[Chap.6-3] The Memory Hierarchy

Young Ik Eom (yieom@skku.edu, 031-290-7120)
Distributing Computing Laboratory
Sungkyunkwan University
http://dclab.skku.ac.kr

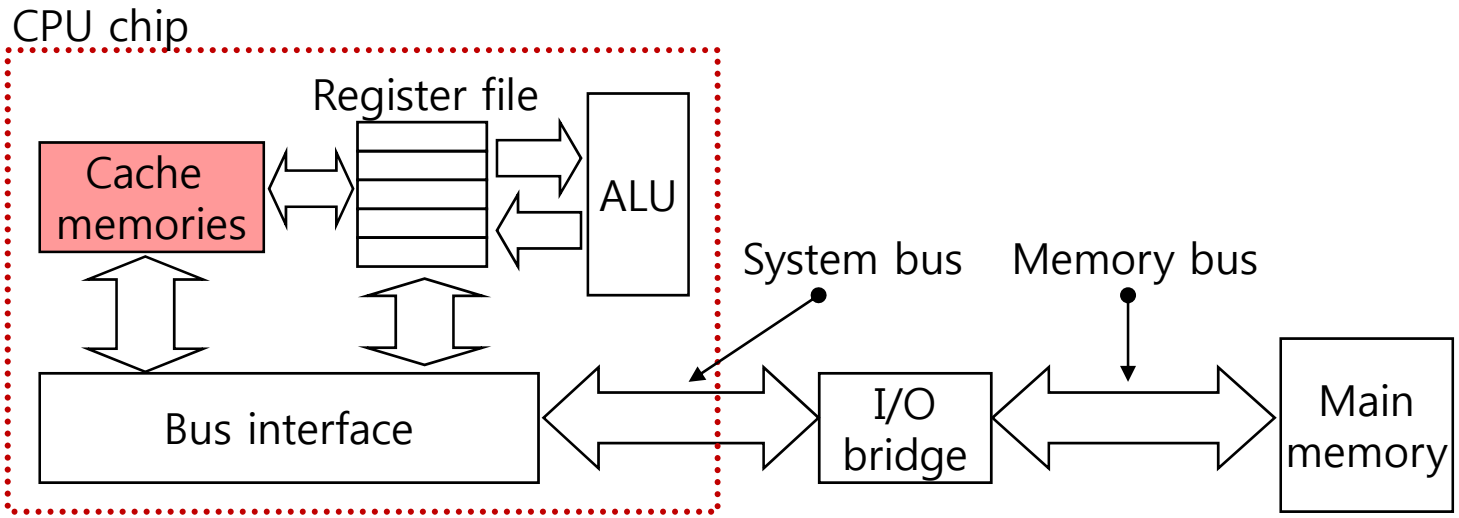
SKK UNIVERSITY

Contents

- Storage technologies
- Locality
- Memory hierarchy
- Cache memories
- Writing cache-friendly code
- Impact of caches on program performance

Cache Memories

Typical system structure



- Access times
  - L1 cache (1~4 clock cycles)
  - L2 cache (about 10 clock cycles)
  - L3 cache (about 50 clock cycles)

Cache Memories

Generic cache organization

- Assume a memory of $M = 2^m$ addresses
- Cache (S, E, B, m) → 메모리 주소 지정을 위한 bit 수
  - $S = 2^s$ cache sets
  - $E = 2^e$ cache lines in each set
  - A data block of $B = 2^b$ bytes in each line

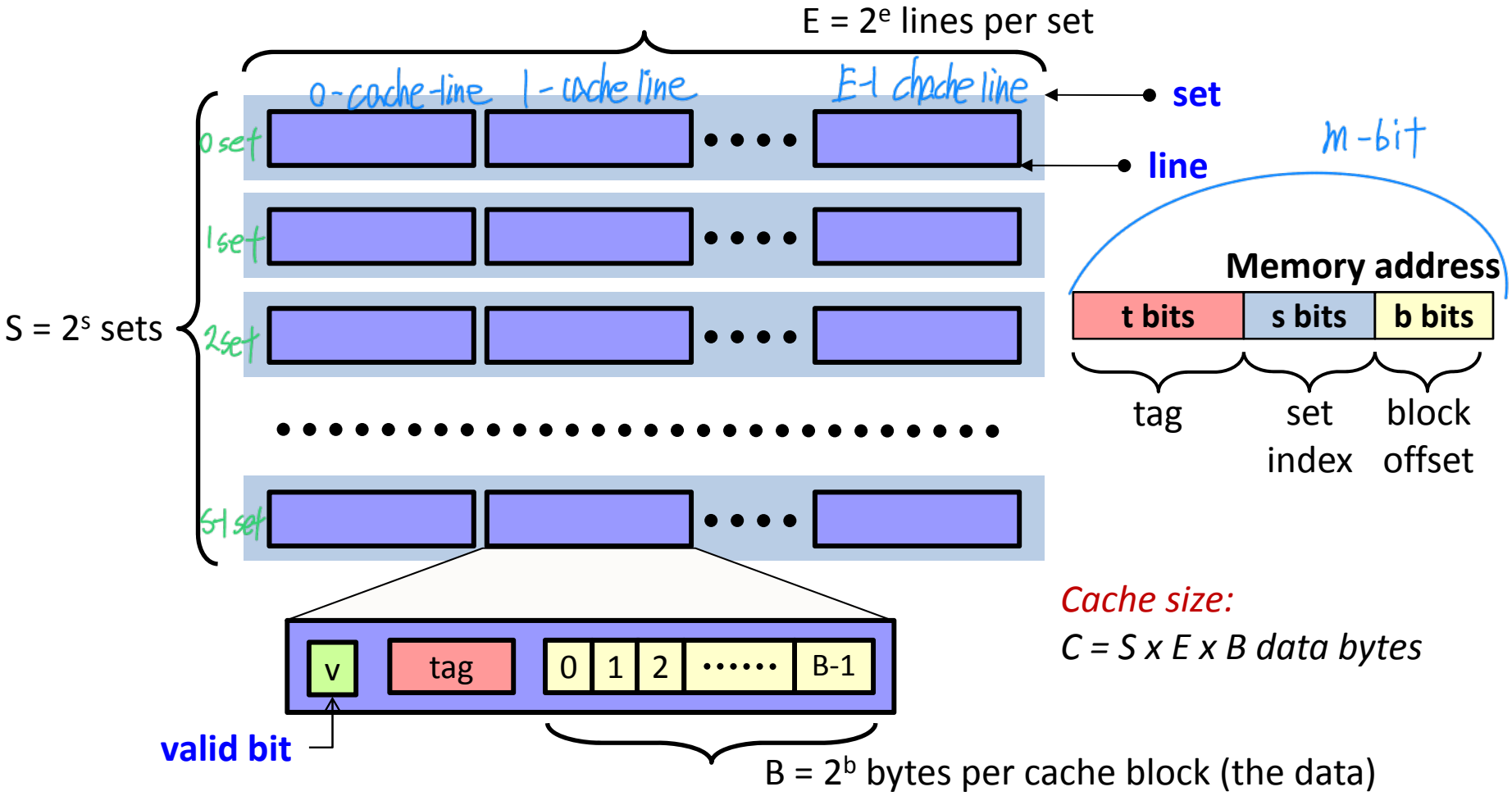
각 cache line에 부여되는 정보

- Valid bit
  - ✓ Indicates whether or not the line contains meaningful information
- Tag bit ($t = m - b - s$)
  - ✓ Identifies the block stored in the cache line

- Cache size $C = S \times E \times B$

Cache Memories

Generic cache organization

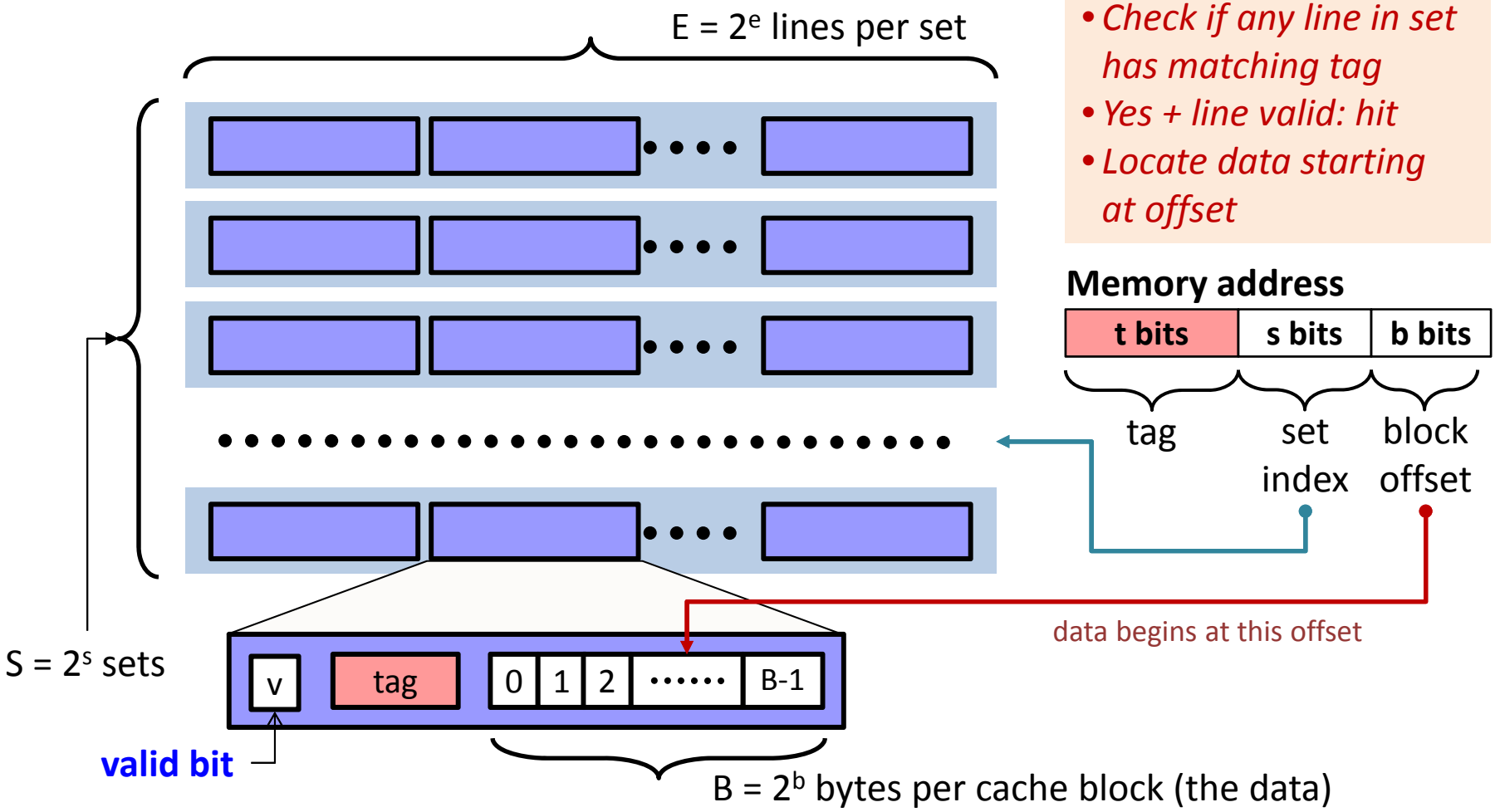


Cache size:

C = S x E x B data bytes

Cache Memories

Cache read

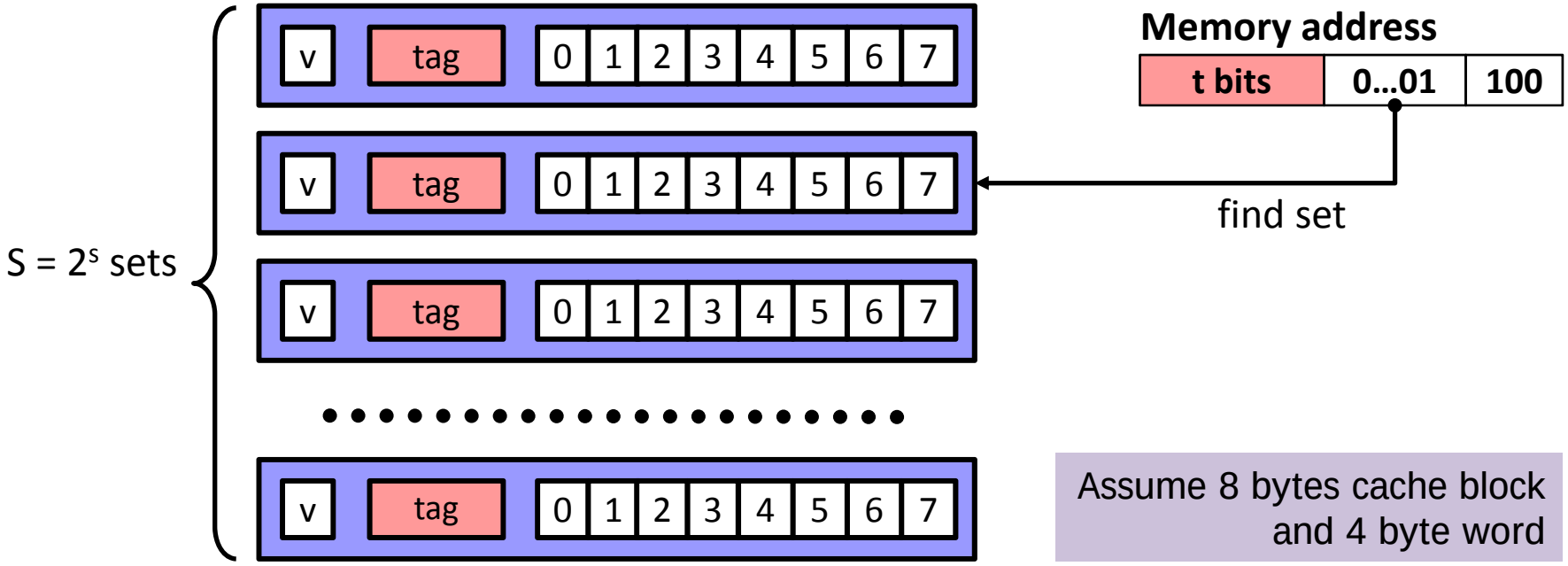


- *Locate set*
- *Check if any line in set has matching tag*
- *Yes + line valid: hit*
- *Locate data starting at offset*

Cache Memories

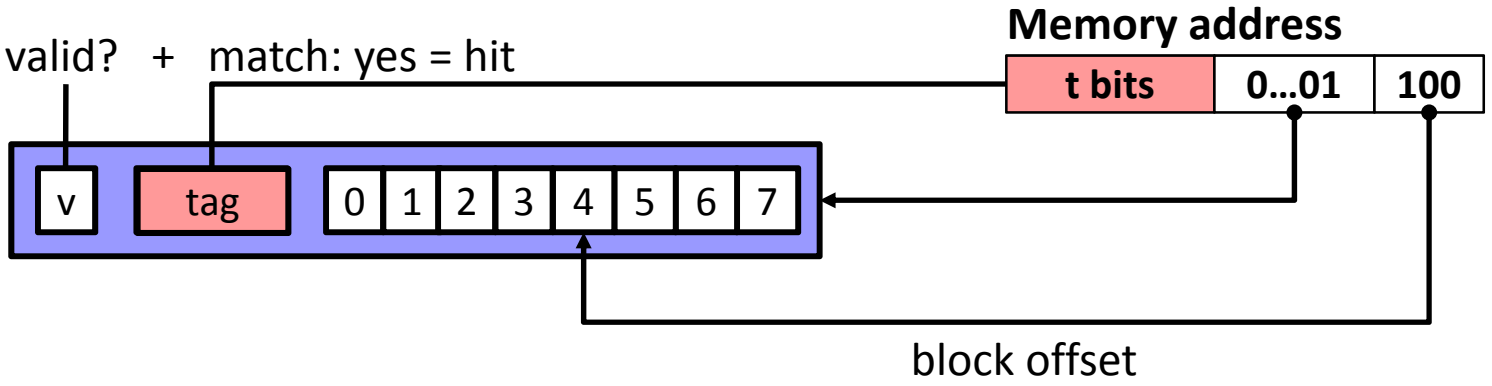
Direct mapped cache

- 1 line per set (E = 1)



