Contemporary Cache Structure

- **Cache** connects to the processor via: data, address and control lines
- **Data and Address lines** also attach to **data and address buffers:**
  - These in turn attach to **system bus** to reach the main memory

Contemporary Cache Structure

- When a cache **hit** occurs:
  - Data and Address Buffers **disabled**
  - **Communication** between **processor** and **cache**
- When a cache **miss** occurs:
  - **Data** is loaded from **main memory** on the **data buffers** available on **system bus**
  - **Word** transferred to **cache** and **processor**

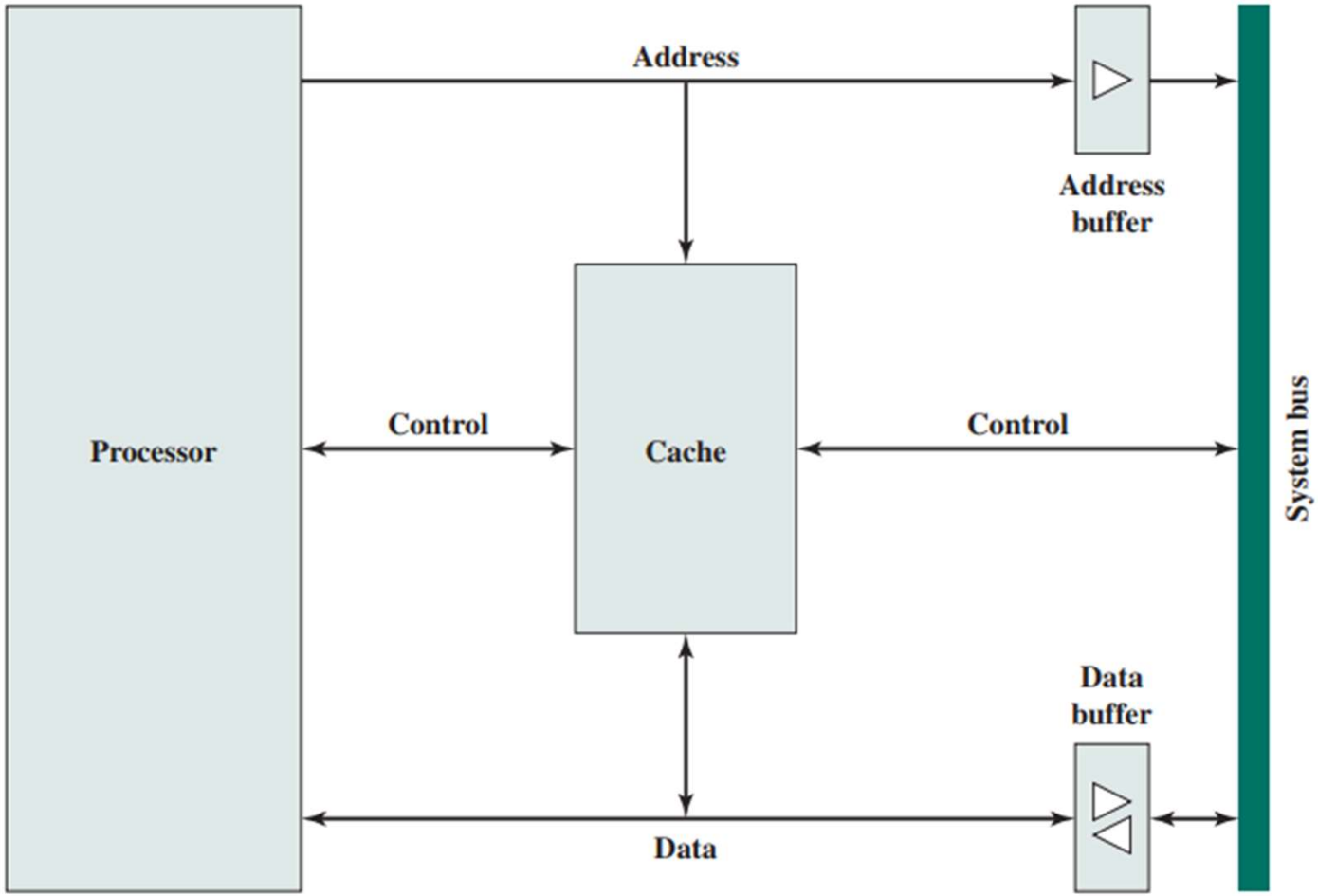


Figure 4.6 Typical Cache Organization

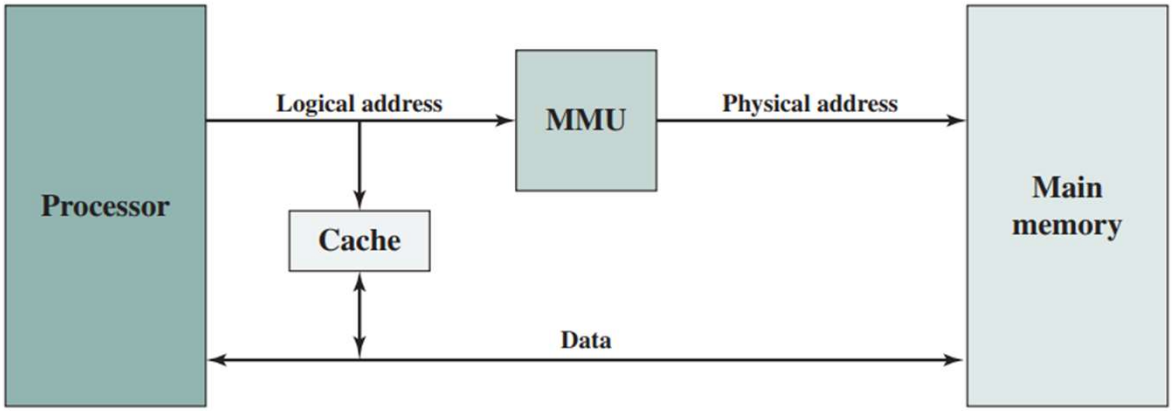
4.3 Elements of Cache Design

Elements of cache design

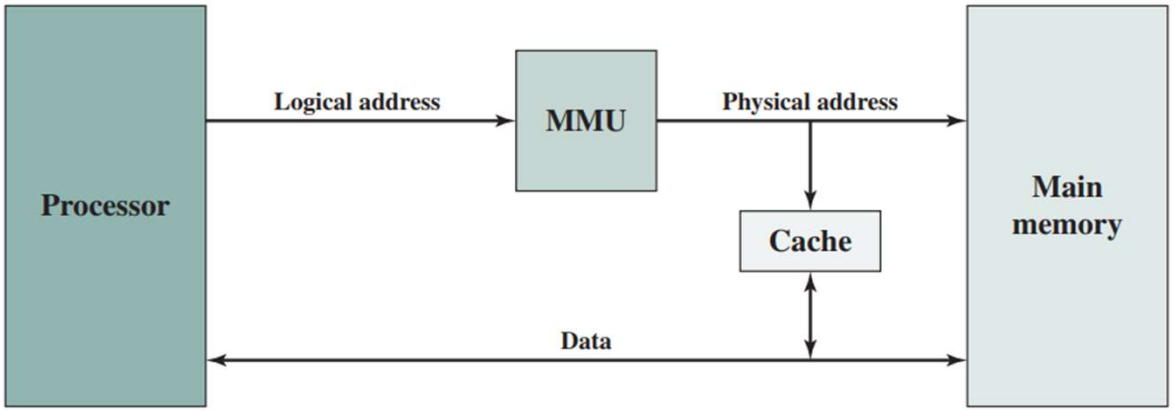
- **Cache Address:**
  - Majority of Modern computers support virtual **memory**
  - With **Virtual memory:**
    - Address field contains virtual address
    - Then how to read/write to and from physical memory?
    - **MMU** a hardware component:
      - Translates each virtual address to physical address

Elements of cache design

- **Cache Address:**
  - With virtual addresses:
    - Cache may be placed between processor and MMU or may be between Processor and Memory



(a) Logical cache



(b) Physical cache

Figure 4.7 Logical and Physical Caches

Elements of cache design

- **Logical cache (Virtual):**
  - Stores data using virtual addresses
  - Directly accessed via processor
  - No need to go through MMU
  - Faster
- **Physical cache:**
  - Stores data using physical addresses
  - slower

Elements of cache design

- **Cache Size:**
  - Should be as minimum as possible
  - Larger the cache, larger the circuit, slower access time

