Large Scale JavaScript on Client and Server

Module 5: Scalable JavaScript in NodeJS

Shawn Wildermuth
Wilder Minds LLC
@shawnwildermuth

pluralsight
hardcore dev and IT training

Agenda

- **Scalable JavaScript in NodeJS**
  - What is NodeJS
  - Is server-side JavaScript different?
  - Maintainable JavaScript in NodeJS
  - Scalable JavaScript in NodeJS
  - Testable JavaScript in NodeJS


PoE-Integrated 10/100/1000Base-T Ports (1 - 24) -- Ports are IEEE Auto MDI/MDI-X
40110
40112
40114
40118
40120
40122
40126
40128
40130
40111
40115
40117
40121
50226

WHAT IS NODEJS?

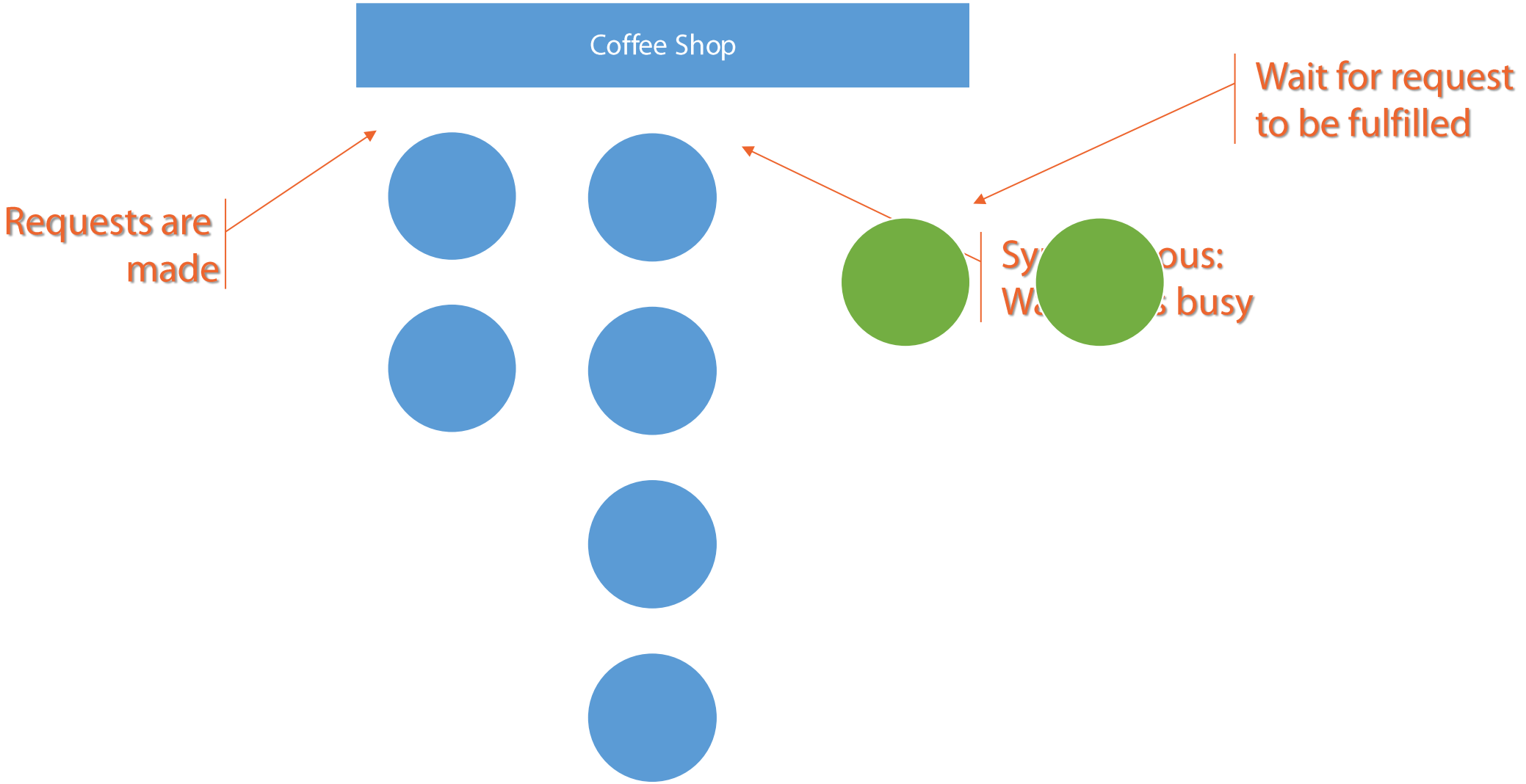
Platform for executing JavaScript based on non-blocking I/O, event-driven model that enables fast, scalable network applications.

