Lecture-8

Chapter - 4

Computer Organization and Architecture Designing - William Stallings

Cache Memory

Memory Hierarchy

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

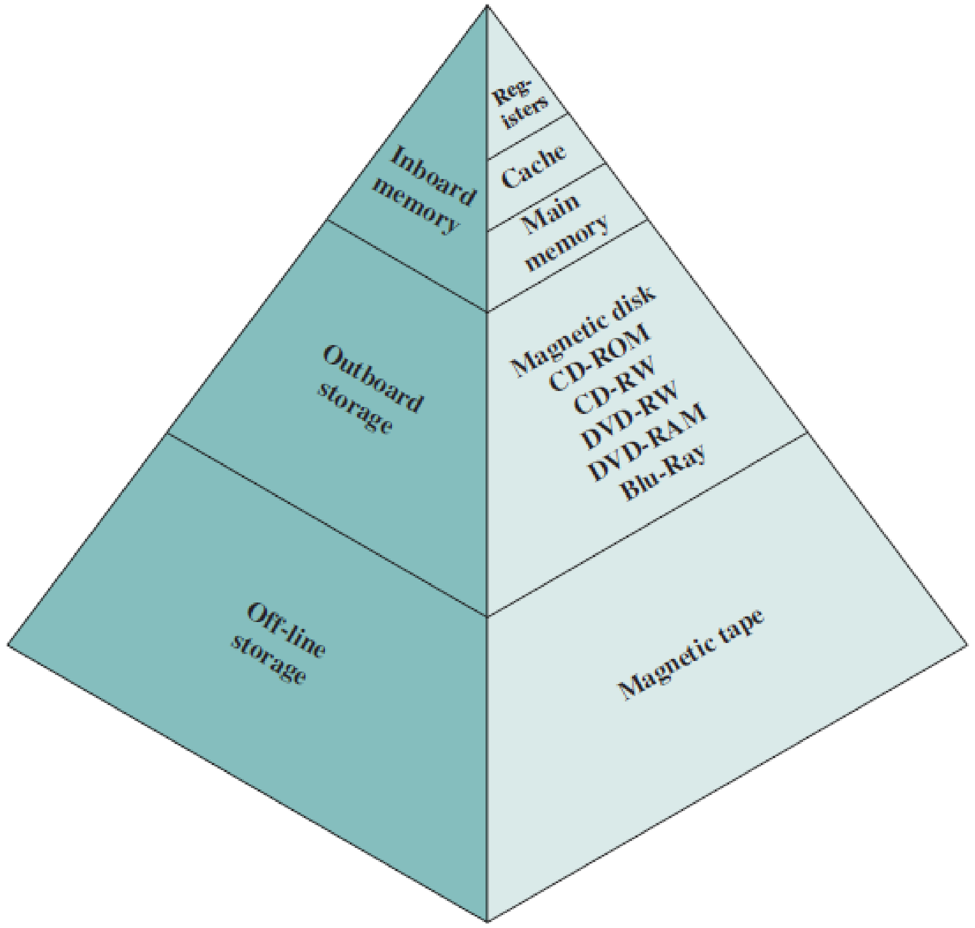


Fig: Memory Hierarchy Diagram

Performance

The two most important characteristics of memory are **capacity** and **performance**. Three performance parameters are used:

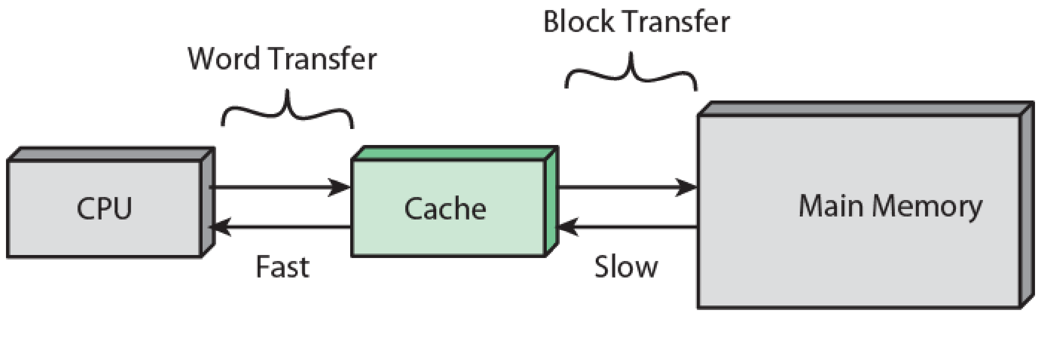
- **Access time (latency)**
  - Time between presenting the address and getting the valid data
- **Memory Cycle time**
  - Time may be required for the memory to “recover” before next access
  - Cycle time is access + recovery
- **Transfer Rate**
  - Rate at which data can be moved or transfered

