

# **2021 VS CLOJURESCRIPT. STILL ALIVE?**

Alex Bykbaiev

# ABOUT



Alex Bykbaiev, Frontend Engineer at Hopin.

# AGENDA

- ClojureScript, what is it??
- Stuff I like in ClojureScript
- There is always a small fly in the ointment
- How and why we use ClojureScript in production
- Some resources to learn more about ClojureScript
- Q&A

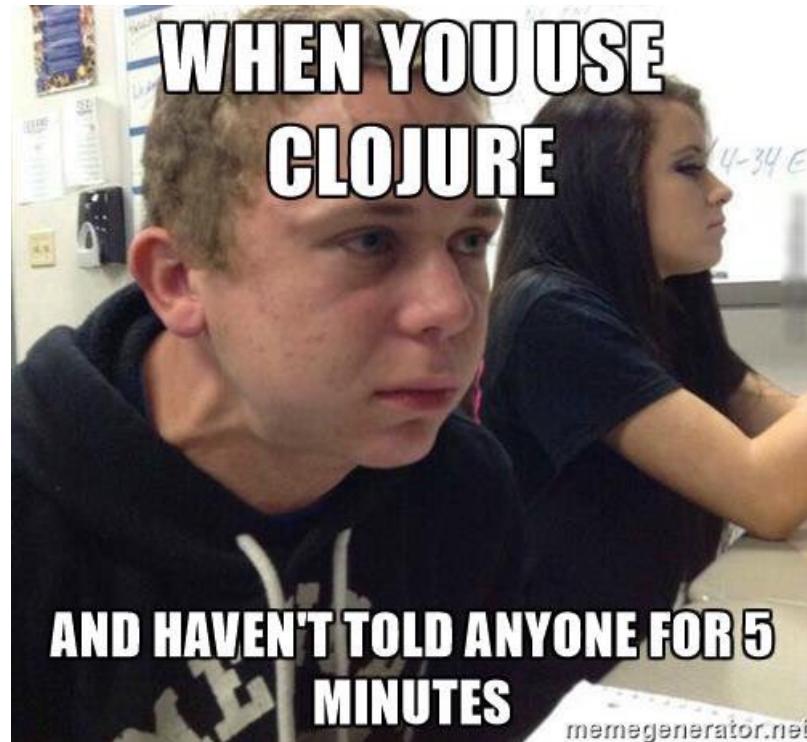
# CLOJURESCRIPT, WHAT IS IT??

Clojure is a dynamically typed Lisp for  
JVM.

ClojureScript is same for JavaScript.



# STUFF I LIKE IN CLOJURESCRIPT



# MANAGEMENT OF CONTROL FLOW



# REPL (READ-EVAL-PRINT LOOP)

The screenshot shows a development environment with two main panes. The left pane is a code editor titled 'r.h.results' containing Clojure code. The right pane is a terminal titled 'REPL Local: dev' showing the results of a Facebook API request.

**Code Editor (r.h.results):**

```
1(ns replexl.handler.results
2  (:require [clojure.repl :refer :all]
3            [clojure.pprint :refer [pprint]]
4            [replexl.env :as env]
5            [org.httpkit.client :as http]
6            [cheshire.core :as json]))
7
8(def q "zz top")
9
10@(http/request
11  {:method :get
12   :url "https://graph.facebook.com/v2.8/search"
13   :query-params
14   {"q" q
15    "fields" "id, name, fan_count"
16    "type" "page"
17    "access_token" env/fb-token}})
18
19(def mock-data
20  {:search "zztop"
21   :search-results
22   (repeat 3
23     {:page-name "ZZ Top"
24      :n-likes 3242
25      :last-post "Back to La Grange!"}))}
26
27(defn handle [req]
28  (let [;; FIXME
29    results-data mock-data]
30    [[:div.container
31     [:div.row
32      [:div.col-md-6.col-md-offset-3
33        [:div
34          [:h1 "Search results"]
35          [:div
36            [:p
37              (count (:search-results results-data)) " search results for "
38              [:b (:search results-data)] ":"]
39            [:div
40              (for [sr (:search-results results-data)]
41                [:div.panet.panet-default
42                  ...]]))])])])
```

