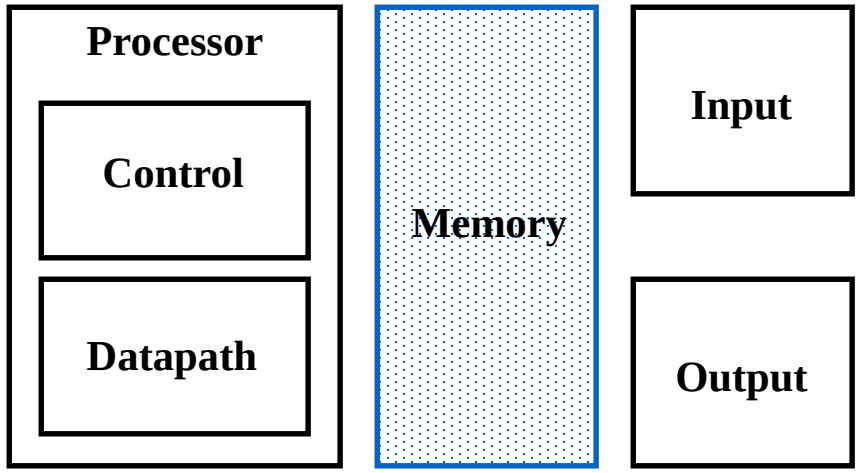
Memory Organization

for better performance and cost effective systems

Computer Organization and Architecture, William Stallings

The Big Picture: Where are We Now?

- The Five Classic Components of a Computer
- Memory is usually implemented as:
  - Dynamic Random Access Memory (DRAM) - for main memory
  - Static Random Access Memory (SRAM) - for cache



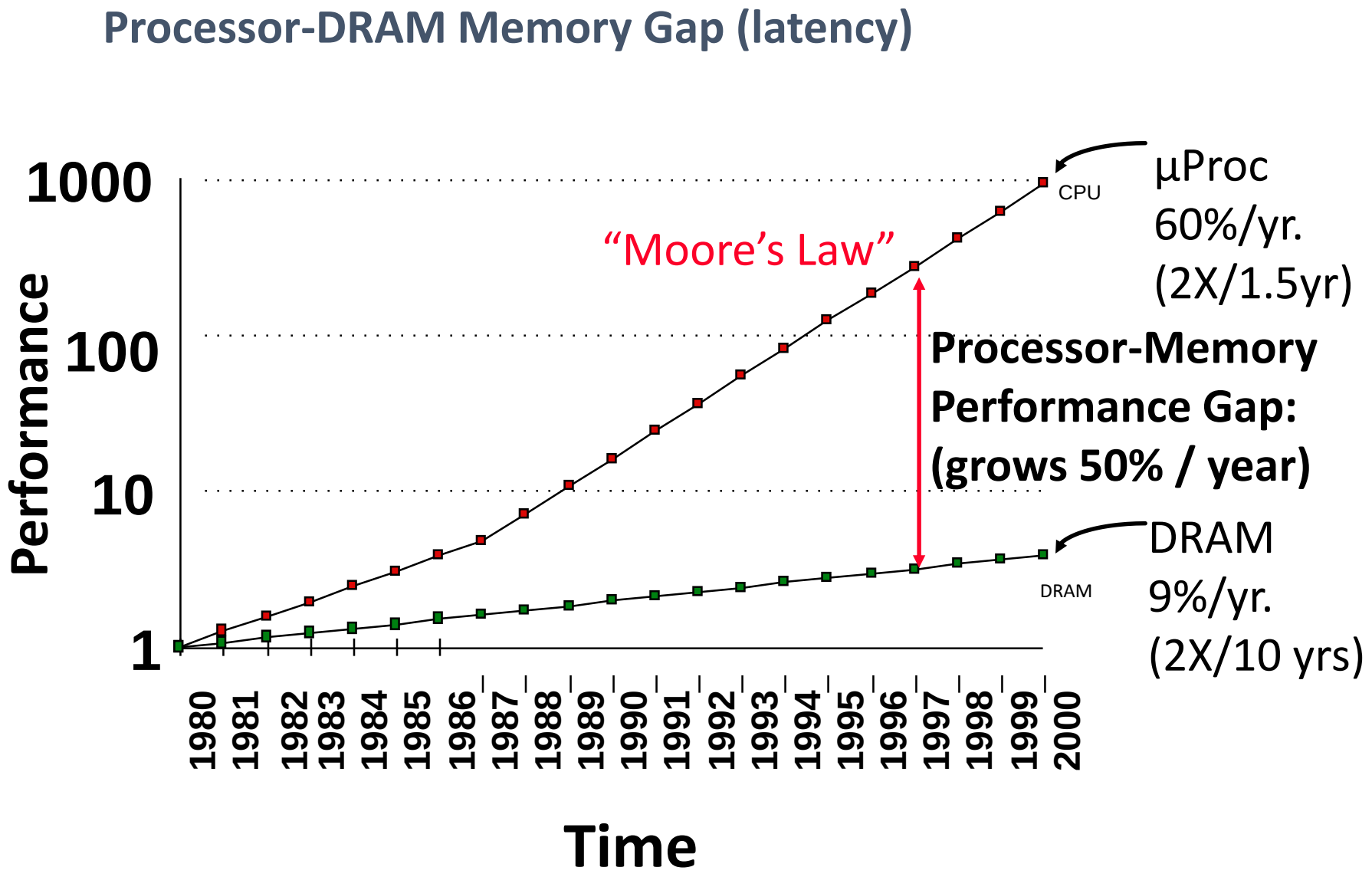
Characteristics of memory systems

- Location
  - CPU ; registers
  - Internal (on board) ; cache and RAM
  - External ; HDD
- Capacity
  - Word size and number of words (i.e. number of memory locations)
- Access method
  - Sequential ; Tape
  - Random ; RAM or ROM
  - Direct or semi-random ; HDD

Characteristics of memory systems

- Physical type
  - Semiconductor ; RAM, ROM
  - magnetic ; HDD, FDD
  - Optical ; CDs, DVDs
- Physical characteristics
  - Volatile and non-volatile
  - Erasable and non- erasable

Why should we bother about Memory speed?



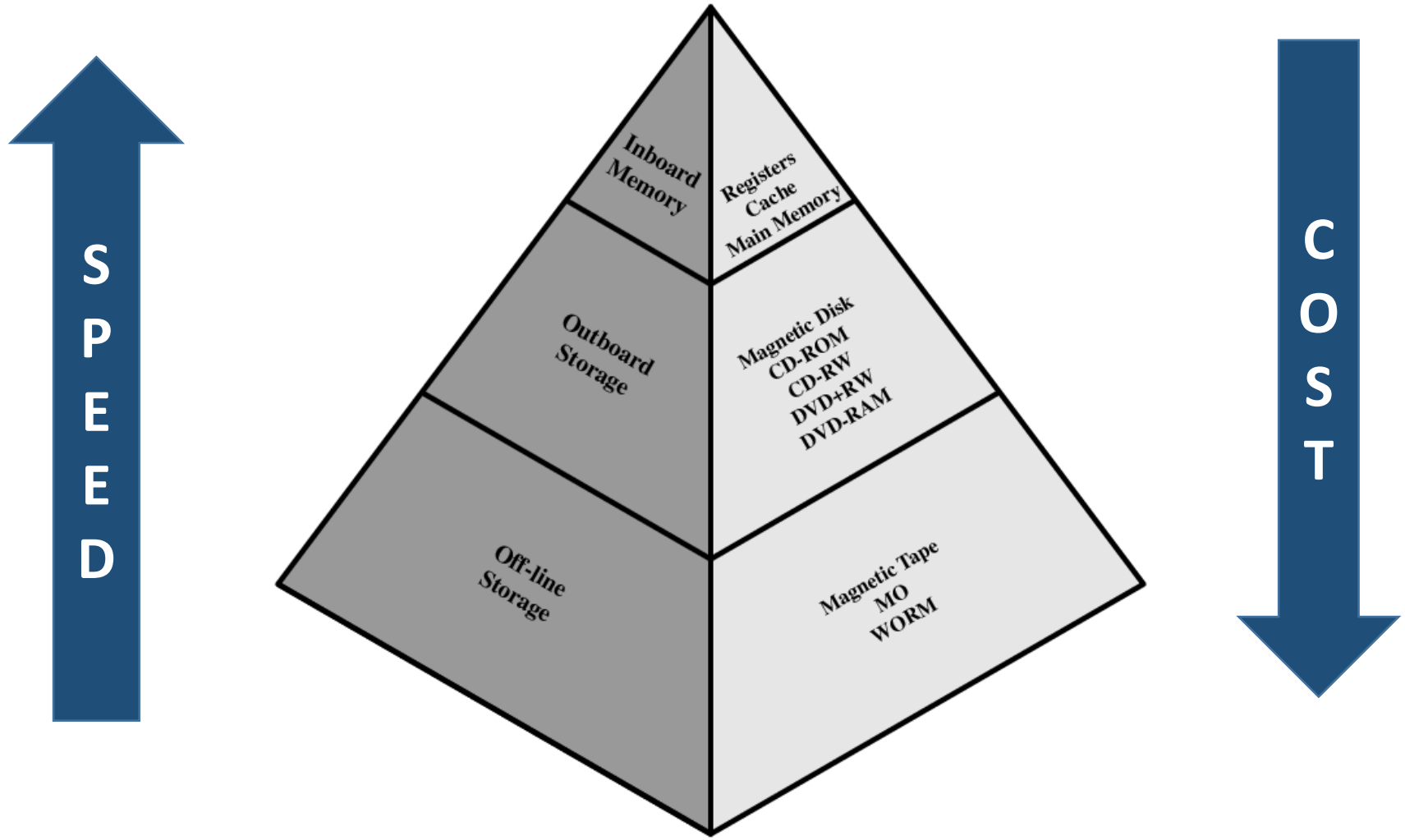
Characteristics of memory systems

- Performance
  - Access time
    - For RAM access time is the time between presenting an address to memory and getting the data on the bus
  - Cycle time
    - Primarily applied to RAM; access time + additional time before a second access can start
    - — memory system needs to recover for the next access
    - — Function of memory components and system bus, not the processor

