2021 VS CLOJURESCRIPT. STILL ALIVE?

Alex Bykbaiev

ABOUT

Alex Bykbaiev, Frontend Engineer at Hopin.

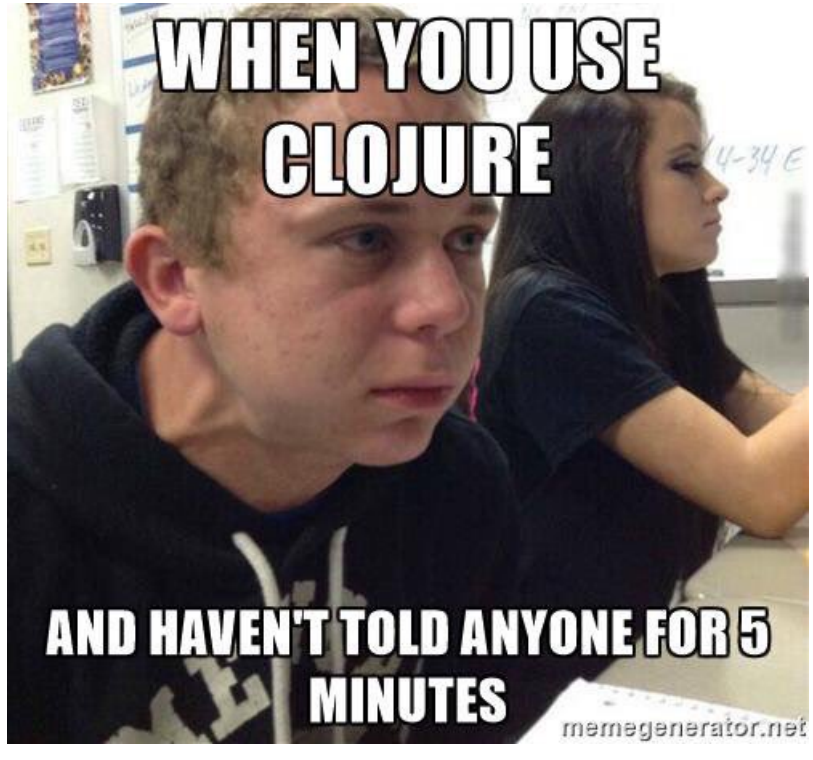
AGENDA

- 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??

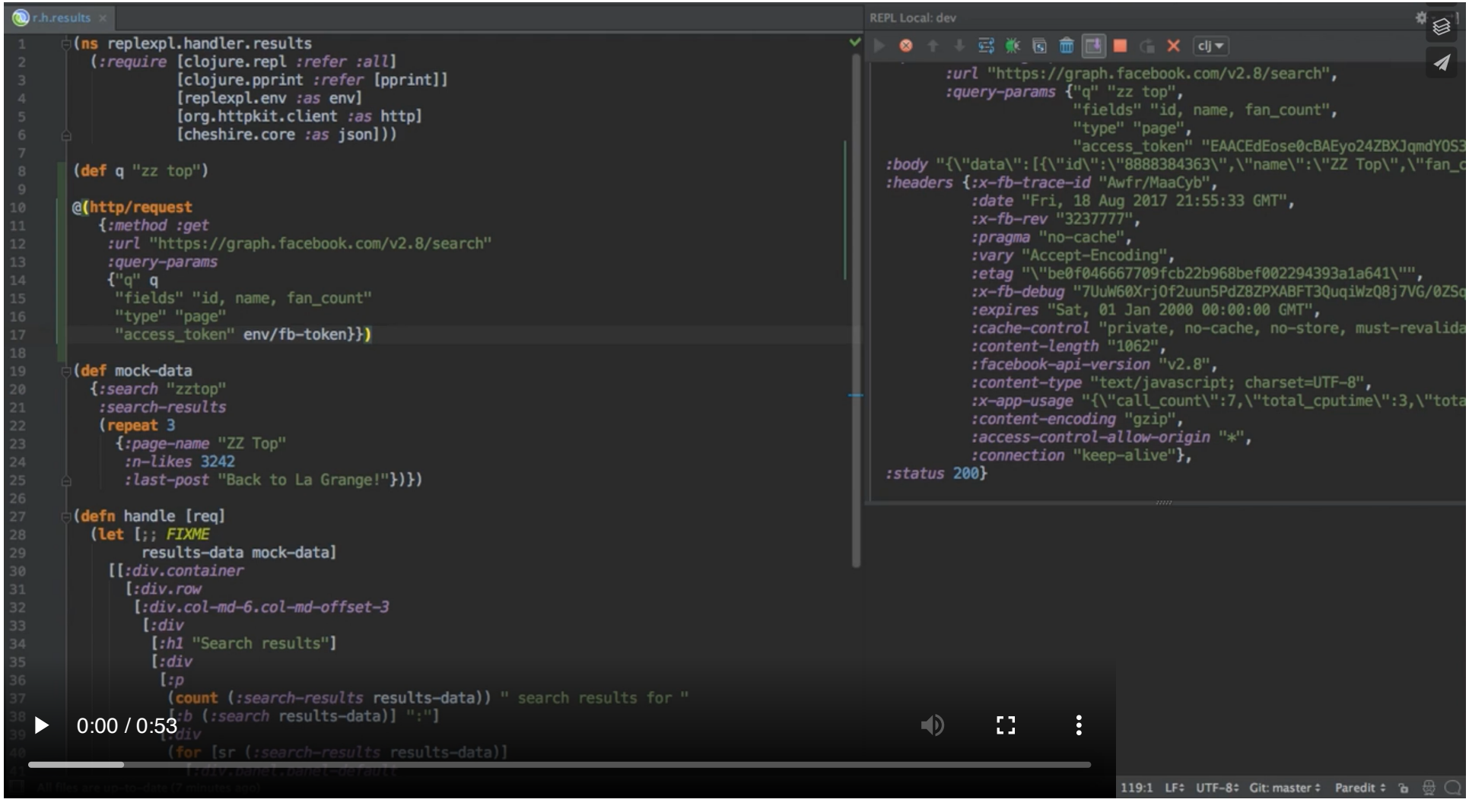
Clojure is a dynamically typed Lisp for JVM.
ClojureScript is same for JavaScript.

