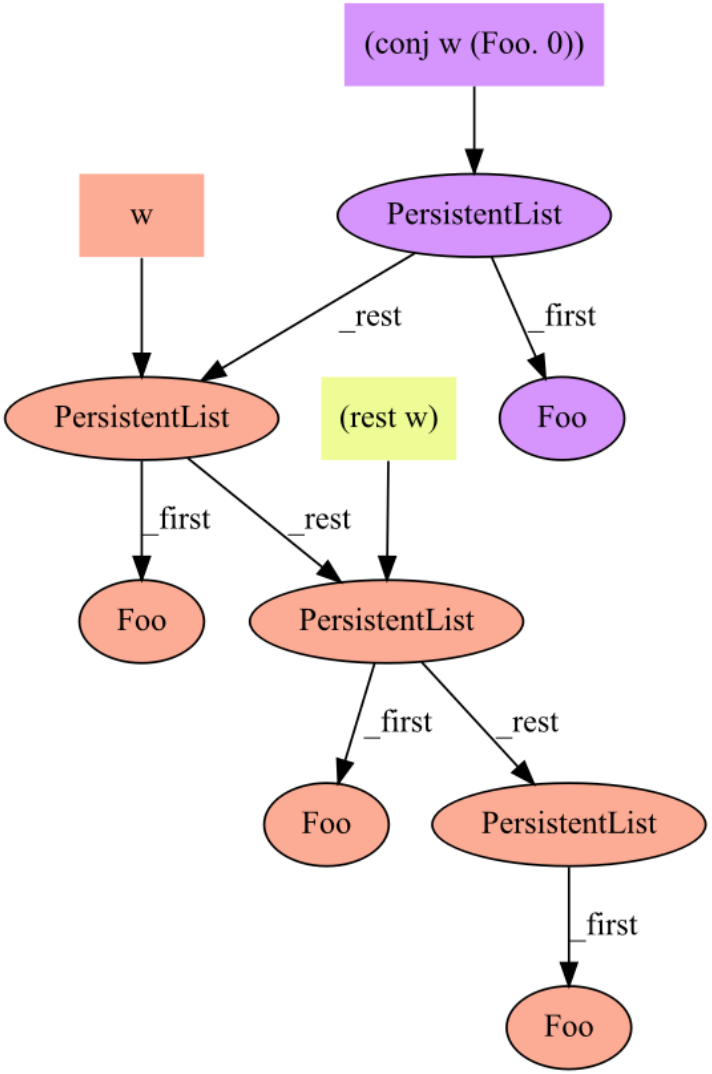
Immutable Collections from Clojure and Scala

Steven Reynolds
steven@geotoolkit.net

Clojure Houston User Group

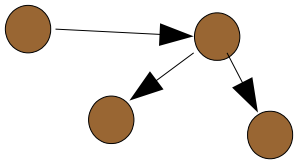


Slides & graphs will be available at
www.slreynolds.net

Code is on github:
http://stevenreyn.github.io/RegionViewer/

Immutable Collections
Why bother?

- Simpler to maintain consistent state

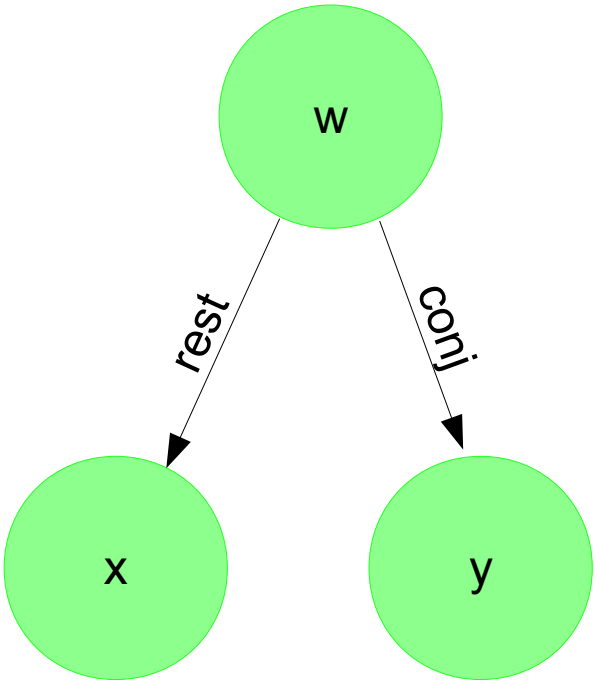


- Automatically thread safe
- Good for big data (D. Wampler)

