

A world map in grayscale with four colored dots (one red, one green, and two red) located in the European region. The title 'Fully Persistent B-Trees' is overlaid in large red font.

Fully Persistent B-Trees

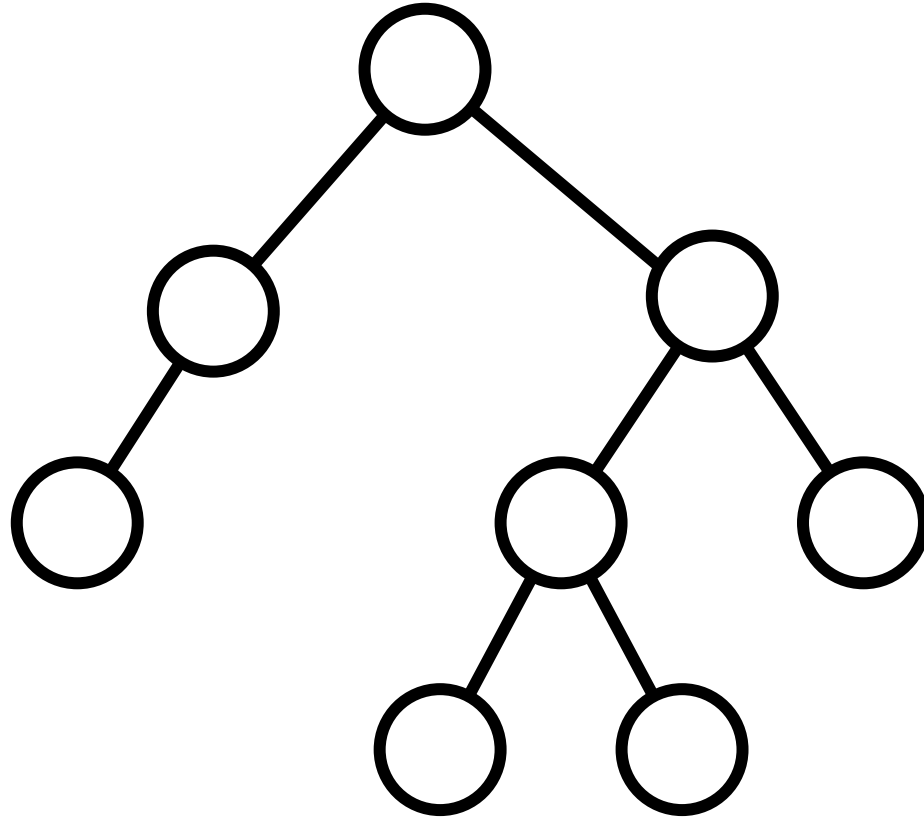
Gerth Stølting Brodal
Konstantinos Tsakalidis
Aarhus University, Denmark

madaLGO 
CENTER FOR MASSIVE DATA ALGORITHMICS

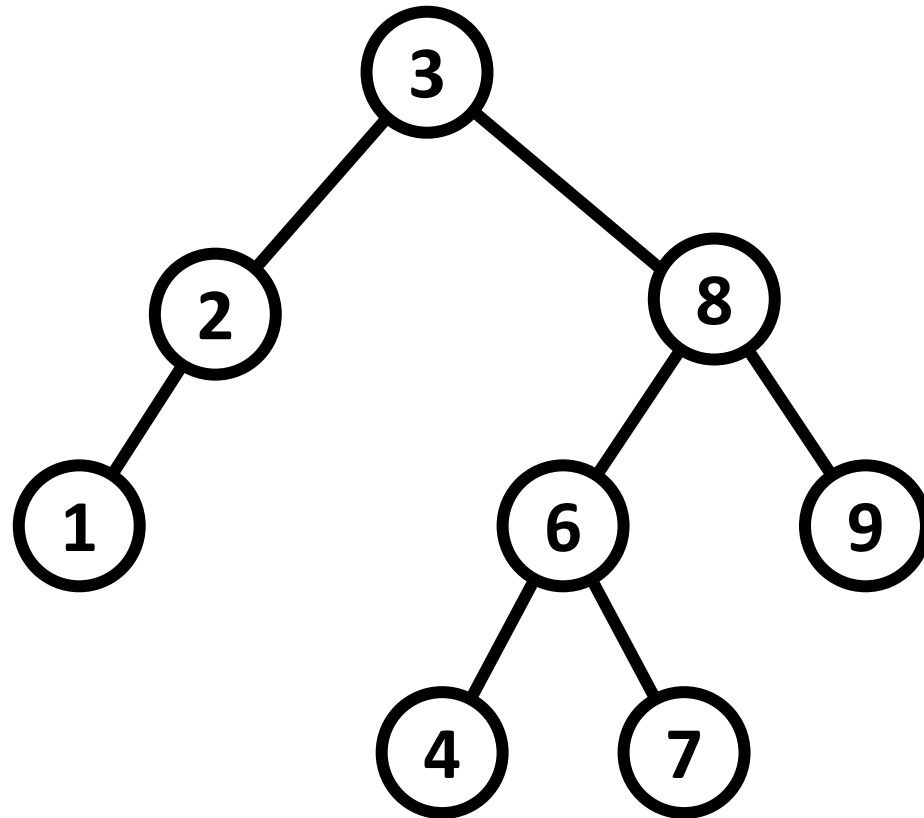
Spyros Sioutas
Ionian University, Corfu, Greece

