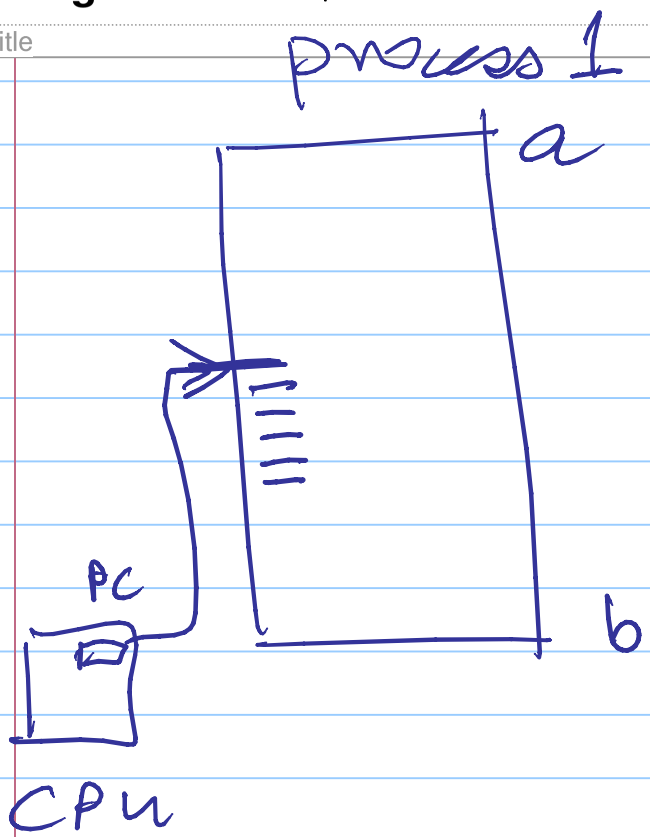
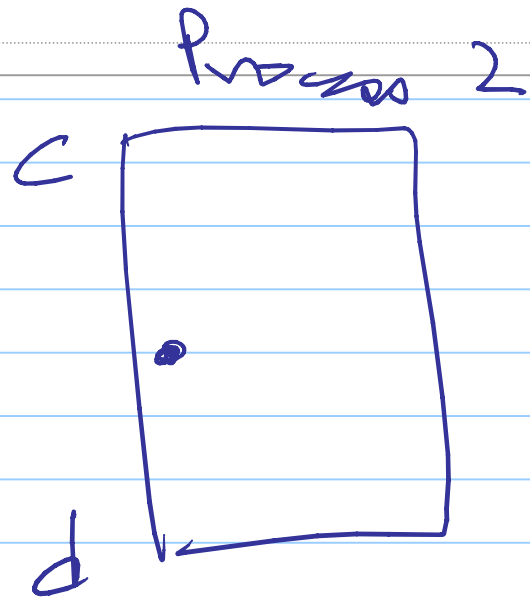
CSE 330: Operating Systems

