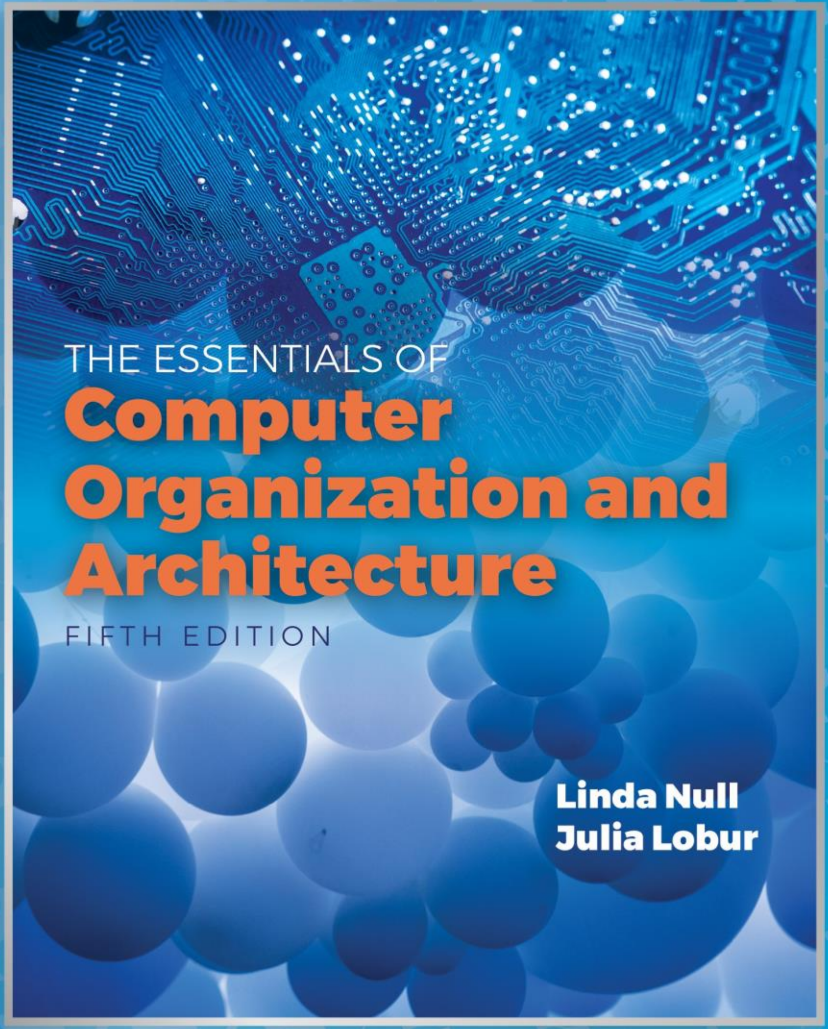
This is the second lecture of Chapter 6

Chapter 6

Memory (B)

