Memoria Cache

Corso di Architettura degli Elaboratori (teoria)

Dott. Francesco De Angelis
francesco.deangelis@unicam.it

Scuola di Scienze e Tecnologie - Sezione di Informatica

Architettura degli Elaboratori e Laboratorio

William Stallings
Computer Organization and Architecture
8th Edition

Chapter 4

Cache Memory

Characteristics of memory

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

Location

- CPU
  - registers
- Internal
  - For control purpose in CPU, cache
- External
  - Disk, and other I/O

Capacity

- Word size
  - The natural unit of organisation
- Number of words
  - or Bytes

Unit of Transfer

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

Access Methods (1)

- Sequential
  - Start at the beginning and read through in order
  - Access time depends on location of data and previous location
  - e.g. tape
- Direct
  - Individual blocks have unique address
  - Access is by jumping to vicinity plus sequential search
  - Access time depends on location and previous location
  - e.g. disk

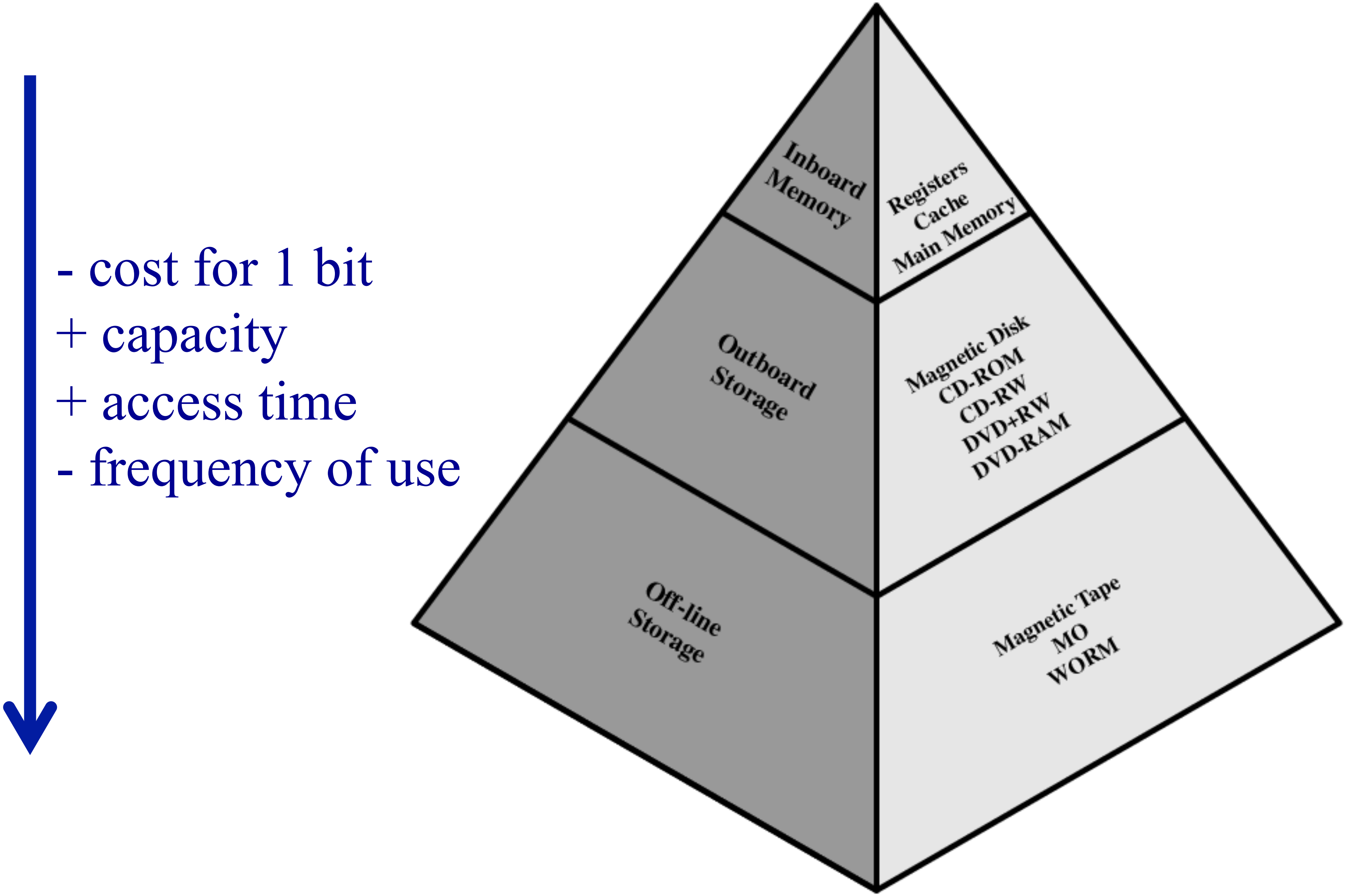
Access Methods (2)

- Random
  - Individual addresses identify locations exactly
  - Access time is independent of location or previous access
  - e.g. RAM
- Associative
  - Data is located by a comparison with contents of a portion of the store
  - Access time is independent of location or previous access
  - e.g. cache

Memory Hierarchy

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

Memory Hierarchy - Diagram



- cost for 1 bit

+ capacity

+ access time

- frequency of use

Performance

- Access time (latency)
  - Time between presenting the address to the memory 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
  - Recovery is due to bus, not memory!
- Transfer Rate
  - Rate at which data can be moved

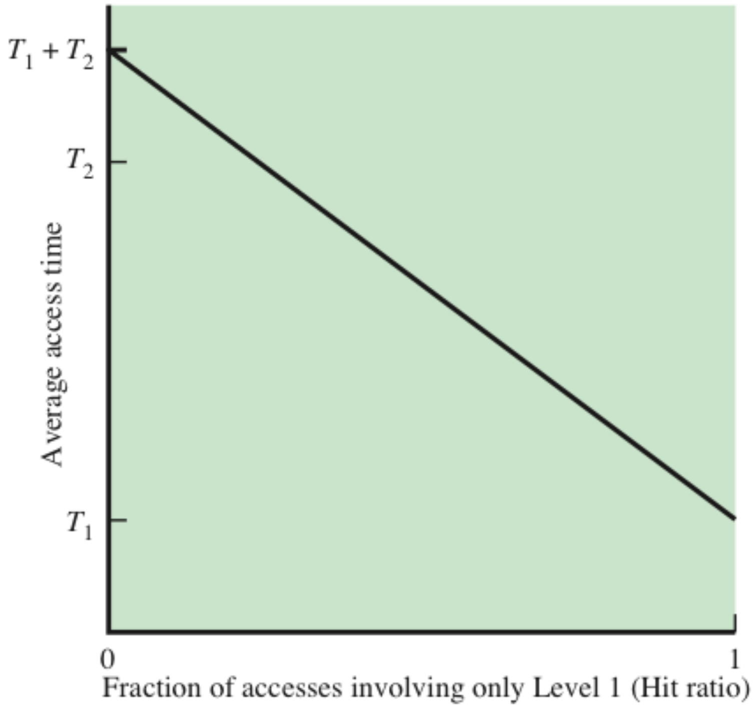
Example with 2 levels of memory

- Access time: 0,01 μs for level 1; 0,1 μs for level 2
- Use: 95% level 1
- Word form level 2 “travel” through level 1 to reach the CPU

Average access time:

$(0,95)(0,01\ \mu s) +$
$(0,05)(0,01\ \mu s + 0,1\ \mu s)$

$= 0,015\ \mu s$



Physical Types

- Semiconductor
  - RAM (random access, volatile)
  - ROM (non-volatile)
- Magnetic
  - Disk & Tape
- Optical
  - CD & DVD
- Others
  - Bubble
  - Hologram

Physical Characteristics

- Decay
- Volatility
- Erasable
- Power consumption

Organisation

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

The Bottom Line

- We can state the constraint of memory design into:
- How much?
  - Capacity
- How fast?
  - Time to access
- How expensive?
  - Money

Hierarchy List

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

So you want fast?

