Performance from Aligning Smalltalk & Javascript Classes

Dr. Dave Mason

Department of Computer Science
Ryerson University

Motivation

- building a dataflow environment (similar to this morning's talk) for the web
- Amber is very cool, but too slow and complex to interface with native objects
- curious how close one can make Javascript to Smalltalk

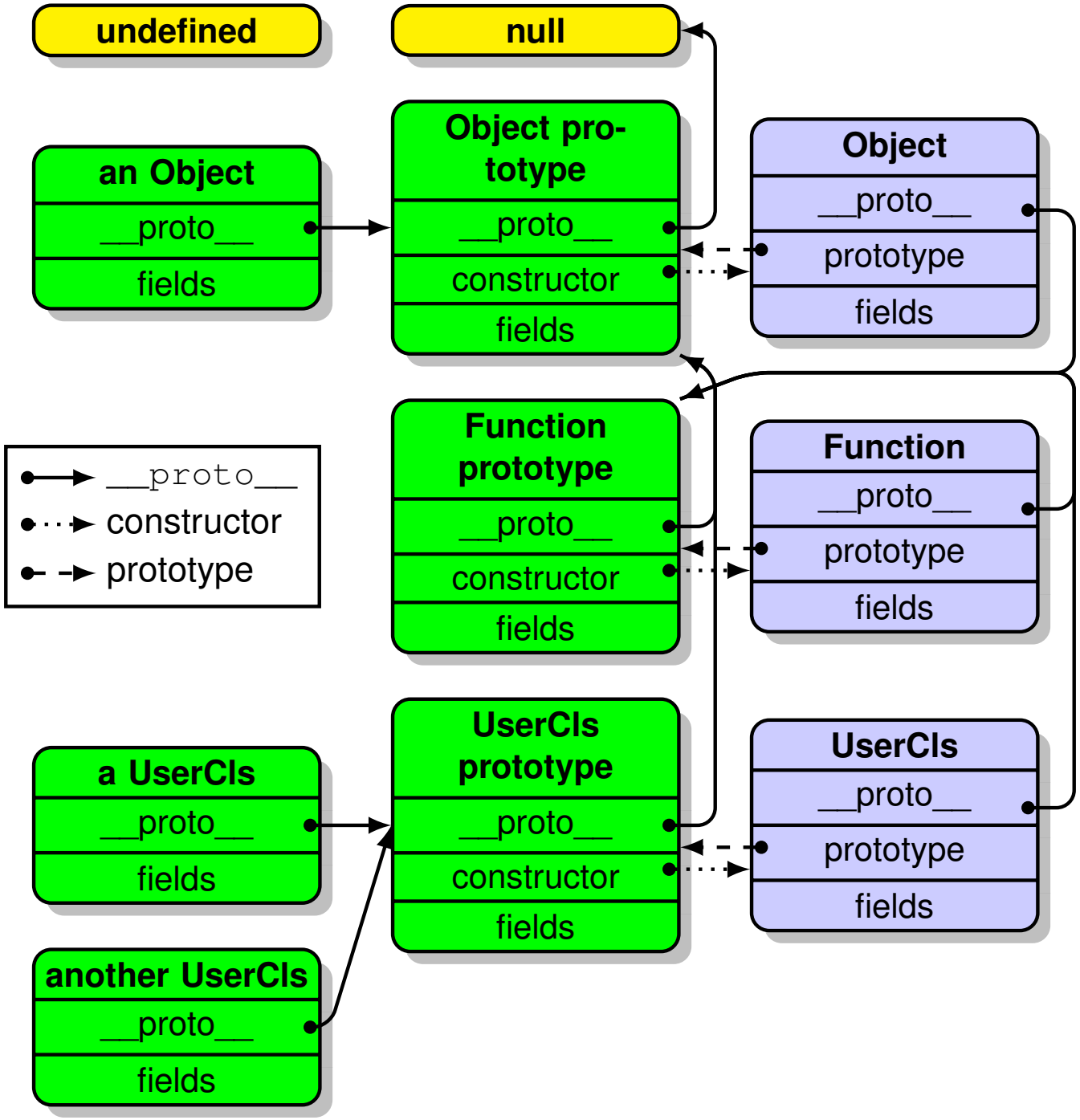
Motivation

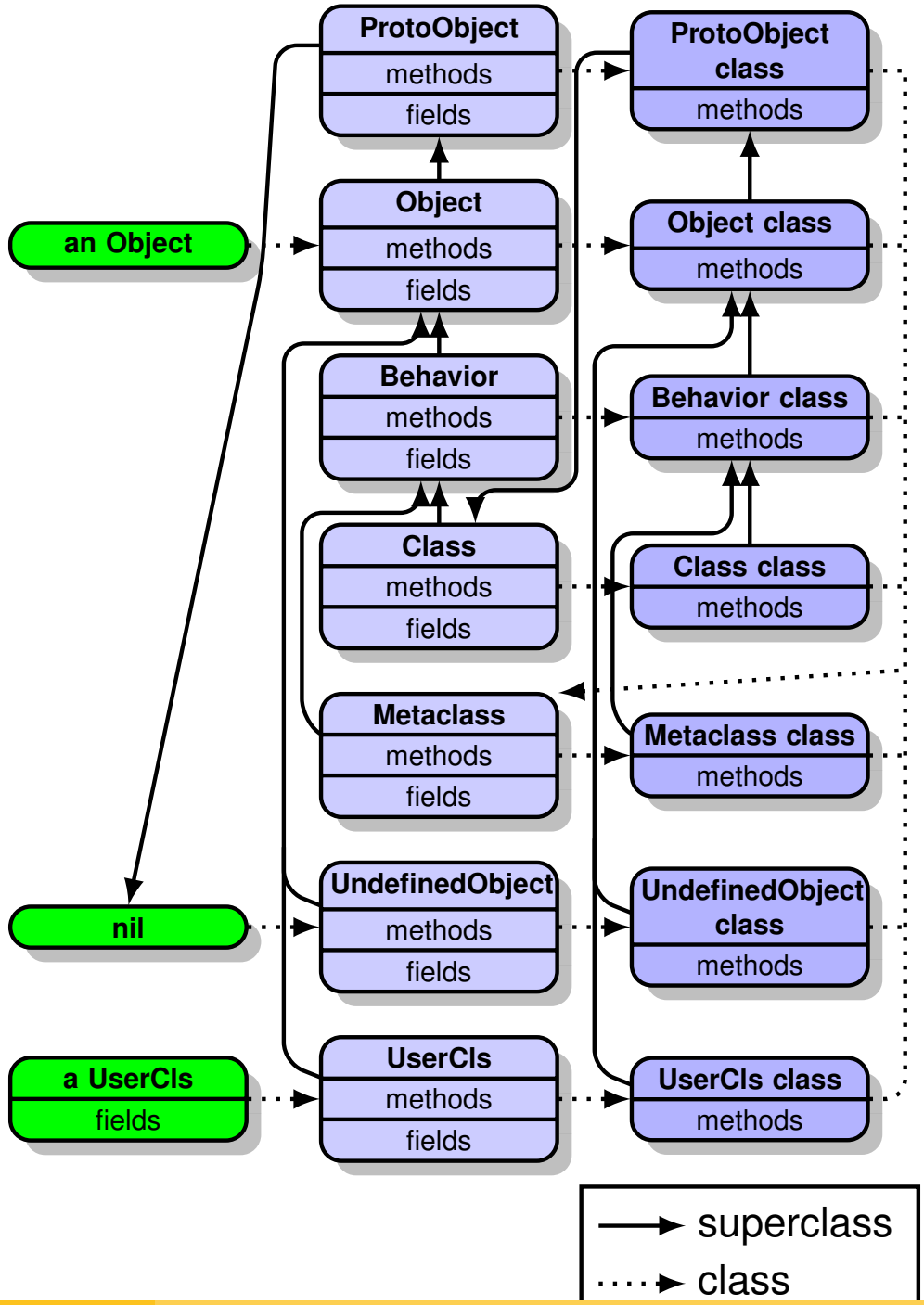
- building a dataflow environment (similar to this morning's talk) for the web
- Amber is very cool, but too slow and complex to interface with native objects
- curious how close one can make Javascript to Smalltalk

Motivation

- building a dataflow environment (similar to this morning's talk) for the web
- Amber is very cool, but too slow and complex to interface with native objects
- curious how close one can make Javascript to Smalltalk

```
1  // Animal
2  function Animal(name) {
3    this.name = name
4  }
5  Animal.prototype = {  // methods
6    canWalk: true,
7    sit: function() {
8      this.canWalk = false
9      alert(this.name + '␣sits␣down.')
10   }
11 }
12 // Rabbit
13 function Rabbit(name) {
14   this.name = name
15 }
16 Rabbit.prototype = inherit(Animal.prototype)
17 Rabbit.prototype.jump = function() {   // methods
18   this.canWalk = true
19   alert(this.name + '␣jumps!')
20 }
21 // Usage
22 var rabbit = new Rabbit('Sniffer')
23 rabbit.sit()   // Sniffer sits.
24 rabbit.jump()  // Sniffer jumps!
```


undefined
null
an Object
__proto__
fields
Object prototype
__proto__
constructor
fields
Object
__proto__
prototype
fields
__proto__
constructor
prototype
Function prototype
__proto__
constructor
fields
Function
__proto__
prototype
fields
a UserCls
