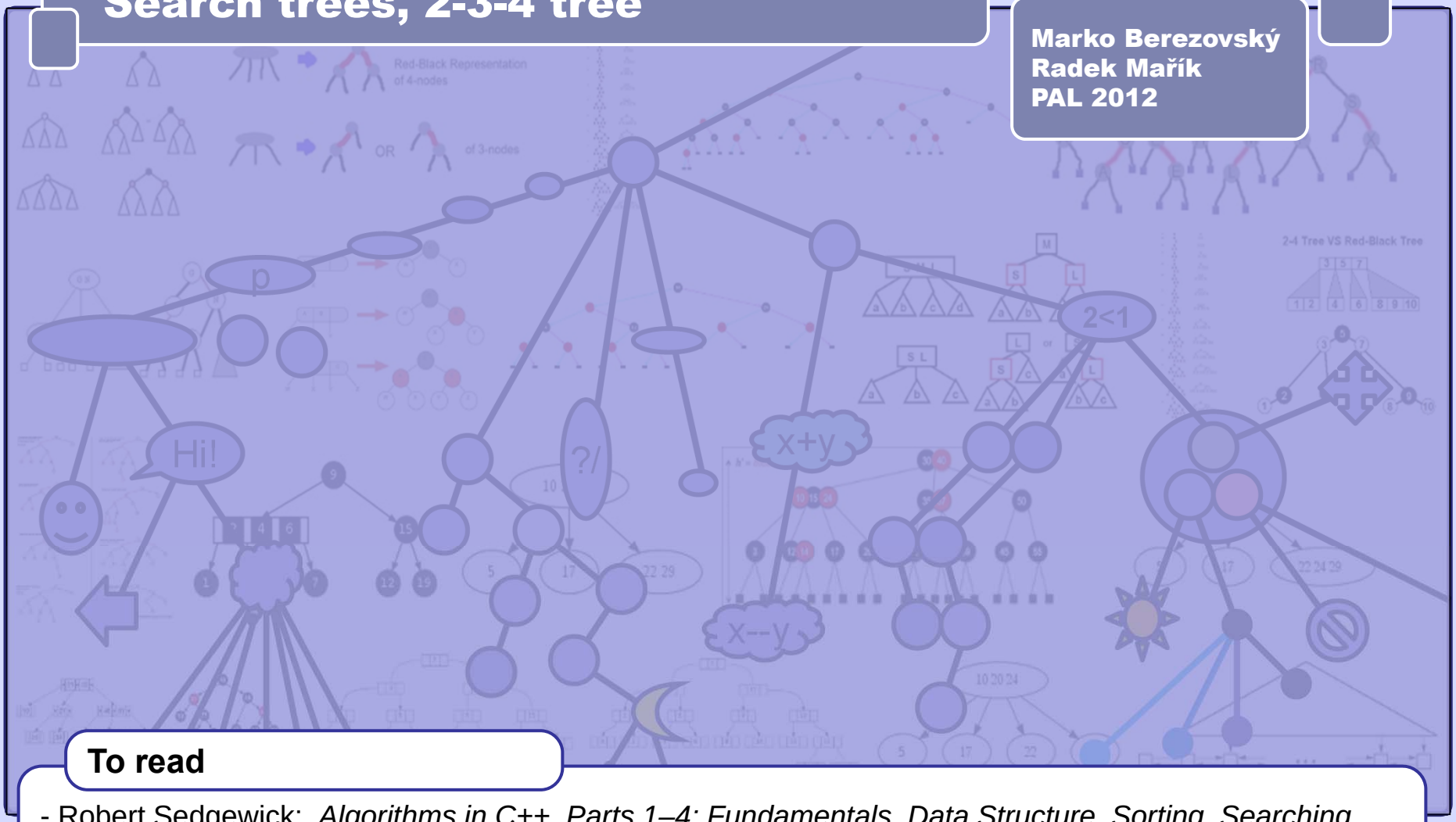
Search trees, 2-3-4 tree

Marko Berezovský
Radek Mařík
PAL 2012



To read

- Robert Sedgewick: *Algorithms in C++, Parts 1–4: Fundamentals, Data Structure, Sorting, Searching, Third Edition*, Addison Wesley Professional, 1998
- http://www.cs.helsinki.fi/u/mluukkai/tirak2010/B-tree.pdf
- (CLRS) Cormen, Leiserson, Rivest, Stein: *Introduction to Algorithms*, 3rd ed., MIT Press, 2009

