William Stallings
Computer Organization and Architecture
7th Edition

Chapter 4

Cache Memory

Characteristics

- Location
- Capacity
- Adressable units ($2^A=N$) (A:Adr.Line bits)
- Unit of transfer
- Access method
- Performance
- Physical type
- Physical characteristics
- Organisation

Location

- CPU
- Internal (main)
- External (secondary)

Capacity

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

Unit of Transfer

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

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
  - Random in nature. 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

Performance

- Access time
  - Time between presenting the address and getting the valid data (Random type), time to position read-write mechanism (non random)
- 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 (for random access =1/cycle time)

Method 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
- Access time is variable

Direct access

- Involves a shared read-write mechanism
- Individual blocks or records have a unique address based on physical location
- Access time is variable

Random access

- Each addressable location in memory has a unique, physically wired-in addressing mechanism
- The time to access a given location is independent of the sequence of prior accesses and is constant
- Any location can be selected at random and directly addressed and accessed
- Main memory and some cache systems are random access

Associative

- A word is retrieved based on a portion of its contents rather than its address
- Each location has its own addressing mechanism and retrieval time is constant independent of location or prior access patterns
- Cache memories may employ associative access

Capacity and Performance:

The two most important characteristics of memory

Three performance parameters are used:

Access time (latency)

- For random-access memory it is the time it takes to perform a read or write operation
- For non-random-access memory it is the time it takes to position the read-write mechanism at the desired location

Memory cycle time

- Access time plus any additional time required before second access can commence
- Additional time may be required for transients to die out on signal lines or to regenerate data if they are read destructively
- Concerned with the system bus, not the processor

Transfer rate

- The rate at which data can be transferred into or out of a memory unit
- For random-access memory it is equal to 1/(cycle time)

For non random access memory:

$T_N = T_A + N/R$

T_N : Average time to read or write N bits

T_A : Average access time

N : Number of bits

R : Transfer rate in bps

The Bottom Line

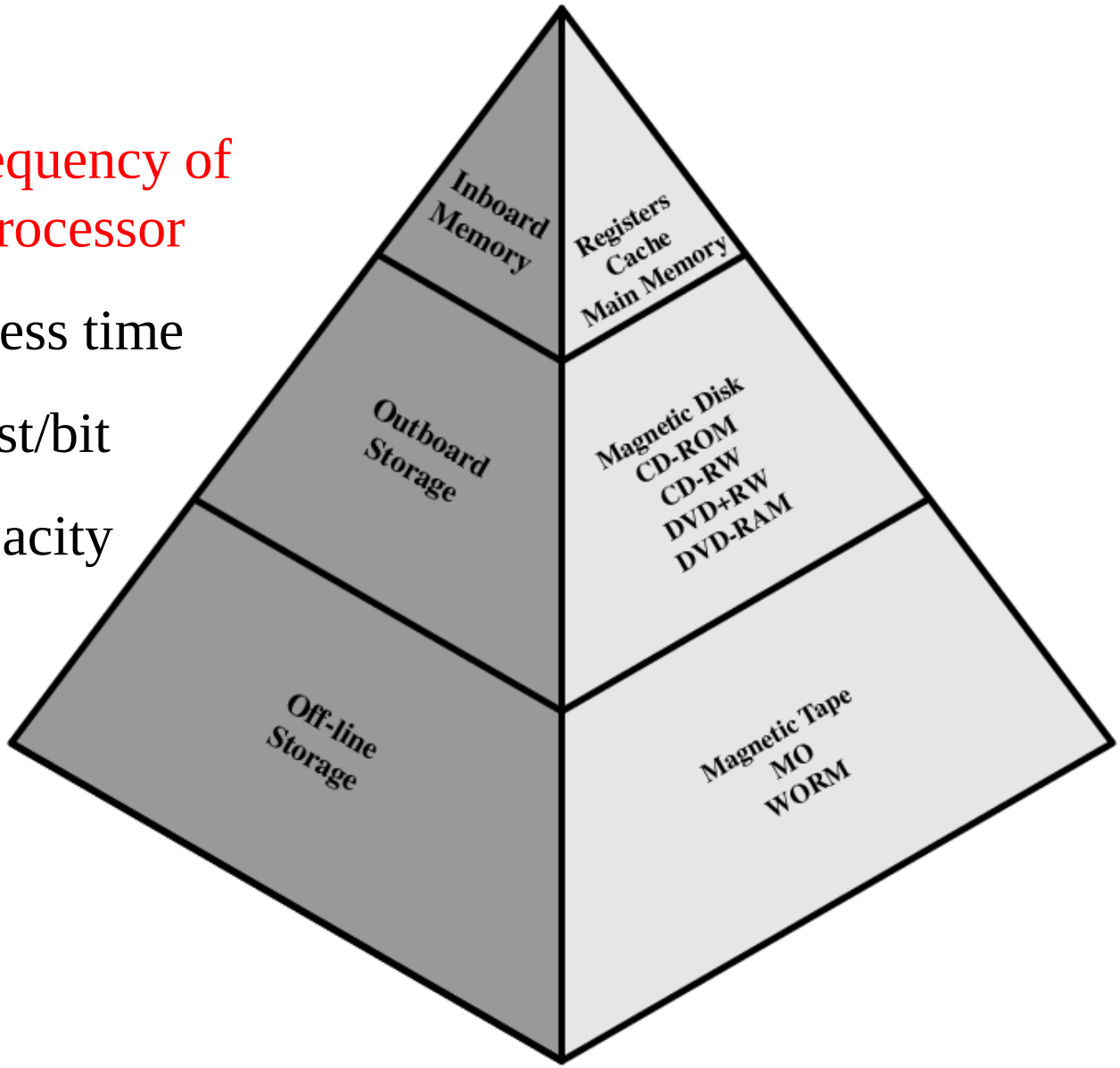
- How much?
  - Capacity (open ended)
- How fast?
  - Time is money (keep up with the processor)
- How expensive?
  - Reasonably priced in comparison to other components

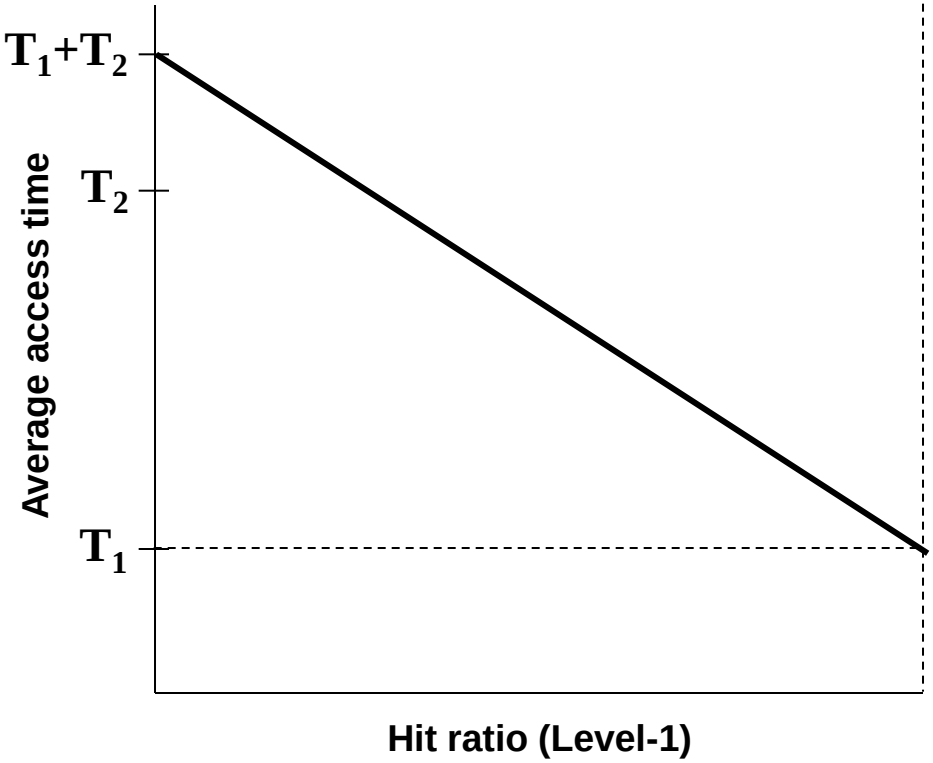
Memory Hierarchy

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

Memory Hierarchy - Diagram

- Decraesing frequency of access by the processor
- Increasing access time
- Decraesing cost/bit
- Increasing capacity





Level-1	1000 words	$T_1 = 0.01\mu s$
Level-2	100,000 words	$T_2 = 0.1\ \mu s$

If 95% found in cache: Average access time = (0.95)(0.01)+(0.05)(0.01+0.1) = 0.015 μs

Physical Types

- Semiconductor
  - RAM
- 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

Hierarchy List

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

So you want fast?

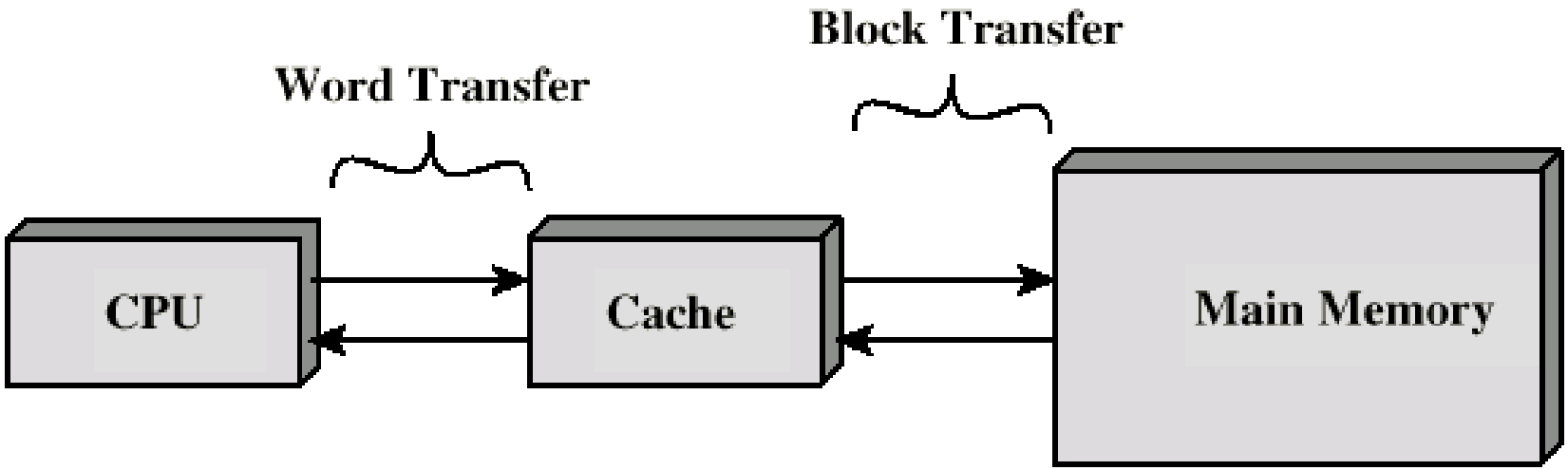
- It is possible to build a computer which uses only static RAM (see later)
- This would be very fast
- This would need no cache
  - How can you cache cache?
- This would cost a very large amount

