# Lost in Transduction :clojureD 2017, Christophe Grand @

ソフィア・コッポラ監

100

下品

返し点

で君に会え

What happened?

#### seqs

- simple model
- persistent
- strict laziness
- GC under pressure

seqs

#### chunked seqs

- less allocs
- more locality
- relax laziness
- branch my impl
  - → little support

seqs

chunked seqs

reducers

no alloc

• ~lazy

- not persistent
- fragmentation
  - map vs r/map

#### Meanwhile in core.async

async/map, async/filter, etc.

more fragmentation

against Clojure values

#### Clojure Rationale

It is better to have 100 functions operate on one data structure than to have 10 functions operate on 10 data structures.

Alan J. Perlis

clojure.org/about/rationale

seqs

chunked seqs

reducers

transducers

- no alloc
- ~lazy
- not persistent
- no fragmentation
  - → need context
- branch my impl
  - → wider support

#### Transducing contexts

- transduce
- sequence (2-arg)
- into (3-arg)
- eduction

- async/chan
- async/pipeline

#### chunked vs transducers

- Branch impl
   B
- impl contains iteration logic
- can't write only the chunked branch

- Branch impl
- impl contains
   completion logic
- **can** write only the transducer

#### Oh the irony!

=> (class (seq (sequence (map inc) (list 1 2 3)))
clojure.lang.ChunkedCons

## Transducers enable reuse

Bonus: they are more efficient

#### What's a transducer?

### Formally

A finite-state transducer (FST) is a finite-state machine with two memory tapes, following the terminology for Turing machines: an input tape and an output tape. This contrasts with an ordinary finitestate automaton, which has a single [input] tape.



The Free Encyclopedia

#### Informally

- An automaton is a predicate on sequences
- A transducer is an arbitrary transformation of sequences to sequences

#### Clojure transducers

- Take the « output tape » returns the « input tape »
- A « tape » is a reducing function: (f acc x) to write x

What's **in** a transducer?

#### The identity transducer

(fn [rf] ; the downstream reducing function
 (fn ; the transformed reducing function
 ([] (rf)) ; init
 ([acc] (rf acc)) ; complete
 ([acc x] (rf acc x)))) ; step



#### Caution: Bughazard



- init (0 arg) may not even be called!
   (e.g. sequence or 4-arg transduce)
- the only sensible thing to do is (rf)
- corollary #1: the accumulator shouldn't be touched
- corollary #2: mutable state



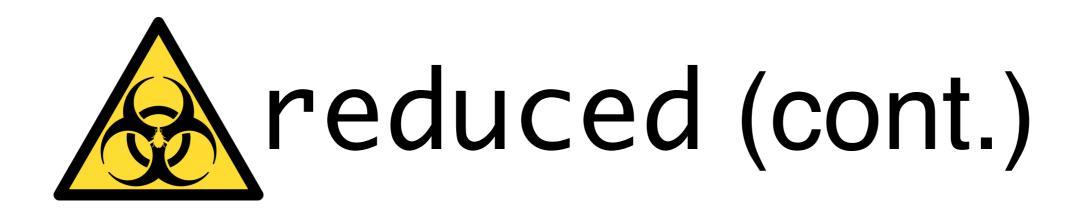
- completion (1 arg) MUST call downstream completion (eg (rf acc))
- step (2 args) MUST NOT call completion
  - just signal it... with **reduced**!

### Nobody expects reduced!



### reduced

- think twice about each (rf acc x) call, do you want to:
  - propagate?
  - stop no matter what? (e.g. take at 0 and downstream reduced)
  - wrap because you have nested reduces
- use reduced?, ensure-reduced and unreduced.



- (rf acc x) may be called from completion (flushing state)
  - → don't forget to unreduced before (rf acc)
- Corollary #3: if you get a reduced, make sure to not call step (2 args) again



- A transducer is a stateful process, not a reducing function
- Root of all transducer quirks!

```
(defprotocol Tape
  (put! [tape x]
    "writes x and returns false if tape is cut")
  (cut! [tape] "terminates tape"))
```



#### Implement with care

I'm pretty sure I messed up...

#### xforms: The Lost Levels

- my style changed to use transducers a lot, mainly (into dst xform src)\*
- some were missing: behold xforms!
- **xform** denotes a transducer in clojure.core
- <u>github.com/cgrand/xforms</u>
- clj, cljs and self-hosted cljs

\*filterv, mapv are legacy.



