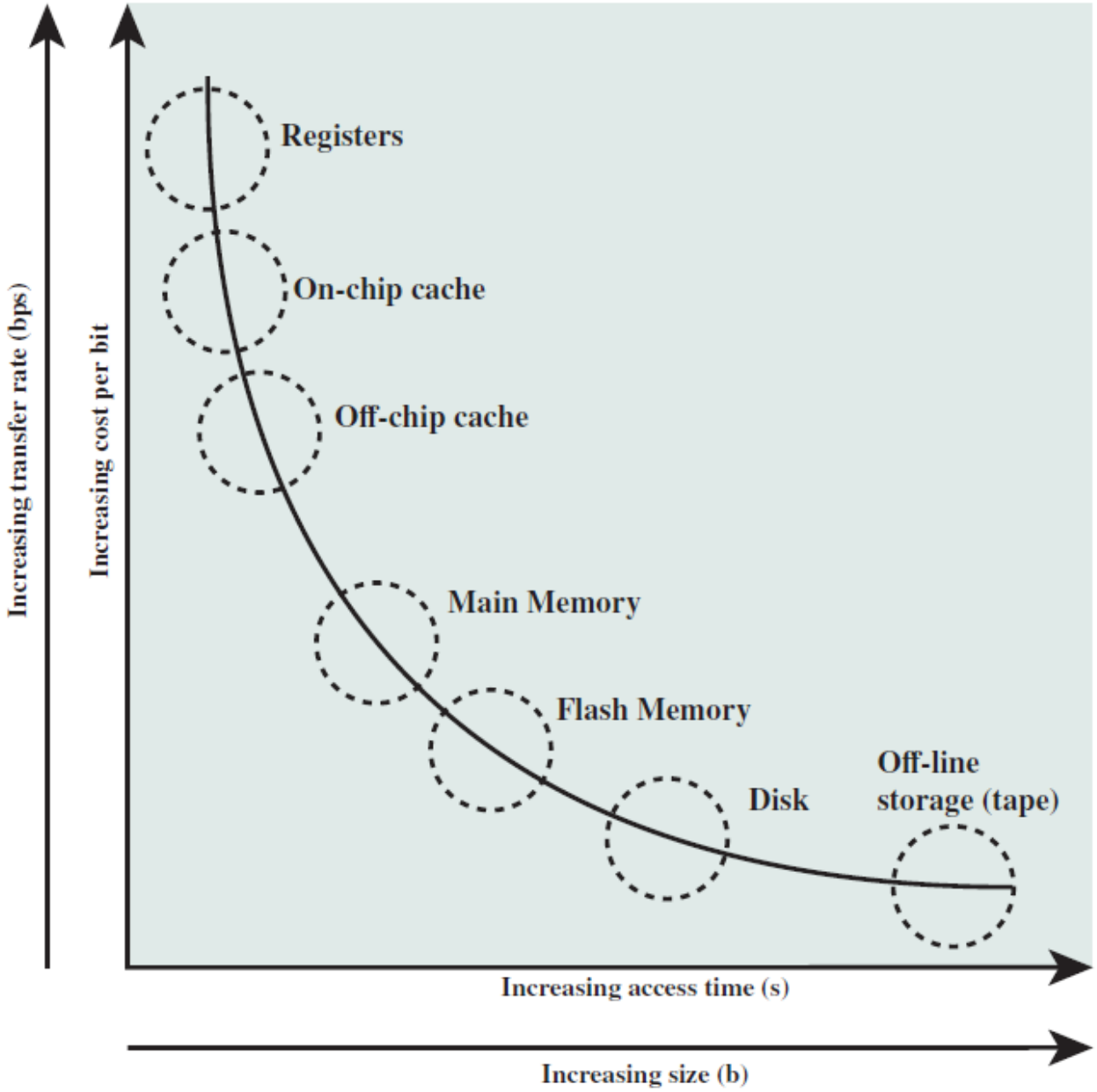
- Multiple levels of memory with different speeds and sizes.
- The faster memory is close to the processor and the slower, less expensive memory is below it.



Memory Hierarchy

- Going down the hierarchy:
  - Decreasing cost per bit
  - Increasing capacity
  - Increasing access time
  - Decreasing frequency of access of the memory by the processor





Relative Cost, Size, and Speed characteristics across the Memory Hierarchy

Example

- CPU clock period = 1 ns

- Assume:
  - 3 levels of memory: L1, L2, and L3
  - 95% of fetches come from L1, access time: 2 ns
  - 4.999995% of fetches come from L2, access time: 10 ns
  - 0.000005% of fetches come from L3, access time: 13 ms

- Average access time = 0.95 * 2 + 0.04999995 * 10 + 0.00000005 * 13 * 10^6 = 3.05 ns = 3 times CPU clock period

- Acceptable average access time
  - Closer to L1 access time

Locality of Reference

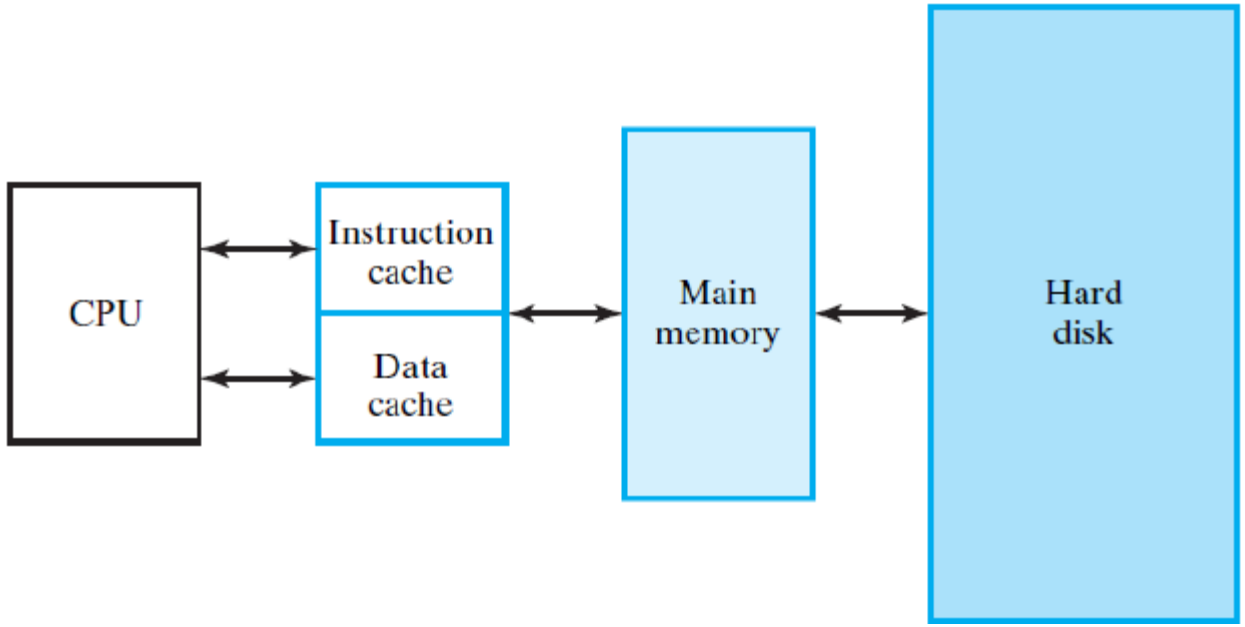
- During the course of the execution of a program, memory references tend to cluster
- Programs access a small proportion of their address space at any time

- **Temporal locality**
  - Items accessed recently are likely to be accessed again soon
  - If an item is referenced, it will tend to be referenced again soon
  - e.g., instructions in a loop

- **Spatial locality**
  - Items near those accessed recently are likely to be accessed soon
  - If an item is referenced, items whose addresses are close by will tend to be referenced soon
  - e.g., sequential instruction access, array data

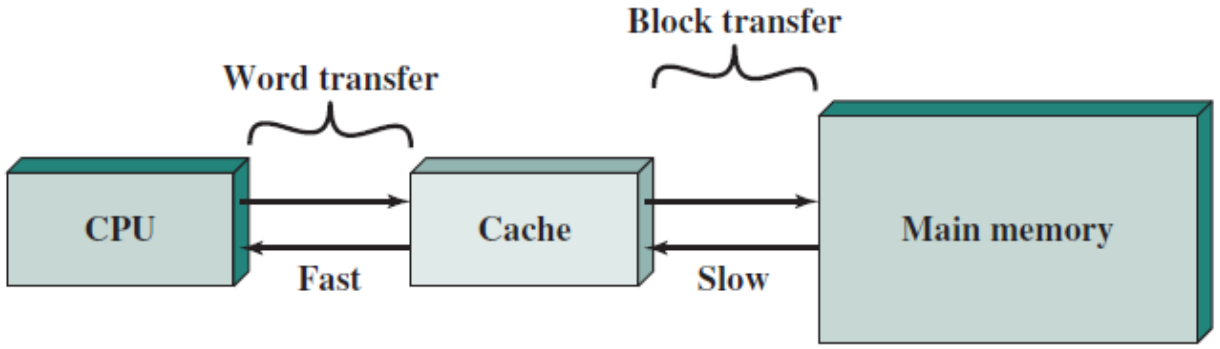
Taking Advantage of Locality

- Memory hierarchy
- Store everything on disk
- Copy recently accessed (and nearby) items from disk to smaller main memory
- Copy more recently accessed (and nearby) items from main memory to smaller cache memory

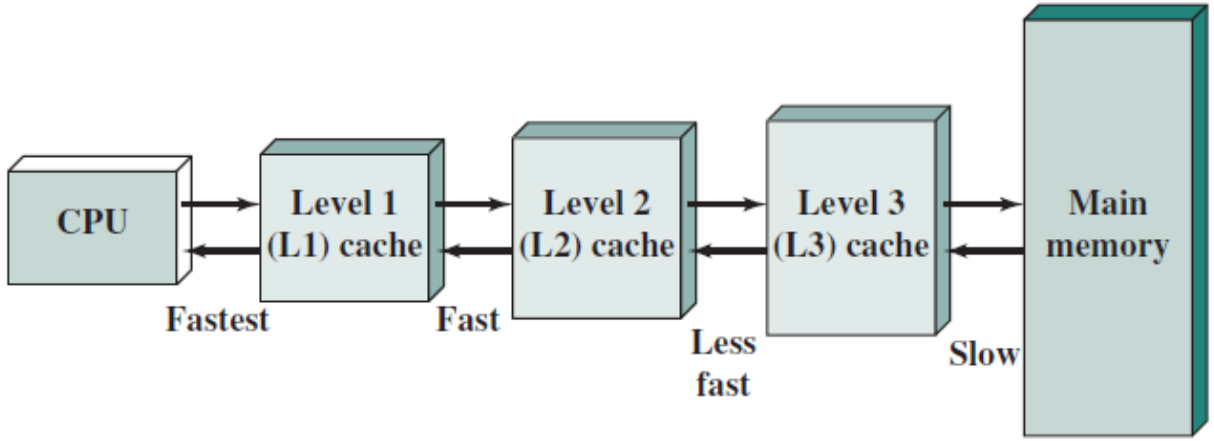


Cache

- *Cache: a safe place for hiding or storing things*
- Represent the level of the memory hierarchy between the processor and main memory
- Small amount of fast memory
- May be located on CPU chip