Locality of Reference

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

Cache

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



Cache and Main Memory

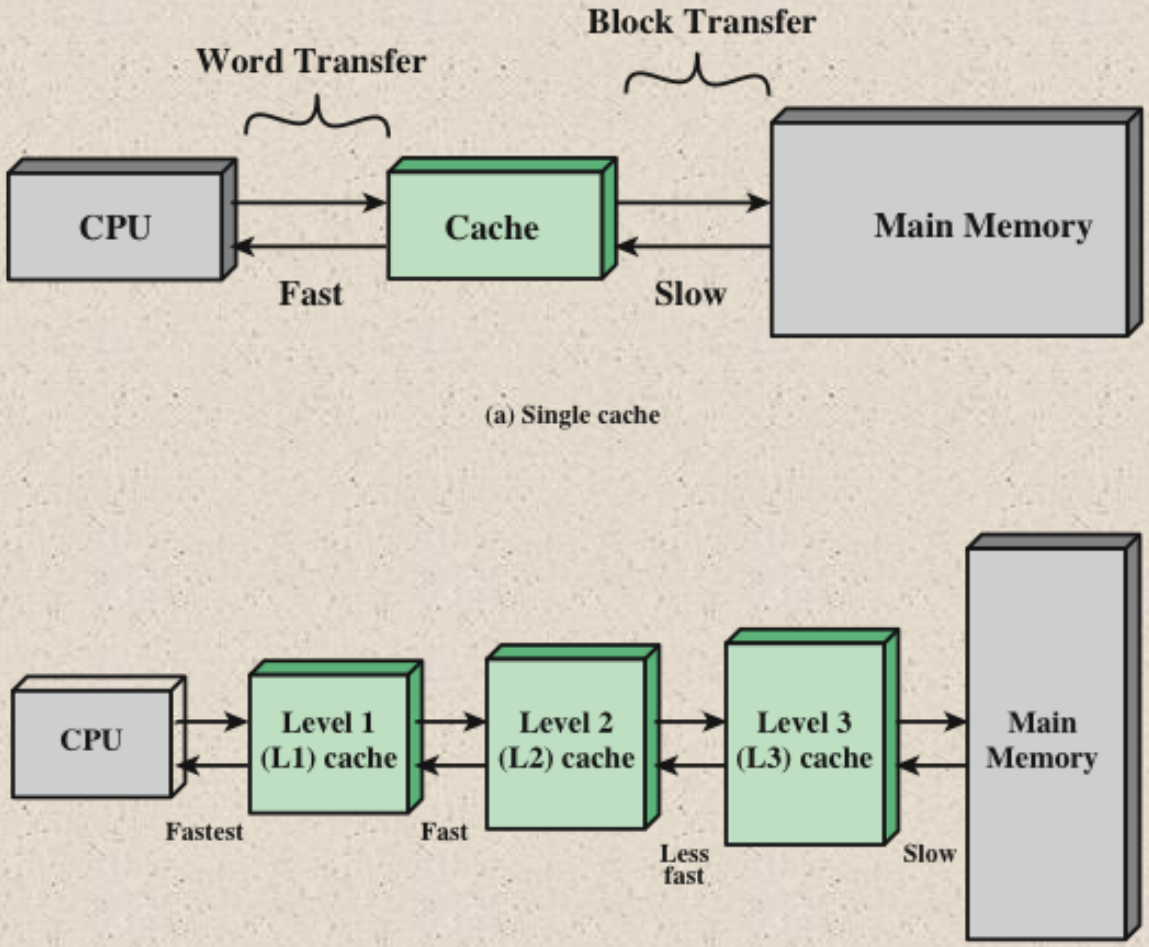
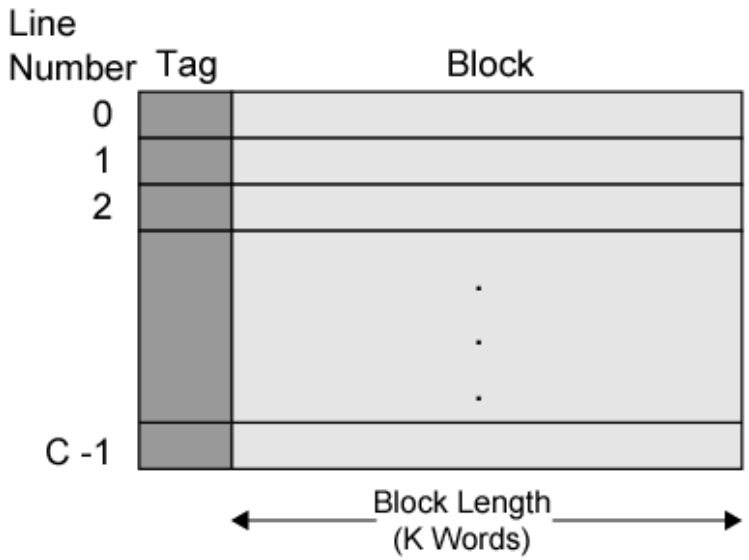
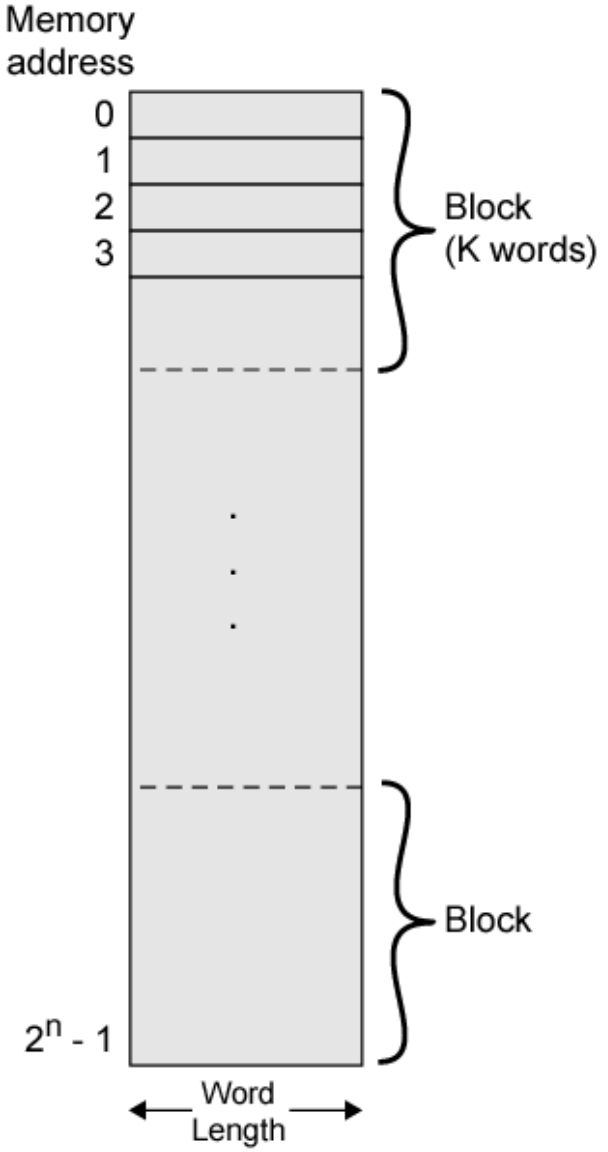


Figure 4.3 Cache and Main Memory

Cache/Main Memory Structure



(a) Cache



(b) Main memory

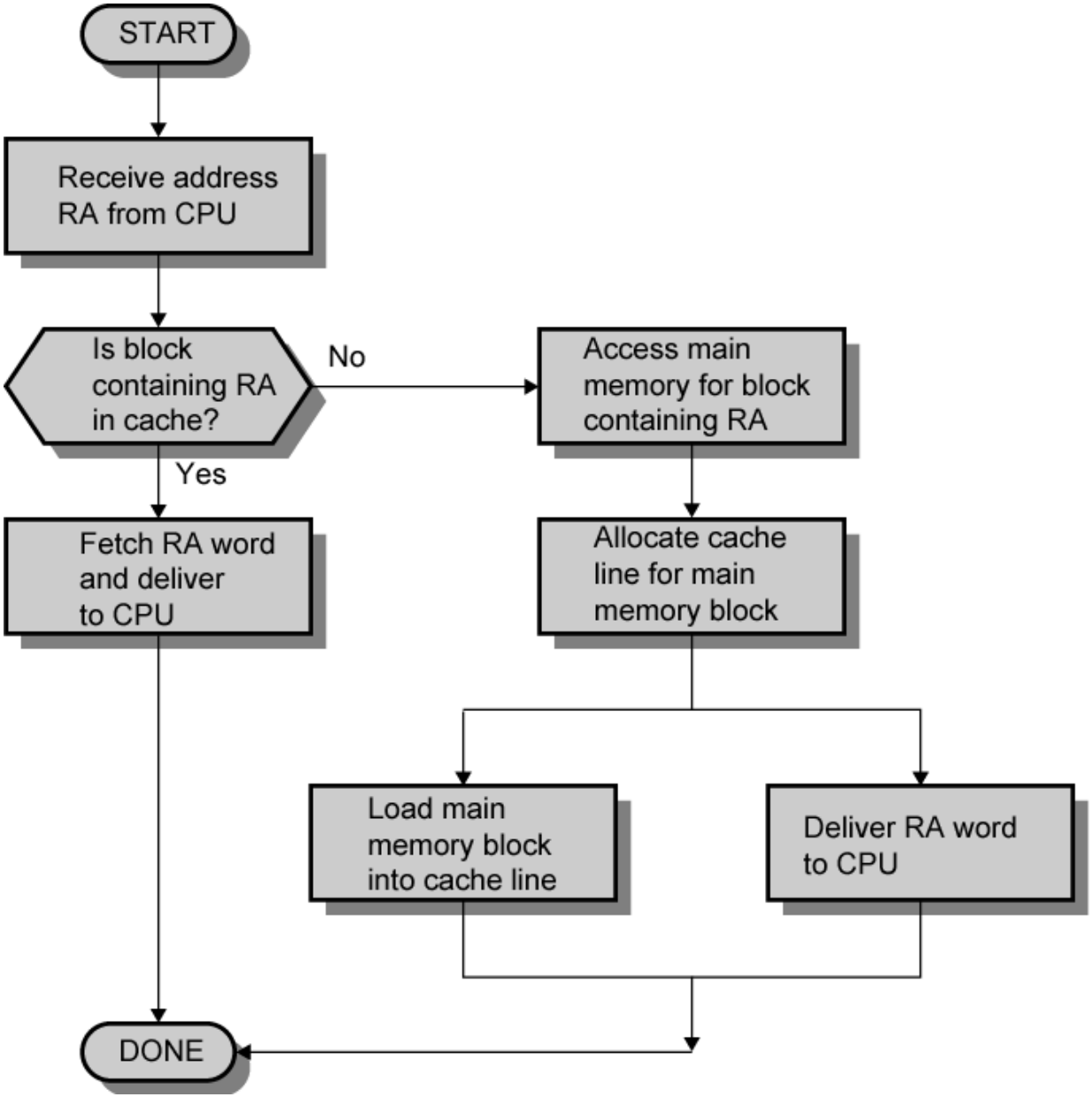
There are $M = 2^n/K$ Blocks in main memory

