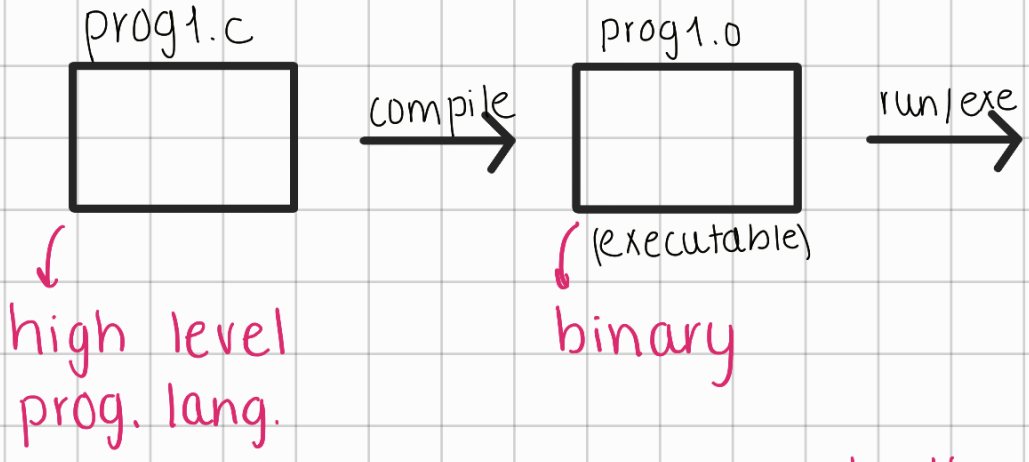
Lecture 2- Process Management.

Resource management
↳ process management
↳ memory management
↳ file system management
↳ I/O devices management
↳ security and protection

} OS responsible for managing these resources.

I- What is a process?

- the process is a **program** in execution.
↳ **program**: set of instructions in machine language



terminal: `>./prog1` → execute the program ↓ process.

Example
program: steps of making a cake.
process: CPU is the baker, he uses the ingredients (data), and follows the recipe (instructions) to make the cake

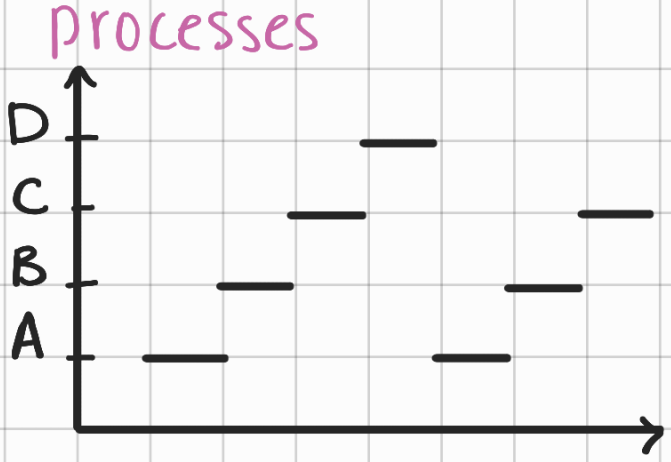
Actual Process Simplified

- When a **program** is run in a computer, its **instructions** are loaded into the **RAM**.
- The **CPU** **fetches** these **instructions** from **memory**, one at a time, **decodes** them, and then **executes** the **instructions**. (process)

Program → static
Process → dynamic

Multi programming mode (time-sharing model)

- execute several processes at the same time
- only one **CPU** → shared resource



(the time is very little, hence unnoticeable)

↳ it seems that 4 independent processes are happening in parallel (**pseudo-parallelism**)