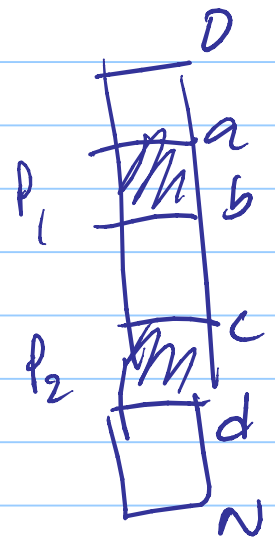
Class: 05 **Date:** 1/23

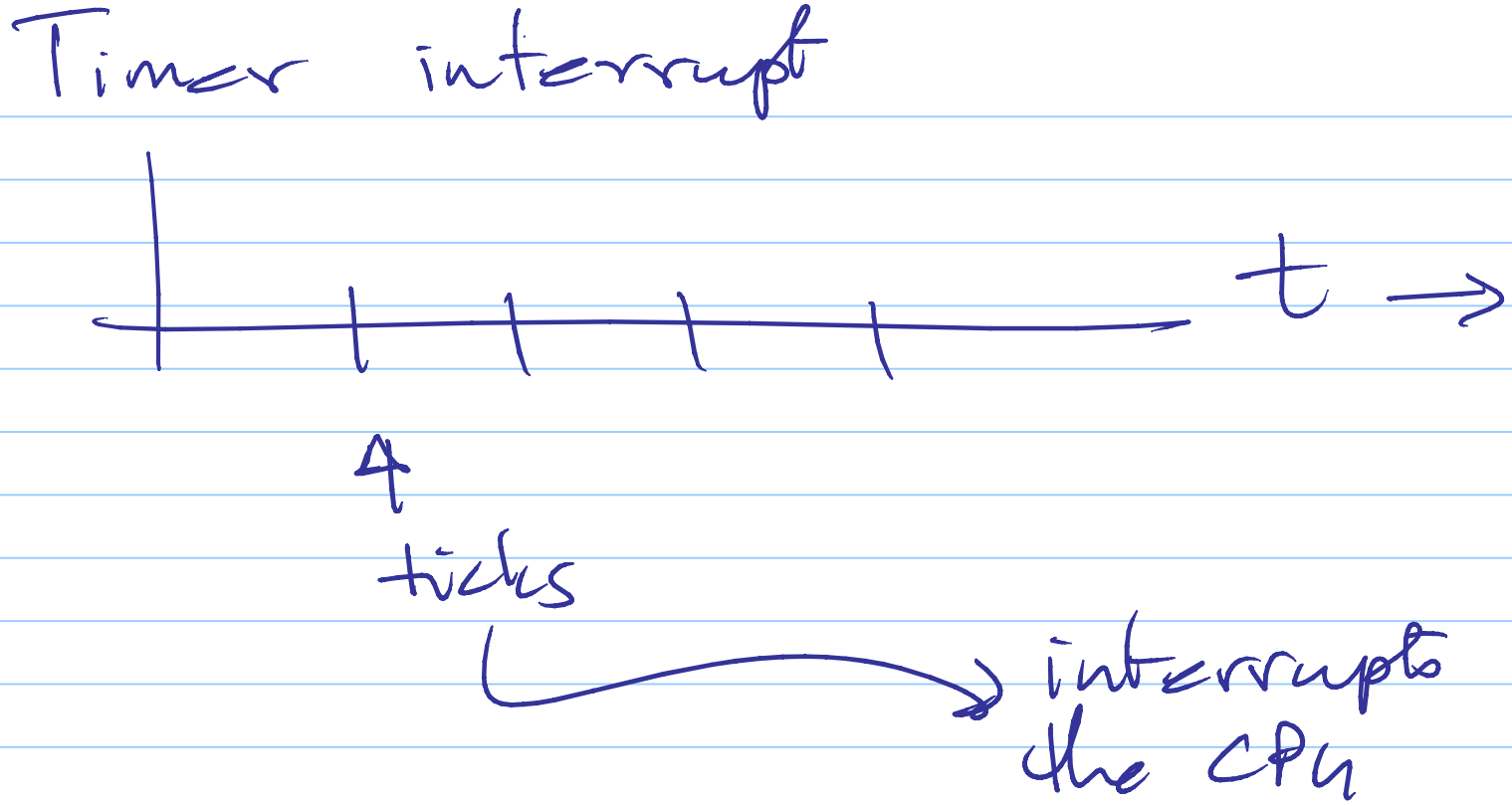
Spring 2018

Note Title

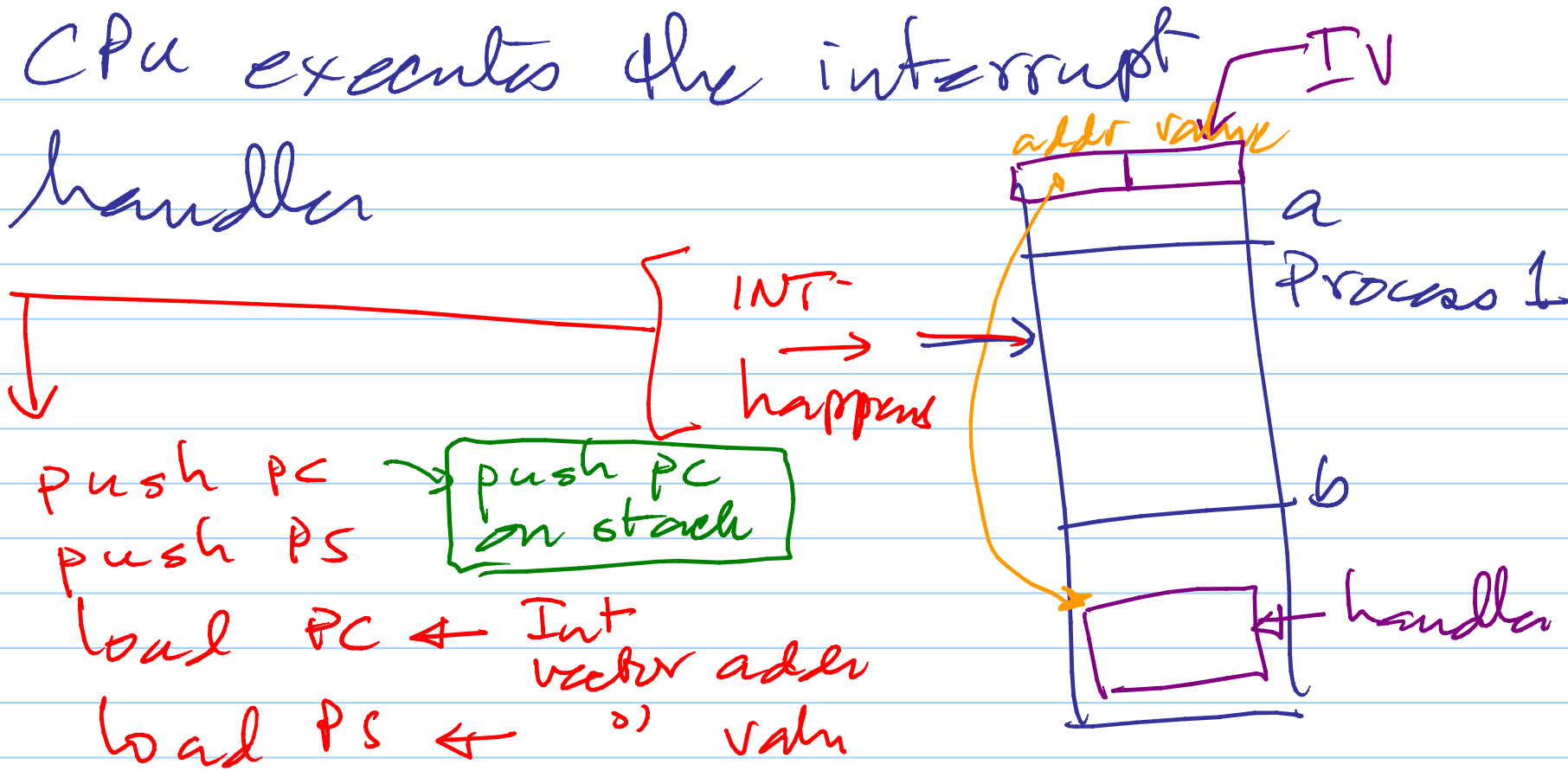


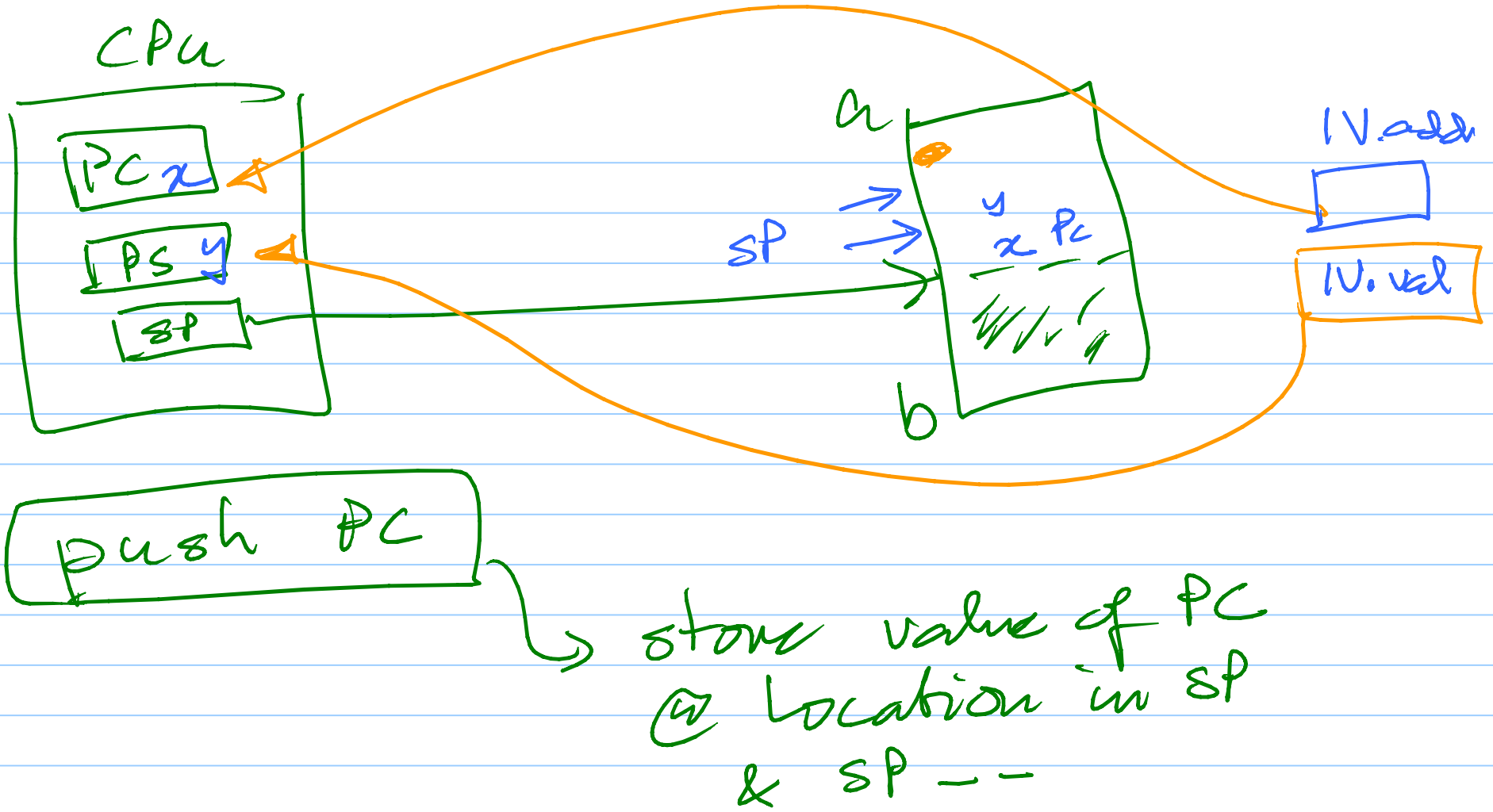