Hierarchy List

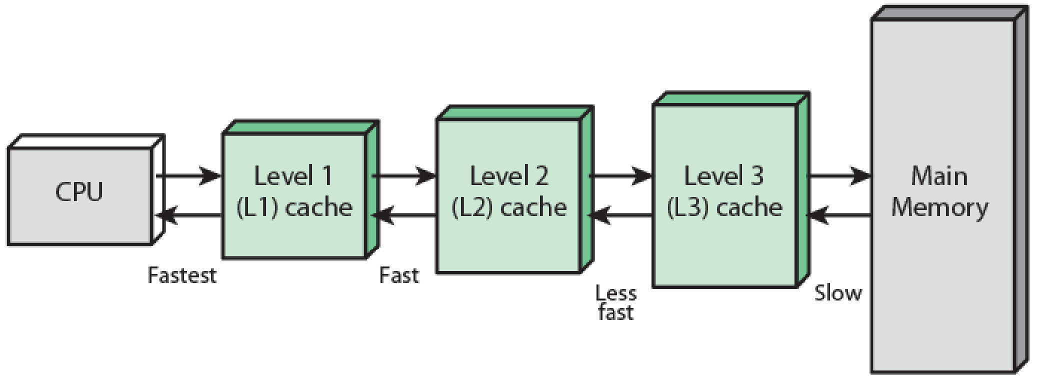
- Registers
- L1 Cache
- L2 Cache
- Main memory
- Disk cache
- Disk
- Optical
- Tape

