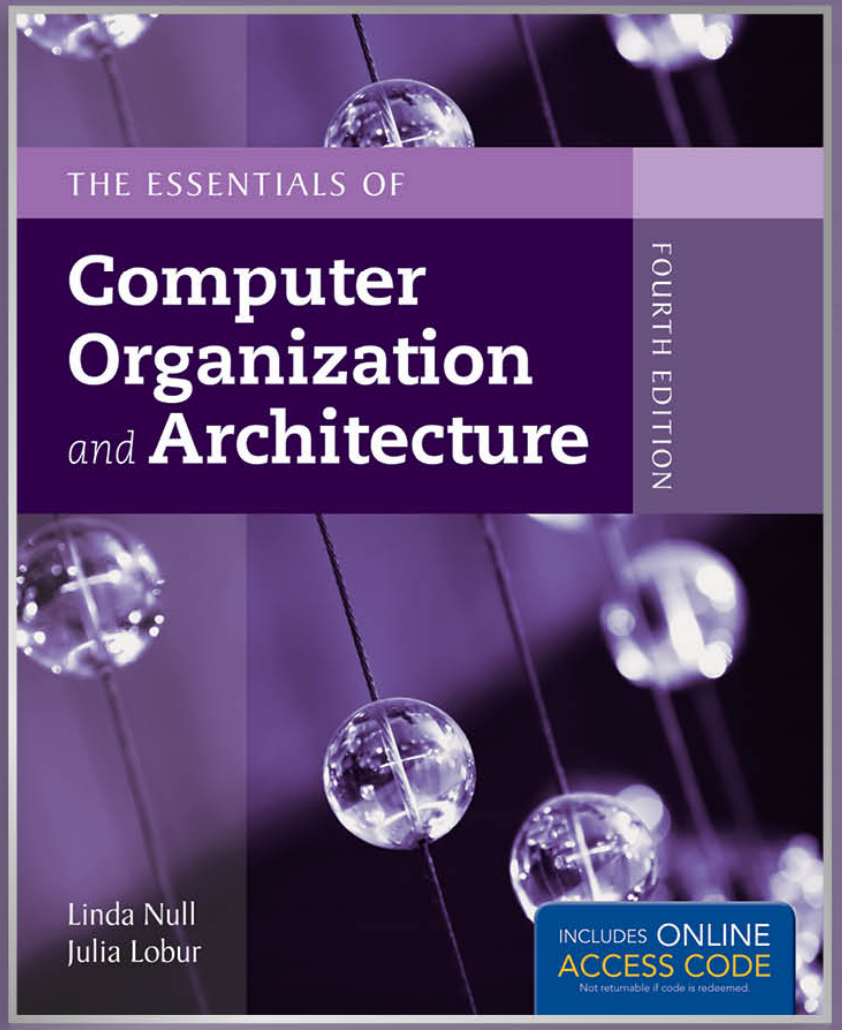
Chapter 6
Memory



Chapter 6 Objectives

- Master the concepts of **hierarchical memory organization**.
- Understand how each **level of memory** contributes to **system performance**.
- Master the concepts behind **cache memory.**

6.1 Introduction

- Memory lies at the heart of the stored-program computer.
- In previous chapters, we studied the **components from which memory is built** and the ways in which memory is accessed by various Instruction Set Architectures.
- In this chapter, we focus on **memory organization**. A clear understanding of these ideas is essential for the **analysis of system performance**.

6.2 Types of Memory

- There are two kinds of **main memory**: *random access memory, **RAM**, and read-only-memory, **ROM***.
- There are two types of RAM, **dynamic RAM** (DRAM) and **static RAM** (SRAM).
- **DRAM consists of capacitors** that slowly leak their charge over time. Thus, they **must be recharged** every few milliseconds to prevent data loss.
- DRAM is "**cheap**" memory owing to its simple design.

