CSE 531: Distributed and Multiprocessor Operating Systems

Fall 2020

Class: 07 Date: 9/16

Note Title

Context switching

↓

scheduler → Q of PCBs

TCB

[process
thread
Control
Block]

↳ kernel data structure
— info about a process —

process **context**

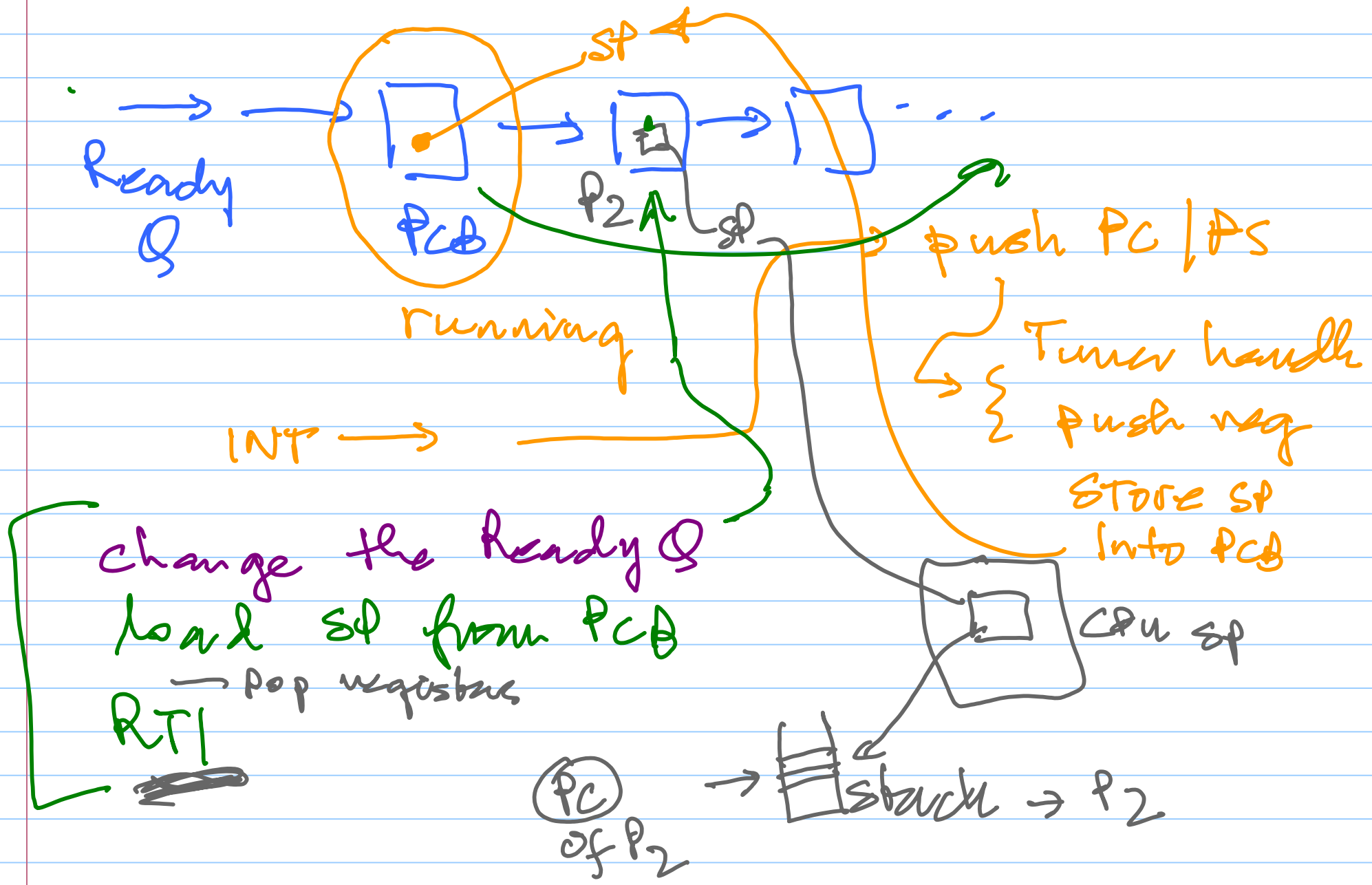
↳ info needed to restart a process

⇑ process not running!

