

## Cache Cont'd

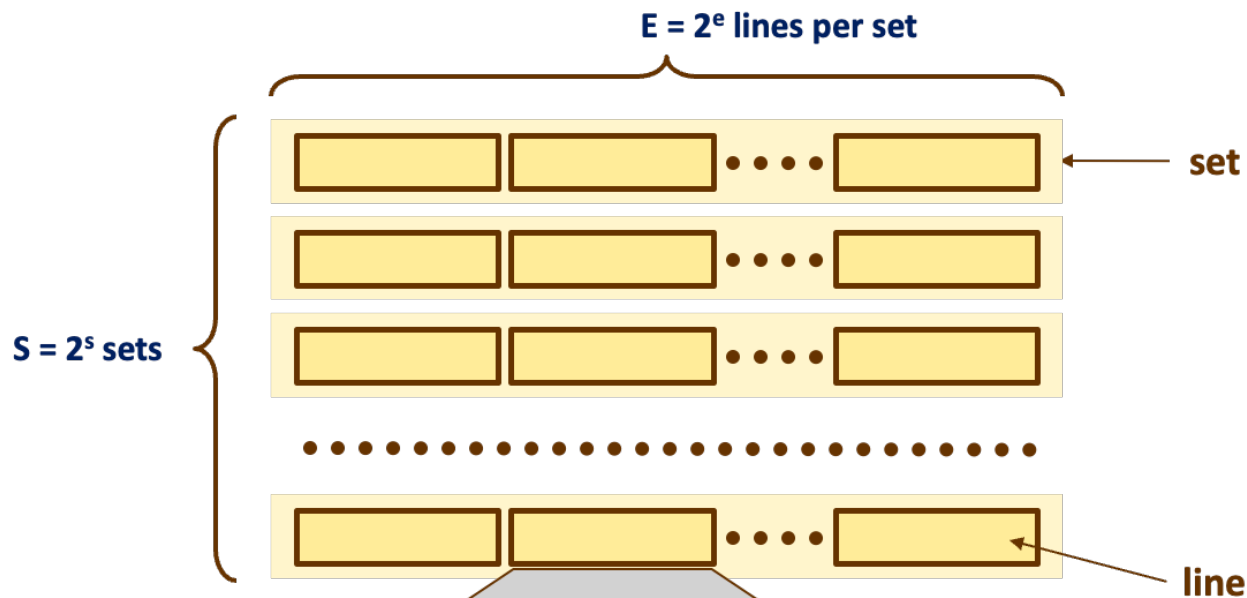
---



**Fang Song**  
Portland State University

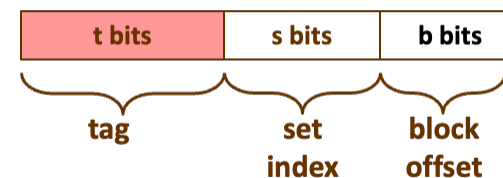
Slides adapted from  
CS205@PSU(Prof. Li) / CS15-213 @CMU

# Recall: Cache Org. and Read



1. Locate set
2. Check if any line in set has matching tag
3. Yes + line valid: hit
4. Locate data starting at offset

Address of word:



$B = 2^b$  bytes per cache block (the data)

**Cache size:**  
 $C = S \times E \times B$  data bytes

# Understanding Cache Parameters

Question: What different values of S, E, B imply?

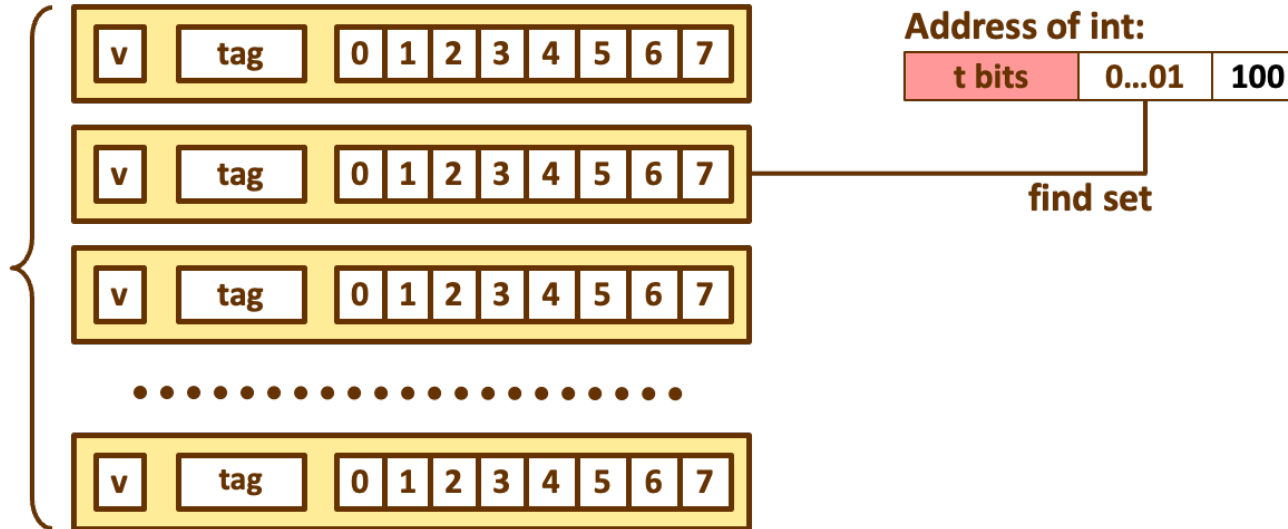
- E = 1 (Direct mapped caches) “assigned seating”
- S = 1 (Fully associative caches) “open seating”
  - Cache has a single set; a memory address can map to any cache line.
- General cases are called **Set associative** caches. “assigned coach”
  - A memory address can map to any cache line within a fixed set.

