Cache Memory – Part II

Dr. B. R. Bhowmik
Dept. of CSE
NIT Karnataka

Elements of Cache Design

- Addressing: Logical and Physical
- Size
- Mapping Function: Direct, Associative, and Set Associative
- Replacement Algorithm: Least recently used (LRU), First in first out (FIFO), Least frequently used (LFU), and Random
- Write Policy: Write through, Write back, and Write once
- Block and Line Size
- Number of Caches: Single or two level, and Unified or split

Cache Addressing

A. Where does cache sit?
   - Between processor and virtual memory management unit
   - Between MMU and main memory

B. Logical cache (virtual cache) stores data using virtual addresses
   - Processor accesses cache directly, not thorough physical cache
   - Cache access faster, before MMU address translation
   - Virtual addresses use same address space for different applications
     - Must flush cache on each context switch

C. Physical cache stores data using main memory physical addresses

Cache Size

- The size of the cache must be small enough so that the overall average cost per bit is close to that of main memory alone and large enough so that the overall average access time is close to that of the cache alone.
- Cache size does matter for
  a) Cost : More cache is expensive
  b) Speed
     i. More cache is faster (up to a point)
     ii. Checking cache for data takes time

Mapping Function

- Cache of 64 kByte
- 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)

N.B.: Cache mapping already discussed. See Cache Memory – Part I. Additionally, it is provided here.

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
- Figure 9 shows direct mapping from main memory to cache.

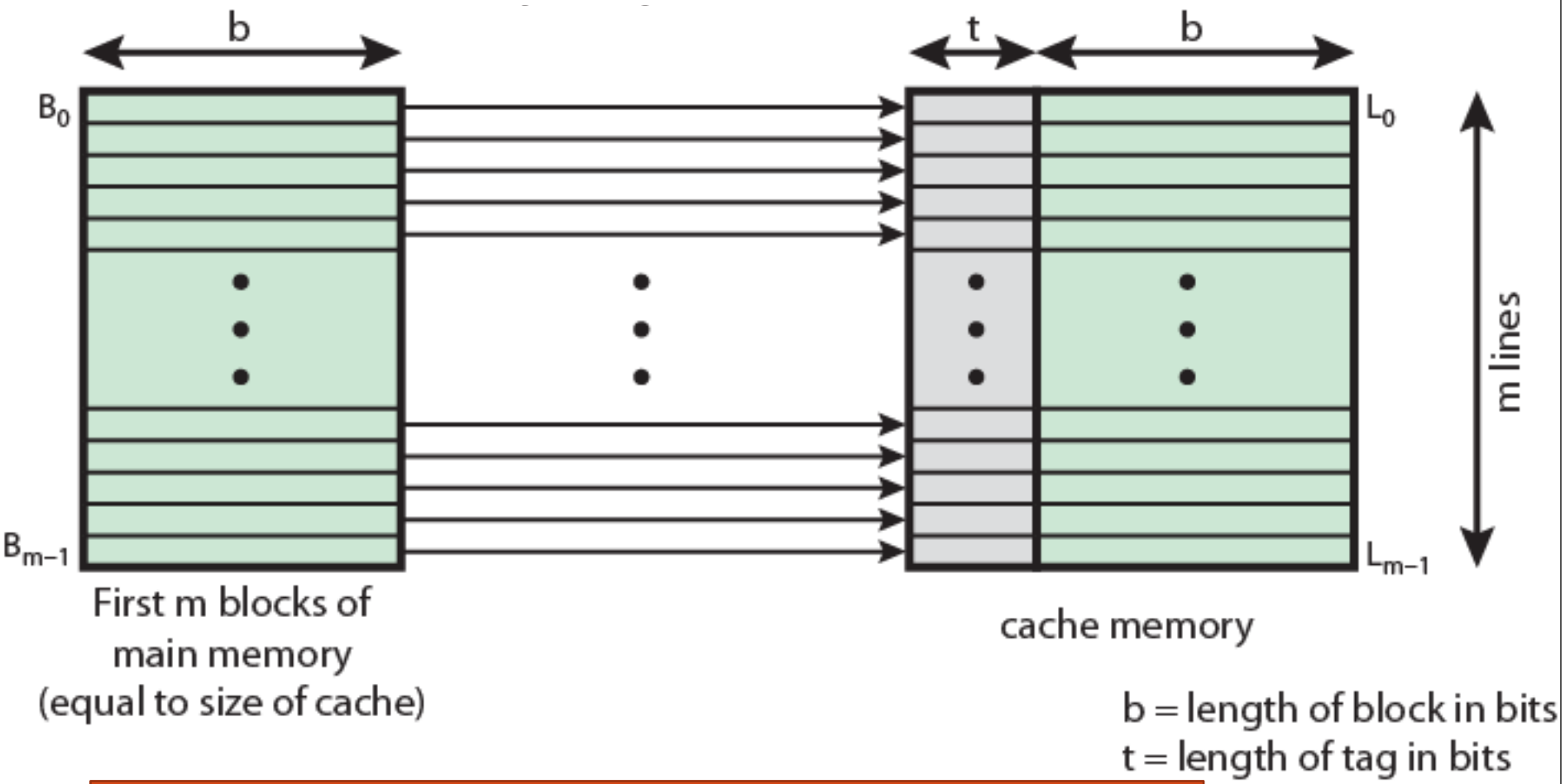

Figure 9: Direct mapping from main memory to cache.

