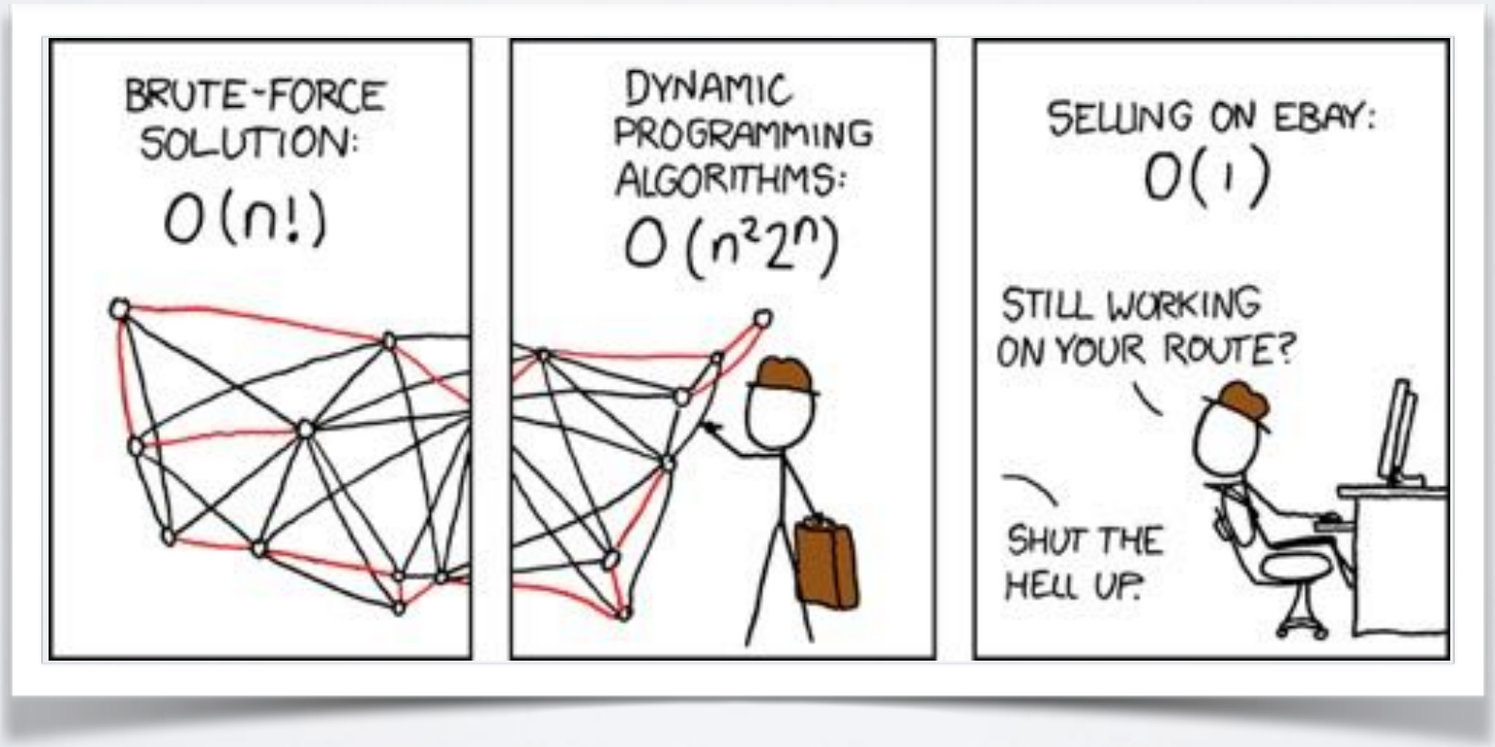
ОСНОВЫ ПРОГРАММНОГО КОНСТРУИРОВАНИЯ

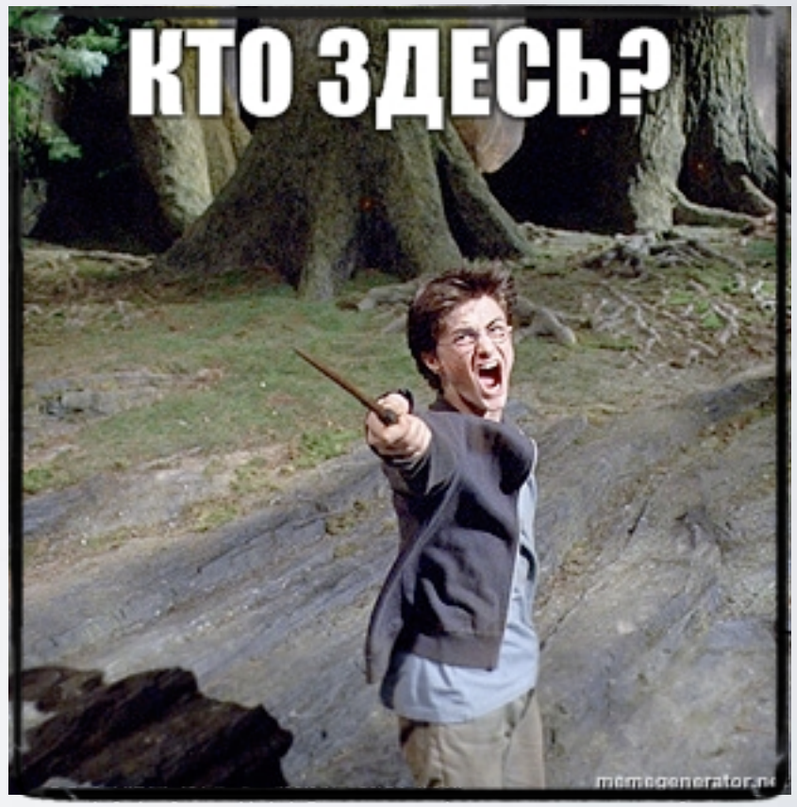
Лекция № 8

23 октября 2017 г.



ПРОФЕССИЯ: ПРОГРАММИСТ

- Пожарные борются с огнем.
- Педагоги борются с детьми.
- Полиция борется с преступностью.
- **Программисты борются …**



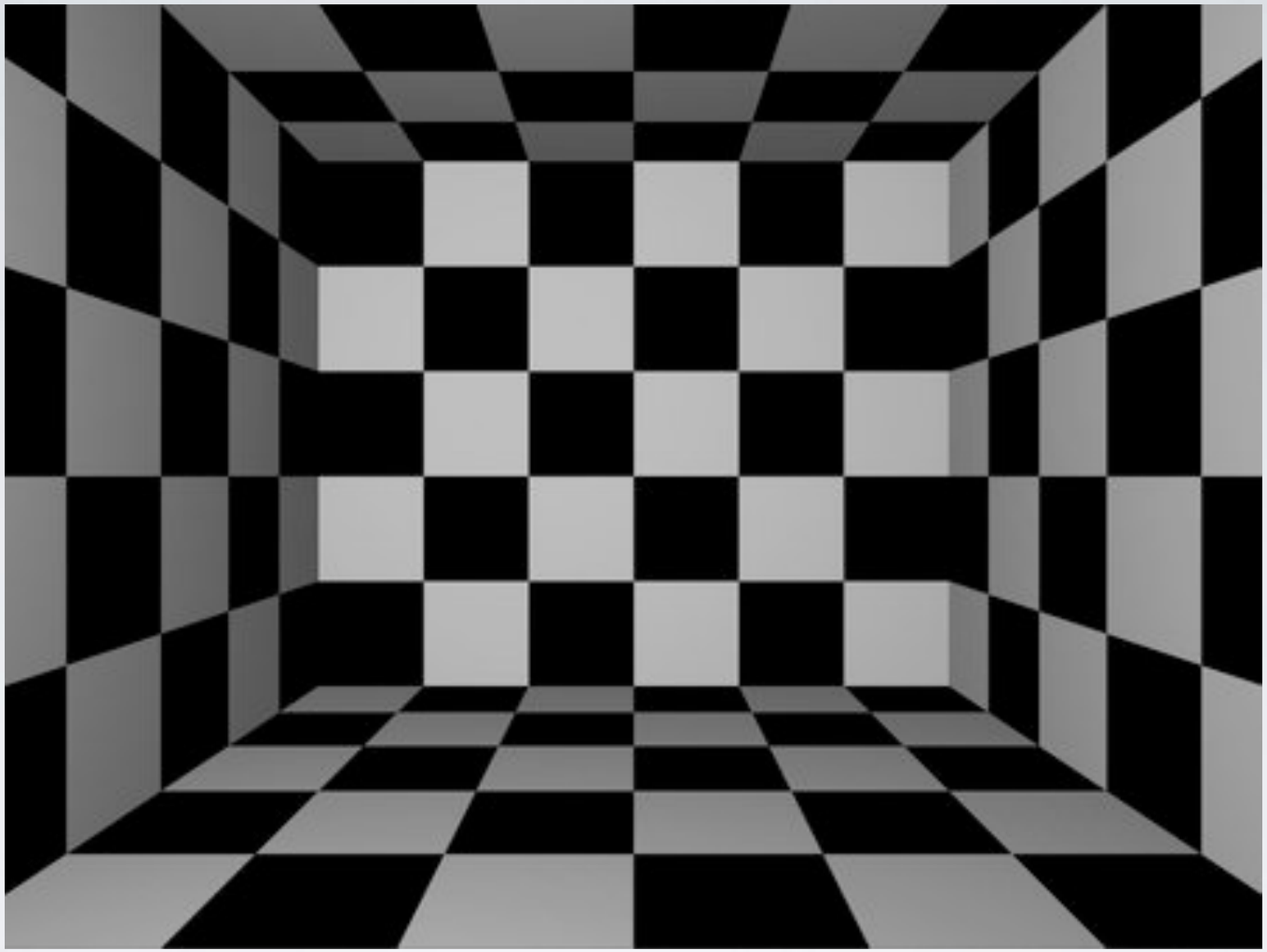
... со сложностью!

ЧТО СЛОЖНЕЕ?

РАБОТА С ЧЕРНЫМИ ЯЩИКАМИ

```
#include "blackbox.h"

int main() {
    blackbox_prepare();
    return blackbox_work_hard();
}
```



ЕСЛИ НЕТ ЧЕРНОГО ЯЩИКА?

Сделать его с помощью других ящиков!

Динамическая память: malloc и free

Низкоуровневые функции работы с сетью (connect, recv, send, close)

Динамический массив

Скачивание файла по адресу

Очередь

Распознавание и чтение форматов (jpg, png, ...)

