can you read this?

CS 537

"operating systems"

2 lectures

- -> Intro
- -> Real info: process, CPU virtualization

why study OS?

=> security

=> apps / portability

=> reliability

=> file / program mgmt

=> how computers work

=> interesting

=> but, there's so much more

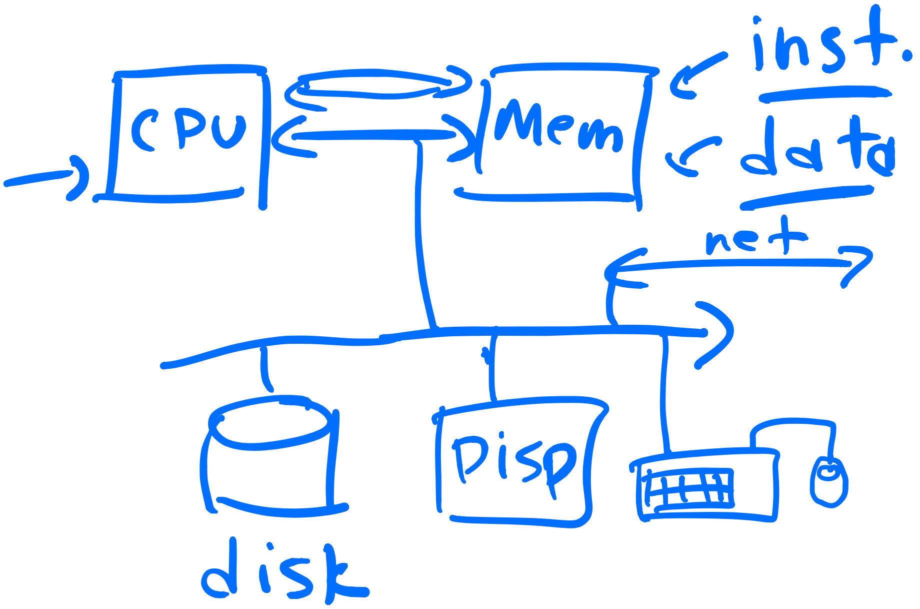
=> had to take it

Background:

→ CS 367/400 : Program

→ CS 354 :

→ how computer works



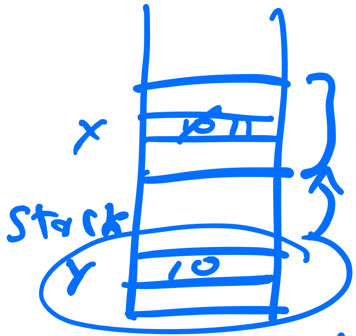
fetch ←
decode ←
execute ←

low-level → C prog. level

→ know C (somewhat)

```
void inc(int x){
    x=x+1;
}
```

```
int y=10;
inc(y);
```



→ ? ⇒ why unchanged

should know:

- code
- heap
- stack

OS : what is it?

=> Remzi

Arpaci-Dusseau

Course overview

Virtualization

{ (one) Physical => many virtual }

-> CPU -> Memory

Illusion

running program:

{ => own CPU
=> own private memory }

Key aspects:

-> **efficient**
-> **secure** (**restricted**)

CPU virtualization :

1 CPUs => many v. CPUs

time sharing :

A | B | C | A | ...

