

# Средства симуляции ЦП и ОС и изучение поведения программ

Лекция №2



Державин Андрей  
Шурыгин Антон

➤ **Квиз**

- Домашнее задание №1
- Введение в интерпретаторы
- Оптимизации
- Домашнее задание №2

- Квиз

➤ **Домашнее задание №1**

- Введение в интерпретаторы
- Оптимизации
- Домашнее задание №2

- Квиз
- Домашнее задание №1

➤ **Введение в интерпретаторы**

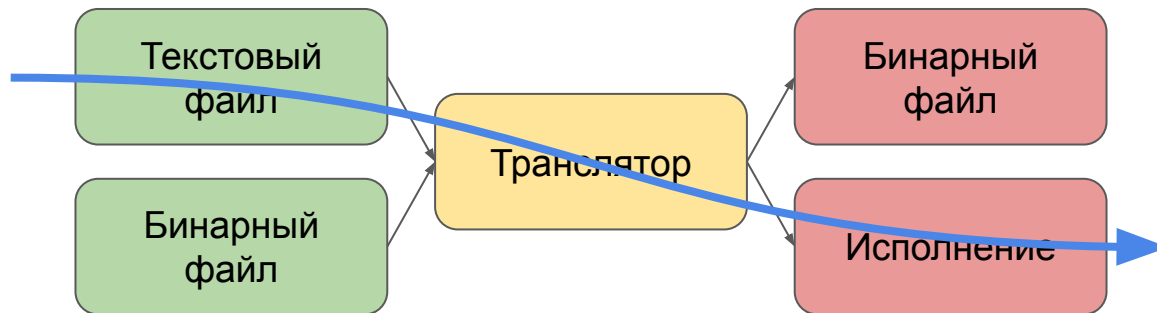
- Оптимизации
- Домашнее задание №2

# Определение

- Что такое интерпретатор?

# Определение

- Что такое интерпретатор?

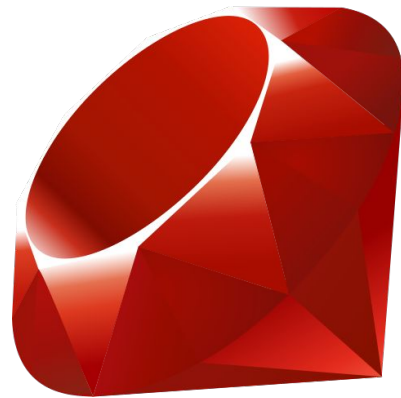
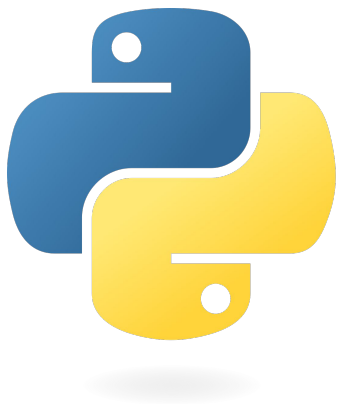


# Определение

- Интерпретатор - один из видов трансляторов
- Какие существуют интерпретаторы?

# Определение

- Интерпретатор - один из видов трансляторов
- Какие существуют интерпретаторы?





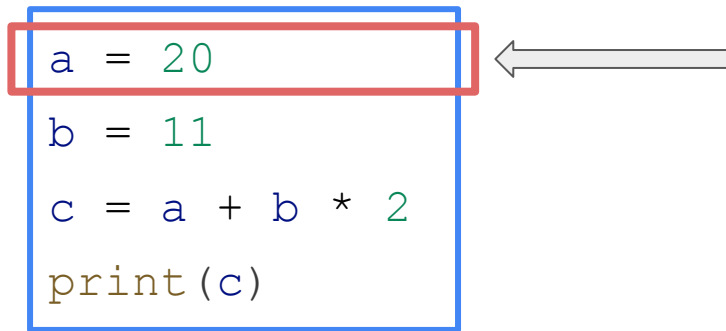
# Определение

- Интерпретатор - один из видов трансляторов

```
a = 20  
b = 11  
c = a + b * 2  
print(c)
```

