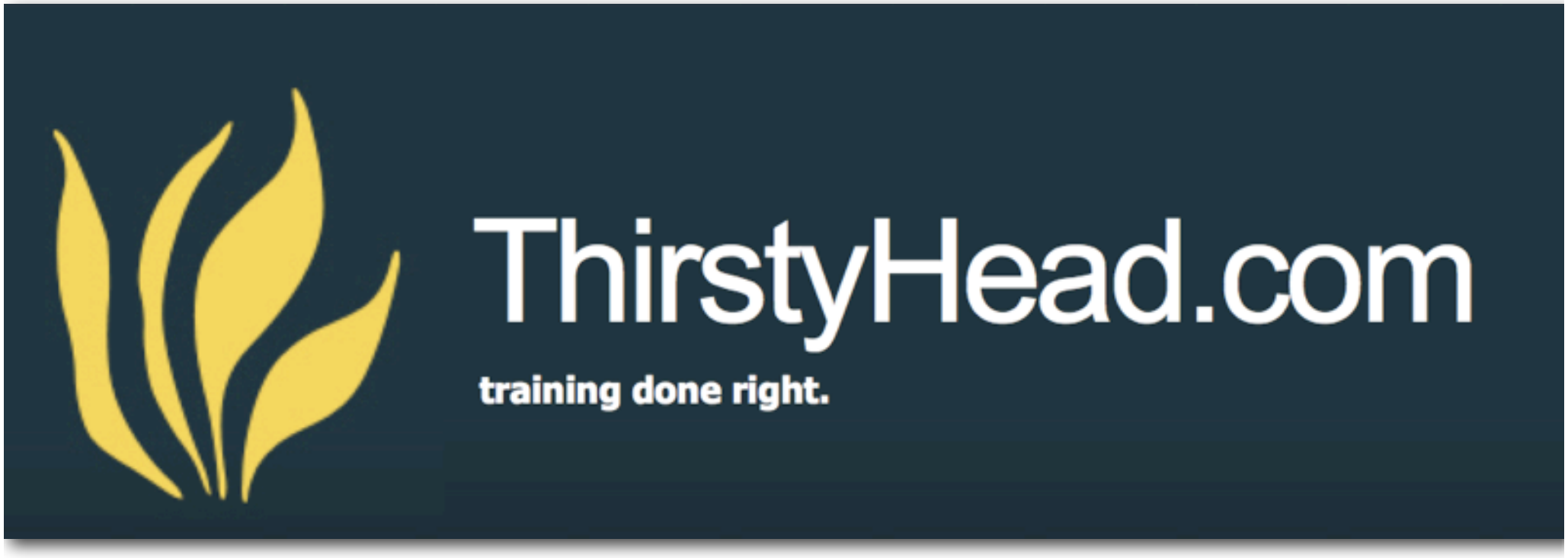
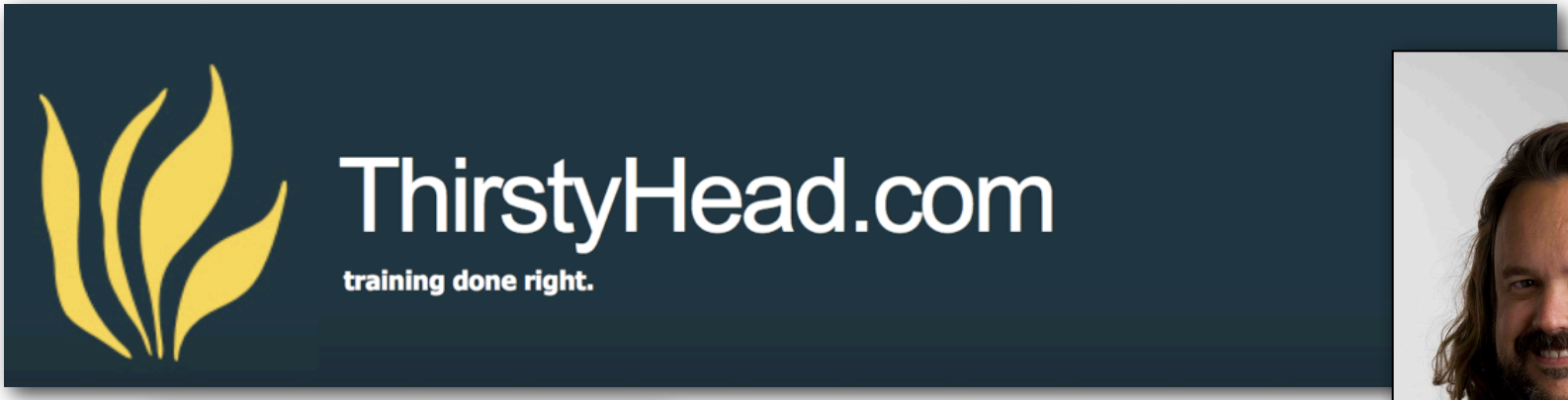
JavaScript for Java Developers

Scott Davis, ThirstyHead.com

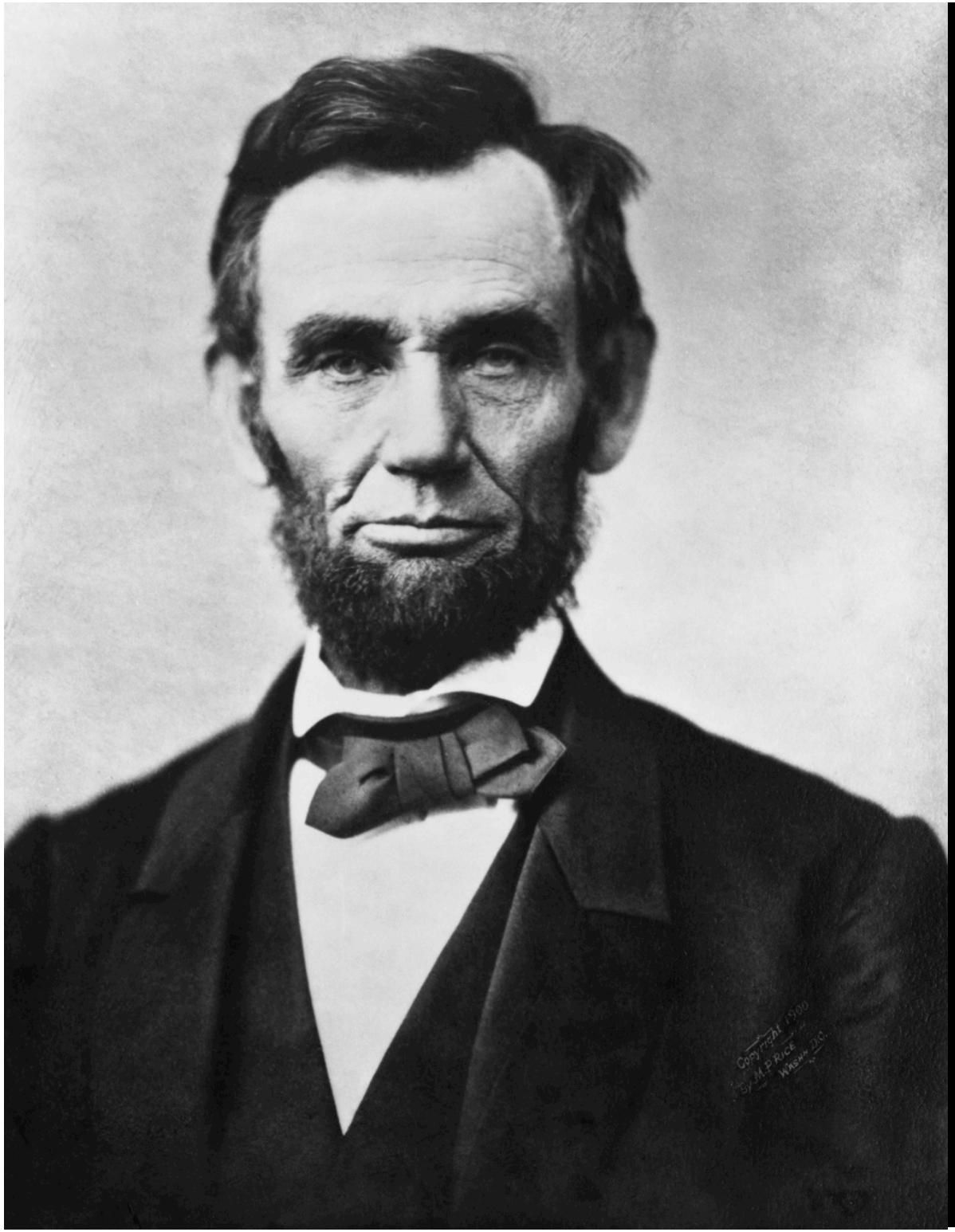

ThirstyHead.com
training done right.


ThirstyHead.com
training done right.


Scott Davis
@scottdavis99


The Pragmatic Programmers
GIS For Web Developers
Adding Where to Your Web Applications
The Pragmatic Programmers
Groovy Recipes
Greasing the Wheels of Java
Scott Davis
Jason Rudolph
Getting Started with Grails Second Edition
A modern web framework for the Java platform
ENTERPRISE SOFTWARE DEVELOPMENT SERIES
InfoQ


HTML



"If this is coffee, please bring me some tea;

If this is tea, please bring me some coffee..."

Abraham Lincoln

Coffee != Tea

Java != JavaScript

JavaScript sounds like it has something to do with Java. It doesn't.

Apart from some superficial syntactical similarities, they have nothing in common.

Java is to JavaScript as ham is to hamster.

History

Strong vs. Weak Typing

Block vs. Function scope

Object-oriented vs. Functional

Compiling vs. Minifying

History

Strong vs. Weak Typing

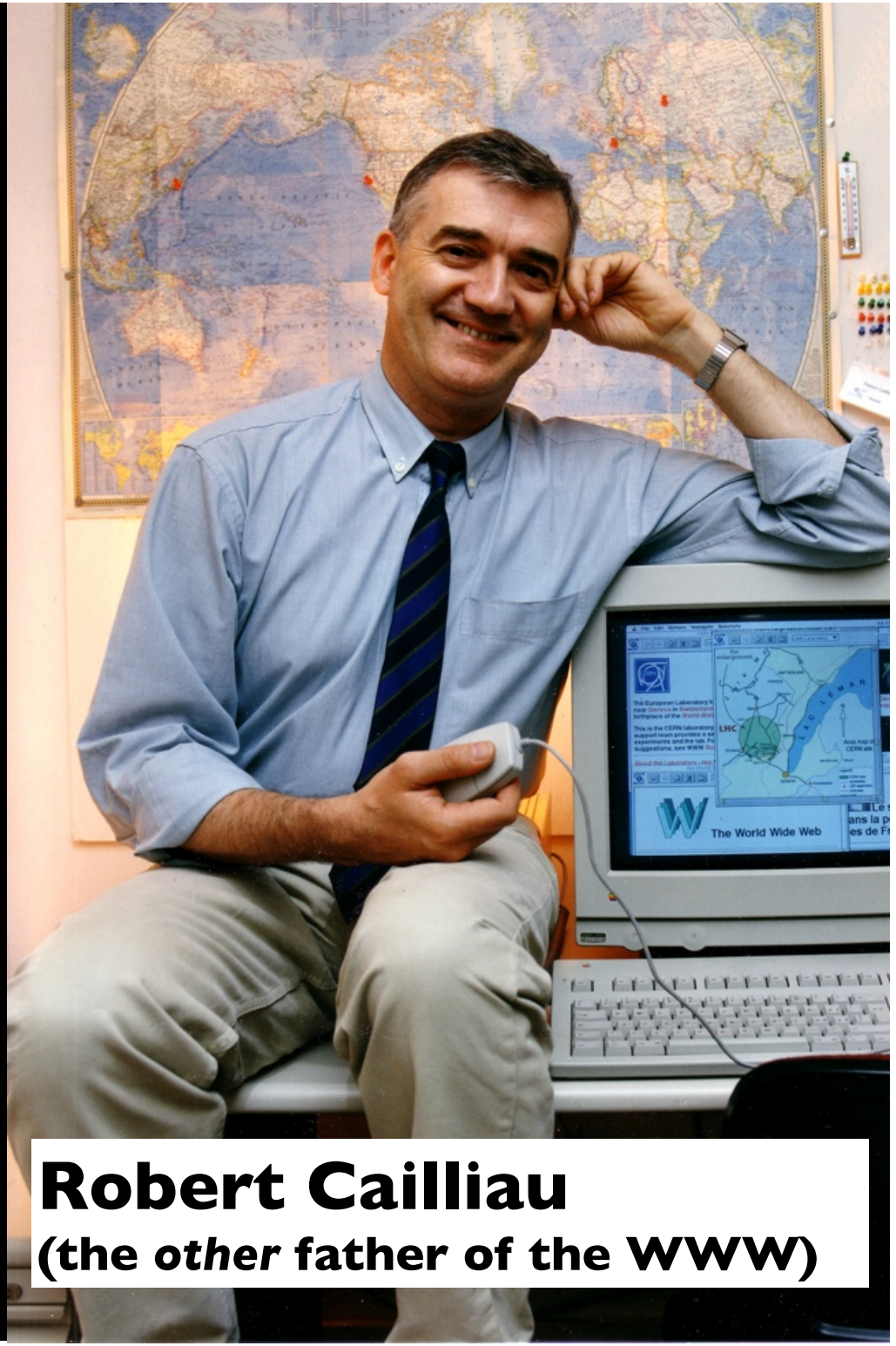
Block vs. Function scope

Object-oriented vs. Functional

Compiling vs. Minifying

[In 1990] I was convinced that we needed to build-in a programming language, but the developers, Tim [Berners-Lee], were very much opposed.