Cache and Main Memory

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

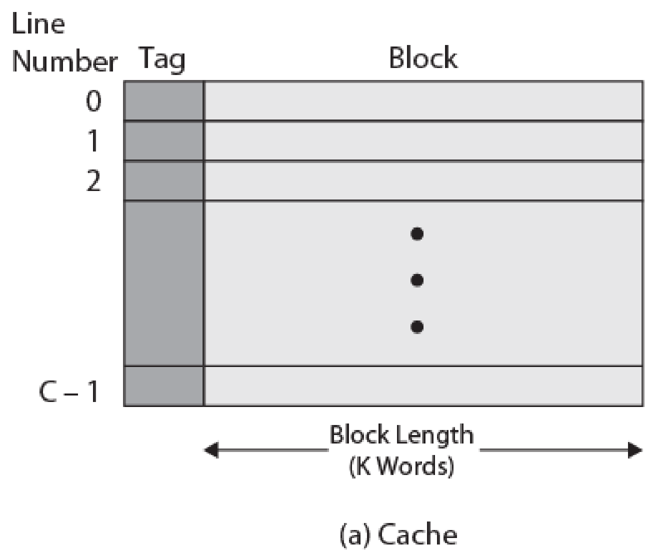


(a) Single cache

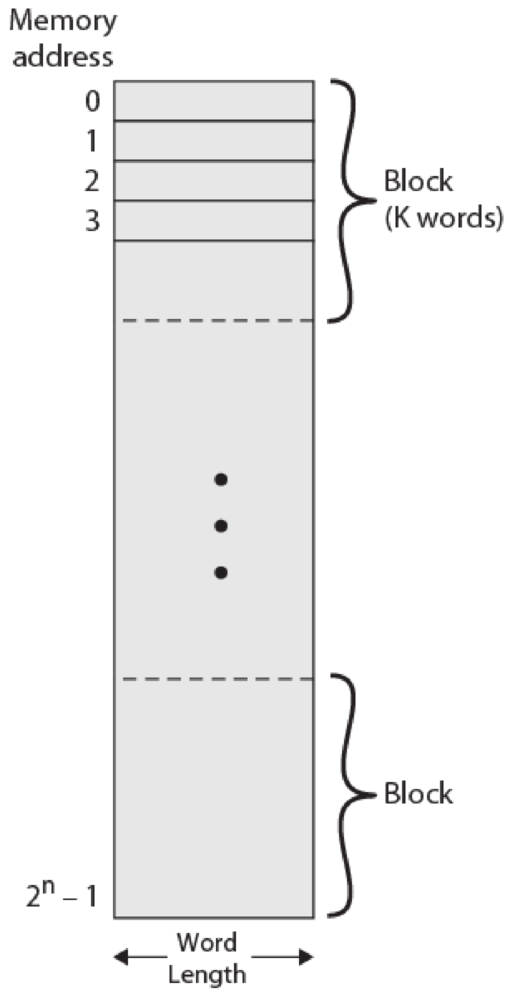


(b) Three-level cache organization

Cache/Main Memory Structure



(a) Cache



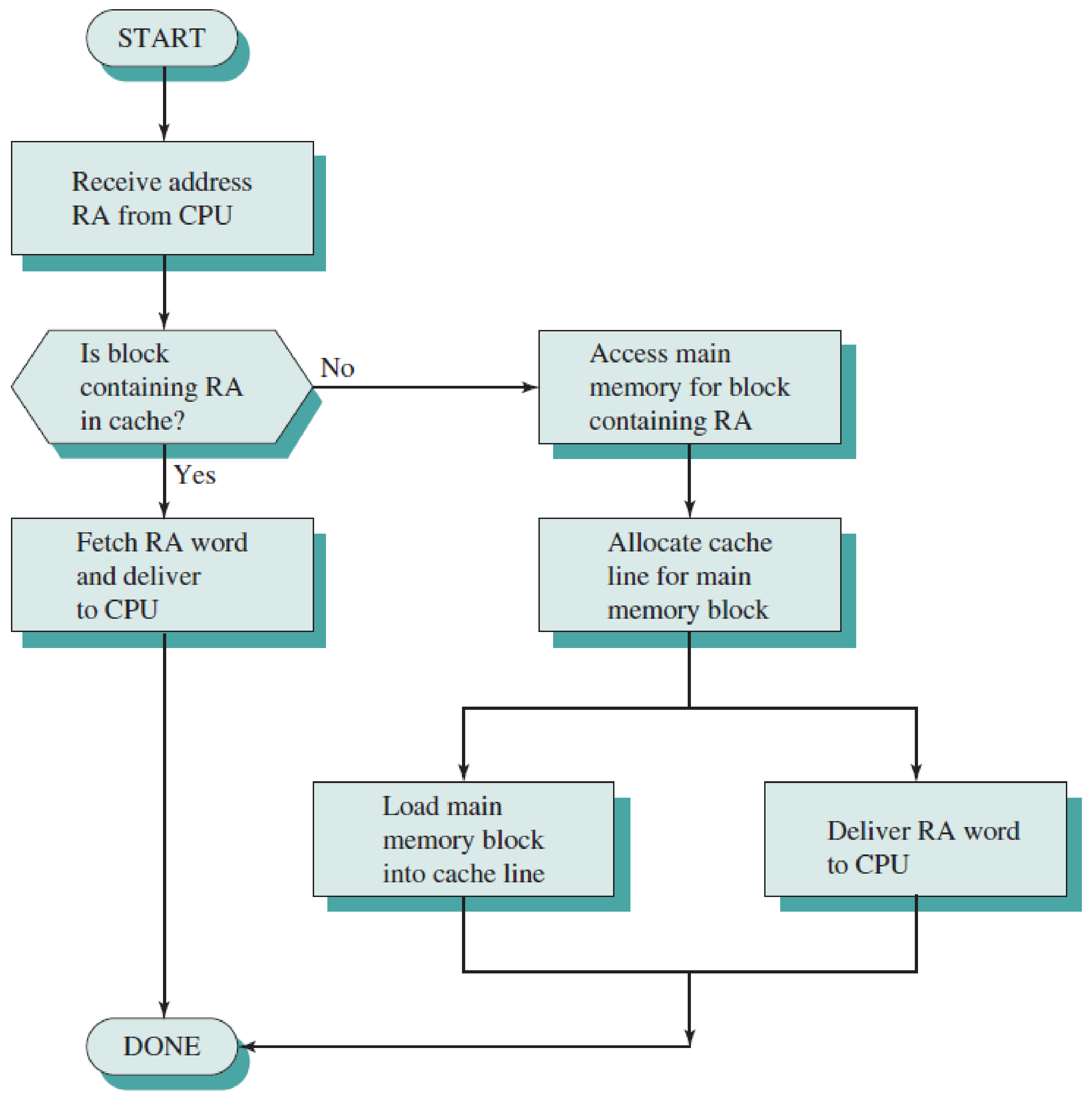
(b) Main memory

Following figures depicts the structure of a cache/main-memory system. Main memory consists of up to 2^n addressable words, with each word having a unique n-bit address. For mapping purposes, this memory is considered to consist of a number of fixed-length blocks of K words each. That is, there are $M = 2^n / K$ blocks in main memory. The cache consists of m blocks, called lines.3 Each line contains K words,