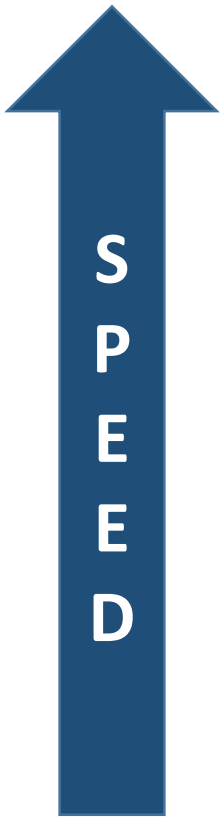
Semiconductor RAM

- Static RAM
  - Flip-flop based – data is in flip-flops
  - Fast
  - Bulky
  - expensive
- Dynamic RAM
  - Capacitor based - data is in form of charge on the capacitor
  - Needs refreshing of data
  - Compact
  - Cheap
  - Slower

Memory Hierarchy - Diagram



Hierarchy List



- Registers – semiconductor memory on the CPU; fastest
- Cache – Static RAM placed close to CPU
- Main memory – Dynamic RAM placed on board
- Disk – for bulk data
- Optical – for bulk data
- Tape – for bulk data

The Bottom Line

- Performance of CPU depends on access time of memory.
- How to build a system which is having best performance and at the same time low cost?
- How much?
  - Capacity KB , MB, GB, TB ?
- How fast?
  - Access / Transfer Rate
- How expensive?
  - $$$$$

fast and cost effective system

- Is it possible to build a computer which uses only static RAM (large capacity of fast memory)?
- This would be a very fast computer
- also very costly

- To reduce cost, can we use the below mentioned configuration?
  - Kilo bytes of cache (SRAM)
  - Mega bytes of DRAM
  - Giga bytes of secondary memory
  - This will definitely cost less
- will this give best performance?

Locality of reference

- We know that code should be loaded in main memory for it to execute
- One good observation is that during the course of the execution of a program, memory references tend to cluster
- e.g. programs - loops, nesting, …

  data – strings, lists, arrays, …

- Therefore, answer to the previous question on performance is YES!!!
- WE GET THE BEST PERFORMANCE AT LESS PRICE
- Its all possible only due to ‘ locality of reference’

Principle of locality

- Temporal locality (locality in time): if an item is referenced, it will tend to be referenced again soon.

- Spatial locality (locality in space): if an item is referenced, items whose addresses are close by will tend to be referenced soon.

```
// Multiply the two matrices together
  for ( ty = 0 ; ty < BLOCK_SIZE ; ty++ ){
    for ( tx = 0 ; tx < BLOCK_SIZE ; tx++ ){
      Csub = 0.0 ;
      for (k = 0; k < BLOCK_SIZE; ++k ){
         Asub = As[ty][k ] ;
         Bsub = Bs[k ][tx] ;
         Csub += Asub * Bsub ;
      }
      c = wB * BLOCK_SIZE * by + BLOCK_SIZE * bx;
      C[c + wB * ty + tx] += Csub;
    }// for tx ;
  }// for ty
```

for-loop is temporal locality

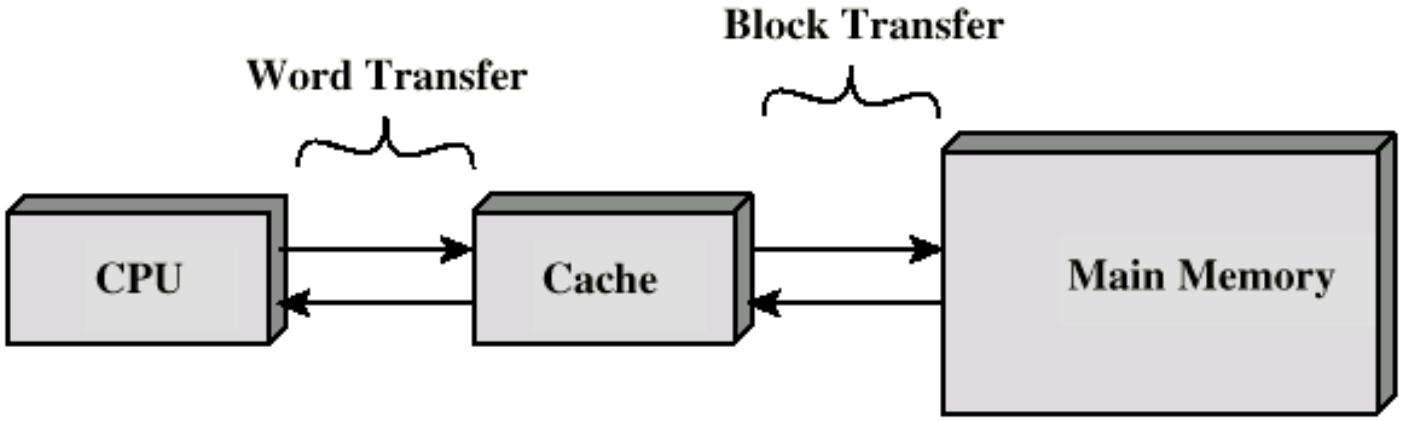
array is spatial locality

The Principal of Locality of reference

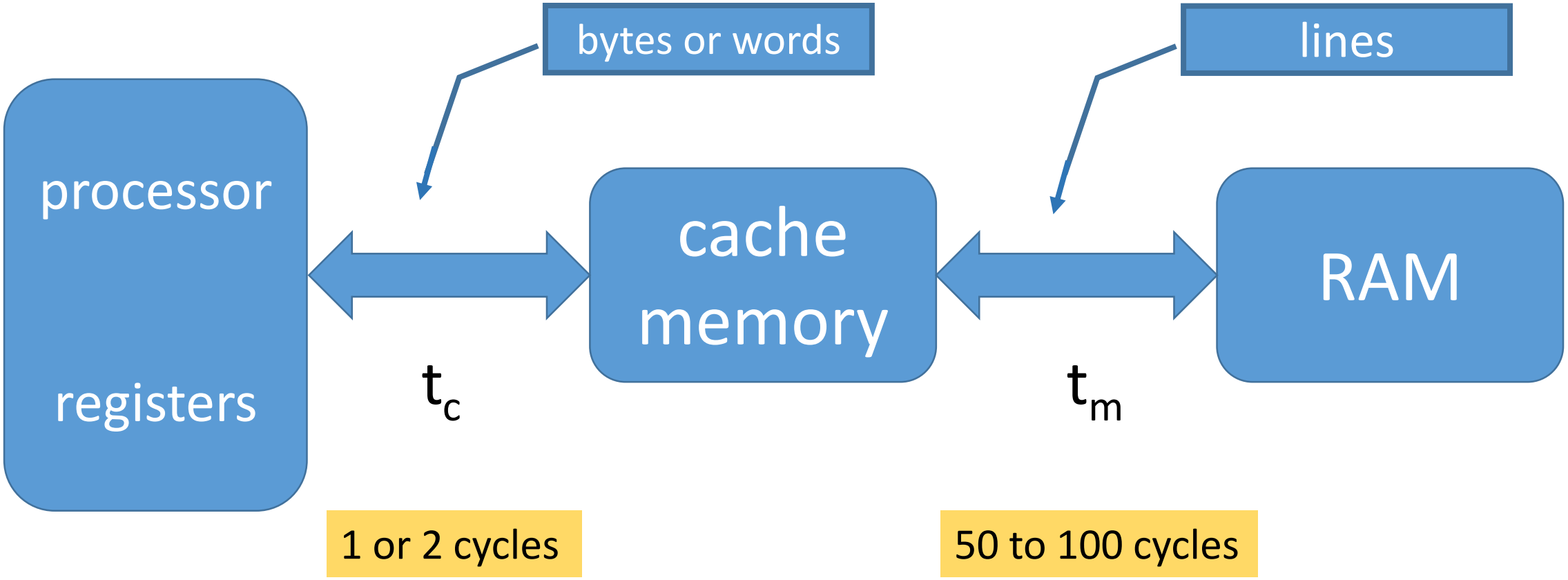
- How can we use very fast memory effectively?
- Temporal Locality (Close in Time)
  - Memory that has been accessed recently is most likely to be accessed again sooner
- Spatial Locality (Close in location)
  - Memory that is close to memory that has been accessed recently is most likely to be accessed
- Organize memory in blocks
  - Keep blocks likely to be used soon in very fast memory
  - Keep the next most likely blocks in medium fast memory
  - Keep those not likely to be used soon in slower memory

Cache memory

- Small amount of fast memory
- Sits between main memory (RAM) and CPU
- May be located on CPU chip or in system
- Objective is to make slower memory system look like fast memory.

