ITMO UNIVERSITY

Saint Petersburg, Russia

What about Binary Search Trees?

Vitaly Aksenov

aksenov.vitaly@gmail.com

Supported Operations

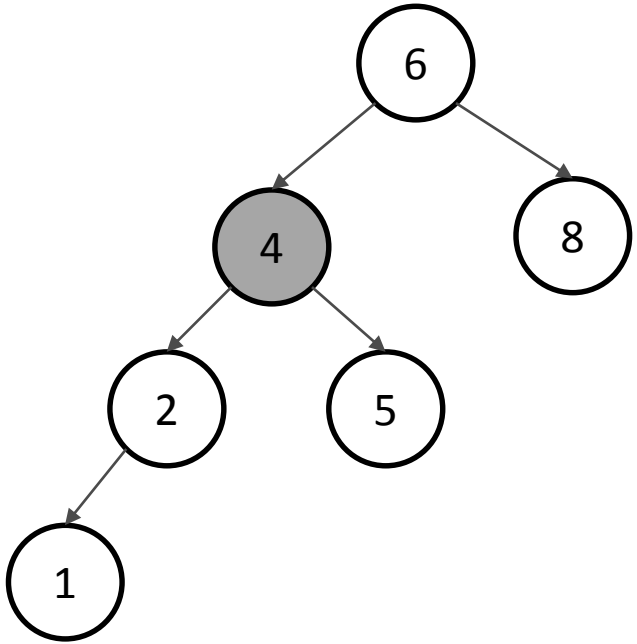
- `Insert(v)`
  - True, if v does not exist
  - False, if v exists
- `Delete(v)`
  - True, if v exists
  - False, if v does not exist
- `Contains(v)`
  - True, if v exists
  - False if v does not exist
- `Range(l, r)`
  - Out of the scope

Types of BSTs

- External
- Internal
- Partially-external

Partially-external BST

- The standard BST
- Two types of nodes: `DATA` (white) and `ROUTING` (grey)
- Invariant: each `ROUTING` node has two children
- The set consists of `DATA` nodes

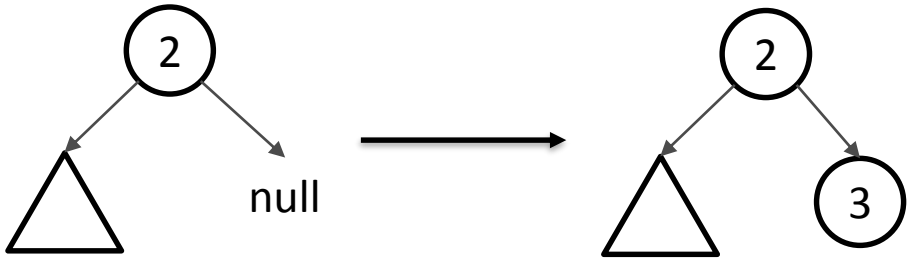


Partially-external BST. Traversal.

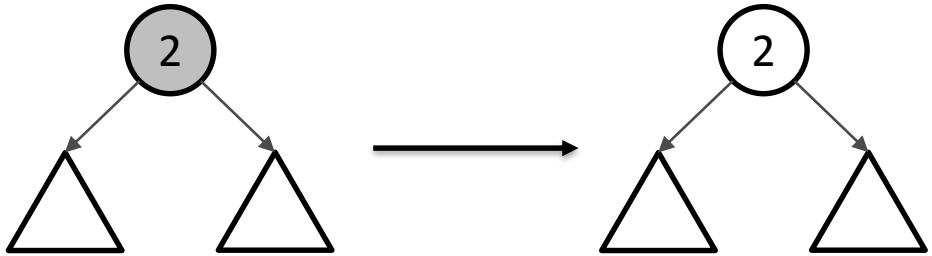
```
Window search(Node root, K key) {
    gprev = null, prev = null, curr = root
    while (curr.key != key && curr != null) {
        if (curr.key < key) {
            <gprev, prev, curr> = <prev, curr, curr.r>
        } else {
            <gprev, prev, curr> = <prev, curr, curr.l>
        }
    }
    return <gprev, prev, curr>
}
```

Partially-external BST. Insert.

