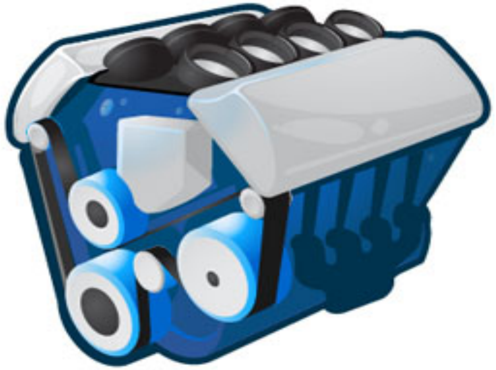
The V8 JavaScript Engine

Design, Implementation, Testing and Benchmarking

Mads Ager, Software Engineer



Agenda

- Part 1: What is JavaScript?
- Part 2: V8 internals
- Part 3: V8 testing and benchmarking

What is JavaScript?

- The main programming language of the web
- Dynamically typed scripting language
- C-like syntax

JavaScript types

- Basic types
    - Undefined (undefined)
    - Null (null)
    - Boolean (true, false)
    - String ("a", 'a', ...)
    - Number (1, 1.23, ...)
- Object
    - Everything else

JavaScript objects

- An object is a collection of properties
- A property has a name and a value
- Properties can be added and removed on the fly

```
var o = {};  // Empty object.o.x = 42;    // Adding property.o.x;          // Accessing property (result:
42).o.y;          // Accessing non-existing property              // (result: undefined).delete o.x;  //
Removing property.
o.x;          // Accessing non-existing property.
```

JavaScript objects

- No classes in JavaScript
- All JavaScript objects are constructed from functions

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}

var p = new Point(1, 2);
var q = new Point(2, 3);

var o = {};  // Basically shorthand for "new Object()"
var a = [];  // Basically shorthand for "new Array()"
```

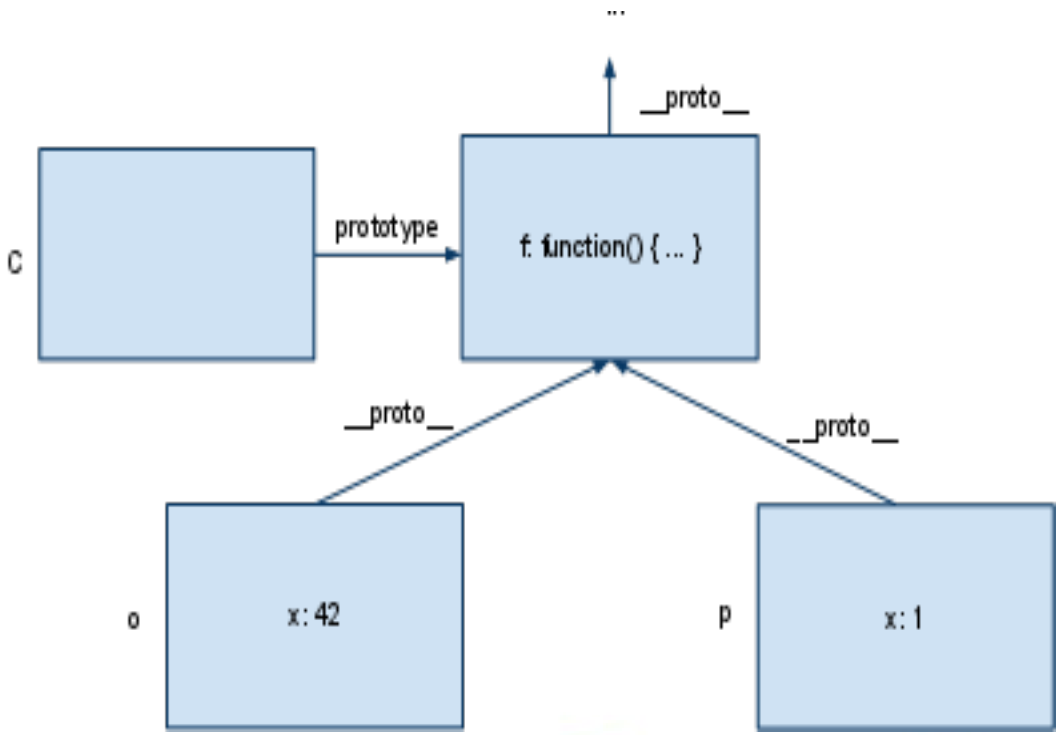
JavaScript objects

- All objects have a prototype property which can be either another object or null
- When accessing a property on an object the entire prototype chain is searched

```
function C(x) { this.x = x; }
C.prototype.f = function() {
  return this.x;
}

var o = new C(42);
o.f();  // Result: 42

var p = new C(1);
p.f();  // Result: 1
```



JavaScript objects

- Properties that should be available on all objects can be added to Object.prototype

- Properties that should be available on all function objects can be added to Function.prototype

JavaScript objects

- Prototype chains can be used to model inheritance

```
Function.prototype.inherit = function(super) {
  var tmpCtr = function() {};
  tmpCtr.prototype = super.prototype;
  this.prototype = new tmpCtr();
  this.super_ = super;
}
function X(x) { this.x = x; }
X.prototype.get_x = function() { return this.x; };
function Y(x, y) {
  Y.super_.call(this, x);
  this.y = y;
}
Y.inherit(X);
Y.prototype.get_y = function() { return this.y; };
var y = new Y(1, 2);
y.get_x(); y.get_y();
```

JavaScript types (continued)

- No types specified in the source code

- Duck typing: "If it looks like a duck and quacks like a duck, it is a duck to me"
    - All that matters is the properties, not the type

JavaScript types (continued)

- Type conversion happen at runtime to make operations make sense
- Type conversion can even be controlled by the programmer

```
var x = 1 + 2;
x = 1 + "2";
x = 1 + {};

var o = {};
o.toString = function() { return "point"; }
x = 1 + o;
```

JavaScript in browsers

- Document Object Model (DOM) objects exposed as javascript objects
- Event model: onload, onclick, ...

```
<html><body>
<script>
function changeColor(color) {
  document.getElementById('p').style.color = color;
}
</script>
<p id="p">
Changing the color of some text.
</p>
<button type="button" onclick="changeColor('blue')">blue</button>
<button type="button" onclick="changeColor('red')">red</button>
</body></html>
```

Agenda

- Part 1: What is JavaScript?
- Part 2: V8 internals
- Part 3: V8 testing and benchmarking

What is V8?

- JavaScript engine built from scratch in Aarhus, Denmark
- Goal was to raise the performance bar for JavaScript
- Fully open source to allow other browser vendors to use the ideas from V8

Design goals

- Make real object-oriented programs run fast
- Fast property access
- Fast function calls
- Fast and scalable memory management

The challenge

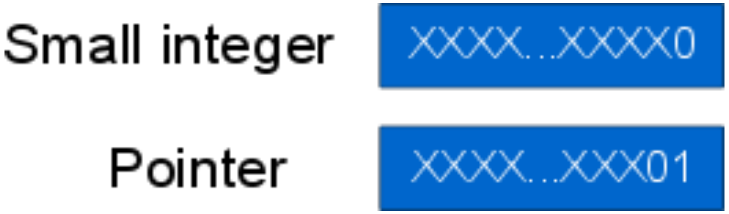
- JavaScript is very dynamic
- Properties are added and removed on the fly
- Prototype chains can be altered on the fly

Design decisions

- Introduce a notion of type called 'hidden classes' or 'maps' behind the scenes
- Generate native code
- Use a technique called inline caching for property access and function calls
- Use precise generational garbage collection

V8 memory model

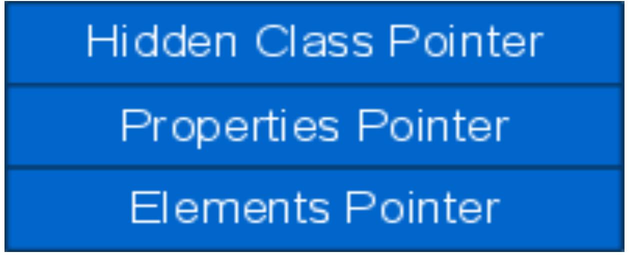
- 32-bit tagged pointers
- Objects are 4-byte aligned, so two bits available for tagging
- Small 31-bit signed integers are immediate values distinguished from pointers by tags



Small integer XXXX...XXXX0

Pointer XXXX...XXX01

- Base JavaScript objects consists of three words



Hidden Class Pointer

Properties Pointer

Elements Pointer

Hidden classes

- Wanted to take advantage of optimization techniques from statically typed object oriented languages
- Introduced the concept of hidden classes to get there
- Hidden classes group objects that have the same structure

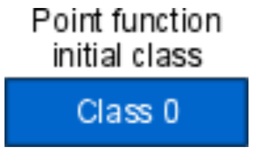
Hidden Classes by Example

- JavaScript objects constructed in the same way should get the same hidden class

