



信息科学与技术学院

School of Information Science and Technology

# CS 110

## Computer Architecture

### Cache III Direct-mapped & Set-Associative Cache

**Instructors:**

**Chundong Wang, Siting Liu & Yuan Xiao**

Course website: [https://toast-](https://toast-lab.sist.shanghaitech.edu.cn/courses/CS110@ShanghaiTech/Spring-2025/index.html)

[lab.sist.shanghaitech.edu.cn/courses/CS110@ShanghaiTech/Spring-2025/index.html](https://toast-lab.sist.shanghaitech.edu.cn/courses/CS110@ShanghaiTech/Spring-2025/index.html)

**School of Information Science and Technology (SIST)**

**ShanghaiTech University**

2025/4/22

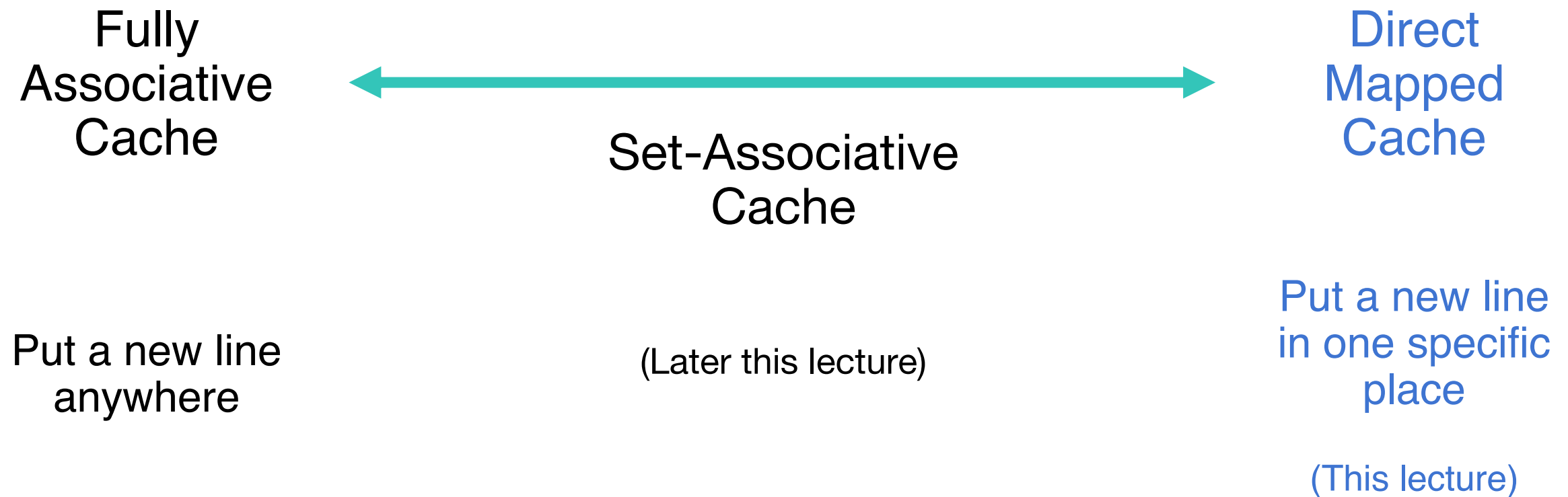
# Administratives

- Mid-term II tentatively May 15th; you can bring 2-page A4-sized double-sided cheat sheet, handwritten only!
- Project 2.1 released, start early, ddl May 5th!!!
- HW5 released, ddl May. 7th.
- Lab 9 checking this week. Lab 10 will be released and checked next week (longan nano RISC-V board).
  - CS110-only students need the board for lab 10 & 11 only;
  - CS110P project 4 will use this board to develop a game;
  - Keep them really well, because you have to return the board after lab/project checking;
- Discussion this week on cache
- Discussion Apr. 28th on midterm II review

# Administratives

- Special arrangement for Lab sessions during Labour Day Holiday
  - Apr. 27th, 18:00-19:40: **Thursday** lab sessions (Lab 7/8/9) to check Lab 10; (SIST 1B-106, 1B-108, SIST 1B-110)
  - Apr. 28th, 18:00-19:40: Monday lab sessions (Lab 1/2) to check Lab 10;
  - Apr. 29th, 19:50-21:30: Tuesday lab sessions (Lab 3/4/5/6) to check Lab 10;
  - May 6th/8th/12th: Tuesday/Thursday/Monday lab sessions to check Lab 11, respectively, which is to prepare for project 4, Longan nano RISC-V development. The circuit boards will be distributed during the lab sessions.

# Cache Design: Placement Policies

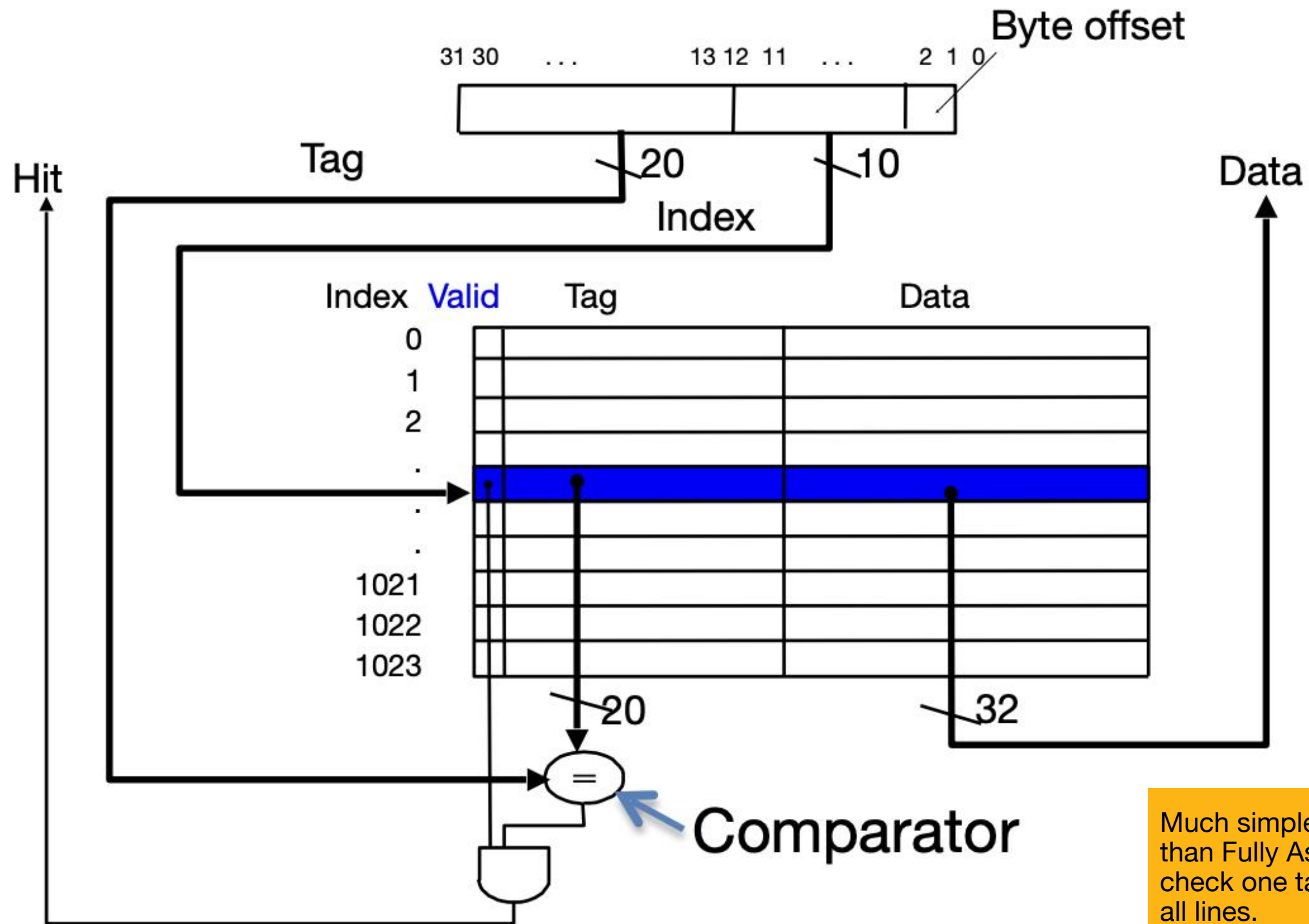


Fully associative caches need expensive hardware.

