C335
Computer Structures

Memory Hierarchies (II)

Dr. Liqiang Zhang

Department of Computer and Information Sciences

Adapted from Morgan Kaufmann and others

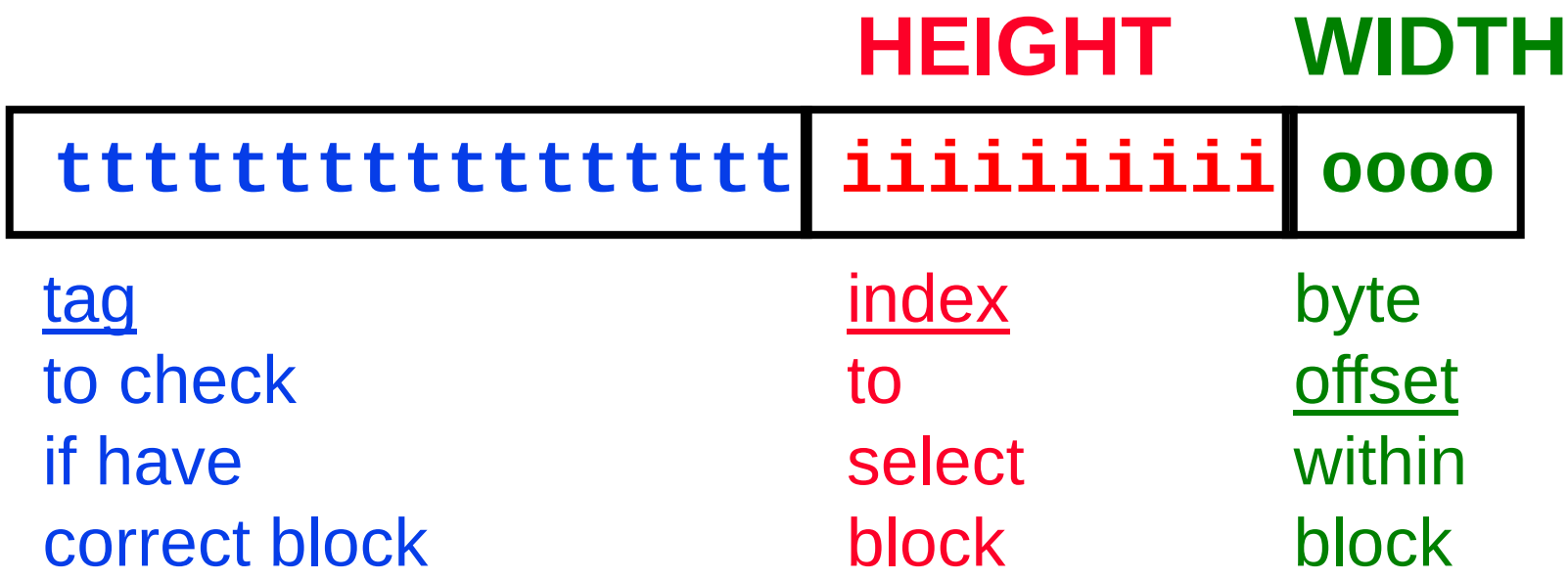
Direct-Mapped Cache Terminology

- All fields are read as unsigned integers.
- Index: specifies the cache index (which "row" of the cache we should look in)
- Offset: once we've found correct block, specifies which byte within the block we want -- I.e., which "column"
- Tag: the remaining bits after offset and index are determined; these are used to distinguish between all the memory addresses that map to the same location

Issues with Direct-Mapped

Tag	Index	Offset

- Since multiple memory addresses map to same cache index, how do we tell which one is in there?
- What if we have a block size > 1 byte?
- Answer: divide memory address into three fields



Memory address: TIO

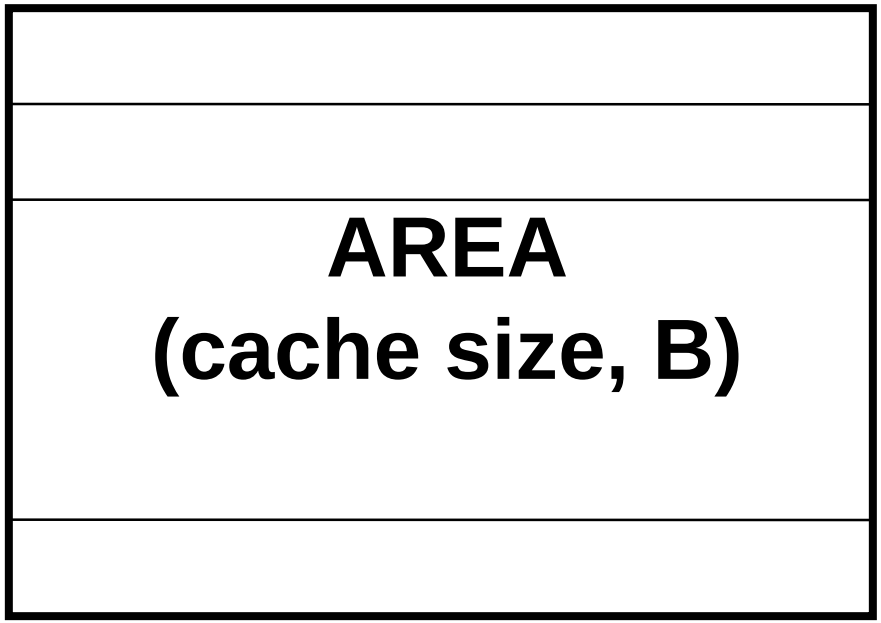
AREA (cache size, B)
= HEIGHT (# of blocks)
* WIDTH (size of one block, B/block)

$2^{(I+O)} = 2^I * 2^O$

Tag	Index	Offset

WIDTH
(size of one block, Bytes/block, 2^O)

HEIGHT
(# of blocks, 2^I)