```
function Point(x, y) {
 this.x = x;
 this.y = y;
}
var p1 = new Point(0,1);
var p2 = new Point(2,3);
```

Hidden Classes by Example

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}
var p1 = new Point(0,1);
var p2 = new Point(2,3);
```

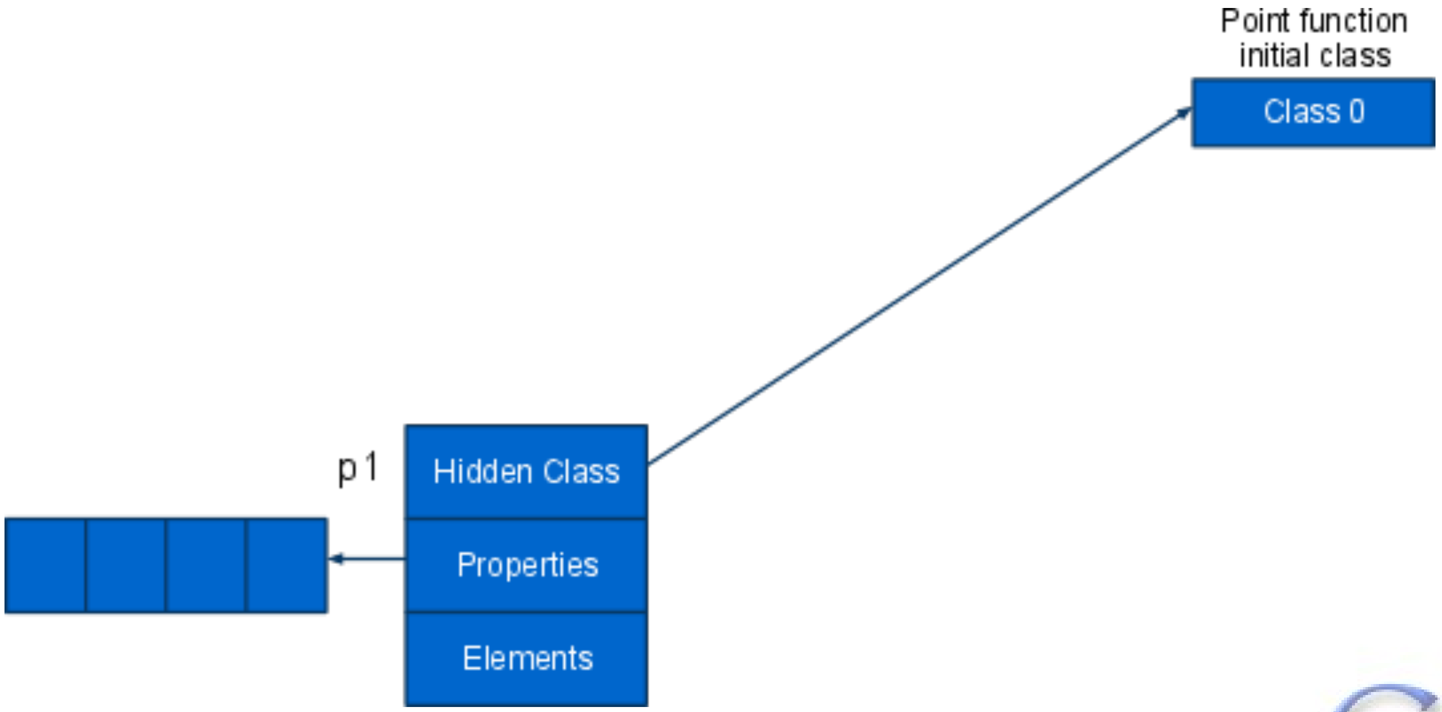


Point function initial class

Class 0

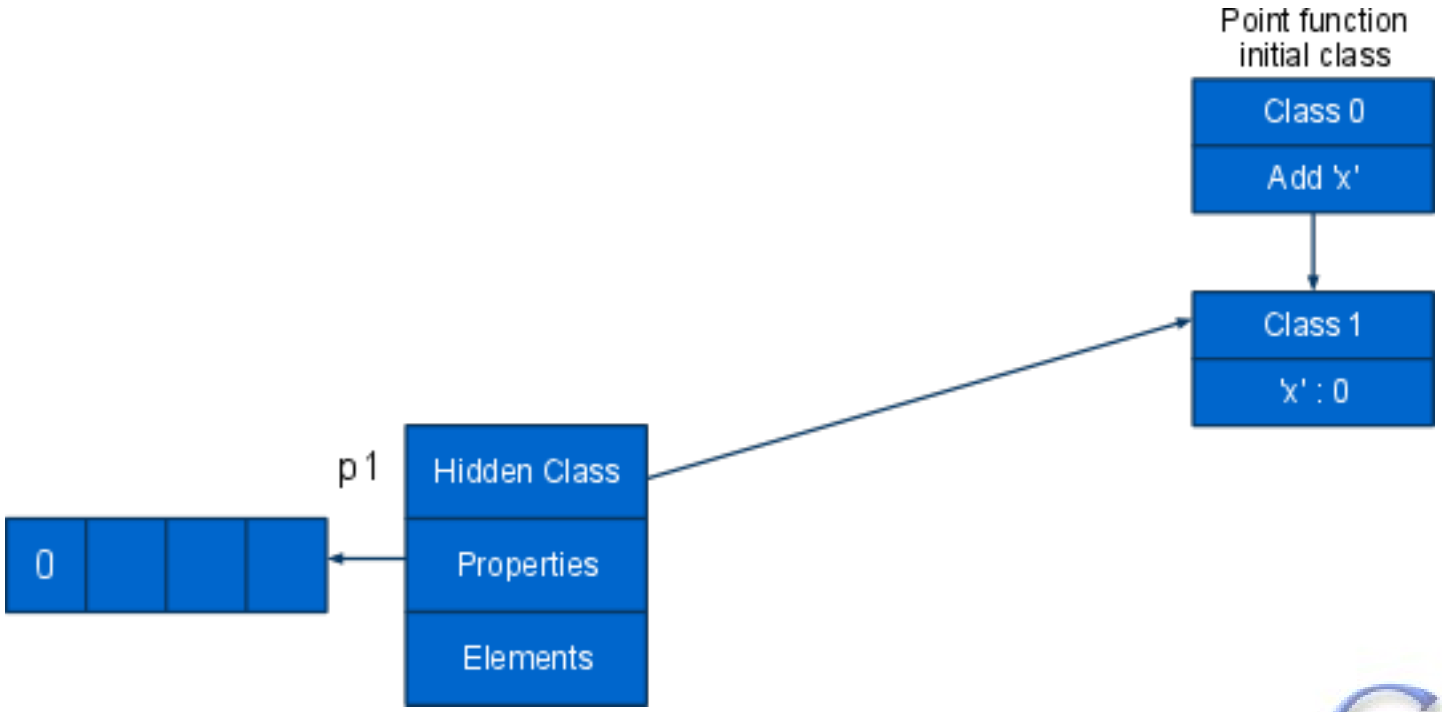
Hidden Classes by Example

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}
var p1 = new Point(0,1);
var p2 = new Point(2,3);
```



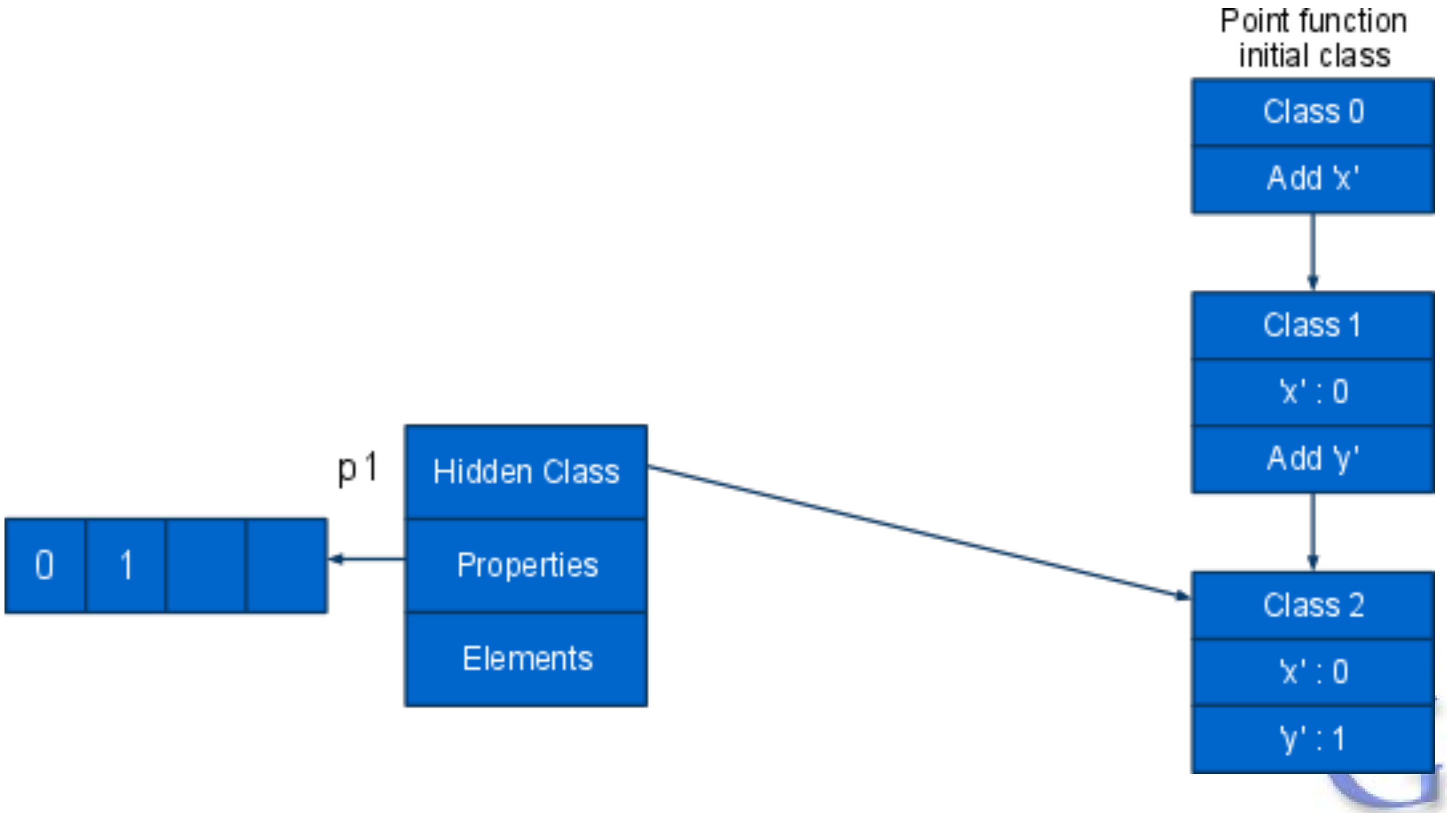
Hidden Classes by Example

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}
var p1 = new Point(0,1);
var p2 = new Point(2,3);
```



