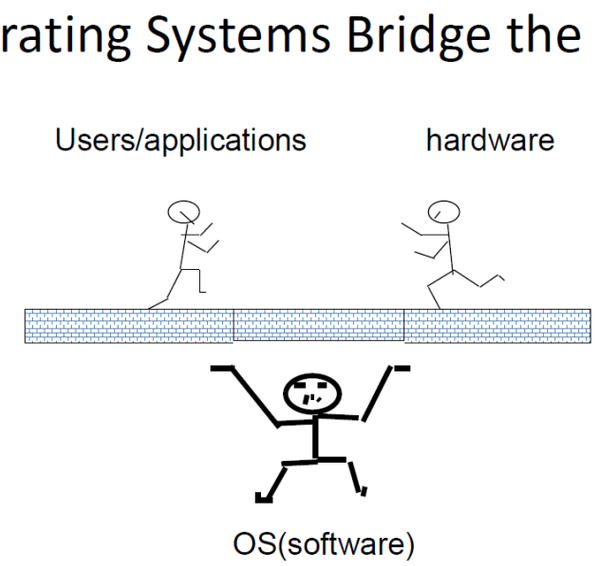
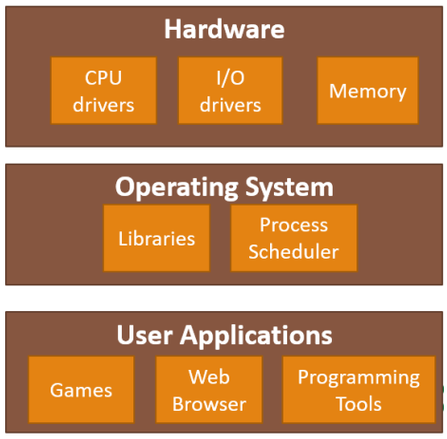
Background on Operating Systems



A computer system can be summarized into three components:

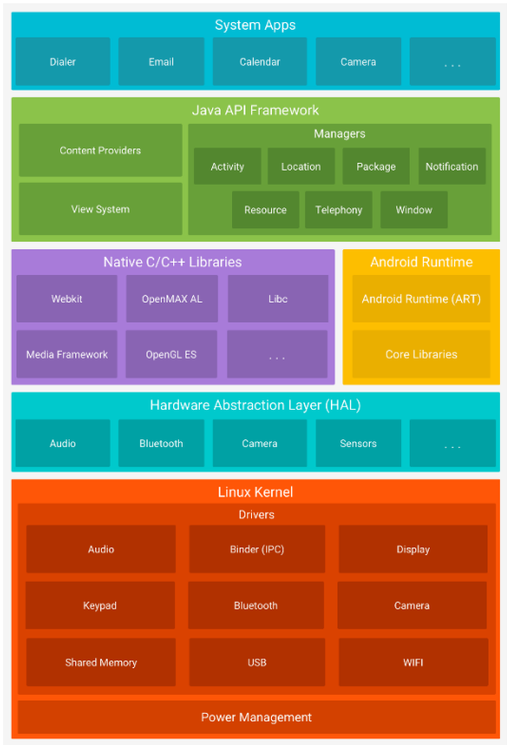
- Hardware components – physical components and drivers.
- OS – coordinates use of hardware with user applications
- Applications – typical applications used by a computer user



The primary tasks of an OS:

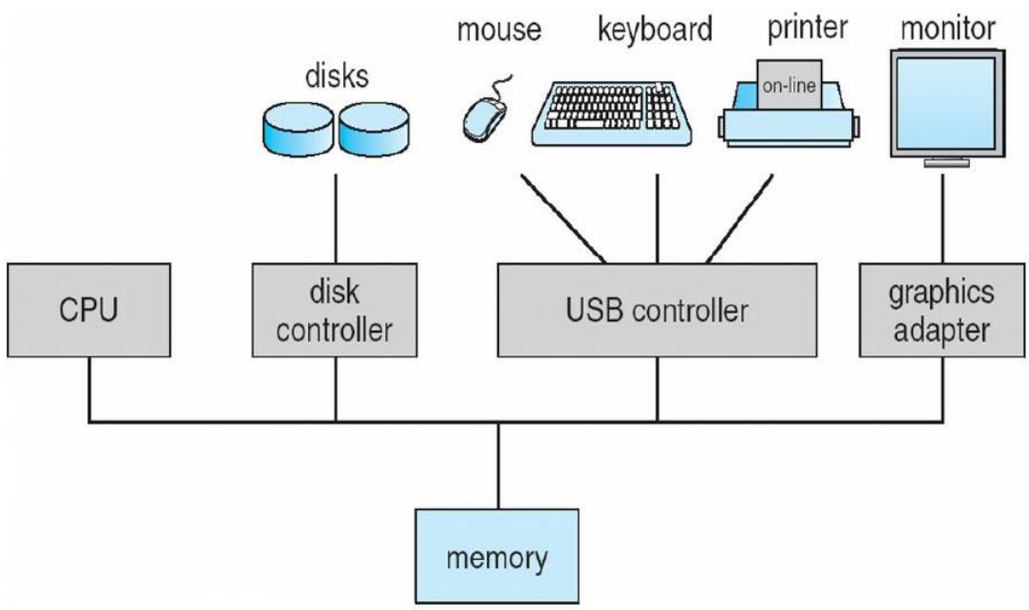
- A bridge/API to hardware resources.
- Resource allocator – allocates resources such as CPU and memory to an application. Decides how to share resources among other user programs.
- Control program – controls behavior of user programs to avoid improper use of computer resources (CPU and memory hoggers, programs with infinite loops, etc).
- Security - Must protect application's data from one another (unless designed to share the data). Allocate resources fairly and efficiently. Settles conflicting requests for resources. Prevent errors and improper use of the hardware
- Improvisation – find a way to improvise when hardware resources are scarce. Provides an illusion of dedicated machine with infinite memory and processors.

Architecture of android



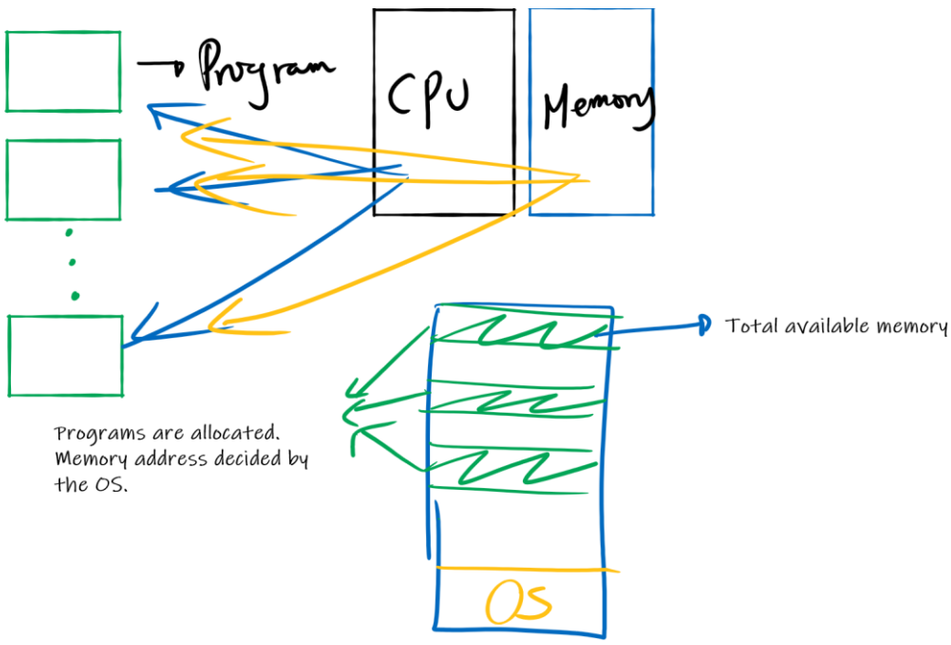
An example of primary tasks in action:

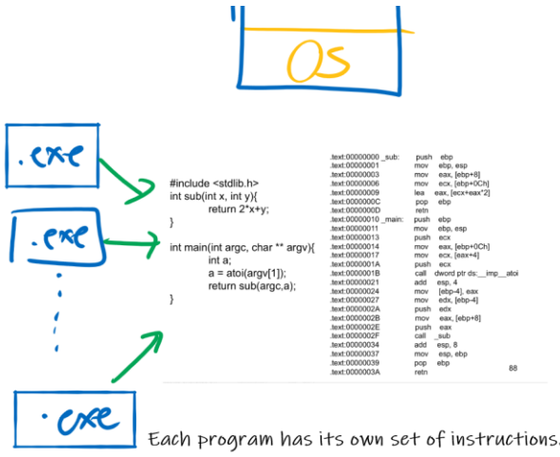
A bridge/API to hardware resources

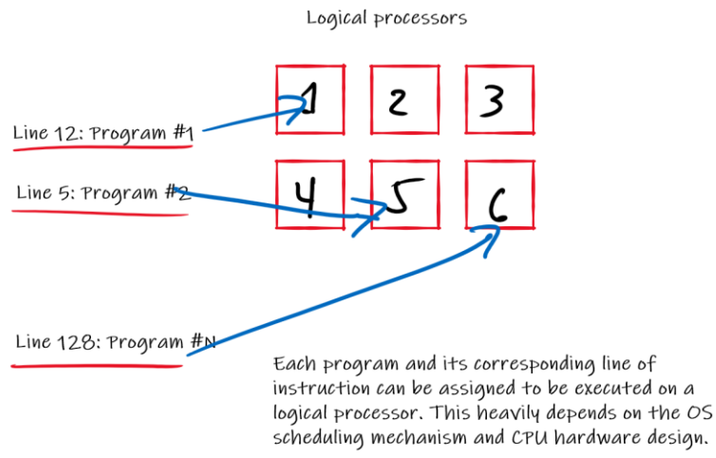


- How do we display something on-screen? → display interfacing, screen buffers, refresh rates.
- How do we access input devices? → getch(), keyin(), std::in API calls.
- How do we access storage? How do we organize our files in storage? → file directory systems implementation, user access permissions.

Resource allocator



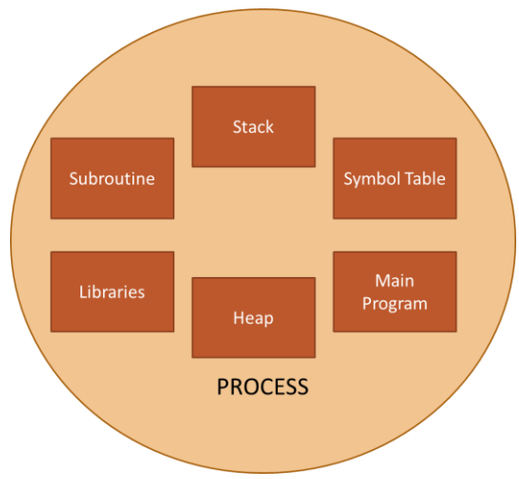




Control program
The OS ensures that all applications receive a fair number of resources. Each program has components where allocations are already pre-defined.

