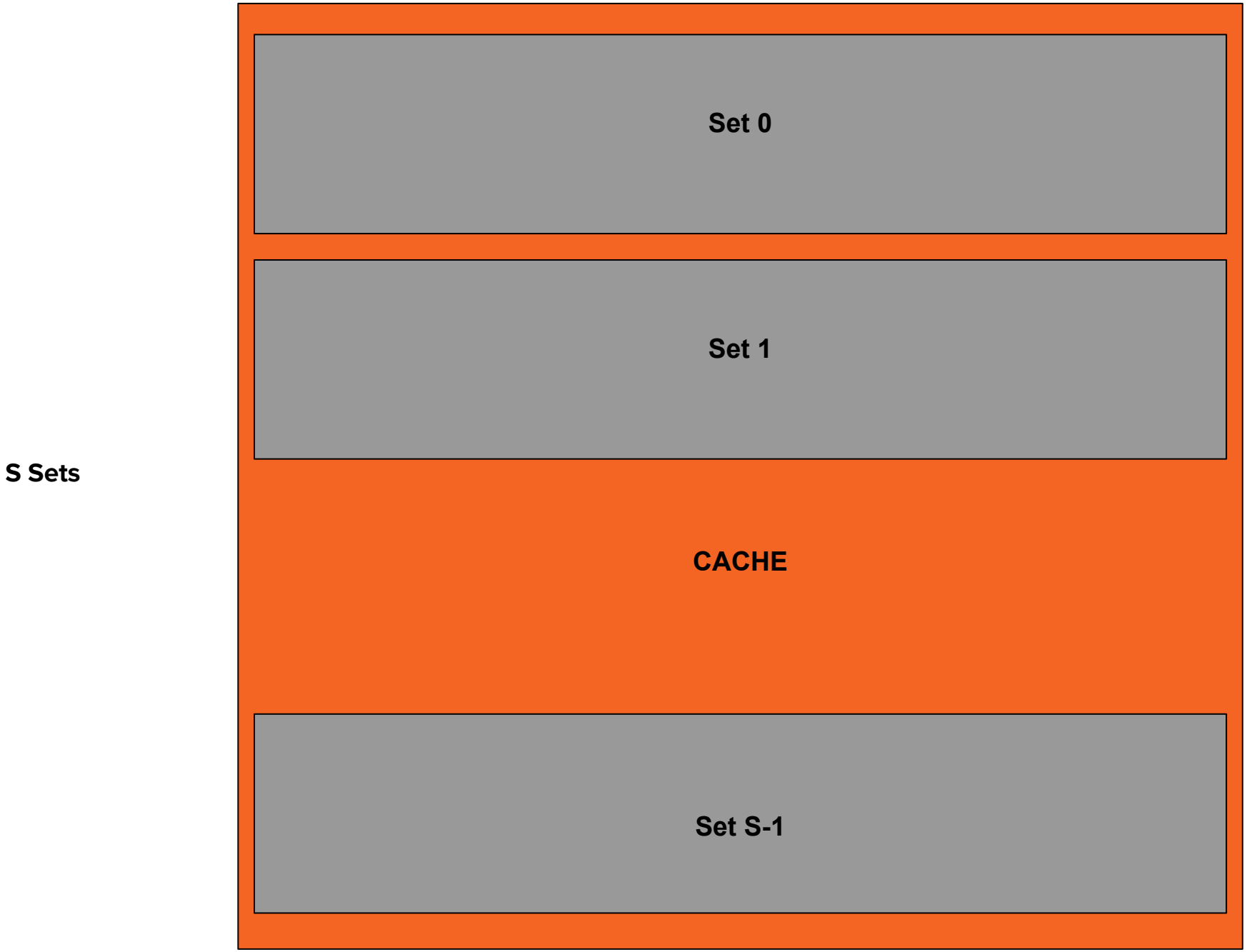
Cache Organization II

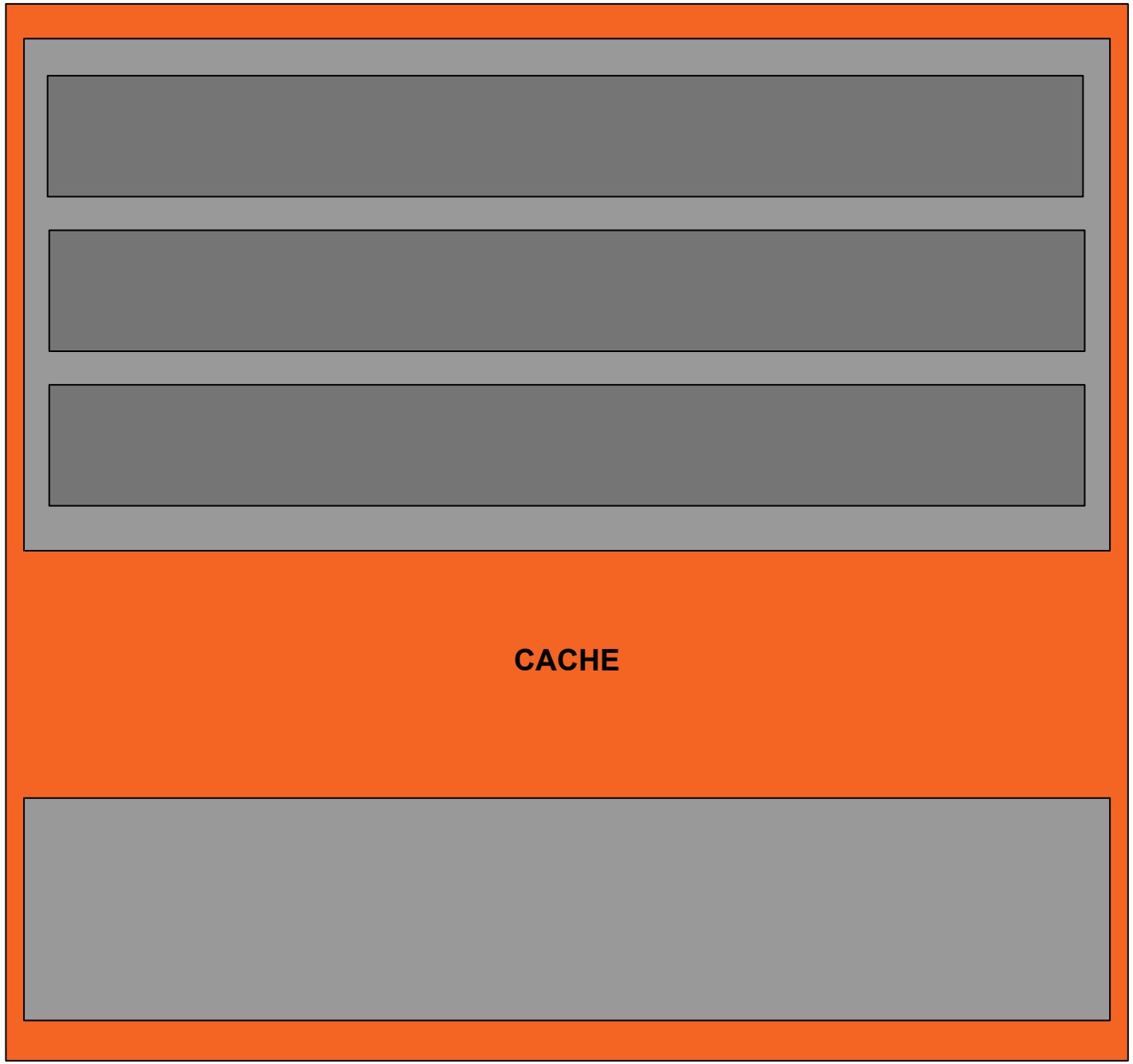
March 28, 2016 . Ganesh Kumar

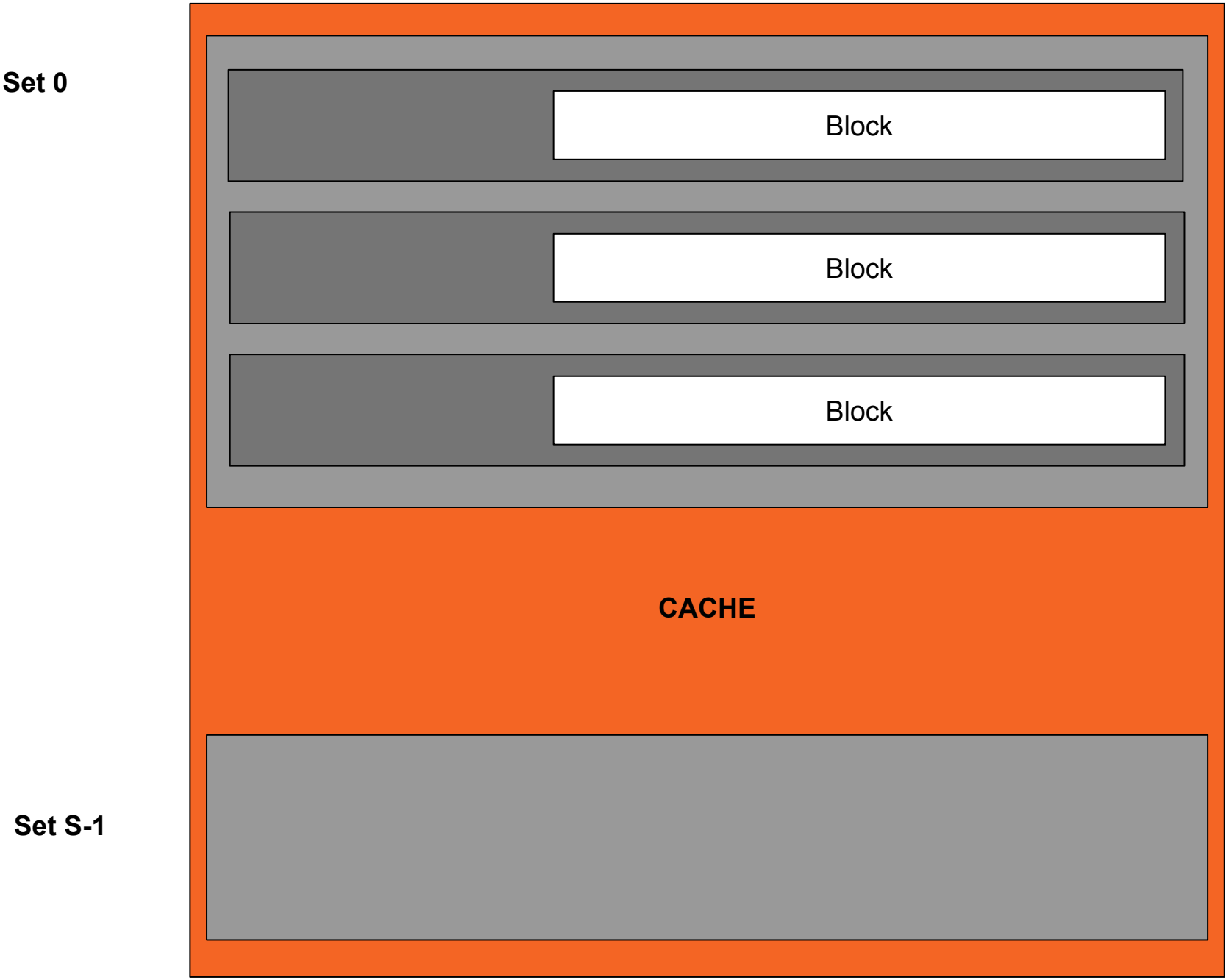

© R.Baird

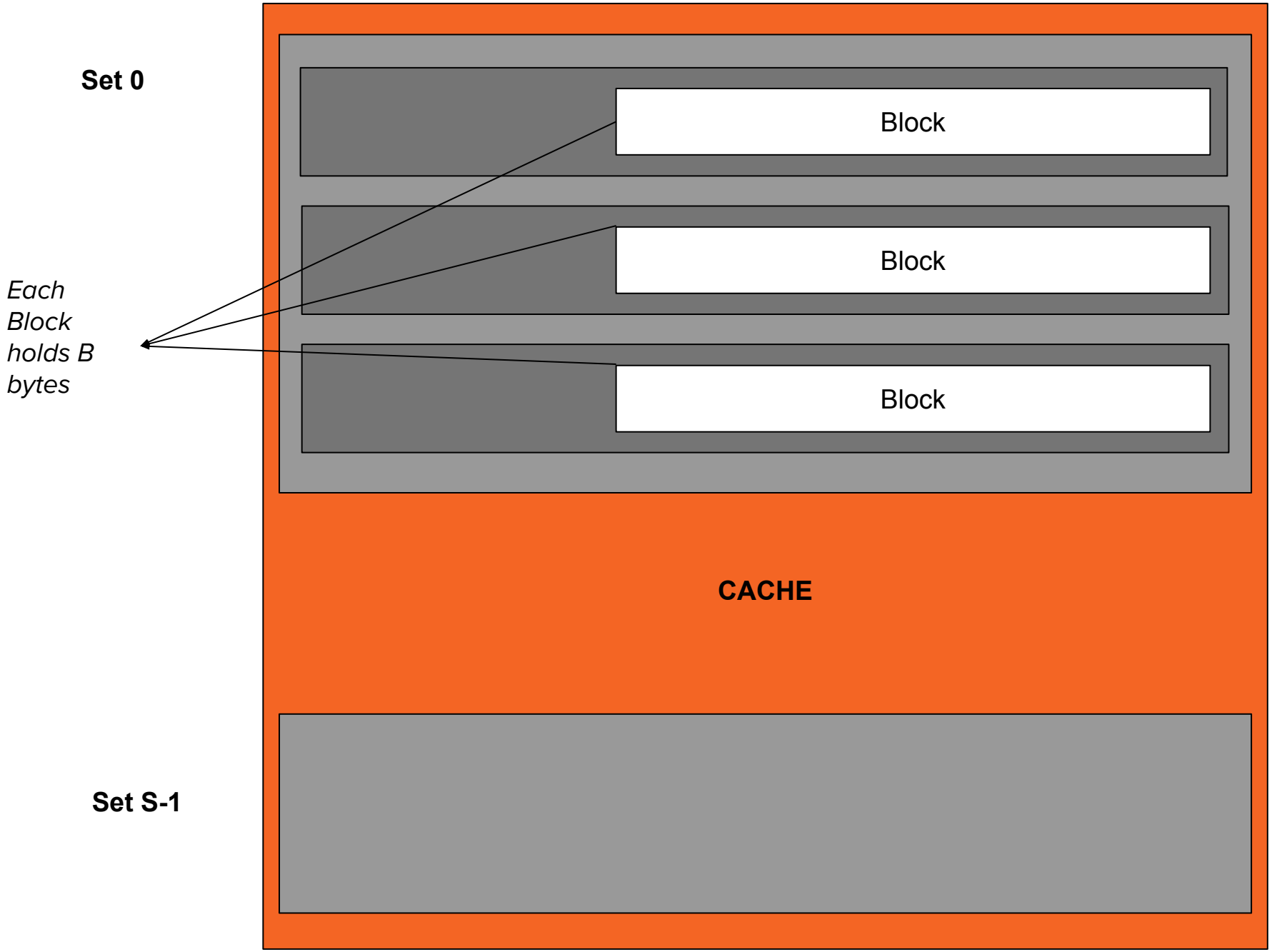

HOME
SWEET
HOME

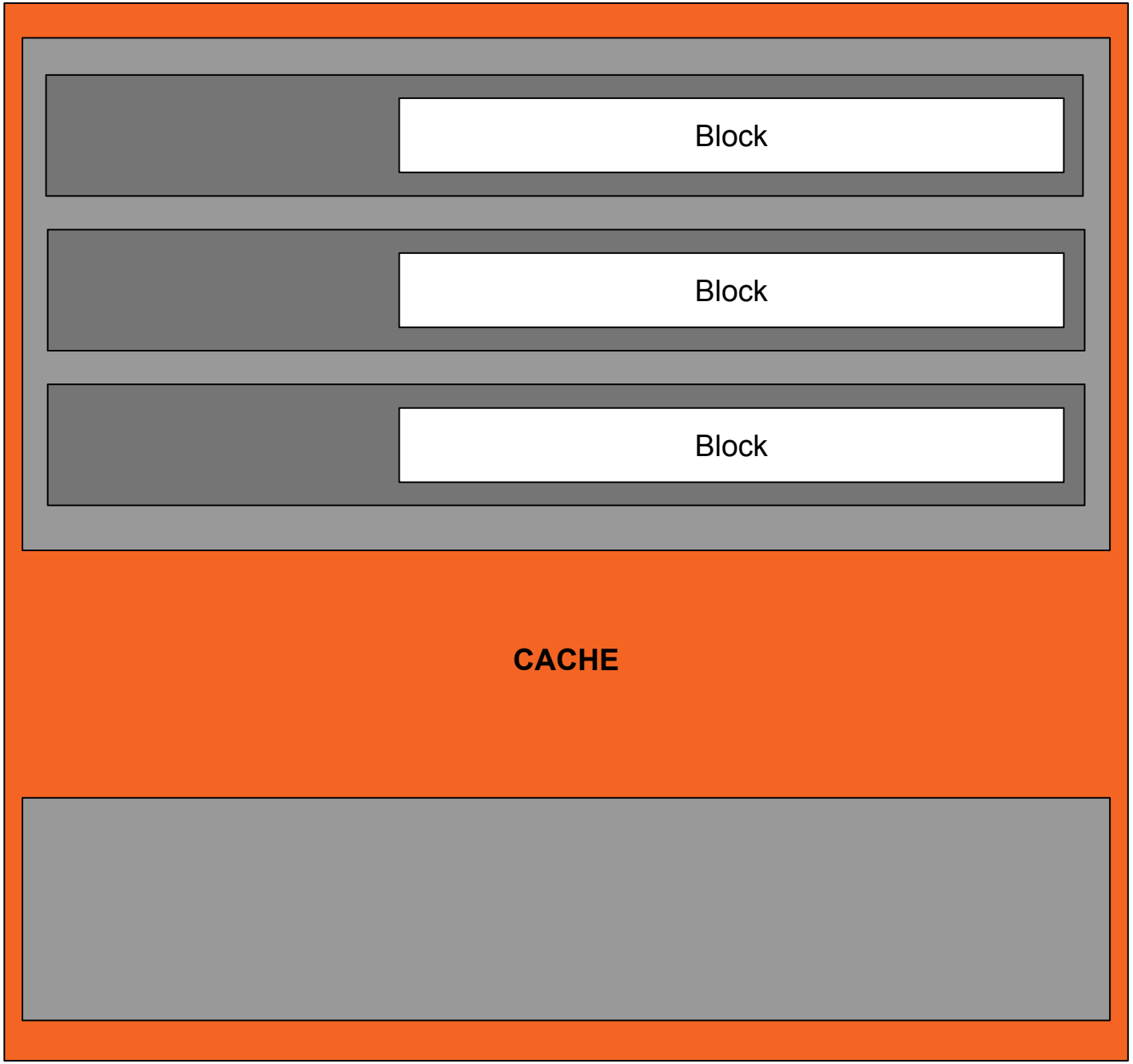

CACHE


Set 0
Set 1
S Sets
CACHE
Set S-1


Set 0
E *Lines* in **each** Set.
