Cache Memories

Locality

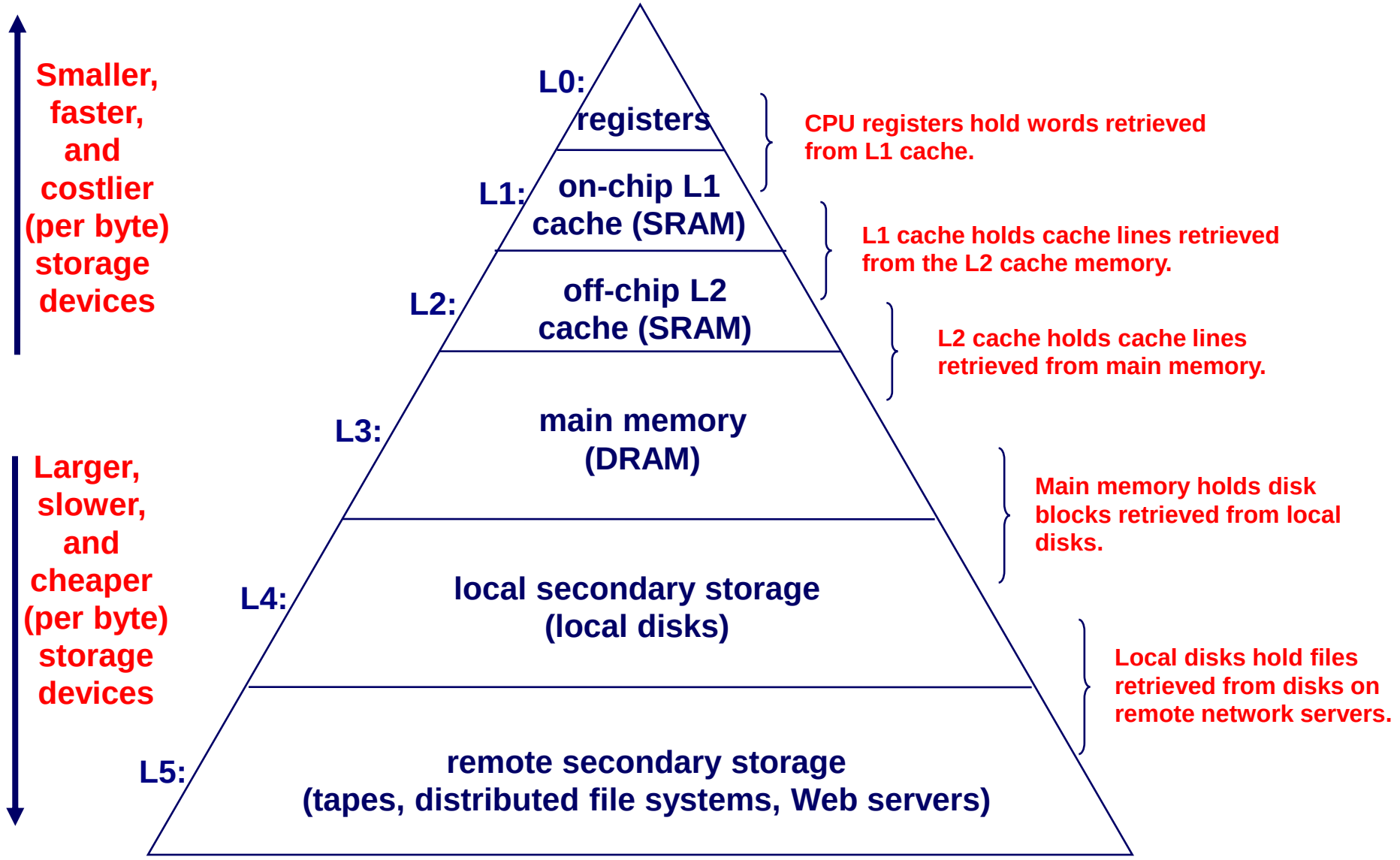
- Principle of Locality:
  - Programs tend to reuse data and instructions near those they have used recently, or that were recently referenced themselves
  - Spatial locality: Instructions in close proximity to a recently executed instruction are likely to be executed soon.
  - Temporal locality: Recently executed instructions is likely to be executed again very soon.

Locality Example:

```
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
return sum;
```

- Data
  - Reference array elements in succession (stride-1 reference pattern): Spatial locality
  - Reference sum each iteration: Temporal locality
- Instructions
  - Reference instructions in sequence: Spatial locality
  - Cycle through loop repeatedly: Temporal locality