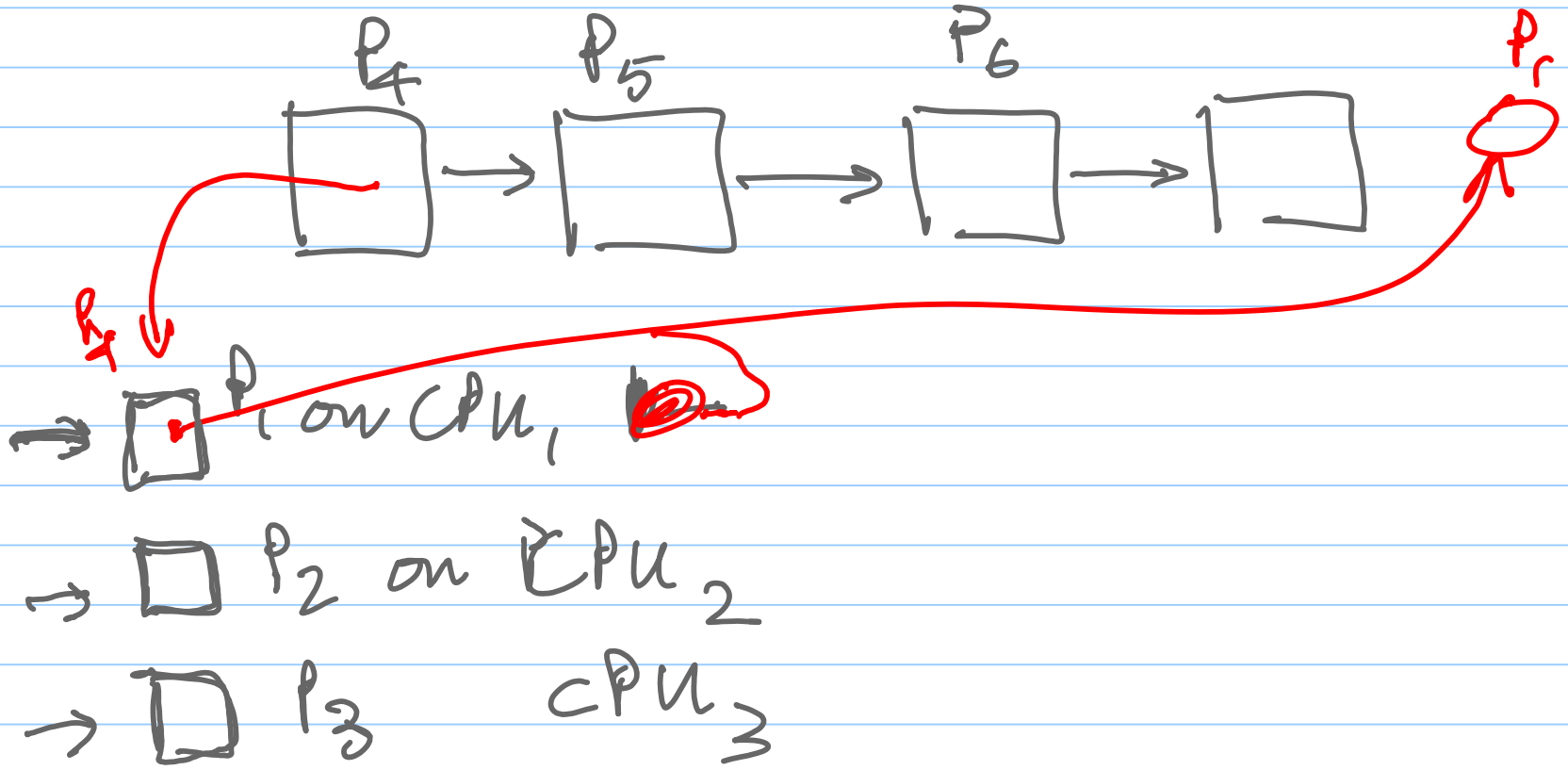
PC, PS, SP, registers & memory info

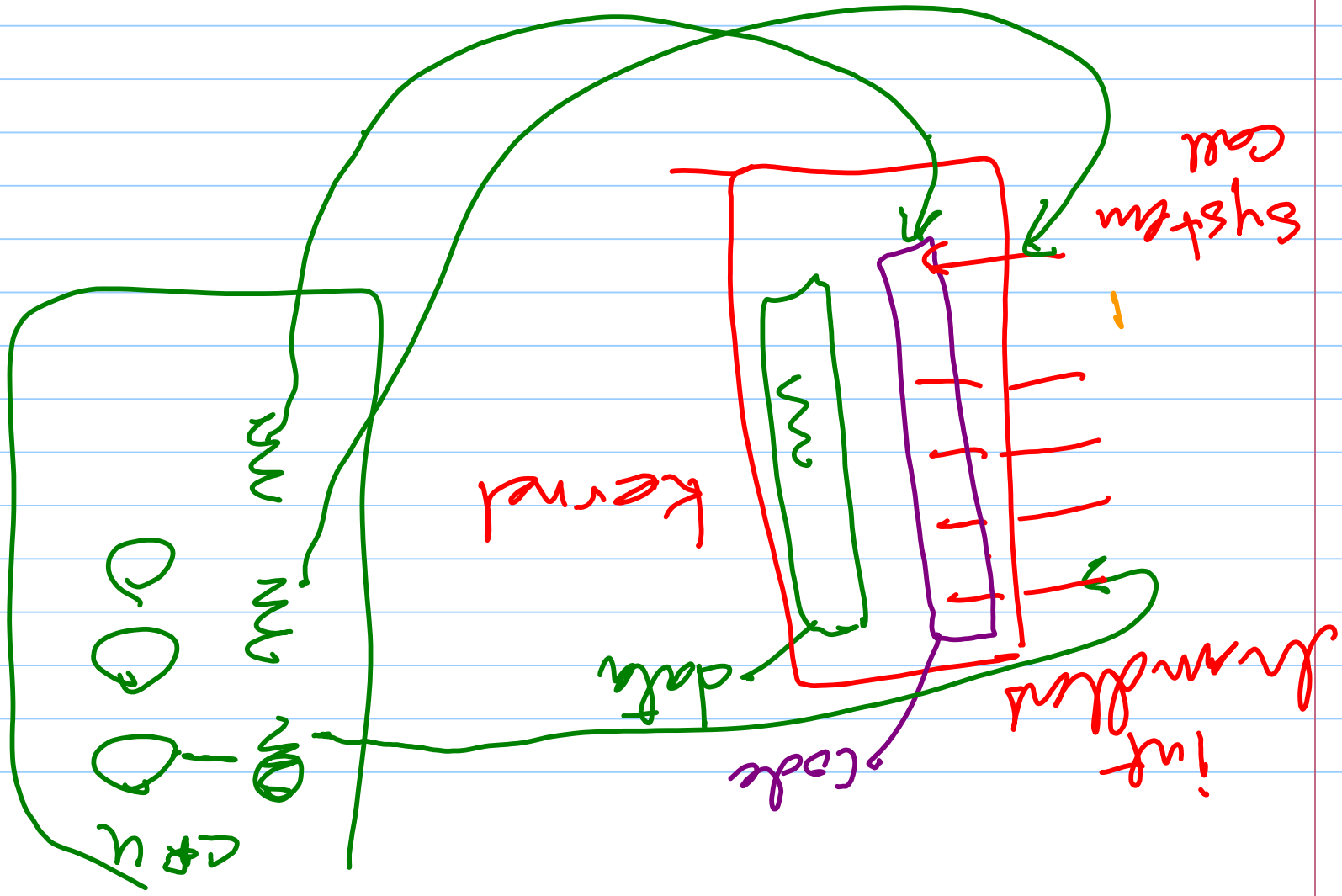
stack

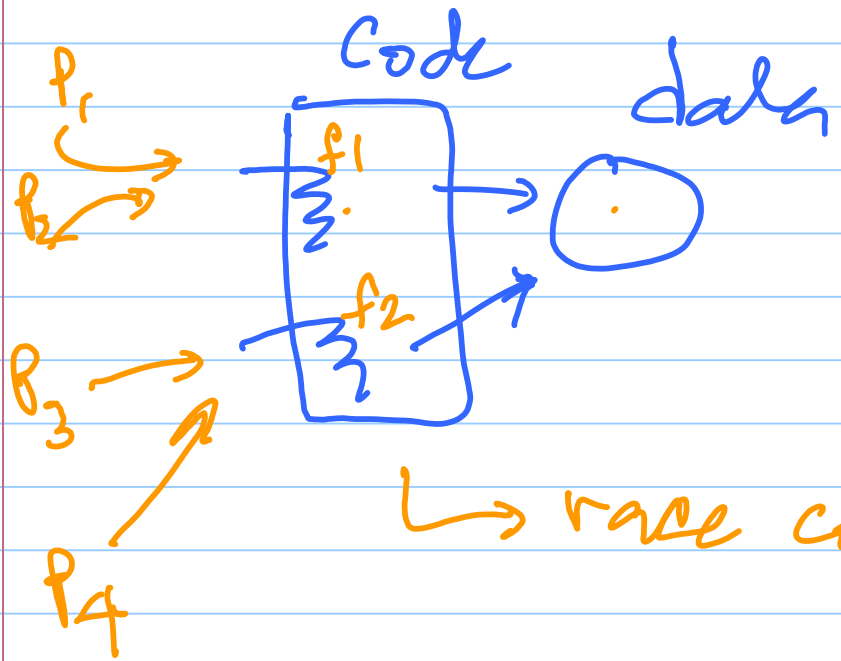
1 single CPU


Ready Q
PCB
SP
P2
SP
running
push PC | PS
Timer handle
{ push reg
Store SP into PCB
INT
change the Ready Q
load SP from PCB
pop registers
RTI
CPU sp
PC of P2
stack → P2

multiple CPUs




CPU
kernel
data
code
initialized
system
call



↳ race conditions → spin locks

Atomicity

- indivisible execution

OR

- divisible execution that appears indivisible

Serializable → - a parallel execution that is equiv to some serial ex.

Atomic
Cache
data
data

Semaphores

Semaphore_type

Sem S1, S2;

init value ← (1)

value

wait → P(S) → A: if [S>0 → S--] (atomic)

else goto A;

BUSY WAIT

signal → V(S)

S++

Atomic

P(S) while(1) { if [S>0 S--] break;

∞ loop

}

Implement Semaphores

1. — single processor.
2. — multiple processor.
3. — mutex ~~locks~~ semaphores..