In this example, E=3
CACHE
Set S-1


Set 0
Block
Block
Block
CACHE
Set S-1


Set 0
Block
Block
Block
Each Block holds B bytes
CACHE
Set S-1



Size of the Cache = S x E x B bytes

(not including overhead such as tag and valid bits)

A Direct Mapped Cache Example

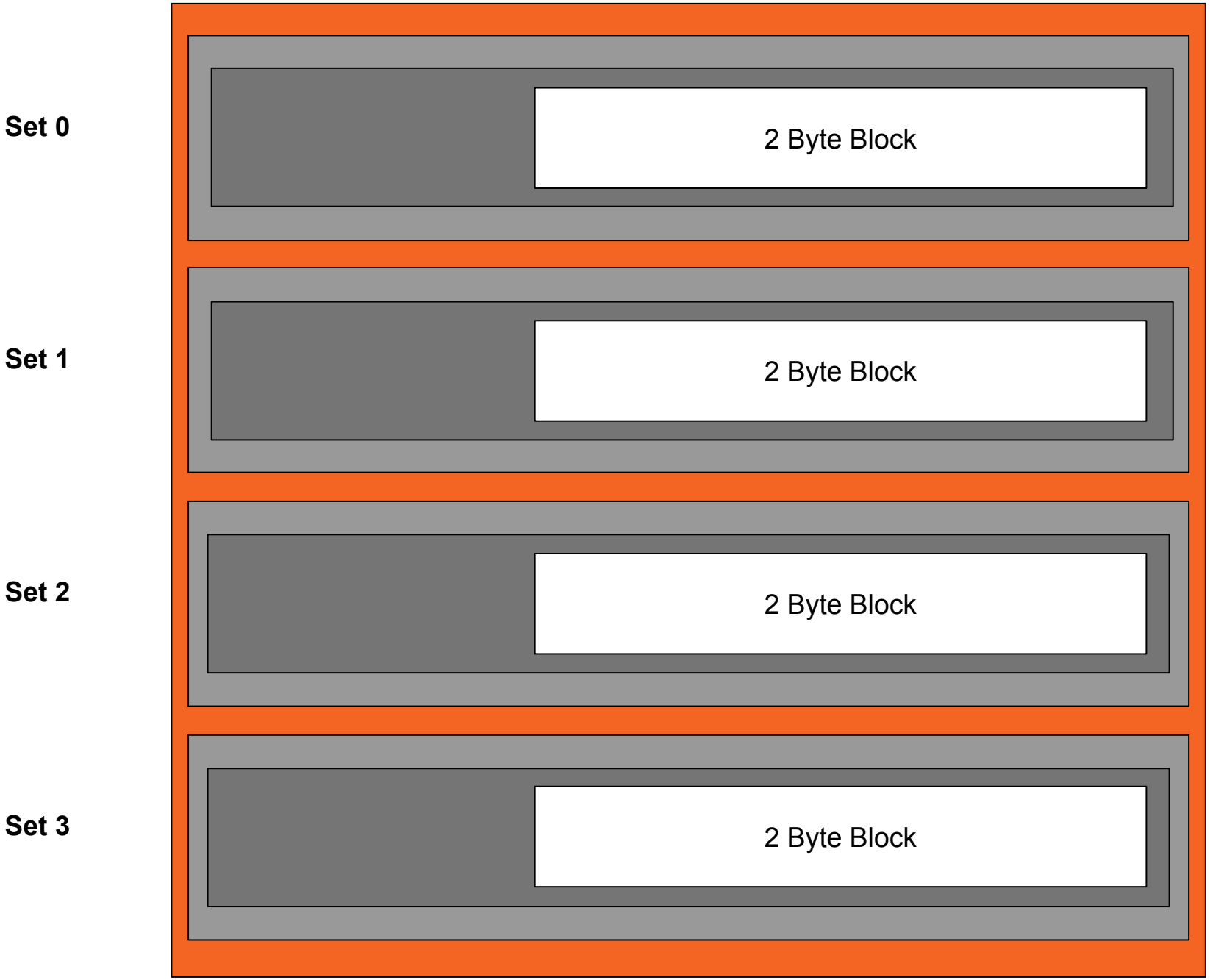
Description

(S, E, B, m) = (4, 1, 2, 4)

