Computer Organization and Architecture (CPE343)

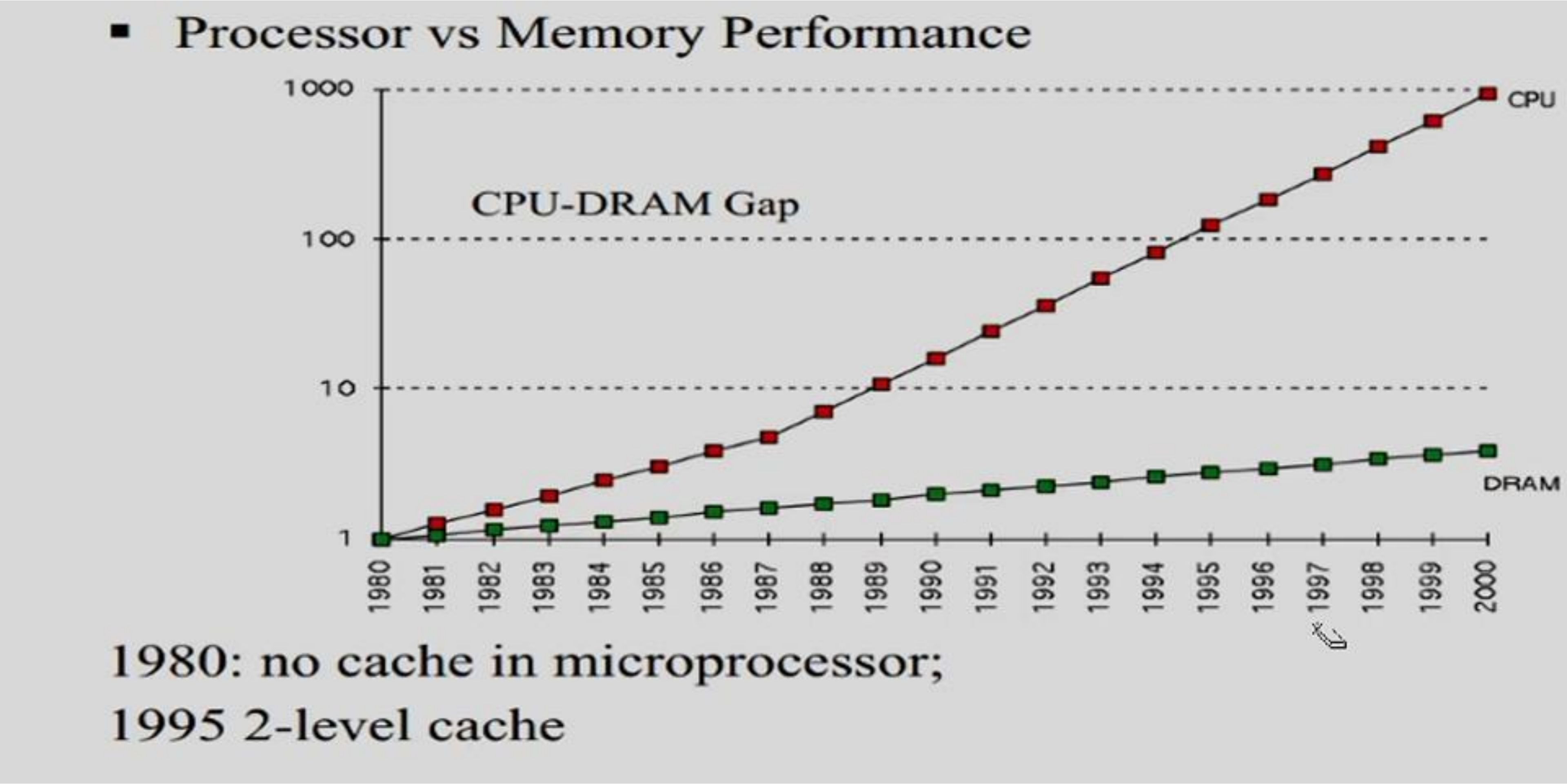
Dr. Muhammad Naeem Awais

naeem.awais@cuilahore.edu.pk

Cache Memory

- Processor Memory Performance Gap
- Relationship of Caches and Pipeline
- Memory Hierarchy
- CPU-Cache Interaction
- General Organization of Cache
- Addressing Cache
- Types of Cache
- Block Placement
- Block Identification
- Block Replacement
- Write Strategy

Processor Memory Performance Gap



- Processor vs Memory Performance

1980: no cache in microprocessor;
1995 2-level cache

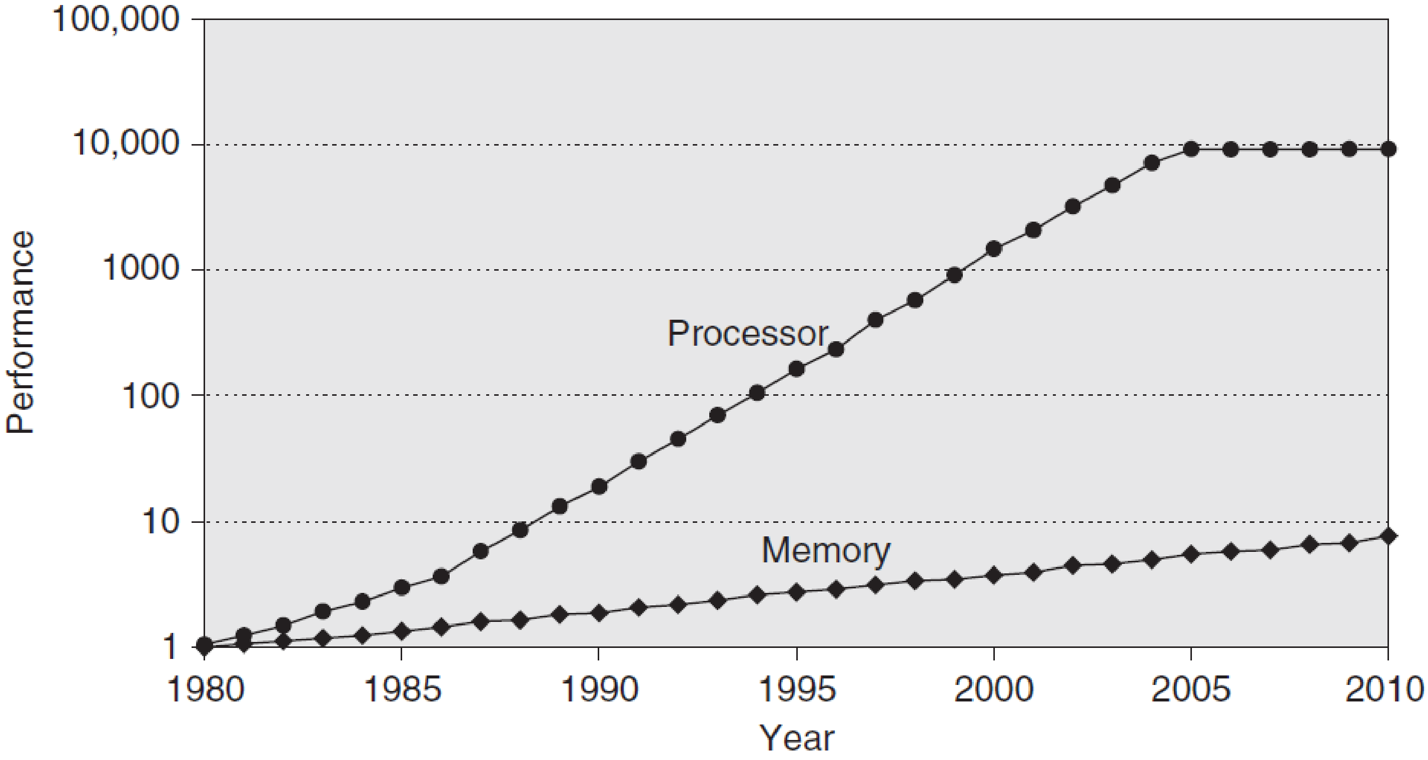
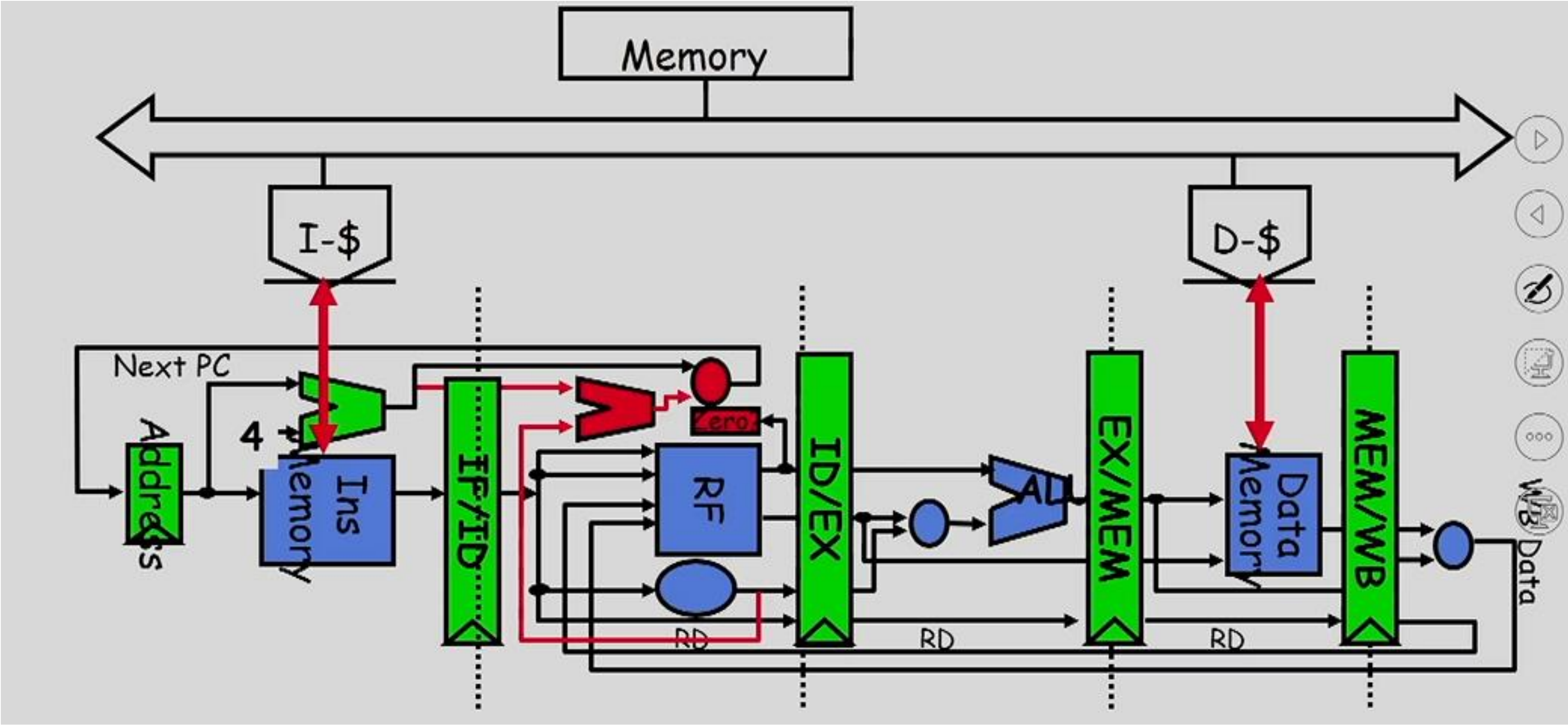


Figure 2.2 Starting with 1980 performance as a baseline, the gap in performance, measured as the difference in the time between processor memory requests (for a single processor or core) and the latency of a DRAM access, is plotted over time. Note that the vertical axis must be on a logarithmic scale to record the size of the processor–DRAM performance gap. The memory baseline is 64 KB DRAM in 1980, with a 1.07 per year performance improvement in latency (see Figure 2.13 on page 99). The processor line assumes a 1.25 improvement per year until 1986, a 1.52 improvement until 2000, a 1.20 improvement between 2000 and 2005, and no change in processor performance (on a per-core basis) between 2005 and 2010; see Figure 1.1 in Chapter 1.

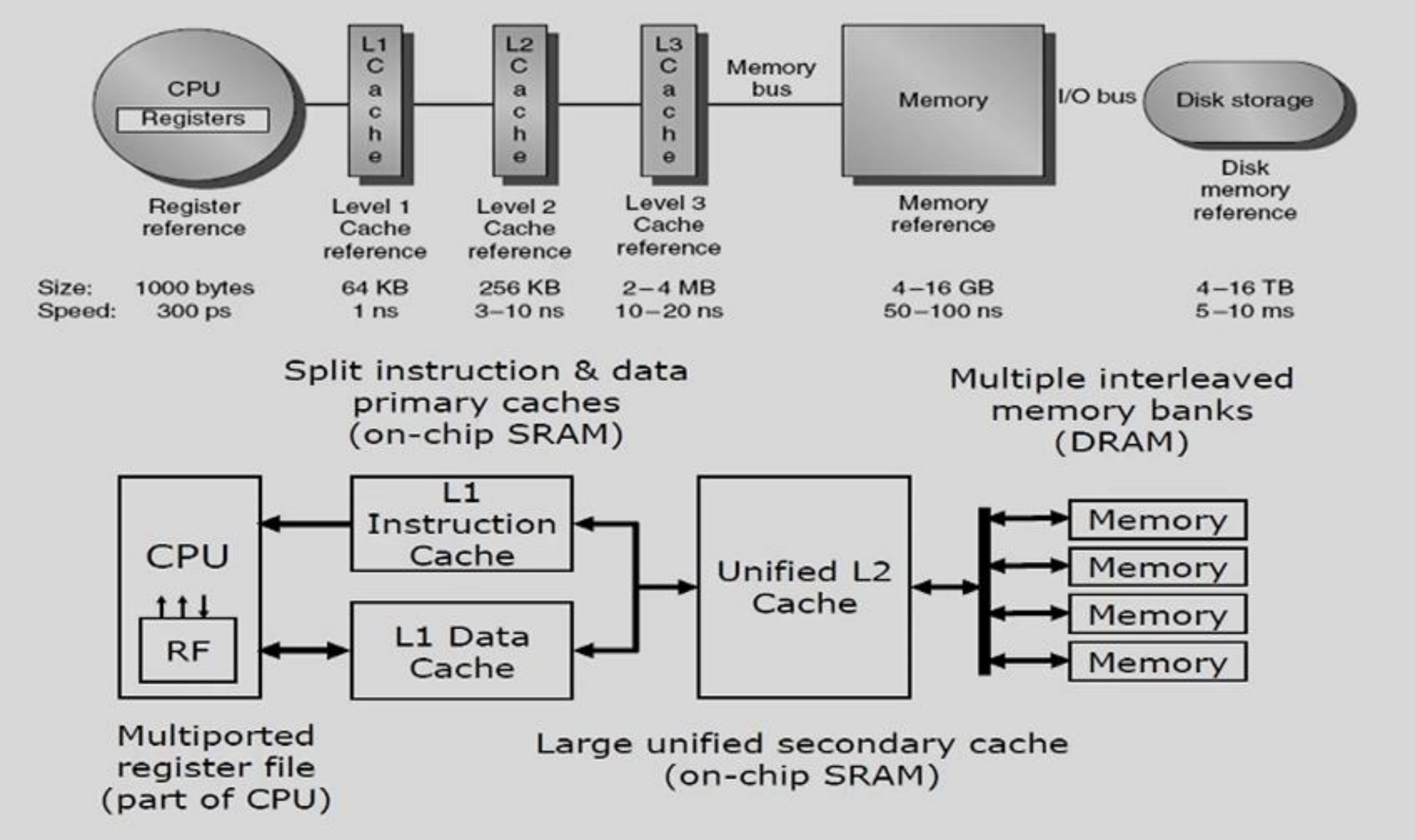
Relationship of Caches and Pipeline



Role of Memory

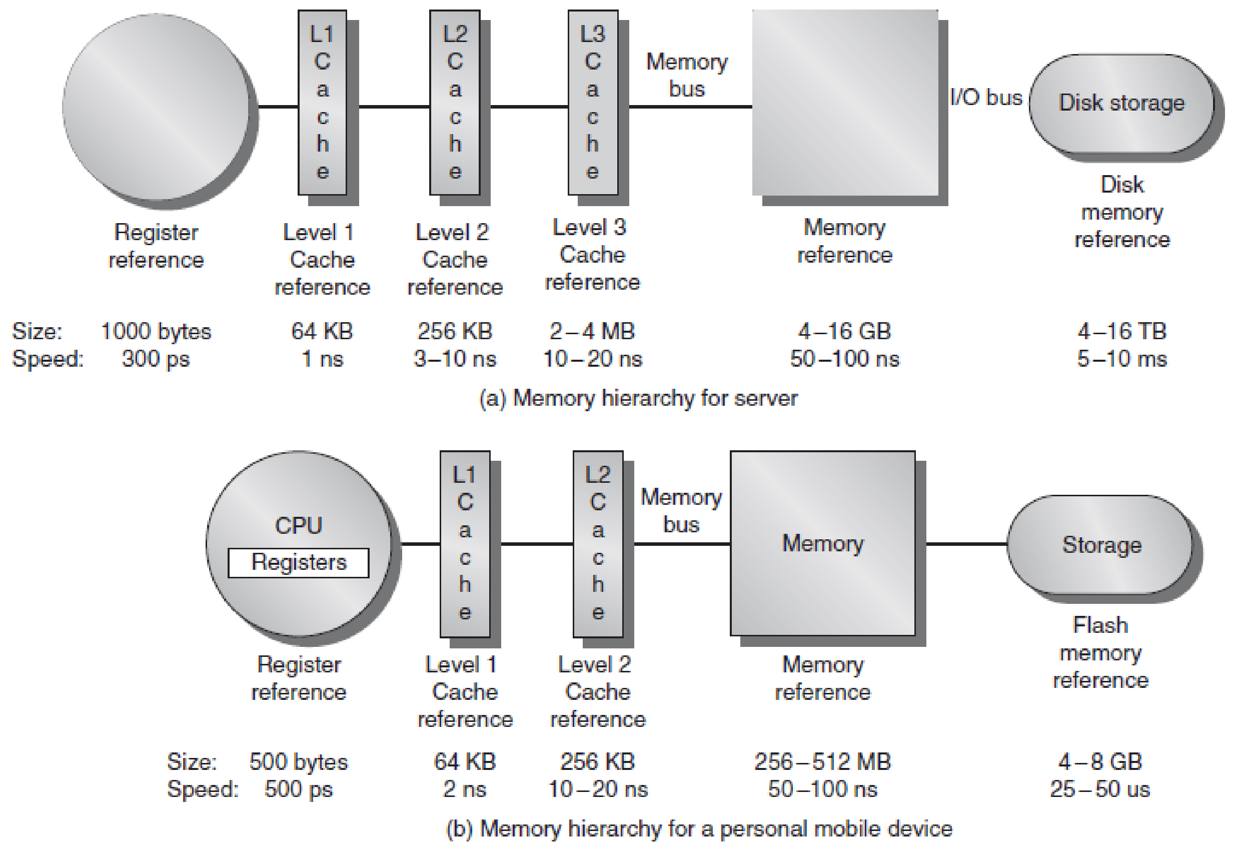
- Programmers want unlimited amount of fast memory
- Create the illusion of a very large and fast memory
- Implement the memory of a computer as a hierarchy
- Multiple levels of memory with different speeds and sizes
- Entire addressable memory space available in largest, slowest memory
- Keep the smaller and faster memories close to the processor and the slower, large memory below that

