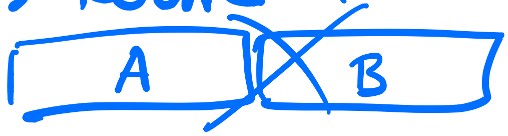
Today

537 : Part 3

⇒ CPU virtualization
- → mechanisms
  (Limited Direct Execution)
  LDE
- → policies : scheduler
  which process should run?

last time:
- → Shortest Job First (SJF)
  (assumes knowledge of run time)
- → Round Robin :



not RR

ABAB ....

time slice (quantum)
=> responsive (interactive)

Today: develop real sched policy => [classic Unix scheduler]
=> Multi-level Feedback Queue (MLFQ)

Later today: [Virtual Memory]

Later later today: Project 2a
(Processes, Shell)

Life <-> Work

OS sched Policy: MLFQ

Problem: Don't know very much about processes!
like to learn: short jobs, longer running?

(how? ⇒ measure: using past to predict future)

many queues: job is on one queue @ any given time (might change over time)

priority:			
highest	Q_2		
middle	Q_1	B, C	
low	Q_0	D	

each queue has priority

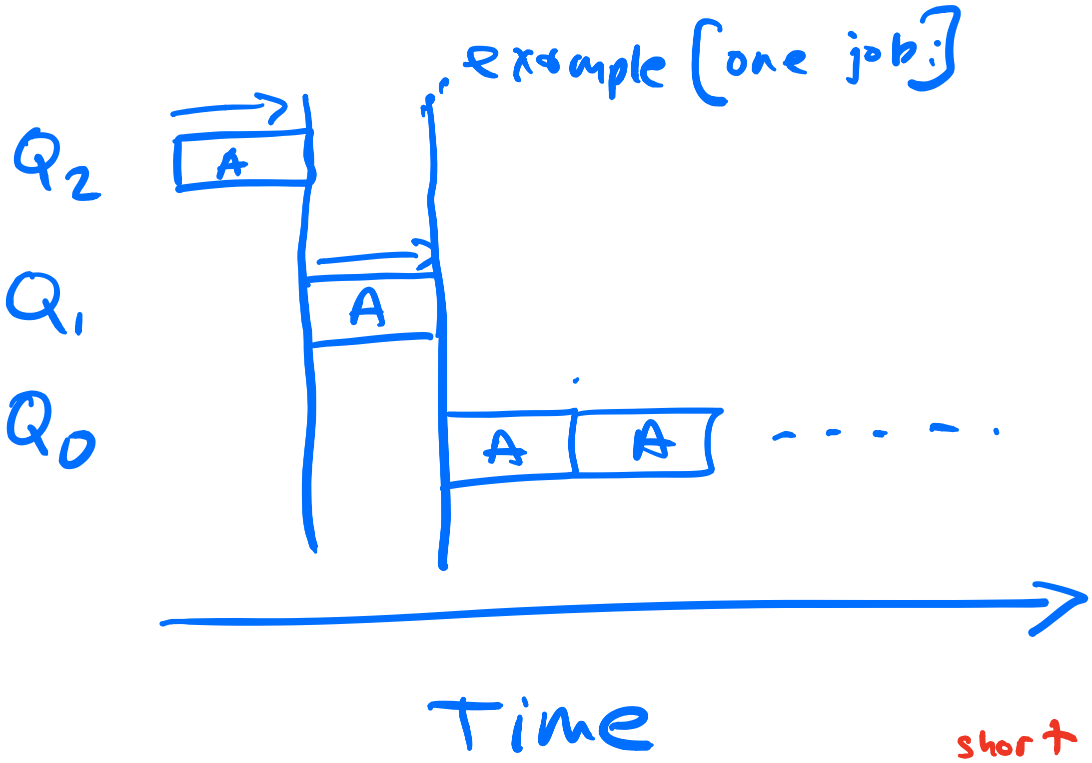
Rules:

1) if Priority(A) > Priority(B)
   ⇒ A runs (B doesn't)

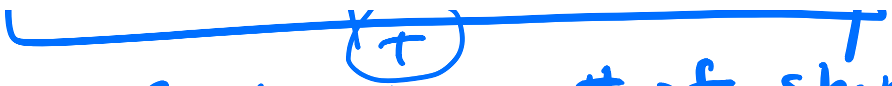
2) if Pri(A) == Pri(B)
   → Round Robin between them

3) Start: Highest Priority

4) if process uses time slice @ given priority,
⇒ at end of time slice, move down one level



Example 2:
long-running job: A
short running C_1, C_2, ..., C_N



$C_1, \ldots, C_N$: large # of short jobs!

