CSE 3421
Introduction to Computer Architecture

Chapter 6:
Storage and Other I/O Topics

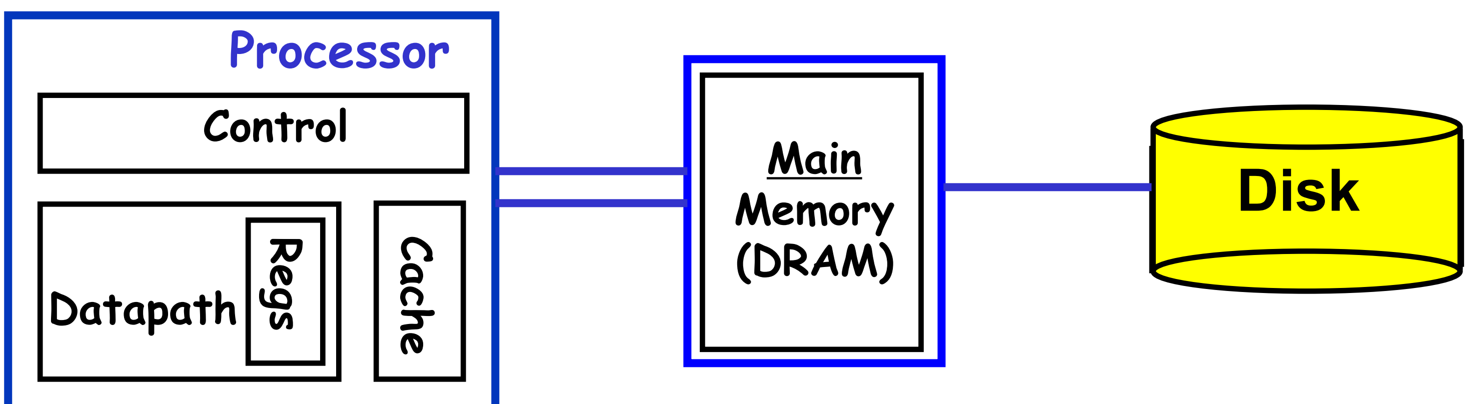
Xiaodong Zhang

Some Slides from CMU are used.

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

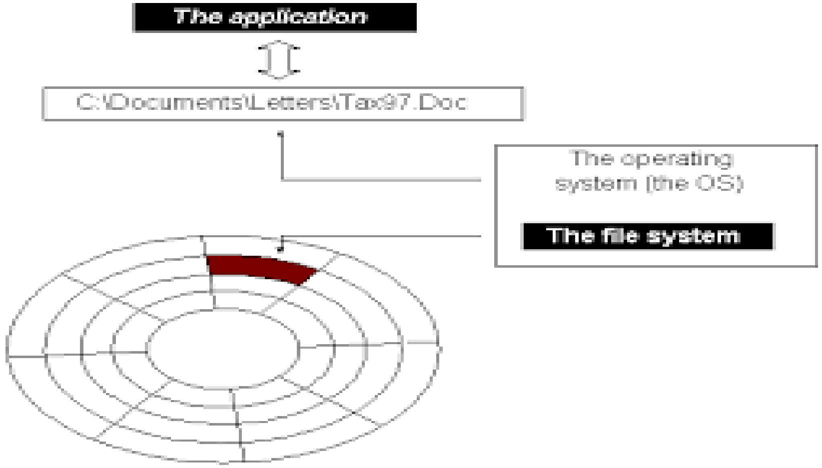
Disk Storage is the Permanent Home of Data

- 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

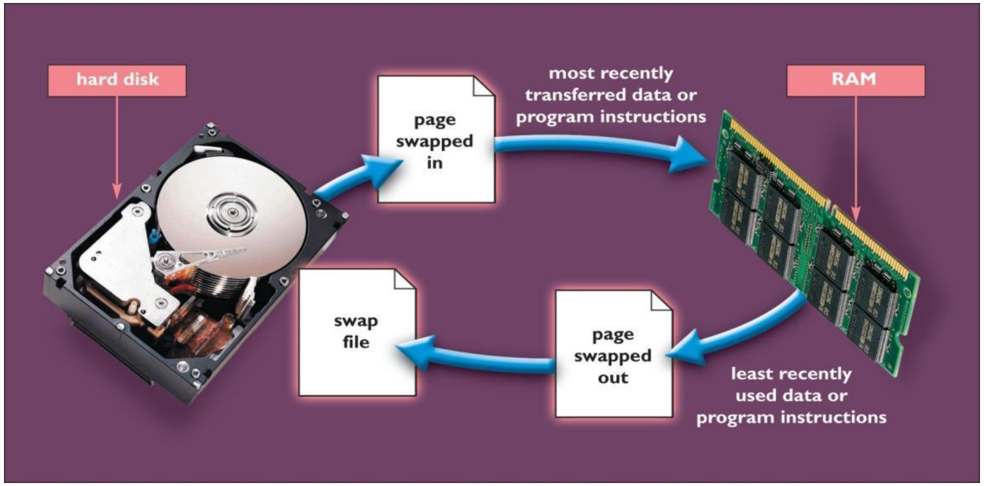


A disk is partitioned into three regions

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



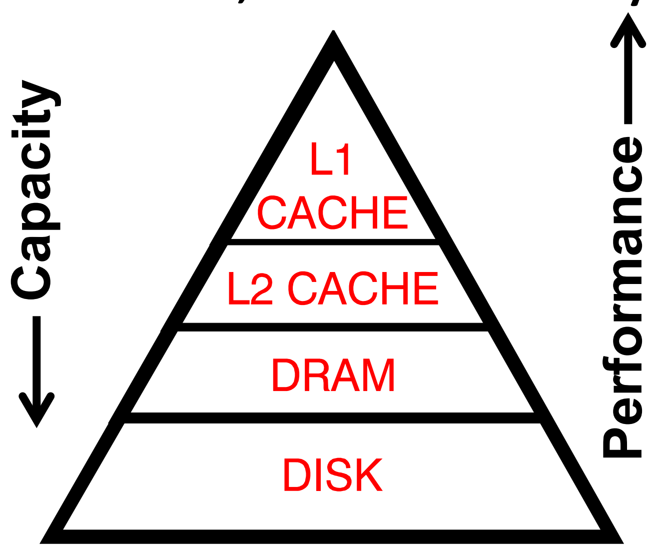
- OS Swapping Space (e.g., running your programs)



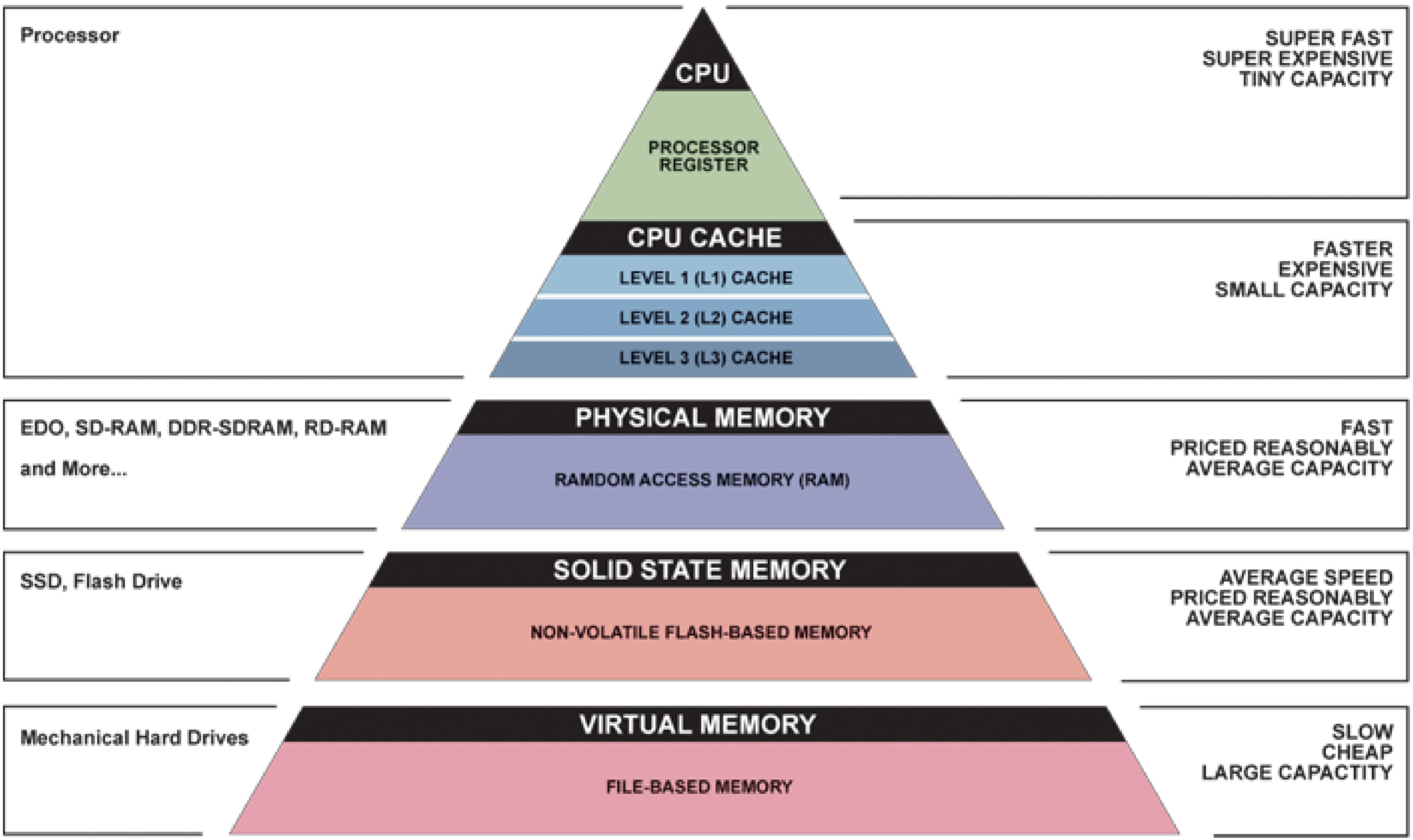
- Files for System Administration (OS kernels)

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



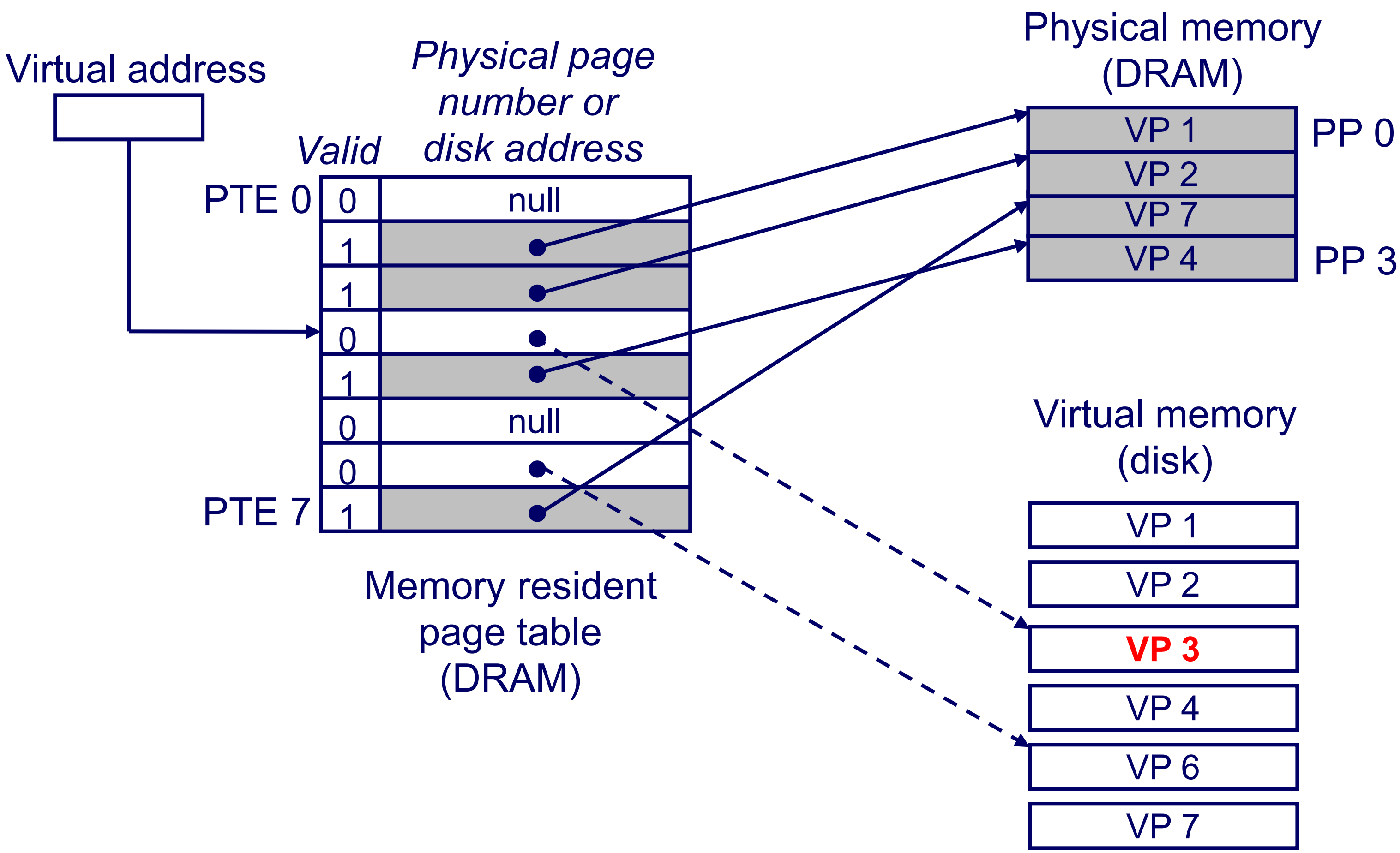
An architectural view of memory hierarchy



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?

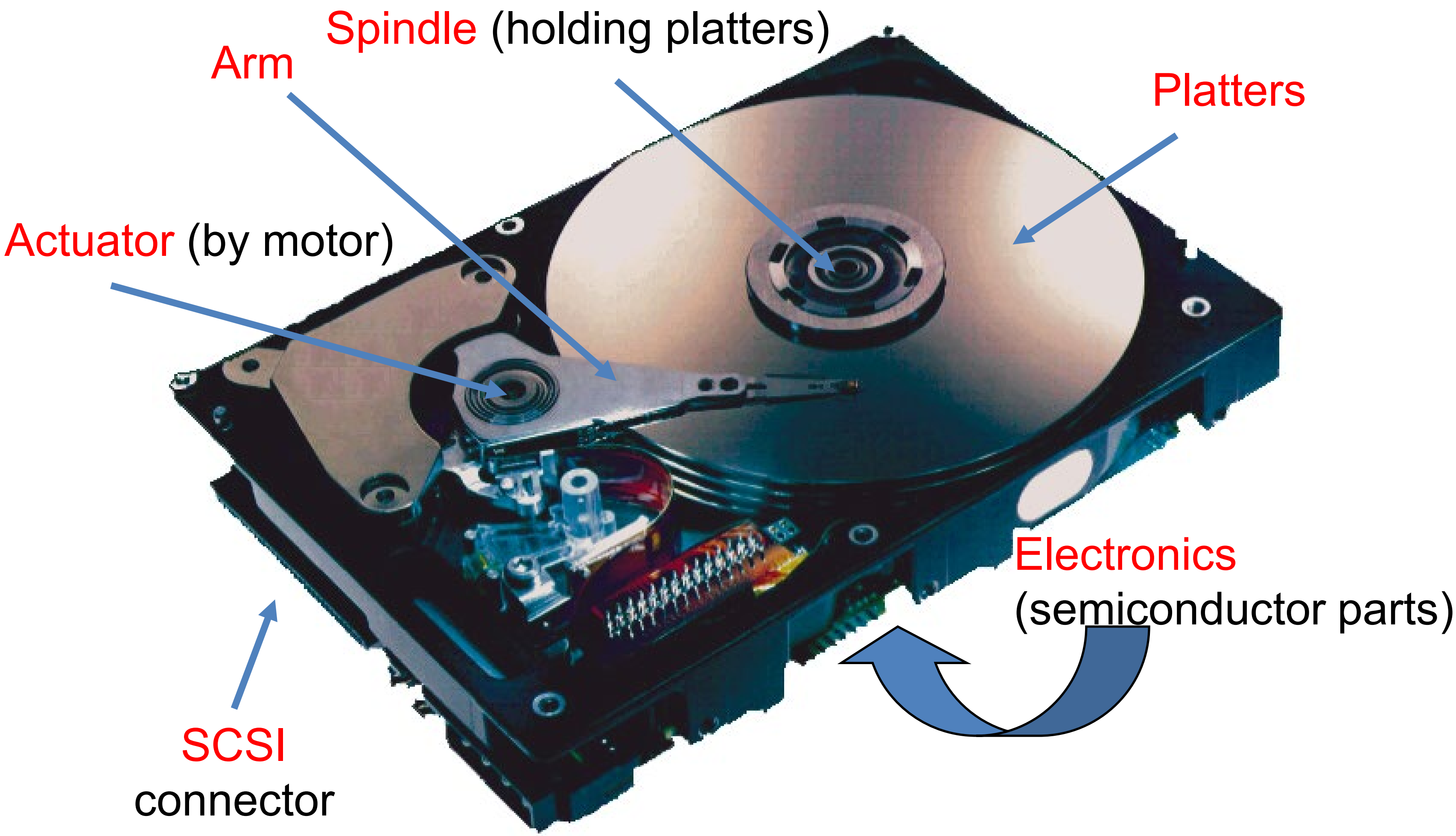


Image courtesy of Seagate Technology

A Record Player Reads Music Data from a Record

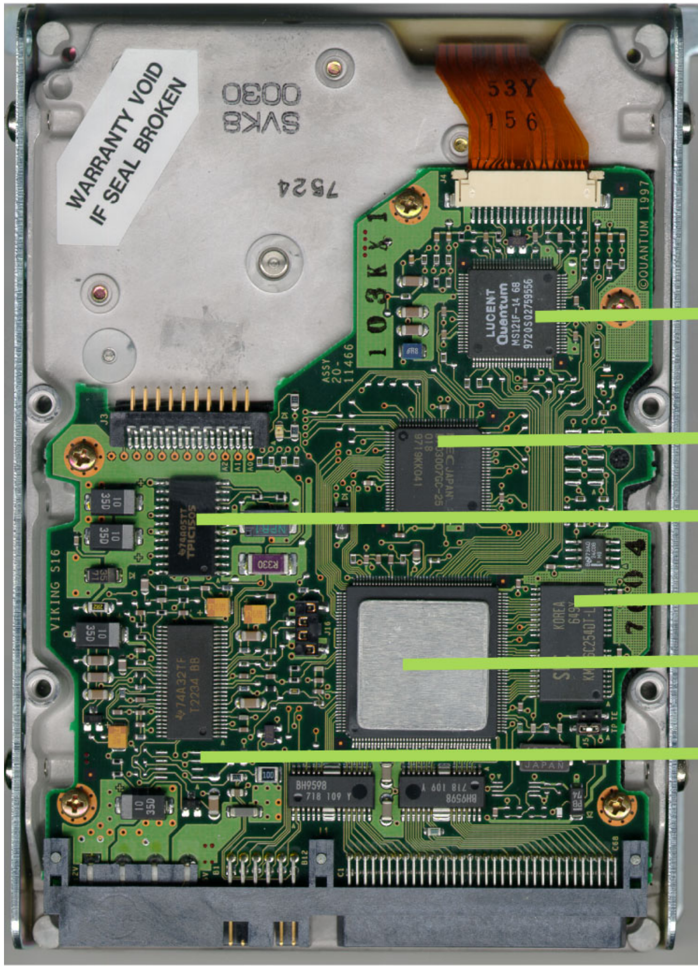
By Thomas Edison

In 1877, called

Phonograph

Disk Electronics

Quantum Viking (circa 1997)



