
Google
Developers

# Web Languages and VMs

or why fast code is always in fashion

Lars Bak & Kasper Lund
Software Engineers at Google Inc.

*Hacking on VMs side by side the last 13 years...*

Smalltalk
Hotspot
Crankshaft
Beta
CLDC
JVM
OOVM
Dart
V8
Self

# So Why Are We Here?

Speed fuels application innovation

Web browsers are faster than ever, but are they fast enough?

We will convince you that Dart takes performance to the next level


# Remember the Browsers of 2006?

1998 2000 2002 2005 **2006**

2008

**Firefox 2.0**

**Internet Explorer 7.0**

**Safari 2.0**

**Opera 9.0**

# Browser Performance Beliefs in 2006

Browsers were believed to be "fast enough"

- Web apps like Gmail and Google Maps ran fine
- JavaScript was inherently too slow for heavy client side computations
- JavaScript execution was not perceived as a bottleneck

Performance was evaluated using micro-benchmarks

- Emphasis was put on loops and simple arithmetic
- Dynamic dispatching and memory management were sadly neglected

# SunSpider: bitwise-and

Benchmark from SunSpider version 0.9.1

JavaScript

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

# SunSpider: bitwise-and

Benchmark from SunSpider version 0.9.1

JavaScript

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

... but this always yields zero?

# SunSpider: bitwise-and

Benchmark from SunSpider version **1.0**

JavaScript

```
bitwiseAndValue = 4294967296;
for (var i = 0; i < 600000; i++)
    bitwiseAndValue = bitwiseAndValue & i;
if (bitwiseAndValue != 0) throw "ERROR: bad result...";
```

... and this always yields zero!

# V8 Design Choice: What to Optimize For?

(1) Optimize for current apps and benchmarks

- Simple and incremental approach
- Attempt to make things 10% better

(2) Optimize for the apps of the future

- Support heavy client side computations
- Turn the browser into a scalable application platform
- Enables a new class of web apps

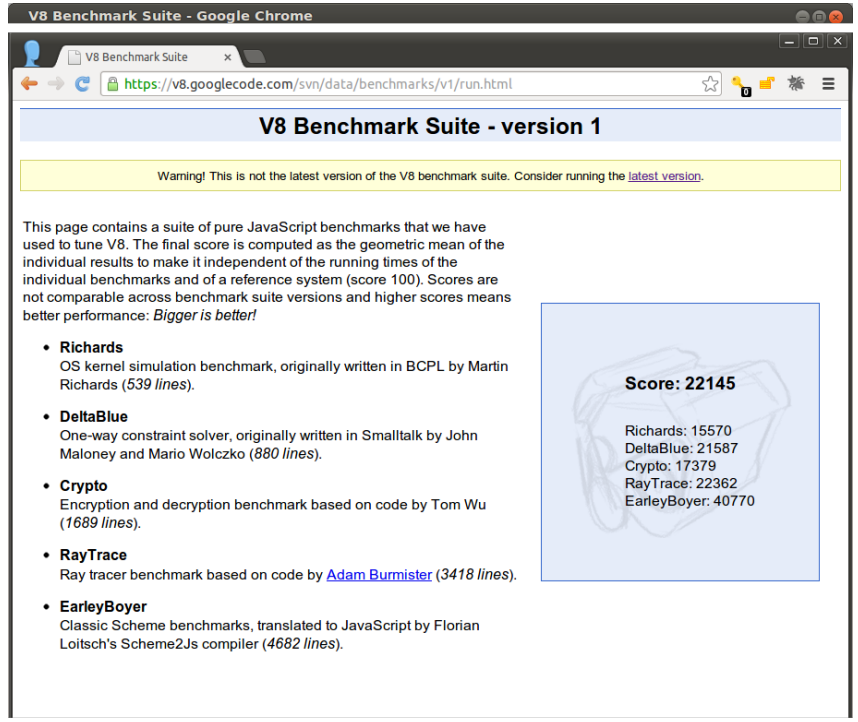
# V8 Benchmark Suite

Consists of benchmarks that are

- Structured and mostly object-oriented
- Designed to push the limits
- Proven valuable in the context of other languages

