

---

# CSE 3421

## Introduction to Computer Architecture

### Chapter 6: Storage and Other I/O Topics

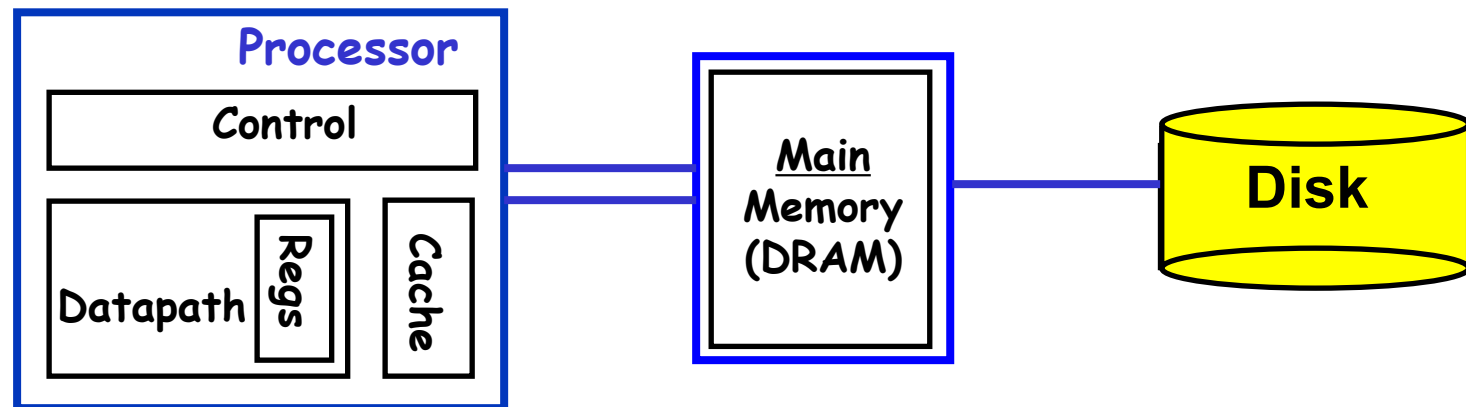
Xiaodong Zhang

Some Slides from CMU are used.

[Adapted from *Computer Organization and Design, 4<sup>th</sup> Edition*, Patterson & Hennessy, © 2008, MK]

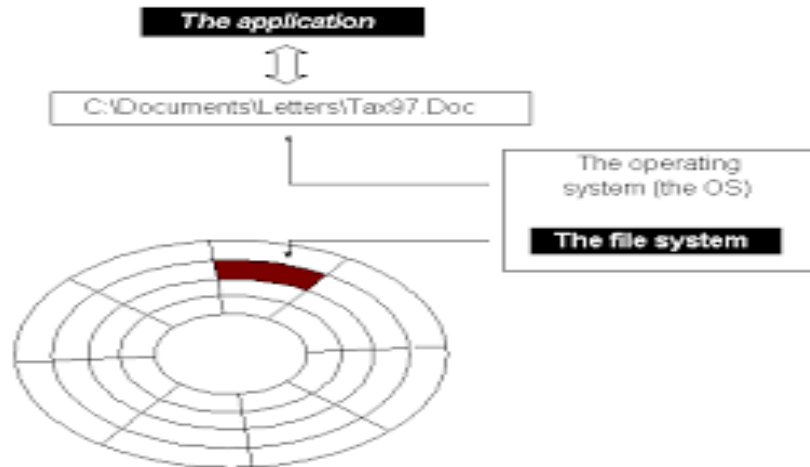
# Disk Storage is the Permanent Home of Data

- Operated by CPU/OS via **Disk Controller**
- DRAM directly interacts with Disk with the help of OS
  - DRAM needs disk for **virtual memory** functions
  - Users/applications/systems need disk to write and read their files
  - DRAM is a **dynamic execution space** (temporary and buffering); Disk is a **permanent home** (non-volatile) for data and files

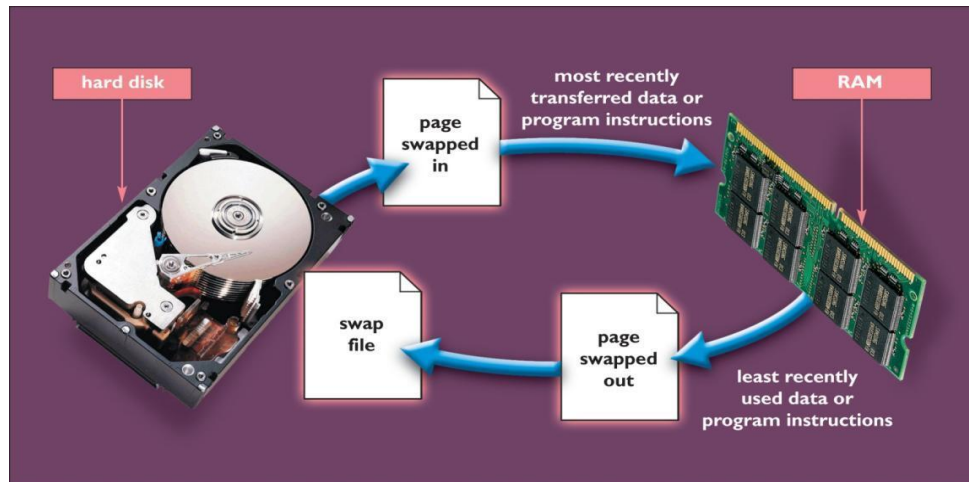


# A disk is partitioned into three regions

- **User file system areas** (e.g., writing your homework)



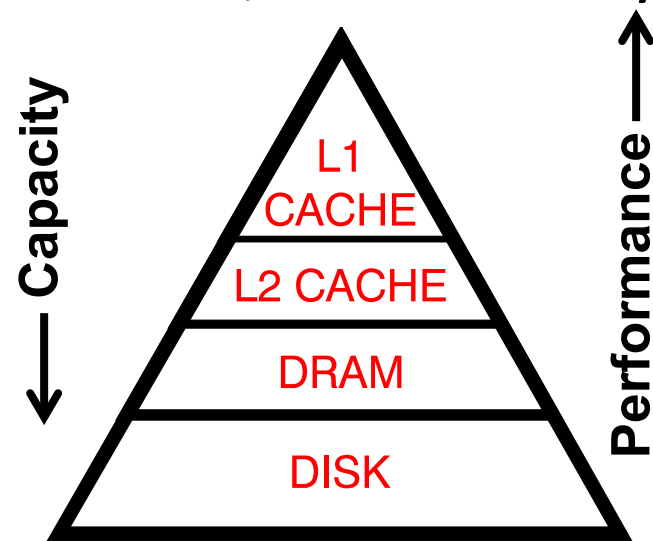
- **OS Swapping Space** (e.g., running your programs)



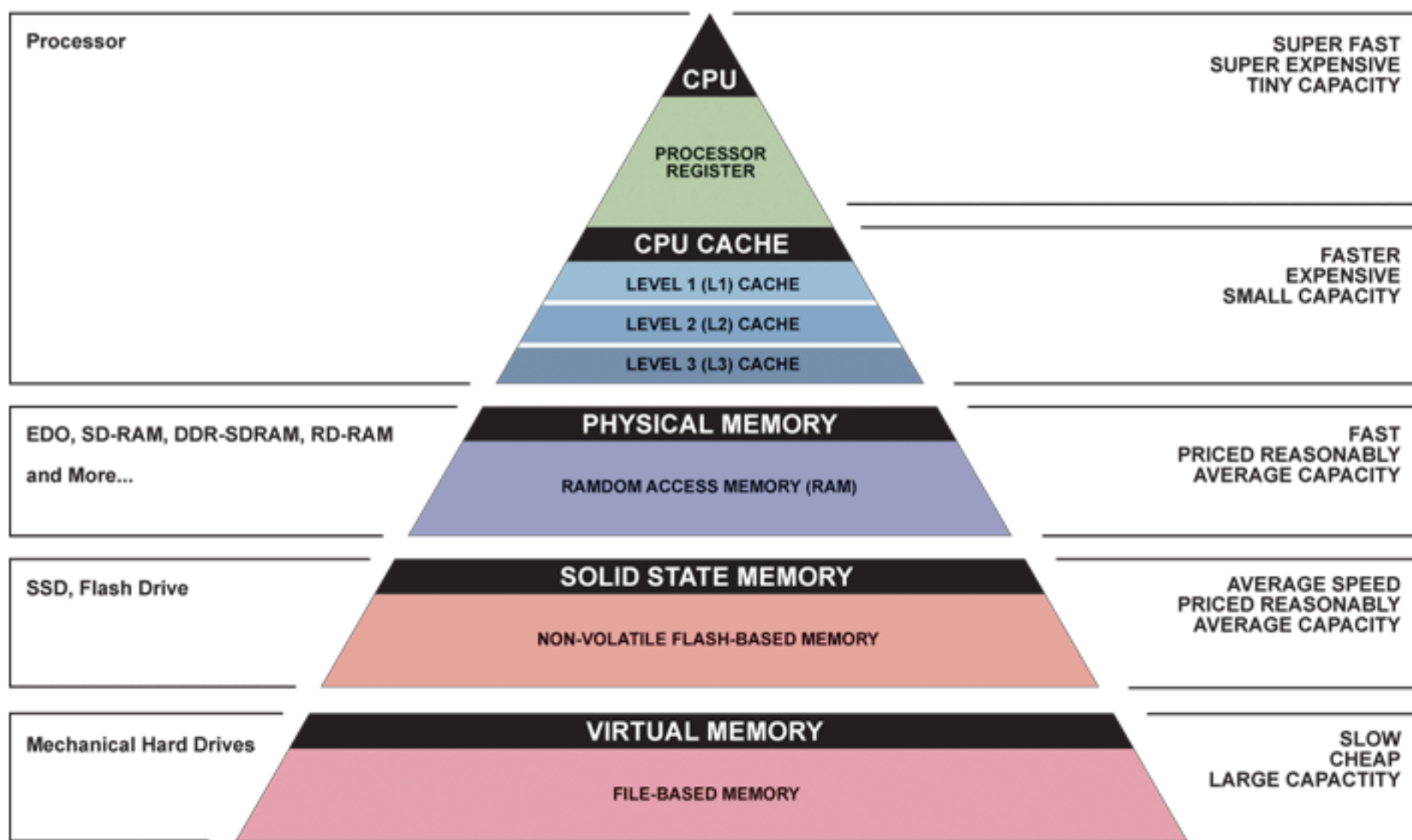
- **Files for System Administration** (OS kernels)

# Memory/storage hierarchies

- Balancing performance with cost
  - Small memories are **fast but expensive**
  - Large memories are **slow but cheap**
- Exploit locality to get the best of both worlds
  - locality = re-use/nearness of accesses
  - allows most accesses to use small, fast memory



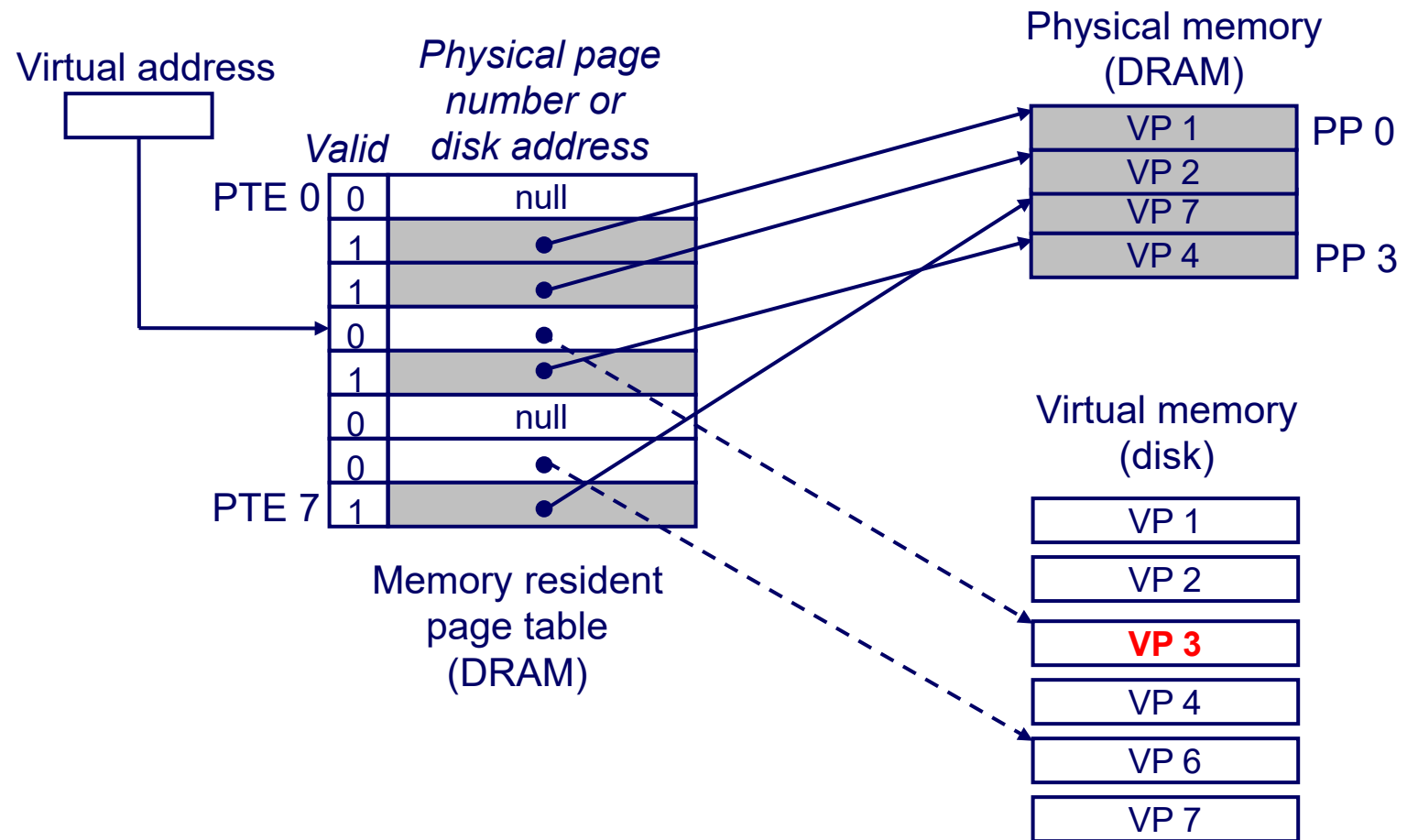
# An architectural view of **memory hierarchy**



# Page Faults Trigger Disk Accesses

A *page fault* is caused by a reference to a VM page that is not in physical (main) memory

—Example: An instruction references data blocks contained in **VP 3**, a miss that triggers a page fault exception

