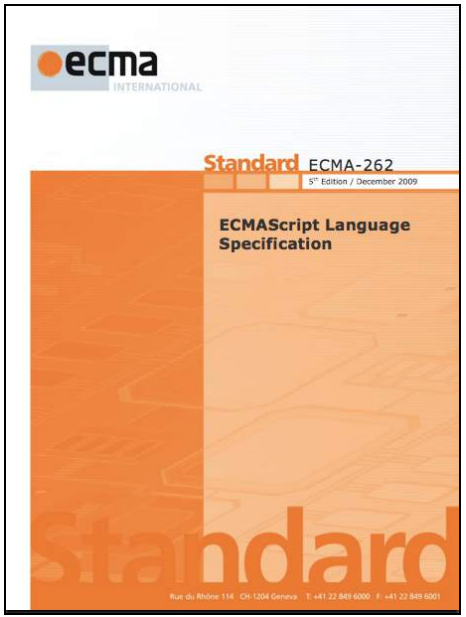
ECMAScript 2015: A Better JavaScript for the Ambient Computing Era

Talk, by Allen Wirfs-Brock
Mozilla Research Fellow
Project Editor, Ecma-262 (The JavaScript Standard)
@awbjs

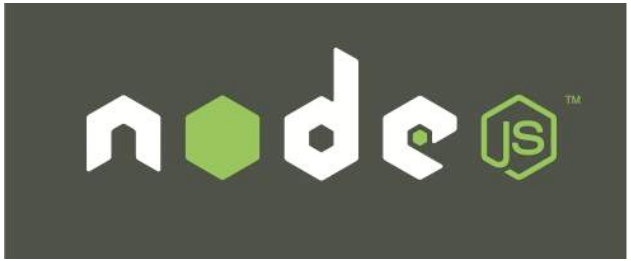
The Ambient Computing Era

Each Computing Era has had Canonical Programming Languages

- Corporate Computing Era – COBOL/Fortran
- Personal Computing Era – C/C++ family
- Ambient Computing Era – JavaScript ??



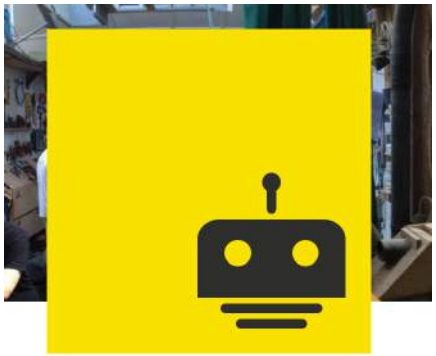
Not Just In Web Browsers



http://nodejs.org/

Firefox OS

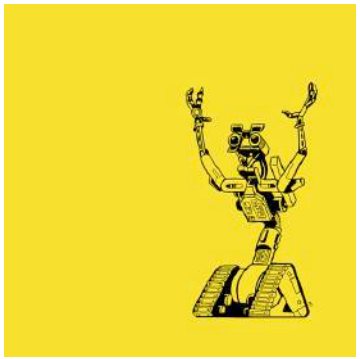
https://developer.mozilla.org/en-US/Firefox_OS



NodeBots

Robots powered by JavaScript

http://nodebots.io/



Johnny-Five

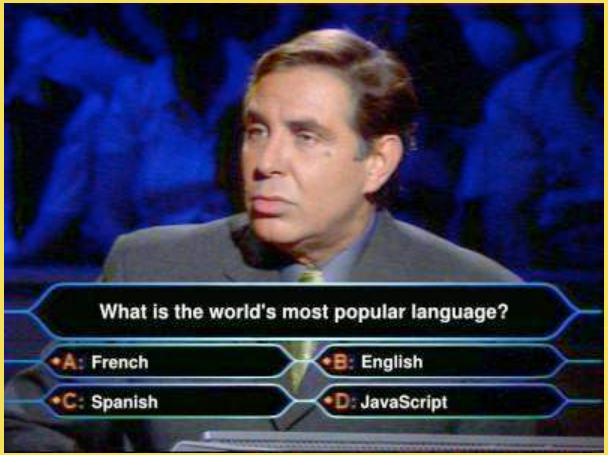
https://github.com/rwaldron/johnny-five

Why JavaScript?
Because "Worse is Better"

Dick Gabriel
http://www.dreamsongs.com/WorseIsBetter.html

The economics of ubiquity.

- ✓ It's already there
- ✓ Widest reach
- ✓ Lowest risk
- ✓ Write libraries and apps once
- ✓ Single knowledge and skill set

