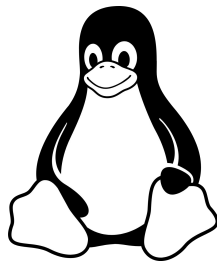
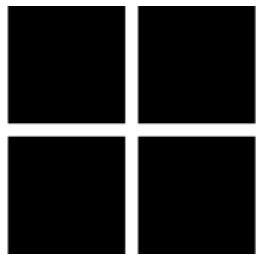
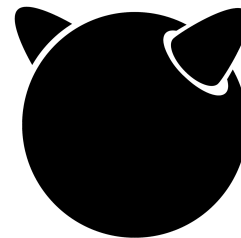


# Operating systems



macOS



and (немного про) computer  
architecture



## Давайте договоримся

- Мы уважаем друг друга и вместе создаем доброжелательную среду, чтобы нам было комфортно и безопасно взаимодействовать
- В сообщениях или устном обмене мыслями **не переходим на личности**, и не обсуждаем острые темы, мы – коллеги
- Все вопросы по курсу, которые возникли вне лекции, **пишите в общий чат**, а не в личку, чтобы все могли научиться и добавить что-то свое
- Я готов отвечать на вопросы каждый день с **20:00 до 23:00**
- **Не стесняйтесь делиться идеями** и обсуждать задания в чатике
- Глупых вопросов не бывает, понимание даст вам уверенность в знаниях
- Я готов к **обратной связи**, потому хочу сделать этот курс лучше



## Требования к студентам

- Базовое знание языка C
- *Использовать C++ можно, но с ограничениями, которые я буду оговаривать перед выдачей лаб*
- Уверенное пользование утилитами Unix в терминале (cd, ls, cat, chmod, chown, rm, ln, и тп.)
- Быть уверенным пользователем вашей любимой ОС, в которой вы будете делать лабы
- Знание курса Архитектуры ЭВМ на базовом уровне

# ? О чём будет курс?

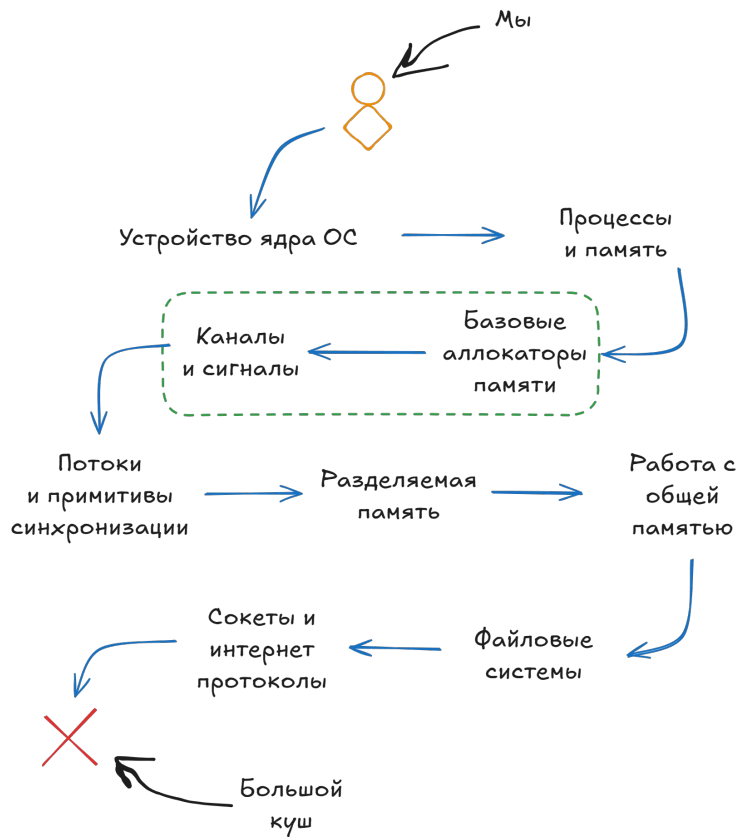
Вы умеете писать программы на C и Python. Скоро и на C++, и на других языках тоже! И знаете, что:

- Код программ можно компилировать или интерпретировать;
- Программы можно запустить в системе;
- На компьютере могут исполняться много программ;
- Можно каким-то образом задействовать больше одного ядра для ускорения вычислений;
- Программы как-то умеют обращаться к оборудованию и ресурсам операционной системы.

