CACHE DESIGN (I)

Lecturer: Hui Annie Guo

h.guo@unsw.edu.au

K17-501F

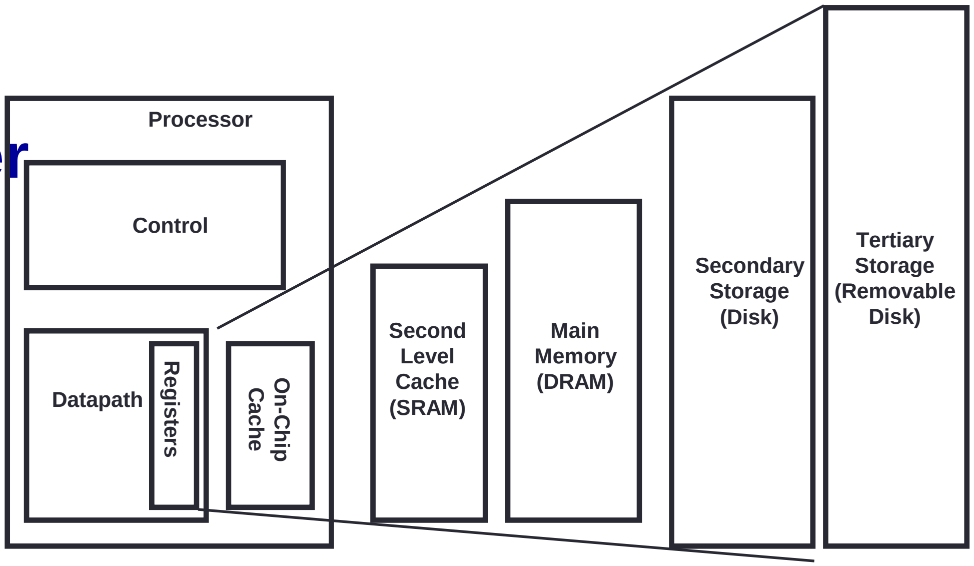
Lecture overview

- **Topics**
  - **Basic cache structures**
    - **Direct mapped cache**
    - **Fully associative cache**
    - **Set associative cache**

- **Suggested reading**
  - **H&P Chapter 5.3**

Recall: Overview of memory hierarchy

- registers ↔ memory
  - by compiler/programmer
- cache ↔ memory
  - by the hardware
- memory ↔ disks
  - by hardware and operating system (virtual memory)
  - by the programmer



Cache

- A hardware component on the processor chip
  - a smaller, faster memory
- To achieve a high cache hit rate, data need to be dynamically transferred between cache and main memory
  - Based on the principle of locality
- A data block may contain multiple bytes/words
  - Further discussion will follow

Cache (cont.)

- To control of the operation of cache, four issues should be addressed
  - Where to put a memory block in cache?
    - Block placement strategy
  - How to find a memory block in cache?
    - Block identification
  - If there are no free spaces in cache, which block will be replaced by a new memory block?
    - Block replacement
  - When memory data is updated, how is cache involved in the write operation?
    - Write strategy

Cache (cont.)

