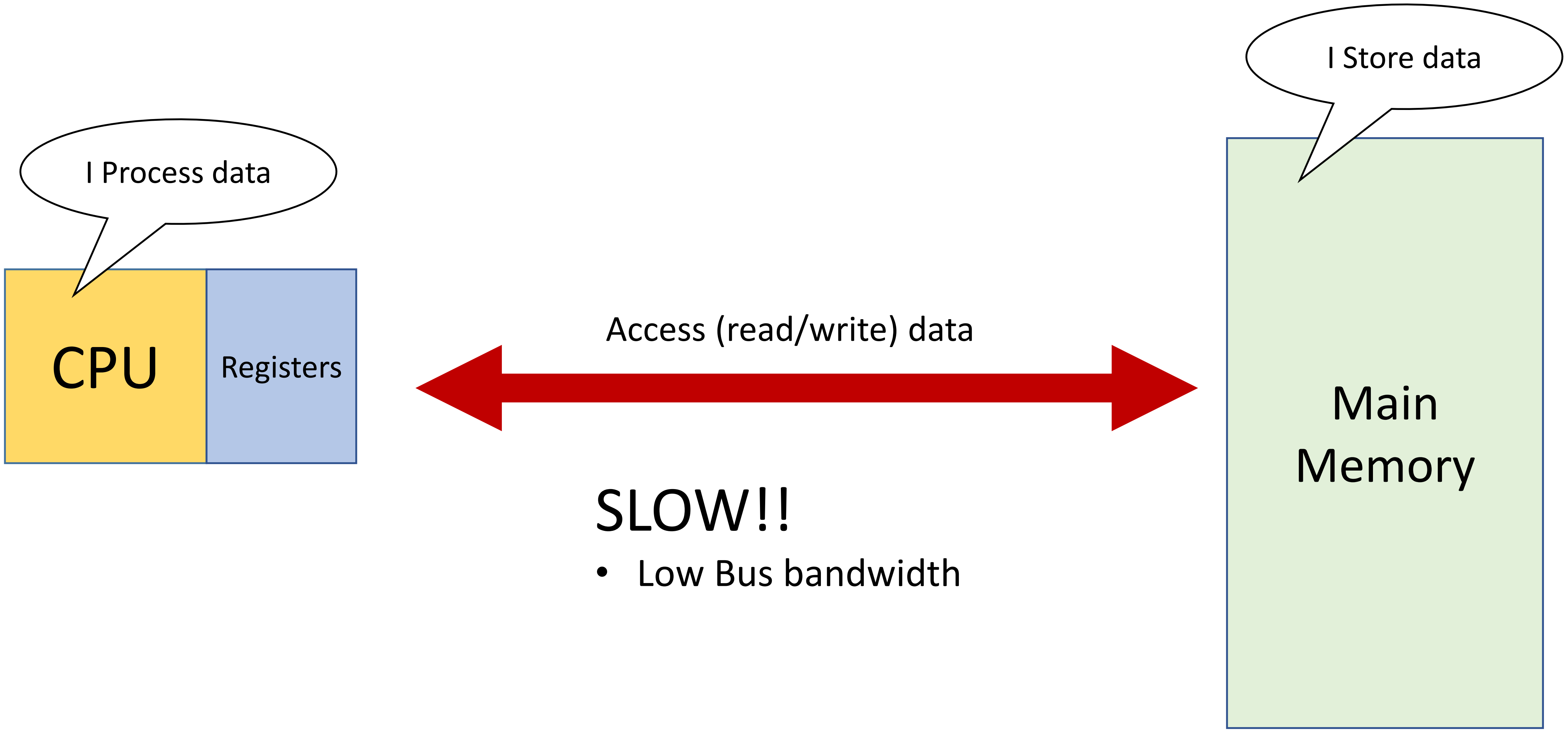
Cache Memory

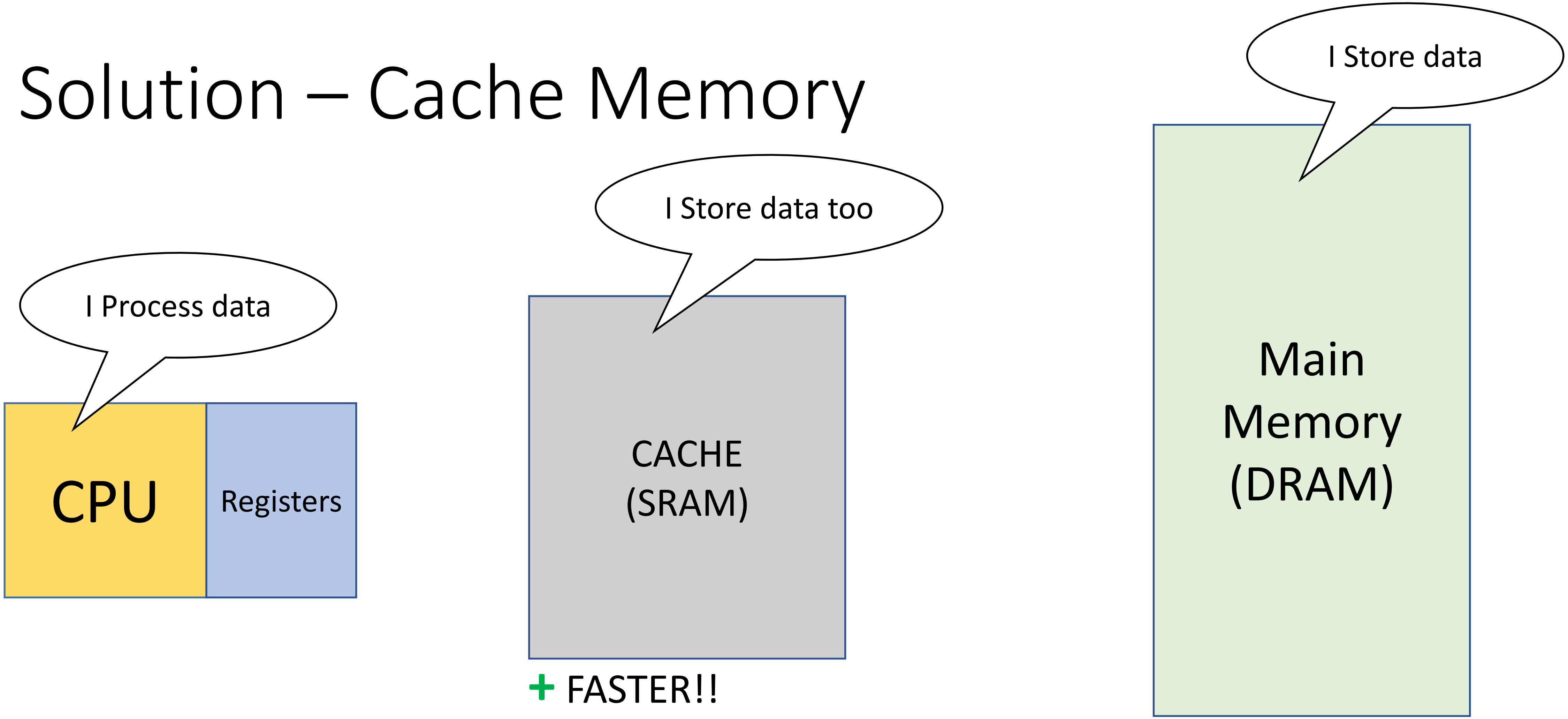
From what we know until now



SLOW!!

- Low Bus bandwidth

Solution – Cache Memory



I Process data

CPU

Registers

I Store data too

CACHE (SRAM)

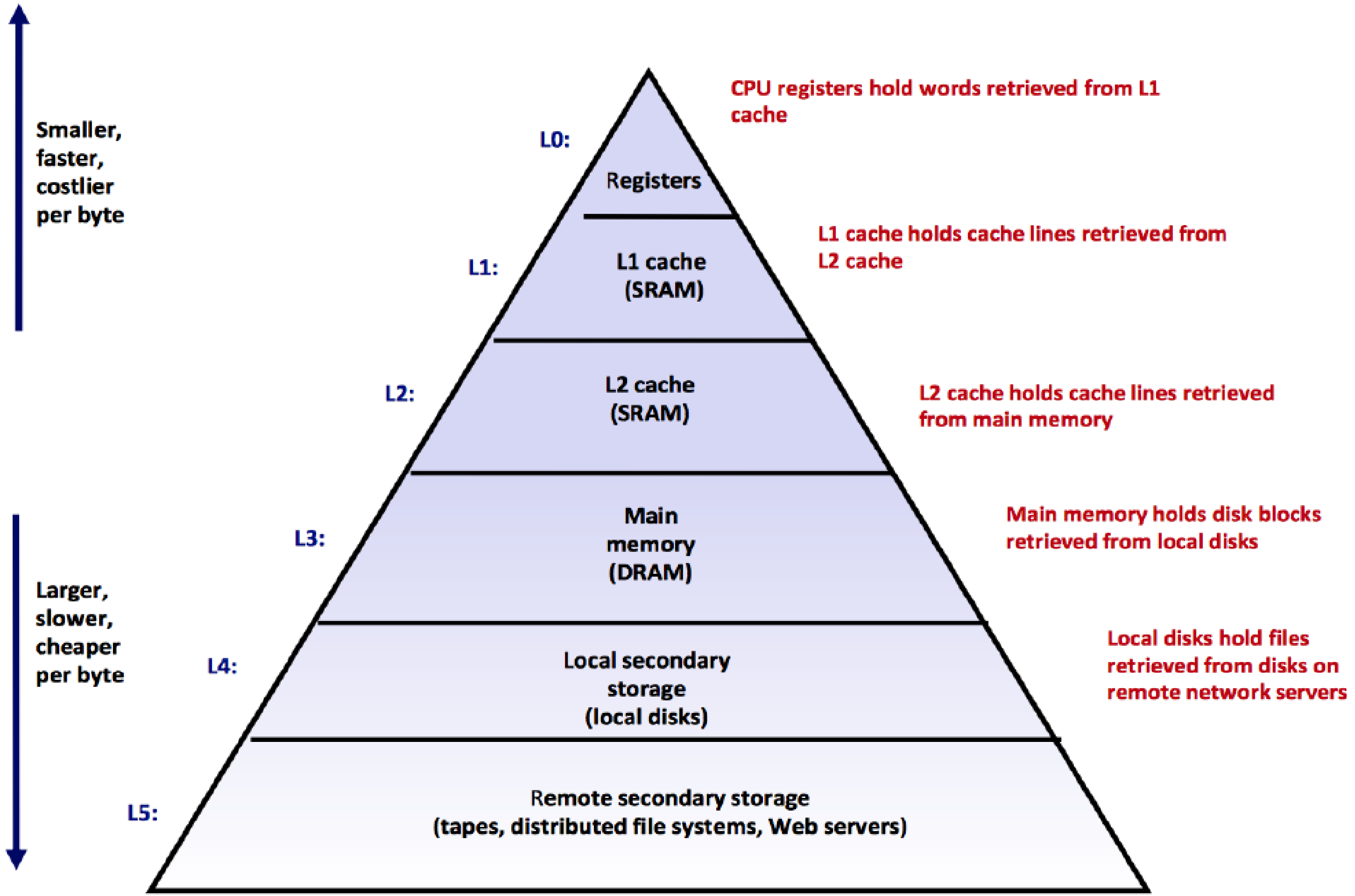
I Store data

Main Memory (DRAM)