STUFF I LIKE IN CLOJURESCRIPT



MANAGEMENT OF CONTROL FLOW

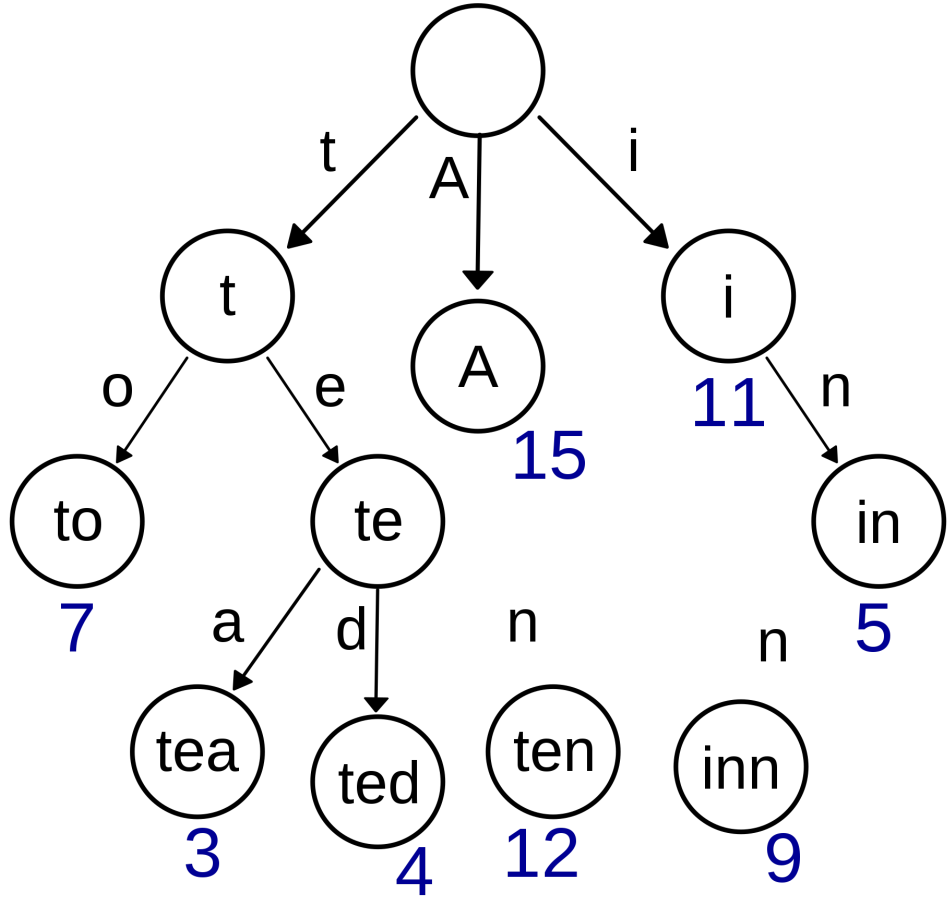
REPL (READ–EVAL–PRINT LOOP)



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.

```clojure
(require '[clojure.set :as s])

(def set' #{1 2 3 10})

(set' 0)
=> nil

(set' 10)
=> 10

(filter #{"rose" "marigold" "hibiscus"}
        ["orchid" "sunflower" "hibiscus" "daisy"
         "gerbera" "lavender" "marigold"])
=> ("hibiscus" "marigold")

