inst.eecs.berkeley.edu/~cs61c

UC Berkeley CS61C : Machine Structures

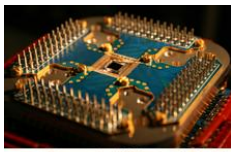
Lecture 32 – Caches II

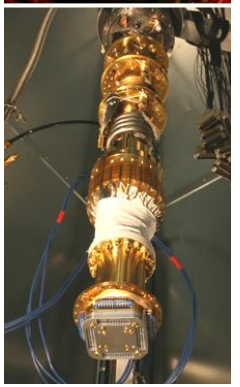
2007-04-09

Lecturer SOE Dan Garcia

www.cs.berkeley.edu/~ddgarcia

Experts weigh in on Quantum CPU ⇒





Most "profoundly skeptical" of the demo. D-Wave has provided almost no details of system. Scott Aaronson (Cal PhD): "it's as useful for solving problems as a roast-beef sandwich". Prof Vazirani: "they have misleaded the public by calling it a 'practical quantum computer'; no speedup over classical computers". D-Wave: It's a prototype!

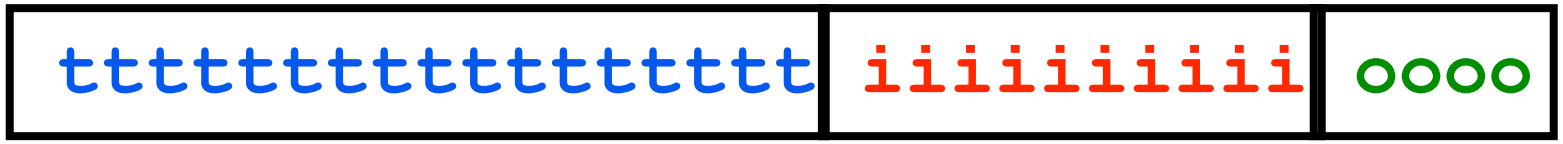
technologyreview.com/Infotech/18495/

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

HEIGHT WIDTH



tag to check if have correct block

index to select block

byte offset within block

Direct-Mapped Cache Terminology

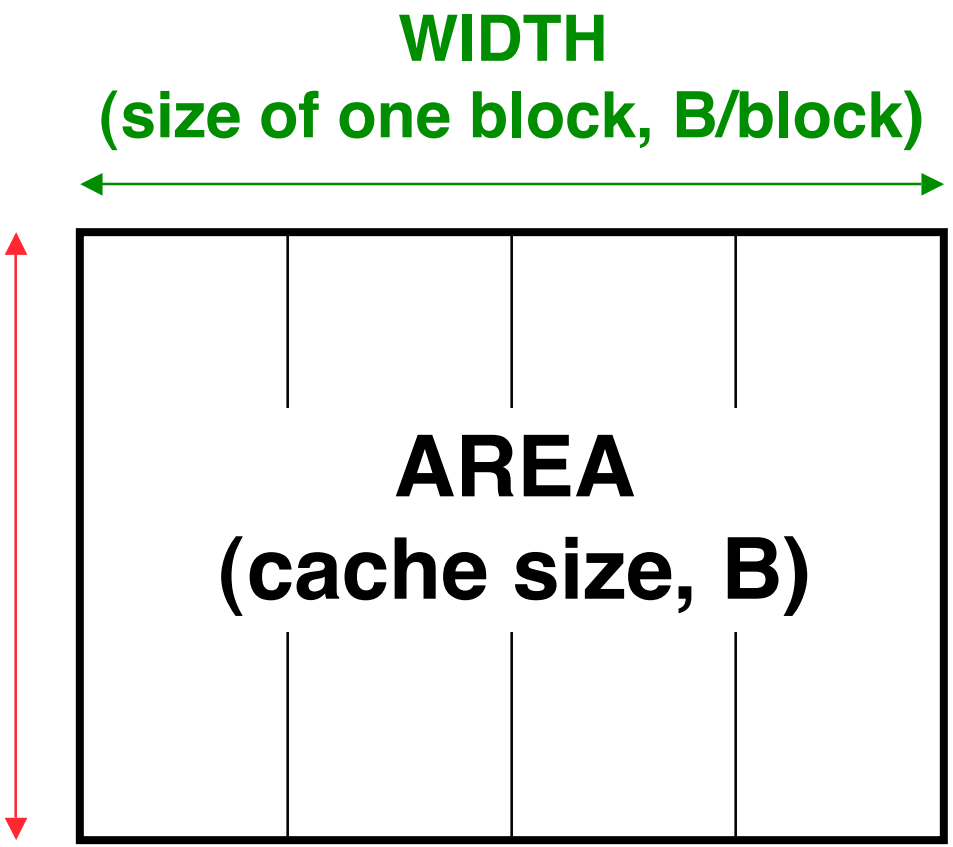
- All fields are read as unsigned integers.
- Index: specifies the cache index (which "row" of the cache we should look in)
- Offset: once we've found correct block, specifies which byte within the block we want -- I.e., which "column"
- Tag: the remaining bits after offset and index are determined;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

TIO Dan's great cache mnemonic

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

$$2^{(H+W)} = 2^H * 2^W$$

Tag	Index	Offset



Caching Terminology

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

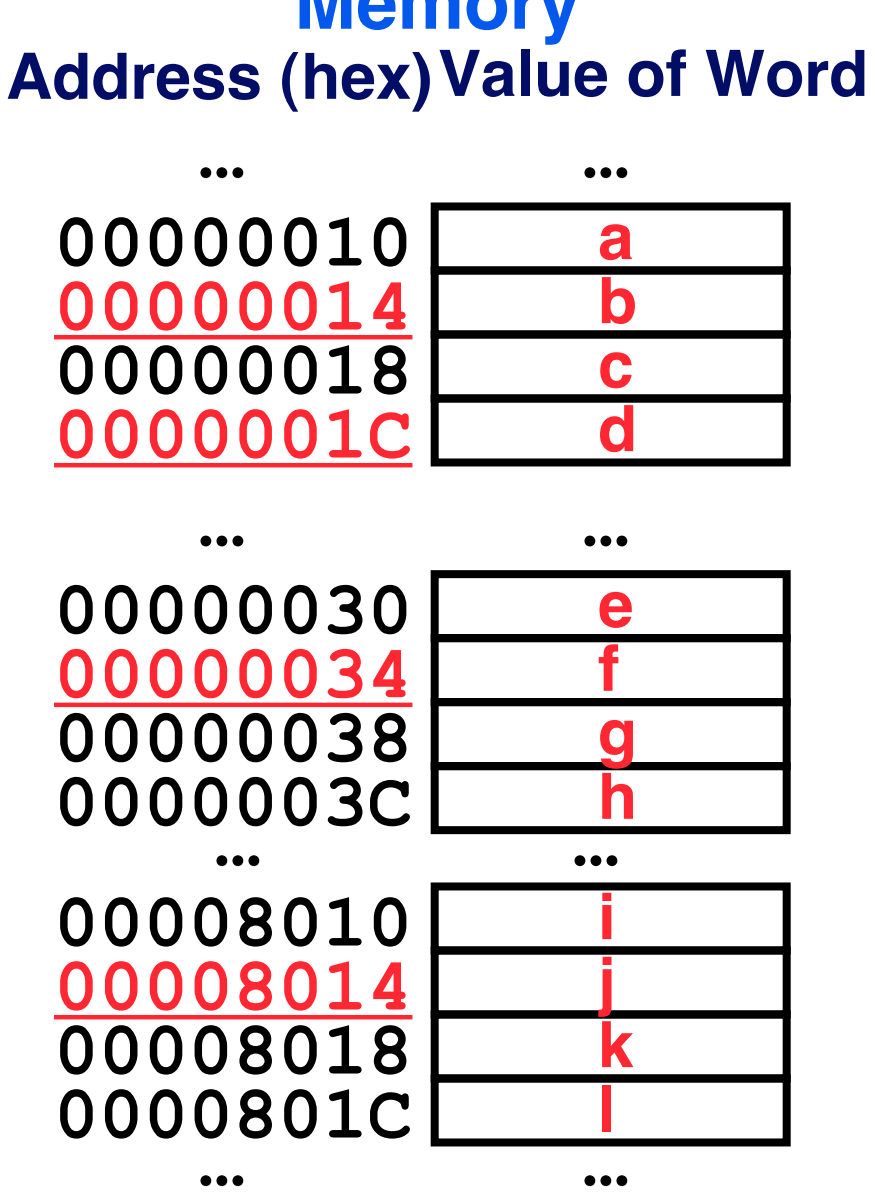
1. cache hit:
   cache block is valid and contains proper address, so read desired word