The net result is that the programming-vacuum filled itself with ***the most horrible kluge in the history of computing:*** JavaScript.



Robert Cailliau
(the *other* father of the WWW)

version 0.9 beta

File Edit View Go Bookmarks Options Directory Help

Back Forward Home Reload Images Open Find Stop

Location: about:

Guided Tour | What's New | Questions | Net Search | Net Directory | Newsgroups

Mosaic Netscape version 0.9 beta

This is *BETA* software subject to the license agreement set forth in the README file.
Please read and agree to all terms before using this software.

Report any problems to win_cbug@mcom.com.

COMPUTERWORLD
THE VOICE OF IT MANAGEMENT

The A-Z of Programming Languages: JavaScript

Brendan Eich created JavaScript in 1995 with the aim to provide a "glue language" for Web designers and part grown to become one of the most widely used languages on the planet.

Naomi Hamilton (Computerworld) | 31 July, 2008 21:04

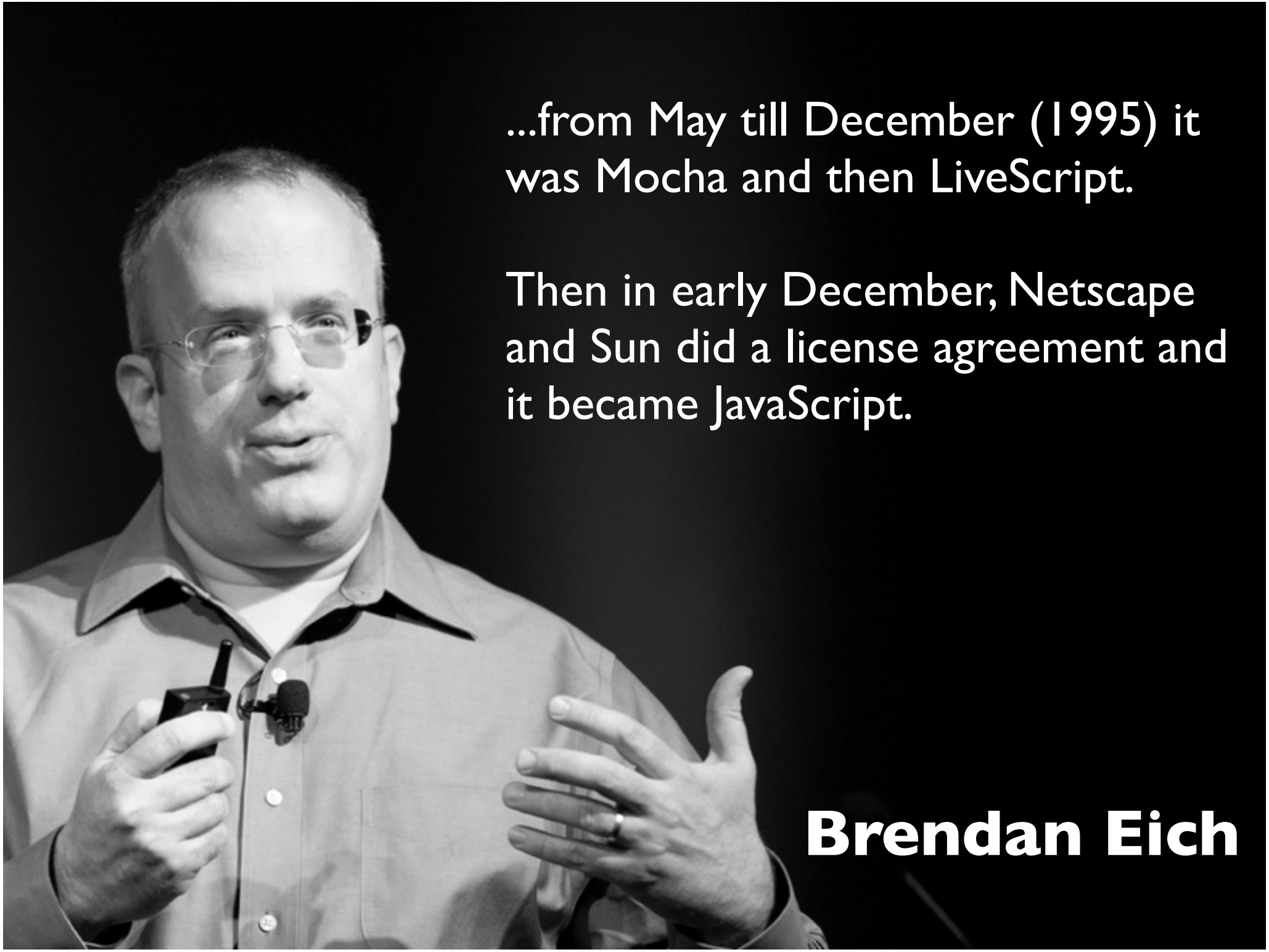
I joined Netscape on 4 April 1995, with the goal of embedding the Scheme programming language, or something like it, into Netscape's browser.

Scheme (programming language)

From Wikipedia, the free encyclopedia

This article is about the programming language. For other uses, see Scheme.

Scheme is one of the two main dialects of the programming language Lisp. Unlike Common Lisp, the other main dialect, Scheme follows a minimalist design philosophy specifying a small standard core with powerful tools for language extension. Its compactness and elegance have made it popular with educators, language designers, programmers, implementors, and hobbyists.

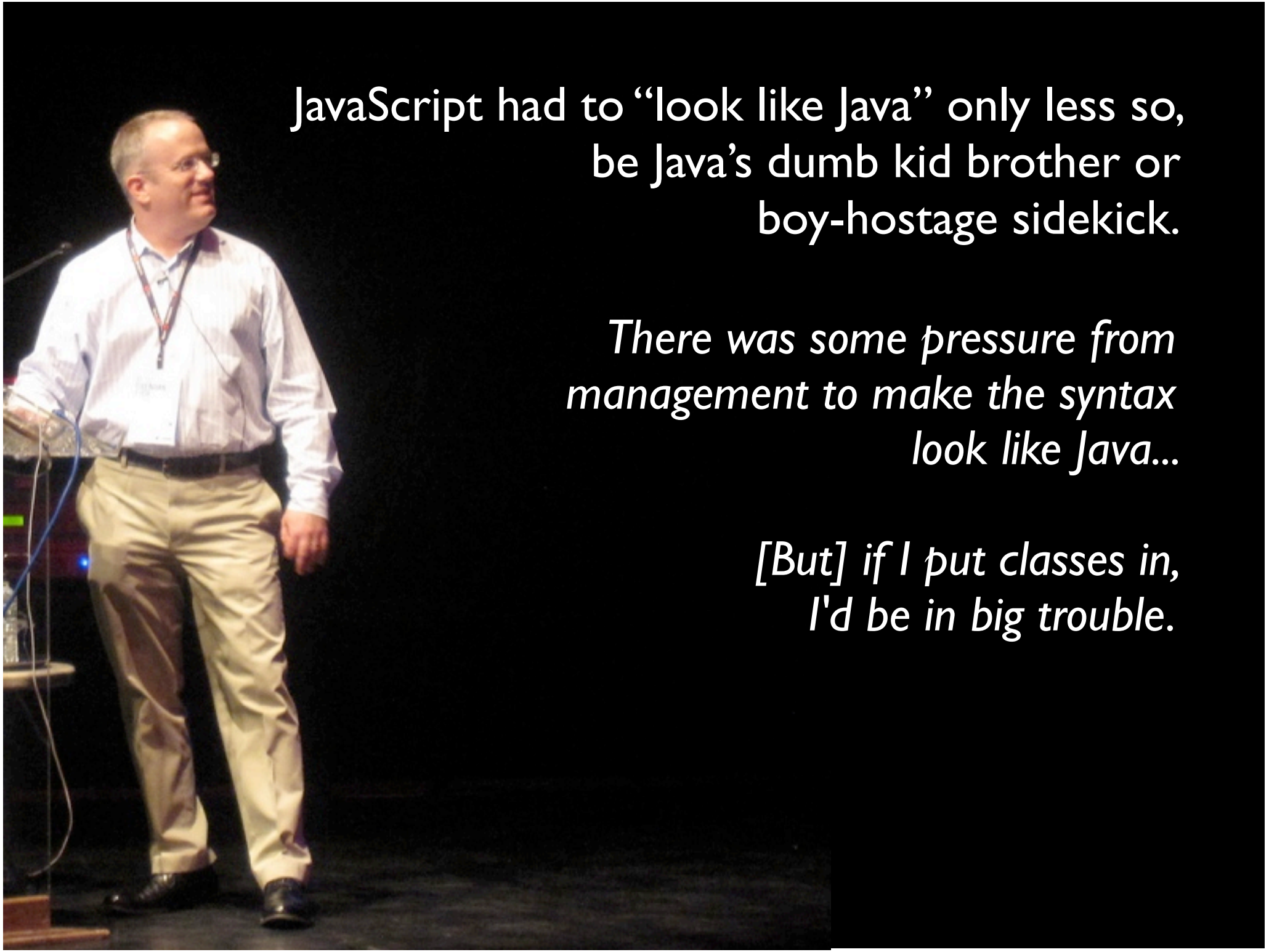

...from May till December (1995) it was Mocha and then LiveScript.
Then in early December, Netscape and Sun did a license agreement and it became JavaScript.
Brendan Eich

The big debate inside Netscape therefore became, "Why two languages? Why not just Java?"

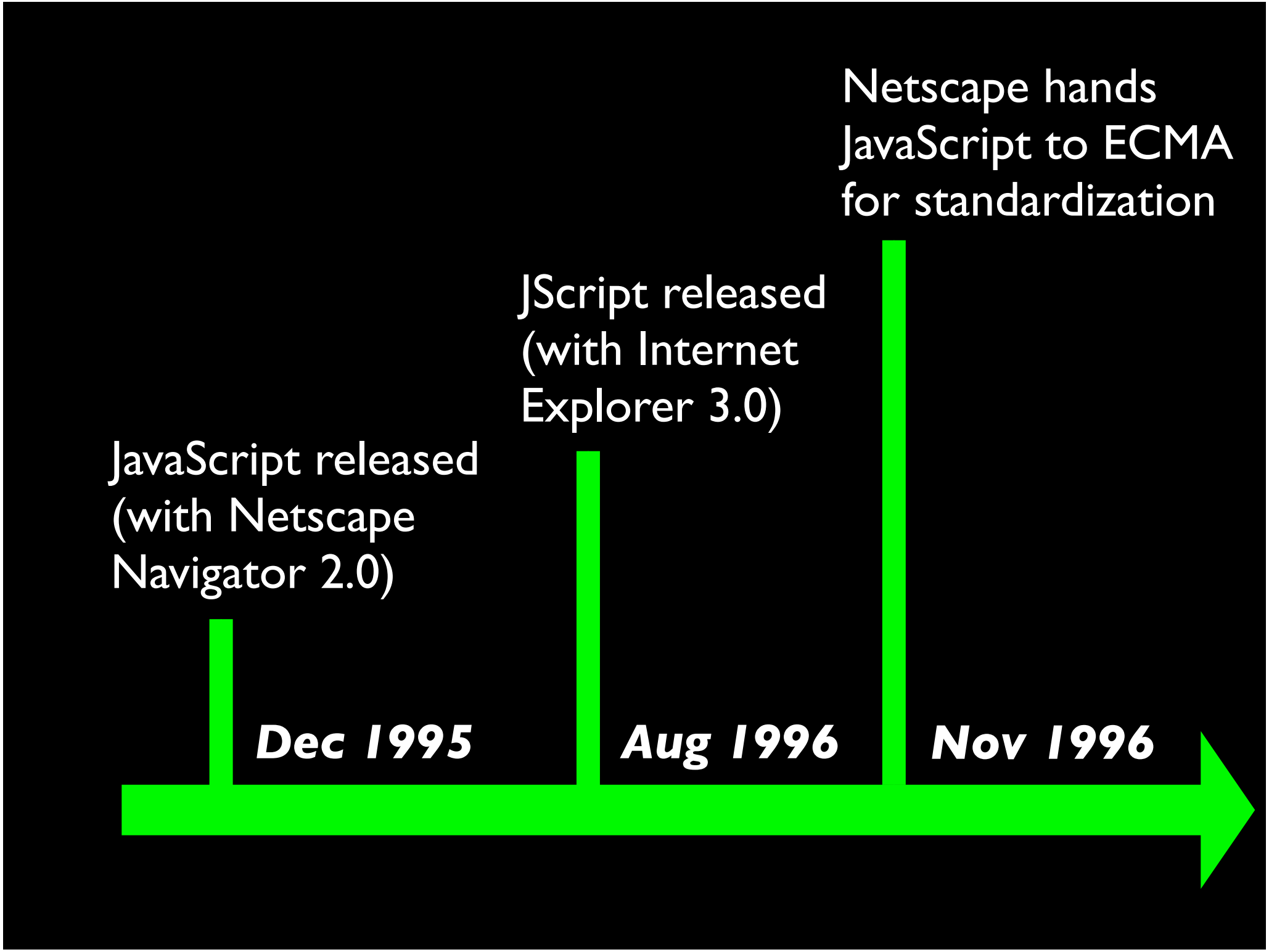
The answer was that two languages were required to serve the two mostly-disjoint audiences...

1. the component authors...

2. and the "scripters", amateur or pro, who would write code directly embedded in HTML.


JavaScript had to “look like Java” only less so,
be Java’s dumb kid brother or
boy-hostage sidekick.
There was some pressure from management to make the syntax look like Java...
[But] if I put classes in, I'd be in big trouble.

