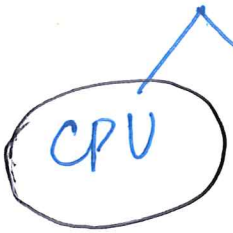
CPU Virtualization

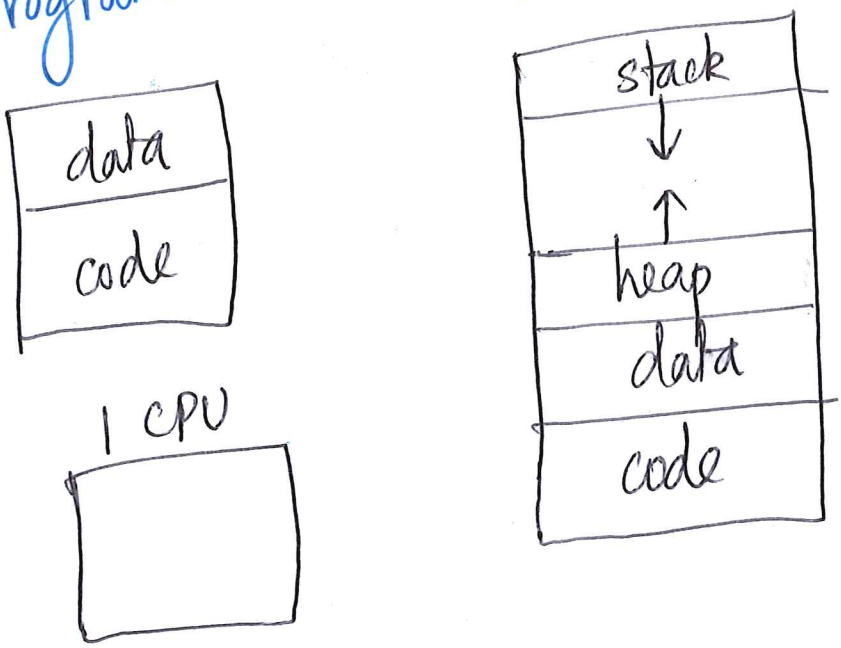
Review

- OS
- Virtual , Concurrency, Persistence

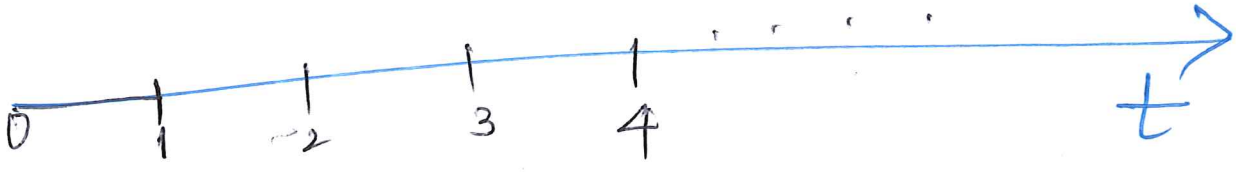


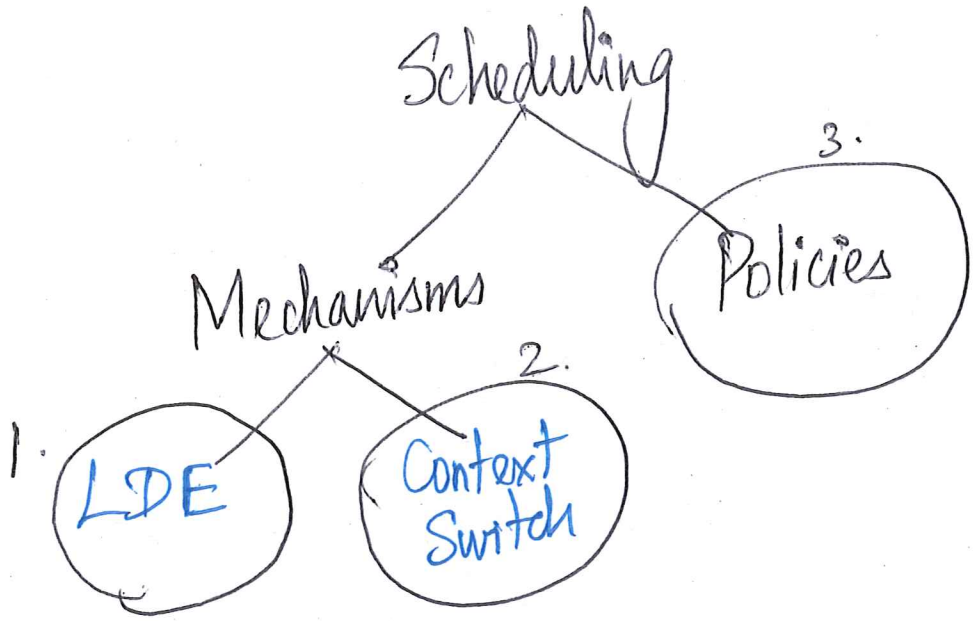
- Process - machine state ⟹ PCB.

