Modern JavaScript: The Smalltalk Influence

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

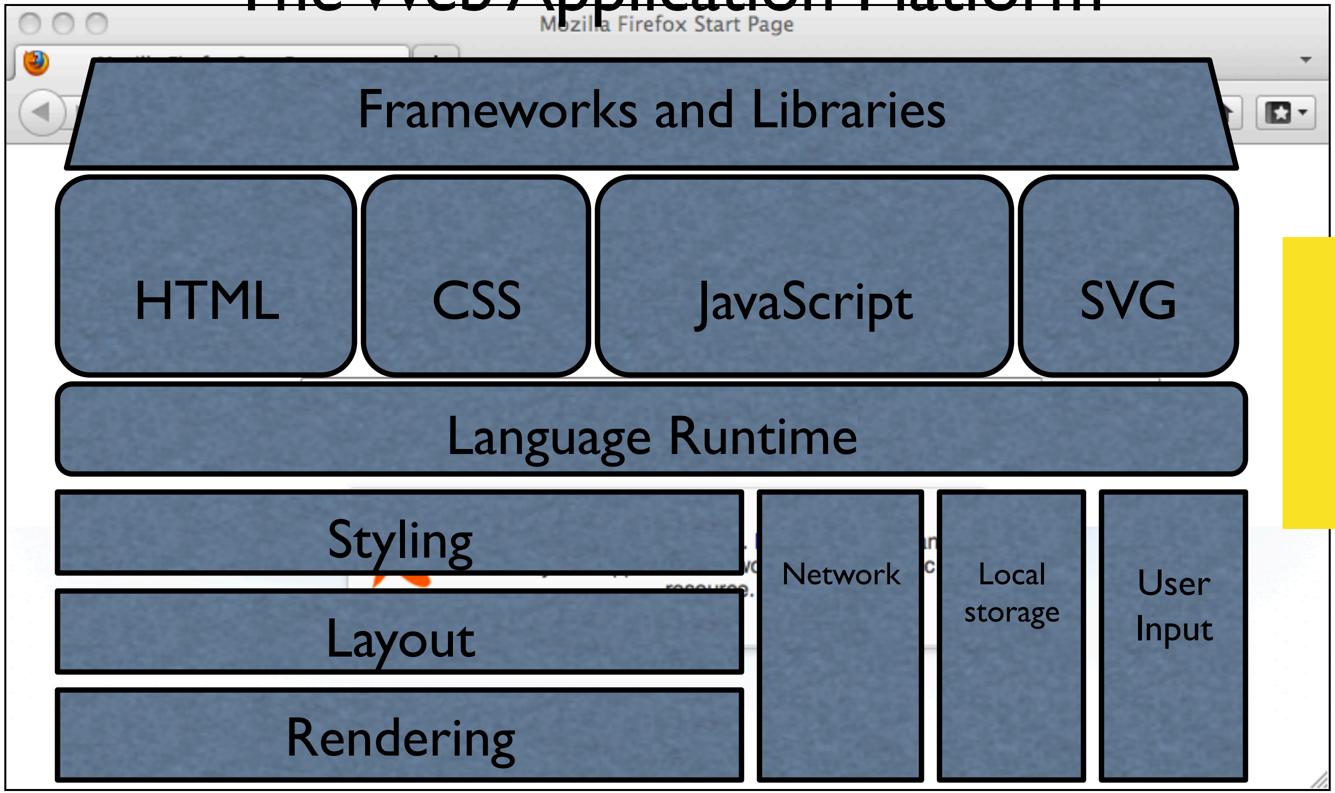
Smalltalks 2014
November 7, 2014
Córdoba, Argentina

Every Computing Era Has a Dominant Application Platform

- Corporate Computing Era: IBM Mainframes
- Personal Computing Era: Microsoft/Intel PC
- Ambient Computing Era: The Web is the Platform?

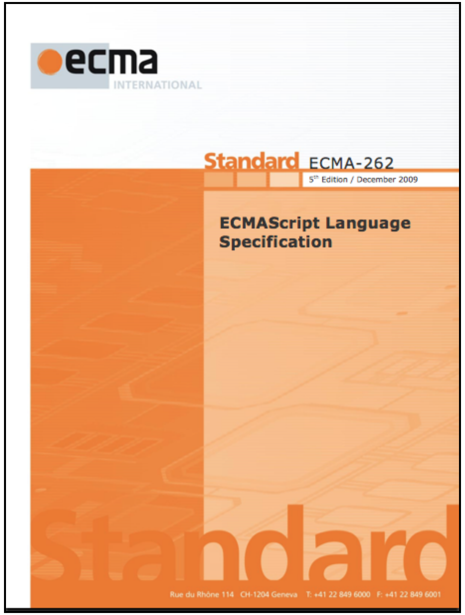
The Web Is the Platform

The Web Application Platform



Each Computing Era has had Canonical Programming Languages

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



Why JavaScript?

Because "Worse is Better"

Dick Grabriel
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?

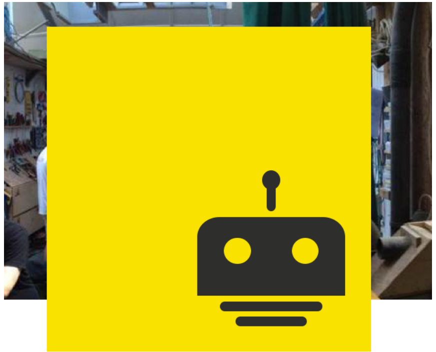
JS

Not Just In Web Browsers

http://nodejs.org/

Firefox OS

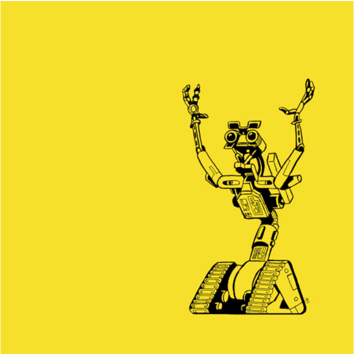
https://developer.mozilla.org/en-US/Firefox_OS



NodeBots

Robots powered by JavaScript

http://nodebots.io/



Johnny-Five

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


TEN
"SEVEN DAYS
IN MAY"
THE JOHN FRANKENHEIMER-JOEL PRODUCTION
PRODUCED BY EDWARD LEWIS · DIRECTED BY JOHN FRANKENHEIMER
A PARAMOUNT RELEASE

JavaScript Influences

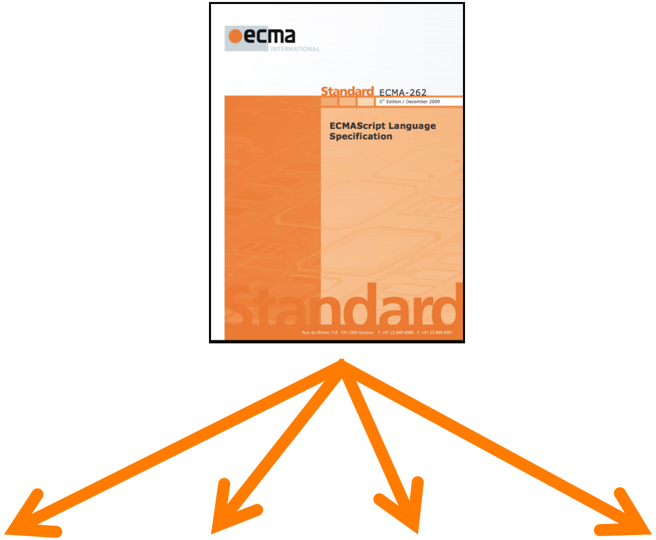
- Scheme
  - ✓ First class closures
- Self
  - ✓ Objects with individual behaviors
  - ✓ Prototypal Inheritance
- C (Java)
  - ✓ Syntax
  - ✓ Corporate Strategy Tax

JavaScript Early history

- 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 JavaScript
- Developed by Technical Committee 39 (TC-39) of Ecma International
- Issued as a Ecma-262 and ISO/IEC 16262
- Not part of W3C