Cache operation – overview

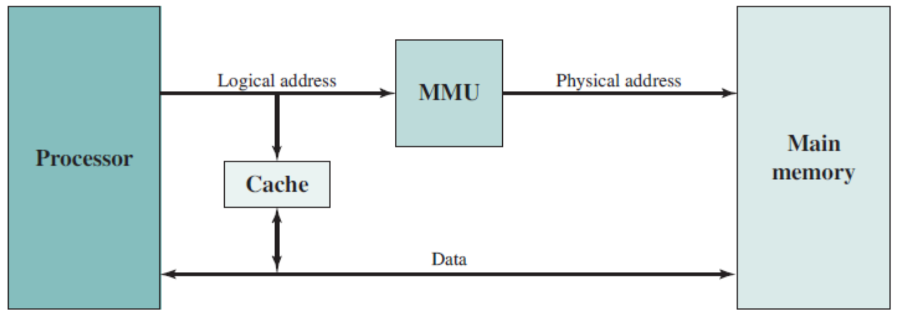
- CPU requests contents of memory location
- Check cache for this data
- If present, get from cache (fast)
- If not present, read required block from main memory to cache
- Then deliver from cache to CPU
- Cache includes tags to identify which block of main memory is in each cache slot

Cache Read Operation - Flowchart

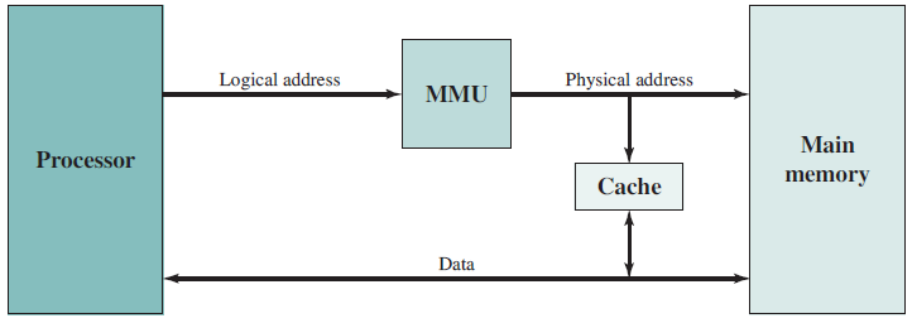

START
Receive address RA from CPU
Is block containing RA in cache?
No
Yes
Access main memory for block containing RA
Fetch RA word and deliver to CPU
Allocate cache line for main memory block
Load main memory block into cache line
Deliver RA word to CPU
DONE

Cache 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, 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
- Physical cache stores data using main memory physical addresses



(a) Logical cache



(b) Physical cache

Mapping Function

Because there are fewer cache lines than main memory blocks, an algorithm is needed for mapping main memory blocks into cache lines. Further, a means is needed for determining which main memory block currently occupies a cache line. The choice of the mapping function dictates how the cache is organized. **Three** techniques can be used: direct, associative, and set associative. We examine each of these in turn. In each case, we look at the general structure and then a specific example.

