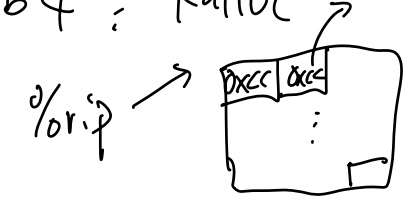
Week 11.b
CS 5600
03/22 2023

1. Last time
2. mmap
3. I/O architecture

- no OH
- Lab 4 : kalloc



INT *n*/INTO/INT 3—Call to Interrupt Procedure

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description
CC	INT 3	NP	Valid	Valid	Interrupt 3—trap to debugger.
CD *ib*	INT *imm8*	I	Valid	Valid	Interrupt vector specified by immediate byte.
CE	INTO	NP	Invalid	Valid	Interrupt 4—if overflow flag is 1.

1. Last time

- Page fault
- "Paging" (overcommitting Mem)

Cost:

memory access time ~ 100ns (// t_m)
SSD access time ~ 1 ms = 10^6 ns (// t_d)

0.001% — 0.1%, 0.01%

QUESTION: what does page fault probability (p) need to be to ensure that paging hurts performance by less than 10%?

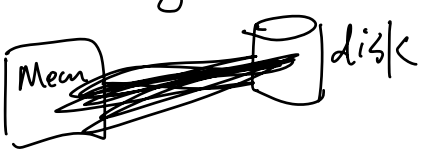
$$\Rightarrow P \cdot t_d + (1-P) \cdot t_m \leq 1.1 \cdot t_m$$

$$P \cdot t_d + t_m - P \cdot t_m \leq 0.1\, t_m$$

$$P(t_d - t_m) \leq 0.1\, t_m$$

$$P \leq \frac{0.1\, t_m}{t_d - t_m} \approx \frac{10^2 ns}{10^6 ns} = \frac{1}{10^5}$$

thrashing



VM: 50 page, PM: 40 Page

$20\% \cdot t_d + 80\% \cdot t_m = 0.2 \cdot 10^6 ns$ ≥ 0.001%

fm =100ns = 2000x SLOWER

- Mmap

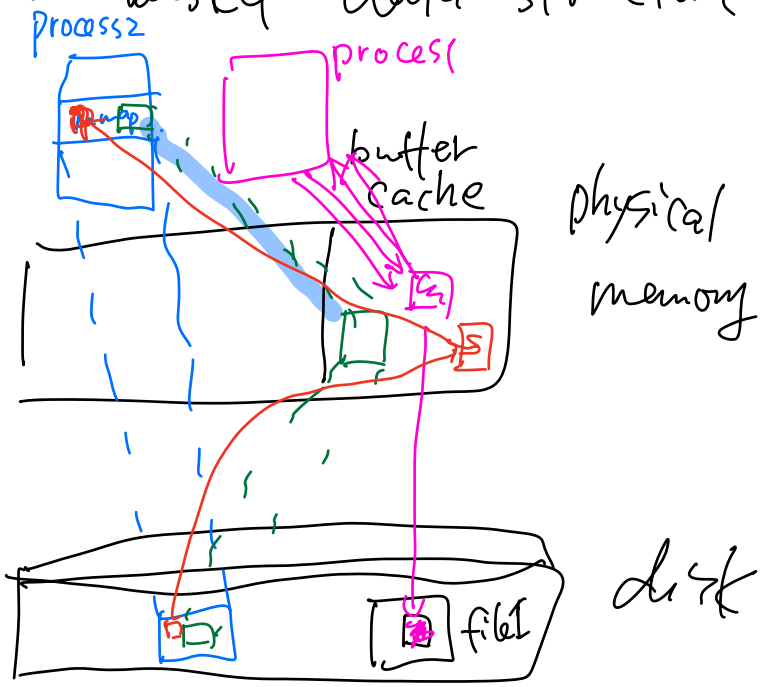
Some syscalls:

```
fd = open(pathname, mode)
write(fd, buf, sz)
read(fd, buf, sz)
```

A new syscall:

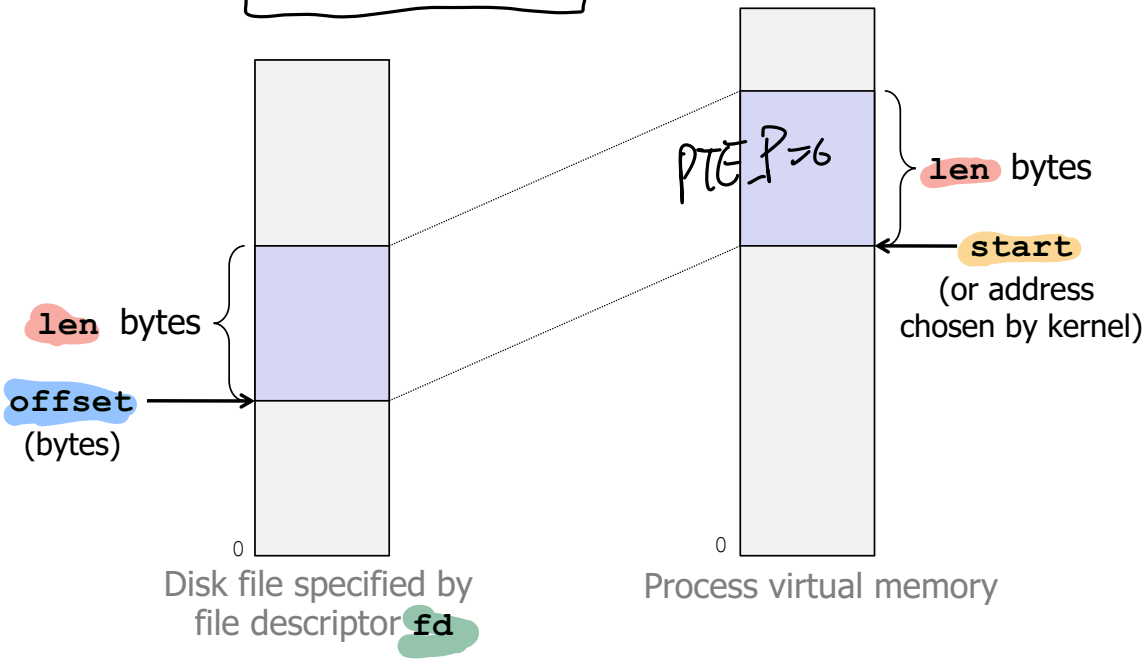
```
void* mmap(void* addr, size_t len, int prot, int flags,
           int fd, off_t offset);
```

- Why mmap is faster?
- Other use cases?
  - Large file
  - shared data structure (MAP_SHARED)
  - file-based data structure
- impl?



User-Level Memory Mapping

```
void *mmap(void *start, int len,
           int prot, int flags, int fd, int offset)
```



```
1   CS5600, Handout week11.a
2
3   /* file: mmap.c */
4
5   #include <fcntl.h>
6   #include <stdio.h>
7   #include <stdlib.h>
8   #include <sys/mman.h>
9   #include <sys/stat.h>
10  #include <sys/types.h>
11  #include <unistd.h>
12
13  void mmapwrite(int fd, int size);
14  void normalwrite(int fd, int size);
15
16  int main(int argc, char **argv) {
17      struct stat stat;
18      int fd;
19
20      if (argc != 2) {  // Check for required cmd line arg
21          printf("usage: %s <filename>\n", argv[0]);
22          exit(0);
23      }
24
25      /* Copy input file to stdout */
26      if ((fd = open(argv[1], O_RDONLY, 0)) < 0)
27          perror("open");
28
29      fstat(fd, &stat);
30
31      // option 1
32      mmapwrite(fd, stat.st_size);
33
34      /* // option 2
35       * normalwrite(fd, stat.st_size);
36       */
37
38      close(fd);
39
40      return 0;
41  }
42
43  void mmapwrite(int fd, int size) {
44
45      /* Ptr to memory mapped area */
46      char *bufp;
47
48      bufp = mmap(NULL, size, PROT_READ, MAP_PRIVATE, fd, 0);
49
50      write(STDOUT_FILENO, bufp, size);
51
52      return;
53  }
54
55
56  void normalwrite(int fd, int size) {
57
58      char *buf = malloc(size);
59
60      read(fd, buf, size);
61
62      write(STDOUT_FILENO, buf, size);
63
64      return;
65  }
```

entire file

offset

I don't care where it maps to"

