

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

medium.com/@shivekkhurana
github.com/shivekkhurana
twitter.com/shivek_khurana



Built an LMS from 2012 to 2016

Long projects == Living with
your past mistakes



Followed all best practices

Changed the build system thrice



JS ecosystem was maturing in 2012

Is it mature now?



PARCEL



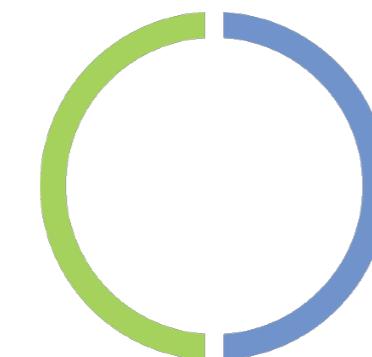
rollup.js



The JS community is blessed

But raw JS is not





Clojure/Script

LISP Functional Immutable Homoiconic

Thread Safe

Dynamically typed

Hosted

Joy to use



Clojure/Script Crash Course

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

— *Nolan Bushnell (Founder, Atari)*



;; 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



;; List

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

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



;; Vector, Sets & Maps

[1 2]

(vector 1 2)

#{1 3}

(set 1 2)

{:name "Shivek"
:age 24}



;; LISP

(1 2 3)

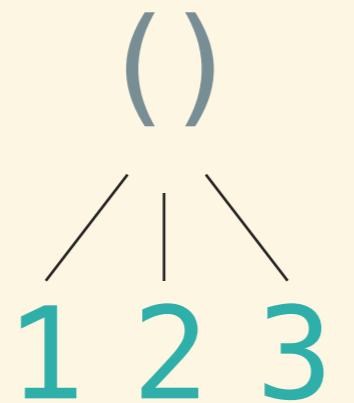
(+ 1 2 3)

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

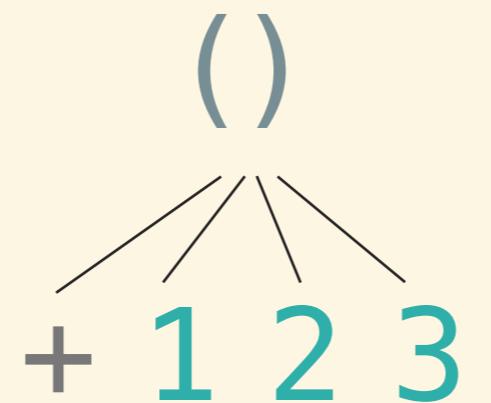


;; LISP

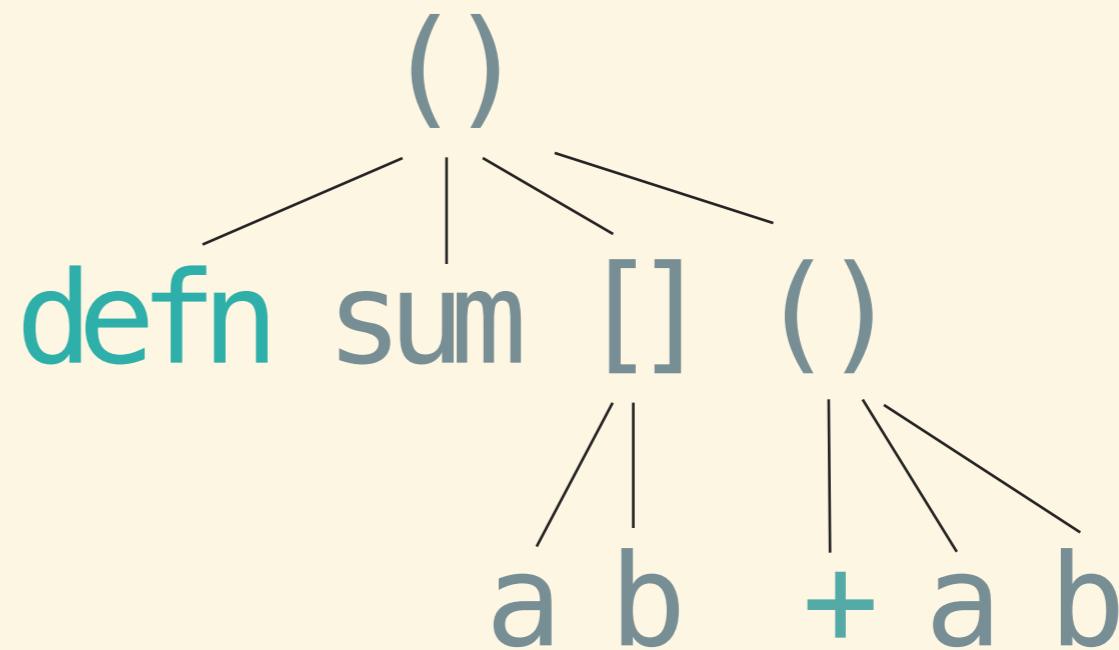
(1 2 3)



