

# Concurrent Red-Black Trees

Franck van Breugel

DisCoVeri Group, Department of Computer Science and Engineering  
York University, Toronto

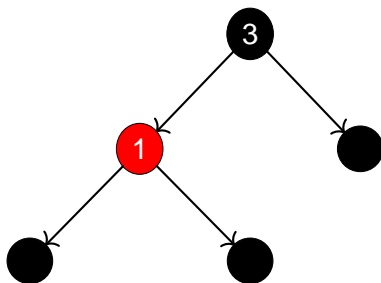
January 26, 2010

# Red-Black Tree

A *red-black tree* is a binary search tree the nodes of which are coloured either red or black and

- the root is black,
- every leaf is black,
- if a node is red, then both its children are black,
- for every node, every path from that node to a leaf contains the same number of black nodes.

[Bayer, 1972] and [Guibas and Sedgewick, 1978]



## Theorem

*A red-black tree with  $n$  internal nodes has height at most  $2 \log_2(n + 1)$ .*

## Corollary

*The SET operations ADD and CONTAINS can be implemented in  $O(\log_2(n))$ .*

The class `java.util.TreeSet`

```
1 class TreeSet<T>  
2 {  
3     boolean add(T element)  
4     boolean contains(T element)  
5     ...  
6 }
```

has been implemented by means of a red-black tree.

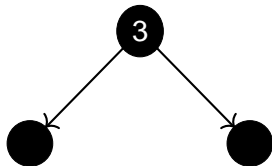
This implementation is **not** synchronized.

# Concurrent Operations on a Red-Black Tree

```
1 add(3);  
2 add(1);  
3 (add(2) || print(contains(1)))
```

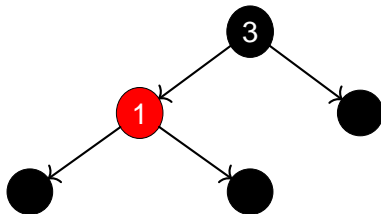
# Concurrent Operations on a Red-Black Tree

1 `add(3);`



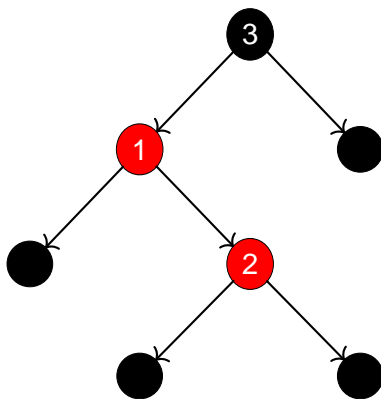
# Concurrent Operations on a Red-Black Tree

```
1 add(3);  
2 add(1);
```



# Concurrent Operations on a Red-Black Tree

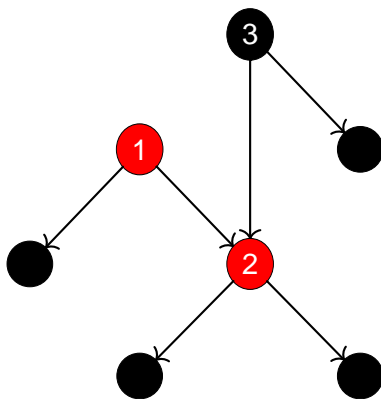
```
1 add(3);  
2 add(1);  
3 (add(2) || print(contains(1)))
```





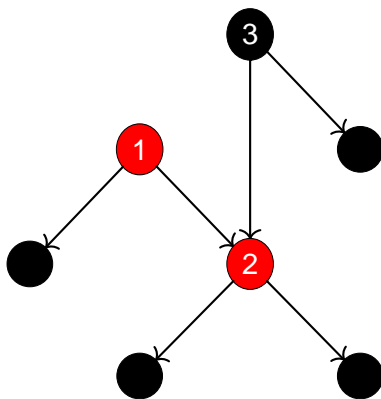
# Concurrent Operations on a Red-Black Tree

```
1 add(3);  
2 add(1);  
3 (add(2) || print(contains(1)))
```



# Concurrent Operations on a Red-Black Tree

```
1 add(3);  
2 add(1);  
3 (add(2) || print(contains(1)))
```



# Concurrent Red-Black Trees

With the arrival of multicore machines, implementations of data structures such as Set should support concurrency.