Memory Hierarchy


CPU
Registers
L1 Cache
L2 Cache
L3 Cache
Memory bus
Memory
I/O bus
Disk storage

	Register reference	Level 1 Cache reference	Level 2 Cache reference	Level 3 Cache reference	Memory reference	Disk memory reference
Size:	1000 bytes	64 KB	256 KB	2–4 MB	4–16 GB	4–16 TB
Speed:	300 ps	1 ns	3–10 ns	10–20 ns	50–100 ns	5–10 ms

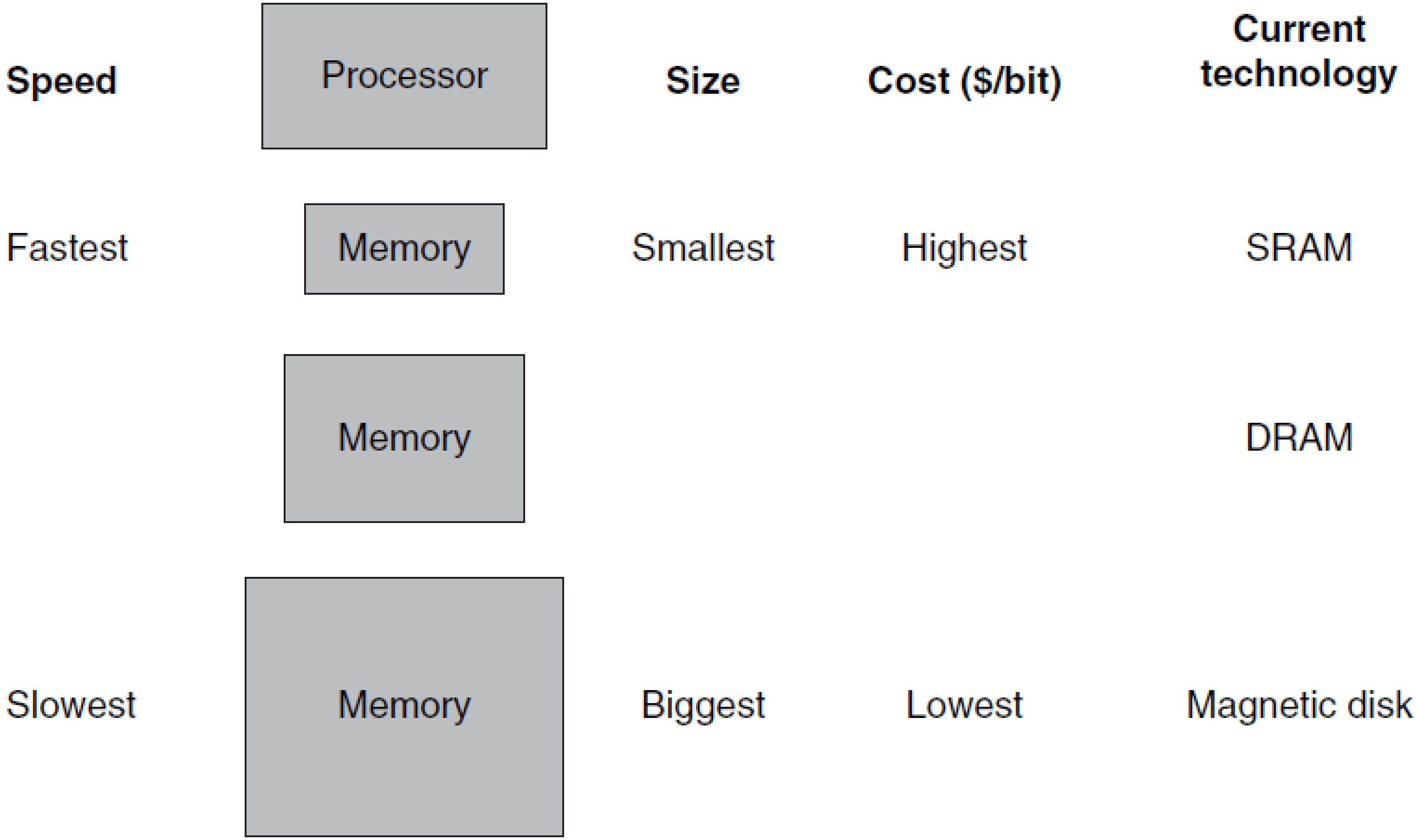
Split instruction & data primary caches (on-chip SRAM)
Multiple interleaved memory banks (DRAM)
CPU
RF
L1 Instruction Cache
L1 Data Cache
Unified L2 Cache
Memory
Memory
Memory
Memory
Multiported register file (part of CPU)
Large unified secondary cache (on-chip SRAM)



(a) Memory hierarchy for server

(b) Memory hierarchy for a personal mobile device

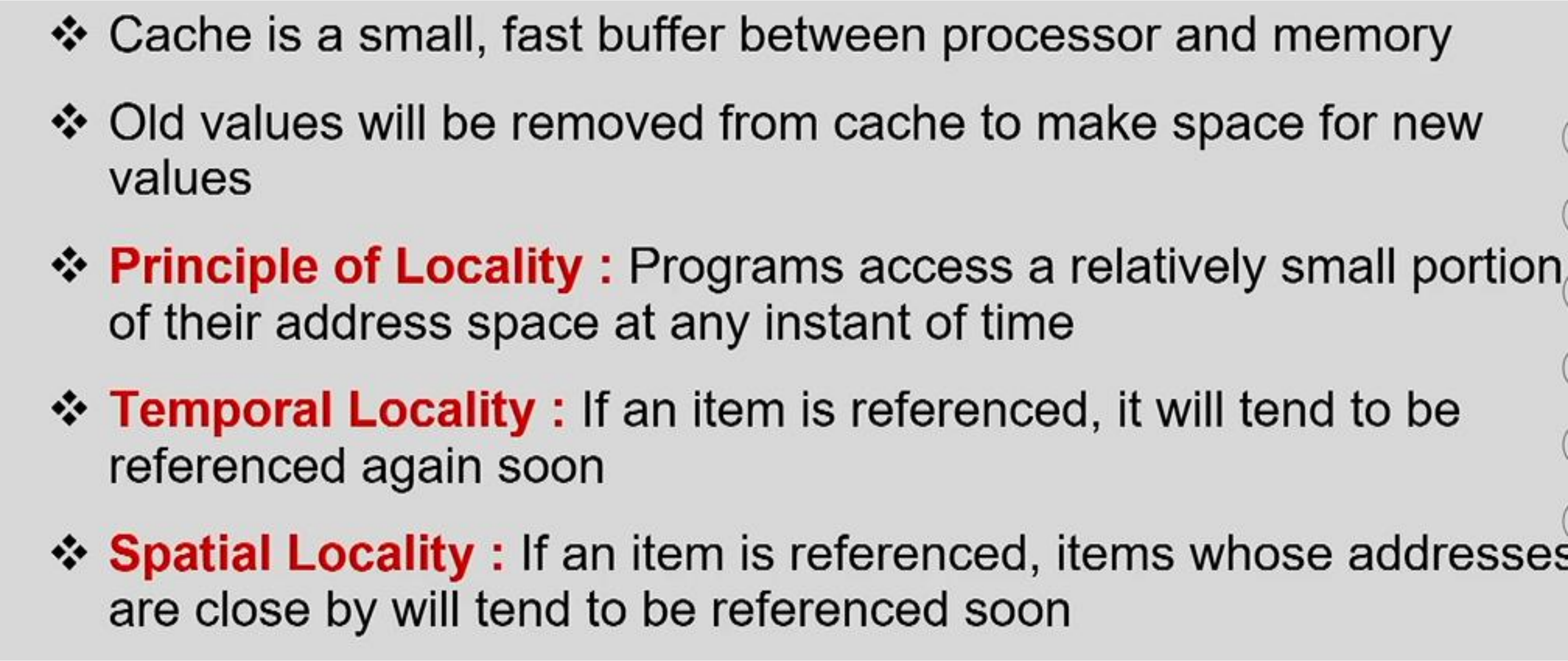
Figure 2.1 The levels in a typical memory hierarchy in a server computer shown on top (a) and in a personal mobile device (PMD) on the bottom (b). As we move farther away from the processor, the memory in the level below becomes slower and larger. Note that the time units change by a factor of 10^9—from picoseconds to milliseconds—and that the size units change by a factor of 10^{12}—from bytes to terabytes. The PMD has a slower clock rate and smaller caches and main memory. A key difference is that servers and desktops use disk storage as the lowest level in the hierarchy while PMDs use Flash, which is built from EEPROM technology.



Speed	Processor	Size	Cost ($/bit)	Current technology
Fastest	Memory	Smallest	Highest	SRAM
	Memory			DRAM
Slowest	Memory	Biggest	Lowest	Magnetic disk

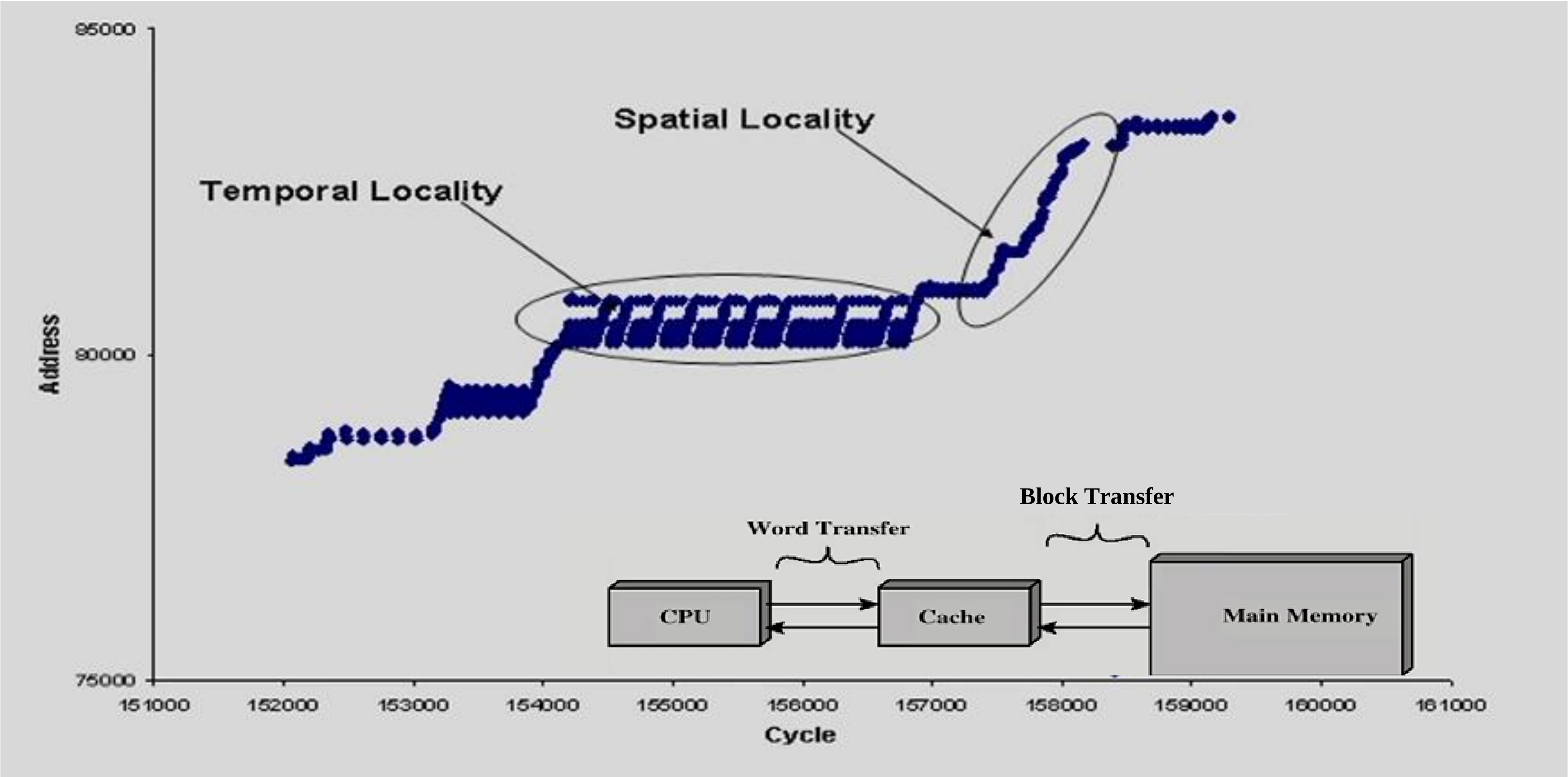
FIGURE 5.1 The basic structure of a memory hierarchy. By implementing the memory system as a hierarchy, the user has the illusion of a memory that is as large as the largest level of the hierarchy, but can be accessed as if it were all built from the fastest memory. Flash memory has replaced disks in many personal mobile devices, and may lead to a new level in the storage hierarchy for desktop and server computers; see Section 5.2.

Cache Memory-Introduction

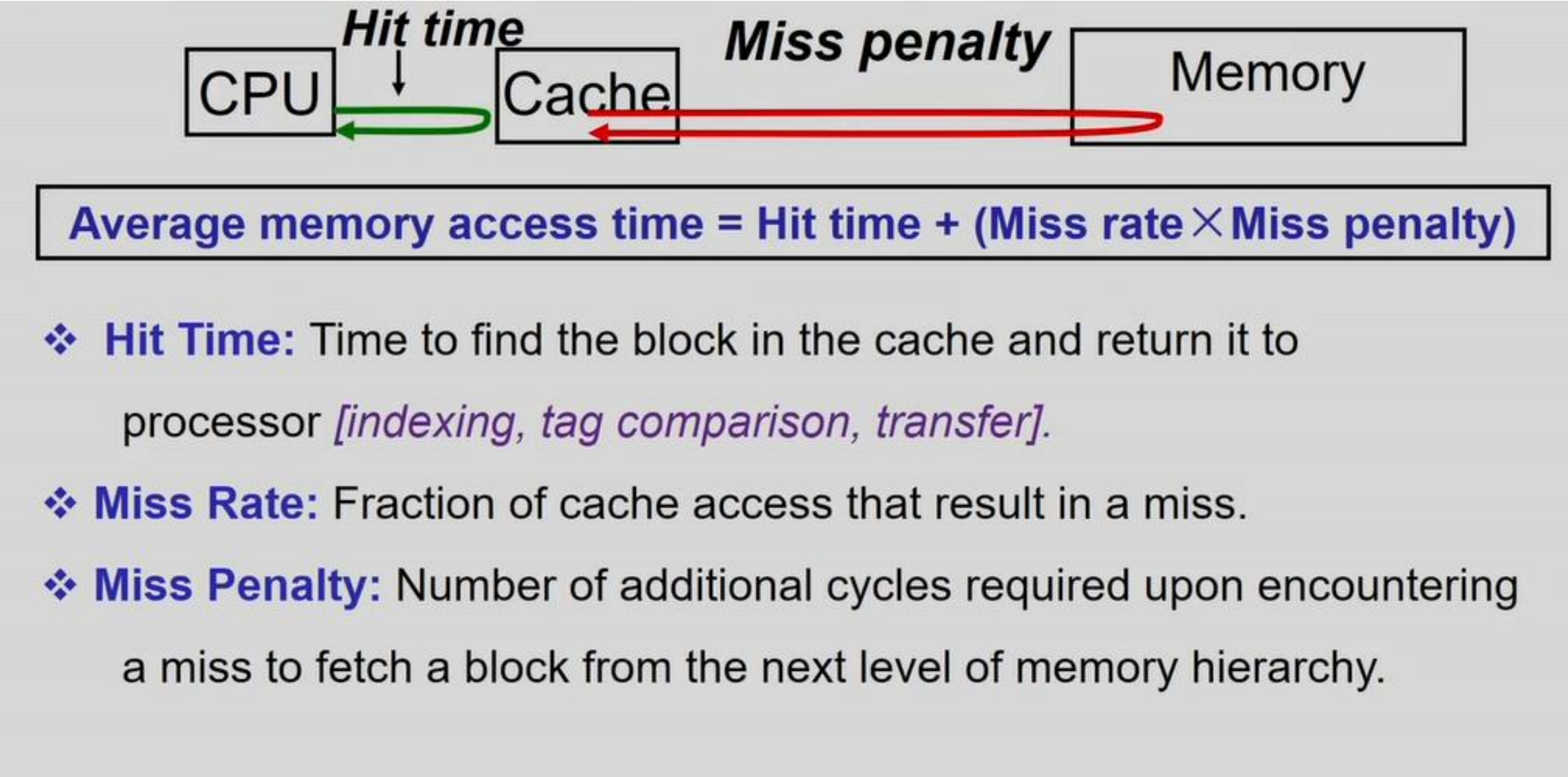


- Cache is a small, fast buffer between processor and memory
- Old values will be removed from cache to make space for new values
- **Principle of Locality :** Programs access a relatively small portion of their address space at any instant of time
- **Temporal Locality :** If an item is referenced, it will tend to be referenced again soon
- **Spatial Locality :** If an item is referenced, items whose addresses are close by will tend to be referenced soon

Access Patterns



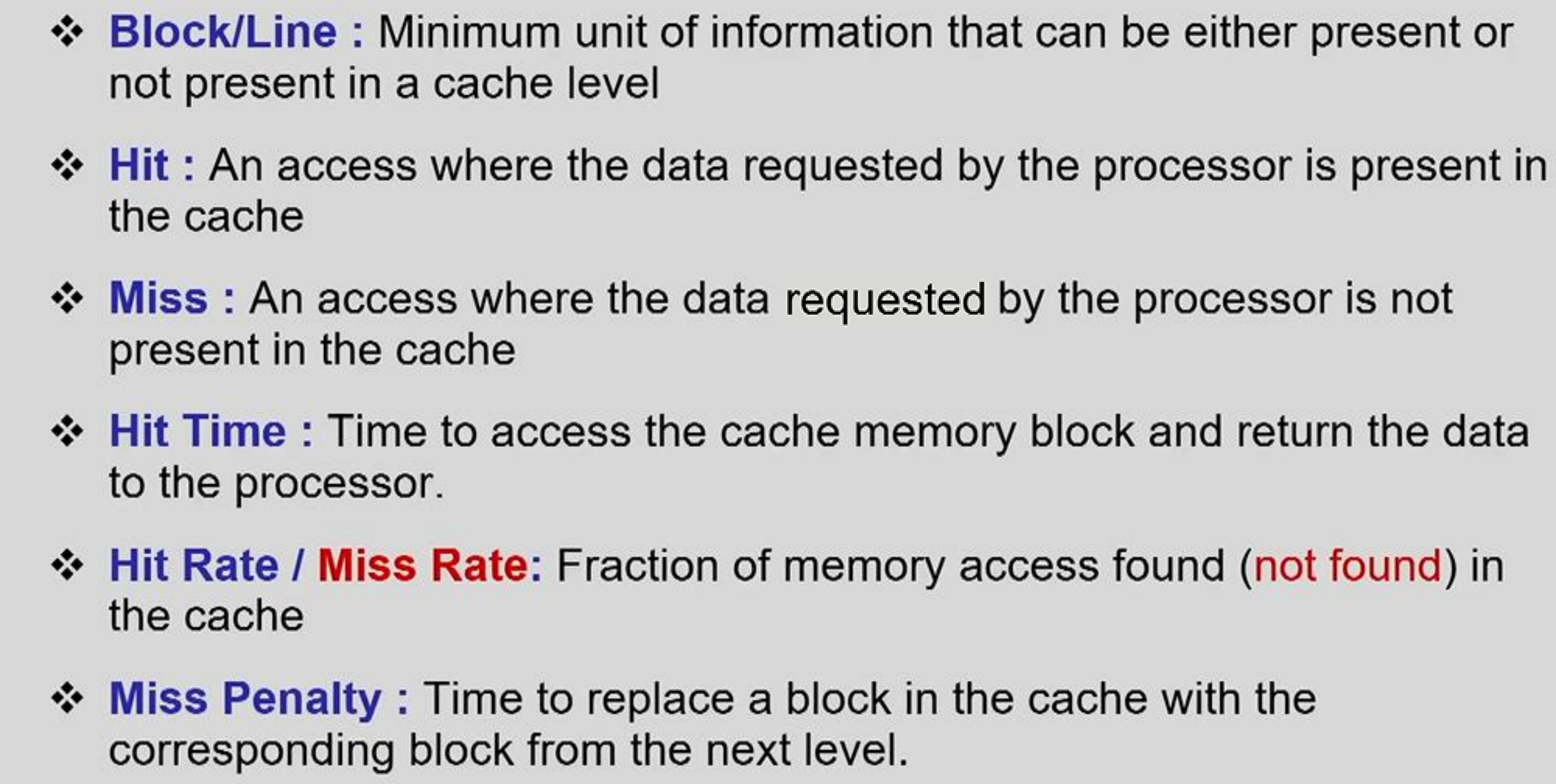
Accessing Cache Memory



Average memory access time = Hit time + (Miss rate × Miss penalty)