Measures the performance of

- Dynamic method calls and property accesses
- Memory management
- Closure creation and invocation




# Up, Up and Away!

Performance improvements from 2006 - 2013

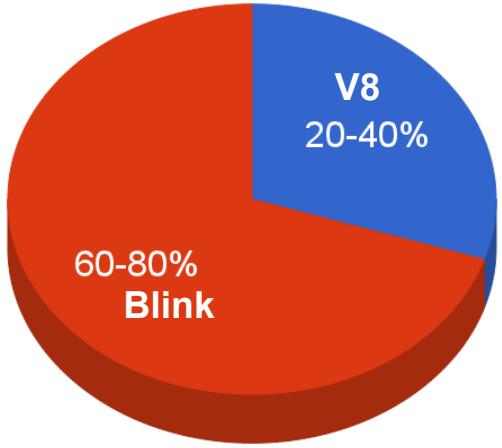
- JavaScript executes more than **100x** faster
- Average heap sizes are up and GC pauses are down
- Benchmark numbers are frequently reported in the press

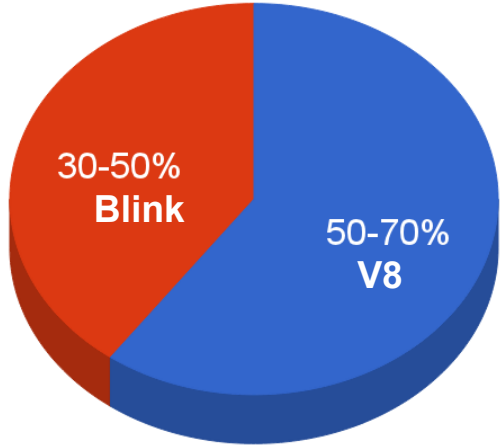
Web apps have become huge

- **amazon.com** ~ 600K JavaScript
- **cnn.com** ~1500K JavaScript
- **espn.com** ~ 900K JavaScript

# Where Is the Time Spent Today?

Twitter
Facebook
YouTube
Wikipedia




Gmail
Google Docs
Google Search
Google Calendar

*No matter how fast the web engines get, the extra performance is devoured by inventive web developers*

# The Web Anno 2013

Web developers are pushing the limits and demand

- Predictable and higher performance
- Consistent frame rates for games
- Support for large scale app development

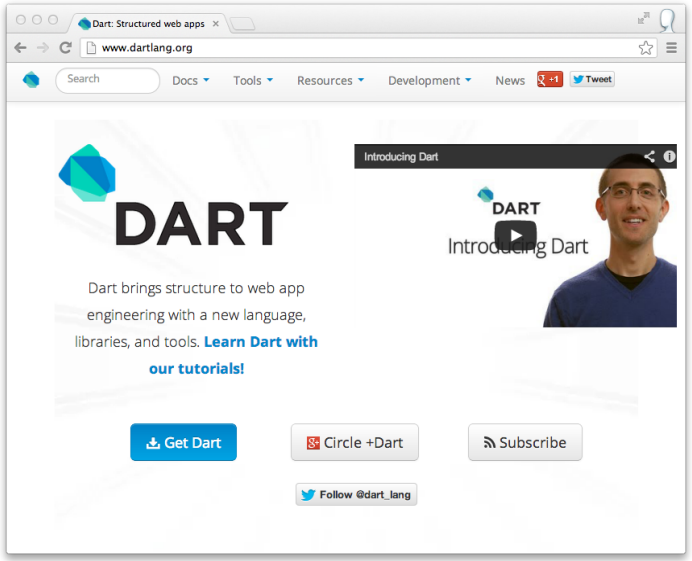
It only took a **100x** performance increase to change people's expectations

