# CS2106
# Introduction to OS

Office Hours, Week 13

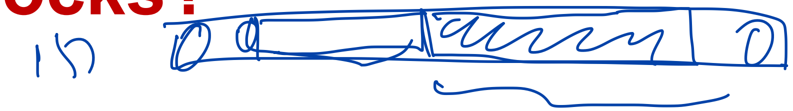
# Sectors and Partitions

**What’s the relationship between partition and sector? Does each partition correspond to a sector on the disk? Or does a partition occupy more than 1 sector's worth of space on the disk, or does a sector contain multiple partitions?**

- Sector is a smallest unit of access of the disk
- A partition occupies a large number of sectors

# Logical Blocks and Partitions

**Is it correct to say that disk is made up of several partitions, and within 1 partition, there is a 1D array of logical blocks that store file data. So if there are multiple partitions on the disk, there are multiple arrays of logical blocks?**



- It is OK to think that way.
- Usually, we abstract the whole disk as a 1D array.
- But you can think of each partition being a 1D sub-array within the big 1D array.
- Note that most file systems require partitions to be contiguous sub-arrays within the 1D array.
- 1 logical block = 1 sector (usually)

# Is everything stored on the disk?

**Is it correct to say that all the components of the partition (e.g., file info, directory structure etc) are all stored on the disk? I'm confused as during the lecture, it was said that certain file info can also be stored in memory?**

- Everything is on the disk,
  - or whatever else is the storage device (SSD, DNA, …)
- However, certain information is kept in memory as well, to avoid tremendous cost of traversing pointer-like data structures on the disk

# Open-file table

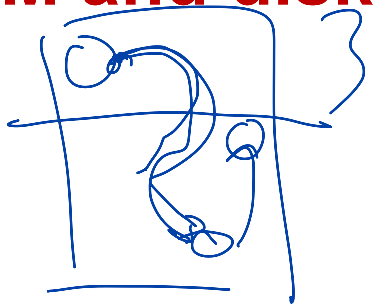
**Is the open file table stored in RAM/Memory?** ✓

- Yes
- Note that open-file table is not persistent!
- The open-file table holds information about **open** files
  - i.e., it tracks active usage of files by processes

# FAT

**In slide 13, it say the FAT table is stored in memory. Does that mean that the FAT table is stored both in RAM and disk?**

- Yes!
- FAT table must exist on the disk
- FAT table should exist in memory too
  - Otherwise, it would be extremely slow to traverse the FAT table if it’s only on the disk
  - FAT table usually has poor locality properties, and caching it partially doesn’t work



# FAT

**For slide 13, there was mention that "FAT keep tracks of all disk blocks in a partition". Does partition here refer to a partition of the disk, or does it refer to partition in the physical memory?**

- Partition on the disk

# Indexed Allocation

**For indexed allocation, each file has an index block. Is there another table that maps each file to their corresponding index block?**