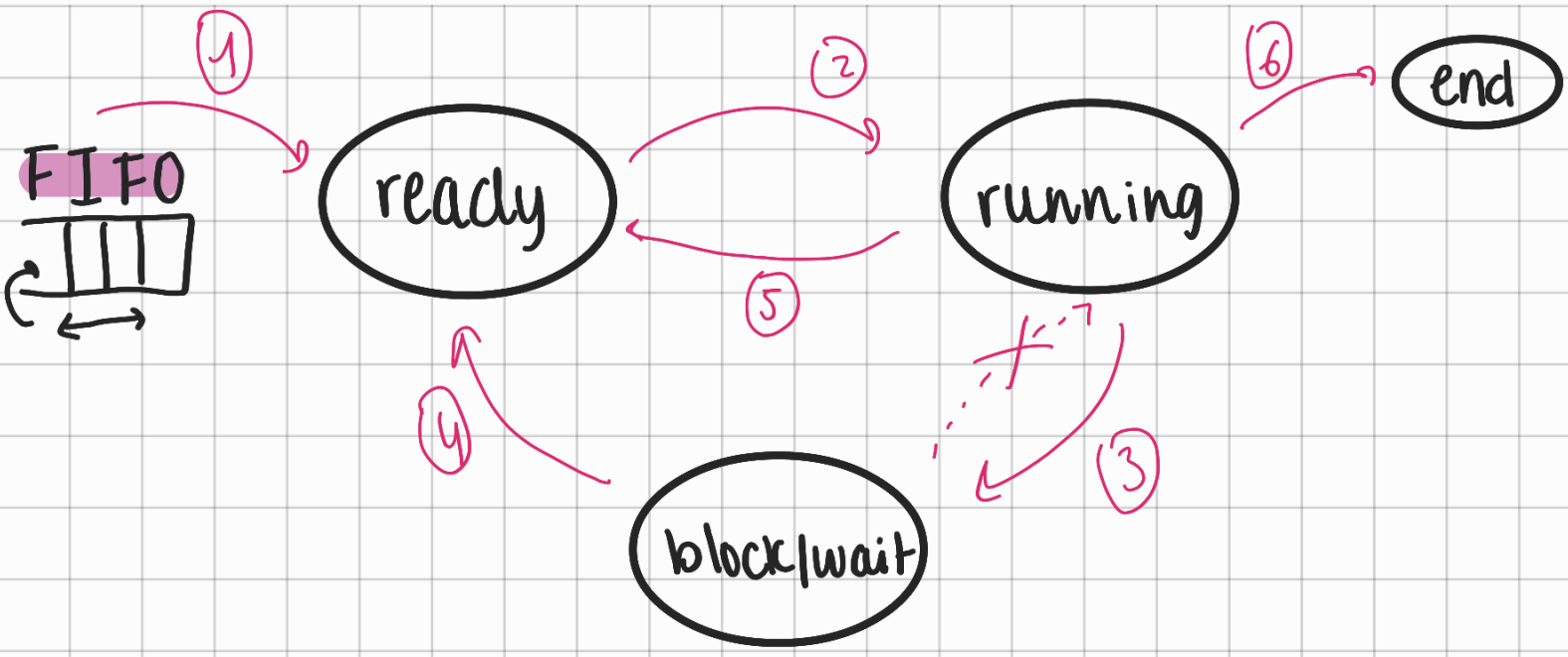
- each process has a programming instruction, input, output, state, set of registers.
- a single processor (CPU) may be shared among several processes
- some scheduling algorithim is used to determin when to terminate a process and serve anotherone

Each process has several attributes:

1- process id → pid (unique identifier)
2- state → to describe
3- priority
4- general purpose registers
   ↳ PC (program counter)
   ↳ IR for decode (instruction register)
   ↳ accumulator
5- list of opened files
6- list of opened devices (I/O devices)
7- protection

State of the process

- refers to the condition or status of a running program / process.
- each process can be in one of several states:
  ↳ **running**: the process is currently being executed by the CPU
  ↳ **ready**: the process was loaded into memory and is prepared to run but is waiting for its turn to execute on the CPU.
  ↳ **blocked / waiting**: the process is in a state of waiting, often because it's waiting for some event, such as user input from an I/O device.
  ↳ **terminated / exit**: the process has completed its execution and has been terminated. It may be waiting for the operating system to clean up resources (freeing memory, closing open files...) or for the parent process to collect exit status (determine out come of the execution



1 the process is created
2 _ the process gets to the CPU
_ the dispatcher elects the process to execute
3 the process makes I/O request so its blocked till the availability of the request
4 I/O request are now available
5 the process is not ended but its quantum time (slice of time) is expired or another process with higher priority arrives to ready queue (context switching)
6 the process is ended

FIFO: "First input, first out put" refers to a data processing principle in which the earliest data or request received is the first to be processed or served.