=) never ending

=) what happens to A? (long running)?

=) [**Starvation**]

How to ensure long-running jobs make progress?

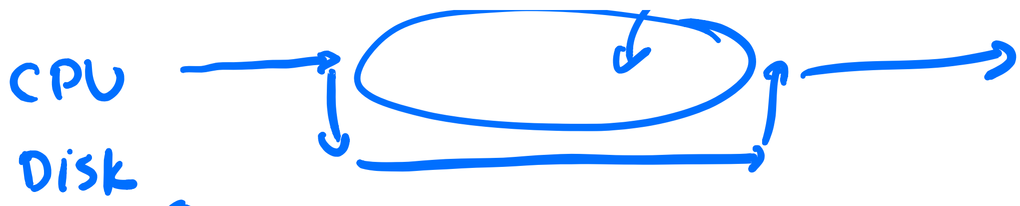
General idea: long-running need to move up

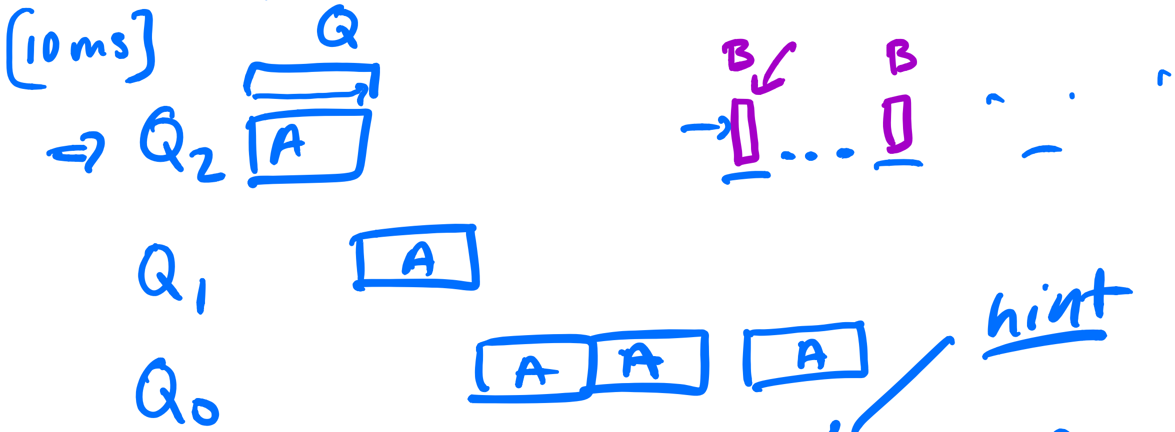
[Rule: Every T seconds, move all jobs to highest priority]

=) nature of job might change between interactive and batch

new worry : [I/O]

run something else her





A : long running

B : I/O job

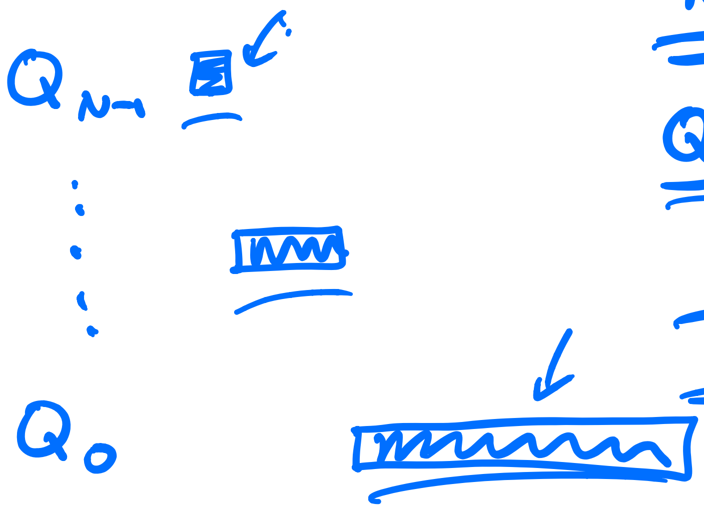
naive (hint) rule for I/O: if job runs for less than timeslice, stay at same level