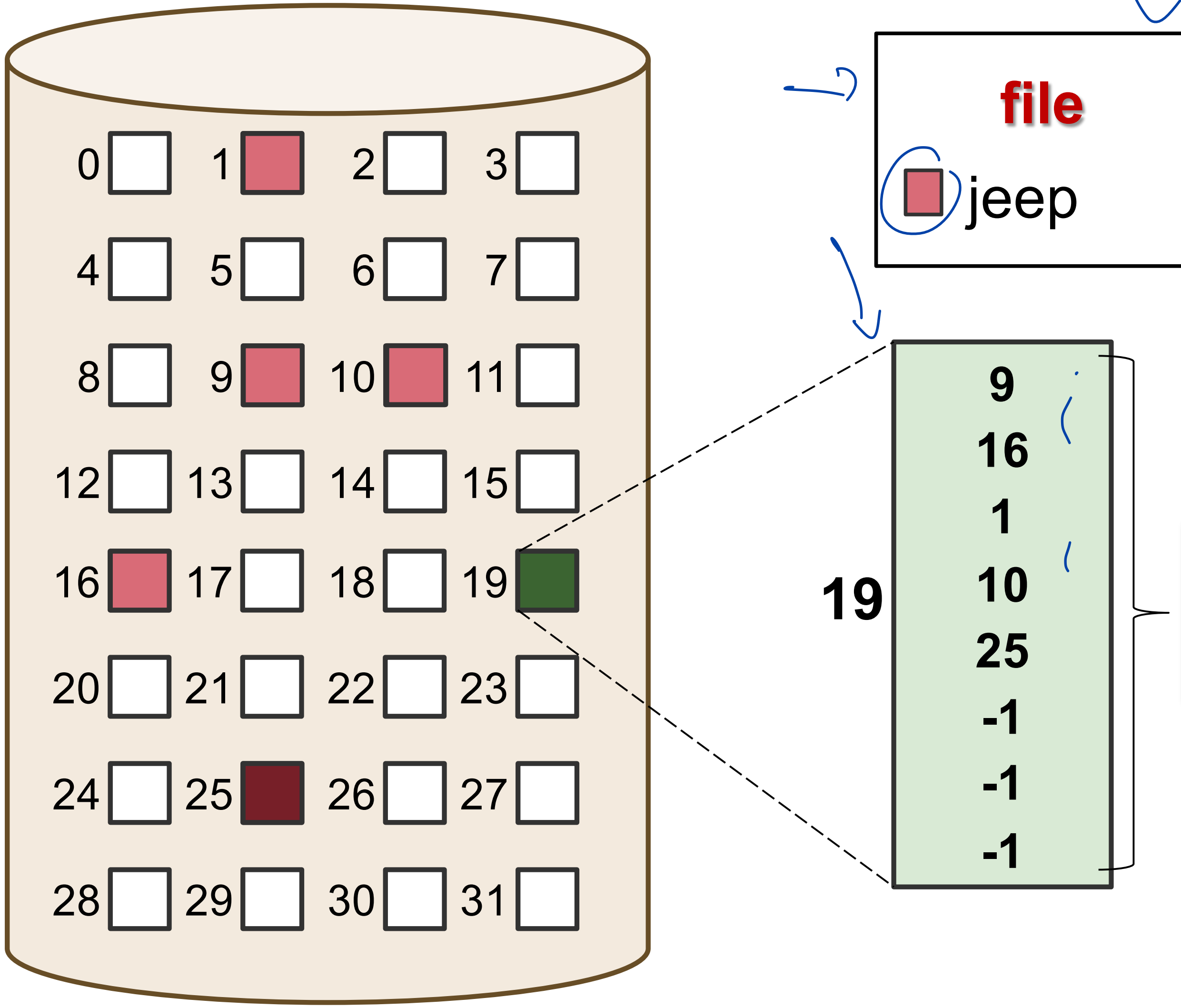
- For each file the OS needs to keep track only of the starting index

**How do we know which index block is for which file?**

- This information is actually kept in the directory!
  - Or there's a pointer to it from the directory

# Indexed Allocation (L12 Slide 16)

| file | index block |
|---|---|
| 🟥 jeep | 19 |



19

9
16
1
10
25
-1
-1
-1

Disk Block used by this file (in order)

# Indexed Allocation

**For indexed allocation, based on slide 16, it seems that the per file table is stored inside the disk at a particular logical block. However, it was mentioned during lecture that this per file table is stored in memory? So where is it stored?**

- All such structures are stored on the disk both
- However, it becomes very easy to bring such a per-file block into memory when you open a file
  - Because it's much smaller than giant system-wide FAT
  - The OS loads the structure in memory for 10000x faster access

# Indexed Allocation

**In slide 15, it was mention that: Only index block of opened file needs to be in memory. I thought index block number is part of file info which is stored within the partition on the disk? Why does index block number have to be stored in memory?**

- Index block IS part of the file
- But that block is cached in memory when the file is open
  - Allows quick access to any part of the open file
  - Important for performance

# FAT vs. Indexed

**Is it correct to say that FAT & indexed allocation both store a pointer table just that FAT stores the table in memory while Indexed stores it in disk. Thus, indexed will be slower compared to FAT?**

- No
- **FAT** has one, **system-wide** table for all files
- **Indexed** allocation uses **per-file** tables, which are small.
- Both of them can be kept in memory
- FAT table must be in memory **at all times** for the system to work well
  - Else, the file system would be unacceptably slow
- Indexed blocks are stored on the disk and loaded only when a **particular file** is used (**on demand**).