NODEJS

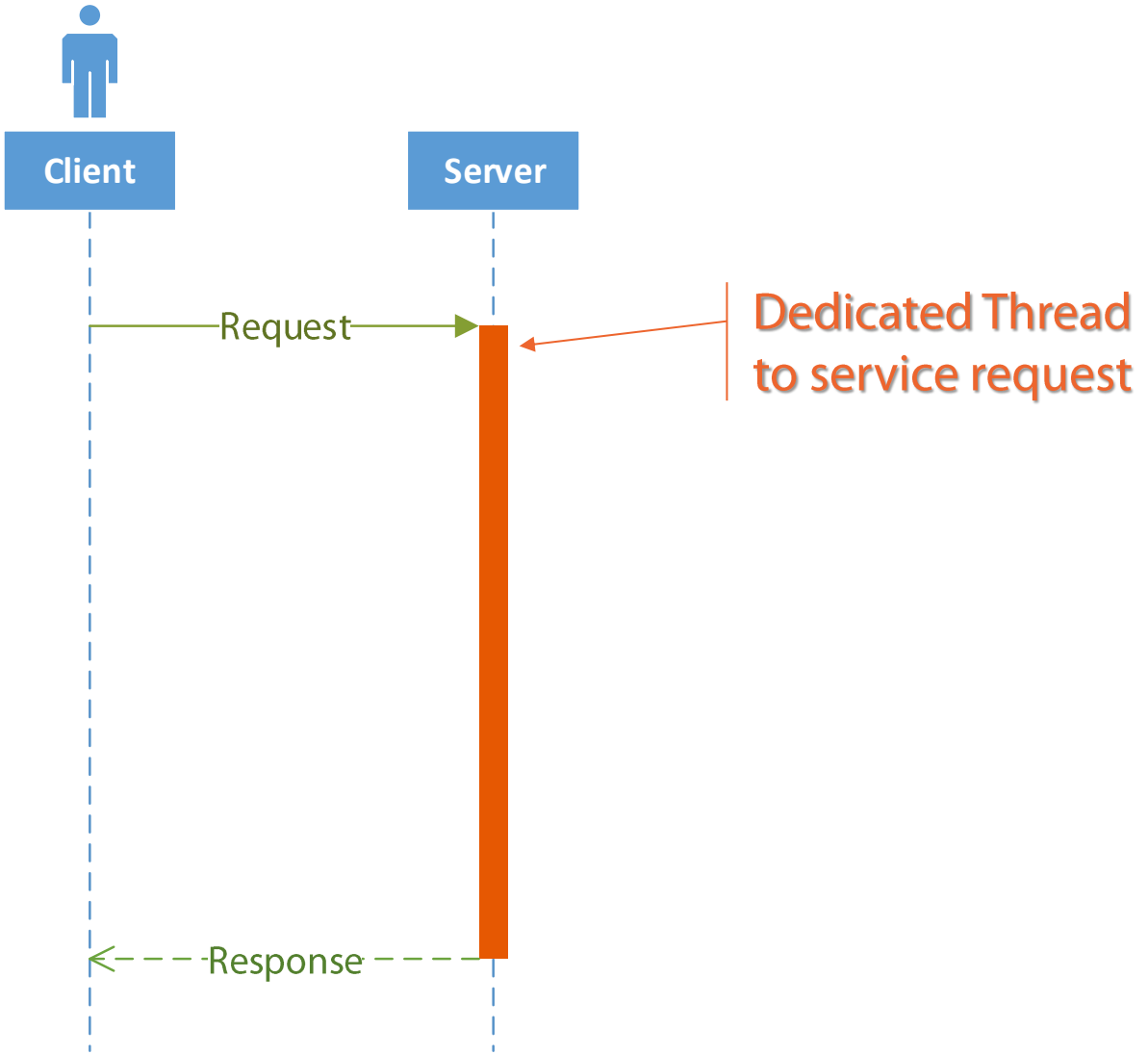
Techniques for Maintainable JavaScript Work with Most Frameworks