$C << M$

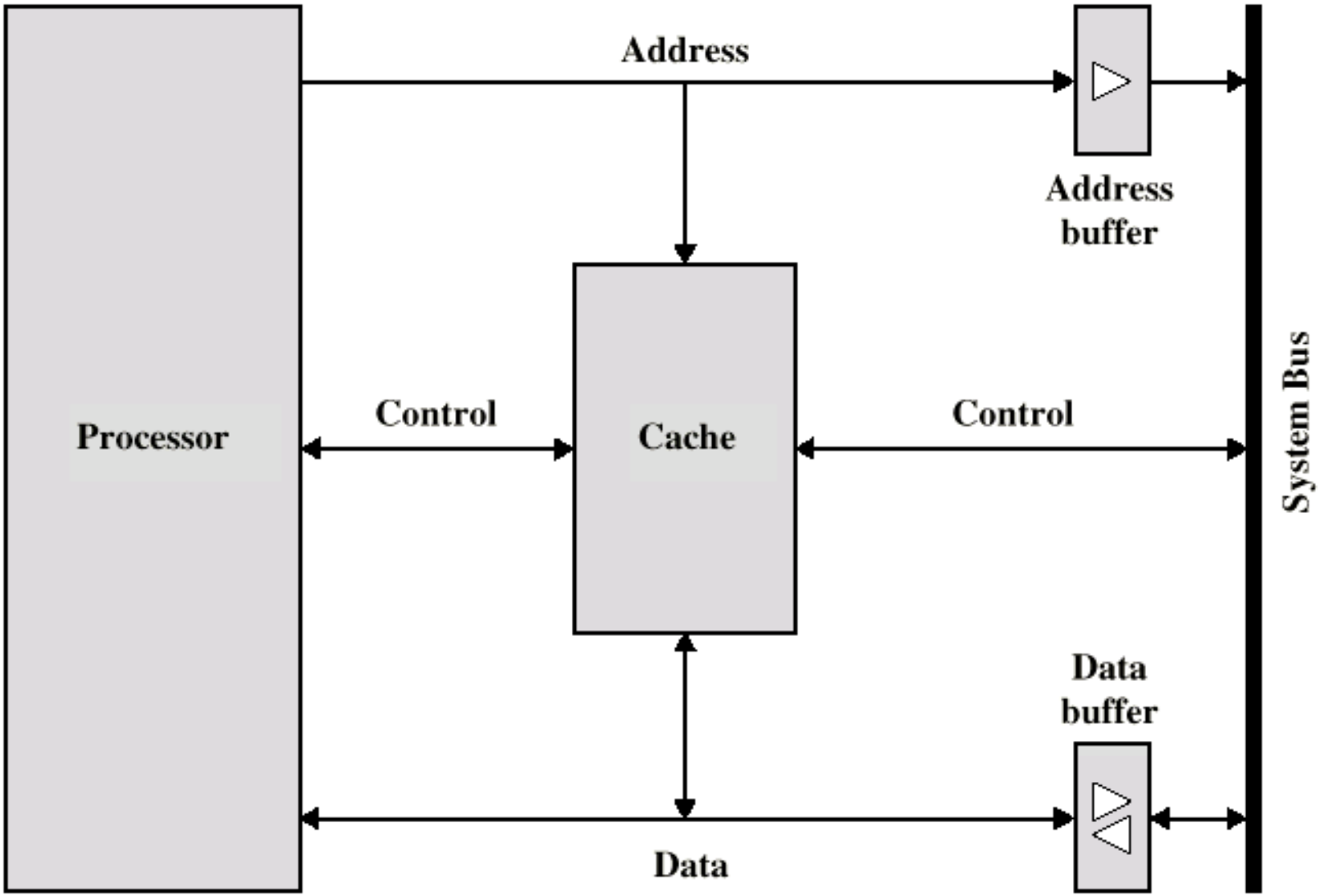
Cache operation – overview

- 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



Typical Cache Organization



Elements of Cache Design

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

Table 4.2 Elements of Cache Design

Size does matter

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

Cache size:

Small enough, average cost/bit ≅ to that of main memory alone

Large enough, average access time ≅ to that of cache alone

Large caches tend to be slightly slower because of the increasing number of addressing gates

Table 4.3

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	—	—
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 G6	Mainframe	1999	256 kB	8 MB	—
Pentium 4	PC/server	2000	8 kB/8 kB	256 KB	—
IBM SP	High-end server/ supercomputer	2000	64 kB/32 kB	8 MB	—
CRAY MTA[b]	Supercomputer	2000	8 kB	2 MB	—
Itanium	PC/server	2001	16 kB/16 kB	96 KB	4 MB
Itanium 2	PC/server	2002	32 kB	256 KB	6 MB
IBM POWER5	High-end server	2003	64 kB	1.9 MB	36 MB
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	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

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

[b] Both caches are instruction only; no 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
- Three techniques can be used:

Direct

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

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

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

Mapping Function

- Cache of 64kByte
- Cache block of 4 bytes
  - —i.e. cache is 16k (2^{14}) lines of 4 bytes
- 16MBytes main memory
- 24 bit address
  - —(2^{24}=16M)
- Main memory consists of 4M blocks of 4 byte each.

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

i = j modulo m

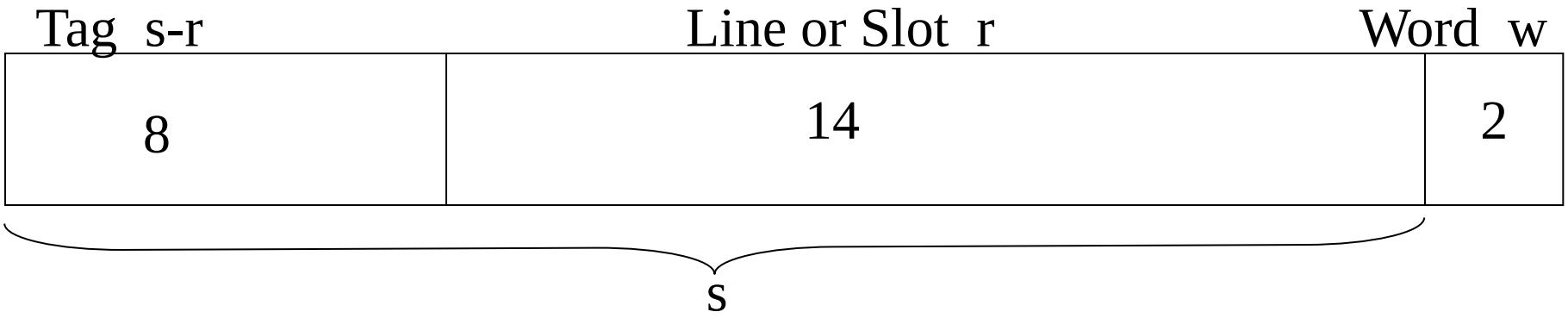
i = cache line number

j = main memory block number

m = number of lines in cache

- 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

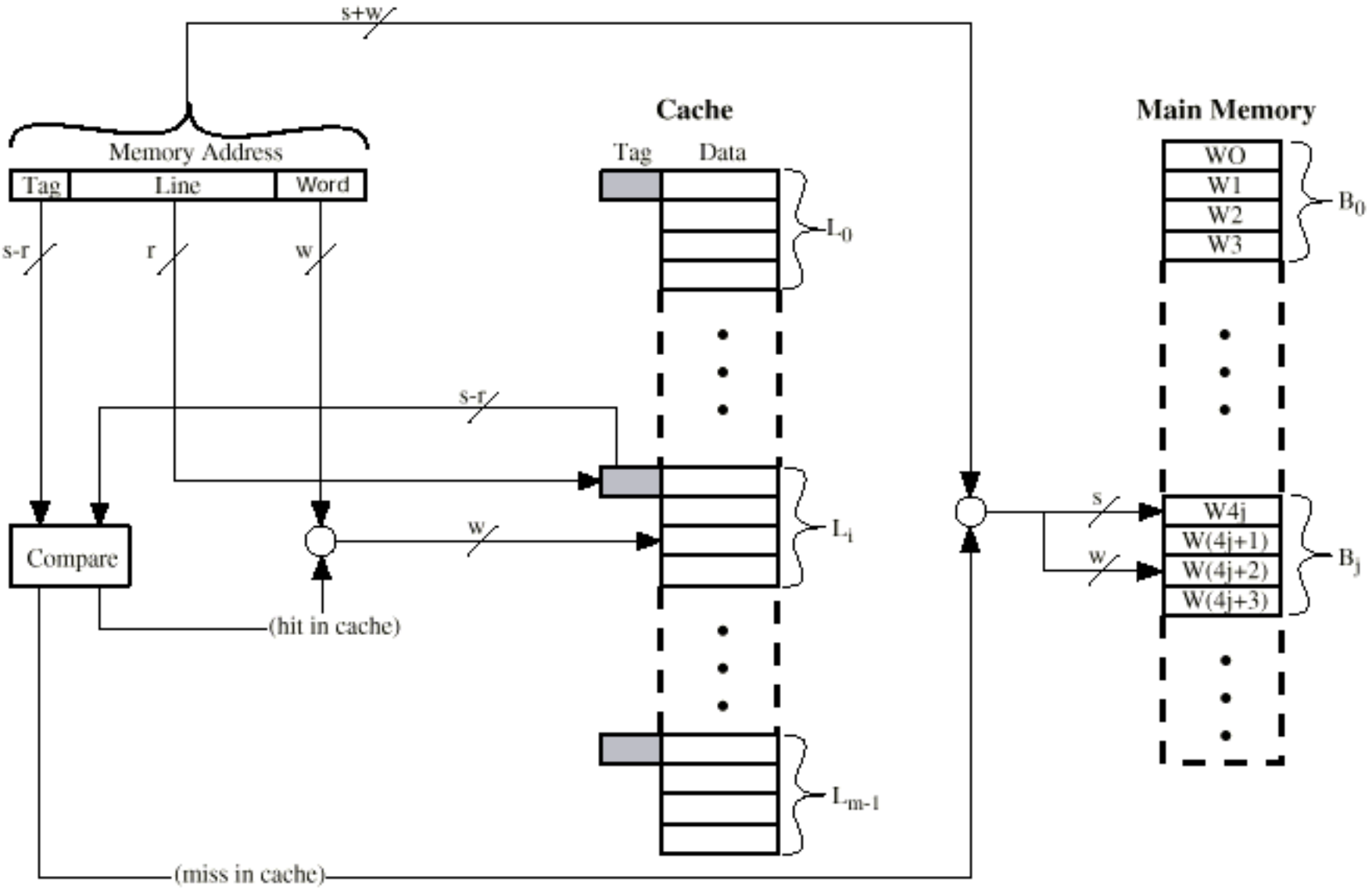


- 24 bit address (2^{24} = 16M main memory)
- 2 bit word identifier (2^2 = 4 byte block)
- 22 bit block identifier (s)
  - 8 bit tag (s-r =22-14)
  - 14 bit slot or line
- No two blocks in the same line have the same Tag field
- Check contents of cache by finding line and checking Tag