Благодаря этому курсу вы обретете понимание всех этих перечисленных аспектов и будете уметь разрабатывать эффективные и производительные программы

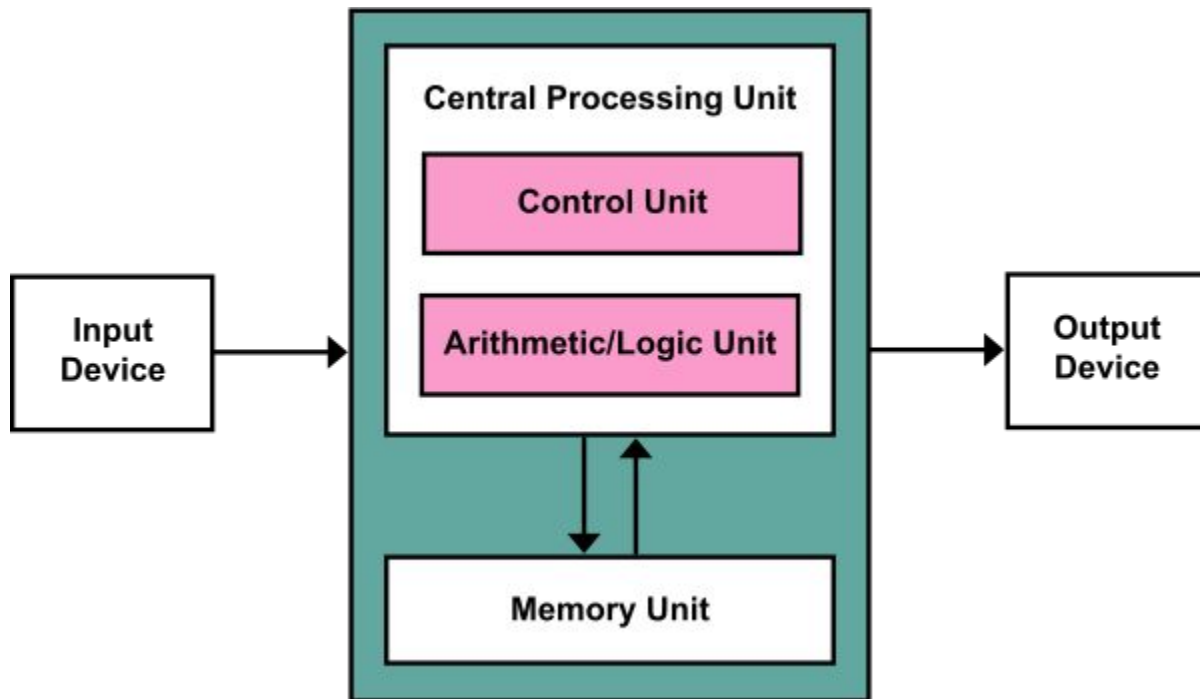


# Дорожная карта





# Машина фон Неймана





# Зачем нужна ОС?

**Операционная система** — это совокупность пользовательских и системных программ

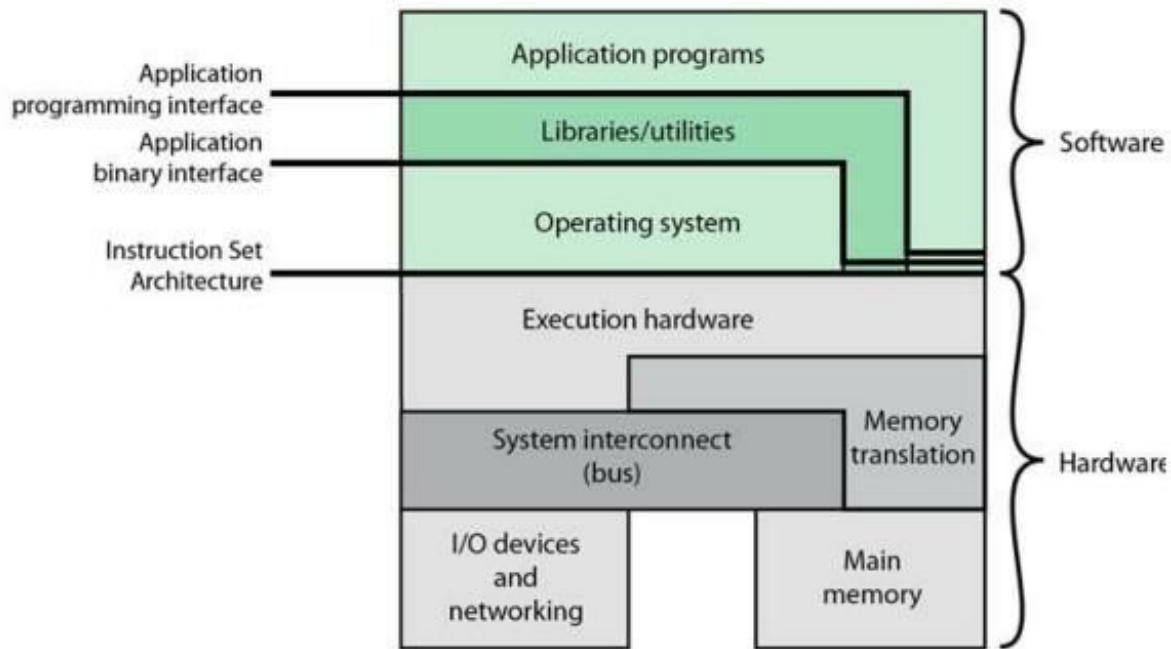
