

If we are going to transpile JavaScript, why not use ClojureScript?

Shivek Khurana

JUXT



About me

- Apps since 2008, Clojure since 2017
- Consultant at Juxt Ltd.
- Enterprise Apps
- Currently building a system for Vodafone

JUXT



Built an LMS from 2012 to 2016

Long projects == Living with your past mistakes

JUXT



Followed best practices



JUST



JS ecosystem was maturing in 2012

Is it mature now?



PARCEL



SVELTE



rollup.js

JUXT



JS Community Conundrum

LI + EC = HVOD + C

JUX^T



JS Community Conundrum

$$LI + EC = HVOD + C$$

Language
Imperfections

Enthusiastic
Community

High Velocity
Open Source
Development

Abandoned
Libs & Ideas

JUXT

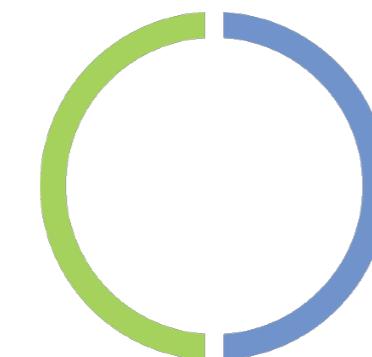


Stability

- Add new features after periods of inactivity

JUXT





Clojure/Script

LISP Functional Immutable Homoiconic

Thread Safe

Dynamically typed

Hosted

Joy to use

JUXT



Clojure/Script Crash Course

*All the best games are easy
to learn and difficult to
master.*

— Nolan Bushnell (Founder, Atari)

JUST



;; Native Data Types

;; Character

\a
\b
\newline
\tab

;; String

“hello”
“Line 1\nline2”

;; Keywords

:keyword

;; Regex

#"[a-z]+"

;; Bool

true, false

;; Nothing

nil

;; Number

255 ;; int
012 ;; octal
0xff ;; hex
2r1111 ;; radix
3.14 ;; standard float
1.35e-12 ;; scientific float
22/7 ;; fraction

JUST



;; List

;; function calls are written as a list
`(max 1 2) ;; => 2`

;; to not call the first element
`(list 1 3 5) ;; => (1 3 5)`
`'(1 3 5) ;; => (1 3 5)`
`(quote (1 3 5)) ;; => (1 3 5)`



;; Vector, Sets & Maps

[1 2]

[1, 2]

(vector 1 2)

#{1 3}

(set 1 2)