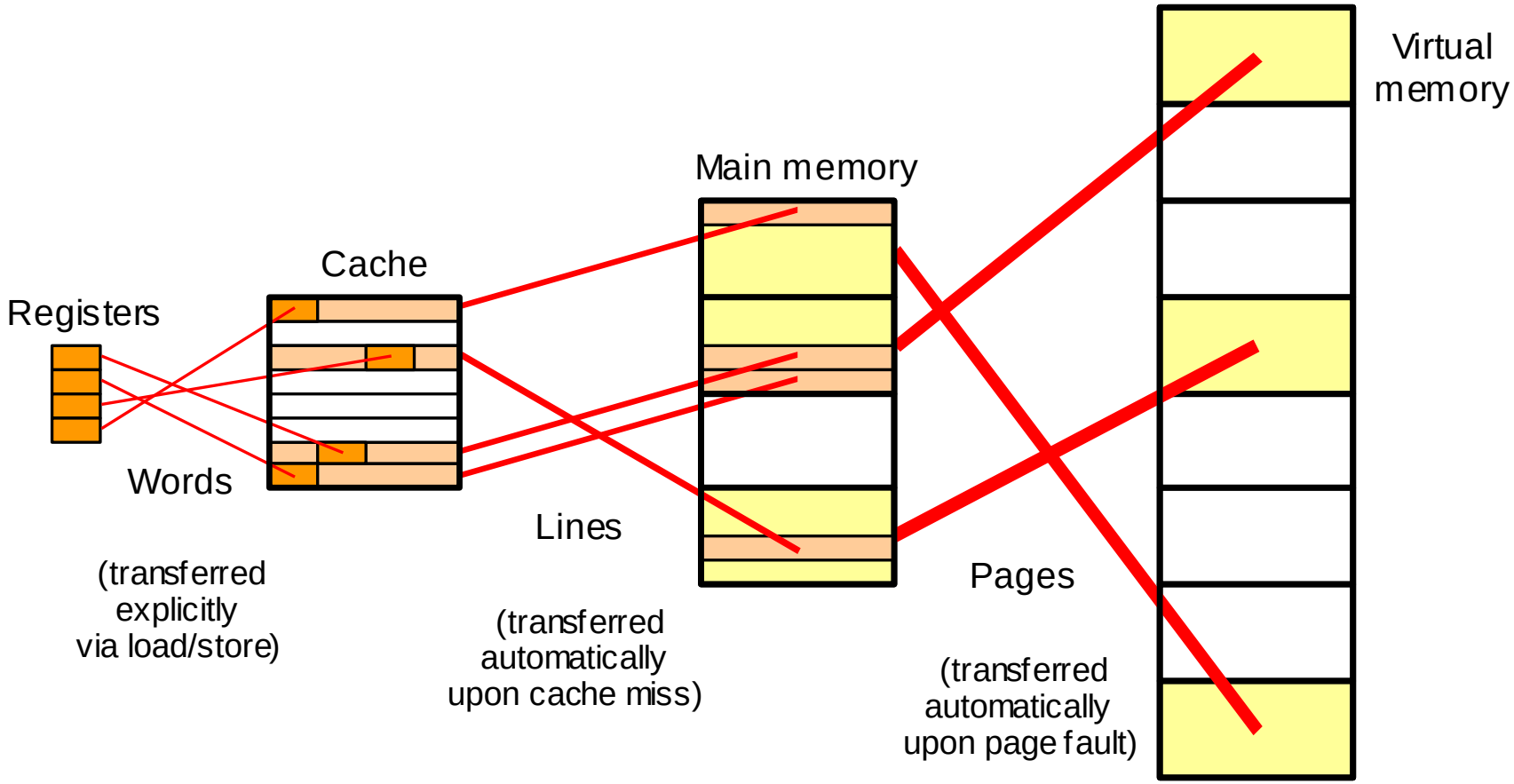


Cache memory



How much would be typical these access time in terms of cycles?

Memory Hierarchy: The Big Picture

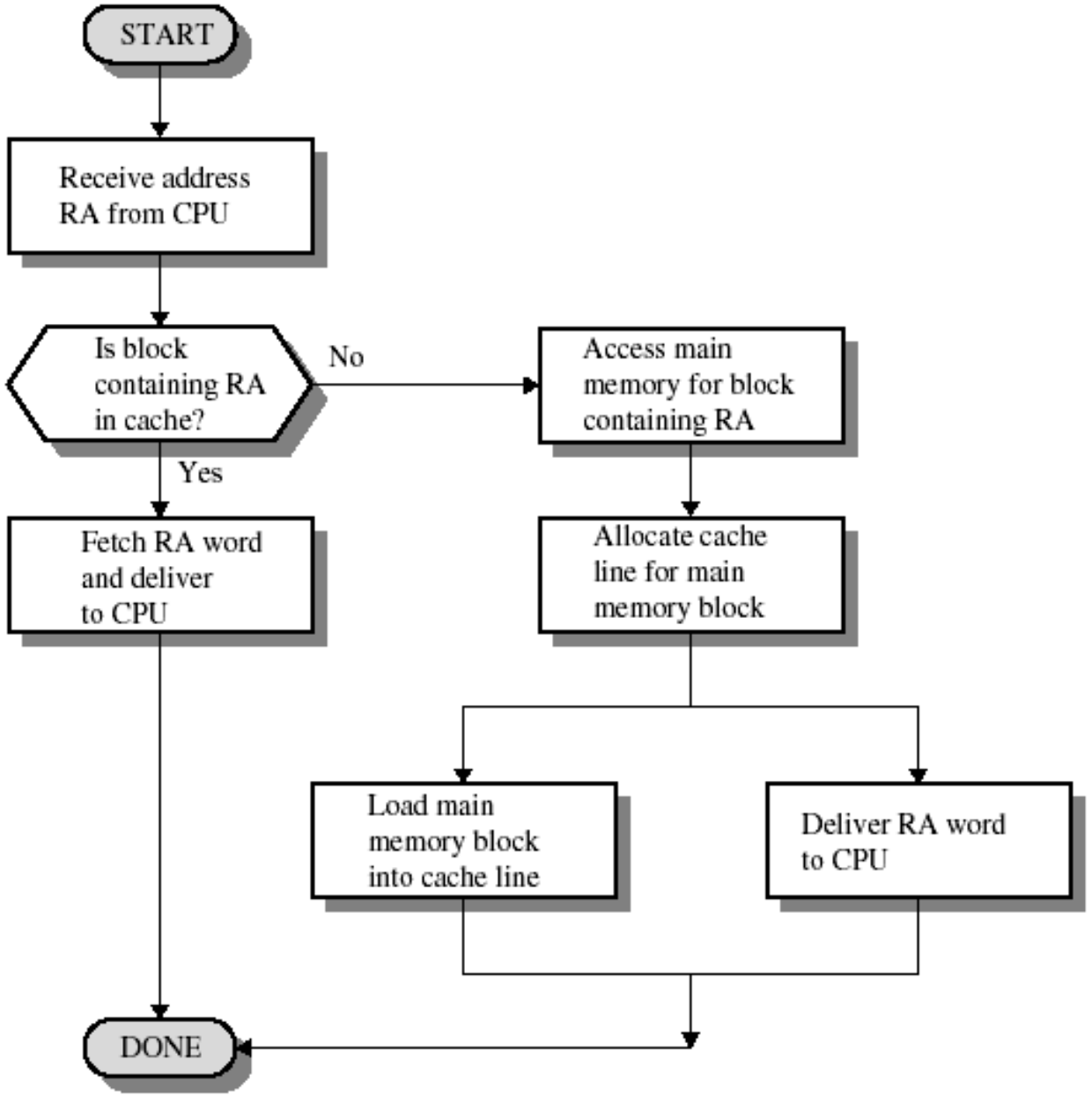


Data movement in a memory hierarchy.

Cache operation – overview

- CPU requests contents of memory location
- Check if this requested data is in cache
- If found in cache, it’s a hit;
- Else, it’s a miss ;
- Cache "misses" are unavoidable, i.e., every piece of data and code must be loaded at least once
- What does a processor do during a miss?
  - It waits for the data to be loaded from RAM to cache.
  - Loading is done in two possible ways:
    1. read required block from main memory to cache then deliver from cache to CPU (cache physically between CPU and bus)
    2. read required block from main memory to cache and simultaneously deliver to CPU (CPU and cache both receive data from the same data bus buffer)

Cache Operation



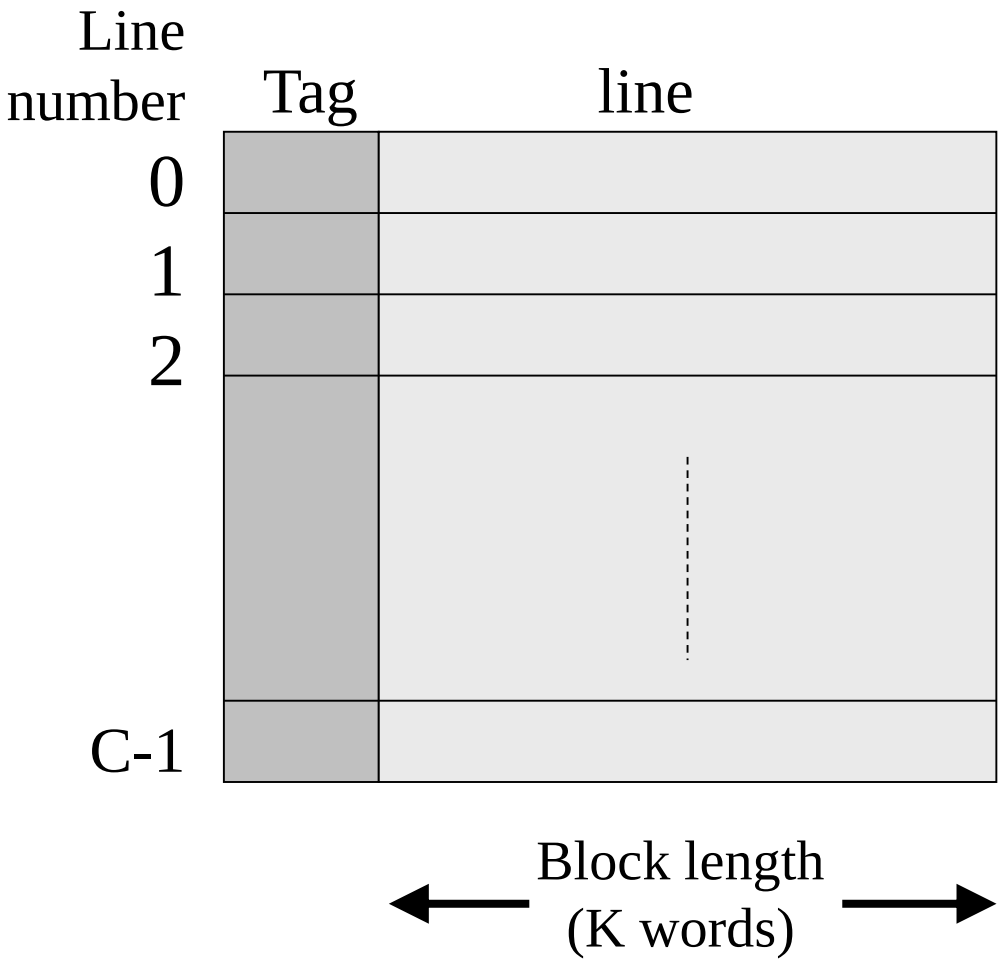
Few terms

- Cache hit rate, h
  - number of times the CPU found the required data in cache
  - Expressed in percentage
- Cache miss, (1-h)
  - number of times the CPU did not find the required data in cache
  - Expressed in percentage; typical numbers 3%, 10% etc.
- Average access time, $\bar{t} = h * t_c + (1 - h) * (t_m + t_c)$
- Cache efficiency, $\Lambda = \frac{t_c}{\bar{t}}$

