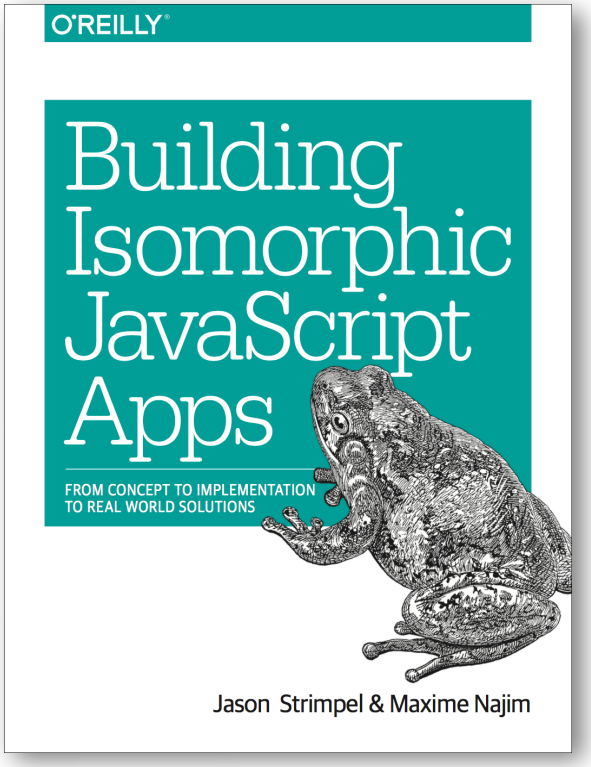
ISOMORPHIC JAVASCRIPT WITH NASHORN

Maxime Najim

Monday, October 26, 2015

About Me



Final Release Date: April 2016

```
var me = {
    name: "Maxime Najim",
    title: "Software Architect",
    work: "@WalmartLabs",
    org: "@Platform"
    twitter: "@softwarecrafts"
}
```

Why am I talking about **JavaScript** at **JavaOne**?

"**Java** is to **JavaScript** as ham is to hamster"

Jeremy Keith

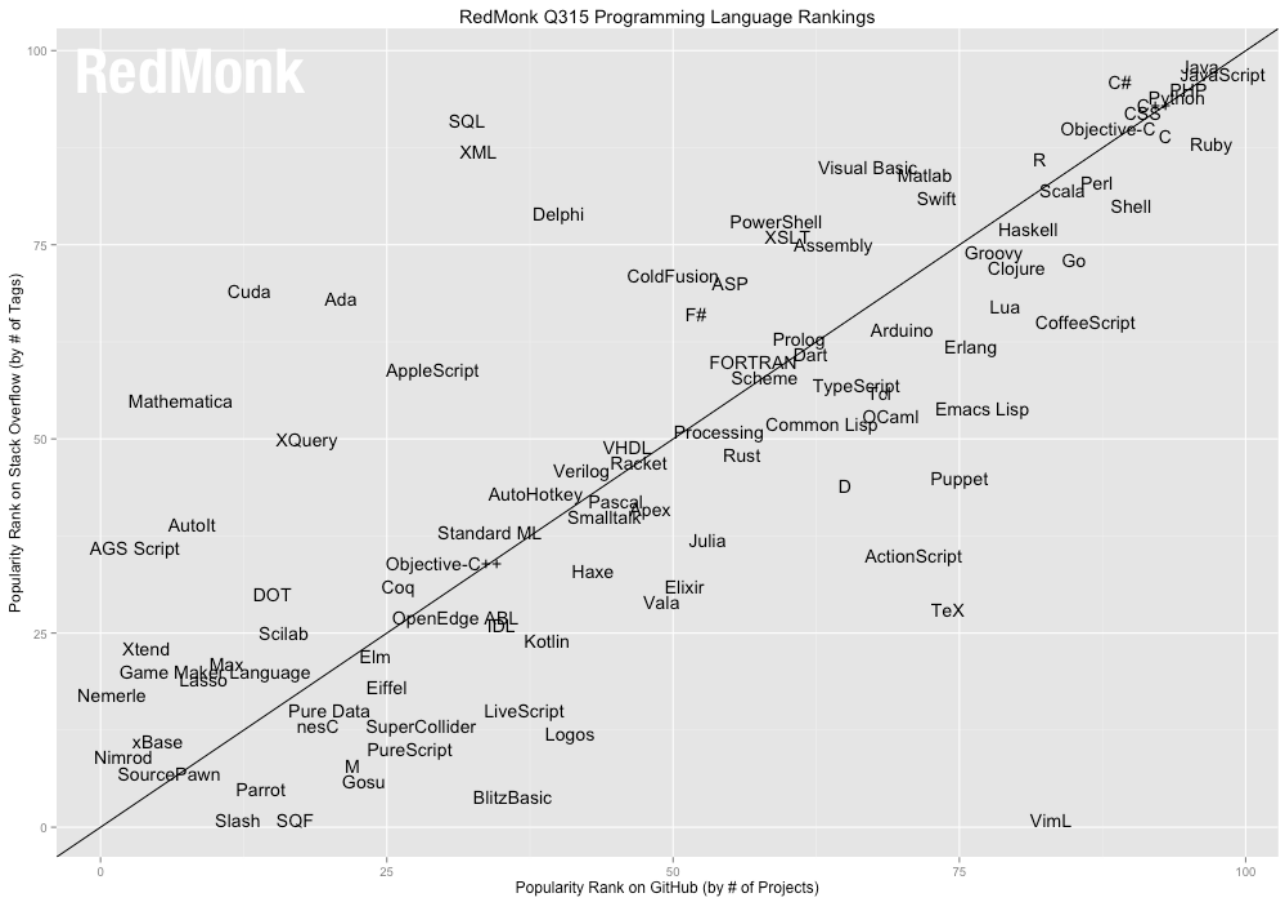
http://javascriptisnotjava.io

Three reasons why Java developers should be talking about **JavaScript**...

Reason 1:

Developers from different backgrounds are converging on **JavaScript**

RedMonK Programming Language Rankings



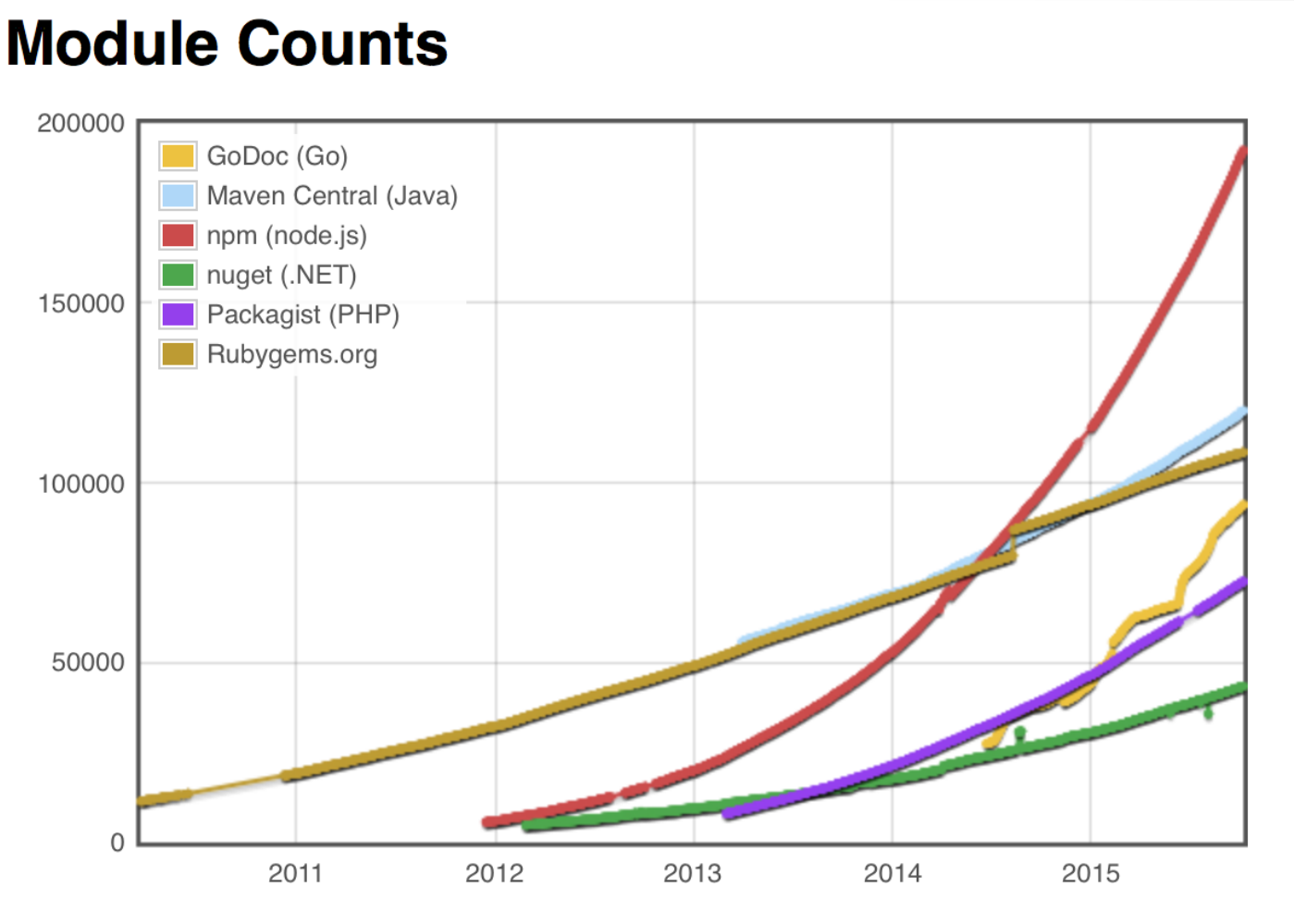
Top 20

1 **JavaScript**
2 **Java**
3 PHP
4 Python
5 C#
5 C++
5 Ruby
8 CSS
9 C
10 Objective-C
11 Perl
11 Shell
13 R
14 Scala
15 Go
15 Haskell
17 Matlab
18 Swift
…

JavaScript's standard package repository is the fasting growing and most active package public repository.

* More people are actively working on JavaScript projects

* More likely to find open-source solutions



source: modulecounts.com

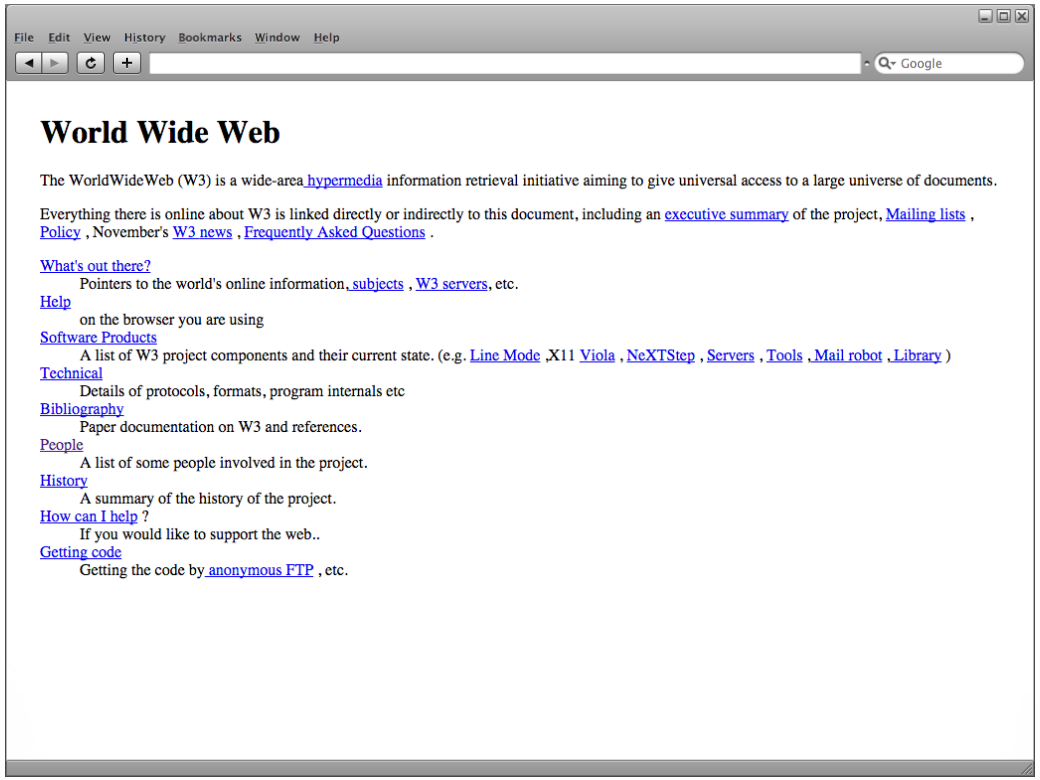
Reason 2:
JavaScript is the **platform** for building rich and highly interactive web apps