__proto__
fields
another UserCls
__proto__
fields
UserCls prototype
__proto__
constructor
fields
UserCls
__proto__
prototype
fields


ProtoObject
methods
fields
ProtoObject class
methods
an Object
Object
methods
fields
Object class
methods
Behavior
methods
fields
Behavior class
methods
Class
methods
fields
Class class
methods
Metaclass
methods
fields
Metaclass class
methods
nil
UndefinedObject
methods
fields
UndefinedObject class
methods
a UserCls
fields
UserCls
methods
fields
UserCls class
methods
superclass
class

Amber

- instances of `Number` are represented by the Javascript *number* objects
- `Boolean`, and `Date` are similarly directly mapped
- `String`, `Symbol` and `Character` both use Javascript *strings*
- `OrderedCollection` is mapped to Javascript *arrays*
- to map to valid Javascript identifiers, message-names are prepended with _ and have every colon (`:`) replaced by _
- because Javascript identifiers have only a single look-up mechanism, instance variables are prepended with `@`, which means they need to be looked up via the indexing method (`obj['@foo']`) because `obj.@foo` is invalid syntax.

Amber

- instances of `Number` are represented by the Javascript *number* objects
- `Boolean`, and `Date` are similarly directly mapped
- `String`, `Symbol` and `Character` both use Javascript *strings*
- `OrderedCollection` is mapped to Javascript *arrays*
- to map to valid Javascript identifiers, message-names are prepended with _ and have every colon (`:`) replaced by _
- because Javascript identifiers have only a single look-up mechanism, instance variables are prepended with `@`, which means they need to be looked up via the indexing method (`obj['@foo']`) because `obj.@foo` is invalid syntax.

