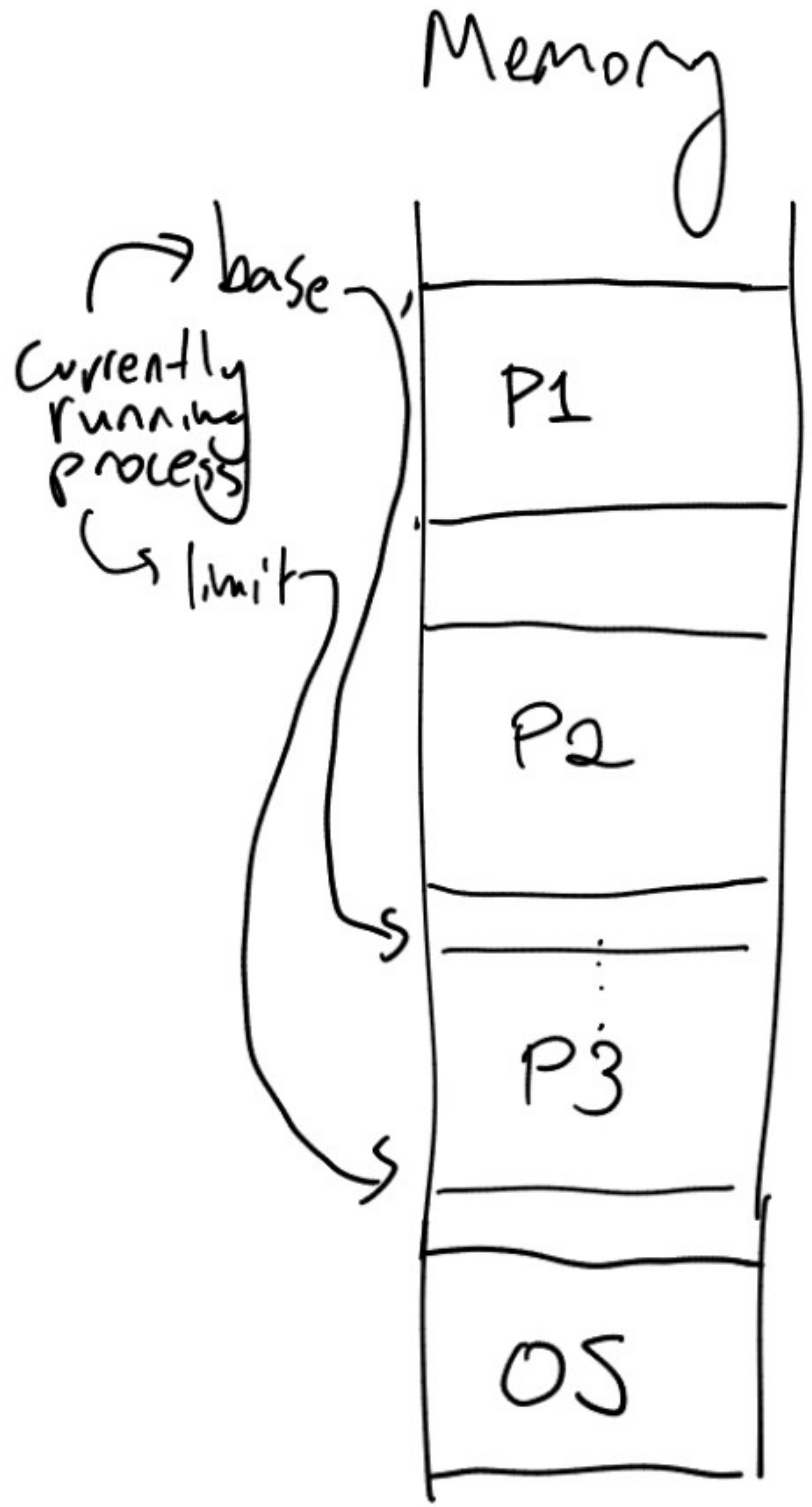
Lecture 3: Processes

- Context switching. PCBs
- Initialization
- Traps (exceptions, interrupts, syscalls)
- Privilege