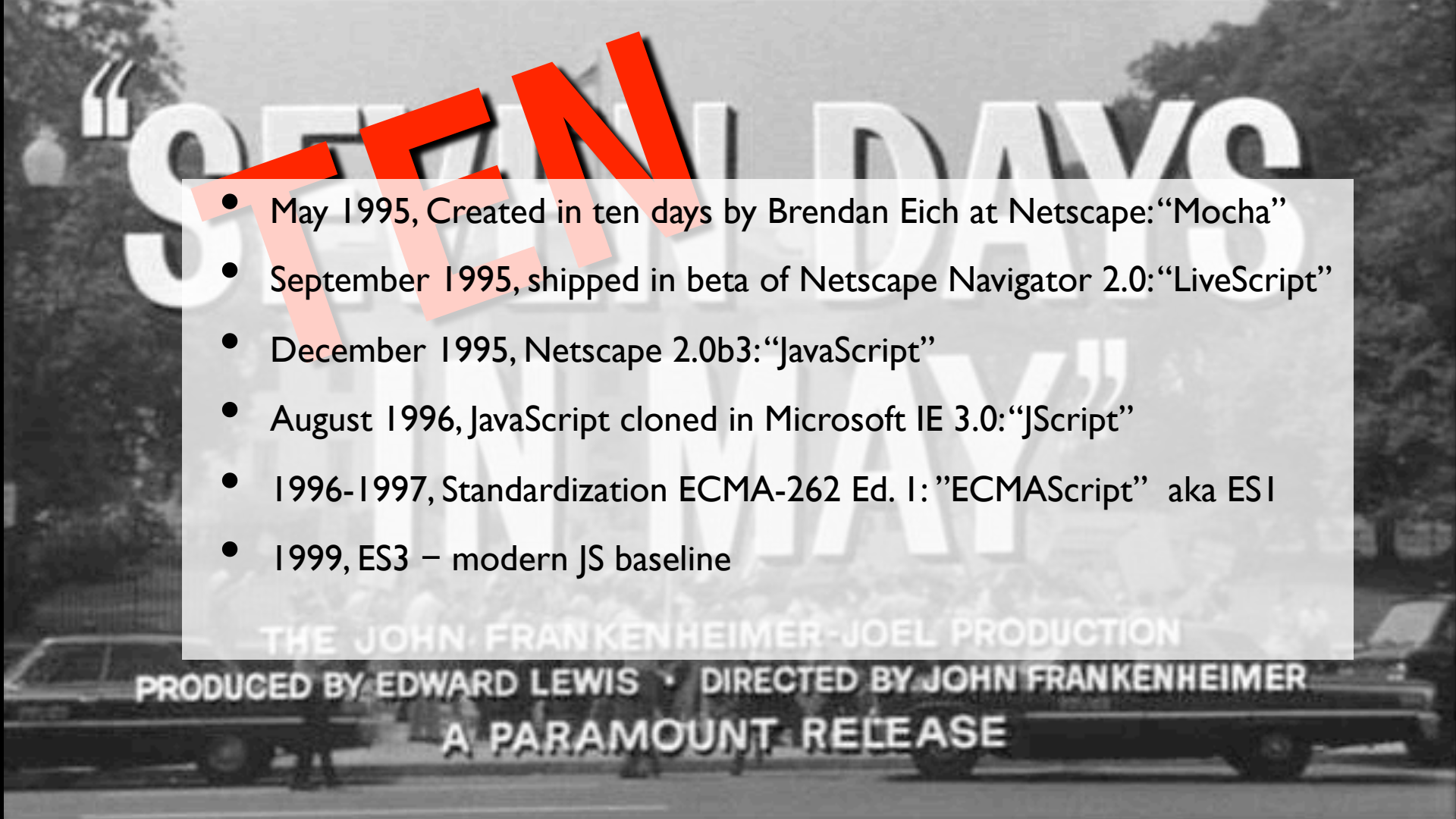


http://odetocode.com/Blogs/scott/archive/2009/03/18/signs-that-your-javascript-skills-need-updating.aspx

Is it even possible to replace it?

JS

JavaScript Early history

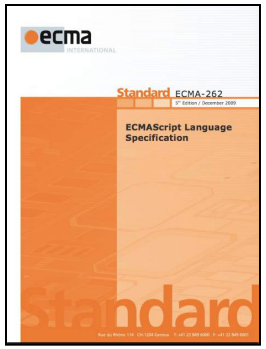


TEN

- May 1995, Created in ten days by Brendan Eich at Netscape: "Mocha"
- September 1995, shipped in beta of Netscape Navigator 2.0: "LiveScript"
- December 1995, Netscape 2.0b3: "JavaScript"
- August 1996, JavaScript cloned in Microsoft IE 3.0: "JScript"
- 1996-1997, Standardization ECMA-262 Ed. 1: "ECMAScript" aka ES1
- 1999, ES3 − modern JS baseline

What is ECMAScript?

- ECMAScript is the name of the international standard that defines the JavaScript programming language
- Developed by Technical Committee 39 (TC-39) of Ecma International
- Issued as a Ecma-262 and ISO/IEC 16262
- Not part of W3C



Google V8 | Mozilla SpiderMonkey | Microsoft Chakra | Webkit JSCore

JavaScript Implementations

Interoperability is TC-39's highest priority

- A detailed and highly prescriptive algorithmic specification
- Large, non-normative test suite for implementers

ecmascript test262

http://test262.ecmascript.org/

8.7.2 PutValue (V, W)

1. If Type(*V*) is not Reference, throw a **ReferenceError** exception.
2. Let *base* be the result of calling GetBase(*V*).
3. If IsUnresolvableReference(*V*), then
   a. If IsStrictReference(*V*) is **true**, then
      i. Throw **ReferenceError** exception.
   b. Call the [[Put]] internal method of the global object, passing GetReferencedName(*V*) for the property name, *W* for the value, and **false** for the *Throw* flag.
4. Else if IsPropertyReference(*V*), then
   a. If HasPrimitiveBase(*V*) is **false**, then let *put* be the [[Put]] internal method of *base*, otherwise let *put* be the special [[Put]] internal method defined below.
   b. Call the *put* internal method using *base* as its **this** value, and passing GetReferencedName(*V*) for the property name, *W* for the value, and IsStrictReference(*V*) for the *Throw* flag.
5. Else *base* must be a reference whose base is an environment record. So,
   a. Call the SetMutableBinding (10.2.1) concrete method of *base*, passing GetReferencedName(*V*), *W*, and IsStrictReference(*V*) as arguments.
6. Return.

The following [[Put]] internal method is used by PutValue when *V* is a property reference with a primitive base value. It is called using *base* as its **this** value and with property *P*, value *W*, and Boolean flag *Throw* as arguments. The following steps are taken:

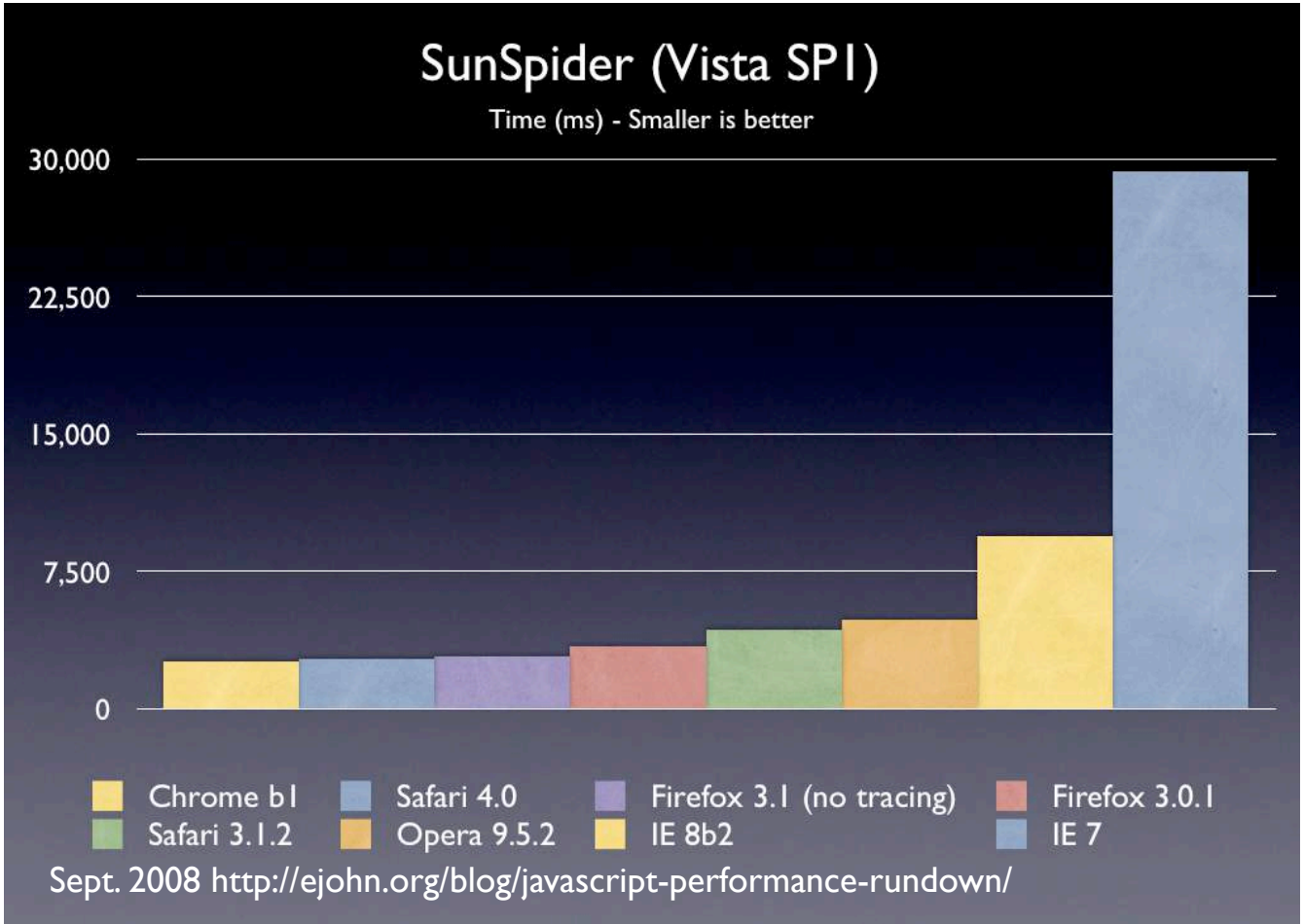
1. Let *O* be ToObject(*base*).
2. If the result of calling the [[CanPut]] internal method of *O* with argument *P* is **false**, then
   a. If *Throw* is **true**, then throw a **TypeError** exception.
   b. Else return.
3. Let *ownDesc* be the result of calling the [[GetOwnProperty]] internal method of *O* with argument *P*.
4. If IsDataDescriptor(*ownDesc*) is **true**, then
   a. If *Throw* is **true**, then throw a **TypeError** exception.
   b. Else return.
5. Let *desc* be the result of calling the [[GetProperty]] internal method of *O* with argument *P*. This may be either an own or inherited accessor property descriptor or an inherited data property descriptor.
6. If IsAccessorDescriptor(*desc*) is **true**, then
   a. Let *setter* be *desc*.[[Set]] (see 8.10) which cannot be **undefined**.
   b. Call the [[Call]] internal method of *setter* providing *base* as the **this** value and an argument list containing only *W*.
7. Else, this is a request to create an own property on the transient object *O*
   a. If *Throw* is **true**, then throw a **TypeError** exception.