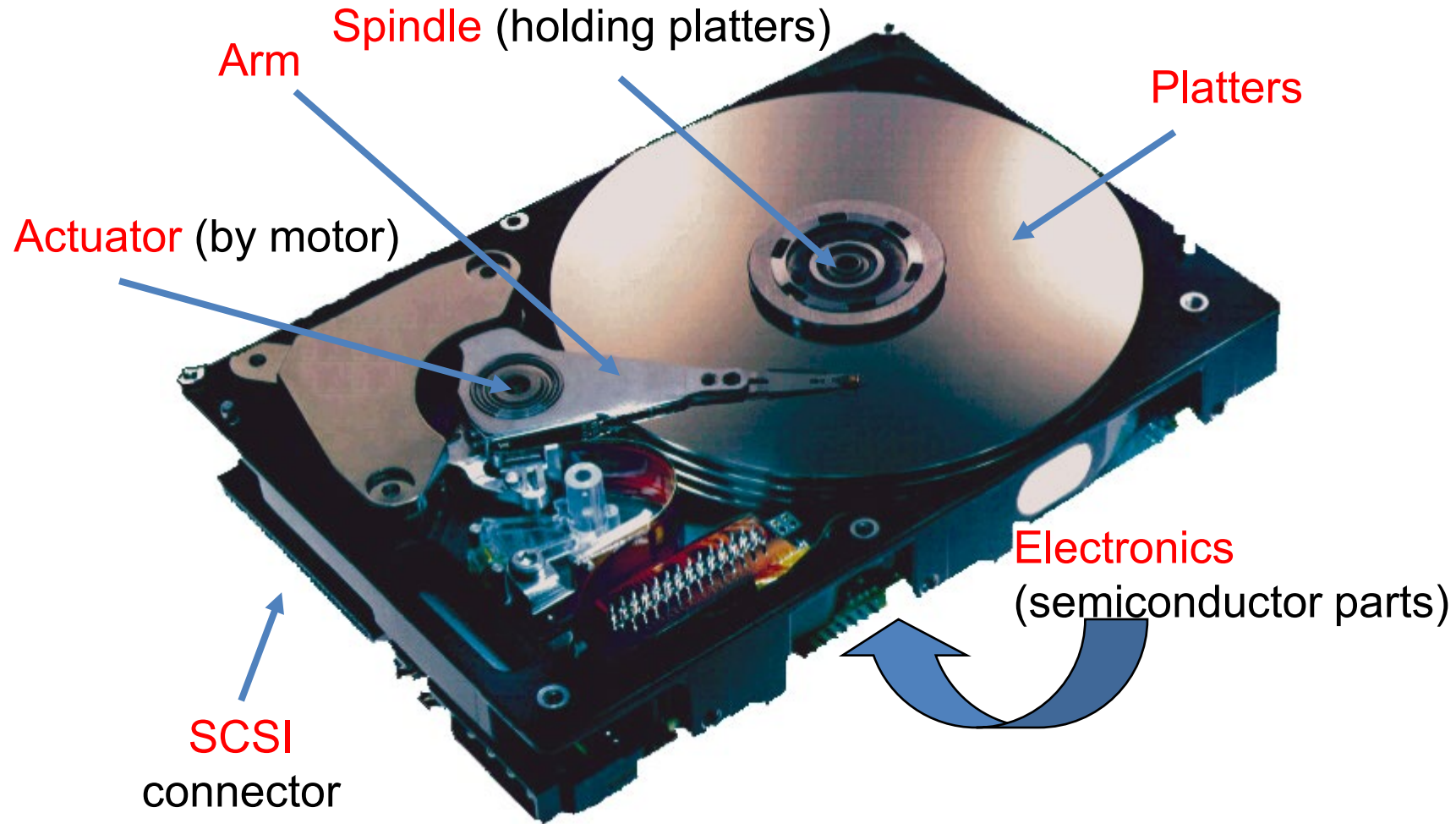


# Disk-based storage in computers

- Persistence

- Storing data for lengthy periods of time
  - DRAM/SRAM is “volatile”: contents lost if power lost
  - Disks are “non-volatile”: contents survive power outages
- To be useful, it must also be possible to find it again
  - this brings in many unique and challenging data organization, consistency, and management issues

# What's Inside A Disk Drive?



*Image courtesy of Seagate Technology*



# A Record Player Reads Music Data from a Record

By Thomas Edison

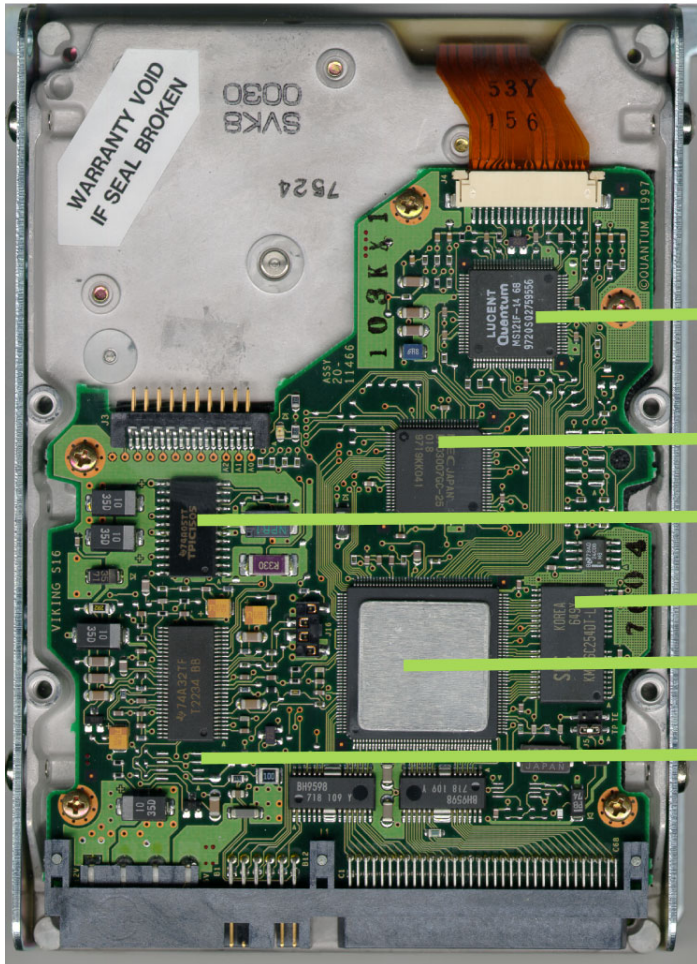
In 1877, called

Phonograph



# Disk Electronics

## Quantum Viking (circa 1997)



6 Chips

Just like a small computer  
– processor, memory,  
network interface

R/W Channel

- Connect to disk

uProcessor

32-bit, 25 MHz

- Control processor

Power Array

2 MB DRAM

- Cache memory

Control ASIC

SCSI, servo, ECC

- Control ASIC

Motor/Spindle

- Connect to motor

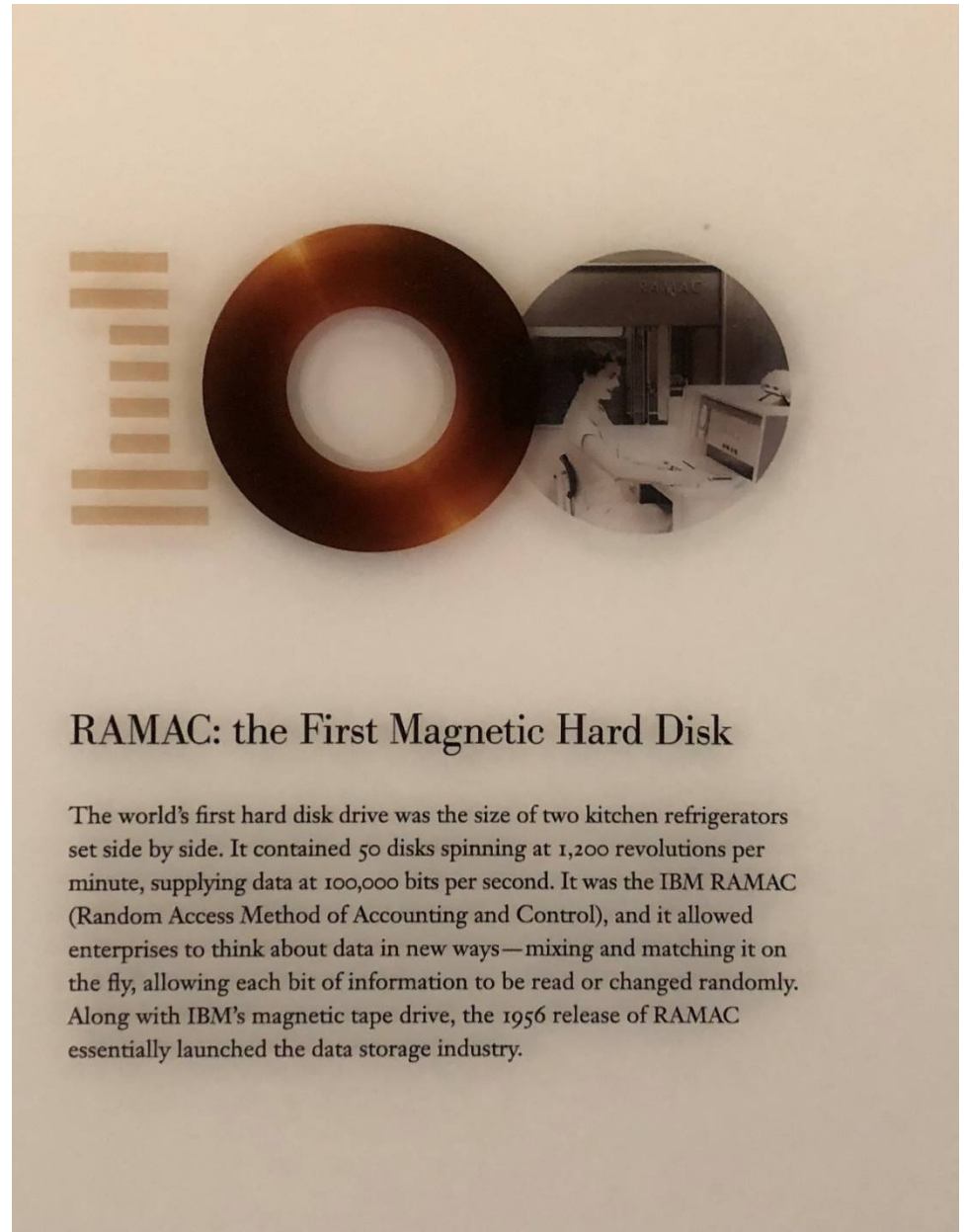
# IBM 100: 100 Innovations in 100 years (1911-2011)

A lot of innovations in hardware and software lay the foundation for daily computing operations today

Several of them are related to this class

# Seminal Contributions in Computing from IBM

- 1956: **Hard Disks**
- 5 MBytes
- 1,200 RPM



## RAMAC: the First Magnetic Hard Disk

The world's first hard disk drive was the size of two kitchen refrigerators set side by side. It contained 50 disks spinning at 1,200 revolutions per minute, supplying data at 100,000 bits per second. It was the IBM RAMAC (Random Access Method of Accounting and Control), and it allowed enterprises to think about data in new ways—mixing and matching it on the fly, allowing each bit of information to be read or changed randomly. Along with IBM's magnetic tape drive, the 1956 release of RAMAC essentially launched the data storage industry.



# Seminal Contributions in Computing from IBM

- 1957: Fortran and Magnetic Tapes



## FORTRAN: the Pioneering Programming Language

Programming early computers meant using an arcane “machine code” specific to each computer. IBM programmer John Backus found a better solution. In 1957, he and his team produced the first high-level language, FORTRAN (for FORMula TRANslator). A FORTRAN program could run on any system with a FORTRAN compiler, which translated Backus’ code to machine code almost as efficiently as a good programmer. For the first time, code was comprehensible to people other than programmers, giving mathematicians and scientists the ability to write programs they could share on different systems. FORTRAN was a significant step toward freeing software from the constraints of its hardware.



## Magnetic Tape Storage

In the late 1940s, inspired in part by Bing Crosby’s pioneering use of magnetic tape to record his radio shows, IBM engineers started experimenting with tape as a data storage successor to the punched card. 3M developed tape to IBM specifications, while IBM worked on reels with rapid start and stop times, moving tape at 100 to 200 inches per second. The engineers hit upon the idea of using a vacuum column to suck in loops of tape and buffer it from the jarring stops and starts. In 1952, IBM announced the first magnetic tape storage unit, the IBM 726.

# Seminal Contributions in Computing from IBM

- 1959: **IBM Mainframe**



## IBM 1401: the Mainframe

In 1959, IBM introduced the 1401, the first high-volume, stored-program, core-memory transistorized mainframe computer. Its versatility in running enterprise applications of all kinds helped it become the most popular computer model in the world in the early 1960s. IBM also introduced the 1403 chain printer, which launched the era of high-speed, high-volume impact printing. The 1403 was unsurpassed in quality until the advent of the laser printer in the 1970s. The 1401 was the first computer system in the world to reach 10,000 unit sales.

# Seminal Contributions in Computing from IBM

- 1962: **Speech Recognition**



## Pioneering Speech Recognition

At the Seattle World's Fair in 1962, IBM showcased the world's most advanced speech recognition system, the "Shoebox." It could understand 16 words, including the numbers zero through nine as well as minus, plus, subtotal, total, false and off. Visitors to the IBM pavilion could speak to the Shoebox via microphone, often looking on in amazement as it printed answers to simple arithmetic. After the Shoebox breakthrough, the development of speech recognition accelerated, aided by the exponential growth in computing power. The technology significantly increased computing access for people with vision, mobility and other impairments. Today, speech recognition is pervasive, and features a broad vocabulary and astonishing accuracy.

# Seminal Contributions in Computing from IBM

- 1964: IBM 360



## System/360: From Computers to Computer Systems

Few products in history have had the massive impact that the IBM System/360 has had—on technology, on the way the world works, or on the organization that created them. The System/360 ushered in the era of computer compatibility—for the first time allowing models across a product line, and even from other companies, to work with each other. It marked a turning point in the emerging field of information science. After System/360, the industry no longer talked about automating particular tasks with “computers.” Now, technology providers talked about managing complex processes through “computer systems.”



# Seminal Contributions in Computing from IBM

- 1968: **DRAM Memory**



## DRAM: the Invention of On Demand Data

In the mid 1960s, IBM researcher Bob Dennard developed the world's first one-transistor memory, calling it "dynamic random access memory," or DRAM. Finally, mainframes could be outfitted with short-term memory to act as a buffer to the data stored on disk drives. The memory chips would hold information the computer was working on right then, so it could go back to the disk drive only when it needed something new. This vastly sped up the process of accessing and using stored information. DRAM instantly made computer memory smaller, denser and cheaper, all while requiring less power.

# Seminal Contributions in Computing from IBM

- 1971: **Virtual Memory**  
in IBM OS/360



## System/360: From Computers to Computer Systems

Few products in history have had the massive impact that the IBM System/360 has had—on technology, on the way the world works, or on the organization that created them. The System/360 ushered in the era of computer compatibility—for the first time allowing models across a product line, and even from other companies, to work with each other. It marked a turning point in the emerging field of information science. After System/360, the industry no longer talked about automating particular tasks with “computers.” Now, technology providers talked about managing complex processes through “computer systems.”

# Seminal Contributions in Computing from IBM

- 1971: Floppy Disks



## The Floppy Disk

The IBM engineers who developed the floppy disk never could have dreamed that it would soon become instilled in the fabric of consumers' lives. It was originally designed for large-scale systems, as a more efficient form factor for IBM's System/370 mainframe data loads. But soon, the disk's small size and ever-increasing storage capabilities led to its adoption by smaller systems as well. Usable, durable and flexible, the floppy disk quickly became ubiquitous as the preferred storage medium for the emerging personal computer industry.

# Seminal Contributions in Computing from IBM

- 1971: Relational Database

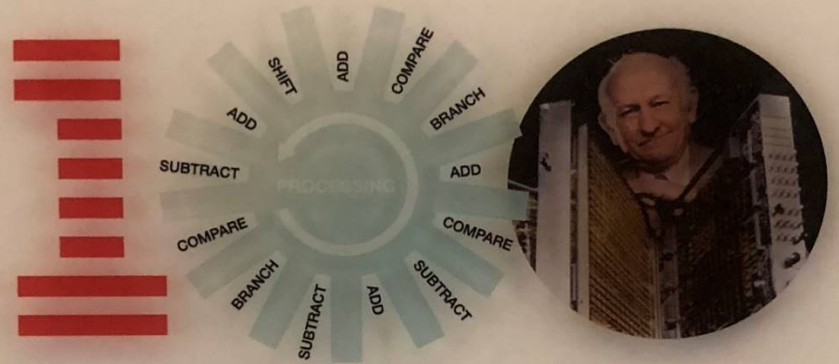


## Relational Database

Until the mid 1970s, computers sorted information using rigid, one-off database programs. IBM researcher E.F. “Ted” Codd wanted to improve the way data was sorted and handled. He sought to create a generalized description of how to store, update and extract data with accuracy, and query responses so any changes to data produced consistent results. In 1970, Codd completed his definition of the relational database which became the foundation for IBM DB2 products.

# Seminal Contributions in Computing from IBM

- 1980: RISC Processor



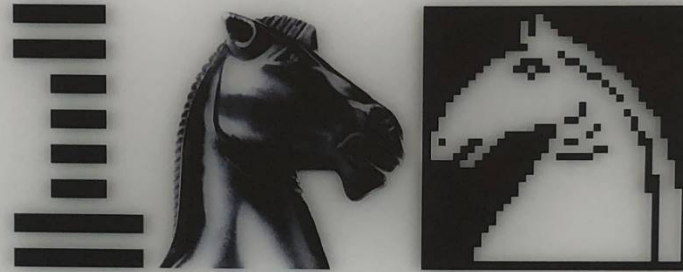
## RISC Architecture

The first prototype computer employing RISC (reduced instruction set computer) architecture was developed at IBM in 1980. By allowing commands to access previously unused memory space, RISC enabled computers to work approximately twice as fast as other machines on the same number of circuits. RISC was an important innovation in system design because it eliminated wasted space in the information pipeline, and was widely viewed as the dominant computing architecture of the future. Its creator, John Cocke, received for his efforts the U.S. National Medal of Science (1994) and the U.S. National Medal of Technology (1991).



