# Lesson 11

**Memory Hierarchies**

# Introduction

- From the earliest days of computing, programmers have wanted unlimited amounts of fast memory
- We will see an illusion for creating such memories.
- The principle of locality states that programs access a relatively small portion of their address space at any instant of time.
- two different types of locality:
  - Temporal locality (locality in time)
  - Spatial locality (locality in space)

# Two Types Of Locality

- Temporal locality (locality in time):
  - If an item is referenced, it will tend to be referenced again soon. If you recently brought a book to your desk to look at, you will probably need to look at it again soon.
  - Temporal locality states that if a data location is referenced then it will tend to be referenced again soon.
- Spatial locality (locality in space):
  - If an item is referenced, items whose addresses are close by will tend to be referenced soon.
  - Spatial locality states that if a data location is referenced, data locations with nearby addresses will tend to be referenced soon.

# Example:

- Bob is building a fence behind his house. He uses a hammer to attach a board to the rail. Bob then measures and cuts the next board.
  - The likelihood that Bob will need the hammer again is an example of temporal  locality.
- We take advantage of the principle of locality by implementing the memory of a computer as a *memory hierarchy*

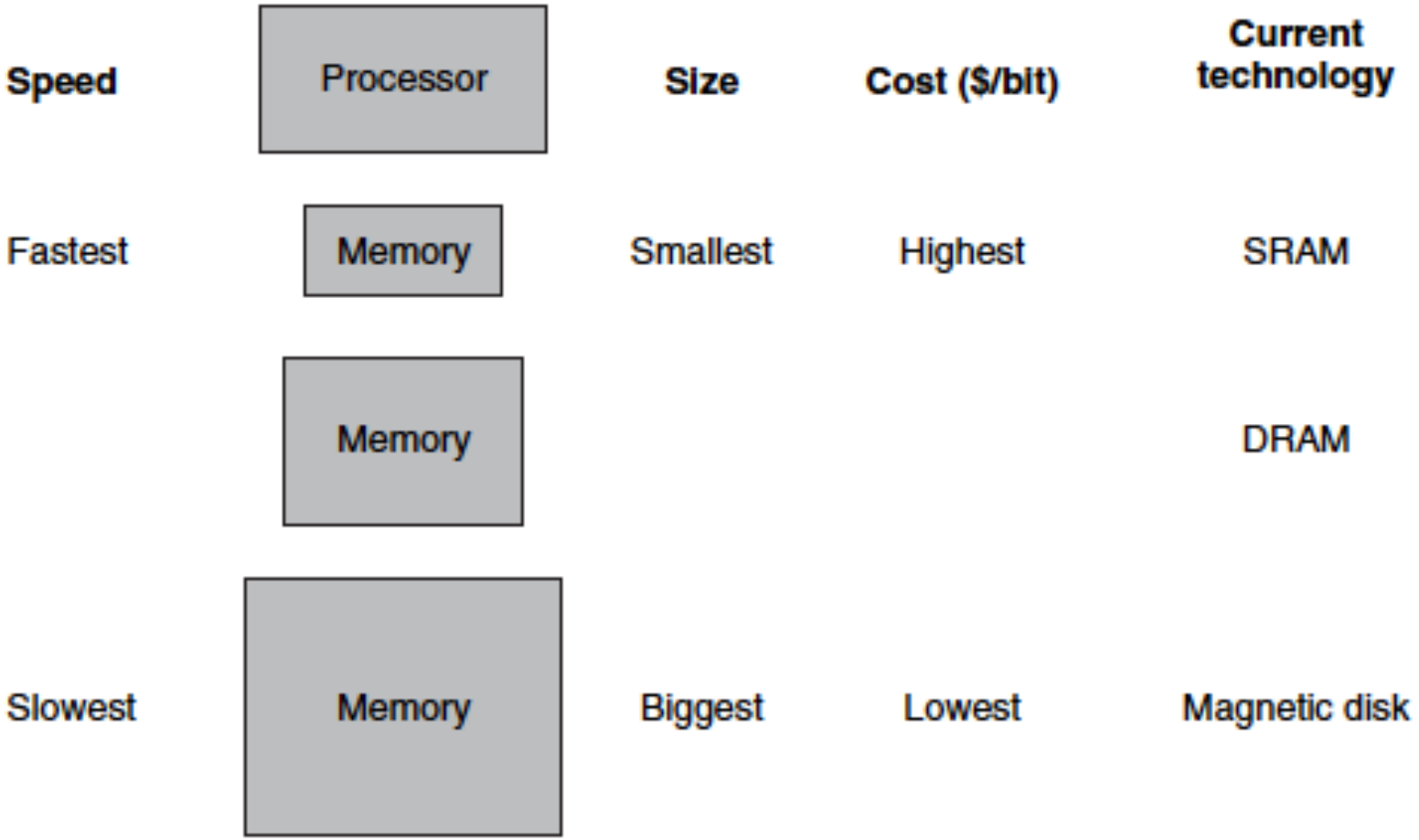
# Memory hierarchy:

- We take advantage of the principle of locality by implementing the memory of a computer as a *memory hierarchy*
- Memory hierarchy is a structure that uses multiple levels of memories; as the distance from the processor increases, the size of the memories and the access time both increase.
- By implementing the memory system as a hierarchy, the user has the illusion of a memory that is as large as the largest level of the hierarchy, but can be accessed as if it were all built from the fastest memory
- It is important to know that registers are in the processor

# Memory hierarchy:

- A memory hierarchy can consist of multiple levels, but data is copied between only two adjacent levels at a time
- **Block (or line)**: The minimum unit of information that can be either present or not present in a cache.

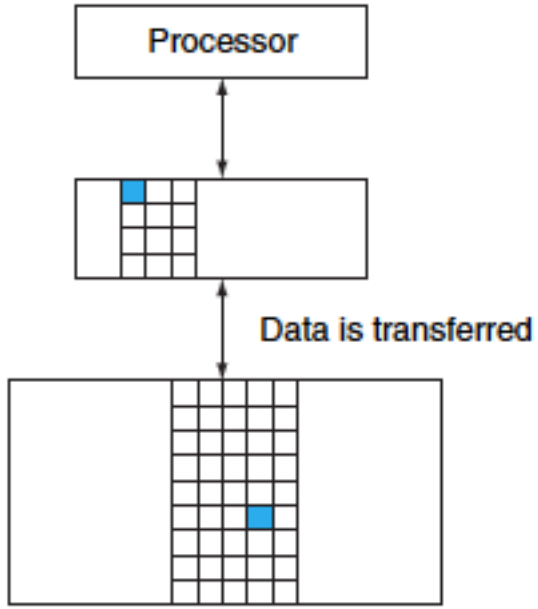
# The Basic Memory Structure



# Memory Structure

- Flash memory has replaced disks in many personal mobile devices, and may lead to a new level in the storage hierarchy for desktop and server computers

# Block transfer




- A memory hierarchy can consist of multiple levels.
- Data is copied between only two adjacent levels at a time
- The upper level—the one closer to the processor—is smaller and faster than the lower level, since the upper level uses technology that is more expensive
- ***Block (or line)***: The minimum unit of information that can be either present or not present in a cache.

# Hit Rate and Miss Rate

- If the data requested by the processor appears in some block in the upper level, this is called a *hit*
- The *hit rate*, or *hit ratio*, is the fraction of memory accesses found in the upper level; it is often used as a measure of the performance of the memory hierarchy.
- If the data is not found in the upper level, the request is called a *miss*. The lower level in the hierarchy is then accessed to retrieve the block containing the requested data.
- The *miss rate* (1 - hit rate) is the fraction of memory accesses not found in the upper level

# Hit time and Miss penalty definitions

- ***Hit time***: The time required to access a level of the memory hierarchy, including the time needed to determine whether the access is a hit or a miss.
- ***Miss penalty***: The time required to fetch a block into a level of the memory hierarchy from the lower level, including the time to access the block, transmit it from one level to the other, insert it in the level that experienced the miss, and then pass the block to the requestor.

