Cache Memory

So you want fast?

- It is possible to build a computer which uses only static RAM
- This would be very fast
- This would cost a very large amount

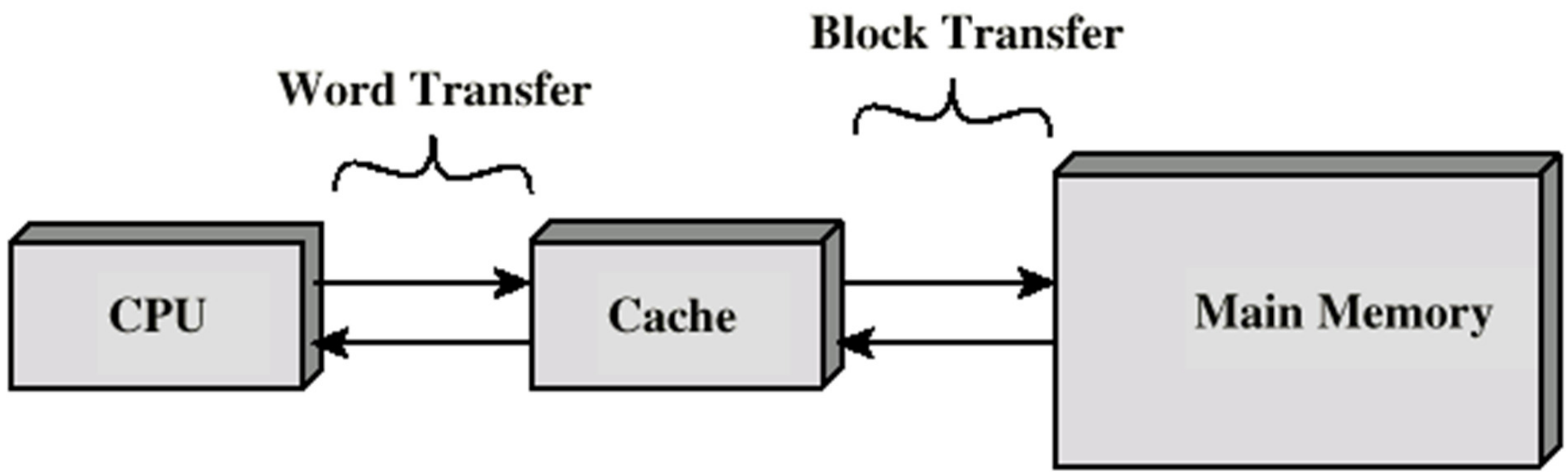
- Alternatives??

Locality of Reference

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

Cache

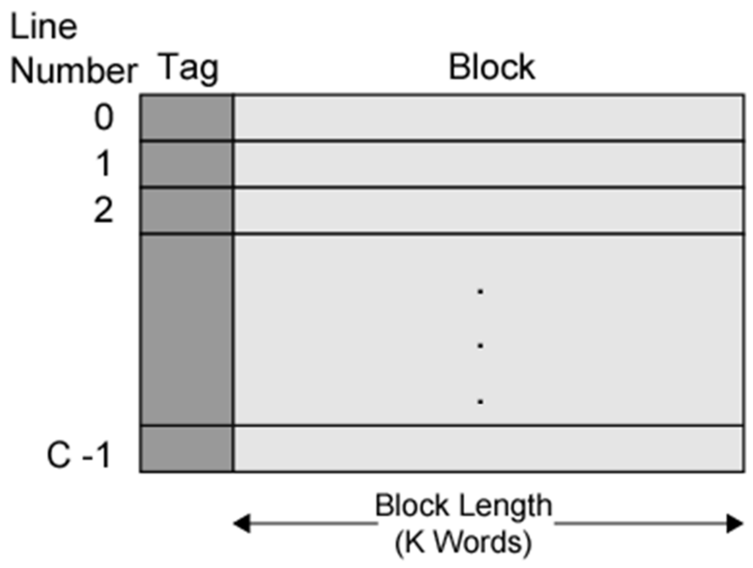
- Small amount of fast memory
- Sits between normal main memory and CPU
- May be located on CPU chip or module