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

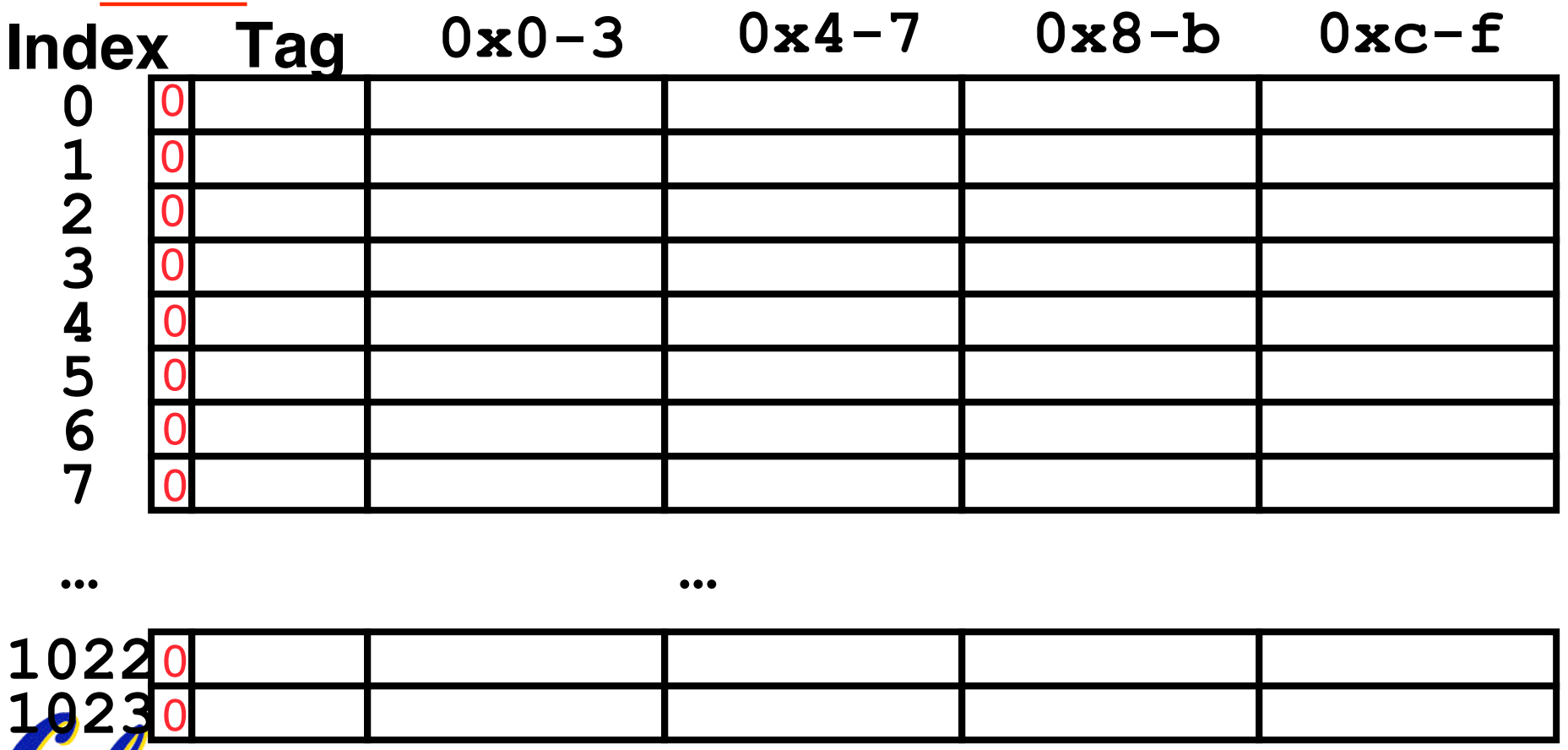
- 4 Addresses:
  - `0x00000014, 0x0000001C, 0x00000034, 0x00008014`
- 4 Addresses divided (for convenience) into Tag, Index, Byte Offset fields

```
00000000000000000000 0000000001 0100
00000000000000000000 0000000001 1100
00000000000000000000 0000000011 0100
00000000000000000010 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)



1. Read 0x00000014

- 00000000000000000000 0000000001 0100