- Program vs Process



CPU : P_1 P_1 P_2 P_2



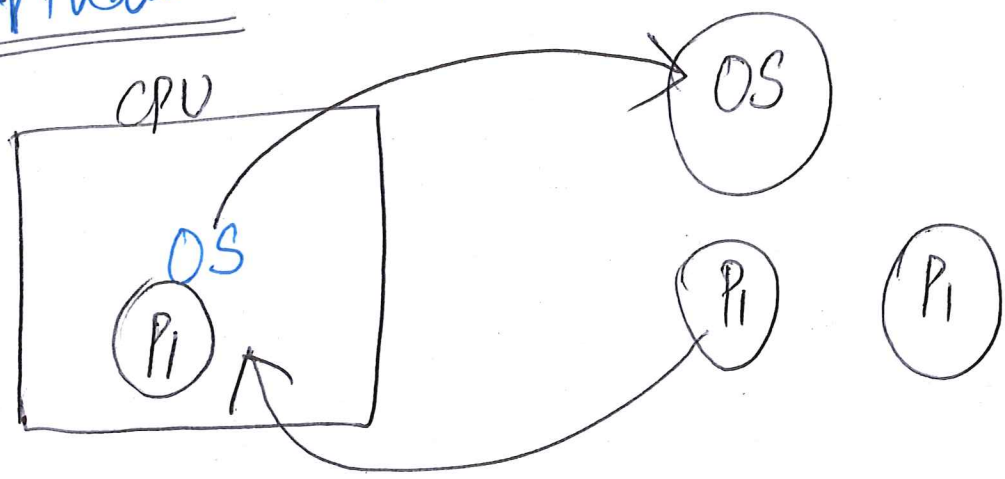


Scheduler – component of OS.

