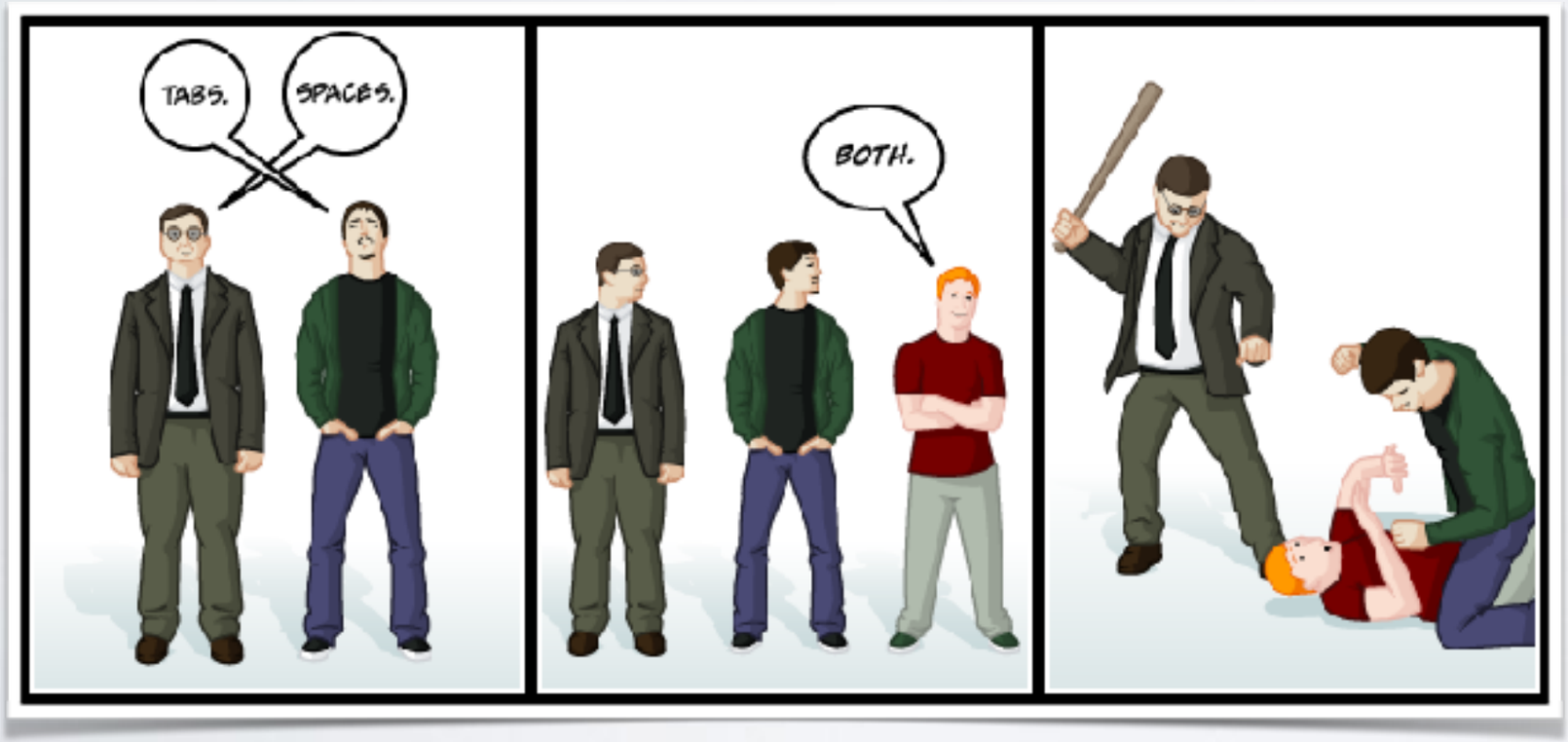
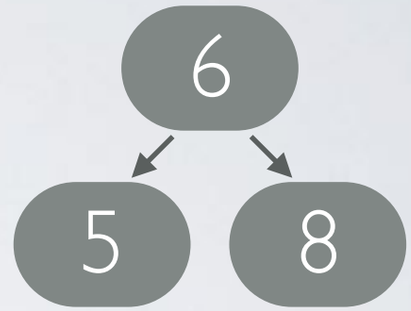
ОСНОВЫ ПРОГРАММНОГО КОНСТРУИРОВАНИЯ

Лекция № 11

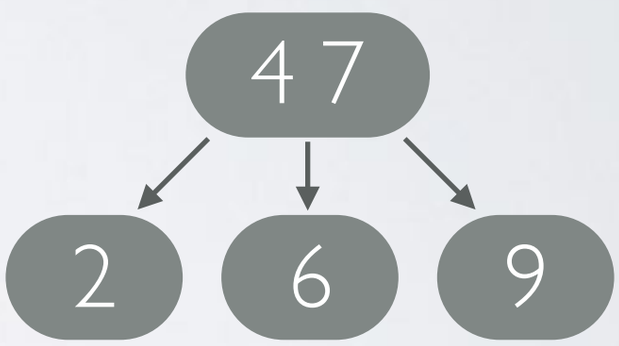
13 ноября 2017 г.

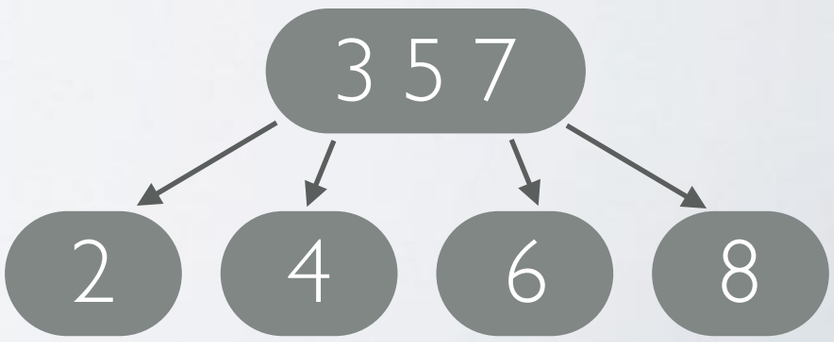


2-3-4 ДЕРЕВЬЯ



- Дерево поиска, узлы которого:
  - либо пусты;
  - либо 2-узел: 1 значение, 2 поддерева;
  - либо 3-узел: 2 значения, 3 поддерева;
  - либо 4-узел: 3 значения, 4 поддерева.
- Всегда идеально сбалансировано: высоты всех поддеревьев равны.