Types of process

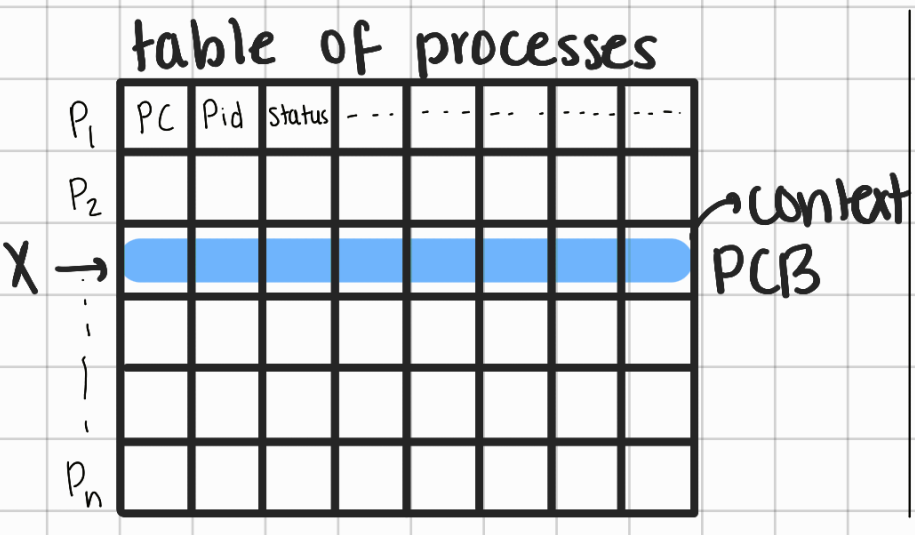
↳ CPU bound → it needs long time CPU (mathematical simulations)

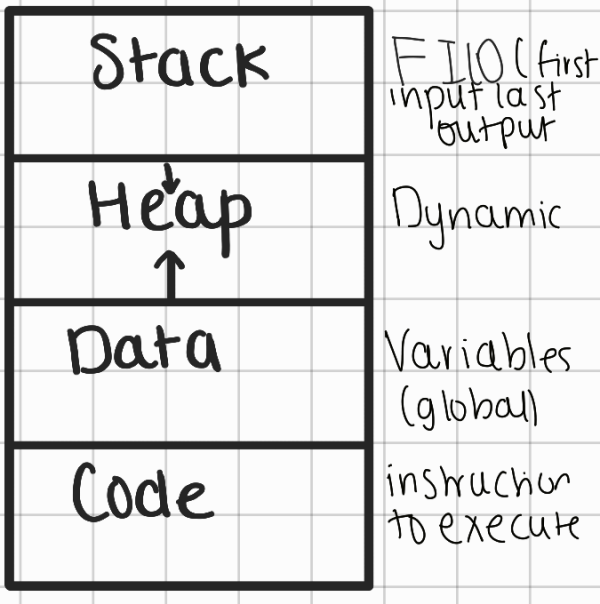
↳ I/O bound → it needs long time I/O (reading files)

Process Creation

PCB: Process Control Block, contains the attributes of each process (pid, state, priority, ...)

in simpler terms: PCB is like a file that the computer uses to keep track of everything it's doing for each running program, such as what it's working on, where it's saved, and how much time it has used so far.





- When a process is created, it is assigned a process control block (PCB), which is stored in the the table of processes located in the operating system's memory. The PCB contains essential info. about the process, including its process identifier (PID), status, priority, and pointers to the memory segments associated with this process, (Stack, Heap, Data, Code)

How Process is Created

1- In UNIX, processes are created using a mechanism called "fork"

2- A process (referred to as the "parent") can create a new process (the "child") by duplicating itself. This duplication includes copying the current state of the parent process such as its code, data, and file descriptors.

3- When you start a UNIX-based system, the first process created has a process identifier (PID) of 1 and is known as "init". Init is responsible for initializing the system and starting other processes.

- The "fork" function is a system call provided by the "unistd.h" library in UNIX. When you call "fork()" in a program, it creates a new child process that is essentially a copy of the current (parent) process. The child process starts executing from the same point in the code as the parent process.