# Seminal Contributions in Computing from IBM

- 1997: Deep Blue



## Deep Blue

In the mid 1980s, two Ph.D. students at Carnegie Mellon University, Murray Campbell and Feng-hsiung Hsu, set out to build a chess machine that could beat the best human player. IBM Research hired the two scientists and gave them the resources to build Deep Blue, a dedicated chess-playing supercomputer. In 1997, in a historic match, Deep Blue became the first computer to defeat a reigning world chess champion. "In brisk and brutal fashion," *The New York Times* reported, "the IBM computer Deep Blue unseated humanity, at least temporarily, as the finest chess playing entity on the planet." After a noted absence, Deep Blue led the way for IBM's return to the supercomputing business.

# Seminal Contributions in Computing from IBM

- 2011: **Watson AI Machine**

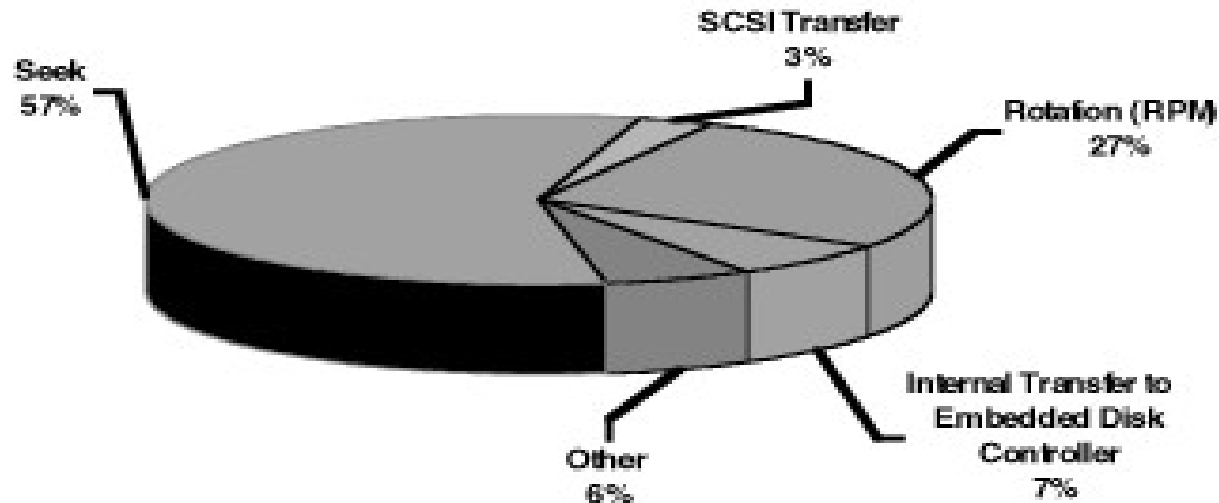


## A Computer Called Watson

IBM's computer, code-named "Watson" leverages leading-edge Question-Answering technology, allowing the computer to process and understand natural language. It incorporates massively parallel analytical capabilities to emulate the human mind's ability to understand the actual meaning behind words, distinguish between relevant and irrelevant content, and ultimately, demonstrate confidence to deliver precise final answers. In February of 2011, Watson made history by not only being the first computer to compete against humans on television's venerable quiz show, *Jeopardy!*, but by achieving a landslide win over prior champions Ken Jennings and Brad Rutter.

# High power/latency from mechanical operations

Relative Size of Disk I/O Time Components for Random I/O



- **A dominant disk access time comes from mechanical operations**
  - Seek (57%) + rotation (27%) + data fetch (7%) + other overhead (6%) = 97%
  - After data is prepared from disk, data transfer time via SCSI bus is only 3%

Source: Configuration and Capacity Planning for Solaris Servers, Sun Microsystems

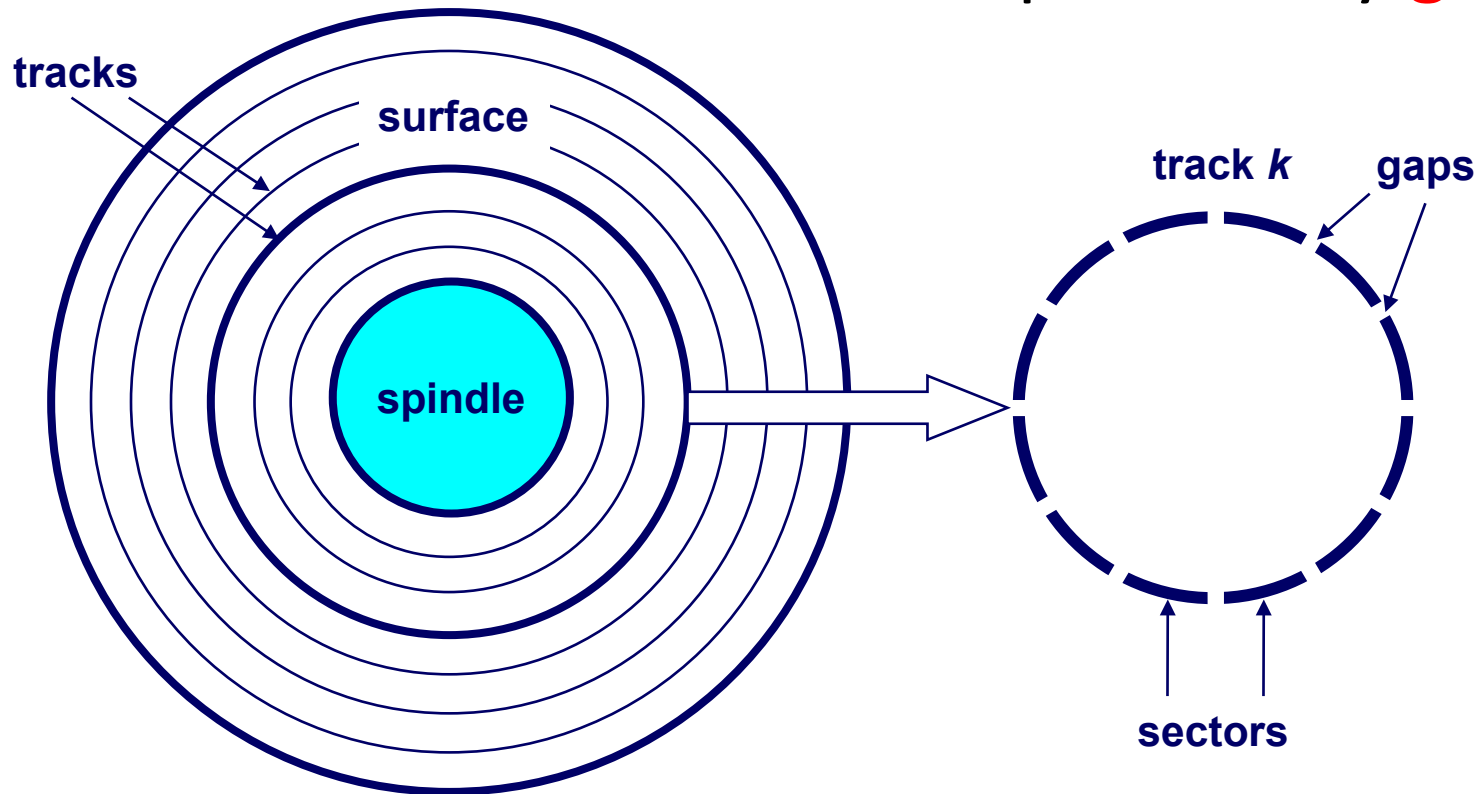


# Disk “Geometry”

Disks contain **platters**, each with two **surfaces**

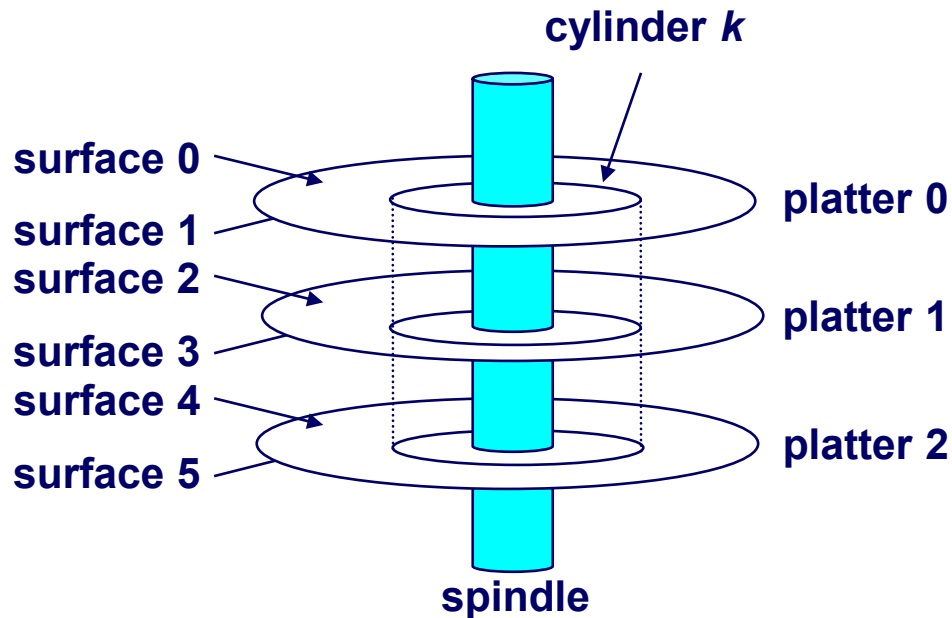
Each surface organized in concentric rings called **tracks**

Each track consists of **sectors** separated by **gaps**



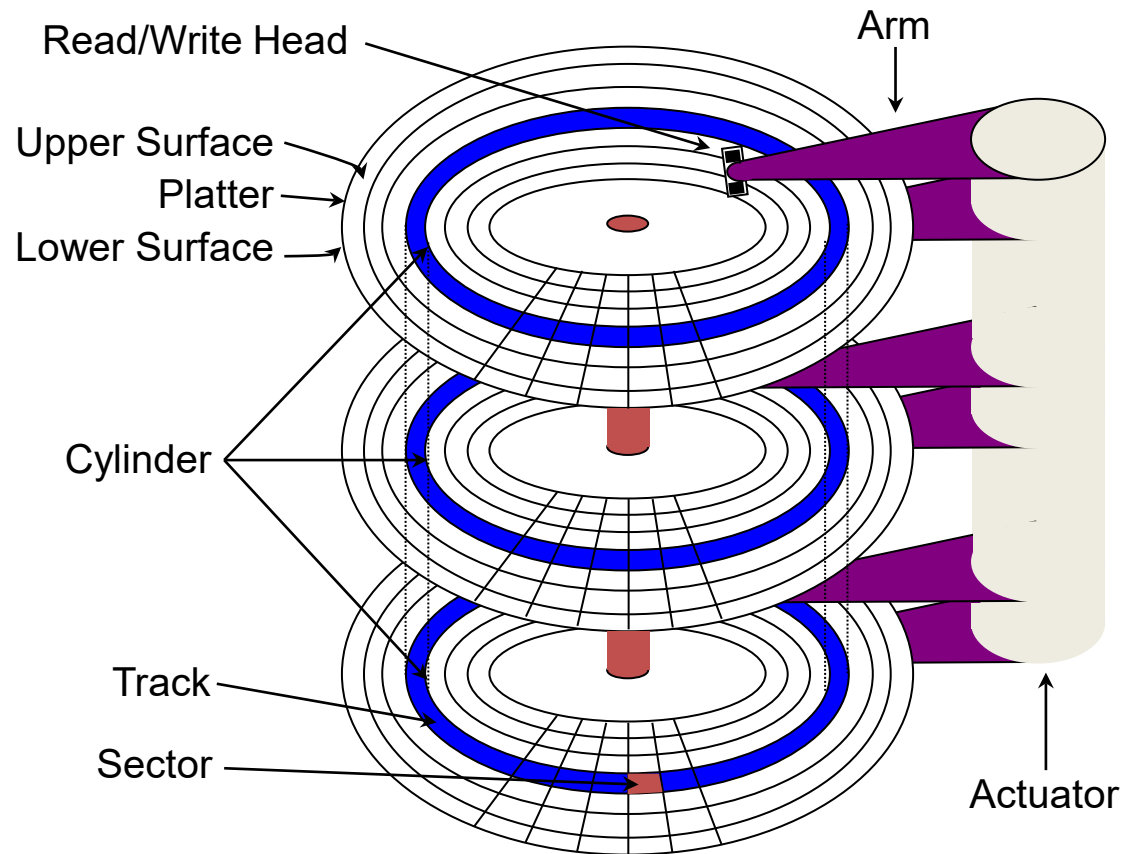
# Disk Geometry (Multiple-Platter View)

Aligned tracks form a cylinder



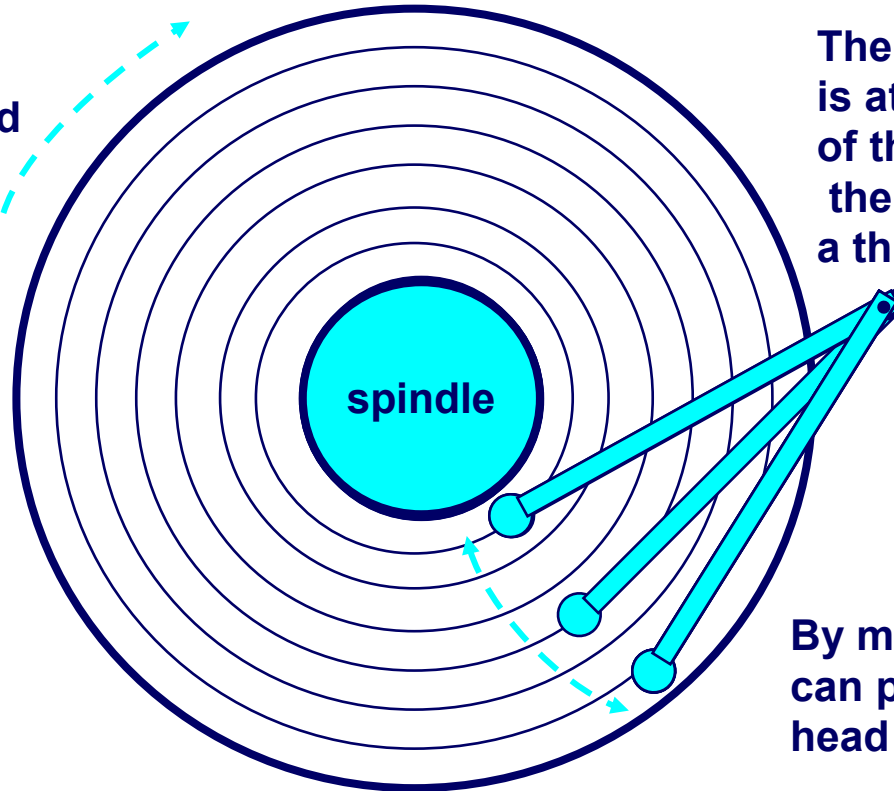
A **cylinder** consists all the tracks of equal diameter, vertically it forms a “cylinder” for parallel data accessing