```

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

MACROS FOR STYLED COMPONENTS

JS

```
const SubItemNumber = styled.span`
    flex: 0 0 ${polished.rem(54)};
    margin-left: auto;
    text-align: end;
    overflow: hidden;
    text-overflow: ellipsis;
`
...
<SubItemNumber>
    123
</SubItemNumber>
```

MACROS FOR STYLED COMPONENTS

CLJS

```
(defstyled sub-item-number
           :span
           {:flex (str "0 0 " (polished/rem 54))
            :margin-left "auto"
            :text-align "end"
            :overflow "hidden"
            :text-overflow "ellipsis"})
...
(sub-item-number "123")
...
(js/React.createElement
    (-> component meta :react-component)
    #js {}
    "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  (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}
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

```clojure
(def numbers [1 2 3 4 5 6 7 8 9 0])

(take 2 (filter odd? (map inc numbers)))
=> (3 5)

(->> numbers
     (map inc)
     (filter odd?)
     (take 2))
=> (3 5)
```

THREADING MACROS

Thread-as macro

```clojure
(def points {:vals [1 2 3 4 5 6 7 8 9 0]})

(take 2 (filter odd? (map inc (get points :vals))))
=> (3 5)

(as-> points $
      (get $ :vals)
      (map inc $)
      (filter odd? $)
      (take 2 $))
=> (3 5)
```

THREADING MACROS

Thread-some macros

```clojure
(def user {:name "Alex"})

(-> user
    (get :mental-age) ;; => nil
    (str " years"))
=> " years"

(some-> user
        (get :mental-age)
        (str " years"))
=> nil
```

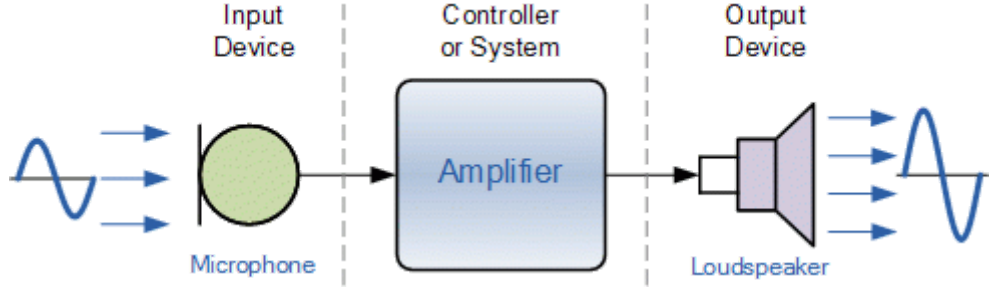
THREADING MACROS

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.



```clojure
(def xs [1 2 3 4 5 6 7 8 9 0])

(->> xs
     (map inc)
     (filter odd?)
     (take 2))
=> (3 5)

;; map = transform => reducer => reducer

(def tr (comp (map inc)
              (filter odd?)
              (take 2)))

(reduce (tr conj) [] xs)
```

```clojure

;; map = transform => reducer => reducer

(def tr (comp (map inc)
              (filter odd?)
              (take 2)))

(reduce (tr conj) [] xs)
=> [3 5]

(into [] tr xs)
=> [3 5]


(transduce tr conj [] xs)
```

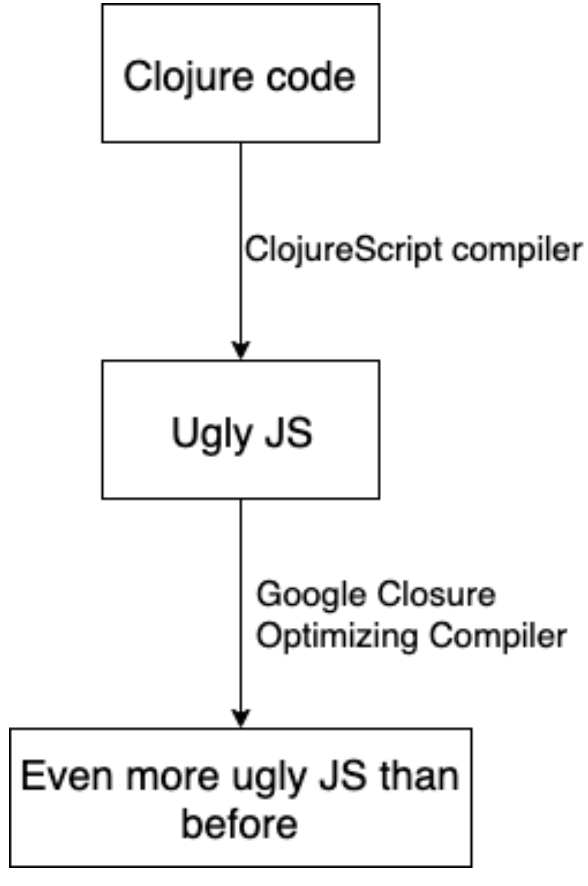
TRANSDUCERS

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