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

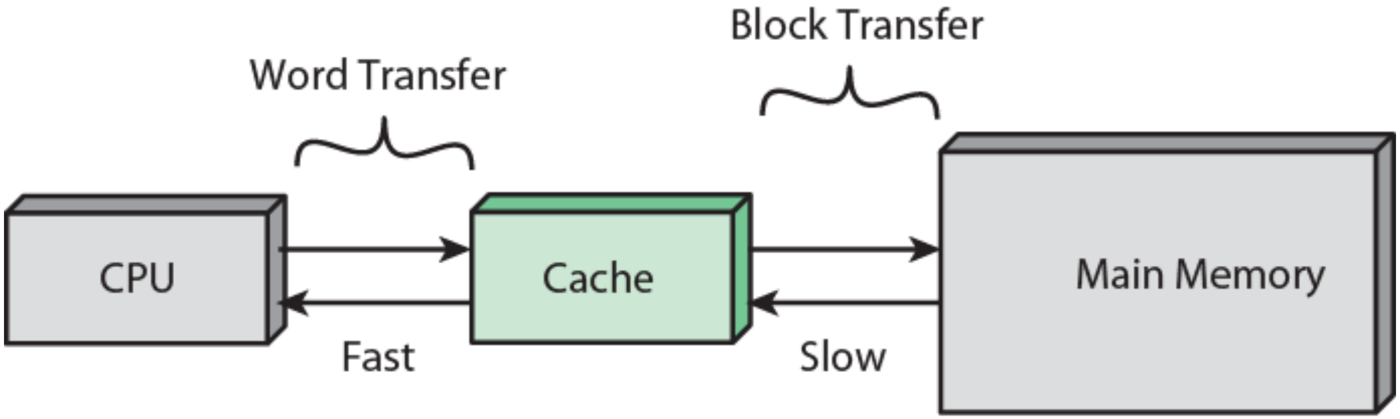
Locality of Reference

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

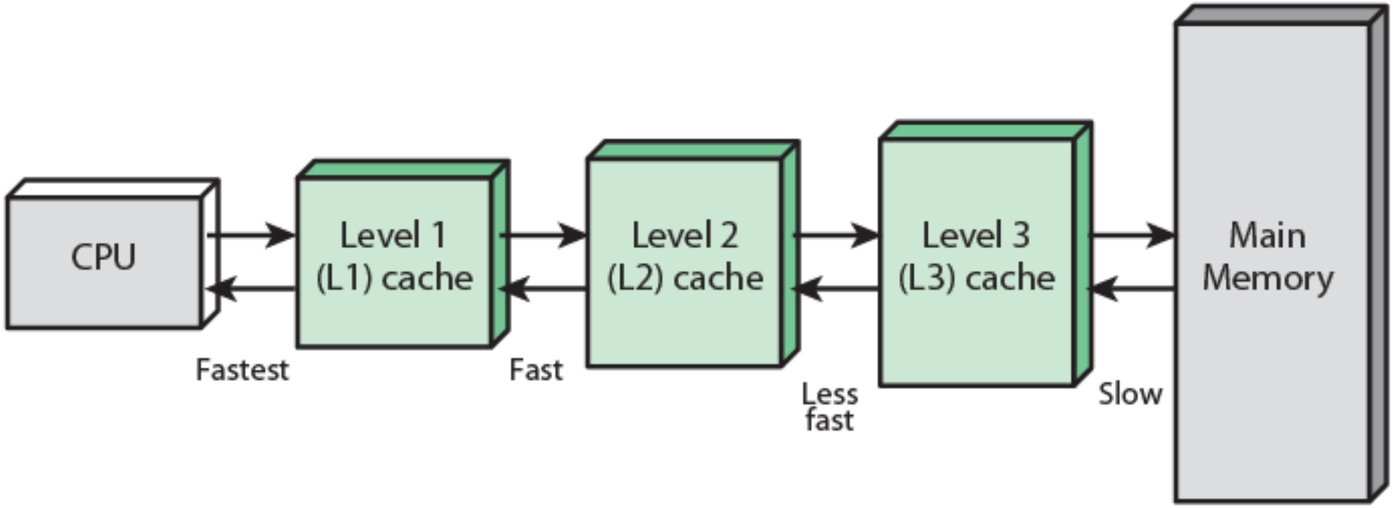
Cache

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

Cache and Main Memory

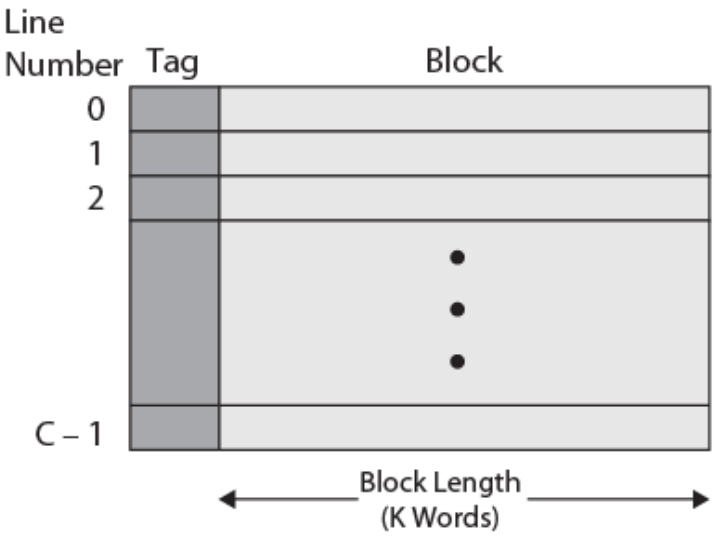


(a) Single cache

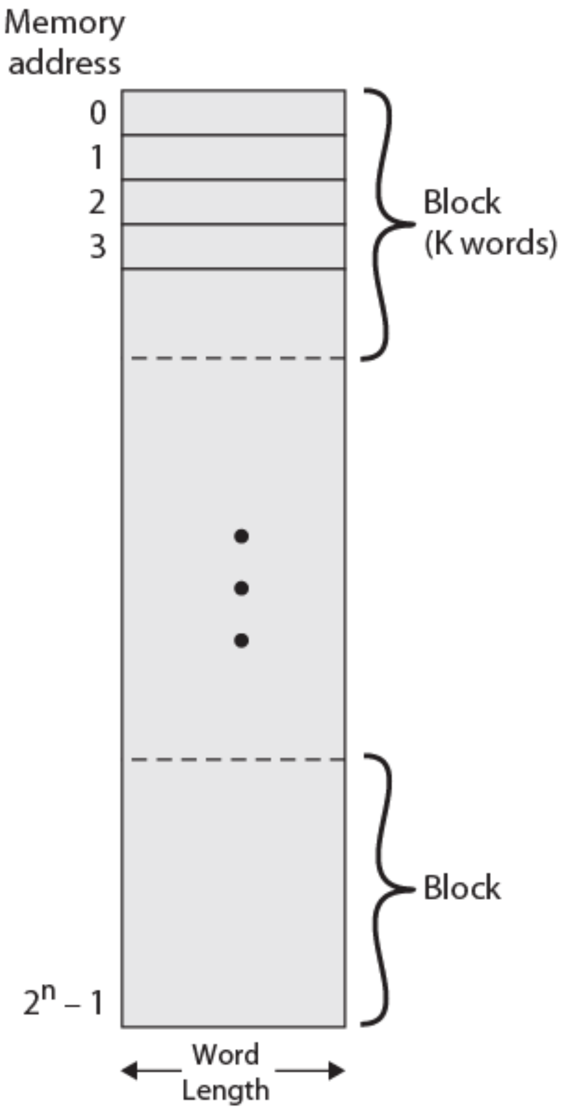


(b) Three-level cache organization

Cache/Main Memory Structure



(a) Cache



(b) Main memory

Addresses of n bits

Blocks of K words

M = $2^n/K$ blocks in main memory

C lines of cache

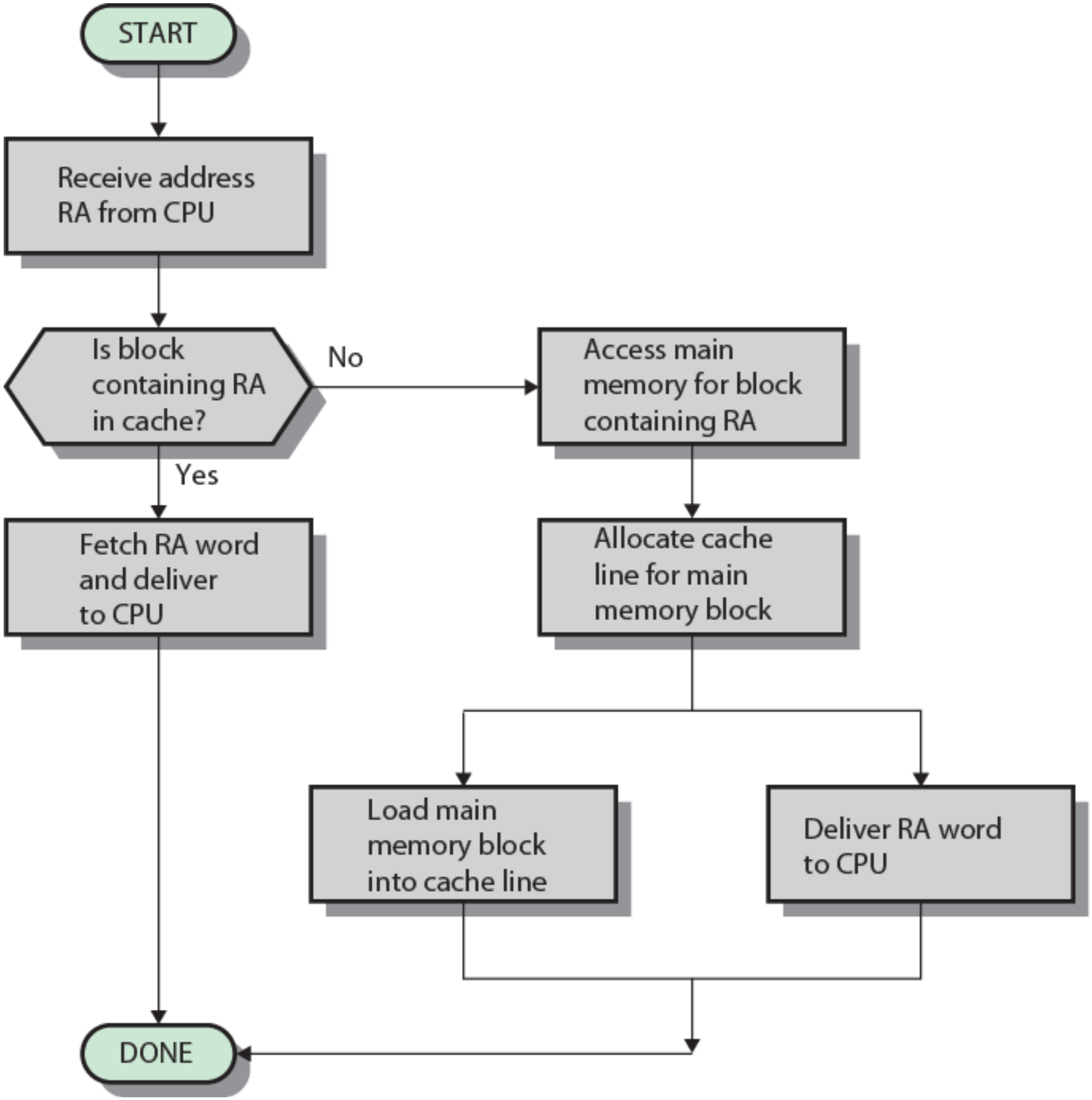
K words for each line

C<<M

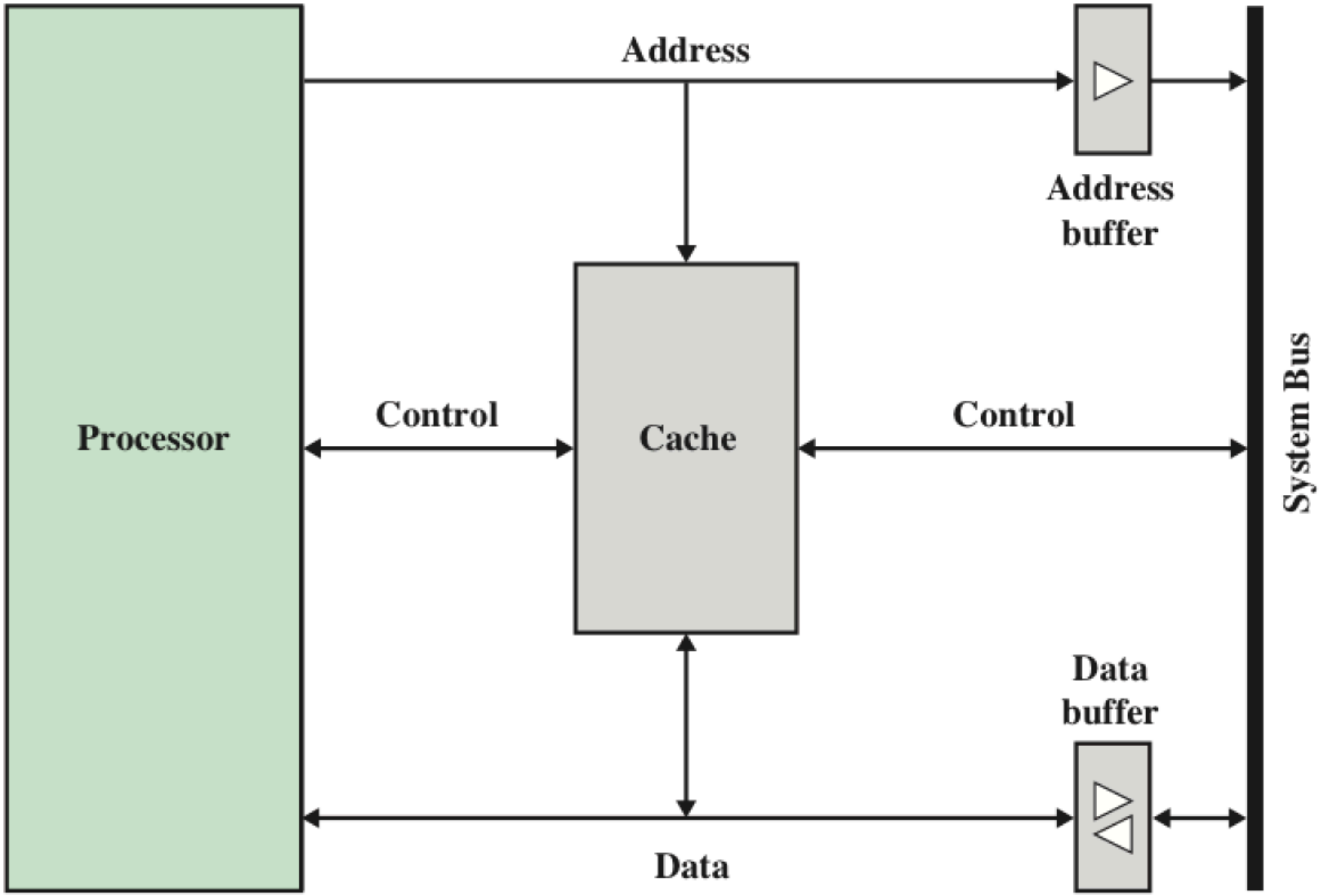
Cache operation – overview

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

Cache Read Operation - Flowchart


START
Receive address RA from CPU
Is block containing RA in cache?
No
Yes
Access main memory for block containing RA
Fetch RA word and deliver to CPU
Allocate cache line for main memory block
Load main memory block into cache line
Deliver RA word to CPU
DONE

Typical Cache organization



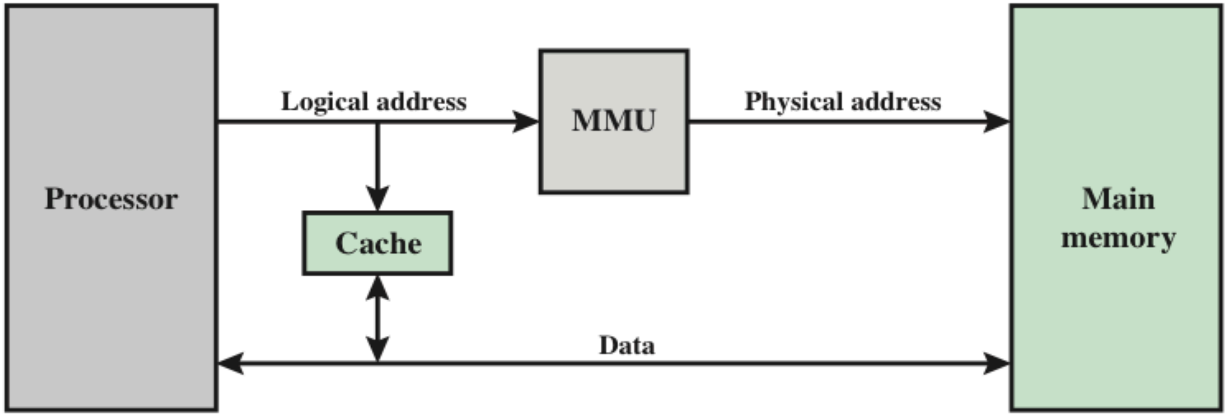
Cache Design

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

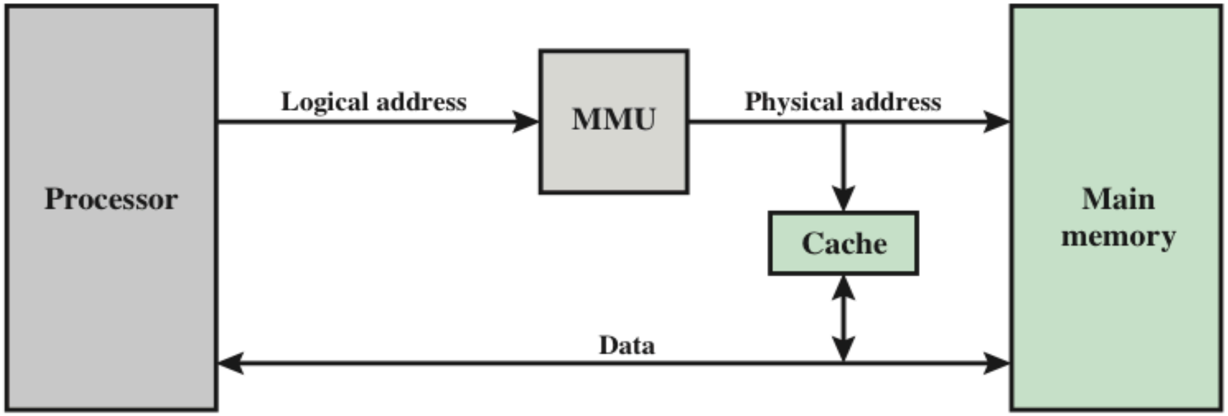
Cache Addressing

- Where does cache sit?
  - Between processor and virtual memory management unit