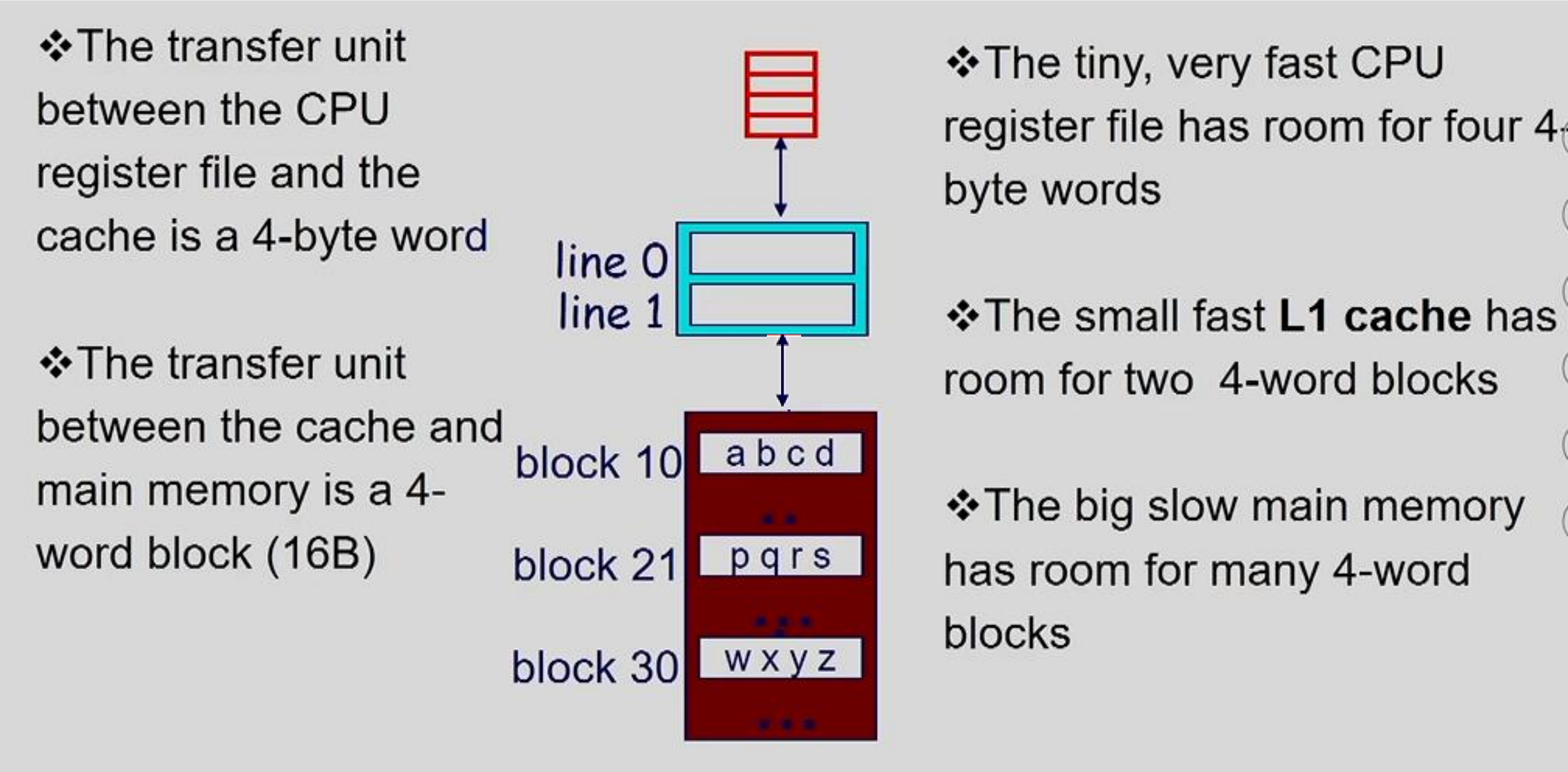
- **Hit Time:** Time to find the block in the cache and return it to processor *[indexing, tag comparison, transfer].*
- **Miss Rate:** Fraction of cache access that result in a miss.
- **Miss Penalty:** Number of additional cycles required upon encountering a miss to fetch a block from the next level of memory hierarchy.

Cache Fundamentals



- **Block/Line :** Minimum unit of information that can be either present or not present in a cache level
- **Hit :** An access where the data requested by the processor is present in the cache
- **Miss :** An access where the data requested by the processor is not present in the cache
- **Hit Time :** Time to access the cache memory block and return the data to the processor.
- **Hit Rate / Miss Rate:** Fraction of memory access found (not found) in the cache
- **Miss Penalty :** Time to replace a block in the cache with the corresponding block from the next level.

CPU-Cache Interaction



❖The transfer unit between the CPU register file and the cache is a 4-byte word

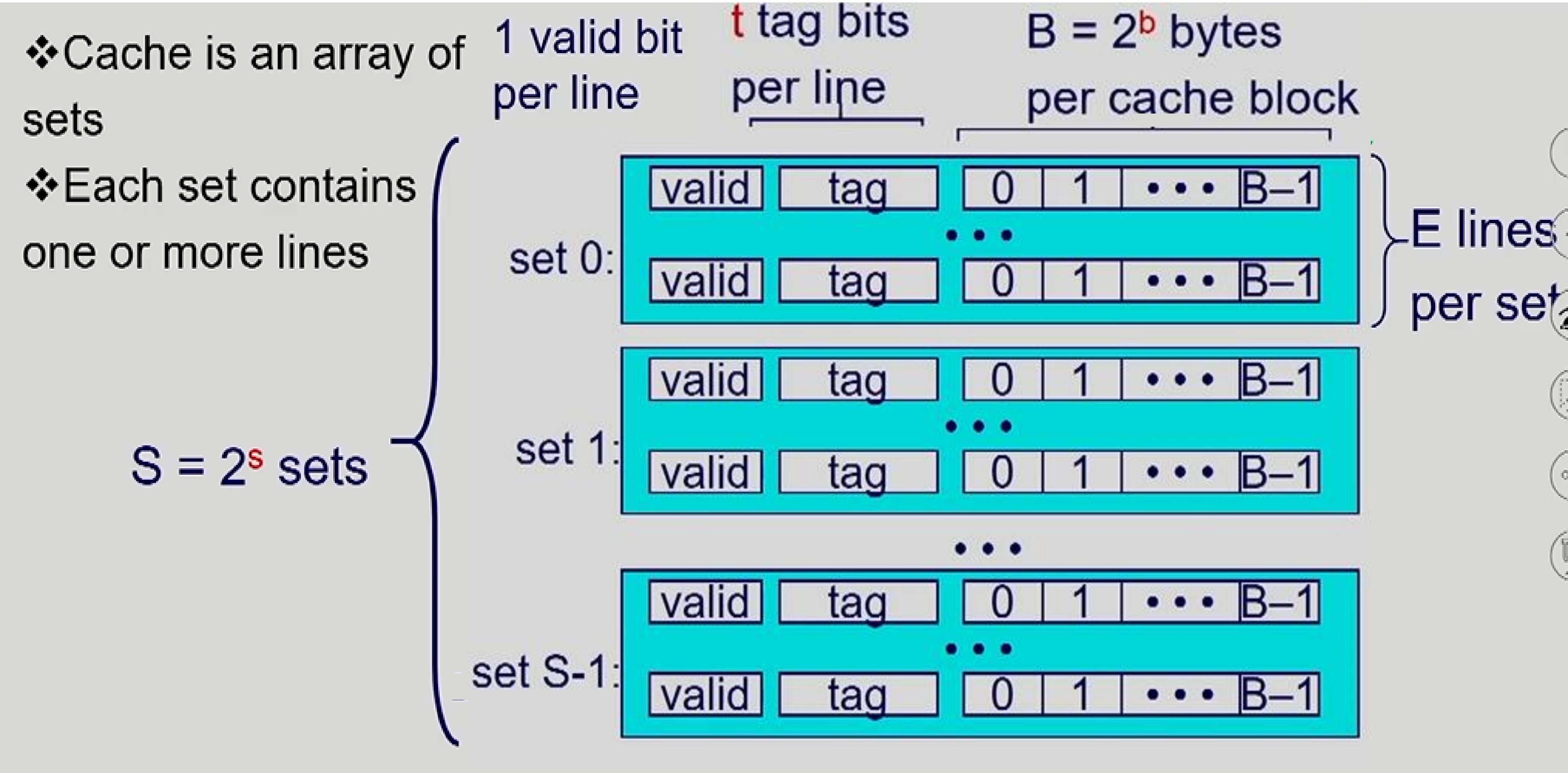
❖The transfer unit between the cache and main memory is a 4-word block (16B)

❖The tiny, very fast CPU register file has room for four 4-byte words

❖The small fast **L1 cache** has room for two 4-word blocks

❖The big slow main memory has room for many 4-word blocks

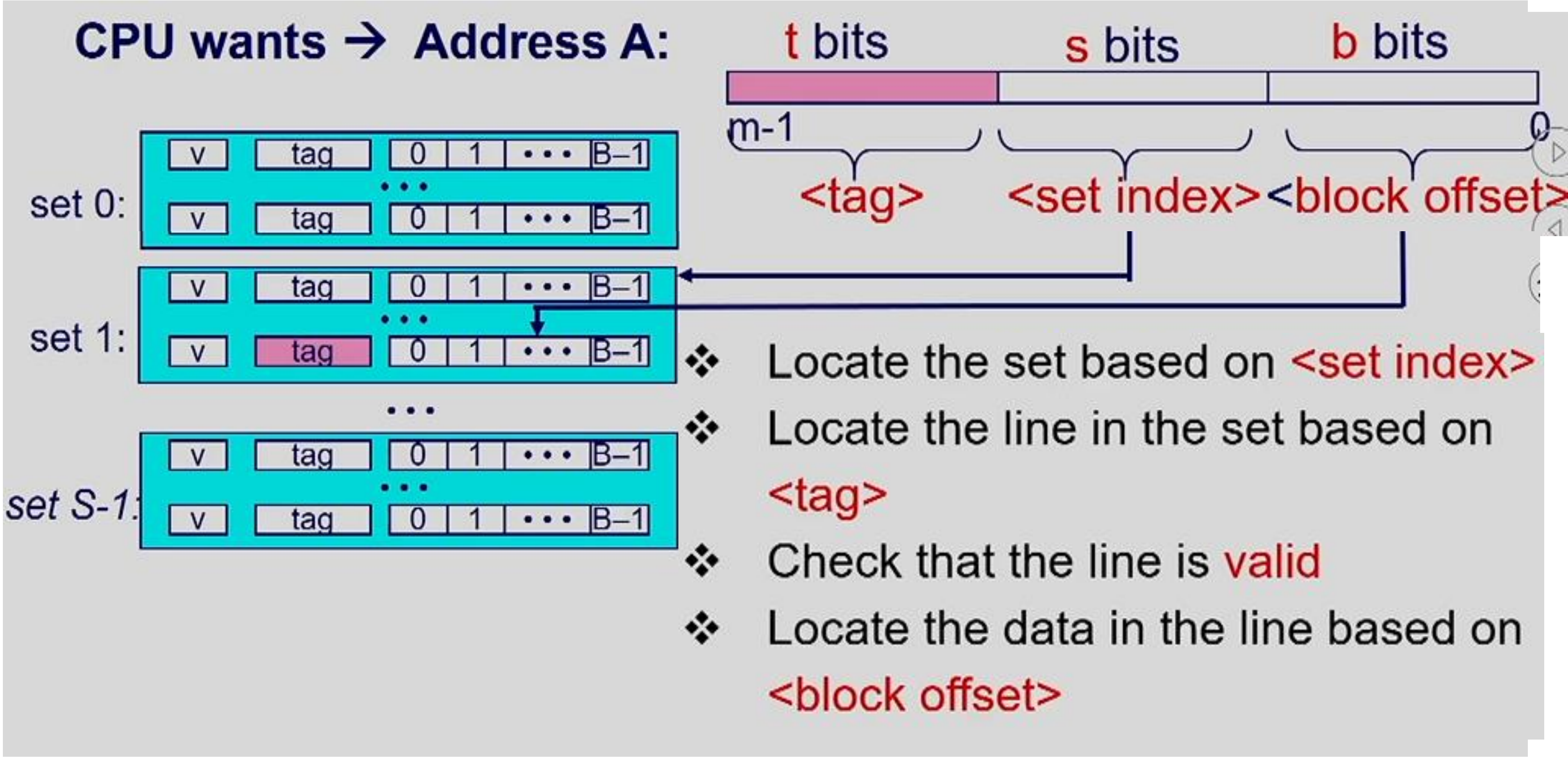
General Organization of Cache



- ❖Cache is an array of sets
- ❖Each set contains one or more lines

Cache size: C = B x E x S data bytes

Addressing Cache

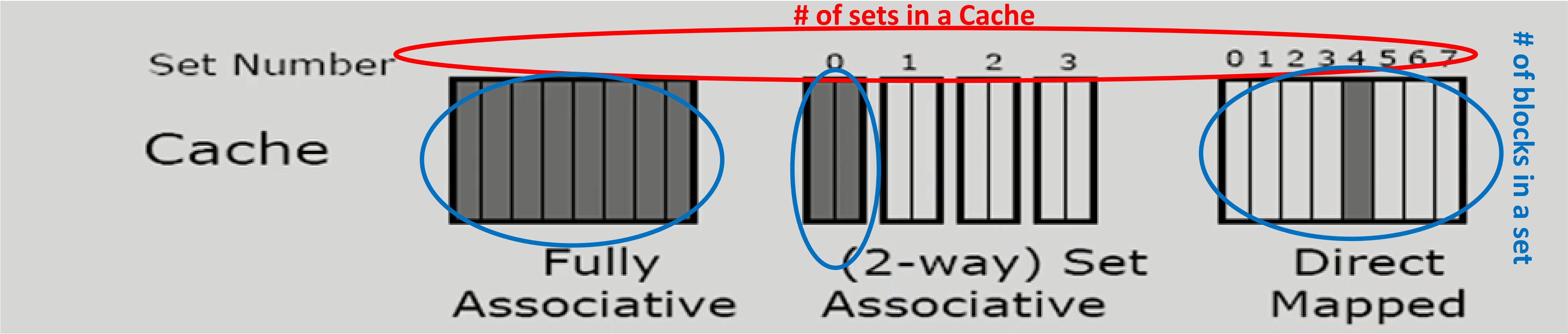


- Locate the set based on <set index>
- Locate the line in the set based on <tag>
- Check that the line is valid
- Locate the data in the line based on <block offset>

