# Immutable data structures: AVL, RRB-Vector, Finger-tree

Flipflop team

Jordi Bertran de Balanda  Morane Gruenpeter  Guillaume Hivert
Mourtala Issa Saidou  Etienne Mauffret  Darius Mercadier

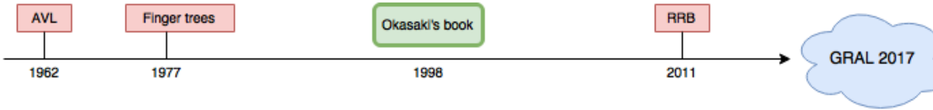
Université Pierre et Marie Curie

2 February 2017

# Scientific context

## Immutability

" Functional programming languages have the curious property that all data structures are automatically persistent." (Okasaki, 1998)

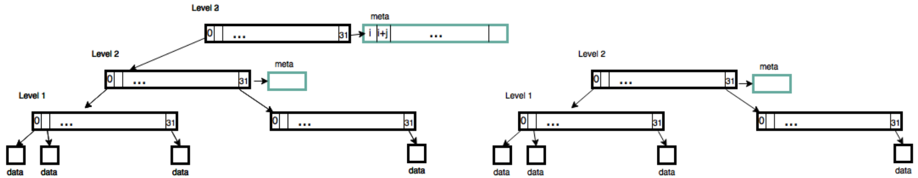



- How can we implement immutable data structures in an imperative language ?

# AVL- from mutability to immutability

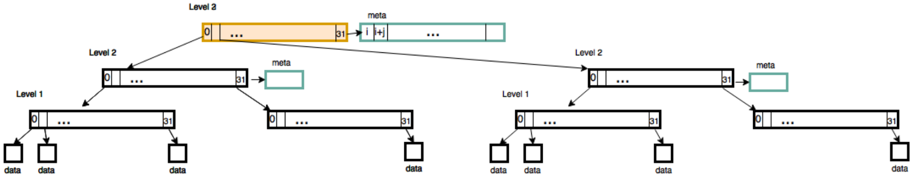
# Relaxed-Radix-Balanced vectors - merge

IF (left RRB tree level > right RRB tree level )



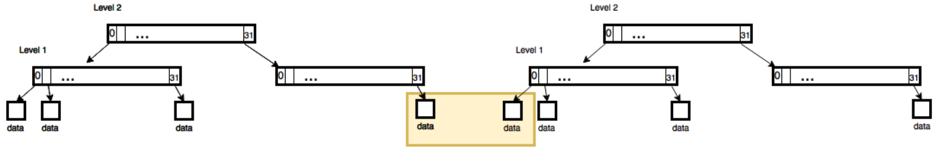
# Relaxed-Radix-Balanced vectors - merge

Add as child



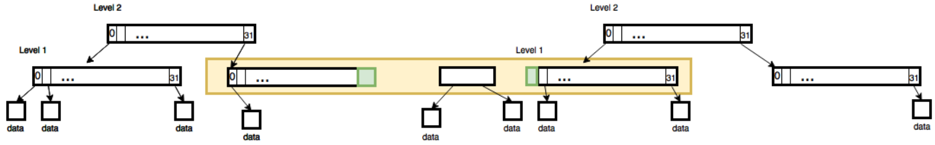
# Relaxed-Radix-Balanced vectors - merge

Two RRB vectors with same number of levels [1]