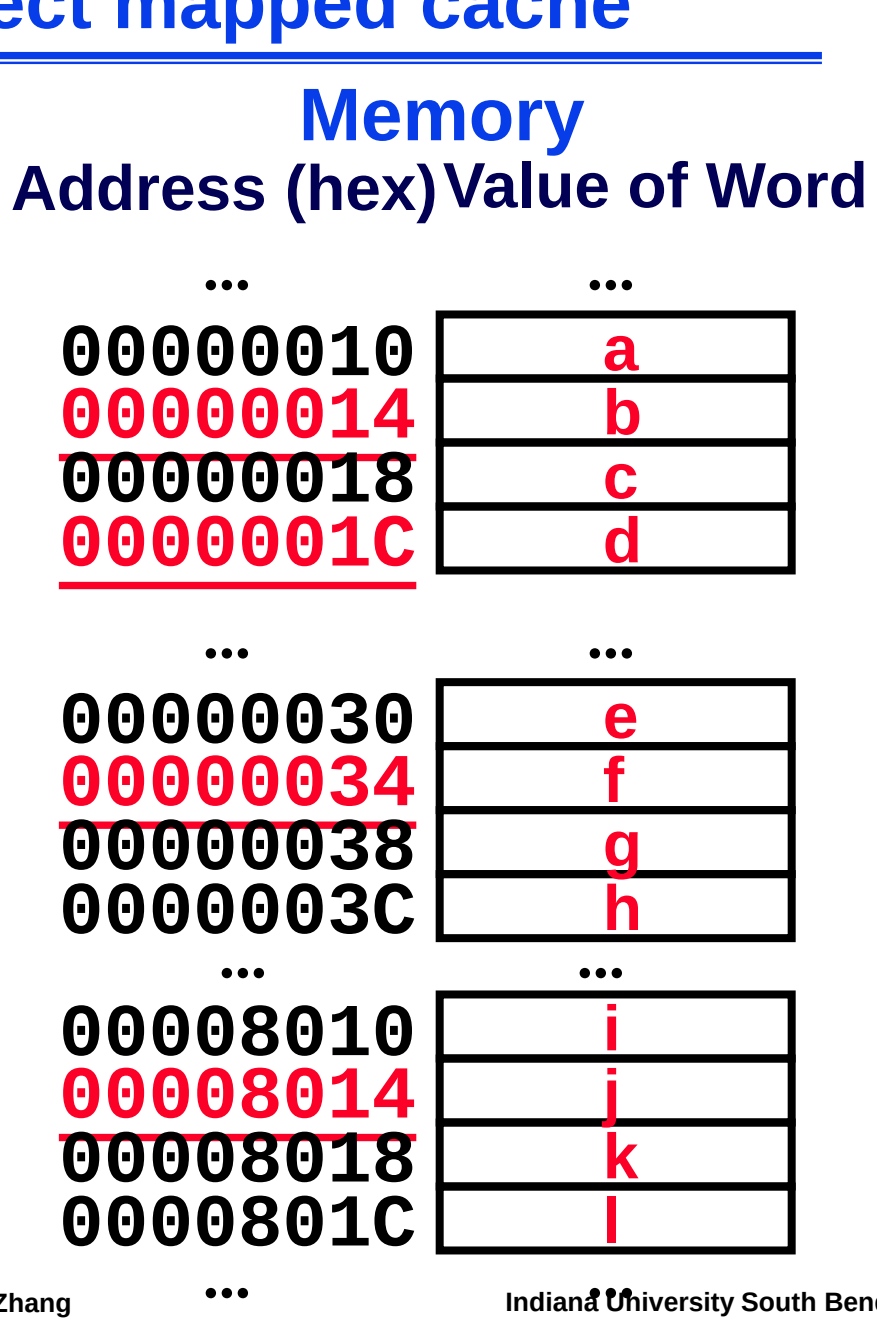
Caching Terminology

- When we try to read memory,
  3 things can happen:

1. cache hit:
   cache block is valid and contains proper address, so read desired word
2. cache miss:
   nothing in cache in appropriate block, so fetch from memory
3. cache miss, block replacement:
   wrong data is in cache at appropriate block, so discard it and fetch desired data from memory (cache always copy)

Accessing data in a direct mapped cache

- Ex.: 16KB of data, direct-mapped, 4 word blocks
- Read 4 addresses
  1. 0x00000014
  2. 0x0000001C
  3. 0x00000034
  4. 0x00008014
- Memory values on right:



Accessing data in a direct mapped cache

- 4 Addresses:
  - `0x00000014, 0x0000001C,`
    `0x00000034, 0x00008014`
- 4 Addresses divided (for convenience) into Tag, Index, Byte Offset fields

```
00000000000000000 0000000001 0100
00000000000000000 0000000001 1100
00000000000000000 0000000011 0100
00000000000000010 0000000001 0100
```

Tag Index Offset

16 KB Direct Mapped Cache, 16B blocks

- Valid bit: determines whether anything is stored in that row (when computer initially turned on, all entries invalid)

Index	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	0					
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
…				…		
1022	0					
1023	0					

