Cache
Lecture Content

- **Memory in General**
- **Memory Hierarchy**
- **Caches**

Cache
Learning Objectives

- Understand the concept and intention of caches
- Understand the key concepts of cache design
- Distinguish different forms of cache and understand their pros and cons
- Understand the demand for multiple cache levels and performance implications of cache design

Computer Memory

- Although seemingly simple in concept, memory exhibits perhaps the widest range of
  - type
  - technology
  - organization
  - performance
  - cost
- No single technology is the optimal for all requirements
- Consequently, computers have a hierarchy of memory subsystems

- Today, we will discuss an essential memory: cache memory

Key Characteristics of Computer Memory Systems

- **Location**
  - Internal
  - External
- **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/non-erasable
- **Organization**
  - Memory modules

Characteristics of Memory Systems

- **Location**
  - Refers to whether memory is internal and external to the computer
  - Internal memory is often equated with main memory
  - Processor requires its own local memory, in the form of registers
  - Cache is another form of internal memory
  - External memory consists of peripheral storage devices that are accessible to the processor via I/O controllers
- **Capacity**
  - Memory is typically expressed in terms of bytes

Characteristics of Memory Systems

- **Unit of transfer**
  - Often the number of electrical lines into and out of the memory module
  - The number of bits read out of or written into memory at a time

Characteristics of Memory Systems

- **Unit of transfer**
  - Often the number of electrical lines into and out of the memory module
  - The number of bits read out of or written into memory at a time

  - **Word**: The “natural” unit of organization of memory.
    - Typically equal to the number of bits used to represent an integer and to the instruction length.
    - Many Exceptions, e.g., Intel has an huge variety of instruction lengths
    - In this lecture, we consider a word being either 32 Bit or 64 Bit

Characteristics of Memory Systems

- **Unit of transfer**
  - Often the number of electrical lines into and out of the memory module
  - The number of bits read out of or written into memory at a time

  - **Word**: The "natural" unit of organization of memory.
    - Typically equal to the number of bits used to represent an integer and to the instruction length
    - Many Exceptions, e.g., Intel has an huge variety of instruction lengths
    - In this lecture, we consider a word being either 32 Bit or 64 Bit

  - **Addressable Units**: The unit which can be directly addressed
    - In some systems, the addressable unit is the word
    - Often it is the byte
    - Havening A bit addresses, the number of addressable units N is 2^A

Methods of Accessing Units of Data

- Sequential Access
- Direct Access
- Random Access
- Associative Access

Methods of Accessing Units of Data

- **Sequential Access**
  - Memory is organized into units of data, called records
  - Access must be made in a specific linear sequence
  - A shared read–write mechanism is used and this must be moved from its current location to the desired location
  - The time to access an arbitrary record is highly variable
- Direct Access
- Random Access
- Associative Access

Methods of Accessing Units of Data

- Sequential Access
- **Direct Access**
  - Shared read–write mechanism
  - Individual blocks or records have a unique address based on physical location
  - Access is accomplished by direct access to reach a general vicinity plus sequential searching, counting, or waiting to reach the final location.
  - Access time is variable
- Random Access
- Associative Access

Methods of Accessing Units of Data

- Sequential Access
- Direct Access
- **Random Access**
  - Each addressable location in memory has a unique addressing mechanism
  - The access time is independent of the sequence and is constant
- Associative Access

Methods of Accessing Units of Data

- Sequential Access
- Direct Access
- Random Access
- **Associative Access**
  - Similar to random access
  - Word is retrieved based on a portion of its contents rather than its address
  - Access time is constant and independent of location or prior access patterns
  - Often used for Caches

Performance of a Memory

- **Access Time (Latency)**
  - Random-access memory: time between address retrieval and delivery or storage of data
  - Others: Time to position the read-write mechanism
- **Memory Cycle Time**
  - Primarily applied to random-access memory
  - Access time + time before a next read can be served
  - May be required for transients to die out on signal lines or to regenerate data if they are read destructively
- **Transfer Rate**
  - This is the rate at which data can be transferred into or out of the memory
  - For random-access memory, it is equal to 1/(cycle time).

The Most Common Memory Forms

- The most common forms are:
  - Semiconductor memory
  - Magnetic surface memory
  - Optical
  - Magneto-optical

- Several physical characteristics of data storage are important:
  - Volatility
    - Magnetic-surface memories are nonvolatile
    - Semiconductor memory may be either volatile or nonvolatile
  - Non-erasable memory
    - Semiconductor memory of this type is known as read-only memory (ROM)
  - For random-access memory the organization is a key design issue
    - Organization refers to the physical arrangement of bits to form words

Cache
Lecture Content

- Memory in General
- Memory Hierarchy
- Caches

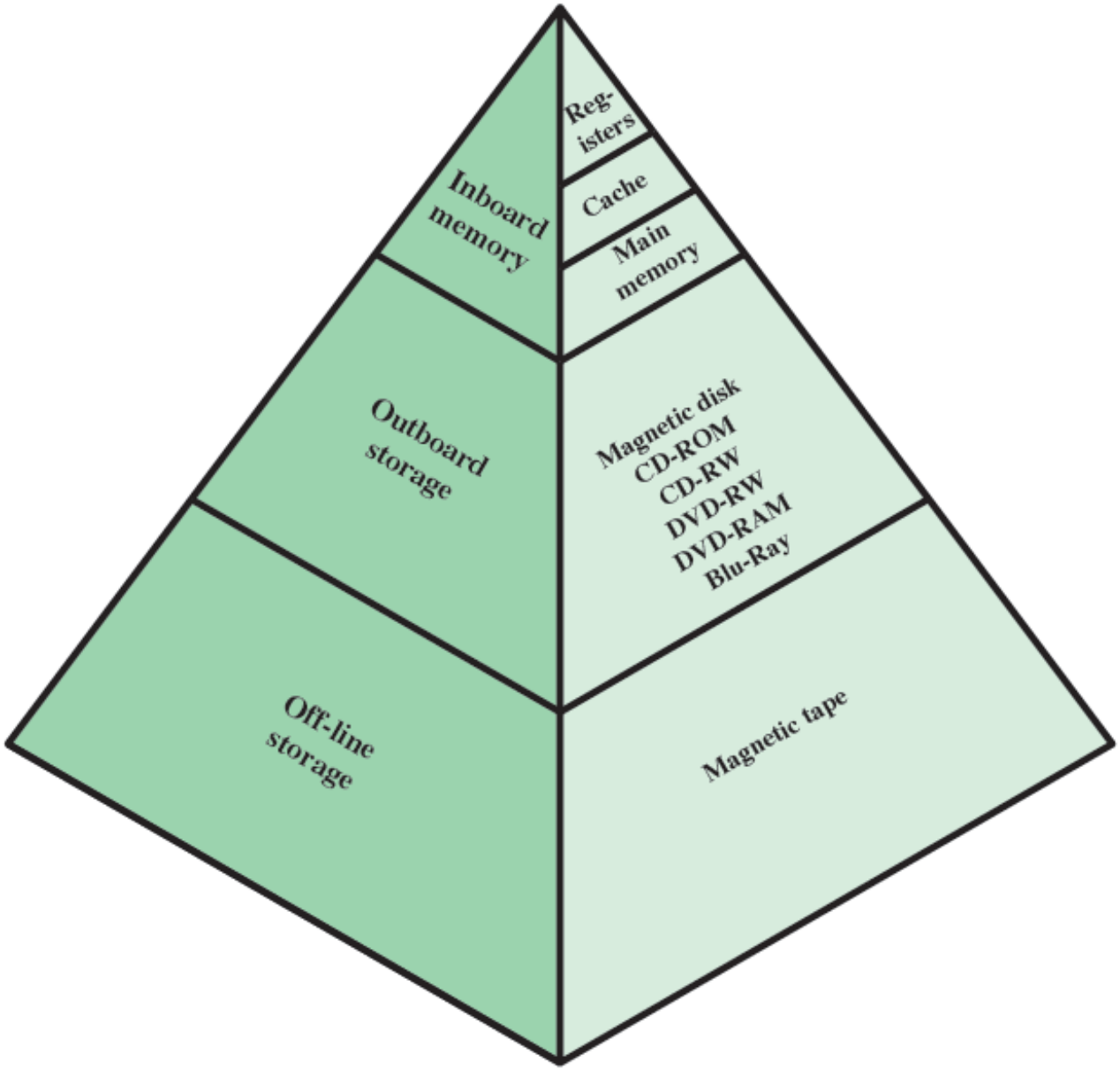
The Memory Dilemma

- Designing computer's memory can be summed up by:
  - 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 Memory Dilemma

- Designing computer's memory can be summed up by:
  - 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:**
  - Not to rely on a single memory component or technology
  - Employ a memory hierarchy