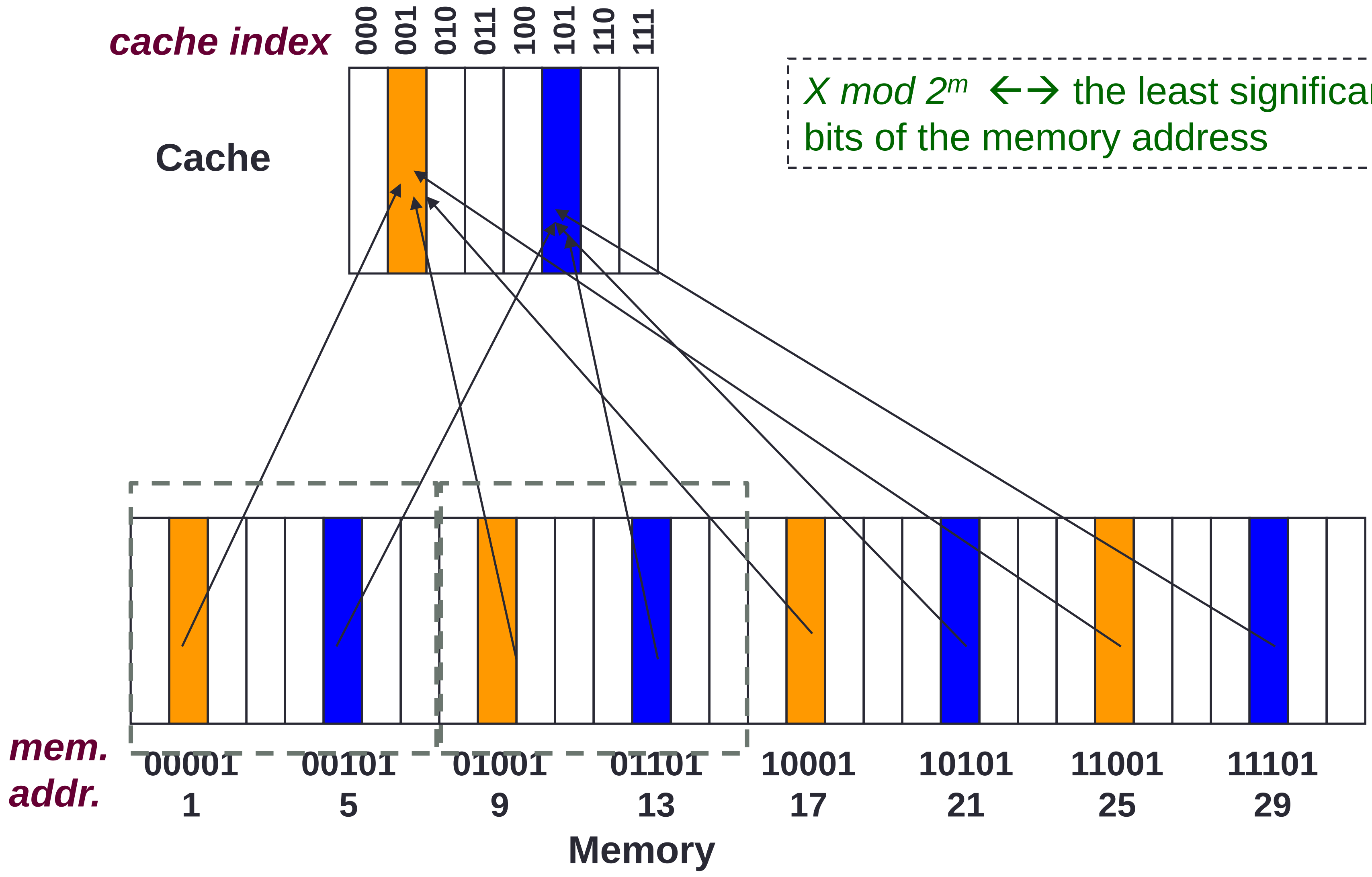
- **There are three typical cache structures**
  - **Directed mapped cache**
  - **Fully associative cache**
  - **Set associative cache**

Direct mapped cache

- **A memory block can map to one and only one cache location**
  - **For a cache of 2^m blocks, a memory block with (block) address X maps to the cache location**

    X mod 2^m
- **See an example in the next slide**

Direct mapped cache - example



X mod 2^m ←→ the least significant *m* bits of the memory address

Cache basics fields

- **A cache consists of multiple entries**
- **A cache entry has at least following fields**
  - **Valid**
    - **Indicating whether the cache entry holds valid data**
  - **Data**
    - **Storing memory data blocks**
      - A data block (cache line) is the minimum amount of memory data that can be transferred between cache and main memory
  - **Tag**
    - **Identifying the memory data block cached**

Index	V	Tag	Data
000			
001			
010			
011			
100			
101			
110			
111			

Example

- **Assume the following memory locations are accessed in sequence. How is the cache updated?**
  - **10110, 11010, 10000, 00010**

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Example (cont.)

- **Assume the following memory locations are accessed in sequence. How is the cache updated?**
  - **10110, 11010, 10000, 00010**
    - **After handling a miss on reference to address 10110**

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10	mem[10110]
111	N		

Example (cont.)

- **Assume the following memory locations are accessed in sequence. How is the cache updated?**
  - **10110, 11010, 10000, 00010**
    - **After handling a miss on reference to address 11010**

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	mem[11010]
011	N		
100	N		
101	N		
110	Y	10	mem[10110]
111	N		

Example (cont.)

- **Assume the following memory locations are accessed in sequence. How is the cache updated?**
  - **10110, 11010, 10000, 00010**
    - **After handling a miss on reference to address 10000**

Index	V	Tag	Data
000	Y	10	mem[10000]
001	N		
010	Y	11	mem[11010]
011	N		
100	N		
101	N		
110	Y	10	mem[10110]
111	N		

Example (cont.)