Timer interrupt
t →
4
ticks
interrupts
the CPU

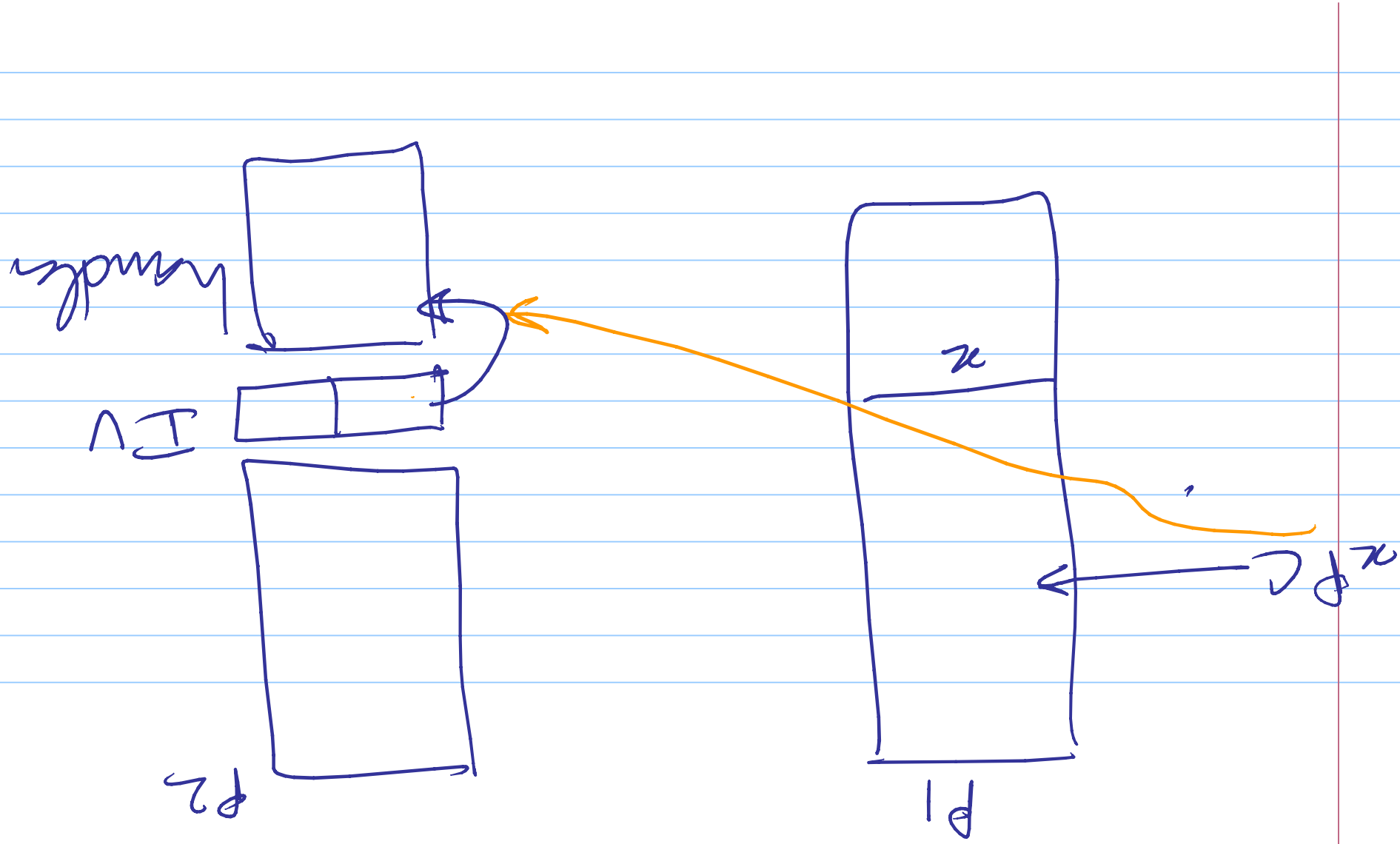
CPU executes the interrupt handler



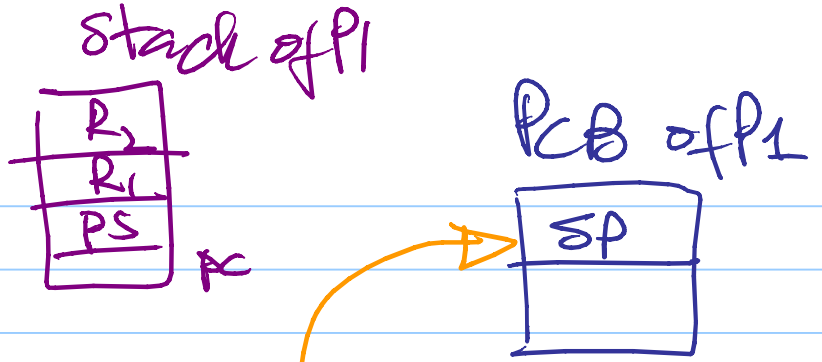


push PC

→ store value of PC @ location in SP & SP--


x
p1
p2

Int handler



SAVE CONTEXT

```
push R1
push R2
  etc
```

} on stack of P_1

STO SP → PCB of P_1 (SP field)

LOAD SP ← PCB of P_2 (SP field)