JavaScript vs. JScript vs. ECMAScript


Netscape hands JavaScript to ECMA for standardization
JScript released (with Internet Explorer 3.0)
JavaScript released (with Netscape Navigator 2.0)
Dec 1995
Aug 1996
Nov 1996

ECMA

European Country Music Association

EAST COAST MUSIC ASSOCIATION

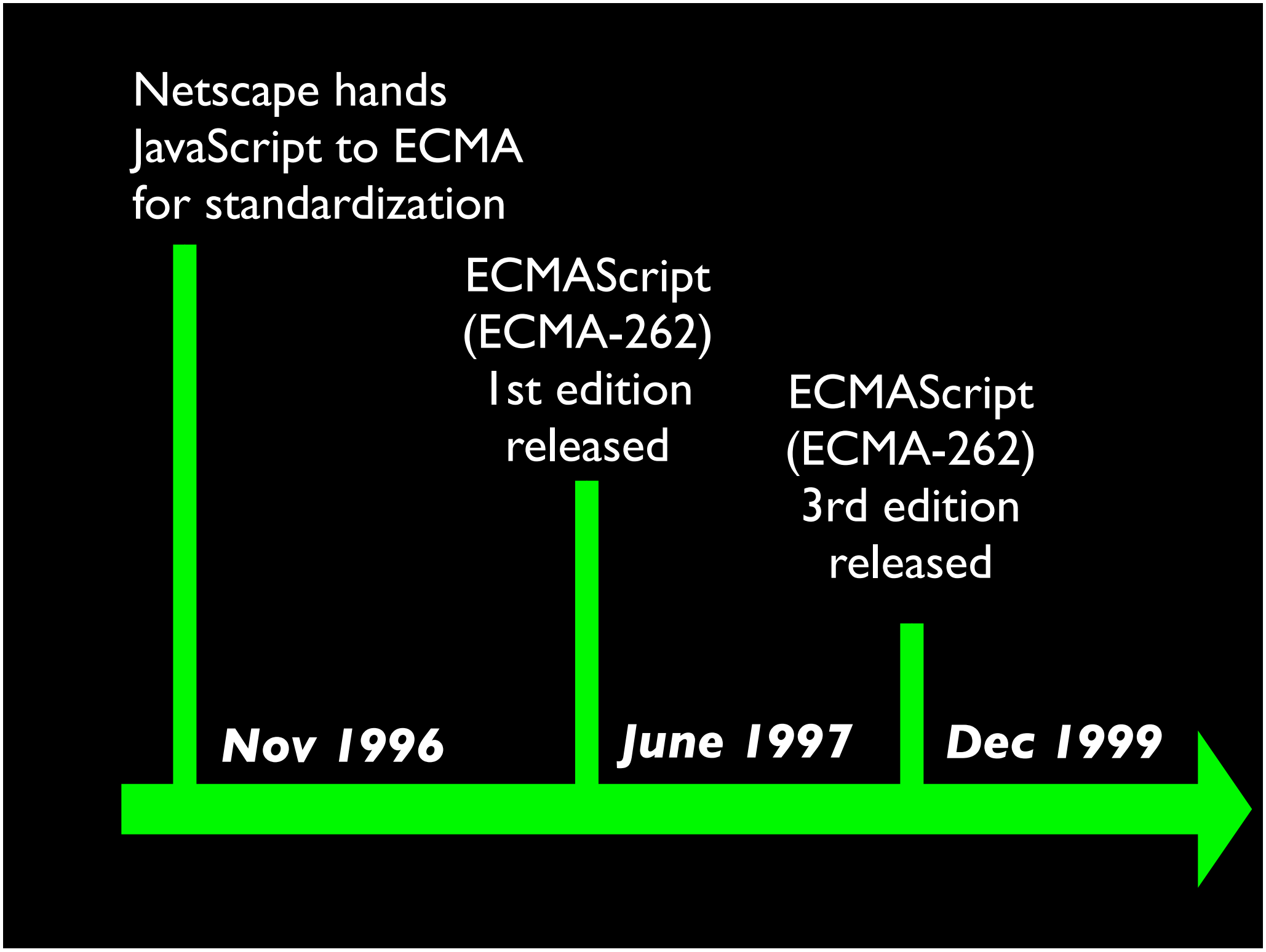
CANADIAN MUSIC. EAST COAST STYLE.

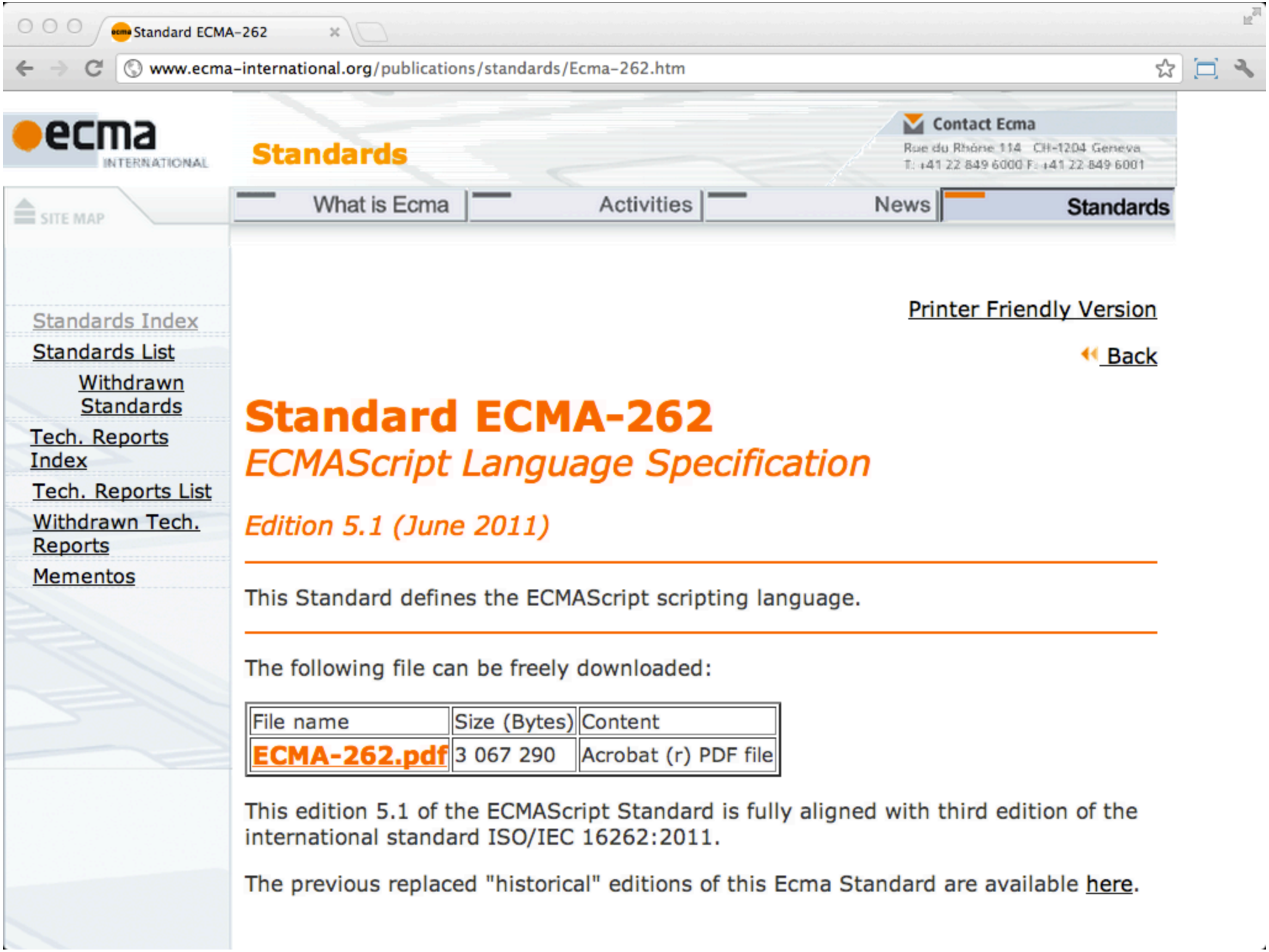
ECMA

European
Carton Makers
Association

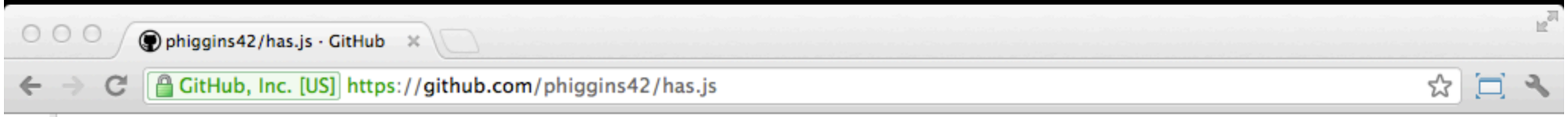
ecma INTERNATIONAL

This meeting was held on April 27, 1960, in Brussels; it was decided that an association of manufacturers should be formed which would be called **European Computer Manufacturers Association** or for short ECMA, and a Committee was nominated to prepare the formation of the Association and to draw up By-laws and Rules.


Netscape hands JavaScript to ECMA for standardization
ECMAScript (ECMA-262) 1st edition released
ECMAScript (ECMA-262) 3rd edition released
Nov 1996
June 1997
Dec 1999


Standard ECMA-262
www.ecma-international.org/publications/standards/Ecma-262.htm
ecma INTERNATIONAL
Standards
Contact Ecma
Rue du Rhône 114 CH-1204 Geneva
T: +41 22 849 6000 F: +41 22 849 6001
SITE MAP
What is Ecma
Activities
News
Standards
Standards Index
Standards List
Withdrawn Standards
Tech. Reports Index
Tech. Reports List
Withdrawn Tech. Reports
Mementos
Printer Friendly Version
Back
Standard ECMA-262
ECMAScript Language Specification
Edition 5.1 (June 2011)
This Standard defines the ECMAScript scripting language.
The following file can be freely downloaded:
File name
Size (Bytes)
Content
ECMA-262.pdf
3 067 290
Acrobat (r) PDF file
This edition 5.1 of the ECMAScript Standard is fully aligned with third edition of the international standard ISO/IEC 16262:2011.
The previous replaced "historical" editions of this Ecma Standard are available here.

Test for ES 5 browser support using Has.js:



Currently, the testing convention is `has('somefeature')` returns *Boolean*, e.g.:

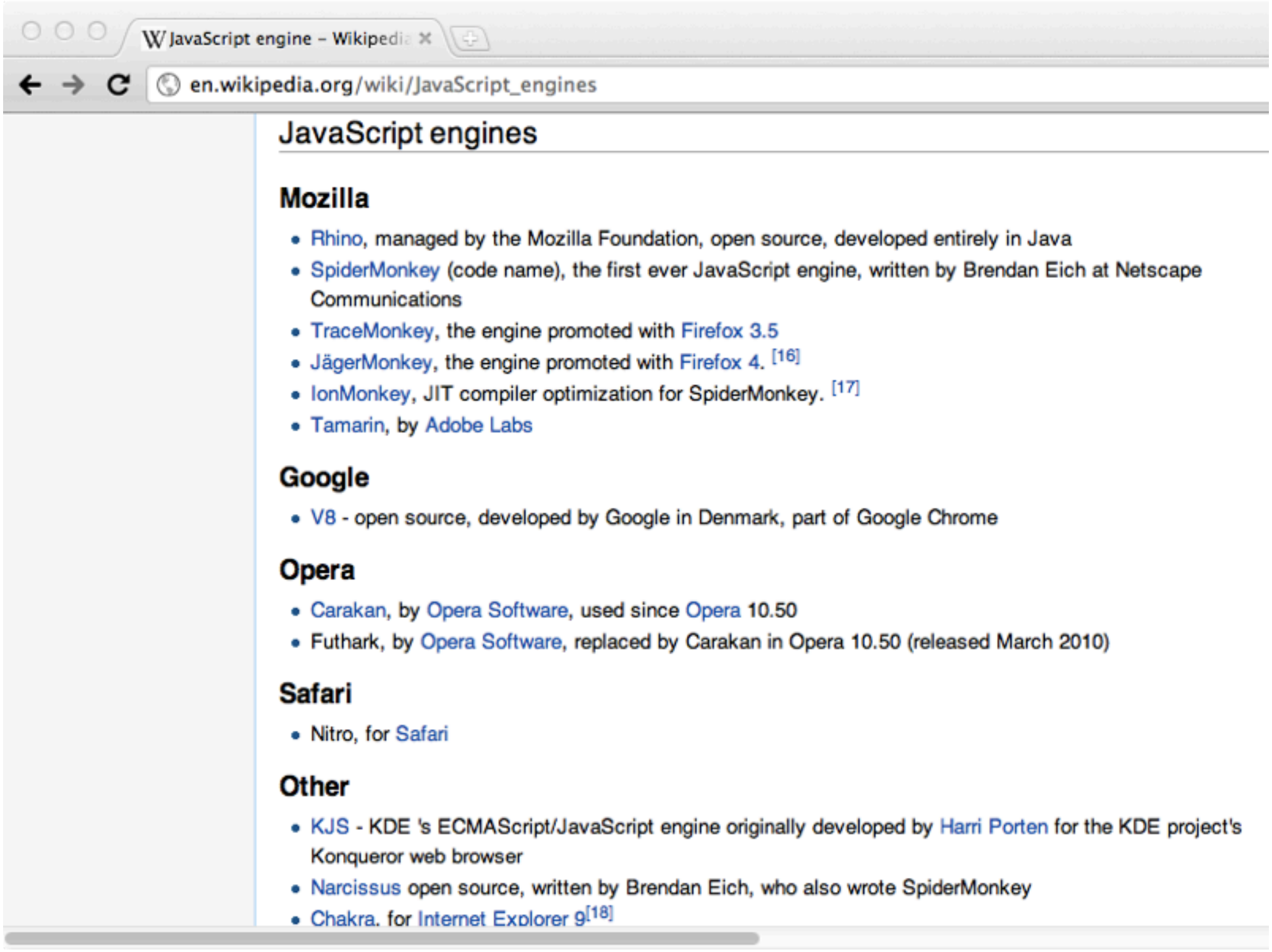
```
if(has("function-bind")){
    // your enviroment has a native Function.prototype.bind
}else{
    // you should get a new browser.
}
```

In the real world, this may translate into something like:

```
mylibrary.trim = has("string-trim") ? function(str){
    return (str || "").trim();
} : function(str){
    /* do the regexp based string trimming you feel like using */
}
```

By doing this, we can easily defer to browser-native versions of common functions, augment prototy
supplement the natives, or whatever we choose.


