

























Cache Memory

1-Initial State

Request Byte
Address

000	 		
001	 		
010	 		
011	 		
100	 		
101	 		
110	 		
111	 		

8 Blocks

00000	
00011	
01000	
01011	
10000	
10011	
11000	
11011	

32 Blocks

2.

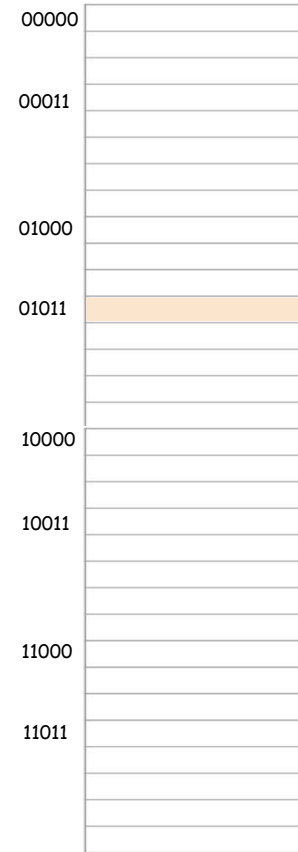
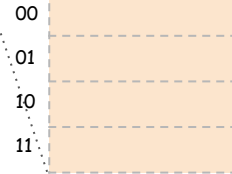
Request Byte
Address

0101110



000	--	0	
001	--	0	
010	--	0	
011	--	0	
100	--	0	
101	--	0	
110	--	0	
111	--	0	

8 Blocks



32 Blocks

0101100
0101101
0101110
0101111

Some
Data
Here

3.

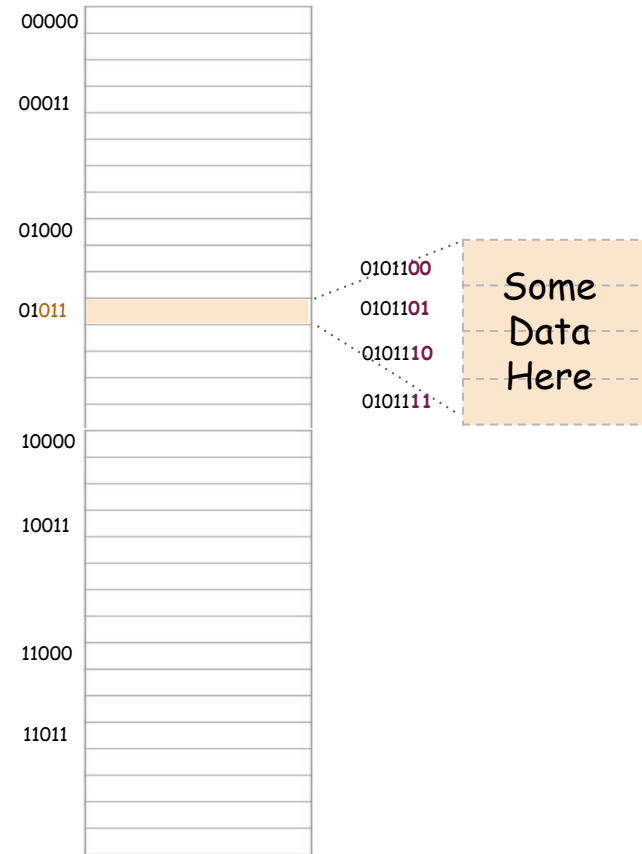
Request Byte
Address

0101110

000	--	0	
001	--	0	
010	--	0	
011	0 1	1	Mem(01011)
100	--	0	
101	--	0	
110	--	0	
111	--	0	

8 Blocks

A diagram showing a 4-bit bus with address lines 00, 01, 10, and 11. The bus is labeled "Some Data Here".



32 Blocks

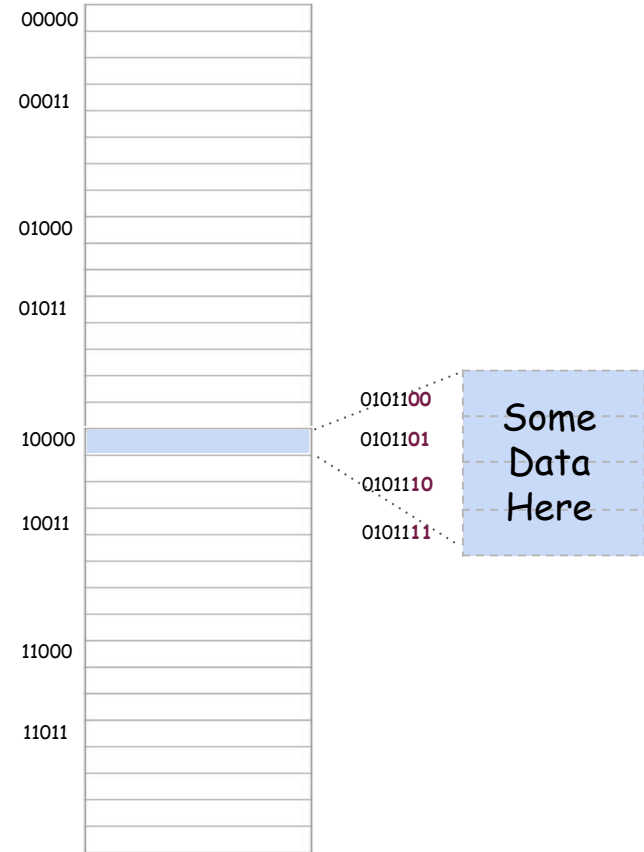
5.

Request Byte
Address

1000001

000	1 0	1	Mem(10000)
001	--	0	
010	--	0	
011	0 1	1	Mem(01011)
100	--	0	
101	--	0	
110	--	0	
111	--	0	

8 Blocks



32 Blocks

0101100



8 Blocks

11011

32 Blocks

7.

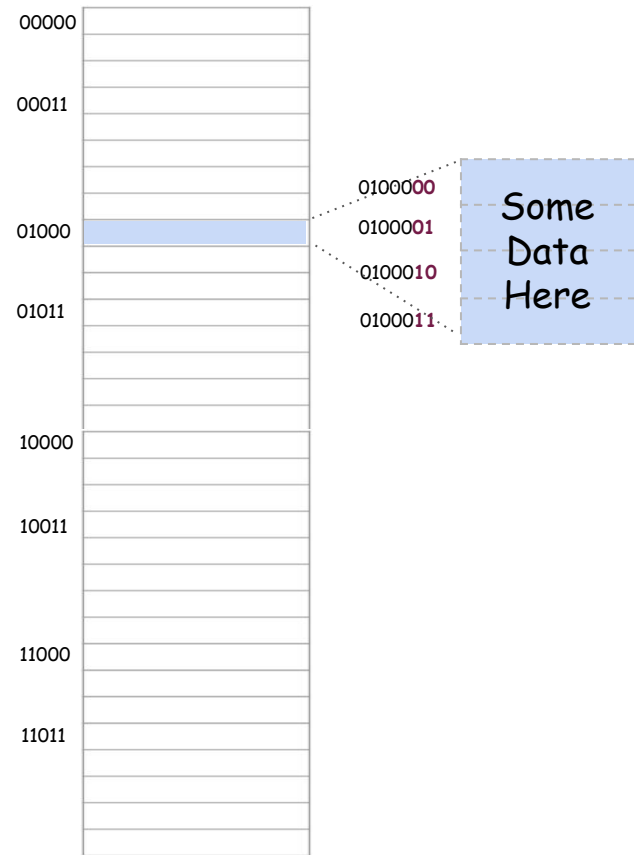
Request Byte
Address

0100011

Cache Miss

000	1 0	1	Mem(10000)
001	--	0	
010	--	0	
011	0 1	1	Mem(01011)
100	--	0	
101	--	0	
110	--	0	
111	--	0	

8 Blocks



32 Blocks

7.

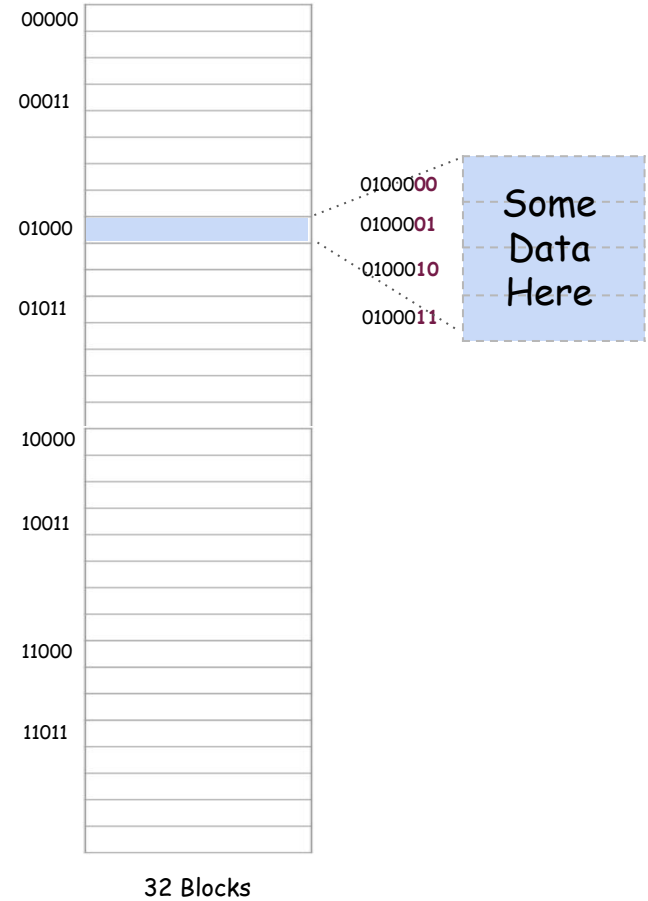
Request Byte
Address

0100011

000	01	1	Mem(01000)
001	--	0	
010	--	0	
011	01	1	Mem(01011)
100	--	0	
101	--	0	
110	--	0	
111	--	0	

8 Blocks

replace



Cache Performance

$$\text{MissRate} = \# \text{CacheMisses} / \# \text{CacheAccesses}$$

Average Memory
Access Time



The diagram shows the formula $AMAT = HitTime + MissRate \times MissPenalty$ enclosed in a dashed rectangular box. An arrow points from the text 'Average Memory Access Time' to the 'AMAT' term. Another arrow points from the 'MissRate' term in the formula above to the 'MissRate' term in this formula.