# System-wide table vs. per-X table

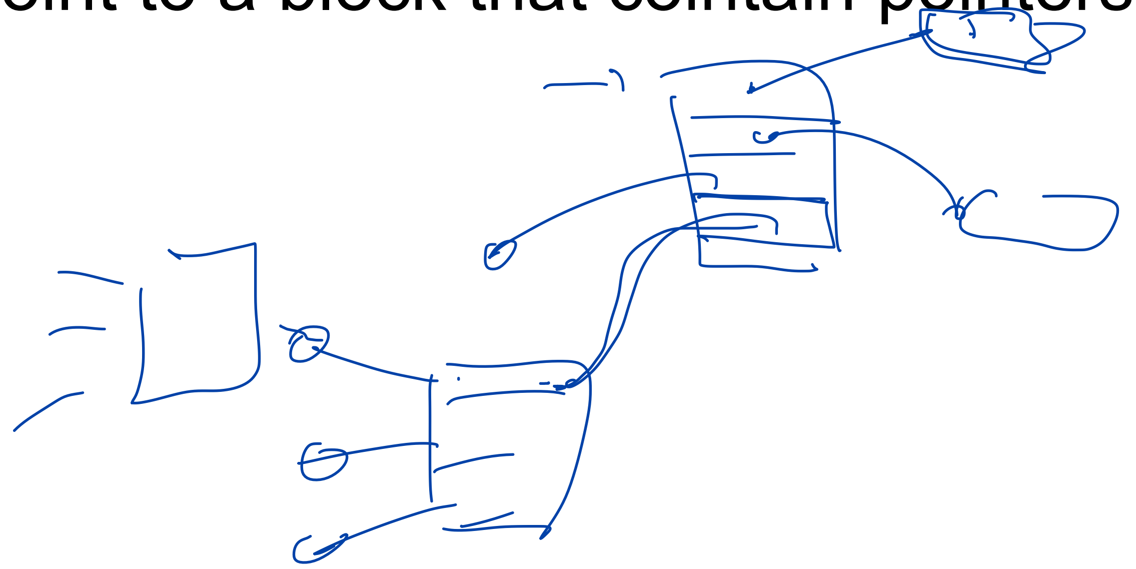
**What is a systemwide table?**

- A table of which there’s only one instance in the system
  - Regardless of the number of “entry holders” (users, files, processes, …)
- E.g., inverted page table is a system-wide table
  - As opposed to direct, per-process page tables
- System wide open-file table keeps info for all processes
  - As opposed to per-process
- FAT keeps the info for all files
  - As opposed to per-file, like indexed

# Hierarchical Indexing

**Can you explain what's the difference between single indirect and direct pointers?**

- Direct pointers: directly point to a file block
- Single indirect pointers: point to a block that cointain pointers to file blocks
  - Like a page directory