Скачивалка изображений

Алгоритм resize для изображений

ДЕКОМПОЗИЦИЯ

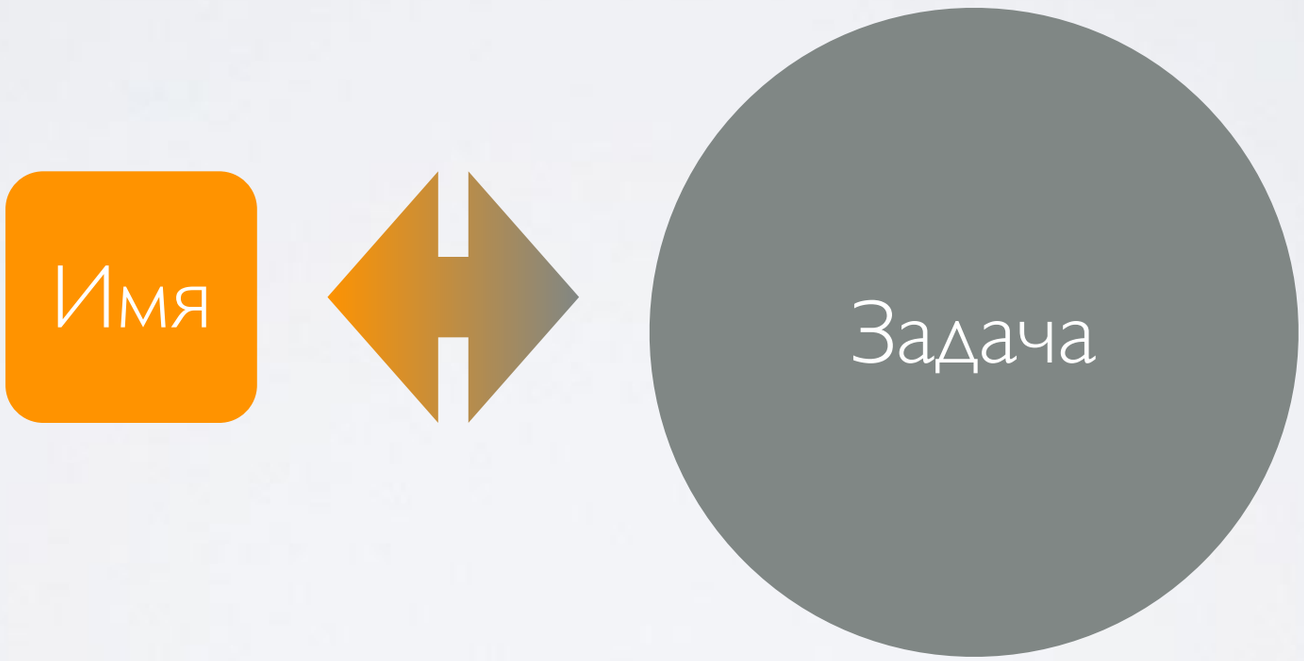
Нечто сложное

Нечто более простое

Нечто более простое

Нечто более простое

ПЕРВОЕ ПРЕДСТАВЛЕНИЕ МОДУЛЯ



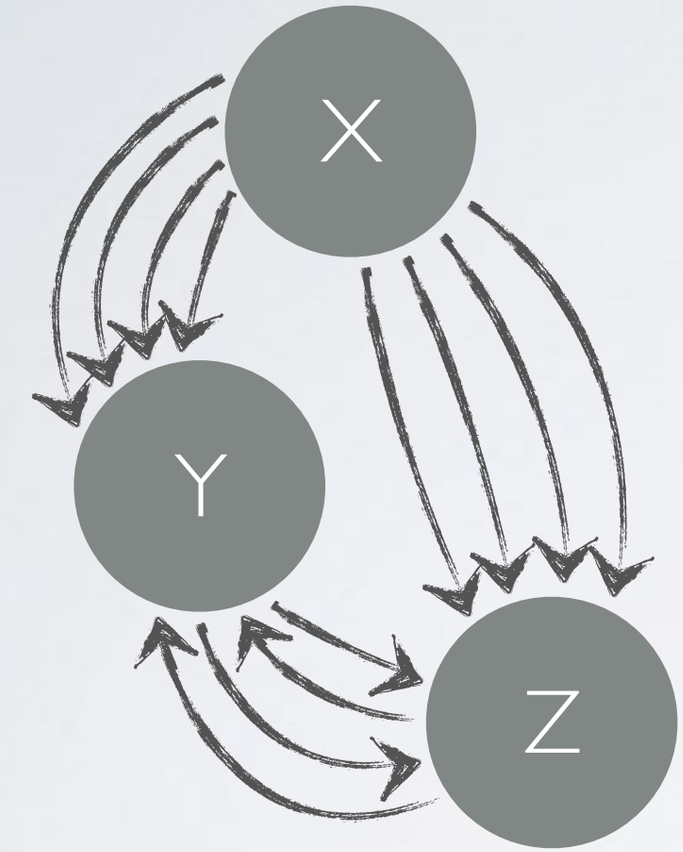
ПРОСТЕЙШИЕ СПОСОБЫ ДЕКОМПОЗИЦИИ

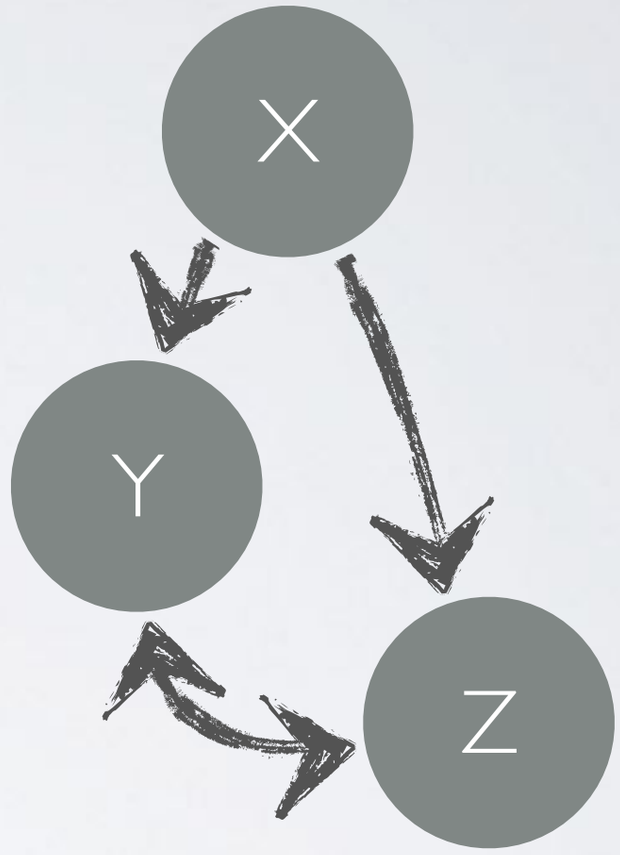
- Взаимодействие с пользователем (**U**ser **I**nterface) – отдельный модуль.
- Обработка разных структур данных – в разных модулях (**list.c**, **stack.c**, **queue.c,** …).
- Элементы функциональности, близкие по смыслу – в один модуль (**io.c**, **parse.c**, **errors.c**).
- Если процесс естественным образом разбивается на отдельные шаги, то каждый шаг – в свой модуль (**prepare.c**, **get_data.c**, **process.c**, **output.c**).

УТОЧНЕНИЕ ПРЕДСТАВЛЕНИЯ

- Модули состоят из функций.
- Функции вызывают другие функции, из своего модуля и из других модулей.
- Вызовы функций – связи между модулями.

КАРТИНКИ СВЯЗЕЙ

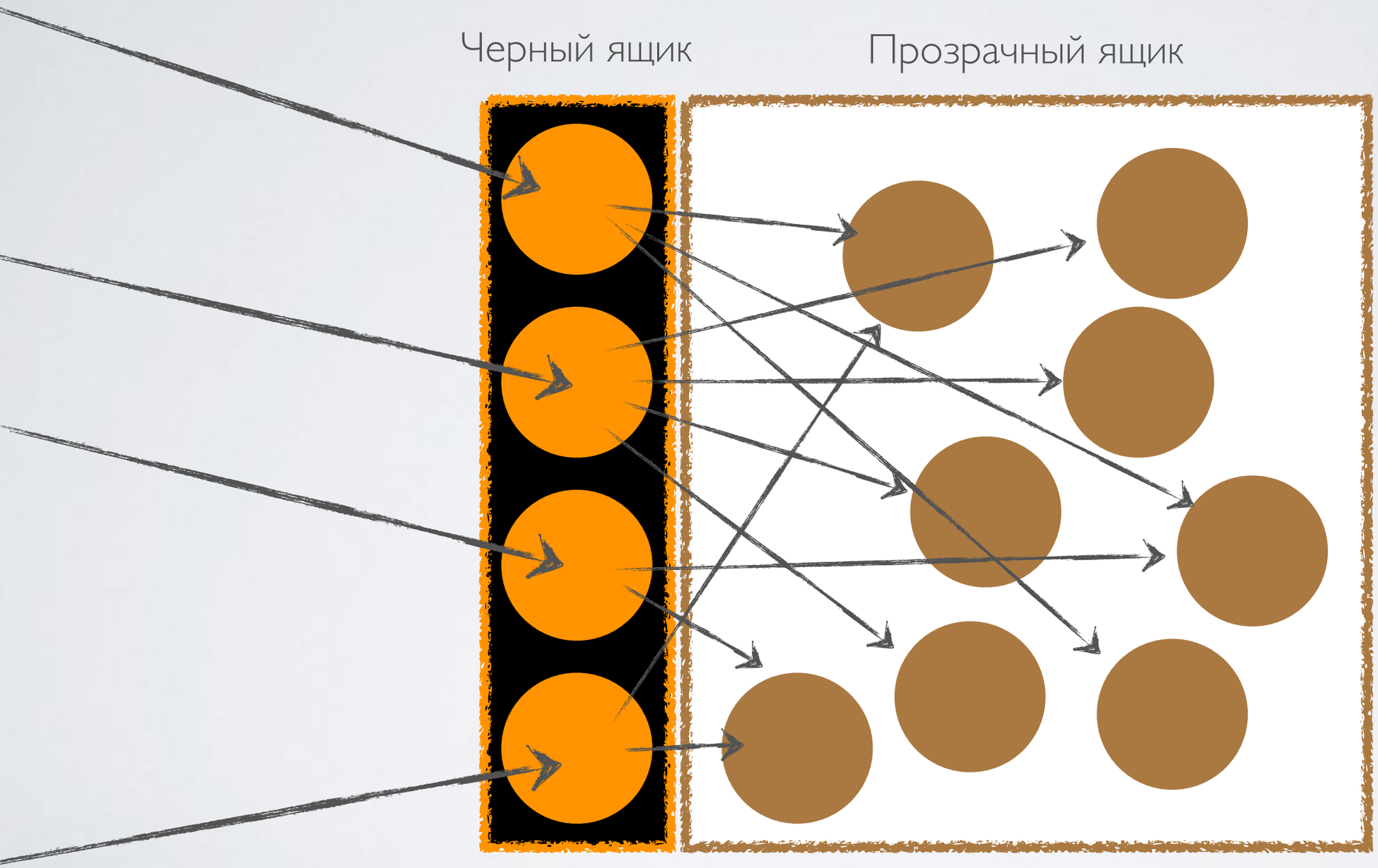




Что проще?