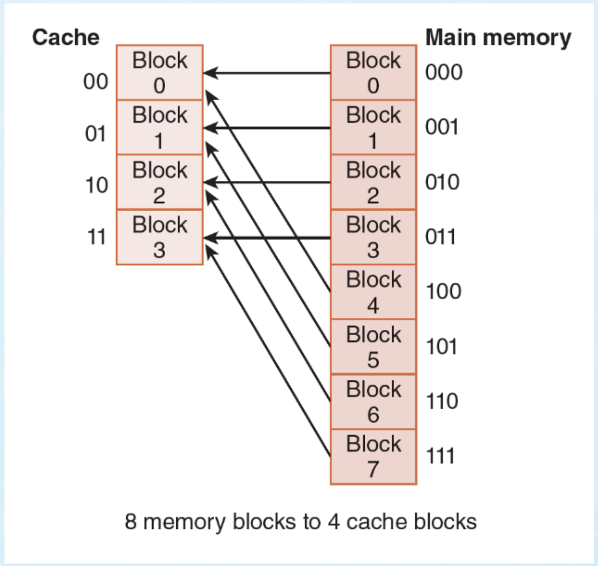


Quick review of last lecture

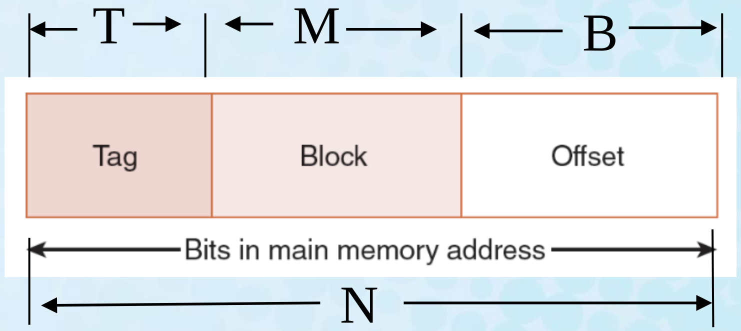
- Types of Memory
  - DRAM, SRAM, ROM
- The Memory Hierarchy
  - Registers, Cache, Main Memory, Virtual Memory
  - Data transfer to/from registers in words
  - Data transfer between Cache and Main Memory in blocks
  - Terminologies:
    - hit, miss, hit rate, miss rate, hit time, miss penalty
  - Principle of locality
- Cache Placement Schemes
  - Direct Mapping

6.4 Cache Memory

With direct mapped cache consisting of 4 blocks of cache, block *X* of main memory maps to cache block *Y* = *X* mod 4.



8 memory blocks to 4 cache blocks



Memory size = 2^N

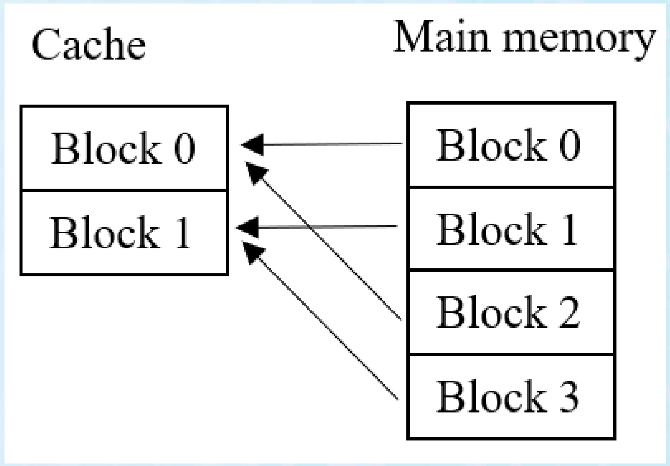
# of Cache blocks = 2^M

Block size = 2^B

$T = N - M - B$

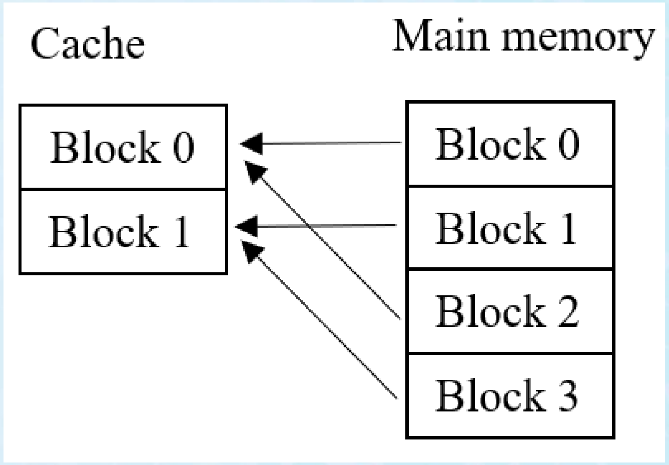
6.4 Cache Memory (6 of 45)

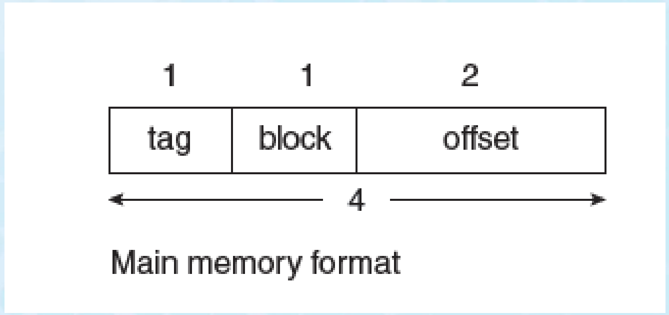
- Example 6.1: Consider a byte-addressable main memory consisting of 4 blocks, and a cache with 2 blocks, where each block is 4 bytes.
- This means Block 0 and 2 of main memory map to Block 0 of cache, and Blocks 1 and 3 of main memory map to Block 1 of cache.
- Using the tag, block, and offset fields, we can see how main memory maps to cache as follows.