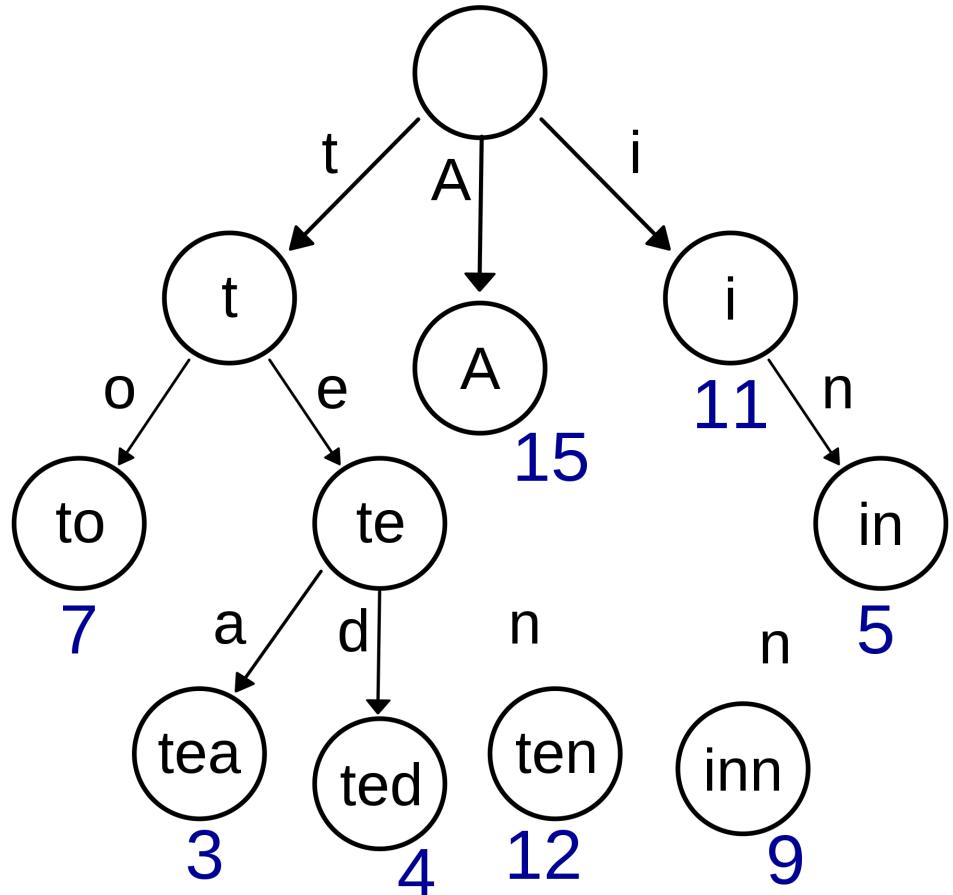
**Terminal (REPL Local: dev):**

```
:url "https://graph.facebook.com/v2.8/search",
:query-params {"q" "zz top",
               "fields" "id, name, fan_count",
               "type" "page",
               "access_token" "EAACEdEose0cBAEyo24ZBXJqmdYOS3
:body "{\"data\": [{\"id\": \"8888384363\", \"name\": \"ZZ Top\", \"fan_c
:headers {\"x-fb-trace-id\" "Awfr/MaaCyb",
          \"date\" "Fri, 18 Aug 2017 21:55:33 GMT",
          \"x-fb-rev\" "3237777",
          \"pragma\" "no-cache",
          \"vary\" "Accept-Encoding",
          \"etag\" "\"be0f046667709fc22b968bef002294393a1a641\"",
          \"x-fb-debug\" "7UuW60Xrj0f2uun5PdZ8ZXABFT3QuqiWzQ8j7VG/0ZSc
          \"expires\" "Sat, 01 Jan 2000 00:00:00 GMT",
          \"cache-control\" "private, no-cache, no-store, must-revalidat
          \"content-length\" "1062",
          \"facebook-api-version\" "v2.8",
          \"content-type\" "text/javascript; charset=UTF-8",
          \"x-app-usage\" "{\"call_count\":7, \"total_cputime\":3, \"total_
          \"content-encoding\" "gzip",
          \"access-control-allow-origin\" "*",
          \"connection\" "keep-alive"}, \"status\" 200}
```

## REPL-Based Development Demo

# COLLECTIONS

All collections in ClojureScript are persistent (immutability + structural sharing).



# LIST

List provides efficient access to the first elements  
(linked list under the hood).

```
1 (def list' '(1 2 3 4))  
2  
3 (let [xs '(1 2 3)      ;; (1 2 3)  
4       ys (cons 0 xs)] ;; (0 1 2 3)  
5   (identical? xs (rest ys)))  
6 => true
```

# VECTOR

Vector provides efficient index access to its elements.

```
1 (def vec' [1 2 3 4])  
2  
3 (vec' 0)  
4 => 1
```

# MAP

Map is a collection abstraction that allows you to store key/value pairs.

```
1 (def map' { :a 1 :b 2} )  
2  
3 (get map' :a)  
4 => 1  
5  
6 (map' :a)  
7 => 1  
8  
9 (:a map')  
10 => 1
```

# SET

Set is unordered collection of unique items.

```
1 (require '[clojure.set :as s])
2
3 (def set' #{1 2 3 10})
4
5 (set' 0)
6 => nil
7
8 (set' 10)
9 => 10
10
11 (filter #{"rose" "marigold" "hibiscus"}
12      ["orchid" "sunflower" "hibiscus" "daisy"
13      "gerbera" "lavender" "marigold"])
14 => ("hibiscus" "marigold")
15
```