Insert a leaf.
`insert(3)`

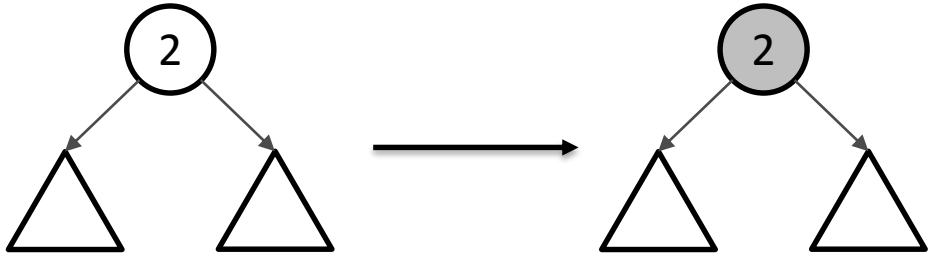


Insert in ROUTING.
`insert(2)`

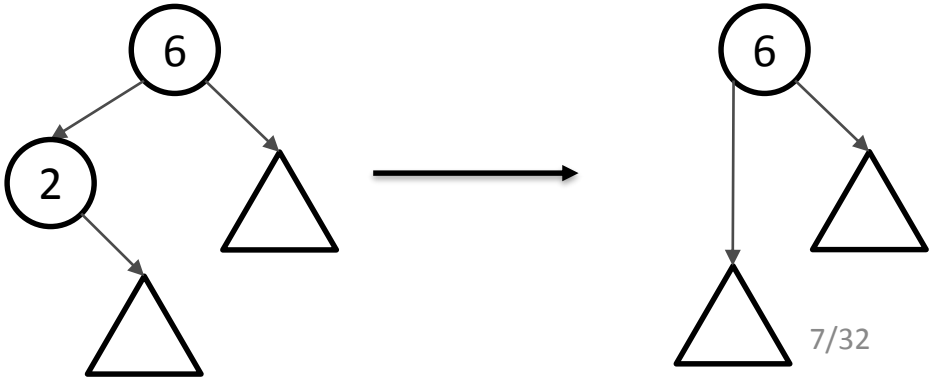


Partially-external BST. Remove (I)

Removal of a node with two children.
`remove(2)`

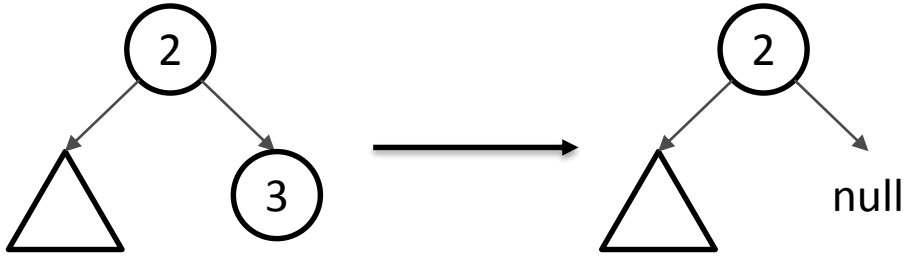


Removal of a node with one child.
`remove(2)`

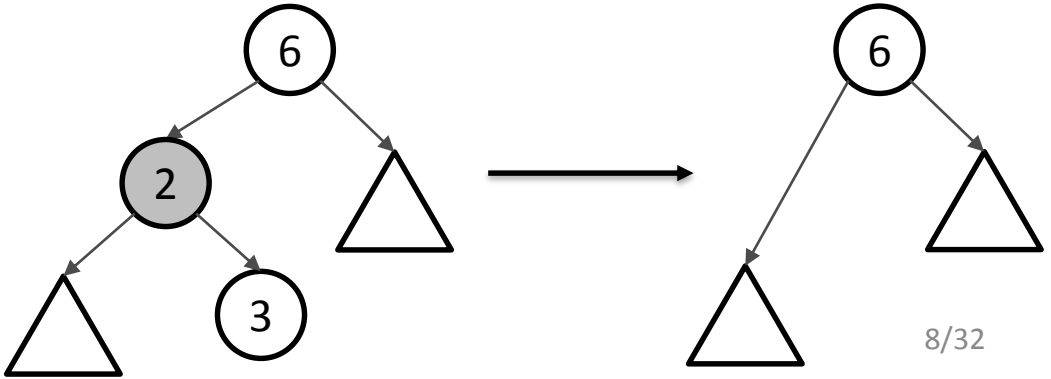


Partially-external BST. Remove (II)