See PAL webpage for references

A **2-3-4 search tree** is either empty or it contains three types of nodes:

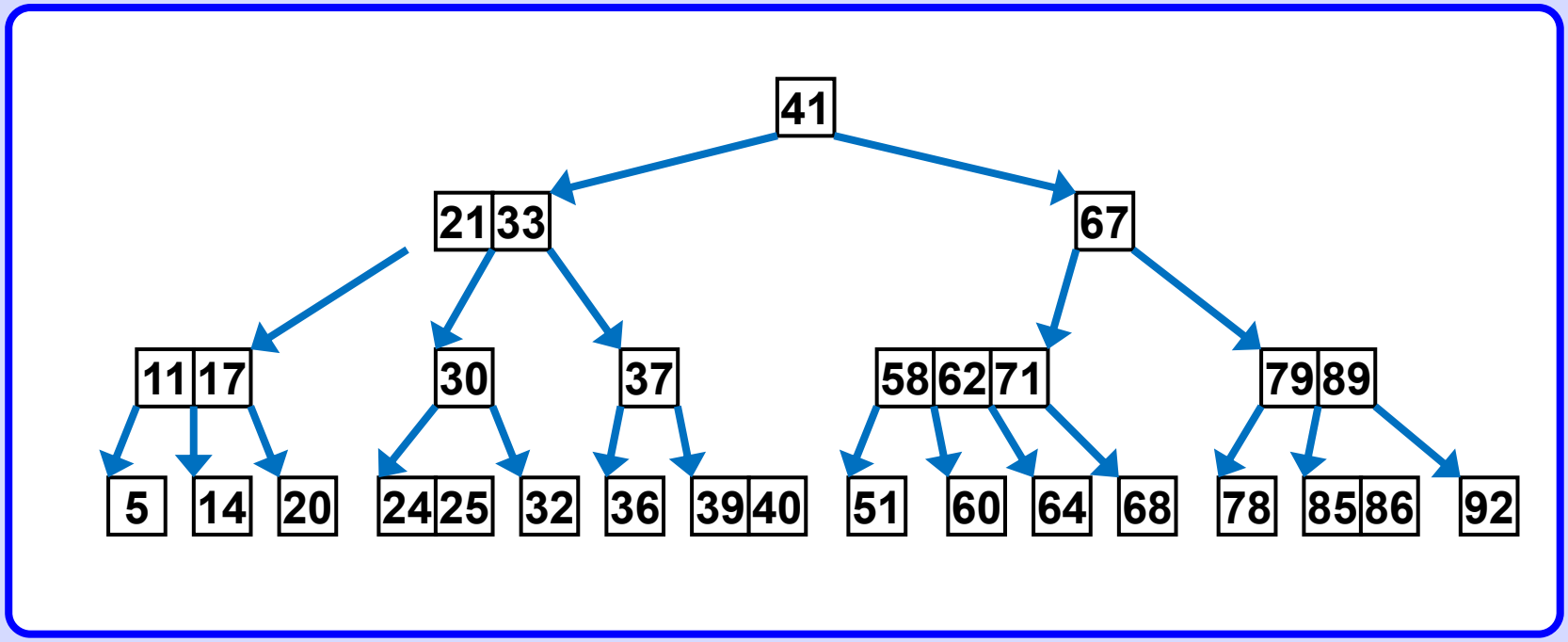
2-node, with one key, left link to a tree with smaller keys, and right link to a tree with larger keys;

3-node, with two keys, a left link to a tree with smaller keys, a middle link to a tree with key values between the node's keys and a right link to a tree with larger keys;

4-node, with three keys and four links to trees with key values defined by the ranges subtended by the node's keys.