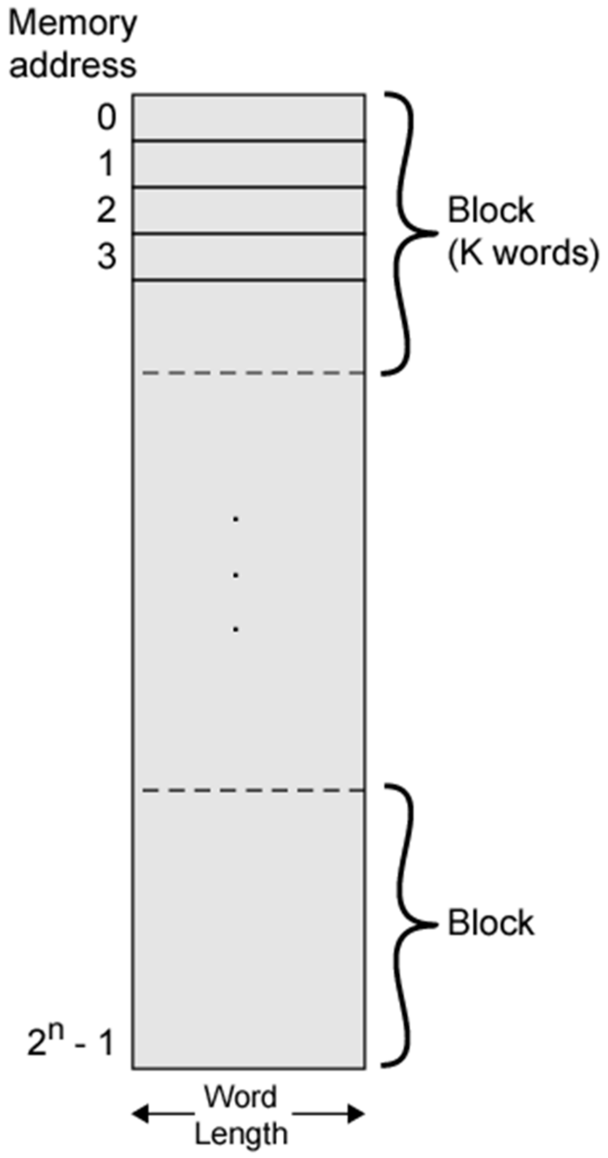
Hierarchy List

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

Cache/Main Memory Structure



(a) Cache

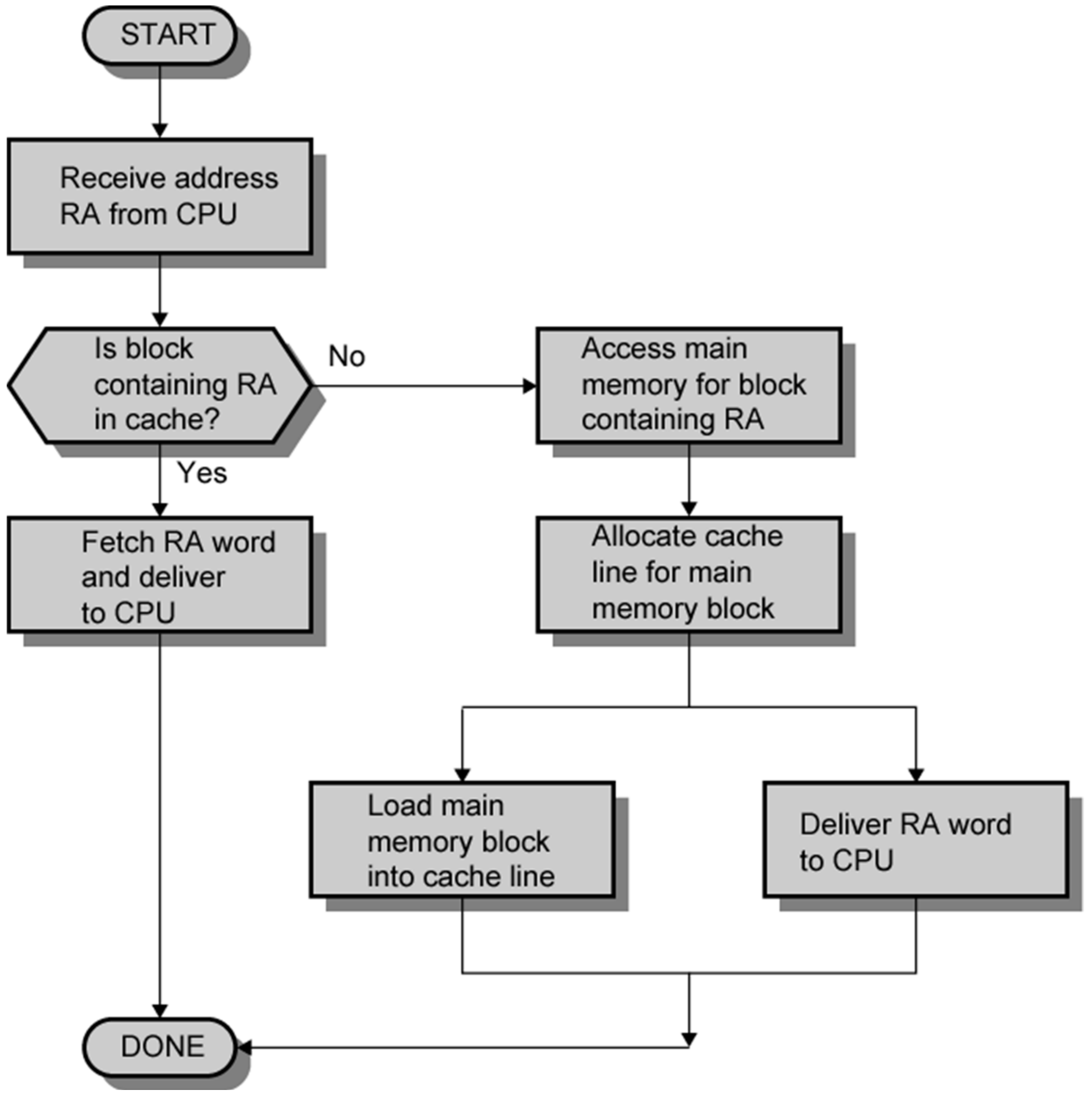


(b) Main memory

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

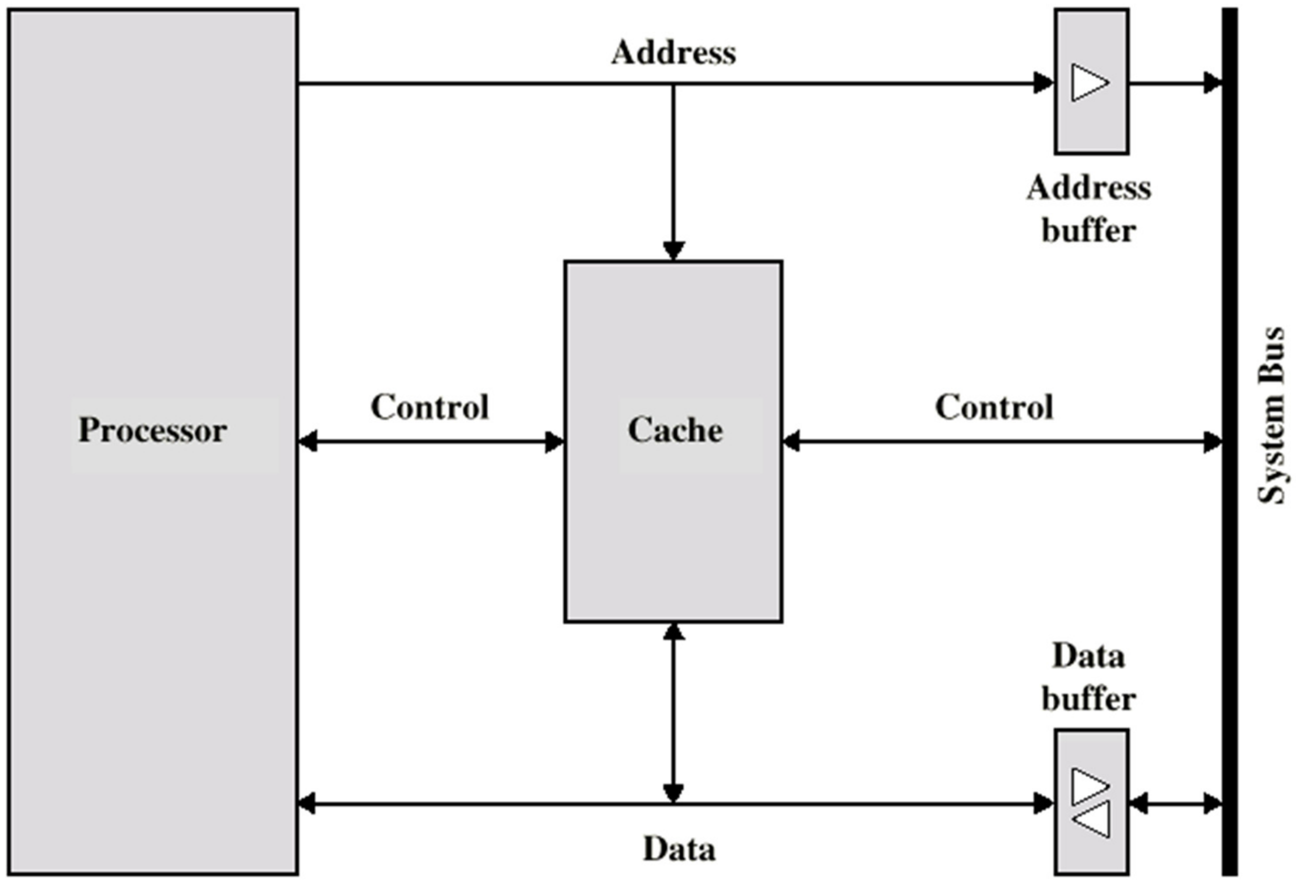
Cache Design

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

Size does matter

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

Typical Cache Organization



Write Policy

- Must not overwrite a cache block unless main memory is up to date
- Multiple CPUs may have individual caches
- I/O 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
- Lots of traffic
- Slows down writes

- Remember bogus write through caches!

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
- I/O must access main memory through cache
- N.B. 15% of memory references are writes

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)

