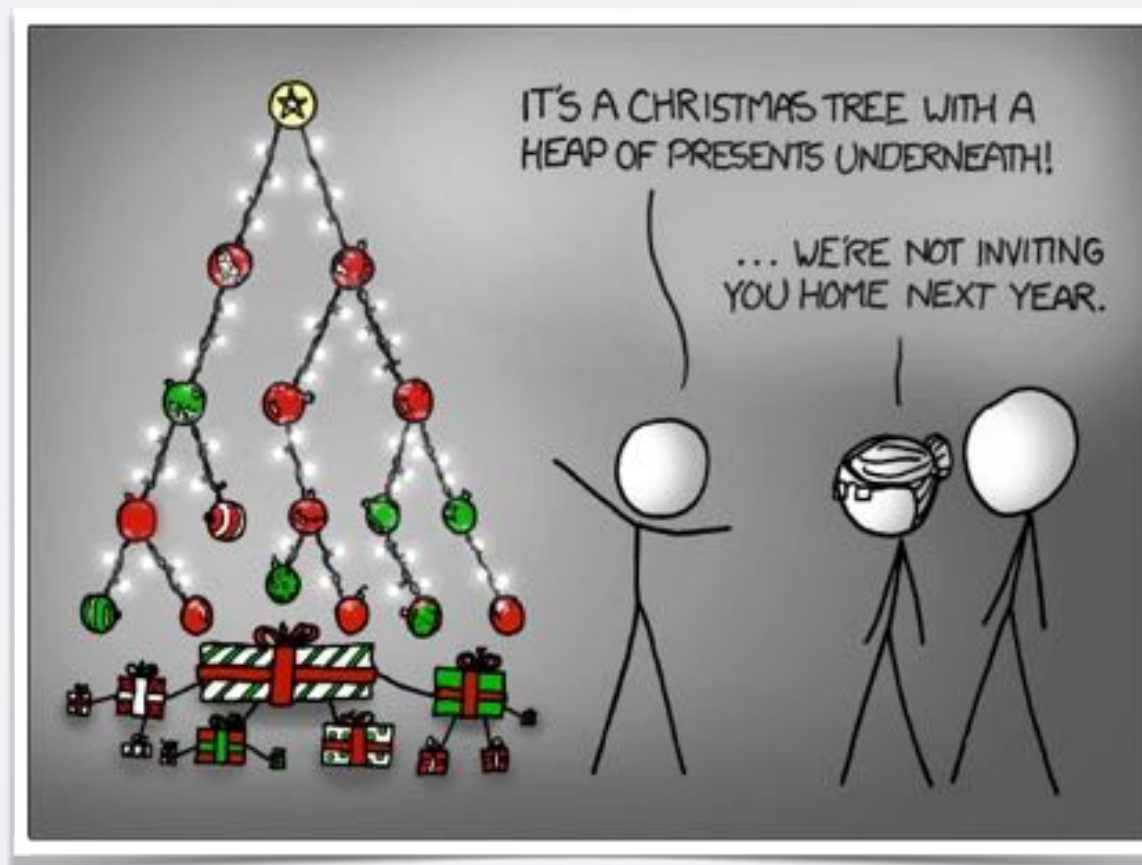


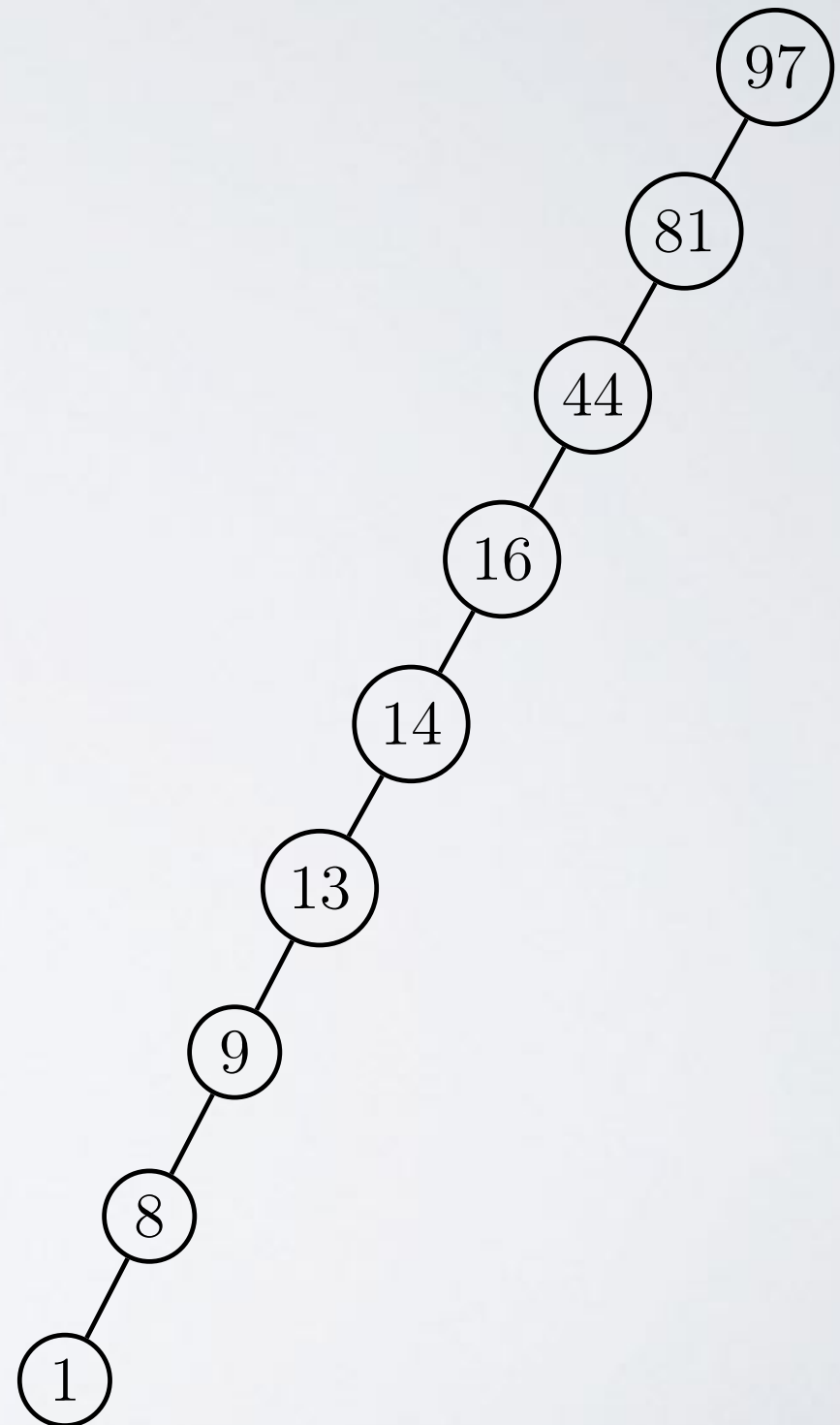
# ОСНОВЫ ПРОГРАММНОГО КОНСТРУИРОВАНИЯ

Лекция № 9  
31 октября 2016 г.