6.2 Types of Memory

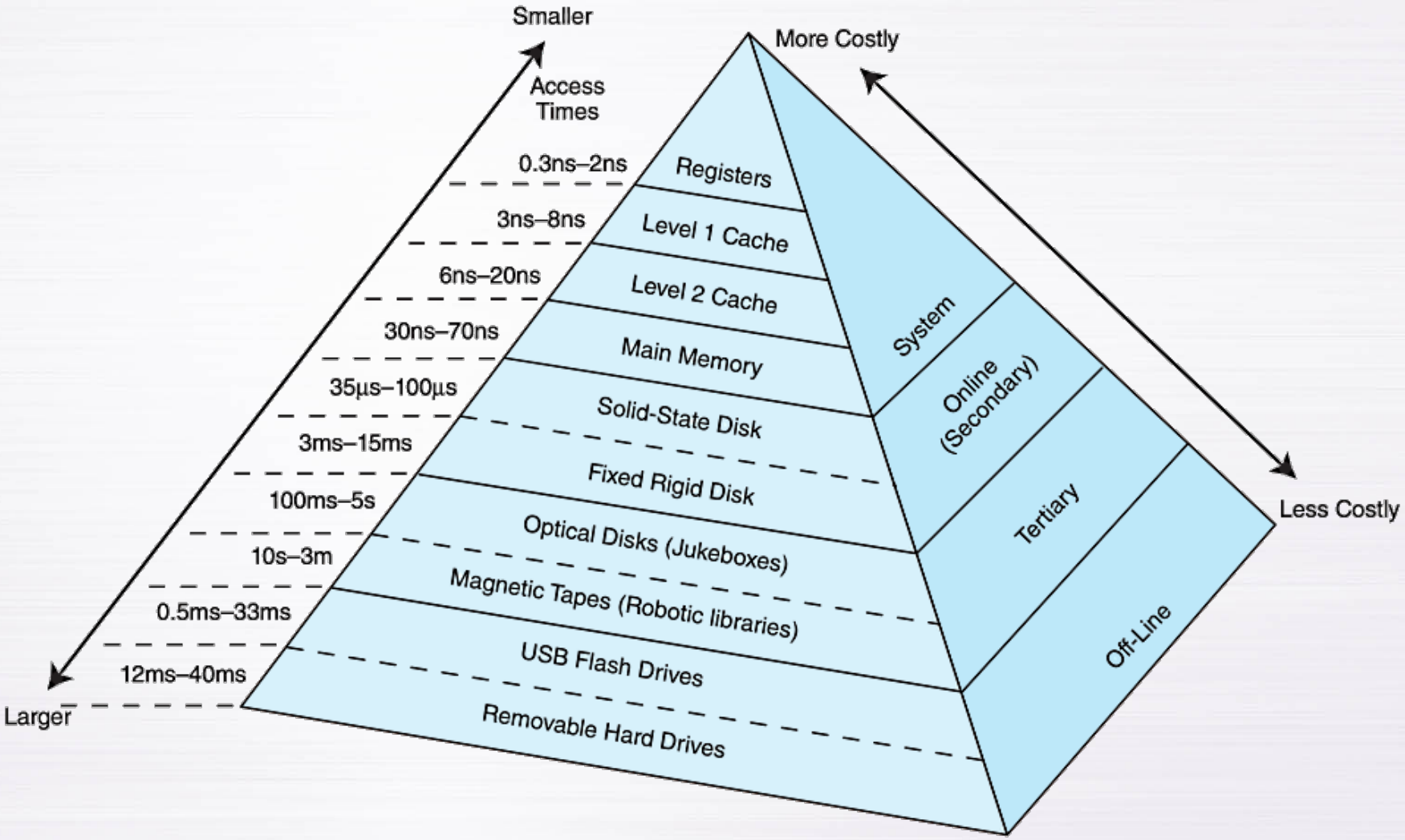
- **SRAM** consists of circuits **similar to the D flip-flop** that we studied in Chapter 3.
- SRAM is **very fast** memory and it doesn't need to be recharged like DRAM does. It is **used to build cache memory.**
- **ROM** is used to **store permanent data** that persists even while the system is turned off.

6.3 The Memory Hierarchy

- **Faster memory is more expensive than slower memory.**
- To provide the **best performance** at the **lowest cost**, memory is organized in a **hierarchical** fashion.
- **Small, fast storage** elements are **kept in the CPU**, **larger, slower main memory** is accessed through the **data bus**.
- **Larger, permanent storage** in the form of disk drives is **still further from the CPU**.

6.3 The Memory Hierarchy

- **Storage organization** can be thought of as a **pyramid**:

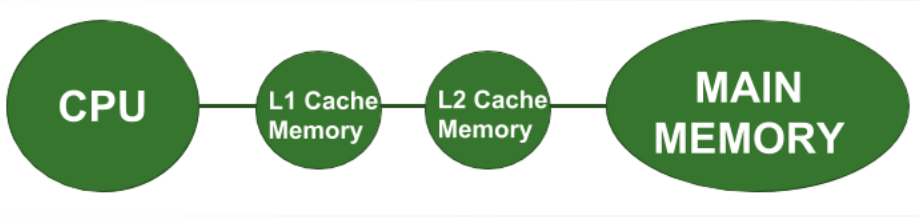


6.3 The Memory Hierarchy

- We are most interested in the memory hierarchy that involves **registers, cache, main memory**, and virtual memory.
- **Registers** are storage locations available **on the processor itself.**
- **Virtual memory** is typically implemented using a hard drive; it **extends the address space from main memory to the hard drive.**
- Virtual memory provides more space. **Cache memory provides speed.**

6.3 The Memory Hierarchy

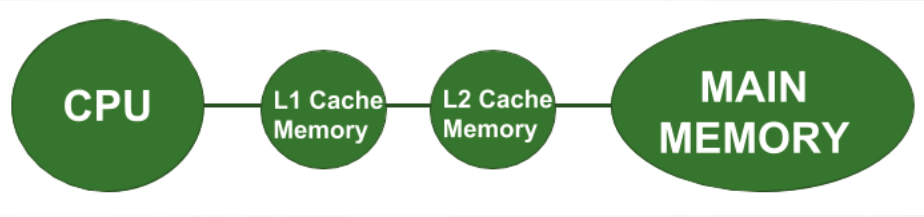
- To access a particular piece of data, the **CPU** first **sends a request** to its nearest memory, usually the **cache**.