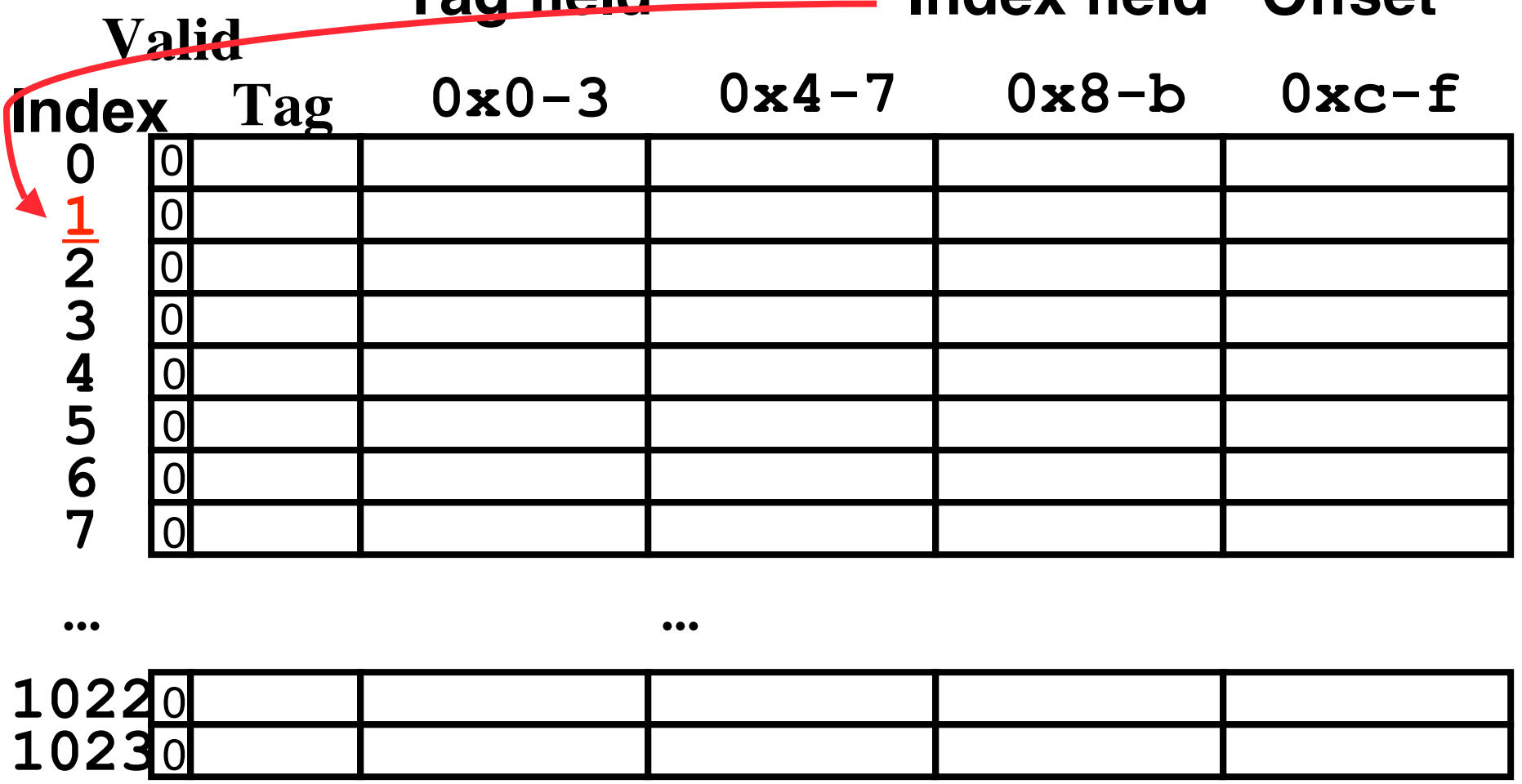
Tag field | Index field | Offset

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)

- 00000000000000000000 0000000001 0100

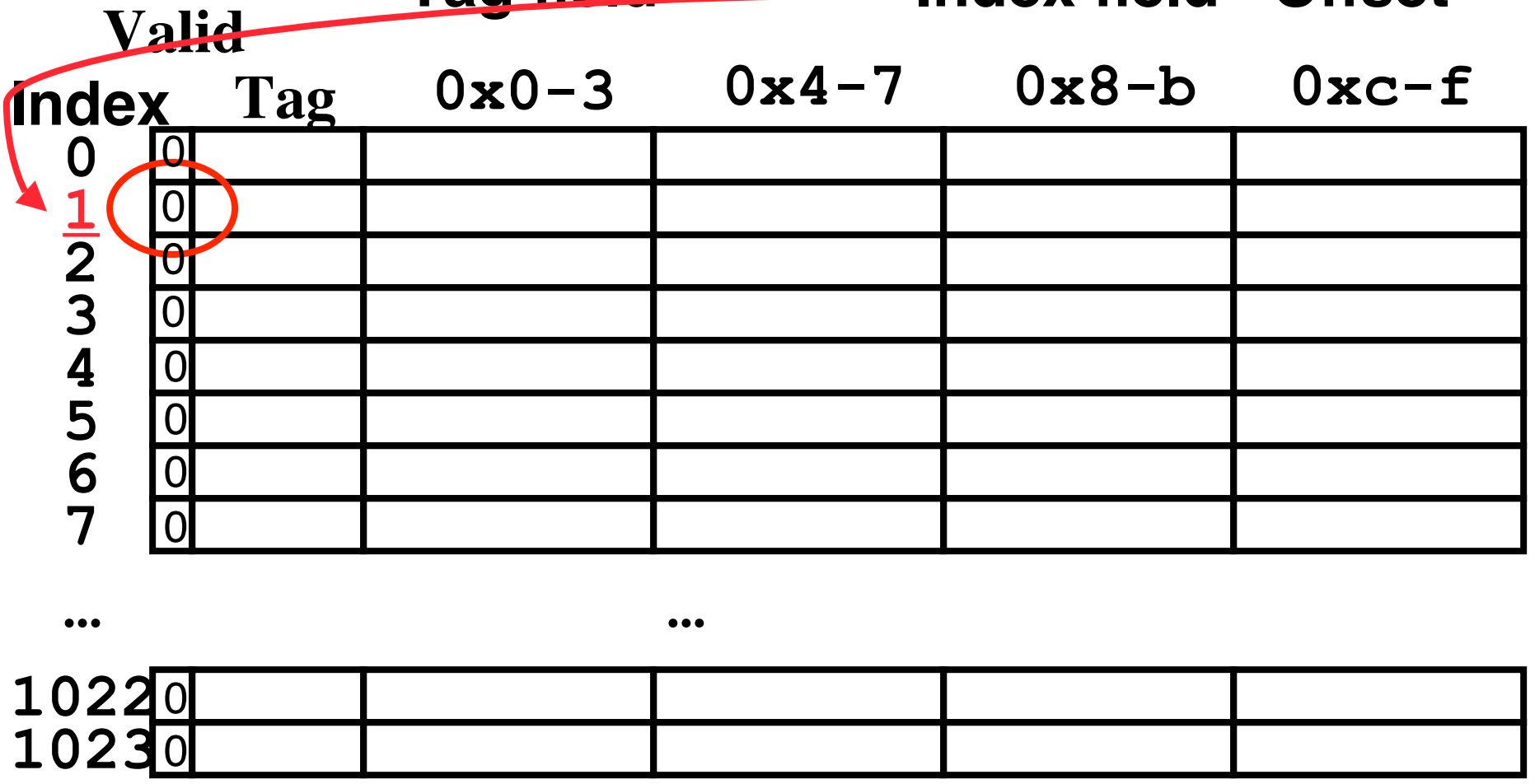
Tag field | Index field | Offset



No valid data

- 00000000000000000000 0000000001 0100

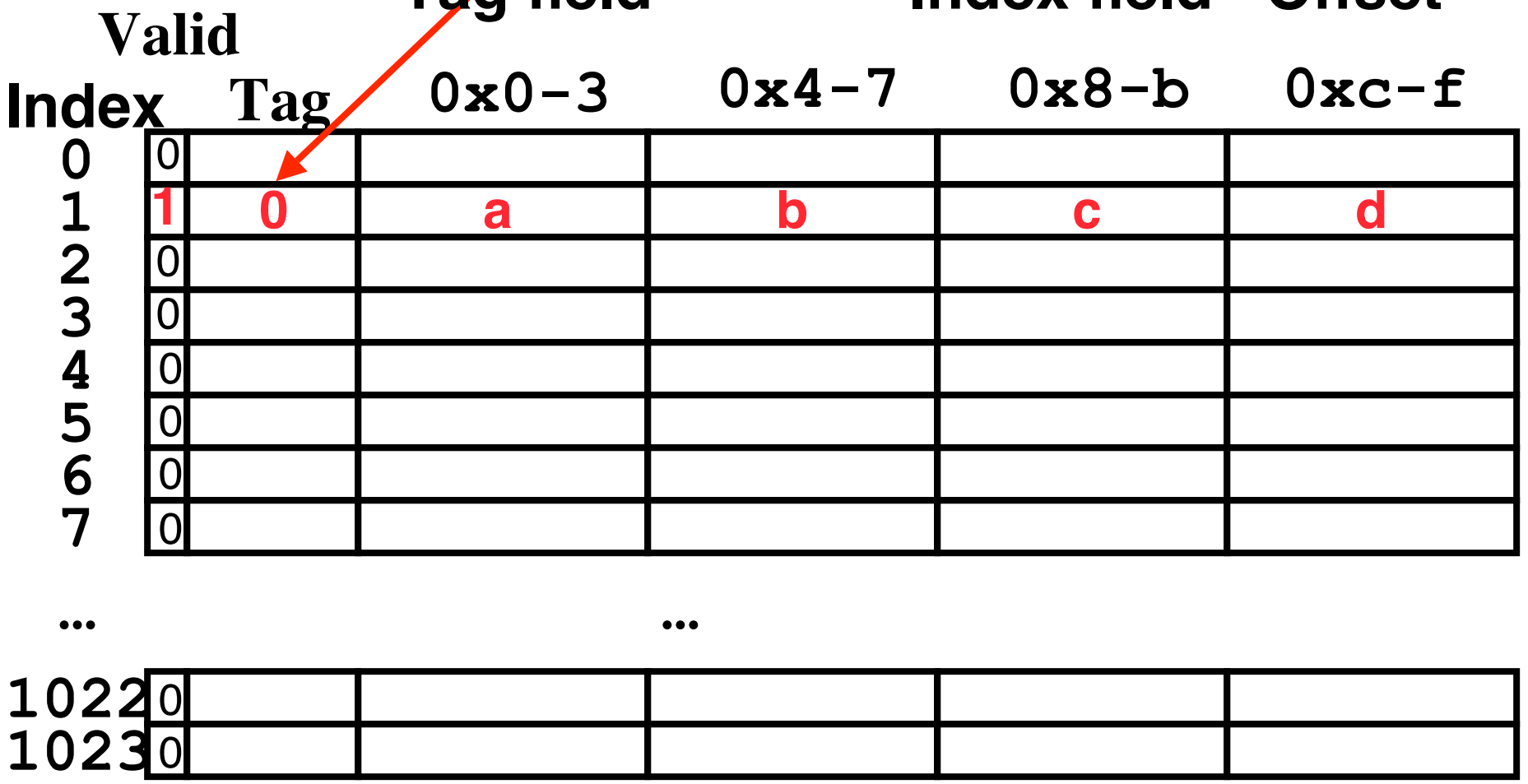
Tag field | Index field | Offset



So load that data into cache, setting tag, valid

- 00000000000000000000 0000000001 0100

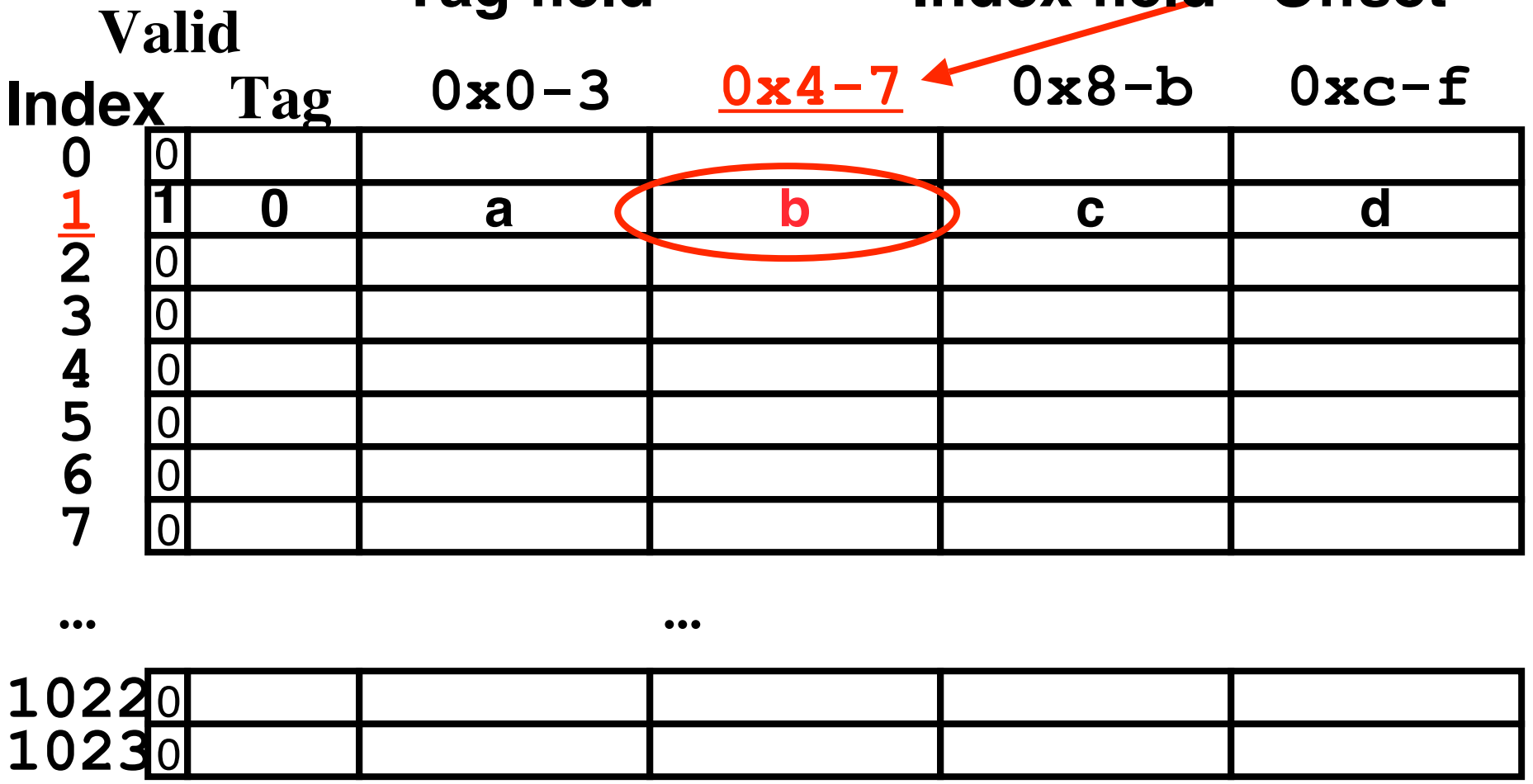