AND: All links to empty trees, ie. all leaves, are at the same distance from the root, thus the tree is **perfectly balanced**.

A **2-3-4 search tree** is structurally a **B-tree of order 4**.



Note 2-nodes, 3-nodes, 4-node, same depth of all leaves.

Find: As in B-tree

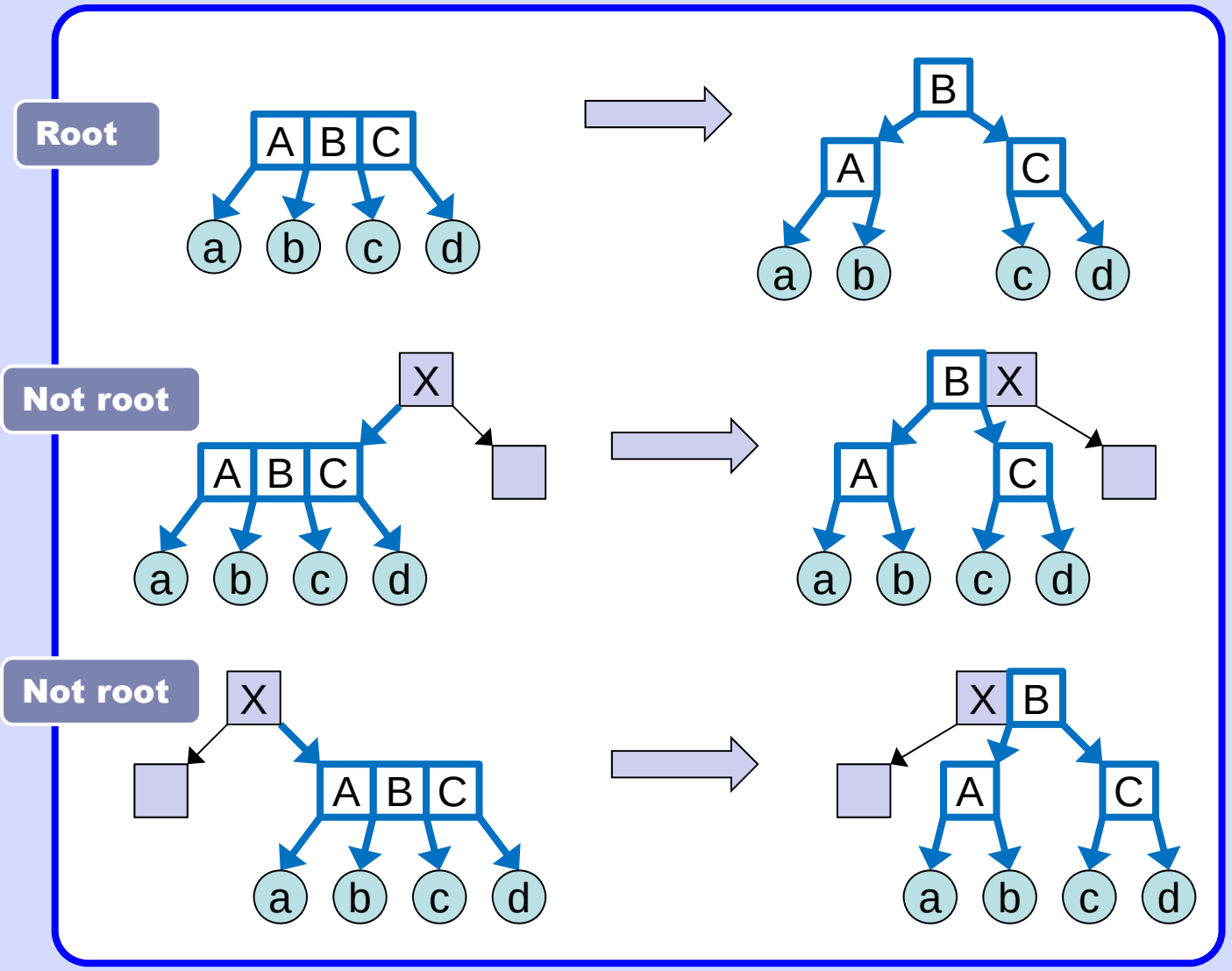
Insert: As in B-tree: Find the place for the inserted key x in a leaf and store it there. If necessary split the leaf.

