The Other Data Structures

@jonasenlund

About me

- Live 250km northwest of here
- Work for a Non-Profit organization called Akvo
    - Mobile phone based field surveys
    - Used in post-Earthquake Nepal and post-“Cyclone Pam” in Vanuatu for damage assessment
    - Water point mapping and monitoring in Africa, India, Indonesia etc.
    - Some Clojure(Script) and lots of Java(script)

akvoflow

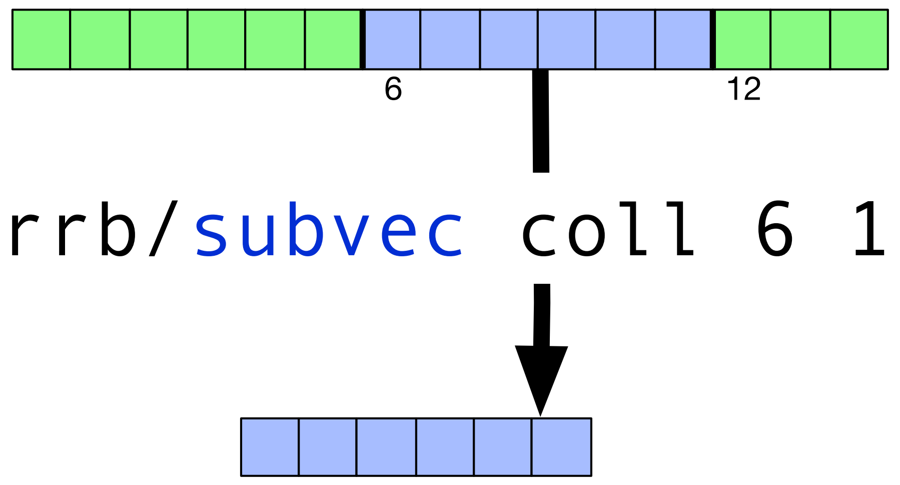
Agenda

- Persistent Data Structures!
- Many interesting (non-core) data structures available:
  - priority-maps, ctries, int-maps/sets, etc.
- Focus on **core.rrb-vector** and **data.avl**
  - Contrib libraries
  - Available for Clojure and ClojureScript
  - Both implementations by Michał Marczyk

core.rrb-vector

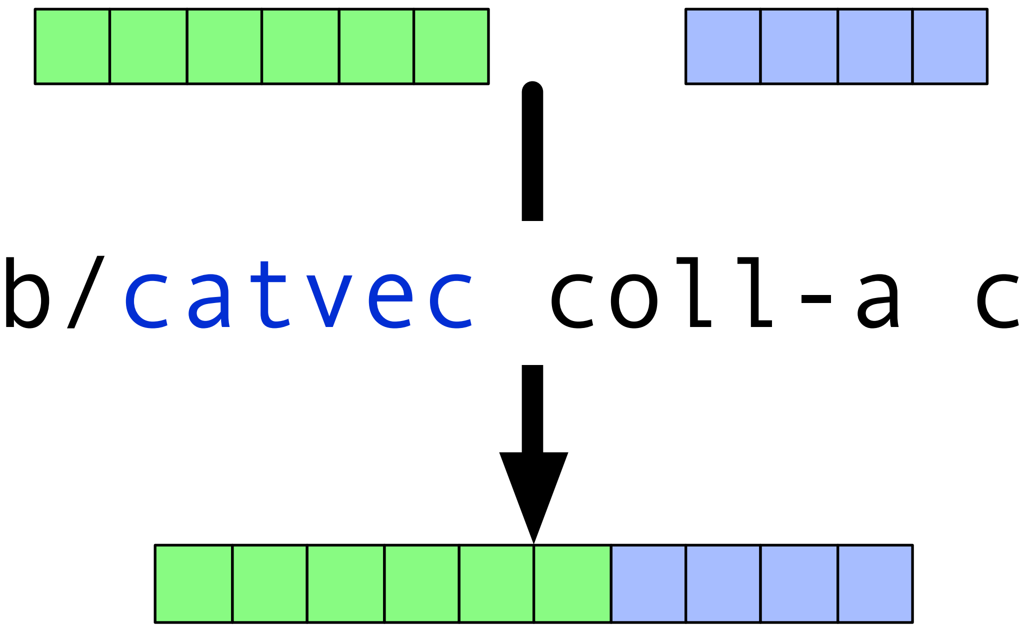
- Based on the paper **“RRB-Trees: Efficient Immutable Vectors”** by Bagwell & Rompf
- Similar to built in Clojure vectors with two key additions

