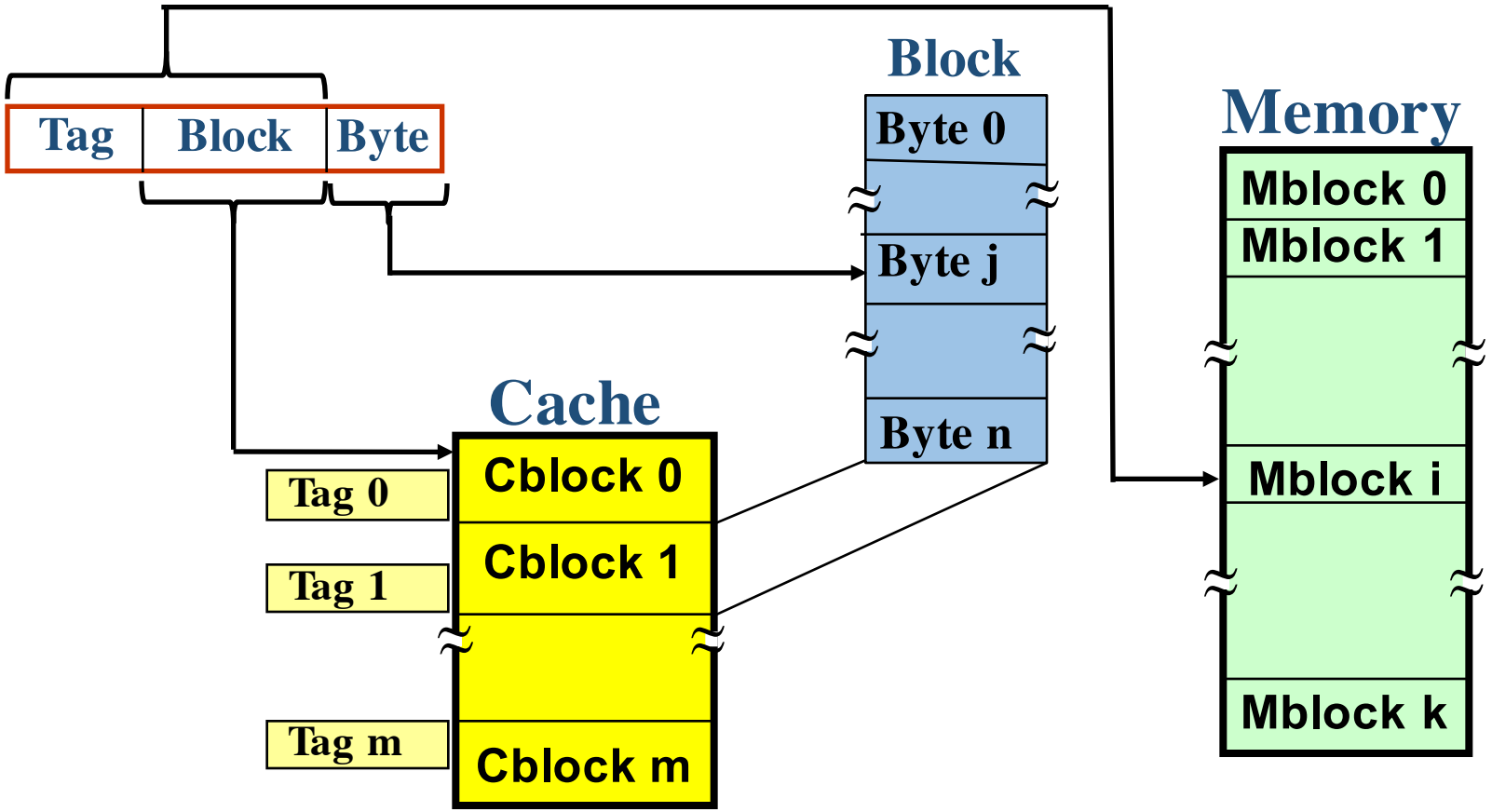
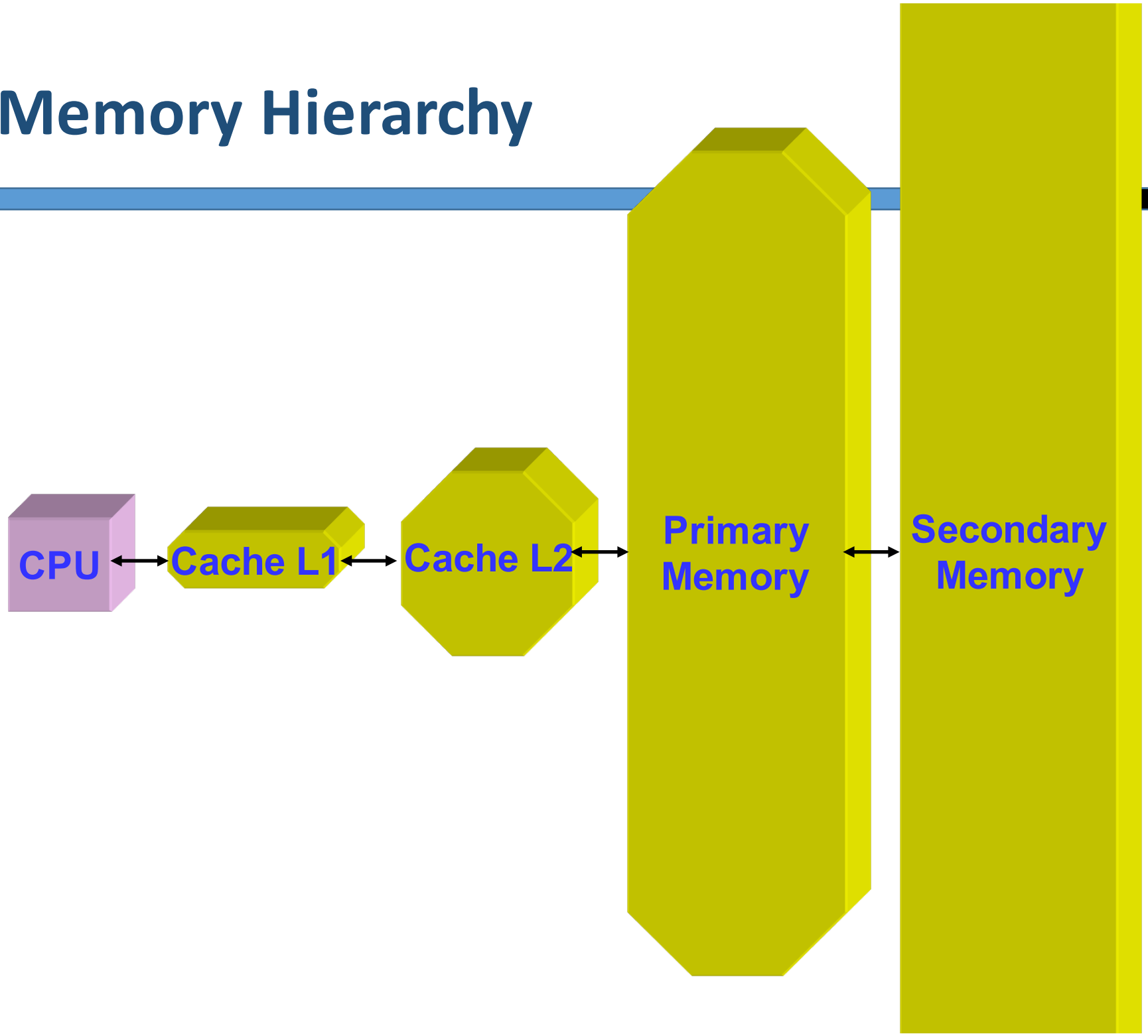
Caches: Fundamentals