# Определение

- Интерпретатор - один из видов трансляторов



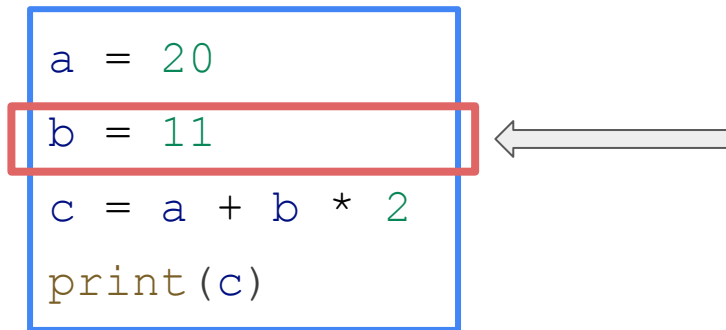
The diagram illustrates the execution of a code block. A blue rectangular box contains four lines of code: `a = 20`, `b = 11`, `c = a + b * 2`, and `print(c)`. The first line, `a = 20`, is highlighted with a red rectangular border. A grey arrow points from the right side of this red border towards the right edge of the slide, indicating the flow of execution or the current state of the interpreter.

```
a = 20  
b = 11  
c = a + b * 2  
print(c)
```

# Определение

- Интерпретатор - один из видов трансляторов

```
a = 20
b = 11
c = a + b * 2
print(c)
```



# Определение

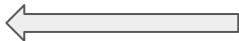
- Интерпретатор - один из видов трансляторов

```
a = 20
```

```
b = 11
```

```
c = a + b * 2
```

```
print(c)
```



# Определение

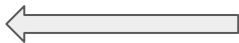
- Интерпретатор - один из видов трансляторов

```
a = 20
```

```
b = 11
```

```
c = a + b * 2
```

```
print(c)
```



# Определение

- Интерпретатор - один из видов трансляторов

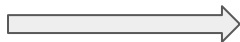
```
a = 20
b = 11
c = a + b * 2
print(c)
```

```
addi x1, x0, 20
addi x2, x0, 11
add x2, x2, x2
add x3, x2, x1
```

# Определение

- Интерпретатор - один из видов трансляторов

```
a = 20  
b = 11  
c = a + b * 2  
print(c)
```



```
addi x1, x0, 20  
addi x2, x0, 11  
add x2, x2, x2  
add x3, x2, x1
```

# Определение

- Интерпретатор - один из видов трансляторов

```
a = 20  
b = 11  
c = a + b * 2  
print(c)
```



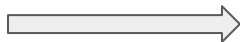
```
addi x1, x0, 20  
addi x2, x0, 11  
add x2, x2, x2  
add x3, x2, x1
```



# Определение

- Интерпретатор - один из видов трансляторов

```
a = 20  
b = 11  
c = a + b * 2  
print(c)
```

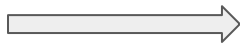


```
addi x1, x0, 20  
addi x2, x0, 11  
add x2, x2, x2  
add x3, x2, x1
```

# Определение

- Интерпретатор - один из видов трансляторов

```
a = 20  
b = 11  
c = a + b * 2  
print(c)
```



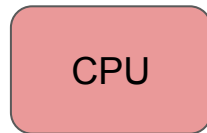
```
addi x1, x0, 20  
addi x2, x0, 11  
add x2, x2, x2  
add x3, x2, x1
```

# Программная модель

- Какой главный модуль в нашей модели?