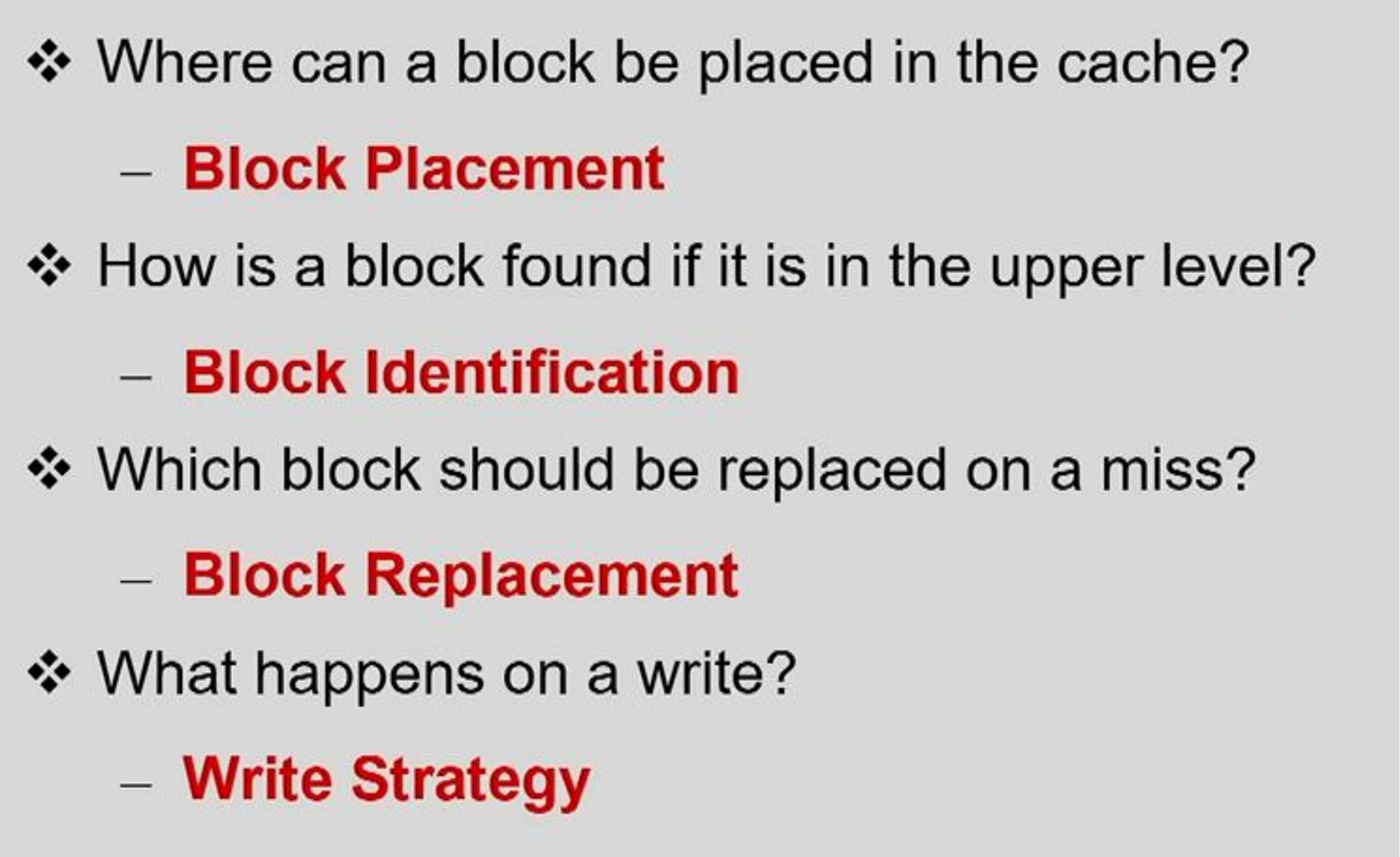
Types of Cache

- Direct Mapped Cache
- Set Associative Cache
- Fully Associative Cache

Types of Cache

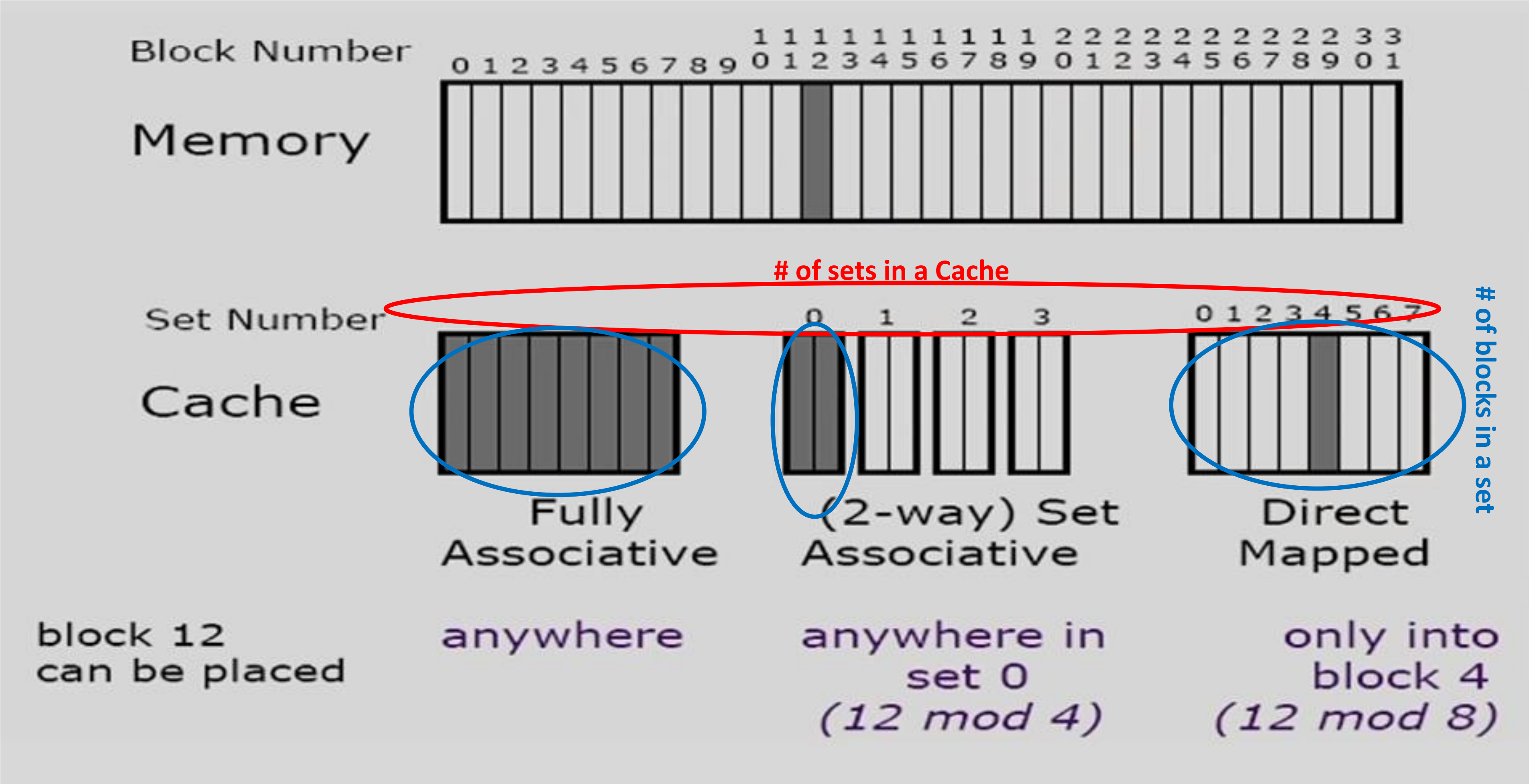


Four Cache Memory Design Choices



- Where can a block be placed in the cache?
  - **Block Placement**
- How is a block found if it is in the upper level?
  - **Block Identification**
- Which block should be replaced on a miss?
  - **Block Replacement**
- What happens on a write?
  - **Write Strategy**

Block Placement



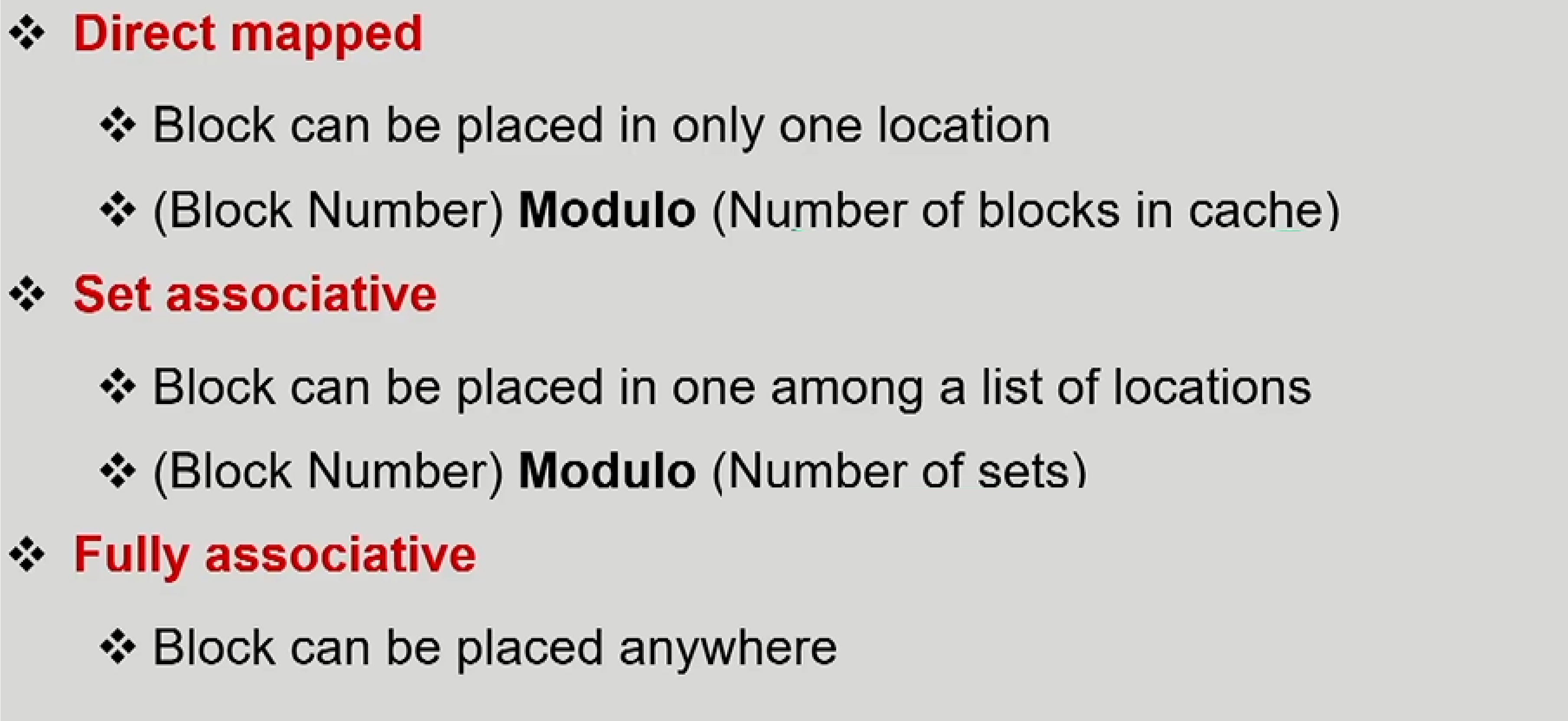
block 12 can be placed

anywhere

anywhere in set 0 (12 mod 4)

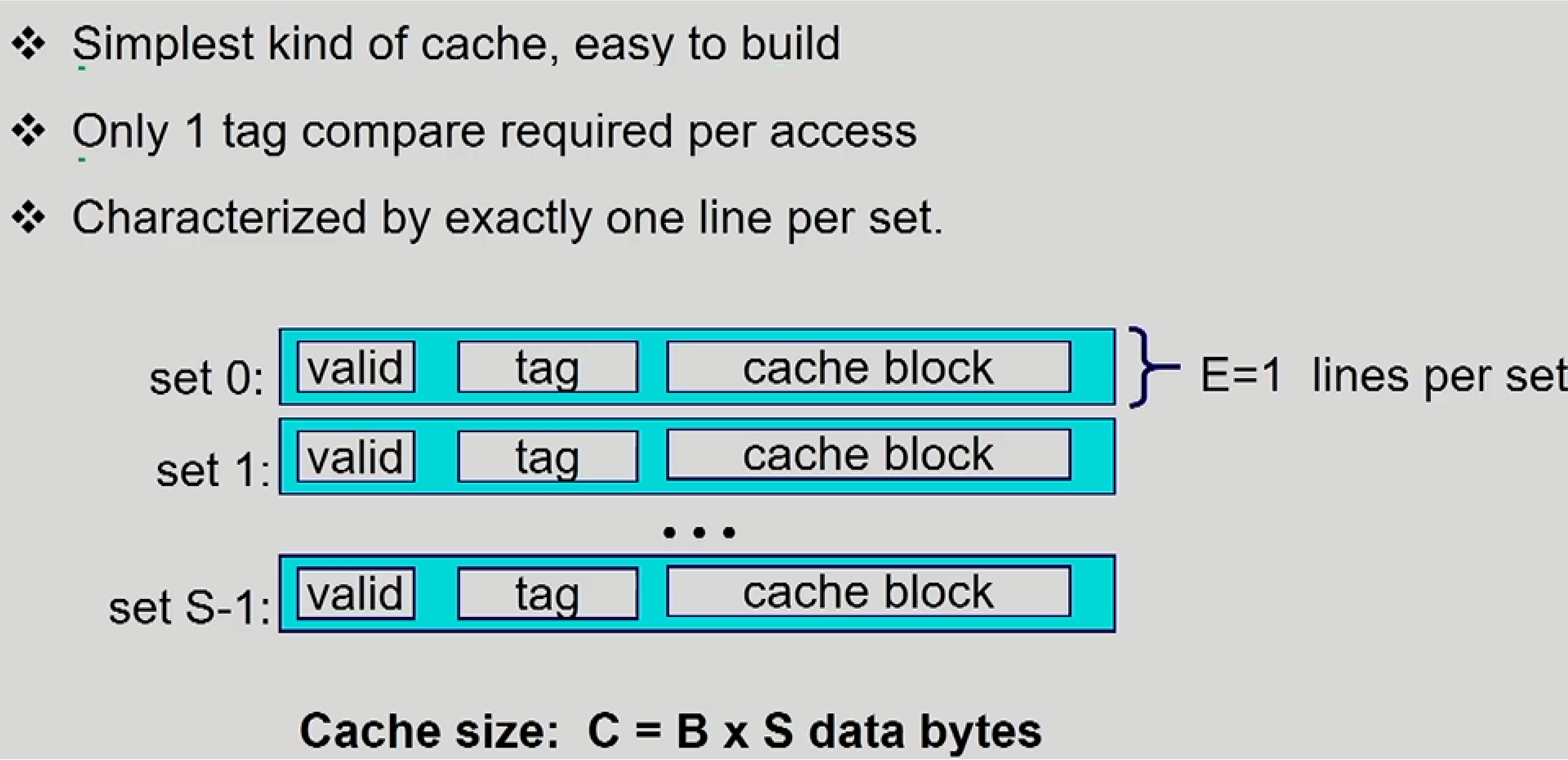
only into block 4 (12 mod 8)

Cache Mapping/Block Placement



- **Direct mapped**
  - Block can be placed in only one location
  - (Block Number) **Modulo** (Number of blocks in cache)
- **Set associative**
  - Block can be placed in one among a list of locations
  - (Block Number) **Modulo** (Number of sets)
- **Fully associative**
  - Block can be placed anywhere

Direct-Mapped Cache



- Simplest kind of cache, easy to build
- Only 1 tag compare required per access
- Characterized by exactly one line per set.

set 0: valid | tag | cache block } E=1 lines per set

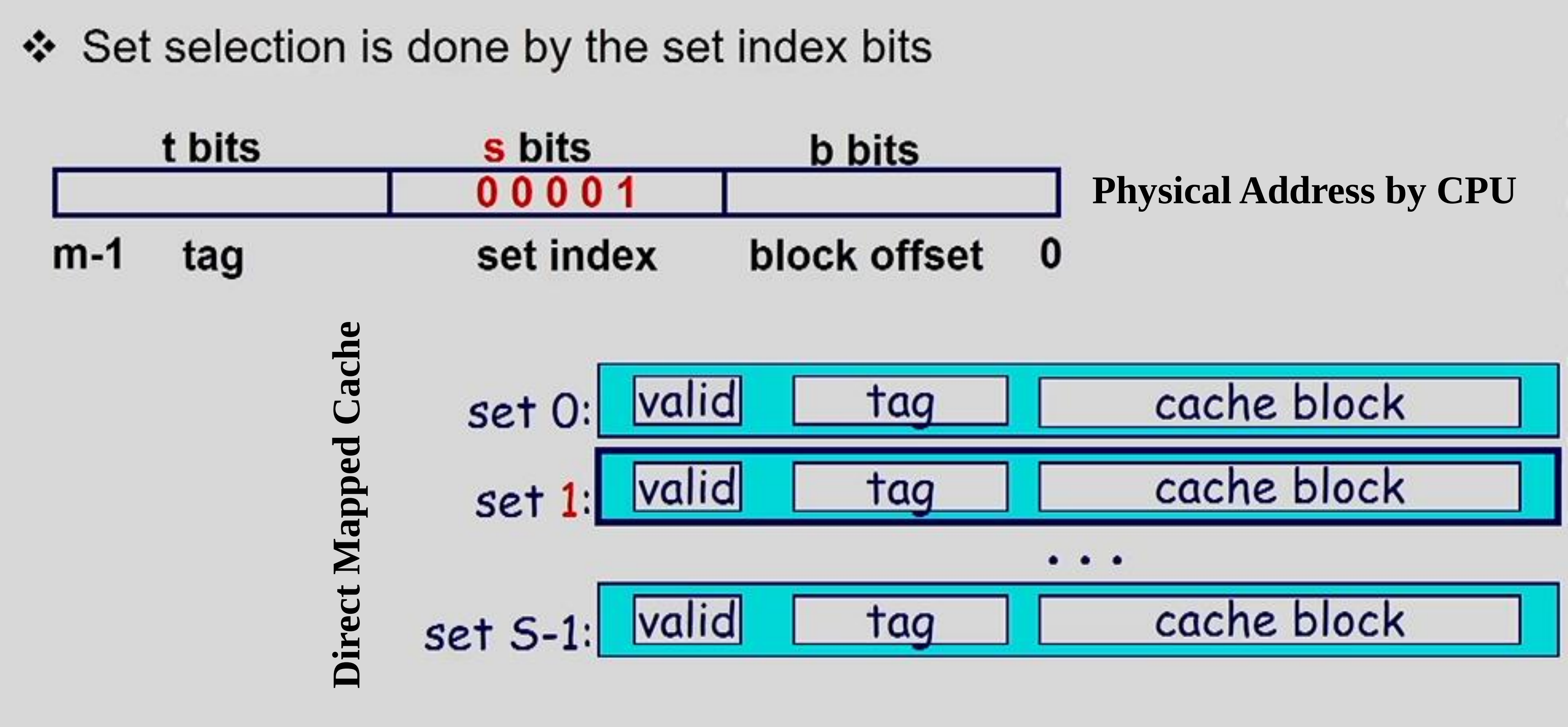
set 1: valid | tag | cache block

• • •

set S-1: valid | tag | cache block

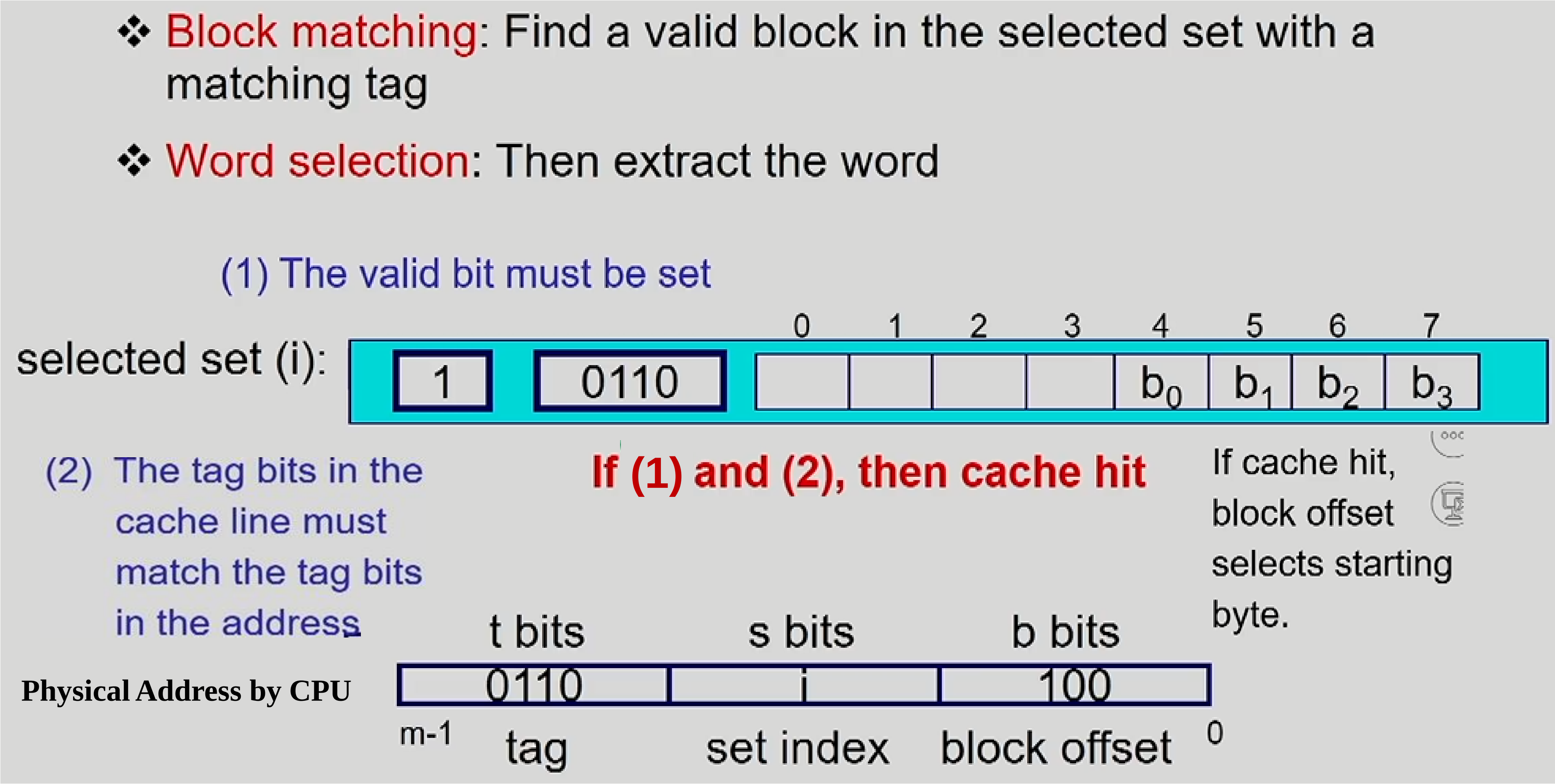
Cache size: C = B x S data bytes

Accessing Direct-Mapped Caches



- Set selection is done by the set index bits

Accessing Direct-Mapped Caches



- Block matching: Find a valid block in the selected set with a matching tag
- Word selection: Then extract the word