Sem → int

int

P(S) → { Disable

while (S==0) { ENABLE;

DISABLE; }

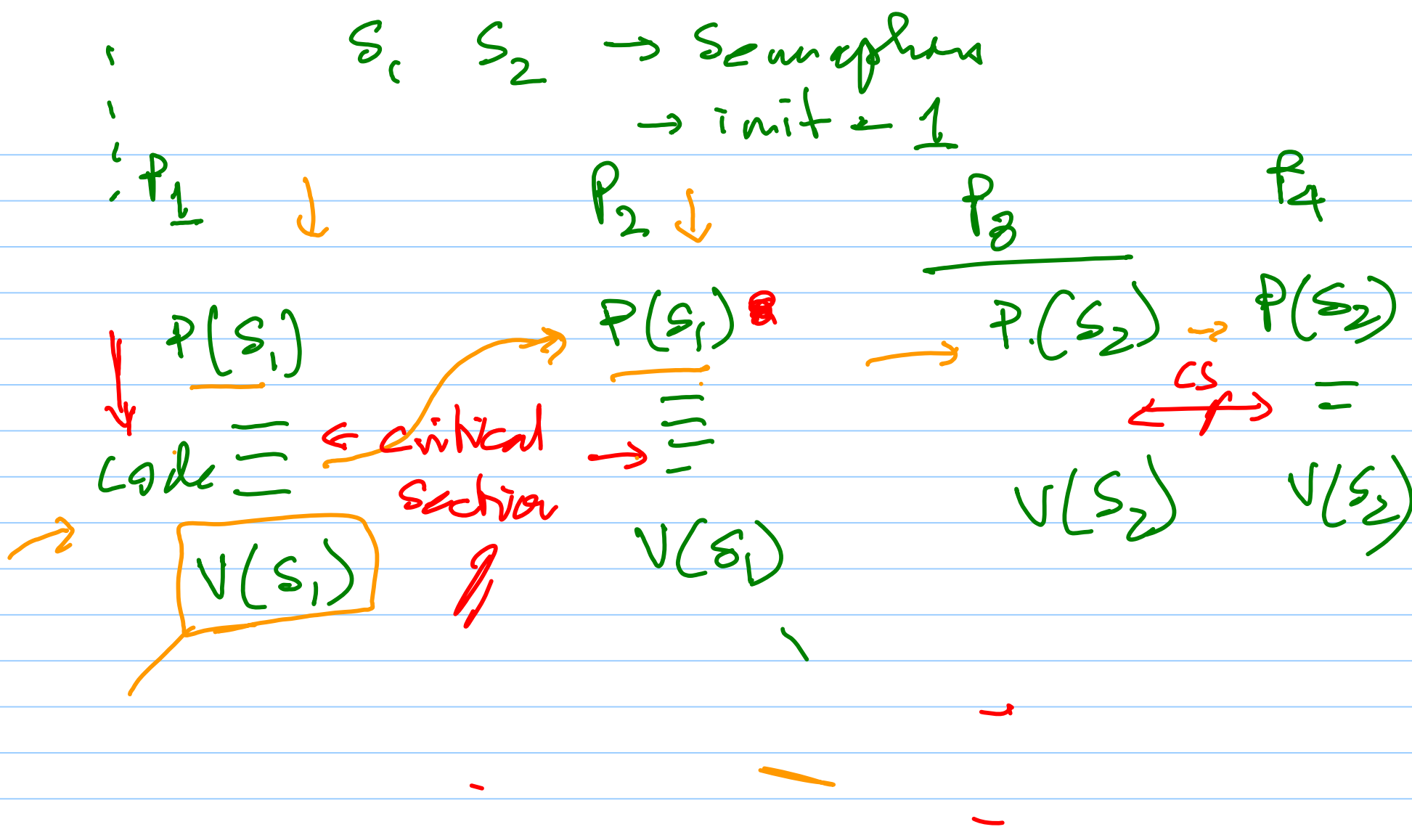
S--

ENABLE }

Single CPU

V(S) → disable ; S++ ; enable ;

atomic


S1 S2 → Semaphores
→ init = 1
P1
P2
P3
P4
P(S1)
code
critical section
V(S1)
P(S1)
V(S1)
P(S2)
CS
V(S2)
P(S2)
V(S2)

Sem { int count_
int lock_

P(S) {

spinlock (s.lock)

while (s.count == 0) { spinunlock (s.lock)

spinlock (s.lock)

s.count --

spinunlock (s.lock)