Elements of cache design

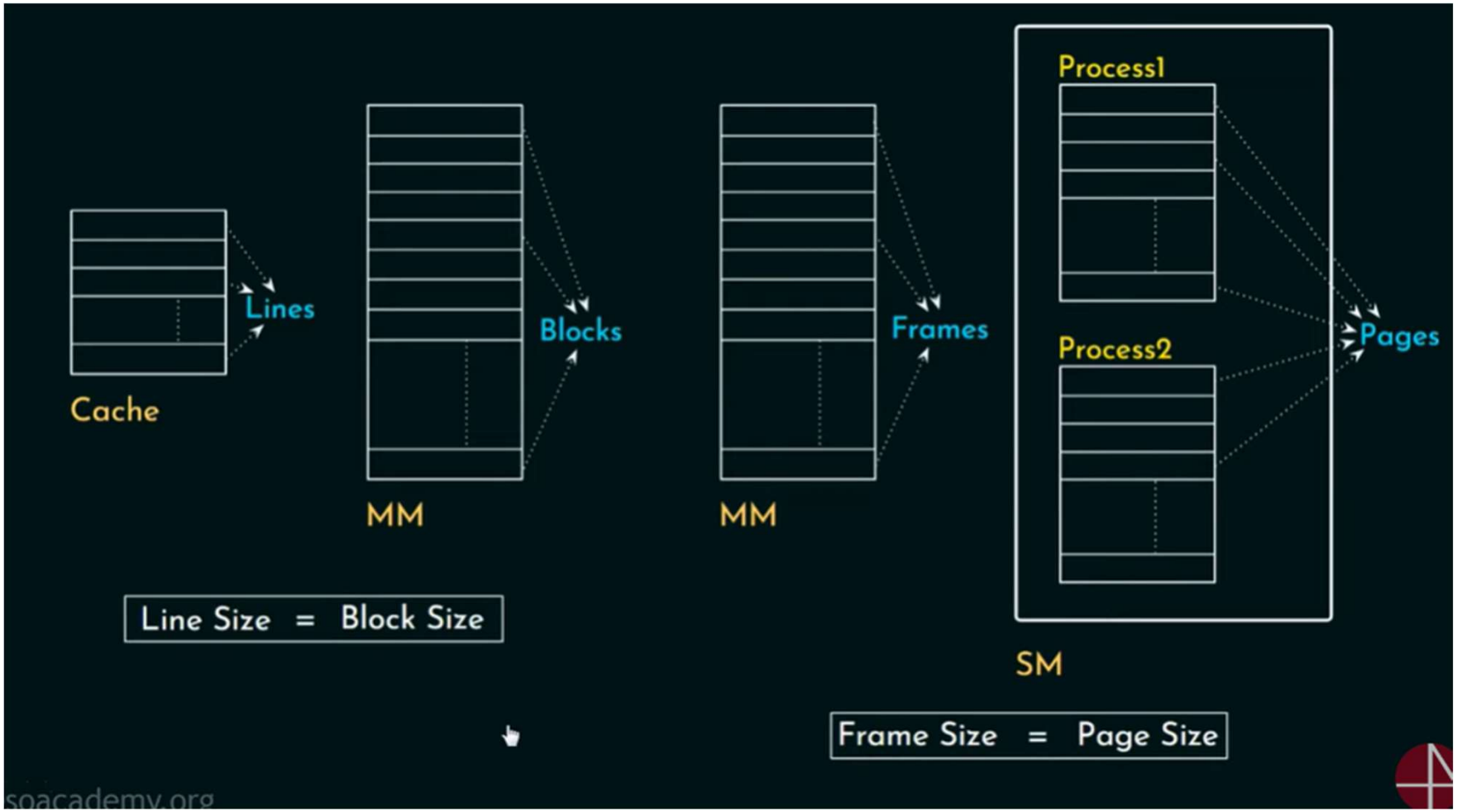
- **Mapping Function:**
  - Fewer cache lines than the memory blocks
  - An **algorithm**:
    - Needed for mapping cache lines to memory blocks
  - Also a **means** to determine:
    - which memory **block** currently **occupies** a **cache line**

Elements of cache design

- **Mapping Function:**
  - Direct
  - Associative
  - Set Associative

Direct Memory Mapping

- It is called Direct Memory mapping because:
  - All memory blocks are directly mapped onto the cache lines