On this Lesson

- Overview of memory hierarchy
- Why is a cache effective?
- Organization of a cache
- Cache configurations
  - *Direct Mapped*
  - *Fully Associative*
  - *Block-Set Associative*
- Cache hits and misses

Memory Hierarchy



The Cache

- Is the mechanism used to compensate for the speed difference between the CPU and primary memory (memory latency)
- Is faster and smaller than primary memory
- Maintains the values of the most frequently accessed memory locations
- Is usually invisible to the CPU (non architectural)

Effectiveness of the Cache

Is due to two factors:

- ***Temporal Locality***

  In a given period of time memory accesses go to a small group of repeated memory locations.

  This mostly due to loops that repeat accesses to loop variables and instructions fetches.

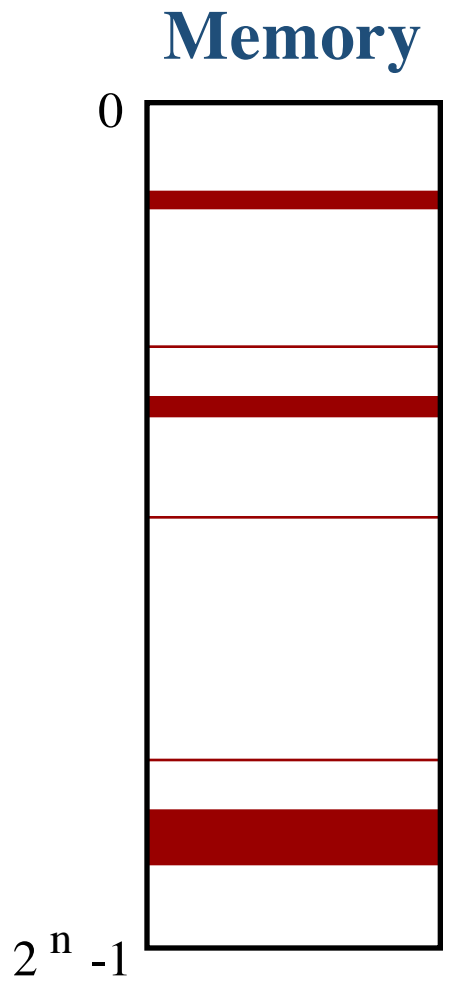
- ***Spatial locality***

  In a given period of time memory accesses go to adjacent memory locations

  This mostly due to loops that execute adjacent sequential instructions, and loop variables that are allocated contiguous memory locations.

Memory Access Pattern Generated by most Programs

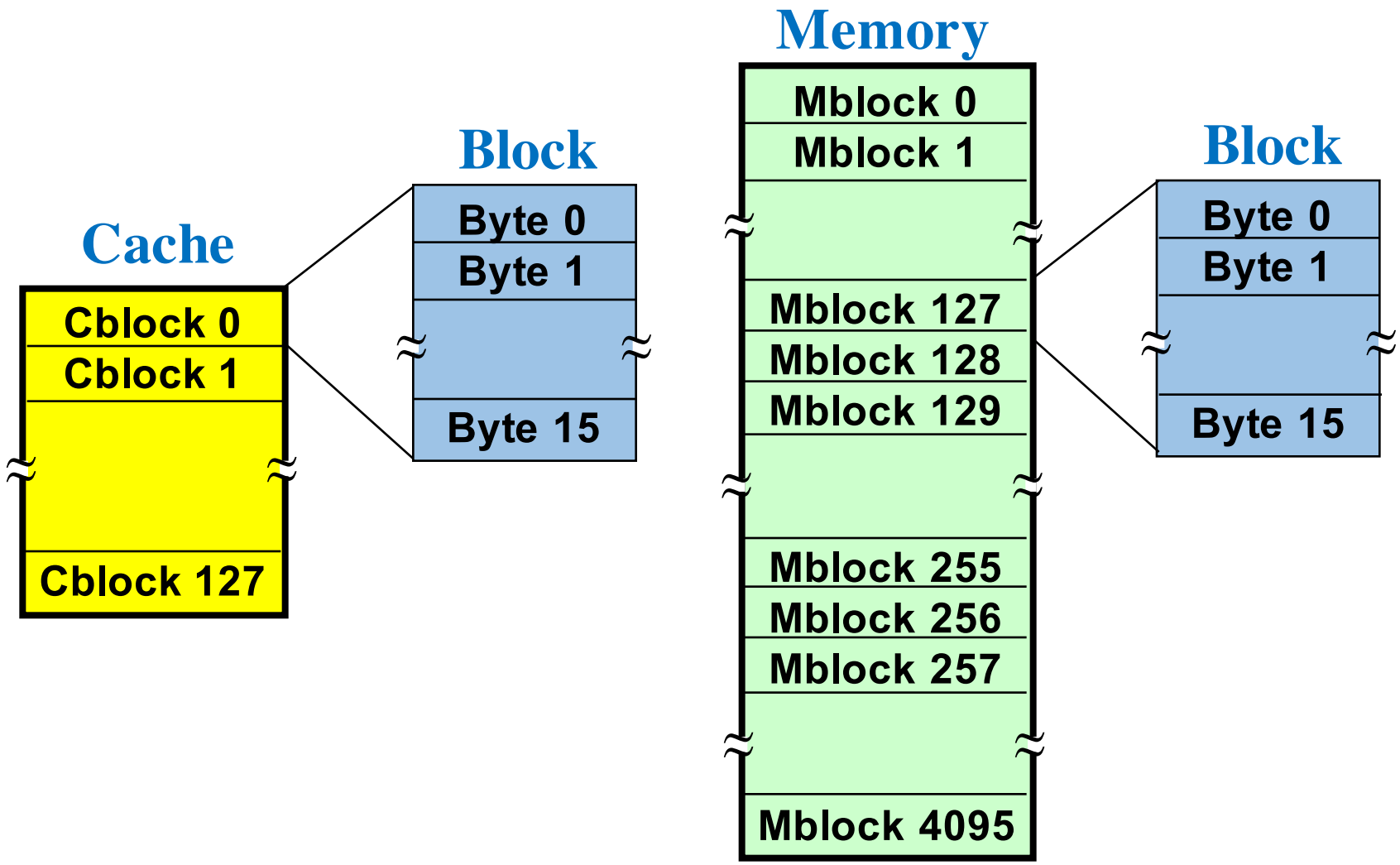
On a given period of time most of the memory accesses go to blocks of contiguous memory locations that are repeated.