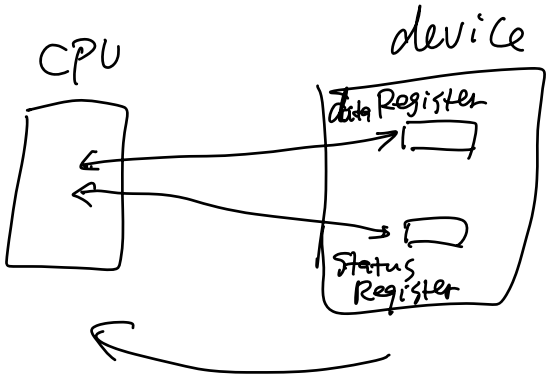
Question:
Which runs faster, option 1 or option 2? by how much?

Exercise:
Try to run both options by yourself:

```
$ cat /dev/urandom | head -c 1000000000 > 1G.file
$ make mmap
$ time ./mmap 1G.file > /dev/null

$ vim mmap.c
// switch to option 2
$ make mmap
$ time ./mmap 1G.file > /dev/null
```

I/O



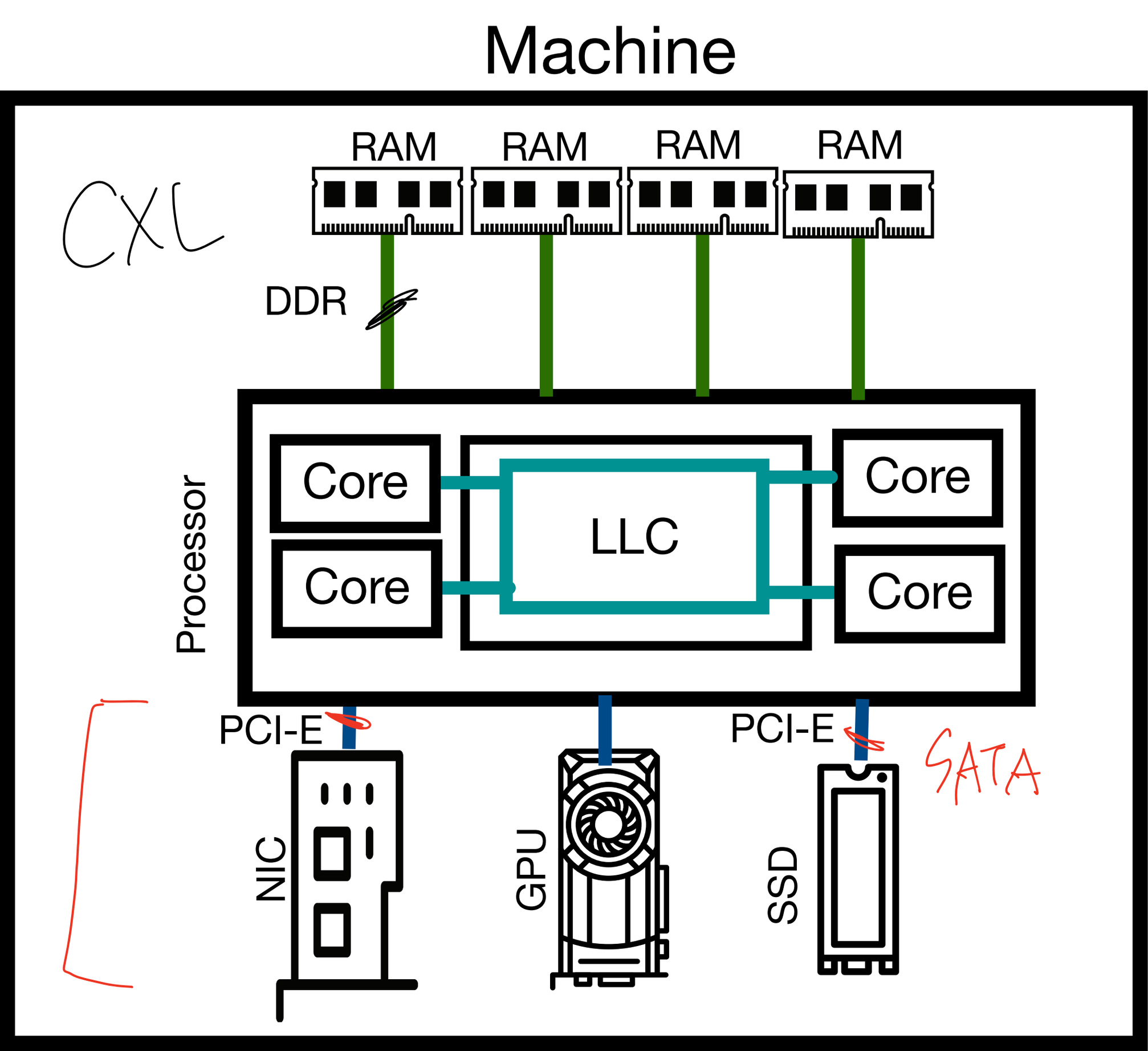
PS/2 (old keyboard/mouse)

