Haxe React “Magic”

Or: how I learned to stop building everything myself and love the JavaScript ecosystem.

The JavaScript ecosystem is ripe for plunder

- 250K public JavaScript packages
- Most active language on Github
- Google, Facebook, Microsoft, Netflix,…
  - All fighting for developers' attention

The JavaScript ecosystem is ripe for plunder

The JavaScript ecosystem is ripe for plunder

- Qualified monkeys on expensive typewriters?
- Bound to stumble on a good idea eventually!

First, understand the enemy

- Writing effective externs requires good understanding
- Learn how things work in vanilla JavaScript first

First, understand the enemy

- Anything you can do in JavaScript can be expressed in Haxe (depending on how dirty you're willing to feel)
- For articles on Haxe-JS interaction: http://philippe.elsass.me

Here's the plan

1. React
2. Code splitting
3. JS bundlers
4. Profit!

React: all the cool kids are doing it

- Essentially a nice way of building views
- Refreshingly productive and straight forward

React: all the cool kids are doing it

- XML in your JavaScript? (heresy!)
- It's called JSX

```
var CommentBox = React.createClass({
  render: function() {
    return (
      <div className="commentBox">
        Hello, world! I am a CommentBox.
      </div>
    );
  }
});
```

React: all the cool kids are doing it

```
var CommentBox = React.createClass({
  render: function() {
    return (
      React.createElement('div', {className: "commentBox"},
        "Hello, world! I am a CommentBox."
      )
    );
  }
});
```

React: all the cool kids are doing it

- Fancy binding syntax (uni-directional)

```
const className = 'commentBox';
const hello = 'Hello, world! I am a CommentBox.';
return (
  <div className={className}>
    {hello}
  </div>
);
```

React: all the cool kids are doing it

- React is a Virtual DOM engine from Facebook/Instagram
- Not a framework / not an architecture (unlike Angular)
- Straightforward and robust

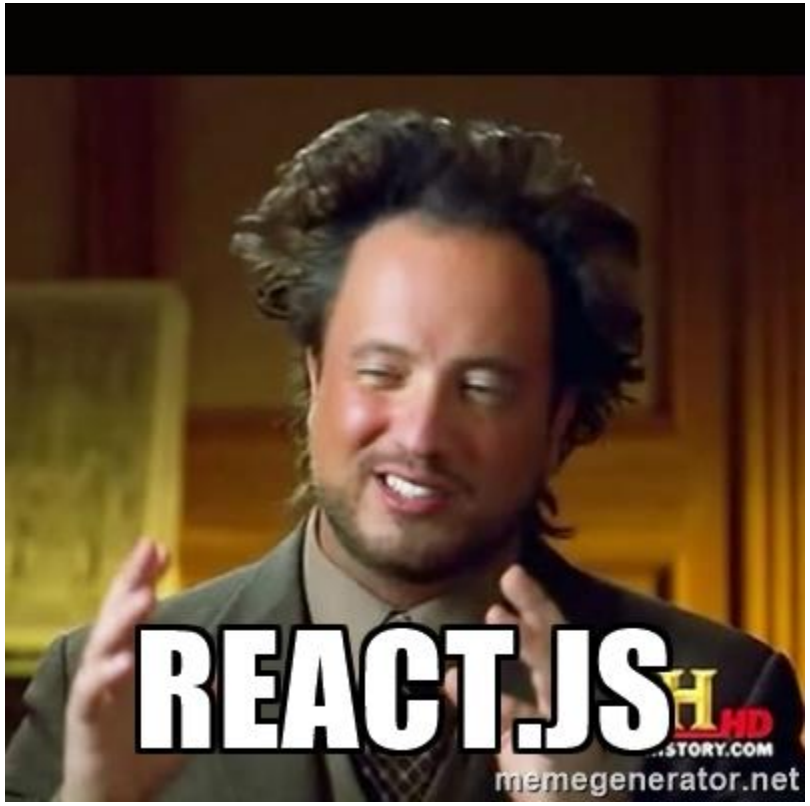
React: Native ???