  - Between MMU and main memory
- Logical cache (virtual cache) stores data using virtual addresses
  - Processor accesses cache directly
  - Cache access faster, before MMU address translation
  - Virtual addresses use same address space for different applications
    - Must flush cache on each context switch
- Physical cache stores data using main memory physical addresses

Physical vs Logical Cache



(a) Logical Cache



(b) Physical Cache

Size does matter

- Cost
  - More cache is expensive
- Speed
  - More cache is faster (up to a point)
  - Checking cache for data takes time

Comparison of Cache Sizes

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

Mapping Function – Example data

- Cache of 64kByte (2^{16})
- Cache block of 4 bytes (2^2)
  - i.e. cache is 16k (2^{14}) lines of 4 bytes

- 16MBytes main memory (2^{24})
- 24 bit address
  - (2^{24}=16M)
- We address a single byte
- We have 4M (2^{22}) blocks of 4 byte

Direct Mapping

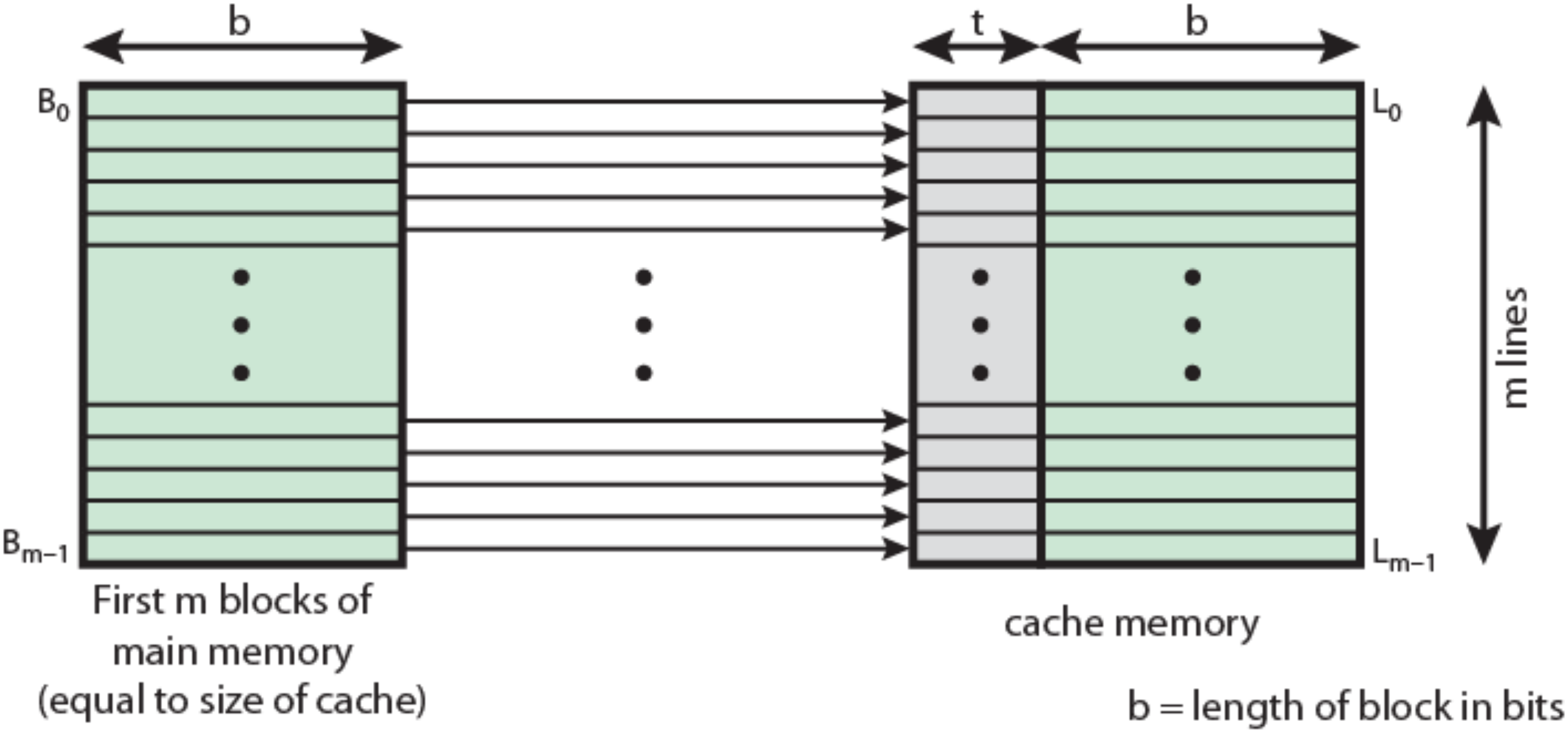
- Each block of main memory maps to only one cache line
  - i.e. if a block is in cache, it must be in one specific place
- Address is in two parts
- Least Significant w bits identify unique word
- Most Significant s bits specify one memory block
- The MSBs are split into a cache line field r and a tag of s-r (most significant)