Amber

- instances of `Number` are represented by the Javascript *number* objects
- `Boolean`, and `Date` are similarly directly mapped
- `String`, `Symbol` and `Character` both use Javascript *strings*
- `OrderedCollection` is mapped to Javascript *arrays*
- to map to valid Javascript identifiers, message-names are prepended with _ and have every colon (`:`) replaced by _
- because Javascript identifiers have only a single look-up mechanism, instance variables are prepended with `@`, which means they need to be looked up via the indexing method (`obj['@foo']`) because `obj.@foo` is invalid syntax.

Amber

- instances of `Number` are represented by the Javascript *number* objects
- `Boolean`, and `Date` are similarly directly mapped
- `String`, `Symbol` and `Character` both use Javascript *strings*
- `OrderedCollection` is mapped to Javascript *arrays*
- to map to valid Javascript identifiers, message-names are prepended with _ and have every colon (`:`) replaced by _
- because Javascript identifiers have only a single look-up mechanism, instance variables are prepended with `@`, which means they need to be looked up via the indexing method (`obj['@foo']`) because `obj.@foo` is invalid syntax.

Amber

- instances of `Number` are represented by the Javascript *number* objects
- `Boolean`, and `Date` are similarly directly mapped
- `String`, `Symbol` and `Character` both use Javascript *strings*
- `OrderedCollection` is mapped to Javascript *arrays*
- to map to valid Javascript identifiers, message-names are prepended with _ and have every colon (`:`) replaced by _
- because Javascript identifiers have only a single look-up mechanism, instance variables are prepended with `@`, which means they need to be looked up via the indexing method (`obj['@foo']`) because `obj.@foo` is invalid syntax.

Amber

- instances of `Number` are represented by the Javascript *number* objects
- `Boolean`, and `Date` are similarly directly mapped
- `String`, `Symbol` and `Character` both use Javascript *strings*
- `OrderedCollection` is mapped to Javascript *arrays*
- to map to valid Javascript identifiers, message-names are prepended with _ and have every colon (`:`) replaced by _
- because Javascript identifiers have only a single look-up mechanism, instance variables are prepended with `@`, which means they need to be looked up via the indexing method (`obj['@foo']`) because `obj.@foo` is invalid syntax.

Semantic Map

- Everything is an object. *null, undefined*
- `doesNotUnderstand:` message to the object. *undefined* $\rightarrow$ *exception*
- Boolean only:`true` and `false`; otherwise signal a `mustBeBoolean` error.
- `self` refers to the current object and `super` refers to the current object, but with method resolution starting with the superclass of the current code.
- A return from a Smalltalk block returns from the method in which the block is statically defined.

Semantic Map

- Everything is an object. *null, undefined*
- `doesNotUnderstand:` message to the object. *undefined* $\rightarrow$ *exception*
- Boolean only:`true` and `false`; otherwise signal a `mustBeBoolean` error.
- `self` refers to the current object and `super` refers to the current object, but with method resolution starting with the superclass of the current code.
- A return from a Smalltalk block returns from the method in which the block is statically defined.

Semantic Map

- Everything is an object. *null, undefined*
- `doesNotUnderstand:` message to the object. *undefined* $\rightarrow$ *exception*
- Boolean only:`true` and `false`; otherwise signal a `mustBeBoolean` error.
- `self` refers to the current object and `super` refers to the current object, but with method resolution starting with the superclass of the current code.
- A return from a Smalltalk block returns from the method in which the block is statically defined.

Semantic Map

- Everything is an object. *null, undefined*
- `doesNotUnderstand:` message to the object. *undefined* $\rightarrow$ *exception*
- Boolean only:`true` and `false`; otherwise signal a `mustBeBoolean` error.
- `self` refers to the current object and `super` refers to the current object, but with method resolution starting with the superclass of the current code.
- A return from a Smalltalk block returns from the method in which the block is statically defined.

Semantic Map

- Everything is an object. *null, undefined*
- `doesNotUnderstand:` message to the object. *undefined* $\rightarrow$ *exception*
- Boolean only:`true` and `false`; otherwise signal a `mustBeBoolean` error.
- `self` refers to the current object and `super` refers to the current object, but with method resolution starting with the superclass of the current code. In Javascript, `this` refers to the object from-which the name lookup was done that lead to the current function executing.
- A return from a Smalltalk block returns from the method in which the block is statically defined.

Semantic Map

- Everything is an object. *null, undefined*
- `doesNotUnderstand:` message to the object. *undefined* $\rightarrow$ *exception*
- Boolean only:`true` and `false`; otherwise signal a `mustBeBoolean` error.
- `self` refers to the current object and `super` refers to the current object, but with method resolution starting with the superclass of the current code.

- A return from a Smalltalk block returns from the method in which the block is statically defined.

Semantic Map

- A return from a Smalltalk block returns from the method in which the block is statically defined.