In the past **decade**, we've seen the **Web** evolve…

The **Web** is no longer simply documents **linked** together

Web Evolution

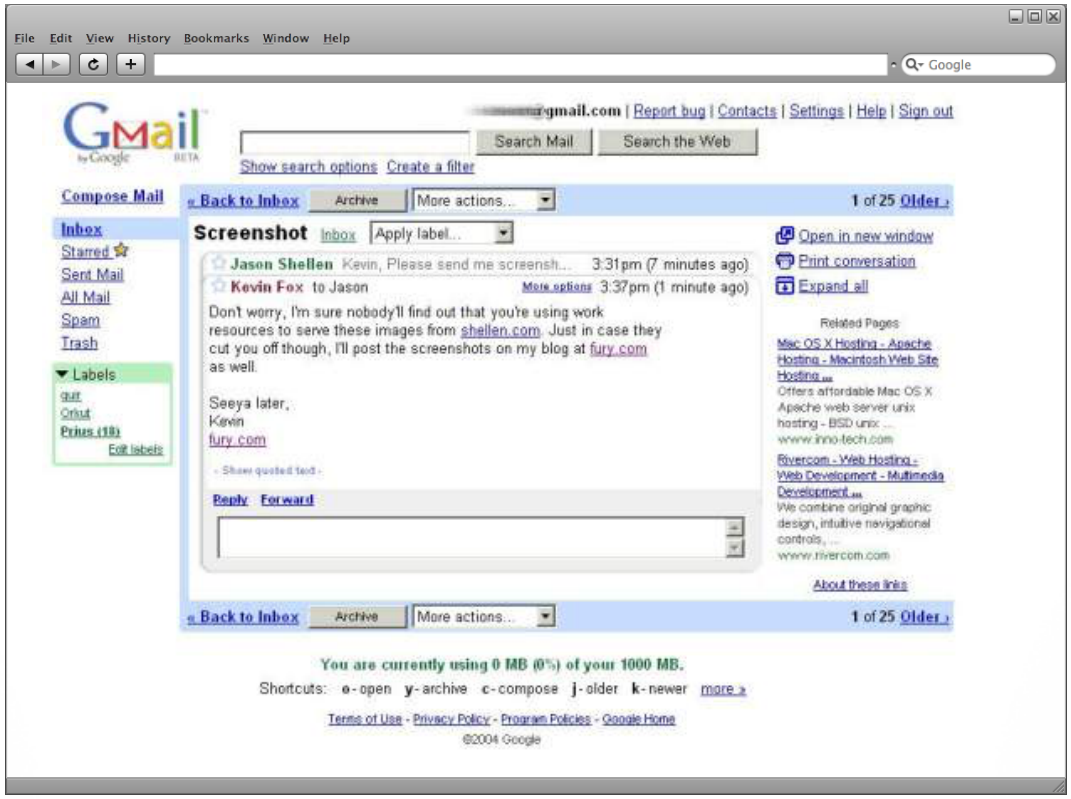
1990's - Initial Web Era



The world's first web page: http://info.cern.ch/hypertext/WWW/TheProject.html

Web Evolution

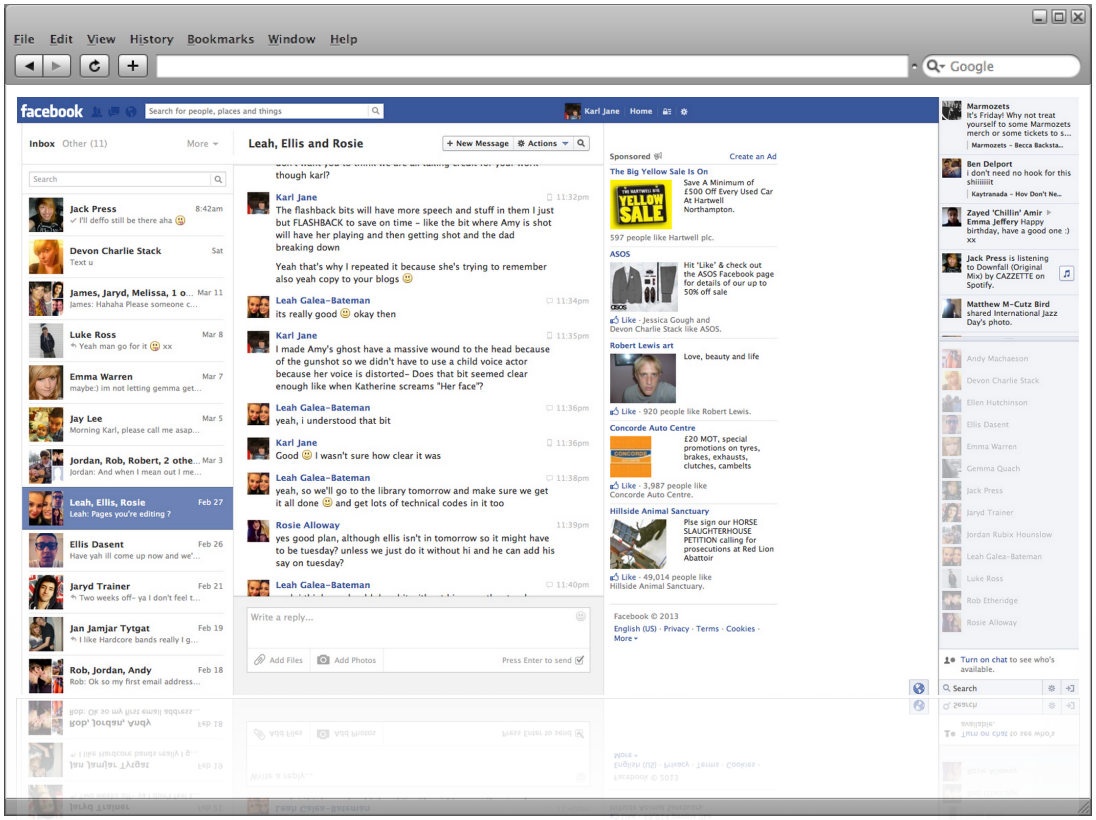
2000's - AJAX Web Era



Gmail (2004)

Web Evolution

2010's - Single Page App Web Era



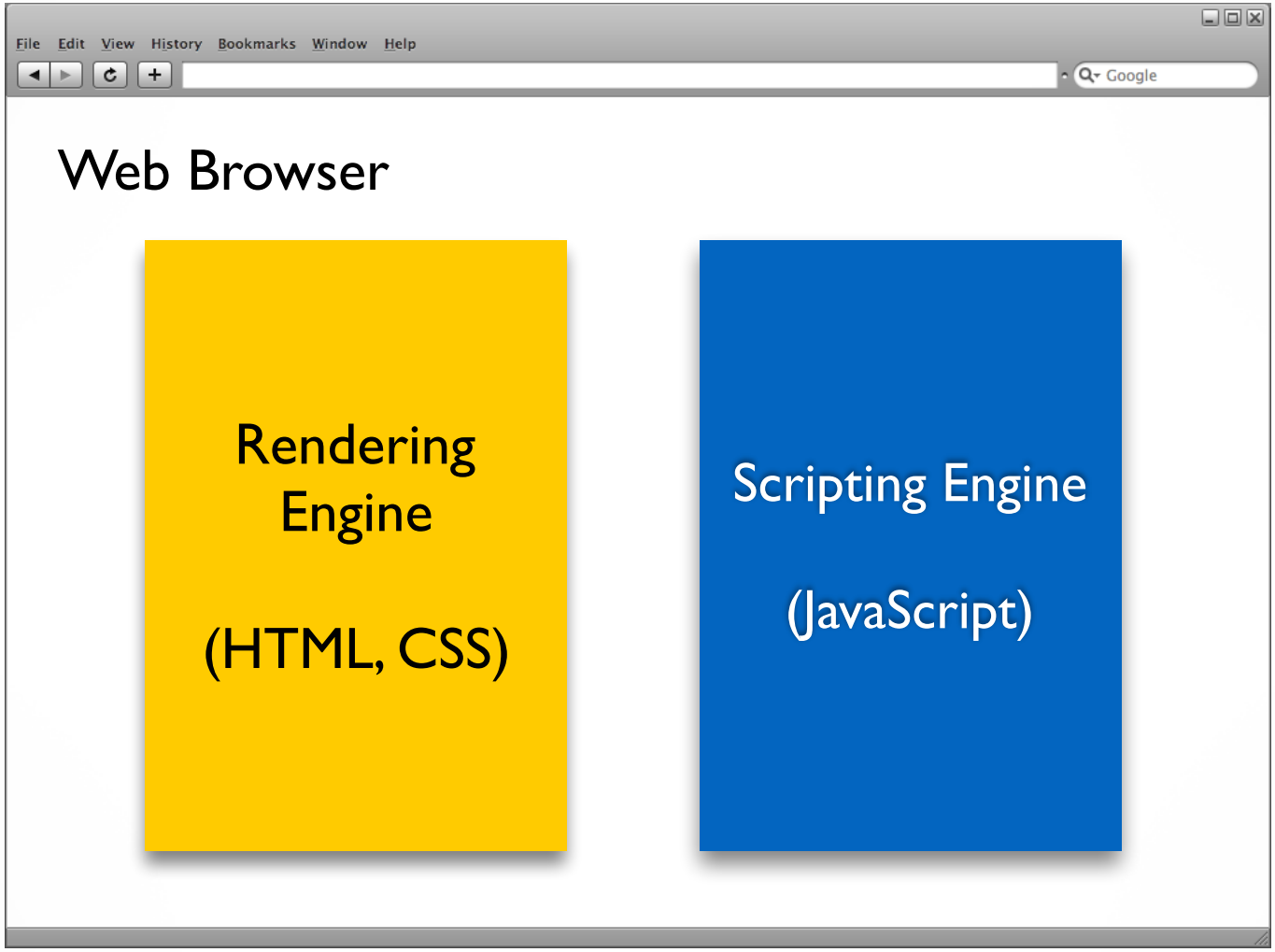
JavaScript has enabled Web ***sites*** to evolve to web ***apps***

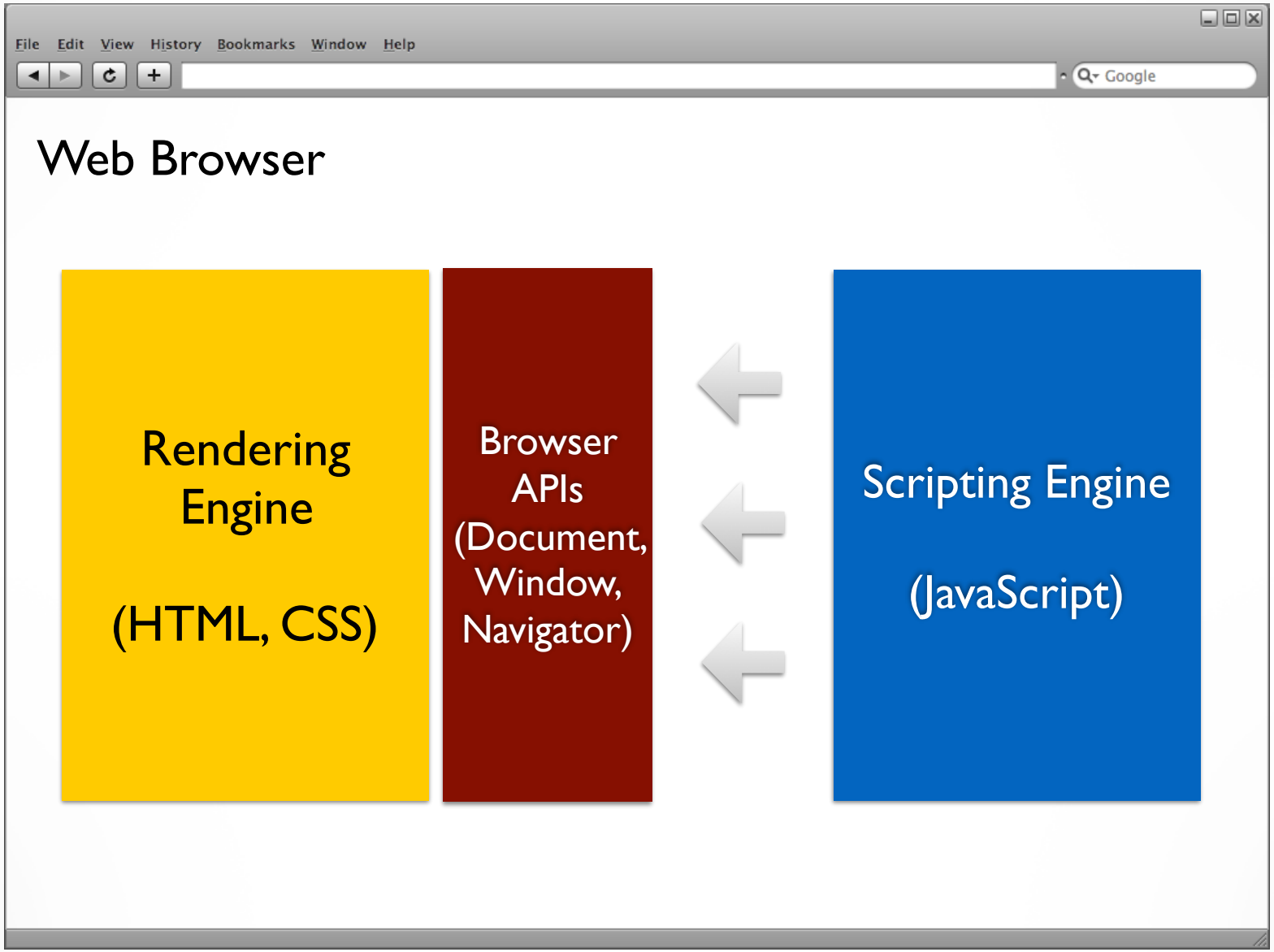
JavaScript in the **browser** has become our app runtime environment

Reason 3:
JavaScript isn't **only** for the **browser**

When people think of **JavaScript** they think of **browser provided APIs**

(e.g. window, document, etc.)