$$AMAT = HitTime + MissRate \times MissPenalty$$

$$AMAT = HitTime + MissRate \times MissPenalty$$

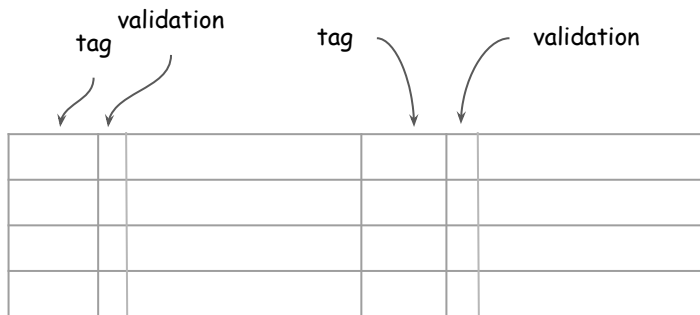
$$MissRate = \#CacheMisses / \#CacheAccesses$$

Cache Miss Categories (3Cs)

- Compulsory (First Access is always a miss)
- Capacity (program working set is larger than cache capacity)
- Conflict (several blocks are mapped to same block frame)

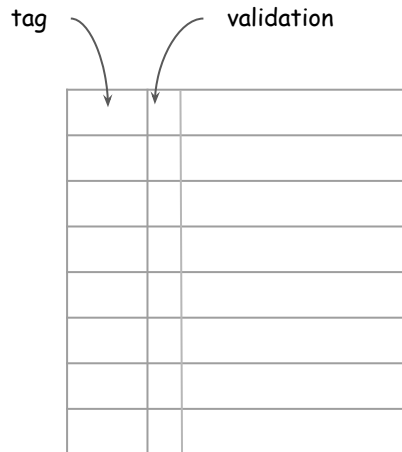
Associative Cache

2-ways set associative



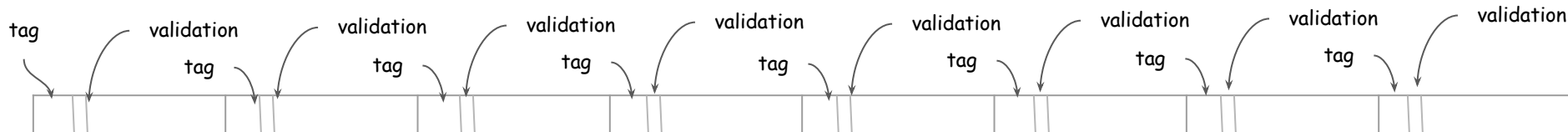
#Sets \neq #Blocks

1-way set associative
(direct mapping)



#Sets = #Blocks

8-ways set associative
(fully associative)



#Sets = 1
(cacheIndex=0)

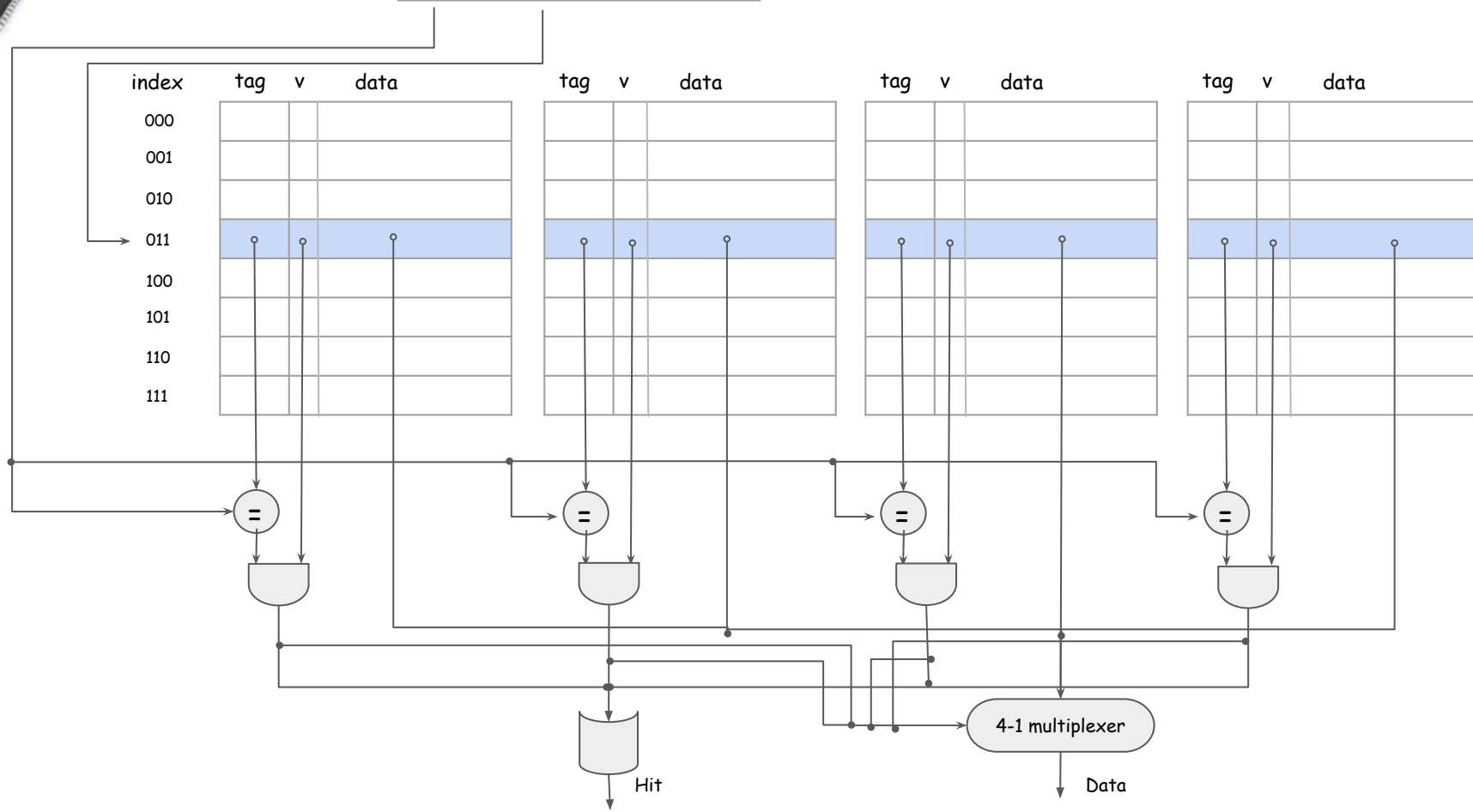
$$\text{CacheSize} = \text{\#Sets} \times \text{Ways} \times \text{\#BlockSize}$$

Associative Cache



Request Byte Address

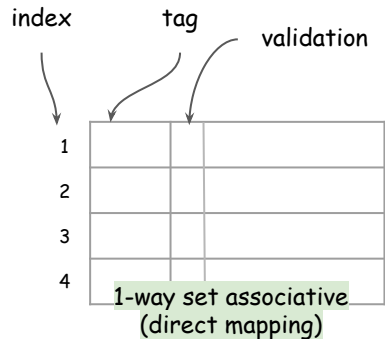
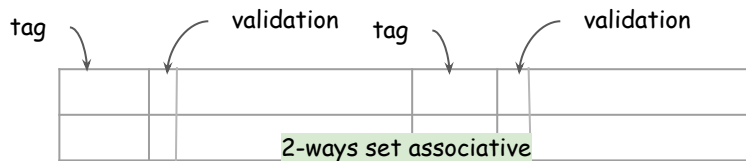
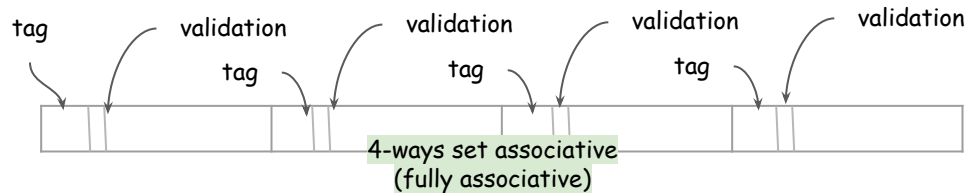
tag	index	offset
-----	-------	--------



Associative Cache

Which cache organization reduces misses?

What type of misses can this organization reduce (compulsory, capacity, conflict) ?



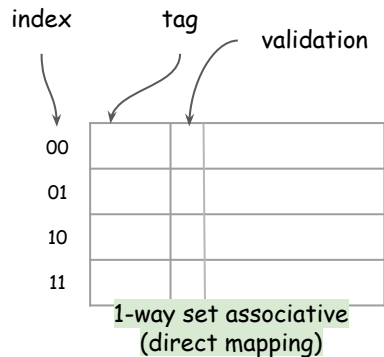
Exercise:

Assume there are three small caches, each consisting of four one-word blocks. One cache is fully associative, a second is two-way set associative, and the third is direct mapped.

Find the number of misses for each cache organization, given the following sequence of block addresses: 0, 8, 0, 6, and 8. (16 bits address)

Exercise Solution:

Step 1



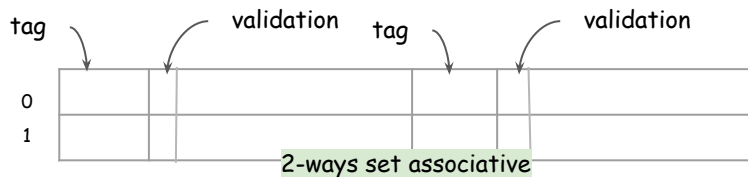
Offset=1 bit
Index = 2 bits
Tag = 13 bits

Miss

Total no. misses = 1

Request Byte Address

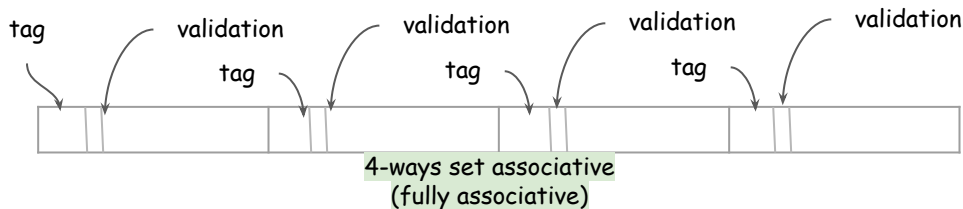
$(0)_{10} = (0000000000000000)_2$



Offset=1 bit
Index = 1 bits
Tag = 14 bits

Miss

Total no. misses = 1



Offset=1 bit
Index = 0 bits
Tag = 15 bits

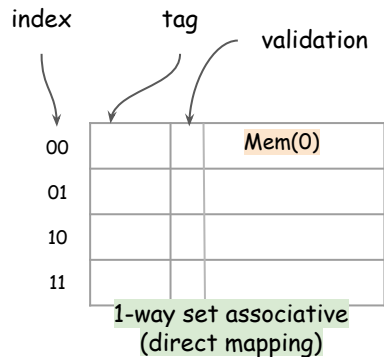
Miss

Total no. misses = 1

Associative Cache

Exercise Solution:

Step 2



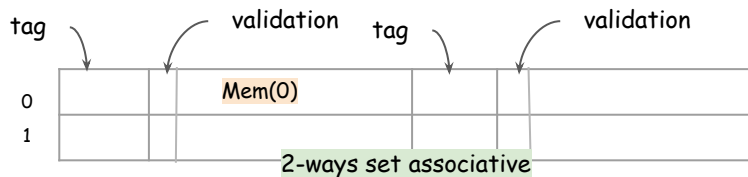
Offset=1 bit
Index = 2 bits
Tag = 13 bits

Miss

Total no. misses = 2

Request Byte Address

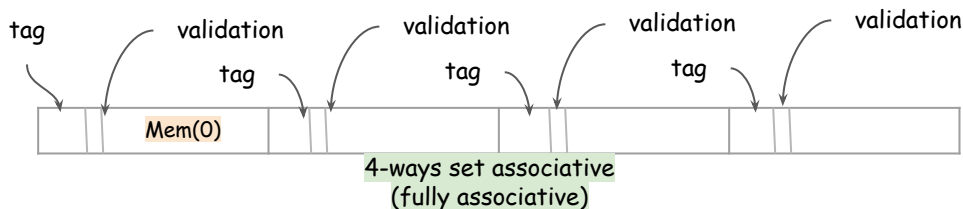
$(8)_{10} = (0000000000001000)_2$



Offset=1 bit
Index = 1 bits
Tag = 14 bits

Miss

Total no. misses = 2



Offset=1 bit
Index = 0 bits
Tag = 15 bits

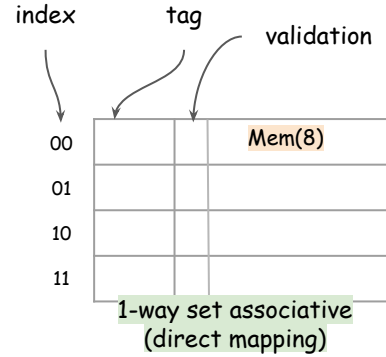
Miss

Total no. misses = 2

Associative Cache

Exercise Solution:

Step 3

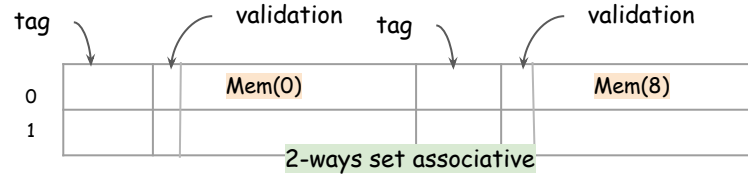


Miss

Offset=1 bit
Index = 2 bits
Tag = 13 bits

Total no. misses = 3

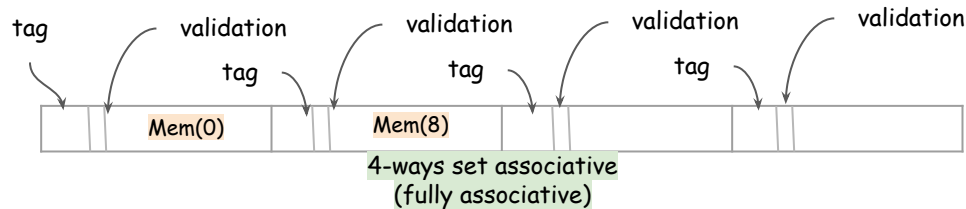
Request Byte Address

$$(0)_{10} = (0000000000000000)_2$$


Hit

Offset=1 bit
Index = 1 bits
Tag = 14 bits

Total no. misses = 2



Hit

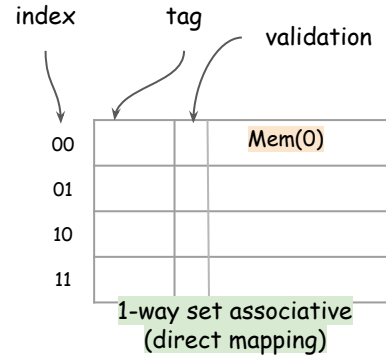
Offset=1 bit
Index = 0 bits
Tag = 15 bits

Total no. misses = 2

Associative Cache

Exercise Solution:

Step 4

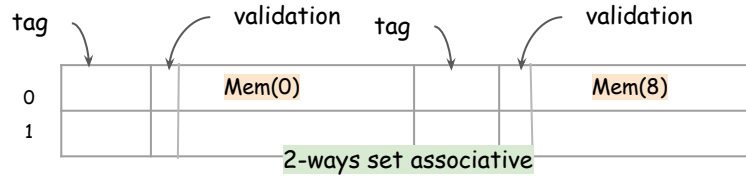


Miss

Offset=1 bit
Index = 2 bits
Tag = 13 bits

Total no. misses = 4

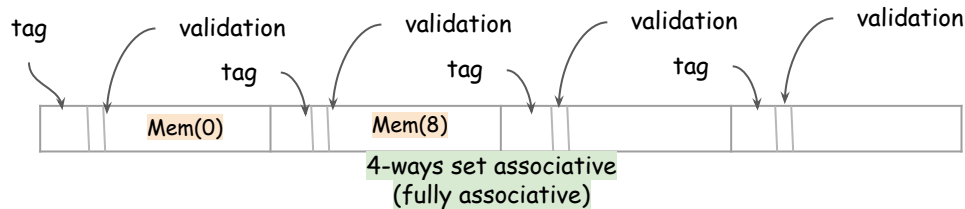
Request Byte Address

$$(6)_{10} = (0000000000000110)_2$$


Miss

Offset=1 bit
Index = 1 bits
Tag = 14 bits

Total no. misses = 3



Miss

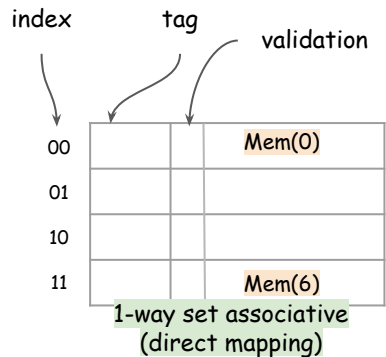
Offset=1 bit
Index = 0 bits
Tag = 15 bits

Total no. misses = 3

Associative Cache

Exercise Solution:

Step 5



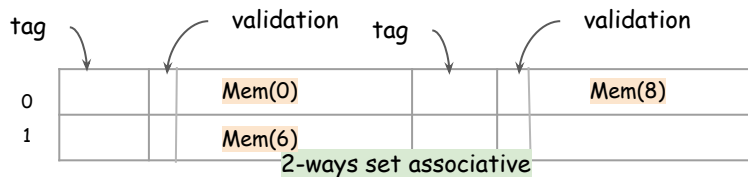
Offset=1 bit
Index = 2 bits
Tag = 13 bits

Miss

Total no. misses = 5

Request Byte Address

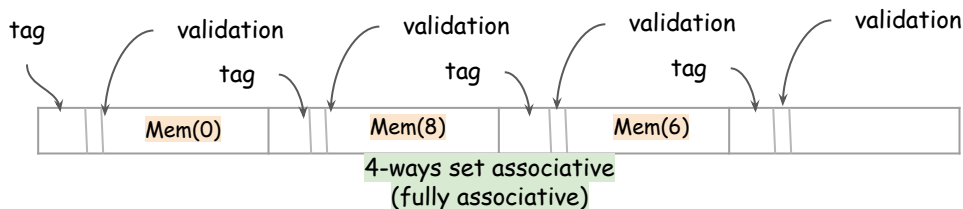
$(8)_{10} = (0000000000001000)_2$



Offset=1 bit
Index = 1 bits
Tag = 14 bits

Hit

Total no. misses = 3



Offset=1 bit
Index = 0 bits
Tag = 15 bits

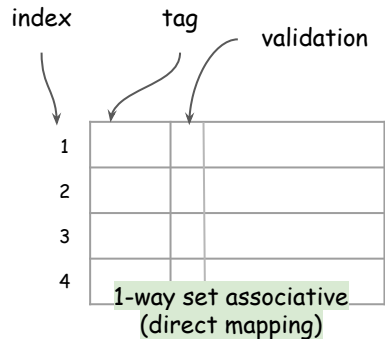
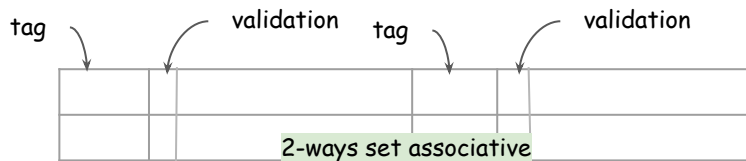
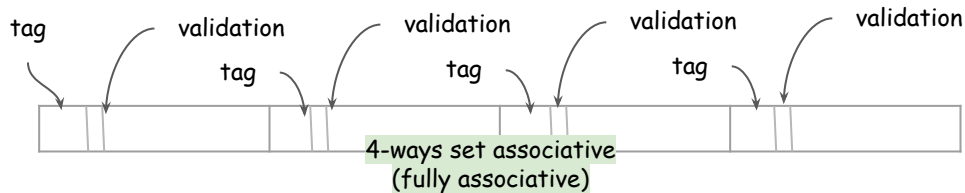
Hit