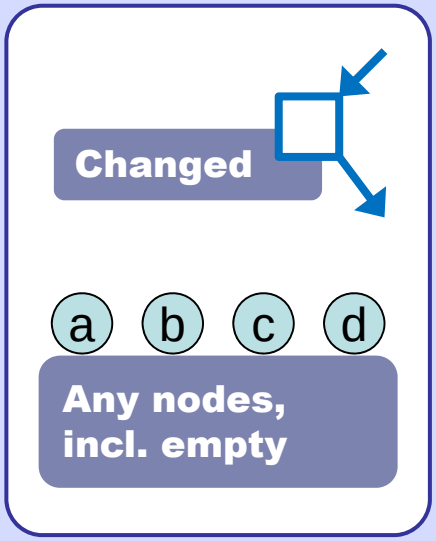
Additional **insert** rule:
In our way down the tree, whenever we reach a ***4-node***, we split it into two ***2- nodes***, and move the middle element up to the parent node.
This strategy prevents the following from happening:
After inserting a key it might happen in B tree that it is necessary to split all the nodes going from inserted key back to the root. Such outcome is considered to be time consuming.

Splitting 4-nodes on the way down results in sparse occurence of 4-nodes in the tree, thus it never happens that we have to split nodes recursively bottom-up.

Delete: As in B-tree

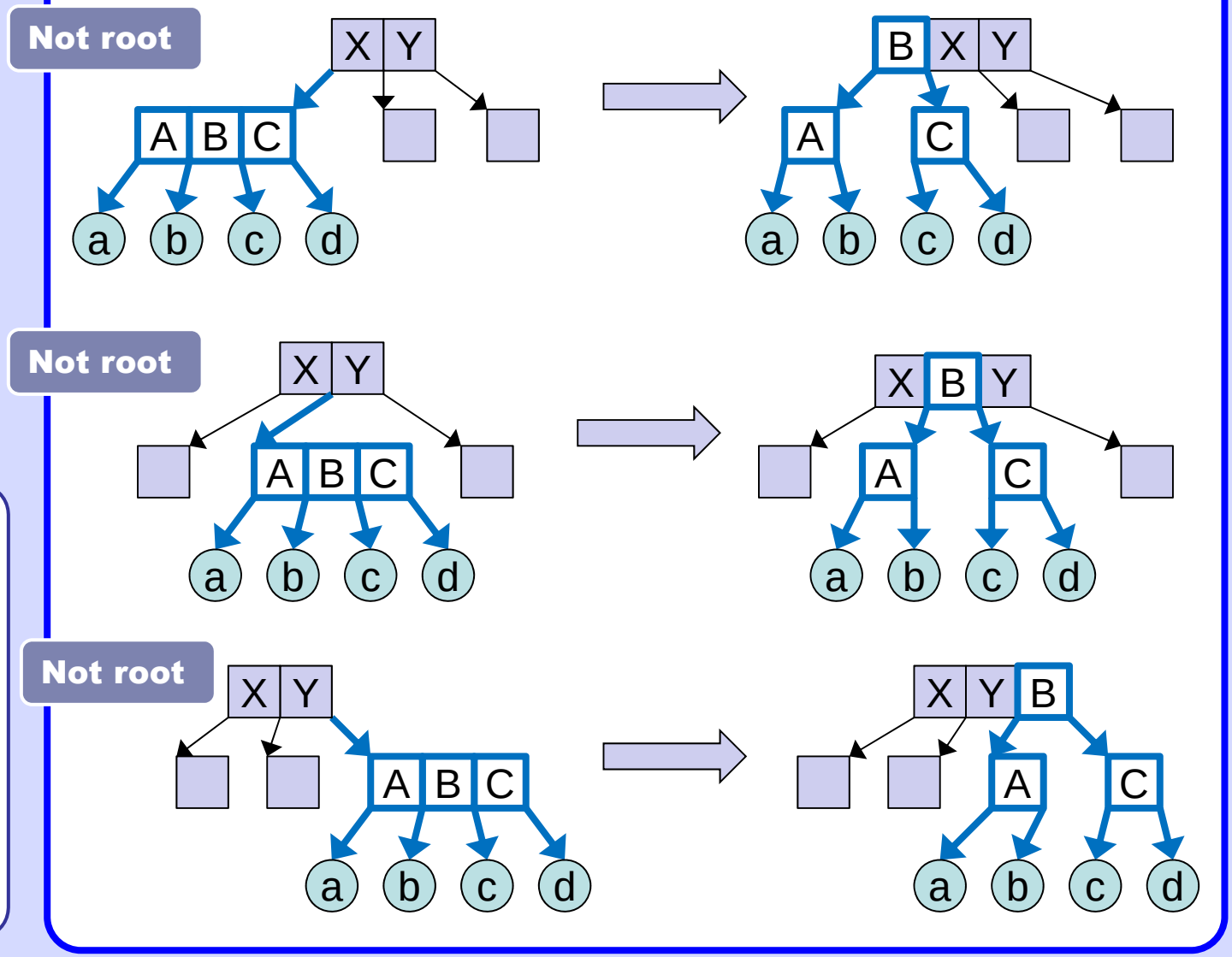
Insert: Splitting strategy

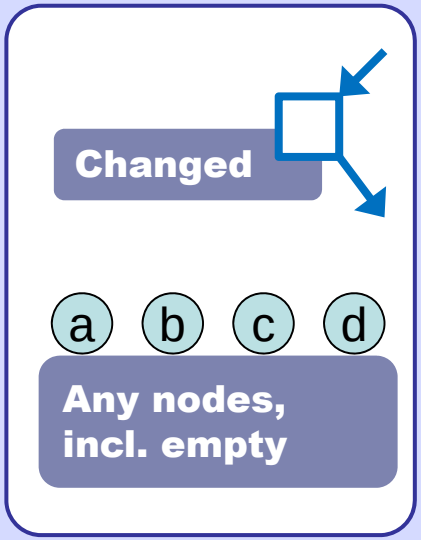




Note that splitting changes the height of the 2-3-4 tree only when the root is splitted.