# Swap Partition

**Have we covered swap partitions/memory yet?**

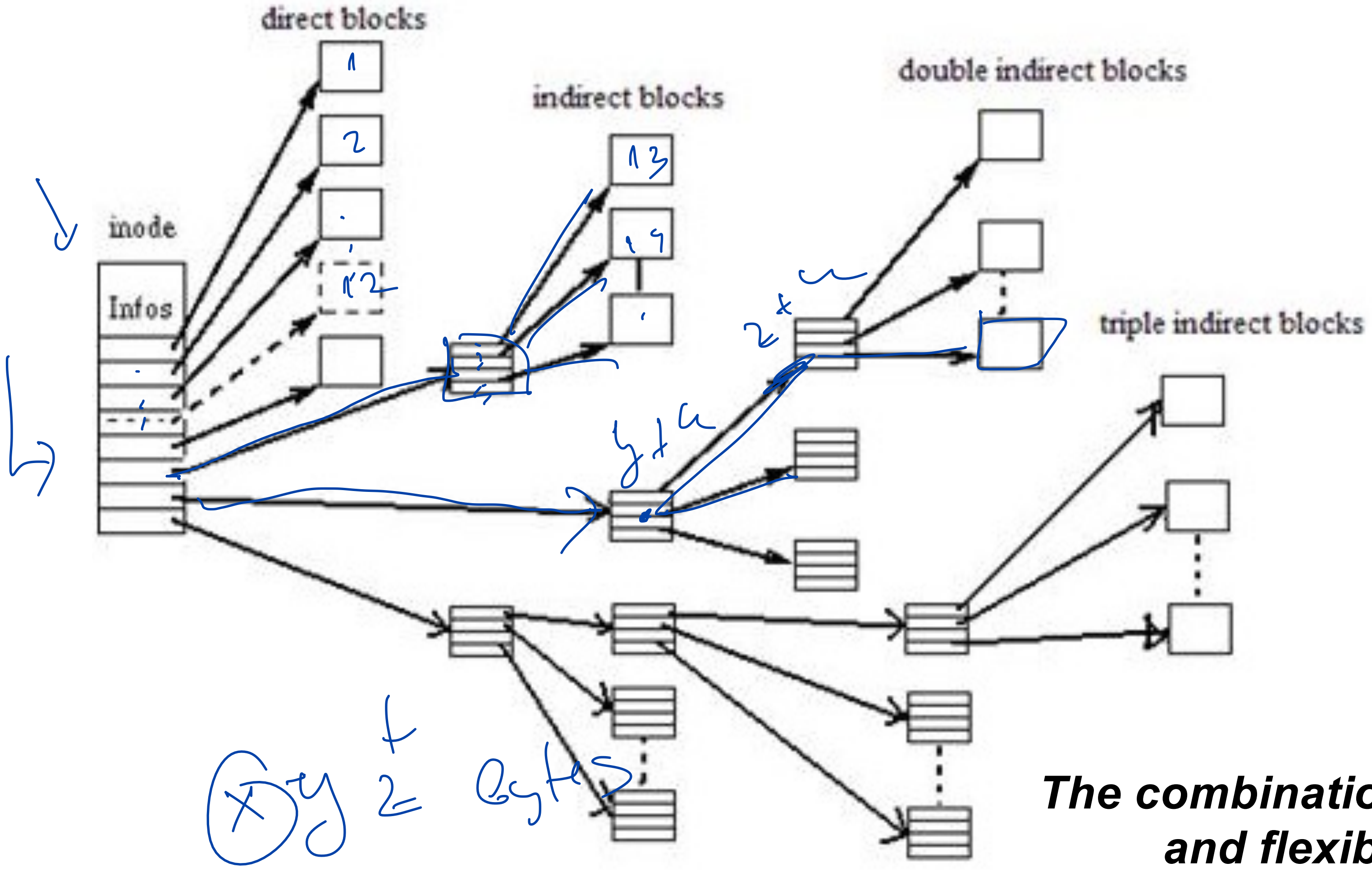
- No

**If not, will this topic be covered?**

- We will not
- However, it is simply a collection of pages
- Easier to manage than file system

# I-Node Unrolled (L12 S20)

13



- 12 **direct pointers** that point to disk block directly
- 1 **single indirect block**
  - contains a number of **direct pointers**
- 1 **double indirect** block
  - points to a number of **single indirect blocks**
- 1 **triple indirect block**
  - points to a number of **double indirect blocks**

***The combination ensures efficiency for small files and flexibility (still support larger files)***

# I-Node

**What if a triple indirect block is not enough to store all the data?**

- All file systems have limitations on the size of the file
  - ext4 i-node supports files up to 16 TiB (terabytes)
  - In practice, hard to reach that limit
- Files bigger than 16TiB  files are extremely rare
  - used only in astronomy, CERN, meteorology
- It is not hard to design a specialized file system for monster files
  - No fundamental limitation to file size

# I-Node

**Is the I-Node scheme the implementation of the Linux ex4 file system?**

- Yes!

# File Info

**Are the read/write permissions stored in the metadata of the files?**

- Yes!