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

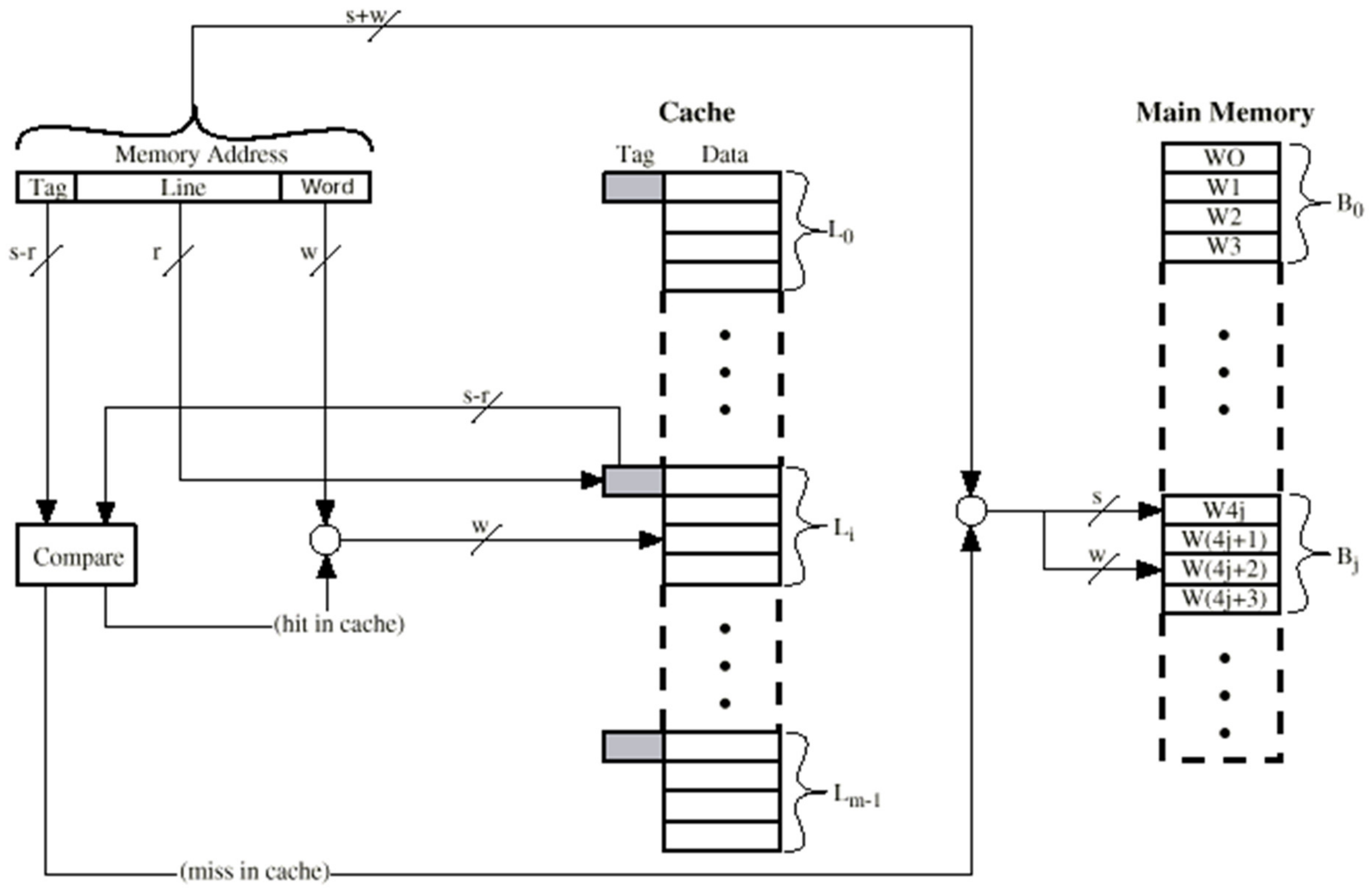
Tag s-r	Line or Slot r	Word w
8	12	4

- 24 bit address
- 4 bit word identifier (16 byte block)
- 20 bit block identifier
  - 8 bit tag (=20-12)
  - 12 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 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

