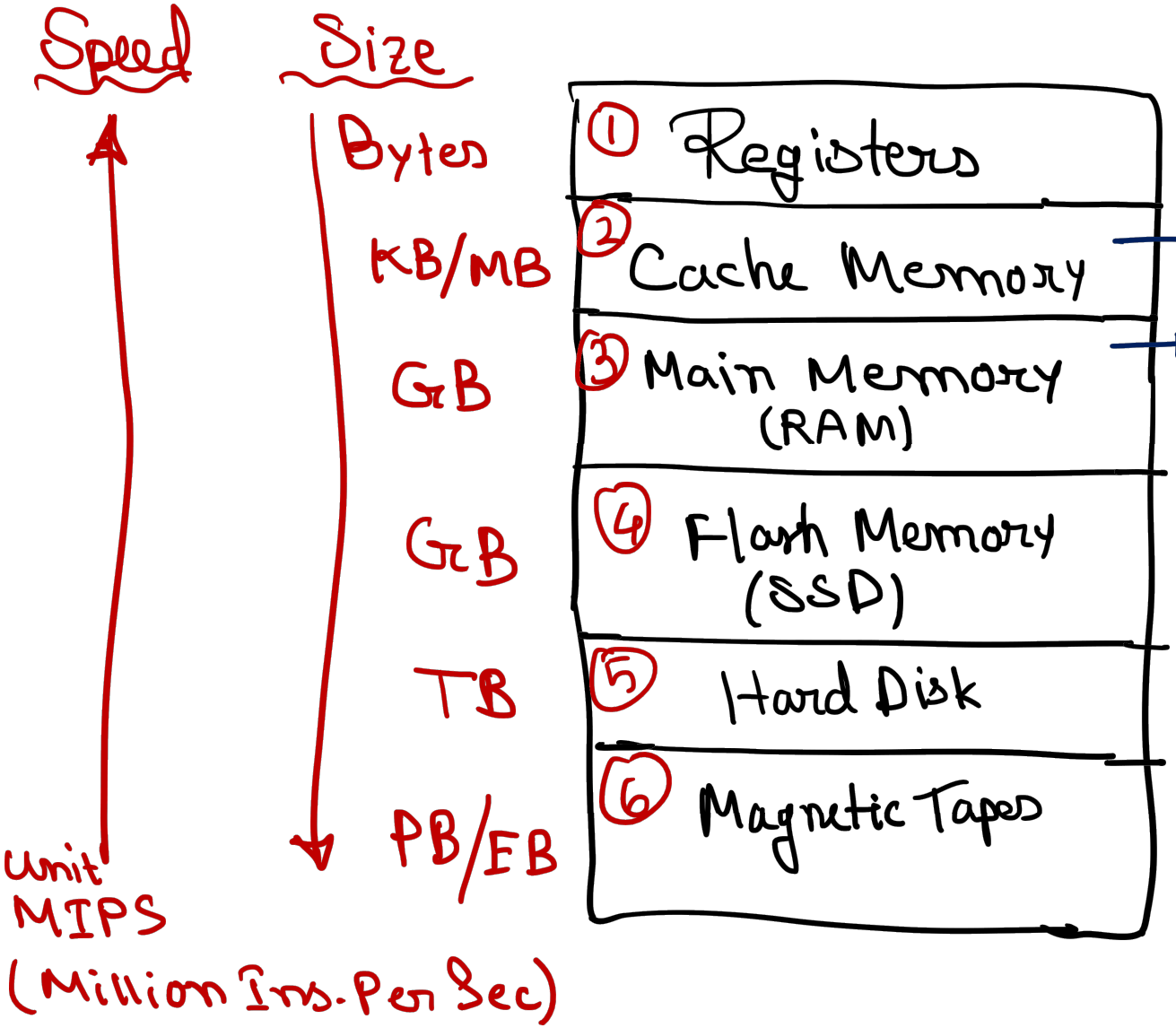
Memory Hierarchy (Access time, Speed, Size, Cost)

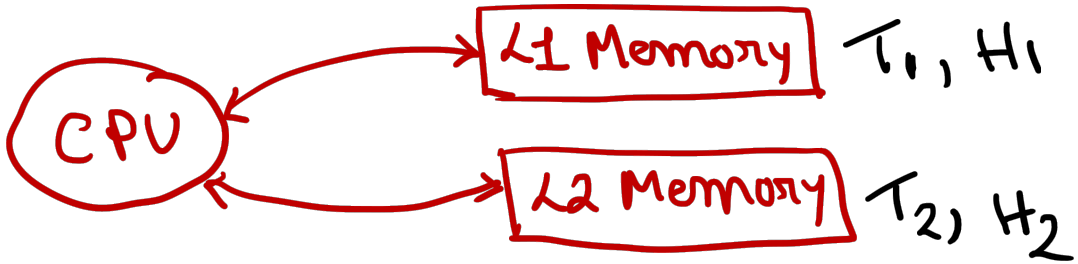


1 Byte = 8 bits
KB (Kilobyte): 1024 bytes
MB (Megabyte): 1024 kilobytes
GB (Gigabyte): 1024 megabytes
TB (Terabyte): 1024 gigabytes
PB (Petabyte): 1024 terabytes

1 & 2 → Internal Memory
3 → Primary Memory
5, 4 → Secondary "
6 → Tertiary Memory

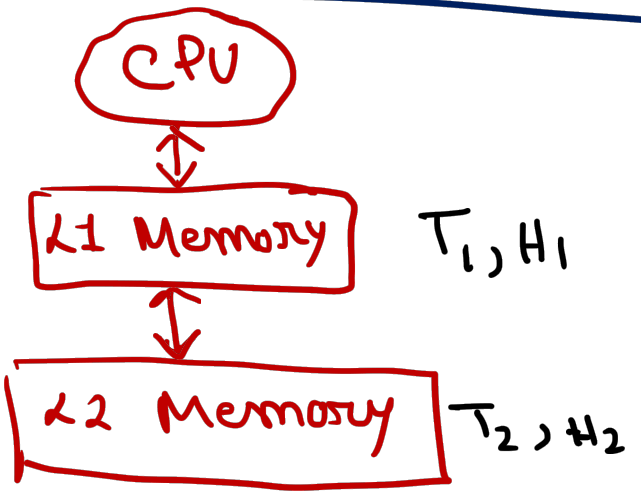
* 2 Level Memory Organization :-

Parallel Organization



Avg Access Time (T_{av}) = $T_1 H_1 + (1 - H_1) T_2$

Heirarchical Organization

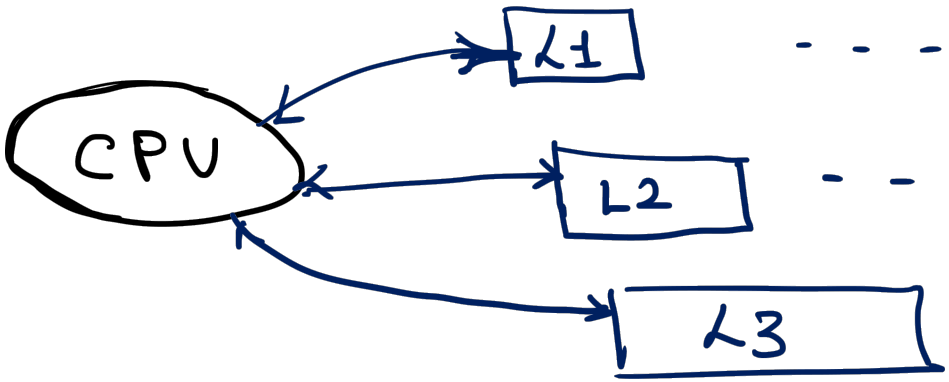


$T_{av} = H_1 T_1 + (1 - H_1)(T_1 + T_2)$

Note:- L1 is faster than L2 :- Access Time $T_1 < T_2$

* 3 Level Memory Organization

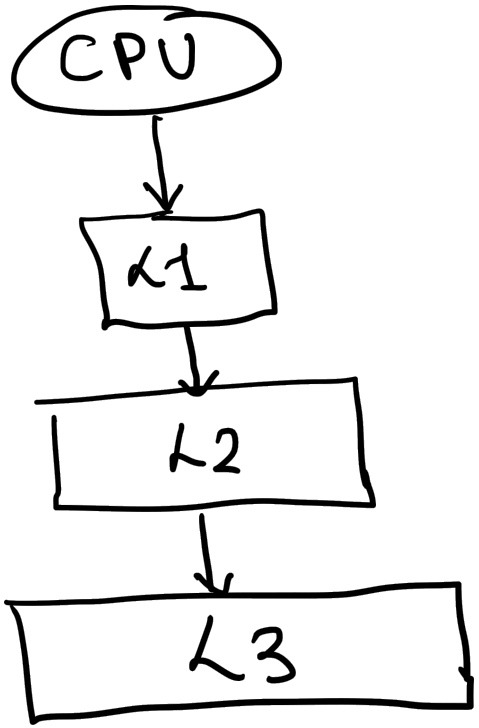
Parallel Organization :-



	Access Time	Hit Ratio
L1	T_1	H_1
L2	T_2	H_2
L3	T_3	H_3

∴ Avg Memory Access Time,

$$T_{avg} = H_1T_1 + (1-H_1)H_2T_2 + (1-H_1)(1-H_2)T_3$$

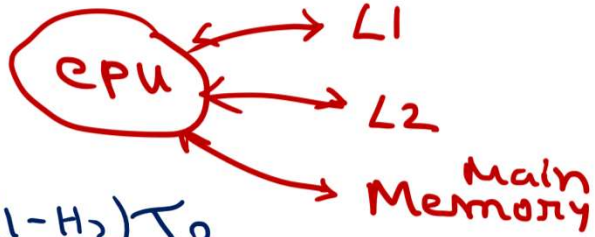


	Access Time	Hit Ratio
L1	T_1	H_1
L2	T_2	H_2
L3	T_2	H_3

$$T_{avg} = T_1 H_1 + (1-H_1) H_2 (T_1+T_2) + (1-H_1)(1-H_2)(T_1+T_2+T_3)$$

Consider a system of two level caches. Access time of L1 & L2 and main memory are 1ns, 10ns & 500ns. The Hit ratio of L1 & L2 are 0.8 & 0.9. What is the average access time of system ignoring search time is cache.

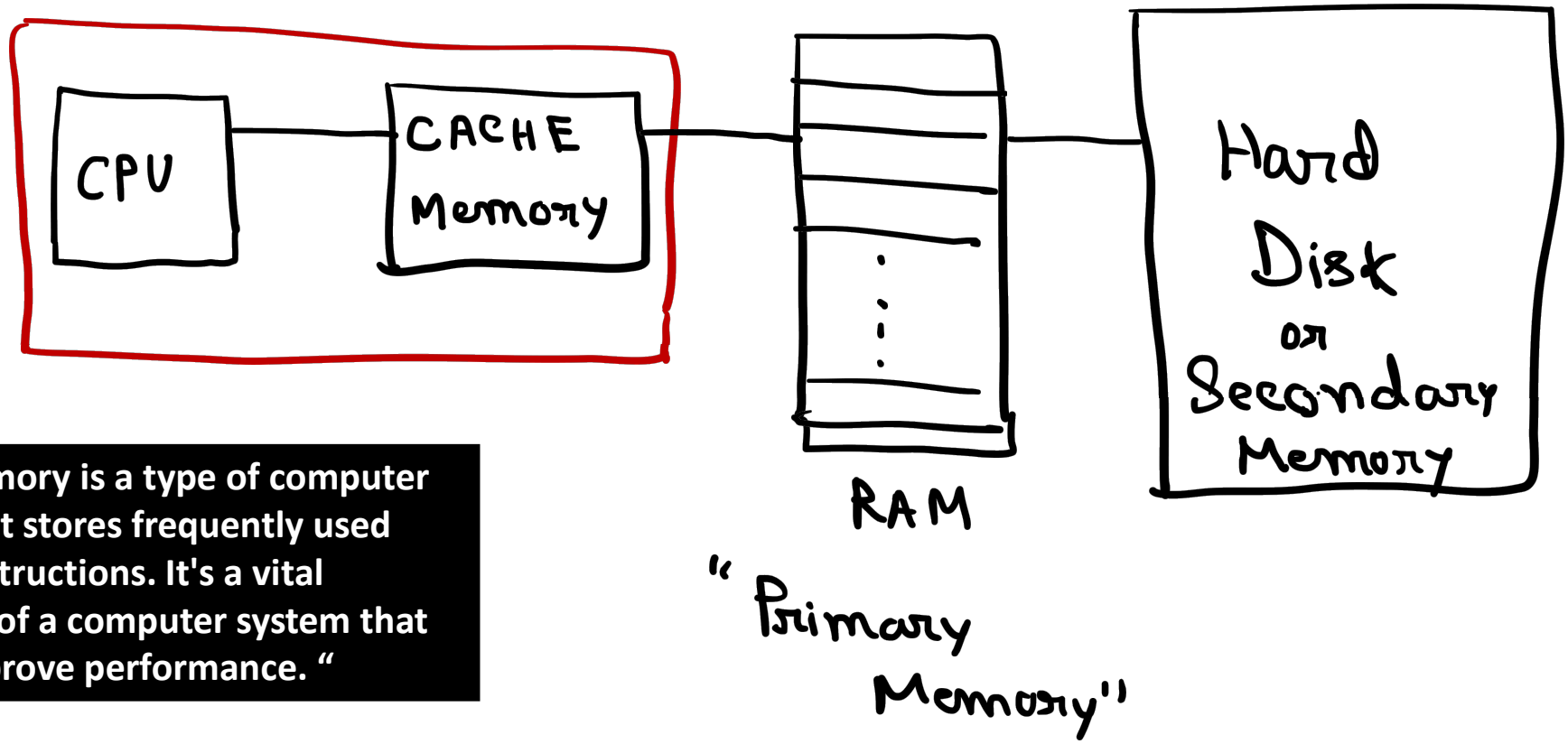
Ans: ∴ Avg Memory Access Time,



$$T_{avg} = H_1T_1 + (1-H_1)H_2T_2 + (1-H_1)(1-H_2)T_3$$

$$T_{avg} = 12.6\,ns \quad \text{(Ans)}$$

CACHE MEMORY



" Cache memory is a type of computer memory that stores frequently used data and instructions. It's a vital component of a computer system that helps to improve performance. "

How does cache memory work?

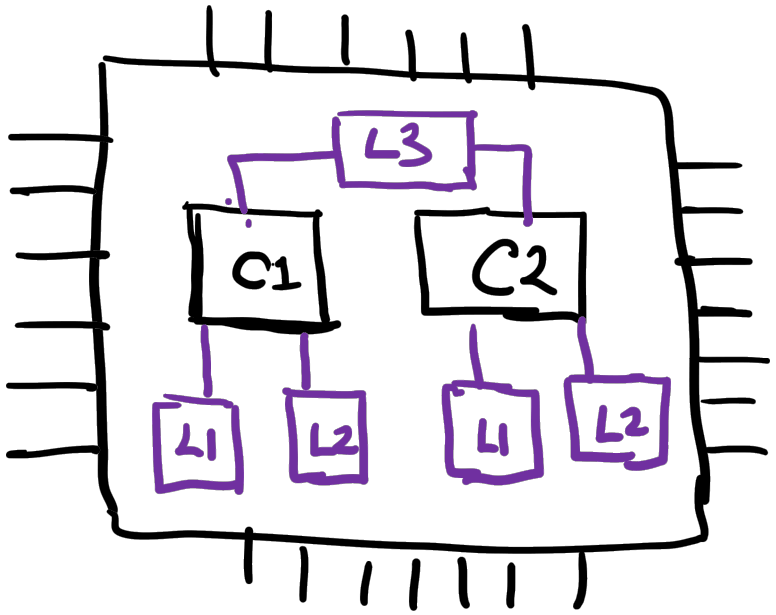
- Cache memory is a smaller, faster memory that's located closer to the processor core.
- It stores copies of data from frequently used main memory locations.
- This reduces the need to access main memory multiple times for the same piece of data.
- This results in faster data retrieval and improved system performance.

Types of cache memory

- Level 1 cache: The fastest portion of the CPU cache, located on the CPU itself.
- Level 2 cache: Connected to (but outside) the CPU.
- Level 3 cache: Specialized memory developed to improve the performance of L1 and L2.