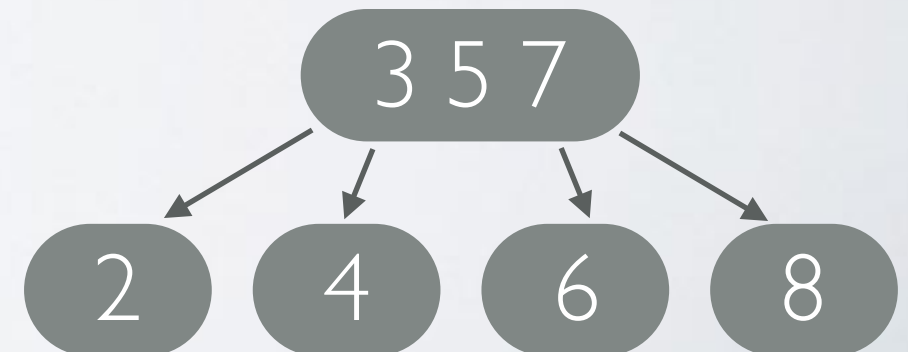
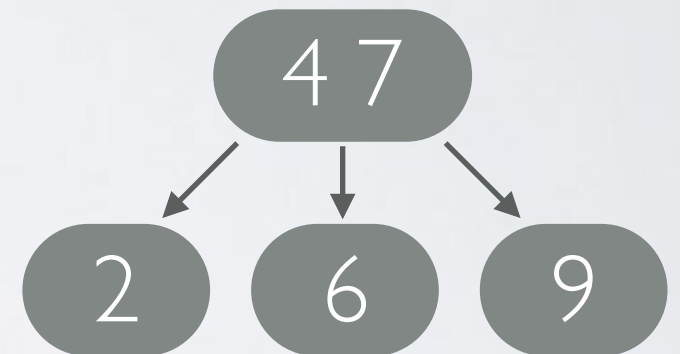
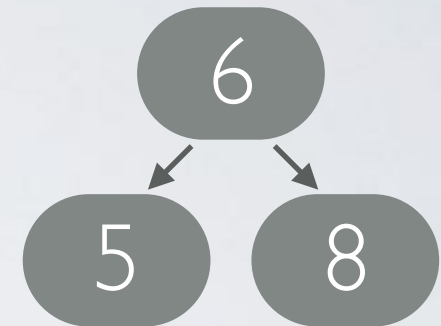
# РЕШЕНИЯ ПРОБЛЕМЫ «КРИВЫХ» ДЕРЕВЬЕВ

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



# 2-3-4 ДЕРЕВЬЯ

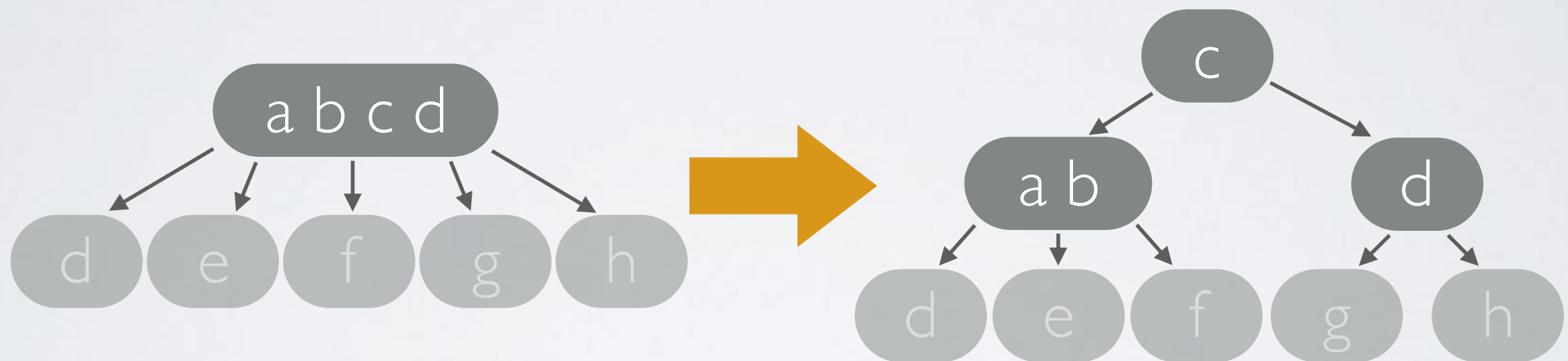
- Дерево поиска, узлы которого:
  - либо пусты;
  - либо 2-узел: 1 значение, 2 поддерева;
  - либо 3-узел: 2 значения, 3 поддерева;
  - либо 4-узел: 3 значения, 4 поддерева.
- Всегда идеально сбалансировано: высоты всех поддеревьев равны.