Kostas Tsihclas
Aristotle University of Thessaloniki, Greece

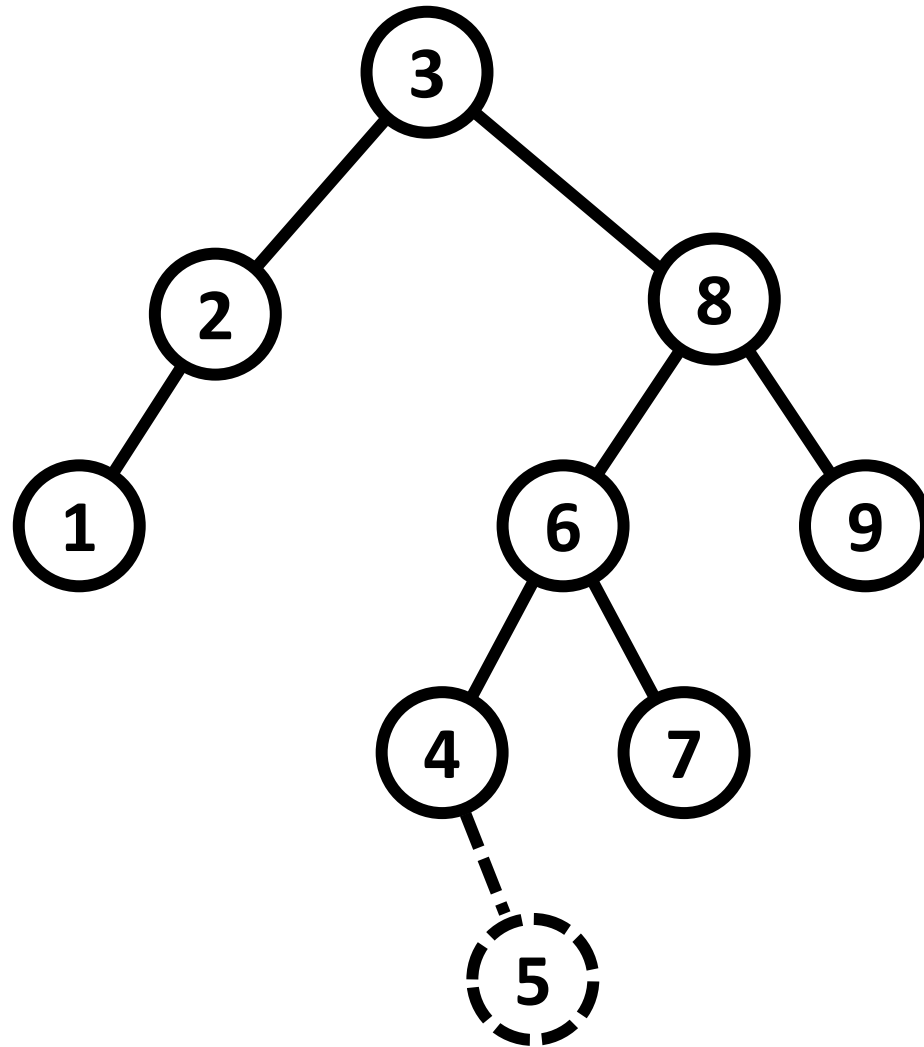
Binary Tree