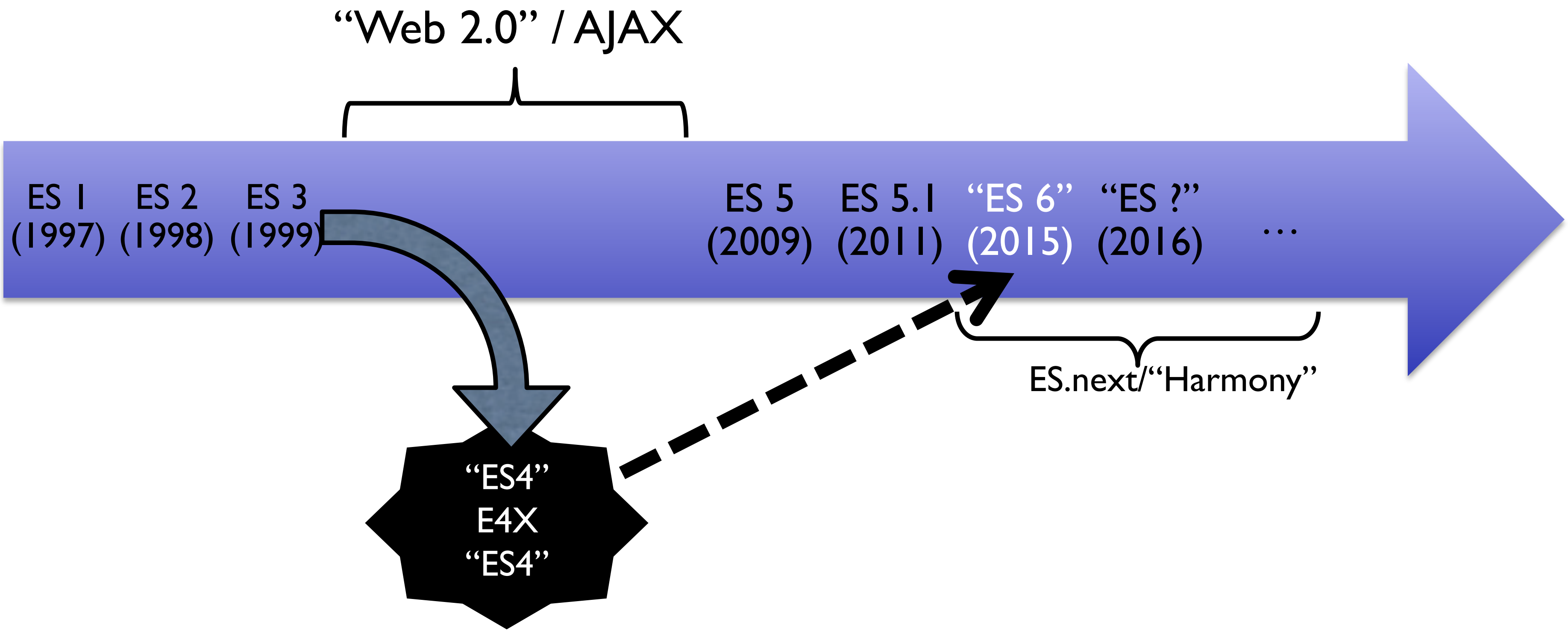


Google	Mozilla	Microsoft	Webkit
V8	SpiderMonkey	Chakra	JSCore

JavaScript Implementations


jQuery
Google
intel
ebay
NETFLIX
YAHOO!

The ECMAScript Standard Timeline



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.

Things TC-39 focused on for ES 6

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


1990
LADY NOEL STEWART

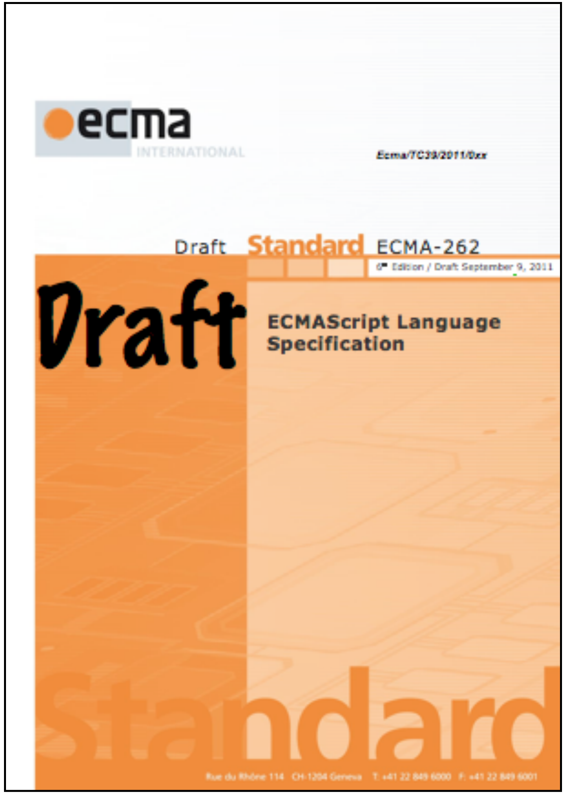
What Kind of Language Is JavaScript?

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

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

Some ECMAScript 6 Enhancements

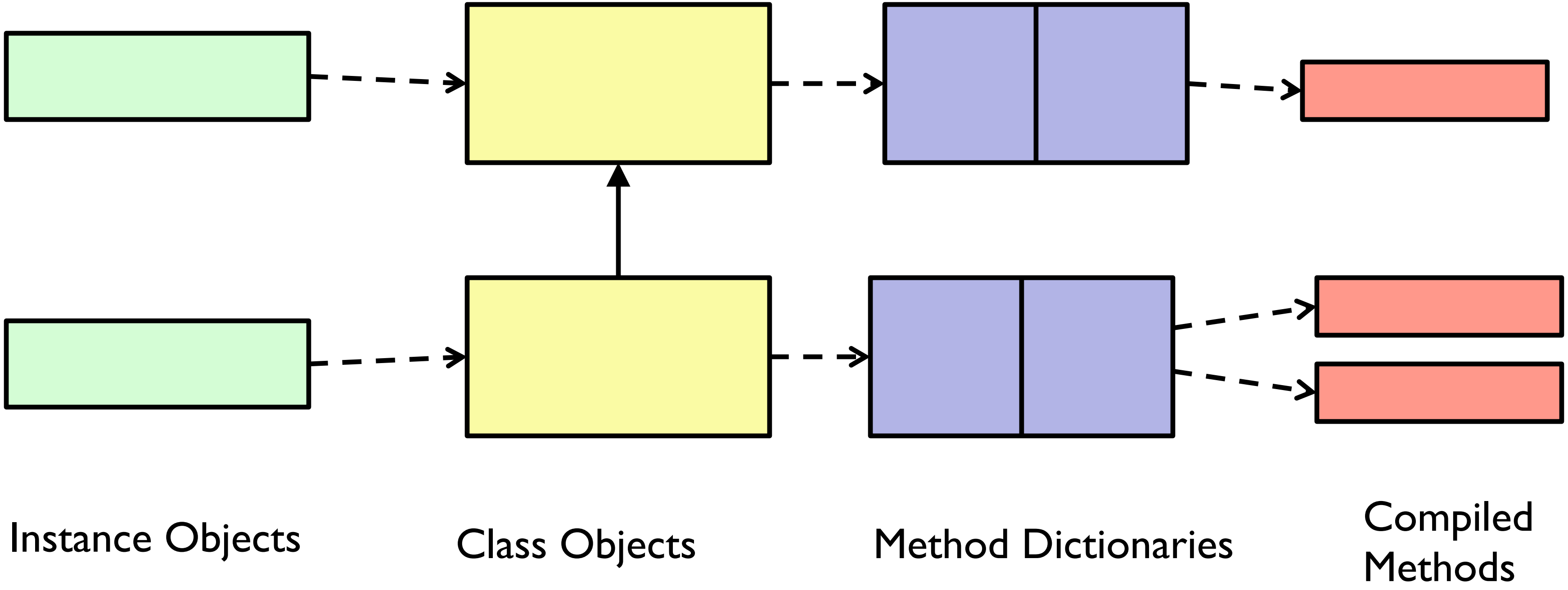
- More concise and expressive syntax
- Modules and Sanding-boxing
- Class Declarations
- Block scoped declarations
- Control abstraction via iterators and generators
- Promises
- String interpolation/Internal DSL support
- Subclassable built-ins
- Binary Array Objects with Array methods
- Built-in hash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