```clojure
(def xs [1 2 3 4 5 6 7 8 9 0])

(->> xs
     (map inc)
     (filter odd?)
     (take 2))
```

```javascript
cljs.user.xs = new cljs.core.PersistentVector(
    null, 10, 5, cljs.core.PersistentVector.EMPTY_NODE, [(1
);
cljs.core.take.call(
    null,(2),cljs.core.filter.call(
        null,cljs.core.odd_QMARK_,cljs.core.map.call(
            null,cljs.core.inc,cljs.user.xs
        )
    )
);
```

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 %
+	**src**	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

```clojure
(require 'goog.object)

(def at-runtime (clj->js {:a {:b {:c 1}}}))
=> #js {:a #js {:b #js {:c 1}}}

(def at-compile-time #js {:a #js {:b #js {:c 1}}})
=> #js {:a #js {:b #js {:c 1}}}

(-> heavy-js-object
    (js->clj :keywordize-keys true)
    (get-in [:a :b :c]))

(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

```clojure
(def input [{"ab" [1 0]} {"bc" [1 1]} {"de" [1 0]}
           {"ab" [2 1]} {"ab" [0 1]}])

;; Need to get {"ab" [3 2] "bc" [1 1] "de" [1 0]}

(defn merge-hashes-naive [coll]
    (apply merge-with (fn [a b] (map + a b)) coll))

(merge-hashes-naive input)
=> {"ab" (map + [0 1] (map + [2 1] [1 0])) ...}

(merge-hashes-naive large-input)
=> java.lang.StackOverflowError
```

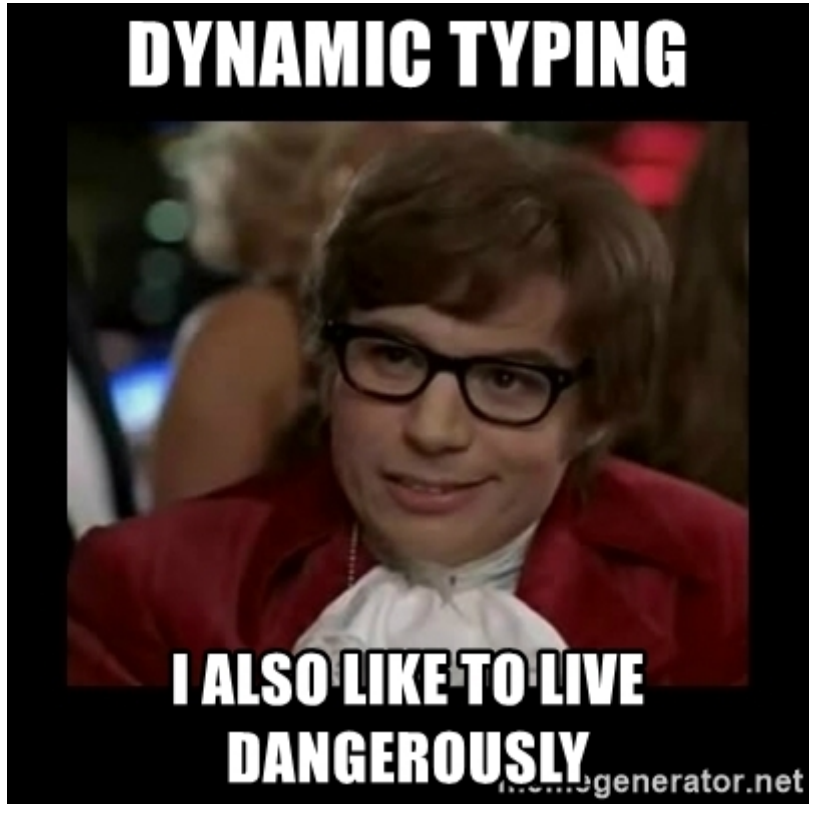
Clojure lazy evaluation and stack overflow exceptions