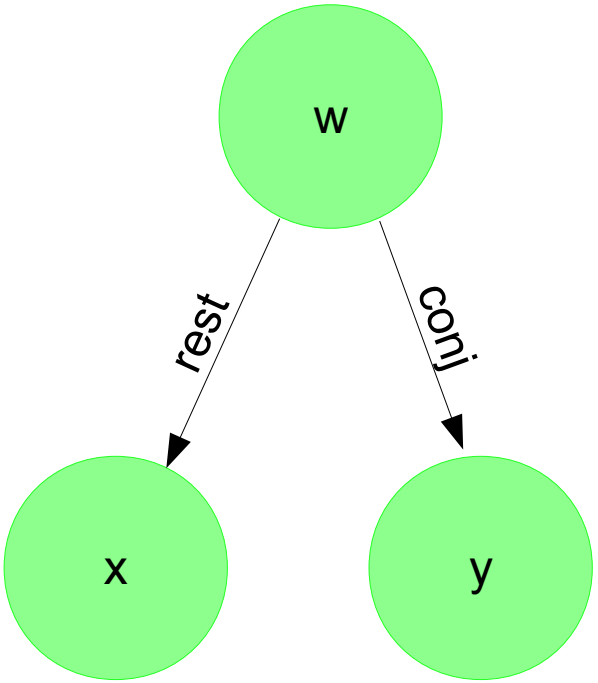
Immutable Data Structures don't change

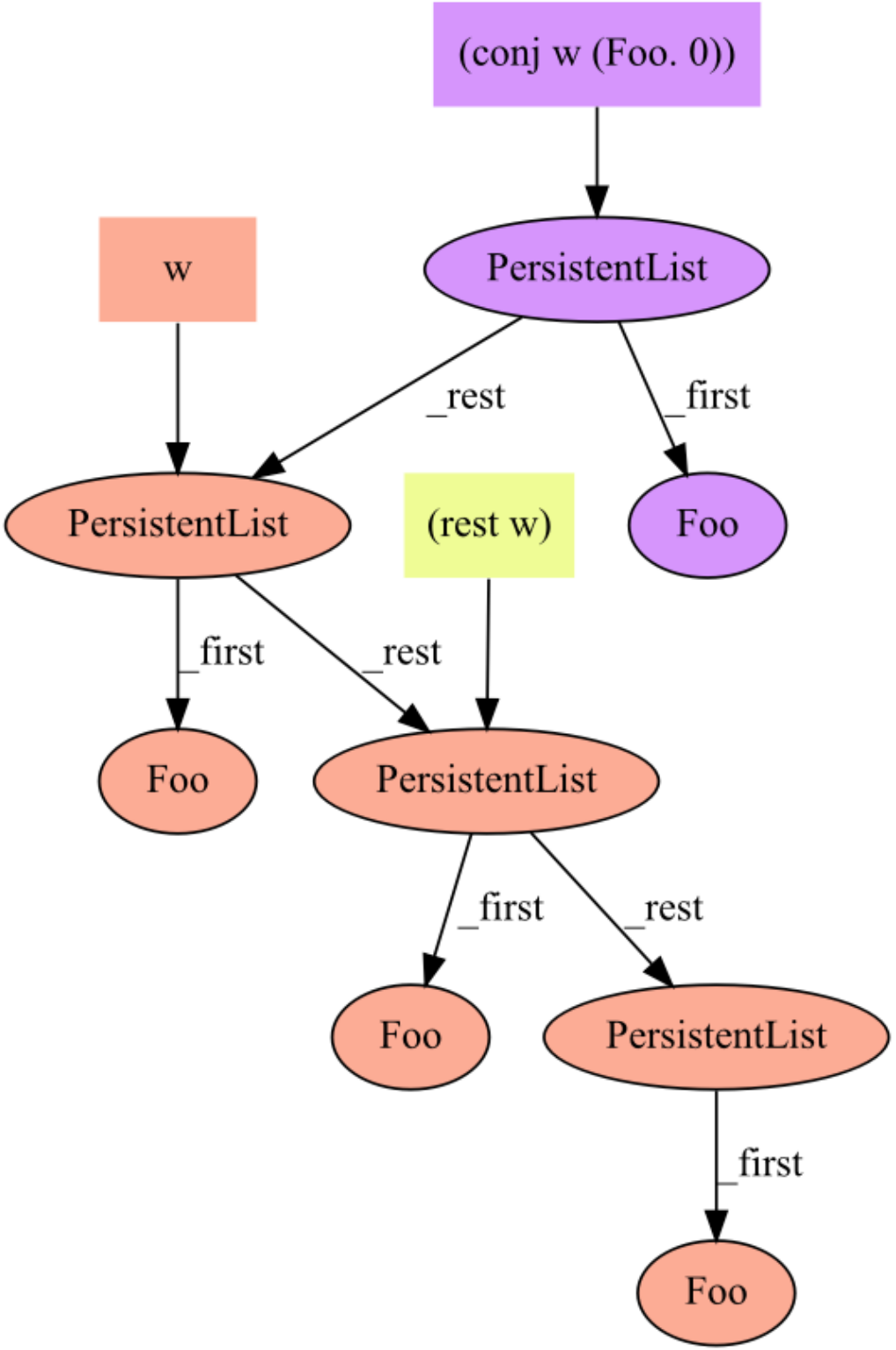
```
(let [w (list ...)
      x (rest w)
      y (conj  w (Foo. 0))]
    ...
```

When you operate on one,
You get back a new one

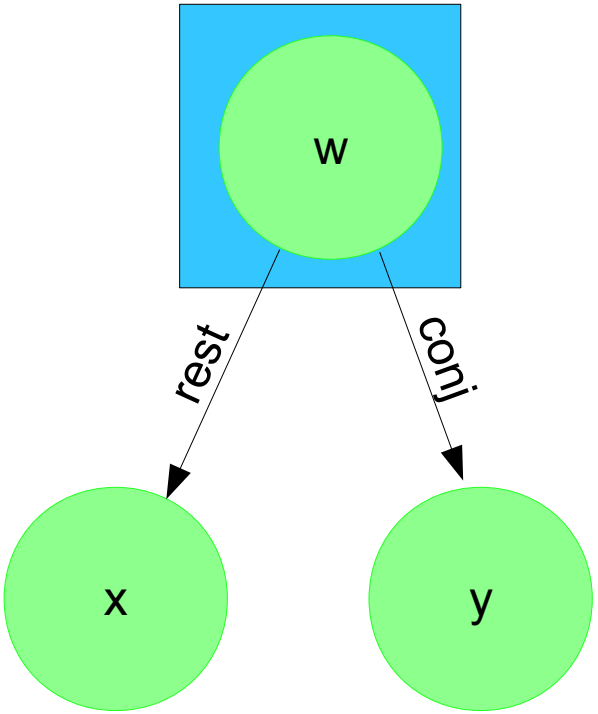


```
(let [w (list ...)
      x (rest w)
      y (conj  w (Foo. 0))]
  ...
```





```
(let [w (list ...)
      x (rest w)
      y (conj  w (Foo. 0))]
  ...
```