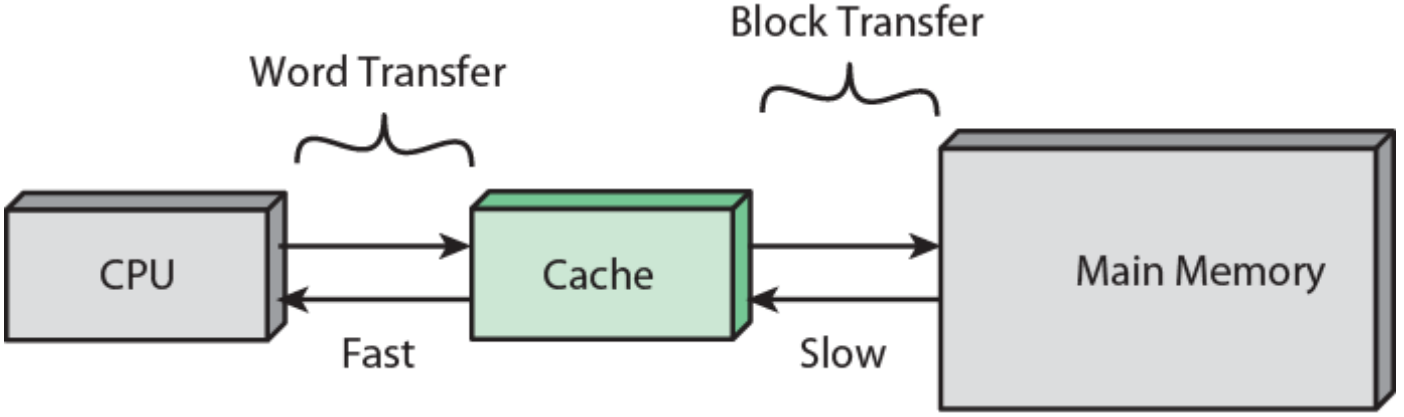
An Example Memory Hierarchy



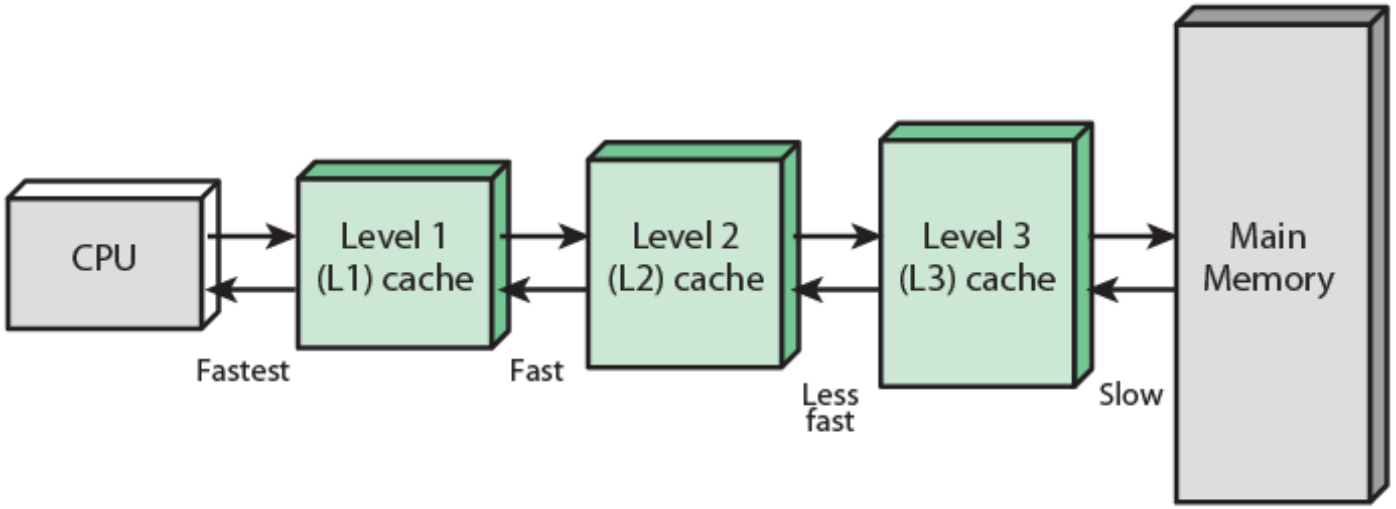
Cache

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

Cache and Main Memory



(a) Single cache



(b) Three-level cache organization

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 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
- Physical cache stores data using main memory physical addresses

Size does matter

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

Mapping Function

- The memory system has to quickly determine if a given address is in the cache
- The cache memory is divided into blocks or lines.
- 1 word = 1 byte
- 1 block = 16 words
- Cache size = 2K words

  No of blocks in cache = 2k words/ 16 words

  = 128 blocks
- main memory = 64K words = 16 bit address

  Where (2^{16}=64 K words)

  No of blocks in main memory = 64k words/16 words = 4096 blocks.

