Large Scale JavaScript on Client and Server

Module 3: Scalable JavaScript

Shawn Wildermuth
Wilder Minds LLC
@shawnwildermuth

pluralsight
hardcore dev and IT training

Agenda

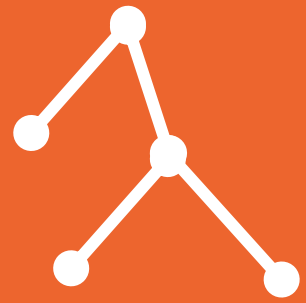
- **Scalable JavaScript**
  - What is Scalable JavaScript?
  - Improve
  - Optimize
  - Compose


803
RIETZ
x
sin
tan
arc

JavaScript Scalability Problems

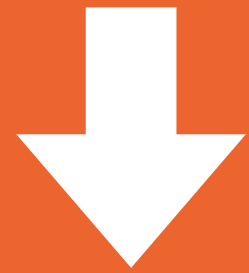
Runtime

Browser/Mobile Differences Matter



Parse

Every Line of Code is Expensive

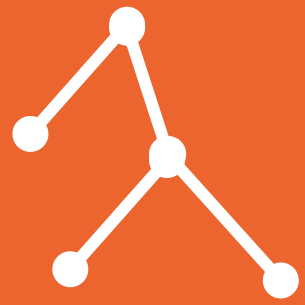


Download

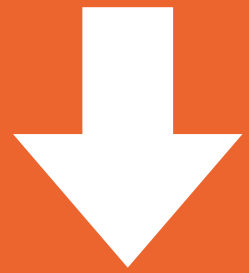
Empty Cache Still Common

JavaScript Scalability *Solutions*

Runtime



Parse



Download

Improve

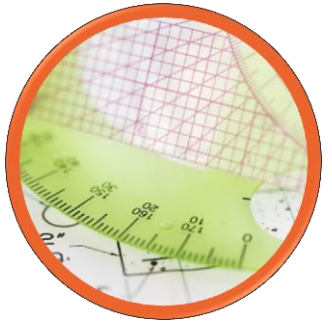
Optimize

Compose

IMPROVE

Write Less Code

App Architecture