(conj w (Foo. 0))

w

PersistentList

_rest

_first

PersistentList

(rest w)

Foo

_first

_rest

Foo

PersistentList

_first

_rest

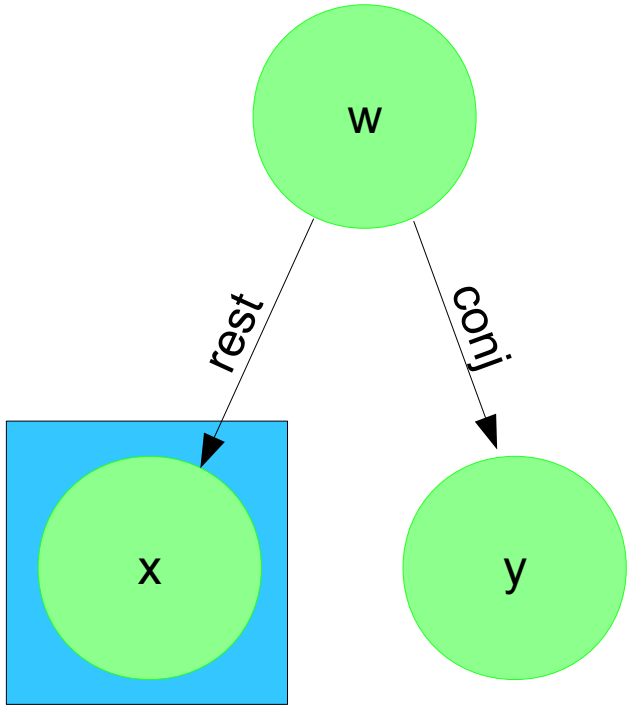
Foo

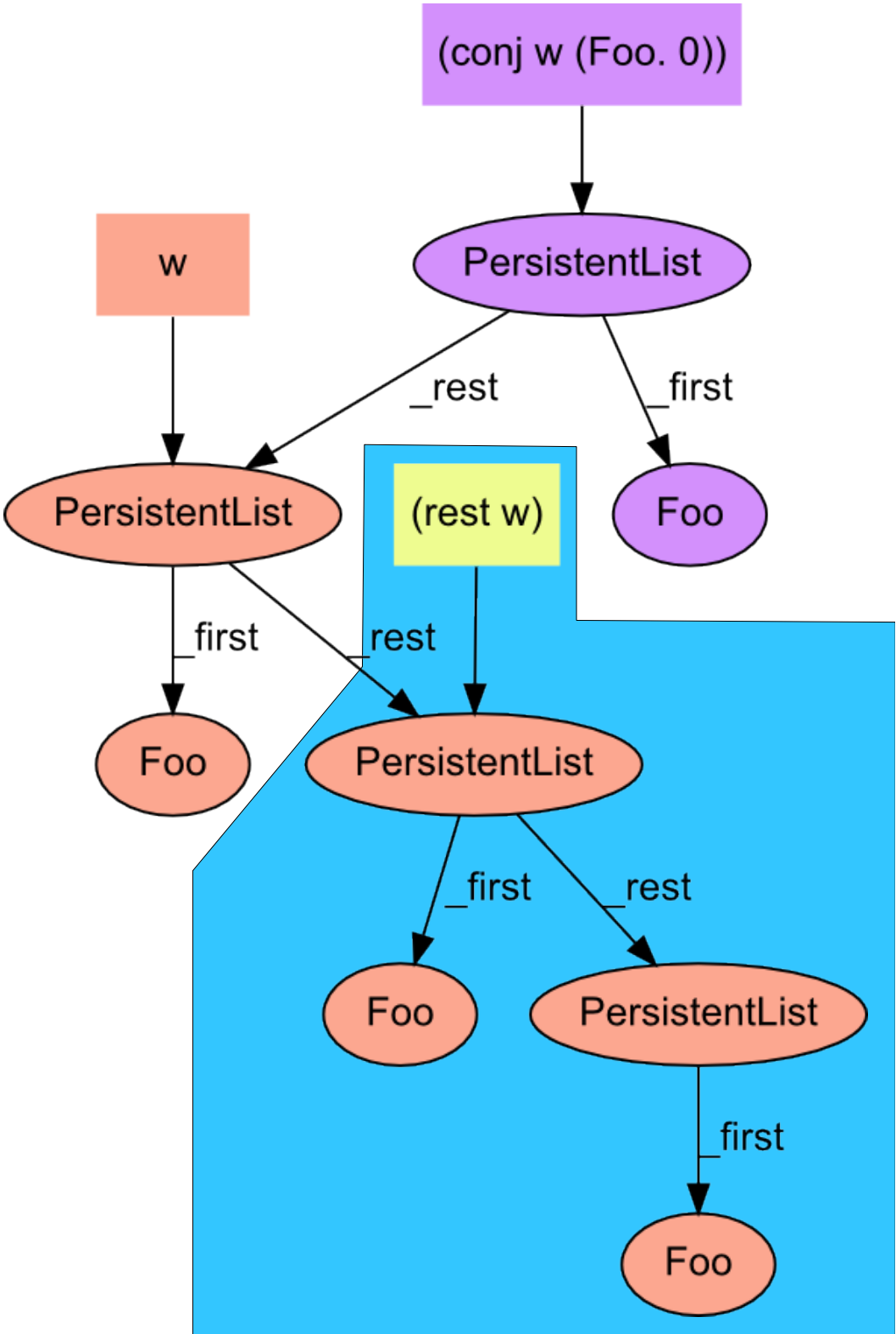
PersistentList

_first

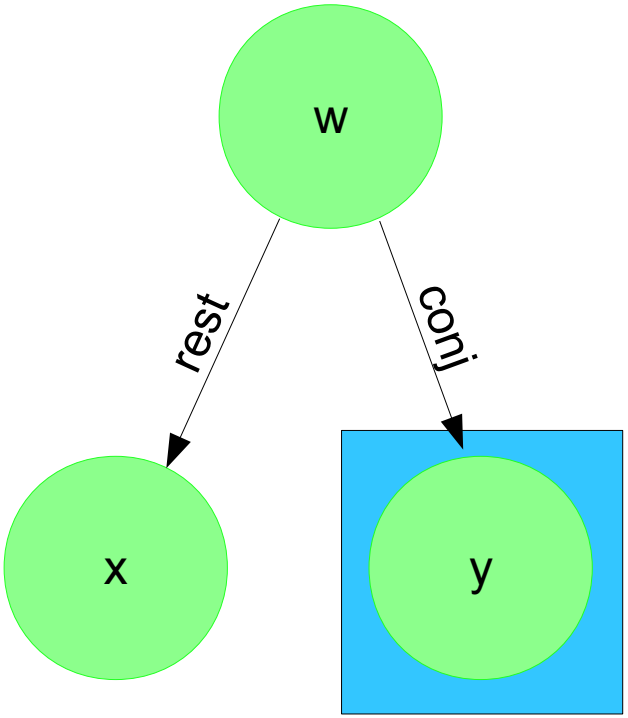
Foo

```
(let [w (list ...)
      x (rest w)
      y (conj  w (Foo. 0))]
  ...
```





```
(let [w (list ...)
      x (rest w)