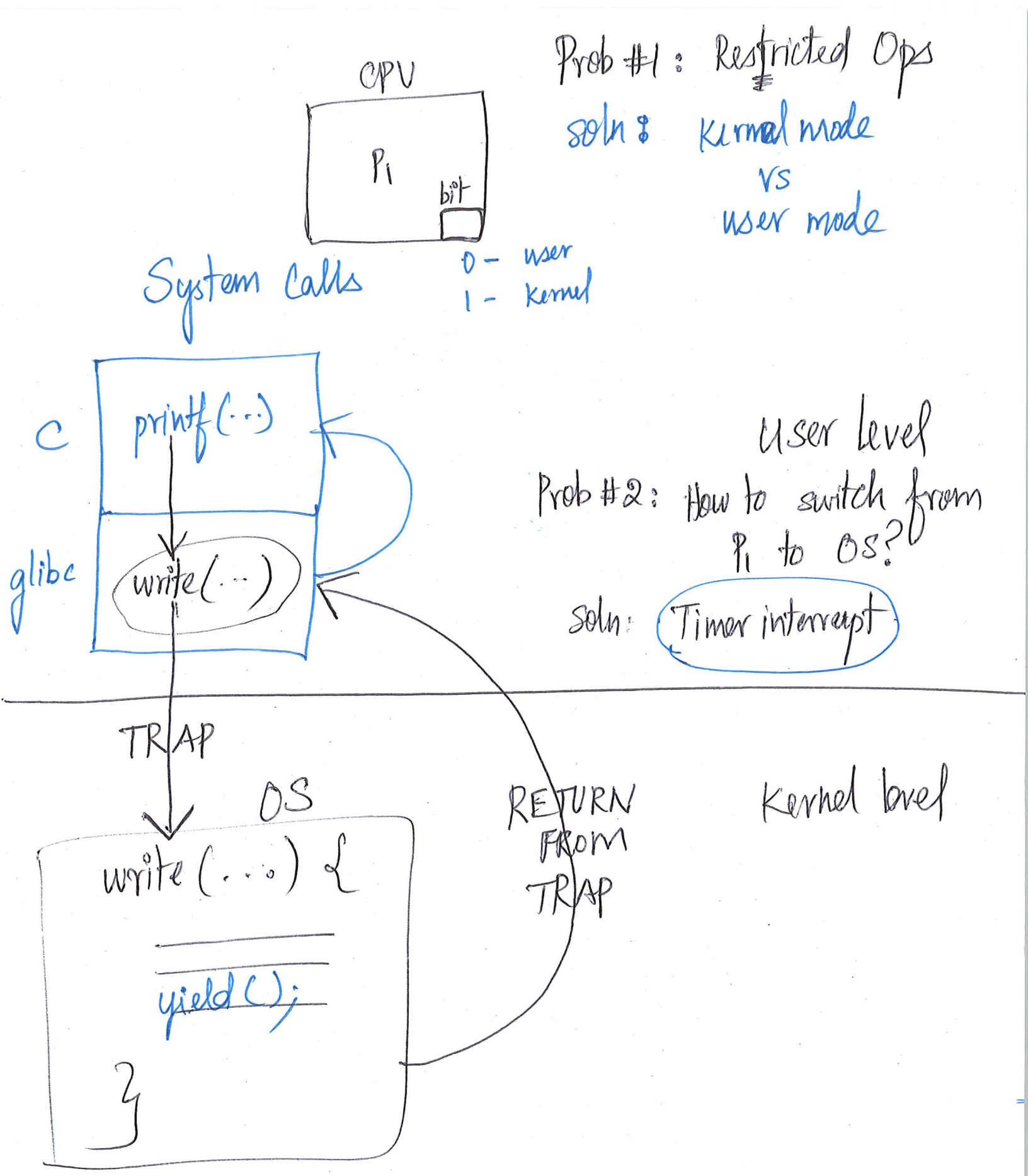
LDE

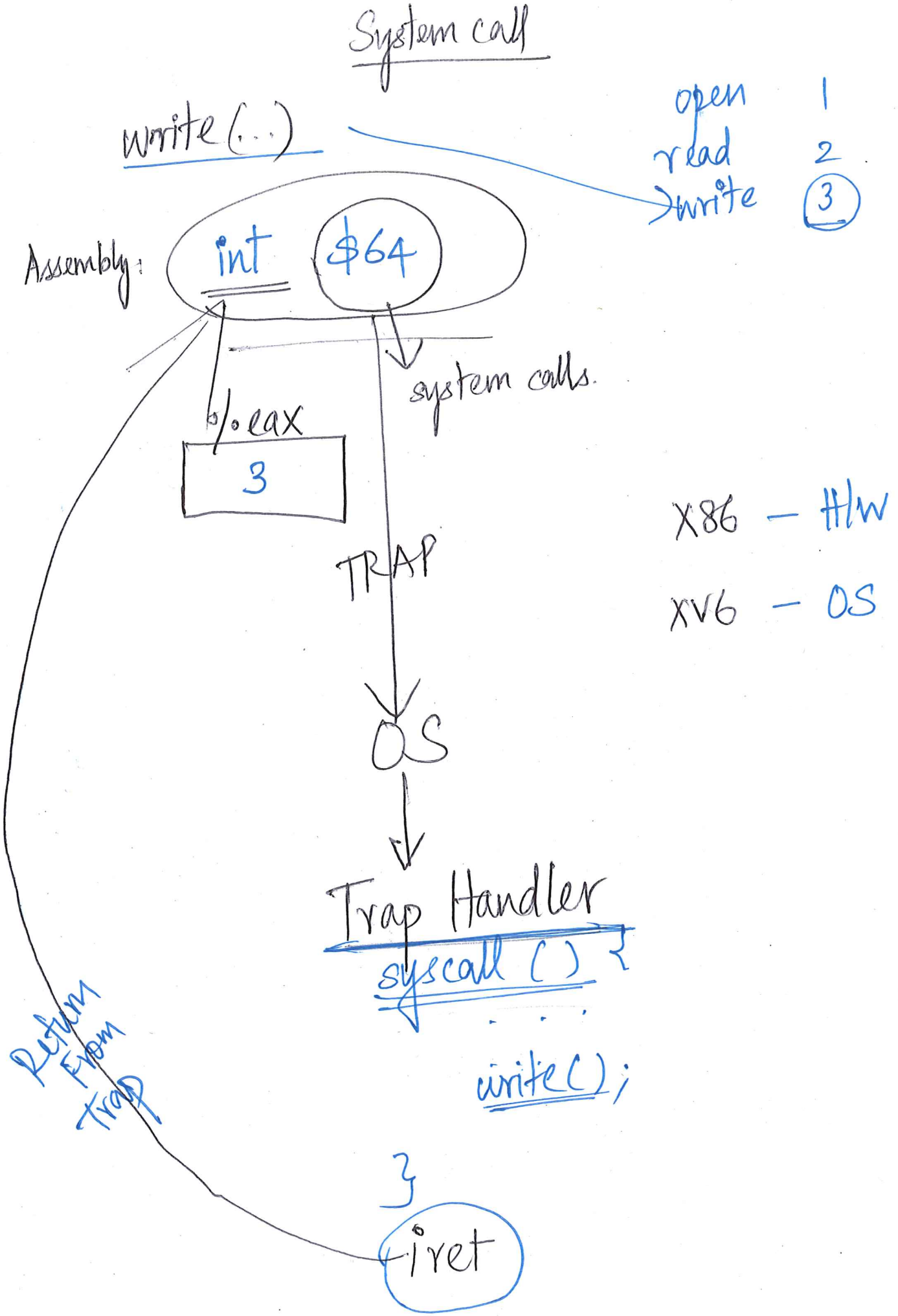
Direct Execution.

Q: How can ~~we~~ the OS efficiently virtualize the CPU?