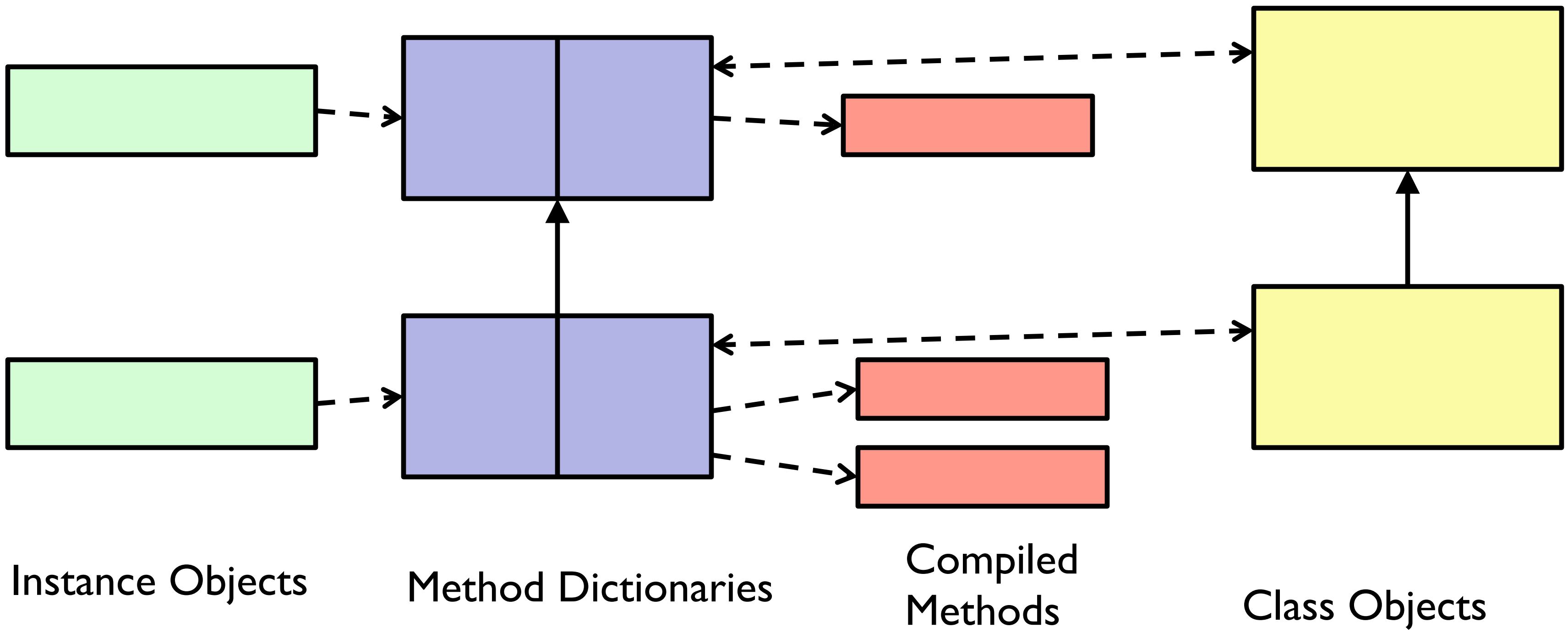
ES 5.1: 250 pages
ES 6 draft: 651 pages

https://github.com/lukehoban/es6features

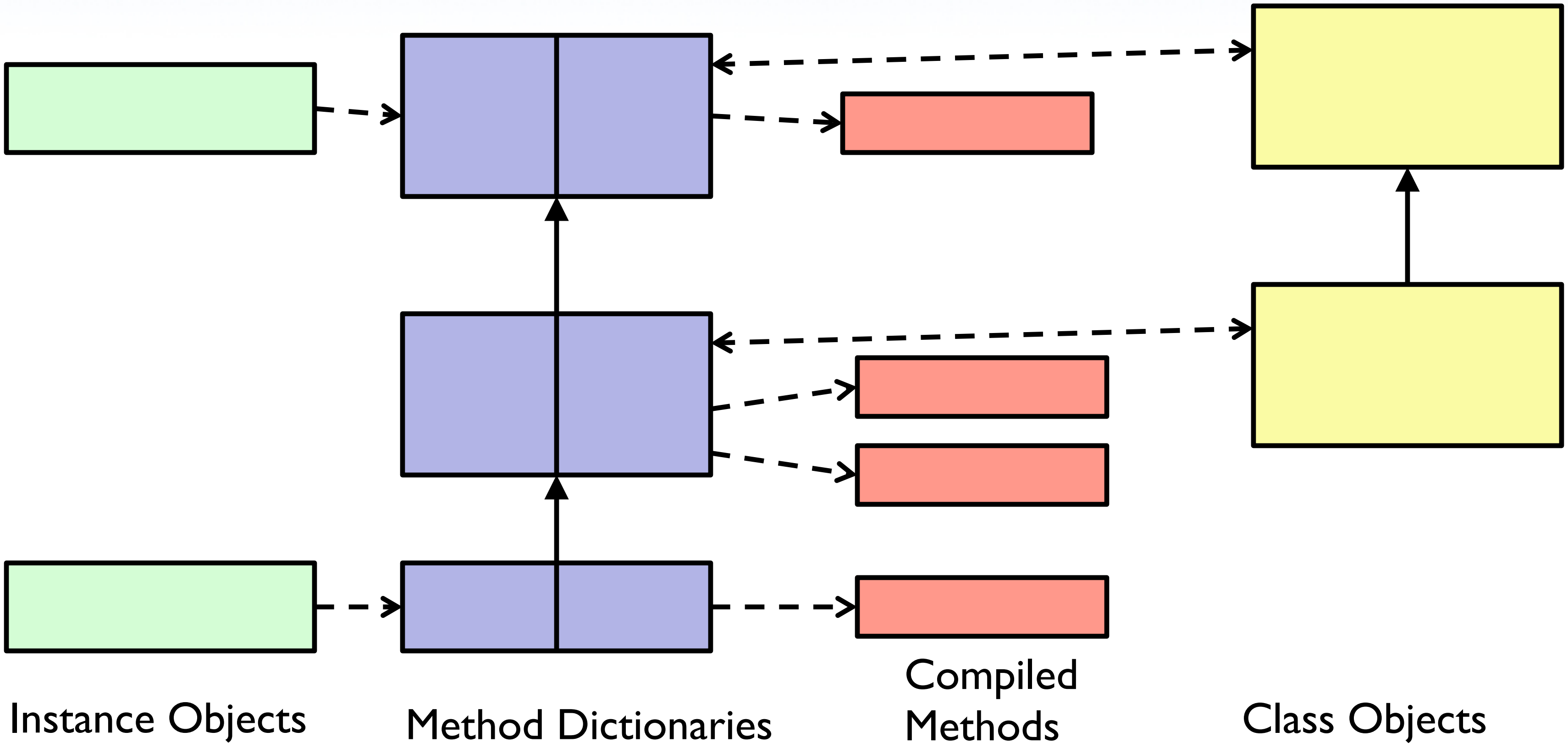
Classic Smalltalk Objects



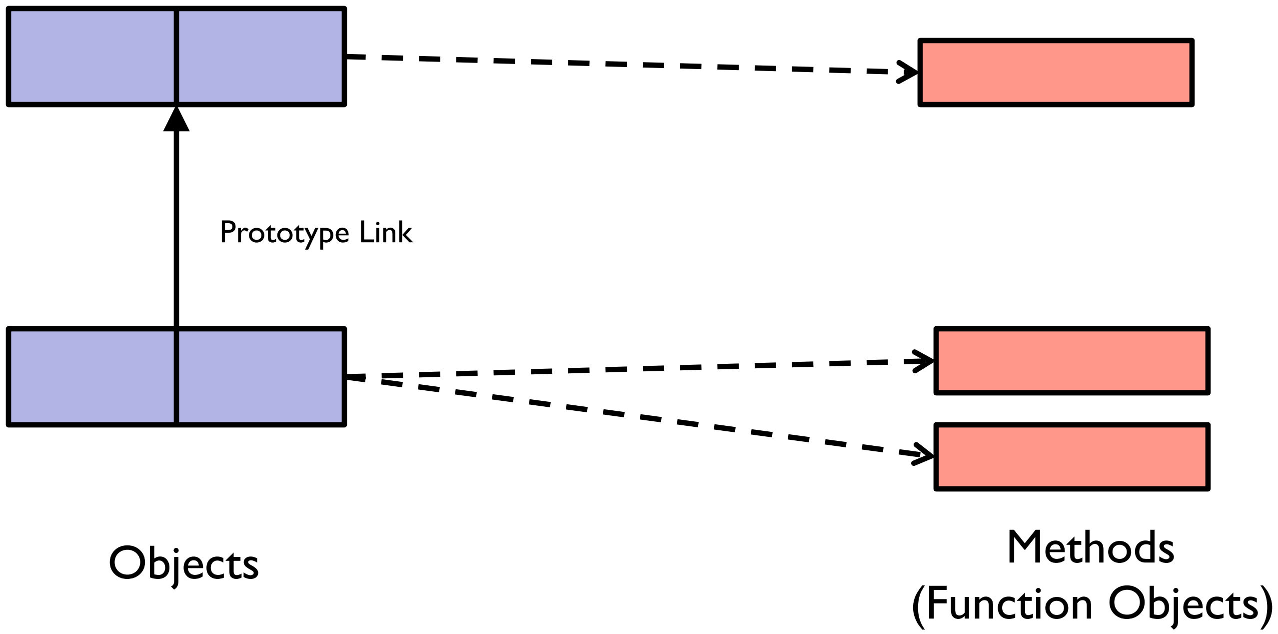
VSE Smalltalk Objects



VSE Smalltalk Objects + Per-instance methods



JavaScript Object Model



Creating a JavaScript object

```
var o = new Object;
o.counter = 0;
o.incr = function (n) {
   this.o+=n;
};
o.toString = function() {
   return
     this.counter.toString();
};
```