Cache and Memory Organization

- Caches allocate a portion of the content of main memory.
- The data allocated in the cache is organized in blocks (cache lines) of 2^k bytes.
- The cache view main memory also as organized in blocks of 2^k bytes
- The cache intercepts memory access from the CPU directed to the main memory (loads, stores, fetches)
- CPU addresses are intended for a memory array of 2^n bytes but the caches translate them to its block organization

Organization of a 2^7-block Cache of 2^4 Bytes and a 2^{16}-byte Memory



CACHE Configurations

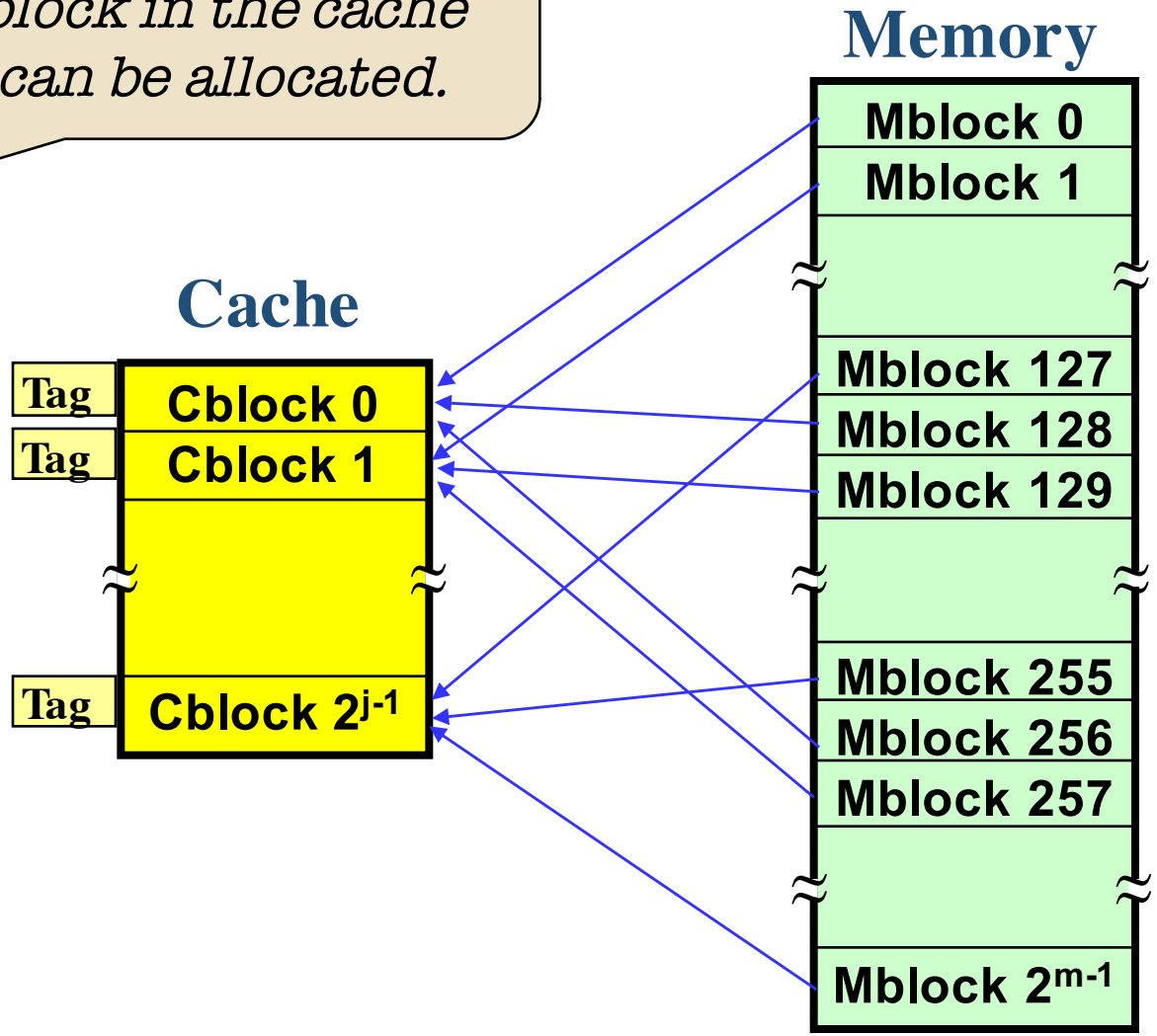
- There are three cache configurations:
  - *Direct Mapped*
  - *Fully Associative*
  - *Block-Set Associative*
- They differ on how the blocks from main memory are allocate into the cache
- The address provided by the CPU is broken into fields according to the cache configuration.
- The cache management system must determine if the CPU address belongs to a block of memory allocated in the cache (a **hit**) or not (a **miss**).