The Memory Hierarchy

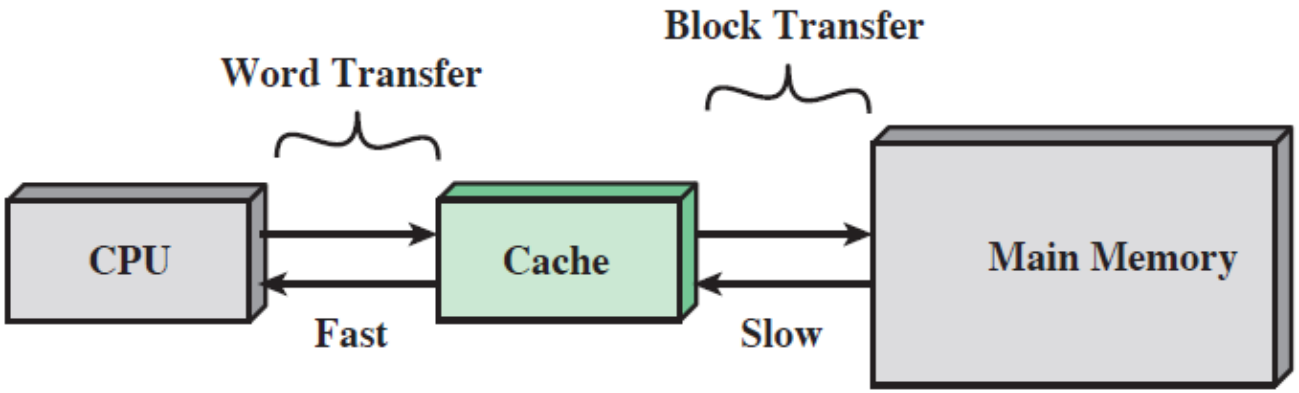


The Memory Hierarchy

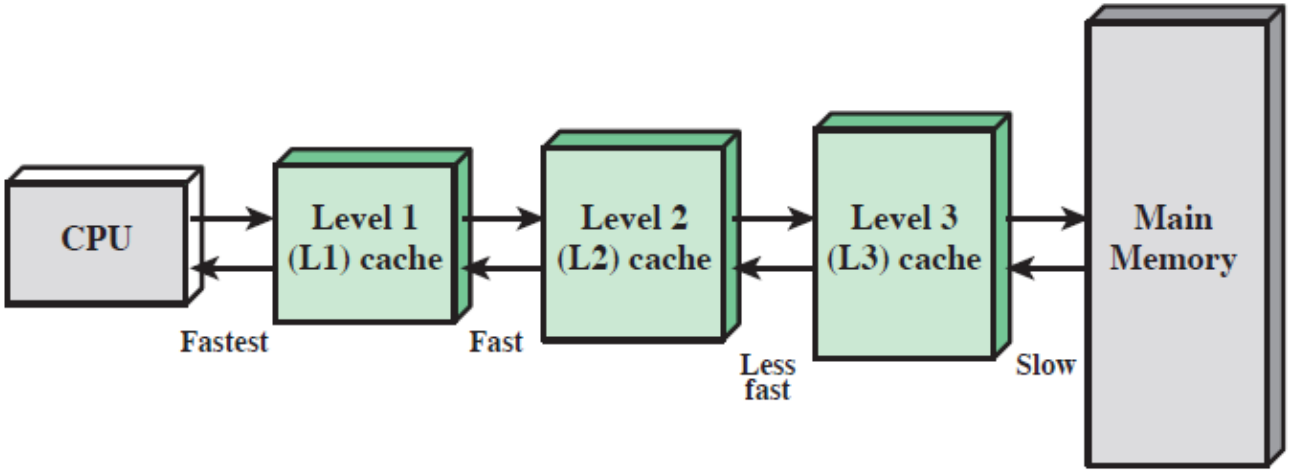
- When you traverse from top to bottom:
  - Decreasing cost per bit
  - Increasing capacity
  - Increasing access time
  - Decreasing frequency of access of the memory by the processor

- The key is the last:
  - Only if this condition hold, the pyramid makes sense
  - Normally it is the case
  - Referred as the locality of reference