- If the **data is not in cache, then main memory is queried.** If the data is not in main memory, then the request goes to disk.
- Once the **data is located**, then the data, and a number of its nearby **data elements are fetched into cache memory**.

6.3 The Memory Hierarchy

- This leads us to some definitions:



  - A ***hit*** is when **data is found** at a given memory level.
  - A ***miss*** is when it is **not found**.
  - The ***hit rate*** is the **percentage of time data is found** at a given memory level.
  - The ***miss rate*** is the percentage of time it is not (**1 - hit rate**).
  - The ***hit time*** is the **time required to access data at a given memory level**.
  - The ***miss penalty*** is the **time to process a miss**, including the time that it takes to replace a block of memory plus the time it takes to deliver the data to the processor.

6.3 The Memory Hierarchy

- An entire **block of data is copied after a hit** because the ***principle of locality*** tells us that **once a byte is accessed, it is likely that nearby data will be accessed soon**.
- There are three forms of locality:
  - ***Temporal locality*** – Recently accessed data elements tend to be accessed again.
  - ***Spatial locality*** - Accesses tend to cluster.
  - ***Sequential locality*** - Instructions tend to be accessed sequentially.

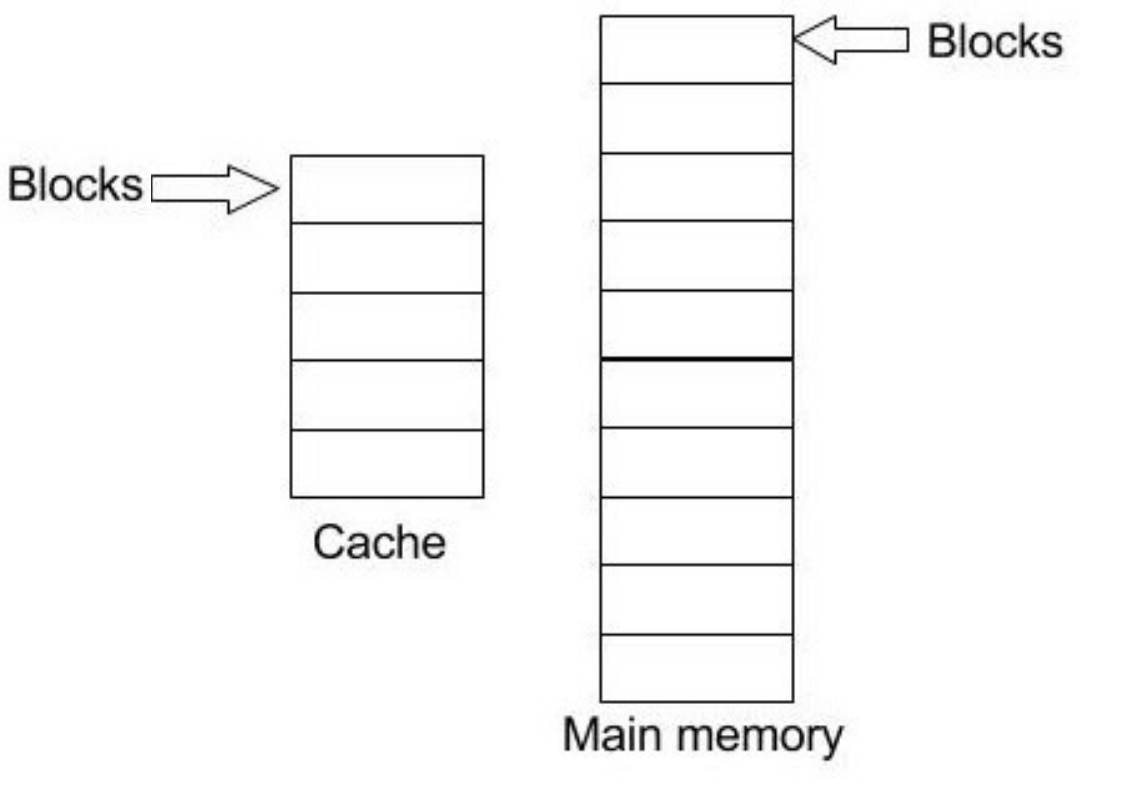
```
sum = 0;
for (i = 0; i < n; i++)
  sum += a[i];
return sum;
```

6.4 Cache Memory

- The purpose of **cache memory** is to **speed up accesses by storing recently used data closer to the CPU.**
- Although **cache is much smaller than main memory**, **its access time is a fraction** of that of main memory.
- Unlike **main memory**, which is **accessed by address**, **cache** is typically **accessed by content.**
- Because of this, a **single large cache** memory isn't always desirable-- it **takes longer to search**.