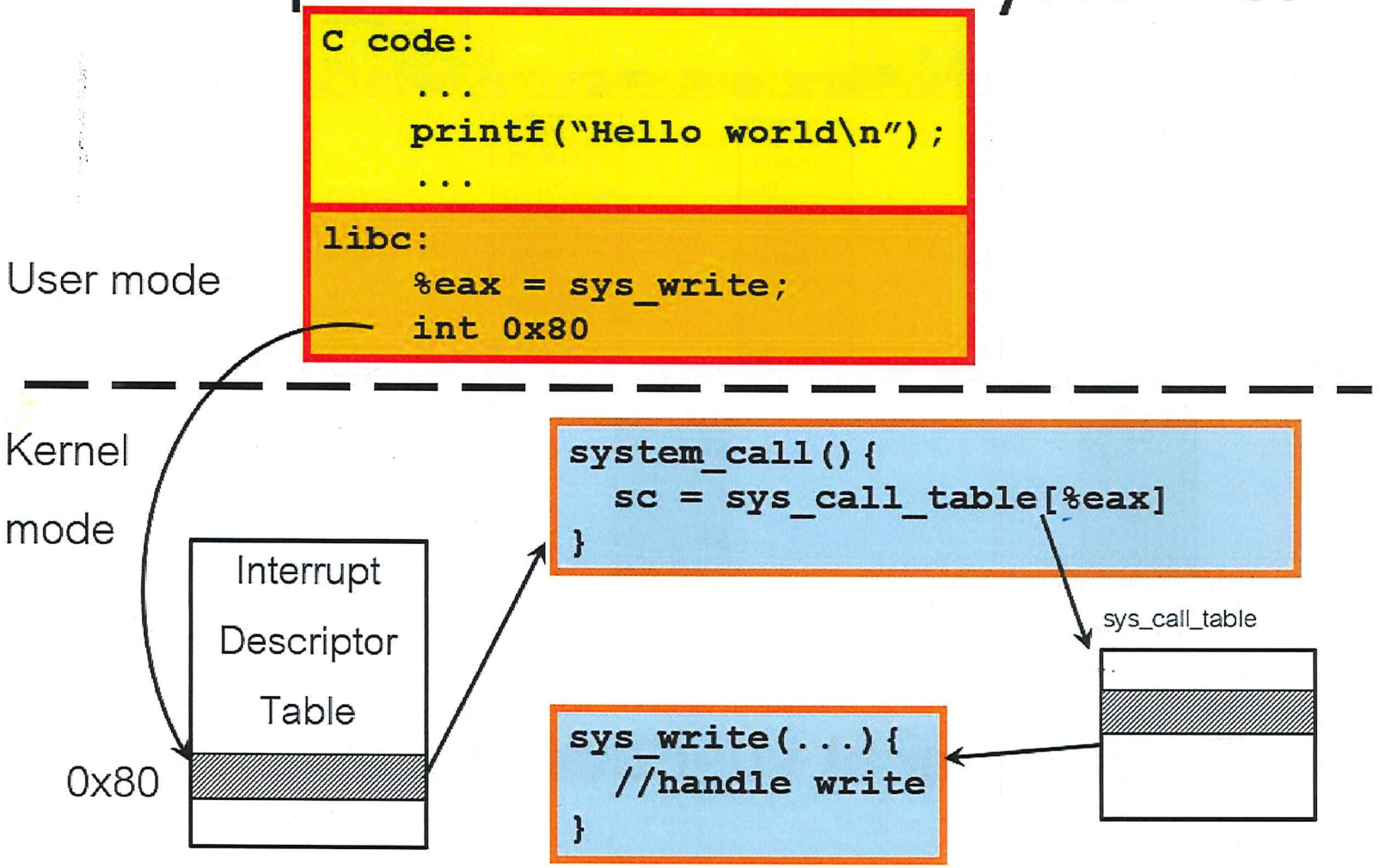


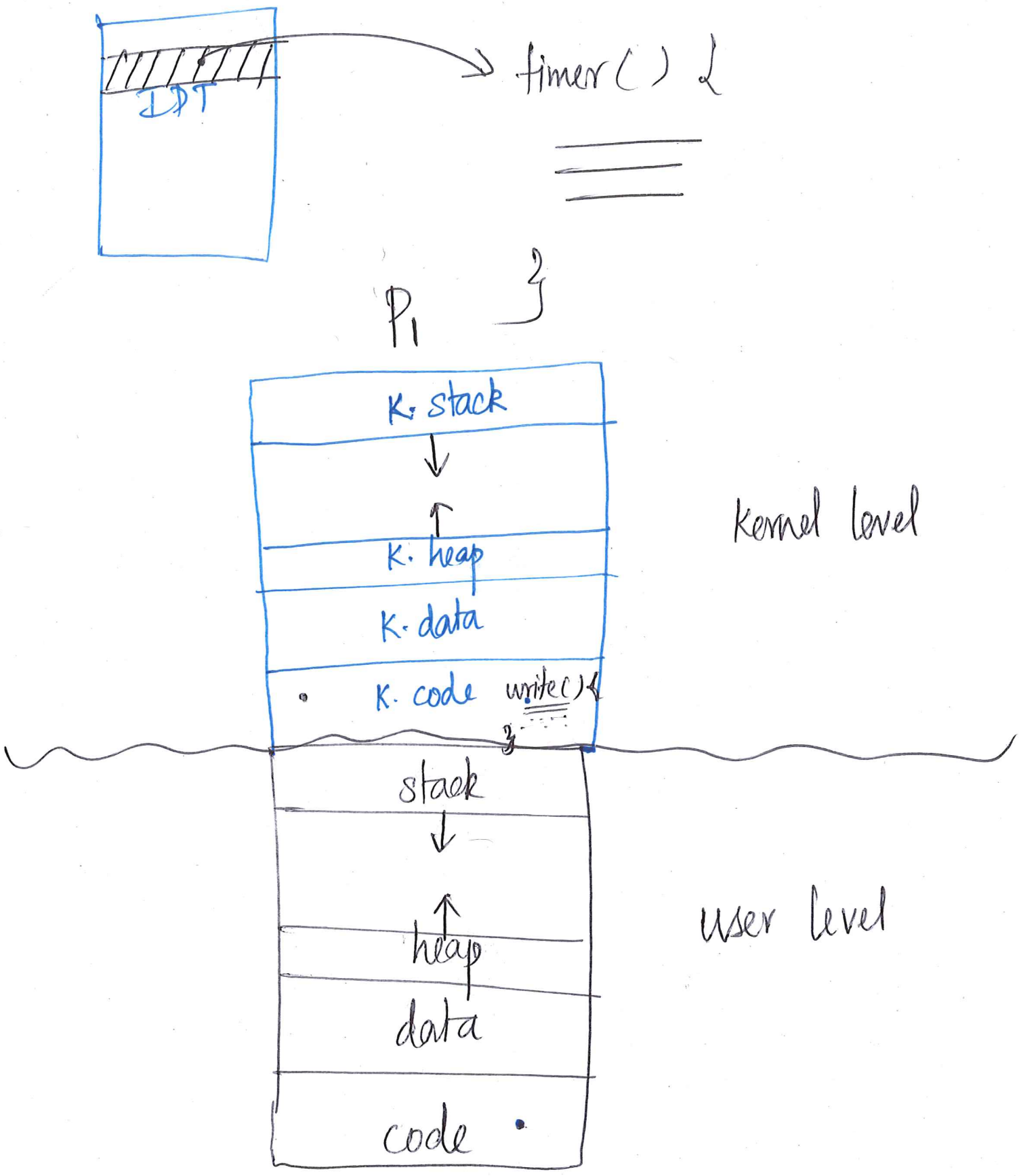
OS P_1 OS P_2 OS P_1 ....