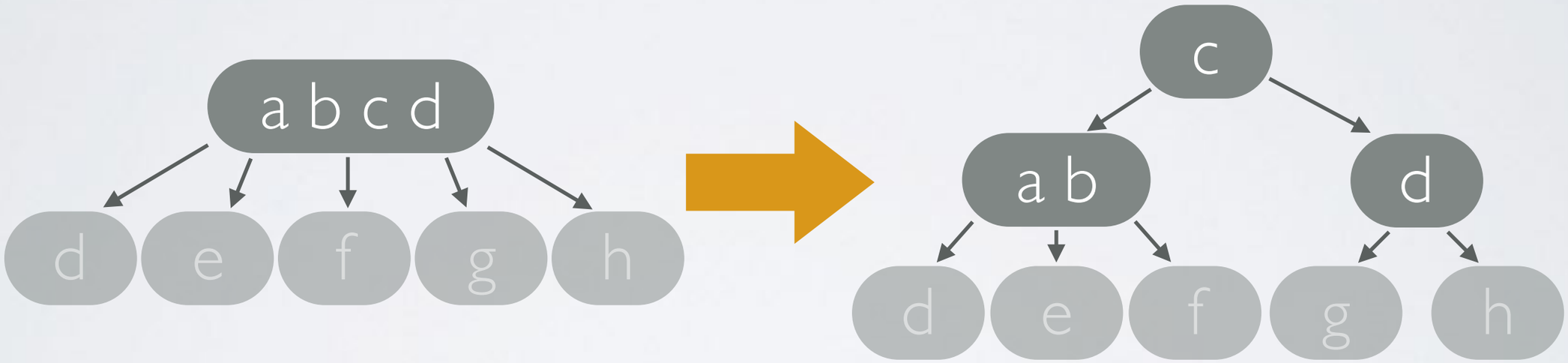




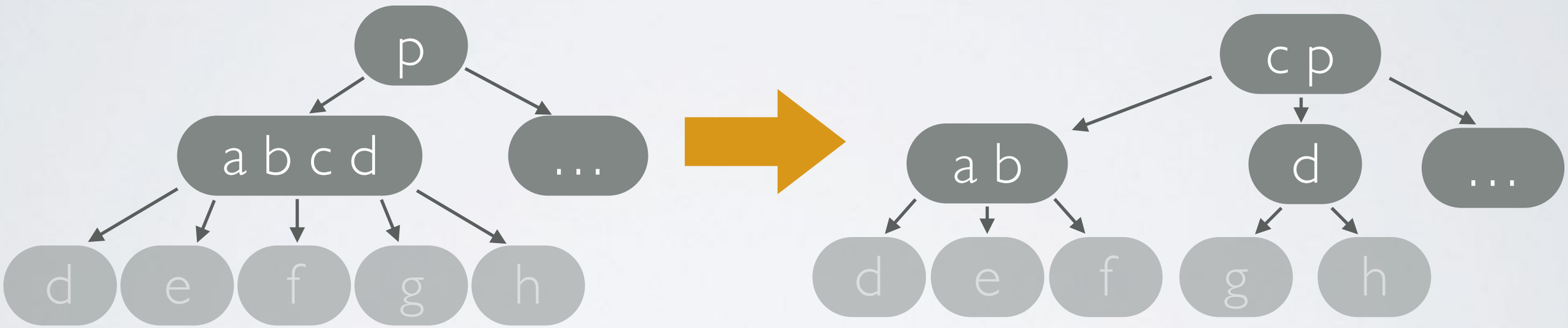
2-3-4 ДЕРЕВЬЯ: ПОИСК И ВСТАВКА

- Поиск как в обычном дереве поиска.
- Вставка в 2-узел: превращаем его в 3-узел.
- Вставка в 3-узел: превращаем его в 4-узел.
- Вставка в 4-узел: временно создаем 5-узел, вытаскиваем одно из значений и добавляем его в родителя.

ВСТАВКА:
5-УЗЕЛ КАК КОРЕНЬ



ВСТАВКА:
5-УЗЕЛ С РОДИТЕЛЕМ

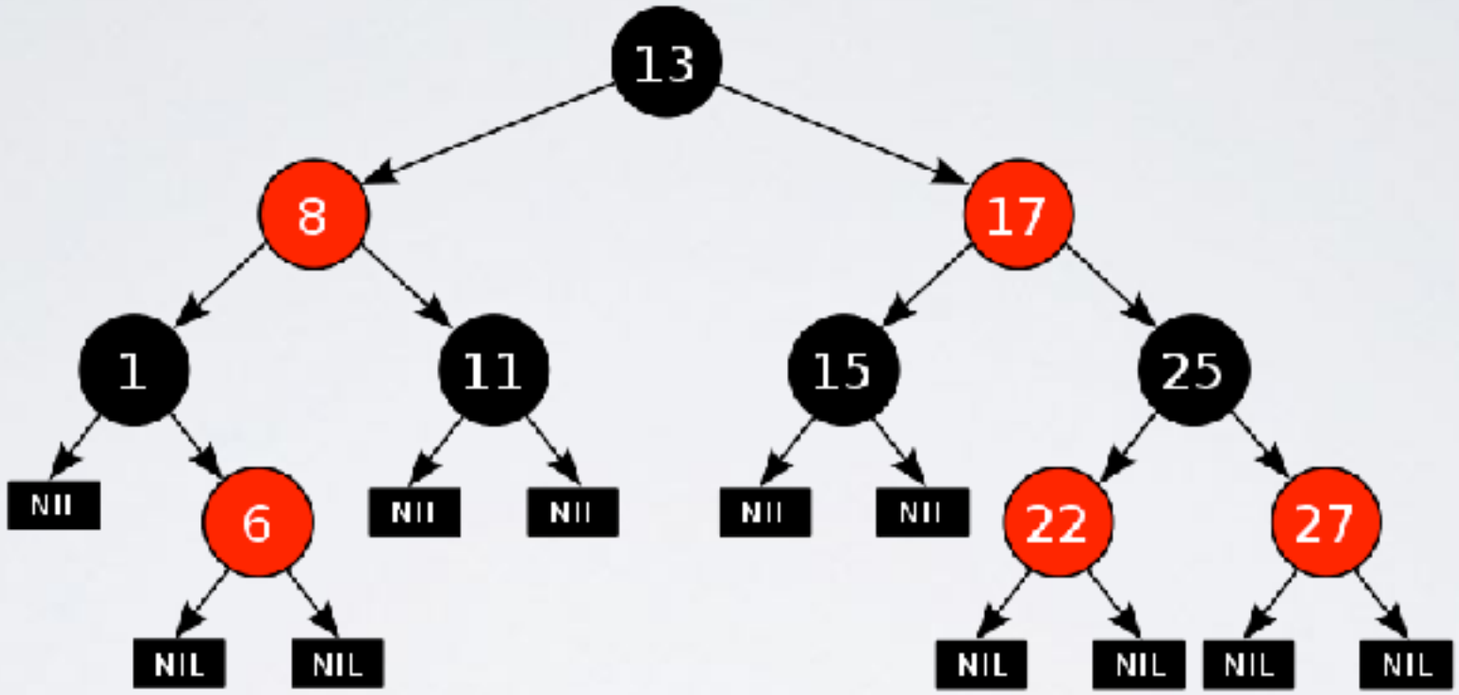


Вытягиваем одно из значений на уровень выше и продолжаем рекурсивно идти наверх, если получился новый 5-узел.

