Real-World JavaScript Performance

Toon Verwaest
Camillo Bruni

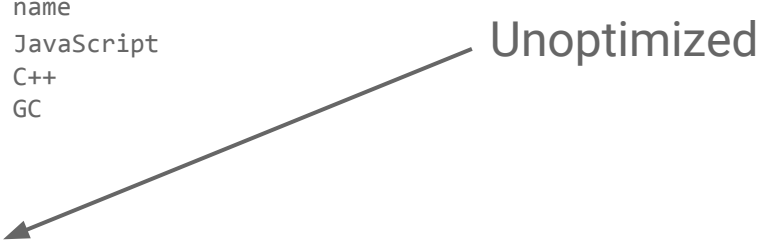
```
var _sunSpiderStartDate = new Date();

bitwiseAndValue = 4294967296;
for (var i = 0; i < 600000; i++)
     bitwiseAndValue = bitwiseAndValue & i;

var _sunSpiderInterval = new Date() - _sunSpiderStartDate;
record(_sunSpiderInterval);
```

The web is evolving and so should the JavaScript benchmarks that measure its performance. Today, we are releasing *Octane*, a JavaScript benchmark suite that aims to measure a browser's performance when running the *complex and demanding web applications that users interact with daily.*

Most of the existing JavaScript benchmarks run artificial tests that were created on an ad-hoc basis to stress a specific JavaScript feature. Octane breaks with this tradition and extends the former V8 Benchmark Suite with 5 new benchmarks created from *full, unaltered, well-known web applications and libraries*. A high score in the new benchmarks *directly translates to better and smoother performance in similar web applications*.

***Octane**: the JavaScript benchmark suite for the modern web, Chromium Blog, 2012*

```
[Summary]:
  ticks  total    nonlib   name
    828   85.4%    86.2%   JavaScript
    125   12.9%    13.0%   C++
     40    4.1%     4.2%   GC

[JavaScript]:
   ticks  total  nonlib
    144   14.9%   15.0%  ~L.SolveVelocityConstraints box2d.js:288:370
     87    9.0%    9.1%  Stub: BinaryOpICStub
     47    4.9%    4.9%  Stub: BinaryOpICStub {2}
     29    3.0%    3.0%  Stub: LoadFieldStub
     29    3.0%    3.0%  Stub: BinaryOpICStub {1}
     27    2.8%    2.8%  ~L.SolvePositionConstraints box2d.js:294:65
     26    2.7%    2.7%  ~u.Initialize box2d.js:302:292
     26    2.7%    2.7%  CallFunction_ReceiverIsNotNullOrUndefined
     22    2.3%    2.3%  ~k.SynchronizeTransform box2d.js:208:81
```



Score: 6009

Google

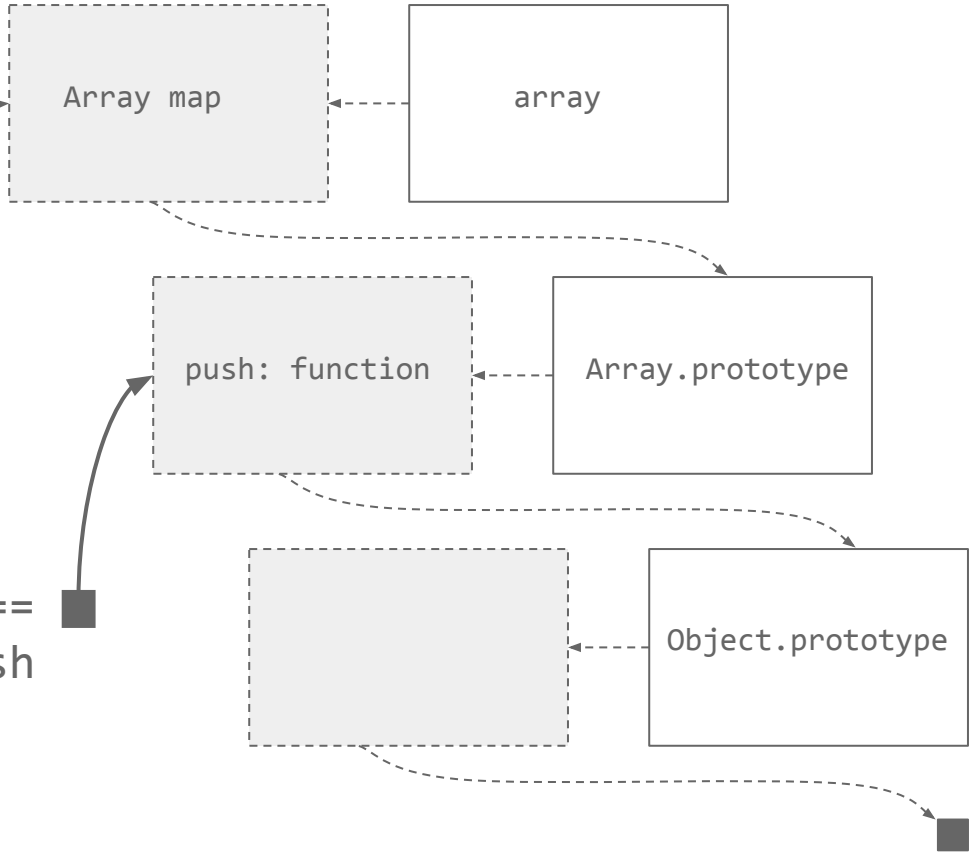
```
array.push(1)
```

```
this[this.length] = 1
```

```
function ArrayPush() {
  // Check for null or undefined.
  CHECK_OBJECT_COERCIBLE(this, "Array.prototype.push")

  var array = TO_OBJECT(this)
  var n = TO_LENGTH(array.length)
  var m = arguments.length

  if (m > 2^53 - 1 - n) throw new TypeError

  for (var i = 0; i < m; i++) {
    array[i+n] = arguments[i]
  }

  var new_length = n + m
  array.length = new_length
  return new_length
}
```

```
array.push(1)
```

```
if array.map ==
    && array.prototype.map ==
  return Array.prototype.push
```

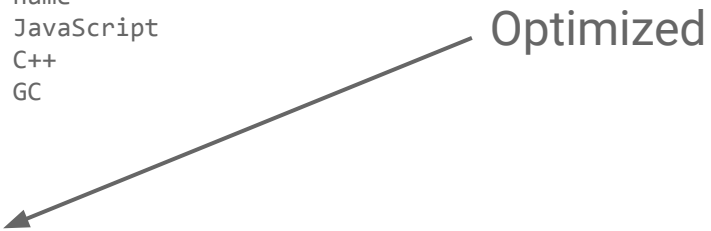


compile-time prechecks

this[this.length] = value

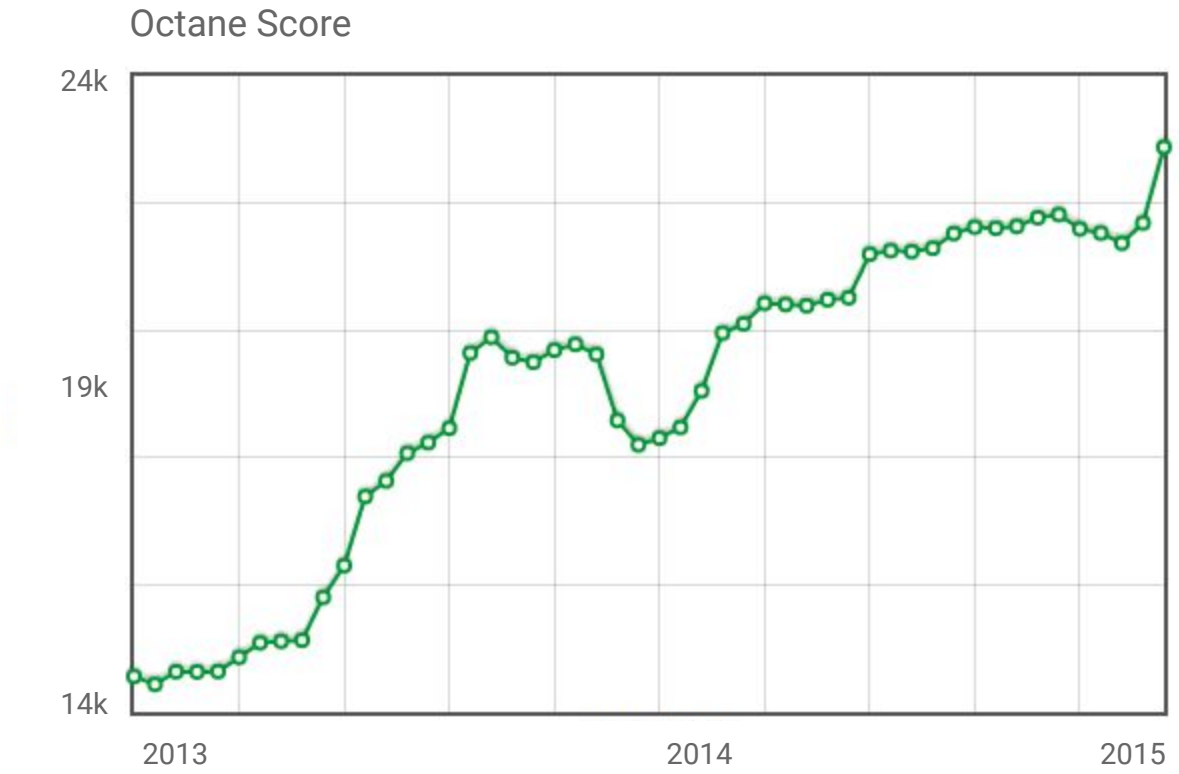
```
case kArrayPush: {
  if (!CanInlineArrayResizeOperation(receiver_map)) return false;
  ElementsKind elements_kind = receiver_map->elements_kind();

  // If there may be elements accessors in the prototype chain, the fast
  // inlined version can't be used.
  if (receiver_map->DictionaryElementsInPrototypeChainOnly()) return false;
  Handle<JSObject> prototype(JSObject::cast(receiver_map->prototype()));
  BuildCheckPrototypeMaps(prototype, Handle<JSObject>());

  // Protect against adding elements to the Array prototype, which needs to
  // route through appropriate bottlenecks.
  if (isolate()->IsFastArrayConstructorPrototypeChainIntact() &&
      !prototype->IsJSArray()) {
    return false;
  }

  const int argc = args_count_no_receiver;
  if (argc != 1) return false;

  HValue* value_to_push = Pop();
  HValue* array = Pop();
  Drop(1);  // Drop function.

  HInstruction* new_size = NULL;
  HValue* length = NULL;

  {
    NoObservableSideEffectsScope scope(this);

    length = Add<HLoadNamedField>(
        array, nullptr, HObjectAccess::ForArrayLength(elements_kind));
    new_size = AddUncasted<HAdd>(length, graph()->GetConstant1());
    bool is_array = receiver_map->instance_type() == JS_ARRAY_TYPE;
    HValue* checked_array = Add<HCheckMaps>(array, receiver_map);
    BuildUncheckedMonomorphicElementAccess(
        checked_array, length, value_to_push, is_array, elements_kind,
        STORE, NEVER_RETURN_HOLE, STORE_AND_GROW_NO_TRANSITION);
    if (!ast_context()->IsEffect()) Push(new_size);
    Add<HSimulate>(ast_id, REMOVABLE_SIMULATE);
    if (!ast_context()->IsEffect()) Drop(1);
  }
  ast_context()->ReturnValue(new_size);
  return true;
}
```