File Edit View History Bookmarks Window Help
Google
Web Browser
Rendering Engine
(HTML, CSS)
Scripting Engine
(JavaScript)


File Edit View History Bookmarks Window Help
Google
Web Browser
Rendering Engine
(HTML, CSS)
Browser APIs (Document, Window, Navigator)
Scripting Engine
(JavaScript)

Firefox

Gecko

(HTML, CSS)

Spider Monkey

(JavaScript)

IE

Trident

(HTML, CSS)

Chakra

(JavaScript)

Chrome

Blink

(HTML, CSS)

V8

(JavaScript)

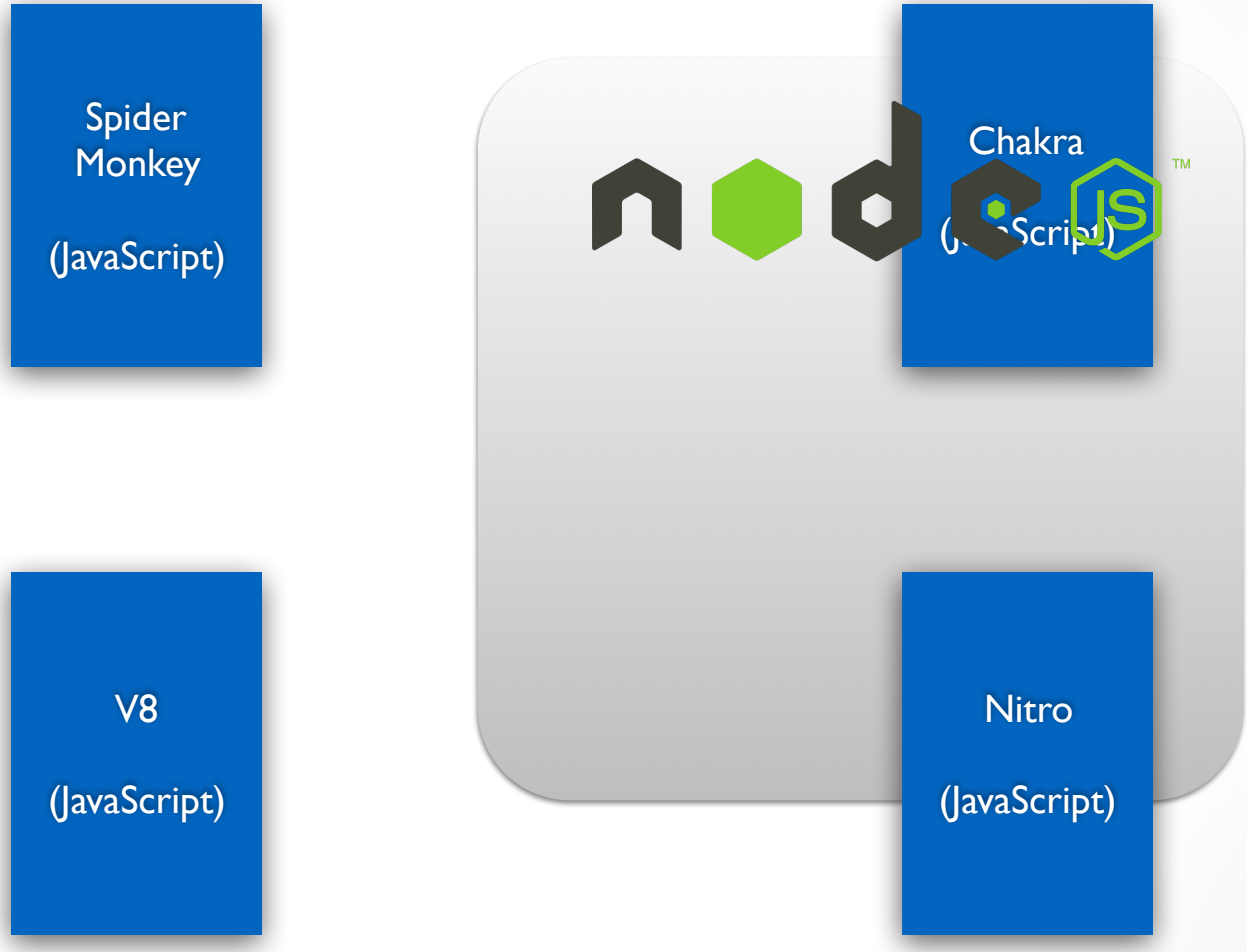
Safari

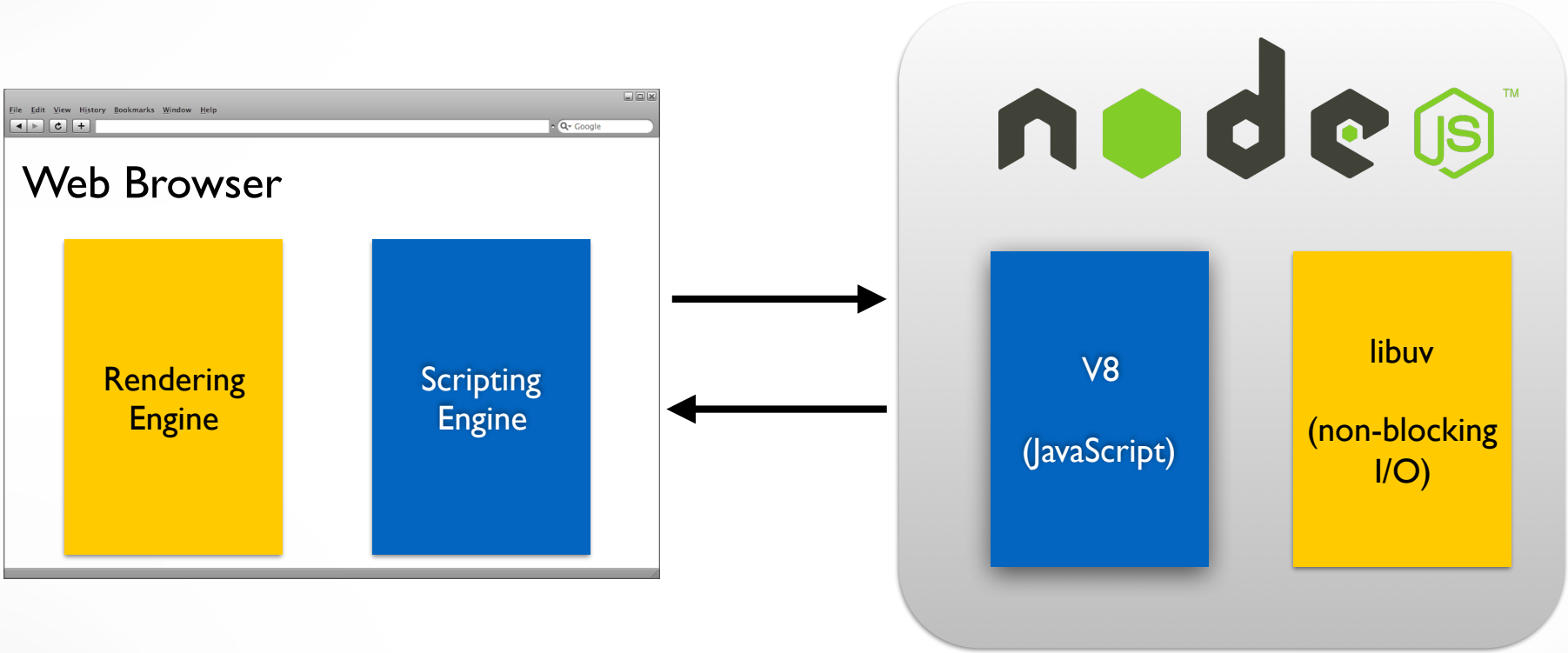
WebKit

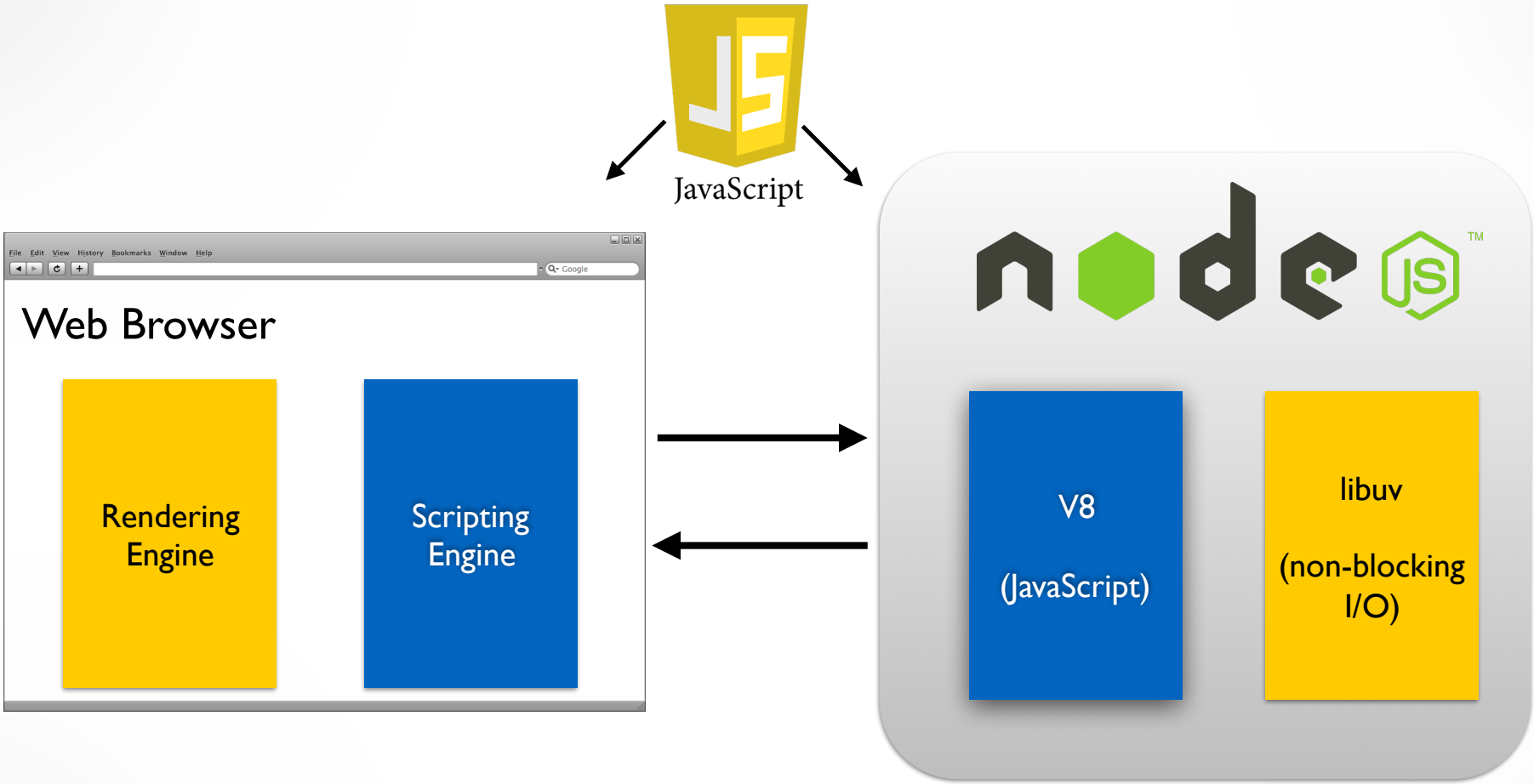
(HTML, CSS)

Nitro

(JavaScript)


Spider
Monkey
(JavaScript)
Chakra
(JavaScript)
V8
(JavaScript)
Nitro
(JavaScript)


File Edit View History Bookmarks Window Help
Google
Web Browser
Rendering Engine
Scripting Engine
node
JS
TM
V8
(JavaScript)
libuv
(non-blocking I/O)


JavaScript
Web Browser
Rendering Engine
Scripting Engine
V8
(JavaScript)
libuv
(non-blocking I/O)

Isomorphic JavaScript

a.k.a Universal JavaScript, Portable JavaScript, Shared JavaScript

Isomorphic JavaScript

JavaScript code that runs both on the backend web application server and the client.