In the remainder of this talk, three concurrent implementations of red-black trees are presented.

# The Monitor Solution

```
1 RedBlackTree : monitor
2 begin
3   procedure add(element : int ,
4                 result added : boolean)
5   procedure contains(element : int ,
6                     result contains : boolean)
7 end
```

# The Readers-Writers Solution

The processes of the first class, named *writers*, must have exclusive access, and the processes of the second class, the *readers*, may share the resource with an unlimited number of other readers.

# The Readers-Writers Solution

The processes of the first class, those that call `ADD`, must have exclusive access, and the processes of the second class, those that call `CONTAINS`, may share the red-black tree with an unlimited number of such processes.

# The Readers-Writers Solution

```
1 contains(element : int) : boolean
2 [manipulate shared variables, wait]
3 manipulate red-black tree
4 [manipulate shared variables, signal]
```

Carla Schlatter Ellis. Concurrent Search and Insertion in AVL Trees. *IEEE Transactions on Computers*, 29(9):811–817, September 1980.

Carla Schlatter Ellis. *The Design and Evaluation of Algorithms for Parallel Processing*. PhD thesis, University of Washington, Seattle, 1979.



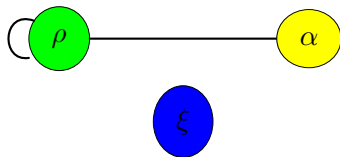


# The Main Idea

Processes lock the nodes of the red-black tree in three different ways:

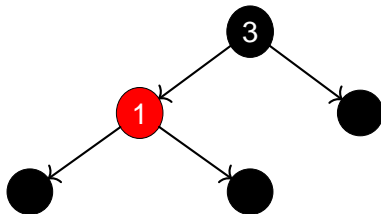
- $\rho$ -lock: lock to read
- $\alpha$ -lock: lock to exclude writers
- $\xi$ -lock: exclusive lock

Although a node can be locked by multiple processes, there are some restrictions.



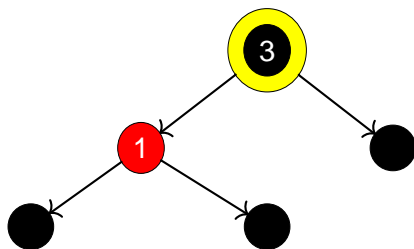
# Example Revisited

```
1 add(3);  
2 add(1);
```



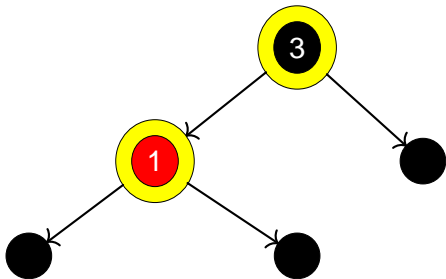
# Example Revisited

```
1 add(3);  
2 add(1);  
3 (add(2) || print(contains(1)))
```



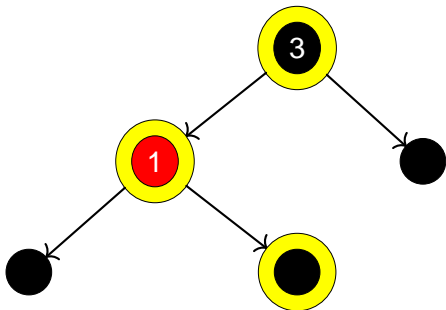
# Example Revisited

```
1 add(3);  
2 add(1);  
3 (add(2) || print(contains(1)))
```



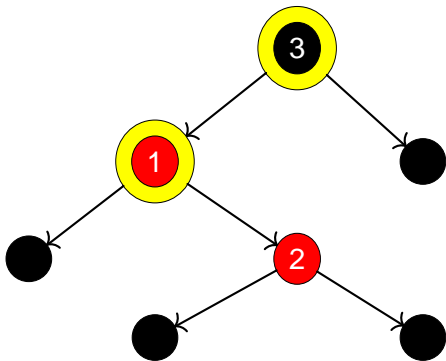
# Example Revisited

```
1 add(3);  
2 add(1);  
3 (add(2) || print(contains(1)))
```