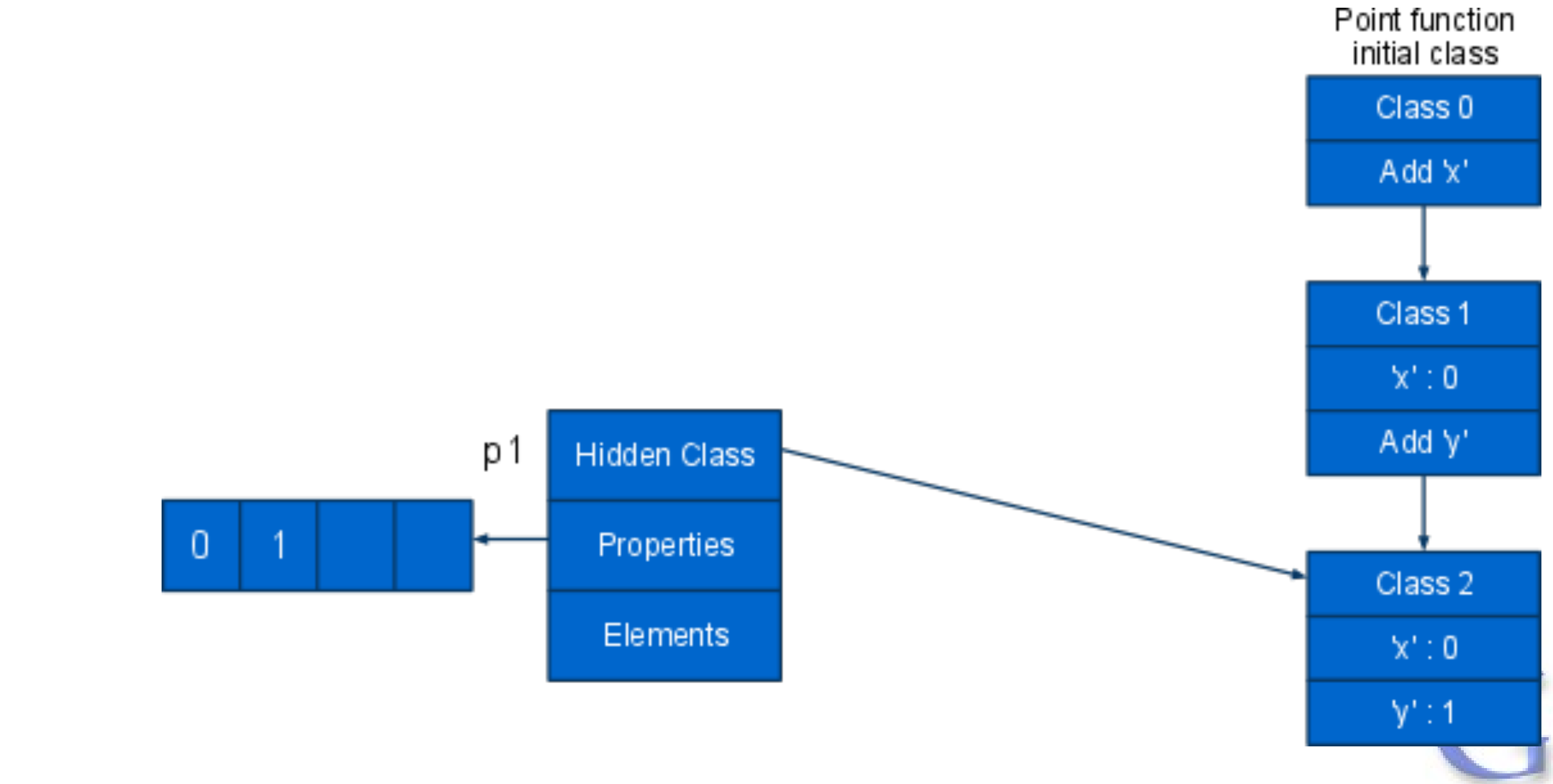
Hidden Classes by Example

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}
var p1 = new Point(0,1);
var p2 = new Point(2,3);
```



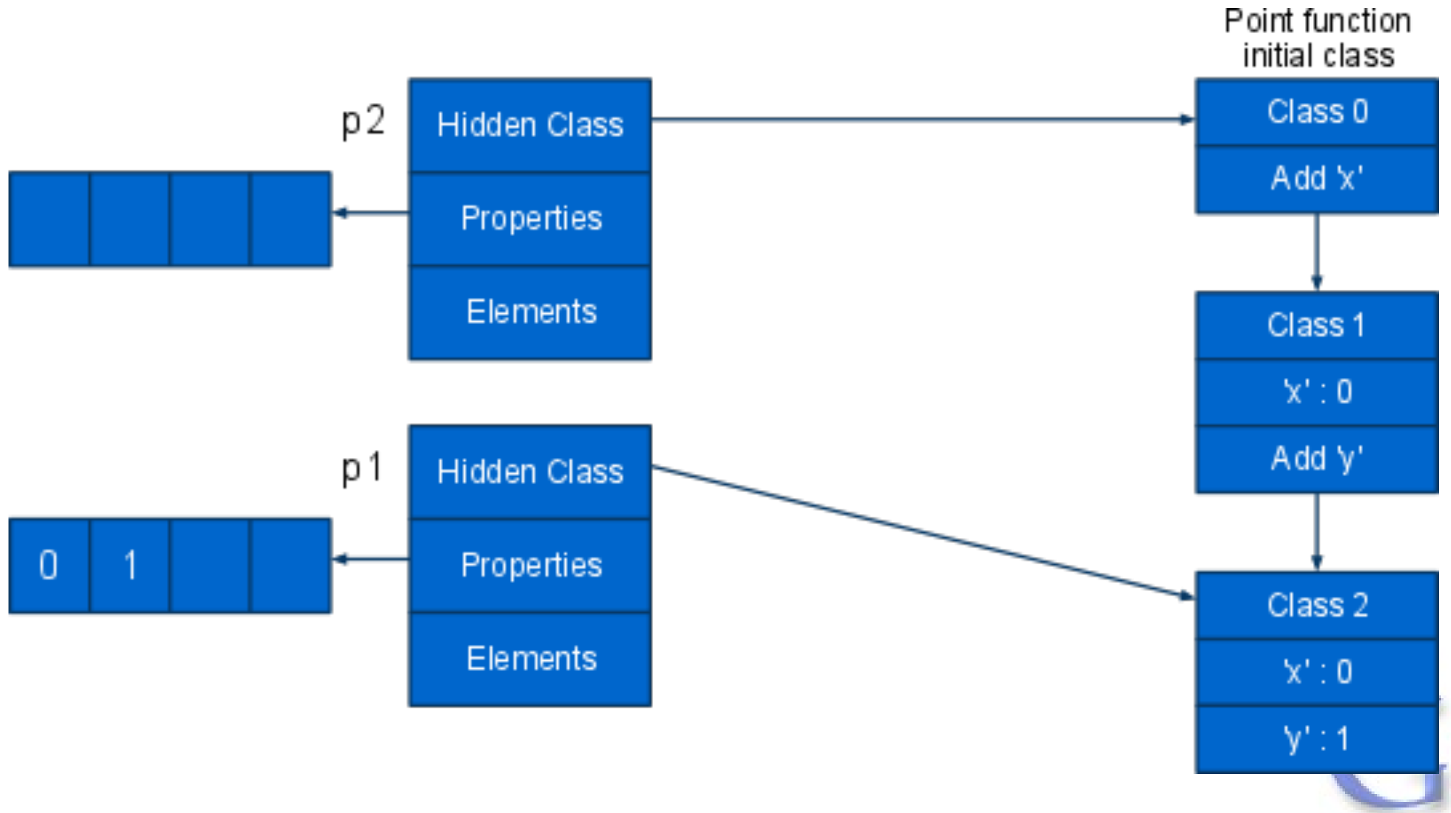
Hidden Classes by Example

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}
var p1 = new Point(0,1);
var p2 = new Point(2,3);
```