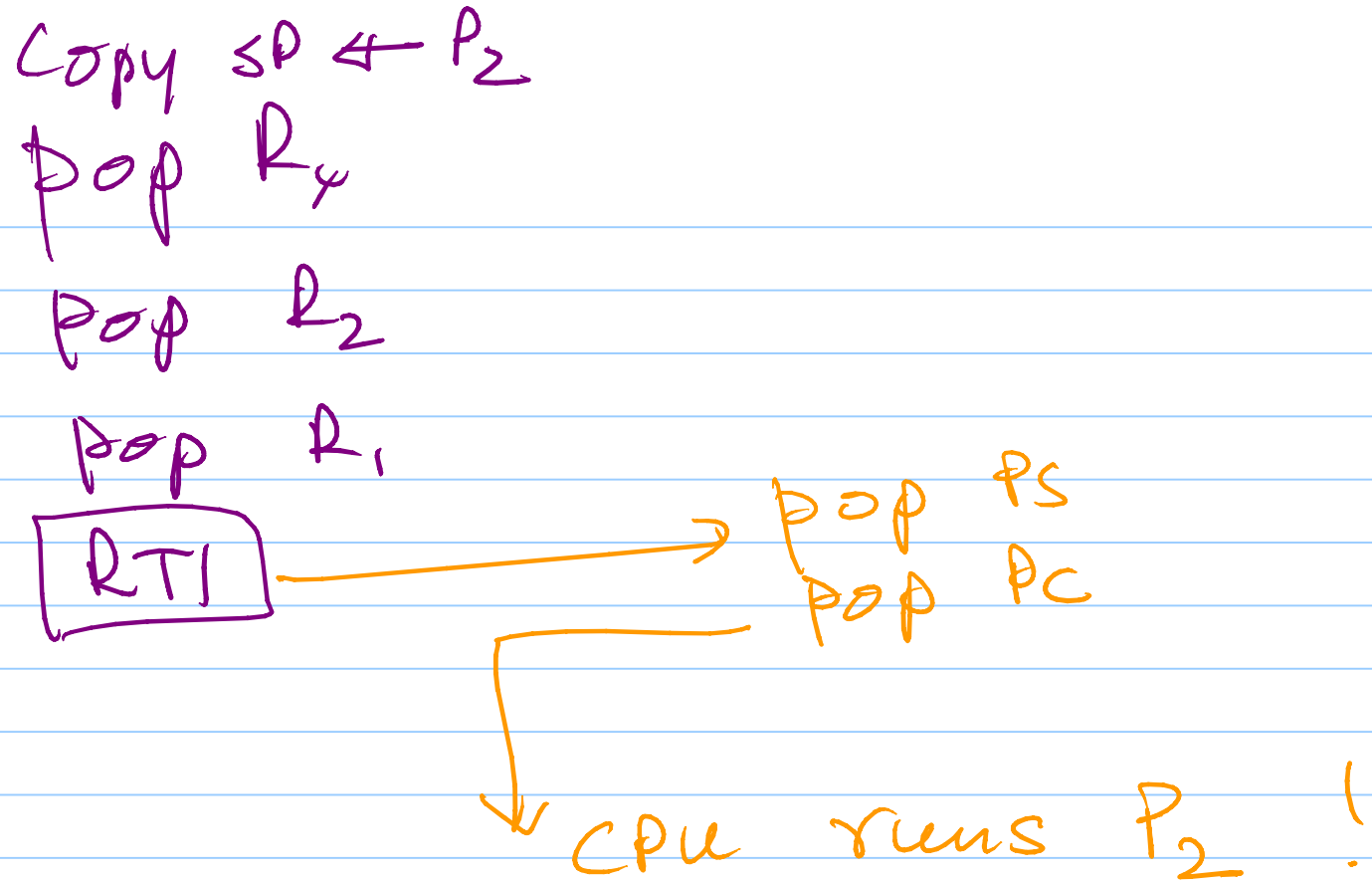
```
pop R2
pop R1
```

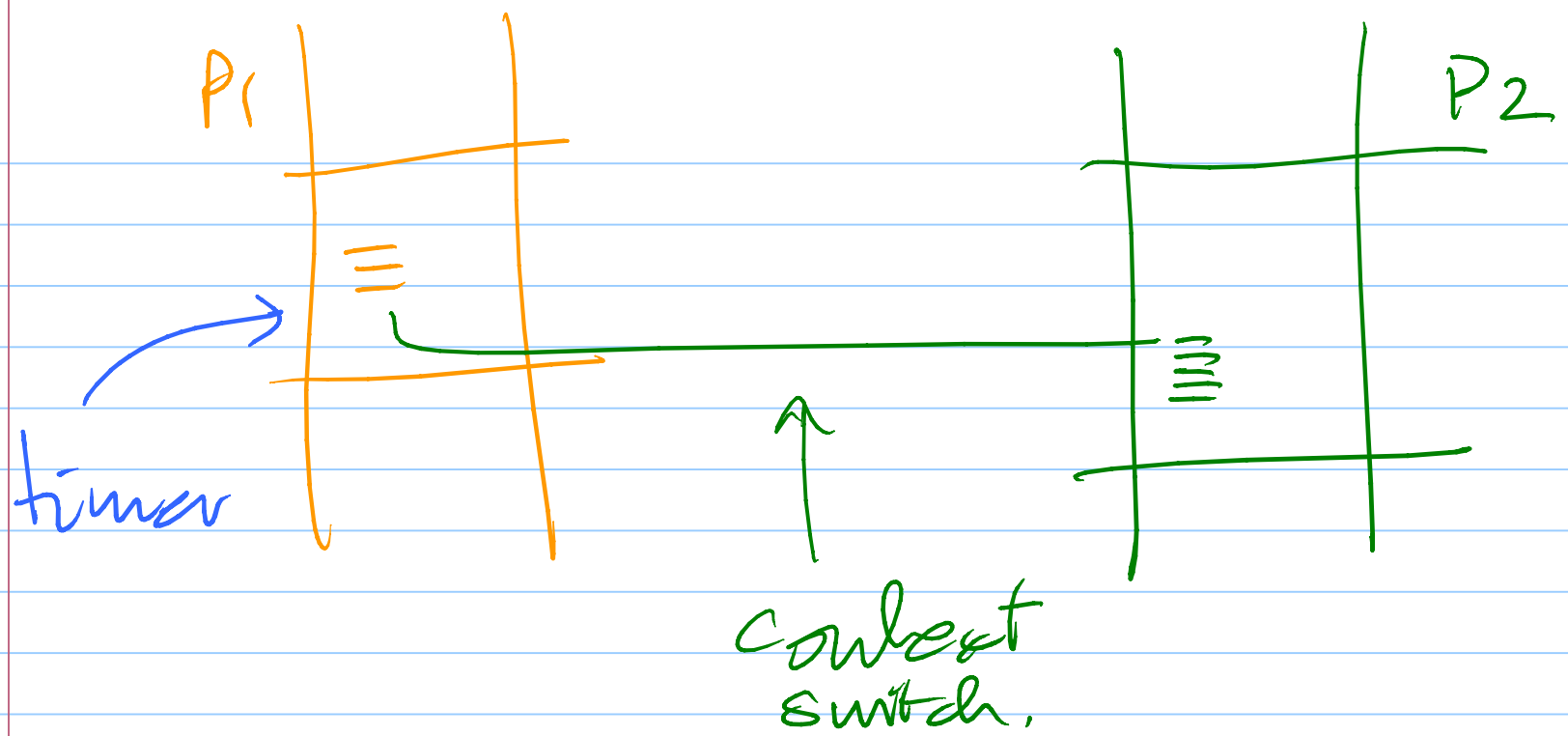
push R_x → copy R_x to location pointed by sp

sp − −

pop R_x → copy mem contents from location pointed by sp to R_x

sp + +


Copy SP ← P2
pop Ry
pop R2
pop R1
RTI
pop PS
pop PC
CPU runs P2 !