{:name "Shivek"
:age 24
[:a :b] :c
{:alpha "b"} :kappa}

JUX^T



;; LISP – LISt Processing

```
'(1 2 3)
```

```
(+ 1 2 3)
```

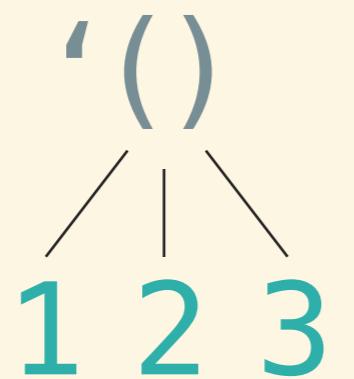
```
(defn sum [a b]  
  (+ a b))
```

JUX^T



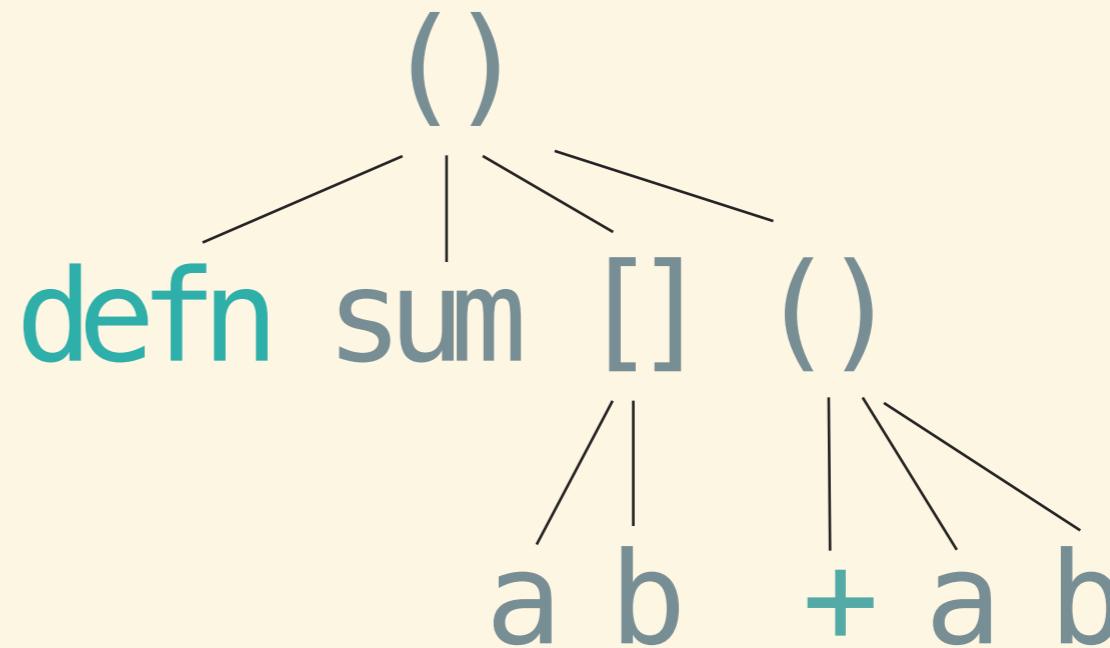
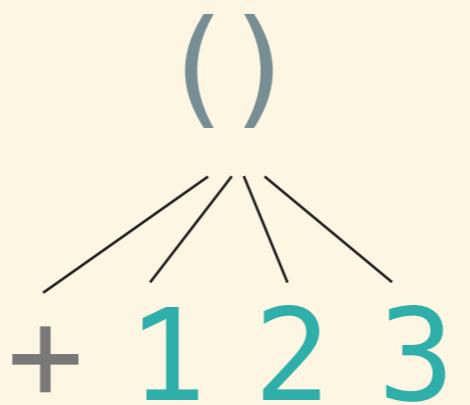
;; LISP

' (1 2 3)



(+ 1 2 3)

(defn sum [a b]
(+ a b))



;; Functional

```
(defonce person
  {:_name "Vienaa"
   :age 24})
```

```
(get person :name)
(get person :age)
```

```
class Person {
  init(name, age) {...}
  getName() => {...}
  getAge() => {...}
}

const v = Person("Viena", 24)
v.getName()
v.getAge()
```

JUXT



;; Immutable

```
(defonce count 0)
(inc count) ;; => 1
count ;; => 0
```

JUXT



;; Atoms
;; thread safe/ mutable constructs

```
(defonce count (atom 0))  
(swap! count inc) ;; => 1  
count ;; => 1  
(reset! count 4) ;; => 4  
count ;; => 4
```

JUXT



;; Homoiconic
;; Code is data

' (1 2 4)

[2 1]

(defn sum [a b] (+ a b))

JUXT



Compiles to JavaScript using Google Closure Compiler



JUXT



Google Closure Compiler

- In prod for 10+ years
- Compiles JS to better JS

JUXT



Google Closure Library

- Google's Standard Lib for JS
- Cross Platform

JUXT



;; Interop with JS

```
(.getElementById js/document "root")
```

```
document.getElementById("root")
```

```
(new js/FormData)
```

```
new FormData()
```

```
(.-height js/window)
```

```
window.height
```

```
(clj->js {:hello :world})
```

```
#js {:hello "world"}
```

JUXT



;; Consuming NPM Packages

```
( :require ["react-dom/server" :refer [renderToString]] )  
( renderToString react-element)
```

JUXT



;; Write Node Libs

```
(ns demo.lib)
(defn hello []
  (prn "hello")
  "hello")

{...
 :builds {:library {:target      :node-library
                     :output-to "out/demo-library/lib.js"
                     :exports    {::hello demo.lib/hello}}}}
```

JUXT



;; Write Node Libs

```
$ cd out/demo-library
$ node
> var x = require('./lib');
undefined
> x.hello()
hello
'hello'
```

JUXT



;; Write NPM Packages

```
(defn ^:export my-function [] ...)
```

JUXT



;; HTML/React Components ;; using Hiccup

```
(defn Settings [username realname email]
  [:div {:class "pa2 ma3"}
    [:h3 username]
    [:div realname]
    [:div email]
    [:button {:onClick #()} "Edit"]])
```