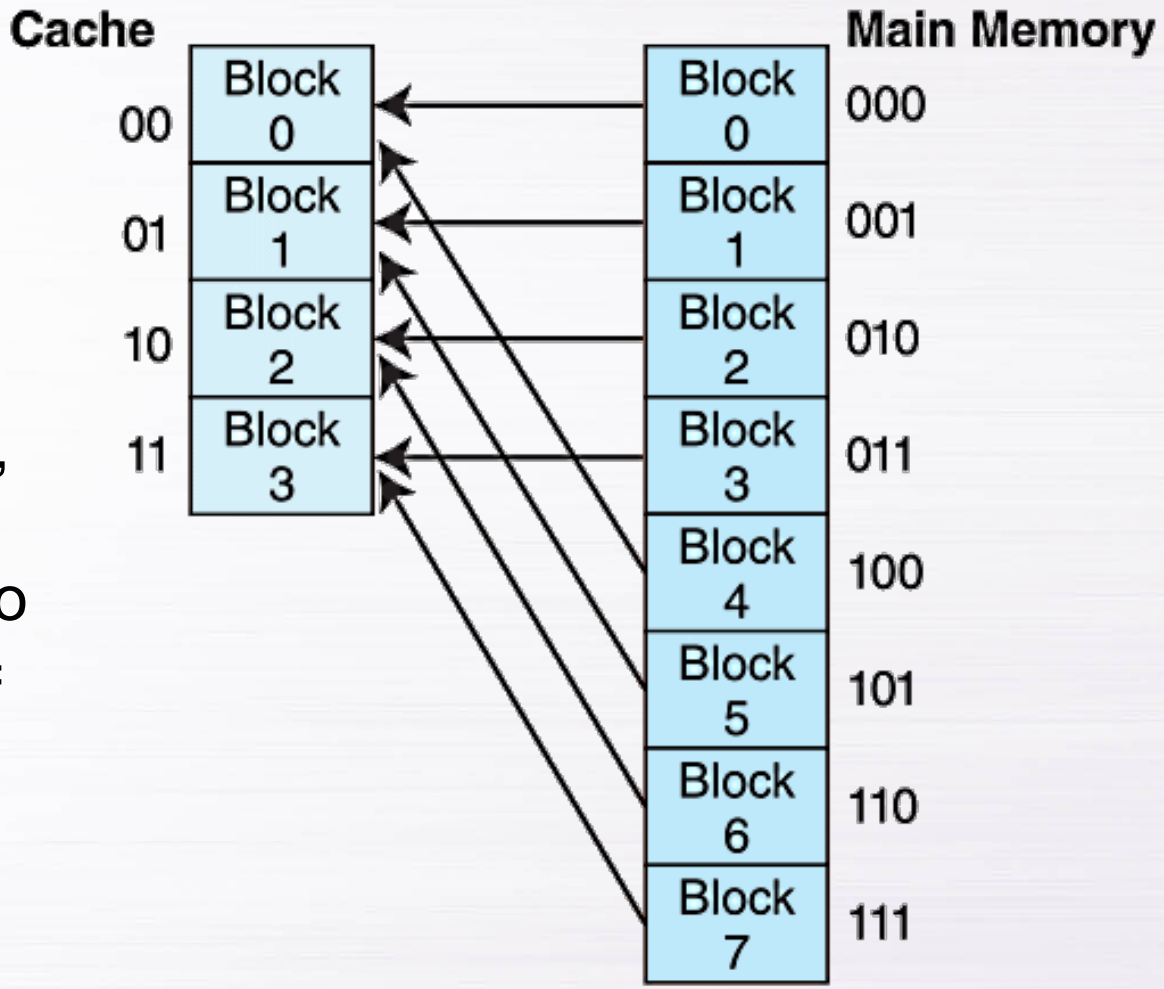
6.4 Cache Memory

- **Cache mapping** schemes are used to determine which main memory blocks are brought to which cache blocks