Direct Mapping
Address Structure

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

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

Direct Mapping from Cache to Main Memory



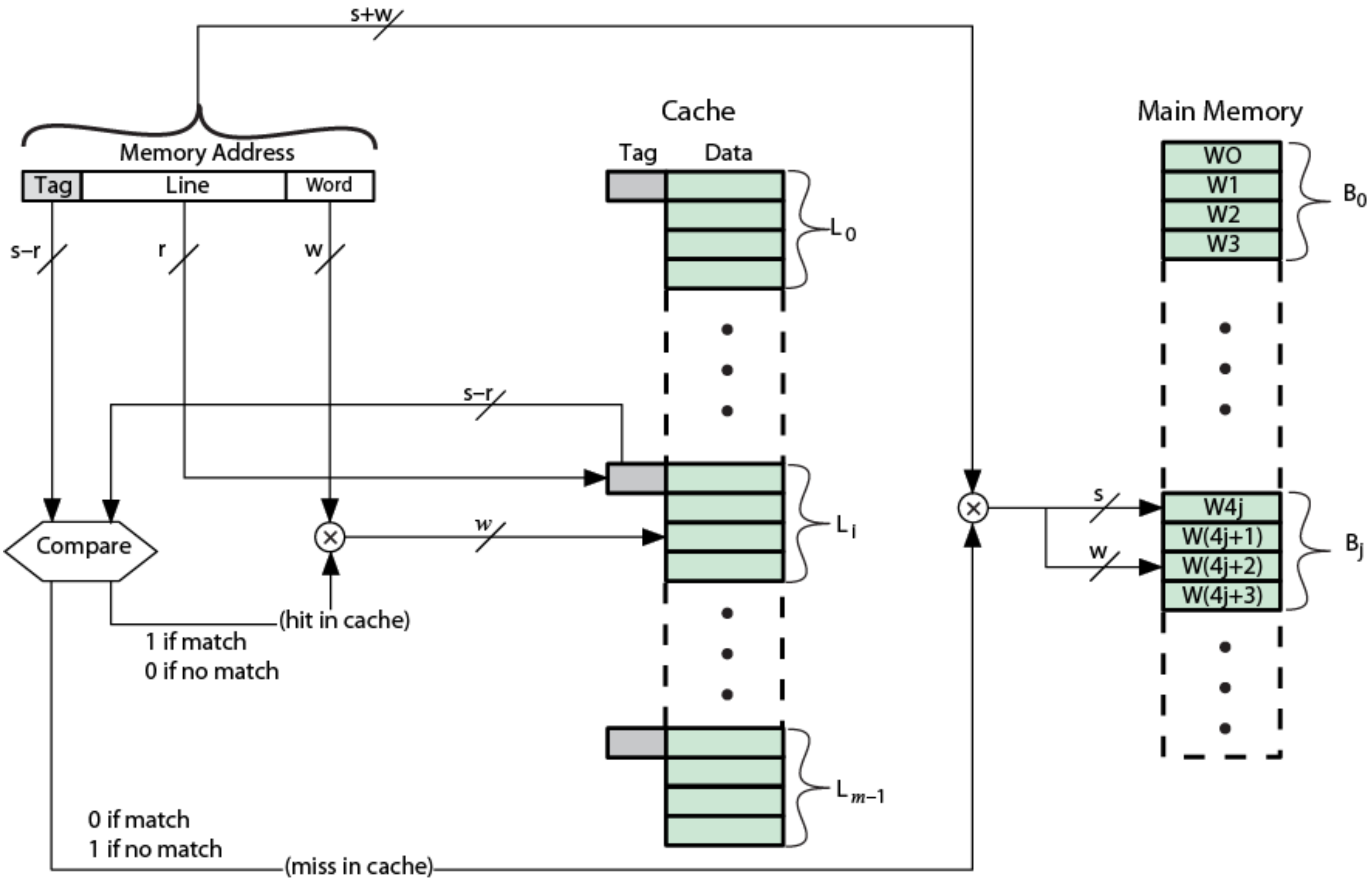
b = length of block in bits
t = length of tag in bits

(a) Direct mapping

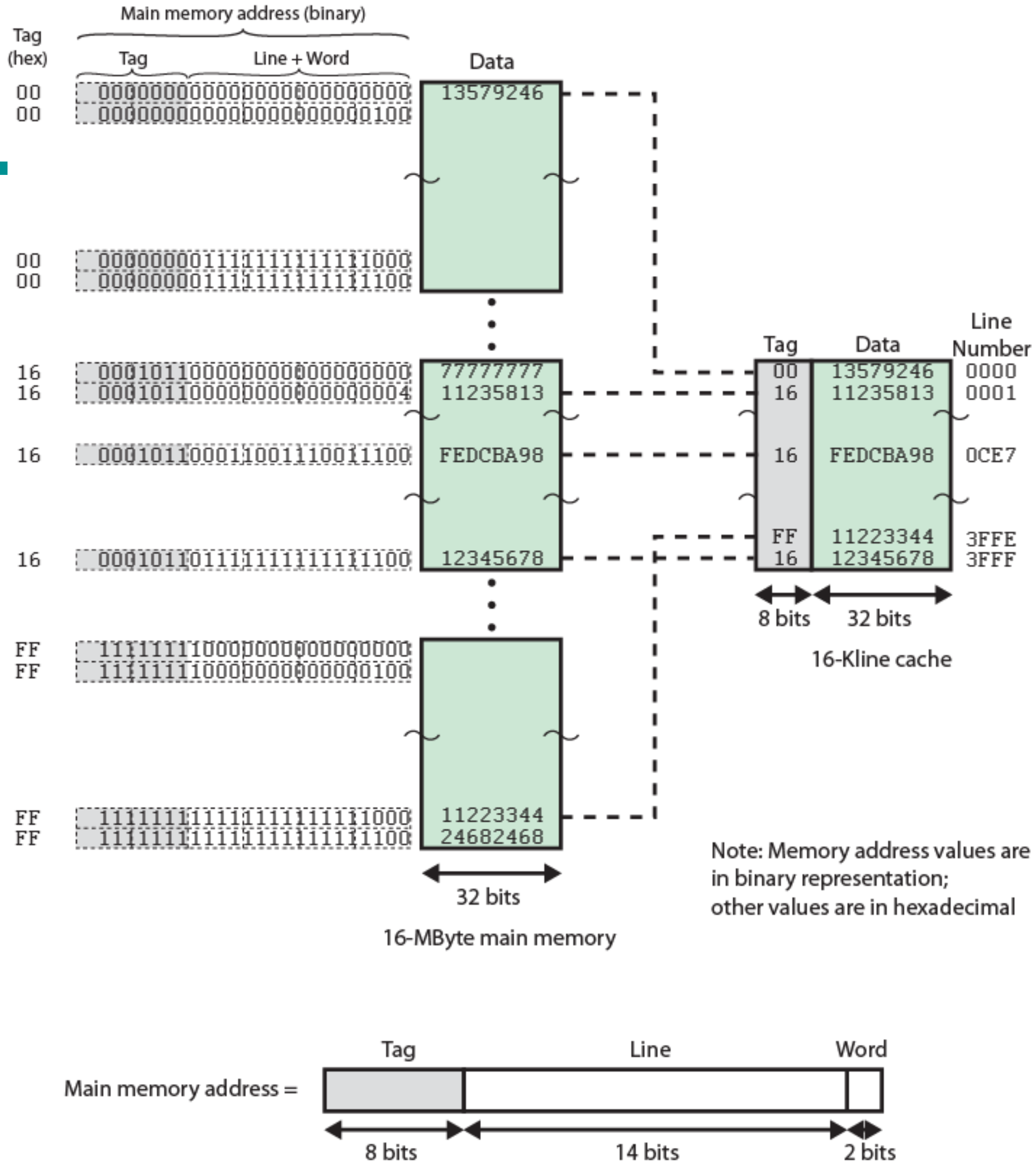
Direct Mapping
Cache Line Table

Cache line	Main Memory blocks held
0	0, m, 2m, 3m…2s-m
1	1,m+1, 2m+1…2s-m+1
…	
m-1	m-1, 2m-1,3m-1…2s-1

Direct Mapping Cache Organization



Direct Mapping Example



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
- 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