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

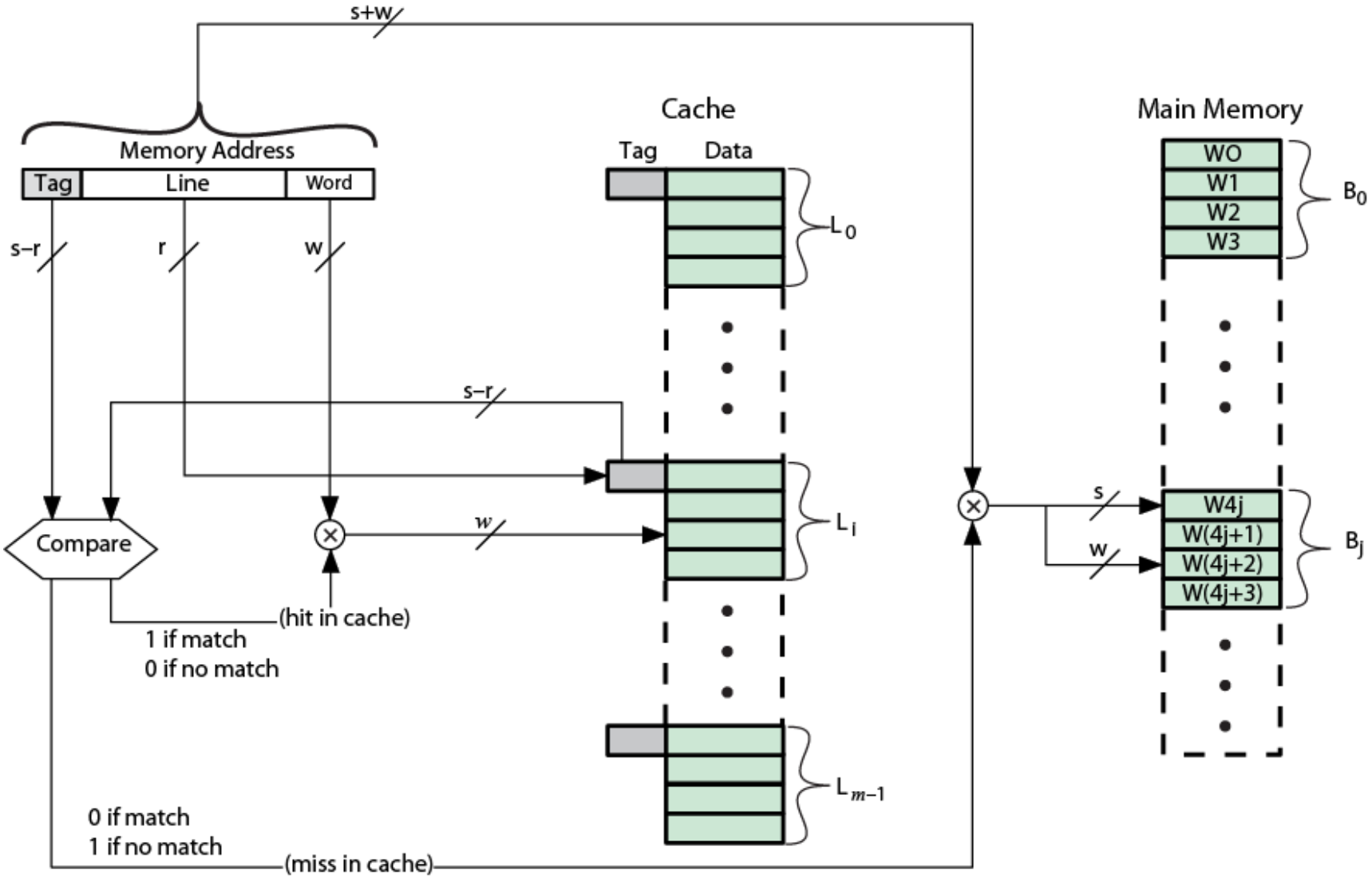

Figure 10: Direct Mapping Cache Organization

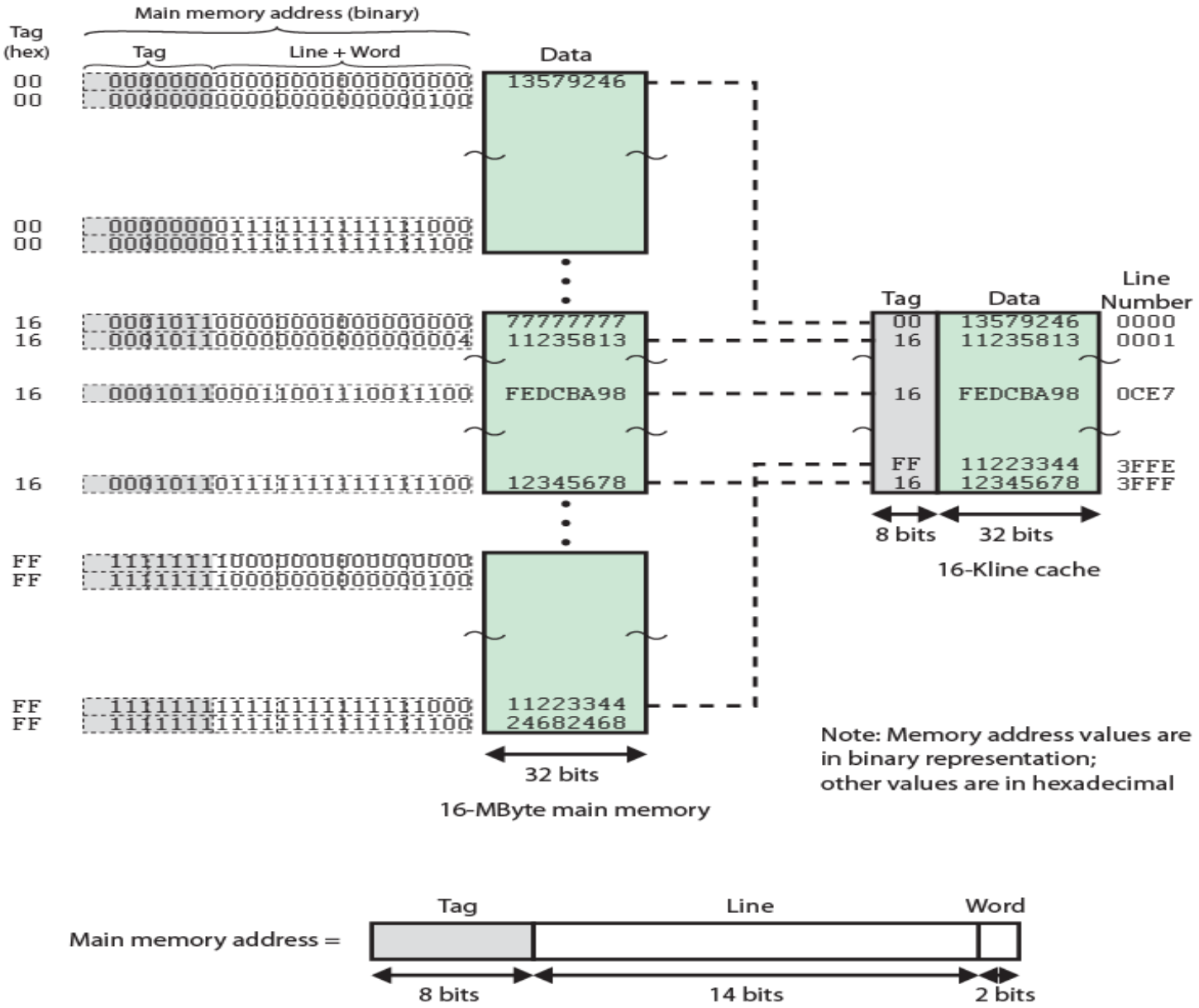

Figure 11: Direct Mapping Example

Direct Mapping Summary

- Address length = (s + w) bits
- Number of addressable units = 2s+w words or bytes
- Block size = line size = 2w words or bytes
- Number of blocks in main memory = 2s+ w/2w = 2s
- Number of lines in cache = m = 2r
- Size of tag = (s – r) bits

Direct Mapping pros & cons

- The direct mapping technique is simple and inexpensive to implement.
- Its main disadvantage is that there is a fixed cache location for any given block. Thus, if a program happens to reference words repeatedly from two different blocks that map into the same line, then the blocks will be continually swapped in the cache, and the hit ratio will be low (a phenomenon known as thrashing).

Victim Cache

- Victim cache is proposed as an approach to reduce the conflict misses of direct mapped caches without affecting its fast access time.
- Advantages: Victim cache
  - Lowers miss penalty
  - Remembers what was discarded
    - Already fetched
    - Use again with little penalty
  - Is Fully associative
  - Has size 4 to 16 cache lines
  - Resides between direct mapped L1 cache and next memory level.

