

ClojureScript

for the web

Michiel Borkent

[@borkdude](https://twitter.com/borkdude)

Øredev, November 6th 2014




Michiel Borkent ([@borkdude](https://twitter.com/borkdude))

- Clojure(Script) developer at **FINALIST**
open IT oplossingen
- Clojure since 2009
- Former lecturer, taught Clojure


Agenda


- Why ClojureScript?
- The Language
- The Ecosystem

Warning






William Morgan
@wm



 Follow

i love functional programming. it takes smart people who would otherwise be competing with me and turns them into unemployable crazies


 Reply  Retweeted  Favorite ... More

RETWEETS


923

FAVORITES

875



8:53 PM - 30 Dec 2009



Reply to @wm

Why ClojureScript?



Current status

- JavaScript is everywhere, but not a robust and concise language - [wat](#)

Requires discipline to only use "the good parts"

- JavaScript is taking over: UI logic from server to client
- JavaScript is not going away in the near future
- Advanced libraries and technologies exist to optimize JavaScript: (Google Closure, V8)

Out of the Tar Pit

Ben Moseley

ben@moseley.name

Peter Marks

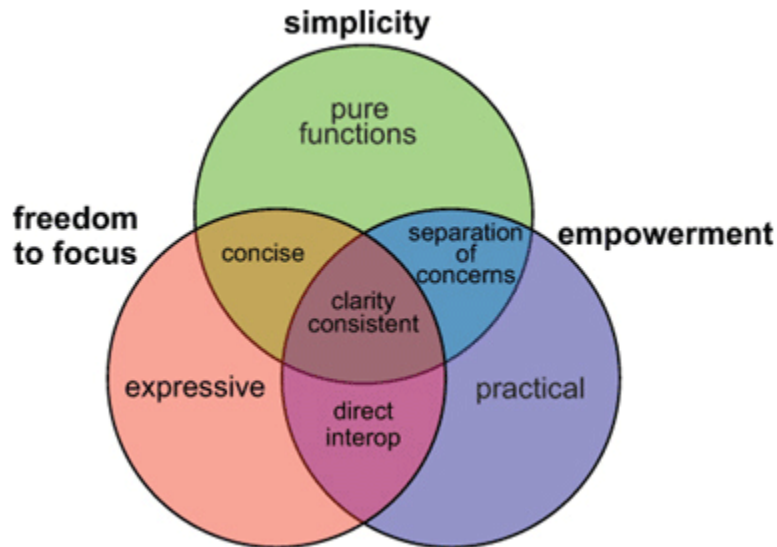
public@indigomail.net

February 6, 2006

tl;dr:

- complexity is biggest problem in software
- mutability + control: more state, more complexity
- immutability + FP: less state, less complexity

Clojure(Script) promotes



source: <http://www.drdobbs.com/architecture-and-design/the-clojure-philosophy/240150710>

ClojureScript?

- Released June 20th 2011
- Client side story of Clojure ecosystem
- Serves Clojure community:
 - 50%* of Clojure users also use ClojureScript
 - 93%** of ClojureScript users also use Clojure
- ClojureScript targets JavaScript by adopting Google Closure
 - libraries: `goog.provide/require` etc.
 - optimization: dead code removal

*<http://cemerick.com/2013/11/18/results-of-the-2013-state-of-clojure-clojurescript-survey/>

** <http://blog.cognitect.com/blog/2014/10/24/analysis-of-the-state-of-clojure-and-clojurescript-survey-2014>

The Language

Such parens...