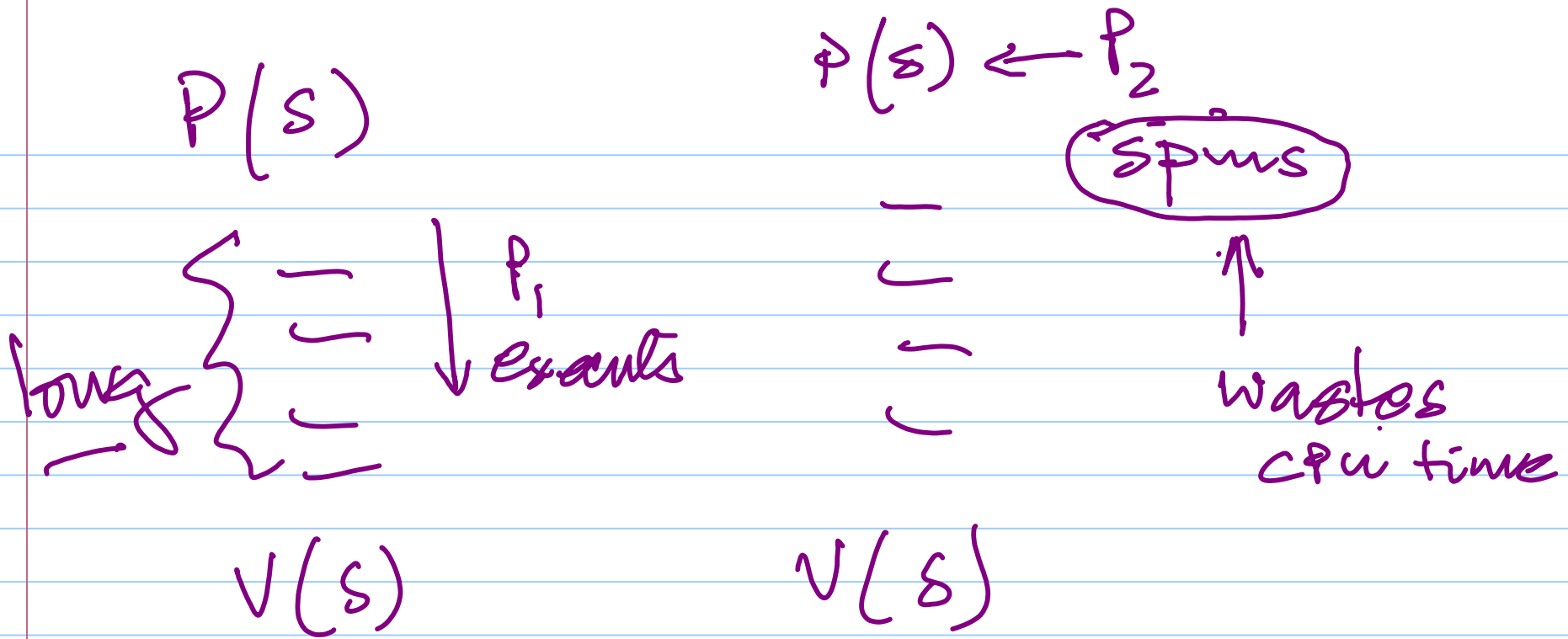
}

while test&set

V(S) { spinlock (s.lock)

s.count ++

spinunlock (s.lock)


P(s)
long
P1
exams
V(s)
P(s) ← P2
spins
wastes
cpu time
V(s)

mutex semaphores

→ [int counter
int lock
queue of PCBs — Q]