- Potential outside the browser
- React-Native aims to drive native (iOS, Android, Win, Mac,…) using a JavaScript Virtual DOM
- Similar approach than *Titanium Appcelerator*

- *Warning: expert native knowledge is still recommended*

Haxe-React

- First experimented by @fponticelli, leader of the JavaScript assimilation
- New approach from scratch by *Massive Interactive*'s @dpeek and @elsassph
- Fully leverages Haxe type system and compiler



Haxe-React

- Nearly as fancy, macro-powered, syntax

```
override function render()
{
    return jsx('
        <div className="commentBox">
            Hello, world! I am a CommentBox.
        </div>
    ');
}
```

Haxe-React

- Supports the same binding syntax

```
var className = 'commentBox';
var hello = 'Hello, world! I am a CommentBox.';
return jsx('
    <div className={className}>
        {hello}
    </div>
');
```

Haxe-React

- Or Haxe string-interpolation syntax (advantage: completion)

```
var className = 'commentBox';
var hello = 'Hello, world! I am a CommentBox.';
return jsx('
    <div className=$className>
        ${hello.toUpperCase()}
    </div>
');
```

Haxe-React

- JSX parser is complete but has a few limitations (whitespace)
- Generator supports all the advanced features like spread operator and optimized syntax (inline JSON instead of React.createElement)

Haxe-React

- Lots of externs for JS React components are already available https://github.com/tokomlabs/haxe-react-addons
- Could be extended to other Virtual DOM engines, like https://github.com/Matt-Esch/virtual-dom or a Haxe-based one ;)

```
// Render jsx
return jsx('
    <form className="commentForm">
      <input
        type="text"
        placeholder="Your name"
        value={hello("world", true, [1,2,3,4], 4, {that: "is", the: Type.prop})}
        onChange={handleAuthorChange}
      />
      <input
        type="text"
        placeholder="Say something..."
        value={state.text}
        onChange={handleTextChange}
      />
      Some text
      <input type="submit" value="Post" />
    </form>
');
```

Atom Haxe+JSX highlighting: https://github.com/massiveinteractive/haxe-react/issues/23

How to link React library with Haxe?

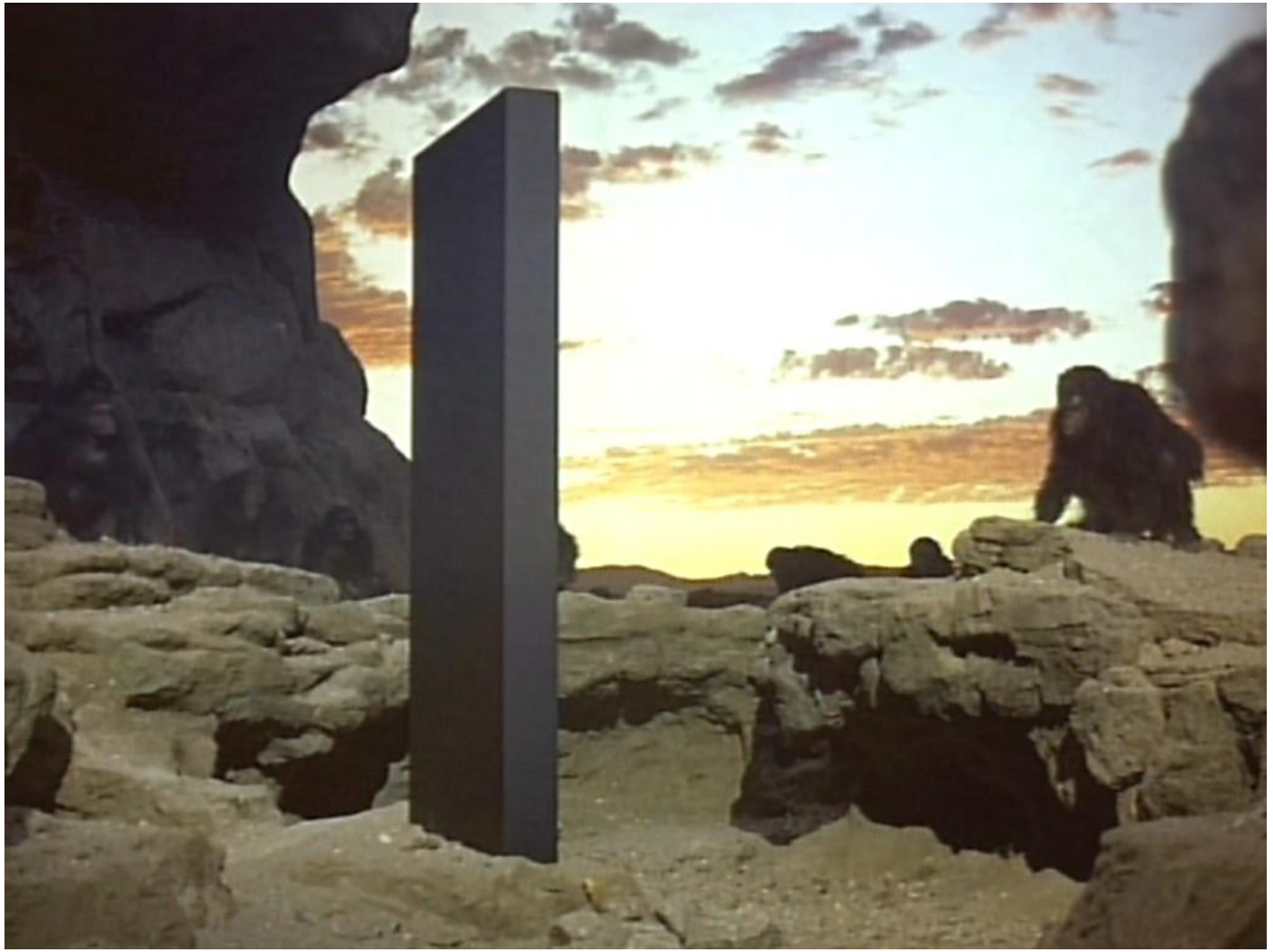
- Ambient (global), with script in the HTML
  - Compile with: `-D react_global`