6 Chips

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

R/W Channel

uProcessor
32-bit, 25 MHz

Power Array

2 MB DRAM

Control ASIC
SCSI, servo, ECC

Motor/Spindle

- Connect to disk
- Control processor
- Cache memory
- Control ASIC
- Connect to motor

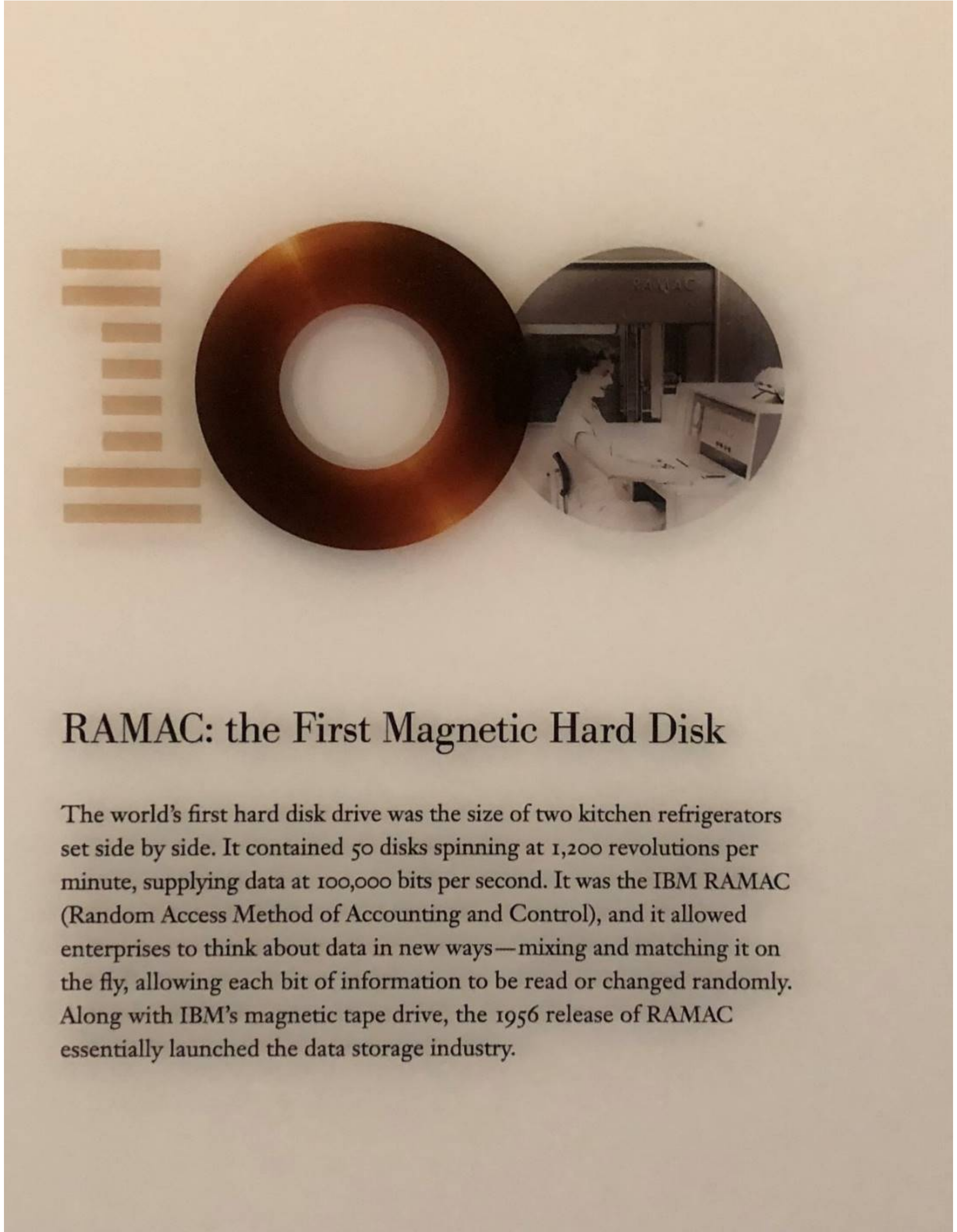
IBM 100: 100 Innovations in 100 years (1911-2011)

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

Several of them are related to this class

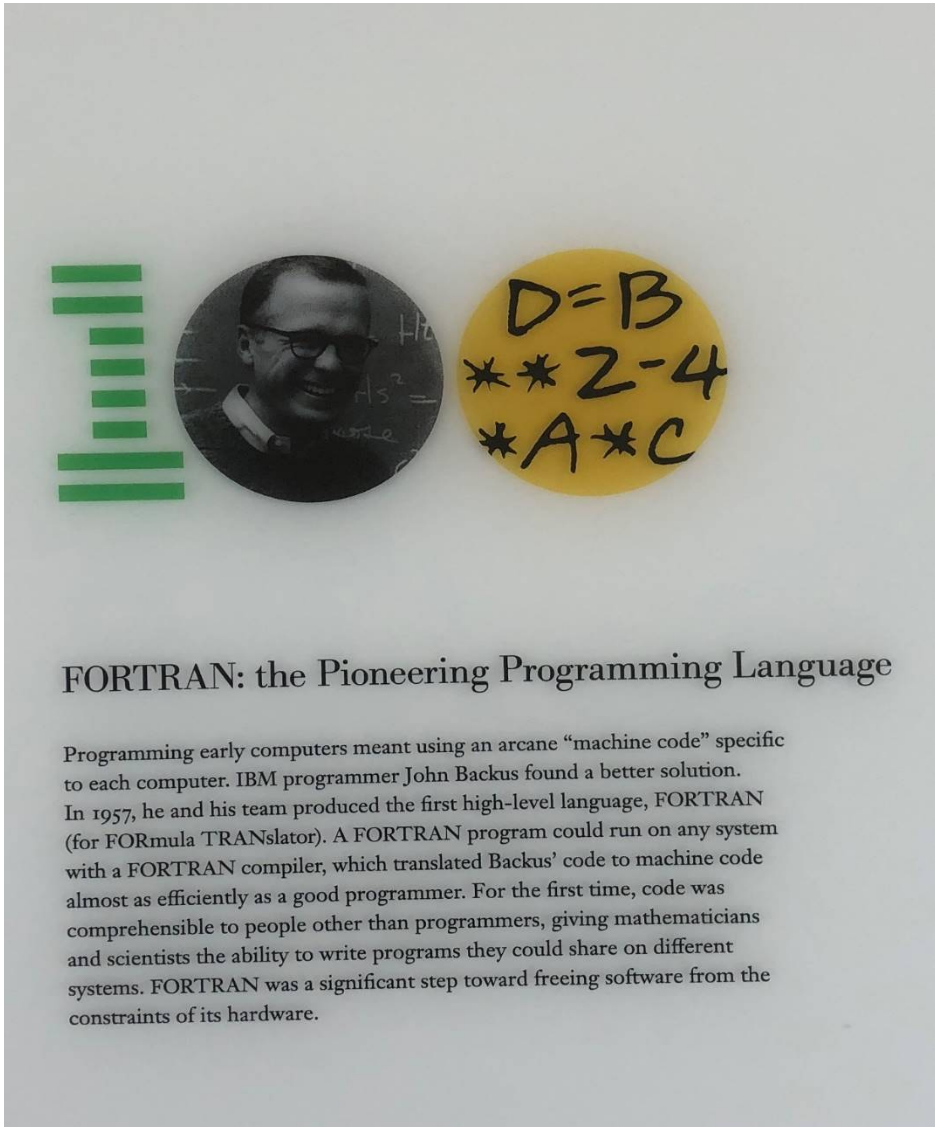
Seminal Contributions in Computing from IBM

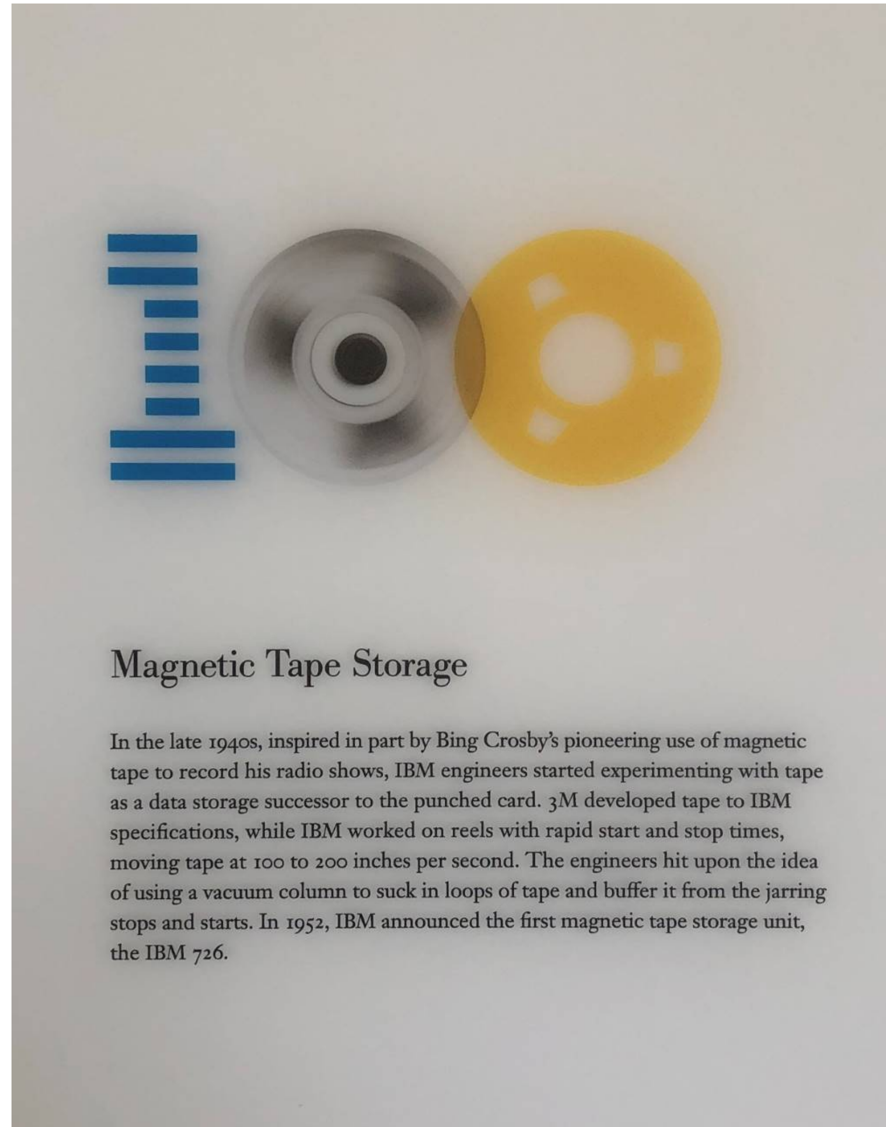
- 1956: Hard Disks
- 5 MBytes
- 1,200 RPM