- + FASTER!!
- + Closer to Processor
- – More Expensive
- – Smaller Capacity

- – SLOWER
- – Far from Processor
- + Cheaper
- + Larger Capacity

Memory Hierarchy



CPU registers hold words retrieved from L1 cache

L1 cache holds cache lines retrieved from L2 cache

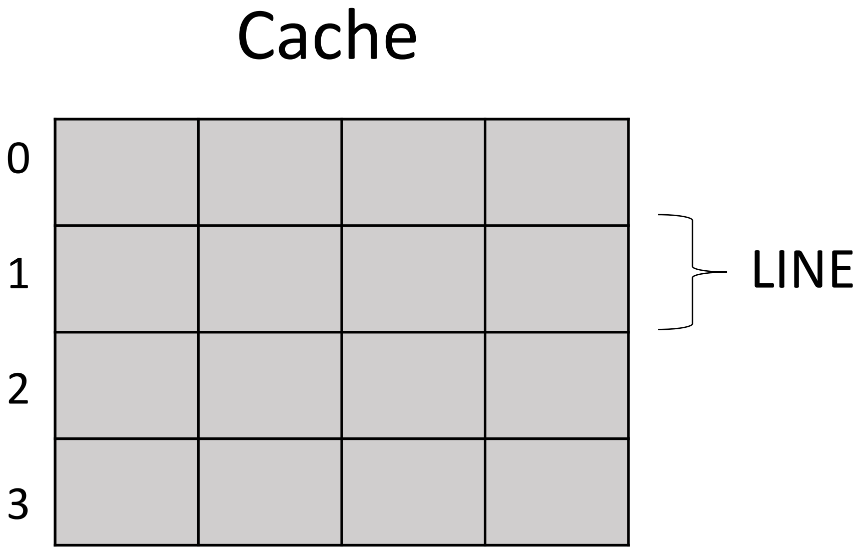
L2 cache holds cache lines retrieved from main memory

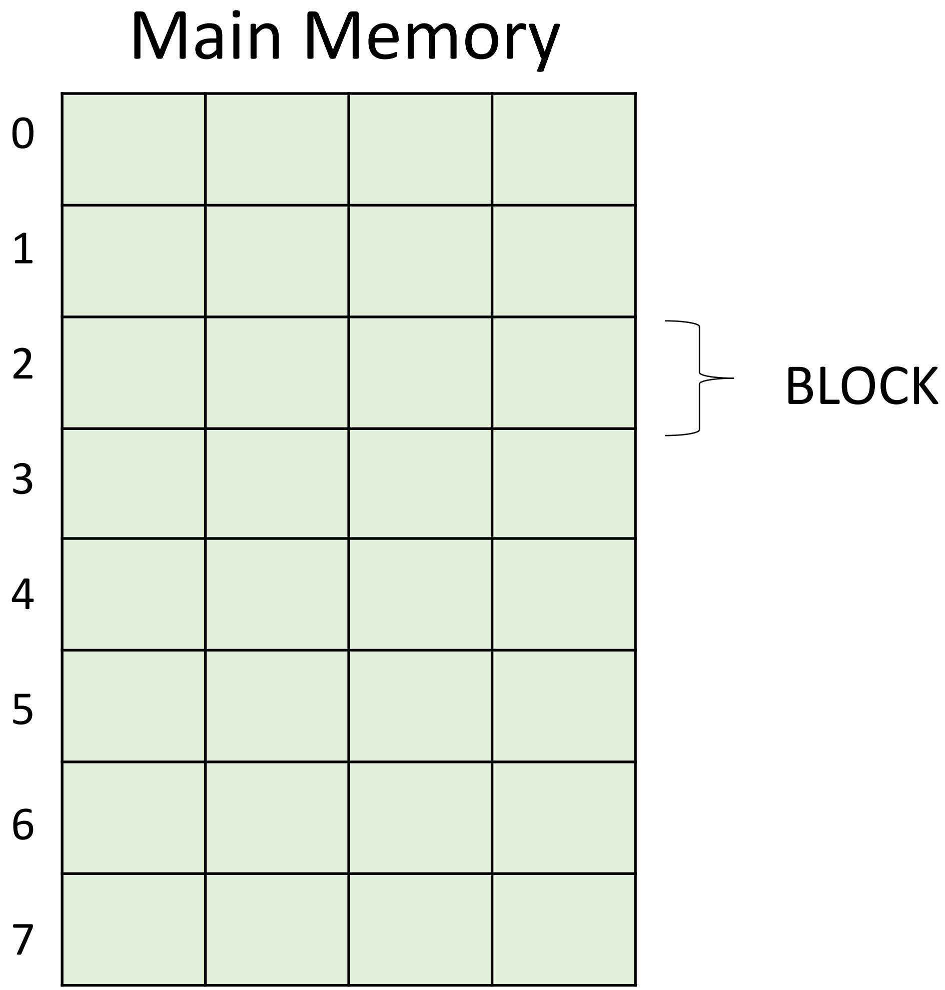
Main memory holds disk blocks retrieved from local disks

Local disks hold files retrieved from disks on remote network servers

https://slidetodoc.com/carnegie-mellon-the-memory-hierarchy-15-213-introduction/

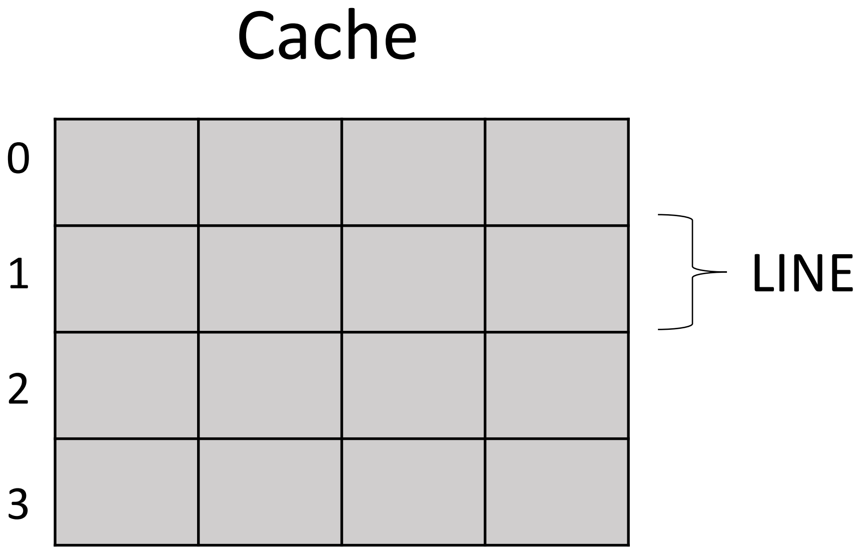
Looking Inside Cache and Main Memory

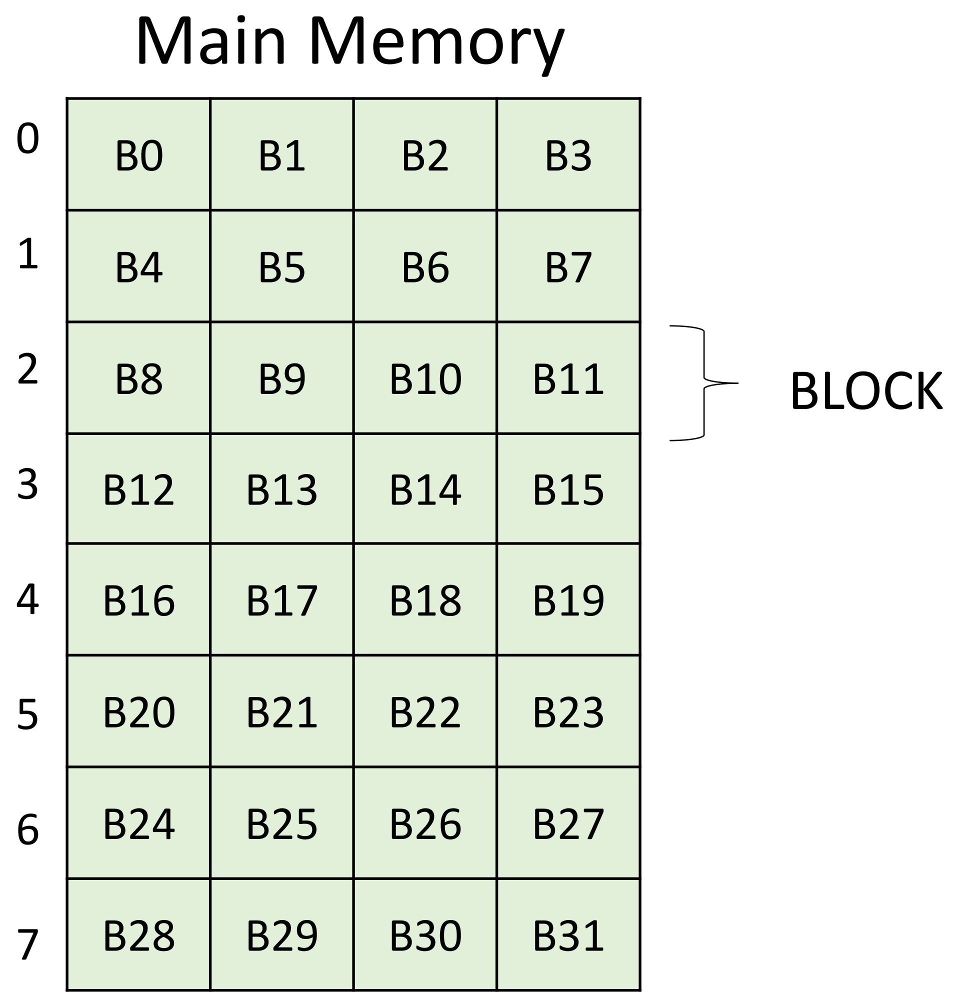




A collection of Cache Lines is called a Cache **SET**.

Looking Inside Cache and Main Memory

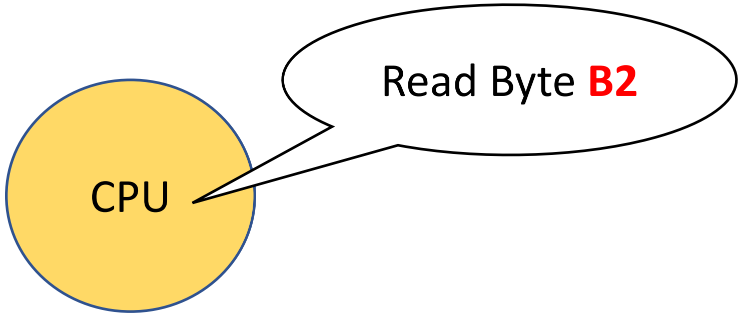




0	B0	B1	B2	B3
1	B4	B5	B6	B7
2	B8	B9	B10	B11
3	B12	B13	B14	B15
4	B16	B17	B18	B19
5	B20	B21	B22	B23
6	B24	B25	B26	B27
7	B28	B29	B30	B31

