2017 in ClojureScript

Web Performance & JavaScript Ecosystem

Clojure/Conj 2017

@anmonteiro90

$ whoami

Ladder

LUMO

Why?

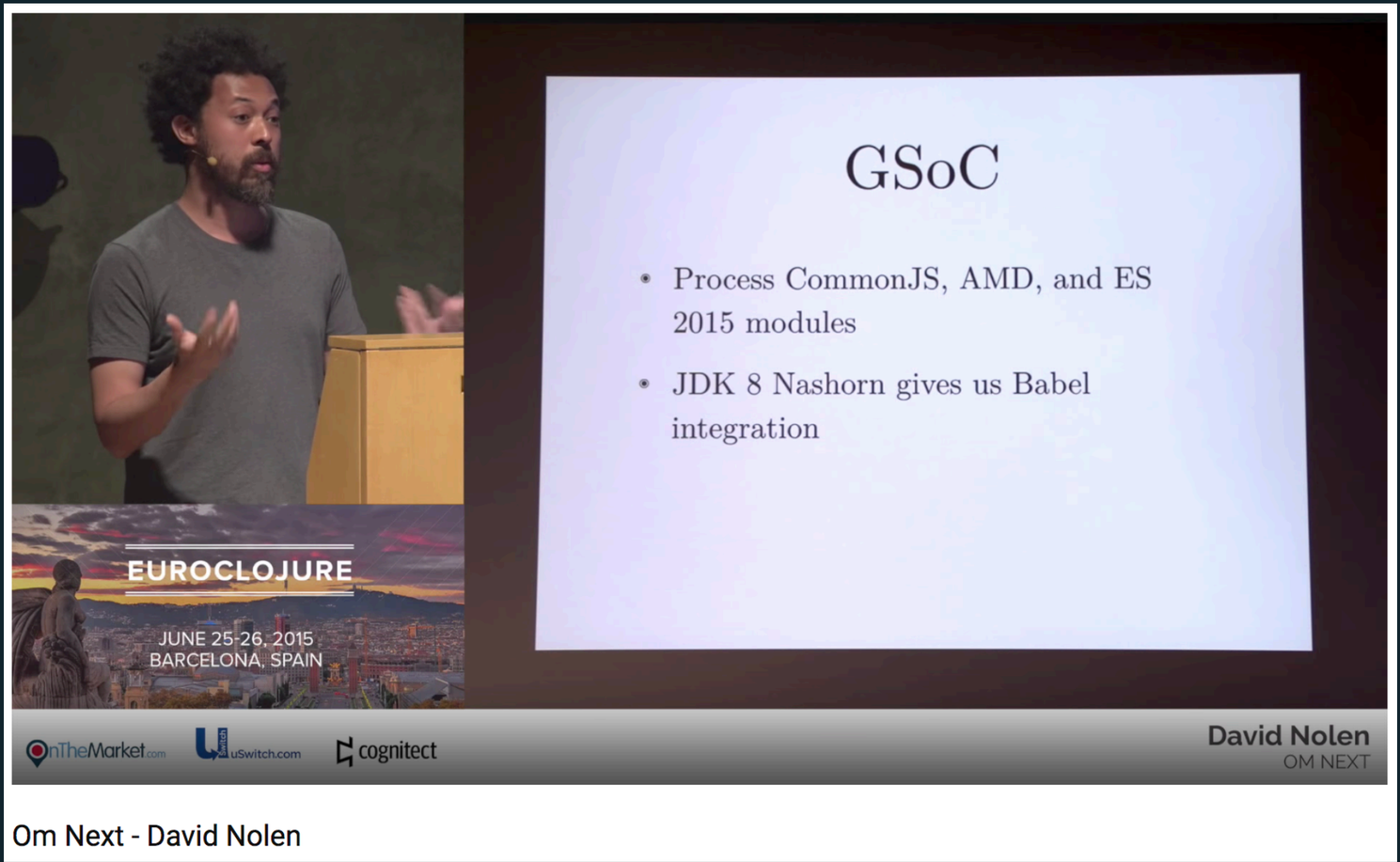
Rich Hickey - ClojureScript release

Before we start

- As of ClojureScript 1.9.946
- Google Closure or Closure Compiler, not to be confused with Clojure
  - github.com/google/closure-compiler

2015

2015: Initial JavaScript Module Support



Om Next - David Nolen

```clojure
(ns my-project.core
  (:require [clojure.string :as string]))

(defn a-function []
  ...)
```

becomes

```javascript
goog.provide('my_project.core');
goog.require('cljs.core');
goog.require('clojure.string');

my_project.core.a_function = function() {
  ...
};
```

But JavaScript modules may be CommonJS:

```
// my-module.js
var other = require('./other-module');

function aFunction() {
  ...
}

module.exports = {
  aFunction: aFunction,
};
```

But JavaScript modules may be ES6 / ES2017:

```
// my-module.js
import other from './other-module';

export default function aFunction() {
  ...
}
```

But JavaScript modules may be AMD:

```
// my-module.js
define(
  'my-module',
  [],
  function() {
    return {
      aFunction: function aFunction() {
        ...
      }
    };
  });
```

All different kinds of module formats

- **Solution**:
  - convert CommonJS / ES6 / AMD -> Closure Library format (goog.*)

```
{:foreign-libs [{:file "/path/to/my-module.js"
                 :module-type :commonjs
                 :provides ["my.module"]}]}
```