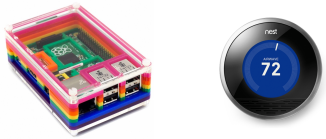
Server

Client

Web

Mobile

IoT



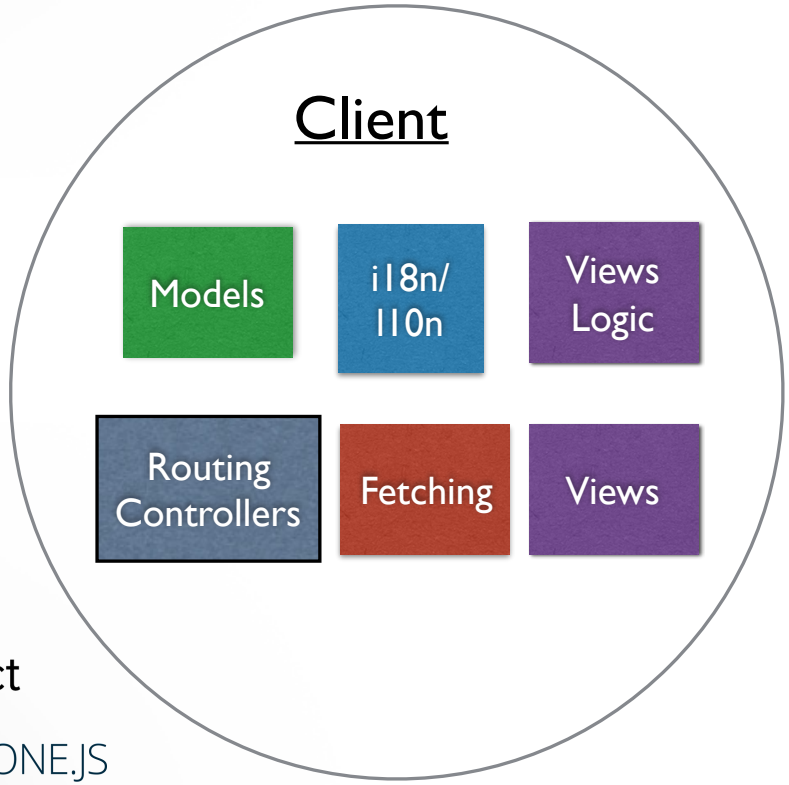
Isomorphic JavaScript

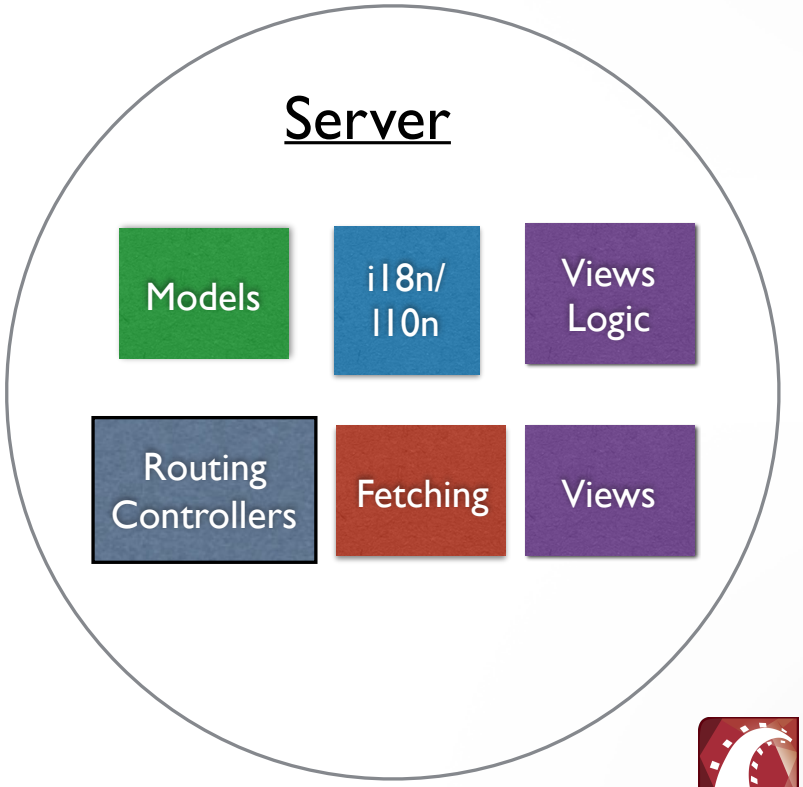
1. Staying DRY (Don’t-Repeat-Yourself) - using the same code base improves code maintenance.

2. Server Side Rendering of Single Page Applications (very critical to the business)

1) Staying DRY with **Isomorphic JavaScript**

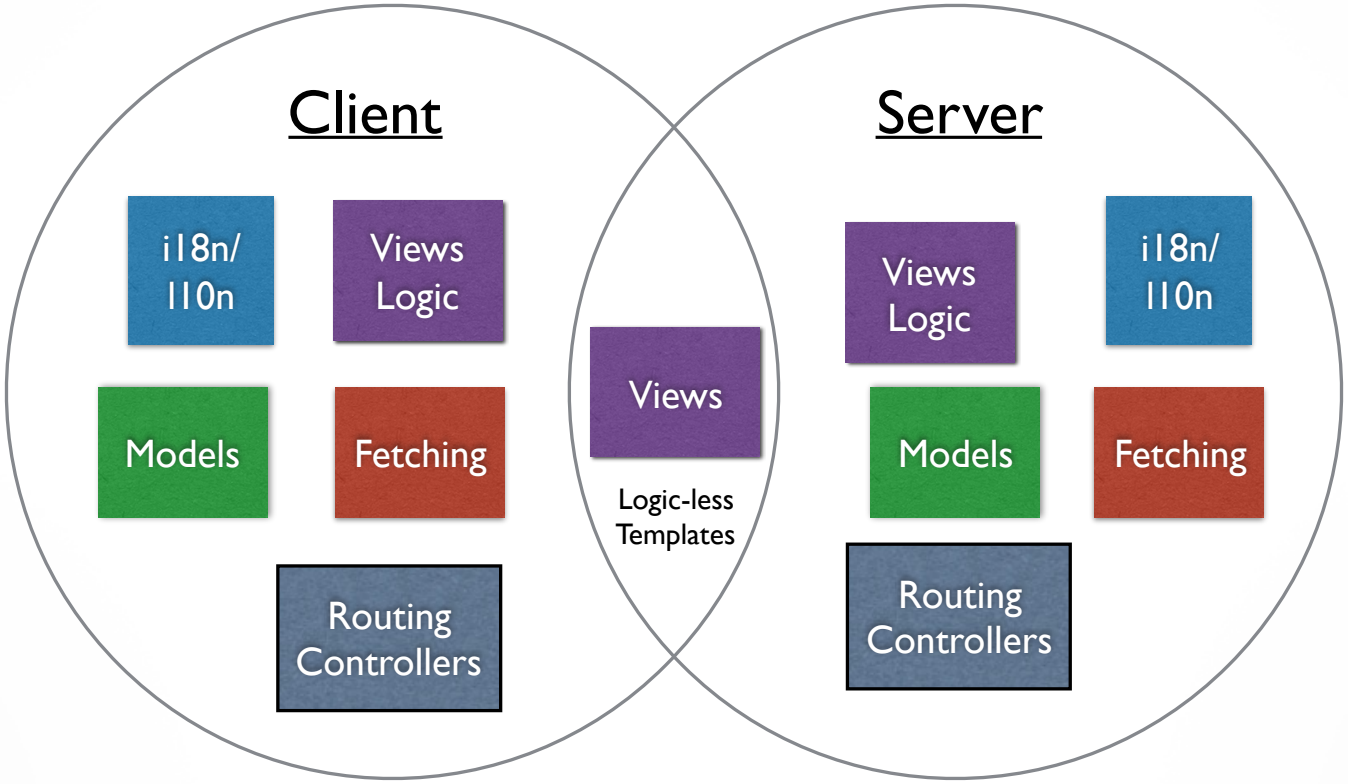
Staying DRY



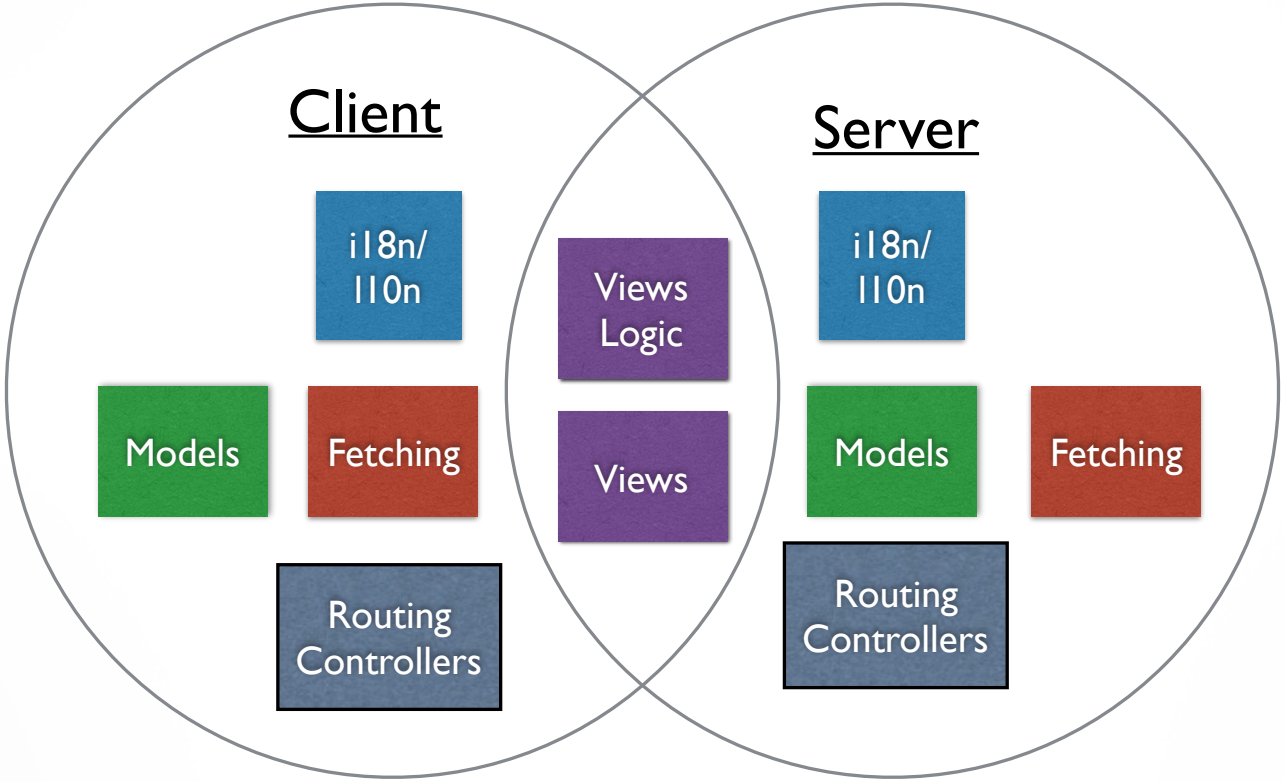


React

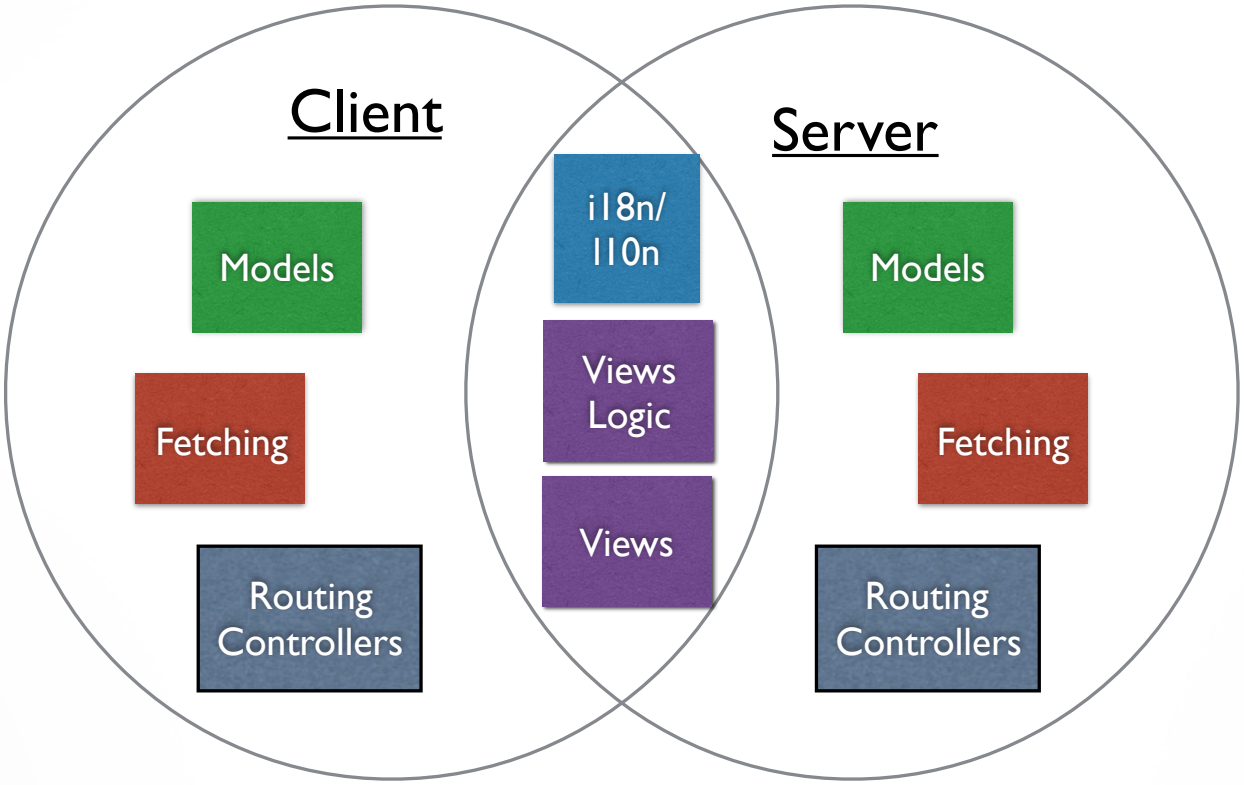
Staying DRY


Client
i18n/
l10n
Views
Logic
Models
Fetching
Routing
Controllers
Views
Logic-less
Templates
Server
Views
Logic
i18n/
l10n
Models
Fetching
Routing
Controllers