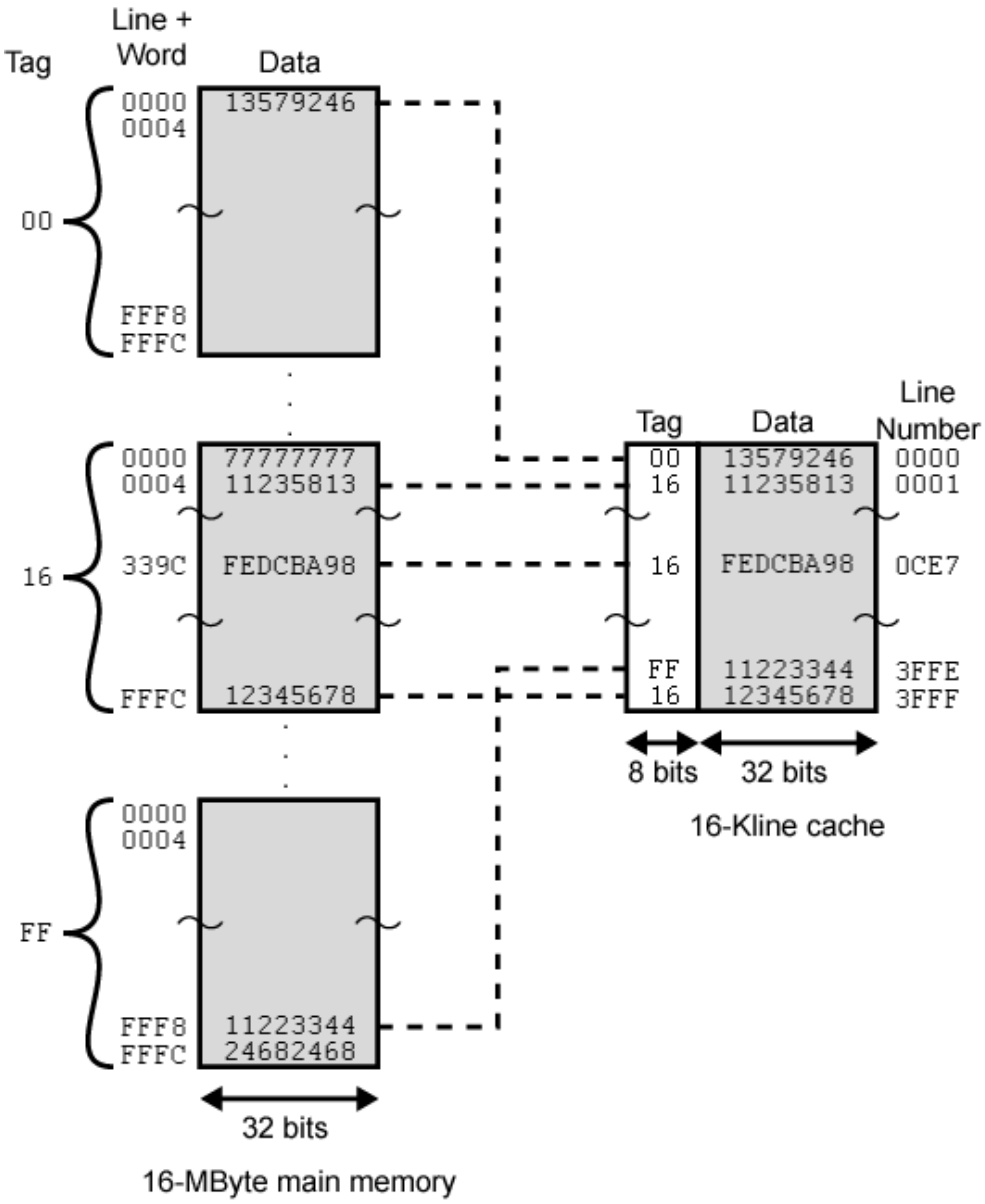
Direct Mapping Cache Line Table

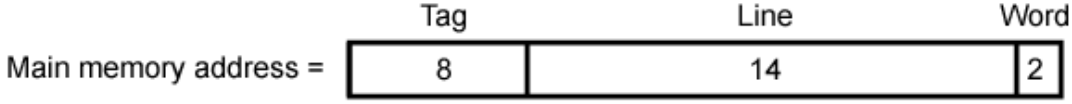
Cache line	Main Memory blocks held
0	0, m, 2m, 3m…2^s-m
1	1,m+1, 2m+1…2^s-m+1
m-1	m-1, 2m-1,3m-1…2^s-1

Direct Mapping Cache Organization



Direct Mapping Example





Direct Mapping Example

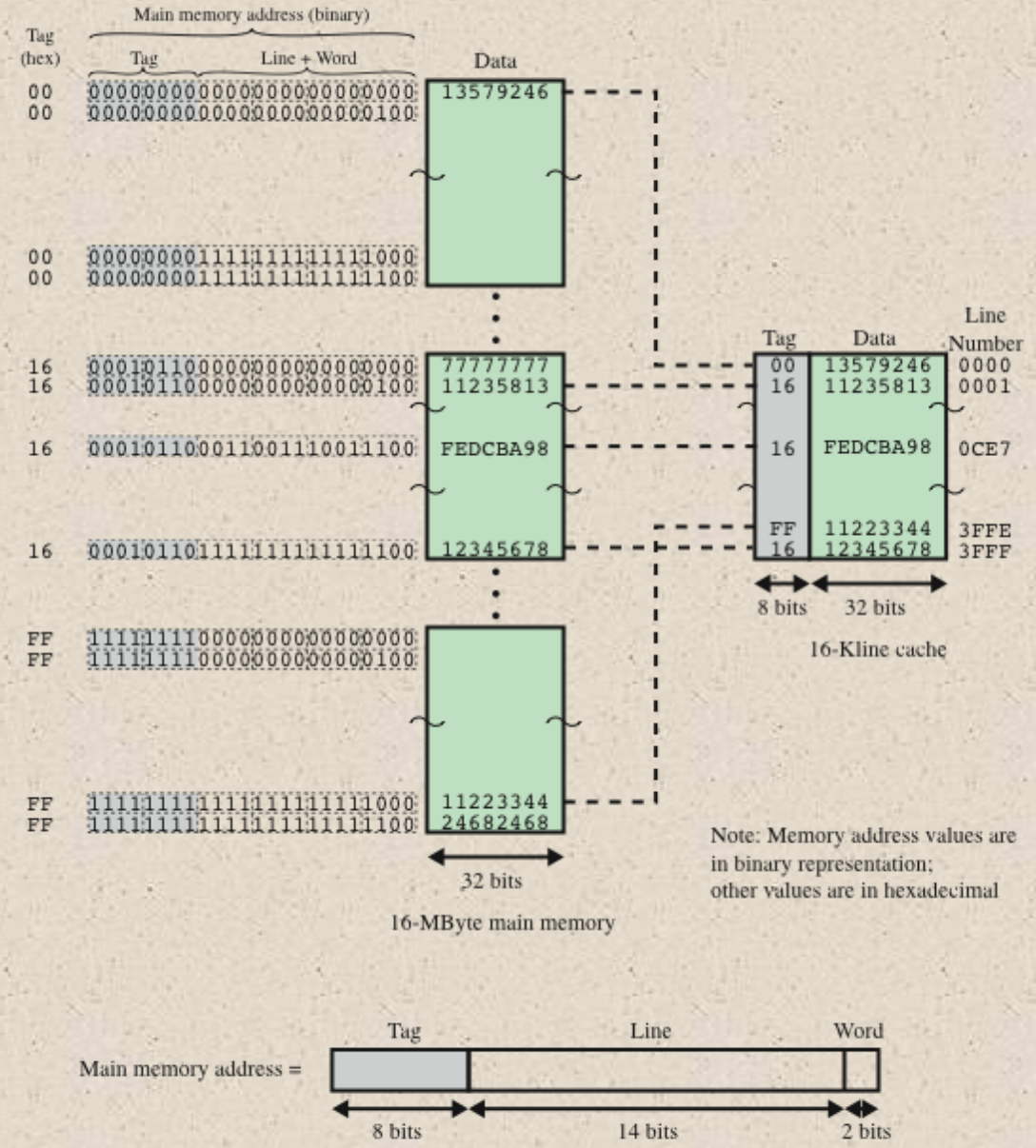


Figure 4.10 Direct Mapping Example

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
- 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

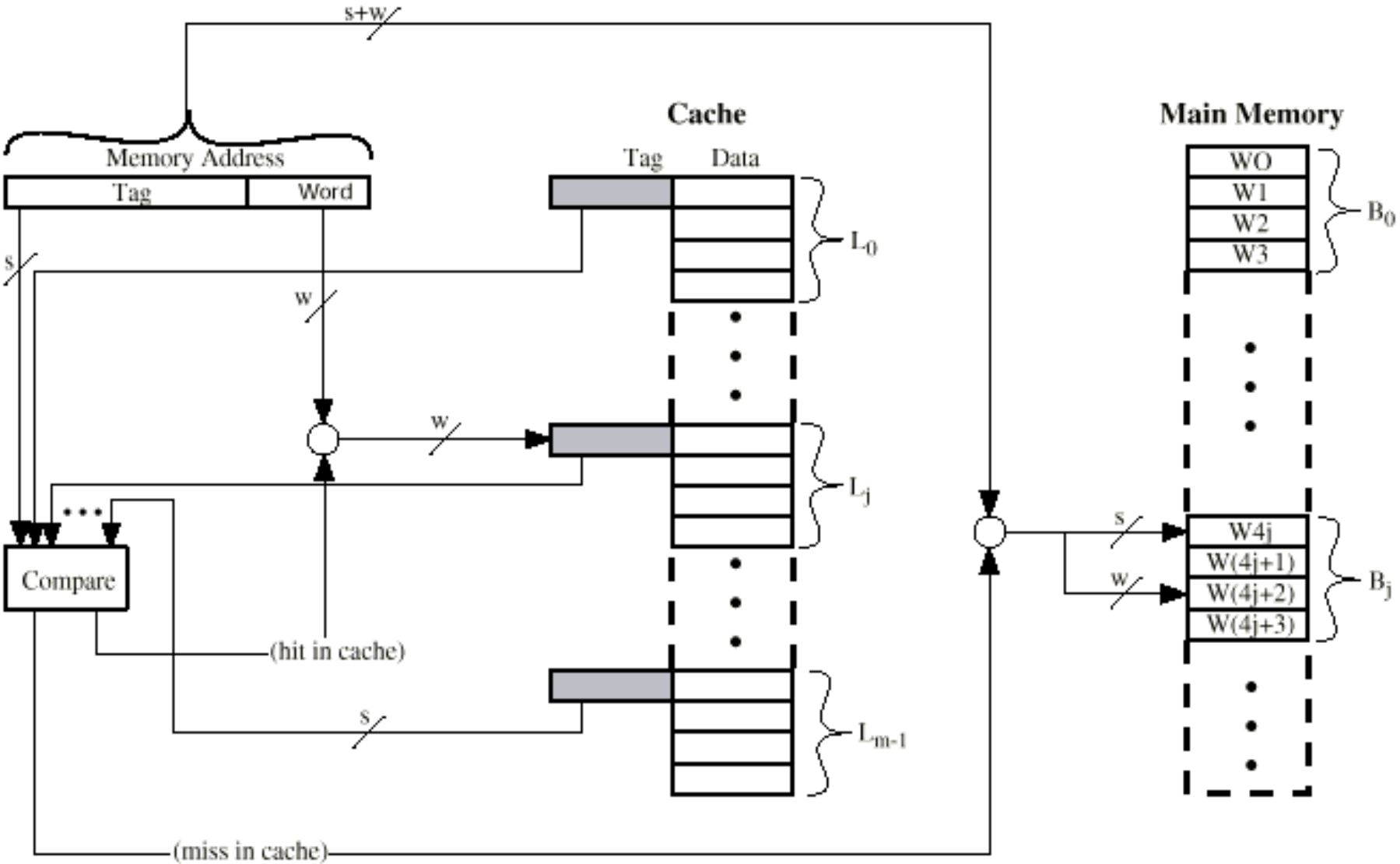
Direct Mapping pros & cons

- 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 (trashing)