=> "Gaming" the scheduler

II better accounting: if job uses up quantum, moves down

MLFQ

Parameters:



N Queues

Q_i: quantum length (per queue)

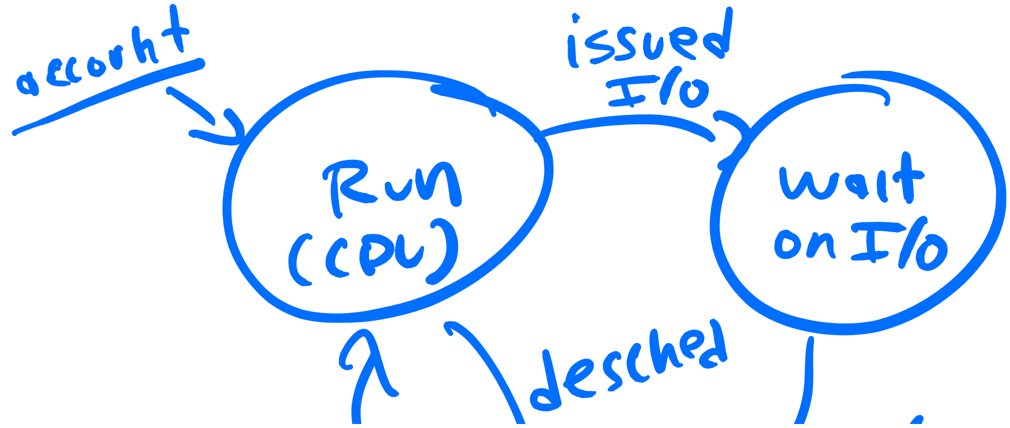
T: reset period

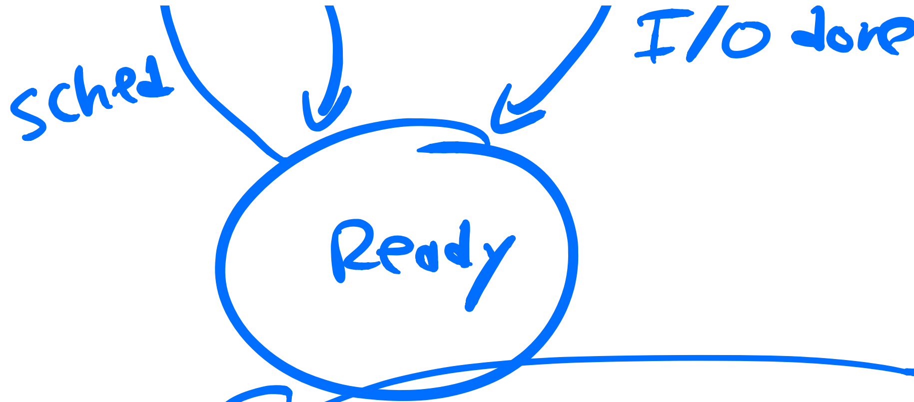
⇒ How to configure:

⇒ mystery

(use defaults)

⇒ new: ML





Break: **Best Sports-themed Movie**

- => ~~Waterboy~~
- => ~~Caddyshack~~
- => ~~Sandlot~~
- => ~~Dodgeball~~
- => **Miracle** ←
- => ~~Shaolin Soccer~~
- => ~~Space Jam~~
- => ~~Blades of Glory~~
- => ~~Remember the Titans~~
- => ~~McFarland~~
- => ~~Balls of Fury~~

Rocky
II
III
IV
V

original Karate Kid

~~Bad~~ good? news : class tomorrow

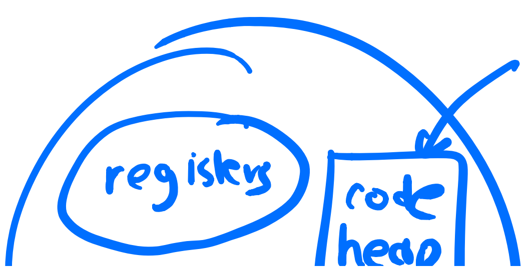
lack of knowledge → ~~1~~ ~~2~~ [3] 4 5 6 7