6.4 Cache Memory (7 of 45)

- Example 6.1: Cont'd. Consider a byte-addressable main memory consisting of 4 blocks, and a cache with 2 blocks, where each block is 4 bytes.
- First, we need to determine the address format for mapping.
  - Main memory address has 4 bits because there are a total of $2^4 = 16$ bytes
  - Each block is 4 bytes, so the offset field must contain 2 bits;
  - There are 2 blocks in cache, so the block field must contain 1 bit;
  - This leaves 1 bit for the tag.





6.4 Cache Memory (8 of 45)

- Example 6.1: Cont'd.
  - Suppose we need to access main memory address 3_{16} (0x0011 in binary). If we partition 0x0011 using the address format from Fig. a, we get Fig. b.
  - Thus, the main memory address 0x0011 maps to cache block 0.
  - Figure c shows this mapping, along with the tag that is also stored with the data.

Fig. a

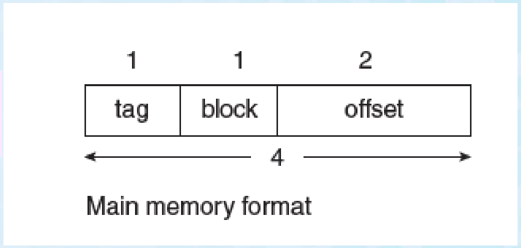


Fig. b

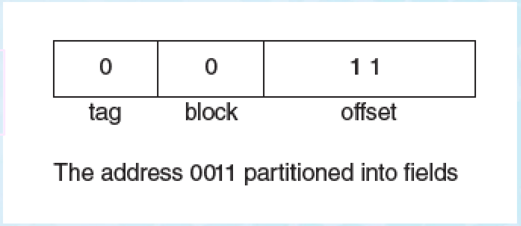
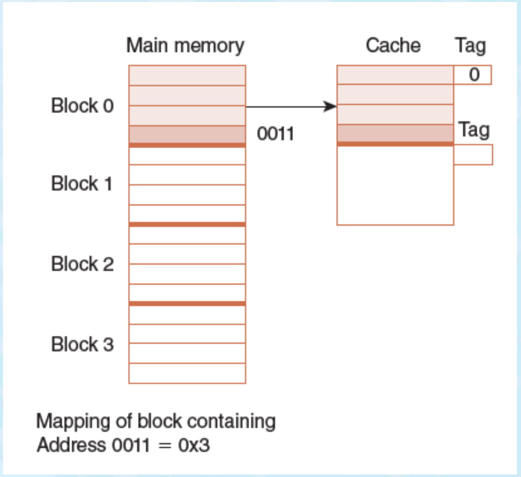
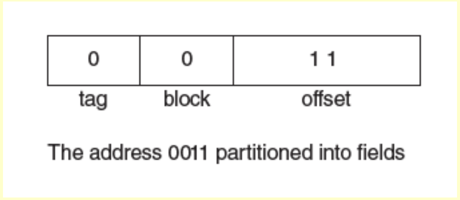


Fig. c

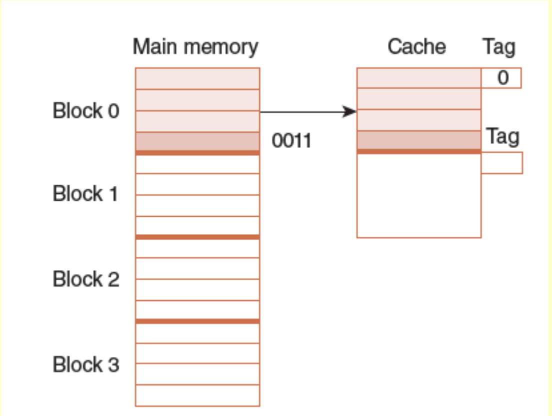


The next slide illustrates another mapping.