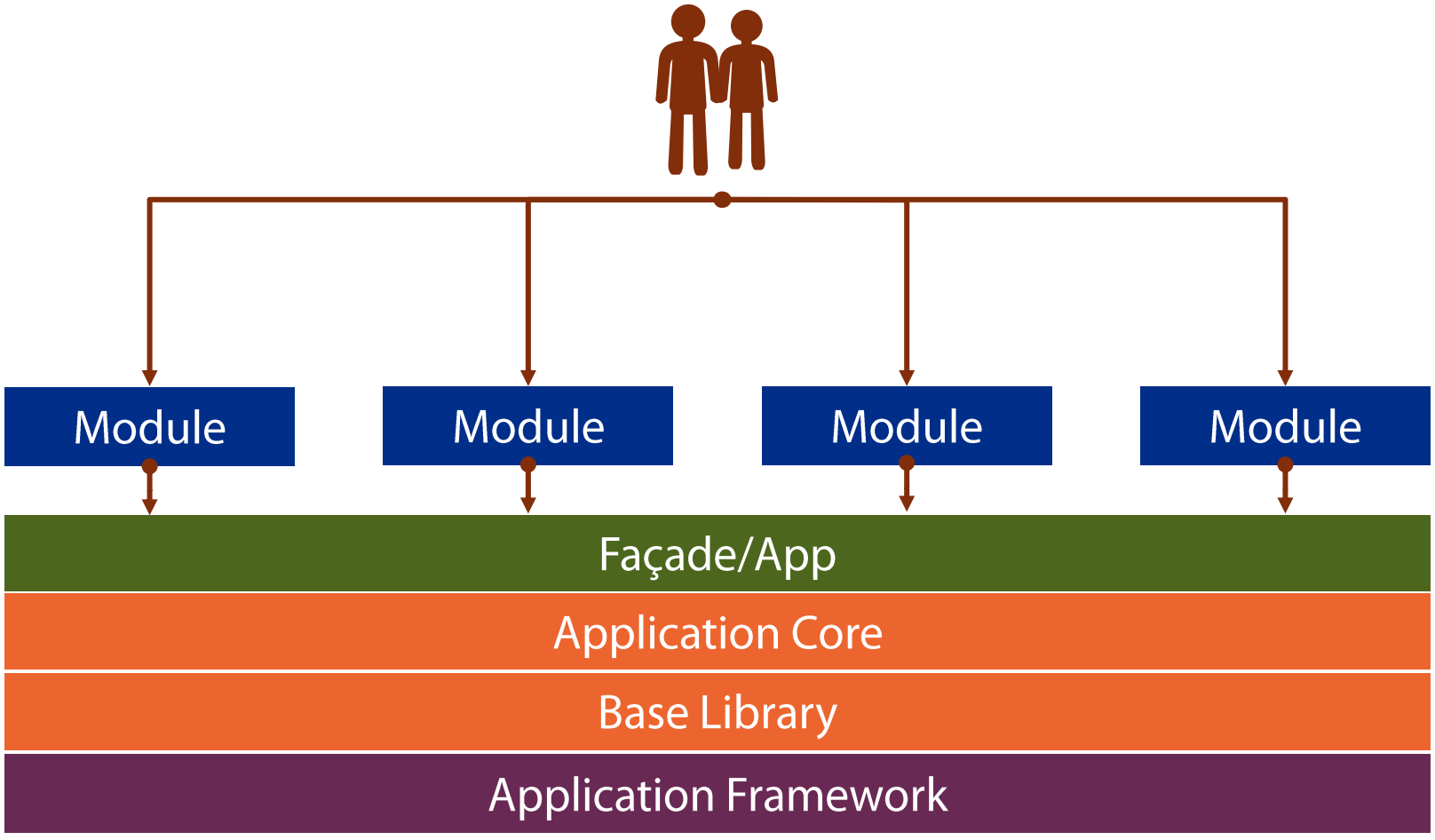
Smarter UI Coding

WRITING LESS CODE

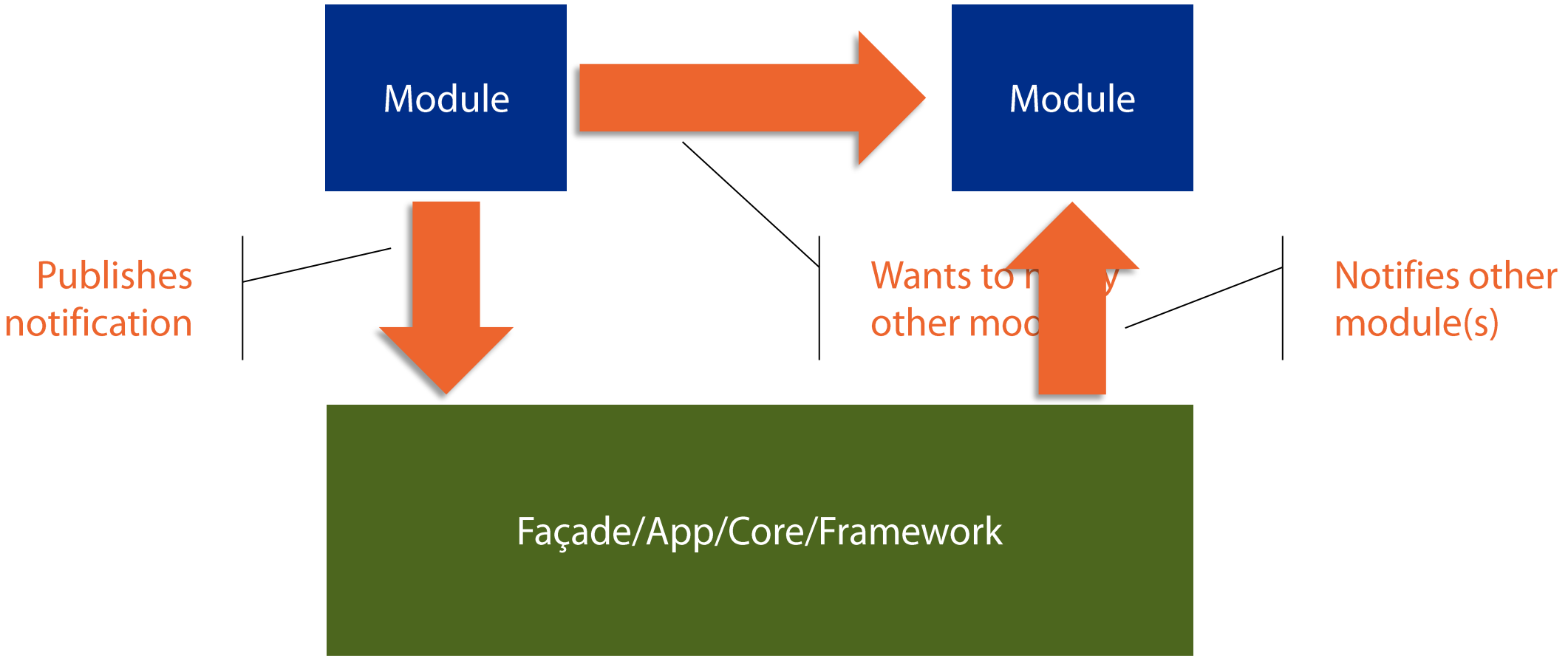
Code size impact performance

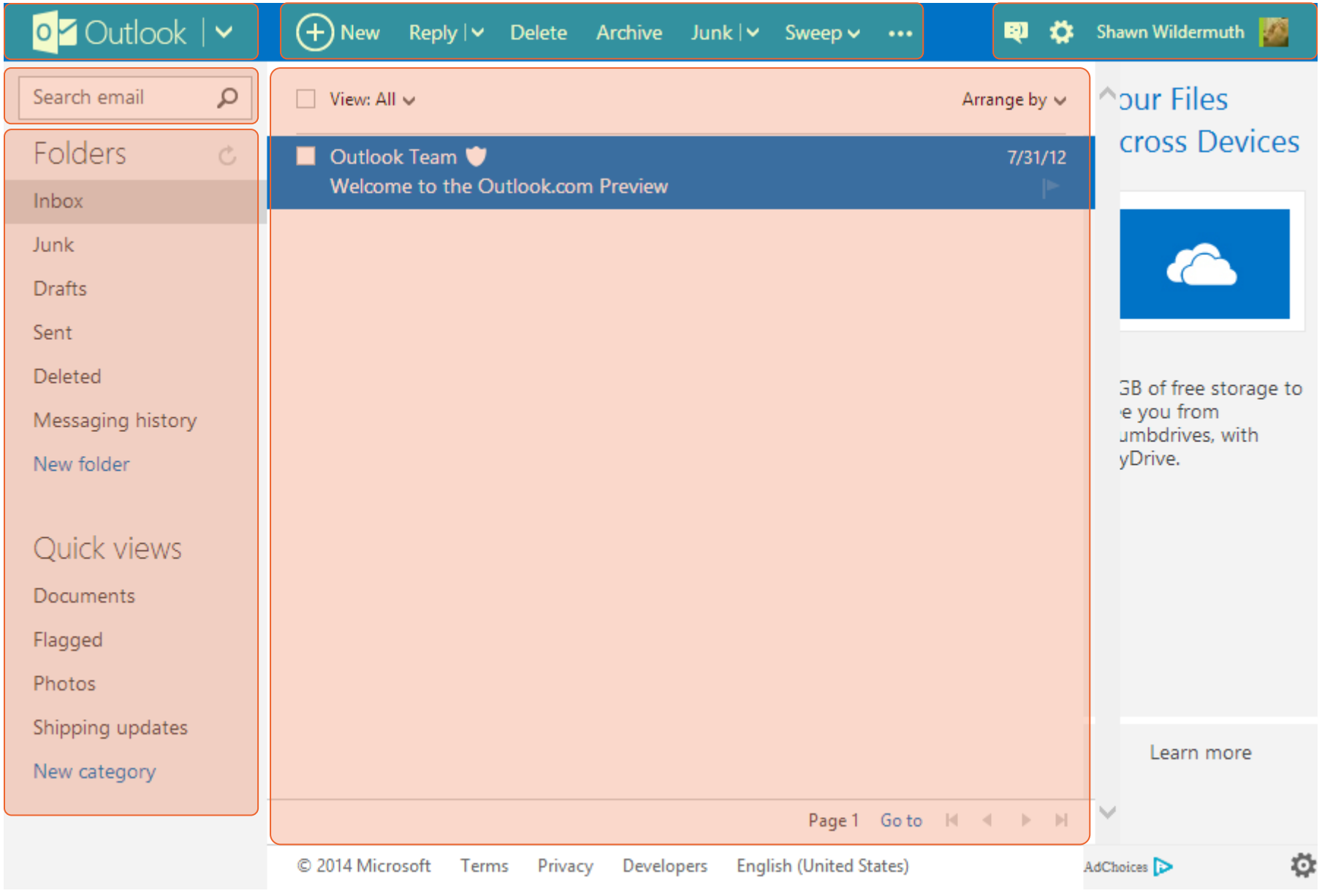
- Parsing is a bottleneck
    - Not linear with code size
    - JIT/Object Code not cached
- Running nothing is really quick
    - Deferring operations until necessary is important
    - The user will wait as they go; but not on page load