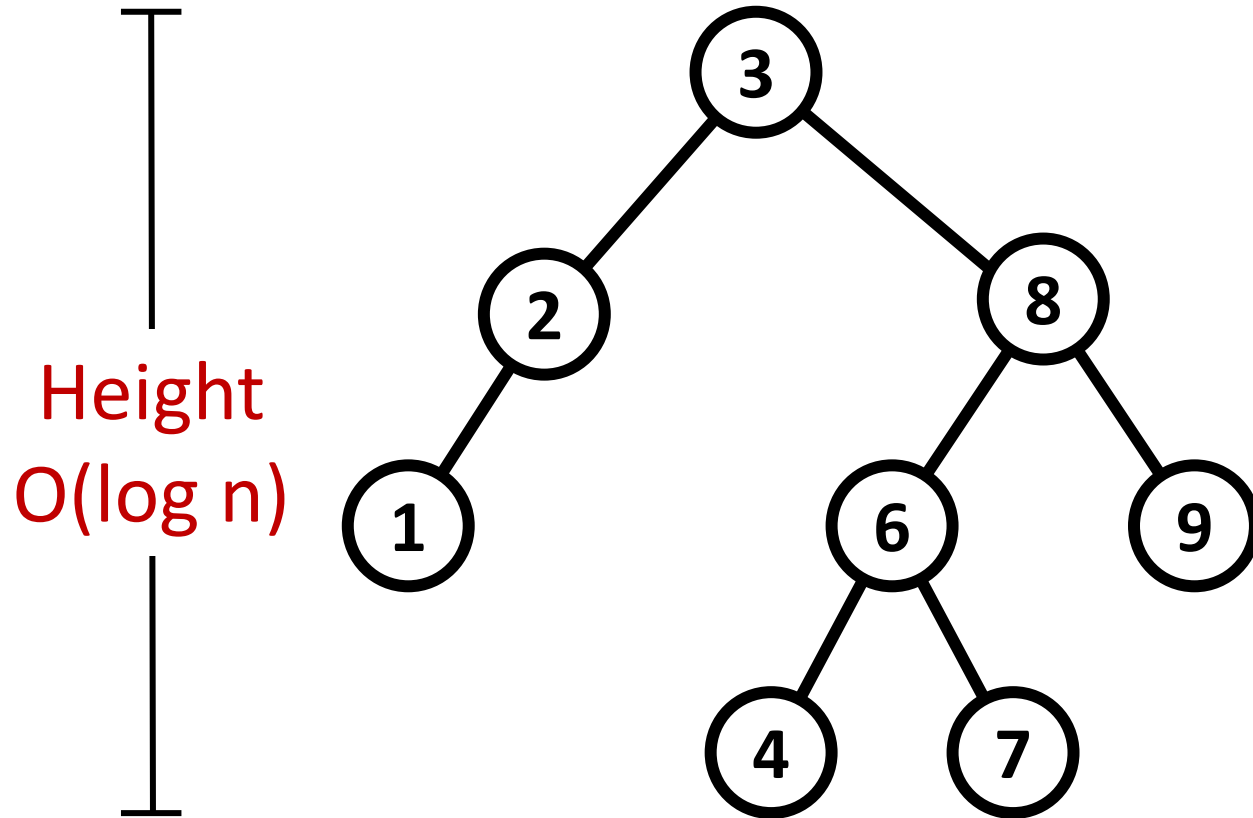
Binary Search Tree



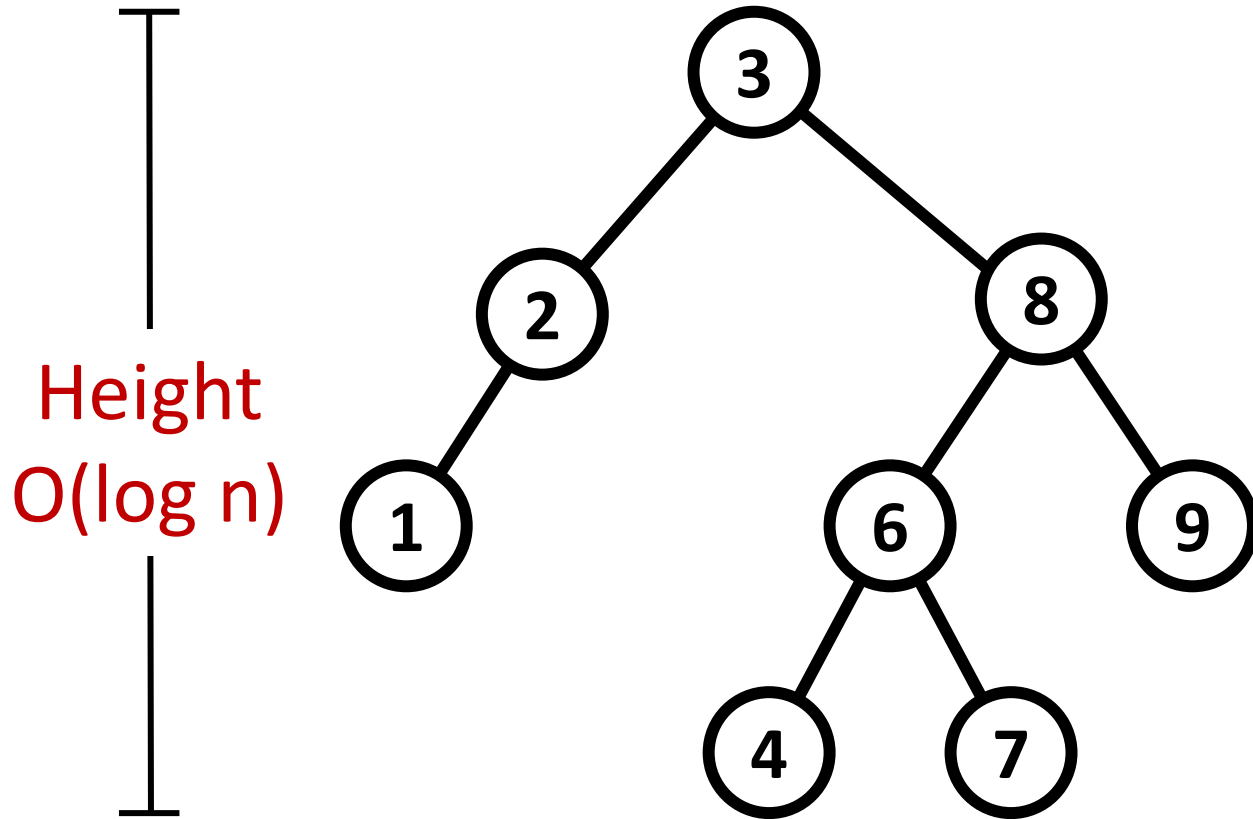