A Typical Example

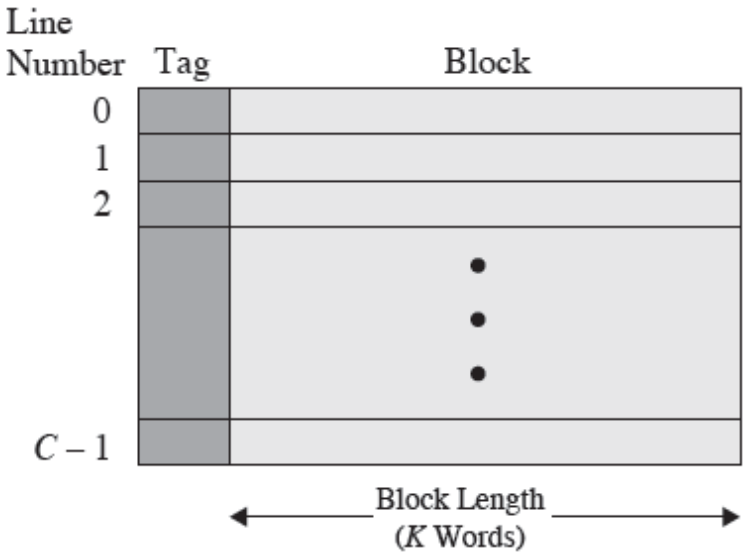


(a) Single cache

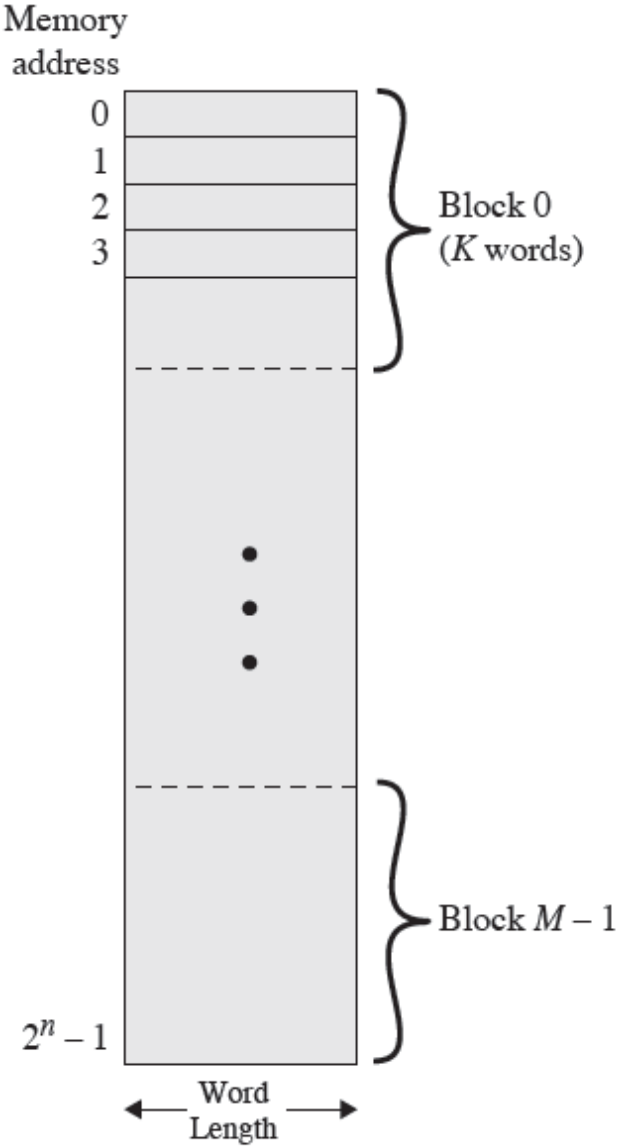


(b) Three-level cache organization

Cache and Main Memory Structure

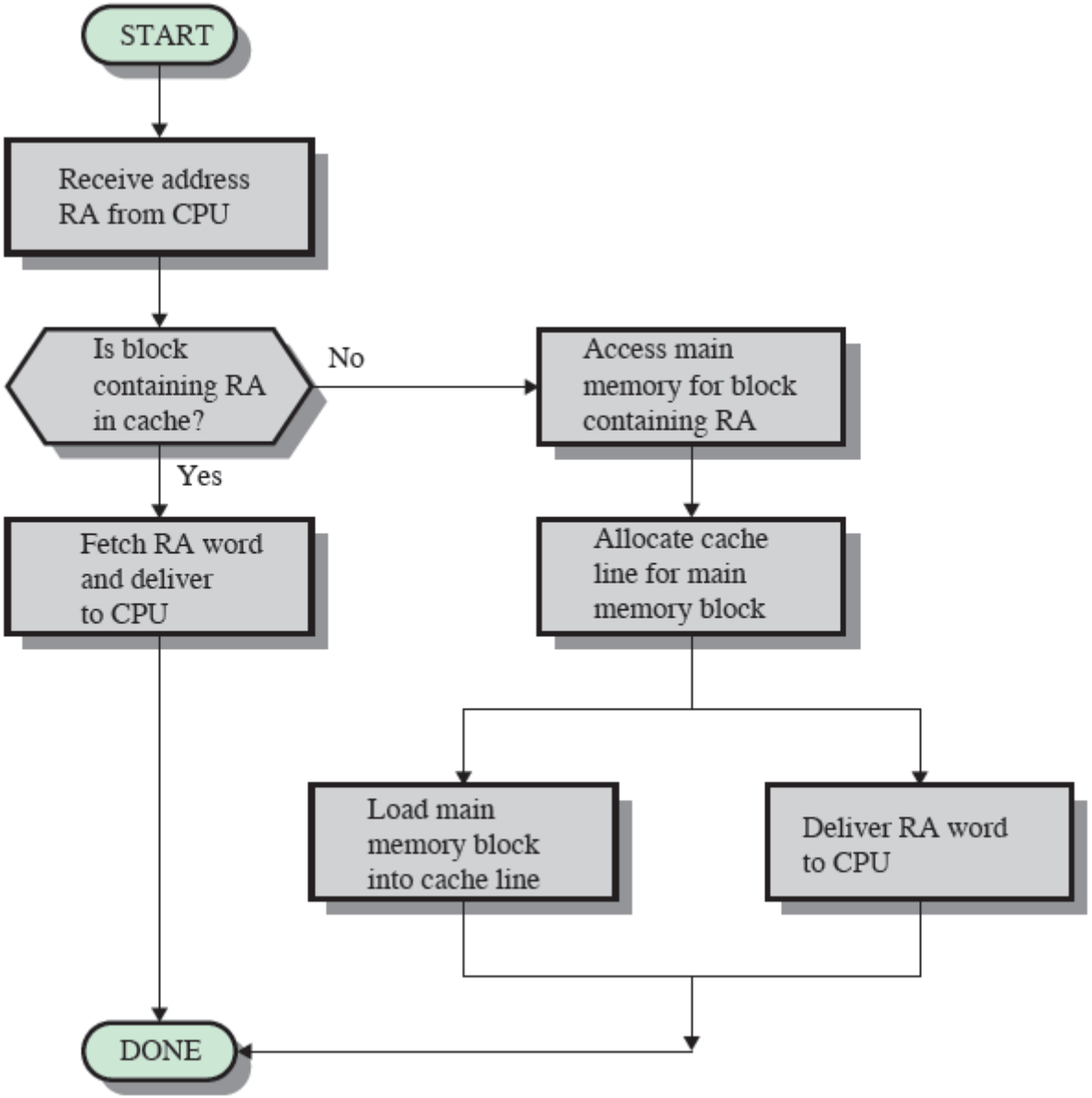


(a) Cache

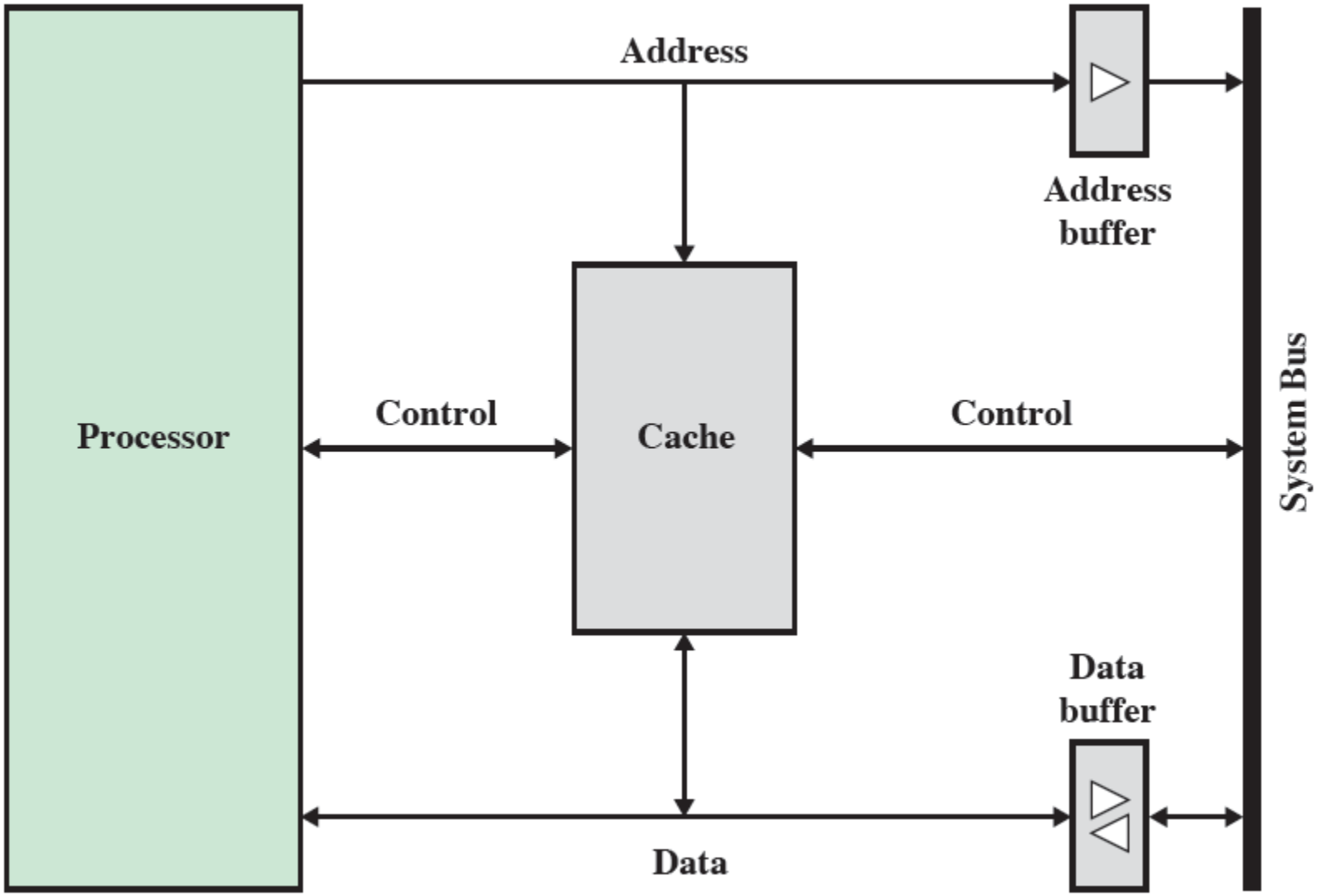


(b) Main memory

Access Flowchart



Typical Realization



Cache
Lecture Content

- Memory in General
- Memory Hierarchy
- Caches

Elements of Cache Design

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

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	—	—
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 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	—
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
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	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

Cache

- Direct
- Associative
- Set Associative

Cache

- **Direct**
  - The simplest technique
  - Maps each block of main memory into only one possible cache line
- Associative
- Set Associative

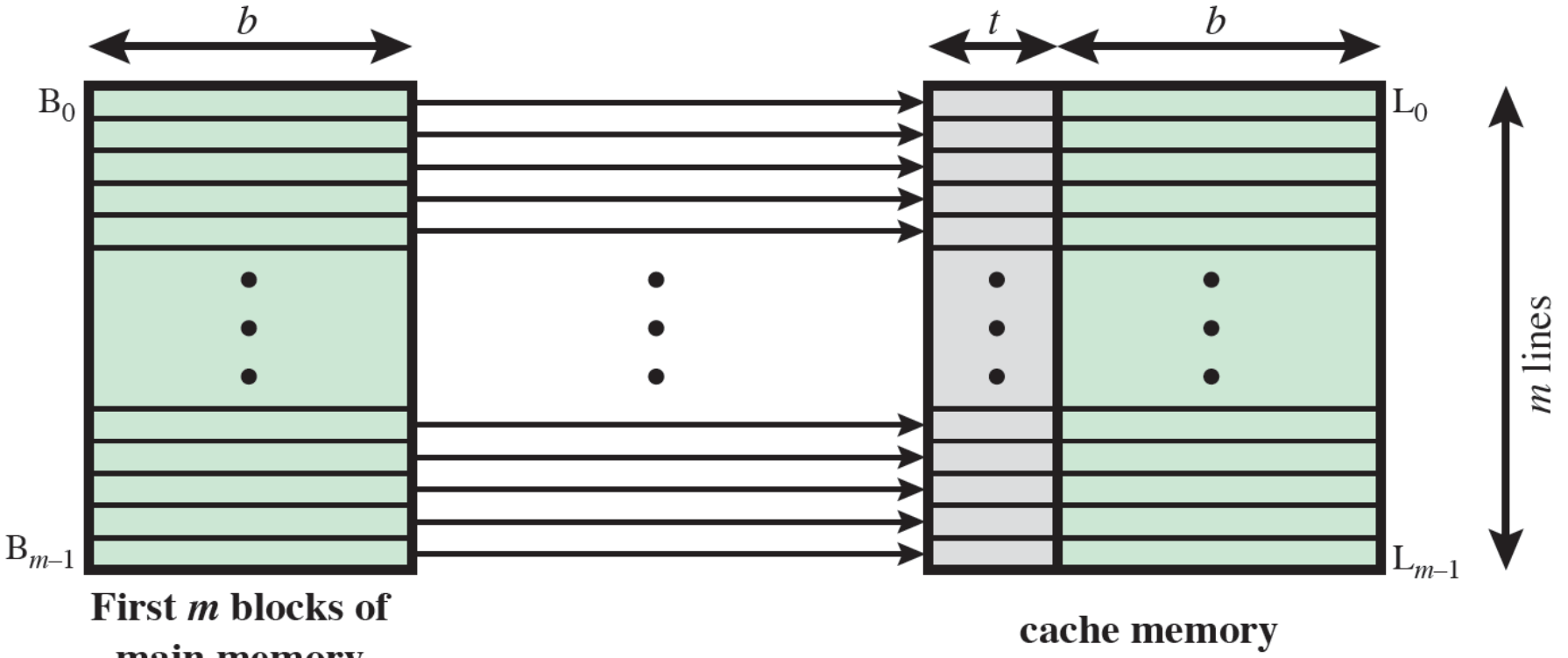
Cache

- Direct
- **Associative**
  - Permits each main memory block to be loaded into any line of the cache
  - The cache control logic interprets a memory address simply as a Tag and a Word field
  - To determine whether a block is in the cache, the cache control logic must simultaneously examine every line's Tag for a match
- Set Associative