“True” subvector



```
(rrb/subvec coll 6 12)
```

Concatenation



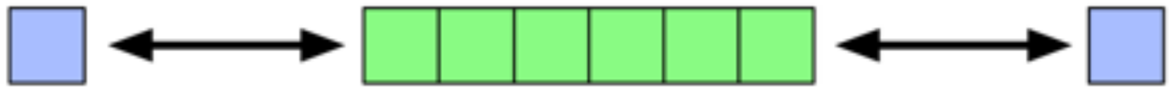
```
(rrb/catvec coll-a coll-b)
```

core.rrb-vector

- Both operations work on existing Clojure(script) vectors at O(log(n)) complexity.
- But:
  - Iteration (especially via ‘reduce’) will be slower.
  - Not as battle tested

Usage

- Brandon Bloom's **fipp** uses rrb-vectors as a **double-ended queue**.



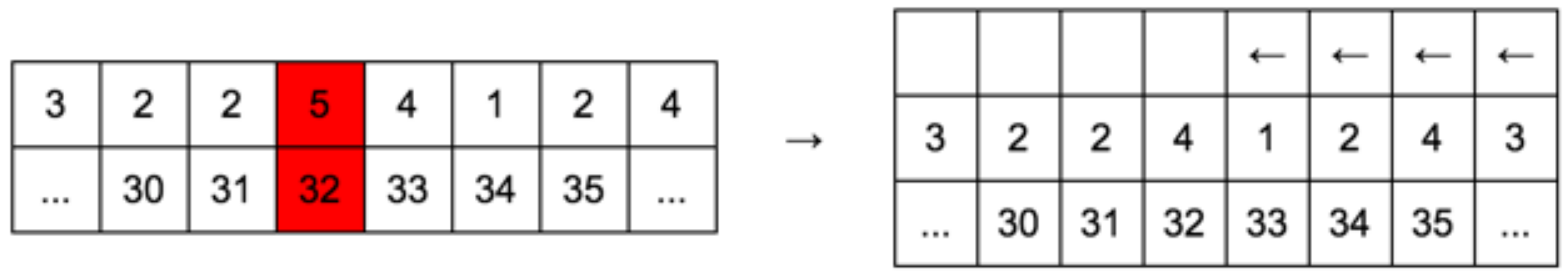
```
(defn conjlr [l deque r]
  (rrb/catvec [l] deque [r]))
```

- Using Clojure's Persistent Vector would make **conjlr** O(n) instead of O(log(n)).

Clojure Cup 2014

- Idea: Analyze git diffs (`@@ -s1,c1 +s2,c2 @@`) to track line-by-line file changes
- Parse these “hunks” into **`:insert`**, **`:edit`** and **`:delete`** operations.
- Keep a vector of “line edit counts”

`[:delete 32 1]`

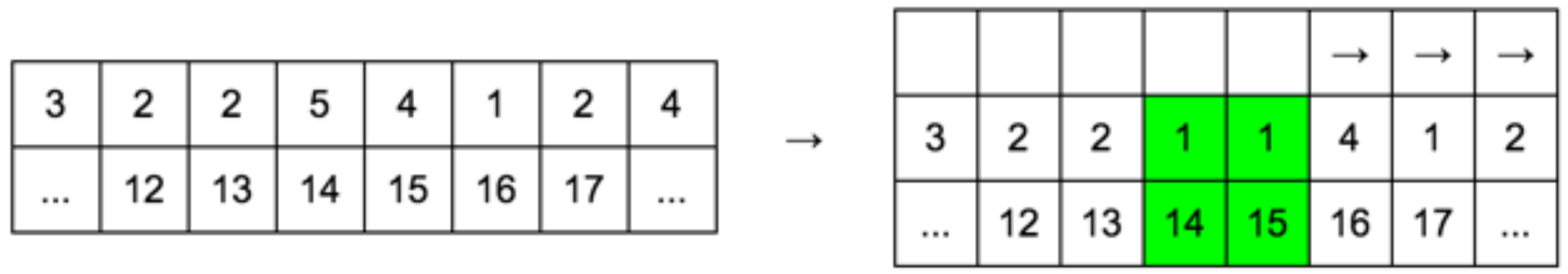


3	2	2	🟥 5	4	1	2	4
...	30	31	🟥 32	33	34	35	...

→

				←	←	←	←
3	2	2	4	1	2	4	3
...	30	31	32	33	34	35	...