↓

**If yes, does it mean chmod is changing the metadata**

- Yes!
- Doesn’t change the file, only the metadata

# File Info

**The File info in slide 34 is referring to the data structure as the last pointer on slide 30 ("Store only file name and points to some data structure for other info")?**

- Correct

# File System Operations

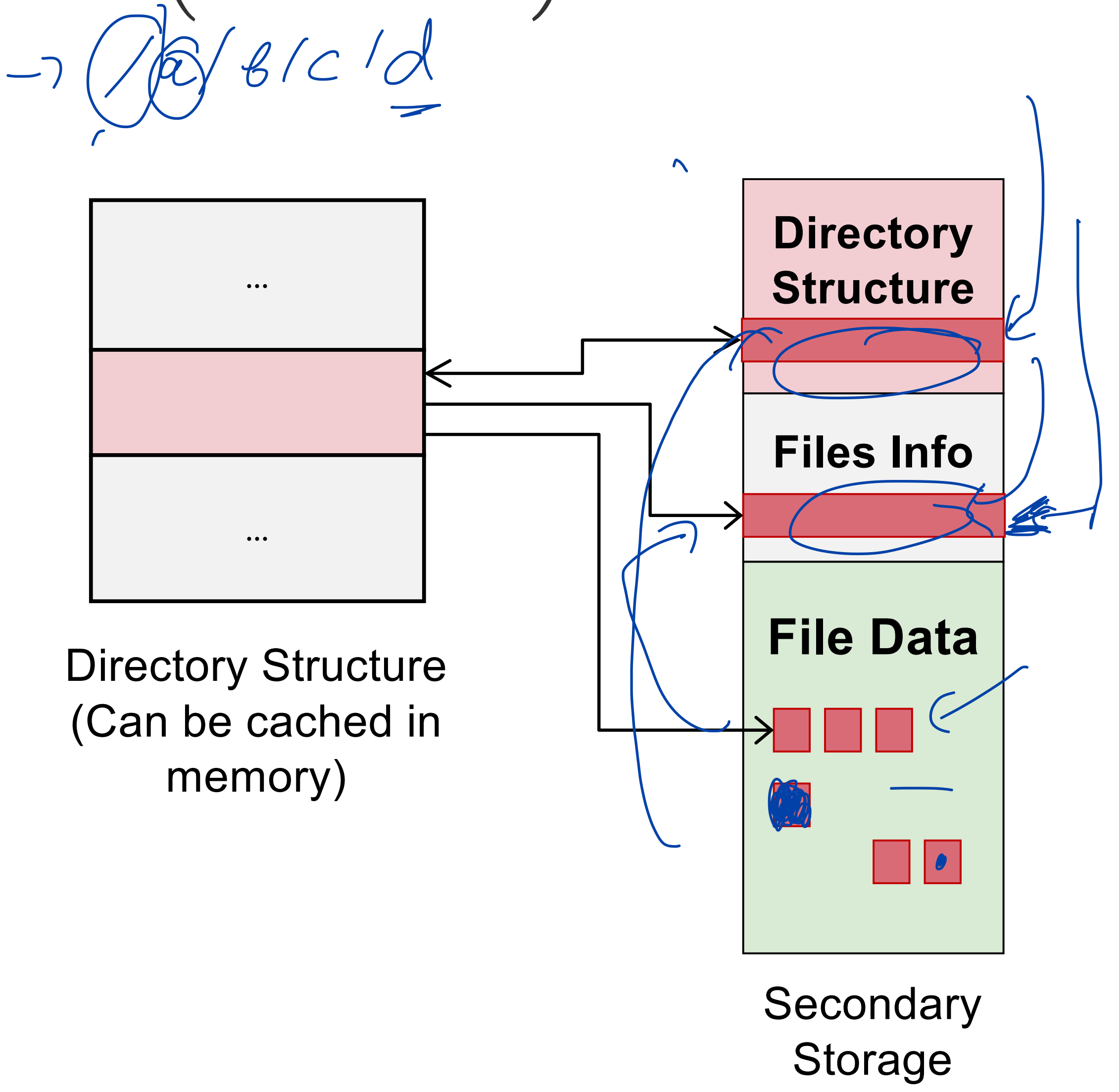
**Could you walk through the file creation diagram on slide34? I am unsure of the exact sequence of events that happen during file creation, e.g. do we access the directory structure first, then access file info, then access file data or?**

- The exact sequence depends on the implementation details of the concrete file system