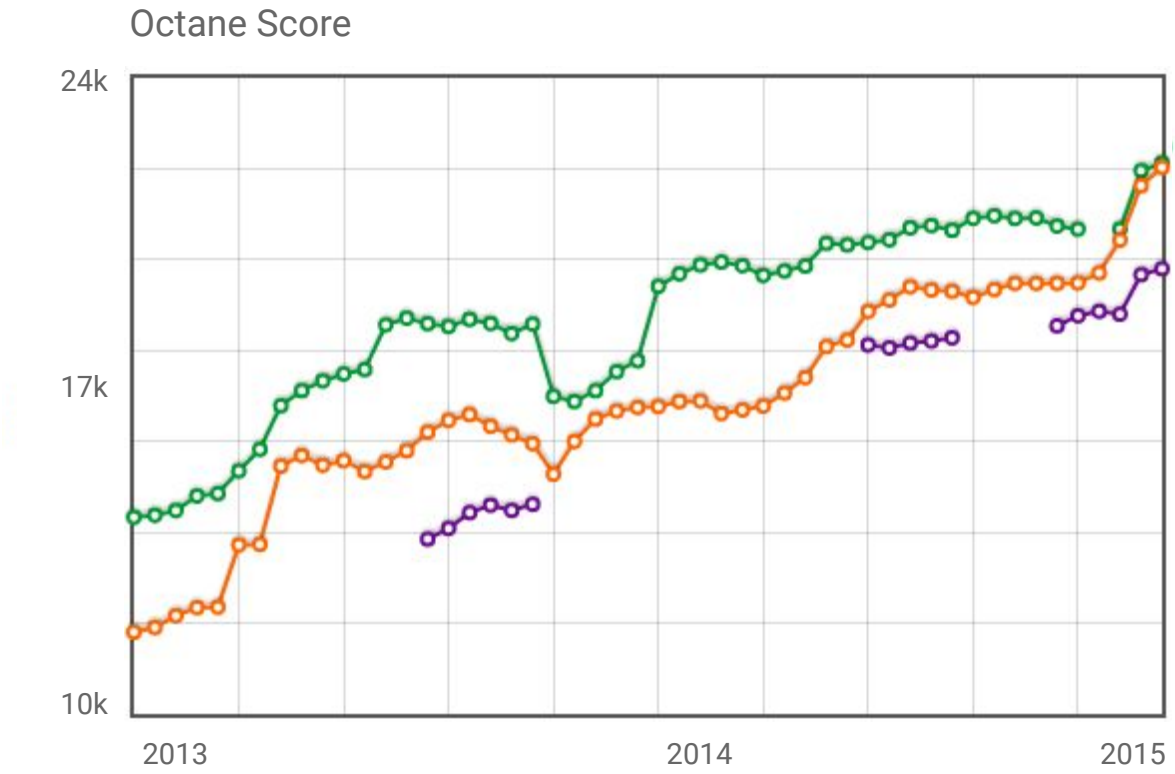
```
[Summary]:
  ticks  total    nonlib   name
   688   71.7%     72.4%   JavaScript
   252   26.3%     26.5%   C++
    48    5.0%      5.1%   GC

[JavaScript]:
   ticks  total  nonlib
    185   19.3%   19.5%  *L.SolveVelocityConstraints box2d.js:288:370
     79    8.2%    8.3%  *L.SolvePositionConstraints box2d.js:294:65
     41    4.3%    4.3%  *L.Initialize box2d.js:282:404
     33    3.4%    3.5%  *M.EdgeSeparation box2d.js:58:9
     32    3.3%    3.4%  *k.SynchronizeTransform box2d.js:208:81
     29    3.0%    3.1%  *M.CollidePolygons box2d.js:62:303
     21    2.2%    2.2%  Builtin: ArgumentsAdaptorTrampoline
```



Score: 51508

Google


Octane Score
24k
19k
14k
2013
2014
2015


Octane Score
24k
17k
10k
2013
2014
2015


Ad

What if the benchmarks aren't representative of actual workloads?

Compiler writes would be steered towards implementing optimizations that have little effect in the real world.

An Analysis of the Dynamic Behavior of JavaScript Programs, PLDI 2010

What makes a good benchmark? (stable, reproducible, single number)

… for VM developers? (priorities, meaningful insights)

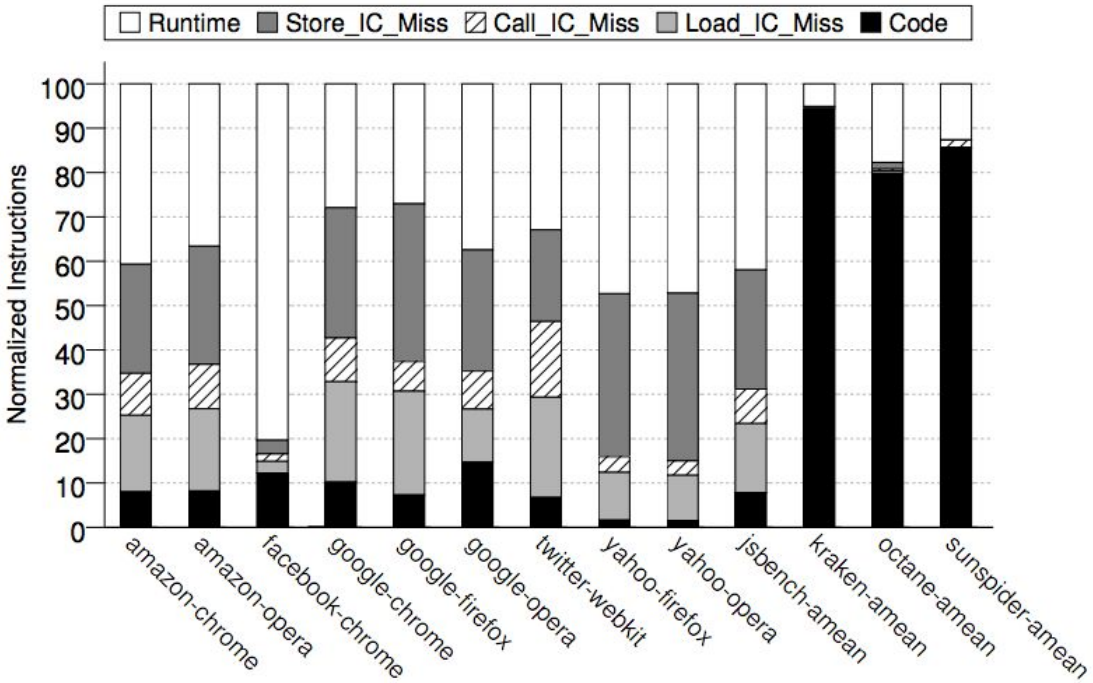
… for “benchmarketing”? (cross-browser, not gameable)

… for application developers? (represents patterns I want to use, cross-browser)

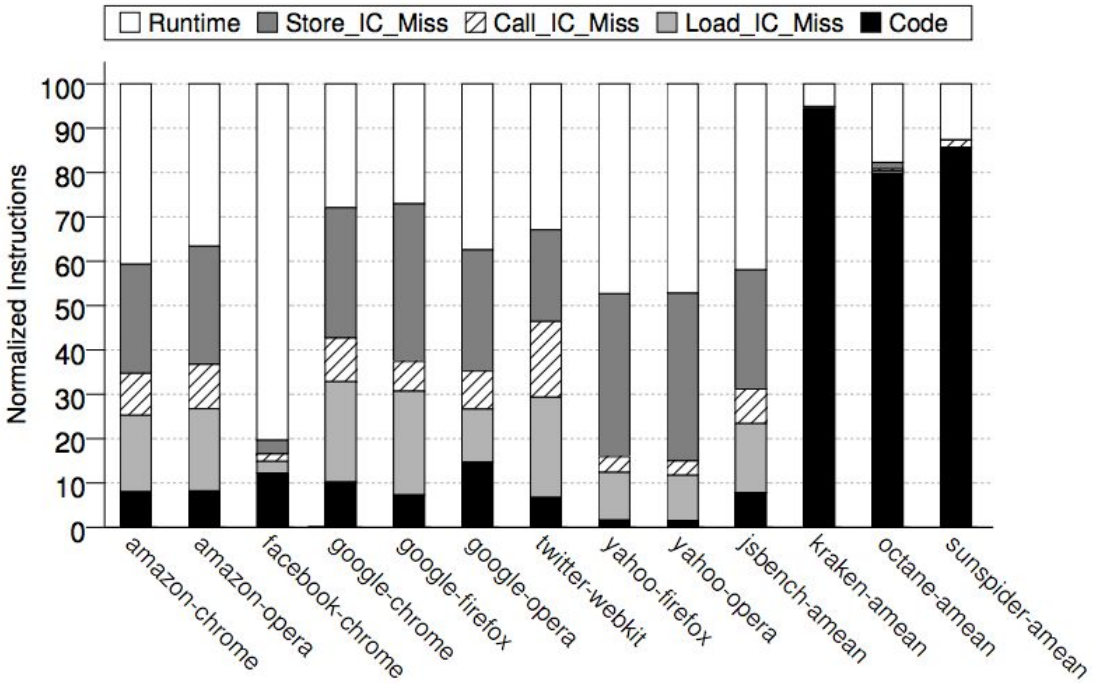
… for users? (improvements positively affect my entire experience)

```
var _sunSpiderStartDate = new Date();

bitwiseAndValue = 4294967296;
for (var i = 0; i < 600000; i++)
     bitwiseAndValue = bitwiseAndValue & i;

var _sunSpiderInterval = new Date() - _sunSpiderStartDate;
record(_sunSpiderInterval);
```

- Global scope, forgetting var
- bitwiseAndValue & X == 0, X & 0 == 0
- Self-timed
- OSR



