

CS 3650 – Computer Systems
Spring 2024
Peter Desnoyers

Lecture 24, Thur Apr 4 2024

How system calls, FUSE etc work

```
main() {
```

```
    fd = open("file", O_RDONLY)
```

```
    char buf [size]
```

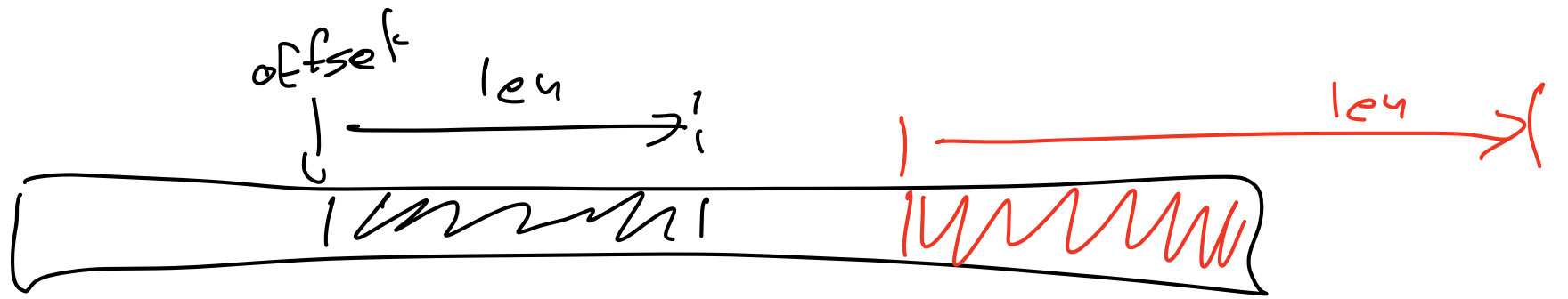
```
    while ((len = read(fd, buf, sizeof(buf))) > 0) {
```

```
        // do something  
        (data in buf)
```

```
    }
```



```
fs_read(path, buf, len, offset, ..)
```



Networking & sockets

client

server

socket → fd

connect



fd' ← accept

fd ← socket

fd¹
fd²
fd³

⋮

bind()

listen()

fd_set S

FD_ZERO (&S)

FD_SET (fd1, &S)