(1) The valid bit must be set

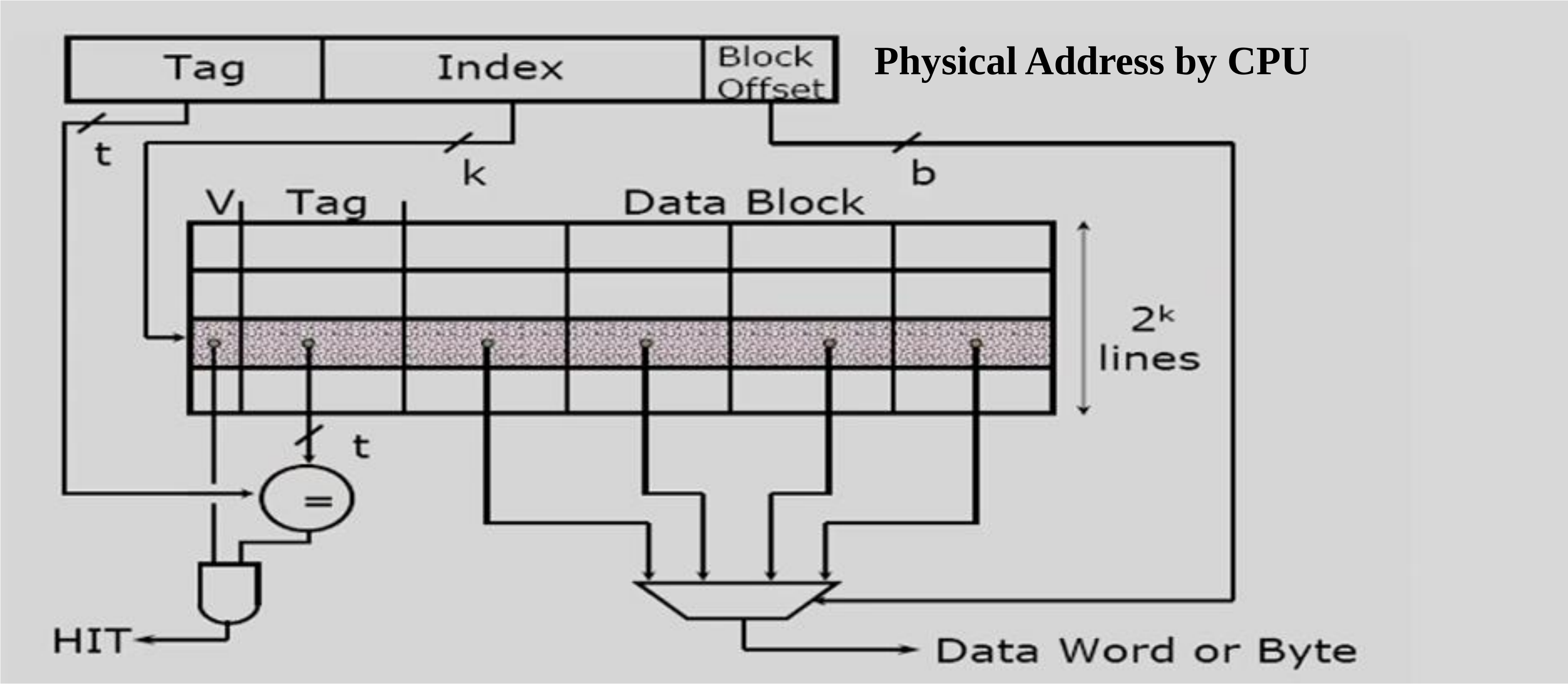
(2) The tag bits in the cache line must match the tag bits in the address

If (1) and (2), then cache hit

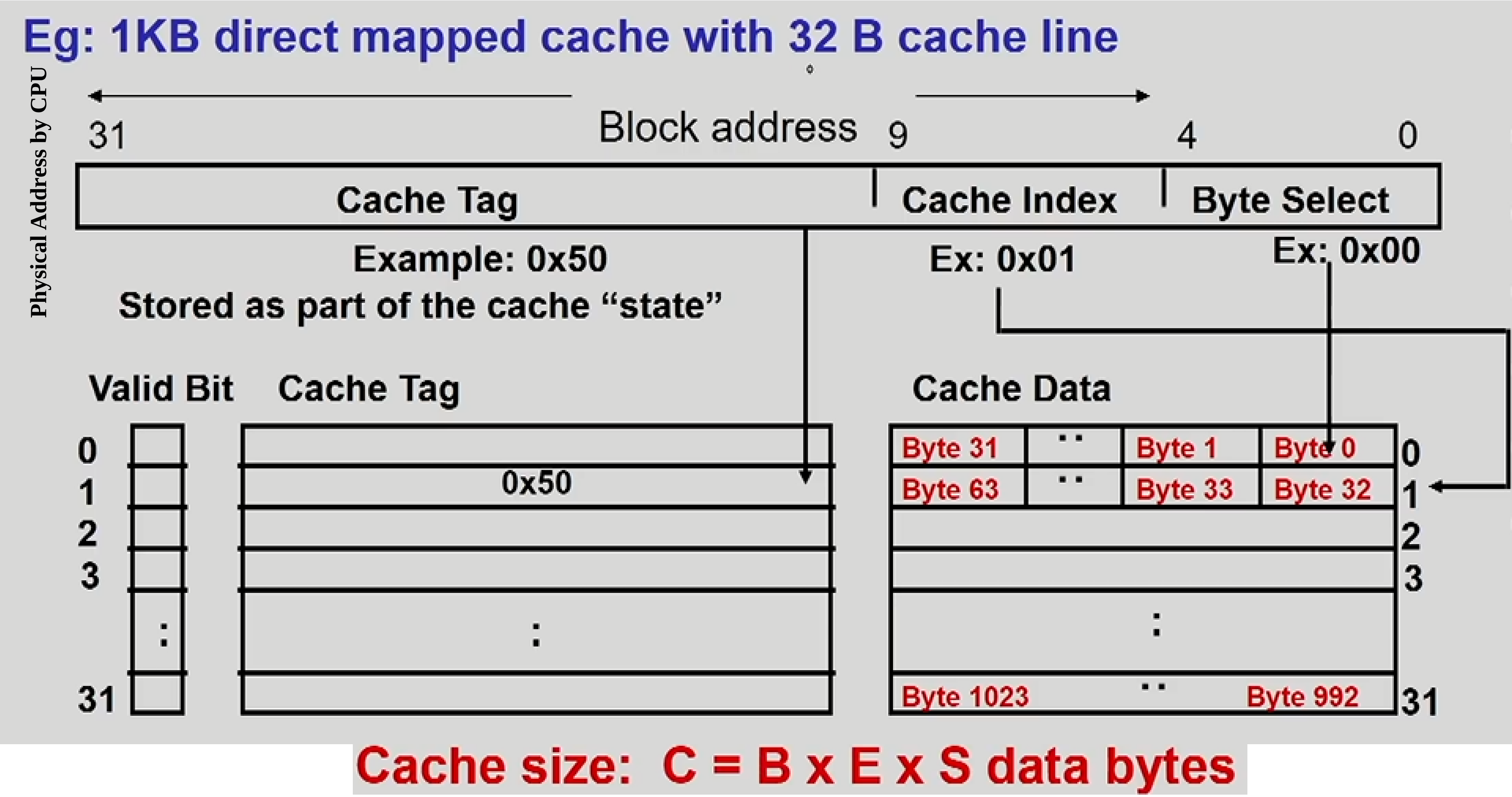
If cache hit, block offset selects starting byte.

Physical Address by CPU

Block Identification-Direct Mapped



Direct Mapped Cache



Eg: 1KB direct mapped cache with 32 B cache line

Physical Address by CPU

Block address

31 9 4 0

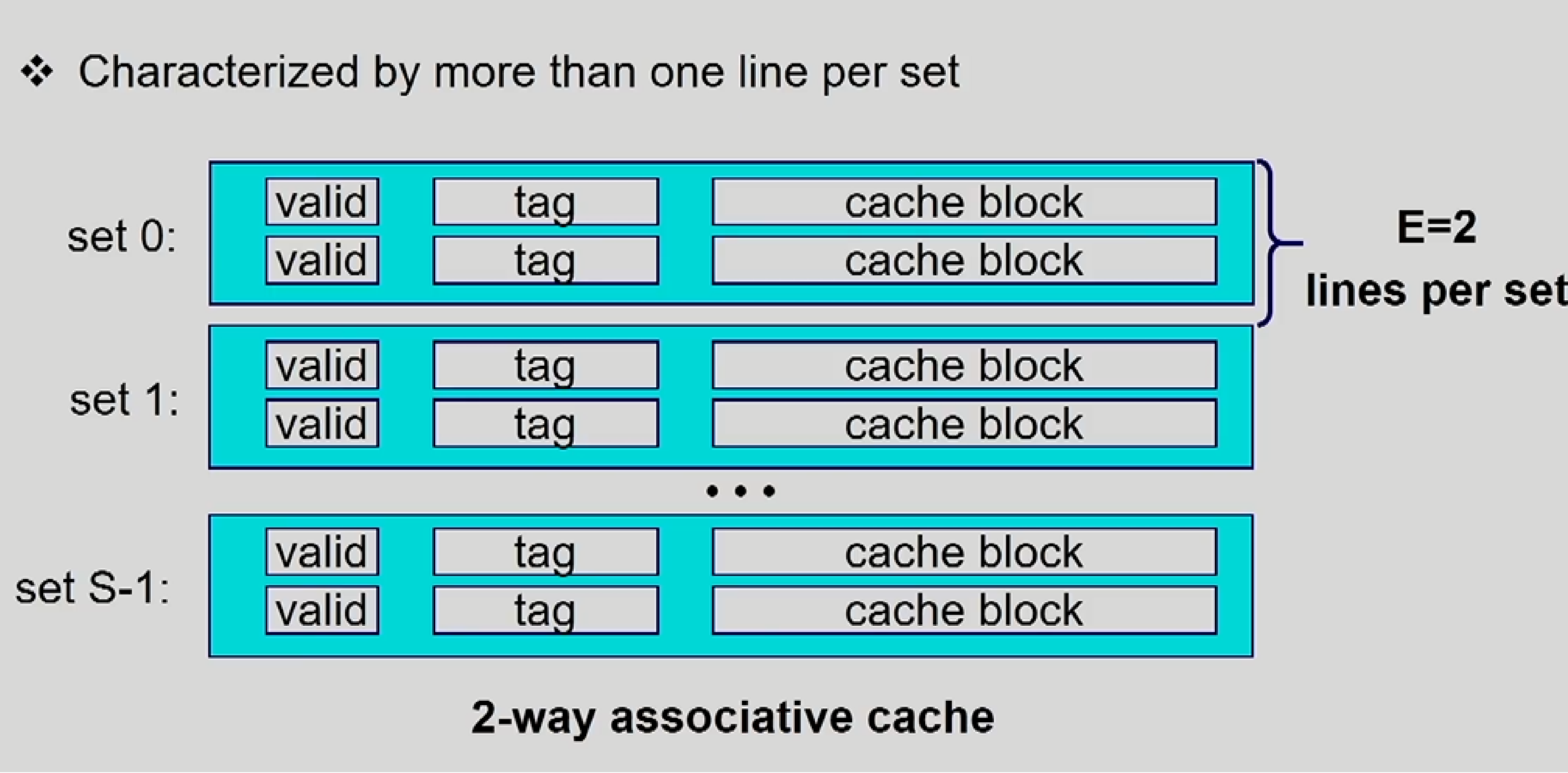
Cache Tag	Cache Index	Byte Select
Example: 0x50	Ex: 0x01	Ex: 0x00

Stored as part of the cache "state"

	Valid Bit	Cache Tag	Cache Data				
0			Byte 31	..	Byte 1	Byte 0	0
1		0x50	Byte 63	..	Byte 33	Byte 32	1
2							2
3							3
:	:	:	:				
31			Byte 1023	..		Byte 992	31

Cache size: C = B x E x S data bytes

Set Associative Cache



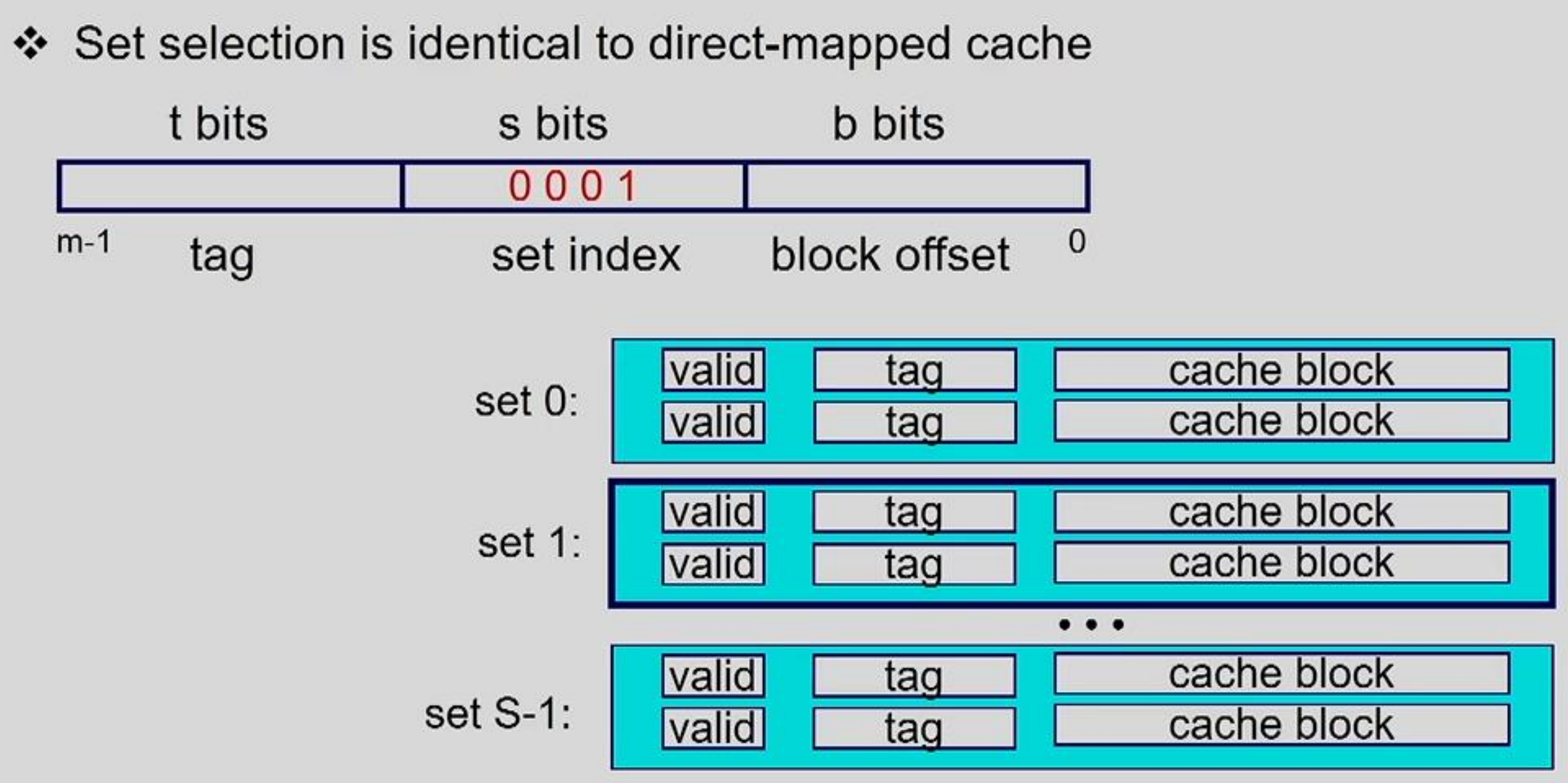
- Characterized by more than one line per set

set 0:	valid	tag	cache block
	valid	tag	cache block
set 1:	valid	tag	cache block
	valid	tag	cache block
• • •			
set S-1:	valid	tag	cache block
	valid	tag	cache block