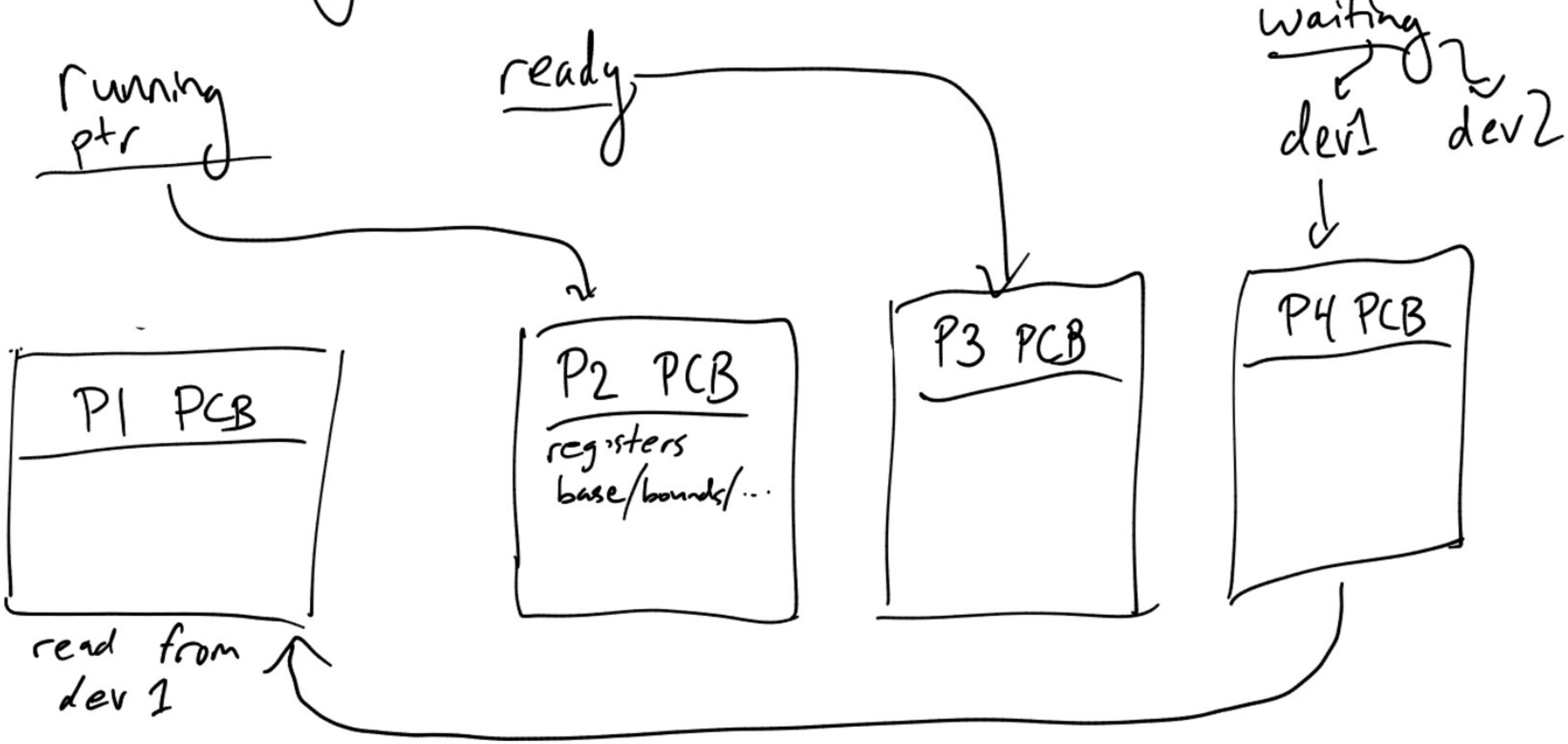
Need to save data about process:

Process control block (PCB) (data structure) stored in OS's region of memory

- memory (base & limit registers)
- registers (IP & SP & ...)
- processor flags.



OS's memory region



Context switch: changing from 1 proc. to another

- update state of PCBs
- load regs for new proc
- jump to IP of new proc.

Need privileged instructions to isolate applications

- Change base/bounds
- Modify int. vector

Privilege bit is a flag in the processor
- if clear: priv. instructions cause exceptions instead of executing.

Terminology: "kernel mode" (priv.) vs. "user mode" (unpriv.)
- also called "superuser mode" or "ring 0"
- not to be confused with "root" or "administrator"

When to set/reset priv. bit?

initialization: want priv. bit set.

- interrupt
- process termination
- HW exceptions
- request from processes

set priv. bit

system calls

to write OS, need to write 4 functions

traps: anything that causes priv. to get set.

- interrupts: cause branch to IH, set priv. bit
- system calls: requests from applications to OS
  cause a branch to syscall handler routine: location stored in ded. register.
  sets priv. bit.
- HW exceptions: causes branch to exc. handler routine
  set priv. bit.

priv. bit cleared during "return from syscall" instruction

(not necessarily a "return")

whenever priv. bit becomes high, must jump to predetermined address (inside kernel) to prevent abuse (thus "trap")

Init. routine:

- Set up IV, syscall handler, exc. handler
- Set up devices
- set up "init" process
  ↳ read config., launch other processes
- "return" from syscall → init process.