- ➔ 4 sets
- ➔ 1 line per set (Direct Mapped!)
- ➔ 2 bytes per block
- ➔ 4-bit address space

Address Split

- ➔ s = 2 bits
- ➔ b = 1 bit
- ➔ t = m - (s + b) => 1 bit


Set 0
2 Byte Block
Set 1
2 Byte Block
Set 2
2 Byte Block
Set 3
2 Byte Block

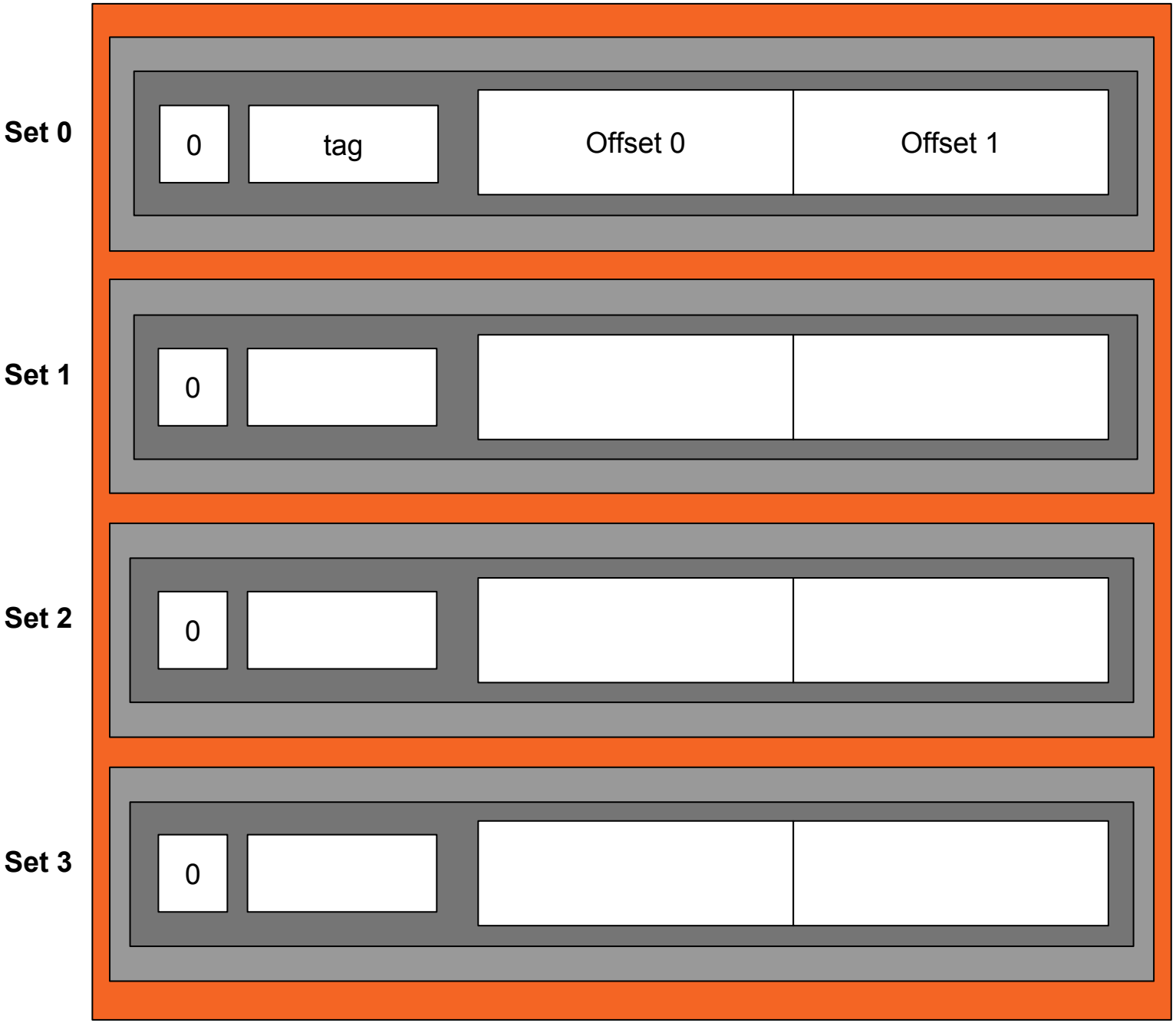
Address (decimal)	Address bits: Tag bits ($t = 1$)	Index bits ($s = 2$)	Offset bits ($b = 1$)	Block number (decimal)
0	0	00	0	0
1	0	00	1	0
2	0	01	0	1
3	0	01	1	1
4	0	10	0	2
5	0	10	1	2
6	0	11	0	3
7	0	11	1	3
8	1	00	0	4
9	1	00	1	4
10	1	01	0	5
11	1	01	1	5
12	1	10	0	6
13	1	10	1	6
14	1	11	0	7
15	1	11	1	7

Figure 6.32 **4-bit address space for example direct-mapped cache.**

Points to Note

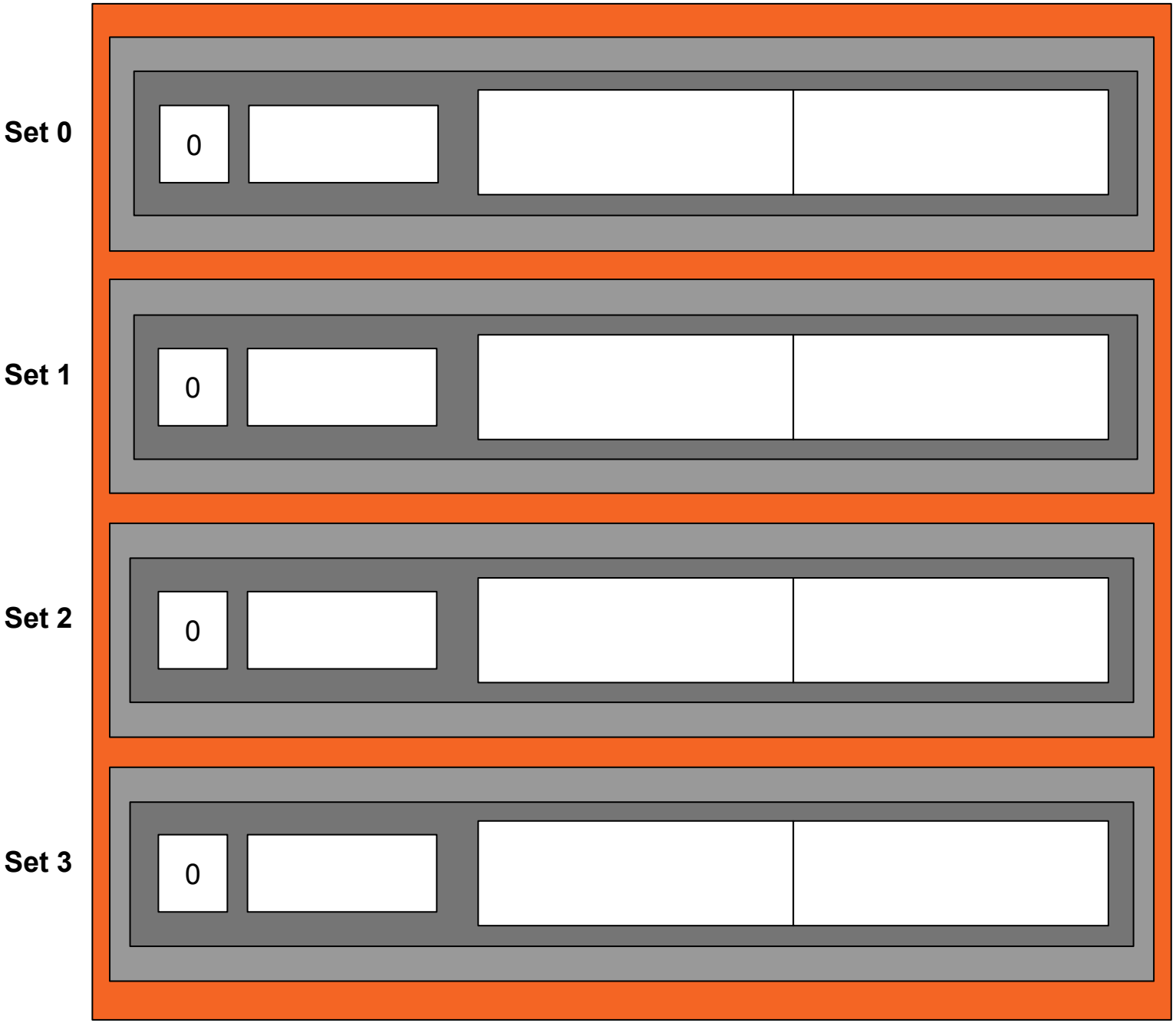
- ➔ Tag + Index bits together uniquely identify each block in memory.
- ➔ 8 memory blocks but only 4 cache sets. Multiple blocks map to same cache set i.e. they have the same cache bit.
- ➔ Blocks that map to the same set can be differentiated/identified by using the tag bits.

Initial State of the cache



Read word (in this case, a byte) at address 0000.