2-3-4 ДЕРЕВЬЯ: АНАЛИЗ

- Высота дерева: $\log_4(N) \leq h(N) \leq \log_2(N)$.
- Всегда идеально сбалансировано.
- Очень трудоемкая реализация, но идея-то хорошая!

КРАСНО-ЧЕРНЫЕ ДЕРЕВЬЯ
RED-BLACK TREES



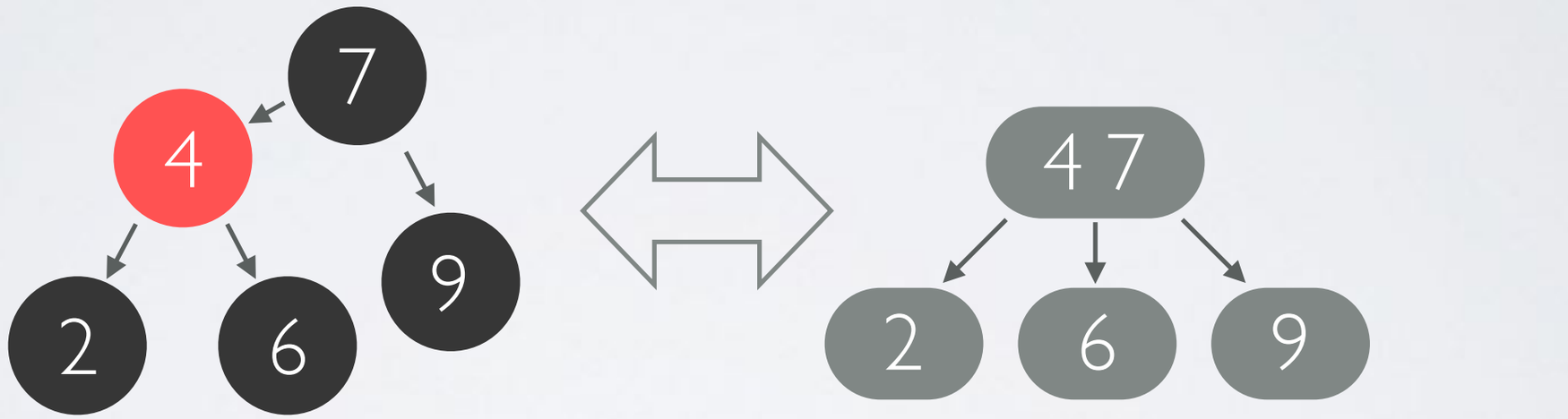
1. Все узлы либо **красные**, либо **черные**. Корень **черный**.
2. Потомки **красного** узла **черные**.
3. Все листья (NIL) **черные**.
4. Пути от любого узла до потомков содержат одинаковое количество **черных** узлов.
5. **(Следствие)** Пути от корня до двух любых узлов отличаются не более чем в 2 раза.

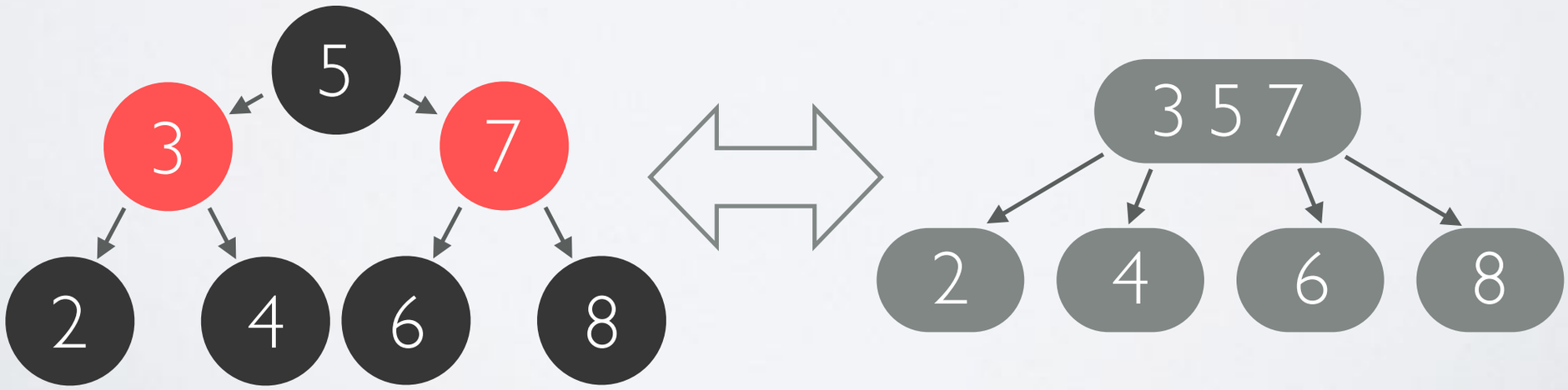
RB TREE: АНАЛИЗ

- Описаны и изучены в 1970-ые, с тех пор стандарт де-факто.
- Производительность сравнима с AVL-деревьями.
- Реализация сложна. Шесть возможных случаев вставки и симметричные им…
  - И еще столько же на удаление…

RB КАК 2-3-4

Красный узел будем интерпретировать как часть родителя, а не как отдельный узел:

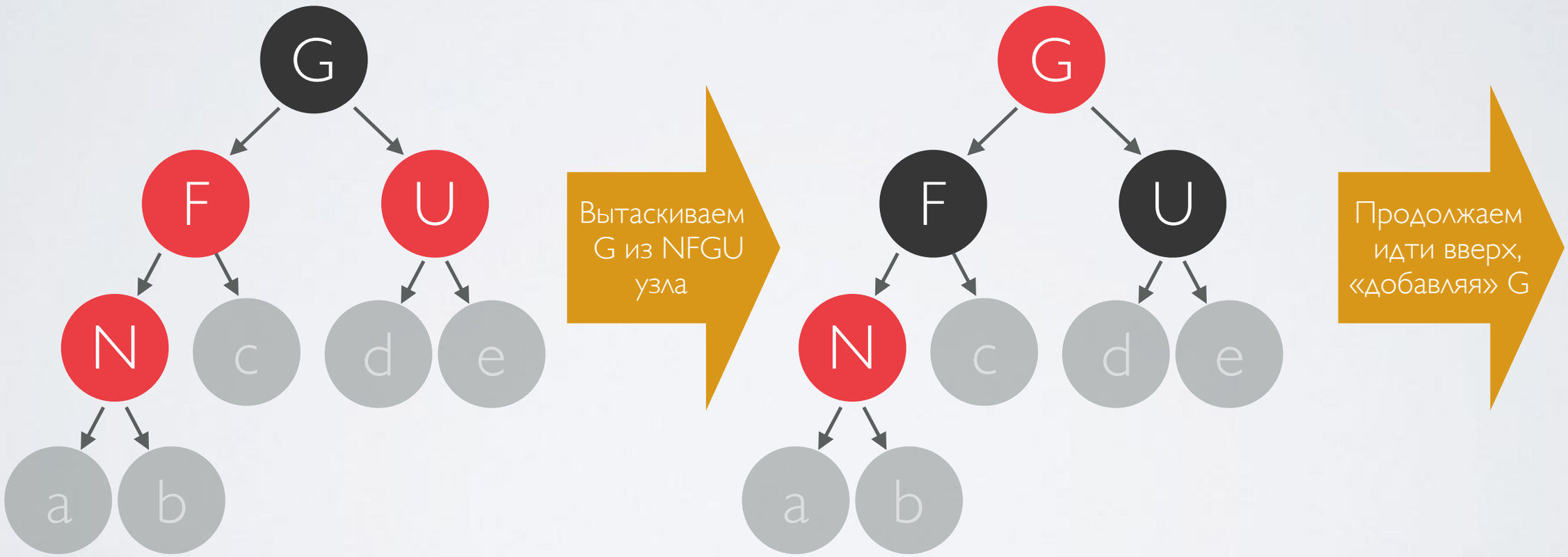




RB ВСТАВКА

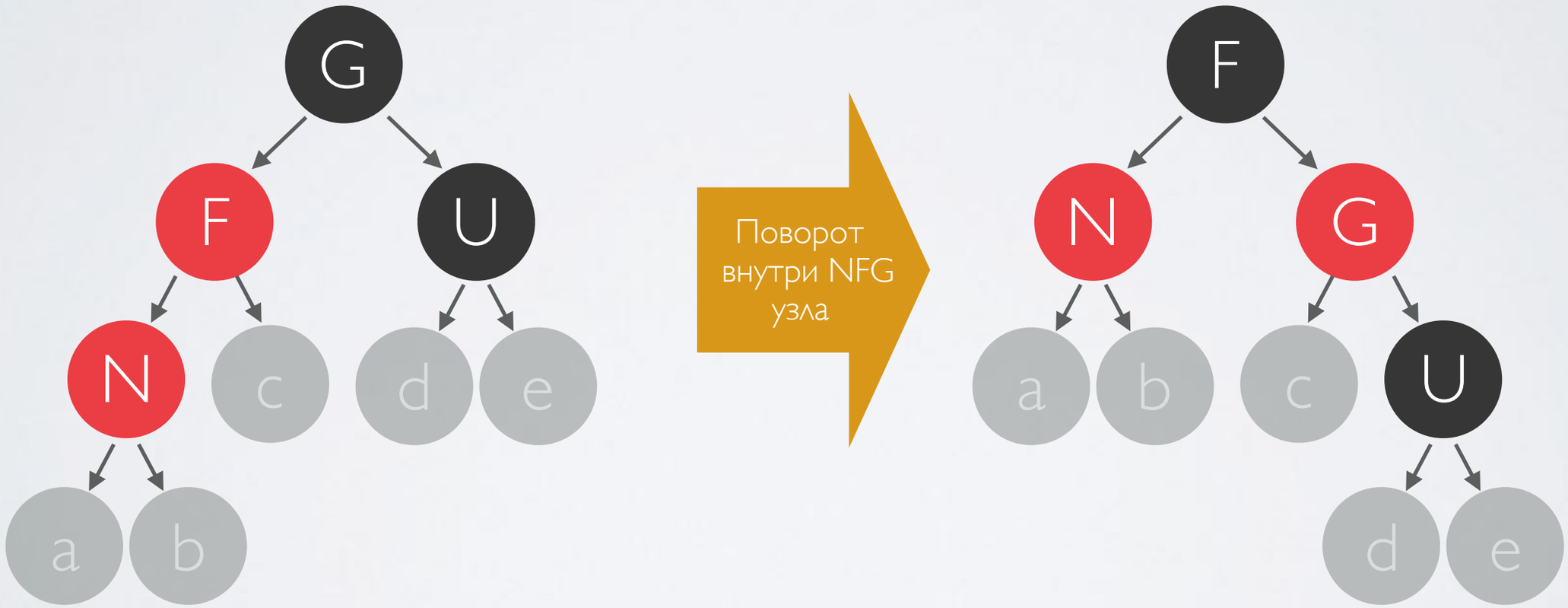
- Вставляемый узел – **красный**.
- Вставка в корень – нет проблем (красим в **черный** цвет).
- Вставка, когда отец **черный** – нет проблем.
- Если отец **красный**, то…