Dynamic Binary Search Tree



Balanced Binary Search Tree

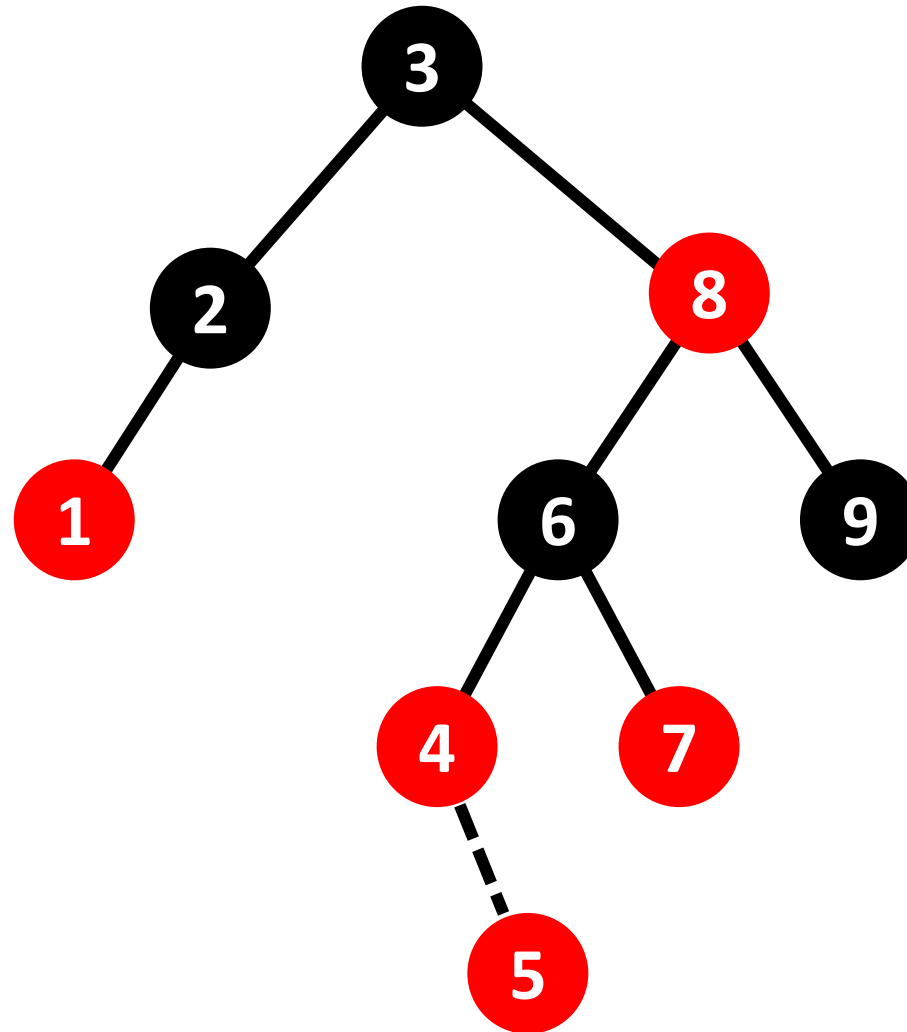


Report(2,7)