Cache

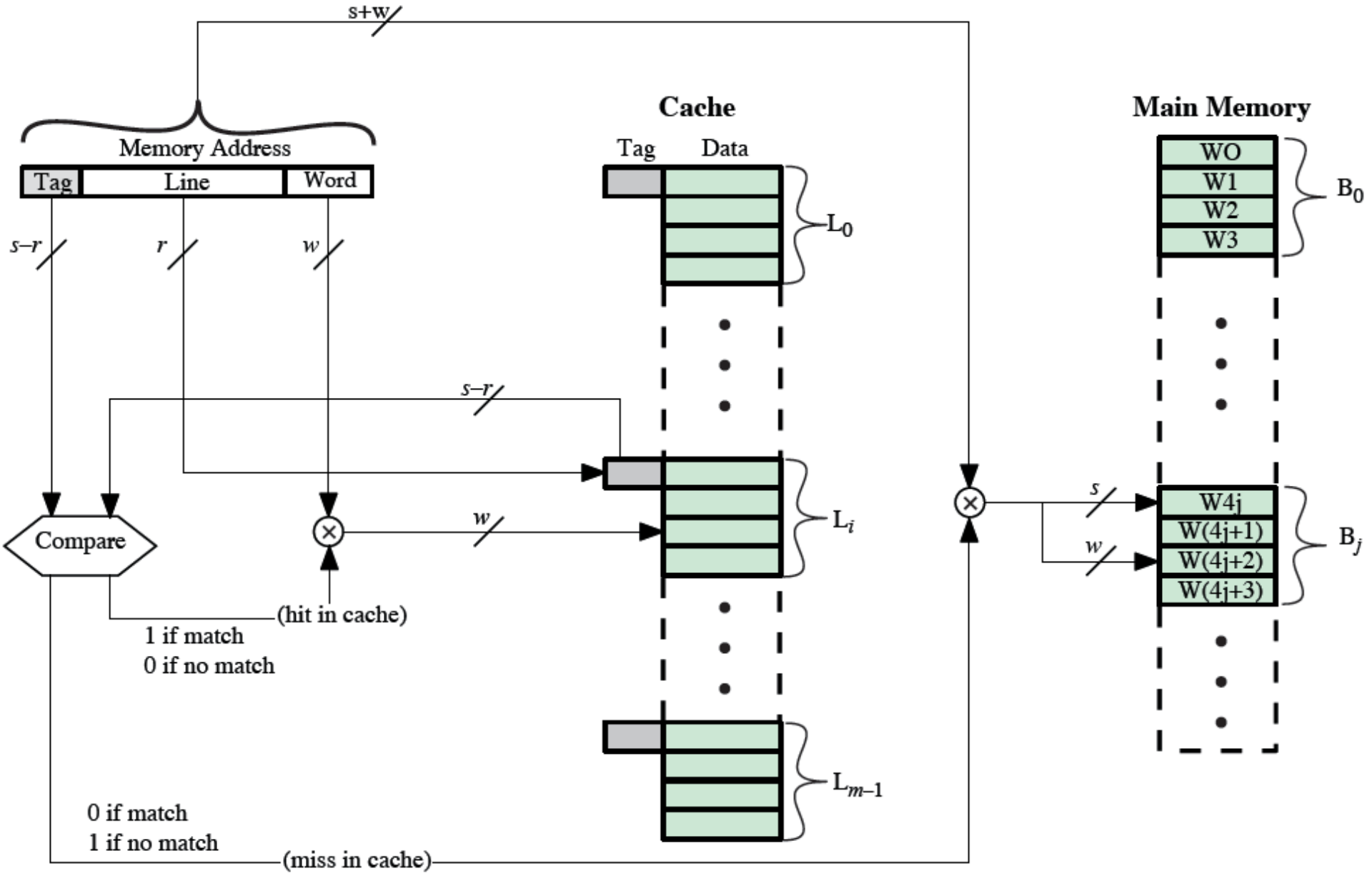
- Direct
- Associative
- **Set Associative**
  - A compromise that exhibits the strengths of both the direct and associative approaches while reducing their disadvantages

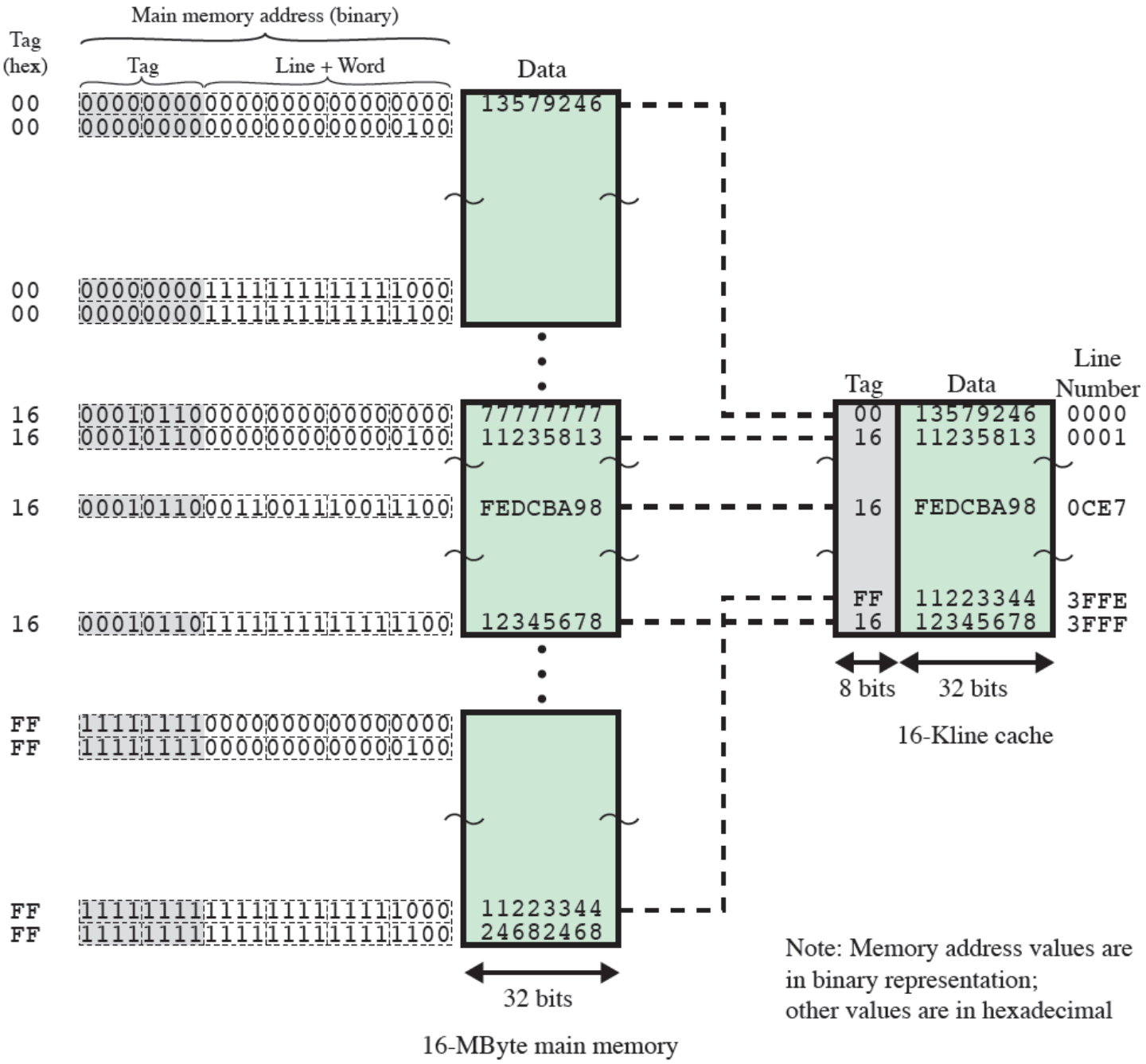
Direct Mapping



b = length of block in bits
t = length of tag in bits

Direct Mapping Cache Organization




Main memory address (binary)
Tag (hex)
Tag
Line + Word
Data
00 00000000000000000000000000
00 00000000000000000000000100
13579246
00 00000000111111111111111000
00 00000000111111111111111100
77777777
16 00010110000000000000000000
16 00010110000000000000000100
11235813
16 00010110001100110011001100
FEDCBA98
16 00010110111111111111111100
12345678
FF 11111111000000000000000000
FF 11111111000000000000000100
11223344
FF 11111111111111111111111000
FF 11111111111111111111111100
24682468
32 bits
16-MByte main memory
Tag
Data
Line Number
00 13579246 0000
16 11235813 0001
16 FEDCBA98 0CE7
FF 11223344 3FFE
16 12345678 3FFF
8 bits
32 bits
16-Kline cache
Note: Memory address values are in binary representation; other values are in hexadecimal