# Direct Mapped Cache (E=1)

- Each memory address maps to a single cache line.
  - Advantage: Simplest to implement

Example: Cache block size 8 bytes

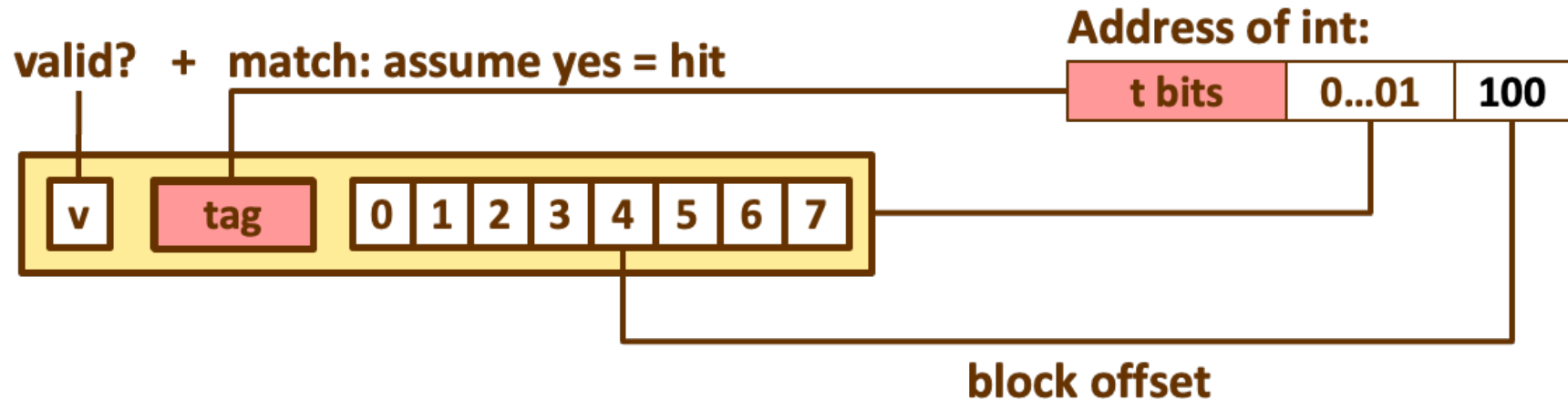
$S = 2^s$  sets



# Direct Mapped Cache (E=1, B=8)

- ◎ Each memory address maps to a single cache line.
  - Advantage: Simplest to implement

Example: Cache block size 8 bytes



No match? Then old line is evicted and replaced

# Direct Mapped Cache: Exercise 1

0000	0x00
0001	0x11
0010	0x22
0011	0x33
0100	0x44
0101	0x55
0110	0x66
0111	0x77
1000	0x88
1001	0x99
1010	0xAA
1011	0xBB
1100	0xCC
1101	0xDD
1110	0xEE
1111	0xFF

**Cache** (S,E,B) = (4,1,1)

Index Tag Data

00		
01		
10		
11		

$m = 4$  bits of address = 16 bytes of memory

$S = 4$  sets

$s = \log S = 2$  Index is 2 **LSB** of address

$E = 1$  line per set

$b = \log B = 0$

$B = 1$  byte per line/block

$t = 4 - (s+b) = 2$  Tag is 2 **MSB** of address

**t (2 bits) s (2 bits)**

# Direct Mapped Cache: Exercise 1

0000	0x00
0001	0x11
0010	0x22
0011	0x33
0100	0x44
0101	0x55
0110	0x66
0111	0x77
1000	0x88
1001	0x99
1010	0xAA
1011	0xBB
1100	0xCC
1101	0xDD
1110	0xEE
1111	0xFF

**Cache** (S,E,B) = (4,1,1)

Index Tag Data

00		
01		
10		
11		

$m = 4$  bits of address = 16 bytes of memory

$S = 4$  sets

$s = \log S = 2$  Index is 2 **LSB** of address

$L = 1$  line per set

$b = \log B = 0$

$B = 1$  byte per line/block

$t = 4 - (s+b) = 2$  Tag is 2 **MSB** of address

**t (2 bits) s (2 bits)**

# Direct Mapped Cache: Exercise 1

0000	0x00
0001	0x11
0010	0x22
0011	0x33
0100	0x44
0101	0x55
0110	0x66
0111	0x77
1000	0x88
1001	0x99
1010	0xAA
1011	0xBB
1100	0xCC
1101	0xDD
1110	0xEE
1111	0xFF

Cache (S,E,B) = (4,1,1)