What is a Polyfill?
remysharp.com/2010/10/08/what-is-a-polyfill/
remy sharp's b:log
About [code] and all that jazz
"Full Frontal evolution in action with @chrismahon www.flickr.com/photos/remysharp/692..." 1 minute ago
Blog
jQuery
Projects
Twitter Apps
Talks
About
Work
« WebSockets in PhoneGap Projects
CSS selector for :parent targeting (please) »
8
OCT
What is a Polyfill?
A polyfill, or polyfiller, is a piece of code (or plugin) that provides the technology that you, the developer, expect the browser to provide natively. Flattening the API landscape if you will.
Alex Sexton also classifies polyfilling as a form of Regressive Enhancement. I think that sums it up nicely.
Paul also defines it as:
A shim that mimics a future API providing fallback functionality to older browsers.



JavaScript engines

Mozilla

- Rhino, managed by the Mozilla Foundation, open source, developed entirely in Java
- SpiderMonkey (code name), the first ever JavaScript engine, written by Brendan Eich at Netscape Communications
- TraceMonkey, the engine promoted with Firefox 3.5
- JägerMonkey, the engine promoted with Firefox 4. [16]
- IonMonkey, JIT compiler optimization for SpiderMonkey. [17]
- Tamarin, by Adobe Labs

Google

- V8 - open source, developed by Google in Denmark, part of Google Chrome

Opera

- Carakan, by Opera Software, used since Opera 10.50
- Futhark, by Opera Software, replaced by Carakan in Opera 10.50 (released March 2010)

Safari

- Nitro, for Safari

Other

- KJS - KDE 's ECMAScript/JavaScript engine originally developed by Harri Porten for the KDE project's Konqueror web browser
- Narcissus open source, written by Brendan Eich, who also wrote SpiderMonkey
- Chakra, for Internet Explorer 9[18]

Desktop Browsers

Smartphones, Tablets

Flash, PDFs, desktop widgets

Internet-enabled TVs, Blu-ray players

Coming soon to a server near you:

History

Strong vs. Weak Typing

Block vs. Function scope

Object-oriented vs. Functional

Compiling vs. Minifying


"Coffee should be
black as hell,
strong as death,
and sweet as love."
- Turkish proverb

Strongly-typed (Java):

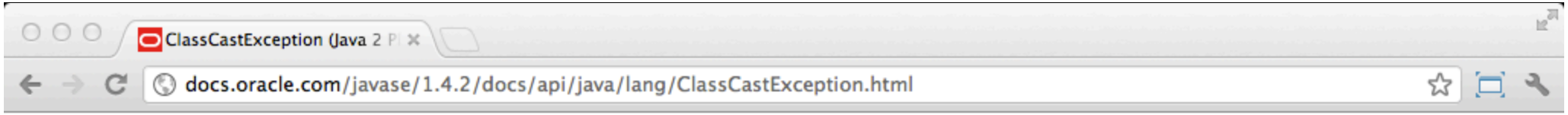
Strongly-typed (Java):

```
String name = “Scott”;
```

```
Date now = new Date();
```

```
Person p = new Person(name, now);
```

```
Person p = “Scott”;
// throws ClassCastException
```



java.lang

Class ClassCastException

java.lang.Object
  └ java.lang.Throwable
      └ java.lang.Exception
          └ java.lang.RuntimeException
              └ **java.lang.ClassCastException**

All Implemented Interfaces:

Serializable

---

public class **ClassCastException**
extends RuntimeException

Thrown to indicate that the code has attempted to cast an object to a subclass of which it is not an instance. For example, the following code generates a `ClassCastException`:

```
Object x = new Integer(0);
System.out.println((String)x);
```

Since:

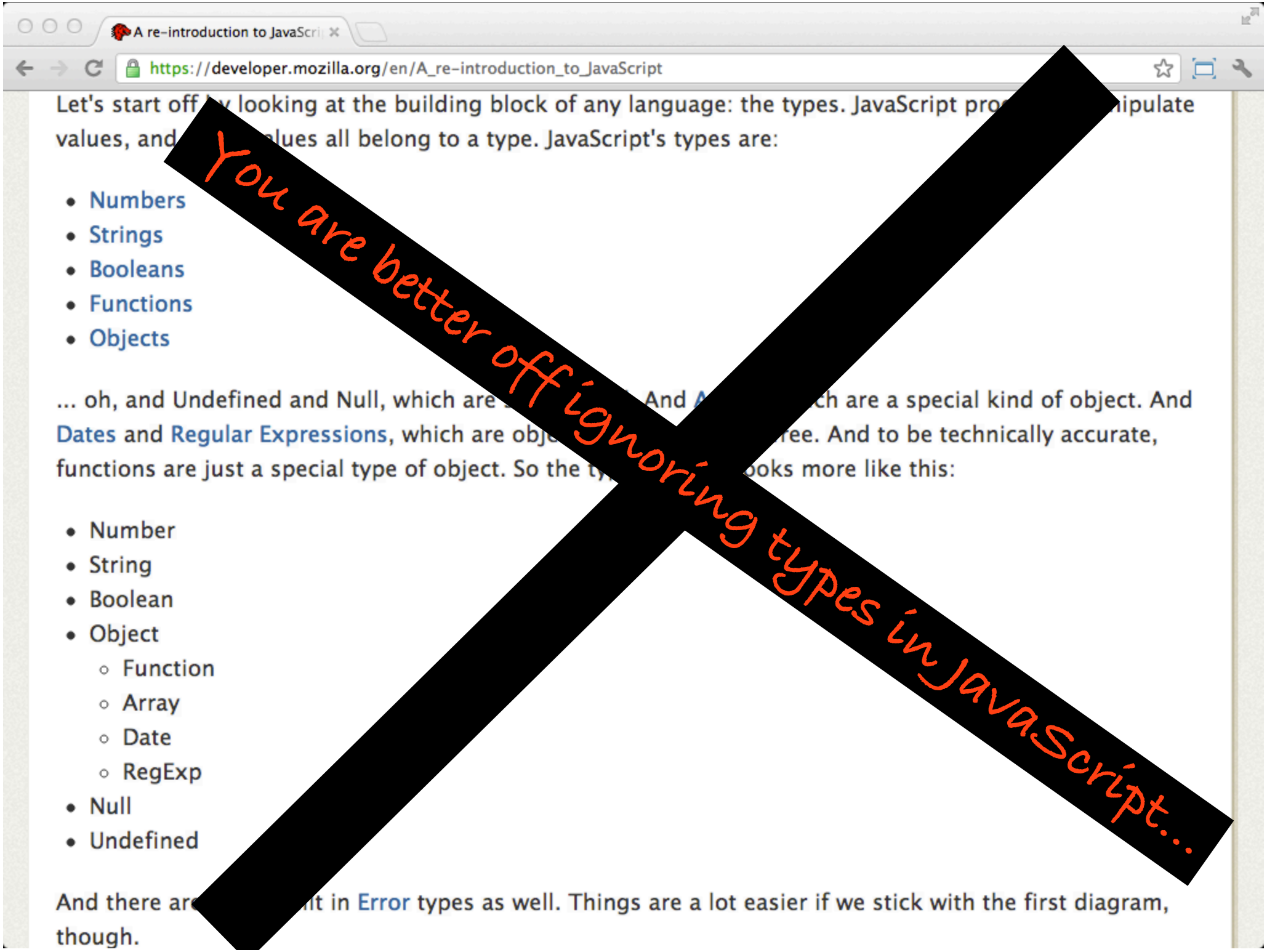
JDK1.0

See Also:

Weakly-typed (JavaScript):

```
var name = “Scott”;
var now = new Date();
var p = new Person(name, now);
```

Notice I didn't say that JavaScript was "typeless"...


A re-introduction to JavaScri
https://developer.mozilla.org/en/A_re-introduction_to_JavaScript
Let's start off by looking at the building block of any language: the types. JavaScript pro ipulate values, and lues all belong to a type. JavaScript's types are:
Numbers
Strings
Booleans
Functions
Objects
... oh, and Undefined and Null, which are And ch are a special kind of object. And Dates and Regular Expressions, which are obj ee. And to be technically accurate, functions are just a special type of object. So the t ooks more like this:
Number
String
Boolean
Object
Function
Array
Date
RegExp
Null
Undefined
And there ar lt in Error types as well. Things are a lot easier if we stick with the first diagram, though.
You are better off ignoring types in JavaScript...

== vs. ===

(aka JavaScript “Truthiness”)

```java
public class JavaType{
  public static void main(String[] args){
    System.out.println(1 == 1);                 //true
    System.out.println("Java" == "Java");       //true

    String lang = "Java";
    System.out.println("Java" == lang);         //true
    String anotherLang = new String("Java");
    System.out.println(anotherLang == lang);    //false (?!?)
  }
}
```

In Java:

For primitives, use == for value equality

For classes, use == for instance equality

Autoboxing:

Eliminates the difference between primitives and objects, eh?

```
1  public class Autobox{
2    public static void main(String[] args){
3      Integer a = 1;
4      Integer b = 1;
5      System.out.println(a == b);    //true
6
7      Integer c = new Integer(1);
8      Integer d = new Integer(1);
9      System.out.println(c == d);    //FALSE!
10
11     System.out.println(a == c);    //FALSE!
12
13     // Even the assignment operator
14     // can get trivially confused...
15     Integer i = 0;
16     int j = i;
17
18     Integer k = null;
19     j = k;  // Exception in thread "main"
20             //java.lang.NullPointerException
21   }
22 }
```

```
1  console.log(1 == 1)                                  //true
2  console.log("JavaScript" == "JavaScript");           //true
3
4  console.log(1 == true);                              //true
5  console.log(0 == false);                             //true
6  console.log("" == false);                            //true
7  console.log("0" == false);                           //true
8
```

In JavaScript:

== tests for "truthy" equality

In other words, == does type coercion

```
1  console.log(1 == 1)                                  //true
2  console.log("JavaScript" == "JavaScript");           //true
3
4  console.log(1 == true);                              //true
5  console.log(0 == false);                             //true
6  console.log("" == false);                            //true
7  console.log("0" == false);                           //true
8
9  console.log("true" == true);                         //FALSE!
10 console.log(false == "false");                       //FALSE!
11
12 console.log(0 == "0");                               //true
13 console.log(0 == "");                                //true
14 console.log("" == "0");                              //FALSE!
15
```

In JavaScript:

== tests for "truthy" equality

In other words, == does type coercion

```
1  console.log(1 == 1)                                  //true
2  console.log("JavaScript" == "JavaScript");           //true
3
4  console.log(1 == true);                              //true
5  console.log(0 == false);                             //true
6  console.log("" == false);                            //true
7  console.log("0" == false);                           //true
8
9  console.log("true" == true);                         //FALSE!
10 console.log(false == "false");                       //FALSE!
11
12 console.log(0 == "0");                               //true
13 console.log(0 == "");                                //true
14 console.log("" == "0");                              //FALSE!
15
16 var lang = "JavaScript"
17 var anotherLang = new String("JavaScript")
18 console.log(lang == anotherLang);                     //true
```

In JavaScript:

== tests for "truthy" equality

In other words, == does type coercion

(Now do you believe me?)

You are better off
ignoring types
in JavaScript...

In JavaScript:

=== tests for "strict" equality

In other words, NO type coercion

```
1  console.log(1 === 1)                                  //true
2  console.log("JavaScript" === "JavaScript");           //true
3
4  console.log(1 === true);                              //FALSE!
5  console.log(0 === false);                             //FALSE!
6  console.log("" === false);                            //FALSE!
7  console.log("0" === false);                           //FALSE!
8
9  console.log("true" === true);                         //FALSE!
10 console.log(false === "false");                       //FALSE!
11
12 console.log(0 === "0");                               //FALSE!
13 console.log(0 === "");                                //FALSE!
14 console.log("" === "0");                              //FALSE!
15
16 var lang = "JavaScript"
17 var anotherLang = new String("JavaScript")
18 console.log(lang === anotherLang);                     //FALSE!
```

History

Strong vs. Weak Typing

Block vs. Function scope

Object-oriented vs. Functional

Compiling vs. Minifying

Block Scope (Java):

```java
public class JavaScope{
  public static void main(String[] args) {
    String name = "Scott";
    for(int i=0;i<3;i++){
      String message = name + " ate " + i + "cookies";
      System.out.println(message);
    }

    // Will this work?
    System.out.println(i);
    System.out.println(message);
  }
}
```

```
$ javac JavaScope.java
JavaScope.java:10: cannot find symbol
symbol  : variable i
location: class JavaScope
    System.out.println(i);
                       ^
JavaScope.java:11: cannot find symbol
symbol  : variable message
location: class JavaScope
    System.out.println(message);
                       ^
2 errors
```

Is there Global scope in Java?