Мера сложности системы – количество связей.

МЕРЫ ПРОТИВ СЛОЖНОСТИ



ИНТЕРФЕЙС И РЕАЛИЗАЦИЯ

- Адресная книга:
  - **Интерфейс**: операции (добавить запись, найти по имени).
  - **Реализация** операций: код функций, основанный на конкретном представлении хранилища и выбранных алгоритмах, а также вспомогательные функции.

УСТРОЙСТВО МОДУЛЕЙ В С

- Каждый модуль состоит как минимум из двух файлов:
  - **module_name.h** – заголовочный файл, содержит определение констант (#define, enum), типов данных (struct, typedef) и прототипы функций интерфейса.
  - **module_name.c** – файл с кодом функций интерфейса и вспомогательных функций.
- Использование модуля:
  - **#include "module_name.h"**

RATIONAL.H

```c
#ifndef RATIONAL_H
#define RATIONAL_H

#include <stdio.h>

typedef struct _Rational {
    int numer;
    int denom;
} Rational;

void rat_create(Rational *res, int a, int b);

void rat_add(Rational *result, Rational *a, Rational *b);
void rat_sub(Rational *result, Rational *a, Rational *b);
void rat_mul(Rational *result, Rational *a, Rational *b);
void rat_div(Rational *result, Rational *a, Rational *b);

void rat_power(Rational *result, Rational *r, int power);

int rat_to_i(Rational *a);
double rat_to_f(Rational *a);

int rat_compare(Rational *a, Rational *b);
void rat_print(Rational *a, FILE *fp);

#endif
```

RATIONAL.C

```c
#include "rational.h"

static void normalize(Rational *rat);

void rat_mul(Rational *result, Rational *a, Rational *b) {
    result->numer = a->numer * b->numer;
    result->denom = a->denom * b->denom;

    normalize(result);
}

/* ... */

static void normalize(Rational *rat) {
    /* ... */
}
```

MAIN.C

```c
#include "rational.h"

int main(void) {
    Rational p, q, r;

    rat_create(&p, 1, 2);
    rat_create(&q, 1, 3);
    rat_add(&r, &p, &q);
    rat_print(&r, stdout);
    return 0;
}
```

СТАНДАРТНАЯ БИБЛИОТЕКА C

- Набор интерфейсов!
  - **stdio.h** – ввод/вывод.
  - **string.h** – работа со строками.
  - **math.h** – математические функции.
  - **time.h** – функции работы со временем и датой.
  - **stdlib.h** – прочие важные функции.

ПОЧЕМУ ИНТЕРФЕЙС ВАЖНЕЕ РЕАЛИЗАЦИИ?

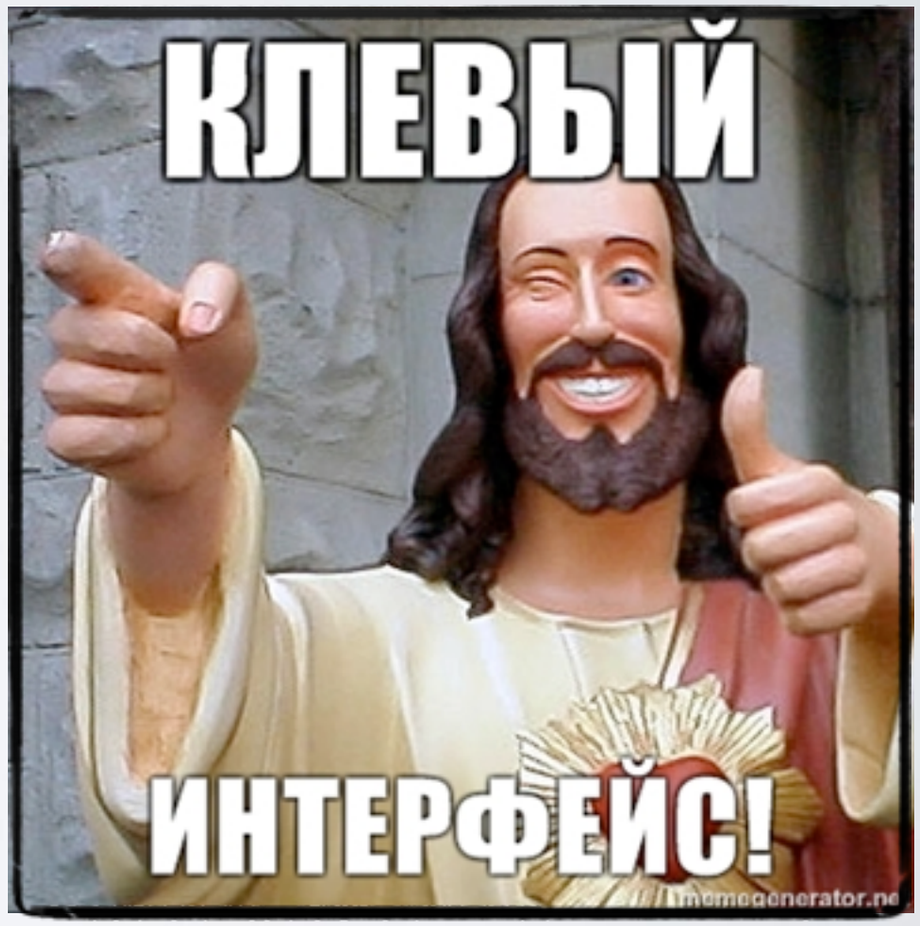
- На один интерфейс может быть несколько реализаций (но не наоборот!).
- Если интерфейс удачный, модуль легче повторно использовать.
- Если интерфейс выпущен в широкую публику, его гораздо сложнее изменить.
- Интерфейс — это то, что увидят другие, реализацию могут и не увидеть.

ОПЕРАЦИОННАЯ СИСТЕМА КАК ИНТЕРФЕЙС

- What would you do if you had to do it over again?
- *Ken Thompson (the creator of UNIX)*: I'd spell creat with e.

ХОРОШИЙ ИНТЕРФЕЙС

- Легко изучить.
- Легко использовать, даже без документации.
- Сложно использовать неправильно.
- Код, написанный с помощью этого интерфейса, легко читать и поддерживать.
- Достаточная сила (количество возможностей).
- Легкая расширяемость.
- Подходит пользователям, которые будут его использовать.

