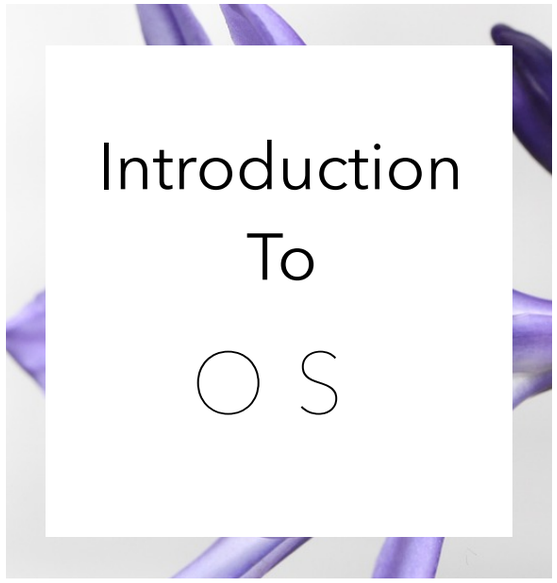


50.005 CSE

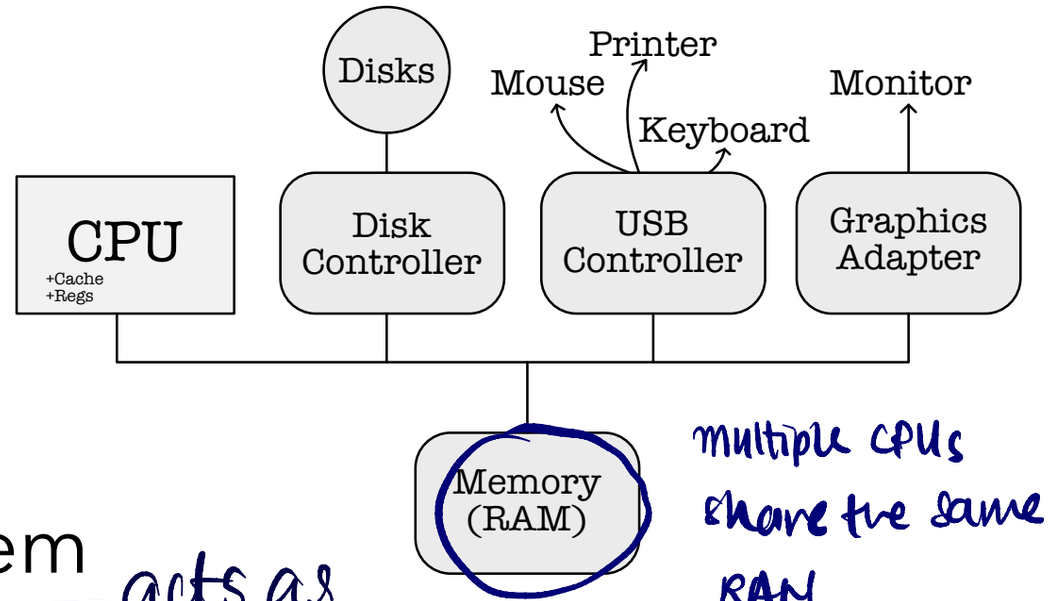
Natalie Agus
Information Systems Technology and Design
SUTD

WHAT IS AN OS?

made of codes

One of the four parts of computer system:

- Hardware
- Operating System
- Programs
- User



multiple CPUs share the same RAM → OS needs to manage them

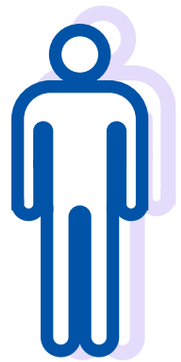
acts as intermediary, so that your computer is:

- ✓ convenient to use
- ✓ efficient

DUAL MODE

dual mode is possible if it is supported by hardware

↳ control unit prevents JMP when PC31 is 1



User

Restricted, for security

Kernel

More privileges, access to hardwares like memory location

uninterruptible

System call

↓

want access to hardwares

WHAT IS AN OS **FOR**?

Functions of the OS:

- Resource allocator
- Control program executions

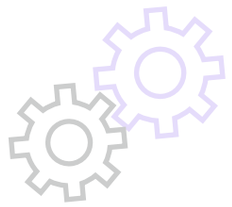
allocate memory / cpu time
manage conflicting requests

↳ prevents illegal access / improper use of the hardware

The heart of the OS is the **Kernel**

↳ running at all times

HOW DOES OS RUN?