ECMAScript: Troubled Adolescence

- 2000: ES4, attempt 1
- 2003-4: E4X, XML extensions for ECMAScript
- 2005-7: ES4, attempt 2
- 2008: ES4 abandoned
- 2009: ES5: “use strict”, JSON, Object.create, etc.

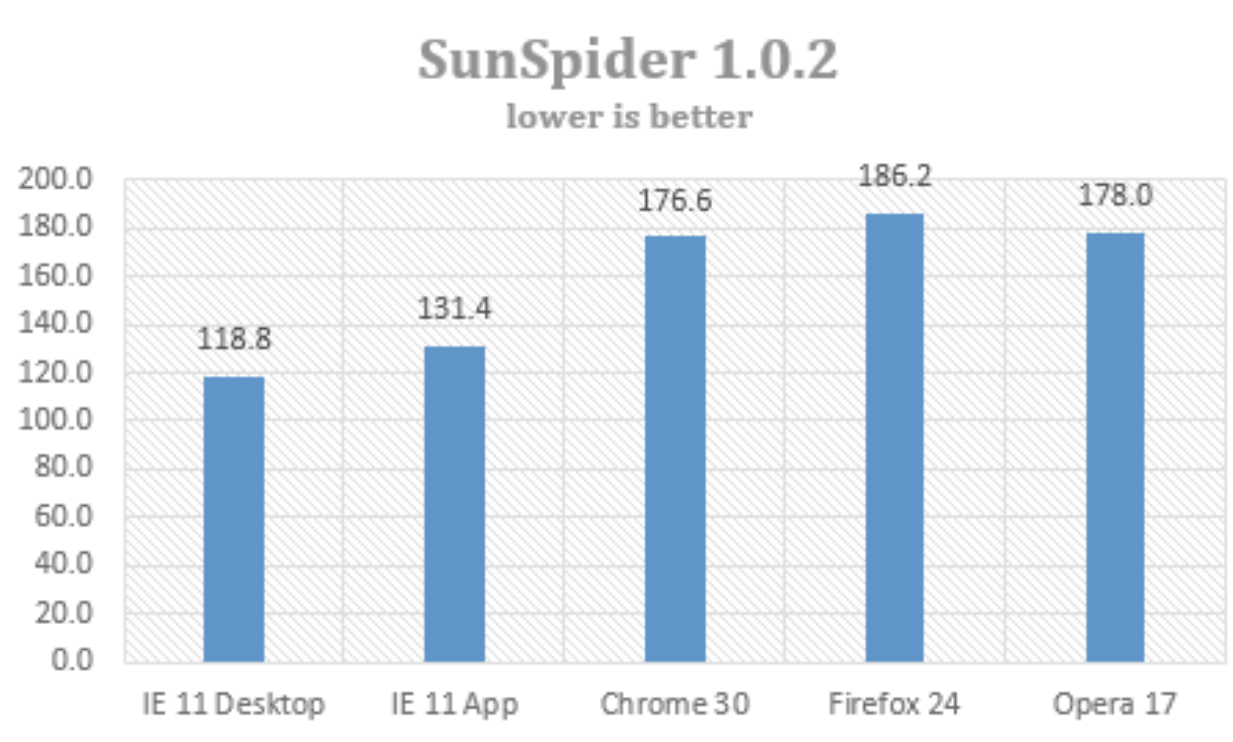
JavaScript Performance 2008

Milliseconds



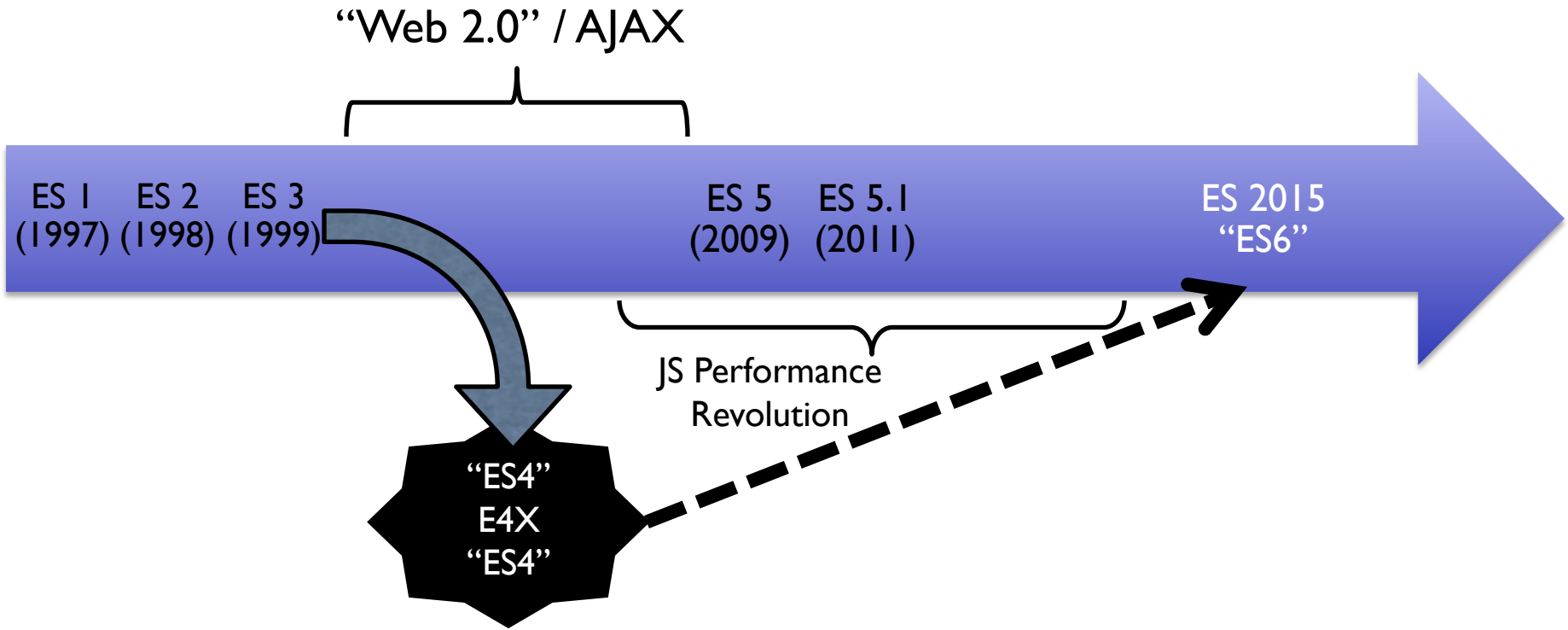
JavaScript Performance 2013

Milliseconds



http://www.7tutorials.com/browser-wars-does-internet-explorer-11-deliver-browsing-performance

The ECMAScript Standard Timeline



Things TC-39 focused on for ES 6

- Modularity