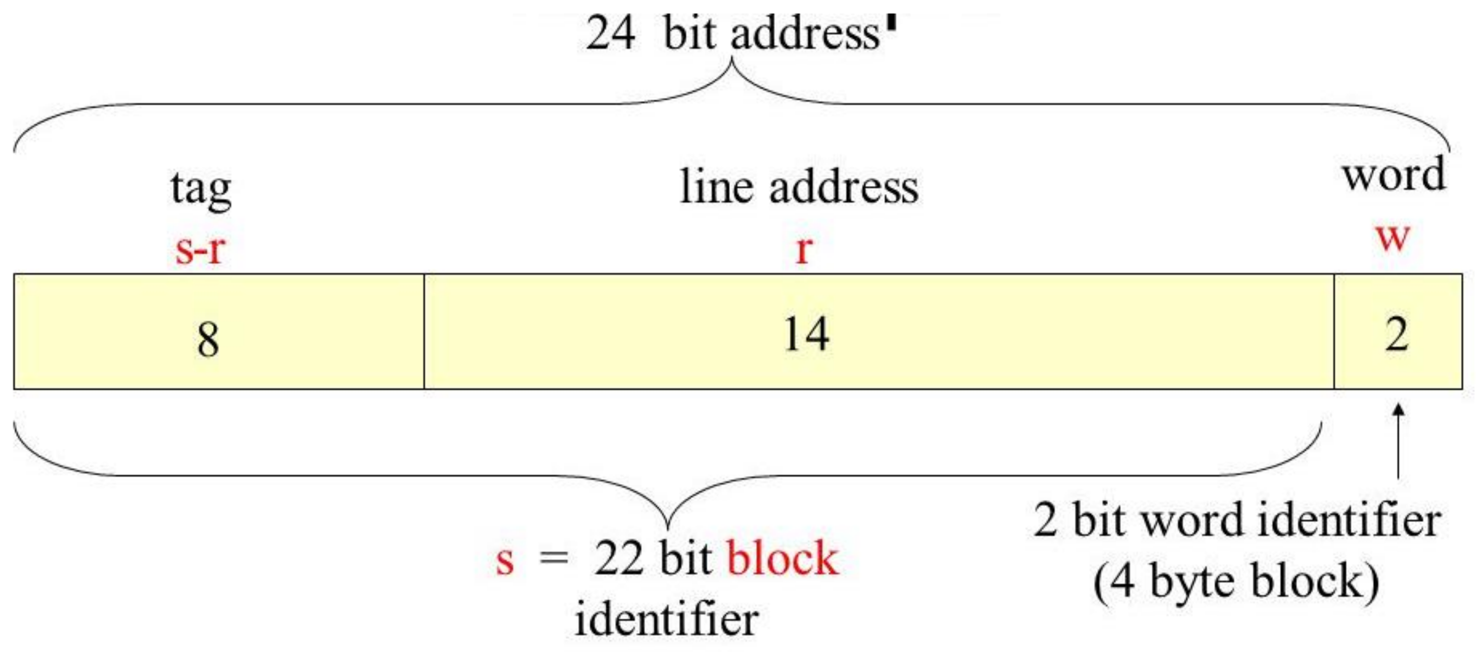
- The cache can hold 64 Kbyte (*65536 bytes*)
- Data can be transferred between main memory and the cache in block of 4 bytes each
  - i.e. cache is 16k = 2^{14} lines of 4 bytes each (*16384 lines*)
- Main memory consists of 16Mbytes, 24 bit address directly addressable $2^{24} = 16M$

  --i.e. 4M blocks of 4 bytes each

Direct Mapping

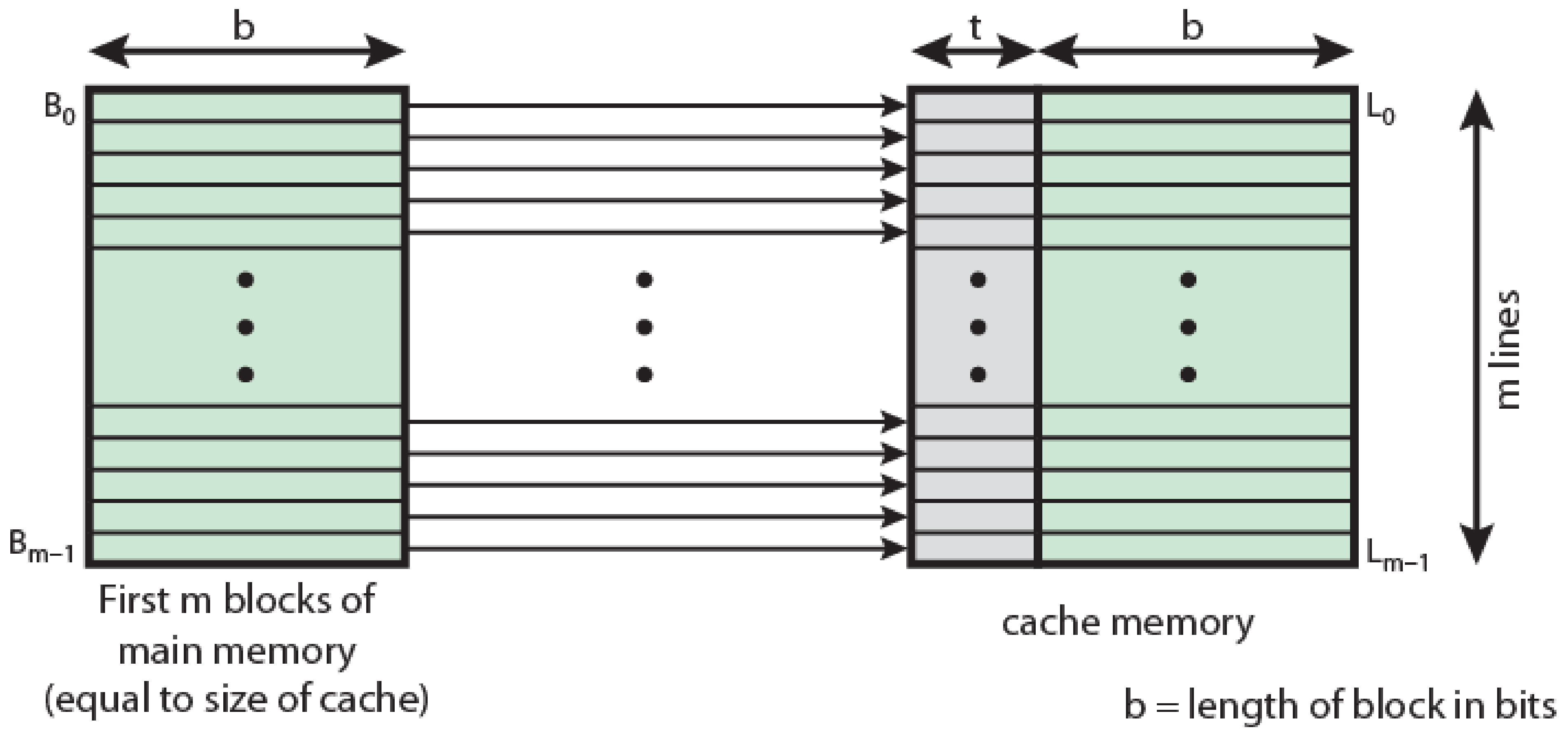
- Each block of main memory maps to only one cache line
  - i.e. if a block is in cache, it must be in one specific place
- Address is in two parts
- Least Significant w bits identify unique word
- Most Significant s bits specify one memory block
- The MSBs are split into a cache line field r and a tag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