1. Computer ON

Loads BIOS

2.BIOS Scans

called "firmware", the piece of code that does this scanning

Prep all attached devices to CPU in a state for OS

3. BIOS Loads MBR

Master Boot Record (disk address 0), contains info about disk partition as well as bootstrap code. The bootstrap code points to the bootloader

3. MBR Loads Bootloader

Bootloader is at bootsector (usually the start of the disk partition). The bootloader points to the entry codes of the OS. If there's multiple OS, then bootloader will ask user which OS to start

4. OS starts

OS bootstraps and starts

PURPOSES OF AN OS

Controls and coordinate use of hardware among various applications (programs) and users

1. Hardware & I/O control

WHAT ARE I/O DEVICES?

Input:

- Graphic tables
- Cameras
- Barcode reader
- Gamepad
- Joystick
- Mouse
- Keyboard
- Microphone
- Scanner
- Touchpads
- Pen input

Output:

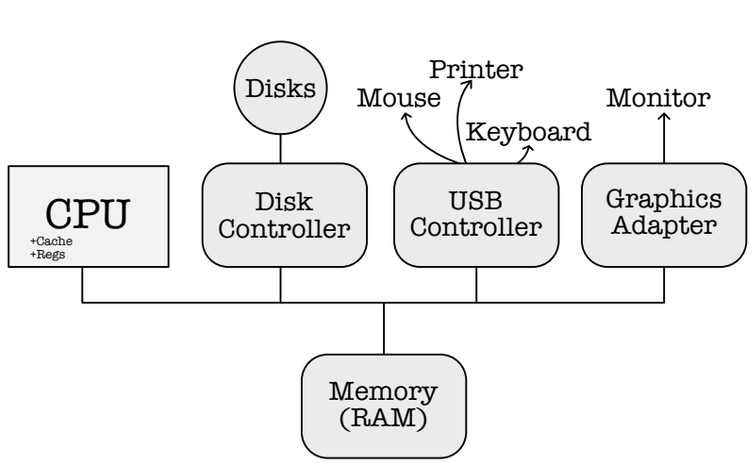
- Monitor
- Printer
- Projector
- Speaker
- Headphones
- Monitors

Both:

- Modems
- Network cards
- Touch screen
- FAX
- Sound card

HOW I/O DEVICES WORK

- CPU, I/O device controllers, and I/O devices act **independently** of one another
- Each I/O device has a **controller** (whats why you install *drivers*) that comes with a ***local buffer***
- I/O devices and controllers can run code an starts activity on its own too



(A) CPU wants to move data to/from device controller buffer from/to memory (RAM)

(B) I/O happens when data is moved from/to device controller buffer to/from device

We need **coordination** to do step (A) and (B) above: with *interrupts*

OS does this

HOW I/O DEVICES WORK

these are within your PC / laptop

I/O external device

notify Kernel by interrupt

request. Once request granted, transfer to RAM

transfer

CPU

USB/ Device Controller

local buffer

I/O Device

device buffer

General purpose computer **comes with device controllers.** It has its own registers and storage units, and can run simple code **independently**.