- Better Abstraction Capability
  - Better functional programming support
  - Better OO Support
- Expressiveness and Clarity
- Better Compilation Target
- Things that nobody else can do

In an evolutionary manner

What Kind of Language Is JavaScript?

- Functional?
- Object-oriented?
  - Class-based?
  - Prototype-based?
- Permissive?
- Secure?

Photo by crazybarefootpoet @ flickr (CC BY-NC-SA 2.0)

TC-39?: It's not like this…

…and not like this, either

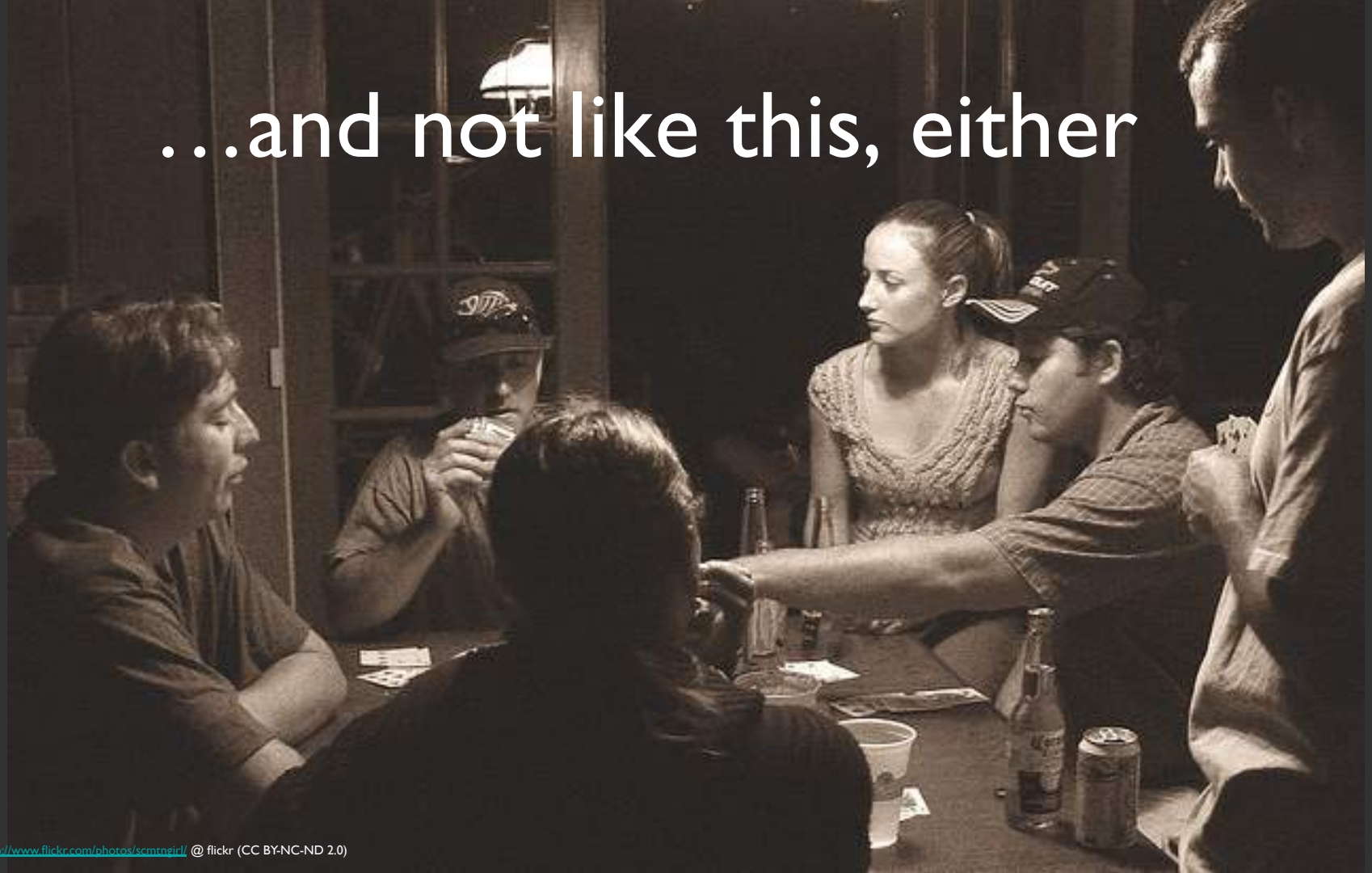
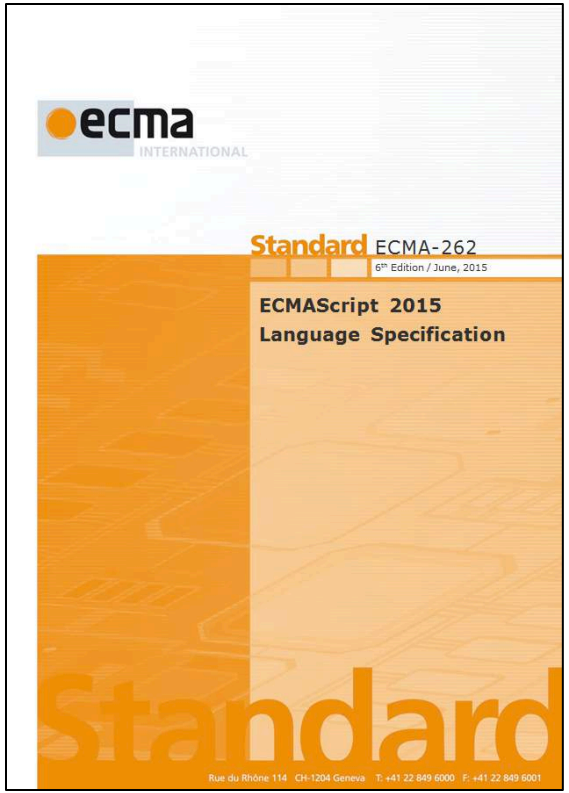