**Ядро** — её основная часть, отвечающая за управление ресурсами и взаимодействие между оборудованием и приложениями

По сути, ядро это тоже программа, которая параллельно исполняется вместе со всеми другими, но за кулисами, обеспечивает вам базовую работу пользовательских программ и драйверов и предоставляет абстрактный интерфейс к оборудованию



# Структура

Higher level of abstraction



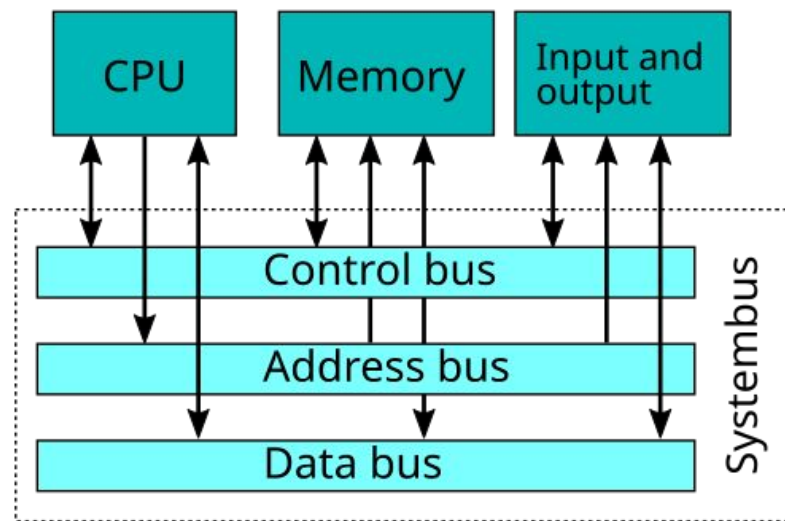
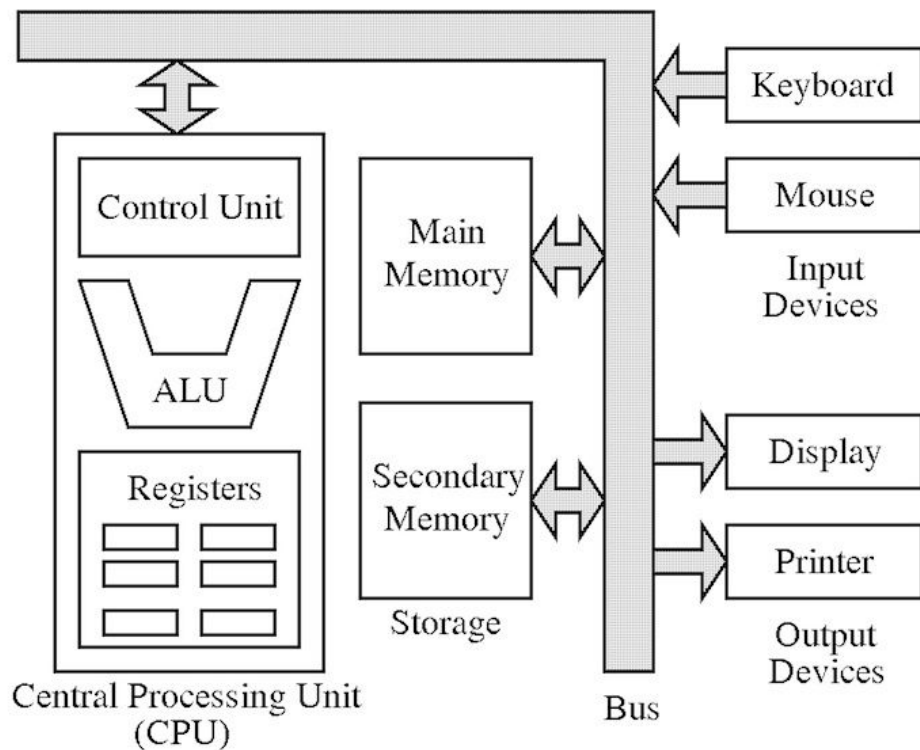
Lower level of abstraction