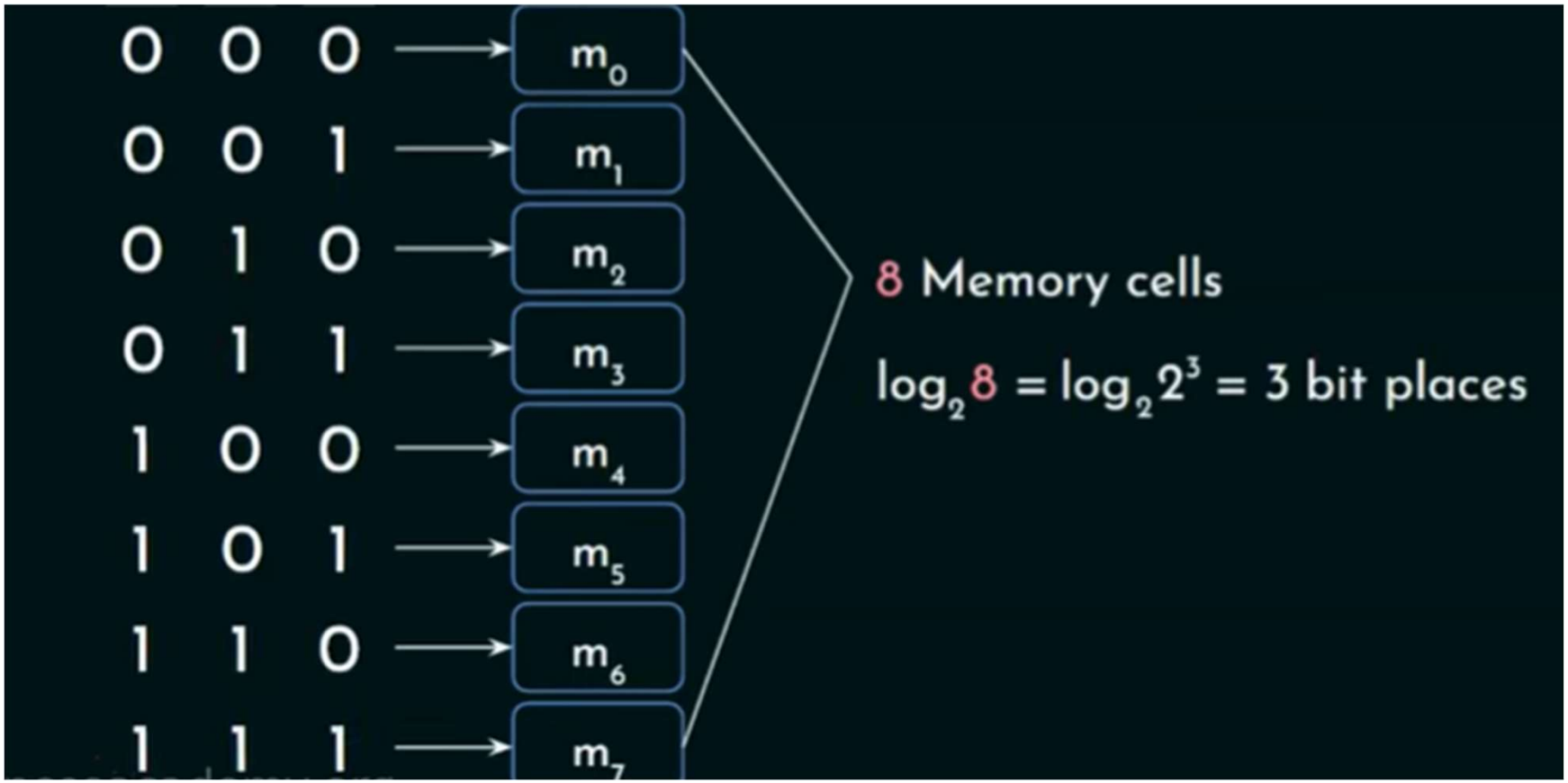

Lines
Cache
Blocks
MM
Frames
MM
Process1
Process2
Pages
SM
Line Size = Block Size
Frame Size = Page Size
soacademy.org