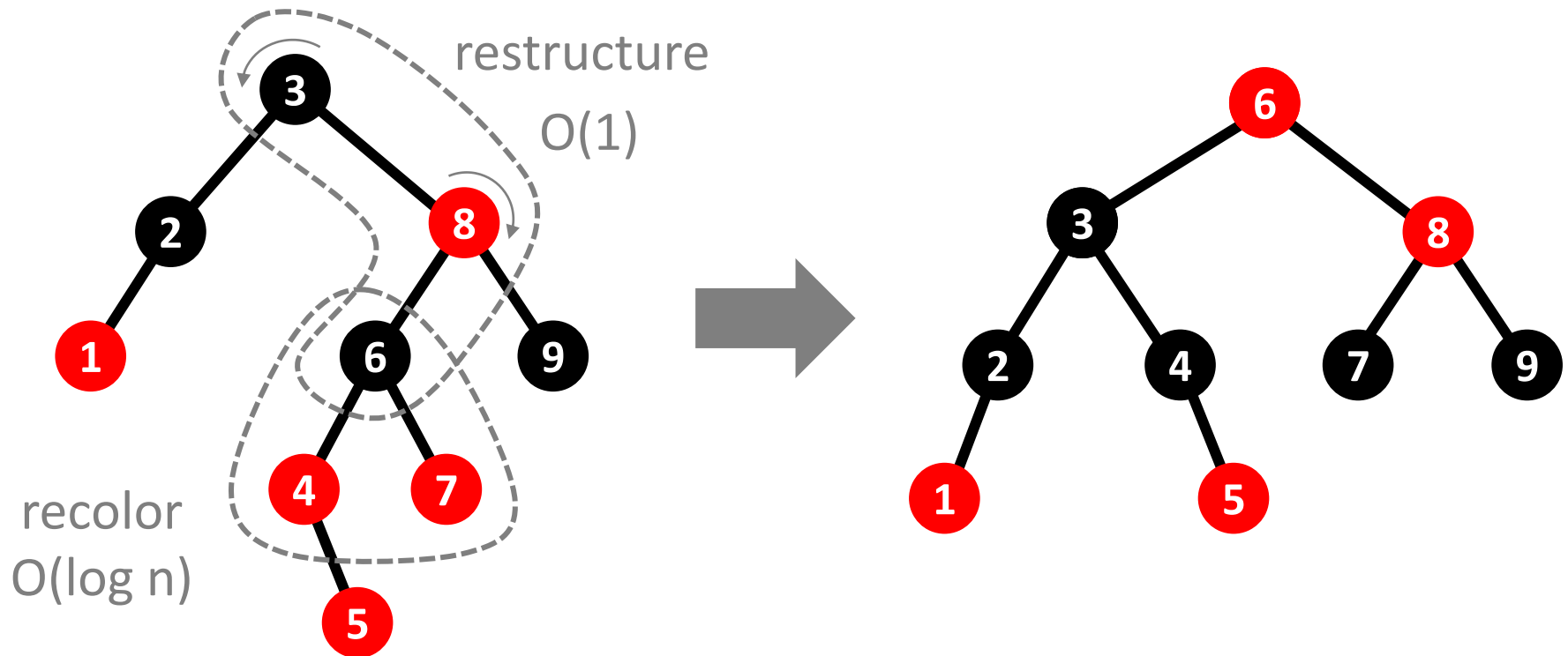


Time $O(\log n + t)$

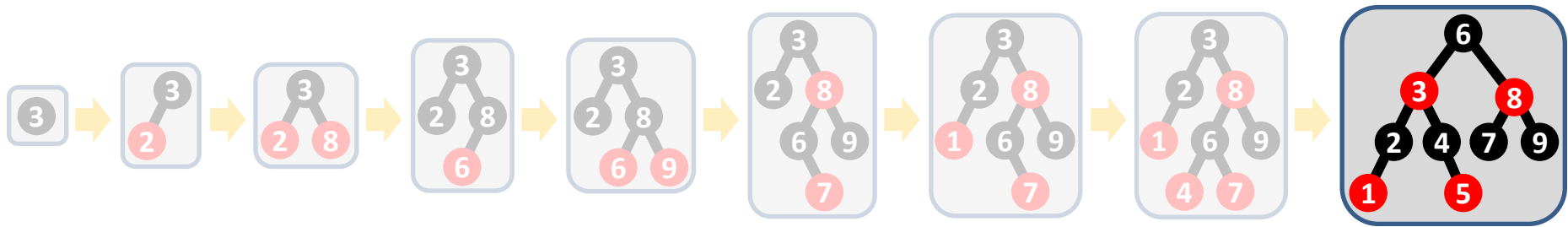
Red-Black Tree



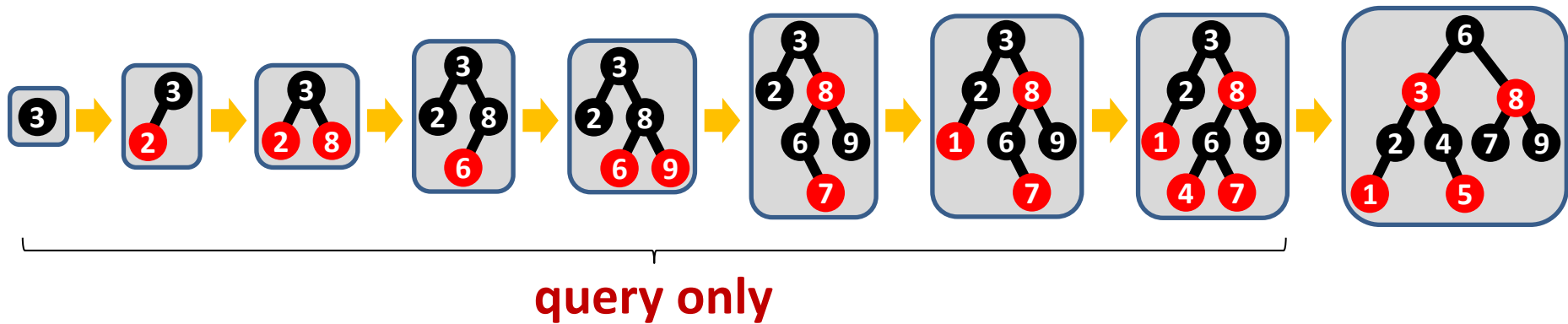
Red-Black Tree - rebalancing



Ephemeral Red-Black Tree



Partially Persistent Red-Black Tree



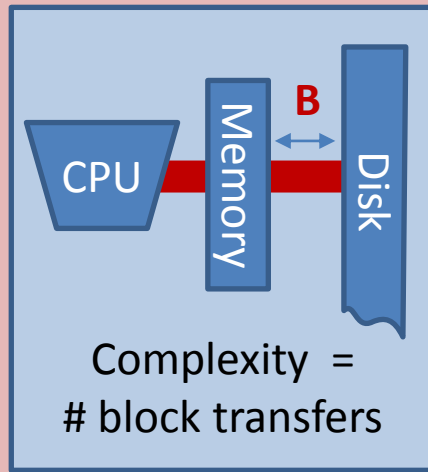
Fully Persistent **Red-Black** Tree

Contributions of Driscoll et al.