Index	Tag	Data
00	00	0x00
01		
10		
11	00	0x33

10 0xFF

Access pattern

0000  
0011  
1000  
0011  
1000

miss  
miss  
tag 10 ≠ 00 miss  
hit!  
10 = 10 hit!

$$\text{miss rate} = \frac{3}{5} = 60\%$$

t (2 bits) s (2 bits)



# Direct Mapped Cache: Exercise 2

0000	0x00
0001	0x11
0010	0x22
0011	0x33
0100	0x44
0101	0x55
0110	0x66
0111	0x77
1000	0x88
1001	0x99
1010	0xAA
1011	0xBB
1100	0xCC
1101	0xDD
1110	0xEE
1111	0xFF

**Cache** (S,E,B) = (2,1,2)

Index Tag Data1 Data0

0			
1			

b =

s =

t = 4 - (s+b) =

# Direct Mapped Cache: Exercise 2

0000	0x00
0001	0x11
0010	0x22
0011	0x33
0100	0x44
0101	0x55
0110	0x66
0111	0x77
1000	0x88
1001	0x99
1010	0xAA
1011	0xBB
1100	0xCC
1101	0xDD
1110	0xEE
1111	0xFF

Cache (S,E,B) = (2,1,2)

Index Tag Data1 Data0

0			
1			

$$b = 1$$

$$s = 1$$

$$t = 4 - (s+b) = 2$$

<b>t (2 bits)</b>	<b>s (1 bit)</b>	<b>b (1 bit)</b>
-------------------	------------------	------------------

# Direct Mapped Cache: Exercise 2

0000	0x00
0001	0x11
0010	0x22
0011	0x33
0100	0x44
0101	0x55
0110	0x66
0111	0x77
1000	0x88
1001	0x99
1010	0xAA
1011	0xBB
1100	0xCC
1101	0xDD
1110	0xEE
1111	0xFF

Cache (S,E,B) = (2,1,2)

Index	Tag	Data1	Data0
0	00	0x00	0x11
1	00	0x22	0x33

Access pattern

0000  
 0001  
 0010  
 0011  
 0100  
 0101

miss

hit (0x11)

miss

hit

01 ≠ 00 miss

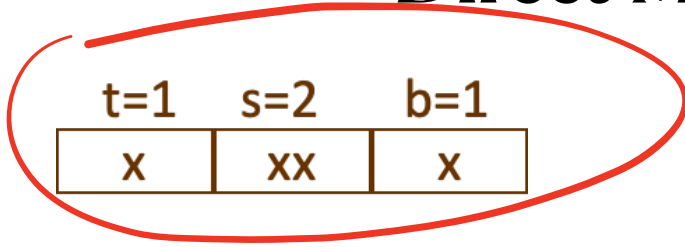
01 = 01 hit

(benefit of locality)

t (2 bits) | s (1 bit) | b (1 bit)

miss rate  $\frac{3}{6} = 50\%$

# Direct-Mapped Cache Simulation



$M=16$  byte addresses,  $B=2$  bytes/block,

$S=4$  sets,  $E=1$  Blocks/set

Address trace (reads, one byte per read):

0 [0000<sub>2</sub>], miss  
 1 [0001<sub>2</sub>], hit  $m[1]$   
 7 [0111<sub>2</sub>],  
 8 [1000<sub>2</sub>],  
 0 [0000<sub>2</sub>]

*Handwritten notes:* A red arrow points from the '1' in the address [0001<sub>2</sub>] to the '1' in the block index  $m[1]$ . A red arrow points from the '1' in the address [0111<sub>2</sub>] to the '1' in the block index  $m[1]$ . A red arrow points from the '0' in the address [0000<sub>2</sub>] to the '0' in the block index  $m[0]$ . A red arrow points from the '0' in the address [0000<sub>2</sub>] to the '0' in the block index  $m[0]$ .

	v	Tag	Block
Set 0	1	0	$m[0-1]$
Set 1			
Set 2			
Set 3			

# Practice Problem

Consider the following code that runs on a system with a cache of the form  $(S,E,B,m) = (512,1,32,32)$



```
int array[4096];  
for (i=0; i<4096; i++)  
    sum += array[i];
```

Assuming sequential allocation of the integer array.

What is the maximum number of integers from the array that are stored in the cache at any point in time?

$$B = 32 = 8 \text{ int (4 Byte/integer)}$$

$$\# \text{ int in cache} = 8 \text{ int/block} \times 1 \text{ Block/set} \times 512 \text{ set} = 4096$$