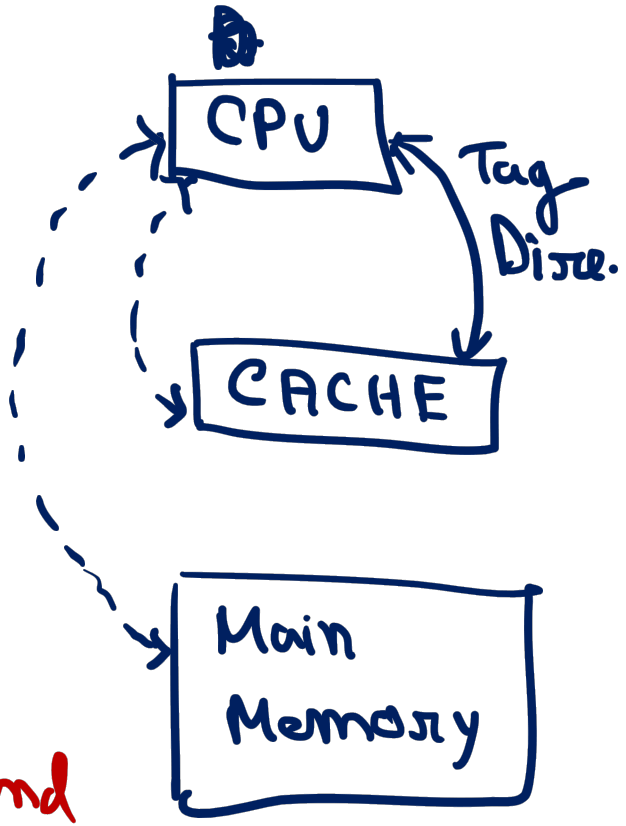
eg:-

Dual Core



* Cache Related Term

- → Chache Hit (Chache Latency)
- → Tag Directory (Data Structure)
- → Chache Miss (Miss Latency)
- → Page Fault (Page Fault Service Time)
- → Page Hit

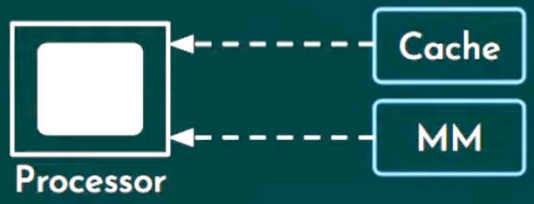


↓
if data is not found in main memory, then with the help of OS, data veterive to main Memory

Memory Interfacing – Solved PYQs

Q1: A cache memory needs an access time of 30 ns and main memory 150 ns, what is the average access time of CPU (assume hit ratio = 80%)?

Sol.



Access Time

T_{cache} = 30 ns

T_{MM} = 150 ns

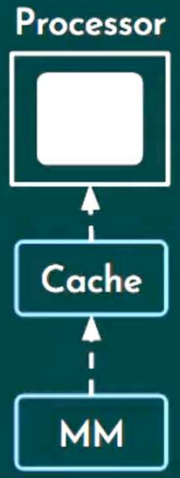
Hit Ratio

H_{cache} = 80% = 0.8

$T_{avg} = H_{cache} T_{cache} + (1-H_{cache}) T_{MM}$

$= 0.8 \times 30 + (1 - 0.8) \times 150$ ns

$= 24 + 0.2 \times 150 = 54$ ns



Access Time

T_{cache} = 30 ns

T_{MM} = 150 ns

Hit Ratio

H_{cache} = 80% = 0.8

$T_{avg} = H_{cache} T_{cache} + (1-H_{cache}) (T_{cache} + T_{MM})$

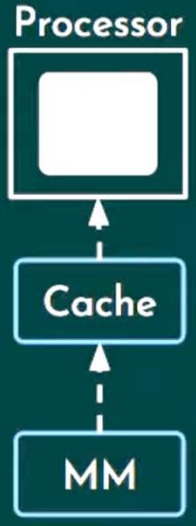
$= 0.8 \times 30 + (1 - 0.8) \times (30+150)$ ns

$= 24 + 0.2 \times 180 = 60$ ns

Q2: Assume that for a certain processor, a read request takes 50 nanoseconds on a cache miss and 5 nanoseconds on a cache hit. Suppose while running a program, it was observed that 80% of the processor's read requests result in a cache hit. The average read access time in nanoseconds is _____.

GATE CS 2015

Sol.



Access Time

$T_{cache} + T_{MM} = 50$ ns

$T_{cache} = 5$ ns

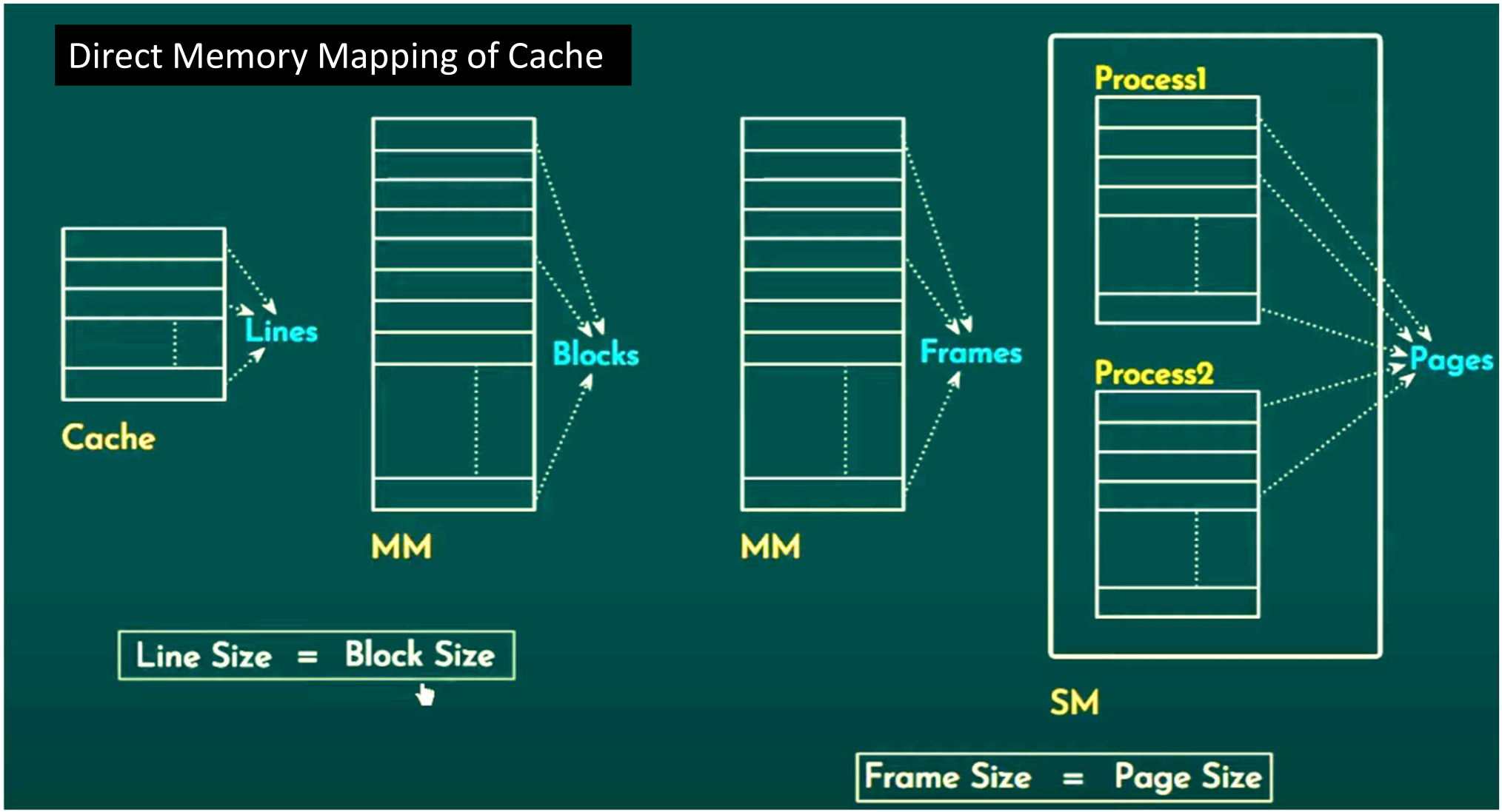
Hit Ratio

$H_{cache} = 80\% = 0.8$

$$T_{avg} = H_{cache} T_{cache} + (1-H_{cache}) (T_{cache} + T_{MM})$$

$$= 0.8 \times 5 + (1 - 0.8) \times 50 \text{ ns}$$

$$= 4 + 0.2 \times 50 = 14 \text{ ns}$$


Direct Memory Mapping of Cache
Lines
Cache
Blocks
MM
Frames
MM
Process1
Process2
Pages
SM
Line Size = Block Size
Frame Size = Page Size

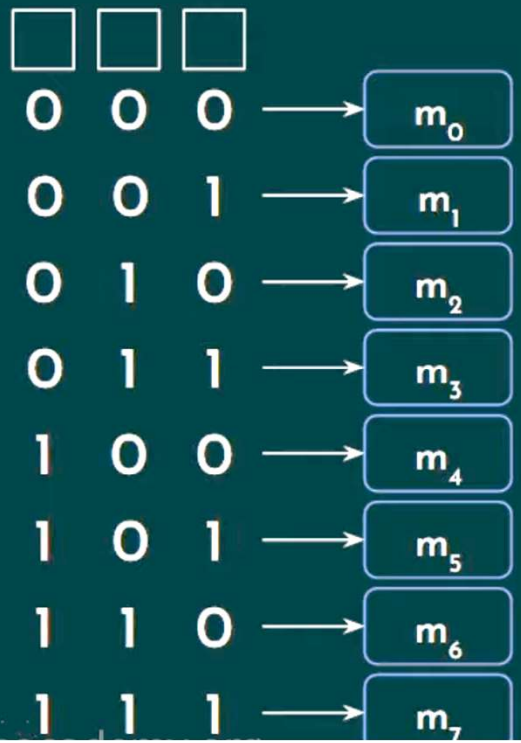
Word: Smallest Addressable Unit of Memory.

- Byte-Addressable Memory: 1 word = 1 Byte

Main Memory Size : 64 words i.e. (0, 1, . . . ,63)

Block Size : 4 words

No. of Blocks in MM : 64 / 4 = 16

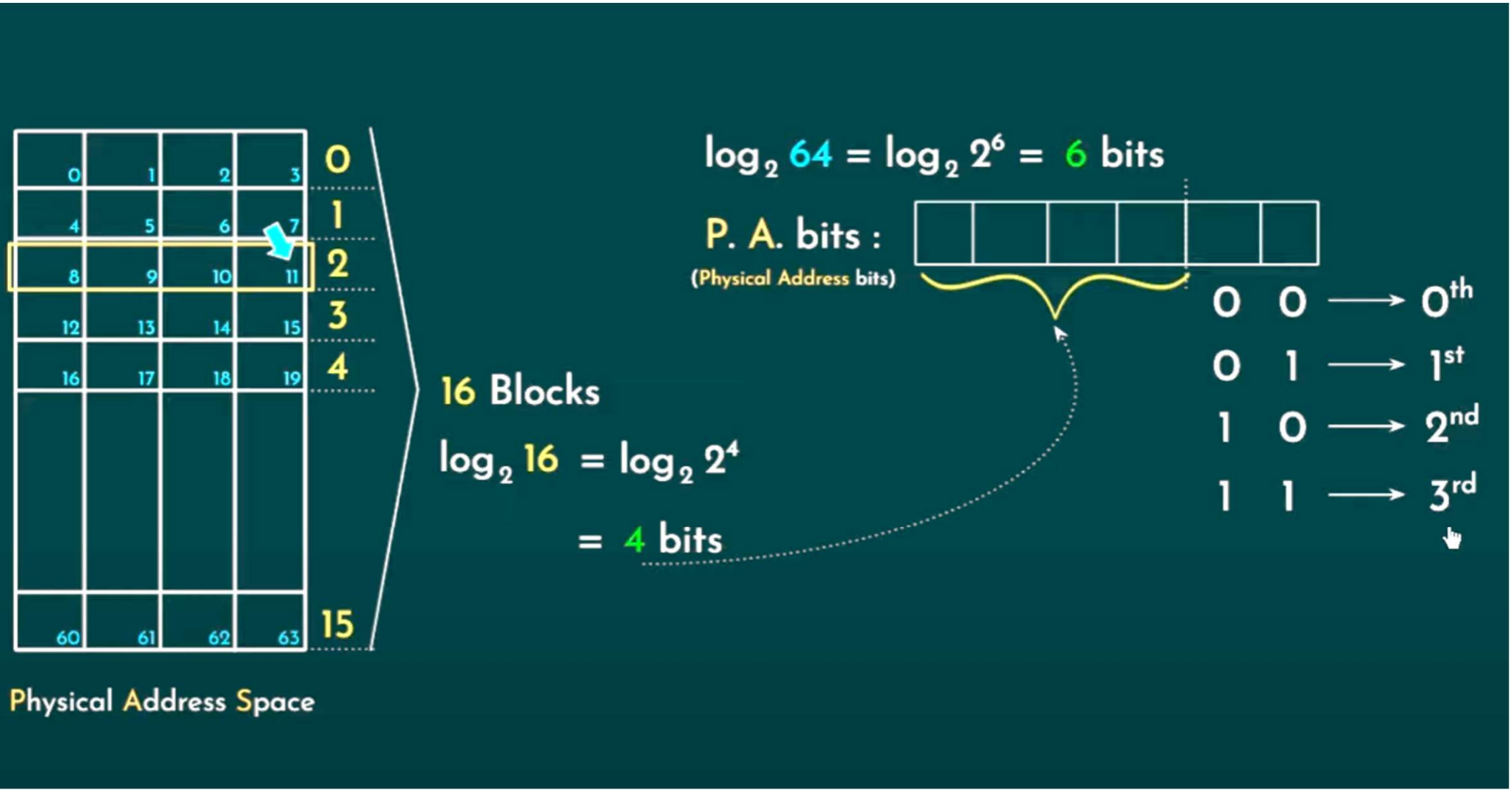


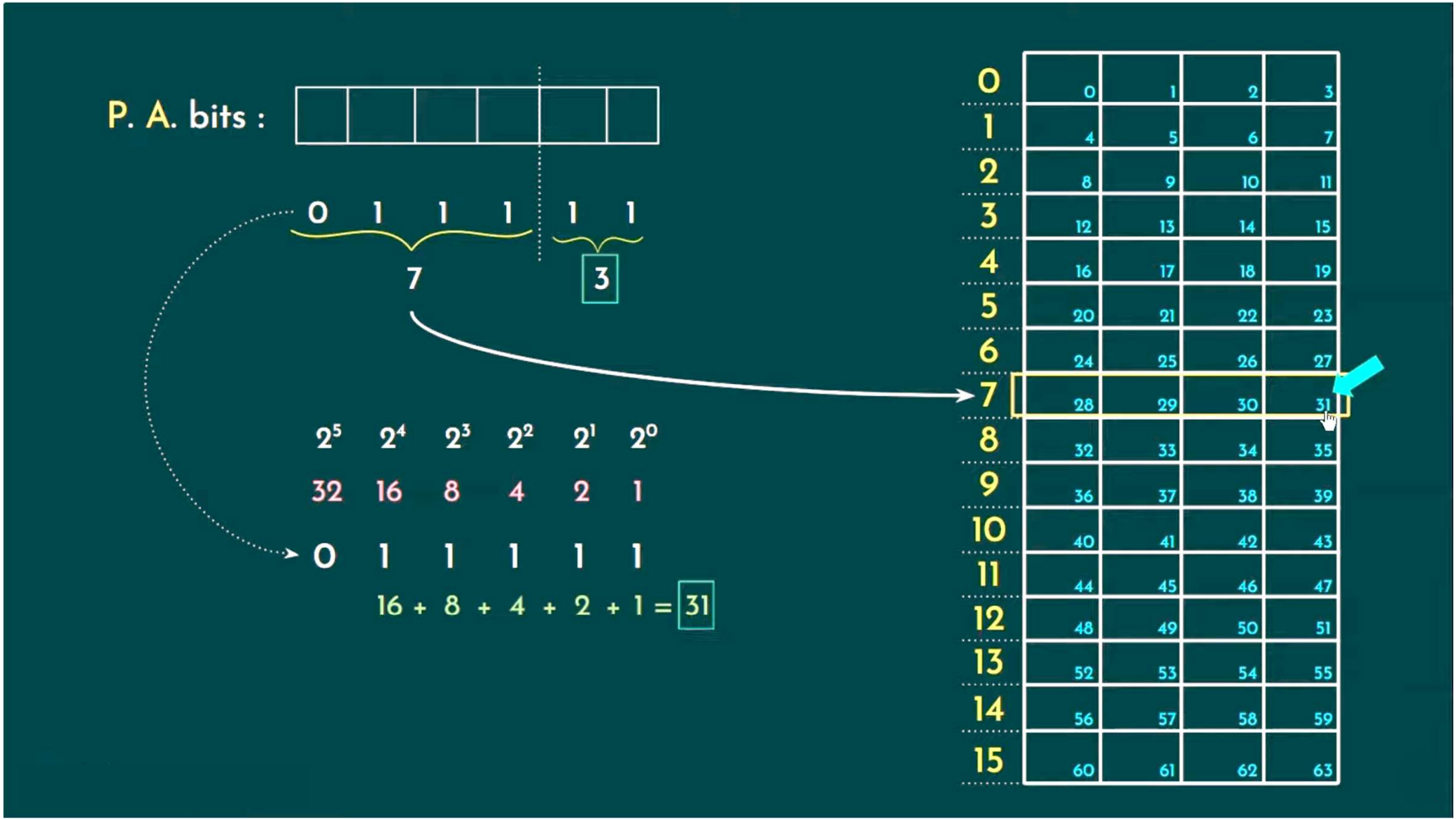
64 words

$\log_2 64 = \log_2 2^6$

= 6 bits

0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15
4	16	17	18	19
5	20	21	22	23
6	24	25	26	27
7	28	29	30	31
8	32	33	34	35
9	36	37	38	39
10	40	41	42	43
11	44	45	46	47
12	48	49	50	51
13	52	53	54	55
14	56	57	58	59
15	60	61	62	63