Photo by http://www.flickr.com/photos/scmtrngirl/ @ flickr (CC BY-NC-ND 2.0)


jQuery
Google
intel
ebay
NETFLIX
YAHOO!

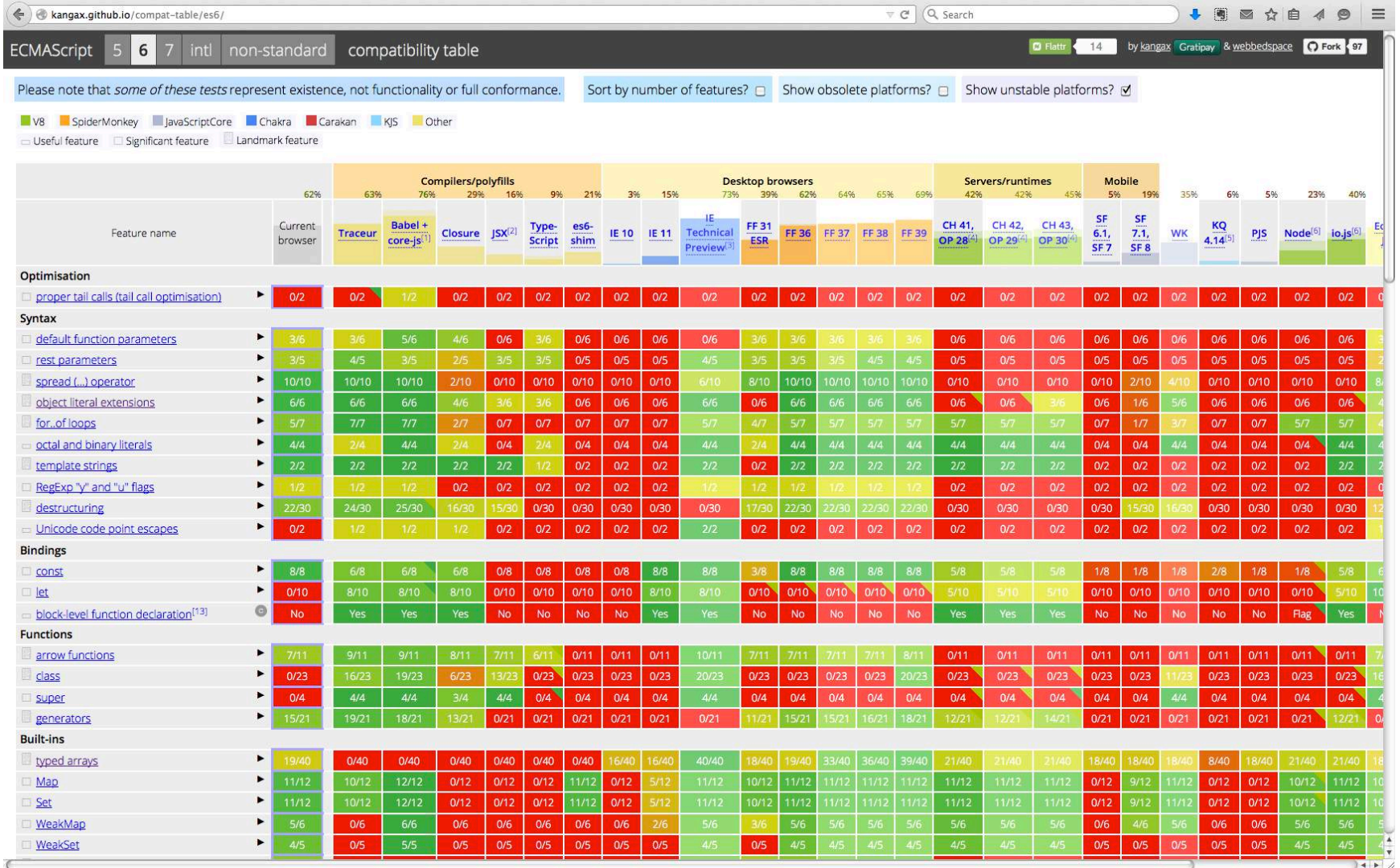
Some ECMAScript 2015 Enhancements

- More concise and expressive syntax
- Modules
- Class Declarations
- Block scoped declarations
- Control abstraction via iterators and generators
- Promises
- String interpolation/Internal DSL support
- Subclassable built-ins
- Binary Array Objects with Array methods
- Built-in hash Maps and Sets + weak variants.
- More built-in Math and String functions
- Improved Unicode support



ES 5.1: 250 pages
ES 2015: 591 pages

https://github.com/lukehoban/es6features



kangax.github.io/compat-table/es6/

ECMAScript 5 6 7 intl non-standard compatibility table

Flattr 14 by kangax Gratipay & webbedspace Fork 97

Please note that *some of these tests* represent existence, not functionality or full conformance.