Hit and Miss Management

- If there is a hit:
  - *The CPU memory request is served from the corresponding block allocated in the cache*
- In there is a miss
  - *The cache must transfers the corresponding block from memory an places it to the corresponding block in the cache.*
  - *The CPU memory request is served from the newly allocated block in the cache*

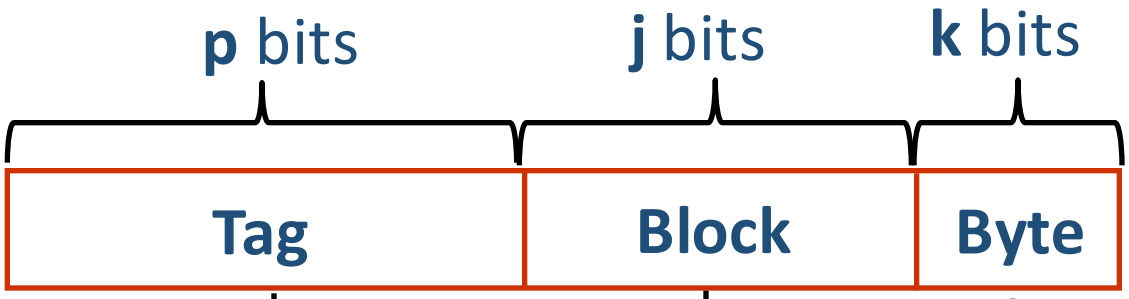
Direct Mapped Cache

Every memory block has a specific block in the cache where it can be allocated.



Direct Mapped Cache: CPU Address Mapping

CPU Address of n bits (memory of 2^n bytes**)**



Byte within cache block
(2^k bytes per block)

Block within cache
(2^j blocks on the cache)

Identifier of memory block allocated on the cache block
(one of 2^p blocks can be allocated on each cache block)

2^{n-k} blocks on memory

Determining Address Mapping Fields for Direct-Mapped Caches

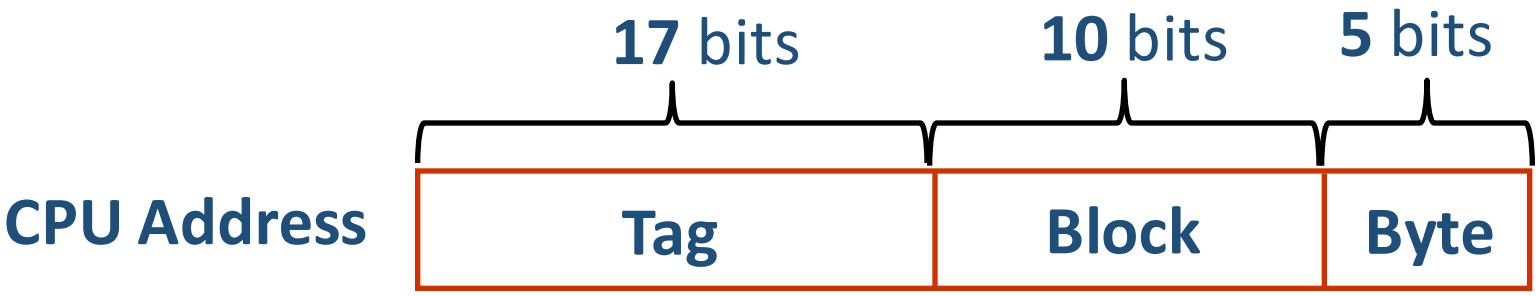
Memory of 2^{32} bytes, cache of 1024 blocks of 32 bytes each

Memory size = 2^{32}, then **n**=32

Number of cache blocks = 1024 = 2^{10}, then **j**=10

Block size = 32 = 2^5, then **k**=5

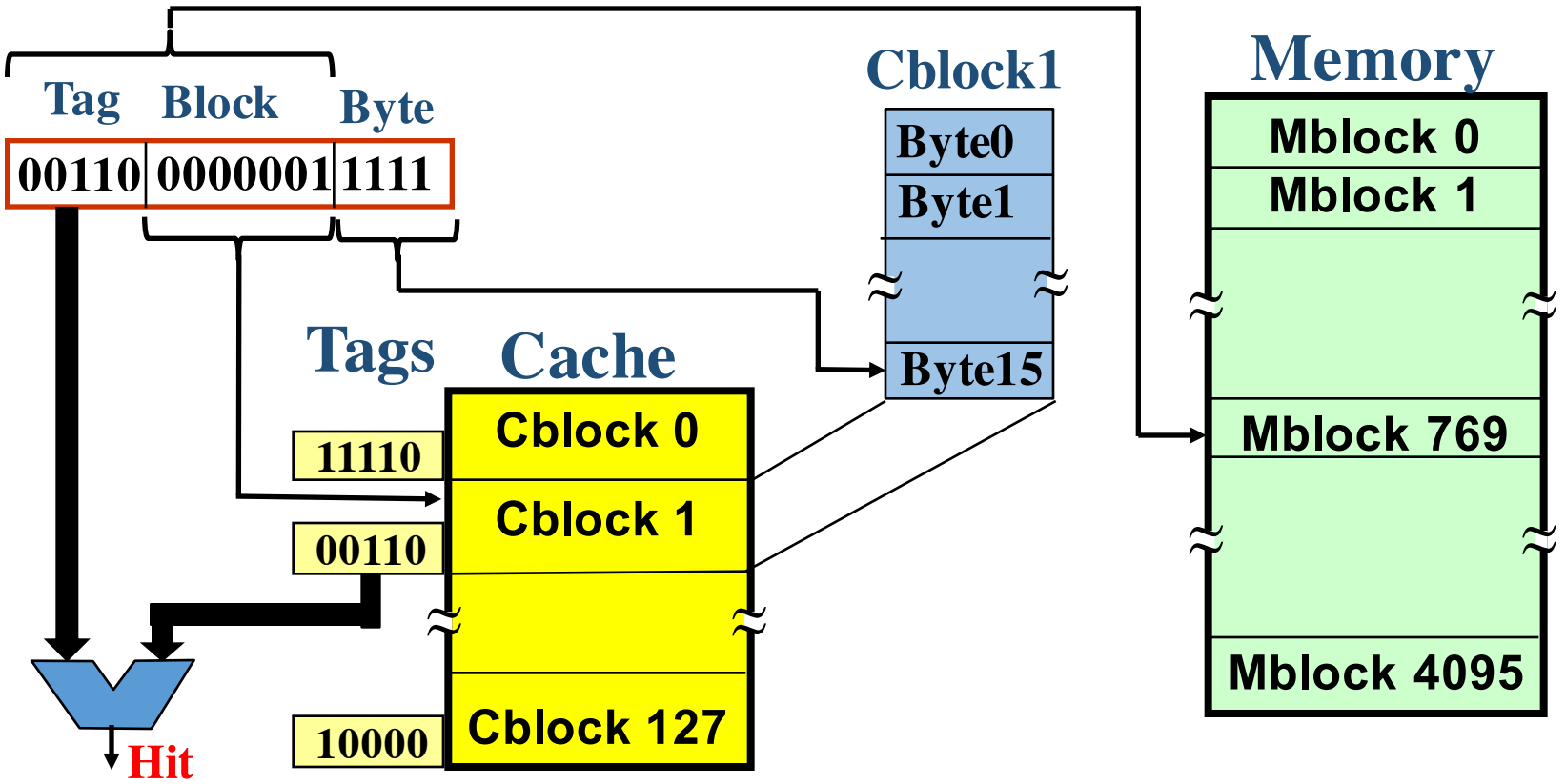
Then, **p** = **n** - (**j** + **k**) = 32 – (10 + 5) = 17