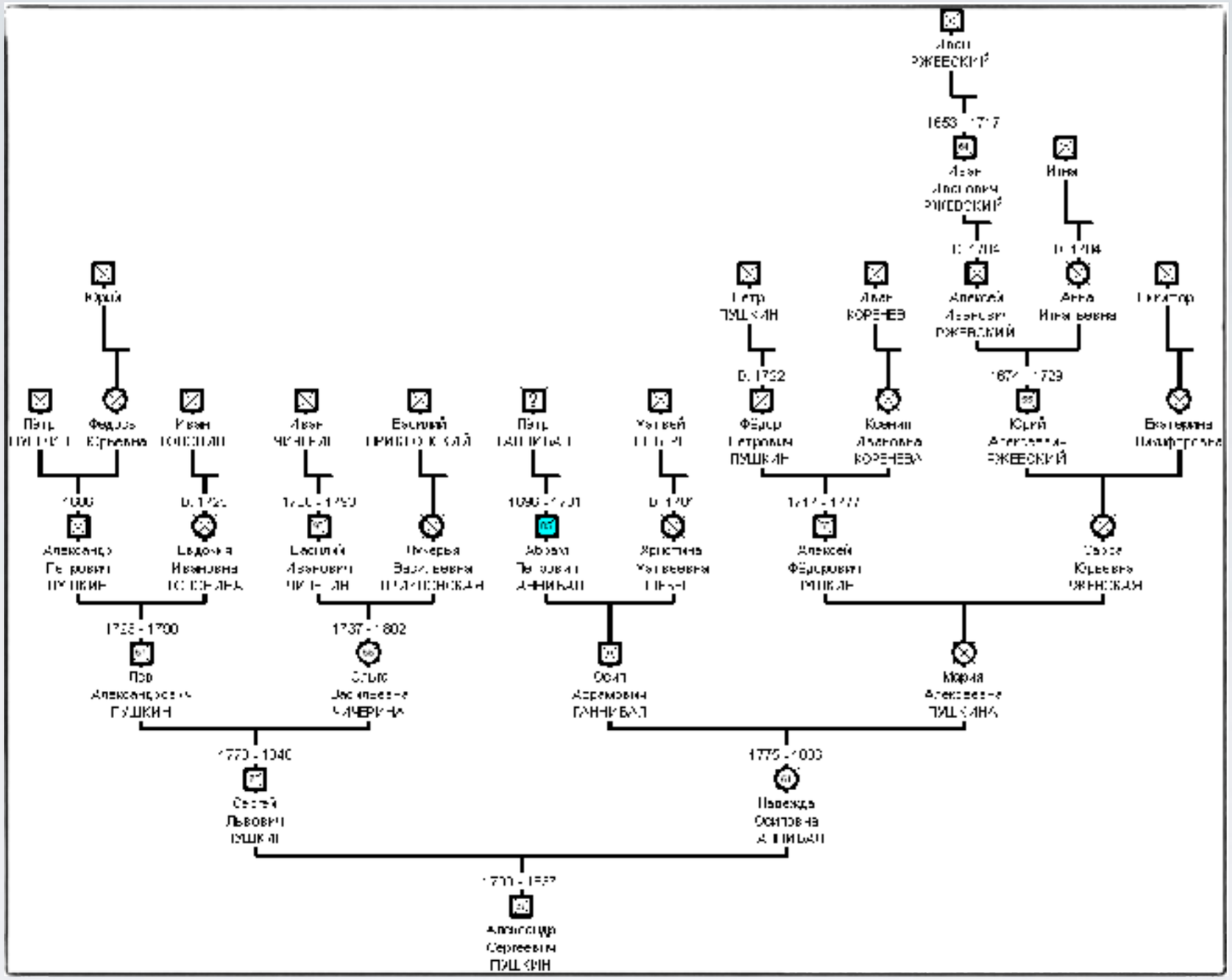


ЗНАКОМЫЕ СТРУКТУРЫ ДАННЫХ

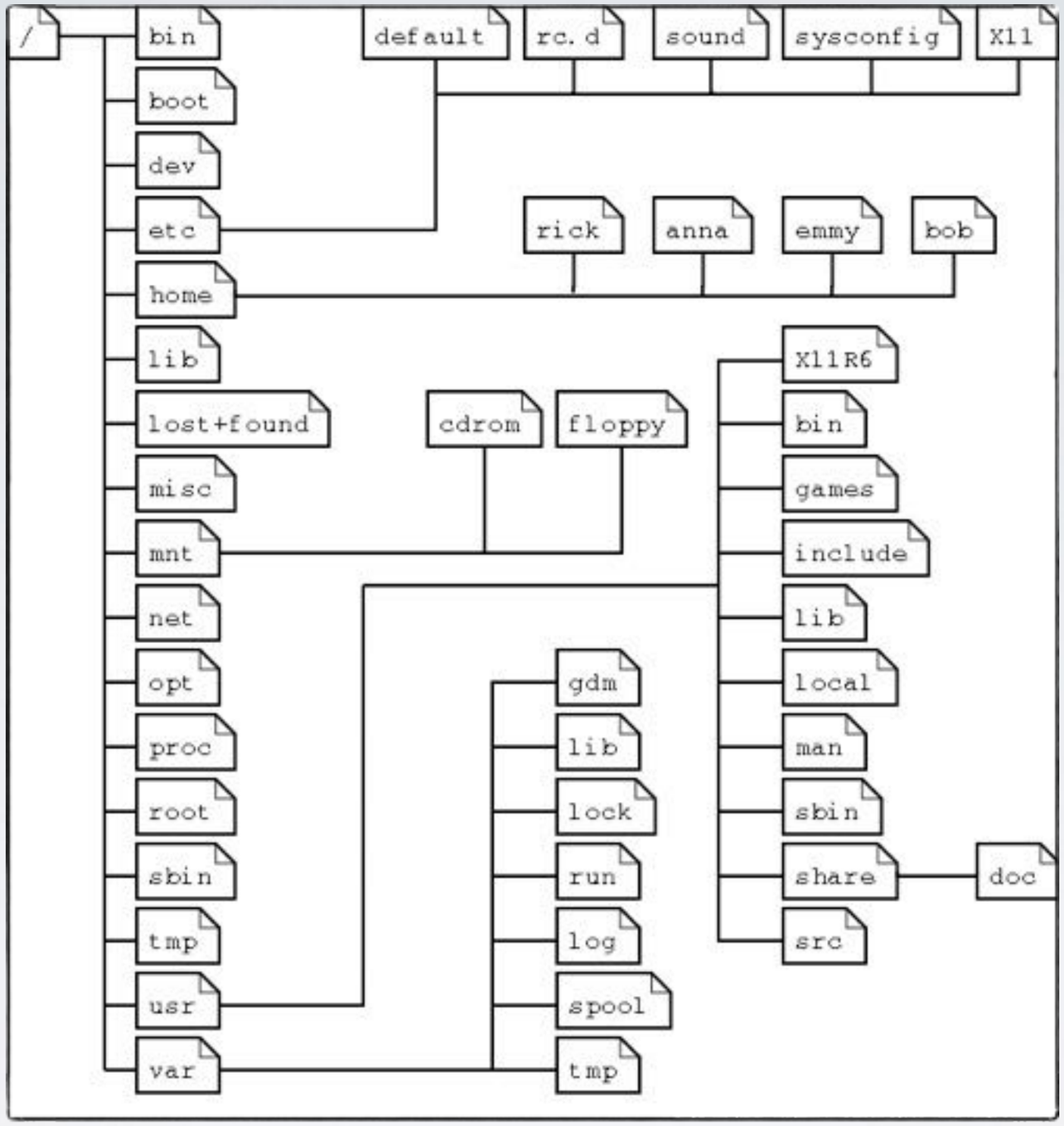
Операции в **массивах** и **связных списках**:

- Вставка: O(N) и O(1).
- Удаление: O(N) и O(1).
- Доступ по индексу: O(1) и O(N).
- Поиск по значению: O(N) и O(N).

ДАЛЕЕ: НАЙТИ ОБЩЕЕ



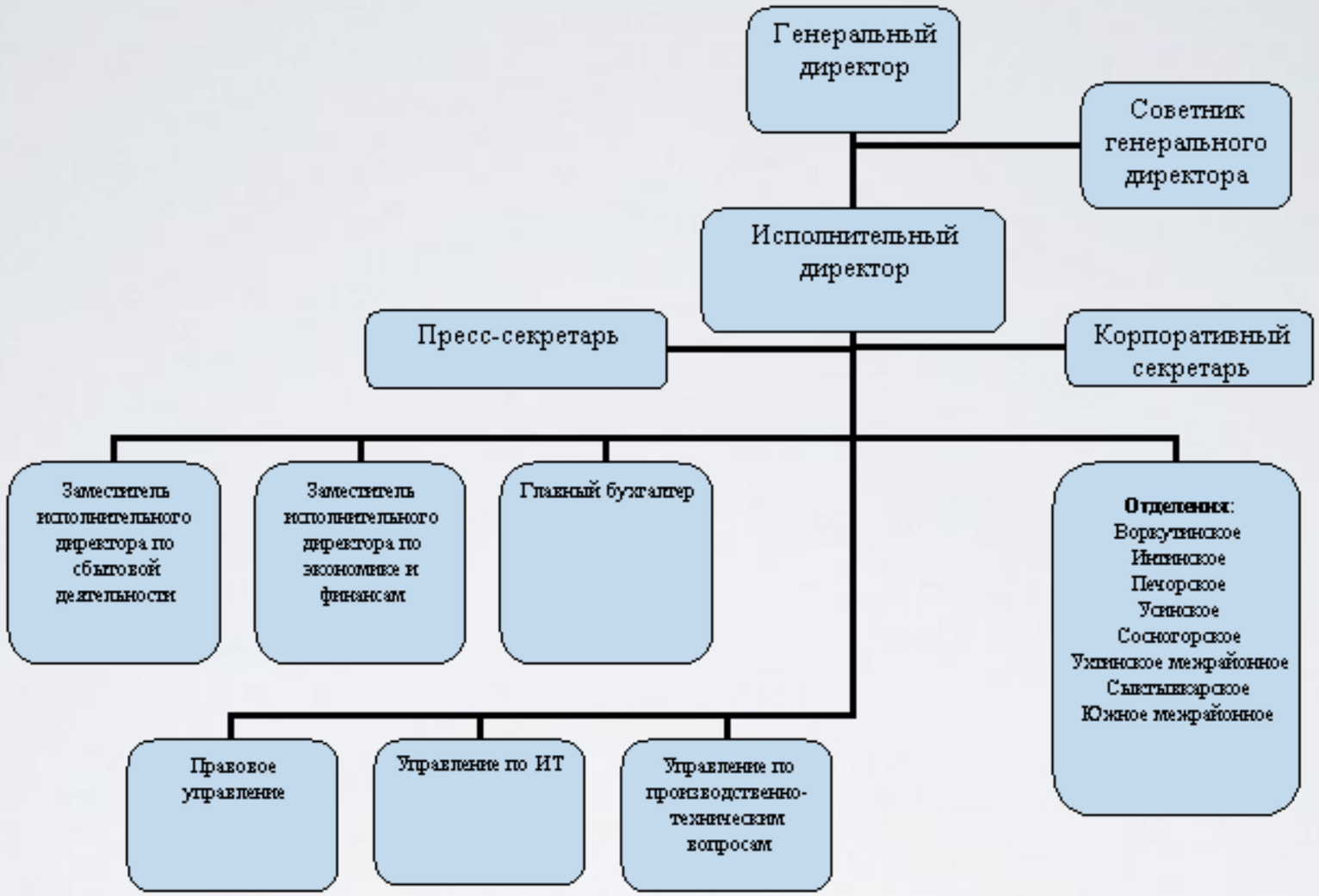
Генеалогическое дерево



Файлы и каталоги

```
<client>
  <address>
    <street>5401 Julio Ave.</street>
    <city>San Jose</city>
    <state>CA</state>
    <zip>95116</zip>
  </address>
  <phone>
    <work>4084630000</work>
    <home>4081111111</home>
    <cell>4082222222</cell>
  </phone>
  <fax>4087776666</fax>
  <email>love2shop@yahoo.com</email>
</client>
```