2015: Initial Code Splitting Support

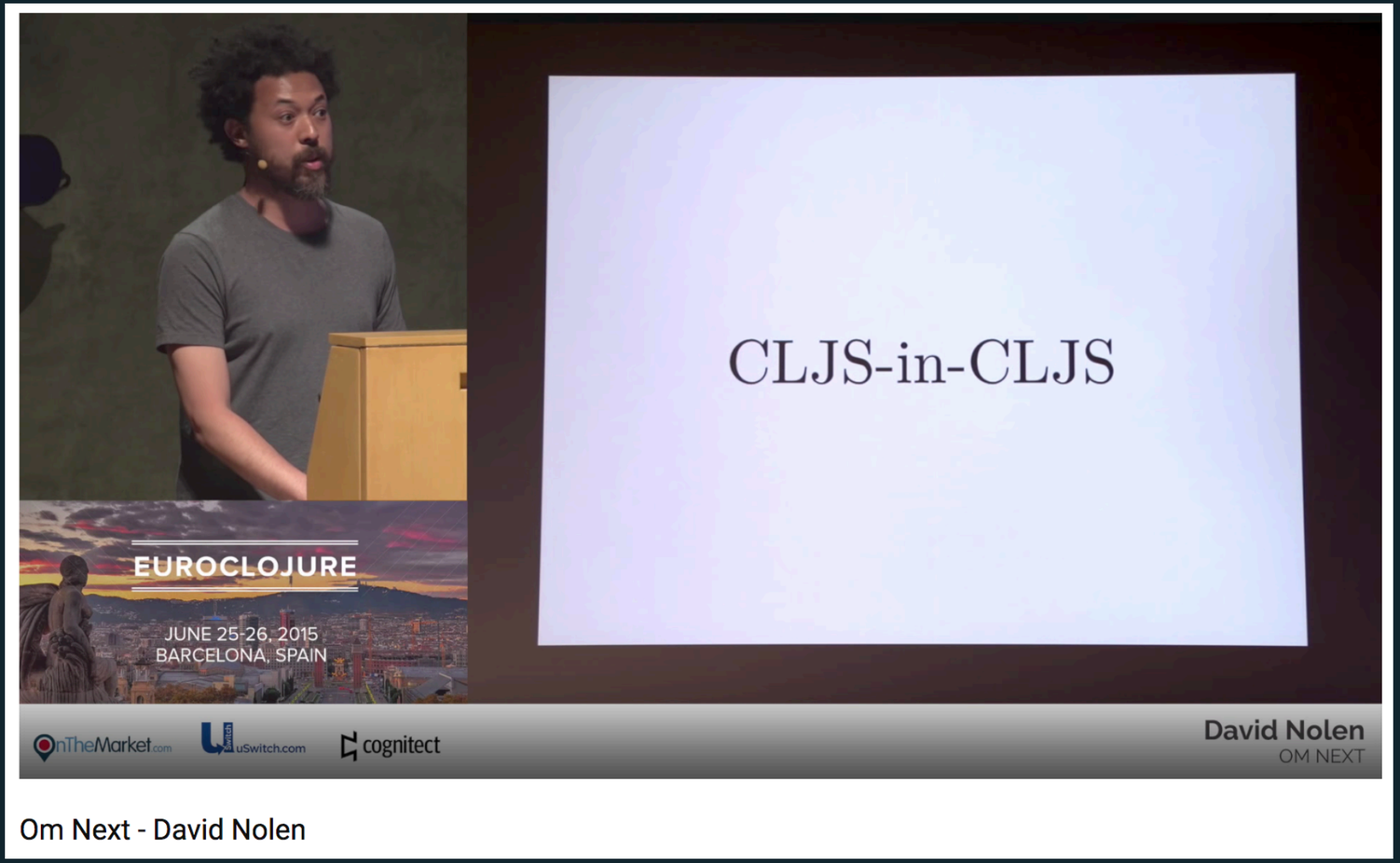
Hello Google Closure Modules

23 FEBRUARY 2015

By now you may have heard some buzz about webpack, a tool for managing web application assets. webpack can manage images and stylesheets, but only the facilities for managing JavaScript sources and more specifically the facilities for *code-splitting* are of interest to us in this post. We'll briefly look at webpack's support for splitting and compare it to a little known feature of the Google Closure Compiler: Google Closure Modules.

swannodette.github.io/2015/02/23/hello-google-closure-modules

2015: Self-hosted Compiler



Om Next - David Nolen

ClojureScript Next, Om Next - David Nolen

2016

2016: clojure -> cljs namespace aliasing

```
;; you write:
(require 'clojure.test)

;; the compiler knows you're requiring:
(require 'cljs.test)

;; in reality, becomes:
(require '[cljs.test :as clojure.test])
```