## А что делает ядро?

1. Распоряжается физическими и абстрактными ресурсами и занимается мультиплексированием во времени и пространстве
2. Сохраняет целостность данных: защищает пользователей и программы
3. Увеличивает уровень абстракции, предоставляя программный интерфейс

## Взаимодействие с устройствами





# Memory-mapped I/O

С точки зрения процессора, ввод-вывод выглядит как чтение и запись обычной памяти:

```
#define VGA_VIDEO_WIDTH 80

volatile uint16_t *const vga_memory = (uint16_t *)0xb8000;

static void put_char(uint8_t x, uint8_t y,
                    char c, uint8_t color) {
    vga_memory[y * VGA_VIDEO_WIDTH + x] = (uint16_t)((color << 8) | (c & 0x7F));
}
```



## Port-mapped I/O (x86)

Отдельный механизм для обмена данными с устройствами, иницируемого специальными инструкциями процессора: [in](#) и [out](#)

Каждое устройство подключено к процессору через уникальный "порт" — это как почтовый адрес, который процессор использует для общения с устройством

Например в порт 0x60 записываются сканкоды клавиатуры

```
loop:
    /* Store the scancode to al. */
    in $0x60, %al
    cmp %al, %cl
    jz loop
    mov %al, %cl
    PRINT_HEX <%al>
    PRINT_NEWLINE
    jmp loop
```

# Прерывания

Опрашивать устройства вручную неудобно, потому что мы должны сами задавать в коде в какое время это делать

Было бы здорово, если устройства сами могли сигнализировать о готовности подать нам данные. Для этого существуют **прерывания!**

Существуют три видов таких прерываний:

1. Exceptions (исключительные ситуации — например, деление на 0);
2. Hardware interrupts;
3. Software interrupts;



# Прерывания

```
dot_product(int x1, int y1, int x2, int y2): ; edi, esi, edx, ecx
    mov     eax, edi ; result = x1
    imul    eax, edx ; result *= x2
    mov     edx, eax ; lhs = result
    mov     eax, esi ; result = y1
    imul    eax, ecx ; result *= y2 ← rip (x86_64 instruction pointer)
    add     eax, edx ; result += lhs
    ret     ; result is in eax
```