0 00 0
Set = 00

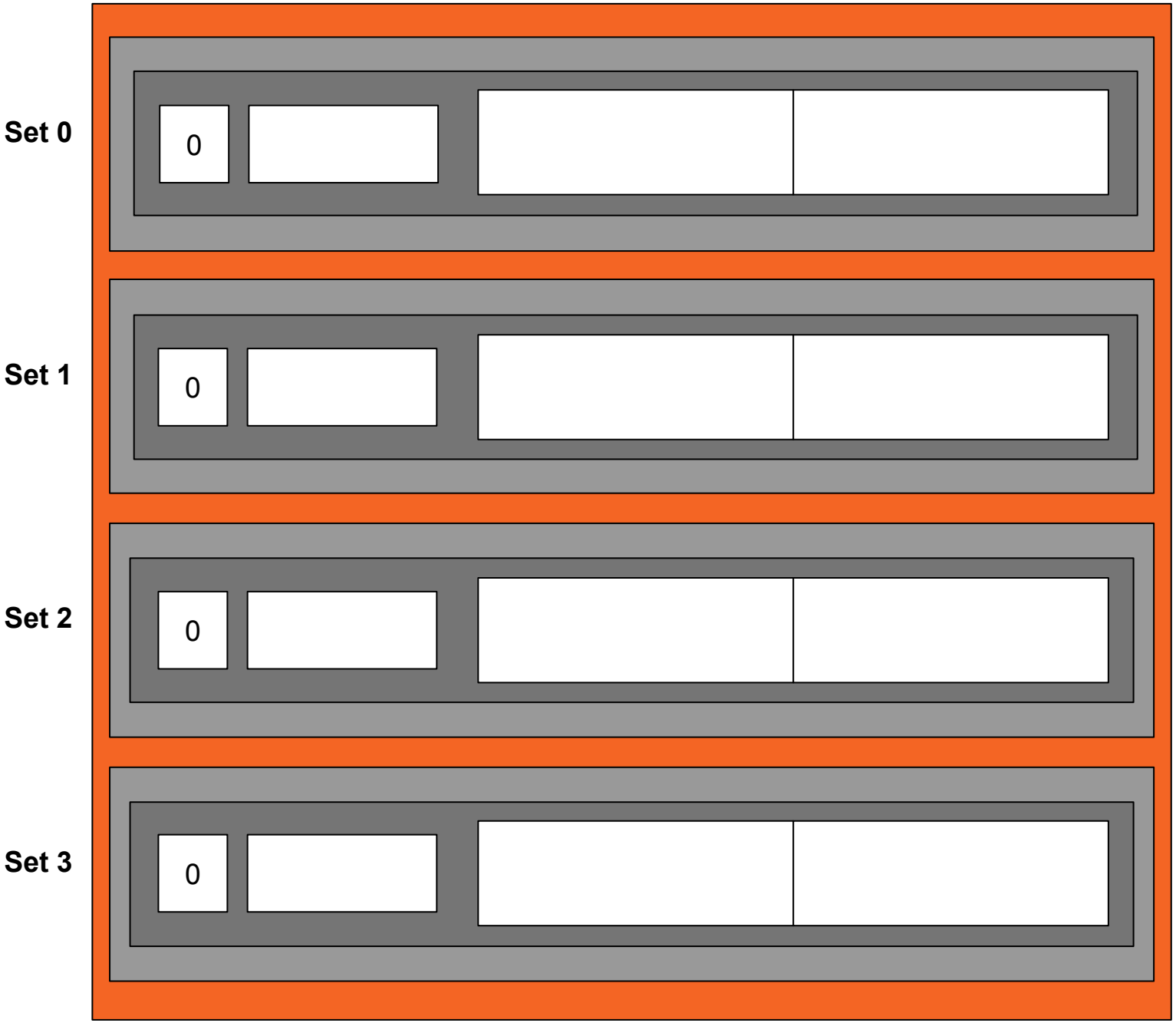


Read word (in this case, a byte) at address 0000.

0 00 0
Set = 00

Valid bit is 0!
Cache miss!

Fetch block 0 from memory



Read word (in this case, a byte) at address 0000.

0 00 0
Set = 00

Valid bit is 0!
Cache miss!

Fetch block 0 from memory

Read word at address 0001.

0 00 1
Set = 00
Tag = 0
Offset = 1

Cache hit!

Read word at address 1101.

1 10 1
Set = 10
Tag = 1
Offset = 1

Cache line in set 2 is not valid.

Cache miss!!

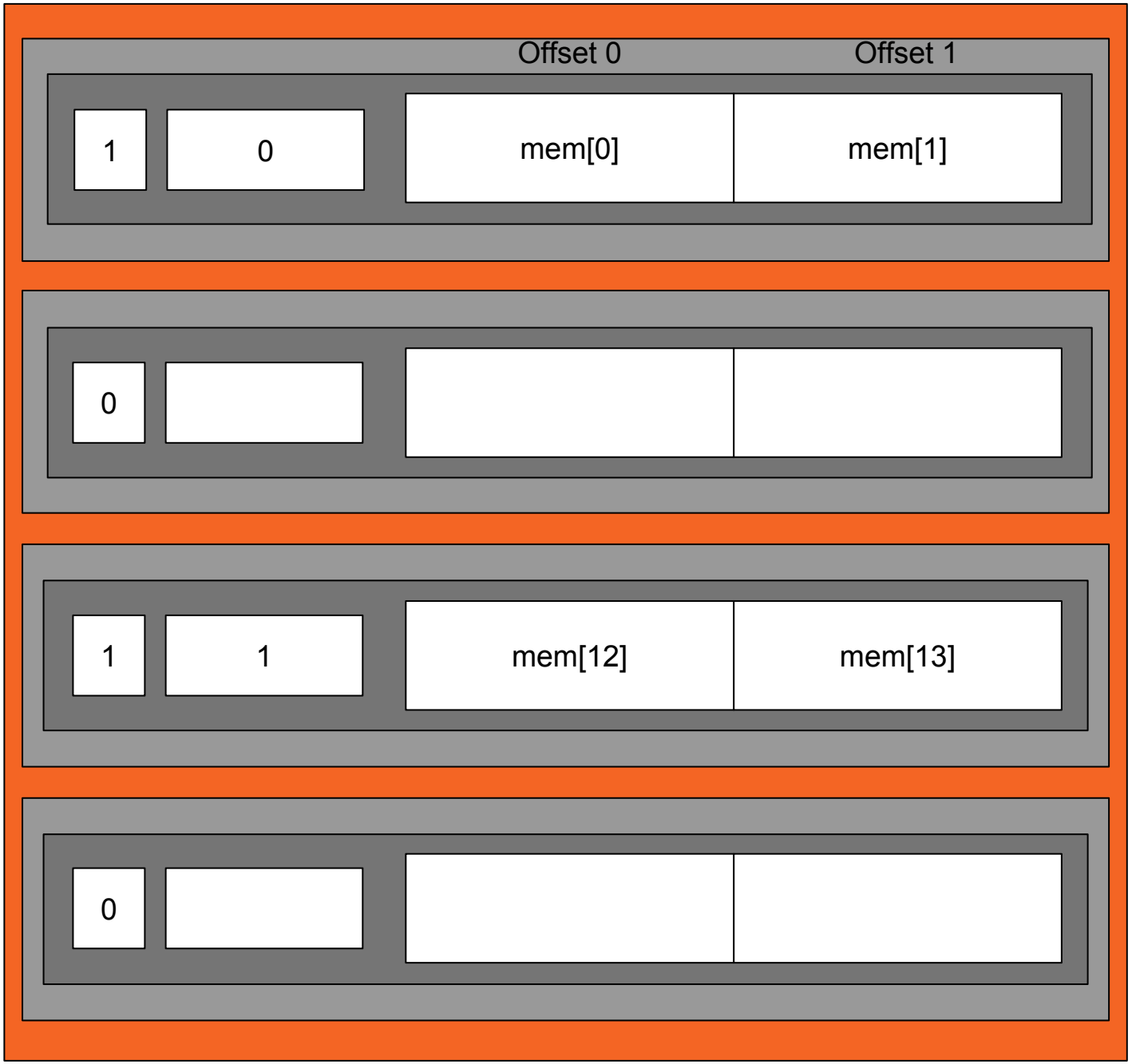
Read word at address 1101.

1 10 1
Set = 10
Tag = 1
Offset = 1

Cache line in set 2 is not valid.

Cache miss!!

So load that block into set 2



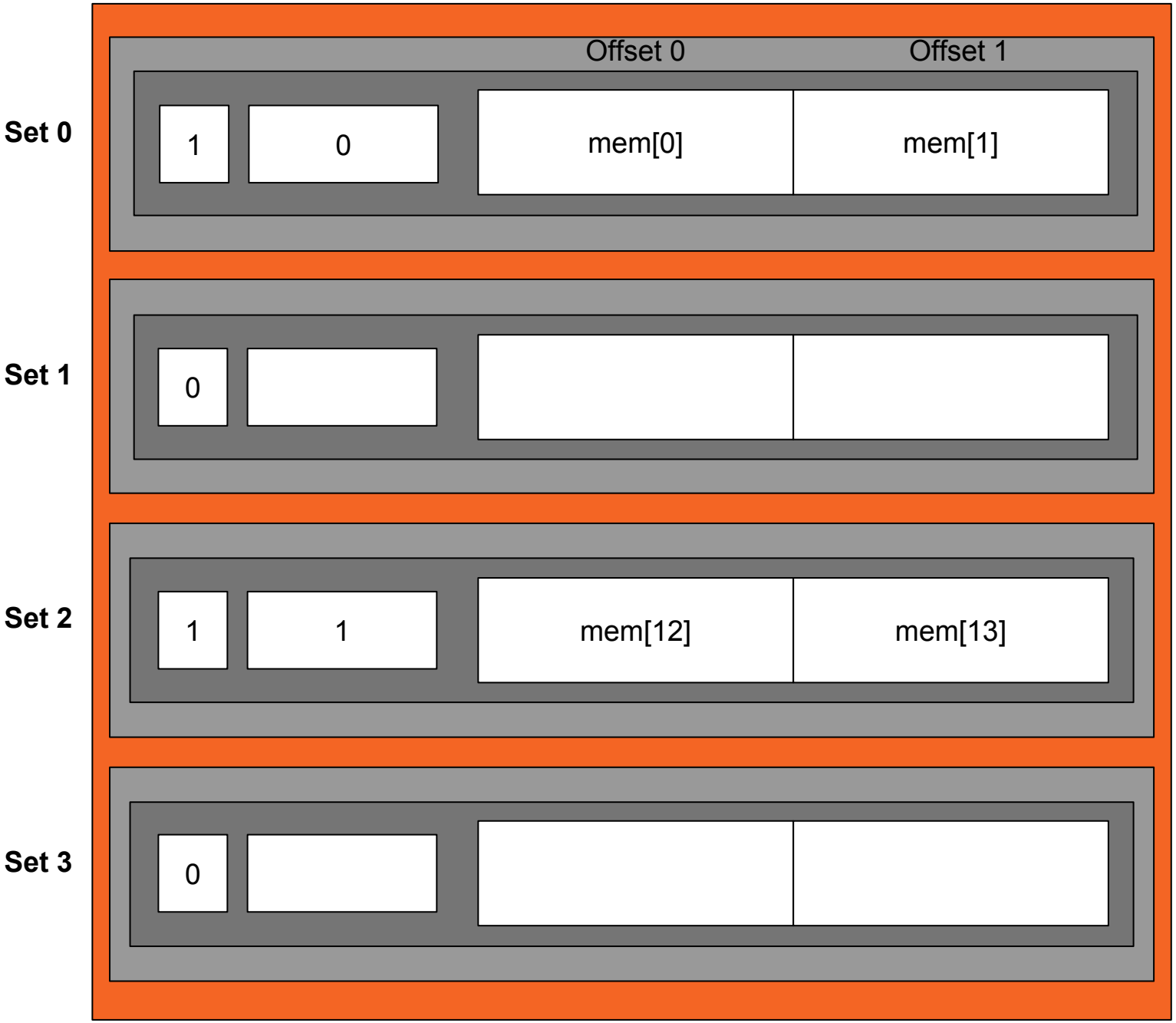
Read word at address 1000.