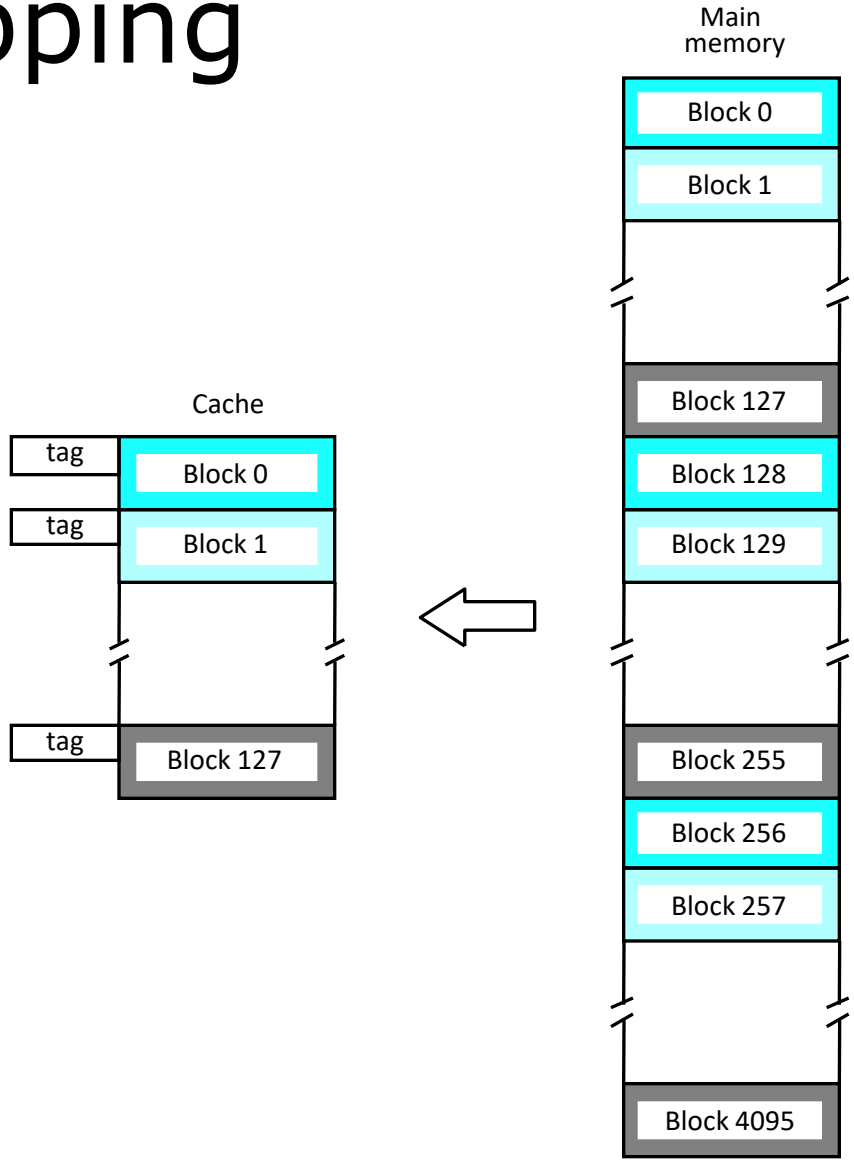
Mapping Function

- Main memory 64K words = 16 bit address
  - 1 word = 1 byte
  - 4 bit word identifier (16 byte block)
  - 12 bit block identifier (16 -4)

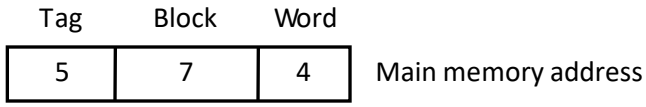
  In main Memory the address has 2 portions.

  1. to identify a word
  2. to identify a block

Direct Mapping



Direct-mapped cache.

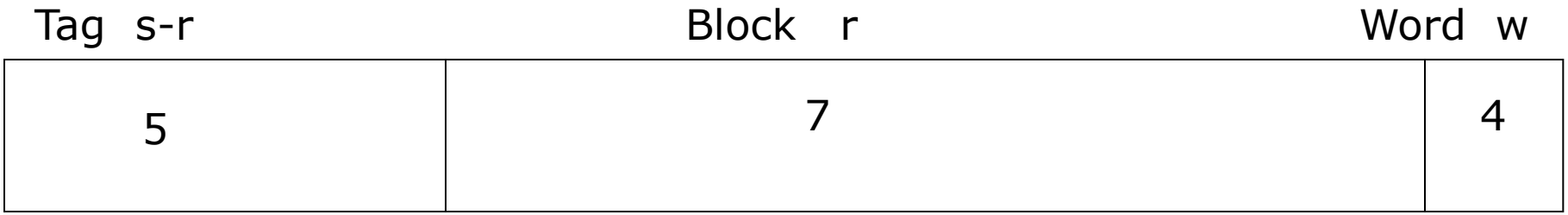


Direct Mapping

- 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 block field r and a tag of s-r (most significant)

Direct Mapping Address Structure



- Main memory 64K words = 16 bit address
  - 1 word = 1 byte
  - 4 bit word identifier (16 byte block)
  - 12 bit block identifier (16 -4)

Cache Memory:

  - 5 bit tag
  - 7 bit block (since 128 block)
  - 4 bit to identify a word in block(since 16 words in block)
- No two blocks in the same line have the same Tag field
- Check contents of cache by finding line and checking Tag