# Disk Structure



# Disk Operation (Single-Platter View)

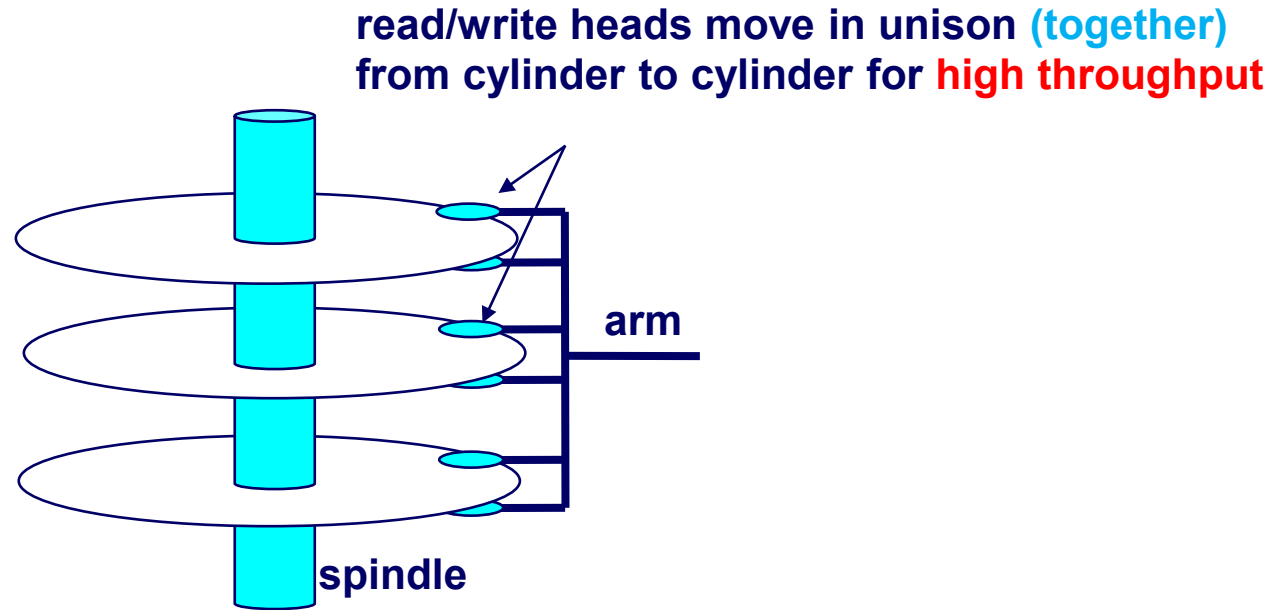
The disk surface spins at a fixed rotational rate



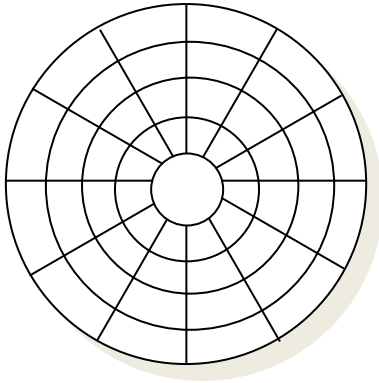
The read/write *head* is attached to the end of the *arm* and flies over the disk surface on a thin cushion of air

By moving radially, the arm can position the read/write head over any track

# Disk Operation (Multi-Platter View)



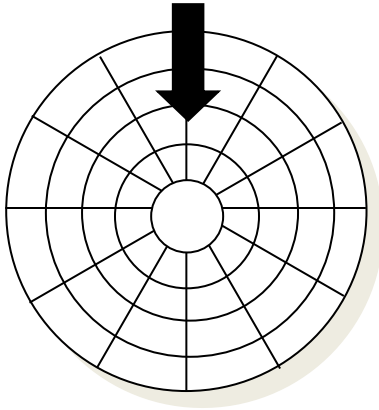
# Disk Structure - top view of single platter



Surface organized into tracks

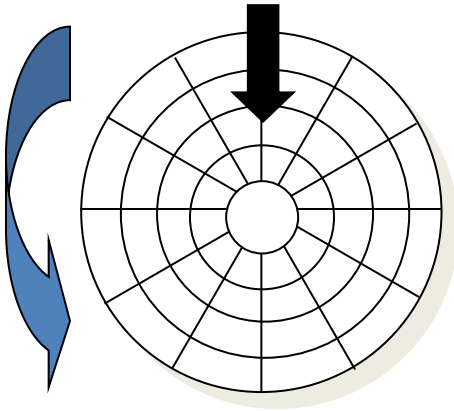
Tracks divided into sectors

# Disk Access



Head in position above a track

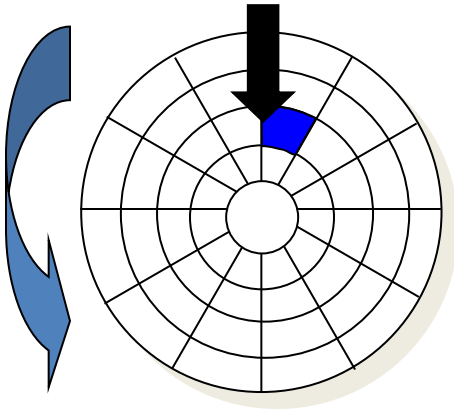
# Disk Access



Rotation is counter-clockwise

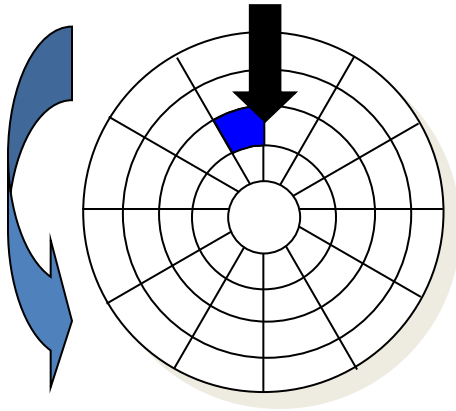


# Disk Access – Read



About to read blue sector

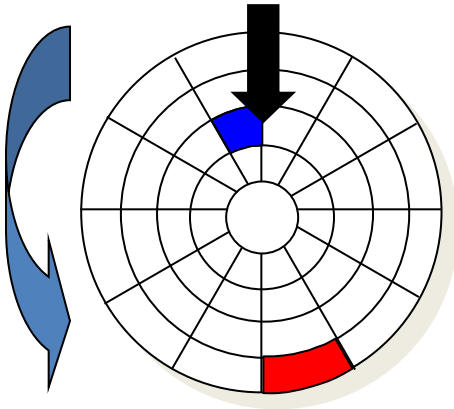
# Disk Access – Read



After **BLUE**  
read

After reading blue sector

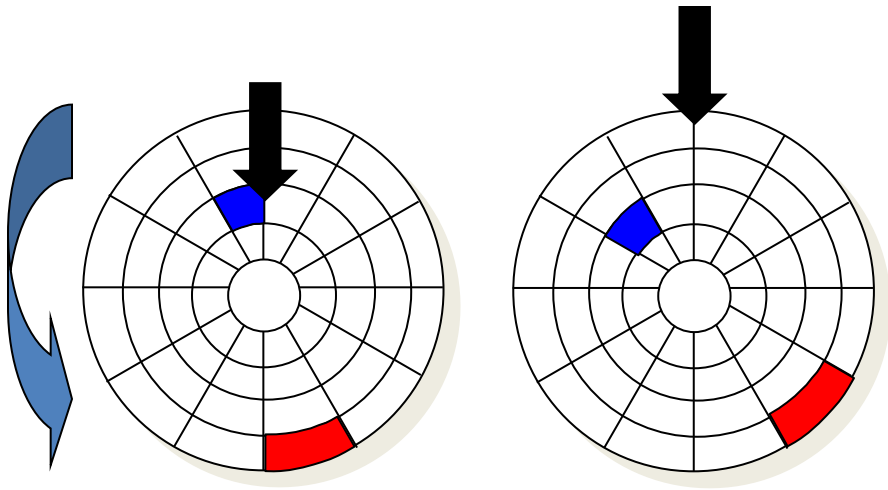
# Disk Access – Read



After BLUE  
read

Red request scheduled next

# Disk Access – Seek

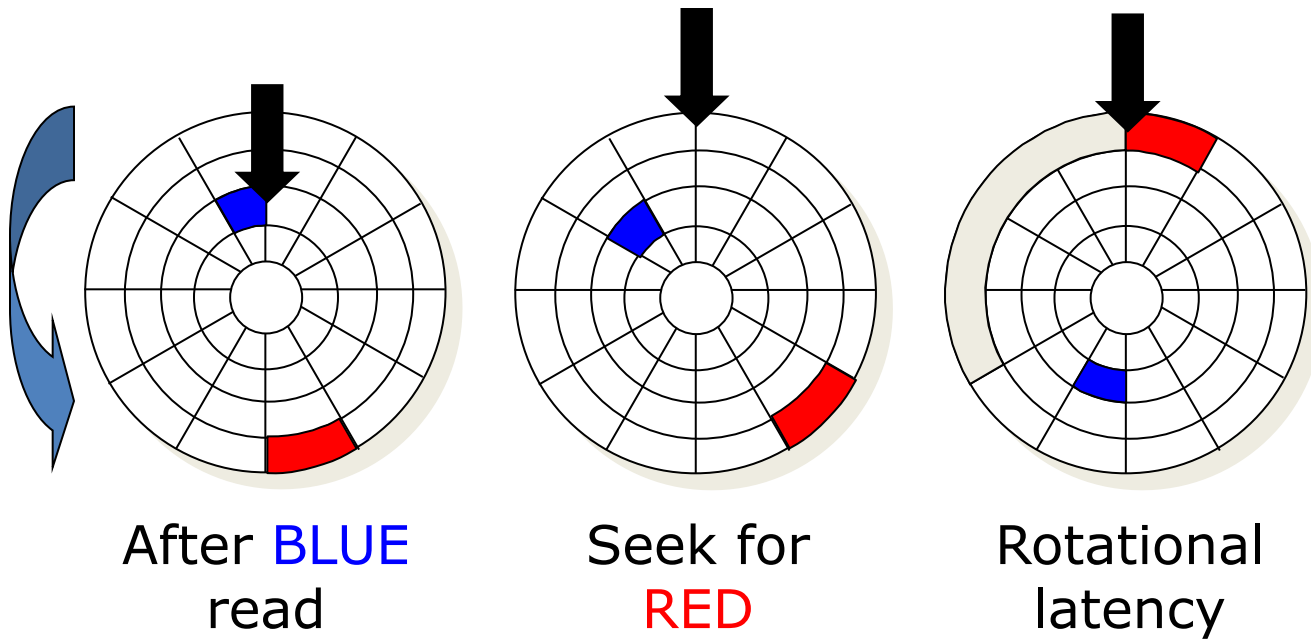


After **BLUE**  
read

Seek for  
**RED**

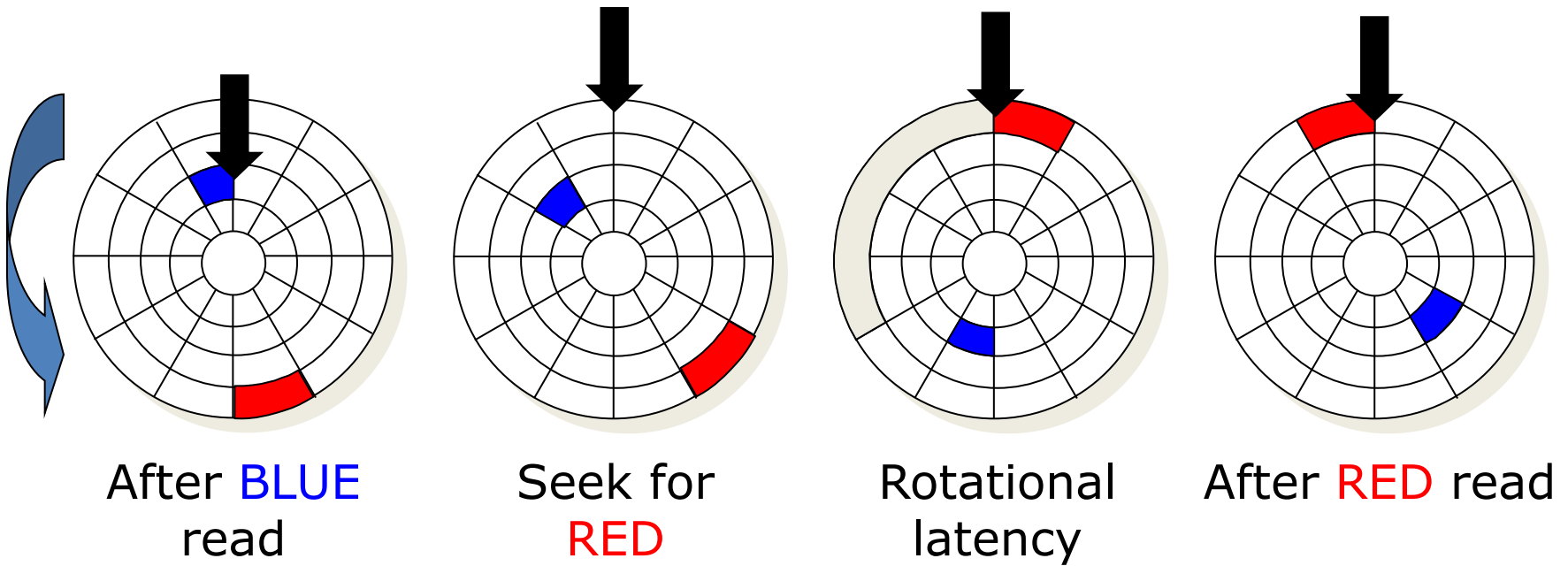
Seek to red's track

# Disk Access – Rotational Latency



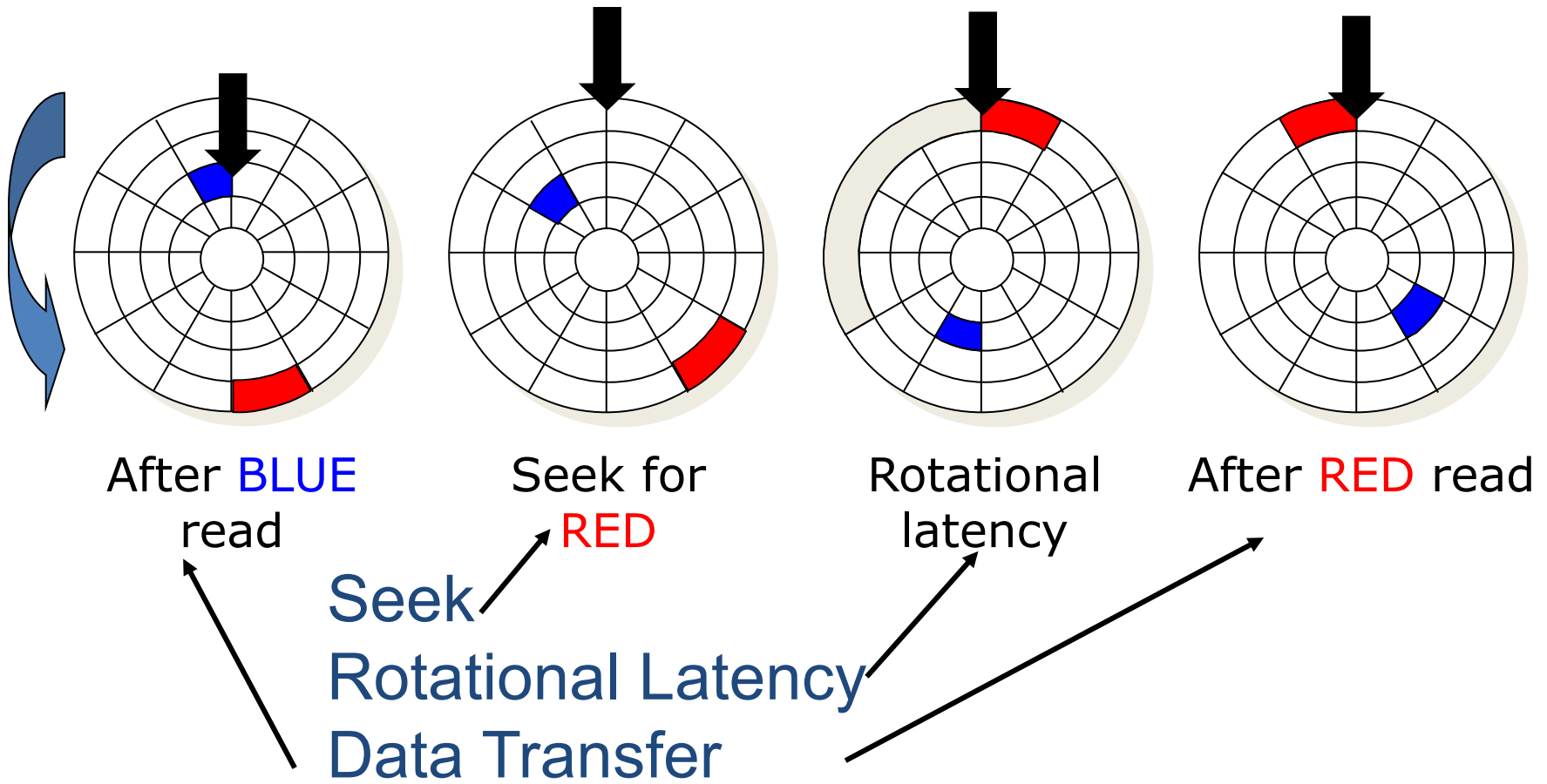
Wait for red sector to rotate around

# Disk Access – Read



Complete read of red

# Disk Access – Service Time Components



# Disk Access Time

Average time to access a specific sector is close to

$$T_{\text{access}} = T_{\text{avg-seek}} + T_{\text{avg-rotation}} + T_{\text{avg-transfer}}$$