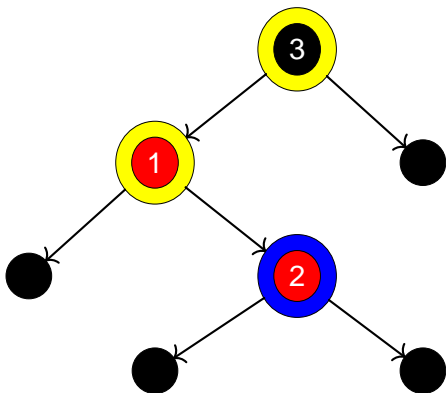
# Example Revisited

```
1 add(3);  
2 add(1);  
3 (add(2) || print(contains(1)))
```



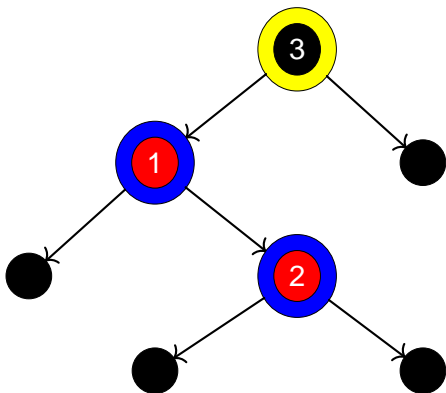
# Example Revisited

```
1 add(3);  
2 add(1);  
3 (add(2) || print(contains(1)))
```



# Example Revisited

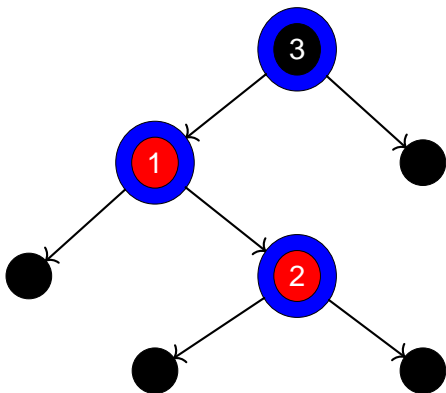
```
1 add(3);  
2 add(1);  
3 (add(2) || print(contains(1)))
```





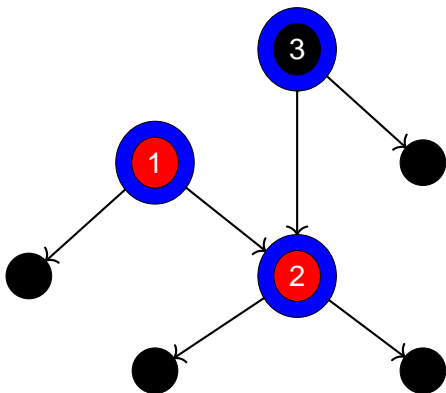
# Example Revisited

```
1 add(3);  
2 add(1);  
3 (add(2) || print(contains(1)))
```



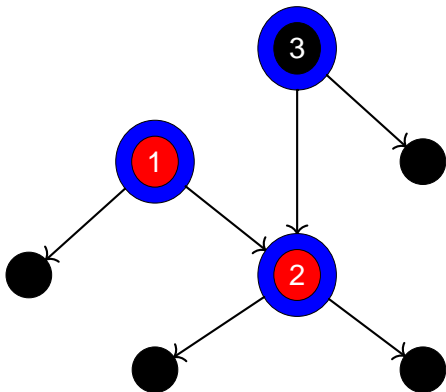
# Example Revisited

```
1 add(3);  
2 add(1);  
3 (add(2) || print(contains(1)))
```



# Example Revisited

```
1 add(3);  
2 add(1);  
3 (add(2) || print(contains(1)))
```



## Plan

- implement all three algorithms
- compare their performance

## Challenges

- adjust algorithm for AVL trees to red-black trees
- modify red-black tree algorithms of [Cormen, Leiserson, Rivest and Stein, 2001]
- when a process unlocks a node, which of the processes that are waiting to lock the node is chosen? (not addressed in the paper, PhD thesis is not available)