P(S) {
if sem not avail (positive) → put process to sleep
}

mutex → mutual exclusion

→ a kind of lock or semaphore that does locking

→ kind of lock or sem that does not use busy waiting

Mutex sem → blocking sem

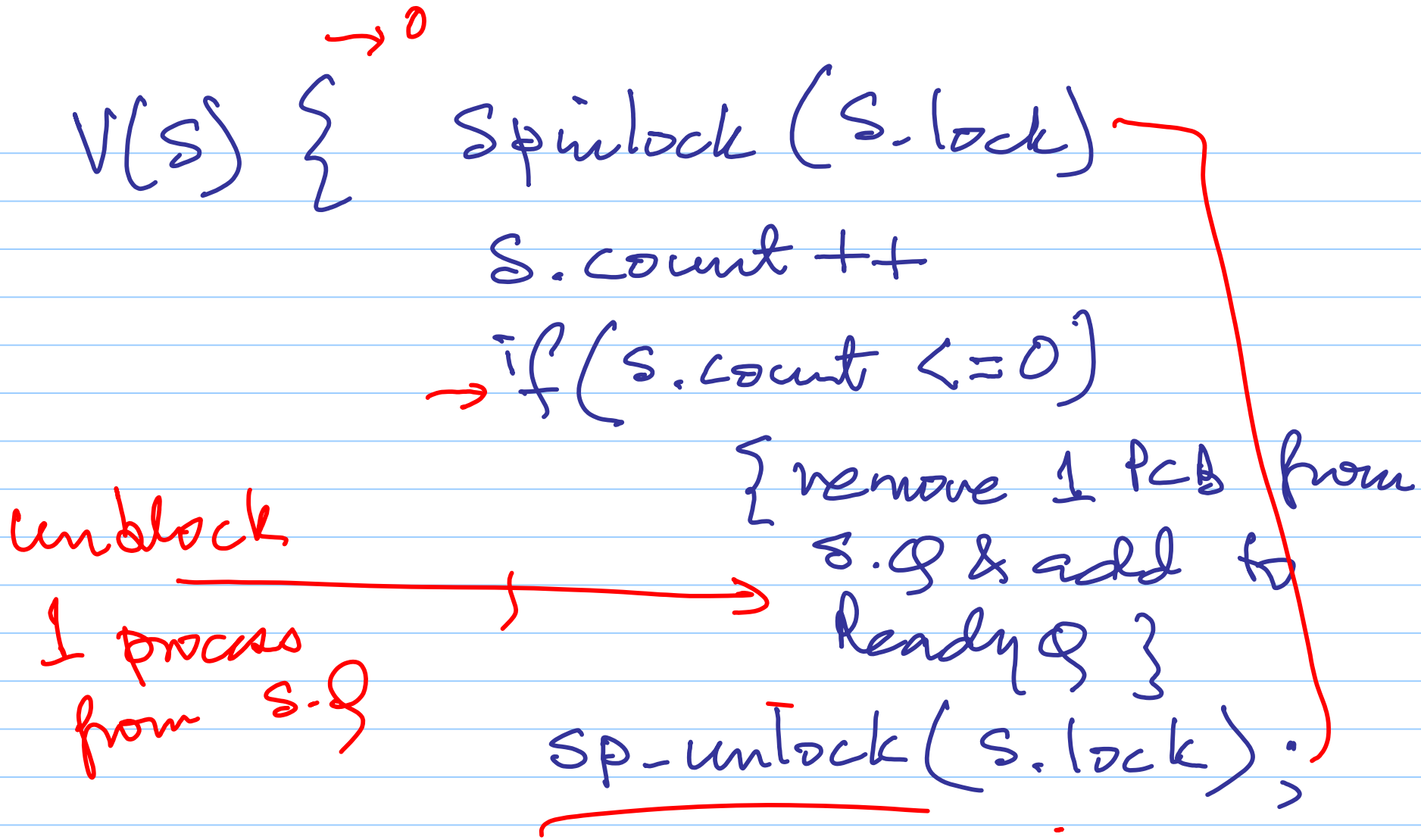
```
P(s) {  → sp_lock ( s.lock)
        s.count --
        if (s.count < 0) {
Block       =
forever     =
            else
              sp_unlock (s.lock)
        }
```