1. Read 0x00000014

- 00000000000000000000 0000000001 0100

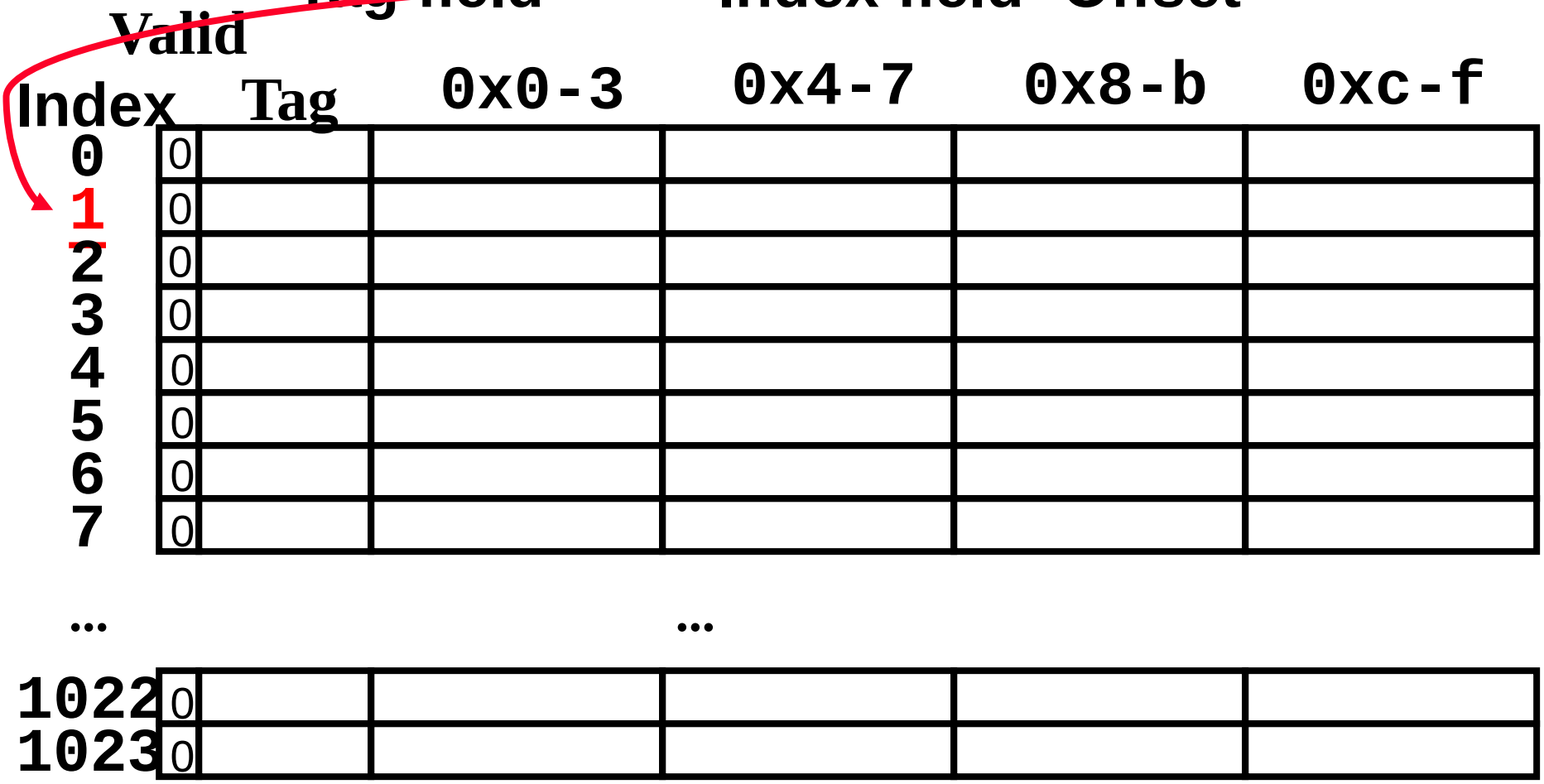
Tag field | Index field | Offset

Index	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	0					
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
…				…		
1022	0					
1023	0					

So we read block 1 (0000000001)

- 00000000000000000 0000000001 0100

Tag field | Index field | Offset



No valid data

- 00000000000000000000 0000000001 0100

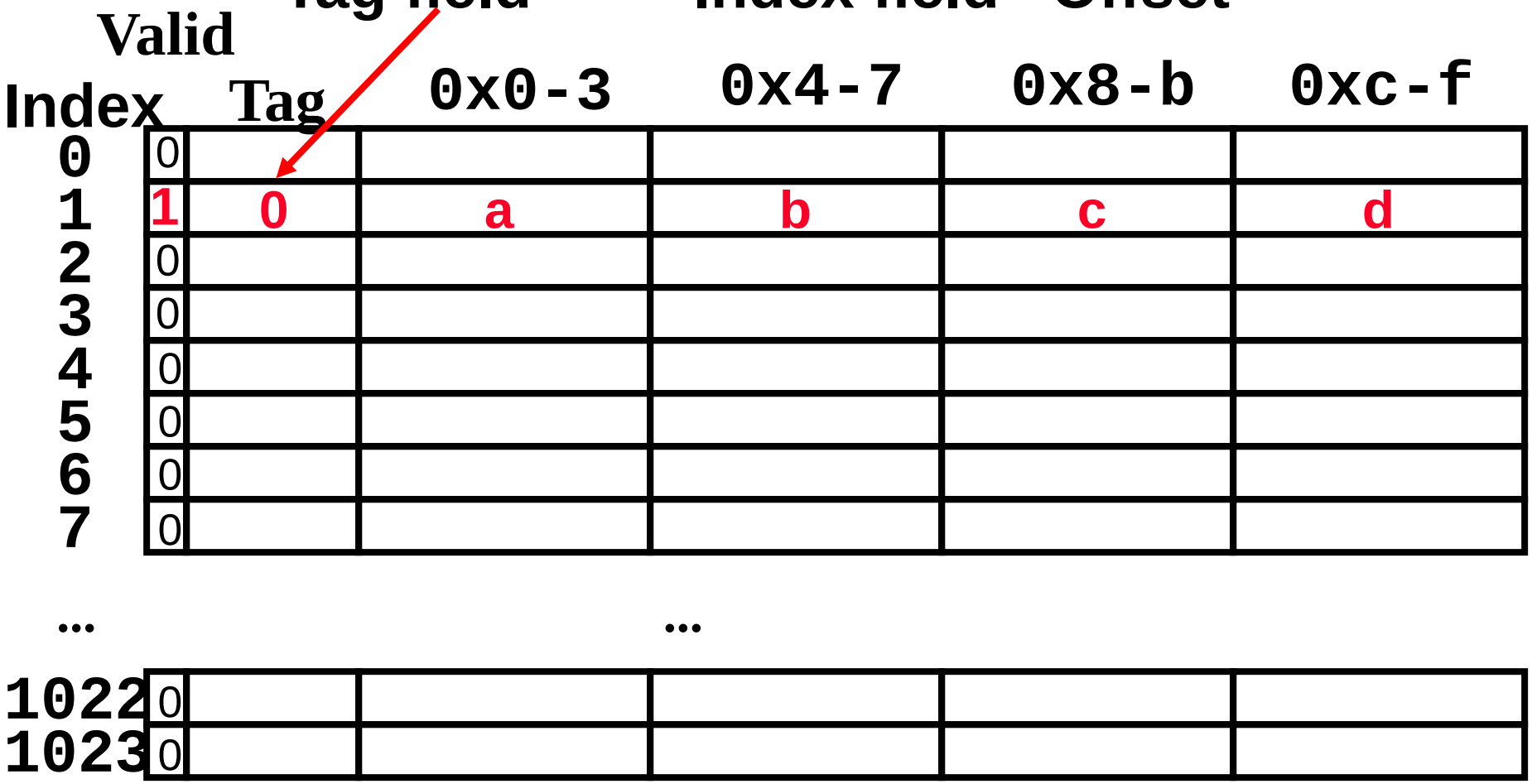
Tag field | Index field | Offset

Index	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	0					
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
…				…		
1022	0					
1023	0					

So load that data into cache, setting tag, valid

- 00000000000000000000 0000000001 0100

Tag field | Index field | Offset



Read from cache at offset, return word b

- 0000000000000000 0000000001 0100

Tag field | Index field | Offset

Index	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	0	a	b	c	d
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
…				…		
1022	0					
1023	0					

2. Read 0x0000001C = 0…00 0..001 1100

- 00000000000000000000 0000000001 1100

Tag field | Index field | Offset

Index	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	0	a	b	c	d
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
…				…		
1022	0					
1023	0					

Index is Valid

- 00000000000000000000 0000000001 1100

Tag field | Index field | Offset

Index	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	0	a	b	c	d
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
...			...			
1022	0					
1023	0					