8 9 10 11 12 13 14 ← knowledge

Virtual Memory :

multi-cores
Linux CFS } later

Process : running program

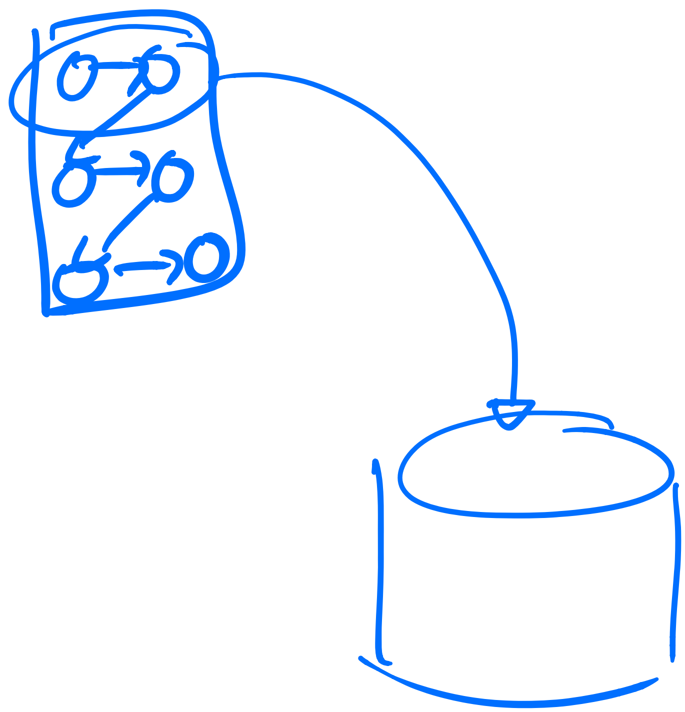


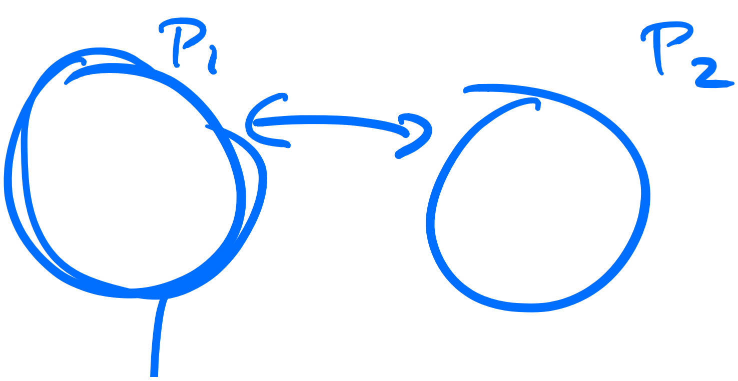
memory
=> virtual address

Space

[32-bit AS,
64-bit AS]

goals illusion of
⇒ large memory = [ease]
⇒ private (protected) memory
⇒ efficient memory





OS ↓ o → o → . . .

efficiency : Limited **Direct Execution**

(loads, stores, inst fetch)

[s/w ⇒ software]

Mechanisms :

- → h/w (hardware)
- → OS support

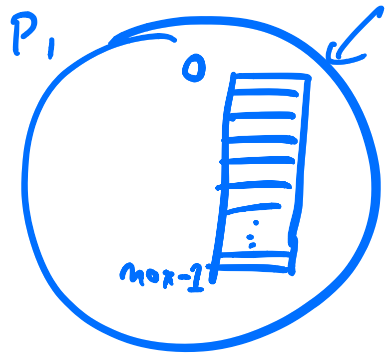
Memory Accesses:

- → instruction fetch
- → explicit loads, stores
  - loads: CPU ← mem
  - stores: CPU → mem

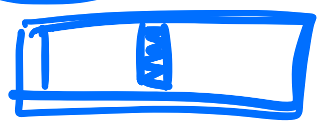
need: **interpose**

Address Translation:

illusion



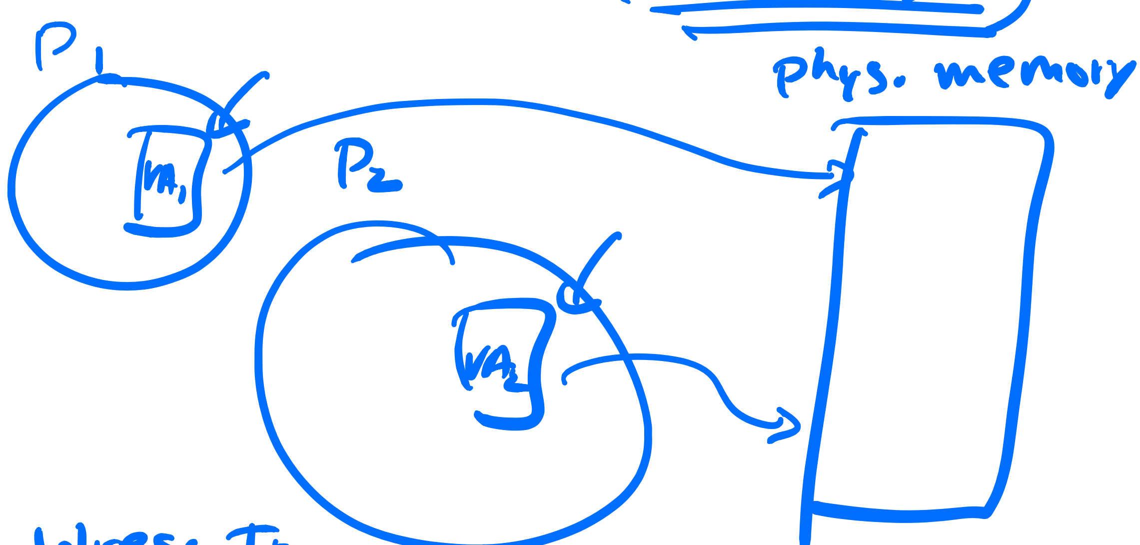
byte: 8 bits



[disk:
512 bytes]

[~~read~~
~~modify~~
write]

=> tonight's performance lesson (by Google)



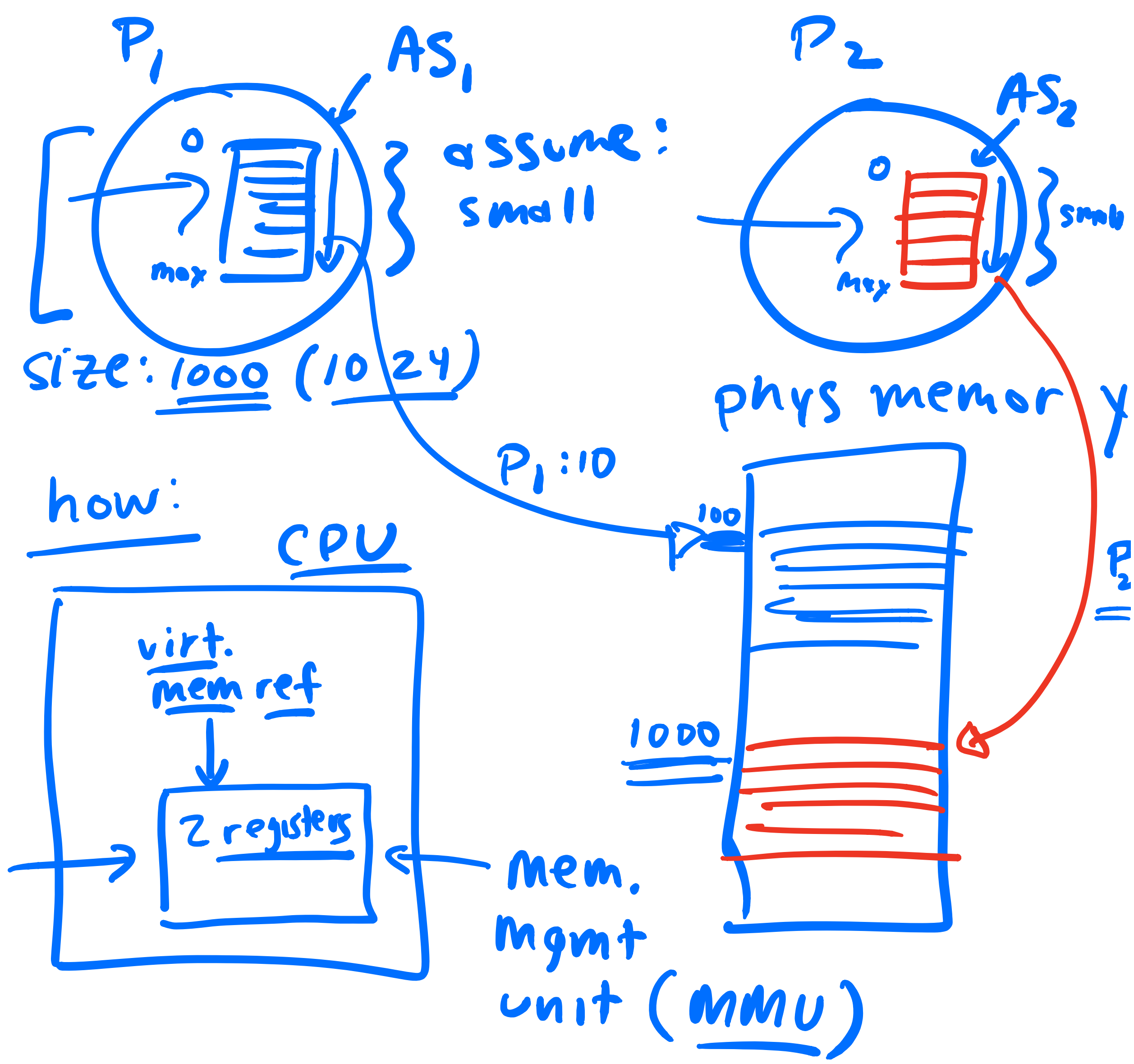
Address Translation
=> on every mem. reference,