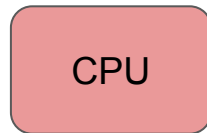
# Программная модель

- Ядро исполняет инструкции



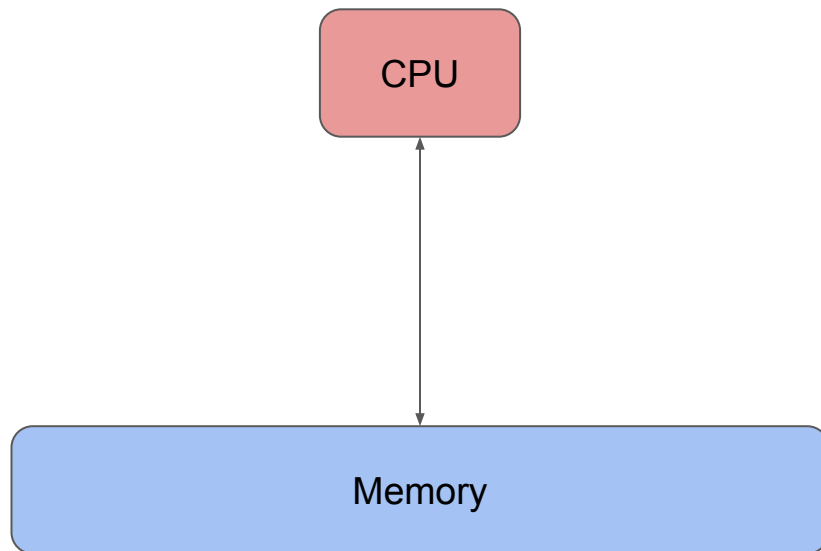
# Программная модель

- Ядро исполняет инструкции
- Что ещё необходимо?