      y (conj  w (Foo. 0))]
  ...
```



(conj w (Foo. 0))

w

PersistentList

_rest

_first

PersistentList

(rest w)

Foo

_first

_rest

Foo

PersistentList

_first

_rest

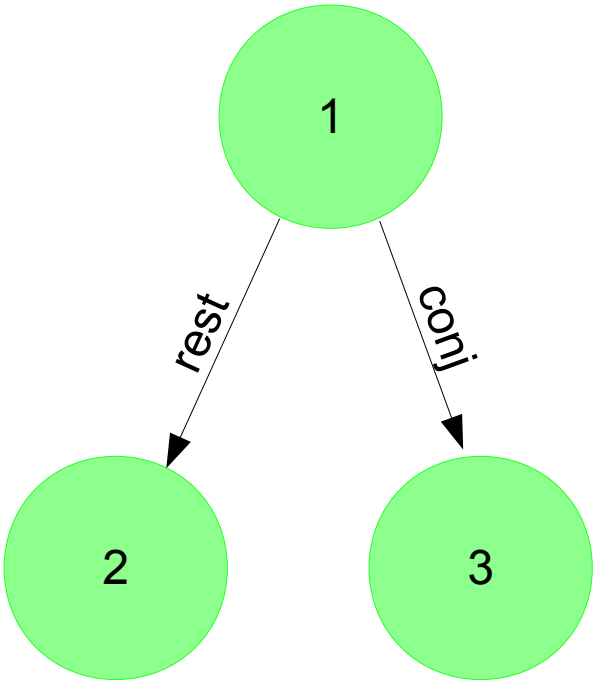
Foo

PersistentList

_first

Foo

```
(let [w (list ...)
      x (rest w)
      y (conj  w (Foo. 0))]
    ...
```



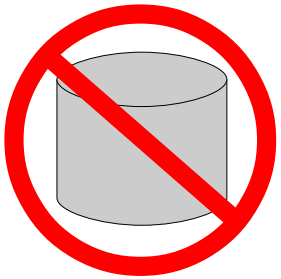
Multiple versions of
the same data structure