## Seek time ( $T_{\text{avg-seek}}$ )

- Time to position head in the target track
- Typical  $T_{\text{avg-seek}} = 3\text{-}5\text{ ms}$  (given by a disk vendor)

## Rotational latency ( $T_{\text{avg-rotation}}$ )

- Time for the target sector to pass under r/w head (**best case:** current sector, **worst case:** whole rotation,  **$\frac{1}{2}$  is average rotation**)
- $T_{\text{avg-rotation}} = \frac{1}{2} \times \frac{1}{\text{RPMs}} \times 60\text{ sec}/1\text{ min}$ 
  - e.g., 3ms for 10,000 RPM disk

## Transfer time ( $T_{\text{avg-transfer}}$ )

- Time to read the bits in the target sector (scan the entire sector)
- $T_{\text{avg transfer}} = \text{whole track rotation time} \times \frac{1}{(\text{avg \# sectors/track})}$ 
  - e.g., 0.006ms for 10,000 RPM disk with 1,000 sectors/track
  - given 512-byte sectors,  $\sim 85\text{ MB/s}$  data transfer rate ( $512\text{B} / 6 \times 10^{-6}\text{s}$ )



# Disk Transfer Time

## Transfer time ( $T_{\text{avg-transfer}}$ )

– **Sector Transfer Time** to read the bits in the target sector (scan the entire sector)

= whole track rotation time  $\times$   $1/(\text{avg \# sectors/track})$

- for 10,000 RPM disk with 1,000 sectors/track, we have
- $1/\text{RPMs} \times 60 \text{ sec}/1 \text{ min} \times 1/1000 \text{ sectors/track} = 0.006 \text{ ms}$

– given 512-Byte sectors, we have sector size / sector transfer time

=  $512\text{B} / 6 \times 10^{-6}\text{s} = 85.33 \text{ MB/s}$  data transfer rate

## In comparison:

First chip (Intel 4004, 1971): 750KHz, 12 bit address (4KBytes)

Today (Intel Core i7): 4.2GHz, 64 bit address (18.4EB,  $2^{18}$ )

5,600 times faster in CPU, disk improvement is only a few times

# Disk Access Time Example

## Given:

- Rotational rate = 7,200 RPM (rotation per minute)
- Average seek time = 5 ms
- Avg # sectors/track = 1000

## Derived average time to access random sector:

- $T_{\text{avg rotation}} = 1/2 \times (60 \text{ secs}/7200 \text{ RPM}) \times 1000 \text{ ms/sec} = 4 \text{ ms}$
- $T_{\text{avg transfer}} = 60/7200 \text{ RPM} \times 1/1000 \text{ secs/track} \times 1000 \text{ ms/sec} = 0.0083 \text{ ms}$
- $T_{\text{access}} = 5 \text{ ms} + 4 \text{ ms} + 0.008 \text{ ms} = 9.008 \text{ ms}$

## Important points:

- Access time dominated by **seek time** and rotational latency
- **First bit** in a sector is the most expensive, the rest are free
- SRAM access time is about 4 ns/double-word, DRAM about 60 ns
  - Need about **~100,000 times longer** to access a word on disk than in DRAM

# Interacting between OS and Disk

- OS is memory-address-centric
  - Byte addressable
  - Mapping to cache blocks
  - Allocate pages in both DRAM and the shared cache
- Disk has its own address space
  - A sector (512 Bytes) is a basic unit of data
  - How does OS write data to and read data from disk?
- A “Block Device” makes this happen
  - OS and disk communicate to each other by “blocks”

# Moving Archived Goods from Home to a Storage House

- Setting a standard for the sizes of box
  - Small, medium, and large
- Setting a standard for transportation vehicles
  - Size of the truck, speed and throughput
- Setting a standard of an interface
  - ID of the home, number of boxes and types
  - Management of moving-in and moving-out

# Disk storage as an array of “blocks”



OS's view of storage device  
(as exposed by SCSI or IDE/ATA protocols)

- **“logical block”** size is determined at booting time by OS
  - OS virtual memory system: 4-8 KB, buffer cache: 1 KB
  - Database block size: 8 KB
- The HDD sector size is 512 bytes (a physical disk unit)
  - A 4 KB logical block would be allocated in 8 contiguous sectors
- A **“block device”**: a storage device reads/writes a block of a fixed size at a time
- Note: **SCSI** (small computer system interface), **IDE** (Integrated Drive Electronics) and **ATA** (AT bus Attachment) are disk interface standards

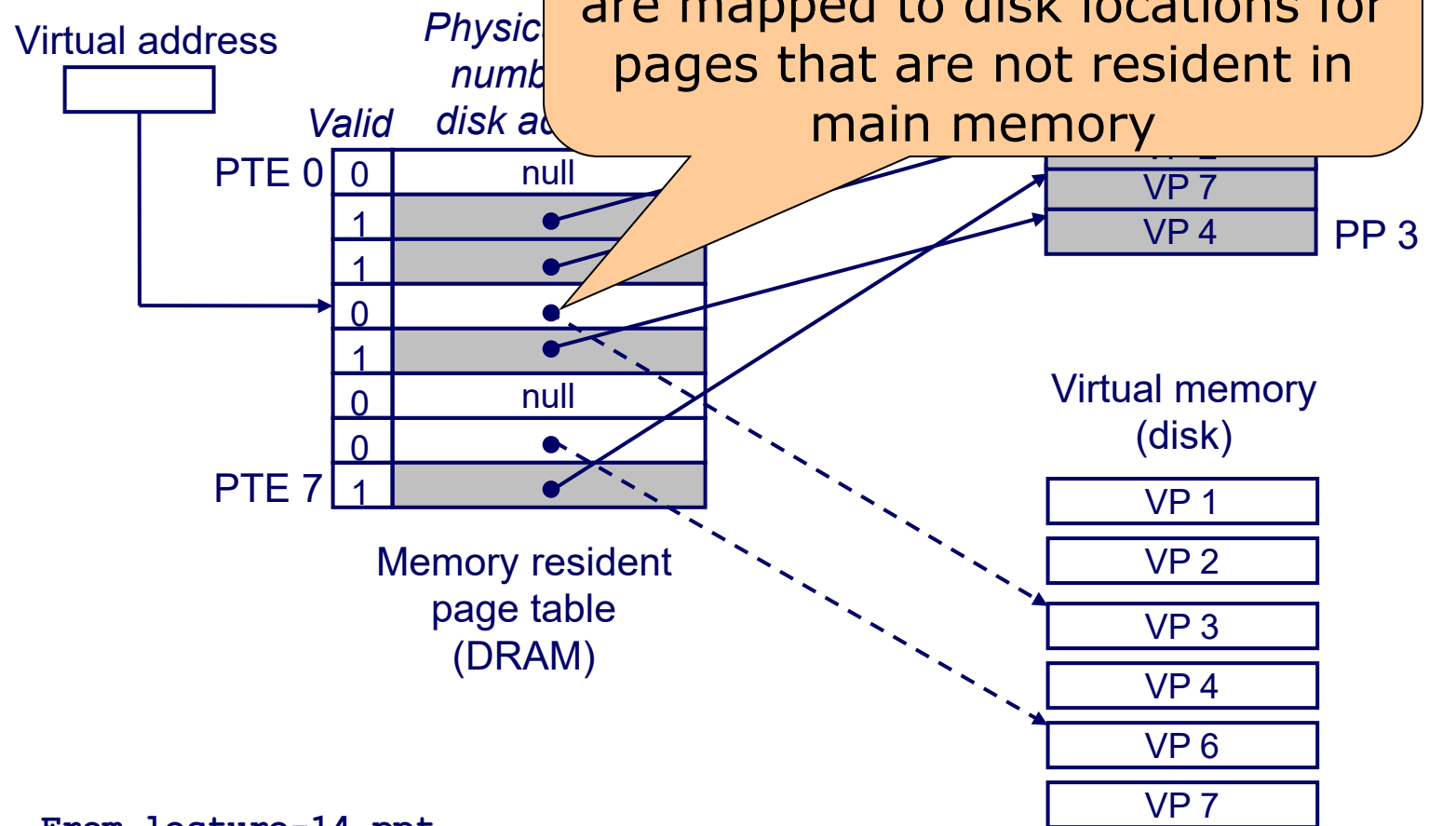
# Disk Controller

- Disk is not directly managed by OS
  - OS only knows the physical memory space
  - By MMU, OS creates a page table for each process
- Disk controller managed the data stored in disk
  - OS interact with the disk controller for I/O
  - The disk controller takes LBNs from OS to find them or write them in the disk
  - The logical blocks managed by OS are read from the disk and physically stored there by the disk controller

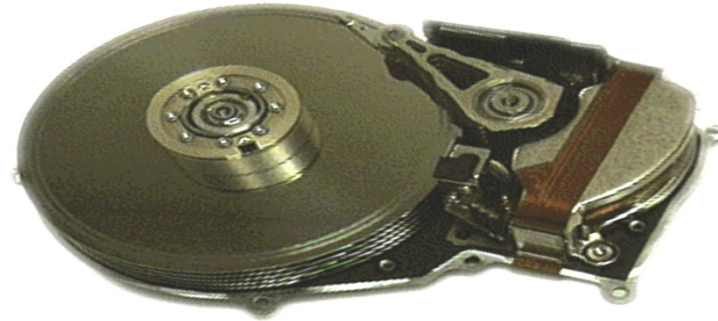
# Page Faults

A *page fault* is caused by a reference to a VM page that is not in physical (main) memory

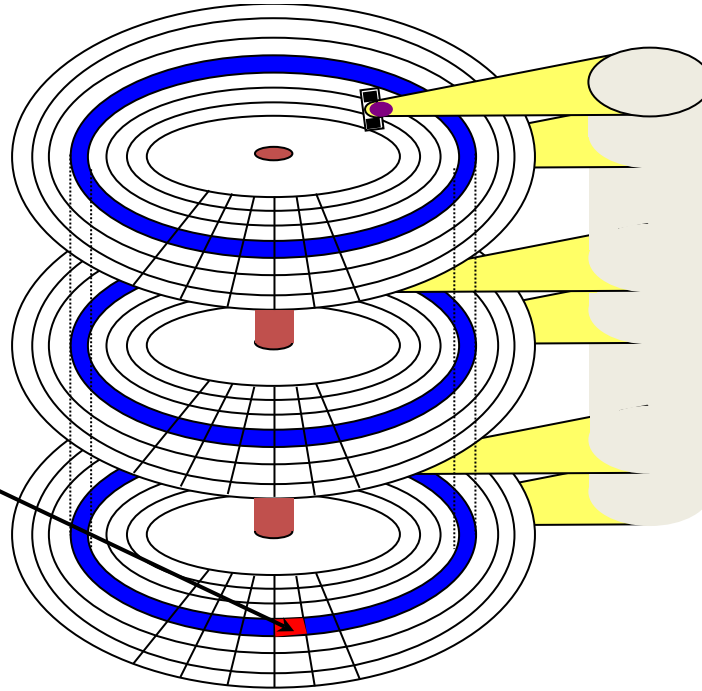
—Example: An instruction references a page that is not in physical memory, causing a page fault exception