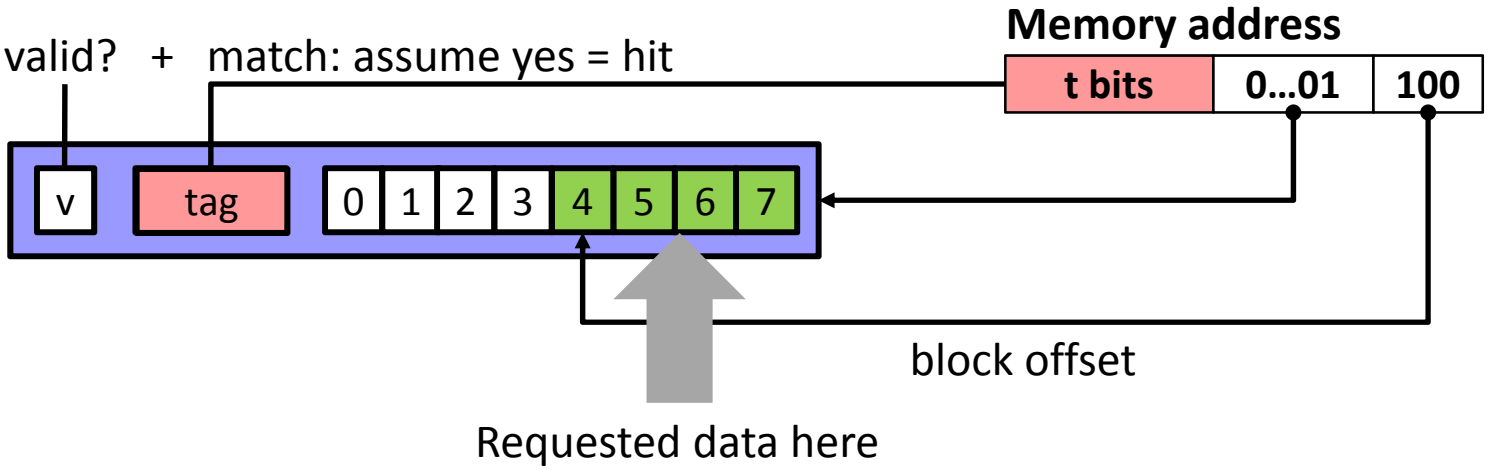
Cache Memories

Direct mapped cache



Cache Memories

Direct mapped cache



When no match, old line is evicted and replaced

Cache Memories

Direct mapped cache

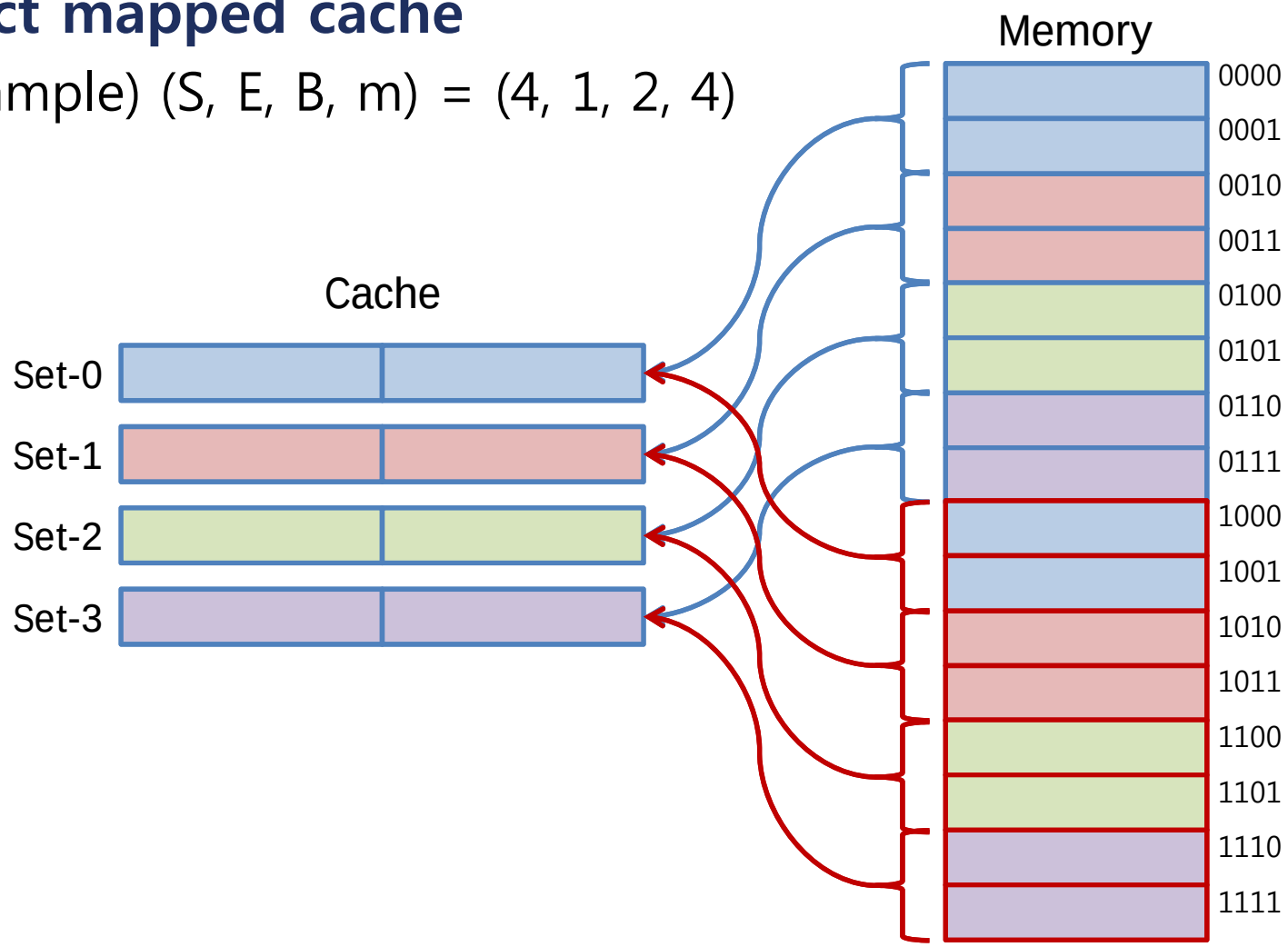
- Example) (S, E, B, m) = (4, 1, 2, 4)
  - 4 sets, 1 line per set, 2 bytes per block, 4-bit addresses
  - Assume each word is a single byte
  - 4-bit address space →

Address (decimal)	Address bits: Tag bits ($t = 1$)	Address bits: Index bits ($s = 2$)	Address bits: 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

Cache Memories

Direct mapped cache

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



Cache Memories

Direct mapped cache

- The concatenation of the tag and set index bits uniquely identifies each block in memory
  - 000x into B0, 001x into B1, …
- Multiple blocks can map into the same cache set
  - B0 and B4 into set-0, B1 and B5 into set-1, …
- Blocks that map to the same cache set are uniquely identified by the tag
  - B0 with tag 0 and B4 with tag 1,
    B1 with tag 0 and B5 with tag 1, …

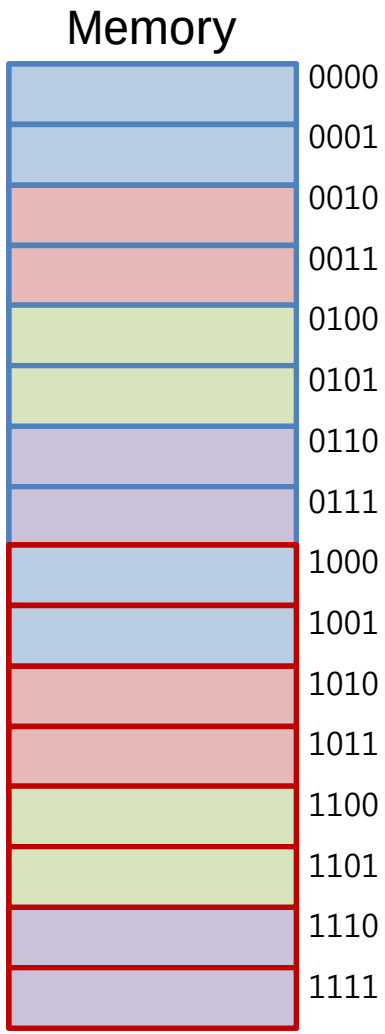
Cache Memories

Direct mapped cache

- A scenario for a sequence of reads

- Initial cache (empty)

Set	Valid	Tag	Byte 0	Byte 1
0	0			
1	0			
2	0			
3	0			



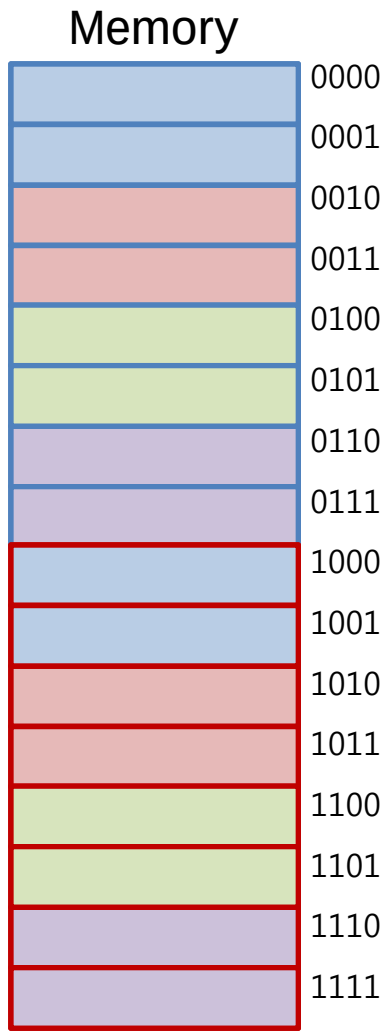
Cache Memories

Direct mapped cache

- A scenario for a sequence of reads

- Read word at address 0 (cache miss)

Set	Valid	Tag	Byte 0	Byte 1
0	0			
1	0			
2	0			
3	0			



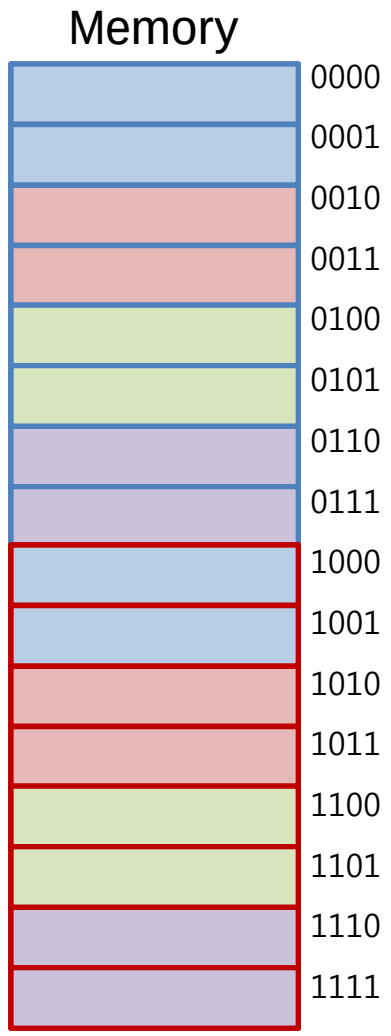
Cache Memories

Direct mapped cache

- A scenario for a sequence of reads

- Read word at address 0 (cache miss)

Set	Valid	Tag	Byte 0	Byte 1
0	1	0	m[0]	m[1]
1	0			
2	0			
3	0			



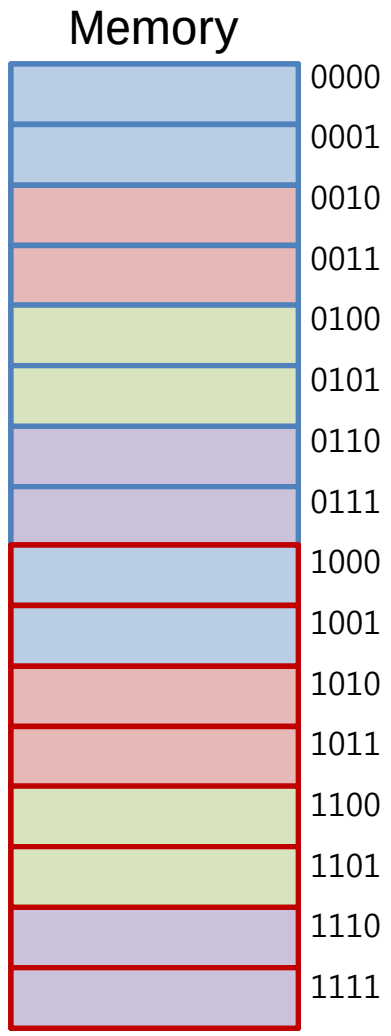
Cache Memories

Direct mapped cache

- A scenario for a sequence of reads

- Read word at address 1 (cache hit)