$f(x) \rightarrow (f\ x)$

JavaScript - ClojureScript

<code>no implementation</code>	<code>(ns my.library (:require [other.library :as other]))</code>
<code>var foo = "bar";</code>	<code>(def foo "bar")</code>
<code>// In JavaScript // locals are mutable function foo(x) { x = "bar"; }</code>	<code>;; this will issue an error (defn foo [x] (set! x "bar"))</code>

JavaScript - ClojureScript

```
if (bugs.length > 0) {  
  return 'Not ready for release';  
} else {  
  return 'Ready for release';  
}
```

```
(if (pos? (count bugs))  
  "Not ready for release"  
  "Ready for release")
```

```
var foo = {bar: "baz"};  
foo.bar = "baz";  
foo["abc"] = 17;
```

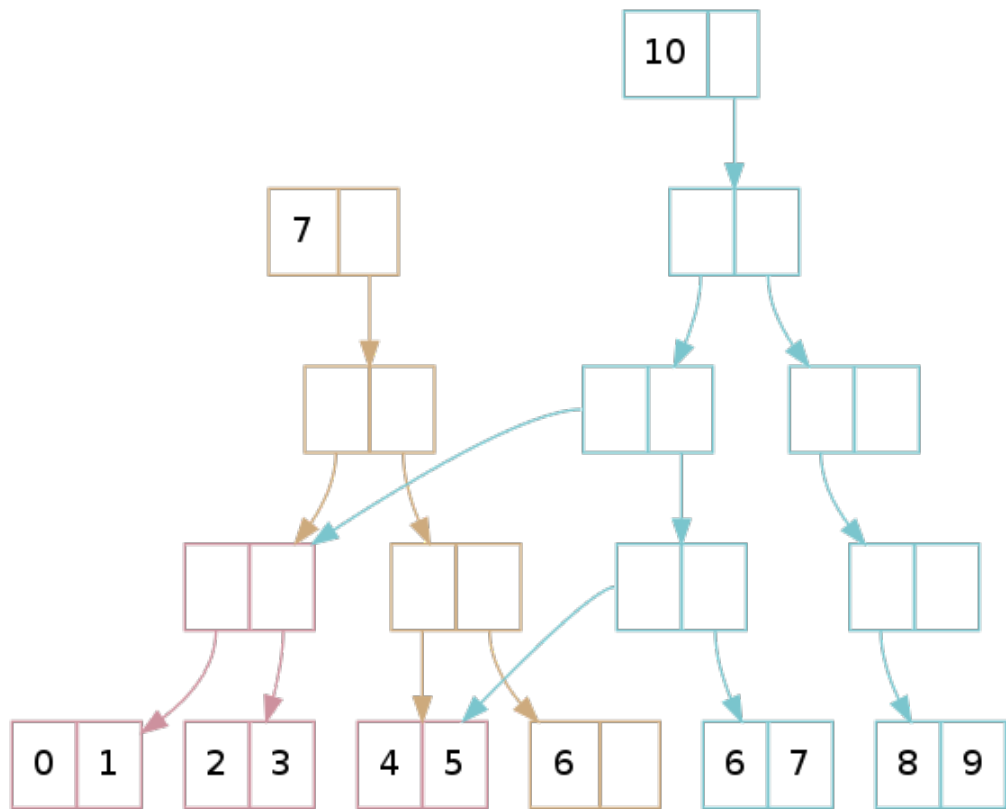
```
(def foo (js-obj "bar" "baz"))  
(set! (.bar foo) "baz")  
(aset foo "abc" 17)
```

Core language features

- persistent immutable data structures
- functional programming
- sequence abstraction
- isolation of mutable state (atoms)
- Lisp: macros, REPL
- `core.async`

Persistent data structures

```
(def v (vector))  
(def v [])  
(def v [1 2 3])  
(conj v 4) ;; => [1 2 3 4]  
(get v 0) ;; => 1  
(v 0) ;; => 1
```



source: <http://hypirion.com/musings/understanding-persistent-vector-pt-1>

Persistent data structures

```
(def m (hash-map))  
(def m {})  
(def m {:foo 1 :bar 2})  
(conj m [:baz 3])  
;; => {:foo 1 :bar 2 :baz 3}  
(assoc m :foo 2) ;; => {:foo 2 :bar 2}  
(get m :foo) ;; = > 2  
(m :foo) ;; = > 2  
(dissoc m :foo) ;; => {:bar 2}
```