LINE SIZE = BLOCK SIZE

Accessing Data



CPU

Read Byte **B2**

Cache

0				
1				
2				
3				

Step 1: Search in CACHE

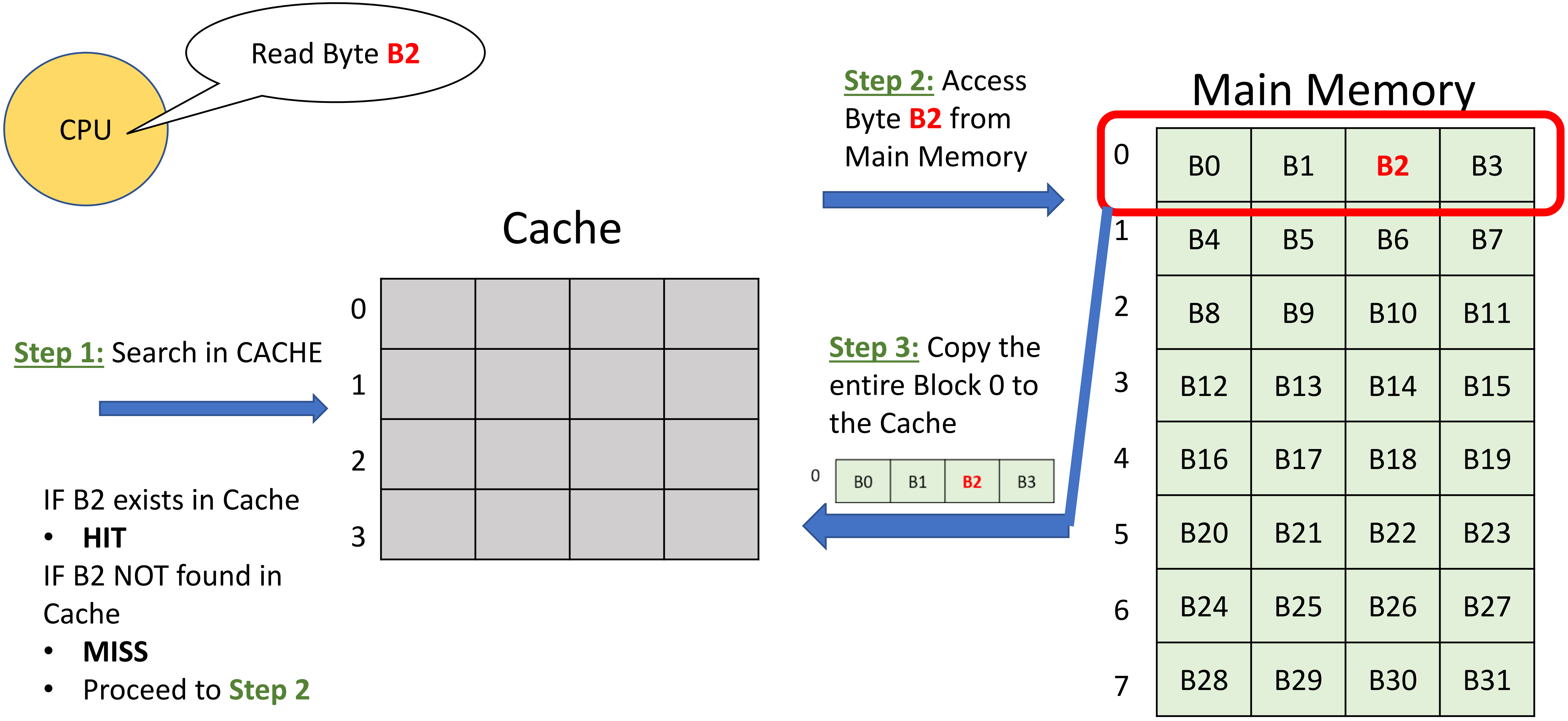
IF B2 exists in Cache
- **HIT**

IF B2 NOT found in Cache
- **MISS**
- Proceed to **Step 2**

Main Memory

0	B0	B1	B2	B3
1	B4	B5	B6	B7
2	B8	B9	B10	B11
3	B12	B13	B14	B15
4	B16	B17	B18	B19
5	B20	B21	B22	B23
6	B24	B25	B26	B27
7	B28	B29	B30	B31

Accessing Data



CPU

Read Byte **B2**

Step 1: Search in CACHE

IF B2 exists in Cache

- **HIT**

IF B2 NOT found in Cache

- **MISS**
- Proceed to **Step 2**

Cache

0				
1				
2				
3				

Step 2: Access Byte **B2** from Main Memory

Step 3: Copy the entire Block 0 to the Cache

0	B0	B1	**B2**	B3

Main Memory

0	B0	B1	**B2**	B3
1	B4	B5	B6	B7
2	B8	B9	B10	B11
3	B12	B13	B14	B15
4	B16	B17	B18	B19
5	B20	B21	B22	B23
6	B24	B25	B26	B27
7	B28	B29	B30	B31

Two Important Questions

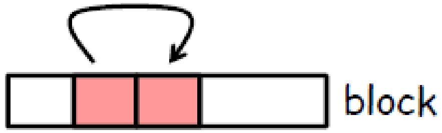
- Why did I copy the entire Block 0 to the cache?
- In which Cache Line will I place Block 0?

Two Important Questions

- Why did I copy the entire Block 0 to the cache?
- In which Cache Line will I place Block 0?

Locality

- “Tendency of a processor to access the same set of memory locations repetitively over a short period of time.”



- Spatial
  - If a Byte is accessed now then the Byte next to it is likely be accessed next
  - The number of Bytes in a block affects spatial locality

- Temporal
  - If a Byte is referenced now then the same Byte is likely be referenced again in the future

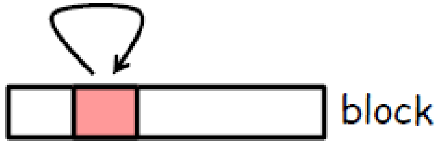


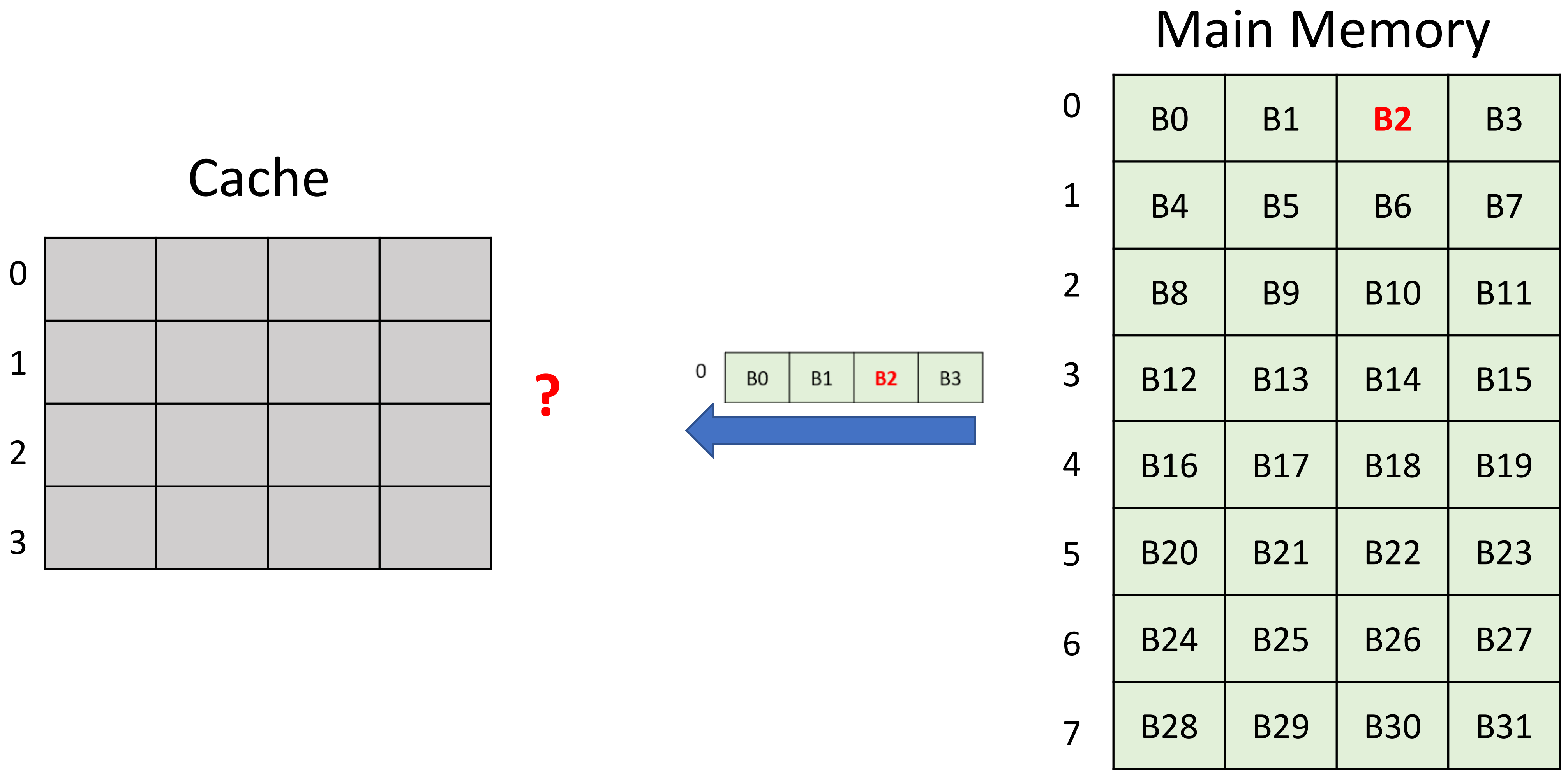
Image Source: Dr Petrucci's Slides - "Cache Memories"

Two Important Questions

- Why did I copy the entire Block 0 to the cache?
- In which Cache Line will I place Block 0?

Mapping

- ‘In which *line* of the cache is one *block* of the main memory mapped to?’



Main Memory

0	B0	B1	B2	B3
1	B4	B5	B6	B7
2	B8	B9	B10	B11
3	B12	B13	B14	B15
4	B16	B17	B18	B19
5	B20	B21	B22	B23
6	B24	B25	B26	B27
7	B28	B29	B30	B31

Types of Mapping

- Direct Mapping
- Fully Associative Mapping
- Set Associative

Consider a Scenario

- BYTE addressable
  - Address associated with each Byte

- Main Memory
  - Total number of Bytes = 64
  - Divided into Blocks
    - Block Size = 4 Bytes
  - How many blocks?
    - 64/4 = 16 Blocks