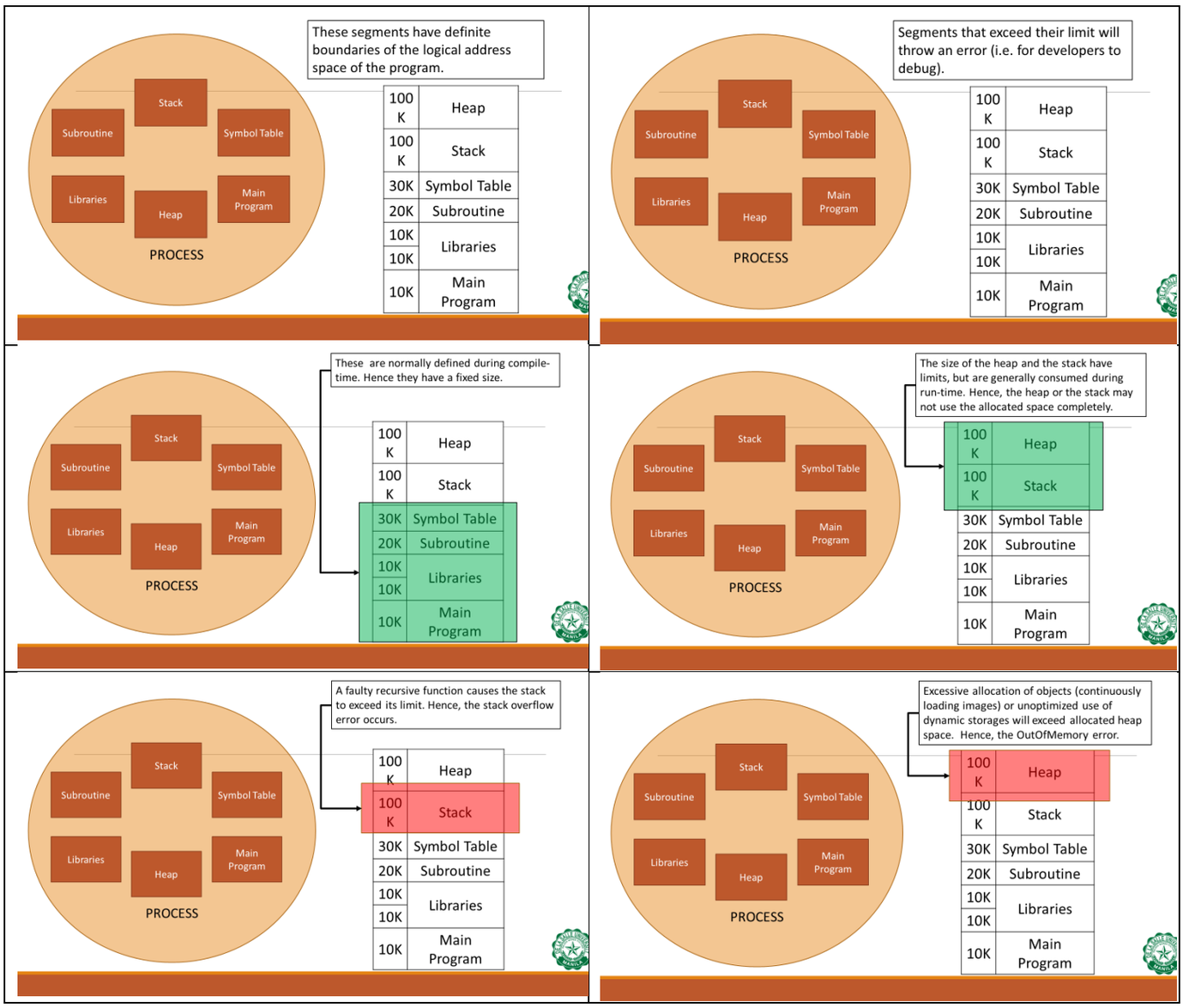
Components of a Program

- Main Program – points to the main function
- Subroutine – contains all functions declared in code
- Libraries – external functions or dependencies required
- Symbol table – holds variables
- Stack – holds temporary variables for function calls. Also stores the return address of a given function call.
- Heap – memory for dynamically allocated objects



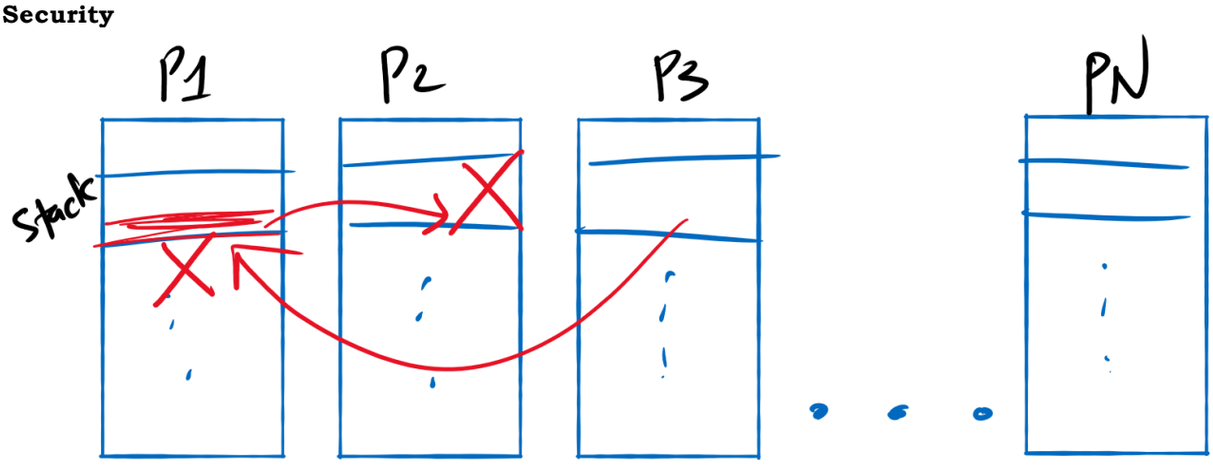
ACTIVITY:

1. Provide a scenario that causes stack to overflow.
2. Provide a scenario that causes the heap to overflow.

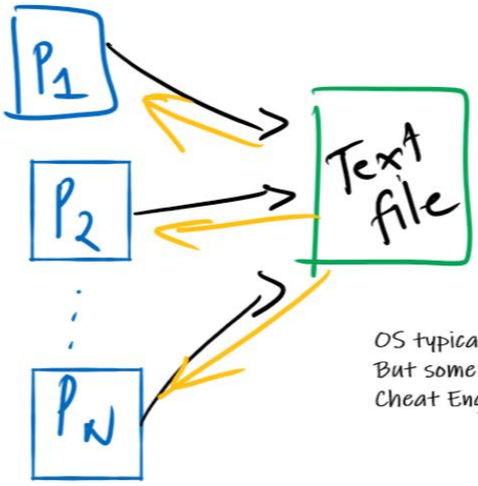


The OS pre-allocates memory on each application. Each application has a certain boundary on their components. Overstepping these boundaries throw errors.

Security

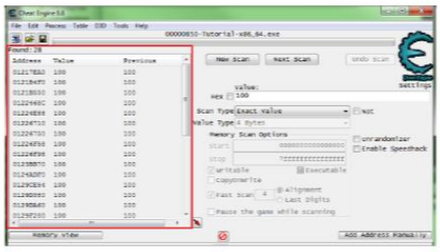


Each process has its own allocation and cannot be accessed by any other process.



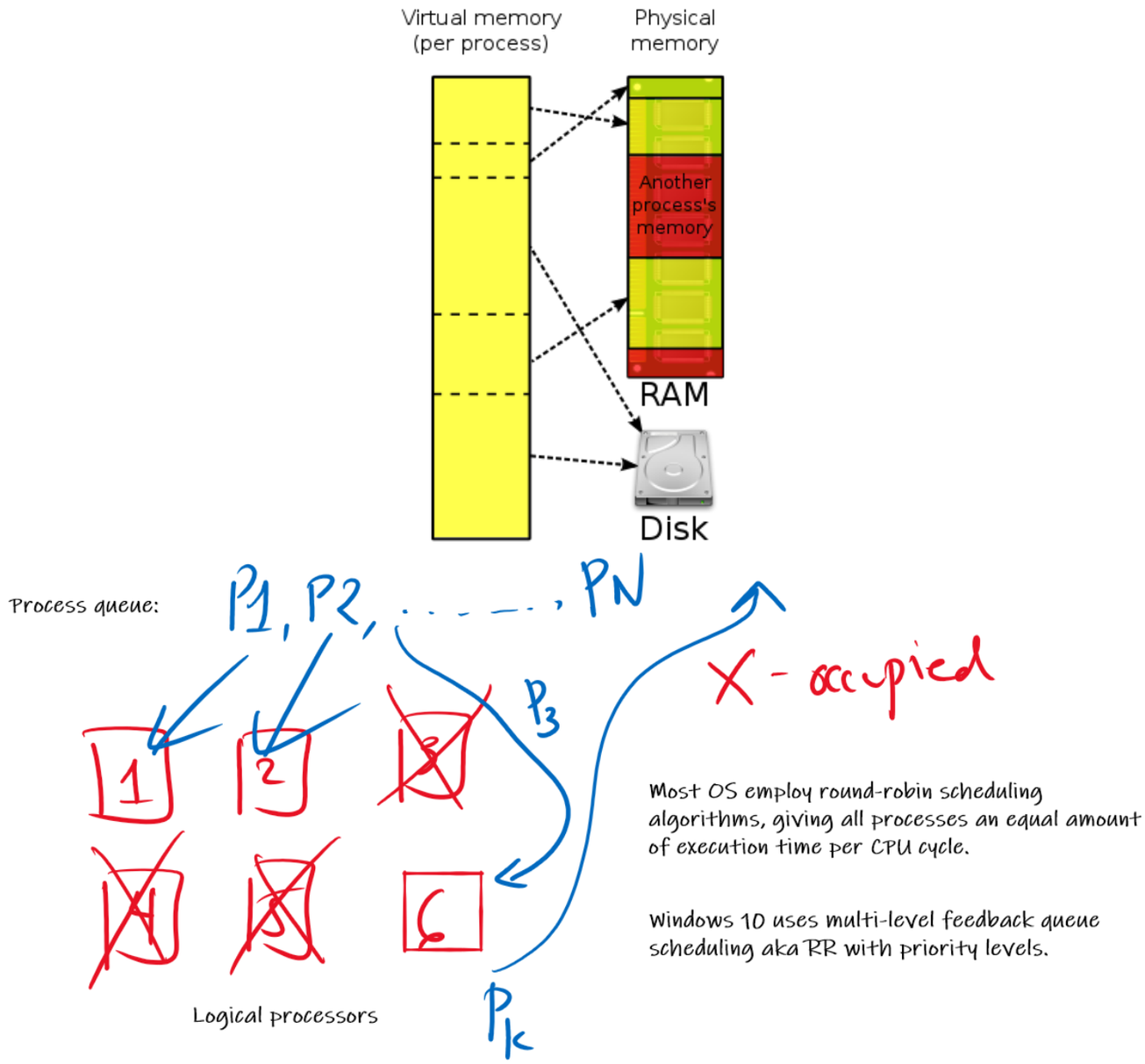
If processes need to share information, each process must have instructions on writing/ reading a shared file.

