

ЦЕНТР ЦИФРОВОГО РАЗВИТИЯ

STUD iT

ЛЕКЦИЯ 1

ЗНАКОМСТВО
ТЕОРИТИЧЕСКИЙ МИНИМУМ ПО CS



КТО МЫ?

МЫ – ЦЕНТР ЦИФРОВОГО РАЗВИТИЯ СТУДЕНЧЕСКОГО СОВЕТА, КОМАНДА РАЗРАБОТЧИКОВ И IT-ЭНТУЗИАСТОВ, КОТОРЫЕ СОЗДАЮТ ТЕХНОЛОГИИ ДЛЯ СТУДЕНТОВ.



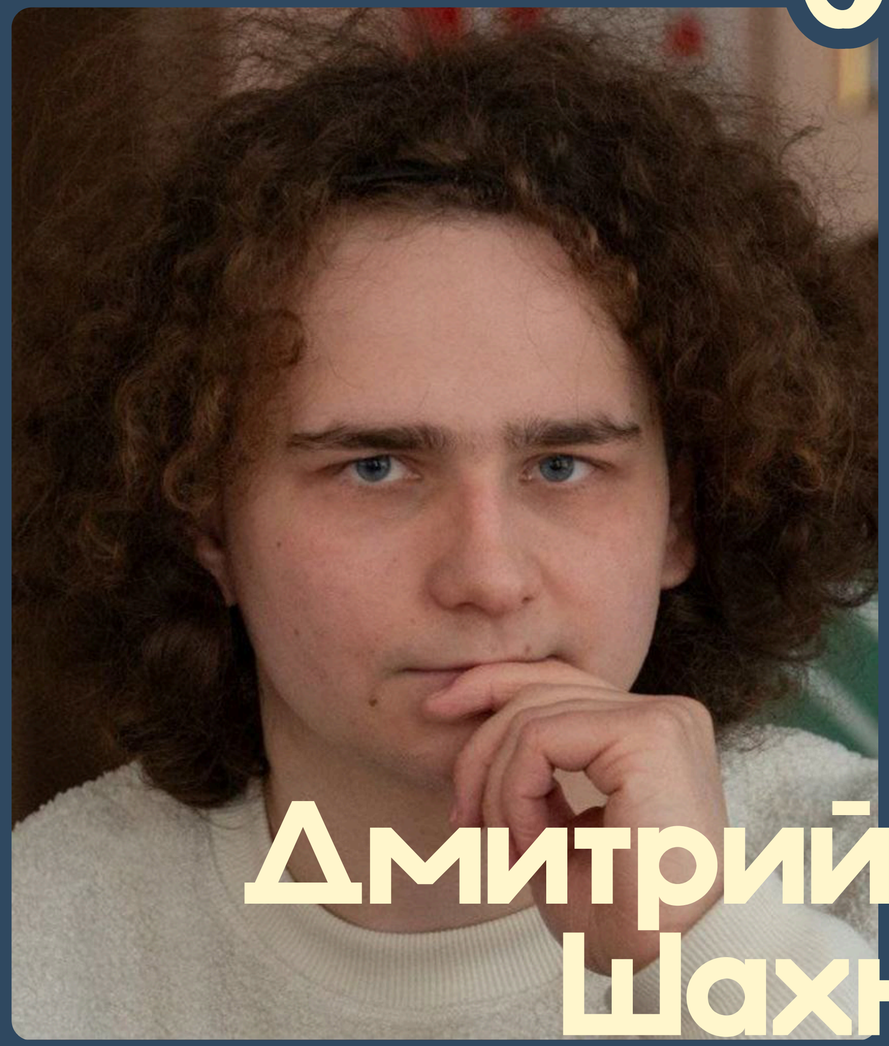
ШМБ 2025

ЧЕМ ЗАНИМАЕМСЯ?

- боты для мероприятий (регистрация, голосования, напоминания)
- внутренние сервисы (боты для организации отчётности активистов)
- сайты и порталы (сайт студенческого совета, мисс очарование, легенды бауманки, студент года)
- другие проектами (поддержка, менторинг, написание кода, etc)
- Администрируем все это добро на наших серверах и виртуальных машинах.
- Каждый год образовательная программа для студентов (все для вас, наших солнышек)

НАША МИССИЯ: УПРОЩАТЬ И АВТОМАТИЗИРОВАТЬ СТУДЕНЧЕСКУЮ ЖИЗНЬ ЧЕРЕЗ ЦИФРОВЫЕ РЕШЕНИЯ.

ЗНАКОМИМСЯ OUR TEAM



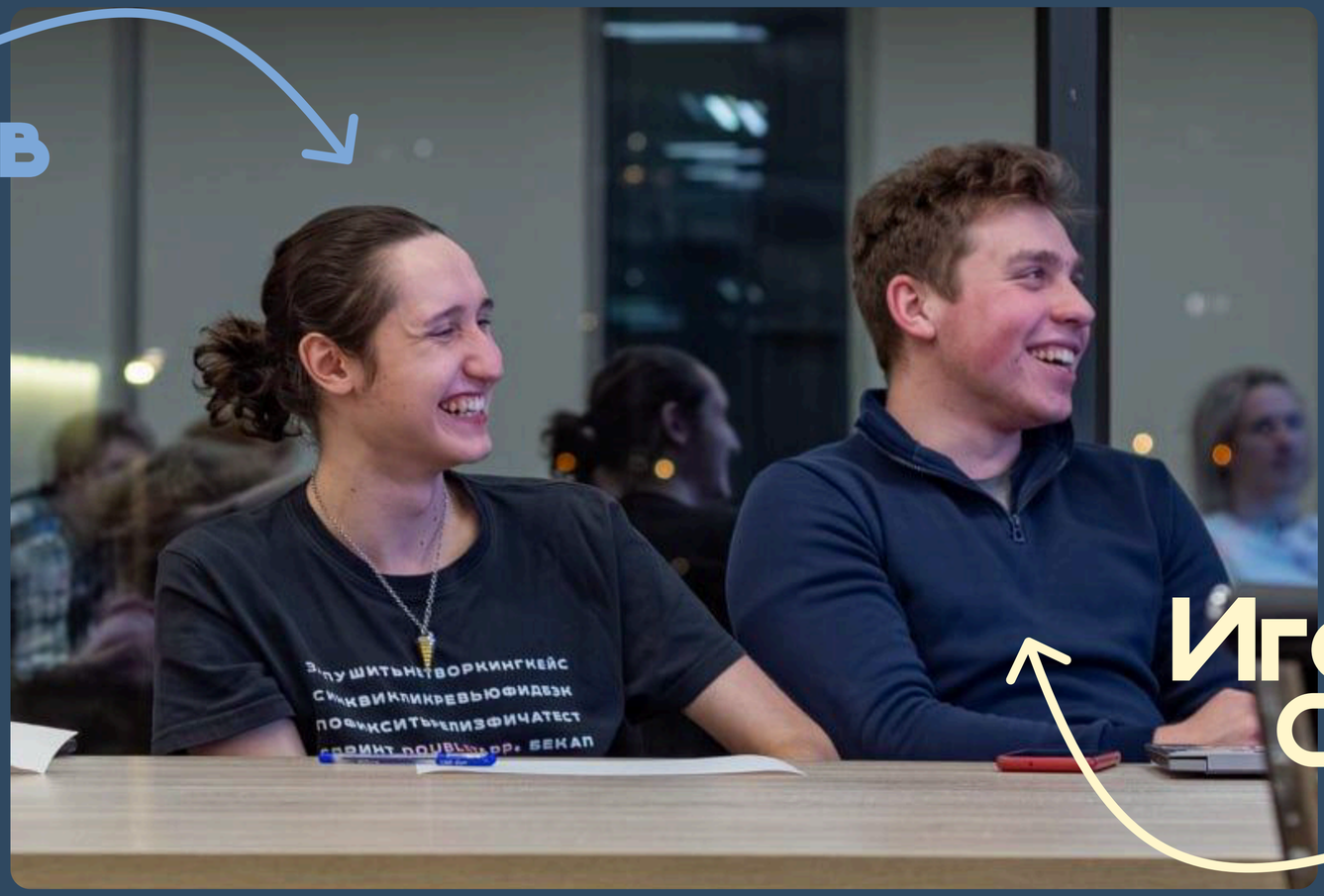
Дмитрий Шахнович

- Лектор и ex-организатор курса ЩЦР
- Окончил ИУ7
- Участник разработки и поддержки проектов ЩЦР
- Работал над проектом для электронного университета МГТУ
- Пишу на чём скажут (Go, python, C, C++, ts, bash, lisp, prolog...)

ЗНАКОМИМСЯ OUR TEAM

Игорь Кабанов

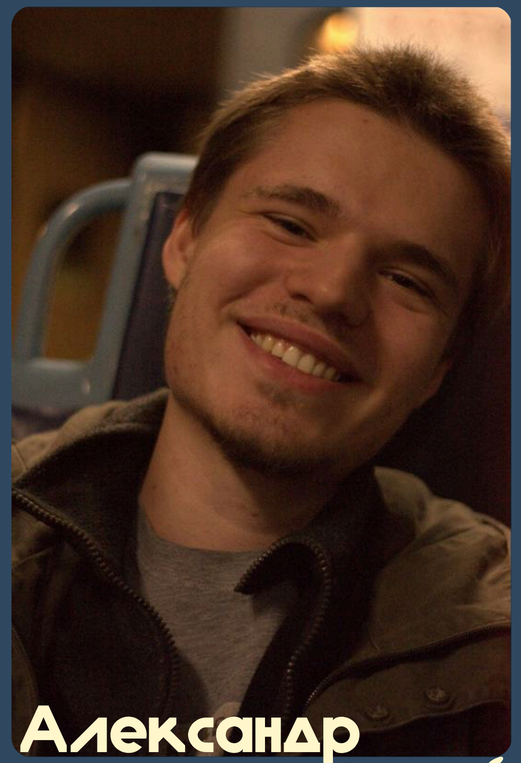
ОТВЕТСВЕННЫЙ ЗА ПРОЕКТЫ И ЖЕЛЕЗО ОТДЕЛА



ГЛАВА ЦЕНТРА ЦИФРОВОГО РАЗВИТИЯ

Игорь Слободянюк

ЗНАКОМИМСЯ OUR TEAM



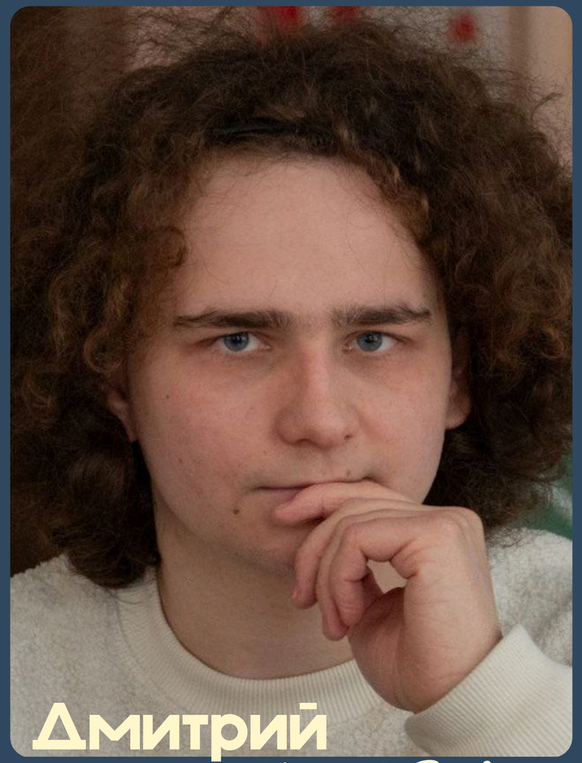
Александр Формаковский

ЕХ-ЖЕЛЕЗНИК ЦЦР



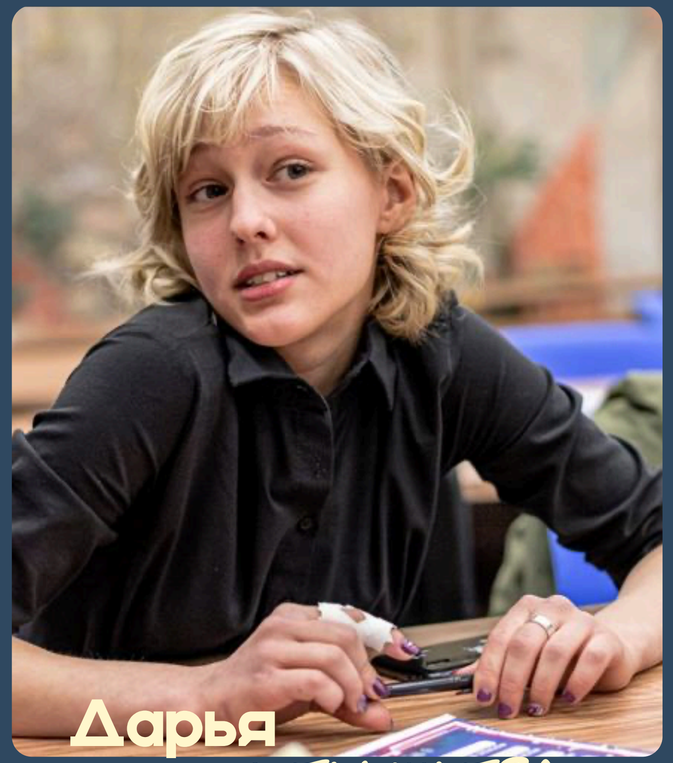
Анндрей Поляков

ЕХ-ГЛАВА ЦЦР



Дмитрий Шахнович

ЕХ-ОБРАЗОВЫВАТЕЛЬ



Дарья Серышева

ОТВЕТСТВЕННАЯ ЗА КОММУНИКАЦИИ

ПЛАН ЛЕКЦИИ

Информатика - это наука
о построении и понимании
абстракций



План лекции

- Зачем всё это - почему программисту полезно знать, что под капотом
- Алгоритмы - что такое решение задачи и есть ли у него пределы
- Архитектура ЭВМ - как железо на самом деле исполняет программу
- Операционные системы - кто стоит между программой и железом
- Языки программирования - на чём мы пишем и как это превращается в команды
- Парадигмы - как мы думаем о коде



ЗАЧЕМ ЭТО ПРОГРАММИСТУ?