Set	Valid	Tag	Byte 0	Byte 1
0	1	0	m[0]	m[1]
1	0			
2	0			
3	0			



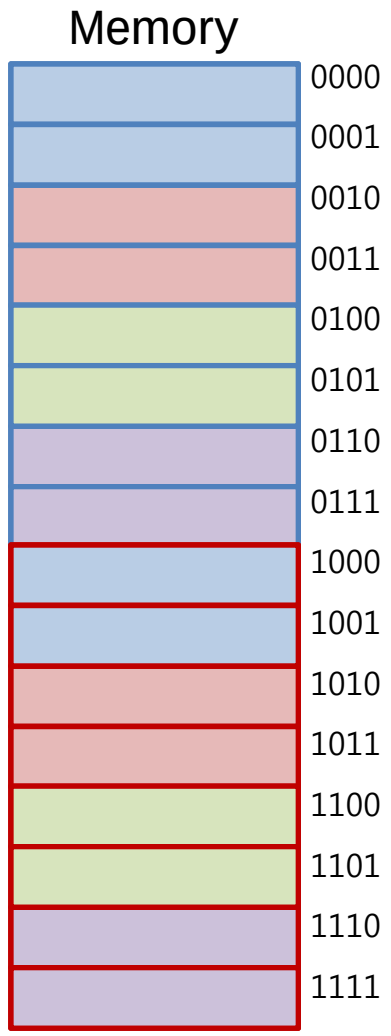
Cache Memories

Direct mapped cache

- A scenario for a sequence of reads

- Read word at address 13 (cache miss)

Set	Valid	Tag	Byte 0	Byte 1
0	1	0	m[0]	m[1]
1	0			
2	0			
3	0			



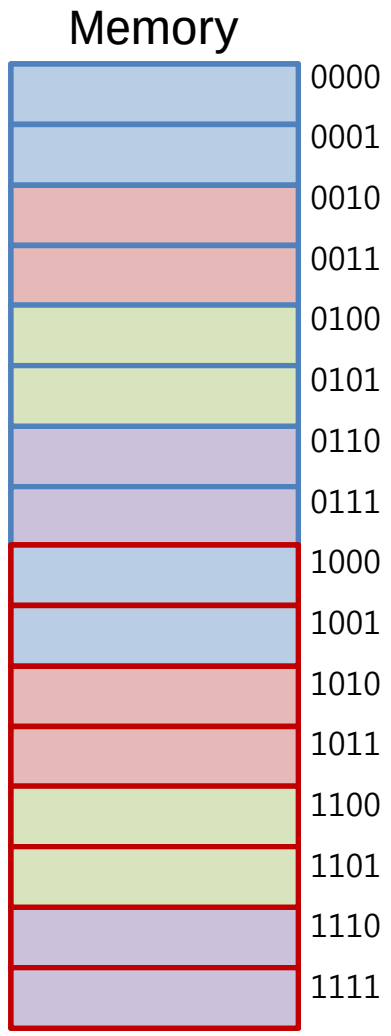
Cache Memories

Direct mapped cache

- A scenario for a sequence of reads

- Read word at address 13 (cache miss)

Set	Valid	Tag	Byte 0	Byte 1
0	1	0	m[0]	m[1]
1	0			
2	1	1	m[12]	m[13]
3	0			



Cache Memories

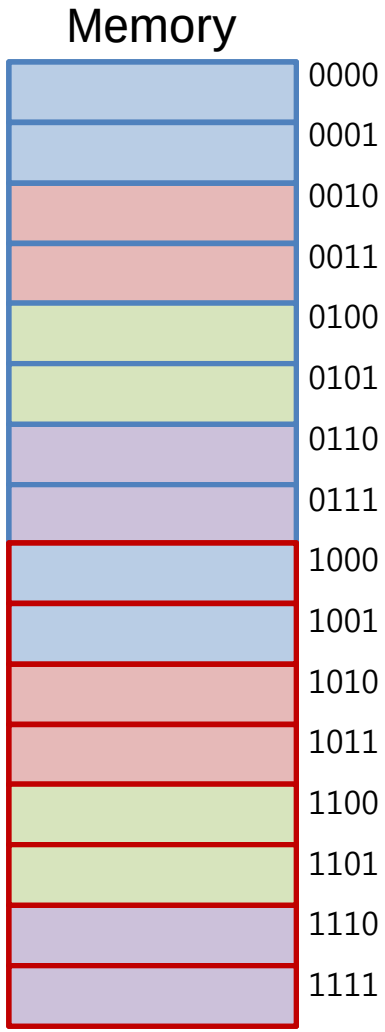
Direct mapped cache

- A scenario for a sequence of reads

No tag match

- Read word at address 8 (cache miss)

Set	Valid	Tag	Byte 0	Byte 1
0	1	0	m[0]	m[1]
1	0			
2	1	1	m[12]	m[13]
3	0			



Cache Memories

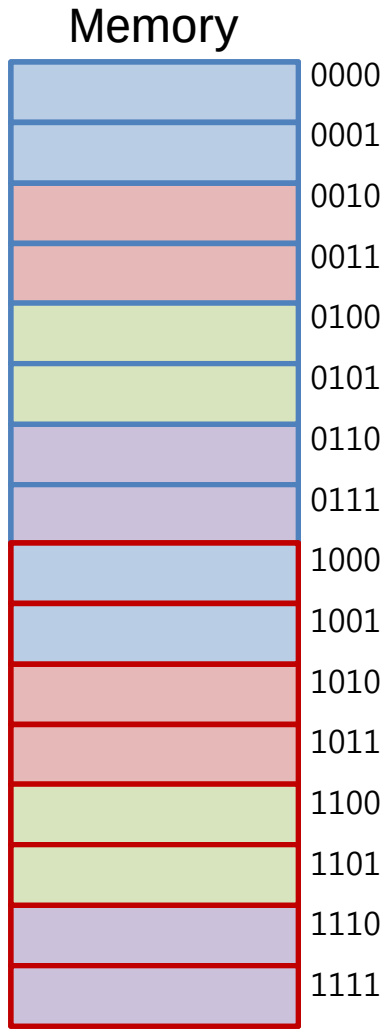
Direct mapped cache

- A scenario for a sequence of reads

No tag match

- Read word at address 8 (cache miss)

Set	Valid	Tag	Byte 0	Byte 1
0	1	1	m[8]	m[9]
1	0			
2	1	1	m[12]	m[13]
3	0			



Cache Memories

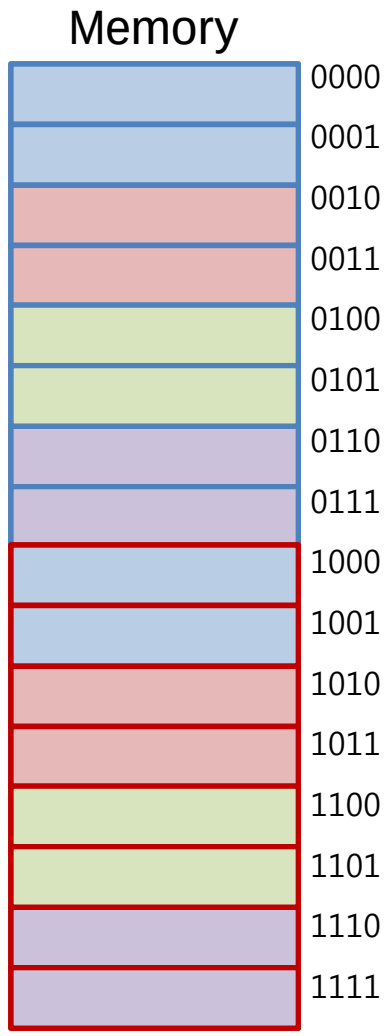
Direct mapped cache

- A scenario for a sequence of reads

No tag match

- Read word at address 0 (cache miss)

Set	Valid	Tag	Byte 0	Byte 1
0	1	1	m[8]	m[9]
1	0			
2	1	1	m[12]	m[13]
3	0			



Cache Memories

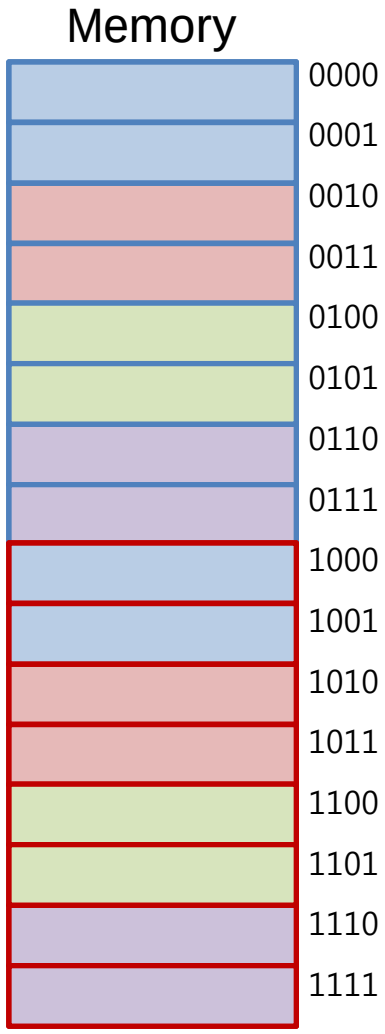
Direct mapped cache

- A scenario for a sequence of reads

No tag match

- Read word at address 0 (cache miss)