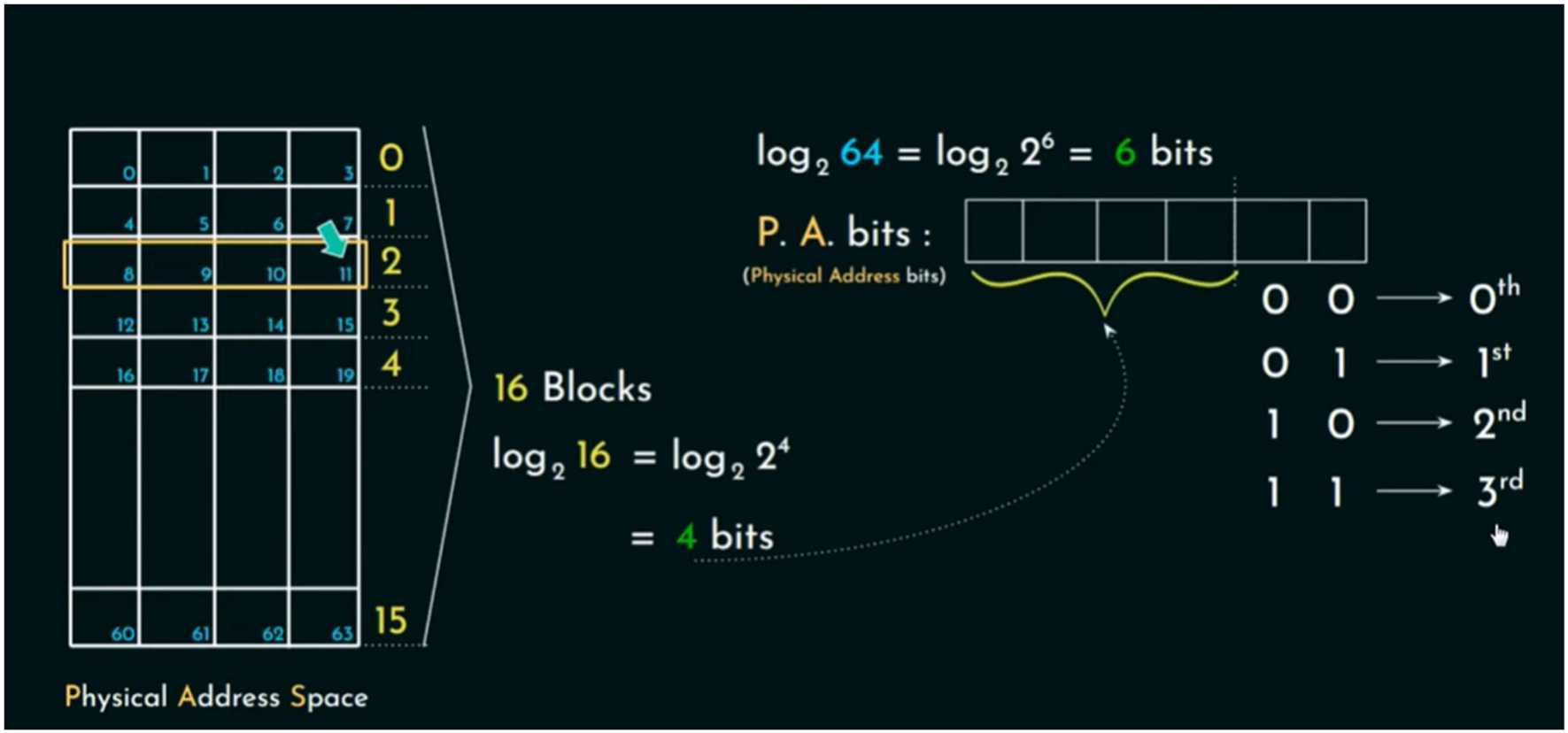
Main Memory Size : 64 words i.e. (0, 1, . . . ,63)