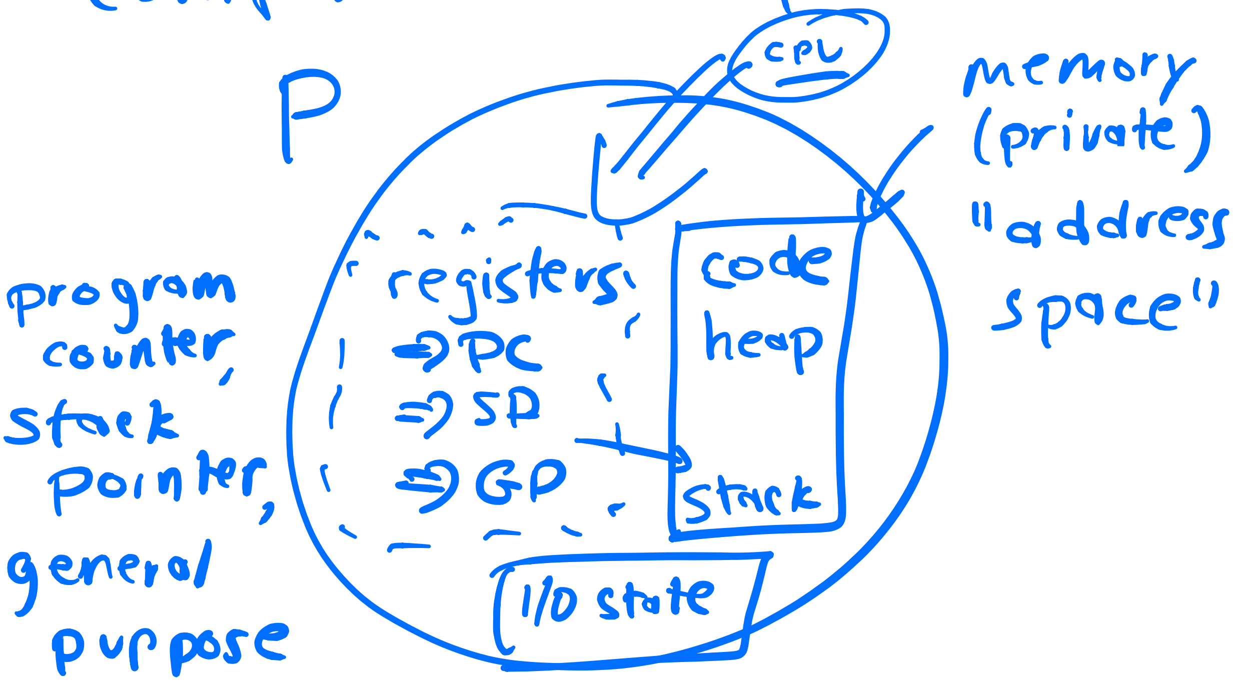
(vs. space sharing)

[multiprogramming]

Abstraction : Process

~ running program

components of a process:



Policies: Higher level decision

Mechanisms : How

Core mechanism:

Limited Direct Execution

Limited – security (protection)

Direct Execution – efficiency

CPUs : fast

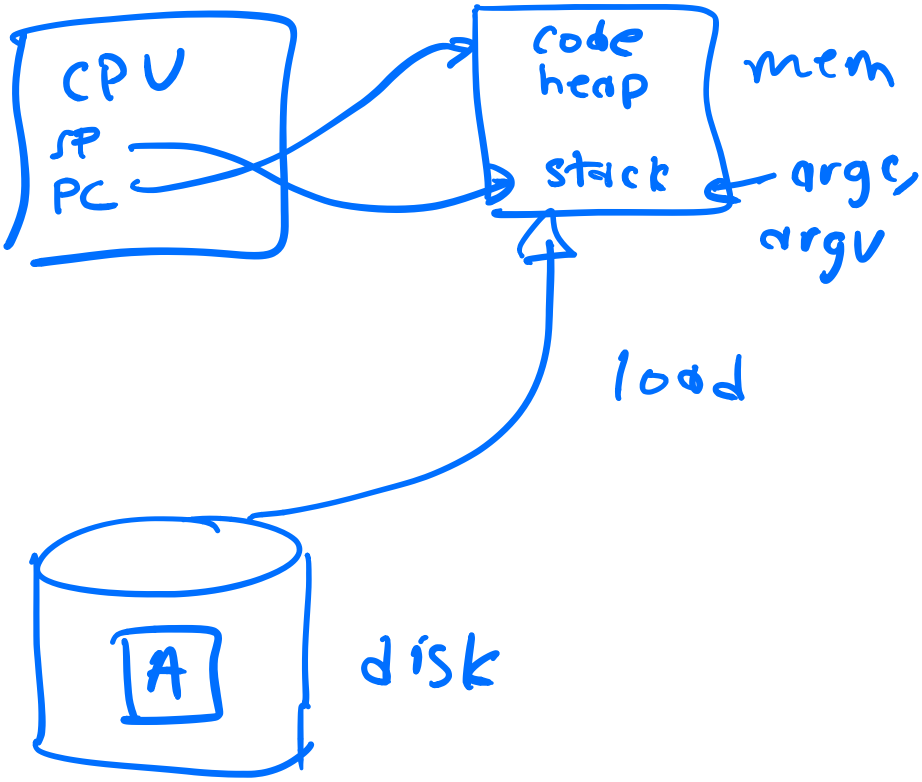
mostly, run directly on CPU (hardware)