Set	Valid	Tag	Byte 0	Byte 1
0	1	0	m[0]	m[1]
1	0			
2	1	1	m[12]	m[13]
3	0			



Cache Memories

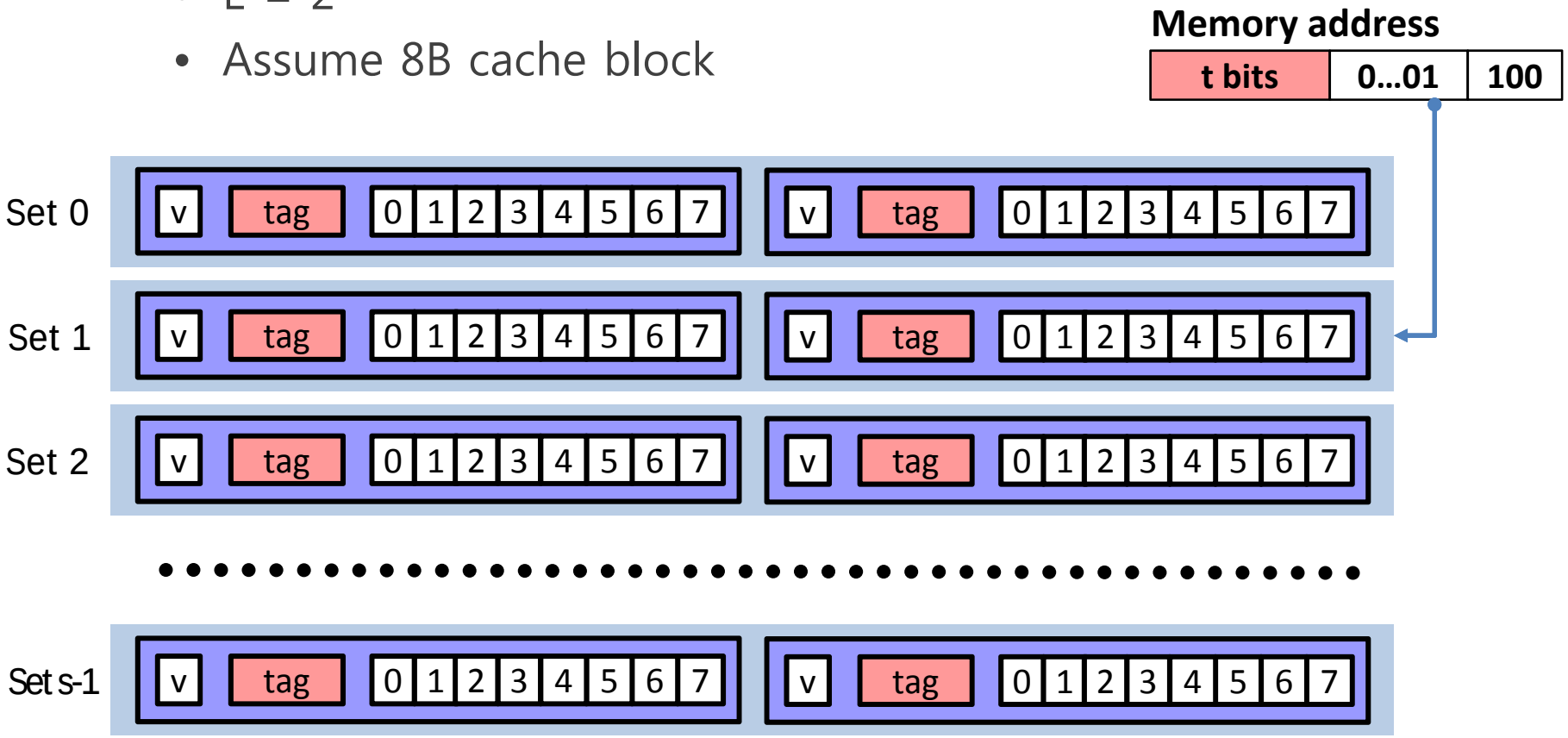
Set associative cache

- Each set holds two or more cache lines
  (1 < E < C/B) → E-way set associative cache
- When E = C/B, fully associative cache

Cache Memories

Set associative cache

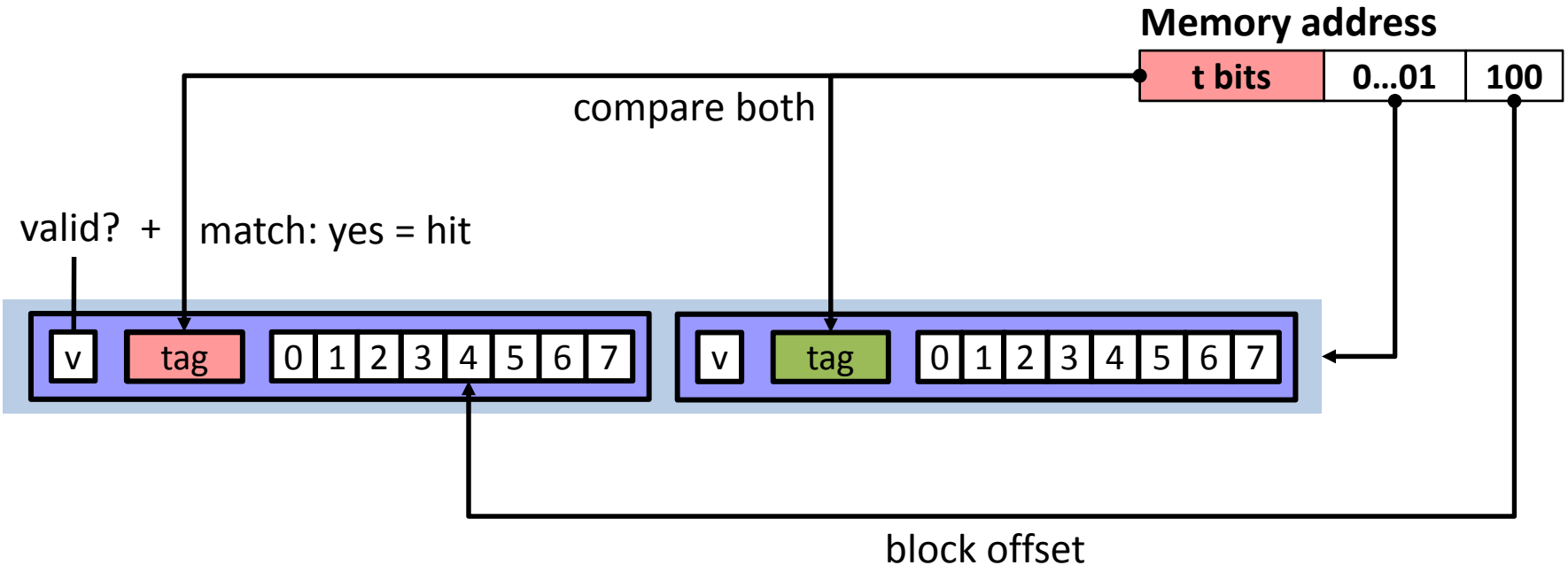
- Example) 2-way set associative cache
  - E = 2
  - Assume 8B cache block