Less hardware.

# Memory blocks  $\gg$  # Cache blocks  
We need to carefully place memory blocks into cache blocks

# Direct Mapped Cache: Hardware



# Direct Mapped Cache

- Placement policy: The data at a memory address can be stored at **exactly one possible block** in the cache.
  - To check for existence in the cache, we only need to look in a **single** location in the cache.

Tag	VB	LRU	Dirty	DATA			

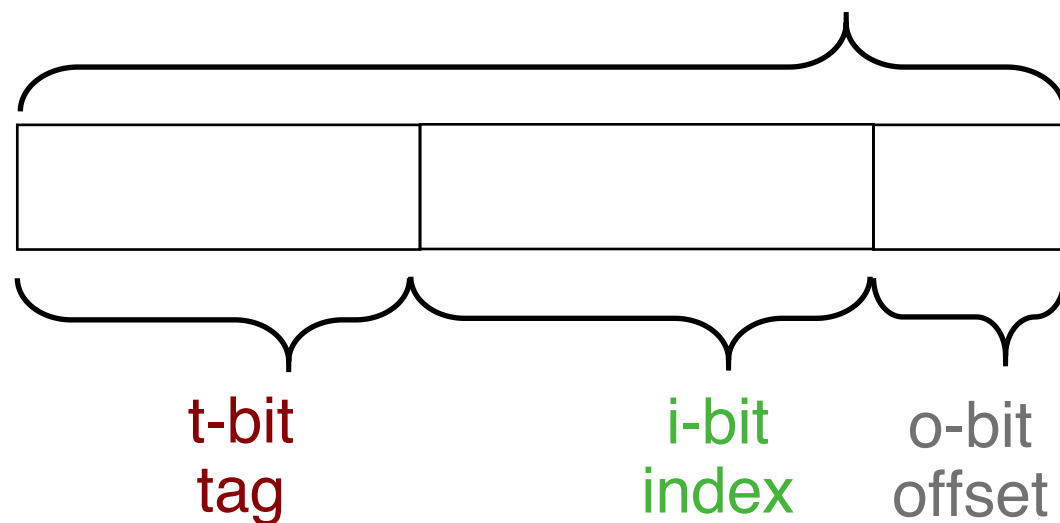
How do we ensure this?

# Direct Mapped Cache

- Placement policy: The data at a memory address can be stored at **exactly one possible block** in the cache.
  - To check for existence in the cache, we only need to look in a **single** location in the cache.

Tag	VB	LRU	Dirty	DATA			

16-bit  
Address



**index** to select  
block in cache

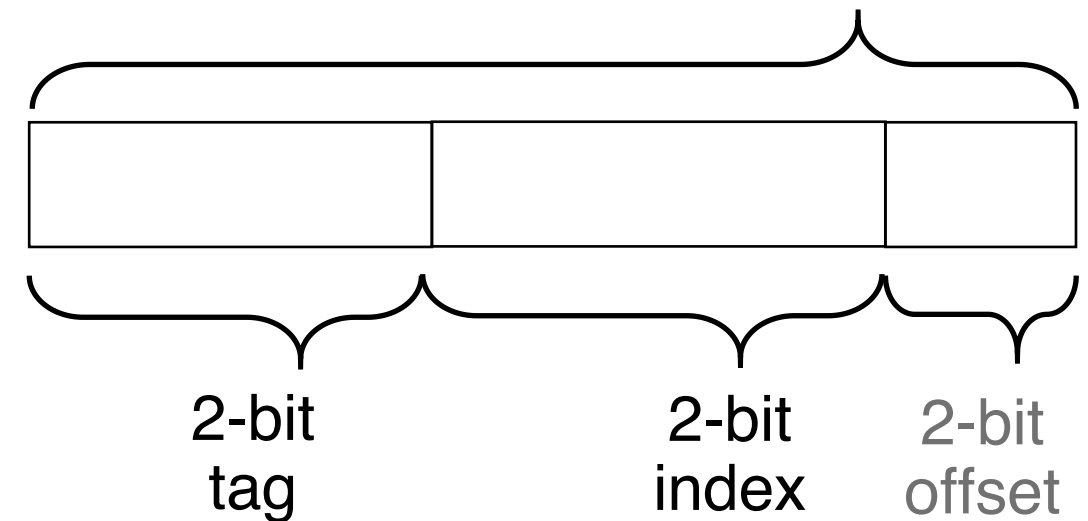
How do we ensure this?

- Add extra field to indicate which exact block to check.

# Direct Mapped Cache-Examples I

- Index number  $k$  goes to the  $k$ th cache block;
- Assume 4B block size; 6-bit address;

Address		
Tag	Index	Offset
00	00	xx
00	01	xx
00	10	xx
00	11	xx
01	00	xx
01	01	xx
01	10	xx
01	11	xx
10	00	xx
10	01	xx
10	10	xx
10	11	xx
11	00	xx
11	01	xx
11	10	xx
11	11	xx



Tag	DATA				Flag

Index 00

Index 01

Index 10

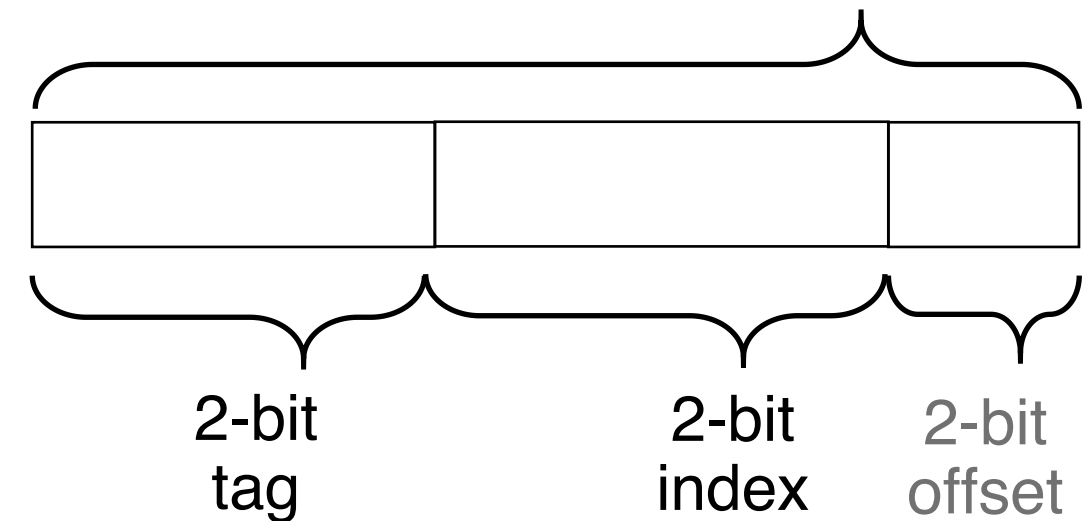
Index 11



# Direct Mapped Cache-Examples I

- Index number  $k$  goes to the  $k$ th cache block;
- Assume 4B block size; 6-bit address;