XML-документ



Организационная структура предприятия

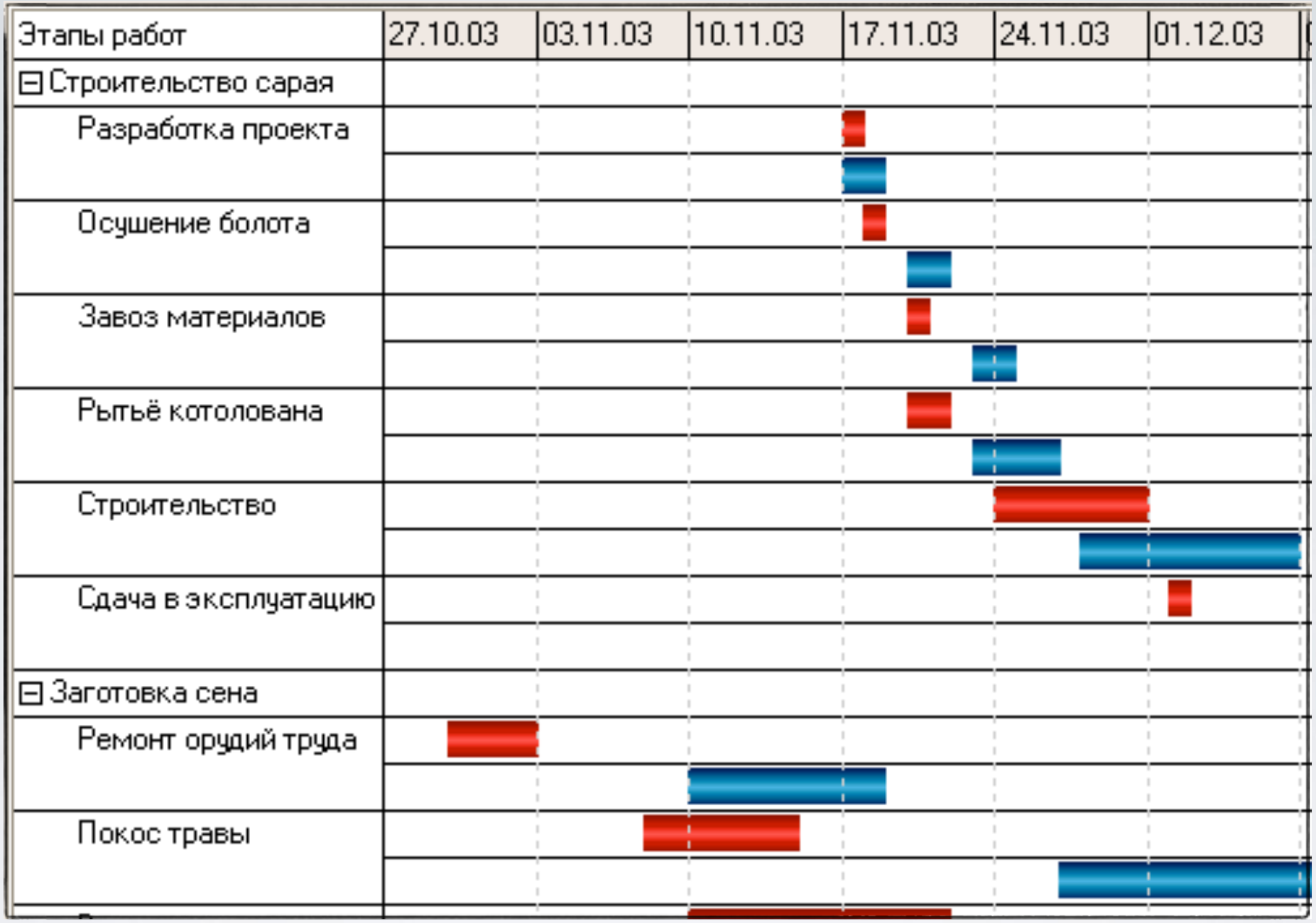
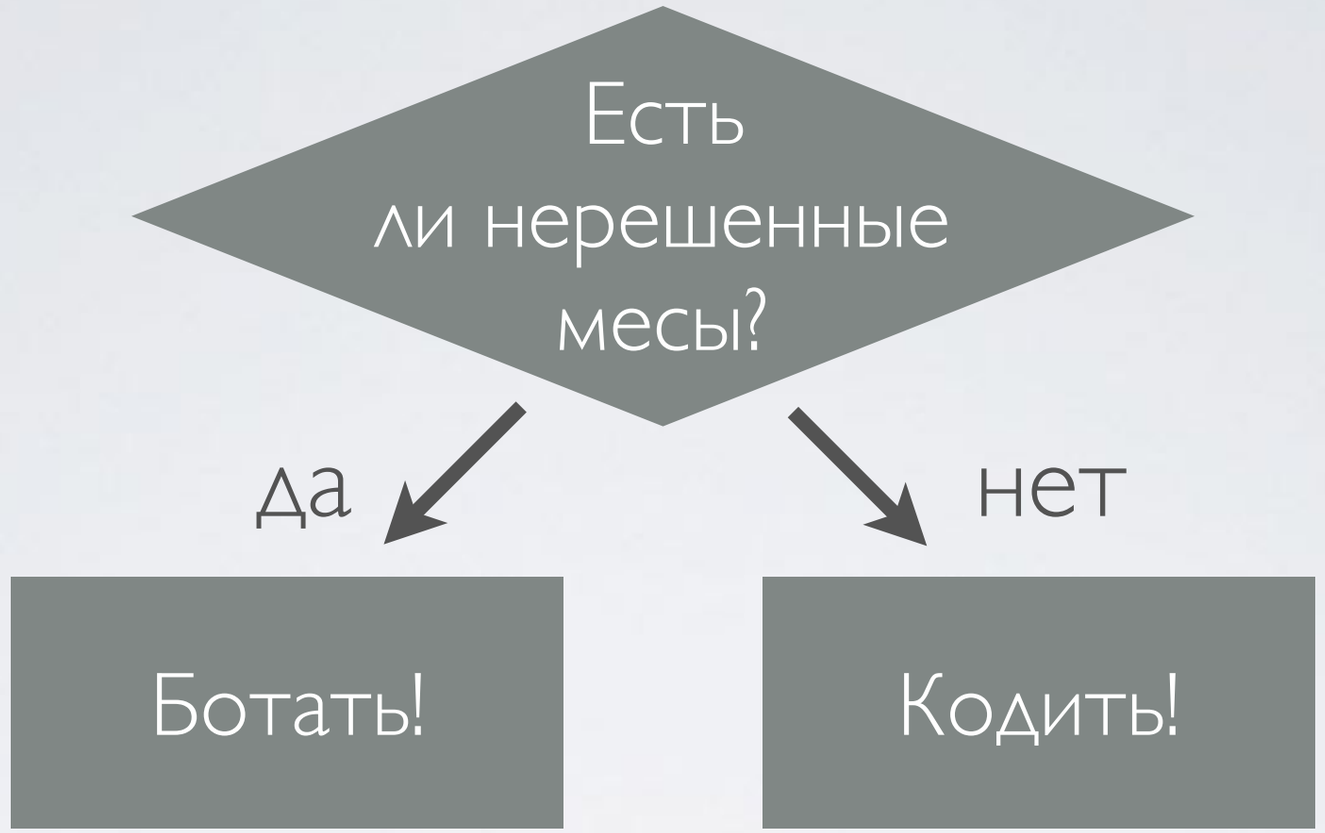


Диаграмма Ганта



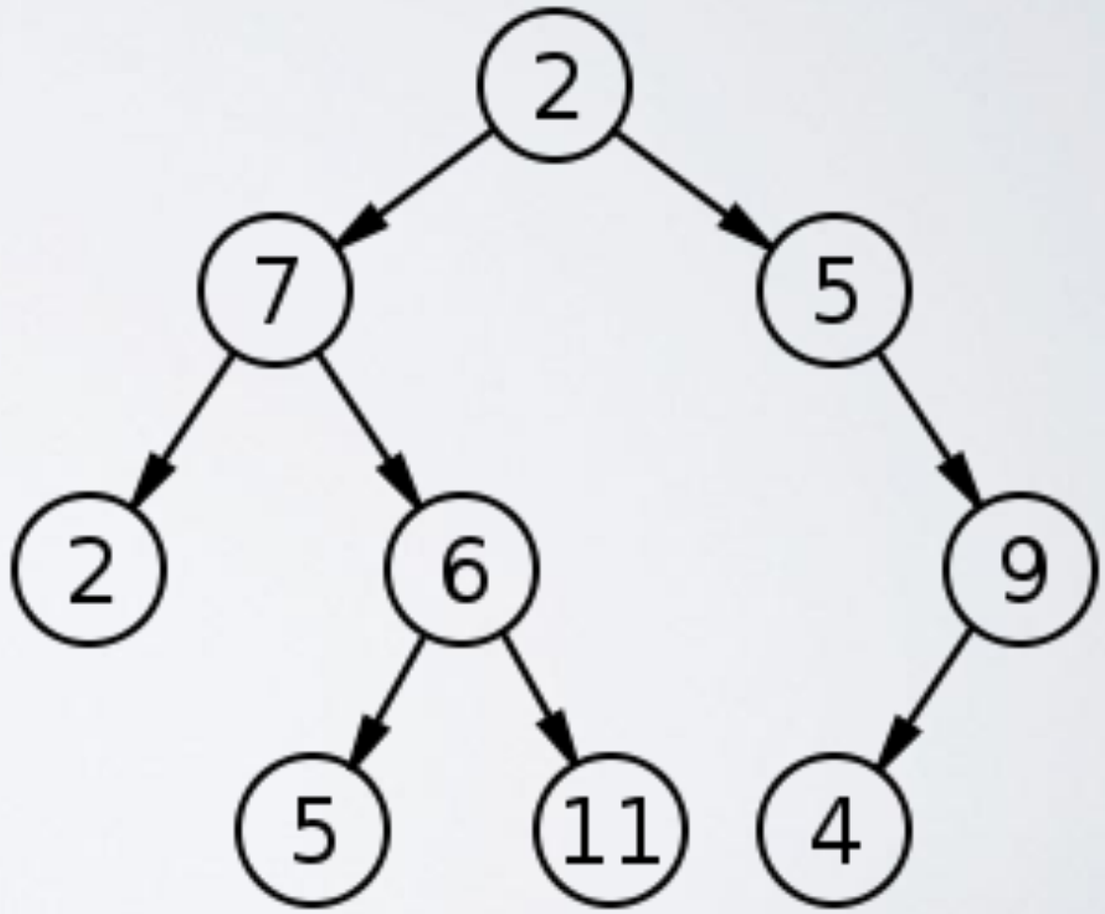
Дерево решений

СТРУКТУРА ДАННЫХ: ДЕРЕВО

- Состоит из элементов (узлов).
- Имеет корень.
- Все остальные узлы, кроме корня, распределены по непересекающимся подмножествам — поддеревьям.

ДРЕВОВЕДЕНИЕ

- **Корень** (2).
- **Внутренние узлы** (2, 7, 5, 6, 9) и **листья** (2, 5, 11, 4).
- **Родитель** (7 для 2 и 6; 9 для 4; 2 для 5 и 7) и **потомки** (дочерние узлы).
- **Сестринские узлы** (5 для 7; 2 для 6; 11 для 5).



ИНТЕРФЕЙС ДЕРЕВА

- Вставка узла.
- Удаление узла.
- Обход дерева (посещение всех узлов).
- Переходы (от потомка к родителю, от сестринского узла к другому сестринскому и т.д.)

ДЕРЕВЬЯ В С

```
struct TreeNode {
    struct TreeNode *parent;
    struct TreeNode **children;
    int nchildren;
    void *data;
};
```

Динамический массив
указателей на дочерние узлы

БИНАРНОЕ ДЕРЕВО

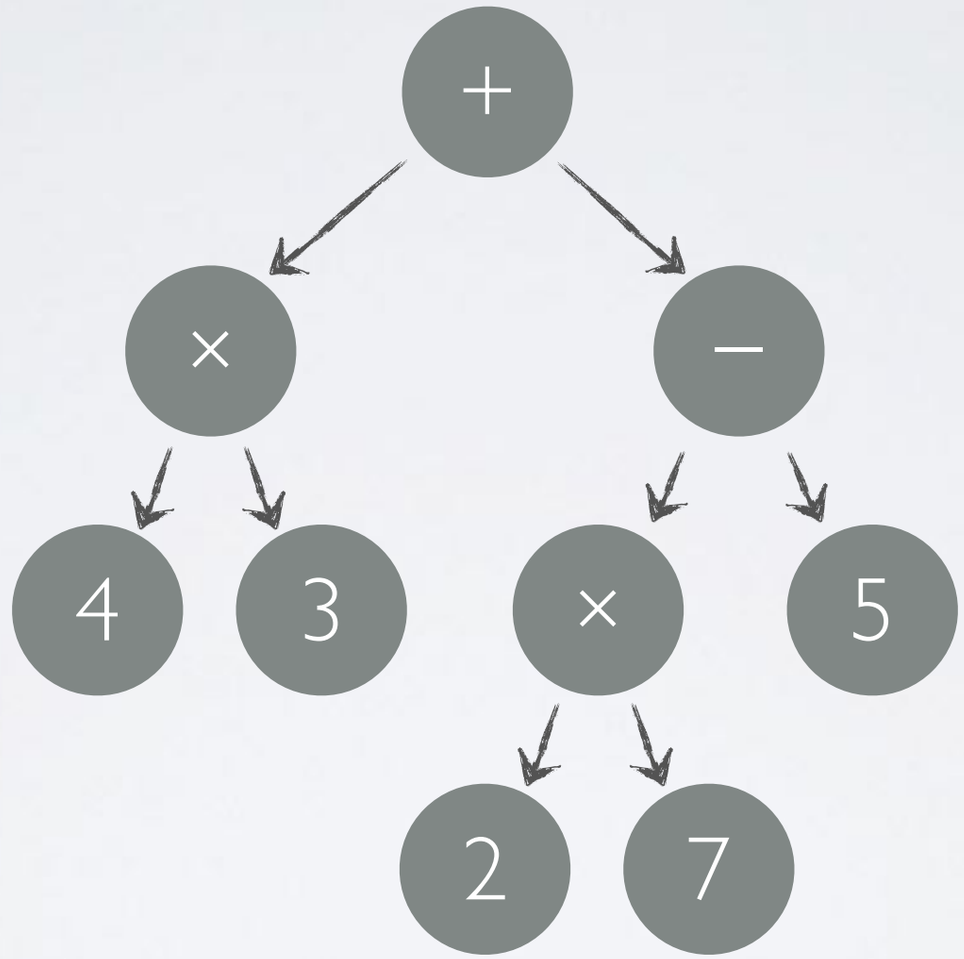
- У каждого узла максимум два потомка: **левый** и **правый**.
- Может быть так, что правый потомок присутствует, а левый — нет.
- Допустимо пустое двоичное дерево.