Cache Memories

Set associative cache

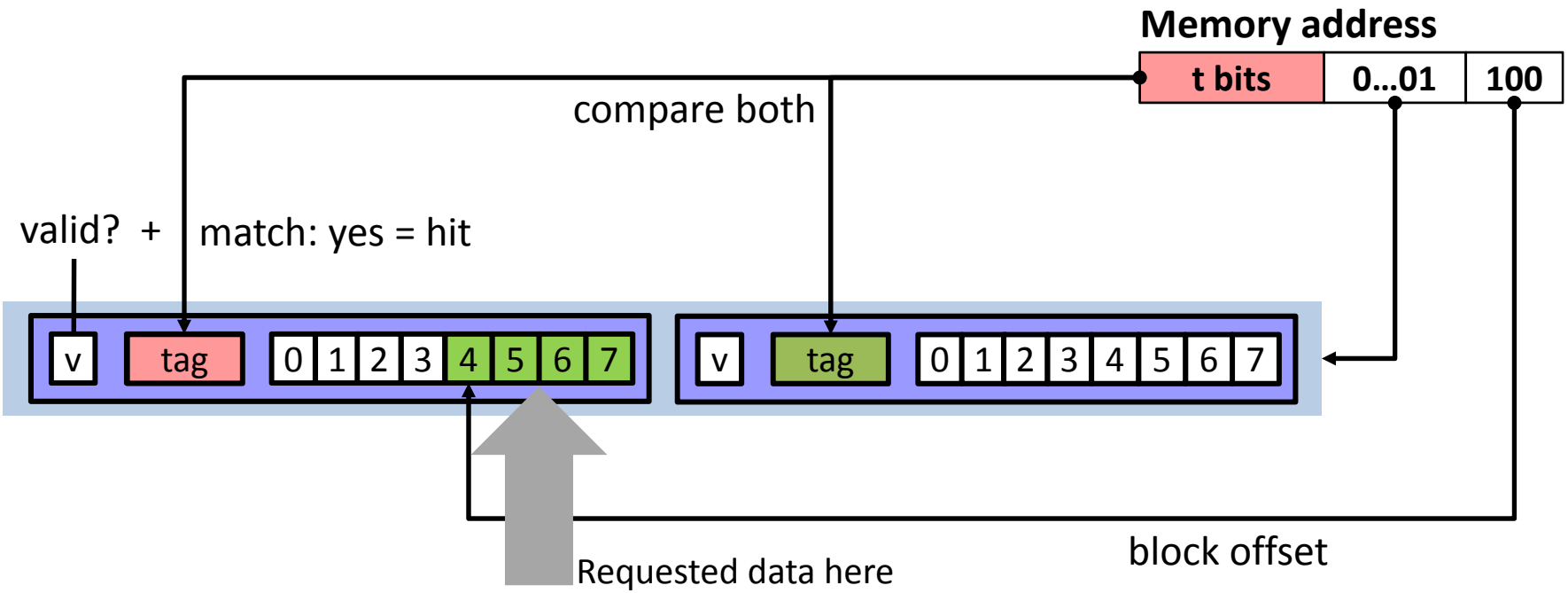
- Example) 2-way set associative cache



Cache Memories

Set associative cache

- Example) 2-way set associative cache

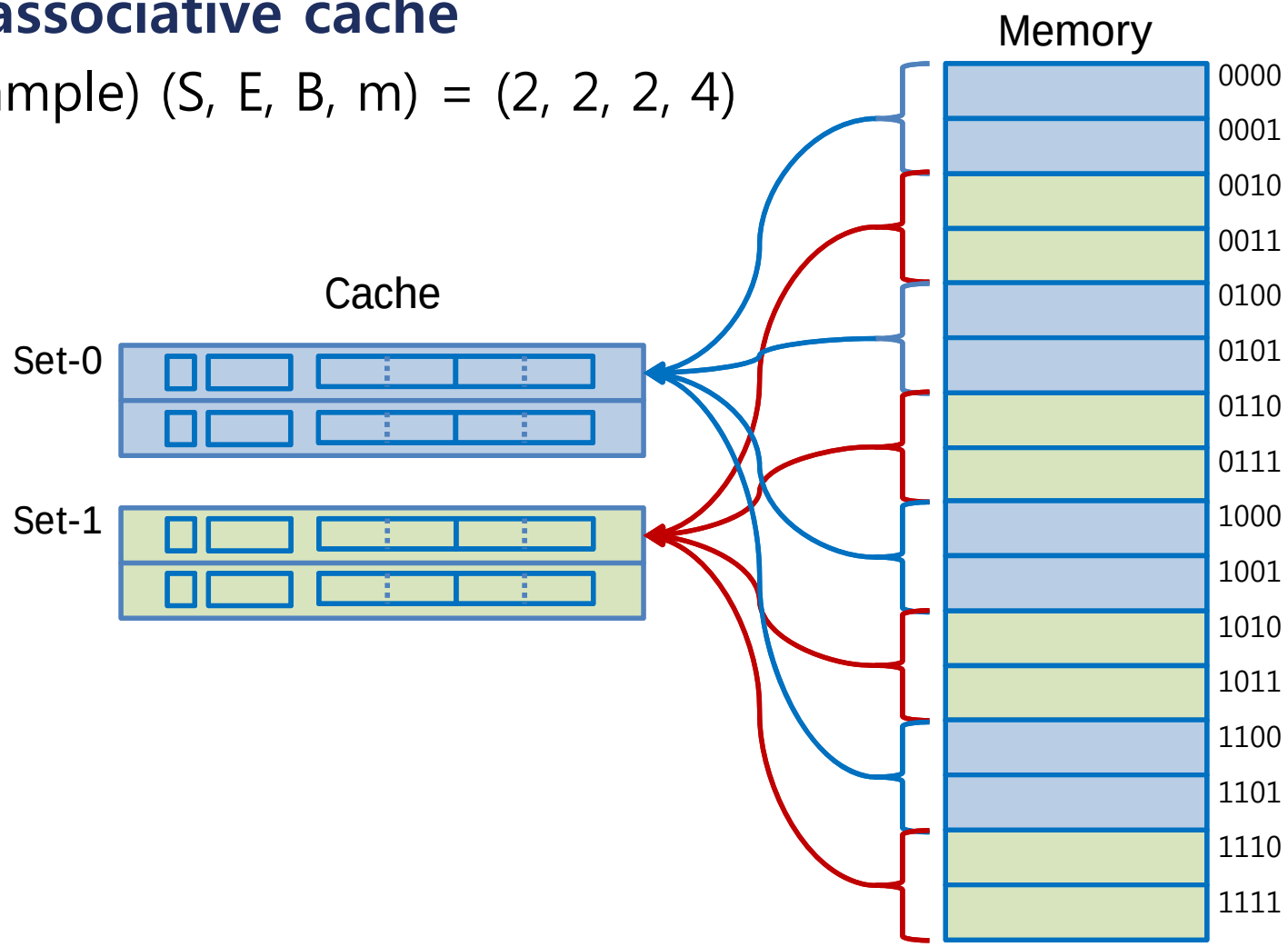


When no match,

- One line in the set is selected for eviction and replaced
- Replacement policies: random, LRU, …

Cache Memories

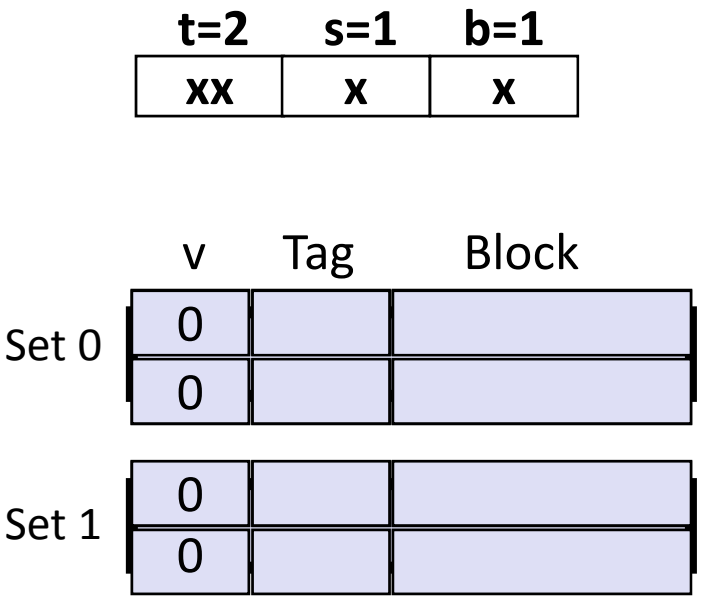
- **Set associative cache**
  - Example) (S, E, B, m) = (2, 2, 2, 4)



Cache Memories

Set associative cache

- A scenario for a sequence of reads



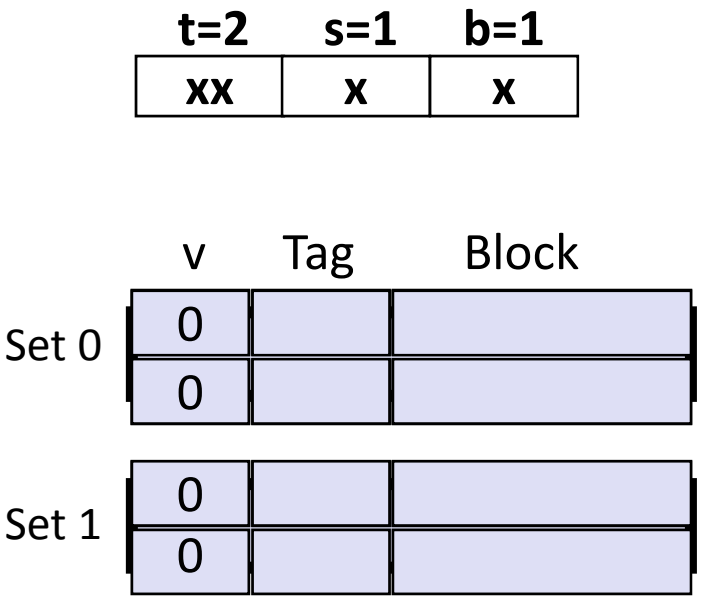
M = 16 bytes, B = 2 bytes, S = 2 sets,
E = 2 blocks/set

Address trace (reads, one byte per read)

Cache Memories

Set associative cache

- A scenario for a sequence of reads



M = 16 bytes, B = 2 bytes, S = 2 sets,
E = 2 blocks/set

Address trace (reads, one byte per read)

0 [0000$_2$], miss

Cache Memories

Set associative cache

- A scenario for a sequence of reads

t=2	s=1	b=1
xx	x	x

	v	Tag	Block
Set 0	1	00	M[0-1]
	0		
Set 1	0		
	0		

M = 16 bytes, B = 2 bytes, S = 2 sets,
E = 2 blocks/set

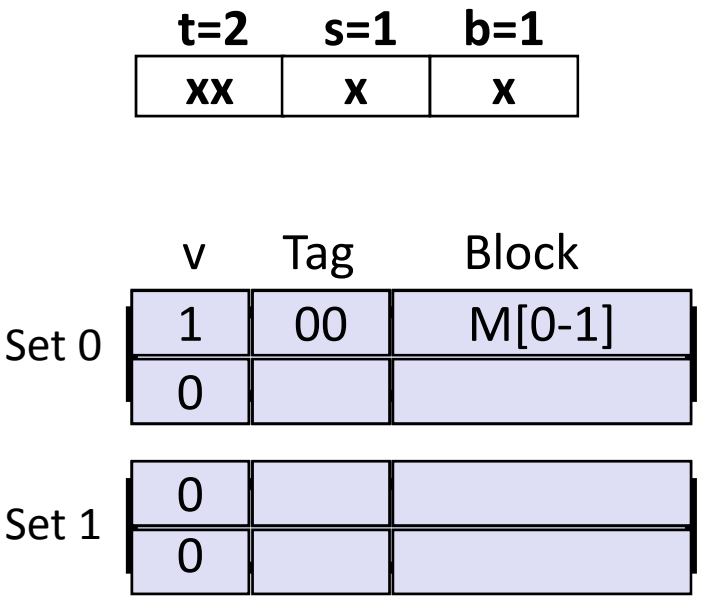
Address trace (reads, one byte per read)

0 [0000$_2$], miss

Cache Memories

Set associative cache

- A scenario for a sequence of reads



M = 16 bytes, B = 2 bytes, S = 2 sets,
E = 2 blocks/set

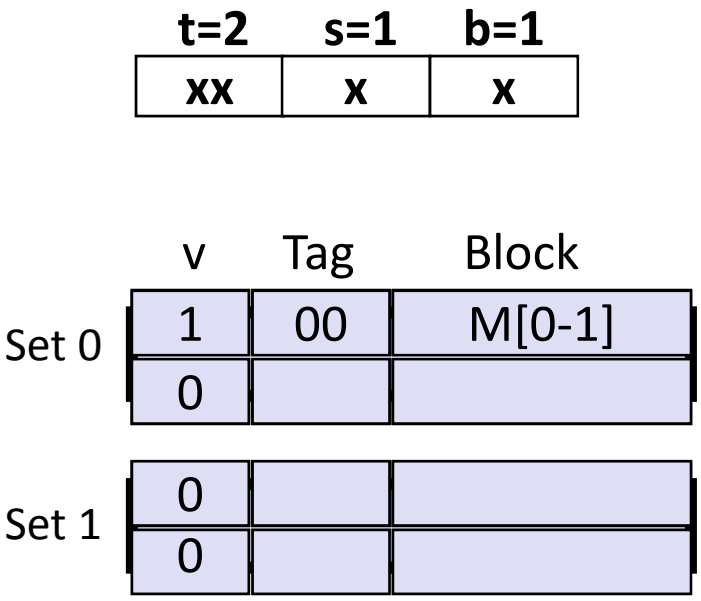
Address trace (reads, one byte per read)