Improving JavaScript Performance by Deconstructing the Type System, PLDI 2014



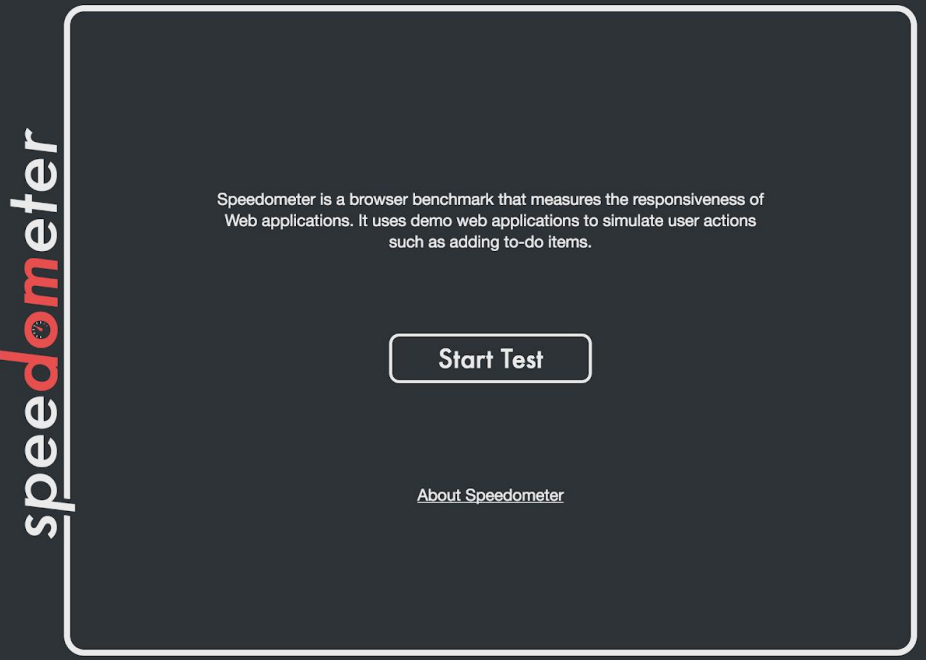
Makes JSBench 36% faster,
Ignores optimizing compiler
(e.g., 8.5x slower on Box2D)

Improving JavaScript Performance by Deconstructing the Type System, PLDI 2014


260
250
240
230
220
210
200

foo.bar is not a function

undefined is not a function


speedometer
Speedometer is a browser benchmark that measures the responsiveness of Web applications. It uses demo web applications to simulate user actions such as adding to-do items.
Start Test
About Speedometer

```
[Summary]:
   ticks    total   nonlib   name
     401    19.6%    20.5%   JavaScript
    1520   74.3%    77.6%   C++
      55     2.7%     2.8%   GC

[JavaScript]:
   ticks  total  nonlib   name
      26    1.3%    1.3%  KeyedLoadIC
      21    1.0%    1.1%  *get ember.js:1917:19
      10    0.5%    0.5%  ~superWrapper ember.js:1227:24
       9    0.4%    0.5%  KeyedLoadIC
       8    0.4%    0.4%  Stub: FastNewSloppyArgumentsStub
       8    0.4%    0.4%  CallFunction_ReceiverIsNotNullOrUndefined
       8    0.4%    0.4%  CallFunction_ReceiverIsAny
       7    0.3%    0.4%  Stub: CEntryStub
       7    0.3%    0.4%  ~propertyDidChange ember.js:2568:27
       6    0.3%    0.3%  ArgumentsAdaptorTrampoline
       5    0.2%    0.3%  Stub: ToBooleanICStub

[C++]:
   ticks  total  nonlib   name
      79    3.9%    4.0%  LookupInRegularHolder
      52    2.5%    2.7%  focusMethodCallback
      40    2.0%    2.0%  __pthread_cond_timedwait
      30    1.5%    1.5%  _IO_vfprintf
      17    0.8%    0.9%  removeChildMethodCallback
```

Speedometer

Google

```
[StoreIC in *ASTSpan+90 at typescript-compiler.js:755 (N->N) 0x1b97251df6a9 <String[7]: minChar>]
[StoreIC in *ASTSpan+143 at typescript-compiler.js:756 (N->N) 0x1b97251df6c9 <String[7]: limChar>]
[StoreIC in *AST+233 at typescript-compiler.js:765 (N->N) 0x1b97251df729 <String[8]: nodeType>]
[StoreIC in *AST+286 at typescript-compiler.js:766 (N->N) 0x1b972518ba39 <String[4]: type>]
[StoreIC in *AST+408 at typescript-compiler.js:767 (N->N) 0x1b97251acc31 <String[5]: flags>]
[StoreIC in *AST+533 at typescript-compiler.js:768 (N->N) 0x1b97251df749 <String[11]: passCreated>]
[StoreIC in *AST+586 at typescript-compiler.js:769 (N->N) 0x1b97251df7c9 <String[11]: preComments>]
[StoreIC in *AST+639 at typescript-compiler.js:770 (N->N) 0x1b97251df7f1 <String[12]: postComments>]
[StoreIC in *AST+692 at typescript-compiler.js:771 (N->N) 0x1b97251df819 <String[11]: docComments>]
[StoreIC in *AST+745 at typescript-compiler.js:772 (N->N) 0x1b97251df841 <String[15]: isParenthesized>]
[StoreIC in *Token+85 at typescript-compiler.js:16139 (N->N) 0x1b97251dfad9 <String[7]: tokenId>]
[StoreIC in *ASTSpan+90 at typescript-compiler.js:755 (N->N) 0x1b97251df6a9 <String[7]: minChar>]
[StoreIC in *ASTSpan+143 at typescript-compiler.js:756 (N->N) 0x1b97251df6c9 <String[7]: limChar>]
[StoreIC in *AST+233 at typescript-compiler.js:765 (N->N) 0x1b97251df729 <String[8]: nodeType>]
[StoreIC in *AST+286 at typescript-compiler.js:766 (N->N) 0x1b972518ba39 <String[4]: type>]
[StoreIC in *AST+408 at typescript-compiler.js:767 (N->N) 0x1b97251acc31 <String[5]: flags>]
[StoreIC in *AST+533 at typescript-compiler.js:768 (N->N) 0x1b97251df749 <String[11]: passCreated>]
[StoreIC in *AST+586 at typescript-compiler.js:769 (N->N) 0x1b97251df7c9 <String[11]: preComments>]
[StoreIC in *AST+639 at typescript-compiler.js:770 (N->N) 0x1b97251df7f1 <String[12]: postComments>]
[StoreIC in *AST+692 at typescript-compiler.js:771 (N->N) 0x1b97251df819 <String[11]: docComments>]
[StoreIC in *AST+745 at typescript-compiler.js:772 (N->N) 0x1b97251df841 <String[15]: isParenthesized>]
[StoreIC in *ASTSpan+90 at typescript-compiler.js:755 (N->N) 0x1b97251df6a9 <String[7]: minChar>]
[StoreIC in *ASTSpan+143 at typescript-compiler.js:756 (N->N) 0x1b97251df6c9 <String[7]: limChar>]
[StoreIC in *AST+233 at typescript-compiler.js:765 (N->N) 0x1b97251df729 <String[8]: nodeType>]
[StoreIC in *AST+286 at typescript-compiler.js:766 (N->N) 0x1b972518ba39 <String[4]: type>]
[StoreIC in *AST+408 at typescript-compiler.js:767 (N->N) 0x1b97251acc31 <String[5]: flags>]
[StoreIC in *AST+533 at typescript-compiler.js:768 (N->N) 0x1b97251df749 <String[11]: passCreated>]
[StoreIC in *AST+586 at typescript-compiler.js:769 (N->N) 0x1b97251df7c9 <String[11]: preComments>]
```

details	73.34%	108537	(N->N)
details	12.8%	18935	(0->.)
details	9.22%	13638	(.->1)
details	0.93%	1382	(1->P)
details	0.81%	1204	(P->P)
details	0.44%	656	(^->1)
details	0.4%	587	(0->1)

Result

Group-Key: filePosition

details	4.29%	22	prologue.js:21 ~ImportNow+49

type [top 20]

details	100%	22	KeyedLoadIC

category [top 20]

details	100%	22	Load

file [top 20]

details	100%	22	prologue.js:21

state [top 20]

details	81.82%	18	(N->N)
details	9.09%	2	(.->1)
details	9.09%	2	(1->N)

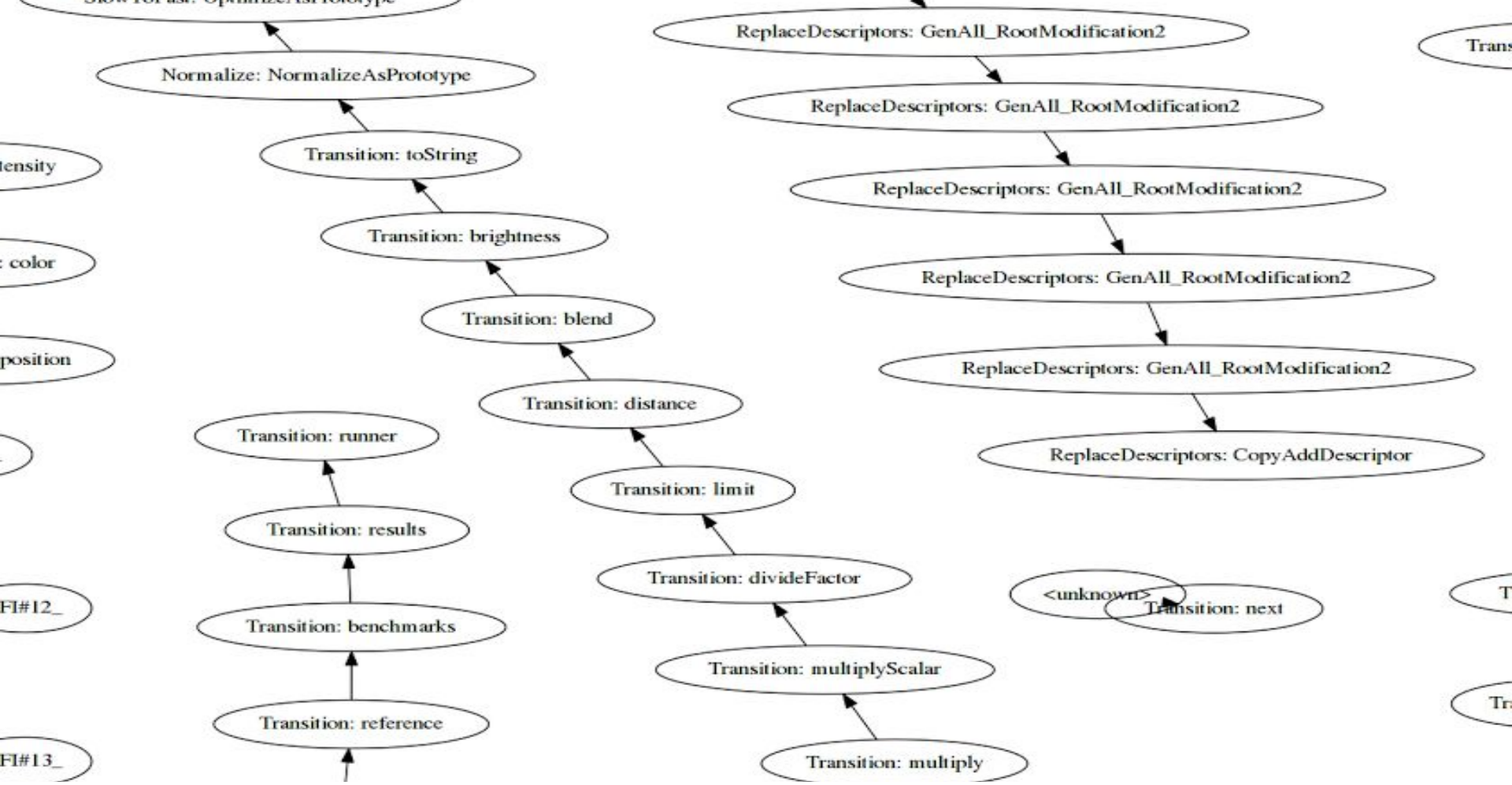
key [top 20]