⋮
fd2
fd3

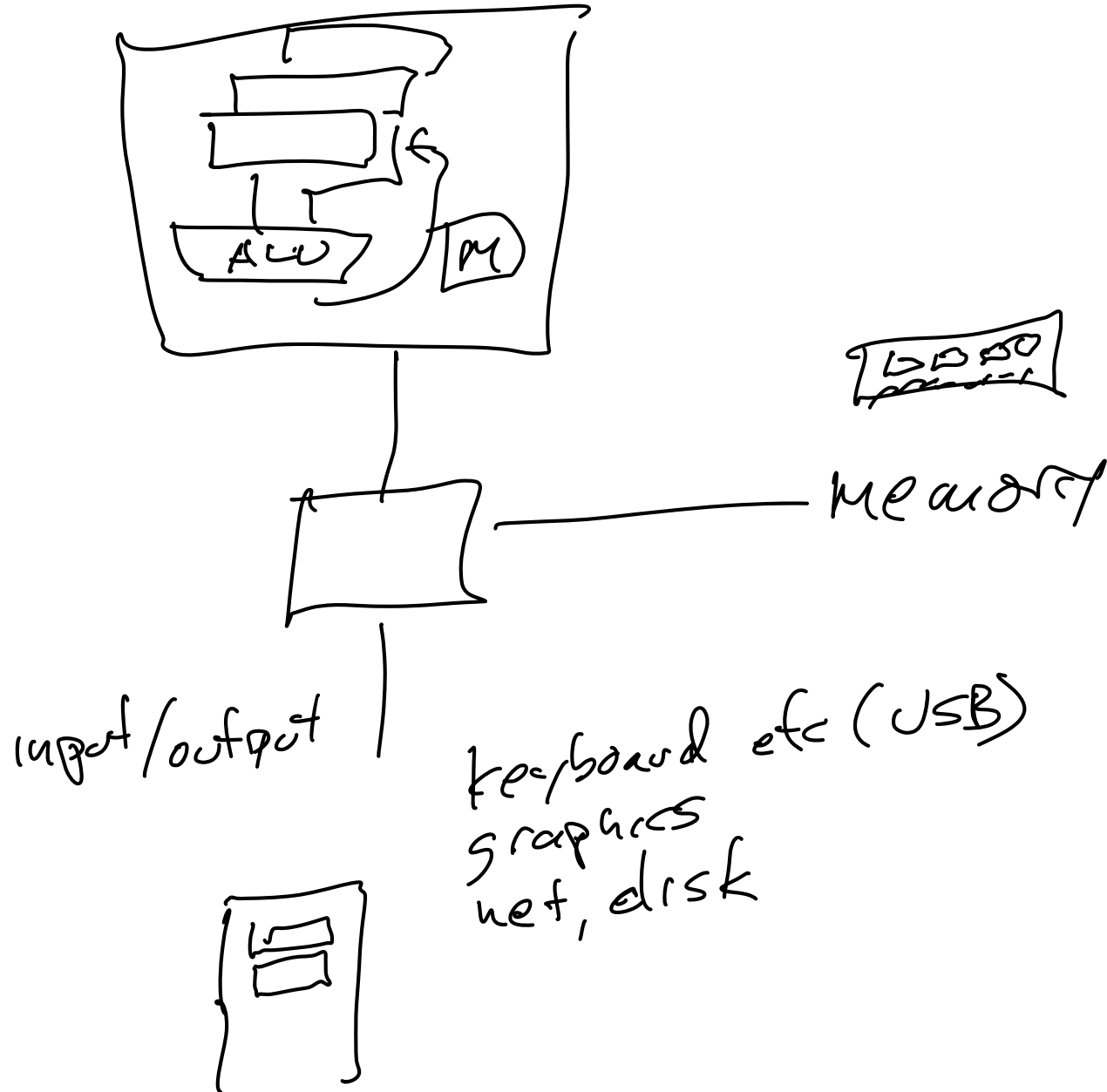
select (&S, NULL, ...)

if FD_ISSET (fd1, &S) ...