0	[00$\underline{0}$0$_2$],	miss
1	[00$\underline{0}$1$_2$],	hit

Cache Memories

Set associative cache

- A scenario for a sequence of reads



M = 16 bytes, B = 2 bytes, S = 2 sets,
E = 2 blocks/set

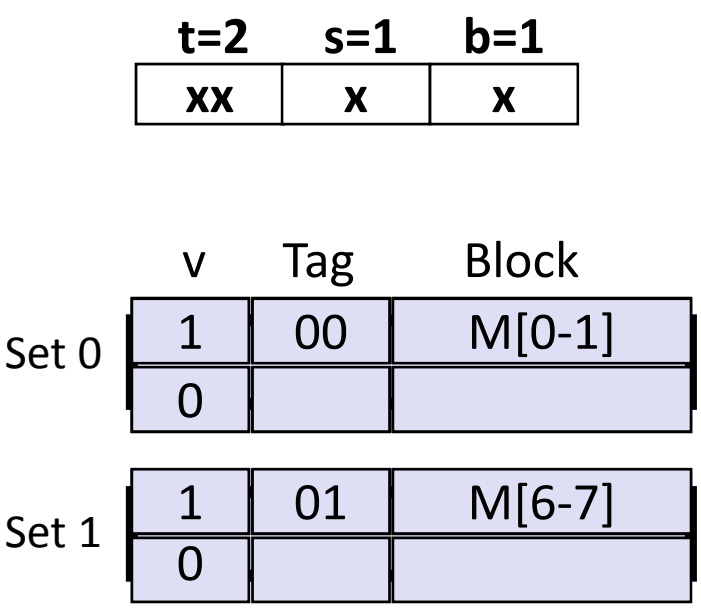
Address trace (reads, one byte per read)

0	[0000$_2$],	miss
1	[0001$_2$],	hit
7	[0111$_2$],	miss

Cache Memories

Set associative cache

- A scenario for a sequence of reads



M = 16 bytes, B = 2 bytes, S = 2 sets,
E = 2 blocks/set

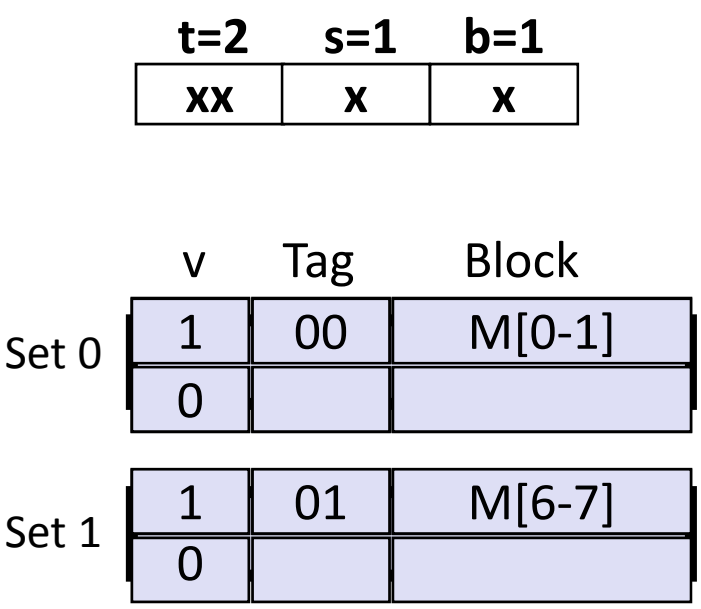
Address trace (reads, one byte per read)

0	[000$_2$],	miss
1	[001$_2$],	hit
7	[011$_2$],	miss

Cache Memories

Set associative cache

- A scenario for a sequence of reads



M = 16 bytes, B = 2 bytes, S = 2 sets,
E = 2 blocks/set

Address trace (reads, one byte per read)

0	[00$\underline{0}$0$_2$],	miss
1	[00$\underline{0}$1$_2$],	hit
7	[01$\underline{1}$1$_2$],	miss
8	[10$\underline{0}$0$_2$],	miss

Cache Memories

Set associative cache

- A scenario for a sequence of reads

t=2	s=1	b=1
xx	x	x

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

M = 16 bytes, B = 2 bytes, S = 2 sets,
E = 2 blocks/set

Address trace (reads, one byte per read)

0	[00$\underline{0}$0$_2$],	miss
1	[00$\underline{0}$1$_2$],	hit
7	[01$\underline{1}$1$_2$],	miss
8	[10$\underline{0}$0$_2$],	miss

Cache Memories

Set associative cache

- A scenario for a sequence of reads

t=2	s=1	b=1
xx	x	x

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

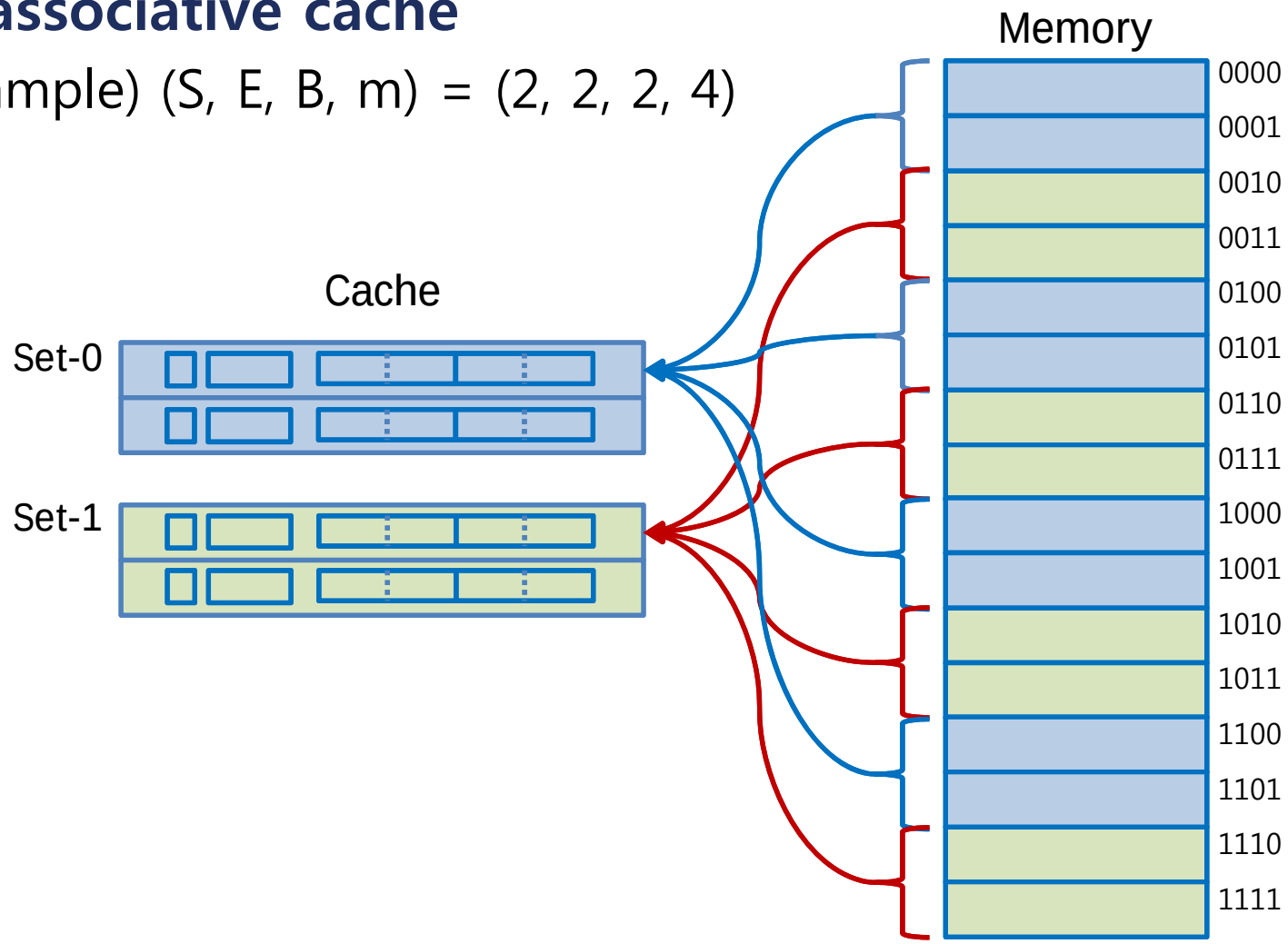
M = 16 bytes, B = 2 bytes, S = 2 sets,
E = 2 blocks/set

Address trace (reads, one byte per read)

0	[00$\underline{0}$0$_2$],	miss
1	[00$\underline{0}$1$_2$],	hit
7	[01$\underline{1}$1$_2$],	miss
8	[10$\underline{0}$0$_2$],	miss
0	[00$\underline{0}$0$_2$]	hit

Cache Memories

- **Set associative cache**
  - Example) (S, E, B, m) = (2, 2, 2, 4)



Cache Memories

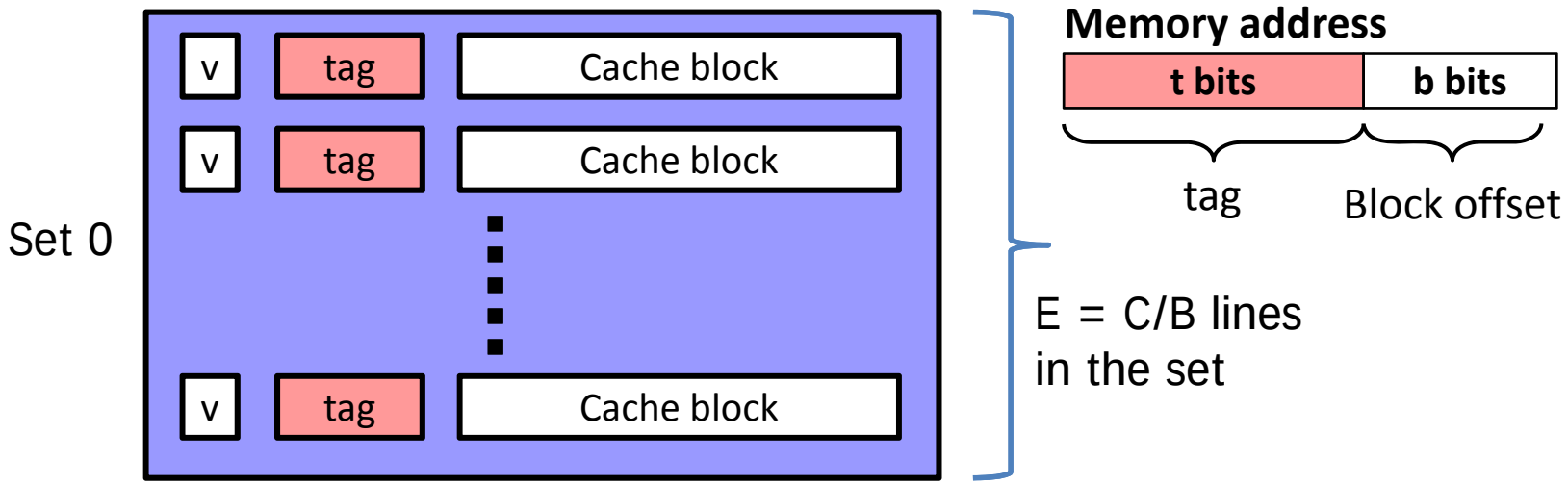
- **Set associative cache**
  - Line replacement policies
    - Random
    - LFU (Least Frequently Used)
    - LRU (Least Recently Used)
    - Etc.