JUXT



;; JS Equivalent

```
function Settings(username, realname, email) {  
  return (<div class="pa2 ma3">  
    <h3>{username}</h3>  
    <div>{realname}</div>  
    <div>{email}</div>  
    <button onClick={() => {...}}>Edit</button>  
  </div>);  
}
```

JUXT



REPL

Developing inside the Runtime

JUXT



Observation Driven Dev

Text Editor

Code



Transpile +
Hot Swap

Runtime (Browser)

Updated App

Perform Actions
Observe

Code

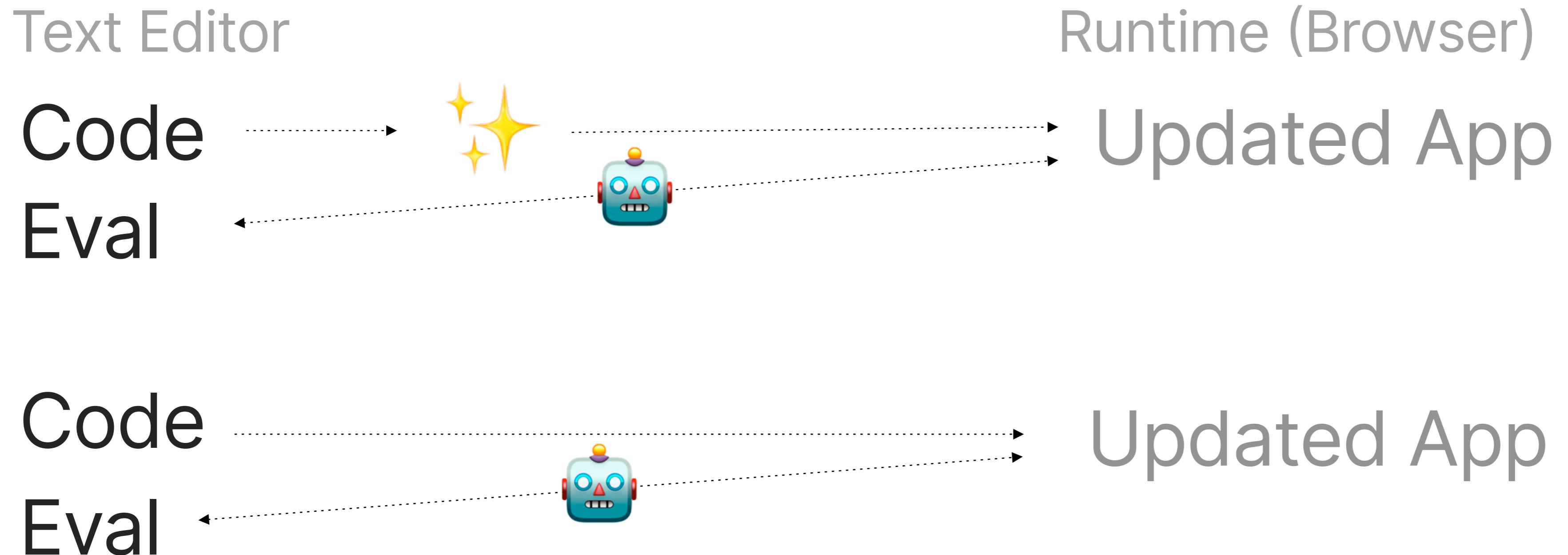
Updated App

Perform Actions
Observe

JUXT



REPL Driven Dev



JUXT



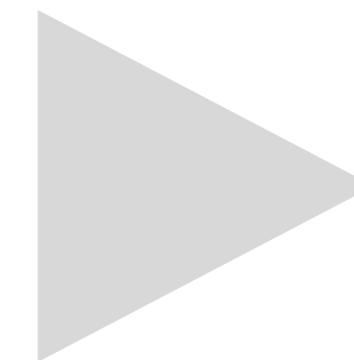
REPL Demo - Tic Tac Toe

State

Lattice

```
[[{:x nil nil}  
[nil :x nil]  
[nil nil :o]]
```

Turn :o



React

:o's turn

x	-	-
-	x	-
-	-	o

Reset

JUXT