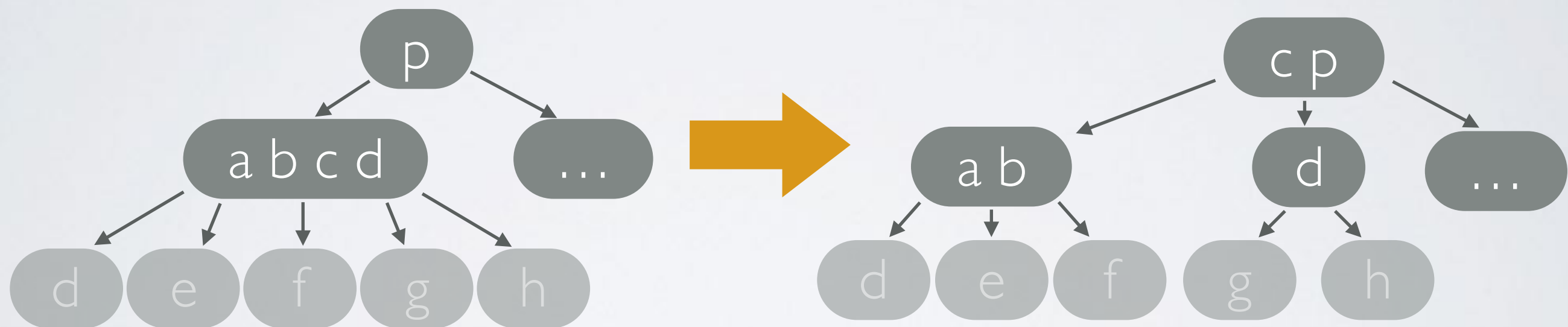
# 2-3-4 ДЕРЕВЬЯ: ПОИСК И ВСТАВКА

- 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



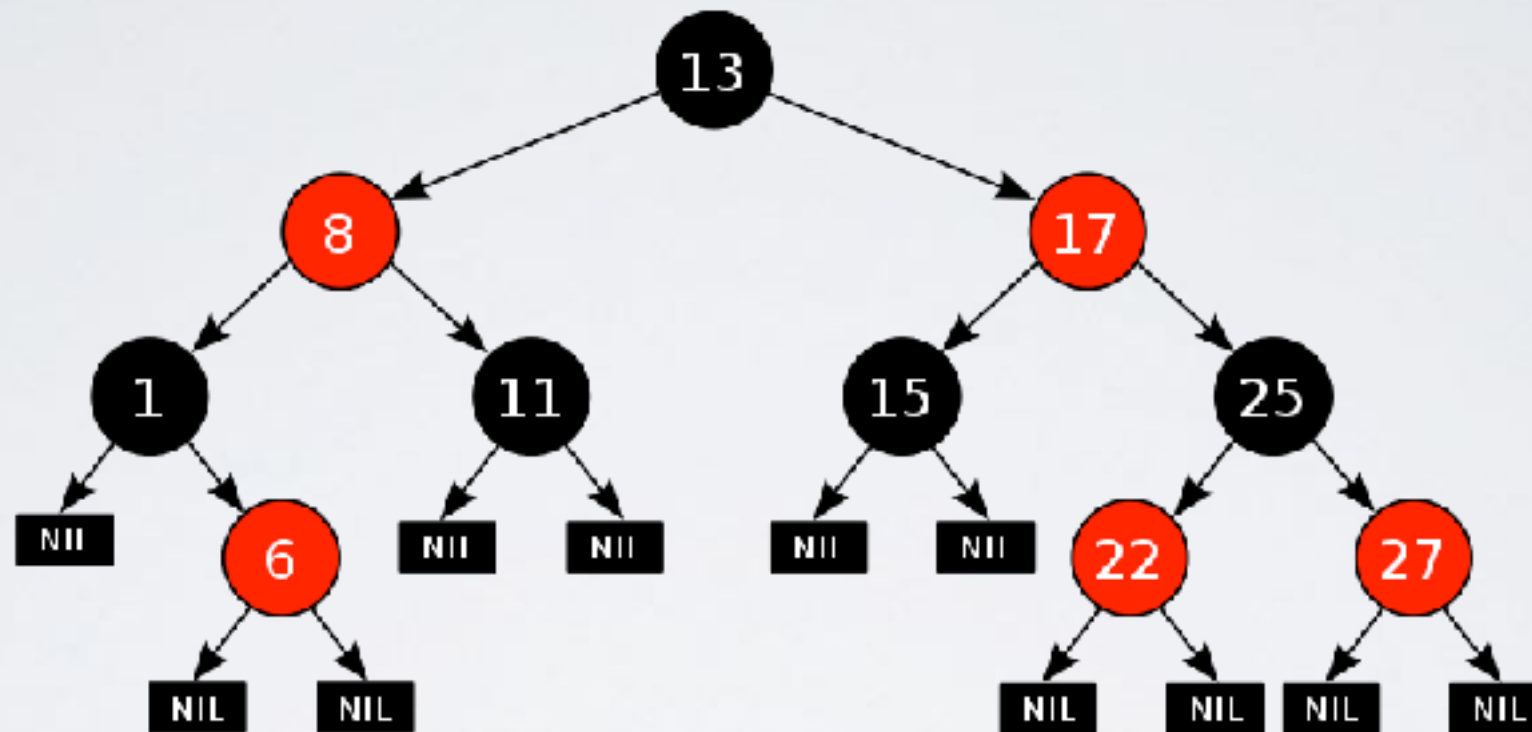
Вытягиваем одно из значений на уровень выше и  
продолжаем рекурсивно идти наверх, если получился  
новый 5-узел.

# 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.

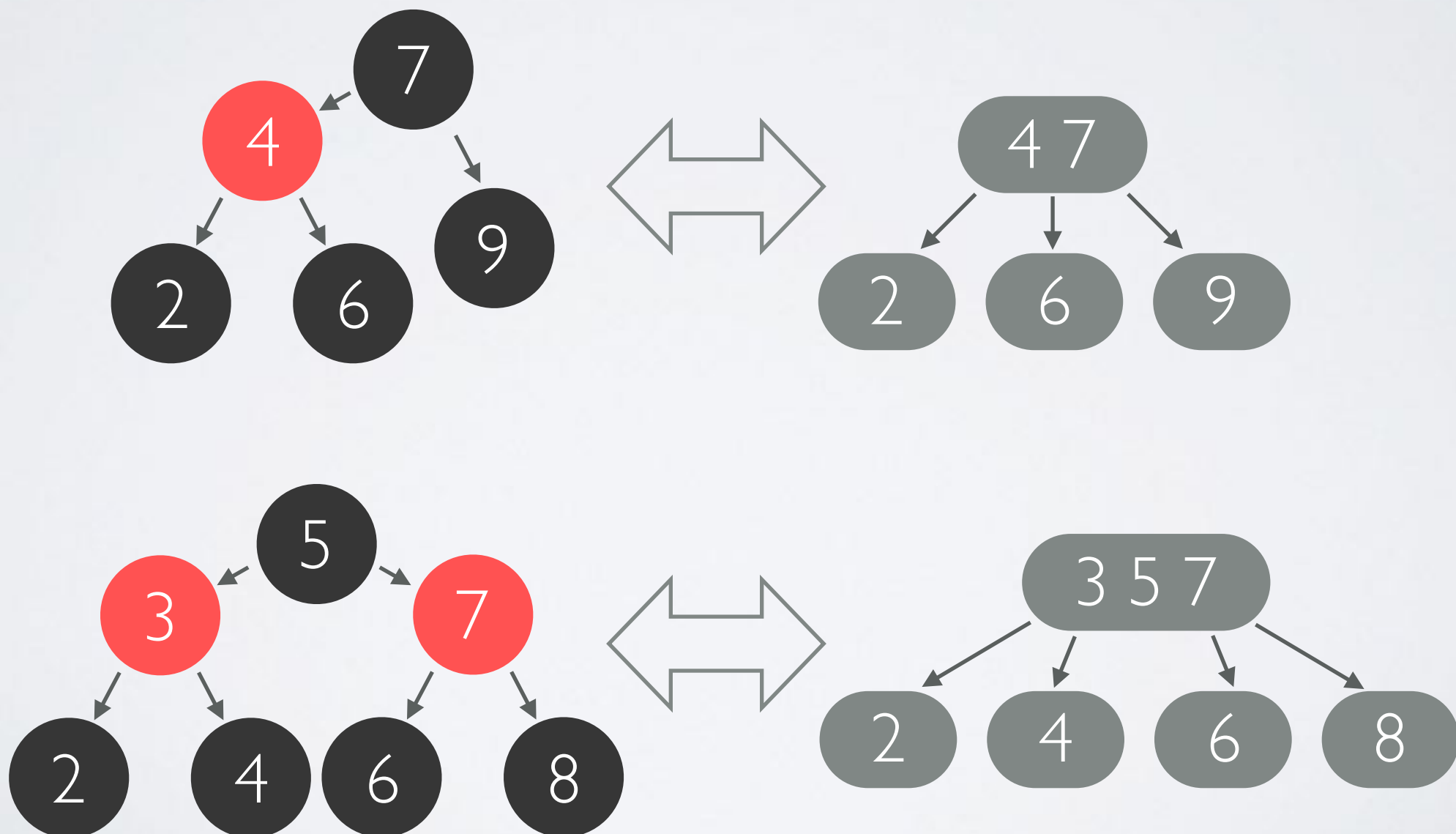


# RB TREE: АНАЛИЗ

- 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

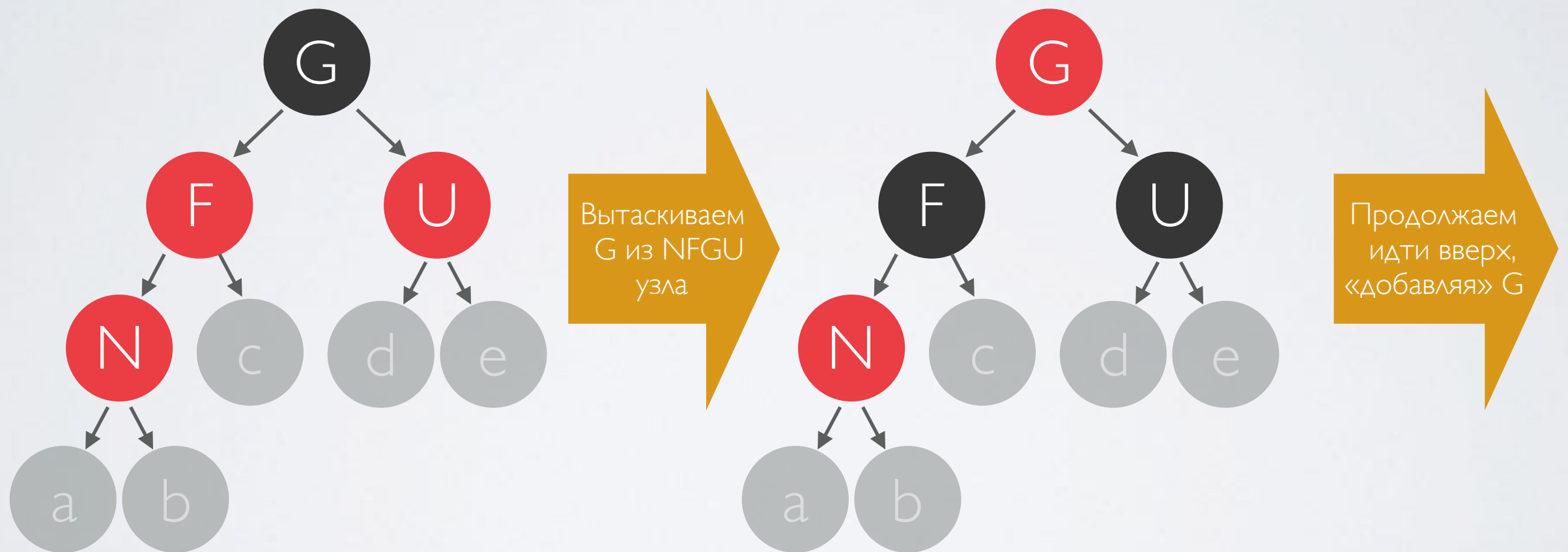
Красный узел будем интерпретировать как часть родителя, а не как отдельный узел:



# RB ВСТАВКА

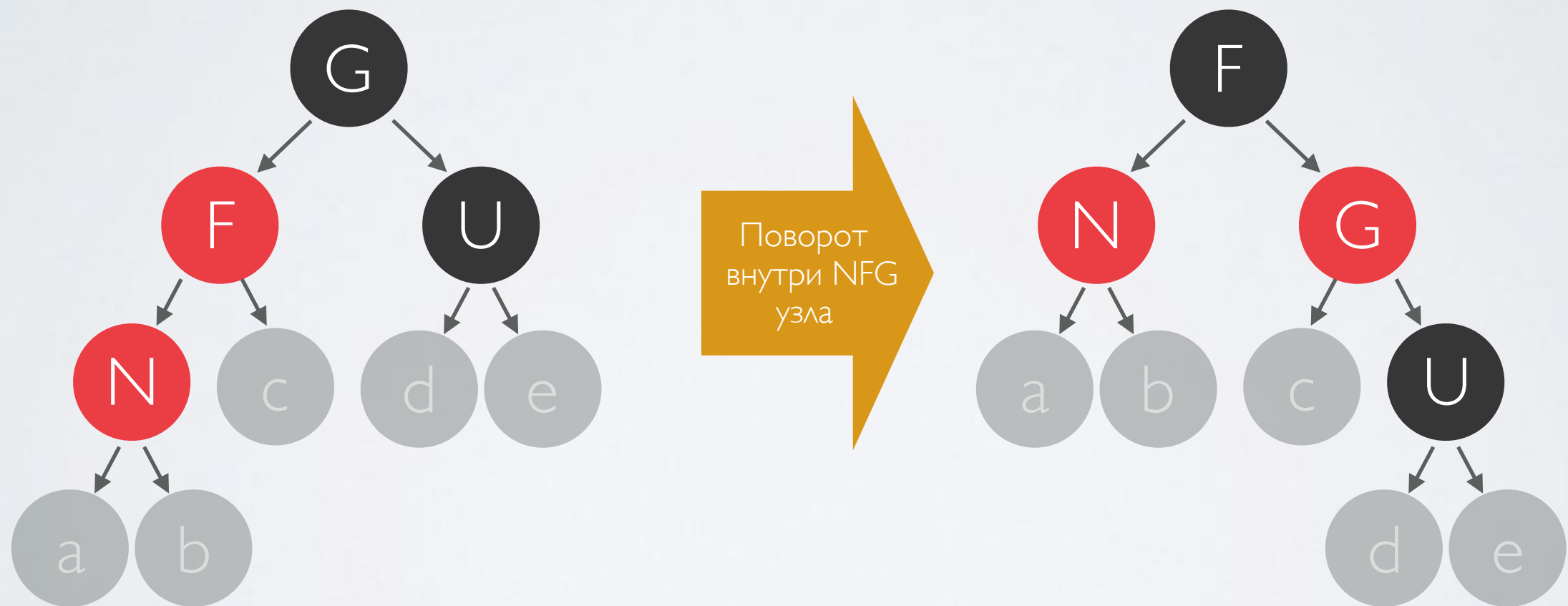
- Вставляемый узел — **красный**.
- Вставка в корень — нет проблем (красим в **черный** цвет).
- Вставка, когда отец **черный** — нет проблем.
- Если отец **красный**, то...

# RB ВСТАВКА: ОТЕЦ И ДЯДЯ **КРАСНЫЕ**



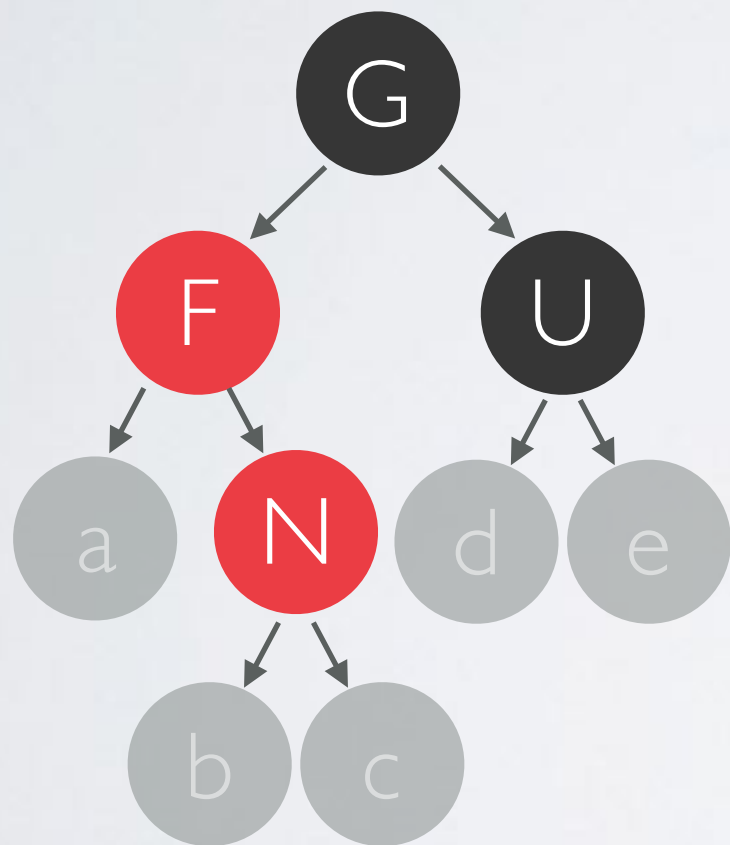
*N — new, F — father, U — uncle, G — grandfather*