1 00 0
Set = 00
Tag = 1
Offset = 0

Cache line in set 0 is valid!

But tag bit does not match.

Cache miss!



Read word at address 1000.

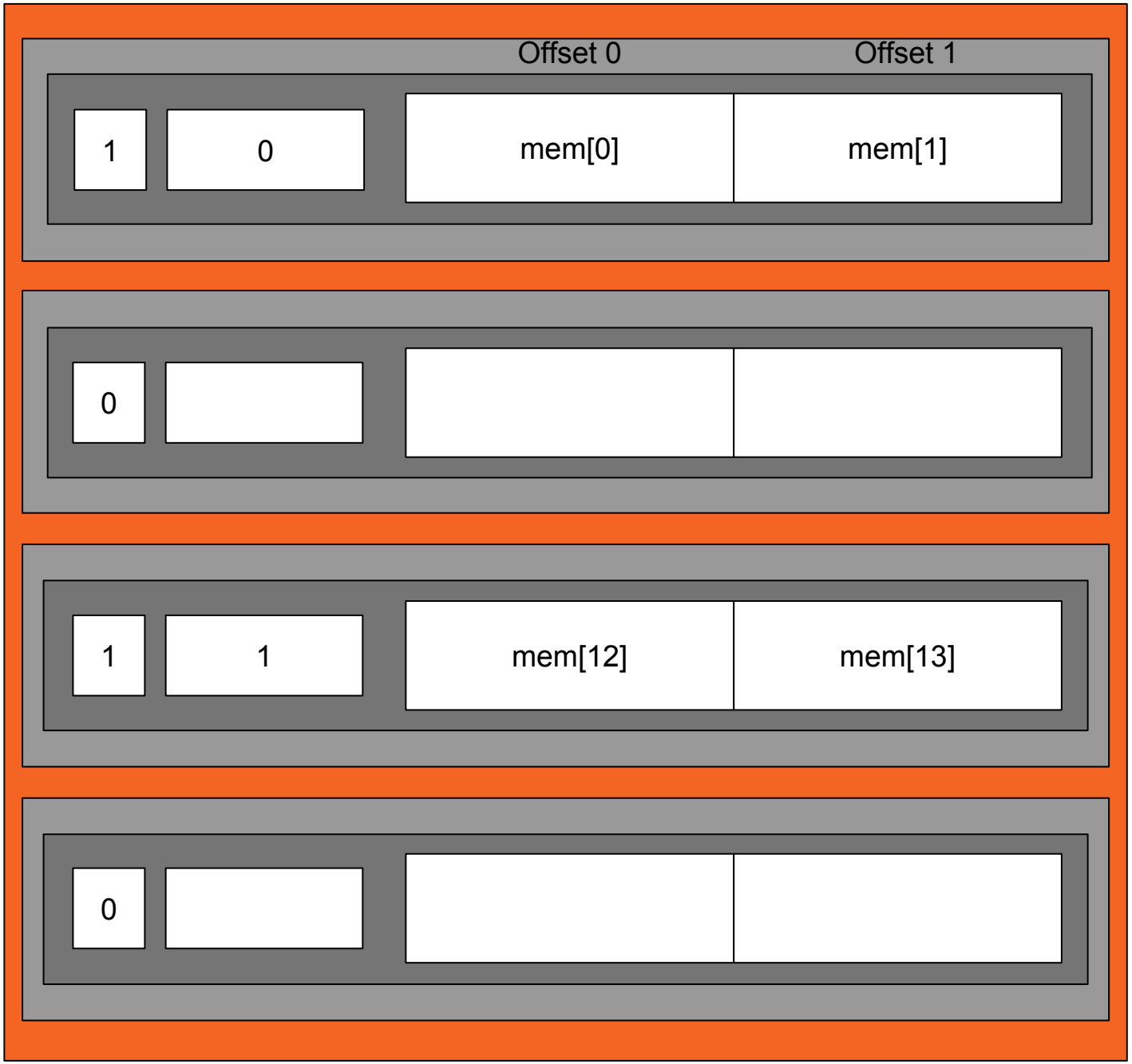
1 00 0
Set = 00
Tag = 1
Offset = 0

Cache miss!

So load block that contains this address 1000 onto cache.

And **replace** the existing line.

(Simple replacement policy!)



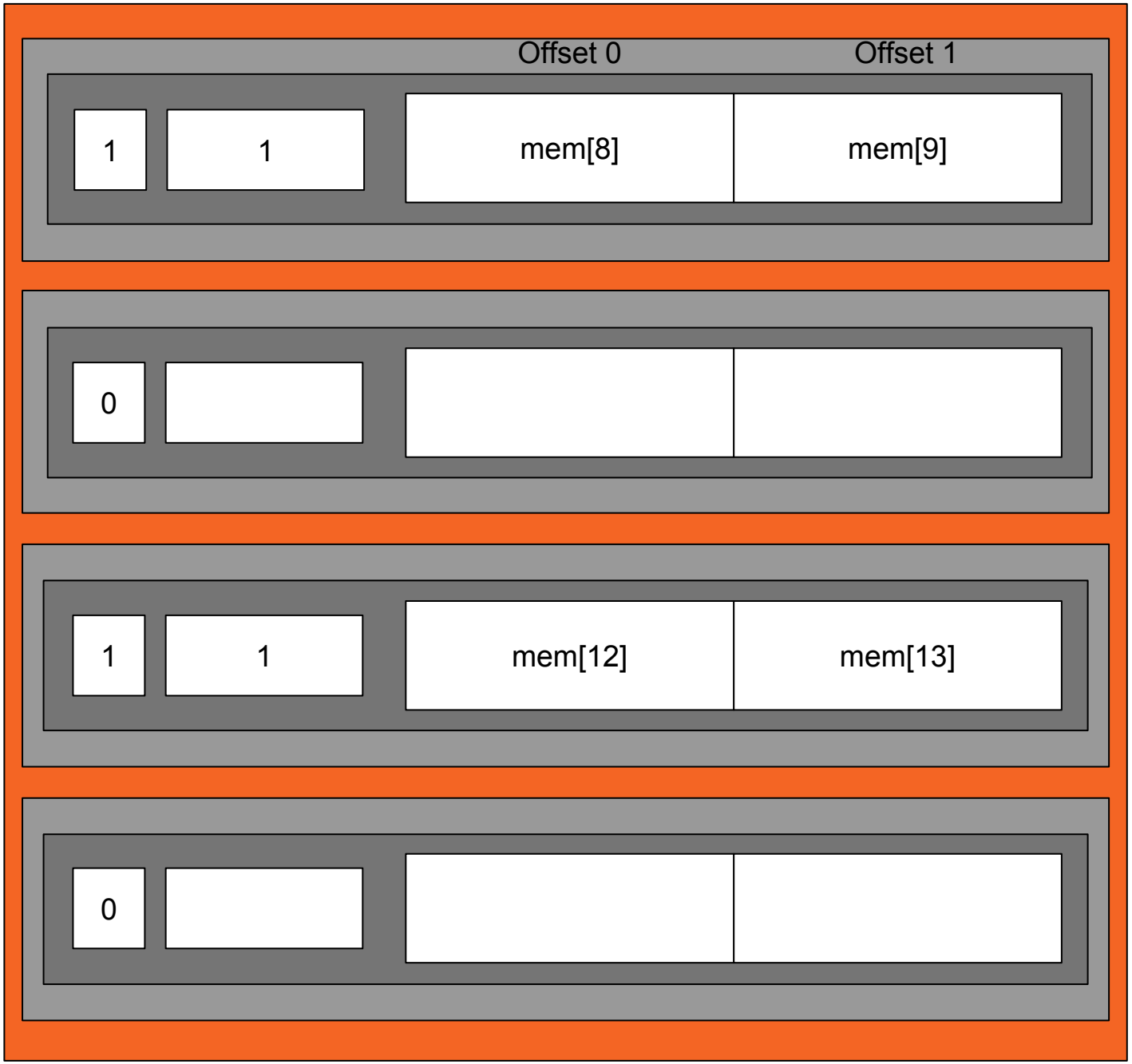
Read word at address 1000.

1 00 0
Set = 00
Tag = 1
Offset = 0

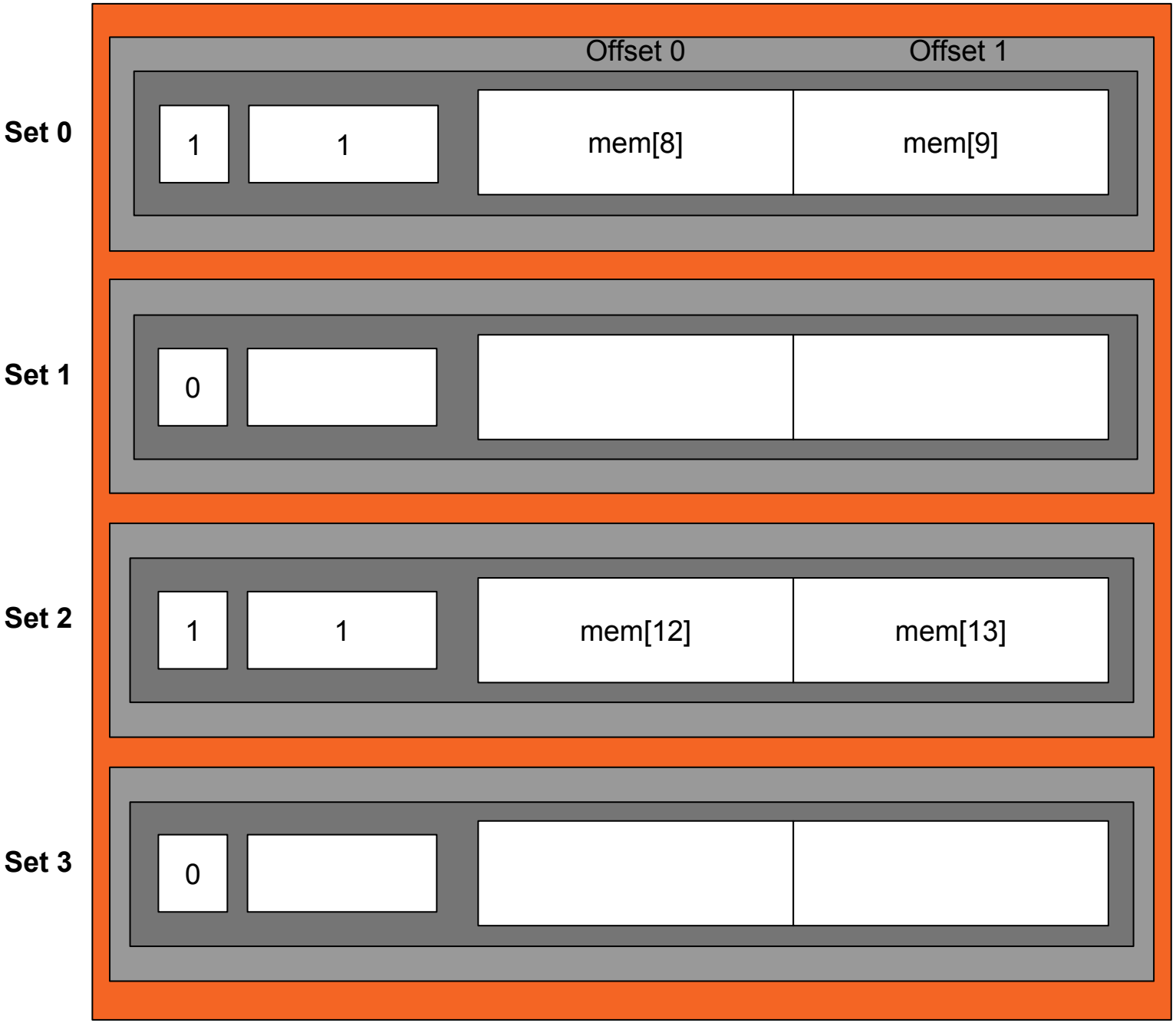
Cache miss!