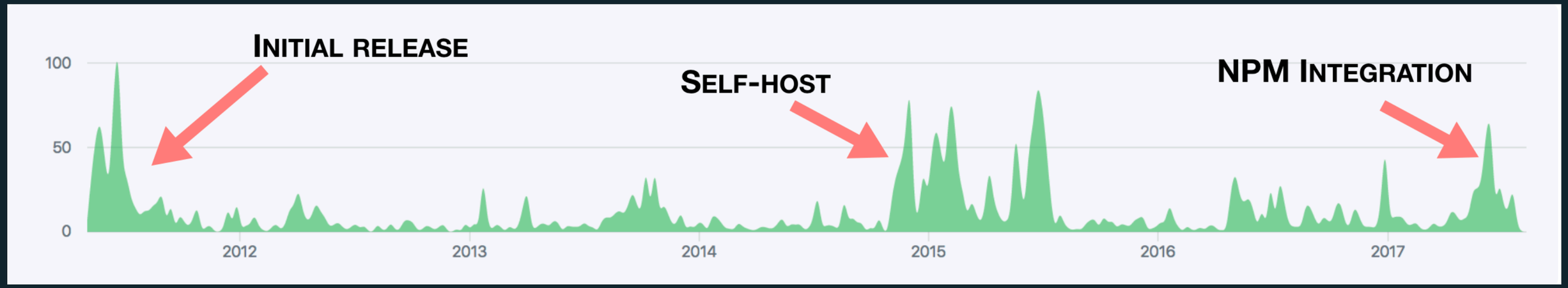
2016: Implicit Macro Inference in `:refer`

```
;; before:
(require '[cljs.test :refer-macros [deftest is]])

;; new:
(require '[cljs.test :refer [deftest is]])


;; putting it all together:
(require '[clojure.test :refer [deftest is]])
```

2016: even more enhancements...

— `require`, `refer-clojure`, optionally out of `ns` form
  — useful for scripting
  — deleted a bunch of REPL code
— extensible data reader support, "`data_readers.cljc`"

2017


INITIAL RELEASE
SELF-HOST
NPM INTEGRATION
100
50
0
2012
2013
2014
2015
2016
2017

Source

Commits on Sep 9, 2017

CLJS-2345: escape paths emitted as args to cljs.core.load_file
cemerick committed with **swannodette** on Aug 31 ✓ — `b74538b`

CLJS-2350: Fix circular dependencies test
anmonteiro committed with **swannodette** 24 days ago ✓

CLJS-2349: Port reset-vals! and swap-vals! ov
anmonteiro committed with **swannodette** 24 days ago ✓

All checks have passed
2 successful checks

✓ **continuous-integration/appveyor/branch** — AppV... Details

✓ **continuous-integration/travis-ci/push** — The Trav... Details

Commits on Aug 27, 2017

CLJS-2336: Call alength once in areduce and amap
mfikes committed with **swannodette** on Aug 26 ✓ — `998933f`

Commits on Aug 26, 2017

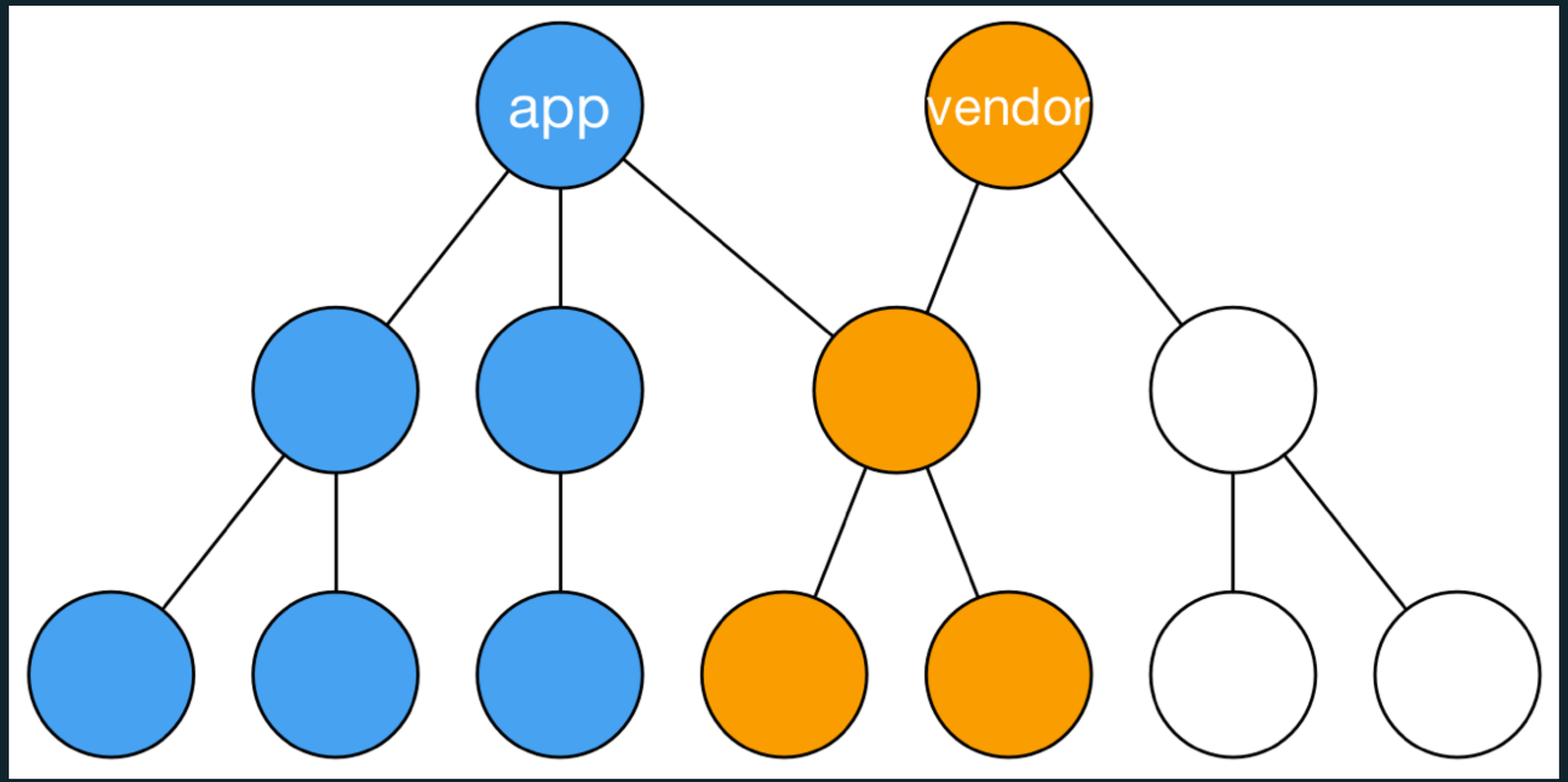
CLJS-2335: Avoid alength on strings
mfikes committed with **swannodette** on Aug 26 ✓ — `bee66e0`