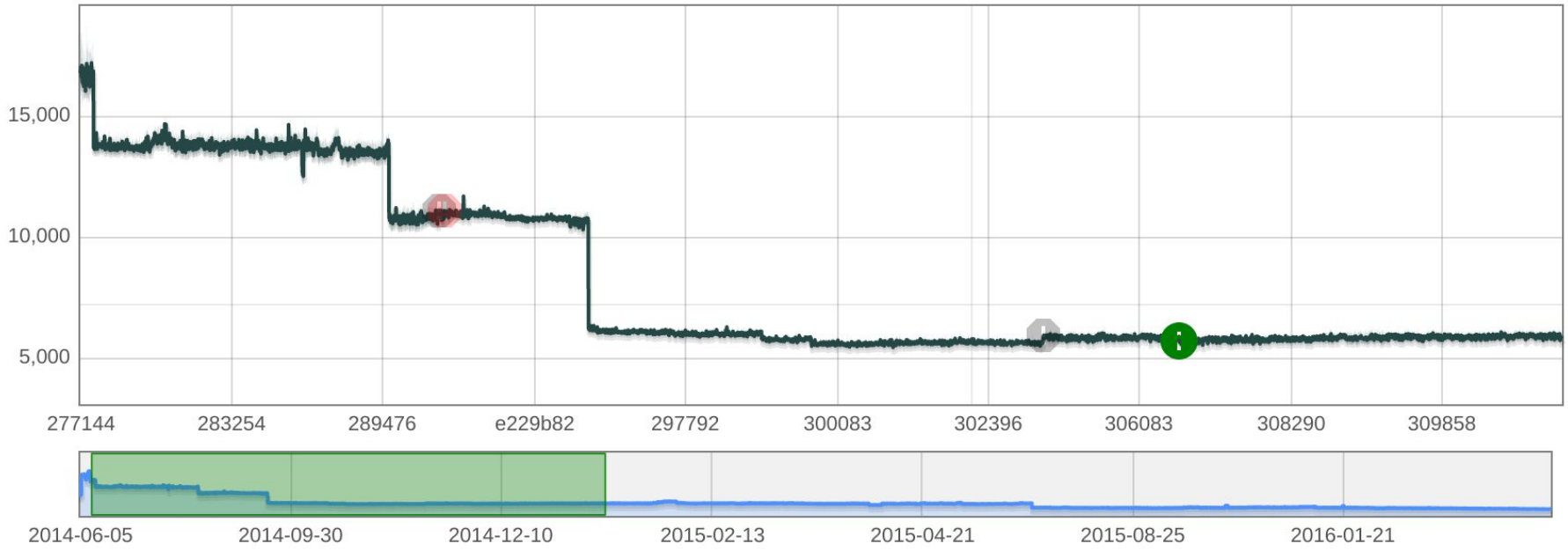
details	100%	22	map=0x1dcb4b0092c9

isNative [top 20]

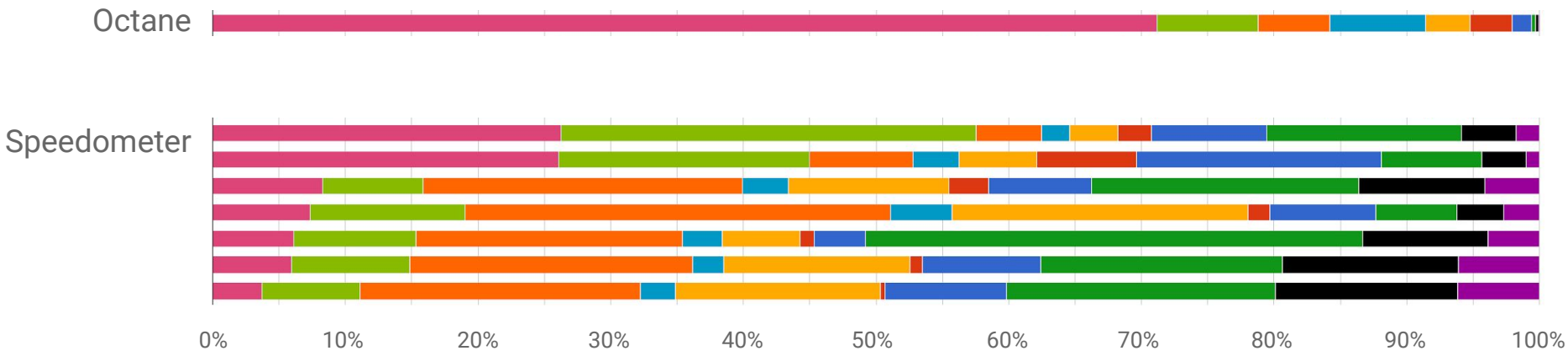
details	100%	22	true

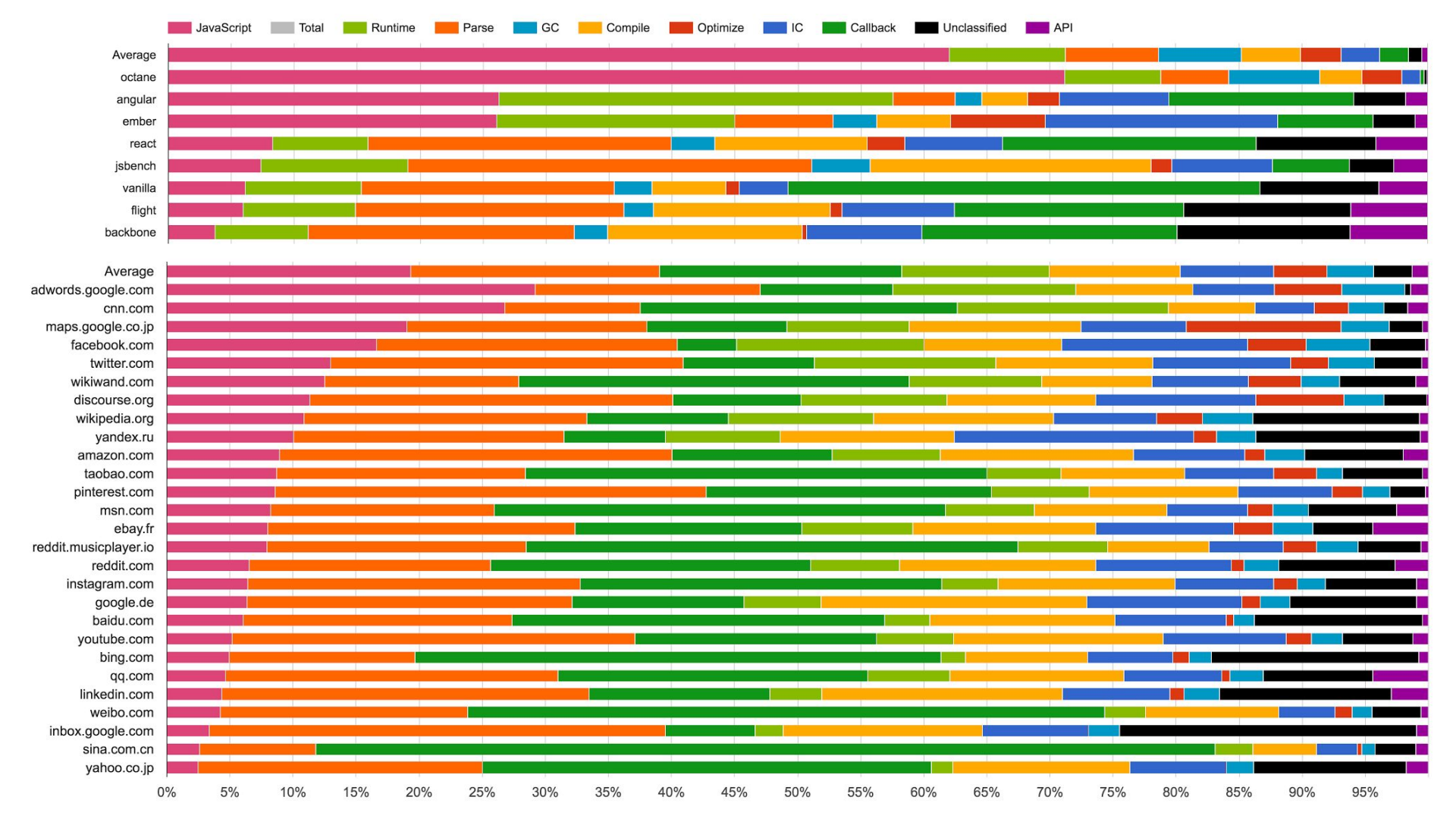
details	1.17%	6	forof.js:91 ~ForOf+287
details	0.78%	4	array-iterator.js:51 ~next+614
details	0.78%	4	string-iterator.js:31 ~next+134
details	0.78%	4	string-iterator.js:37 ~next+336
details	0.78%	4	forof.js:95 *ForOf+85
details	0.78%	4	array-iterator.js:45 ~next+407
details	0.78%	4	string-iterator.js:35 ~next+272
details	0.58%	3	forof.js:92 ~ForOf+603
details	0.58%	3	prologue.js:195 ~PostExperimentals+367