# Cache Terminology

- A memory hierarchy is composed of an upper level and a lower level. Data is requested by the processor. 9 out of 10 requests find the data in the upper level and returns the data in 0.4 ns. The remaining requests require 0.7 ns to return the data.
- Determine the corresponding values for the upper level memory.
- Hit rate = 9 / 10 = 0.9
- Miss rate = (1 – hit rate) = (1 – 0.9) = 0.1
- Hit time: Data in the upper level requires 0.4 ns to retrieve. = 0.4
- Miss penalty: 0.7 ns is required to replace a block in the upper level with the corresponding block from the lower level, and then deliver the block to the processor.

# Memories technologies

- Four primary technologies used in memory hierarchies.
  - Main memory is implemented from DRAM (*dynamic random access memory*)
  - Caches use SRAM (*static random access memory*)
  - Flash memory
  - Magnetic disk

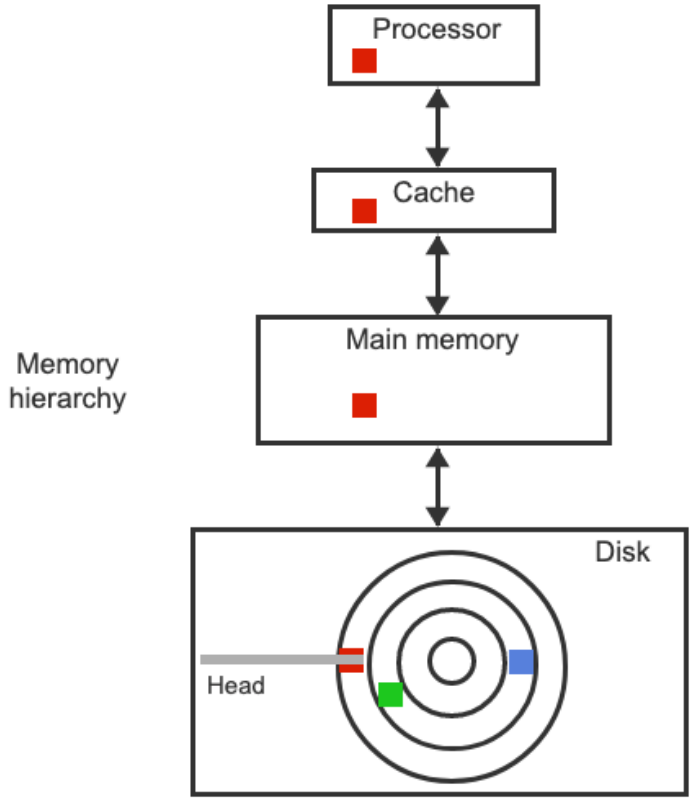
# Memory technologies

- Main memory is implemented from DRAM (*dynamic random access memory*),
- Caches use SRAM (*static random access memory*)
- Flash memory. This nonvolatile memory is the secondary memory in Personal Mobile Devices.
- magnetic disk - largest and slowest level in the hierarchy in servers

# Memory structures

| Memory technology | Typical access time | $ per GiB in 2012 |
|---|---|---|
| SRAM semiconductor memory | 0.5–2.5 ns | $500–$1000 |
| DRAM semiconductor memory | 50–70 ns | $10–$20 |
| Flash semiconductor memory | 5,000–50,000 ns | $0.75–$1.00 |
| Magnetic disk | 5,000,000–20,000,000 ns | $0.05–$0.10 |

# Primary technologies used in memories




Memory hierarchy

Current technology

SRAM

DRAM

Magnetic disk

** Flash memory used in personal mobile devices*

# SRAM Technology

- SRAMs are simply integrated circuits that are memory arrays with (usually) a single access port that can provide either a read or a write.
- SRAMs have a fixed access time to any datum, though the read and write access times may differ.
- SRAMs don't need to refresh and so the access time is very close to the cycle time.
- SRAMs typically use six to eight transistors per bit to prevent the information from being disturbed when read.
- SRAM needs only minimal power to retain the charge in standby mode.