- **Assume the following memory locations are accessed in sequence. How is the cache updated?**
  - **10110, 11010, 10000, 00010**
    - **After handling a miss on reference to address 00010**

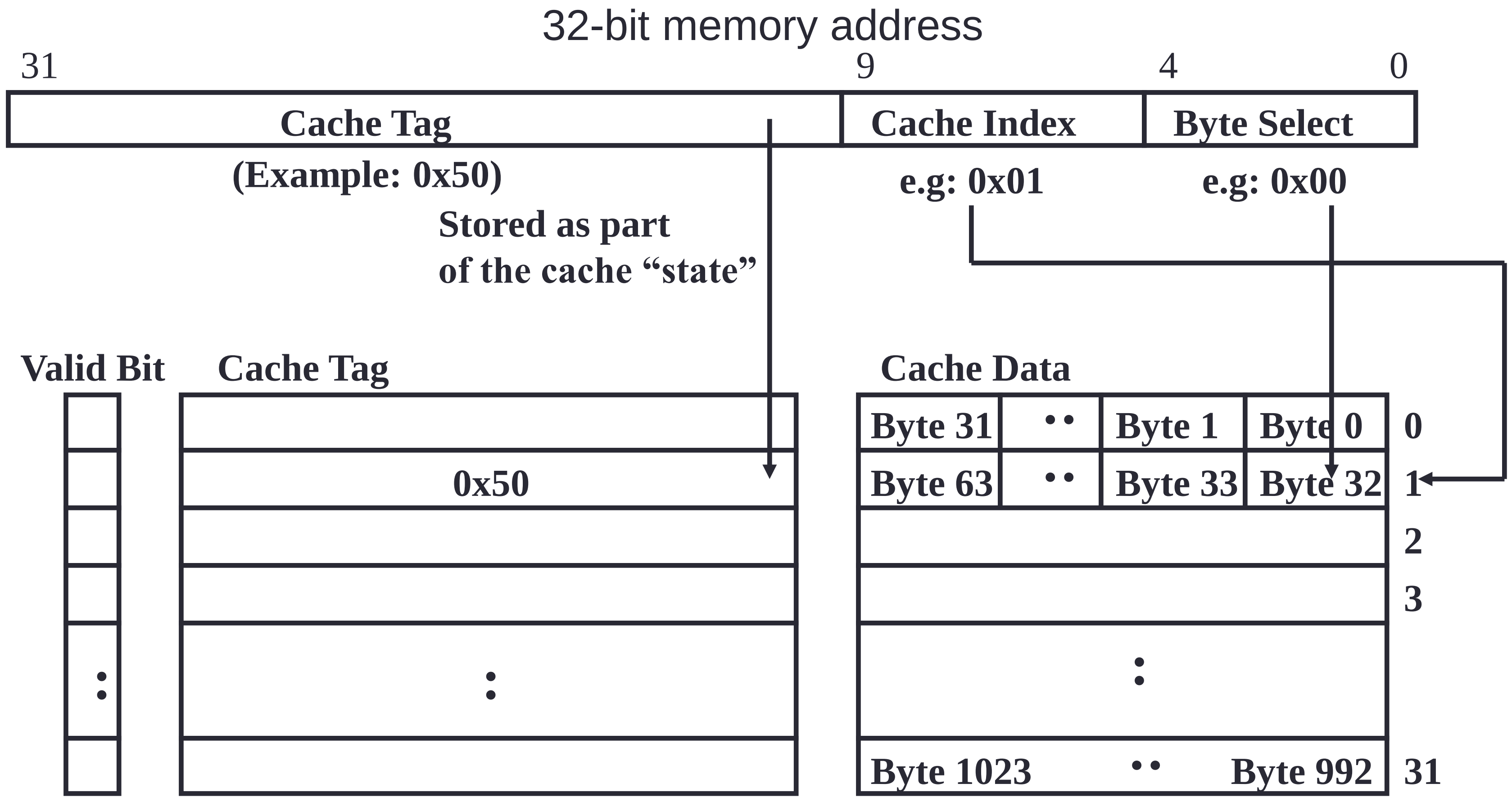
Index	V	Tag	Data
000	Y	10	mem[10000]
001	N		
010	Y	00	mem[00010]
011	N		
100	N		
101	N		
110	Y	10	mem[10110]
111	N		

Cache block

- **A cache block, aka a cache line, is the smallest section of data that can be fetched from cache**
  - **Represented by a single tag**
  - **Often contains multiple bytes/words to facilitate storing spatially related data**
  - **The size is often a power-of-two addressable units**
- **See an example in the next slide**
  - **32-byte block**

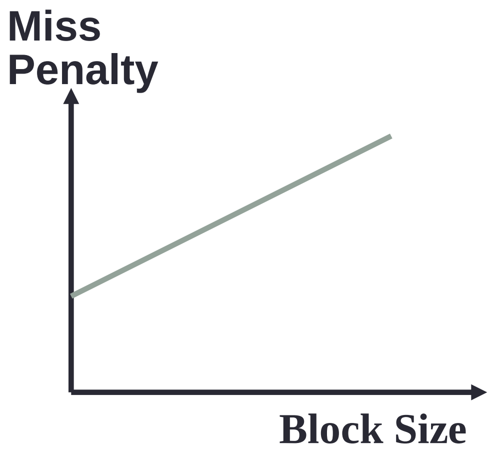
1 KB direct mapped cache with 32-byte blocks