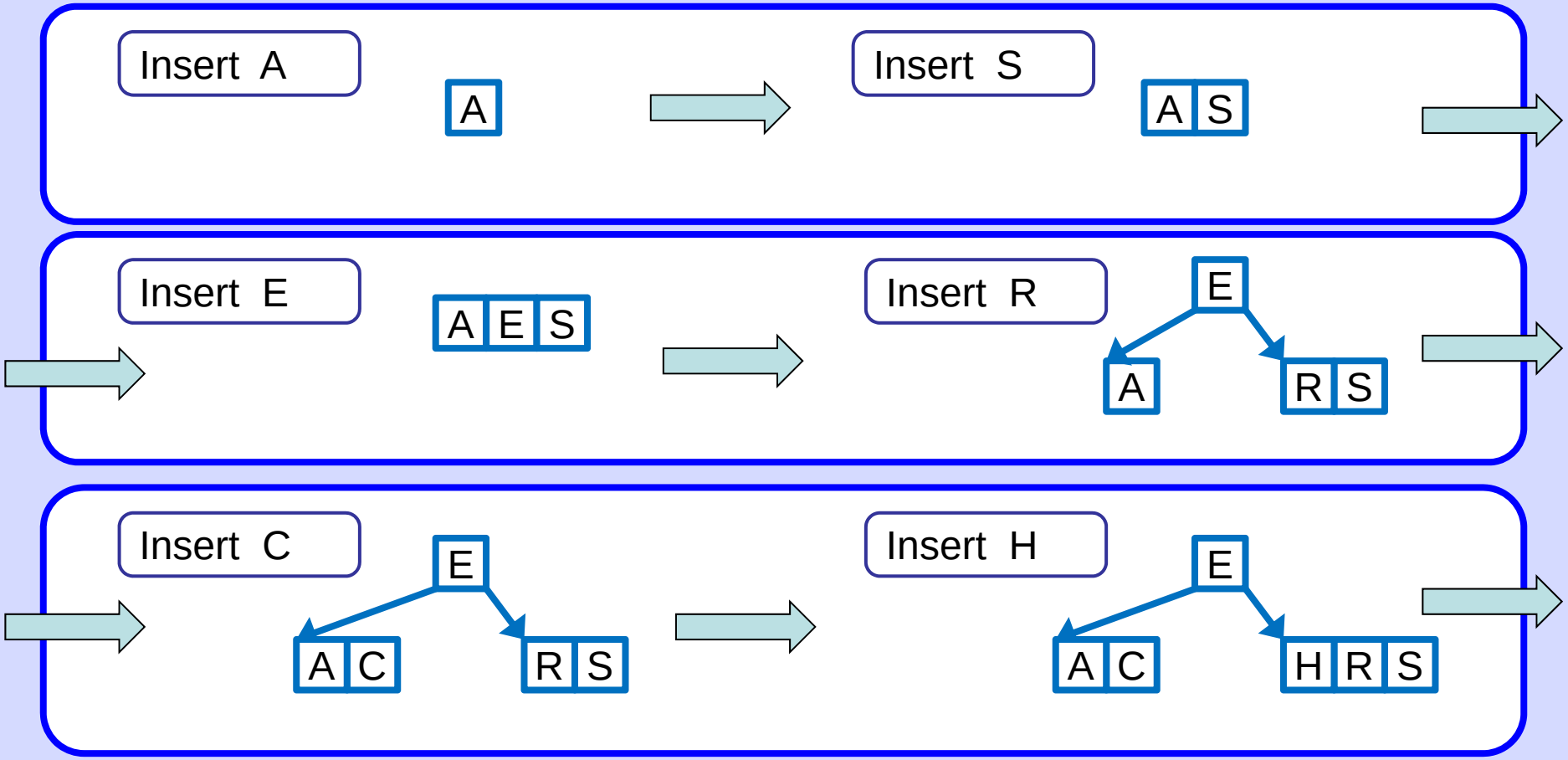
Insert: Splitting strategy



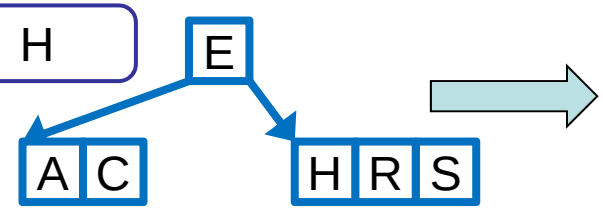


Note that splitting changes the height of the 2-3-4 tree only when the root is splitted.

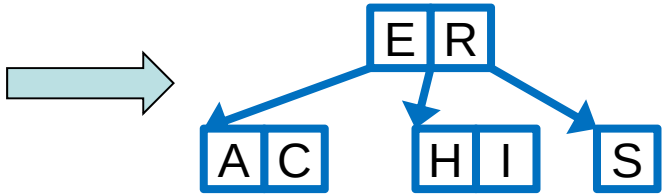
Insert keys into initially empty 2-3-4 tree: A S E R C H I N G X