ВСЕ НЕТРИВИАЛЬНЫЕ АБСТРАКЦИИ В КАКОЙ-ТО СТЕПЕНИ ЯВЛЯЮТСЯ НЕГЕРМЕТИЧНЫМИ

ЗАКОН ПРОТЕКАЮЩИХ АБСТРАКЦИЙ, ДЖОЭЛЬ СПОЛЬСКИ

Чрезмерная зависимость от абстракций без понимания лежащих в их основе технологий может усложнить решение проблем.

Чтобы эффективно работать со сложными системами, инженеру необходимо изучать не только саму абстракцию, но итехнологии, на которых она базируется (А еще это все очень интересно!)



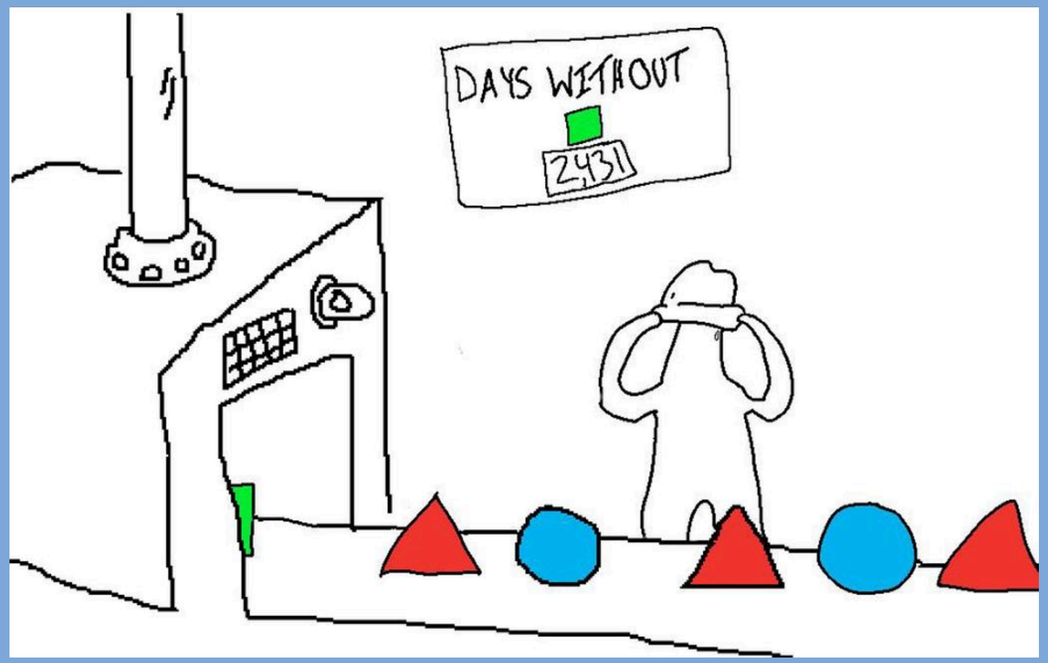
Алгоритмизация

Два вида деятельности



игровая деятельность

- непредсказуема
- тяжело повторить и автоматизировать
- творческий процесс, создающий новое
- примеры: написать песню, придумать какой бот нужен студсовету



механическая деятельность

- предсказуема и повторяема
- создаёт конечный продукт, но не инновации
- примеры: сборка шкафа по инструкции, пересчёт денег, сортировка писем по индексу

Алгоритм

Алгоритм это конечная, точная, детерминированная, упорядоченная последовательность действий, направленная на решение конкретной задачи или класса задач, достижения цели.

Из определения, ключевые свойства:

- Конечность
- Точность
- Детерминированность
- Упорядоченность

Помимо этого, важно что алгоритм описывается для некоторого исполнителя, что определяет набор команд и позволяет реализовать свойство точности.

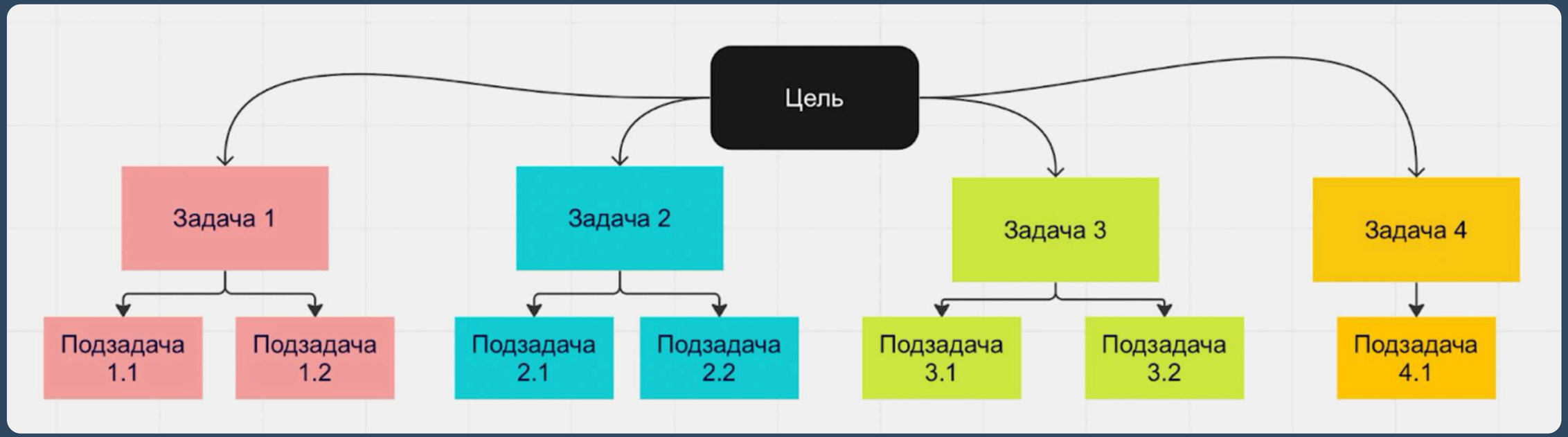


Пример алгоритма



Подумаем, а является ли это алгоритмом?

Декомпозиция

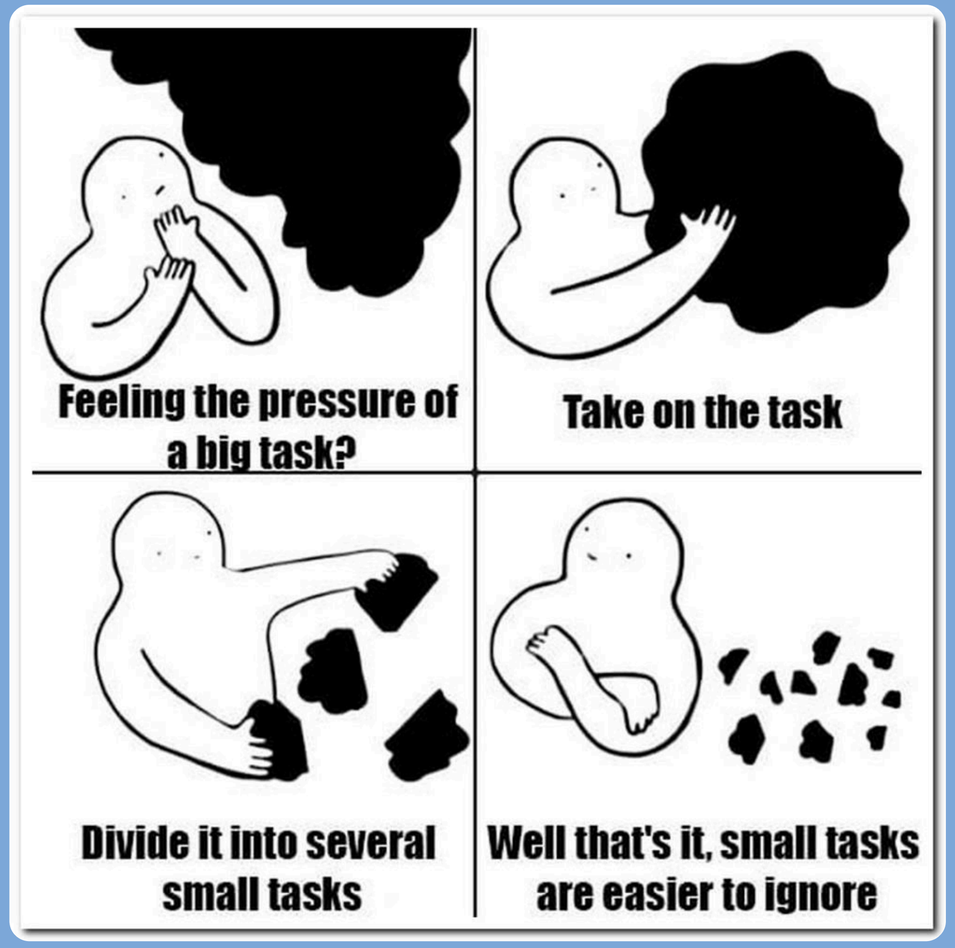


Декомпозиция - это мысленное разбиение большой, сложной проблемы на более мелкие, управляемые подзадачи.

А КОГДА ОСТАНОВИТЬСЯ?

- Дробим, пока подзадача не станет элементарной для исполнителя или не будет решаться «очевидно»

- Хорошая подзадача называется одним глаголом: «проверить формат группы»



Формализация алгоритмов

Словесное описание не всегда однозначно описывает алгоритмы

Хочется, чтобы помимо словесной информации, имелись конструкции, которые дают точки однозначности

Жена говорит программисту:
— Дорогой, сходи, купи булки. И, если есть яйца, возьми десяток.
Программист приходит в магазин:
— Яйца есть?
— Есть.
— Дайте десять булок.



Теорема Бёма-Якопини

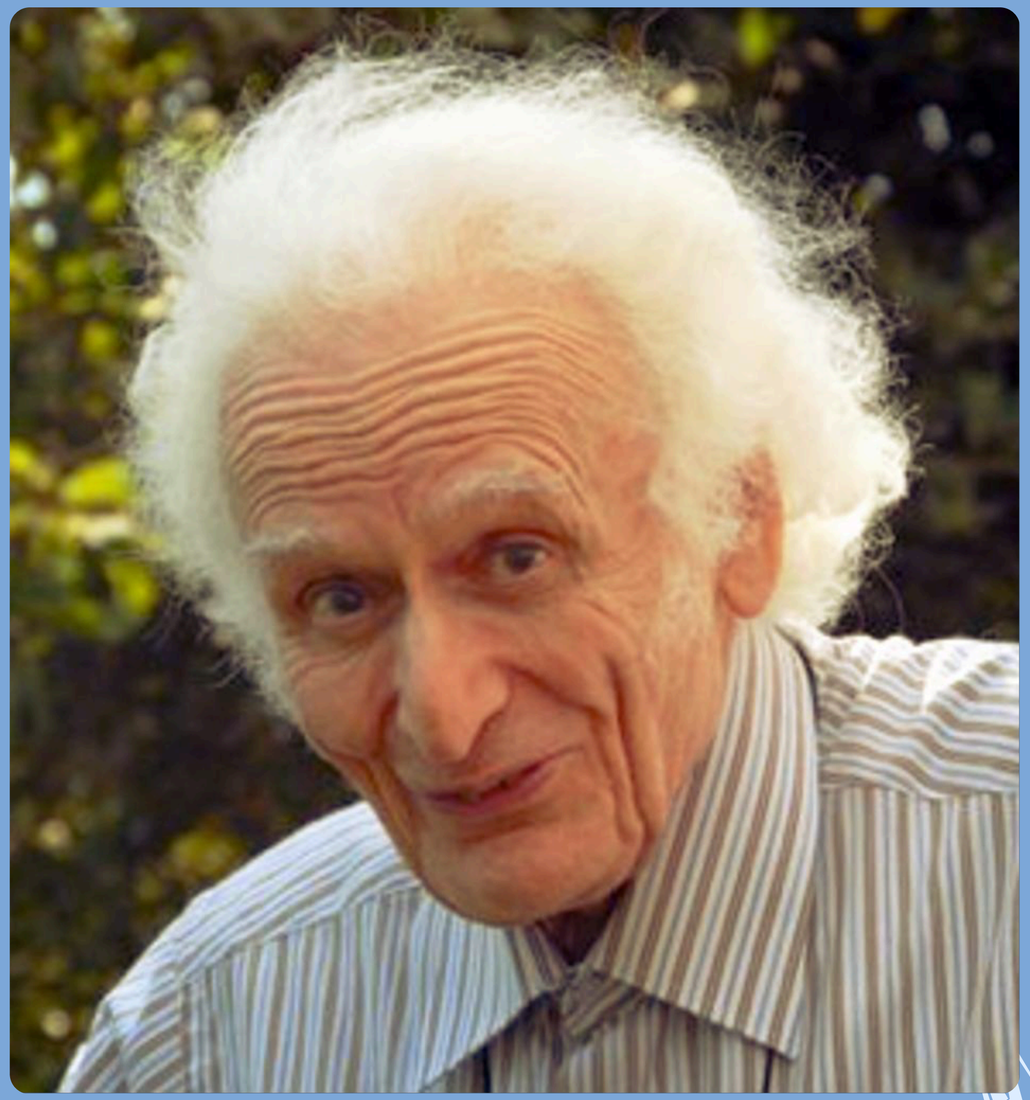
БЁМ!!!