Removal of a leaf with DATA parent.
`remove(3)`



Removal of a leaf with ROUTING parent.
`remove(3)`



ITMO more than a UNIVERSITY

Concurrent partially-external BST. First try.

- The easiest way is to write using locks.
- We need a `deleted` flag for the logical deletion.
- Two policies of taking the locks:
  - A lock on the whole node.
  - Different locks on status and edges.
- Which policy is better?
- We have to remember about the order of the locks, otherwise, there will be a deadlock. Take top-down.

Concurrent partially-external BST. First try.

What is the right way to make the lock. CLH (ReentrantLock in Java) is too resourceful. Spin lock is a good choice: instead of CAS we will spin.

The pseudocode appears to be the following:

```
void lock(lock, check) {
    if (!check()) {
        restart
    }
    lock.lock()
    if (!check()) {
        lock.unlock()
        restart
    }
}
```

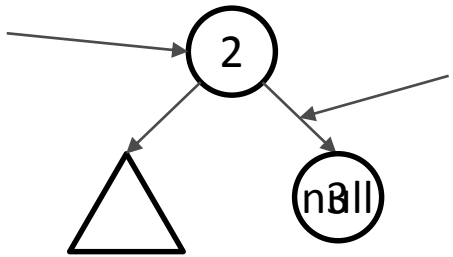
By that, we can do concurrency-optimal BST.

It is a little tricky.

[Aksenov et al., Euro-par 2017]

Concurrent partially-external BST. First try. Example 1.

1. Lock the state.
Check **not** `deleted`.



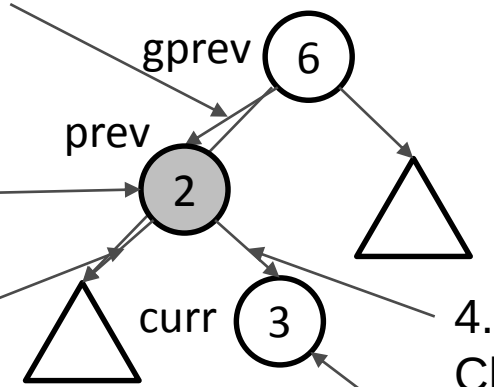
2. Lock the right edge.
Check for `null`.

Concurrent partially-external BST. First try. Example 2.

