Today

CPU virtualization
- mechanisms => limited Direct Execution
- -> Scheduling Policy

Q) How to build a scheduling policy?

Memory Virtualization

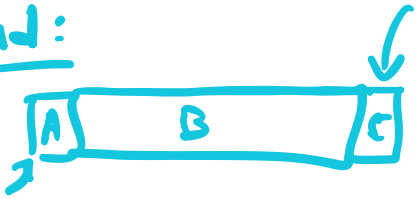
Scheduling

Metric:
- Turnaround Time:
  - $T_{completes} - T_{arrives}$

=> FIFO (FCFS)

"run to completion"

bad:



A, B, C arrive

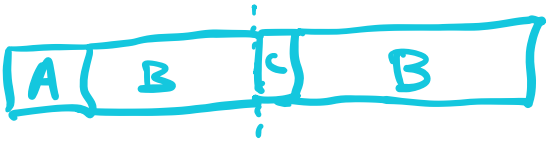
=> SJF "shortest job first"

"generalized" SJF:

STCF (shortest time to completion first)

short long
A, B

preempt B



C arrives,
C very short

=> Metric #2: Response Time:

A, B, C arrive

$T_{first\ runs} - T_{arrives}$

TS

context switch

Policy:
Round Robin

TS → Time Slice (Quantum):
multiple of how often timer interrupt goes off

"Real" Scheduler:

→ don't know job length (how to learn?)

→ workload:

→ interactive

→ long-running ("batch" or "background")

→ jobs ⇒ do I/O (don't just use CPU)

Key: use history to predict future

Policy: Multi-Level Feedback Queue (MLFQ)

priority

high Q_3 → A, B — 10ms

medium Q_2 → C — 20ms

low Q_1 → E, L — 40ms

⇒ a process can be on only 1 Queue

Priority: ⇒ process at highest queue runs

when to change queues?

⇒ each queue has time slice (how long job will run at that priority)

⇒ when time slice is used, move down one level

initial? job ⇒ highest priority

⇒ if >1 job @ some priority, use round robin

Example:

1 long running job: L

occasionally, an "interactive" job shows up

$>_1$, $>_2$, $>_3$

⇒ works

Q_3 L S_1 S_2

Q_2 L

Q_1 LLLLL LLLL LLL ...

another long job: M

Q_3 M . . . . . .

Q_2 M

Q_1 LL MLMLMLML

starvation:

"too many" short jobs arrive + run, long-running jobs may never get service

solve?

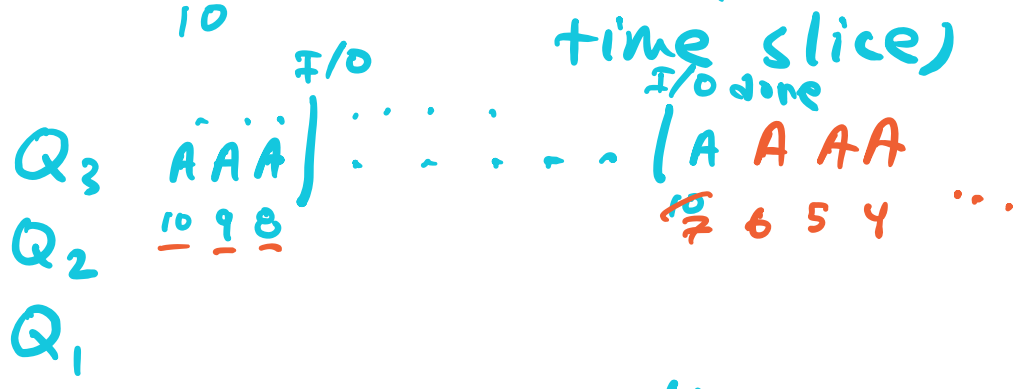
periodically, reset jobs ⇒ top ("high")

what about I/O?

=> approach #7:

if do I/O, stay @ same queue

(reset counter, to run for full time slice)



Problem: "gaming"

=> retaining high priority artificially

Memory Virtualization

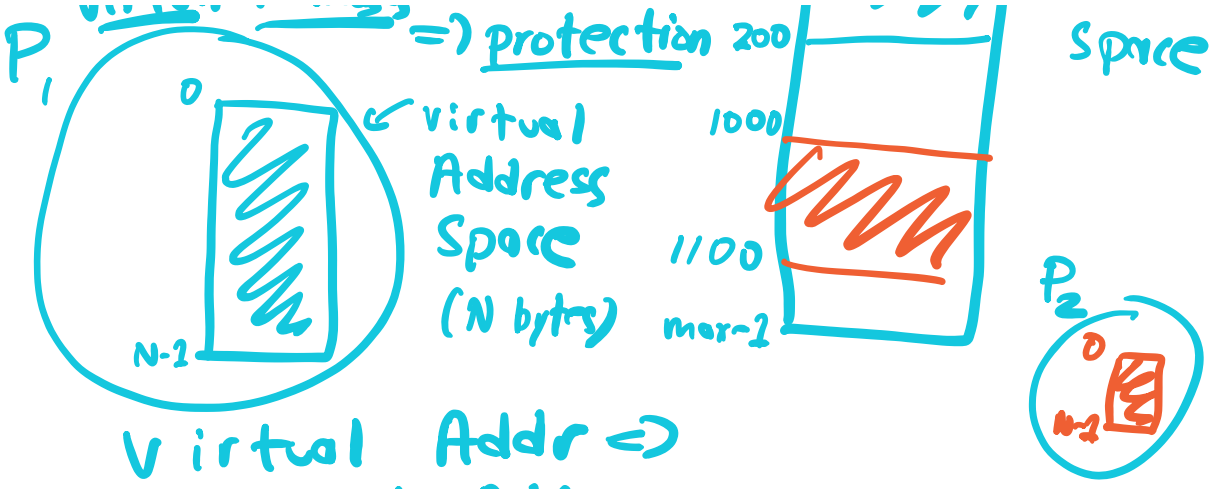
Each Process:

Virtual Memory

C pointers: virtual addresses

=> Shared by all running programs efficiently

Physical Memory



Virtual Addr => Physical Addr.