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/react/15.0.1/react-with-addons.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/react/15.0.1/react-dom.js"></script>
<script src="index.js"></script>
```

- Or using JavaScript require (more on that later)

React: production-ready

- It works great, right now, it's stable and well supported. Enjoy.



Code splitting

- At some point, your app will get too big
- On the web that can be sooner than you think

Code splitting

- Until recently trend was concat/minify
- Now it's all about dependency graphs and optimizing entry-points
- Can we follow this practice in Haxe?

Modular JS

https://github.com/explorigin/modular-js

Modular JS

https://github.com/explorigin/modular-js

- Create one JS file for each Haxe class, using AMD module system
- Then use a JavaScript bundler (more on that later) to recombine it

- Risks: experimental, requires a fully custom JavaScript generator

Modular Haxe

https://github.com/elsassph/modular-haxe-example

Modular Haxe

https://github.com/elsassph/modular-haxe-example

- Break monolithic codebase, with full type integrity, into multiple JS modules
- Natural fit for lazy-loading of large features

- Risk: experimental, requires to lightly patch the compiler output

Modular Haxe – How?

https://github.com/elsassph/modular-haxe-example

- Use `--exclude` or `--excludeFile` to exclude classes/packages
  - Eg. compile Main app excluding Module code,
  - and Module excluding shared code
- Use `@:expose` to mark classes shared between modules

- Patched JS allows the modules to merge their code
  - Think: Flash Runtime Shared Libraries, eg. loading SWFs in the same ApplicationDomain

Modular Haxe – Usage

https://github.com/elsassph/modular-haxe-example

- Lazy loading can’t be simpler

```
import module1.Module1;