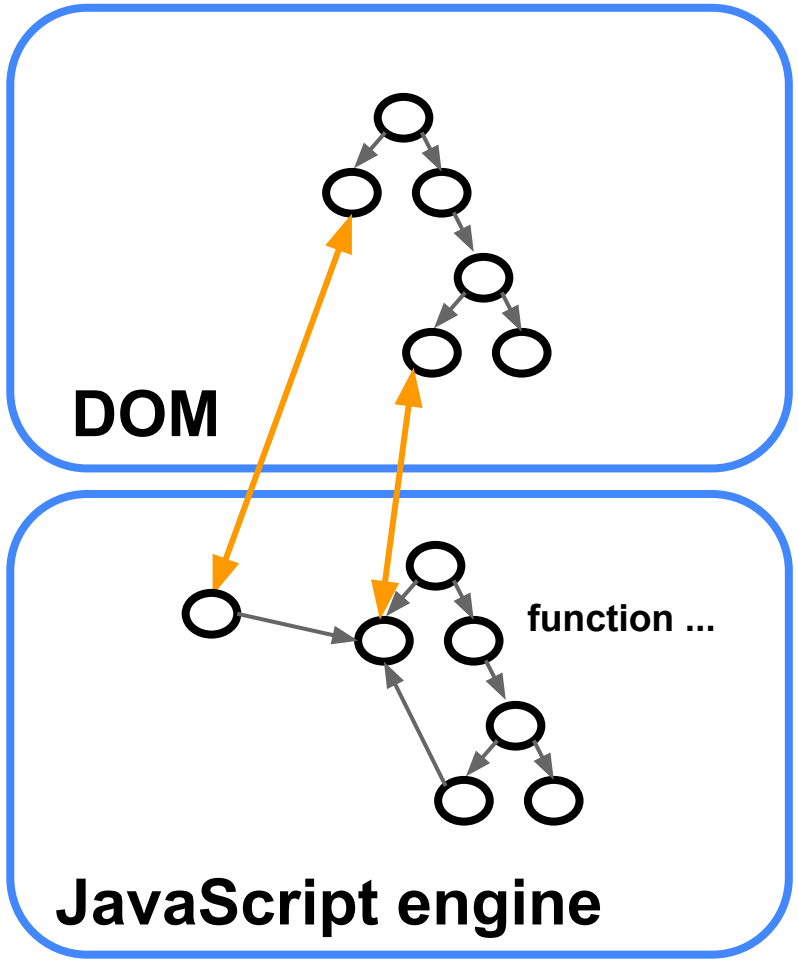
*Let's take a look at the current web technology stack and see how we can improve it!*

# Modern Web Engine Technology

and where does it come from?