Associative Mapping

- Associative mapping overcomes the disadvantage of direct
- 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.

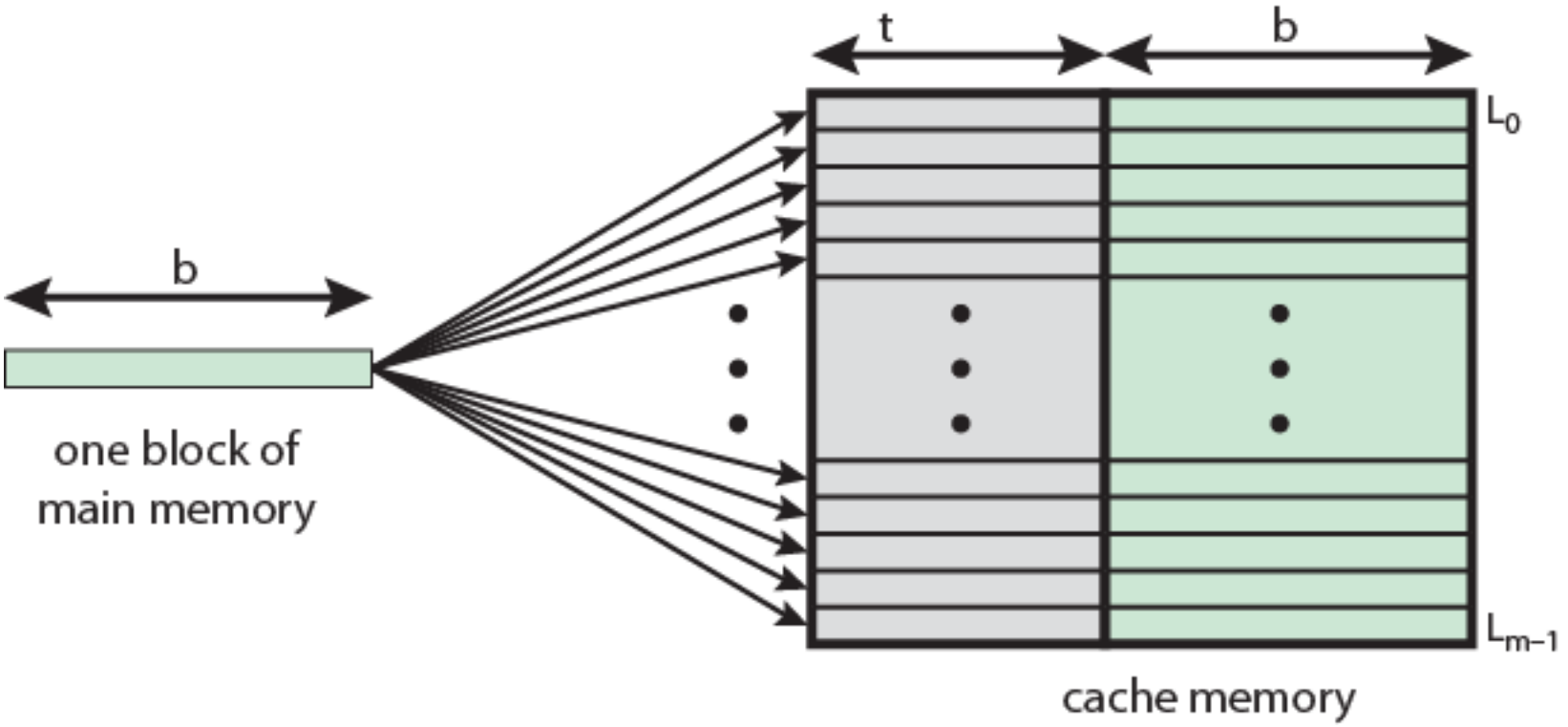

Figure 12: Associative Mapping from Cache to Main Memory

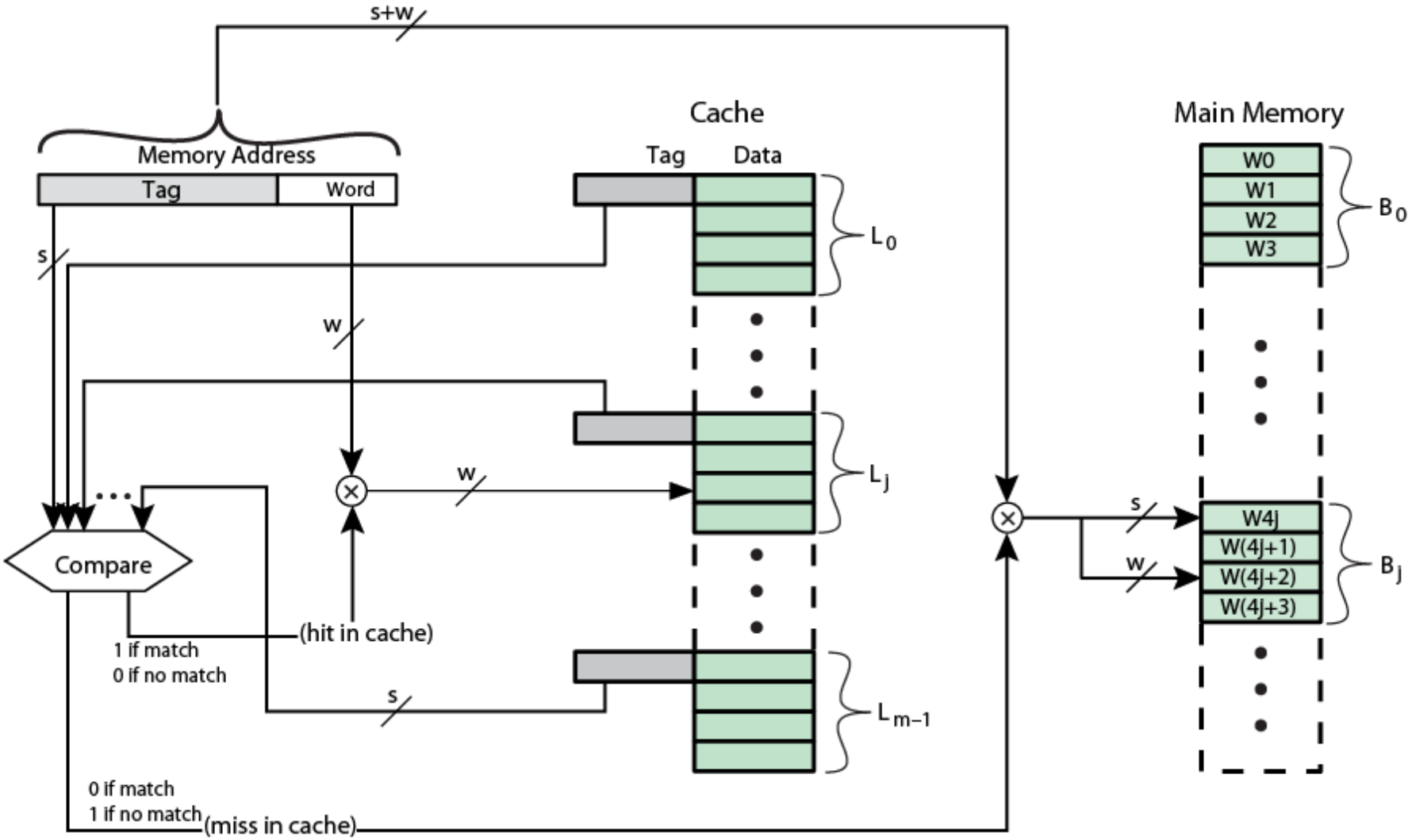