# DRAM Technology

- In a dynamic RAM (DRAM), the value kept in a cell is stored as a charge in a capacitor.
- A single transistor is then used to access this stored charge, either to read the value or to overwrite the charge stored there
- DRAMs use only a single transistor per bit of storage, hence they are much denser and cheaper per bit than SRAM.
- As DRAMs store the charge on a capacitor, it cannot be kept indefinitely and must periodically be refreshed. To refresh the cell, we merely read its contents and write it back.

# DRAM size increased by multiples of four approximately once every three years until 1996, and thereafter considerably slower

| Year introduced | Chip size | $ per GiB | Total access time to a new row/column | Average column access time to existing row |
|---|---|---|---|---|
| 1980 | 64 Kibibit | $1,500,000 | 250 ns | 150 ns |
| 1983 | 256 Kibibit | $500,000 | 185 ns | 100 ns |
| 1985 | 1 Mebibit | $200,000 | 135 ns | 40 ns |
| 1989 | 4 Mebibit | $50,000 | 110 ns | 40 ns |
| 1992 | 16 Mebibit | $15,000 | 90 ns | 30 ns |
| 1996 | 64 Mebibit | $10,000 | 60 ns | 12 ns |
| 1998 | 128 Mebibit | $4,000 | 60 ns | 10 ns |
| 2000 | 256 Mebibit | $1,000 | 55 ns | 7 ns |
| 2004 | 512 Mebibit | $250 | 50 ns | 5 ns |
| 2007 | 1 Gibibit | $50 | 45 ns | 1.25 ns |
| 2010 | 2 Gibibit | $30 | 40 ns | 1 ns |
| 2012 | 4 Gibibit | $1 | 35 ns | 0.8 ns |

# Flash memory

- **Flash memory** is a type of electrically erasable programmable read-only memory (EEPROM)
- Like other EEPROM technologies, writes can wear out flash memory bits.
- To cope with such limits, most flash products include a controller to spread the writes by remapping blocks that have been written many times to less trodden blocks. This technique is called *wear leveling*. .