    - These policies require additional hardware and time
    - But, as we move further down the memory hierarchy, the cost of miss becomes more expensive and it becomes more worthwhile to minimize misses with good replacement policies

Cache Memories

Fully associative cache

- Consists of a single set that contains all cache lines
- No set index bit in the address
- E = C/B
- Line matching and word selection
  work the same as with a set associative cache

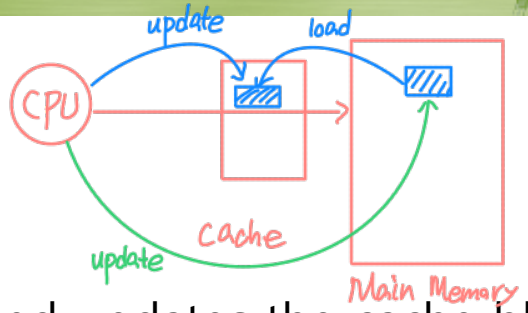


Cache Memories

Issues with writes

- Multiple copies of data exist
  - L1, L2, L3, main memory, disk, …
- What to do on a write-hit?
  - **Write-through** ⟶ 다음 low-level 까지 write 수행
    - ✓ Immediately writes the (updated) cache block to the next lower level
  - **Write-back** ⟶ cache 메모리까지만 저장해두고, 메인 메모리에 저장은 뒤로 미룬다.
    - ✓ Defers the update as long as possible
    - ✓ Writes the updated block to the next lower level only when it is evicted from the cache for replacement
    - ✓ Needs a dirty bit for each cache line (Indicates whether the contents of the cache block is different from the original copy)

Cache Memories



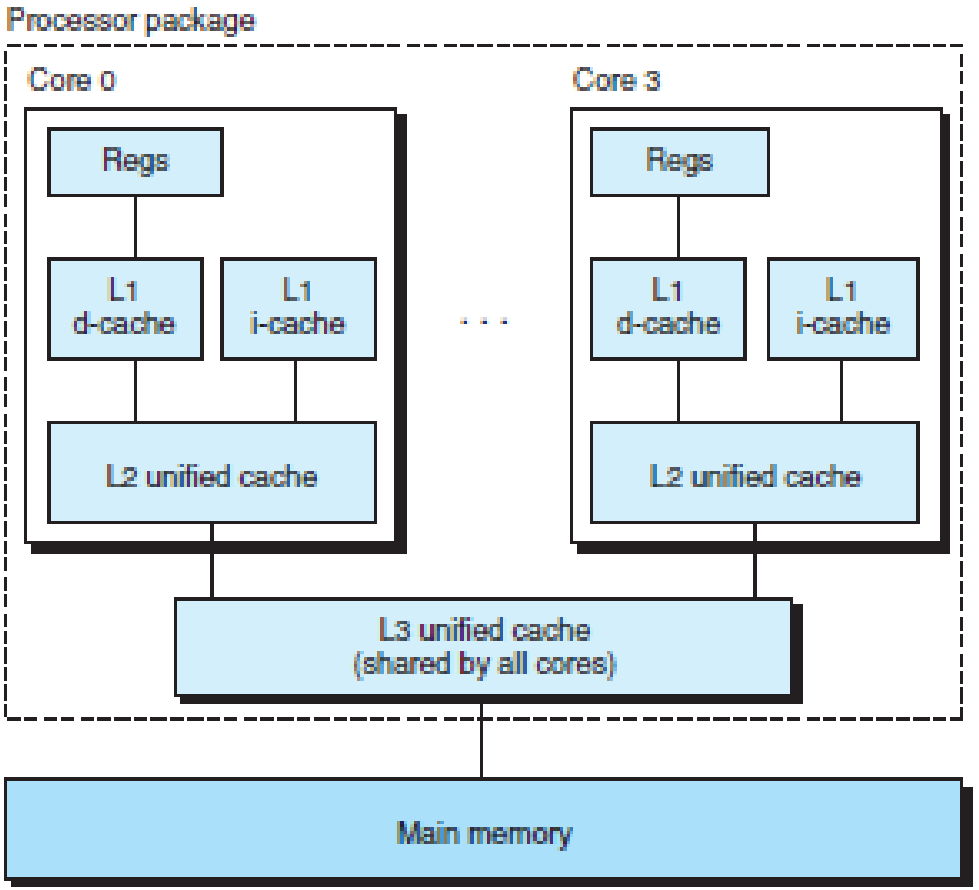
Issues with writes

- What to do on a write-miss?
  - **Write-allocate**
    - ✓ Loads the block into the cache and updates the cache block
    - ✓ Good if more writes to the location follow
  - **No-write-allocate**
    - ✓ Bypasses the cache and writes the data directly to the next lower level
- Typically,
  - Write-through + No-write-allocate
  - **Write-back + Write-allocate**

Cache Memories

Intel Core i7 cache hierarchy

- i-cache
  - Instruction cache
  - Typically read-only
- d-cache
  - Data cache
- Unified cache
  - Holds both instructions and data



Cache Memories

Intel Core i7 cache hierarchy

- Basic characteristics of Intel Core i7 cache

Cache type	Access time (#cycles)	Cache size (C)	Assoc. (E)	Block size (B)	# Sets (S)
L1 i-cache	4	32KB	8	64B	64
L1 d-cache	4	32KB	8	64B	64
L2 unified cache	10	256KB	8	64B	512
L3 unified cache	40~75	8MB	16	64B	8,192

Cache Memories

Cache performance metrics

- Miss rate
  - Fraction of memory references not found in cache
    = *#misses / #references*
  - Typically, (in percentages)
    - ✓ 3~10% for L1
    - ✓ Can be quite small (< 1%) for L2, depending on the size
- Hit rate
  - Fraction of memory references that hit
  - 1 – Miss rate

Cache Memories

Cache performance metrics

- Hit time
  - Time to deliver a word in the cache to the CPU
    - ✓ Includes time to determine whether the line is in the cache
    - ✓ Several clock cycles for L1 caches
- Miss penalty
  - Additional time required because of a miss
    - ✓ Typically 10 cycles from L2 (for L1 miss),
      50 cycles from L3 (for L2 miss),
      and about 200 cycles from main memory (for L3 miss)

Cache Memories

Cache performance metrics

- Note)
  - Huge difference between a hit and a miss
    - ✓ Could be 100×, with just L1 and main memory
  - Would you believe 99% hits is twice as good as 97%?
    - ✓ Consider cache hit time of 1 cycle and miss penalty of 100 cycles
    - ✓ Average (effective) access time
      97% hits: 1 cycle + 0.03 * 100 cycles = **4 cycles**
      99% hits: 1 cycle + 0.01 * 100 cycles = **2 cycles**
  - This is why "miss rate" is used instead of "hit rate"

Writing Cache-friendly Code

- **Basic approach**
  - Make the common case go fast
    - Focus on the inner loops of the core functions
  - Minimize the number of cache misses in the inner loops
    - Repeated references to variables are good (temporal locality)
    - Stride-1 reference patterns are good (spatial locality)

Writing Cache-friendly Code

- **Example)**
  - Array **a** is block aligned
  - 4B words, 4 word cache blocks, initially empty cache

```
int sum_array_rows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```

	j=0	j=1	j=2	j=3	j=4	j=5	j=6	j=7
i=0	1 [m]	2 [h]	3 [h]	4 [h]	5 [m]	6 [h]	7 [h]	8 [h]
i=1	9 [m]	10 [h]	11 [h]	12 [h]	13 [m]	14 [h]	15 [h]	16 [h]
i=2	17 [m]	18 [h]	19 [h]	20 [h]	21 [m]	22 [h]	23 [h]	24 [h]
i=3	25 [m]	26 [h]	27 [h]	28 [h]	29 [m]	30 [h]	31 [h]	32 [h]

Writing Cache-friendly Code

- **Example)**
  - Array **a** is block aligned
  - 4B words, 4 word cache blocks, initially empty cache

```
int sum_array_cols(int a[M][N])
{
    int i, j, sum = 0;

    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];
    return sum;
}
```

	j=0	j=1	j=2	j=3	j=4	j=5	j=6	j=7
i=0	1 [m]	5 [m]	9 [m]	13 [m]	17 [m]	21 [m]	25 [m]	29 [m]
i=1	2 [m]	6 [m]	10 [m]	14 [m]	18 [m]	22 [m]	26 [m]	30 [m]
i=2	3 [m]	7 [m]	11 [m]	15 [m]	19 [m]	23 [m]	27 [m]	31 [m]
i=3	4 [m]	8 [m]	12 [m]	16 [m]	20 [m]	24 [m]	28 [m]	32 [m]

Summary