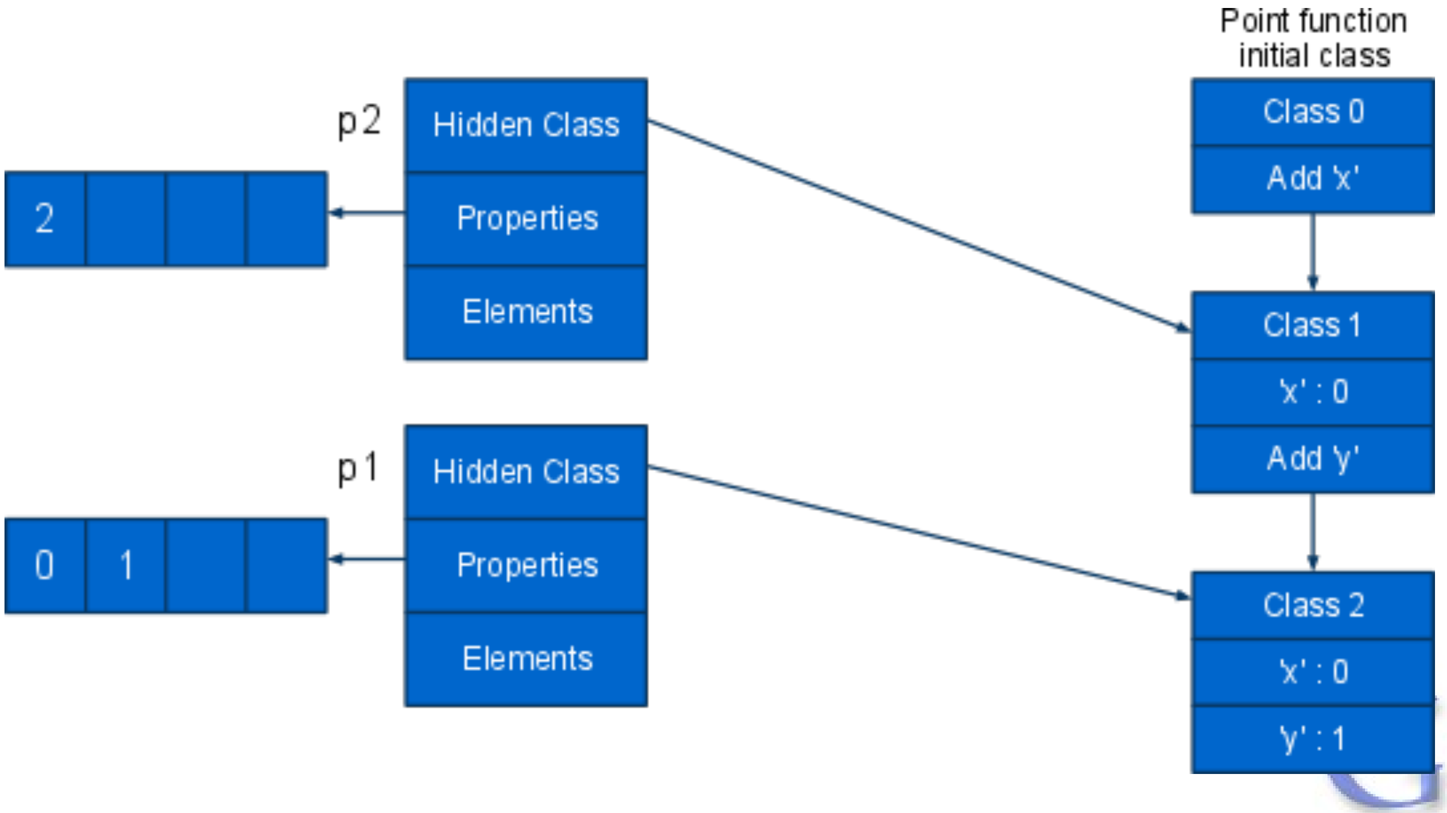
Hidden Classes by Example

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}
var p1 = new Point(0,1);
var p2 = new Point(2,3);
```



Hidden Classes by Example

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}
var p1 = new Point(0,1);
var p2 = new Point(2,3);
```



Point function initial class

Class 0

Add 'x'

Class 1

'x' : 0

Add 'y'