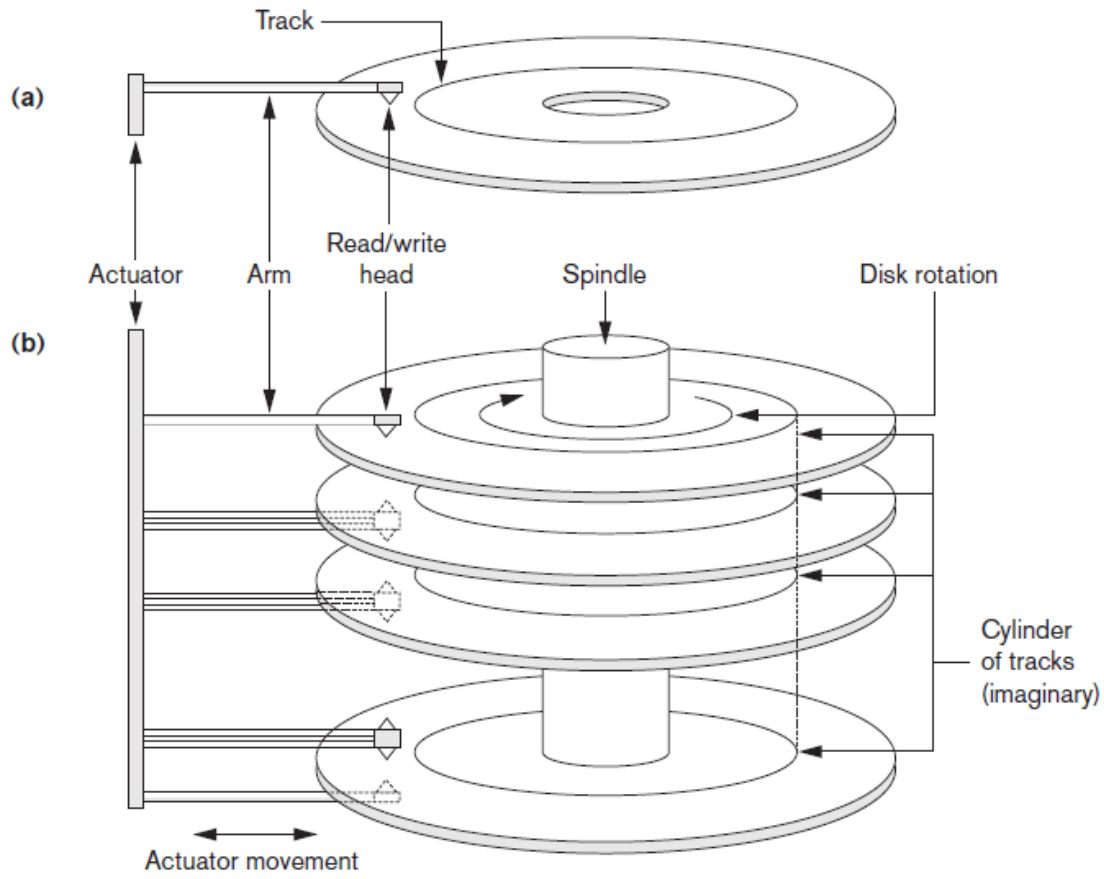
# Disk memory

- A magnetic hard disk consists of a collection of platters, which rotate on a spindle at 5400 to 15,000 revolutions per minute.
- The metal platters are covered with magnetic recording material on both sides, similar to the material found on a cassette or videotape.
- To read and write information on a hard disk, a movable *arm* containing a small electromagnetic coil called a *read-write head* is located just above each surface. The entire drive is permanently sealed to control the environment inside the drive, which, in turn, allows the disk heads to be much closer to the drive surface.