Direct Mapped Cache Example

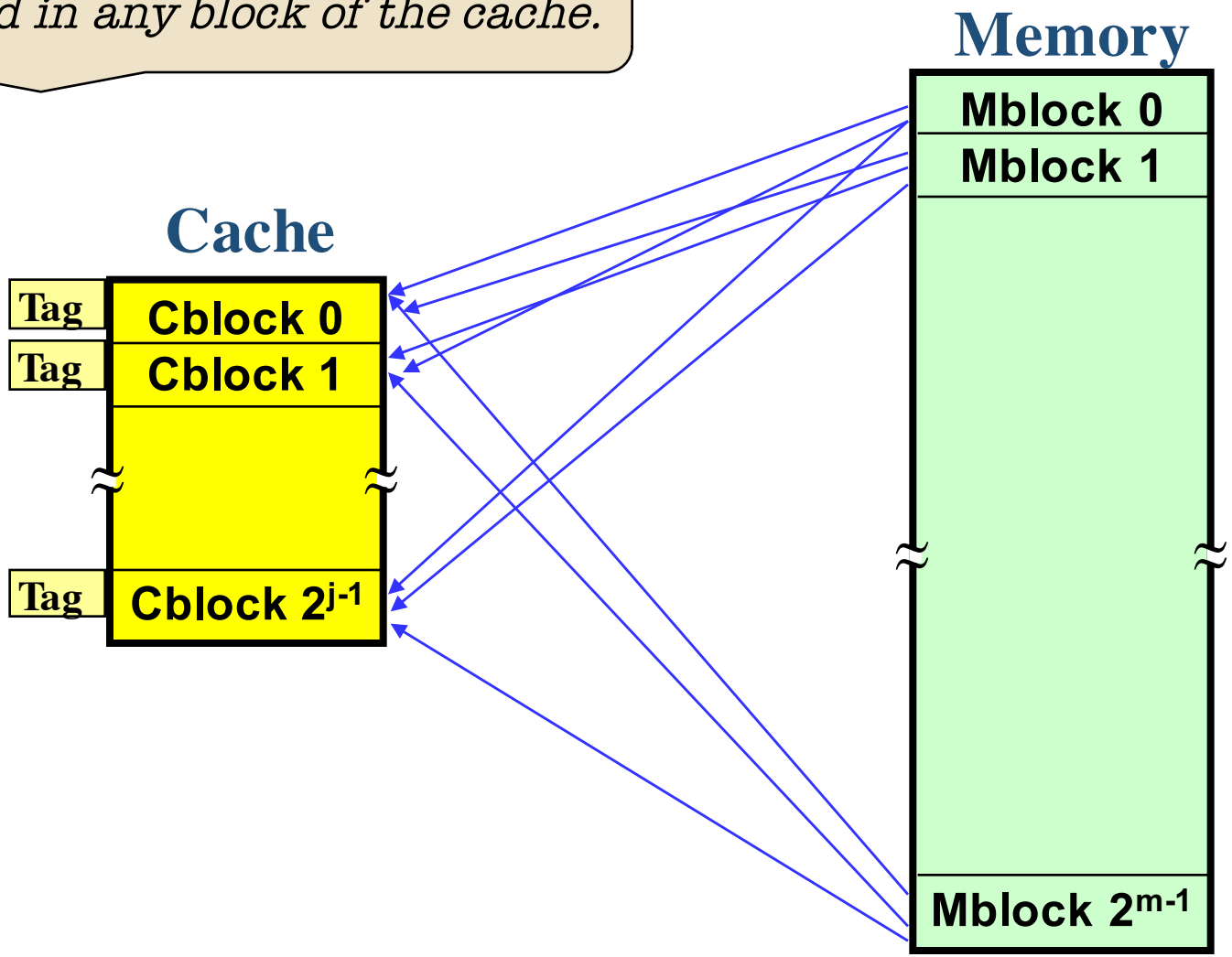
- *Architectural memory of 2^{16} bytes*
- *128-block cache of 16 bytes each*
- *CPU address 0011000000011111*

