JESSE WARDEN | RVA.JS | NOV 3 2015

SYSTEM, WEBPACK, & JSPM

WHAT?

- DevOps & Lean Engineering
- Brief on Angular/React/Backbone & Node Workflow
- Deep Dive in JavaScript Modules

LEAN ENGINEERING PROCESS V02

DEVOPS & LEAN ENGINEERING

ANGULAR AND NODE IN AOWP V3

ANGULAR & NODE WORKFLOW

ANGULAR 2 OVERVIEW

ANGULAR 2

MODULES

JAVASCRIPT

GLOBAL

- In the beginning... there was window

```
window.foo = "bar";
console.log(foo); // bar
function moo()
{
    console.log(foo); // bar
}
```

GLOBAL BAD

- global variables: they fast…
- … but no clue who’s setting them, access control
- global state
- no control

EXTERNAL FILES

- load via block
- parse
- "run"

```
<html>
<head>
<title>alskdjf</title>
<script src="a.js"></script>
<script src="b.js"></script>
</head>
<body></body>
</html>
```

A.JS VS. B.JS

```
function cow()
{
    console.log("cow a");
}
moo = "cheese";
var bessie = "Some cow, yo!";
```

```
function cow()
{
    console.log("cow b");
}
moo = "dat brie doh";
var bessie = "Bee A Cow";
```

SCRIPT PROBLEMS

- order issue
- flat dependency != tree

COMMONJS

- Encapsulation / Global Safe (unless you r t3h l@m3 and use global)
- Module system
- Version Safe (via local node_modules)
- synchronous
- dependent loading
- Node (server or build system)

COMMONJS

```
//------ lib.js ------
var sqrt = Math.sqrt;
function square(x) {
    return x * x;
}
function diag(x, y) {
    return sqrt(square(x) + square(y));
}
module.exports = {
    sqrt: sqrt,
    square: square,
    diag: diag,
};
```

```
//------ main.js ------
var square = require('lib').square;
var diag = require('lib').diag;
console.log(square(11)); // 121
console.log(diag(4, 3)); // 5
```

REQUIRE.JS

- asynchronous
- different than <script defer>
- back when "pages" vs. "views" or "components" nomenclature
- CDN hosted libs
- "load on the fly"
- minor dependency injection
- build via r.js

REQUIRE.JS

```
// config.js
requirejs.config({
    baseUrl: 'js/lib',
    paths: {
        jquery: 'jquery-1.9.0'
    }
});
```

```
// view.js
define(["hbs!some.html", "a"], function(template, a)
{
    console.log(template); // html text
    console.log(a); // my a library
})

// main.js
require(['jquery', 'view'], function ($, view)
{
    $.append(view);
});
```

GRUNT / GULP

- Angular (ok)
- Manual Dependency Ordering (insane, no scale)

```
sourceFiles:      ['*.module.js',
                   '*.js',
                   '**/*.module.js',
                   '**/*.js',
                   '!*.spec.js',
                   '!**/*.spec.js',
                   '!Gruntfile.js',
                   '!gulpfile.js'],
```

```
return new Promise(function(resolve, reject)
{
      gulp.src(CONFIG.client.sourceFiles)
     .pipe(gulp.dest('./build'))
     .on('end', resolve)
     .on('error', reject);
});
```

BROWSERIFY

- "work everywhere"
- Node rocks with CommonJS, let’s use it
- "universal JavaScript"
- i.e. lodash
- build system, not runtime
- supports bundles (atomify, cssify, tsify)
- https://github.com/substack/browserify-handbook

BROWSERIFY

```
// foo.js
module.exports = function (n) { return n * 111 }

// main.js
var foo = require('./foo.js');
console.log(foo(5));
```