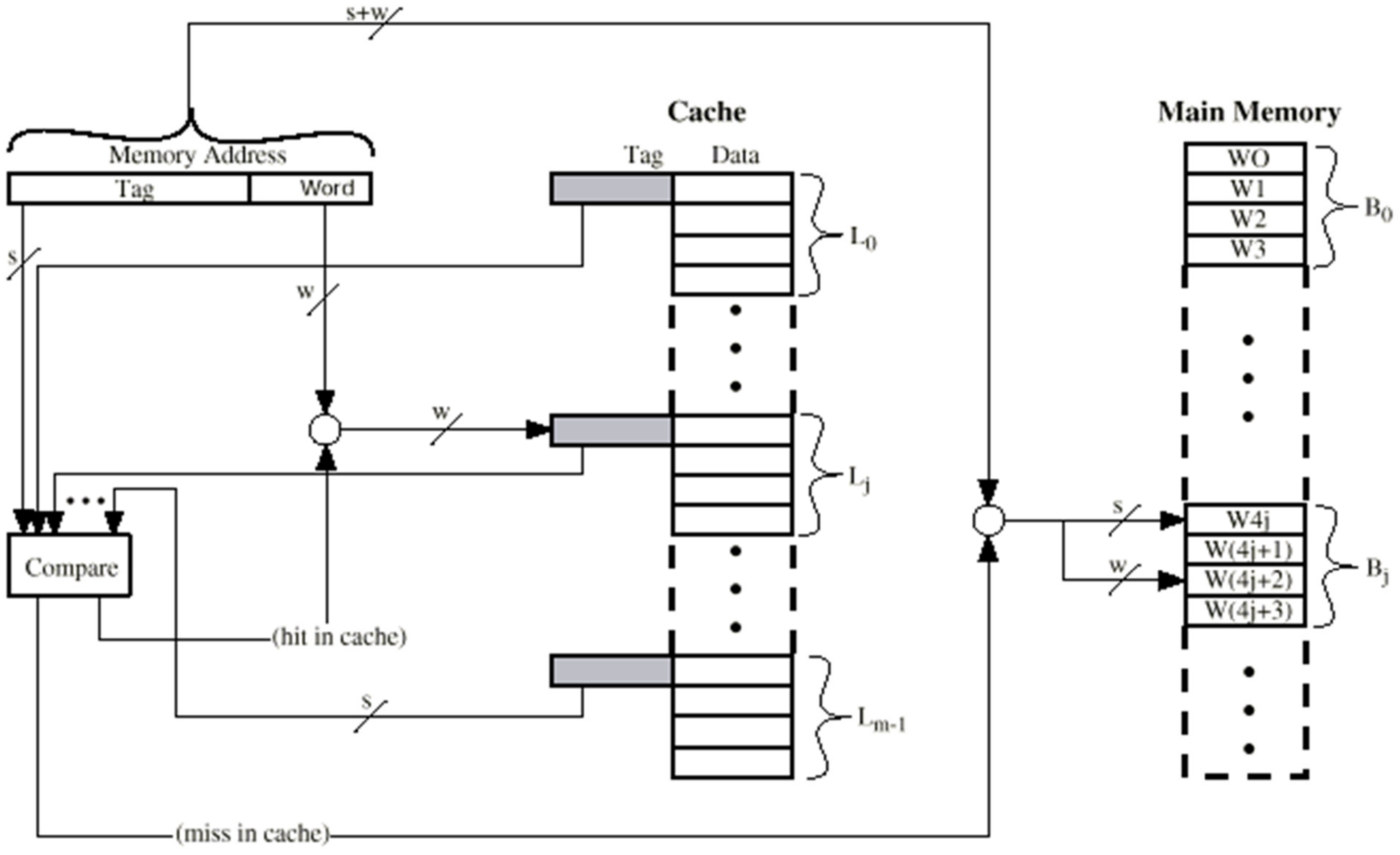
Direct Mapping pros & cons

- 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

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
Address Structure

Tag 20 bit	Word 4 bit

- 20 bit tag stored with each 16 byte block of data
- Compare tag field with tag entry in cache to check for hit
- Least significant 4 bits of address identify which byte is required from 16 byte data
- e.g.

Address	Tag	Data	Cache line
FFFFFC	FFFFF C	24	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 = cache size/2^w
- Size of tag = s bits

Set Associative Mapping

- Cache is divided into a number of sets
- Each set contains a number of lines
- 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