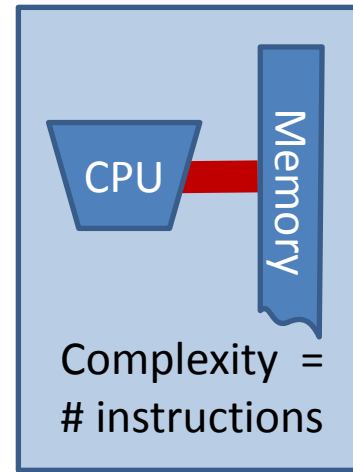
1. **Red-black** trees with $O(1)$ modifications per update (displacement paths)
2. General technique to make pointer based data structures fully persistent

Expensive to record updates performing **many modifications**

IO model



RAM model



B-Trees
Bayer 1972

Incremental B-Trees

Fully Persistent B-Trees

Maintaining Order in a List
Dietz, Sleator 1987

IO Efficient Node Splitting

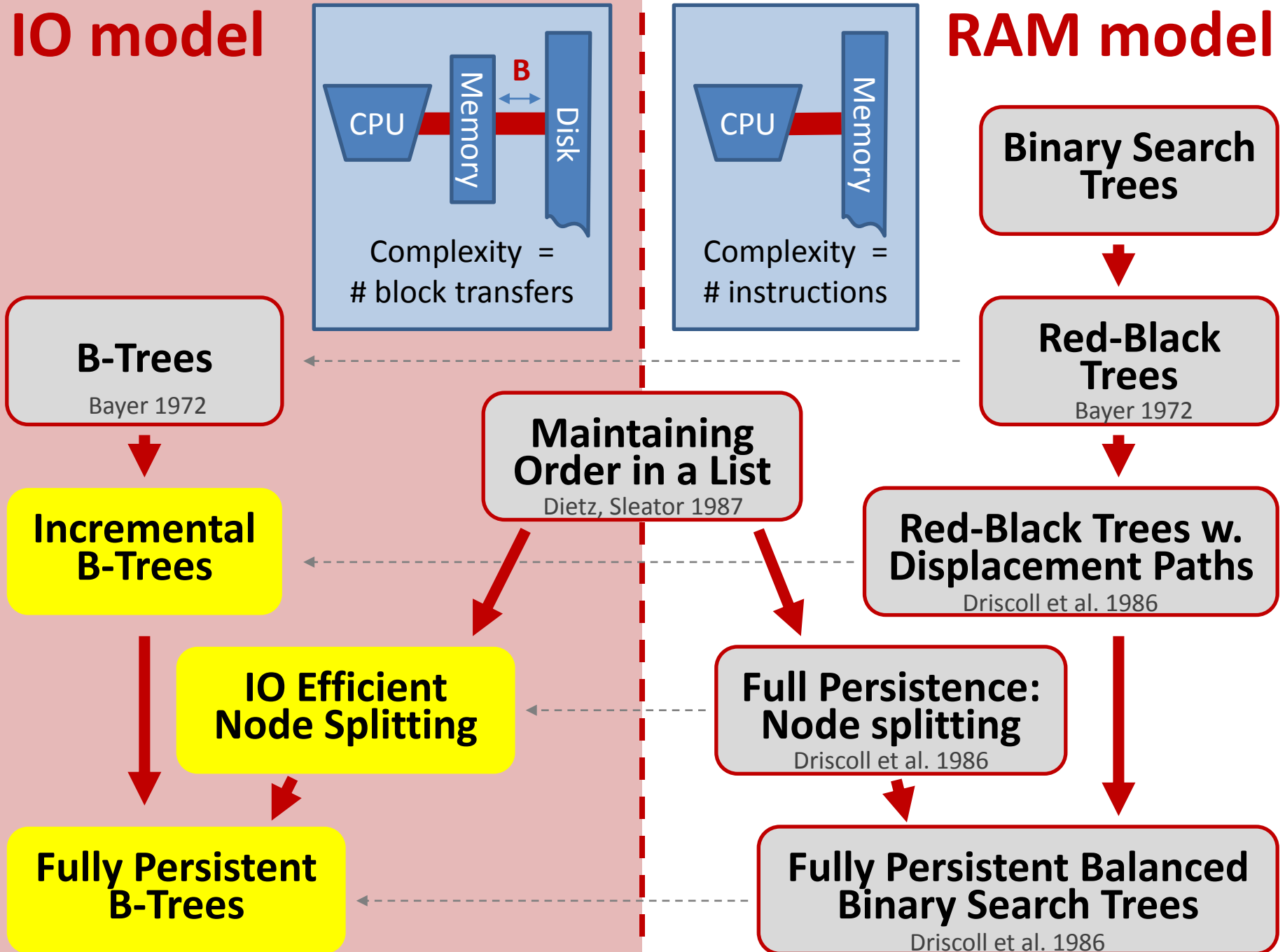
Binary Search Trees

Red-Black Trees
Bayer 1972

Red-Black Trees w. Displacement Paths
Driscoll et al. 1986

Full Persistence: Node splitting
Driscoll et al. 1986

Fully Persistent Balanced Binary Search Trees
Driscoll et al. 1986



Results

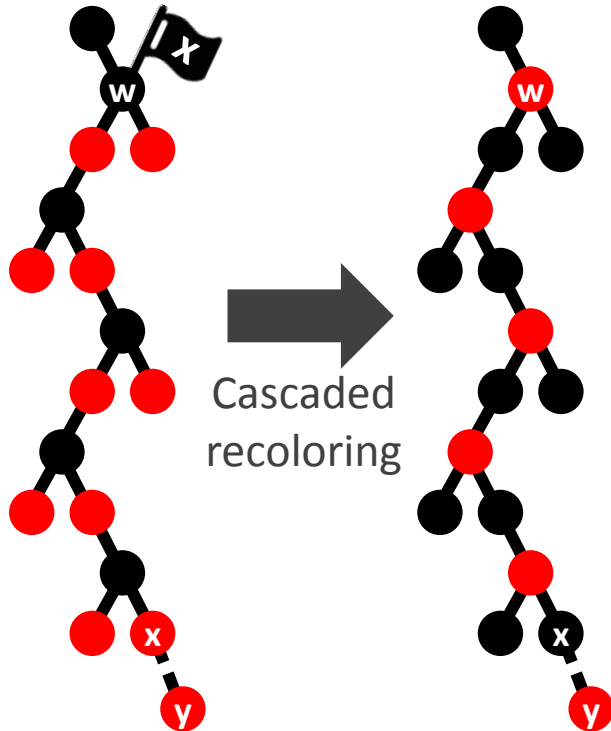
Incremental B-tree with range searches in $O(\log_B n + t/B)$ IOs, updates in $O(\log_B n)$ IOs with $O(1)$ key and pointer modifications, $O(n/B)$ space