Любой алгоритм может быть записан с использованием всего трех базовых управляющих конструкций:

- Следование (Sequence):
Выполнение действий одно за другим

- Ветвление (Selection, условный оператор):
Выполнение разных действий в зависимости от условий

- Повторение (Iteration, цикл):
Многократное выполнение действия



Способы формализации



Блок схема

Способы формализации

Следование	<действие1> <действие2> ...
Ветвление	Если <условие> то <действие1> [иначе <действие2>] Конец-если
Цикл while	Пока <условие> <действие> Конец-цикл
Выбор	Выбор <переменная> <значение1>: <действие1> <значение2>: <действие2> [иначе: <действие3>]
Цикл do-while	Выполнять <действие> пока <условие>
Счетный цикл	Для <переменная> от <значение> до <значение> с шагом <значение> <действие> Конец-цикл

Псевдокод



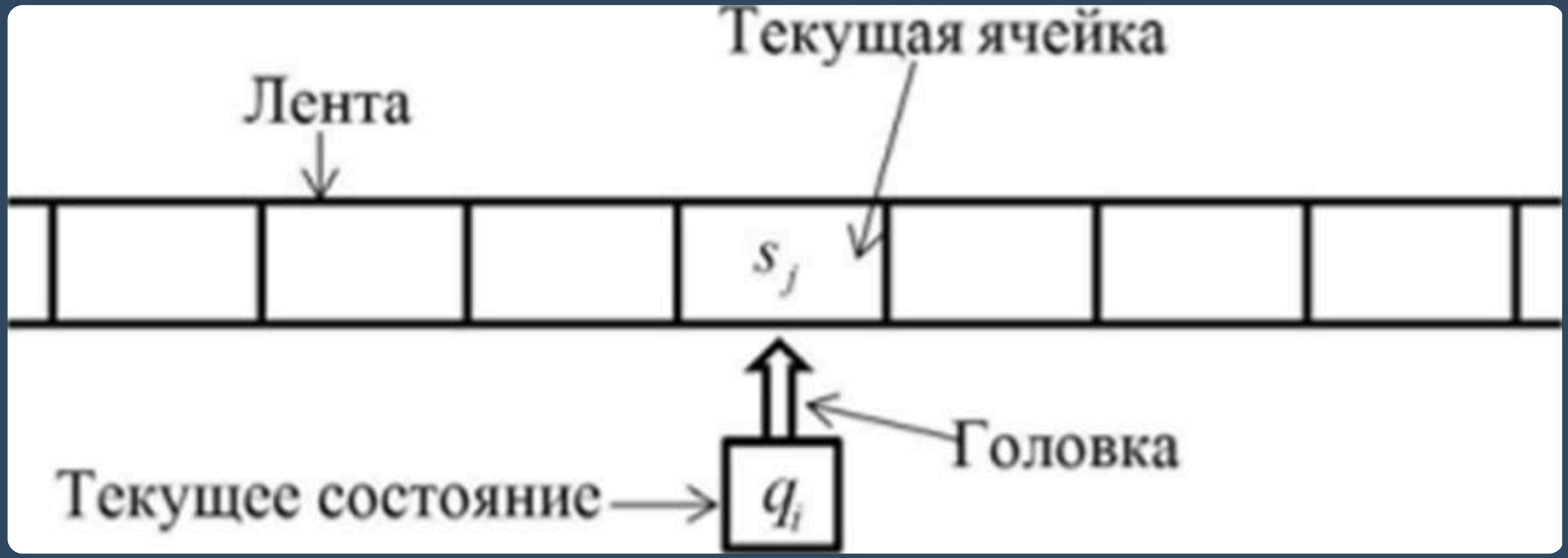
Способы формализации

Следование	`f(n) = <операнд> <оператор> <операнд>....`
Ветвление	
Цикл while	`f(x) = f(x - 1)x, при x > 0, 1 иначе`

Алгебра



Способы формализации



Математические исполнители

Проблема остановки

- Машина Тьюринга: математическая модель исполнителя команд, состоящего из бесконечной ленты, перемещающейся головки считывателя и его состояния

- Тезис Чёрча – Тьюринга: всё, что можно вычислить алгоритмом, можно вычислить машиной Тьюринга

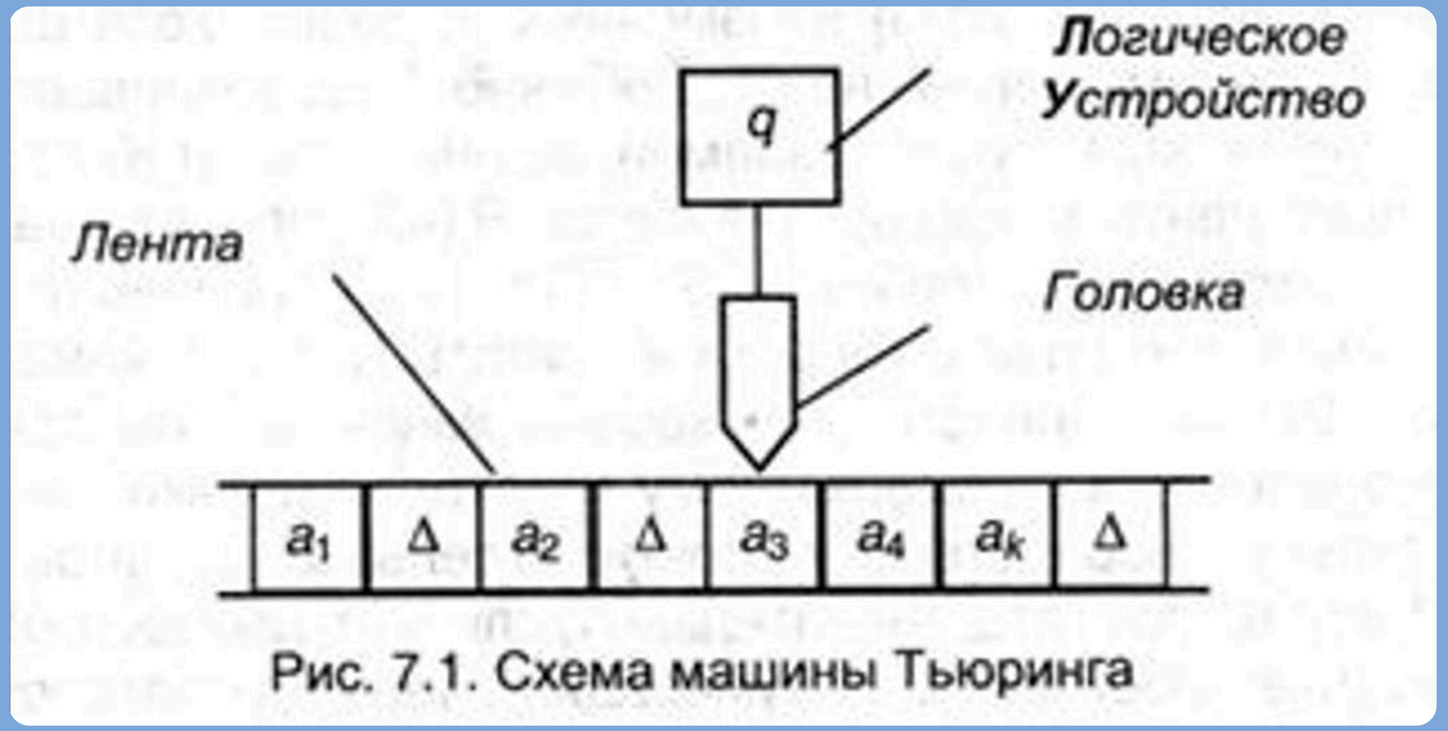


Рис. 7.1. Схема машины Тьюринга

Вопрос: можно ли написать программу, которая для любой программы скажет, завершится ли она?

Проблема остановки

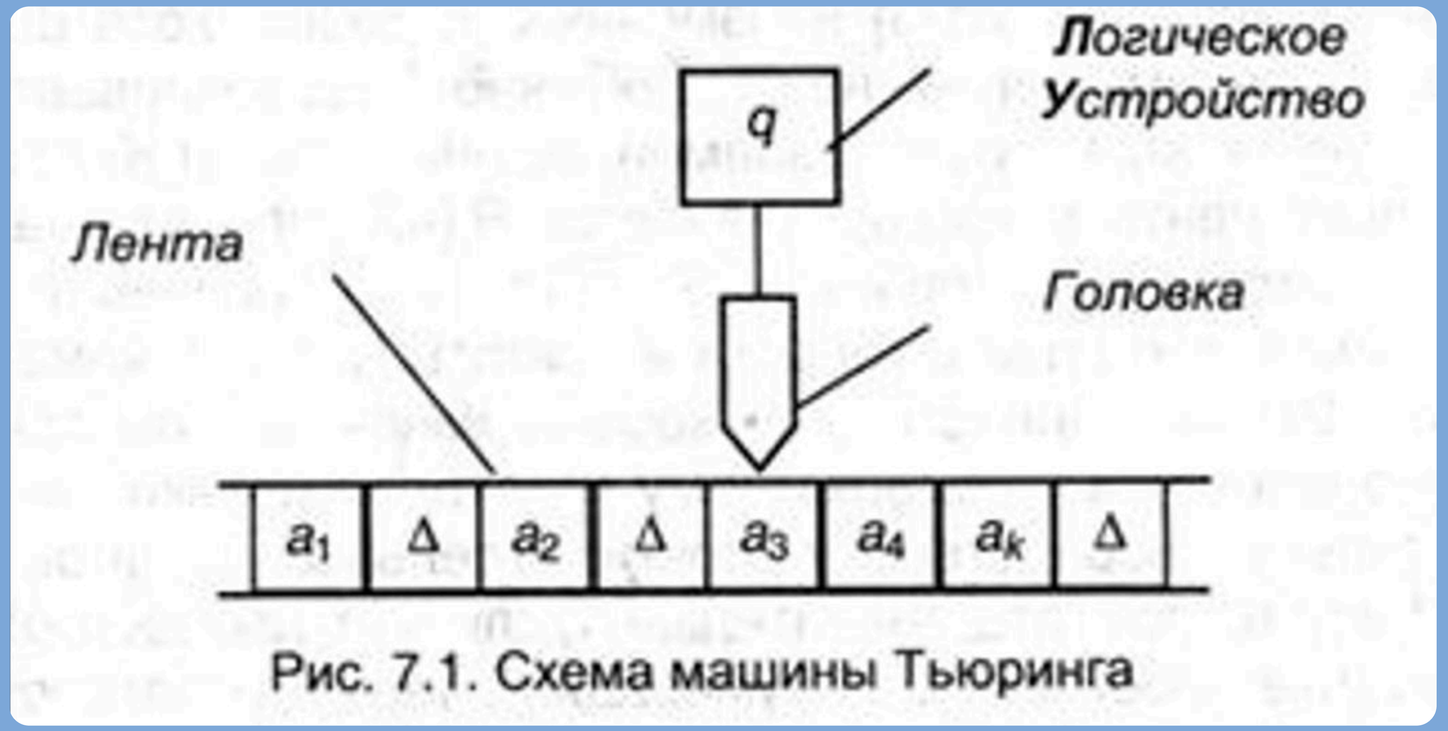


Рис. 7.1. Схема машины Тьюринга

нет, нельзя.
как следствие, это значит, что есть задачи, которые невозможно решить машиной тьюринга, а значит и создать для нее алгоритм.

АРХИТЕКТУРА ЭВМ



Биты

Многие знают, что в компьютере всё работает на двоичной системе счисления, т.е. представляется в виде 0 и 1.

Но нам привычнее работать с десятичной системой счисления.

Вопрос: Почему так?



БИТЫ

Ответ: Компьютеры - электро-механические машины. Легко различить наличие и отсутвие тока, но не уровни “силы”.



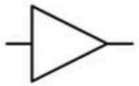
Булева логика

С помощью 0 и 1 можно представить факт наступления или ненаступления какого-либо события.

На основе этого можно построить способ представление логических цепочек (например, событие А наступит, если Б и В).

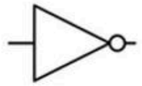
Операндами такой алгебры являются утверждения об истинности фактов или наступлении событий, а операторами логические условия (вентили).

Buffer



Input	Output
0	0
1	1

Inverter



Input	Output
0	1
1	0

AND



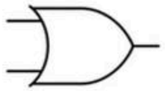
A	B	Output
0	0	0
1	0	0
0	1	0
1	1	1

NAND



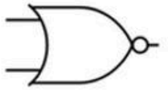
A	B	Output
0	0	1
1	0	1
0	1	1
1	1	0

OR



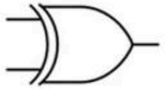
A	B	Output
0	0	0
1	0	1
0	1	1
1	1	1

NOR



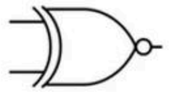
A	B	Output
0	0	1
1	0	0
0	1	0
1	1	0

XOR



A	B	Output
0	0	0
1	0	1
0	1	1
1	1	0

XNOR

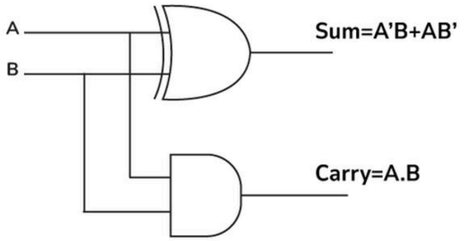


A	B	Output
0	0	1
1	0	0
0	1	0
1	1	1



Применение булевой логики

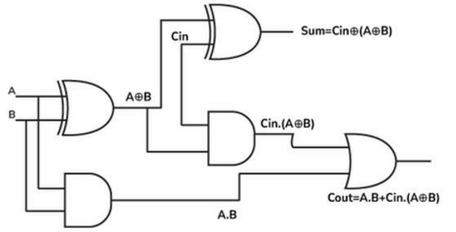
Half Adder



Truth Table

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Full Adder



Inputs			Outputs	
A	B	C	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Примем как операнды, факты о значении разряда двух чисел, а как операцию их сложение.
Возможно используя логические операторы определить, будет ли перенос при их сложении значение в разряде числа

Кодирование

Кодировка - это взаимно-однозначное отображение одного конечного множества в другое

Кодирование позволяет представить различные формы информации в других формах, в том числе в виде последовательности битов.
Если же представить что-то в виде последовательности битов, то это можно обработать булевой алгеброй!