External IO device connected to your computer. Runs **independently.**

Upon I/O interrupt, CPU tends to I/O request

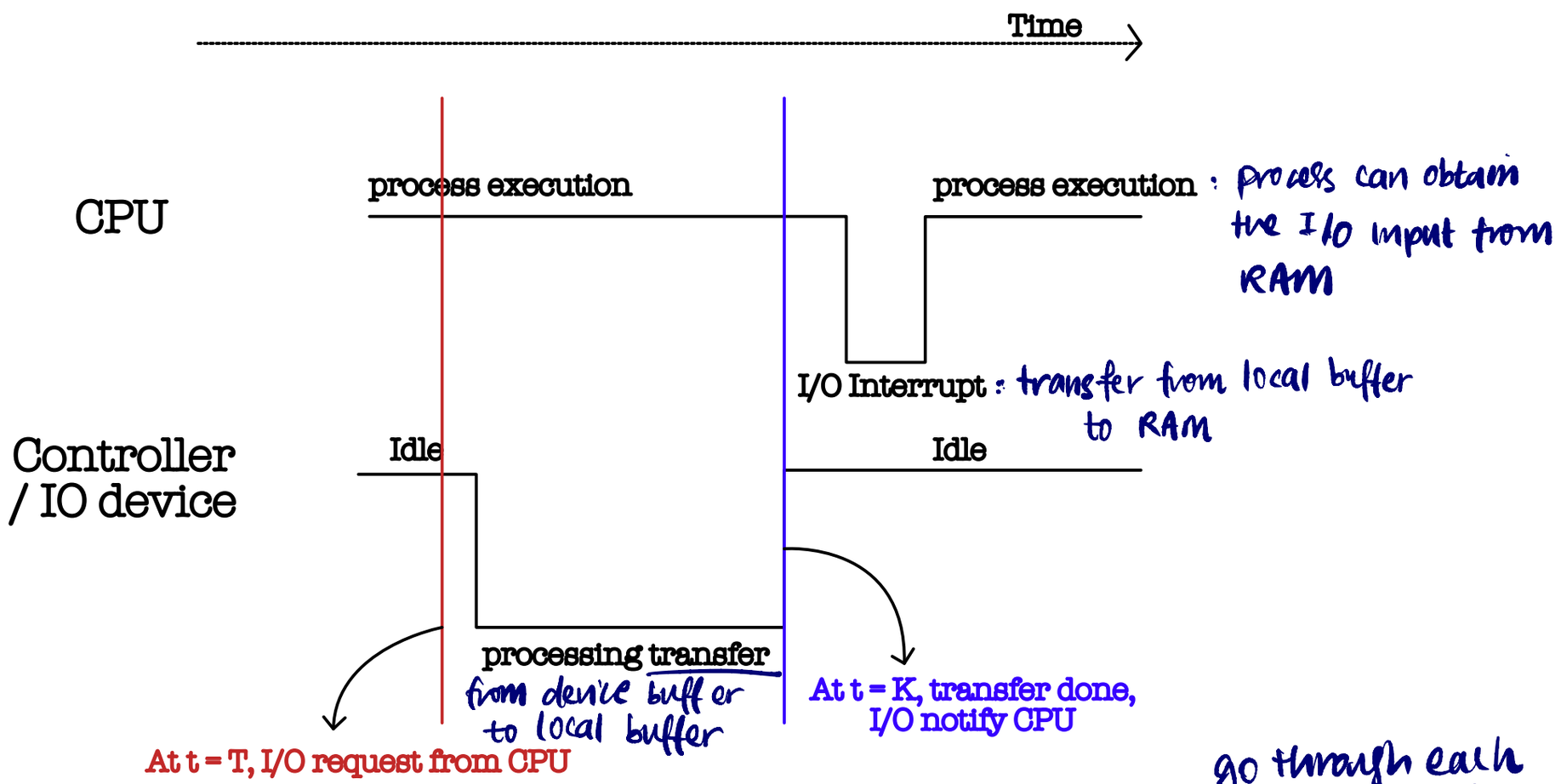
RAM

Save to disk if RAM is full of requested by app

Disk controller + Disk

if not in kernel mode, you can interrupt an interrupt if you have higher PRIORITY
However this comes at a cost of context switch