E=2
lines per set

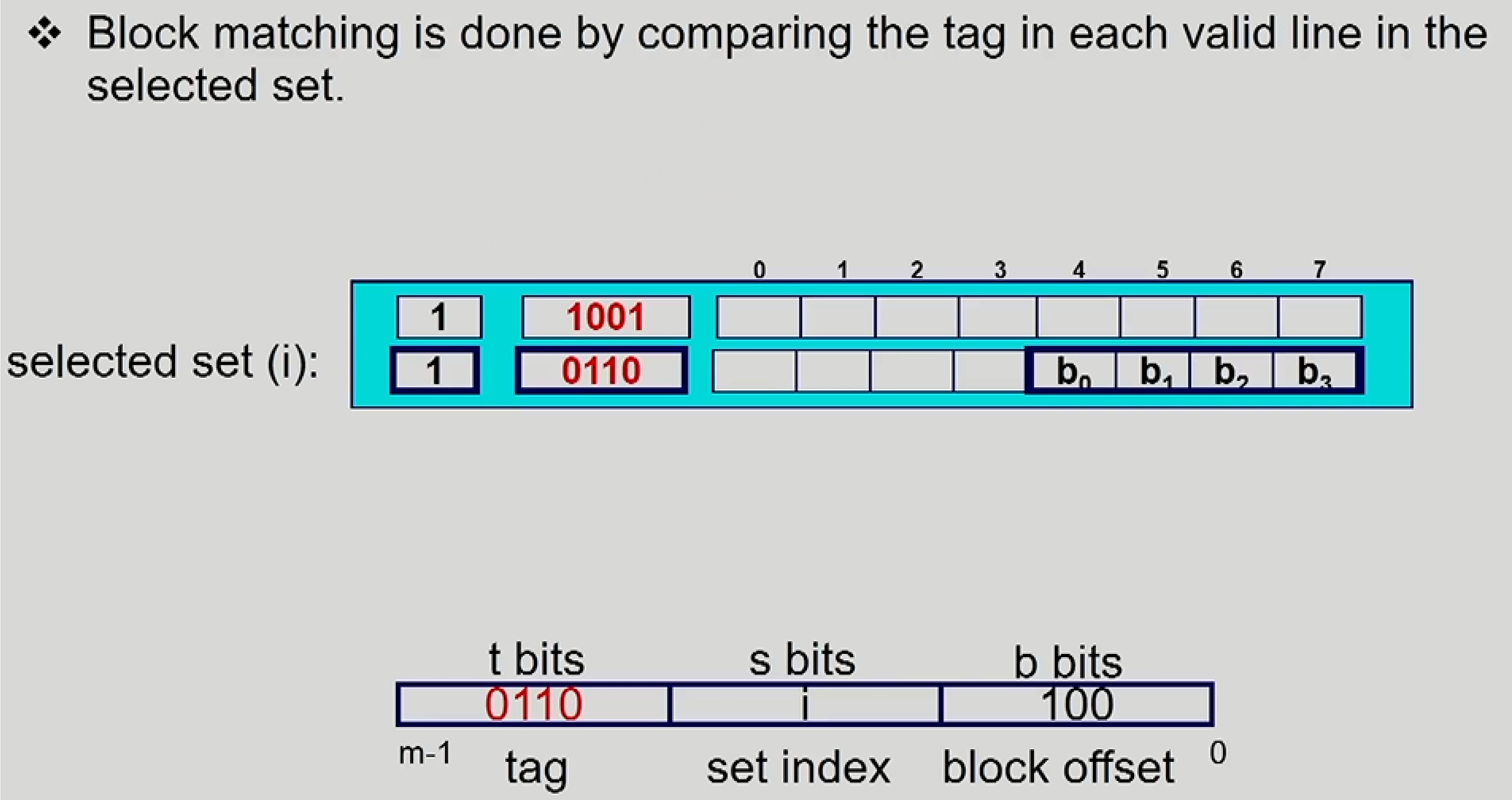
2-way associative cache

Accessing Set Associative Caches



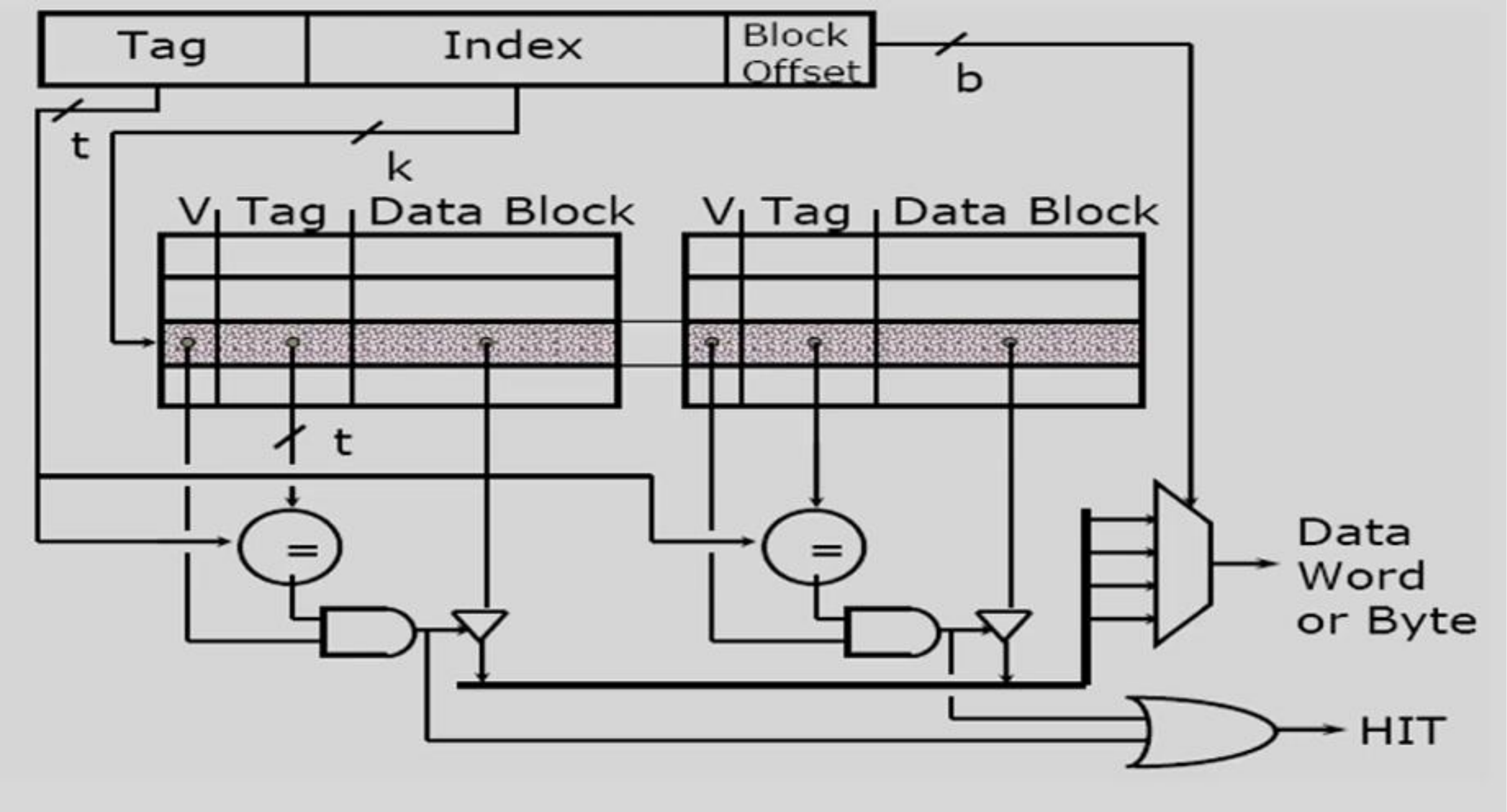
- Set selection is identical to direct-mapped cache

Accessing Set Associative Caches



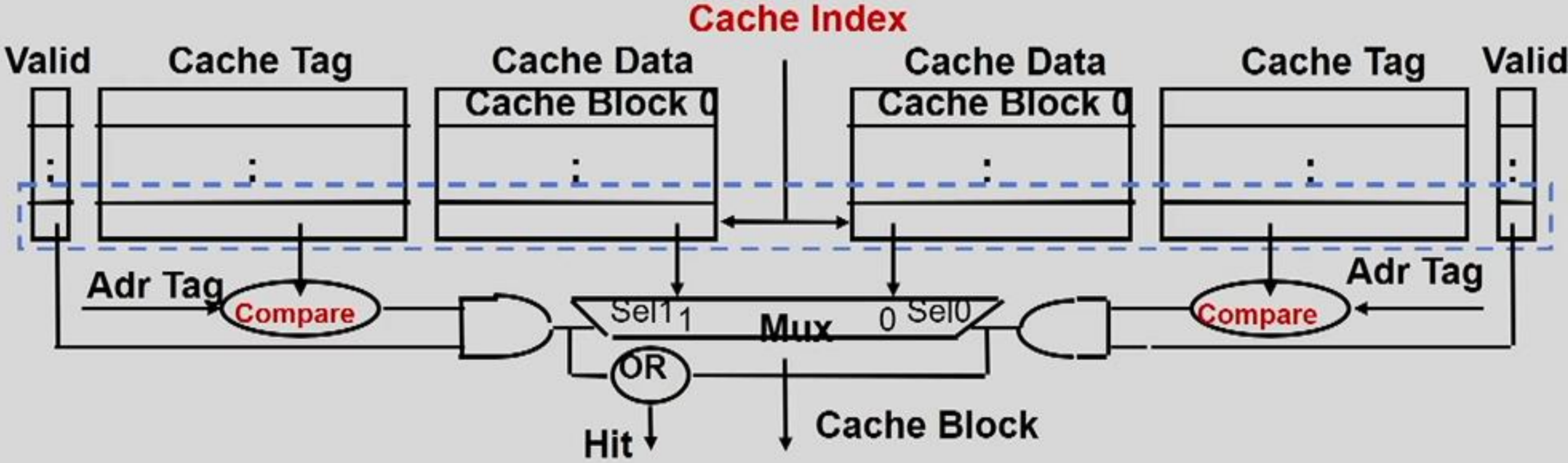
- Block matching is done by comparing the tag in each valid line in the selected set.

Block Identification-Set Associative

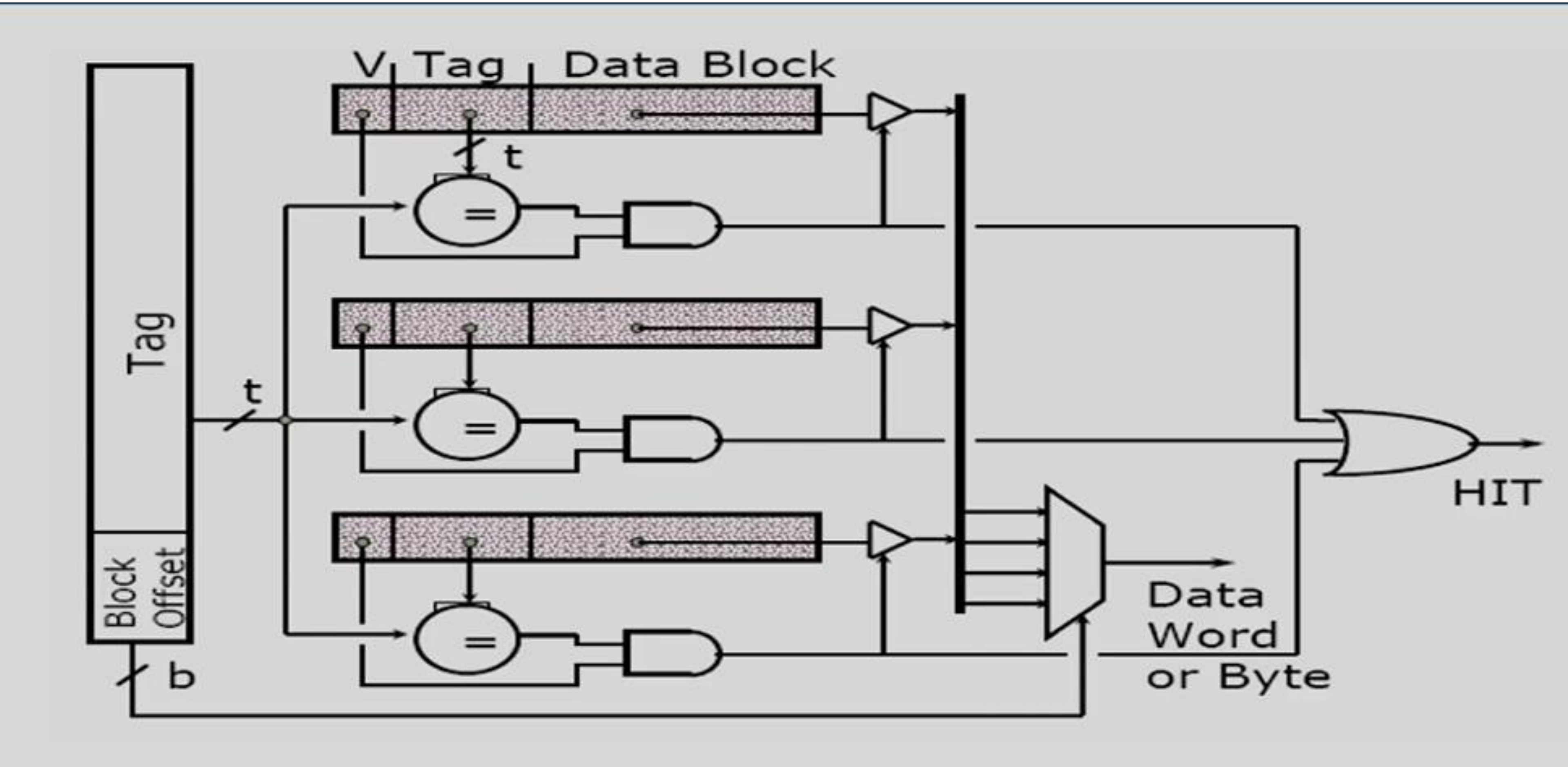


Set Associative Cache

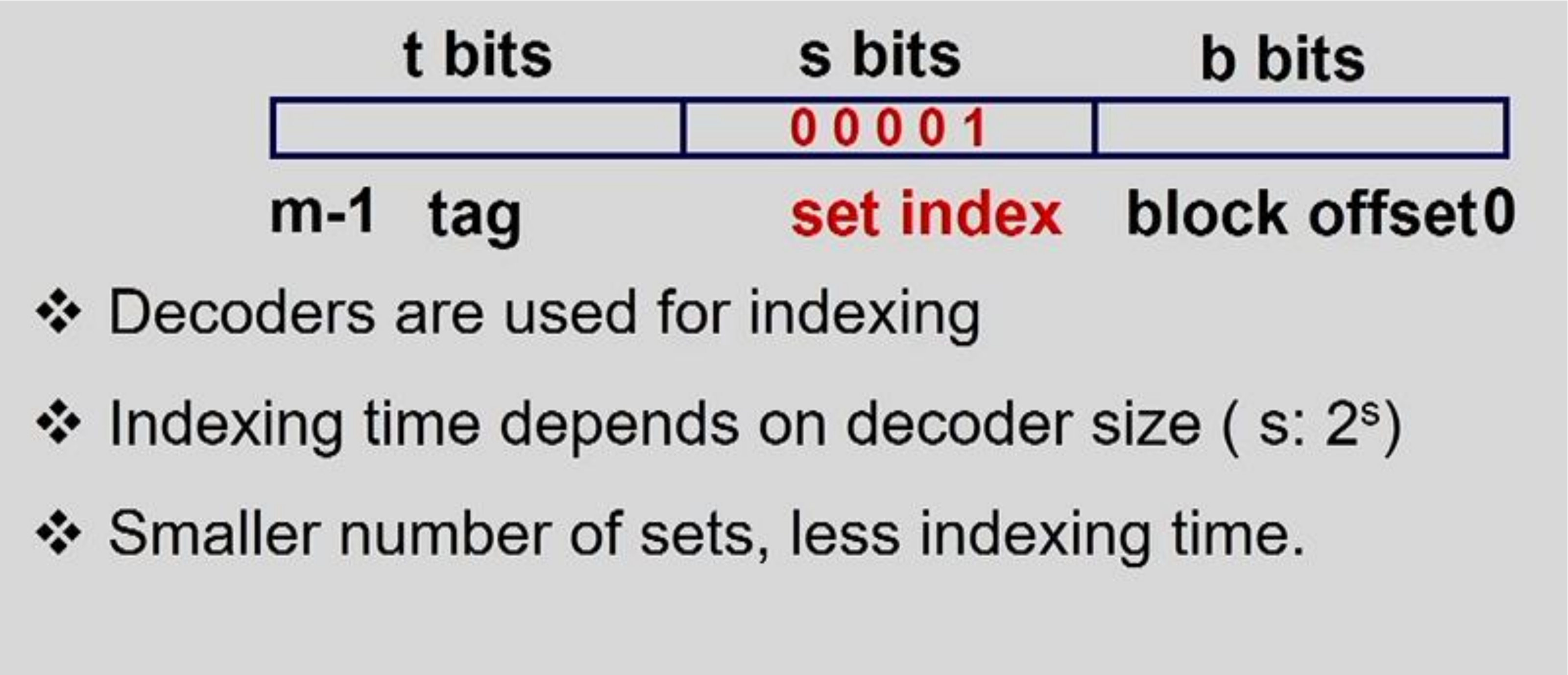
- **2-way set associative**: 2 direct mapped caches in parallel
  - Cache Index selects a set from the cache
  - The two tags in the set are compared to the input in parallel
  - Data is selected based on the tag result



Block Identification-Fully Associative



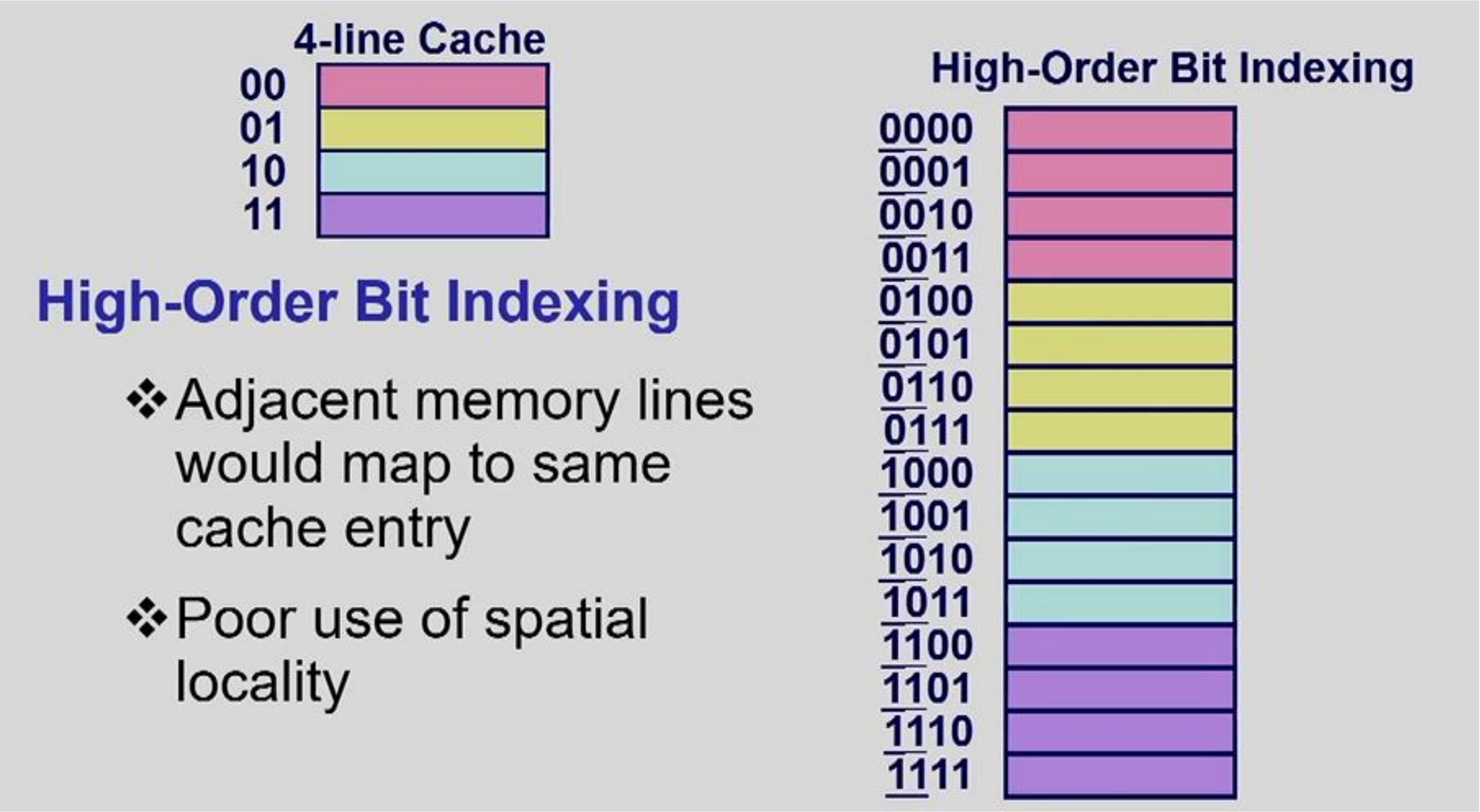
Cache Indexing



- Decoders are used for indexing
- Indexing time depends on decoder size (s: 2^s)
- Smaller number of sets, less indexing time.

- For fast cache access time, no. of sets should be minimum.

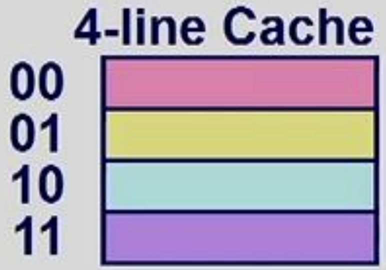
Why use middle bit as Index?



High-Order Bit Indexing

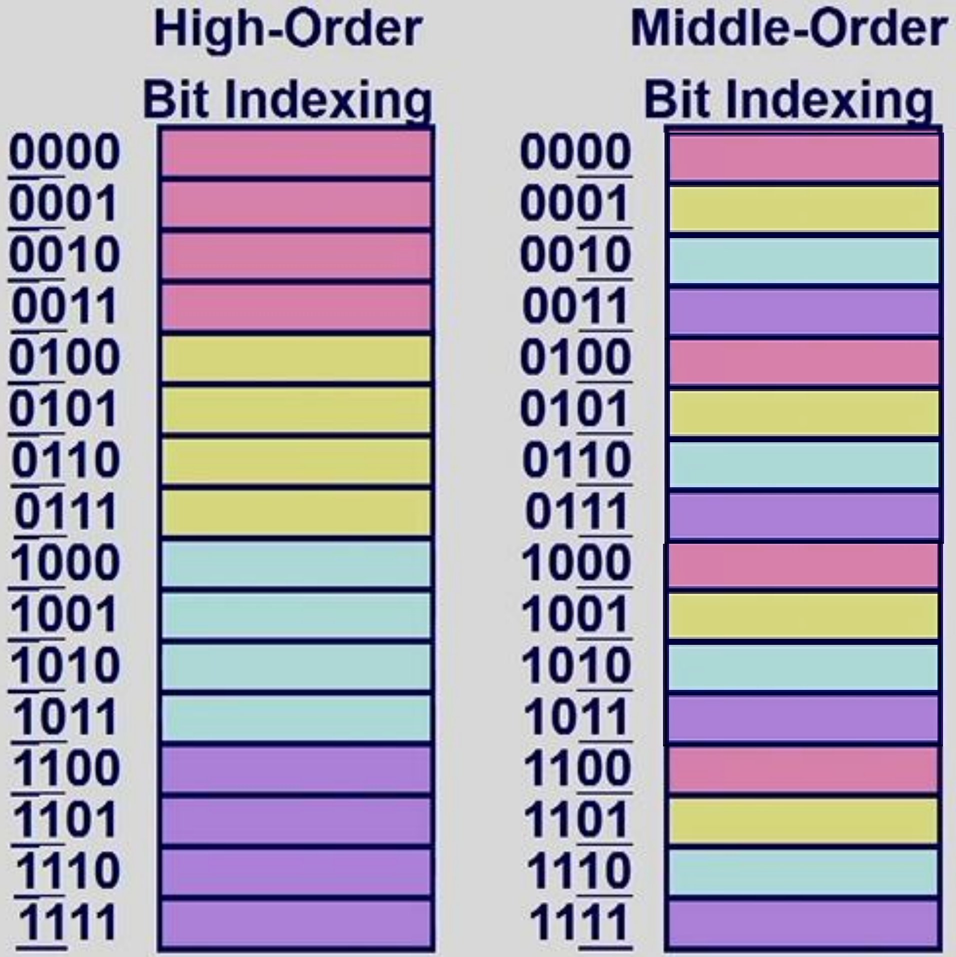
- ❖ Adjacent memory lines would map to same cache entry
- ❖ Poor use of spatial locality

Why use middle bit as Index?