# In device, “blocks” are mapped to physical store

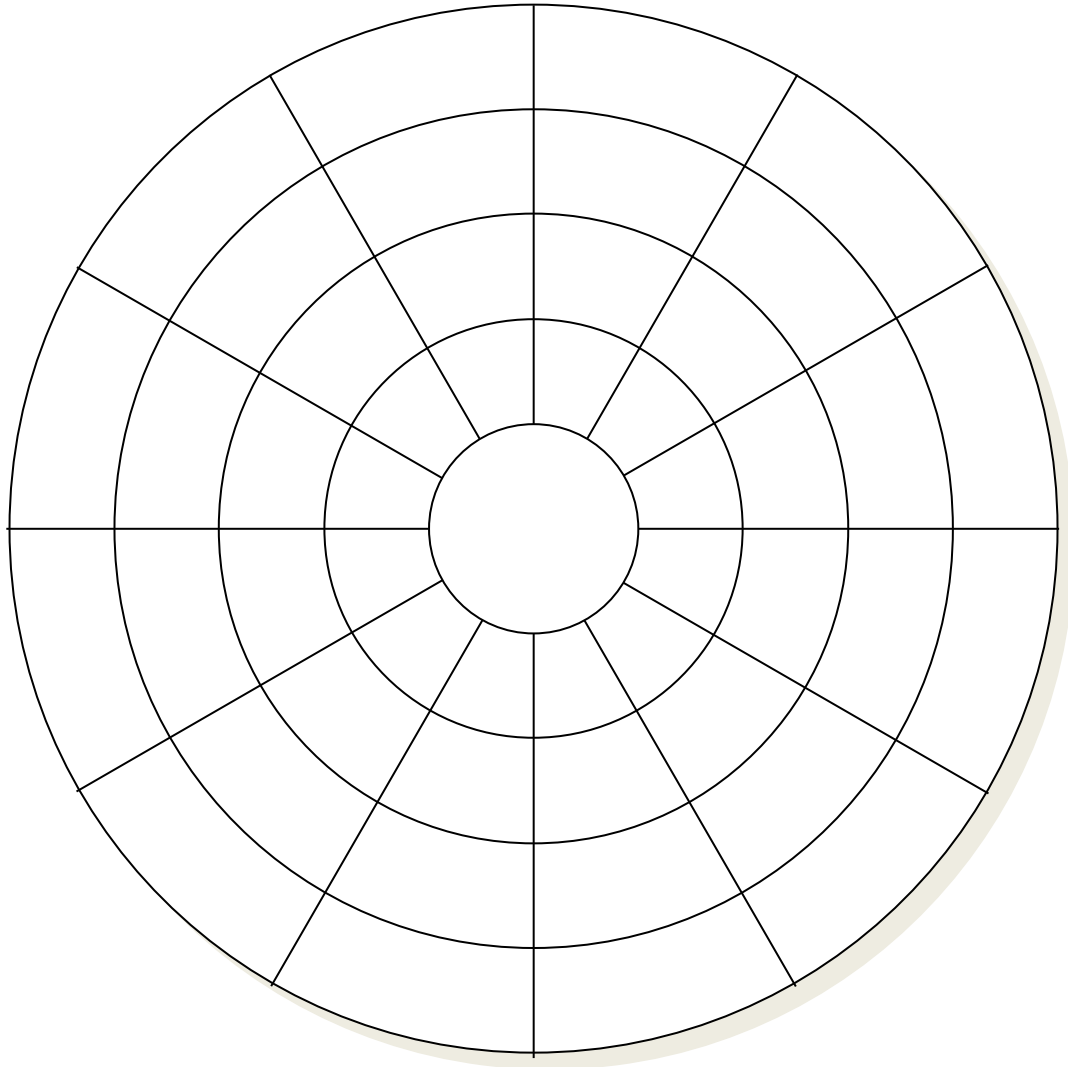


**Disk Sector**  
(512 Bytes as a unit.  
A logical block is  
contiguously stored  
in multiple sectors. )

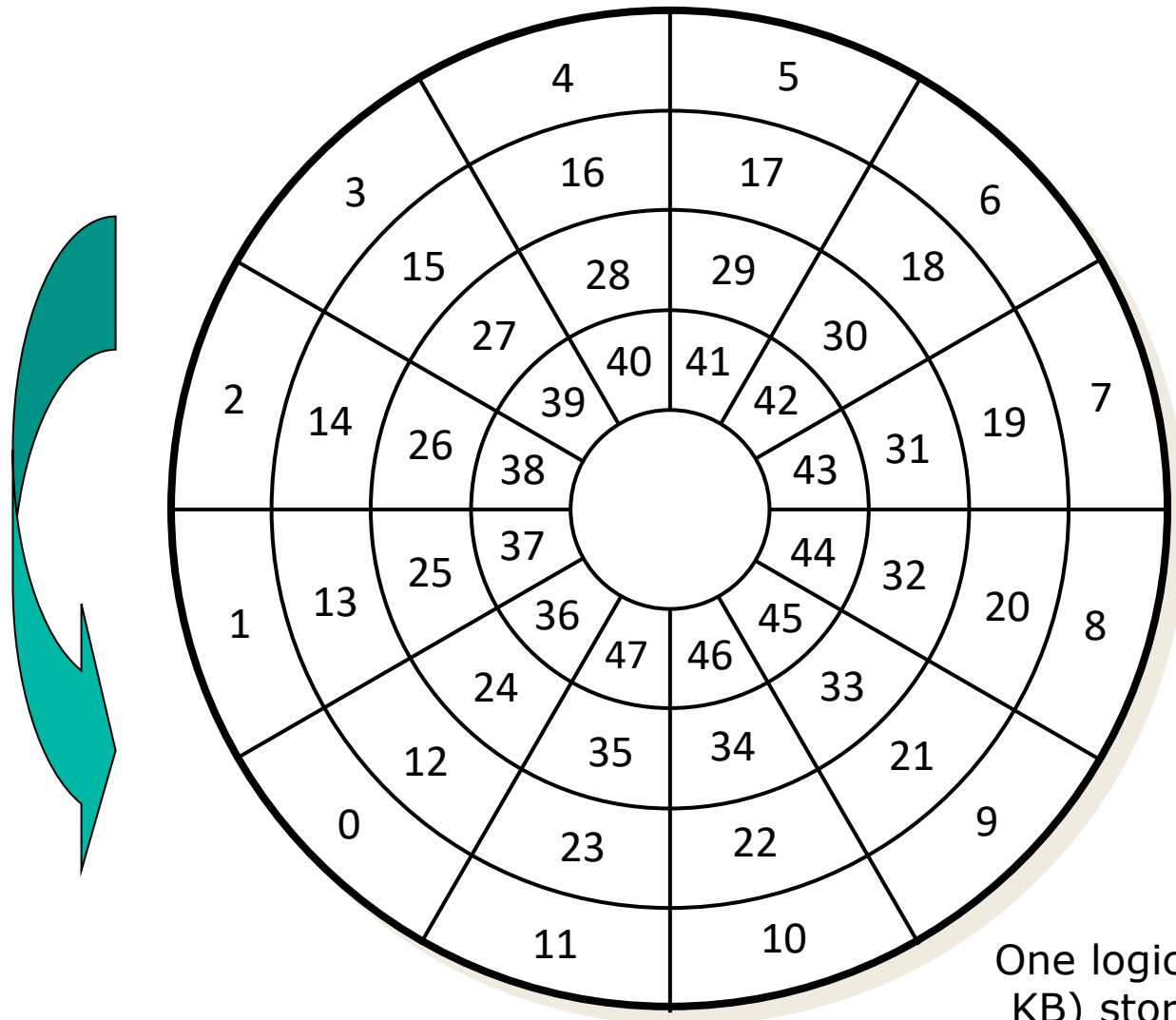




# Physical sectors of a single-surface disk



# LBN-to-physical for a single-surface disk



One logical block (1, 4, or 8 KB) stores in multiple disk sectors (512 Bytes each)

# Disk Capacity

**Capacity:** maximum number of bits that can be stored

- Vendors express capacity in units of gigabytes (GB), where  $1 \text{ GB} = 10^9 \text{ Bytes}$  (Lawsuit pending! Claims deceptive advertising, to Seagate on “not have the capability of performing as advertised”, 2016)

Capacity is determined by these technology factors:

- **Recording density** (bits/in): number of bits that can be squeezed into a 1 inch linear segment of a track
- **Track density** (tracks/in): number of tracks that can be squeezed into a 1 inch radial segment
- **Areal density** (bits/in<sup>2</sup>): product of recording and track density

A house has two areas:

**building area** (tracks in disks) and **effective usage area** (bits in disks)

# Computing Disk Capacity

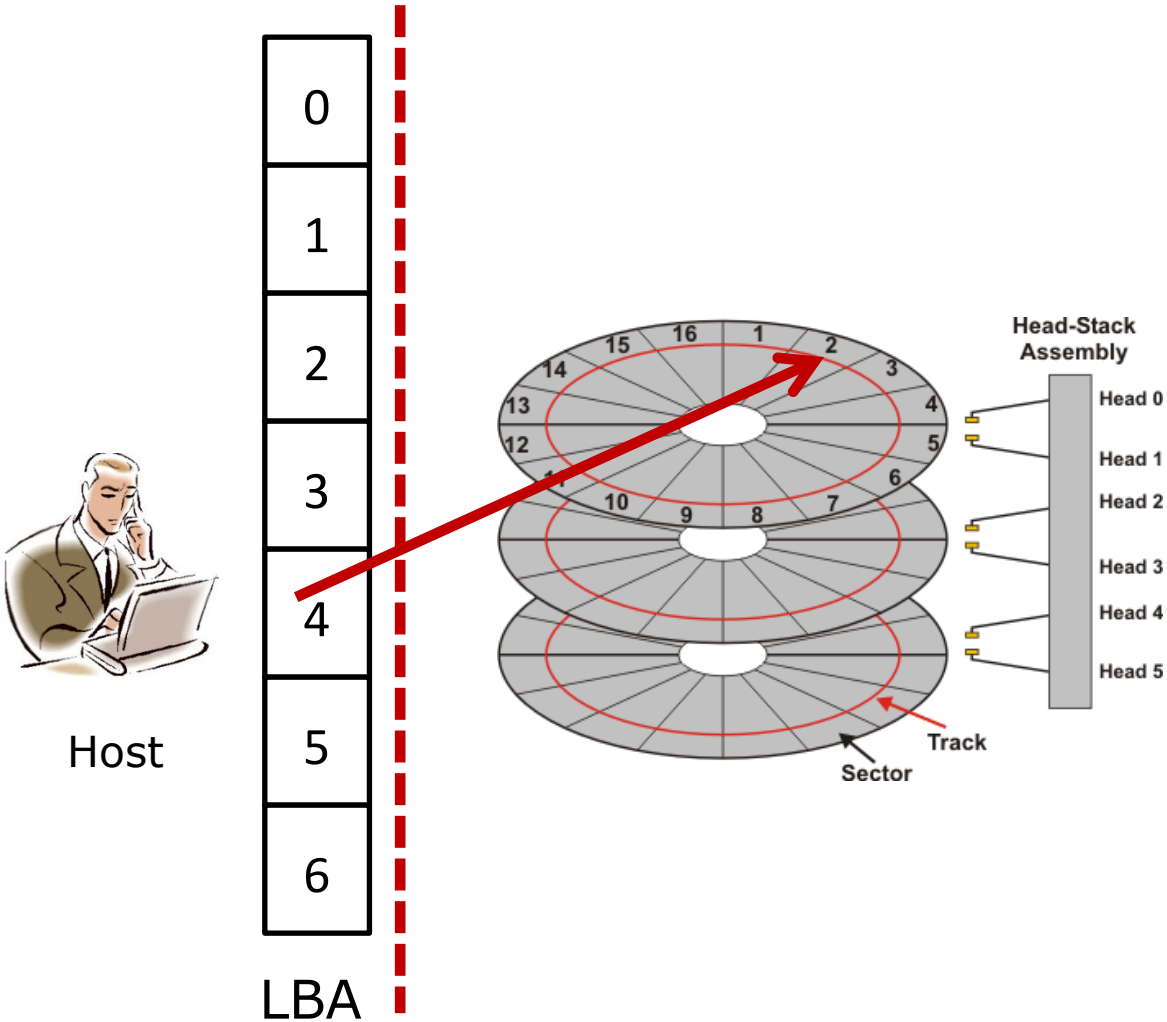
$$\text{Capacity} = (\# \text{ bytes/sector}) \times (\text{avg. } \# \text{ sectors/track}) \times \\ (\# \text{ tracks/surface}) \times (\# \text{ surfaces/platter}) \times \\ (\# \text{ platters/disk})$$

## Example:

- 512 bytes/sector
- 1000 sectors/track (on average)
- 20,000 tracks/surface
- 2 surfaces/platter
- 5 platters/disk

$$\text{Capacity} = 512 \times 1000 \times 20000 \times 2 \times 5 \\ = 102.4 \text{ GB}$$

# Physical data layout in HDD

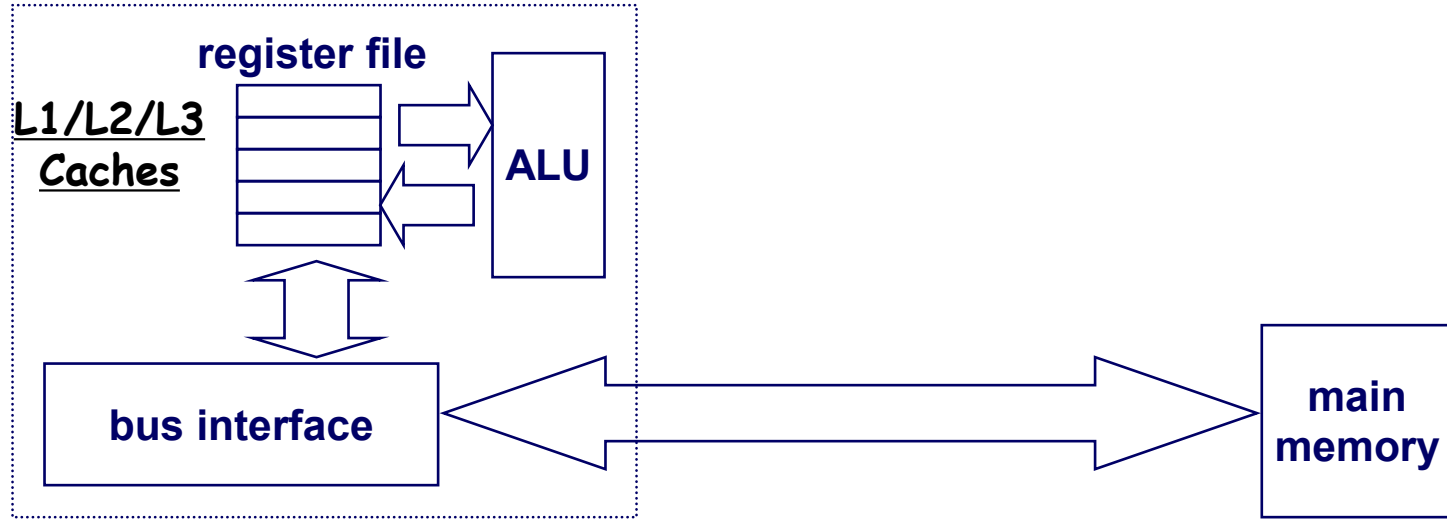


```
Read (LBA, size)
Write(LBA, size)
```

- Data are stored on the surfaces of disk platters
- An array of **logical block addresses** (LBAs) as a logical interface
- LBAs are **statically** mapped to physical block addresses (PBAs), the **sectors** contiguously (size = # blocks)

# Looking back at the hardware

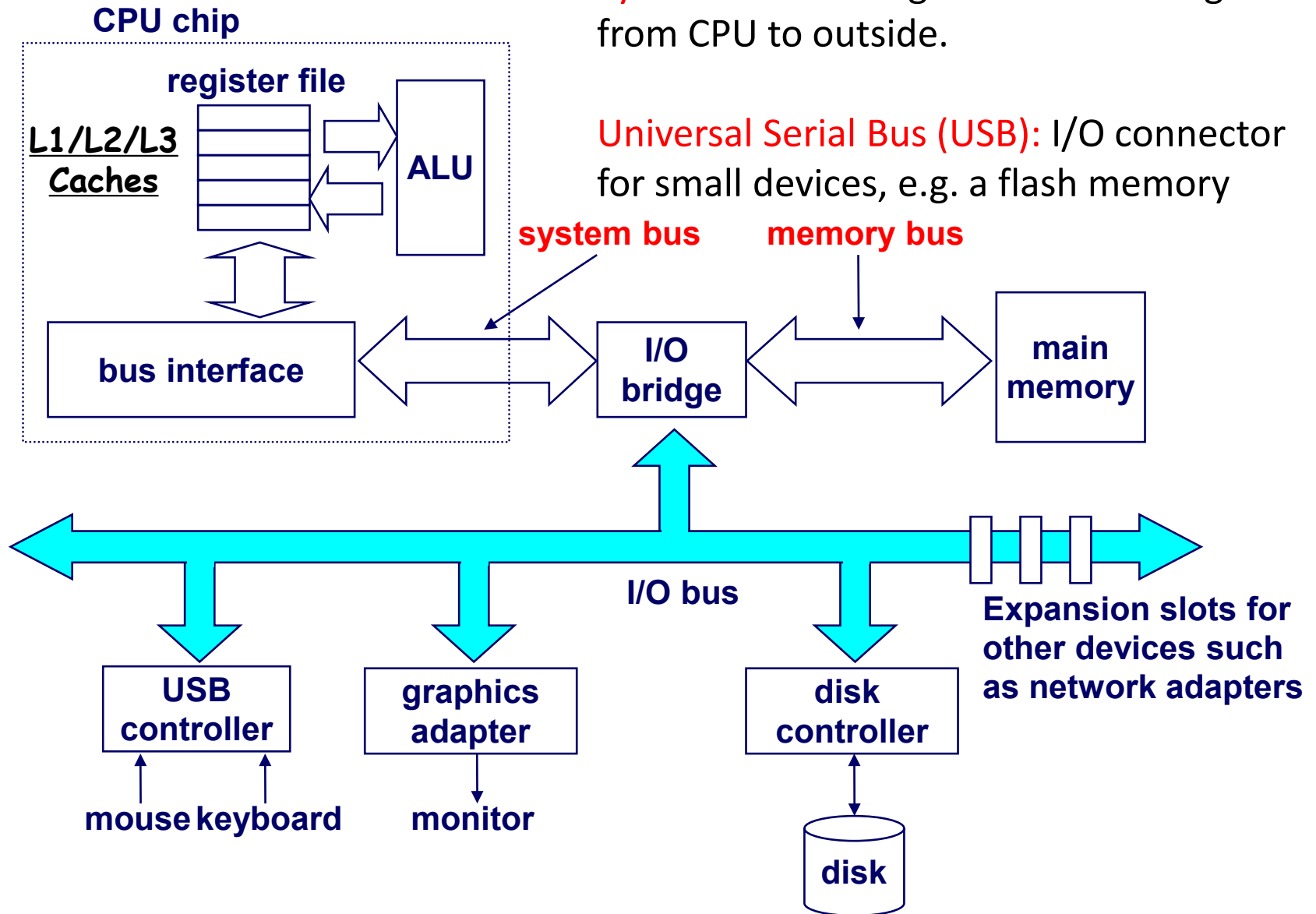
CPU chip (cache is omitted)



# Connecting I/O devices: the I/O Bus

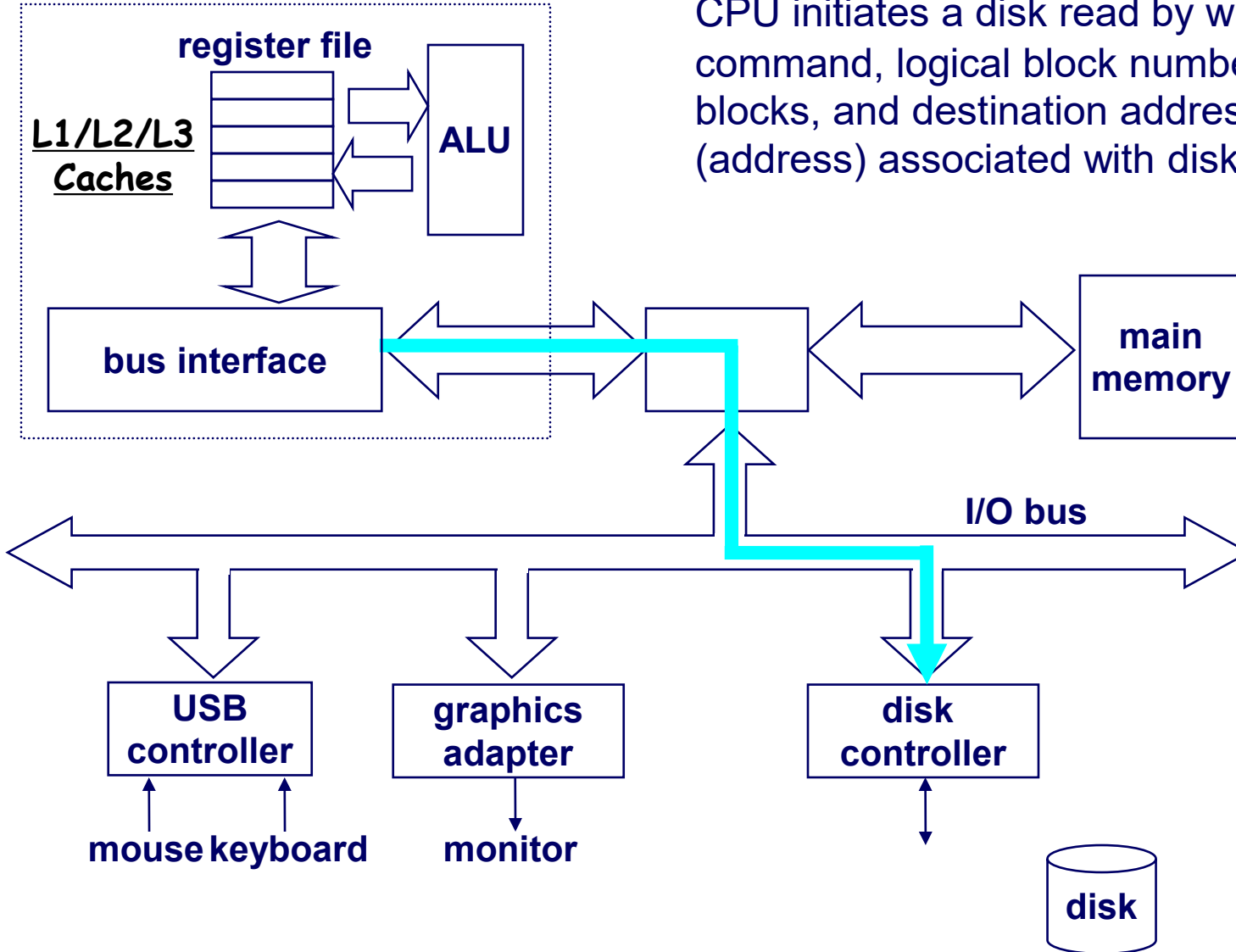
**System bus** is a single bus connecting from CPU to outside.