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

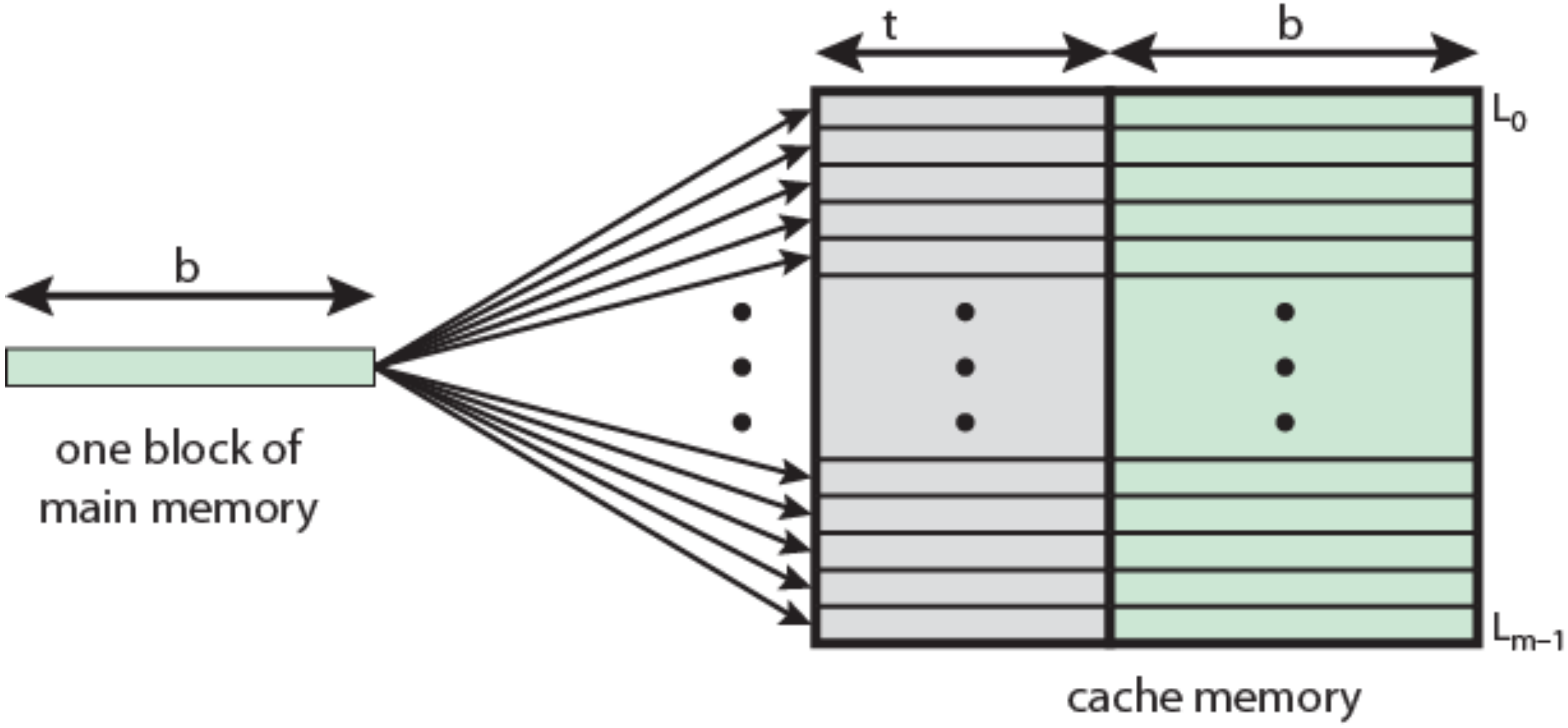
Victim Cache

- Lower miss penalty
- Remember what was discarded
  - Already fetched
  - Use again with little penalty
- Fully associative
- 4 to 16 cache lines
- Between direct mapped L1 cache and next memory level

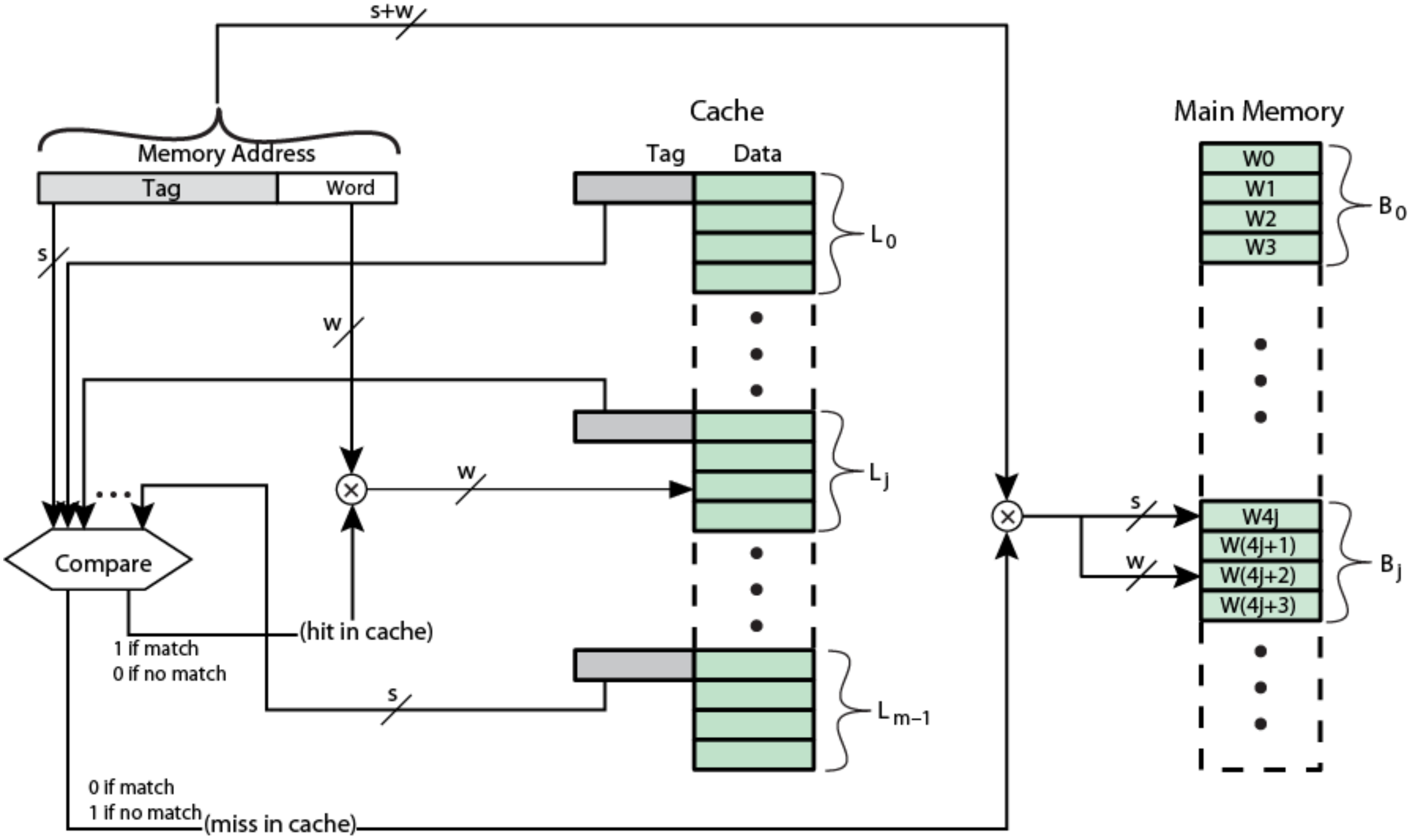
Associative Mapping

- 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 gets expensive

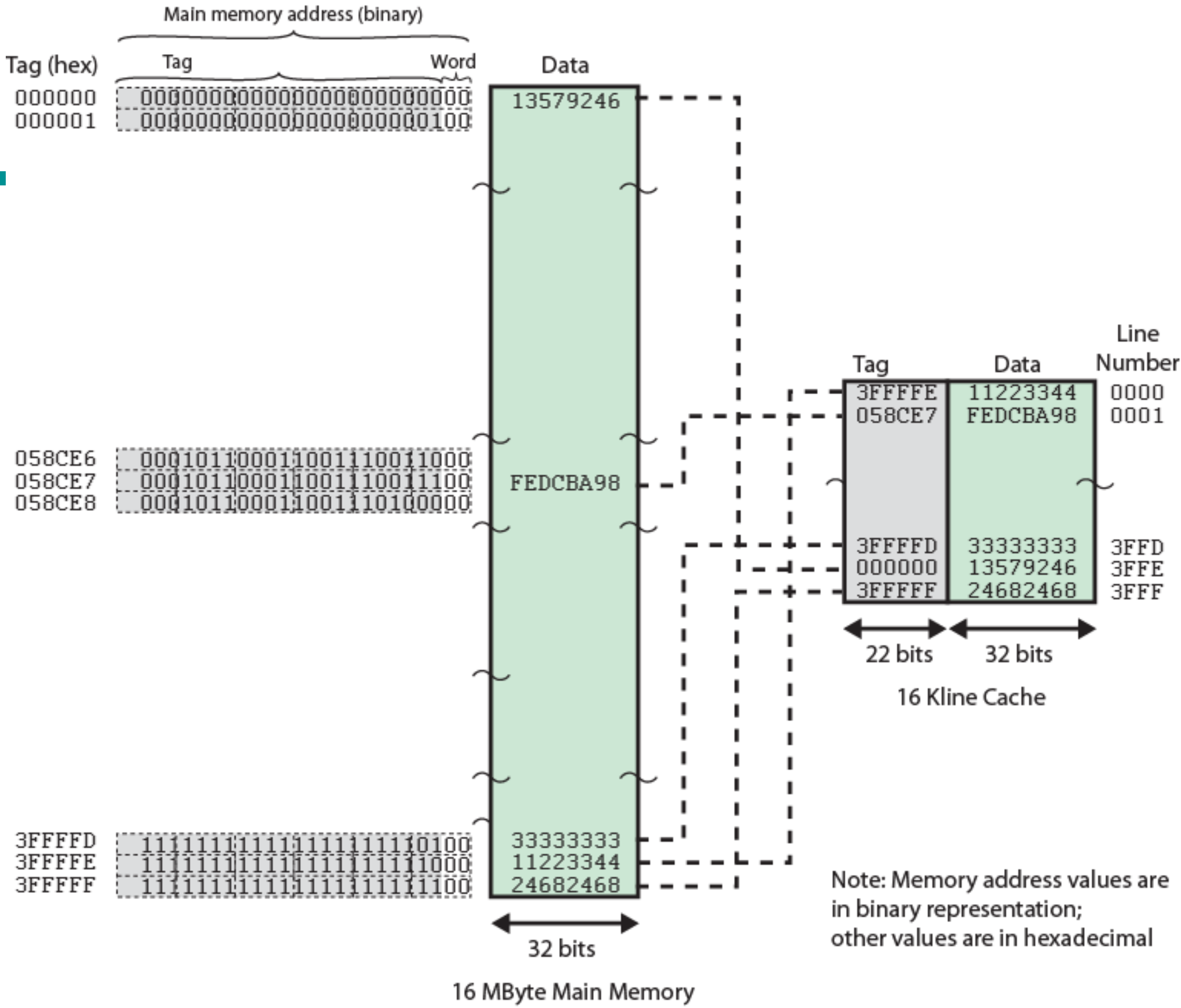
Associative Mapping from Cache to Main Memory