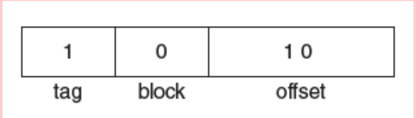
6.4 Cache Memory (9 of 45)



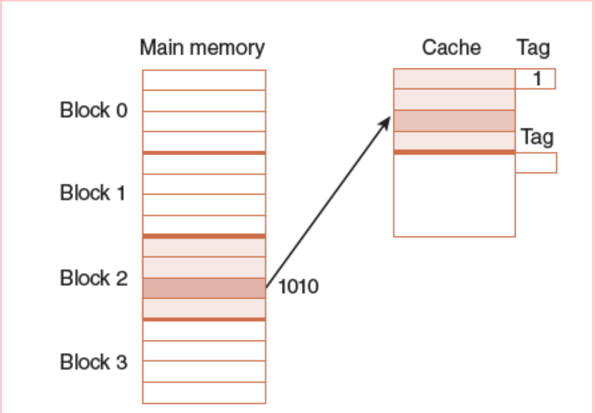
The address 0011 partitioned into fields



Mapping of block containing
Address 0011 = 0x3



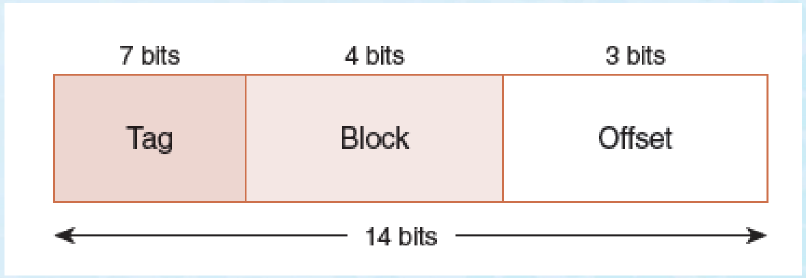
The address 1010 partitioned into fields



Mapping of Block Containing
Address 1010 = 0xA

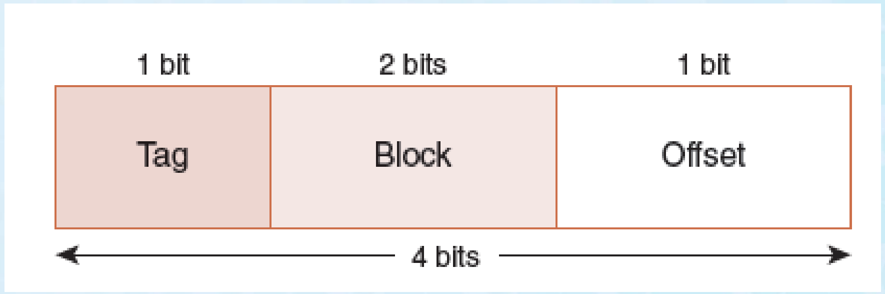
6.4 Cache Memory (10 of 45)

- Example 6.2: Assume a byte-addressable memory consists of 2^{14} bytes, cache has $16 = 2^4$ blocks, and each block has $8 = 2^3$ bytes.
  - The number of memory blocks are: $\frac{2^{14}}{2^3} = 2^{11}$
  - Each main memory address requires 14 bits.
  - The offset field in the rightmost contains 3 bits.
  - The block field consists of the middle 4 bits to select a specific block in cache.
  - The remaining 7 bits make up the tag field.



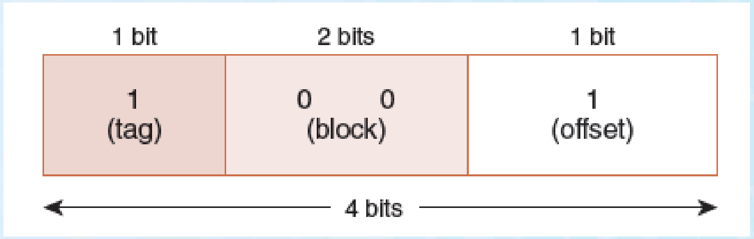
6.4 Cache Memory (11 of 45)