Any **ephemeral** pointer based IO data structure with $O(1)$ indegree can be made **fully persistent** with $O(1)$ overhead per block access and $O(\log_2 B)$ IOs amortized overhead per field modification

Fully persistent B-tree with updates in $O(\log_B n + \log_2 B)$ IOs amortized, range searches in $O(\log_B n + t/B)$ IOs, $O(m/B)$ space

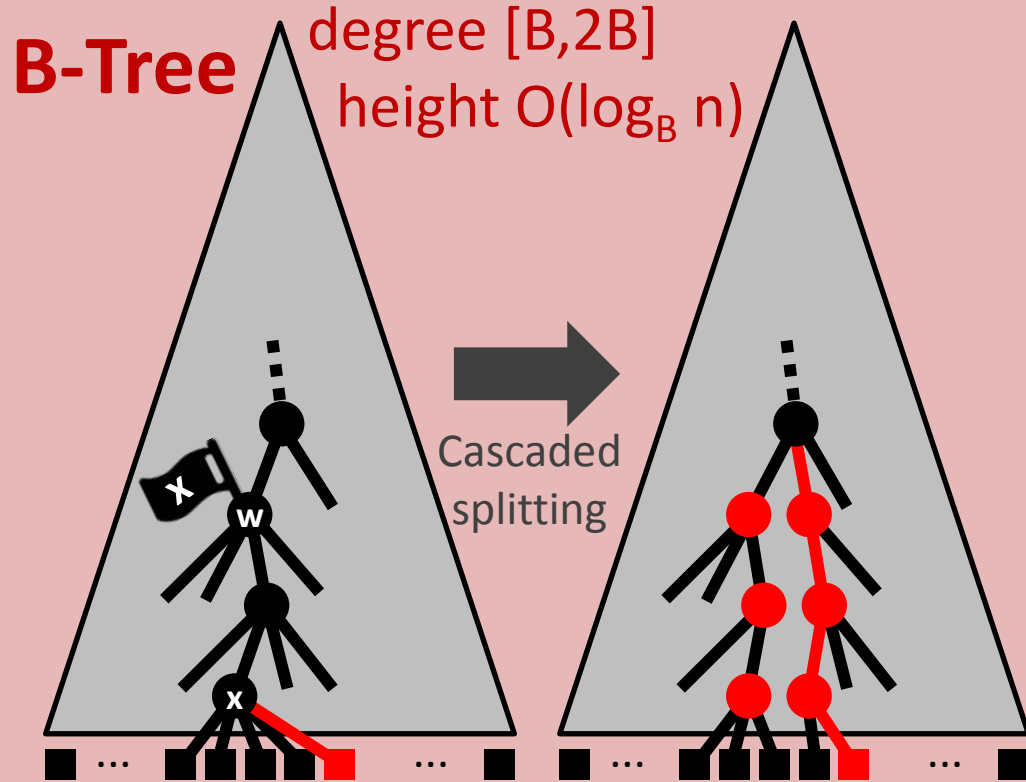
	Persistent B-tree result	Query	Update
Partial {	Arge, Danner, Teh 2003	$\log_B n + t/B$	$\log_B n$
Full {	Lanka, Mays 1991	$(\log_B n + t/B) \cdot \log_B m$	$\log_B n \cdot \log_B m$
	This talk	$\log_B n + t/B$	$\log_B n + \log_2 B$

Incremental Search Tree Updates



Displacement path $w \rightarrow x$
 \equiv flip all colors on path + all red siblings (flag w with x)

Driscoll et al. 1986



Incremental Search Tree Updates

B-Tree

degree $[B, 2B]$

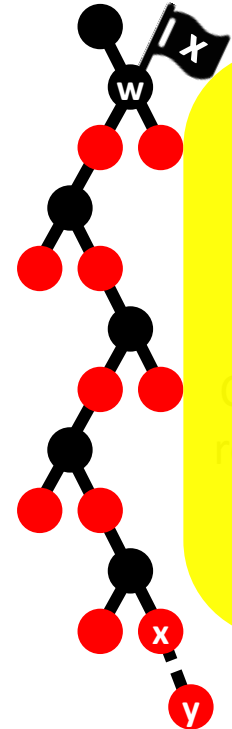
height $O(\log_B n)$

Incremental B-tree

Range searches visit $O(\log_B n + t/B)$ nodes.

Updates...