Imperatively

```
var o = {
  counter: 0,
  incr: function (n) {
    this.o+=n;
  },
  toString: function() {
    return
       this.counter.toString();
  }
};
```

Object Literal

```
var o = {
  counter: 0,
  incr (n) {
    this.o+=n;
  },
  toString () {
    return
       this.counter.toString();
  }
};
```

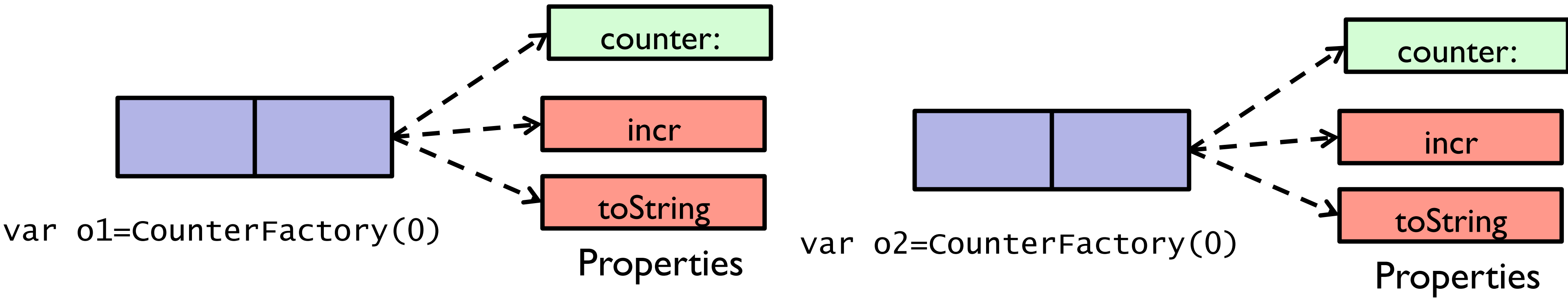
Object Literal (ES6)

Or, Define a Factory

```
function CounterFactory (start) {
  return {
    counter: start,
    incr: function (n) {
      this.counter += n;
    },
    toString: function() {
       return this.counter.toString();
    }
  }
};
```

var o = CounterFactor(0);

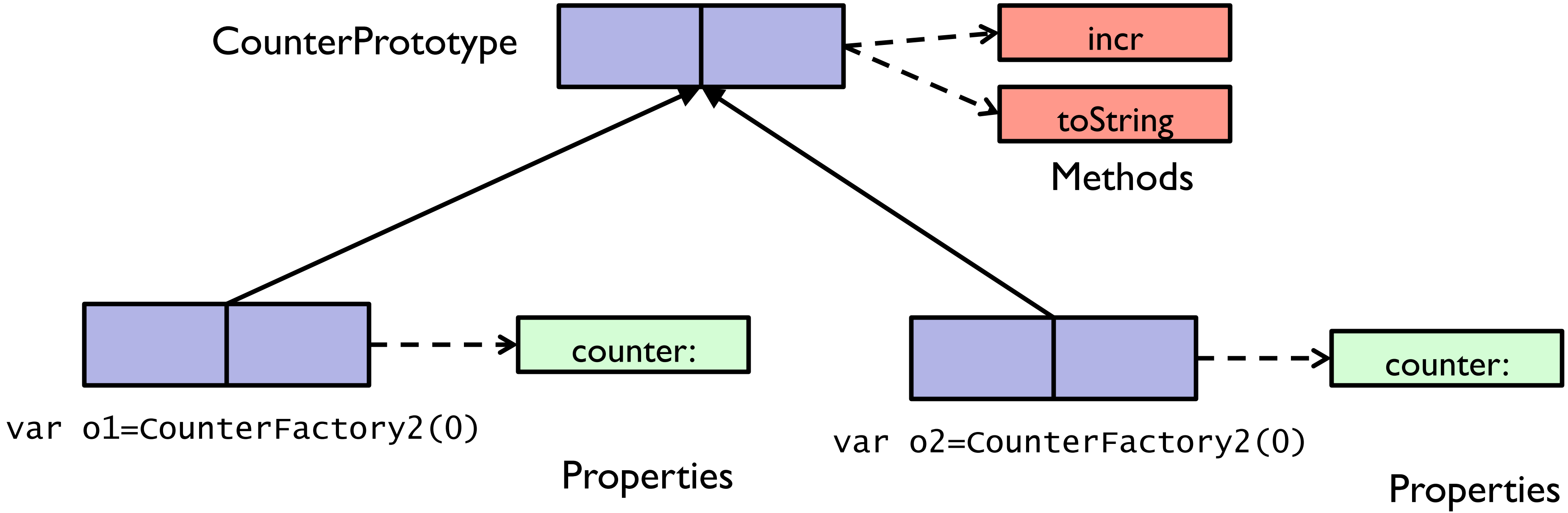
Each object created this way has its own distinct methods



Or, Define a Factory plus a prototype

```
var counterPrototype = {
  incr: function (n) {
    this.counter += n;
  },
  toString: function() {
     return this.counter.toString();
  }
};
function CounterFactory2 (start) {
  var newObj = Object.create(counterPrototype);
  newObj.counter = start;
  return newObj;
};
```

Instance objects share methods via prototype

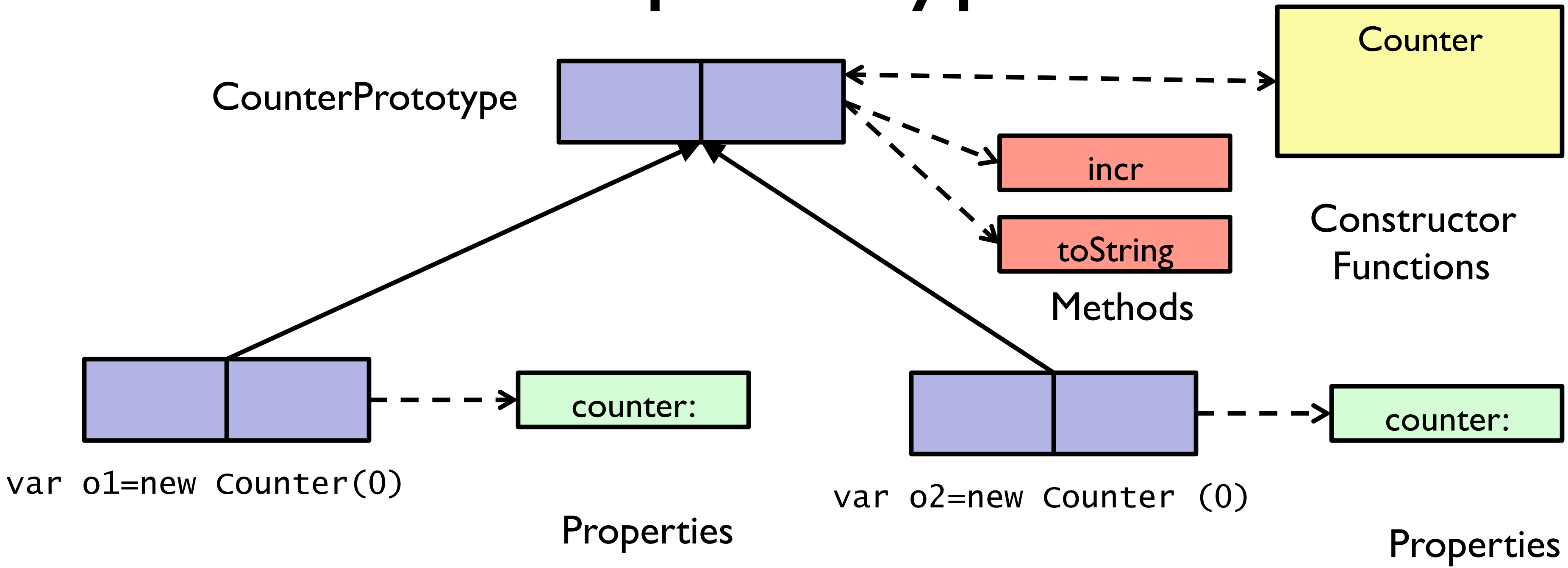


The Constructor Pattern

```
function Counter(start) {
  this.counter = start;
}
Counter.prototype.incr =  function (n) {this.counter += n};
Counter.prototype. toString = function() {return this.counter.toString()};
```

```
var o = new Counter (0);
```