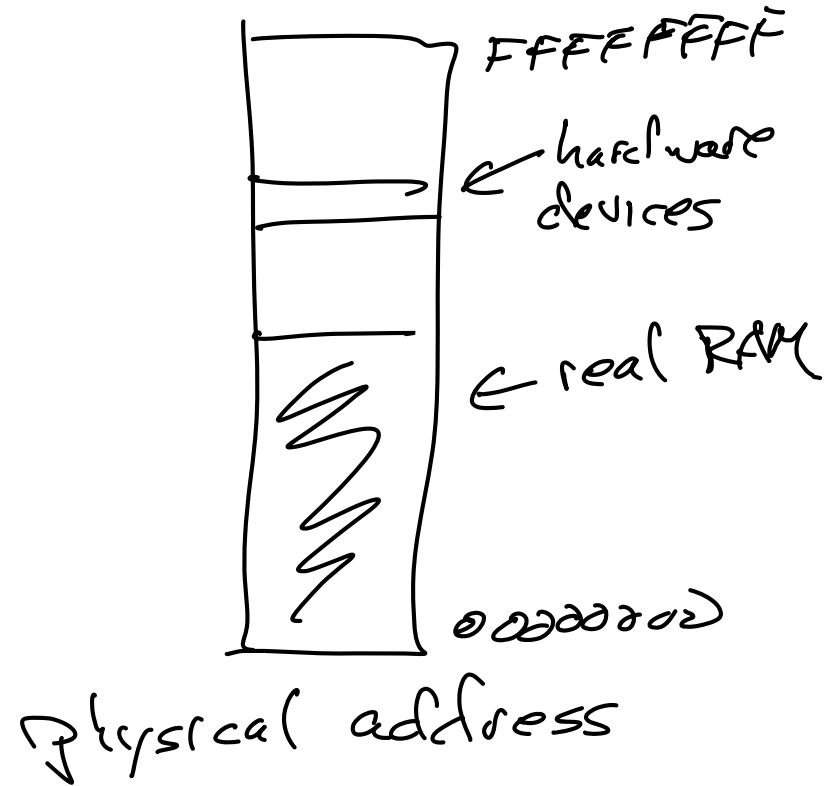
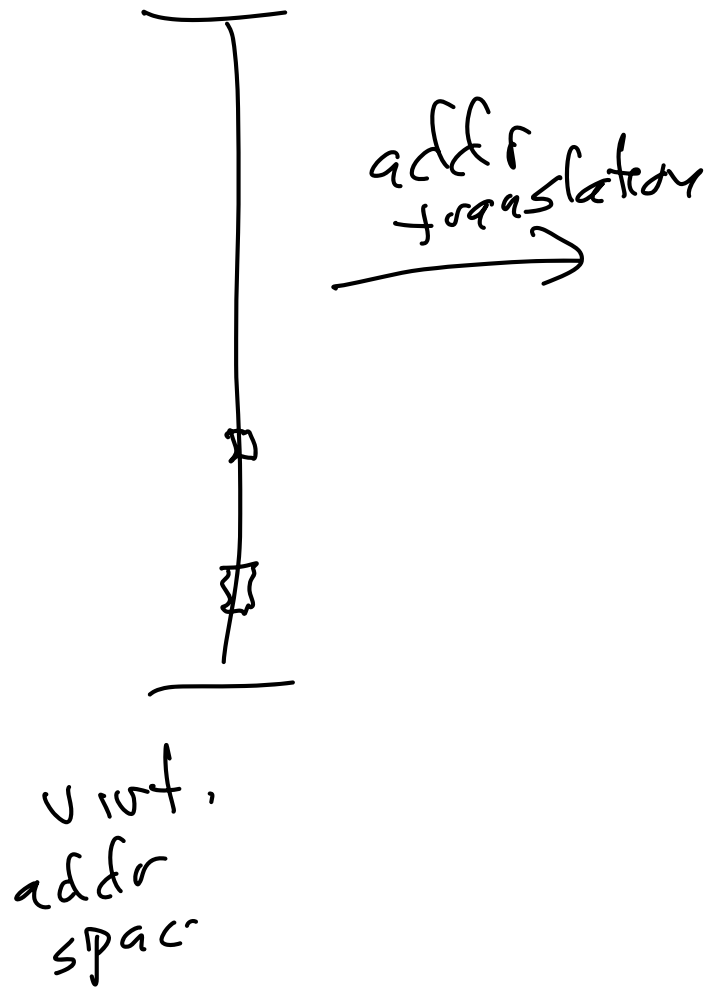
read (fd1, ...)

struct fd_set

Hardware & device drivers



Simple devices - programmed I/O



programmed IO (keyboard)

keyboard controller:

read key:

while ! * (status reg addr)

; // do nothing

return * (value reg addr)

status



value

Interrupts

MOV R1 ← R2

LOAD R3 ← *PTR

INSTR

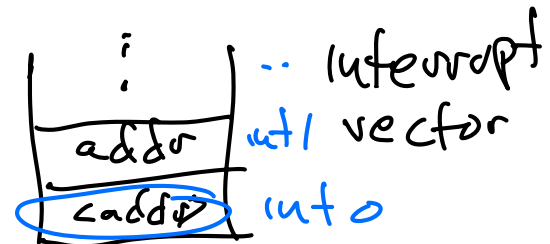
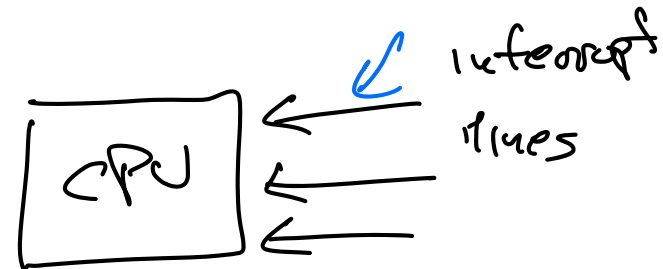
INSTR