Total no. misses = 3

Associative Cache

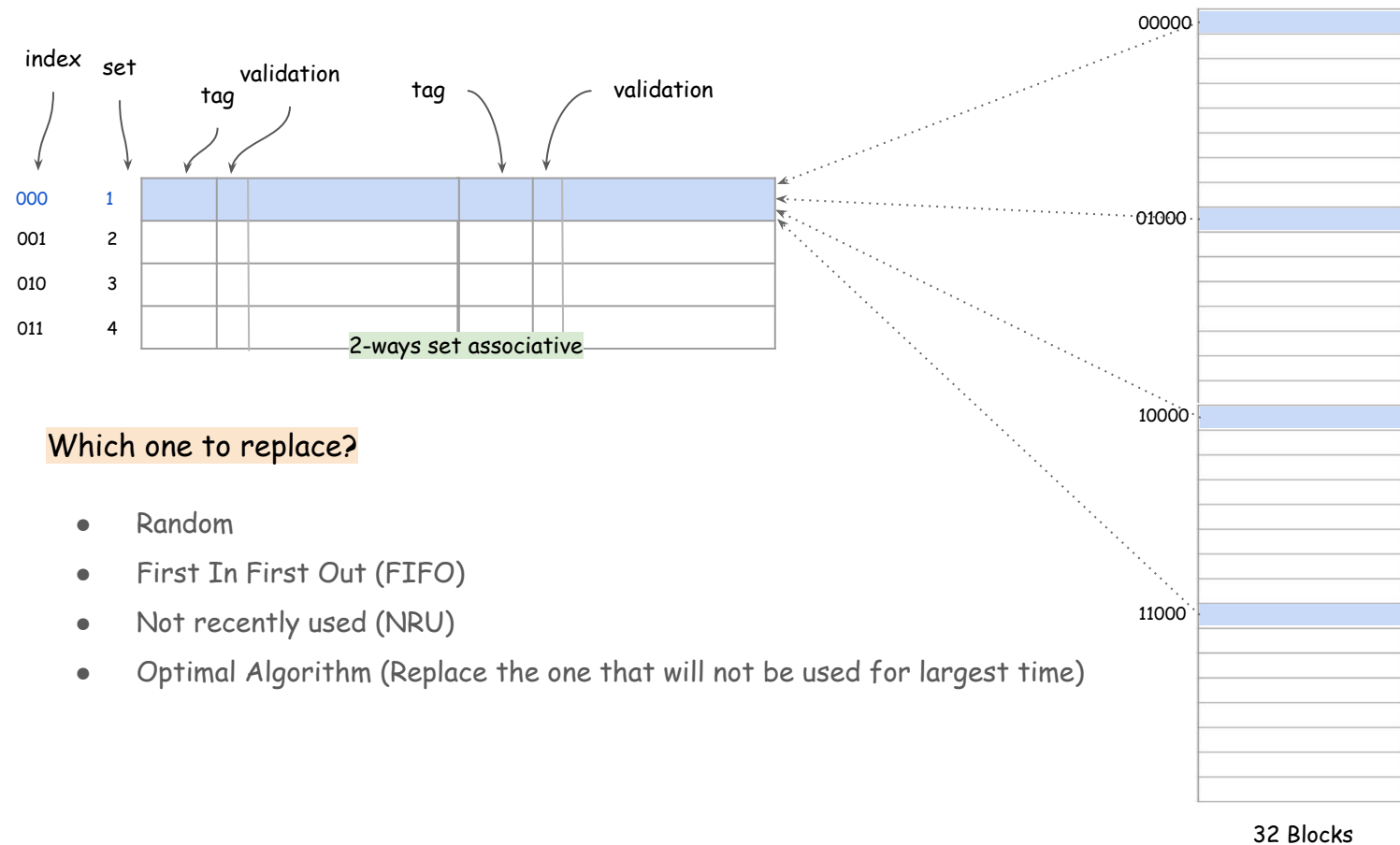
Which cache organization reduces misses?

What type of misses can this organization reduce (compulsory, capacity, conflict) ?



+ Reduce conflict misses
- Increase Hit time
- Increase power consumption

Associative Cache

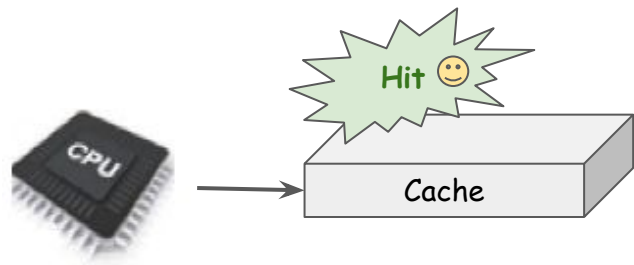


Which one to replace?

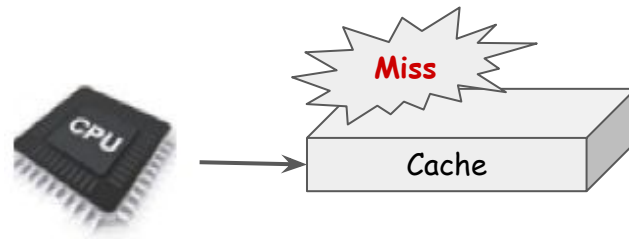
- Random
- First In First Out (FIFO)
- Not recently used (NRU)
- Optimal Algorithm (Replace the one that will not be used for largest time)

Writing Policies

Write Hit

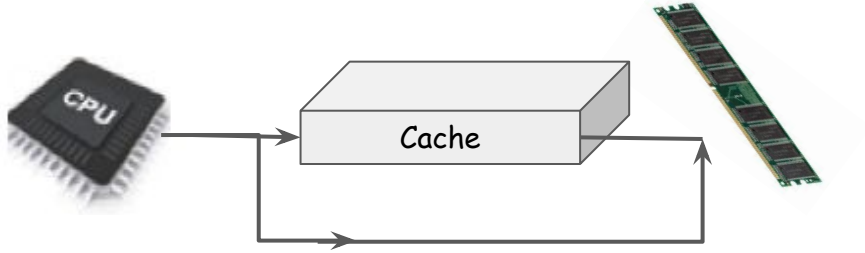


Write Miss



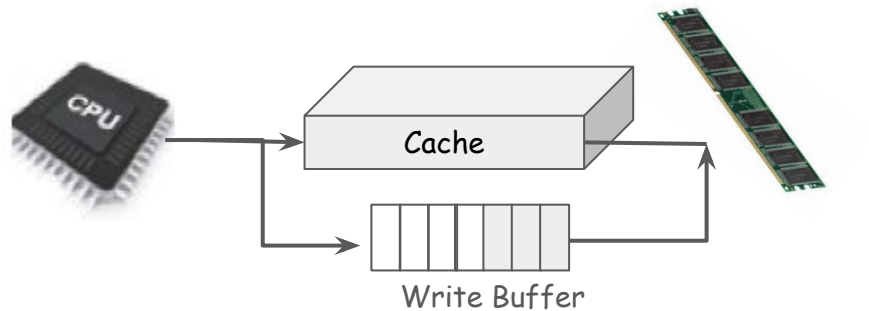
Write Hit

- Write Through Cache



Write Hit

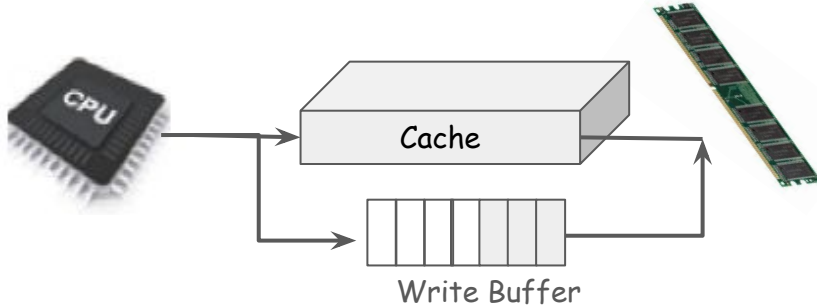
- Write Through Cache



Writing Policies

Write Hit

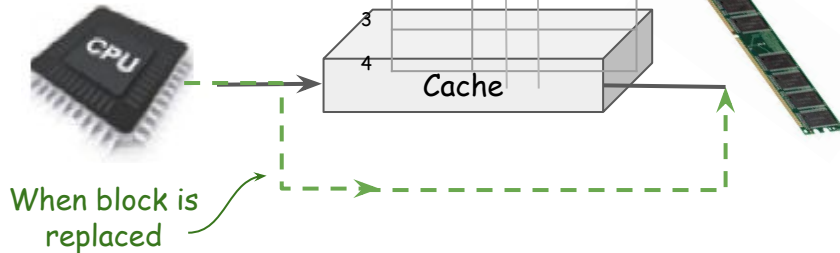
- Write Through Cache



- Write Back Cache

index tag dirty validation

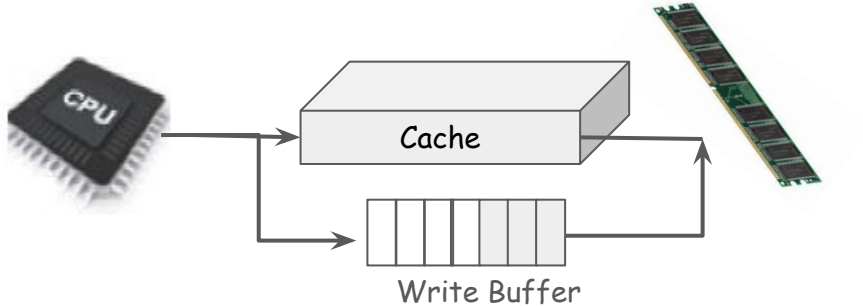
index	tag	dirty	validation
1			
2			
3			
4			



Writing Policies

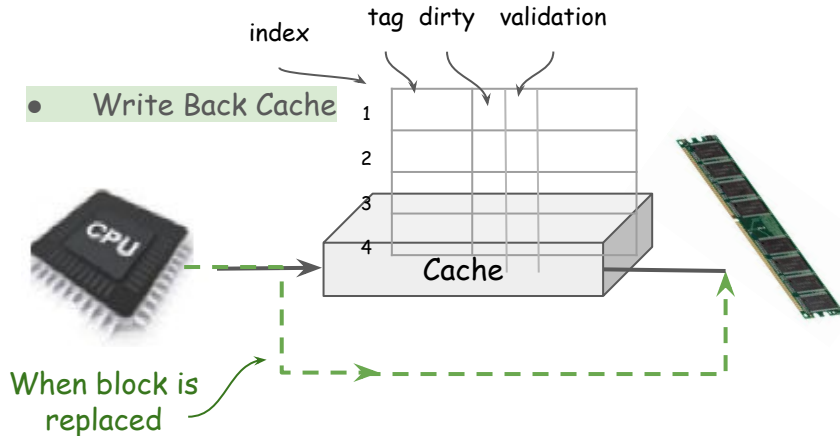
Write Hit

- Write Through Cache



- + Simple Cache logic
- + Cache and memory consistent
- Many writes to memory (more bus traffic)