Commits on Aug 25, 2017

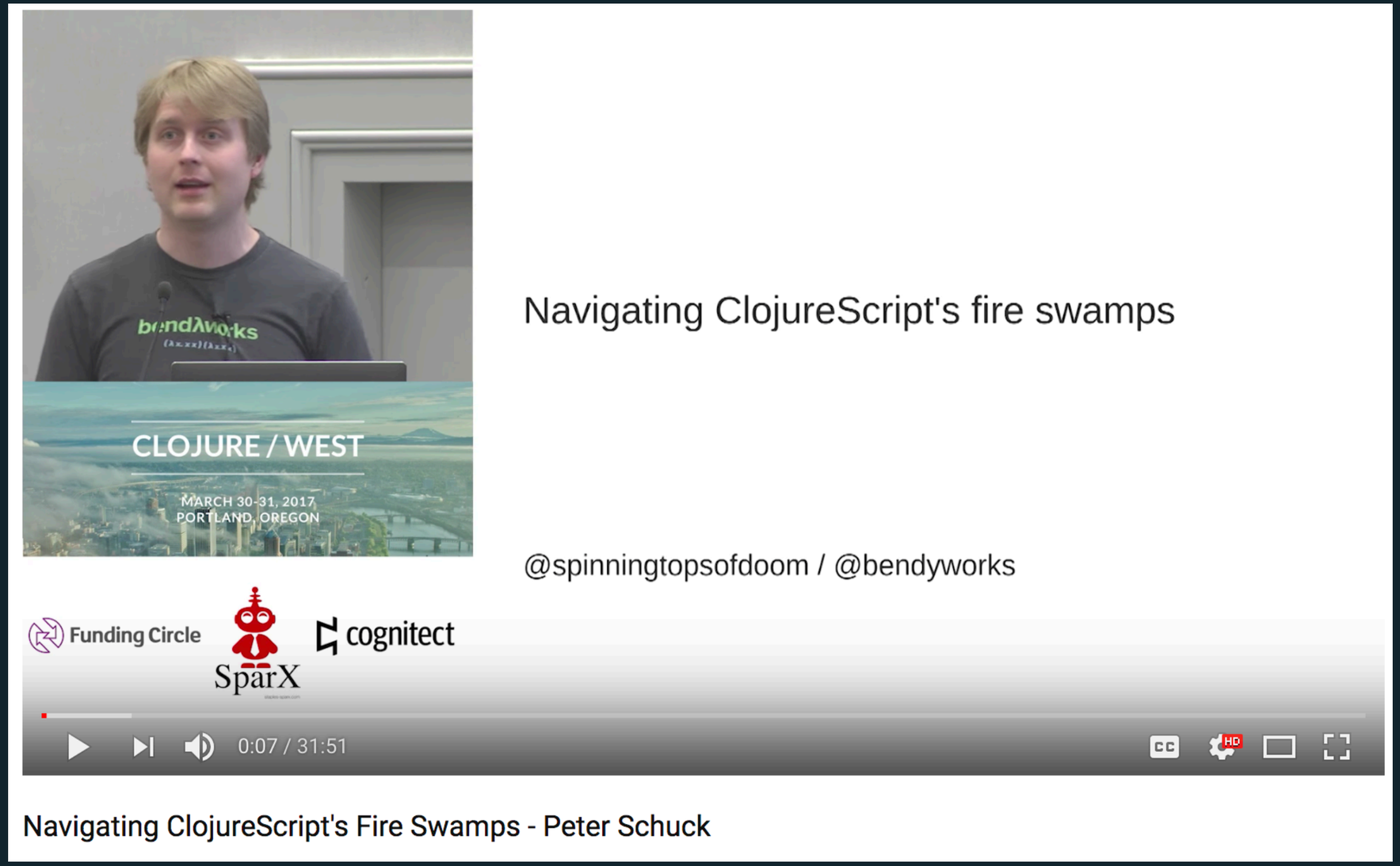
CLJS-2334: Also gather dependencies from foreign-libs that are modules
anmonteiro committed with **swannodette** on Aug 22 ✓ — `79041d1`

CLJS-2333: module-deps.js doesn't correctly compute `main` if aliased... ...
anmonteiro committed with **swannodette** on Aug 21 — `88e1f39`