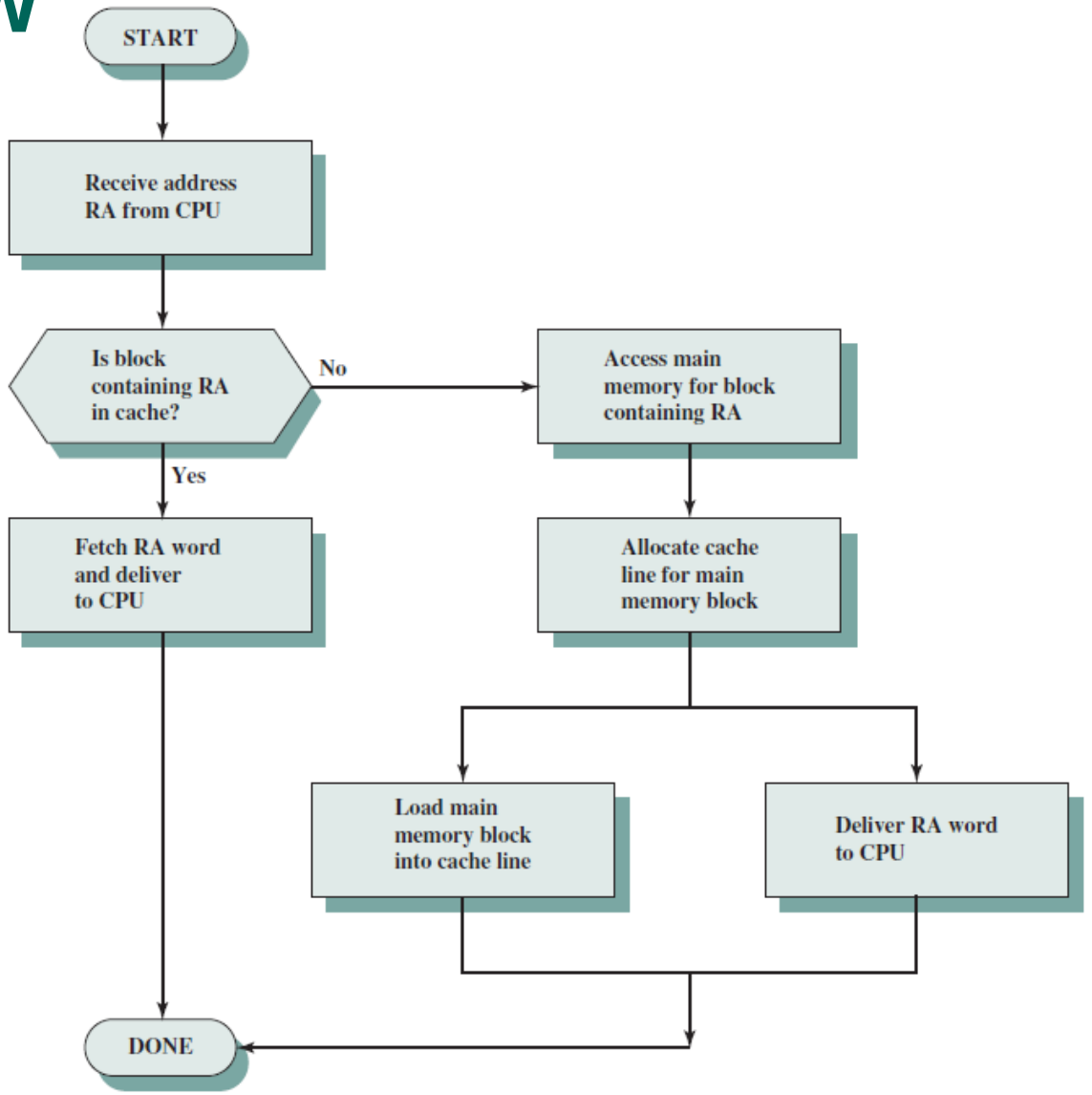
(a) Single cache



(b) Three-level cache organization

Cache Operation - Overview

- Read operation:
  - CPU requests contents of memory location
  - Check cache for data
  - If present, get from cache (fast)
  - If not present, read required block from main memory to cache
  - Then deliver from cache to CPU