PURPOSES OF AN OS

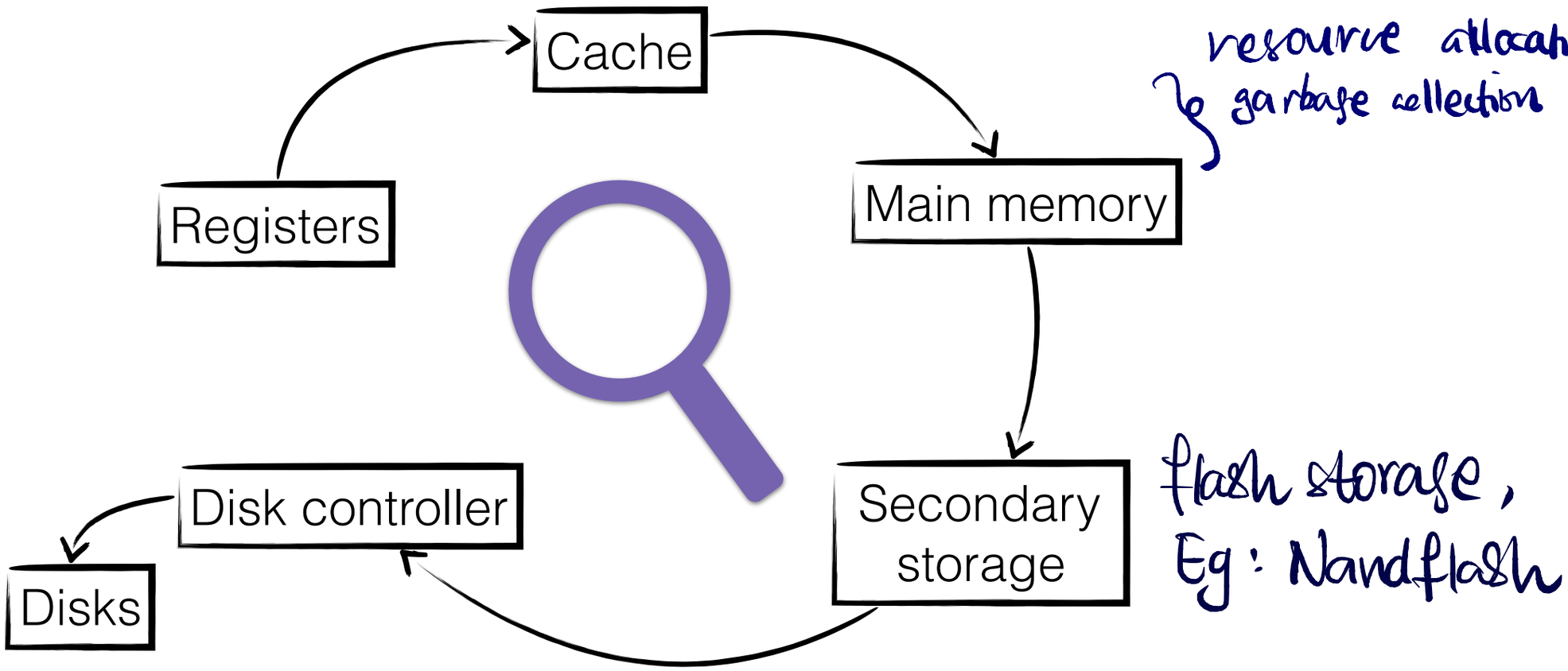


2. Handles interrupts: vectored and polling

upon interrupt, will look at interrupt vectors to see which device made the interrupt