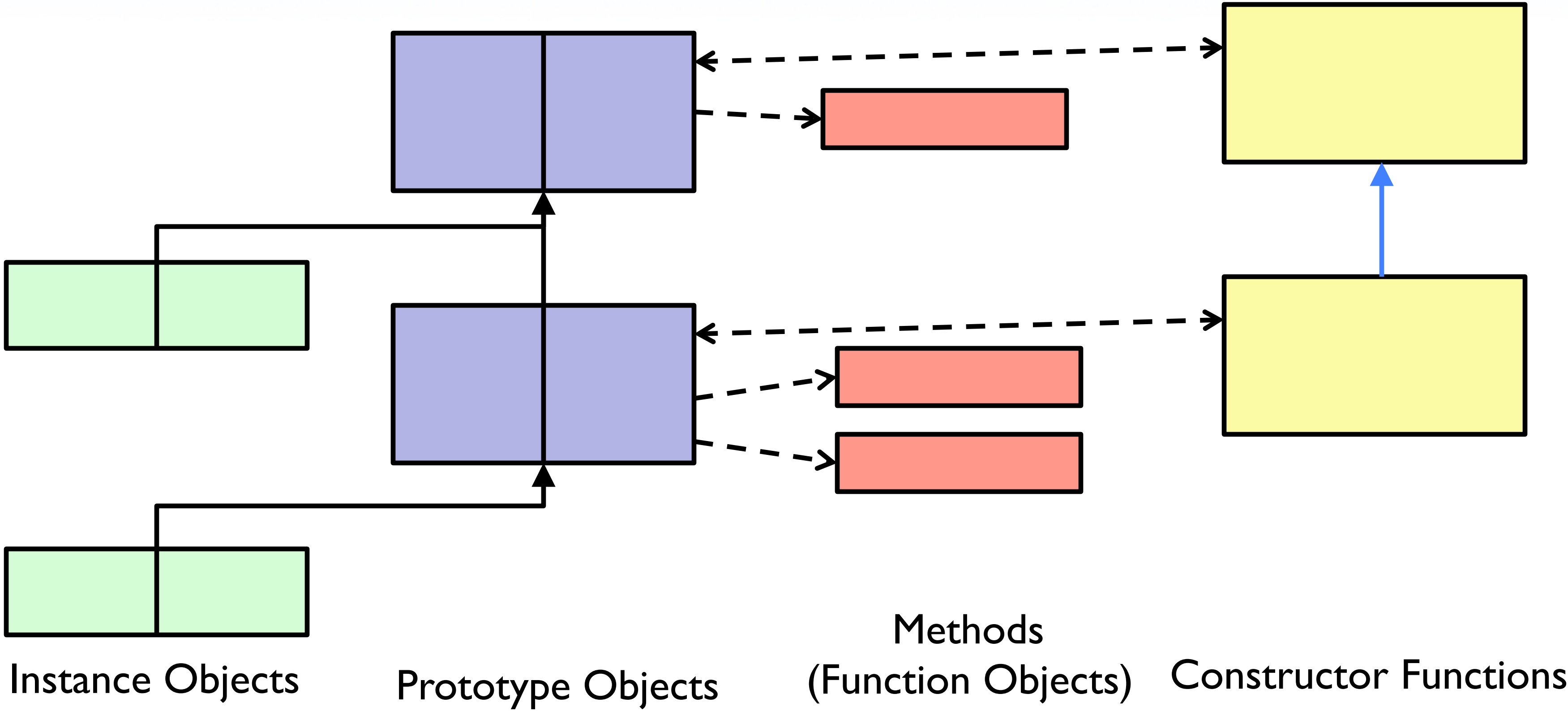
Instance objects share methods via prototype



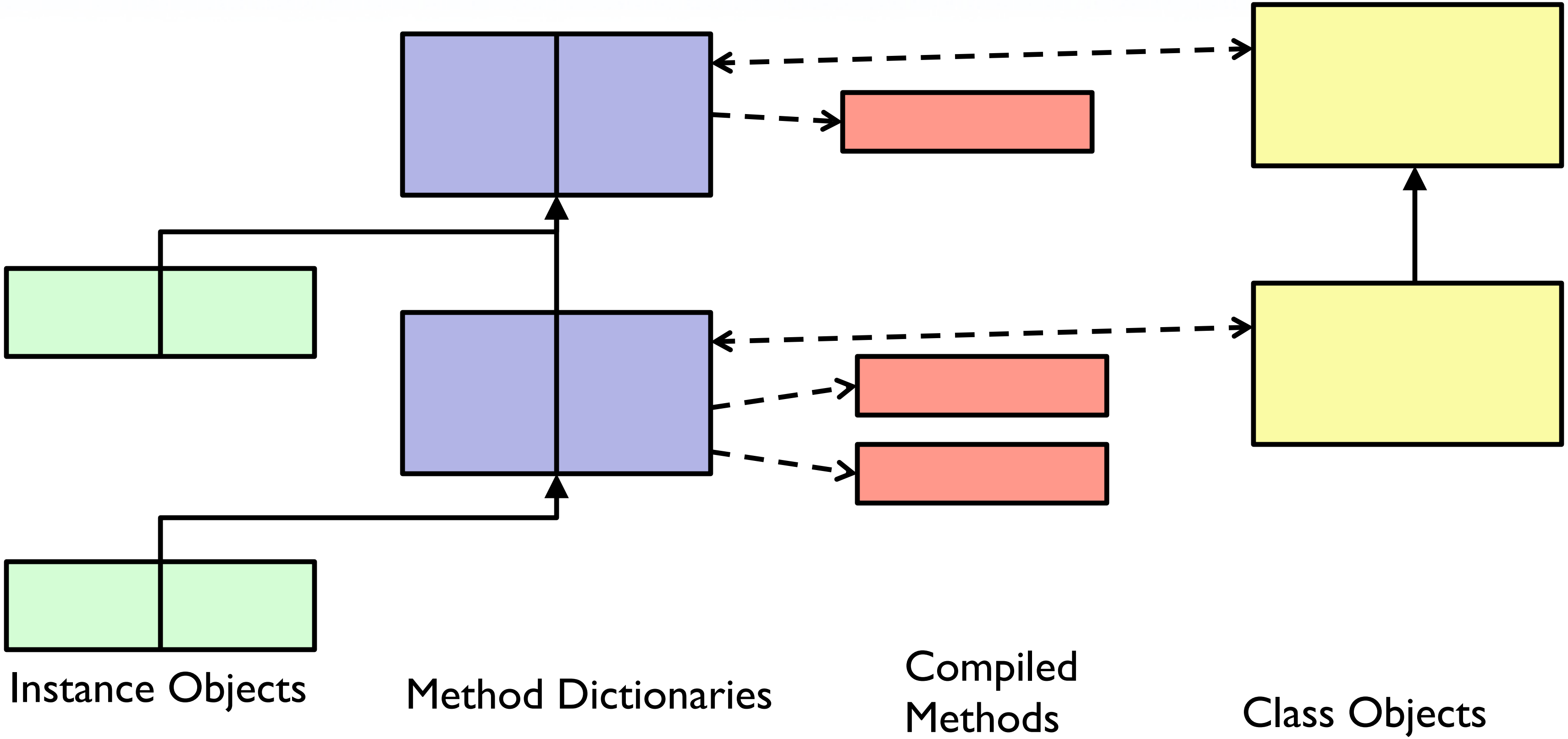
Constructor Pattern with “subclassing”

```
//define Employee as a subclass of Person
function Employee(name,id) {
   Person.call(this, name);   //super initialize: name
   this.id = id;
}
Employee.prototype=Object.create(Person.prototype);
Employee.prototype,.constructor = Employee;  //often forgotten
Employee.__proto__ = Person;                  //class-side inheritance: seldom done
Employee.withId= function(id) {…};            //a “class” method
Employee.prototype.hire = function() {…};
Employee.prototype.fire = function () {…};
…
```

JavaScript “Constructor” Pattern



VSE Smalltalk Objects



Classes in ES 6

```
//ES6: define Employee as subclass of Person

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

Classes Today vs ES 6

```
//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 () {…};


…
```

```
//ES6 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

Arrow Functions

```
var self = this;
var pop = peeps.filter(function(person) {
      return person.age>self.age});
```

Becomes:

```
var pop = peeps.filter(person => person.age>this.age);
```

But Arrow Functions Are Not Quite Smalltalk Blocks

- Return returns from the arrow function rather than the surrounding method.
- There was consider interest in: “block lambdas”:

```
peeps.filter({|person|
  if (person.age>100)return;
  person.age>this.age});
```

- But too many semantics pitfalls with break/continue/return statements.