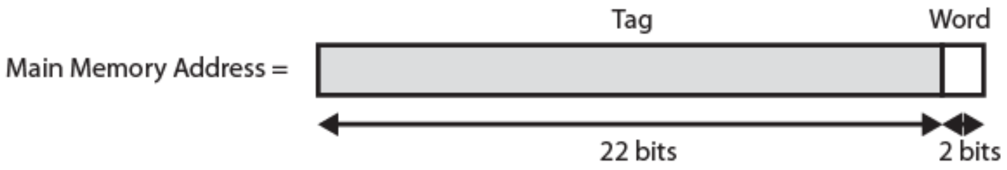
Fully Associative Cache Organization



Associative Mapping Example



Note: Memory address values are in binary representation; other values are in hexadecimal



Associative Mapping Address Structure

Tag 22 bit	Word 2 bit

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

Address	Tag	Data	Cache line
FFFFFC	FFFFFC24682468	3FFF	

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

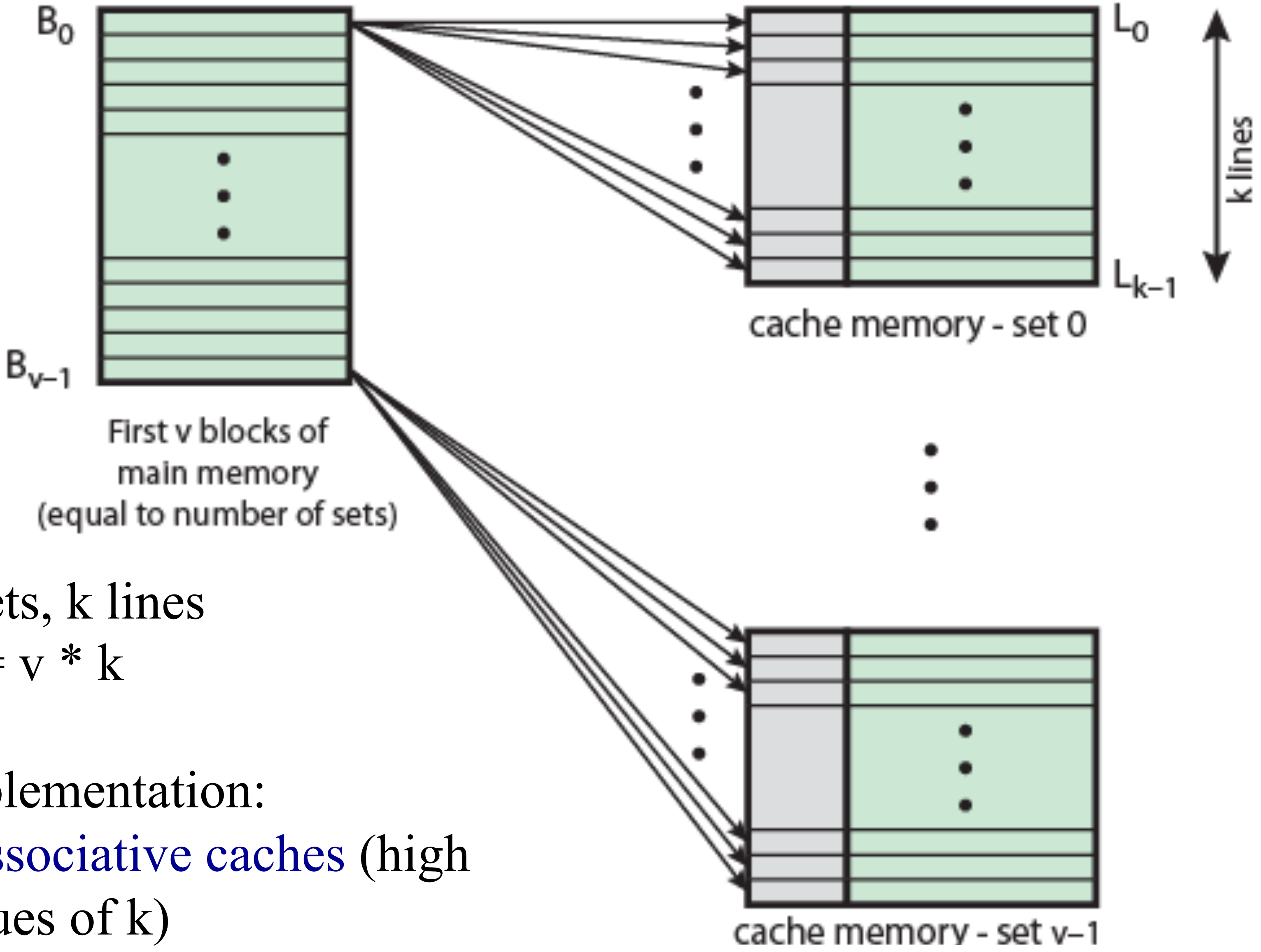
Set Associative Mapping

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

Set Associative Mapping Example

- 13 bit set number
- Block number in main memory is modulo 2^{13}
- 000000, 00A000, 00B000, 00C000 … map to same set

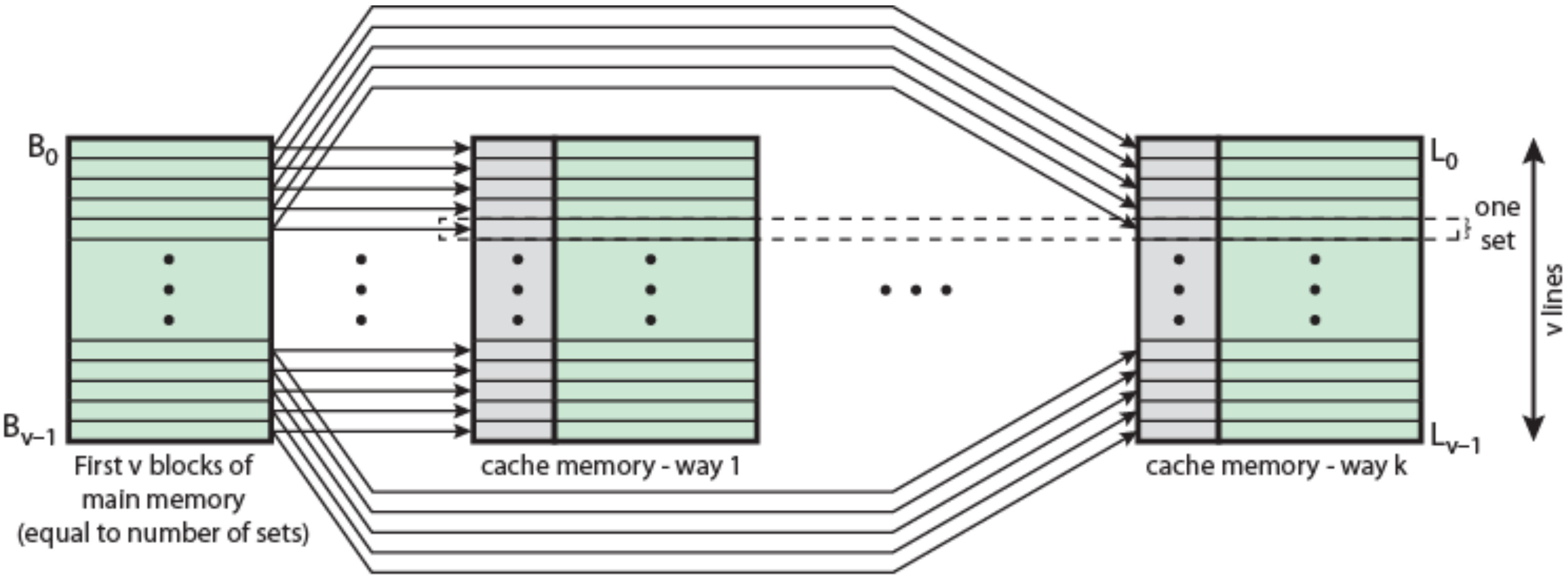
Mapping From Main Memory to Cache: v Associative



v sets, k lines
m = v * k

implementation:
v associative caches (high values of k)