# SET

Set is unordered collection of unique items.

```
5  (set' 0)
6 => nil
7
8  (set' 10)
9 => 10
10
11 (filter #{"rose" "marigold" "hibiscus"}
12      ["orchid" "sunflower" "hibiscus" "daisy"
13       "gerbera" "lavender" "marigold"])
14 => ("hibiscus" "marigold")
15
16 (s/union #{0 1 2} #{1 2 3})
17 => #{0 1 2 3}
18
19 (s/difference #{0 1 2} #{1 2 3})
```

# SET

Set is unordered collection of unique items.

```
12      [ "orchid" "sunflower" "hibiscus" "daisy"
13          "gerbera" "lavender" "marigold" ] )
14 => ("hibiscus" "marigold")
15
16 (s/union #{0 1 2} #{1 2 3})
17 => #{0 1 2 3}
18
19 (s/difference #{0 1 2} #{1 2 3})
20 => #{0}
21
22 (s/intersection #{0 1 2} #{1 2 3})
23 => #{1 2}
24
25 (s/select odd? #{0 1 2})
26 => #{1}
```

# SET

Set is unordered collection of unique items.

```
15
16 (s/union #{0 1 2} #{1 2 3})
17 => #{0 1 2 3}
18
19 (s/difference #{0 1 2} #{1 2 3})
20 => #{0}
21
22 (s/intersection #{0 1 2} #{1 2 3})
23 => #{1 2}
24
25 (s/select odd? #{0 1 2})
26 => #{1}
27
28 (filter odd? #{0 1 2})
29 => (1)
```

# SEQ

```
1  (map inc '(0 1 2))
2  => (1 2 3)
3
4  (map inc [0 1 2])
5  => (1 2 3)
6
7  (map inc #{0 1 2})
8  => (1 2 3)
9
10 (map (comp inc second) {:_a 0 :_b 1 :_c 2})
11 => (3 2 1)
12
13 (into [] '(0 1 2))
14 => [0 1 2]
15
```

# SEQ

```
7  (map inc #'(0 1 2))
8 => (1 2 3)
9
10 (map (comp inc second) {:a 0 :b 1 :c 2})
11 => (3 2 1)
12
13 (into [] '(0 1 2))
14 => [0 1 2]
15
16 (into [] {:a 1 :b 2})
17 => [[:b 2] [:a 1]]
18
19 (concat {:a 1 :b 2} {:b 3})
20 => [[:b 2] [:a 1] [:b 3]]
21
```

# SEQ

```
9
10 (map (comp inc second) {:a 0 :b 1 :c 2} )
11 => (3 2 1)
12
13 (into [] '(0 1 2))
14 => [0 1 2]
15
16 (into [] {:a 1 :b 2})
17 => [[:b 2] [:a 1]]
18
19 (concat {:a 1 :b 2} {:b 3})
20 => [[:b 2] [:a 1] [:b 3]]
21
22 (into {} (concat {:a 1 :b 2} {:b 3} ))
23 => {[b 3] [a 1]}
```

## MACROPROGRAMMING

Clojure is also a dialect of LISP (LISt Processing language) by design. The aim was to make the language more extensible.

Clojurescript allows you to write code that writes code (homoiconicity).

# MACROS

```
1 (defmacro infix
2   [infix]
3     (list (second infix) (first infix) (last infix)))
4
5 (+ 1 1)
6 => 2
7
8 (infix (1 + 1))
9 => 2
```

# MACROS FOR STYLED COMPONENTS

JS

```
1 const SubItemNumber = styled.span`  
2   flex: 0 0 ${polished.rem(54)};  
3   margin-left: auto;  
4   text-align: end;  
5   overflow: hidden;  
6   text-overflow: ellipsis;  
7 `;  
8 ...  
9 <SubItemNumber>  
10   123  
11 </SubItemNumber>
```

# MACROS FOR STYLED COMPONENTS

## CLJS

```
1 (defstyled sub-item-number
2   :span
3   {:flex (str "0 0 " (polished/rem 54)))
4   :margin-left "auto"
5   :text-align "end"
6   :overflow "hidden"
7   :text-overflow "ellipsis"} )
8 ...
9 (sub-item-number "123")
10 ...
11 (js/React.createElement
12   (-> component meta :react-component)
13   #js {}
14   "123")
```

# THREADING MACROS

## Thread-first macro

```
1 (def user {:first-name "Alex"
2                 :last-name   "Bykbaiev"} )
3 => {:first-name "Alex"
4       :last-name "Bykbaiev"}
5
6 (-> user
7       (assoc :full-name
8                 (str (:first-name user)
9                       " "
10                      (:last-name user)))
11       (assoc :mental-age 54))
12 => {:first-name "Alex"
13       :last-name   "Bykbaiev"
14       :full-name   "Alex Bykbaiev"
15       :mental-age 54}
```