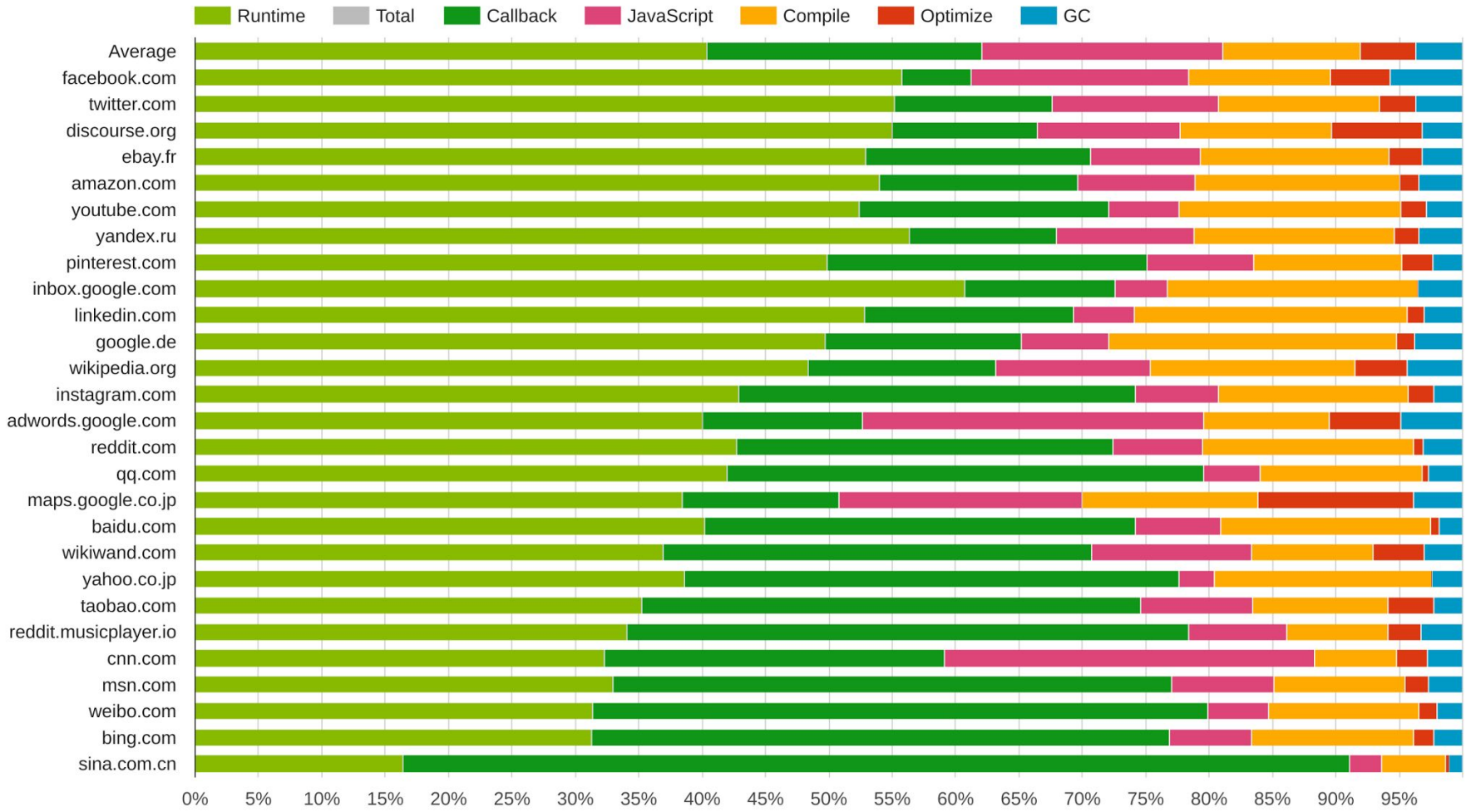

ReplaceDescriptors: GenAll_RootModification2
ReplaceDescriptors: GenAll_RootModification2
ReplaceDescriptors: GenAll_RootModification2
ReplaceDescriptors: GenAll_RootModification2
ReplaceDescriptors: GenAll_RootModification2
ReplaceDescriptors: CopyAddDescriptor
Normalize: NormalizeAsPrototype
Transition: toString
Transition: brightness
Transition: blend
Transition: distance
Transition: limit
Transition: divideFactor
Transition: multiplyScalar
Transition: multiply
Transition: runner
Transition: results
Transition: benchmarks
Transition: reference
<unknown>
Transition: next


15,000
10,000
5,000
277144
283254
289476
e229b82
297792
300083
302396
306083
308290
309858
2014-06-05
2014-09-30
2014-12-10
2015-02-13
2015-04-21
2015-08-25
2016-01-21

```
[Summary]:
   ticks    total     nonlib   name
     401    19.6%      20.5%   JavaScript
    1520    74.3%      77.6%   C++
      55     2.7%       2.8%   GC

[C++ entry points]:
   ticks    cpp   total   name
     207   15.1%   10.1%  v8::internal::Runtime_KeyedLoadIC_Miss
     192   14.0%    9.4%  v8::internal::Runtime_KeyedStoreIC_Miss
     145   10.6%    7.1%  v8::internal::Runtime_CompileOptimized_Concurrent
     125    9.1%    6.1%  v8::internal::Runtime_CompileLazy
     111    8.1%    5.4%  v8::internal::Runtime_LoadIC_Miss
      93    6.8%    4.5%  v8::internal::Builtin_HandleApiCall
      74    5.4%    3.6%  v8::internal::Runtime_StoreIC_Miss
      54    3.9%    2.6%  v8::internal::Runtime_KeyedGetProperty
      49    3.6%    2.4%  v8::internal::Runtime_StackGuard
      41    3.0%    2.0%  v8::internal::Runtime_SetProperty
      18    1.3%    0.9%  v8::internal::Runtime_StringSplit
      18    1.3%    0.9%  v8::internal::Runtime_AllocateInTargetSpace
      17    1.2%    0.8%  v8::internal::Runtime_GetProperty
      16    1.2%    0.8%  v8::internal::Builtin_ObjectDefineProperty
      15    1.1%    0.7%  v8::internal::Runtime_TryInstallOptimizedCode
      14    1.0%    0.7%  v8::internal::Builtin_ObjectProtoToString
      12    0.9%    0.6%  v8::internal::Builtin_ObjectCreate
```


Octane
Speedometer
0%
10%
20%
30%
40%
50%
60%
70%
80%
90%
100%
JavaScript
Callback
Parse
Runtime
Compile
IC
GC
Optimize
Unknown
API


JavaScript
Total
Runtime
Parse
GC
Compile
Optimize
IC
Callback
Unclassified
API
Average
octane
angular
ember
react
jsbench
vanilla
flight
backbone
Average
adwords.google.com
cnn.com
maps.google.co.jp
facebook.com
twitter.com
wikiwand.com
discourse.org
wikipedia.org
yandex.ru
amazon.com
taobao.com
pinterest.com
msn.com
ebay.fr
reddit.musicplayer.io
reddit.com
instagram.com
google.de
baidu.com
youtube.com
bing.com
qq.com
linkedin.com
weibo.com
inbox.google.com
sina.com.cn
yahoo.co.jp
0%
5%
10%
15%
20%
25%
30%
35%
40%
45%
50%
55%
60%
65%
70%
75%
80%
85%
90%
95%


Runtime
Total
Callback
JavaScript
Compile
Optimize
GC
Average
facebook.com
twitter.com
discourse.org
ebay.fr
amazon.com
youtube.com
yandex.ru
pinterest.com
inbox.google.com
linkedin.com
google.de
wikipedia.org
instagram.com
adwords.google.com
reddit.com
qq.com
maps.google.co.jp
baidu.com
wikiwand.com
yahoo.co.jp
taobao.com
reddit.musicplayer.io
cnn.com
msn.com
weibo.com
bing.com
sina.com.cn
0%
5%
10%
15%
20%
25%
30%
35%
40%
45%
50%
55%
60%
65%
70%
75%
80%
85%
90%
95%

Google

Name	Time	Percent	Count
Group-Runtime	**292.3ms**	**20.60%**	**762490#**
GetPropertyNamesFast	66.8ms	4.70%	5166#
Map_TransitionToDataProperty	41.4ms	2.90%	49629#
ForInFilter	28.4ms	2.00%	412154#
CreateObjectLiteral	18.6ms	1.30%	20118#
KeyedGetProperty	17.5ms	1.20%	74721#
ObjectAssign	17.5ms	1.20%	4145#
FunctionPrototypeBind	11.5ms	0.80%	7753#
ArrayPush	7.0ms	0.50%	18237#
GetProperty	5.7ms	0.40%	13102#
ArrayConcat	5.4ms	0.40%	3713#
SetProperty	5.1ms	0.40%	5495#
ObjectKeys	4.8ms	0.30%	470#
NewClosure	4.6ms	0.30%	8353#
CreateArrayLiteralStubBailout	4.3ms	0.30%	3125#
PrototypeMap_TransitionToDataProperty	3.7ms	0.30%	3140#
NewObject	3.2ms	0.20%	1926#
StringReplaceGlobalRegExpWithString	2.8ms	0.20%	5413#
NewStrictArguments	2.7ms	0.20%	2948#
RegExpExec	2.7ms	0.20%	662#
HasOwnProperty	2.5ms	0.20%	18450#

`Object.assign`

`Object.keys`

`Object.prototype.hasOwnProperty`

`Object.prototype.toString`

`in` operator

`Array.prototype.toString / .join`

`Array.prototype.push`

`Function.prototype.bind`

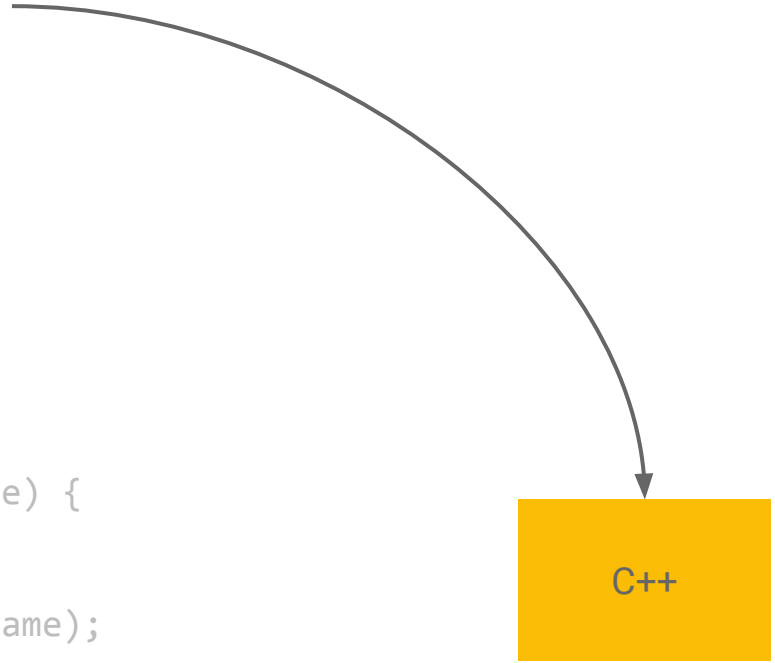
First runtime, then stubs

```
foo.hasOwnProperty("bar")
```