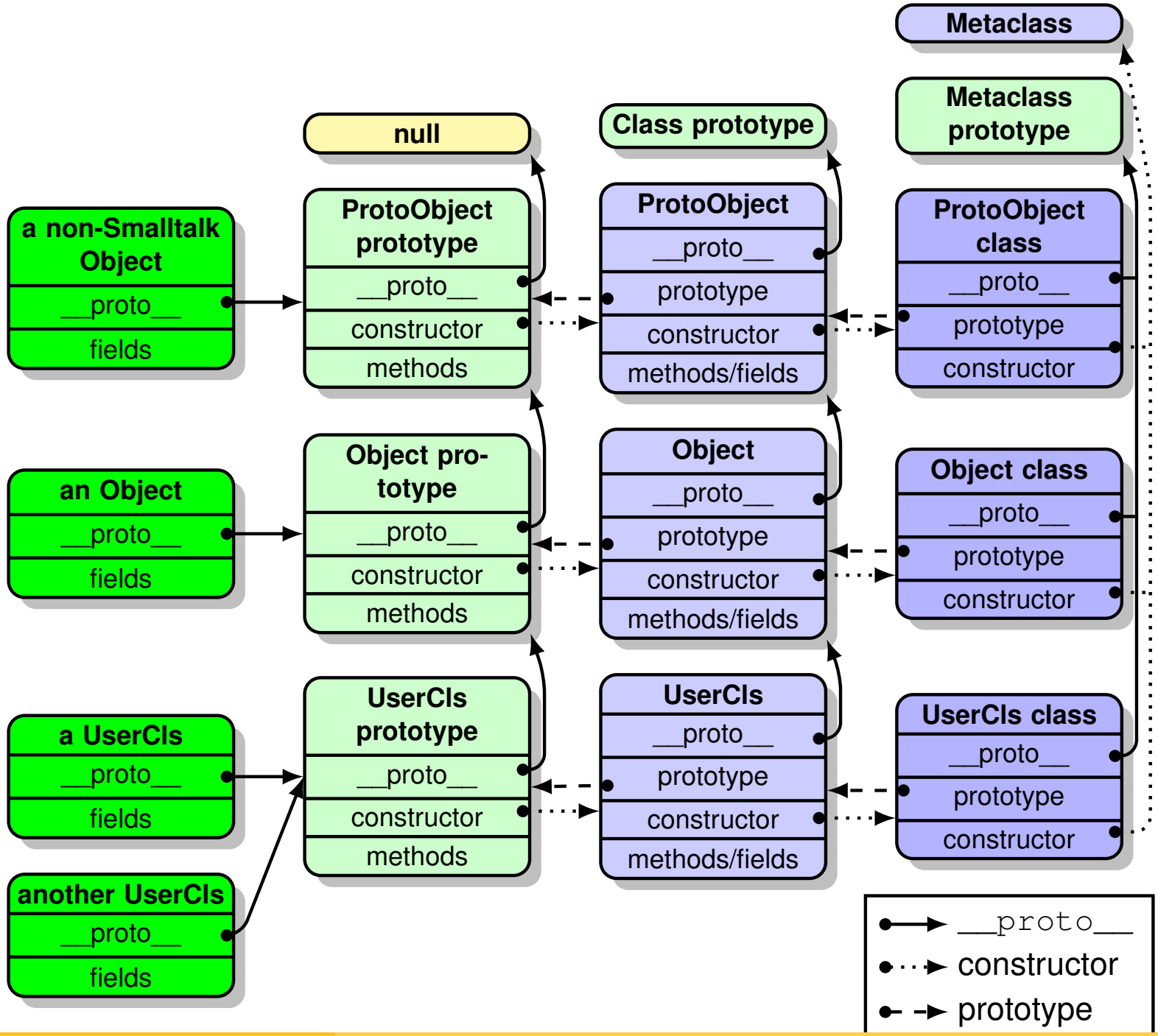
```
1  foo: n
2     n timesRepeat: [
3        ^n
4     ].
5     ^ -1
```

Semantic Map

- A return from a Smalltalk block returns from the method in which the block is statically defined.

```
1  foo: n
2     n timesRepeat: [
3        ^n
4     ].
5     ^ -1
```

```
1   function _foo_(n) {
2    var $exit=Object.create(null);
3    try {
4      n._timesRepeat_(function(){throw n;})
5      } catch ($e) {if ($e===$exit) return $e;throw $e}
6  }
```


Metaclass
Metaclass prototype
null
Class prototype
a non-Smalltalk Object
__proto__
fields
ProtoObject prototype
__proto__
constructor
methods
ProtoObject
__proto__
prototype
constructor
methods/fields
ProtoObject class
__proto__
prototype
constructor
an Object
__proto__
fields
Object pro-totype
__proto__
constructor
methods
Object
__proto__
prototype
constructor
methods/fields
Object class
__proto__
prototype
constructor
a UserCls
__proto__
fields
UserCls prototype
__proto__
constructor
methods
UserCls
__proto__
prototype
constructor
methods/fields
UserCls class
__proto__
prototype
constructor
another UserCls
__proto__
fields
__proto__
constructor
prototype