AUTHORITARITY 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

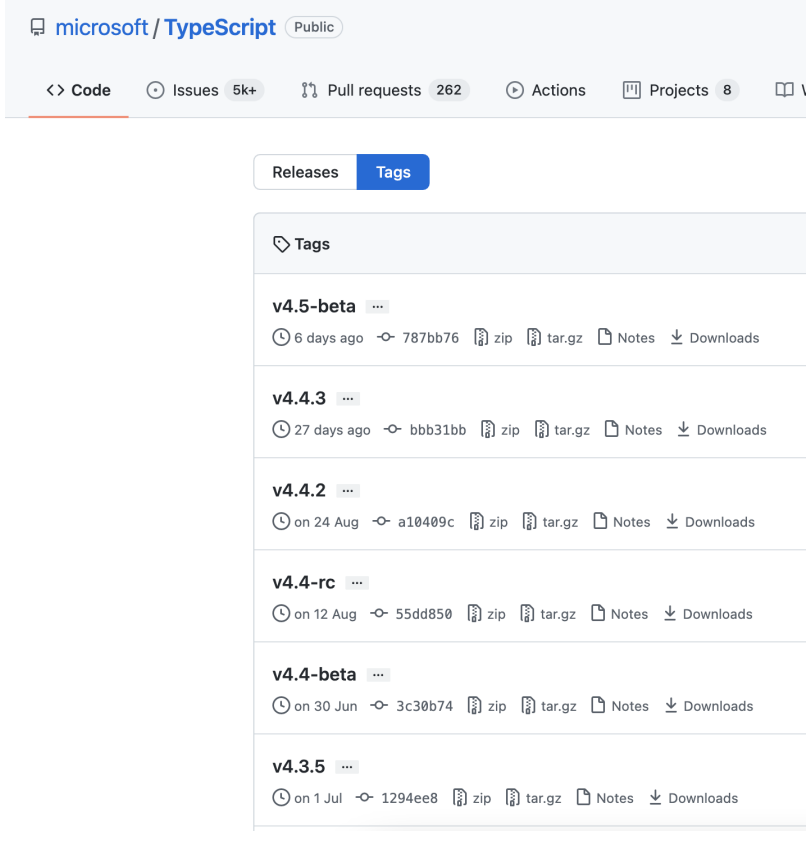


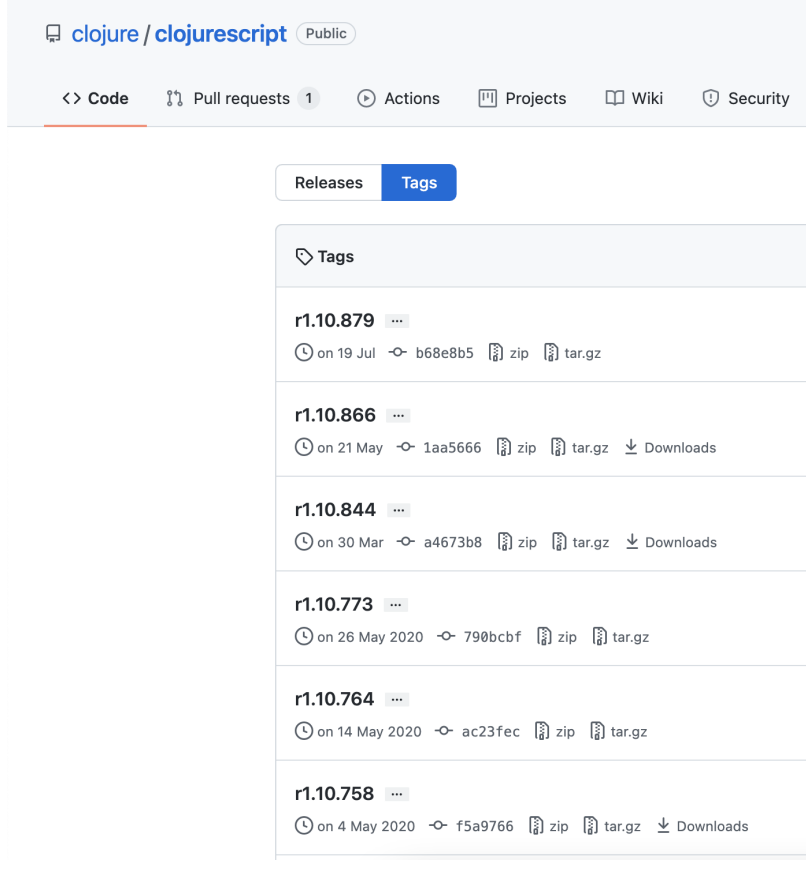
LISP CURSE

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

The Lisp Curse

SUPPORT





clojurists together

NO POPULAR SOURCE OF KNOWLEDGE

JS