Class 2

'x' : 0

'y' : 1

p2

Hidden Class

Properties

Elements

2

p1

Hidden Class

Properties

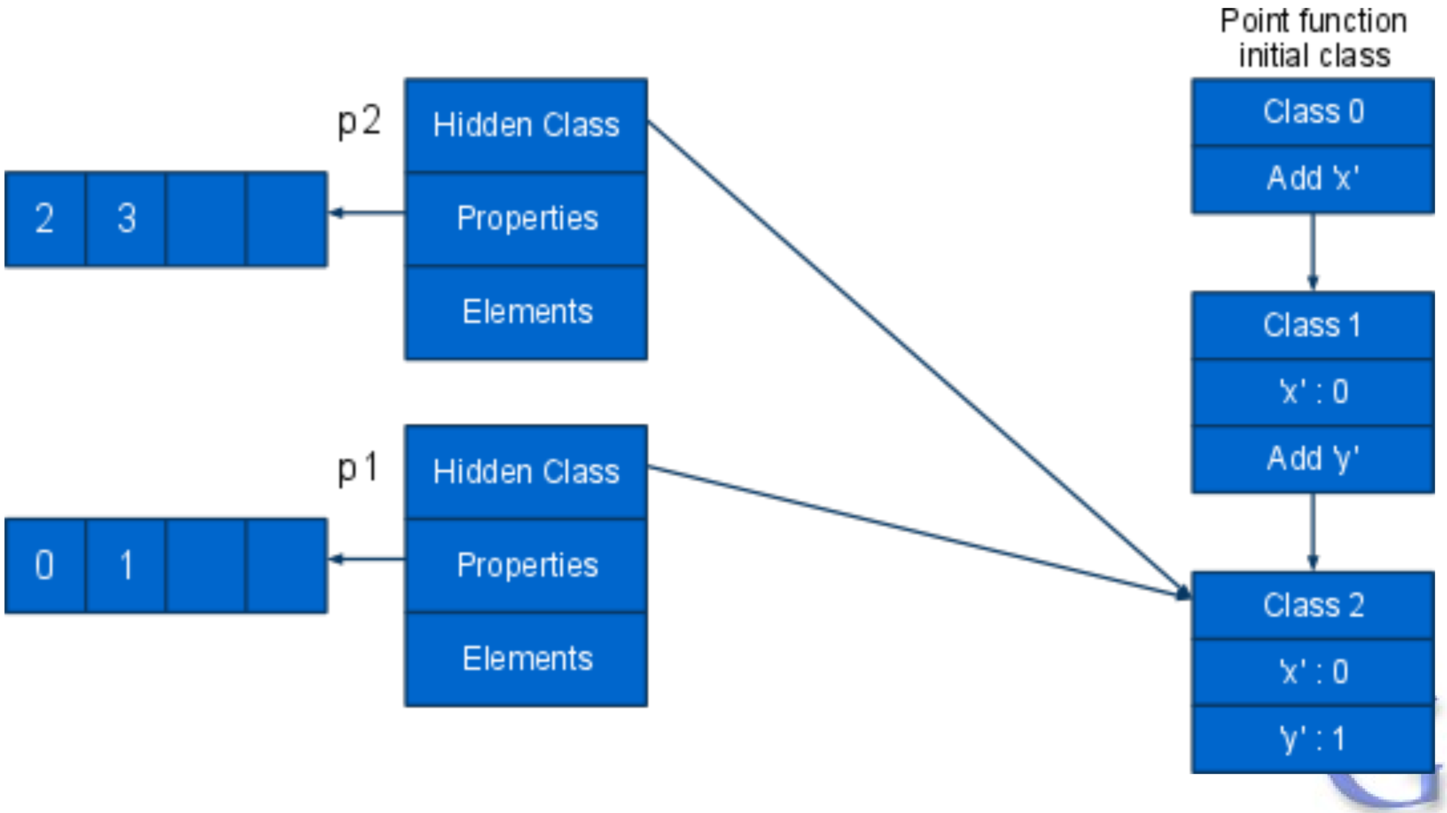
Elements

0

1

Hidden Classes by Example

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}
var p1 = new Point(0,1);
var p2 = new Point(2,3);
```



How Dynamic is JavaScript?

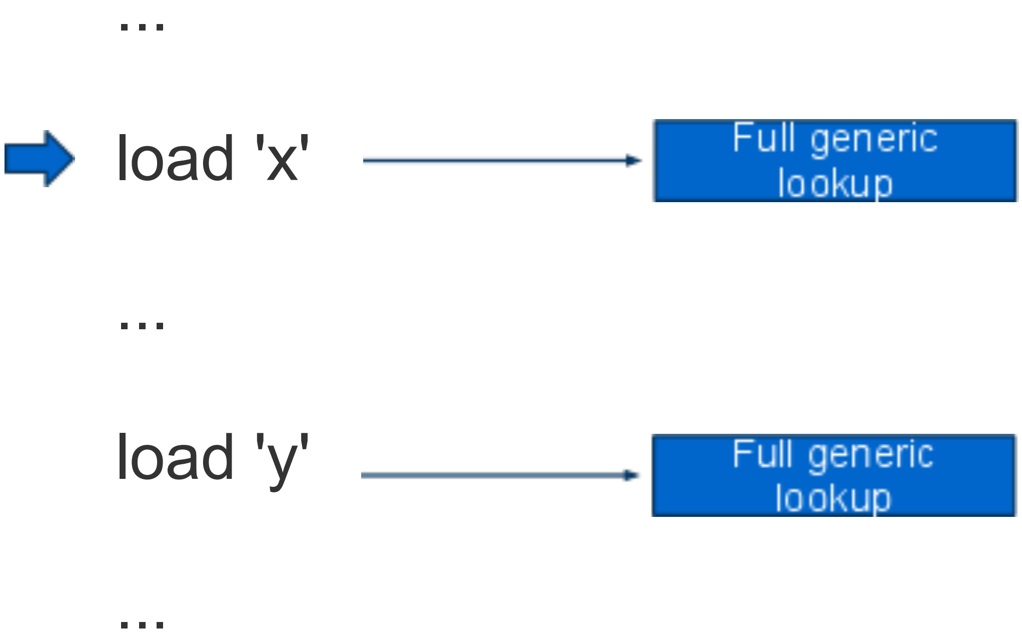
- In 90% of the cases, only objects having the same map are seen at an access site

- Hidden classes provides enough structure to use optimization techniques from more static object-oriented language

- We do not know the hidden class of objects at compile time