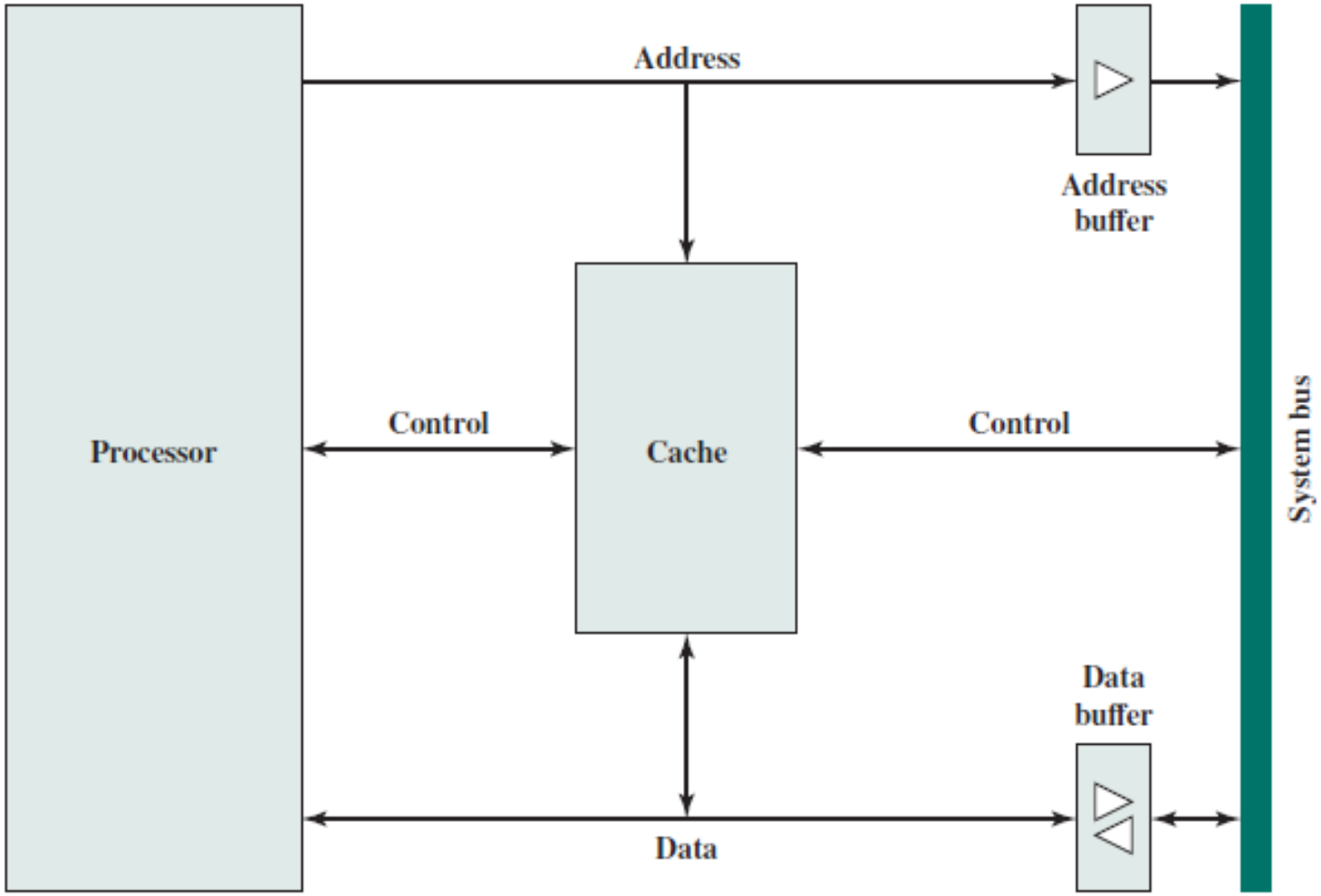
Some Definitions

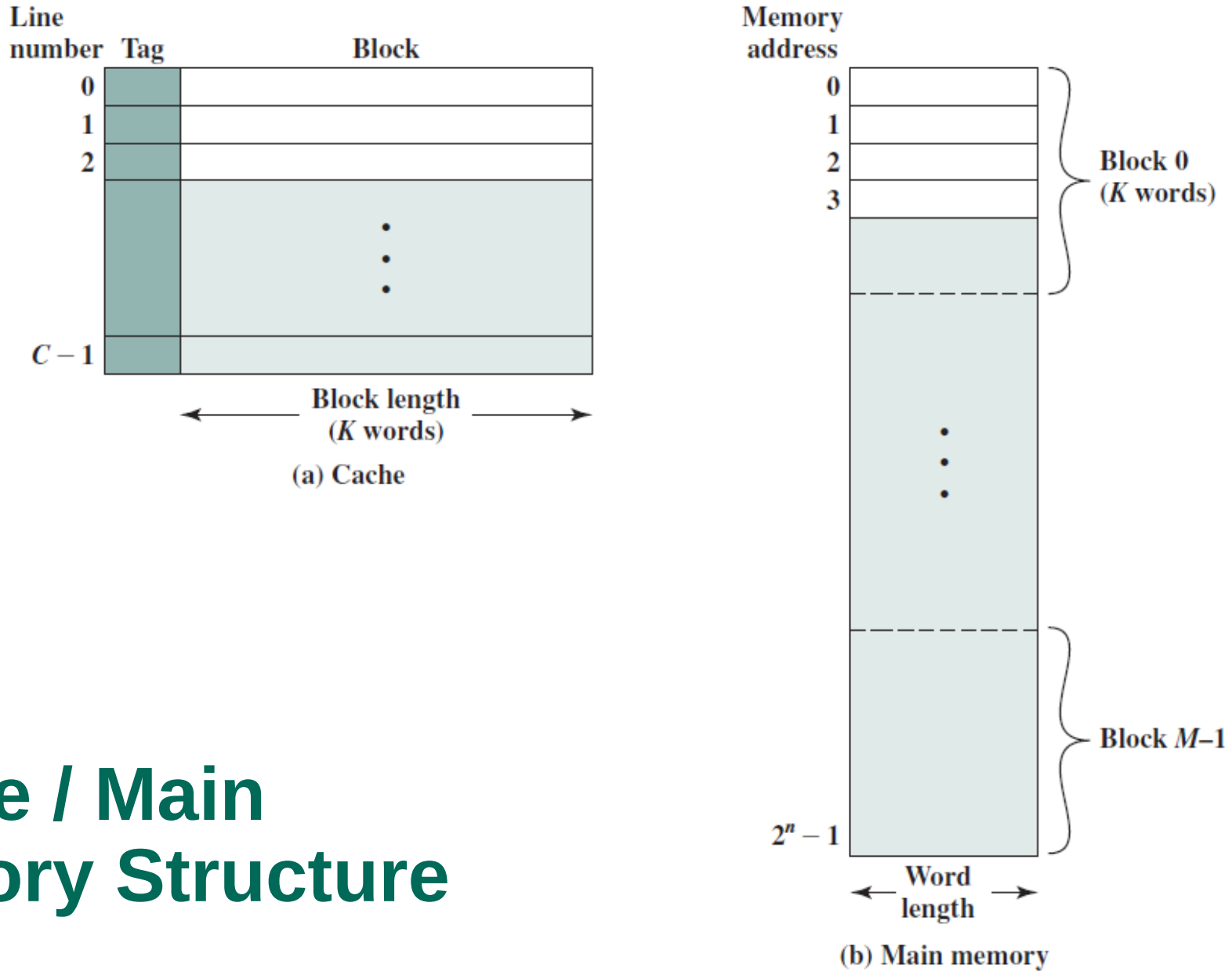
- **Block** (also called a **line**): unit of copying
  - May be multiple words

- If accessed data is present in upper level
  - **Hit**: access satisfied by upper level
    - **Hit ratio**: hits/accesses

- If accessed data is absent
  - **Miss**: block copied from lower level
    - Time taken: **miss penalty**
    - **Miss ratio**: misses/accesses
      = 1 – hit ratio

Typical Cache Organization





Cache / Main Memory Structure

Elements of Cache Design

- Size
- Mapping function
- Replacement algorithm
- Write policy
- Line size
- Number of caches

Cache Size

- Size to be small so that average cost per bit is small
- Size to be large so that average access time is less
- Motivation for minimizing cache size:
  - The larger the cache, the larger the number of gates involved in addressing the cache
  - Large caches tend to be slightly slower than small ones
  - 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.

Cache Sizes of Some Processors

Processor	Type	Year of Introduction	L1 Cache[a]	L2 cache	L3 Cache
IBM 360/85	Mainframe	1968	16 to 32 kB	—	—
PDP-11/70	Minicomputer	1975	1 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 620	PC	1996	32 kB/32 kB	—	—
IBM S/390 G6	Mainframe	1999	256 kB	8 MB	—
Pentium 4	PC/server	2000	8 kB/8 kB	256 kB	—
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
CRAY XD-1	Supercomputer	2004	64 kB/64 kB	1MB	—
IBM POWER6	PC/server	2007	64 kB/64 kB	4 MB	32 MB
IBM z10	Mainframe	2008	64 kB/128 kB	3 MB	24-48 MB
Intel Core i7 EE 990	Workstaton/ Server	2011	6 × 32 kB/32 kB	6 × 1.5 MB	12 MB
IBM zEnterprise 196	Mainframe/ Server	2011	24 × 64 kB/128 kB	24 × 1.5 MB	24 MB L3 192 MB L4
IBM z13	Mainframe/ server	2015	24 × 96 kB/128 kB	24 × 2 MB/2 MB	64 MB L3 480 MB L4
Intel Core i0-7900X	Workstation/ server	2017	8 × 32 kB/32 kB	8 × 1 MB	14 MB

[a] Two values separated by a slash refer to instruction and data caches.

Mapping Function

- Because there are fewer cache lines than main memory blocks, an algorithm is needed for mapping main memory blocks into cache lines
- Choice of **mapping function** dictates how cache is organized
- Three techniques can be used:
  - Direct
  - Associative
  - Set associative

Direct Mapping

- The simplest technique

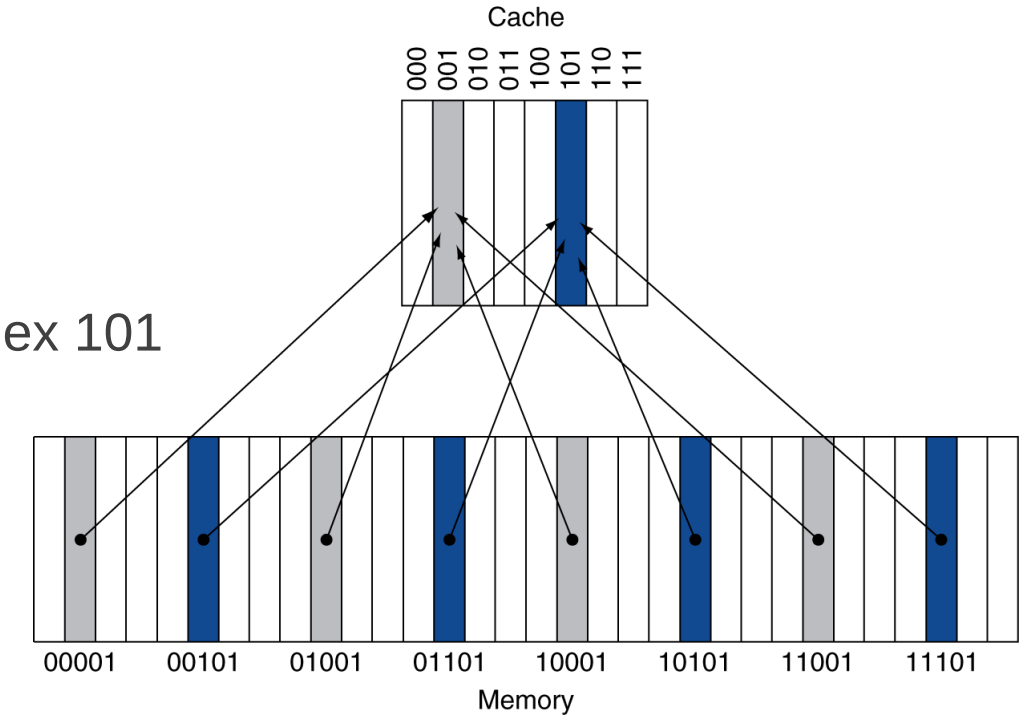
- Each block of main memory maps to **only one** cache line
  - If a block is in the cache, it must be in one specific place

- Mapping is expressed as ***i = j modulo m***
  - *i = cache line number*
  - *j = main memory block number*
  - *m = number of lines in the cache*

Direct Mapping – Simple Example

- Direct mapped: only one choice
  - (Block address) modulo (#Blocks in cache)

- In this example:
  - #Blocks is a power of 2
  - Use low-order address bits
  - e.g, address 29: 11**101** (29%8 = 5) will map to index 101



Tag and Valid Bits

- How do we know which particular block is stored in a cache location?
  - Store block address as well as the data
  - Actually, only need the high-order bits
  - Called the **tag**

- What if there is no data in a location?
  - **Valid bit**: 1 = present, 0 = not present
  - Initially 0

Simple Example

- 8-blocks, 1 word/block, direct mapped
- Initial state of cache:

Block	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Miss	110

Block	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	**Y**	**10**	**Mem[10110]**
111	N		

Word addr	Binary addr	Hit/miss	Cache block
26	11 010	Miss	010

Block	V	Tag	Data
000	N		
001	N		
010	**Y**	**11**	**Mem[11010]**
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Hit	110
26	11 010	Hit	010

Block	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Word addr	Binary addr	Hit/miss	Cache block
16	10 000	Miss	000
3	00 011	Miss	011
16	10 000	Hit	000

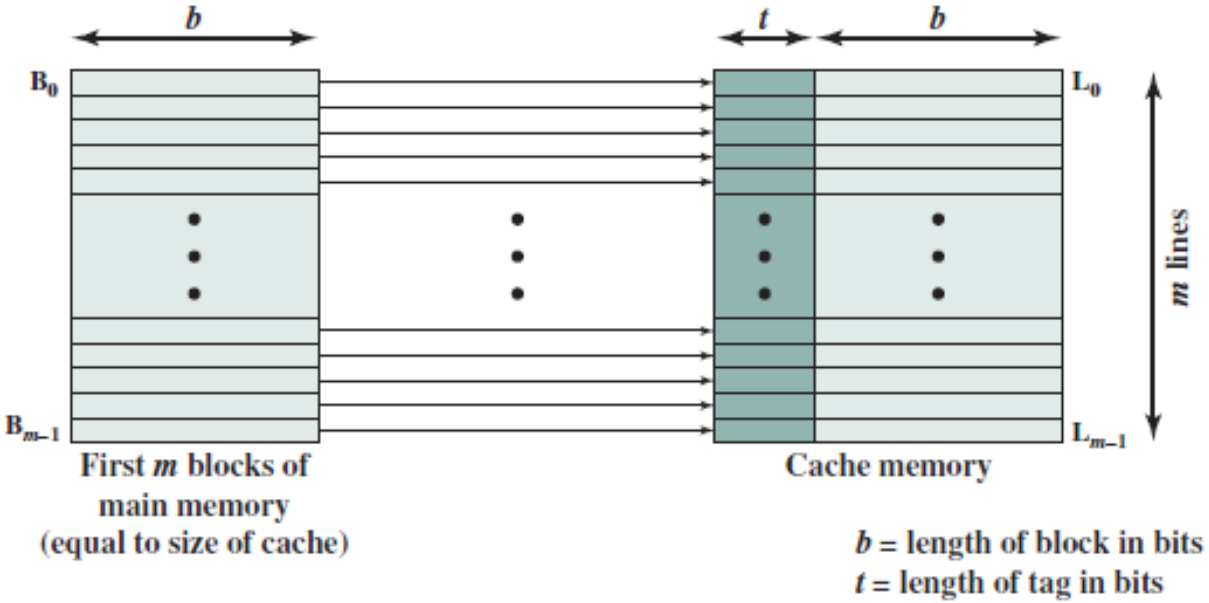
Block	V	Tag	Data
000	**Y**	**10**	**Mem[10000]**
001	N		
010	Y	11	Mem[11010]
011	**Y**	**00**	**Mem[00011]**
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