go through each device and check who made the request

PURPOSES OF AN OS



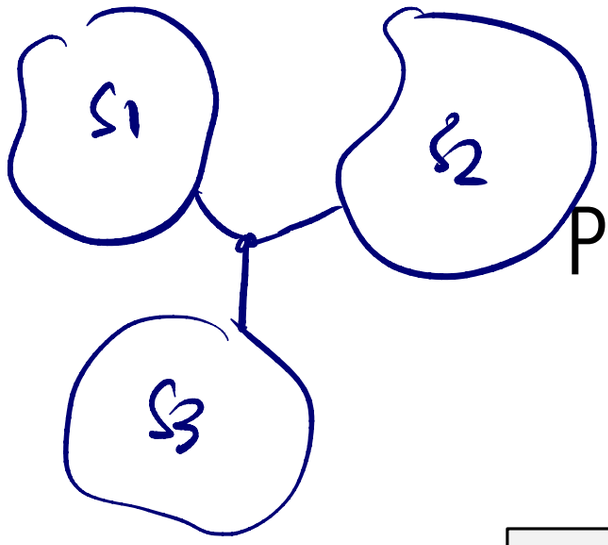
resource allocation
garbage collection

flash storage,
Eg: Nandflash

3. Managing the Storage Hierarchy

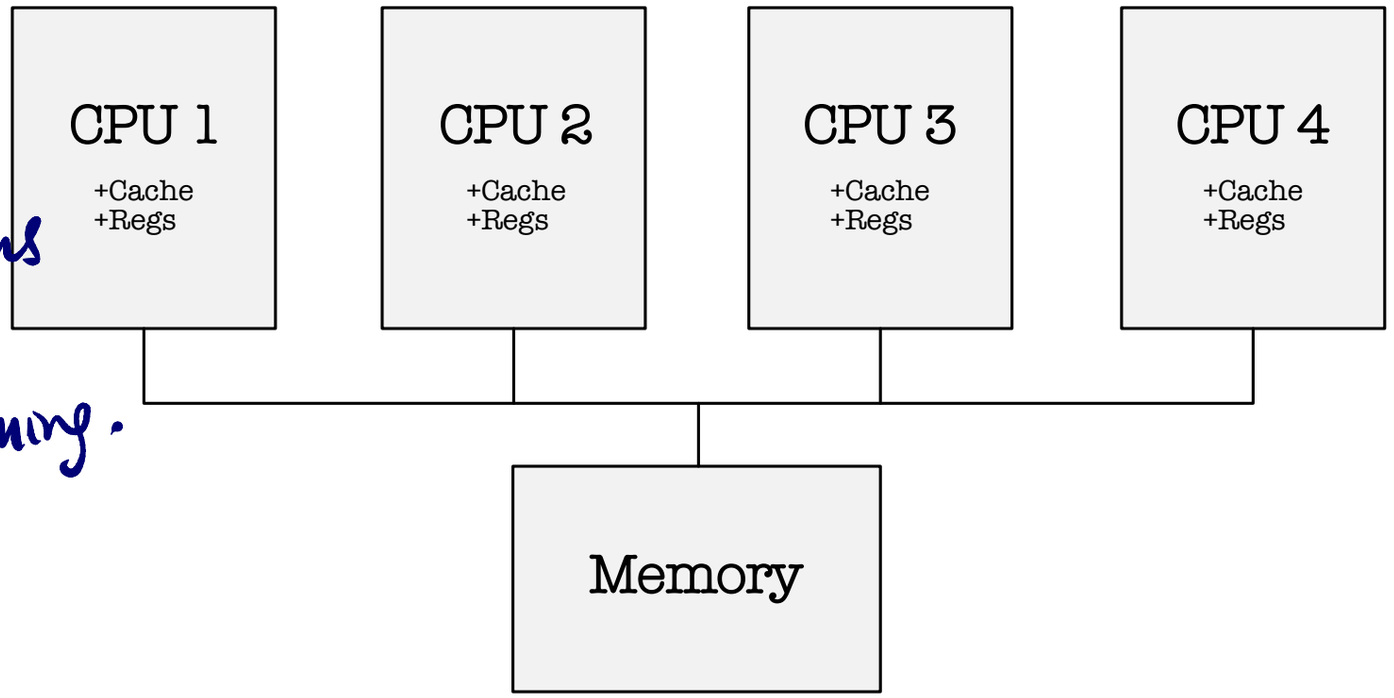
so that data integrity and consistency is guaranteed.

clustered system



PURPOSES OF AN OS

- ✓ high service availability
- ✓ must write programs that utilizes parallel programming.



4. Multiprogramming: process management (Scheduling & context switch)

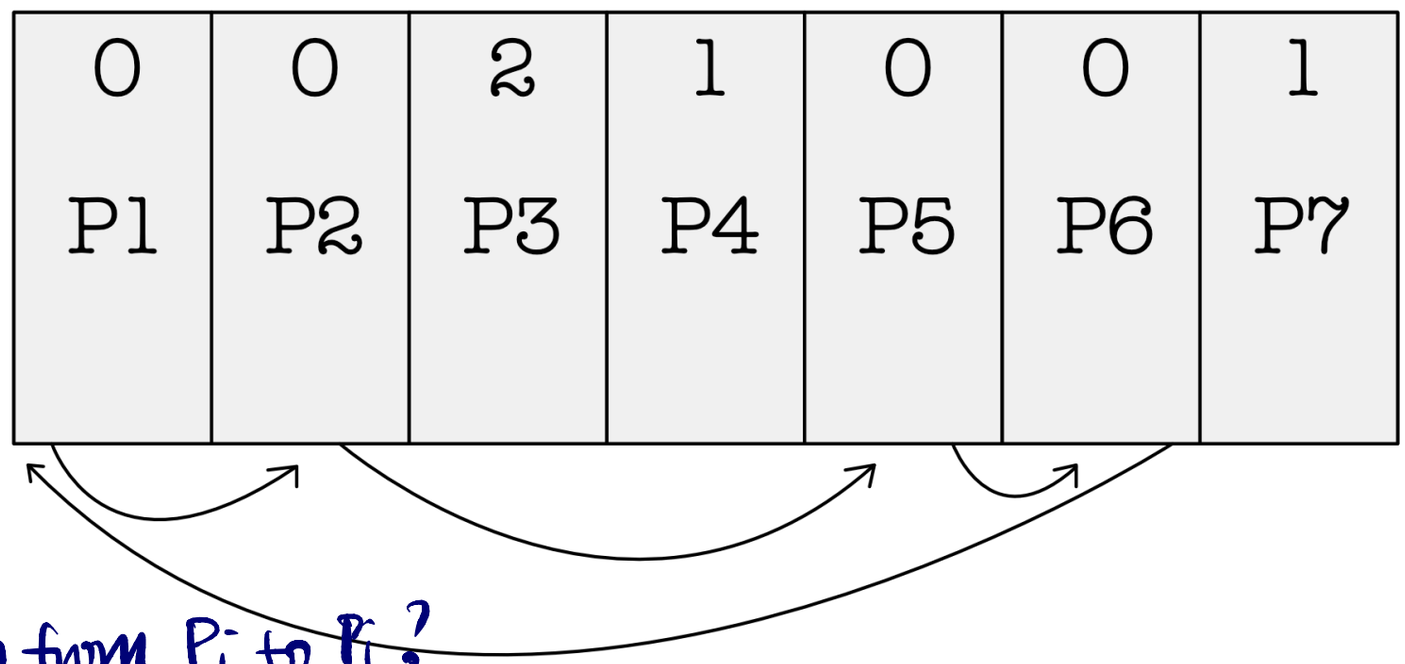
WHY MULTIPROGRAMMING?