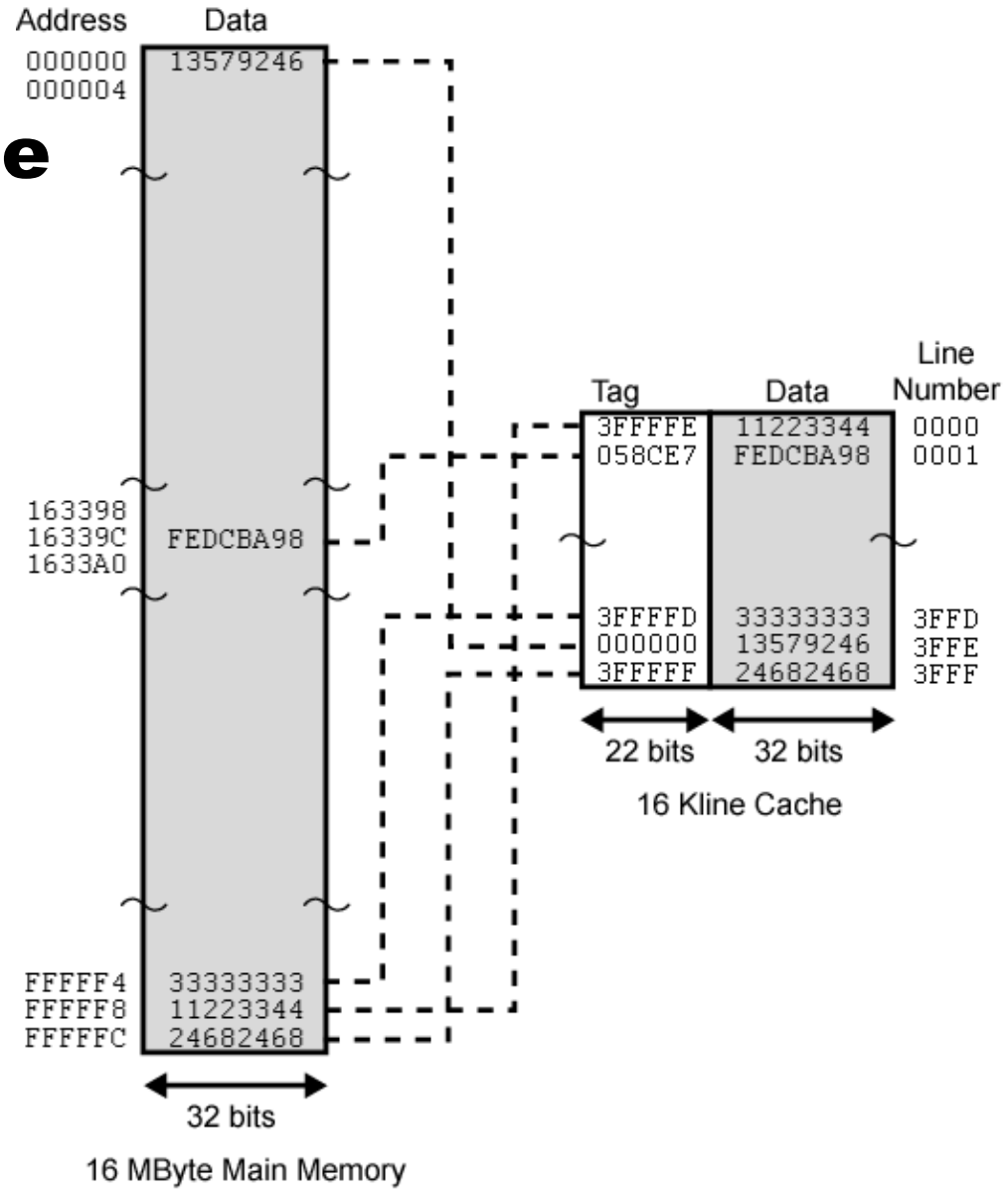
Associative Mapping

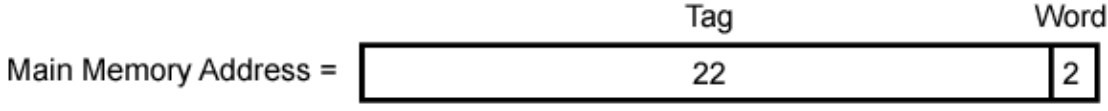
- A main memory block can load into any line of cache
- Memory address is interpreted as tag and word
- Tag uniquely identifies block of memory
- Every line’s tag is examined for a match
- Cache searching gets expensive

Fully Associative Cache Organization



Associative Mapping Example





Associative Mapping Example

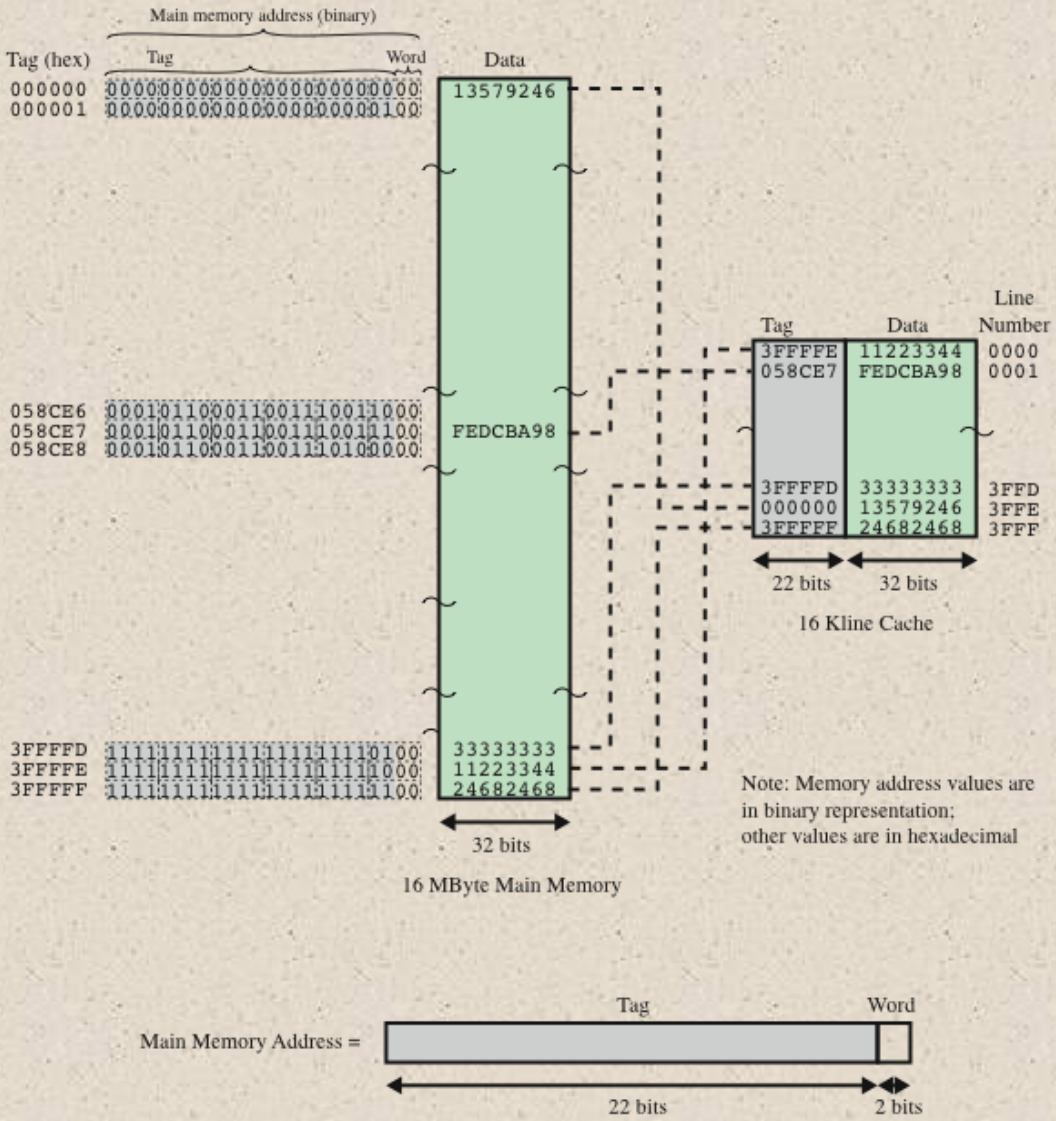


Figure 4.12 Associative Mapping Example

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
- Least significant 2 bits of address identify which 8 bit word is required from 32 bit data block
- e.g.
  - Address Tag Data Cache line
  - FFFFFC 3FFFFF 24682468 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 = not determined by the address format
- 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
  - m = k v
  - i = j modulo v