Block	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	**Y**	**10**	**Mem[10010]**
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

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

- Address is in two parts
- Least significant **w** bits identify unique word
- Most significant **s** bits specify one memory block
- The MSBs are split into a cache line field **r** and a tag of **s - r**

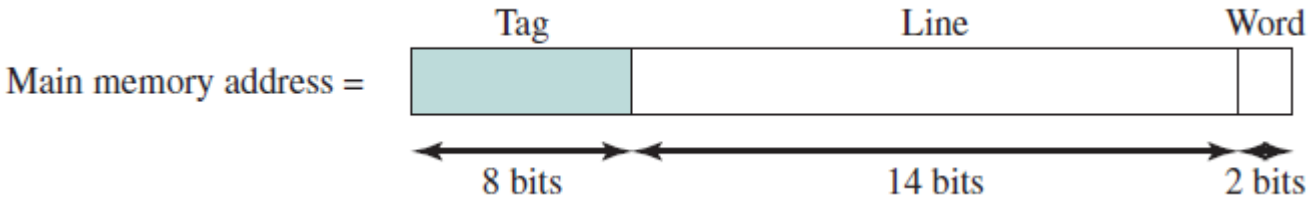
Assumptions

From here on, and for all three cases of mapping:

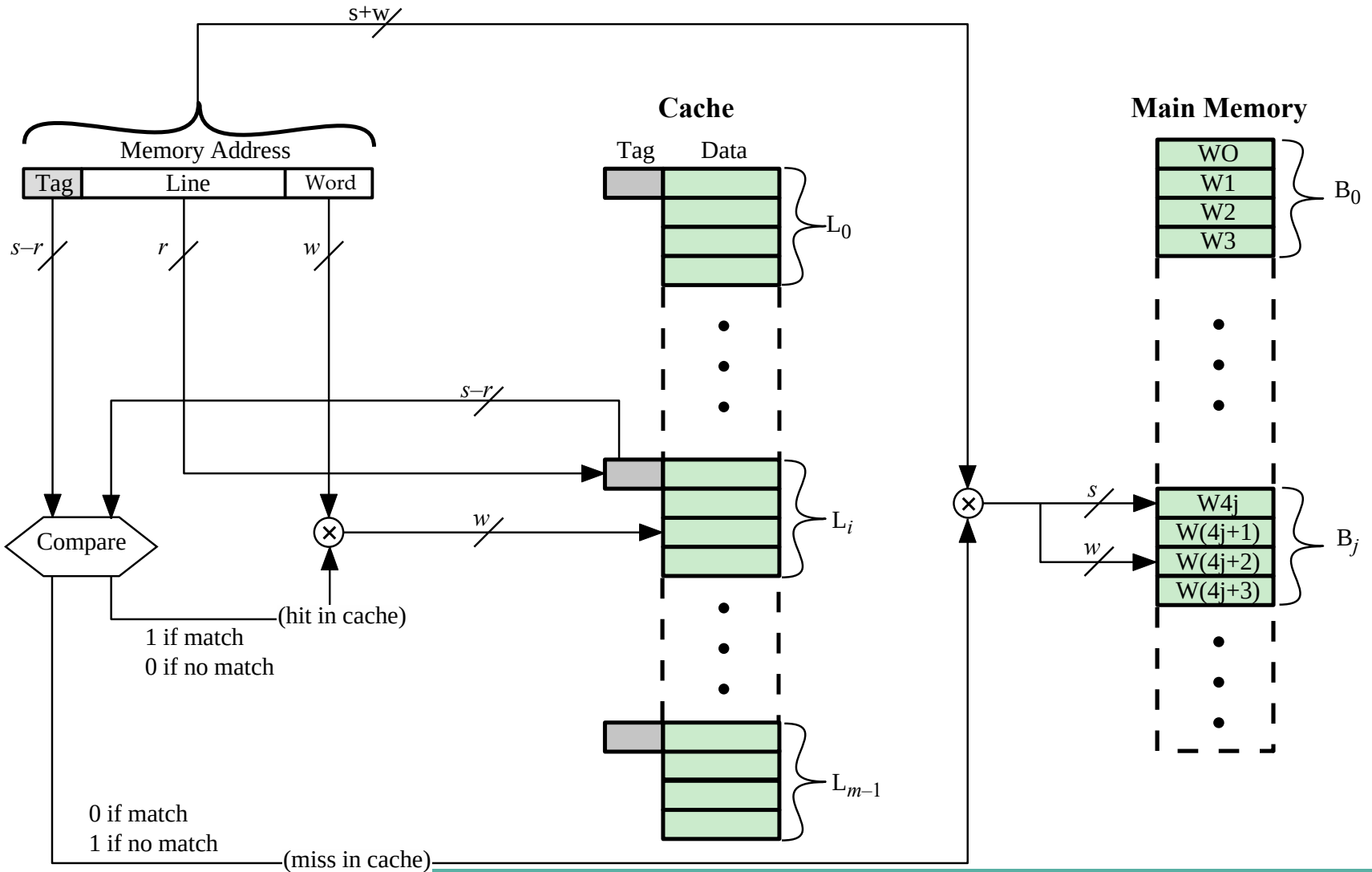
- The cache can hold 64 kB
- Data is transferred between main memory and the cache in blocks of 4 bytes each.
- This means that the cache is organized as 16K = 2^{14} lines of 4 bytes each.
- The main memory consists of 16MB, with each byte directly addressable by a 24-bit address (2^{24} = 16M).
- Thus,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

Direct Mapping Address Structure

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



Direct Mapping Cache Organization



Direct 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 = m = 2^{r}
- Size of cache = 2^{r+w} words or bytes
- Size of tag = (s – r) bits

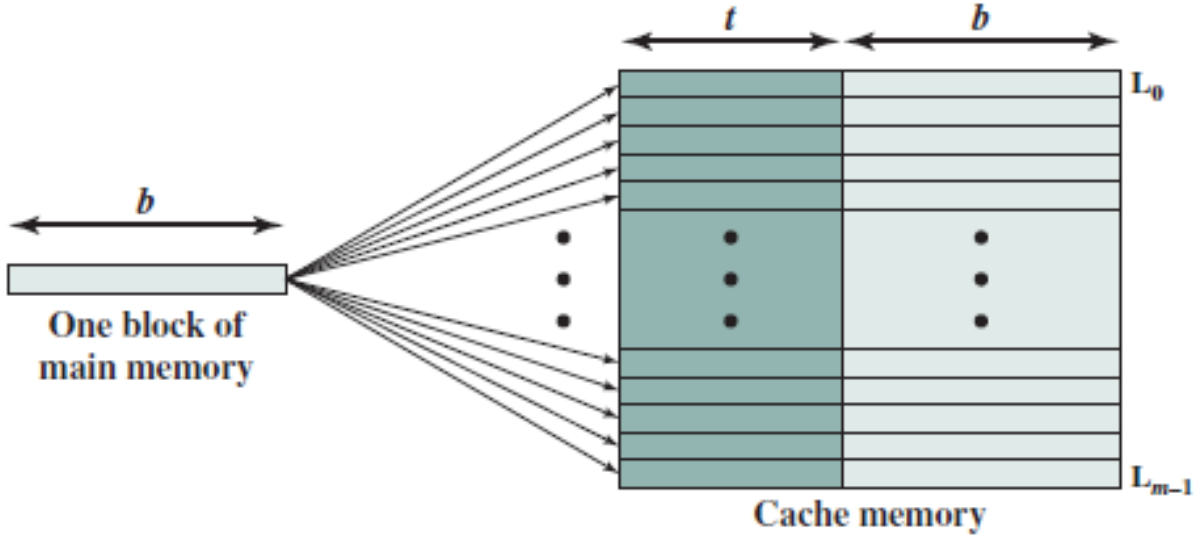
Direct Mapping Pros & Cons

- Pros:
  - Simple
  - Inexpensive to implement

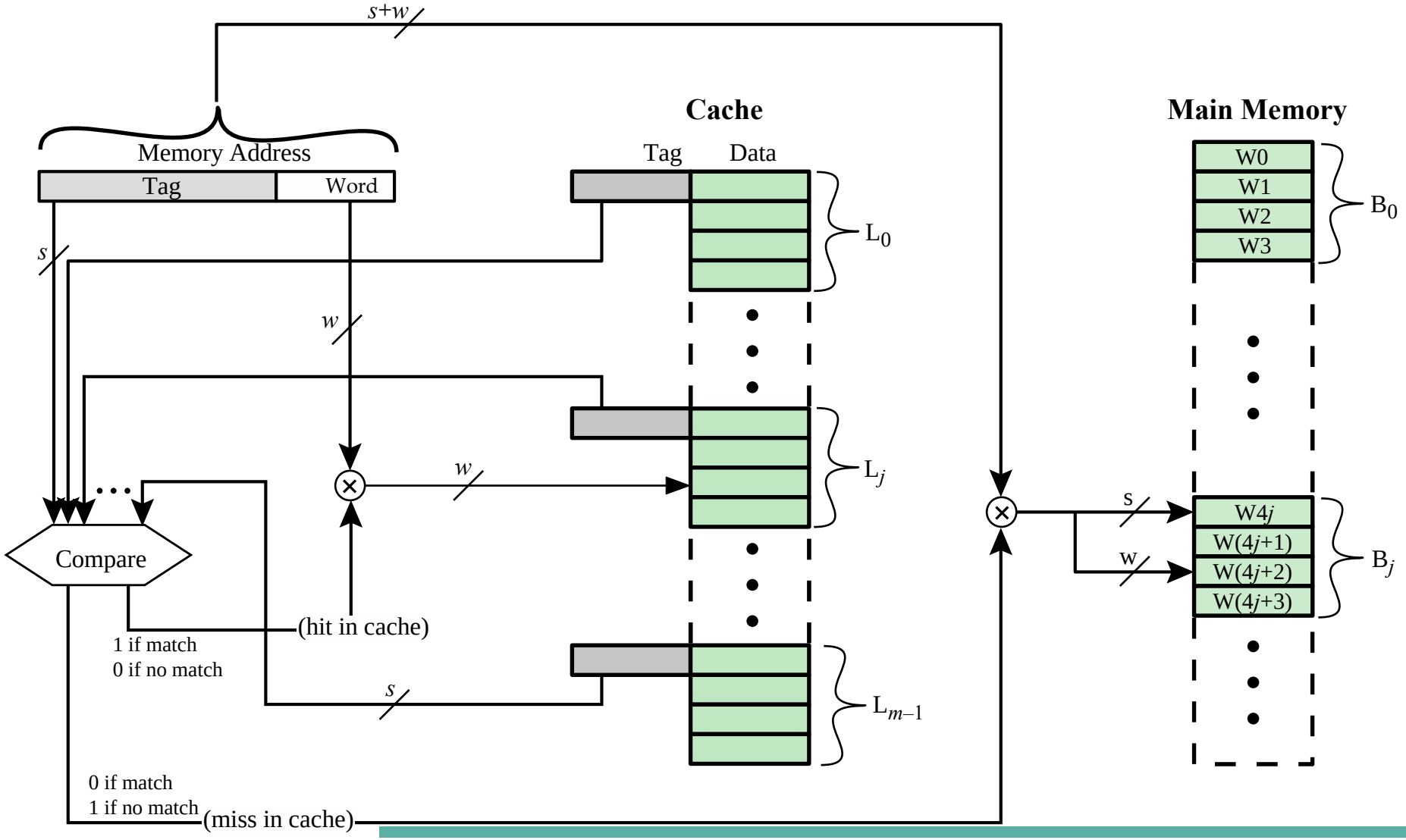
- Cons:
  - Fixed cache location for any given block
    - If a program accesses 2 blocks that map to the same line repeatedly, blocks will be continually swapped in the cache, cache misses are very high (**thrashing**)