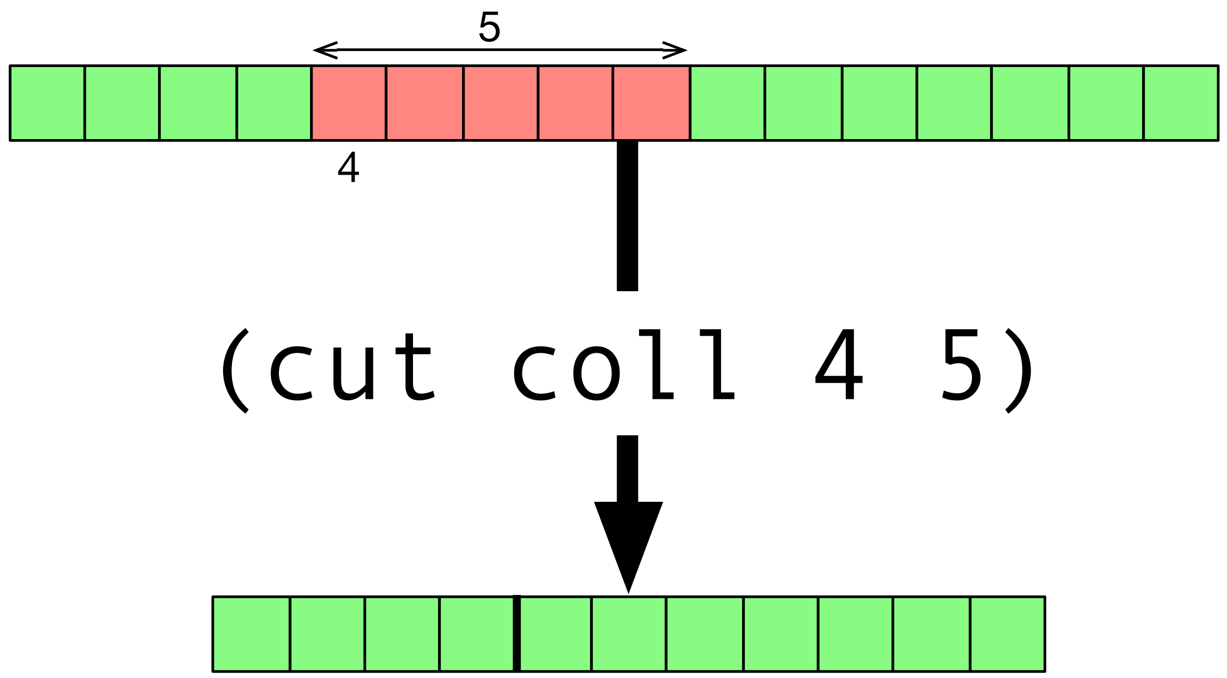
`[:insert 14 2]`



3	2	2	5	4	1	2	4
...	12	13	14	15	16	17	...