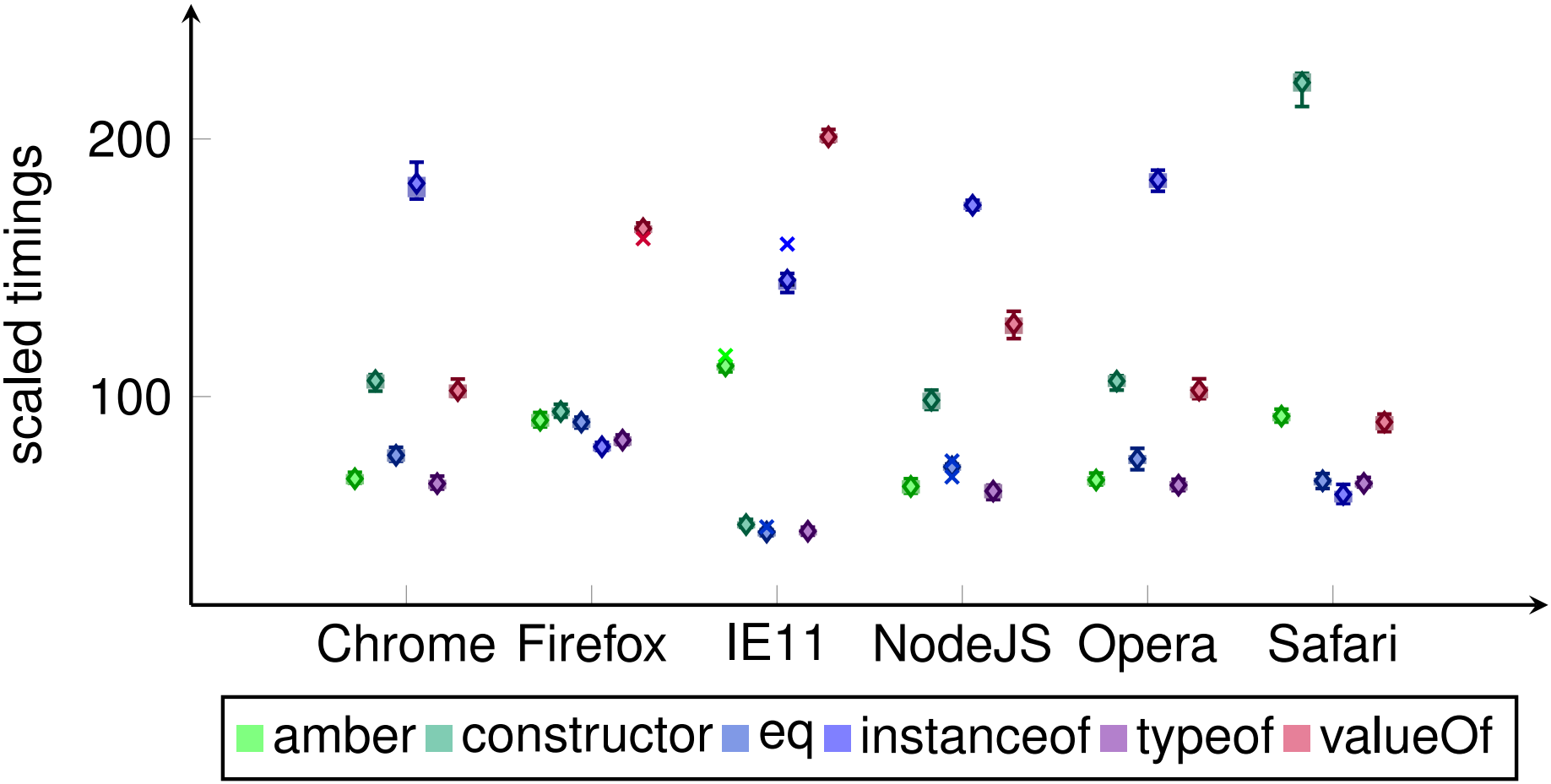
Browser	Test Version	Min. Version	Release Date
Chrome	43.0.2357.65	34	2014-04
Firefox	38.0.1	31	2014-07
Internet-Explorer	11.0.19	11	2013-10
NodeJS	0.12.2	0.10?	2013
Opera	29.0	13	2013
Safari	8.0.6	7.0?	2013

JS Engines supporting `.setPrototypeOf`

Microbenchmarks

- run on an idle Apple Macbook Pro, with 2.5GHz Intel i5 processor
- IE11 on virtual machine
- at least 100,000 operations per “run”
- 10 warmup runs discarded before 10 captured runs

amber	`asBool(v)`
constructor	`(v.constructor===Boolean?v:v.nonbool())`
eq	`(v===true||(v===false?v:v.nonbool()))`
instance	`(v instanceof Boolean?v:v.nonbool())`
typeOf	`(typeof v==="boolean"?v:v.nonbool())`
valueOf	`Boolean.prototype.valueOf.call(v)`