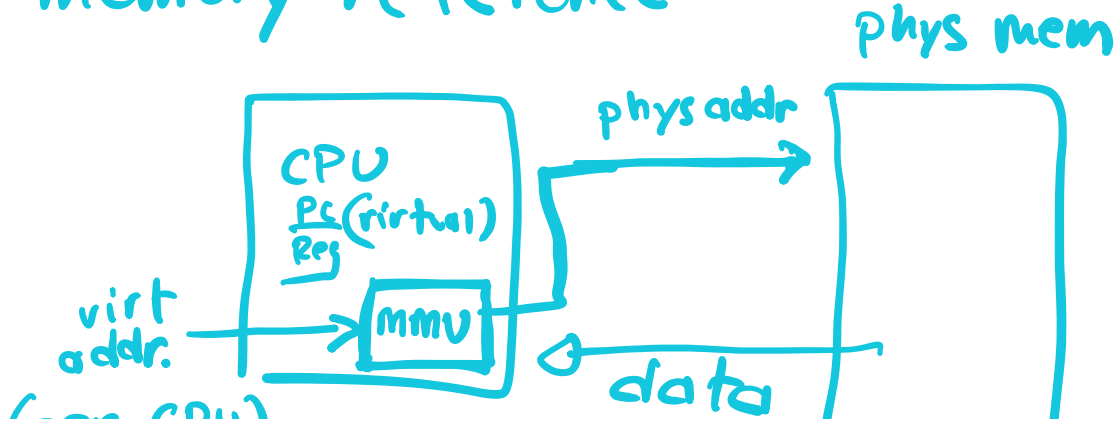
"Address Translation"

=> [this happens on every memory access (virtual)]

e.g. load addr, R1

[PC: virtual address
then load: another virt. addr.]

upon memory reference:



(per CPU)

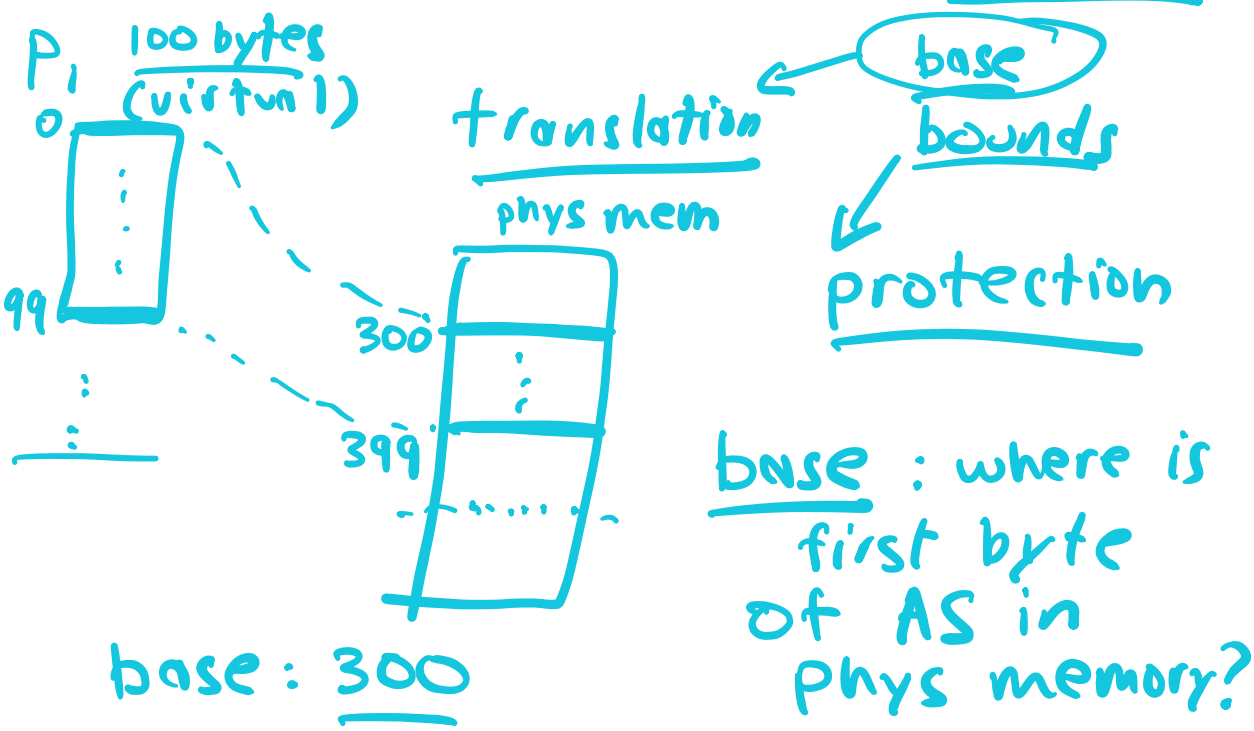
MMU: hardware (on CPU)

memory management unit

(assist in address translation)

simplest form of MMU:

⇒ base / bounds



base: where is first byte of AS in phys memory?

base: 300

on every mem. access:

virt addr → MMU

base + virt. addr ⇒

phys. addr

bounds: check to make sure "legal" address

make qemu-nox