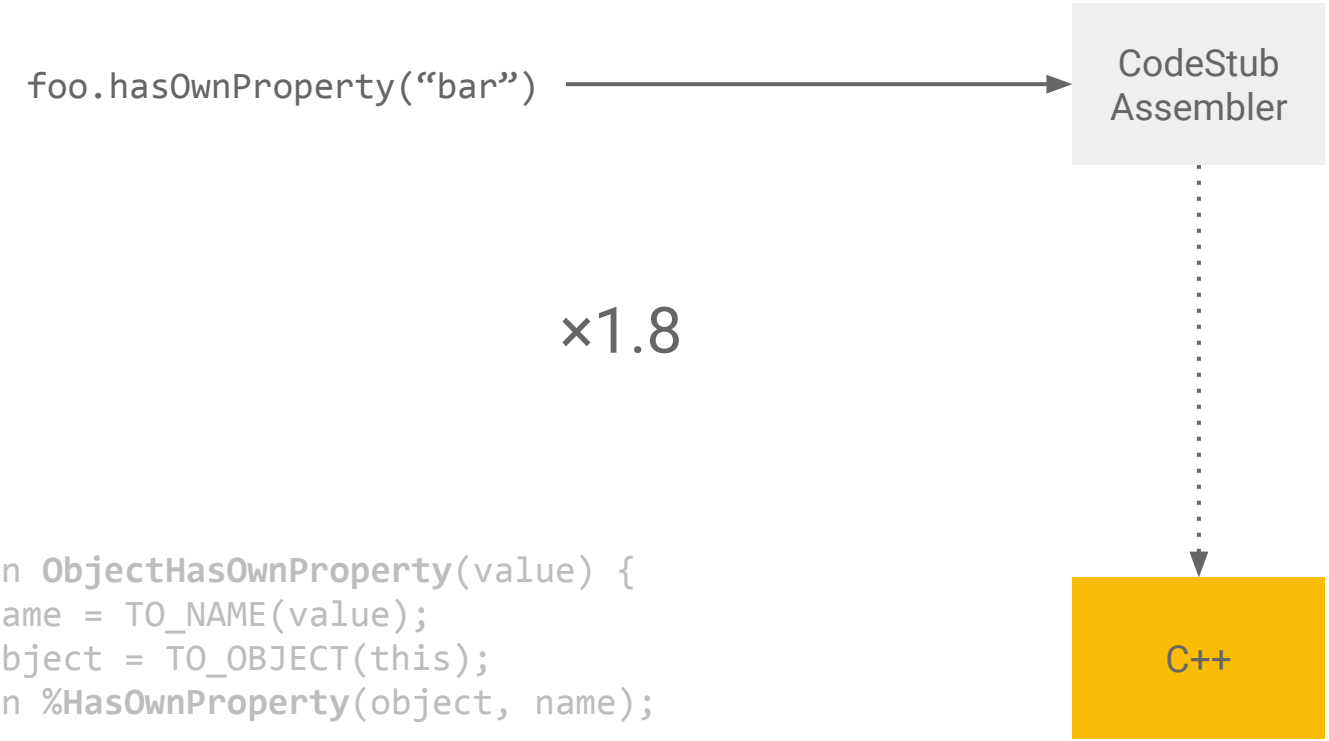
```
function ObjectHasOwnProperty(value) {
  var name = TO_NAME(value);
  var object = TO_OBJECT(this);
  return %HasOwnProperty(object, name);
}
```


C++

```
foo.hasOwnProperty(“bar”)
```

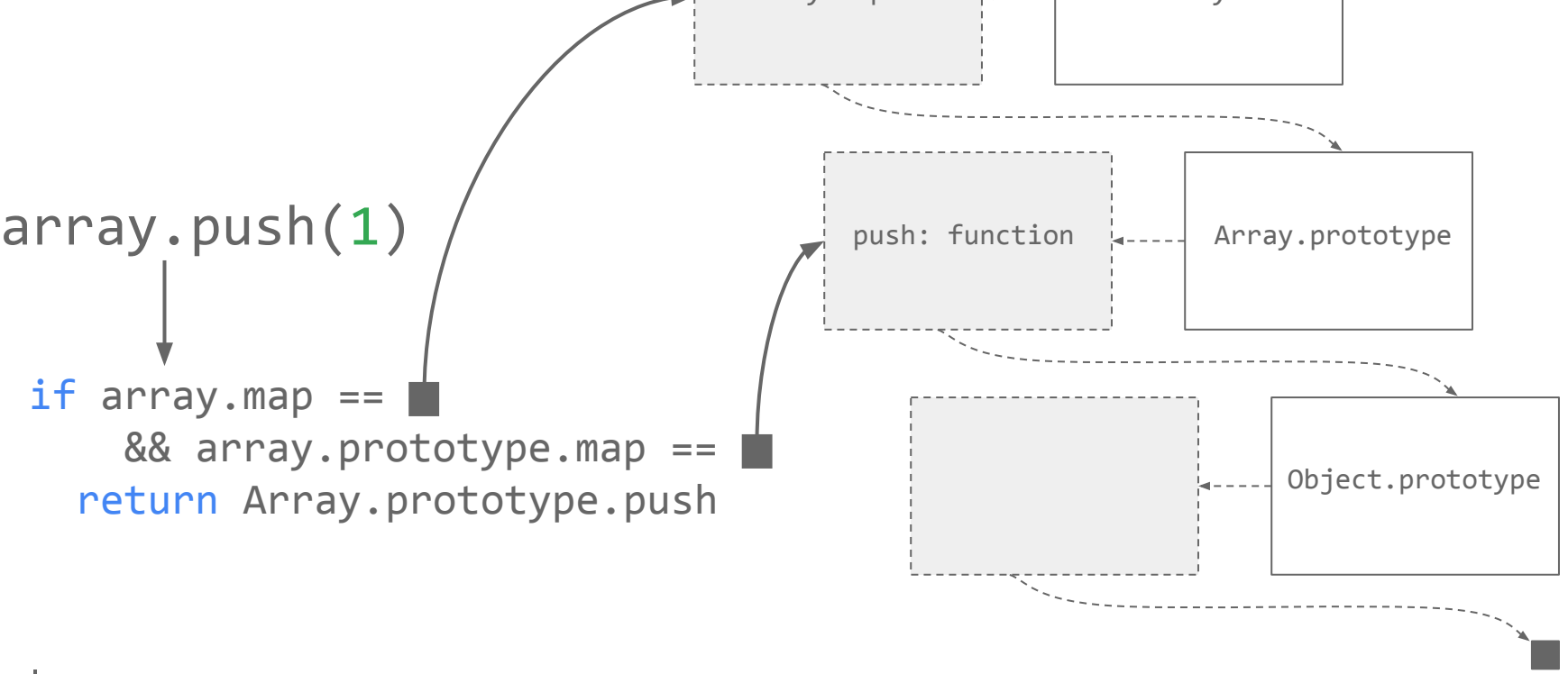



```
function ObjectHasOwnProperty(value) {
  var name = TO_NAME(value);
  var object = TO_OBJECT(this);
  return %HasOwnProperty(object, name);
}
```




```
function ObjectHasOwnProperty(value) {
  var name = TO_NAME(value);
  var object = TO_OBJECT(this);
  return %HasOwnProperty(object, name);
}
```

```
array.push(1)
```


if array.map ==
&& array.prototype.map ==
return Array.prototype.push
Array map
array
push: function
Array.prototype
Object.prototype

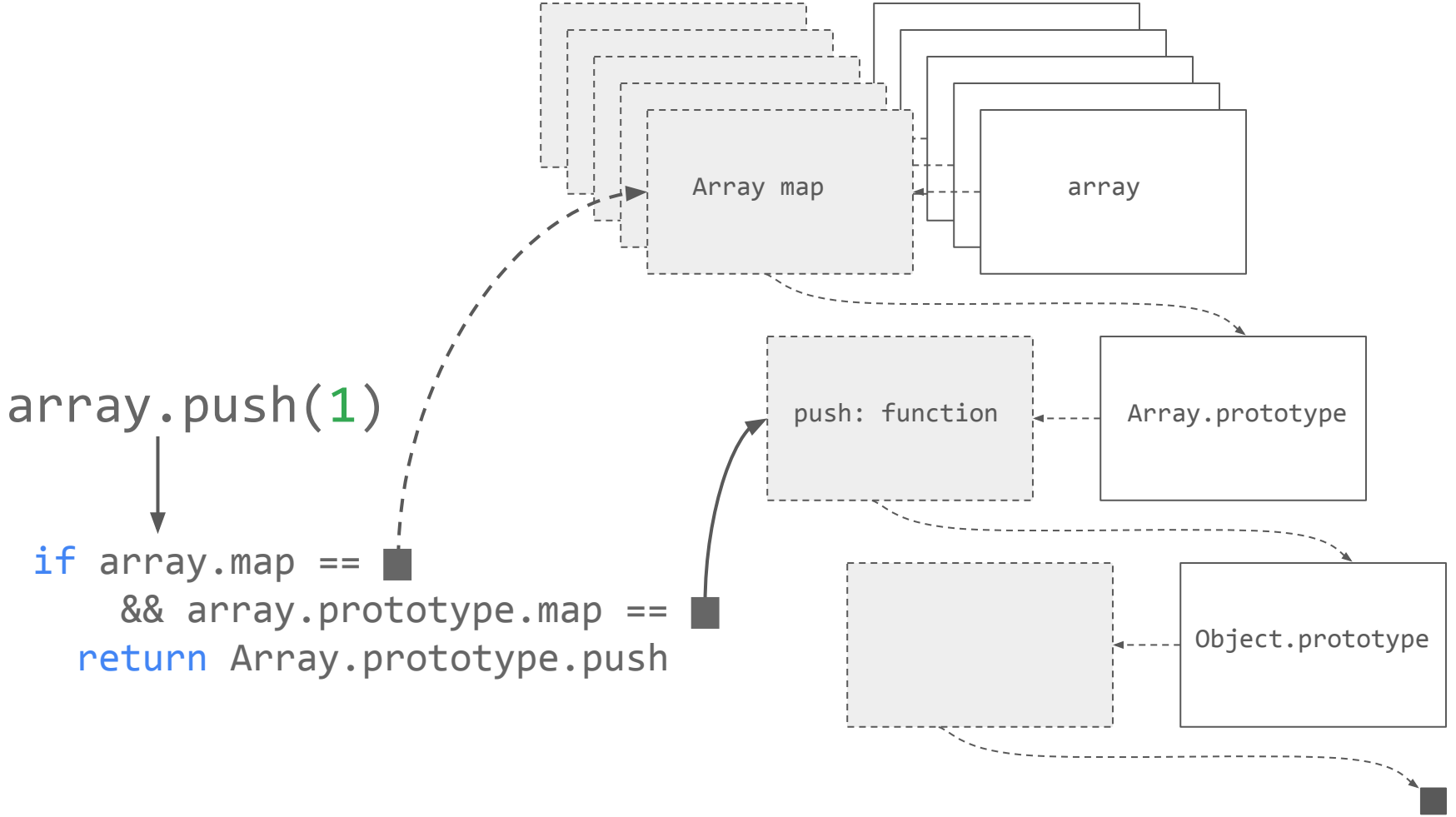
```
array.push(1)
```

```
function push(array, v) {
  array[array.length] = v
}
```