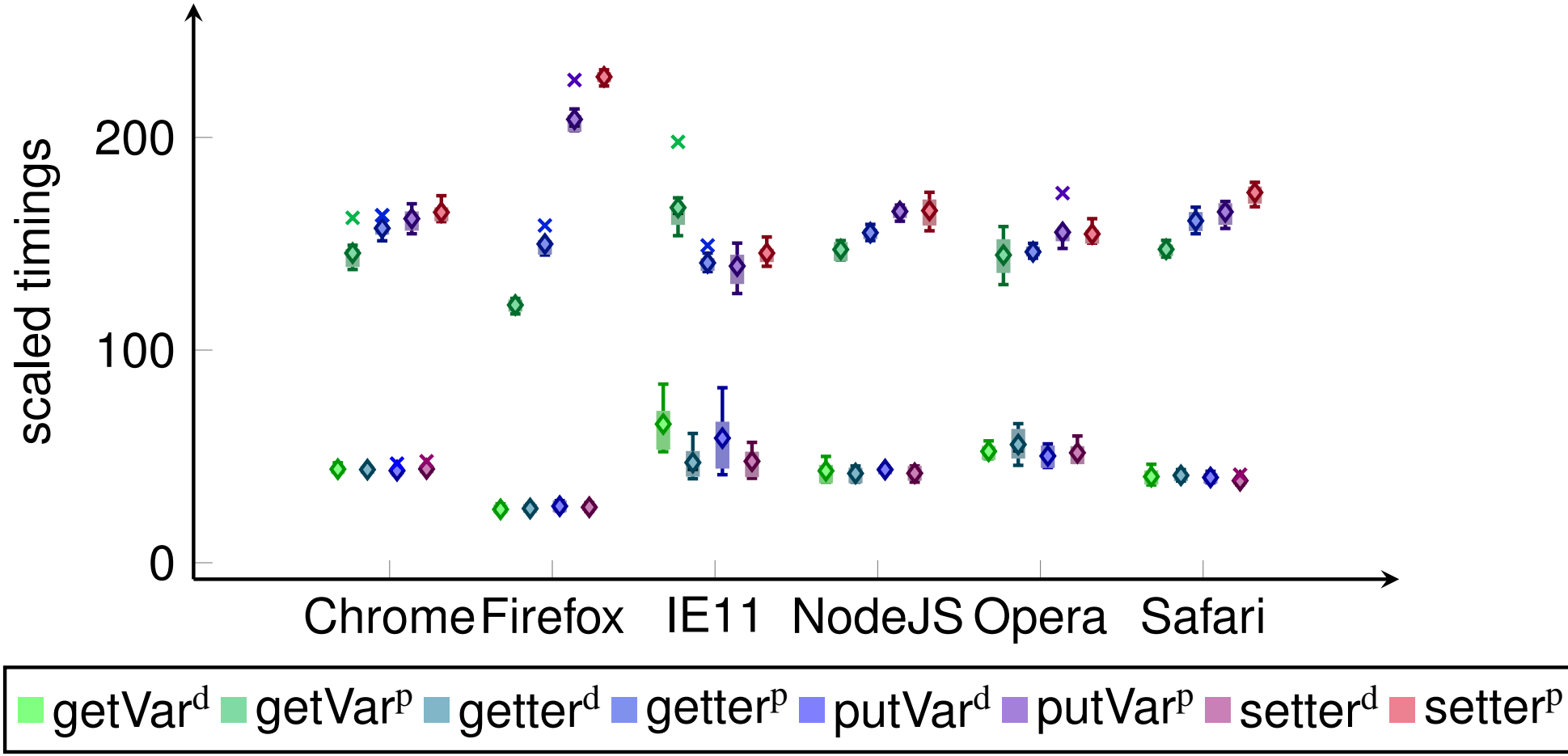


null, undefined and doesNotUnderstand:

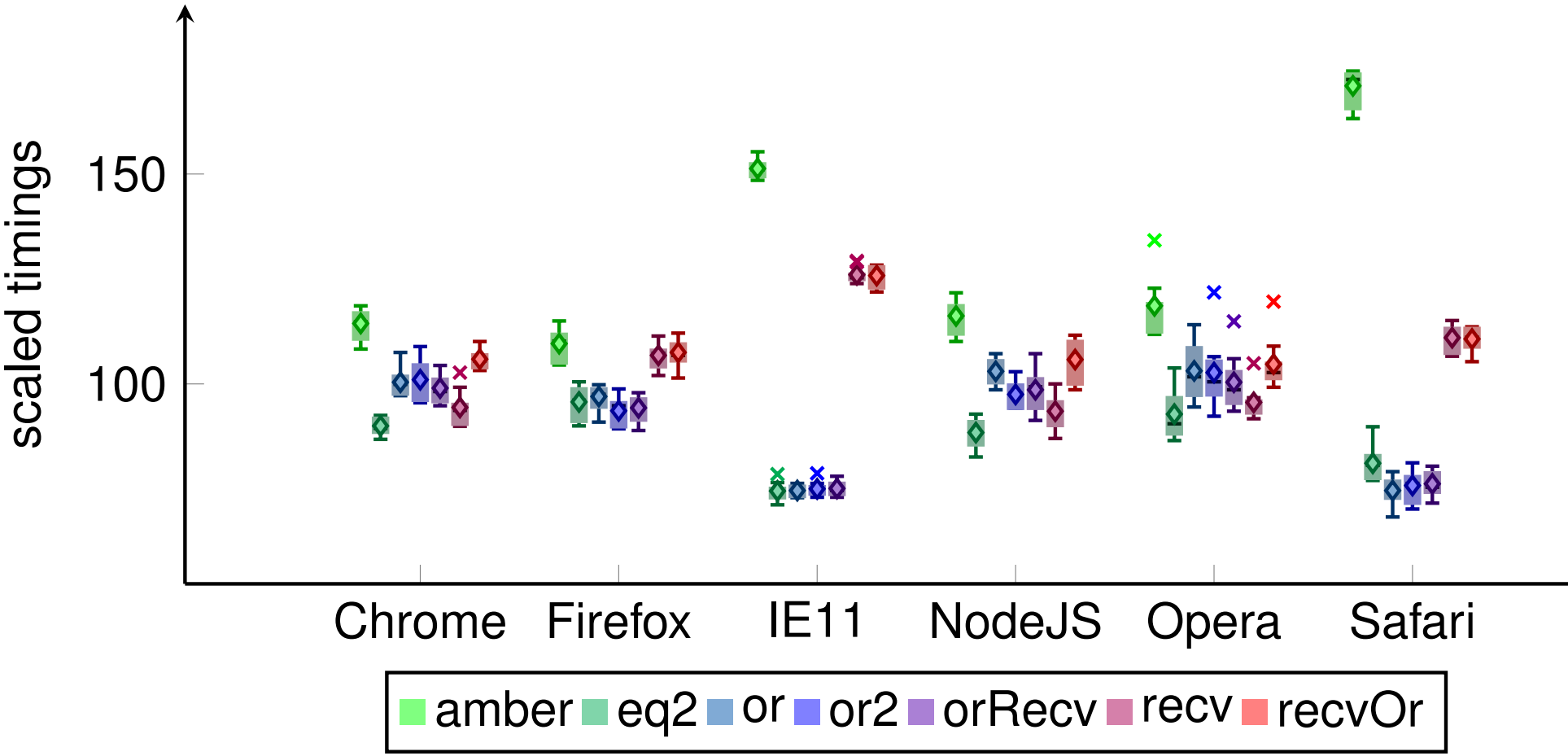
```
1 function $recv(o){
2     if (o == null) return nil;
3     if (typeof o === "object" ||
4         typeof o === "function") {
5           return o.klass != null ? o :
6               globals.JSObjectProxy._on_(o);
7     }
8     return o;
9 }
```

```
1  function MyObject() {
2     this.abc=4;
3  }
4  MyObject.prototype.getter=function getter() {...};
5  MyObject.prototype.setter=function setter(x) {...};
```