ДВОИЧНЫЕ ДЕРЕВЬЯ В C

```
struct TreeNode {
    struct TreeNode *parent;
    struct TreeNode *left, *right;
    void *data;
};
```

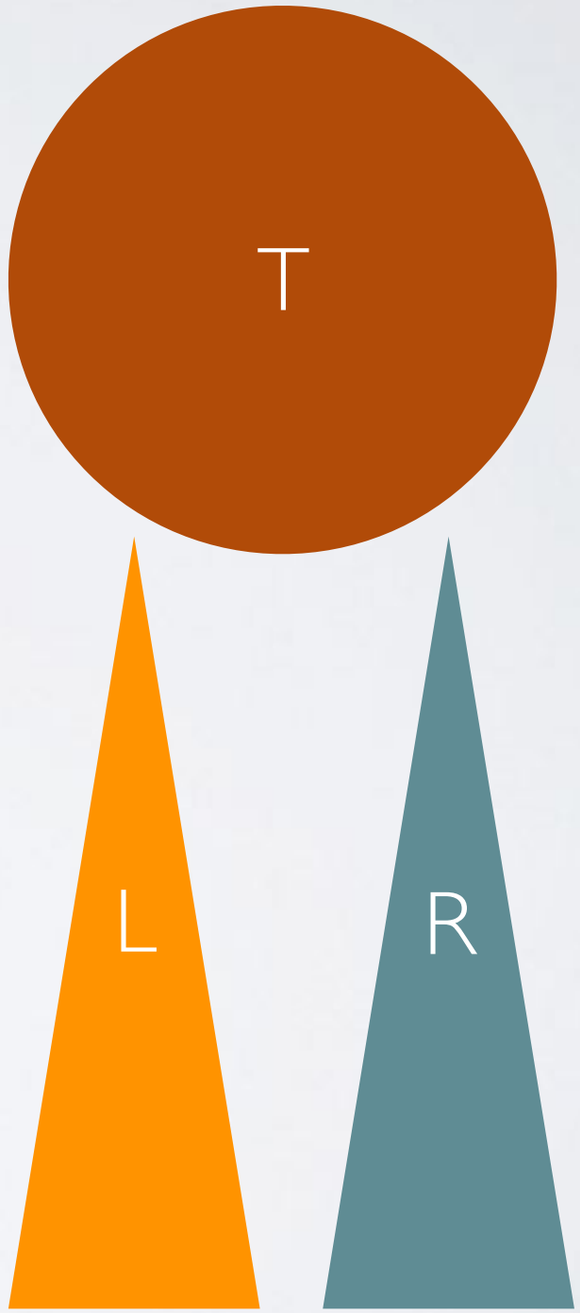
Популярность двоичных деревьев связана с удобством представления и работы с ними

АРИФМЕТИЧЕСКОЕ ВЫРАЖЕНИЕ



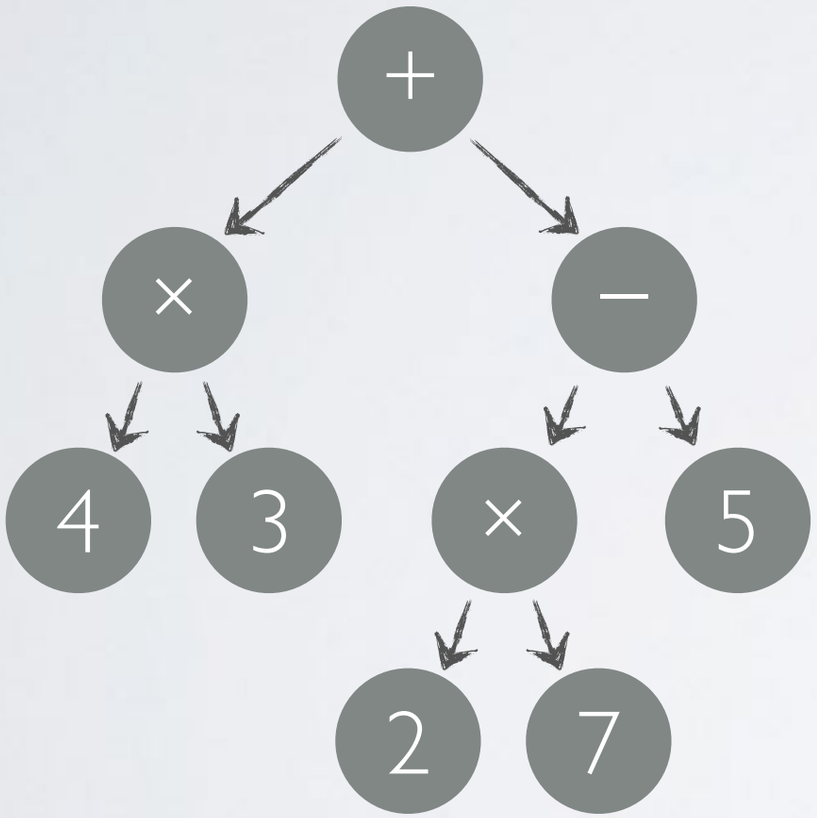
$$4 \times 3 + (2 \times 7 - 5)$$

ОБХОДЫ ДВОИЧНОГО ДЕРЕВА



- Сверху вниз: **T**, **L**, **R**.
- Слева направо: **L**, **T**, **R**.
- Снизу вверх: **L**, **R**, **T**.