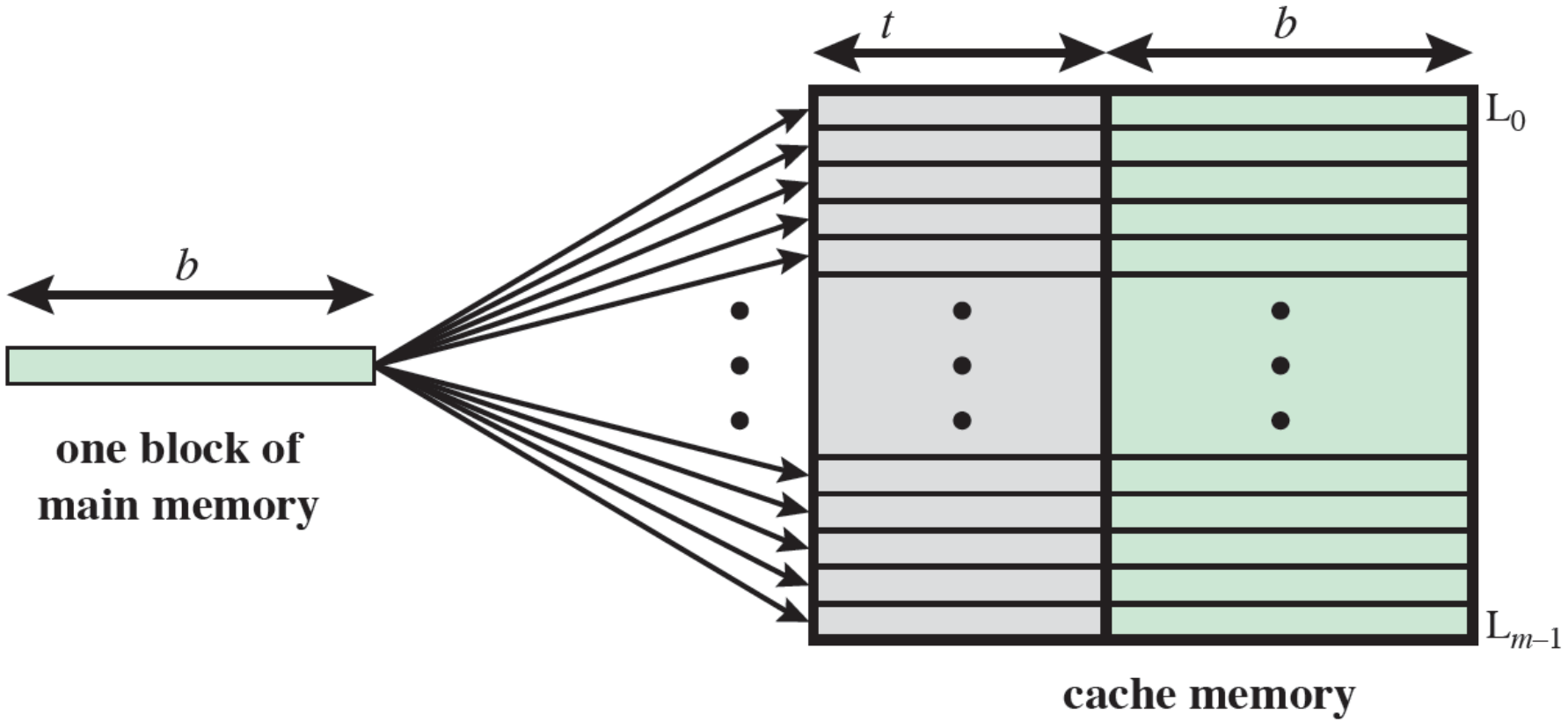
Summary

- 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 tag = (s – r) bits

Victim Cache

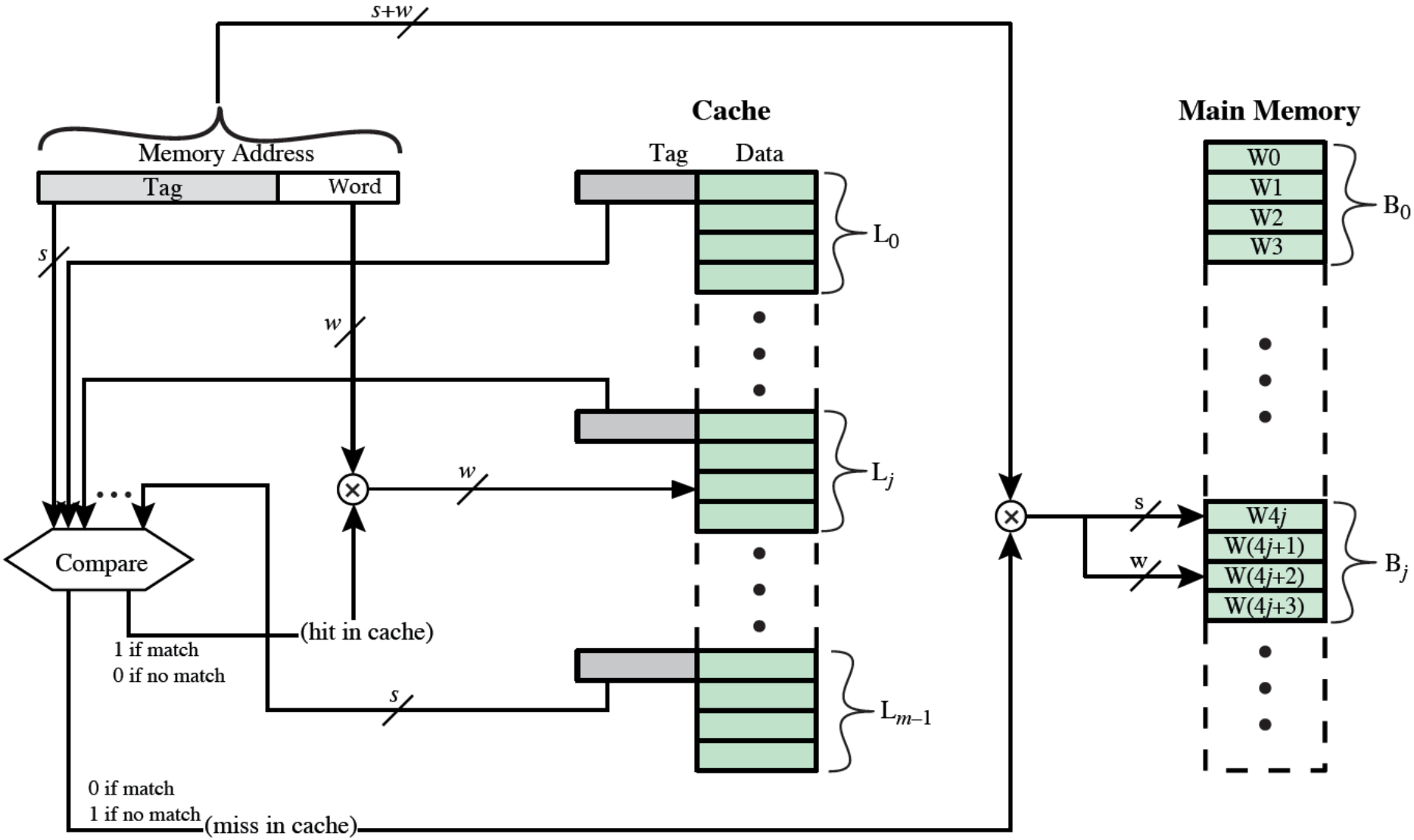
- Originally proposed as an approach to reduce the conflict misses of direct mapped caches without affecting its fast access time
- Fully associative cache
- Typical size is 4 to 16 cache lines
- Residing between direct mapped L1 cache and the next level of memory