Require.module('module1').then(function() {
    var module = new Module1();
});
```

- (helper can even load and apply a CSS file at the same time)

Modular Haxe – Limitation

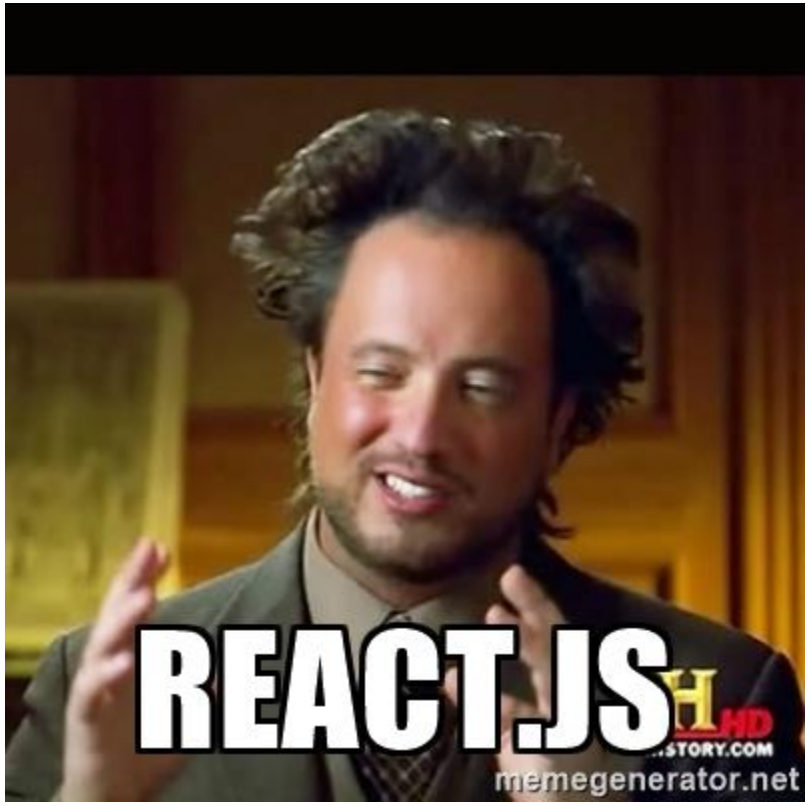
https://github.com/elsassph/modular-haxe-example

- Not designed for fine granularity
- You can't `@:expose` everything (and change existing Haxe libraries)
- Expect some redundancy (core classes, 3rd party libraries)

- Can you guess what would work well?

Modular Haxe + React!

- React library is external and global
- Modularization based on pages or complex components (ex: video player)
- Modules can encapsulate views and related state/logics.
- Minor redundancies to expect and limited `exclude/expose` effort




Intermission

Npm – node.js package manager

- The tool you will use to manage external JavaScript modules
- Learn it, tame it, it’s unavoidable and unstoppable

Npm – node.js package manager

- Also a huge repository of (usually crossplatform) command line tools

```
> npm install http-server -g
> http-server ./www
Starting up http-server, serving ./www on: http://0.0.0.0:8080
Hit CTRL-C to stop the server
```

Module systems

require(‘react’)

Module systems: CommonJS

- Most popular (thanks to *node.js*)
- Synchronous by design
- Needs to be *transformed* for the browser

```
const React = require('react');
const Counter = require('./counter');
```

Module systems: CommonJS

- Fits very naturally with Haxe classes / imports

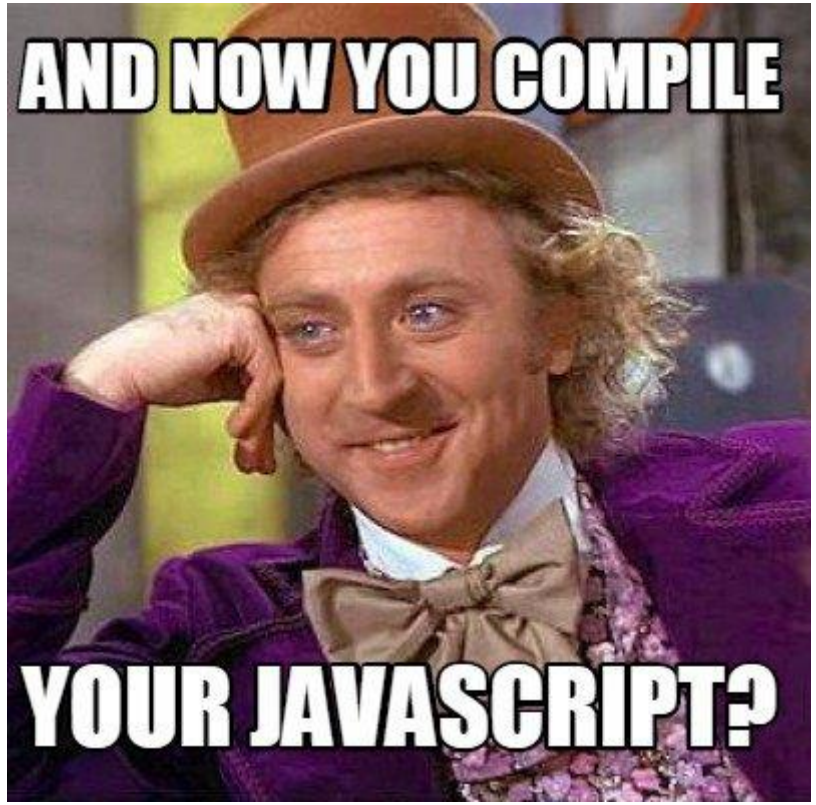
```haxe
// Haxe
@:jsRequire("react", "Component")
extern class ReactComponentOf<TProps, TState, TRefs>