- **Supports Modular JavaScript**
- **Encourages non-blocking (asynchronous) code**
- **Networking and Web Sockets are 1st class citizens**
- **NodeJS is a low-level, fast platform**
  - **But wasn't written in JavaScript**

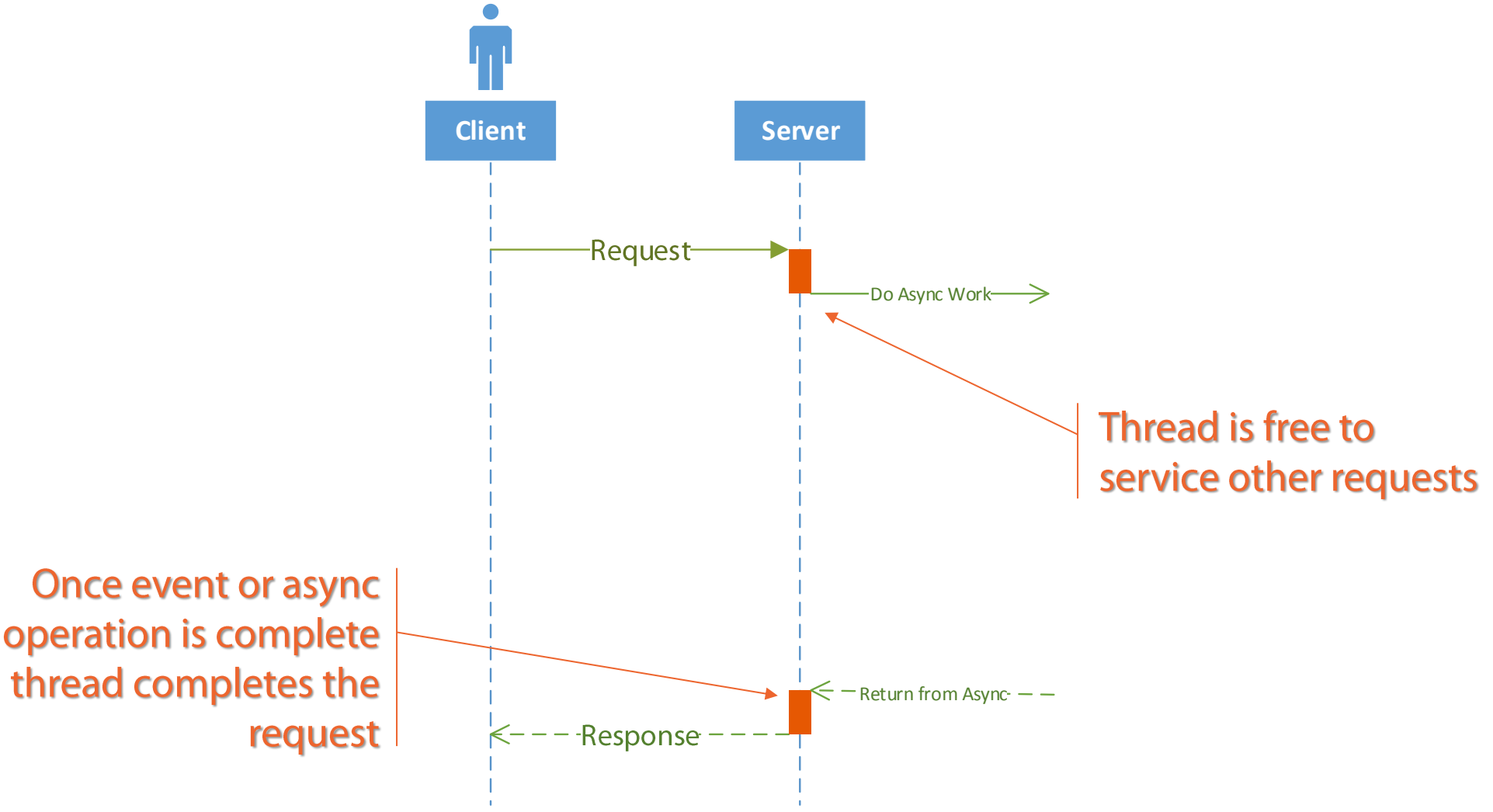
NodeJS and Traditional Web Servers



Traditional Web Server Model



Async Web Server Model



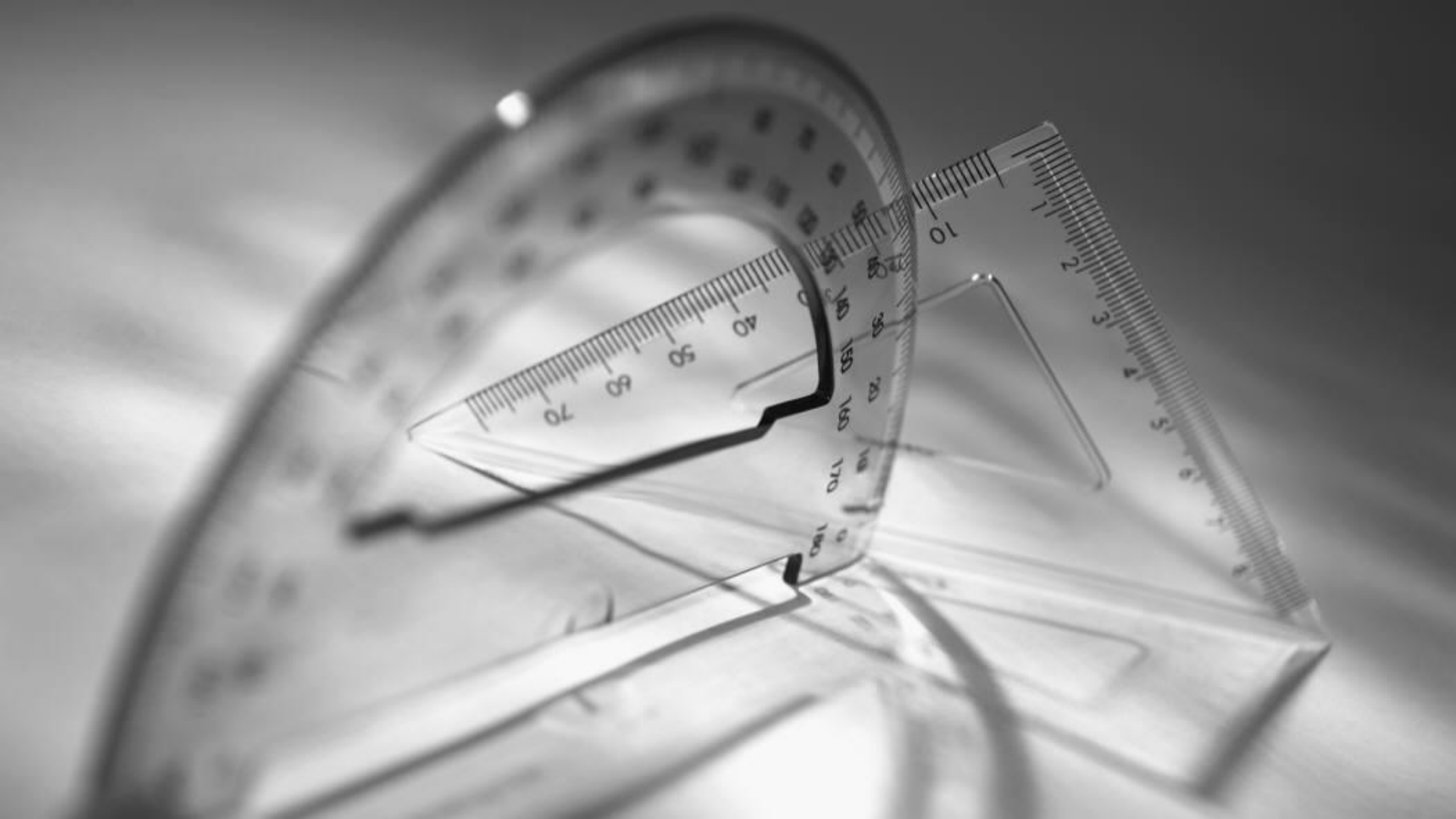
IS JAVASCRIPT DIFFERENT INSIDE OF NODEJS?