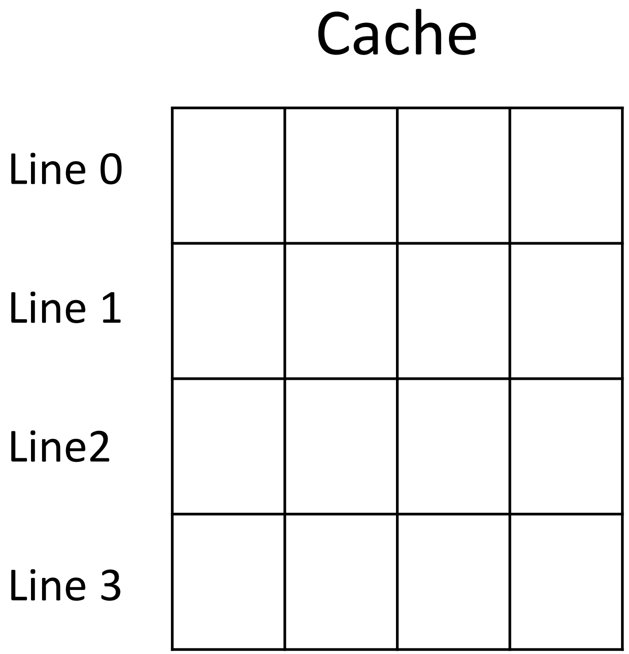
Main Memory

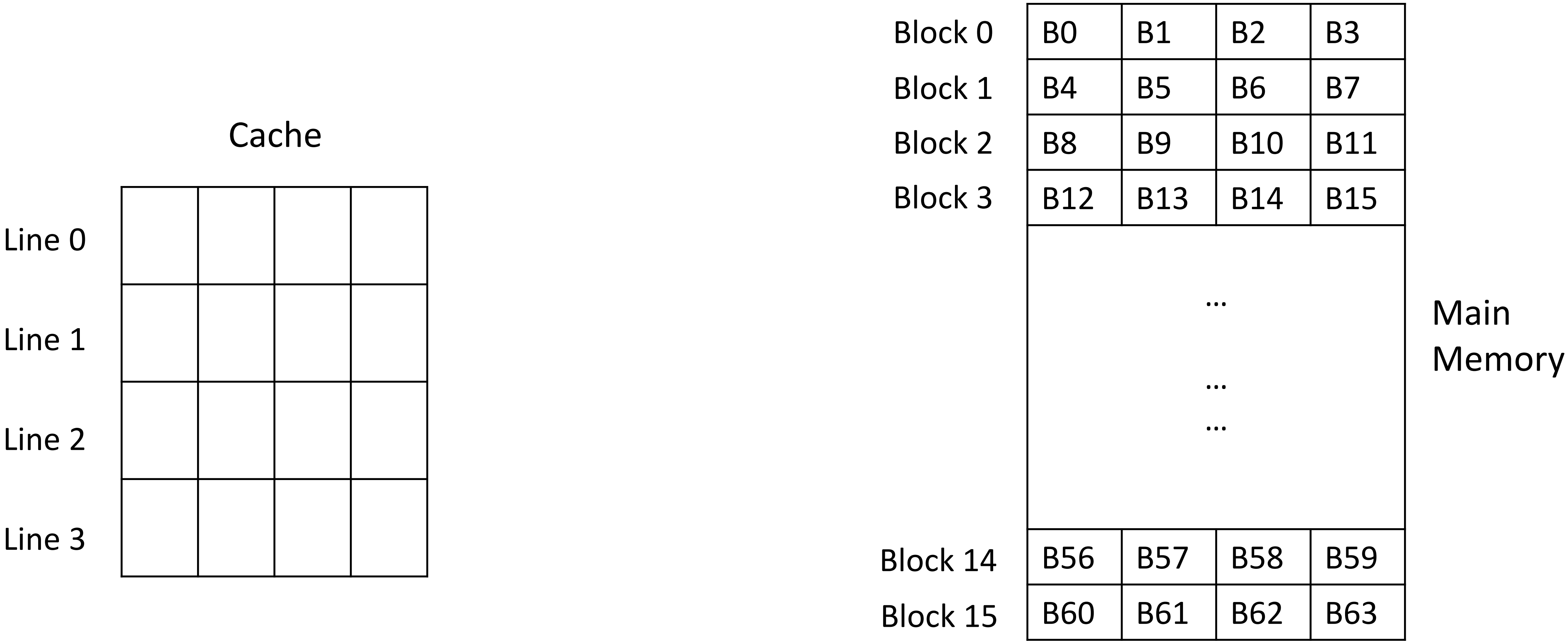
Block 0	B0	B1	B2	B3
Block 1	B4	B5	B6	B7
Block 2	B8	B9	B10	B11
Block 3	B12	B13	B14	B15
	… … …			
Block 14	B56	B57	B58	B59
Block 15	B60	B61	B62	B63

Consider a Scenario

- Byte addressable
  - Address associated with each Byte

- Cache
  - Total number of Bytes = 16
  - Divided into Sets
  - 1 Set = 1 Line
    - Line Size = 4 Bytes
  - How many Lines?
    - 16/4 = 4 Lines





Block 0	B0	B1	B2	B3
Block 1	B4	B5	B6	B7
Block 2	B8	B9	B10	B11
Block 3	B12	B13	B14	B15
	…			
	…			
	…			
Block 14	B56	B57	B58	B59
Block 15	B60	B61	B62	B63

Byte Addressable = Each Byte in Main Memory is identified by an address

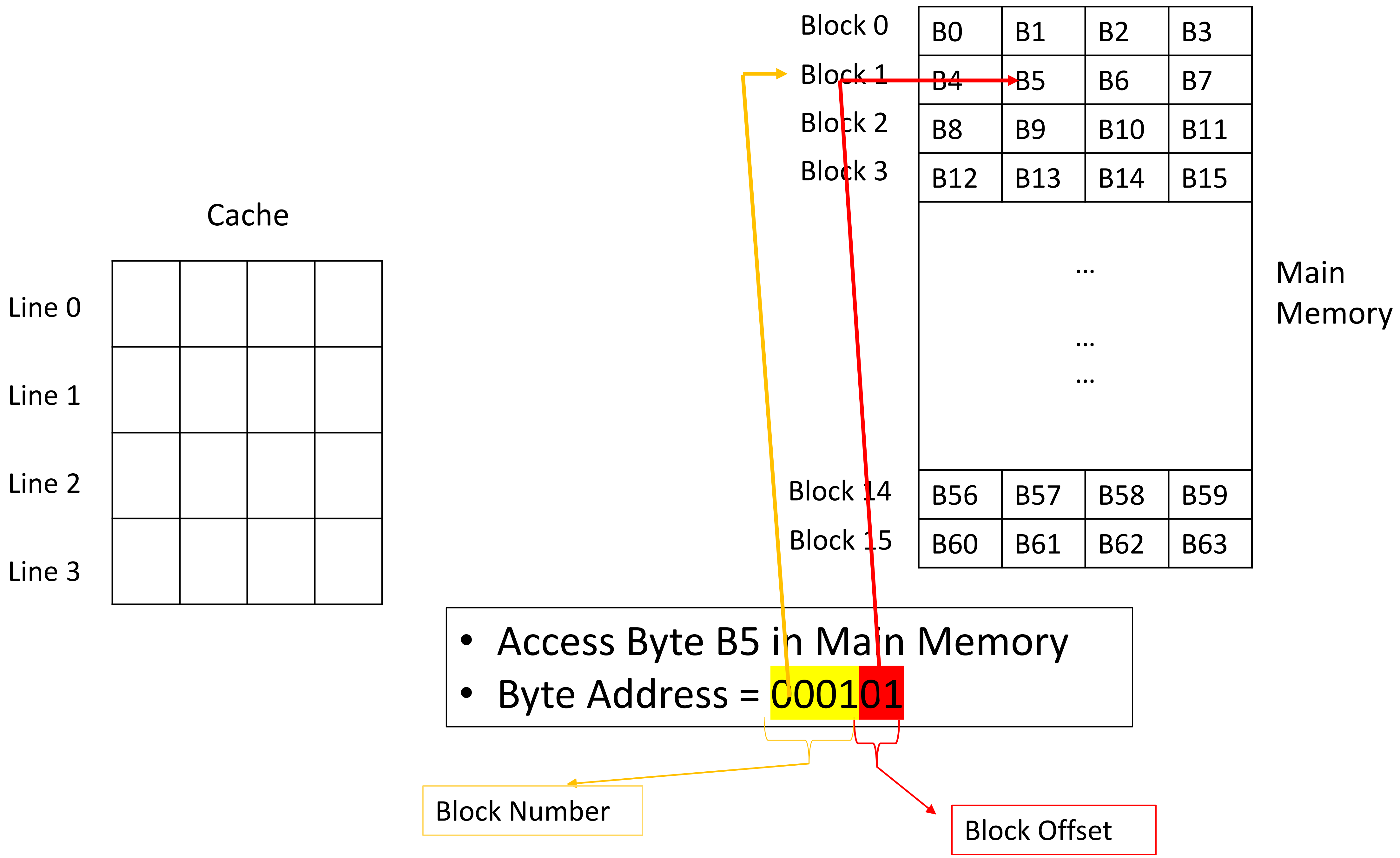
Cache

Line 0				
Line 1				
Line 2				
Line 3				

Block 0	B0	B1	B2	B3
Block 1	B4	B5	B6	B7
Block 2	B8	B9	B10	B11
Block 3	B12	B13	B14	B15
	…			
	…			
	…			
Block 14	B56	B57	B58	B59
Block 15	B60	B61	B62	B63

Main Memory

- Number of Bytes in Main Memory = 64 = 2^6
- How many addresses?
  - 64
- How many bits per address?
  - 6-bit address



Cache

Line 0

Line 1

Line 2

Line 3

Main Memory

Block 0	B0	B1	B2	B3
Block 1	B4	B5	B6	B7
Block 2	B8	B9	B10	B11
Block 3	B12	B13	B14	B15
	…			
	…			
	…			
Block 14	B56	B57	B58	B59
Block 15	B60	B61	B62	B63

- Access Byte B5 in Main Memory
- Byte Address = 000101

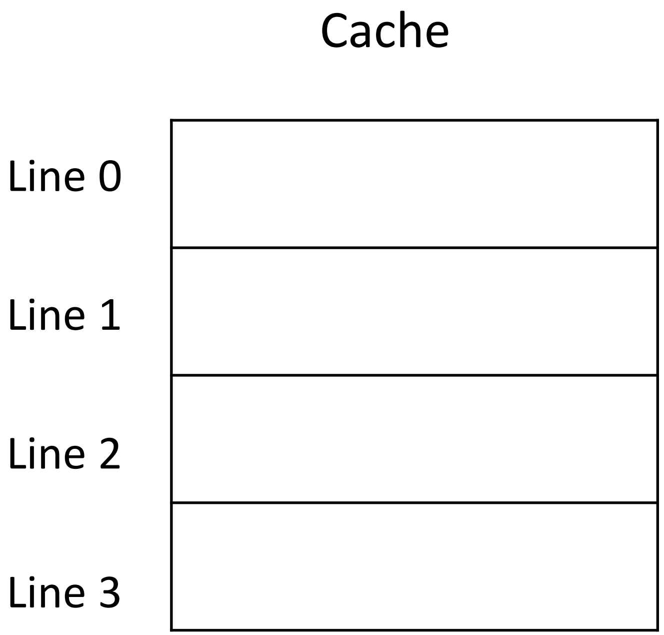
Block Number

Block Offset

Direct Mapping

Direct Mapping

1 Set = 1 Line, so Line Number and Set Number are the same



Block 0	B0	B1	B2	B3
Block 1	B4	B5	B6	B7
Block 2	B8	B9	B10	B11
Block 3	B12	B13	B14	B15
	… … …			
Block 14	B56	B57	B58	B59
Block 15	B60	B61	B62	B63

Main Memory

- Mapped in Round Robin manner
- Line Number/Index = K mod n
  - **K = Block Number**
  - **n = Number of lines**