#### xforms transducers

- regular ones: partition (1 arg), reductions, for, takelast, drop-last, window and window-by-time
- higher-order ones: by-key, multiplex, transjuxt, partition (2+ args)
- aggregating ones (1-item out): reduce, into, transjuxt, last, count, avg, sd, min, minimum, max, maximum, str

#### x/for

- Either a drop-in replacement for for but returns an eduction (non persistent collection)
- Or a transducer when first coll is %

```
(reduce +
  (x/for [x (range 100)
        y (range x)
        :let [x+y (+ x y)]
        :when (even? x+y)]
        x+y))
```

#### x/partition as usual

=> (sequence (x/partition 3) (range 10))
([0 1 2] [3 4 5] [6 7 8])

=> (sequence (x/partition 3 2) (range 10))
([0 1 2] [2 3 4] [4 5 6] [6 7 8])

=> (sequence (x/partition 3 2 [:pad]) (range 10))
([0 1 2] [2 3 4] [4 5 6] [6 7 8] [8 9 :pad])

#### x/partition new tricks

=> (sequence (x/partition 3 (x/reduce +))
 (range 10))
(3 12 21)

#### x/partition two faces

- A transducer
- A transducing context
- At the same time
- Allows to perform computations in one pass

x/by-key

• My favorite xform!

(defn my-group-by [kfn coll] ; beats core/group-by
 (x/into {} (x/by-key kfn (x/into [])) coll))

- Specialized support throughout xforms for handling pairs without alloc
- x/into drop-in replacement for into but with kv-support
- x/into 1-arg returns a transducer

#### 1-pass rollup (advanced)

```
(defn rollup [dimensions valfn]; recursive aggregation (sum)
  (let [[dim & dims] (reverse dimensions)]
    (reduce
      (fn [xform dim]
        (comp
          (x/by-key dim xform)
          (x/transjuxt
            {:detail (x/into {})
             :total (comp x/vals (map :total) (x/reduce +))})))
      (comp (x/by-key dim (map valfn))
        (x/transjuxt
          {:detail (x/into {})
           :total (comp x/vals (x/reduce +))}))
      dims)))
```

#### Advanced example

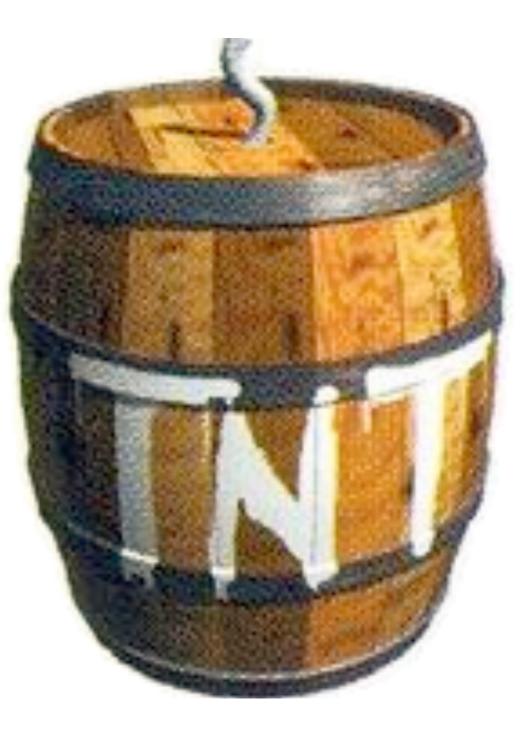
=> (into {} (rollup [:continent :country] :population)
 [{:continent "Europe" :country "France" :population 66}
 {:continent "Europe" :country "Germany" :population 80}
 {:continent "Europe" :country "Belarus" :population 9}
 {:continent "North-America" :country "USA" :population 319}
 {:continent "North-America" :country "Canada" :population 35}])
{:detail
 {"Europe"

{:detail {"France" 66, "Germany" 80, "Belarus" 9}, :total 155},
 "North-America" {:detail {"USA" 319, "Canada" 35}, :total 354}},
:total 509}

## Bringing transducers to new horizons

#### Powderkeg

- Transducers on Apache Spark
- As easy as: (keg/rdd src-rdd (map inc))
- Develop and test without Spark
- github.com/HCADatalab/powderkeg



#### TL;DR

- Transducers unleash reuse (Efficiency is icing on the cake)
- Writing your own is full of pitfalls: compose, compose
- Give xforms a try! (and Powderkeg too if Spark is your thing)

#### Thank you!

@cgrand

github.com/cgrand