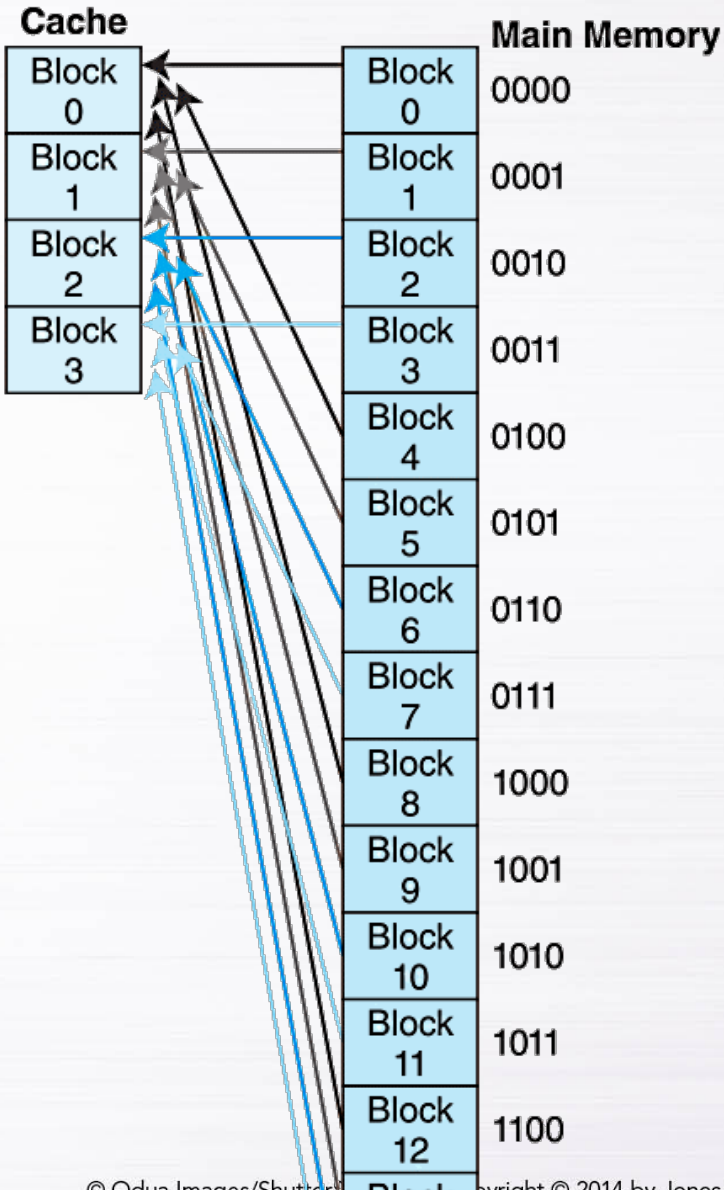
6.4 Cache Memory

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



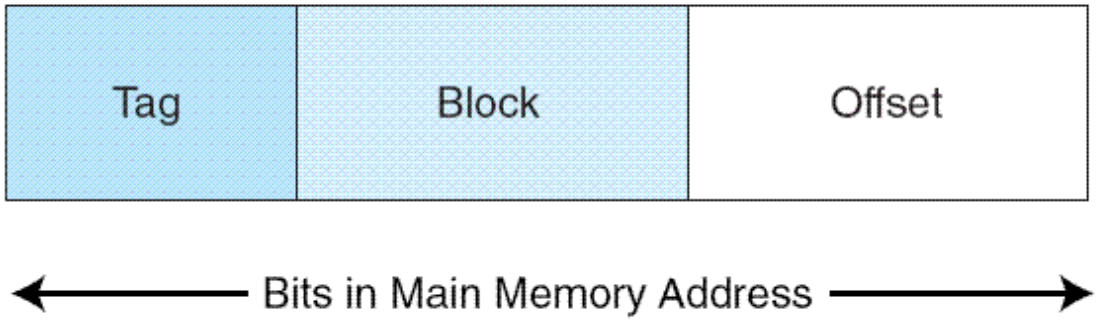
6.4 Cache Memory

- A larger example.



6.4 Cache Memory

- To perform direct mapping, the **binary main memory address is partitioned** into the *fields* shown below.
  - The **offset** field uniquely identifies an **address within a specific block**.
  - The **block** field selects a **unique block** of cache.
  - The **tag** field is whatever is **left over**.



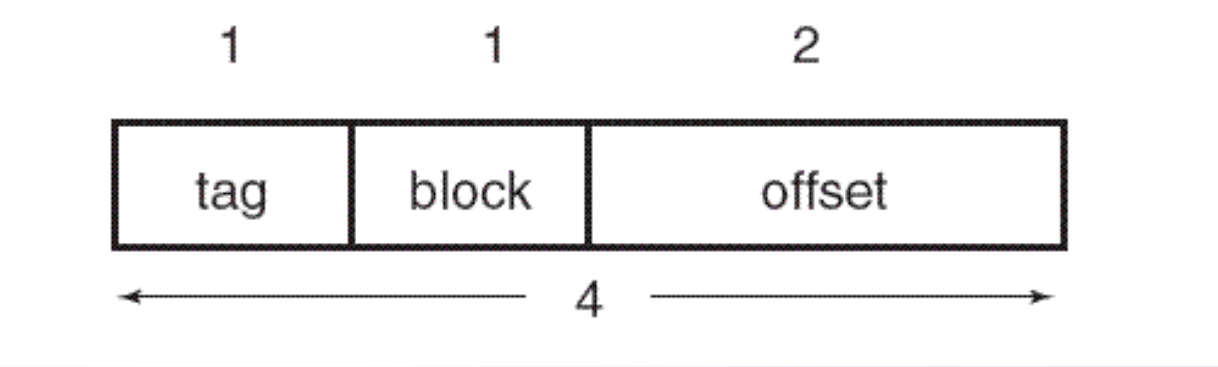
- The sizes of these fields are determined by characteristics of both memory and cache.

6.4 Cache Memory

- 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.