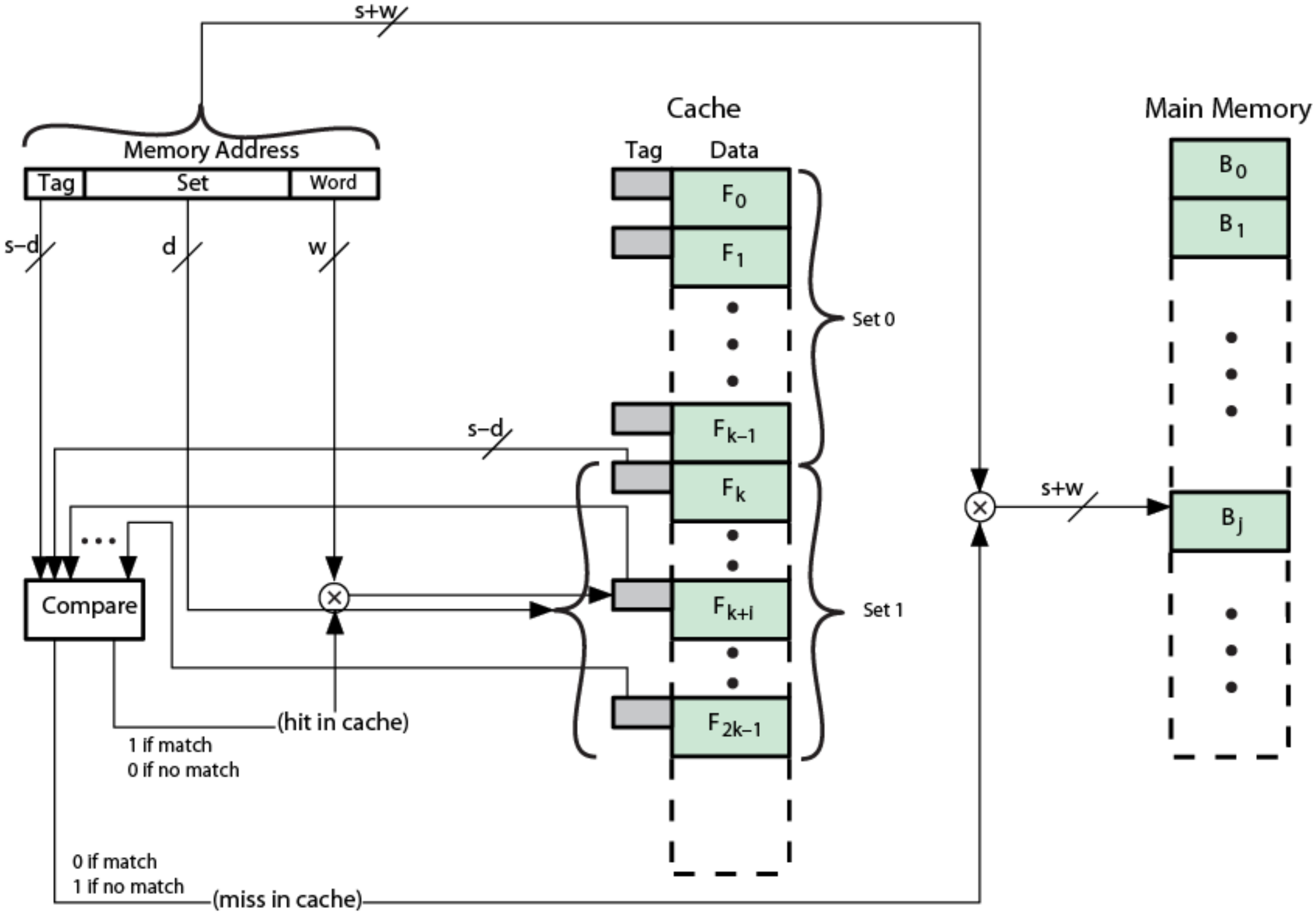
Mapping From Main Memory to Cache: k-way Associative



implementation:

k direct caches (low values of k)

K-Way Set Associative Cache Organization



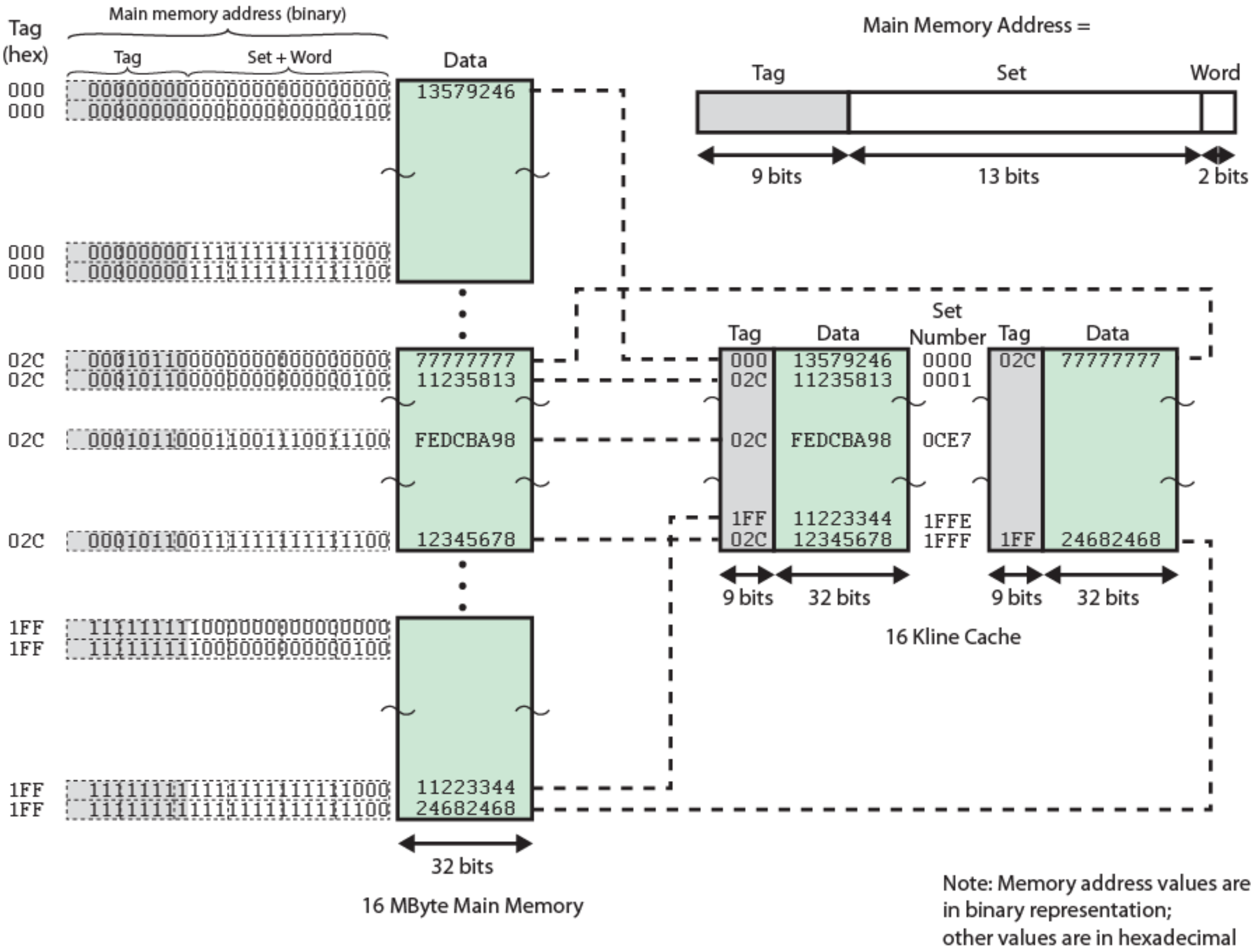
Set Associative Mapping Address Structure

Tag 9 bit	Set 13 bit	Word 2 bit

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

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

Two Way Set Associative Mapping Example


Tag (hex)
Main memory address (binary)
Tag
Set + Word
Data
000 000000000000000000000000 13579246
000 000000000000000000000100
000 000000001111111111111000
000 000000001111111111111100
02C 000101100000000000000000 77777777
02C 000101100000000000000100 11235813
02C 000101100011001110011100 FEDCBA98
02C 000101100111111111111100 12345678
1FF 111111111000000000000000
1FF 111111111000000000000100
1FF 111111111111111111111000 11223344
1FF 111111111111111111111100 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

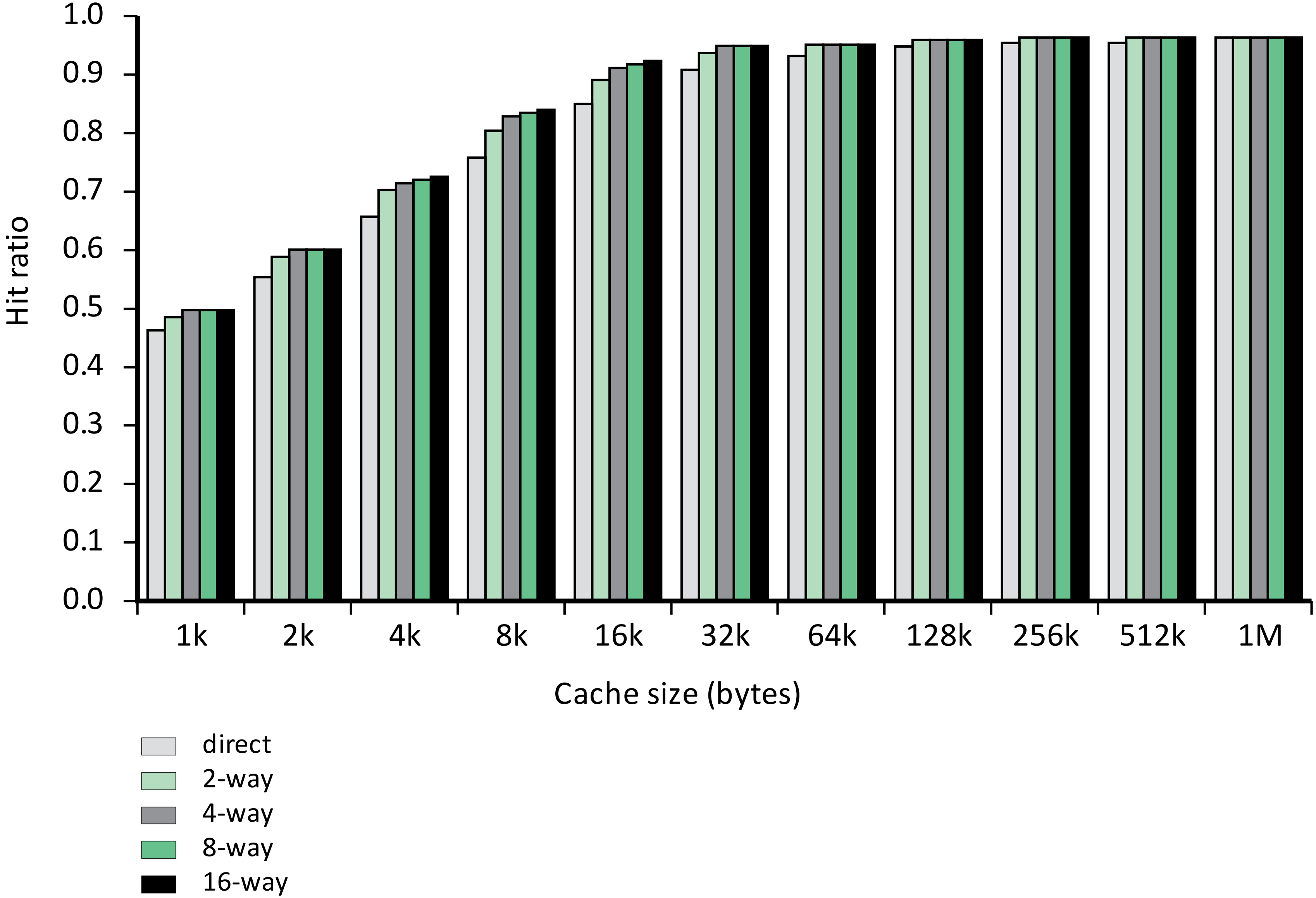
Set Associative Mapping Summary

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

Direct and Set Associative Cache Performance Differences

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

Figure 4.16
Varying Associativity over Cache Size



Replacement Algorithms (1)
Direct mapping

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

Replacement Algorithms (2)
Associative & Set Associative

- Hardware implemented algorithm (speed)
- Least Recently used (LRU)
  - Replace block not used recently
- e.g. in 2 way set associative
  - Which of the 2 block is LRU? "USE" bit!
- First in first out (FIFO)
  - replace block that has been in cache longest
- Least frequently used
  - replace block which has had fewest hits
- Random

Write Policy

- Must not overwrite a cache block unless main memory is up to date
- 2 problems:
  - Multiple CPUs may have individual caches
  - I/O modules may address main memory directly

Write through

- All writes go to main memory as well as cache
- Multiple CPUs can monitor main memory traffic to keep local (to CPU) cache up to date (bus monitoring)
- Lots of traffic
- Slows down writes

- Remember bogus write through caches!