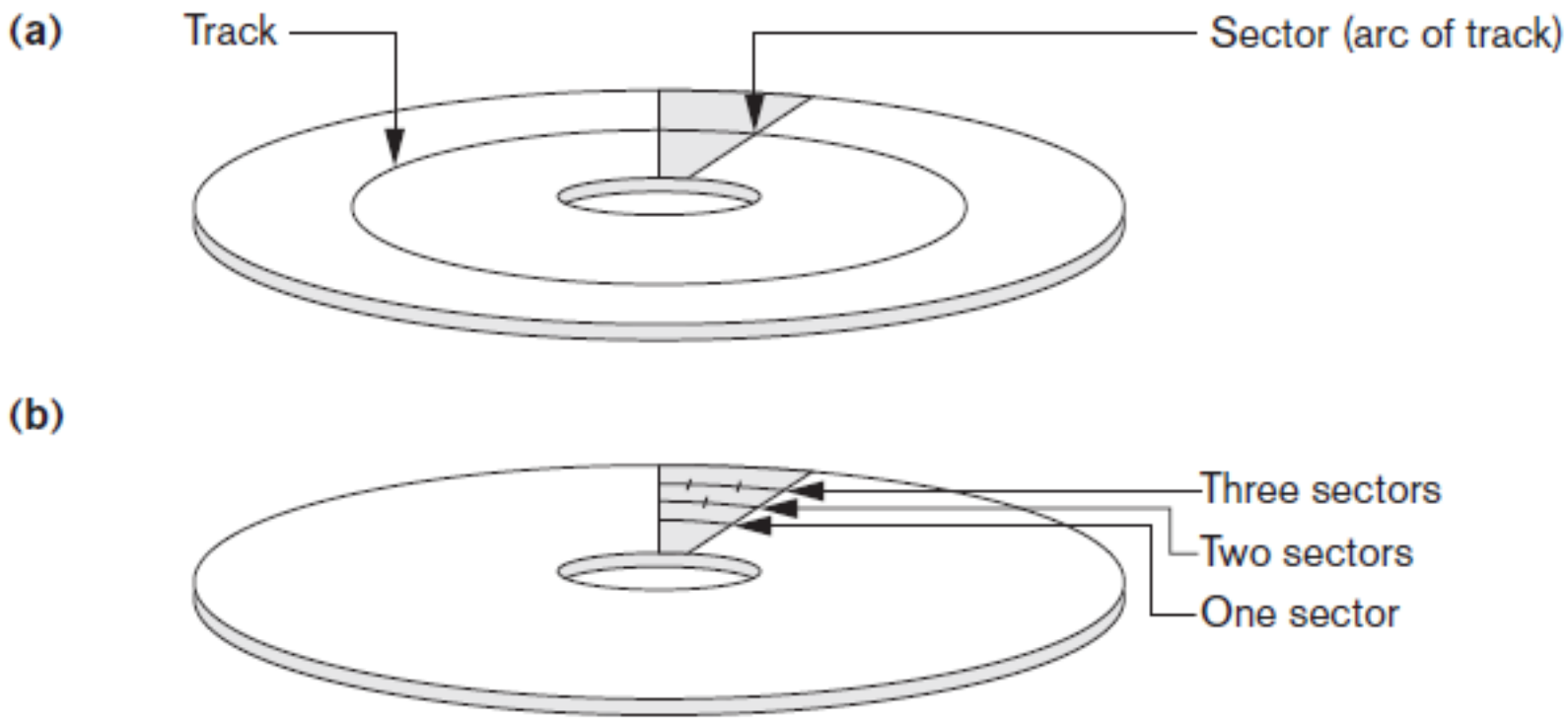
# Disk

- ***Track***: One of thousands of concentric circles that make up the surface of a magnetic disk.
- Tracks divided into blocks or sectors
- ***Sector***: One of the segments that make up a track on a magnetic disk; a sector is the smallest amount of information that is read or written on a disk.
- ***Seek***: The process of positioning a read/write head over the proper track on a disk

# Single-Sided Disk and Disk Pack



# Sectors on a Disk



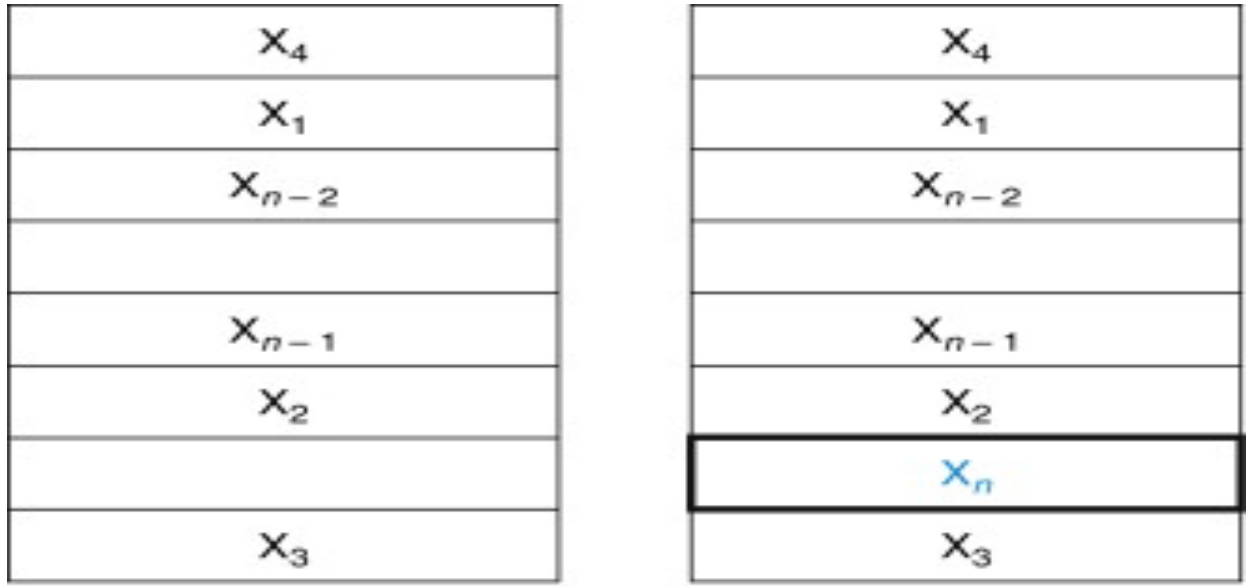
# Disk

- **Rotational latency**: Also called **rotational delay**. The time required for the desired sector of a disk to rotate under the read/write head; usually assumed to be half the rotation time.