Seminal Contributions in Computing from IBM

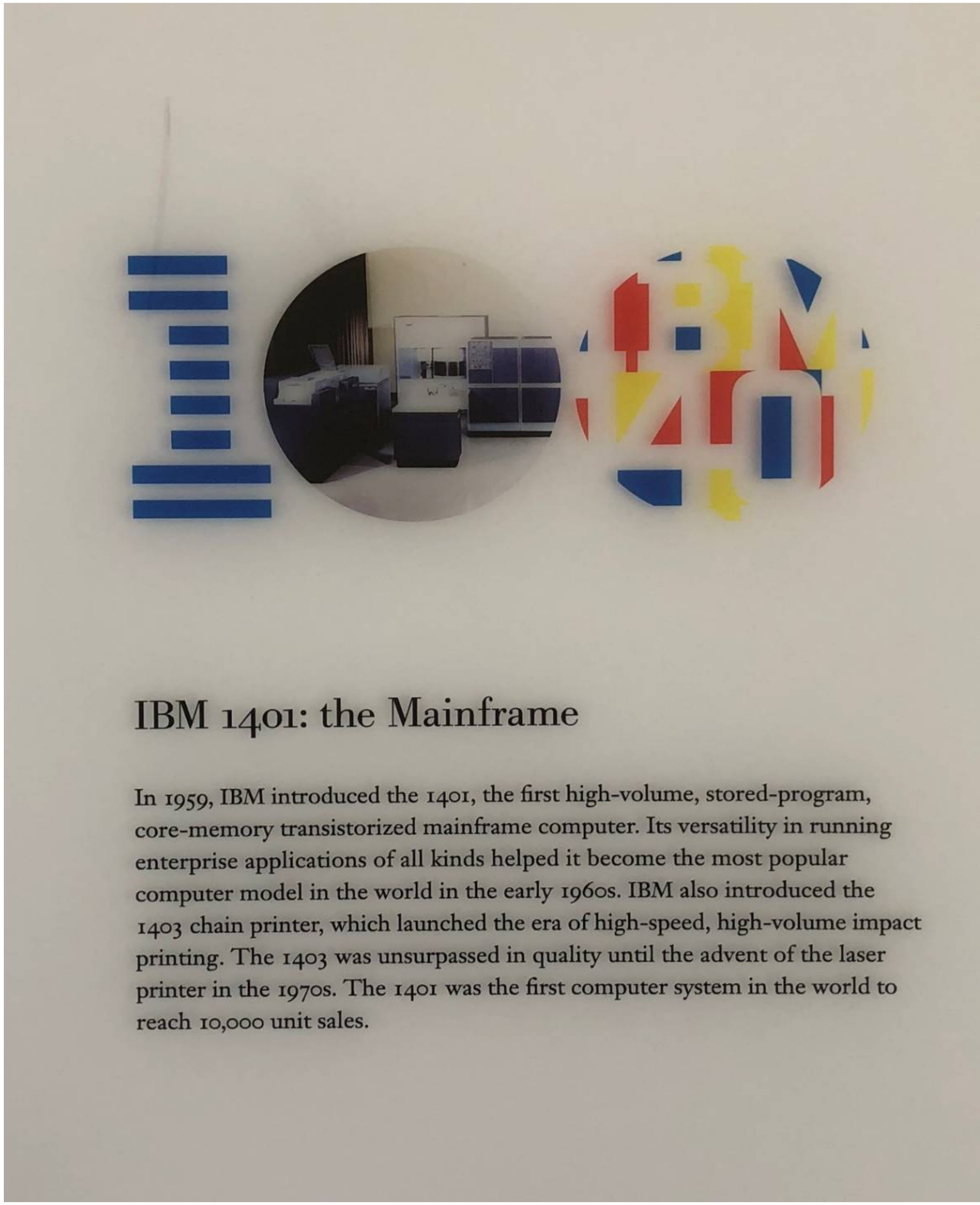
- 1957: Fortran and Magnetic Tapes





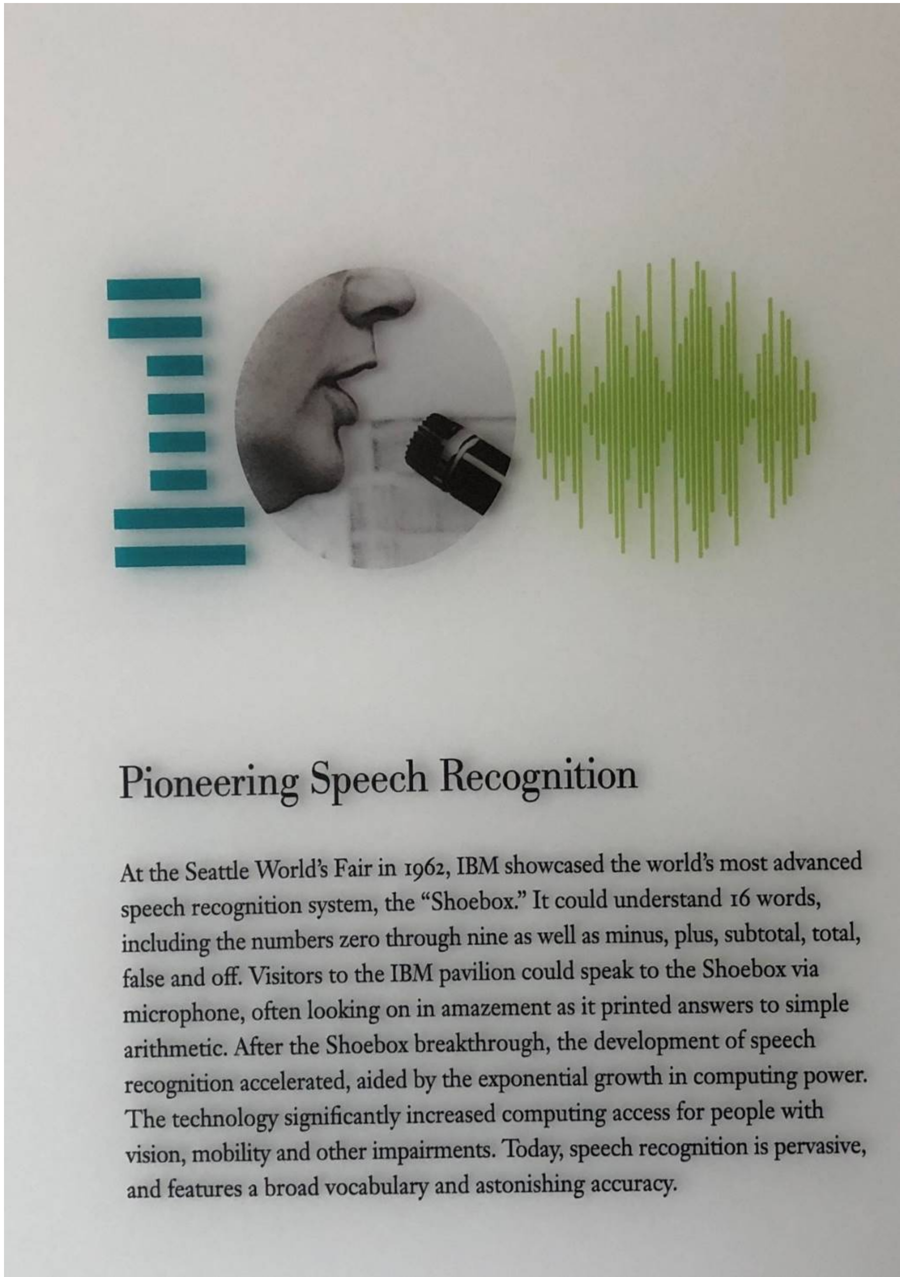
Seminal Contributions in Computing from IBM

- 1959: IBM Mainframe



Seminal Contributions in Computing from IBM

- 1962: Speech Recognition

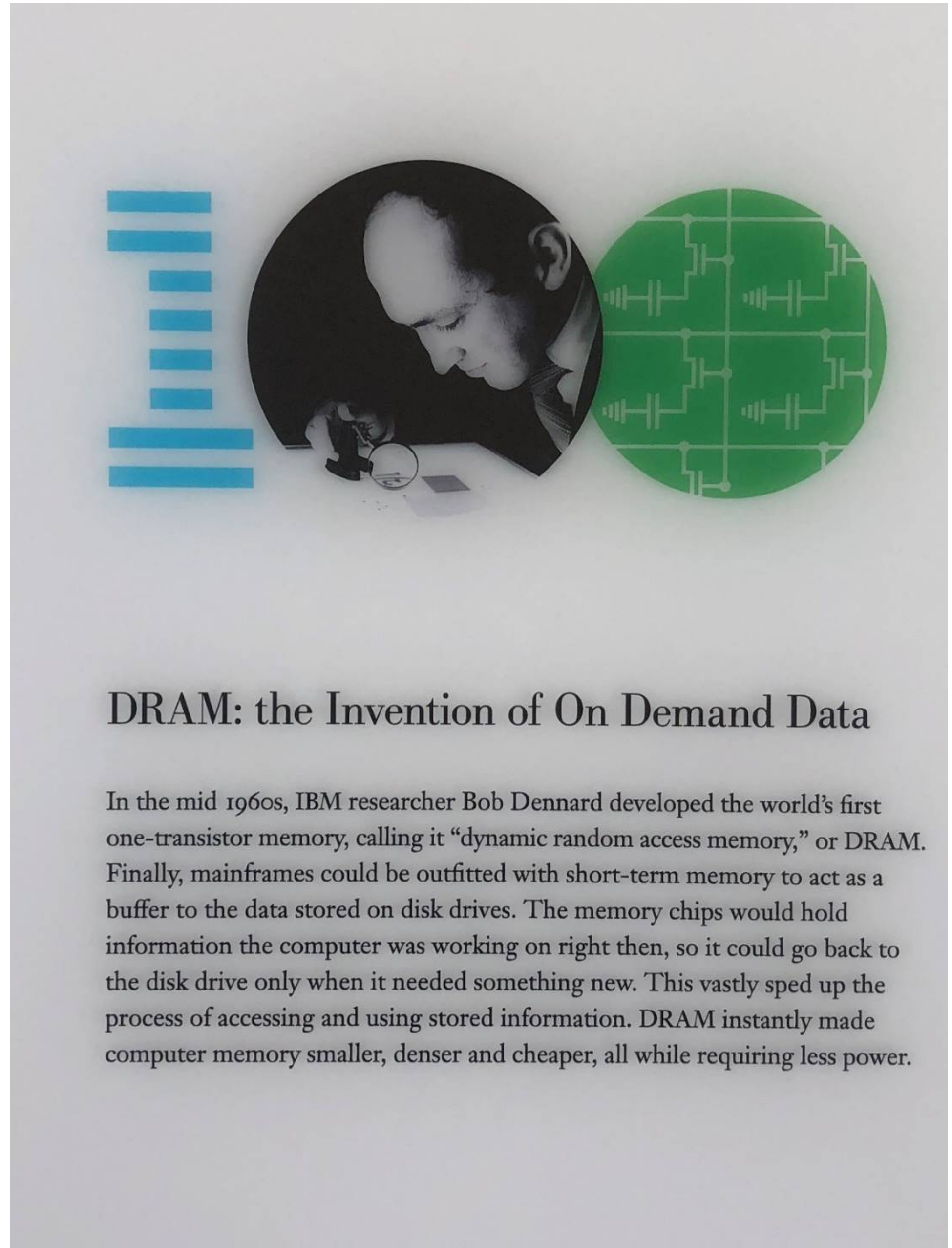


Seminal Contributions in Computing from IBM

- 1964: IBM 360

Seminal Contributions in Computing from IBM

- 1968: DRAM Memory



Seminal Contributions in Computing from IBM

- 1971: Virtual Memory

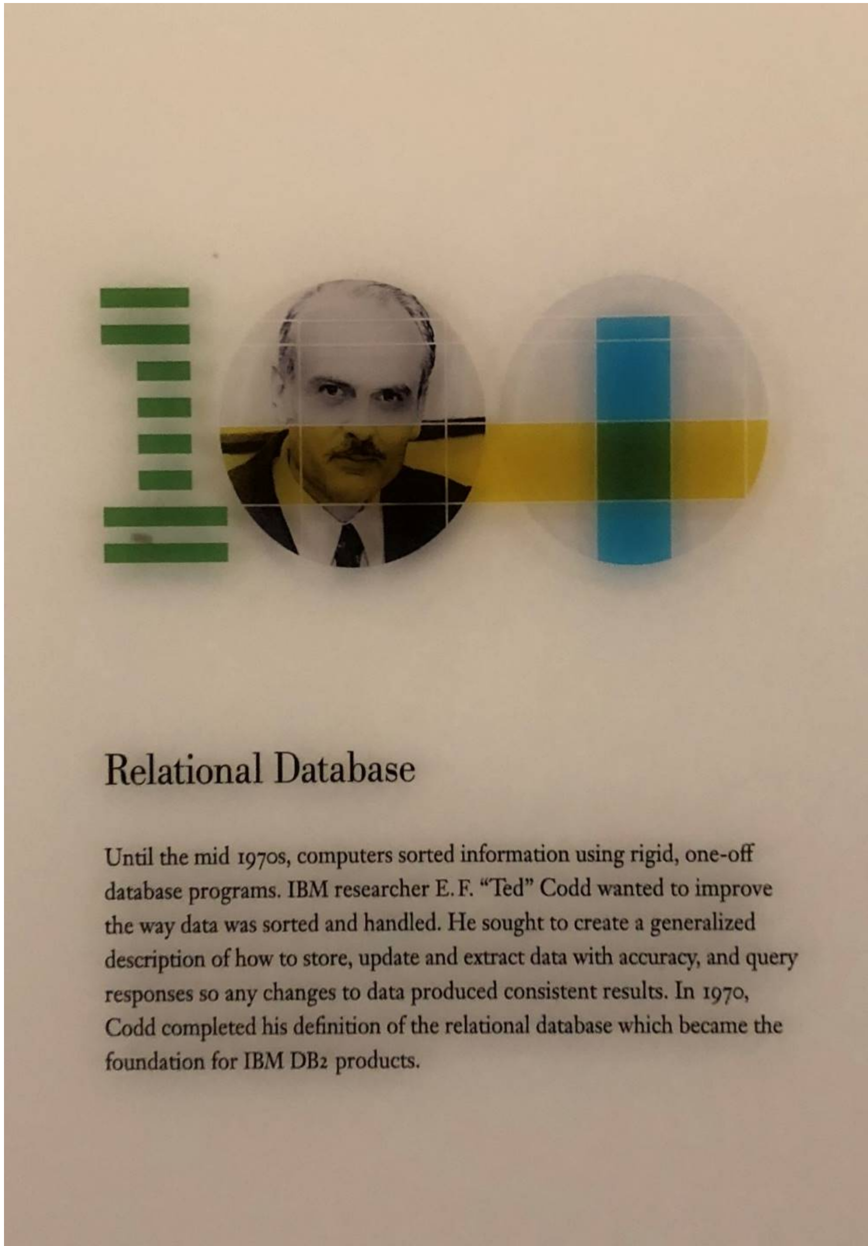
in IBM OS/360

Seminal Contributions in Computing from IBM

- 1971: Floppy Disks

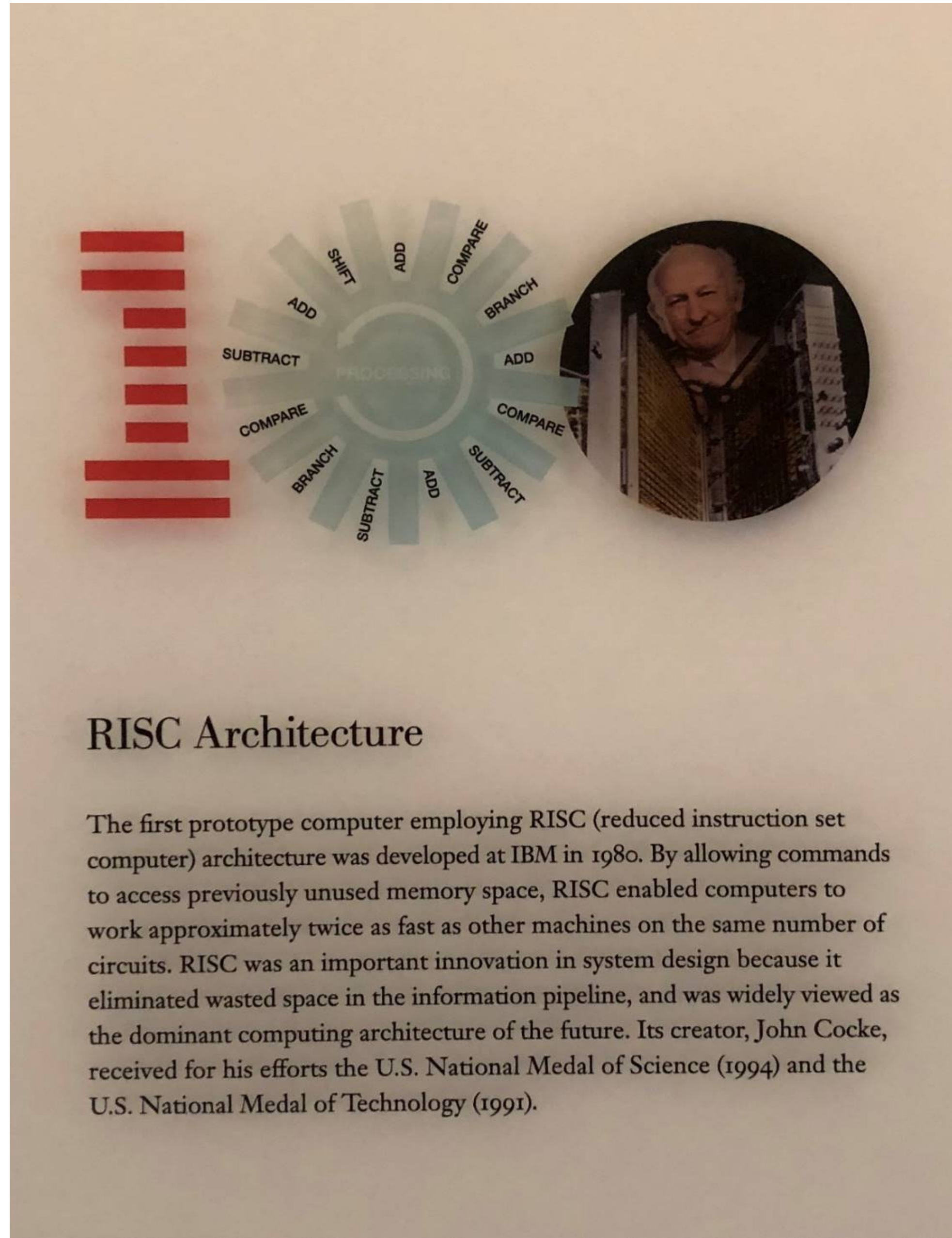
Seminal Contributions in Computing from IBM