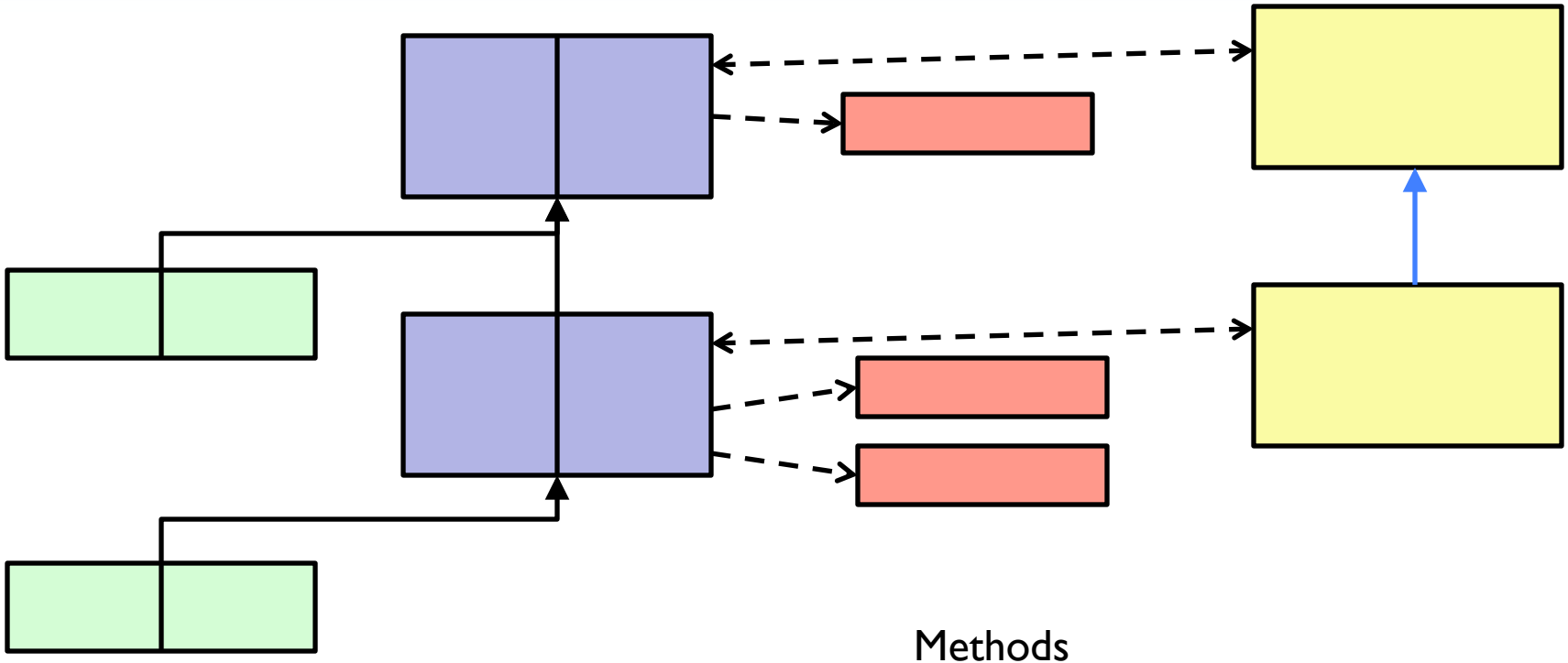
Sort by number of features? ☐ Show obsolete platforms? ☐ Show unstable platforms? ☑

V8 SpiderMonkey JavaScriptCore Chakra Carakan KJS Other

Useful feature Significant feature Landmark feature

Feature name	Current browser	Compilers/polyfills: Traceur	Babel + core-js[1]	Closure	JSX[2]	TypeScript	es6-shim	Desktop browsers: IE 10	IE 11	IE Technical Preview[3]	FF 31 ESR	FF 36	FF 37	FF 38	FF 39	Servers/runtimes: CH 41, OP 28[4]	CH 42, OP 29[4]	CH 43, OP 30[4]	Mobile: SF 6.1, SF 7	SF 7.1, SF 8	WK	KQ 4.14[5]	PJS	Node[6]	io.js[6]	Ec
	62%	63%	76%	29%	16%	9%	21%	3%	15%	73%	39%	62%	64%	65%	69%	42%	42%	45%	5%	19%	35%	6%	5%	23%	40%	
Optimisation																										
proper tail calls (tail call optimisation)	0/2	0/2	1/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0
Syntax																										
default function parameters	3/6	3/6	5/6	4/6	0/6	3/6	0/6	0/6	0/6	0/6	3/6	3/6	3/6	3/6	3/6	0/6	0/6	0/6	0/6	0/6	0/6	0/6	0/6	0/6	0/6	3
rest parameters	3/5	4/5	3/5	2/5	3/5	3/5	0/5	0/5	0/5	4/5	3/5	3/5	3/5	4/5	4/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5	2
spread (...) operator	10/10	10/10	10/10	2/10	0/10	0/10	0/10	0/10	0/10	6/10	8/10	10/10	10/10	10/10	10/10	0/10	0/10	0/10	0/10	2/10	4/10	0/10	0/10	0/10	0/10	8
object literal extensions	6/6	6/6	6/6	4/6	3/6	3/6	0/6	0/6	0/6	6/6	0/6	6/6	6/6	6/6	6/6	0/6	0/6	3/6	0/6	1/6	5/6	0/6	0/6	0/6	0/6	4
for..of loops	5/7	7/7	7/7	2/7	0/7	0/7	0/7	0/7	0/7	5/7	4/7	5/7	5/7	5/7	5/7	5/7	5/7	5/7	0/7	1/7	3/7	0/7	0/7	5/7	5/7	4
octal and binary literals	4/4	2/4	4/4	2/4	0/4	2/4	0/4	0/4	0/4	4/4	2/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	0/4	0/4	4/4	0/4	0/4	0/4	4/4	4
template strings	2/2	2/2	2/2	2/2	2/2	1/2	0/2	0/2	0/2	2/2	0/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	0/2	0/2	0/2	0/2	0/2	0/2	2/2	2
RegExp "y" and "u" flags	1/2	1/2	1/2	0/2	0/2	0/2	0/2	0/2	0/2	1/2	1/2	1/2	1/2	1/2	1/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0
destructuring	22/30	24/30	25/30	16/30	15/30	0/30	0/30	0/30	0/30	0/30	17/30	22/30	22/30	22/30	22/30	0/30	0/30	0/30	0/30	15/30	16/30	0/30	0/30	0/30	0/30	12
Unicode code point escapes	0/2	1/2	1/2	1/2	0/2	0/2	0/2	0/2	0/2	2/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	
Bindings																										
const	8/8	6/8	6/8	6/8	0/8	0/8	0/8	0/8	8/8	8/8	3/8	8/8	8/8	8/8	8/8	5/8	5/8	5/8	1/8	1/8	1/8	2/8	1/8	1/8	5/8	6
let	0/10	8/10	8/10	8/10	0/10	0/10	0/10	0/10	8/10	8/10	0/10	0/10	0/10	0/10	0/10	5/10	5/10	5/10	0/10	0/10	0/10	0/10	0/10	0/10	5/10	10
block-level function declaration[13]	No	Yes	Yes	Yes	No	No	No	No	Yes	Yes	No	No	No	No	No	Yes	Yes	Yes	No	No	No	No	No	Flag	Yes	
Functions																										
arrow functions	7/11	9/11	9/11	8/11	7/11	6/11	0/11	0/11	0/11	10/11	7/11	7/11	7/11	7/11	8/11	0/11	0/11	0/11	0/11	0/11	0/11	0/11	0/11	0/11	0/11	7
class	0/23	16/23	19/23	6/23	13/23	0/23	0/23	0/23	0/23	20/23	0/23	0/23	0/23	0/23	20/23	0/23	0/23	0/23	0/23	0/23	11/23	0/23	0/23	0/23	0/23	16
super	0/4	4/4	4/4	3/4	4/4	0/4	0/4	0/4	0/4	4/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4	4/4	0/4	0/4	0/4	0/4	4
generators	15/21	19/21	18/21	13/21	0/21	0/21	0/21	0/21	0/21	0/21	11/21	15/21	15/21	16/21	18/21	12/21	12/21	14/21	0/21	0/21	0/21	0/21	0/21	0/21	12/21	0
Built-ins																										
typed arrays	19/40	0/40	0/40	0/40	0/40	0/40	0/40	16/40	16/40	40/40	18/40	19/40	33/40	36/40	39/40	21/40	21/40	21/40	18/40	18/40	18/40	8/40	18/40	21/40	21/40	18
Map	11/12	10/12	12/12	0/12	0/12	0/12	11/12	0/12	5/12	11/12	10/12	11/12	11/12	11/12	11/12	11/12	11/12	11/12	0/12	9/12	11/12	0/12	0/12	10/12	11/12	10
Set	11/12	10/12	12/12	0/12	0/12	0/12	11/12	0/12	5/12	11/12	10/12	11/12	11/12	11/12	11/12	11/12	11/12	11/12	0/12	9/12	11/12	0/12	0/12	10/12	11/12	10
WeakMap	5/6	0/6	6/6	0/6	0/6	0/6	0/6	0/6	2/6	5/6	3/6	5/6	5/6	5/6	5/6	5/6	5/6	5/6	0/6	4/6	5/6	0/6	0/6	5/6	5/6	5
WeakSet	4/5	0/5	5/5	0/5	0/5	0/5	0/5	0/5	0/5	4/5	0/5	4/5	4/5	4/5	4/5	4/5	4/5	4/5	0/5	0/5	0/5	0/5	0/5	4/5	4/5	4