Address		
Tag	Index	Offset
00	00	xx
00	01	xx
00	10	xx
00	11	xx
01	00	xx
01	01	xx
01	10	xx
01	11	xx
10	00	xx
10	01	xx
10	10	xx
10	11	xx
11	00	xx
11	01	xx
11	10	xx
11	11	xx



Tag	DATA				Flag

Index 00

Index 01

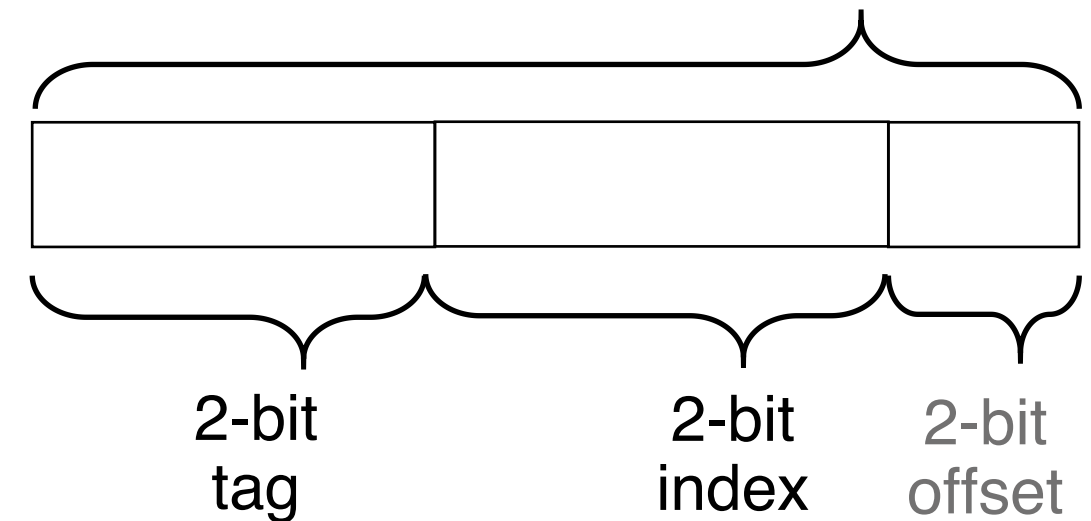
Index 10

Index 11

# Direct Mapped Cache-Examples I

- Index number  $k$  goes to the  $k$ th cache block;
- Assume 4B block size; 6-bit address;

Address		
Tag	Index	Offset
00	00	xx
00	01	xx
00	10	xx
00	11	xx
01	00	xx
01	01	xx
01	10	xx
01	11	xx
10	00	xx
10	01	xx
10	10	xx
10	11	xx
11	00	xx
11	01	xx
11	10	xx
11	11	xx



Tag	DATA				Flag

Index 00

Index 01

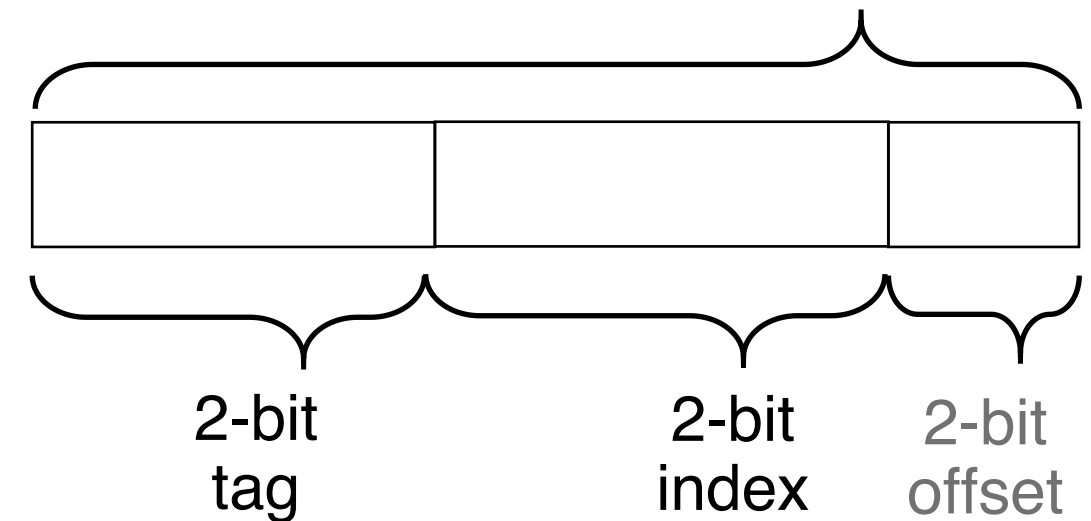
Index 10

Index 11

# Direct Mapped Cache-Examples I

- Index number  $k$  goes to the  $k$ th cache block;
- Assume 4B block size; 6-bit address;

Address		
Tag	Index	Offset
00	00	xx
00	01	xx
00	10	xx
00	11	xx
01	00	xx
01	01	xx
01	10	xx
01	11	xx
10	00	xx
10	01	xx
10	10	xx
10	11	xx
11	00	xx
11	01	xx
11	10	xx
11	11	xx



Tag	DATA				Flag

Index 00

Index 01

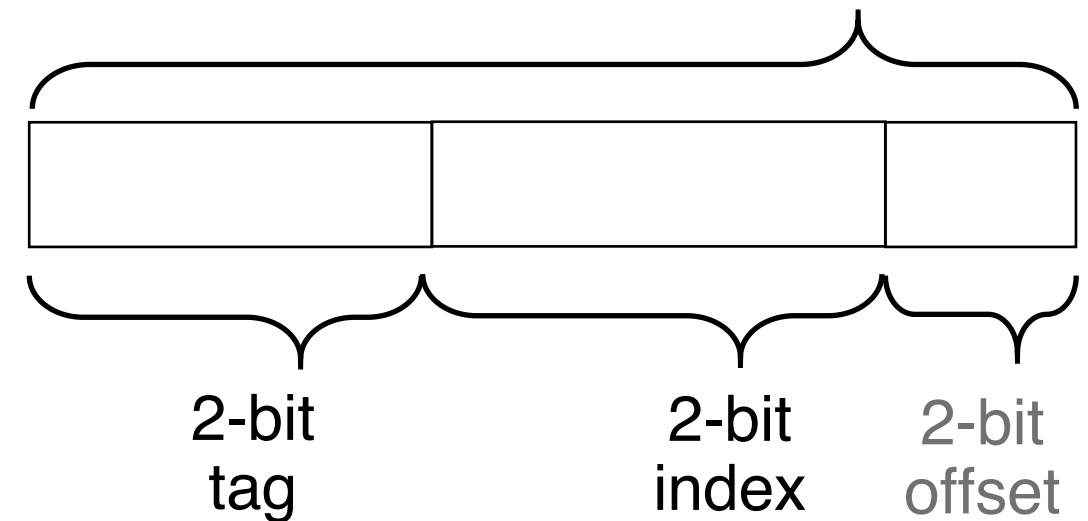
Index 10

Index 11

# Direct Mapped Cache-Examples I

- Index number  $k$  goes to the  $k$ th cache block;
- Assume 4B block size; 6-bit address;

Address		
Tag	Index	Offset
00	00	xx
00	01	xx
00	10	xx
00	11	xx
01	00	xx
01	01	xx
01	10	xx
01	11	xx
10	00	xx
10	01	xx
10	10	xx
10	11	xx
11	00	xx
11	01	xx
11	10	xx
11	11	xx