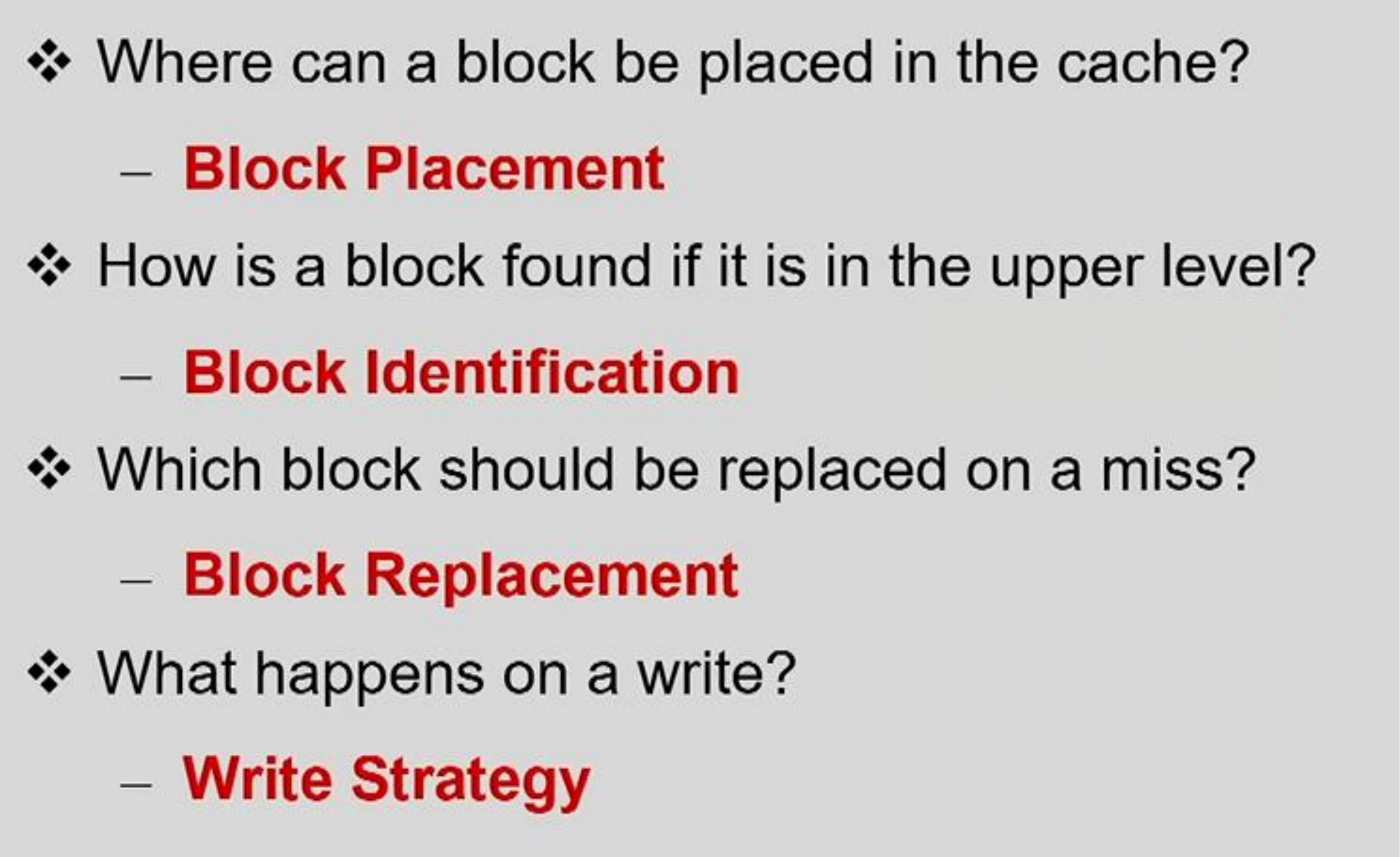
Middle-Order Bit Indexing

- ❖Consecutive memory lines map to different cache lines
- ❖Better use of spacial locality without replacement



	High-Order Bit Indexing	Middle-Order Bit Indexing
0000	🟥	🟥
0001	🟥	🟨
0010	🟥	🟦
0011	🟥	🟪
0100	🟨	🟥
0101	🟨	🟨
0110	🟨	🟦
0111	🟨	🟪
1000	🟦	🟥
1001	🟦	🟨
1010	🟦	🟦
1011	🟦	🟪
1100	🟪	🟥
1101	🟪	🟨
1110	🟪	🟦
1111	🟪	🟪

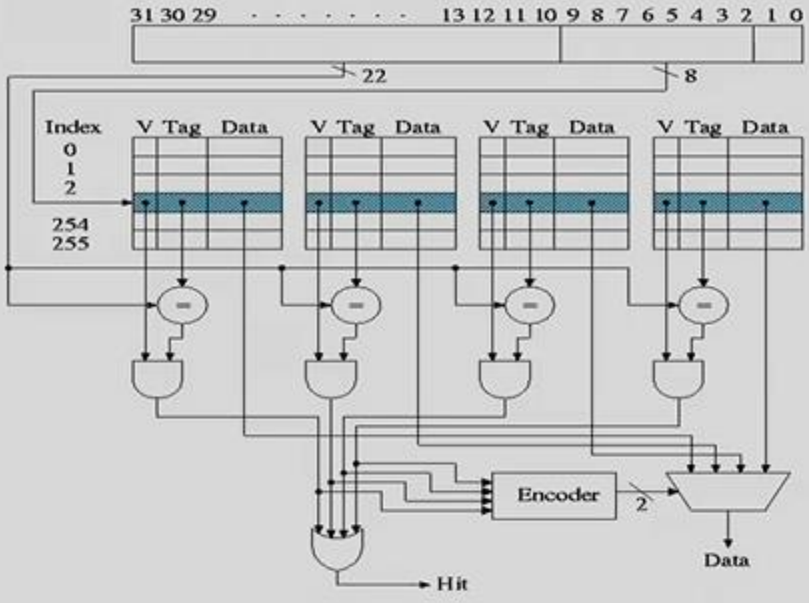
Four Cache Memory Design Choices



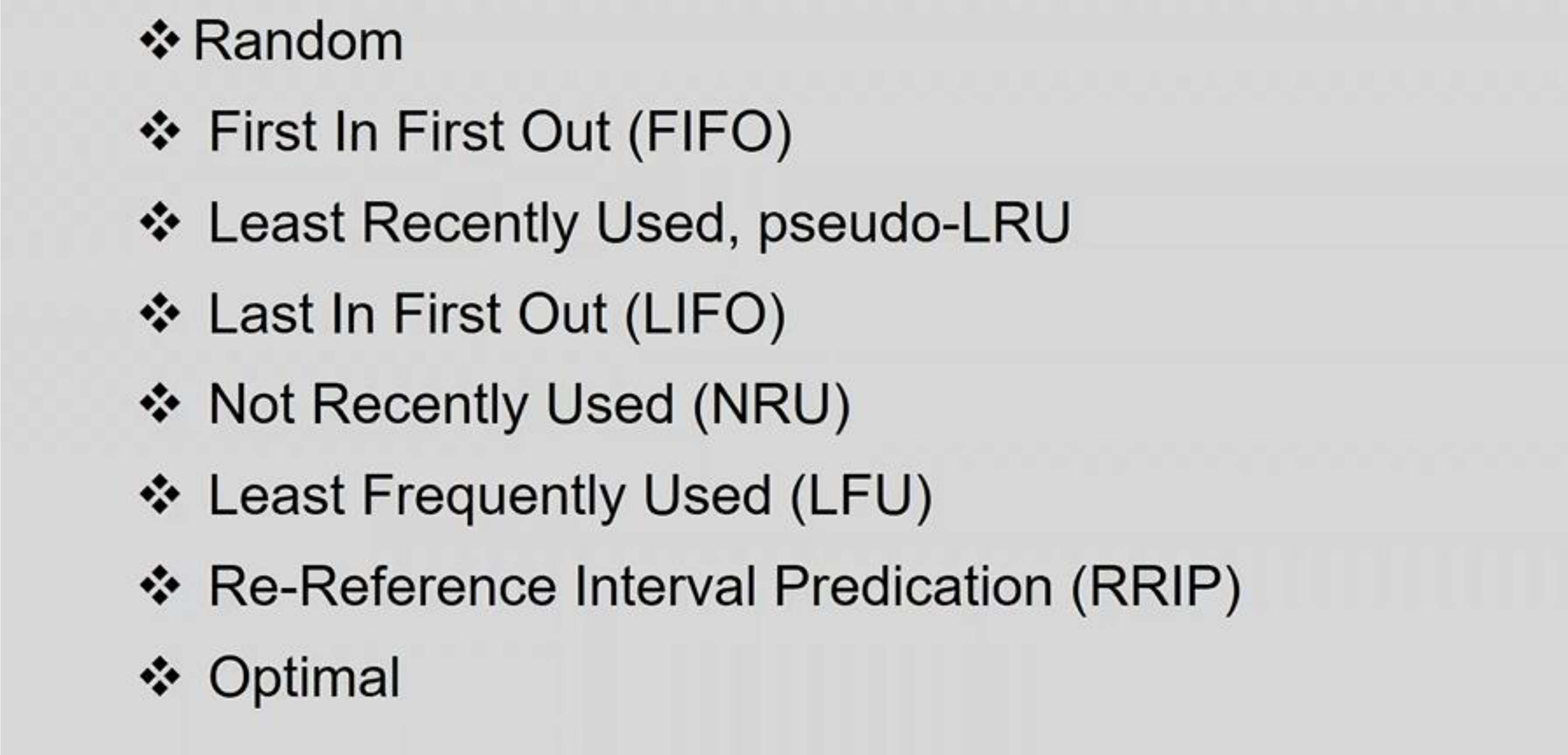
- Where can a block be placed in the cache?
  - **Block Placement**
- How is a block found if it is in the upper level?
  - **Block Identification**
- Which block should be replaced on a miss?
  - **Block Replacement**
- What happens on a write?
  - **Write Strategy**

Block Replacement

- ❖ Cache has finite size. What do we do when it is full?
- ❖ Direct Mapped is Easy
- ❖ Which block in the selected set of a set associative cache?

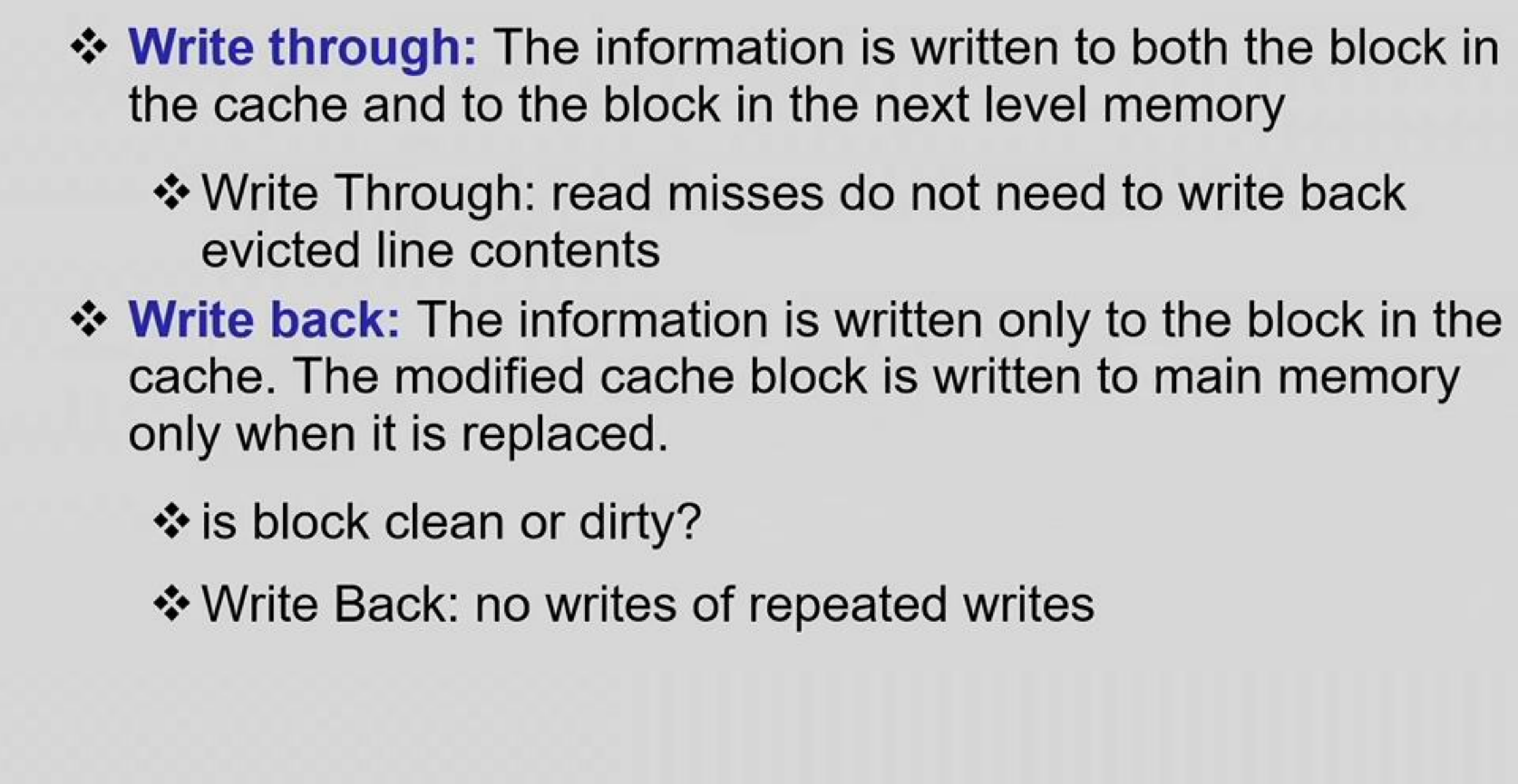


Block Replacement Algorithms



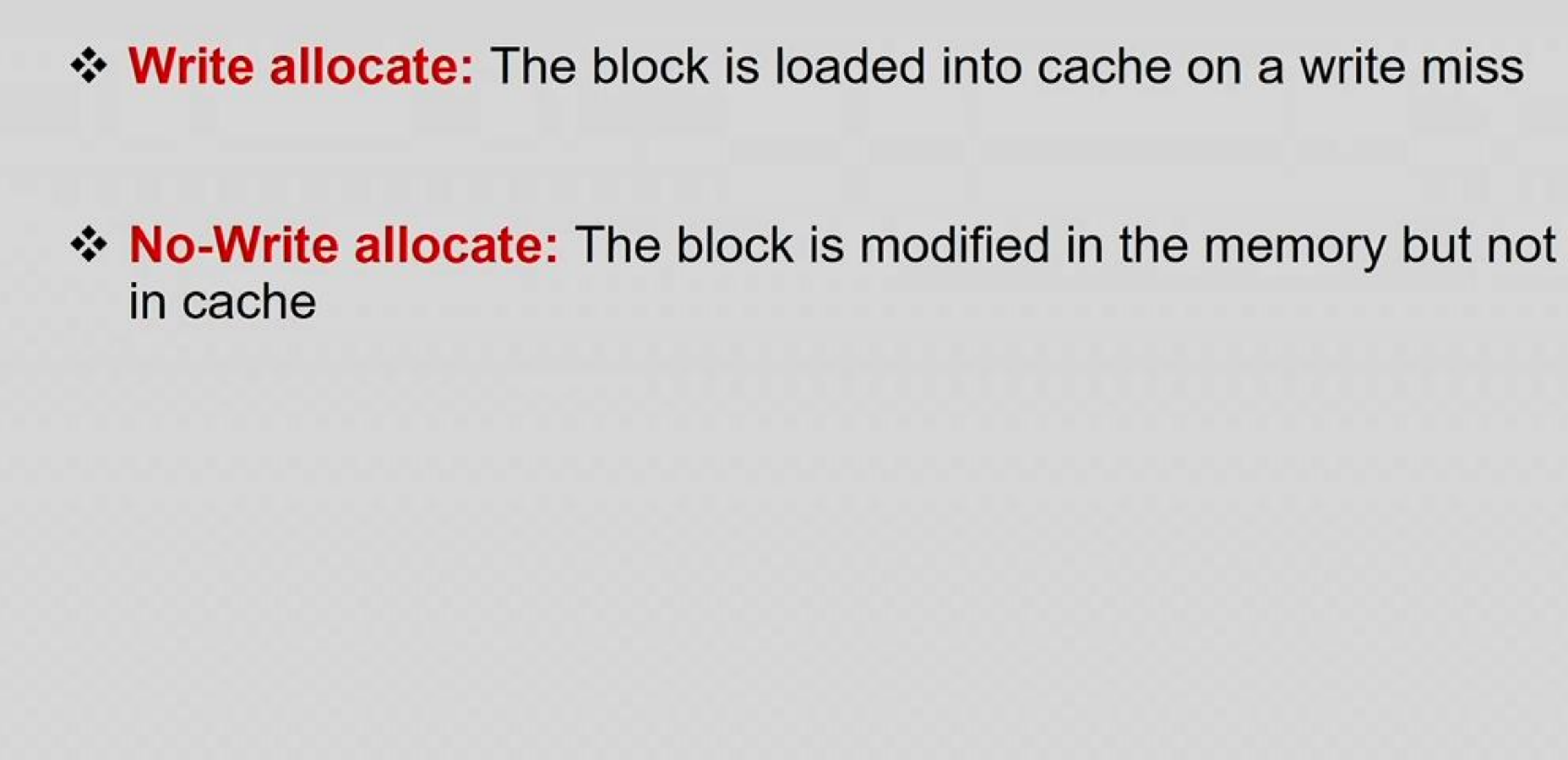
- Random
- First In First Out (FIFO)
- Least Recently Used, pseudo-LRU
- Last In First Out (LIFO)
- Not Recently Used (NRU)
- Least Frequently Used (LFU)
- Re-Reference Interval Predication (RRIP)
- Optimal

Write Strategy



- **Write through:** The information is written to both the block in the cache and to the block in the next level memory
  - Write Through: read misses do not need to write back evicted line contents
- **Write back:** The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.
  - is block clean or dirty?
  - Write Back: no writes of repeated writes