i = cache line number

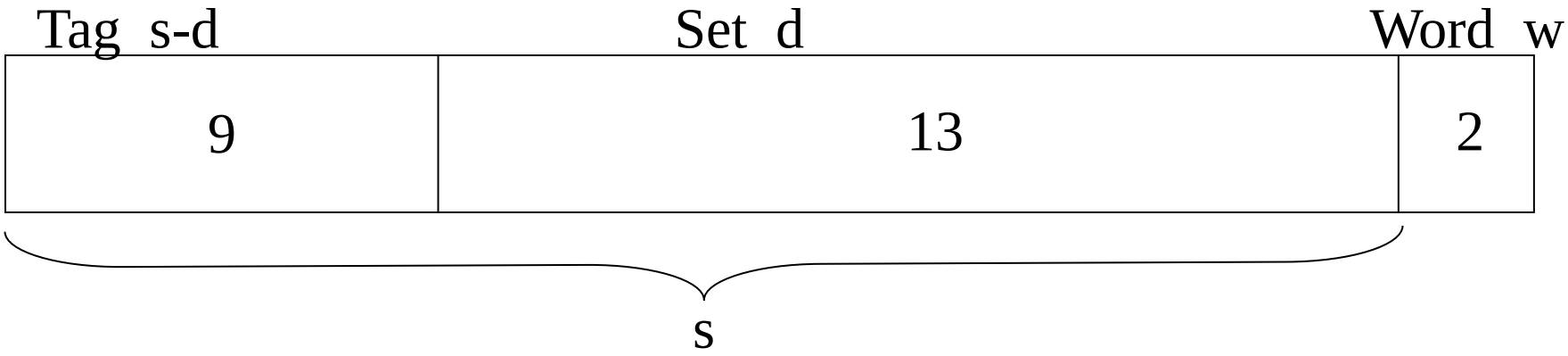
j = main memory block number

m = number of lines in cache

- 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 Summary

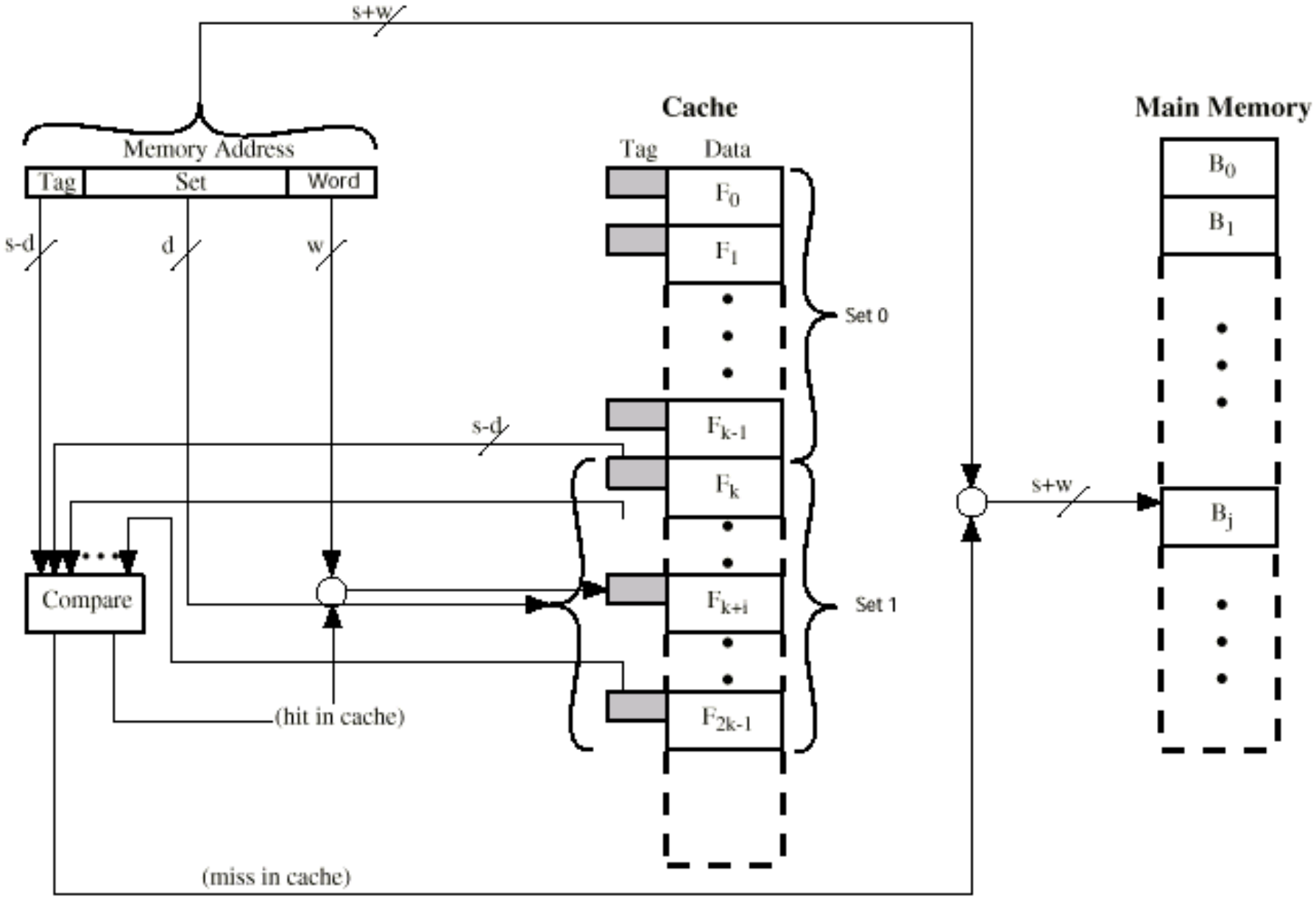
- Address length = (s + w) bits
- Number of addressable units = 2^{s+w} words or bytes
- Block size = line size = 2^{w} words or bytes
- Number of blocks in main memory =
- $2^{s+w}/2^{w} = 2^{s}$
- Number of lines in set = k (k-way set associative mapping)
- Size of set field = d bits
- Number of sets = v = 2^{d}
- Number of lines in cache = kv = k 2^{d}
- Size of tag = (s – d) bits


Tag s-d
Set d
Word w
9
13
2
s

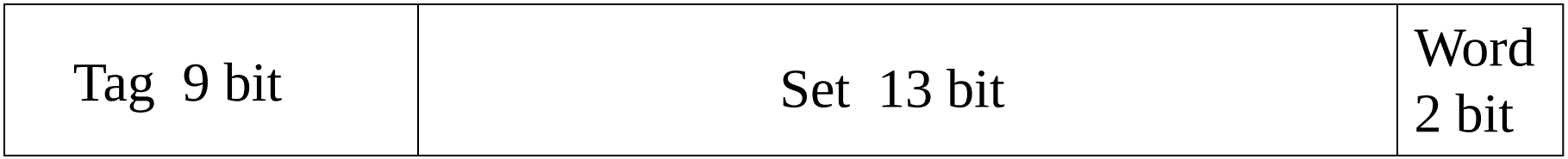
Set Associative Mapping Example

- 13 bit set number
- Block number in main memory is modulo 2^{13}
- 000000, 008000, ..., FF8000 map to same set

k - Way Set Associative Cache Organization

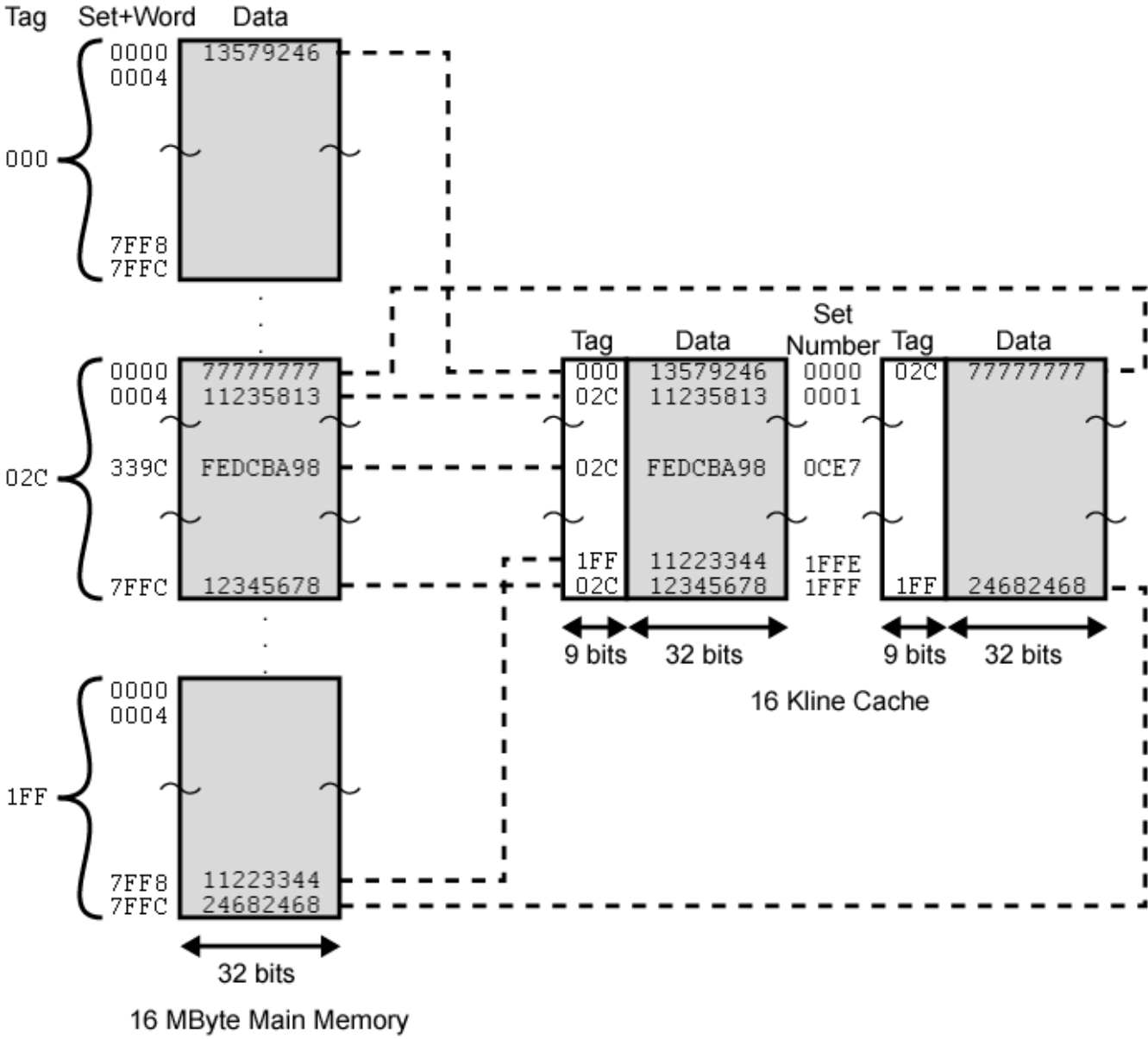


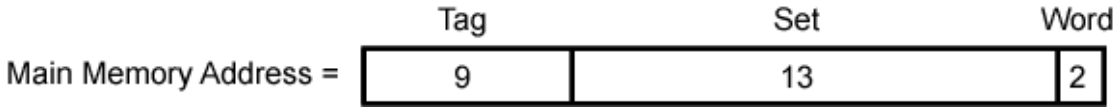
Set Associative Mapping Address Structure



- Use set field to determine cache set to look in
- Compare tag field to see if we have a hit
- e.g
  - Address number — Tag — Data — Set
  - 1FF 7FFC — 1FF — 24682468 — 1FFF
  - 02C 7FFC — 02C — 12345678 — 1FFF

Two Way Set Associative Mapping Example





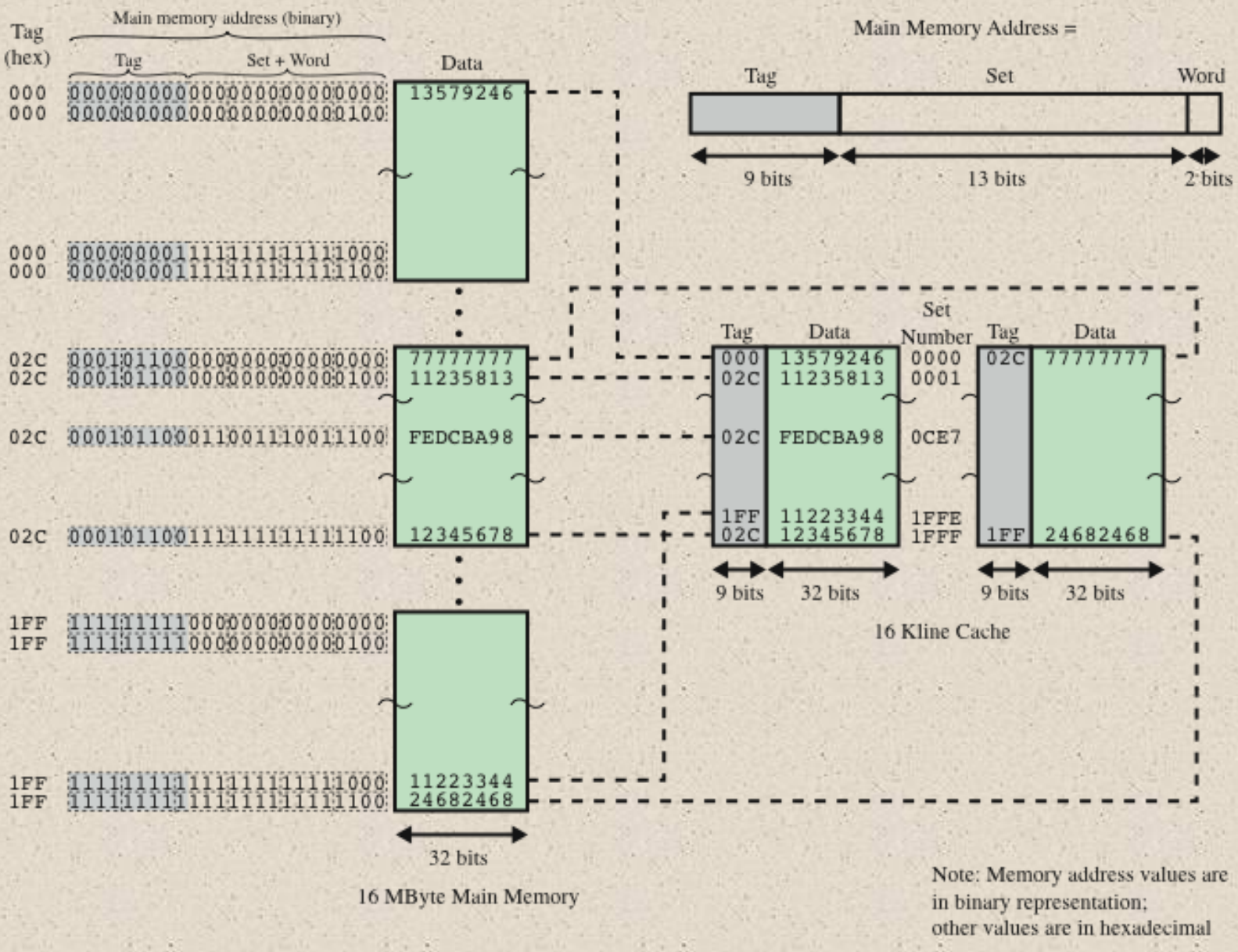


Figure 4.15 Two-Way Set Associative Mapping Example

In the extreme case of :

$v = m, \quad k = 1$

set associative mapping reduces to direct mapping

and for:

$v = 1, k = m$

it reduces to fully associative mapping

2-way organization is the most common set associative organization ($v = m/2, k = 2$).