Tag	DATA				Flag

Index 00

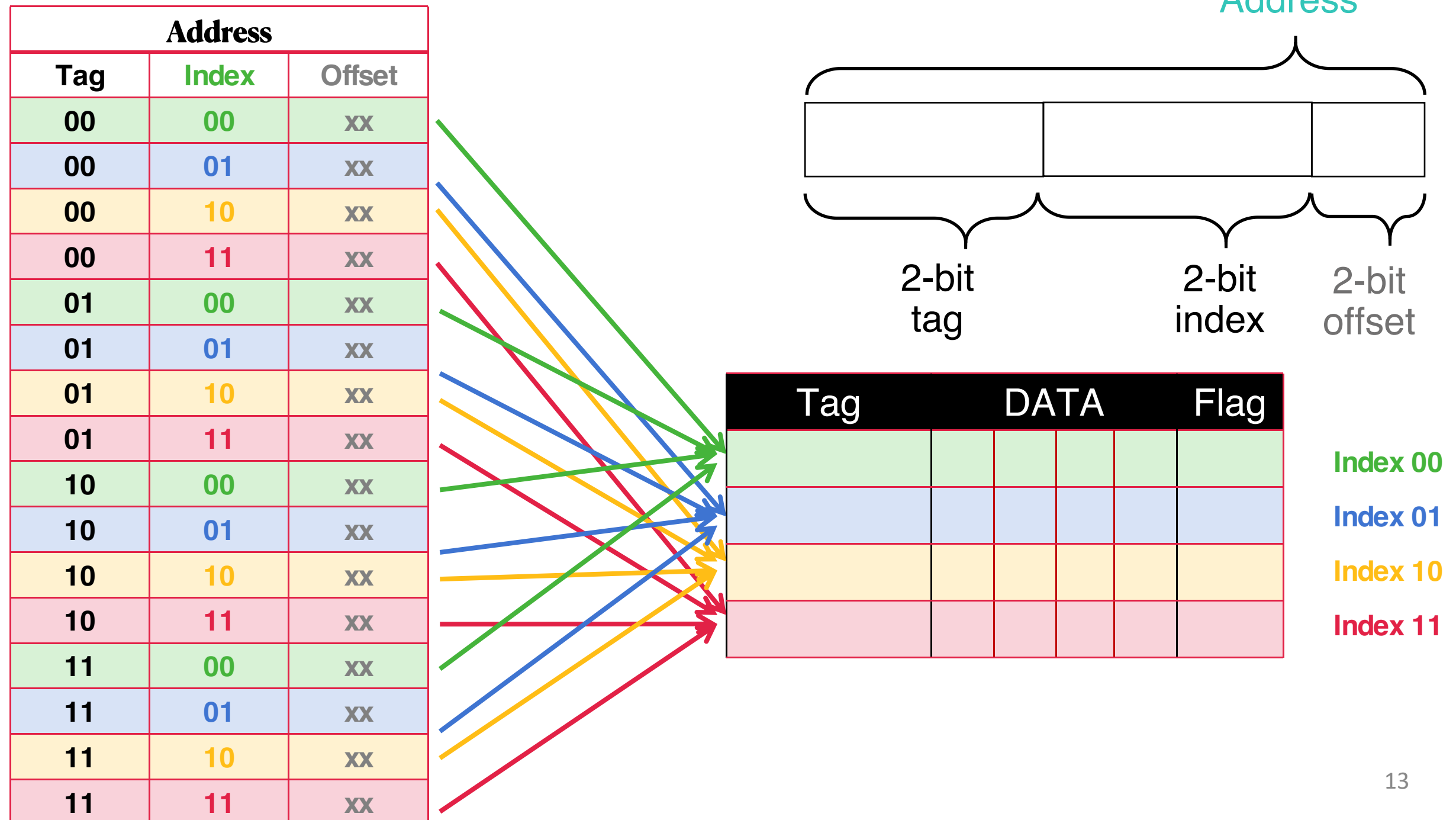
Index 01

Index 10

Index 11

# Direct Mapped Cache-Examples I

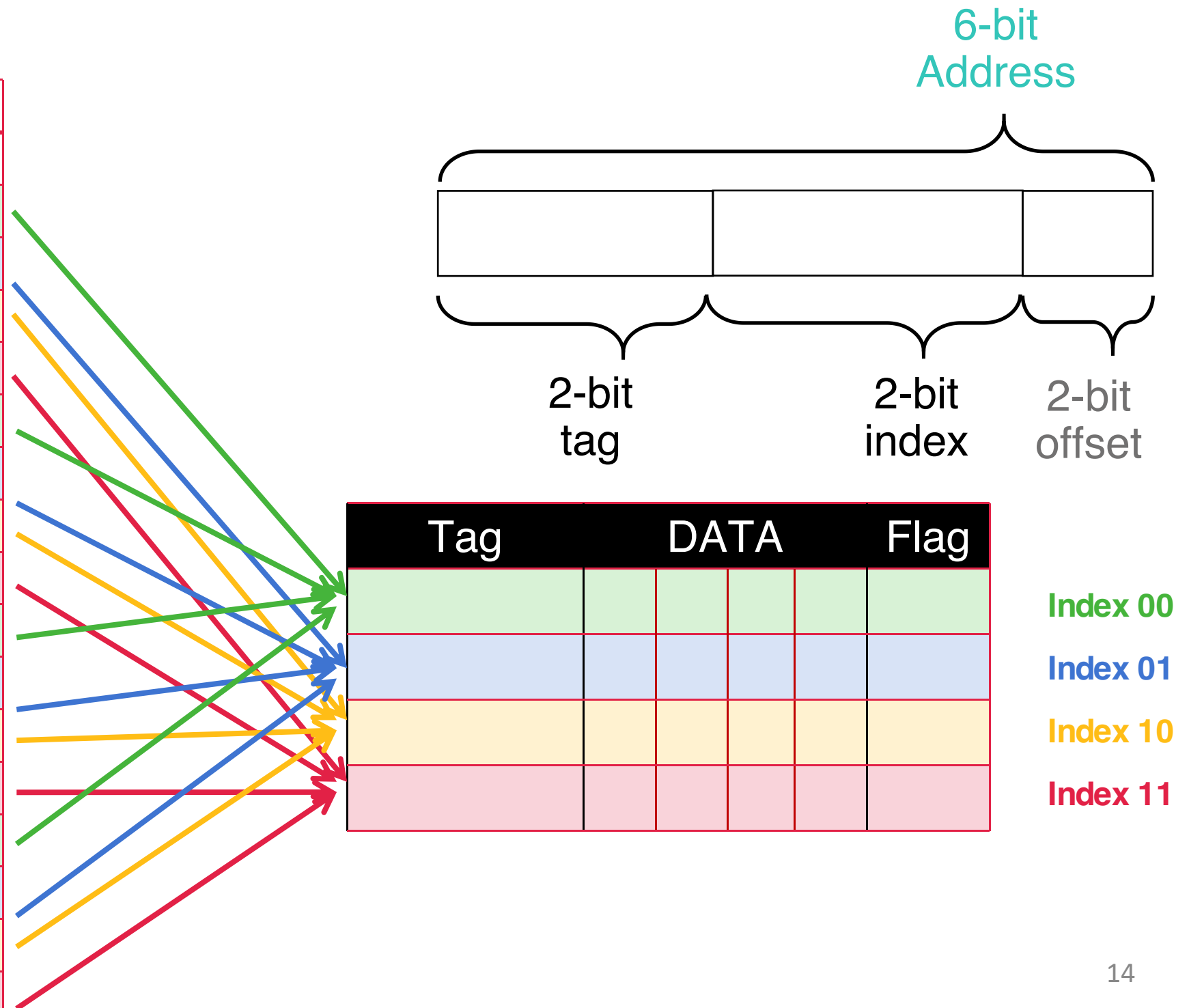
- Different addresses get same block index but different tags.