- 1971: Relational Database



Seminal Contributions in Computing from IBM

- 1980: RISC Processor

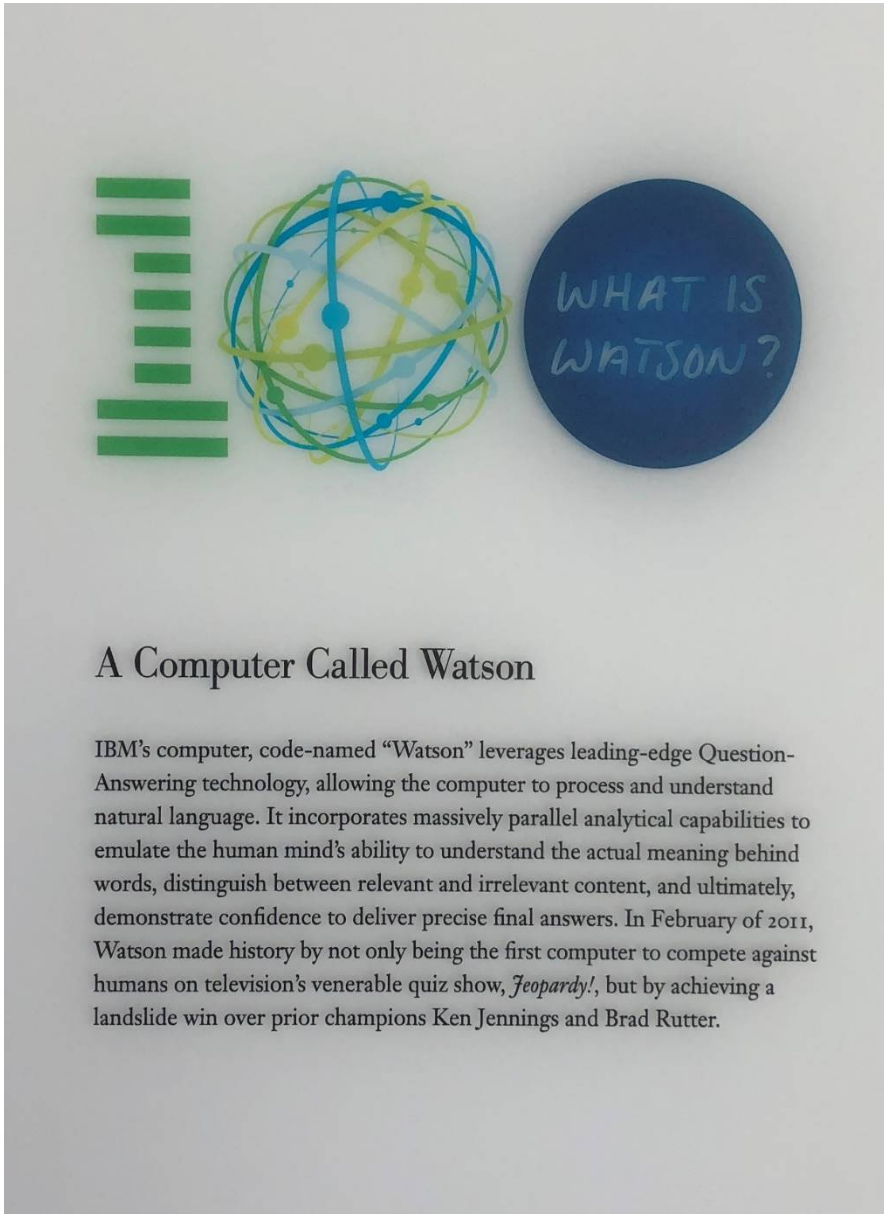


Seminal Contributions in Computing from IBM

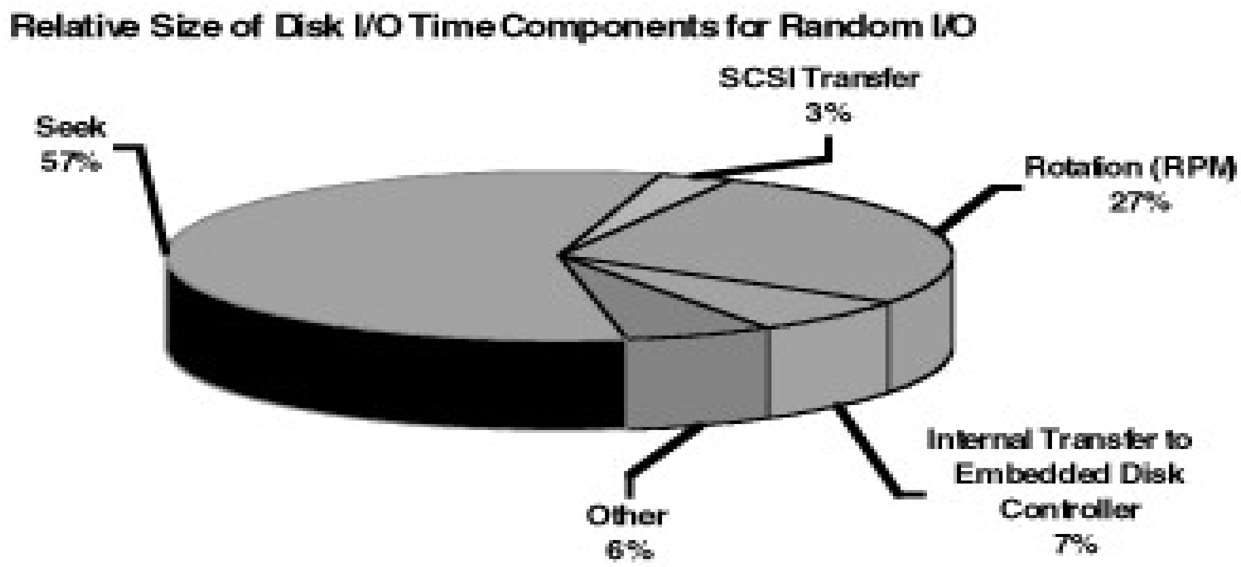
- 1997: Deep Blue

Seminal Contributions in Computing from IBM

- 2011: Watson AI Machine



High power/latency from mechanical operations



- **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%

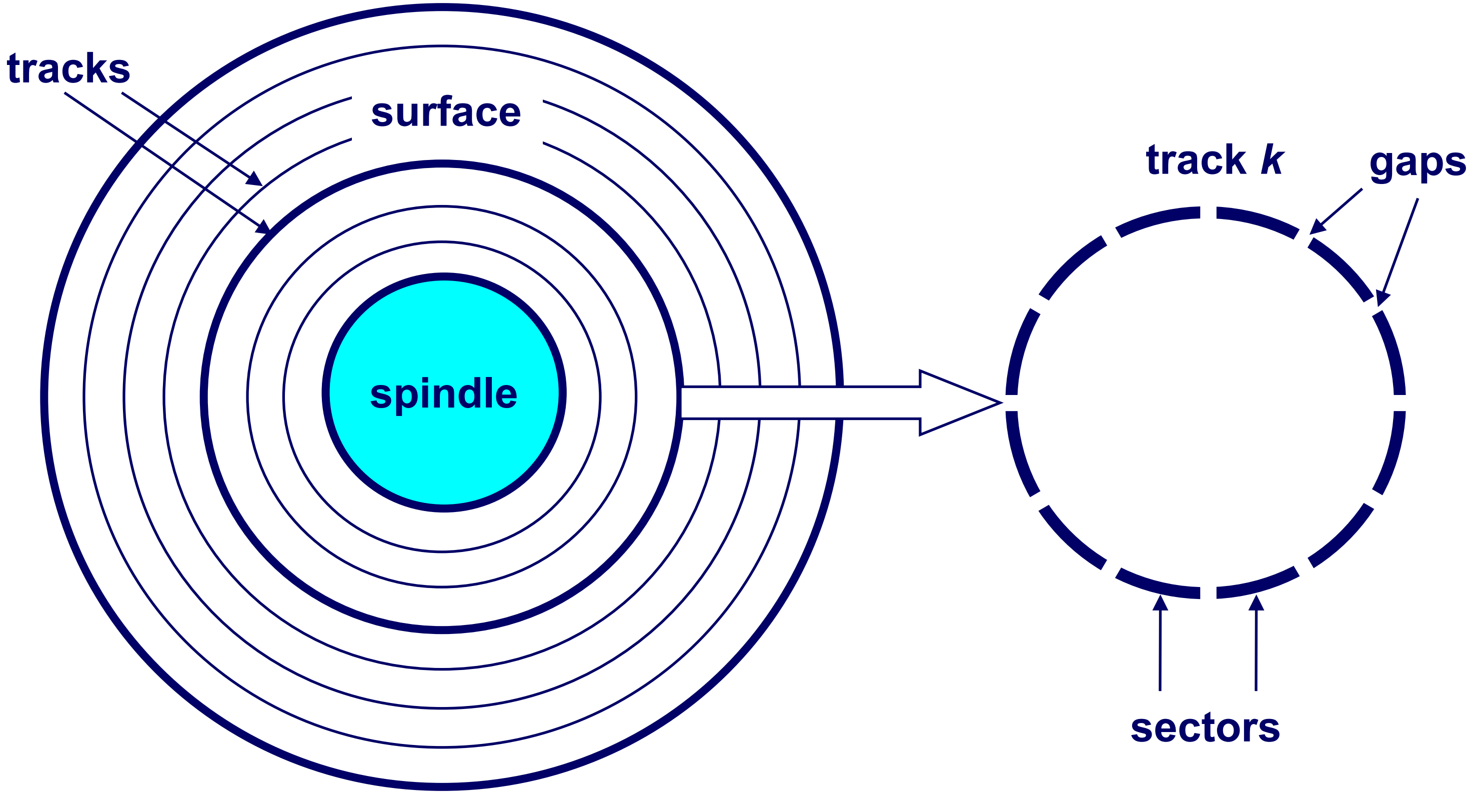
Source: Configuration and Capacity Planning for Solaris Servers, Sun Microsystems

Disk "Geometry"

Disks contain platters, each with two surfaces

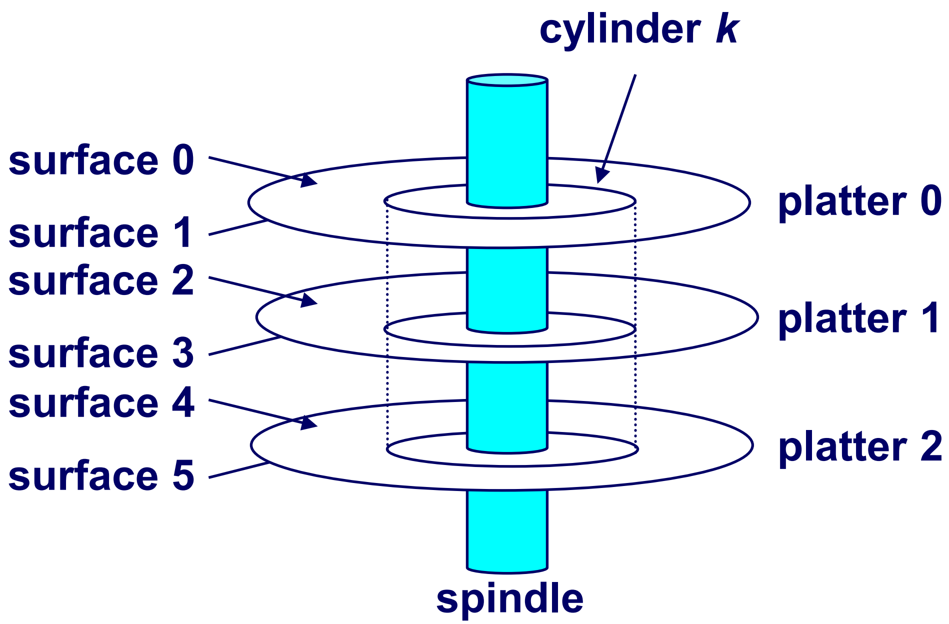
Each surface organized in concentric rings called tracks

Each track consists of sectors separated by gaps



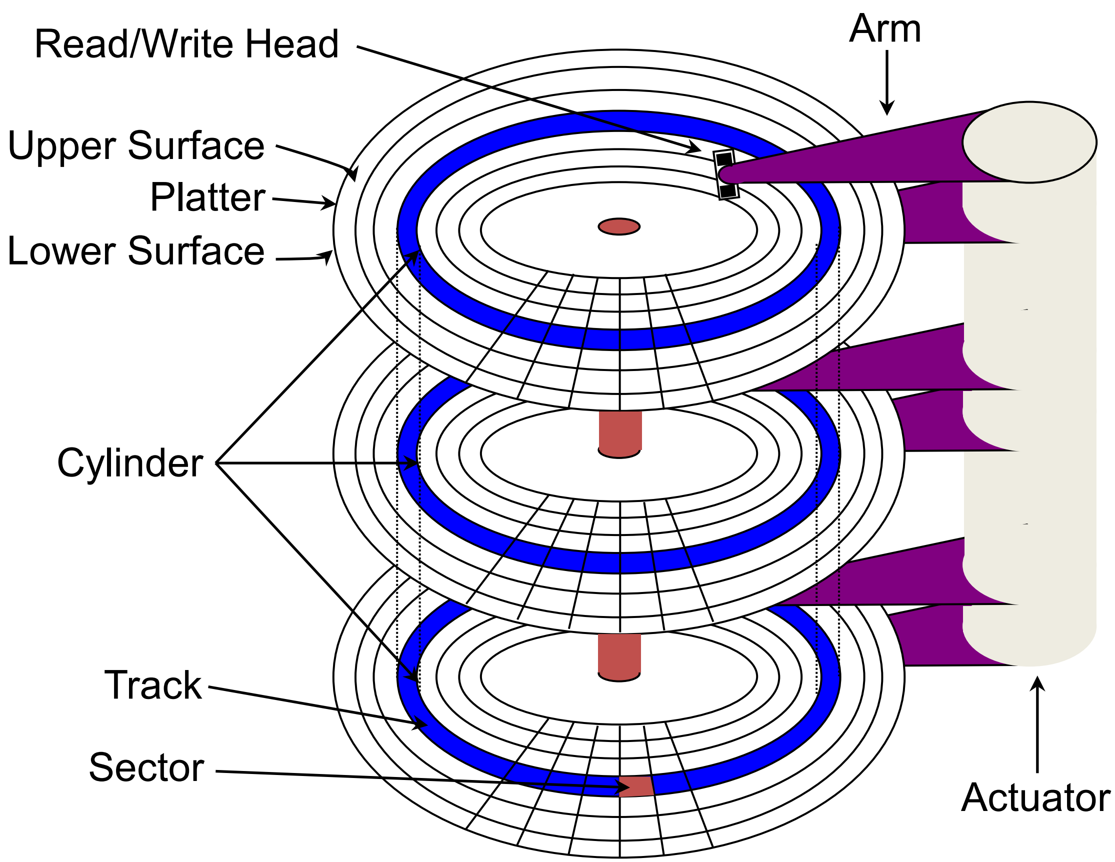
Disk Geometry (Muliple-Platter View)

Aligned tracks form a cylinder



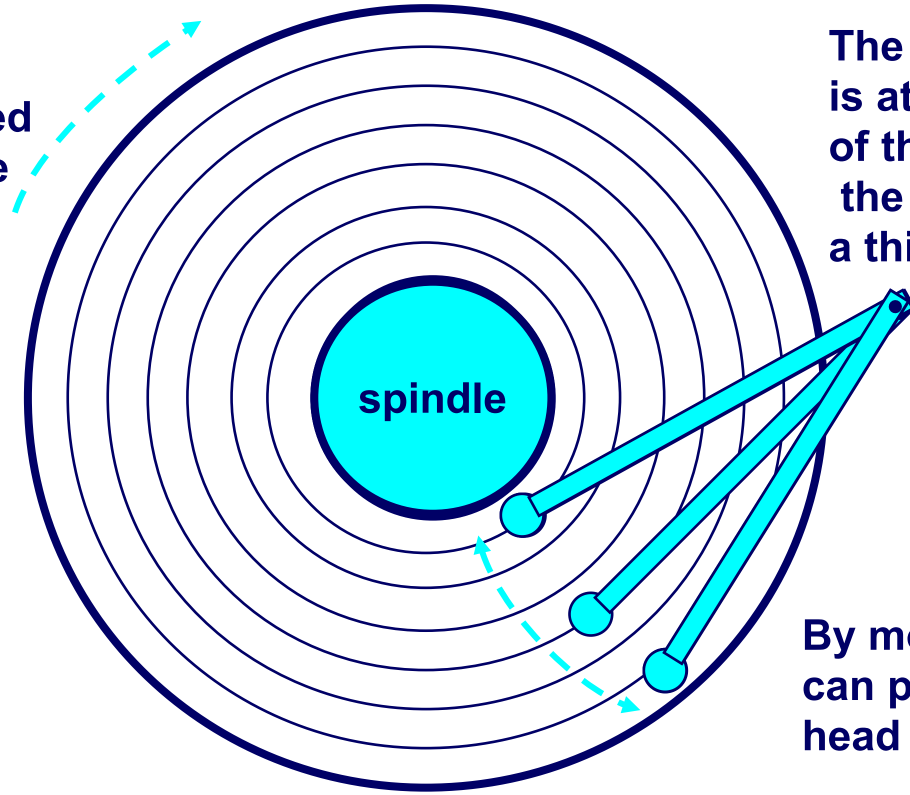
A cylinder consists all the tracks of equal diameter, vertically it forms a “cylinder” for parallel data accessing

Disk Structure



Disk Operation (Single-Platter View)

The disk surface spins at a fixed rotational rate

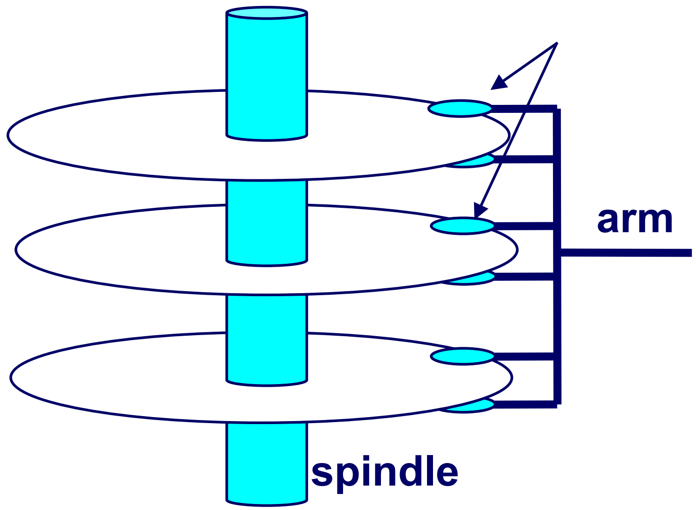


The read/write *head* is attached to the end of the *arm* and flies over the disk surface on a thin cushion of air

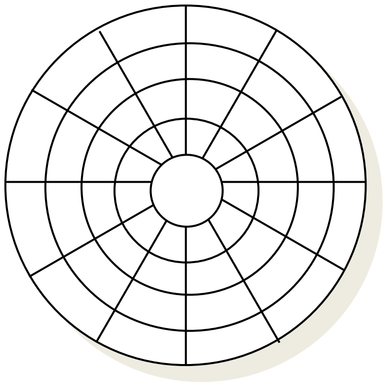
By moving radially, the arm can position the read/write head over any track

Disk Operation (Multi-Platter View)

read/write heads move in unison (together)
from cylinder to cylinder for high throughput



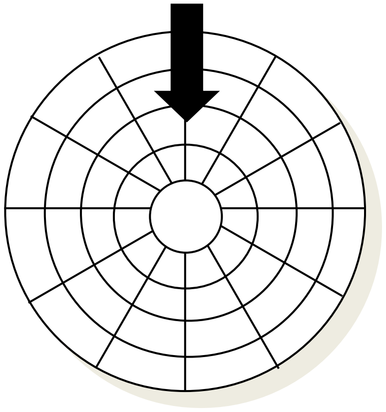
Disk Structure - top view of single platter