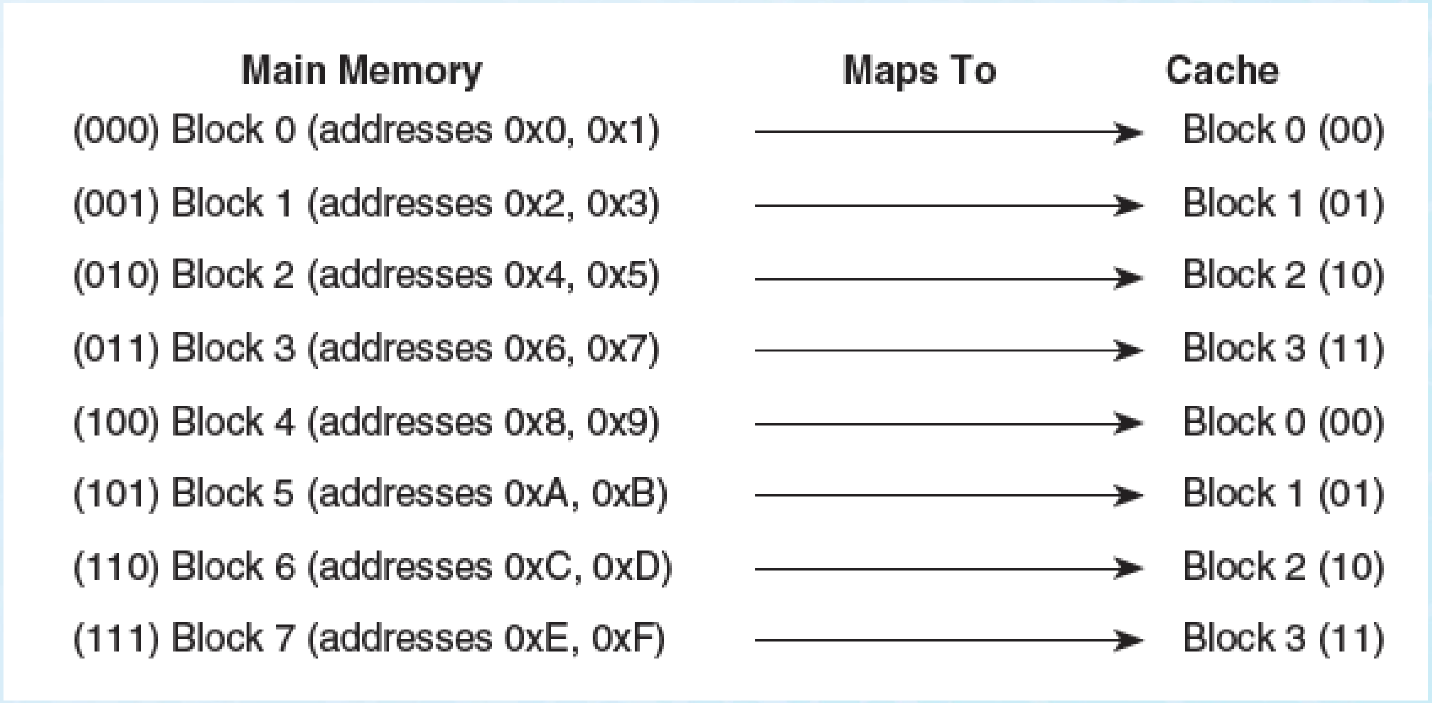
- Example 6.3: Assume a byte-addressable memory consisting of 16 = 2^4 bytes divided into 8 blocks. Cache contains 4 = 2^2 blocks. We know:
  - A memory address has 4 bits.
  - The 4-bit memory address is divided into the fields below.



6.4 Cache Memory (12 of 45)

- Example 6.3: Cont'd. The mapping for memory references is shown below:

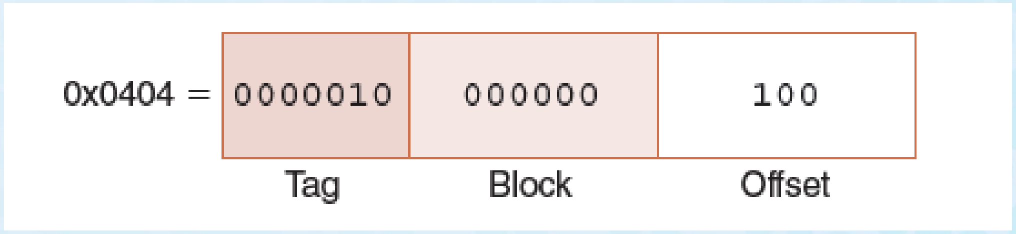




Main Memory	Maps To	Cache
(000) Block 0 (addresses 0x0, 0x1)	→	Block 0 (00)
(001) Block 1 (addresses 0x2, 0x3)	→	Block 1 (01)
(010) Block 2 (addresses 0x4, 0x5)	→	Block 2 (10)
(011) Block 3 (addresses 0x6, 0x7)	→	Block 3 (11)
(100) Block 4 (addresses 0x8, 0x9)	→	Block 0 (00)
(101) Block 5 (addresses 0xA, 0xB)	→	Block 1 (01)
(110) Block 6 (addresses 0xC, 0xD)	→	Block 2 (10)
(111) Block 7 (addresses 0xE, 0xF)	→	Block 3 (11)

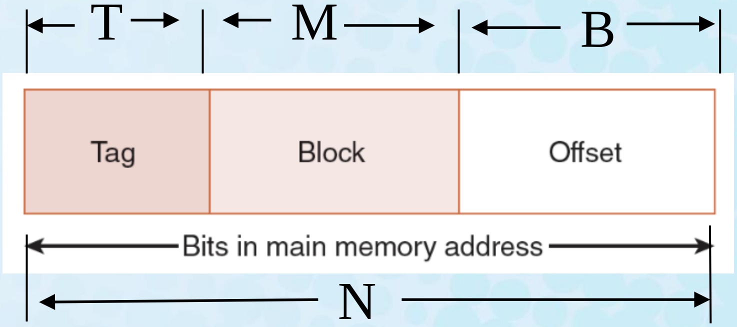
6.4 Cache Memory (13 of 45)

- Example 6.4: Consider 16-bit memory addresses and 64 blocks of cache where each block contains 8 bytes. We have:
  - 3 bits for the offset
  - 6 bits for the block
  - 7 bits for the tag
- A memory reference for 0x0404 maps as follows:



6.4 Cache Memory (14 of 45)

- In summary, direct mapped cache maps main memory blocks in a modular fashion to cache blocks. The mapping depends on:
  - The number of bits in the main memory address (how many addresses exist in main memory).
  - The number of blocks are in cache (which determines the size of the block field).
  - How many addresses (either bytes or words) are in a block (which determines the size of the offset field)?



Memory size = 2^N

# of Cache blocks = 2^M

Block size = 2^B

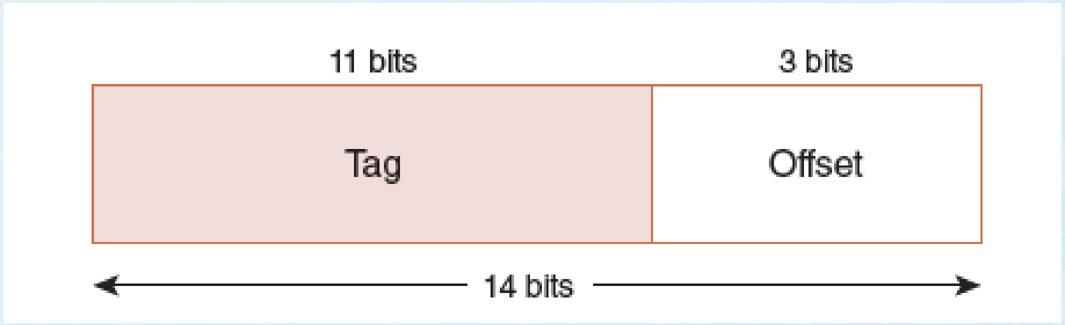
$T = N – M – B$

6.4 Cache Memory (15 of 45)

- Suppose instead of placing memory blocks in specific cache locations based on memory address, we could allow a block to go anywhere in cache.
- In this way, cache would have to fill up before any blocks are evicted.
- This is how *fully associative* cache works.
- A memory address is partitioned into only two fields: the tag and the offset.

6.4 Cache Memory (16 of 45)

- Suppose, as before, we have 14-bit memory addresses and a cache with 16 blocks, each block of size 8. The field format of a memory reference is:



- When the cache is searched, all tags are searched in parallel to retrieve the data quickly.
- This requires special, costly hardware.