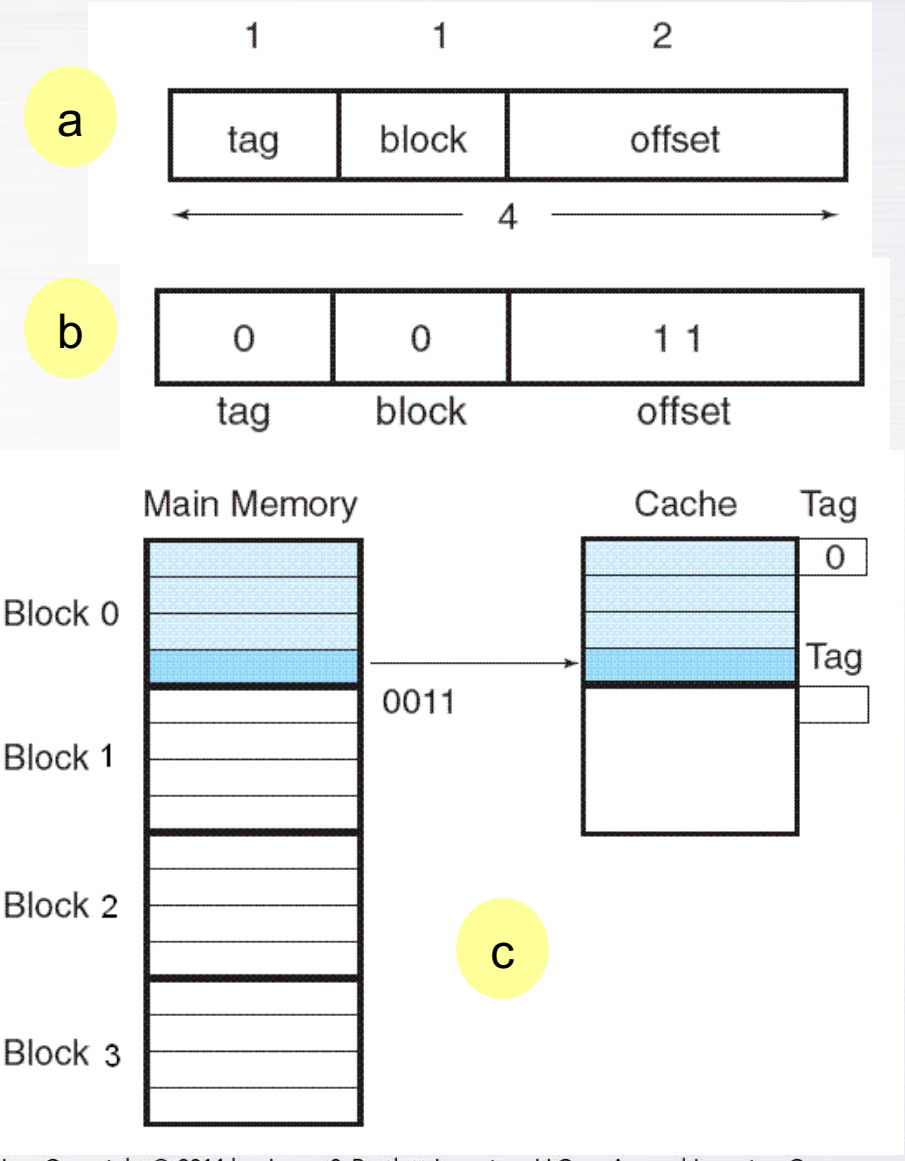
6.4 Cache Memory

- 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**.
  - Each **block** is **4 = 2^2 bytes**, so the **offset** field must contain **2 bits**;
  - there are **2 = 2^1 blocks in cache**, so the **block field** must contain **1 bit**;
  - this **leaves 1 bit for the tag** (as a **main memory address has 4 bits** because there are a total of **4x4 = 16 = 2^4 bytes**).



