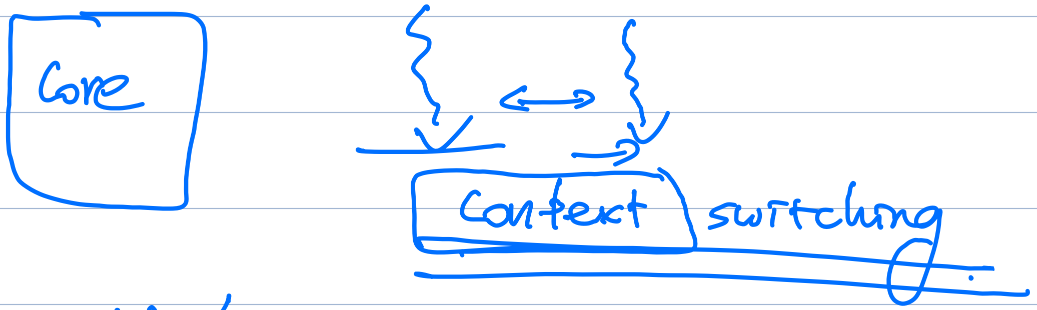
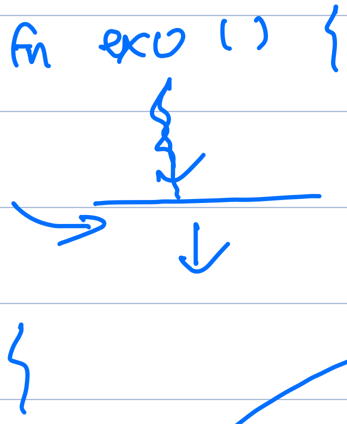
* Context switching.



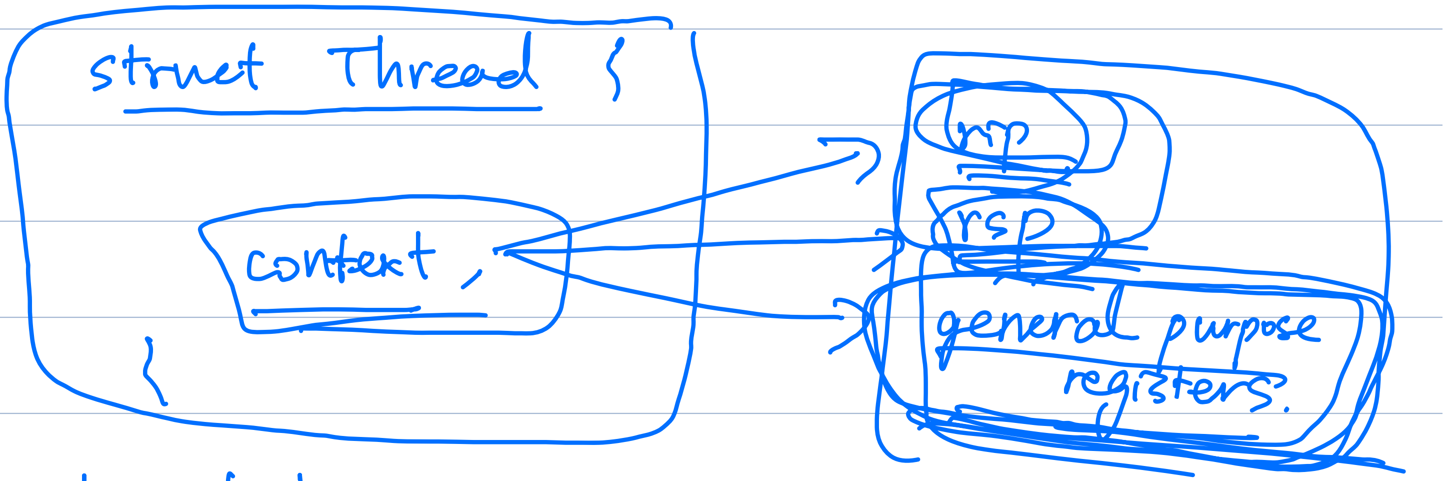


program counter?

- → rip ⇒ next instruction to execute.
- → rsp ⇒ stack.
- → general purpose regristers

"state" = context

save these in memory so that we can retrieve them later & resume the execution.



kernel heap
stack

* Context switching by an interrupt.

→ timer interrupt

① regrster a timer interrupt handler to IDT

⇓

timer interrupt handler ↙ stack for the handler (this is where rip. rsp. et. gott stored.)

⇒ save the context (state)

⇒ pick the next thread to run. (scheduling)

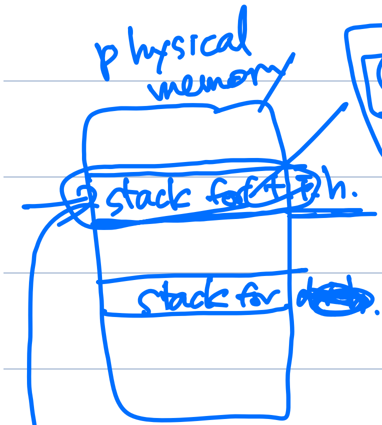
⇒ restore the state for the next thread. (modify IDT to provide new stack space for the new thread)

⇓

signal interrupt handling is over.

⇓

CPU runs from rip



rip
rsp
a few other registers) automatically done by the CPU.

different rip.

struct Thread {

rip
rsp.

stack (interrupt handler stack)

}

rip. rsp. + general purpose registers.