Syscall handler:

- just like a fn call:
  - save callee saved registers
  - check permissions (eg. is process allowed to access device?)
  - do its I/O, update PCB state, ...
  - schedule another process (maybe)

Exc./Int handlers:

Similar to syscall handler, need to save all proc. state since exc/int is unexpected.

Each OS has a list of available system calls:

Monolithic kernel

- Syscalls application level requests:
  - open a file
  - read from open file
  - open network connection
  - launch a new process
  - wait for another process to end
  - exit (term. current process)

} interact with devices calls code in device driver.

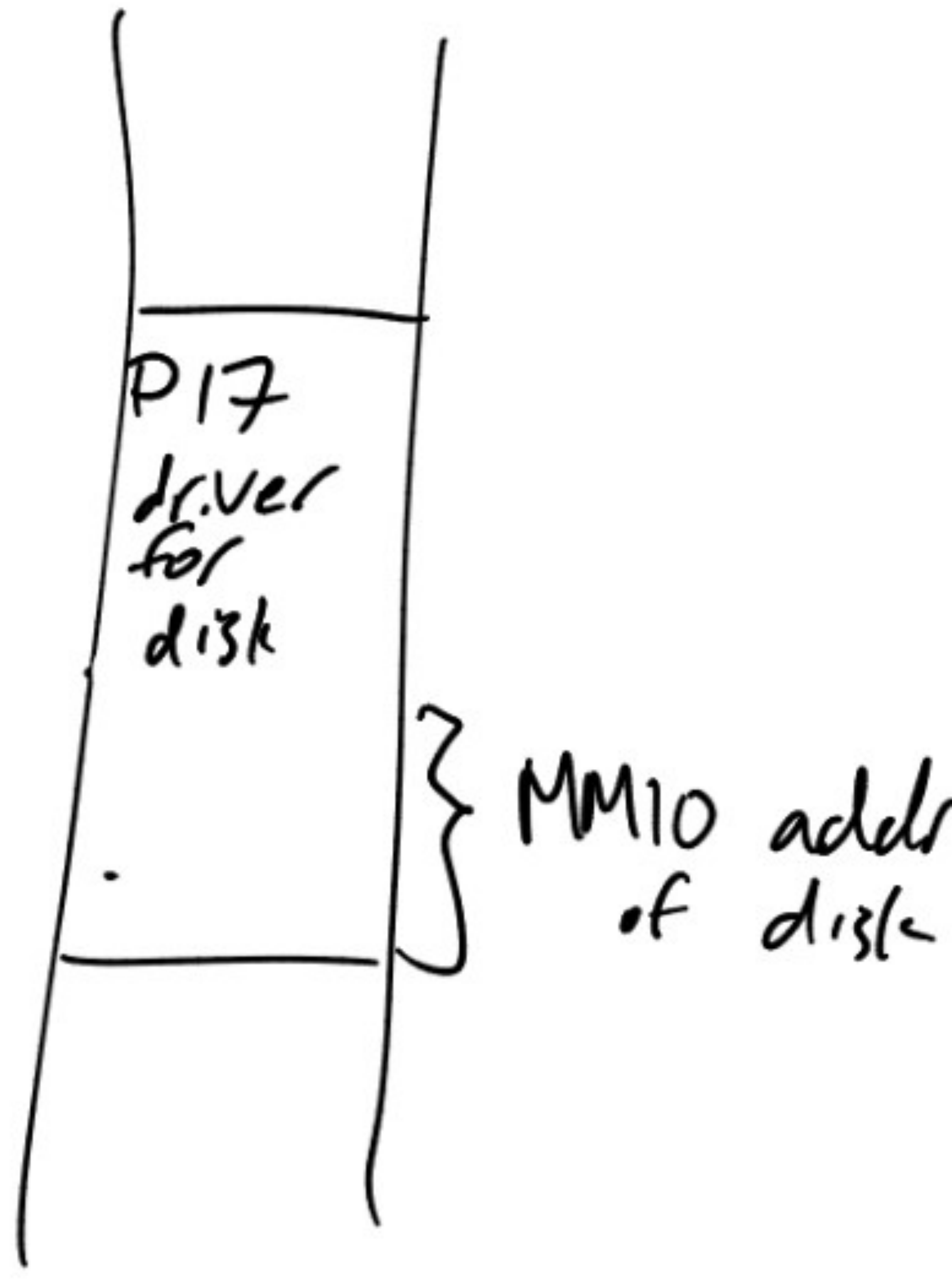
- large amount of kernel code
  " " " bugs

  kernel bugs are <u>bad</u>

Microkernel :

- only syscalls
  - process management
  - communicating between processes

    ex. "$\overset{\text{use}}{\text{send}}$ message" syscall to request the disk process to read from disk
- Some processes (drivers) need access to addresses corresponding to devices



Virtual machine manager (VMM)

- syscalls forwarded to another process (Guest OS).