Surface organized into tracks

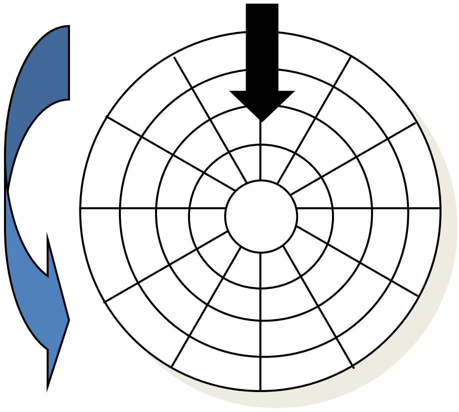
Tracks divided into sectors

Disk Access



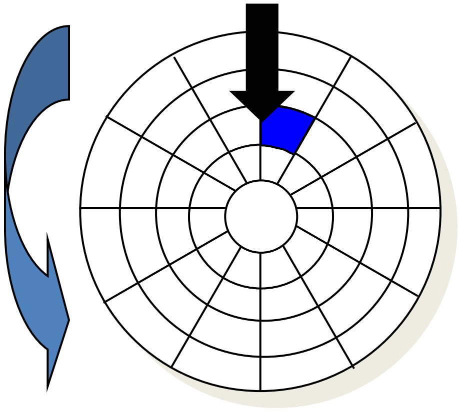
Head in position above a track

Disk Access



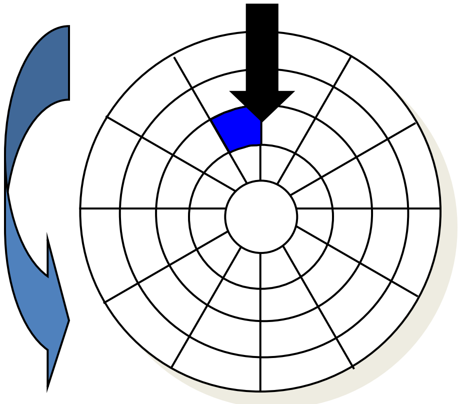
Rotation is counter-clockwise

Disk Access – Read



About to read blue sector

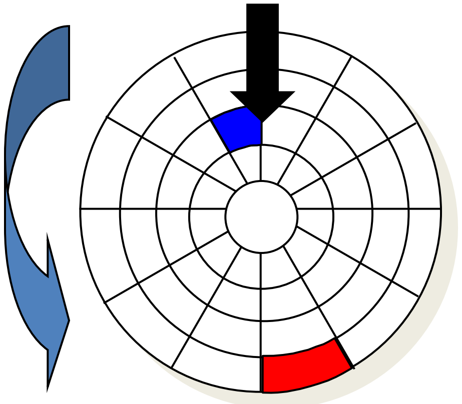
Disk Access – Read



After BLUE read

After reading blue sector

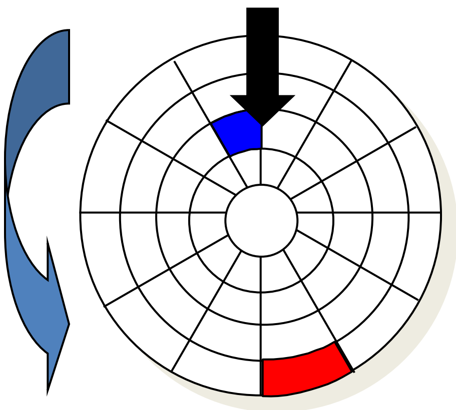
Disk Access – Read



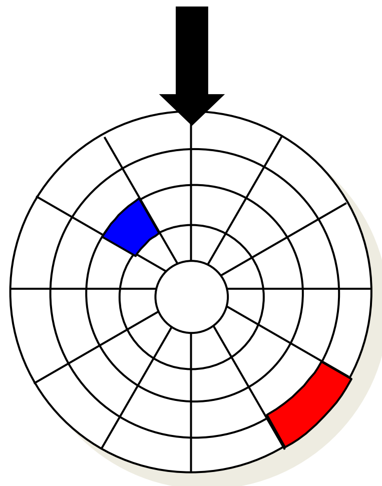
After BLUE
read

Red request scheduled next

Disk Access – Seek



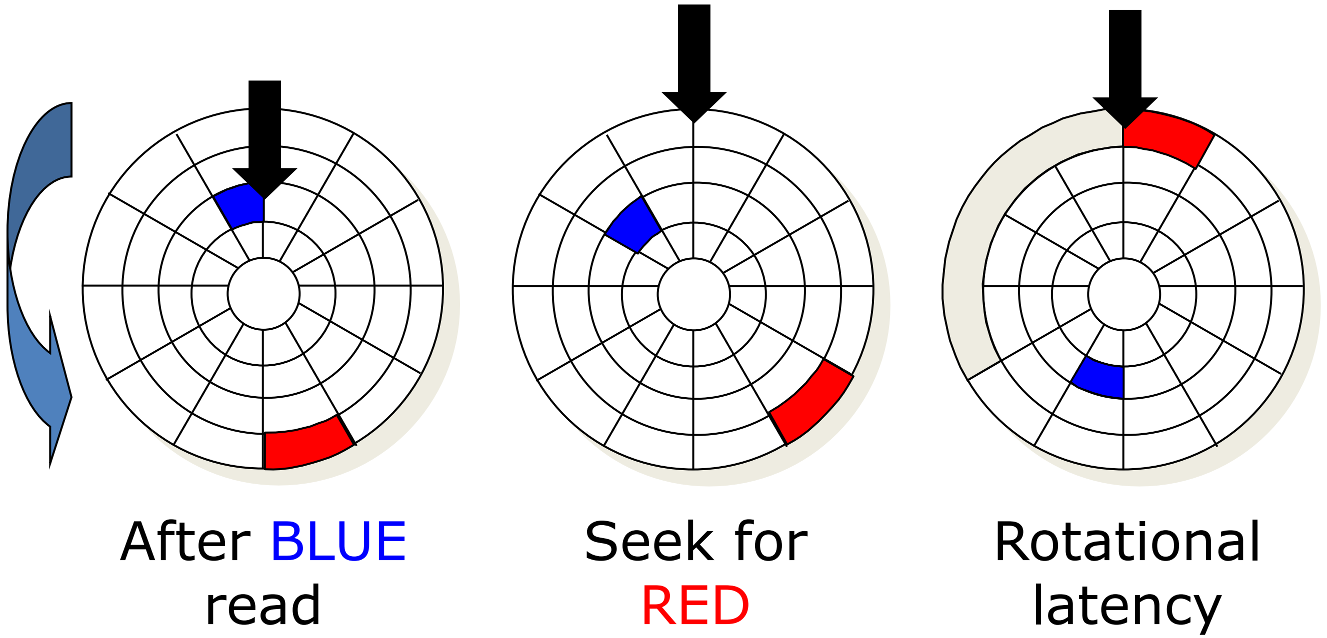
After BLUE read



Seek for RED

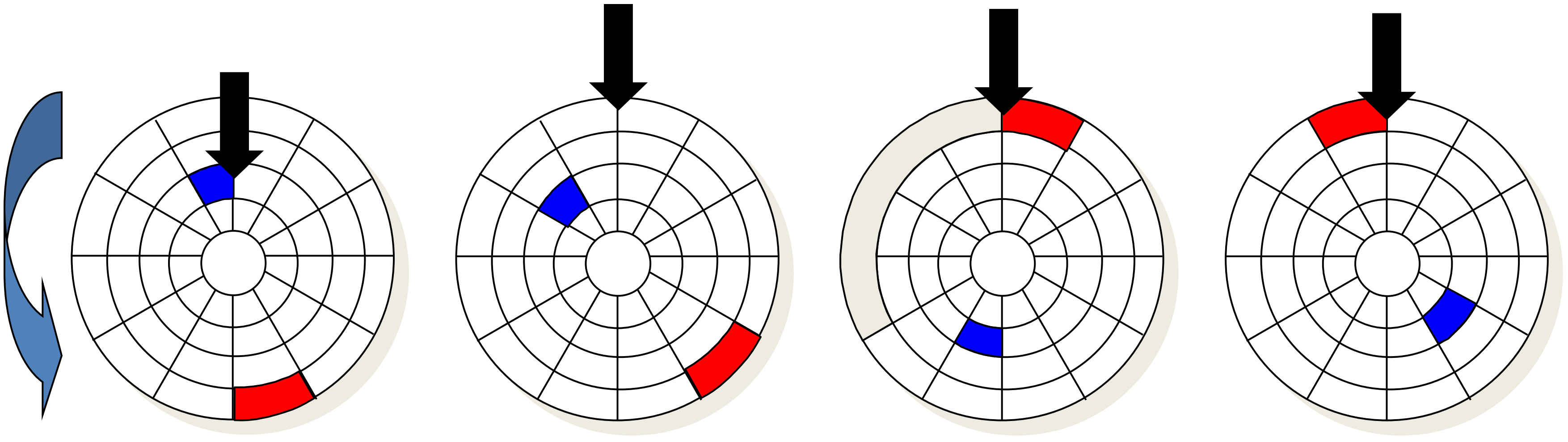
Seek to red’s track

Disk Access – Rotational Latency



Wait for red sector to rotate around

Disk Access – Read



After BLUE read

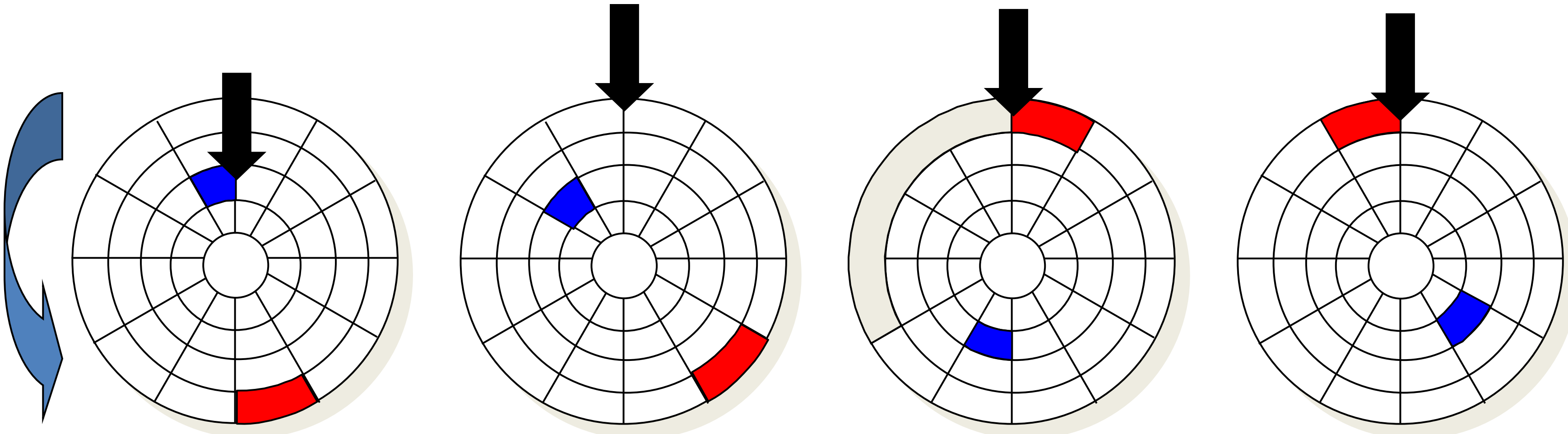
Seek for RED

Rotational latency

After RED read

Complete read of red

Disk Access – Service Time Components



After BLUE read

Seek for RED

Rotational latency

After RED read

Seek

Rotational Latency

Data Transfer

Disk Access Time

Average time to access a specific sector is close to

T_access = T_avg-seek + T_avg-rotation + T_avg-transfer

Seek time (T_avg-seek)

- Time to position head in the target track
- Typical T_avg-seek = 3-5 ms (given by a disk vendor)

Rotational latency (T_avg-rotation)

- Time for the target sector to pass under r/w head (best case: current sector, worst case: whole rotation, ½ is average rotation)
- T_avg-rotation = 1/2 x 1/RPMs x 60 sec/1 min
  - e.g., 3ms for 10,000 RPM disk

Transfer time (T_avg-transfer)

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

Disk Transfer Time

Transfer time (T_avg-transfer)

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

= whole track rotation time x 1/(avg # sectors/track)

  - for 10,000 RPM disk with 1,000 sectors/track, we have
  - 1/RPMs x 60 sec/1 min x 1/1000 sectors/track = 0.006 ms

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

= 512B / $6*10^{-6}$s = 85.33 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, ^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_avg rotation = 1/2 x (60 secs/7200 RPM) x 1000 ms/sec = 4 ms
- T_avg transfer = 60/7200 RPM x 1/1000 secs/track x 1000 ms/sec = 0.0083 ms
- T_access = 5 ms + 4 ms + 0.008 ms = 9.008 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

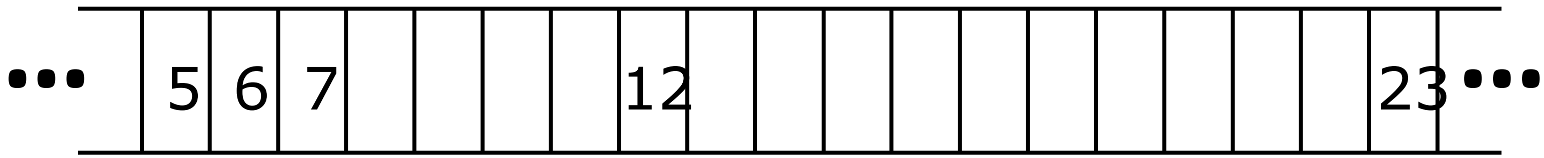
Interacting between OS and Disk

- 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

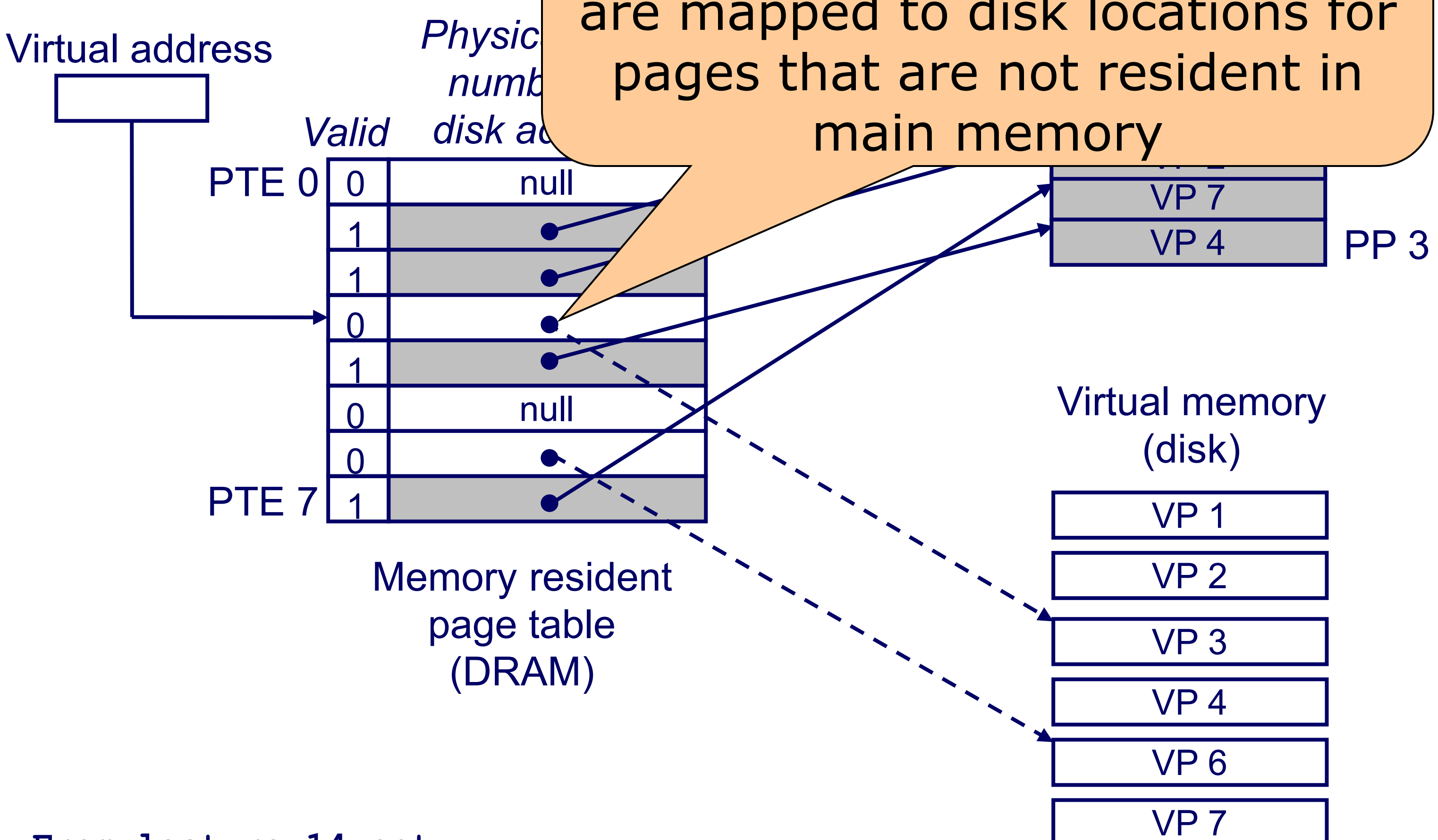
- Disk is not directly managed by OS
  - OS only knows the physical memory space
  - By MMU, OS creates a page table for each process
- Disk controller managed the data stored in disk
  - OS interact with the disk controller for I/O
  - The disk controller takes LBNs from OS to find them or write them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

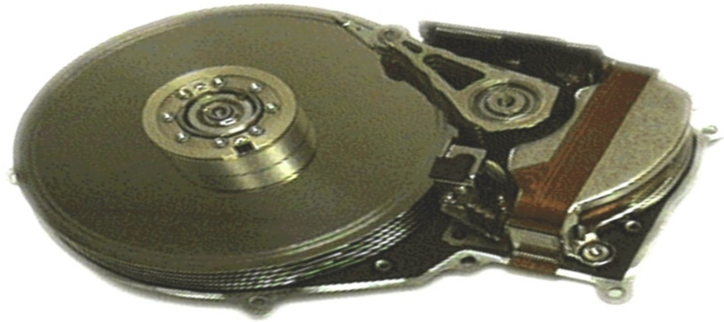
– Example: An instruction refere… …rs a page fault exception

“logical block numbers” in OS domain (virtual page number) are mapped to disk locations for pages that are not resident in main memory