Code Splitting



source



Navigating ClojureScript's Fire Swamps - Peter Schuck

Code Splitting: before

before:

```
{:modules {:public {:output-to "out/js/outer.js"
                    :entries #{"my-project.outer"}}

           :private {:output-to "out/js/inner.js"
                     ;; manually assigned namespaces
                     :entries #{"my-project.inner"
                                "reagent.core"
                                "cljs-time.core"
                                ...}}}}
```

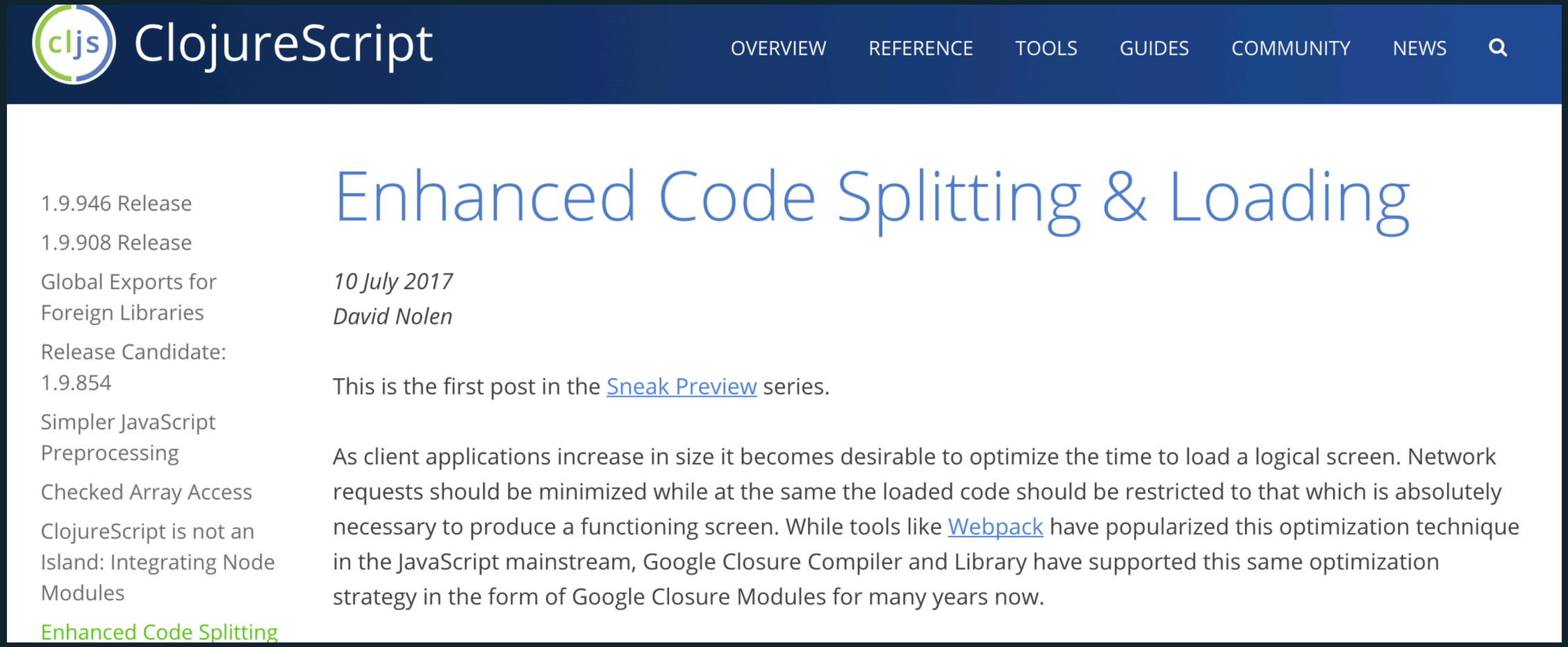
Module loading: before

```
(ns my-project.module-loader
  (:require [goog.module :as module]
            [goog.module.ModuleManager :as module-manager]
            [goog.module.ModuleLoader])
  (:import goog.module.ModuleManager))

(def modules
  ;; ids -> urls
  #js {"inner" "/js/inner.js"
       "outer" "/js/outer.js"})

(def module-info
  ;; module ids -> list of module dependencies.
  #js {"inner" [] "outer" []})
```

Module loading: before (cont.)

```
(def manager (module-manager/getInstance))
(def loader (goog.module.ModuleLoader.))
(.setLoader manager loader)
(.setAllModuleInfo manager module-info)
(.setModuleUris manager modules)

;; loading a module
(.execOnLoad manager "inner"
  (fn []
    ;; module loaded, call render function, change route, etc
    ))

;; inner.cljs
(-> goog.module.ModuleManager .getInstance (.setLoaded "inner"))
```

Enhanced Code Splitting & Module Loading



[clojurescript.org/news/2017-07-10-code-splitting

Enhanced Code Splitting

```
{:modules {:public {:output-to "out/js/outer.js"
                    :entries #{"my-project.outer"}}

           :private {:output-to "out/js/inner.js"
                     ;; assignments automatically calculated
                     :entries #{"my-project.inner"}}}}
```

Enhanced Code Splitting

```
{:modules {:public {:output-to "out/js/outer.js"
                    :entries #{"my-project.outer"}
                    :depends-on [:vendor]}

           :private {:output-to "out/js/inner.js"
                     :entries #{"my-project.inner"}
                     :depends-on [:vendor]}

           :vendor {:output-to "out/js/vendor.js"
                    :entries #{"reagent.core"}}}}
```