Способы кодирования:

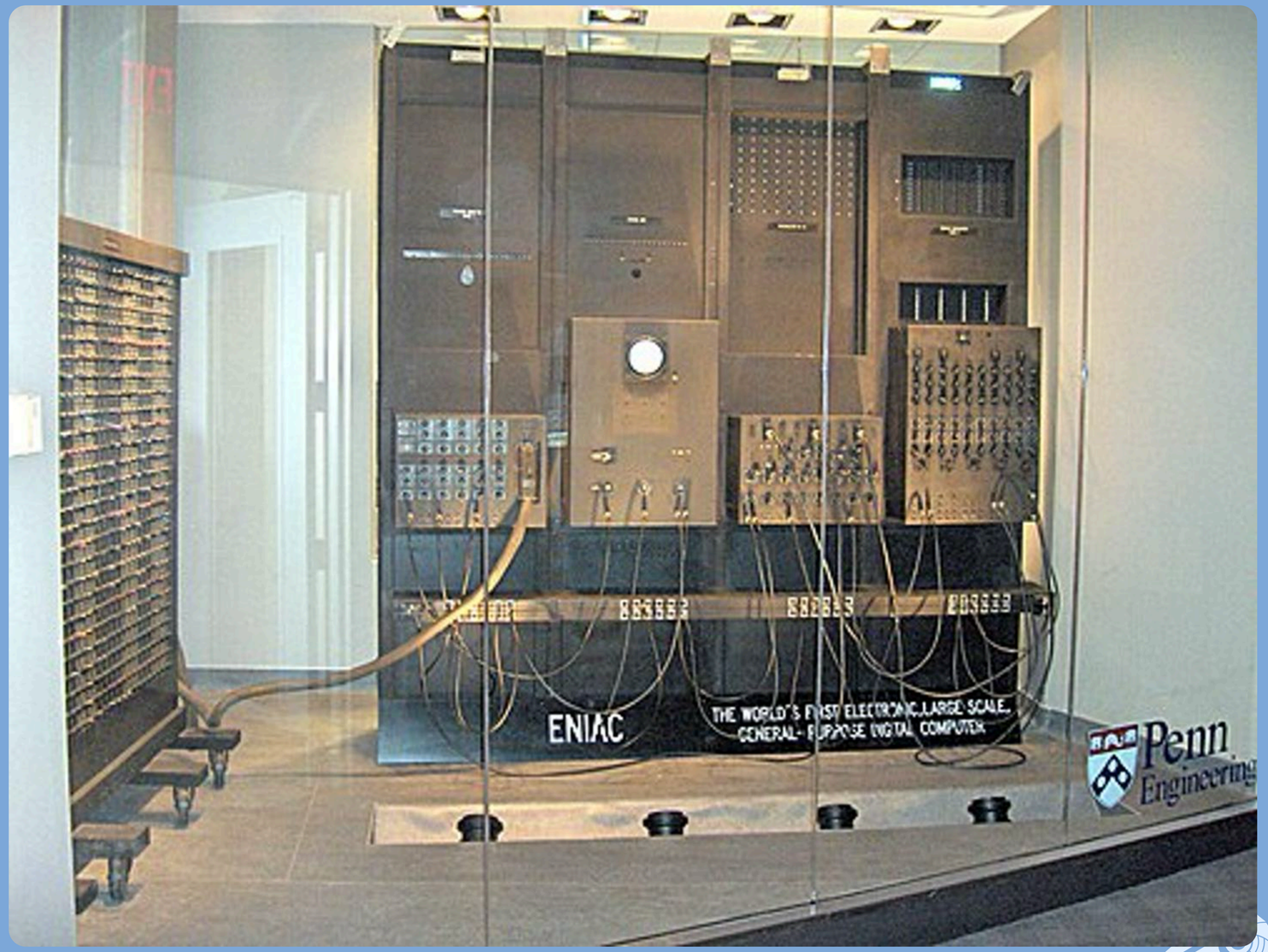
- По правилу: Числа, цвета
- По таблице: Текст

Одни и те же биты означают разное в зависимости от того, как договорились их читать.

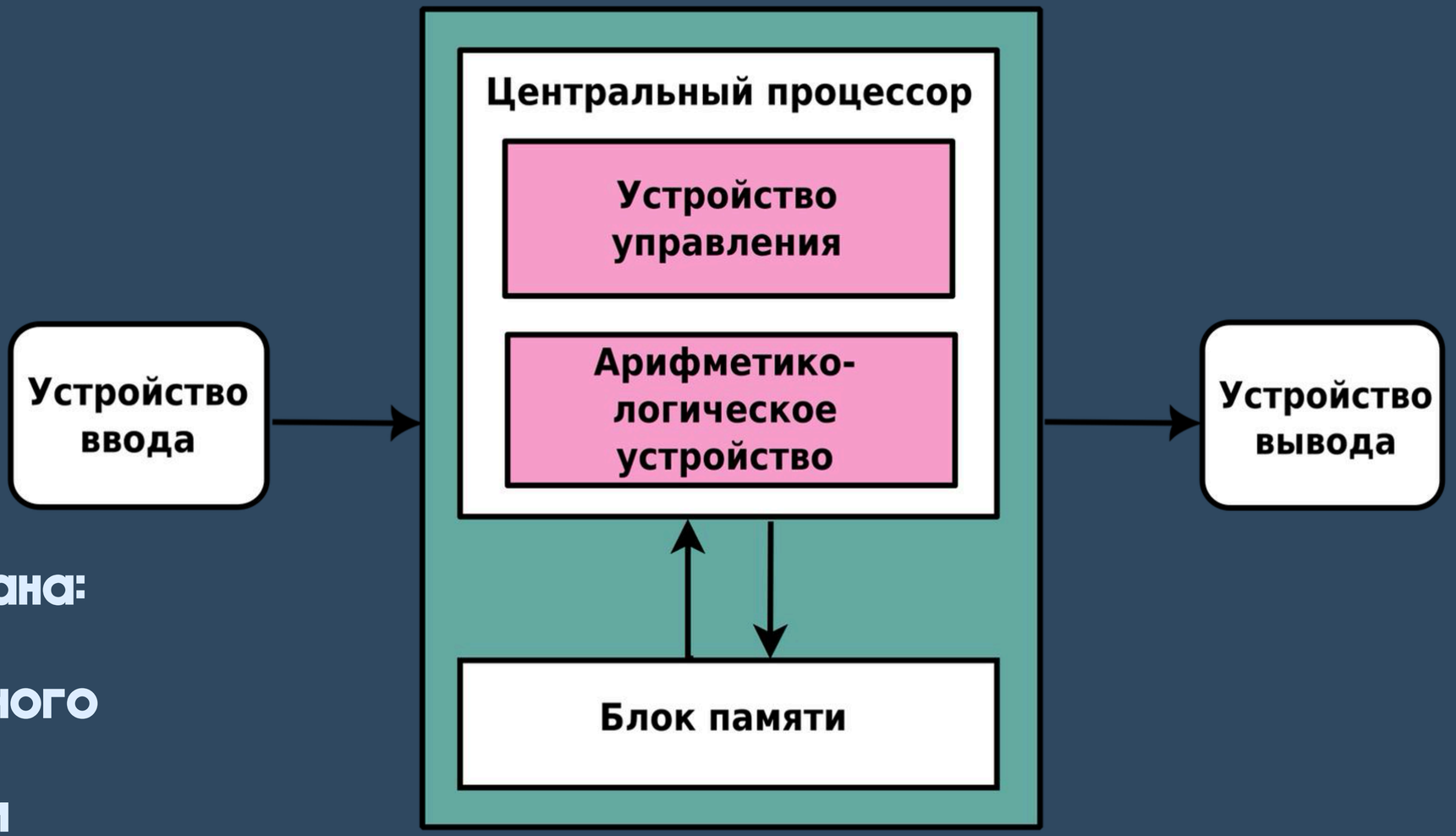


Архитектура ЭВМ

Первые ЭВМ были основаны на бинарной логике и арифметике, но программировались с помощью перекоммутации проводов



Кодирование



Принципы Фон Неймана:

- Принцип программного управления
- Принцип адресности
- Принцип однородности памяти
- Принцип бинарности

Кодирование

Работа CPU циклическая и состоит из строгой последовательности фаз:

- Fetch (Выборка): УУ считывает из памяти следующую команду.

- Decode: УУ расшифровывает полученную команду (разделение команды на операнды и оператор).

- Execute: УУ подает сигналы другим компонентам. Если команда подразумевает операцию, она выполняется в АЛУ. Если нужен доступ к данным - они считываются из памяти.

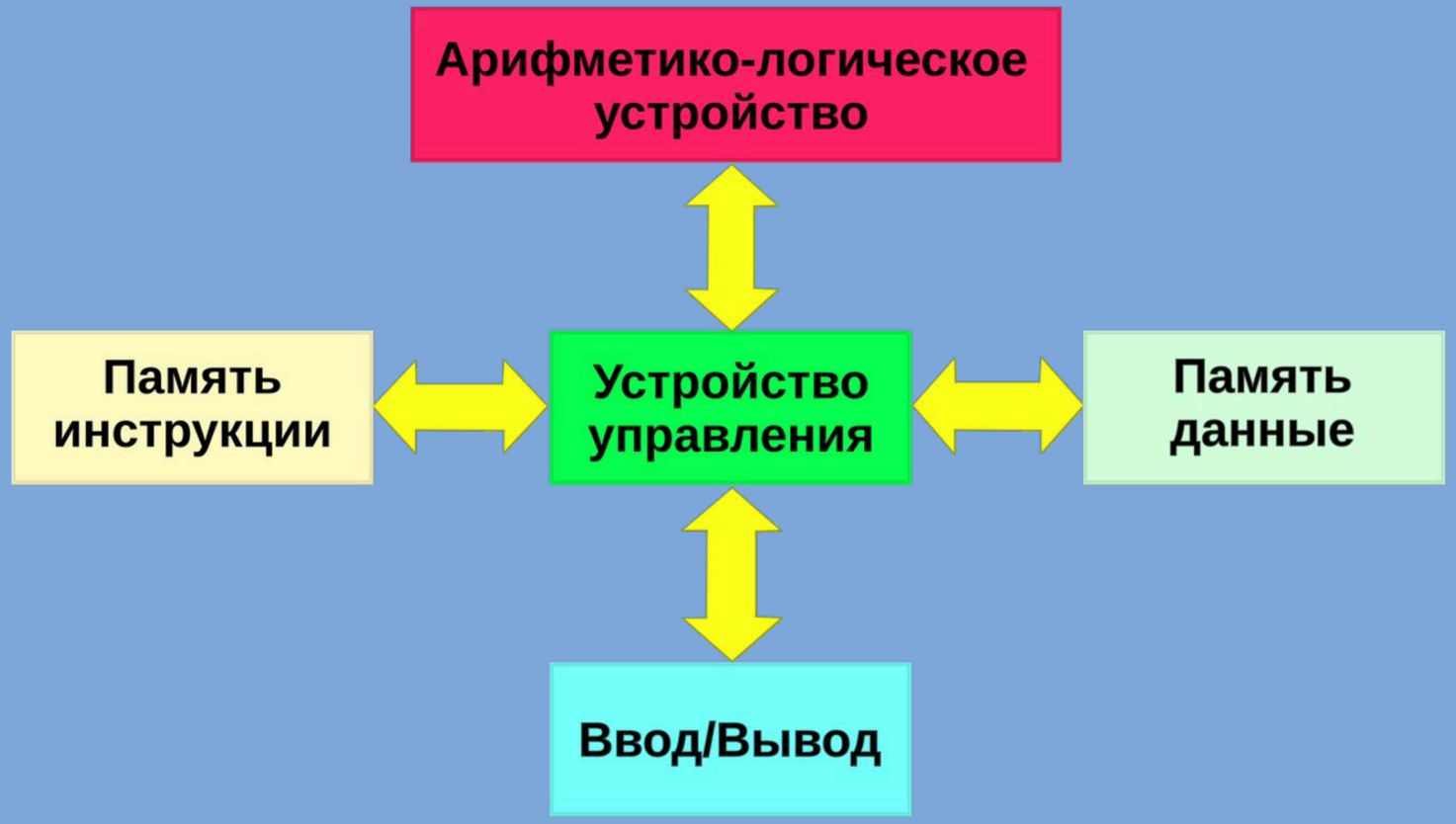
- Writeback: Результат операции записывается обратно в регистр CPU или в память.

- Инкремент PC: Счетчик команд обновляется для указания на следующую инструкцию. Цикл повторяется.



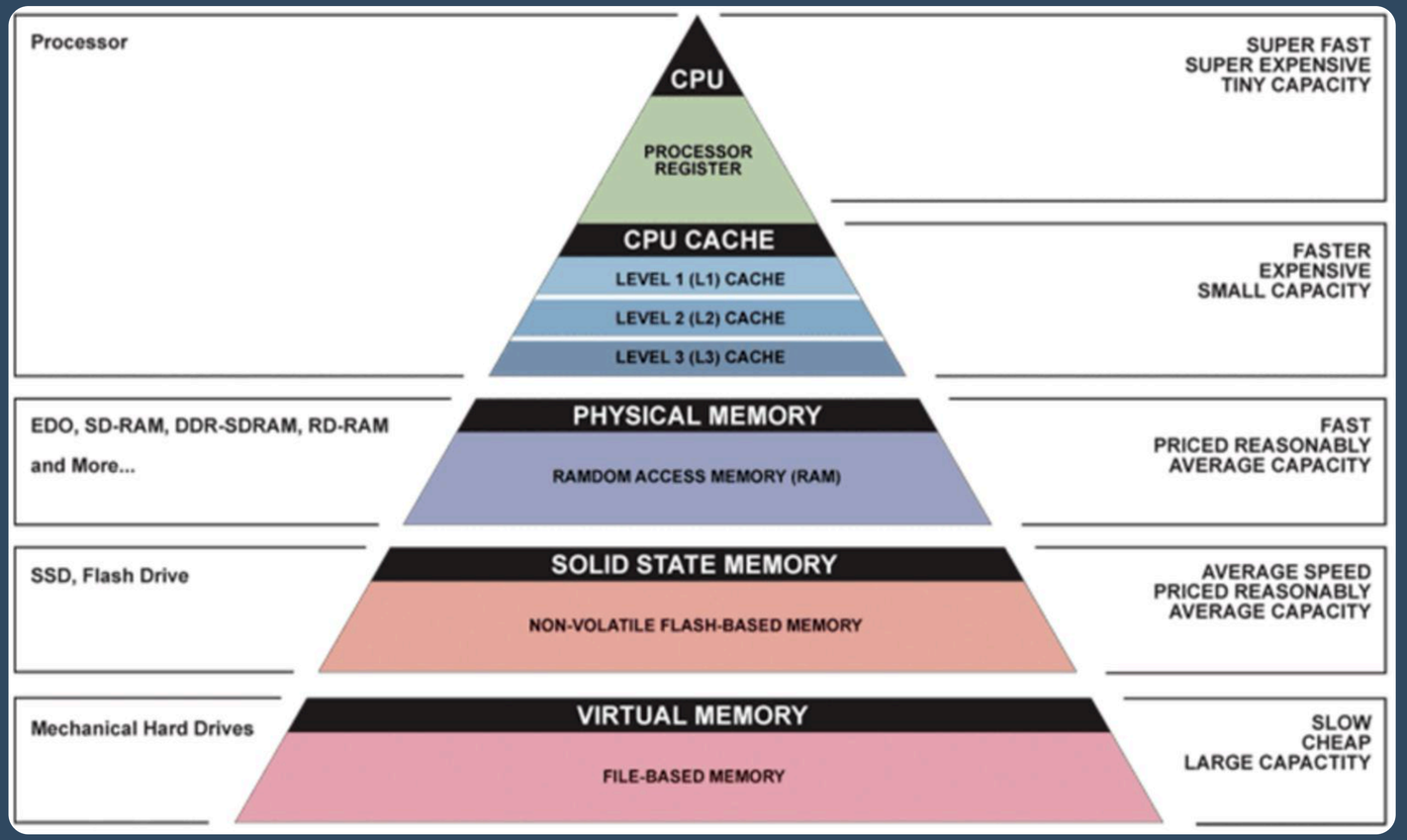
Гарвардская архитектура

В отличие от архитектуры Фон Неймана, в этом варианте возможна одновременное чтение и данных и инструкций, но требует более сложной системы управления