⋮

→ interrupt
PUSH addr

JMP → int handler

⋮
ret



queue of bytes Q

int handler:


read status \rightarrow tmp
push tmp onto Q

read key:

wait until Q not empty
return Q.pop()

 status

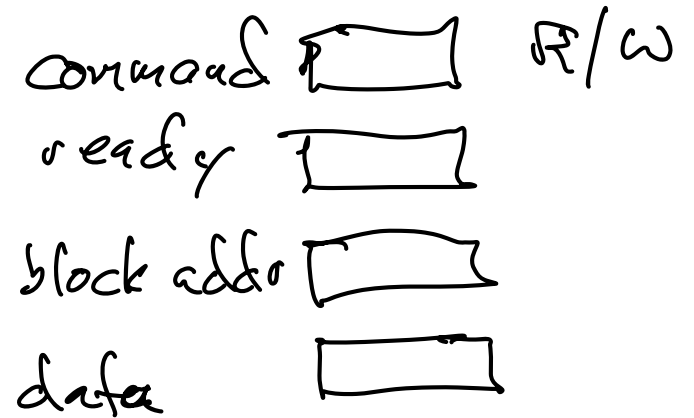
 code

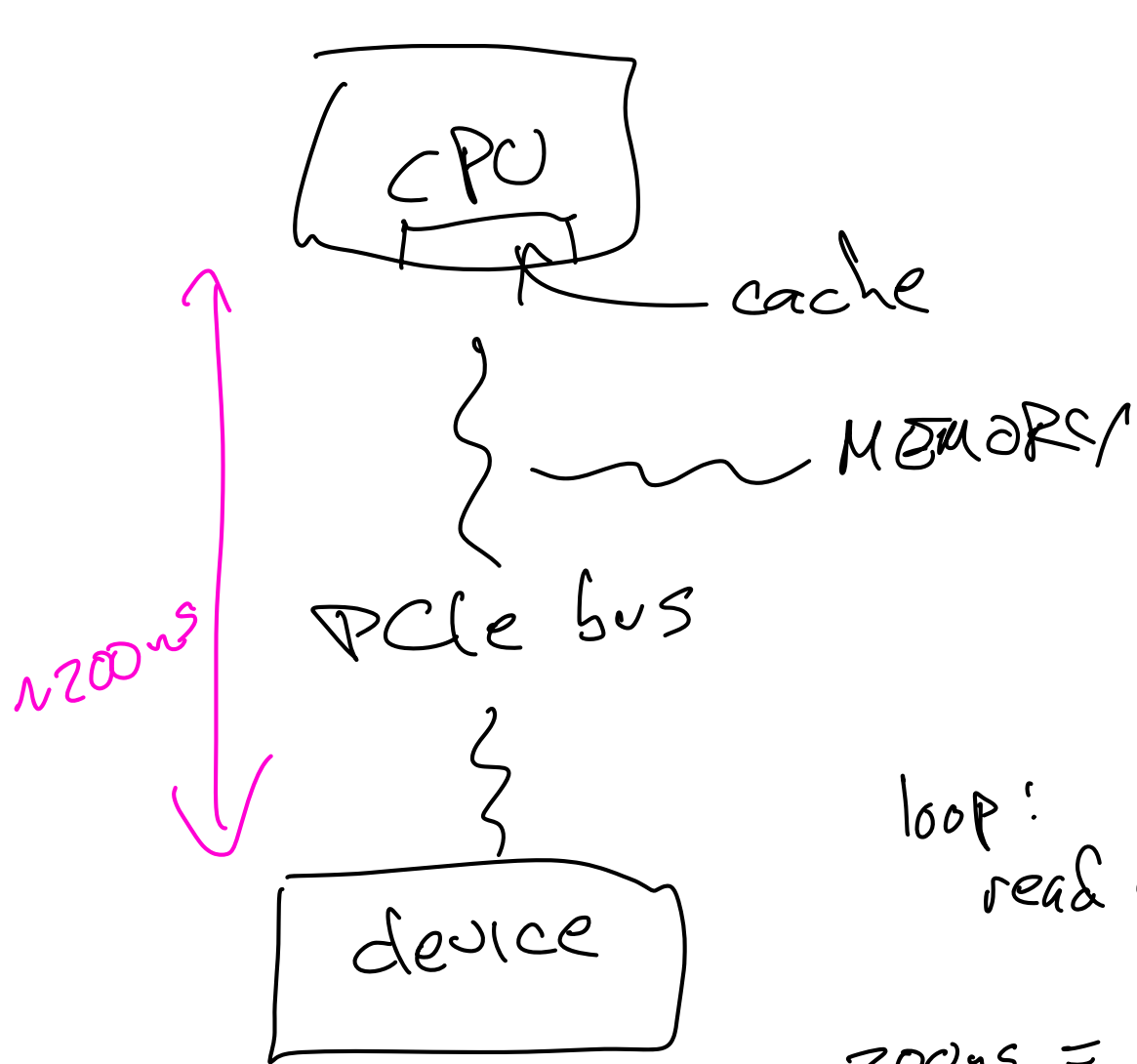
 interrupt

timer interrupt

really simple (slow) disk I/O

```
read : *block_addr = <A>
      *command = READ
      while ! *ready
        wait
      for i in 0..bytes-in-sector
        byte = *data
```





dealing with
latency!