# THREADING MACROS

## Thread-first macro

```
1  (der user {:first-name "Alex"
2                  :last-name "Bykbaiev" } )
3 => {:first-name "Alex"
4      :last-name "Bykbaiev"}
5
6  (-> user
7          (assoc :full-name
8                  (str (:first-name user)
9                        " "
10                         (:last-name user))))
11         (assoc :mental-age 54))
12 => {:first-name "Alex"
13      :last-name "Bykbaiev"
14      :full-name "Alex Bykbaiev"
15      :mental-age 54}
16
```

# THREADING MACROS

## Thread-first macro

```
9          " "
10         (:last-name user)))
11      (assoc :mental-age 54))
12 => {:first-name "Alex"
13      :last-name "Bykbaiev"
14      :full-name "Alex Bykbaiev"
15      :mental-age 54}
16
17 (assoc (assoc user
18             :full-name
19             (str (:first-name user)
20                  " "
21                  (:last-name user))))
22             :mental-age
23             54)
```

# THREADING MACROS

## Thread-last macro

```
1  (def numbers [1 2 3 4 5 6 7 8 9 0])
2
3  (take 2 (filter odd? (map inc numbers)))
4 => (3 5)
5
6  (->> numbers
7      (map inc)
8      (filter odd? )
9      (take 2))
10 => (3 5)
```

# THREADING MACROS

## Thread-as macro

```
1 (def points {:vals [1 2 3 4 5 6 7 8 9 0]} )  
2  
3 (take 2 (filter odd? (map inc (get points :vals))))  
4 => (3 5)  
5  
6 (as-> points $  
7   (get $ :vals)  
8   (map inc $)  
9   (filter odd? $)  
10  (take 2 $))  
11 => (3 5)
```

# THREADING MACROS

## Thread-some macros

```
1 (def user { :name "Alex" })
2
3 (-> user
4     (get :mental-age) ;; => nil
5     (str " years"))
6 => " years"
7
8 (some-> user
9     (get :mental-age)
10    (str " years")))
11 => nil
```

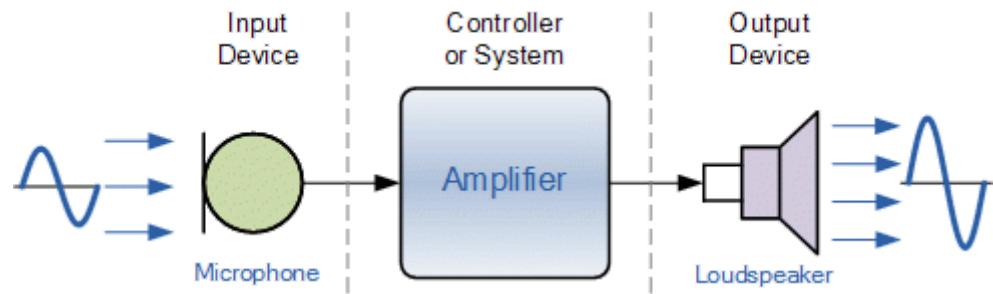
# THREADING MACROS

## Thread-cond macros

```
1 (def month-of-birth "Jan")
2
3 (def motivating-Mars? true)
4
5 (def equalizing-Saturn? false)
6
7 (cond-> []
8   (= month-of-birth "Jan") (conj "Capricornus")
9   (= month-of-birth "Feb") (conj "Aquarius")
10  ; ; ...
11  motivating-Mars? (conj "You would be unstoppable as"
12  equalizing-Saturn? (conj "Strike a balance between"
13 => [ "Capricornus" "You would be unstoppable as the moon for"
```

# TRANSDUCERS

A transducer is a composable higher-order reducer. It takes a reducer as input, and returns another reducer.



```
1 (def xs [1 2 3 4 5 6 7 8 9 0])
2
3 (->> xs
4     (map inc)
5     (filter odd? )
6     (take 2))
7 => (3 5)
8
9 ; ; map = transform => reducer => reducer
10
11 (def tr (comp (map inc)
12                 (filter odd? )
13                 (take 2)))
14
15 (reduce (tr conj) [] xs)
```

```
8
9  ;; map = transform => reducer => reducer
10
11 (def tr (comp (map inc)
12                  (filter odd? )
13                  (take 2)))
14
15 (reduce (tr conj) [ ] xs)
16 => [3 5]
17
18 (into [ ] tr xs)
19 => [3 5]
20
21 (transduce tr conj [ ] xs)
22 => [3 5]
```