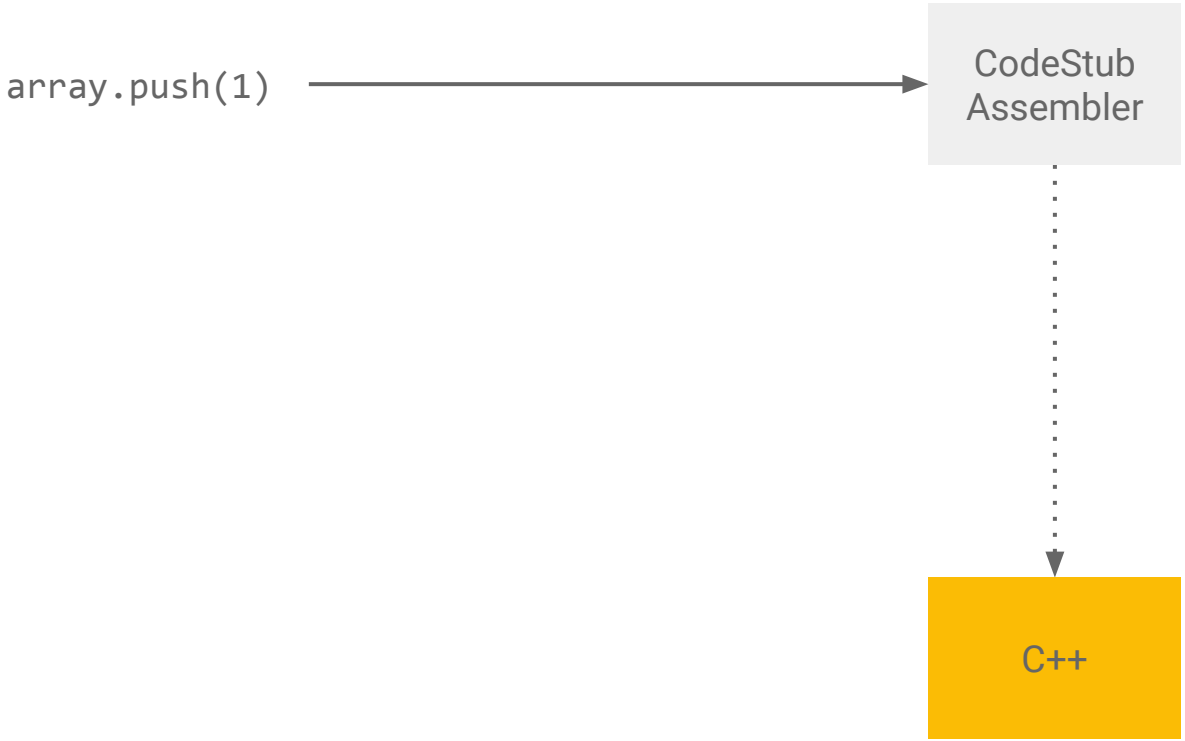
```
array.push(1)
```

```
function push(array, v) {
  array[array.length] = v
}
```

```
a1 = [  1    ]
a2 = [  1.1  ]
a3 = [ “foo” ]
a4 = [ 1,,   ]
a5 = [ 1.1,, ]
a6 = [“foo”,,]
a7 = []; a7[1024*1024] = 1;
```


Array map
array
push: function
Array.prototype
Object.prototype
array.push(1)
if array.map ==
&& array.prototype.map ==
return Array.prototype.push

```
Array.prototype.push = function() {...}
Array.prototype[1] = “a good idea”
Object.prototype.length = “wut?”
Array.prototype = new JSProxy(...)
```


array.push(1)
CodeStub
Assembler
C++

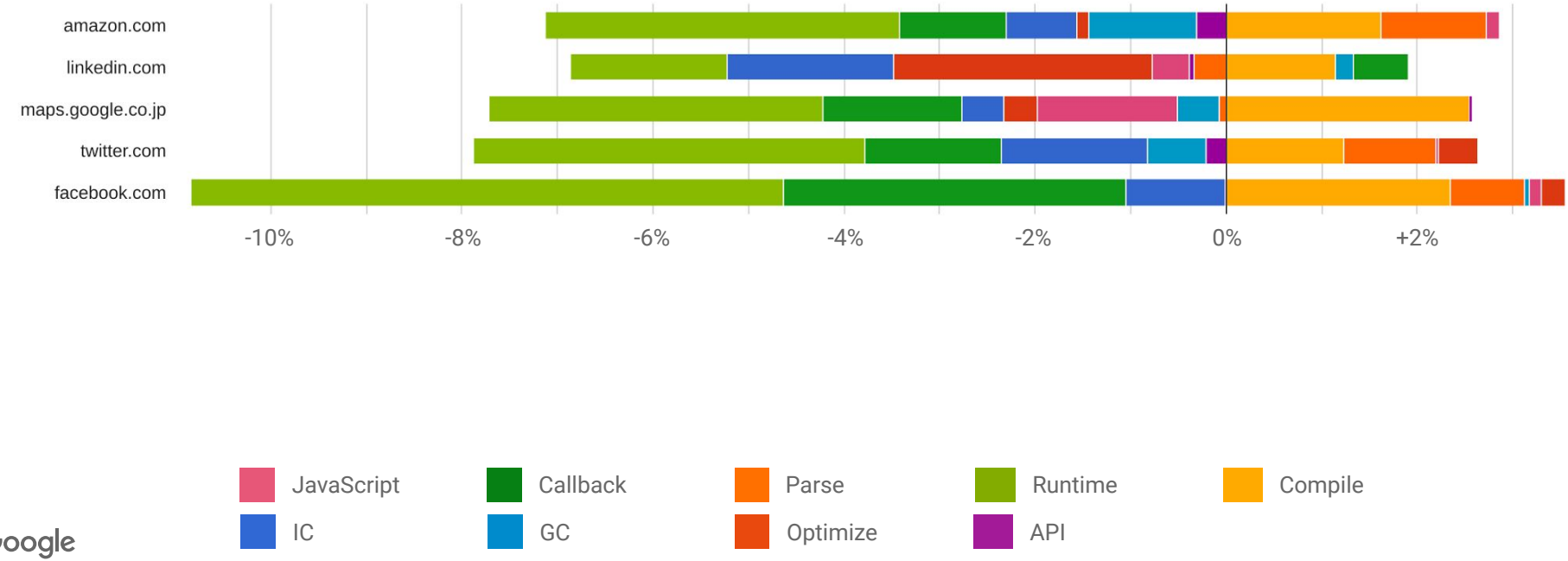
runtime prechecks

this[this.length] = value

```
template <>
HValue* CodeStubGraphBuilder<FastArrayPushStub>::BuildCodeStub() {
  // TODO(verwaest): Fix deoptimizer messages.
  HValue* argc = GetArgumentsLength();
  HInstruction* argument_elements = Add<HArgumentsElements>(false, false);
  HInstruction* object = Add<HAccessArgumentsAt>(argument_elements, argc,
                                                 graph()->GetConstantMinus1());
  BuildCheckHeapObject(object);
  HValue* map = Add<HLoadNamedField>(object, nullptr, HObjectAccess::ForMap());
  Add<HCheckInstanceType>(object, HCheckInstanceType::IS_JS_ARRAY);

  // Disallow pushing onto prototypes. It might be the JSArray prototype.
  // Disallow pushing onto non-extensible objects.
  {
    HValue* bit_field2 =
        Add<HLoadNamedField>(map, nullptr, HObjectAccess::ForMapBitField2());
    HValue* mask =
        Add<HConstant>(static_cast<int>(Map::IsPrototypeMapBits::kMask) |
                       (1 << Map::kIsExtensible));
    HValue* bits = AddUncasted<HBitwise>(Token::BIT_AND, bit_field2, mask);
    IfBuilder check(this);
    check.IfNot<HCompareNumericAndBranch>(
        bits, Add<HConstant>(1 << Map::kIsExtensible), Token::NE);
    check.ThenDeopt(DeoptimizerIsFastPathFailed);
    check.End();
  }

  // Disallow pushing onto arrays in dictionary named property mode. We need to
  // figure out whether the length property is still writable.
  {
    HValue* bit_field3 =
        Add<HLoadNamedField>(map, nullptr, HObjectAccess::ForMapBitField3());
    HValue* mask = Add<HConstant>(static_cast<int>(Map::DictionaryMap::kMask));
    HValue* bit = AddUncasted<HBitwise>(Token::BIT_AND, bit_field3, mask);
    IfBuilder check(this);
    check.IfNot<HCompareNumericAndBranch>(bit, mask, Token::EQ);
    check.ThenDeopt(DeoptimizerIsFastPathFailed);
    check.End();
  }

  // Check whether the length property is writable. The length property is the
  // only default named property on arrays. It's nonconfigurable, hence is
  // guaranteed to stay the first property.
  {
    HValue* descriptors =
        Add<HLoadNamedField>(map, nullptr, HObjectAccess::ForMapDescriptors());
    HValue* details = Add<HLoadKeyed>(
        descriptors, Add<HConstant>(DescriptorArray::ToDetailsIndex(0)),
        nullptr, nullptr, FAST_SMI_ELEMENTS);
    HValue* mask =
        Add<HConstant>(READ_ONLY << PropertyDetails::AttributesField::kShift);
    HValue* bit = AddUncasted<HBitwise>(Token::BIT_AND, details, mask);
    IfBuilder check(this);
    check.If<HCompareNumericAndBranch>(bit, mask, Token::EQ);
    check.ThenDeopt(DeoptimizerIsFastPathFailed);
    check.End();
  }

  HValue* null = Add<HLoadRoot>(Heap::kNullValueRootIndex);
  HValue* empty = Add<HLoadRoot>(Heap::kEmptyFixedArrayRootIndex);
  environment()->Push(map);
  LoopBuilder check_prototypes(this);
  check_prototypes.BeginBody(1);
  {
    HValue* parent_map = environment()->Pop();
    HValue* prototype = Add<HLoadNamedField>(parent_map, nullptr,
                                             HObjectAccess::ForPrototype());

    IfBuilder is_null(this);
    is_null.If<HCompareObjectEqAndBranch>(prototype, null);
    is_null.Then();
    check_prototypes.Break();
    is_null.End();

    HValue* prototype_map =
        Add<HLoadNamedField>(prototype, nullptr, HObjectAccess::ForMap());
    HValue* instance_type = Add<HLoadNamedField>(
        prototype_map, nullptr, HObjectAccess::ForMapInstanceType());
    IfBuilder check_instance_type(this);
    check_instance_type.IfNot<HCompareNumericAndBranch>(
        instance_type, Add<HConstant>(LAST_CUSTOM_ELEMENTS_RECEIVER),
        Token::LTE);
    check_instance_type.ThenDeopt(DeoptimizerIsFastPathFailed);
    check_instance_type.End();

    HValue* elements = Add<HLoadNamedField>(
        prototype, nullptr, HObjectAccess::ForElementsPointer());
    IfBuilder no_elements(this);
    no_elements.IfNot<HCompareObjectEqAndBranch>(elements, empty);
    no_elements.ThenDeopt(DeoptimizerIsFastPathFailed);
    no_elements.End();

    environment()->Push(prototype_map);
  }
  check_prototypes.EndBody();

  HValue* bit_field2 =
      Add<HLoadNamedField>(map, nullptr, HObjectAccess::ForMapBitField2());
  HValue* kind = BuildDecodeField<Map::ElementsKindBits>(bit_field2);

  // Below we only check the upper bound of the relevant ranges to include both
  // holey and non-holey versions. We check them in order smi, object, double
  // since smi < object < double.
  STATIC_ASSERT(FAST_SMI_ELEMENTS < FAST_HOLEY_SMI_ELEMENTS);
  STATIC_ASSERT(FAST_HOLEY_SMI_ELEMENTS < FAST_HOLEY_ELEMENTS);
  STATIC_ASSERT(FAST_ELEMENTS < FAST_HOLEY_ELEMENTS);
  STATIC_ASSERT(FAST_HOLEY_ELEMENTS < FAST_HOLEY_DOUBLE_ELEMENTS);
  STATIC_ASSERT(FAST_DOUBLE_ELEMENTS < FAST_HOLEY_DOUBLE_ELEMENTS);
  IfBuilder has_smi_elements(this);
  has_smi_elements.If<HCompareNumericAndBranch>(
      kind, Add<HConstant>(FAST_HOLEY_SMI_ELEMENTS), Token::LTE);
  has_smi_elements.Then();
  {
    HValue* new_length = BuildPushElement(object, argc, argument_elements,
                                          FAST_HOLEY_SMI_ELEMENTS);

    environment()->Push(new_length);
  }
  has_smi_elements.Else();
  {
    IfBuilder has_object_elements(this);
    has_object_elements.If<HCompareNumericAndBranch>(
        kind, Add<HConstant>(FAST_HOLEY_ELEMENTS), Token::LTE);
    has_object_elements.Then();
    {
      HValue* new_length = BuildPushElement(object, argc, argument_elements,
                                            FAST_HOLEY_ELEMENTS);

      environment()->Push(new_length);
    }
    has_object_elements.Else();
    {
      IfBuilder has_double_elements(this);
      has_double_elements.If<HCompareNumericAndBranch>(
          kind, Add<HConstant>(FAST_HOLEY_DOUBLE_ELEMENTS), Token::LTE);
      has_double_elements.Then();
      {
        HValue* new_length = BuildPushElement(object, argc, argument_elements,
                                              FAST_HOLEY_DOUBLE_ELEMENTS);

        environment()->Push(new_length);
      }
      has_double_elements.ElseDeopt(DeoptimizerIsFastPathFailed);
      has_double_elements.End();
    }
    has_object_elements.End();
  }
  has_smi_elements.End();

  return environment()->Pop();
}
```