# РВ ВСТАВКА: ДЯДЯ **ЧЕРНЫЙ** (НОВЫЙ СЛЕВА)

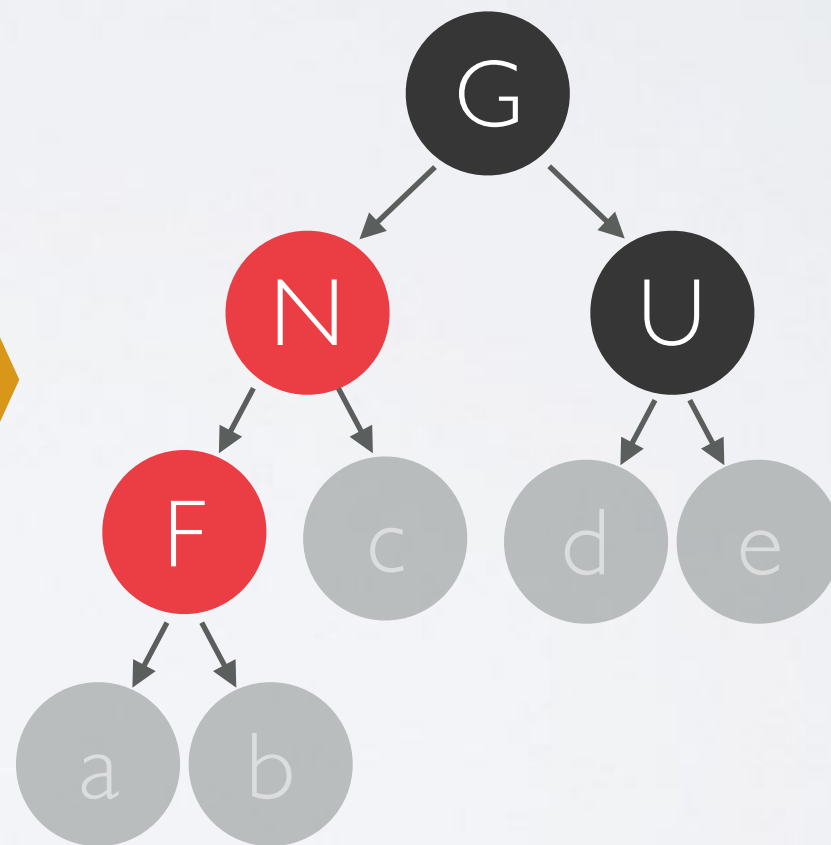


*N – new, F – father, U – uncle, G – grandfather*

# РВ ВСТАВКА: ДЯДЯ **ЧЕРНЫЙ** (НОВЫЙ СПРАВА)



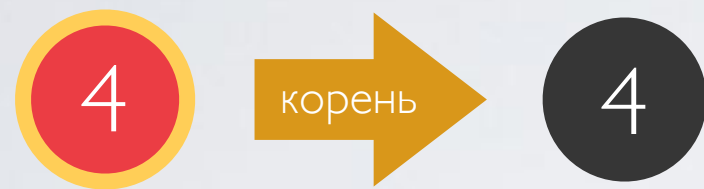
Поворот  
внутри NFG  
узла



См.  
предыдущий  
слайд

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

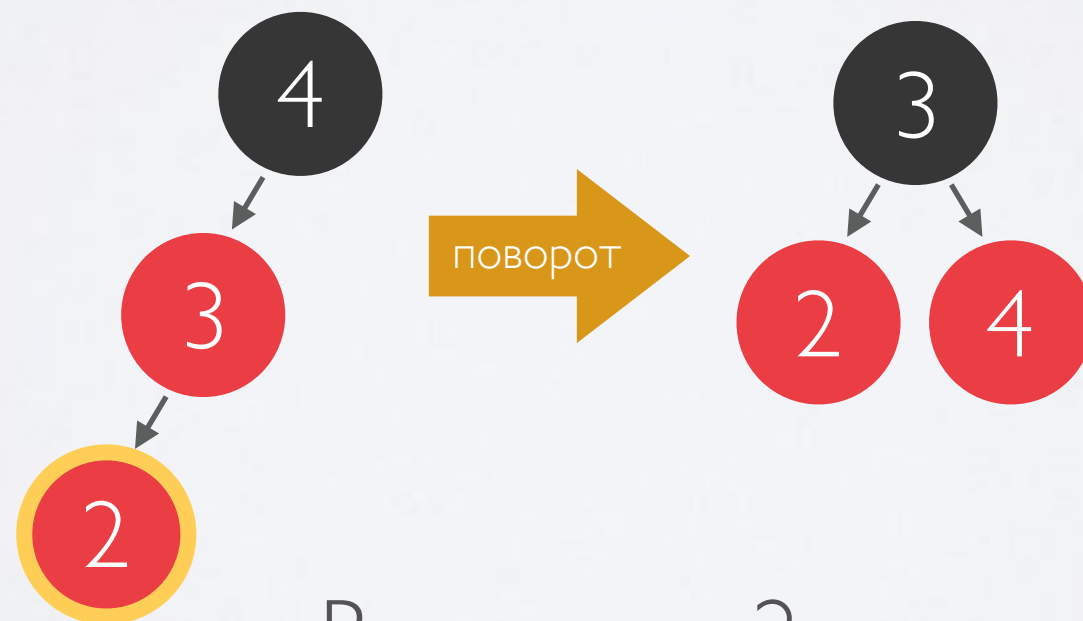
# ПРИМЕР RB



Вставляем 4

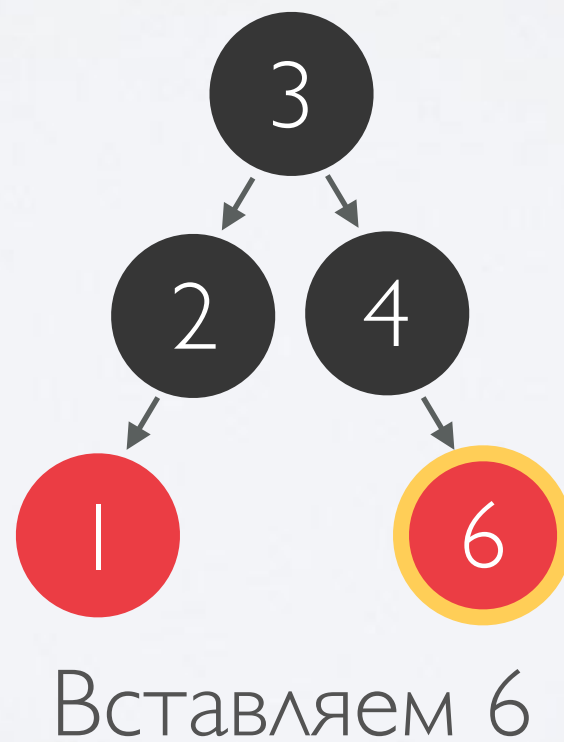
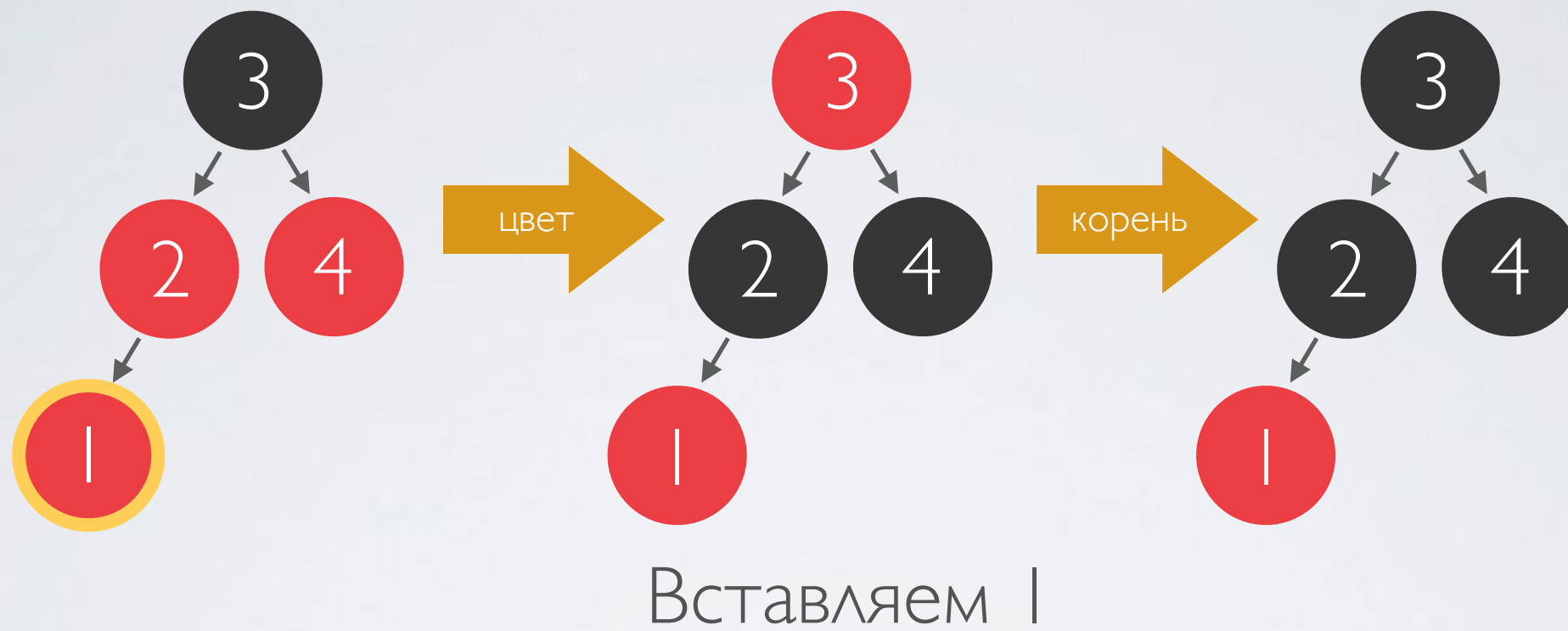