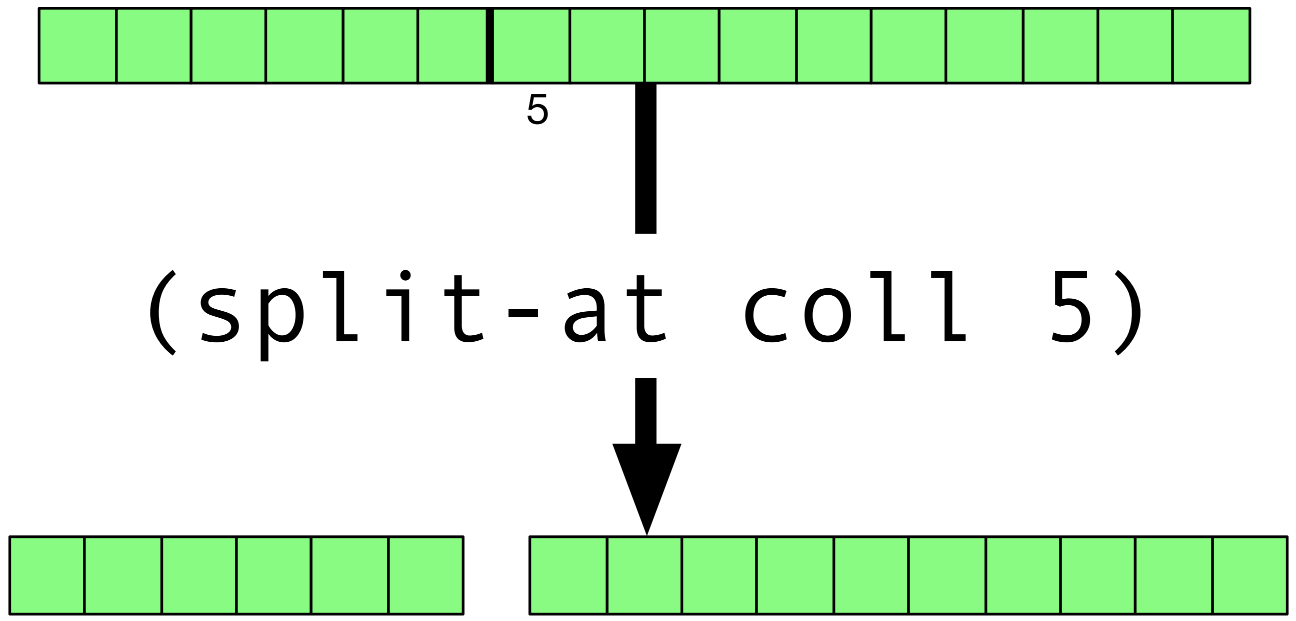
→

					→	→	→
3	2	2	🟩 1	🟩 1	4	1	2
...	12	13	🟩 14	🟩 15	16	17	...

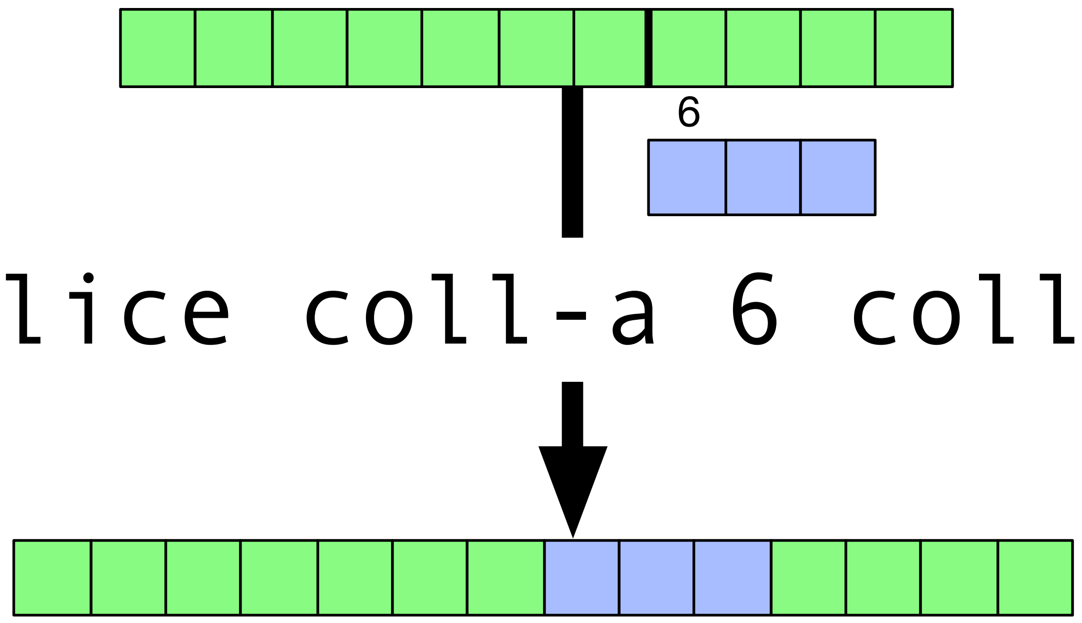


```
(cut coll 4 5)
```

```
(defn cut [coll start length]
  (rrb/catvec (rrb/subvec coll 0 start)
              (rrb/subvec coll (+ start length))))
```



```clojure
(defn split-at [coll n]
  [(rrb/subvec coll 0 n)
   (rrb/subvec coll n)])
```



```
(defn splice [coll-a idx coll-b]
  (let [[left right] (split-at coll-a idx)]
    (rrb/catvec left coll-b right)))
```

core.rrb-vector

- Consider using core.rrb-vector when you need these operations
- For small vectors or one-off concats/subvecs there's probably no win
- Evaluate on a case-by-case basis

data.avl

data.avl use cases

- Datomic pagination:

  1. Query result => data.avl sorted set
  2. Thanks to lazy entities you only need to realise the attribute you sort on
  3. Use rank-queries for page results.

Use cases (2)

- Windowed event data keyed by timestamp

    1. Keep “events” in a sorted set (by timestamp)

    2. Periodically reduce the set using rank queries

    3. Since the subrange result is itself a sorted set there’s never a need for a O(n) operation.

“Data dominates. If you've chosen ***the right data structures*** *and organized things well, the algorithms will almost always be self-evident …”*

*“… **Data structures**, not algorithms,*
are central to programming.”

– Rob Pike