# What's Behind a Web App?




## Hopes

- Low startup latency
- High performance
- Low memory usage
- Small pauses

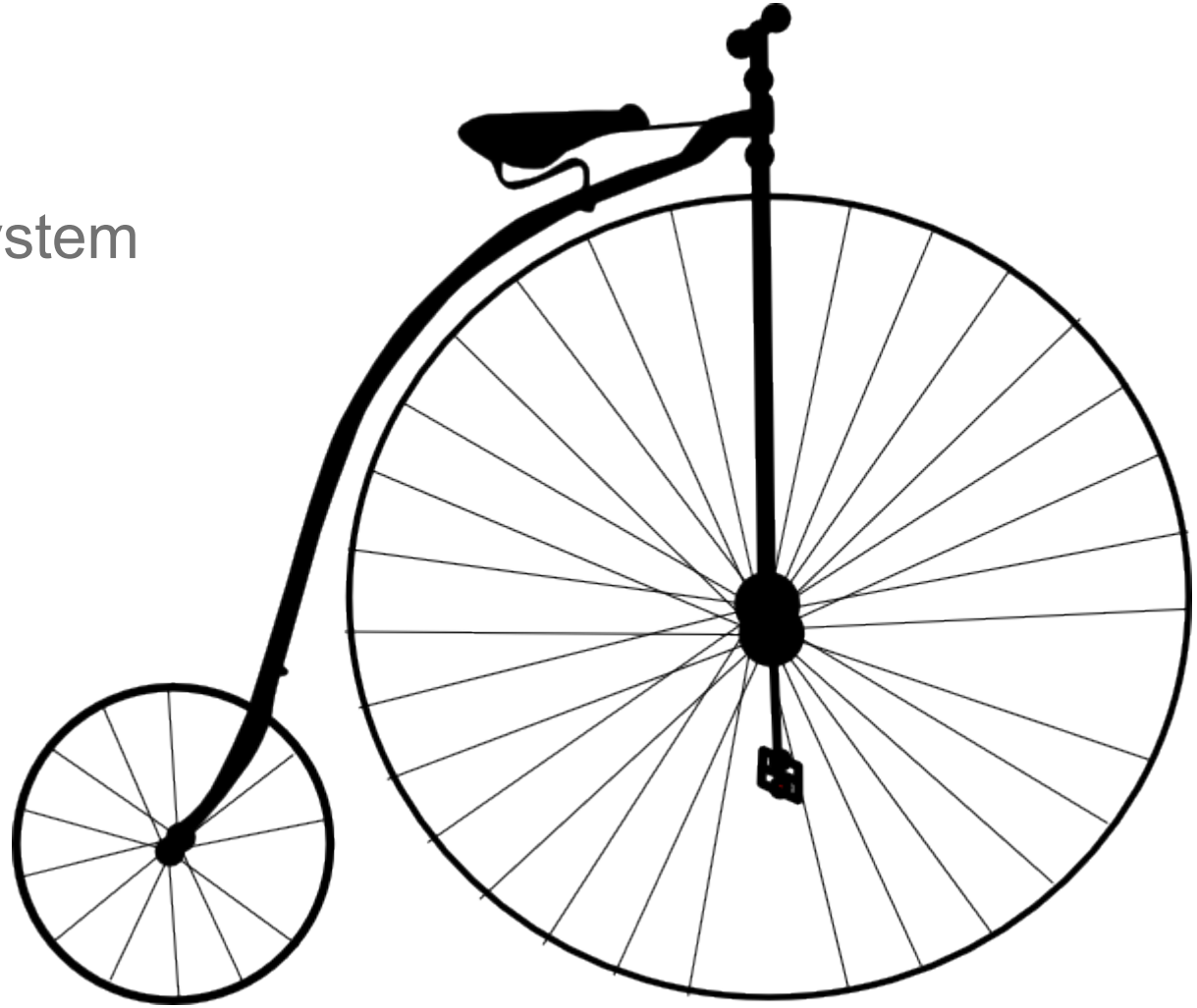
## Fears

- Big GC pauses
- Memory leaks
- Erratic performance

# Picture of a JavaScript Engine Anno 2006

Parser
Interpreter
Simple memory management system

*Simple but slooooooow...*



# Picture of a Modern JavaScript Engine

Parsing
Multi-tier adaptive compilation
Deoptimization
Generational garbage collection
Code flushing
Debugging and profiling support

*Complex but fast...*

# Let's Look Inside the V8 Engine

How is JavaScript code made fast?