Вставляем 3



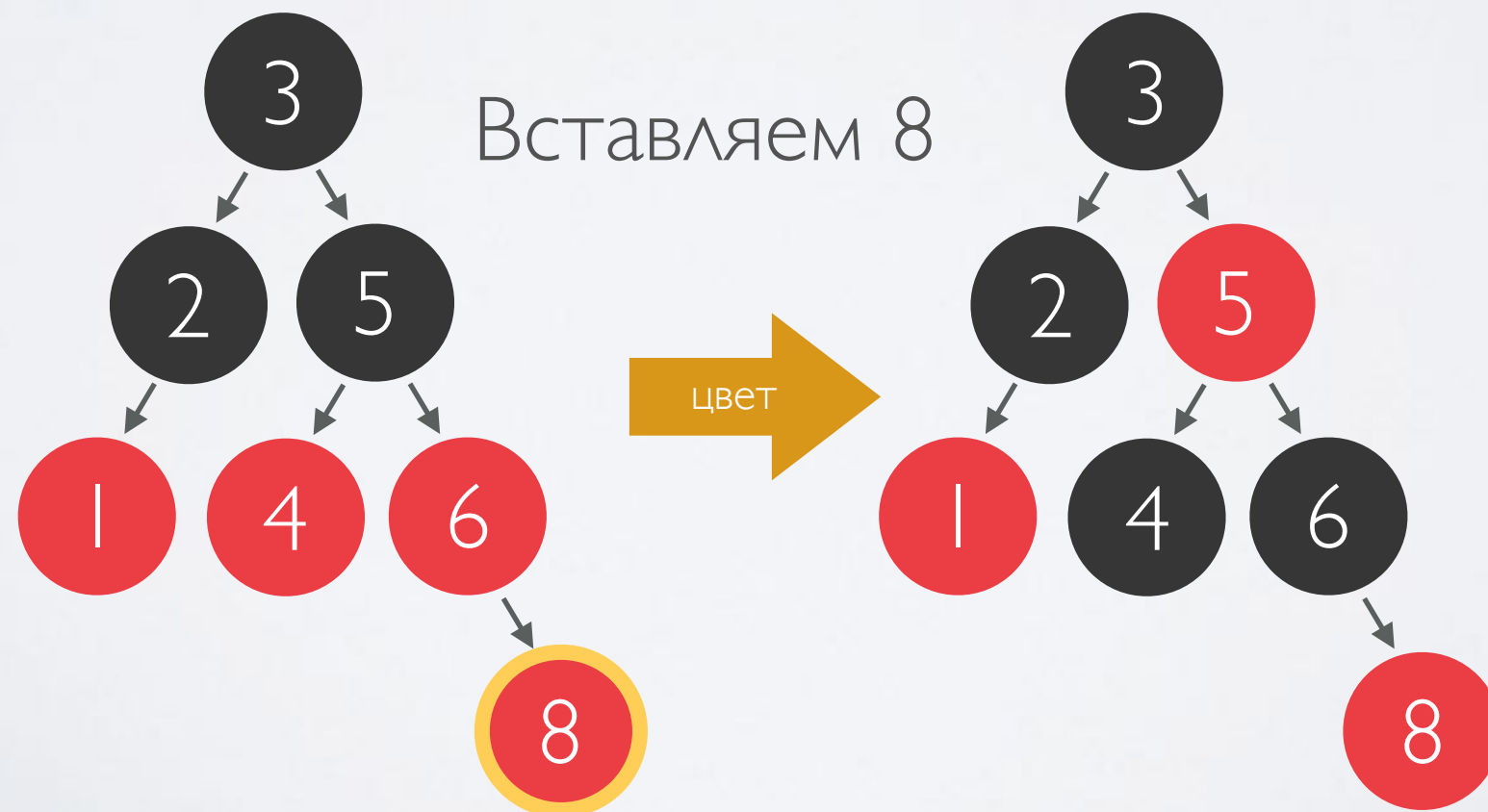
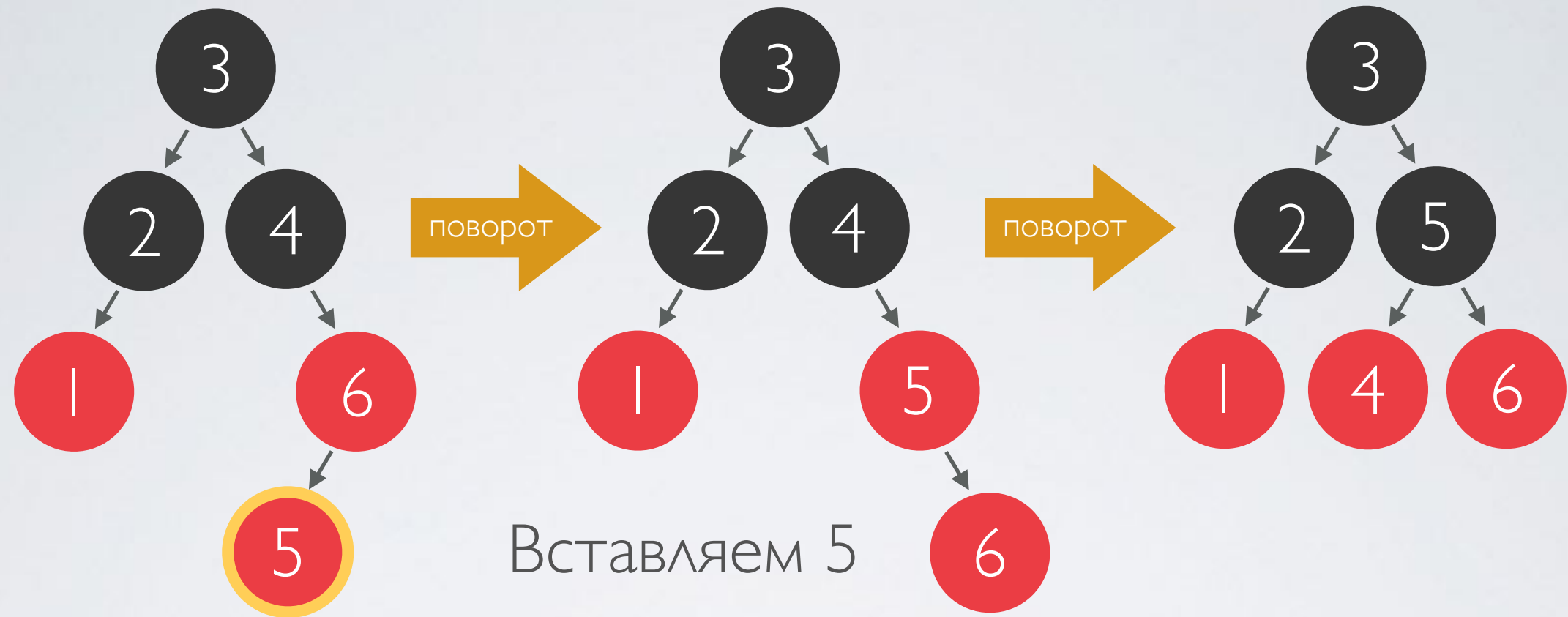
Вставляем 2