Equivalent Adjectives

Immutable DS – They don't change

Functional DS – Code does not mutate data or re-assign variables

Persistent DS – You can access and use old versions

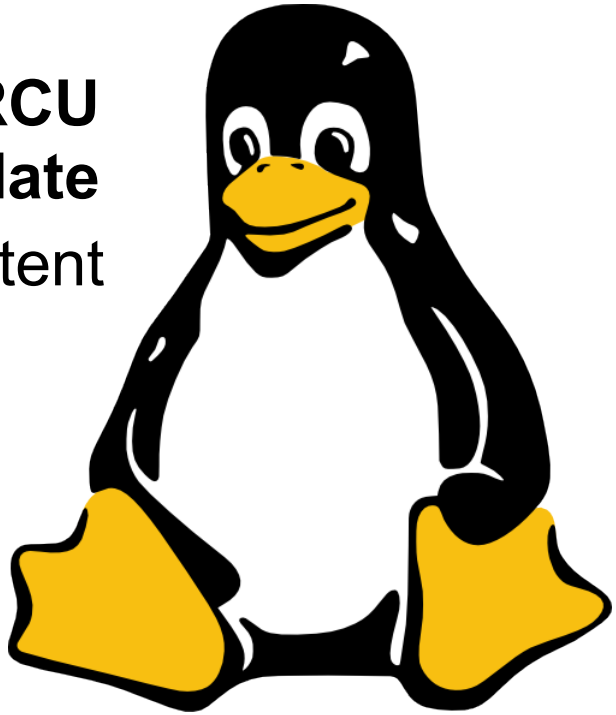


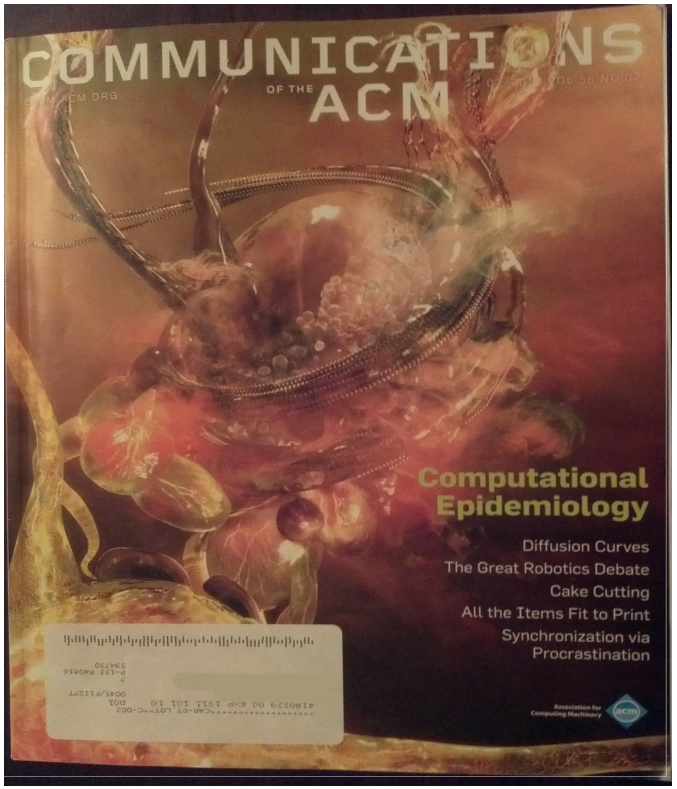
Persistent Data Structures are very common

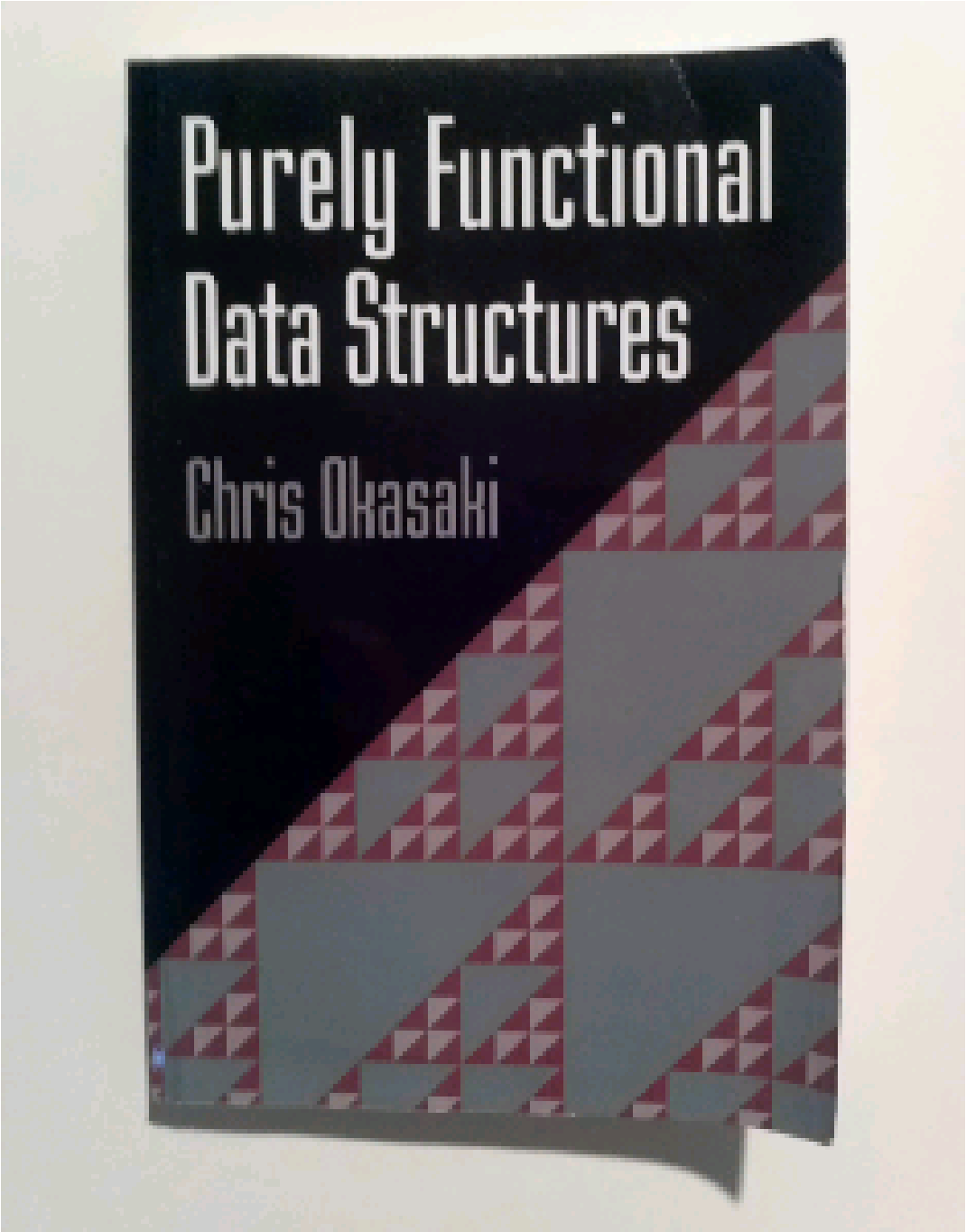
RCU

Read Copy Update

Partially Persistent






Purely Functional
Data Structures
Chris Okasaki

... functional programming's stricture against destructive updates (i.e. assignments) is a staggering handicap, tantamount to confiscating a master chef's knives.