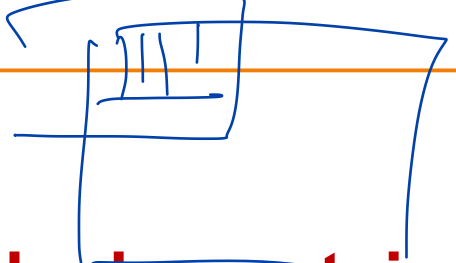
# File Creation: Illustration (L12 S34)

```
Create( filePath, …)
```

User Program

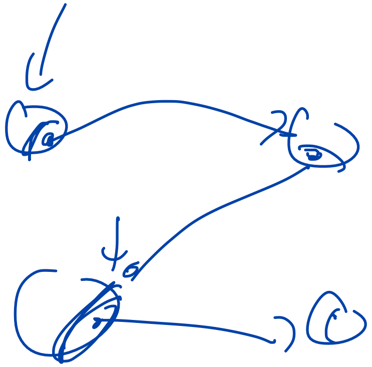


# Free Space Linked List



**In slide 24, could you explain "Each disk block contains a number of free disk block numbers"? Also, why is there high overhead here and what does the last line "Can be mitigated by storing the free block list in free blocks" mean**

- High overhead? Subjective judgement
  - It would take a few percent of otherwise useable storage
- Free space is not used → You can use free space to keep the information about the free space ☺
- Basically, embed the linked list into the free space



# Free Space Bitmap

**For free space management, if we use a bitmap data structure, isn’t this bitmap part of partition details? Shouldn't it be stored within the partition which is on the disk? Why is the bitmap stored in memory?**

- Storing it in memory allows you to quickly find and allocate a new block for a file.

# Miscellaneous

disk (block-based device)

**How does block- or object-based storage differ from the file-based solutions we've learnt so far?**

- What we have covered is a block-based storage based of files

flat system (no directories)

- Object-based storage is often used for in-memory storage
  - You want to store an object in other computer's DRAM
  - Example: many key-value stores (Memcached, Redis, etc)

# Miscellaneous

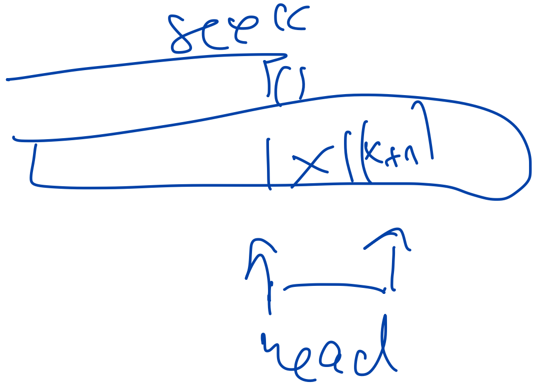
**Why not use a binary search tree or B-tree for storing file entries/information in a directory?**

- There’s a variety of data structures one can use
  - We’ve covered the most basic ones
- B-trees have been used for files
- Very large directories may benefit from BST-like organization

# Miscellaneous

**If a process has a cursor pointing to a location in file, e.g. record x, and the process reads record x. Upon reading the record, will cursor automatically move to the next record such that if the process reads the file again, it reads the next record?**

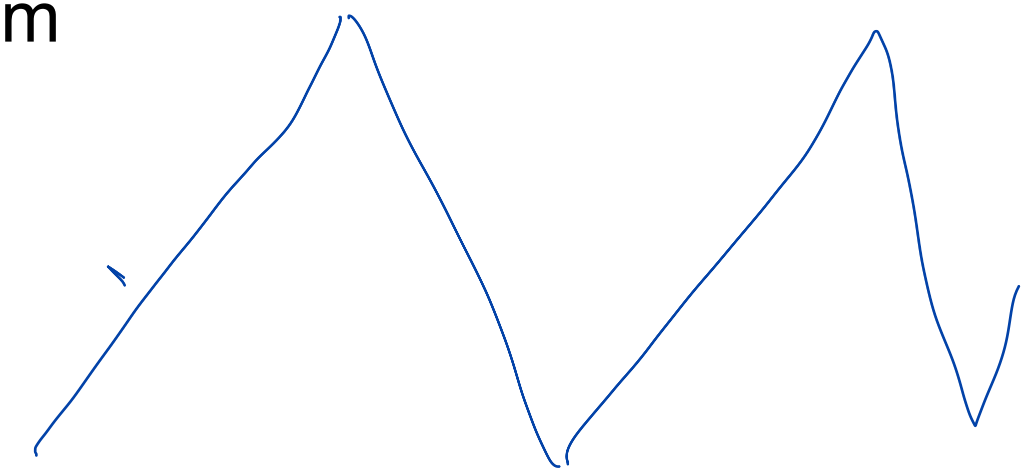
- Yes, read() will also advance the position of the cursor