Иерархия памяти

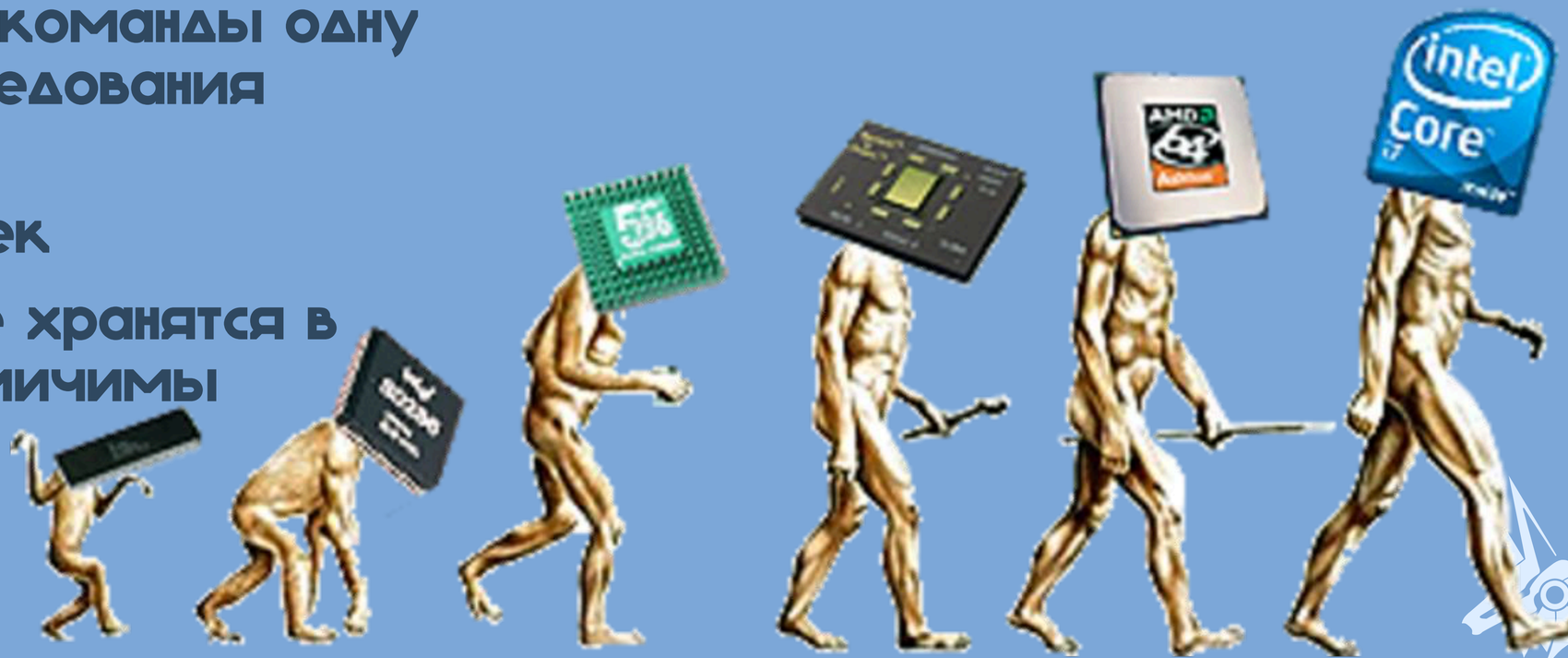
В действительности, есть различные виды памяти, обращение к которым занимает различное количество тактов процессора.
Более того, в зависимости от устройства, работа с памятью может быть совершенно разной.



Выводы из архитектуры

Исходя из устройства современных ЭВМ имеем следующие факты:

- Компьютер состоит из процессора, памяти и устройств ввода-вывода
- Процессор исполняет команды одну за одной в порядке следования
- Память состоит из пронумерованных ячеек
- Программы и данные хранятся в одном месте и не различимы

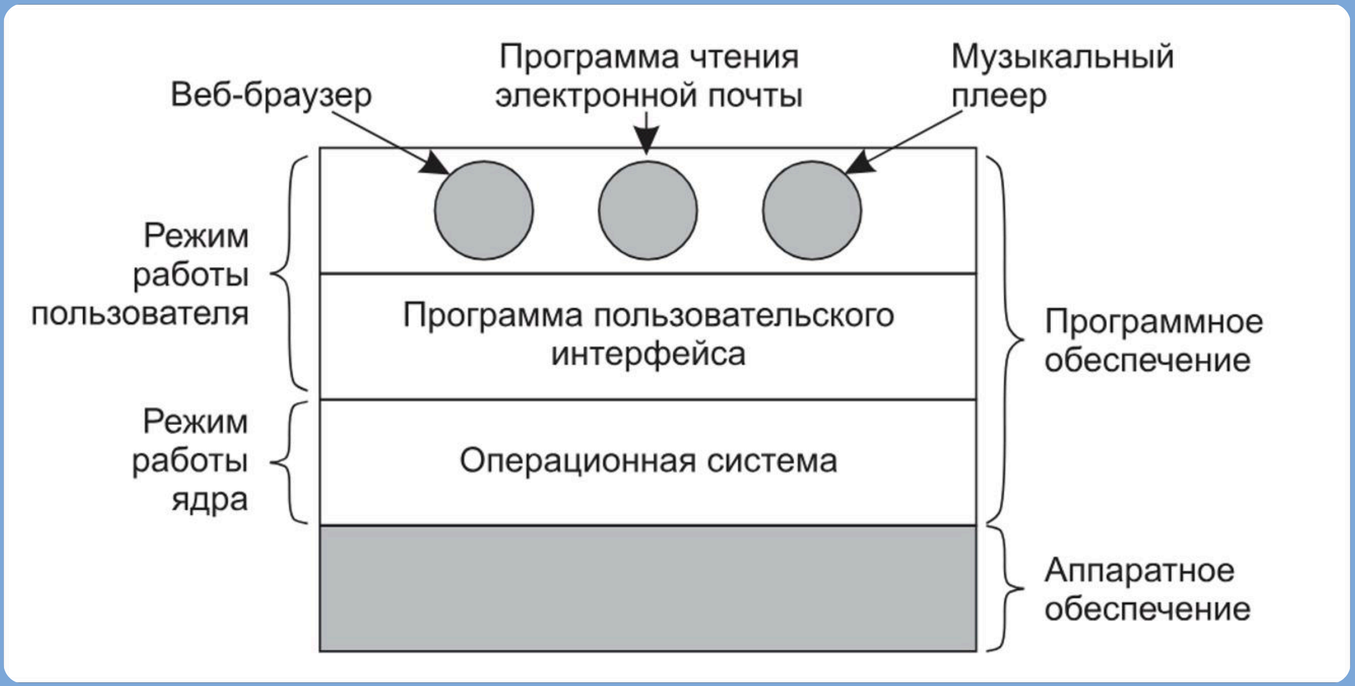


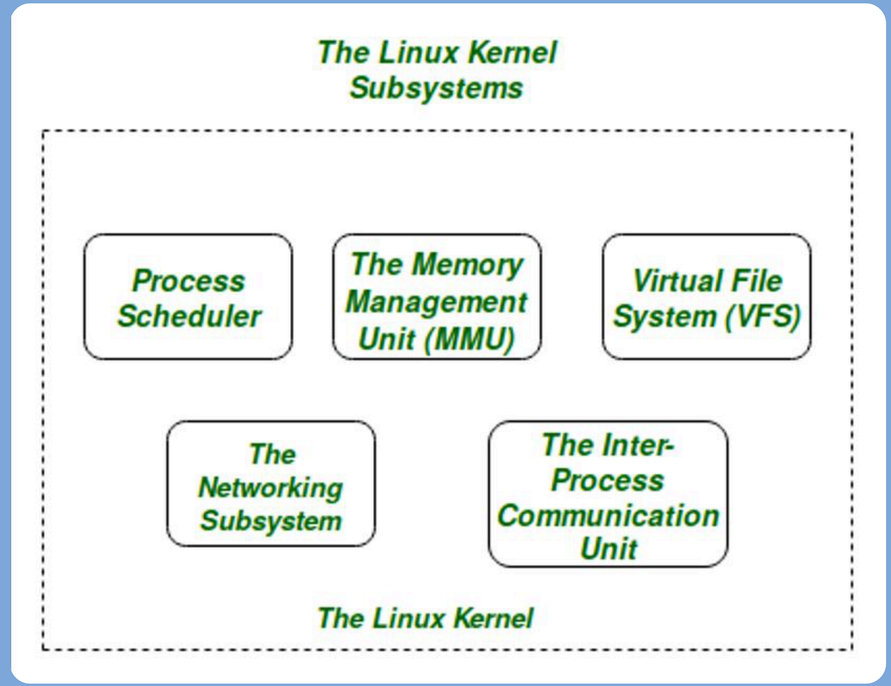
ОПЕРАЦИОННЫЕ СИСТЕМЫ



Операционная система

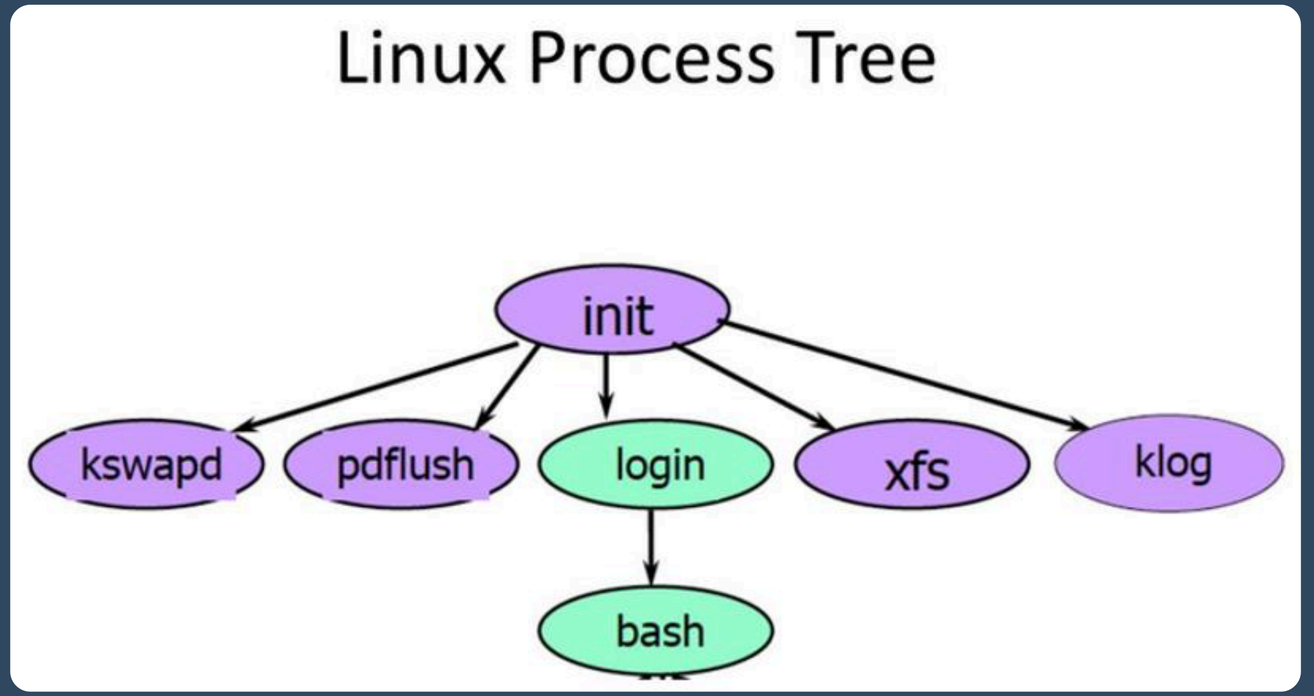
Операционная система - это ПО, которое предоставляют прикладным программистам (и
прикладным программам, естественно) вполне понятный абстрактный набор ресурсов взамен
неупорядоченного набора аппаратного обеспечения и управляют этими ресурсами.





Управление процессами

Проблема: Компьютер один, а программ хочется запустить много и не ограничивать себя. Как дать каждой программе возможность выполняться одновременно или так, чтобы мы не замечали их последовательность?



Управление процессами

Переключение контекста - ОС сохраняет состояние процесса (регистры, счётчик команд) и восстанавливает состояние другого. Именно это создаёт иллюзию одновременности.

Решение: Операционная система хранит информацию о запущенных программах и переключает их.

Процесс - это программа, в стадии выполнения (переменные, код, значения регистров, состояние и д.р.)

Планировщик - часть ядра ОС, которая решает, какому процессу (или потоку) дать доступ к CPU в следующий момент времени.