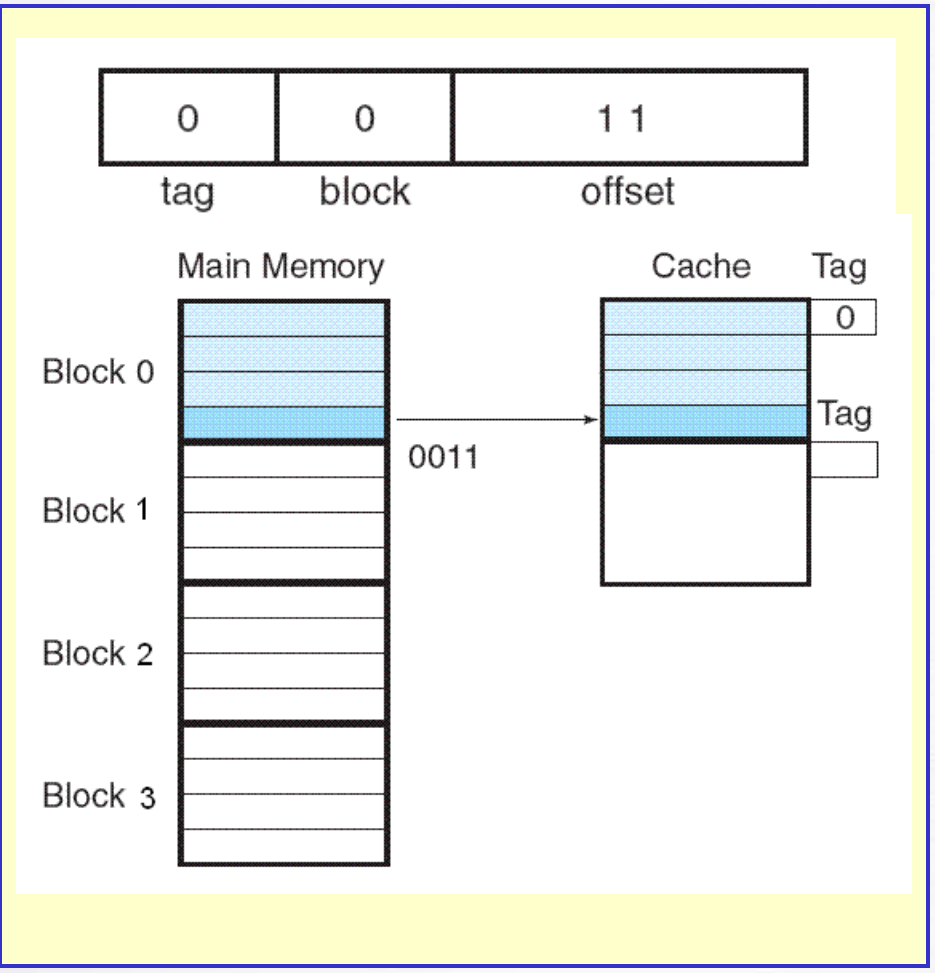
6.4 Cache Memory

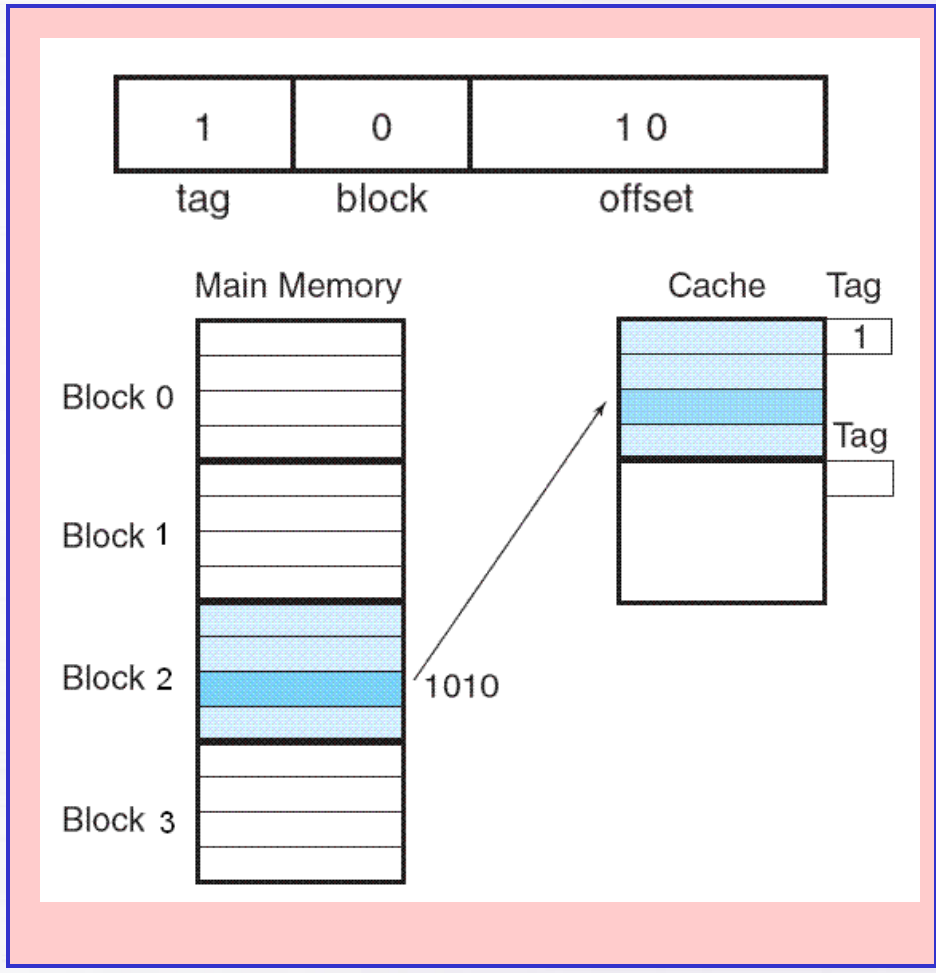
- EXAMPLE 6.1 Cont'd
  - Suppose we need to **access main memory address 3_{16}** (0x0011 in binary). If we partition **0x0011** using the address format from Figure a, we get Figure b.
  - Thus, the main memory address 0x0011 **maps to cache block 0**.
  - Figure c shows this mapping, along with the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



The next slide illustrates another mapping.