Tag field | Index field | Offset



Read from cache at offset, return word b

- 00000000000000000000 0000000001 0100

Tag field | Index field | Offset



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

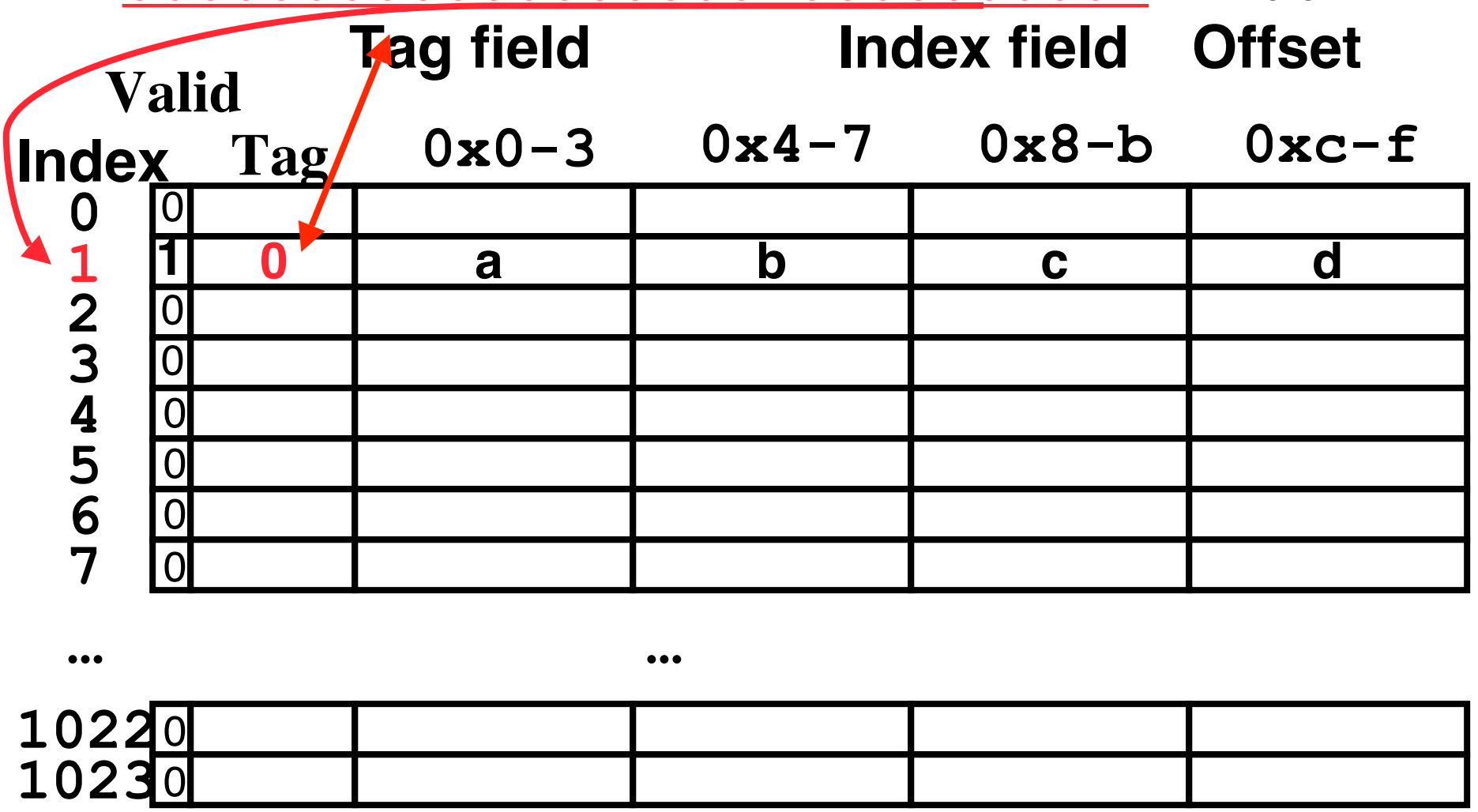
Tag field | Index field | Offset

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

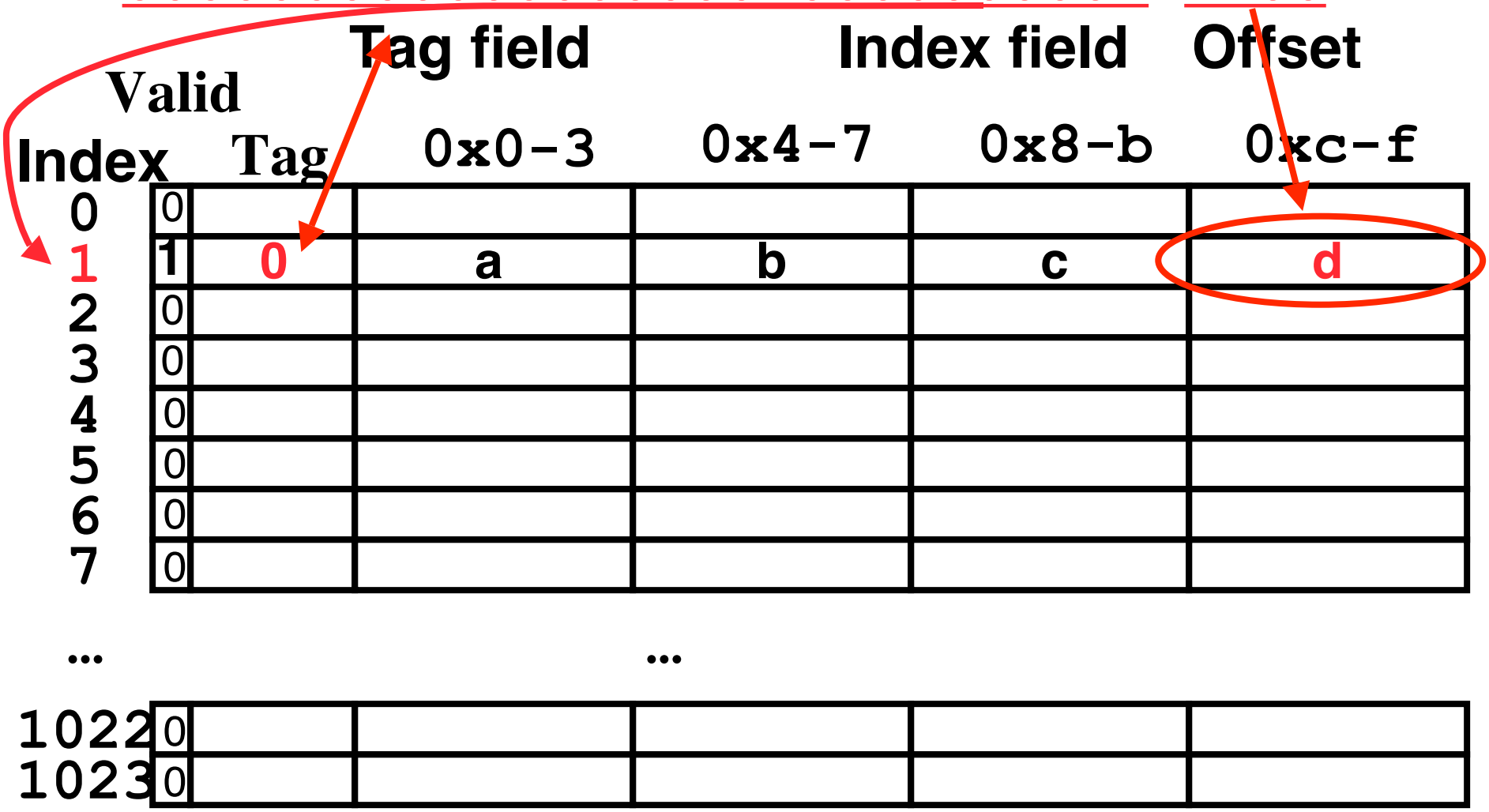
- 00000000000000000000 0000000001 1100

Tag field | Index field | Offset



Index Valid, Tag Matches, return d