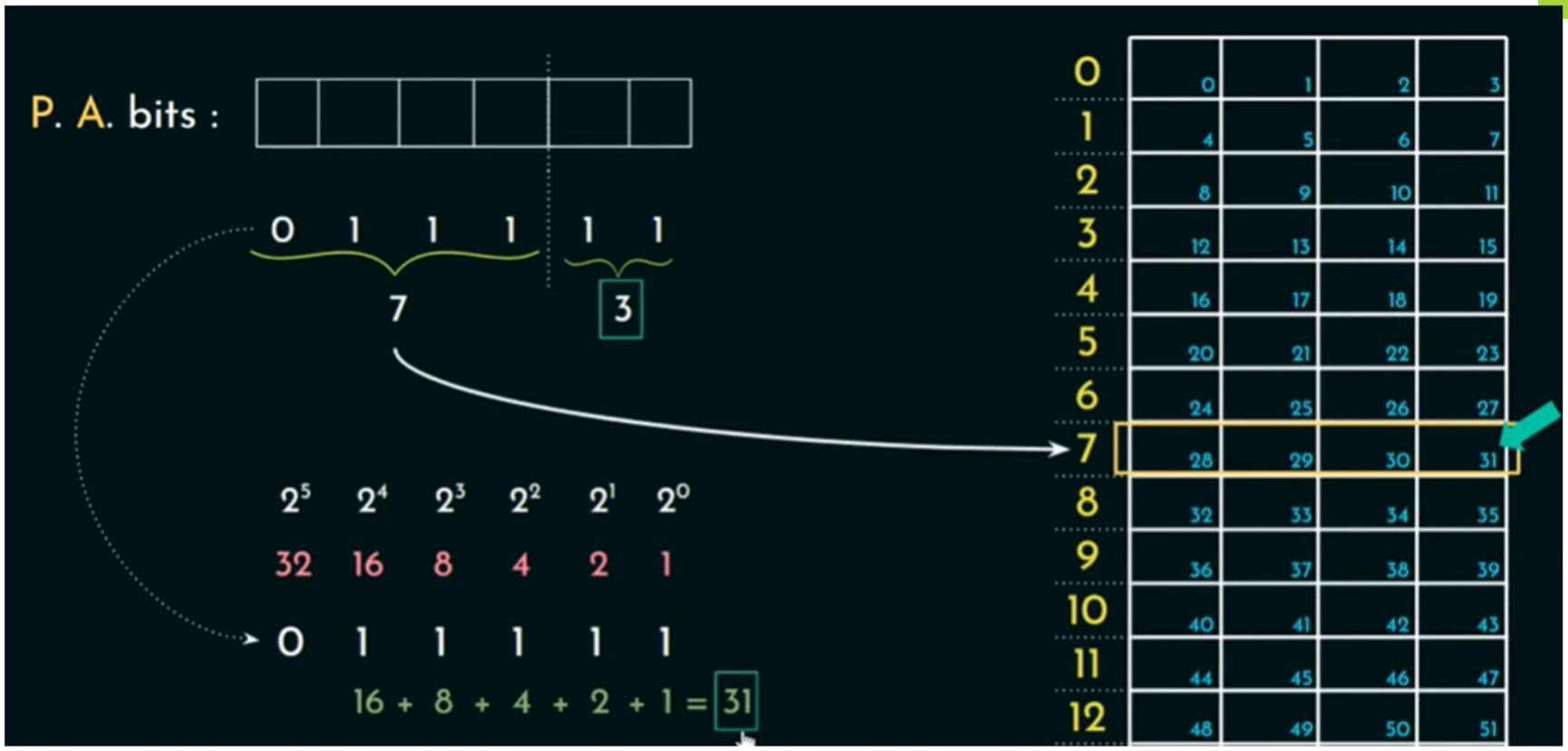
Block Size : 4 words

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

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 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1
m0
m1
m2
m3
m4
m5
m6
m7
8 Memory cells
log2 8 = log2 2^3 = 3 bit places