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

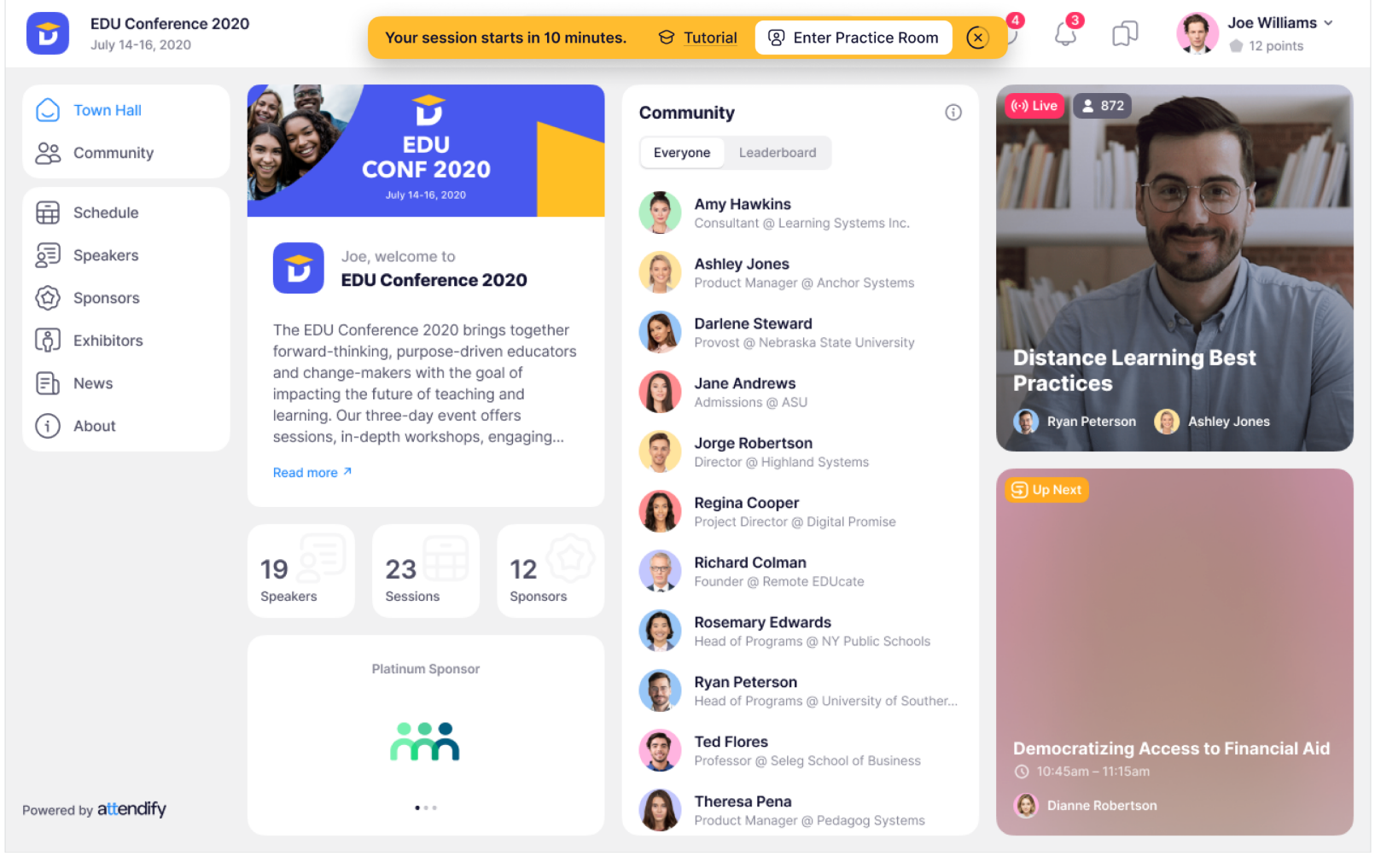
CLOJURESCRIPT

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

JOB OPPORTUNITIES

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

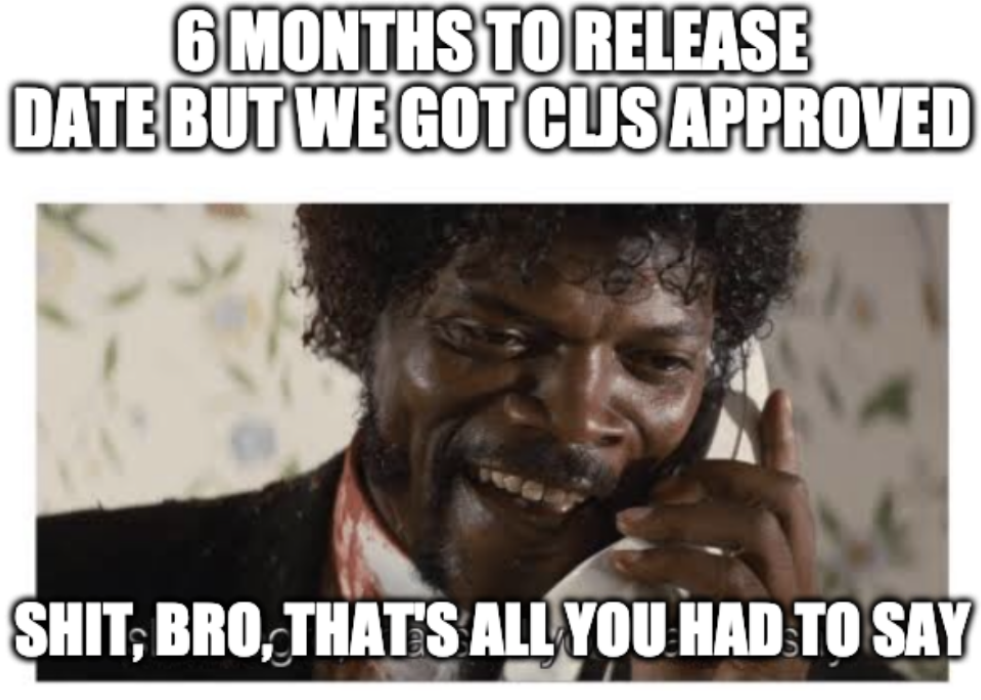
HOW AND WHY WE USE CLOJURESCRIPT IN PRODUCTION