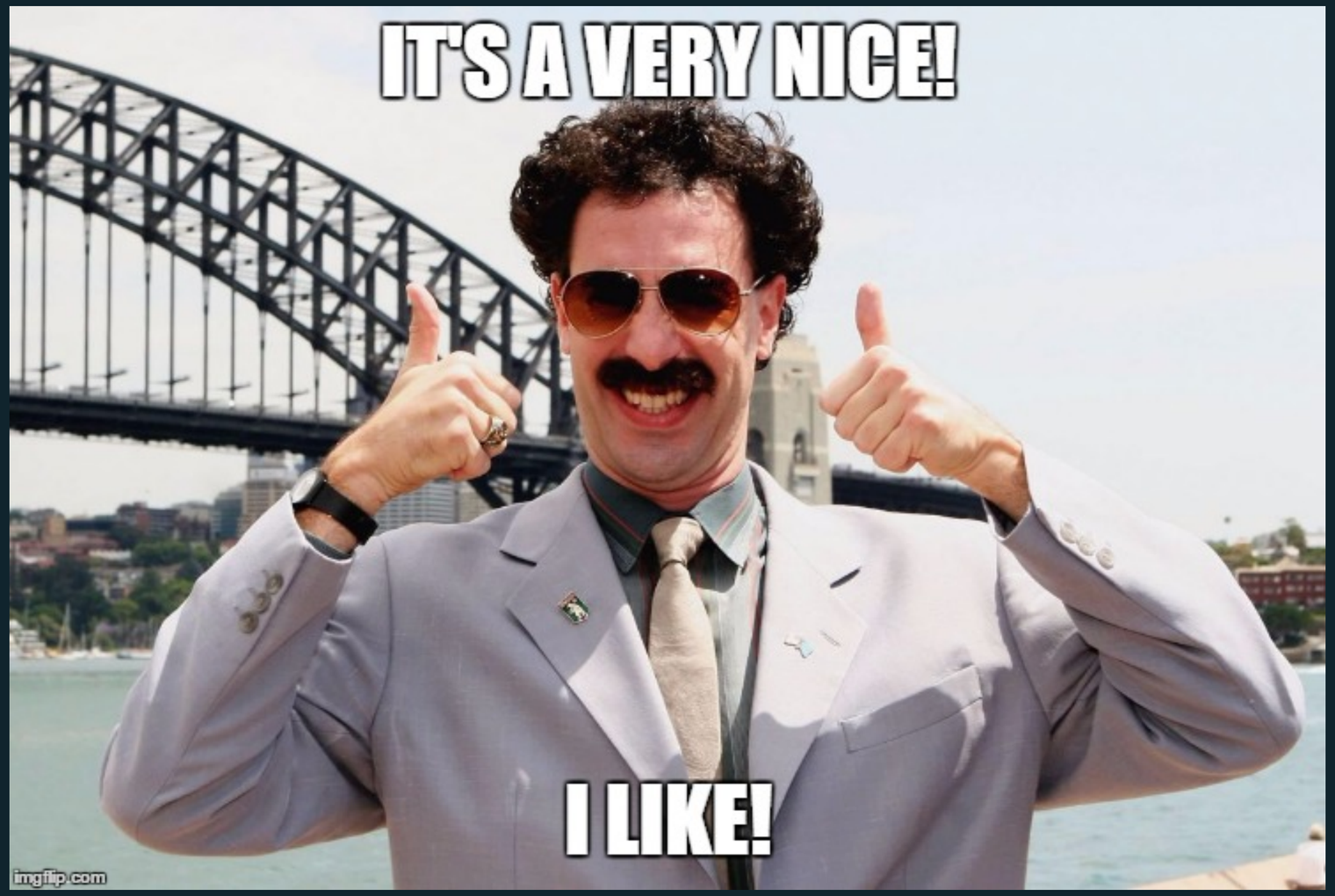
Enhanced Code Splitting

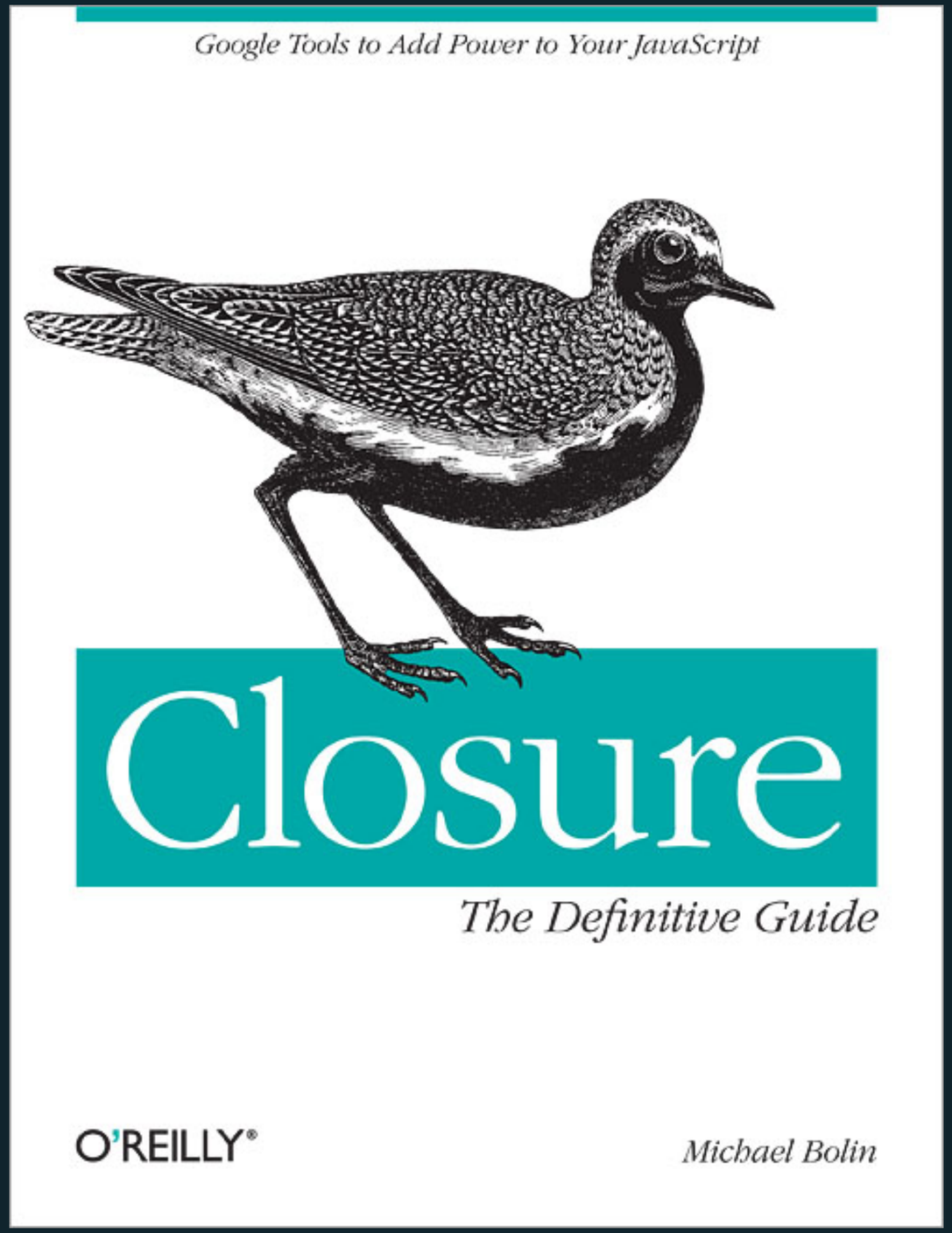
```
{:modules {:home-page {:output-to "out/js/home.js"
                       :entries #{"my-project.home"}
                       :depends-on [:common]}

           :about-page {:output-to "out/js/inner.js"
                        :entries #{"my-project.about"}
                        :depends-on [:common]}

           :common {:output-to "out/js/common.js"
                    ;; empty entries!
                    :entries #{}}}}
```

Enhanced Module Loading

— cljs.loader & cljs.core/resolve

```
(ns my-project.homepage
  (:require [cljs.loader :as loader]))

(loader/load :about-page
  (fn [e]
    ((resolve 'my-project.about-page/render!))))

(loader/set-loaded! :home-page)
```


IT'S A VERY NICE!
I LIKE!
imgflip.com


Google Tools to Add Power to Your JavaScript
Closure
The Definitive Guide
O'REILLY®
Michael Bolin

Link


npm
BABEL
JS
REQUIRE·JS

ClojureScript is not an Island: Integrating Node Modules

12 July 2017
António Nuno Monteiro

This is the second post in the Sneak Preview series.

ClojureScript has had first class JavaScript interop since its initial release in 2011. Similarly to Clojure, embracing the host has always been an explicit goal. While the above is true from a syntax standpoint, integrating with external JavaScript libraries historically came at the cost of some manual work [1] (manually assembled bundles or packaging led by community efforts).