1. Lock the left child.
Check the link.

2. Lock the state.
`state == ROUTING`
Check **`not`** `deleted`.

3. Lock the left child.
Check the link.

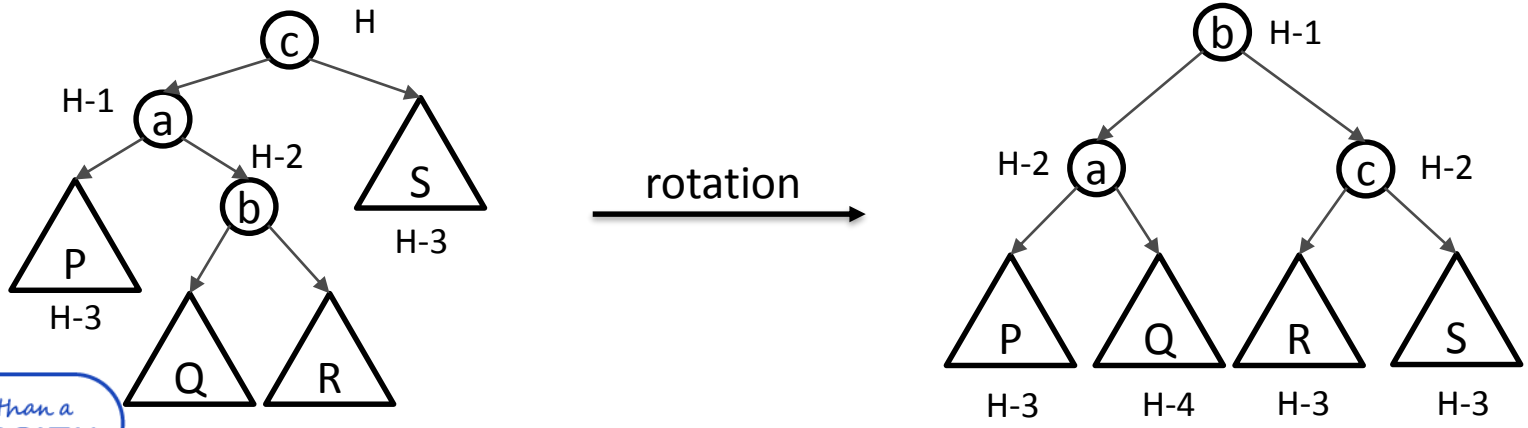


4. Lock the right child.
Check the link.

5. Lock the state.
Check **`not`** `deleted`.
Check that leaf.

What about the performance?

- The proposed algorithm works badly on skewed workloads.
- We can add a rebalancing.



What about the performance?

- Relaxed AVL-balancing [Bouge et al., Height-relaxed AVL rebalancing: …, 1998]
  - Store the height of the left and the right child.
  - Compare and rotate them on the way from the bottom to the top.
  - Correctness: the last thread will do everything good.
- Other balancing strategies:
  - Top-down. See Chromatic Tree.
  - Rebuild the whole subtree from scratch. See C-IST.

Partially-external balanced BST. Attempt 1.

[Bronson et al., A Practical Concurrent Binary Search Tree, 2010] did the first practical balanced BST, but a little bit tricky and complicated.

- `get(v)` works in a hand-over-hand locking manner, but optimistically. Runs the recursion, reads and checks. If something has changed, the algorithm revert recursion back.
- Deletions and rotations change the versions of the vertex. Sometimes the rotations do well – we use the bitmask to check that.

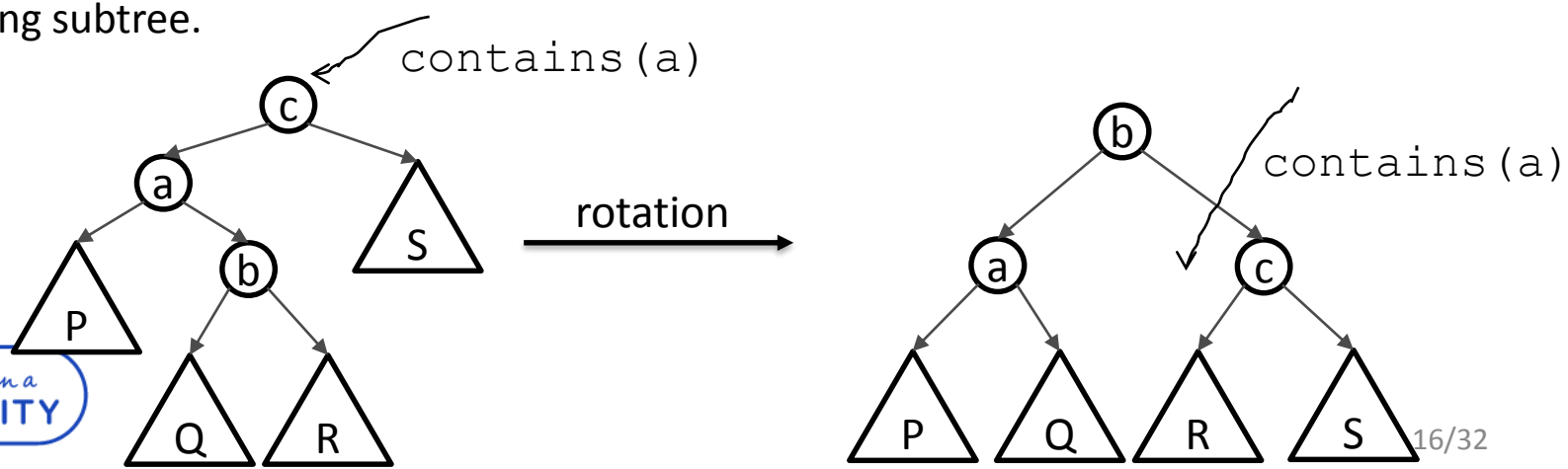
Partially-external balanced BST. Attempt 1.