0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
16 17 18 19
60 61 62 63
0
1
2
3
4
15
Physical Address Space
16 Blocks
log2 16 = log2 2^4
= 4 bits
log2 64 = log2 2^6 = 6 bits
P. A. bits :
(Physical Address bits)
0 0 → 0th
0 1 → 1st
1 0 → 2nd
1 1 → 3rd


P. A. bits :
0 1 1 1 1 1
7
3
2^5 2^4 2^3 2^2 2^1 2^0
32 16 8 4 2 1
0 1 1 1 1 1
16 + 8 + 4 + 2 + 1 = 31
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
16 17 18 19
20 21 22 23
24 25 26 27
28 29 30 31
32 33 34 35
36 37 38 39
40 41 42 43
44 45 46 47
48 49 50 51
52 53 54 55
56 57 58 59
60 61 62 63

Cache Size : 16 words

Line Size : 4 words

No. of Lines in Cache : 16 / 4 = 4 i.e. 0, 1, 2, 3

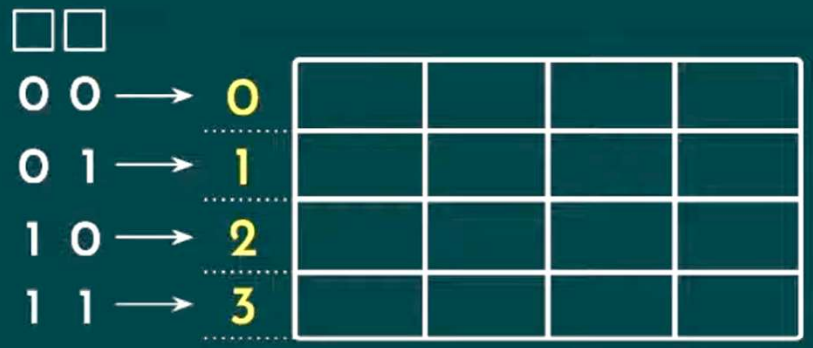
Block Size : 4 words

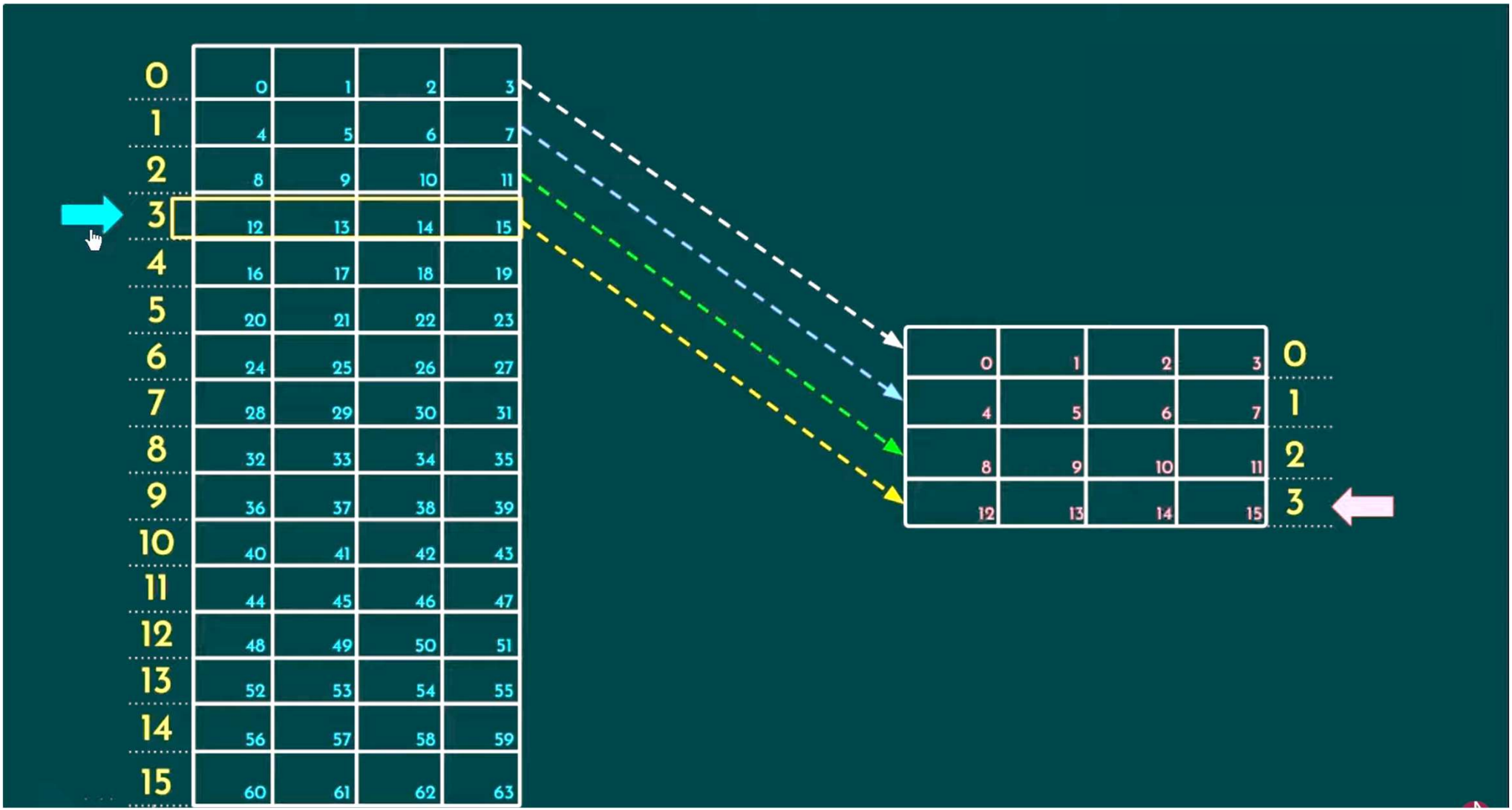
Block Size = Line Size

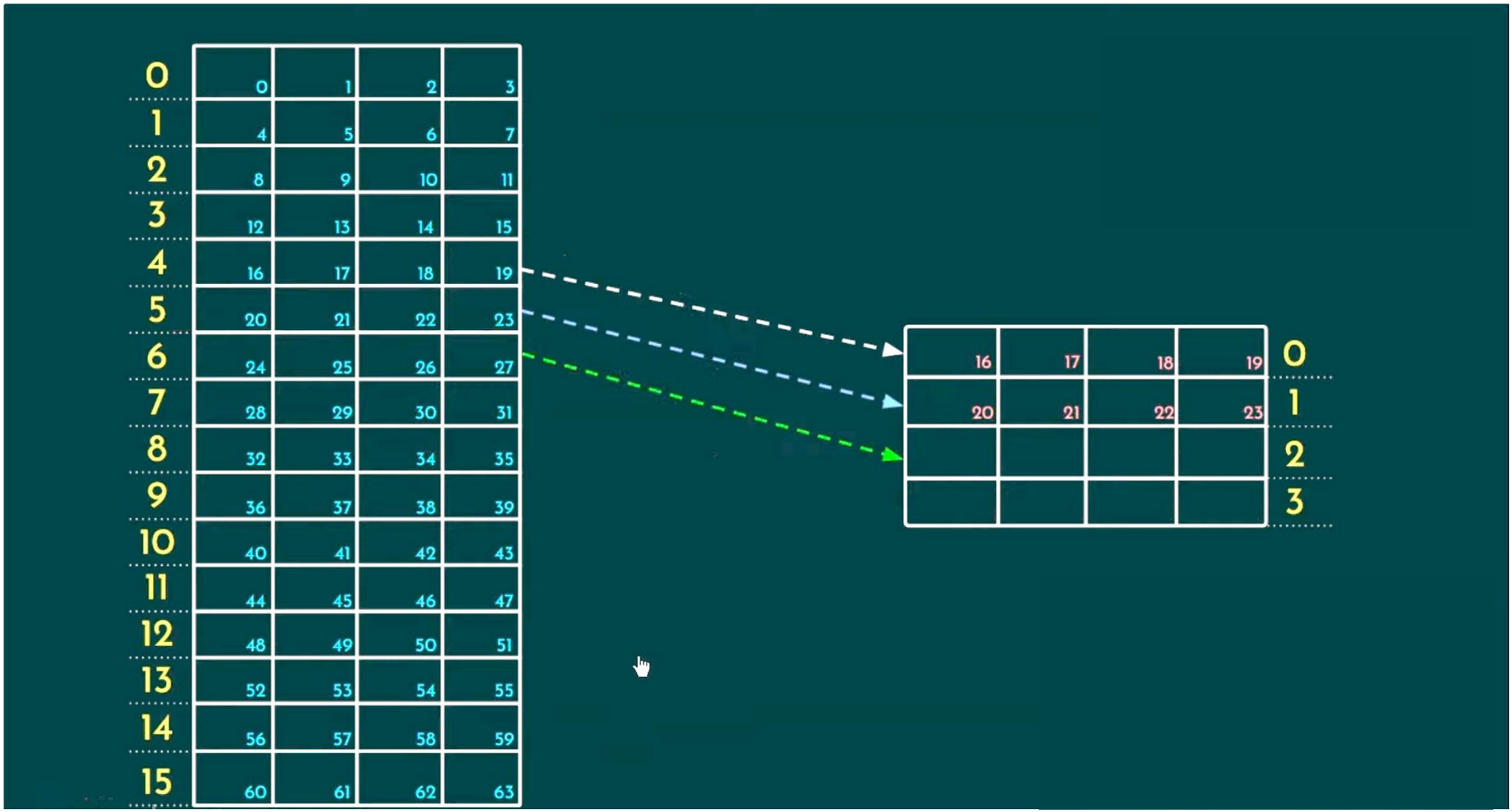
4 Lines

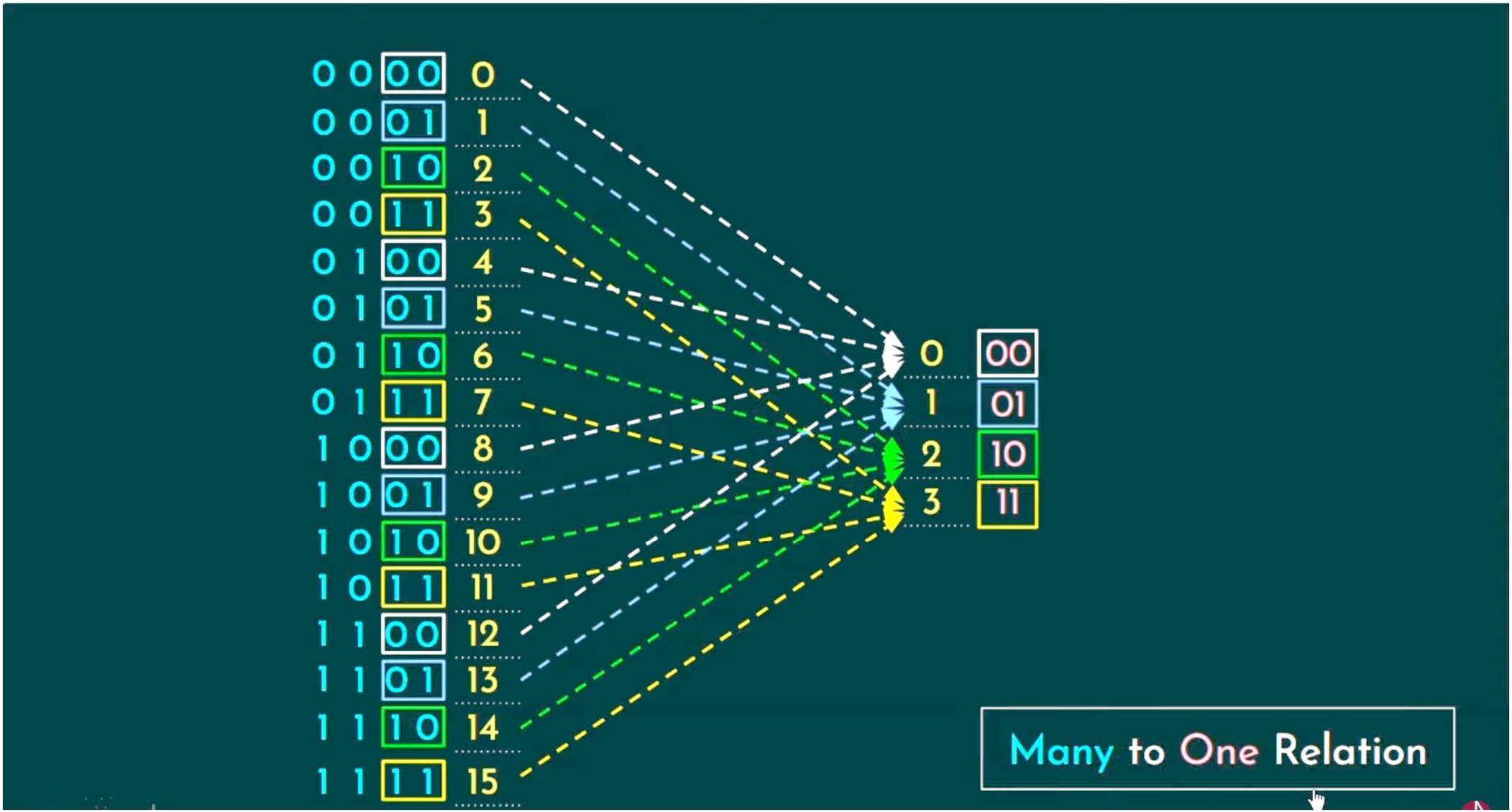
$\log_2 4 = \log_2 2^2$

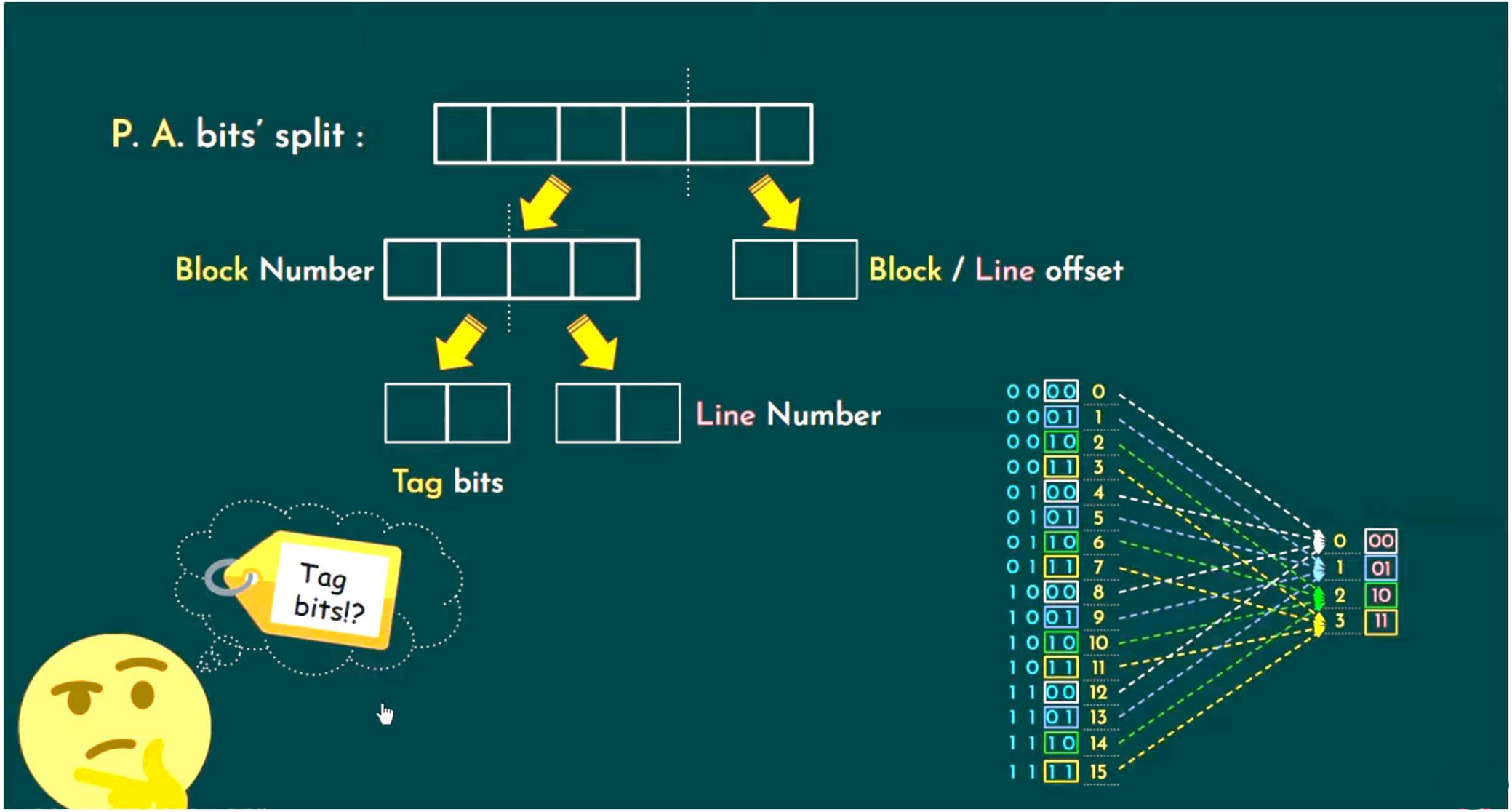
= 2 bits

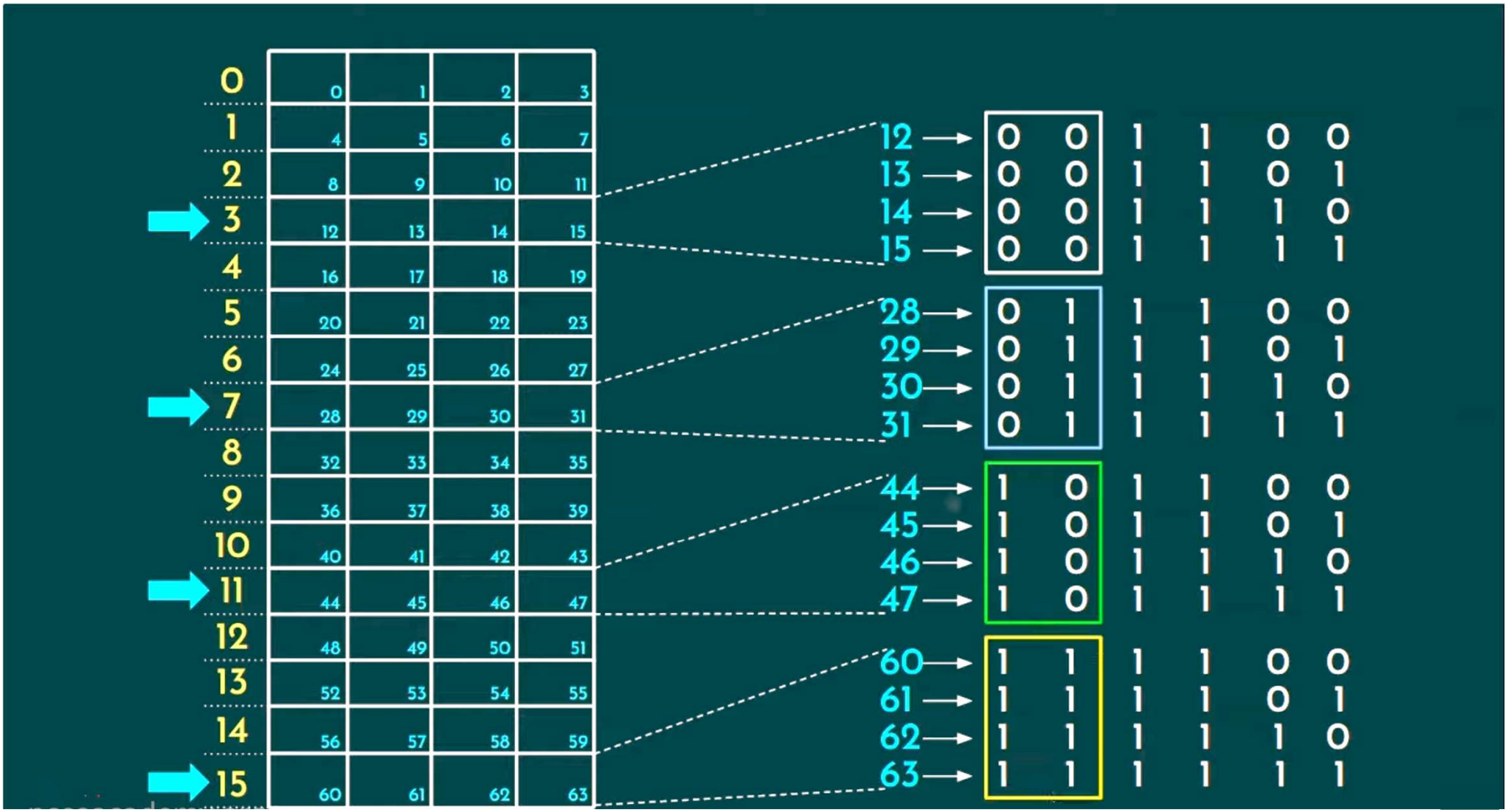



0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
16 17 18 19
20 21 22 23
24 25 26 27
28 29 30 31
32 33 34 35
36 37 38 39
40 41 42 43
44 45 46 47
48 49 50 51
52 53 54 55
56 57 58 59
60 61 62 63
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
0 1 2 3


0 1 2 3
0 0 1 2 3
1 4 5 6 7
2 8 9 10 11
3 12 13 14 15
4 16 17 18 19
5 20 21 22 23
6 24 25 26 27
7 28 29 30 31
8 32 33 34 35
9 36 37 38 39
10 40 41 42 43
11 44 45 46 47
12 48 49 50 51
13 52 53 54 55
14 56 57 58 59
15 60 61 62 63
16 17 18 19
20 21 22 23
0 1 2 3


0 0 0 0 0
0 0 0 1 1
0 0 1 0 2
0 0 1 1 3
0 1 0 0 4
0 1 0 1 5
0 1 1 0 6
0 1 1 1 7
1 0 0 0 8
1 0 0 1 9
1 0 1 0 10
1 0 1 1 11
1 1 0 0 12
1 1 0 1 13
1 1 1 0 14
1 1 1 1 15
0 00
1 01
2 10
3 11
Many to One Relation


P. A. bits' split :
Block Number
Block / Line offset
Line Number
Tag bits
Tag bits!?
0 0 00 0
0 0 01 1
0 0 10 2
0 0 11 3
0 1 00 4
0 1 01 5
0 1 10 6
0 1 11 7
1 0 00 8
1 0 01 9
1 0 10 10
1 0 11 11
1 1 00 12
1 1 01 13
1 1 10 14
1 1 11 15
0 00
1 01
2 10
3 11


0 0 1 2 3
1 4 5 6 7
2 8 9 10 11
3 12 13 14 15
4 16 17 18 19
5 20 21 22 23
6 24 25 26 27
7 28 29 30 31
8 32 33 34 35
9 36 37 38 39
10 40 41 42 43
11 44 45 46 47
12 48 49 50 51
13 52 53 54 55
14 56 57 58 59
15 60 61 62 63
12 → 0 0 1 1 0 0
13 → 0 0 1 1 0 1
14 → 0 0 1 1 1 0
15 → 0 0 1 1 1 1
28 → 0 1 1 1 0 0
29 → 0 1 1 1 0 1
30 → 0 1 1 1 1 0
31 → 0 1 1 1 1 1
44 → 1 0 1 1 0 0
45 → 1 0 1 1 0 1
46 → 1 0 1 1 1 0
47 → 1 0 1 1 1 1
60 → 1 1 1 1 0 0
61 → 1 1 1 1 0 1
62 → 1 1 1 1 1 0
63 → 1 1 1 1 1 1

Ex 1: MM Size: 4 GB
Cache Size: 1 MB
Block Size: 4 KB

1. P.A. bits' split?
2. Tag directory size?

Sol. MM Size = 4 GB = $2^2 \times 2^{30}$ B = $2^{(2+30)}$ B = 2^{32} B

∴ No. of P.A. bits = $\log_2 2^{32}$ = 32

Block Size = 4 KB = $2^2 \times 2^{10}$ B = 2^{12} B

No. of Blocks in MM = $2^{32} / 2^{12}$ = 2^{20}

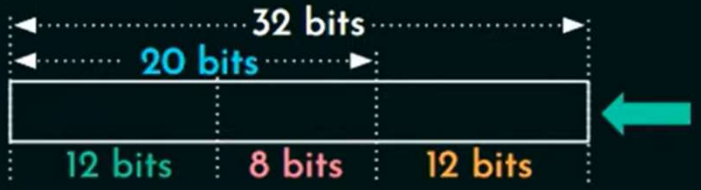
∴ Block number bits = $\log_2 2^{20}$ = 20

∴ Block offset = $\log_2 2^{12}$ = 12

Cache Size = 1 MB = 1×2^{20} B = 2^{20} B

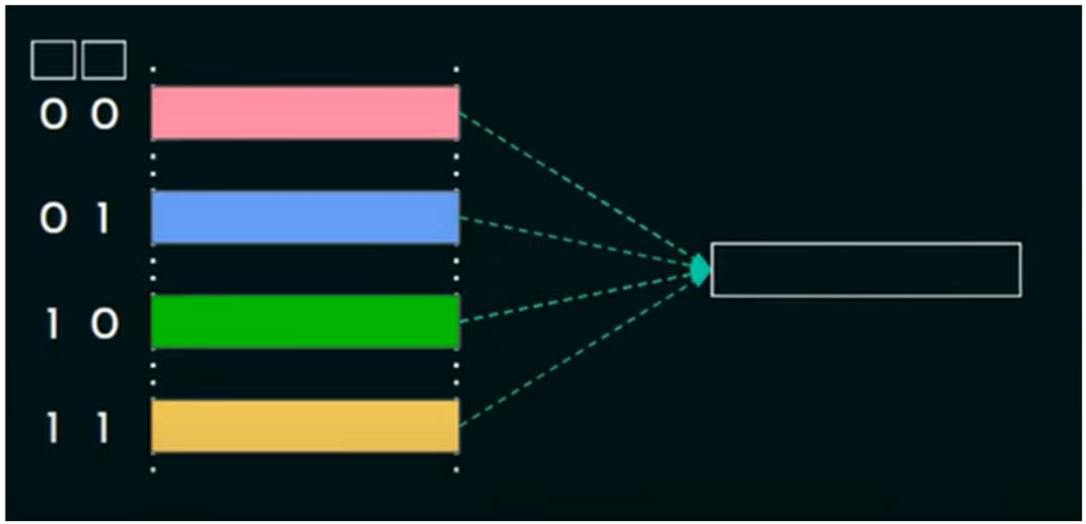