n = 16,
k = 4, j = 7, p = 5,
m = 12



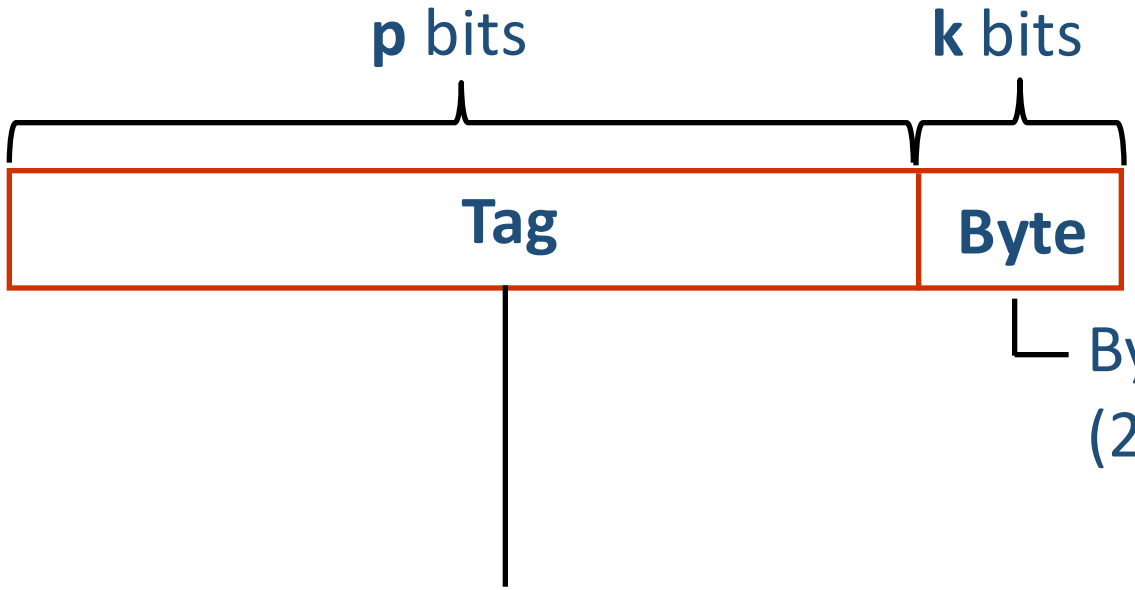
Fully Associative Cache

Any memory block can be allocated in any block of the cache.



Fully Associative Cache: CPU Address Mapping

n-bit Address & fully associative cache of 2^j blocks



Byte within cache block
(2^k bytes per block)

Identifier of memory block allocated on a cache block
(Any of 2^p memory blocks can be allocated on any cache block)

2^{n-k} blocks on memory

Determining Address Mapping Fields for Fully Associative Caches

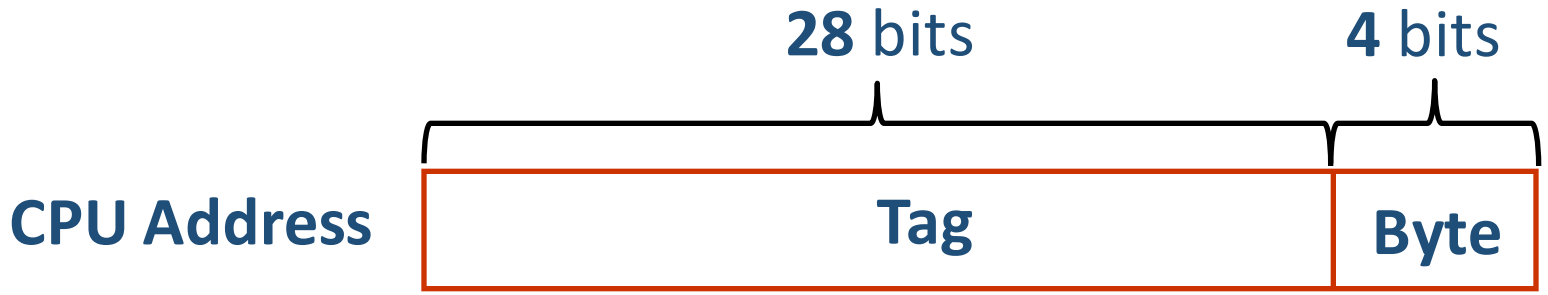
Memory of 2^{32} bytes, fully associative cache of 2048 blocks of 16 bytes each

Memory size = 2^{32}, then **n**=32

Number of cache blocks = 2048 = 2^{11}, then **j**=11

Block size = 16 = 2^4, then **k**=4

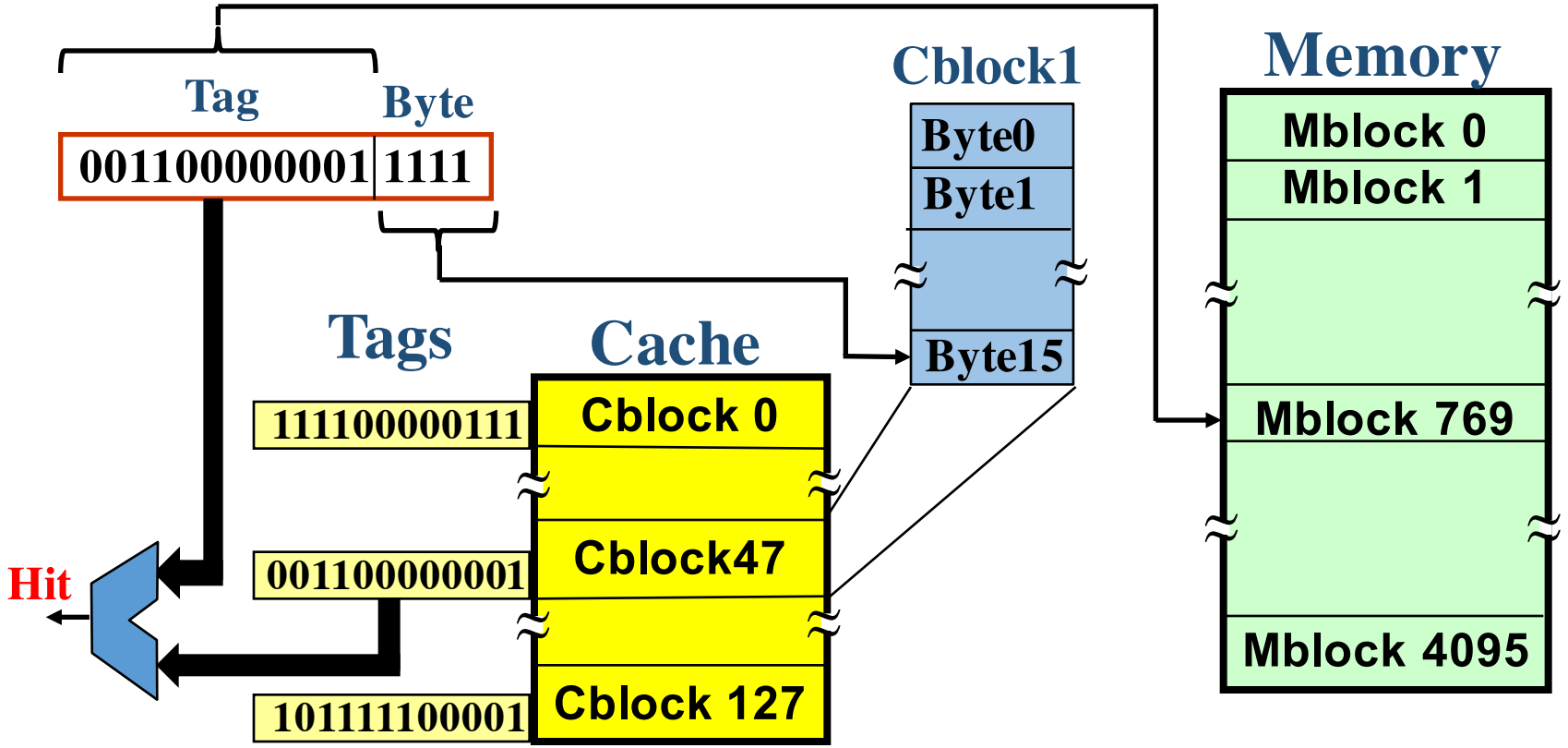
Then, **p** = **n** - **k** = 32 – 4 = 28