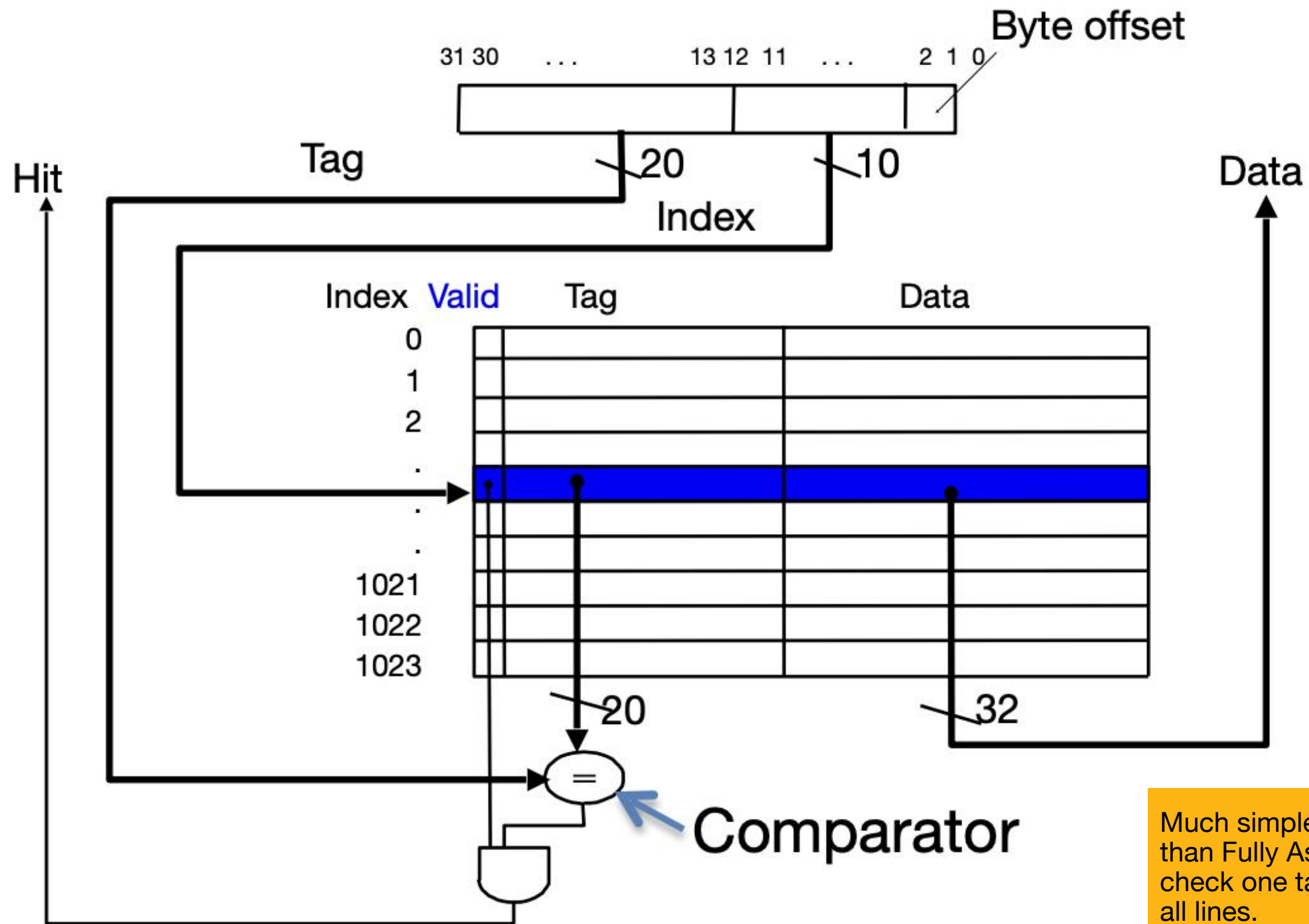
# Direct Mapped Cache-Examples I

- Replacement: Lines with **same block index** replace each other.

Address		
Tag	Index	Offset
00	00	xx
00	01	xx
00	10	xx
00	11	xx
01	00	xx
01	01	xx
01	10	xx
01	11	xx
10	00	xx
10	01	xx
10	10	xx
10	11	xx
11	00	xx
11	01	xx
11	10	xx
11	11	xx



# Direct Mapped Cache: Hardware



Much simpler to implement than Fully Associative! Just check one tag/line, and not all lines.

# Direct Mapped Cache-Examples II

- Index number  $k$  goes to the  $k$ th cache block;
- Assume 8B block size; 7-bit address;

Address		
Tag	Index	Offset
00	00	xxx
00	01	xxx
00	10	xxx
00	11	xxx
01	00	xxx
01	01	xxx
01	10	xxx
01	11	xxx
10	00	xxx
10	01	xxx
10	10	xxx
10	11	xxx
11	00	xxx
11	01	xxx
11	10	xxx
11	11	xxx

Address      maps to      with tag of  
0x37

Tag	DATA								Flag

Index 00

Index 01

Index 10

Index 11



# Direct Mapped Cache-Examples II

- Index number  $k$  goes to the  $k$ th cache block;
- Assume 8B block size; 7-bit address;

Address		
Tag	Index	Offset
00	00	xxx
00	01	xxx
00	10	xxx
00	11	xxx
01	00	xxx
01	01	xxx
01	10	xxx
01	11	xxx
10	00	xxx
10	01	xxx
10	10	xxx
10	11	xxx
11	00	xxx
11	01	xxx
11	10	xxx
11	11	xxx

Address      maps to      with tag of  
0x37

Tag	DATA								Flag
01									

Index 00

Index 01

Index 10

Index 11

# Direct Mapped Cache-Examples II

- Index number  $k$  goes to the  $k$ th cache block;
- Assume 8B block size; 7-bit address;

Address		
Tag	Index	Offset
00	00	xxx
00	01	xxx
00	10	xxx
00	11	xxx
01	00	xxx
01	01	xxx
01	10	xxx
01	11	xxx
10	00	xxx
10	01	xxx
10	10	xxx
10	11	xxx
11	00	xxx
11	01	xxx
11	10	xxx
11	11	xxx

Address      maps to      with tag of

0x37

0x7F

Tag	DATA								Flag
01									

Index 00

Index 01

Index 10

Index 11

# Direct Mapped Cache-Examples II

- Index number  $k$  goes to the  $k$ th cache block;
- Assume 8B block size; 7-bit address;

Address		
Tag	Index	Offset
00	00	xxx
00	01	xxx
00	10	xxx
00	11	xxx
01	00	xxx
01	01	xxx
01	10	xxx
01	11	xxx
10	00	xxx
10	01	xxx
10	10	xxx
10	11	xxx
11	00	xxx
11	01	xxx
11	10	xxx
11	11	xxx