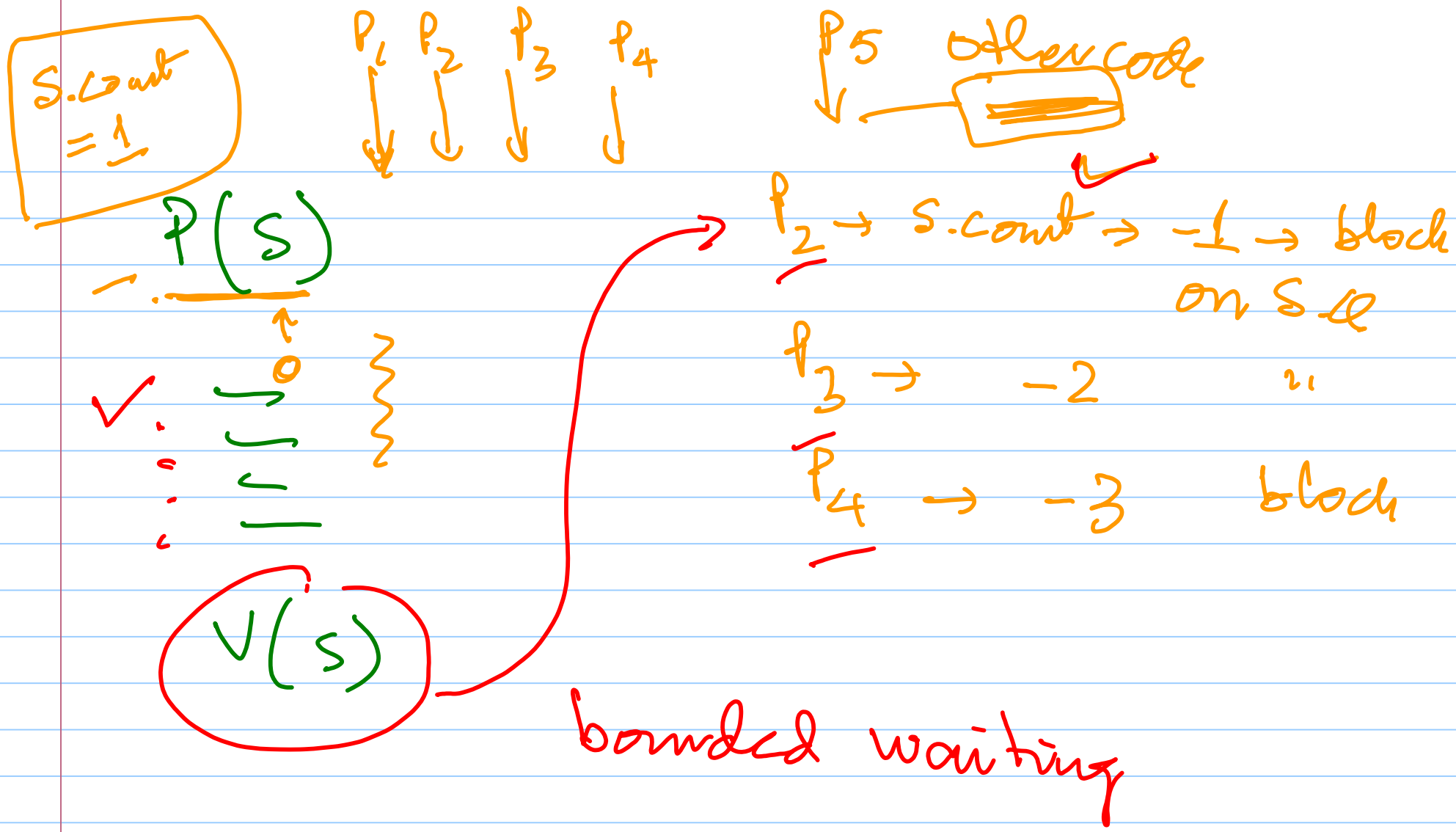
unlock

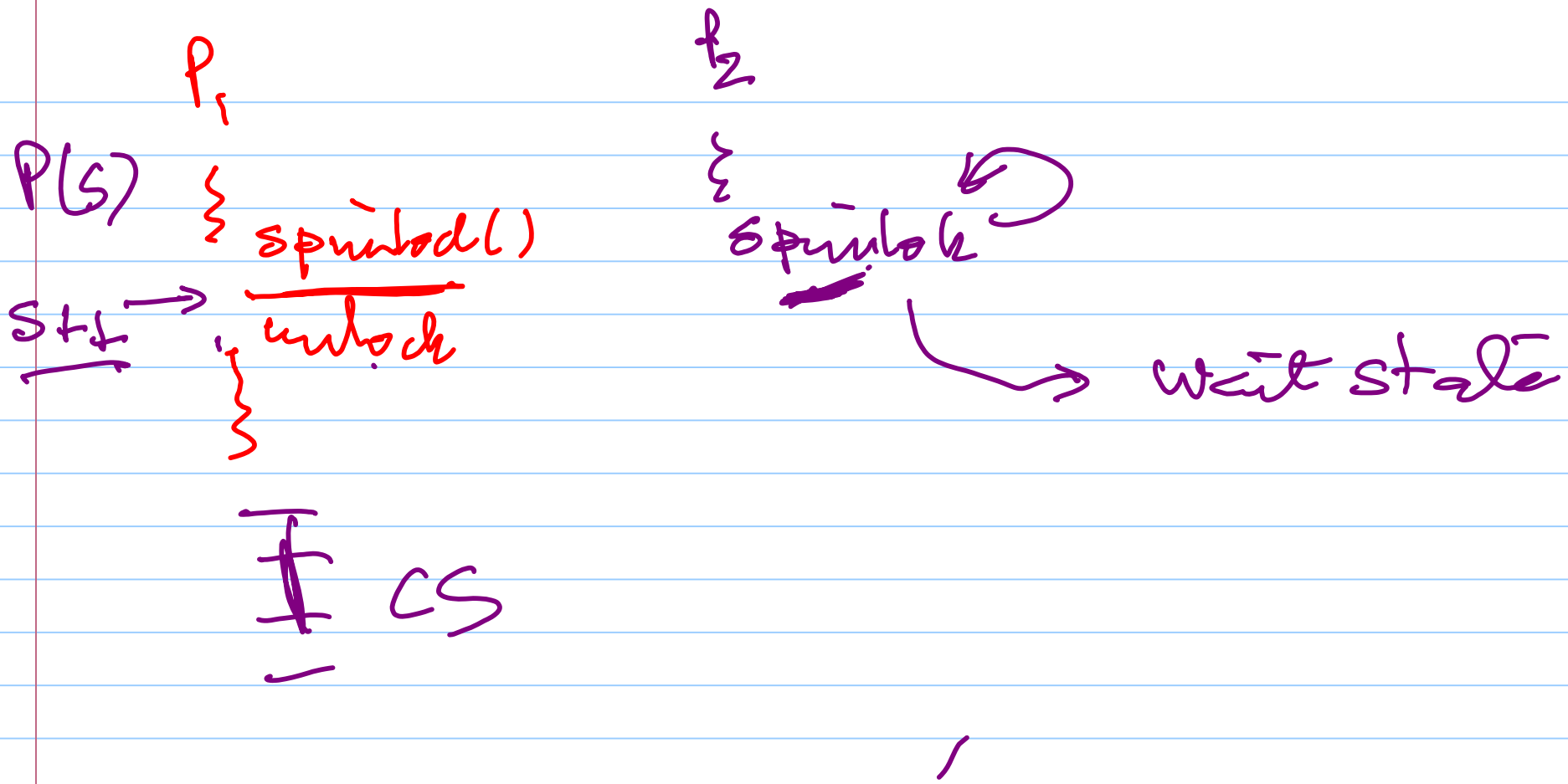
Remove current PCB from Ready Q

Add it to s.Q

context switch


V(S) {
Spinlock (S.lock)
S.count ++
if (S.count <= 0)
{remove 1 PCB from S.Q & add to Ready Q }
sp_unlock(S.lock);
unblock 1 process from S.Q