(+ 1 2 3)



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



;; Functional

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

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

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

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



;; Immutable

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



;; Atoms
;; thread safe/ mutable constructs

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



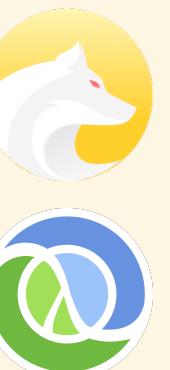
;; Homoiconic
;; Code is data

(1 2 4)

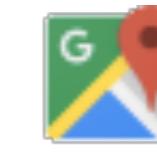
{:name "Shivek" :age 24}

[2 1]

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



Compiles to JavaScript using Google Closure Compiler



Google Closure Compiler

- In prod for 10+ years
- Compiles JS to better JS
- Parse > Analyse > Remove Dead Code
- Minify & Rewrite what's left



Google Closure Library

- Google's Standard Lib for JS
- Cross Platform



;; Interop with JS

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

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

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

```
new FormData()
```

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

```
window.height
```

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

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



;; Consuming NPM Packages

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



;; Write NPM Packages

```
(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}}}}
```



;; Write NPM Packages

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



;; Write NPM Packages

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



;; HTML/React Components ;; using Hiccup

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



;; 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>);  
}
```



Switching stacks

Weird Syntax
Immutable
No classes
Small Community
Talent/ Training



Switching stacks

Weird Syntax
Immutable
No classes
Small community
Talent/ Training

Brackets become invisible
Immutable ❤️
Functional ❤️
Small but brilliant community
Things are changing



Differences from JS

Built-in objects

Classes

Functions

Prototypes

Mutability

Statements (Syntax)



Differences from JS

Built-in Objects
Classes

Functions
Prototypes

Mutability
Statements (Syntax)

Built in Data Structures



Functions



Immutability



REPL

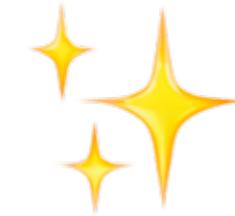
Developing inside the Runtime



Observation Driven Dev

Text Editor

Code



Transpile +
Hot Swap

Runtime (Browser)

Updated App

Perform Actions
Observe

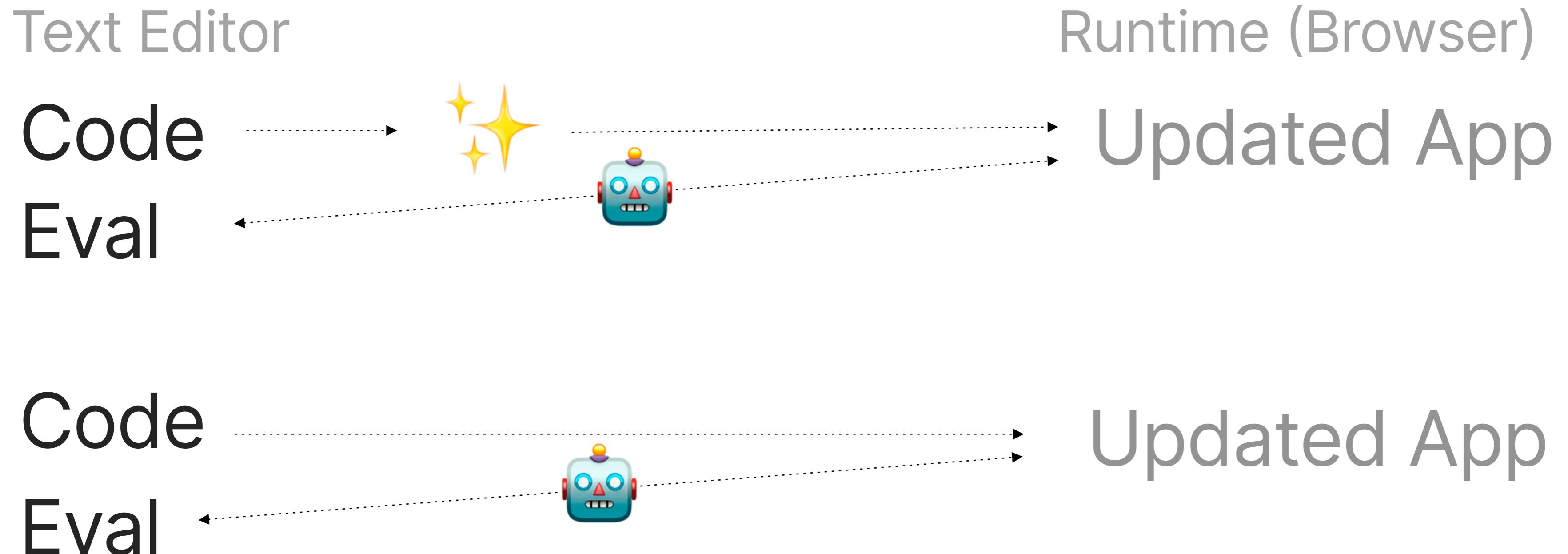
Code

Updated App

Perform Actions
Observe



REPL Driven Dev



REPL Demo - Tic Tac Toe

State

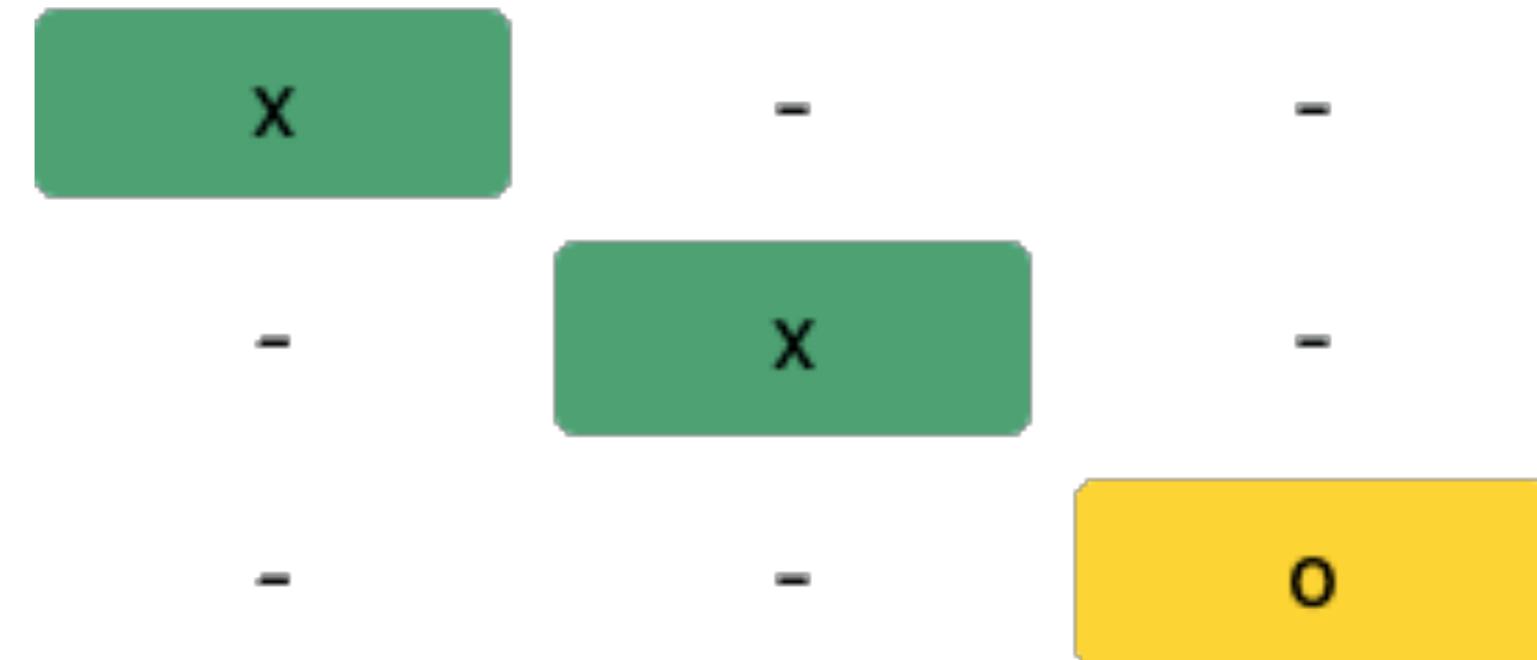
Lattice

```
[[:x nil nil]  
[nil :x nil]  
[nil nil :o]]
```

Turn :x

React

:o's turn



Reset