Direct Mapping

1 Set = 1 Line , so Line Number and Set Number are the same

Cache

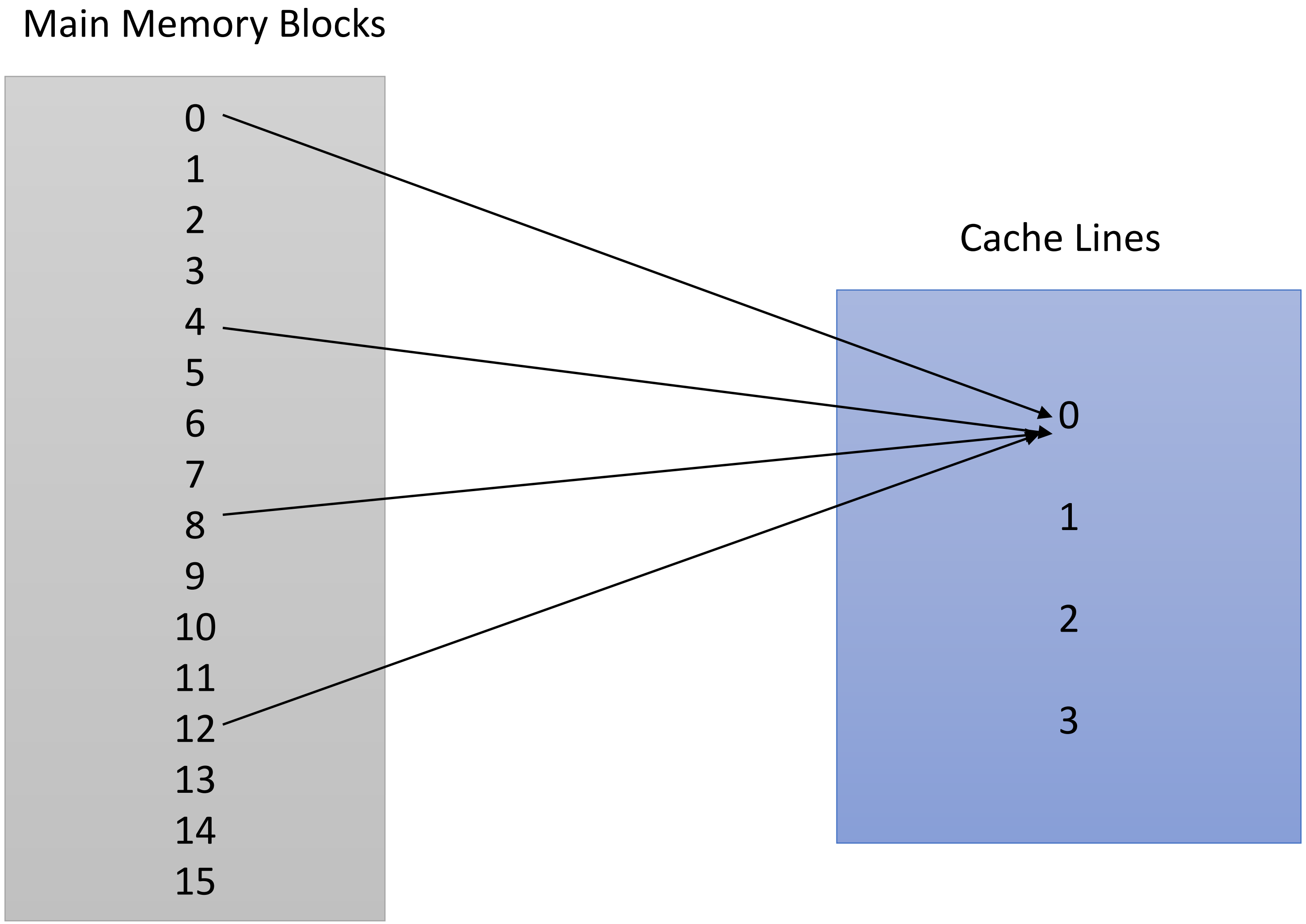
Line 0	Block 0/4/8/12
Line 1	Block 1/5/9/13
Line 2	Block 2/6/10/14
Line 3	Block 3/7/11/15

Main Memory

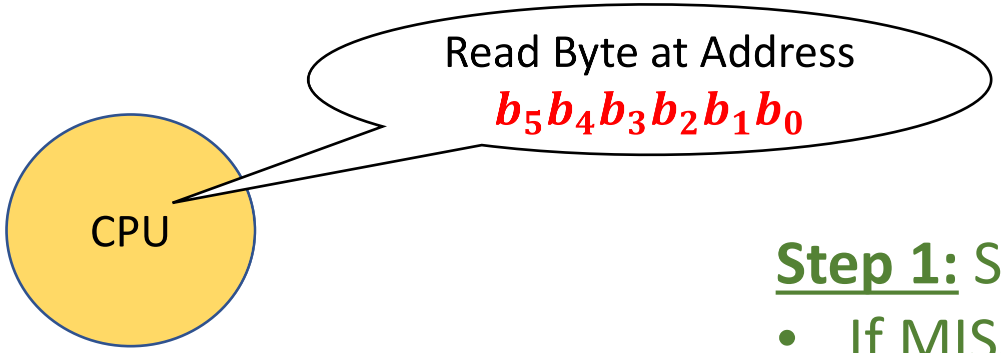
Block 0	B0	B1	B2	B3
Block 1	B4	B5	B6	B7
Block 2	B8	B9	B10	B11
Block 3	B12	B13	B14	B15
	… … …			
Block 14	B56	B57	B58	B59
Block 15	B60	B61	B62	B63

- Mapped in Round Robin manner
- Line Number/Index = K mod n
  - **K = Block Number**
  - **n = Number of lines**

Direct Mapping – Many to One Mapping



Direct Mapping



Read Byte at Address
$b_5b_4b_3b_2b_1b_0$

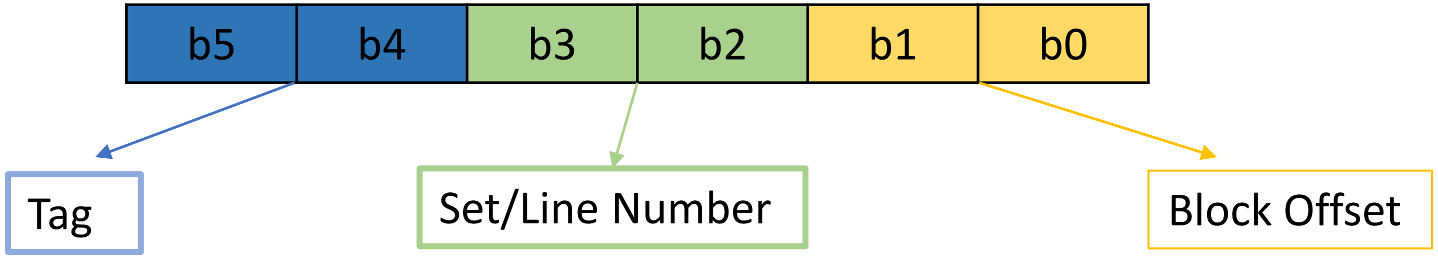
Step 1: Search in Cache

- If MISS, go to **Step 2**

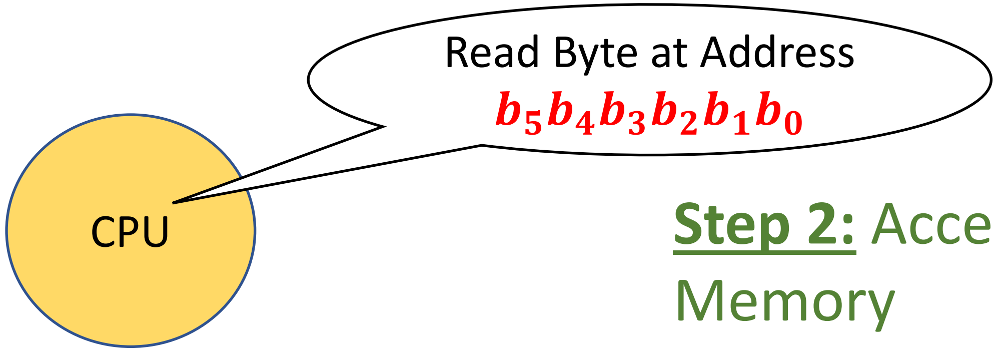
Cache

Line 0	Block **0/4/8/12**
Line 1	Block **1/5/9/13**
Line 2	Block **2/6/10/14**
Line 3	Block **3/7/11/15**

Byte address

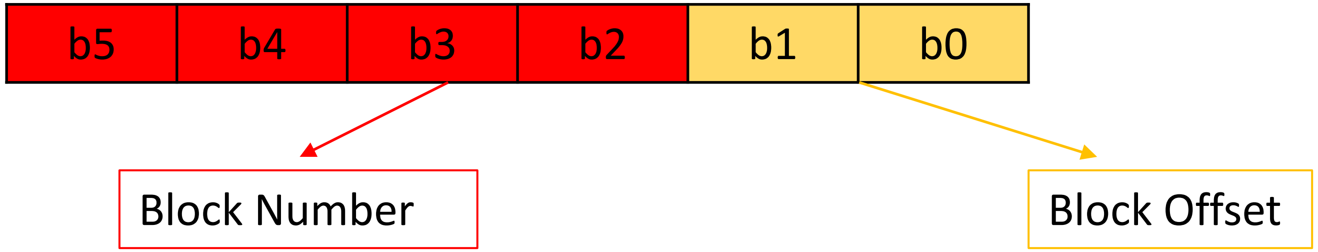


Direct Mapping



Step 2: Access From Main Memory

Byte address

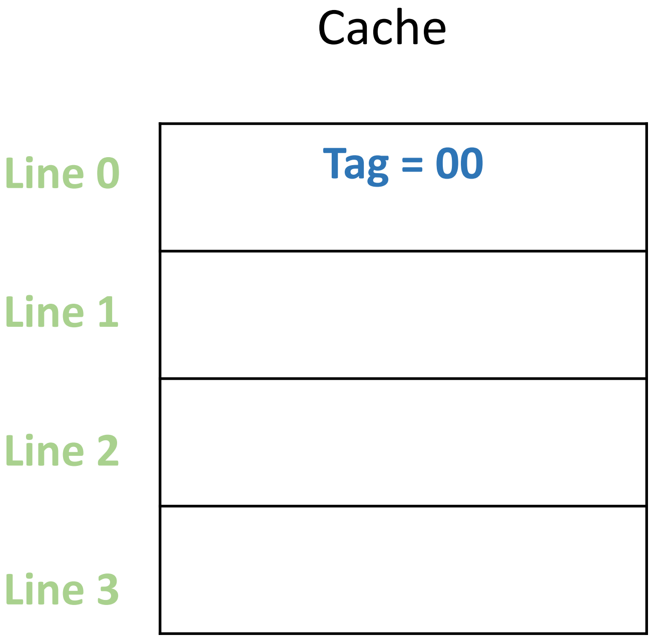


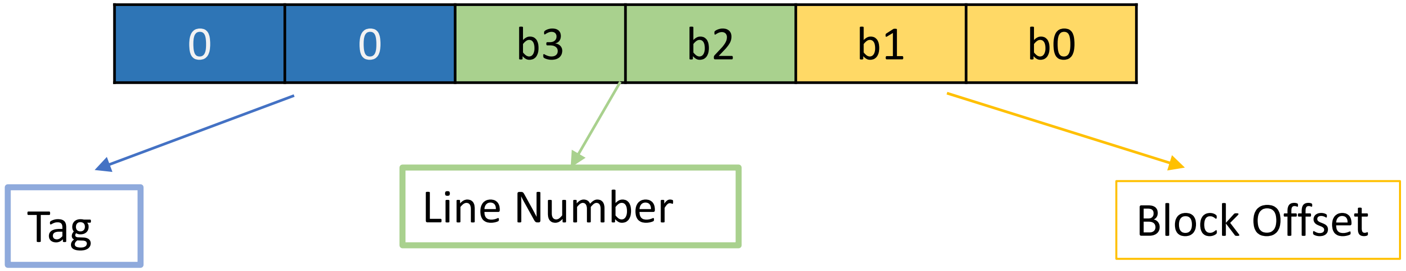
Block 0	B0	B1	B2	B3
Block 1	B4	B5	B6	B7
Block 2	B8	B9	B10	B11
Block 3	B12	B13	B14	B15
	… … …			
Block 14	B56	B57	B58	B59
Block 15	B60	B61	B62	B63

Main Memory

Direct Mapping

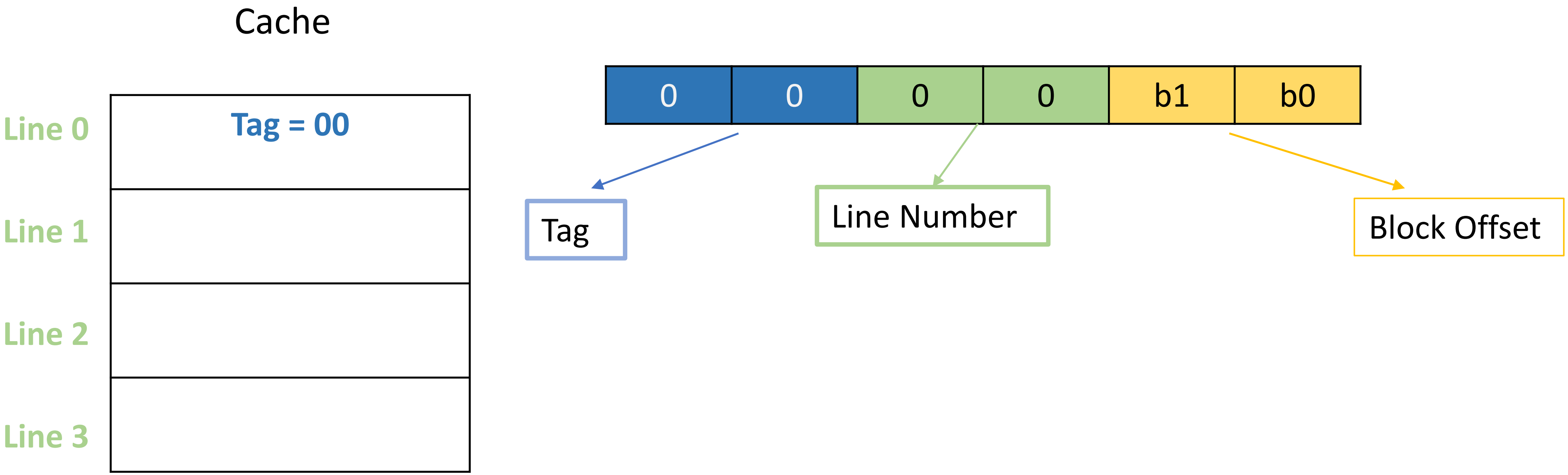
- Suppose Line 0 of the cache has Tag ’00’
  - What are the addresses of the Bytes present in line 0 of the cache?