Index valid, Tag Matches

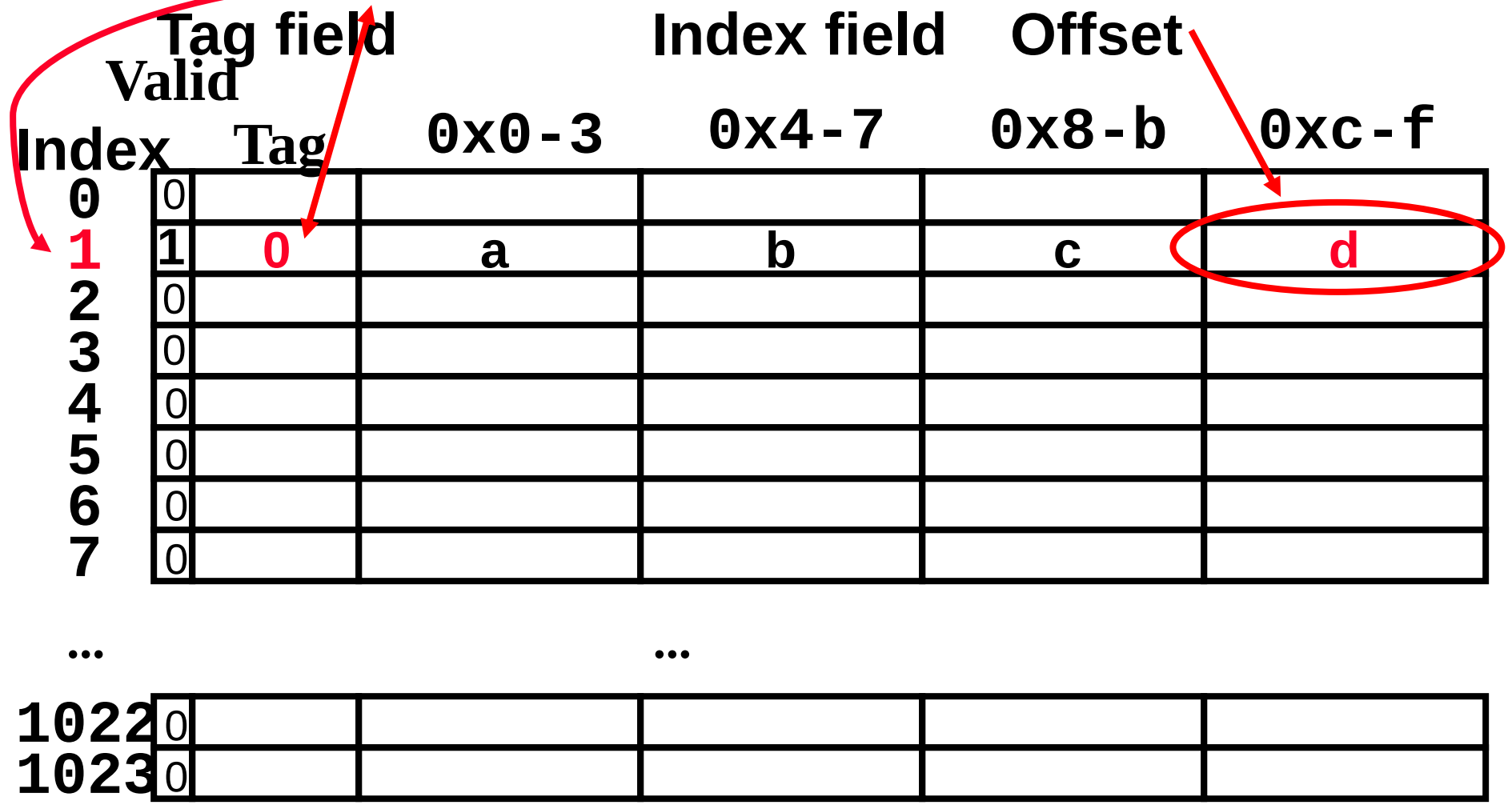
- 00000000000000000000 0000000001 1100

Tag field | Index field | Offset

Index	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	0	a	b	c	d
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
…			…			
1022	0					
1023	0					