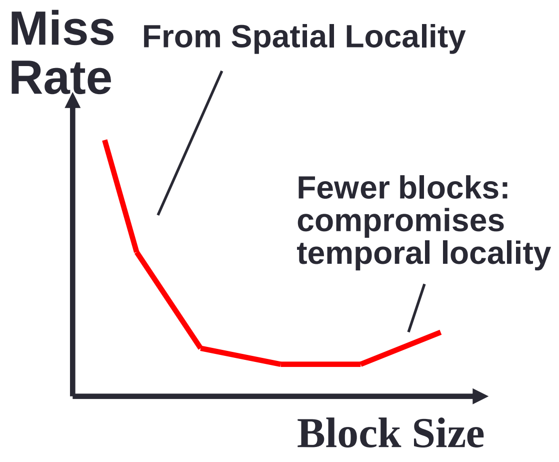
- **Give the 32-bit memory byte address**
  - **The uppermost (*32 - 10*) bits of the address are the cache tag**
  - **The lowest 5 bits are the byte select**

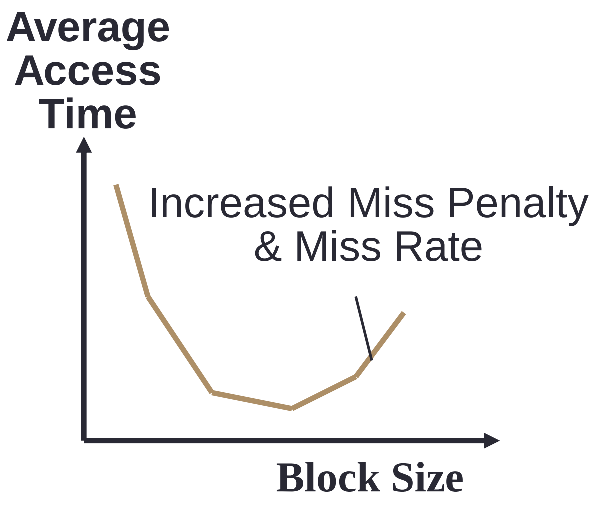


Block size

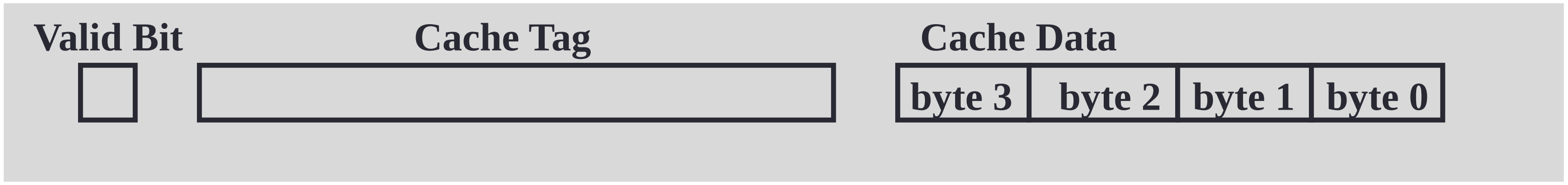
- Larger block size takes advantage of spatial locality
- But larger block size means larger miss penalties
  - It takes longer time to copy a block
  - If the block size is too big, there are too few blocks in cache and the miss rate will go up
    - One extreme case is discussed in the next slide
- Trade-off should be played based on the average access time
  - Hit Time + Miss Penalty x Miss Rate