ES6 Modules

```
//------ lib.js ------
    export const sqrt = Math.sqrt;
    export function square(x) {
        return x * x;
    }
    export function diag(x, y) {
        return sqrt(square(x) + square(y));
    }
```

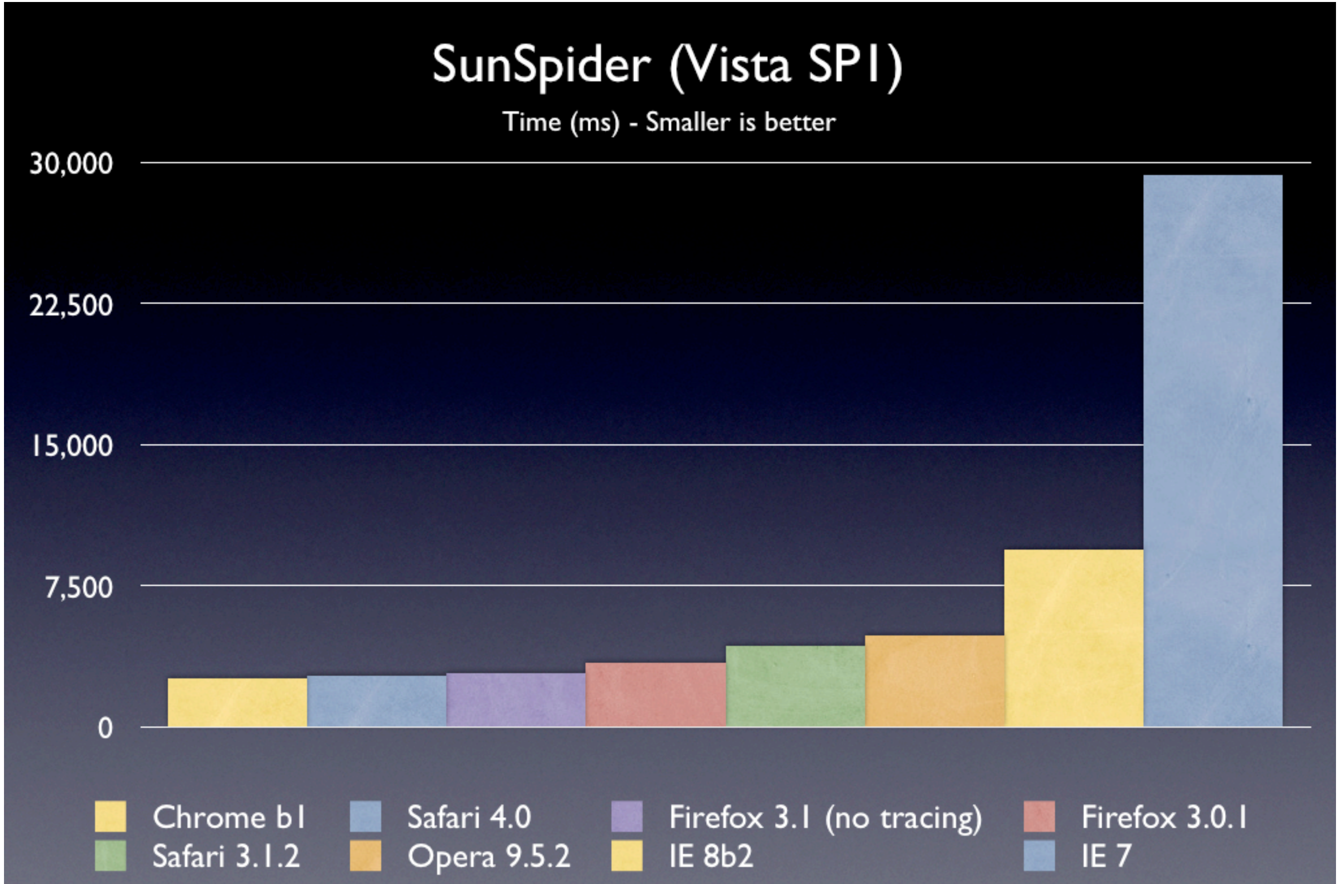
```
//------ main.js ------
    import { square, diag } from 'lib';
    console.log(square(11)); // 121
    console.log(diag(4, 3)); // 5
```

or

```
//------ main.js ------
    import * as lib from 'lib';
    console.log(lib.square(11)); // 121
    console.log(lib.diag(4, 3)); // 5
```

JavaScript Performance 2008

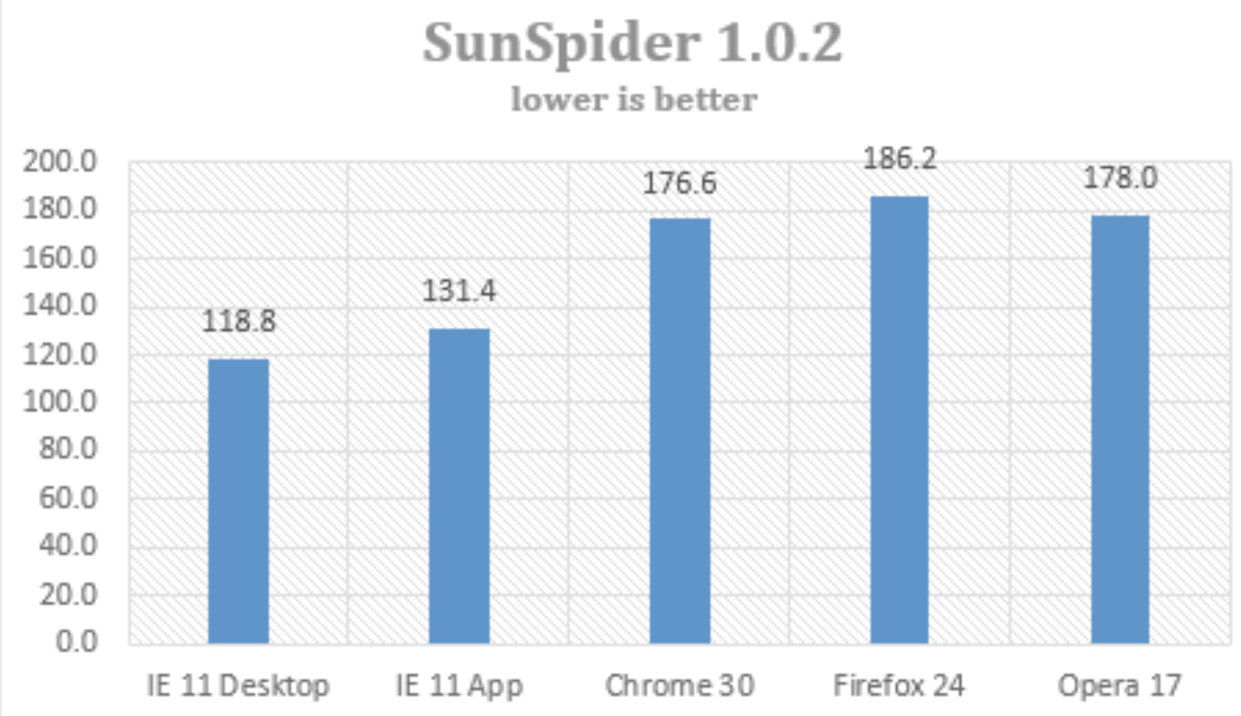
Milliseconds



Sept. 2008 http://ejohn.org/blog/javascript-performance-rundown/

JavaScript Performance 2013

Milliseconds



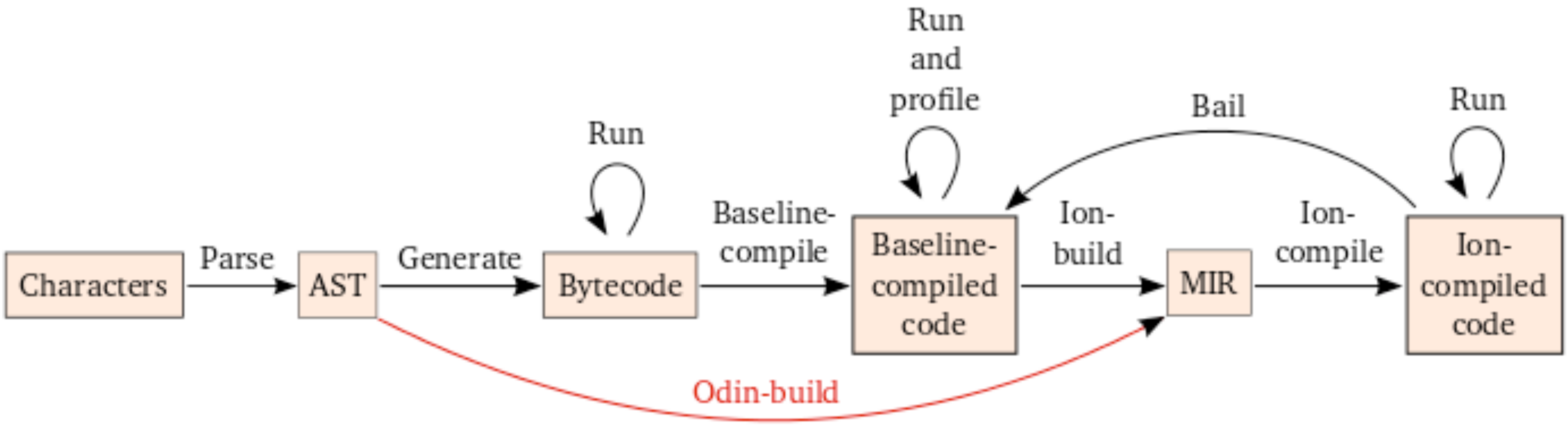
http://www.7tutorials.com/browser-wars-does-internet-explorer-11-deliver-browsing-performance