// JavaScript
var React_Component = require("react").Component;
```

Module systems: AMD

- Asynchronous by design
- Browser-friendly, using *SystemJS* or *RequireJS* at runtime
- Can be *bundled* into one JS

```
define(['react', './counter'], function (React, Counter) {
```

- This is what *modular-js* generates


AND NOW YOU COMPILE
YOUR JAVASCRIPT?

Module bundling

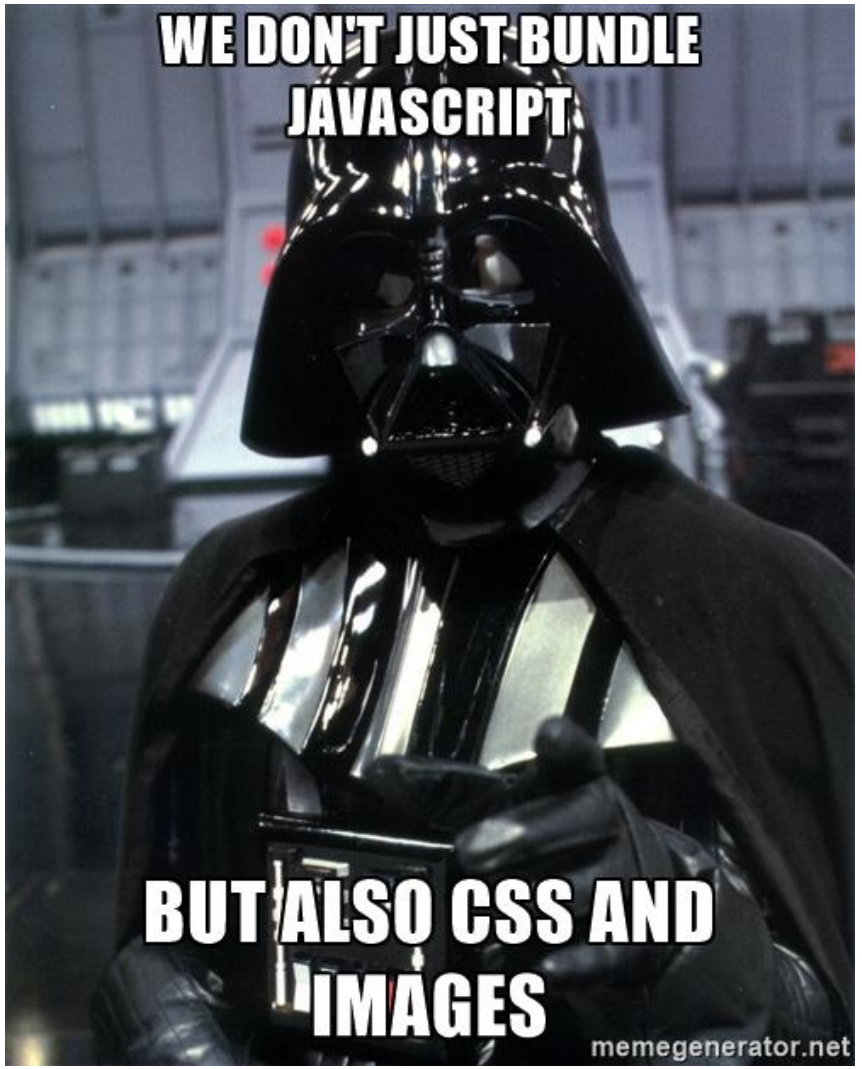
Aka compiling JavaScript

Module bundling

- Most bundlers support both CommonJS and AMD modules
- Bundlers package modules and their dependencies into one or more (code splitting) JS *bundles*
- Dependencies can be transformed/processed by *loaders*

Module bundling

- Dependencies are not only code: *loaders* can also transform other assets, like raw text, JSON, CSS,… and even images.
- Which means you can easily package assets in your JS bundles.



Choosing a bundler

https://webpack.github.io/docs/comparison.html

(slightly biased)

Browserify

- Oldest, so “less cool”
- Very simple, fast and lightweight tool
- Loaders are plugins (batteries not included)