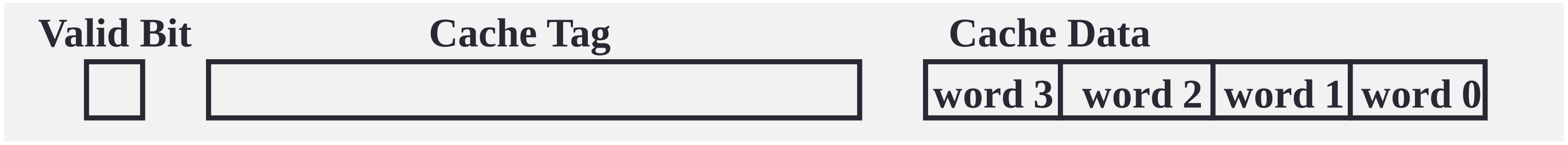
One extreme case: 1-block cache



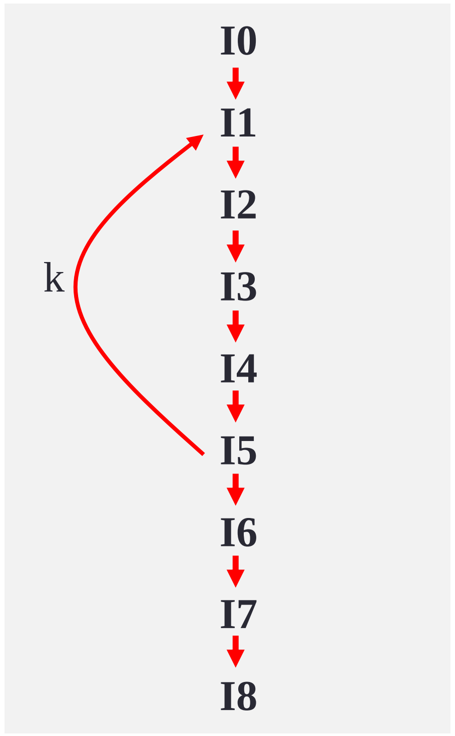
- **Only one entry in the cache**
  - **cache size = 1 cache block**
- **Potential problem?**
  - **Ping Pong effect/thrashing**
    - **The locality says** if a memory data item is accessed, it will likely be accessed again soon**. But it is unlikely that it will be accessed again immediately!!!**
    - **Causes large <u>conflict misses</u>**
      - Will be discussed later
    - **An example is given in the next slide**

Ping-Pong effect - example

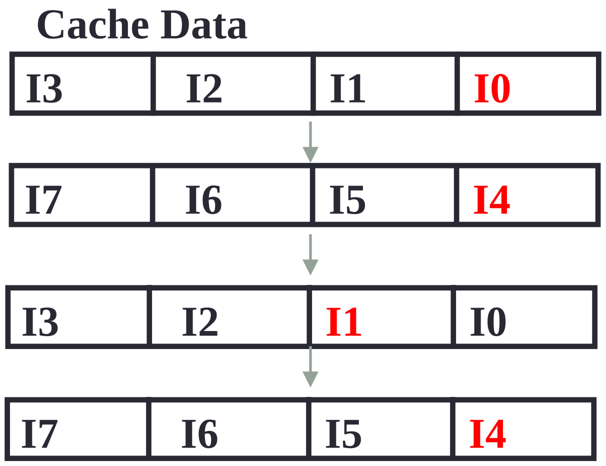
- 1-block cache for instruction memory



- Instruction execution flow
  - instruction size: 1 word



- Cache data field contents



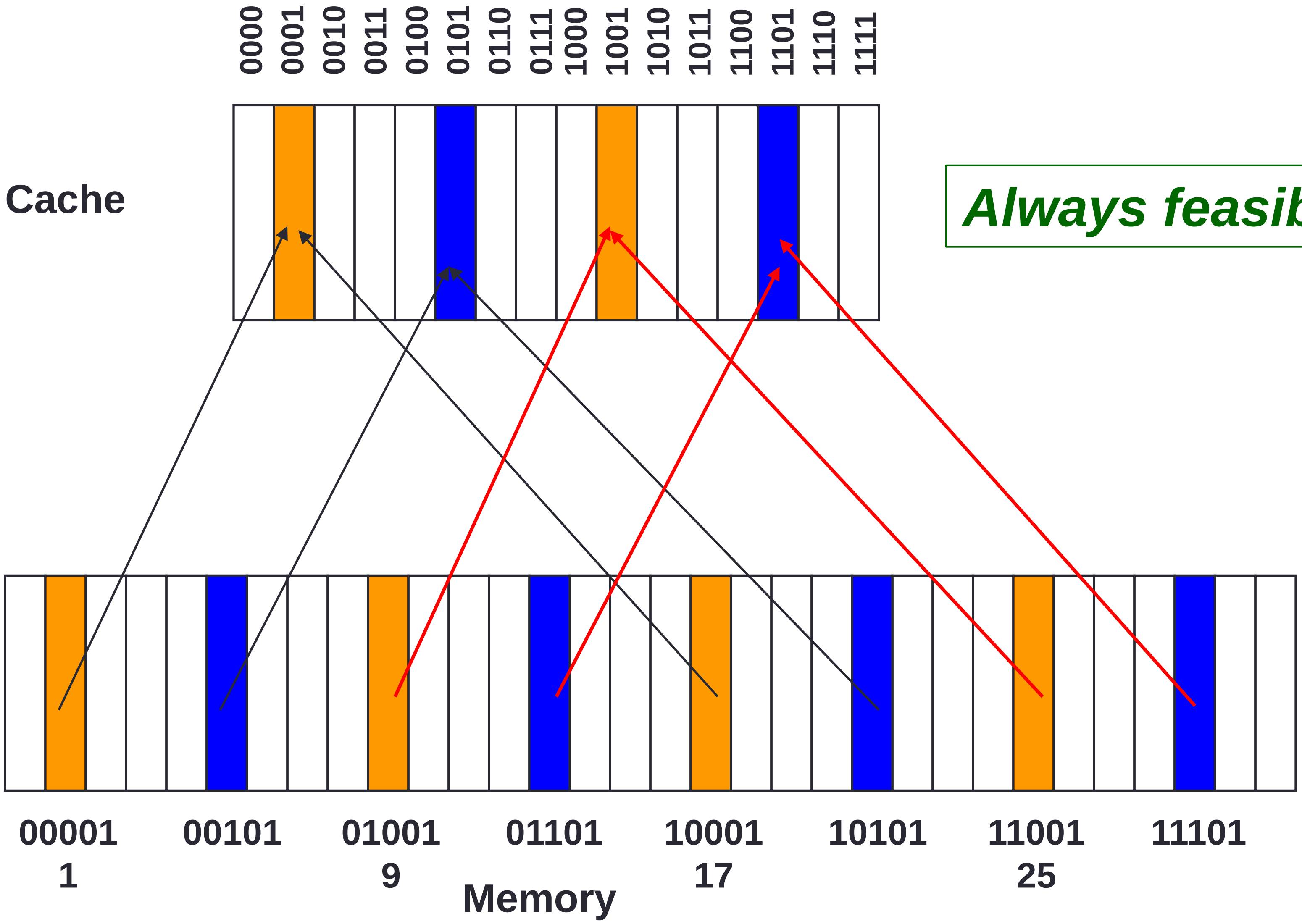
I3	I2	I1	I0
I7	I6	I5	I4
I3	I2	I1	I0
I7	I6	I5	I4