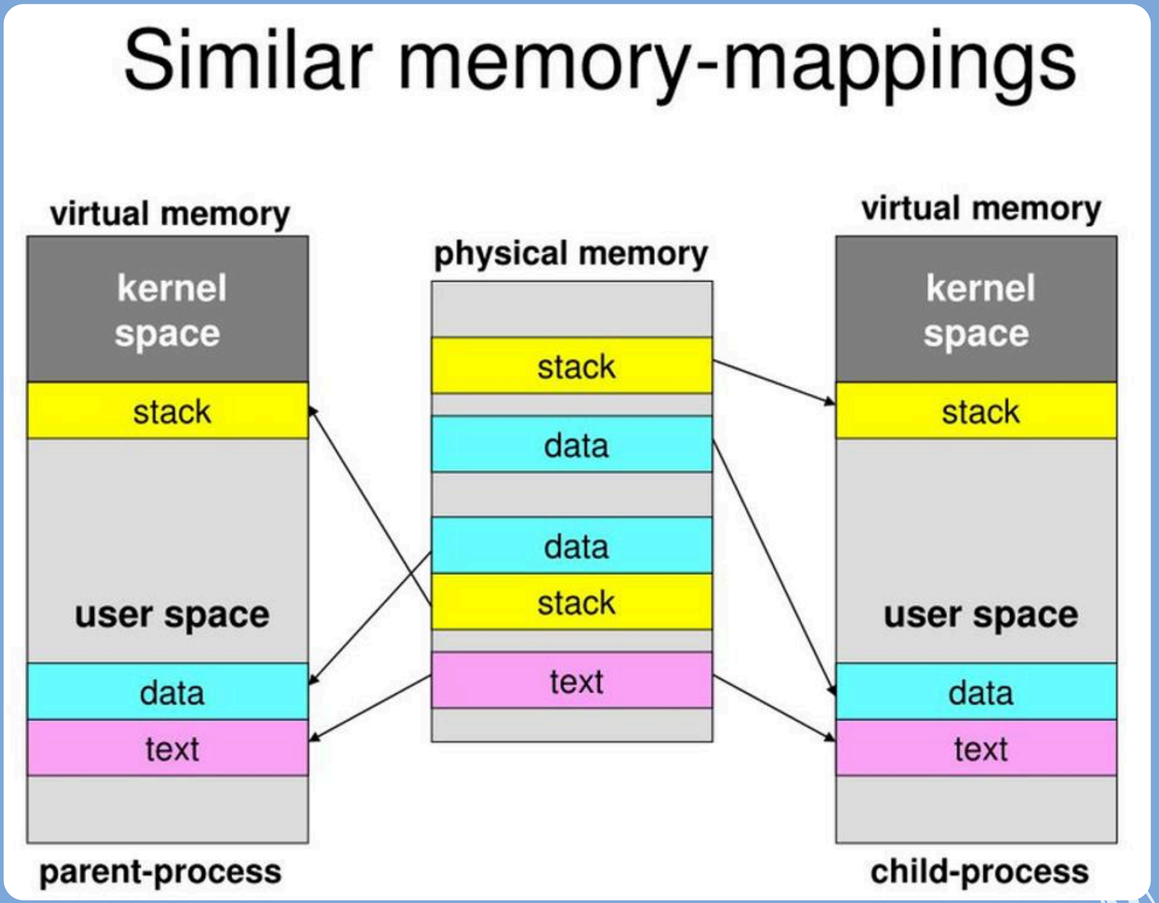


Управление памятью

Проблема:

У всех программ одна физическая память (RAM).

- **Как поделить память между всеми процессами?**
- **Как изолировать их друг от друга?**



Управление памятью

Решение: Виртуальная память, механизм, который предоставляет программам иллюзию того, что они обладают не ограниченной (ограниченным размером машинного слова) оперативной памятью и, при обращении программы к участкам виртуальной памяти, транслирует их на участки физической памяти компьютера.
Изоляция процессов достигается за счёт того, что процесс не имеет доступ к той физической памяти, на которую не ссылается не один из участков его виртуальной памяти.

Страница - блок памяти фиксированного размера (обычно 4 КБ), единица трансляции
MMU - блок процессора, который эту трансляцию выполняет аппаратно



Управление УВВ

Проблема: есть множество разных устройств для хранения информации и обмена ей, у каждого из которых свои способы хранения информации и взаимодействия с процессором.

Невозможно разработать ПО так, чтобы оно само работало с каждым из устройства ввода-вывода (их слишком много).

Решение: Файловая система это порядок, определяющий способ организации, хранения и именования данных на носителях информации в компьютерах.

Файл - это именованная область данных на носителе информации.

Файлы позволяют работать с данными как с единым целым независимо от того, как устроены данные в устройстве.



Управление УВВ

Решение:

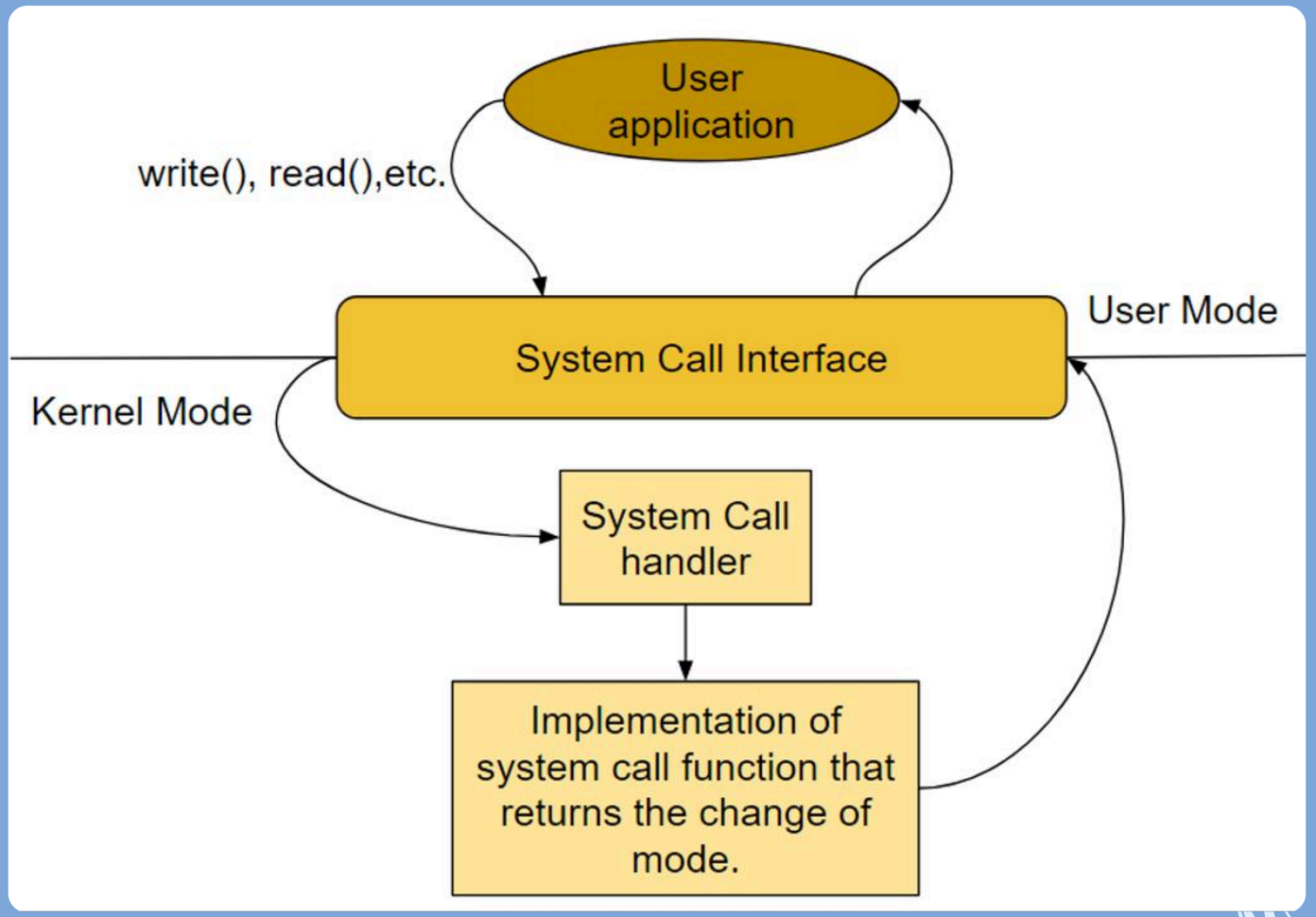
Драйверы - специальные программы, которые написаны в соответствии с определёнными требованиями операционной системы, и которые предоставляют установленный интерфейс для работы с устройствами, независимо от деталей. Драйверы могут быть загружены в ядро операционной системы без пересборки.



Предоставление API

Проблема:

Разработчикам прикладных программ нужно получать доступ к аппаратным ресурсам системы, при этом не сломав остальные процессы и саму систему.



Предоставление API

Решение:

Операционная система предоставляет интерфейс прикладного программирования (API) - набор системных вызовов (syscalls).

По умолчанию программа выполняется в режиме пользователя - без доступа к ресурсам системы.

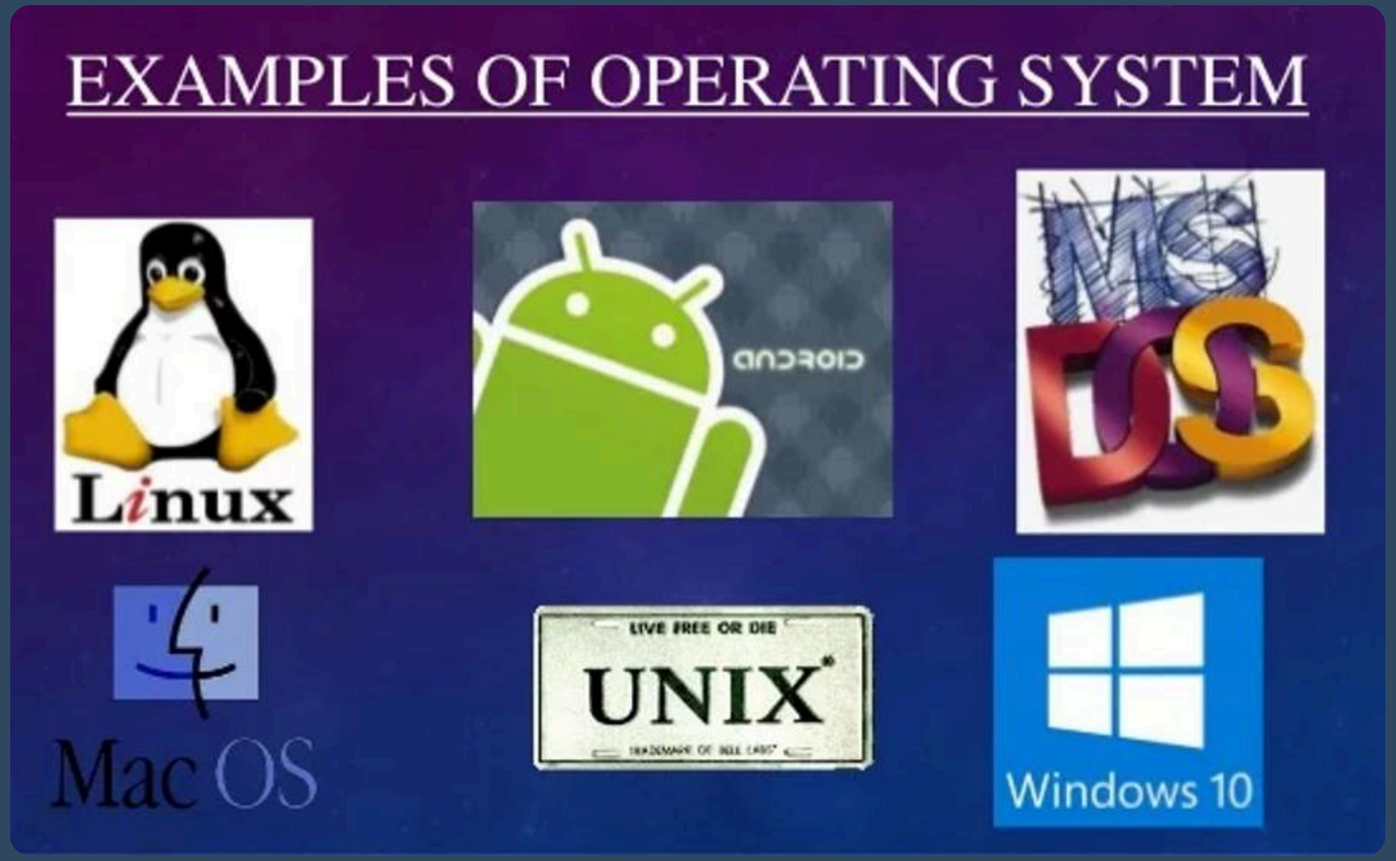
Системный вызов - функция, которая переводит программу в режим ядра, где доступны ресурсы системы.

Системные вызовы написаны разработчиками операционок - они знают, что делают.