– No

Class Scope (Java):

```
1  public class JavaScope2{
2    String name = "Scott";
3
4    public static void main(String[] args) {
5      JavaScope2 me = new JavaScope2();
6      me.sayHello();
7    }
8
9    public void sayHello(){
10     System.out.println("Hello " + name);
11     String name = "Venkat";
12     System.out.println("Hello " + name);
13   }
14 }
```

shadowed instance variable

```
$ java JavaScope2
  Hello Scott
  Hello Venkat
```

Can you shadow local variable in Java?

```
1  public class JavaScope{
2    public static void main(String[] args) {
3      String name = "Scott";
4      for(int i=0;i<3;i++){
5        // Will this work?
6        String name = "Venkat";
7        String message = name + " ate " + i + "cookies";
8        System.out.println(message);
9      }
10   }
11 }
```

```
$ javac JavaScope.java
JavaScope.java:6: name is already
defined in main(java.lang.String[])
      String name = "Venkat";
             ^
1 error
```

In Java:

- block scope
- default is private

In JavaScript:

???

Pathological Globalness (JavaScript)

Q: Which variables are global?

```
<script>
  var name = "Scott"
  city = "Broomfield"

  var checkCity = function(c){
    if(c == "Broomfield"){
      state = "CO"
    }
  }
</script>
```

A: All except **c**

(Anything declared outside of a function is global)

(Anything declared without var is global)

Function Scope (JavaScript)

Q: Which variables are global?

```
var names = ["Scott", "Venkat"]

var maxLength = function(n){
  var max = 0
  for(i=0; i<n.length; i++){
    if(n[i].length > max){
      max = n[i].length
    }
  }
  return max
}
```

*A: All except **n** and **max***

(Only declared with var + inside of a function is private)

Function Scope (cont.)

Solution: IIFE (Immediately-Invoked Function Expression)

```
(function(){
  var name = "Scott"
  var hello = function(){ ... }
})(this);
```

http://benalman.com/news/2010/11/immediately-invoked-function-expression/

Variable Hoisting (JavaScript)

Q: What does hello() print?

```
var name = "Scott"

var hello = function(){
  console.log("Hi " + name);
  var name = "Venkat"
  console.log("Hi " + name);
}

hello()
```

A:

```
$ node hello.js
  Hi undefined
  Hi Venkat
```

Variable Hoisting (cont.)

```
var name = "Scott"

var hello = function(){
   console.log("Hi " + name);
   var name = "Venkat"
}
```

```
var name = "Scott"

var hello = function(){
   var name // all vars are "hoisted"
            // to the top of the f()
   console.log("Hi " + name);
   name = "Venkat"
}
```

In Java:

- block scope
- default is private

In JavaScript:

- function scope
- default is global

History

Strong vs. Weak Typing

Block vs. Function scope

Object-oriented vs. Functional

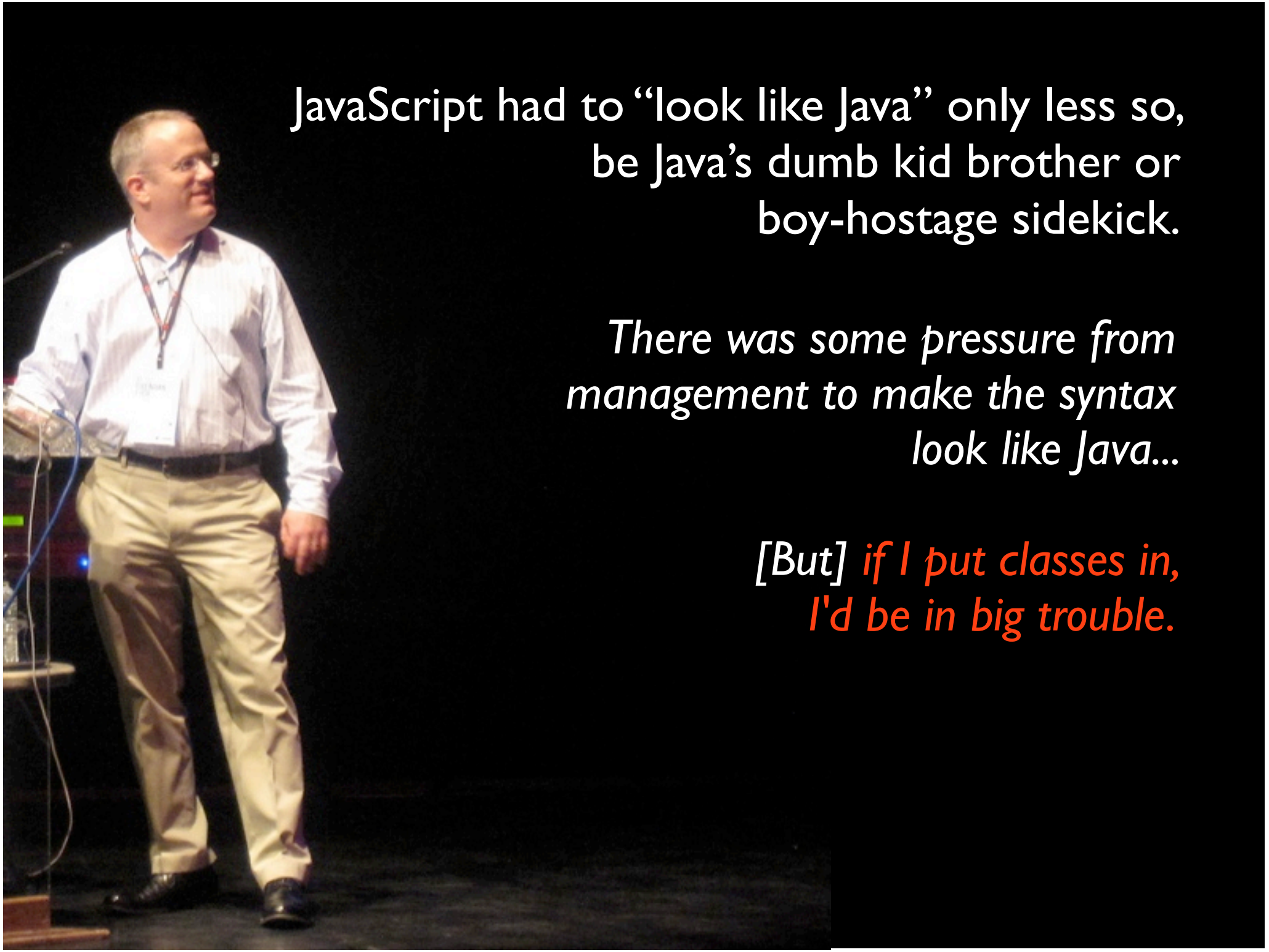
Compiling vs. Minifying

Java is Object-Oriented

```
1  public class Person{
2    private String name;
3    private String city;
4
5    public Person(String name, String city){
6      this.name = name;
7      this.city = city;
8    }
9
10   public String getName(){ return this.name; }
11   public void setName(String name){ this.name = name; }
12
13   public String getCity(){ return this.city; }
14   public void setCity(String city){ this.city = city; }
15
16   public String toString(){
17     return name + " lives in " + city;
18   }
19 }
```

Java is Object-Oriented (cont.)

```
1 public class PersonTest{
2   public static void main(String[] args) {
3     Person p = new Person("Scott", "Broomfield");
4     System.out.println(p);
5   }
6 }
```


JavaScript had to "look like Java" only less so,
be Java's dumb kid brother or
boy-hostage sidekick.
There was some pressure from
management to make the syntax
look like Java...
[But] if I put classes in,
I'd be in big trouble.

COMPUTERWORLD
THE VOICE OF IT MANAGEMENT

The A-Z of Programming Languages: JavaScript

Brendan Eich created JavaScript in 1995 with the aim to provide a "glue language" for Web designers and part grown to become one of the most widely used languages on the planet.

Naomi Hamilton (Computerworld) | 31 July, 2008 21:04

I joined Netscape on 4 April 1995, with the goal of embedding the Scheme programming language, or something like it, into Netscape's browser.

Java was in some ways a negative influence.

I didn't want to have anything "classy."

So I swerved from that and it caused me to look at Self and do prototypes.

Self (programming language)

From Wikipedia, the free encyclopedia

Self is an object-oriented programming language based on the concept of *prototypes*. Essentially an extreme dialect of Smalltalk, it was used mainly as an experimental test system for language design in the 1980s and 1990s. In 2006, Self was still being developed as part of the Klein project, which was a Self virtual machine written fully in Self. The latest version is 4.4, released in July 2010.

Prototype-based programming

From Wikipedia, the free encyclopedia

Prototype-based programming is a style of object-oriented programming in which classes are not present, and behavior reuse (known as inheritance in class-based languages) is performed via a process of cloning existing objects that serve as prototypes. This model can also be known as *classless*, *prototype-oriented* or *instance-based* programming. Delegation is the language feature that supports prototype-based programming.

JavaScript has Object Literals

```
var person = {
   name: "Scott Davis",
   city: "Broomfield"
}

var person2 = {
   name: "Venkat Subramaniam",
   state: "CO"
}

console.log(person.name);
```

JSON == “JavaScript Object Notation”

JavaScript has Object Literals (cont.)

```
// DANGER!!! DON'T DO THIS!
// toString() is needlessly duplicated for each instance

var person = {
   name: "Scott Davis",
   city: "Broomfield",
   toString: function(){
     return this.name + " lives in " + this.city
   }
}

var person2 = {
   name: "Venkat Subramaniam",
   city: "Broomfield",
   toString: function(){
     return this.name + " lives in " + this.city
   }
}

console.log(person.toString());
```

```
// NOTE: Person isn't an object literal
// Person is a function
// (using prototype to share a function instance)

var Person = function(){
   this.name
   this.city
}

Person.prototype.toString = function(){
   return this.name + " lives in " + this.city
}

var person = new Person()
person.name = "Scott Davis"
person.city = "Broomfield"

console.log(person.toString());
```

Objects in JavaScript are Dynamic (i.e. They are mutable at runtime...)

```
var person = new Person()
person.name = "Scott Davis"
person.city = "Broomfield"
console.log(person.toString());


var person2 = new Person()
person2.name = "Venkat Subramaniam"
person2.city = "Broomfield"
person2.state = "CO"
person2.toString = function(){
  return this.name + " won't tell you where he lives"
}
console.log(person2.toString());
```