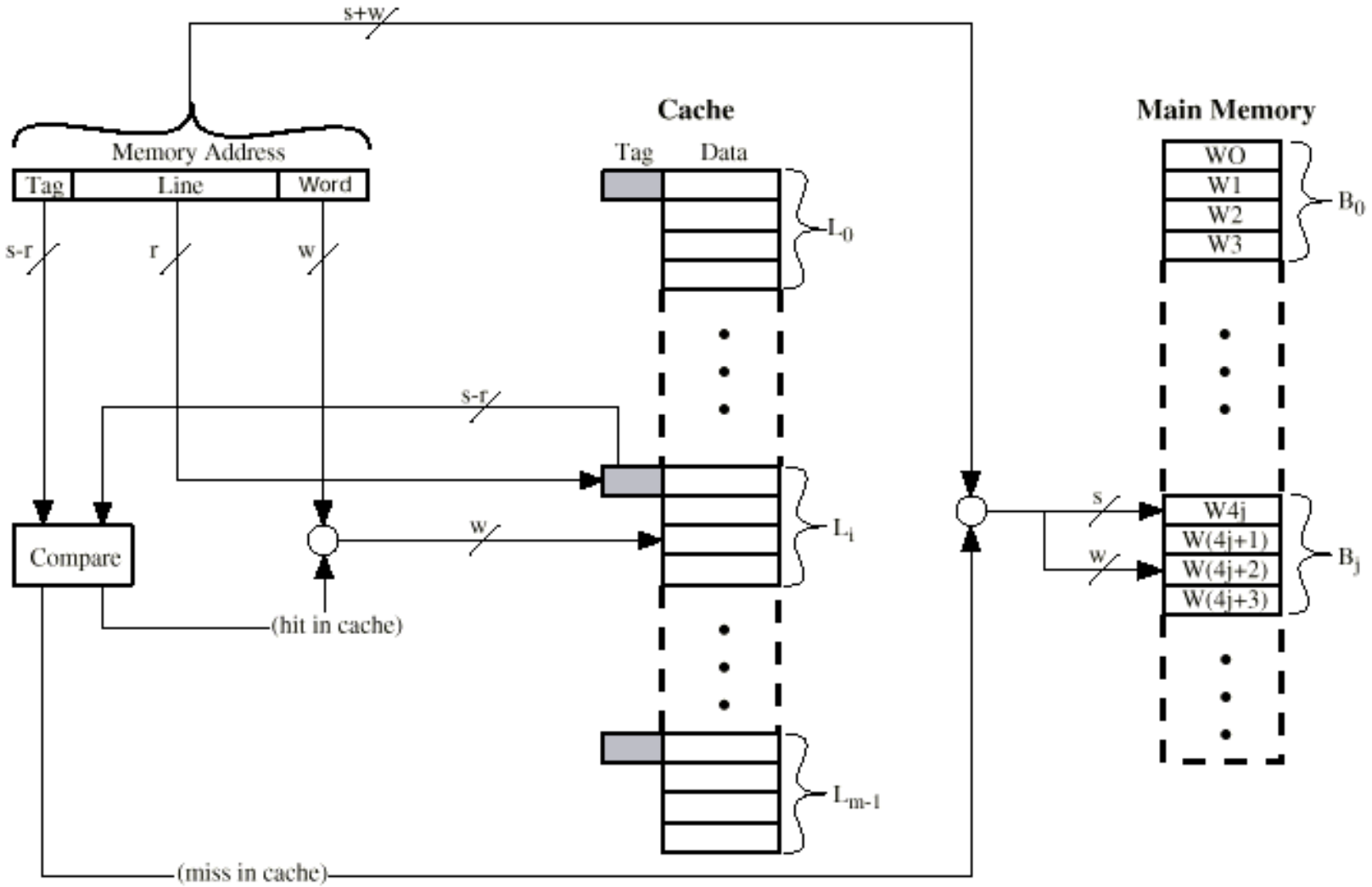
Direct Mapping

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

- Mapping function is i = j modulo m
  (i = j % m)

where i = cache line(block) number
j = main memory block number
m = number of cache line

Direct Mapping Cache Organization



Direct Mapping pros & cons

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

Direct Mapping Question:

Assume a computer has 32 bit addresses. Each block stores 16 words. A direct-mapped cache has 256 blocks. In which block (line) of the cache would we look for each of the following addresses? Addresses are given in hexadecimal for convenience.

a) 1A2BC012

b) 12345678

[Tag , Block , Word]

Answers:

- Of the 32 bit address, the last four bits denote the word on the line. Since four bits is used for one hex digit, the last digit of the address is the word on the line.
- With 256 blocks in the cache, we need 8 bits to denote the block number.
- Tag , Block , Word [20 bit , 8 bit , 4 bit]
- a) this would be block 01, which is block 1
- b) this would be 67 which is block 103 (remember, 67 is a hex value)

Associative Mapping

- A main memory block can load into any line(block) 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