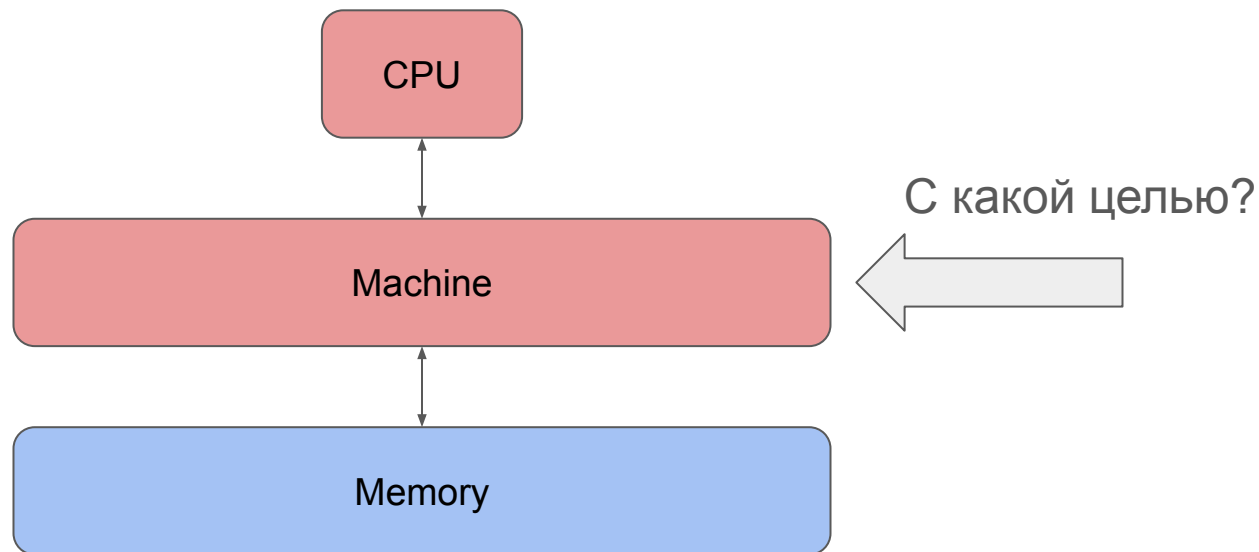
# Программная модель

- Ядро исполняет инструкции
- Память - хранит код и данные



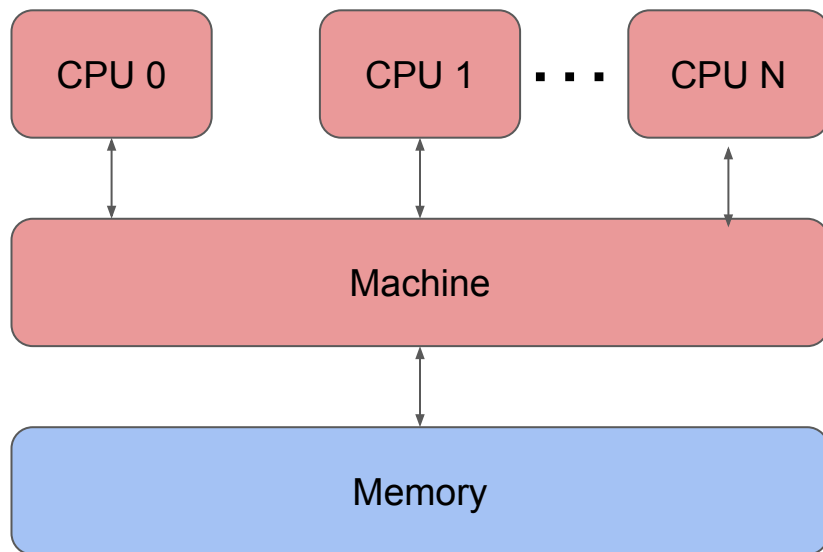
# Программная модель

- Ядро исполняет инструкции
- Память - хранит код и данные



# Программная модель

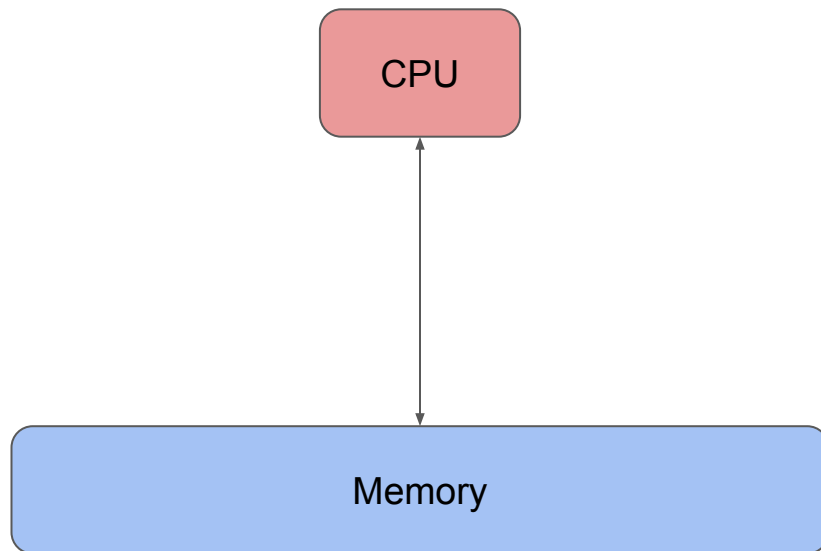
- Ядро исполняет инструкции
- Память - хранит код и данные





# Программная модель

- Как бы вы реализовали простую модель?



# Программная модель

- Простейшая модель на C

```
typedef uint32_t Register;
const size_t kNumRegs = 32;

struct CpuState {
    Register gpr_regs[kNumRegs];
    Memory *memory;
};

struct Memory {
    uint8_t *data;
};
```

# Стадии интерпретатора

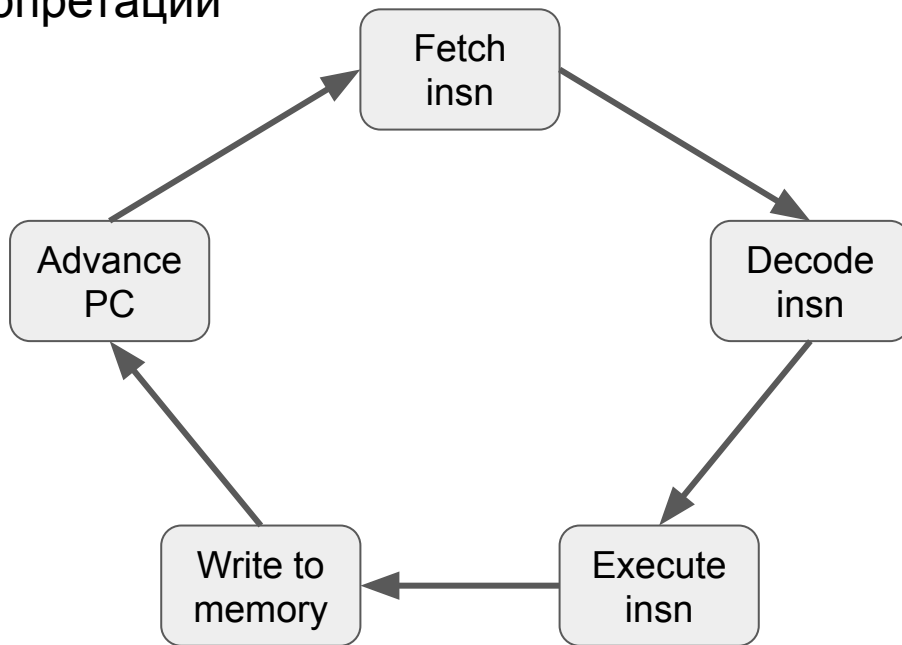
- Откуда процессор берет следующую инструкцию?