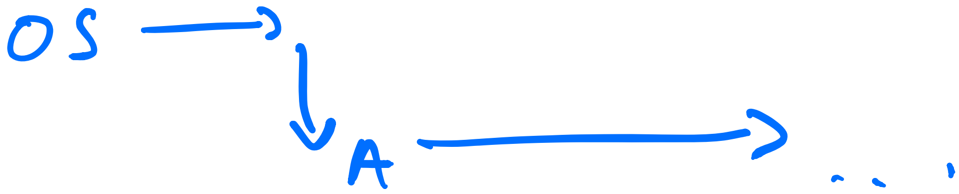
(not limited)

Direct Execution :

=) OS : first prog to run

want : run prog "A"





Problems:

=> what if "A" (user process) wants to do something **restricted**?

=) what if OS wants to stop "A", run "B"?
(OS : how to regain control?)

=) what if "A" does something that is slow? (disk I/O)

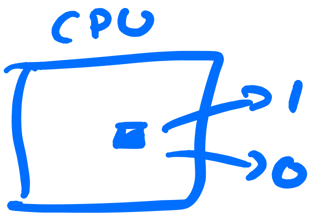
Problem #: restricted ops
(in controlled way)

mode : per CPU bit

=) OS "kernel mode"
OS can do anything

=) user program "user mode"
can only do limited # of things

how to get into these modes?
how to transition?



@ boot time:
boot in kernel mode

want to run user prog:
=> special inst that
both 1) transitions into user mode
2) jumps to some location in user program

user prog: wants to do something restricted (disk I/O)
1) => kernel mode
2) jumps into kernel

[illegible]

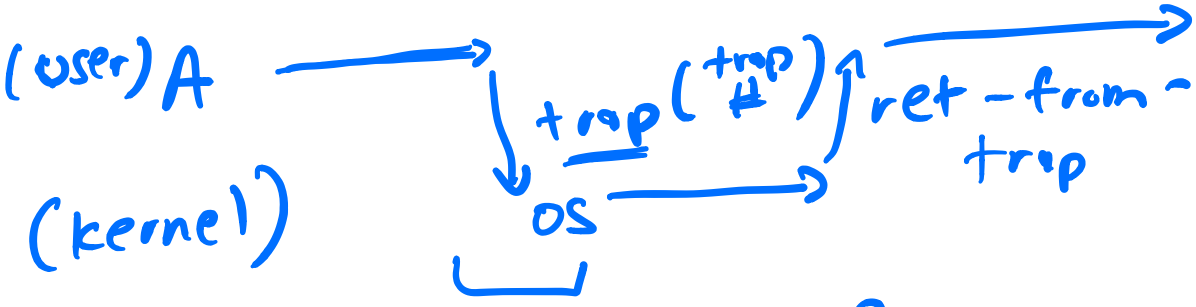
(but restricted jump)

two instructions:

trap / return from trap.