# Disk Scheduling

**Hi prof, the scheduling order for C-Scan algorithm depends on the speed of disk head movement right?**

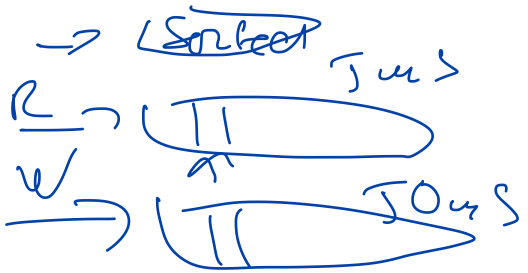
- In actual implementations it depends on both head movement speed and acceleration abilities, as well as the rotation speed
- We don't have the time to talk about physical devices in this course
- No need to know for the exam



# Can you recap new I/O Scheduling Algos?

optimal order

- **Deadline** - 3 queues for I/O requests:
  - “Sorted” queue –requests sorted by sector number (for speed)
  - Read FIFO - read requests stored chronologically
  - Write FIFO - write requests stored chronologically



- **noop (No-operation)** - no sorting
- **cfq (Completely Fair Queueing)** - time slice and per-process sorted queues
- **bfq (Budget Fair Queuing) (Multiqueue) -** fair sharing based on the number of sectors requested

# Follow-up Courses

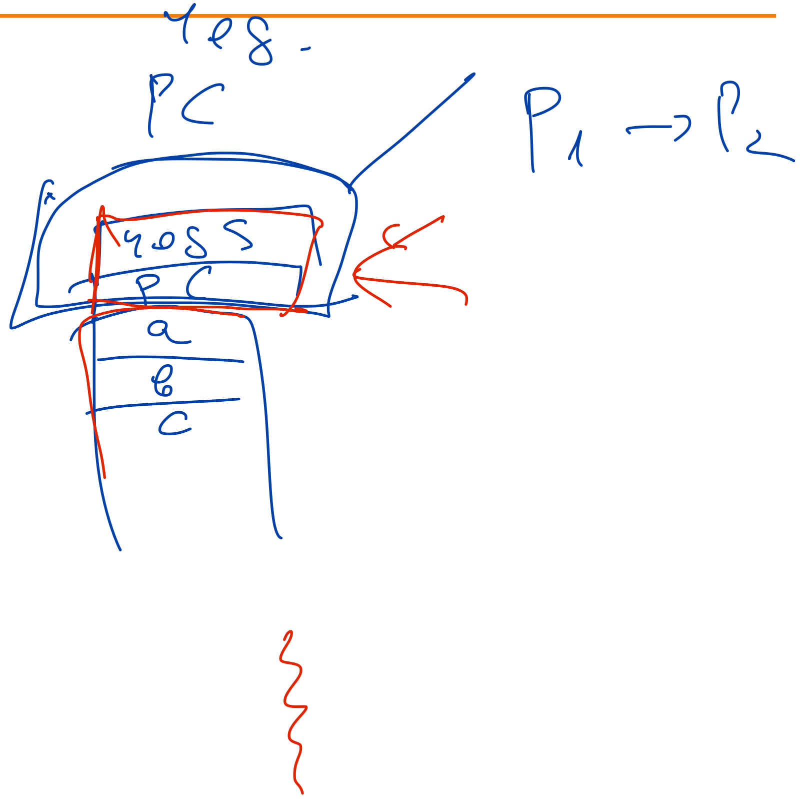
What are some good follow up modules to take if I enjoyed what was covered in this module?

- Advanced Operating Systems ✓
- Compiler design ✓
- Advanced Computer Architecture/Multicore Architectures
- System Security courses
- Distributed Systems
- Parallel Computing
- DNA-based storage

int a, b, c = 0;

a = b+c

-?



# Thank you!