Rules for managing multiple processes:

① a single (user) program cannot keep CPU busy at all times

② CPU always have something to do

↓ so that it gives the illusion that your machine can multitask = being efficient



kernel owns Process table

↳ it might be on cache / RAM / disk

"swap space"

when do you switch from P_i to P_j?

↳ if P_i is waiting for I/O or

↳ the turn for P_i is up

Context switch

① save all states (Regs, cache, stack, etc) of P_i

② Load all states of P_j

③ Resume P_j

MULTIPROGRAMMING

WHY

1. **Be efficient in organizing / scheduling jobs or data, since CPU can only execute 1 instruction per clock cycle**

2. **Allows timesharing: context switch so rapidly so that users still see it as interactive computing**

Note: but not "too rapid" either because context switch is an OVERHEAD.
CPU cannot do actual work when doing context switch (meaningful)

HOW

1. **Response time is fast enough**

2. **Always have at least 1 program active at any time**

3. **If RAM is full, swap with disk**

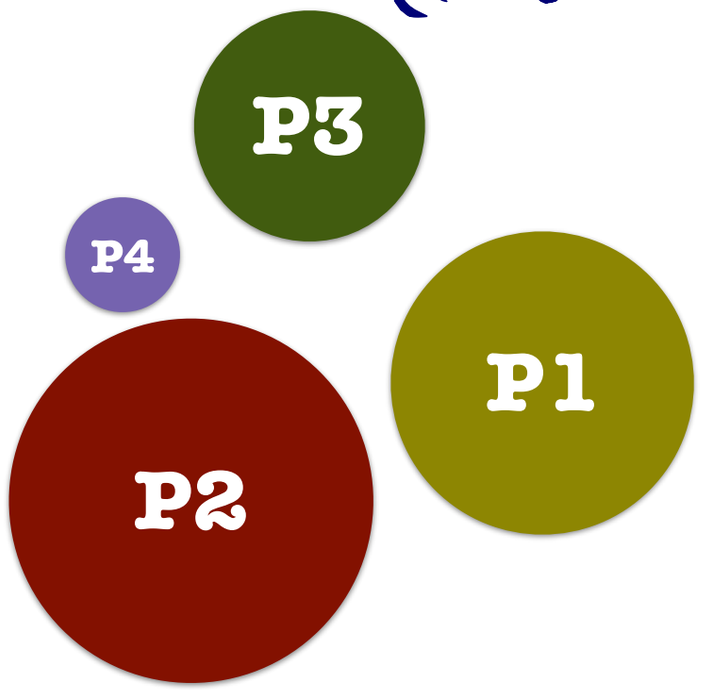
Program : passive entity, code/data on your disk

VS

more info on → PROCESS MANAGEMENT

↓ active entity

(Program in execution)



1. Create/delete user & kernel processes (week 2)
2. Schedule & sync process comms
3. Manage threads → (week 3)
4. Deadlock handling → (week 5)
5. Garbage collector (free resources) → RAM & cache space

Kernel's process manager

System has many processes, **multiplexed**

takes turn to run

Note on some keywords:

① Concurrency: multiplex process executions so it has the illusion that they run at the same time

② parallelism: processes executed at the same time at different CPUs (multicore system)

Summary

1. what is an OS? software that manages hardwares
2. Dual mode of OS → user, → kernel
3. Purpose of OS :
   1. → resource allocator → hardware & I/O control, handle interrupts, manage storage hierarchy
   2. → control program execution → process management (scheduling & context switch), security