Functional programming

```
(def r (->>
      (range 10)      ;; (0 1 2 .. 9)
      (filter odd?)   ;; (1 3 5 7 9)
      (map inc)))      ;; (2 4 6 8 10)
;; r is (2 4 6 8 10)
```

Functional programming

```
:: r is (2 4 6 8 10)
```

```
(reductions + r)
```

```
:: => (2 6 12 20 30)
```

```
(reduce + r)
```

```
:: => 30
```

Sequence abstraction

Data structures as seqs

(`first` [1 2 3]) ;;=> 1

(`rest` [1 2 3]) ;;=> (2 3)

General seq functions: `map`, `reduce`, `filter`, ...

(`distinct` [1 1 2 3]) ;;=> (1 2 3)

(`take` 2 (`range` 10)) ;;=> (0 1)

See <http://clojure.org/cheatsheet> for more

Sequence abstraction

Most seq functions return lazy sequences:

```
(take 2 (map
```

```
  (fn [n] (js/alert n) n)
```

```
  (range)))
```

side effect

infinite lazy sequence of numbers

Isolation of state

one of possible
pre-React
patterns

```
(def app-state (atom []))  
  
(declare rerender)  
  
(add-watch app-state ::rerender  
  (fn [k a o n]  
    (rerender o n)))
```

function called
from event
handler

```
(defn add-todo [text]  
  (let [tt (.trim text)]  
    (if (seq tt)  
      (swap! app-state conj  
        {:id (get-uuid)  
         :title tt  
         :completed false}))))
```

new todo

adapted from: <https://github.com/dfuenzalida/todo-cljs>

Lisp: macros

```
(map inc  
  (filter odd?  
    (range 10))))
```

thread last macro

```
(->>  
  (range 10)  
  (filter odd?)  
  (map inc))
```

Lisp: macros

```
(macroexpand  
  '(->> (range 10) (filter odd?)))
```

```
; ; => (filter odd? (range 10))
```

```
(macroexpand  
  '(->> (range 10) (filter odd?) (map inc)))
```

```
; ; => (map inc (filter odd? (range 10)))
```


Lisp: macros

JVM Clojure:

```
(defmacro defonce [x init]
  `(when-not (exists? ~x)
    (def ~x ~init)))
```

ClojureScript:

```
(defonce foo 1)
(defonce foo 2) ;; no effect
```

notes:

- macros must be written in JVM Clojure
- are expanded at compile time
- generated code gets executes in ClojureScript

LISP: Browser REPL (weasel)

todos



What needs to be done?

Double-click to edit a todo

```
todo-cljs.brepl=> (brepl)
<< started Weasel server on ws://0.0.0.0:9005 >>
Type `:cljs/quit` to stop the ClojureScript REPL
nil
cljs.user=>
```

core.async

```
(def ch (chan))
```

```
(go (loop []  
      (if-let [msg (<! ch)]  
        (do  
          (.log js/console msg)  
          (recur))  
        (println "terminating loop...")))))
```

```
(events/listen (by-id "button1") EventType.CLICK  
               #(put! ch "hello world!"))
```

```
(events/listen (by-id "button2") EventType.CLICK  
               #(close! ch))
```

The Ecosystem

Debugging

Source maps let you debug ClojureScript directly from the browser

The screenshot displays a web browser's developer tools interface, specifically the 'Sources' panel, which is used for debugging JavaScript code. The left sidebar shows a file explorer view of the project structure, including folders like 'closure', 'cognitect', 'com/cognitect', 'drag', and 'webinars', and files like 'main.cljs' and 'main.js'. The main pane shows the source code of 'main.cljs' with line 16 highlighted: `(and (< (Math/abs (- x (:x black-hole-pos))) 50) (< (Math/abs (- y (:y black-hole-pos))) 50))`. The bottom pane shows the 'Call Stack' with frames for 'close_QMARK_', '(anonymous function)', 'goog.events.fireListener', 'goog.events.handleBrowserEvent_', and '(anonymous function)'. The 'Scope Variables' pane shows 'this: Window', 'x: 197', and 'y: 116'.

```
12 (def black-hole-pos {:x 400 :y 400})
13 (def draggable (atom {:x 100 :y 100 :alive? true}))
14
15 (defn close? [x y]
16   (and (< (Math/abs (- x (:x black-hole-pos))) 50)
17        (< (Math/abs (- y (:y black-hole-pos))) 50)))
18
19 (defn get-client-rect [evt]
20   (let [r (.getBoundingClientRect (.-target evt))]
21     {:left (.-left r), :top (.-top r)}))
22
23 (defn draggable-button []
24   (let [mouse-move-handler
```