Примеры OS

- Windows
- MacOS
- Linux
- UNIX
  - Генетический
  - По торговой марке
  - По функциональности
- Android



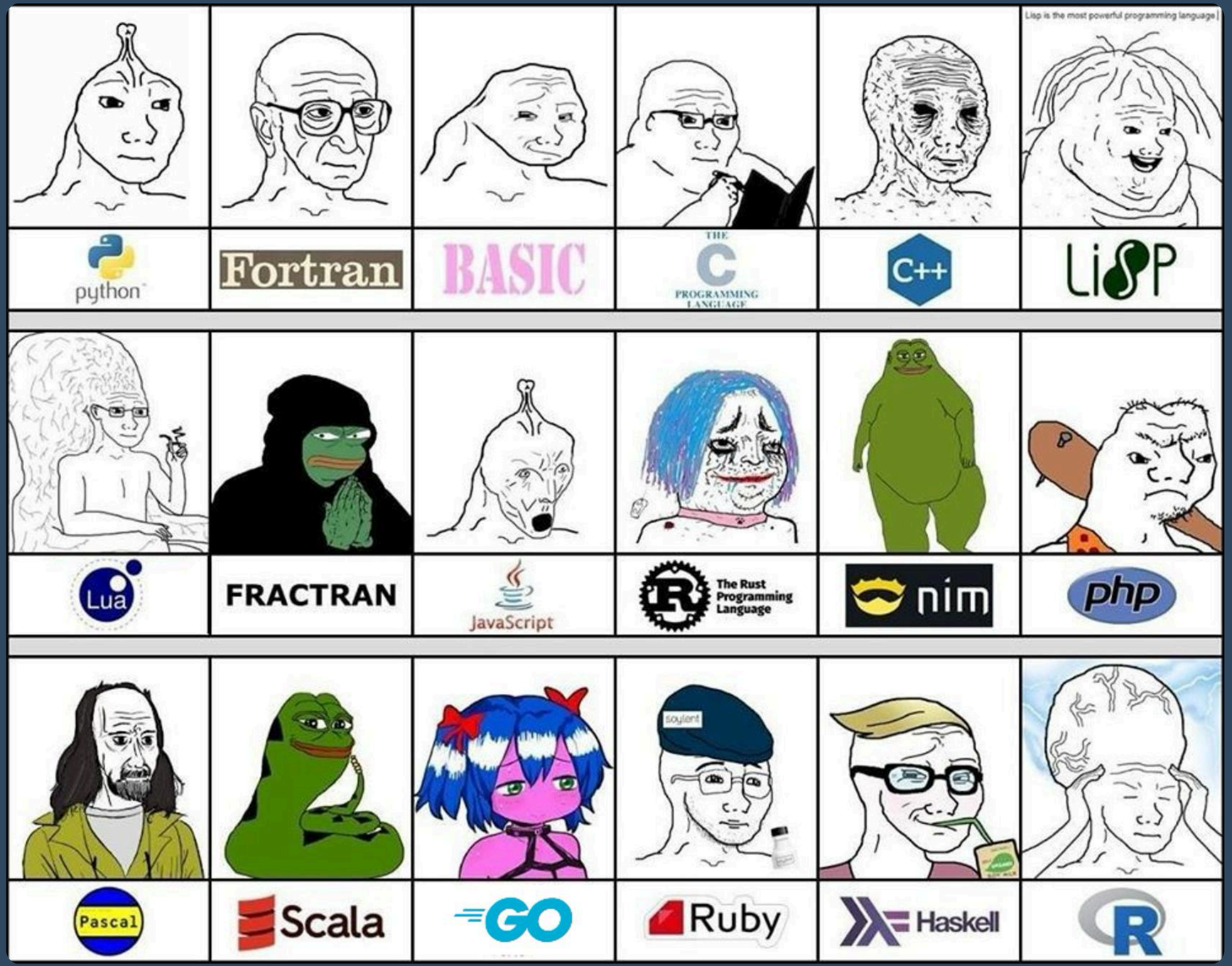
ЯЗЫКИ ПРОГРАММИРОВАНИЯ



ИНТЕРАКТИВ

Давайте попробуем вспомнить как можно больше языков программирования




Lisp is the most powerful programming language
python
Fortran
BASIC
THE C PROGRAMMING LANGUAGE
C++
LiSP
Lua
FRACTRAN
JavaScript
The Rust Programming Language
nim
php
Pascal
Scala
GO
soylent
Ruby
Haskell
R

Состовляющие

Язык программирования - это формальная знаковая система, предназначенная для записи компьютерных программ.

- **Синтаксис:** Набор правил, определяющих структуру и допустимые конструкций языка. Синтаксис можно подразделить на три части:

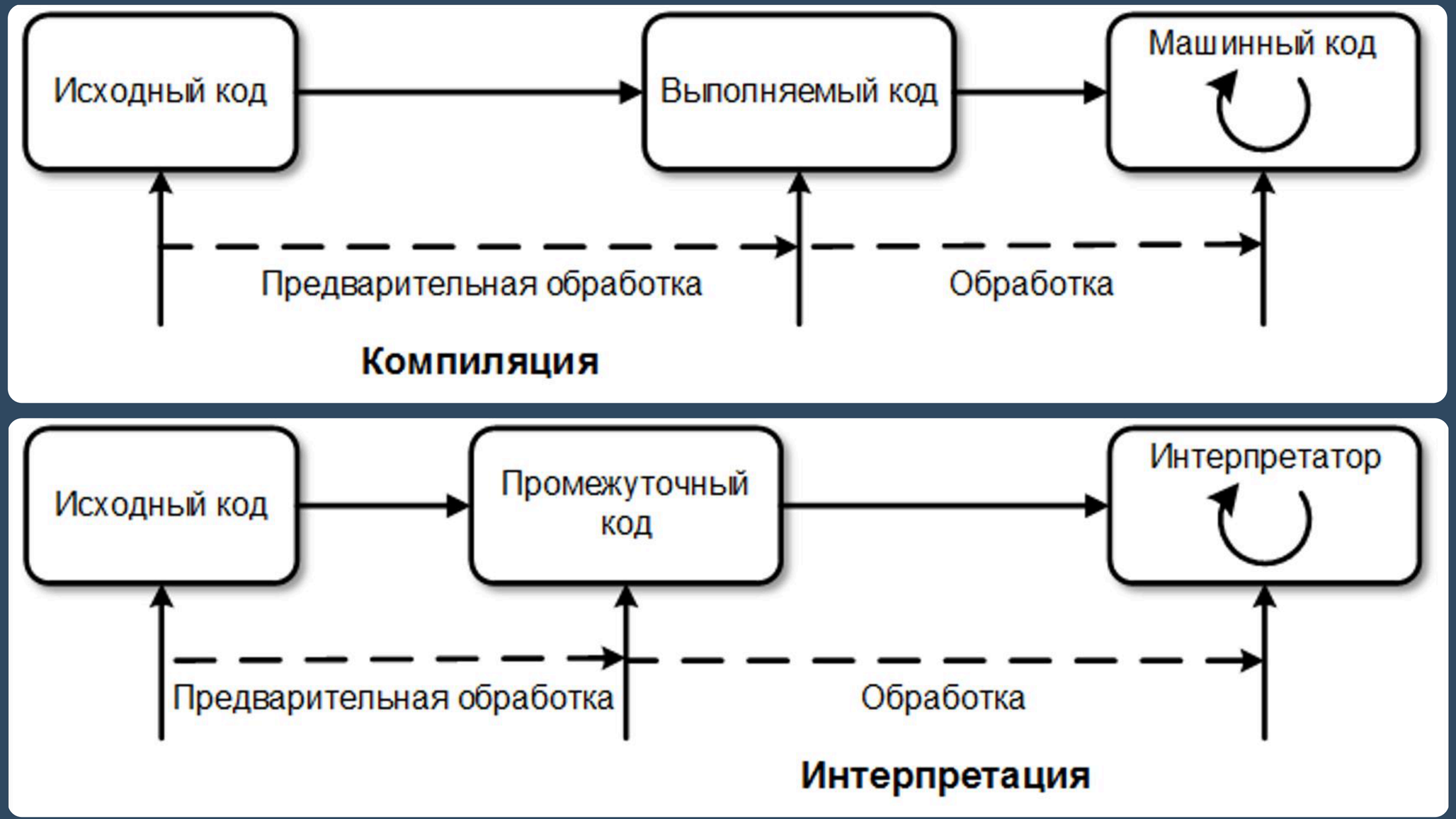
Алфавит: Определяет множество допустимых атомарных символов.
Лексемы (слова): Определяет множество допустимых комбинаций атомарных символов.
Грамматика: Определяет множество допустимых комбинаций слов.

- **Семантика:** Значение, которое несут синтаксические конструкции

- **Прагматика:** Практические аспекты использования языка.



Виды трансляции



Уровни абстракции

When I code some stupid stuff that will obviously break my computer:

Python

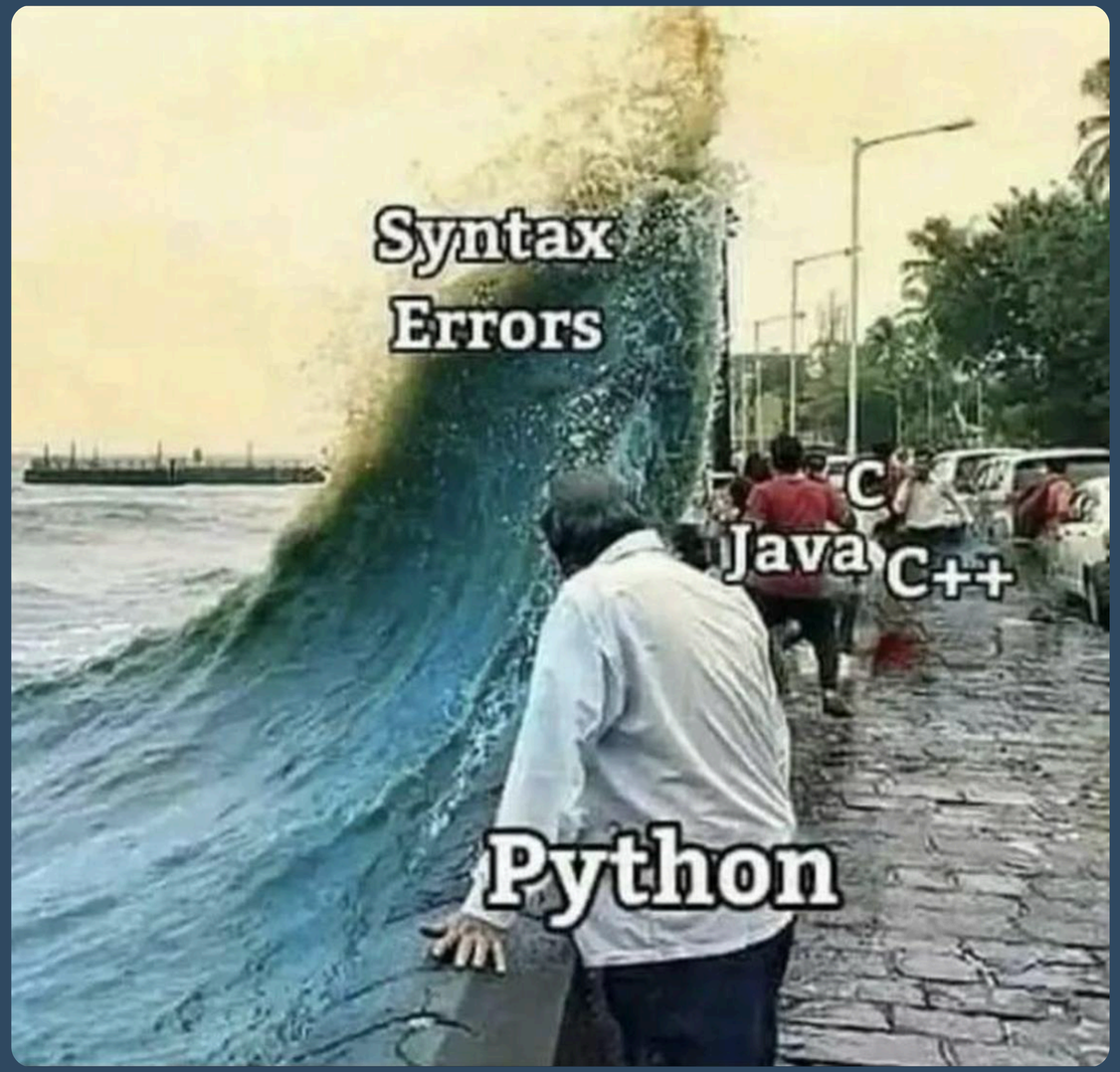


C++





Типизация



Вид	Описание	Примеры
Строгая (strict/static)	Типы проверяются на этапе компиляции, часто требуется их явно задать	Go, C, C++, TypeScript, Rust
Нестрогая (loose/dynamic)	Типы проверяются только на этапе работы программы	Python, JavaScript, Ruby

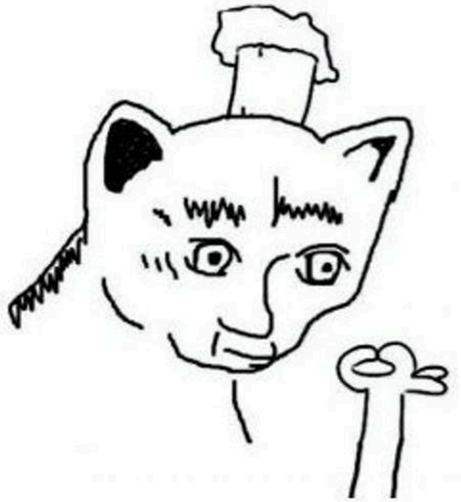
Вид	Описание	Примеры
Сильная (Strong)	Язык запрещает неявные преобразования типов	Python, Typescript, Rust, Go
Слабая (Weak)	Язык делает неявные преобразования типов, которые могут приводить к проблемам	Javascript, C, C++