RB ВСТАВКА:
ОТЕЦ И ДЯДЯ КРАСНЫЕ



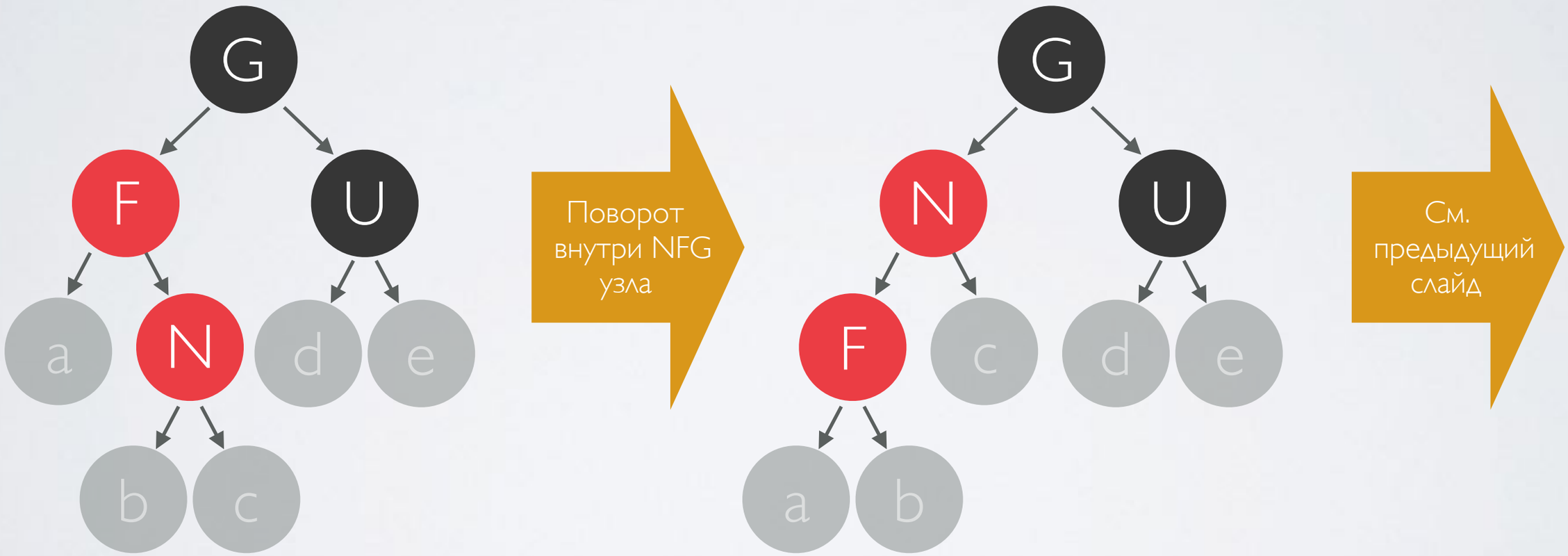
N – new, F – father, U – uncle, G - grandfather

RB ВСТАВКА:
ДЯДЯ **ЧЕРНЫЙ** (НОВЫЙ СЛЕВА)



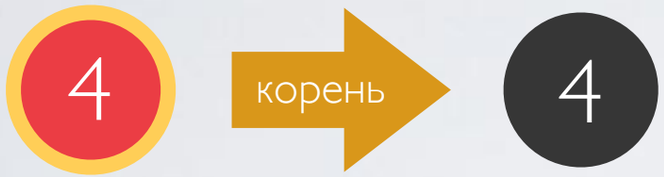
N – new, F – father, U – uncle, G - grandfather

RB ВСТАВКА:
ДЯДЯ **ЧЕРНЫЙ** (НОВЫЙ СПРАВА)