No. of Lines in Cache = $2^{20} / 2^{12}$ = 2^8

∴ Line number bits = $\log_2 2^8$ = 8



No. of Tag bits : P.A. bits - (Line no. bits + offset) = 32 - (8+12) = 12

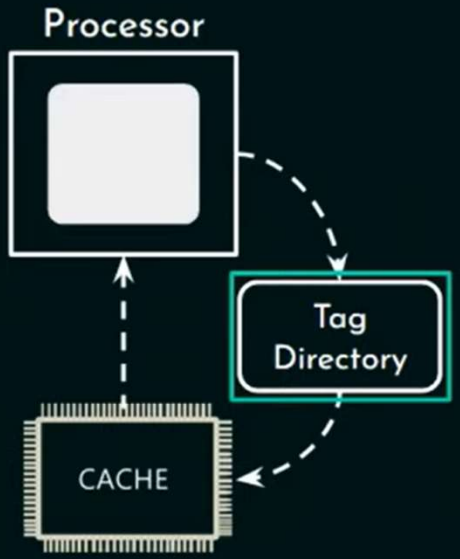
1 Byte	=	8 bits	
1 KB	=	1024 B(ytes)	= 2^{10} B
1 MB	=	1024 KB	= 2^{20} B
1 GB	=	1024 MB	= 2^{30} B



Ex 1: MM Size: 4 GB
Cache Size: 1 MB
Block Size: 4 KB

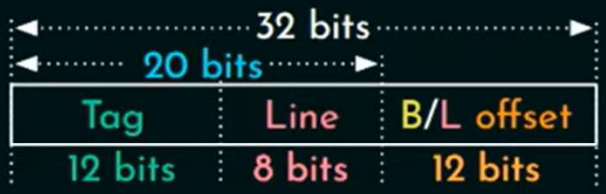
1. P.A. bits' split?
2. Tag directory size?

Sol.



Tag directory:

- Keeps primarily the record of Tag bits, cache line-wise.
- No. of entries = No. of Cache lines.



No. of Lines in Cache = 2^8

No. of Tag bits = 12

Tag directory size = 2^8 x 12 bits = 3072 bits

Ex 2: MM Size: 256 MB
Cache Size: 512 KB

- No. of tag bits?

Sol. Tag bits: Identifies the MM block residing in the Cache Line.

Block Size = Line Size

$\log_2$ (MM Size : Cache Size)

MM Size = 256 MB
$= 2^8 \times 2^{20}$ B
$= 2^{28}$ B

Cache Size = 512 KB
$= 2^9 \times 2^{10}$ B
$= 2^{19}$ B

$\therefore$ No. of Tag bits : $\log_2 (2^{28} / 2^{19}) = \text{Log}_2\, 2^{(28-19)}$
$= \text{Log}_2\, 2^9 = 9$

Ex 3: Byte-addressable MM Size: 16 GB
Block Size: 16 KB
No. of Tag bits: 10

- Cache size?

Sol. MM Size = 16 GB = $2^4 \times 2^{30}$ B = 2^{34} B

∴ No. of P.A. bits = $\log_2 2^{34}$ = 34

Block Size = 16 KB = $2^4 \times 2^{10}$ B = 2^{14} B

∴ Block offset = $\log_2 2^{14}$ = 14

No. of Line number bits : P.A. bits - (Tag bits + offset) = 34 - (10+14) = 10

∴ No. of Cache Lines = 2^{10}

Line Size = 2^{14} B

Cache Size = $2^{10} \times 2^{14}$ B = $2^{(10+14)}$ B = 2^{24} B = $2^4 \times 2^{20}$ B = 16 MB

Q1: A direct mapped cache memory of 1 MB has a block size of 256 bytes. The cache has an access time of 3 ns and a hit rate of 94%. During a cache miss, it takes 20 ns to bring the first word of a block from the main memory, while each subsequent word takes 5 ns. The word size is 64 bits. The average memory access time in ns (round off to 1 decimal place) is _____.

GATE CS 2020

Sol. Block Size = 256 B

Word Size = 64 bits = 8 x 8 bits = 8 B(ytes)

$\therefore$ No. of words per Block = 256 B / 8 B = $2^8 / 2^3 = 2^{(8-3)} = 2^5$

- 1 → 20 ns
- (2^5-1) → 5 ns (each)

$H_{cache} = 94\% = 0.94$

$T_{cache} = 3$ ns

T_{MM} = 20 ns for the 1st
5 ns for the rest
(for every block)

$T_{avg} = H_{cache} T_{cache} + (1-H_{cache}) (T_{cache} + T_{MM})$

$= 0.94 \times 3 + (1 - 0.94) \times (3 + ((20 \times 1) + ((2^5-1) \times 5)))$ ns

$= 2.82 + 0.06 \times (3 + (20 + (31 \times 5))$ ns

$= 2.82 + 0.06 \times (3+20+155)$ ns $= 2.82 + 0.06 \times 178$ ns

$= 2.82 + 10.68$ ns $= 13.5$ ns

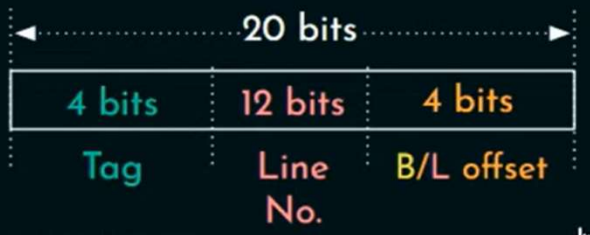
Q2: Consider a machine with a byte addressable main memory of 2^{20} bytes, block size of 16 bytes and a direct mapped cache having 2^{12} cache lines. Let the addresses of two consecutive bytes in main memory be $(E201F)_{16}$ and $(E2020)_{16}$. What are the tag and cache line address (in hex) for main memory address $(E201F)_{16}$?