# The Basics Of Caches

- *Cache* was the name chosen to represent the level of the memory hierarchy between the processor and main memory in the first commercial computer to have this extra level.
- Today, the term is also used to refer to any storage managed to take advantage of locality of access.

# The cache just before and just after a reference to a word Xn that is not initially in the cache

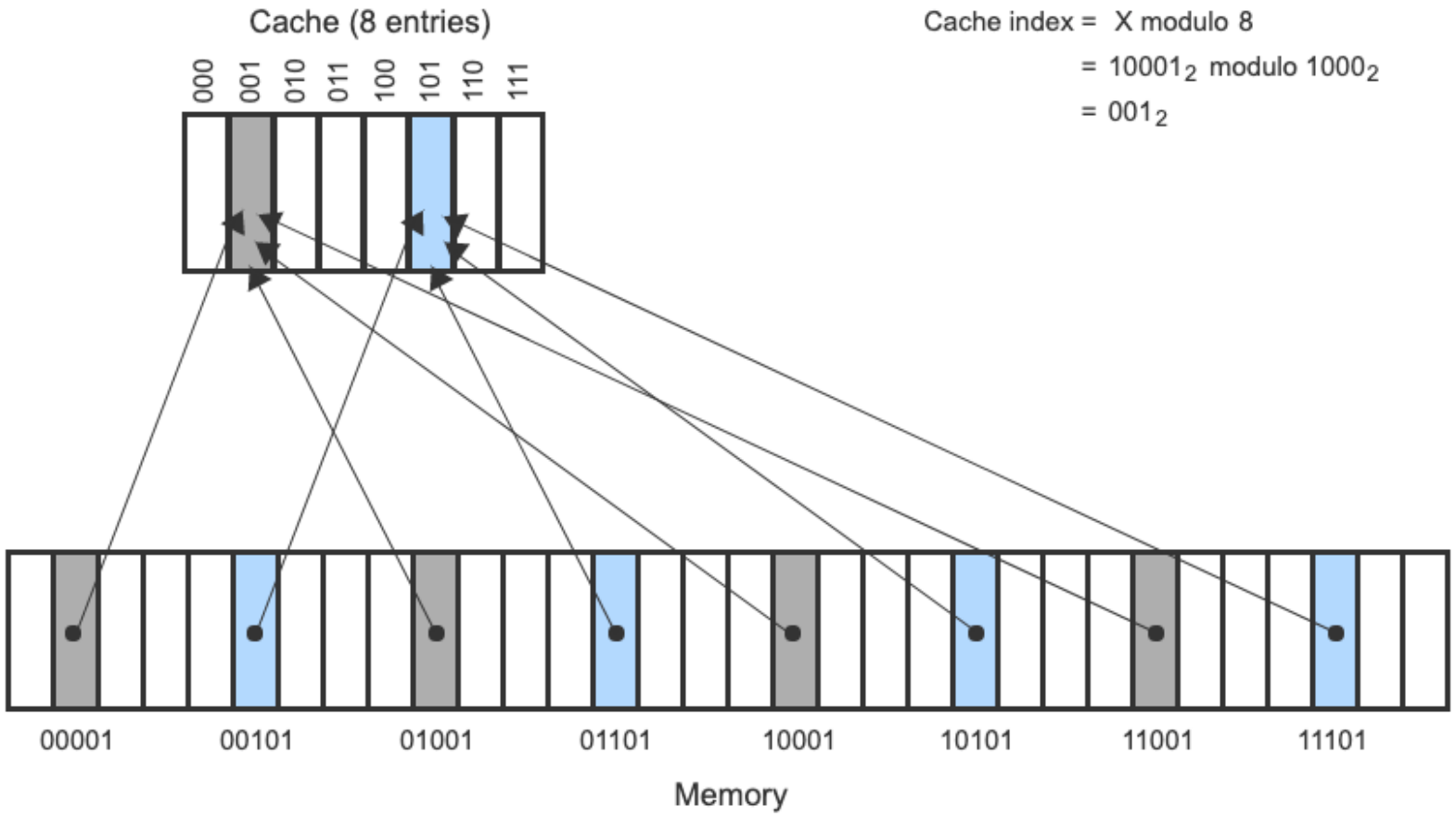



a. Before the reference to $X_n$ b. After the reference to $X_n$

# Cache

- How do we know if a data item is in the cache?
  - Direct mapped cache - assign a location in the cache for each word in memory is to assign the cache location based on the address of the word in memory. ***Direct-mapped cache***: A cache structure in which each memory location is mapped to exactly one location in the cache. In a directly mapped cache we can use the mapping (block address) modulo (number of blocks in the cache)