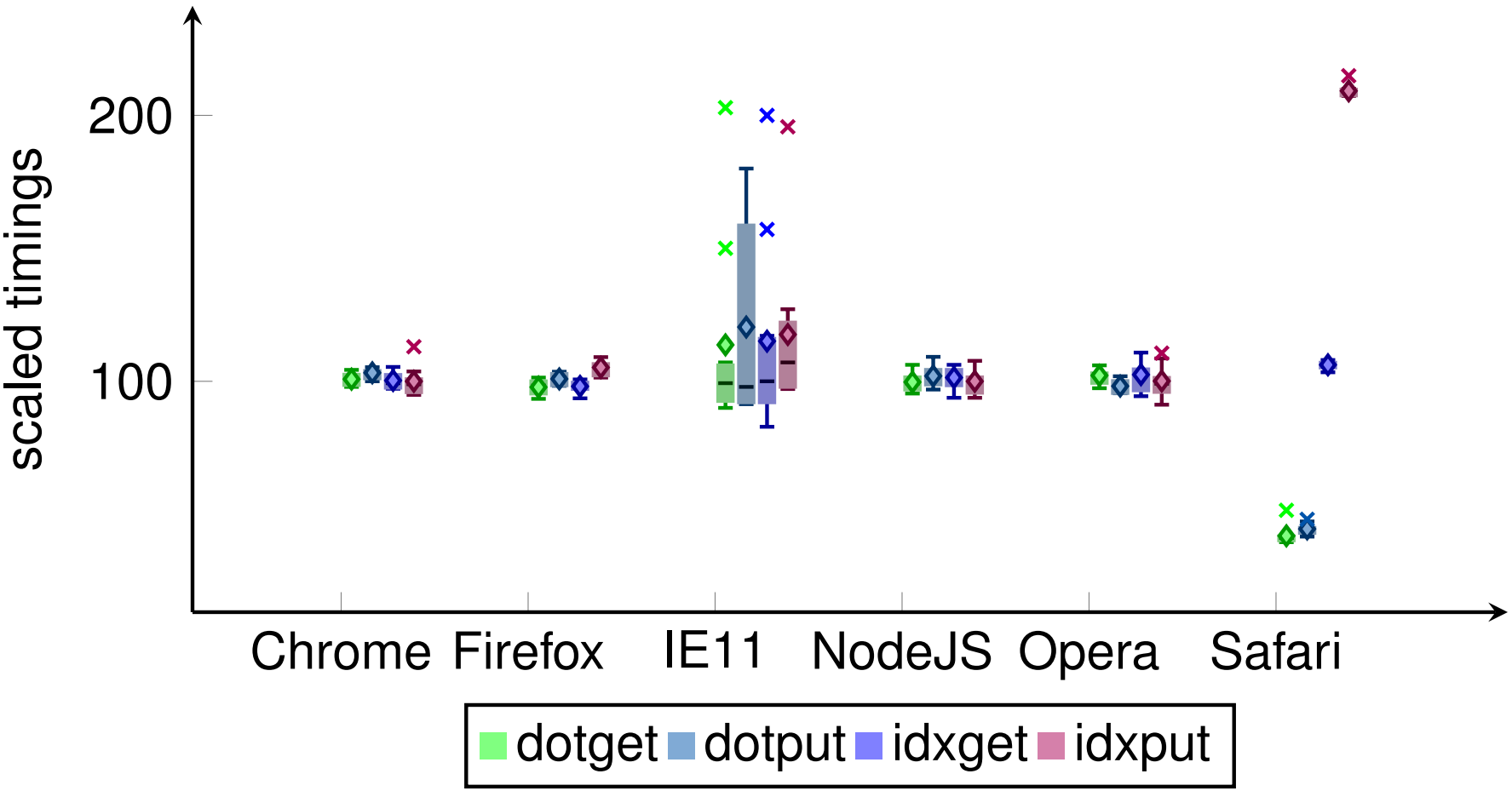
```
1 getvar: obj for: n
2    | tot |
3    tot := 0.
4    n timesRepeat: [
5       tot := tot + obj abc
6    ].
7    ^ tot
```


scaled timings
200
100
0
Chrome
Firefox
IE11
NodeJS
Opera
Safari
getVar^d
getVar^p
getter^d
getter^p
putVar^d
putVar^p
setter^d
setter^p