How did JavaScript get fast

- Smalltalk-inspired JIT technology
- Restarted the dynamic language VM innovation that stalled when commercial Smalltalks became legacy
  - ✓ Dynamic specialization based on
  - ✓ Runtime monitoring/tracing
  - ✓ Driving classic optimization algorithms
  - ✓ Multiple execution strategies

Large teams / multi-year development projects

A Modern JS Engine

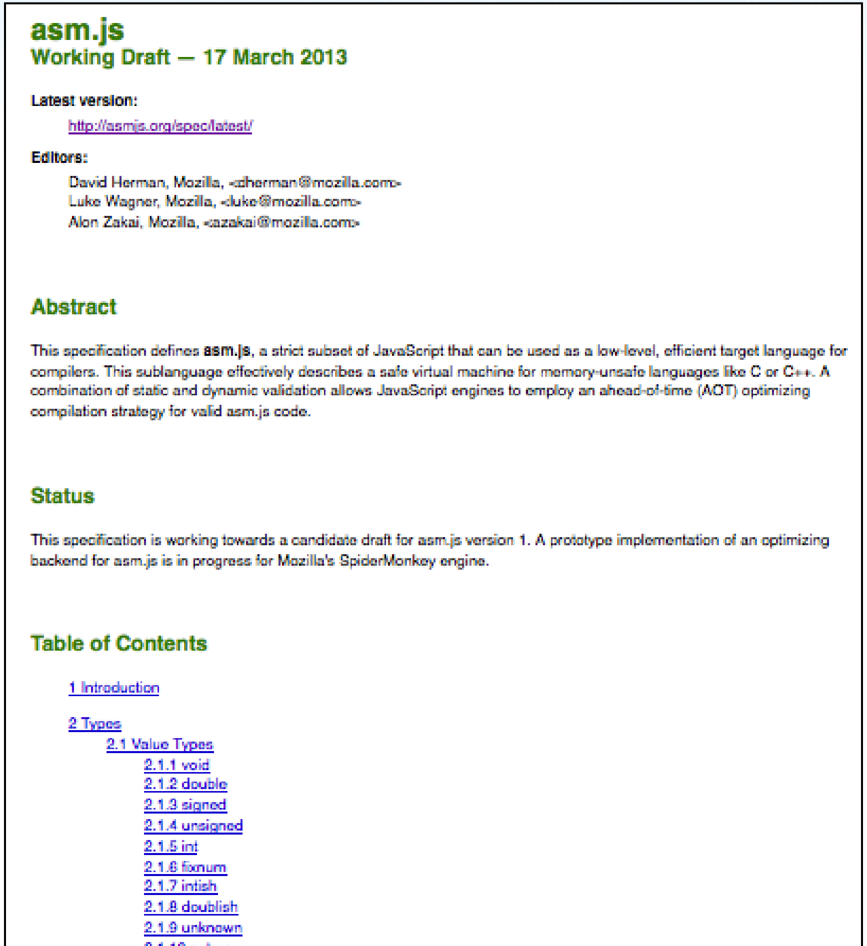


Mozilla SpiderMonkey circa 2014

https://blog.mozilla.org/luke/2014/01/14/asm-js-aot-compilation-and-startup-performance/

asm.js – C level Performance

- Subset of JavaScript that approximates a classic Von Neumann computer
- asm.js code executes identically on any JavaScript engine
- But a JS engine may recognize asm.js code and optimize for it.
- asmjs.org
- https://wiki.mozilla.org/Javascript:SpiderMonkey:OdinMonkey



asm.js
Working Draft — 17 March 2013

Latest version:

http://asmjs.org/spec/latest/

Editors:

David Herman, Mozilla, <dherman@mozilla.com>
Luke Wagner, Mozilla, <luke@mozilla.com>
Alon Zakai, Mozilla, <azakai@mozilla.com>

Abstract

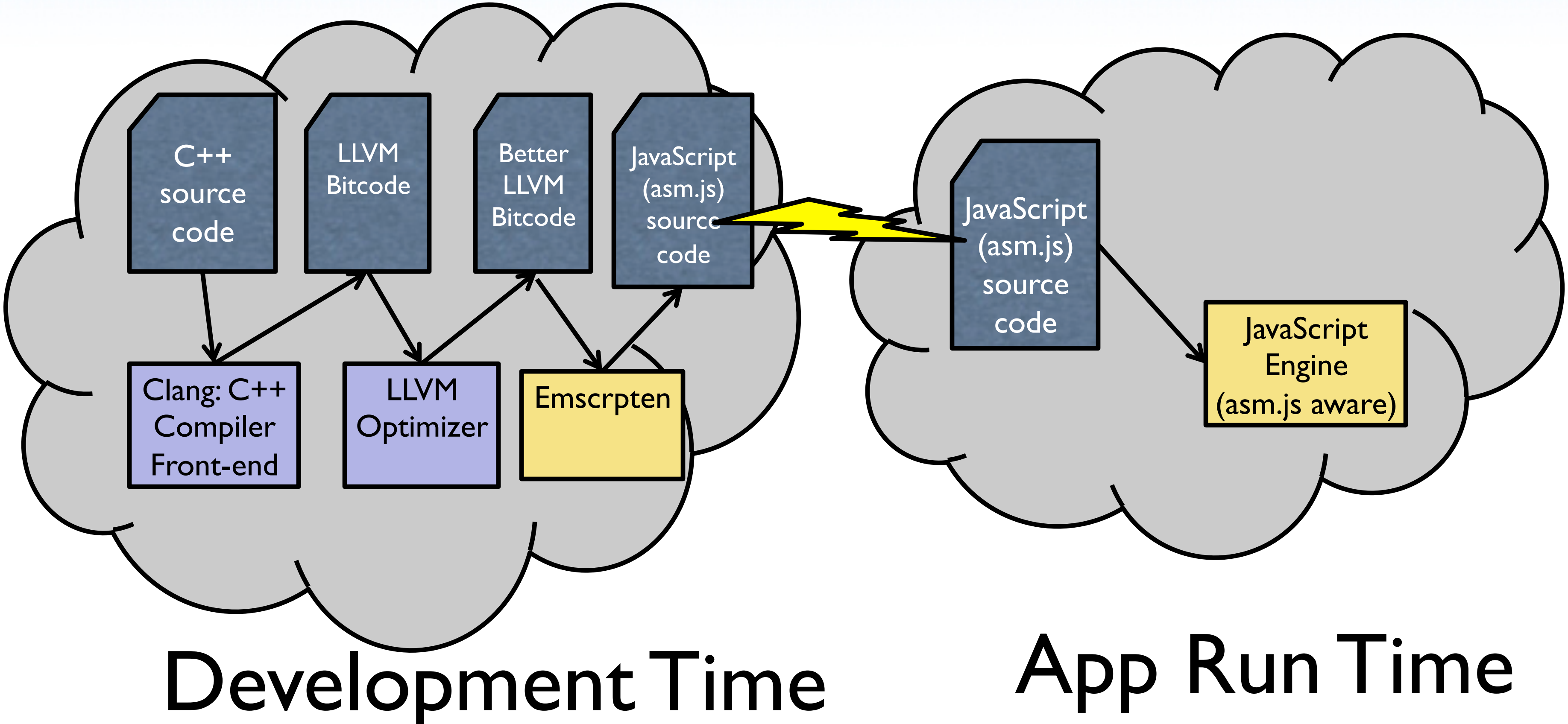
This specification defines **asm.js**, a strict subset of JavaScript that can be used as a low-level, efficient target language for compilers. This sublanguage effectively describes a safe virtual machine for memory-unsafe languages like C or C++. A combination of static and dynamic validation allows JavaScript engines to employ an ahead-of-time (AOT) optimizing compilation strategy for valid asm.js code.