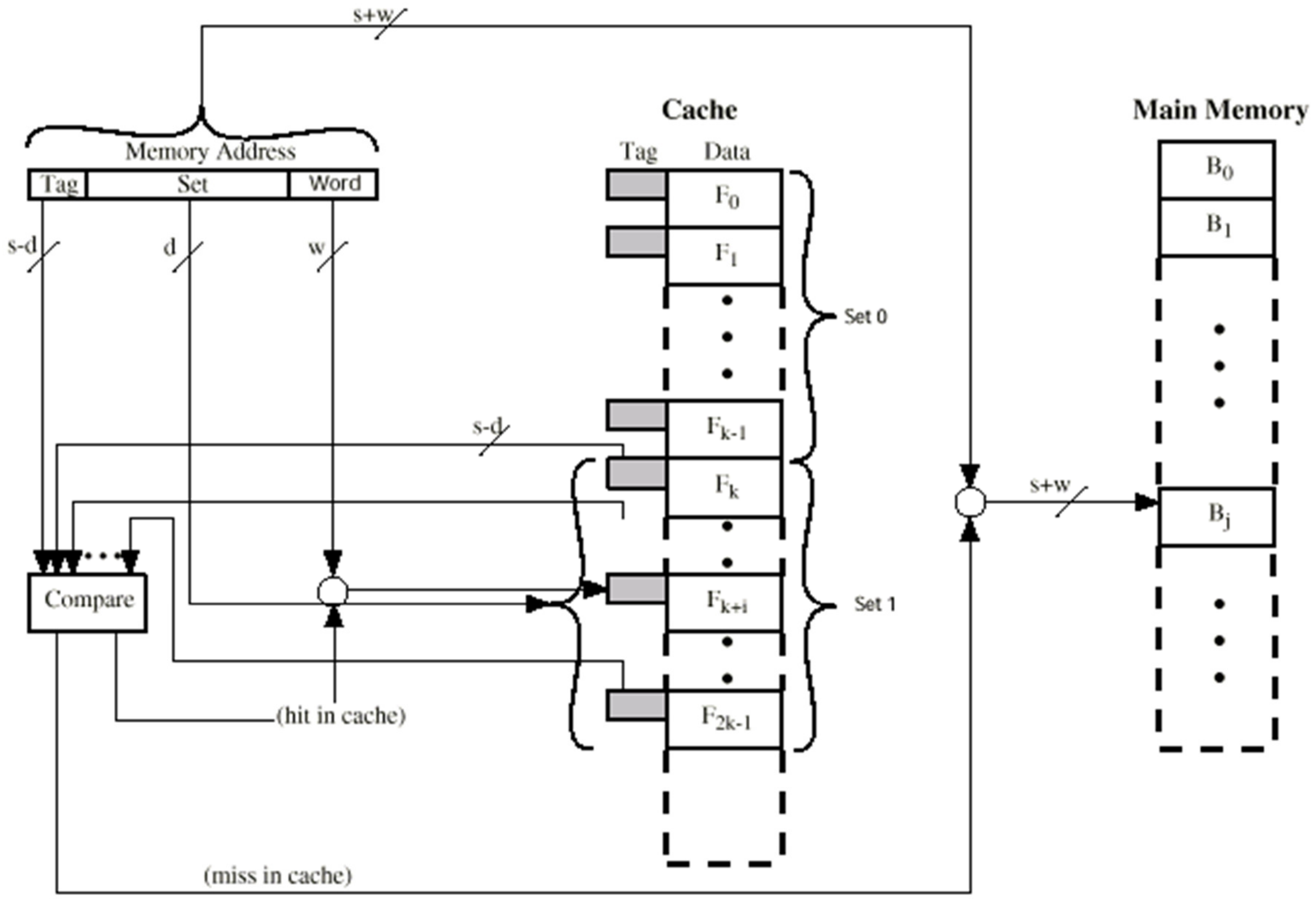
Set Associative Mapping

- The cache is divided in *v* sets
- Each set consists of *k* lines
- Number of lines in the cache
  - $m = v \times k$
- The mapping function:
  - $i = j$ modulo v
- Where
  - i = cache set number
  - j = main memory block number

K way set associative Mapping Cache Line Table

Set no	Main Memory blocks held
0	0, v, 2v, 3v,…,2^s-v
1	1,v+1, 2v+1,…,2^s-v+1
v-1	v-1, 2v-1,3v-1,…,2^s-1

K Way Set Associative Cache Organization



Set Associative Mapping Address Structure

Tag 8 bit	Set 12 bit	Word 4 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
1F 17F B	1F	11	17E
20 17E C	20	12	17E

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

Mapping: Example

- A block set associative cache consists of a total of 64 lines divided into 4-line sets. The main memory contains 4096 blocks, each consisting of 128 words.
  - What is the size of main memory and cache memory
  - How many bits are there in a main memory address
  - How many bits are there in each of the TAG, SET and WORD fields.

Mapping: Example

- A cache consists of a total of 64 lines. The main memory contains 4096 blocks, each consisting of 128 words.
  - Give the main memory address format in terms of TAG, SET/LINE and WORD format, if
    - direct mapping is used
    - Fully associative mapping is used.

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)
- e.g. in 2 way set associative
  - Which of the 2 block is lru?
- First in first out (FIFO)
  - replace block that has been in cache longest
- Least frequently used
  - replace block which has had fewest hits