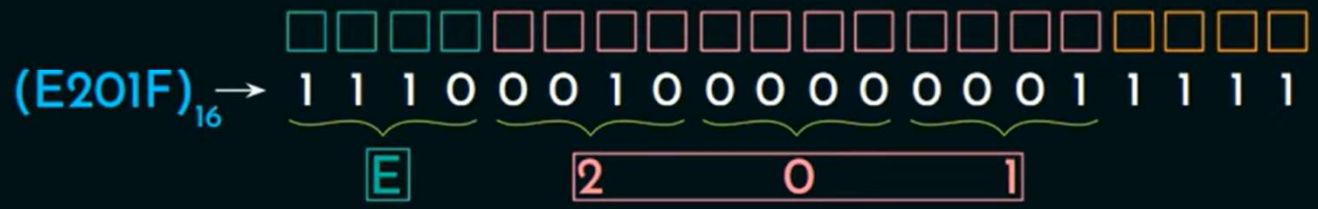
GATE CS 2015

Sol. MM Size = 2^{20} B $\quad \therefore$ No. of P.A. bits = 20

Block Size = 16 B $\quad \therefore$ Block offset = 4

= 2^4 B





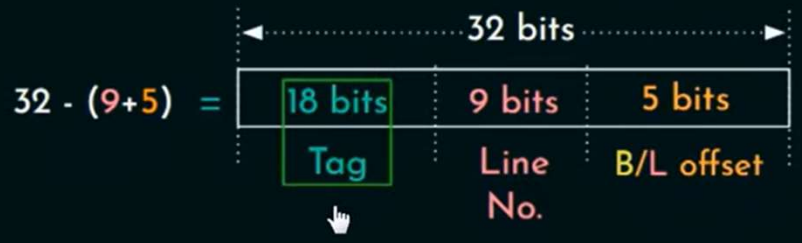
Q3: Consider a machine with byte addressable memory of 2^{32} bytes divided into blocks of size 32 bytes. Assume a direct mapped cache having 512 cache lines is used with this machine. The size of tag field in bits is _____.

GATE CS 2017

Sol. MM Size = 2^{32} B $\therefore$ No. of P.A. bits = 32

Block Size = 32 B = 2^5 B $\therefore$ Block offset = 5

No. of cache lines = 512 = 2^9 $\therefore$ Line no. bits = 9



Q4: An 8 KB direct-mapped write-back cache is organized as multiple blocks, each of size 32-bytes. The processor generates 32-bit addresses. The cache controller maintains the tag information for each cache block comprising of the following.
1 Valid bit, 1 Modified bit
As many bits as the minimum needed to identify the memory block mapped in the cache. What is the total size of memory needed at the cache controller to store meta-data (tags) for the cache?

GATE CS 2011

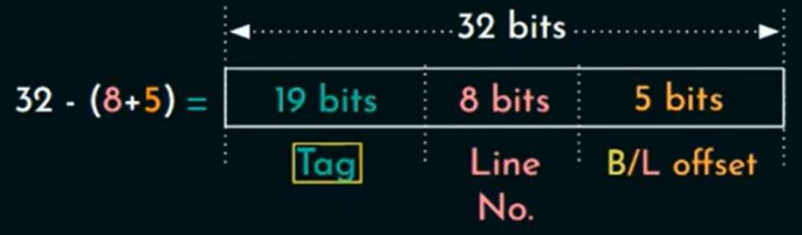
Sol. Cache Size = 8 KB = 2^{13} B

Block Size = 32 B = 2^5 B

No. of cache lines = $2^{13} / 2^5 = 2^8$

∴ Block offset = 5

∴ Line no. bits = 8

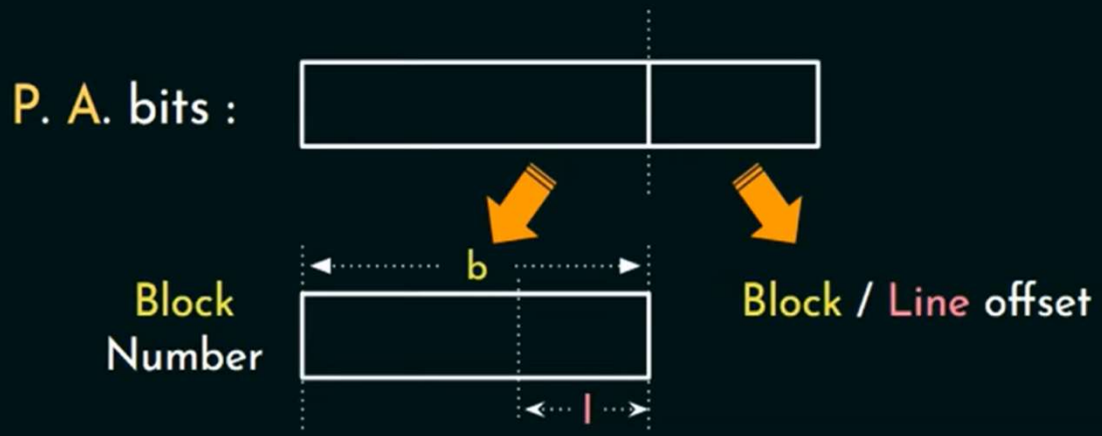


No. of P.A. bits = 32

Tag directory entry = 1 + 1 + 19 = 21

∴ Tag directory size = 21 x 2^8 = 21 x 256 = 5376 bits

Disadvantages of Direct memory Mapping

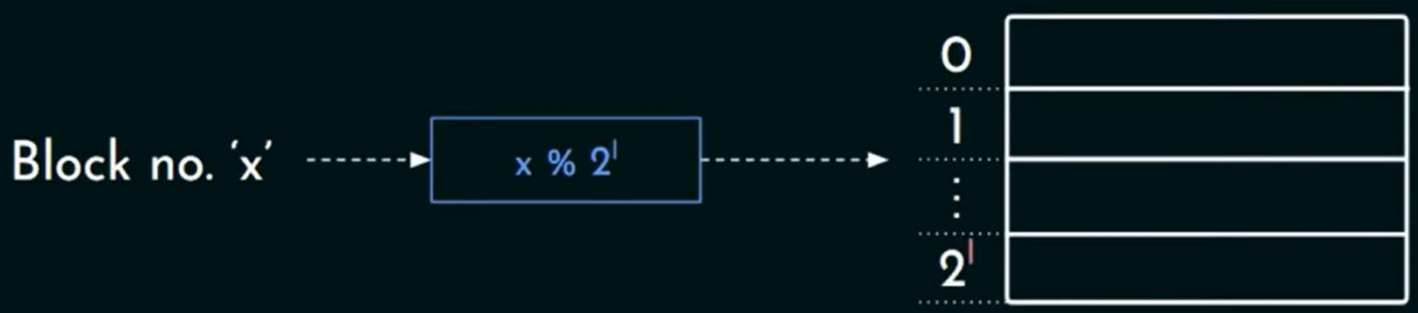


Block no. bits = 'b'

∴ No. of MM Blocks = 2^b

Line no. bits = 'l'

∴ No. of Cache lines = 2^l



"7, 8, 6, 10, 11, 14"

7 % 4 = 3 8 % 4 = 0 6 % 4 = 2

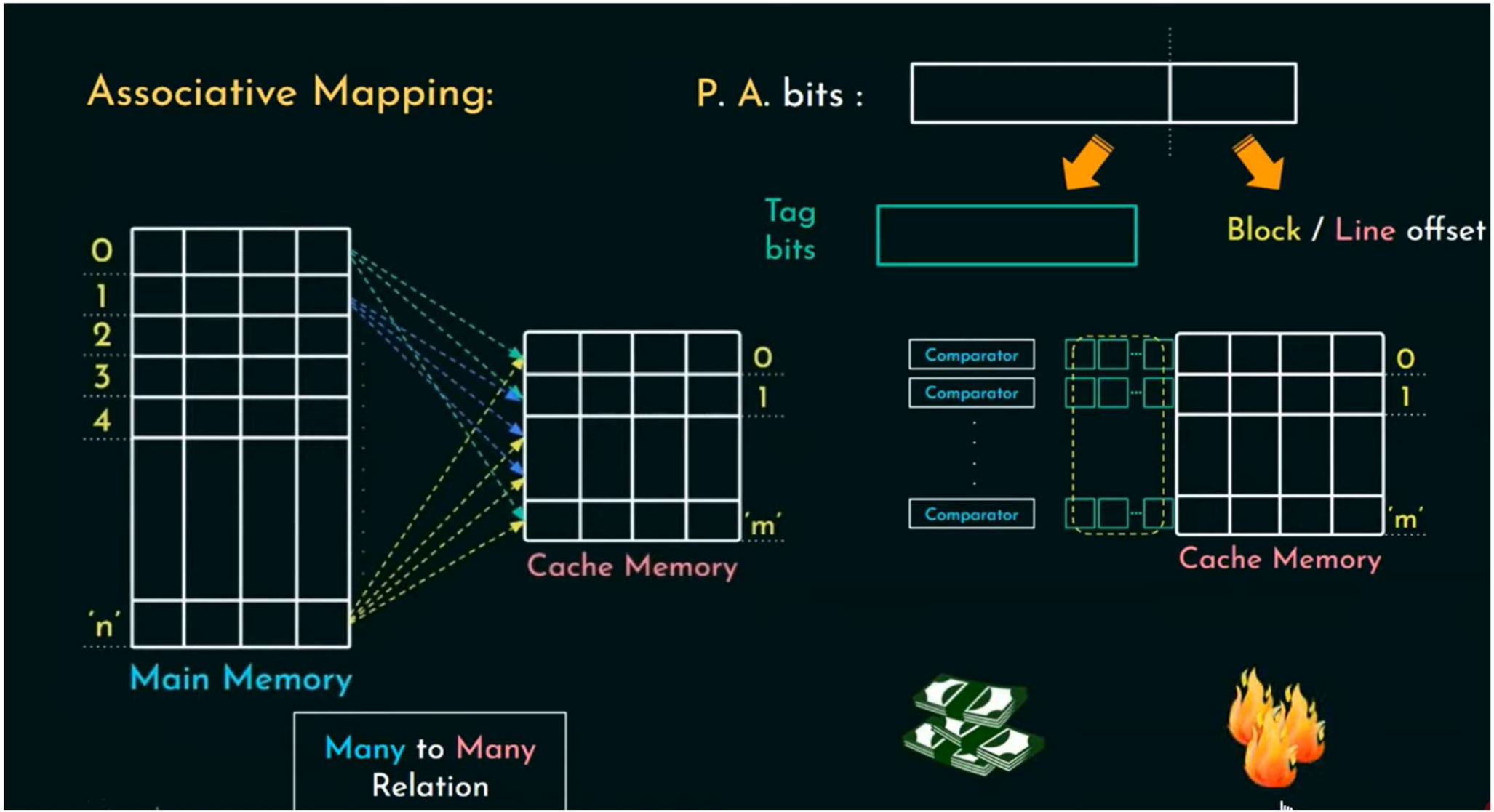
10 % 4 = 2 11 % 4 = 3 14 % 4 = 2

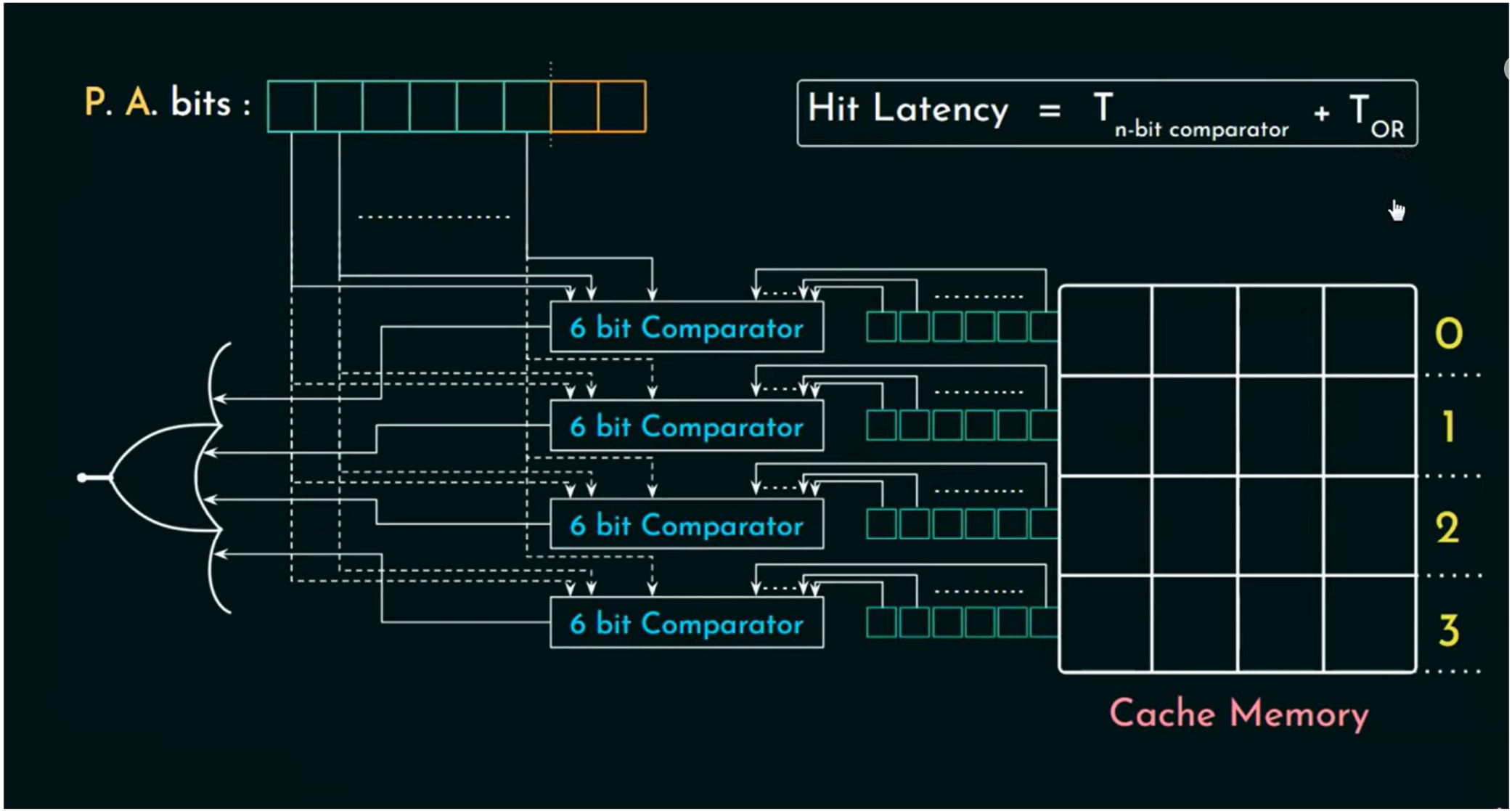
Conflict Miss

"2, 6, 10, 14, 18"

2 % 4 = 2 6 % 4 = 2 10 % 4 = 2

14 % 4 = 2 18 % 4 = 2


Associative Mapping:
P. A. bits :
Tag bits
Block / Line offset
0
1
2
3
4
'n'
Main Memory
0
1
'm'
Cache Memory
Comparator
Comparator
Comparator
0
1
'm'
Cache Memory
Many to Many Relation


P. A. bits :
Hit Latency = T n-bit comparator + T OR
6 bit Comparator
6 bit Comparator
6 bit Comparator
6 bit Comparator
0
1
2
3
Cache Memory

Q: MM Size = 2 GB

Block Size = 2 KB

Comparator Delay = 15n nanoseconds.

Delay of Multi-input OR gate = 7 nanoseconds.

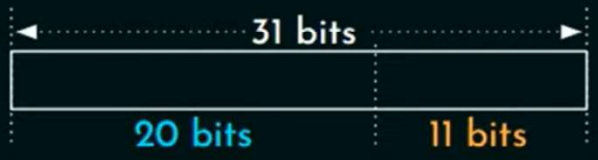
- Hit latency = ?

Sol. MM Size = $2^1 \times 2^{30} = 2^{31}$ B ∴ P.A. bits = 31

Block Size = 2^{11} B ∴ Block offset = 11

No of Tag bits = (31 - 11) = 20

Hit Latency = (15 x 20) + 7 = 307 nsec



Q: A certain processor uses a fully associative cache of size 16 kB, The cache block size is 16 bytes. Assume that the main memory is byte addressable and uses a 32-bit address. How many bits are required for the Tag and the Index fields respectively in the addresses generated by the processor?