Direct Mapping Address Structure


24 bit address
tag
s-r
line address
r
word
w
8
14
2
s = 22 bit block identifier
2 bit word identifier (4 byte block)

- 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
t
b
B_0
B_{m-1}
L_0
L_{m-1}
m lines
First m blocks of main memory (equal to size of cache)
cache memory
b = length of block in bits
t = length of tag in bits

(a) Direct mapping

Direct Mapping Summary

- Address length = (s + w) bits (i.e. 22+2 = 24 bits)
- Number of addressable units = 2^{s+w} words or bytes (i.e. 16Mbytes)
- Block size = line size = 2^w words or bytes (i.e. $2^2 = 4$ bytes)
- Number of blocks in main memory

  $= 2^{s+w}/2^w = 2^s$ (i.e. 4194304)
- Number of lines in cache = m = 2^r (i.e. 16384)
- Size of tag = (s – r) bits

Video Link: https://youtu.be/eObN3u3eAnU

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

Video Link: https://youtu.be/sLCJJdz0WAg

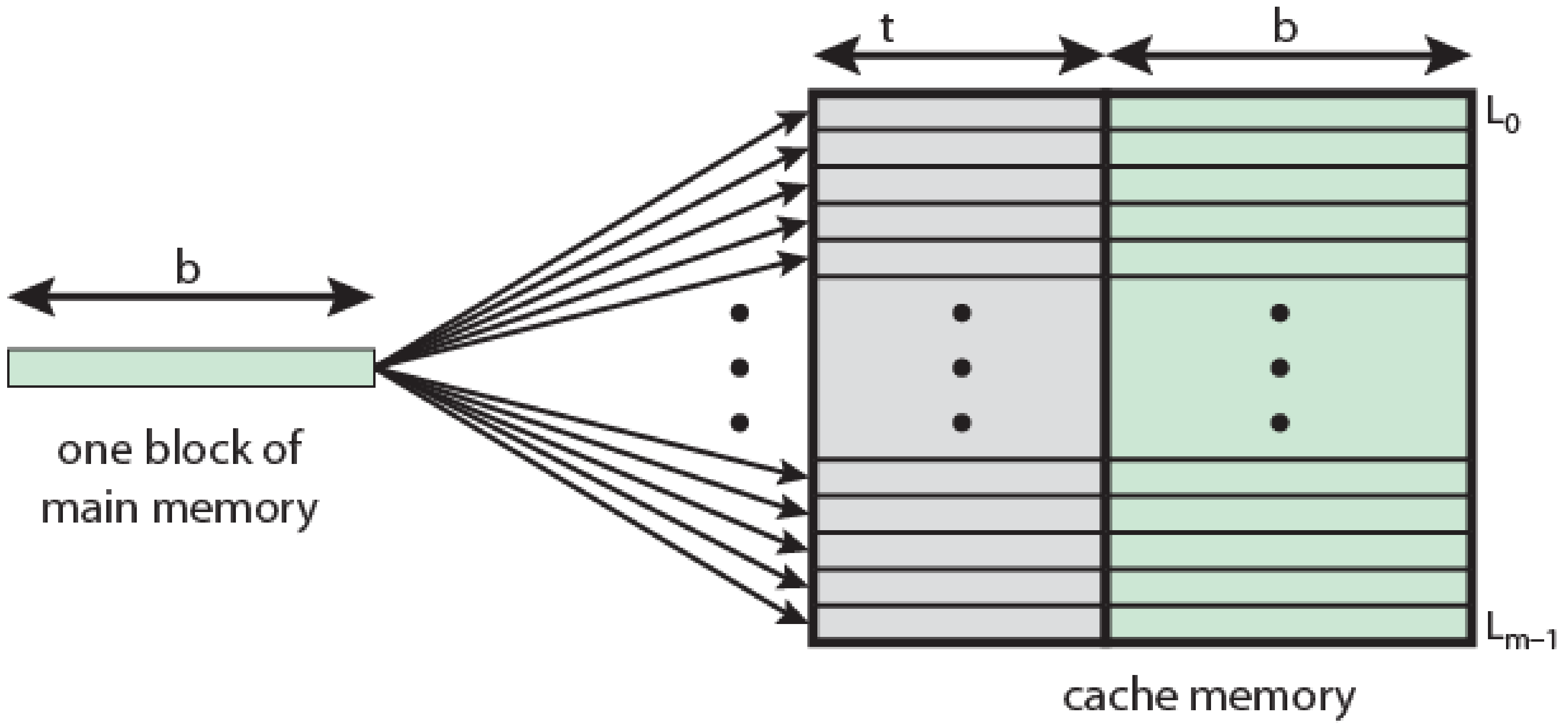
Associative Mapping Address Structure

Tag	Word
22 bits	2 bits

- 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	FFFFFC	24682468	3FFF

Associative Mapping from Cache to Main Memory



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
- 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

Video Link: https://youtu.be/pFndaJARM4Q

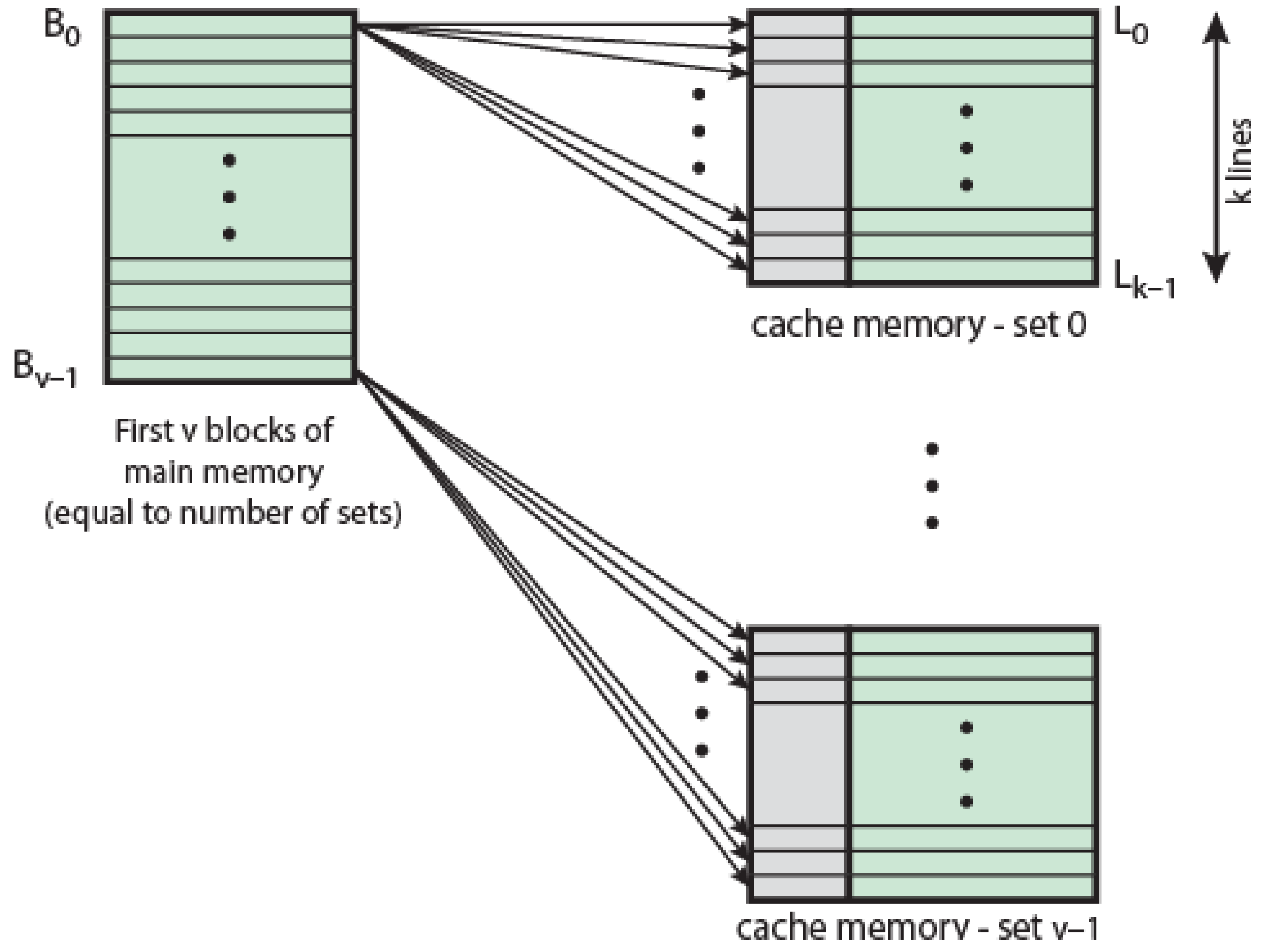
Set Associative Mapping Address Structure