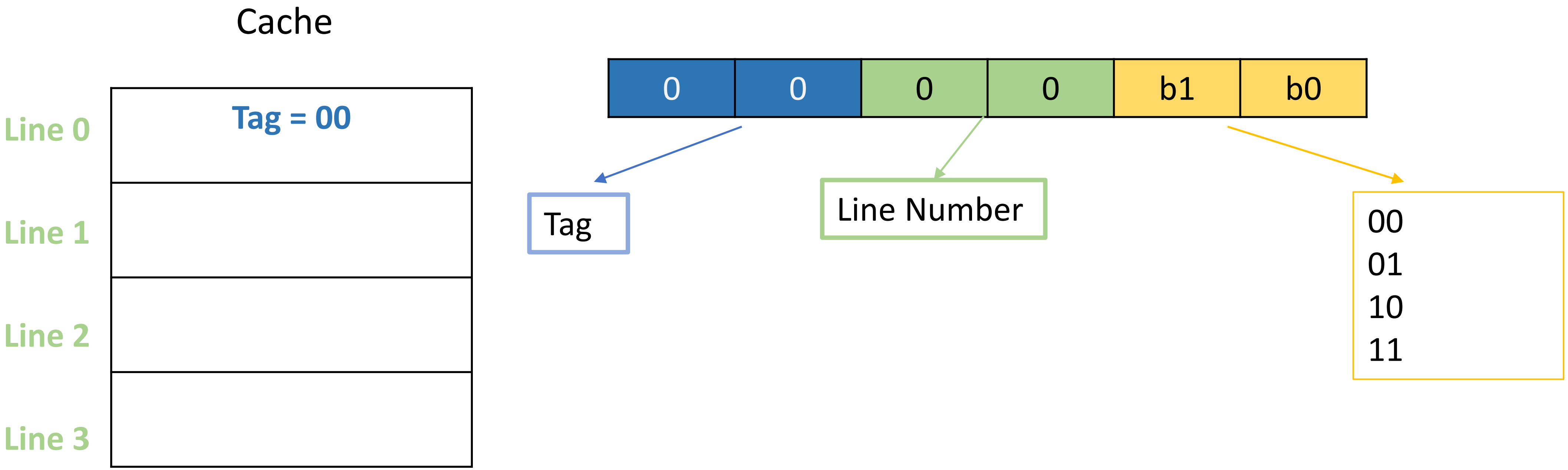
Direct Mapping

- Suppose Line 0 of the cache has Tag ’00’
  - What are the addresses of the Bytes present in line 0 of the cache?



Direct Mapping

- Suppose Line 0 of the cache has Tag ’00’
  - What are the addresses of the Bytes present in line 0 of the cache?



Direct Mapping

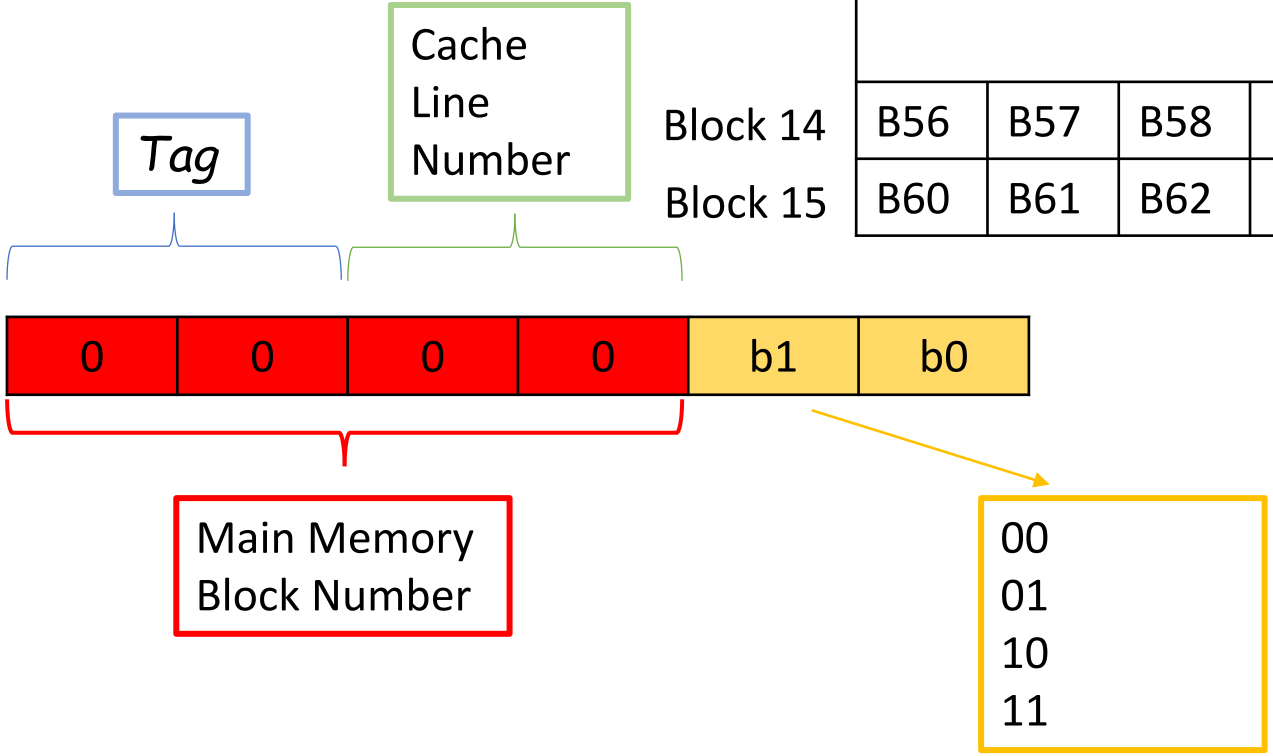
- Suppose Line 0 of the cache has Tag ’00’
  - What are the addresses of the Bytes present in line 0 of the cache?

	B0	B1	B2	B3
Block 0	B0	B1	B2	B3
Block 1	B4	B5	B6	B7
Block 2	B8	B9	B10	B11
Block 3	B12	B13	B14	B15
	…			
	…			
	…			
Block 14	B56	B57	B58	B59
Block 15	B60	B61	B62	B63

Main Memory

Cache

Tag 00 Line 0	B0, B1, B2, B3
Line 1	
Line 2	
Line 3	



Tag

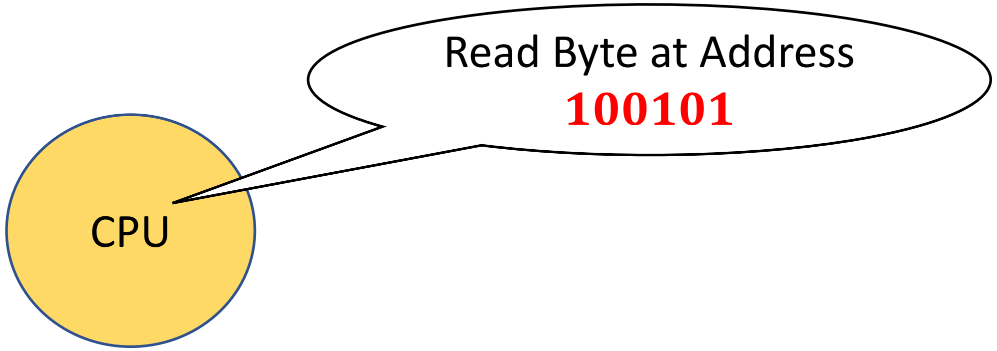
Cache Line Number

0	0	0	0	b1	b0

Main Memory Block Number

00
01
10
11

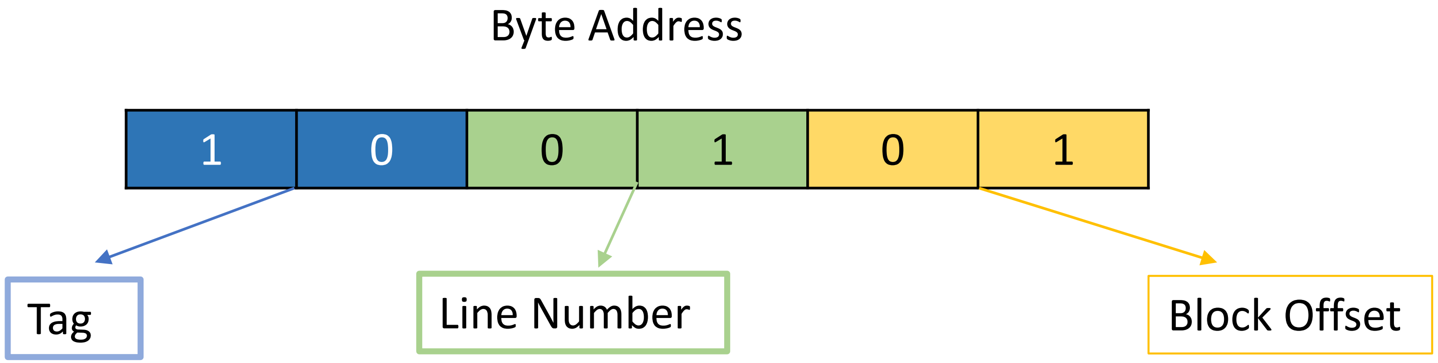
Sequence of Actions



Read Byte at Address
100101

Cache

	Cache	Valid Bit
Line 0	Tag = 01	1
Line 1	Tag = 11	1
Line 2		0
Line 3	Tag = 10	1

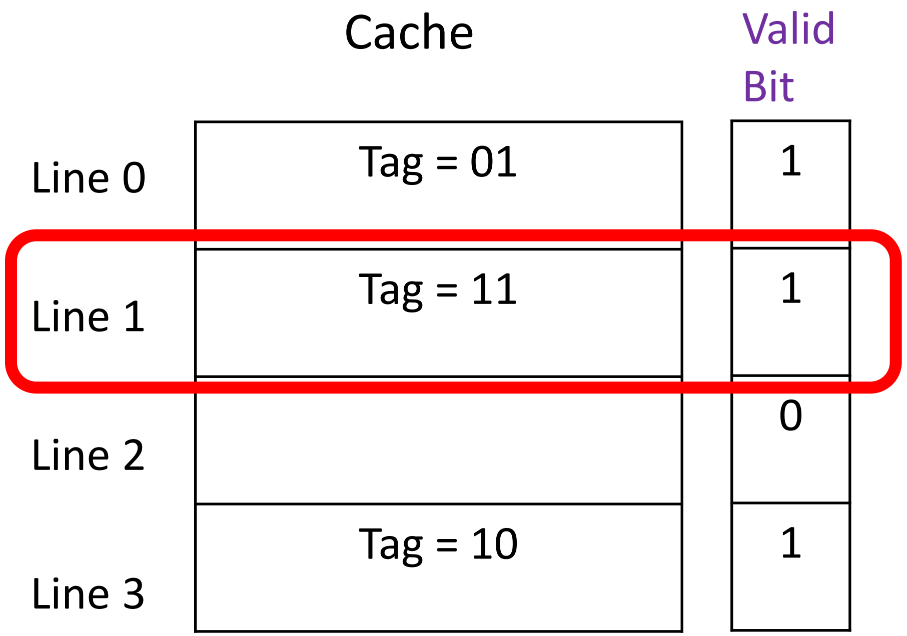


Step 1: Go to **Line Number** 01

Step 2: Check **Valid Bit.** If Valid Bit = 0, then it’s a **MISS**. If Valid Bit = 1, go to Step 3