Fully Associative Cache Example

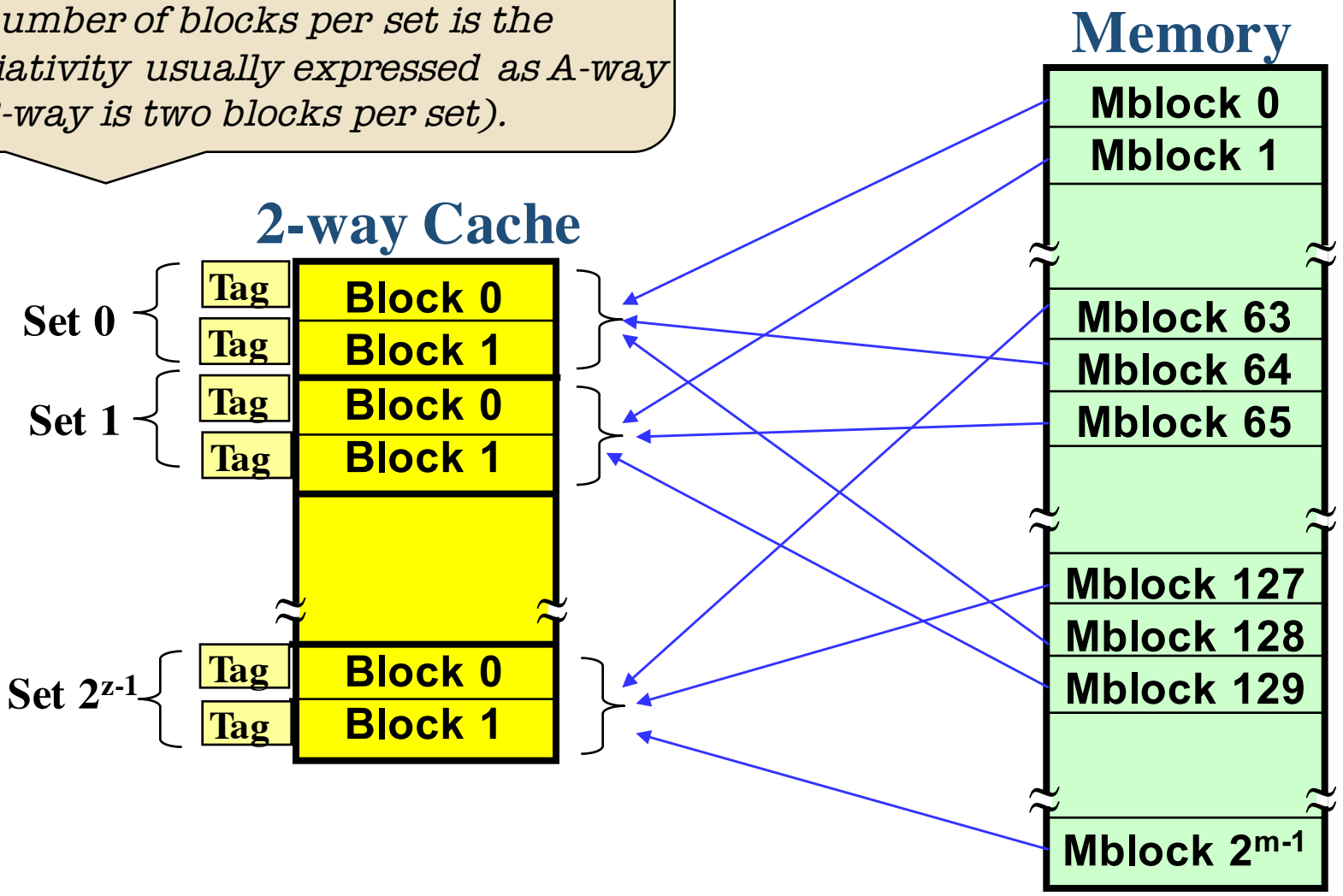
- *Architectural memory of 2^{16} bytes*
- *128-block cache of 16 bytes each*
- *CPU address 0011000000011111*

n = 16,
k = 4, j = 7, p = 12,
m = 12



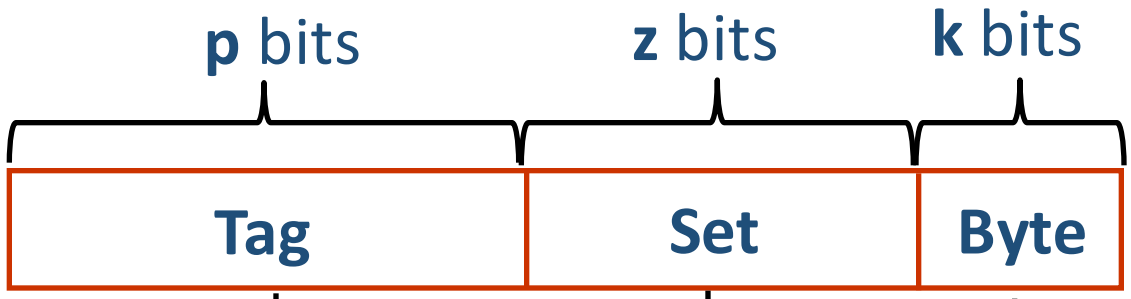
Block-Set Associative Cache

A memory block can be allocated in any block of a specific set in the cache. The number of blocks per set is the associativity usually expressed as A-way (i.e. 2-way is two blocks per set).