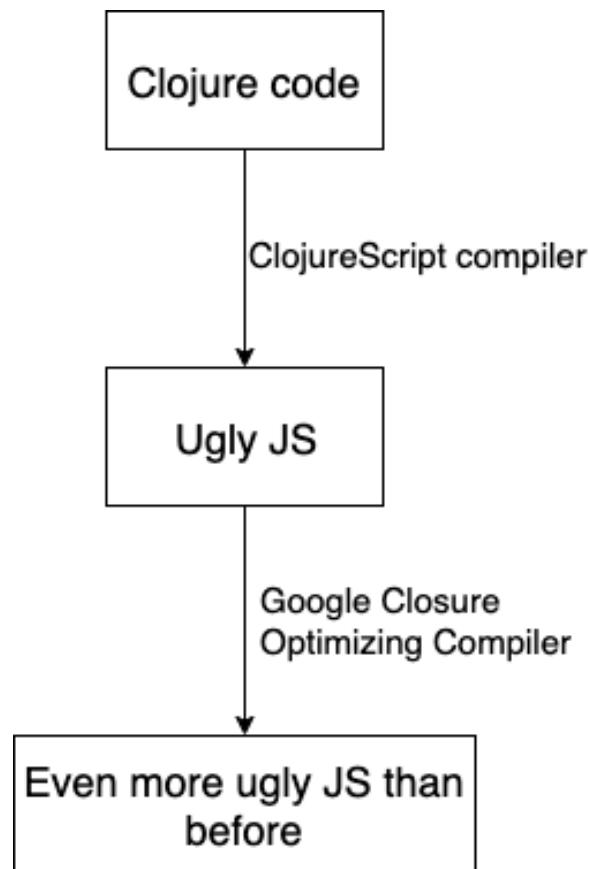
# TRANSDUCERS

## Details:

Returns a lazy sequence of the first `n` items in `coll`. Returns all the items if there are fewer than `n`.

Returns a stateful transducer when no collection is provided.

# PERFORMANCE



```
1 (def xs [1 2 3 4 5 6 7 8 9 0])
2
3 (->> xs
4     (map inc)
5     (filter odd? )
6     (take 2))
```

```
1 cljs.user.xs = new cljs.core.PersistentVector(
2     null, 10, 5, cljs.core.PersistentVector.EMPTY_NODE, [(1
3 )];
4 cljs.core.take.call(
5     null,(2),cljs.core.filter.call(
6         null,cljs.core.odd_QMARK_,cljs.core.map.call(
7             null,cljs.core.inc,cljs.user.xs
8             )
9         )
10 );
```

# ECOSYSTEM

- Leiningen ~ npm (dependencies + scripts)
- shadow-cljs ~ module bundler
- Reagent/Om/Rum - React wrappers
- Re-frame - Redux on steroids



Reagent



# THERE IS ALWAYS A FLY IN THE OINTMENT



# BUNDLE SIZE

**Module: :main [JS: 474.49 KB] [GZIP: 124.56 KB]**

Group	Optimized	%
+ org.clojure/clojurescript @ mvn: 1.10.879	192.3 KB	40.6 %
+ react-dom @ npm: 16.9.0	111.25 KB	23.5 %
+ <b>src</b>	36.83 KB	7.8 %
+ org.clojure/google-closure-library @ mvn: 0.0-20201211-3e6c510d	36.55 KB	7.7 %
+ reagent @ mvn: 0.9.1	19.75 KB	4.2 %
+ org.clojure/tools.reader @ mvn: 1.3.6	16.03 KB	3.4 %
+ cljs-ajax @ mvn: 0.7.3	15.82 KB	3.3 %
+ re-frame @ mvn: 0.10.7	13.53 KB	2.9 %
+ bidi @ mvn: 2.1.5	8.8 KB	1.9 %
+ react @ npm: 16.9.0	6.57 KB	1.4 %
+ scheduler @ npm: 0.15.0	5.44 KB	1.1 %
+ Generated Files	3.14 KB	0.7 %
+ kibu/pushy @ mvn: 0.3.8	2.83 KB	0.6 %
+ com.cognitect/transit-js @ mvn: 0.8.874	2.15 KB	0.5 %
+ object-assign @ npm: 4.1.1	978	0.2 %
+ day8.re-frame/http-fx @ mvn: v0.2.0	940	0.2 %
+ com.cognitect/transit-cljs @ mvn: 0.8.269	348	0.1 %

RealWorld Comparison 2020

# PERFORMANCE



```
1 (require 'goog.object)
2
3 (def at-runtime (clj->js {:a {:b {:c 1}}}))
4 => #js {:a #js {:b #js {:c 1}}}
5
6 (def at-compile-time #js {:a #js {:b #js {:c 1}}})
7 => #js {:a #js {:b #js {:c 1}}}
8
9 (-> heavy-js-object
10     (js->clj :keywordize-keys true)
11     (get-in [:a :b :c]))
12
13 (goog.object/getValueByKeys heavy-js-object "a" "b" "c")
```

*"Note that js->clj is not optimized for speed and the transit.cljs library is recommended for parsing large amounts of JSON data."*