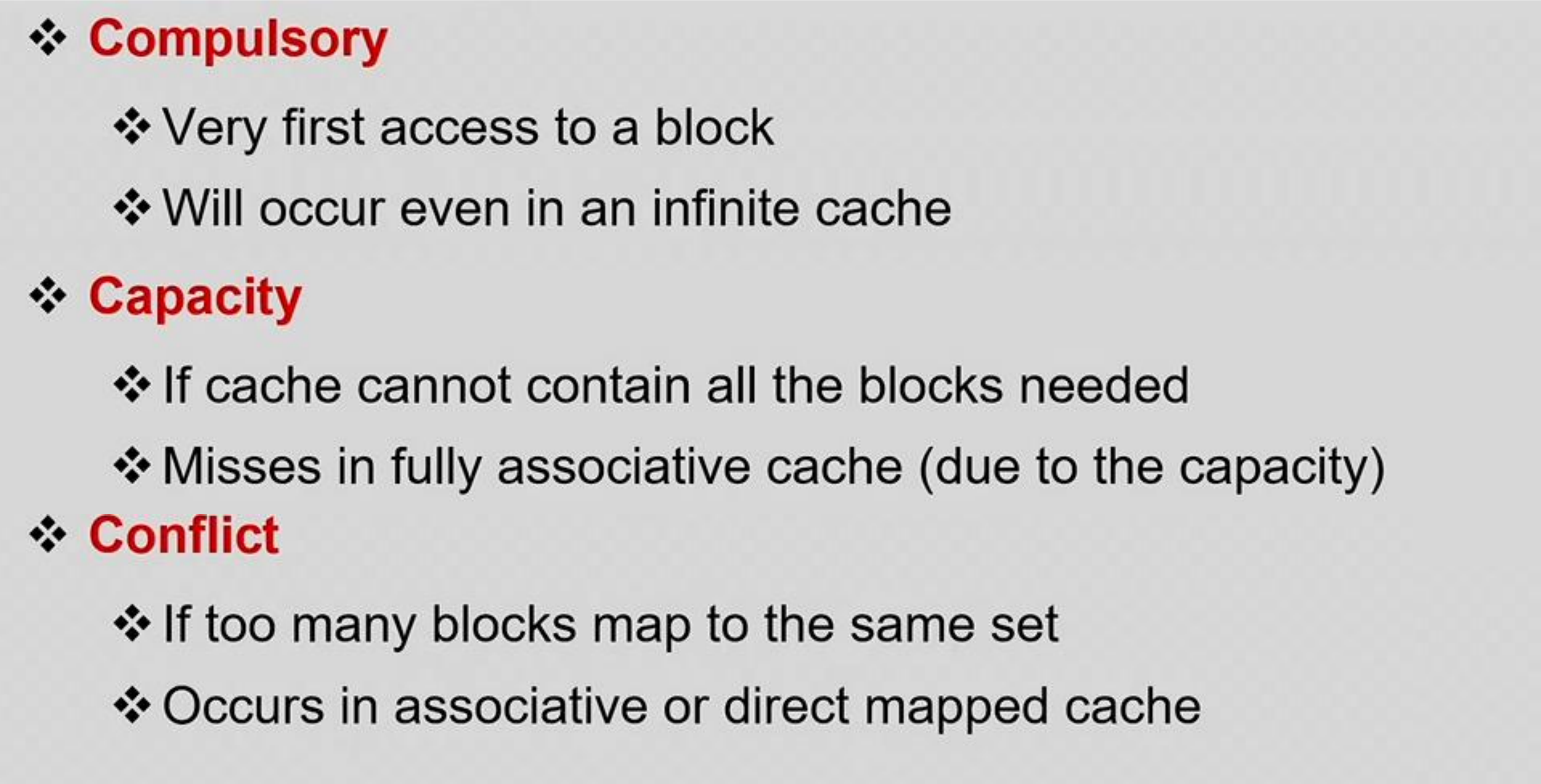
What About Write Miss?



- **Write allocate:** The block is loaded into cache on a write miss

- **No-Write allocate:** The block is modified in the memory but not in cache

Types of Cache Misses



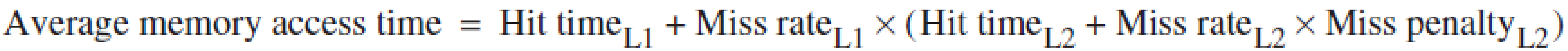
- **Compulsory**
  - Very first access to a block
  - Will occur even in an infinite cache
- **Capacity**
  - If cache cannot contain all the blocks needed
  - Misses in fully associative cache (due to the capacity)
- **Conflict**
  - If too many blocks map to the same set
  - Occurs in associative or direct mapped cache

Cache Formulae

- Cache Size (C) = No. of sets(S) x Associativity(E) x Block Size(B)

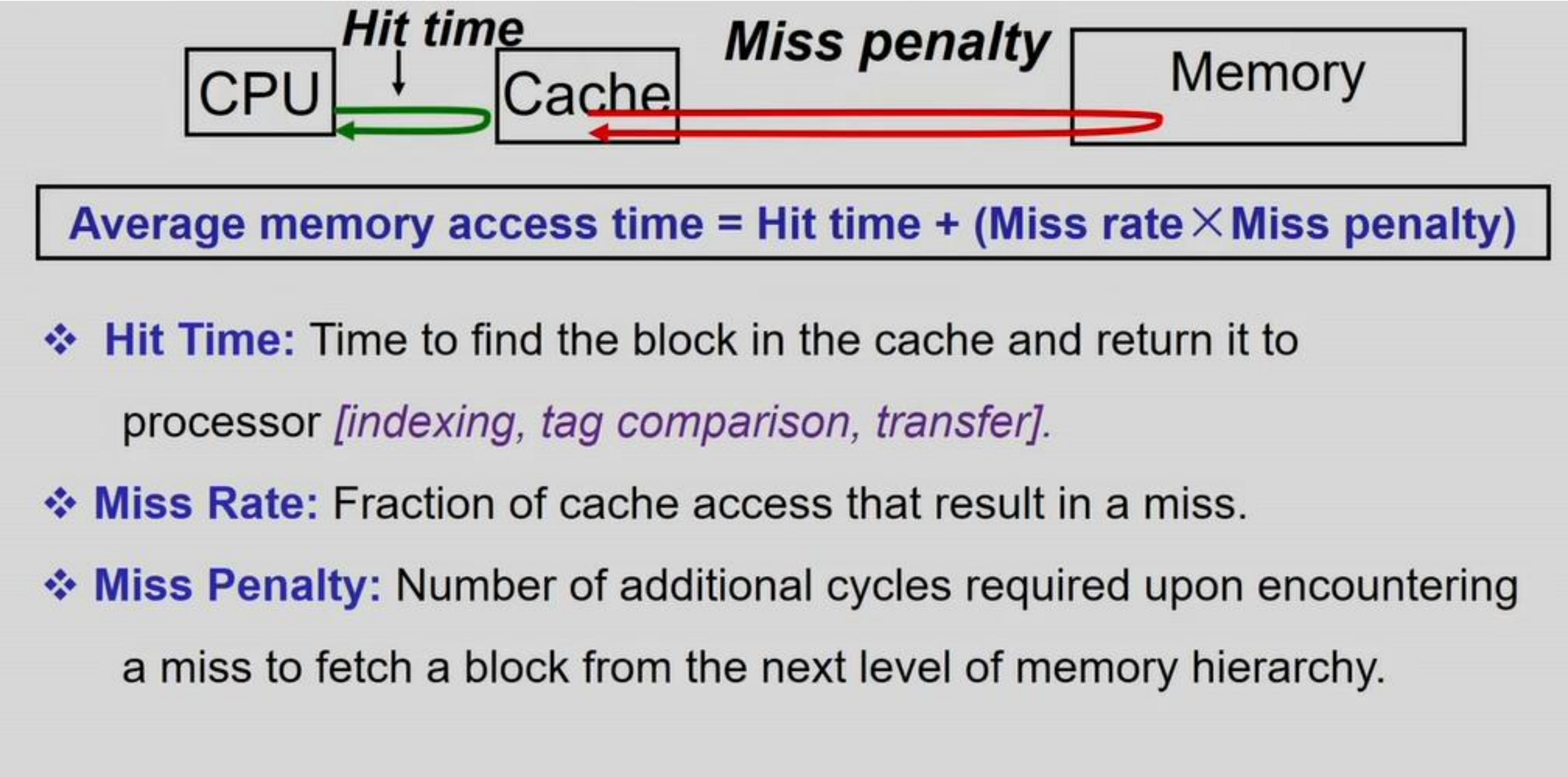
- AMAT = Hit time + Miss rate x Miss penalty

- AMAT of 2 level Cache



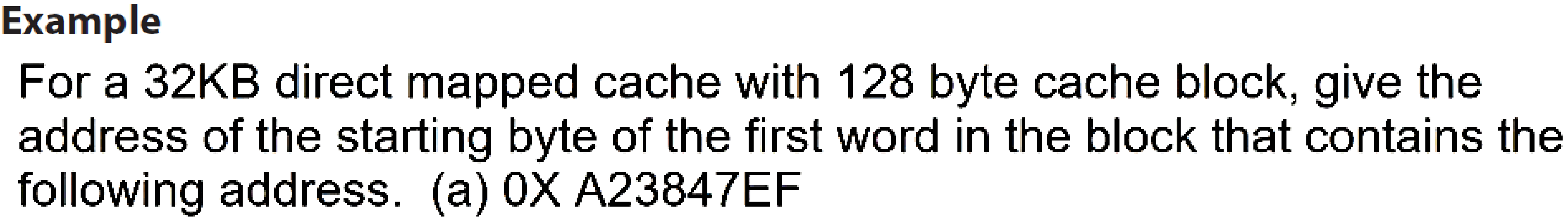
$$\text{Average memory access time} = \text{Hit time}_{L1} + \text{Miss rate}_{L1} \times (\text{Hit time}_{L2} + \text{Miss rate}_{L2} \times \text{Miss penalty}_{L2})$$

Accessing Cache Memory



Average memory access time = Hit time + (Miss rate × Miss penalty)

- ❖ **Hit Time:** Time to find the block in the cache and return it to processor *[indexing, tag comparison, transfer].*
- ❖ **Miss Rate:** Fraction of cache access that result in a miss.
- ❖ **Miss Penalty:** Number of additional cycles required upon encountering a miss to fetch a block from the next level of memory hierarchy.



Example

For a 32KB direct mapped cache with 128 byte cache block, give the address of the starting byte of the first word in the block that contains the following address. (a) 0X A23847EF

- Number of bits required for the cache block off-set field
- Number of sets in the cache
- Number of bits required for the cache set-index field
- Number of bits required for the cache tag field
- Address of the starting byte of the first word in the block

Example

A cache has 256 KB capacity, 1B word, 128 byte block size and 4 way set associative. The system is using 32 bit address. Given the following addresses, which set of cache will be searched and specify which byte of the selected cache block will be forwarded if it is a hit in cache? (a) 0X ABC89987

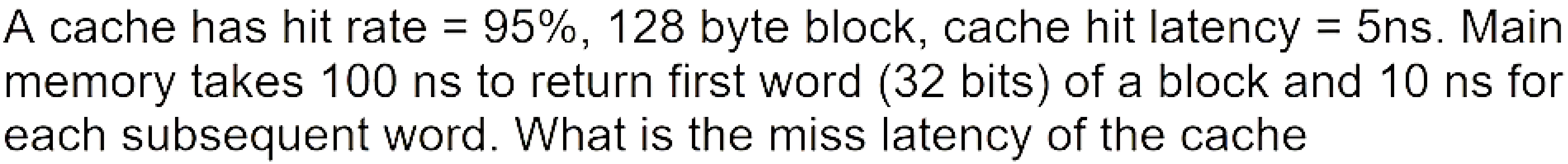
Example

A 64KB cache has 16 byte blocks. If addresses are 32 bits, how many bits are used the tag, index, and offset in this cache?

Example

It takes 2.5 ns to access a tag array value and 4 ns to access a data array, 1 ns to perform hit/ miss comparison and 1ns to return the selected data to processor. (a) What is the cache hit latency of the system? (b) What would be the hit latency of the system if both tag and data array access take 2.5 ns and hit/ miss comparison take 1 ns?

Example



A cache has hit rate = 95%, 128 byte block, cache hit latency = 5ns. Main memory takes 100 ns to return first word (32 bits) of a block and 10 ns for each subsequent word. What is the miss latency of the cache

- Miss Latency
- AMAT

Example

Assume a 2-level cache system with the following specifications.
L1 Hit Time = 1 cycle, L1 Miss Rate = 2.5%, L2 Hit Time = 6 cycles, L2 Miss Rate = 10% (% L1 misses that miss), L2 Miss Penalty = 120 cycles.
Compute the average memory access time.

- AMAT

$$\text{Average memory access time} = \text{Hit time}_{L1} + \text{Miss rate}_{L1} \times (\text{Hit time}_{L2} + \text{Miss rate}_{L2} \times \text{Miss penalty}_{L2})$$

Example

Consider a direct-mapped Cache with 64 blocks and a block size of 16 bytes. To what block number does byte address 1200 map?

Example

Consider a 2-way set associative cache with 64 blocks and a block size of 16 bytes. To which set number does byte address 1200 map?

Example

Caches are important to providing a high-performance memory hierarchy to processors. Below is a list of 32-bit memory address reference, given as word addresses:

21, 166, 201, 143, 61, 166, 62, 133, 111, 143, 144, 61, 144, 61, 62, 134

- What will be the final cache contents for a Fully associative cache with one word blocks and a total size of 8 blocks?
- Use LRU replacement. For each reference identify the index bits, the tag bits, the block offset bits, and if it is a hit or a miss

Cache index	Address in Cache	Hit/Miss

SOLUTION

Cache index	Address in Cache	Hit/Miss
0	21,111	MM
1	166,143	MM
2	201,144	MM
3	143,61	MM
4	61,144	MM
5	166,61	MM
6	62	MH
7	133,134	MM

Example

Caches are important to providing a high-performance memory hierarchy to processors. Below is a list of 32-bit memory address reference, given as word addresses:

21, 166, 201, 143, 61, 166, 62, 133, 111,143, 144, 61

- What will be the final cache contents for a Direct Mapped cache with two word blocks and a total size of 32 blocks?
- Use LRU replacement. For each reference identify the index bits, the tag bits, the block offset bits, and if it is a hit or a miss

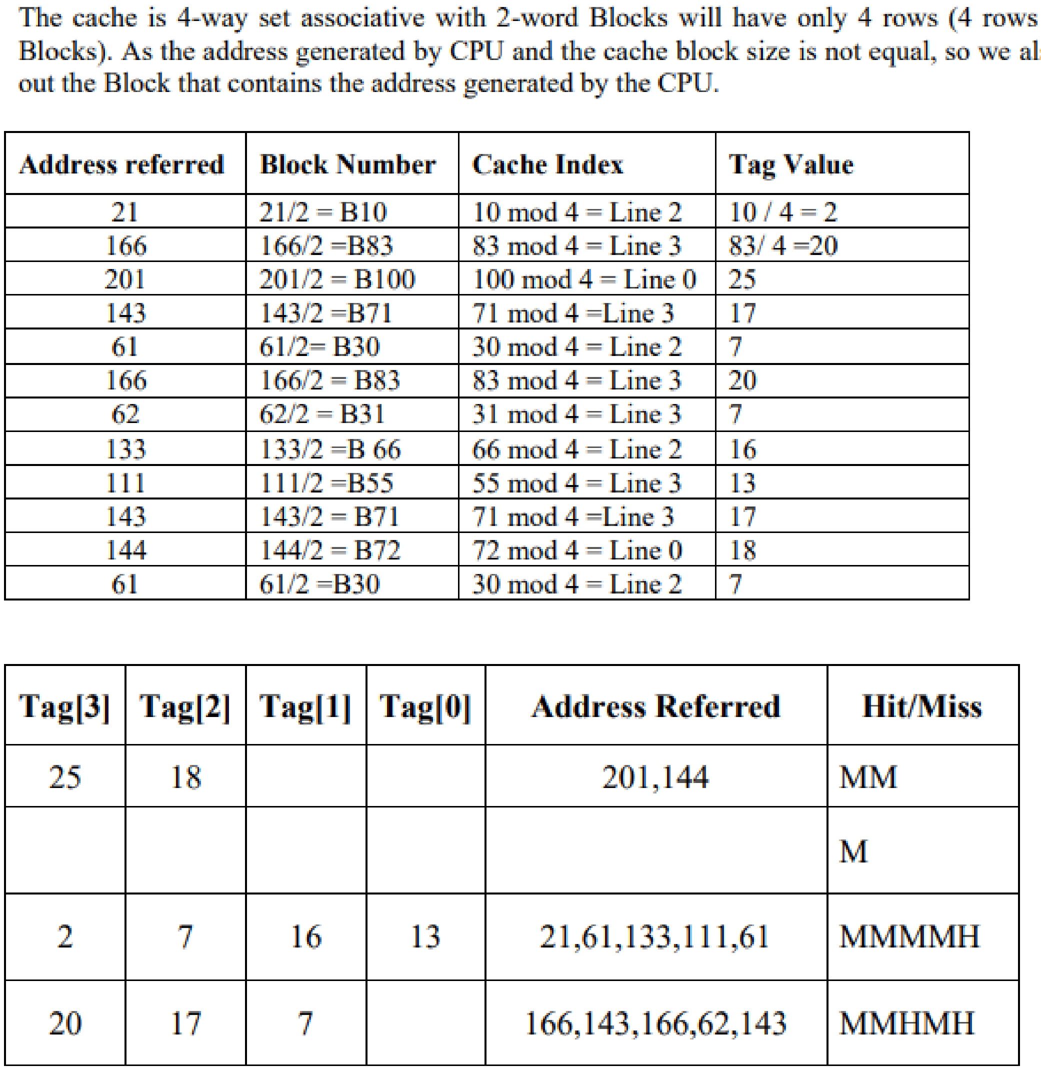
Example

Caches are important to providing a high-performance memory hierarchy to processors. Below is a list of 32-bit memory address reference, given as word addresses:

21, 166, 201, 143, 61, 166, 62, 133, 111,143, 144, 61

- What will be the final cache contents for a 4-way set associative cache with two word blocks and a total size of 16 blocks?
- Use LRU replacement. For each reference identify the index bits, the tag bits, the block offset bits, and if it is a hit or a miss

# of sets= total blocks/associativity= 16/4=4
(Block number) mod (no. of sets) →10 mod 4= 2



The cache is 4-way set associative with 2-word Blocks will have only 4 rows (4 rows x 4-way = 16 Blocks). As the address generated by CPU and the cache block size is not equal, so we also need to find out the Block that contains the address generated by the CPU.

Address referred	Block Number	Cache Index	Tag Value
21	21/2 = B10	10 mod 4 = Line 2	10 / 4 = 2
166	166/2 =B83	83 mod 4 = Line 3	83/ 4 =20
201	201/2 = B100	100 mod 4 = Line 0	25
143	143/2 =B71	71 mod 4 =Line 3	17
61	61/2= B30	30 mod 4 = Line 2	7
166	166/2 = B83	83 mod 4 = Line 3	20
62	62/2 = B31	31 mod 4 = Line 3	7
133	133/2 =B 66	66 mod 4 = Line 2	16
111	111/2 =B55	55 mod 4 = Line 3	13
143	143/2 = B71	71 mod 4 =Line 3	17
144	144/2 = B72	72 mod 4 = Line 0	18
61	61/2 =B30	30 mod 4 = Line 2	7

Tag[3]	Tag[2]	Tag[1]	Tag[0]	Address Referred	Hit/Miss
25	18			201,144	MM
					M
2	7	16	13	21,61,133,111,61	MMMMH
20	17	7		166,143,166,62,143	MMHMH