JavaScript Class "Constructor" Pattern



Classes ES5 vs ES 2015

```
//ES5 define Employee as subclass of Person

function Employee(name,id) {
   Person.call(name);
   this.id = id;
}
Employee.prototype=Object.create(Person.prototype);
Object.defineProperty(Employee.prototype, “constructor”,
  {value:Employee,enumerable:false,configurable: true});
Employee.__proto__ = Person;
Employee.withId = function (id) {…}
Employee.prototype.hire = function() {…};
Employee.prototype.fire = function () {…};

…
```

```
//ES2015 define Employee as subclass of Person

class Employee extends Person {
   constructor(name,id) {
     super(name);
     this.id = id;
   }
   hire () {…}
   fire () {…}
   static withId (id) {…}
   …
}
```

Both create the same object structure

The closure in loop problem

```
function f(x) {
    for (var p in x) {
        var v = doSomething(x, p);
        obj.addCallback(
              function(args){
                    handle(v, p, args)
        )};
    }
}
…
obj.runCallbacks();
```

var hoisting causes the problem

```
function f(x) {
    var p;
    var v;
    for (var p in x) {
        var v = doSomething(x, p);
        obj.setCallback(
                function(args){
                        handle(v, p, args)
        )};
     }
}
…
obj.runCallbacks();
```

ES6 can't redefine the scoping of `var`

```
function f(x) {
    for (var p in x) {
        var v = doSomething(x, p);
        if (v === somethingSpecial) break;
    }
    if (v === somethingSpecial) ...
}
```

Other local scoping WTFs

```
function f(x,x) {
    var x;
    for (var x in obj) {
         if (obj[x] === somethingSpecial) {
            var x = 0;
            ...
         }
    }
    function x() { doSomething()}
    x();
}
```

Fixing closure in loop problem: Add a new block scoped declaration

```
function f(x) {
    for (~~var~~ let p in x) {
        ~~var~~ let v = doSomething(x, p);
        obj.setCallback(
                function(args){
                        handle(v, p, args)
        )};
    }
}
…
obj.runCallbacks();
```

Want to avoid new `let` WTFs

```
//duplicate declarations
function f() {
    let x = 1;
    let x = 2;
}
```

```
//duplicate let and parameter
function h(x) {
    let x = 1;
}
```

```
//duplicate let and function
function h() {
    let x = 1;
    function x() {}
}
```

```
//duplicate let and var
function g() {
    let x = 1;
    var x = 2;
}
```

```
//hoist var to/over let
function ff() {
    let x = 1;
    if (pred) {
      var x;
    }
}
```

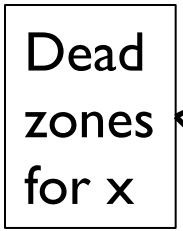
Static Errors

Avoid inconsistent scoping WTFs

```
//Forward or outer reference
let x = 1;
function f() {
    console.log(x);//1,undefined, error?
    let x = 2;
}
```

Within any block, a name must have single consistent binding.

And bogus forward references




```
//Forward reference
let x = 1;
function f() {
    x = 2;
    console.log(x);//2, 3,undefined, error?
    let x = 3;
}

//Forward reference const
function f() {
    console.log(x);//3,undefined, error?
    const x = 3;
}
```

Some forward references are desirable

```
//Forward reference
let x = 1;
function f() {
    const r1 = function(n) {n>0?r2(n-1):0}
    const r2 = function(n) {n>0?r1(n-1):0}
    r1(10);
}
```

Static dead zones are too restrictive

Temporal Dead Zones:

- Error based upon time of actual access
- Not upon position in source code
- Must be initialized prior to access or assignment

Some ES6 Declaration Rules

- Single unique binding for any name in a scope.
- Multiple `var` and top-level `function` declarations for the same name are allowed. (Still one binding per name) Just like ES1-5
- All other multiple declarations are errors: var/let, let/let, let/const, class/function, etc.
- var declarations hoist to top level and auto initialized to undefined. Just like ES1-5
- Can't hoist a `var` over any other declaration of same name (except a top-level function)
- Runtime error, for accessing or assigning to an uninitialized binding
- let, const, class declarations are dead until initialized (TDZ).

What about Block Scoped Functions?

```
//block scoped function
function f() {
    if (pred) {
        function g() {}
    }
    g( );
}
```

```
//duplicate declarations
function f() {
    if (pred) {
        function g() {}
        g( );
        function g() {}
    }
}
```

Why not, ECMAScript 1-5 didn't allow function declarations in blocks.

But real browsers agree...

```
//block scoped function?
function f() {
    if (pred) {
        function g() {}
        g( );
    }
}
```

and real browsers, disagree...

```
function f() {
    g( ); //is g defined?
    if (pred) {
        function g() {}
        g( );
    }
}
```

```
function f() {
    if (pred) {
        function g() {}
        g( ); //which g??
    }
    function g() {}
}
```

Disagreement on non-standard features is an standardization opportunity.

Unfortunately, all browsers agree:

```
//block scoped function?
function f() {
    if (pred) {
        function g() {}
        g( );
    }
    g( ); //calls g from block
}
```

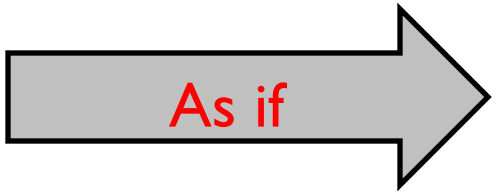
Web interoperability requires leaking some block level function declarations!