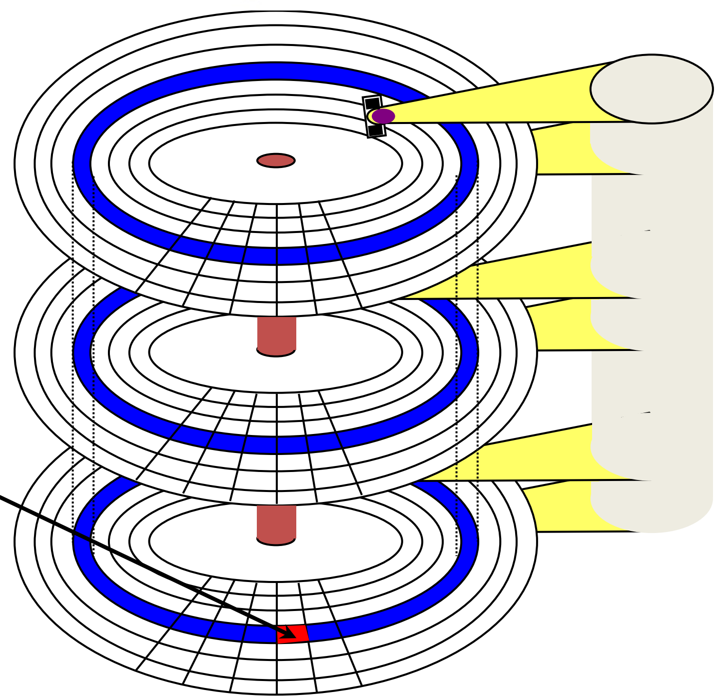


From lecture-14.ppt

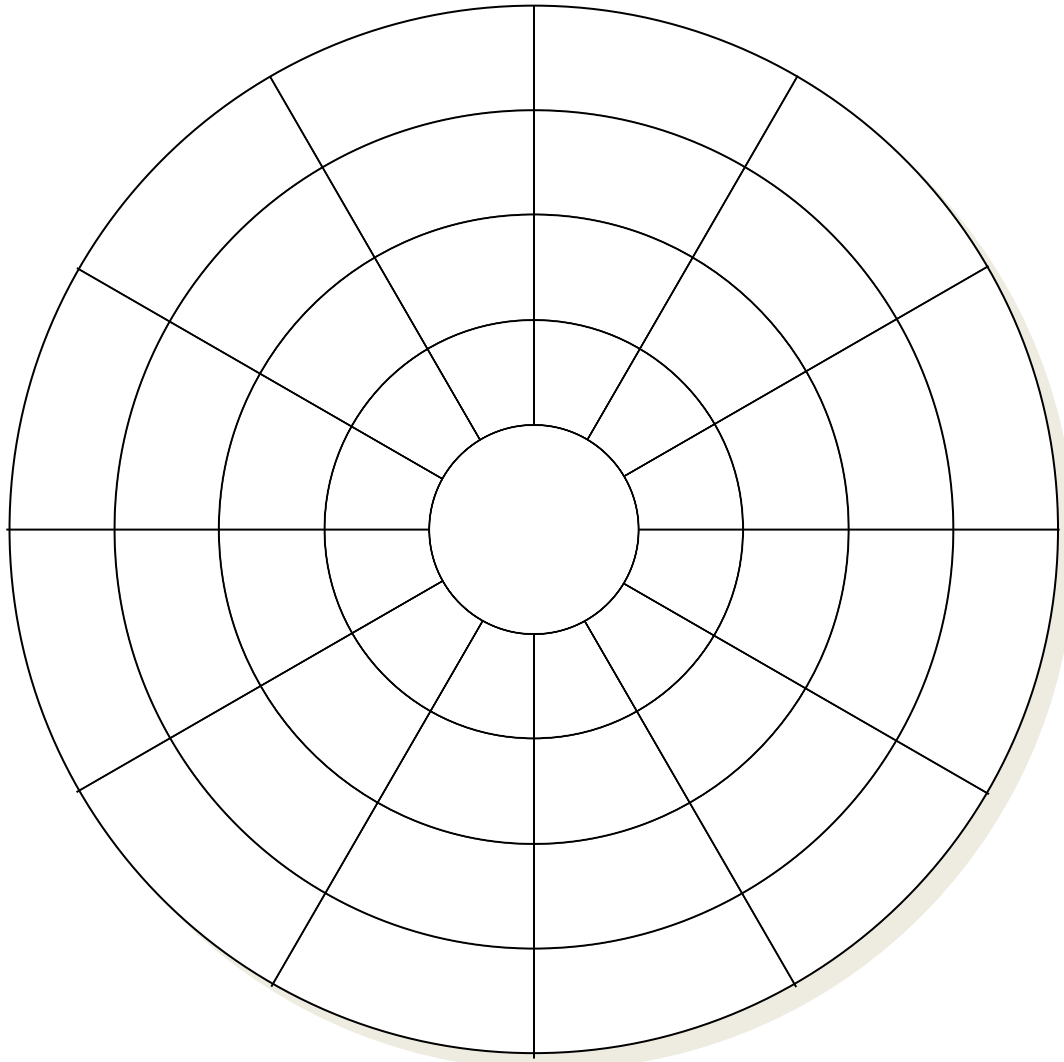
In device, “blocks” are mapped to physical store



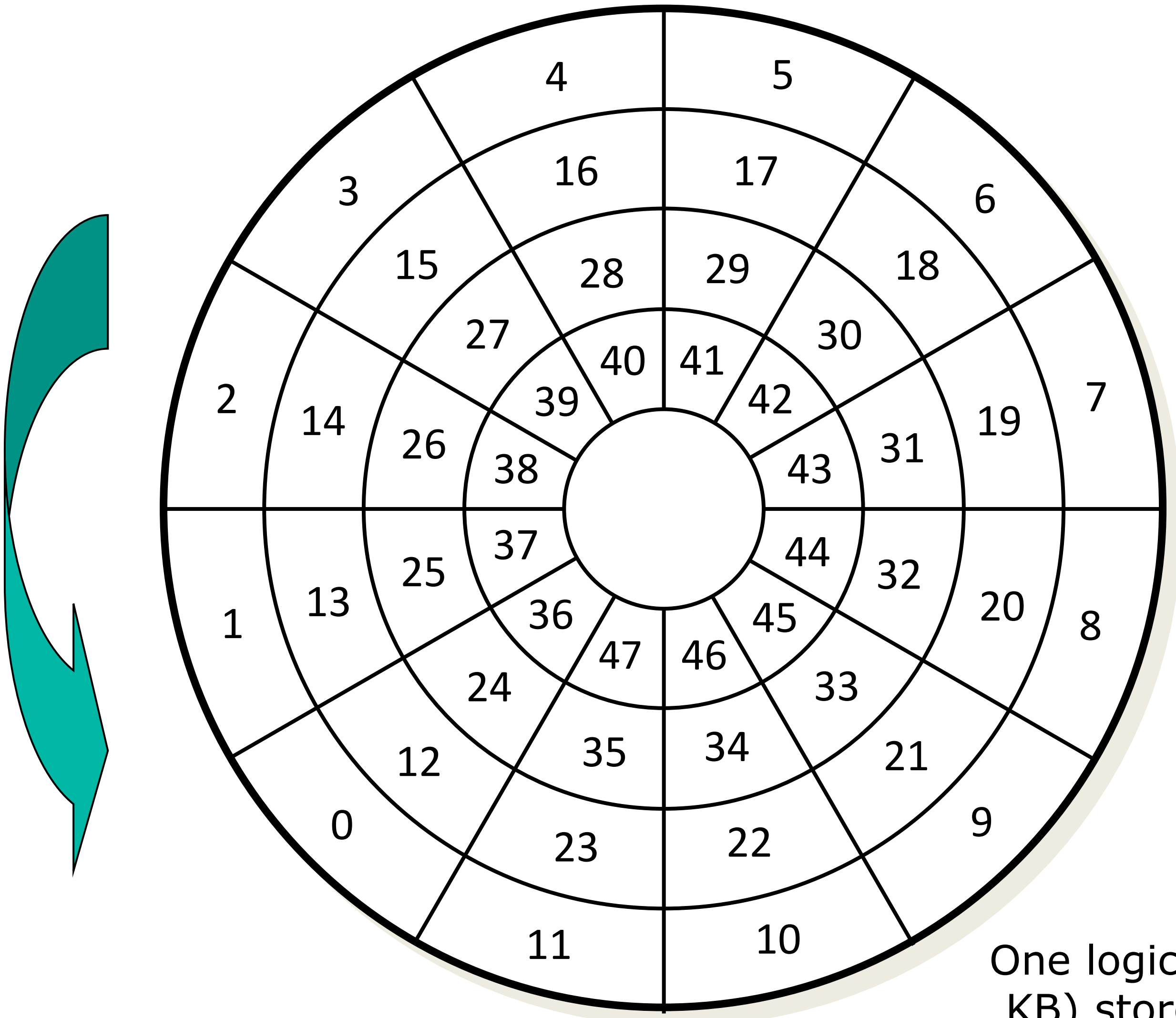
Disk Sector
(512 Bytes as a unit.
A logical block is
contiguously stored
in multiple sectors.)



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 GB = 10^9 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^2): 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

Capacity = (# bytes/sector) x (avg. # sectors/track) x

(# tracks/surface) x (# surfaces/platter) x

(# platters/disk)

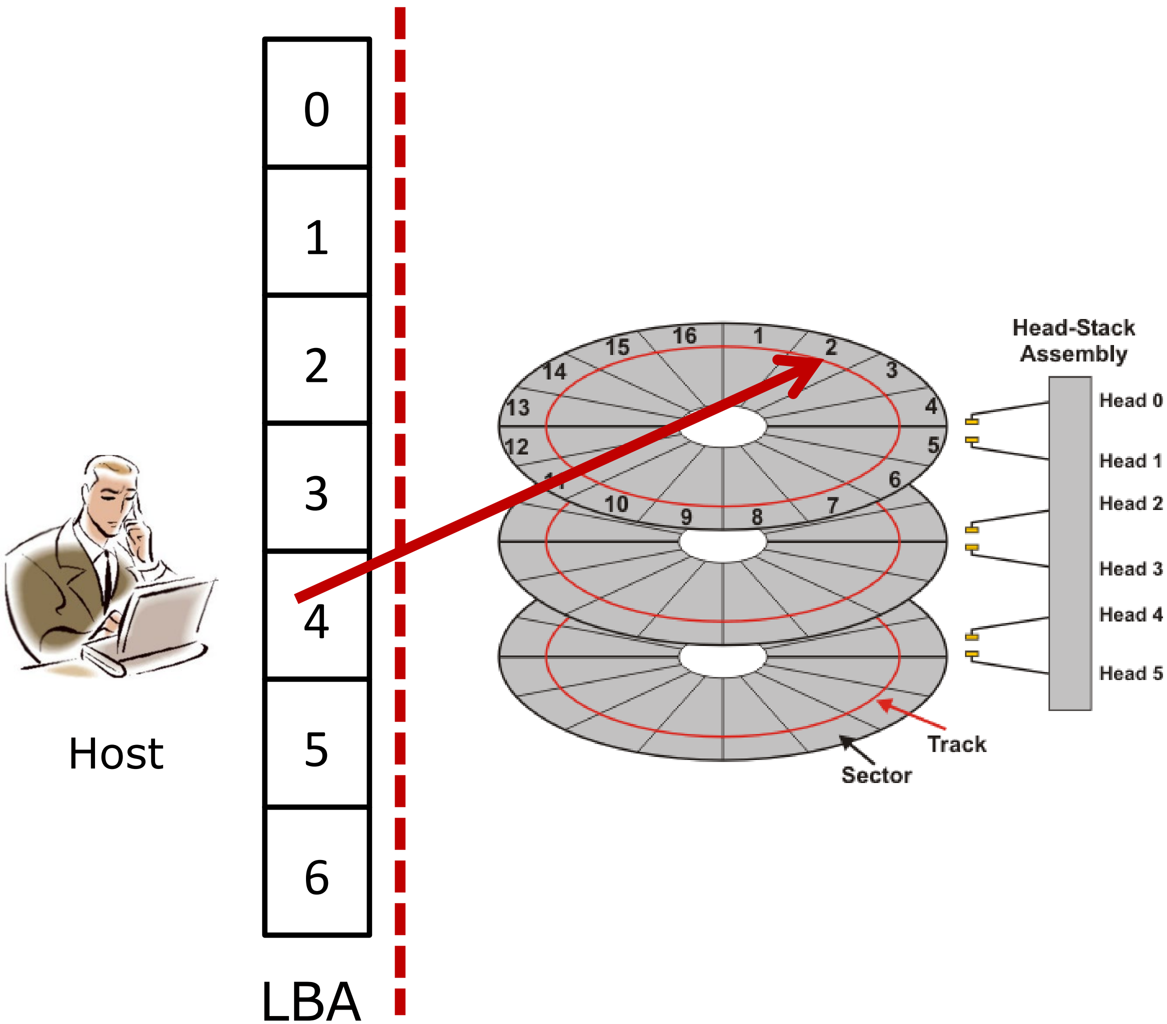
Example:

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

Capacity = 512 x 1000 x 20000 x 2 x 5

= 102.4 GB

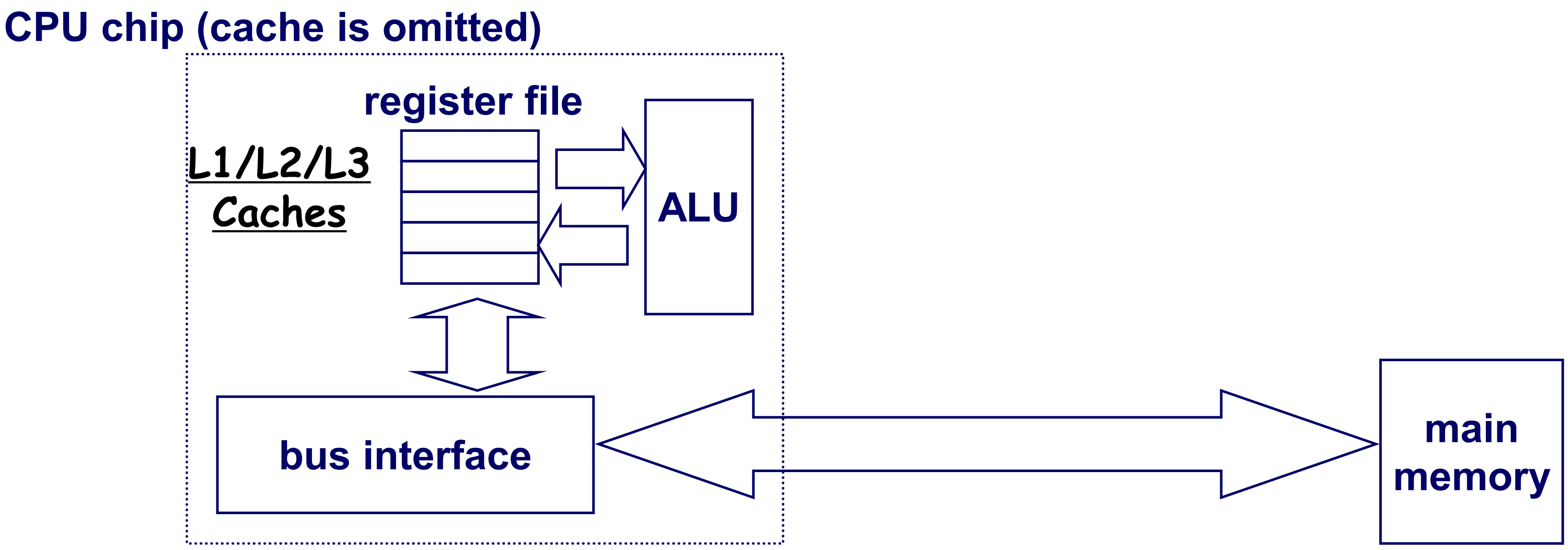
Physical data layout in HDD




```
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)

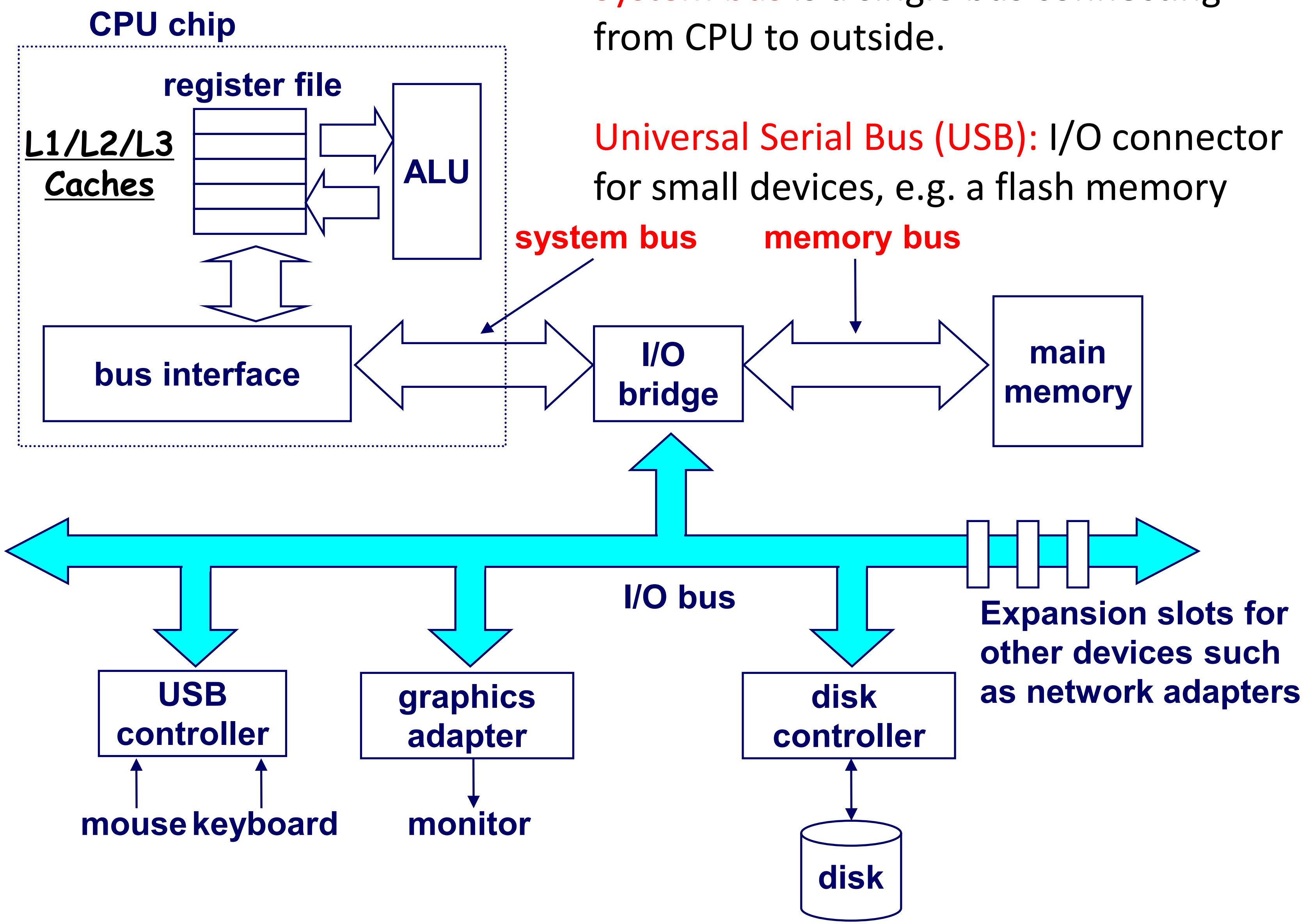
Looking back at the hardware



Connecting I/O devices: the I/O Bus

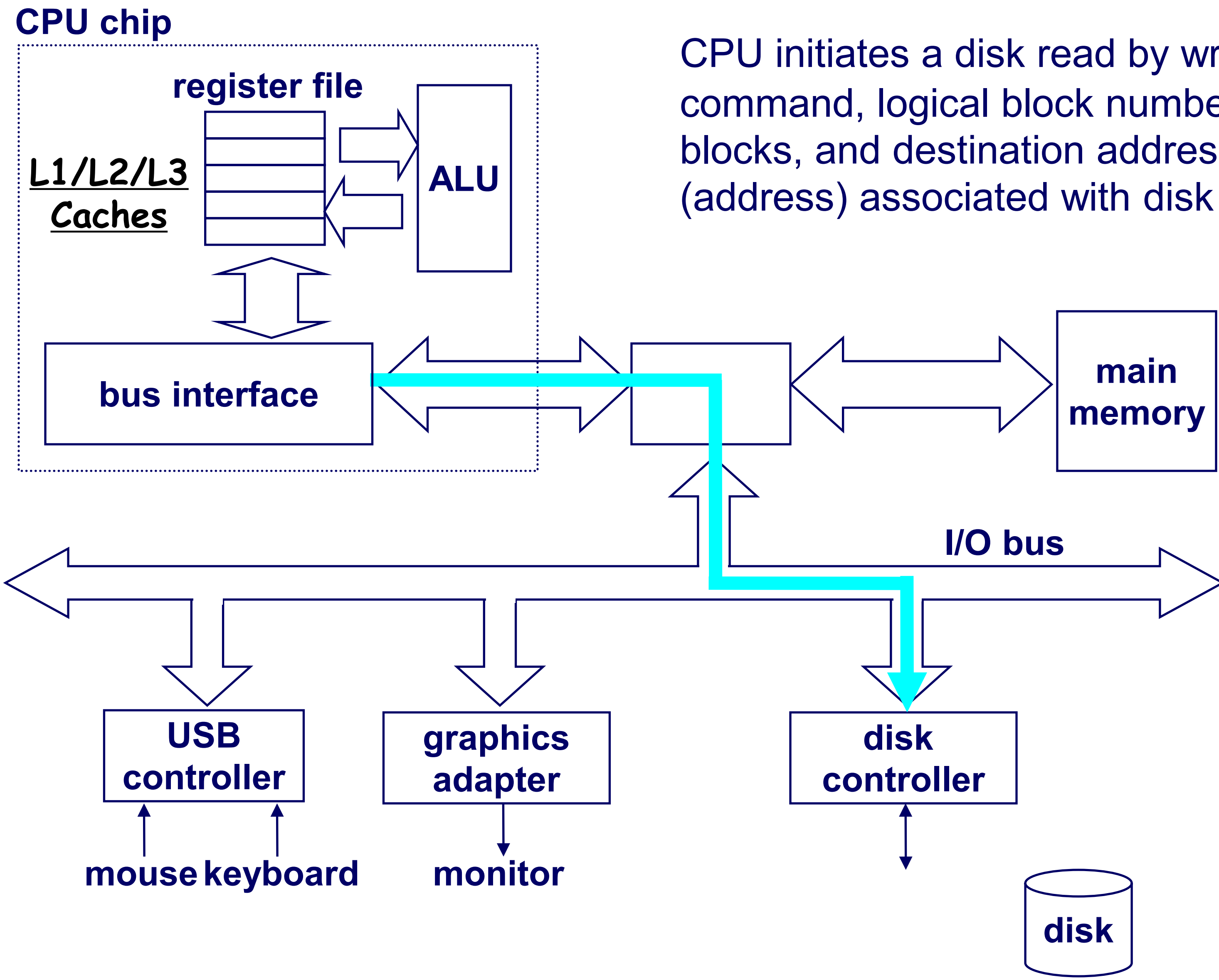
System bus is a single bus connecting from CPU to outside.

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



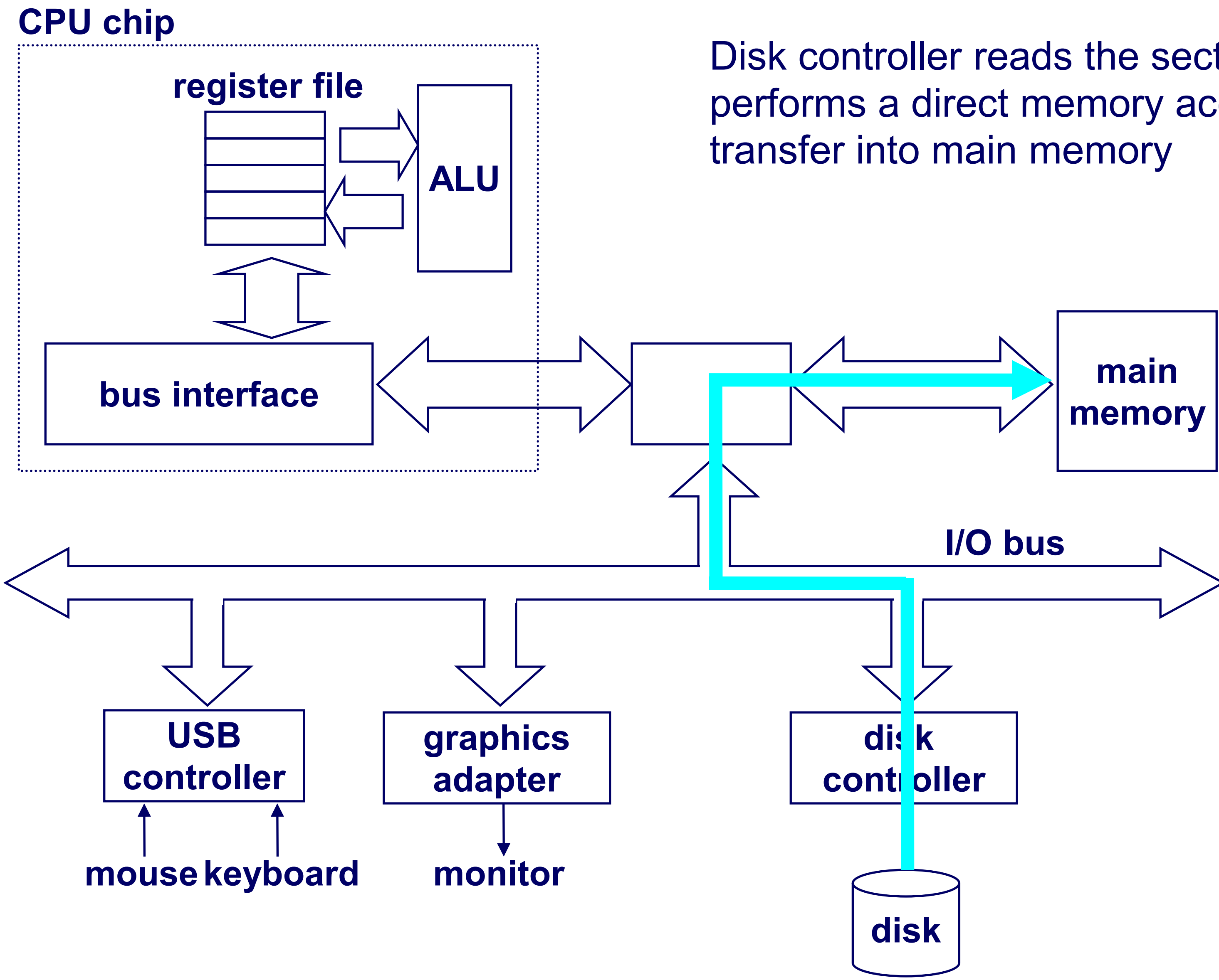
Reading from disk (1)

CPU 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



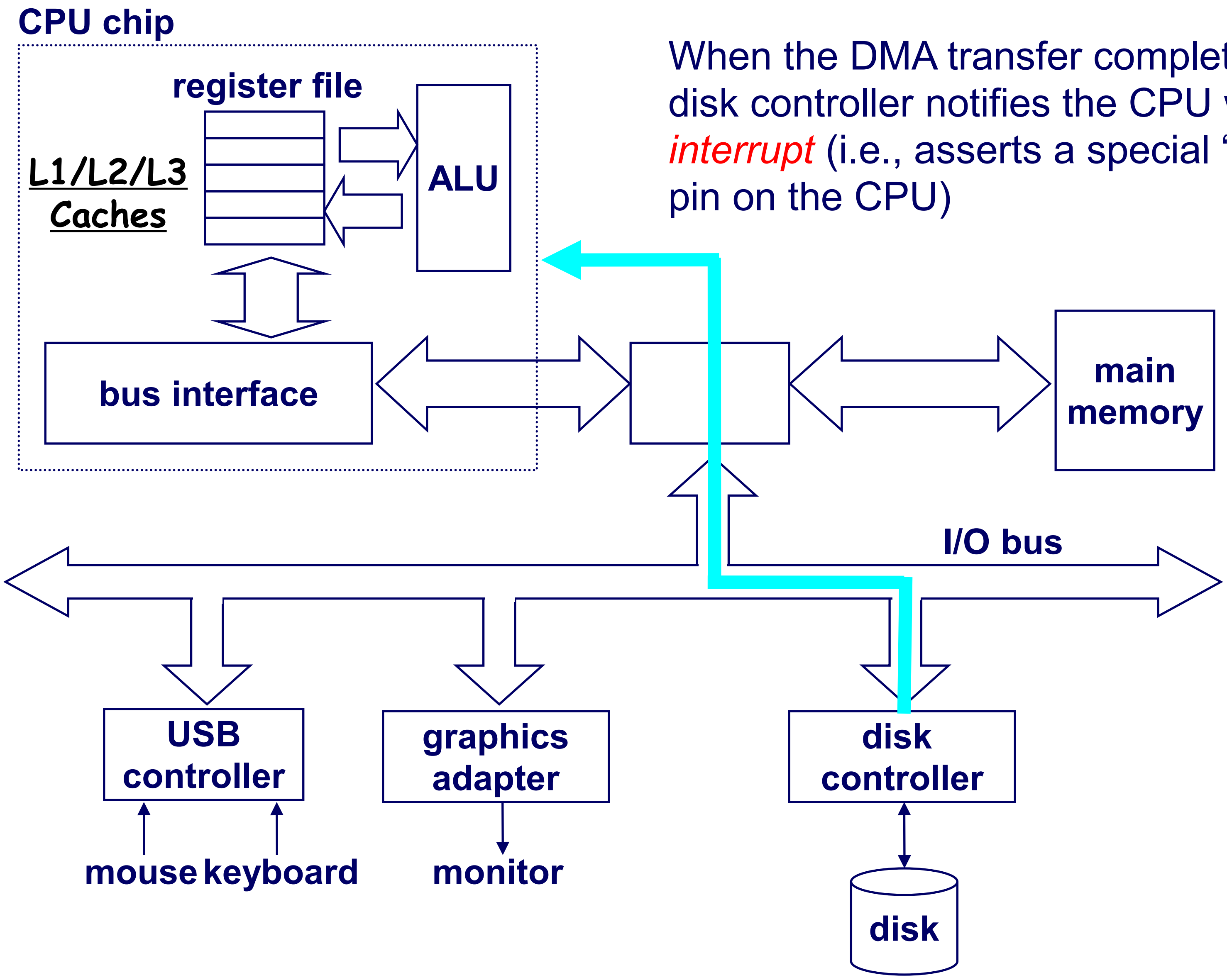
Reading from disk (2)

Disk controller reads the sectors and performs a direct memory access (DMA) transfer into main memory

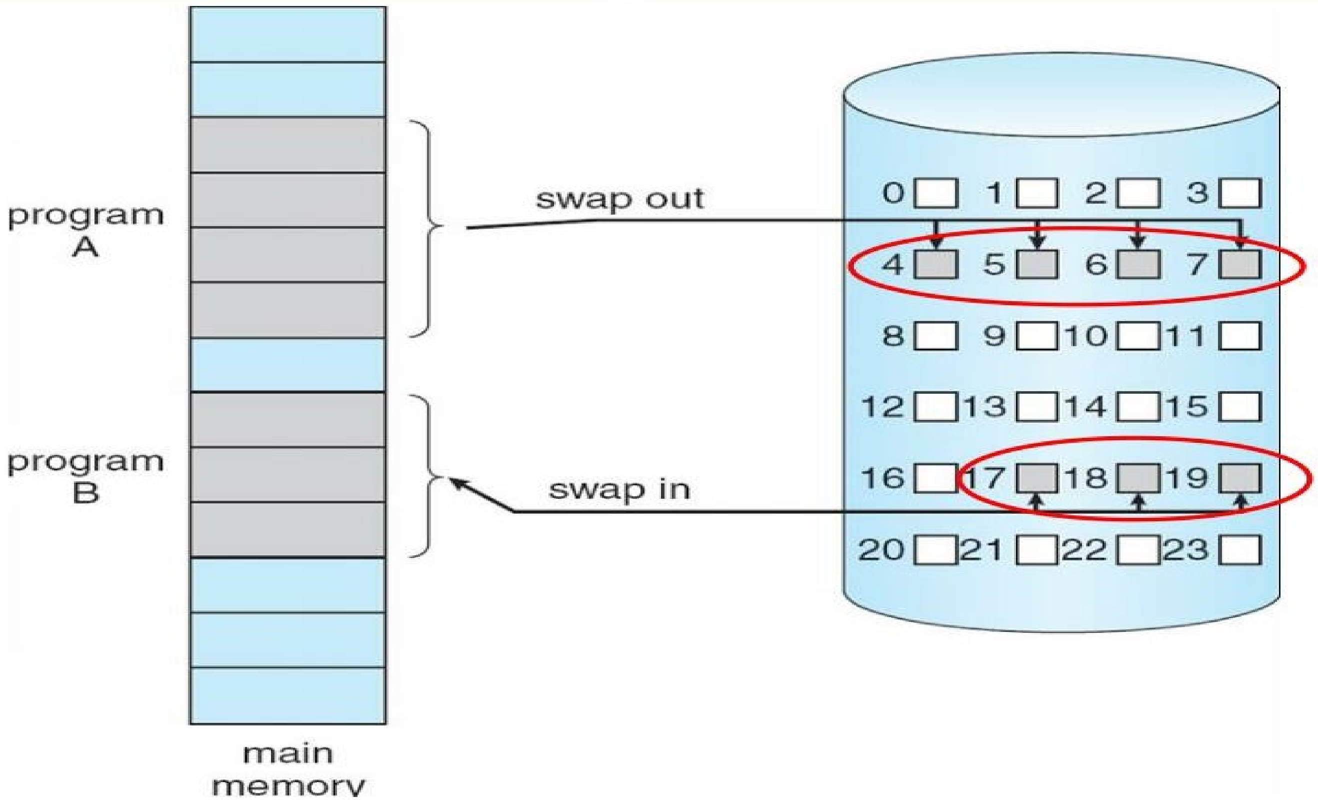


Reading from disk (3)

When the DMA transfer completes, the disk controller notifies the CPU with an *interrupt* (i.e., asserts a special “interrupt” pin on the CPU)



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

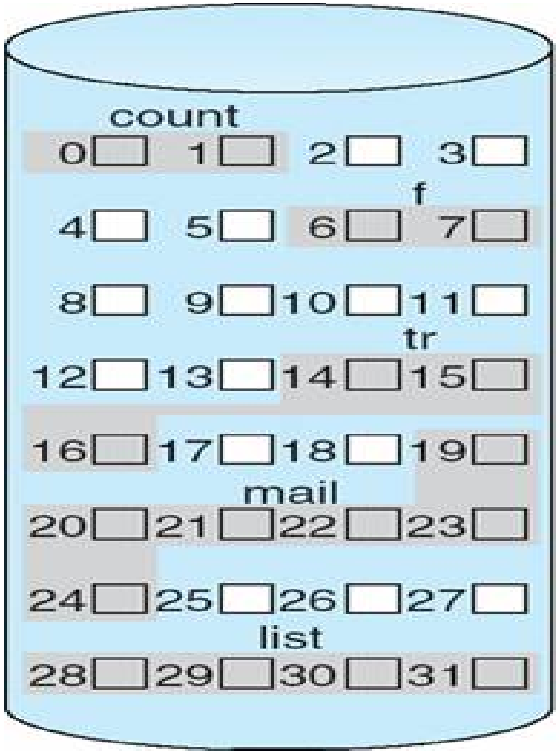
The two are bridged by another mapping table in the disk controller

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



directory

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

Contiguous Allocation

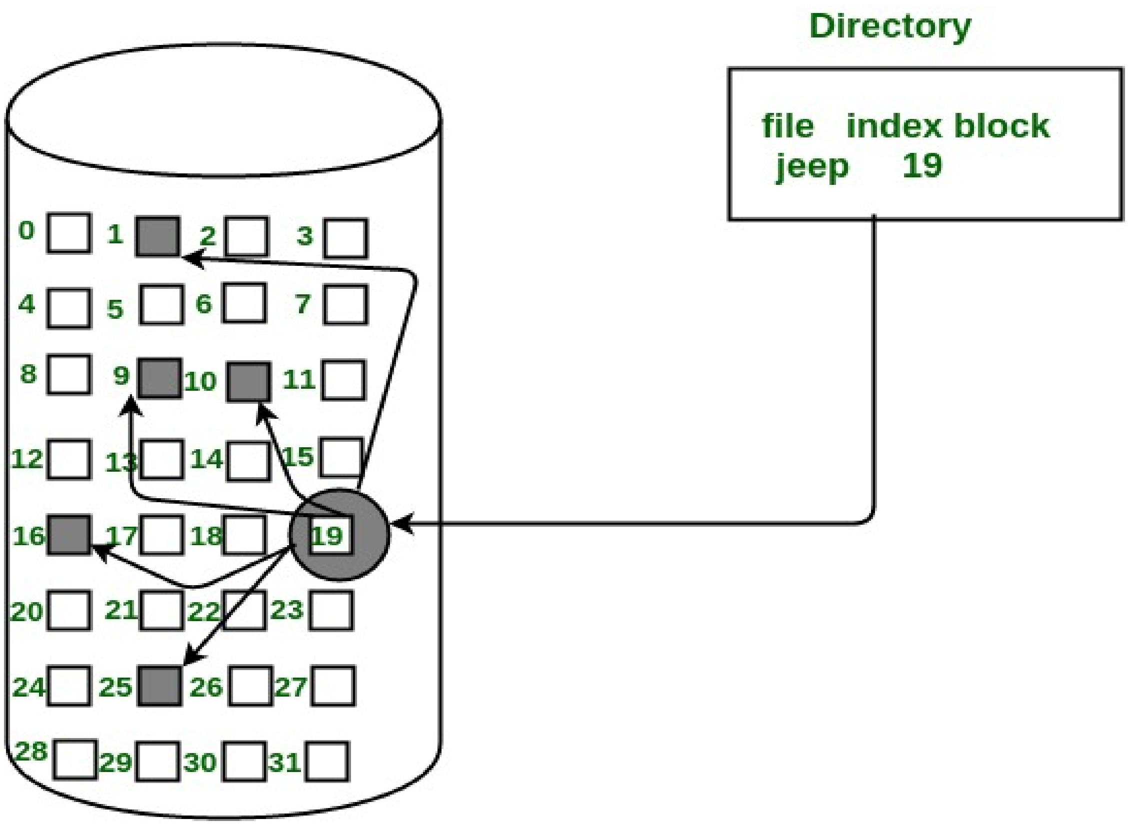