(A) 24 bits and 0 bits

(B) 28 bits and 4 bits

(C) 24 bits and 4 bits

(D) 28 bits and 0 bits

GATE CS 2019

Sol. Cache Size = 16 KB = $2^{(4+10)}$ B = 2^{14} B

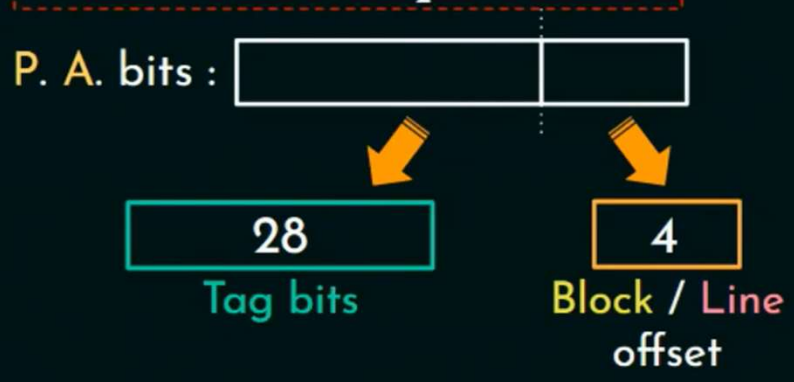
Block Size = 16 B = 2^4 B

∴ Block offset = 4

P. A. bits : 32

Tag bits = 32 - 4 = 28

∴ Index bits = $\log_2 2^{(14-4)}$ = 10

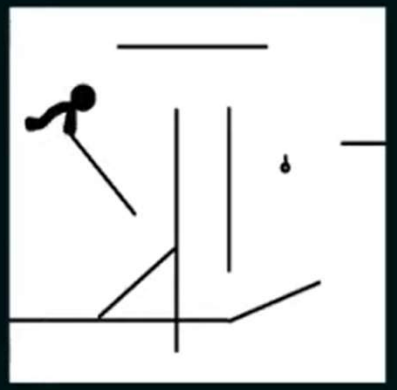


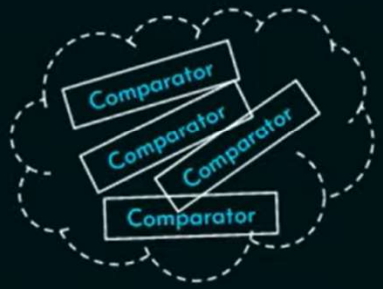
Direct Mapping

Direct Mapping

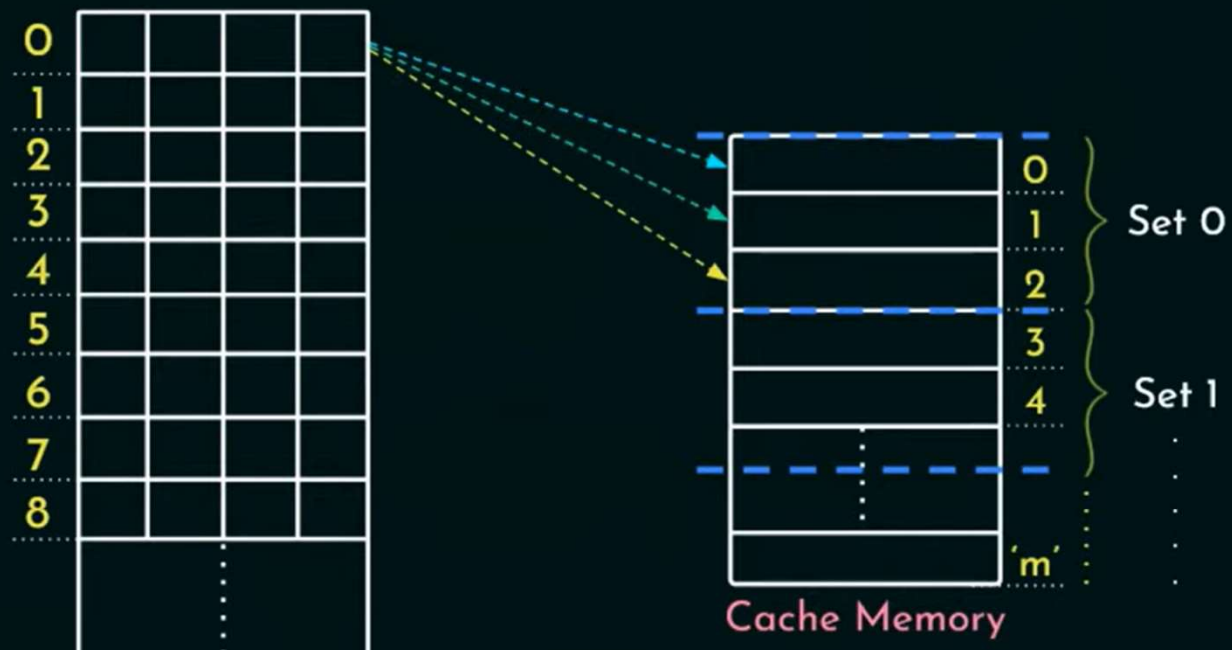
Associative Mapping

Associative Mapping





Set Associative Mapping:



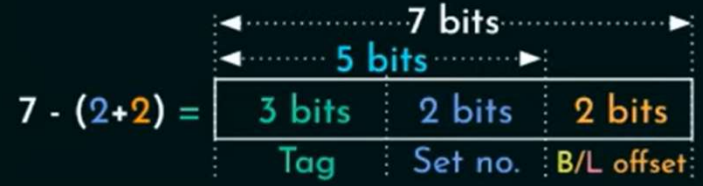
1 set = k lines

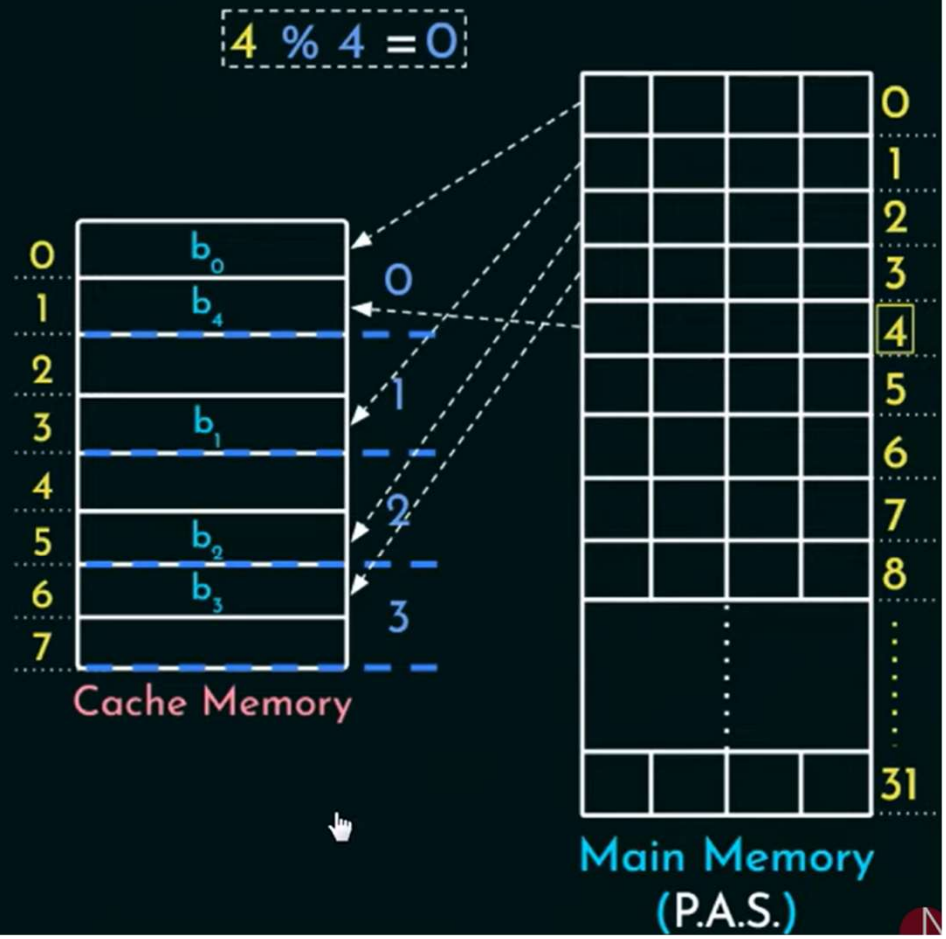
∴ k-way Set Associative

Ex : Byte addressable MM Size: 128 B
Cache Size: 32 B
Block Size: 4 B
2-way Set Associative | 1 set = 2 lines

P.A.S. = 128 B = 2^7 B ∴ P.A. bits = 7
Block Size = 2^2 B ∴ offset = 2
∴ No. of MM Blocks = $2^7 / 2^2$
= 2^5

Cache Size = 2^5 B
∴ No. of lines = $2^5 / 2^2 = 2^3$
∴ No. of sets = $2^3 / 2^1 = 2^2$





Cache Memory Mapping – A Comparative Study

Byte addressable MM Size: 8 GB $= 2^{33}$ B $\therefore$ No. of P.A. bits = 33

Block Size: 2 KB $= 2^{11}$ B $\therefore$ Block offset = 11

Cache Size: 4 MB $= 2^{22}$ B $\therefore$ No. of Cache Lines $= 2^{22} / 2^{11} = 2^{11}$

- Direct Mapping: $2^{33} / 2^{22} = 2^{11}$

- Associative Mapping: 33 - 11 =

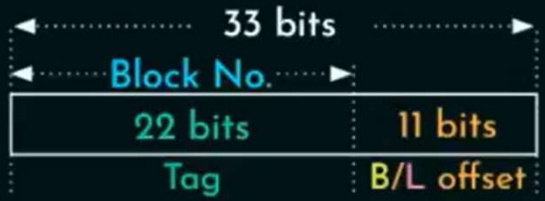
- "4-way" Set Associative Mapping: 33 - (9 + 11) =

→ 1 set = 4 lines = 2^2 lines

$\therefore$ # sets = $2^{11} / 2^2 = 2^9$

Direct Mapping

Associative Mapping



"4-way" Set Associative Mapping

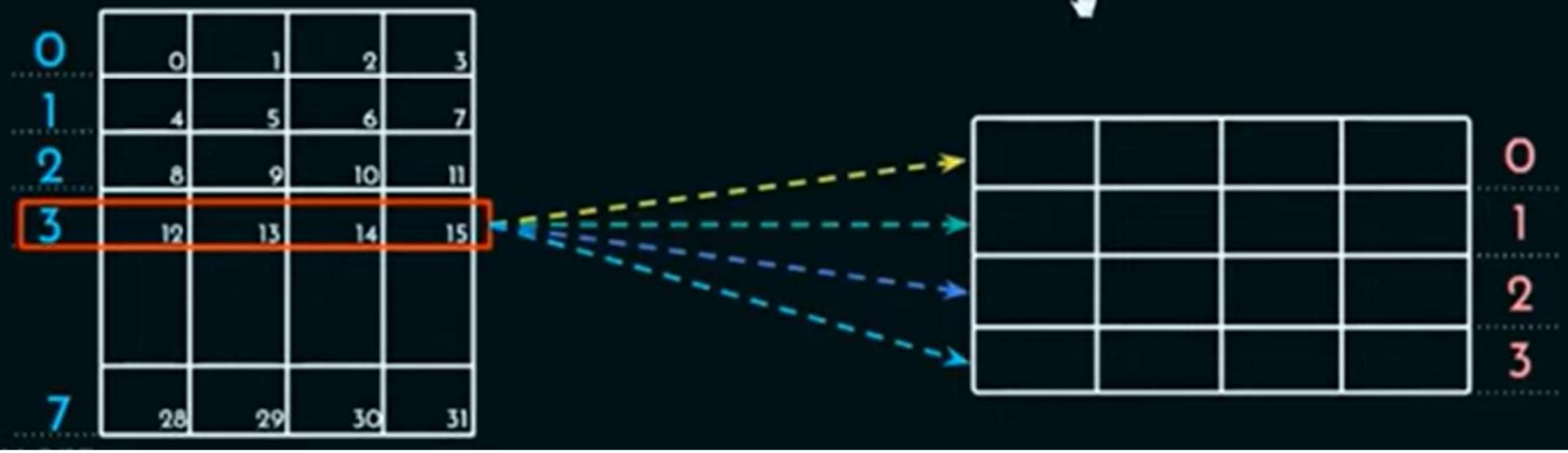
Cache Memory Mapping Techniques	Tag bits	# Cache lines	Tag Directory Size	# Comparators	Comparator Type
Direct Mapping	11	2^{11}	2^{11} x 11 bits	1	11 bit
Associative Mapping	22	2^{11}	2^{11} x 22 bits	2^{11}	22 bit
4-way Set Associative Mapping	13	2^{11}	2^{11} x 13 bits	4	13 bit

Cache Design

1. Block Placement:
    - Where to place the Main Memory Block in the Cache?
2. Block Identification:
    - How to find the Main Memory Block in the Cache?
3. Block Replacement:
    - During a Cache Miss, how to choose which entry to replace from the Cache?
4. Write Strategy:
    - How are the updations propagated?

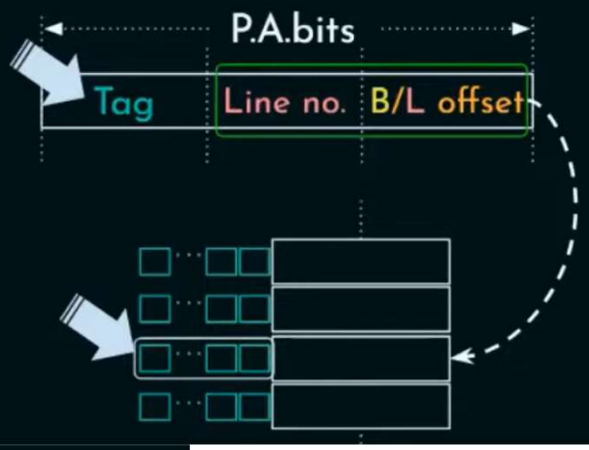
1. Block Placement:

- Direct Mapping: Block No. mod #lines
- Set Associative Mapping: Block No. mod #sets
- Associative Mapping: Anywhere!!

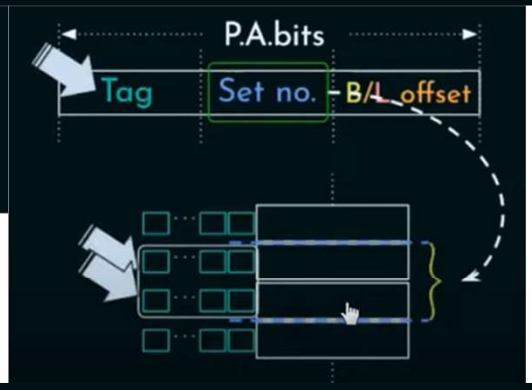


2. Block Identification:

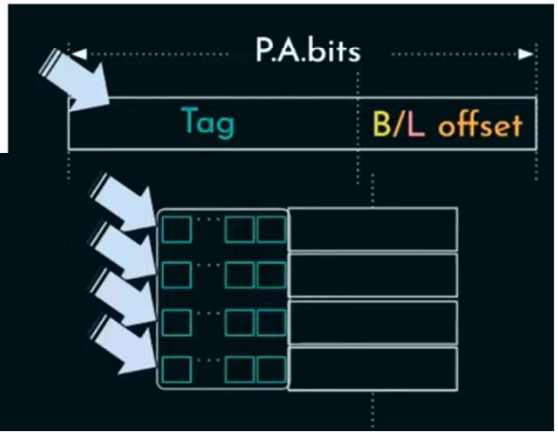
- **Direct Mapping:**
  - ✓ Find the potential match using Line no. & Offset bits.
  - ✓ Compare the Tag bits.



- **Set Associative Mapping:**
  - ✓ Find the set using Set Index.
  - ✓ Compare the Tag bits parallely.



- **Associative Mapping:**
  - ✓ Find the potential match comparing all the Tag bits associated to every line, simultaneously.

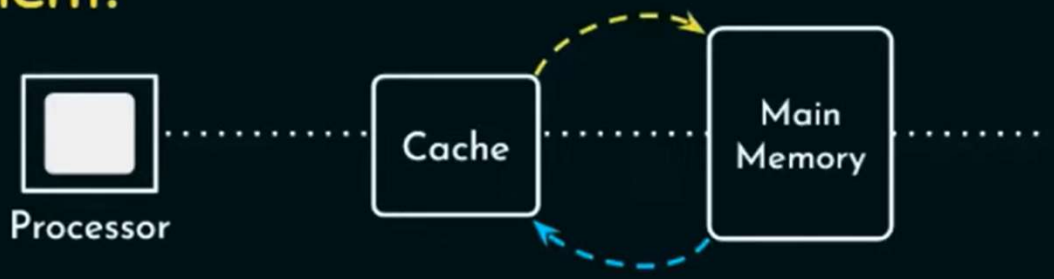


3. Block Replacement:

- Cache is Limited in Size.
- What should be done,
  - a. When the cache is full? a.k.a. Capacity Miss
  - b. When the potential match can't be found?
    - i. Compulsory Miss ii. Conflict Miss

✓ Replace a "Block" residing in cache with the new block request & move the replaced "Block" into the next level of Memory Hierarchy.

3. Block Replacement:



- Which "Block" to replace?