- 00000000000000000000 0000000001 1100



3. Read 0x00000034 = 0…00 0..011 0100

- 00000000000000000000 0000000011 0100

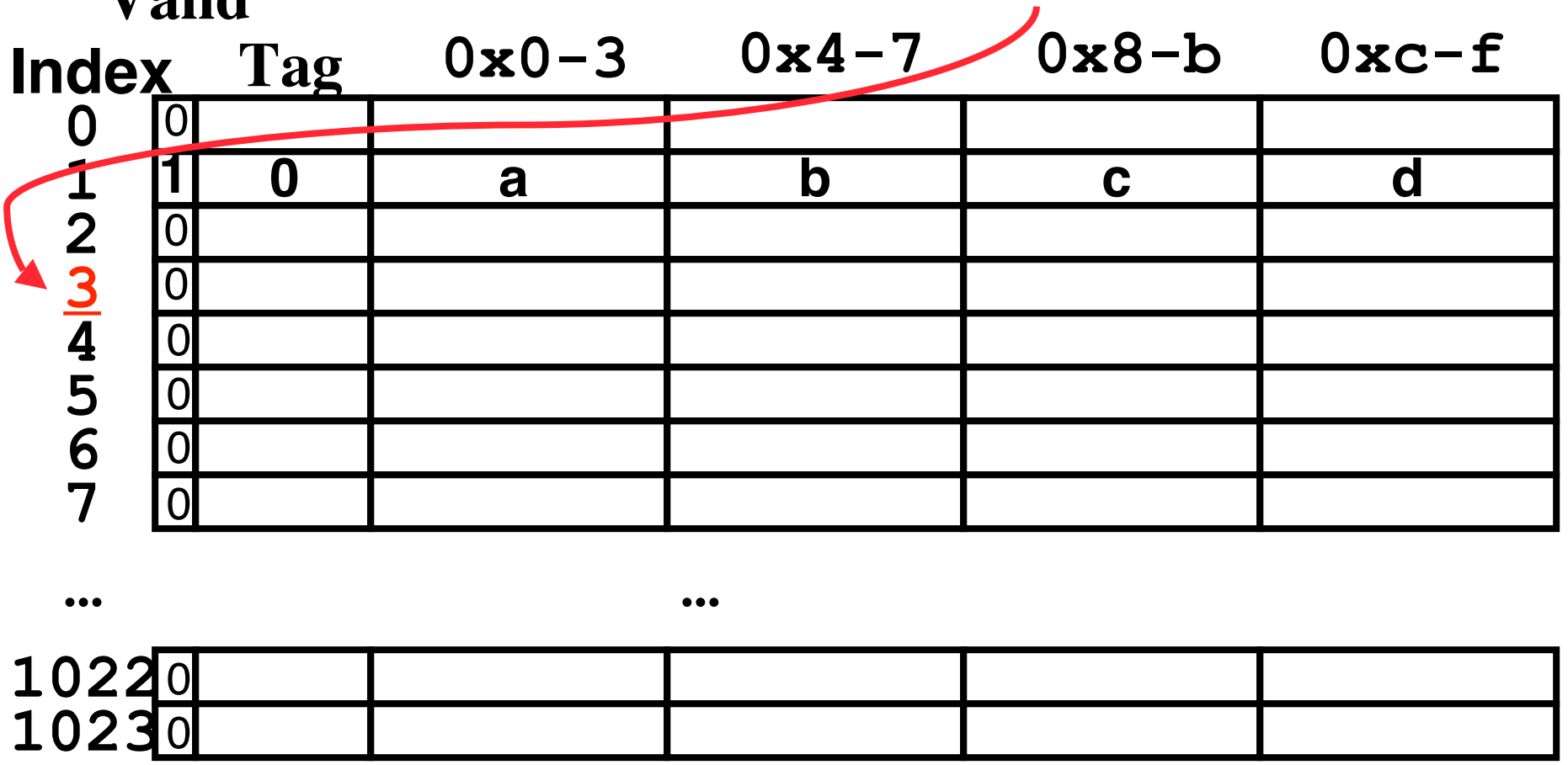
Tag field | Index field | Offset

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

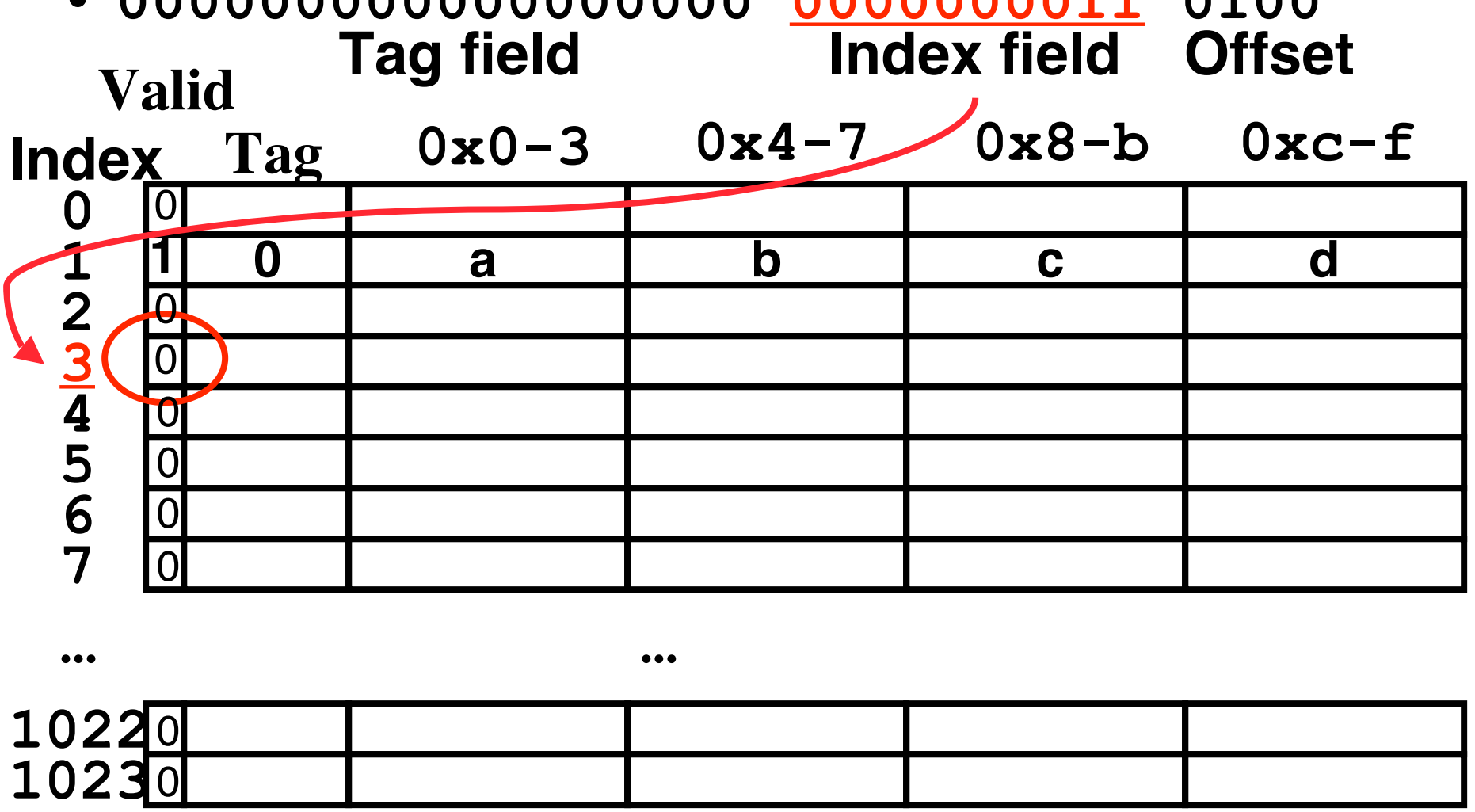
Tag field | Index field | Offset



No valid data

- 00000000000000000000 0000000011 0100

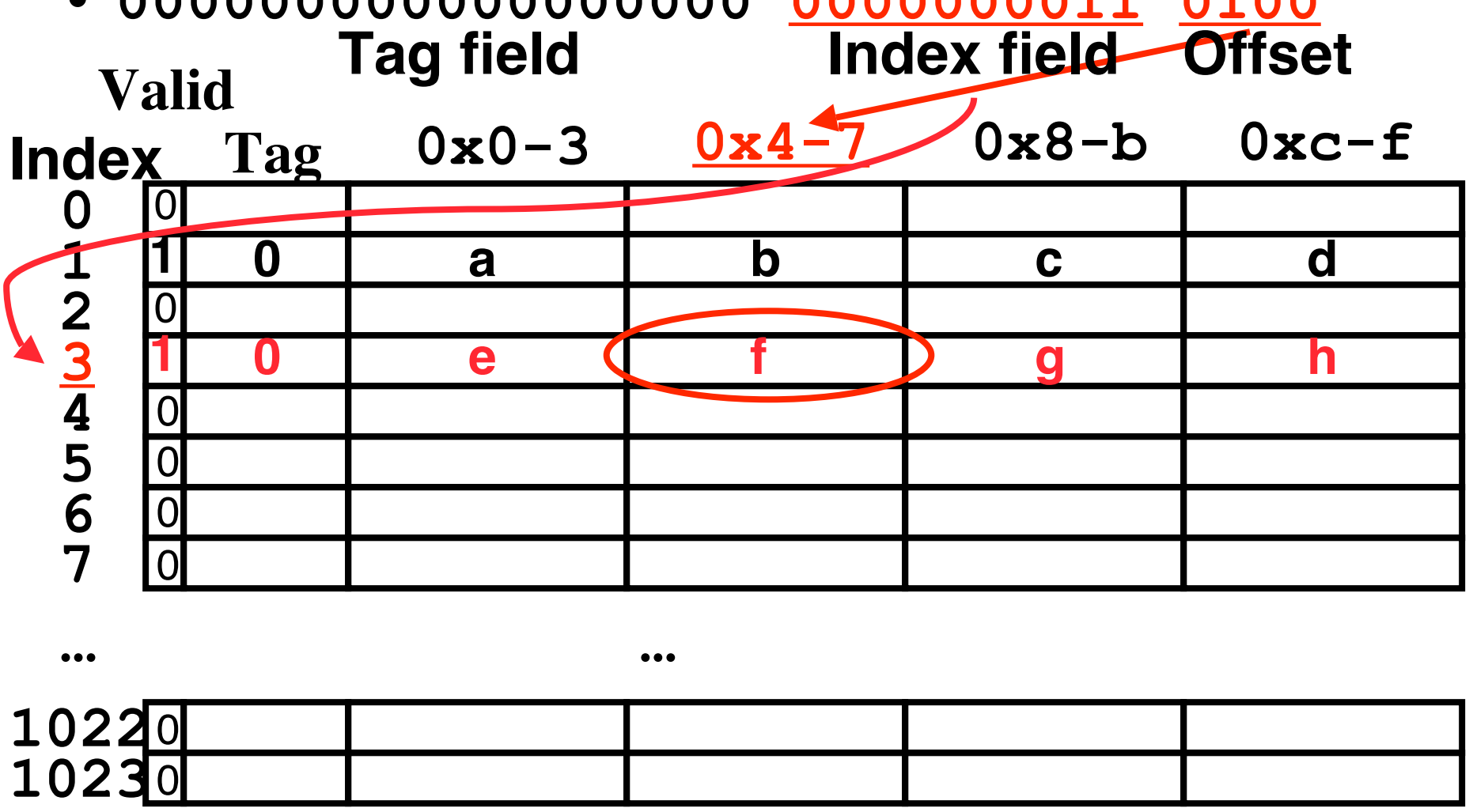
Tag field | Index field | Offset



Load that cache block, return word f

- 00000000000000000000 0000000011 0100