N – new, F – father, U – uncle, G - grandfather

ПРИМЕР RB

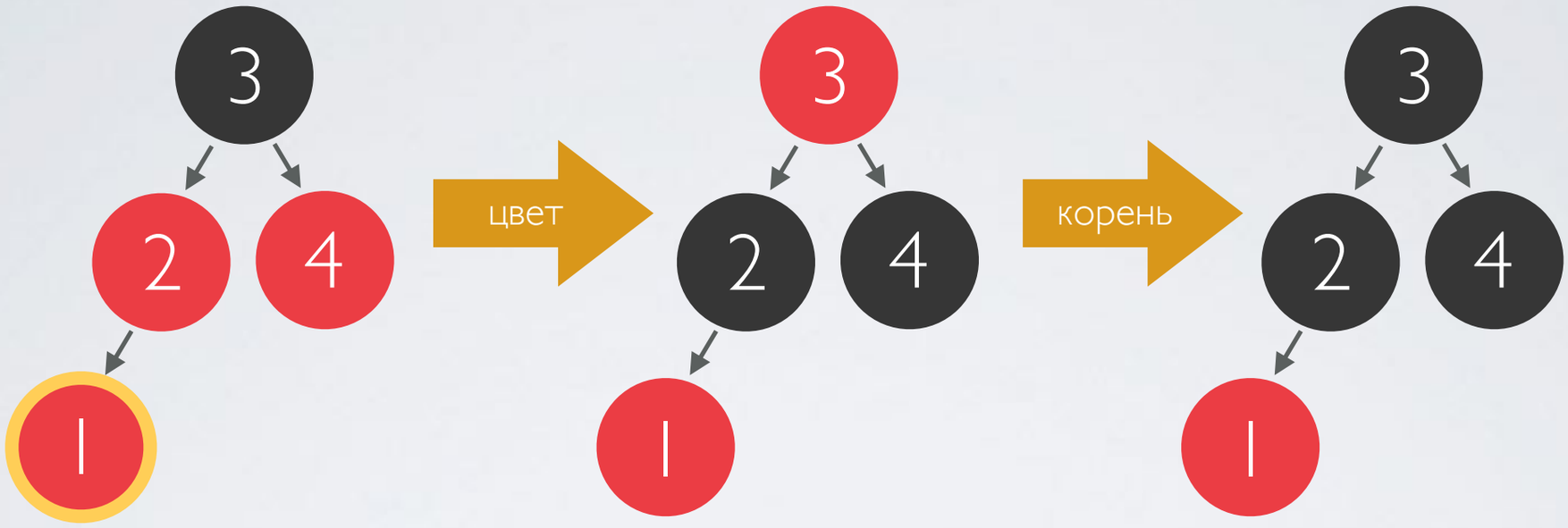


Вставляем 4

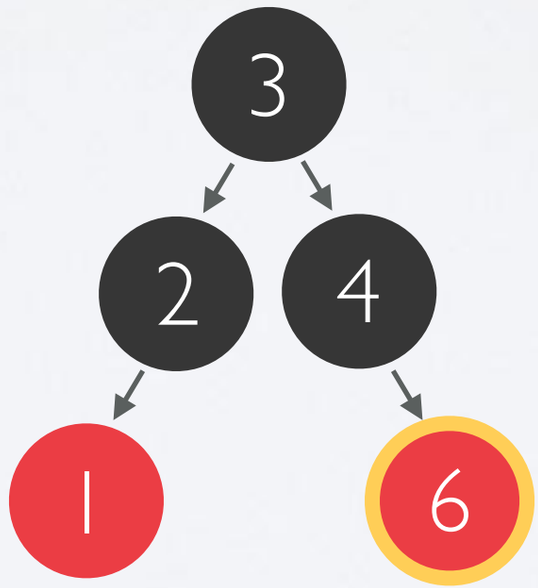
Вставляем 3

Вставляем 2

ПРИМЕР RB

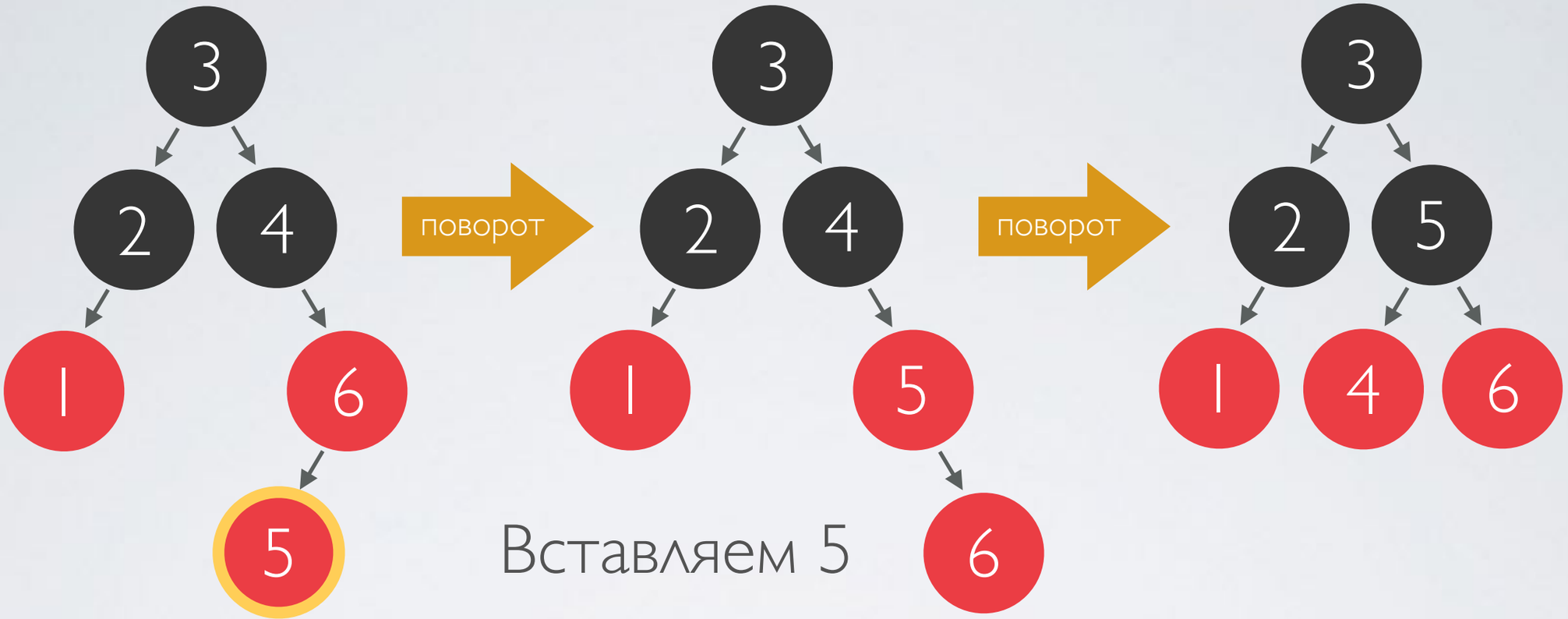


Вставляем 1



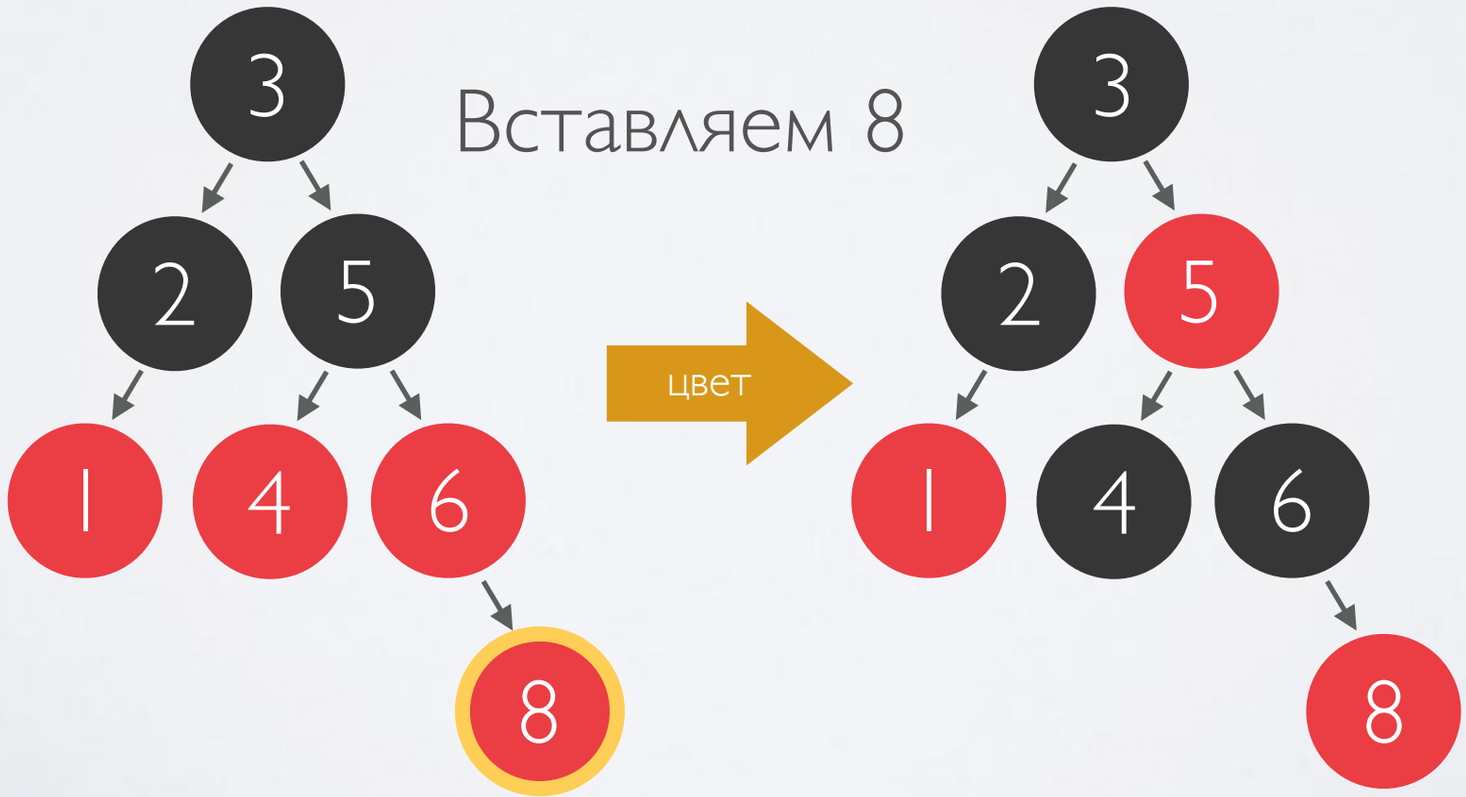
Вставляем 6

ПРИМЕР RB



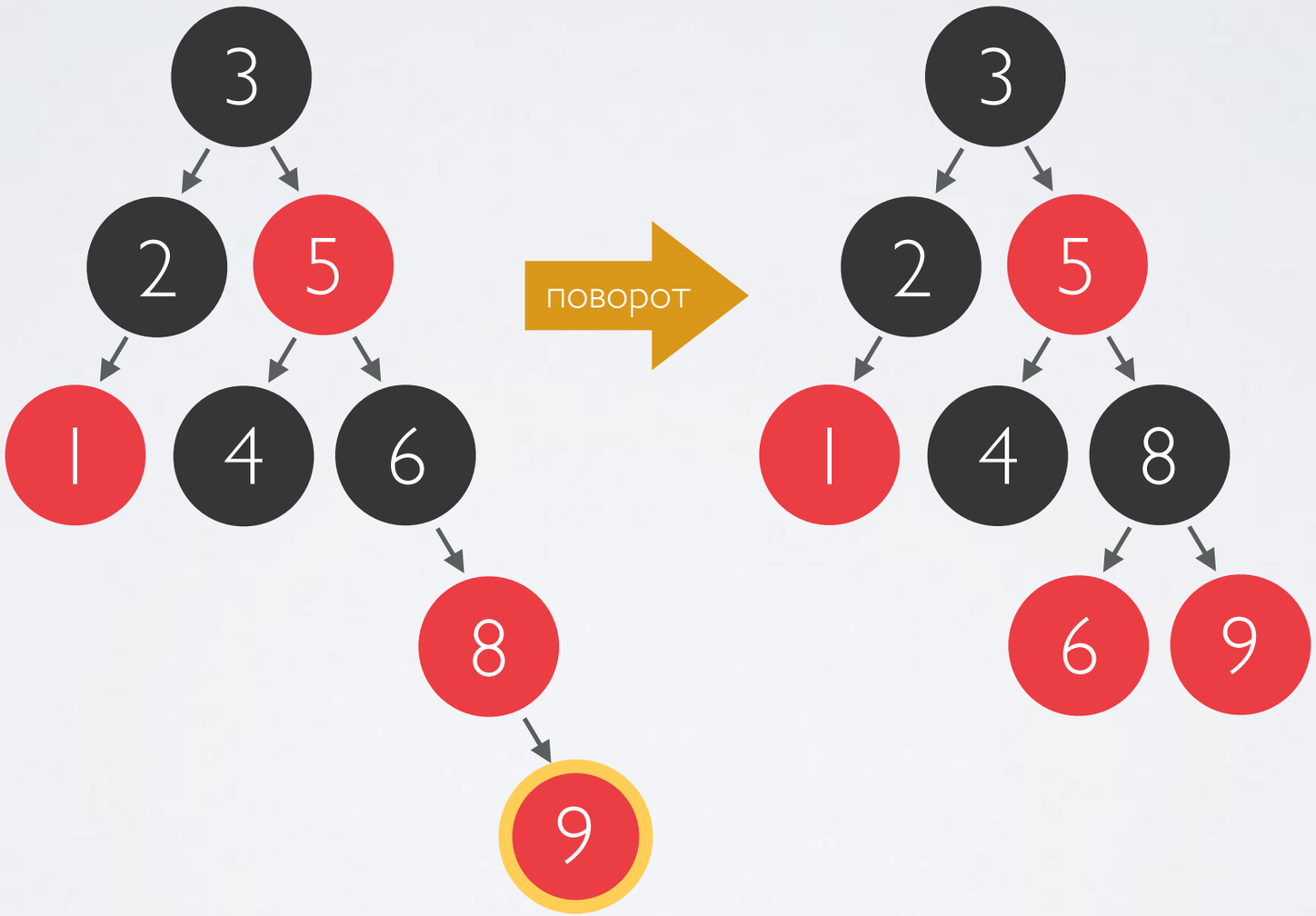
Вставляем 5

Вставляем 8



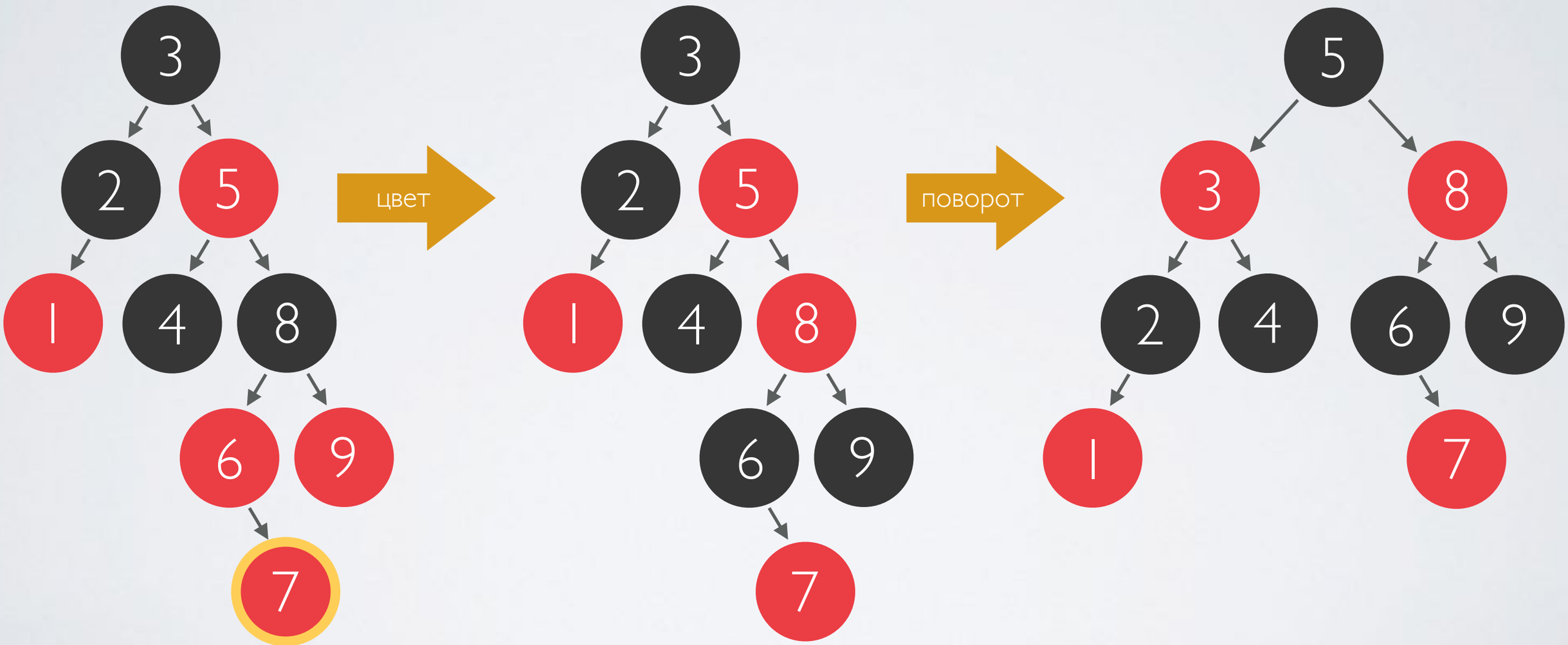
ПРИМЕР RB

Вставляем 9



ПРИМЕР RB

Вставляем 7

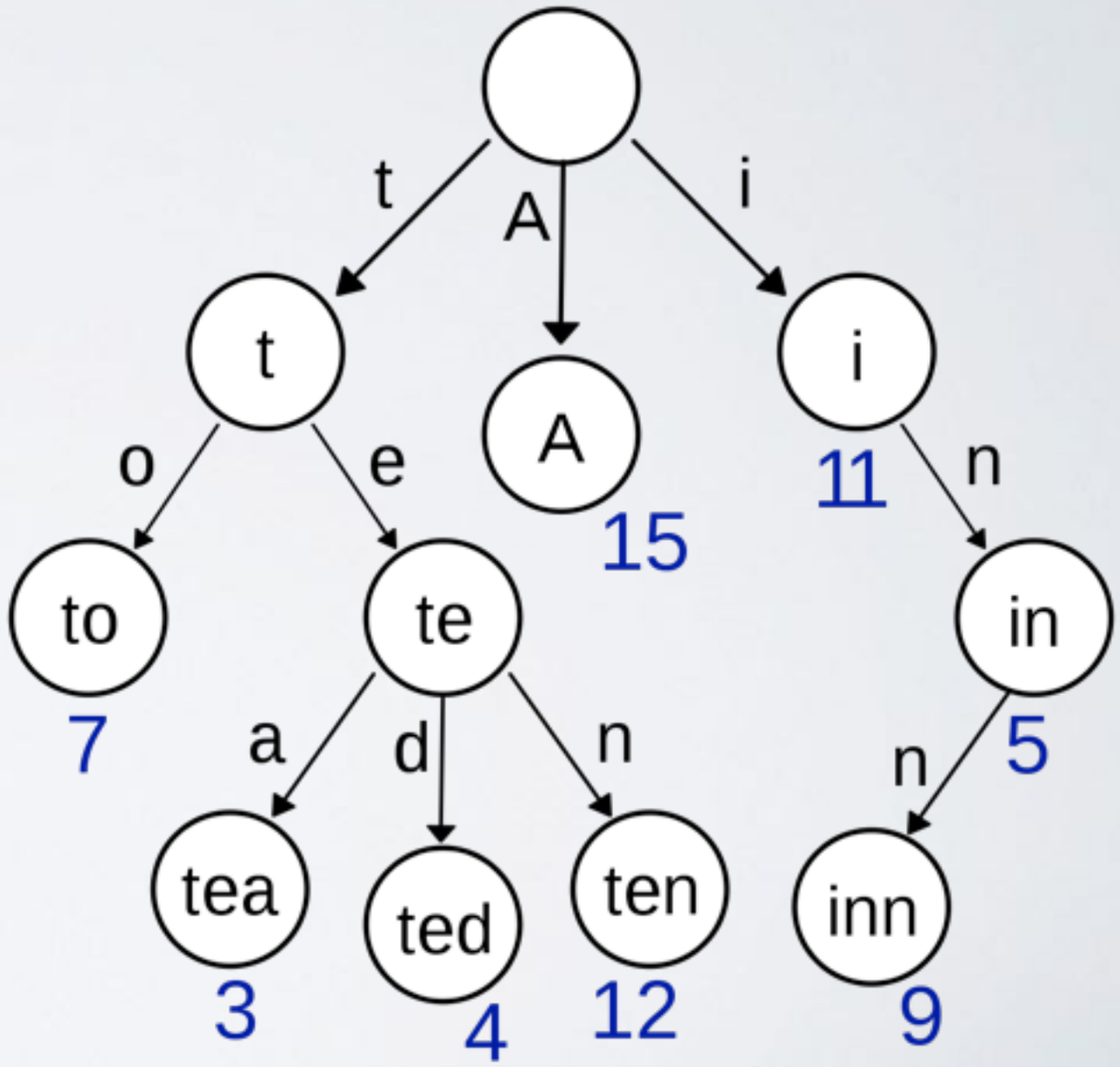


НЕ-БИНАРНЫЕ И НЕ-ПОИСК

Тоже деревья, но для другого