Address      maps to      with tag of

0x37

0x7F

Tag	DATA								Flag
01									
11									

Index 00

Index 01

Index 10

Index 11

# Terminology for Direct Mapped Cache

- Cache capacity/size: total size of the cache (C)
- Cache block size:  $C_B \rightarrow$  decides the number of offset bits (o)

$$2^o = C_B$$

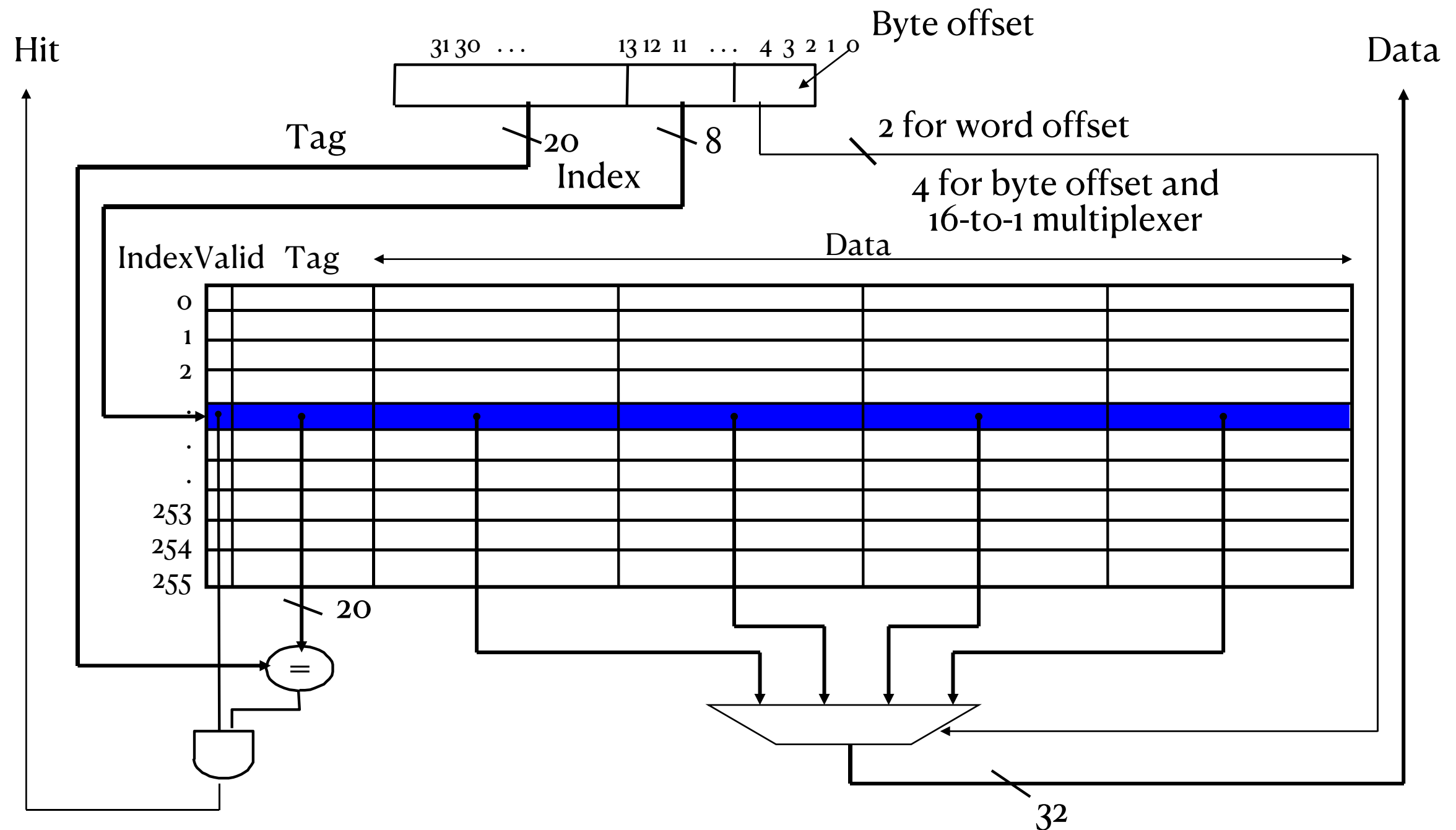
- Number of cache blocks (#cache block, N)

$$N * C_B = C$$

- Bit width of memory address (w): 6-bit in our examples
- Bit width of Index (i):  $\log_2(\text{\#cache blocks, } N)$
- Bit width of Tag (t):  $t = w - b - i = 2$  bits
- Hardware implication: comparators? actual storage requirement?

Tag	DATA								Flag
									Index 00
									Index 01
01									Index 10
11									Index 11

# Hardware Implementation



# Direct Mapped Cache-Working Examples

- Suppose 12-bit address, 8B cache blocks, 4 cache blocks
- $w = 12$ -bit width;  $o = 3$ -bit width;
- $i = 2$ -bit width;  $t = 12 - 3 - 2 = 7$  bit width
- 1. Load byte @0x43F
  - 0100 0011 1111

Tag	DATA							valid	
								0	Index 00
								0	Index 01
								0	Index 10
								0	Index 11

# Direct Mapped Cache-Working Examples

- Suppose 12-bit address, 8B cache blocks, 4 cache blocks
- $w = 12$ -bit width;  $o = 3$ -bit width;
- $i = 2$ -bit width;  $t = 12 - 3 - 2 = 7$  bit width
- 1. Load byte @0x43F
  - $0100\ 0011\ 1111$   
       7-bit 2-bit 3-bit  
       tag index offset

Tag	DATA	valid	
		0	Index 00
		0	Index 01
		0	Index 10
		0	Index 11

# Direct Mapped Cache-Working Examples

- Suppose 12-bit address, 8B cache blocks, 4 cache blocks
- $w = 12$ -bit width;  $o = 3$ -bit width;
- $i = 2$ -bit width;  $t = 12 - 3 - 2 = 7$  bit width
- 1. Load byte @0x43F
  - $0100\ 0011\ 1111$   
     7-bit   2-bit   3-bit  
     tag   index offset

Tag	DATA	valid	
		0	Index 00
		0	Index 01
		0	Index 10
0x21	s o m e d a t a	1	Index 11



# Direct Mapped Cache-Working Examples

- Suppose 12-bit address, 8B cache blocks, 4 cache blocks
- $w = 12$ -bit width;  $o = 3$ -bit width;
- $i = 2$ -bit width;  $t = 12 - 3 - 2 = 7$  bit width
- 1. Load byte @0x43F
- 2. Load byte @0x234
- $0010\ 0011\ 0100$   
                     7-bit    2-bit    3-bit  
                     tag    index    offset

Tag	DATA	valid	
		0	Index 00
		0	Index 01
0x11	s o m e d a t a	1	Index 10
0x21	s o m e d a t a	1	Index 11

# Direct Mapped Cache-Working Examples

- Suppose 12-bit address, 8B cache blocks, 4 cache blocks
- $w = 12$ -bit width;  $o = 3$ -bit width;
- $i = 2$ -bit width;  $t = 12 - 3 - 2 = 7$  bit width
- 1. Load byte @0x43F
- 2. Load byte @0x234
- 3. Load halfword @0x022
  - $\underbrace{0000}_{7\text{-bit tag}} \underbrace{0010}_{2\text{-bit index}} \underbrace{0010}_{3\text{-bit offset}}$

Tag	DATA								valid	
0x01	s	o	m	e	d	a	t	a	1	Index 00
									0	Index 01
0x11	s	o	m	e	d	a	t	a	1	Index 10
0x21	s	o	m	e	d	a	t	a	1	Index 11

# Direct Mapped Cache-Working Examples

- Suppose 12-bit address, 8B cache blocks, 4 cache blocks
- $w = 12$ -bit width;  $o = 3$ -bit width;
- $i = 2$ -bit width;  $t = 12 - 3 - 2 = 7$  bit width
- 1. Load byte @0x43F
- 2. Load byte @0x234
- 3. Load halfword @0x022
- 4. Load word @0x43C

• 0100 0011 1100  
       └───┬──┬──┘  
       7-bit 2-bit 3-bit  
       tag index offset

Tag	DATA	valid	
0x01	s o m e d a t a	1	Index 00
		0	Index 01
0x11	s o m e d a t a	1	Index 10
0x21	s o m e d a t a	1	Index 11

Cache  
HIT!!!