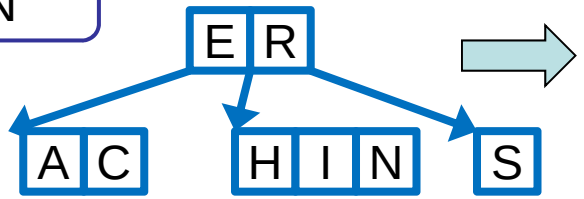
... Insert H



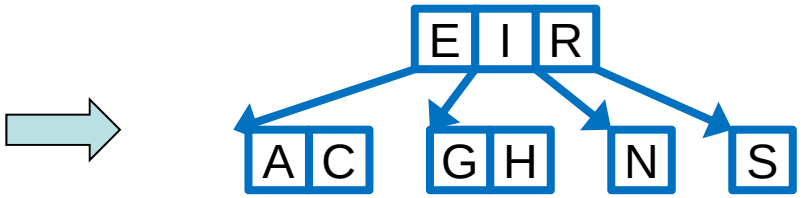
Insert I



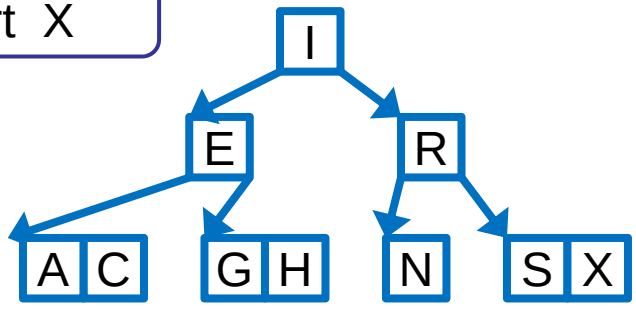
Insert N



Insert G



Insert X

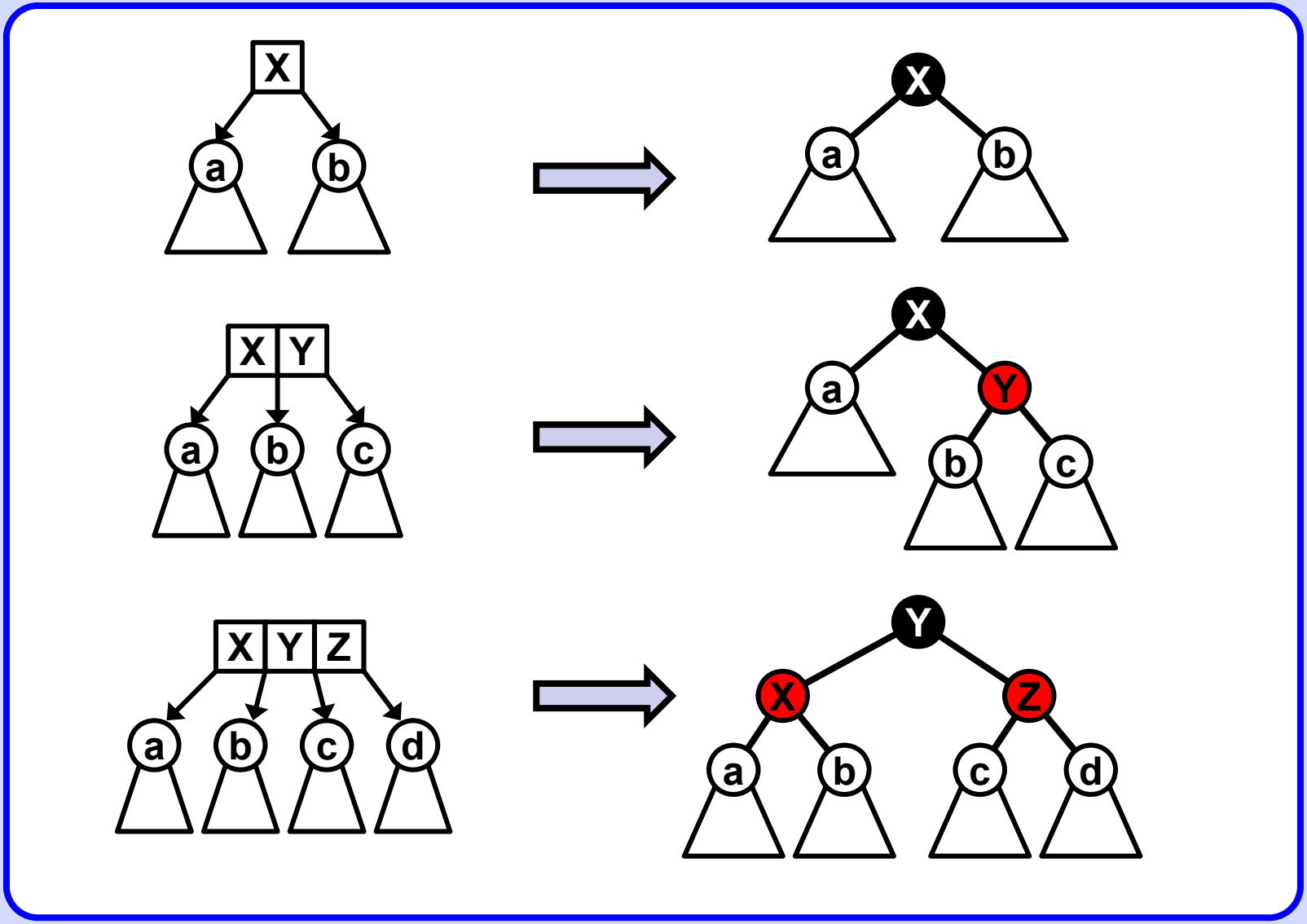


Note seemingly unnecessary split of EIR 4-node during insert of G.

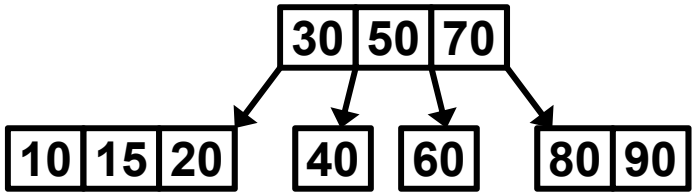
Results of an experiment with N uniformly distributed random keys from range {1, ..., 10^9 } inserted into initially empty 2-3-4 tree:

N	Tree depth	2-nodes	3-nodes	4-nodes
10	2	6	2	0
100	4	39	29	1
1000	7	414	257	24
10 000	10	4 451	2 425	233
100 000	13	43 583	24 871	2 225
1 000 000	15	434 671	248 757	22 605
10 000 000	18	4 356 849	2 485 094	224 321

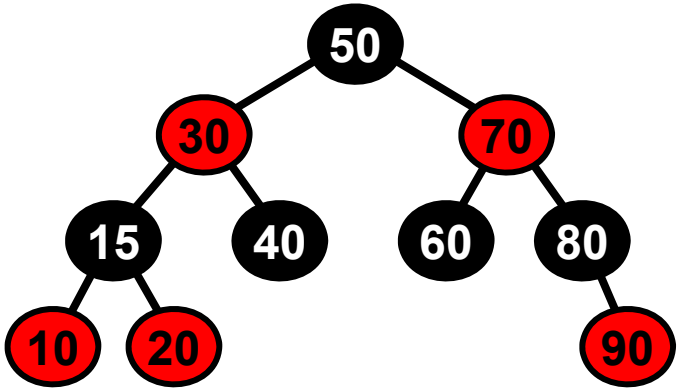
Relation of a 2-3-4 tree to a red-black tree



Relation of a 2-3-4 tree to a red-black tree



Original 2-3-4 tree



Equivalent r-b tree

Ben Pfaff: **Performance Analysis of BSTs in System Software**

Stanford University, Department of Computer Science

Conclusions:

- ...Unbalanced BSTs are best when randomly ordered input can be relied upon;
- if random ordering is the norm but occasional runs of sorted order are expected, then red-black trees should be chosen.
- On the other hand, if insertions often occur in a sorted order, AVL trees excel when later accesses tend to be random,
- and splay trees perform best when later accesses are sequential or clustered.

Some consequences:

Managing virtual memory areas in OS kernel:
... Many kernels use BSTs for keeping track of VMAs:
Linux before 2.4.10 used AVL trees, OpenBSD and later versions of Linux use red-black trees, FreeBSD uses splay trees, and so does Windows NT for its VMA equivalents...

tree / time in msec / order

Memory management supporting web browser

BST	AVL	RB	splay
15.67	3.65	3.78	2.63
4	2	3	1

Artificial uniformly random data

BST	AVL	RB	splay
1.63	1.67	1.64	1.94
1	3	2	4

Secondary peer cache tree

BST	AVL	RB	splay
3.94	4.07	3.78	7.19
2	3	1	4

Processing identifiers cross-references

BST	AVL	RB	splay
4.97	4.47	4.33	4.00
4	3	2	1