Line 16, Column 1

Scope Variables Watch Expressions

Call Stack

- close_QMARK_ main.cljs:16
- (anonymous function) main.cljs:29
- goog.events.fireListener events.js:741
- goog.events.handleBrowserEvent_ events.js:862
- (anonymous function) events.js:276

Local

- this: Window
- x: 197
- y: 116

Global Window

Leiningen

- Used by 98% of Clojure users
- Clojure's Maven
- Managing dependencies
- Running a REPL
- Packaging and deploying
- Plugins:
 - lein cljsbuild
 - lein figwheel



```
(defproject example "0.1.0-SNAPSHOT"
  :description "FIXME: write this!"
  :url "http://example.com/FIXME"

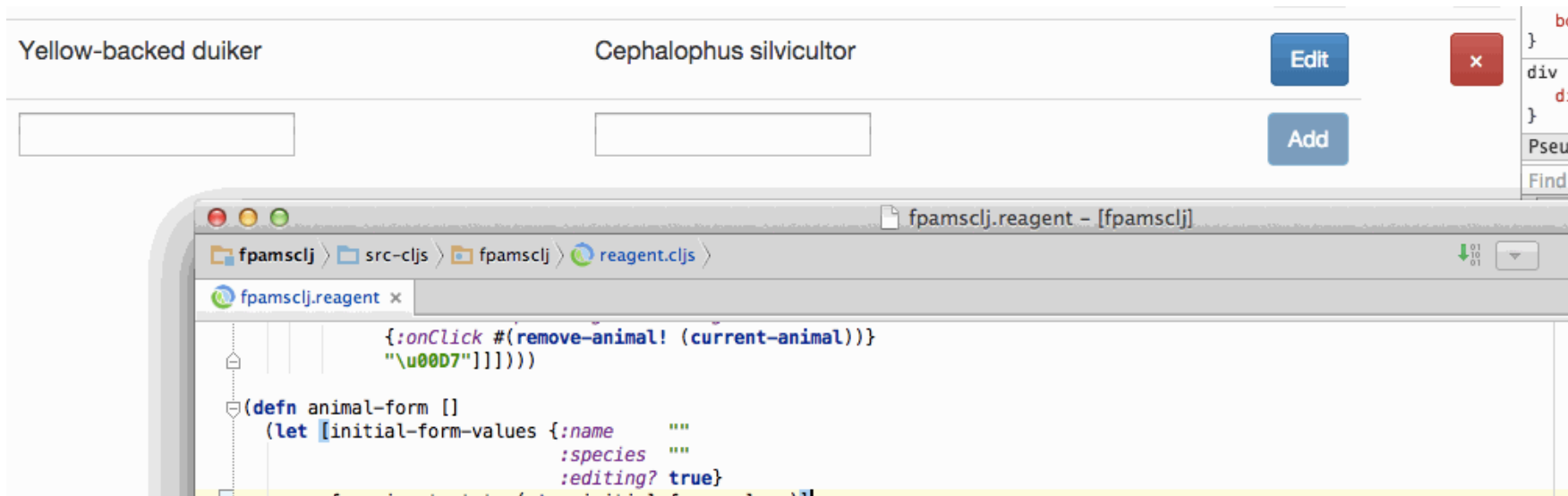
  :dependencies [[org.clojure/clojure "1.6.0"]
                 [org.clojure/clojurescript "0.0-2311"]]

  :plugins [[lein-cljsbuild "1.0.4-SNAPSHOT"]]

  :source-paths ["src"]

  :cljsbuild {:builds [{:id "example"
                        :source-paths ["src"]
                        :compiler {
                          :output-to "example.js"
                          :output-dir "out"
                          :optimizations :none
                          :source-map true}}]})
```

figwheel: live code reloading



Editors

Most popular:

- Emacs
- Cursive Clojure (IntelliJ)
- Vim + vim-fireplace
- Light Table
- Counterclockwise (Eclipse)

```
(go (let [response
          (<! (http/delete (str "/animals/"
                             (:id a)))]
      (if (= (:status response)
              200)
          (swap! animals-state disj a))))

(defn update-animal! [a]
  (go (let [response
            (<! (http/put (str "/animals/" (:id a))
                          {:edn-params a}))
            updated-animal (:body response)]
      (swap! animals-state
        (fn [old-state]
          (conj
            (set (filter (fn [other]
                          (not= (:id other)
                                (:id a)))
                          old-state))
            updated-animal))))))

;;; end crud operations

(defn field-input-handler
  "Returns a handler that updates value in atom map,
  under key, with value from onChange event"
  [atom key]
  (fn [e]
    (swap! atom
      assoc key
      (.. e -target -value))))

(defn input-valid? [atom]
  (and (seq (→ @atom :name))
       (seq (→ @atom :species))))

(defn editable-input [atom key]
  (if (:editing? @atom)
      [input {:type "text"}
        ]
      ))

--:--- crud.cljs 27% of 4.7k (78,29) (Clojure MRev , 80co
```

cljs.core.typed

```
(ns foo)

(ann parse-int [string -> number])

(defn parse-int [s]
  (js/parseInt s))

(parse-int 3)
```

Function foo/parse-int could not be applied to arguments:

Domains:
string

Arguments:

(clojure.core.typed/Val 3)

Ranges:
number

in: (foo/parse-int 3)

ClojureScript interfaces to React

Talk today @ 17:40-18:20



initial commit

swannodette authored on Dec 3, 2013



cfb4639



Initial version

holmsand authored on Dec 16, 2013



12566ce



Example

hide

Seconds Elapsed: 75

```
(defn timer-component []  
  (let [seconds-elapsed (atom 0)]  
    (fn []  
      (js/setTimeout #(swap! seconds-elapsed inc) 1000)  
      [:div  
        "Seconds Elapsed: " @seconds-elapsed])))
```

Community

[Google Groups](#)

IRC: #clojure and #clojurescript on [freenode](#)

[Planet Clojure](#)

[Reddit](#)

[Meetup.com](#)

Interesting libraries

- [cljs-http](#): ajax + core.async
- [cljx](#): code sharing between clj and cljs
- [cljs-test](#): unit testing
- [dommy](#): dom manipulation
- [crate](#): hiccup style HTML templating
- [sente](#): websockets + core.async
- [transit](#): conveying values between languages
- [datascript](#): functional database in cljs
- [garden](#): css generation from Clojure

Thank you!

Trash can



Good Parts Reconsidered


- I stopped using `new` years ago.
- I have stopped using `Object.create`.
- I have stopped using `this`.
- I have stopped using `null`.
- I have stopped using falsiness.






Brief history of ClojureScript

June 20th 2011: first release of ClojureScript

initial commit [Browse code](#)

 **richhickey** authored on Jun 3, 2011 0 parents commit 515900f9762102987bda7d53b919dafc0b6c0580

 Showing **2 changed files** with **833 additions** and **0 deletions**. Unified Split

422  cljs.org <> 

...	...	@@ -0,0 +1,422 @@
	1	+* ClojureScript
	2	+* Rationale

Brief history of ClojureScript

Early 2012: first release of lein cljsbuild

Leiningen plugin to make ClojureScript development easy

98% of Clojure users use Leiningen

Possible optimization levels include

- : whitespace removes comments and whitespace
- : simple renames local variables to compress JavaScript
- : advanced: aggressively renames and strips away unused

