Concurrent Red-Black Trees

Franck van Breugel

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

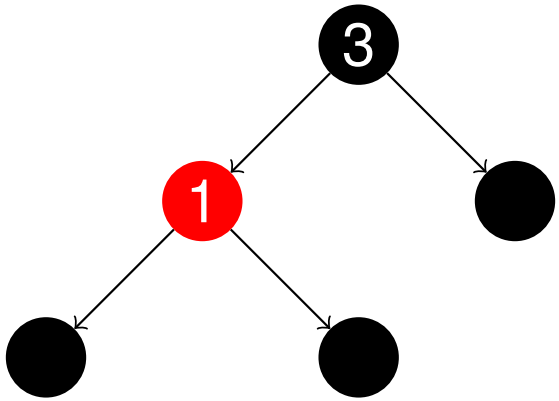
October 27, 2015

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]

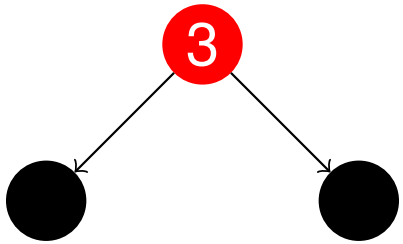


Concurrent Operations on a Red-Black Tree

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

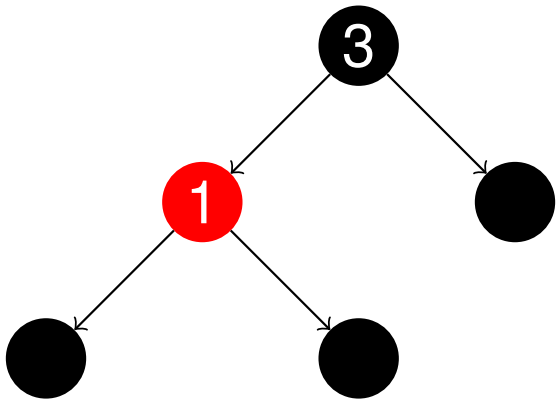
Concurrent Operations on a Red-Black Tree

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



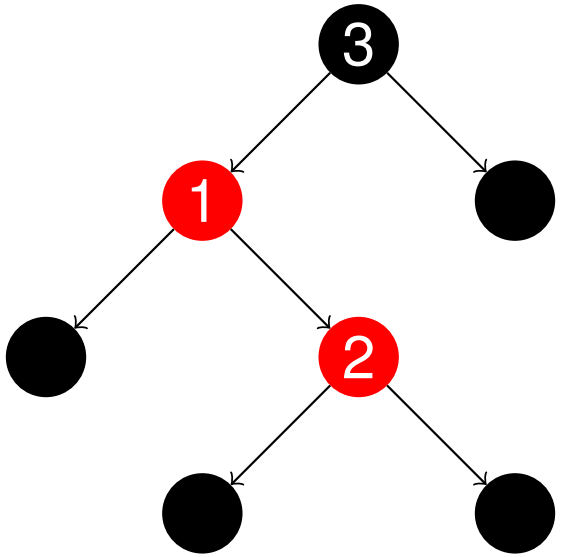
Concurrent Operations on a Red-Black Tree

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



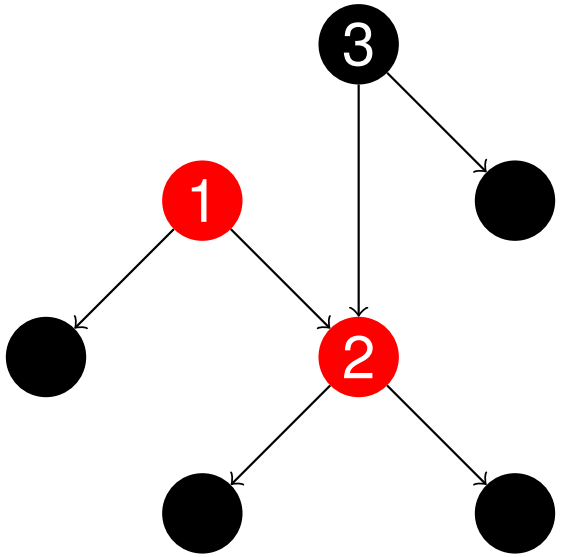
Concurrent Operations on a Red-Black Tree