S.count = 1
P1
P2
P3
P4
P5
other code
P(S)
V(S)
P2 → S.count → -1 → block on S.Q
P3 → -2 "
P4 → -3 block
bounded waiting


P1
P(S)
spinlock()
unlock
stt
CS
t2
spinlock
wait state

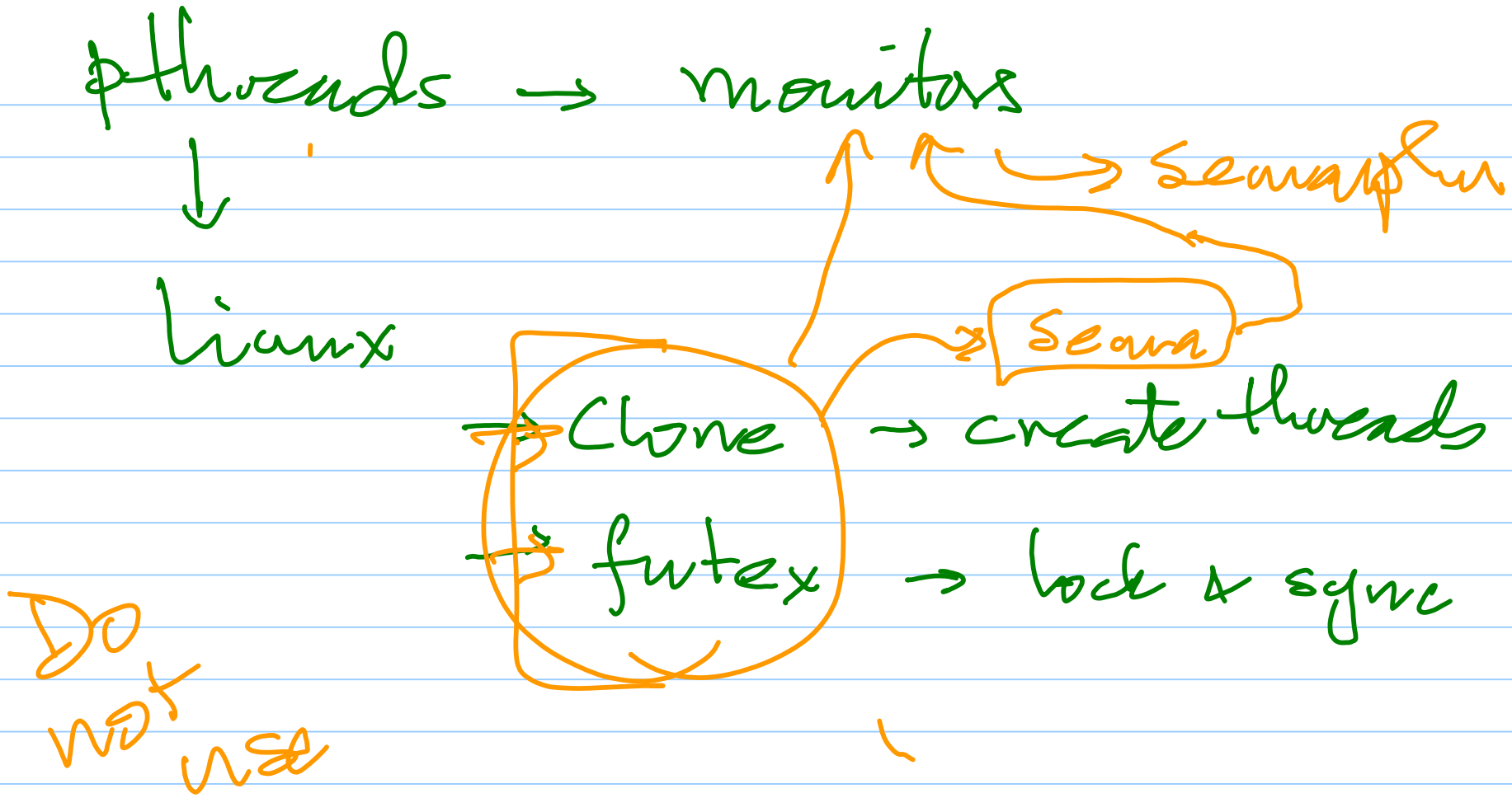
multicore, mutex semaphores

(single core → replace lock by Disable)

↓

Can be used as mutex locks

" " " synch methods


pthreads → monitors
↓
linux
semaphore
sem
clone → create threads
futex → lock & sync
Do not use