Императивное программирование



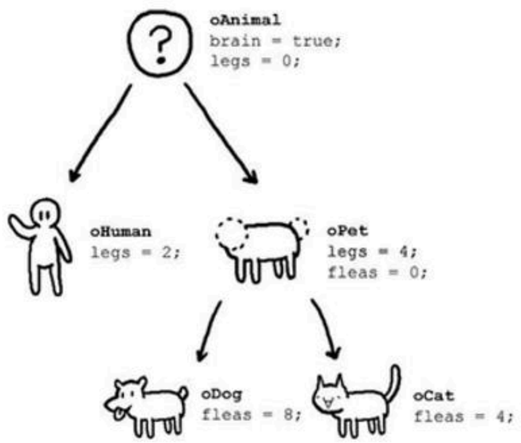
Императивное программирование

1. Взять пельмень
2. Сварить пельмень
3. Смазиком BELISSIMO

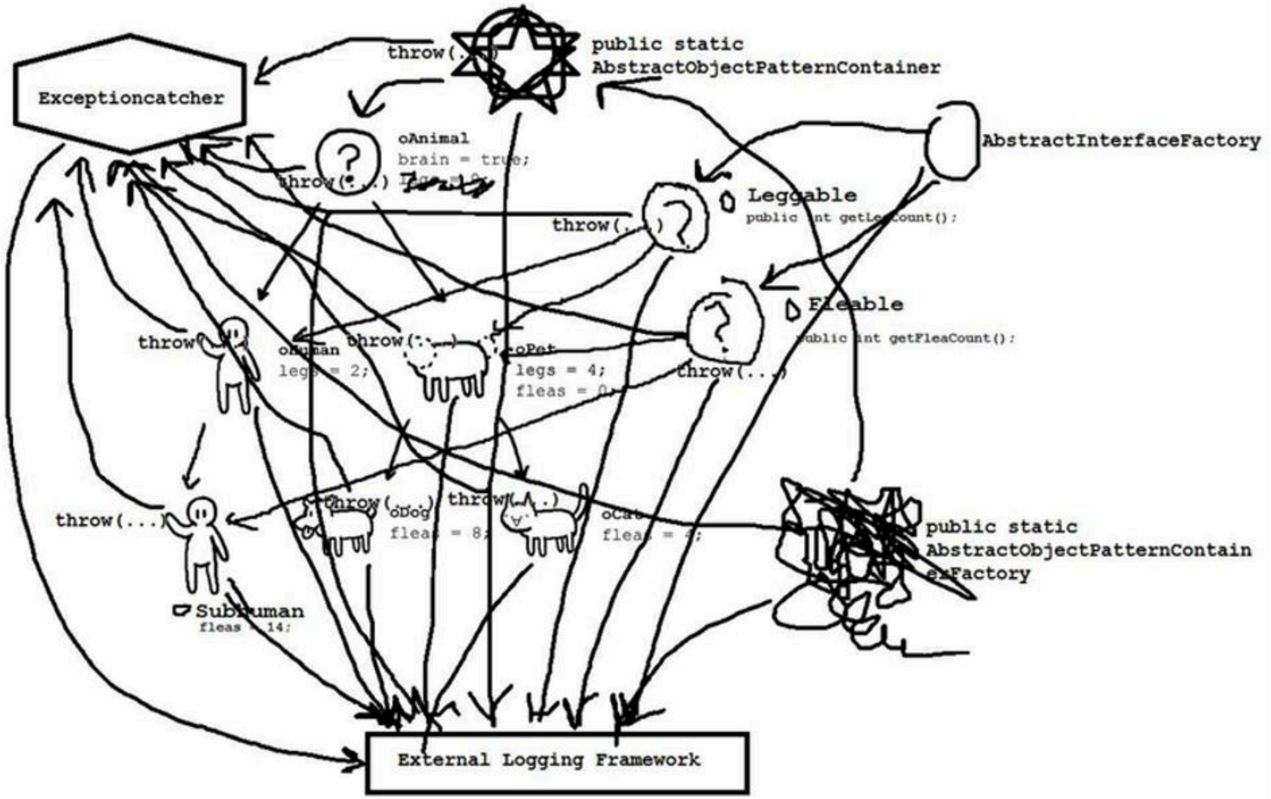


ООП

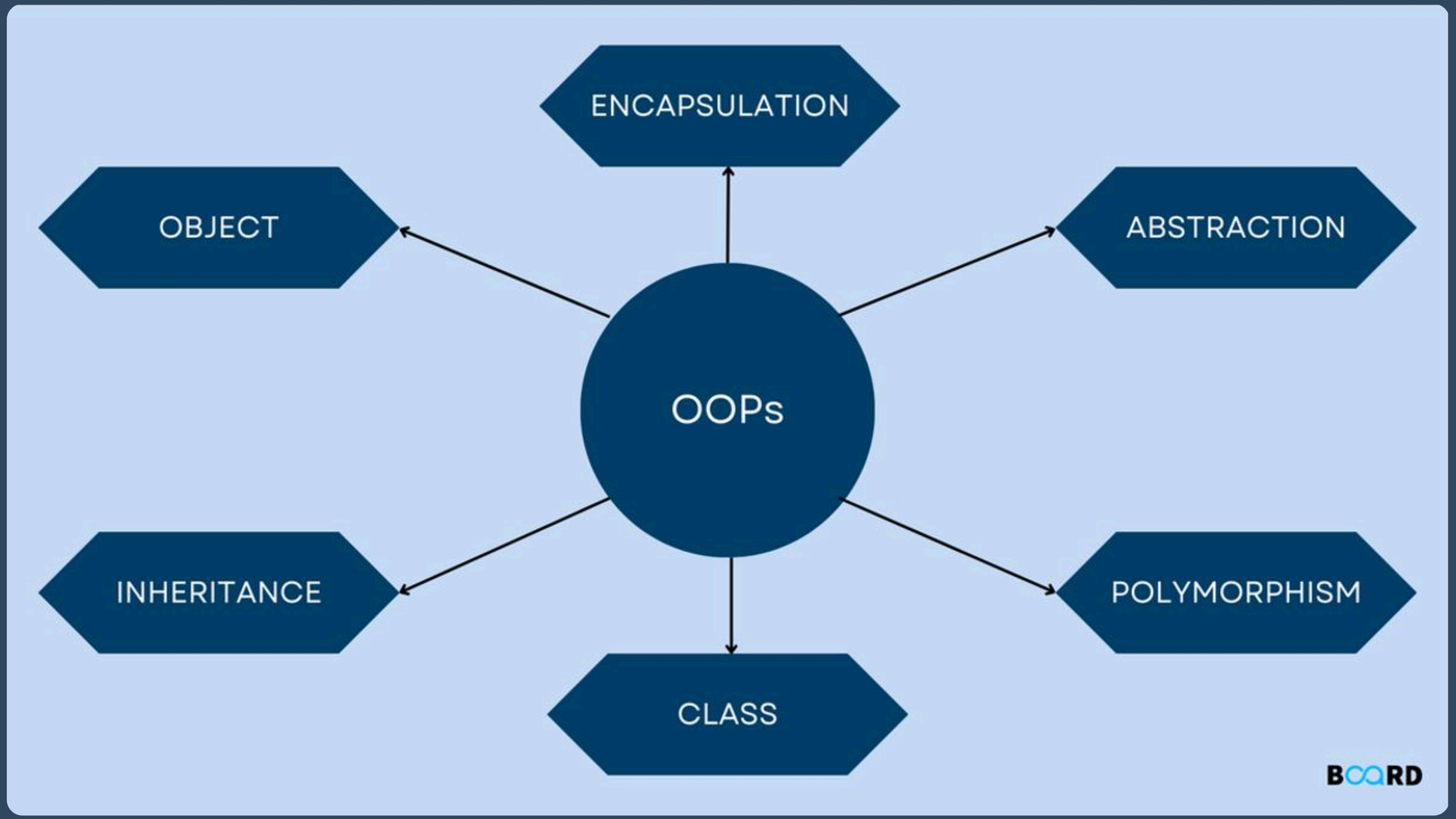
What OOP users claim



What actually happens

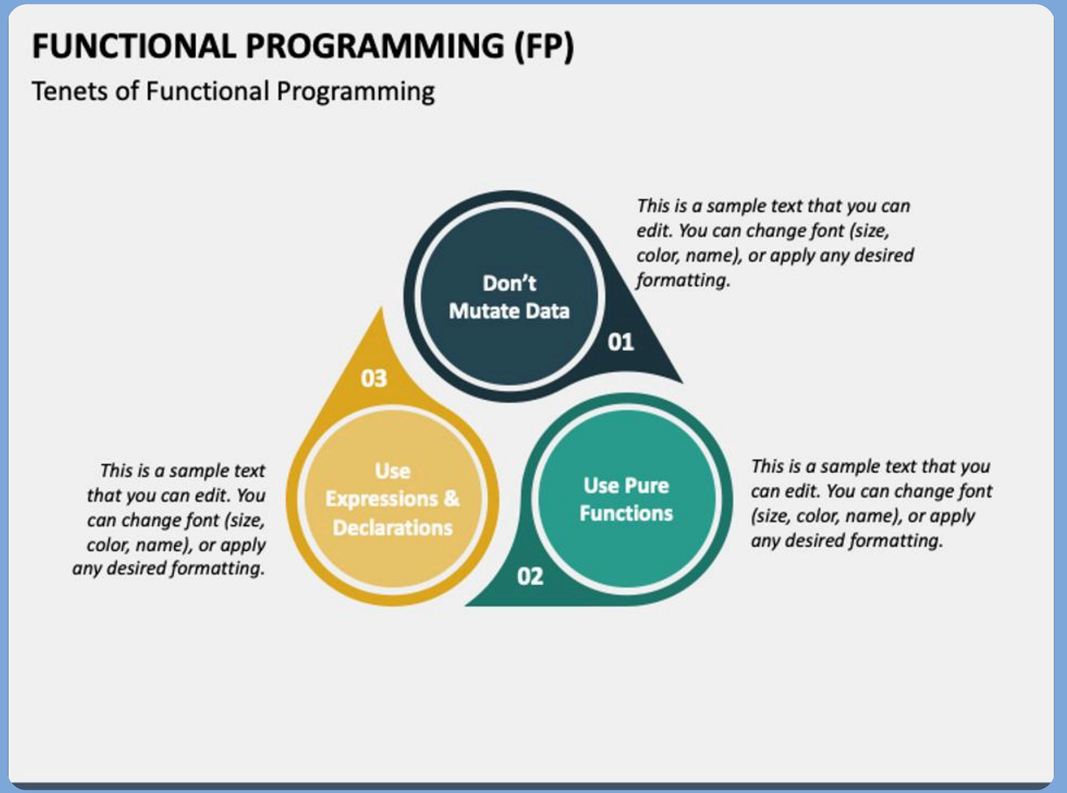


ООП


ENCAPSULATION
OBJECT
ABSTRACTION
OOPs
INHERITANCE
POLYMORPHISM
CLASS
BOARD

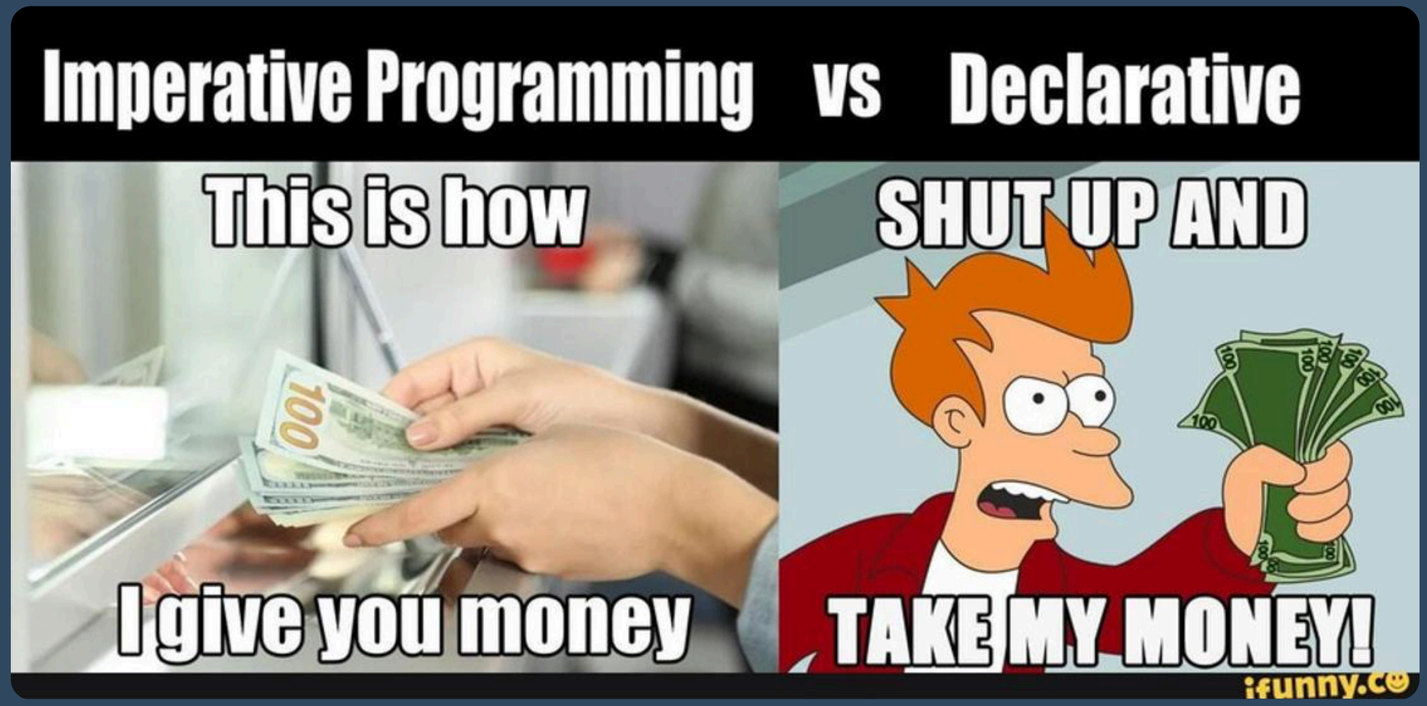
ФУНКЦИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ





ДЕКЛАРАТИВНОЕ ПРОГРАММИРОВАНИЕ





МЕТАПРОГРАММИРОВАНИЕ



```
namespace 🔵 = std;
using 🔢 = int;
using 💀 = void;
using 🕐 = time_t;
using 👌 = bool;
#define 👂 auto
#define 👀 enum
#define 👎 false
#define 👍 true
#define 😡 "evil"
#define 💪 🔵::make_shared
#define 🍸 virtual
#define 🖥️ 🔵::cout
#define 🚽 🔵::endl
template<class 🔮>
using 🚂 = 🔵::vector<🔮>;
template<class 🔮>
using 🖕 = 🔵::shared_ptr<🔮>;

👀 🤷 { 🙈, 🙉, 🙊, 🐵 };
🔢 🎲() { return 🔵::rand(); }
👌 😎() { return 👎; }

struct 🍴 { 🍸 💀 👐() = 0; };
struct 🍎 : 🍴 { 🍸 💀 👐() { 🖥️ << "🍎" << 🚽; }; };
struct 🍉 : 🍴 { 🍸 💀 👐() { 🖥️ << "🍉" << 🚽; }; };
struct 🍒 : 🍴 { 🍸 💀 👐() { 🖥️ << "🍉" << 🚽; }; };
struct 🍓 : 🍴 { 🍸 💀 👐() { 🖥️ << "🍓" << 🚽; }; };
struct 🍈 : 🍴 { 🍸 💀 👐() { 🖥️ << "🍈" << 🚽; }; };
struct 🍏 : 🍴 { 🍸 💀 👐() { 🖥️ << "🍏" << 🚽; }; };

🔢 main()
{
    if (😎() == 👎)
        🖥️ << "💩" << 🚽;

    🚂<🖕<🍴>> 🥘 = { 💪<🍎>(), 💪<🍉>(), 💪<🍒>(), 💪<🍈>(), 💪<🍏>() };

    for (👂 🍏 : 🥘)
        🍏->👐();
```

СПАСИБО ЗА ВНИМАНИЕ!



ОБРАТНАЯ СВЯЗЬ