CPU
P_i
bit
0 - user
1 - kernel
System Calls
C
printf(...)
glibc
write(...)
TRAP
OS
write(...) {
yield();
}
RETURN FROM TRAP
Prob #1: Restricted Ops
soln: kernel mode vs user mode
User level
Prob #2: How to switch from P_i to OS?
soln: Timer interrupt
Kernel level


System call
write(...)
open 1
read 2
write 3
Assembly:
int $64
%eax
3
system calls.
TRAP
OS
Trap Handler
syscall() {
. . .
write();
}
iret
Return From trap
X86 - H/W
XV6 - OS

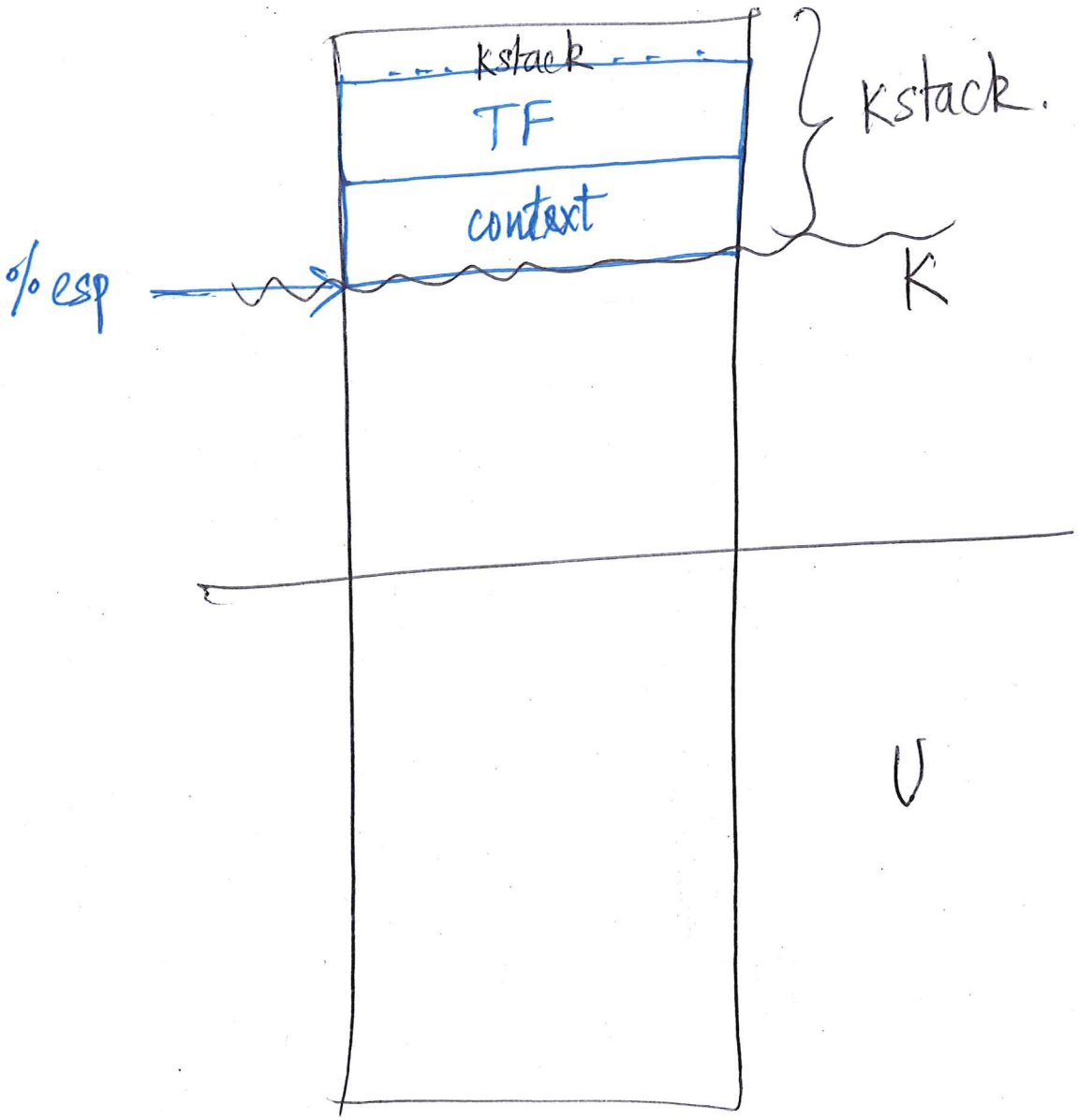
Example: Linux "write" system call

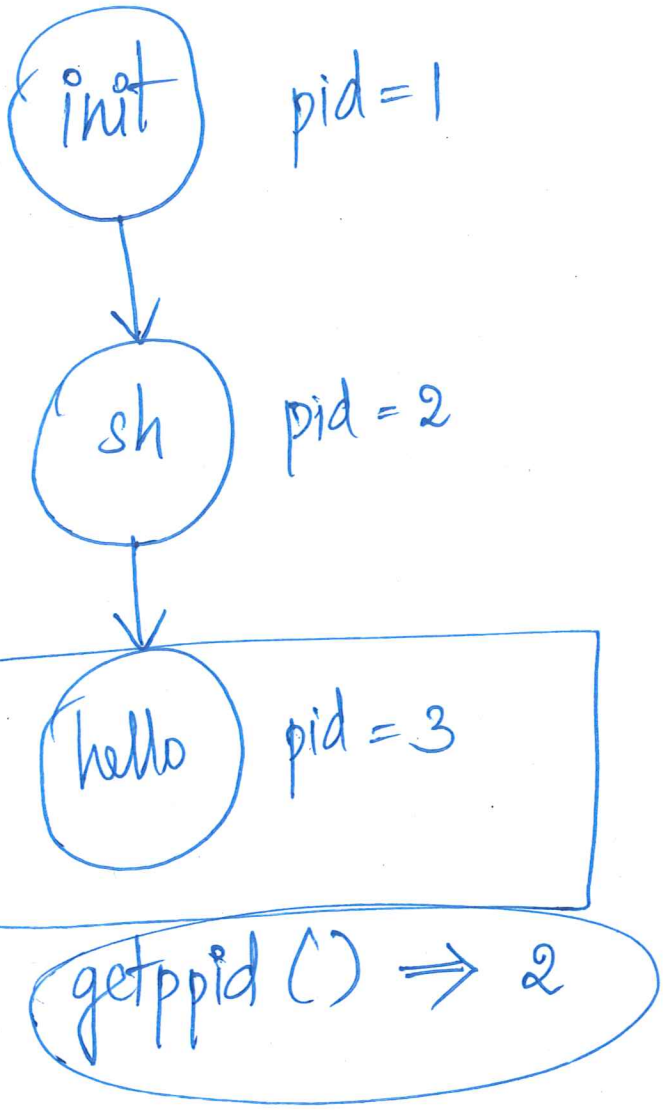


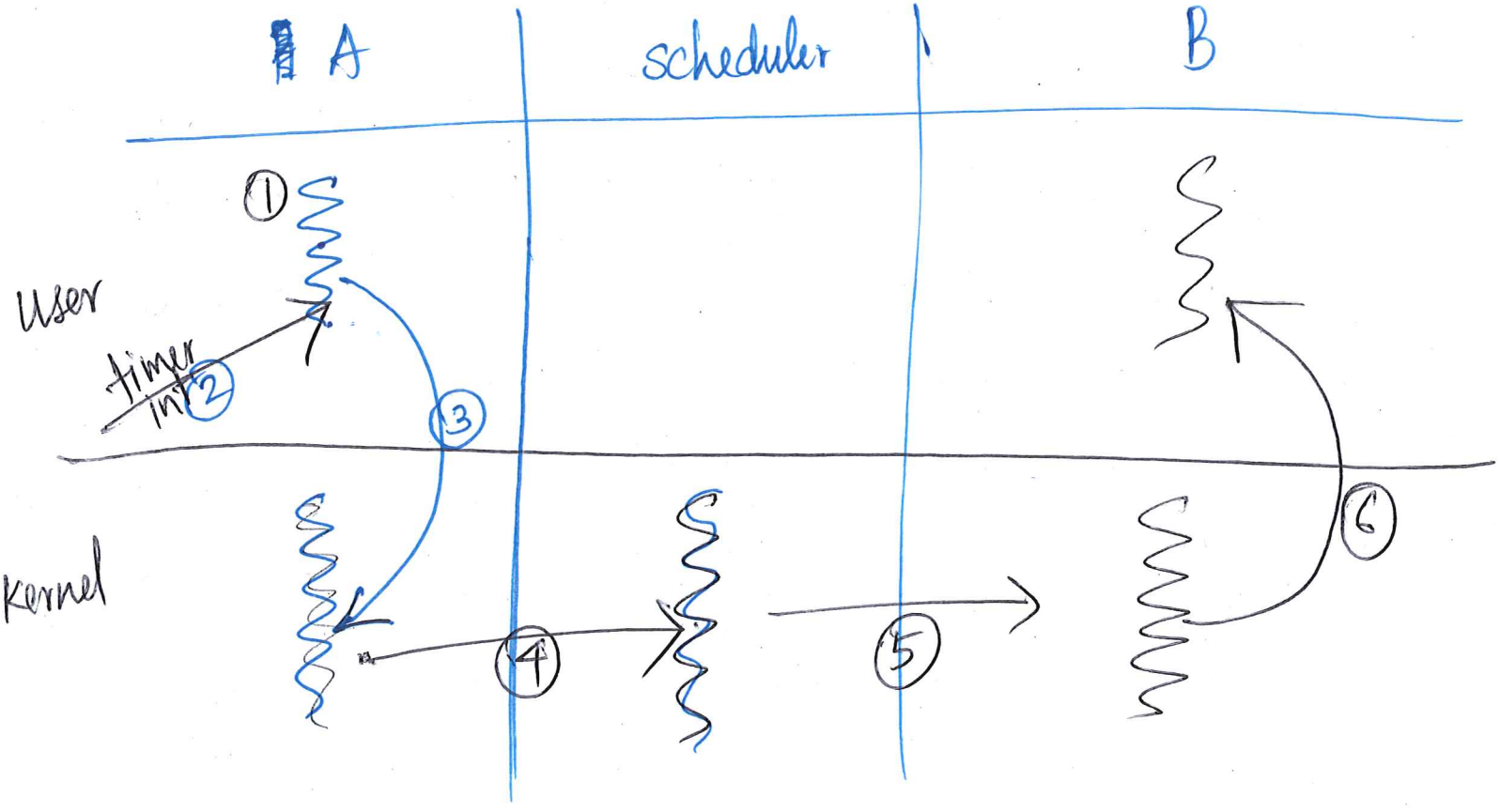

IDT
timer( ) {
P1
K. stack
K. heap
K. data
K. code
write( ) {
stack
heap
data
code
Kernel level
user level