P1
P2
timer
contact switch.

lots of processes

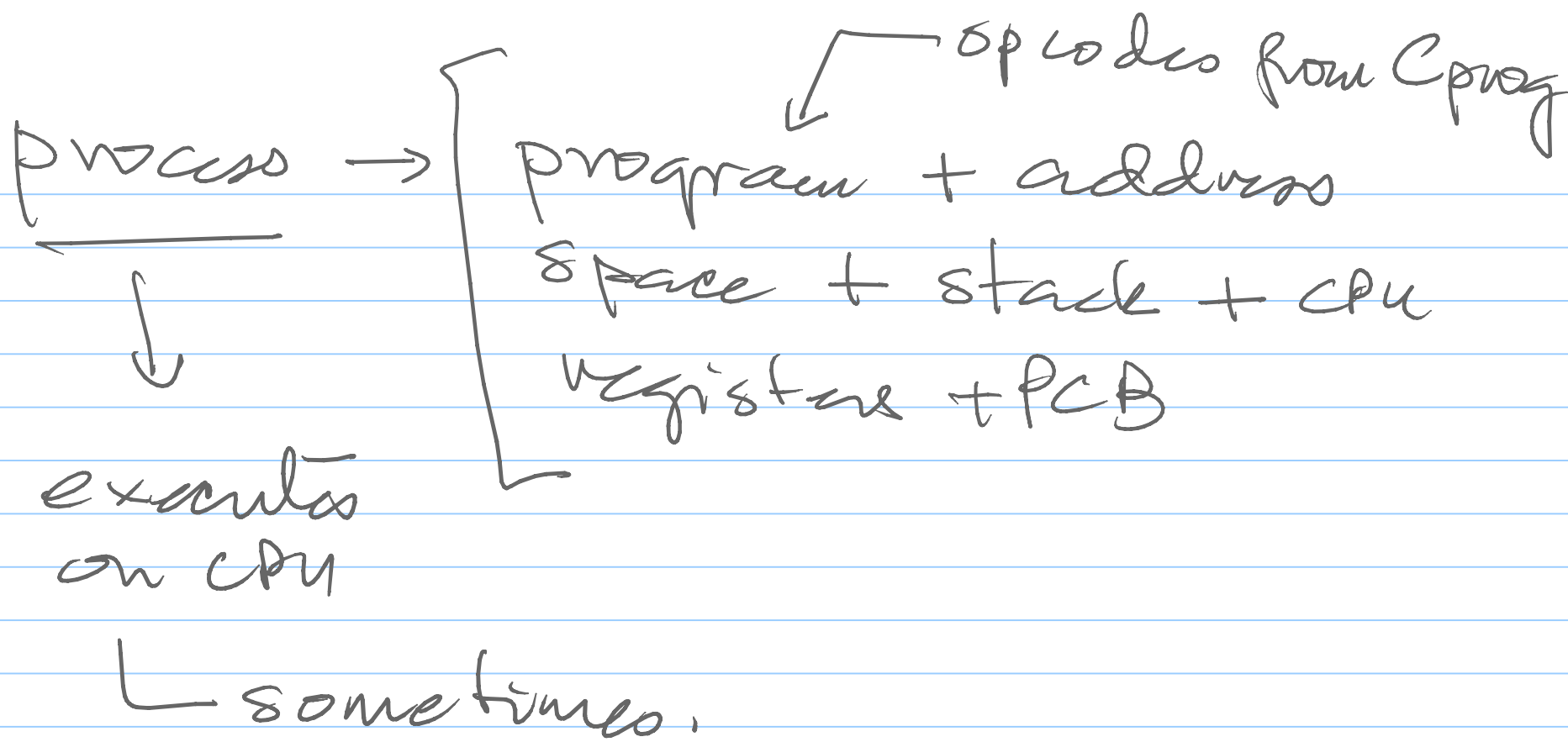
" " PCBs

each process has a stack

stack contains register / PC / PS
(only when process not running)

context switch → switch from running process to any other

scheduler —


process
program + address space + stack + CPU registers + PCB
opcodes from Cprog
executes on CPU
sometimes.

Thread → the activity in a process

→ a 'process' can have multiple threads

→ threads are like processes

→ threads share data of a process

int i = 0

main

thread

processo

{

i++

print (i)