- Chris Okasaki

Warning

Graphs show internal details that may change
Based on clojure 1.5.1 and scala 2.10.2

Outline of Talk

Introduction
Familiar Java objects
List
Map
Vector
Clojure and Scala
Clojure Transients
Scala par
Measured Performance

show graphs here

Hash Array Mapped Trie (HAMT)

Phil Bagwell

It's a tree
Hash(Key) encoded in path from root } ⇨ a trie

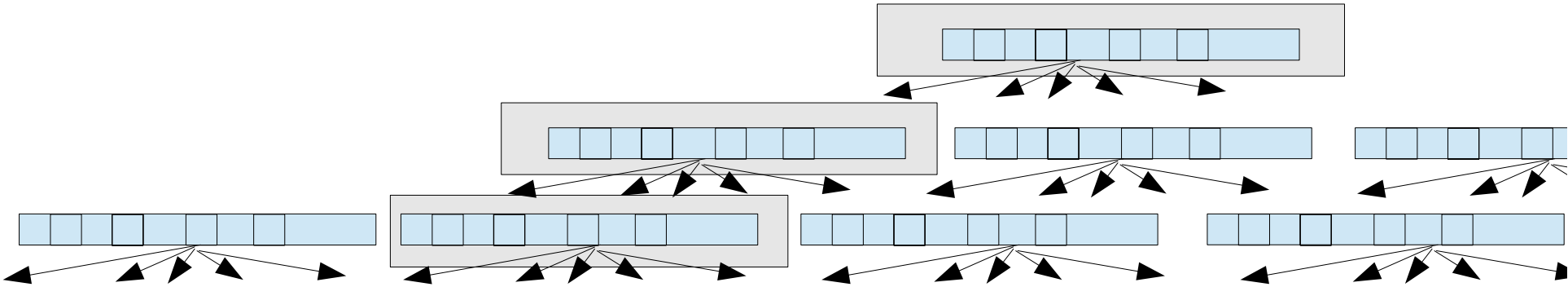
5 bits of the hash at a time
32 way branching - broad and shallow

Uses integer bitmap to indicate occupied hash slots =>
elements array is densely packed