Index Valid, Tag Matches, return d

- 00000000000000000000 0000000001 1100



3. Read 0x00000034 = 0…00 0..011 0100

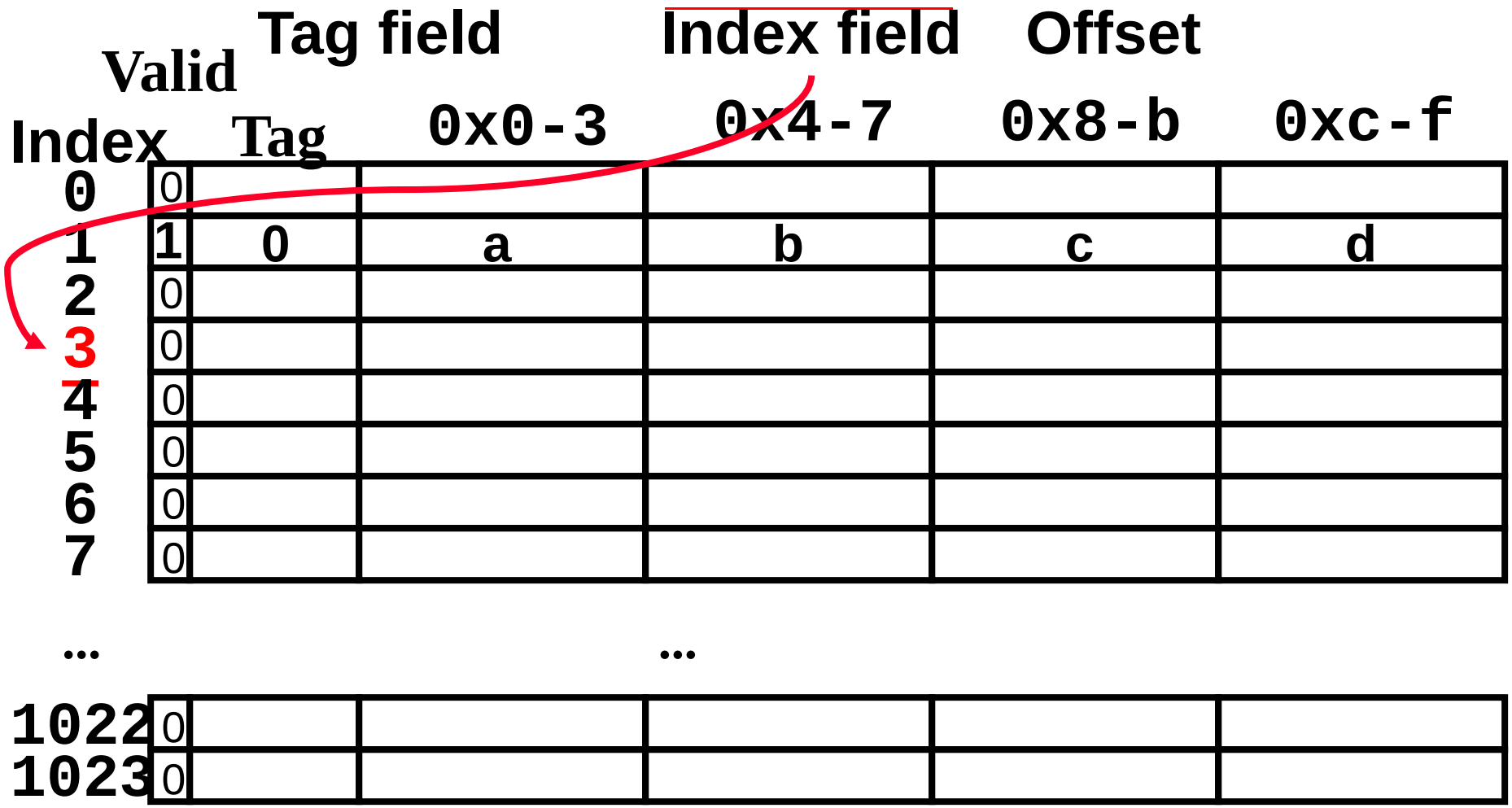
- 00000000000000000000 0000000011 0100

Tag field | Index field | Offset

Index	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	0	a	b	c	d
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
…				…		
1022	0					
1023	0					