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

Cache Size : 16 words

Block Size : 4 words

Line Size : 4 words

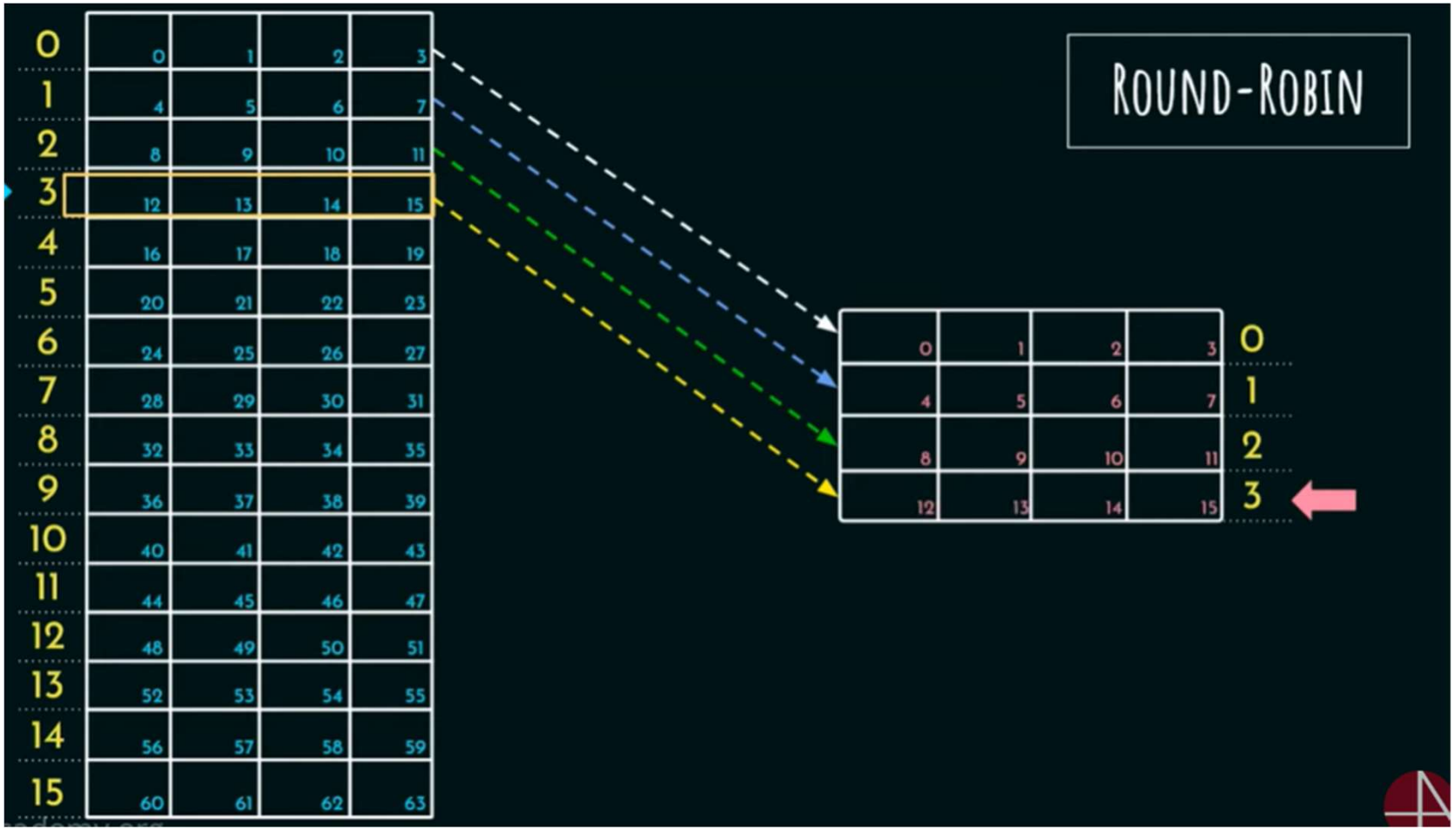
Block Size = Line Size

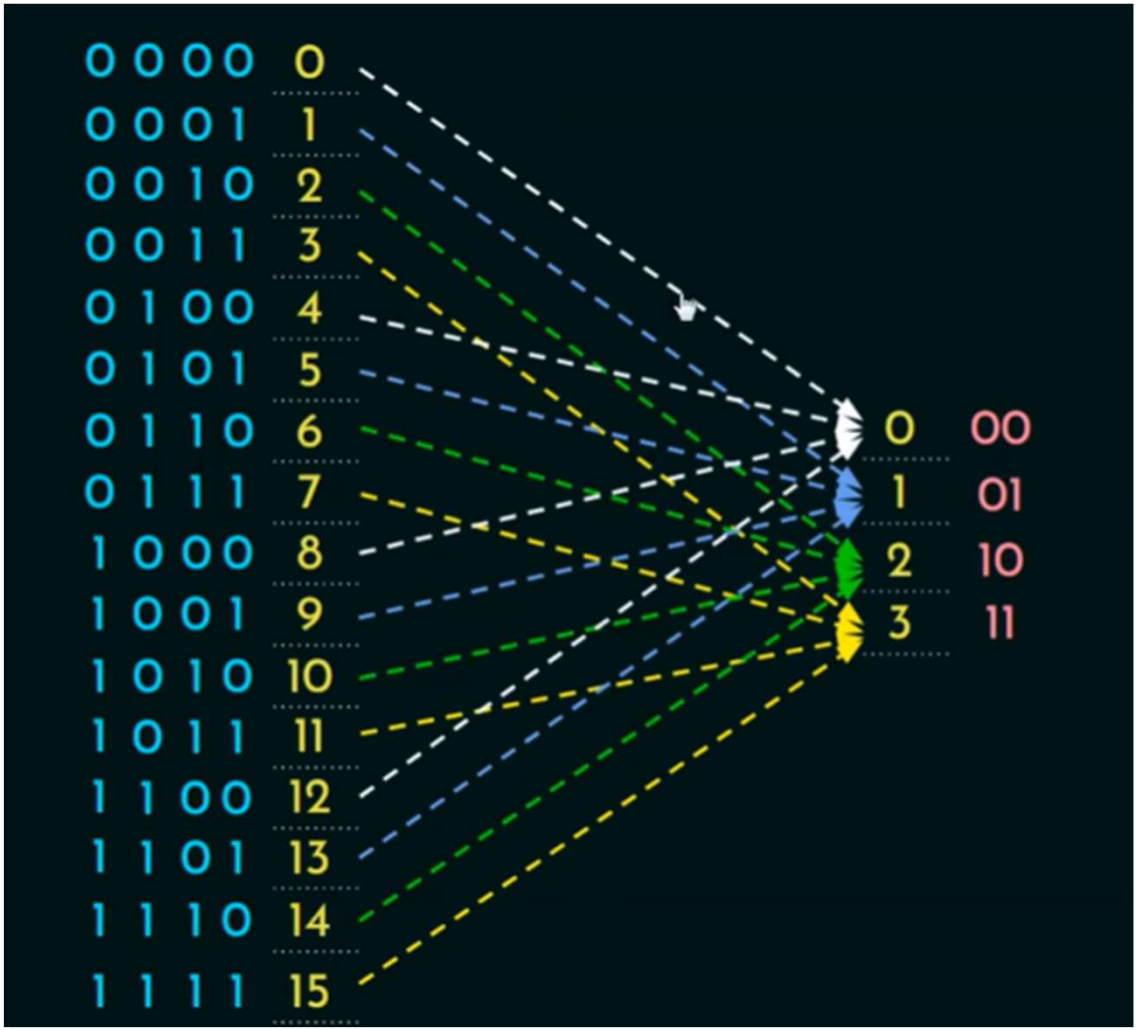
No. of Lines in Cache : 16 / 4 = 4 i.e. 0, 1, 2, 3