ПРЕФИКСНОЕ ДЕРЕВО (БОР, TRIE)

- A: 15
- to: 7
- tea: 3
- ted: 4
- ten: 12
- i: 11
- in: 5
- inn: 9



Сложность вставки и поиска?

O(длины строки)

ОЧЕРЕДЬ С ПРИОРИТЕТОМ

- АТД очередь с приоритетом (priority queue). Основные операции:
  - добавление элементов с некоторым приоритетом;
  - извлечение элемента с максимальным приоритетом.
- Эффективно реализуется через двоичную кучу («пирамида», binary heap), элемент с максимальным приоритетом в корне

ОЧЕРЕДЬ С ПРИОРИТЕТОМ: ИЗВЛЕЧЕНИЕ

- Запоминаем элемент из корня
- Переставляем последний элемент пирамиды в корень
- «Топим» корень для наведения порядка (функция sink из пирамидальной сортировки)
- Возвращаем запомненный элемент

ОЧЕРЕДЬ С ПРИОРИТЕТОМ: ДОБАВЛЕНИЕ

- Добавляем последний элемент в конец пирамиды
- «Всплываем» последний элемент для наведения порядка (функция swim, аналогичная sink)

ОЧЕРЕДЬ С ПРИОРИТЕТОМ: СЛОЖНОСТЬ

- Реализация через неотсортированный массив:
  - добавление за O(1), извлечение за O(N)
- Реализация через отсортированный массив:
  - добавление за O(N), извлечение за O(1)
- Реализация через пирамиду:
  - добавление и извлечение за O(log N)

КОНЕЦ ОДИННАДЦАТОЙ ЛЕКЦИИ