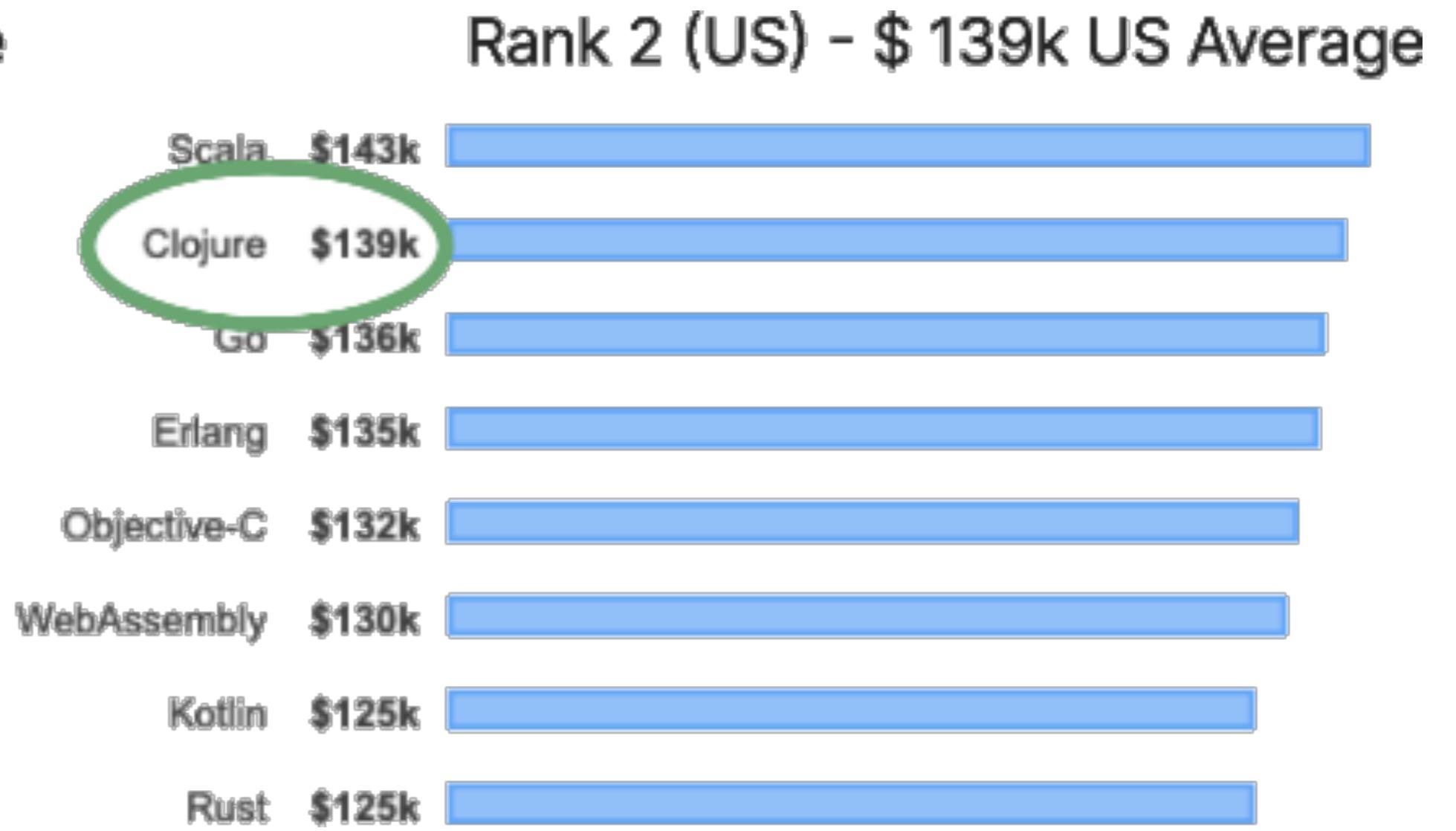
With REPL - Develop only a function

- Precisely load the relevant function
- Test with inline feedback
- Rich Comments in git history

JUXT



Clojure Devs are best paid



clojure.spec

- Optional type library
- Define types as predicates
- Assert/ Validate/ Generate/ Conform

JUXT



;; valid?

```
(:require [clojure.spec.alpha :as s])
```

```
(s/valid? even? 100) ;; => true
```

```
(s/valid? odd? 2) ;; => false
```

```
(s/valid #(< 5 %) 10) ;; => true
```

```
(s/valid? #{:club :diamond :heart :spade} :club) ;; => true
```

JUXT



;; Registry

```
;; Global registry using ns keyword
(ns spec.defs)
(s/def ::suit #{:club :diamond :heart :spade})
```

```
;; Check
(ns my.app)
(s/valid? :spec.defs/suit :club)
```