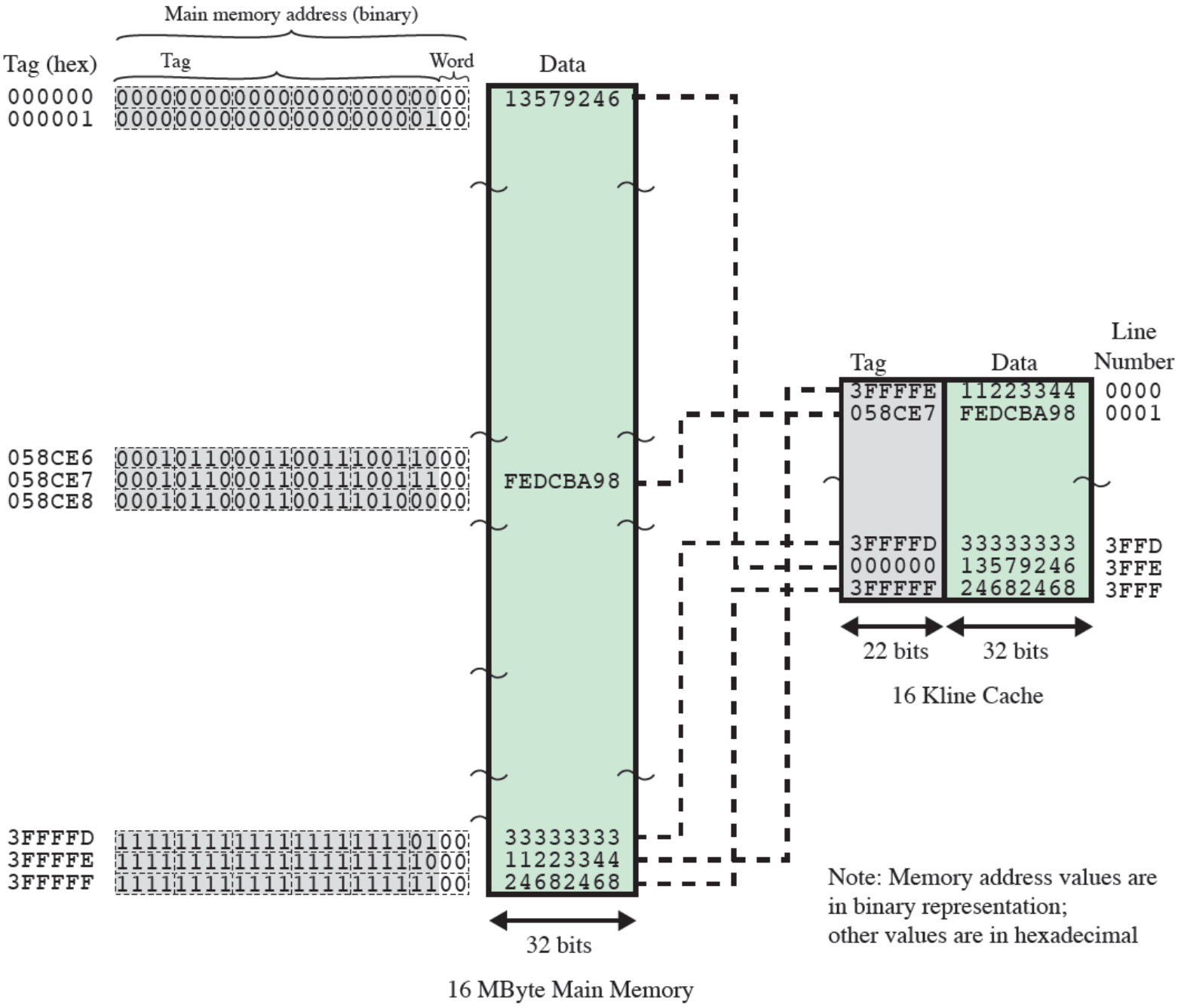
Fully Associative Cache



(b) Associative mapping

Fully Associative Cache



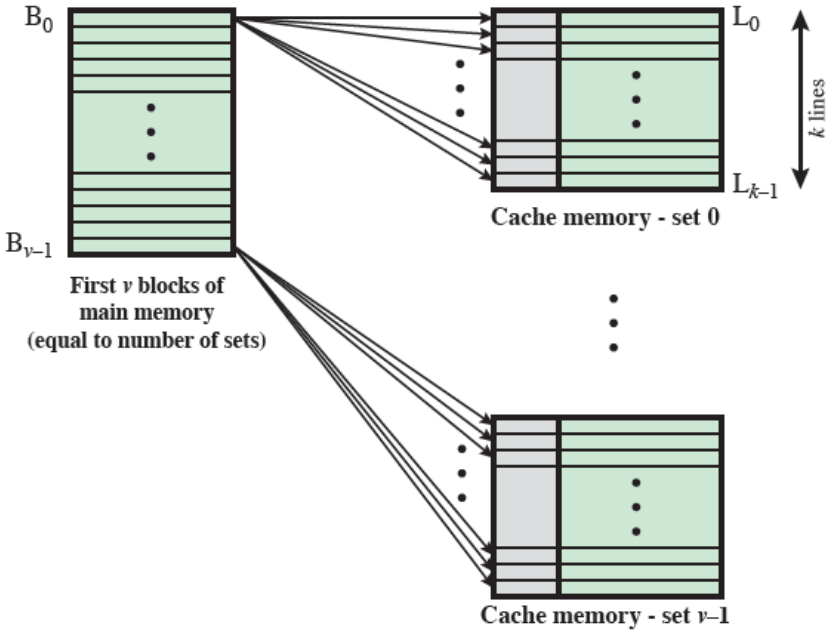

Main memory address (binary)
Tag (hex)
Tag
Word
Data
000000
000001
00000000000000000000000000
00000000000000000000000100
13579246
058CE6
058CE7
058CE8
00010110001100111001 1000
00010110001100111001 1100
00010110001100111010 0000
FEDCBA98
3FFFFD
3FFFFE
3FFFFF
11111111111111111111110100
11111111111111111111111000
11111111111111111111111100
33333333
11223344
24682468
32 bits
16 MByte Main Memory
Line Number
Tag
Data
3FFFFE
058CE7
11223344
FEDCBA98
0000
0001
3FFFFD
000000
3FFFFF
33333333
13579246
24682468
3FFD
3FFE
3FFF
22 bits
32 bits
16 Kline Cache
Note: Memory address values are in binary representation; other values are in hexadecimal

Associative Cache

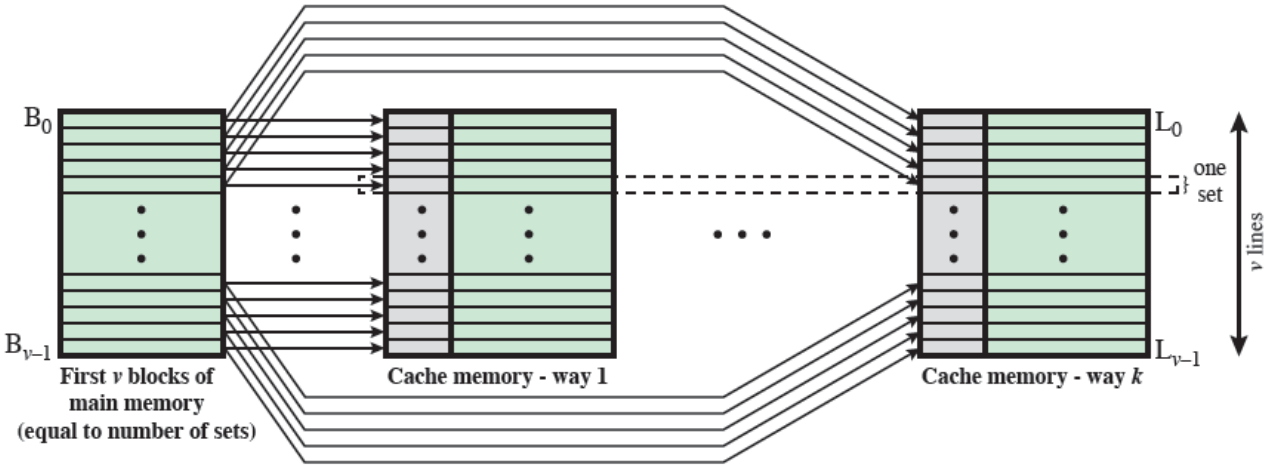
- Address length = (s + w) bits
- Number of addressable units = 2^{s+w} words or bytes
- Block size = line size = 2^{w} words or bytes
- Number of blocks in main memory = $2^{s+w}/2^{w} = 2^{s}$
- Number of lines in cache = undetermined
- Size of tag = s bits