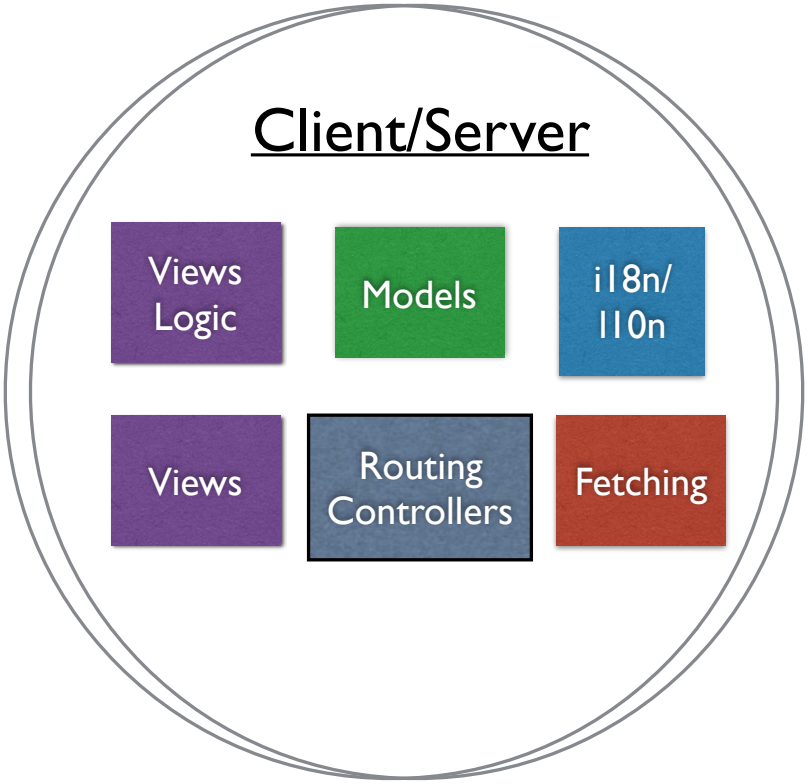
Staying DRY


Client
Server
i18n/
l10n
Views
Logic
i18n/
l10n
Models
Fetching
Views
Models
Fetching
Routing
Controllers
Routing
Controllers

Staying DRY


Client
Server
Models
Fetching
Routing Controllers
i18n/ l10n
Views Logic
Views
Models
Fetching
Routing Controllers

Staying DRY

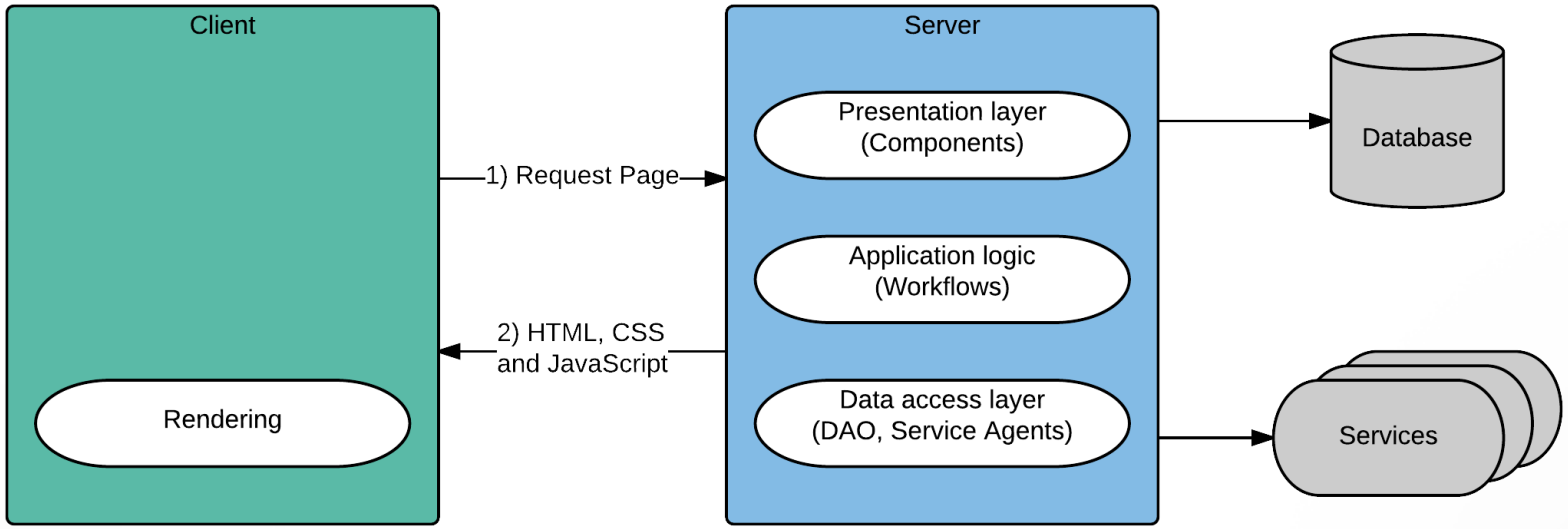

Client/Server
Views Logic
Models
i18n/ l10n
Views
Routing Controllers
Fetching

2) Server Side Rendering of **Single Page Applications**

Web Evolution

1990's - Initial Web Era

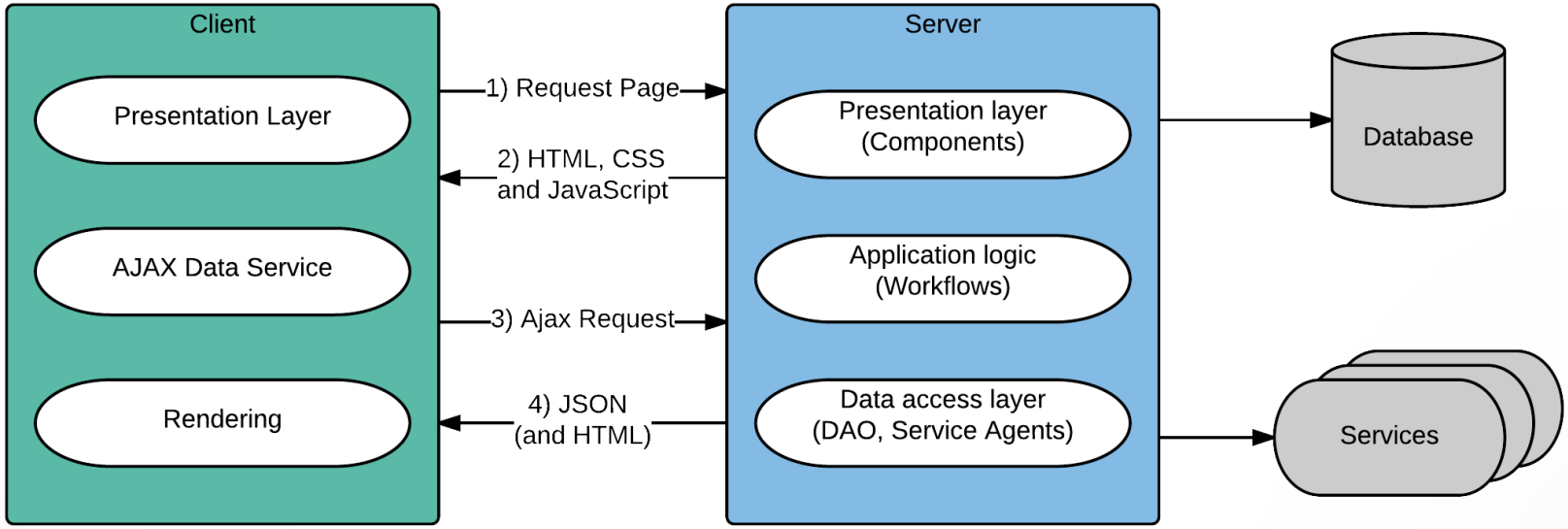
Client-Server Model



Web Evolution

2000's - AJAX Era

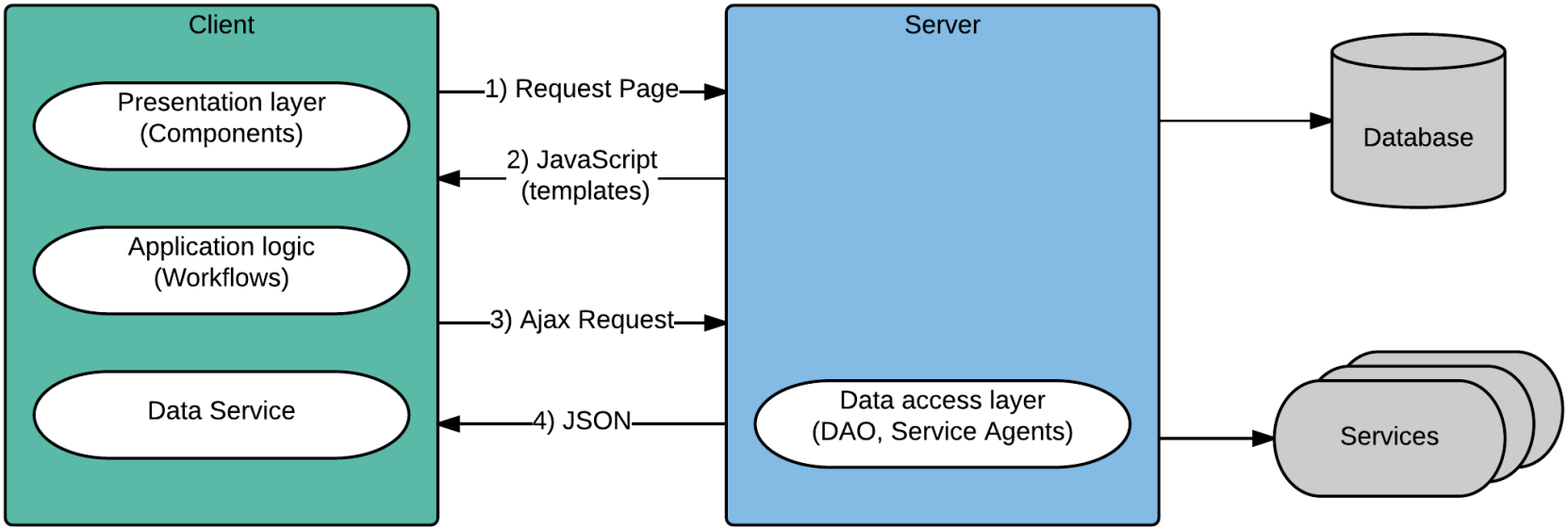
Client-Server Model



Web Evolution

2010's - Single Page App Era

Client-Server Model



Single Page App

Initial Server Markup

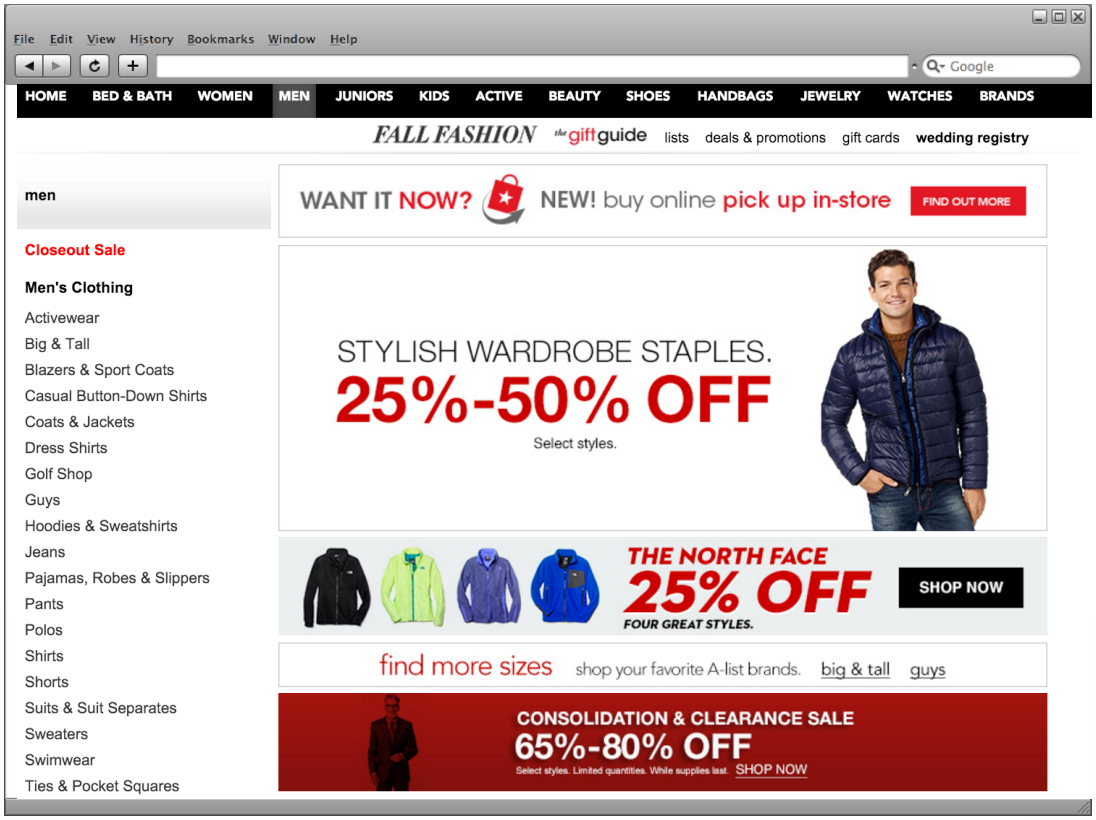
```
<html>
  <head>
  <title>SPA App</title>
      <script src="/app-bundle.js"></script>
  </head>
  <body>
  </body>
</html>
```