translate virtual memory addr:

⇒ phys memory addr

Mechanism #1:

Dynamic Relocation ("or" Base/Bounds)



[2 registers / CPU]

→ **base** : address in phys mem where AS of currently running process starts

→ **bounds** :

MMU

VA ↓

Phys Address:

⇒ Base + virt address

⇒ check : w/in bounds

↓

Phys Addr. → (Physical) memory

→ every address in user program is [virtual address]

Modern OS: (LATER)

sharing of memory

⇒ Lesson #2:

(sometimes)

when?

Base: Translation

Bounds: Protection

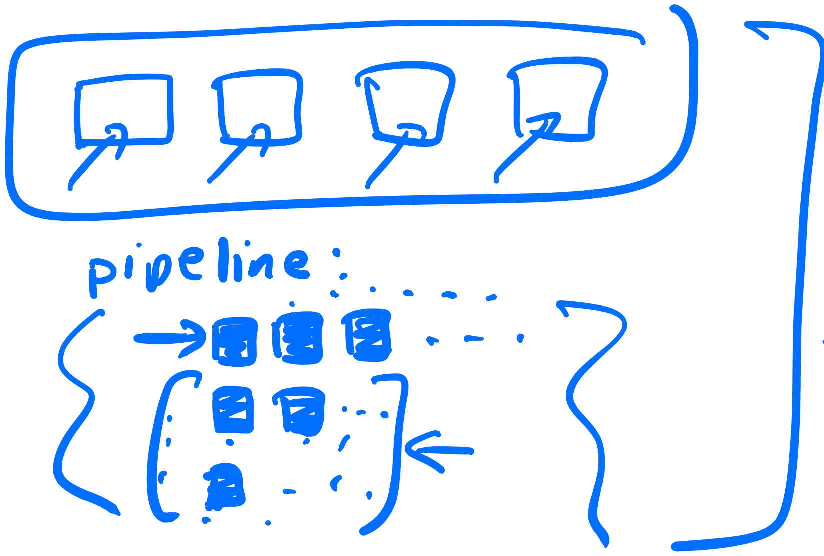
(check w/in Addr. Space)

CPU (abstractly):

```
while (1) {
    Fetch (pc)
    Decode
    Execute  ← [limit]
}
                MMU: [bounds]
```

if OK: go ahead

if not OK: raise exception



(ignore)

ASIDE

not OK: h/w raises exception

(illegal memory access)

=> H/W: OS -> Process: bad mem access

@ boot: OS set up exception handlers

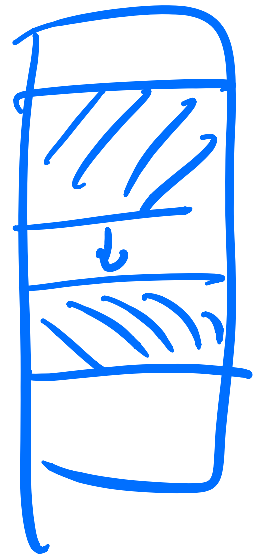
=> OS: kill process

Base/Bounds:

{ Fast
Simple } +

{ hard to
→ allocate
(contiguous)
→ hard to
expand } −

{ → hard to
share
memory }



[→ OS roles]

Question #2 : (Best non-sports themed movie)