Associative Mapping

- A main memory block can load into any line of cache
- Memory address is interpreted as tag and word
- Tag uniquely identifies block of memory
- Every line's tag is examined for a match
- Cache searching becomes expensive



Fully Associative Cache Organization



Associative Mapping Address Structure

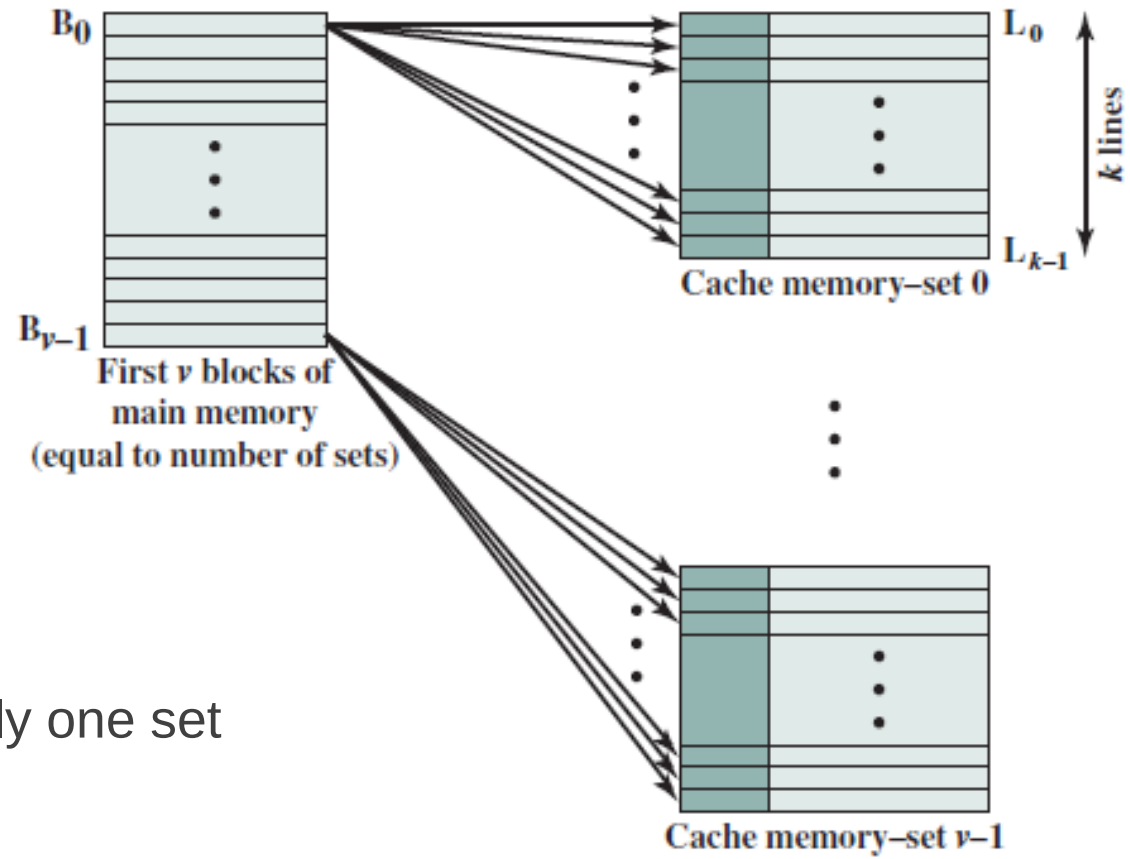
- 22 bit tag stored with each 32 bit block of data
- Compare tag field with tag entry in cache to check for hit

Associative Mapping Summary

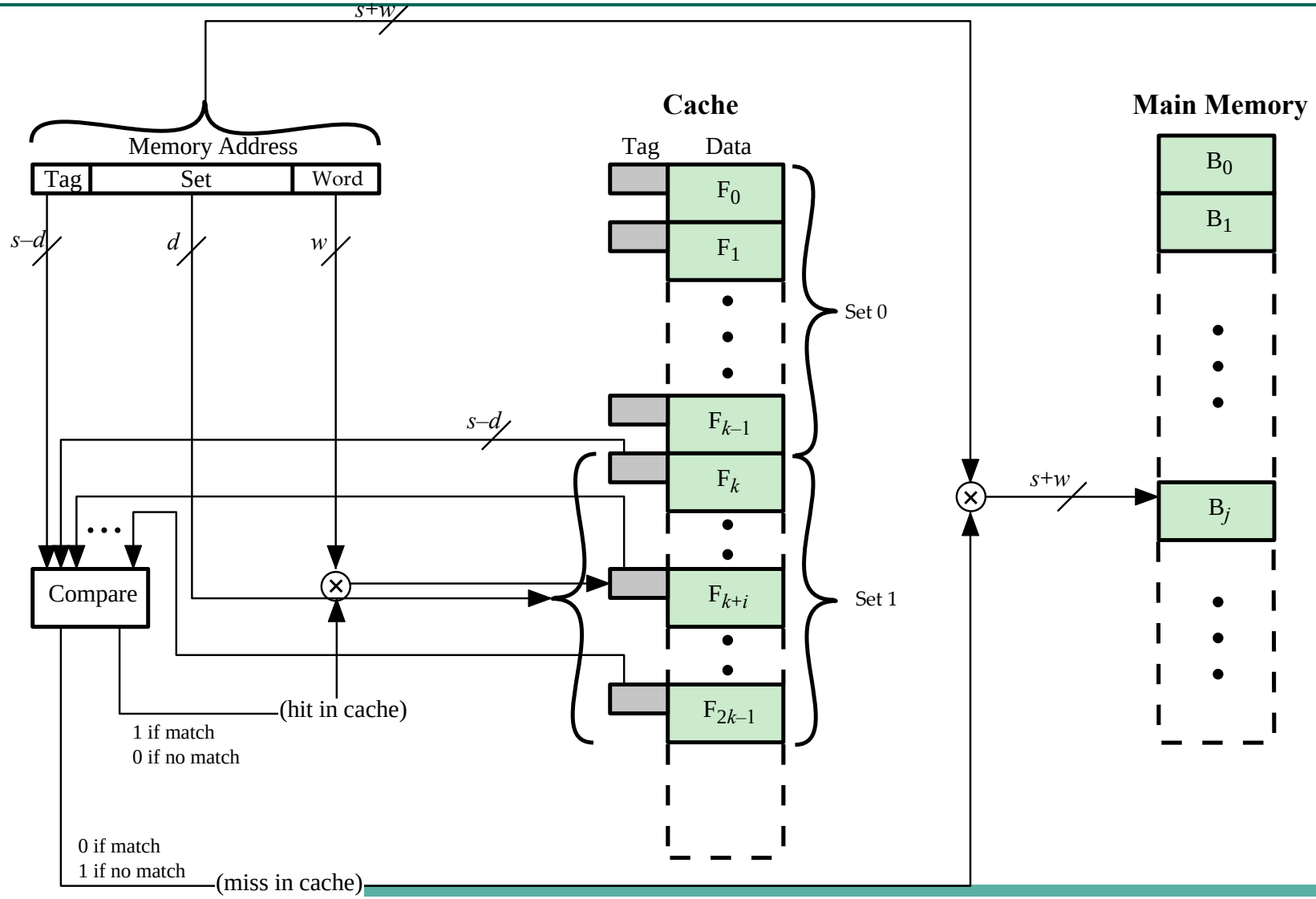
- Address length = (s + w) bits
- Number of addressable units = 2^{s+w} words or bytes
- Block size = line size = 2^{w} words or bytes
- Number of blocks in main memory = $2^{s+ w}/2^{w} = 2^{s}$
- Number of lines in cache = undetermined
- Size of tag = s bits