✓ Cache Replacement Policies:

a. Random Replacement
b. FIFO & LIFO
c. Recency Based Policies
d. Frequency Based Policies
e. Optimal Replacement/Belady's Optimal Algorithm

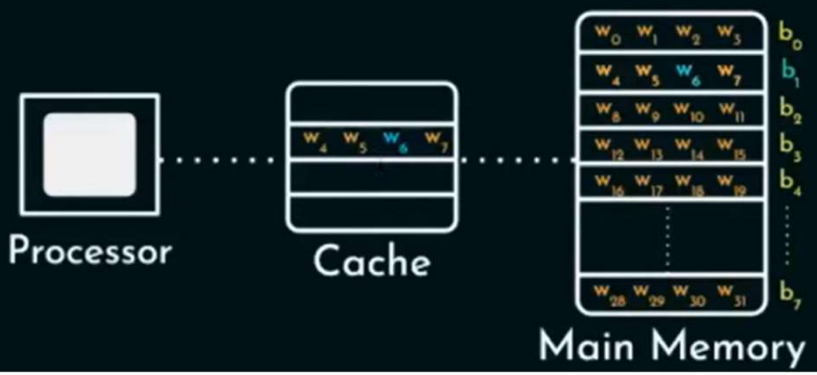
WRITE STRATEGY

4. Write Strategy:

- When?
  - ✓ Processor needs to modify data word.
- Situations:
  1. Write Hit: The data is present in the cache.
     - 1.1. Write Through:
       - -- Cache & Main Memory are updated simultaneously.
       - -- Used during lesser Write operations.

📌 Reliable & helps in Data Recovery.

📌 Delayed Data writes.



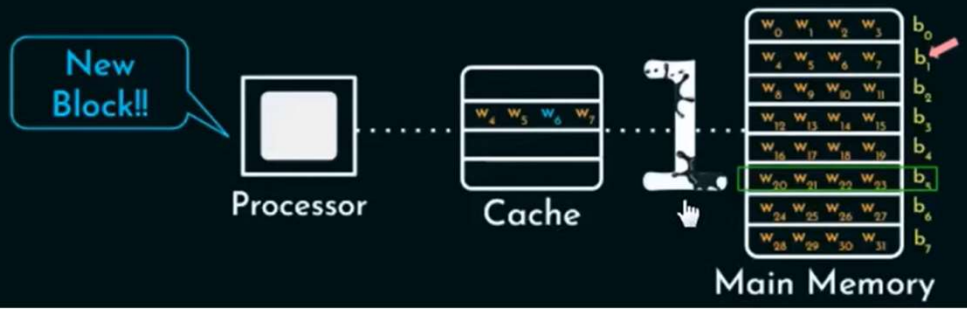
Situations:

1. Write Hit: The data is present in the cache.

   1.2. Write Back / Write Deferred:

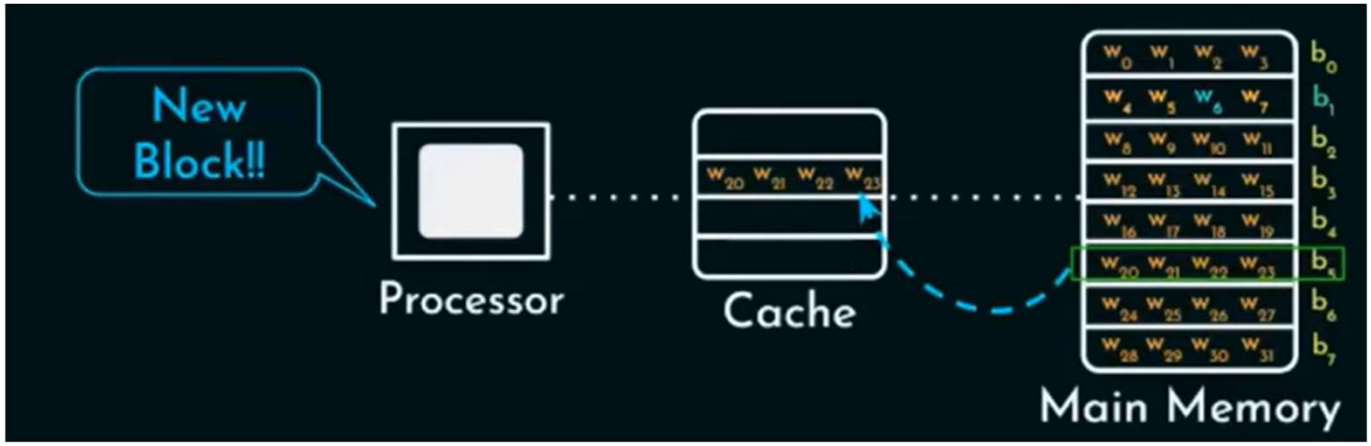
   -- Cache is updated in real time & uses "Dirty-bit".

   -- Main Memory is updated when replacement takes place.

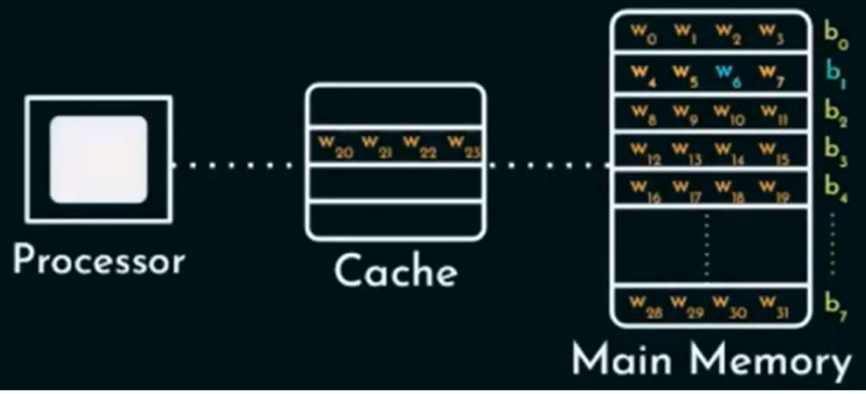


Faster.

Data recovery is impossible.



- Situations:
  2. Write Miss: The data is absent from the cache.
     2.1. Write Allocate:
          -- Data is brought into the cache first, then updated.
          -- Can work equally with Write-through & Write-back.

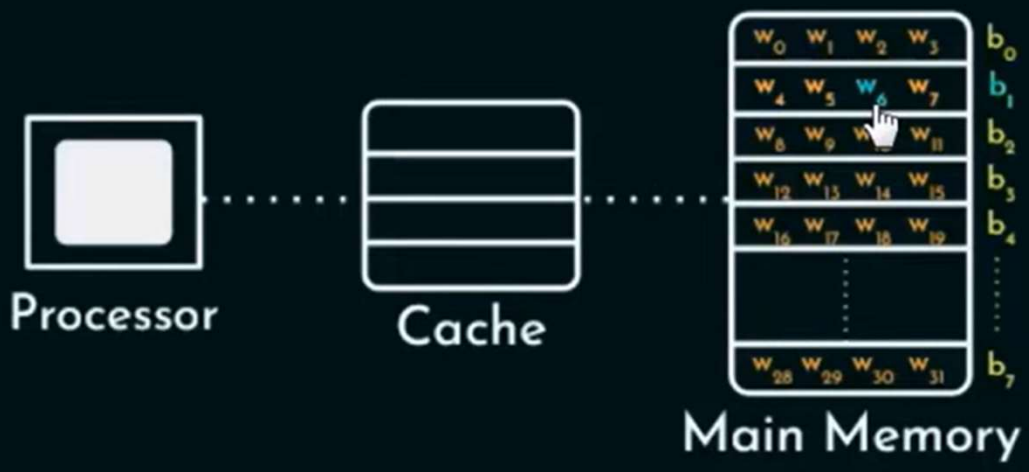


- Situations:

2. Write Miss: The data is absent from the cache.

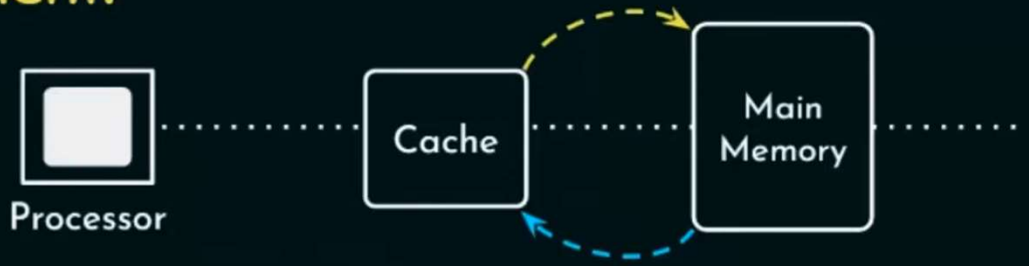
2.2. No-Write Allocate:

-- Data is updated directly in the Main Memory.



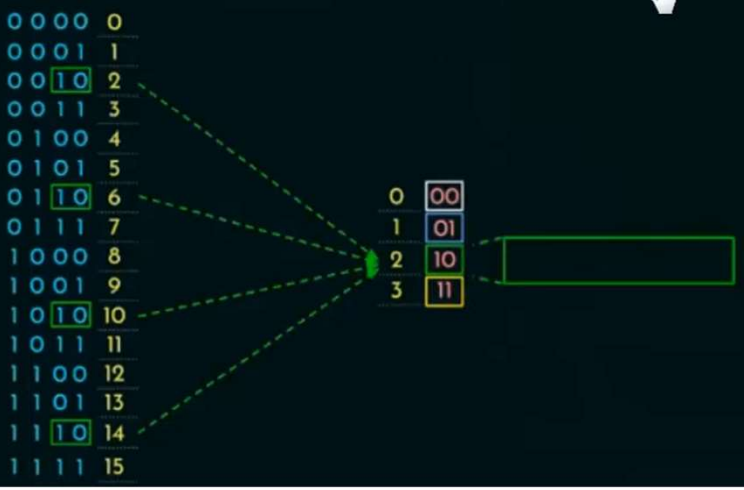
Cache Replacement Policies - RR, FIFO, LIFO, & Optimal

Block Replacement:

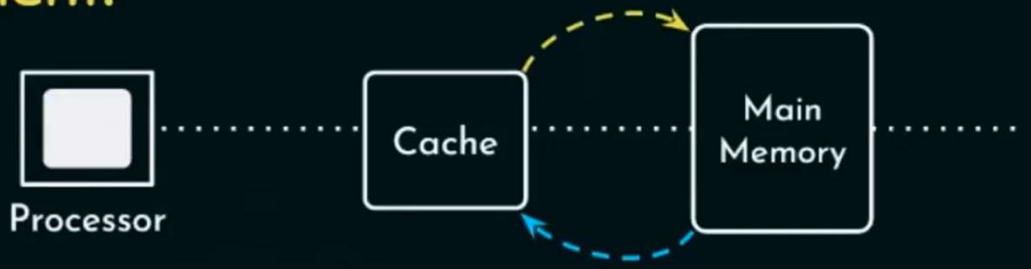


- Which "Block" to replace?

➢ Direct mapping: -- No need for decision!!



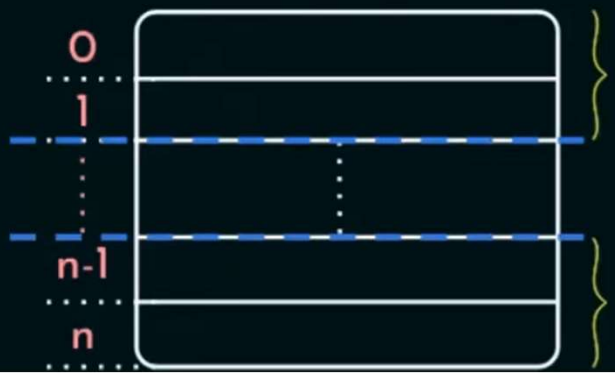
Block Replacement:



- Which "Block" to replace?

➢ Direct mapping: -- No need for decision!!

➢ Associative & Set Associative mapping:



-- Availability of choices.

✓ Cache Replacement Policies

➡ Reduce Cache Misses.

➡ Minimize "miss penalty".

a. Random Replacement:

-- Evict any block from cache at random.

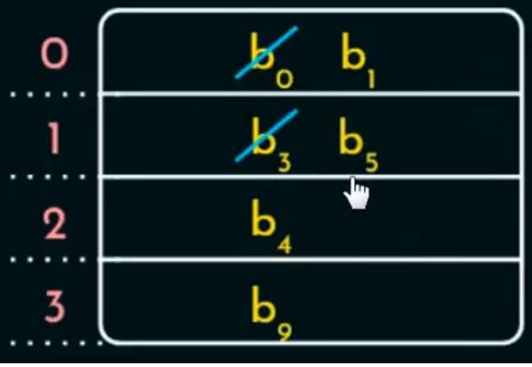
-- Access information is not needed.

-- Not implemented. (used to be implemented in ARM architectures)

b. FIFO:

-- Evicts blocks from cache in their order of arrival.

-- Cache behaves as a First-In-First-Out queue.



Q: Consider a fully associative cache memory with 4 lines that implements FIFO cache replacement policy. For the following block requests,

2, 3, 4, 7, 6, 3, 4, 7, 5, 4, 7, 8

What is Miss and Hit ratio, respectively?

Sol.

$$\text{Miss Ratio} = \frac{\text{No. of Misses}}{\text{Total no. of Requests}} \times 100\%$$

$$= \frac{7}{12} \times 100\% = 58.33\%$$

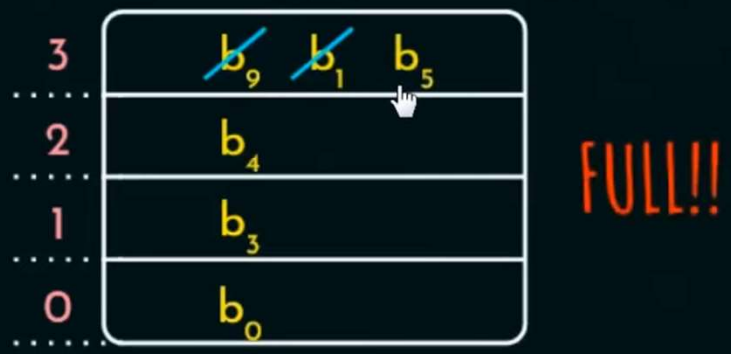
$$\therefore \text{Hit Ratio} = (100 - 58.33)\% = 41.66\%$$

No. of Misses : 7 = 4 Compulsory Misses + 3 Capacity Misses

b. LIFO:

-- Evicts the most recently added block i.e. Last-In-First-Out.

-- Cache behaves as a STACK.



c. Optimal Replacement / Belady's Optimal Algorithm:

-- Evicts the block that won't be referred for the longest period of time in future.

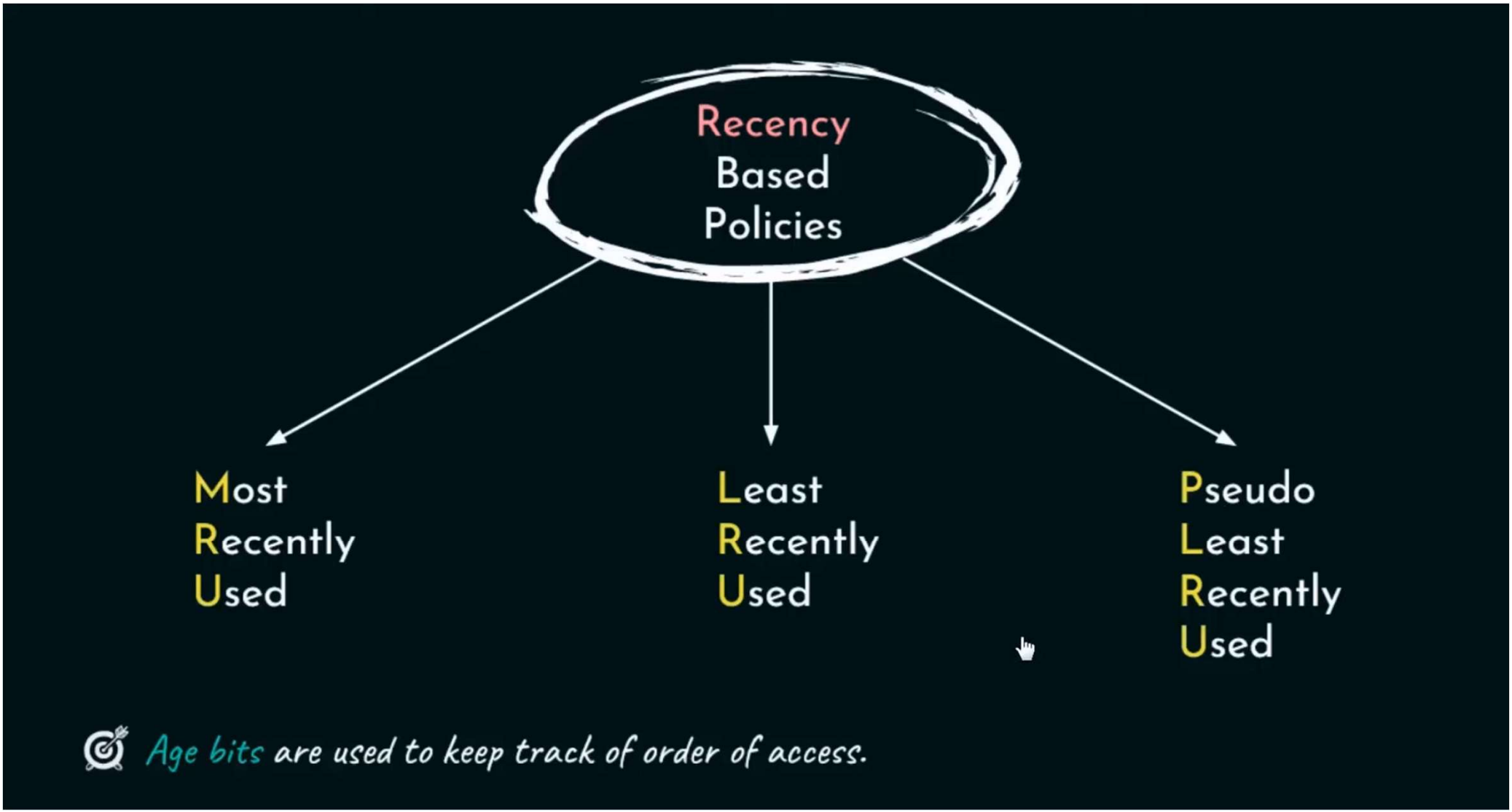
-- Prediction of Block requests is IMPOSSIBLE!!