6.4 Cache Memory (17 of 45)

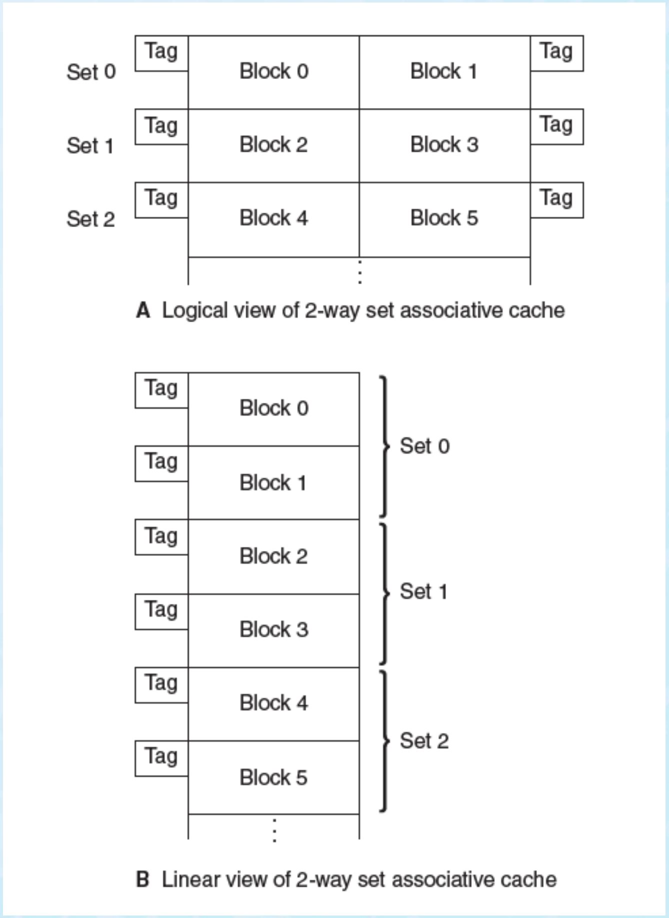
- You will recall that direct mapped cache evicts a block whenever another memory reference needs that block.
- With fully associative cache, we have no such mapping, thus we must devise an algorithm to determine which block to evict from the cache.
- The block that is evicted is the *victim block*.
- There are a number of ways to pick a victim, we will discuss them shortly.

6.4 Cache Memory (18 of 45)

- Set associative cache combines the ideas of direct mapped cache and fully associative cache.
- An *N*-way set associative cache mapping is like direct mapped cache in that a memory reference maps to a particular location in cache.
- Unlike direct mapped cache, a memory reference maps to a set of several cache blocks, similar to the way in which fully associative cache works.
- Instead of mapping anywhere in the entire cache, a memory reference can map only to the subset of cache slots.

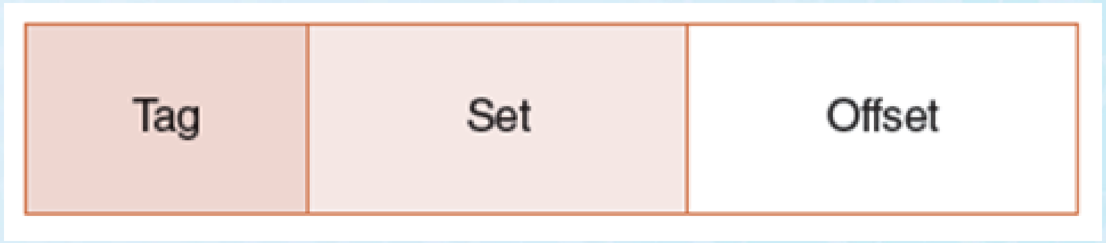
6.4 Cache Memory (19 of 45)

- The number of cache blocks per set in set associative cache varies according to overall system design.
  - For example, a 2-way set associative cache can be conceptualized as shown in the schematic below.
  - Each set contains two different memory blocks.


Set 0
Set 1
Set 2
Tag
Block 0
Block 1
Block 2
Block 3
Block 4
Block 5
A Logical view of 2-way set associative cache
B Linear view of 2-way set associative cache

6.4 Cache Memory (20 of 45)

- In set associative cache mapping, a memory reference is divided into three fields: tag, set, and offset.
- As with direct-mapped cache, the offset field chooses the byte within the cache block, and the tag field uniquely identifies the memory address.
- The set field determines the set to which the memory block maps.


Tag
Set
Offset