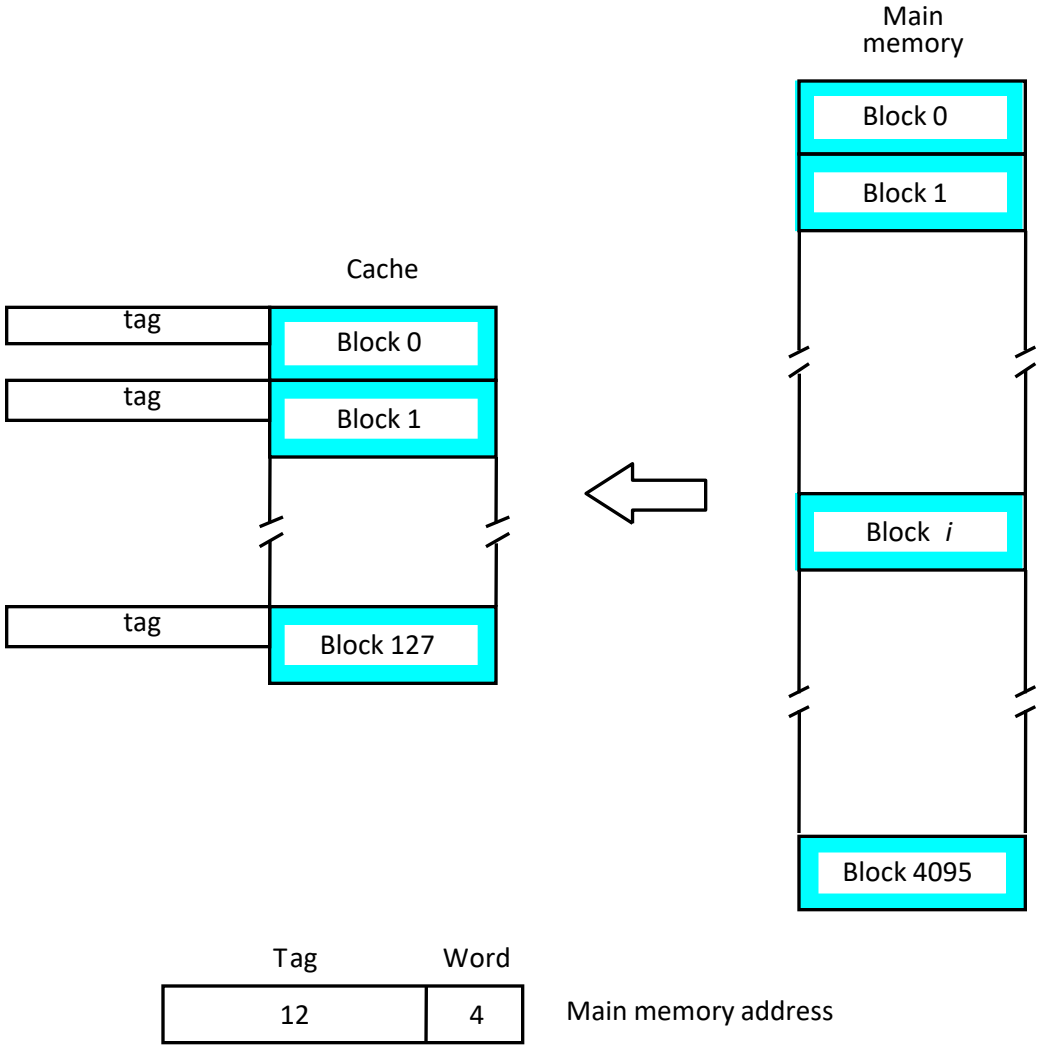


Figure 5.16. Associative-mapped cache.

Associative Mapping Address structure

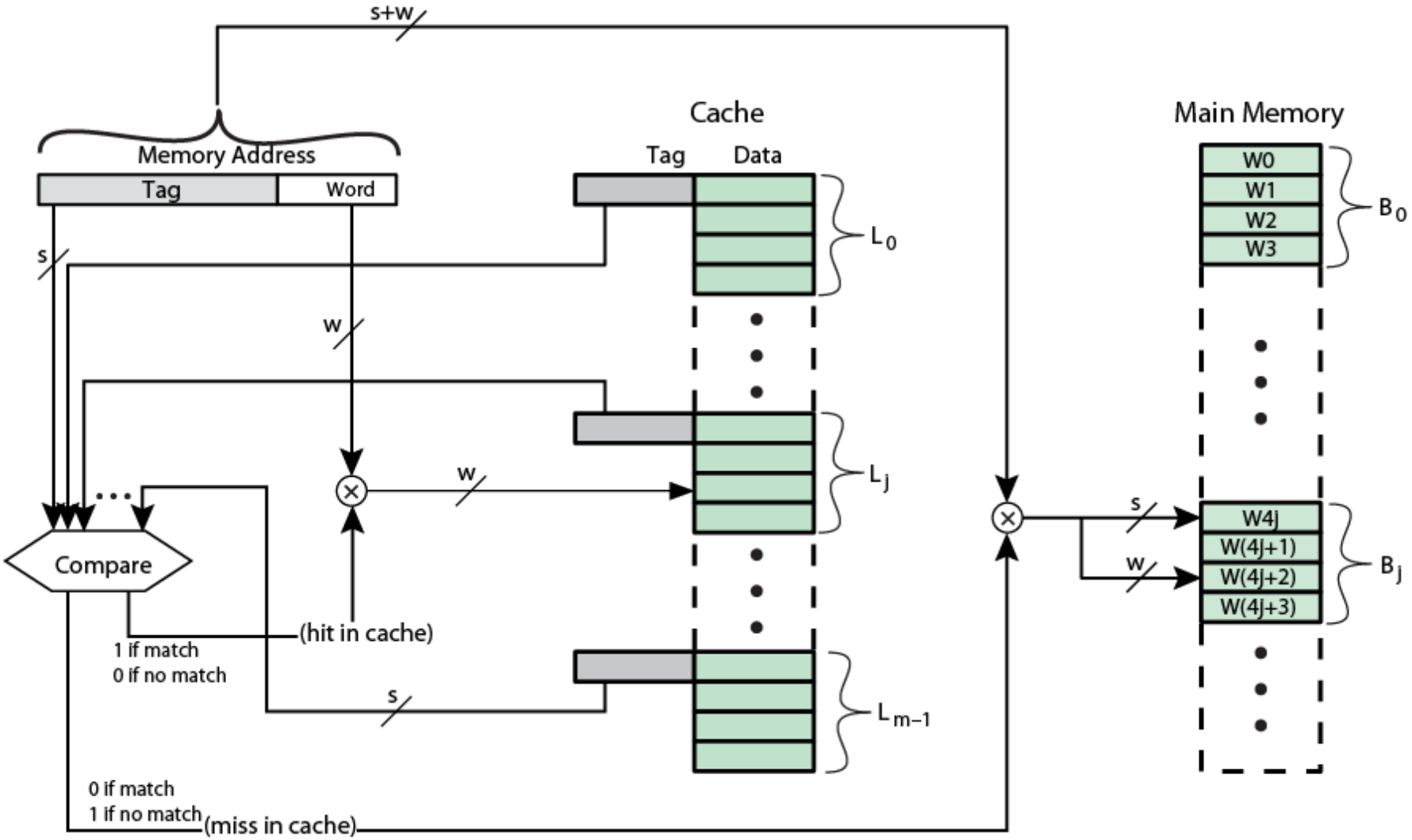
- Compare tag field with tag entry in cache to check for hit
- Least significant 4 bits of address identify which word is required from a block
- Associative search is used(parallel)