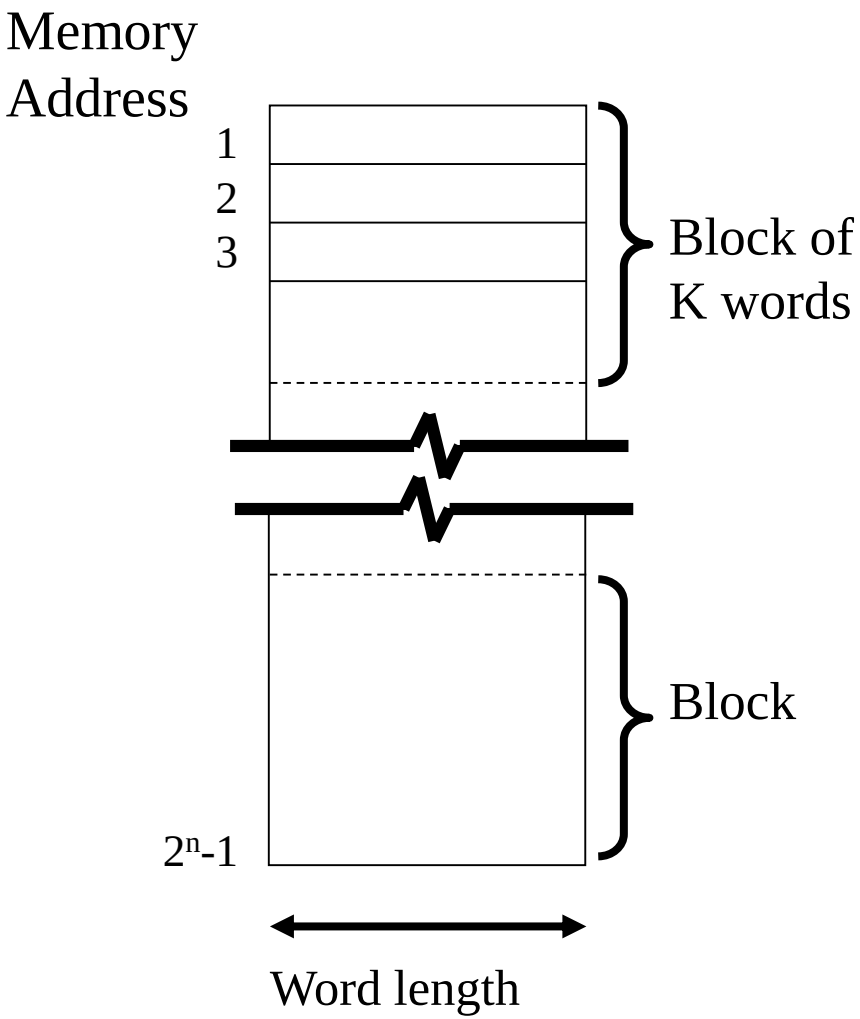
Cache Structure

- Cache is organized as ‘lines’ of k number of consecutive memory locations
- Memory is divided into ‘blocks’ of k number of consecutive memory locations
- ‘tag’ bits for each line of cache;
- tags store info (id) about the corresponding line

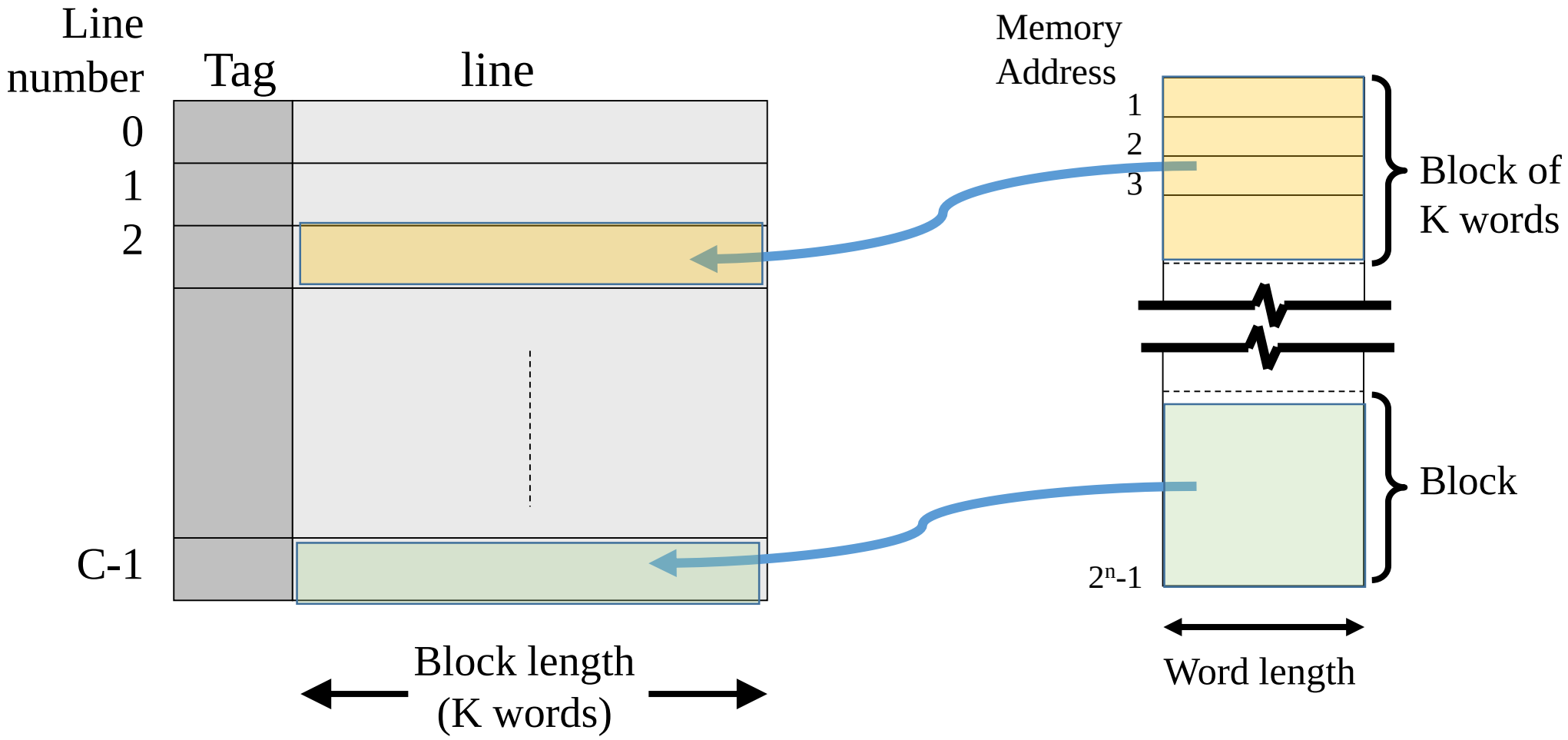
Cache Structure



Memory divided into blocks



Cache Structure



how are these lines and blocks linked and tracked?

Block Placement in cache line - Mapping

Where can a block be placed in cache?

We have 3 options as below.

1. Anywhere in cache - fully associative (or associative mapped)
2. In one predetermined place - direct-mapped
3. In a limited set of places - set-associative mapped
   - Hybrid of direct mapped and fully associative