Tag field | Index field | Offset



4. Read 0x00008014 = 0…10 0..001 0100

- 00000000000000010 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					

So read Cache Block 1, Data is Valid

- 00000000000000010 0000000001 0100

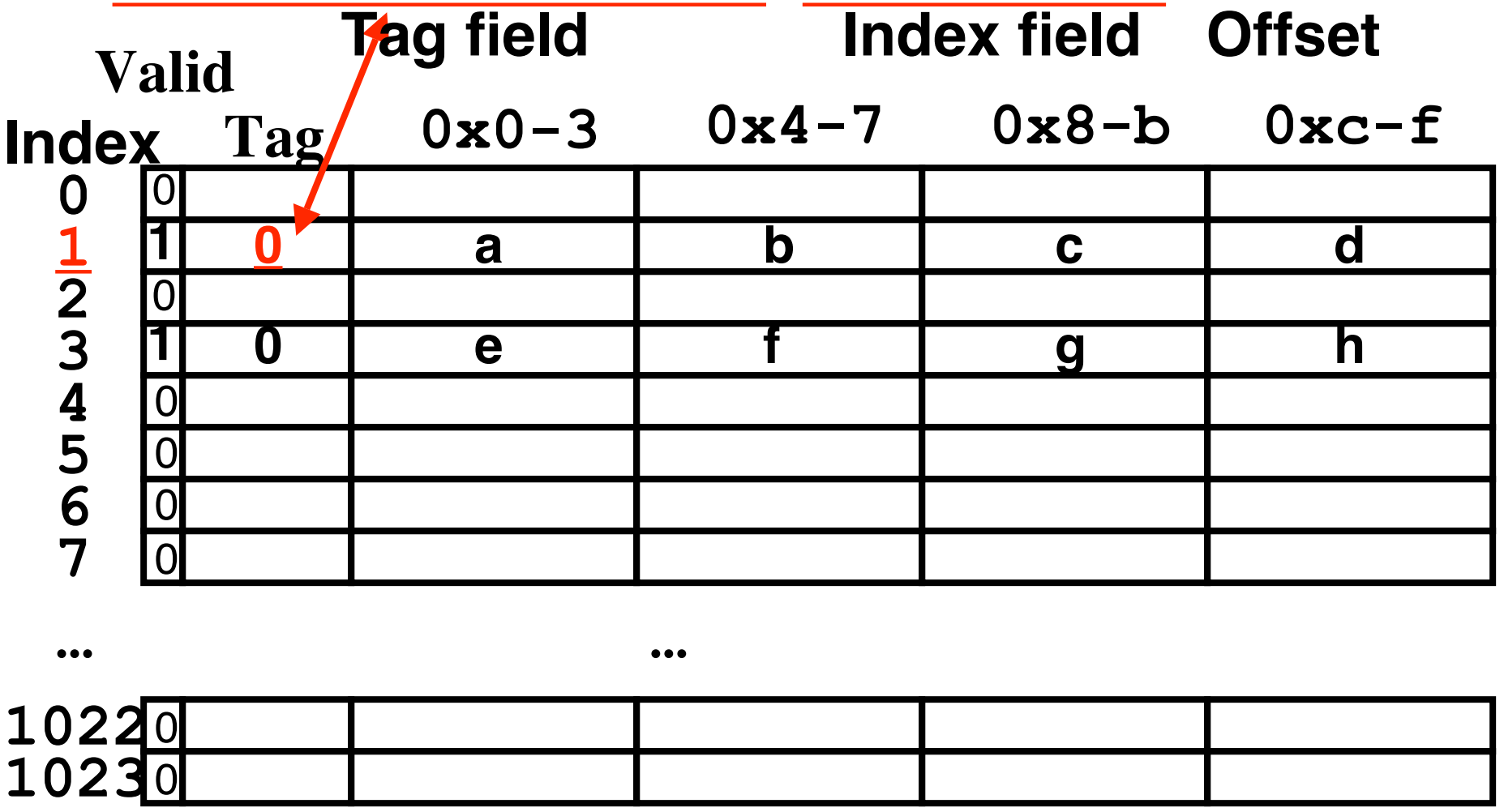
Tag field | Index field | Offset

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

Tag field | Index field | Offset



Miss, so replace block 1 with new data & tag

- 00000000000000010 0000000001 0100