Brief history of ClojureScript

April 2012:

persistent data structures were ported



PersistentHashMap ported from Clojure ...

michalmarczyk authored on Apr 11, 2012 ➔ David Nolen committed on Apr 20, 2012



Light Table

June 2012

Funded as Kickstarter Project

Interactive, experimental IDE written in
ClojureScript, running on Node Webkit

Became open source early 2014

```
(defn move [me]
  (let [speed 5
        vx (cond
              (input? :left) (- speed)
              (input? :right) speed
              :else 0)]
    moved (update-in me [:x] + vx))
  (if (zero? vx)
    me
    (if-let [block (colliding? moved)]
      (let [block-edge (if (< vx 0)
                          (+ (:x block) (:w block) (:x me))
                          (- (:x block) (:x me)))]
        (assoc me :x block-edge))
      moved))))
```

```
(defn gravity [{:keys [vy y] :as me}]
  (let [g 0.5
        vy (or vy 0)
        neue-vy (+ vy g)
        dir (if (< neue-vy 0) :up :down)]
    moved (update-in me [:y] + neue-vy))
  (if-let [block (colliding? moved)]
    (let [block-edge (if (= dir :up)
                          (+ (:y block) (:h block) (:y me))
                          (- (:y block) (:y me)))]
      (assoc me :y block-edge
                :jumping (= dir :up)
                :vy 0))
    (-> moved
        (assoc :vy neue-vy))))
```

7,317

backers

\$316,720

pledged of \$200,000 goal

0

seconds to go



Project by

Chris Granger
San Francisco, CA

K First created · 4 backed

f **Chris Granger** 189 friends

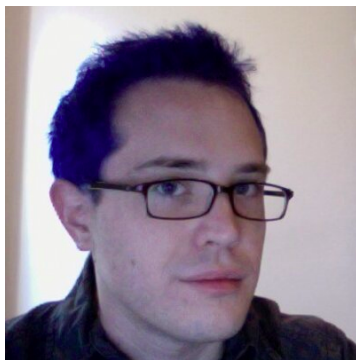
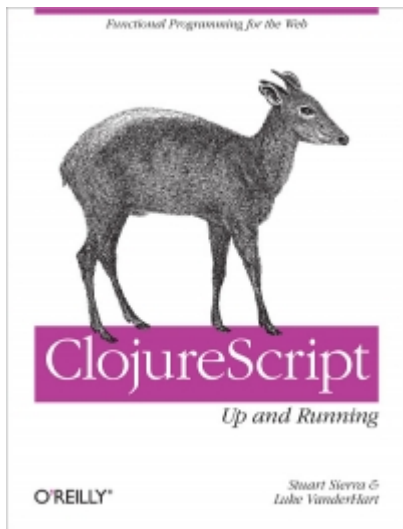
@ chris-granger.com

[See full bio](#)

[Contact me](#)

Brief history of ClojureScript

October 2012: ClojureScript Up and Running - O'Reilly



Brief history of ClojureScript

TRANSDUCERS ARE COMING

August 2014



Posted by Rich Hickey on August 6, 2014



David Nolen
@swannodette



Following

Transducers are a huge perf win for ClojureScript core.async, goodbye intermediate garbage for sequence-like ops