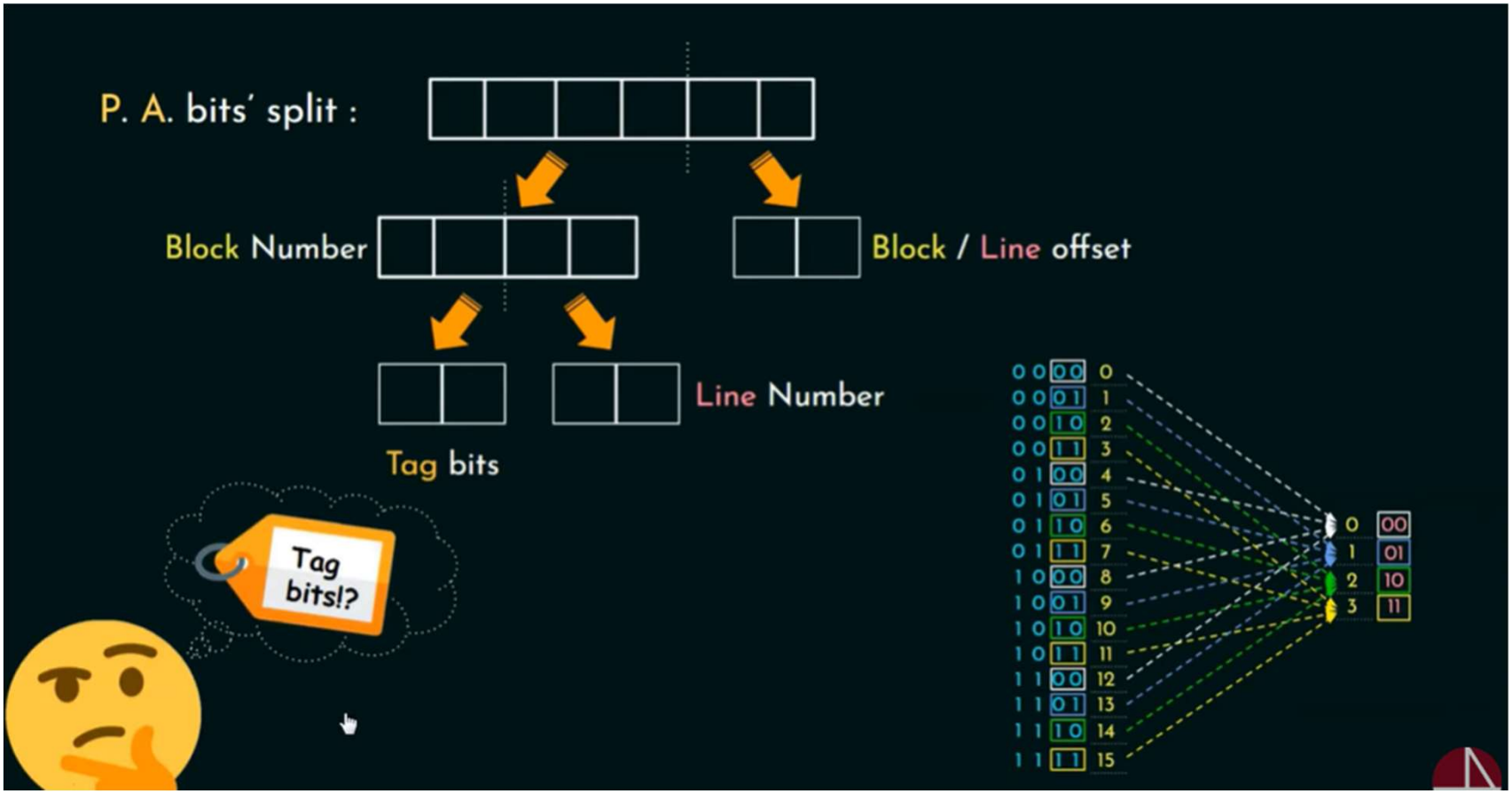
4 Lines

$\log_2 4 = \log_2 2^2$

= 2 bits


ROUND-ROBIN
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
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 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


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