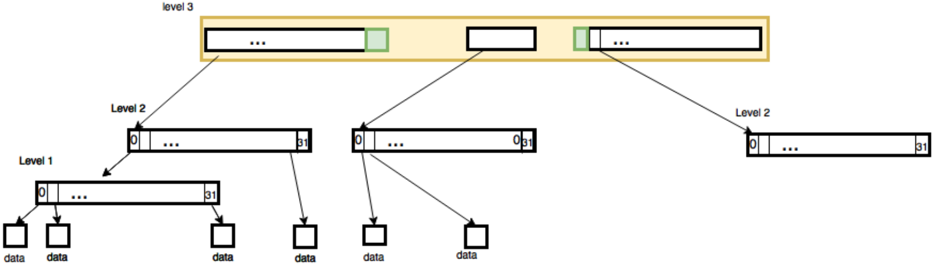

1. metadata is omitted to simplify figure

# Relaxed-Radix-Balanced vectors - merge

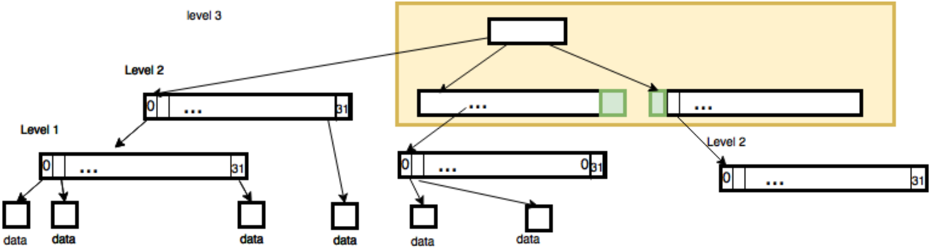



UPMC SORBONNE UNIVERSITÉS

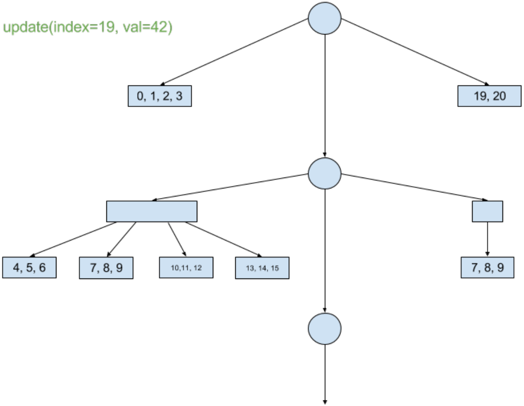
# Relaxed-Radix-Balanced vectors - merge



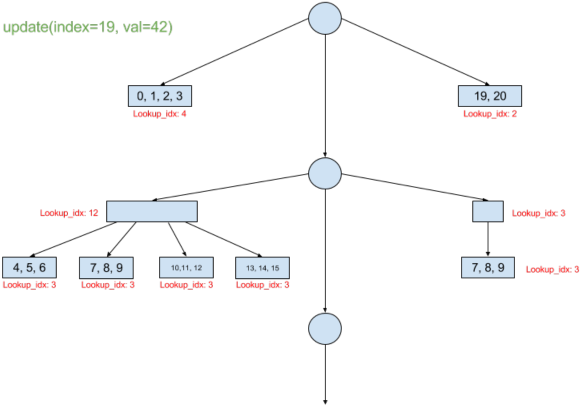
# Relaxed-Radix-Balanced vectors - merge