4-way organization ($v = m/4, k = 4$) makes a little improvement for a relatively small additional cost.

Varying Associativity Over Cache Size

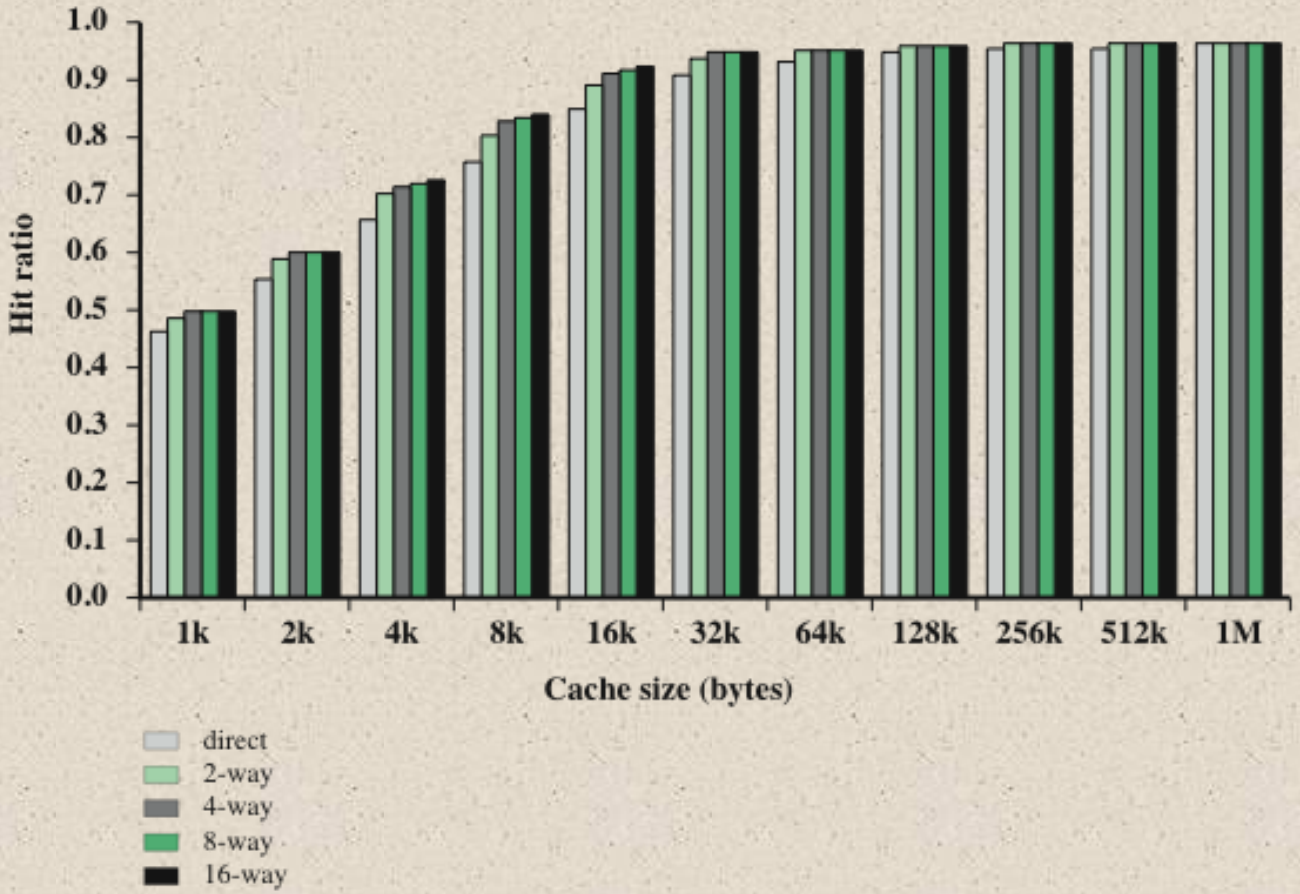
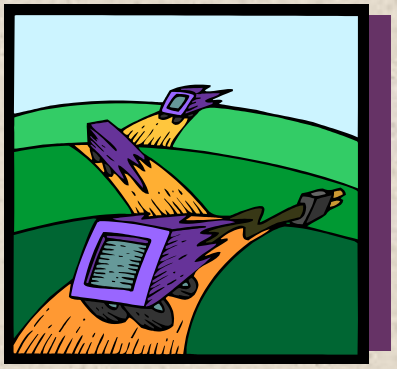


Figure 4.16 Varying Associativity over Cache Size

Replacement Algorithms (1)
Direct mapping

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

Replacement Algorithms



- 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 Algorithms (2) Associative & Set Associative

- Hardware implemented algorithm (speed)
- Least Recently used (LRU)
  - Replace the block that stayed longest with no reference to it
- e.g. in 2 way set associative
  - Which of the 2 block is lru? (set the USE bit of the referenced line to 1 and the other to 0)
- First in first out (FIFO)
  - replace block that has been in cache longest
- Least frequently used (LFU)
  - replace block which has had fewest hits
- Random
  - Slightly inferior performance to others

Write Policy

- Must not overwrite a cache block unless main memory is up to date (if the old block has not been altered, it may be overwritten)
- Multiple CPUs may have individual caches
- I/O may address main memory directly
- If a word is altered only in the cache, corresponding memory word is invalid
- If the I/O device has altered main memory, cache word is invalid
- If a word is altered in one cache, it invalidates the corresponding words in other caches (need cache coherency)

Write Policy

When a block that is resident in the cache is to be replaced there are two cases to consider:

If the old 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

There are two problems to contend with:

More than one device may have access to main memory

A more complex problem occurs 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 conceivably invalidate a word in other caches

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
- Lots of traffic
- Slows down writes

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 (portions of memory are invalid)
- ~ 15% of memory references are writes (in HPC %33 - % 50)

Possible approaches to cache coherency:

- Bus watching with write through

  Each cache controller monitors the address lines to detect write operations to memory by other bus masters

- Hardware transparency

  Additional hardware is used to ensure that all updates to memory are reflected to all caches

- Noncacheable memory

  More than one processor share a portion of memory which is designed to be noncacheable.

Line Size

- As the block size increases from very small to larger sizes, the hit ratio will at first increase (principle of locality)
- The hit ratio will begin to decrease, as the block becomes even bigger (the probability of reusing the newly fetched information becomes less than the one that has to be replaced)
- Larger blocks reduce the number of blocks that fit into cache
- As a block becomes larger, each additional word is farther from the requested one.

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

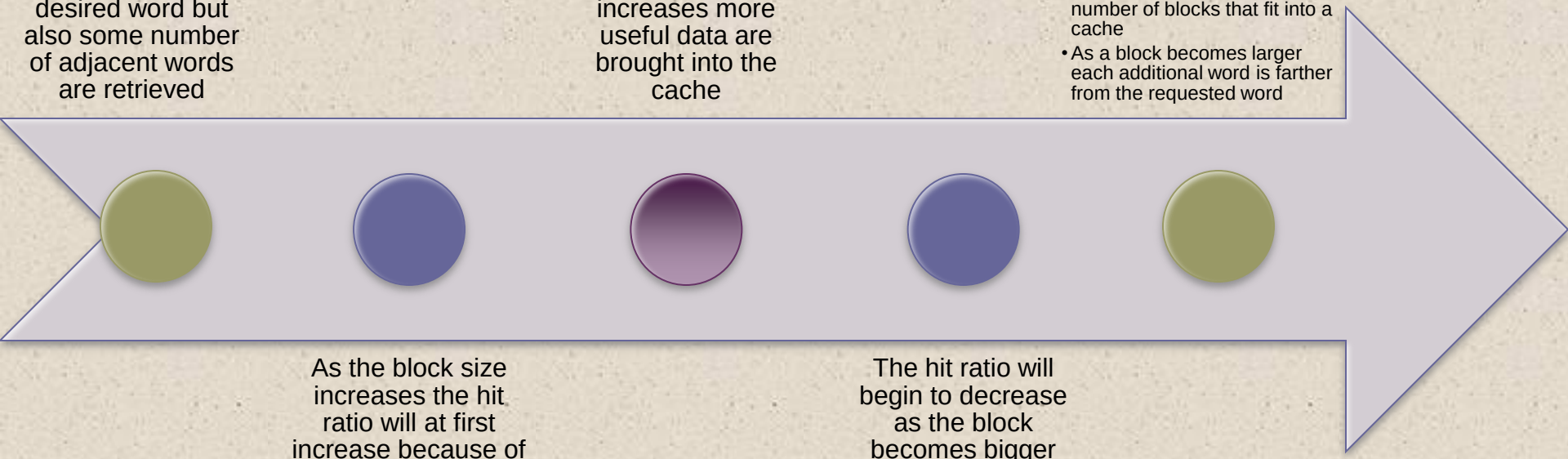
As the block size increases the hit ratio will at first increase because of the principle of locality

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



Number of Caches (Multilevel)

- When caches were originally introduced, a typical system had a single cache (external)
- As the integration density increased, external caches bacame on-chip. That increased speed further (no external bus activity)
- Most microprocessors incorporated L1, L2 and L3 on-chip caches.
- However, the use of multilevel caches does complicate all of the design issues related to caches (size, replacement algorithm, write policy, etc.)