- We use runtime code generation and a technique called inline caching

Inline Caching



...

load 'x'

Full generic lookup

...

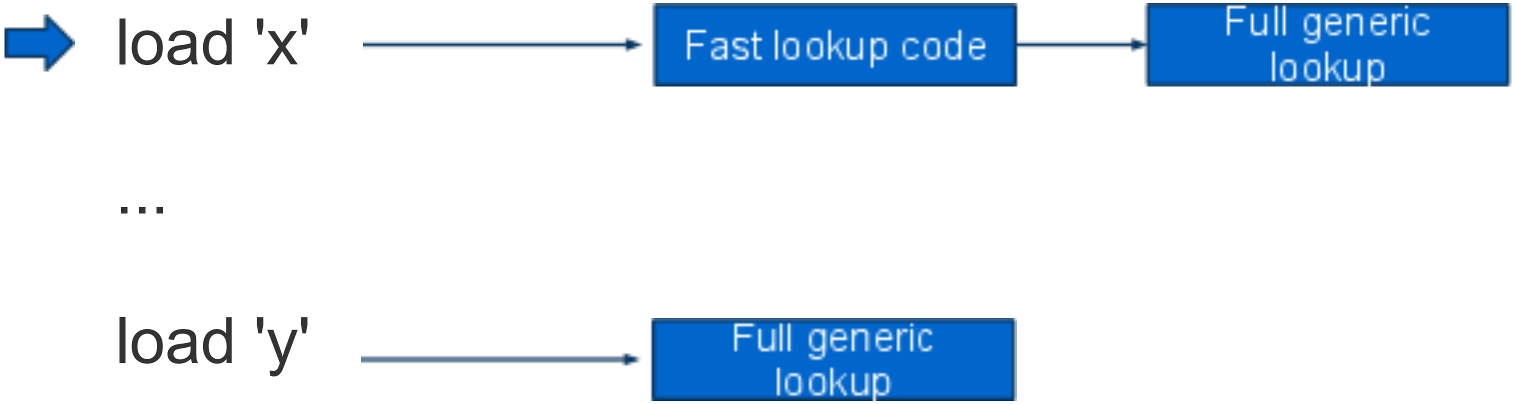
load 'y'

Full generic lookup

...

Inline Caching

...



load 'x'

Fast lookup code

Full generic lookup

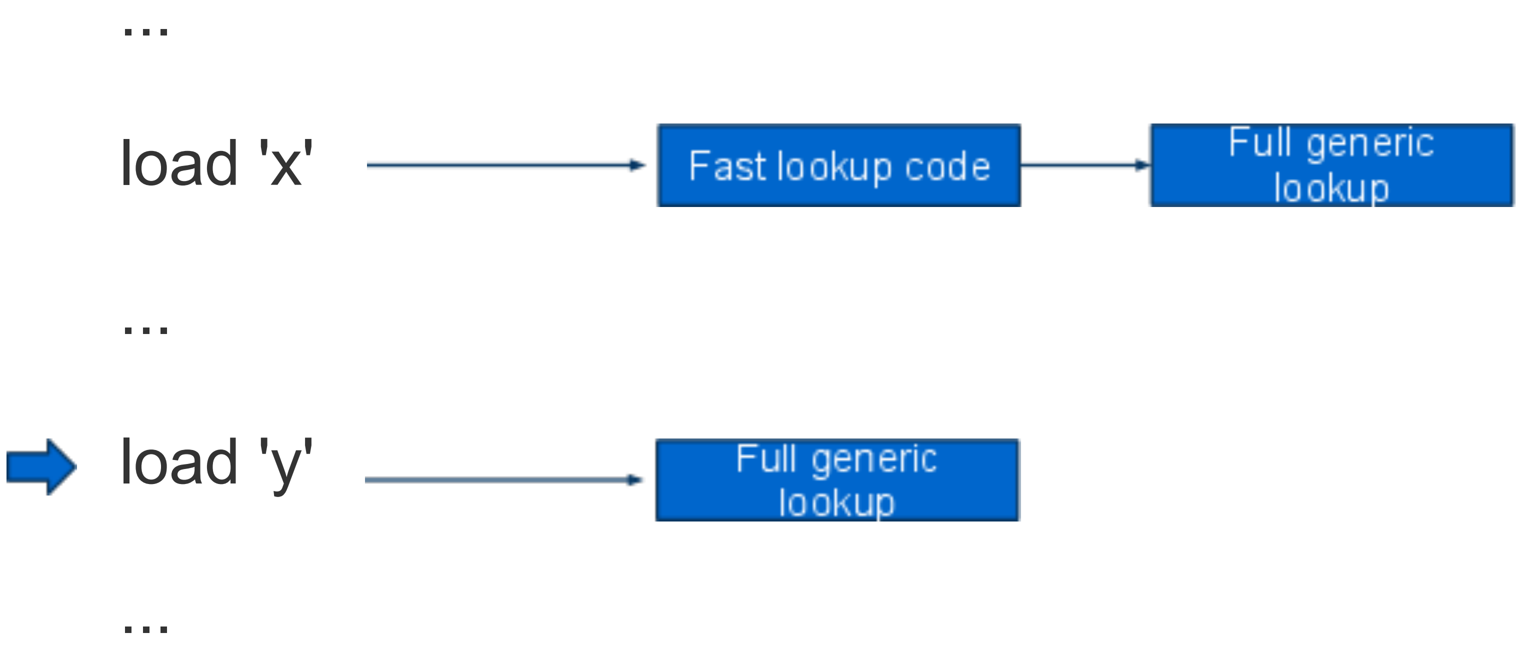
...

load 'y'

Full generic lookup

...

Inline Caching



...

load 'x'

Fast lookup code

Full generic lookup

...

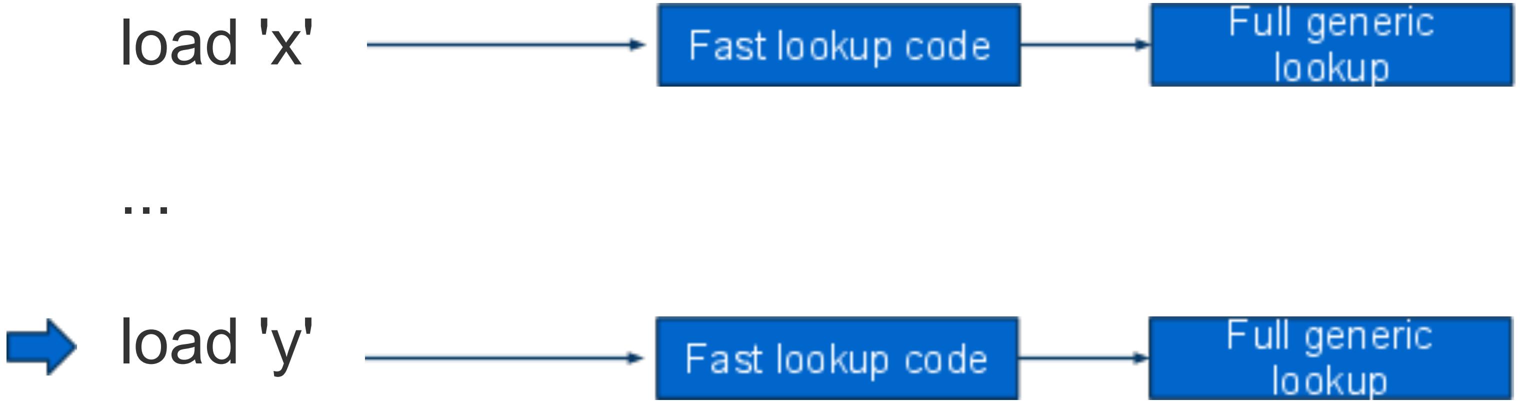
load 'y'

Full generic lookup

...

Inline Caching

...



load 'x'

Fast lookup code

Full generic lookup

...

load 'y'

Fast lookup code

Full generic lookup

...

Monomorphic Load Inline Cache