# A direct-mapped cache with eight entries showing the addresses of memory words between 0 and 31 that map to the same cache locations



# Cache

- ***Tag***: A field in a table used for a memory hierarchy that contains the address information required to identify whether the associated block in the hierarchy corresponds to a requested word.
- ***Valid bit***: A field in the tables of a memory hierarchy that indicates that the associated block in the hierarchy contains valid data.

# Accessing a cache using the tag and valid bits.

Memory reference:

$10111_{two}$ hit

$10000_{two}$ miss (fetched from memory)

$11101_{two}$ miss

| Index | V | Tag | Data |
|---|---|---|---|
| 000 | Y | $10_{two}$ | Memory ($10000_{two}$) |
| 001 | Y | $11_{two}$ | Memory ($11001_{two}$) |
| 010 | N | | |
| 011 | N | | |
| 100 | Y | $00_{two}$ | Memory ($00100_{two}$) |
| 101 | N | | |
| 110 | N | | |
| 111 | Y | $10_{two}$ | Memory($10111_{two}$) |

Invalid data

# Handling cache misses

- ***Cache miss***: A request for data from the cache that cannot be filled because the data is not present in the cache.

# The steps to be taken on an instruction cache miss

1. Send the original PC value (current PC - 4) to the memory.
2. Instruct main memory to perform a read and wait for the memory to complete its access.
3. Write the cache entry, putting the data from memory in the data portion of the entry, writing the upper bits of the address (from the ALU) into the tag field, and turning the valid bit on.
4. Restart the instruction execution at the first step, which will refetch the instruction, this time finding it in the cache.

# Handling writes

- **Write-through**: A scheme in which writes always update both the cache and the next lower level of the memory hierarchy, ensuring that data is always consistent between the two.
- **Write buffer**: A queue that holds data while the data is waiting to be written to memory.
- **Write-back**: A scheme that handles writes by updating values only to the block in the cache, then writing the modified block to the lower level of the hierarchy when the block is replaced.

# Measuring and improving cache performance

$$\text{CPU time} = (\text{CPU execution clock cycles} + \text{Memory-stall clock cycles}) \times \text{Clock cycle time}$$

$$\text{Memory-stall clock cycles} = (\text{Read-stall cycles} + \text{Write-stall cycles})$$

$$\text{Read-stall cycles} = \frac{\text{Reads}}{\text{Program}} \times \text{Read miss rate} \times \text{Read miss penalty}$$

# Measuring and improving cache performance

$$\text{Write-stall cycles} = \left( \frac{\text{Writes}}{\text{Program}} \times \text{Write miss rate} \times \text{Write miss penalty} \right) + \text{Write buffer stalls}$$

$$\text{Memory-stall clock cycles} = \frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty}$$

We can also factor this as

$$\text{Memory-stall clock cycles} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}$$

# Calculating cache performance.

- Assume the miss rate of an instruction cache is 2% and the miss rate of the data cache is 4%. If a processor has a CPI of 2 without any memory stalls, and the miss penalty is 100 cycles for all misses, determine how much faster a processor would run with a perfect cache that never missed. Assume the frequency of all loads and stores is 36%.

# Fully associative cache:

- A cache structure in which a block can be placed in any location in the cache.
- To find a given block in a fully associative cache, all the entries in the cache must be searched because a block can be placed in any one. To make the search practical, it is done in parallel with a comparator associated with each cache entry.

# Set-associative cache:

- A cache that has a fixed number of locations (at least two) where each block can be placed
- (Block number) modulo ((number of sets in the cache)

# Reading

- 5.1 – 5.4