What is the common interoperable browser semantics for block function declarations?

- A function declared only once in a block can be called from within the block at any point following the actual declaration.
- A function declared only once in a block, can be referenced from any point that follows the block in the surrounding function.

Making block scoped functions compatible with intersection semantics

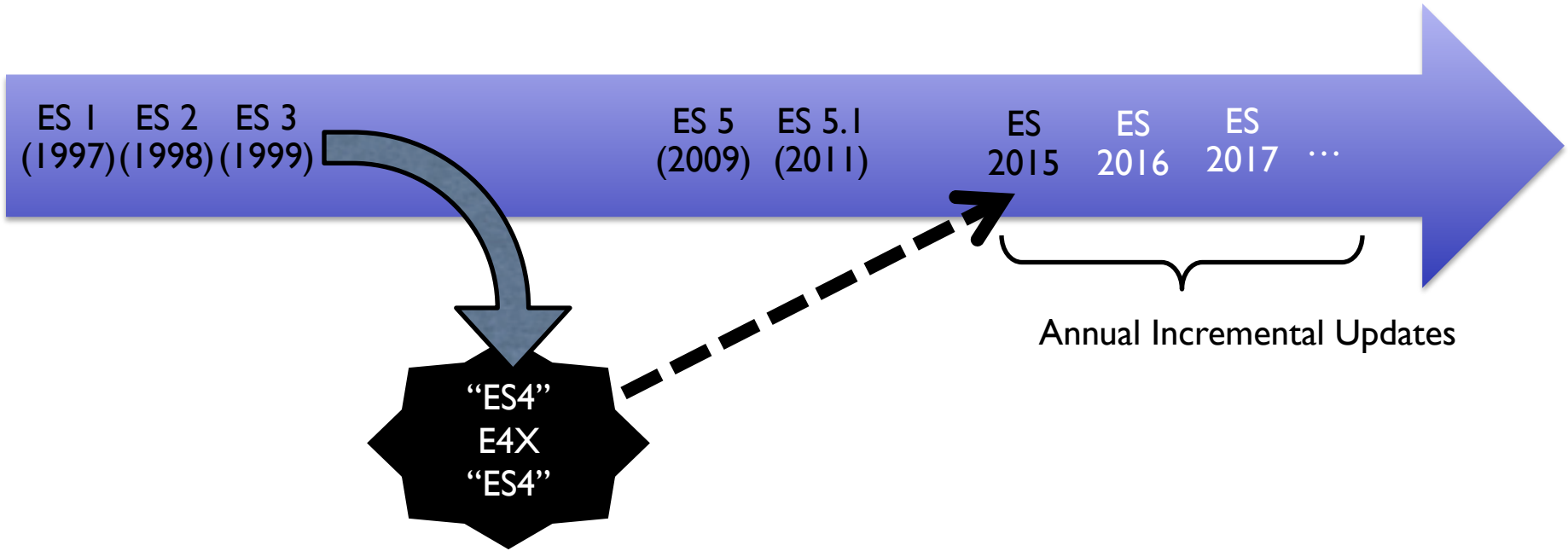
```
function f() {
    if (pred) {
        function g() {}
        }
    g();
}
```

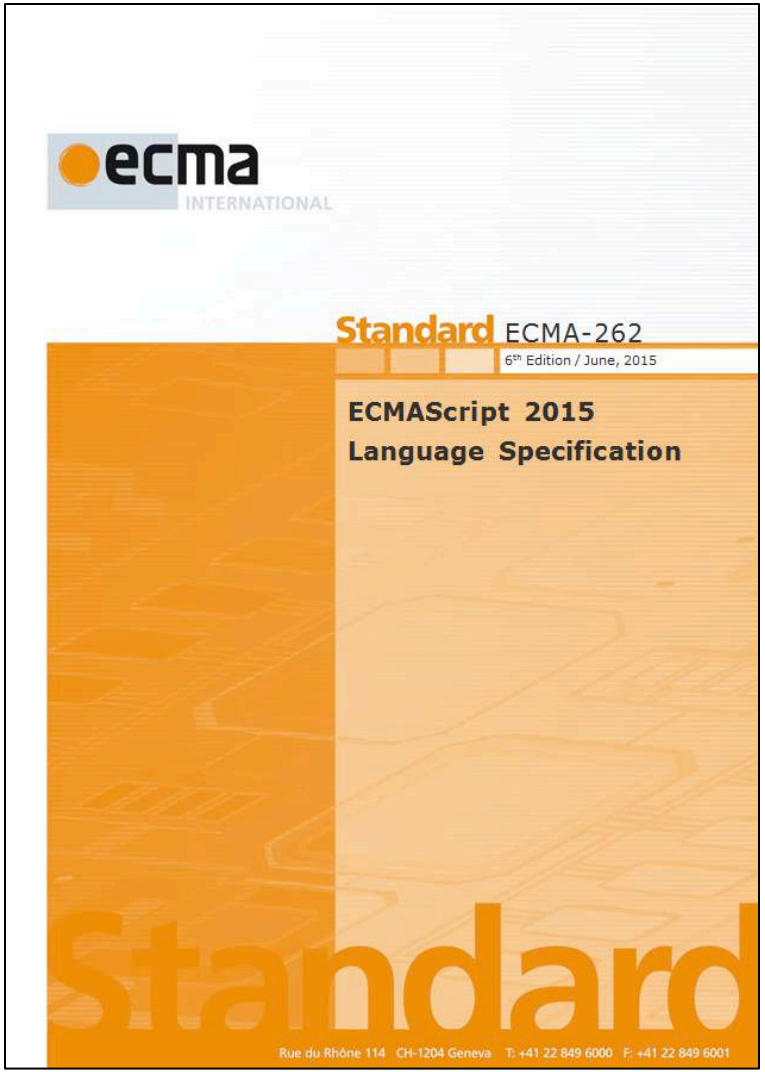



```
function f() {
    var g;
    function __g(v) {g=v}
    if (pred) {
        function g() {}
        __g(g);
    }
   g( );
}
```

But only if treating the function declaration as a hoisted var declaration would not be an error.

The ECMAScript Standard Timeline





- It's real
- The specification is done
- Transpilers and polyfills available today
- It's being implemented in your favorite bowers right now
- It's the foundation for the next 10-20 years of JavaScript evolution

It Has Legs


GYPSOPHILA
ZINNIAS

ECMAScript Resources

ES6 Specification Drafts

http://wiki.ecmascript.org/doku.php?id=harmony:specification_drafts

ES6 Feature Summary

https://github.com/lukehoban/es6features

ES6 translators and tools

https://github.com/addyosmani/es6-tools

The TC-39 ECMAScript Design Discussion Mail List

https://mail.mozilla.org/listinfo/es-discuss

Test262: The Offical ECMAScript Implementation Test Suite

http://test262.ecmascript.org/

TC39 on Github

http://test262.ecmascript.org/

Please report bugs

http://bugs.ecmascript.org