→ jump into kernel
(but @ restricted location)
elevate "privilege" (user → kernel)

save enough register state so that we can return properly



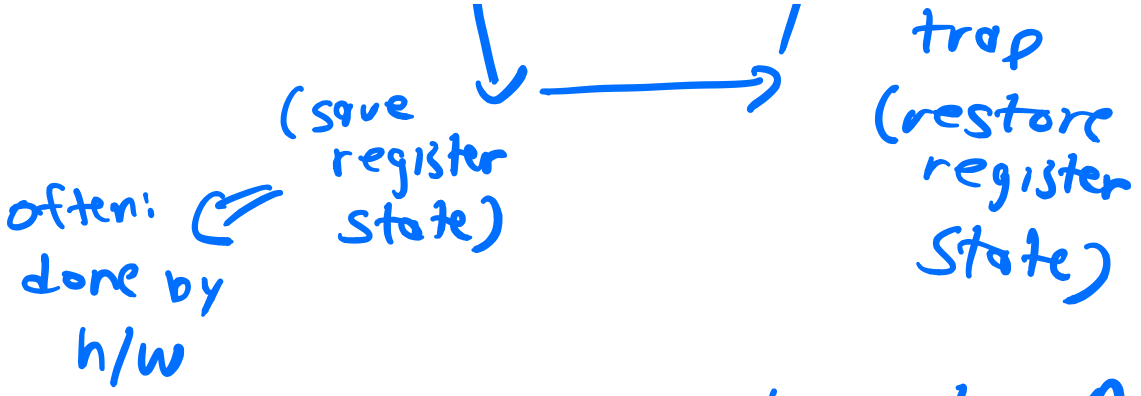
(system calls)

@ boot time: OS

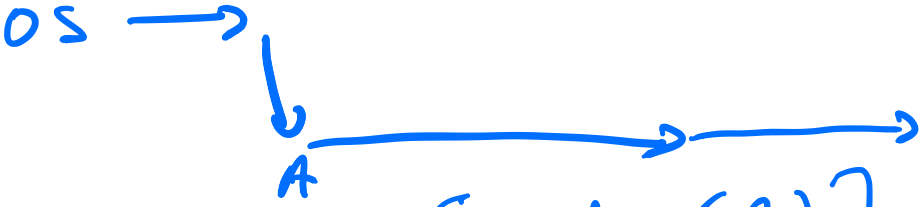
⇒ kernel mode

⇒ set up trap handlers
(issuing special instruction:
tell H/w where
trap handlers
are in OS memory)

save/restore "state" of
(register)
process:



Problem #2 : How to stop A, run B ?



```
while (1)
    ;
```

cooperative :

⟹ hope that A doesn't do bad stuff

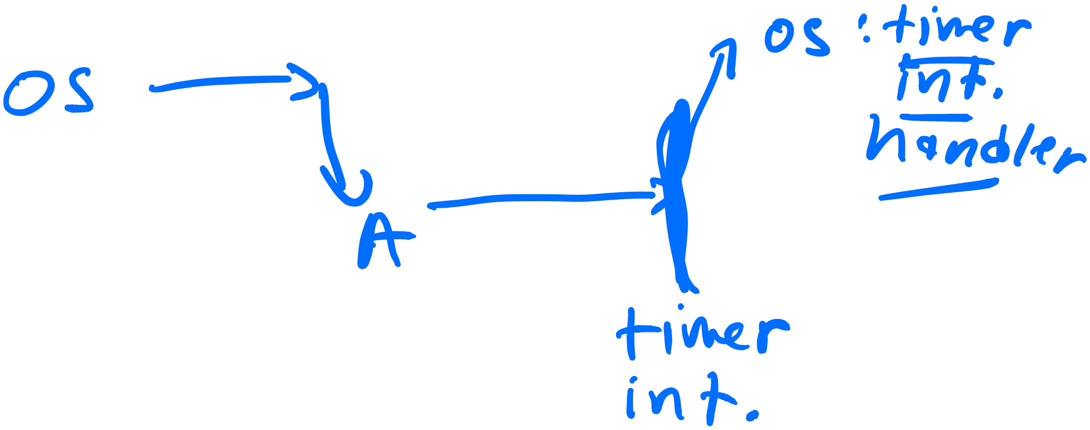
non-cooperative : (**preemptive**)

based on h/w support:

timer interrupt

@ boot : OS

- kernel mode
- installs trap handlers
- start interrupt timer


OS
A
timer int.
OS : timer int. handler