Block-Set Associative Cache: CPU Address Mapping

n-bit Address & **A**-way associative cache of 2^j blocks



Byte within cache block
(2^k bytes per block)

Set within cache
(2^z sets = cache size/associativity = 2^j/**A**)

Identifier of memory block allocated on the cache set
(one of 2^p blocks can be allocated on each cache set)

2^{n-k} blocks on memory

Determining Address Mapping Fields for Block-Set Associative Caches

Memory of 2^{32} bytes, 16-way cache of 4096 blocks of 64 bytes each

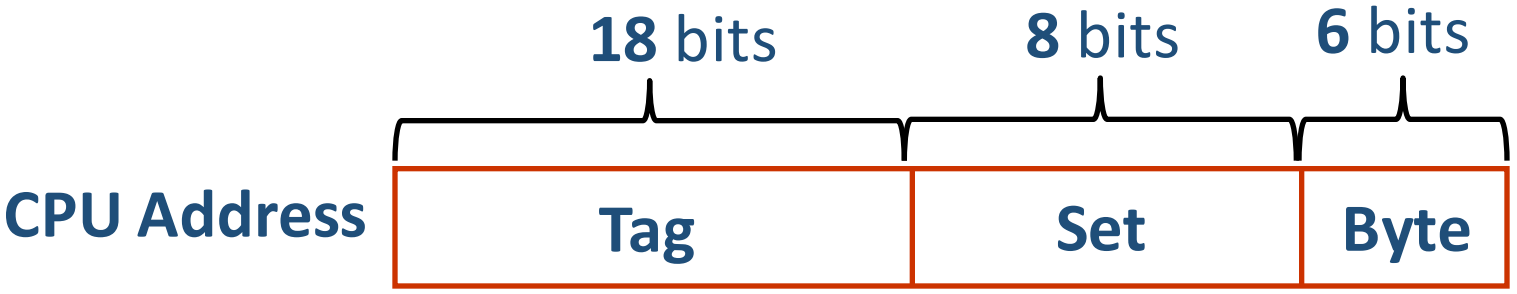
Memory size = 2^{32}, then **n**=32

Number of cache blocks = 4096 = 2^{12}, then **j**=12

Block size = 64 = 2^{6}, then **k**=6

Number of sets = $2^{j}/A = 2^{12}/16 = 2^{12}/2^{4} = 2^{8}$, then **z**=8

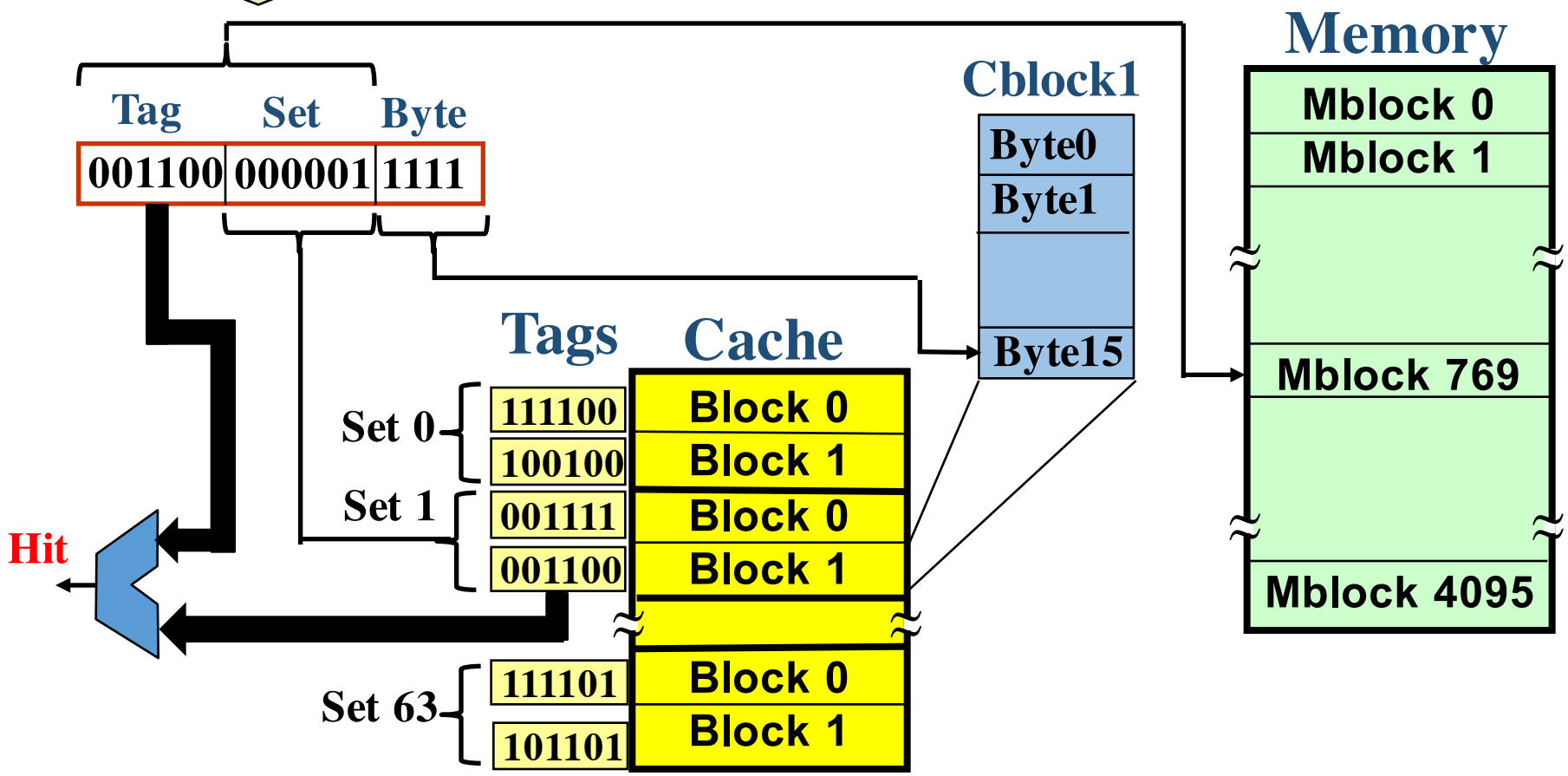
Then, **p** = **n** - (**z** + **k**) = 32 – (8 + 6) = 18



Block Set Associative Cache Example

- *Architectural memory of 2^{16} bytes*
- *128-block cache of 16 bytes each*
- *2-way associative (2 blocks per set)*
- *CPU address 0011000000011111*

n = 16,
k = 4, j = 7, p = 6,
z = 6, m = 12



Lesson Outcomes

- Understand why a cache is effective in reducing memory latency
- Know how caches are organized
- Understand the CPU memory address mapping for the three common cache configurations
- Understand how to determine if a CPU memory access results in a hit or a miss for any cache configuration