- Random

Replacement Algorithms (3)
Least Recently Used (LRU)

- Program usually stays in localized area for a reasonable period of time.
- There is a high probability that the blocks that have been referenced recently will be referenced again soon.
- When a block is to be overwritten, it is sensible to overwrite the one that has gone the longest time without being referenced.
- This block is called the **Least Recently Used (LRU) block** and the technique is called the LRU replacement policy.

Replacement Algorithms (4)
Least Recently Used (LRU)

- Consider four-line set in a set-associative cache
- Control bits:
  - TAG bits
  - 2-bit counter for each line(to track the LRU block)
  - d_bit: dirty bit
  - f_bit: occupied bit
- Initially reset all the counters, d_bit and f_bit

Replacement Algorithms (5)
Least Recently Used (LRU)

- A cache hit occurs:
  - set the counter value to 0 of this cache line
  - for other counters, if the value is less than the referenced line, increment the counter value
  - otherwise, do not change the counter value
- A cache miss occurs:
  - Set is not full
  - Set is full

Replacement Algorithms (6)
Least Recently Used (LRU)

- Set is not full:
  - set the counter value to 0 of the cache line
  - set F_bit to 1
  - increment the counter value of other lines whose f_bit is 1
- Set is full:
  - the line with highest counter value is removed; write back if d_bit is 1
  - new block is transferred to this line
  - reset d_bit and the counter
  - other counter values are incremented by 1

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