# Direct Mapped Cache-Working Examples

- Suppose 12-bit address, 8B cache blocks, 4 cache blocks
- $w = 12$ -bit width;  $o = 3$ -bit width;
- $i = 2$ -bit width;  $t = 12 - 3 - 2 = 7$  bit width
- 1. Load byte @0x43F
- 2. Load byte @0x234
- 3. Load halfword @0x022
- 4. Load word @0x43C
- 5. Load byte @0xF4D

Tag	DATA	valid	
0x01	s o m e d a t a	1	Index 00
0x7A	s o m e d a t a	1	Index 01
0x11	s o m e d a t a	1	Index 10
0x21	s o m e d a t a	1	Index 11

- $\underbrace{1111\ 0100}_{7\text{-bit tag}} \underbrace{11}_{2\text{-bit index}} \underbrace{01}_{3\text{-bit offset}}$

# Do We still Need LRU?

- Suppose 12-bit address, 8B cache blocks, 4 cache blocks
- $w = 12$ -bit width;  $o = 3$ -bit width;
- $i = 2$ -bit width;  $t = 12 - 3 - 2 = 7$  bit width
- 1. Load byte @0x43F
- 2. Load byte @0x234
- 3. Load halfword @0x022
- 4. Load word @0x43C
- 5. Load byte @0xF4D
- 6. Load word @0x120

Tag	DATA	valid	
0x01	s o m e d a t a	1	Index 00
0x7A	s o m e d a t a	1	Index 01
0x11	s o m e d a t a	1	Index 10
0x21	s o m e d a t a	1	Index 11

0001 0010 0000

Cache  
MISS!!!

No other choice but to replace  
00-indexed cache block

# No Replacement Policy Required!!!

- Suppose 12-bit address, 8B cache blocks, 4 cache blocks
- $w = 12$ -bit width;  $o = 3$ -bit width;
- $i = 2$ -bit width;  $t = 12 - 3 - 2 = 7$  bit width
- 1. Load byte @0x43F
- 2. Load byte @0x234
- 3. Load halfword @0x022
- 4. Load word @0x43C
- 5. Load byte @0xF4D
- 6. Load word @0x120

Tag	DATA	valid	Victim evicted
0x09	s o m e d a t a	1	Index 00
0x7A	s o m e d a t a	1	Index 01
0x11	s o m e d a t a	1	Index 10
0x21	s o m e d a t a	1	Index 11

0001 0010 0000

Cache  
MISS!!!

# What about Write Policy?

- 1. Load byte @0x43F
- 2. Load byte @0x234
- 3. Load halfword @0x022
- 4. Load word @0x43C
- 5. Load byte @0xF4D
- 6. Load word @0x120
- 7. Store byte @0xF48, 0x0

Tag	DATA	valid	
0x09	s o m e d a t a	1	Index 00
0x7A	s o m e d a t a	1	Index 01
0x11	s o m e d a t a	1	Index 10
0x21	s o m e d a t a	1	Index 11

- $\underbrace{1111}_{7\text{-bit tag}} \underbrace{0100}_{2\text{-bit index}} \underbrace{1000}_{3\text{-bit offset}}$

Cache  
HIT!!!

- Write-through vs. write-back

# What about Write Policy?

- 7. Store byte @0xF48, 0x0

1111 0100 1000  
 7-bit 2-bit 3-bit  
 tag index offset

Cache  
HIT!!!

Tag	DATA	valid	dirty	
0x09	s o m e d a t a	1	0	Index 00
0x7A	s o m e d a t 0	1	1	Index 01
0x11	s o m e d a t a	1	0	Index 10
0x21	s o m e d a t a	1	0	Index 11

- Write-back
  - Update the cache block and set dirty bit;
  - Wait until it is replaced by another cache block;



# What about Write Policy?

- 7. Store byte @0xF48, 0x0

1111 0100 1000  
 └───┬───┬───┘  
 7-bit 2-bit 3-bit  
 tag index offset

Cache  
HIT!!!

Tag	DATA	valid
0x09	s o m e d a t a	1
0x7A	s o m e d a t 0	1
0x11	s o m e d a t a	1
0x21	s o m e d a t a	1

Index 00

Index 01

Index 10

Index 11

- Write-back
  - Update the cache block and set dirty bit;
  - Again, do not need LRU or replacement policy;
  - Wait until it is replaced by another cache block;
- Write-through
  - Write both the cache block and memory, **do not** need dirty bit;
  - Write buffer stops CPU from stalling if memory cannot keep up
  - Write buffer may have multiple entries to absorb bursts of writes

# What about Write Policy?

- 7. Store byte @0x300, 0x0

0011 0000 0000

7-bit 2-bit 3-bit  
tag index offset

Cache  
MISS!!!

Tag	DATA	valid	dirty	
0x18	s o m e d a t a 0	1	1	Index 00
0x7A	s o m e d a t a	1	0	Index 01
0x11	s o m e d a t a	1	0	Index 10
0x21	s o m e d a t a	1	0	Index 11

- Write-allocate & Write-back
  - Load the cache block, update the cache block and set dirty bit;
  - Again, do not need LRU or replacement policy;
  - Wait until it is replaced by another cache block;

# What about Write Policy?

- 7. Store byte @0x300, 0x0

0011 0000 0000

7-bit 2-bit 3-bit  
tag index offset

Cache  
MISS!!!

Tag	DATA	valid
0x18	s o m e d a t a 0	1
0x7A	s o m e d a t a	1
0x11	s o m e d a t a	1
0x21	s o m e d a t a	1

Index 00

Index 01

Index 10

Index 11

- Write-allocate & Write-back
  - Load the cache block, update the cache block and set dirty bit;
  - Again, do not need LRU or replacement policy;
  - Wait until it is replaced by another cache block;
- Write-allocate & Write-through
  - Load the cache block, update the cache block and memory (next-level);
  - Do not need LRU or replacement policy or dirty bit;

# What about Write Policy?

- 7. Store byte @0x300, 0x0

0011 0000 0000  
 └───┬───┬───┘  
 7-bit 2-bit 3-bit  
 tag index offset

Cache  
MISS!!!

Tag	DATA	valid
0x09	s o m e d a t a	1
0x7A	s o m e d a t a	1
0x11	s o m e d a t a	1
0x21	s o m e d a t a	1

Index 00

Index 01

Index 10

Index 11

- Non-write-allocate
  - Directly update the (next-level) memory;

# Direct Mapped Cache-Working Example II

- Worst case reference case with a 4B cache blocks;
- 4 cache blocks with index  $i = 2$ -bit width;
- $t = 12 - 2 - 2 = 8$  bit width;
- Load the 0th element and the 4th element of an `int` array alternately;
- Cold start with an empty cache;

- $\text{xxxx xxx0 } \textcolor{green}{00} \textcolor{violet}{00}$  Address of the 0th element  

$\underbrace{\text{xxxx xxx0}}_{\text{8-bit tag}}$

$\underbrace{\textcolor{green}{00}}_{\text{2-bit index}}$

$\underbrace{\textcolor{violet}{00}}_{\text{2-bit offset}}$

Cache  
MISS!!!