Update

We keep the previous tree
Add new nodes where needed

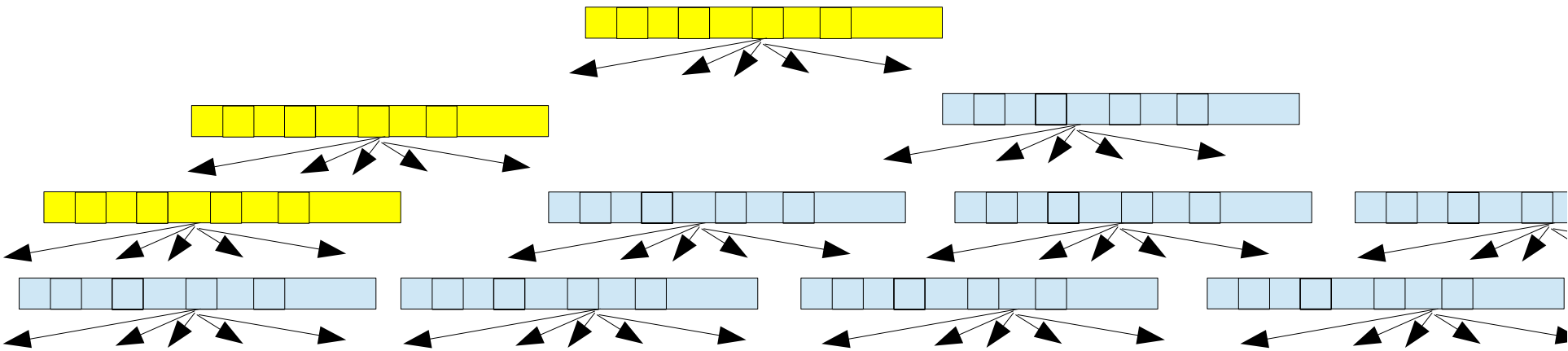
Path Copying



Update

We keep the previous tree
Add new nodes where needed

Path Copying



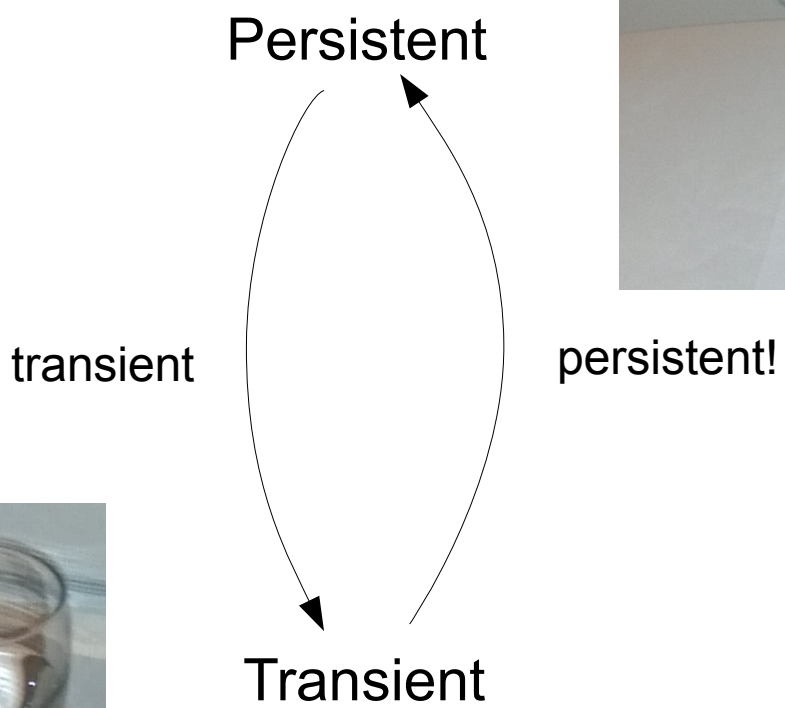
Cost is $O(\log_{32} n)$

Effectively constant

show graphs here

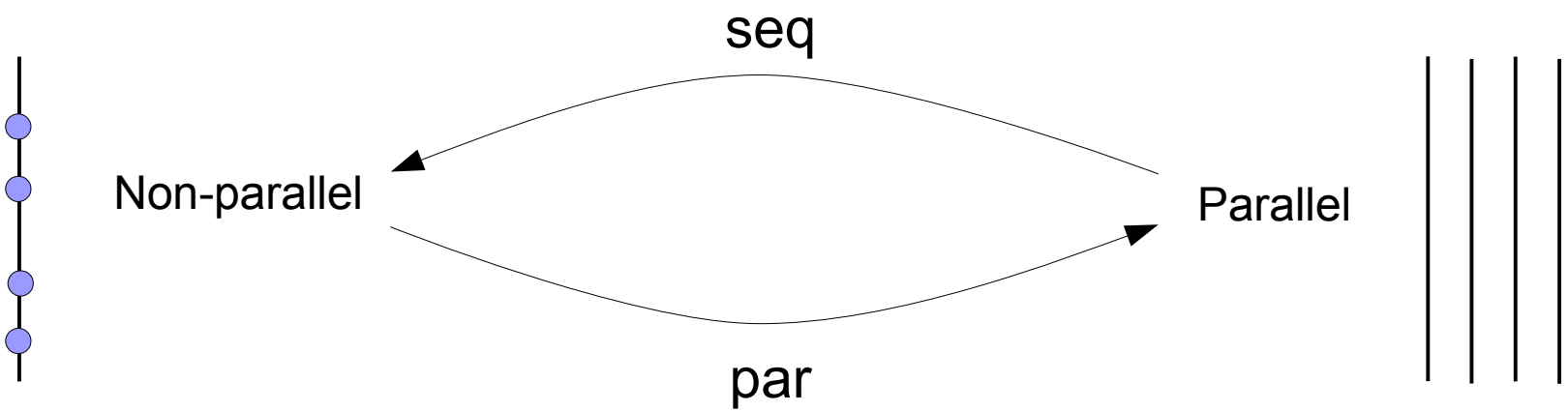
Transients

Clojure

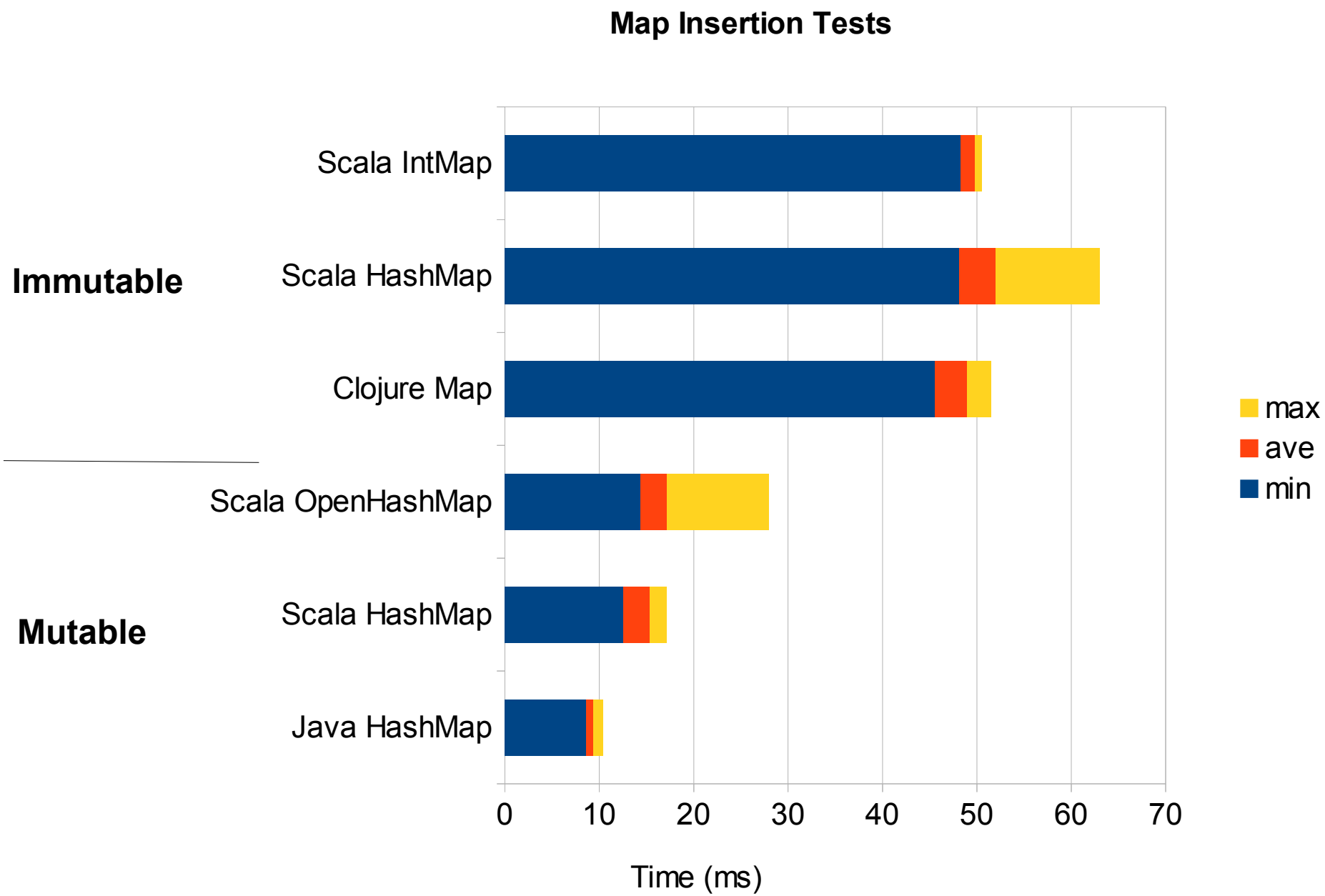


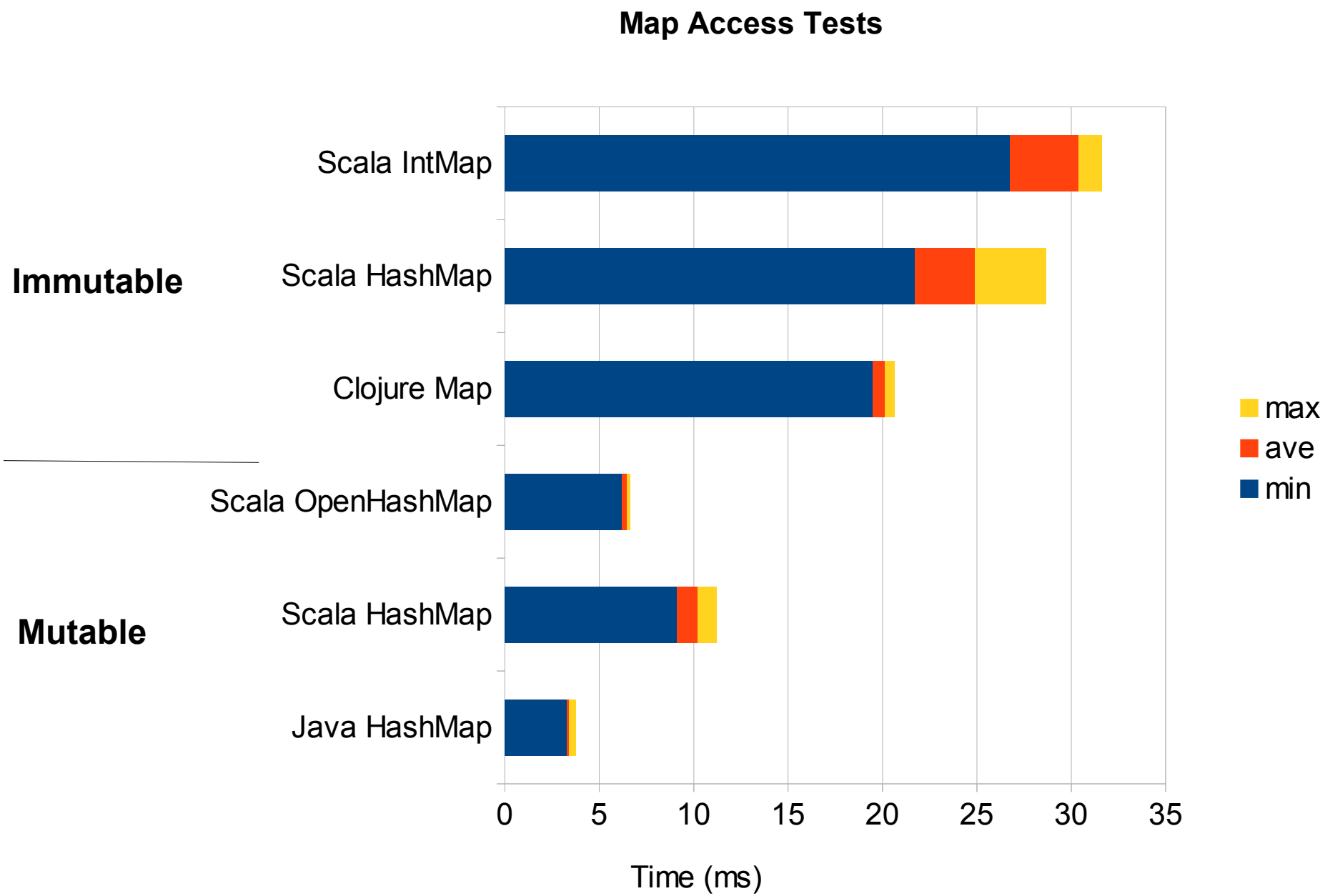
Parallel

Scala



show graphs here


Map Insertion Tests
Immutable
Mutable
Scala IntMap
Scala HashMap
Clojure Map
Scala OpenHashMap
Scala HashMap
Java HashMap
0
10
20
30
40
50
60
70
Time (ms)
max
ave
min


Map Access Tests
Immutable
Mutable
Scala IntMap
Scala HashMap
Clojure Map
Scala OpenHashMap
Scala HashMap
Java HashMap
0
5
10
15
20
25
30
35
Time (ms)
max
ave
min

Thank you

Phil Bagwell, Aleksandar Prokopec, Tiark Rompf
Rich Hickey
Martin Orderski
Clojure Developers
Scala Developers
Chris Okasaki
GraphViz Project

URLs

Slides & graphs: www.slreynolds.net

github:
http://stevenreyn.github.io/RegionViewer/