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



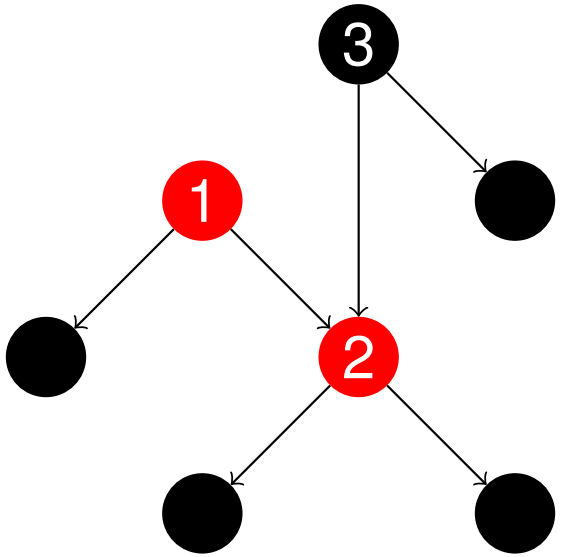
Concurrent Operations on a Red-Black Tree

```
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)))
```

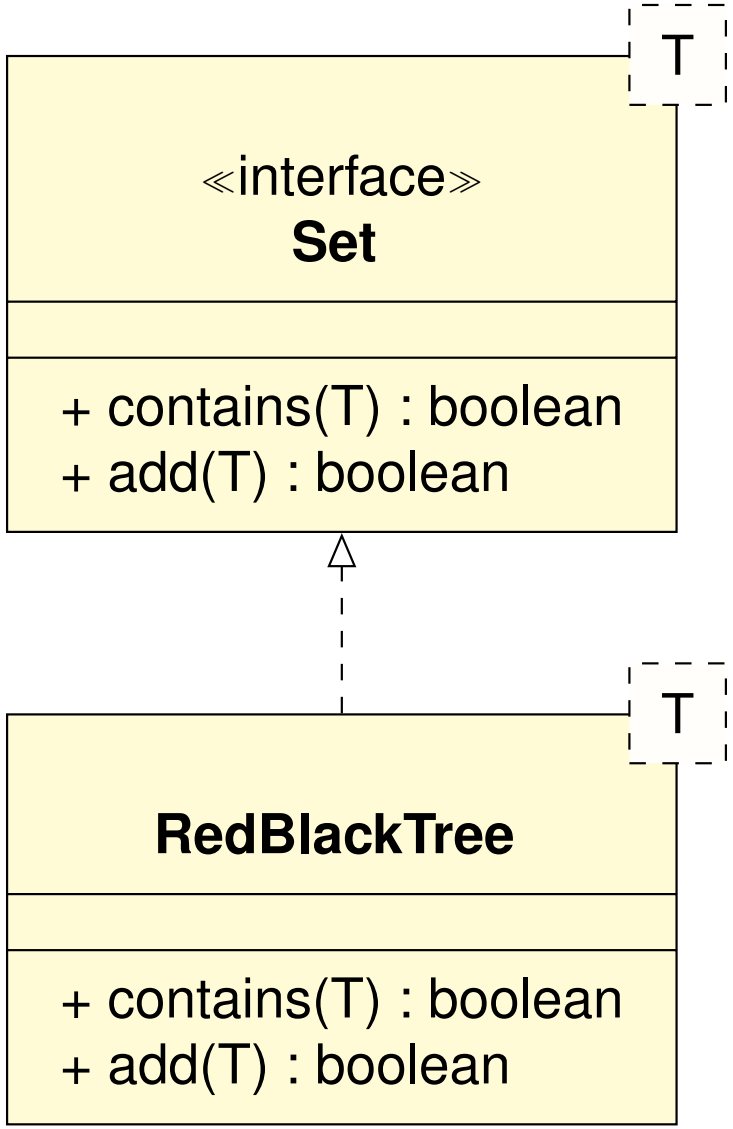


Concurrent Operations on a Red-Black Tree

Can we reproduce this interleaving?