JUXT



;; Composing predicates

```
(s/def ::name-or-id (s/or :name string?  
                           :id    int?))  
(s/valid? ::name-or-id "abc") ;; => true  
(s/valid? ::name-or-id 100) ;; => true  
(s/valid? ::name-or-id :foo) ;; => false
```

JUXT



;; explain

```
(s/explain ::name-or-id :foo)
;; :foo - failed: string? at: [:name] spec: :user/name-or-id
;; :foo - failed: int? at: [:id] spec: :user/name-or-id
```

JUXT



;; Spec domain entities

```
(def email-regex #"[^a-zA-Z0-9._%+]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,63}$")
(s/def ::email-type (s/and string? #(re-matches email-regex %)))

(s/def ::id int?)
(s/def ::first-name string?)
(s/def ::last-name string?)
(s/def ::email ::email-type)

(s/def ::person (s/keys :req [::id ::first-name ::email]
                         :opt [::last-name]))
```

JUXT



Spec Demo

Form (React)

State

first-name
last-name
email

Validation

(*s/valid?* ::person %)

API (Stub)

POST /person

JUST



Spec Benefits

- Optional opt in
- Generative testing
- Spec code (not just data)*

<https://clojure.org/guides/spec>



Switching stacks

Weird Syntax
No classes
Small Community
Talent/ Training

JUX^T



Switching stacks

Weird Syntax
No classes
Small community
Talent/ Training

Brackets become invisible
Functional ❤️
Brilliant community
Can be improved

JUXT



Build Systems



Lein + Figwheel

<https://leinigen.org/>

<https://figwheel.org/>



Shadow CLJS

<https://shadow-cljs.org>

JUXT



React Wrappers



Reagent

<https://reagent-project.github.io/>

*Reagent is what
React hopes to be
one day*

— Luke Whorton
(<https://medium.com/@lwhorton/>)

JUXT



React Wrappers



Reagent

<https://reagent-project.github.io/>

hx

hx

<https://github.com/Lokeh/hx>

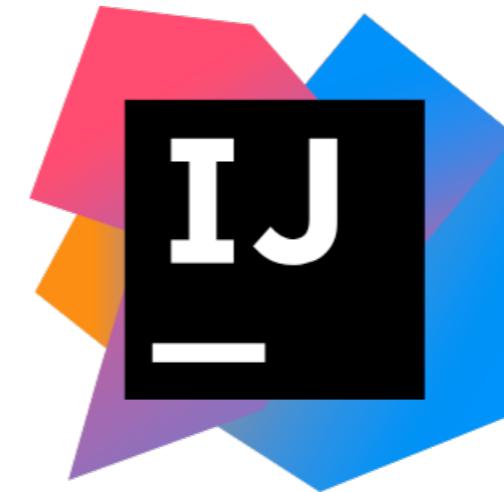
JUXT



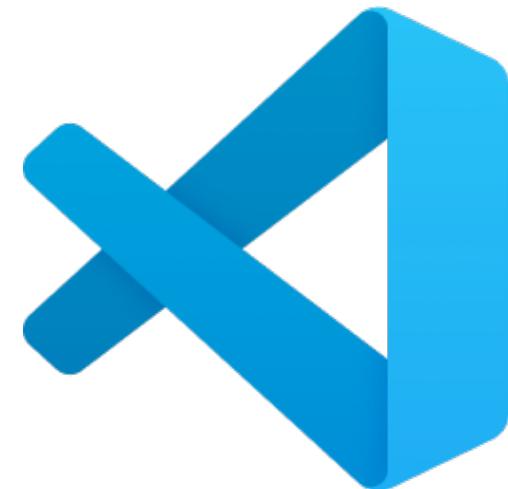
Editors



(cider[])



Cursive



(()) *Calva*



Fireplace (?)