Class exercise

- **If the cache shown in the previous slide is re-structured into two blocks with the same cache size, how are the cache contents changed?**

Solution 1: making the cache size bigger

- **Example, double the cache size**



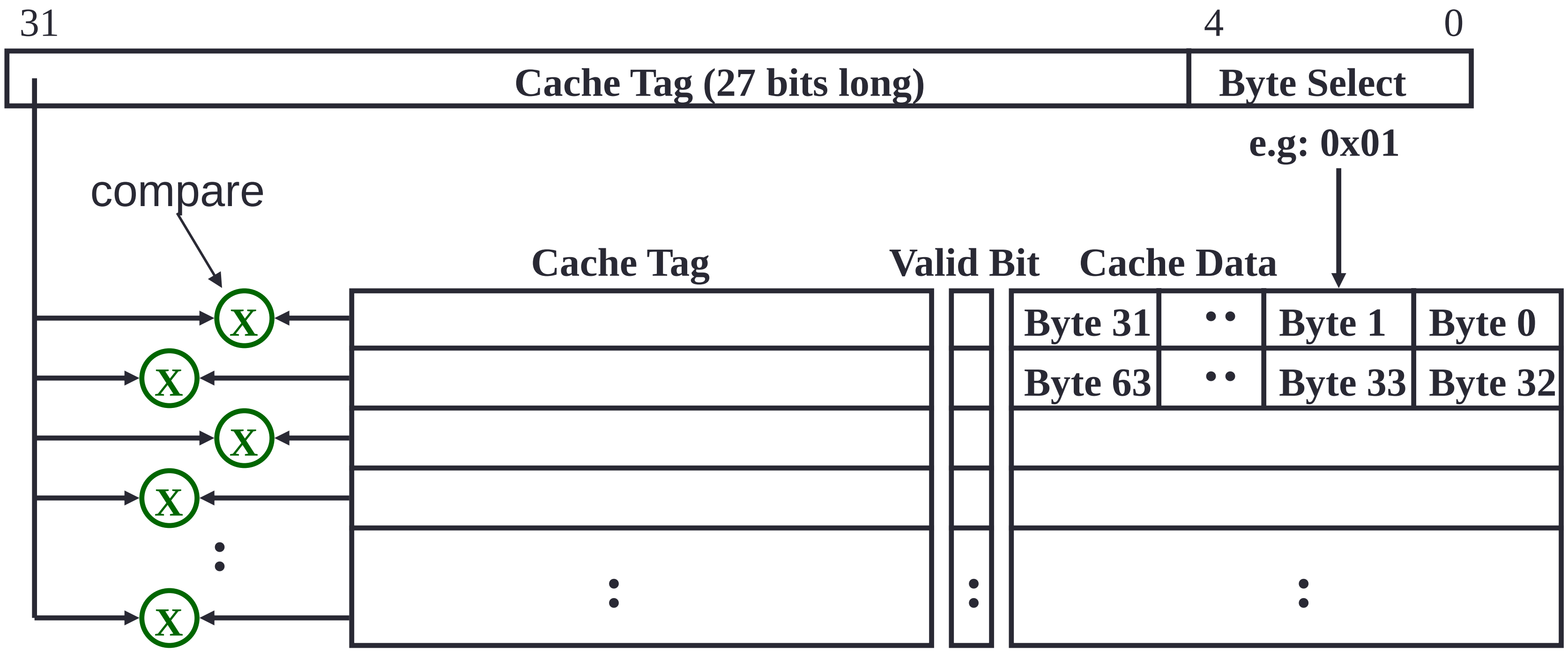
Always feasible?