So load block that contains this address 1000 onto the cache.

And **replace** the existing line.



What happens if we try to read word at address 0000 again?



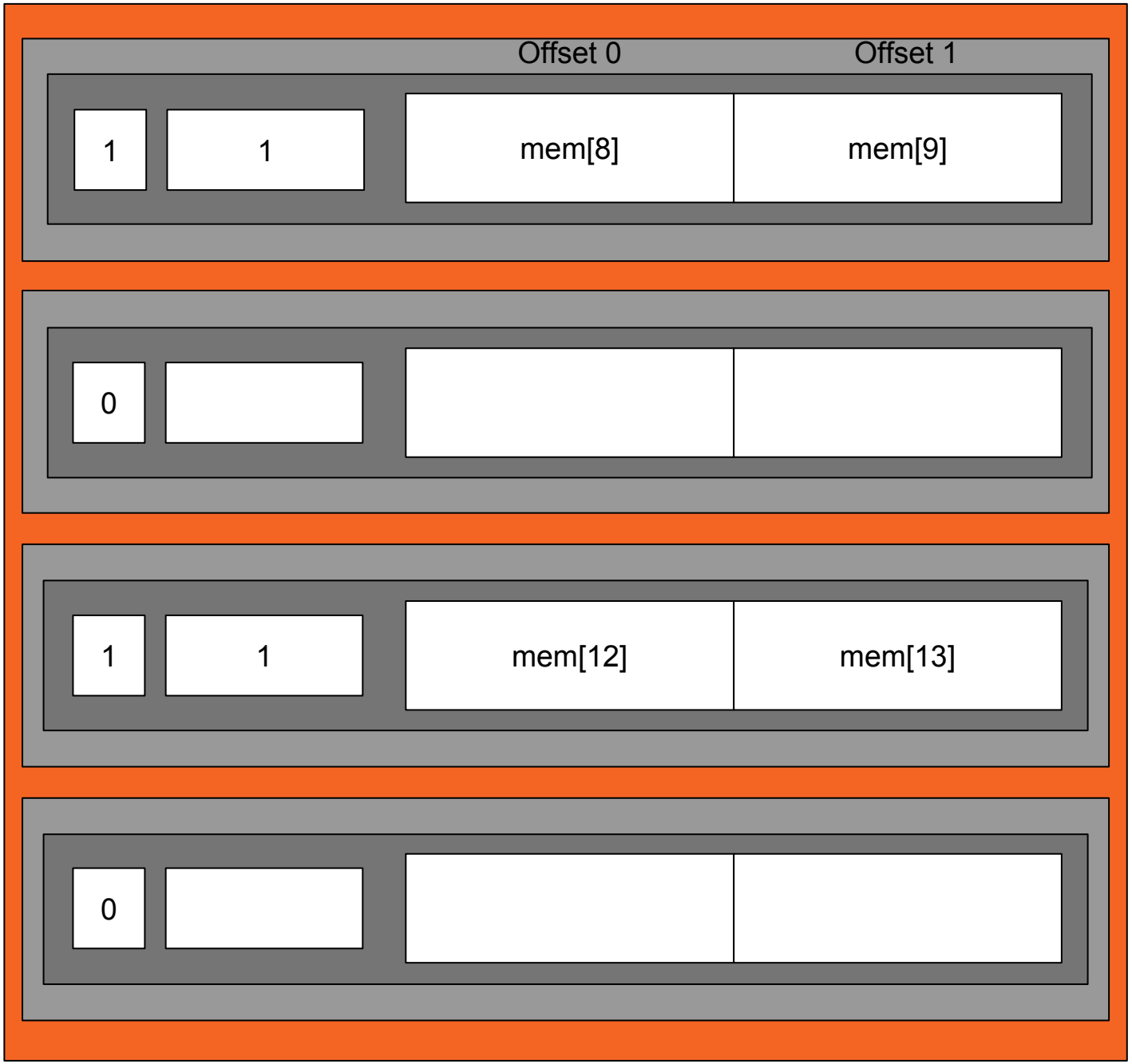
What happens if we try to read word at address 0000 again?

And after that, read 1000?

And so on?

Conflict misses.

(Because there is free space in the cache and yet we get misses)



Any way we can avoid this?

More cache lines!

Set Associative Caches!

3/28

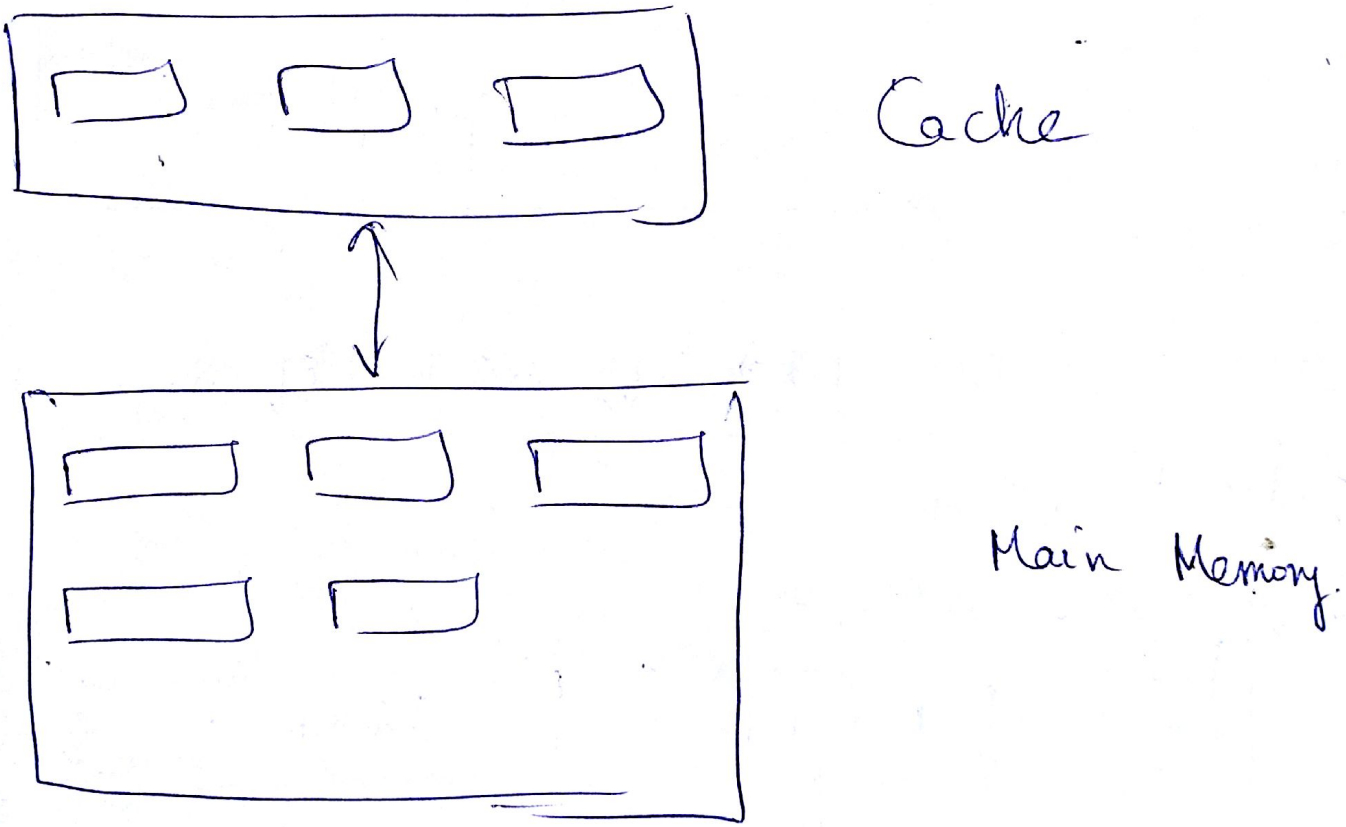
Today

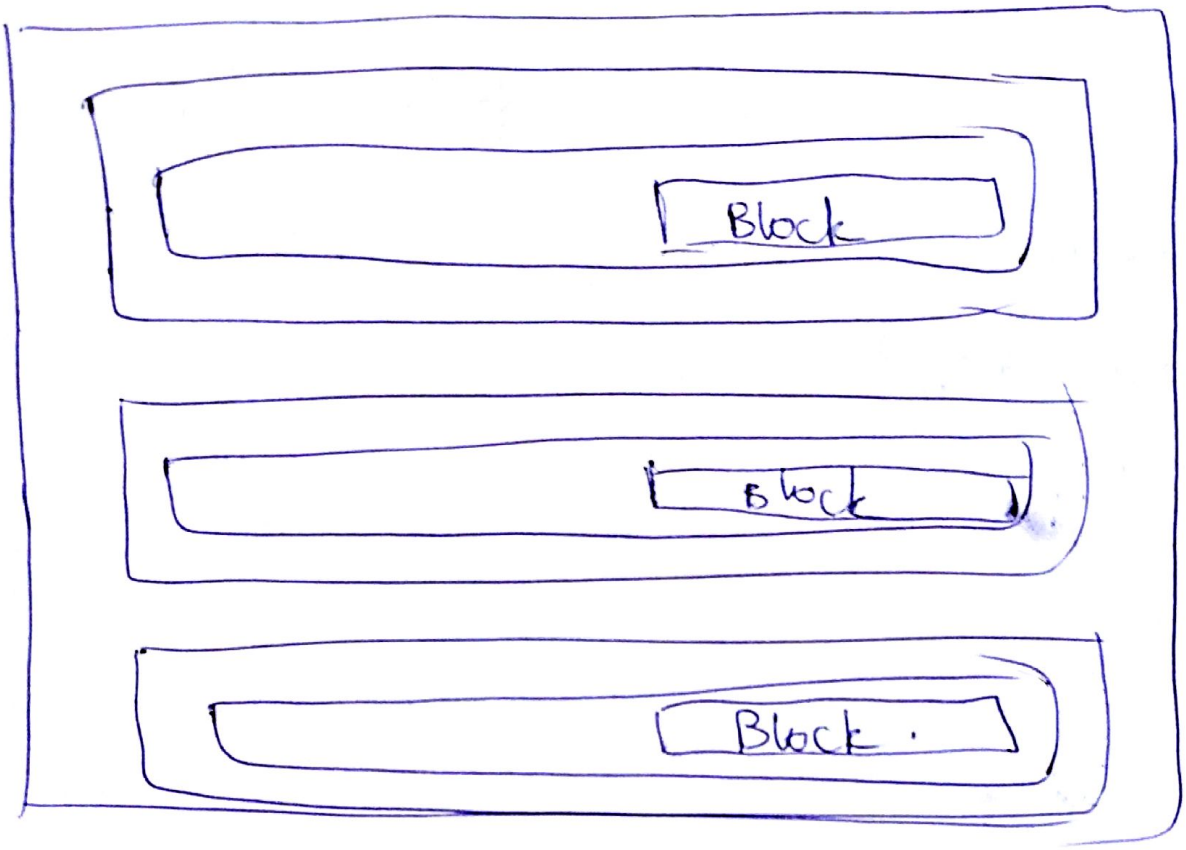
- Section 6.4
- Midterms

→ Review.