ОБХОДЫ ДЕРЕВА ВЫРАЖЕНИЯ



Сверху вниз: префиксная запись

+ × 4 3 − × 2 7 5

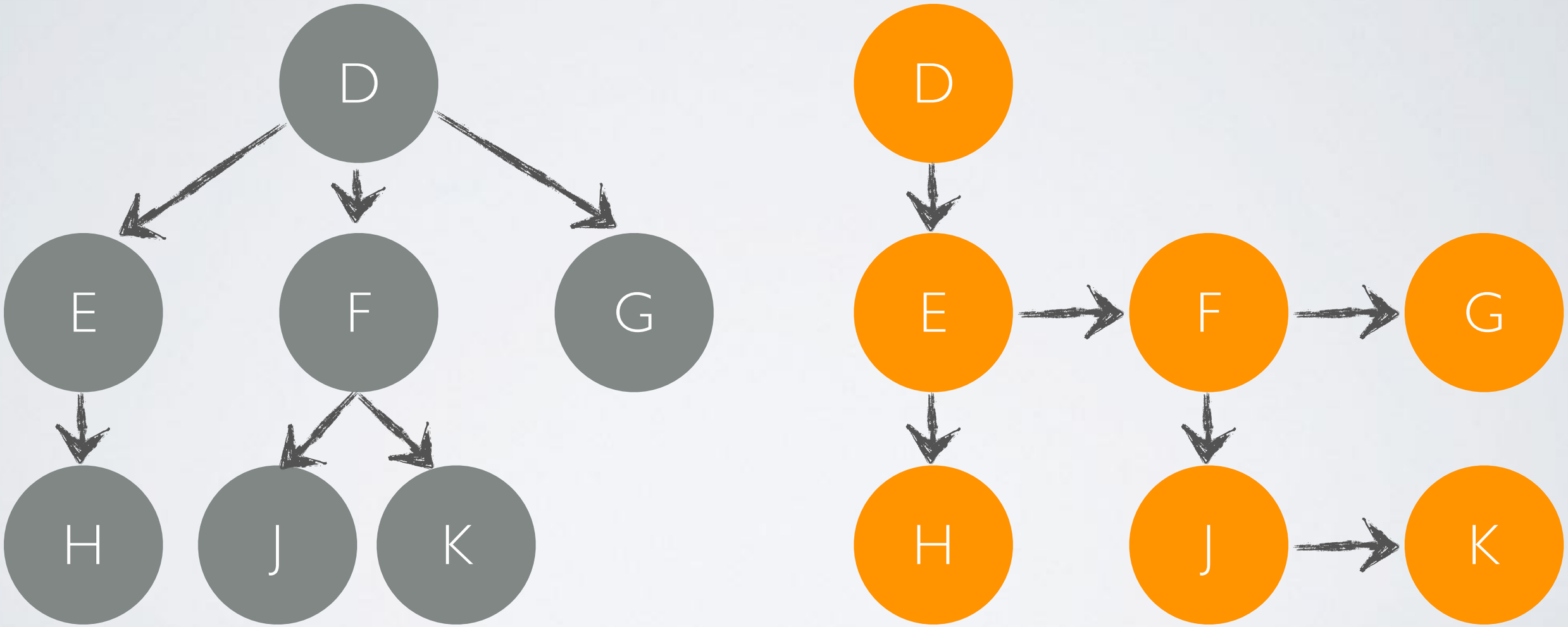
Слева направо: инфиксная запись

4 × 3 + 2 × 7 − 5

Снизу вверх: постфиксная запись

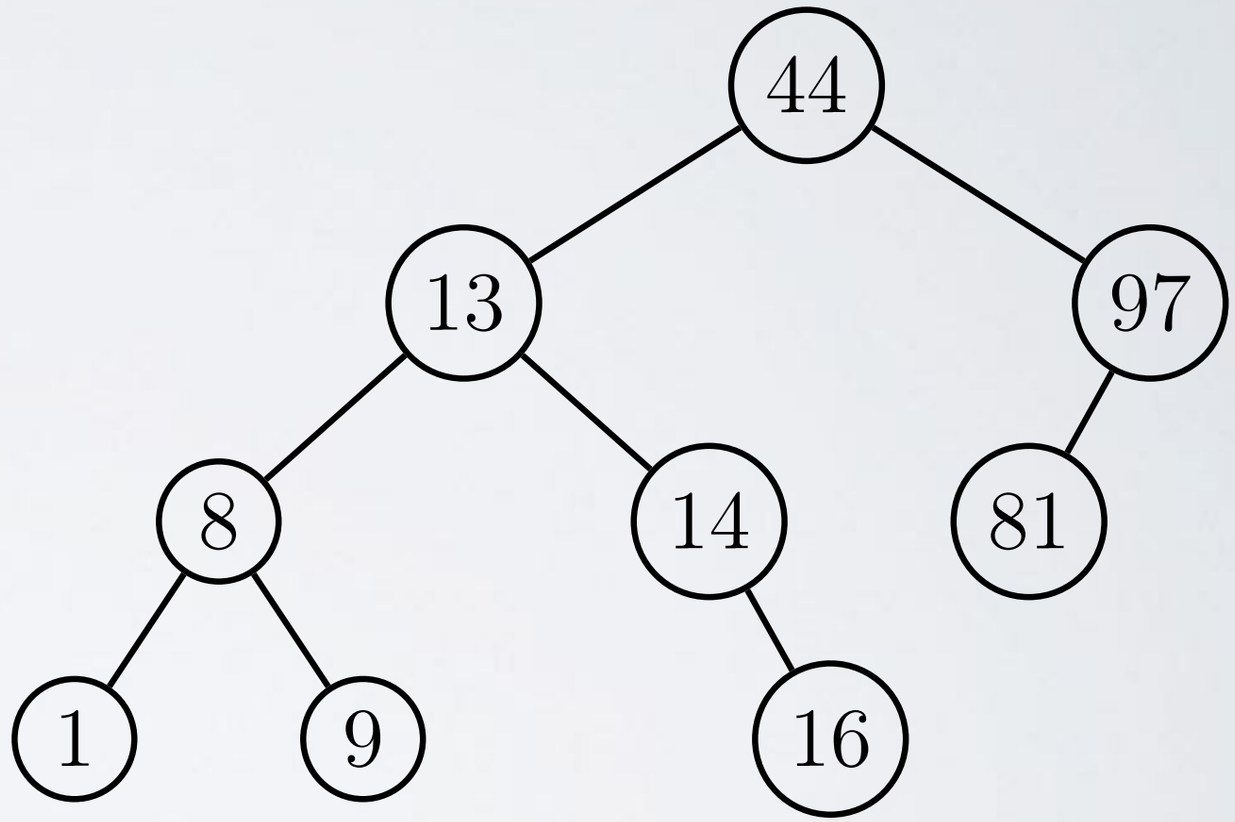
4 3 × 2 7 × 5 − +

ПРЕОБРАЗОВАНИЕ ЛЮБОГО ДЕРЕВА В ДВОИЧНОЕ



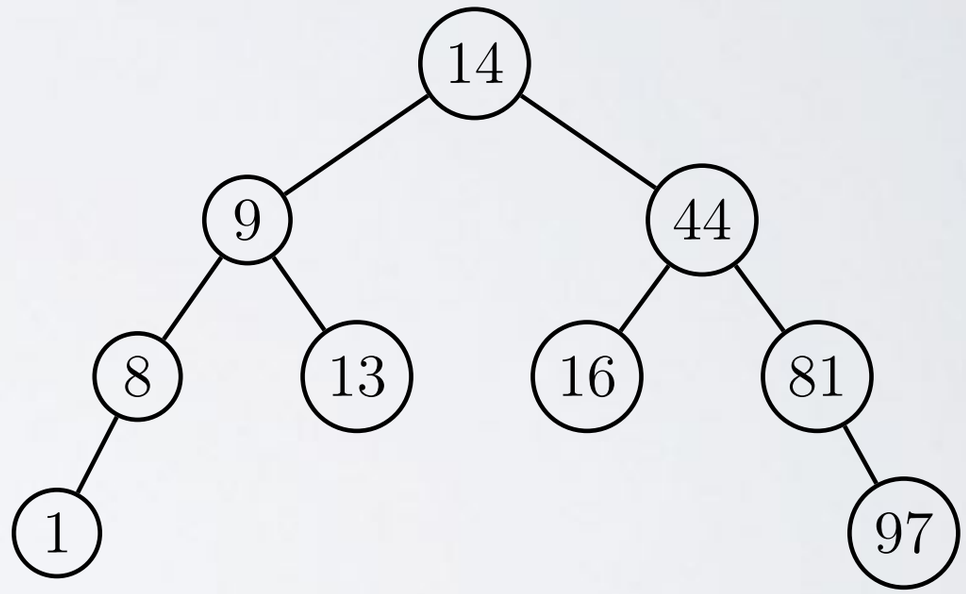
ДЕРЕВО ПОИСКА

- BST (Binary Search Tree).
- Каждому узлу **n** сопоставлен ключ **k(n)**.
- **k(x) < k(n)** для **x ∈ L(n)** — левое поддерево.
- **k(y) > k(n)** для **y ∈ R(n)** — правое поддерево.



ИНТЕРФЕЙС ДЕРЕВА ПОИСКА

- Поиск элемента по ключу
- Вставка элемента по ключу
- Удаление элемента по ключу
- Перечисление всех ключей

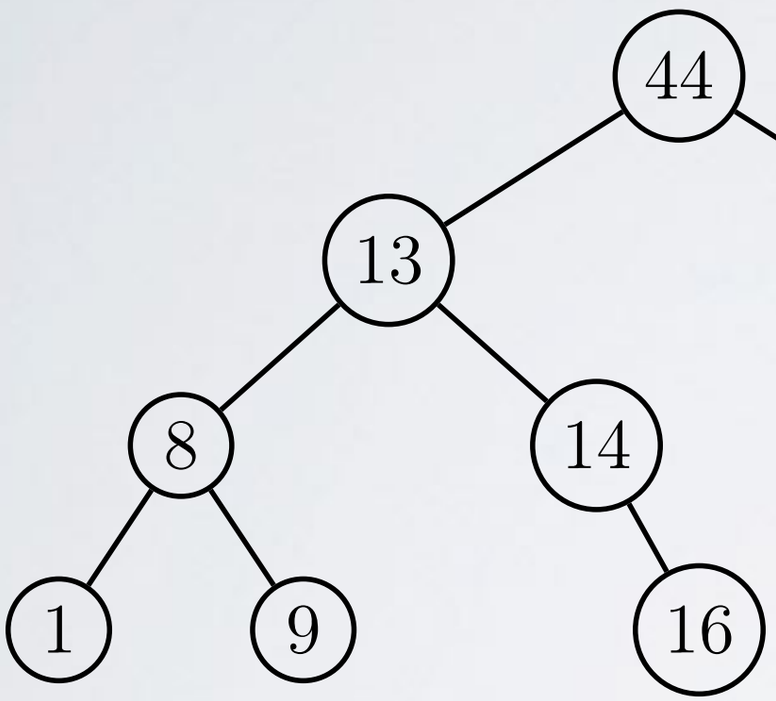


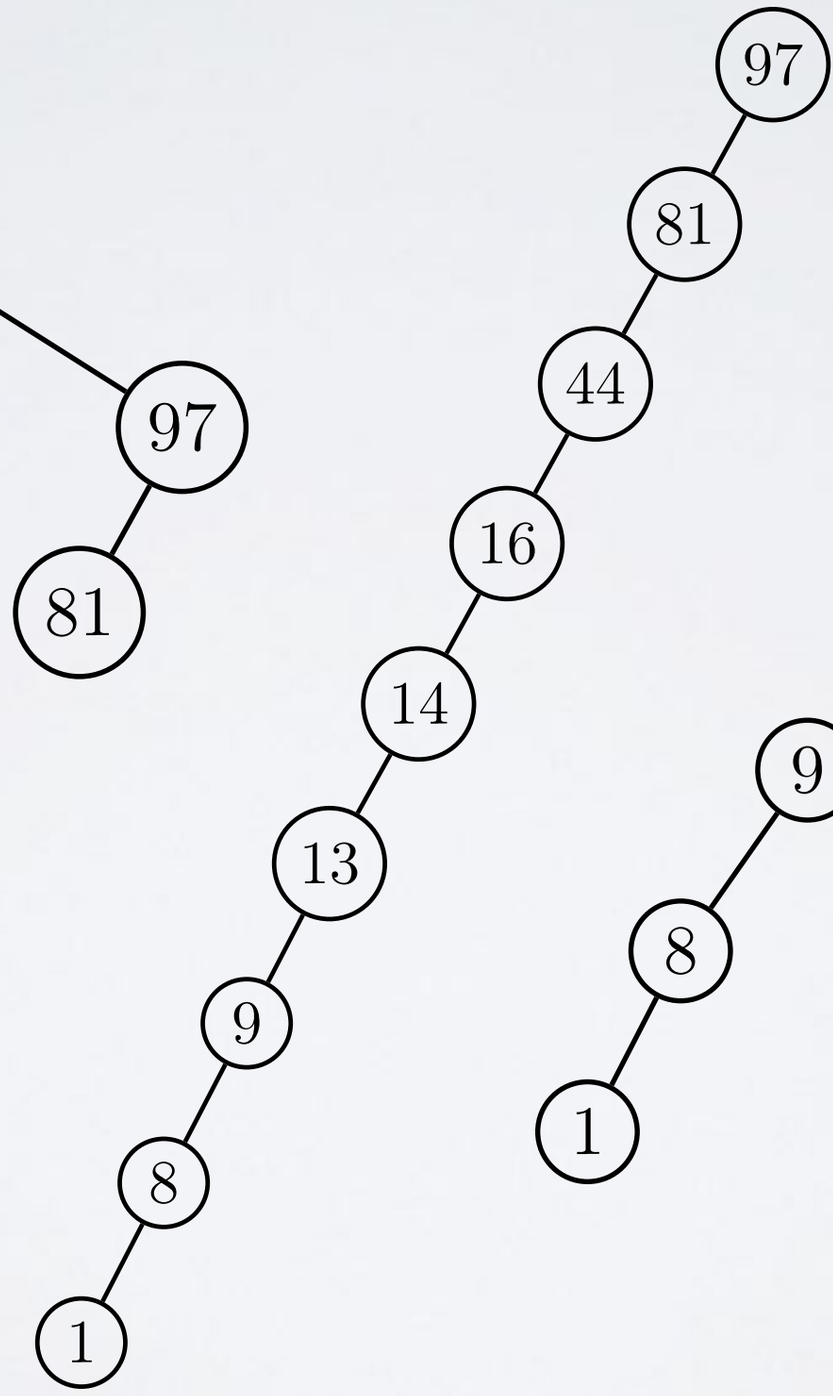
Поиск и вставка за O(h(N))!