Set Associative Mapping

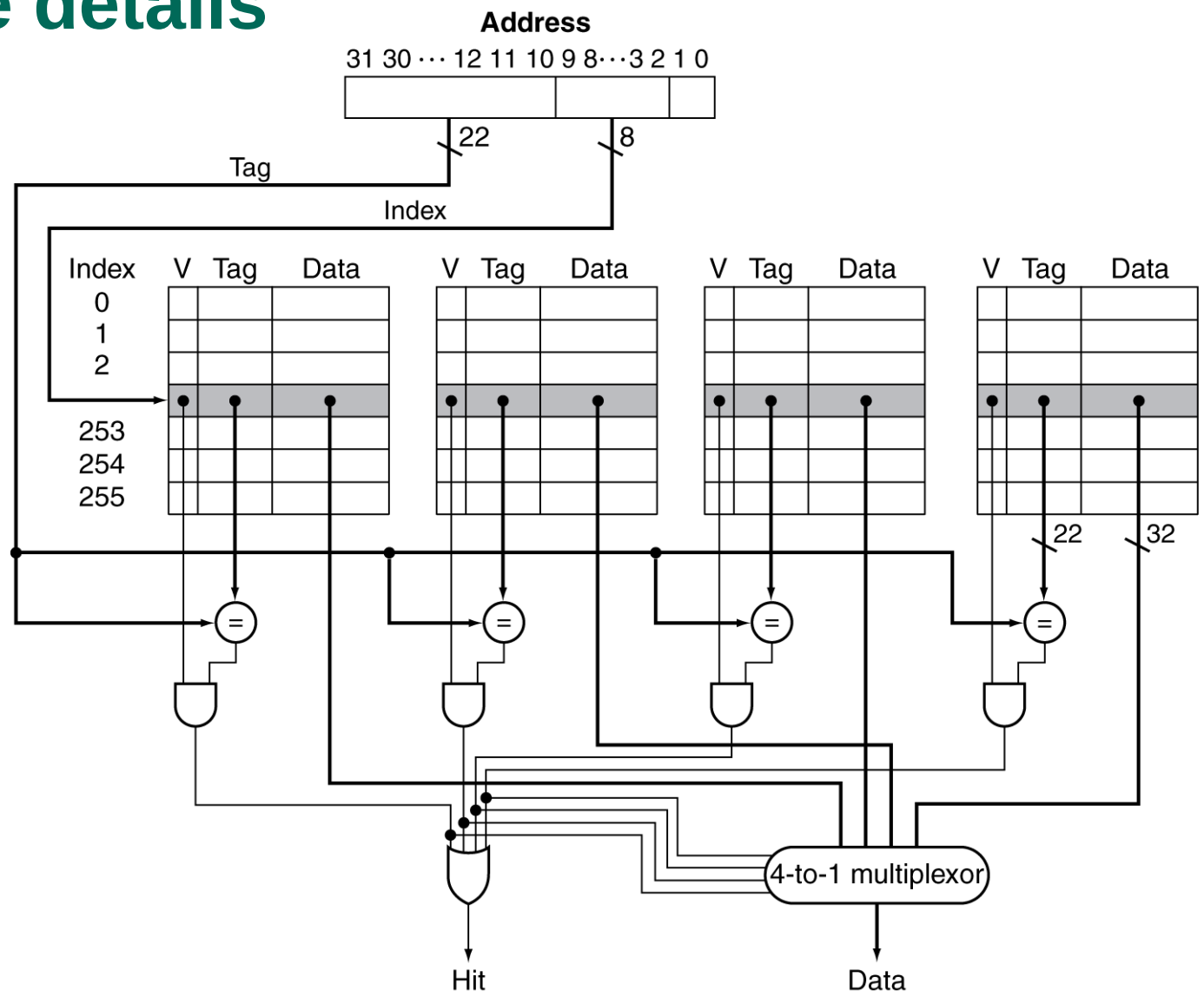
- Compromise that exhibits the strengths of both the direct and associative approaches while reducing their disadvantages
- Cache is divided into a number of **sets**
- Each set contains a number of lines **k**
- A given block maps to any line in a given set
- Block number determines which set
  (Block number) modulo (#Sets in cache)
  - Search all entries in a given set at once
  - ***k* comparators** (less expensive)
- e.g. 2 lines per set
  - 2 way associative mapping
  - A given block can be in one of 2 lines in only one set



K-Way Set Associative Cache Organization



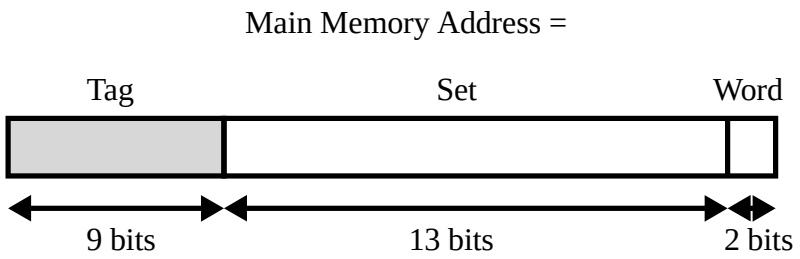
4 way Set Associative Cache Organization – more details



Set Associative Mapping Address Structure

- Memory address has three fields: Tag, Set, and Word.
- Use set field to determine cache set to look in
- Compare tag field to see if we have a hit
- 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

- For 2-way set associative:



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

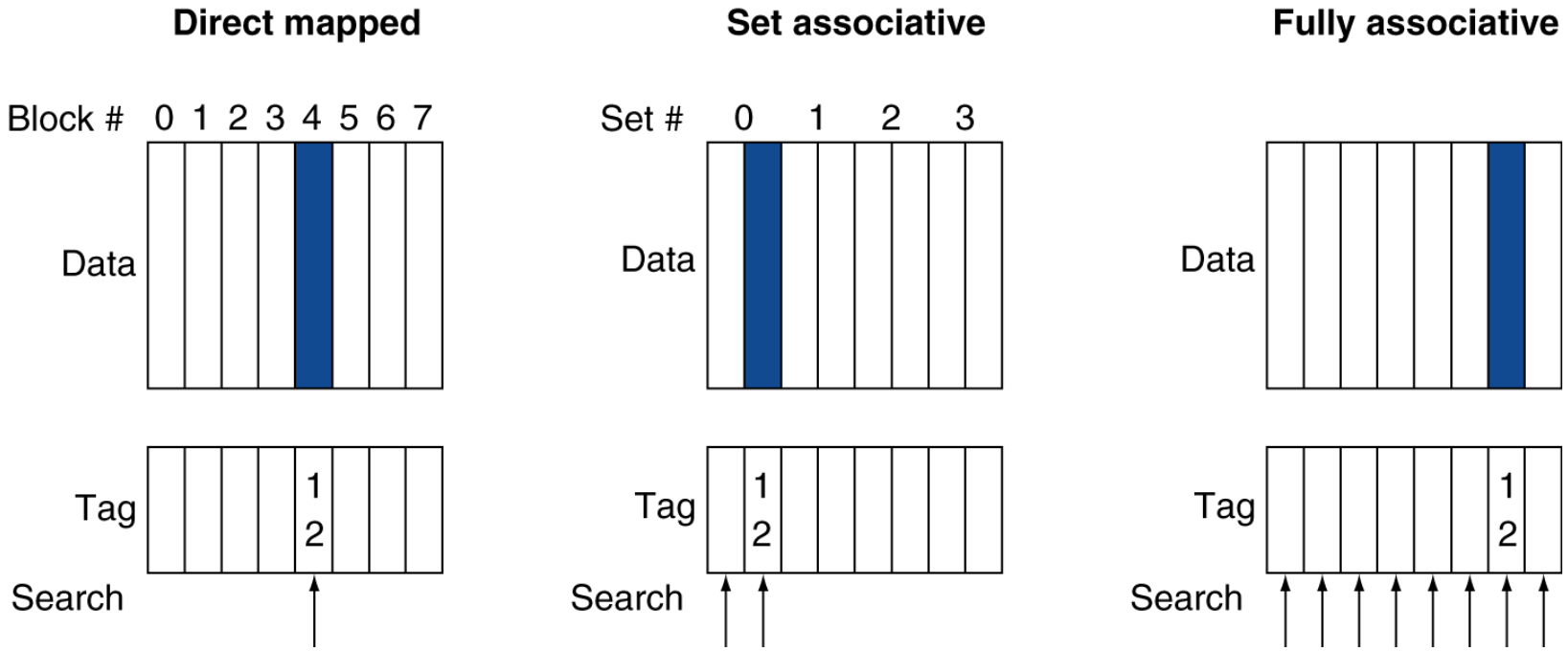
$$m = v \times k$$
$$i = j \text{ modulo } v$$

where

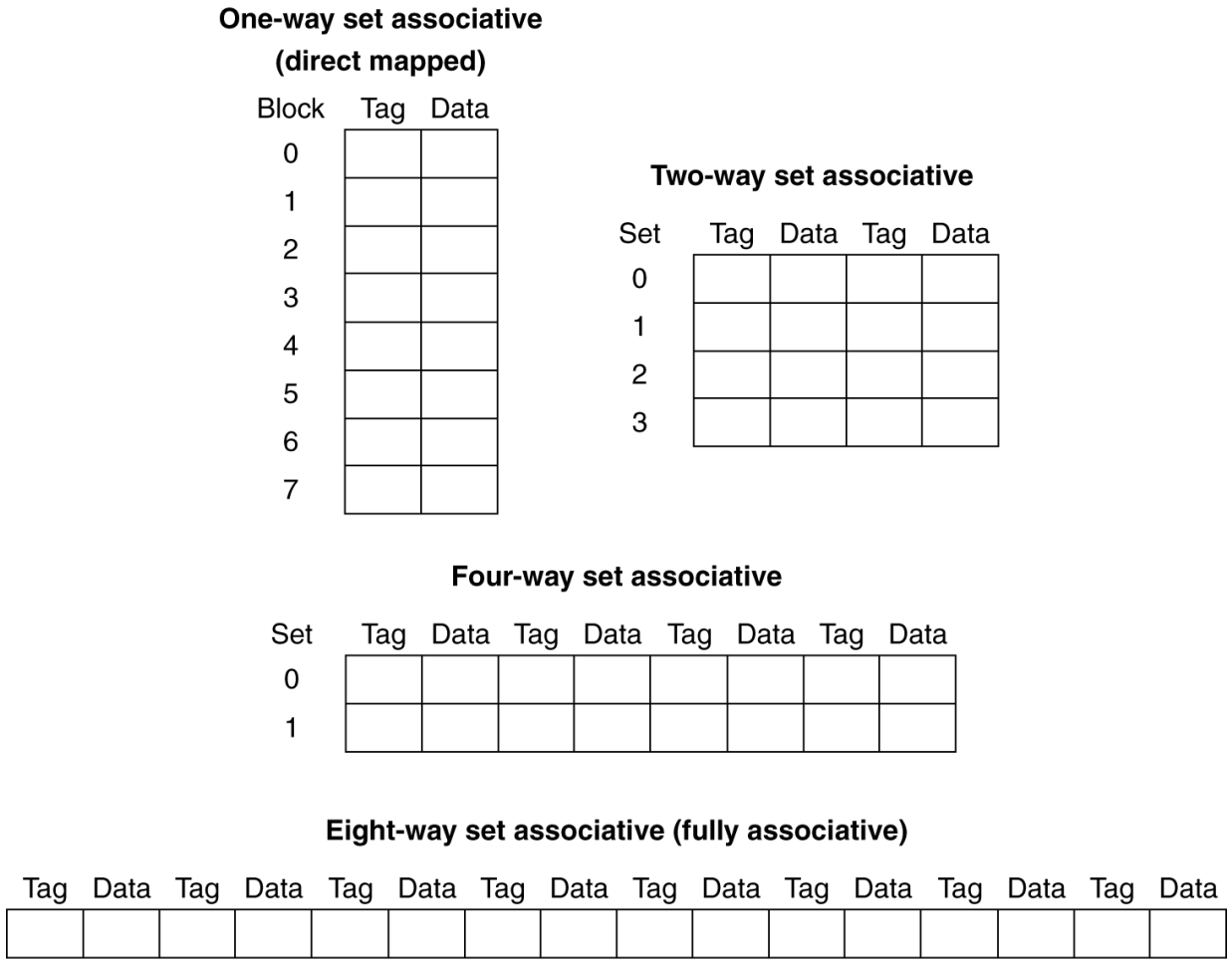
i = cache set number
j = main memory block number
m = number of lines in the cache
v = number of sets
k = number of lines in each set