Tag 9 bit	Set 13 bit	Word 2 bit

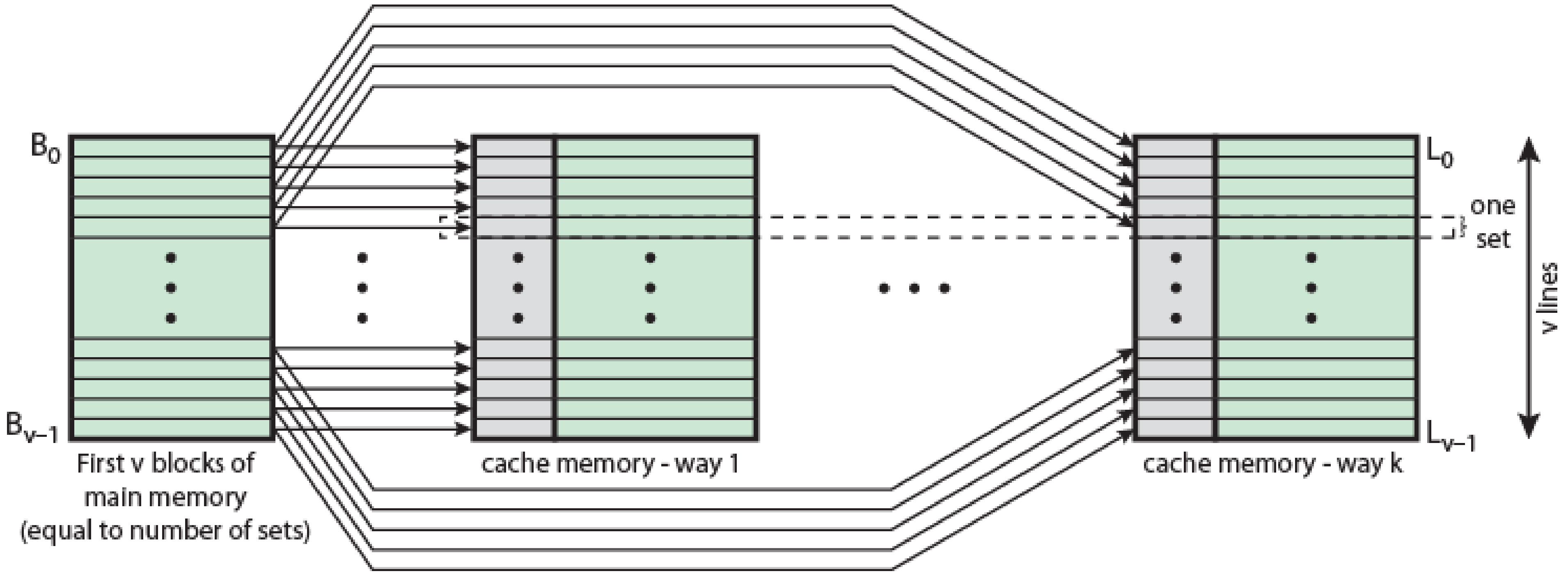
- Use set field to determine cache set to look in
- Compare tag field to see if we have a hit
- e.g

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

Mapping From Main Memory to Cache: v Associative



Mapping From Main Memory to Cache: k-way Associative



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^{d}
- 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

That's All
Thank You