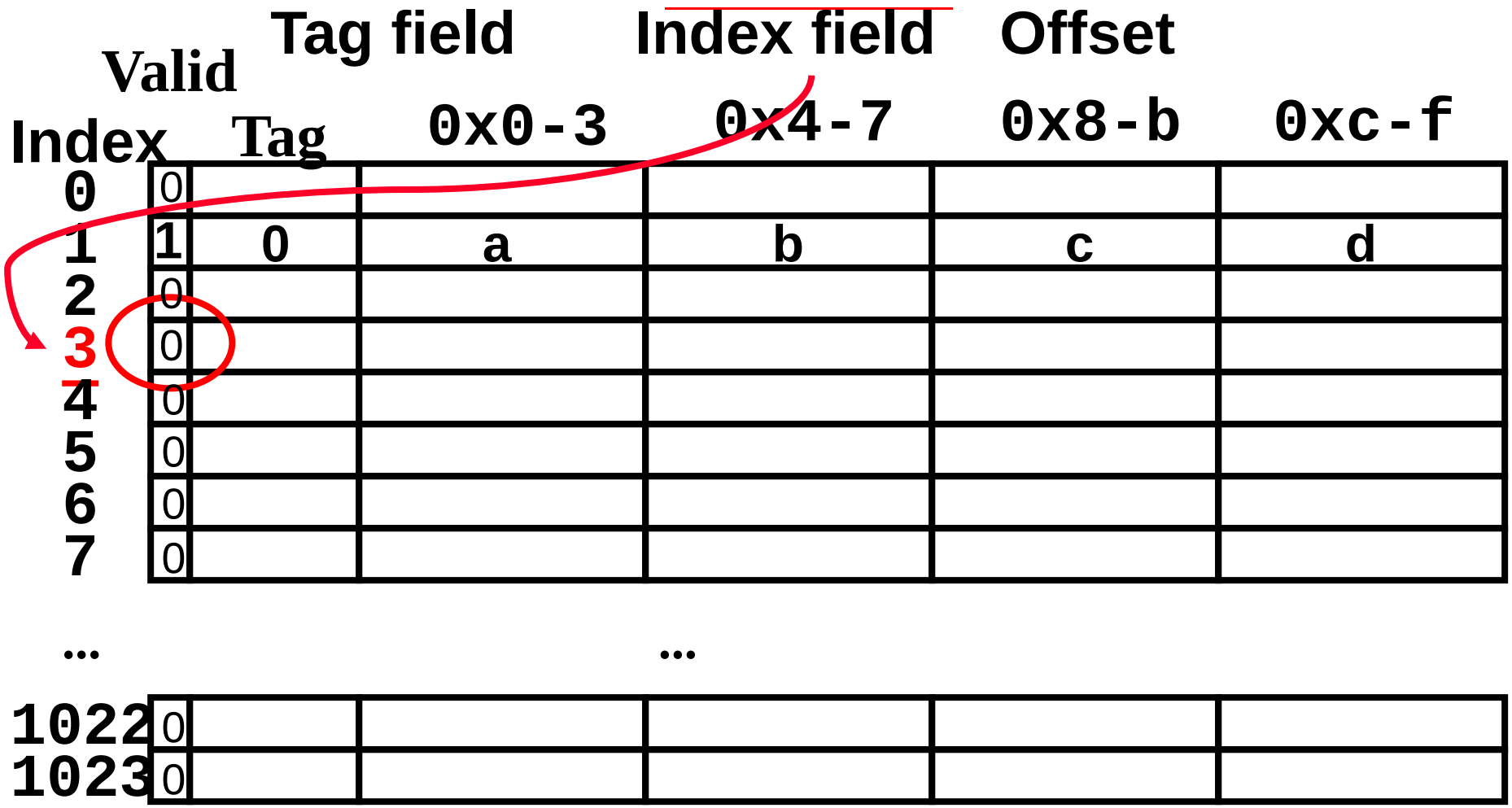
So read block 3

- 00000000000000000000 0000000011 0100



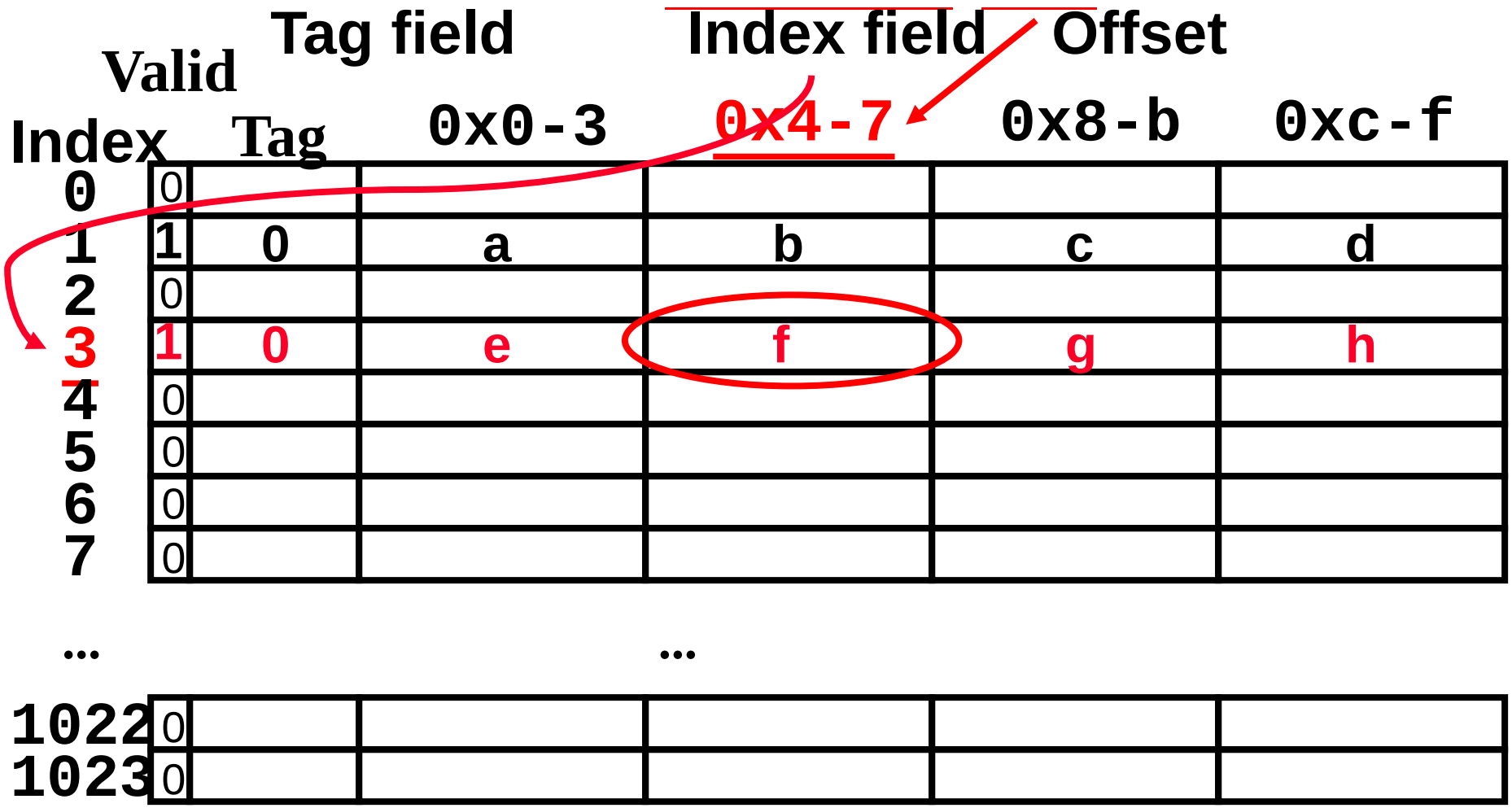
No valid data

- 00000000000000000000 0000000011 0100



Load that cache block, return word f

- 00000000000000000000 0000000011 0100



4. Read 0x00008014 = 0…10 0..001 0100

- 00000000000000010 0000000001 0100

Index	Valid	Tag	Tag field 0x0-3	0x4-7	Index field 0x8-b	Offset 0xc-f
0	0					
1	1	0	a	b	c	d
2	0					
3	1	0	e	f	g	h
4	0					
5	0					
6	0					
7	0					
…				…		
1022	0					
1023	0					

So read Cache Block 1, Data is Valid

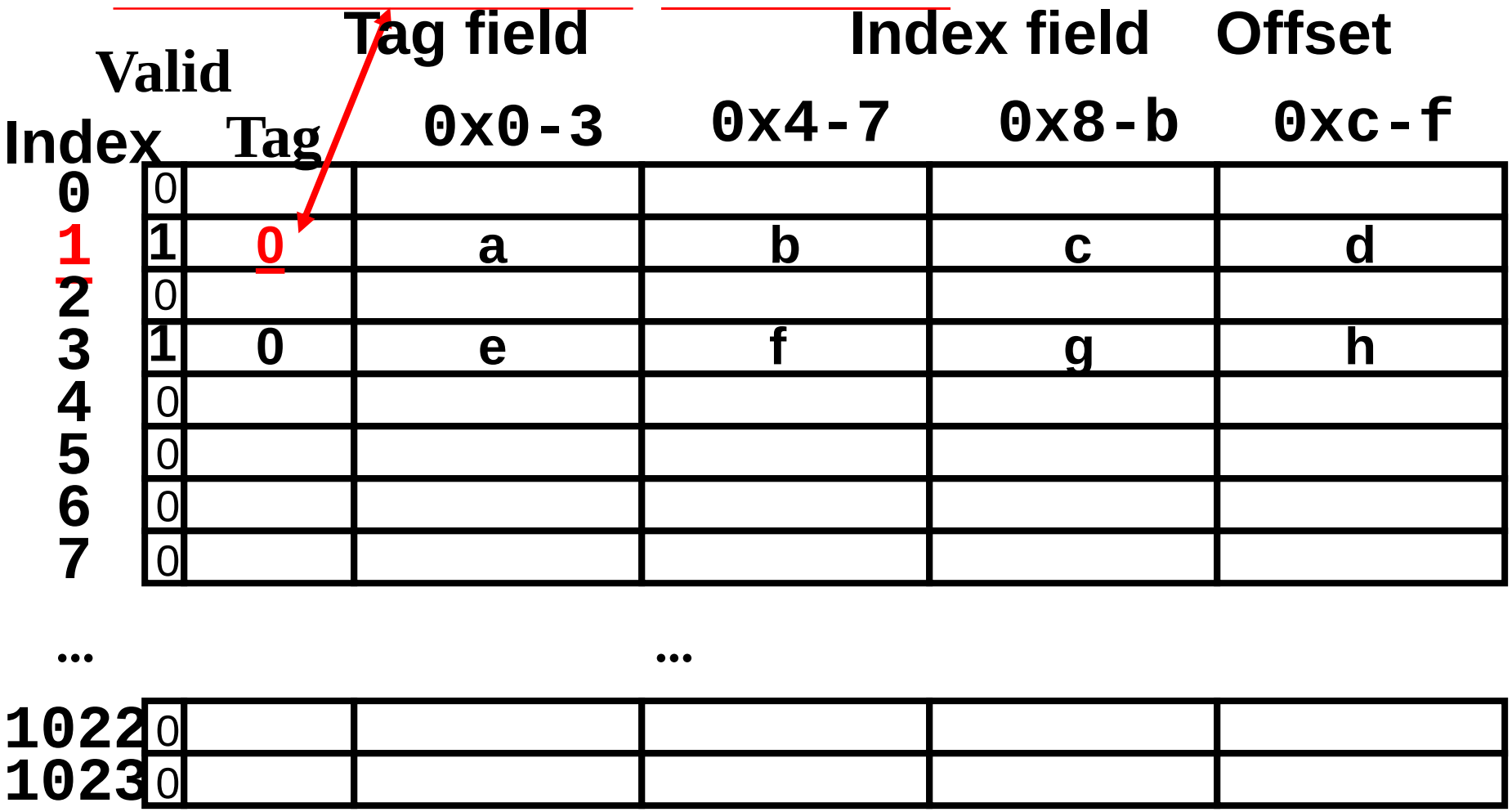
- 0000000000000010 0000000001 0100

Tag field | Index field | Offset

Index	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	0	a	b	c	d
2	0					
3	1	0	e	f	g	h
4	0					
5	0					
6	0					
7	0					
…				…		
1022	0					
1023	0					