Tag	DATA
xxxx xxx0	

# Understanding Cache Misses: 3Cs

- **C**ompulsory (cold start or process migration, 1st reference):
  - First access to block impossible to avoid; small effect for long running programs
  - Solution: increase block size (increases miss penalty; very large blocks could increase miss rate)
- **C**apacity:
  - Cache cannot contain all blocks accessed by the program
  - Solution: increase cache size (may increase access time)
- **C**onflict (**C**ollision):
  - Multiple memory locations mapped to the same cache location, conflict even when the cache has not reached full capacity.
  - Solution 1: increase cache size
  - Solution 2: increase associativity (may increase access time)

# Understanding Cache Misses: 3Cs

- **C**ompulsory (cold start or process migration, 1st reference):
  - First access to block impossible to avoid; small effect for long running programs
  - Solution: increase block size (increases miss penalty; very large blocks could increase miss rate)
- **C**apacity (last lecture, in FA cache, we have to evict victim when cache is full)
  - Cache cannot contain all blocks accessed by the program
  - Solution: increase cache size (may increase access time)
- **C**onflict (**C**ollision):
  - Multiple memory locations mapped to the same cache location, conflict even when the cache has not reached full capacity.
  - Solution 1: increase cache size
  - Solution 2: increase associativity (may increase access time)

# Compulsory Miss

- Worst case reference case with a 4B cache blocks;
- 4 cache blocks with index  $i = 2$ -bit width;
- $t = 12 - 2 - 2 = 8$  bit width;
- Load the 0th element and the 4th element of an `int` array alternately;
- Cold start with an empty cache;

Tag	DATA
xxxx xxx0	a[0]

- $\underbrace{\text{xxxx xxx0}}_{\substack{\text{8-bit} \\ \text{tag}}} \underbrace{\text{00}}_{\substack{\text{2-bit} \\ \text{index}}} \underbrace{\text{00}}_{\substack{\text{2-bit} \\ \text{offset}}} \quad \text{Address of the 0th element}$





# Working Example II (Cont'd)

- Worst case reference case with a 4B cache blocks;
- 4 cache blocks with index  $i = 2$ -bit width;
- $t = 12 - 2 - 2 = 8$  bit width;
- Load the 0th element and the 4th element of an `int` array alternately;
- Cold start with an empty cache;

Tag	DATA
xxxx xxx0	a[0]

• xxxx xxx0 0000 Address of the 0th element

• xxxx xxx1 0000 Address of the 4th element

8-bit tag    2-bit index    2-bit offset  
 └──────────┘    └─┘ └─┘

Cache  
MISS!!!

# Working Example II (Cont'd)

- Worst case reference case with a 4B cache blocks;
- 4 cache blocks with index  $i = 2$ -bit width;
- $t = 12 - 2 - 2 = 8$  bit width;
- Load the 0th element and the 4th element of an `int` array alternately;
- Cold start with an empty cache;

Tag	DATA
xxxx xxx1	a[4]

• xxxx xxx0 0000 Address of the 0th element

• xxxx xxx1 0000 Address of the 4th element

8-bit tag    2-bit index    2-bit offset  
 └──────────┘    └─┘ └─┘

Cache  
MISS!!!

# Working Example II (Cont'd)

- Worst case reference case with a 4B cache blocks;
- 4 cache blocks with index  $i = 2$ -bit width;
- $t = 12 - 2 - 2 = 8$  bit width;
- Load the 0th element and the 4th element of an `int` array alternately;
- Cold start with an empty cache;

Tag	DATA
xxxx xxx1	a[4]

- xxxx xxx0 0000 Address of the 0th element
- xxxx xxx1 0000 Address of the 4th element
- xxxx xxx0 0000 Address of the 0th element
 

$\underbrace{\hspace{2em}}$   
8-bit  
tag

$\underbrace{\hspace{1em}}$   
2-bit  
index

$\underbrace{\hspace{1em}}$   
2-bit  
offset



# Working Example II (Cont'd)

- Worst case reference case with a 4B cache blocks;
- 4 cache blocks with index  $i = 2$ -bit width;
- $t = 12 - 2 - 2 = 8$  bit width;
- Load the 0th element and the 4th element of an `int` array alternately;
- Cold start with an empty cache;

Tag	DATA
xxxx xxx0	a[0]

- xxxx xxx0 0000 Address of the 0th element
- xxxx xxx1 0000 Address of the 4th element
- xxxx xxx0 0000 Address of the 0th element
 

$\underbrace{\hspace{2em}}$   
8-bit  
tag

$\underbrace{\hspace{1em}}$   
2-bit  
index

$\underbrace{\hspace{1em}}$   
2-bit  
offset



# Working Example II (Cont'd)

- Worst case reference case with a 4B cache blocks;
- 4 cache blocks with index  $i = 2$ -bit width;
- $t = 12 - 2 - 2 = 8$  bit width;
- Load the 0th element and the 4th element of an `int` array alternately;
- Cold start with an empty cache;

Tag	DATA
xxxx xxx0 0000	a[0]

- xxxx xxx0 0000 Address of the 0th element
  - xxxx xxx1 0000 Address of the 4th element
  - xxxx xxx0 0000 Address of the 0th element
  - xxxx xxx1 0000 Address of the 4th element
- 8-bit tag
2-bit index
2-bit offset



# Working Example II (Cont'd)

- Worst case reference case with a 4B cache blocks;
- 4 cache blocks with index  $i = 2$ -bit width;
- $t = 12 - 2 - 2 = 8$  bit width;
- Load the 0th element and the 4th element of an `int` array alternately;
- Cold start with an empty cache;

Tag	DATA
xxxx xxx1	a[4]

- xxxx xxx0 0000 Address of the 0th element
- xxxx xxx1 0000 Address of the 4th element
- xxxx xxx0 0000 Address of the 0th element
- xxxx   xxx1   0000  
{   {   {  
 8-bit   2-bit   2-bit  
 tag   index   offset

 Address of the 4th element



# Understanding Cache Misses: 3Cs

- **C**ompulsory (cold start or process migration, 1st reference):
  - First access to block impossible to avoid; small effect for long running programs
  - Solution: increase block size (increases miss penalty; very large blocks could increase miss rate)
- **C**apacity (last lecture, in FA cache, we have to evict victim when cache is full)
  - Cache cannot contain all blocks accessed by the program
  - Solution: increase cache size (may increase access time)
- **C**onflict (**C**ollision):
  - Multiple memory locations mapped to the same cache location, conflict even when the cache has not reached full capacity.
  - Solution 1: increase cache size
  - Solution 2: increase associativity (may increase access time)

In this class, we will only distinguish between **compulsory and non-compulsory misses**.

# 3Cs: Question

- Can **C**onflict Misses occur on a fully associative cache?



# Identify 3Cs if You Must

- Run an address trace against a set of caches.
  - (thanks Prof. Kubiawicz for the algorithm).
- 1. First, consider an infinite-size, fully-associative cache. For every miss that occurs now, consider it a **compulsory miss**.
- 2. Next, consider a finite-sized cache (of the size you want to examine) with full-associativity. Every miss that is not in #1 is a **capacity miss**.
- 3. Finally, consider a finite-sized cache with finite-associativity. All of the remaining misses that are not #1 or #2 are **conflict misses**.
  - Fully associative, ... ..., 16-way, 8-way, 4-way, 2-way, 1-way

# Summary on FA/DM caches

- **Fully Associative (FA)**

- **Placement policy:** Data at any memory address can be associated with any cache block
- **Expensive hardware:** **check all blocks**
- **Write policies:** write-back, write-through
- Must decide **replacement policy** (LRU, MRU, FIFO, etc.)

- **Direct Mapped (DM)**

- **Placement policy:** Each memory address is associated with exactly one possible line in the cache.
- **Simpler hardware:** **check one line**
- **Write policies:** write-back, write-through
- **No replacement policy**, because we know which line to replace
- **Has conflict misses**