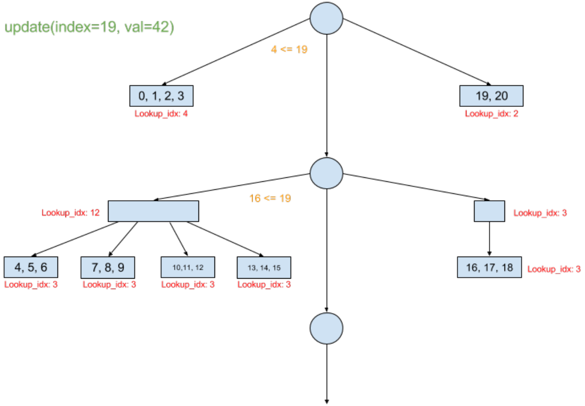
# Finger trees - Update



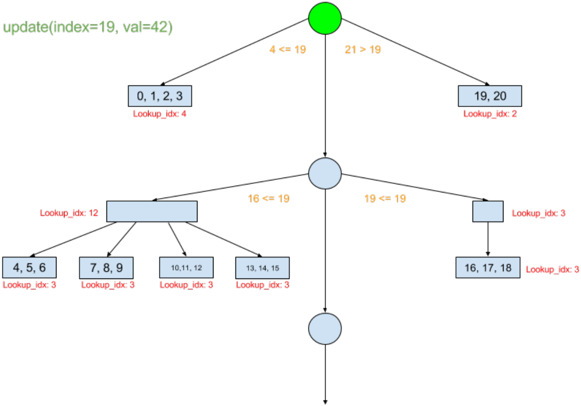
# Finger trees - Update



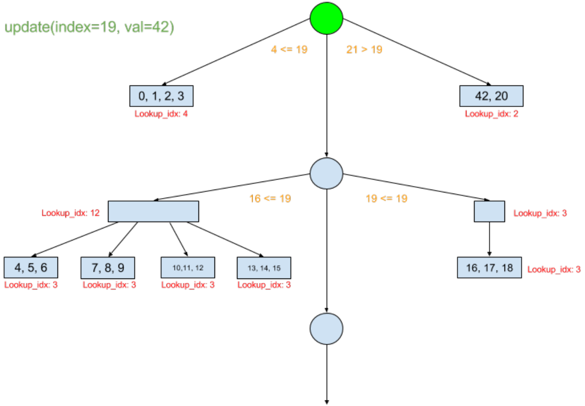
# Finger trees - Update



# Finger trees - Update



# Finger trees - Update



# Immutable structures : complexity

- Same time Complexity as the classic data structures
- Low Memory complexity for classic operations

Table – Spatial Complexity in different immutable data-structures

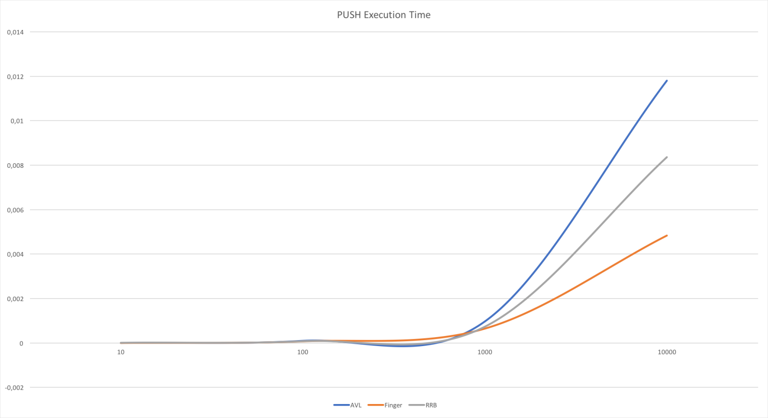
| Operation | RRB | finger trees | AVL trees |
|---|---|---|---|
| Update | $\mathcal{O}(log_m(n))$ | $\mathcal{O}(log_4(n))$ | – |
| Insert/Push | $\mathcal{O}(log_m(n))$ | $\mathcal{O}(log_4(n))$ Amort.$\mathcal{O}(1)$ | $\mathcal{O}(log_2(n))$ |
| Remove/Pop | $\mathcal{O}(log_m(n))$ | $\mathcal{O}(log_4(n))$ Amort.$\mathcal{O}(1)$ | $\mathcal{O}(log_2(n))$ |
| Splitting | $\mathcal{O}(log_m(n))$ | $\mathcal{O}(log_4(n))$ | $\mathcal{O}(n)$ |
| Merge | $\mathcal{O}(log_m(n))$ | $\mathcal{O}(log_4(n))$ | $\mathcal{O}(n+m)$ $\mathcal{O}(m.log_2(n))$ |

# Immutable structures : complexity

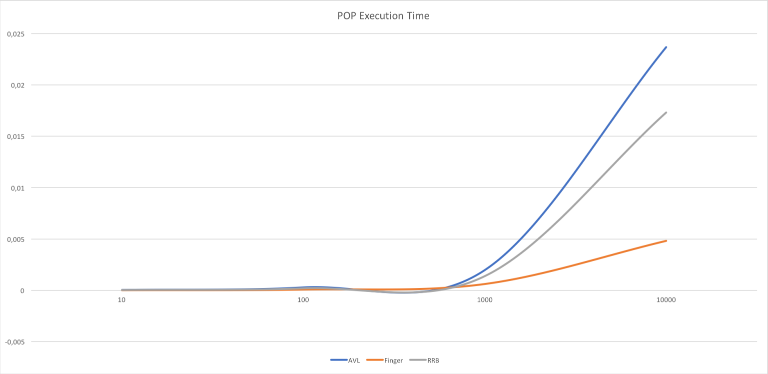
Complexity of a structure with $n$ elements,
with $k$ deletion and $r$ the number of update