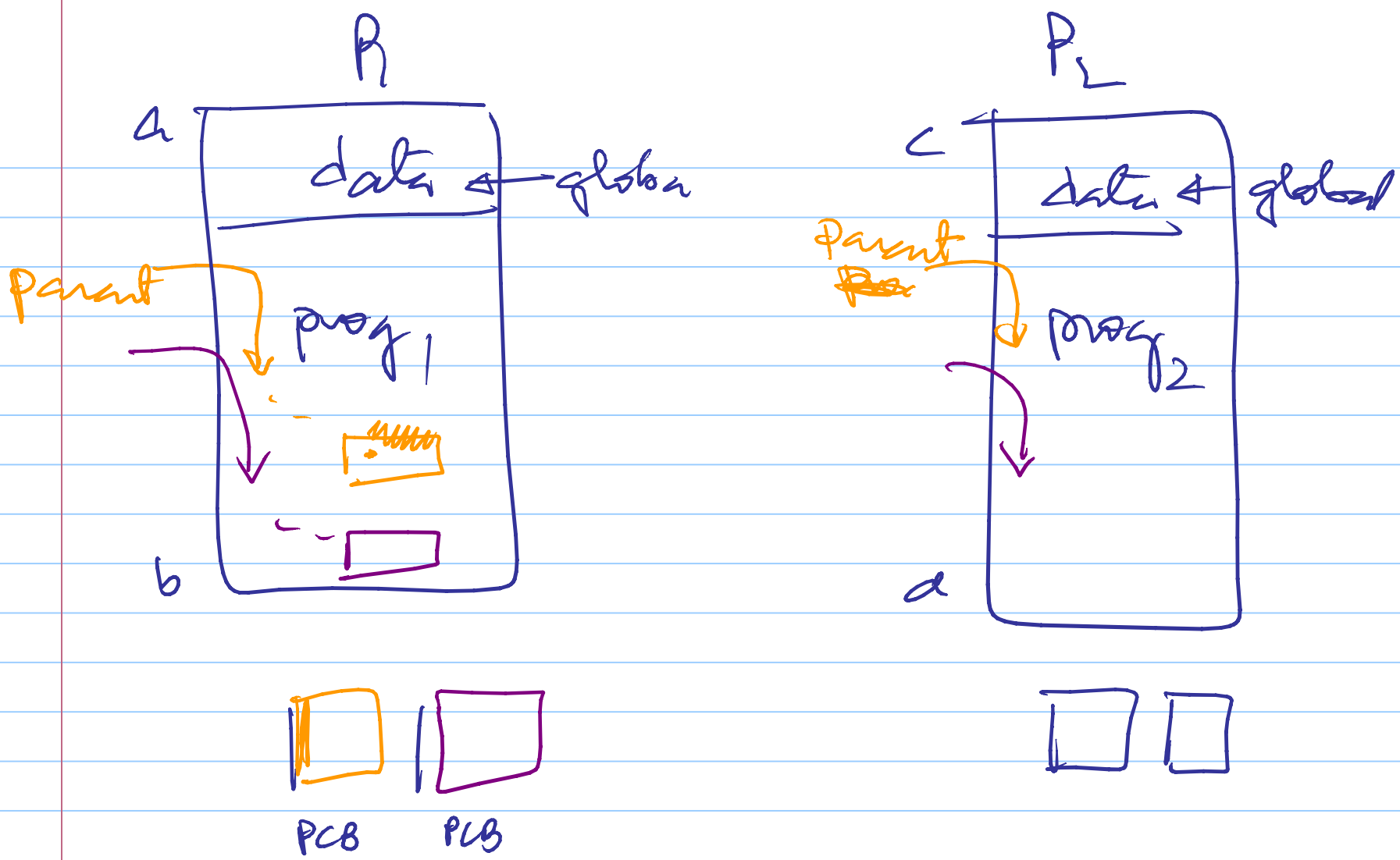
i++

print (i)

. . .

}

1 2 3 4 5 6


P1
a
data
global
parent
prog1
b
P2
c
data
global
Parent
prog2
d
PCB
PCB

Parent process

↳ thread (parent)