- Lots of misinformation around its (lack of) features but it’s as capable as the cool kids: https://t.co/Wi0G3spkIB

Webpack

- Somehow the most popular / cutting edge
- Provides many loaders out of the box
- “Cumbersome” to configure


Jeremy Raines
@jraines
Follow
nothing just setting up my ES6 Webpack React Babel Node Flux Redux static site generator

JSPM

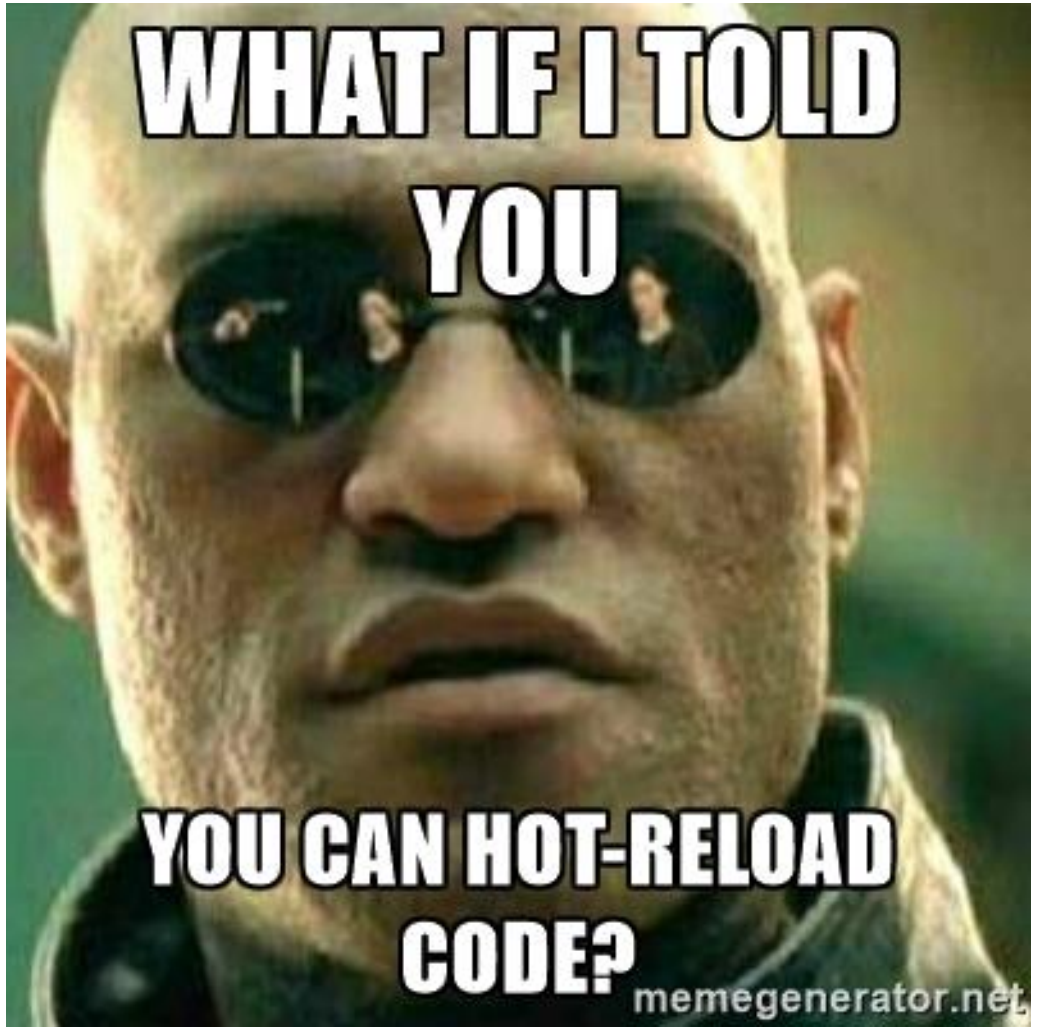
- Package manager and bundler
- Based on SystemJS (dynamic modules loader)
- Simple to use
- Renewed popularity (it used to be slow)
- Smart code-splitting (*bundle arithmetic*)

And using Haxe?

Any bundler will work, really.

One more thing…

Putting it all together


WHAT IF I TOLD YOU
YOU CAN HOT-RELOAD CODE?
memegenerator.net

Hot Module Replacement

- Aka Hot Reload, Live Reload,…
- Must have feature if you’re not has-been
- Some assets auto-reload, like CSS.
- For code it’s not magic: the application needs special logic to *handle* changes and refresh itself
- Every module bundler has its API.

Haxe + React + Hot reload

http://github.com/elsassph/haxe-react-hot-boilerplate