→ Direct Mapped Cache

Example

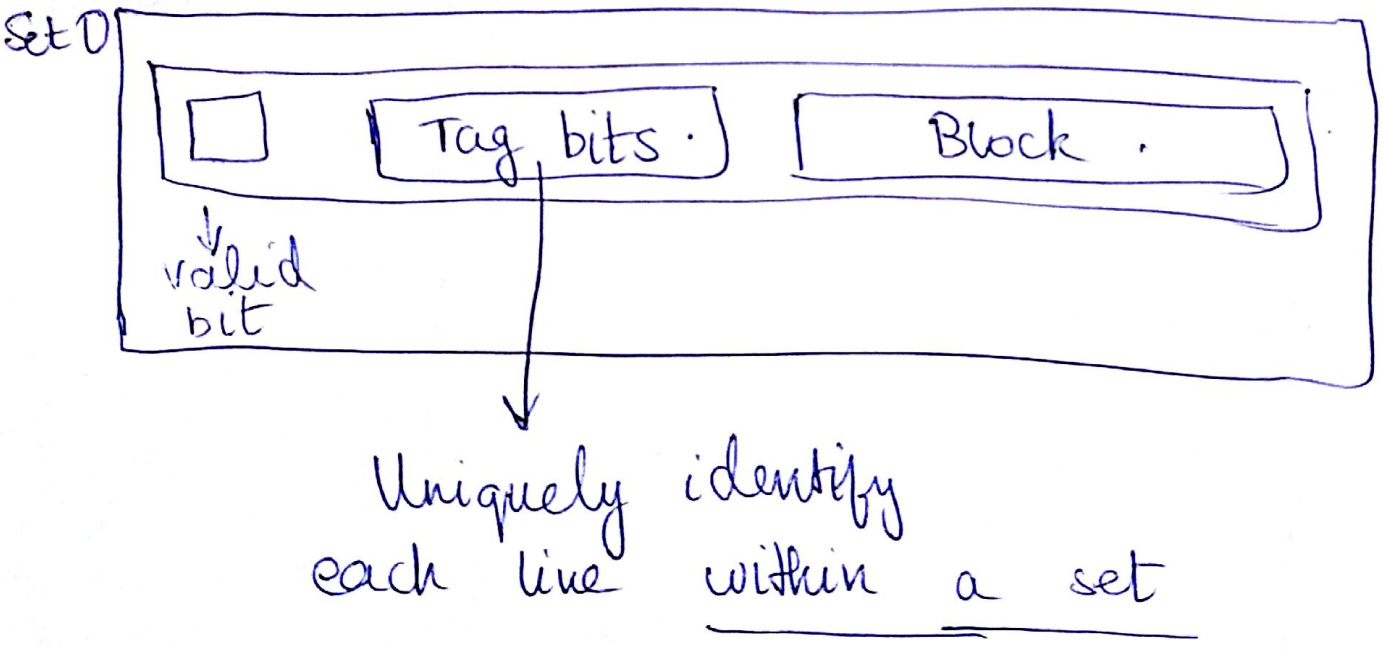




S = 3

E = 1

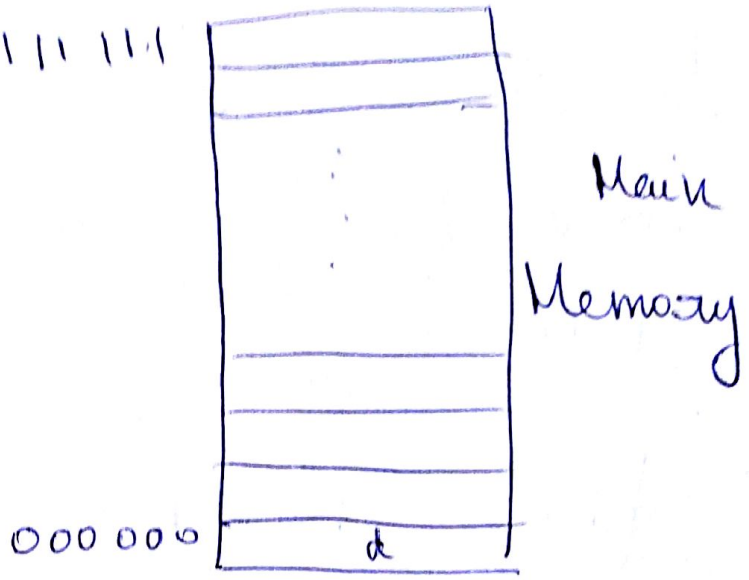
How do we identify each of these blocks ?



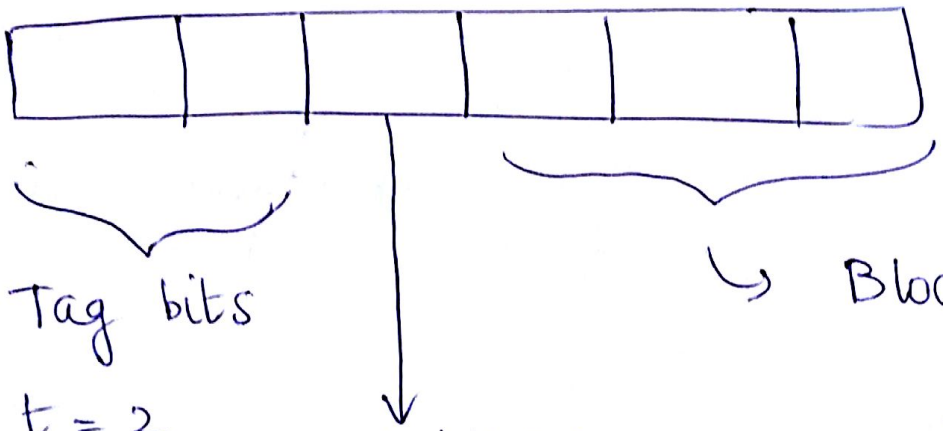
Uniquely identify each line within a set

Example.

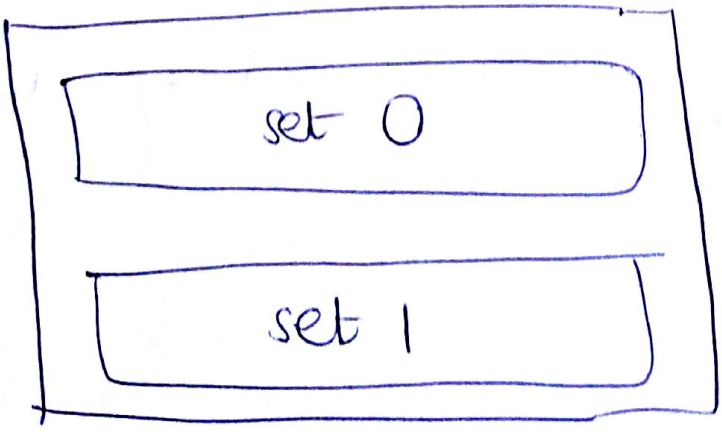
6 - bit address space



Split this address (Arbitrary)



$\#$ of Sets $= 2^s = 2$ Sets.



Cache.

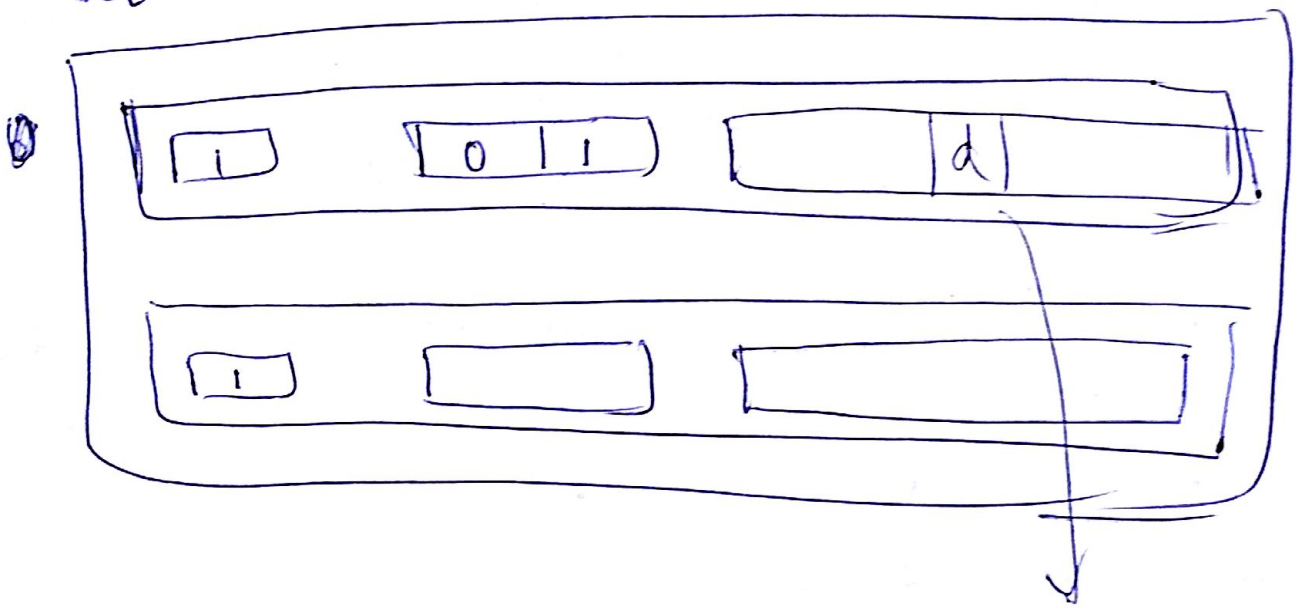
Data item at address 1 0 0 1 1 0

Which set? set 0!

Tag bits

Search for d at address 0 1 0 1 0 1

Set 0



E = 2

Somewhere in here.

set bits + tag bits

↓

they uniquely identify a line within the cache!