↳ child

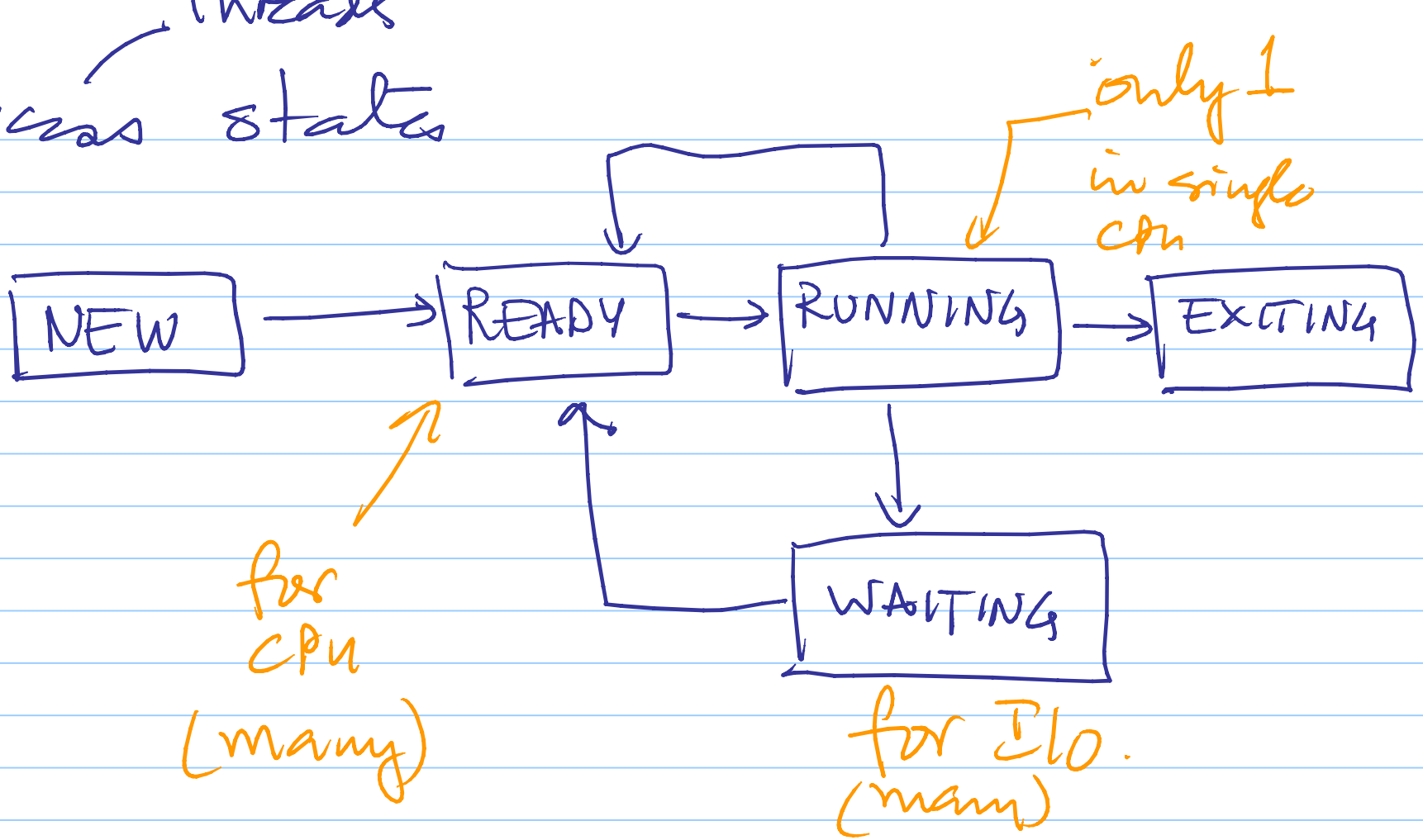
↳ child

share the global data and heap and code

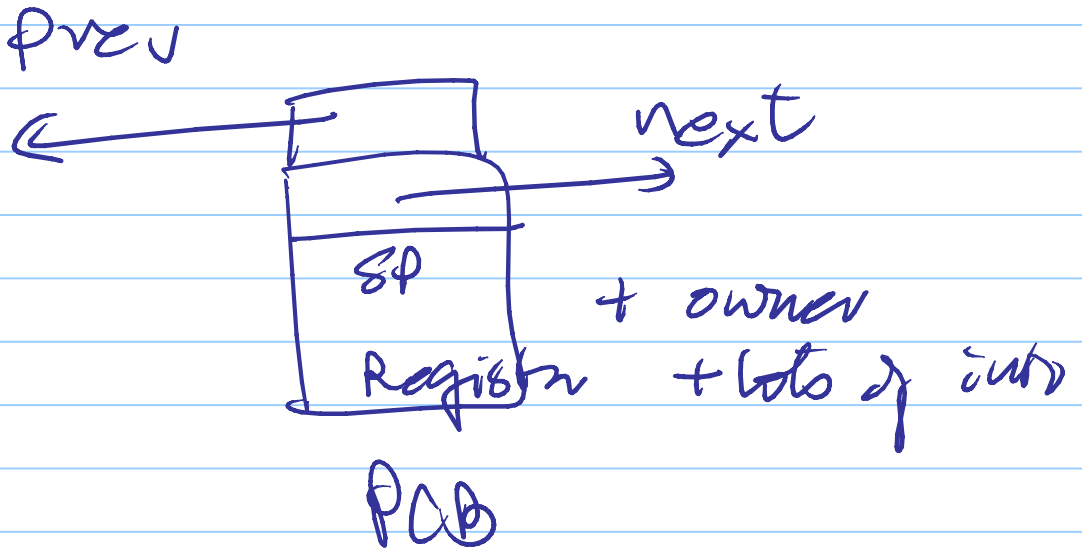
But do not share stack & PCB / TCB

Threads

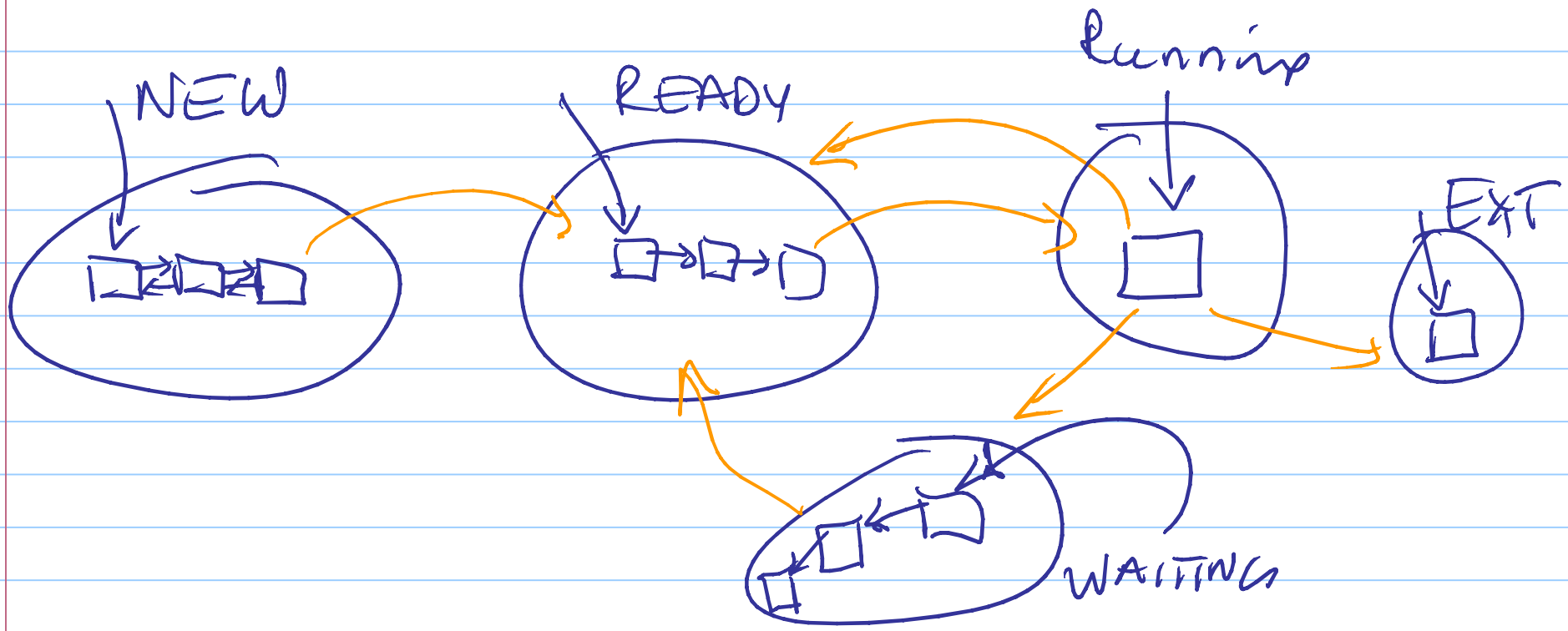
Process states



Scheduling Queues



Scheduling queues



CPU scheduler
READY
RUN
WAIT