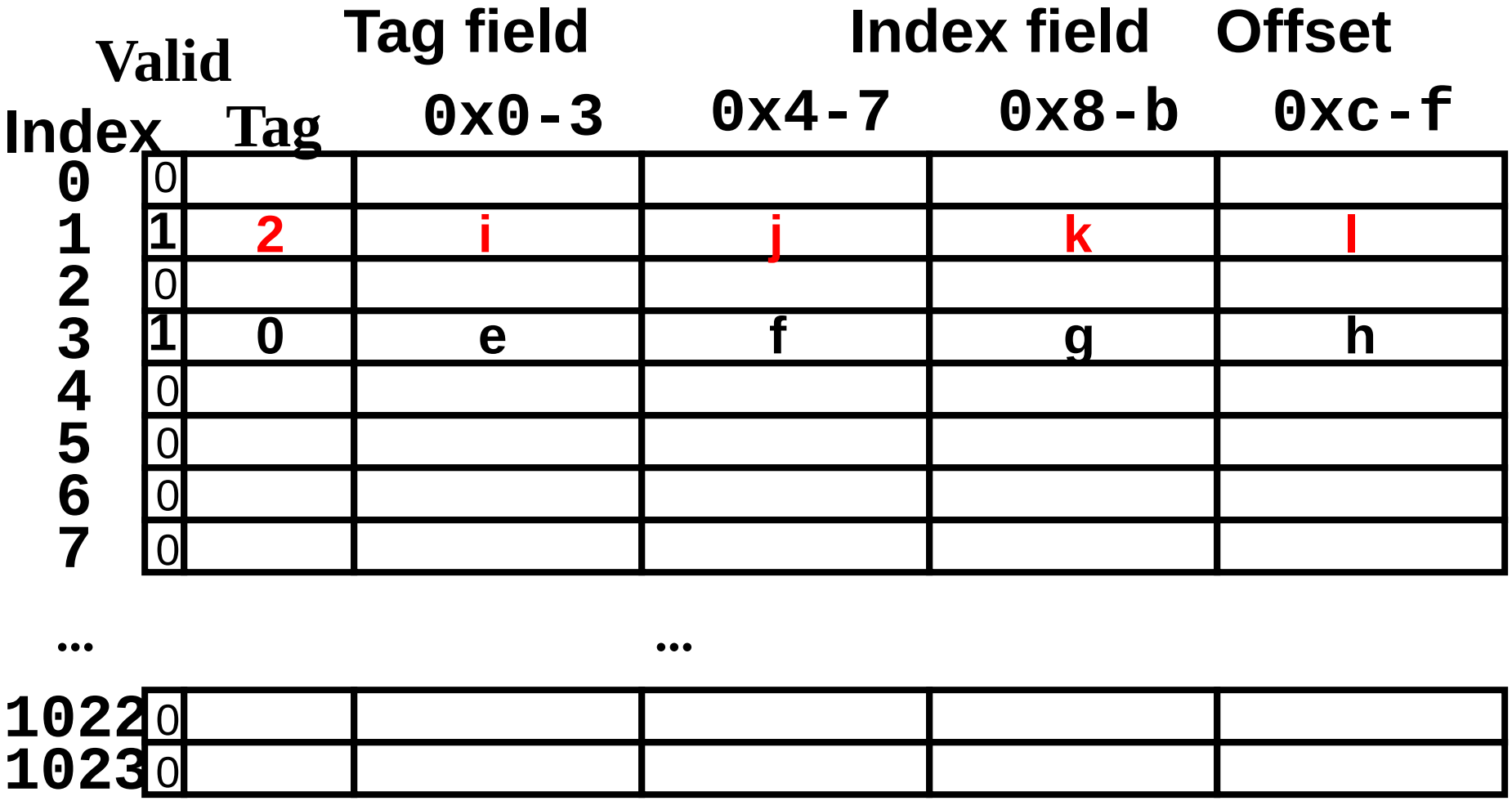
Cache Block 1 Tag does not match (0 != 2)

- 0000000000000010 0000000001 0100



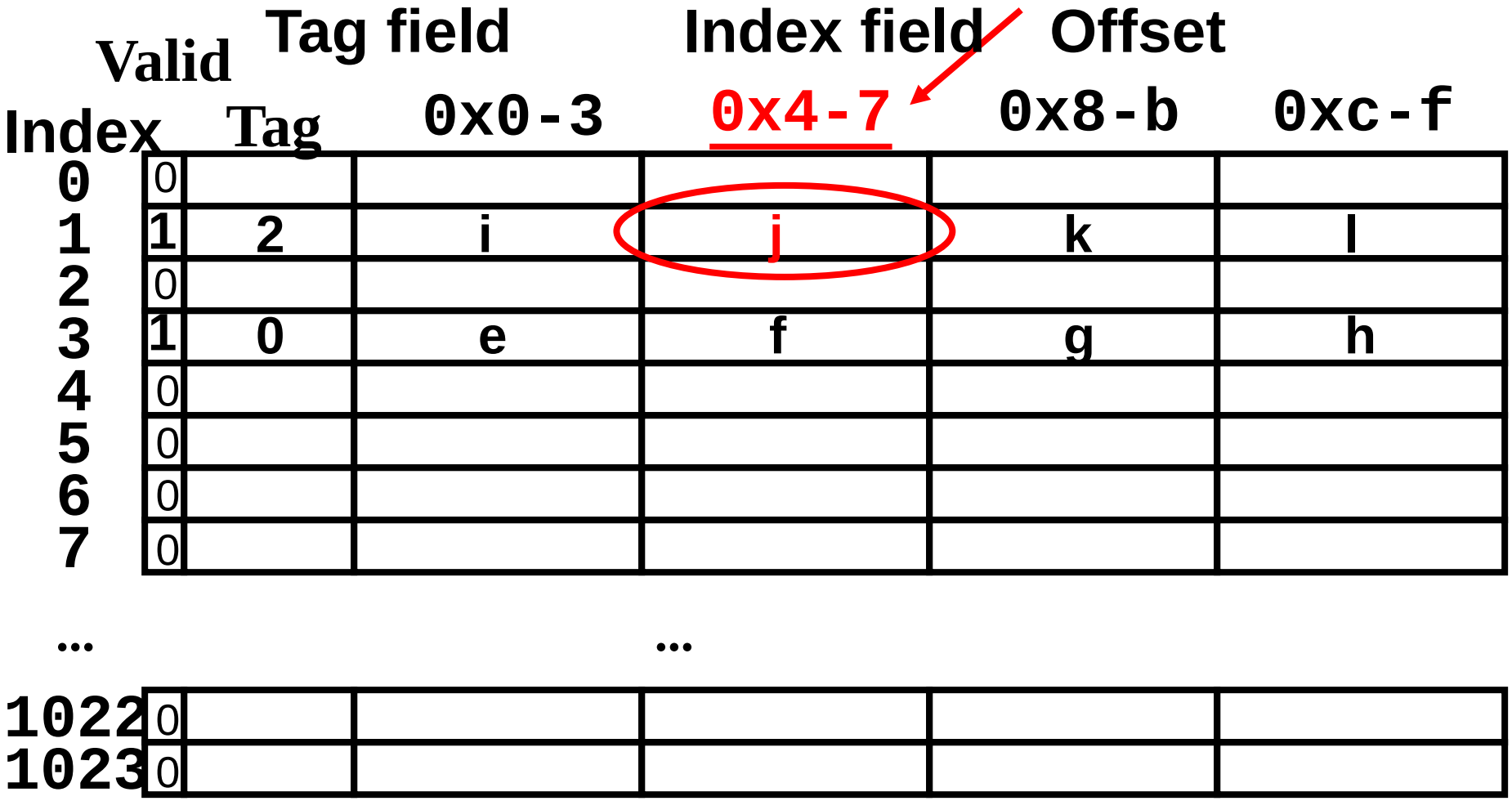
Miss, so replace block 1 with new data & tag