**Universal Serial Bus (USB):** I/O connector for small devices, e.g. a flash memory



# Reading from disk (1)

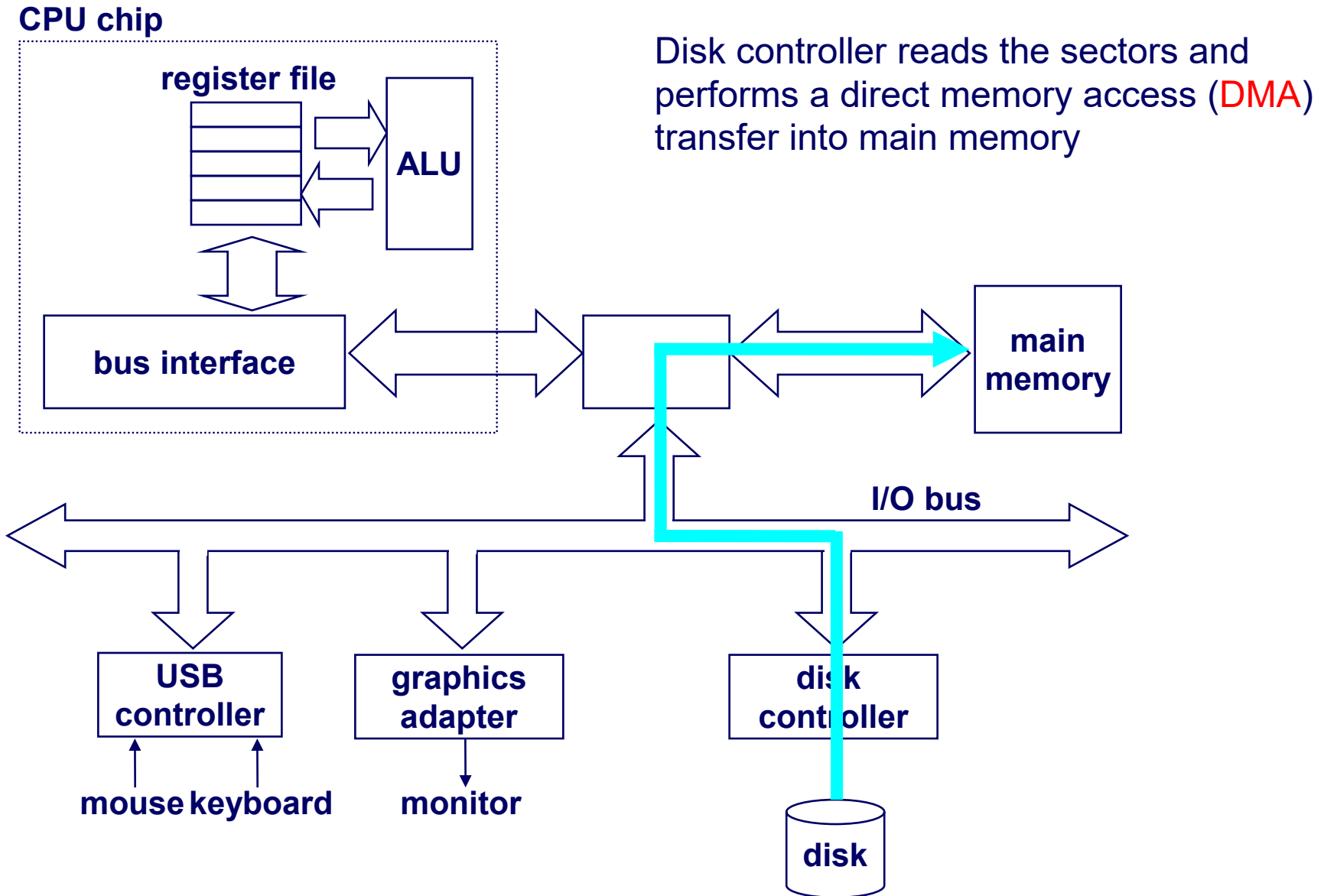
## CPU chip



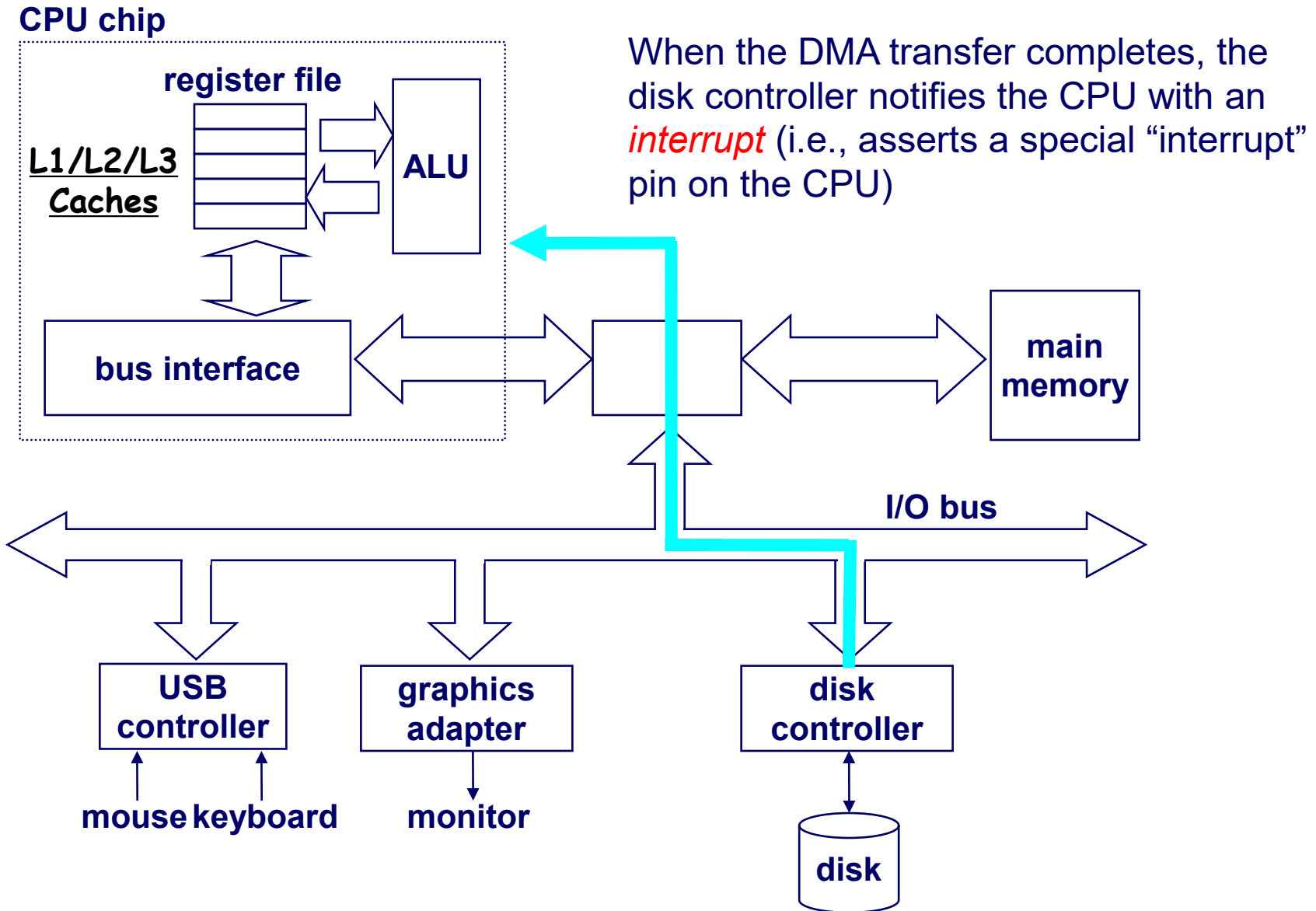
CPU initiates a disk read by writing a `READ` command, logical block number, number of blocks, and destination address to a **port** (address) associated with disk controller



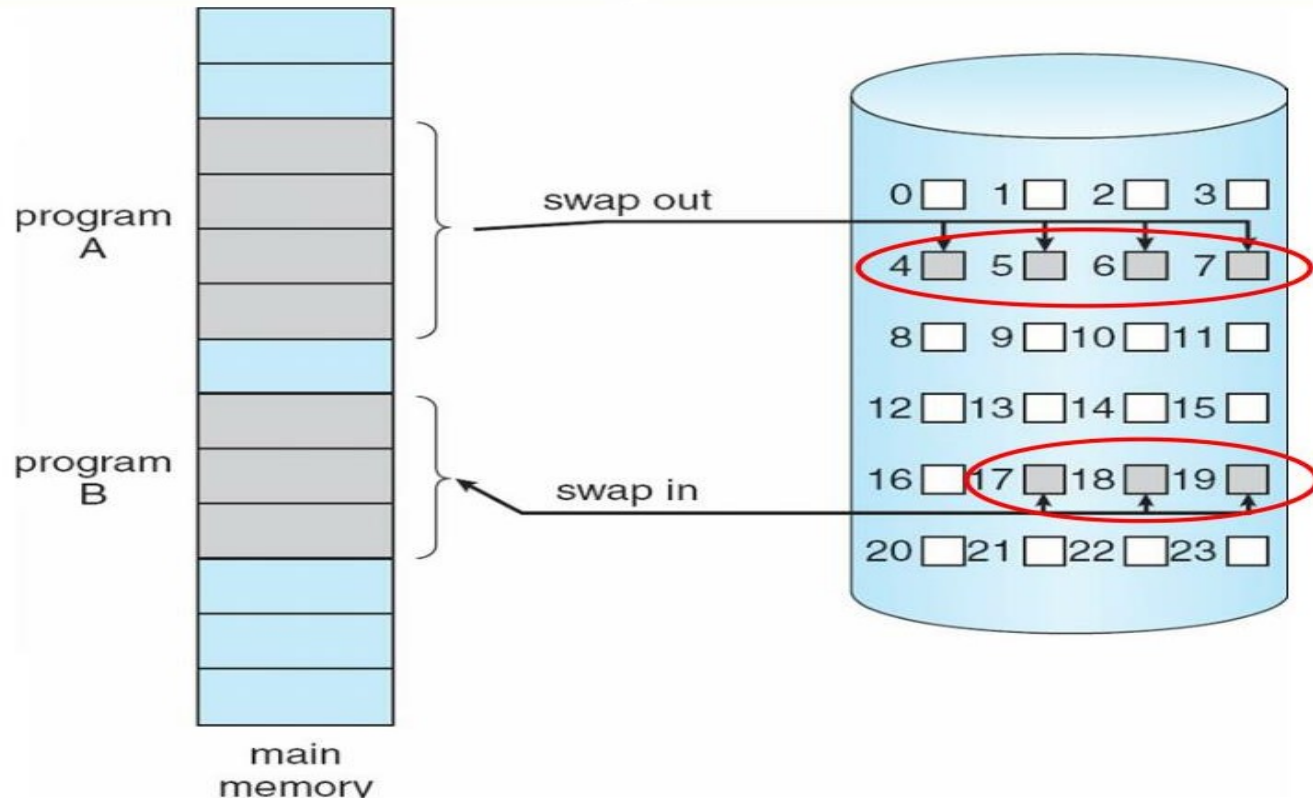
# Reading from disk (2)



# Reading from disk (3)



# Relationship between **physical pages in memory** (logical blocks by OS) and **sectors in disks**



Identification of memory pages: **memory address** + process ID

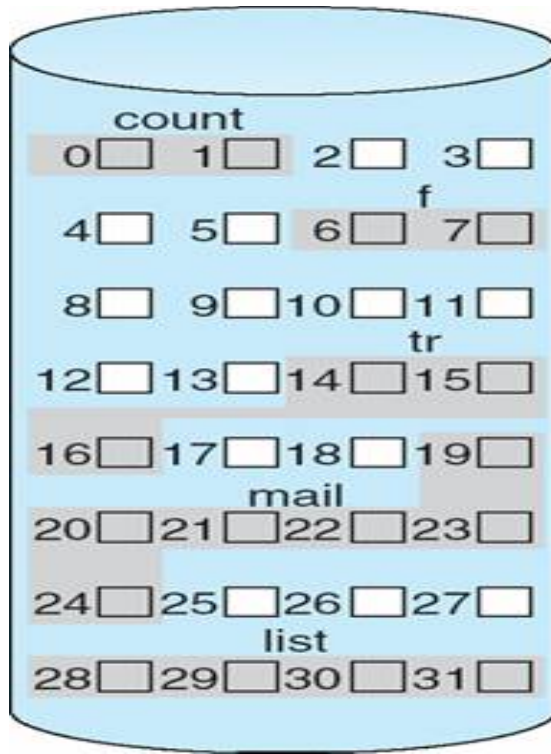
Identification of disk sectors: starting **LBA (logical block Address)** + file ID

The two are bridged by another mapping table in the disk controller

# File System

- A file is the basic data component for all
  - Text, programs, tables, images, and others
  - Non-volatile storage devices (HDD, SSD) are the home
- A file system is an important component in OS
  - It organizes files at logical block level and interact with the disk controller to retrieve and store:
  - (1) how logical files are mapped to physical storage
  - (2) how to help users to access and manipulate files in an efficient way

# File System Directory guides Disk Reads and Writes



file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

## Contiguous Allocation

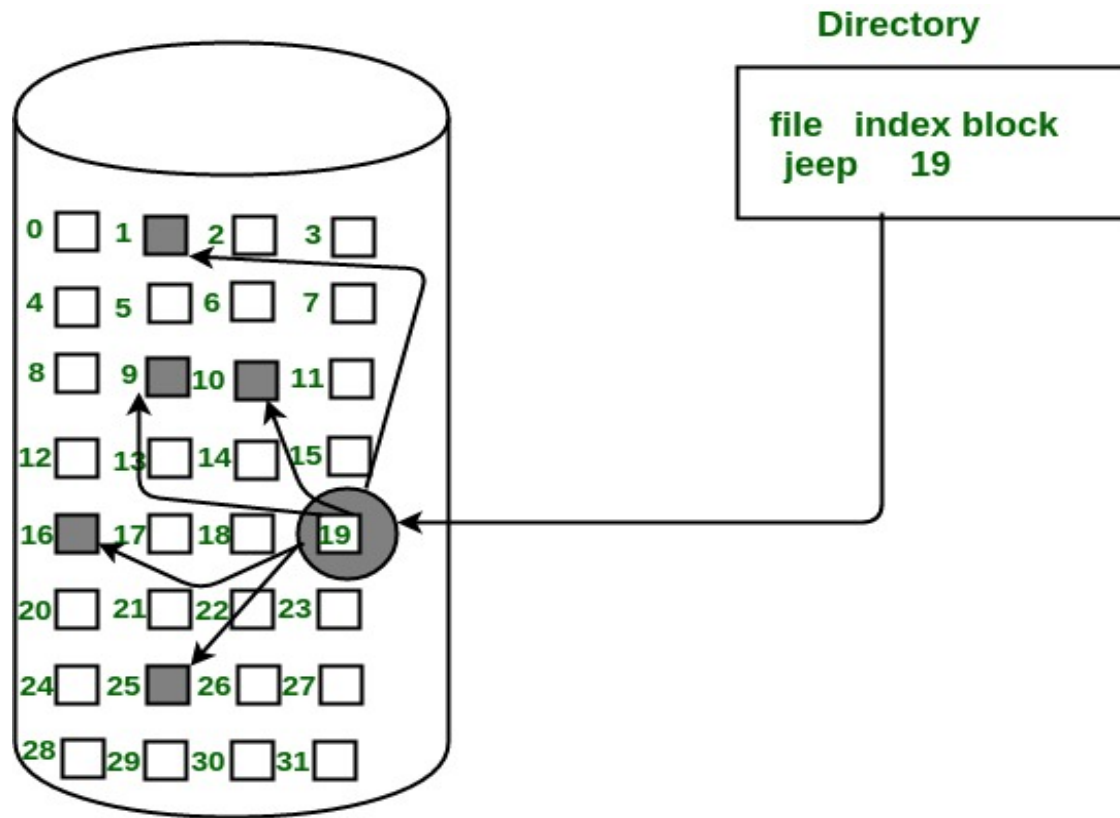
Logical blocks are contiguously allocated in disk **sector by sector** under a **file name** (and a user ID), **a starting sector number** and the **length (# converted blocks)**