- Write Back Cache

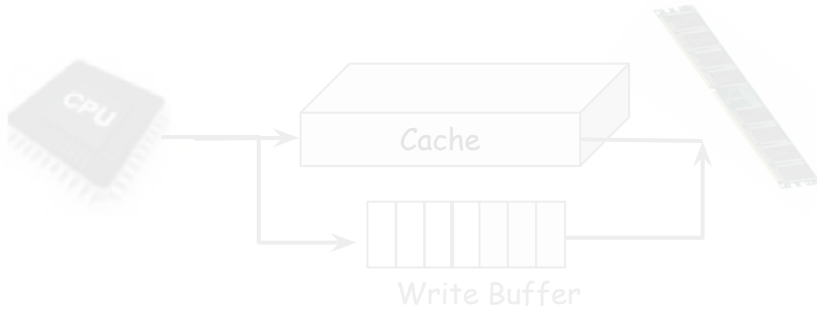


- + Fewer writes to memory
- Cache and memory inconsistent
- Requires dirty bit

Writing Policies

Write Hit

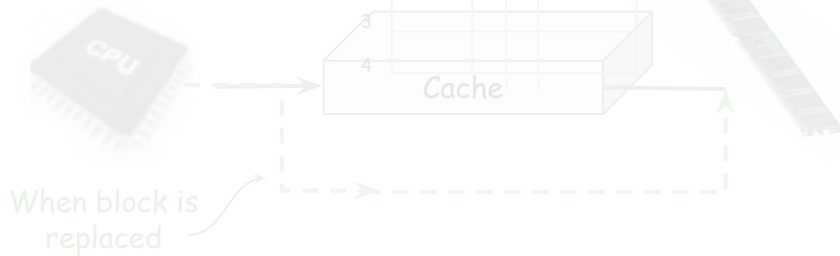
- Write Through Cache



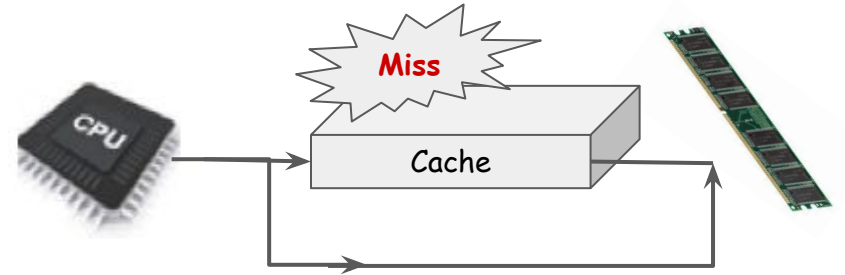
- Write Back Cache

index tag dirty validation

index	tag	dirty	validation
1			
2			
3			
4			



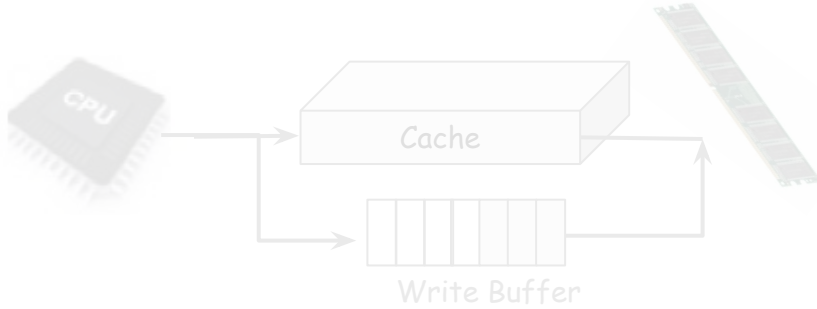
Write Miss



Writing Policies

Write Hit

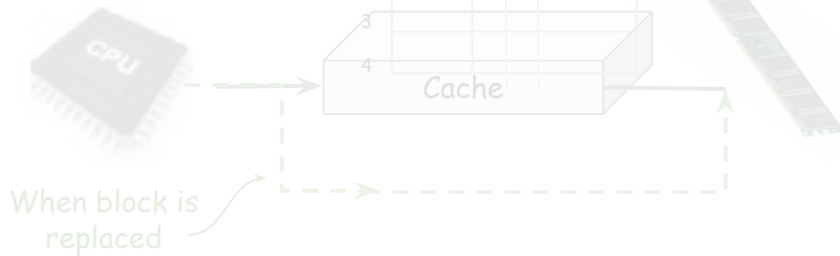
- Write Through Cache



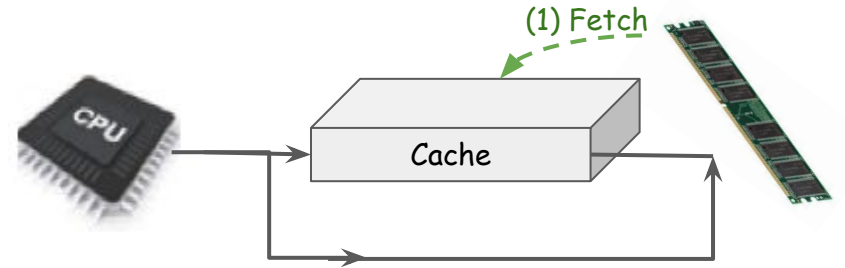
- Write Back Cache

index tag dirty validation

index	tag	dirty	validation
1			
2			
3			
4			



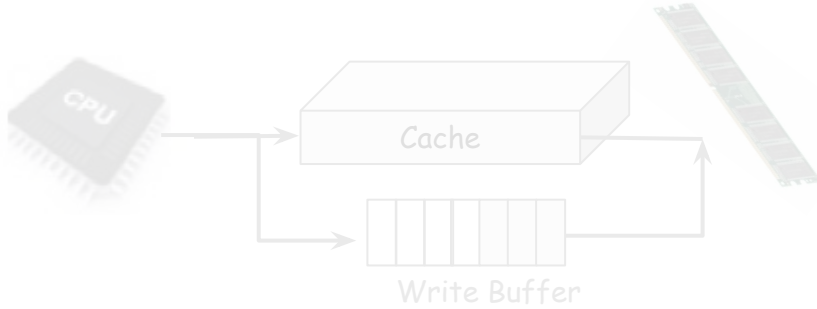
Write Miss



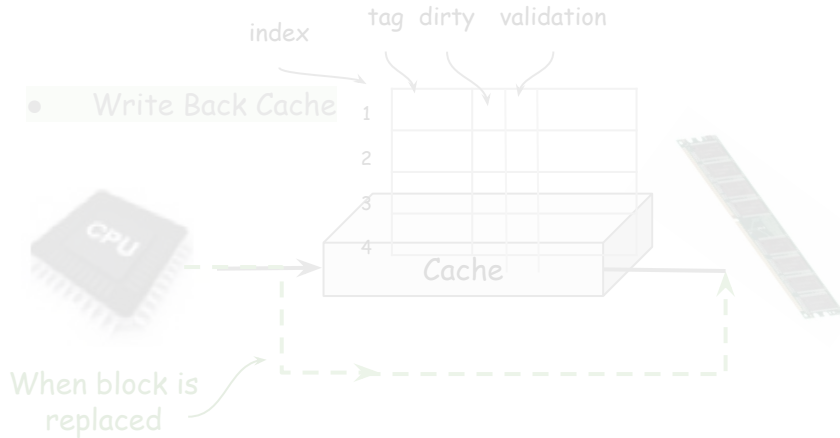
Writing Policies

Write Hit

- Write Through Cache

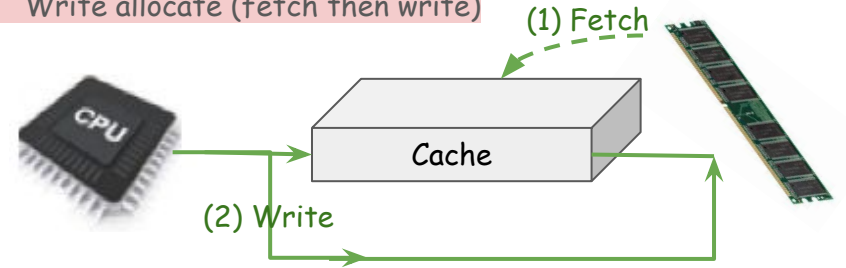


- Write Back Cache



Write Miss

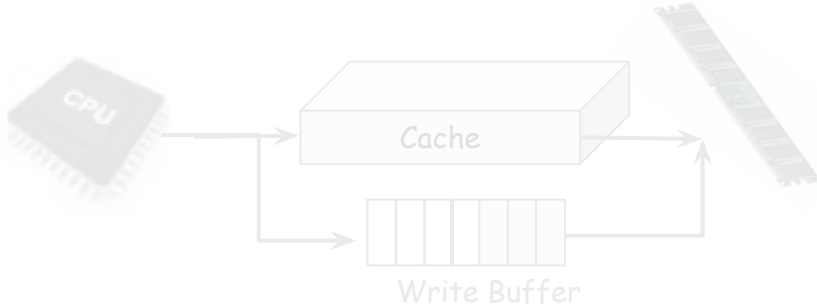
- Write allocate (fetch then write)



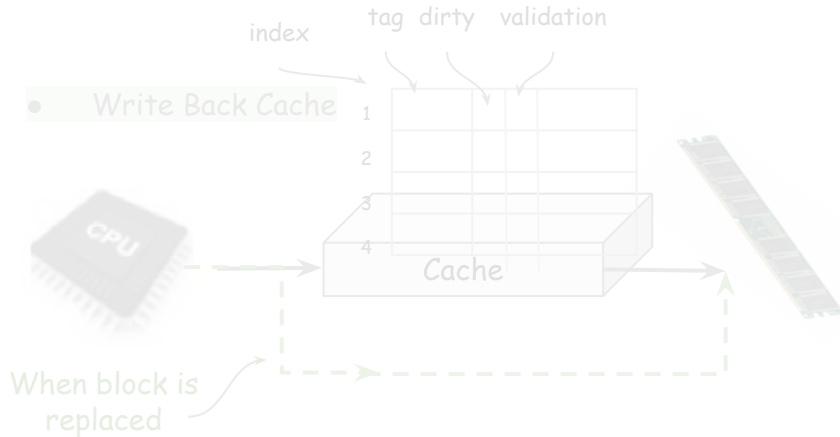
Writing Policies

Write Hit

- Write Through Cache

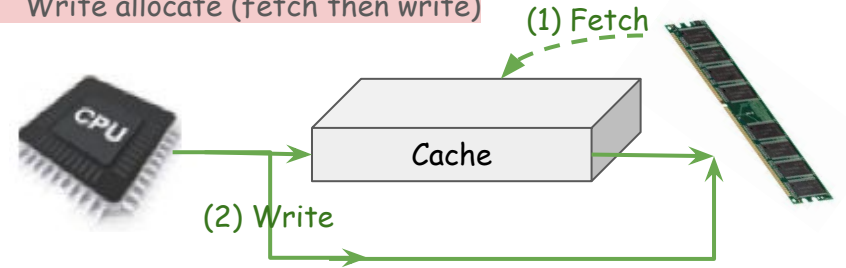


- Write Back Cache

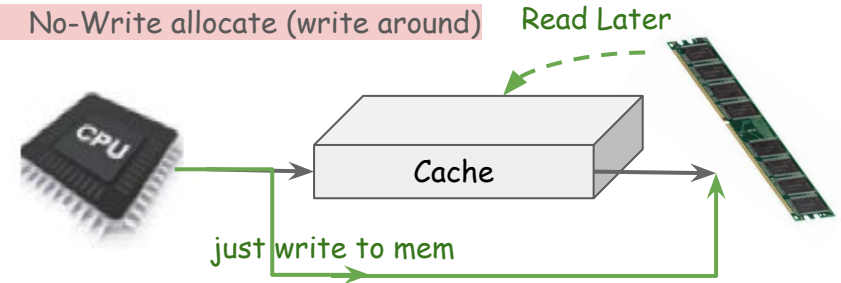


Write Miss

- Write allocate (fetch then write)



- No-Write allocate (write around)