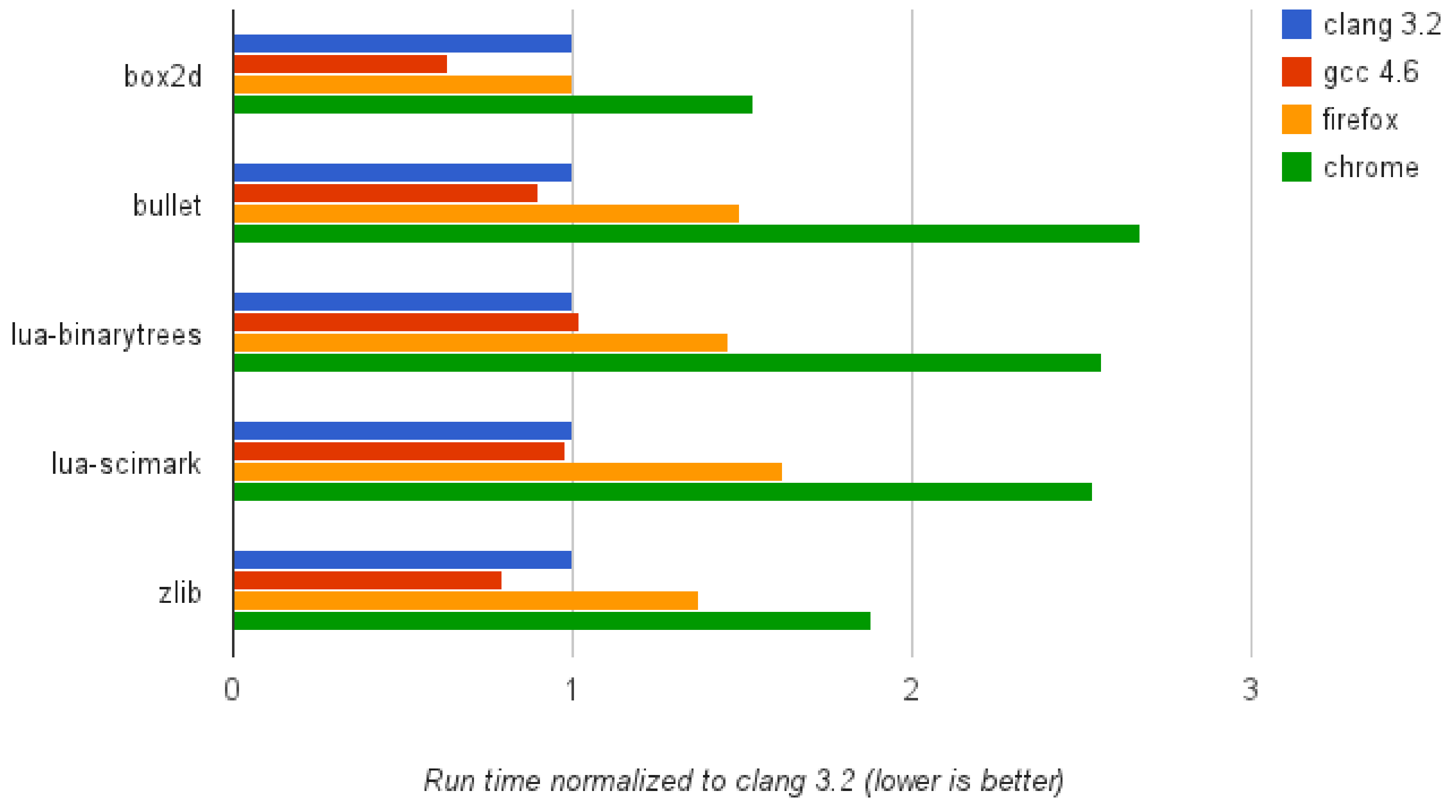
Status

This specification is working towards a candidate draft for asm.js version 1. A prototype implementation of an optimizing backend for asm.js is in progress for Mozilla's SpiderMonkey engine.

Table of Contents

C++ to JavaScript





people.mozilla.org/~lwagner/gdc-pres/gdc-2014.html

Unity game engine heading to the browser without plug-ins

WebGL and asm.js offer an alternative to the Web player.

by **Peter Bright** - Mar 18 2014, 11:15am PDT

56

Mozilla and Unity today announced that Unity 5, to be released later this year, will include an early access preview of a version of the 3D engine that supports WebGL and asm.js, enabling plug-in-free access to the Web.

The Unity game engine has found huge success among game developers as it can target Windows, iOS, Android, OS X, Linux, PlayStation 3, Xbox 360, and more. Unity games can also be deployed on the Web, but this function currently uses a browser plug-in, the Unity Web Player. The early access will remove the need for the plug-in. Initially, it will only support desktop Firefox and desktop Chrome, due to their performance and (in Firefox's case) explicit support for the high performance asm.js subset.

GDC 2014

Gallery: The experimental controllers and retro rarities of GDC 2014

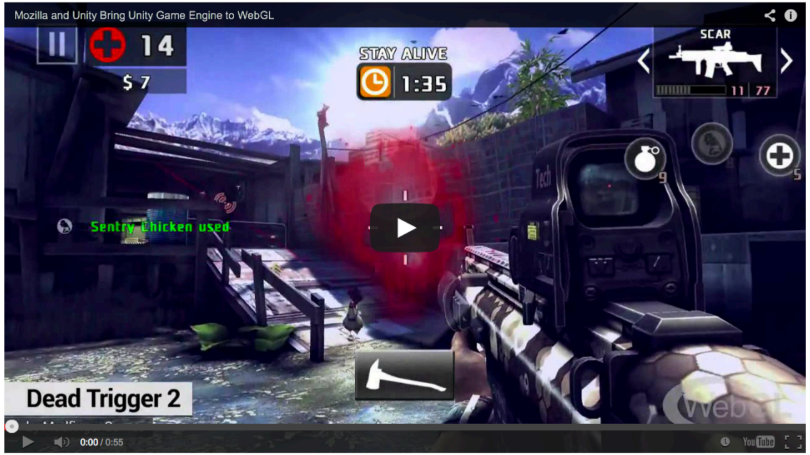
***Goat Simulator* preview: Goat of the year**

Steam Controller prototype version 2 impressions: Buttoned up

Project Morpheus impressions: Sony proves it's serious about virtual reality

Oculus Rift "DK2" eyes-on: Finally, VR without the Ocu-latency

View all...



A bunch of Unity games running in the browser with WebGL

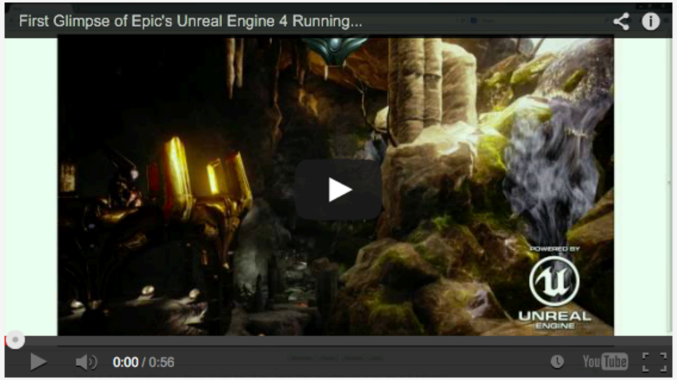
MAR 12 2014

Mozilla and Epic Preview Unreal Engine 4 Running in Firefox

Mozilla

0

Today, Epic Games and Mozilla are demonstrating how the Web is continuing to evolve as a powerful platform for gaming by providing a sneak peek of Epic's Soul and Swing Ninja demos, running in Firefox at near-native speeds. This video is the first glimpse of Unreal Engine 4 running on the Web without plugins.



Mozilla and Epic Games have showed the power of the Web as a platform for gaming by porting Unreal Engine 3 to the Web and showcasing Epic Citadel, using asm.js, a supercharged subset of JavaScript pioneered by Mozilla. In less than 12 months, optimizations have increased the performance of Web applications using asm.js from 40% to within 67% of native, and we expect it to get even faster. This performance opens up new opportunities for giving users an astonishing and delightful experience, from within their choice of Web browser. Any modern browser can run asm.js content, but specific optimizations currently present only in Firefox, ensure the most consistent and smooth experience.

Why not a web bytecode engine?

asm.js code is just YAIR

(Yet Another Intermediate Representation)

Worse is better!

```
function strlen([illegible]) {
    ptr = ptr|0;
    var curr = 0;
    curr = ptr;
    while (MEM8[cu[illegible]|0 != 0) {
      curr = (curr + [illegible]|0;
    }
    return (curr - ptr)|0;
}
```

```
[illegible]
[illegible]ptr, int[illegible]
local curr, int32, ptr
label loop
index8 indx, heap, curr
[illegible]
[illegible] exit
add curr, curr, [illegible]
goto loop
label exit
sub tmp, curr, ptr
return sub
```

ECMAScript Resources

The Official ECMAScript 5.1 Specification (HTML)

http://www.ecma-international.org/ecma-262/5.1/

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/

Please report bugs

http://bugs.ecmascript.org