Solution 2 – multiple entries for a memory block

- **Fully associative cache**
- **Set associative cache**

Fully associative cache

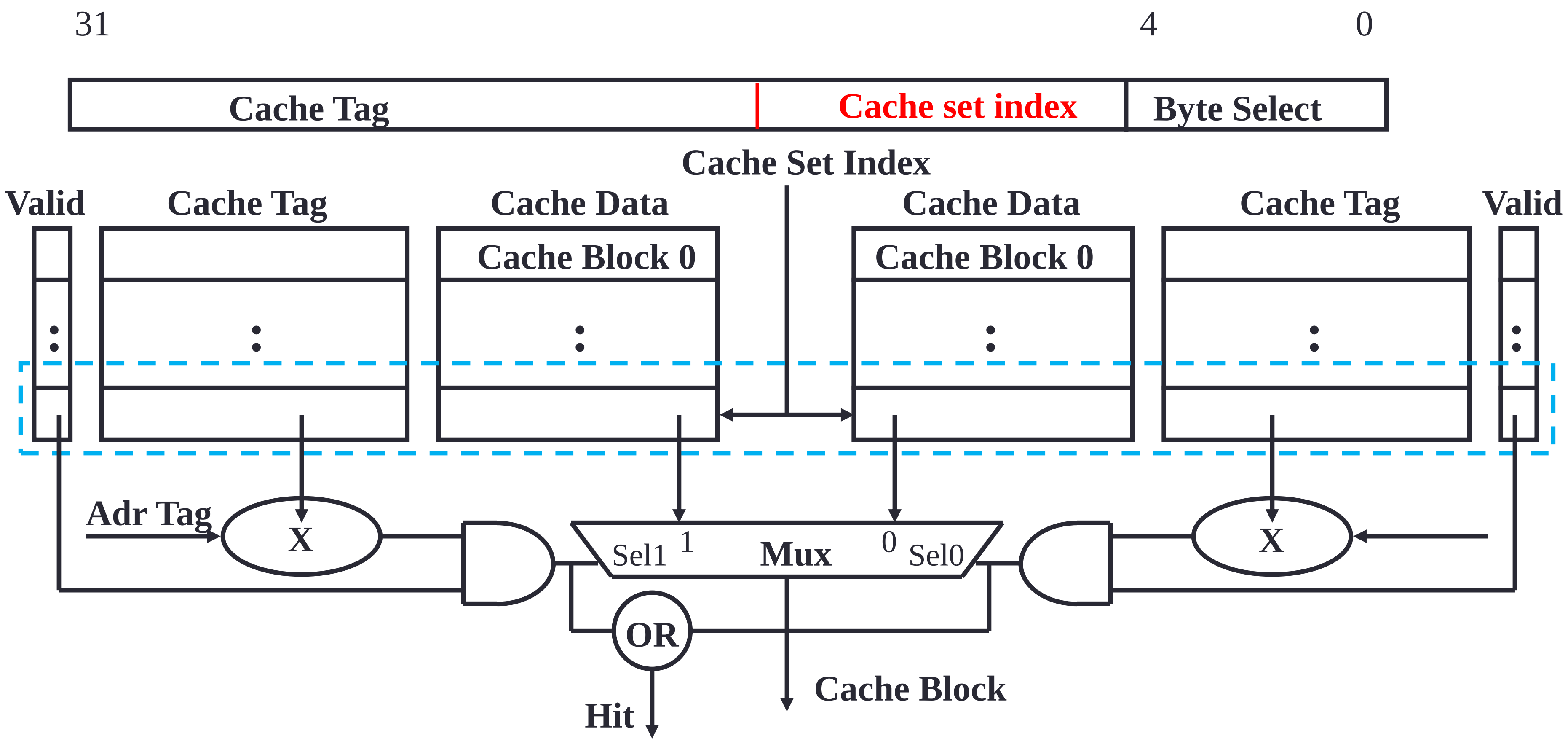
- **A memory block can be mapped to any location in the cache.**
- **Full cache needs to be searched for a memory block**
  - **Assume cache size is *M* blocks. We need *M* comparators for a fully associative cache -- costly**



n-way set associative cache

- Cache is divided into sets
- Each set has *n* blocks
  - A memory block can be mapped to a set and can be stored in any location in the set
- *n* comparators are required to search a block in a set
- An example is given in the next slide