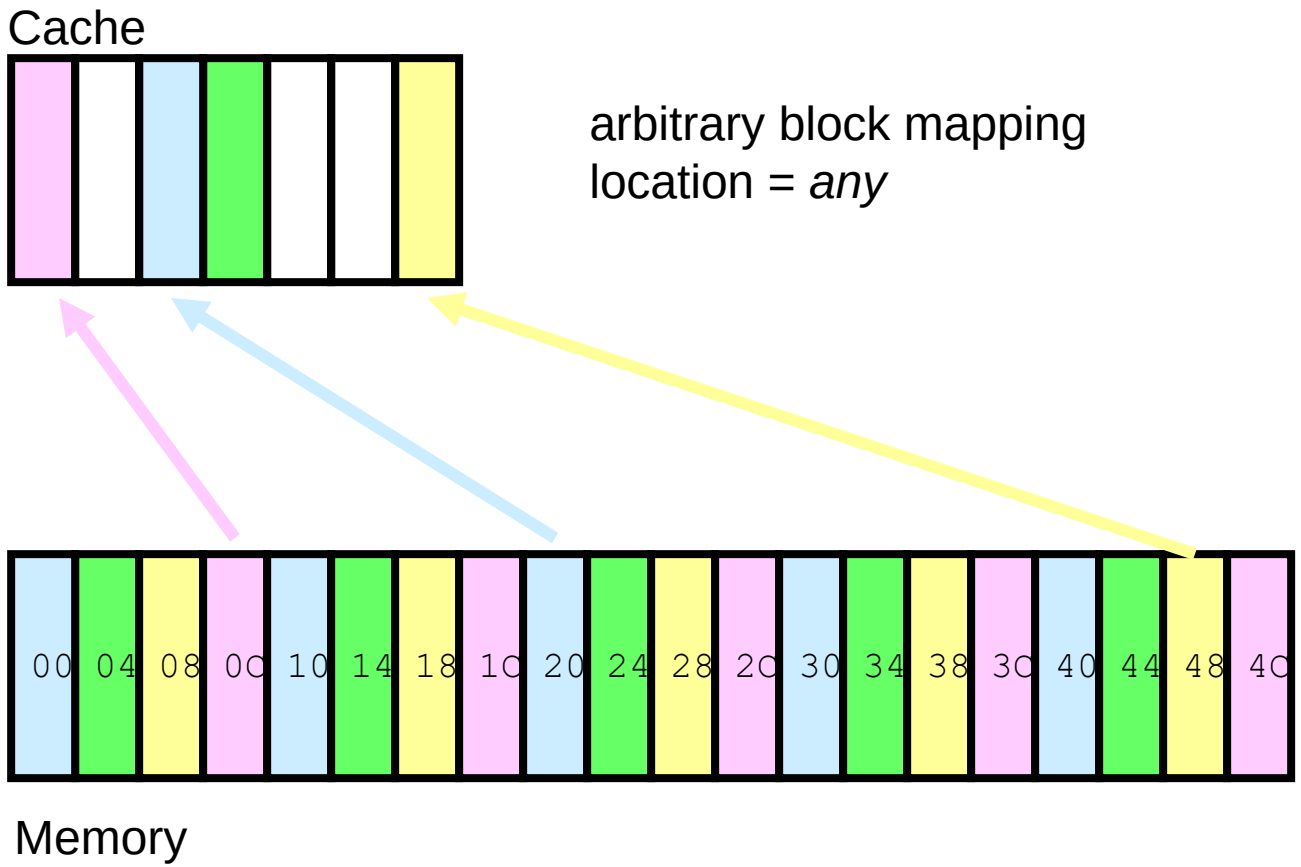
Block Placement in cache line - Mapping

- Where can block be placed in cache?
  1. Anywhere in cache - fully associative
     - Compare tag to every block in cache
  2. In one predetermined place - direct-mapped
     - Use part of address to calculate block location in cache
     - Compare cache block with tag to check if block present
  3. In a limited set of places - set-associative
     - Hybrid of direct mapped and fully associative
     - Use portion of address to calculate set (like direct-mapped)
     - Place in any block in the set
     - Compare tag to every block in set

Fully-associative Mapping

- A main memory block can load into any line of cache
- Every line's tag must be examined for a match
- Cache searching gets expensive and slower
- Memory address is interpreted as:
  - Least significant w bits = word position within block
  - Most significant s bits = tag used to identify which block is stored in a particular line of cache

Fully Associative Block Placement



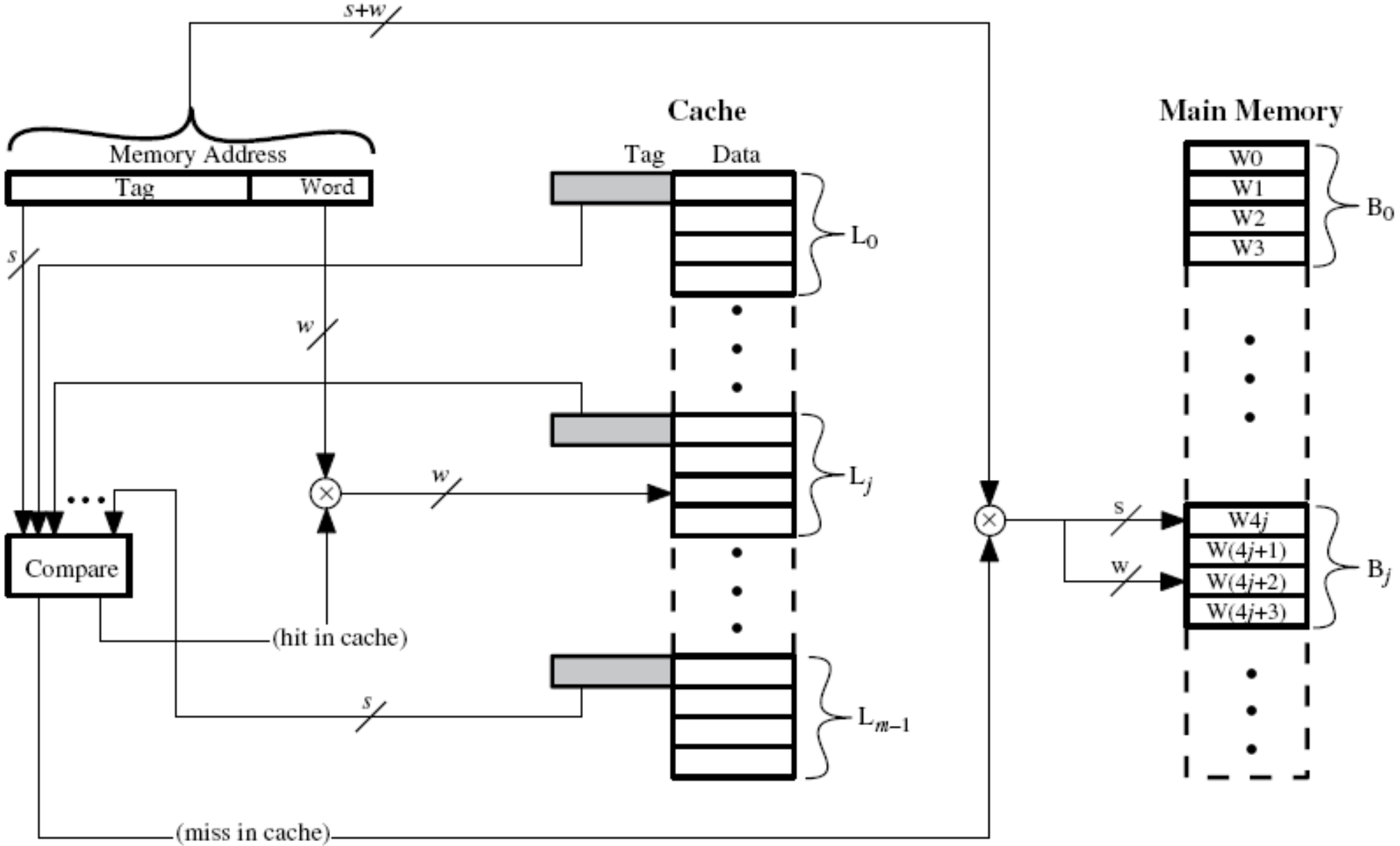
We can see that there are 4 bytes (32 bits) in each block.

Associative Mapping Address Structure Example

Tag – s bits (22 in example)	Word – w bits (2 in ex.)

- Let the address bits be 24 bits (address space?)
- 22 bit tag stored with each 32 bit block of data
- How many blocks are there in this example?
- Compare tag field with tag entry in cache to check for hit
- Least significant 2 bits of address identify which of the four 8 bit words is required from 32 bit data block

Fully Associative Cache Organization



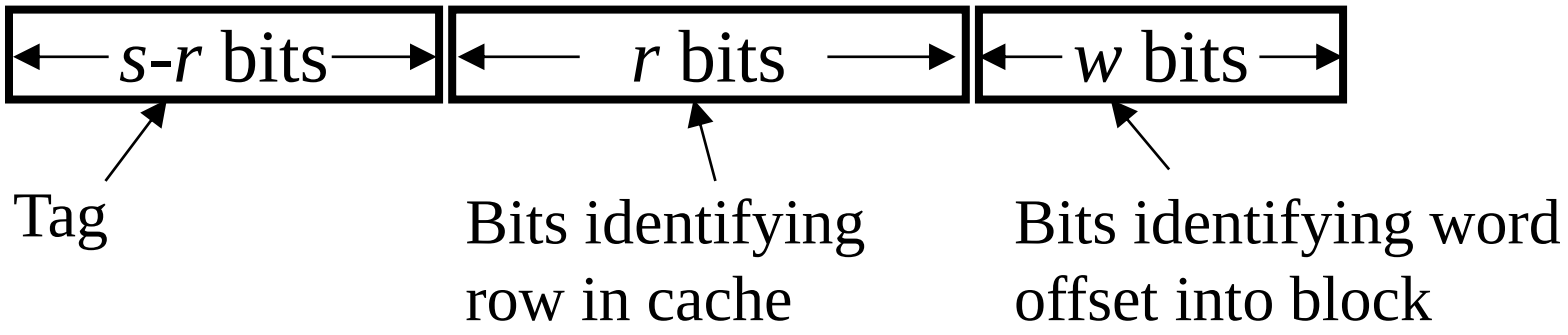
Associative Mapping Summary

- Address length = (s + w) bits
- Number of addressable locations = 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 (can be of any capacity)
- Size of tag = s bits
- Low value cache miss is expected

Direct Mapping Address Structure

Each main memory address (*s*+*w* bits) can by divided into three fields

- Least Significant *w* bits identify unique word within a block
- Remaining bits (*s*) specify which block in memory. These are divided into two fields
  - Least significant *r* bits of these *s* bits identifies which line in the cache
  - Most significant *s*-*r* bits uniquely identifies the block within a line of the cache



Direct Mapping Cache Line Table

number of lines = m
Number of blocks in main memory = 2^S

(S is the number of bits required to identify a block in main memory.)