Tag field | Index field | Offset

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					
...			...			
1022	0					
1023	0					

And return word j

- 0000000000000010 0000000001 0100

Tag field | Index field | Offset

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					
...			...			
1022	0					
1023	0					

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

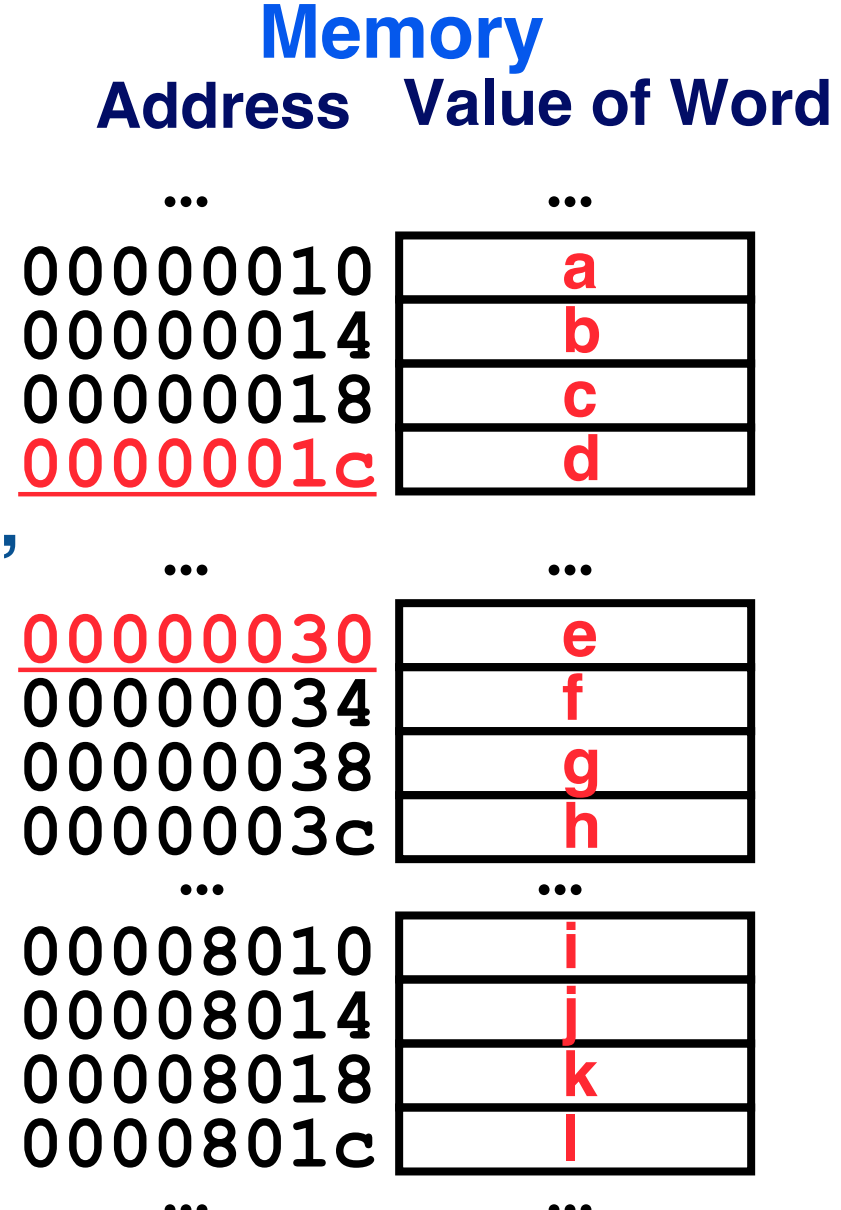
- 0x00000030 a hit

  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



Peer Instructions

1. All caches take advantage of spatial locality.
2. All caches take advantage of temporal locality.
3. On a read, the return value will depend on what is in the cache.

	ABC
0:	FFF
1:	FFT
2:	FTF
3:	FTT
4:	TFF
5:	TFT
6:	TTF
7:	TTT

And in Conclusion…

- **Mechanism for transparent movement of data among levels of a storage hierarchy**
  - set of address/value bindings
  - address ⇒ index to set of candidates
  - compare desired address with tag
  - service hit or miss
    - load new block and binding on miss