The problem with the previous tree:

- `get(v)` is not wait-free…

What did stop us?

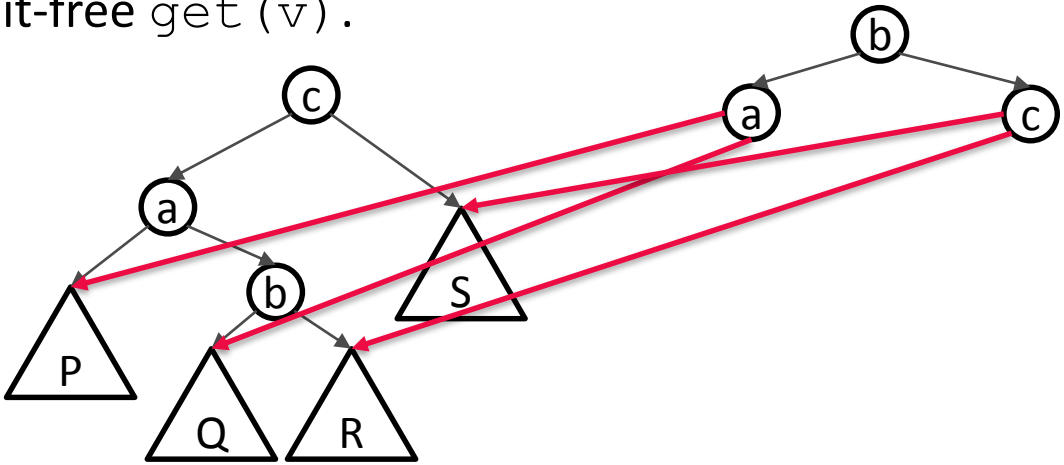
- When we are doing in wait-free manner, someone can rotate, and we can get to the wrong subtree.



Partially-external balanced BST. Attempt 2.

[Crain et al., A Contention-Friendly Binary Search Tree, 2013] proposed a simple idea: on rotation take a lock on the nodes, then copy and link them in the right order.

By that we get wait-free `get(v)`.



Partially-external balanced BST. Attempt 2.

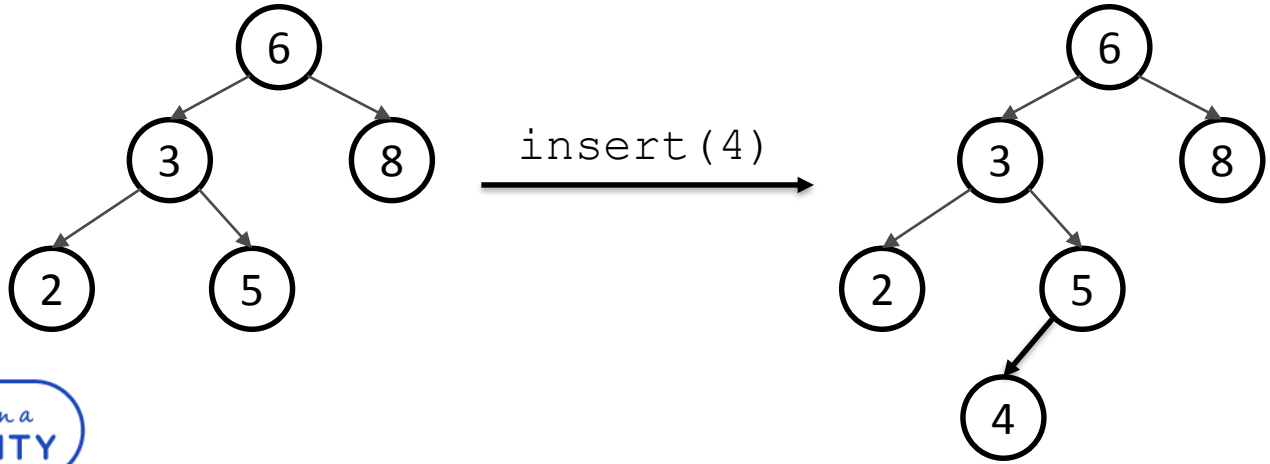
What else was proposed there?