↩ Reply ↻ Retweet ★ Favorited ... More

RETWEETS

28

FAVORITES

22

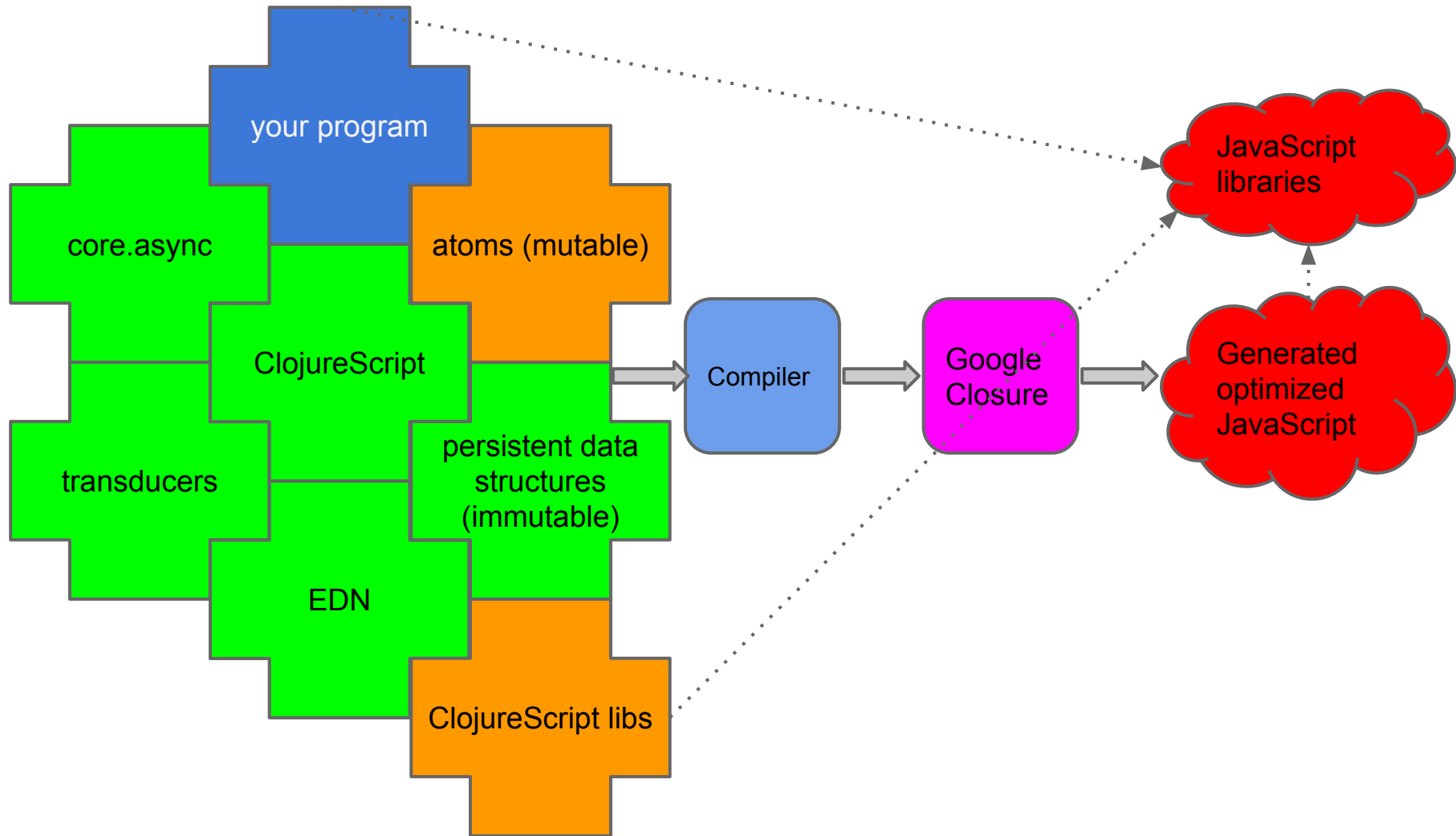


6:59 PM - 19 Aug 2014

Transducers

```
(->>  
  (range 10)  
  (filter odd?)  
  (map inc))
```

```
(def xform  
  (comp  
    (filter odd?)  
    (map inc)))  
  
;; lazy  
(sequence xform  
          (range 10))  
  
;; strict  
(into [] xform (range 10))
```

Lisp: macros

```
(if (< x 5)  
  :foo  
  (if (> x 10)  
    :bar  
    :baz) )
```

```
(cond (< x 5)      :foo  
      (> x 10)     :bar  
      :else        :baz)
```



cond macro

Frameworks

- Pedestal
- Hoplon
- Luminus (curated collection of libs)