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

- Each block of main memory maps to only one cache line – i.e. if a block is in cache, it will always be found in the same place.
- Line number is calculated using the following function

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

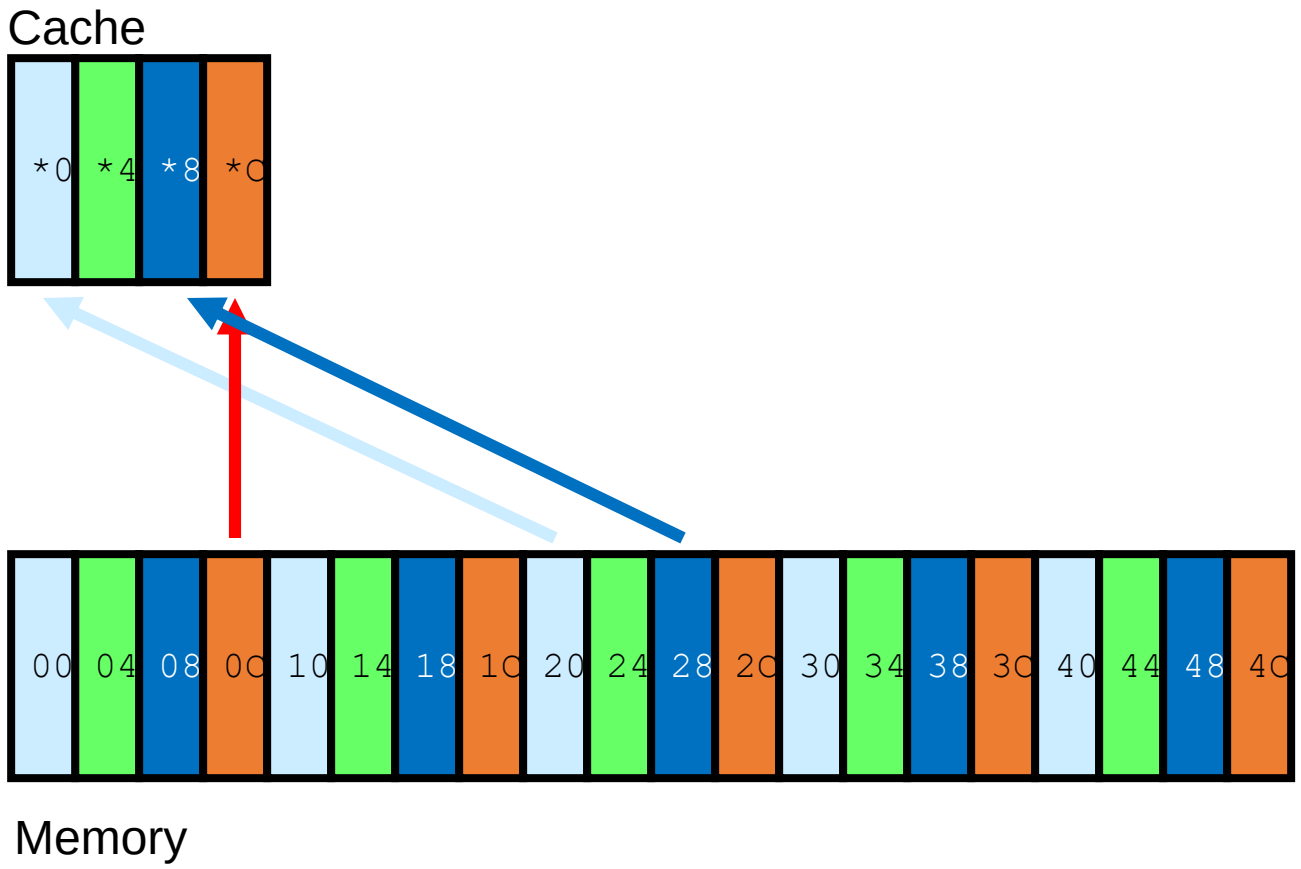
where

i = cache line number

j = main memory block number

m = number of lines in the cache

Direct Mapping illustration



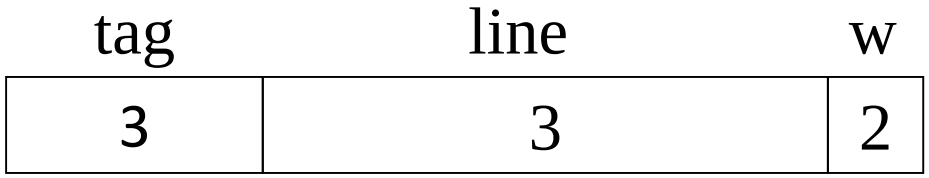
Direct mapping - illustration

Consider, main memory address of 8 bits and block size of 4 bytes

- 256 bytes of main memory
- 64 blocks in main memory

Consider, 8 lines cache, each has a length of 4 bytes

- 32 bytes of cache
- Need 3 bits to identify a line in the cache
- Need 2 bits to identify a byte in a line



A look at cache

Line number	Tag (3 bits)	Cache line
0	xyz	4 bytes of data in each line
1	pqr	
2	abc	
3		
….		
7		

Block numbers that can sit in each line

Pre determined 8 blocks can sit here

Pre determined 8 blocks can sit here

⋮

Pre determined 8 blocks can sit here

Direct mapping- illustration

tag	line	w
3	3	2

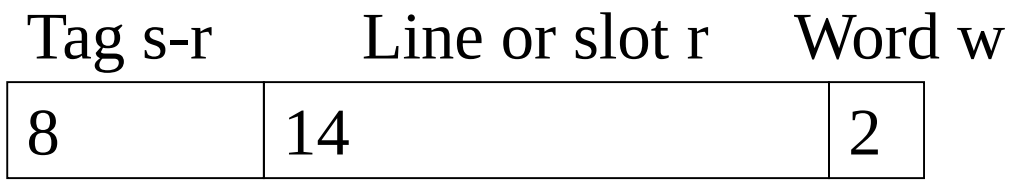
memory address (*s*+*w* bits)
S bits specify which block in memory.
Least significant *r* bits of these *s* bits identifies which line in the cache

0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0
0	0	0	0	1	1	1	0
0	0	1	0	1	0	0	0
0	0	1	1	0	0	0	0
0	1	1	1	1	0	0	0
0	1	1	1	1	1	0	0
1	1	1	1	1	1	1	1

00h	Block 0, line 0
04h	Block 1, line 1
08h	Block 2, line 2
0Eh	Block 3, line 3
28h	Block 10, line 2
30h	Block 12, line 4
78h	Block 30, line 6
7Ch	Block 31, line 7
FFh	Block 63, line 7

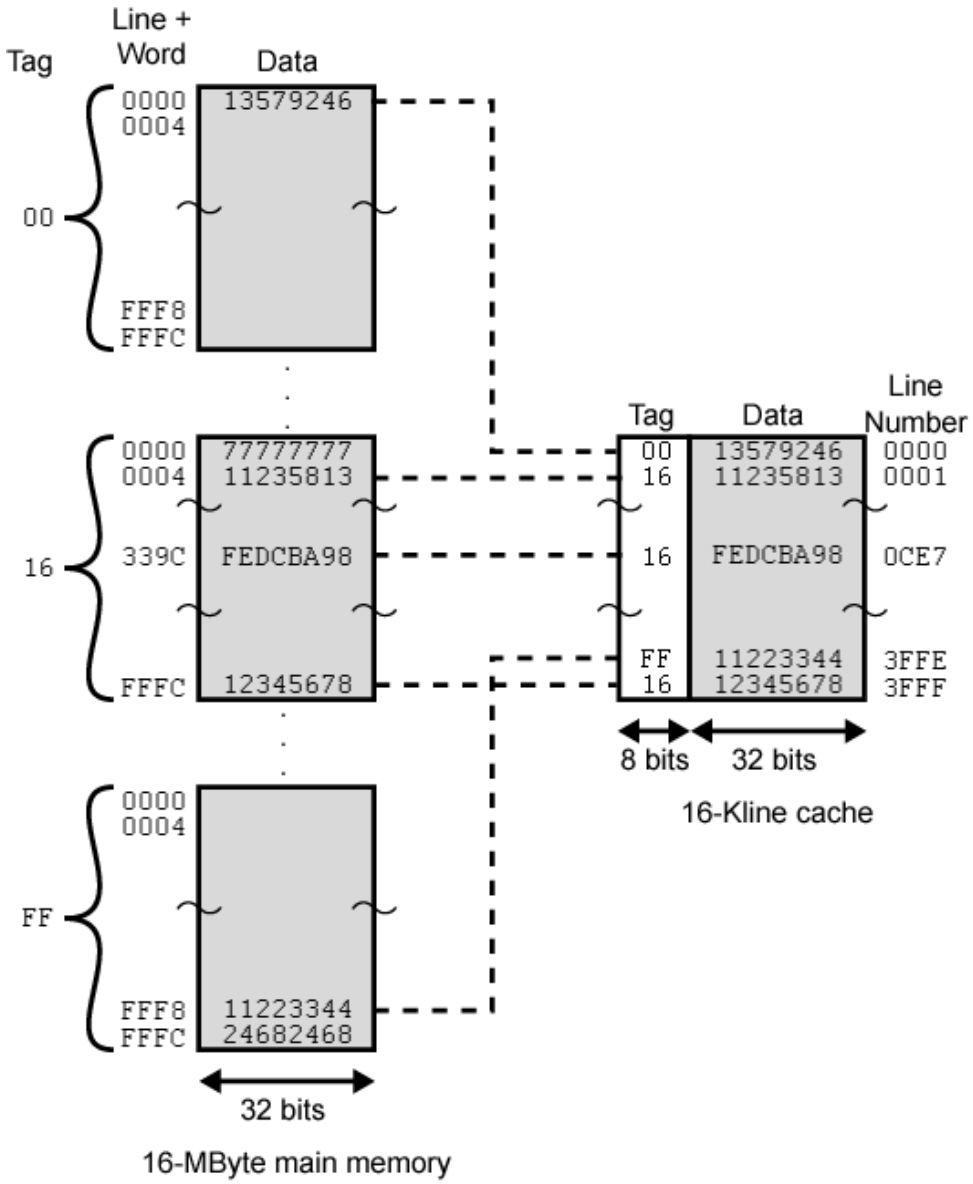
00h
04h
08h
0Eh
28h
30h
78h
7Ch
FFh

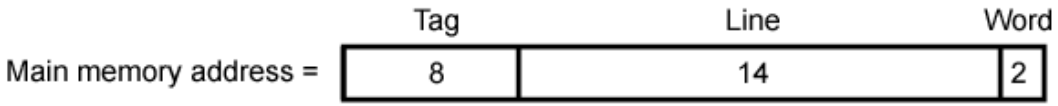
Direct Mapping Address Structure Example