Smart Application Architecture



Smart Application Architecture




Outlook
New
Reply
Delete
Archive
Junk
Sweep
Shawn Wildermuth
Search email
Folders
Inbox
Junk
Drafts
Sent
Deleted
Messaging history
New folder
Quick views
Documents
Flagged
Photos
Shipping updates
New category
View: All
Arrange by
Outlook Team
7/31/12
Welcome to the Outlook.com Preview
Page 1
Go to
our Files
cross Devices
GB of free storage to
e you from
umbdrives, with
yDrive.
Learn more
© 2014 Microsoft
Terms
Privacy
Developers
English (United States)
AdChoices

SMARTER UI CODE

Be Lazy

- Avoid pre-drawing hidden UI wherever possible
- Cache drawn HTML but remove from DOM
    - Have to manage invalidating your own cache though
- Redrawing usually cheaper than partial updates
- Do DOM Manipulation off DOM
    - (remove, edit, insert)
- Use InnerHTML to avoid construction DOM elements

Optimize

- **Make your source code as small as possible**
  - Improves parse performance
- **Automate**
  - Create a ‘Build’ step
- **Removing Debug Code Sections**
  - Use conditional compilation

min-i-fy

/min-uh-fahy/

Verb

1. The process of removing all unnecessary characters from source code without changing its functionality.

MINIFICATION BASICS

Compresses JS (et al.) into smaller package

- Is commodity – doesn't matter which you choose
- May need to plan for it in your codebase
  - Relying on function parameter names is the problem
- Should automate
  - VS Web Essentials, GruntJS, etc.

Before Minification

```
// test.js
(function ($) {
  $(document).ready(function () {
    init("start here");
  });

  function init(msg) {
    $(".btn").text(msg);
  }

})(jQuery);
```

After Minification

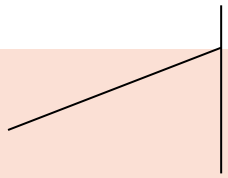
```
(function(n){function t(t){n(".btn").text(t)}n(document).ready(function(){t("start here")})})(jQuery);
```

After Minification

```
(function(n){
  function t(t){ n(".btn").text(t) }

  n(document).ready(function(){
    t("start here")
  })

})(jQuery);
```



Parameter, function, and variable names minified

MINIFICATION OPTIONS

If you're not already minifying, consider these options

- GruntJS – The JavaScript Task Runner
    - Contrib-Uglify – Plugin to do Minification
- Visual Studio + Web Essentials
- WebStorm + UglifyJS
- Sublime + Minify Package

GRUNTJS

Your JavaScript "Build" Step

Installing GruntJS

- **Install NodeJS (http://nodejs.org)**
  - It's just a runtime that GruntJS uses
- **Install GruntJS Globally**

```
> npm install –g grunt-cli
```

- **Install GruntJS Plugins locally**

```
> npm install grunt-contrib-jshint --save-dev
```

Using GruntJS

- **“grunt {Task}”**
  - Looks for gruntfile.js or gruntfile.coffee

```
// gruntfile.js
module.exports = function(grunt) {
  grunt.initConfig({
    // Add plugin configurations
    jshint: {
      all: ['public/js/*.js'],
      options: { multistr: true }
    },
  });
  // Load the plugins
  grunt.loadNpmTasks('grunt-contrib-jshint');
};
```

```
> grunt jshint
```