How many bits are in the tag and offset fields?
Associative Mapping
24 bit addresses
128K bytes of cache
64 byte cache lines(Block size)

- A. tag= 20, offset=4
- B. tag=19, offset=5
- C. tag=18, offset=6
- D. tag=16, offset=8

[Tag , Word]

Fully Associative Cache Organization



Associative Mapping Pros & Cons

Flexible as to which block to replace when a new block is read into the cache.

- need to select one which is not going to be used in the near future.

Complex circuitry is required to examine the tags of all cache lines

Set Associative Mapping

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

Set Associative Mapping

- A compromise of direct and associative methods.
- Cache is divided into a number of sets(v).
- Each set contains a number of lines(block)(k).
- The relationships are

$$m = v \times k$$

$$i = j \text{ modulo } v$$

where

i = cache set number

j = main memory block number

m = number of lines(block) in the cache

Set-Associative Mapp

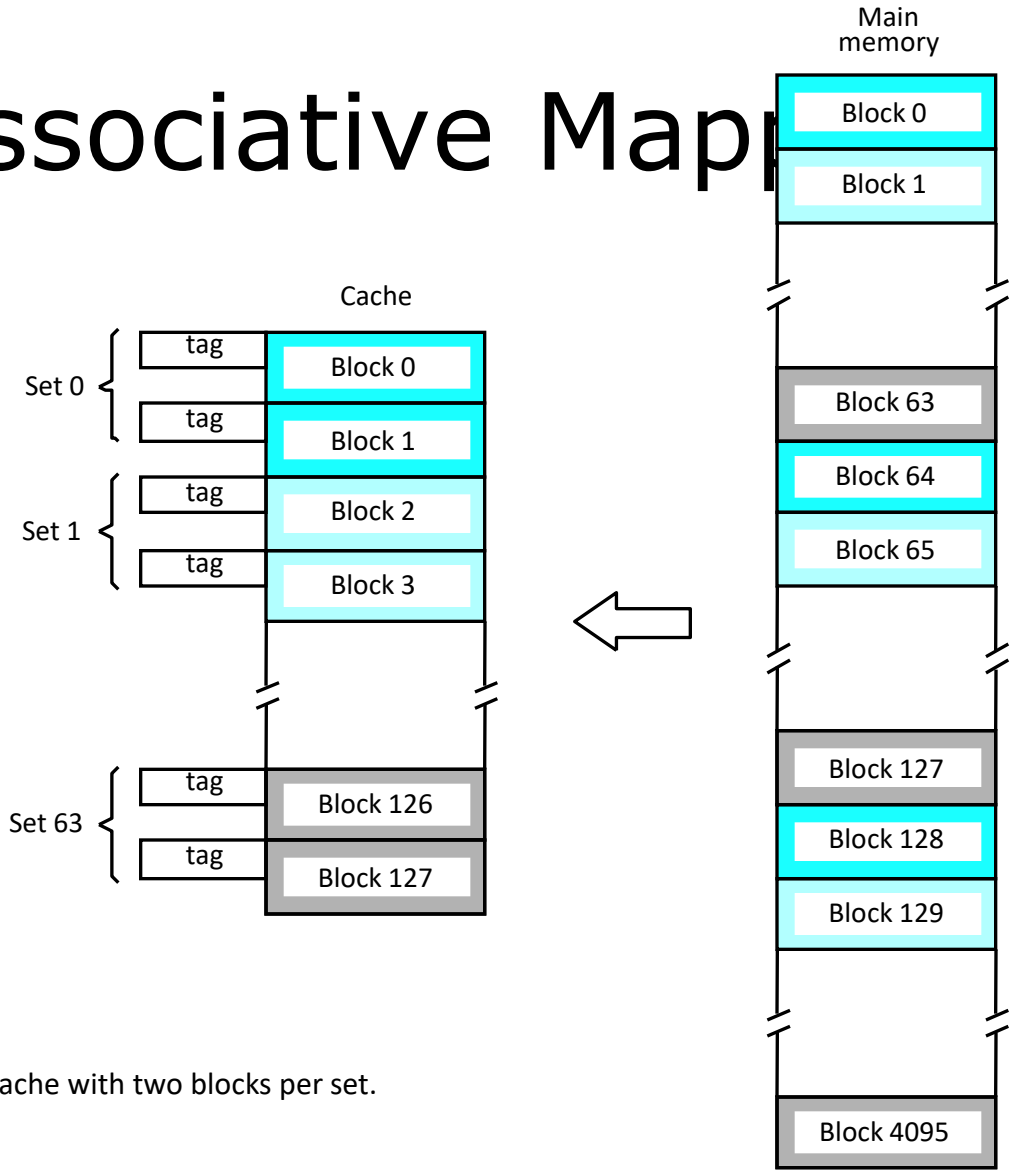
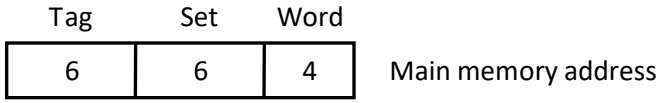
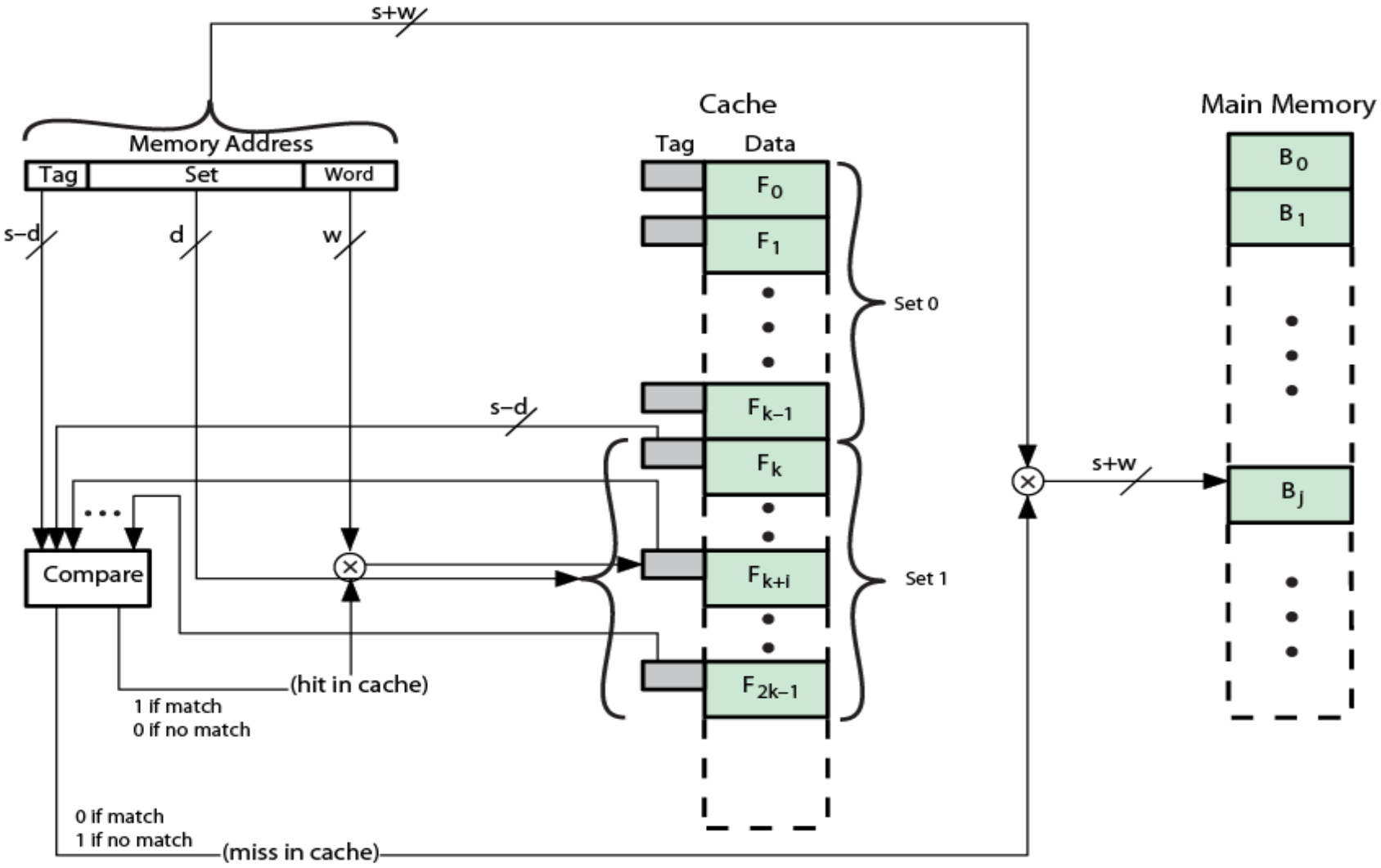


Figure 5.17. Set-associative-mapped cache with two blocks per set.