WE ALL WRITE CLOJURE

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

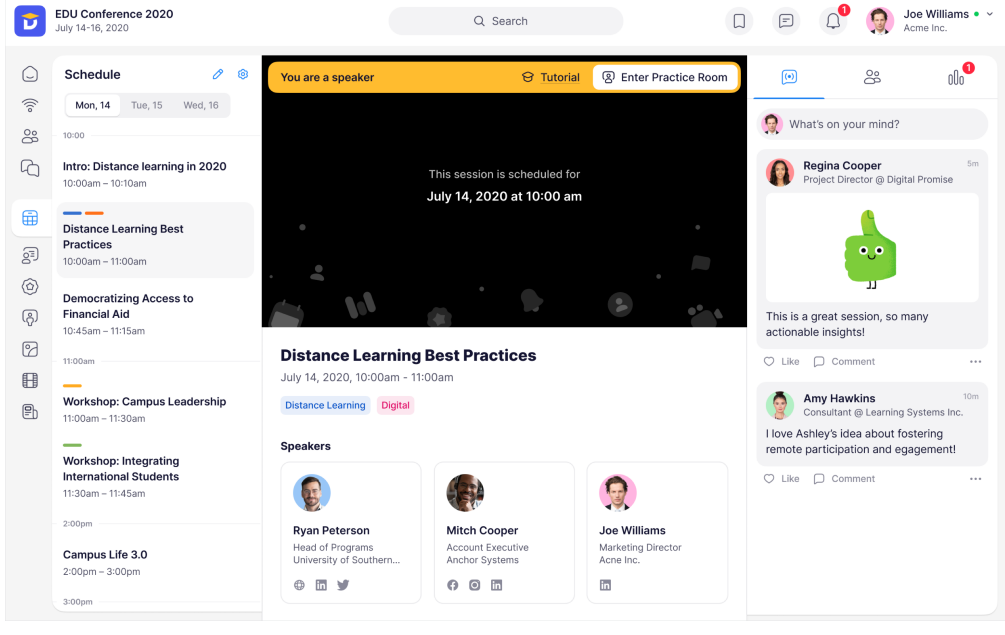
CLOJURESCRIPT IS AN AWESOME TOOL TO MAKE THINGS FAST