clojurescript.org/news/2017-07-12-clojurescript-is-not-an-island-integrating-node-modules

Pre-historic ClojureScript

- bundle everything, consume through `:foreign-libs`
  - CLJSJS works this way
- `:foreign-libs` manual labor
  - initial module support worked this way

Goals for a modern interop story

- How do I consume NPM modules?
  - In a way that Closure understands
    - (to apply sophisticated optimizations)
- How do I integrate JavaScript sources in a ClojureScript project?
  - Including JSX (Babel & co.)

How do I consume NPM modules?

```
;; ClojureScript compiler options
{:npm-deps {:react "15.6.0"
            :react-dom "15.6.0"}
 :install-deps true}
```

```
;; my_project/core.cljs
(ns my-project.core
  (:require [react :refer [createElement]]
            [react-dom]))

(react-dom/render
  (createElement "div" nil "Hello, Clojure/Conj!")
  (js/document.getElementById "app"))
```

How do I consume NPM modules?

```clojure
;; my_project/core.cljs
(ns my-project.core
  (:require [react :refer [createElement]]
            ;; NEW: String requires (arbitrarily nested paths)
            ["react-dom/server" :as react-dom-server]))


(react-dom-server/render-to-str
  (createElement "div" nil "Hello, Clojure/Conj!"))
```

How do I consume NPM modules?

```
// my-module.js
module.exports = function(a, b) {
  ...
};
```

```
;; my_project/core.cljs
(ns my-project.core
  (:require my-module))

(my-module 1 2)
```