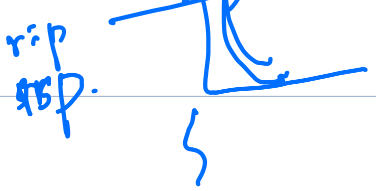
CPU stores what ~~we~~ store. software(kernel)

struct Thread {

4096 field:

rip
rsp

CPU stores automatically using the handler's

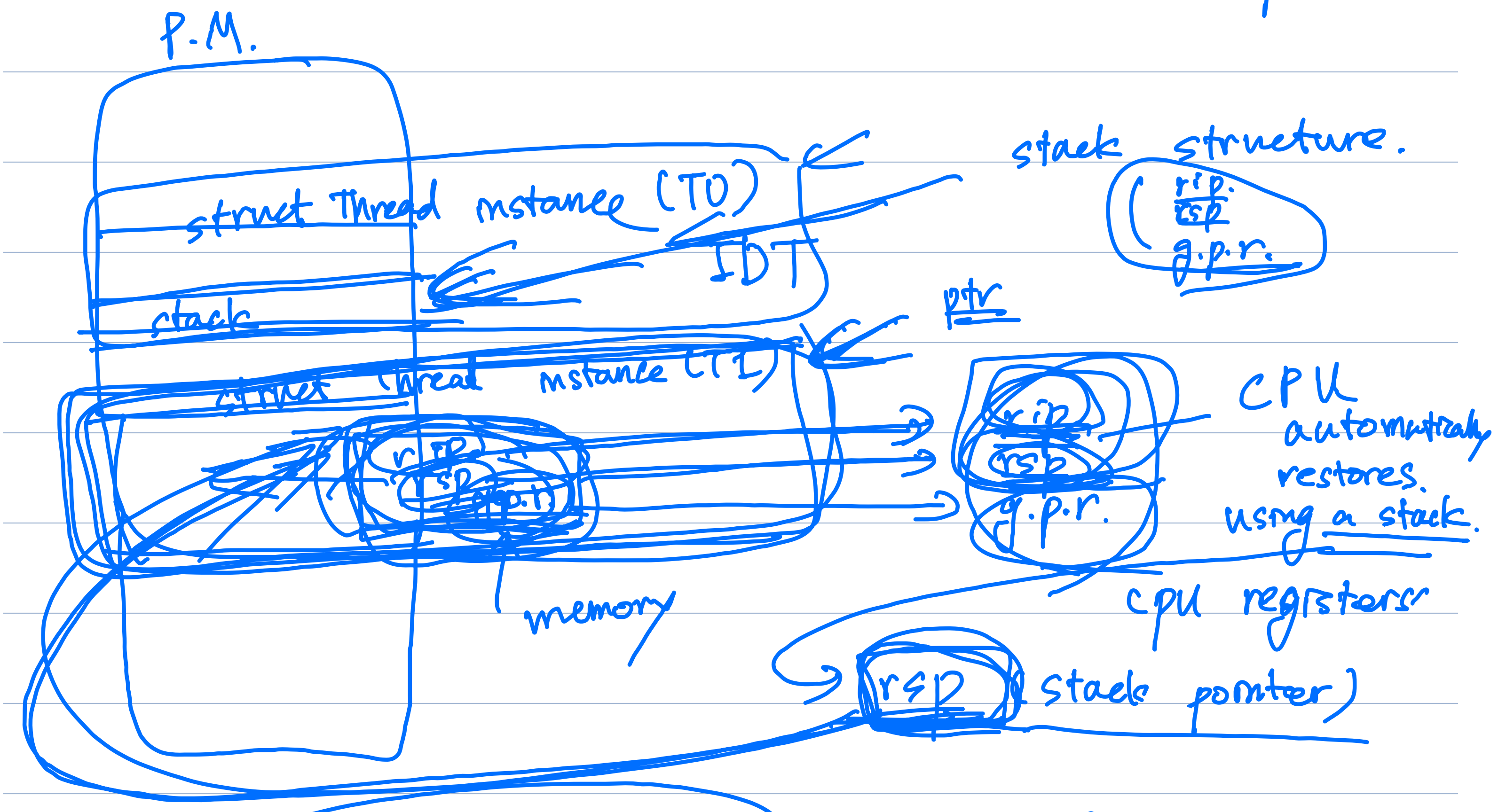


g.p.r.

stack space.

provide stack space ~~to~~ for the interrupt handler.

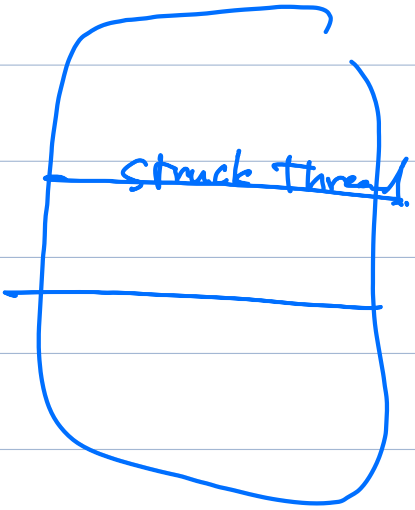
IDT => provide a new stack space



CPU automatically restores. using a stack.

struct Thread{
stack

Box::new(Thread::new( ));



CPU
T0
rip
rsp
rip
rsp
g.p.r
T1
rsp