```
0xf7c0d32d (size = 37):
 0 mov eax,[esp+0x4]
 4 test al,0x1
 6 jz 32
12 cmp [eax+0xff],0xf78fab81
19 jnz 32
25 mov ebx,[eax+0x3]
28 mov eax,[ebx+0x7]
31 ret
32 jmp LoadIC_Miss
```

Monomorphic Load Inline Cache

```
0xf7c0d32d (size = 37):
 0 mov eax,[esp+0x4]        ; receiver load
 4 test al,0x1              ; object check
 6 jz 32
12 cmp [eax+0xff],0xf78fab81
19 jnz 32
25 mov ebx,[eax+0x3]
28 mov eax,[ebx+0x7]
31 ret
32 jmp LoadIC_Miss
```

Monomorphic Load Inline Cache

```
0xf7c0d32d (size = 37):
 0 mov eax,[esp+0x4]        ; receiver load
 4 test al,0x1              ; object check
 6 jz 32
12 cmp [eax+0xff],0xf78fab81 ; class check
19 jnz 32
25 mov ebx,[eax+0x3]
28 mov eax,[ebx+0x7]
31 ret
32 jmp LoadIC_Miss
```

Monomorphic Load Inline Cache

```
0xf7c0d32d (size = 37):
 0 mov eax,[esp+0x4]         ; receiver load
 4 test al,0x1               ; object check
 6 jz 32
12 cmp [eax+0xff],0xf78fab81 ; class check
19 jnz 32
25 mov ebx,[eax+0x3]         ; load properties
28 mov eax,[ebx+0x7]         ; load property
31 ret
32 jmp LoadIC_Miss
```

Monomorphic Load Inline Cache