(tag bits identify a line uniquely within a set)

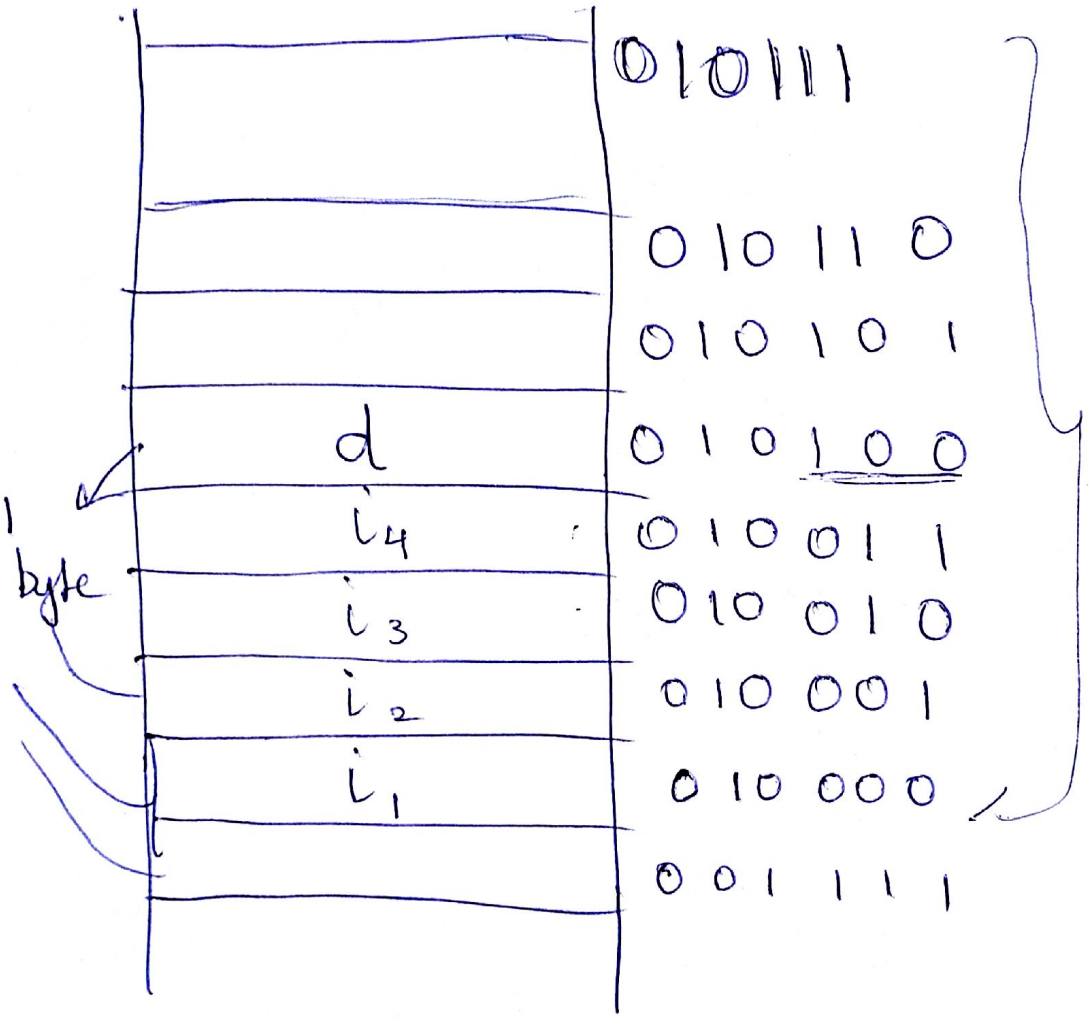
↓

may also be called INDEX bits.

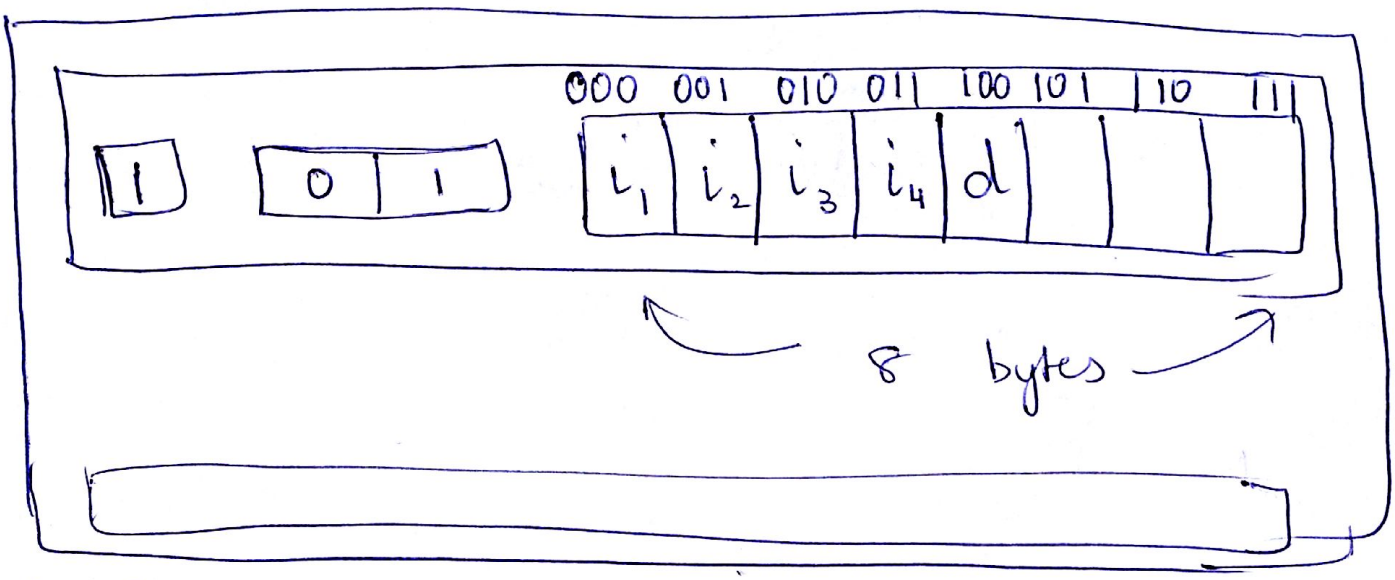
Block offset

b = 3 bits.

Size of the block $B = 2^b$ = 8 bytes.



d is at address 0 1 0 1 0 0



Set 0

Say we have an integer i at

0 1 0 0 0 0

Read 4 bytes starting at offset

0 0 0

To search for a value at a

int, char

memory location 0 1 0 1 0 0,

→ set 0

→ line with tag bits 01

→ looking at the block offset 001

What if there is no line with tag bits 01?

cache miss!

How do we handle it?

→ Read the relevant data block from main memory.
and replace an existing line!

Simple.

→ choose any line with valid bit off (=0). & replace it.

Not so simple

→ All lines have valid bits set (=1)

cache replacement policy

which line / block to replace?

Size of the cache

$= S \times E \times B$ bytes.

(excluding the tag & valid bit(s))

How to describe a cache

(S, E, B, m)

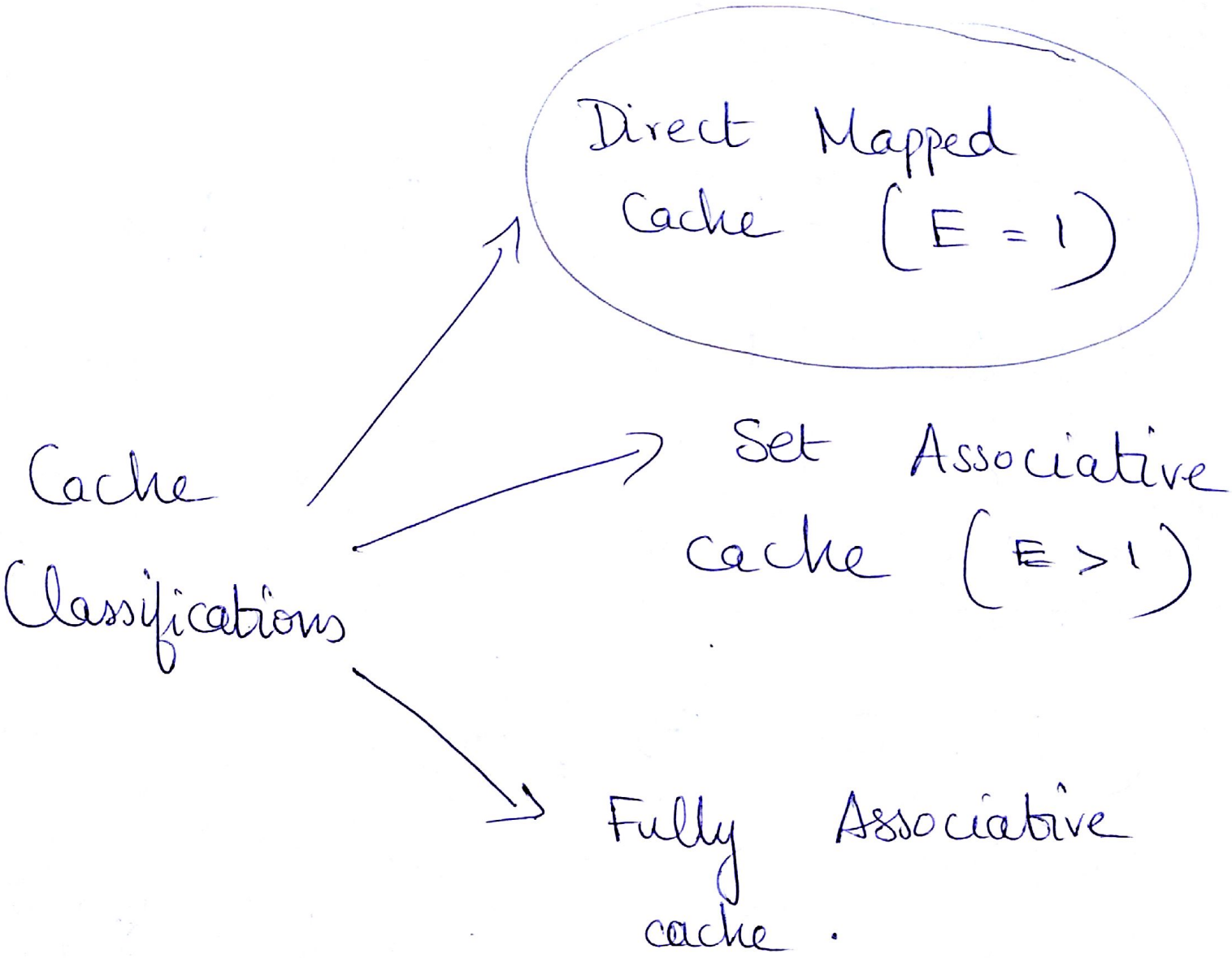
$s = \log_2(S)$

$b = \log_2(B)$

$t = m - (s+b)$

Beyond a threshold it does not make any sense to increase E.

Fig 6.28.



0000 ⟶ 1111

16 addresses