2-way set associative cache



In-class exercise 1

Give an 8-block cache, how is the memory block 12 (i.e. 12 is the block address) is placed in the cache? Assume the cache is

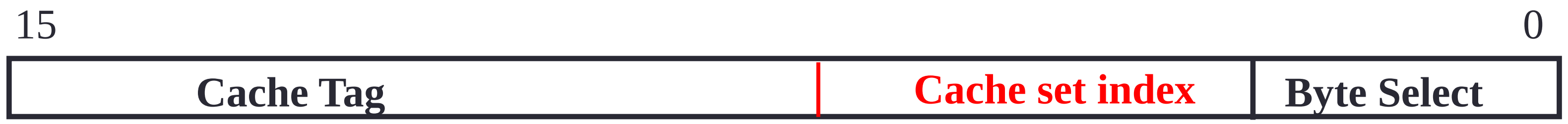
1) **Fully associative**
2) **Direct mapped**
3) **2-way set associative**

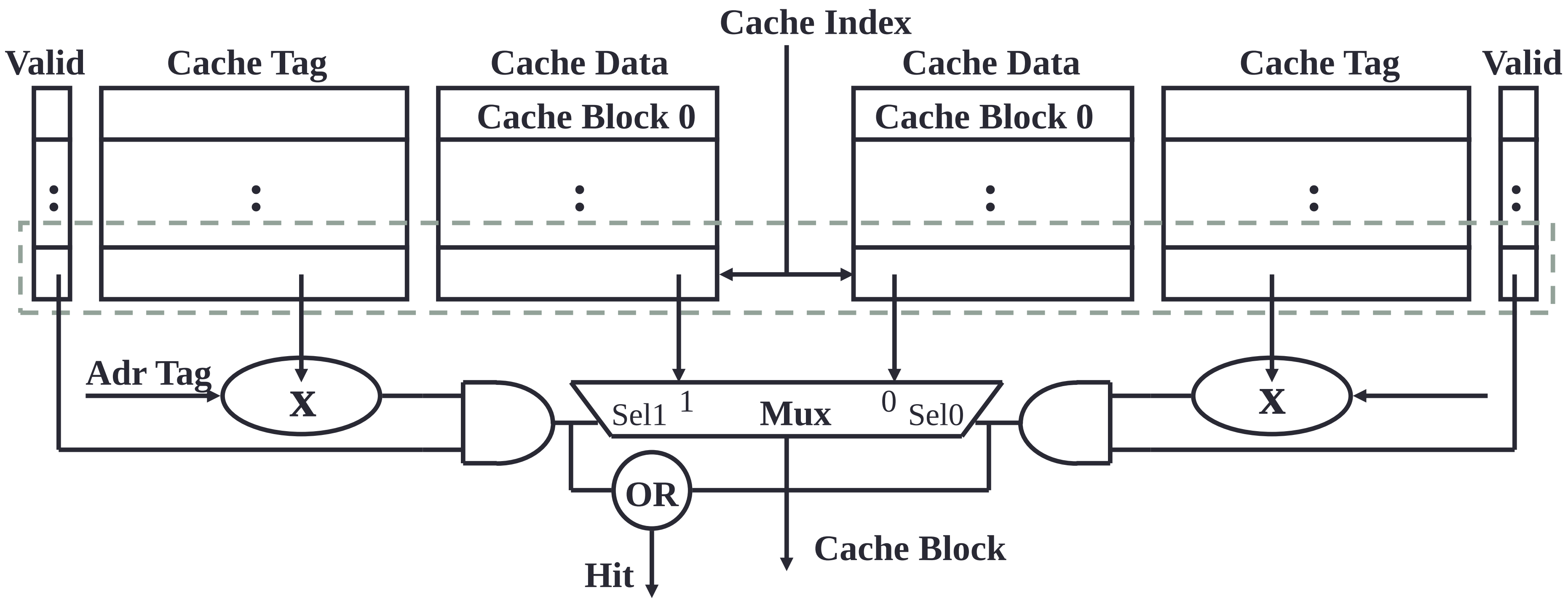
In-class exercise 2

Given a 2-way set associative cache that has

- **Block size: 4 bytes**
- **Cache size: 64 bytes**

How to find whether Mem(0x001B) is cached?


15
0
Cache Tag
Cache set index
Byte Select


Cache Index
Valid
Cache Tag
Cache Data
Cache Block 0
Cache Data
Cache Block 0
Cache Tag
Valid
Adr Tag
X
Sel1
1
Mux
0
Sel0
X
OR
Hit
Cache Block