Set Associative Mapping

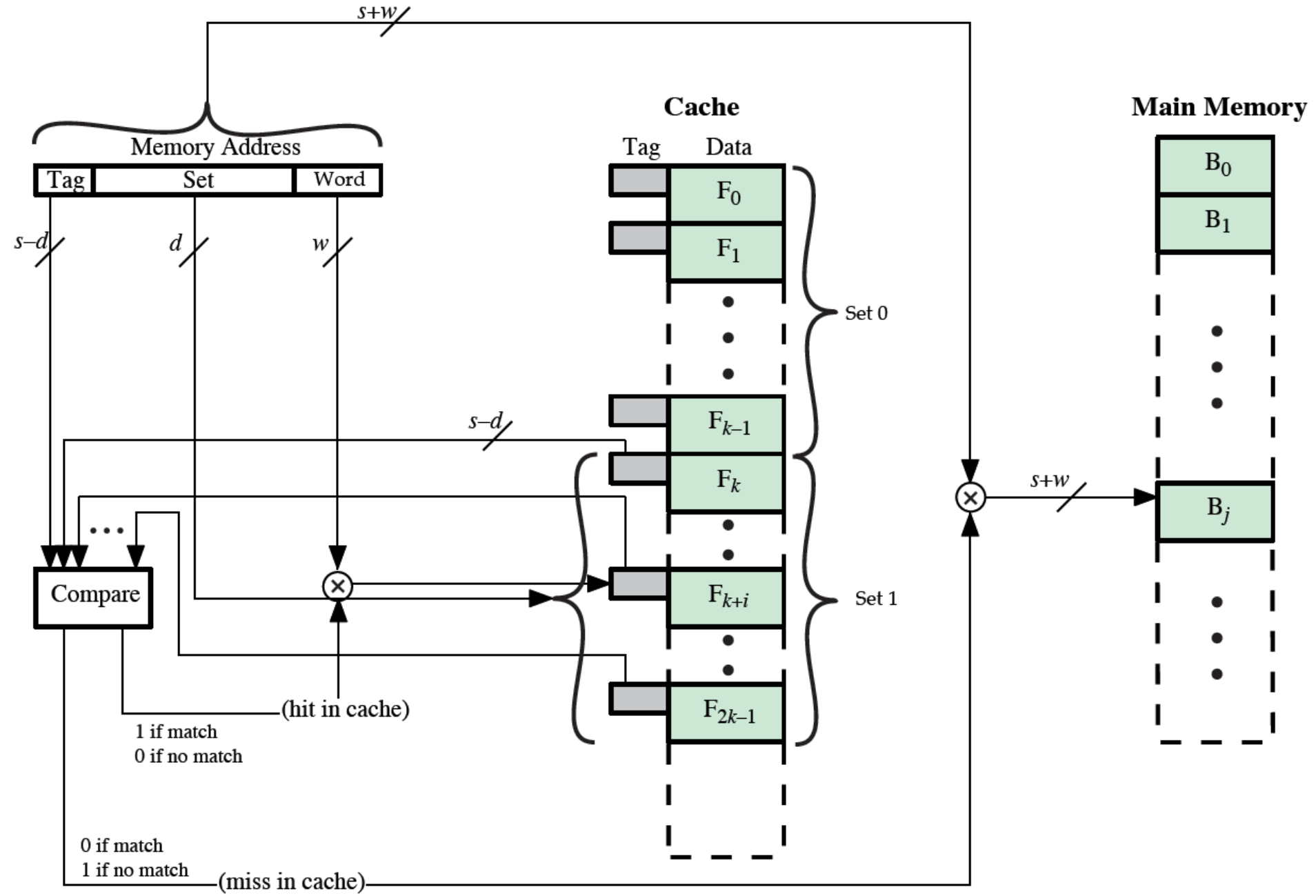
- Compromise that exhibits the strengths of both the direct and associative approaches while reducing their disadvantages
- Cache consists of a number of sets
- Each set contains a number of lines
- A given block maps to any line in a given set
- e.g. 2 lines per set
  - 2 way associative mapping
  - A given block can be in one of 2 lines in only one set