- AVL and RRBVector on $\mathcal{O}((n+k).log_m(n+k))$
- Finger Tree in $\mathcal{O}(n+k+r.log_4(n+k))$
- Small cost to use immutability if using few deletion and update
- Can be costly with more modification than insertion

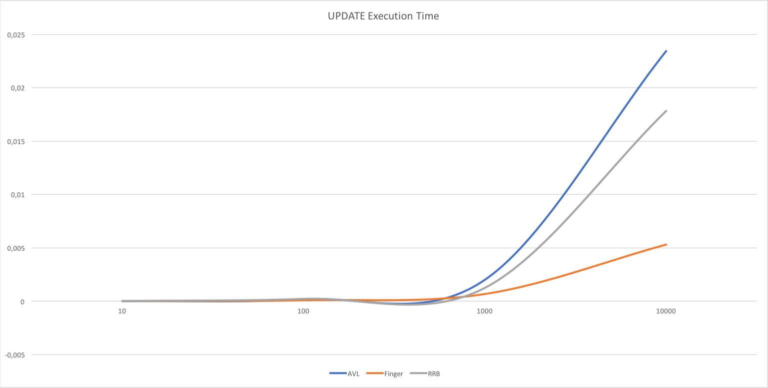
# Benchmark- push



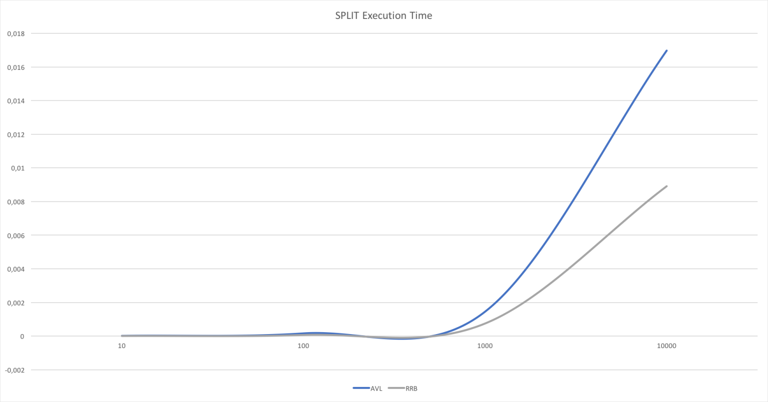
# Benchmark- pop



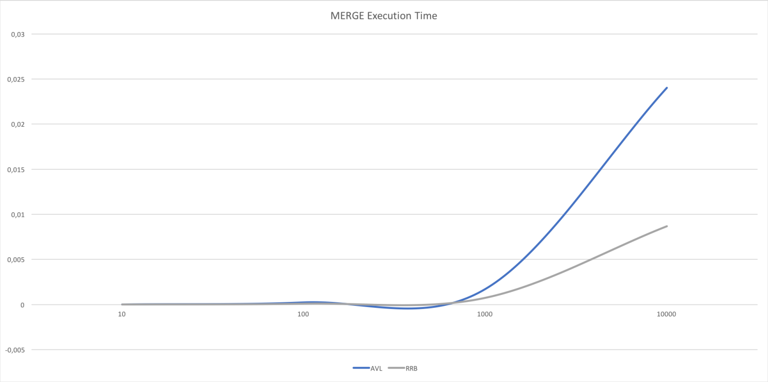
# Benchmark- update



# Benchmark- split



# Benchmark- merge



# Conclusion

RRB-vectors are a good data structure for parallelization

- Efficient splitting and concatenation operations
- Performance of discrete operations is consistently good

Finger trees are great at deque-like operations

- Efficient first and last element access
- Less efficient at split and merge operations