- Main threads and a restructure-daemon.
- Main threads just mark the nodes for removal.
- Restructure-daemon maintains the invariant and rebalances.
- That division simplifies the development.

Internal BST

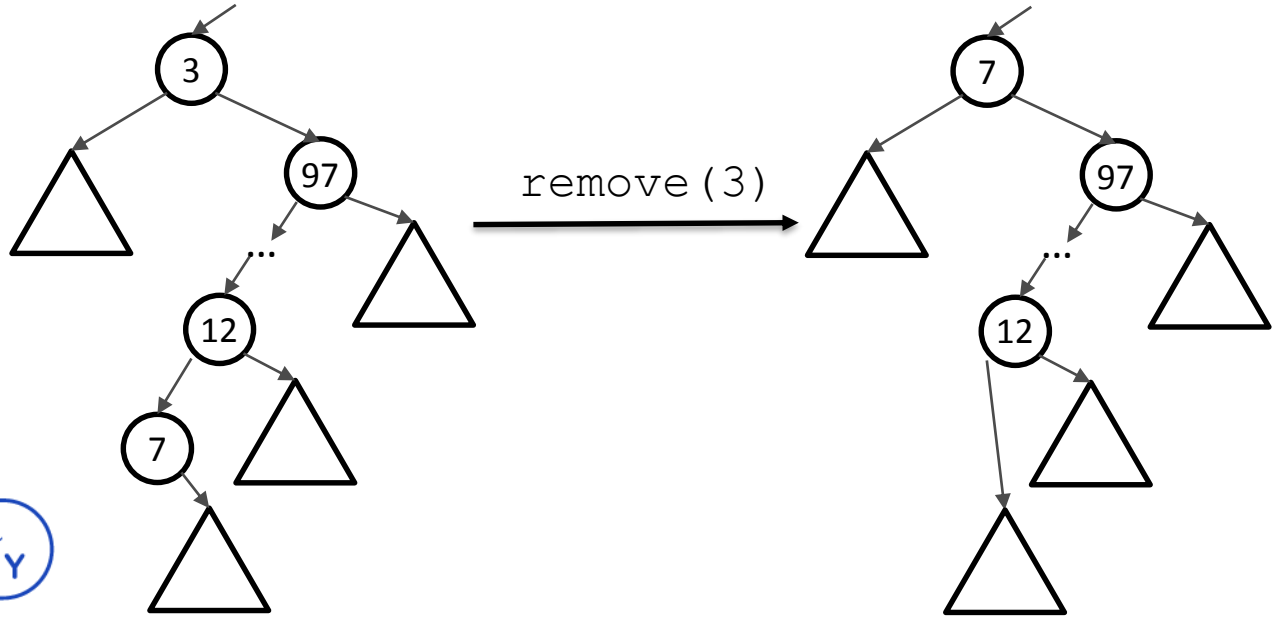
The standard binary search tree.

`insert(v)` just go by the path and insert a leaf.



Internal BST. Remove.

`remove` is not local anymore when remove a node with two children.



Internal BST. Attempt 1.

- Traverse to the target key.
  - Does not work. The keys can bubble up.
- Naïve approach:
  - lock everything on the path.
  - make hand-over-hand locking for `traverse`.
- This is very slow and inefficient.
- Also, we want the tree to be balanced…

Internal BST. Attempt 2.

[Drachsler et al., Practical Concurrent Binary Search Trees via Logical Ordering, 2014]

- `next` in O(1): make a doubly-linked list
- Free wait-free `traverse`. Get to the list and then navigate to the left and to the right.

Internal BST. Attempt 2.