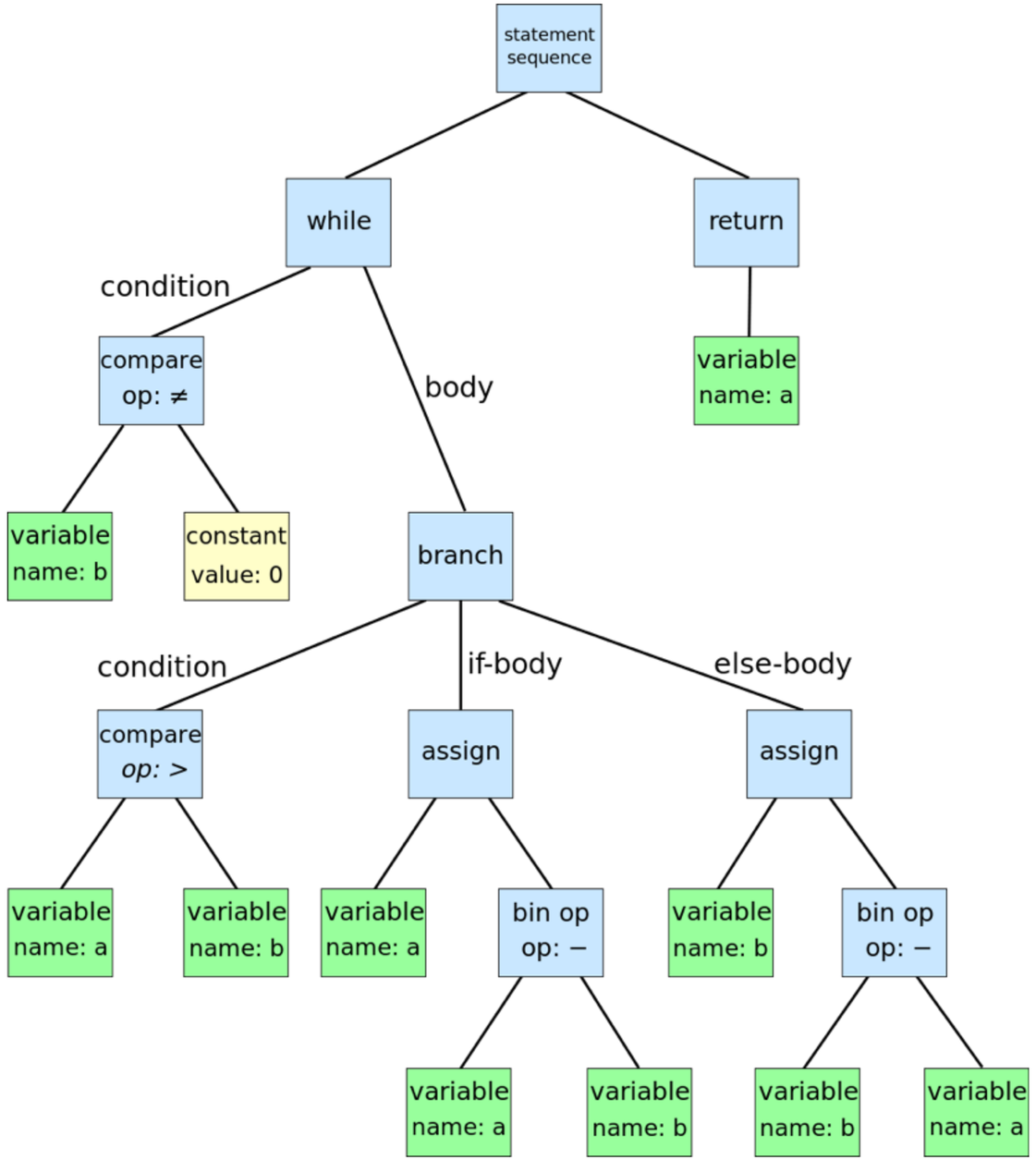
BROWSERIFY

```
browserify robot.js > bundle.js
```

```
<html>
  <body>
    <script src="bundle.js"></script>
  </body>
</html>
```

BROWSERIFY

- static analysis (uses detective)
- creates Abstract Syntax Tree (AST)



BROWSERIFY

- injectable
- hot replace
- via beefy, wzrd, browserify-middleware, etc.

BROWSERIFY

```
var browserify = require('browserify');
var http = require('http');

http.createServer(function (req, res) {
    if (req.url === '/bundle.js') {
        res.setHeader('content-type', 'application/javascript');
        var b = browserify(__dirname + '/main.js').bundle();
        b.on('error', console.error);
        b.pipe(res);
    }
    else res.writeHead(404, 'not found')
});
```

BROWSERIFY

- built in classes for Node in the browser
- transforms (CoffeeScript, TypeScript, CSS, images, etc)
- customizable browser fallback

```
{
  "name": "mypkg",
  "version": "1.2.3",
  "main": "main.js",
  "browser": "browser.js"
}
```

```
// Node
var p = require("mypkg");
// p is main.js exports

// Browser
var p = require("mypkg");
// p is browser.js exports
```

ES6

- ECMAScript 2015
- No one calls it that except for standards people
- OOP: Yay, classes!
- Functional: Yay, new features!
- DevOps: Yay, static dependency tree!
- http://exploringjs.com/es6/ch_modules.html

ES6

```
//------ lib.js ------
export const sqrt = Math.sqrt;
export function square(x) {
    return x * x;
}
export function diag(x, y) {
    return sqrt(square(x) + square(y));
}

//------ main.js ------
import { square, diag } from 'lib';
console.log(square(11)); // 121
console.log(diag(4, 3)); // 5
```

```
//------ main.js ------
import * as lib from 'lib';
console.log(lib.square(11)); // 121
console.log(lib.diag(4, 3)); // 5
```