History

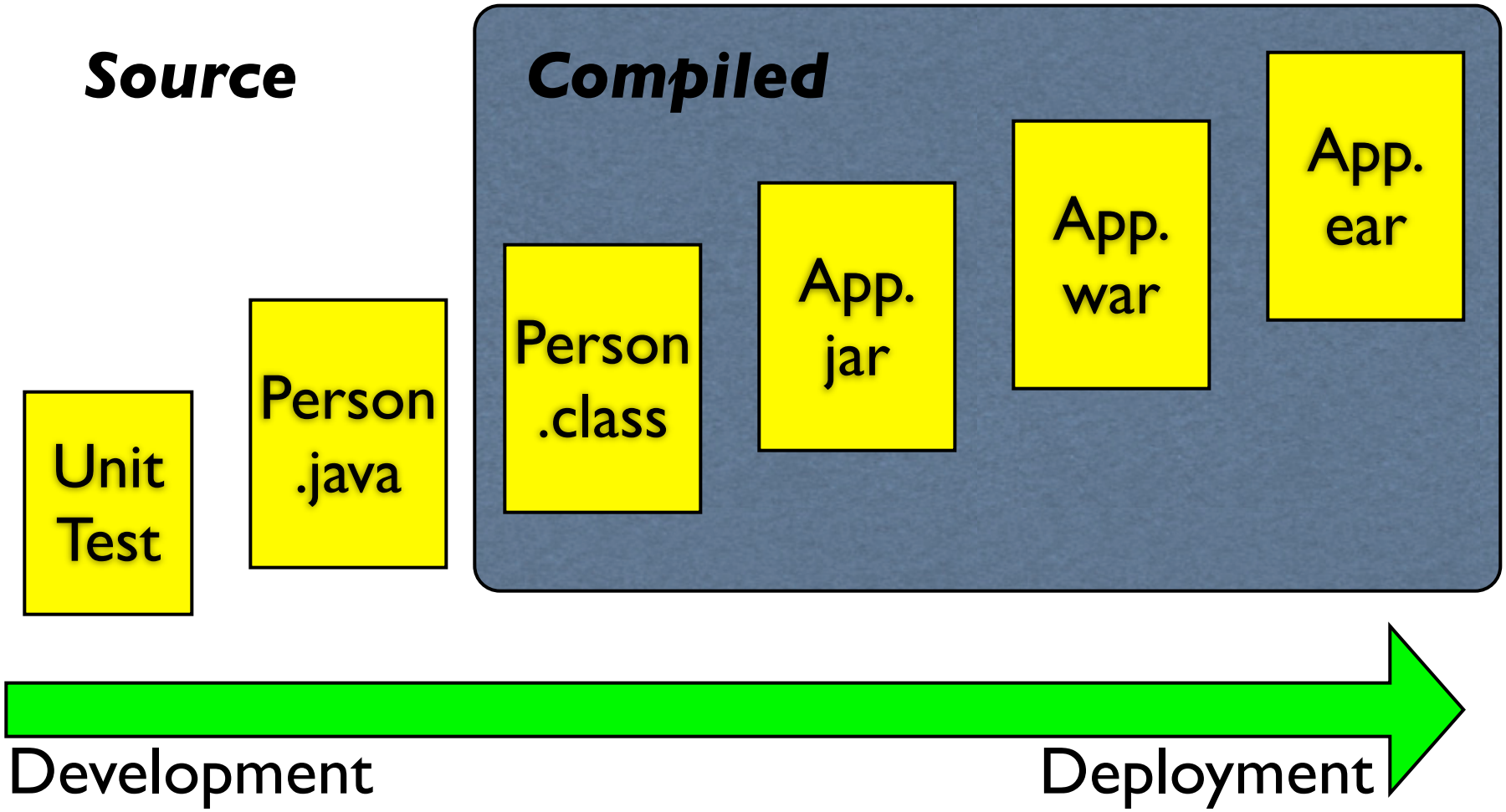
Strong vs. Weak Typing

Block vs. Function scope

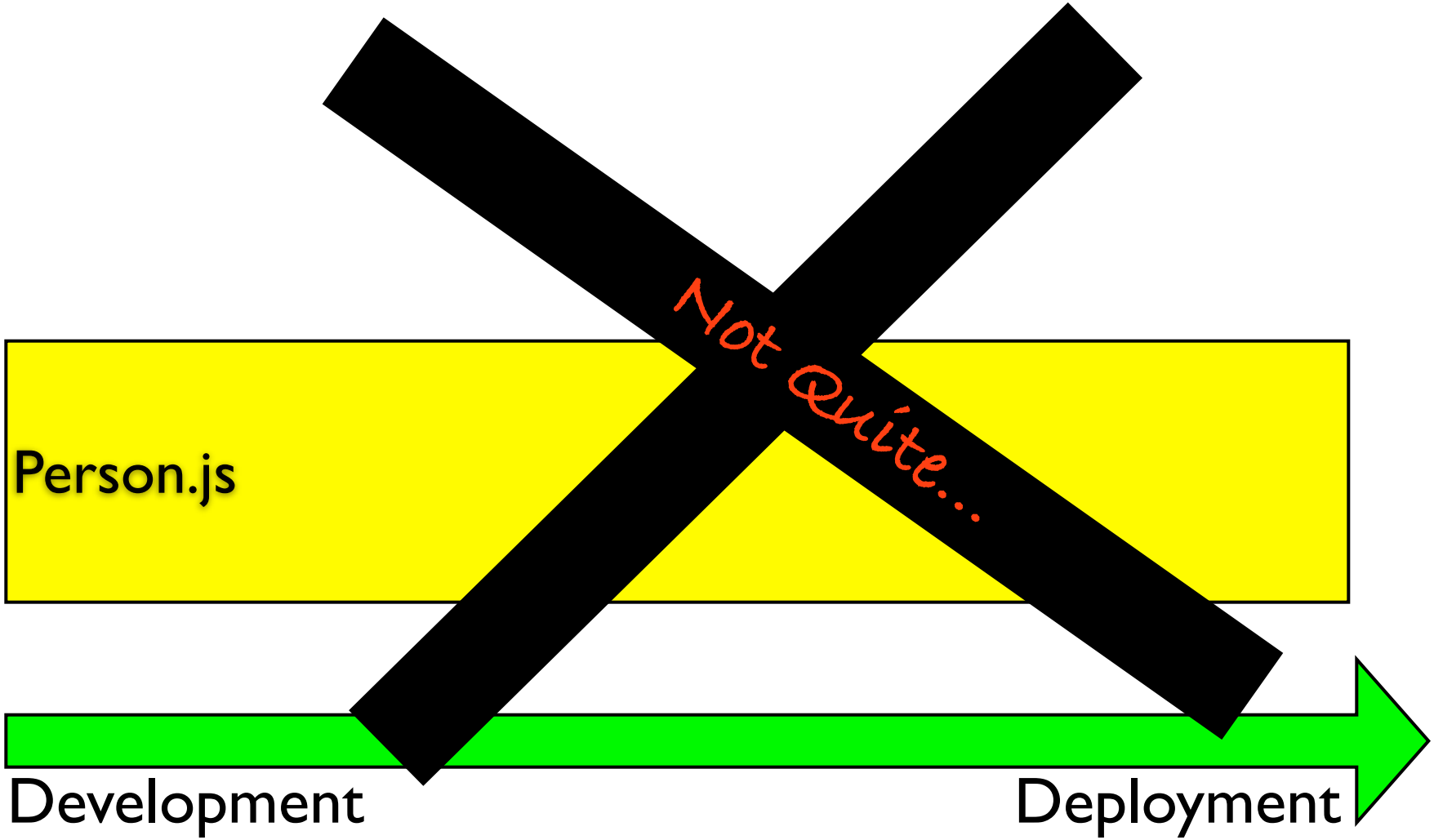
Object-oriented vs. Functional

Compiling vs. Minifying

Java (Compiled)



JavaScript (Interpreted)



Java compiler strips out:

whitespace

carriage returns / line-feeds

comments

unused imports

unreachable code branches

JavaScript minifier strips out:

whitespace

carriage returns / line-feeds

comments

~~unused imports~~

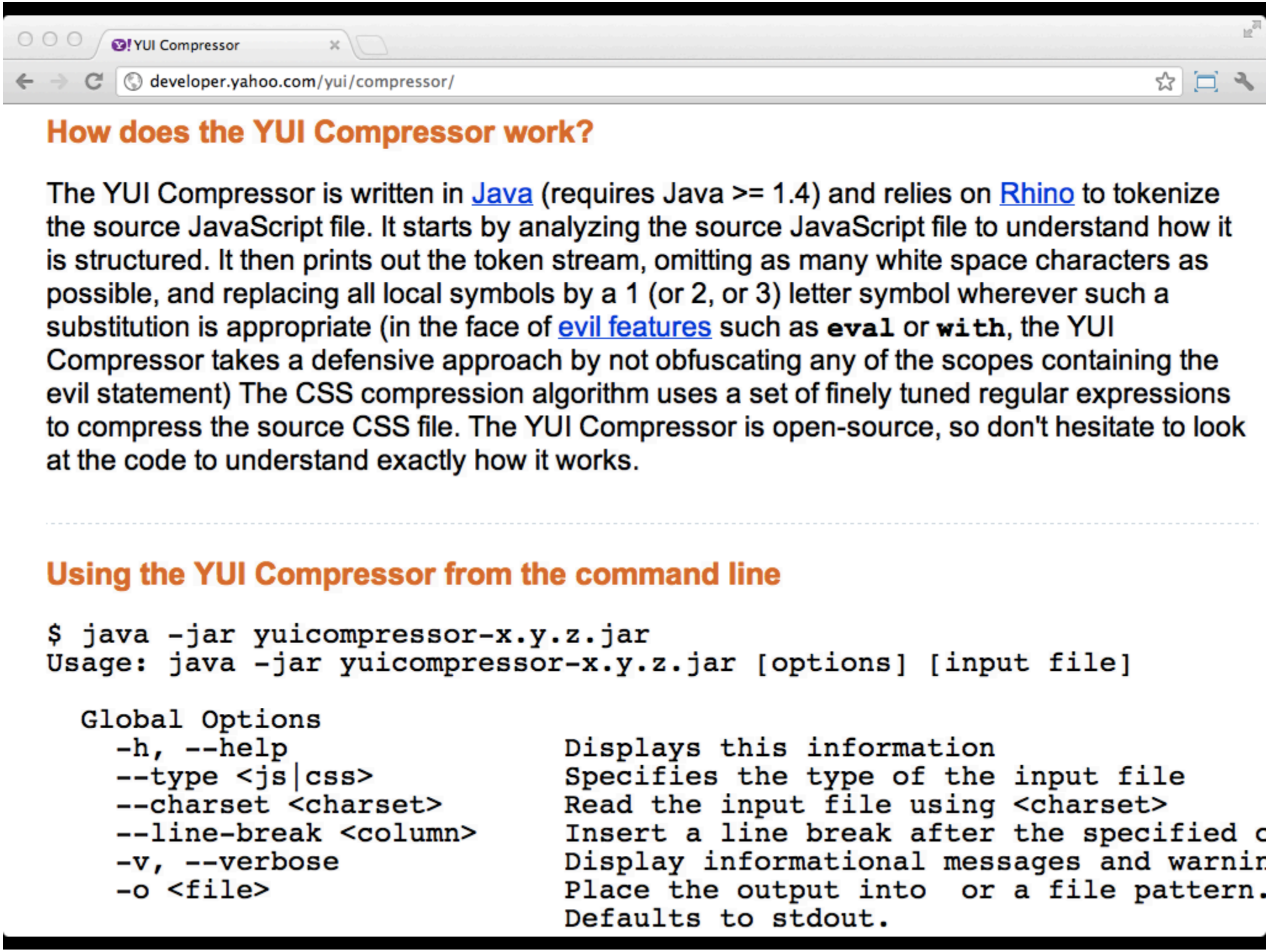
~~unreachable code branches~~

Before:
(253 kb)

```
/*!
 * jQuery JavaScript Library v1.7.2
 * http://jquery.com/
 *
 * Copyright 2011, John Resig
 * Dual licensed under the MIT or GPL Version 2 licenses.
 * http://jquery.org/license
 *
 * Includes Sizzle.js
 * http://sizzlejs.com/
 * Copyright 2011, The Dojo Foundation
 * Released under the MIT, BSD, and GPL Licenses.
 *
 * Date: Wed Mar 21 12:46:34 2012 -0700
 */
(function( window, undefined ) {

// Use the correct document accordingly with window argument (sandbox)
var document = window.document,
  navigator = window.navigator,
  location = window.location;
var jQuery = (function() {
```

After: (95 kb)

```
/*! jQuery v1.7.2 jquery.com | jquery.org/license */
(function(a,b){function cy(a){return f.isWindow(a)?a:a.nodeType===9?a.d
a){var b=F.exec(a);b&&(b[1]=(b[1]||"").toLowerCase(),b[3]=b[3]&&new Reg
.clean(arguments);a.push.apply(a,this.toArray());return this.pushStack(
```



How does the YUI Compressor work?

The YUI Compressor is written in Java (requires Java >= 1.4) and relies on Rhino to tokenize the source JavaScript file. It starts by analyzing the source JavaScript file to understand how it is structured. It then prints out the token stream, omitting as many white space characters as possible, and replacing all local symbols by a 1 (or 2, or 3) letter symbol wherever such a substitution is appropriate (in the face of evil features such as **`eval`** or **`with`**, the YUI Compressor takes a defensive approach by not obfuscating any of the scopes containing the evil statement) The CSS compression algorithm uses a set of finely tuned regular expressions to compress the source CSS file. The YUI Compressor is open-source, so don't hesitate to look at the code to understand exactly how it works.

Using the YUI Compressor from the command line

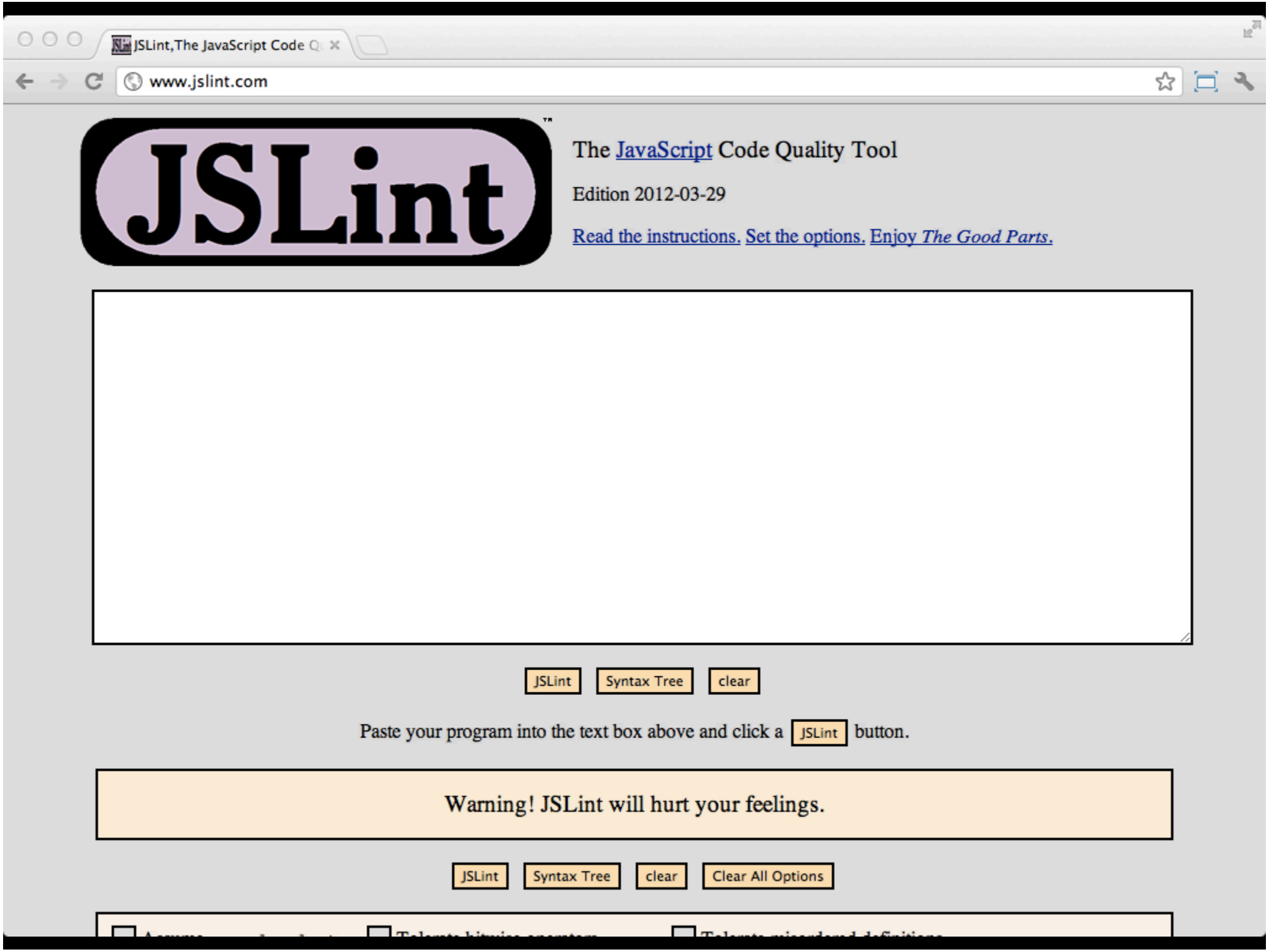
```
$ java -jar yuicompressor-x.y.z.jar
Usage: java -jar yuicompressor-x.y.z.jar [options] [input file]

  Global Options
    -h, --help                  Displays this information
    --type <js|css>             Specifies the type of the input file
    --charset <charset>         Read the input file using <charset>
    --line-break <column>       Insert a line break after the specified c
    -v, --verbose               Display informational messages and warnir
    -o <file>                   Place the output into  or a file pattern.
                                Defaults to stdout.
```