```
0xf7c0d32d (size = 37):
 0 mov eax,[esp+0x4]        ; receiver load
 4 test al,0x1              ; object check
 6 jz 32
12 cmp [eax+0xff],0xf78fab81 ; class check
19 jnz 32
25 mov ebx,[eax+0x3]        ; load properties
28 mov eax,[ebx+0x7]        ; load property
31 ret
32 jmp LoadIC_Miss          ; fallback to
                            ; generic lookup
```

Inline Cache States

- Three inline cache states
  - Uninitialized
  - Monomorphic
  - Megamorphic
- In the megamorphic state a cache of generated stubs is used
- Inline caches are cleared on full garbage collections
  - Allows us to get rid of unused code stubs
  - Gives all inline caches a new chance to hit the monomorphic case

Memory Management

- JavaScript is garbage collected
- Users don't have to worry about deallocating memory, the virtual machine will get rid of unused memory

Memory Management Basics

- Memory is allocated from a piece of memory called the heap which is controlled by the virtual machine

- When there is too little memory left to fulfill an allocation request a garbage collection is triggered

- Basic garbage collection
  - Find all objects reachable from a root set (statics, pointers on the stack)
  - Allow the rest of memory to be reused

Mark-Sweep

- Two phases

- Mark
  - Starting from the root set perform graph traversal marking every object that is reachable

- Sweep
  - Sweep through the entire heap from beginning to end
  - For each object encountered
    - Remove mark if marked and leave alone
    - If not marked make space available for allocation (for instance by adding to a free list)

Copying

- Two separate spaces of the same size

- Always allocate from one space by just moving a pointer

- To collect garbage everything that is reachable is copied to the other space and allocation continues there

- Pros: Fast. Only live data is touched.
- Cons: Wastes space.

Efficient Memory Management

- Precise generational garbage collection
- Two generations
  - Young generation is one small, contiguous space that is collected often
  - Old generation is divided into a number of spaces that are only occasionally collected
    - Code space (executable)
    - Map space (hidden classes)
    - Large object space (>8K)
    - Old data space (no pointers)
    - Old pointer space
- Objects are promoted from young to old generation if they survive

Types of Garbage Collection

- Scavenge
  - Copying collection of only the young generation
  - Pause times ~2ms
- Full non-compacting collection
  - Mark-Sweep collection of both generations
  - Free memory gets added to free lists
  - Might cause fragmentation
  - Pause times ~50ms
- Full compacting collection
  - Mark-Sweep-Compact collection of both generations
  - Pause times ~100ms

Agenda

- Part 1: What is JavaScript?
- Part 2: V8 internals
- Part 3: V8 testing and benchmarking

Testing

- V8 is shipped in Google Chrome every week
- We need V8 to be stable
- We need to push performance and be aggressive
- Therefore, we need a lot of testing

Testing

- For each commit to the V8 repository
    - V8 tests (>1200 tests)
    - Mozilla JS tests (>1900 tests)
    - Sputnik (>5000 tests)
    - ES5 conformance tests (>1200 tests)

- Many test configurations
    - Linux x { ia32, x64, arm simulator } x { release, debug }
    - Windows x { ia32, x64 } x { release, debug }
    - Mac x ia32 x { release, debug }

Testing

- For each V8 commit
  - WebKit layout tests run in Chromium test shell

- Push new version to Chromium twice a week
- After each push a lot of additional testing goes on on the Chromium buildbots

http://build.chromium.org/buildbot/v8
http://build.chromium.org

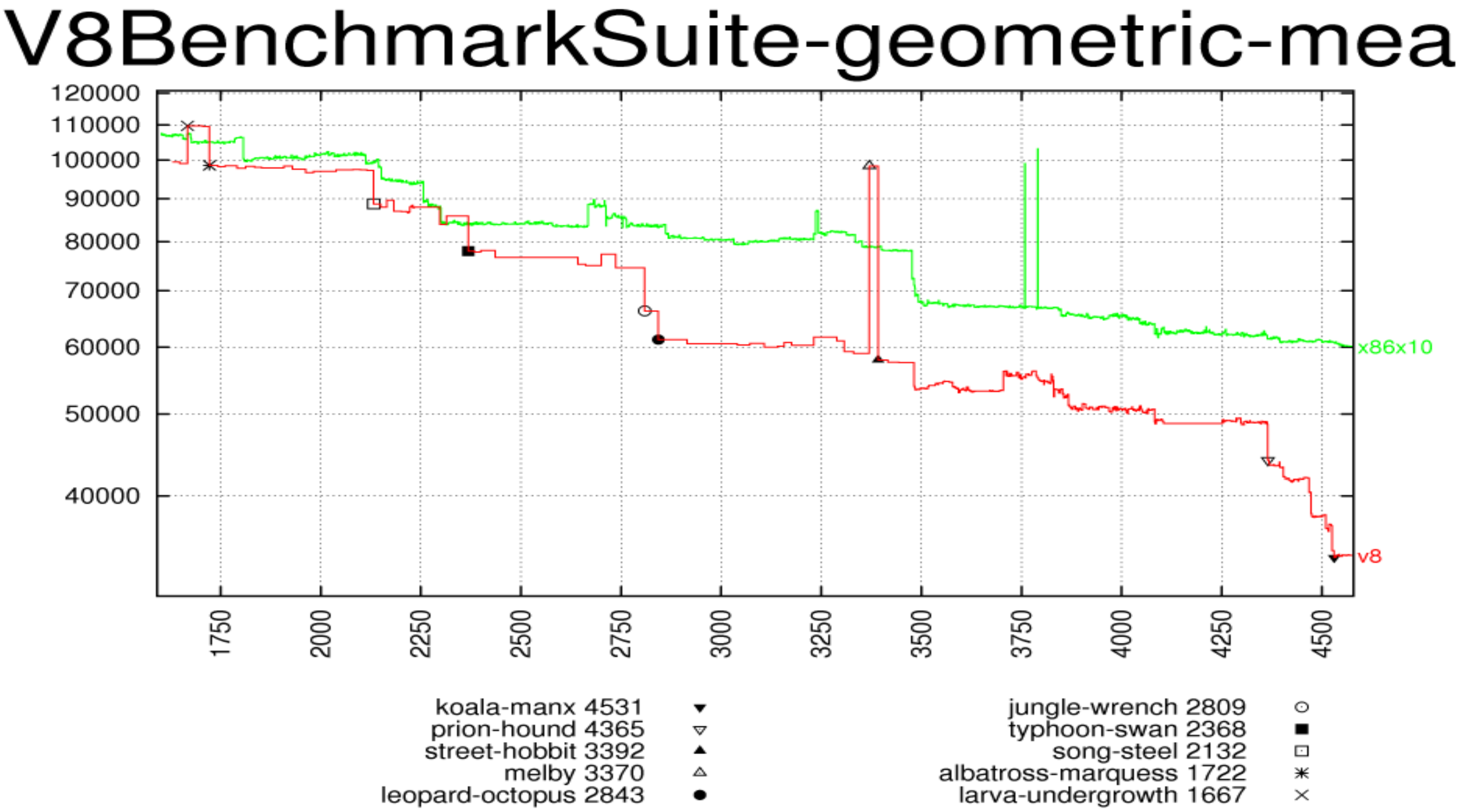
Testing

- Testing is crucial to making progress
- Having good test coverage allows us to make aggressive changes with less fear of breaking stuff
- When fixing a bug you add a new test
- When implementing a new feature you add a new test

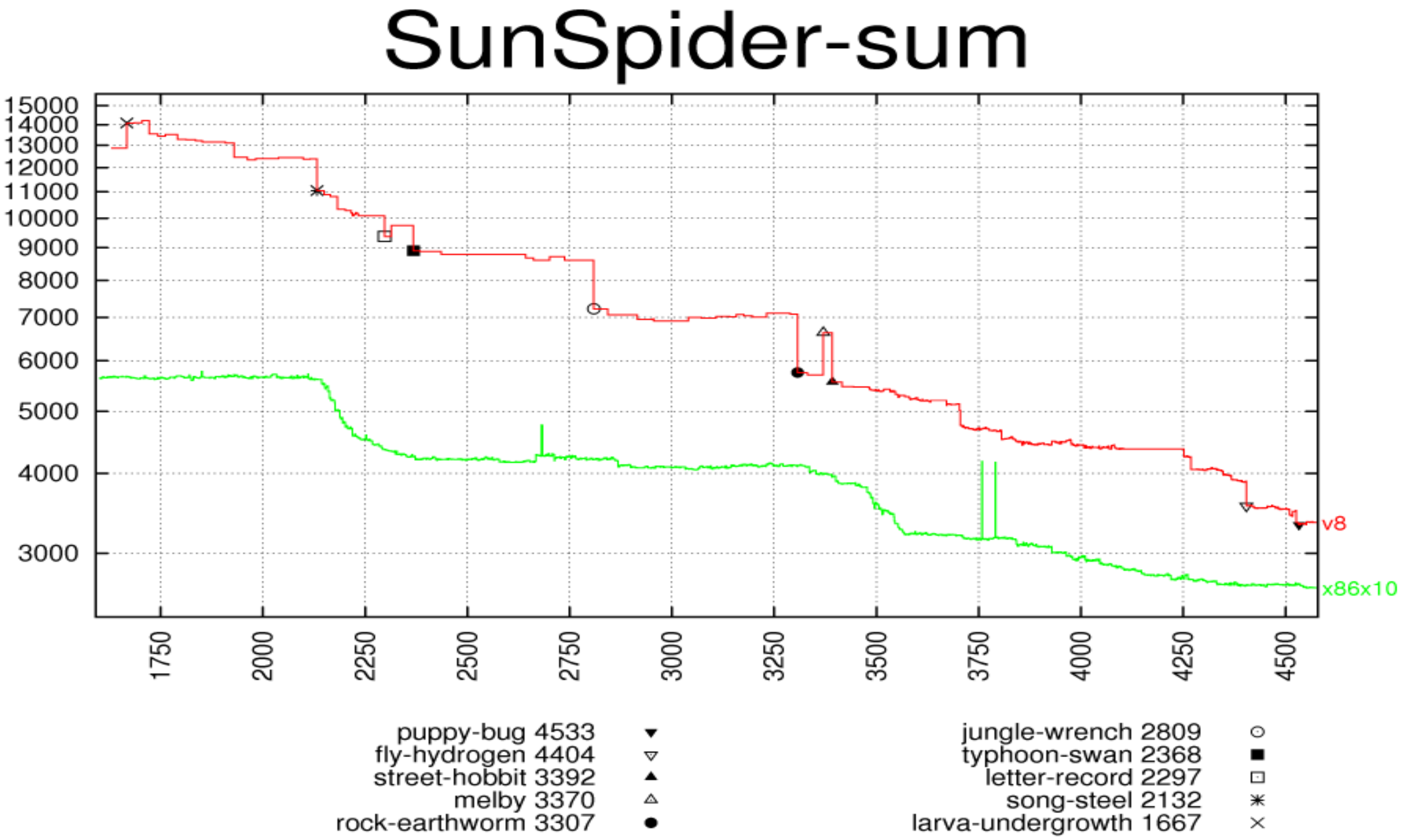
Benchmarking

- We want to push JavaScript performance
- Need to track how we are doing
- We have built an infrastructure that runs a lot of benchmarks on every commit to the V8 repository
- Easy to keep track of progress and easy to spot regressions

Performance tracking

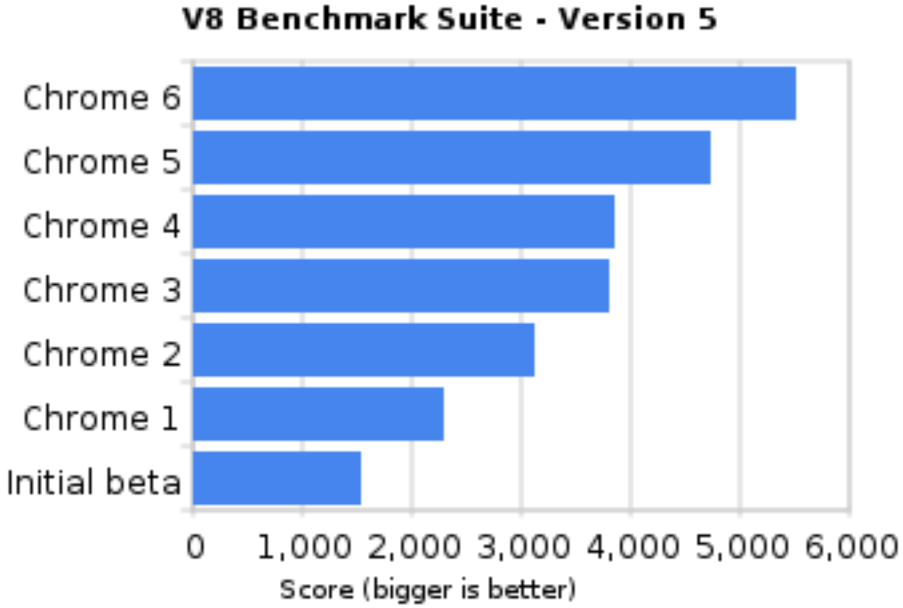


Performance tracking



Performance

- Goal was to push performance
- Performance is increased on every release of Google Chrome



Questions?

http://v8.googlecode.com

http://chromium.org