- One Intel Pentium M processor, 1.6 GHz, Windows XP
  Prints true: 1000000, prints false: 0
- Two Intel Pentium 4 processors, 3 GHz, Linux 2.6.9
  Prints true: 999997, prints false: 3
- Eight Intel Xeon processors, 2.66 GHz, Linux 2.6.9
  Prints true: 1000000, prints false: 0
- Two AMD Athlon 64 X2 Dual Core processors, 2.2 GHz, Linux 2.6.9
  Prints true: 999999, prints false: 1
- Eight Intel Xeon Ten Core processors, 2.27 GHz, Linux 2.6.18
  Prints true: 999947, prints false: 53

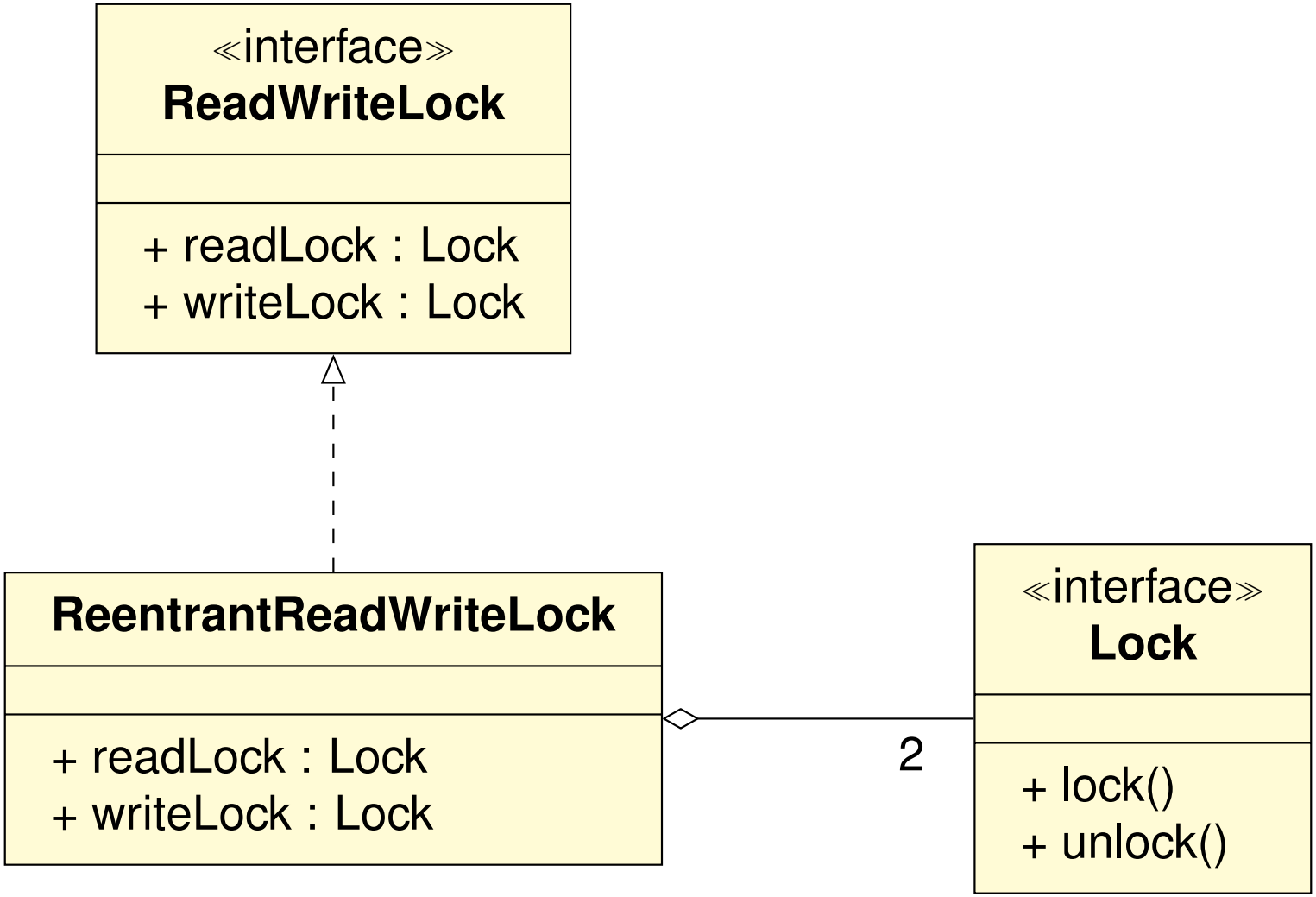
Three Implementations



The Monitor Solution

```
1  package monitor;
2
3  public class RedBlackTree<T extends Comparable<T>>
4     implements Set<T>
5  {
6     public synchronized boolean contains(T element)
7     {
8        ...
9     }
10
11    public synchronized boolean add(T element)
12    {
13       ...
14    }
15 }
```

The Readers-Writers Solution



The Readers-Writers Solution

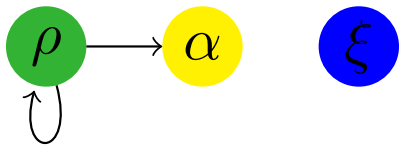
```
1  private ReadWriteLock lock;
2
3  public RedBlackTree()
4  {
5     this.lock = new ReentrantReadWriteLock();
6     ...
7  }
8
9  public boolean contains(T element)
10 {
11    this.lock.getReadLock().lock();
12    ...
13    this.lock.getReadLock().unlock();
14 }
15 ...
```

Locking Nodes

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

- ρ-lock: lock to read
- α-lock: lock to exclude writers
- ξ-lock: exclusive lock

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



Locking Nodes

```
1  public class Node<T>
2  {
3      private int containers;
4      private int state;
5      private boolean writing;
6