Single Page App

Rendering Flow

1. Download skeleton HTML

2. Download the JavaScript

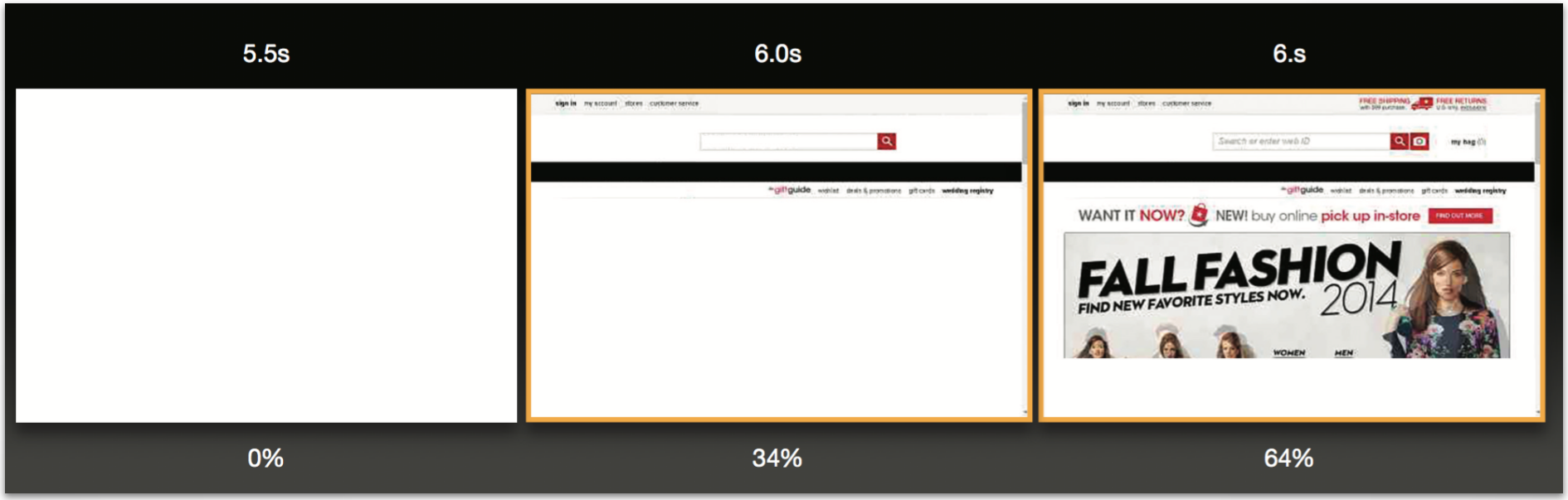


3. Evaluate JavaScript

4. Fetch Data from the API

Single Page App

Timeline

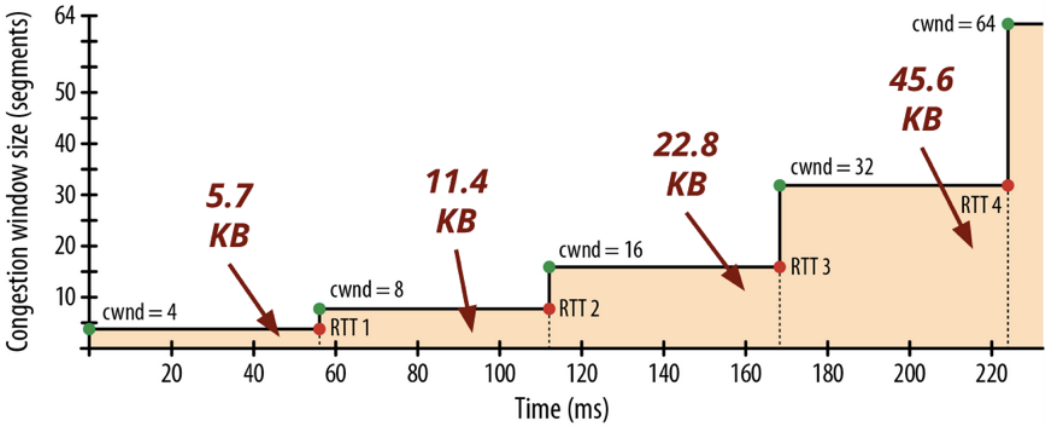


Single Page App

TCP Slow Start

"four roundtrips (…) and hundreds of milliseconds of latency, to reach 64 KB of throughput between the client and server"

"High Performance Browser Networking" by Ilya Grigorik



A congestion control mechanism, "slow start", is built into the TCP protocol to send the data in a growing number of segments to prevent sending more data than the network is capable of transmitting

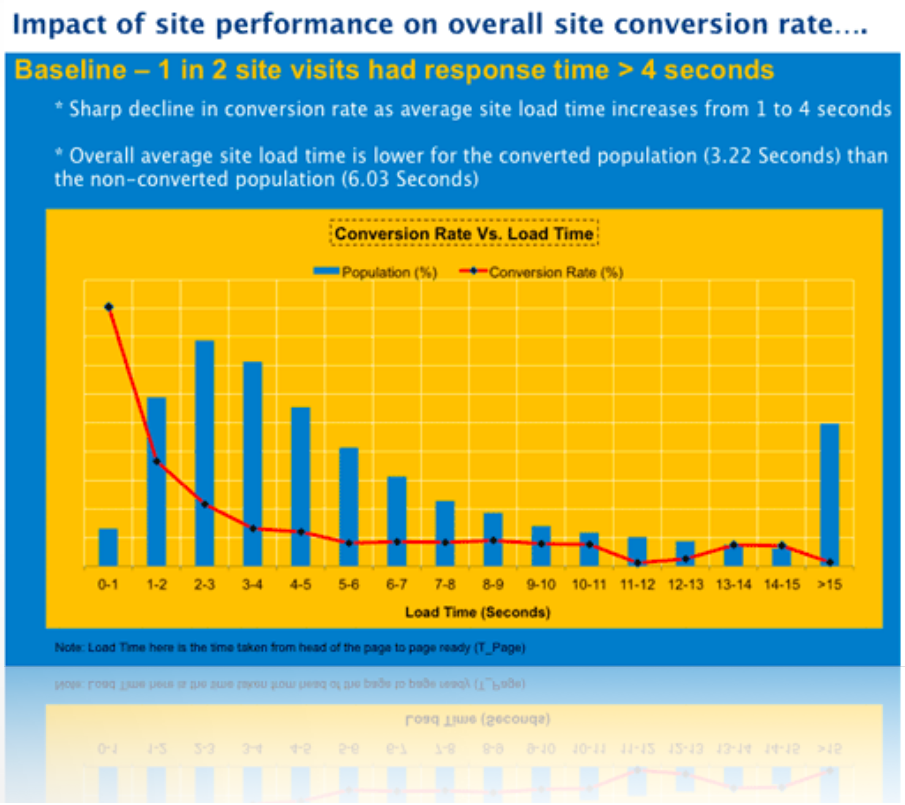
Single Page App

Increasing User Demand

www.radware.com

Single Page App

Time is money



- For every 1 second of improvement, experienced up to a 2% increase in conversions
- For every 100 ms of improvement, grew incremental revenue by up to 1%

Source: http://www.globaldots.com/how-website-speed-affects-conversion-rates

Isomorphic Rendering

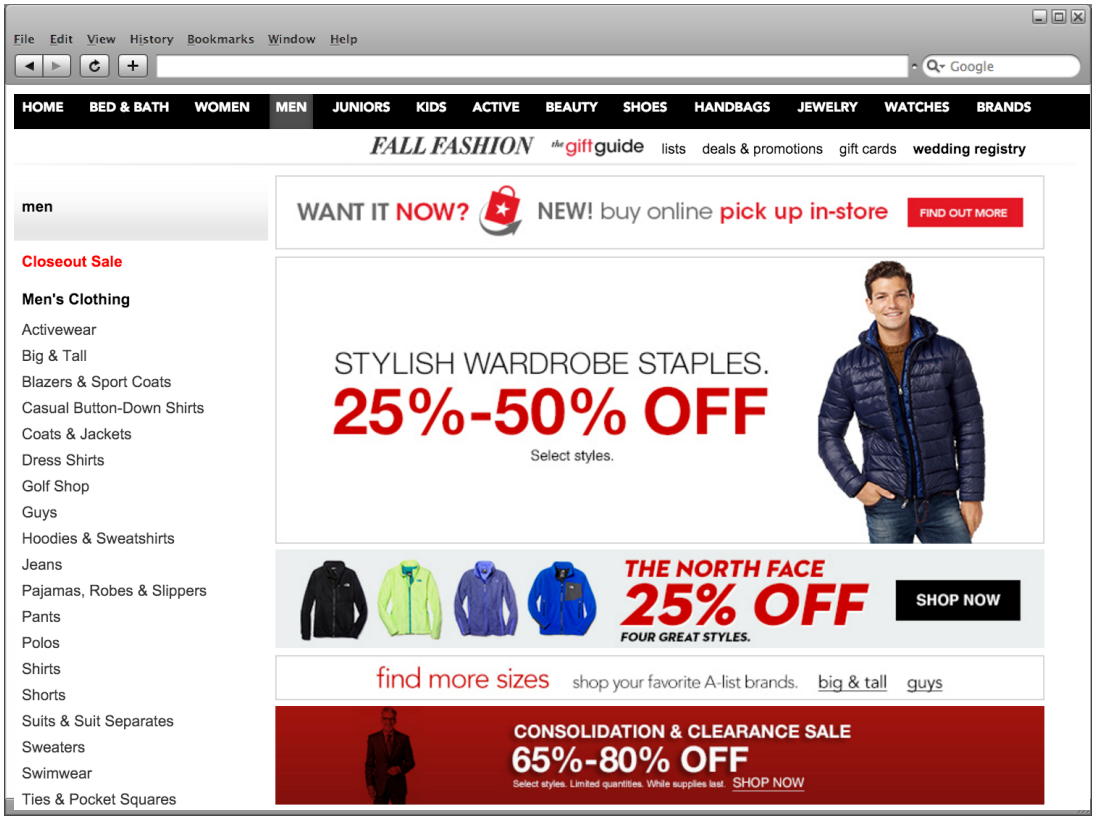
JavaScript rendered on the server and the client.

1. Render the HTML of a JavaScript app on the Server
2. Return the full HTML on a new page request
3. JavaScript loads and bootstraps the application (without destroying and rebuilding the initial HTML)

Isomorphic Rendering

1. Download skeleton HTML

2. Download the JavaScript



3. Evaluate JavaScript

Isomorphic JavaScript

- Important for initial page load performance
- Important for Search Engine Indexing and Optimization (SEO)
- Important for mobile users with low bandwidth
- Important for code maintenance

Isomorphic JavaScript sounds amazing but…

What if my front-end servers are running on **Java**

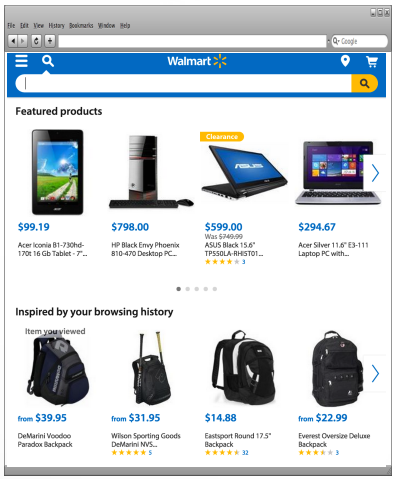
(and are battle tested in production)

Three possible solutions…

Option 1:

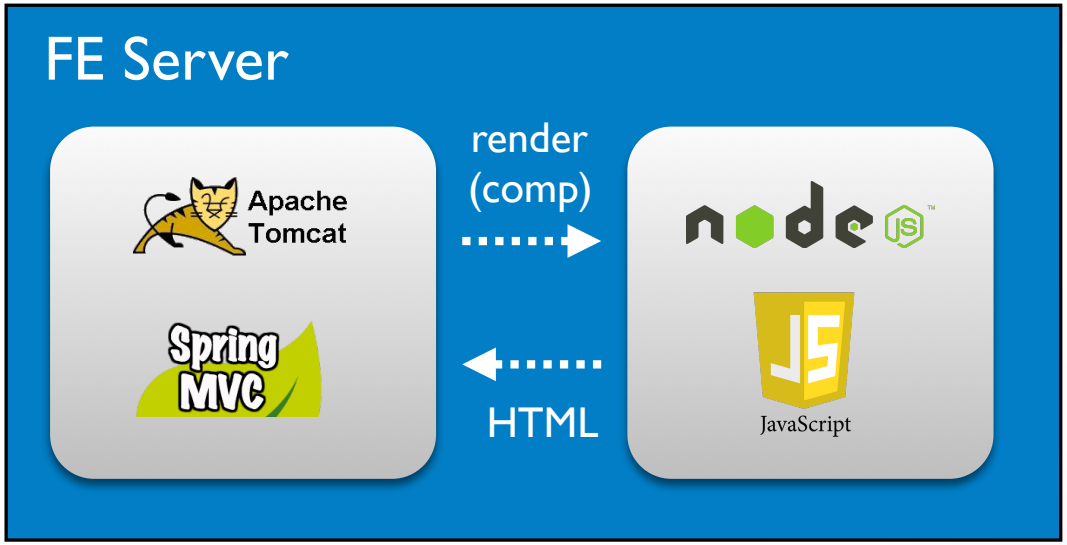
Delegate execution of JavaScript to an external process running **Node.js**