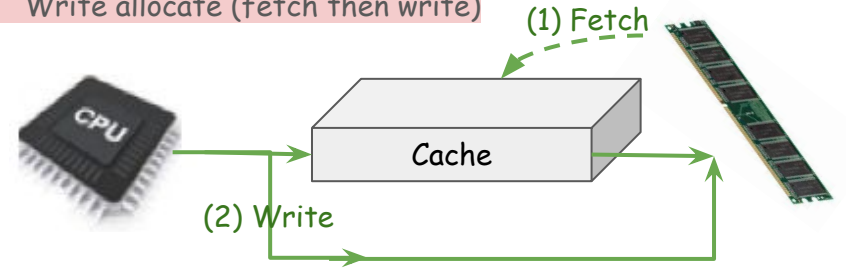
Writing Policies

- + Cache and memory consistent
- More writes

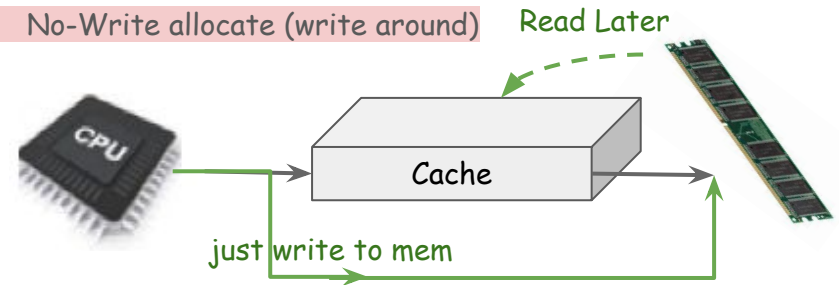
- + Fewer writes

Write Miss

- Write allocate (fetch then write)



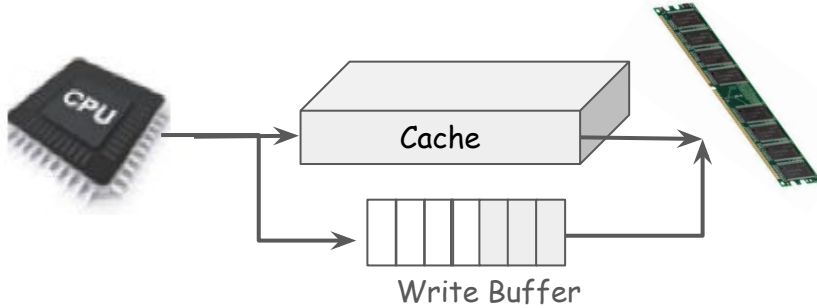
- No-Write allocate (write around)



Writing Policies

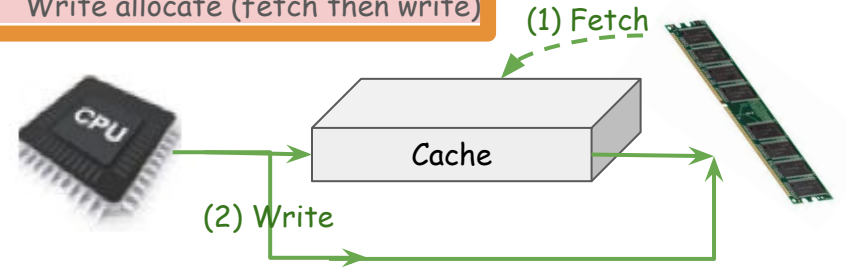
Write Hit

- Write Through Cache

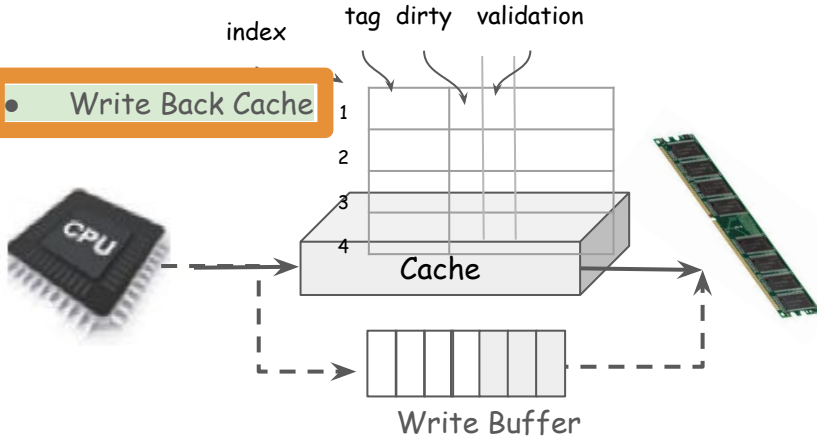


Write Miss

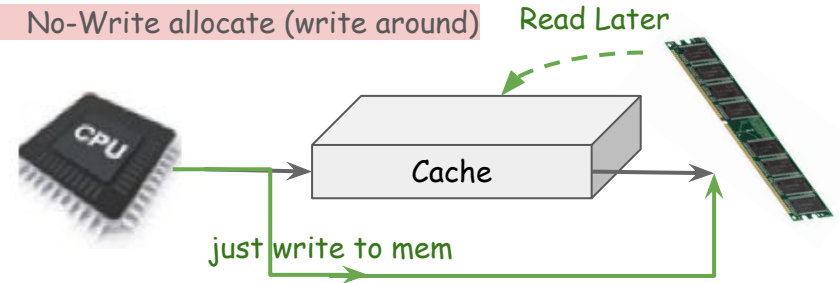
- Write allocate (fetch then write)



- Write Back Cache



- No-Write allocate (write around)



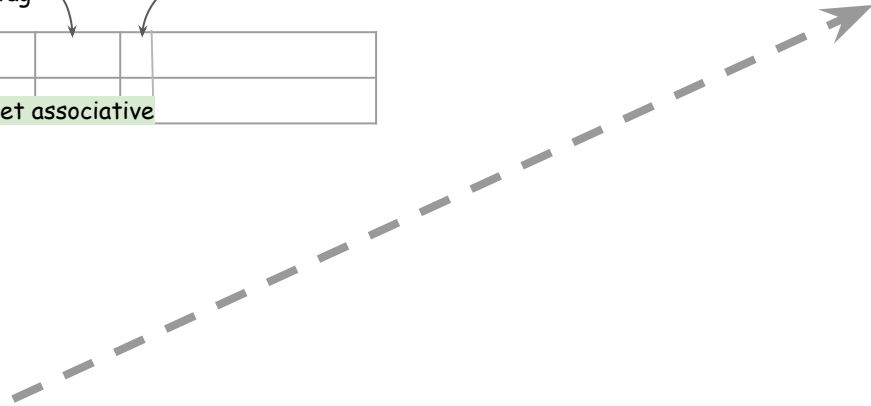
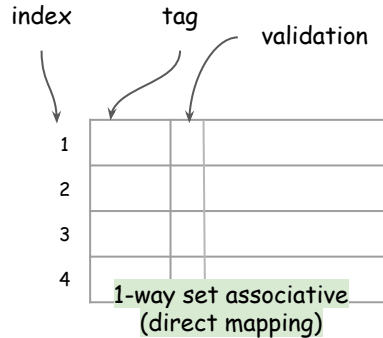
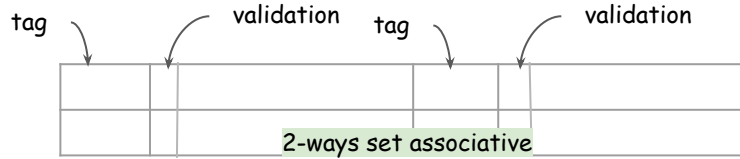
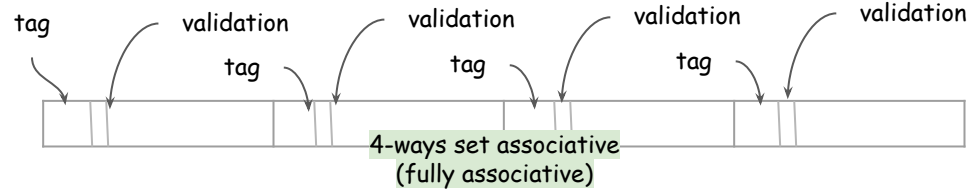
Performance

Average Memory Access Time

$$AMAT = \text{HitTime} + \text{MissRate} \times \text{MissPenalty}$$

✗

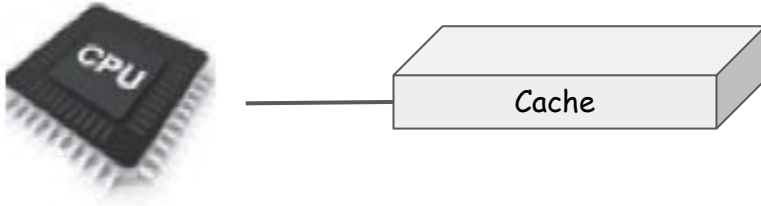
✓



Performance

- Small:
 - Cache Size
- Simple:
 - Low Associativity

1-IF
2-ID
3-EX
4-Mem
5-WB



Average Memory Access Time

$$AMAT = HitTime + MissRate \times MissPenalty$$

Small and Simple
Cache

Performance

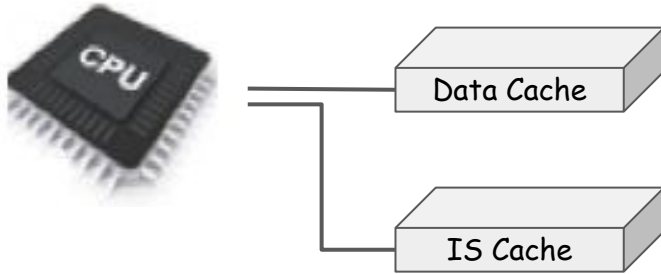
Average Memory Access Time

$$AMAT = HitTime + MissRate \times MissPenalty$$

Small and Simple
Cache

- Small:
 - Cache Size
 - Split Cache
- Simple:
 - Low Associativity
 - Split Cache