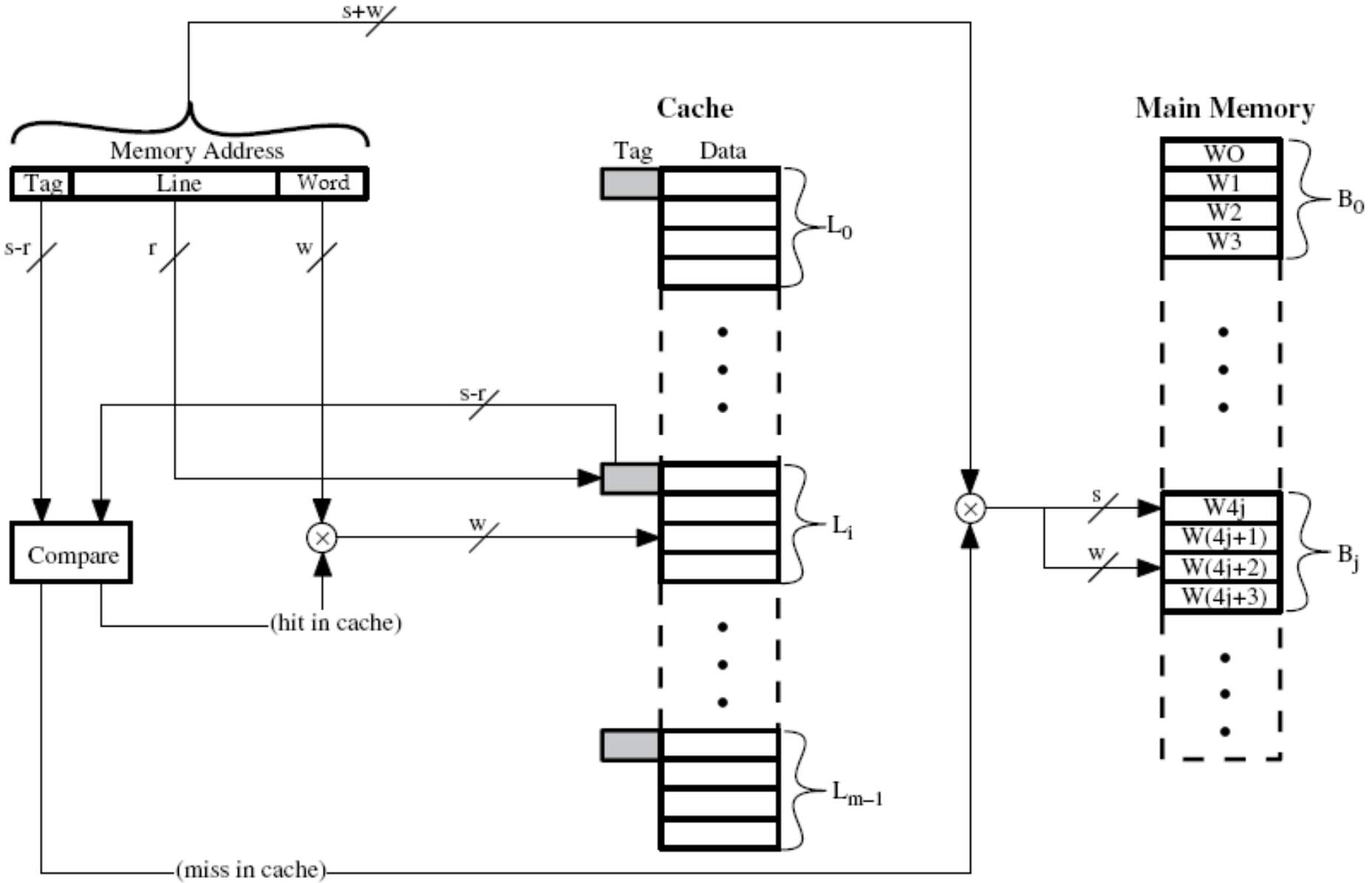
- 24 bit address
- 2 bit word identifier (4 byte block)
- 22 bit block identifier
- 8 bit tag
- 14 bit slot or line
- No two blocks in the same line have the same tag
- Check contents of cache by finding line and comparing tag

Direct mapping





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 width = 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 (thrashing)

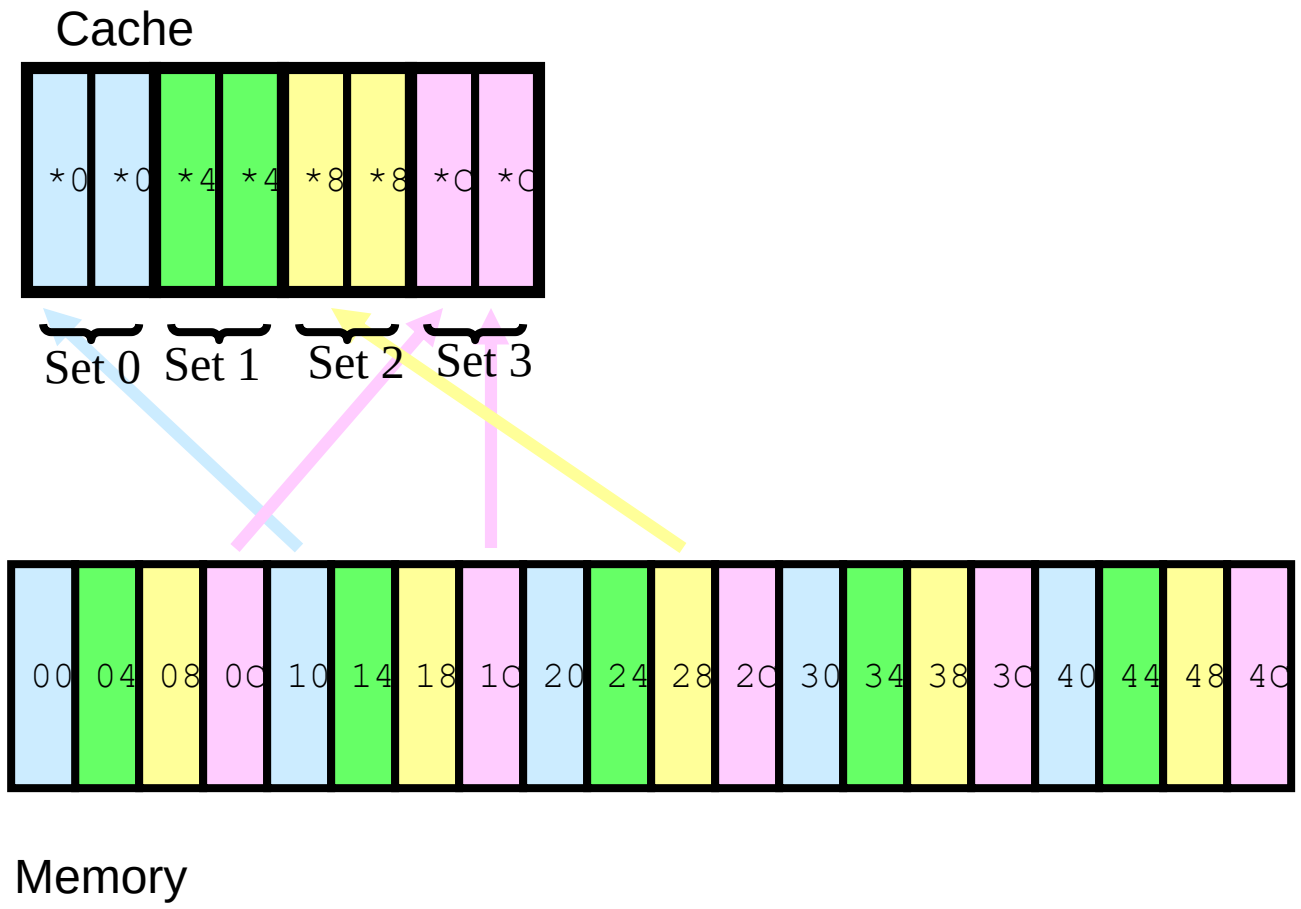
Set Associative Mapping

- Address length is $s + w$ bits
- Cache is divided into a number of sets, $v = 2^d$
- k blocks (or lines) can be contained within each set
- k lines in a cache is called a k-way set associative mapping
- Number of lines in a cache $= v \bullet k = k \bullet 2^d$
- Size of tag = $(s-d)$ bits
- It is viewed as many set of associative mapping.