; ...и где-то очень далеко, в ядре

```
handle_keyboard:
    in $0x60, %al
    ; что-то делаем с этим сканкодом
    iret ; возвращаемся обратно, откуда нас унесли
```



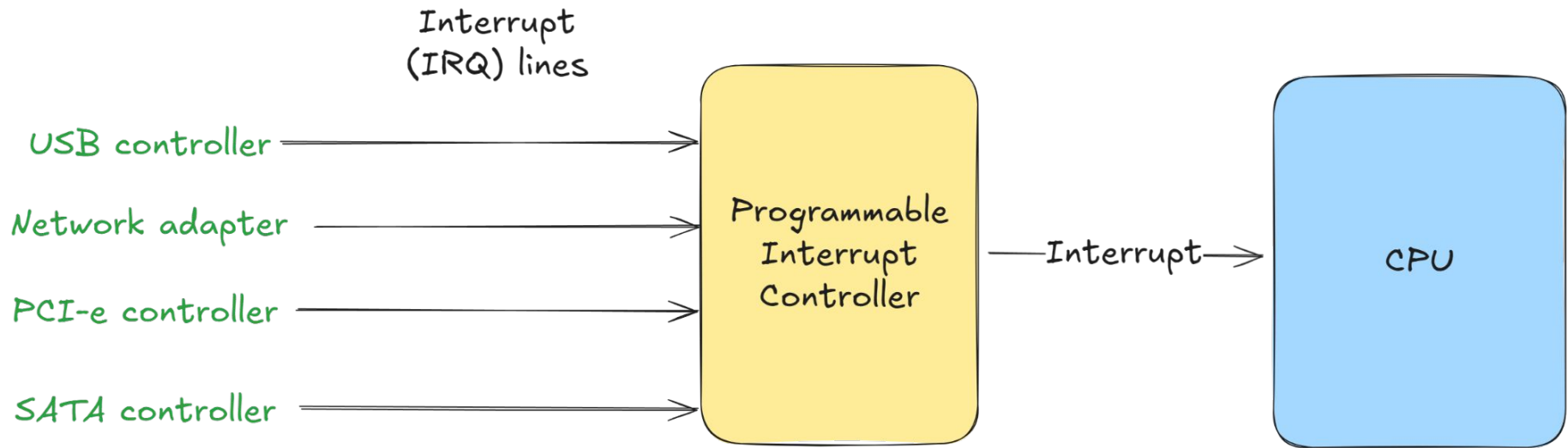
# А кто прерывание сигнализировал?

>buy материнскую плату  
>look inside  
>wires





# А кто прерывание сигнализировал?







# А как их обрабатывать и где?

Ядро заполняет таблицу (массив) из 256 указателей на функции (обработчики) прерываний и записывает в спец. регистр `idtr` (для x86) адрес этого массива

Для обработки прерывания процессор сохраняет в стеке регистры `cs`, `rip` и `eflags`, а затем загружает в `cs` и `rip` адрес обработчика, соответствующего номеру прерывания

После обработки прерывания мы возвращаемся в наш код с помощью `iret` инструкции, которая извлекает из стека три верхних значения и помещает их в регистры `cs`, `rip` и `eflags`



# А когда прерывания происходят?

После исполнения каждой инструкции, процессор проверяет в регистре eflags один битовый флаг IF (interrupt flag) и если он установлен и способность обрабатывать прерывание не маскируется, то процессор начинает обрабатывать прерывание

Процессору не хочется поддерживать приоритет и очередь прерываний, поэтому ответственность за доставку предоставляем отдельному чипу, программируемому контроллеру прерываний

Процессор и PIC коммуницируют через порты или адреса, как и любые другие устройства

# А можно ли самому вызвать прерывание?

Да, можно! И ваши программы делают это постоянно!

Пользовательские программы работают с оборудованием не напрямую, а через абстрактный интерфейс ядра, который реализуется через механизм **системных вызовов** (инструкции `syscall` или `int` на x86)

Системные вызовы нужны, чтобы:

- Управлять процессами
- Управлять файлами и каталогами
- Управлять устройствами
- и многим другим!



# Типы ядер