PERFORMANCE

Onsite events - up to 10k attendees

Virtual events - up to 5k attendees



OUR STACK

Backend

Ring - HTTP abstraction

Compojure - routing

Aleph - async HTTP server

HugSQL - SQL query builder

prismatic/schema - runtime validation

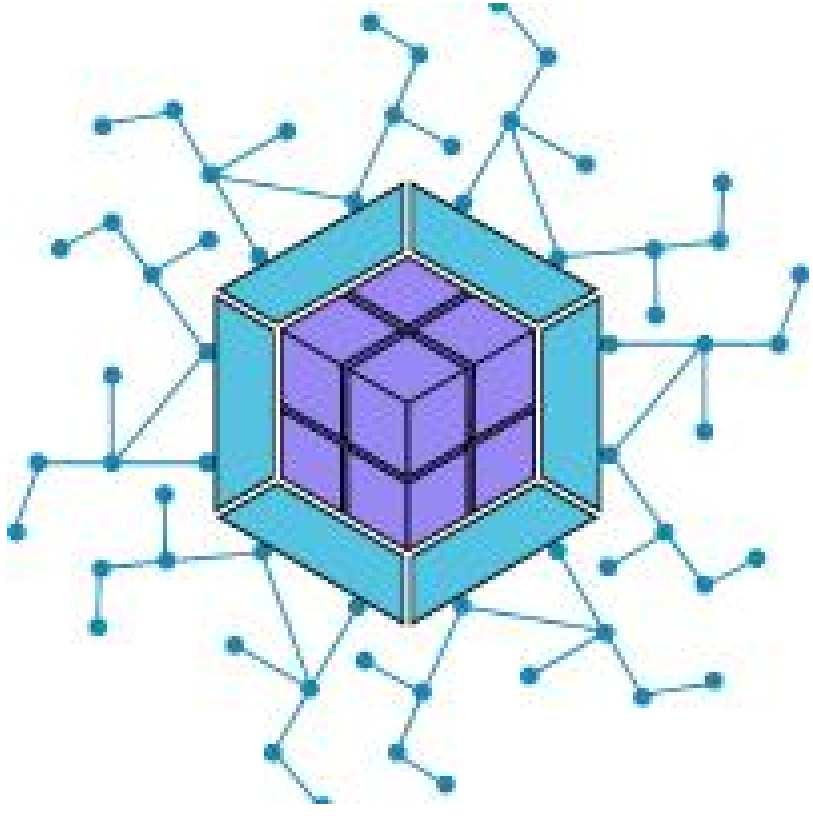
JSON-RPC-ish - communication protocol

Frontend

Reagent/Rum - React wrapper

re-frame - state management

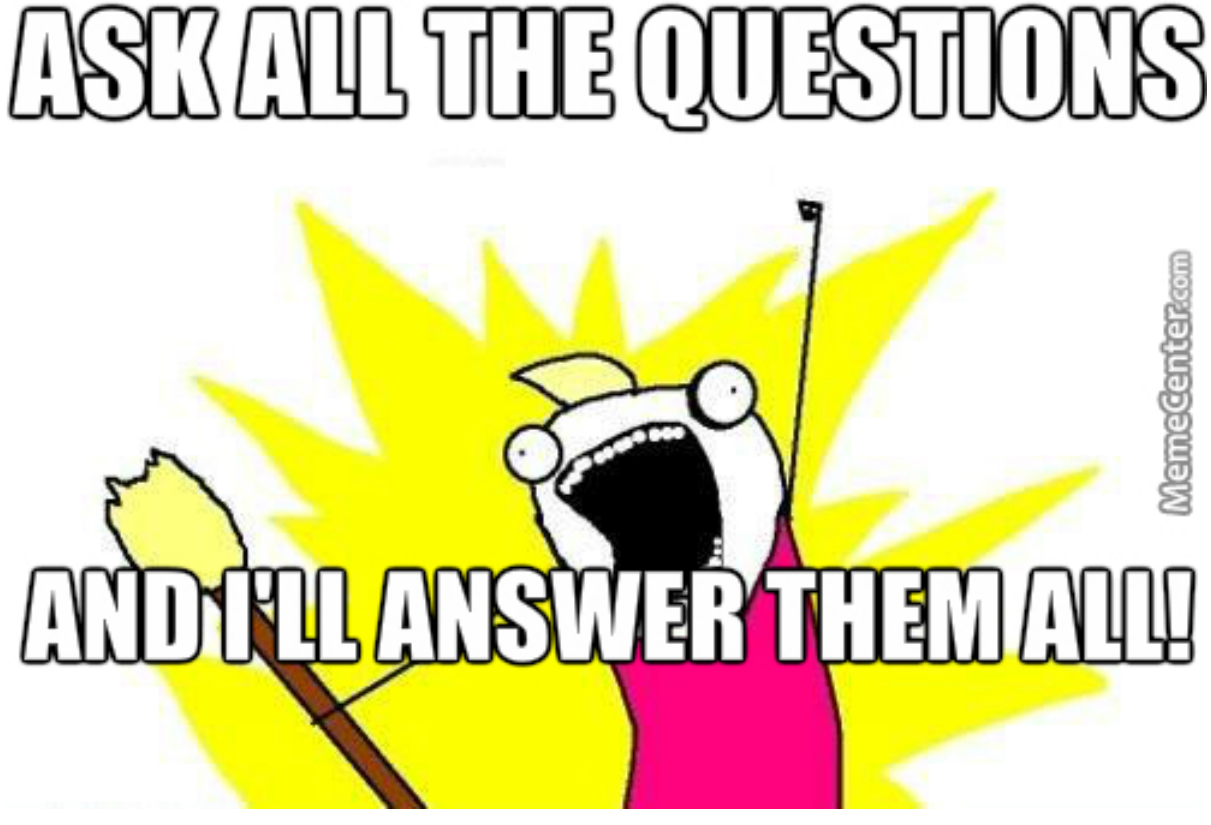
HOW DO WE LIVE IN JS WORLD?



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
AND I'LL ANSWER THEM ALL!
MemeCenter.com

THANKS FOR ATTENTION!