Block Requests : 2, 3, 4, 7, 6, 3, 4, 7, 5, 4, 7, 8

-- Can't be Implemented!!

-- Widely used as a efficiency measuring tool for real Replacement Algorithms.

-- Proposed by L. A. Belady in 1966.

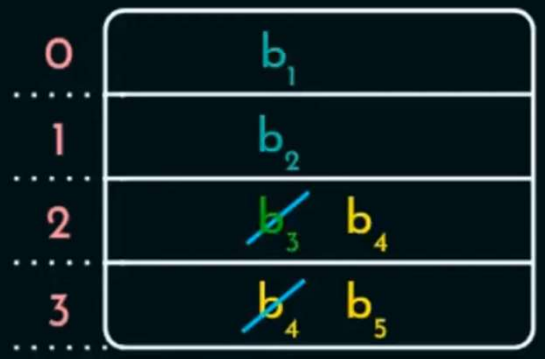

Recency
Based
Policies
Most
Recently
Used
Least
Recently
Used
Pseudo
Least
Recently
Used
Age bits are used to keep track of order of access.

a. Most Recently Used:

-- Evicts most recently referred block.

-- Works well with cyclic patterns.

Block Requests : 1, 2, 3, 4, 5, 1, 2, 3, 4, 1, 2



b. Least Recently Used:

-- Exploits Temporal Locality.

-- Evicts least recently referred block.

Block Requests : 0, 1, 2, 3, 4, 2, 3, 1, 5, 6

LRU

b_3

b_1

b_5

b_6

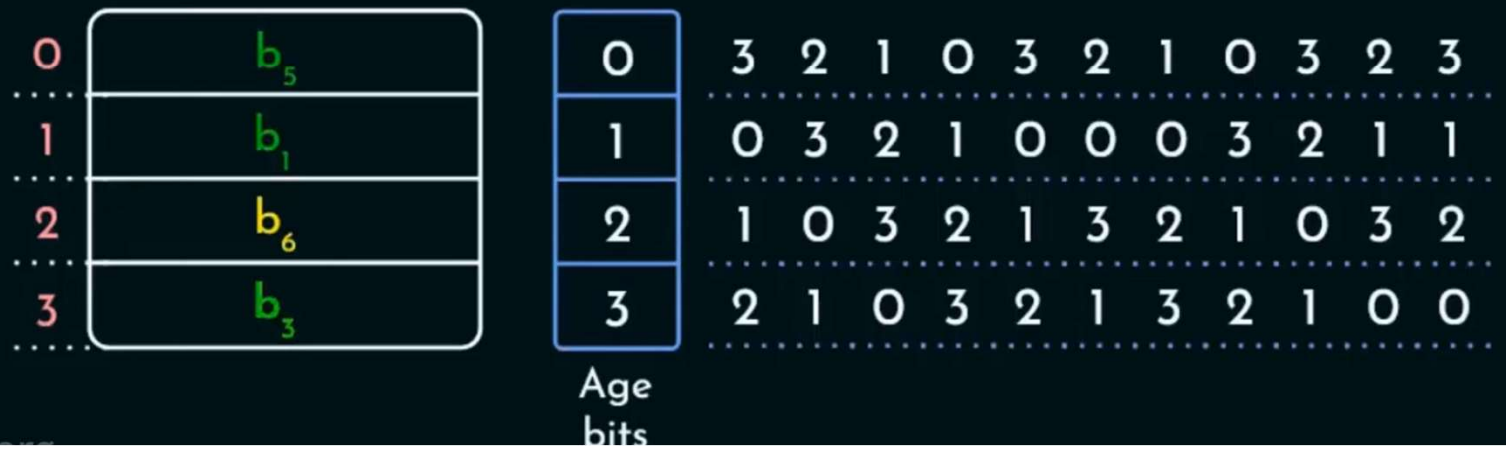
MRU

b. Least Recently Used:

-- Exploits Temporal Locality.

-- Evicts least recently referred block.

Block Requests : 0, 1, 2, 3, 4, 2, 3, 1, 5, 6, 5



b. Least Recently Used:

-- Exploits Temporal Locality.

-- Evicts least recently referred block.

-- Rigorous use of Age bits.

Block Requests : 0, 1, 2, 3, 4, 2, 3, 1, 5, 6, 5

Age bits		
2	(0, 1, 2, 3)	4
1	(0, 1, 3)	3
3	(0, 3)	2
0	(0)	1

Total no. of sequences

$= 4 \times 3 \times 2 \times 1 = 4!$

$\lceil \log_2(4!) \rceil = 5$

$\lceil \log_2(8!) \rceil = 16$

b. Least Recently Used:

-- Exploits Temporal Locality.

-- Evicts least recently referred block.

-- Rigorous use of Age bits & sequence of Age bits.

HUGE OVERHEAD for Caches with Higher Associativity

Block Requests : 0, 1, 2, 3, 4, 2, 3, 1, 5, 6, 5

Total no. of sequences

$= 4 \times 3 \times 2 \times 1 = 4!$

$\lceil \log_2(4!) \rceil = 5$ (per set)

$\lceil \log_2(8!) \rceil = 16$ (per set)

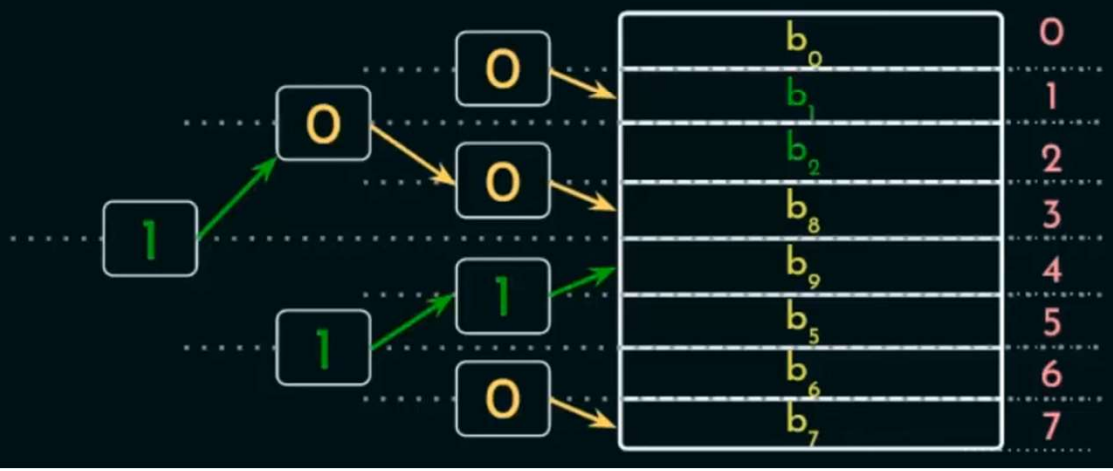
c. Pseudo-Least Recently Used:

-- Generates approximate measures for replacements.

Block Requests : 0, 1, 2, 3, 4, 5, 6, 7, 2, 1, 9, 8

0 → Down

1 → Up



➡ Frequency Based Policy:

- **Least Frequently Used:**

  -- Evicts least frequently referred block.

  -- Frequency is recorded for all blocks present in Cache.

Block Requests : 0, 1, 2, 3, 4, 2, 3, 1, 5, 6

Frequency

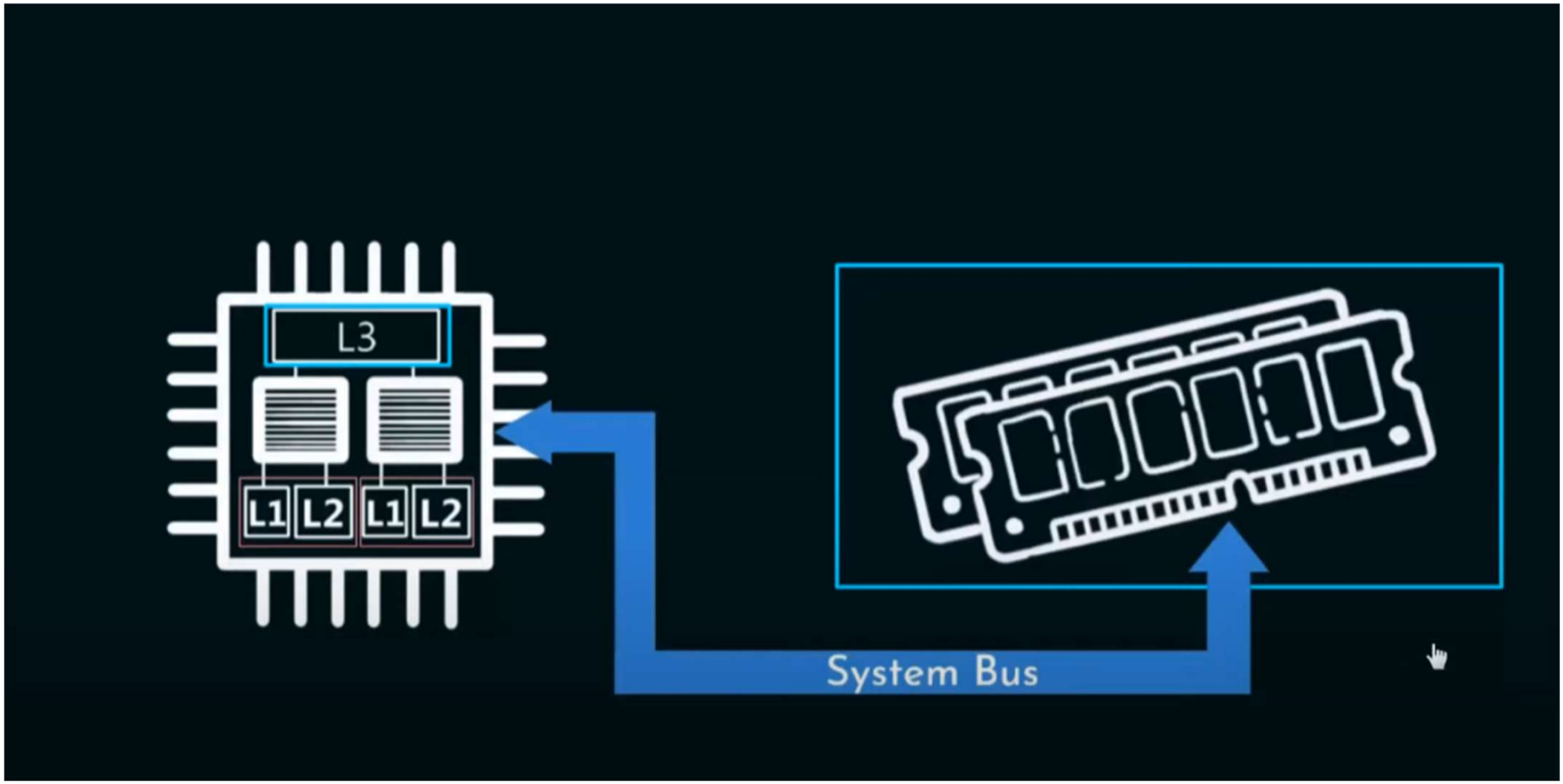
$b_6 = 1$

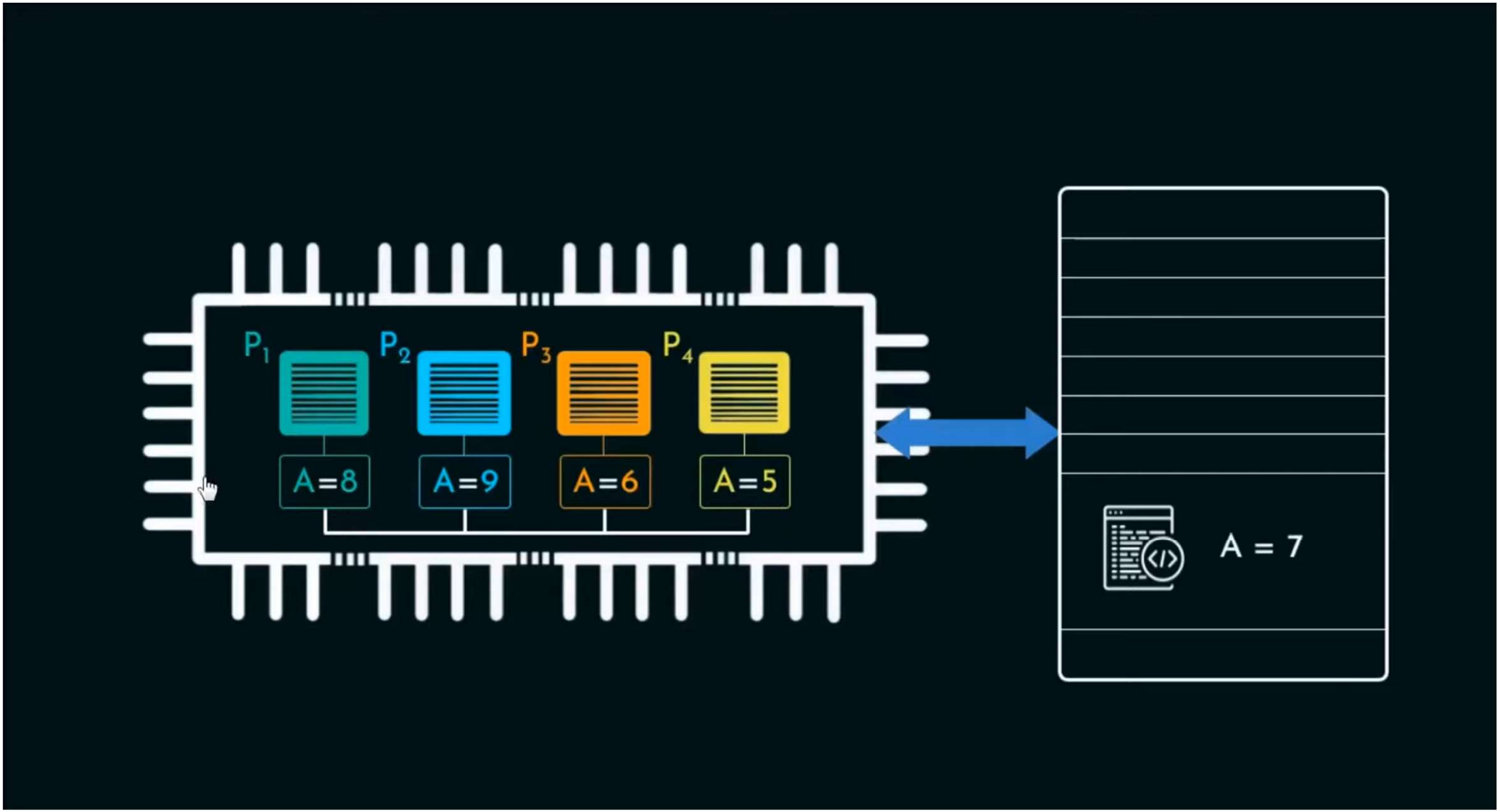
$b_1 = 2$

$b_2 = 2$

$b_3 = 2$

Cache Coherence Problem & Cache Coherency Protocols




P1
P2
P3
P4
A=8
A=9
A=6
A=5
A = 7

Cache Coherence:

- It is the uniformity of shared resource data that ends up stored in multiple local caches.

Cache Coherence Problem:

- It is the challenge of keeping multiple local caches synchronized when one of the processors updates its local copy of data which is shared among multiple caches.