# ПРИМЕР RB



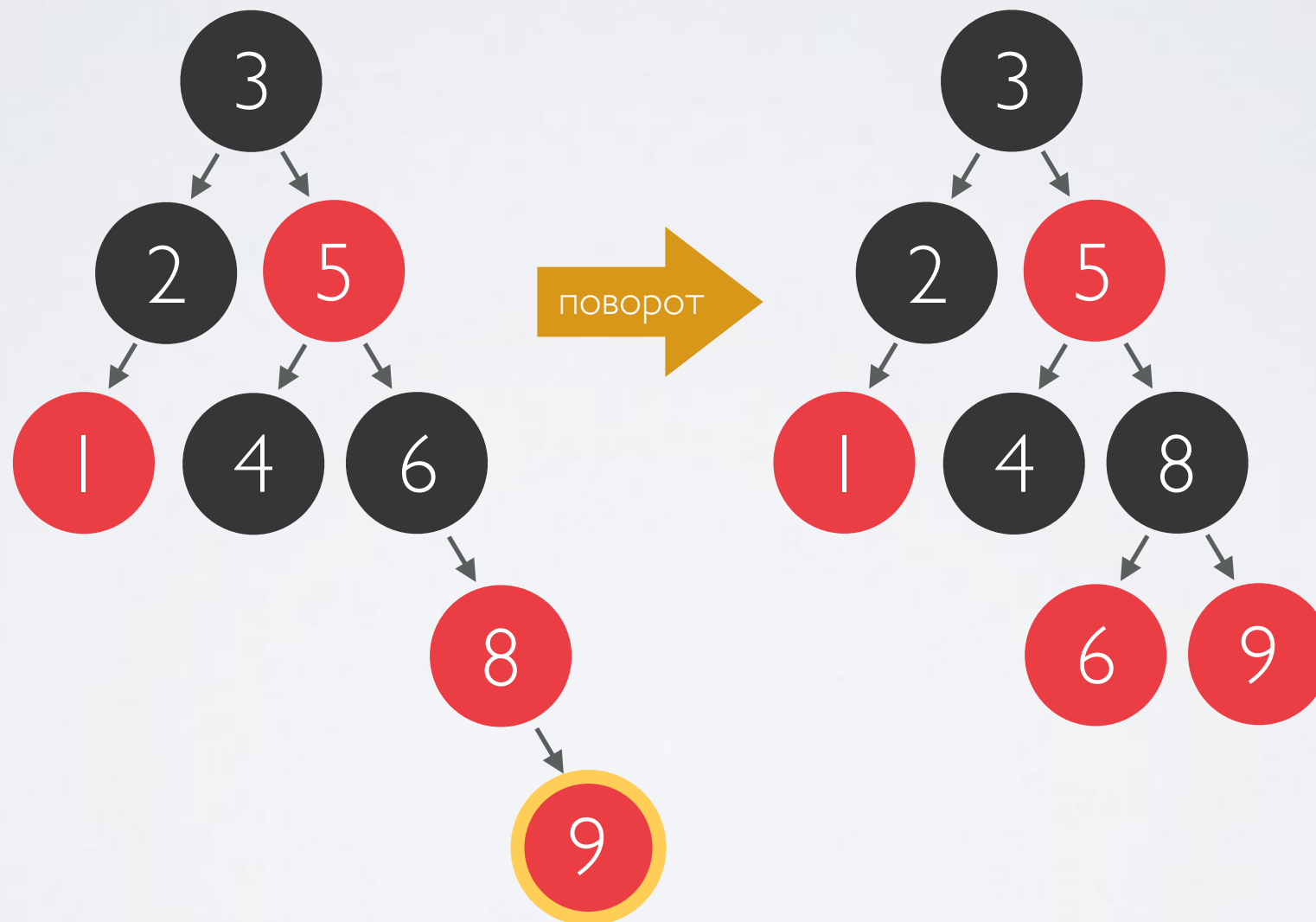


# ПРИМЕР RB



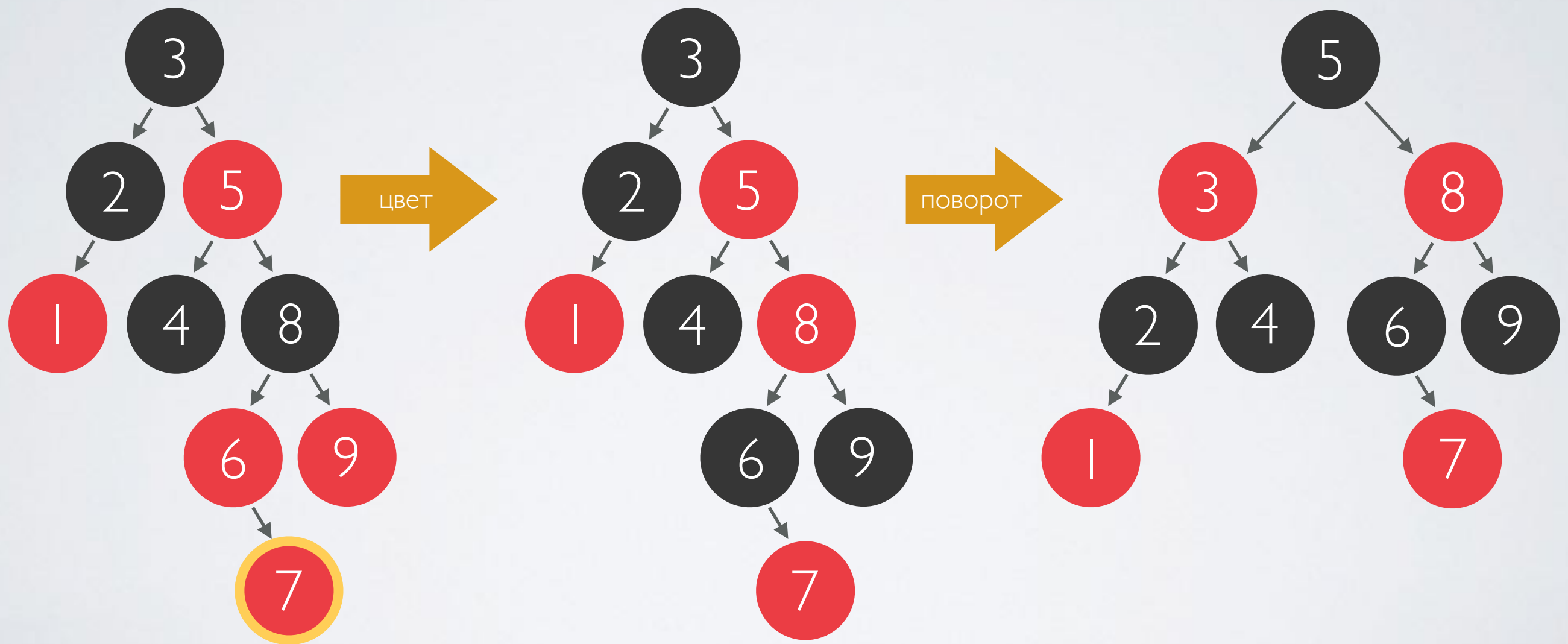
# ПРИМЕР RB

Вставляем 9



# ПРИМЕР RB

Вставляем 7



# ЗАДАЧКИ О ДЕРЕВЬЯХ

Распишите (разрисуйте) процесс добавления в пустое...