ES6

```
//------ myFunc.js ------
export default function () { ··· } // no semicolon!

//------ main1.js ------
import myFunc from 'myFunc';
myFunc();
Or a class:

//------ MyClass.js ------
export default class { ··· } // no semicolon!

//------ main2.js ------
import MyClass from 'MyClass';
let inst = new MyClass();
```

ES6

```
//------ lib.js ------
var sqrt = Math.sqrt;
function square(x) {
    return x * x;
}
function diag(x, y) {
    return sqrt(square(x) + square(y));
}
module.exports = {
    sqrt: sqrt,
    square: square,
    diag: diag,
};

//------ main.js ------
var square = require('lib').square;
var diag = require('lib').diag;
console.log(square(11)); // 121
console.log(diag(4, 3)); // 5
```

ES6

	Scripts	Modules
HTML	<script>	<script type="module">
Top-Level variables are	global	local to module
Value of this at top level	window	undefined
Executed	sync	async
import statement	no	yes
Promise-based API	yes	yes
File extension	.js	.js

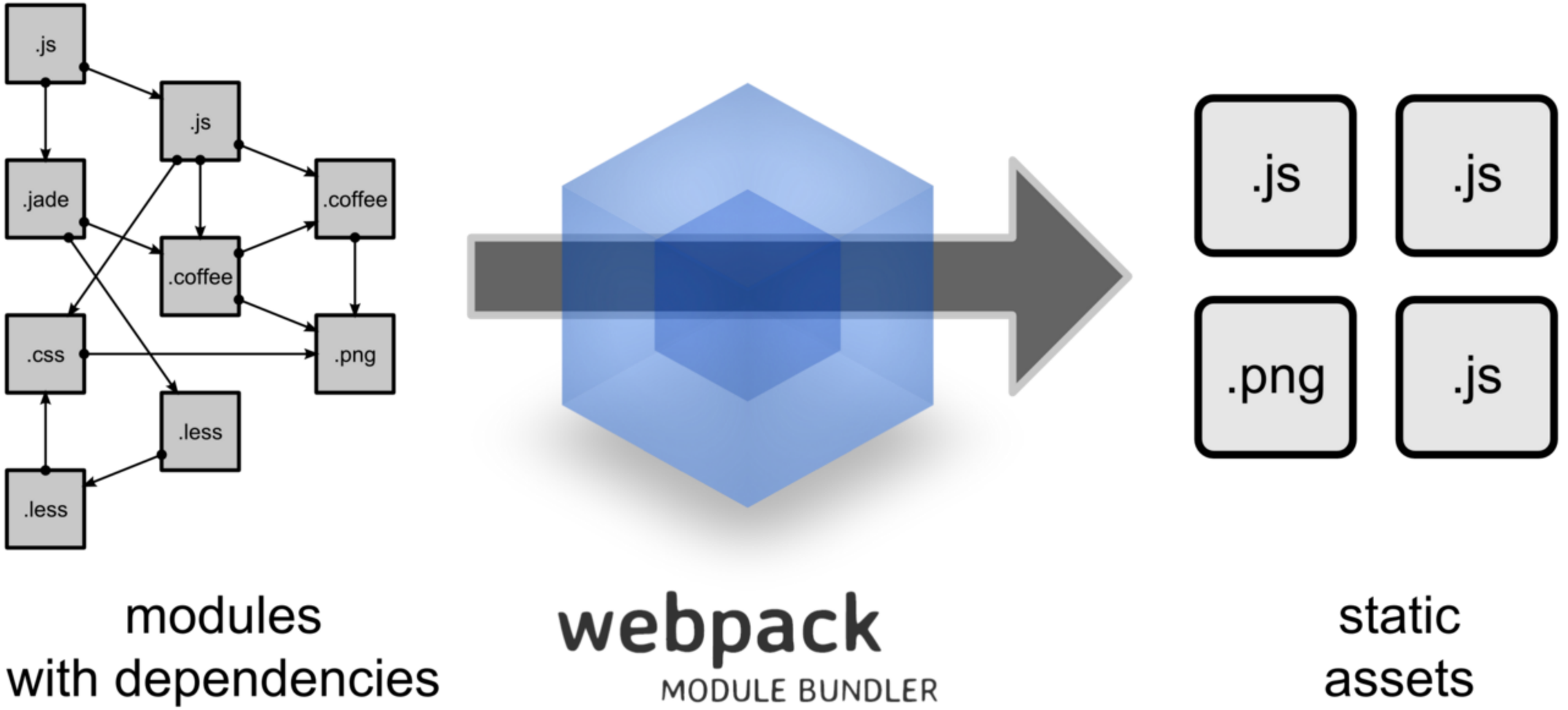
ES6

- better cyclic dependencies
- module renaming on consumer
- import { name1 as localName1, name2 } from ‘src/my_lib';
- static initializer
- import 'src/my_lib';