(a) *v* associative-mapped caches

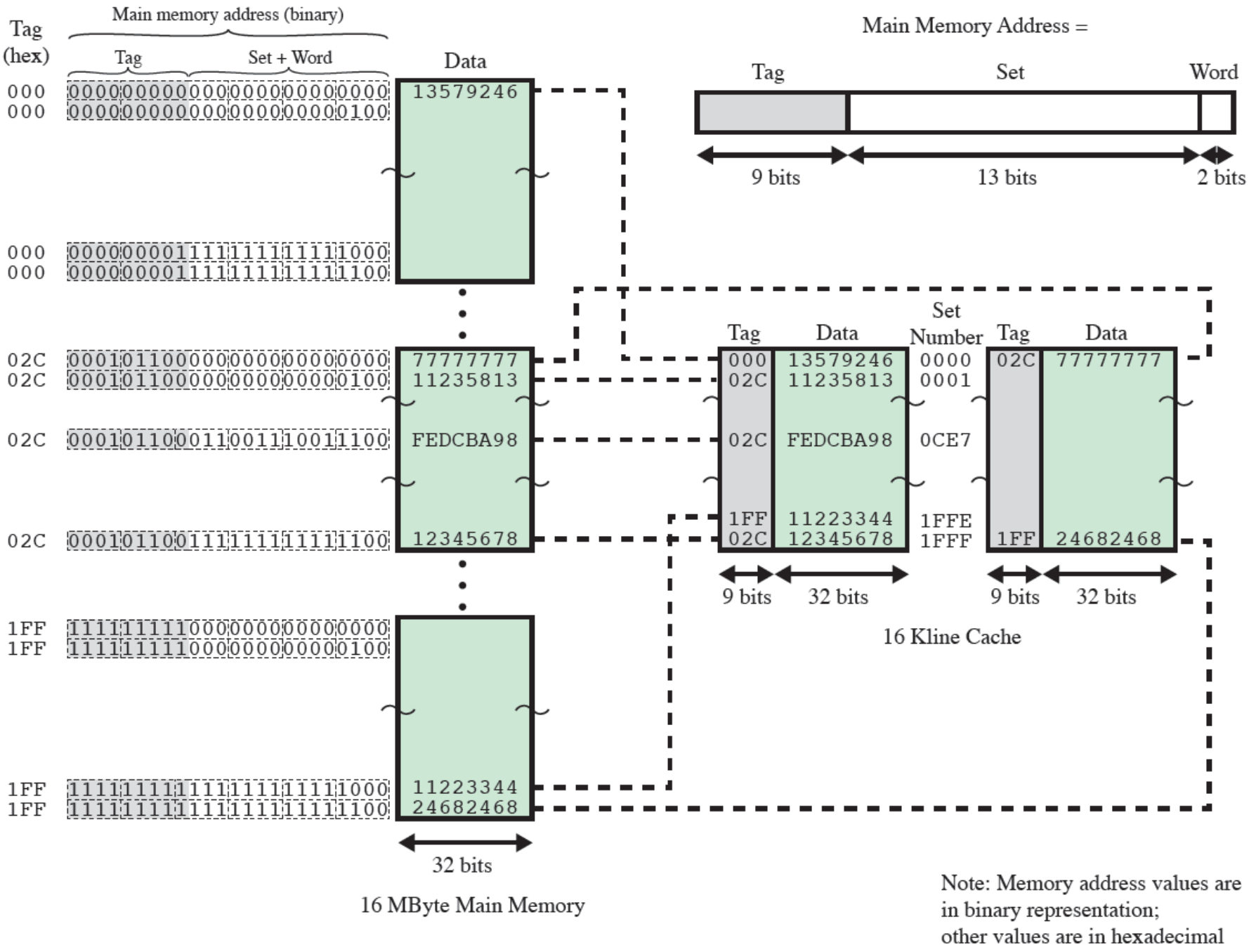


(b) *k* direct-mapped caches

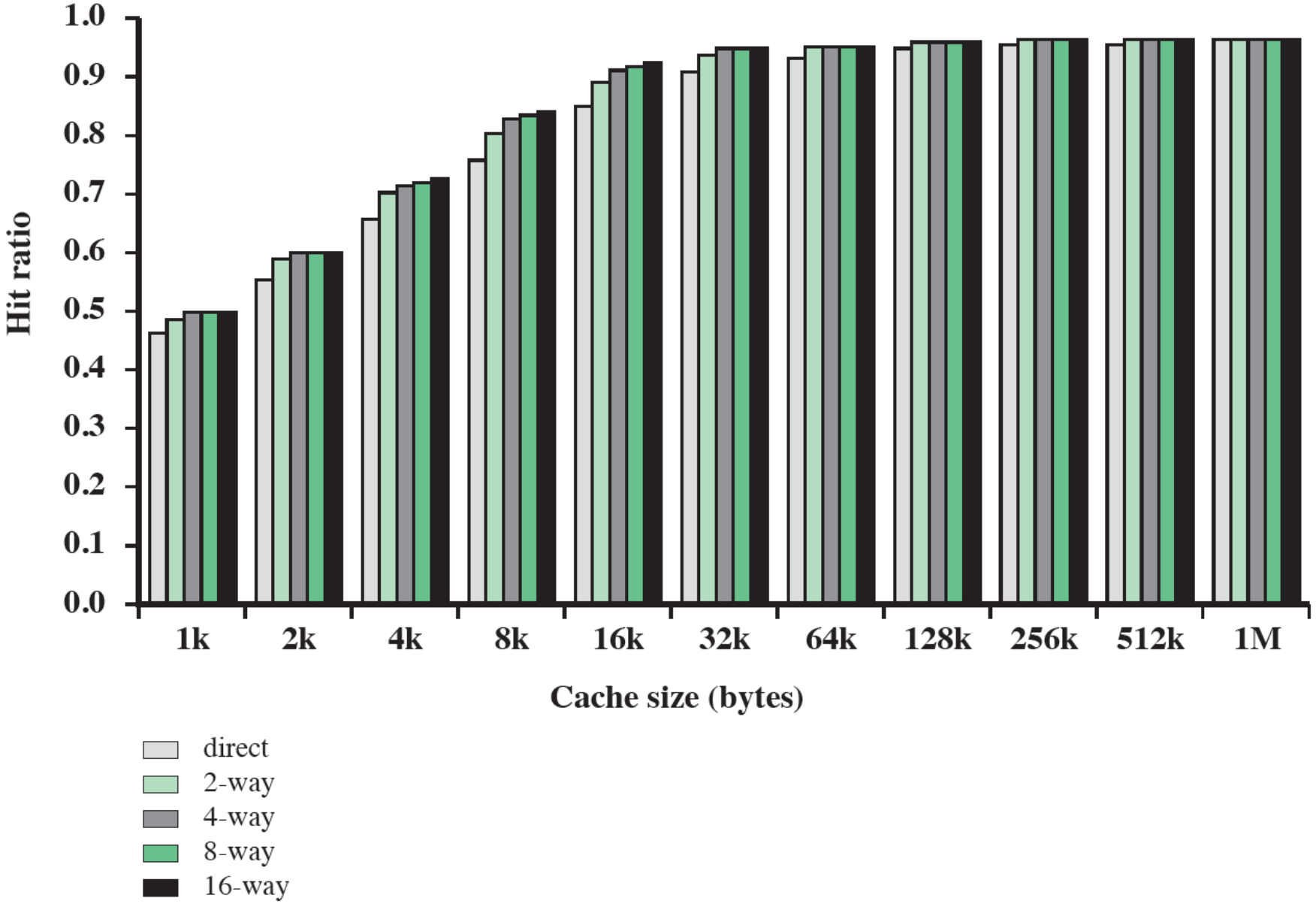

s+w
Memory Address
Tag
Set
Word
s–d
d
w
Cache
Tag
Data
F0
F1
Set 0
Fk–1
Fk
Fk+i
Set 1
F2k–1
s–d
Compare
(hit in cache)
1 if match
0 if no match
0 if match
1 if no match
(miss in cache)
s+w
Main Memory
B0
B1
Bj

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=kv = k * 2^{d}
- Size of cache = $k * 2^{d+w}$ words or bytes
- Size of tag = (s – d) bits


Tag (hex)
Main memory address (binary)
Tag
Set + Word
Data
000 0000000000000000000000000000
13579246
000 0000000000000000000000000100
000 0000000011111111111111111000
000 0000000011111111111111111100
02C 00010110000000000000000000
77777777
02C 00010110000000000000000100
11235813
02C 00010110001100111001100
FEDCBA98
02C 00010110011111111111111100
12345678
1FF 11111111100000000000000000
1FF 11111111100000000000000100
1FF 11111111111111111111111000
11223344
1FF 11111111111111111111111100
24682468
32 bits
16 MByte Main Memory
Main Memory Address =
Tag
Set
Word
9 bits
13 bits
2 bits
Tag
Data
Set Number
Tag
Data
000 13579246
0000
02C 77777777
02C 11235813
0001
02C FEDCBA98
0CE7
1FF 11223344
1FFE
02C 12345678
1FFF
1FF 24682468
9 bits
32 bits
9 bits
32 bits
16 Kline Cache
Note: Memory address values are in binary representation; other values are in hexadecimal

Performance Difference



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 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 Strategies

- **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**
  - Inferior to the others, but still astonishingly good performance

Write Policy

- When a block is to be replaced:
  - The block was not altered => We can simply replace this block
  - The block has been altered => We need to write the current status of this cache line into Memory before overwriting is

- There are two problems to contend with:
  - More than one device may have access to main memory
  - Even more complex: Multi-core processors. A change in one cache could invalidate cache lines in another core's local cache

Write Through vs. Write Back

- **Write through**
  - Simplest technique
  - All write operations are made to main memory as well as to the cache
  - The main disadvantage of this technique is that it generates substantial memory traffic and may create a bottleneck

- **Write back**
  - Minimizes memory writes
  - Updates are made only in the cache
  - Portions of main memory are invalid and hence accesses by I/O modules can be allowed only through the cache
  - This makes for complex circuitry and a potential bottleneck

Line Size

- When a block is placed in the cache some number of adjacent words are retrieved

Line Size

- When a block is placed in the cache 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; more useful data are brought into the cache

Line Size

- When a block is placed in the cache 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; 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

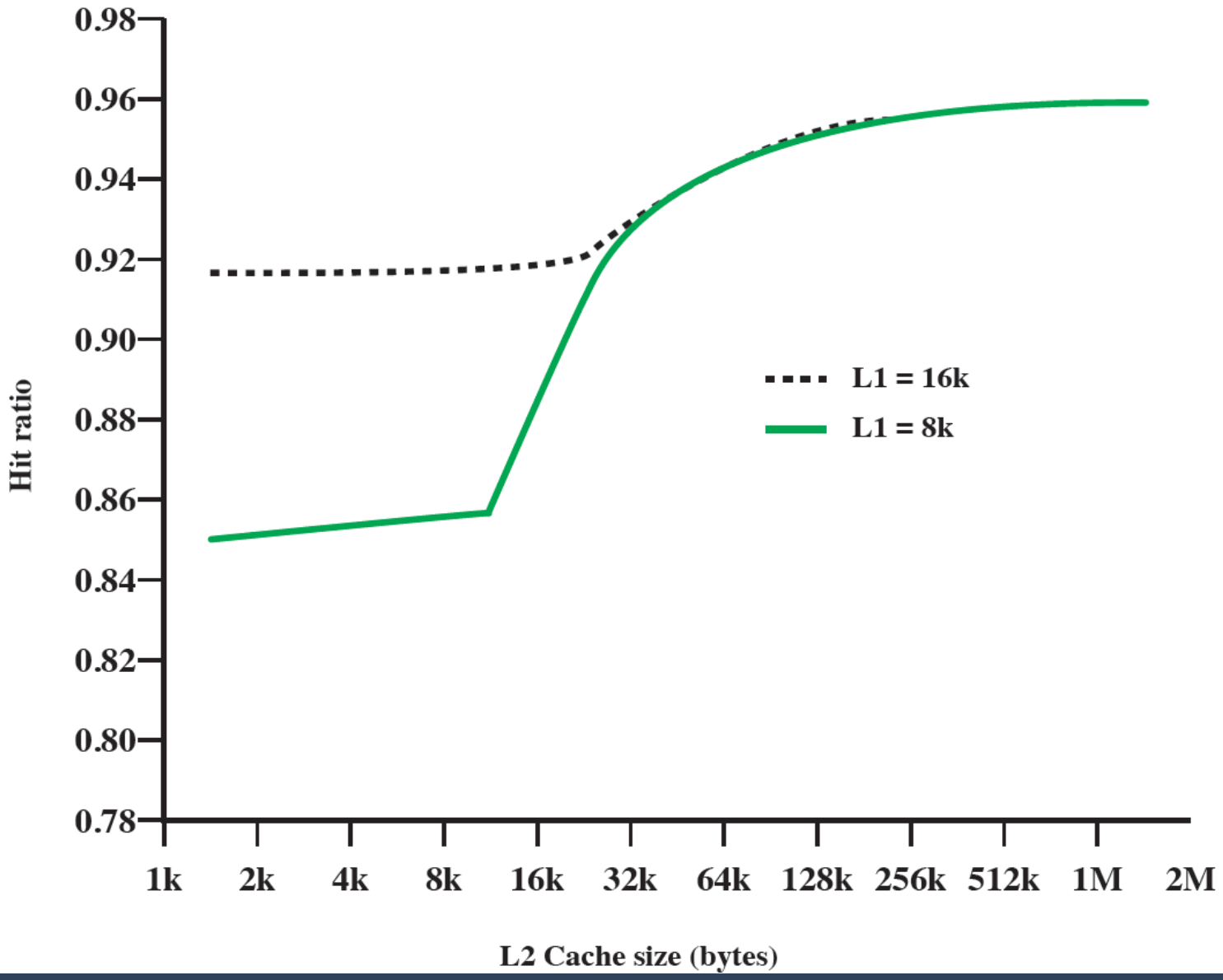
Line Size

- When a block is placed in the cache 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; 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

Multilevel Caches

- Nowadays, caches reside on the same chip as the processor
- This reduces the external bus activity
  - If data is in the on-chip cache, the bus access is eliminated entirely
  - On-chip cache accesses are faster than 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 depend on the hit rates in both the L1 and L2
- The use of multilevel caches complicates all of the design issues
  - sizes
  - replacement algorithm
  - write policy

Hit Ratio L1 & L2 Cache



Split vs. Unified Cache

- **Split Cache**
  - One dedicated to instructions
  - One dedicated to data
  - Both exist at the same level, typically as two L1 caches
  - Eliminates cache contention between instruction fetch/decode unit and execution unit
  - Important for Pipelining

Split vs. Unified Cache

- **Split Cache**
  - One dedicated to instructions
  - One dedicated to data
  - Both exist at the same level, typically as two L1 caches
  - Eliminates cache contention between instruction fetch/decode unit and execution unit
  - Important for Pipelining
- **Unified Cache**
  - Higher hit rate
  - Balances load of instruction and data fetches automatically
  - Only one cache needs to be designed and implemented

Split vs. Unified Cache

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

Problem	Solution	Processor
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 Cache Block Diagram