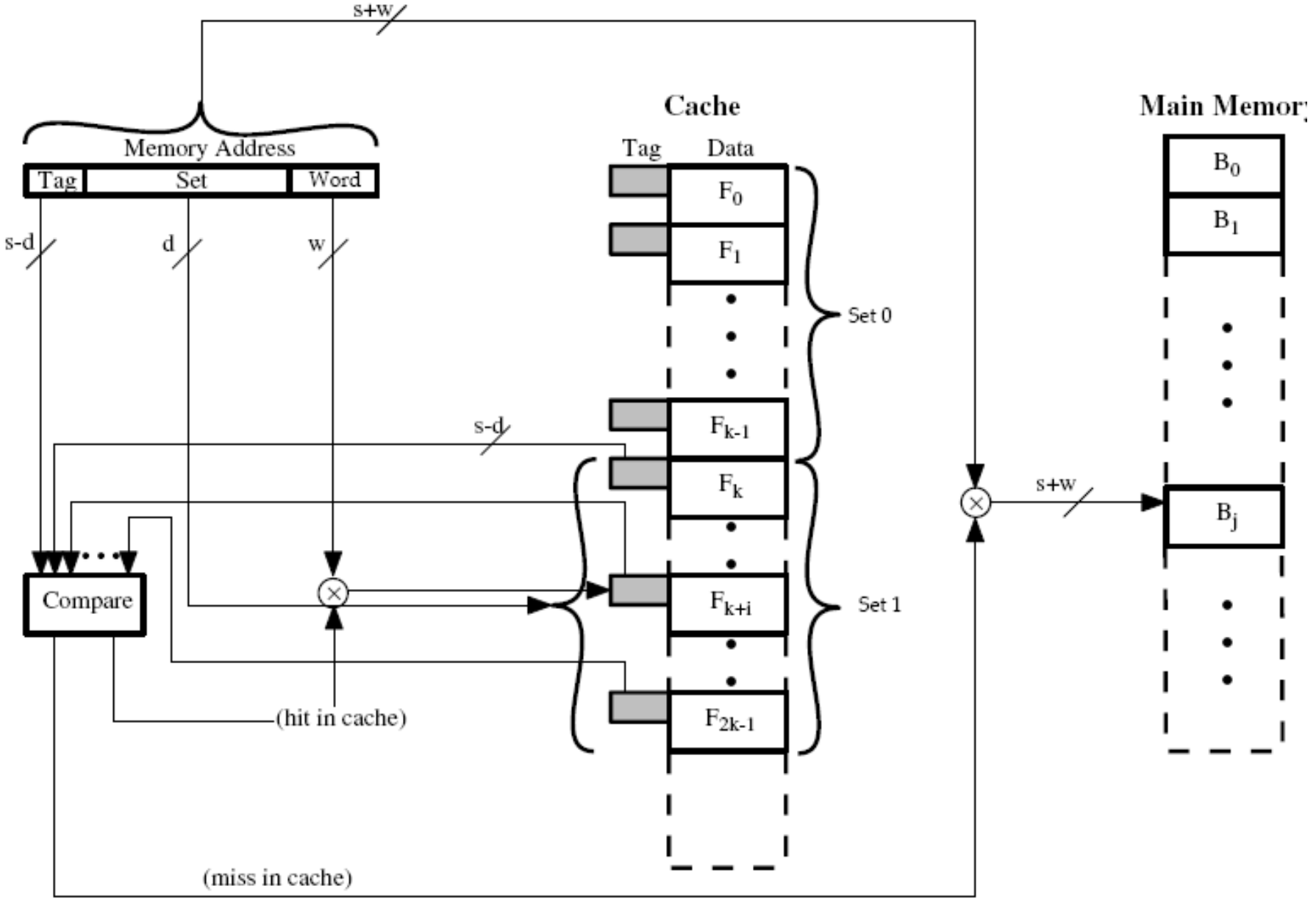
Set Associative Mapping

- Hybrid of Direct and Associative
  k = 1, this is direct mapping
  v = 1, this is associative mapping
- Each set contains a number of lines, basically the total number of lines in cache divided by the number of sets
- A given block maps to any line within its specified set – e.g. Block B can be in any line of set i.
- 2 lines per set is the most common organization.
  - Called 2-way associative mapping
  - A given block can be in one of 2 lines in only one specific set
  - Significant improvement over direct mapping

Set-Associative Block Placement



K-Way Set Associative Cache Organization



Previous direct mapping example- will be used for next mapping illustration

- 24 bit address lines
- 2 bit word identifier (4 byte block)
- 22 bit block identifier
- 8 bit tag
- 14 bit slot or line
- No two blocks in the same line have the same tag
- Check contents of cache by finding line and comparing tag

2 way set associative mapping

- Divides the 16K lines into 2 sets of 8k with 2 lines in each set
- This requires a 13 bit set number
- With 2 word bits, this leaves 9 bits for the tag
- Blocks beginning with the addresses 000000_{16}, 008000_{16}, 010000_{16}, 018000_{16}, 020000_{16}, 028000_{16}, etc. map to the same set, Set 0.
- Blocks beginning with the addresses 000004_{16}, 008004_{16}, 010004_{16}, 018004_{16}, 020004_{16}, 028004_{16}, etc. map to the same set, Set 1.

Tag 9 bits	Set 13 bits	Word 2 bits

Set Associative Mapping Address Structure

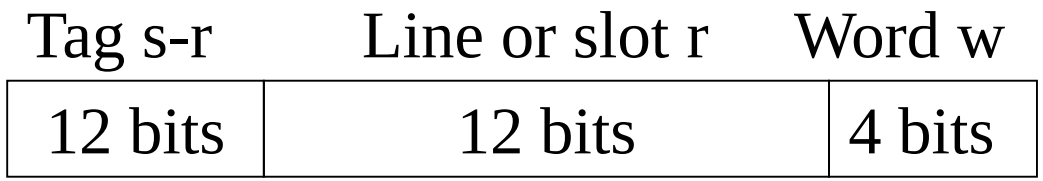
Tag 9 bits	Set 13 bits	Word 2 bits

- Note that there is one more bit in the tag than for this same example using direct mapping.
- Therefore, it is 2-way set associative
- Use set field to determine cache set to look into
- Compare tag field to see if we have a hit

Direct Mapping Exercise

What cache line number will the following addresses be stored at? what will the address range of each block they belong to? Consider a cache with 4K lines of 16 words to a block in a 256 Meg memory space. (how many address bits?)

a) $9ABCDEF_{16}$
b) 1234567_{16}



Cache line number: a) CDEh b) 456h

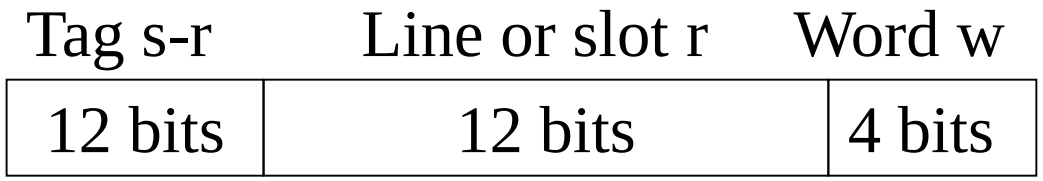
Address range: (a) 9ABCDE0h to 9ABCDEFh (b) 1234560h to 123456Fh

Direct Mapping Exercise

Show the cache memory with tag bits and line number (in binary) which will be occupied by the following addresses? Consider a cache with 4K lines of 16 words to a block in a 256 Meg memory space.

a) $8ABCDAF_{16}$
b) $81A3457_{16}$

How many lines do we have in total? What is the size of cache in bytes?



If next address is 81A3458H, will this be a hit or miss?

Ans: a) tag bits: 1000 1010 1011 corresponding to line no. 1100 1101 1010
b) tag bits: 1000 0001 1010 corresponding to line no. 0011 0100 0101

Direct Mapping Exercise

Assume that a portion of the cache showing tags and line number looks like the table below. Which of the following addresses are contained in the cache?

a) $438EE8_{16}$ miss

b) $F18EFF_{16}$ hit

c) $6B8EF3_{16}$ miss

d) $AD8EF3_{16}$ hit

Tag (binary)	Line number (binary)	Addresses wi/ block			
		00	01	10	11
0101 0011	1000 1110 1110 10				
1110 1101	1000 1110 1110 11				
1010 1101	1000 1110 1111 00				
0110 1011	1000 1110 1111 01				
1011 0101	1000 1110 1111 10				
1111 0001	1000 1110 1111 11				

Fully Associative Mapping Exercise

Assume that a portion of the cache with the tag values looks like the table below. Which of the following addresses are contained in the cache?

a) $438EE8_{16}$ miss

b) $F18EFF_{16}$ hit

c) $6B8EF3_{16}$ miss

d) $AD8EF3_{16}$ hit

Tag (binary)	Addresses wi/ block			
	00	01	10	11
0101 0011 1000 1110 1110 10				
1110 1101 1100 1001 1011 01				
1010 1101 1000 1110 1111 00				
0110 1011 1000 1110 1111 11				
1011 0101 0101 1001 0010 00				
1111 0001 1000 1110 1111 11				

Set Associative Mapping Exercise

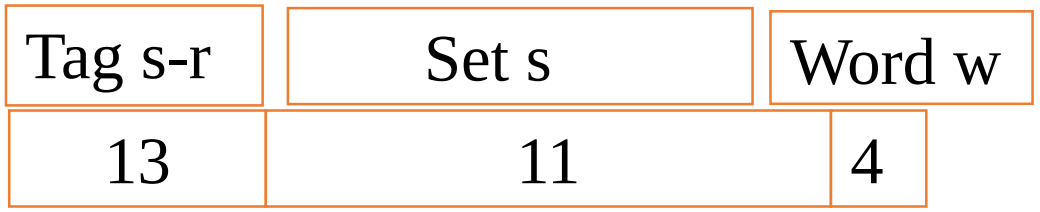
For each of the following addresses, answer the following questions based on a 2-way set associative cache with a total of 4K lines, each line containing 16 words, with the main memory of size 256 Meg memory space:

a) What is the cache set number the identified block be associated with?

b) What will the tag be?

c) What will the address range of the block in which the given address be located?

1. $9ABCDEF_{16}$
2. 1234567_{16}



Ans.: for 9ABCDEFH
set number is : 10011011110
tag: 1001101010111
9ABCDE0H to 9ABCDEFH
For 1234567H
set number is : 10001010110
tag: 0001001000110
1234560H to 123456FH