**The disk storage allocation is irrelevant to the memory addresses!**

# Pro and Con of Contiguous Allocation

- **Best for sequential read accesses in hard disks**
  - E.g., range queries in databases (a sequentially allocated values with an upper and lower boundary)
- **Bad for random accesses**
  - Logical-physical mapping is done one by one, very slow
- **Space utilization is not high**
  - A frequently updated file generates multiple versions in time stamps, which may not be contiguously allocated
- **A file is hard to grow in a contiguous space**
  - An *extent* is reserved contiguous space in disk, and a file may consist one or multiple extents

Disk Allocation in File System can also be “grouped”



Node 19 is **Index Location**

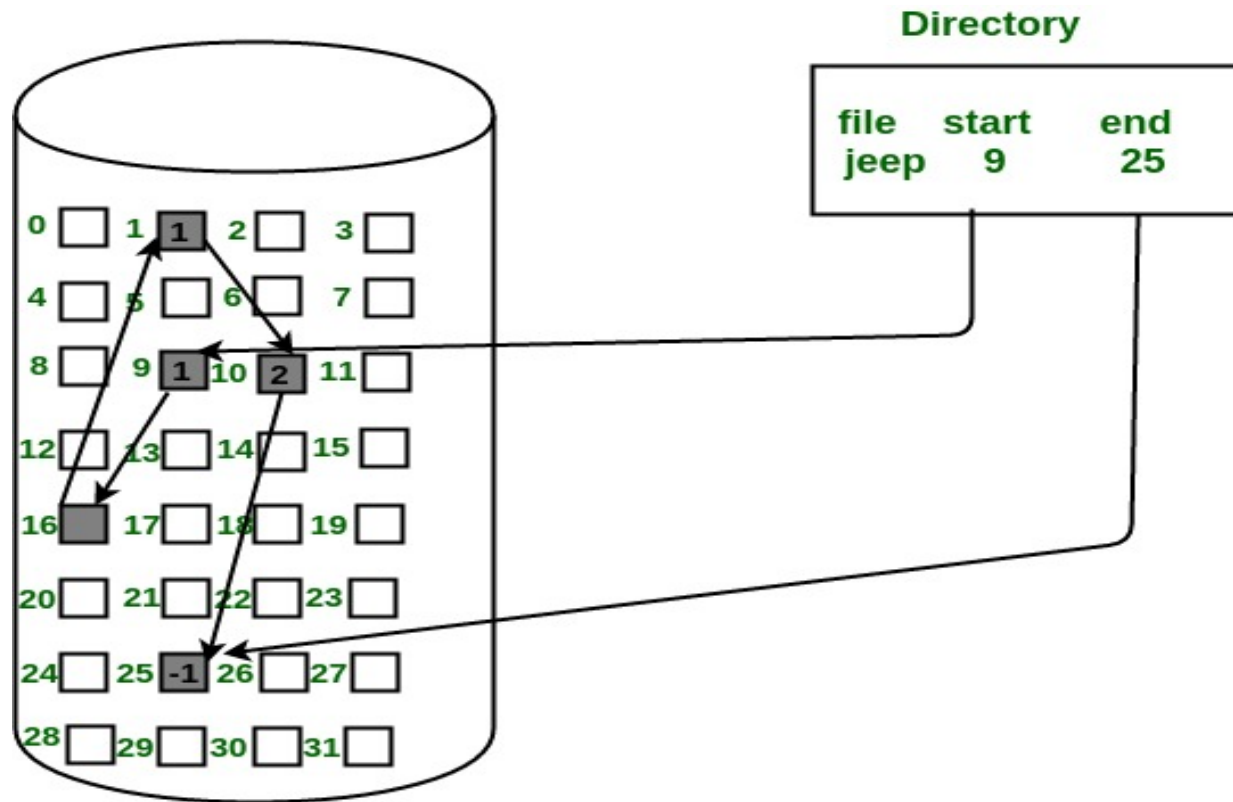
One pointer for all related sectors, additional support is needed in the disk controller.

# Pro and Con of Grouped Allocation

- **Space utilization is high**
  - A file grows and shrinks easily
  - The directory is simple
- **The Index Location can be a bottleneck**
  - It is hard to support a file with many blocks
  - Multiple level index locations are used
  - the management of index location is non-trivial



# Disk Allocation in File System can also be “linked”



A linked-list is formed for all the related sectors, additional support is needed in the disk controller.

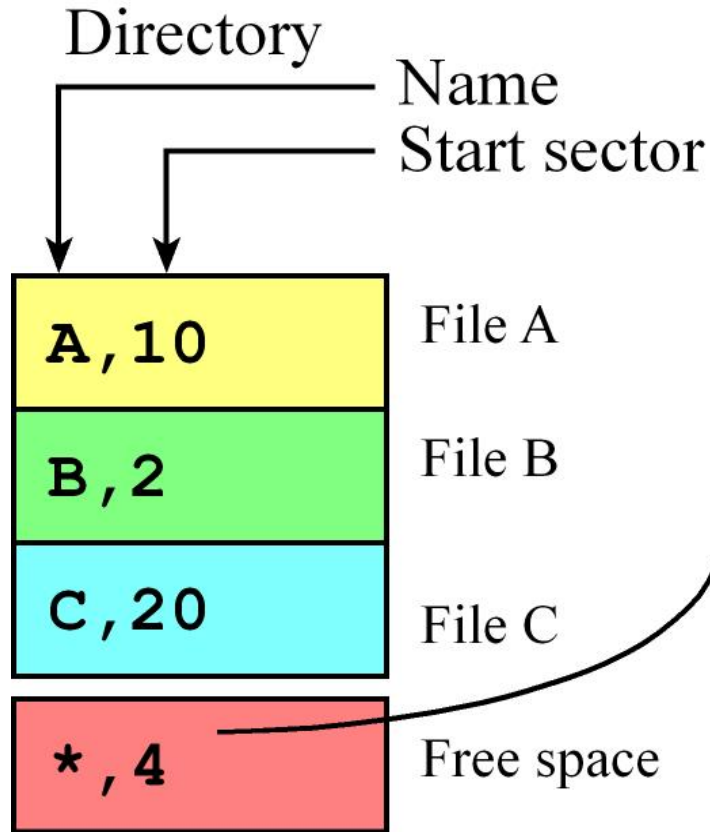
# Pro and Con of Linked Allocation

- Space utilization is high
  - A file grows and shrinks easily
  - The directory is simple
- Accesses to each block are random in disk
  - Seek times cause high latency
  - Any file access must start from the link head
- File Allocation Table (FAT)
  - This implementation creates a table for linked allocation

# File Allocation Table (FAT)

File Allocation Table

Disk

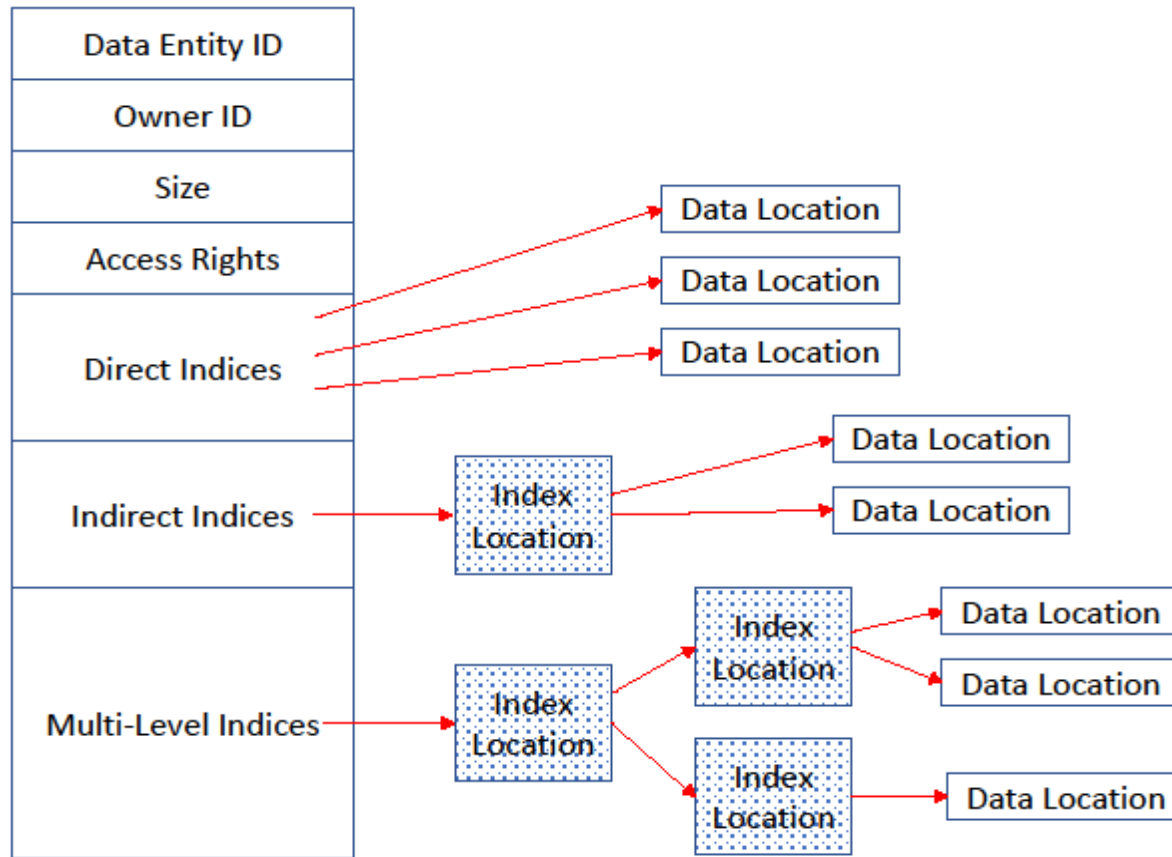


0	x
1	x
2	11
3	12
4	5
5	8
6	0
7	6
8	9
9	15
10	3
11	19
12	0
13	14
14	7
15	16
16	17
17	21
18	0
19	13
20	28
21	22
22	23
23	24
24	25
25	29
26	18
27	26
28	27
29	30
30	31
31	0

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	
29	
30	
31	

Unused LBNs are filled in the linked free space

# Disk Allocation in File System is in “inode”



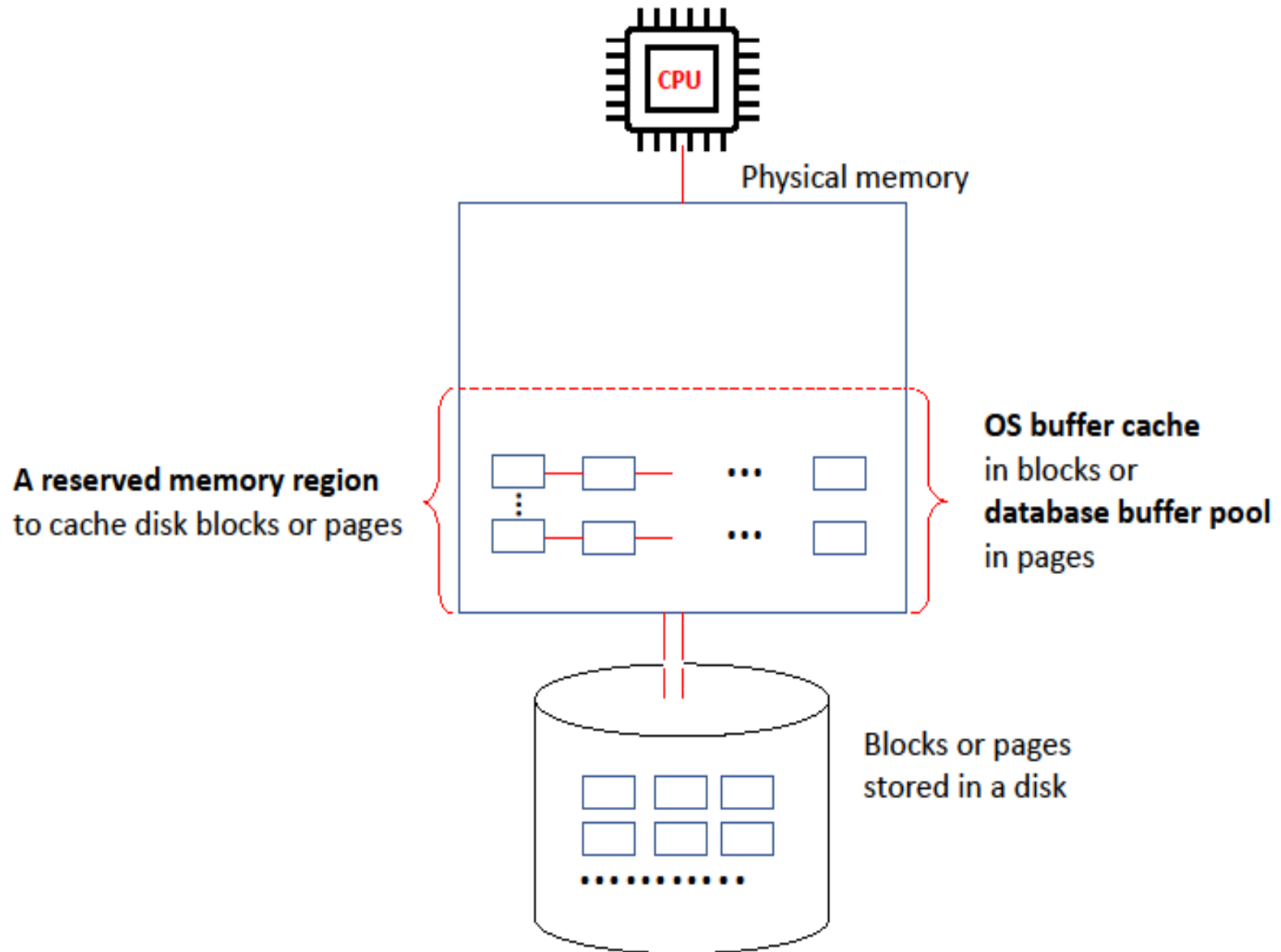
An **Index Node** or Information Node (inode) collects all the related information items for a data entity (multiple files) based on ownership, access rights ...

Inodes are stored in a **special region of disk**, and loaded (cached) to memory when they are used

# Buffer Cache

- **A special reserved region in memory**
  - To cache copies of disk blocks, such as files, inodes, ....
  - Serving as write-back buffer (modified data)
  - Managed by OS (Kernel block size: 1 KB in Linux)
- **Replacement is used all the time**
  - LRU, LIRS, Clock, and Clock-pro
- **Prefetching**
  - By prediction, after reading part of a file, OS fetch can cache the rest of its blocks before asking

# Where is Buffer Cache?



## Differences: Demand Paging, CPU Caching and Buffer Caching

- **Demanding Paging** (Lecture of Virtual Memory)
  - OS/MMU create a page table to each running process, TLB for MRU PT entries (Lifecycle: program execution)
- **CPU Caching** (Lecture of Hardware Caches)
  - During the execution, MRU pages are stored in on-chip caches, or in memory, LRU pages are evicted (Lifecycle: program execution)
- **Buffer caching**
  - A memory cache for data copies between Filesystem/DB and disks (Lifecycle: buffer cache space dependent)

# The Merits and Limits of HDD are Clear

## Why and Why Not SSD?