Figure 13: Fully Associative Cache Organization

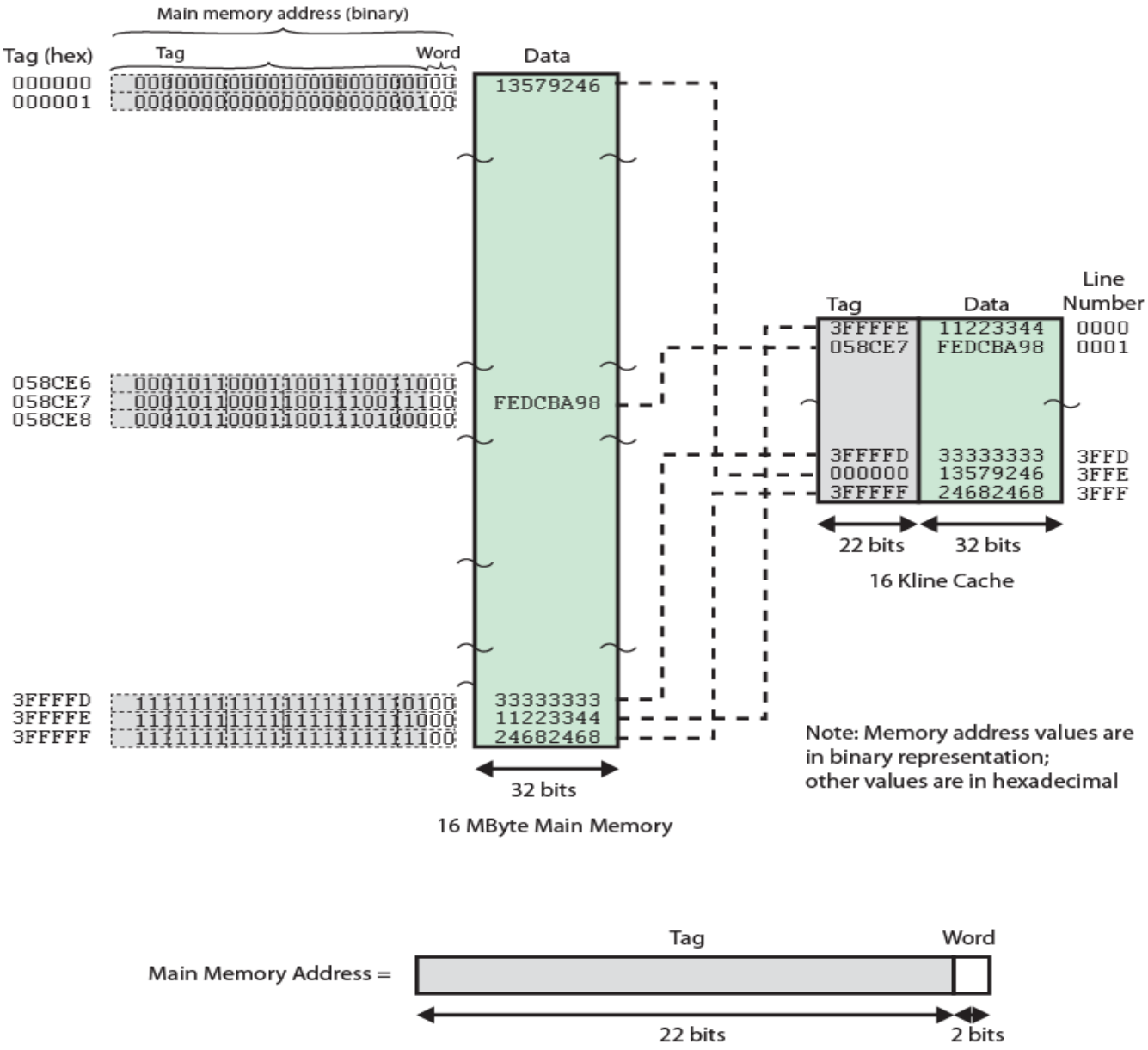

Figure 14: Associative Mapping Example

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 16 bit word is required from 32 bit data block
- Example

Address	Tag	Data	Cache line
FFFFFC	FFFFFC	24682468	3FFF

Associative Mapping Summary

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

Set Associative Mapping

- Set-associative mapping is a compromise that exhibits the strengths of both the direct and associative approaches while reducing their disadvantages.
- 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 Example