Write through with multiple CPU

- Bus monitoring
  - —Lines are invalidates in all caches if altered in one
- Hardware transparency
  - —Lines are updated in all caches
- *Noncacheable* memory
  - —Memory that use cache is not shared
  - —Memory that is shared don’t use cache

Write back

- Updates initially made in cache only
- Update bit for cache slot is set when update occurs
- If block is to be replaced, write to main memory only if update bit is set
- Other caches get out of sync
- I/O must access main memory through cache
- N.B. 15% of memory references are writes

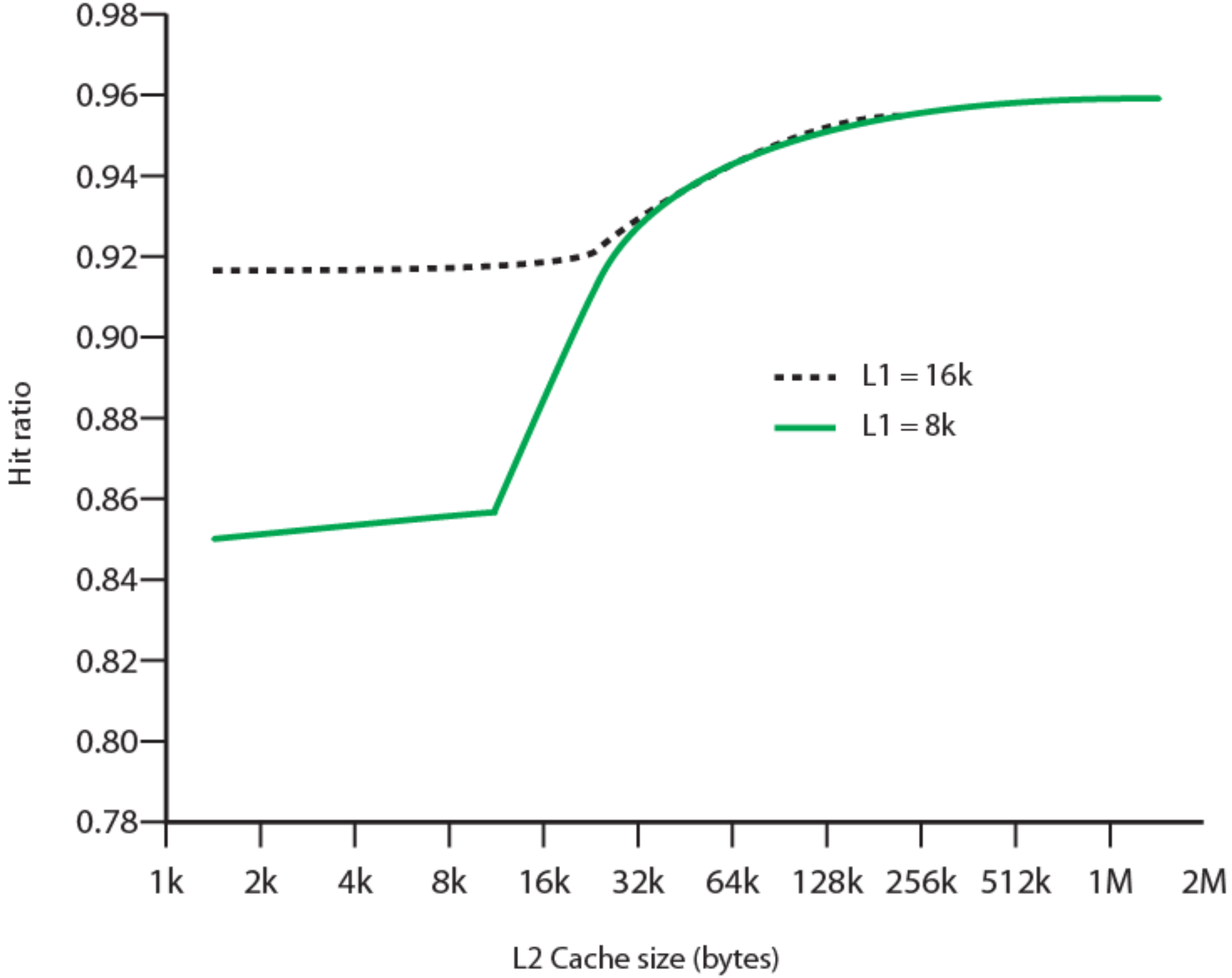
Line Size

- Retrieve not only desired word but a number of adjacent words as well
- Increased block size will increase hit ratio at first
  - the principle of locality
- Hit ratio will decreases as block becomes even bigger
  - Probability of using newly fetched information becomes less than probability of reusing replaced
- Larger blocks
  - Reduce number of blocks that fit in cache
  - Data overwritten shortly after being fetched
  - Each additional word is less local so less likely to be needed
- No definitive optimum value has been found
- 8 to 64 bytes seems reasonable
- For HPC systems, 64- and 128-byte most common

Multilevel Caches

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

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



Unified v Split Caches

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

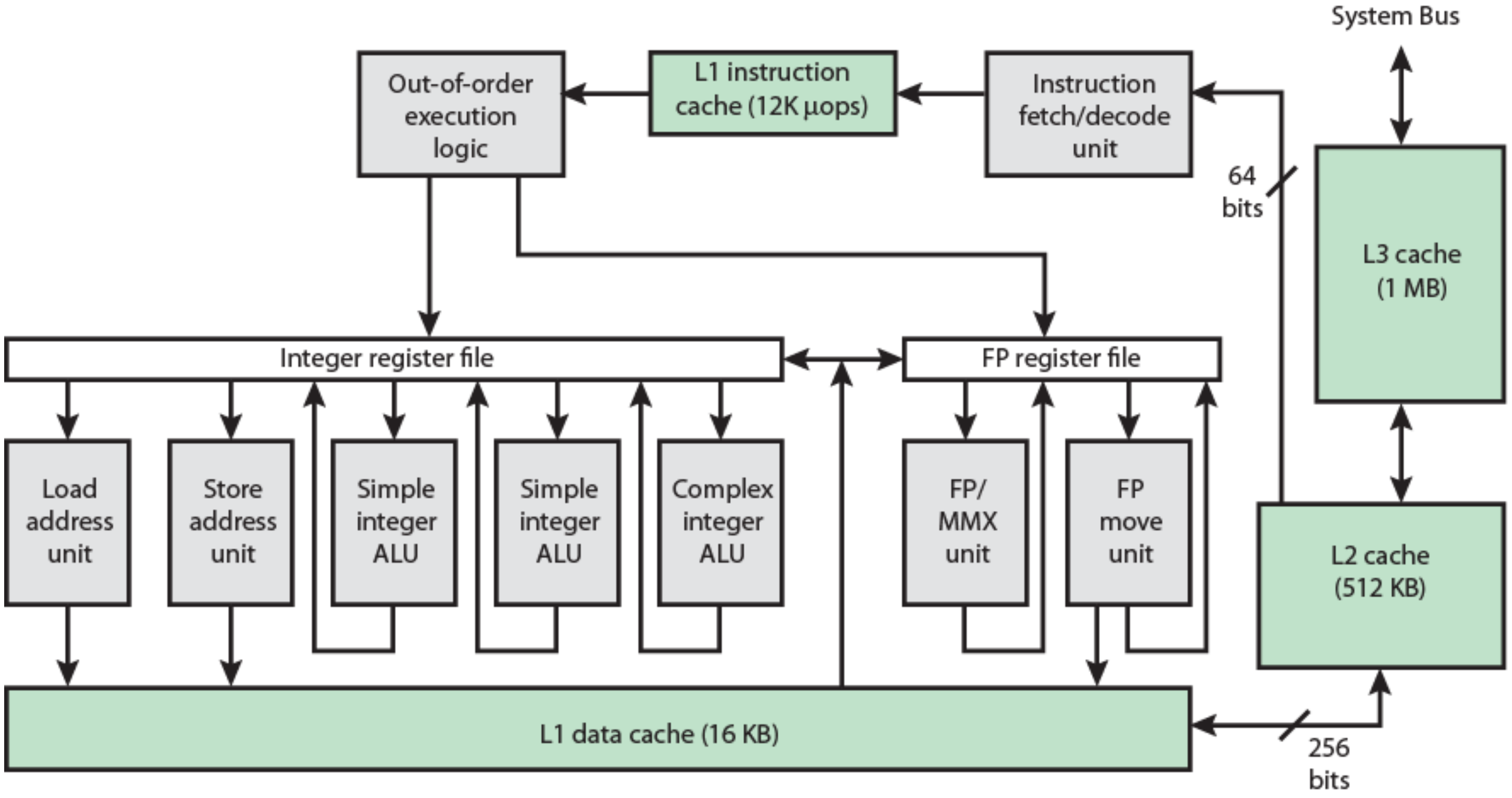
Pentium 4 Cache

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

Intel Cache Evolution

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

Pentium 4 Block Diagram



Pentium 4 Core Processor

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

Pentium 4 Design Reasoning

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