# LAZINESS

```
1 (def input [{ "ab" [1 0] } {"bc" [1 1]} {"de" [1 0]}
2           {"ab" [2 1]} {"ab" [0 1]}])
3
4 ;; Need to get {"ab" [3 2] "bc" [1 1] "de" [1 0]}
5
6 (defn merge-hashes-naive [coll]
7   (apply merge-with (fn [a b] (map + a b)) coll))
8
9 (merge-hashes-naive input)
10 => {"ab" (map + [0 1] (map + [2 1] [1 0]))) ...}
11
12 (merge-hashes-naive large-input)
13 => java.lang.StackOverflowError
```

Clojure lazy evaluation and stack overflow exceptions

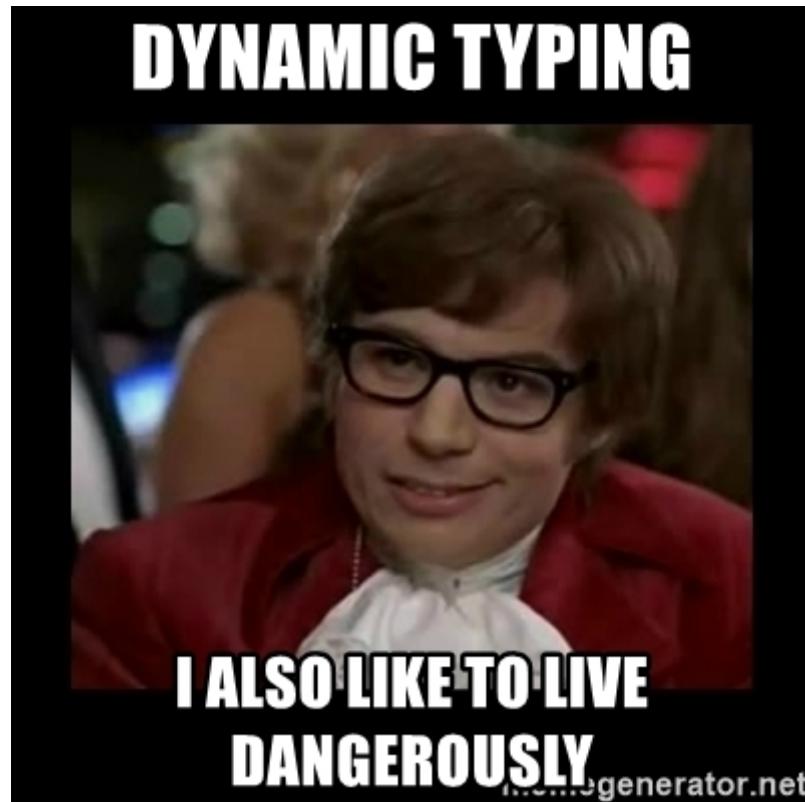
# AUTHORITY IN OPEN SOURCE



*"As a user of something open source you are not thereby entitled to anything at all. You are not entitled to contribute. You are not entitled to features. You are not entitled to the attention of others. You are not entitled to this explanation."*

**Open Source is Not About You**

# DYNAMIC TYPE SYSTEM



## LISP CURSE

*"Lisp is so powerful that problems which are technical issues in other programming languages are social issues in Lisp."*

The Lisp Curse

# SUPPORT

 [microsoft / TypeScript](#) Public

[Code](#) [Issues 5k+](#) [Pull requests 262](#) [Actions](#) [Projects 8](#) [\](#)

[Releases](#) [Tags](#)

 Tags

**v4.5-beta** ...  
🕒 6 days ago -o- 787bb76    

---

**v4.4.3** ...  
🕒 27 days ago -o- bbb31bb    

---

**v4.4.2** ...  
🕒 on 24 Aug -o- a10409c    

---

**v4.4-rc** ...  
🕒 on 12 Aug -o- 55dd850    

---

**v4.4-beta** ...  
🕒 on 30 Jun -o- 3c30b74    

---

**v4.3.5** ...  
🕒 on 1 Jul -o- 1294ee8    

 [clojure / clojurescript](#) Public

[Code](#) [Pull requests 1](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#)

[Releases](#) [Tags](#)

 Tags

**r1.10.879** ...  
🕒 on 19 Jul -o- b68e8b5  

---

**r1.10.866** ...  
🕒 on 21 May -o- 1aa5666   

---

**r1.10.844** ...  
🕒 on 30 Mar -o- a4673b8   

---

**r1.10.773** ...  
🕒 on 26 May 2020 -o- 790bcbf  

---

**r1.10.764** ...  
🕒 on 14 May 2020 -o- ac23fec  

---

**r1.10.758** ...  
🕒 on 4 May 2020 -o- f5a9766   

clojurists together

# NO POPULAR SOURCE OF KNOWLEDGE

## JS

- documentation
- tons of books
- courses
- blog posts
- YouTube channels
- etc.