7      public void readLock() { ... }
8      public void readUnlock() { ... }
9      public void writeLock() { ... }
10     public void writeUnock() { ... }
11     public void exclusiveLock() { ... }
12     public void exclusiveUnlock() { ... }
13 }
```

Sequential Test

```
1  tree ← empty red−black tree
2  set ← empty set
3  do many times
4     element ← random element
5     check whether contains(element) returns
6        the same result for tree and set
7     check whether add(element) returns
8        the same result for tree and set
9     check whether tree is a red−black tree
```

Implemented using JUnit.

Concurrent Tests

```
1 tree ← empty red-black tree
2 do many times concurrently
3    do many times
4       element ← random element
5       add(element)
6 check whether tree is a red-black tree
```

Concurrent Tests

```
1 tree ← empty red−black tree
2 do many times concurrently
3    do many times
4       element ← random even element
5       add(element)
6    do many times
7       element ← random odd element
8       check whether contains(element) returns false
9 check whether tree is a red−black tree
```

Concurrent Tests

```
1  tree ← empty red−black tree
2  for element = 1, ..., n
3    add(element)
4  do many times concurrently
5    do many times
6      element ← random even element
7      add(element)
8    for element = 1, ..., n
9      check whether contains(element) returns true
10 check whether tree is a red−black tree
```

Looking Ahead

- Debug the implementation that locks individual nodes.
- Measure the throughput of the three implementations.