# Today

- ◎ Case study: direct mapped cache

- ◎ Case study: set associative cache

- ◎ Case study: fully associative cache

- ◎ More discussions on Cache

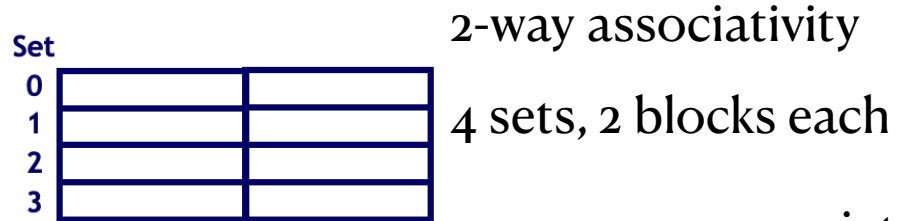
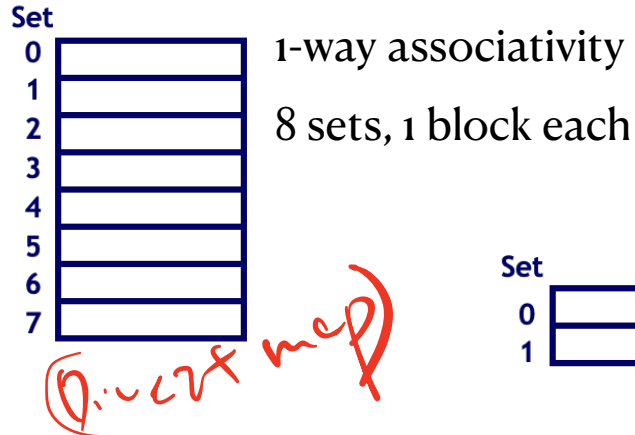
# Recall: Cache Parameters

Question: What different values of S, E, B imply?

- E = 1 (Direct mapped caches) “assigned seating”
- S = 1 (Fully associative caches) “open seating”
  - Cache has a single set; a memory address can map to any cache line.
- General cases are called **Set associative** caches. “assigned coach”
  - A memory address can map to any cache line within a fixed set.

# Set Associative Caches

- Each memory address is assigned to a particular set in the cache, but not to a specific block in the set.
  - Advantage: Reduce conflict misses
- Each set can store multiple distinct blocks/lines
  - If each set has E blocks, the cache is called an E-way associative cache
  - Set sizes range from 1 (direct-mapped) to whole cache (fully associative)



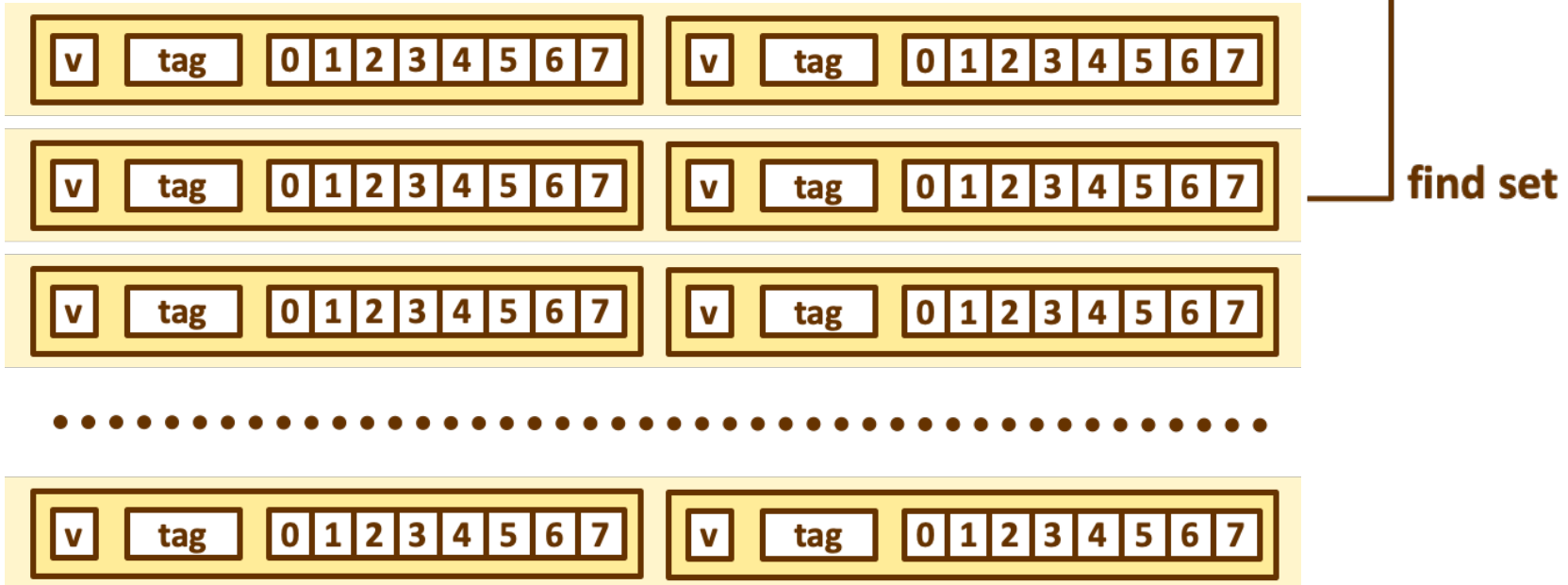


# E-Way Set Associative Cache (E=2)

E = 2: Two lines per set

Cache block size 8 bytes *B*

Address of short int:



# E-Way Set Associative Cache (E=2)

E = 2: Two lines per set

Cache block size 8 bytes

Address of short int:

t bits	0...01	100
--------	--------	-----

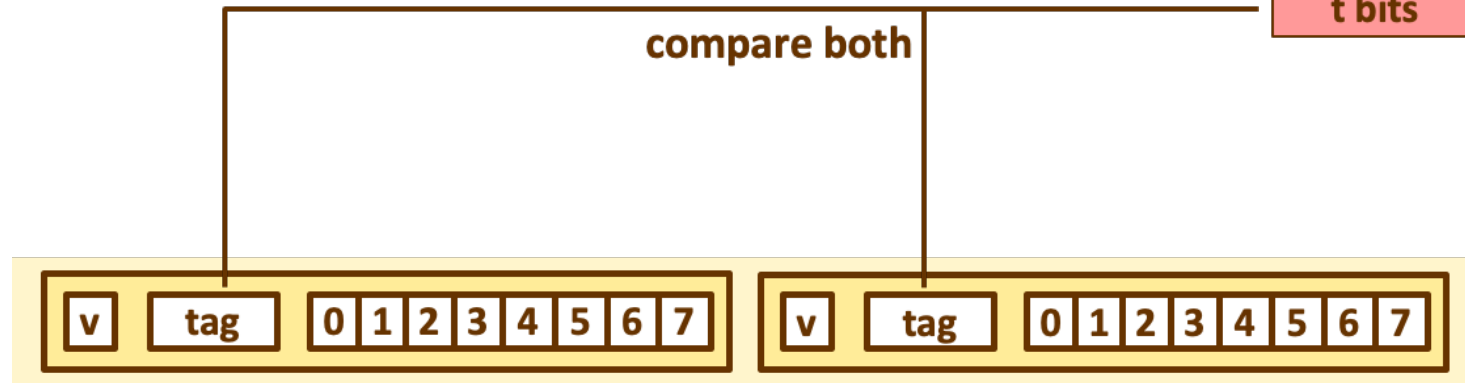


# E-Way Set Associative Cache (E=2)

E = 2: Two lines per set

Cache block size 8 bytes

Address of short int:

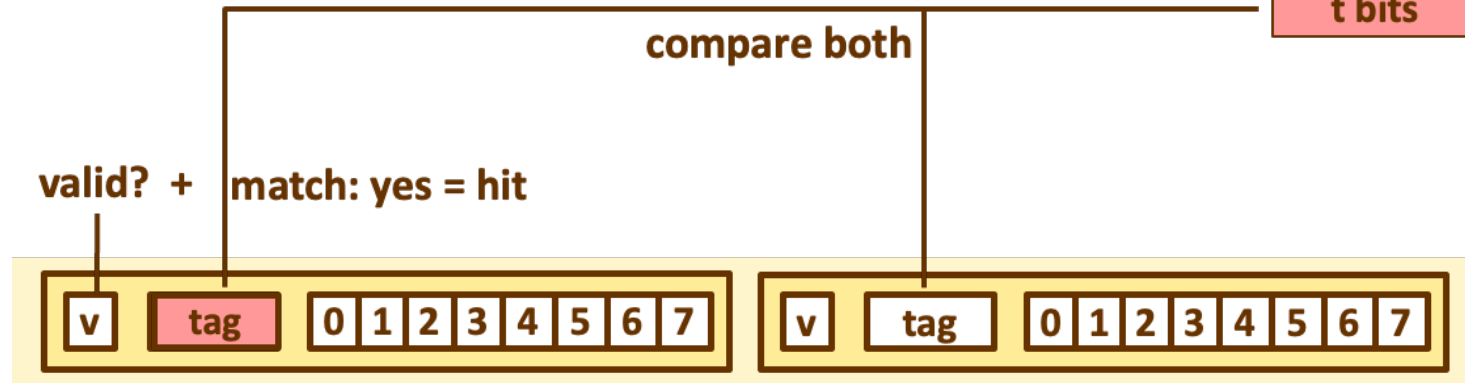


# E-Way Set Associative Cache (E=2)

E = 2: Two lines per set

Cache block size 8 bytes

Address of short int:

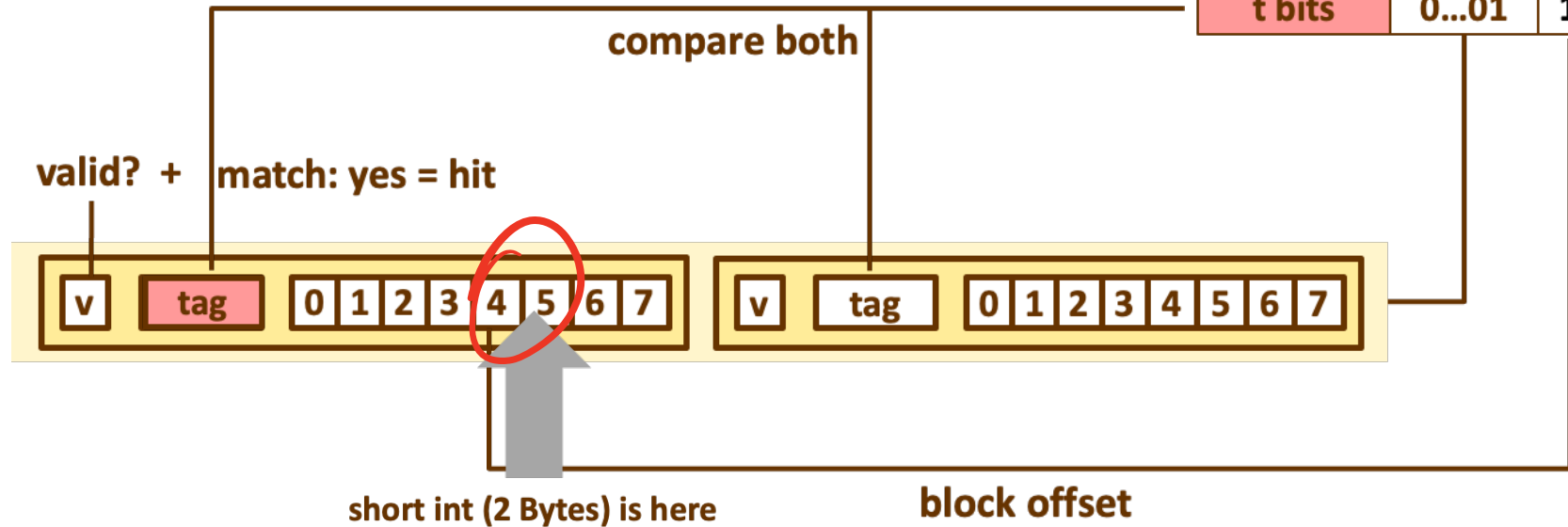


# E-Way Set Associative Cache (E=2)

E = 2: Two lines per set

Cache block size 8 bytes

Address of short int:

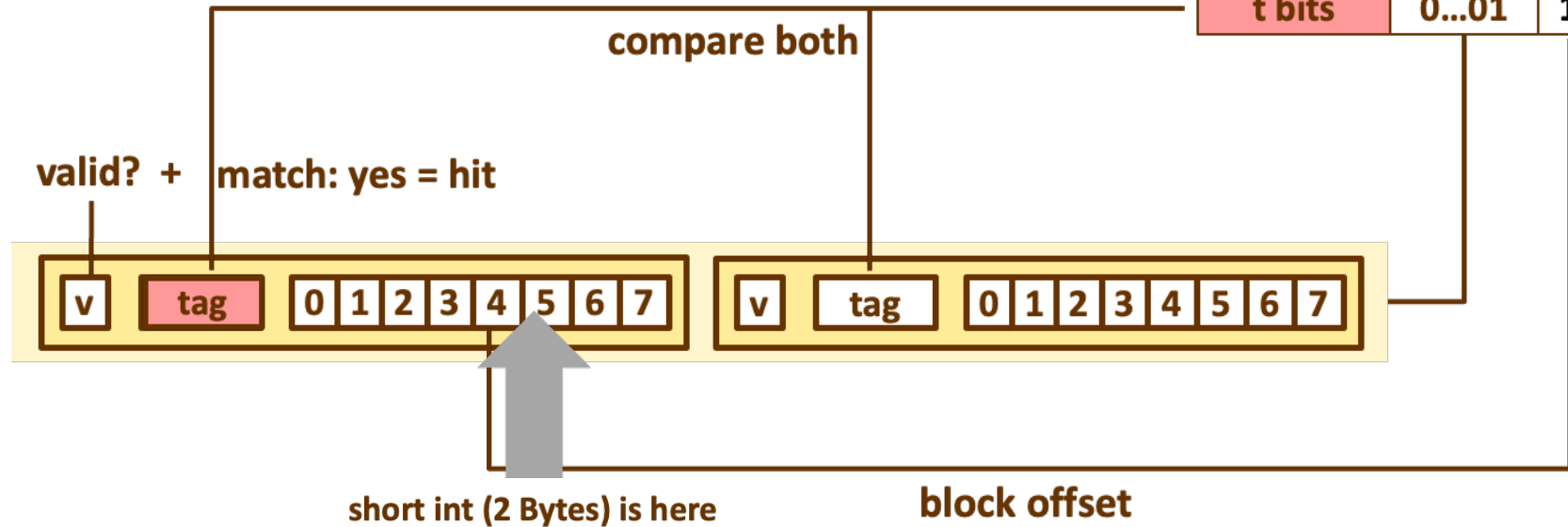


# E-Way Set Associative Cache (E=2)

E = 2: Two lines per set

Cache block size 8 bytes

Address of short int:



No match? One line in set is selected for eviction and replacement  
Replacement policies: random, least recently used (LRU), ...

# 2-Way Set Associative Cache Simulation

t=2	s=1	b=1
XX	X	X

$m = 4$

M=16 byte addresses, B=2 bytes/block,

S=2 sets, E=2 Blocks/set

Address trace (reads, one byte per read):

- 0 [00002], miss
- 1 [00012], hit M[1]
- 7 [01112], miss, M[7]
- 8 [10002], 10 ≠ 00 miss
- 0 [00002]

	v	Tag	Block
Set 0	1	00	M[0-1]
	1	10	M[8-9]
Set 1	1	01	M[6-7]

# Practice Problem 2

Consider a 2-way set associative cache  $(S,E,B,m) = (8,2,4,13)$

- Excluding the overhead of the tags and valid bits, what is the capacity of this cache?

$$C = S \times E \times B = 8 \times 2 \times 4 = 64$$



# Practice Problem 2

Consider a 2-way set associative cache  $(S,E,B,m) = (8,2,4,13)$

- ⊙ Excluding the overhead of the tags and valid bits, what is the capacity of this cache? 64
- ⊙ Draw a figure of this cache