Using GruntJS

- **Can specify default task(s)**

```
// gruntfile.js
module.exports = function(grunt) {
  grunt.initConfig({
    // Add plugin configurations
    jshint: {
      all: ['public/js/*.js'],
      options: { multistr: true }
    },
  });
  // Load the plugins
  grunt.loadNpmTasks('grunt-contrib-jshint');

  grunt.registerTask('default', ['jshint']);
};
```

```
> grunt
```

Using GruntJS

- **UglifyJS plugin to GruntJS helps minification**

```
// gruntfile.js
module.exports = function(grunt) {
  grunt.initConfig({
    uglify: {
      build: {
        src: 'public/js/*.js',
        dest: 'public/js/build/all.min.js'
      }
    }
  });
};
```

Minifying is helpful, but removing code that is only used for debugging is also crucial in delivering only required JavaScript to the browser.

Conditional Compilation in JavaScript

- **Use GruntJS + UglifyJS to accomplish this:**

```
if (typeof DEBUG === undefined) DEBUG = true; // Force

DEBUG && console.log("some info");
if (DEBUG) {
  console.log("other info");
}

function foo() {
  console.log("initial");
}
foo();
```

```
function foo() {
    console.log("initial");
}
foo();
```

Conditional Compilation in JavaScript

- **Global Define and pruning dead code solves this:**

```
// gruntfile.js
...
uglify: {
  options: {
    compress: {
      global_defs: {
        DEBUG: false
      },
      dead_code: true
    }
  },
...
```


40
39
38
37

COMPOSE

Construct your JavaScript as necessary, instead of one code-base

- **Concatenate and Minify into logical code units**
- **Use JavaScript Loaders to do the heavy lifting**
- **May be combined with Dependency Injection**

Minify Into Packages

- Same GruntJS + UglifyJS solves this easily

```
// gruntfile.js
...
grunt.initConfig({
  uglify: {...},
    build: {
      files: {
        'build/main.min.js': ["js/main.js"],
        'build/base.min.js': ["js/destinations.js", "js/newtrip.js"]
      }
    }
  }
});
...
```

JAVASCRIPT LOADERS

A way to incrementally load JavaScript into the browser. Helps scale the size and timing of large scale web applications.

LazyLoadJS

- **Allows lazy loaded JavaScript:**

```
// main.js
// include lazyload.js
$("#loadButton").on("click", function () {

  LazyLoad.js(["js/build/trips.min.js"], function() {
    // Do Work
  });

});
```

Works well with Frameworks that already handle dependencies (e.g. AngularJS)

RequireJS

- **Mixes script loading and dependency management:**

```
// index.html
...
<script src="js/vendor/require.min.js"
        data-main="js/main"></script>
```

```
// main.js (or .min.js)
require(["destModule"], function(destModule) {
  // use destModule as necessary
});
```

Loads scripts when first required

```
// destModule.js
define([], function() { // module pattern
  ...
  return {
    cache: _cache,
  };
});
```

Dependencies and scripts are chained

RequireJS

- **Can be used to load as needed**
  - Still requires that you use RequireJS Modules:

```
// main.js
$("#loadButton").on("click", function () {

  require(["destModule"], function(destModule) {
    // use destModule as necessary
  });

});
```


ADAMS
20

References

- **Joseph Smarr's High Performance JavaScript Slides**
  - http://www.slideshare.net/briandemant/smarr-oscon-2007
- **GruntJS**
  - http://gruntjs.com
- **Web Essentials**
  - http://bitly.com/vswebessentials
- **LazyLoadJS**
  - https://github.com/rgrove/lazyload/
- **RequireJS**
  - http://requirejs.org

Summary

- **Scalable JavaScript**
  - Knowing you have a problem is often the first task of scaling your JavaScript
  - Focusing on better code will make your application scale better
  - Being lazy about your execution is often a better approach
  - Minification will help you improve performance of downloads and parsing
  - Only loading the JavaScript required to do the job is also crucial
  - Late loading JavaScript as it is needed is another useful technique