Logical blocks are contiguously allocated in disk sector by sector under a file name (and a user ID), a starting sector number and the length (# converted blocks)

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**

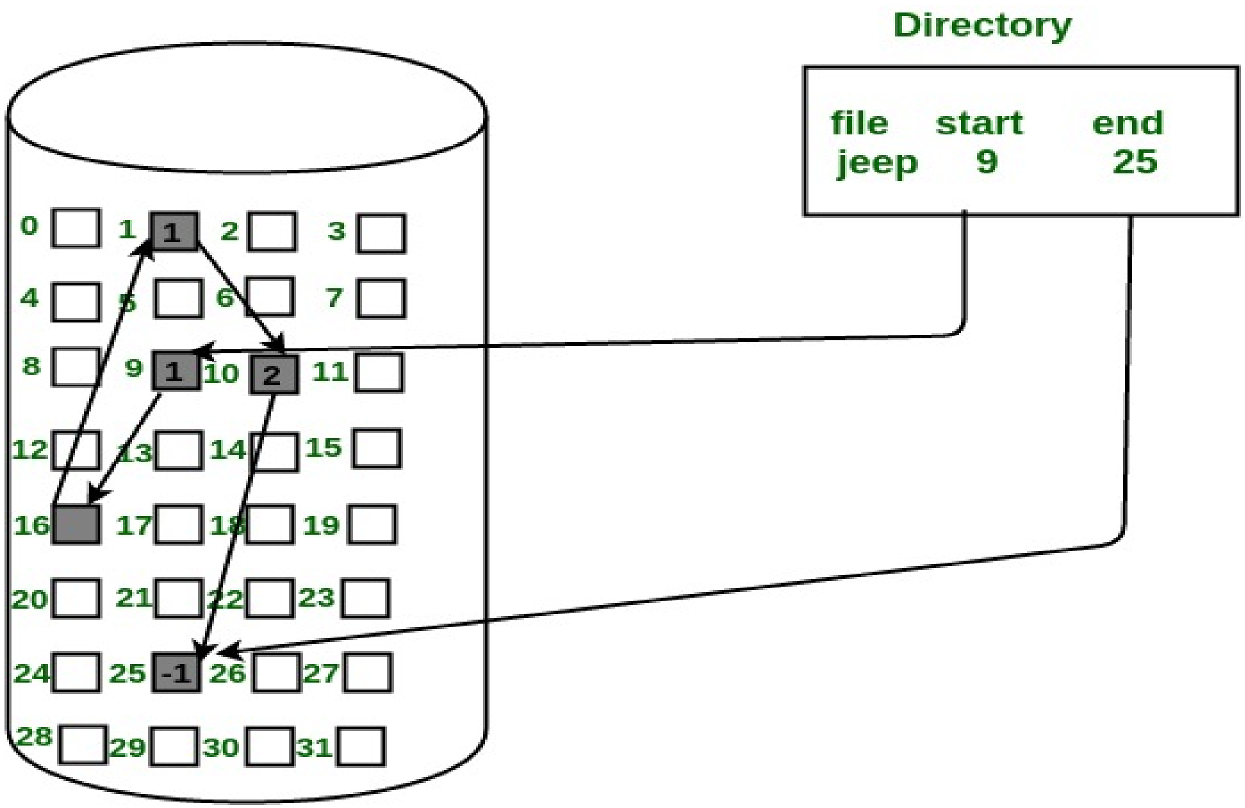
One pointer for all related sectors, additional support is needed in the disk controller.

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"

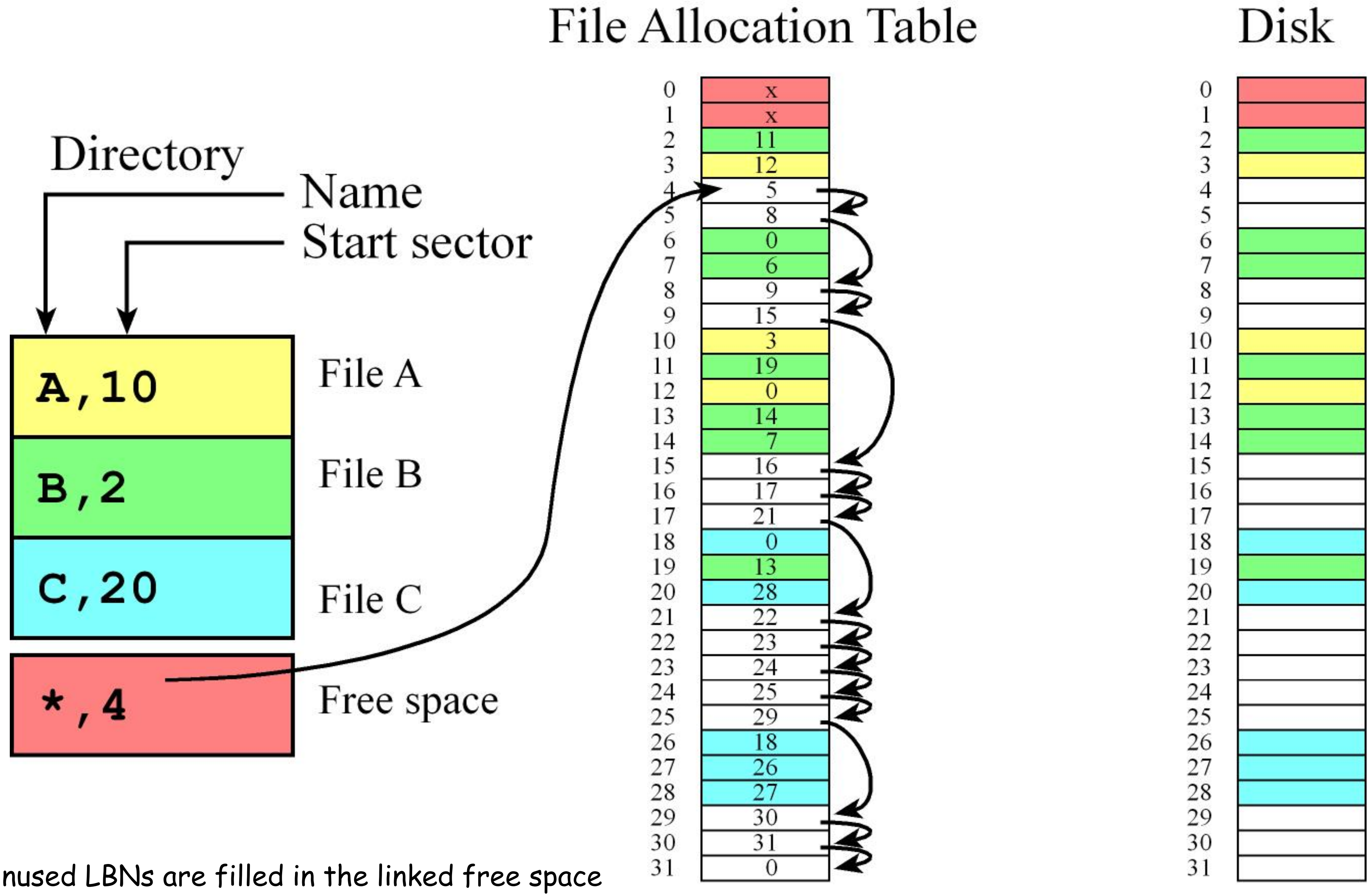


A linked-list is formed for all the related sectors, additional support is needed in the disk controller.

Pro and Con of Linked Allocation

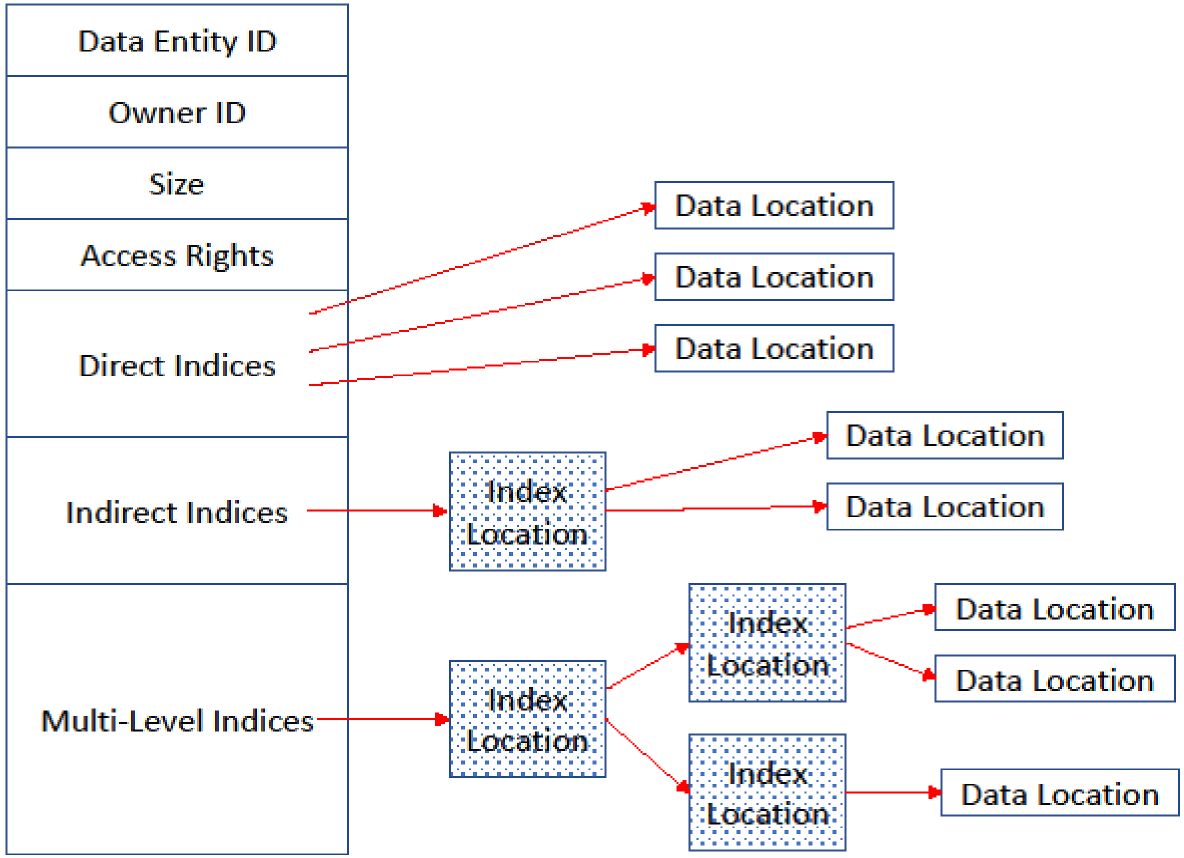
- Space utilization is high
  - A file grows and shrinks easily
  - The directory is simple
- Accesses to each block are random in disk
  - Seek times cause high latency
  - Any file access must start from the link head
- File Allocation Table (FAT)
  - This implementation creates a table for linked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

File Allocation Table (FAT)



Unused LBNs are filled in the linked free space

Disk Allocation in File System is in "inode"



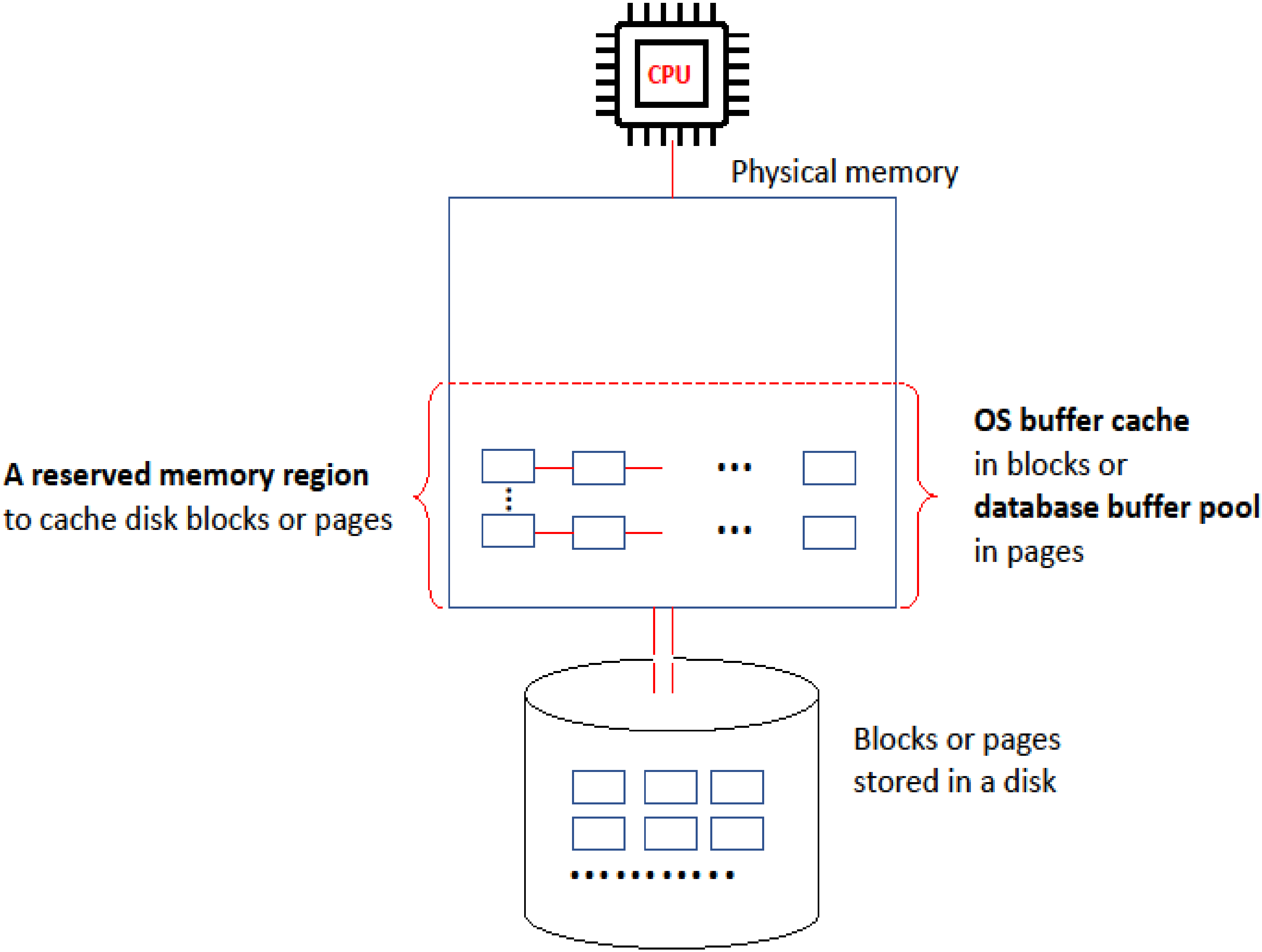
An Index Node or Information Node (inode) collects all the related information items for a data entity (multiple files) based on ownership, access rights …

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?