1-IF
2-ID
3-EX
4-Mem
5-WB

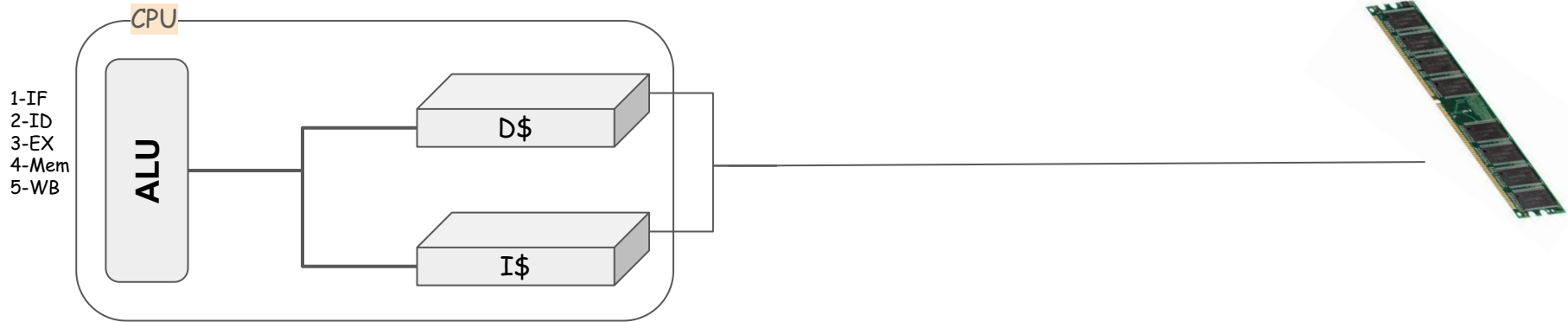


Performance

Average Memory Access Time

$$AMAT = HitTime + MissRate \times MissPenalty$$

✗

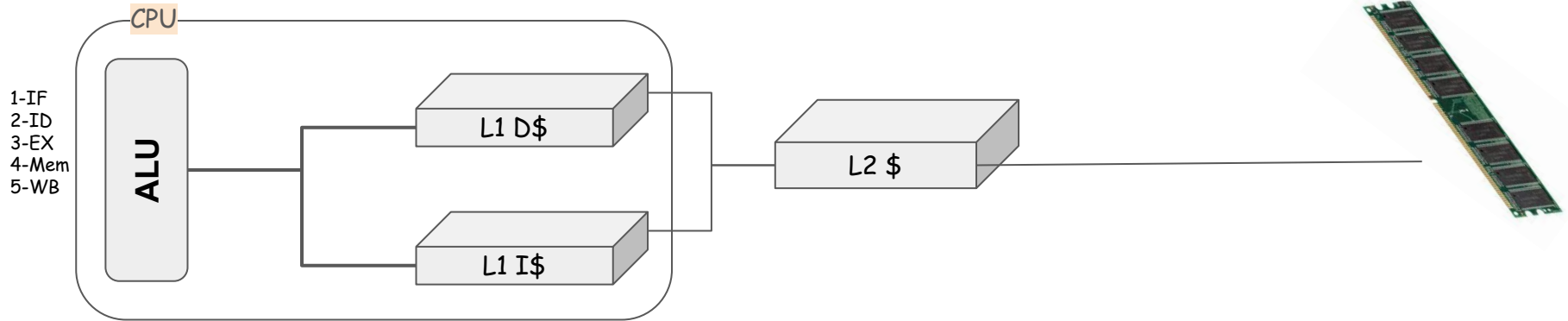


Performance

Average Memory Access Time

$$AMAT = HitTime + MissRate \times MissPenalty$$

Multi-Levels

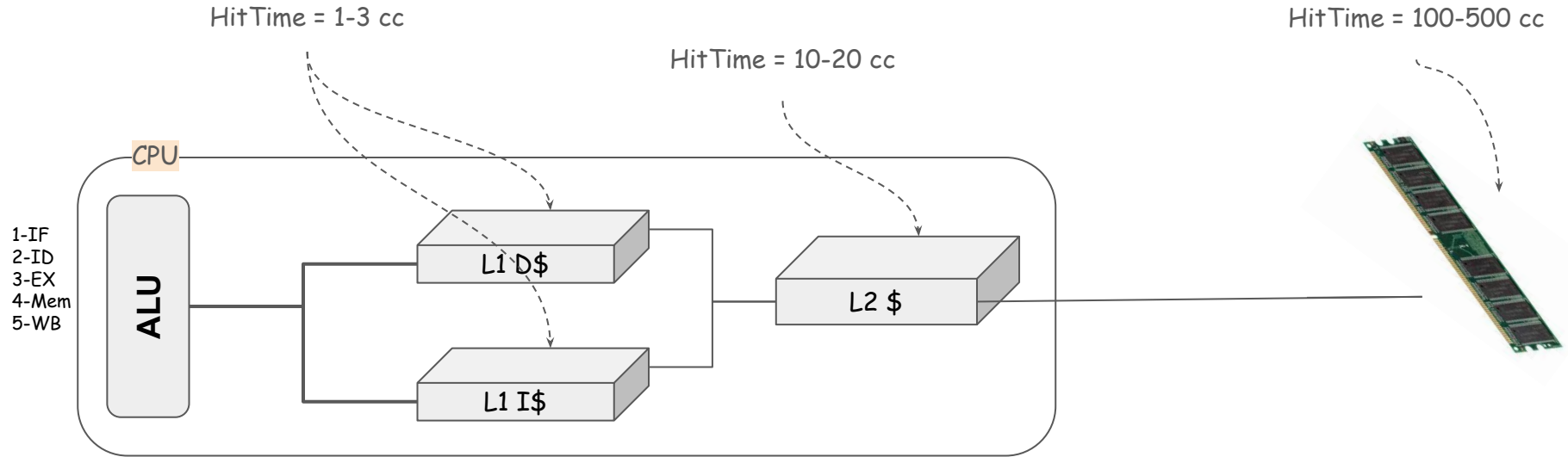


Performance

Average Memory Access Time

$$AMAT = HitTime + MissRate \times MissPenalty$$

Multi-Levels

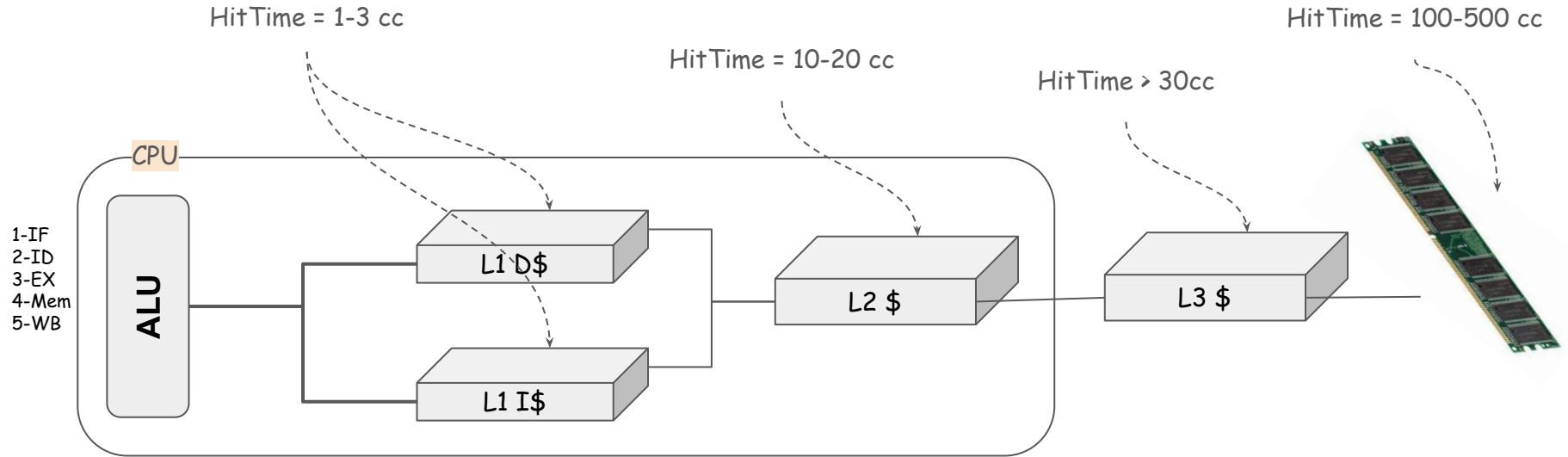


Performance

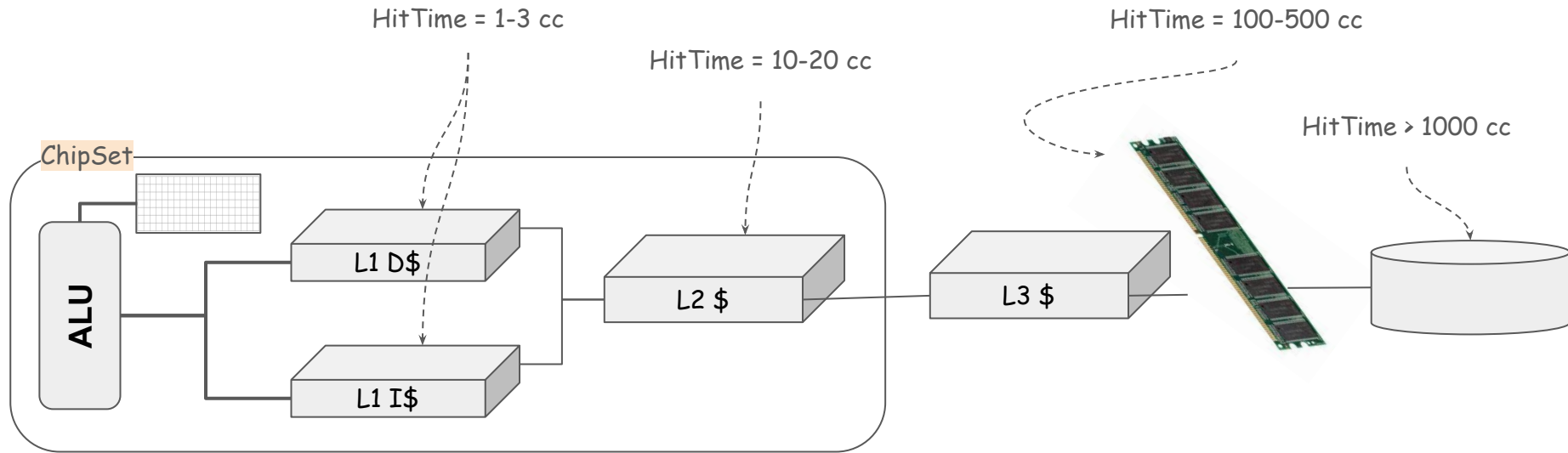
Average Memory Access Time

$$AMAT = HitTime + MissRate \times MissPenalty$$

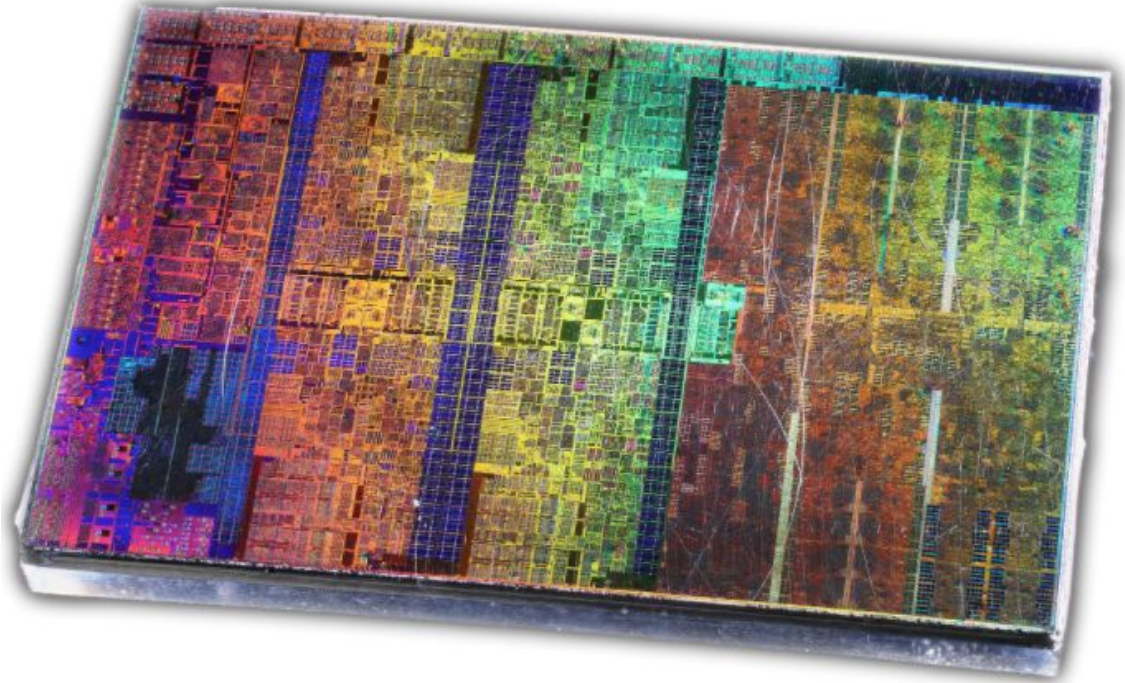
Multi-Levels



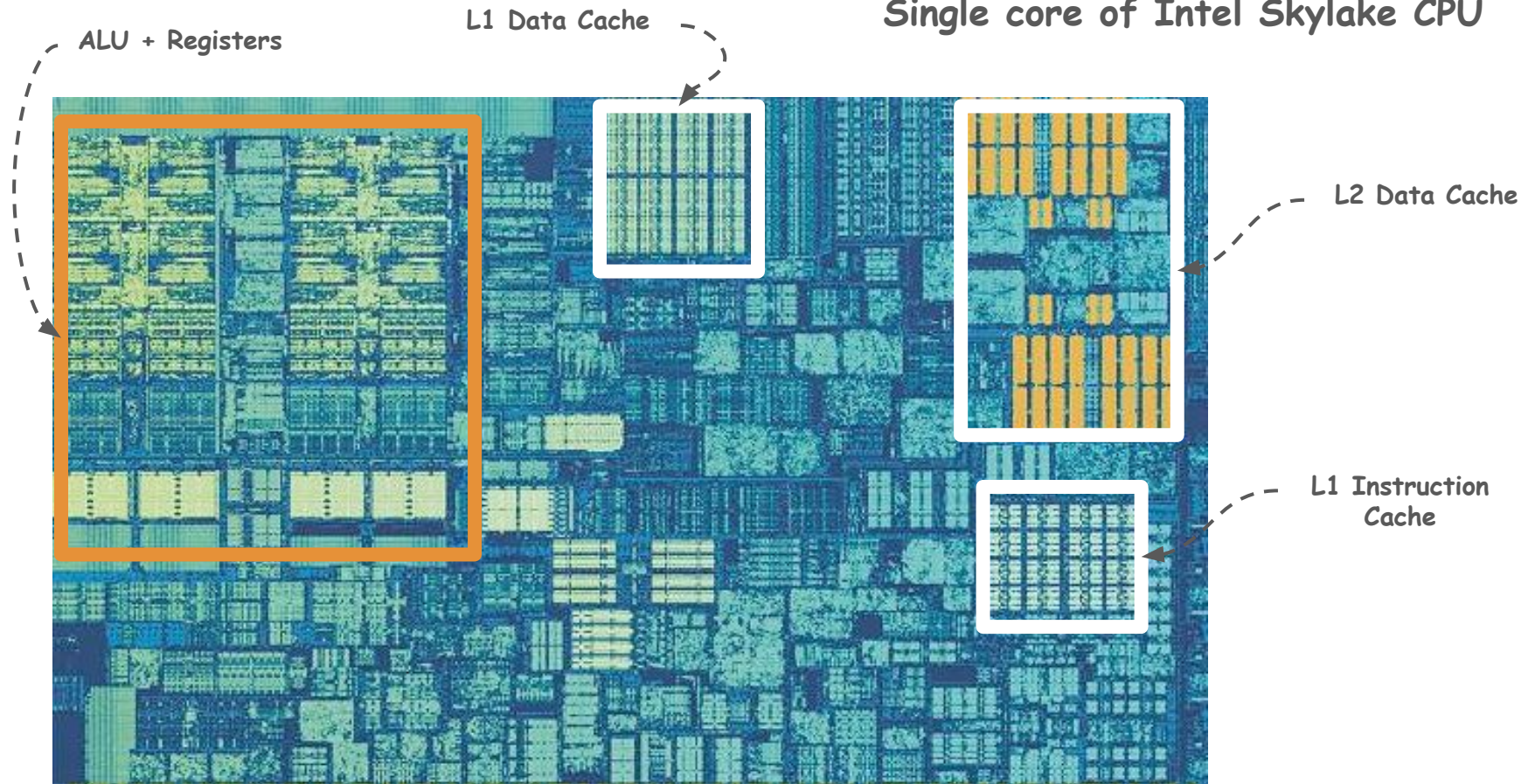
Performance



Intel Skylake CPU



Single core of Intel Skylake CPU



Performance

The screenshot shows the CPU-Z application window with the 'Caches' tab selected. The window has a title bar with standard OS controls and a menu bar with options: CPU, Caches, Mainboard, Memory, SPD, Graphics, Bench, and About. The 'Caches' section displays the following information:

- L1 D-Cache:** Size 32 KBytes, x 8. Descriptor: 8-way set associative, 64-byte line size.
- L1 I-Cache:** Size 32 KBytes, x 8. Descriptor: 8-way set associative, 64-byte line size.
- L2 Cache:** Size 256 KBytes, x 8. Descriptor: 4-way set associative, 64-byte line size.
- L3 Cache:** Size 12 MBytes. Descriptor: 12-way set associative, 64-byte line size.

Below the L3 Cache section, there are empty fields for Size, Descriptor, and Speed. At the bottom of the window, the CPU-Z logo and version 'Ver. 1.92.0.x64' are shown, along with 'Tools', 'Validate', and 'Close' buttons.

The screenshot shows the MacCPUID application window with the 'Cache' tab selected. The window has a title bar with standard OS controls and a menu bar with options: Info, Features, Cache, Performance, and Misc. The 'Cache' section displays a table of cache information:

Cache Type	Size	Assoc	Line Size	Max Threads
L1 Data	32KB	8	64B	1
L1 Instruction	32KB	8	64B	1
L2 Unified	3MB	12	64B	2

Below the table, the 'Cache Information' section provides detailed TLB and prefetching information:

- Instruction TLB: 2 MByte Pages, 4-way, 8 entries or 4 Mbyte page
- Instruction TLB: 4 KByte Pages, 4-way set associative, 128 entries
- Data TLB1: 4 MByte Pages, 4-way set associative, 32 entries
- 64-Byte Prefetching
- Data TLB0: 4 KByte pages, 4-way set associative, 16 entries
- Data TLB0: 4 MByte pages, 4-way set associative, 16 entries
- CLFLUSH Line Size 64

At the bottom of the window, there are buttons for 'Info', 'Features', 'Cache', 'Performance', and 'Misc'. The version 'v3.1' is displayed in the bottom right corner.

$$\text{MissRate} = \# \text{CacheMisses} / \# \text{CacheAccesses}$$

Average Memory
Access Time

$$\text{AMAT} = \text{HitTime} + \text{MissRate} \times \text{MissPenalty}$$

- Small
- Simple
- Low Associativity

- Large Block Size
- Large Cache
- High Associativity

- Small Block Size
- Multi-Level

Which of this functions has good locality?

```
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;  
}
```

```
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;  
}
```

0,0	0,1	0,2
1,0	1,1	1,2
2,0	2,1	2,2

0,0
0,1
0,2
1,0
1,1
1,2
2,0
2,1
2,2

Programs with better locality will tend to have lower miss rates

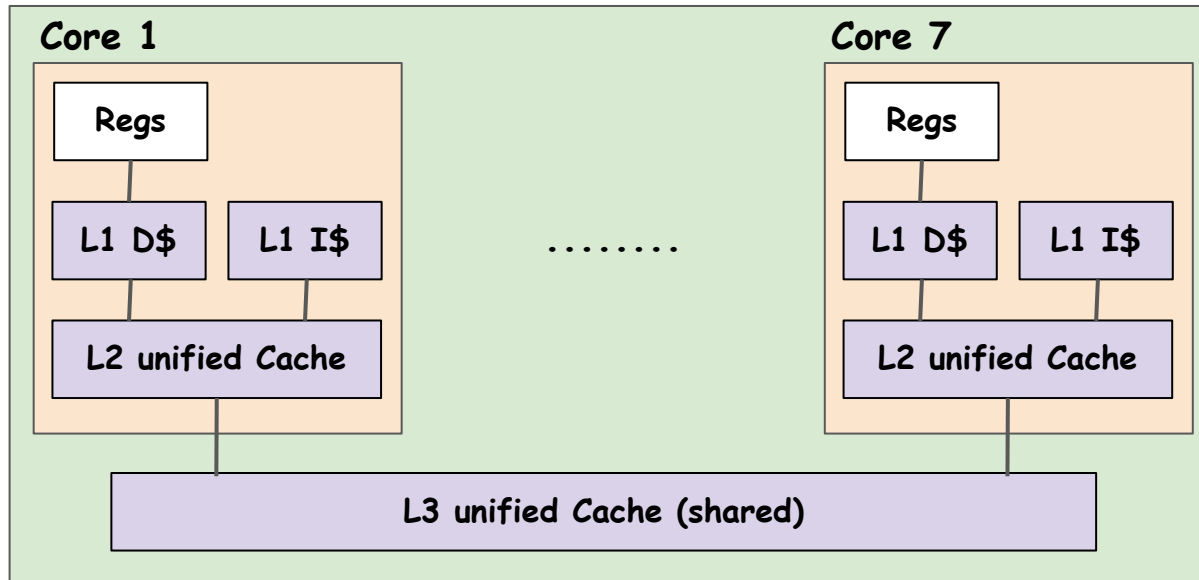
Performance

Exercise:

(Core i7 CPU)

Address length (word size) = 48 bits, BlockSize = 64 Bytes)

L1 write through, L2/L3 write back



L1 (i-cache and d-cache)

Size= 32 KB

Associativity = 4-ways

Access = 4 Cycles

L2 Cache

Size= 256 KB

Associativity = 8-ways

Access = 11 Cycles

L3 Cache

Size= 8 MB

Associativity = 16-ways

Access = 30-40 Cycles

How many bits is the index?

How many bits is the tag?