# Practice Problem 2

Consider a 2-way set associative cache  $(S,E,B,m) = (8,2,4,13)$

- Excluding the overhead of the tags and valid bits, what is the capacity of this cache?
- Draw a figure of this cache


Set	Tag	Data0-3	Tag	Data0-3
0				
1				
2				
3				
4				
5				
6				
7				

# Practice Problem 2

Consider a 2-way set associative cache  $(S,E,B,m) = (8,2,4,13)$

- Excluding the overhead of the tags and valid bits, what is the capacity of this cache?
- Draw a figure of this cache

Set	Tag	Data0-3	Tag	Data0-3
0				
1				
2				
3				
4				
5				
6				
7				

$m = 13$   
  
 8 bits    3 bits    2 bits  
 $s = 3$   
 $b = 2$   
 $t = 13 - (s + b)$   
 $= 8$

- Draw a diagram that shows the parts of the address that are used to determine the cache tag, cache set index, and cache block offset

# Practice Problem 2

Consider a 2-way set associative cache  $(S,E,B,m) = (8,2,4,13)$

- Excluding the overhead of the tags and valid bits, what is the capacity of this cache?
- Draw a figure of this cache

Set	Tag	Data0-3	Tag	Data0-3
0				
1				
2				
3				
4				
5				
6				
7				

- Draw a diagram that shows the parts of the address that are used to determine the cache tag, cache set index, and cache block offset



# Practice Problem 3



Consider a 2-way set associative cache (S,E,B,m) = (8,2,4,13)

Note: Invalid cache lines are left blank

Consider an access to 0x0E34.

*011100110100* *binary*

- What is the block offset of this address?

*00 → 0*

- What is the set index of this address?

*101 → 5*

- What is the cache tag of this address?

*01110001* *71*

- Does this access hit or miss in the cache?

*Hit*

- What value is returned if it is a hit?

Set	Tag	Data0-3	Tag	Data0-3
0	09	86 30 3F 10	00	
1	45	60 4F E0 23	38	00 BC 0B 37
2	EB		0B	
3	06		32	12 08 7B AD
4	C7	06 78 07 C5	05	40 67 C2 3B
5	71	0B DE 18 4B	6E	
6	91	A0 B7 26 2D	F0	
7	46		DE	12 C0 88 37

# Today

- ◎ Case study: direct mapped cache

- ◎ Case study: set associative cache

- ◎ Case study: fully associative cache

- ◎ More discussions on Cache

# Fully Associative Cache (S=1)

- ⦿ A fully associative cache permits data to be stored in any cache block, instead of forcing each memory address into one particular block.
  - There is only 1 set (i.e.  $S=1$  and  $s=0$ )  $C = E * B$
- ⦿ When data is fetched from memory, it can be placed in any unused block of the cache.
- ⦿ Eliminates conflict misses between two or more memory addresses which map to a single cache block.

# Fully Associative Cache: Example

Consider the following fully associative cache:  $(S,E,B,m) = (1,4,1,4)$

- Derive values for number of address bits used for the tag (t), the index (s) and the block offset (b)

$$s = 0$$

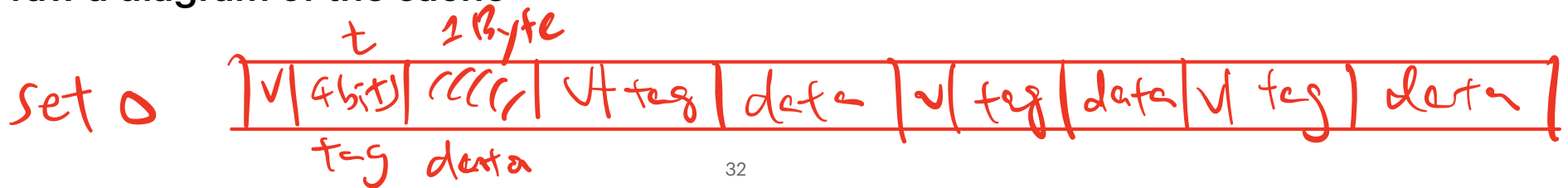
$$b = 0$$

$$t = 4$$

- Draw a diagram of which bits of the address are used for the tag, the set index and the block offset



- Draw a diagram of the cache





# Fully Associative Cache: Example

Consider the following fully associative cache:  $(S,E,B,m) = (1,4,1,4)$

- Derive values for number of address bits used for the tag (t), the index (s) and the block offset (b)

$$s = 0$$

$$b = 0$$

$$t = 4$$

- Draw a diagram of which bits of the address are used for the tag, the set index and the block offset

**t (4 bits)**

- Draw a diagram of the cache

	<b>Tag</b>	<b>Data</b>	<b>Tag</b>	<b>Data</b>	<b>Tag</b>	<b>Data</b>	<b>Tag</b>	<b>Data</b>
<b>s=0</b>	<b>4 bits</b>	<b>1 byte</b>	<b>4 bits</b>	<b>1 byte</b>	<b>4 bits</b>	<b>1 byte</b>	<b>4 bits</b>	<b>1 byte</b>