How do I integrate with JavaScript sources?

```
;; ClojureScript compiler options
{;; OPTIONAL, ClojureScript knows how to index JS sources own deps
 :npm-deps {:react "15.6.0"
            :react-dom "15.6.0"}
 :install-deps true
 :foreign-libs [{:file "path/to/js-dir"
                 :provides ["js.sources"]}]}
```

```
;; my_project/core.cljs
(ns my-project.core
  (:require [js.sources :as js-sources]))

;; call into js-sources!
```

Preprocessors / JS compilers

```
;; ClojureScript compiler options
{:foreign-libs [{:file "path/to/js-dir"
                 :provides ["js.sources"]
                 ;; NEW
                 :preprocess 'my-preprocessor.core/transform-jsx}]}
```

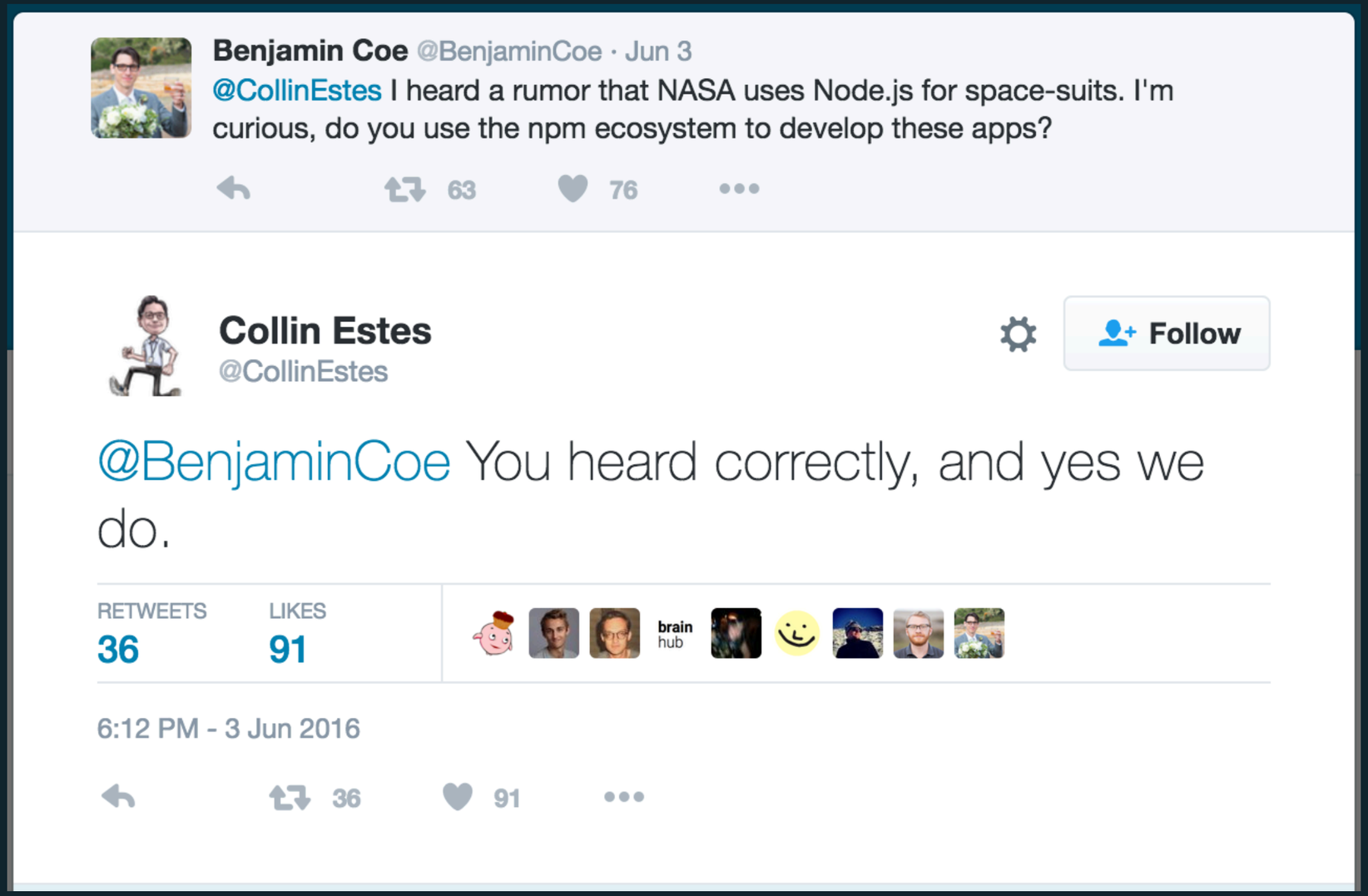
```
(ns my-preprocessor.core)

;; use, e.g. Nashorn to convert JSX -> JavaScript through Babel
;; could also shell out to Node.js, etc
```

No breakage

```
;;before
(ns foo
  (:require [cljsjs.react]))

(def react js/React)


;; now (compiler options):
{:foreign-libs [{:file "/path/to/react.js"
                 :provides ["cljsjs.react" "react"]
                 ;; NEW: provides => JS global variable
                 :global-exports '{cljsjs.react React
                                   react React}}]}

(ns foo
  (:require react))
```

What can ClojureScript do for you in 2017?

- ClojureScript dependencies
- Wider JS ecosystem integration (NPM, Yarn, etc)
- code splitting through Closure
  - first-class chunks, cross module code motion, etc.
- target web, Node.js, React Native, etc.



twitter.com/CollinEstes/status/738767017843515393