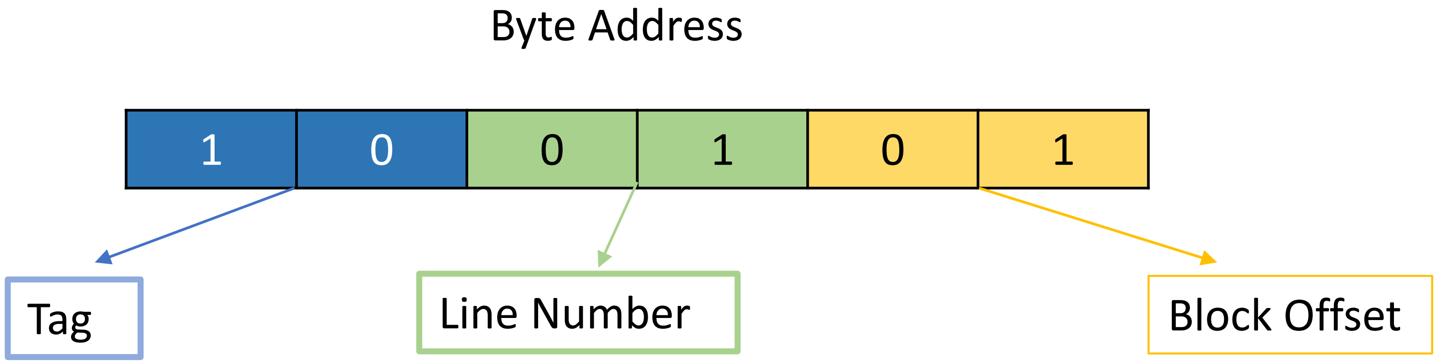
Step 3: If Tag matches, then it’s **HIT**, or else **MISS**

Sequence of Actions



Cache

	Cache	Valid Bit
Line 0	Tag = 01	1
Line 1	Tag = 11	1
Line 2		0
Line 3	Tag = 10	1



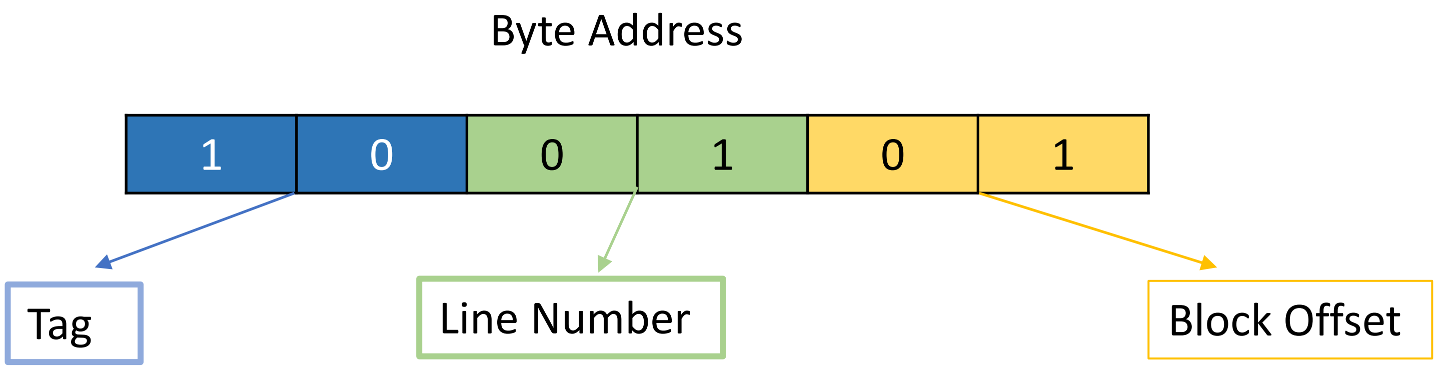
Byte Address

1	0	0	1	0	1

This is a MISS!!

Sequence of Actions

	Cache	Valid Bit
Line 0	Tag = 01	1
Line 1	Tag = 10	1
Line 2		0
Line 3	Tag = 10	1



This is a HIT!!

Disadvantages of Direct Mapping

Cache

Line 0	
Line 1	5
Line 2	
Line 3	

Sequence of Block Numbers requested by CPU

5, 4, 8 , 12, 9, 13

Disadvantages of Direct Mapping

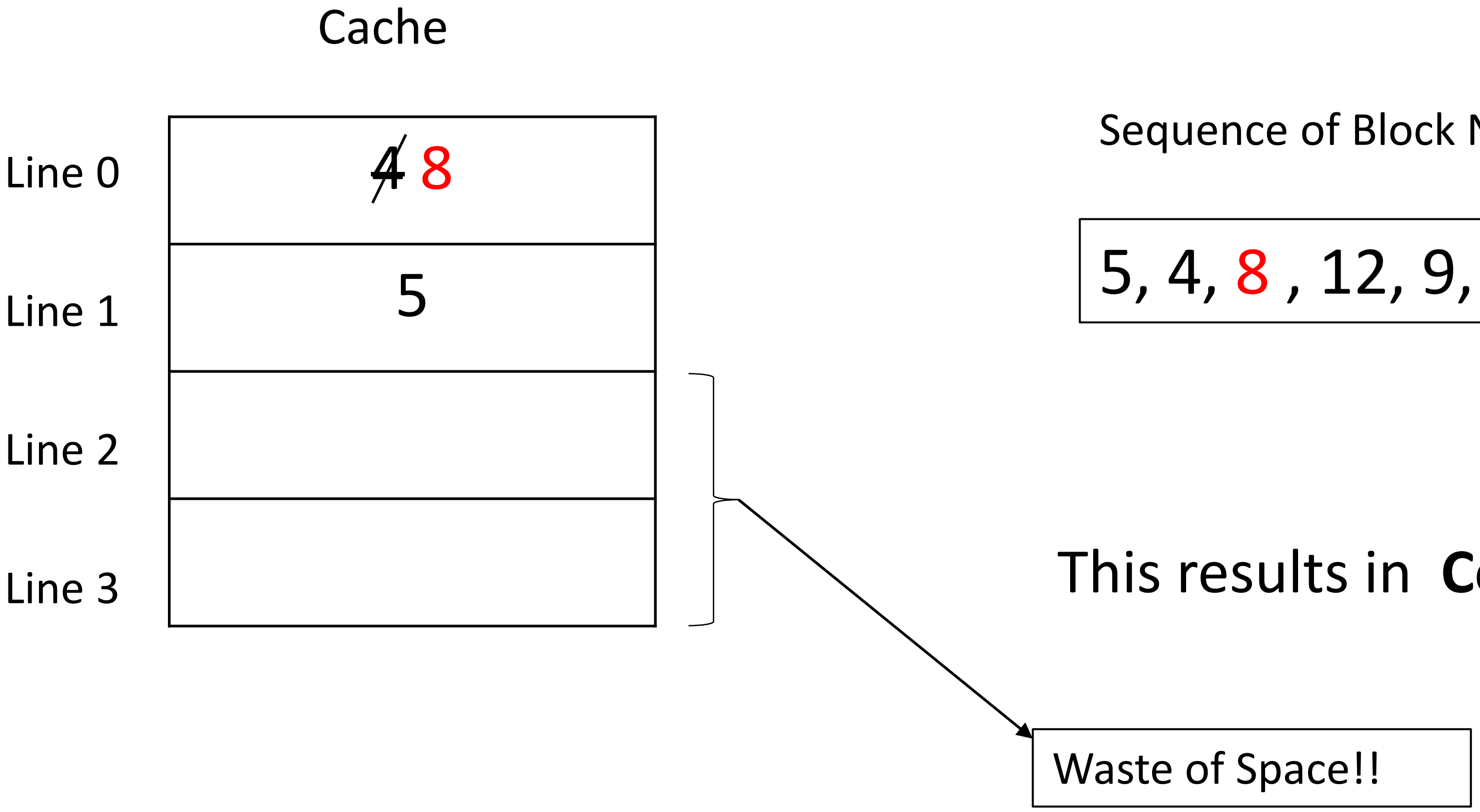
Cache

Line 0	4
Line 1	5
Line 2	
Line 3	

Sequence of Block Numbers requested by CPU

5, 4, 8 , 12, 9, 13

Disadvantages of Direct Mapping



Cache

Line 0	~~4~~ 8
Line 1	5
Line 2	
Line 3	

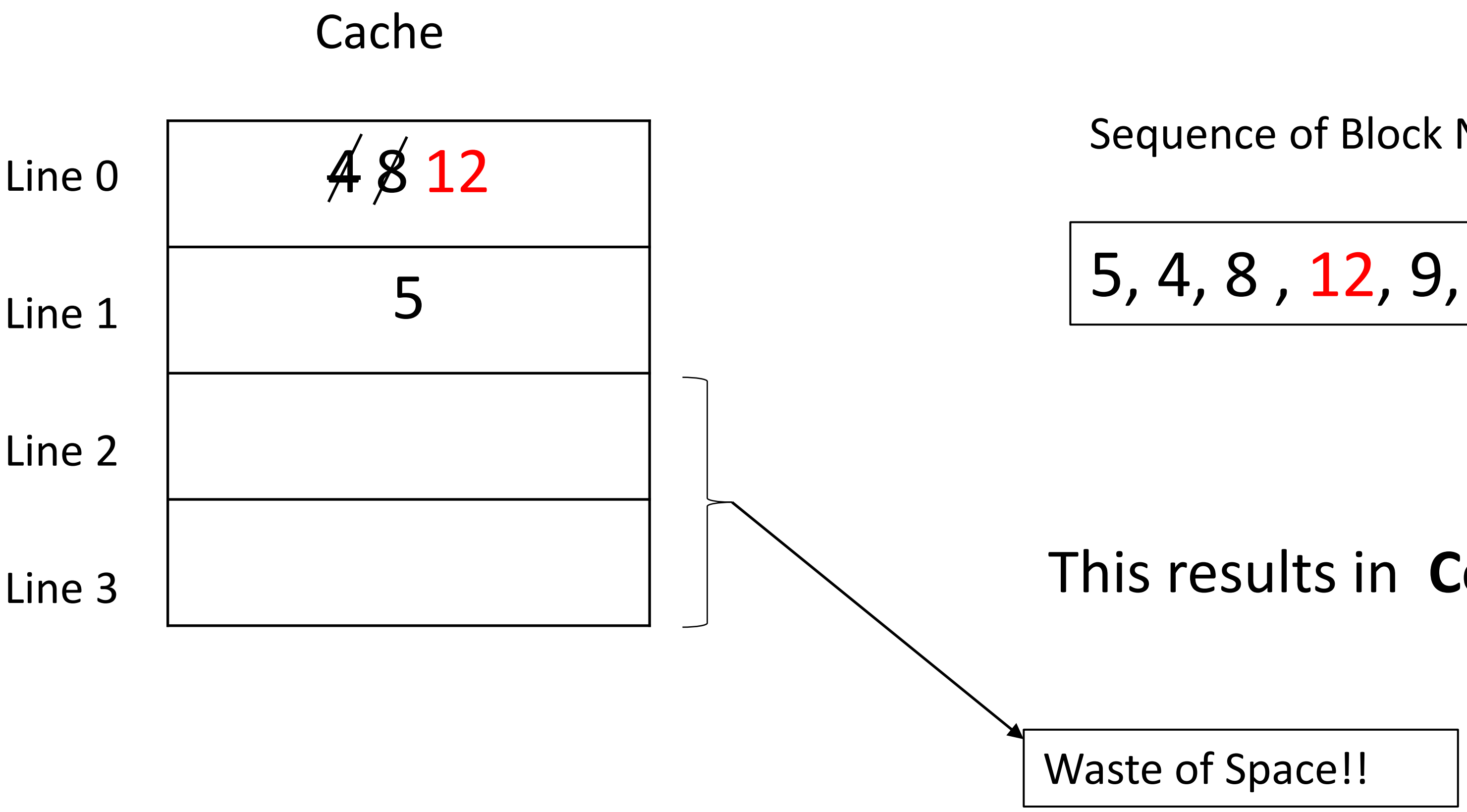
Waste of Space!!