# Fully Associative Cache: Example

Main memory

0000	0x00
0001	0x11
0010	0x22
0011	0x33
0100	0x44
0101	0x55
0110	0x66
0111	0x77
1000	0x88
1001	0x99
1010	0xAA
1011	0xBB
1100	0xCC
1101	0xDD
1110	0xEE
1111	0xFF

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

Tag	Data	Tag	Data	Tag	Data	Tag	Data
0000	0x00	0110	0x00	0001	0x11		

Access pattern

- 0000
- 0110
- 0001
- 0110
- 0010
- 0110
- ...

miss  
miss  
miss  
hit

miss rate = \_\_\_\_\_

t (4 bits)

# Fully Associative Cache: Example

## Main memory

0000	0x00
0001	0x11
0010	0x22
0011	0x33
0100	0x44
0101	0x55
0110	0x66
0111	0x77
1000	0x88
1001	0x99
1010	0xAA
1011	0xBB
1100	0xCC
1101	0xDD
1110	0xEE
1111	0xFF

**Cache** (S,E,B,m) = (1,4,1,4)

Tag	Data	Tag	Data	Tag	Data	Tag	Data

## Access pattern

0000

0110

0001

0110

0010

0110

...

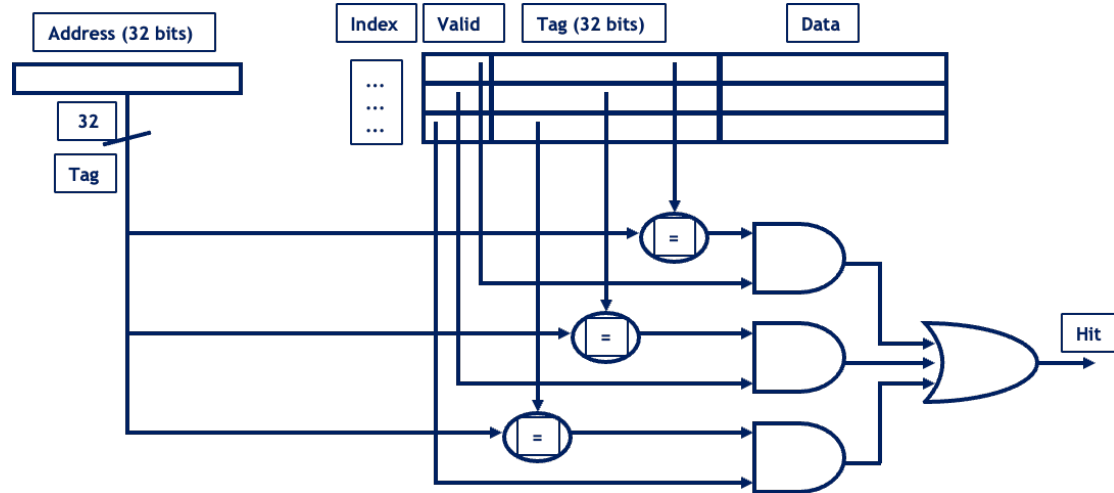
Data Miss rate after first two cold misses = 0 %

**t (4 bits)**

# The Price of Full Associativity

:( A fully associative cache is **expensive** to implement.

- No index field in the address anymore
- Entire address used as the tag, increasing the total cache size.
- Data could be anywhere in the cache, so we must check the tag of every cache block. That's **a lot of comparators!**



# Today

- ◎ Case study: direct mapped cache
- ◎ Case study: set associative cache
- ◎ Case study: fully associative cache
- ◎ More discussions on Cache

# Cache Line Replacement

Any empty block in the correct set may be used for storing data. If there are no empty blocks, the cache will attempt to replace one.

## Replacement Algorithms

- ◎ **Most common: Least Recently Used (LRU)**
  - If a block hasn't been used in a while, it's less likely needed again anytime soon.
- ◎ **First in first out (FIFO):** Replace block that has been in cache longest
- ◎ **Least frequently used:** Replace block which has had fewest hits
- ◎ **Random** *arbitrary*

# Data Writes in Memory Hierarchy

On a computer system, multiple copies of data exist, e.g. L1, L2, main, and disk.

**Question:** What happens when data is written to memory through cache?

**Answer:** Cache needs write policies as well.

- **Write-hit:** Block being written is in cache
- **Write-miss:** Block being written is NOT in cache

# Write-Hit Policies

## ◎ Write-through

- Writes go immediately to main memory (as well as cache)
- Can incur lots of traffic, slows down writes

## ◎ Write-back

- Writes initially made in cache only
- Defer write to memory until replacement of line
- Need a dirty bit (line different from memory or not)
- Can cause inconsistency among caches



# Write-Miss Policies

## ◎ Write-allocate

- When write to address misses, load into cache
- Update line in cache
- Good if more writes to the location follow

## ◎ No-write-allocate

- When write to address misses, write straight to memory. Do not load into cache!
- Good if there are no subsequent reads and writes to address

## Typically

- Write-through + No-write-allocate
- Write-back + Write-allocate

# Cache Design Summary

- ◎ Cache size
- ◎ Associativity
- ◎ Block size
- ◎ Replacement algorithms
- ◎ Write policies
- ◎ Number of caches