Delegate to Node.js



Fetch Page →

← HTML



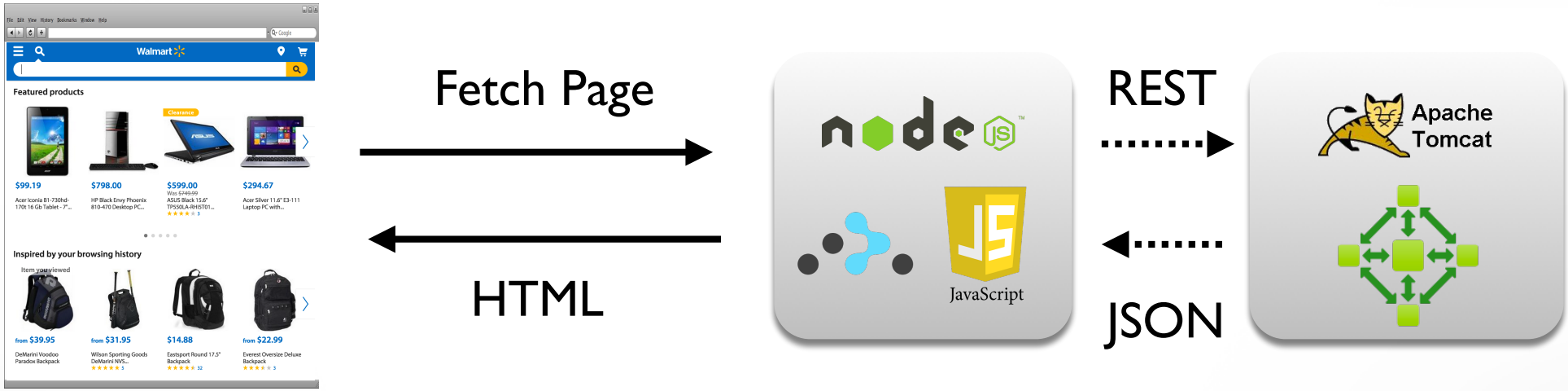
Downsides:

- more complicated deployments
- performance overhead of interacting with an external process

Option 2:

Have **Node** run as a **smart-proxy** in front of **Java**

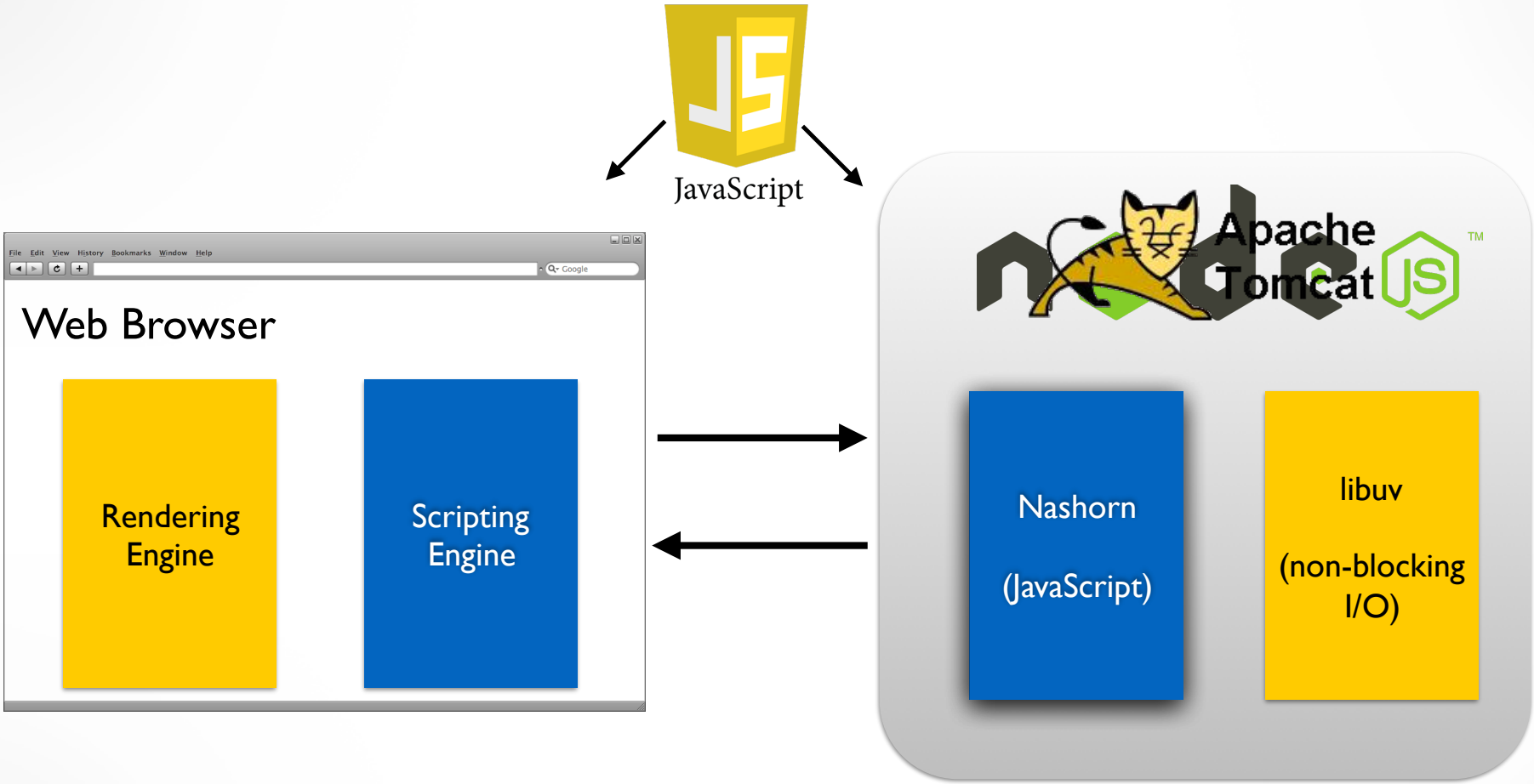
Node as a smart-proxy



Downsides:

- more complicated deployments
- performance overhead of interacting with an external process

Option 3:
Run JavaScript on the **JVM** with **Nashorn**

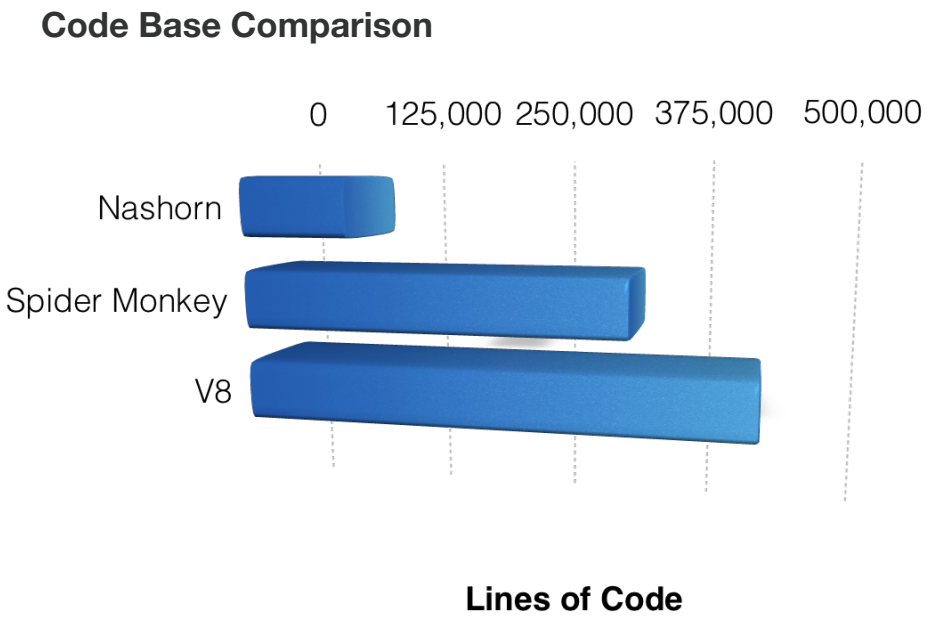

JavaScript
Web Browser
Rendering Engine
Scripting Engine
Apache Tomcat
Nashorn
(JavaScript)
libuv
(non-blocking I/O)

Java 8 Nashorn

- Java's embedded JavaScript engine that comes part of Java 8 (replacing Rhino).
- Nashorn supports the full ECMAScript 5.1 specification plus some extensions. (Future versions of Nashorn (Java 9) will include support for ECMAScript 6).
- It compiles JavaScript to Java bytecode providing interoperability between Java and JavaScript code

Java 8 Nashorn

- Automatic memory management
- State of the art JIT optimizations
- Man decades of high tech and tooling



Java 8 Nashorn

Javascript code can either be evaluated directly by passing javascript strings:

```
ScriptEngine engine = new ScriptEngineManager().getEngineByName("nashorn");
engine.eval("print('Hello World!');");
```

Or by passing a file reader pointing to a **.js** script file:

```
ScriptEngine engine = new ScriptEngineManager().getEngineByName("nashorn");
engine.eval(new FileReader("script.js"));
```

Java 8 Nashorn

Invoking JavaScript functions from Java:

JS Code

```
var helloWorld = function(name) {
    print('Hello, ' + name);
    return "Greetings from JavaScript";
};
```

Java Code

```
ScriptEngine engine = new ScriptEngineManager().getEngineByName("nashorn");
engine.eval(new FileReader("helloWorld.js"));
Invocable invocable = (Invocable) engine;
Object result = invocable.invokeFunction("helloWorld", "JavaOne!");
System.out.println(result);
System.out.println(result.getClass());
```

Output

```
Hello, JavaOne!
Greetings from JavaScript
class java.lang.String
```

Java 8 Nashorn

Sharing i18n code:

JS Code

```
var dateFromNow = function(lang, epoch) {
    moment.lang(lang);
    return moment(epoch).fromNow();
}
```

Java Code

```
public String dateFromNow(String locale, Long timestamp) throws Exception {
    return (String) nashornScriptEngine
                .invokeFunction("dateFromNow", locale, timestamp);
}
```

Java

```
dateFromNow("fr", timestamp);
```

JavaScript

```
dateFromNow('fr', timestamp);
```

il y a 4 ans

Java 8 Nashorn

Sharing View Logic Code:

JS Code

```
Handlebars.registerHelper('dateFormat', dateFormat);
var compiledTemplate = Handlebars.compile(template);
return compiledTemplate(data);
```

Template

```
<ul>
    {{#each comments}}
        <li>{{author}} - {{content}} ({{dateFormat timestamp locale}})</li>
    {{/each}}
</ul>
```

Java Code

```
@RequestMapping("/")
String home(Model model) {
    List comments = commentService.getComments();
    model.addAttribute("comments", comments);
    return "home";
}
```

Output

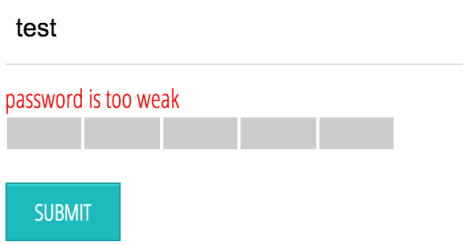
- Pete Hunt - Hey there! (5 days ago)
- Paul O'Shannessy - Nashorn is Great! (10 days ago)

Java 8 Nashorn

Sharing Validation Code:

Client Validation

Server Validation (if Client validation is bypassed)



JS Code

```
function isValidPassword(password){
    var score = scorePasswordStrength(password);
    return score >= 2;
}
```

Java Code

```
public Boolean isValidPassword(String password) {
    try {
        return (Boolean) nashornScriptEngine.invokeFunction("isValidPassword", password);
    } catch (ScriptException | NoSuchMethodException e) {
        throw new RuntimeException(e);
    }
}
```

Server-side **React**

- React.renderToString(..) - returns a string of the rendered component

Component

```
var HelloWorldComponent = React.createClass({
    render: function () {
        return <div>Hello World!</div>;
    }
});

var helloWorld = React.renderToString(<HelloWorldComponent/>);
```

Output

```
<div data-reactid=".fgvrzhg2yo"
    data-react-checksum="-1663559667">
    Hello World!
</div>
```

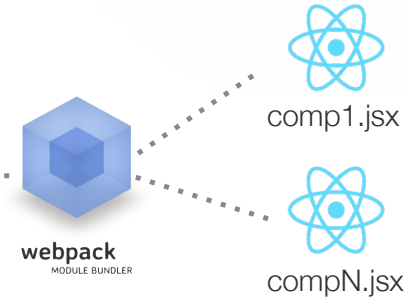
data-react-checksum: checksum of the DOM that is created. This allows React to reuse the DOM from the server when rendering the same component on the client.

Java 8 Nashorn

Server-Side rendering of React.js components from Java:

Script Engine

```
protected NashornScriptEngine initialValue() {
    NashornScriptEngine nashornScriptEngine = (NashornScriptEngine)
            new ScriptEngineManager().getEngineByName("nashorn");
    nashornScriptEngine.eval(read("js/nashorn-polyfill.js"));
    nashornScriptEngine.eval(read("static/server-bundle.js"));
    return nashornScriptEngine;
}
```



context (similar to node's vm module runInThisContext(..))