JUXT



People



Rich Hickey
Creator of Clojure



Stuart Halloway
Core Team Member



Alex Miller
Core Team Member



Stuart Sierra
Component; Reloaded



Malcolm Sparks
Yada/Bidi/Tick



James Reeves
Integrant/ Hiccup/ Medley



Bruce Hauman
Figwheel



Bozhidar Batsov
nrepl

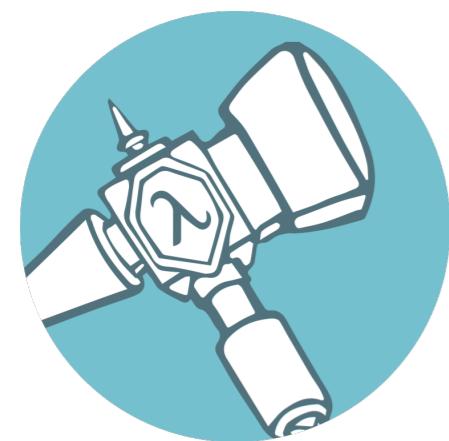


David Nolen
ClojureScript



David Miller
ClojureCLR

Blogs & Docs



bravecjure.com

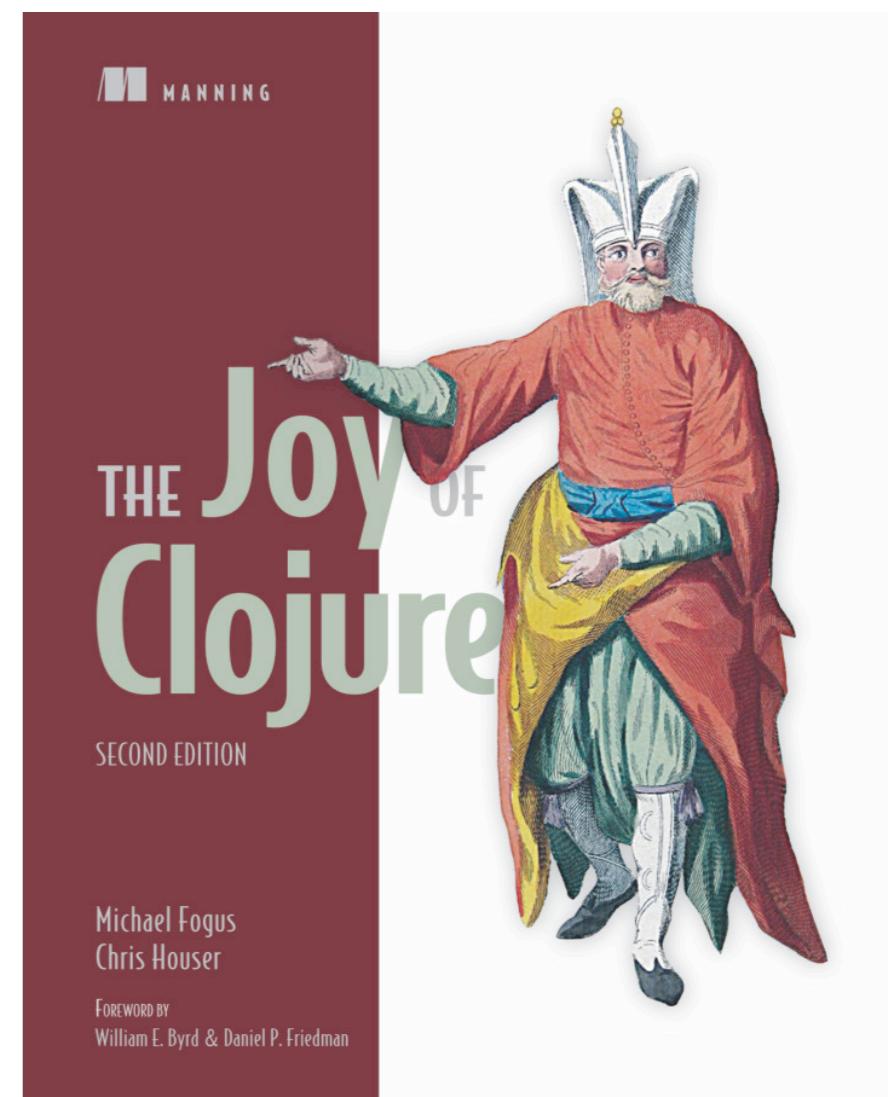
JUXT
juxt.pro/blog

PurelyFunctional.tv
purelyfunctional.tv



CLOJURE KOANS

clojurekoans.com
clojurescriptkoans.com



The Joy of Clojure;
By Fogus & Houser

JUXT





Thank You

Shivek Khurana

Clojure Consultant at JUXT Inc

Independant Consultant at Krim Labs

Around the web

medium.com/@shivekkhurana

github.com/shivekkhurana

twitter.com/shivek_khurana

JUXT