Sequence of Block Numbers requested by CPU

5, 4, 8 , 12, 9, 13

This results in **Conflict MISS!!**

Disadvantages of Direct Mapping



Cache

Line 0	~~4~~ ~~8~~ 12
Line 1	5
Line 2	
Line 3	

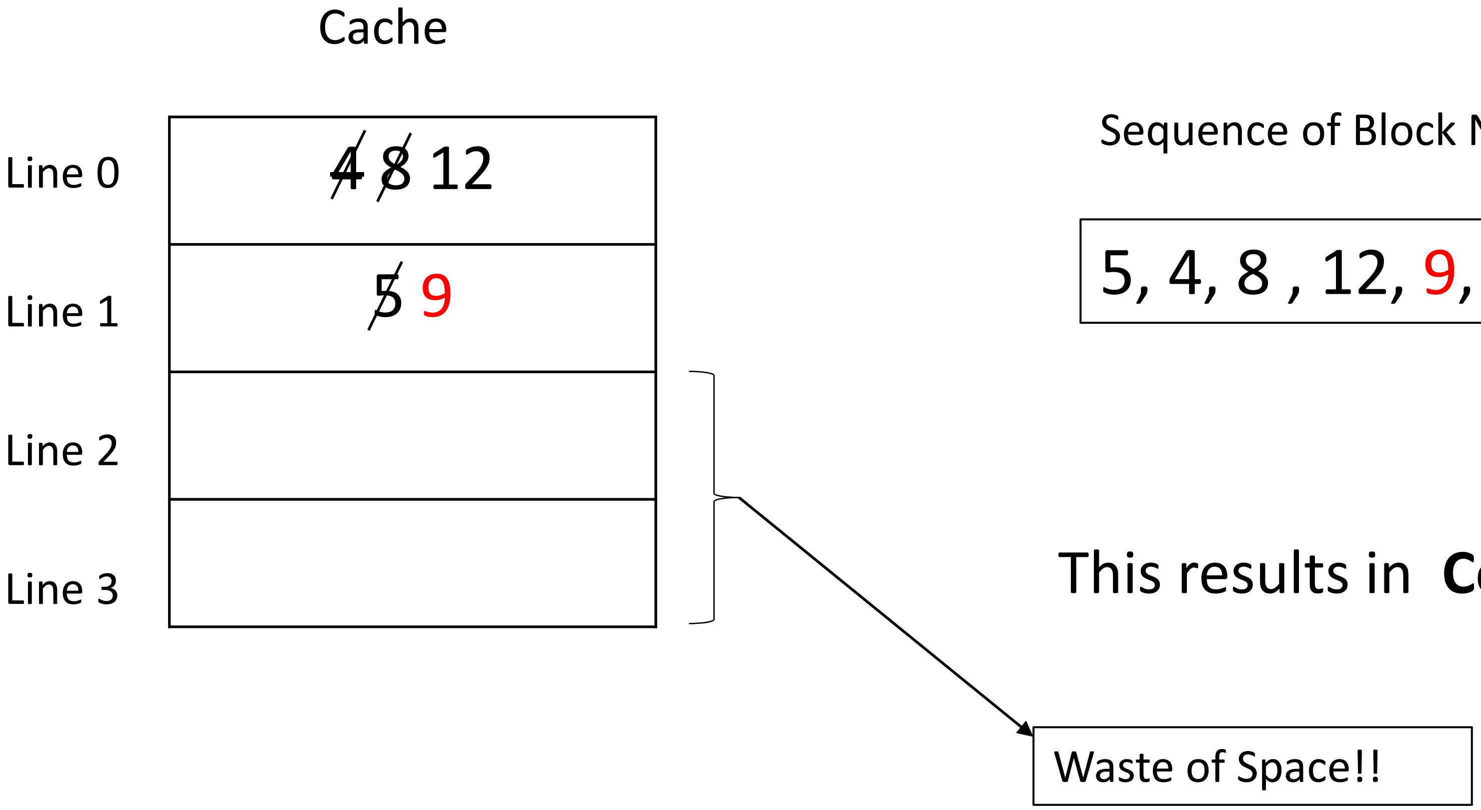
Waste of Space!!

Sequence of Block Numbers requested by CPU

5, 4, 8 , 12, 9, 13

This results in **Conflict MISS!!**

Disadvantages of Direct Mapping



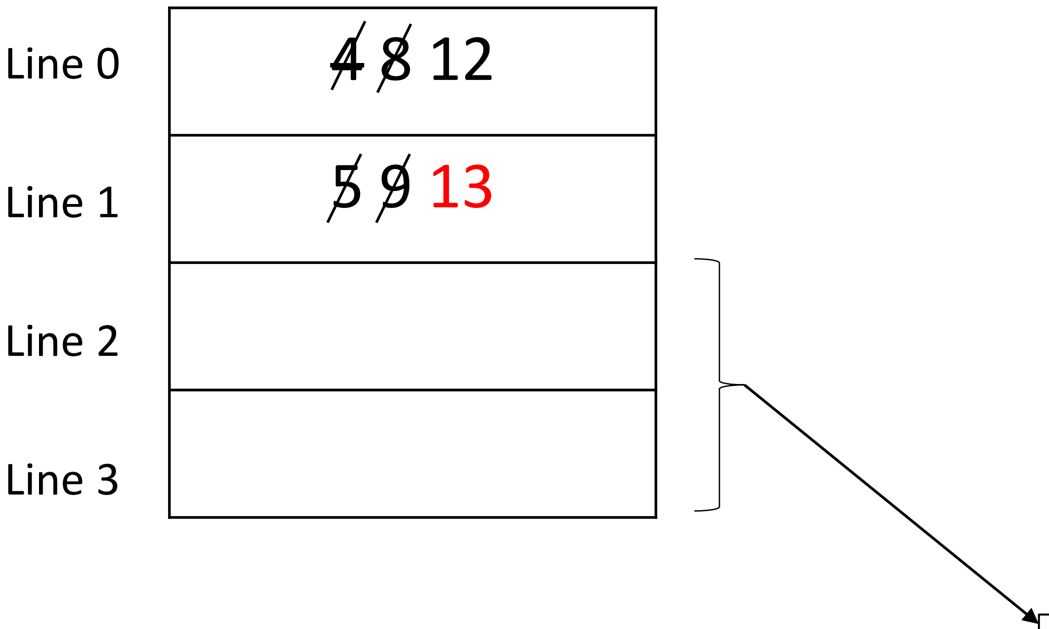
Sequence of Block Numbers requested by CPU

5, 4, 8 , 12, 9, 13

This results in **Conflict MISS!!**

Disadvantages of Direct Mapping

Cache



Line 0	~~4~~ ~~8~~ 12
Line 1	~~5~~ ~~9~~ 13
Line 2	
Line 3	

Sequence of Block Numbers requested by CPU

5, 4, 8 , 12, 9, 13

This results in **Conflict MISS!!**

Waste of Space!!

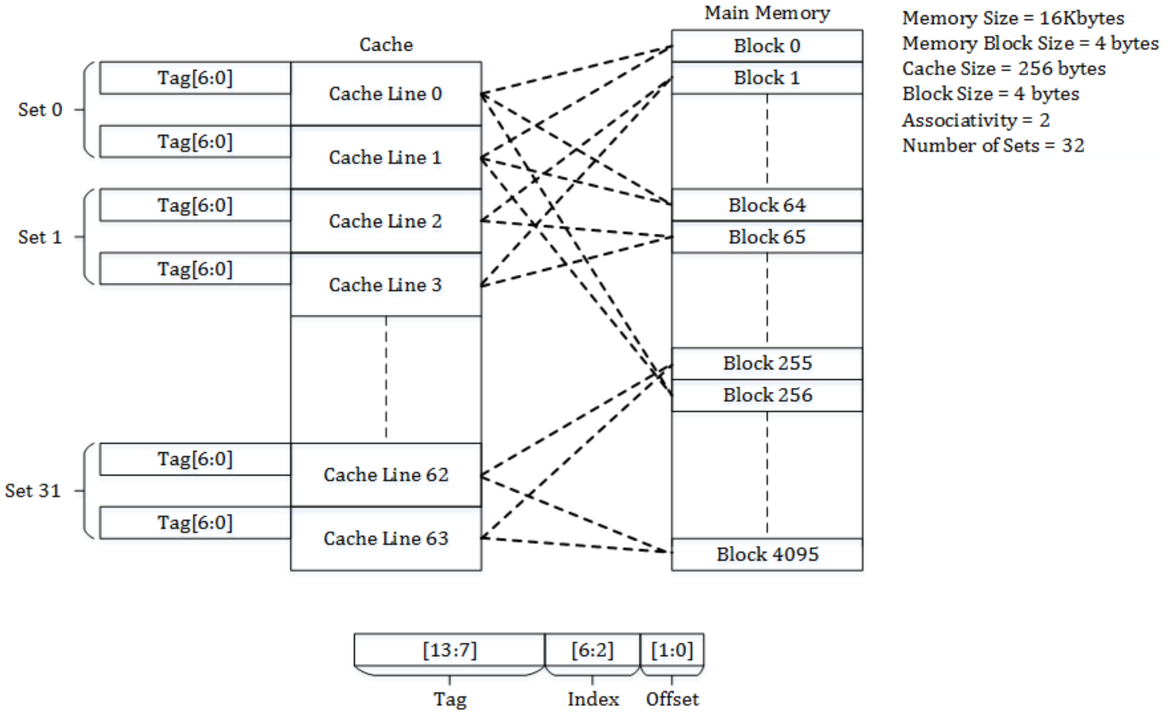
Solutions

- Fully Associative Mapping
  - A Main Memory Block can be mapped to any Cache line.
  - Advantages: Better Cache Hit Rate

  - **Disadvantages**
    - Slow because the valid bit and tag of every cache line has to be compared.
    - Expensive due to the high cost of associative-comparison hardware.

Solutions

- Set Associative Mapping
  - Cache lines grouped into sets
  - A main memory block is mapped to a set
    - Associative Mapping within a set
  - Tradeoff between Direct Mapping and Fully Associative Mapping
  - **Disadvantages**
    - Can still suffer from conflict miss.



- More details in next recitation

References

- https://www.youtube.com/watch?v=VePK5TNgQU8

- https://www.youtube.com/watch?v=N_OJn7jdKCc

- https://en.wikipedia.org/wiki/Cache_placement_policies

- Locality