## CLOJURESCRIPT

- API documentation
- Cheatsheet
- 2-3 books
- blog posts

# JOB OPPORTUNITIES

It's difficult both to find a job and to hire



JAKE-CLARK.TUMBLR

# HOW AND WHY WE USE CLOJURESCRIPT IN PRODUCTION

EDU Conference 2020  
July 14-16, 2020

Your session starts in 10 minutes. Tutorial Enter Practice Room

4 3 Joe Williams 12 points

Town Hall Community Schedule Speakers Sponsors Exhibitors News About

**EDU CONF 2020**  
July 14-16, 2020

**Joe, welcome to EDU Conference 2020**

The EDU Conference 2020 brings together forward-thinking, purpose-driven educators and change-makers with the goal of impacting the future of teaching and learning. Our three-day event offers sessions, in-depth workshops, engaging...

[Read more ↗](#)

**19** Speakers **23** Sessions **12** Sponsors

Platinum Sponsor

Powered by attendify

Community

Everyone Leaderboard

Amy Hawkins Consultant @ Learning Systems Inc.

Ashley Jones Product Manager @ Anchor Systems

Darlene Steward Provost @ Nebraska State University

Jane Andrews Admissions @ ASU

Jorge Robertson Director @ Highland Systems

Regina Cooper Project Director @ Digital Promise

Richard Colman Founder @ Remote EDUcate

Rosemary Edwards Head of Programs @ NY Public Schools

Ryan Peterson Head of Programs @ University of Souther...

Ted Flores Professor @ Seleg School of Business

Theresa Pena Product Manager @ Pedagog Systems

Distance Learning Best Practices

Ryan Peterson Ashley Jones

Up Next

Democratizing Access to Financial Aid

10:45am – 11:15am

Dianne Robertson

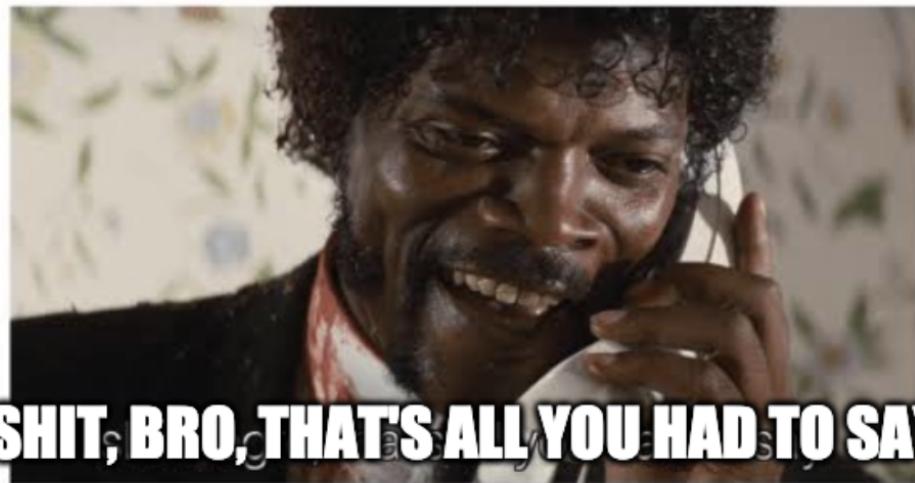
# WE ALL WRITE CLOJURE

When you are the **Frontend** developer,  
but also the **Backend** developer



# CLOJURESCRIPT IS AN AWESOME TOOL TO MAKE THINGS FAST

**6 MONTHS TO RELEASE  
DATE BUT WE GOT CLJS APPROVED**



**SHIT, BRO, THAT'S ALL YOU HAD TO SAY**

# PERFORMANCE

Onsite events - up to 10k attendees

Virtual events - up to 5k attendees

The screenshot displays a virtual event platform interface for the "EDU Conference 2020" held from July 14-16, 2020. The top navigation bar includes a search bar and user profile for "Joe Williams". The main area shows a schedule for Monday, July 14, 2020, with sessions including "Intro: Distance learning in 2020", "Distance Learning Best Practices", "Democratizing Access to Financial Aid", "Workshop: Campus Leadership", "Workshop: Integrating International Students", and "Campus Life 3.0". A highlighted session is "Distance Learning Best Practices" scheduled for 10:00am - 11:00am. The right side features a sidebar for "What's on your mind?" with posts from Regina Cooper and Amy Hawkins, and a "Speakers" section listing Ryan Peterson, Mitch Cooper, and Joe Williams.