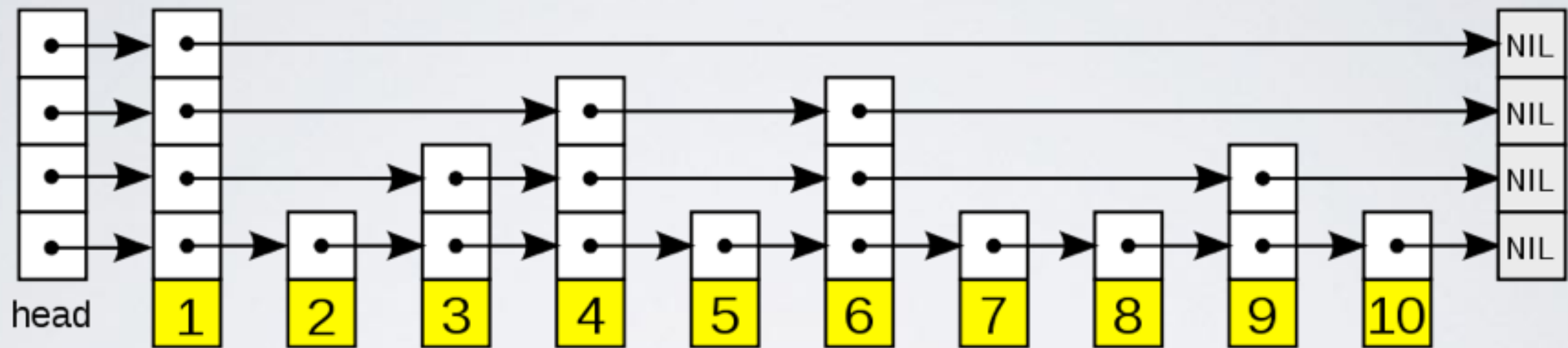
1. AVL-дерево значений: 4, 8, 9, 5, 6, 7.
2. 2-3-4-дерево значений: 2, 8, 5, 4, 9, 6, 7.
3. RB-дерево значений: 3, 6, 5, 9, 8, 7.

Каждая задача  
оценивается в 0,5 у.е.

# НЕ-ДЕРЕВЬЯ

Способы организации быстрого поиска,  
даже в чем-то похожие на деревья, но другие

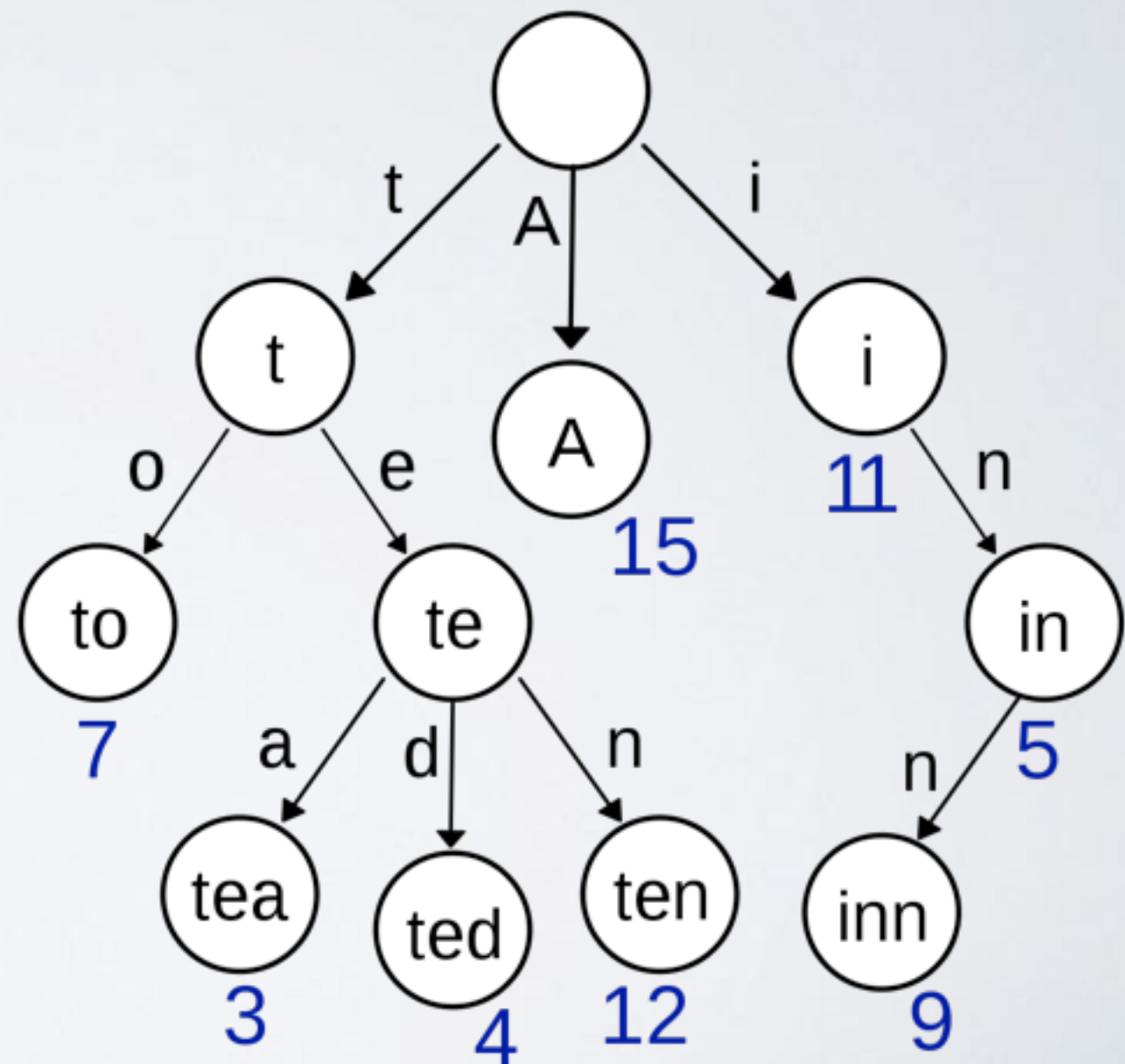
# СЛОЕНЫЕ СПИСКИ (СПИСКИ С ПРОПУСКАМИ)



- Нижний список всегда содержит все элементы.
- Вероятность попадания в список уровнем выше —  $p$  ( $p=1/2$  или  $1/4$ ). Еще выше —  $p^2$  и т.д. (кидаем монету).
- При удалении элемента удаляем его из всех списков.

# ПРЕФИКСНОЕ ДЕРЕВО (БОР, TRIE)

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



# НЕ-ПОИСК

Тоже деревья, но для другого



# 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):
  - добавление и извлечение за  $O(\log N)$ .



КОНЕЦ ДЕВЯТОЙ ЛЕКЦИИ