- 0000000000000010 0000000001 0100



And return word j

- 00000000000000010 0000000001 0100



Do an example yourself. What happens?

- Chose from: Cache Hit, Miss, Miss w. replace
  Values returned: a ,b, c, d, e, ..., k, l
- Read address 0x00000030 ?

  `0000000000000000 0000000011 0000`
- Read address 0x0000001c ?

  `0000000000000000 0000000001 1100`

Cache

Index	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	2	i	j	k	l
2	0					
3	1	0	e	f	g	h
4	0					
5	0					
6	0					
7	0					
...			...			

Answers

- 0x00000030 a hit

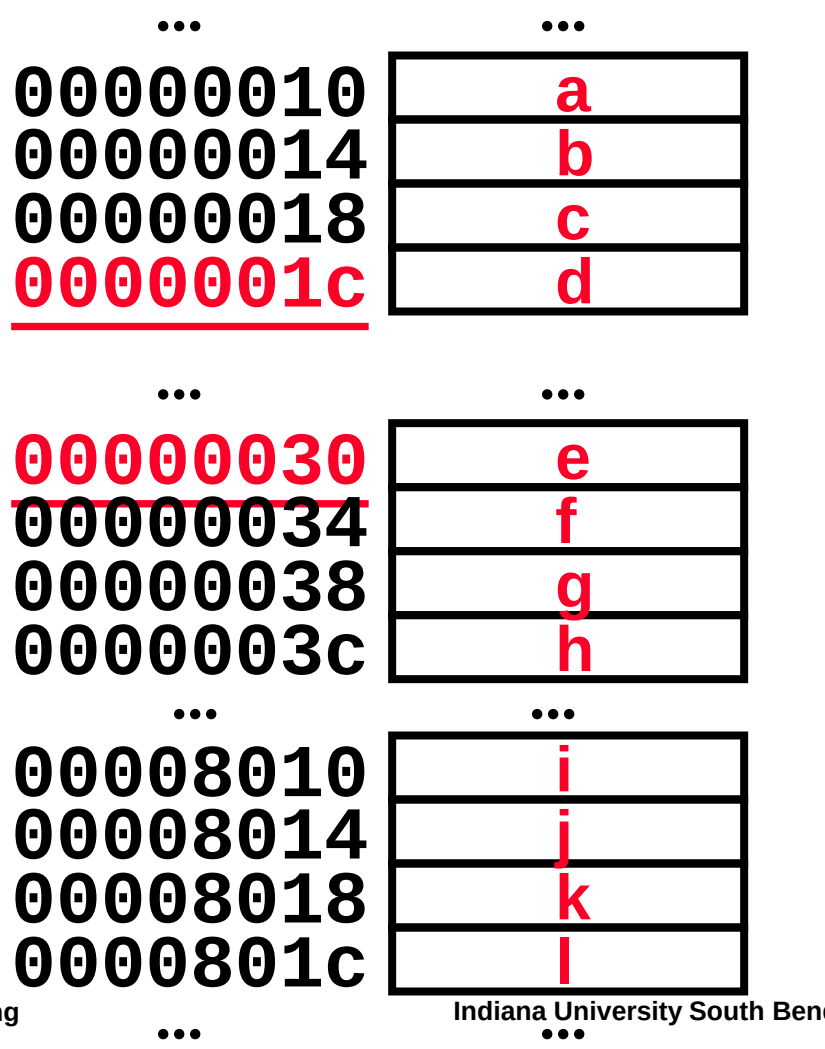
  Index = 3, Tag matches, Offset = 0, value = e

- 0x0000001c a miss

  Index = 1, Tag mismatch, so replace from memory, Offset = 0xc, value = d

- Since reads, values must = memory values whether or not cached:
  - 0x00000030 = e
  - 0x0000001c = d

Memory



Another Example: (1/2)

Here is a series of address references given as word addresses: 2, 3, 6, 10, 7, 12, 2, 18, 11, and 3. Assuming a direct-mapped cache with 8 two-word blocks (total size 16 words) that is initially empty, label each reference in the list as a hit or a miss and show the final contents of the cache. (note: the basic unit is word!)

Tag	Index	Data (two-word block)
	000	
	001	
	010	
	011	
	100	
	101	
	110	
	111	

2 (00000010), 3(00000011), 6(00000110), 10(00001010), 7(00000111), 12(00001100), 2(00000010), 18(00010010), 11(00001011), 3(00000011)

Another Example: (2/2)

Tag	Index	Data (two-word block)
	000	
0000 → 0001	001	[2,3] → [18,19]
	010	
0000	011	[6, 7]
	100	
0000	101	[10, 11]
0000	110	[12, 13]
	111	

2 (00000010) → miss, 3(00000011) → hit, 6(00000110) →miss,
10(00001010) → miss, 7(00000111) → hit, 12(00001100) →miss,
2(00000010) → hit, 18(00010010) → miss, 11(00001011) →hit,
3(00000011) → ?

Note: we did not present the valid bit in the table.

And in Conclusion…

- Direct-Mapped Cache Terminology
  - Divide memory address into three field
    - Tag, Index, and Offset
  - Each cache block contains Index, Tag, and Valid bit
  - To check if a memory unit has been “cached” or not, using the Index field to locate a cache block (row), if Valid bit is 1, compare its Tag flied with the one contained in the cache block (row).