Multilevel 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

Hit Ratio (L1 & L2)
For 8 Kbyte and 16 Kbyte L1

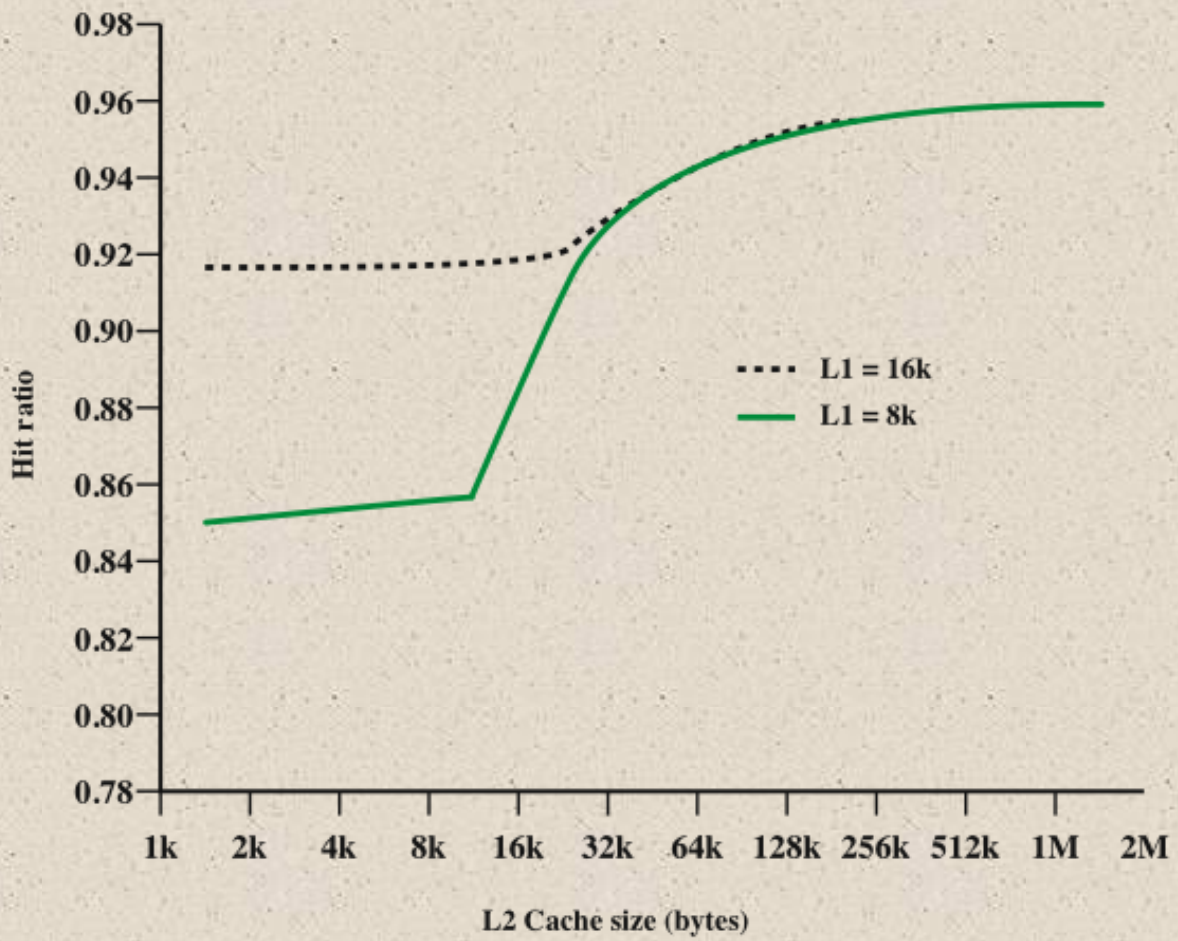


Figure 4.17 Total Hit Ratio (L1 and L2) for 8 Kbyte and 16 Kbyte L1

Split Caches

• Recently, it has become common to split the cache into two: one dedicated to instructions, one dedicated to data.

• Two potential advantages of unified cache:

- It balances the load between instruction and data fetches automatically.

-Only one cache needs to be designed and implemented

• The key advantage of split cache is that it eliminates contention between instruction fetch/decode unit end ececution unit in superscalar machines (parallel instruction execution and the prefetching of predicted future instructions, pipelining)

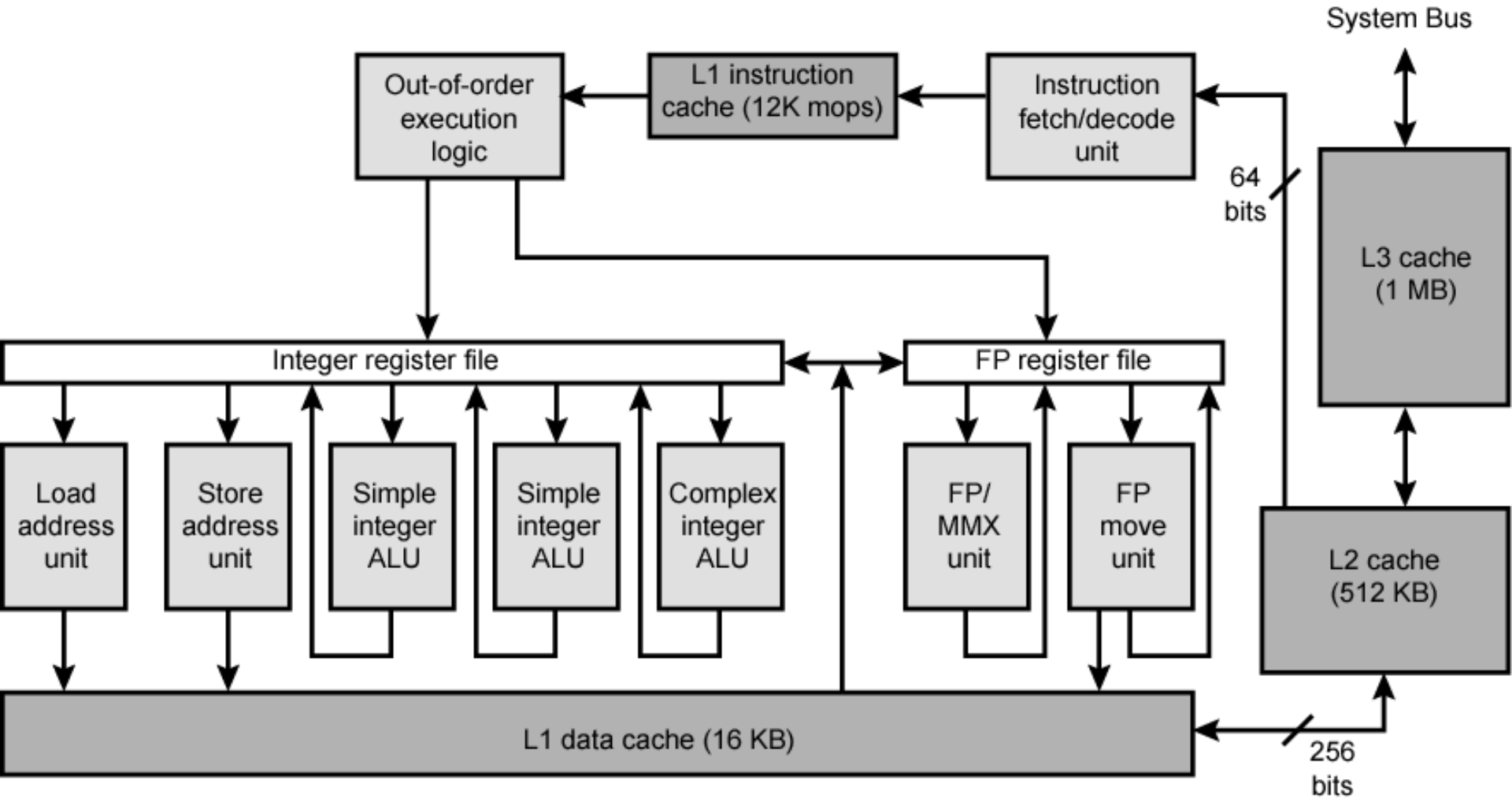
Pentium 4 Cache

- 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-L3 cache and systems bus

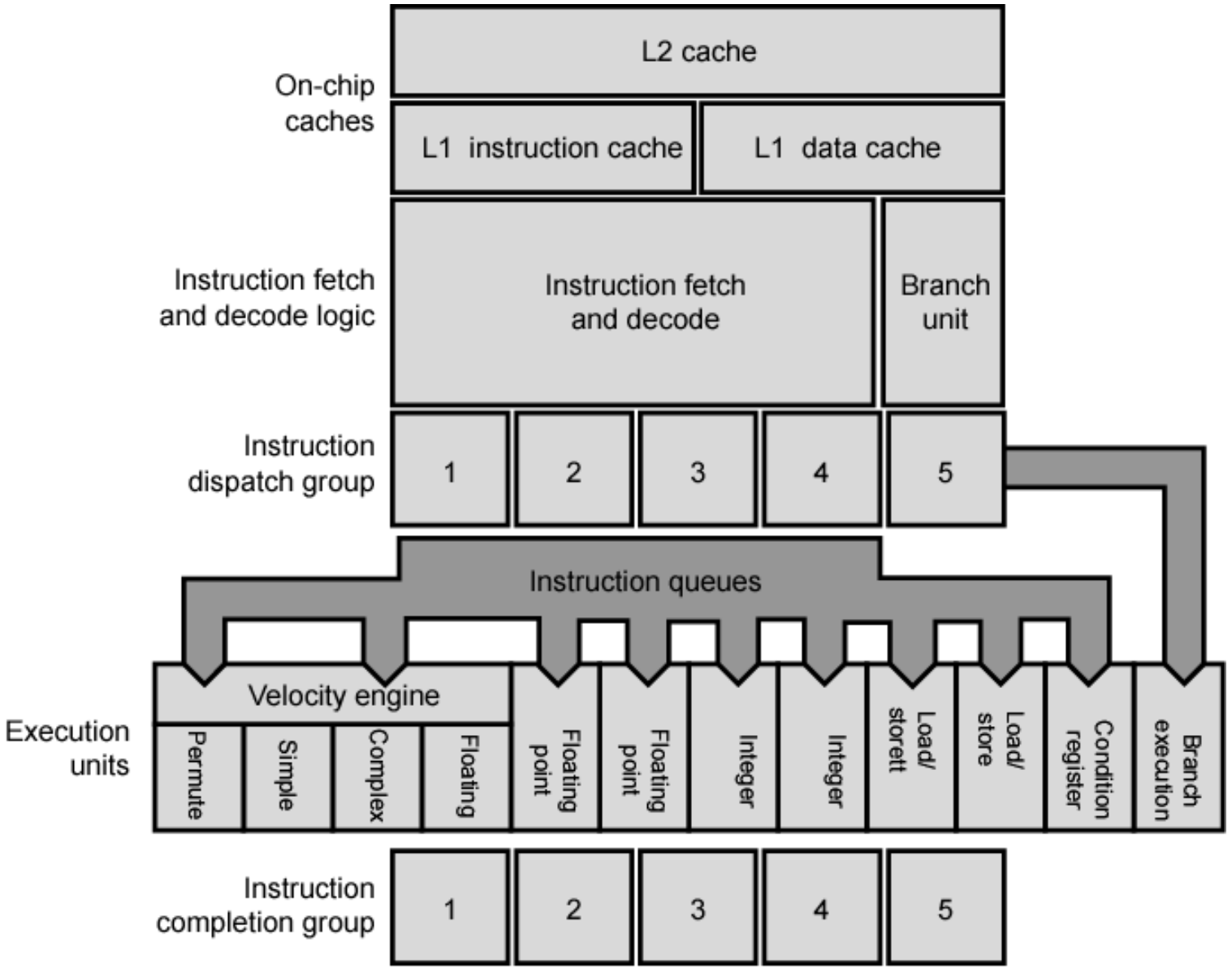
Pentium 4 Design Reasoning

- 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

PowerPC Cache Organization

- 601 – single 32kb 8 way set associative
- 603 – 16kb (2 x 8kb) two way set associative
- 604 – 32kb
- 620 – 64kb
- G3 & G4
  - 64kb L1 cache
    - 8 way set associative
  - 256k, 512k or 1M L2 cache
    - two way set associative
- G5
  - 32kB instruction cache
  - 64kB data cache

PowerPC G5 Block Diagram



Internet Sources

- Manufacturer sites
  - Intel
  - IBM/Motorola
- Search on cache