JAVASCRIPT IN NODEJS

All JavaScript is executed under Google's V8 Engine

- Predictable execution model
- EcmaScript 5 is supported in NodeJS 0.5.1 and later
- V8 supports JIT and Optimizing Compiler

Lots of JavaScript problems go away in NodeJS because you have complete control over the environment.



Why is NodeJS Considered More Scalable?

- **NodeJS style encourages less blocking**
  - While in Java/.NET you have to understand multithreading to accomplish same
  - Non-blocking is the default behavior
  - Supports scale-out
    - NodeJS is single threaded by default
  - Thread is available to answer client requests while you're waiting for I/O

Scaling NodeJS Isn't About JavaScript

Scaling Your Datastore (Memcache et al.)	Geographic Scaling
Load Balancing	Scaling Out (Multi-Instance and Multi-Box)

NodeJS Doesn't Do Any of These Out of the Box

VERTICAL SCALE WORKS

Giving NodeJS more resources (CPU, GPU, memory, disk, network)

- NodeJS is single threaded but can run multiple instances
- No support for GPU use (but coming)
- Fast memory, disk and network just work

HORIZONTAL SCALE IS HARDER

Scaling to multiple servers

- Currently no automatic support
- Can build it with existing networking code, but not trivial
- Let partners do it instead (e.g. AWS, Azure, Heroku)

Large Scale JavaScript in NodeJS?

Maintainable	Scalable	Testable
Modularized	Compose-able	Encapsulate
Separation of Concerns	Loosely Coupled	Test Facades

Maintainable JavaScript is no different from other JavaScript. Modularity and dependency management is important. CommonJS makes this possible in NodeJS.

CommonJS in NodeJS

```
var data = require("./data");

var bar = data.foo;
```

Package, directory or file based dependency management

```
// data.js
exports.foo = {
};
```

Simple expansion of exports to expose dependencies

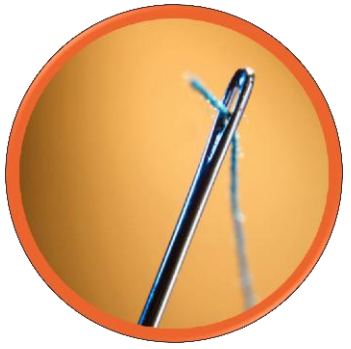
```
// data.js
(function(Data) {
  "use strict";

  Data.foo = {};

})(exports);
```

Combine SEAF, strictness, and aliasing export to simplify creating modules

SCALABLE JAVASCRIPT IN NODEJS



ASYNCHRONY

LESS CODE

OPTIMIZE FOR V8

Callbacks in NodeJS

Many NodeJS libraries still use old-style callbacks

```
var = require("fs");

fs.readFile("./data/cities.json", function(err, data) {
  if (err) {
    handleError(err);
  } else {
    theCities = JSON.parse(data);
  }
});
```