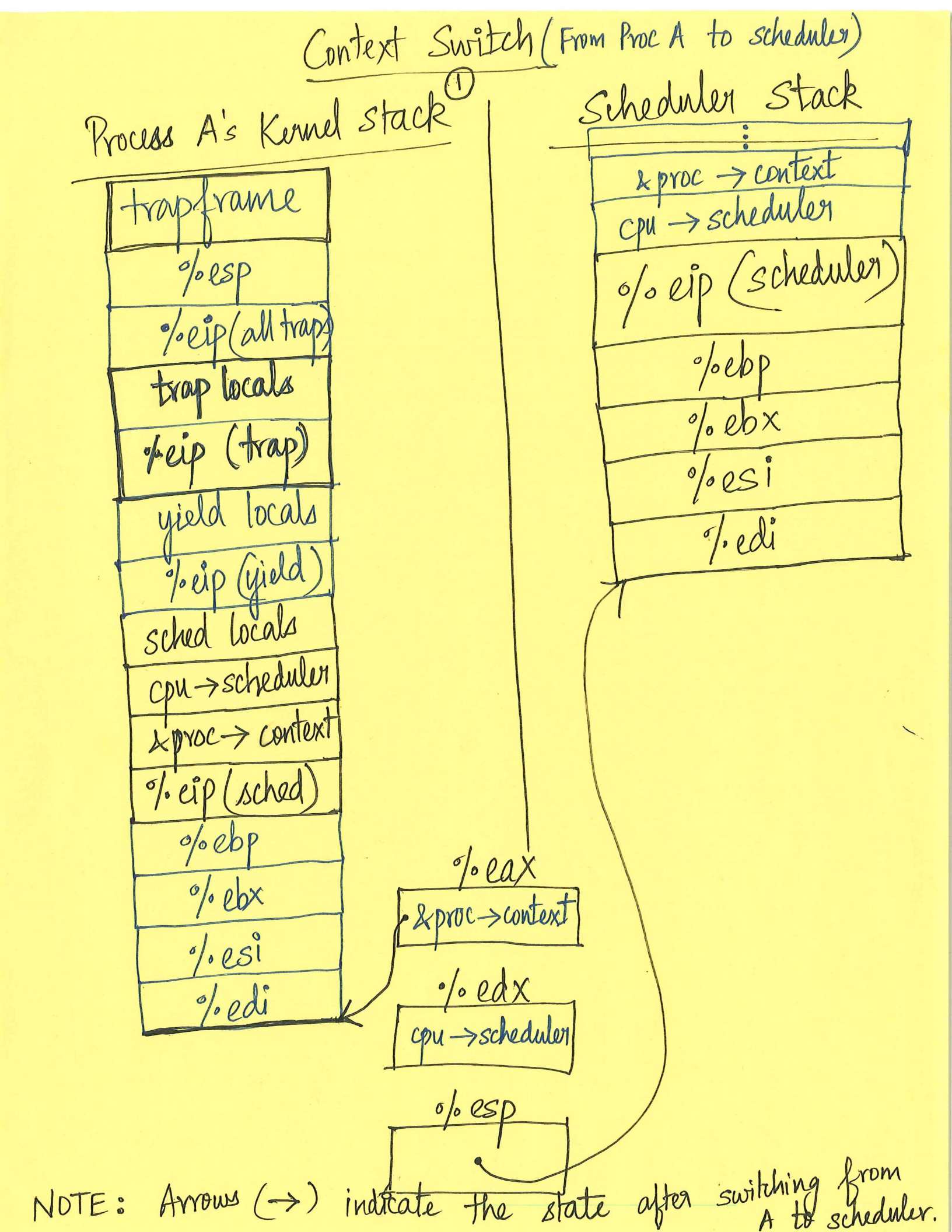
HW → user level regs of P → Trap Frame

OS → kernel level regs of P → Context



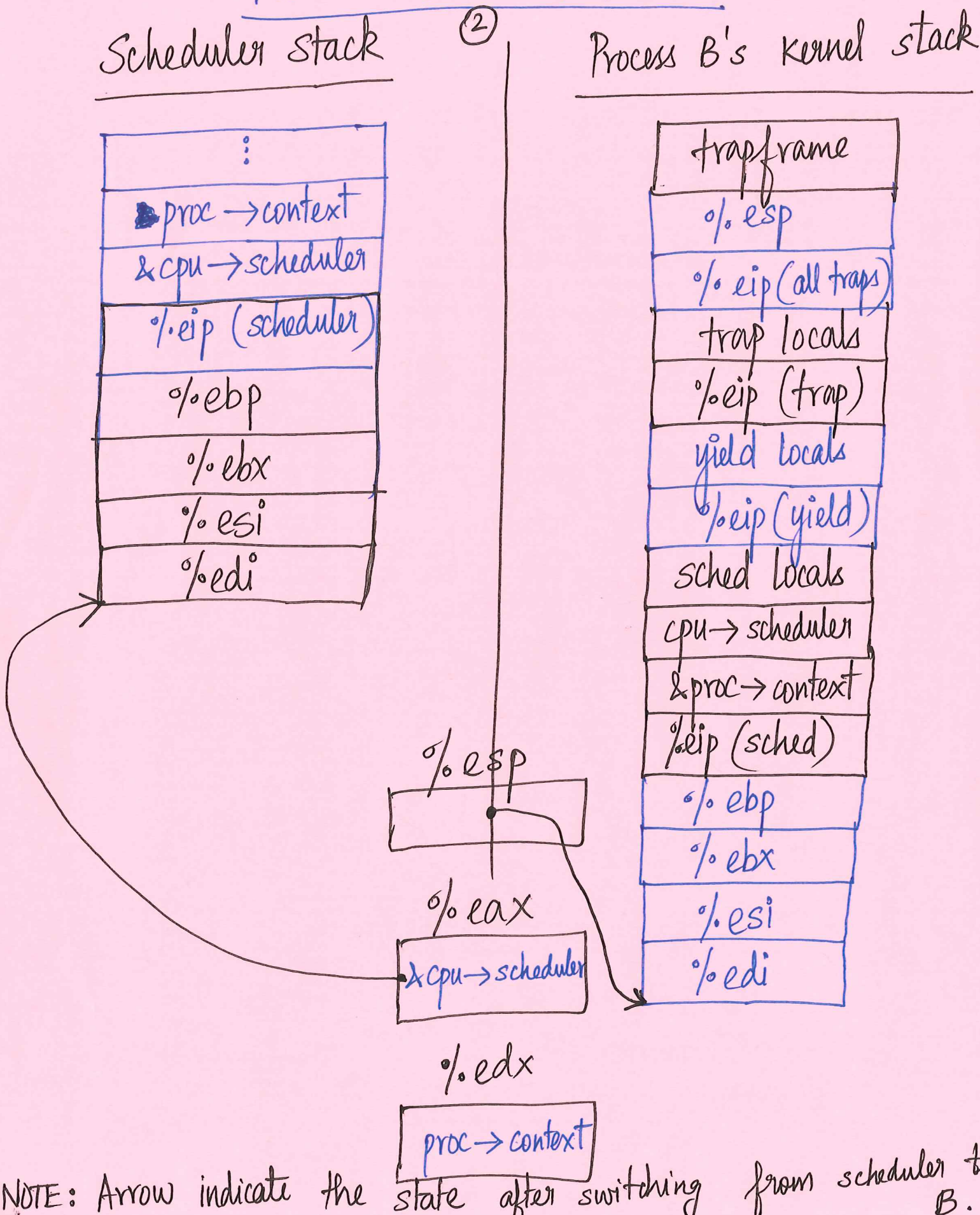

init
pid = 1
sh
pid = 2
hello
pid = 3
getppid () ⇒ 2


A
scheduler
B
User
kernel
timer int
1
2
3
4
5
6


Context Switch (From Proc A to scheduler)
1
Process A's Kernel stack
trapframe
%esp
%eip (all traps)
trap locals
%eip (trap)
yield locals
%eip (yield)
sched locals
cpu->scheduler
&proc->context
%eip (sched)
%ebp
%ebx
%esi
%edi
Scheduler Stack
&proc->context
cpu->scheduler
%eip (scheduler)
%ebp
%ebx
%esi
%edi
%eax
&proc->context
%edx
cpu->scheduler
%esp
NOTE: Arrows (→) indicate the state after switching from A to scheduler.

Context Switch

From scheduler to Proc B



NOTE: Arrow indicate the state after switching from scheduler to B.