OS typically deny any malicious memory accesses. But some tools exist to circumvent this (e.g. Cheat Engine).



Improvisation

In memory management, all OS perform a technique called **virtual memory**, where the OS uses secondary storages (such as HDD/SDD) as main memory whenever RAM is used up. This gives the illusion of a very large main memory.



Why do we need to study operating systems?

- First, we can't create an OS on-par with most commercial OS nowadays → most mainstream OS (Windows/Mac/Linux) already have systems built on top of one another, some of which are older than us! (OS Dev Wiki – beginner mistake hard truth)
- An exposure to systems-level development, which may not be taught from other CS courses. Answers some of the questions like, "who creates and maintains my programming language?", or "who's behind and how does my program work with different OSes"
- Most of the algorithms in OS, can actually be applied to other applications. E.g. real-time and interactive applications. CPU scheduling → real-time scheduling and distribution of items in a game.
- Prerequisite knowledge for designing concurrent and distributed systems → E.g. Netflix video streaming. Different servers work together for you to watch your favorite movie in Netflix → video compression/decompression, video file indexing, media databases. All of these would require coordination algorithms, which are rooted in OS concepts.
- An opportunity to study C++! → It is a powerful language designed for systems development and game development, and you could do and emulate a lot of OS operations, than other languages.

Why is it challenging to study operating systems?

- If we try to create an OS from the ground up, all the syntax-checking, IDEs, and pre-built APIs are not available – we have to code each and every one of them. E.g., Your own stack function, memory allocation function, and threading function.
    - To make things a little bit easier for CSOPESY, we're going to use C++ and its associated system and threading libraries. 😊
- It is difficult to debug an OS as there could be many possible scenarios, combinations of which could be happening. A single test run will differ from another test run. E.g. different sequences of concurrent processes boot up at every run.
    - We also need to code our debugging tools – a memory visualizer, a meaningful format for our log messages, verbosity, etc.
- The program is purely text-based and algorithmic in nature
    - No punchy visualizations or UI. Challenging for developers who are looking for visual elements for evaluating their program.
    - Good UI/UX design should only come after the foundation of the OS is made. E.g., the kernel, the CLI, the scheduler, the memory allocator, etc.