WEBPACK

- better Require.js
- more features / API's than Browserify
- built in vs. plugins/3rd party
- does that matter…?
- … only for ES6
- … AND RUNTIME
- https://webpack.github.io/

WEBPACK


.js
.js
.jade
.coffee
.coffee
.css
.png
.less
.less
modules
with dependencies
webpack
MODULE BUNDLER
.js
.js
.png
.js
static
assets

WEBPACK

```
// dependencies can be written in CommonJs
var commonjs = require("./commonjs");
// or in AMD
define(["amd-module", "../file"], function(amdModule, file) {
    // while previous constructs are sync
    // this is async
    require(["big-module/big/file"], function(big) {
        // for async dependencies webpack splits
        //  your application into multiple "chunks".
        // This part of your application is
        //  loaded on demand (Code Splitting)
        var stuff = require("../my/stuff");
        // "../my/stuff" is also loaded on demand
        //  because it's in the callback function
        //  of the AMD require
    });
});
```

```
require("coffee!./cup.coffee");
// "Loaders" can be used to preprocess files.
// They can be prefixed in the require call
//  or configured in the configuration.
require("./cup");
// This does the same when you add ".coffee" to the extensions
//  and configure the "coffee" loader for /\.coffee$/
```

WEBPACK

- bundle it

```
webpack ./entry.js bundle.js
```

```
<html>
    <head>
        <meta charset="utf-8">
    </head>
    <body>
        <script type="text/javascript" src="bundle.js" charset="utf-8"></script>
    </body>
</html>
```

WEBPACK

- npm install webpack-dev-server -g
- webpack-dev-server –progress –colors
- Code Splitting: http://webpack.github.io/docs/code-splitting.html

ES6-MODULE-LOADER

- programmatically work with modules
- configure module loading
- Loader: NOT part of the standard
- System: NOT part of the standard
- … both are expected to be
- https://github.com/ModuleLoader/es6-module-loader

ES6-MODULE-LOADER

- Provides System.import
- Runtime, not build time
- Assumes Traceur, Babel, or TypeScript
- ES6 circular refs
- paths
- [show basic]

ES6-MODULE-LOADER

```
System.paths['jquery'] = '//ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js';
System.import('jquery').then(function($)
{
    // ...
});
```

ES6-MODULE-LOADER

```
var System = require('es6-module-loader').System;
/*
*  Include:
*    System.transpiler = 'babel';
*  to use Babel instead of Traceur or
*    System.transpiler = 'typescript';
*  to use TypeScript
*/

System.import('some-module').then(function(m) {
    console.log(m.p);
});
```

SYSTEMJS

- Built on ES6-Module-Loader
- loads any module format (CommonJS, AMD, ES5, ES6, None)
- Supports RequireJS-style maps, paths, shims, etc.
- Loader plugin works with CSS, JSON, and Images
- Browser + Node
- Gallons of plugins like Browserify & Webpack have

SYSTEMJS

- Zebra Striping
- Modules
- Standalone

SYSTEMJS

```
<script src="system.js"></script>
<script>
  // set our baseURL reference path
  System.config({
    baseURL: '/app'
  });

  // loads /app/main.js
  System.import('main.js');
</script>
```

SYSTEMJS

```
var System = require('systemjs');

System.transpiler = 'traceur';

// loads './app.js' from the current directory
System.import('./app').then(function(m) {
  console.log(m);
});
```

JSPM

- package manager for SystemJS
- called ES6 module loader (but you know better)
- loads from npm and Github
- dev == load, prod == standalone (or load)
- CLI for installing; use jspm instead of npm
- global registry
- http://jspm.io/

JSPM

- npm should work out of the box
- NodeJS libs for Browser are same as Browserify
- GitHub version is semvar
- package.json by default, or overridden by you
- flattens dependencies

```
jspm install jquery
```

JSPM

- improves package.js with new tags

```
jspm bundle app/main - react + moment build.js
```

```
<!doctype html>
  <script src="jspm_packages/system.js"></script>
  <script src="config.js"></script>
  <script src="build.js"></script>
  <script>
    System.import('app/main.js');
  </script>
```

JSPM

- standalone (production)

```
jspm bundle-sfx app/main.js app.js
```

CONCLUSIONS

- JSPM should be all you need
- Will probably be standard
- Runtime + Build time
- All 3 libraries built-on top of each other
- supports 3 languages and all module formats

JESSE WARDEN

- @jesterxl
- jesse@jessewarden.com
- https://www.youtube.com/user/jesterxl
- http://jessewarden.com/blog/