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

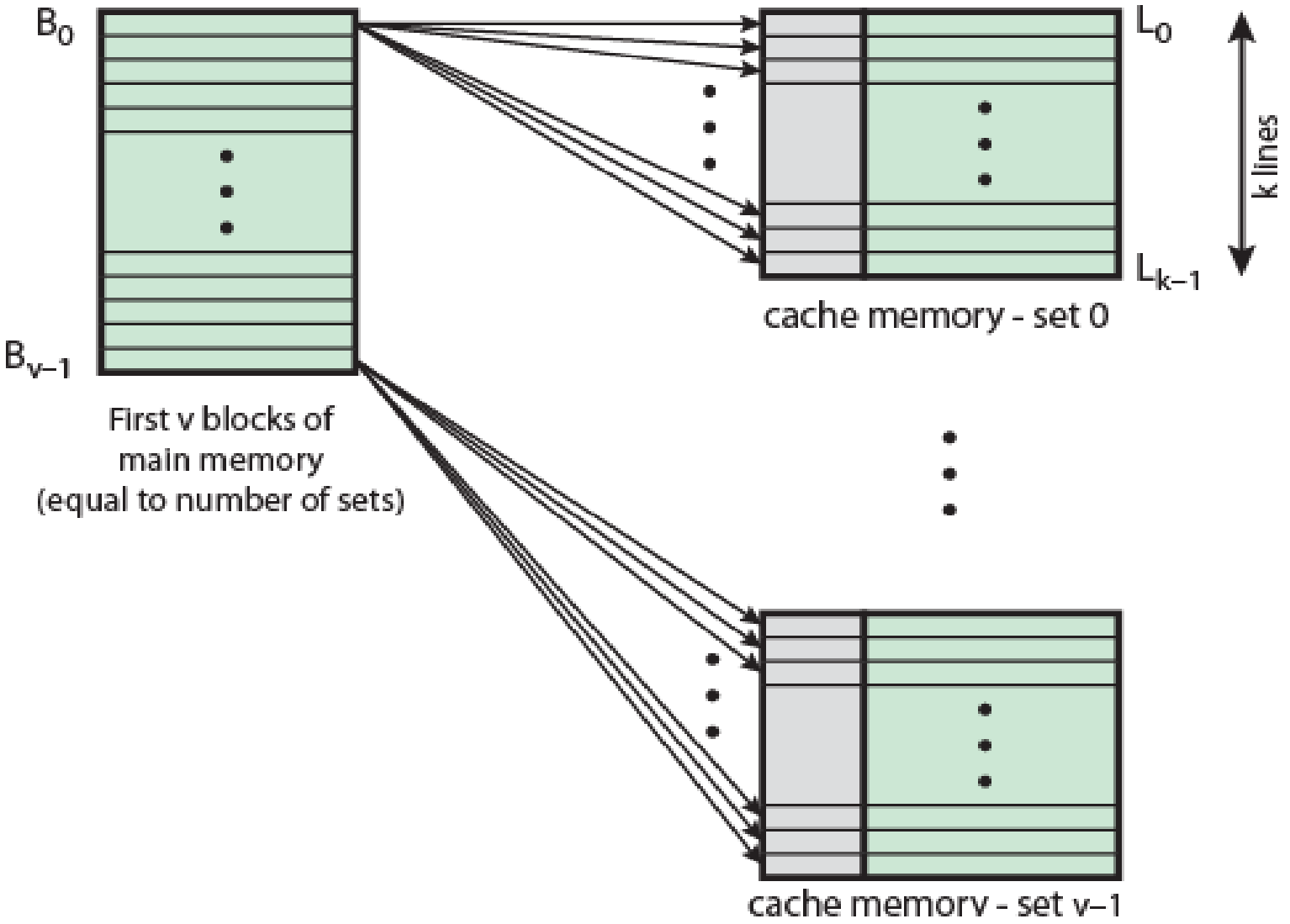

Figure 15: Mapping From Main Memory to Cache: v Associative

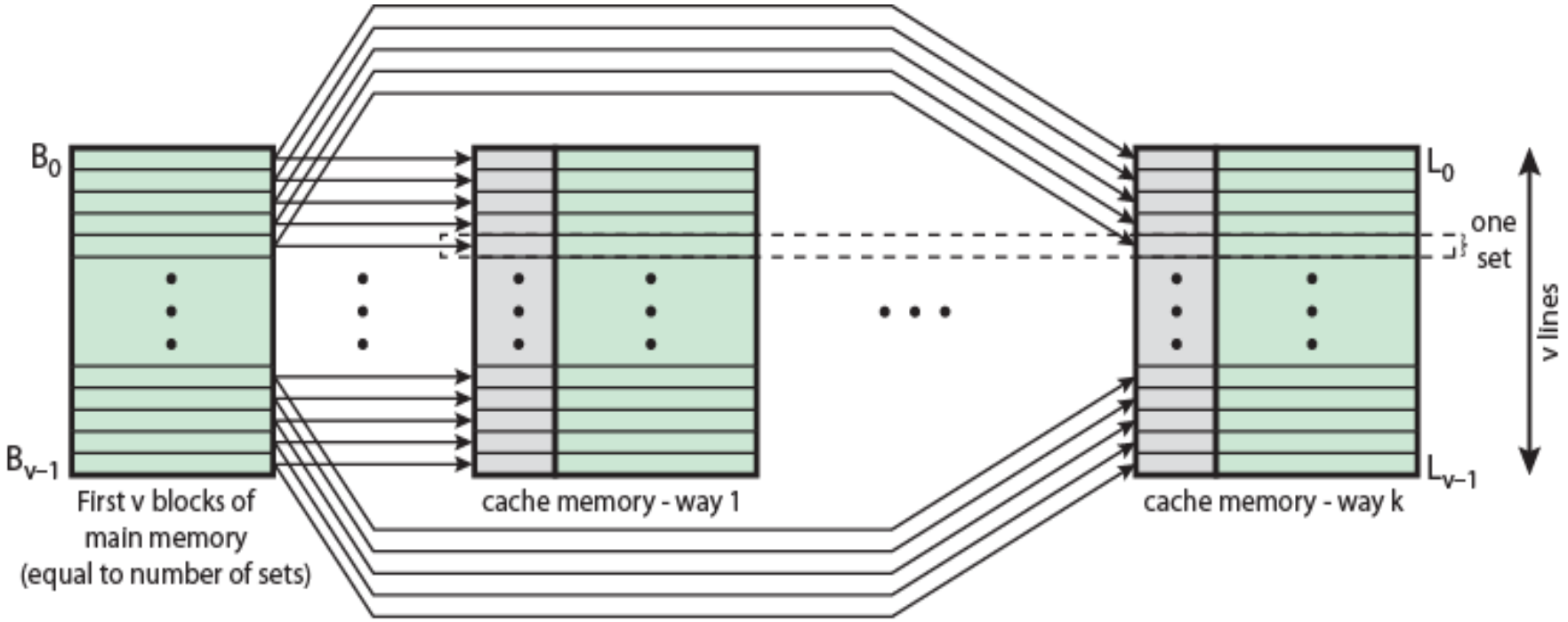

Figure 16: Mapping From Main Memory to Cache: k-way Associative

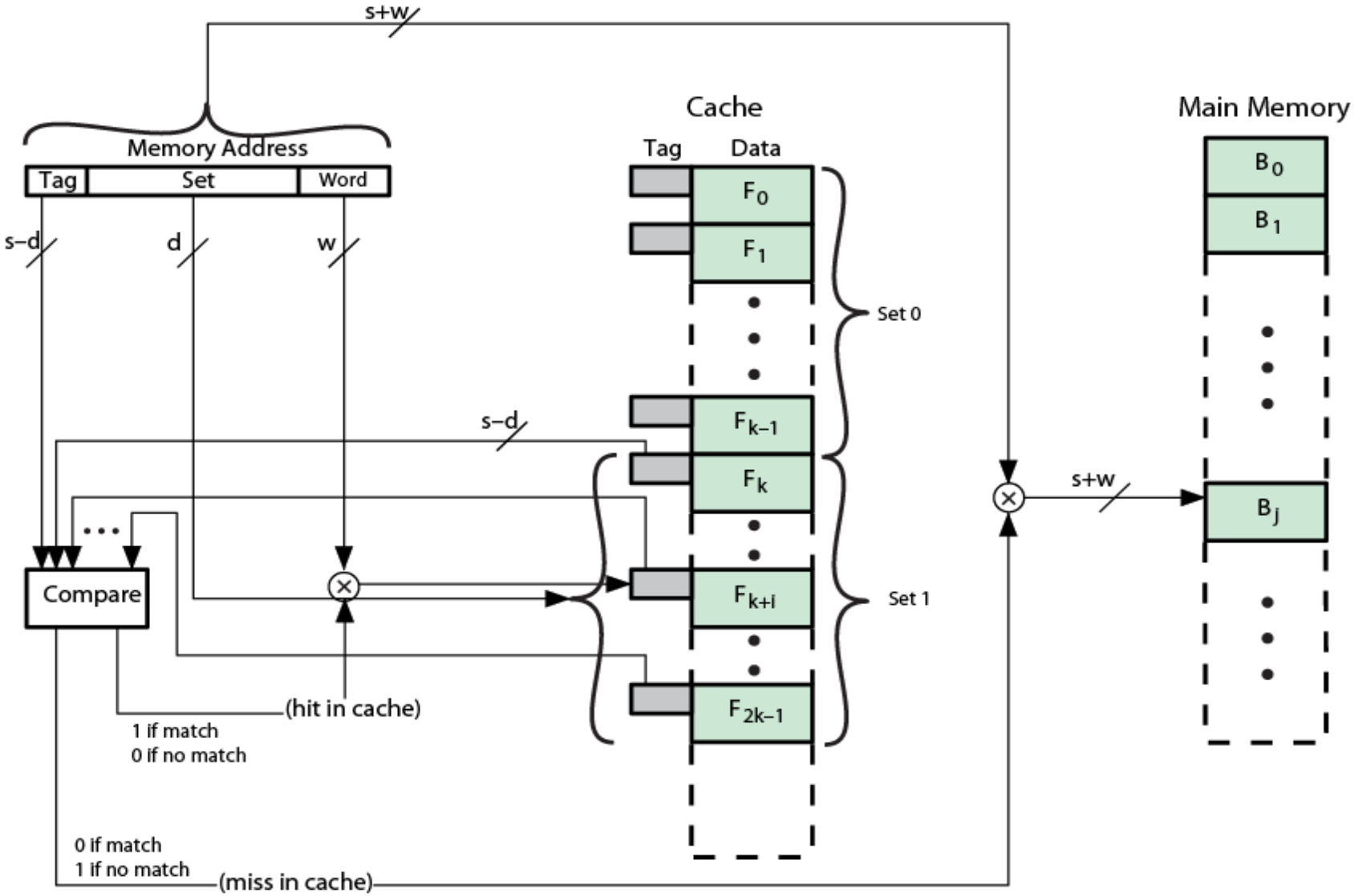

Figure 17: K-Way Set Associative Cache Organization

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
- Example

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

Set Associative Mapping Summary

- Address length = (s + w) bits
- Number of addressable units = 2s+w words or bytes
- Block size = line size = 2w words or bytes
- Number of blocks in main memory = 2d
- Number of lines in set = k
- Number of sets = v = 2d
- Number of lines in cache = kv = k * 2d
- 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

Replacement Algorithms: Direct mapping

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

Replacement Algorithms: 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

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
- 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

Number of Caches : Multilevel Caches

A. High logic density enables caches on chip
   - Faster than bus access
   - Frees bus for other transfers

B. 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 vs. 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: Example

- 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

Thank You