# OUR STACK

## Backend

**Ring** - HTTP abstraction

**Compojure** - routing

**Aleph** - async HTTP server

**HugSQL** - SQL query builder

**prismatic/schema** - runtime validation

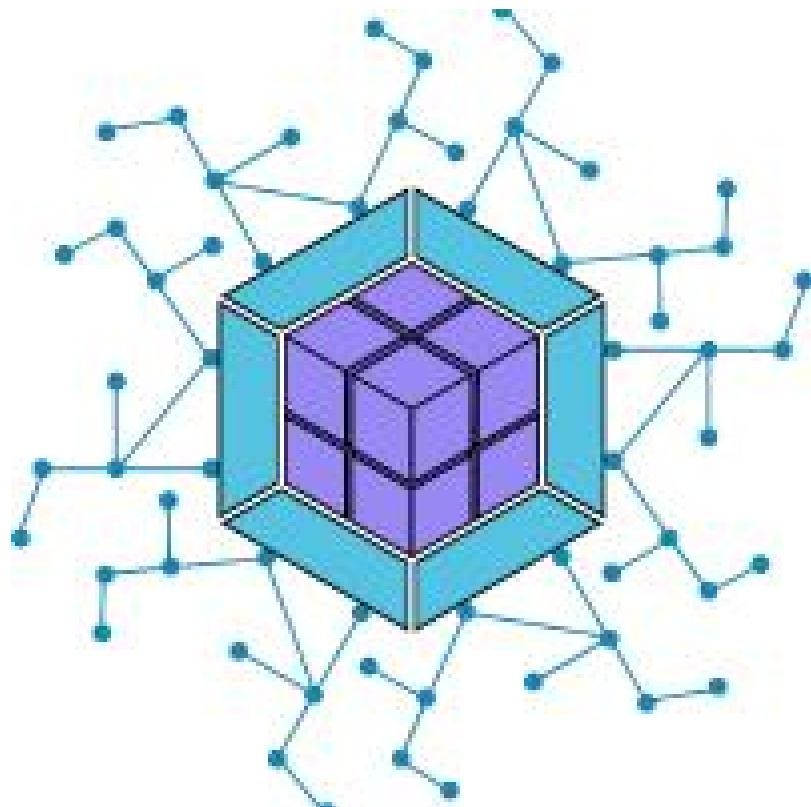
**JSON-RPC-ish** - communication protocol

## Frontend

**Reagent/Rum** - React wrapper

**re-frame** - state management

# HOW DO WE LIVE IN JS WORLD?

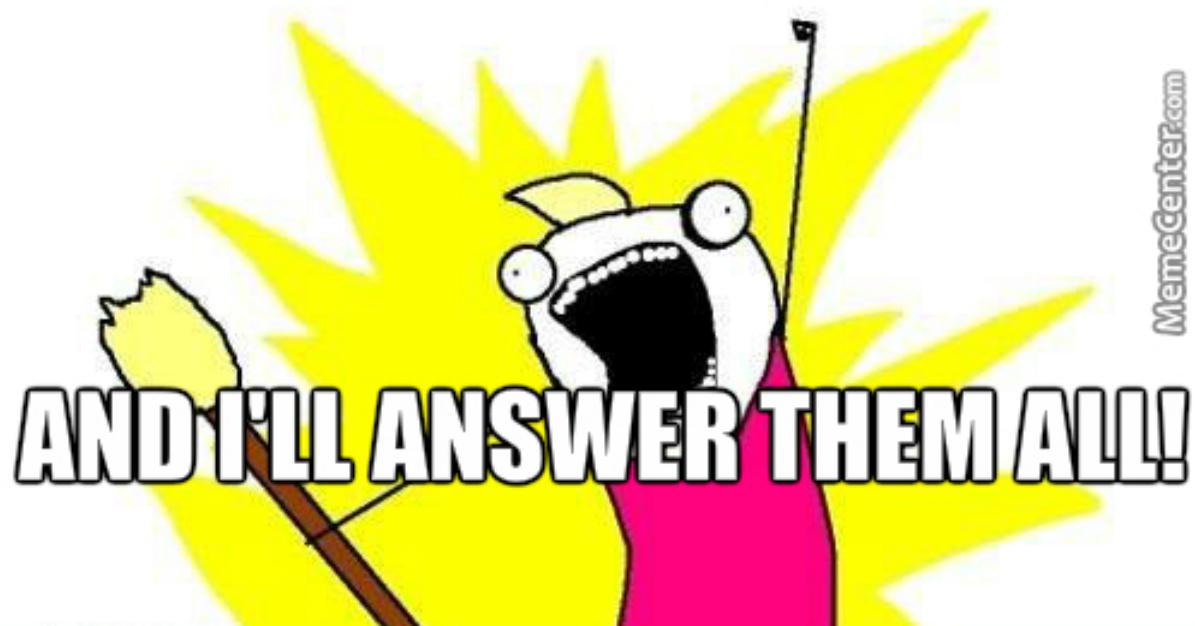


## SOME RESOURCES TO LEARN MORE ABOUT CLOJURESCRIPT

- ClojureScript Unraveled [book](#)
- Clojure for the brave and true [book](#)
- The joy of Clojure [book](#)
- Why I chose ClojureScript over JavaScript [post](#)

Q&A

**ASK ALL THE QUESTIONS**



(MemeCenter.com)

**THANKS FOR ATTENTION!**