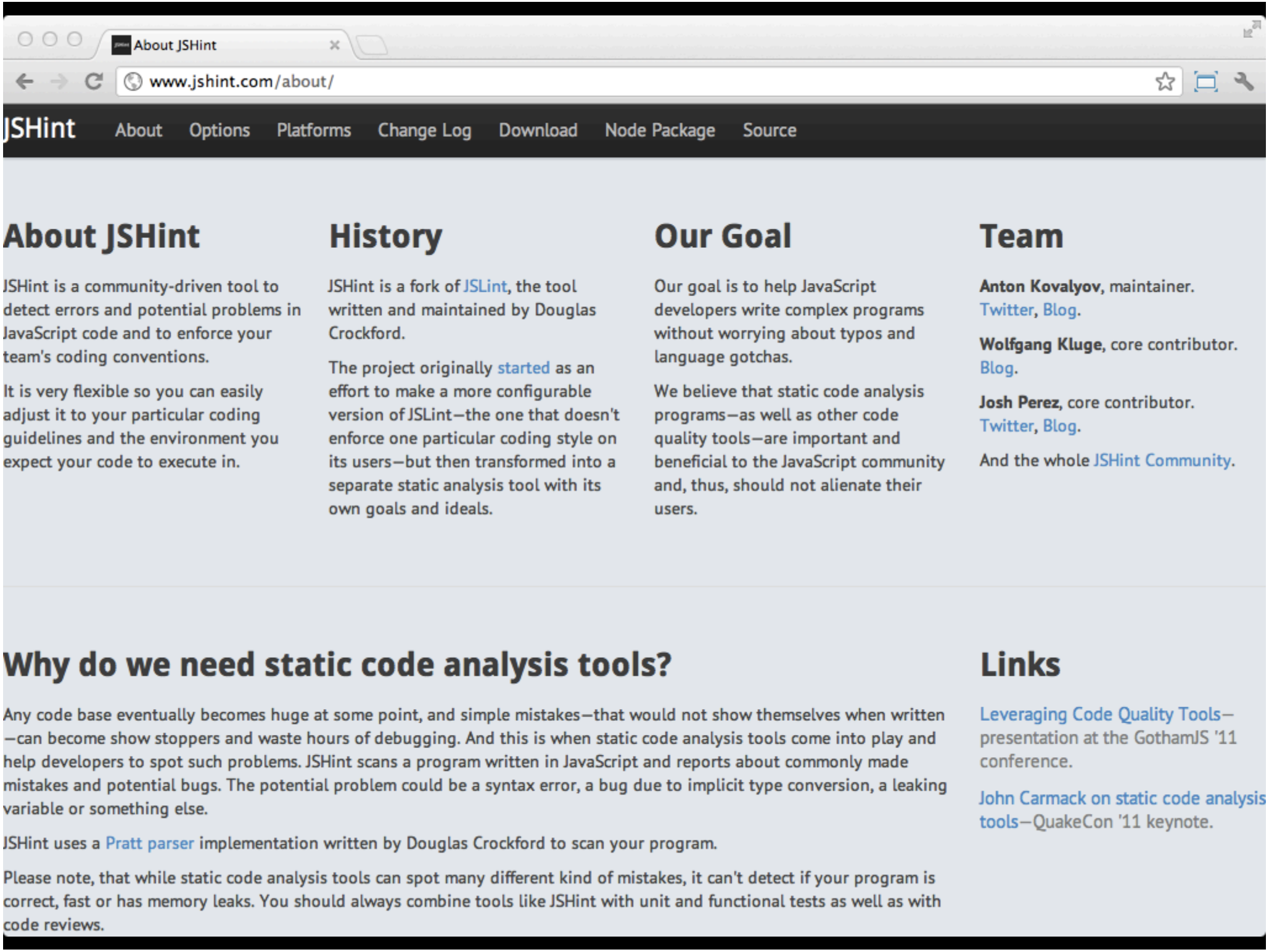
Java compiler:

Verifies syntactic correctness

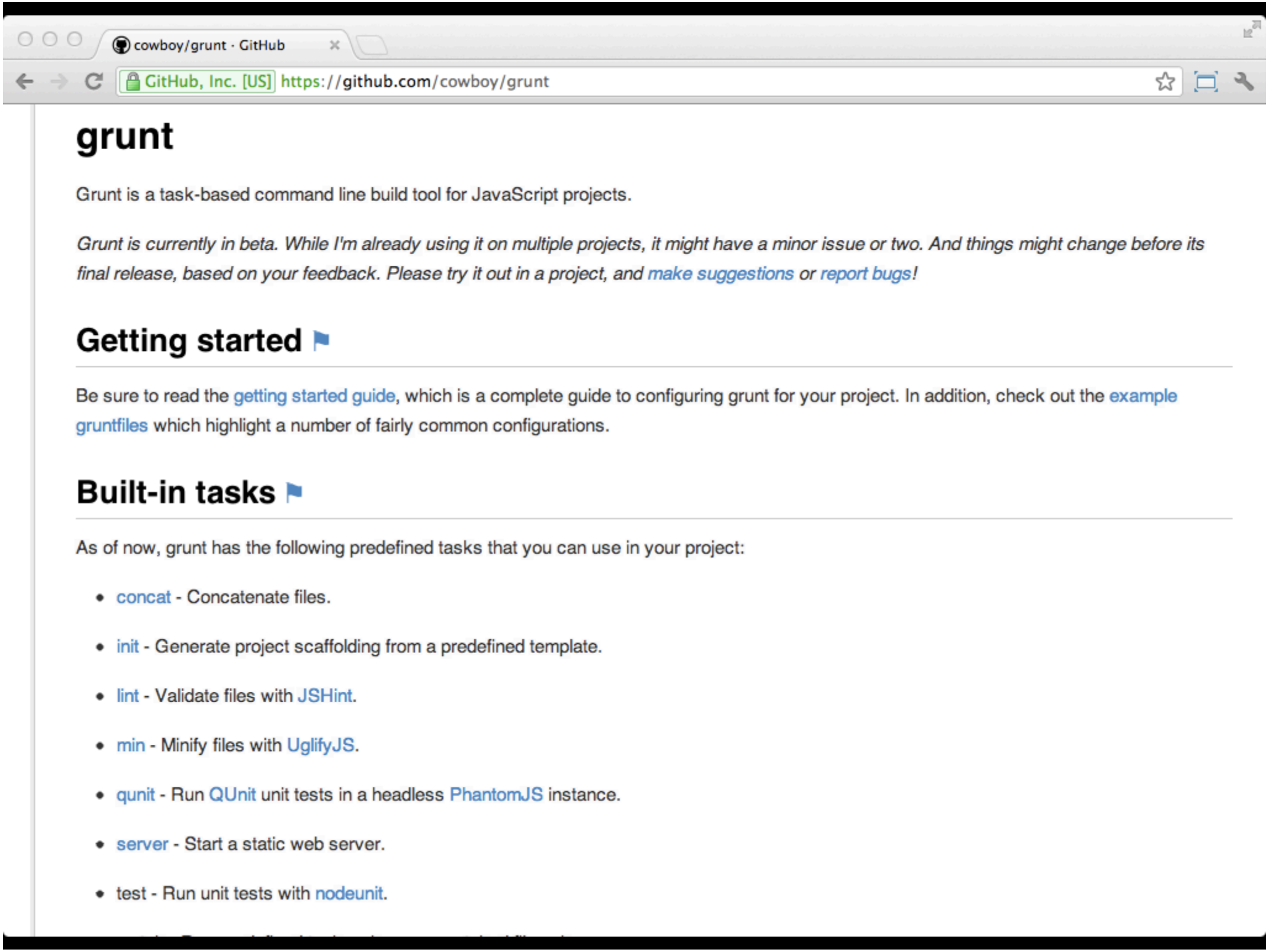
JavaScript lint:

Verifies syntactic correctness


JSLint,The JavaScript Code Q
www.jslint.com
JSLint
The JavaScript Code Quality Tool
Edition 2012-03-29
Read the instructions. Set the options. Enjoy The Good Parts.
JSLint
Syntax Tree
clear
Paste your program into the text box above and click a JSLint button.
Warning! JSLint will hurt your feelings.
JSLint
Syntax Tree
clear
Clear All Options


About JSHint
www.jshint.com/about/
JSHint
About
Options
Platforms
Change Log
Download
Node Package
Source
About JSHint
JSHint is a community-driven tool to detect errors and potential problems in JavaScript code and to enforce your team's coding conventions.
It is very flexible so you can easily adjust it to your particular coding guidelines and the environment you expect your code to execute in.
History
JSHint is a fork of JSLint, the tool written and maintained by Douglas Crockford.
The project originally started as an effort to make a more configurable version of JSLint—the one that doesn't enforce one particular coding style on its users—but then transformed into a separate static analysis tool with its own goals and ideals.
Our Goal
Our goal is to help JavaScript developers write complex programs without worrying about typos and language gotchas.
We believe that static code analysis programs—as well as other code quality tools—are important and beneficial to the JavaScript community and, thus, should not alienate their users.
Team
Anton Kovalyov, maintainer. Twitter, Blog.
Wolfgang Kluge, core contributor. Blog.
Josh Perez, core contributor. Twitter, Blog.
And the whole JSHint Community.
Why do we need static code analysis tools?
Any code base eventually becomes huge at some point, and simple mistakes—that would not show themselves when written—can become show stoppers and waste hours of debugging. And this is when static code analysis tools come into play and help developers to spot such problems. JSHint scans a program written in JavaScript and reports about commonly made mistakes and potential bugs. The potential problem could be a syntax error, a bug due to implicit type conversion, a leaking variable or something else.
JSHint uses a Pratt parser implementation written by Douglas Crockford to scan your program.
Please note, that while static code analysis tools can spot many different kind of mistakes, it can't detect if your program is correct, fast or has memory leaks. You should always combine tools like JSHint with unit and functional tests as well as with code reviews.
Links
Leveraging Code Quality Tools—presentation at the GothamJS '11 conference.
John Carmack on static code analysis tools—QuakeCon '11 keynote.

build script



grunt

Grunt is a task-based command line build tool for JavaScript projects.

Grunt is currently in beta. While I'm already using it on multiple projects, it might have a minor issue or two. And things might change before its final release, based on your feedback. Please try it out in a project, and make suggestions or report bugs!

Getting started

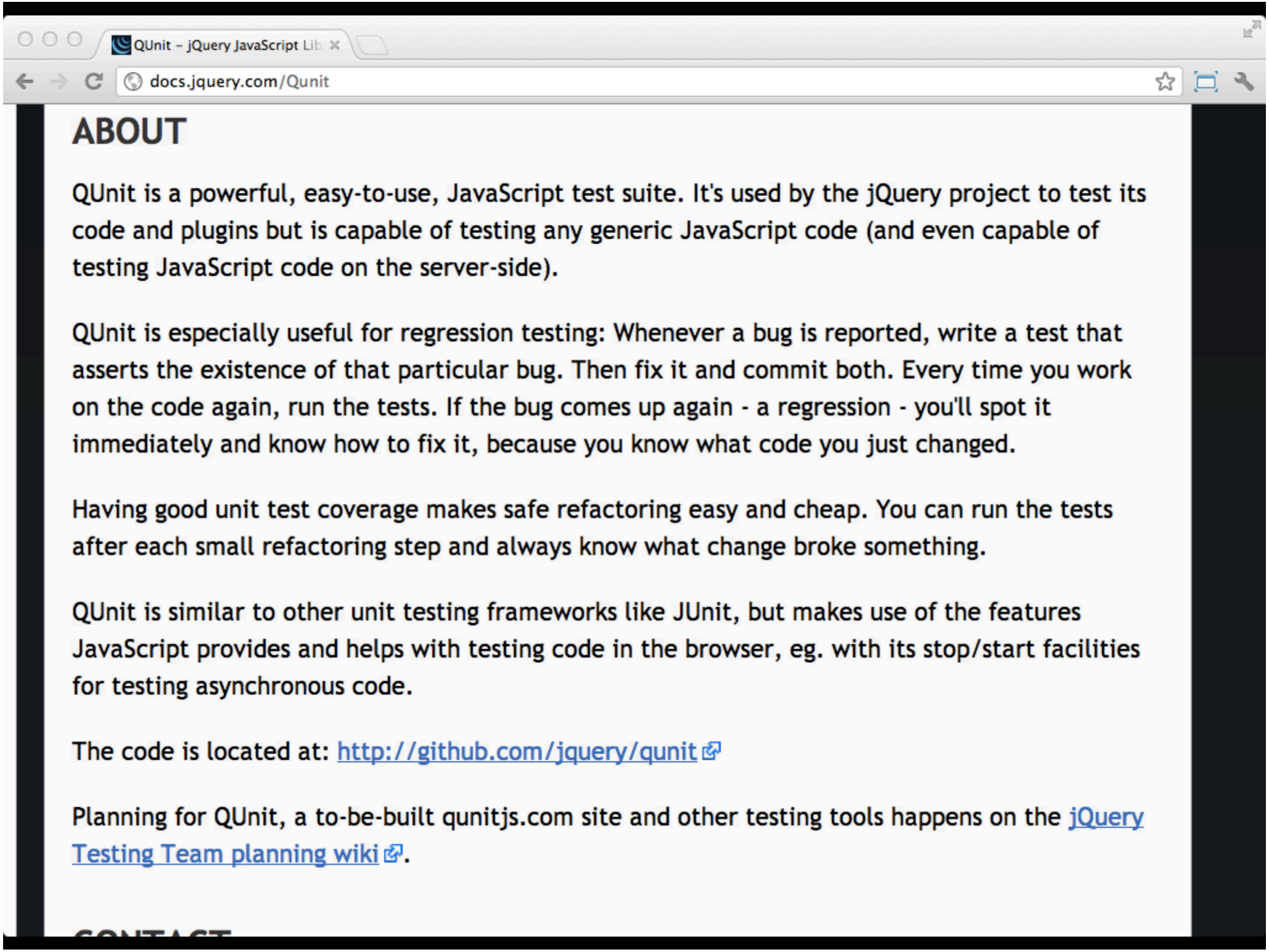
Be sure to read the getting started guide, which is a complete guide to configuring grunt for your project. In addition, check out the example gruntfiles which highlight a number of fairly common configurations.