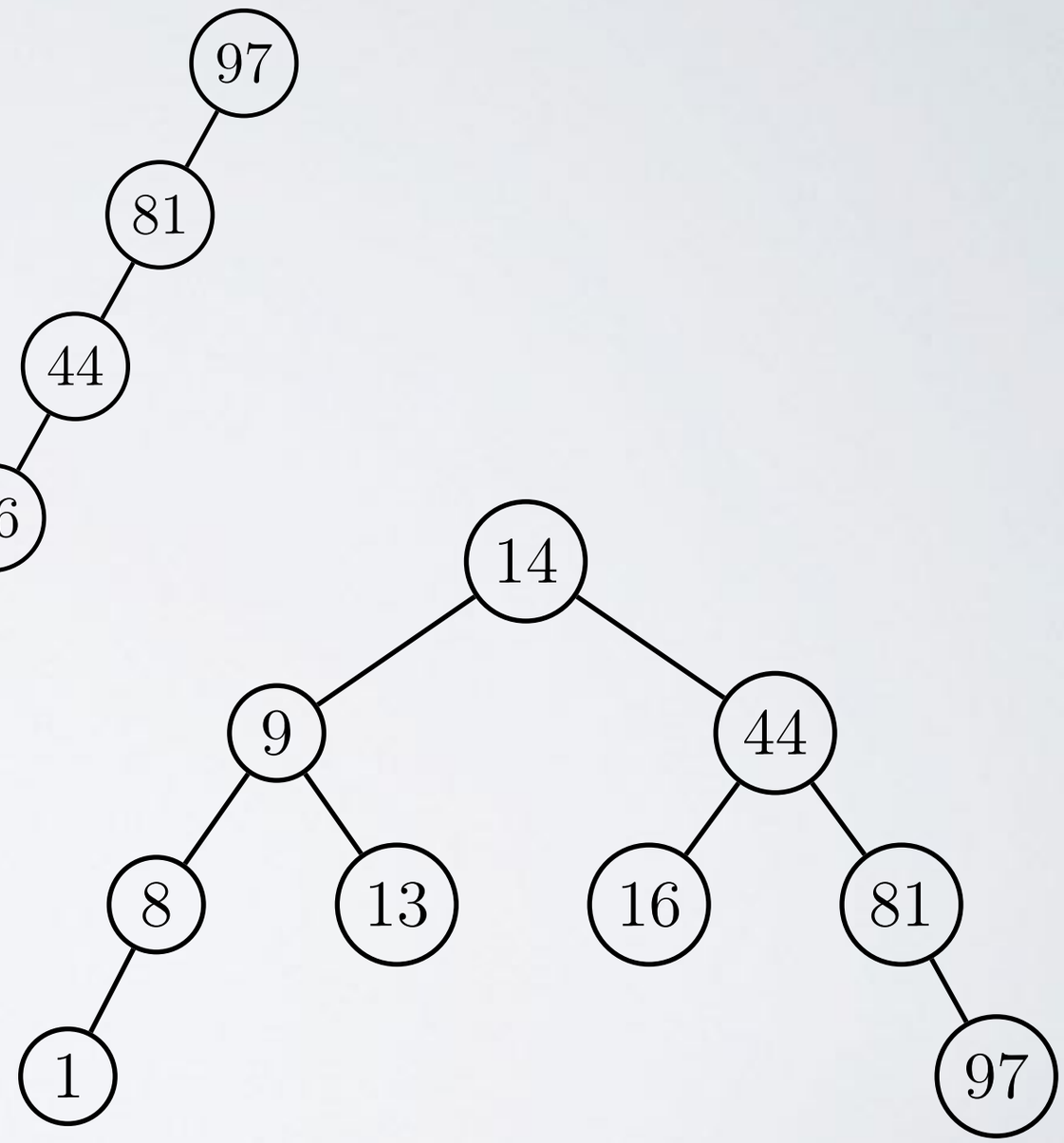
ВЫСОТА ДЕРЕВА ПОИСКА

- Бинарное дерево высоты h содержит максимум $2^h - 1$ узлов.
- Значит высота $h(N) \geq \log(N)$.
- При добавлении случайных элементов $h(N) \sim 2{,}99 \log(N)$. Средняя глубина узла $\sim 1{,}39 \log(N)$.
- Но в худшем случае…

НЕ ВСЕ ДЕРЕВЬЯ ОДИНАКОВО ПОЛЕЗНЫ





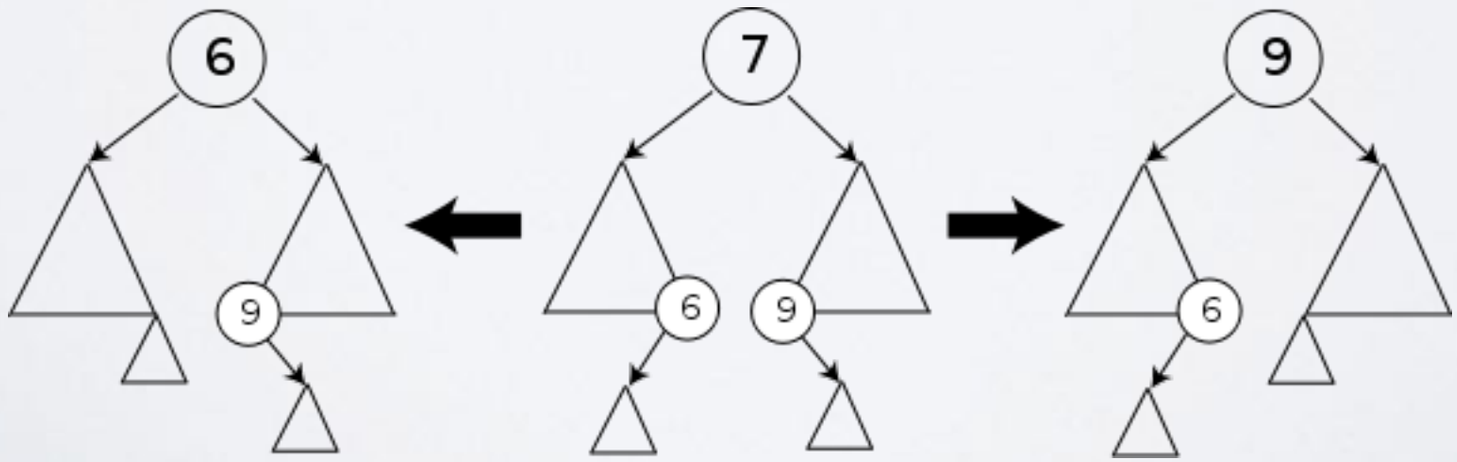


ПРИМЕНЕНИЯ ДЕРЕВА ПОИСКА

- АТД Множество (set)
- АТД Мультимножество (multiset)
- АТД Ассоциативный массив
  (отображение, map, словарь, dictionary)

УДАЛЕНИЕ ЭЛЕМЕНТА

- Если лист (нет потомков), то просто удаляем
- Если потомок один, он заменит удаляемый узел
- Если два потомка, то нужно найти либо самый **правый** узел **левого** поддерева, либо самый **левый** узел **правого** поддерева и поставить на место удаляемого узла

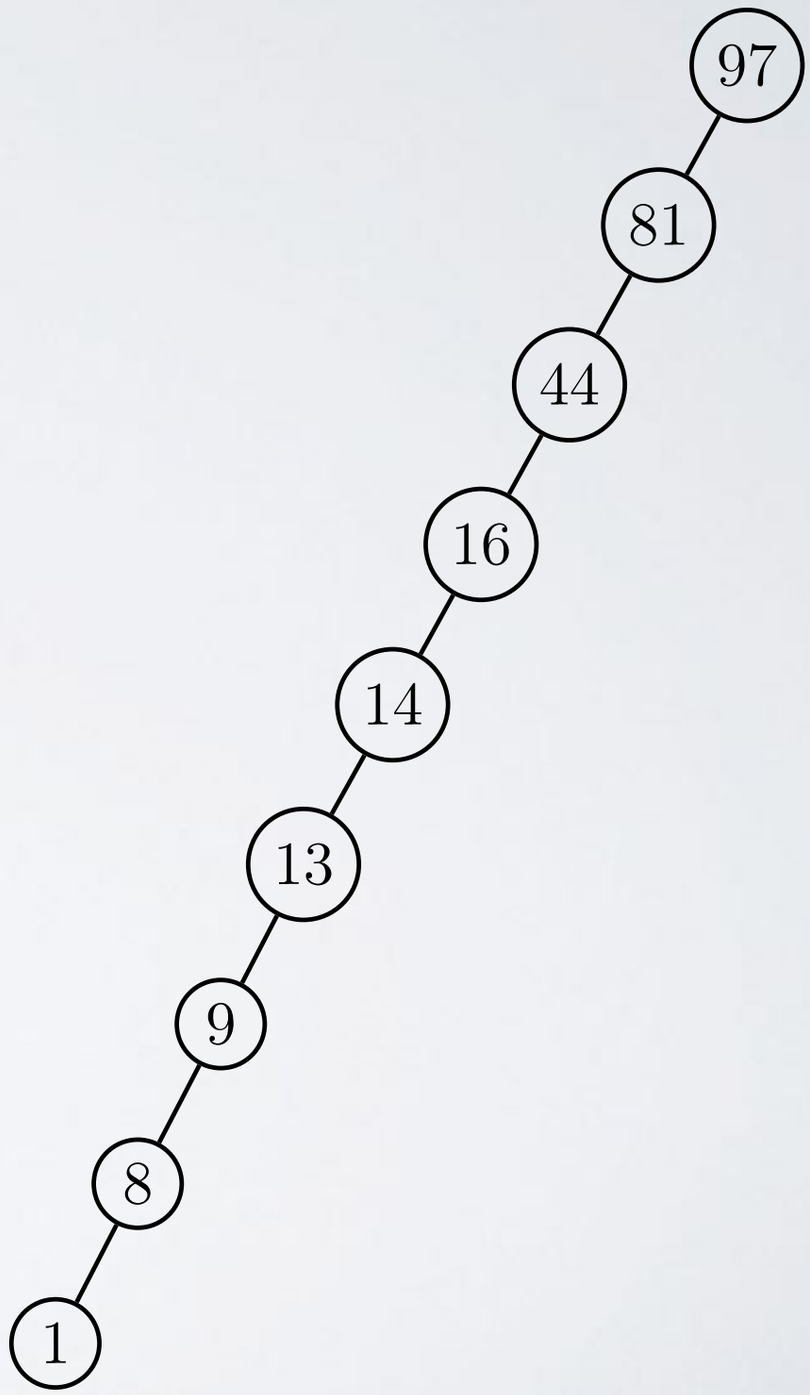


УДАЛЕНИЕ ЭЛЕМЕНТА

- Представленный алгоритм удаления приводит к тому, что высота дерева растет и становится ~ $N^{1/2}$.
- Даже если случайным образом выбирать, с какой стороны брать новый элемент.
- Есть ли способы гарантированно выполнять поиск и вставку за O(log(N))?

РЕШЕНИЯ ПРОБЛЕМЫ «КРИВЫХ» ДЕРЕВЬЕВ

- Восстановление оптимальности:
  - «Выворачивание» *(splay trees)*,
  - АВЛ-деревья,
  - Красно-черные деревья.



everything will be okay

in the end.

if it's not okay,

it's not the end.

(unknown)

КОНЕЦ ВОСЬМОЙ ЛЕКЦИИ

Каждый программист в своей жизни должен …, … и реализовать модуль работы с деревом