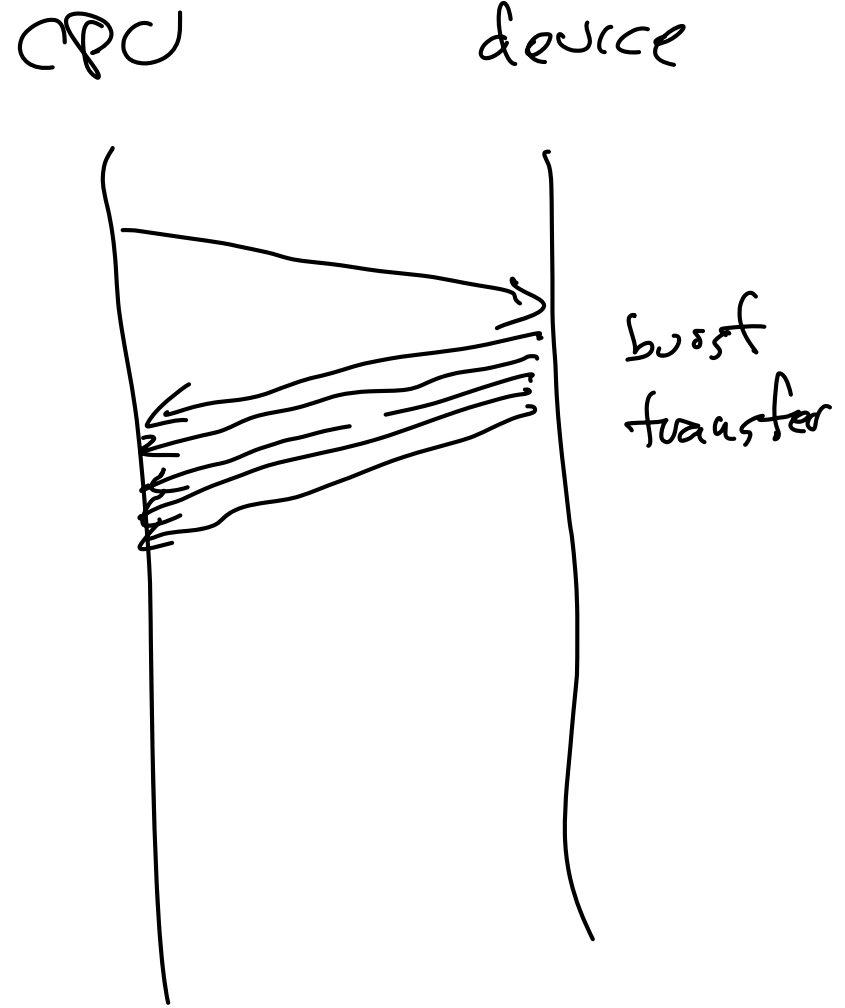
memory: cache

i/o: burst

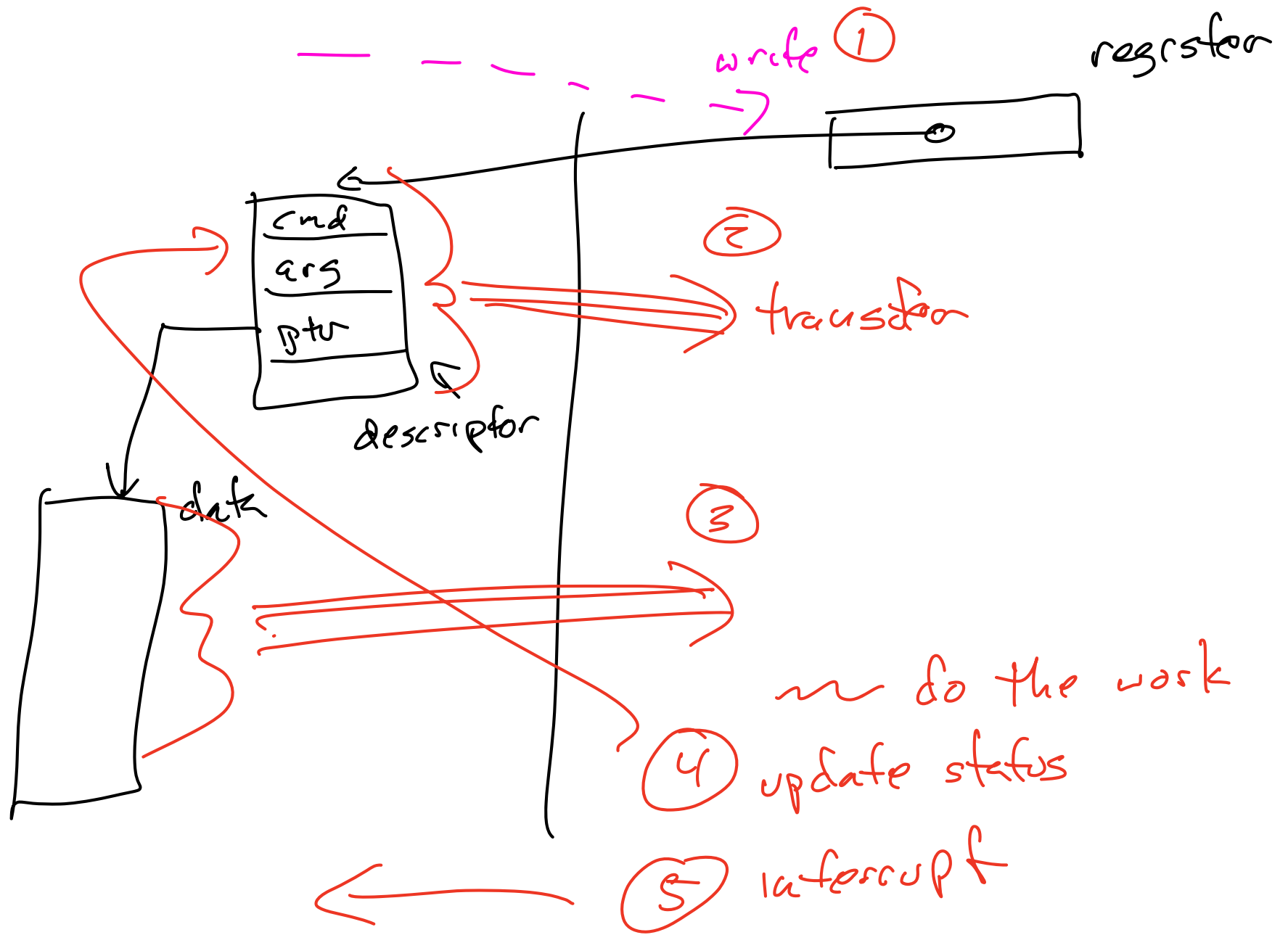
loop:
read 64kfs from PCIe

$$200\text{ns} = 5\text{M/sec}$$

$$\times 8\text{bytes} = 40\text{MB/s}$$



DMA (direct memory access)
on device



Unix-like device driver model

app
↓
read

kernel

prep
descriptor

write addr
→ register

sleep_on(x)
wait()

⋮

handle results

done

driver

wake x ← interrupt
signal →