1. visit $O(\log_B n)$ nodes
2. modify $O(1)$ pointers and keys
3. creates $O(1)$ new empty nodes



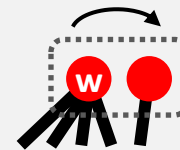
Displacement path $w \rightarrow x$

\equiv flip all colors on path + all red siblings (flag w with x)

Driscoll et al. 1986



Overflow flag
(path $w \rightarrow x$)
Counts twice



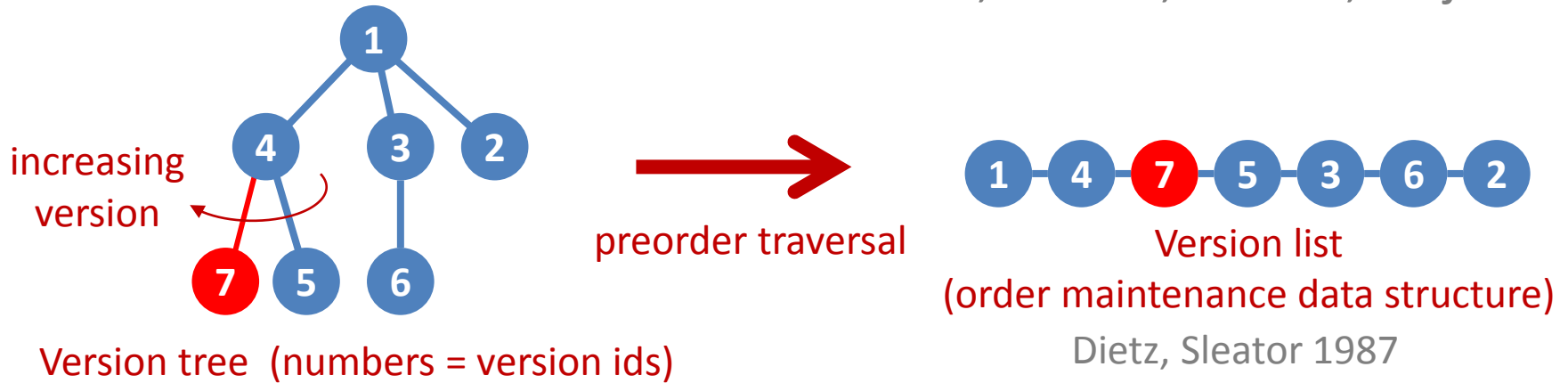
Splitting pair



Ready to overflow

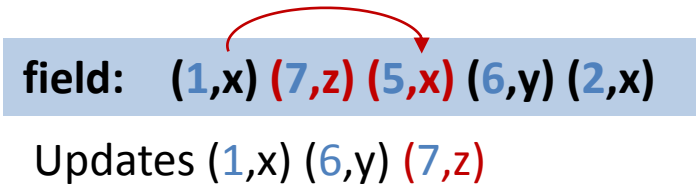
Full persistence : Node Splitting Technique

Driscoll, Sarnak, Sleator, Tarjan 1986

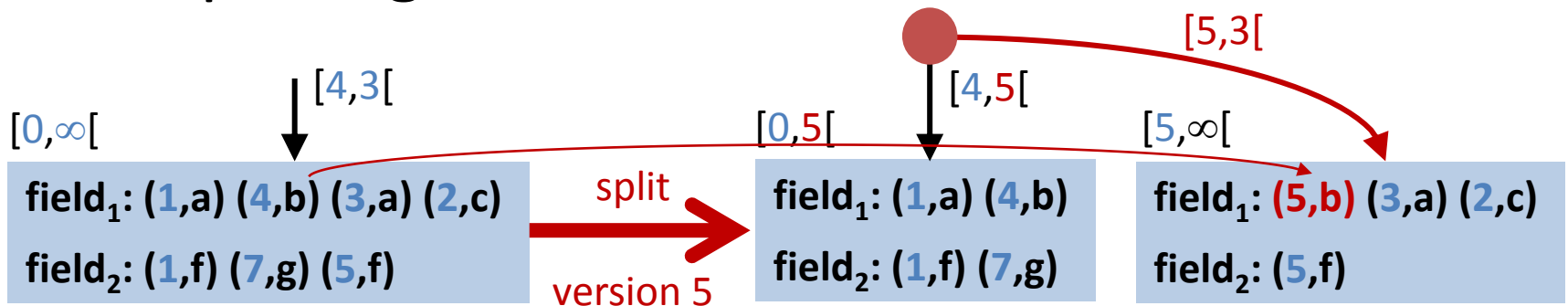


Fat node

Queries = binary search among versions



Node splitting (# ekstra fields $\geq 2 \cdot \text{indegree}$)



Full persistence : Node Splitting Technique

Driscoll, Sarnak, Sleator, Tarjan 1986



1. $O(B)$ fields & updates in a node

2. local version list **LVL** of all version ids in block (2,x)

3. each in-pointer spans $O(1)$ versions in **LVL**

Node splitting (# ekstra fields ≥ 2 indegree)

$[0, \infty[$
field₁: (1, a)
field₂: (1, f)

Bottleneck Insert **5** in ancestor(s) LVL lists using binary search in the LVL

$[4, 5[$
: (1, a) (4, b)
: (1, f) (7, g)

$[5, 3[$
 $[5, \infty[$
field₁: (5, b) (3, a) (2, c)
field₂: (5, f)

Main result

Fully persistent B-tree with updates in $O(\log_B n + \log_2 B)$ IOs, searches in $O(\log_B n + t/B)$ IOs, using space $O(m/B)$

?

Thank You