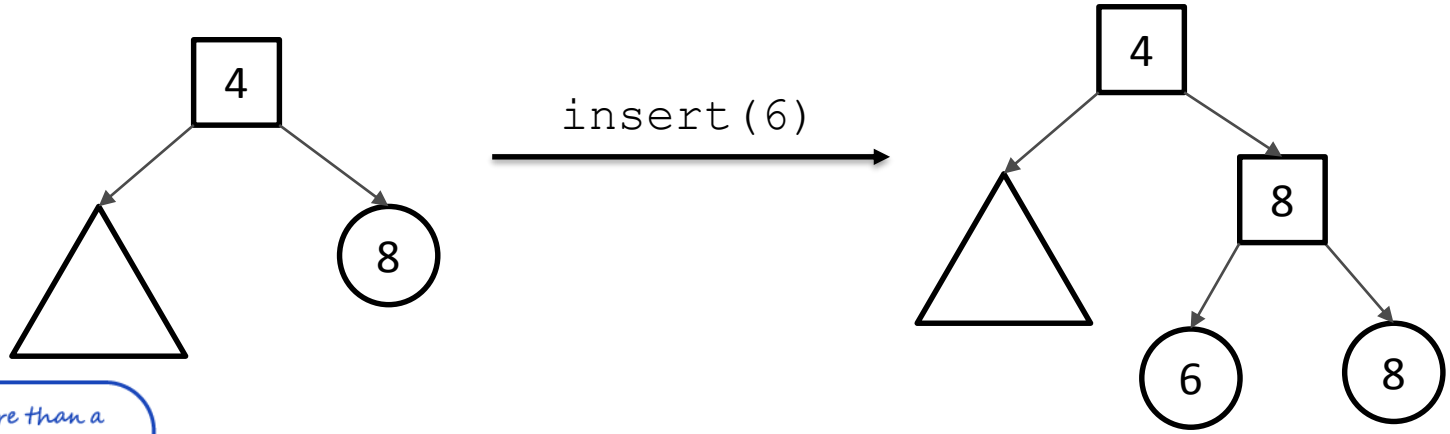
- This is enough for insert and remove.
- The rebalancing is simple – traverse will find the necessary key.
- The problem: a little bit inefficient.
- The original algorithm from the paper is not linearizable. The error was found in 2019. Do not believe papers!

Lock-free BSTs

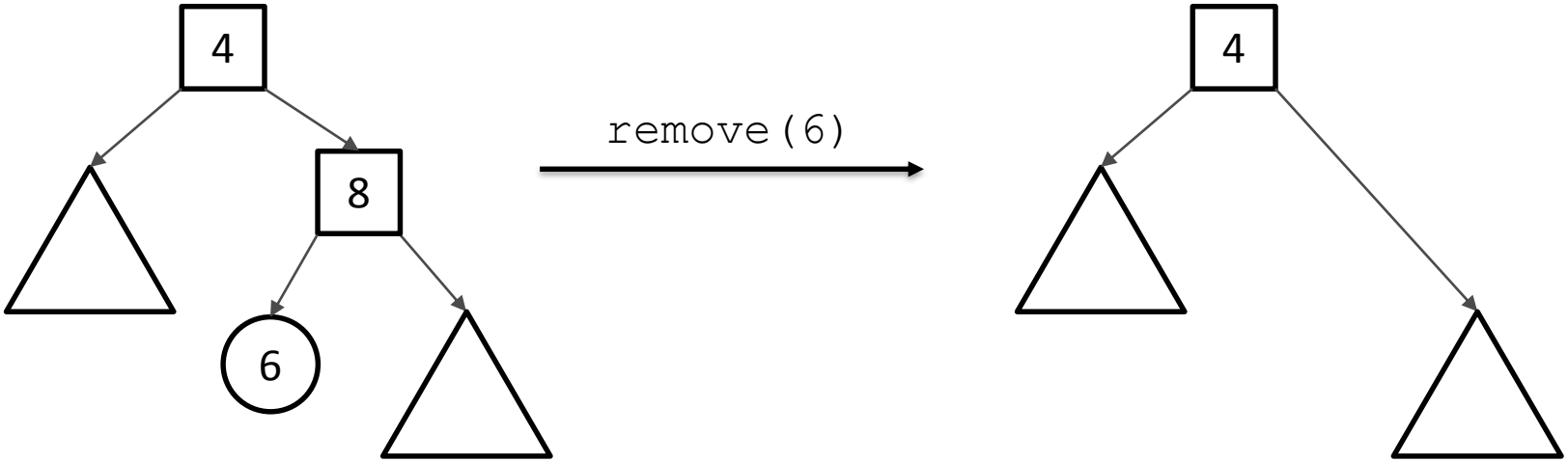
- Previously, we talked about lock-based implementations.
- Can we design lock-free BST?
- Unfortunately, internal and partially-external trees are complicated.
- We could use external trees!