# Стадии интерпретатора

- PC хранит адрес текущей команды
- Какие есть стадии работы процессора?

# Стадии интерпретатора

- PC хранит адрес текущей команды
- 5 стадий интерпретации



# Можификация модели

```
typedef uint32_t Register;
const size_t kNumRegs = 32;
struct CpuState {
    Register pc;
    Register gpr_regs[kNumRegs];
    Memory *memory;
};
struct Memory {
    uint8_t *data;
};
```

# Стадии интерпретатора. Fetch

# Стадии интерпретатора. Fetch

```
Register fetch(CpuState *cpu) {  
    Register bytes = cpu->memory->load(cpu->pc);  
    return bytes;  
}
```



Стадии интерпретатора. Decode

# Стадии интерпретатора. Decode

```
Instruction decode(Register bytes) {  
    Instruction insn{.opcode=get_opcode(bytes)};  
    switch (insn.opc) {  
        case Opcode::kAdd:  
            insn.src1 = get_src1(bytes);  
            insn.src2 = get_src2(bytes);  
            insn.dst = get_dst(bytes);  
            break;  
    }  
    return insn;  
}
```

Стадии интерпретатора. Execute

# Стадии интерпретатора. Execute

```
void execute(CpuState *cpu, Instruction insn) {  
    switch (insn.opc) {  
        case Opcode::kAdd: {  
            auto res=cpu->getReg(insn.src1) + cpu->getReg(insn.src2);  
            cpu->setReg(insn.dst, res);  
            break;  
        }  
        /// ...  
    }  
}
```

Стадии интерпретатора. Write back & PC advance

## Стадии интерпретатора. Write back & PC advance

```
// ..  
case Opcode::kLoad: {  
    auto data = cpu->memory->load(cpu->getReg(insn.src1));  
    cpu->setReg(insn.dst, data);  
    break;  
}  
case Opcode::kJump: {  
    cpu->pc = cpu->getReg(insn.src1);  
    break;  
}  
// ..
```

# Стадии интерпретатора. Write back

- Какие особенности может иметь доступ в память?

# Стадии интерпретатора. Write back

- Режимы адресации ( $\text{addr} = \text{base} + \text{offset} * \text{scale}$  )
- Неявные операнды (push в x86)
- Различные требования по выравниванию (alignment)



- Квиз
- Домашнее задание №1
- Введение в интерпретаторы

➤ **Оптимизации**

- Домашнее задание №2

# Ускорение интерпретатора

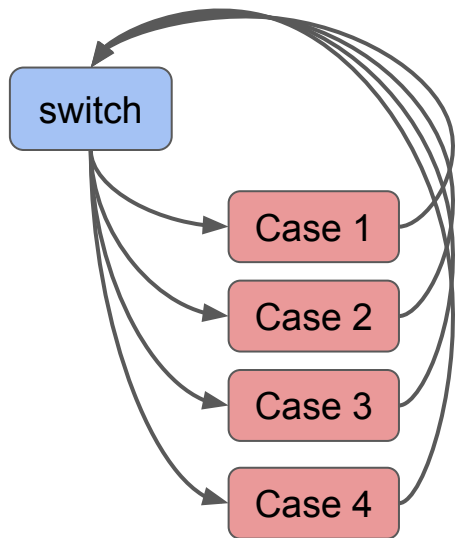
- Какие узкие места можно выделить в представленной модели?

# Ускорение интерпретатора

- Шитый код (threaded code)
- Чем плох один большой switch?