Comparison Example

- Direct mapped: 1-way set associative 12%8 = 4
- Set-associative: 2-way; 4 sets: 12%4 = 0; anywhere in set 0
- Fully associative: anywhere in the 8 lines



Spectrum of Associativity - for a cache with 8 entries



Associativity Example

- Compare 4-block caches
  - Direct mapped, 2-way set associative, fully associative
  - Each cache is 4-1 word line
  - Block access sequence: 0, 8, 0, 6, 8

- Direct mapped

Block address	Line	Hit/miss	Cache content after access			
			0	1	2	3
0	0	miss	**Mem[0]**			
8	0	miss	**Mem[8]**			
0	0	miss	**Mem[0]**			
6	2	miss	Mem[0]		**Mem[6]**	
8	0	miss	**Mem[8]**		Mem[6]	

Associativity Example

- Fully associative

Block address		Hit/miss	Cache content after access			
0		miss	**Mem[0]**			
8		miss	Mem[0]	**Mem[8]**		
0		hit	**Mem[0]**	Mem[8]		
6		miss	Mem[0]	Mem[8]	**Mem[6]**	
8		hit	Mem[0]	**Mem[8]**	Mem[6]	

- 2-way set associative
  - 2 elements per set and 2 sets (indices 0 and 1)

Block address	Line	Hit/miss	Cache content after access			
			Set 0		Set 1	
0	0	miss	**Mem[0]**			
8	0	miss	Mem[0]	**Mem[8]**		
0	0	hit	**Mem[0]**	Mem[8]		
6	0	miss	Mem[0]	**Mem[6]**		
8	0	miss	**Mem[8]**	Mem[6]		

replace the LRU block within a set

Replacement Algorithms

- Once the cache has been filled, when a new block is brought into the cache, one of the existing blocks must be **replaced**
- For direct mapping there is only one possible line for any particular block and no other choice is possible
- For the associative and set-associative techniques a replacement algorithm is needed
- To achieve high speed, an algorithm must be implemented in hardware

Replacement Algorithms

- Least recently used (**LRU**)
  - Most effective
  - Replace that block in the set that has been in the cache longest with no reference to it
  - Because of its simplicity of implementation, LRU is the most popular replacement algorithm
- First-in-first-out (**FIFO**)
  - Replace that block in the set that has been in the cache longest
  - Easily implemented as a round-robin or circular buffer technique
- Least frequently used (**LFU**)
  - Replace that block in the set that has experienced the fewest references
  - Could be implemented by associating a counter with each line
- **Random**

Write Policy

- When a block in cache is to be replaced, there are two cases to consider:
  - If the block in the cache has not been altered, then it may be overwritten with a new block without first writing out the old block.
  - 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.

Write Policy

- Two problems:
  - More than one device may have access to main memory
    - e.g., 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.
    - If the I/O device has altered main memory, then the cache word is invalid.
  - When multiple processors are attached to the same bus and each processor has its own local cache.
    - If a word is altered in one cache, it could invalidate a word in other caches.

Write Through

- On data-write hit, could just update the block in cache
  - But then cache and memory would be inconsistent
- Write through: also update memory
  - Simplest technique
  - All write operations are made to main memory as well as to the cache
- Main disadvantage: generates substantial memory traffic and may create a bottleneck

- Makes writes take longer
  - e.g., if base CPI = 1, 10% of instructions are stores, write to memory takes 100 cycles
    - Effective CPI = 1 + 0.1×100 = 11
- Solution: write buffer
  - Holds data waiting to be written to memory
  - CPU continues immediately
    - Only stalls on write if write buffer is already full

Write Back

- Alternative: On data-write hit, just update the block in cache
  - Keep track of whether each block is **dirty**
- Minimizes memory writes
- When a dirty block is replaced
  - Write it back to memory
- Portions of main memory are invalid and hence accesses by I/O modules can be allowed only through the cache
  - complex circuitry and a potential bottleneck

Line Size

- When a block of data is retrieved and placed in the cache not only the desired word but also some number of adjacent words are retrieved.
- As the block size increases the hit ratio will at first increase because of the principle of locality.
- As the block size increases more useful data are brought into the cache.
- The hit ratio will begin to decrease as the block becomes bigger and the probability of using the newly fetched information becomes less than the probability of reusing the information that has to be replaced.
- Two specific effects come into play:
  - Larger blocks reduce the number of blocks that fit into a cache.
  - As a block becomes larger each additional word is farther from the requested word.

Unified vs Split Caches

- Split cache:
  - One dedicated to instructions
  - One dedicated to data
  - Both exist at the same level, typically as two L1 caches
- Advantages of unified cache:
  - Higher hit rate
    - Balances load between instruction and data fetches automatically
  - Only one cache needs to be designed and implemented
- Advantages of split cache:
  - Eliminates cache contention between instruction fetch/decode unit and execution unit
    - Important in pipelining
- Trend is toward split caches at the L1 and unified caches for higher levels

Measuring Cache Performance

- Components of CPU time
  - Program execution cycles
    - Includes cache hit time
  - Memory stall cycles
    - Mainly from cache misses

- With simplifying assumptions:

$$\text{Memory stall cycles}$$

$$= \frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty}$$

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}$$

Cache Performance Example

- Assume a processor with separate instruction and data caches
  - ideal CPI is 3 clock cycles
  - memory access is 125 cycles
  - I-cache miss rate is 4%
  - D-cache miss rate is 8%
  - 60% of instructions are loads or stores

- What is the processor's actual CPI?

$$\mathbf{CPI_{actual} = CPI_{base} + (instruction\ miss\ cycles) + (data\ miss\ cycles)}$$

$$\mathbf{= 3 + (0.04)(125) + (0.6)(0.08)(125) = 14}$$

Cache Performance Example

- Given
  - I-cache miss rate = 2%
  - D-cache miss rate = 4%
  - Miss penalty = 100 cycles
  - Base CPI (ideal cache) = 2
  - Load & stores are 36% of instructions

- Miss cycles per instruction
  - I-cache: 0.02 × 100 = 2
  - D-cache: 0.36 × 0.04 × 100 = 1.44

- Actual CPI = 2 + 2 + 1.44 = 5.44
  - Ideal CPI is 5.44/2 = 2.72 times faster

Average Access Time

- Hit time is also important for performance

- Average memory access time (**AMAT**)
  - AMAT = Hit time + Miss rate × Miss penalty

- Example
  - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, I-cache miss rate = 5%
  - AMAT = 1 + 0.05 × 20 = 2 clock cycles
    - Or 2 ns