External BST

- The keys are stored in the leaves. The inner nodes store the routing information.



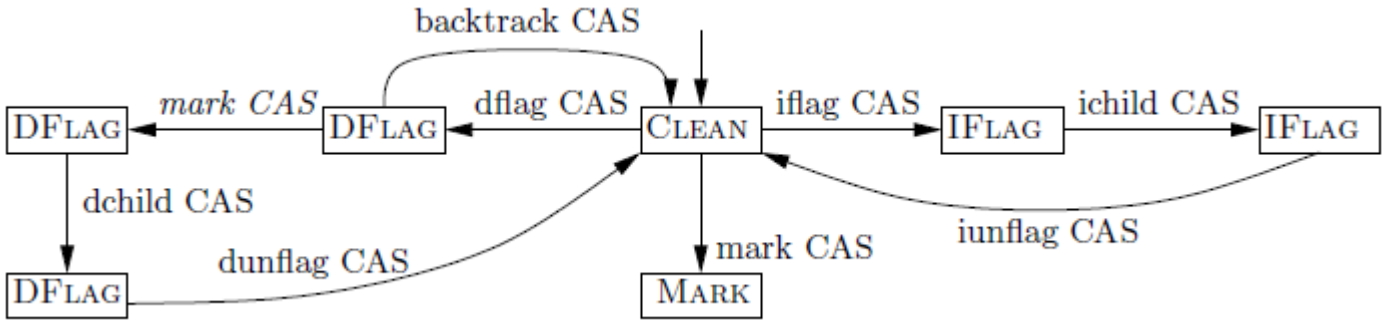
External BST. Remove.



Lock-free External BST. Attempt 1.

First attempt: [Ellen et al., Non-blocking Binary Search Trees, 2010].

- Store the descriptor at each node.
- When accessing the node, help the current descriptor, then store ours.
- However, the scheme is not simple.
- Luckily, wait-free `traverse` is not affected by `remove` and works for external trees



Lock-free External BST. Attempt 2.

- The previous algorithm have a huge overhead on the descriptors:
  - Too much operations with descriptors.
  - Memory allocation.
  - Too coarse-grained on vertices.
- [Natarajan and Mittal, Fast Concurrent Lock-Free Binary Search Trees, 2014] explained how to represent descriptors with two boolean flags on the references to children.

Lock-free Balanced BSTs

- It is hard to add rebalancing to the previous implementations.
- The ultimate approach – CASN. But it is inefficient.
- Two approaches:
  - [Brown et al., A General Technique for Non-blocking Trees, 2014] Based on External Chromatic Tree with LLX and SCX.
  - [Prokopec et al., Analysis and Evaluation of Non-Blocking Interpolation Search Trees] Collaborative rebuild after the number operations.

Wait-free BSTs

[Natarajan et al., Concurrent wait-free red black trees, 2013]

- Top-down red-black trees
- There is a window that goes down. When overlaps with another window – help it.

Performance Overview

- Skip-List?
- Only Java, no bit-descriptors: [Crain et al.]
- C++, without rebalancing: [Natarajan et al.]
- Binary Search Trees do not have high performance. Use:
  - [Srivastava et al., Elimination (a, b)-trees with fast, durable updates, 2022]
  - [Brown et al., Non-blocking interpolation search trees with doubly-logarithmic running time, 2020]

Some performance comparison on C++.

- [Arbel-Raviv et al., Getting to the Root of Concurrent Binary Search Tree Performance, 2018]

Thanks for your attention!

www.ifmo.ru

ITMOre than a UNIVERSITY