Promises in NodeJS

```
> npm install q
```

```
var q = require('q');

var defer = q.defer();

fs.readFile("./data/cities.json", function(err, data) {
  if (err) {
    defer.reject(err);
  } else {
    theCities = JSON.parse(data);
    defer.resolve(theCities);
  }
});

return defer.promise;
```

Same as client-side promise pattern

Async in NodeJS

```
> npm install async
```

```
// Supports parallel execution of set operations
async.each(coll, function(item) {},
  function(err, result) { // callback

  });
```

Async in NodeJS

```
// Supports execution of callback style functions
async.series([  // Call serially
    function() {},
    function() {}
  ],
  function(err, results) { // callback
  });

async.parallel([ // Call in parallel
    function() {},
    function() {}
  ],
  function(err, results) { // callback
  });
```

Node Package Manager

NPM makes including libraries into node simple, but it is your job to remove unneeded libraries. Fewer needed libraries means that NodeJS will be more efficient.

MINIFICATION OF JAVASCRIPT FOR NODEJS IS NOT NECESSARY

TIME SAVED IS IN DOWNLOADING SCRIPT, PARSE WILL BARELY BE FASTER FROM MINIFIED CODE

Improving Performance of JavaScript in V8

Polymorphic

Avoid it - Can negate benefit of caching

Arrays

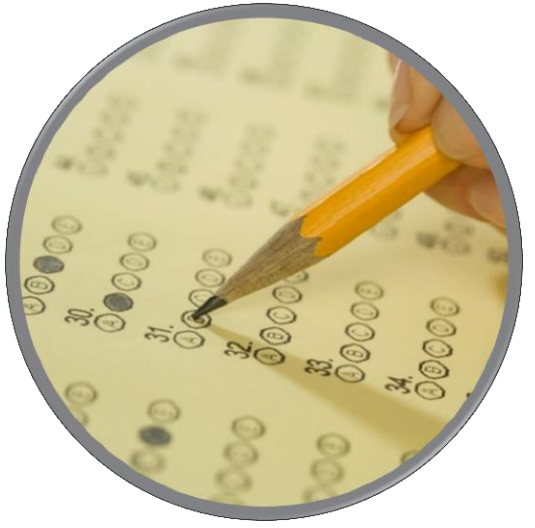
Grow arrays
Don't preallocate

Compilation

Inlining is at function level

UNIT TESTING JAVASCRIPT IN NODEJS

Same testing frameworks – different runner



Jasmine for Server Code

Suite of Tests

Spec

Expectation

```
describe("routes", function() {
  var routes = require("./routes");
  var express = require("express");
  it("has routes", function() {
    var app = express();
    routes.init(app);
    expect(app.routes).toBeDefined();
    expect(app.routes.get).toBeDefined();
    expect(app.routes.get.length).toBeGreaterThan(0);
    expect(app.routes.post.length).toBeGreaterThan(0);
  });
});
```

Using GruntJS NodeJS Unit Testing

- ***grunt-contrib-jasmine*** **doesn't support testing of NodeJS code**
  - Need to use ***grunt-jasmine-node*** instead
  - Can still batch all tests together as single task

grunt-jasmine-node

```
> npm install grunt-jasmine-node --save-dev
```

```
jasmine_node: {
  matchall: true,
  projectRoot: "./src",
  requirejs: false,
  forceExit: true,
  jUnit: {
    report: false,
    savePath: "./build/reports/jasmine/",
    useDotNotation: true,
    consolidate: true
  }
},
```

grunt-jasmine-node

```
// Create a task that runs all unit tests
grunt.registerTask('runtests',
  ['jasmine', 'jasmine_node']);
```

```
> grunt runtests
```

```
> grunt watch:runtests
```

Summary

- **Scalable JavaScript in NodeJS**
  - It is a great solution for server-side development when you have JavaScript skills
  - But it isn’t magically suited to large scale projects.
  - Still needs to be Maintainable, Scalable and Testable.
  - Modularization and CommonJS are keys to maintainable JavaScript on the server
  - For scalable JavaScript, understand your runtime environment
  - You still need to test your code, even if it is just on the server