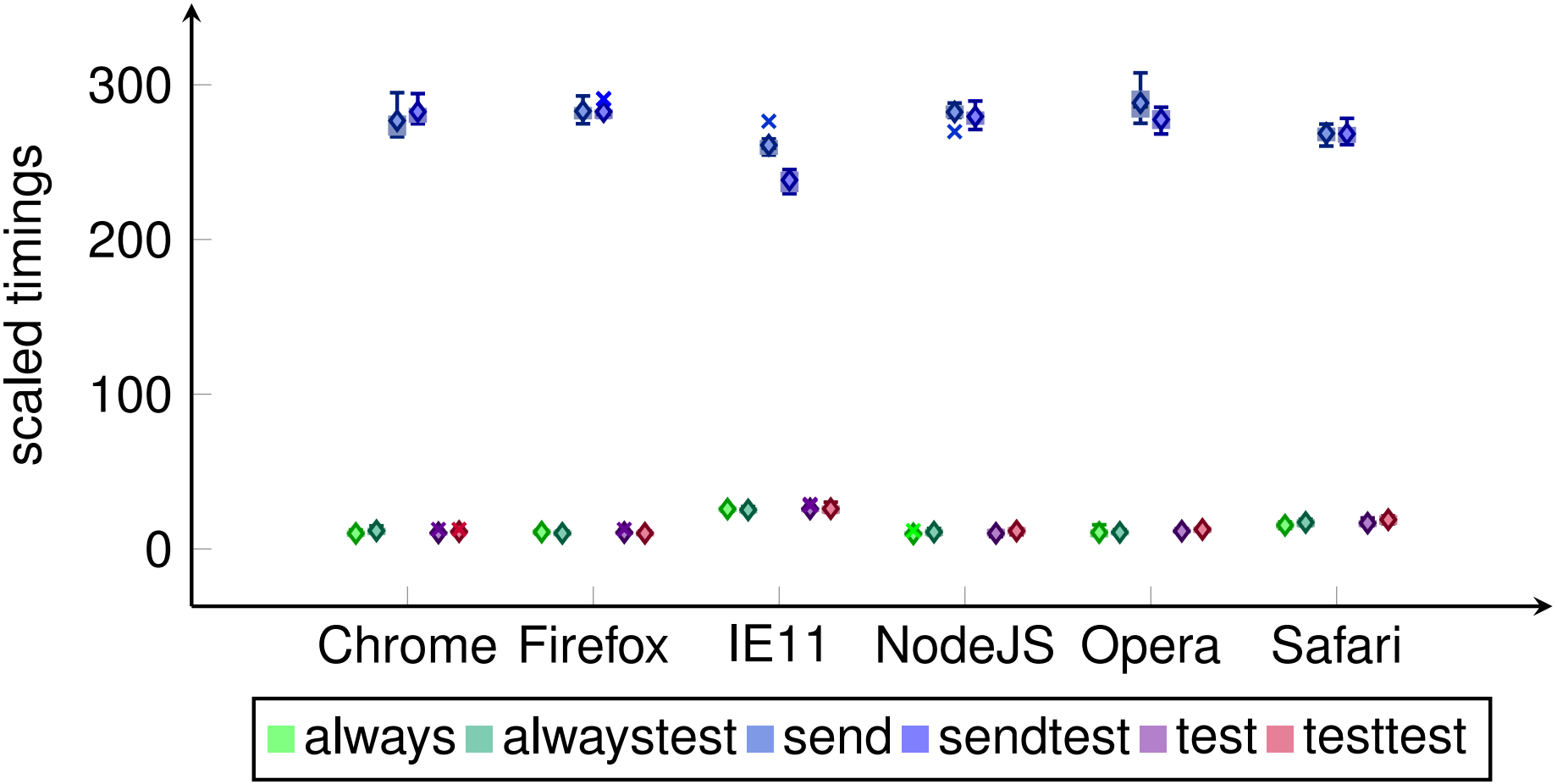
amber	`$amber(f).foo()`
eq2	`(f==null?nil:f).foo()`
or	`(f||(f==null?nil:f)).foo()`
or2	`(f||(f===undefined||f===null?nil:f)).foo()`
or3	`(f||(f===null||f===undefined?nil:f)).foo()`
or4	`(f||(f===null?nil:f===undefined?nil:f)).foo()`
recv	`$recv(f).foo()`
recvOr	`$recvOr(f).foo()`
orRecv	`(f||$recv(f)).foo()`



`dotget`	`o.ghi`
`dotput`	`o.ghi=17`
`idxget`	`o['ghi']`
`idxput`	`o['ghi']=17`



always	`n+m`
alwaystes	`typeof m==="number"?n+m:m.adaptN(n,'+')`
send	`n.__plus_(m)`
sendtest	`n.__plusT_(m)`
test	`typeof n==="number"?n+m:n.__plus_(m)`
testtest	`typeof n==="number"?typeof m==="number"?n+m:m.adaptN(n,'+'):n.__plusT_(m)`



Conclusions

- can make Javascript align well with Smalltalk
- can get close to native Javascript performance
- some surprising performance characteristics

Roadmap

- information for Amber or PharoJS
- currently working on streamlining Amber JS-object handling
- number-handling is next priority

.

Questions?

Thanks to Nicholas Petton, Herbert Vojčík and many others for Amber