6.4 Cache Memory

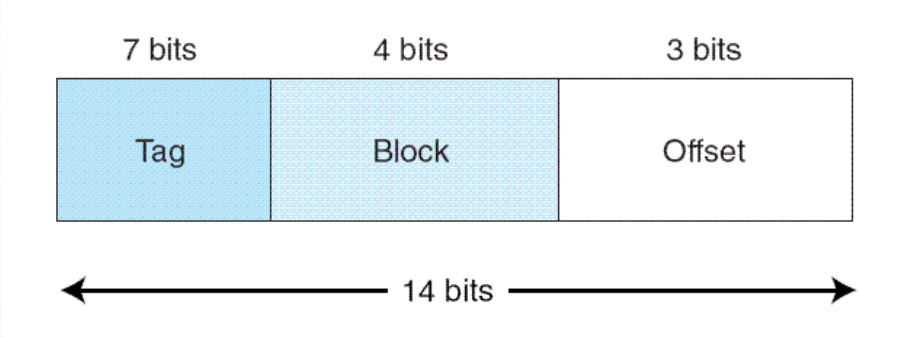




6.4 Cache Memory

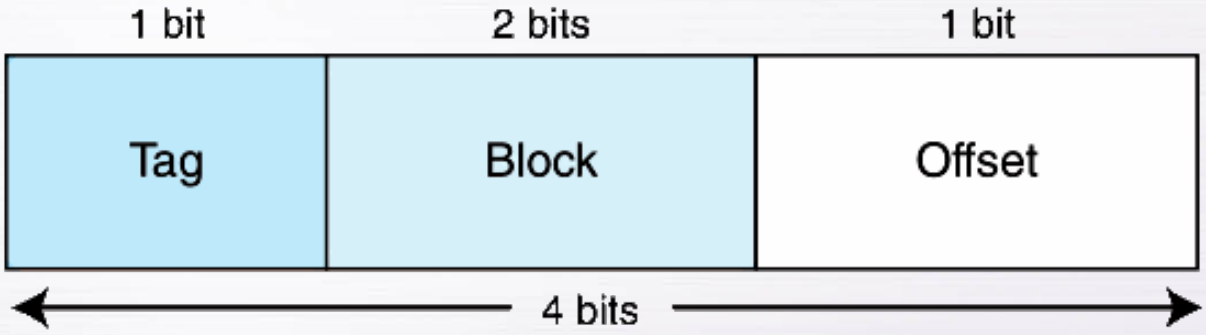
- EXAMPLE 6.2 Assume a byte-addressable **memory** consists of **2^{14} bytes**, **cache has 16 = 2^4 blocks**, and each **block has 8 = 2^3 bytes**.

  - Each **main memory address** requires **14 bits**. Of this 14-bit address field, the rightmost **3 bits** reflect the **offset** field
  - We need **4 bits** to select a specific **block** in cache, so the block field consists of the middle 4 bits.
  - The **remaining 7 bits** make up the **tag** field.



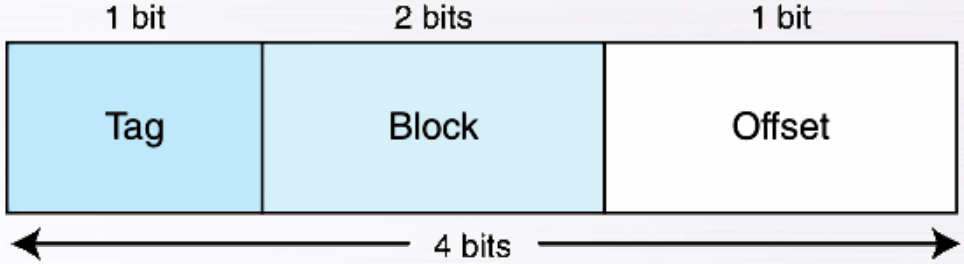
6.4 Cache Memory

- 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**.
  - 16 bytes divided into 8 blocks results a block size of 2, i.e. an offset of 1
  - The 4-bit memory address is **divided into:**



6.4 Cache Memory

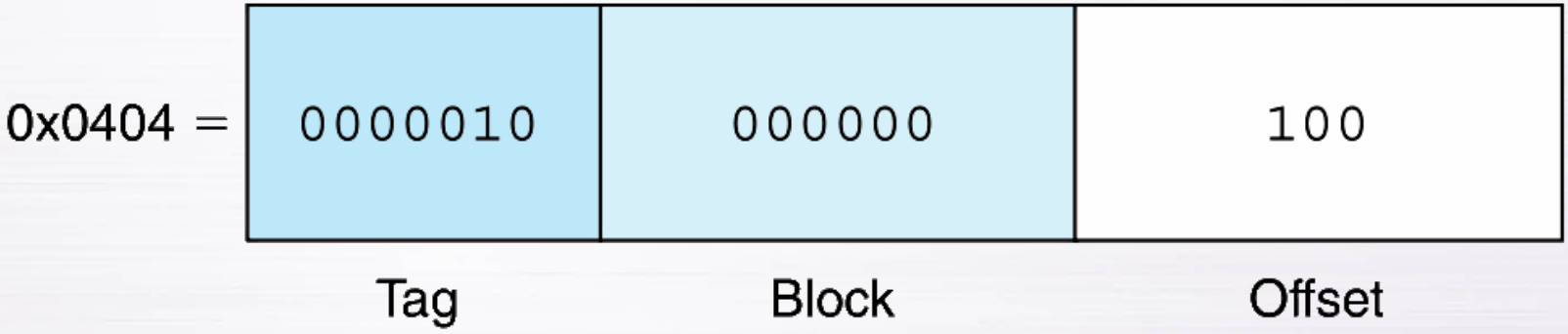
- EXAMPLE 6.3 Cont'd The mapping for memory references is shown below:



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

- 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

- 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**
- The number of **cache blocks**
- Number of **addresses are in a block**

6.4 Class Exercise

Suppose a computer using direct mapped cache has **2^{20} bytes** of byte-addressable **main memory** and a **cache of 32 blocks**, where **each cache block contains 16 bytes**.

- How **many blocks of main memory** are there?
- What is the format of a memory address as seen by the cache (**tag, block, offset**)?
- To which cache block will the memory address **0000 1101 1011 0110 0011** map?