IO Port	Access Type	Purpose
0x60	Read/Write	Data Port
0x64	Read	Status Register
0x64	Write	Command Register

inb / outb / inw / outw

↕

mdq


Machine
RAM
RAM
RAM
RAM
CXL
DDR
Processor
Core
Core
LLC
Core
Core
PCI-E
PCI-E
SATA
NIC
GPU
SSD

```
CS5600 Week11.b

1. Two examples of I/O instructions

  (a) Reading keyboard input

    The code below is an excerpt from WeensyOS.
    (details in PS/2 controller: https://wiki.osdev.org/%228042%22_PS/2_Controller)
    This reads a character typed at the keyboard (which shows up on the
    "keyboard data port" (kEYBOARD_DATAREG)).

    /* Excerpt from WeensyOS x86-64.h and k-hardware.cc */
    // Keyboard programmed I/O
    #define KEYBOARD_STATUSREG 0x64
    #define KEYBOARD_STATUS_READY 0x01
    #define KEYBOARD_DATAREG 0x60

    int keyboard_readc() {
      static uint8_t modifiers;
      static uint8_t last_escape;

      if ((inb(KEYBOARD_STATUSREG) & KEYBOARD_STATUS_READY) == 0) {
        return -1;
      }

      uint8_t data = inb(KEYBOARD_DATAREG);
      uint8_t escape = last_escape;
      last_escape = 0;

      if (data == 0xE0) { // mode shift
        last_escape = 0x80;
        return 0;
      } else if (data & 0x80) { // key release: matters only
                                //  for modifier keys
        int ch = keymap[(data & 0x7F) | escape];
        if (ch >= KEY_SHIFT && ch < KEY_CAPSLOCK) {
          modifiers &= ~(1 << (ch - KEY_SHIFT));
        }
        return 0;
      }

      int ch = (unsigned char) keymap[data | escape];

      if (ch >= 'a' && ch <= 'z') {
        if (modifiers & MOD_CONTROL) {
          ch -= 0x60;
        } else if (!(modifiers & MOD_SHIFT) != \
                   !(modifiers & MOD_CAPSLOCK)) {
          ch -= 0x20;
        }
      } else if (ch >= KEY_CAPSLOCK) {
        modifiers ^= 1 << (ch - KEY_SHIFT);
        ch = 0;
      } else if (ch >= KEY_SHIFT) {
        modifiers |= 1 << (ch - KEY_SHIFT);
        ch = 0;
      } else if (ch >= CKEY(0) && ch <= CKEY(21)) {
        ch = complex_keymap[ch - CKEY(0)].map[modifiers & 3];
      } else if (ch < 0x80 && (modifiers & MOD_CONTROL)) {
        ch = 0;
      }

      return ch;
    }

```



```

  (b) Setting the cursor position

    The code below is also excerpted from WeensyOS. It uses I/O
    instructions to set a blinking cursor. To set the cursor to
    the upper left of the screen, run:  console_show_cursor(0)

    // console_show_cursor(cpos)
    //   Move the console cursor to position 'cpos',
    //   which should be between 0 and 80 * 25.

    void console_show_cursor(int cpos) {
      if (cpos < 0 || cpos > CONSOLE_ROWS * CONSOLE_COLUMNS)
        cpos = 0;

      outb(0x3D4, 14); // Command 14 = upper byte of position
      outb(0x3D5, cpos / 256); // upper byte (256 = 2^8)
      outb(0x3D4, 15); // Command 15 = lower byte of position
      outb(0x3D5, cpos % 256); // lower byte

    }

    // if interested, see details: https://wiki.osdev.org/Text_Mode_Cursor
```