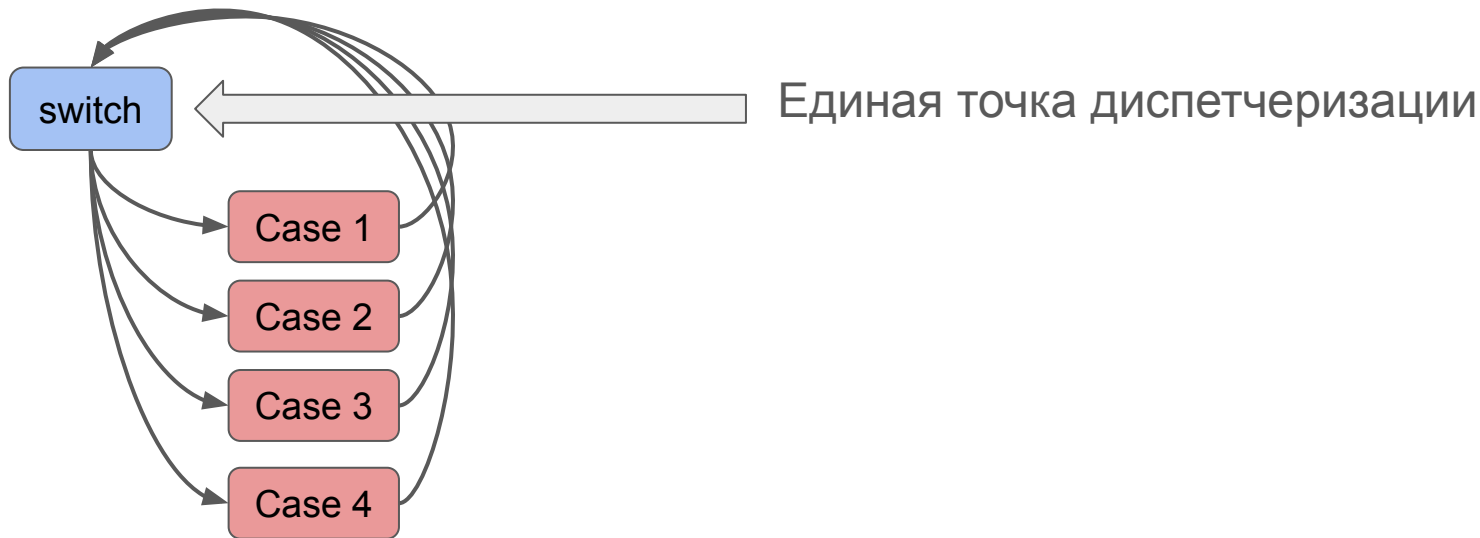
# Ускорение интерпретатора

- Шитый код (threaded code)
- Чем плох один большой switch?



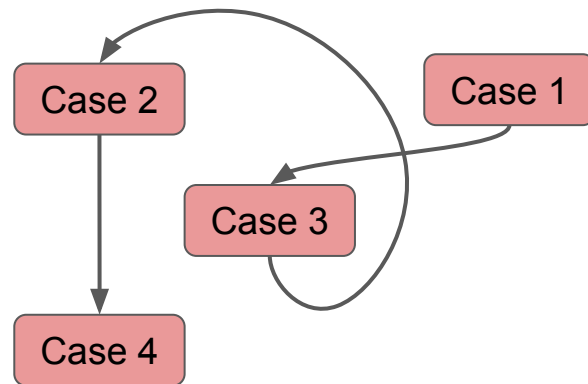
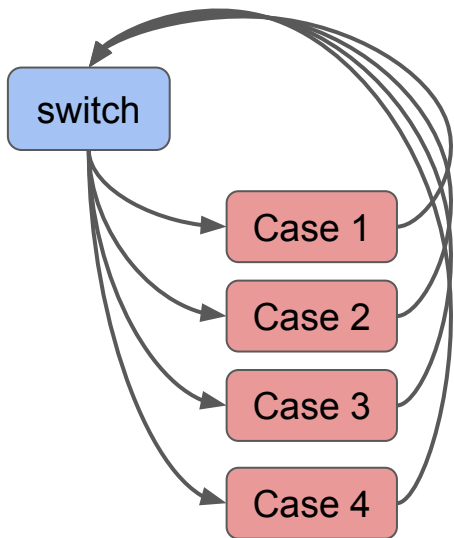
# Ускорение интерпретатора

- Шитый код (threaded code)
- Чем плох один большой switch?