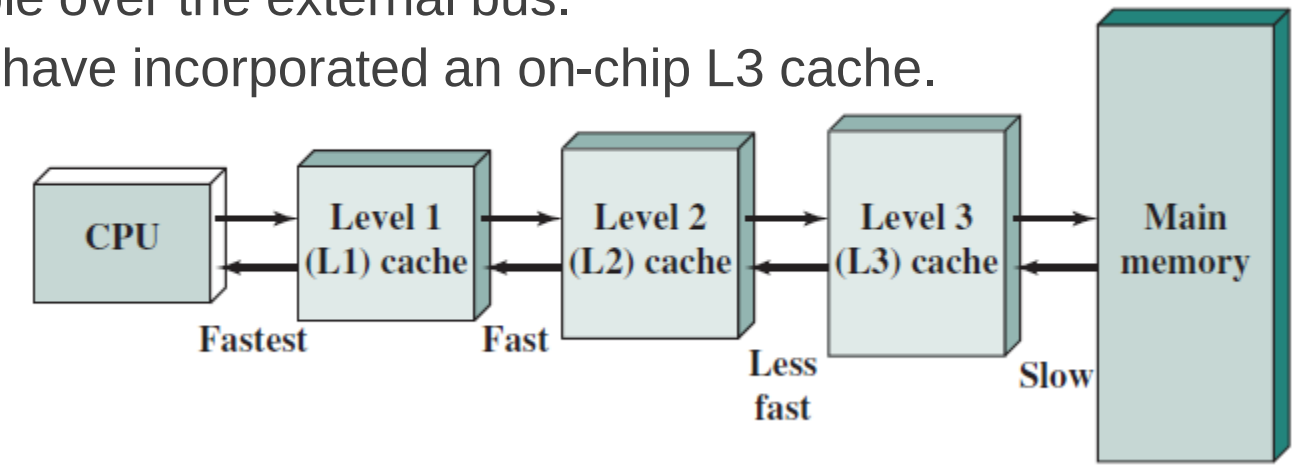
Multi-Level Caches

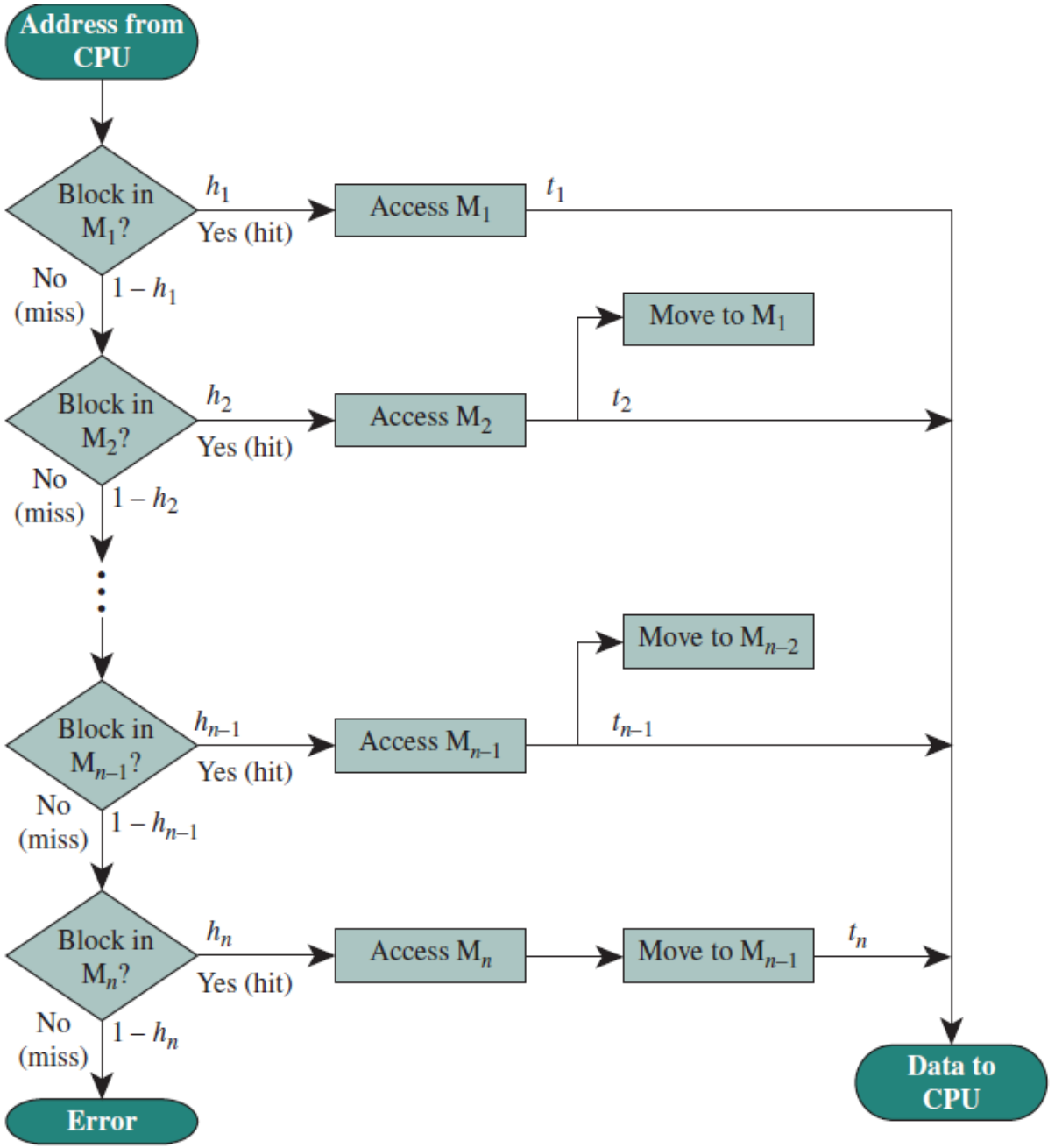
- As logic density has increased it has become possible to have a cache on the same chip as the processor
- The on-chip cache reduces the processor's external bus activity and speeds up execution time and increases overall system performance
  - When the requested instruction or data is found in the on-chip cache, the bus access is eliminated
  - On-chip cache accesses will complete appreciably faster than would even zero-wait state bus cycles
  - During this period the bus is free to support other transfers
- Two-level cache:
  - Internal cache designated as level 1 (L1)
  - External cache designated as level 2 (L2)
- Potential savings due to the use of an L2 cache depends on the hit rates in both the L1 and L2 caches

- The use of multilevel caches complicates all of the design issues related to caches, including size, replacement algorithm, and write policy

Multi-Level Caches

- Primary cache attached to CPU
  - Small, but fast
- Level-2 cache services misses from primary cache
  - Larger, slower, but still faster than main memory
- Main memory services L2 cache misses
- Most contemporary microprocessors have moved the L2 cache onto the processor chip and added an L3 cache
  - Originally, the L3 cache was accessible over the external bus.
  - More recently, most microprocessors have incorporated an on-chip L3 cache.




Address from CPU
Block in M_1?
h_1
Yes (hit)
Access M_1
t_1
No (miss)
$1 - h_1$
Block in M_2?
h_2
Yes (hit)
Access M_2
Move to M_1
t_2
No (miss)
$1 - h_2$
Block in M_{n-1}?
h_{n-1}
Yes (hit)
Access M_{n-1}
Move to M_{n-2}
t_{n-1}
No (miss)
$1 - h_{n-1}$
Block in M_n?
h_n
Yes (hit)
Access M_n
Move to M_{n-1}
t_n
No (miss)
$1 - h_n$
Error
Data to CPU

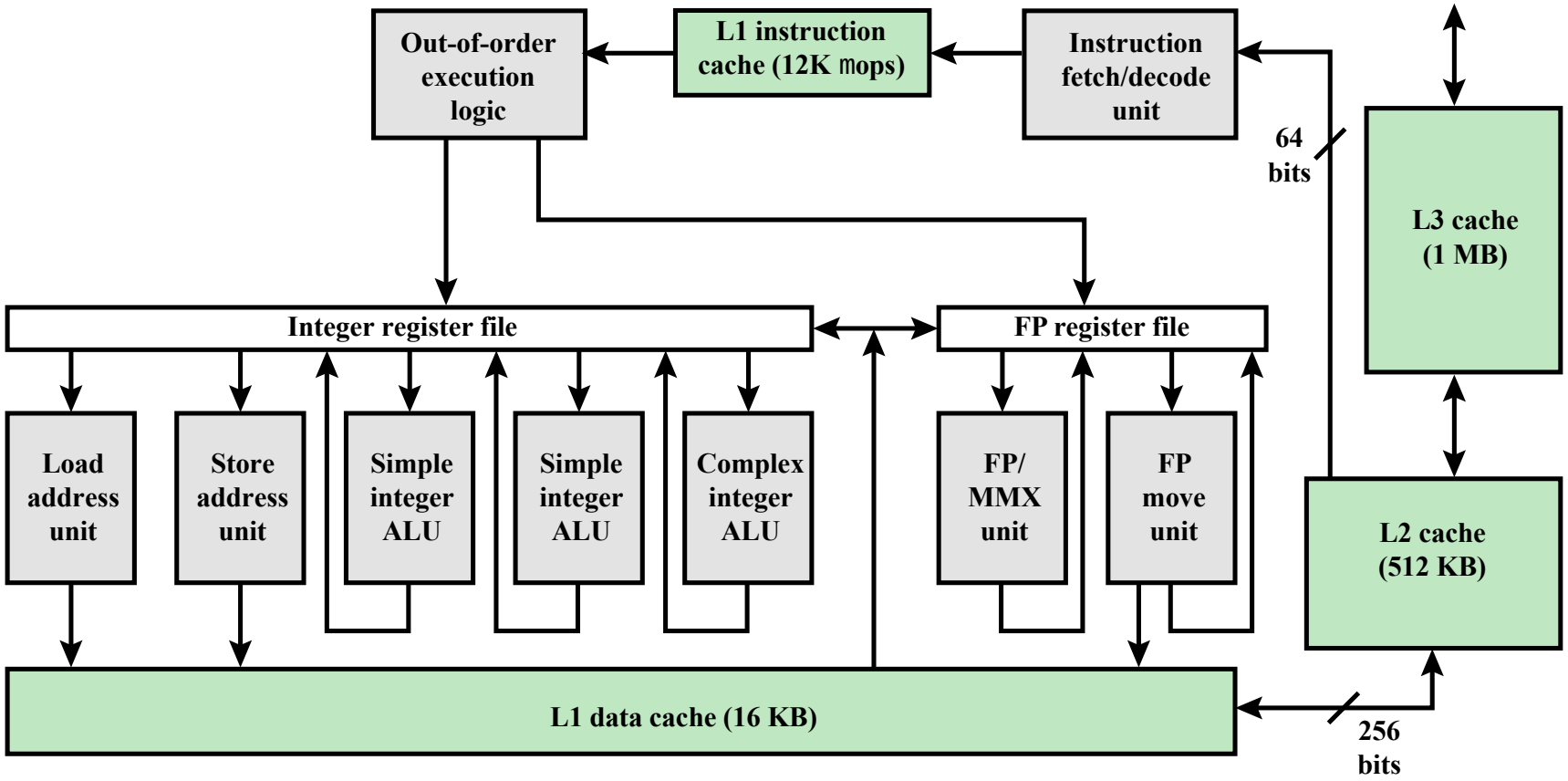
Multi-Level Cache Example

- Given
  - CPU base CPI = 1
  - Clock rate = 4GHz
  - Miss rate/instruction = 2%
  - Main memory access time = 100ns
- With just primary cache
  - Miss penalty = 100ns/0.25ns = 400 cycles
  - Effective CPI = 1 + 0.02 × 400 = 9
- Now add L2 cache
  - Access time = 5ns
  - Global miss rate to main memory = 0.5%
- Primary miss with L2 hit
  - Penalty = 5ns/0.25ns = 20 cycles
- CPI = 1 + 0.02 × 20 + 0.005 × 400 = 3.4
- Performance ratio = 9/3.4 = 2.6

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



Exercise

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

- How is a 16-bit memory address divided into tag, line number, and byte number?
- Into what line would bytes with each of the following addresses be stored?
  - 0001 0001 0001 1011
  - 1100 0011 0011 0100
  - 1101 0000 0001 1101
  - 1010 1010 1010 1010
- Suppose the byte with address 0001 1010 0001 1010 is stored in the cache. What are the addresses of the other bytes stored along with it?
- How many total bytes of memory can be stored in the cache?
- Why is the tag also stored in the cache?

Exercise

- How many total bits are required for a direct-mapped cache with 16 KB of data and 4-word lines, assuming a 32-bit address?

- Include the tag and valid bits in your calculations.

Exercise

- Assuming a cache of 4096 lines, a 4-word line size, and a 32-bit address, find the number of set bits and the number of tag bits for caches that are direct mapped, two-way and four-way set associative, and fully associative.