×2

M49 vs. M53


amazon.com
linkedin.com
maps.google.co.jp
twitter.com
facebook.com
-10%
-8%
-6%
-4%
-2%
0%
+2%
JavaScript
Callback
Parse
Runtime
Compile
IC
GC
Optimize
API

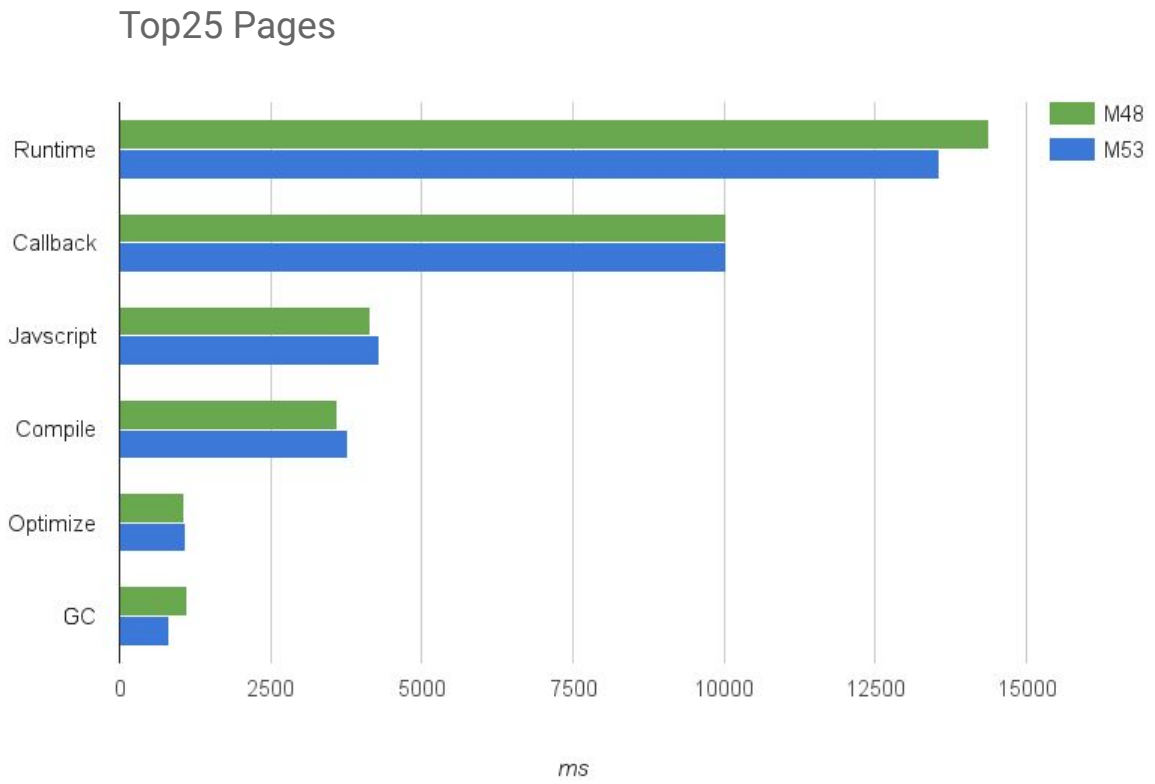
What makes a good benchmark? (stable, reproducible, single number)

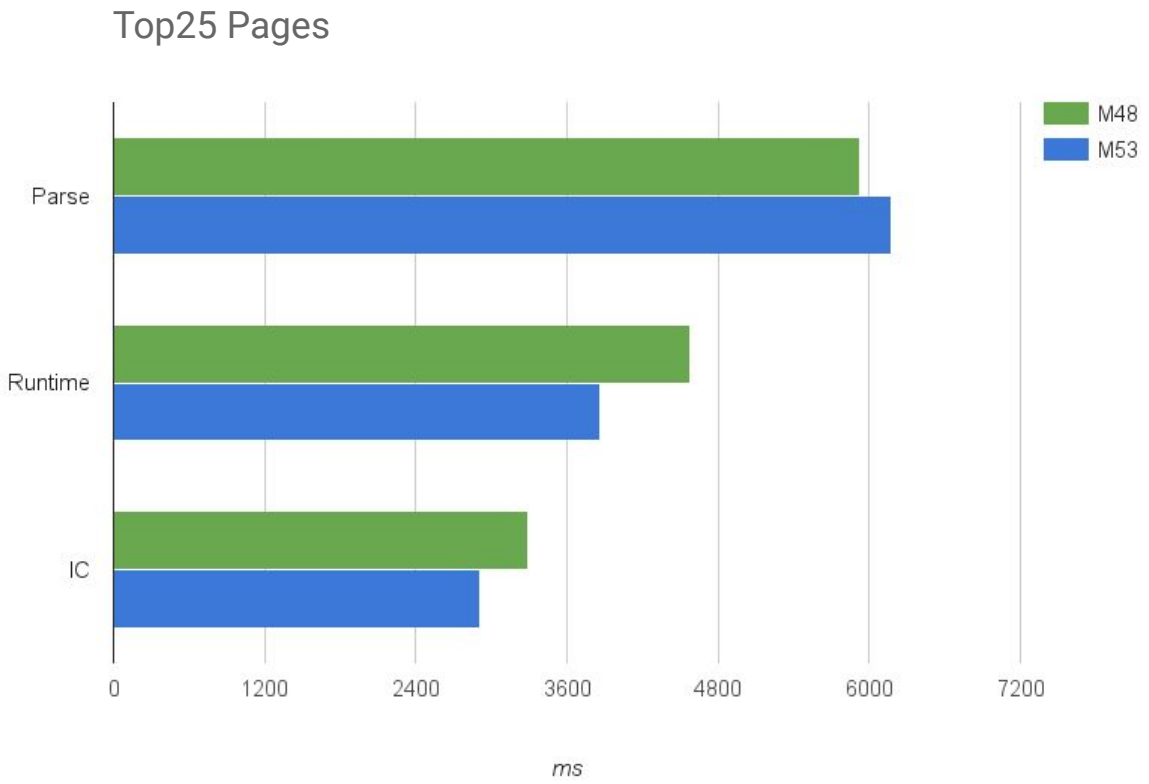
… for VM developers? (priorities, meaningful insights)

… for “benchmarketing”? (cross-browser, not gameable)

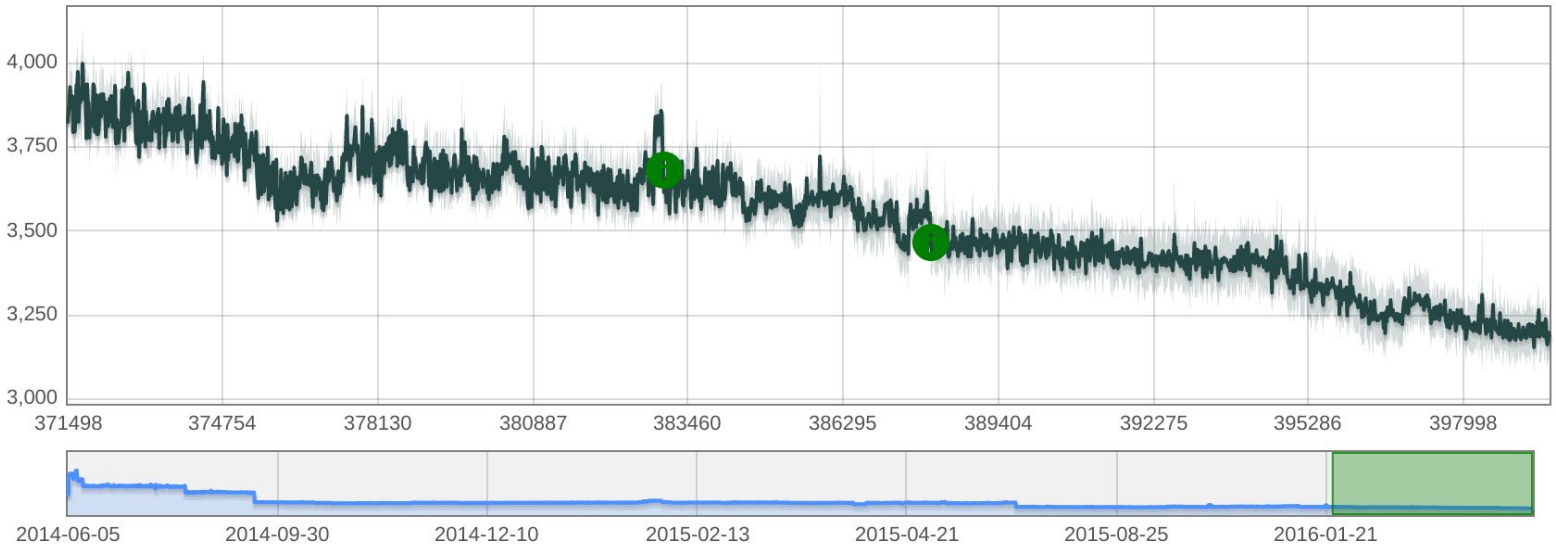
… for application developers? (represents patterns I want to use, cross-browser)

… for users? (improvements positively affect my entire experience)


Top25 Pages
Runtime
Callback
Javscript
Compile
Optimize
GC
0
2500
5000
7500
10000
12500
15000
ms
M48
M53


Top25 Pages
M48
M53
Parse
Runtime
IC
0
1200
2400
3600
4800
6000
7200
ms

20%


4,000
3,750
3,500
3,250
3,000
371498
374754
378130
380887
383460
386295
389404
392275
395286
397998
2014-06-05
2014-09-30
2014-12-10
2015-02-13
2015-04-21
2015-08-25
2016-01-21