Invoke

```
public String renderProduct(List<Product> products) {
    Object html = nashornScriptEngine.invokeFunction("renderServer", products);
    return String.valueOf(html);
}
```

HTML

Output

```
<div class="prod-ProductPage ResponsiveContainer"
data-reactid=".pnamk0vcao" data-react-checksum="-581513169">
…
```

Java 8 Nashorn

Client-Side transition from server-side rendered components

Bootstrap

```
<body>
    <div class="js-content">{{content}}</div>
    <script>window.__state__={{data}}</script>
...
</body>
```

Client Code

```
import React from "react/addons";
import Page from "./page";

import "../styles/base.styl";

const rootEl = document.querySelector(".js-content");

React.render(<Page data={window.__state__}/>, rootEl);
```

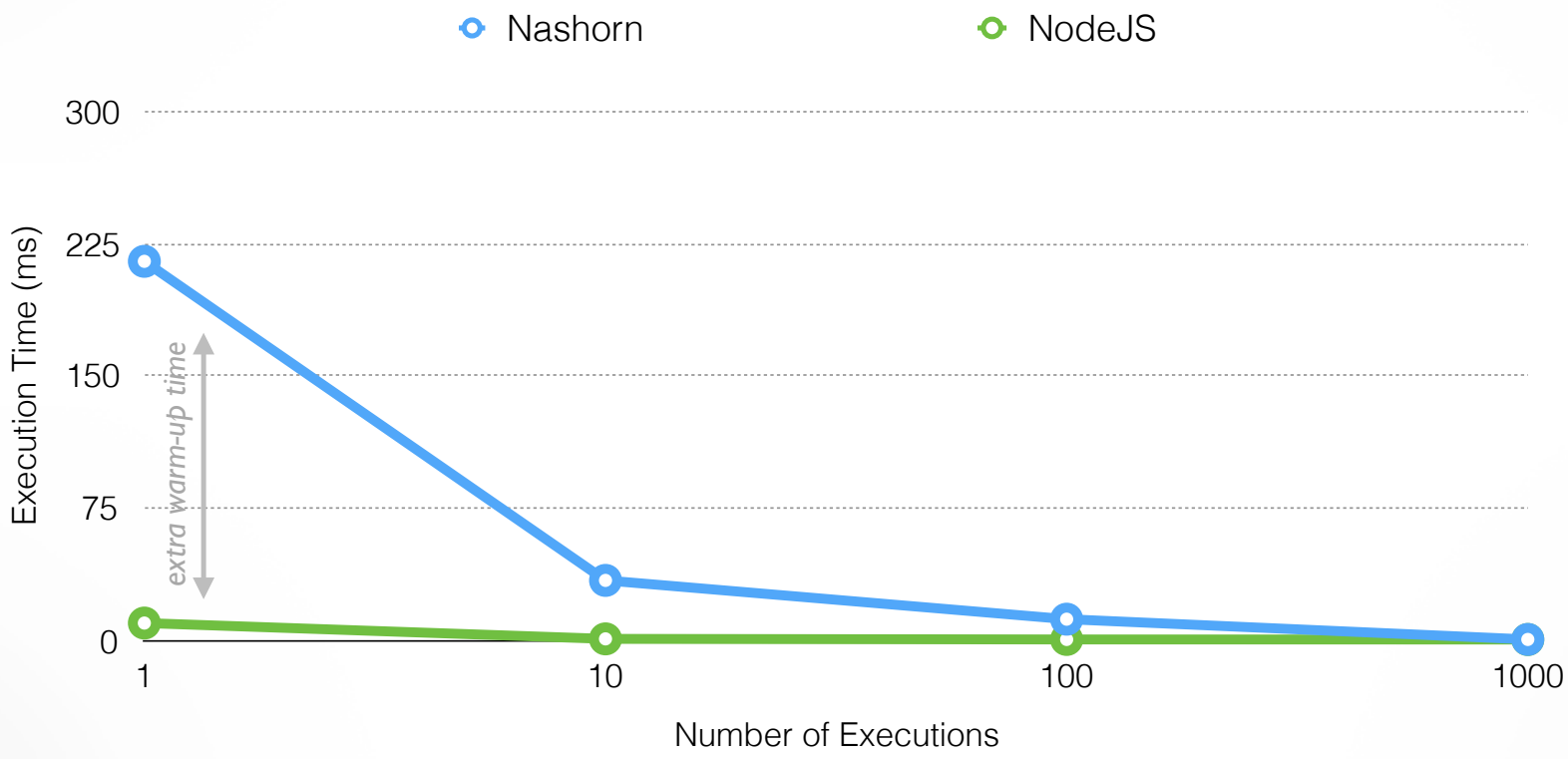
Output

```
<div class="prod-ProductPage ResponsiveContainer"
data-reactid=".pnamk0vcao" data-react-checksum="-581513169">
…
```

Nashorn Concurrency

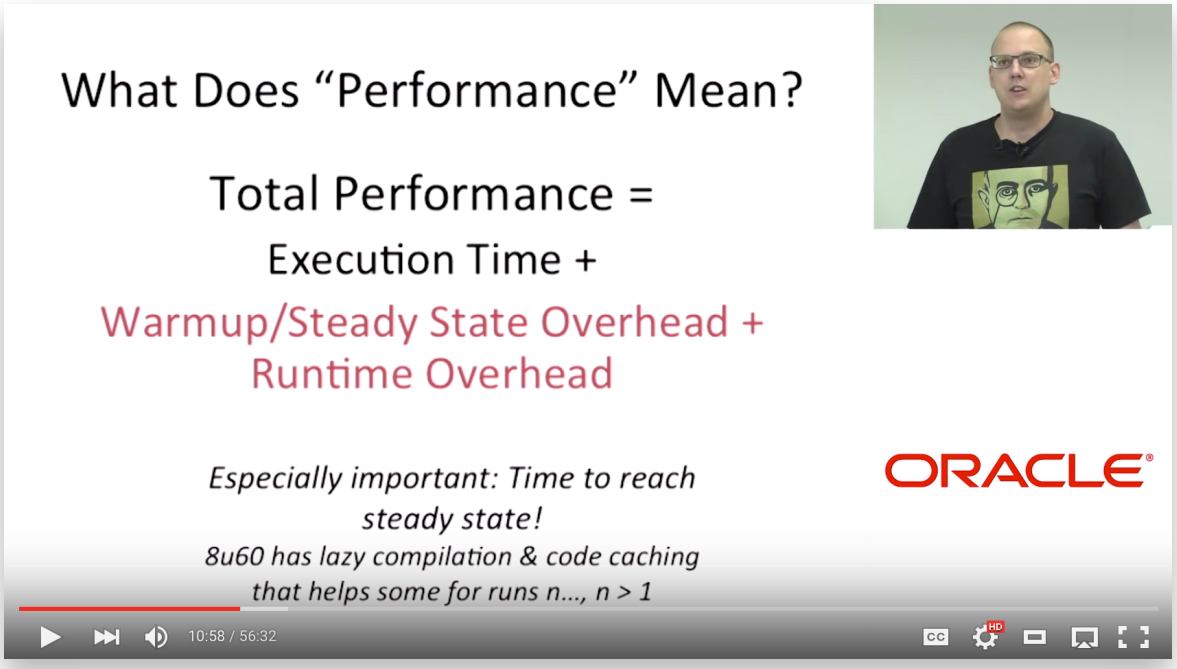
- In web browsers, there is no concurrent execution of your code.
- Thread-safety depends on your Javascript code. Nashorn itself will not make your code thread-safe.
- Use a ThreadLocal<ScriptEngine> when Javascript code is not thread-safe (i.e. Handlebars and React).

Nashorn Performance



https://github.com/maximenajim/java-vs-node-react-rendering-microbenchmark

Nashorn Performance



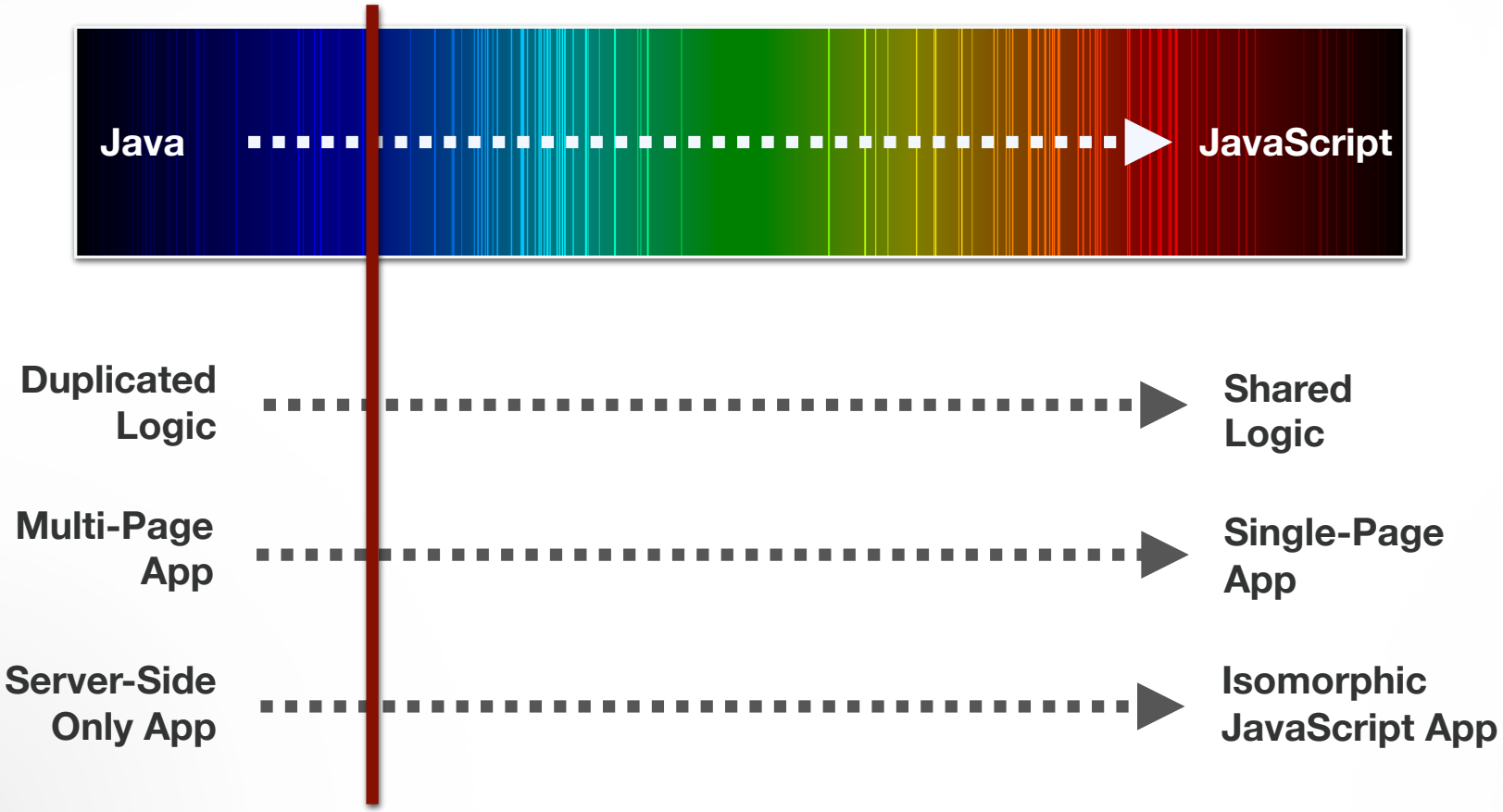
JVMLS 2015 - Nashorn for Java 9 - Marcus Lagergren

https://www.youtube.com/watch?v=aROpSjXr4TU

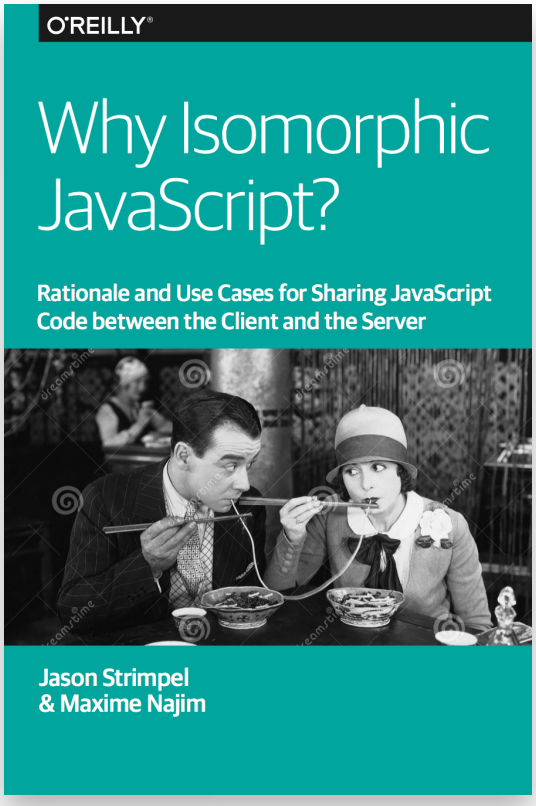
Demo

- https://github.com/maximenajim/isomorphic-validation-nashorn-example
- https://github.com/maximenajim/isomorphic-javascript-nashorn-example
- https://github.com/maximenajim/isomorphic-flux-javascript-nashorn-example

Nashorn Adoption Spectrum



More Info



The Golden Age of JavaScript began when web developers traded in their fat-server, thin-client approach for desktop-like web apps running in the browser. Unfortunately, that approach led to a succession of problems, so now the pendulum is swinging back in the other direction. Companies such as Walmart, Airbnb, Facebook, and Netflix have already adopted a new solution using JavaScript code on both the client and server.

Authors Jason Strimpel and Maxime Najim from WalmartLabs explain that isomorphic JavaScript is the latest in a series of engineering fixes that brings a harmonious equilibrium between the fat-server, fat-client pendulum, which emerged from the Ajax and Single Page Application eras.

Free Download: http://www.oreilly.com/web-platform/free/

Thank You

@softwarecrafts

https://github.com/maximenajim

www.oreilly.com/pub/au/6521

Nashorn vs. NodeJs

- Nashorn is only an implementation of ECMAScript and does not implement things like HTML5 Timers, nor the XMLHttpRequest specification, etc.

- Node.js adopted the browsers' concepts of event loops and task queues to reduce the conceptual gap between server- and client-side JavaScript.

- Luckily, the Nashorn environment is very extensible. Scripts running in the Nashorn engine can manipulate the global scope and access standard Java APIs to extend the environment.

Nashorn vs. NodeJs

- Nodyn project provides the Node.js API on the JVM.
- Avatar.js project brings the Node.js programming model, APIs and libraries to the Java platform. **(For now, the development of Avatar is on hold)**
- Interesting: SpringOne2GX 2014 Replay: Server-side JavaScript with Nashorn and Spring