# Ускорение интерпретатора

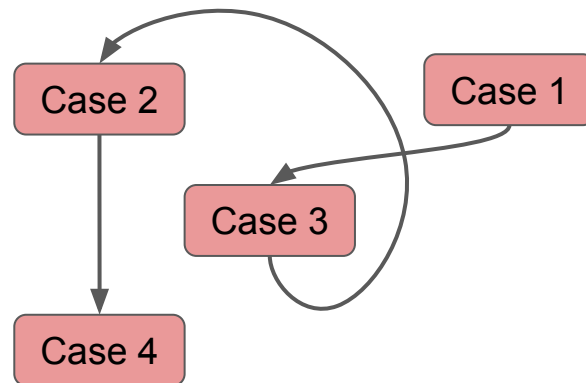
- Шитый код (threaded code)
- Введение нескольких точек диспетчеризации



# Ускорение интерпретатора

- Шитый код (threaded code)
- Введение нескольких точек диспетчеризации
- Пример реализации через GNU extension (лучше не использовать в таком виде)

```
DO_ADD:  
    ADD_handler(cpu, insn);  
    cpu->pc += insn.size;  
    goto label[pc];
```



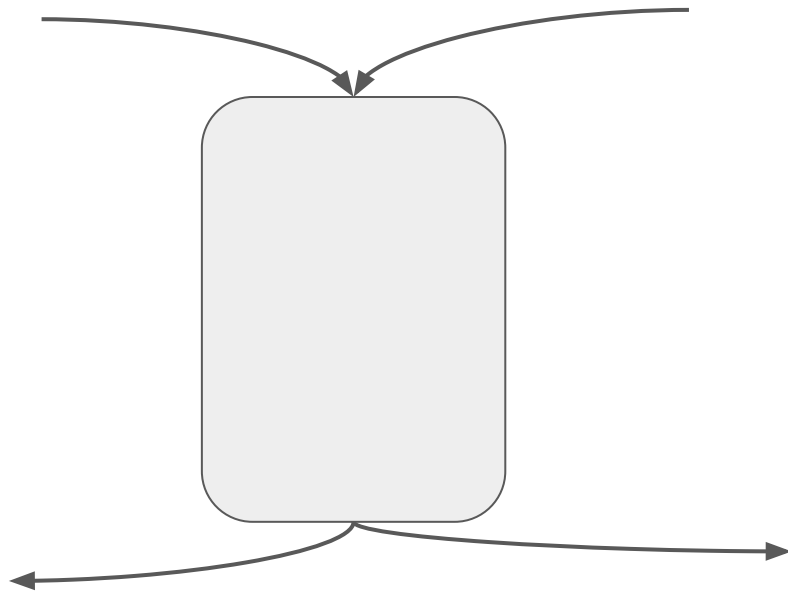
# Ускорение интерпретатора

- Что такое линейный участок кода?



# Ускорение интерпретатора

- Линейный участок кода (Basic Block) - последовательность инструкций, имеющая **одну** точку входа и **одну** точку выхода



# Ускорение интерпретатора

- *Линейный участок кода* (Basic Block) - последовательность инструкций, имеющая **одну** точку входа и **одну** точку выхода
- Какое свойство следуют из определения?

# Ускорение интерпретатора

- *Линейный участок кода (Basic Block)* - последовательность инструкций, имеющая **одну** точку входа и **одну** точку выхода
- Инструкция из Базового блока выполняется  $\Leftrightarrow$  выполняется первая инструкция Базового блока

# Ускорение интерпретатора

- *Линейный участок кода (Basic Block)* - последовательность инструкций, имеющая **одну** точку входа и **одну** точку выхода
- Инструкция из Базового блока выполняется  $\Leftrightarrow$  выполняется первая инструкция Базового блока
- Идея: сохранять декодированные базовые блоки для повторного исполнения