K-Way Set Associative Cache Organization



Set Associative Mapping Address

Tag 6 bit	Set 6 bit	Word 4 bit

- Use set field to determine cache set to look in

  —this determines the mapping of blocks into lines
- Compare tag field to see if we have a hit

  —two lines are examined simultaneously
- If v(set) = m, k = 1, same as direct mapping
- If v = 1, k = m, same as associative mapping
- two- or four-way set associative mappings are common

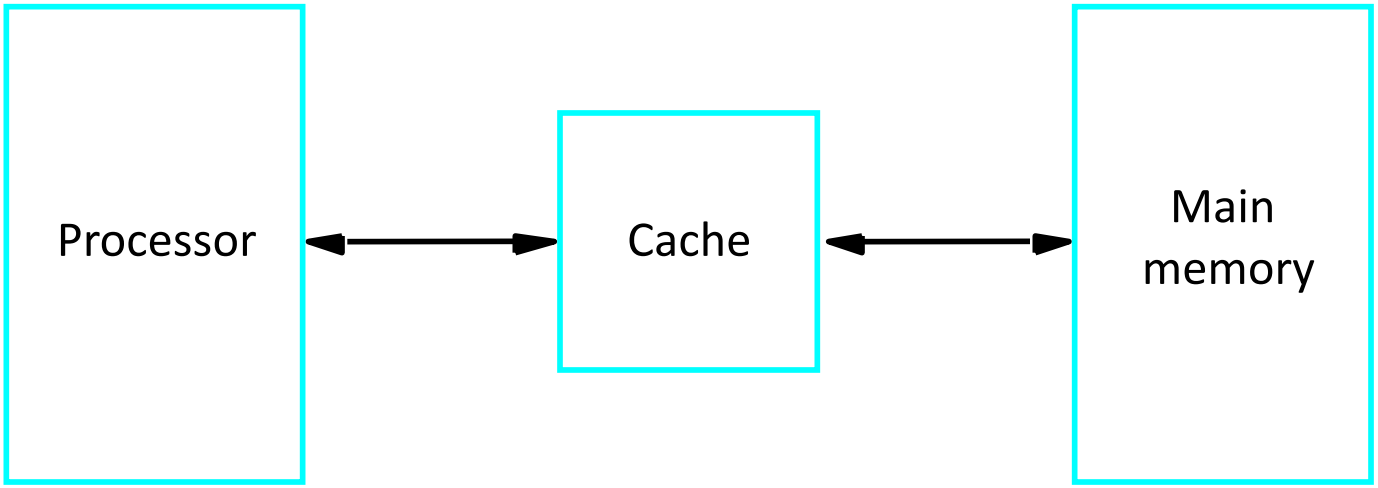
m = number of lines(block) in the cache

Each set contains a number of lines(block)(k).

- A computer system has a 1 GB main memory. It also have a 4K-Byte cache organized as 4-way set-associative, with 4 blocks per set and 64 bytes per block. Calculate the number of bits in the Tag, Set Index, and Byte Offset fields of the memory address format.
- [Tag , set , word]

- Answer: <20, 4 , 6>

Cache



Use of a cache memory.

- Replacement algorithm

Direct mapping

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

Replacement Algorithms

1. Least Recently used (LRU)
2. First in first out (FIFO)

   replace block that has been in cache longest
3. Least frequently used

   replace block which has had fewest hits
4. Random

LRU

Least **R**ecently **U**sed (LRU) algorithm is a Greedy algorithm where the block to be replaced is least recently used. The idea is based on locality of reference, the least recently used block is not likely .

The block reference string 7 0 1 2 0 3 0 4 2 3 0 3 2

Initially we have 4 block slots empty.

Initially we have 4 block slots empty.

Ans: No of fault is 6.

Write Policy

- When a block in the cache is to be replaced, need to consider whether it has been altered.

  If not, that block may be overwritten.

  If so, main memory need to be updated.

Two problems to contend:

More than one device may have access to main memory

Multiple processors with their own caches.

Two techniques:

1. Write through

2. Write back

Write through

- The cache location and the main memory are updated simultaneously.
- Multiple CPUs can monitor main memory traffic to keep local (to CPU) cache up to date
- Lots of traffic
- Slows down writes

Write back or copy back

- Updates are made only in cache

  Update bit (dirty bit) for cache line is set

  If a block is to be replaced, write to main memory only if update bit is set.

- Portions of main memory may be invalid

  accesses by other devices are allowed only through the cache

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…

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

A Real-World Example