Built-in tasks

As of now, grunt has the following predefined tasks that you can use in your project:

- concat - Concatenate files.
- init - Generate project scaffolding from a predefined template.
- lint - Validate files with JSHint.
- min - Minify files with UglifyJS.
- qunit - Run QUnit unit tests in a headless PhantomJS instance.
- server - Start a static web server.
- test - Run unit tests with nodeunit.

Unit Test



ABOUT

QUnit is a powerful, easy-to-use, JavaScript test suite. It's used by the jQuery project to test its code and plugins but is capable of testing any generic JavaScript code (and even capable of testing JavaScript code on the server-side).

QUnit is especially useful for regression testing: Whenever a bug is reported, write a test that asserts the existence of that particular bug. Then fix it and commit both. Every time you work on the code again, run the tests. If the bug comes up again - a regression - you'll spot it immediately and know how to fix it, because you know what code you just changed.

Having good unit test coverage makes safe refactoring easy and cheap. You can run the tests after each small refactoring step and always know what change broke something.

QUnit is similar to other unit testing frameworks like JUnit, but makes use of the features JavaScript provides and helps with testing code in the browser, eg. with its stop/start facilities for testing asynchronous code.

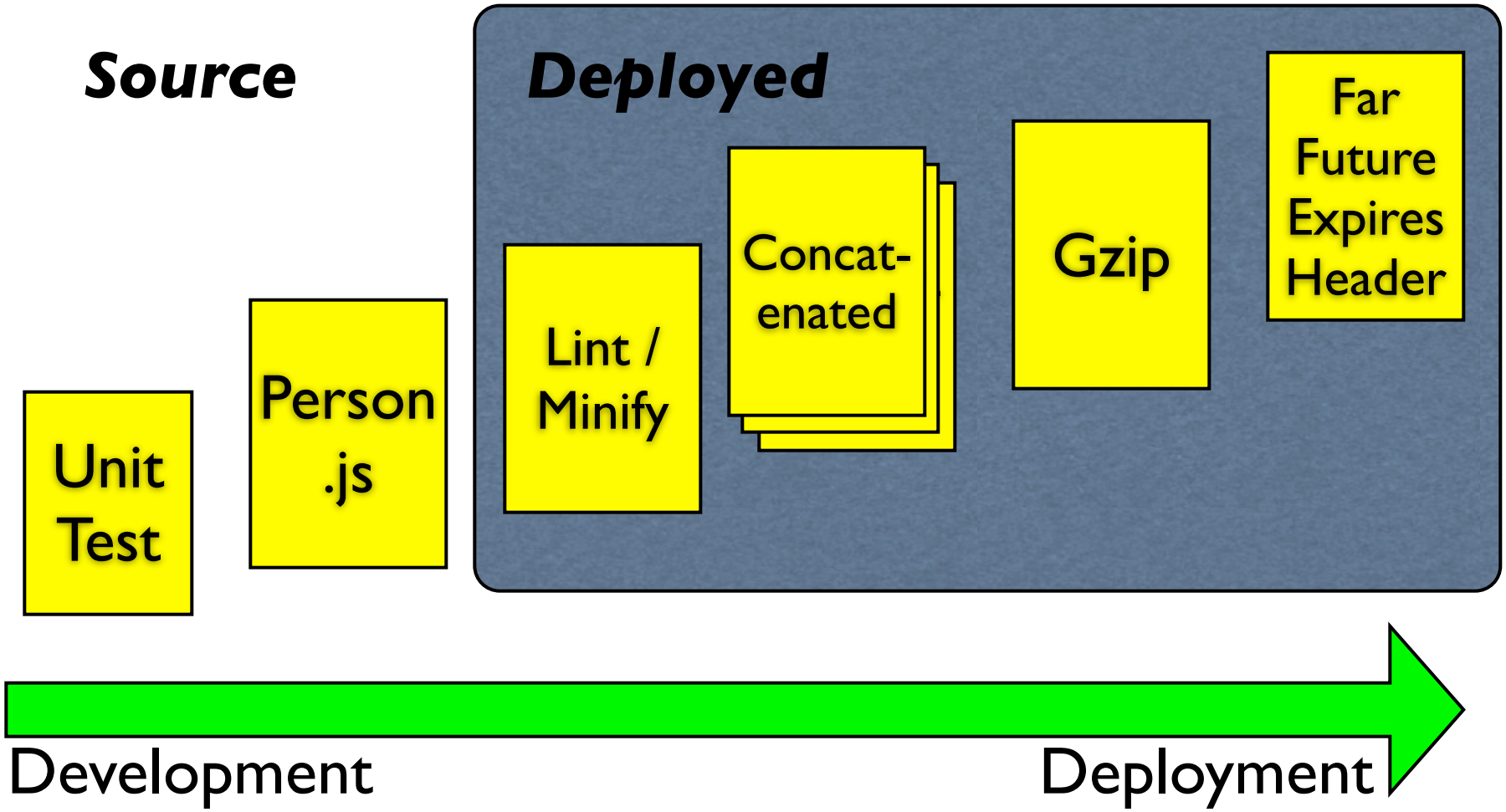
The code is located at: http://github.com/jquery/qunit

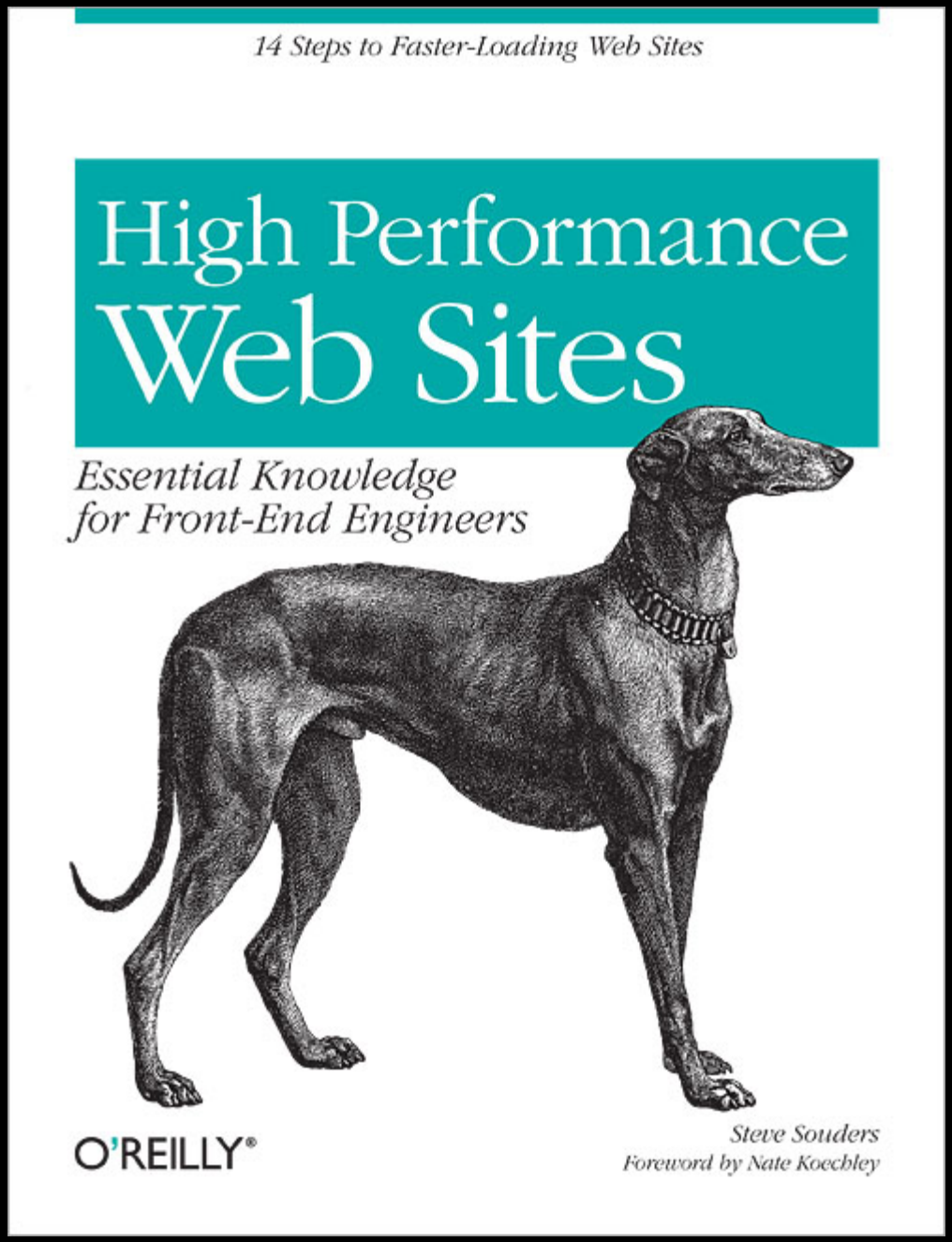
Planning for QUnit, a to-be-built qunitjs.com site and other testing tools happens on the jQuery Testing Team planning wiki.

CONTACT


Jasmine: BDD for your JavaSc
pivotal.github.com/jasmine/
Jasmine
Fork me on GitHub
BDD for JavaScript
Jasmine is a behavior-driven development framework for testing your JavaScript code. It does not depend on any other JavaScript frameworks. It does not require a DOM. And it has a clean, obvious syntax so that you can easily write tests.
describe("Jasmine", function() {
it("makes testing JavaScript awesome!", function() {
expect(yourCode).toBeLotsBetter();
});
});
Jasmine can be run anywhere you can execute JavaScript: a static web page, your continuous integration environment, or server-side environments like Node.js .
Find out more in the documentation

JavaScript (Interpreted)




14 Steps to Faster-Loading Web Sites
High Performance Web Sites
Essential Knowledge for Front-End Engineers
O'REILLY®
Steve Souders
Foreword by Nate Koechley

History

Strong vs. Weak Typing

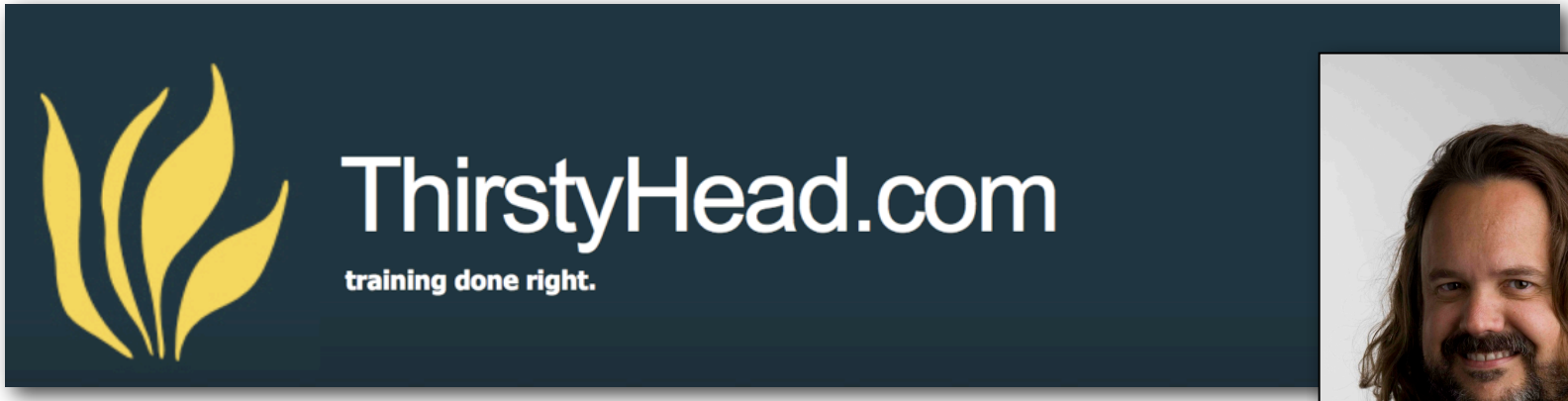
Block vs. Function scope

Object-oriented vs. Functional

Compiling vs. Minifying

JavaScript for Java Developers

Scott Davis, ThirstyHead.com



scott@thirstyhead.com

Questions?

Thanks for your time.

Image Credits

All images courtesy of http://morgueFile.com

except:

http://upload.wikimedia.org/wikinews/en/c/c2/Robert_Cailliau_OnDesk2.jpg
http://www.flickr.com/photos/gen/4774151517/
http://www.flickr.com/photos/strandell/4693852906/
http://www.flickr.com/photos/dburka/2836211647/
http://www.flickr.com/photos/stephaniehobson/4692992276/
http://commons.wikimedia.org/wiki/File:A_metal_bucket.jpg
http://commons.wikimedia.org/wiki/File:Coffee_pot_3.jpg