*Multi-tier adaptive compilation*

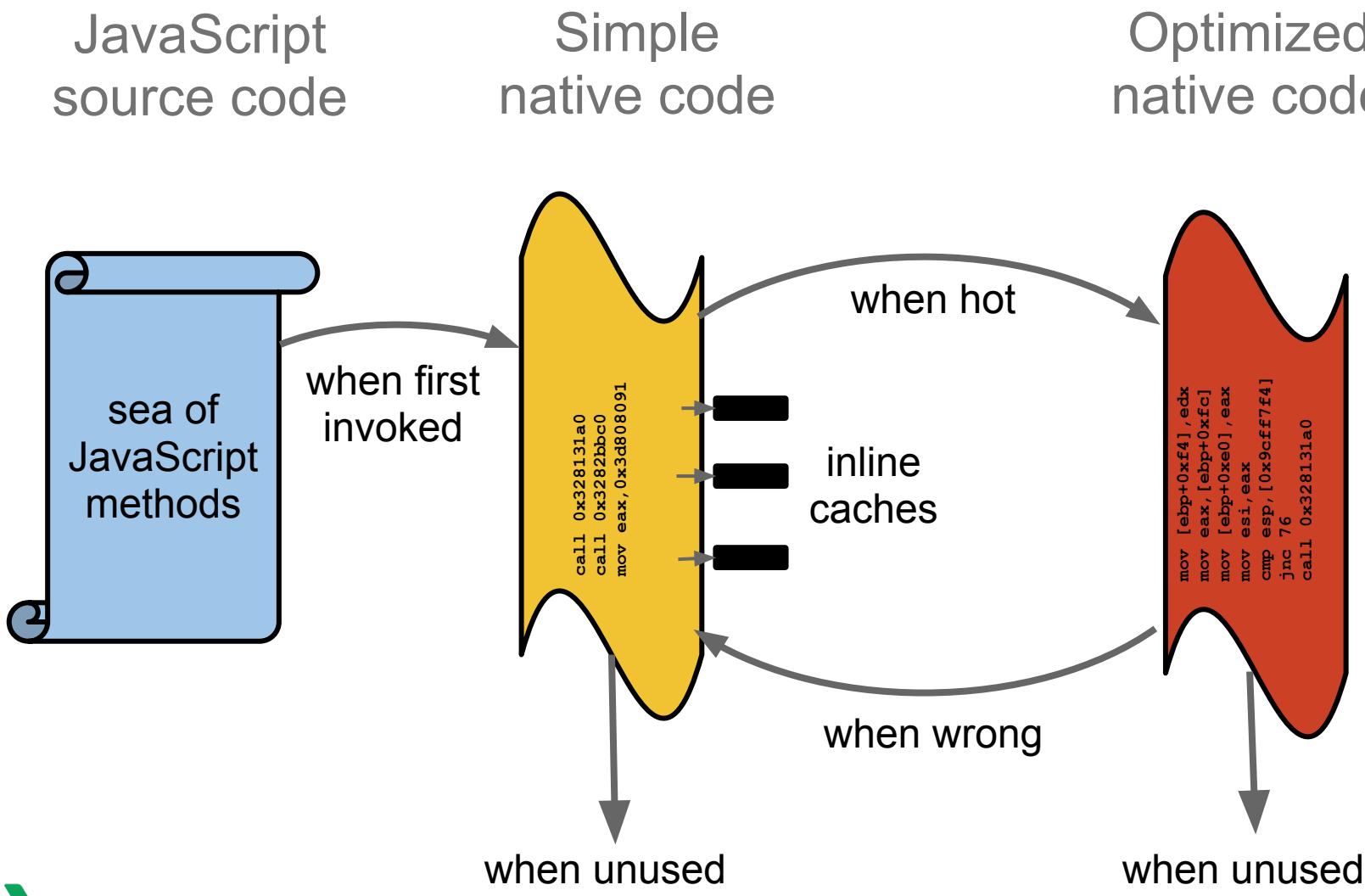
How do we handle large object heaps?

*Generational garbage collection with a twist*

How do JavaScript objects bind to DOM nodes?

*Tracing GC tangoing with reference counting*

# Multi-tier Adaptive Compilation




Runtime type feedback
Aggressive inlining
Deoptimization
On-the-fly code patching
Object morphing

# Did We Invent Multi-tier Adaptive Compilation?

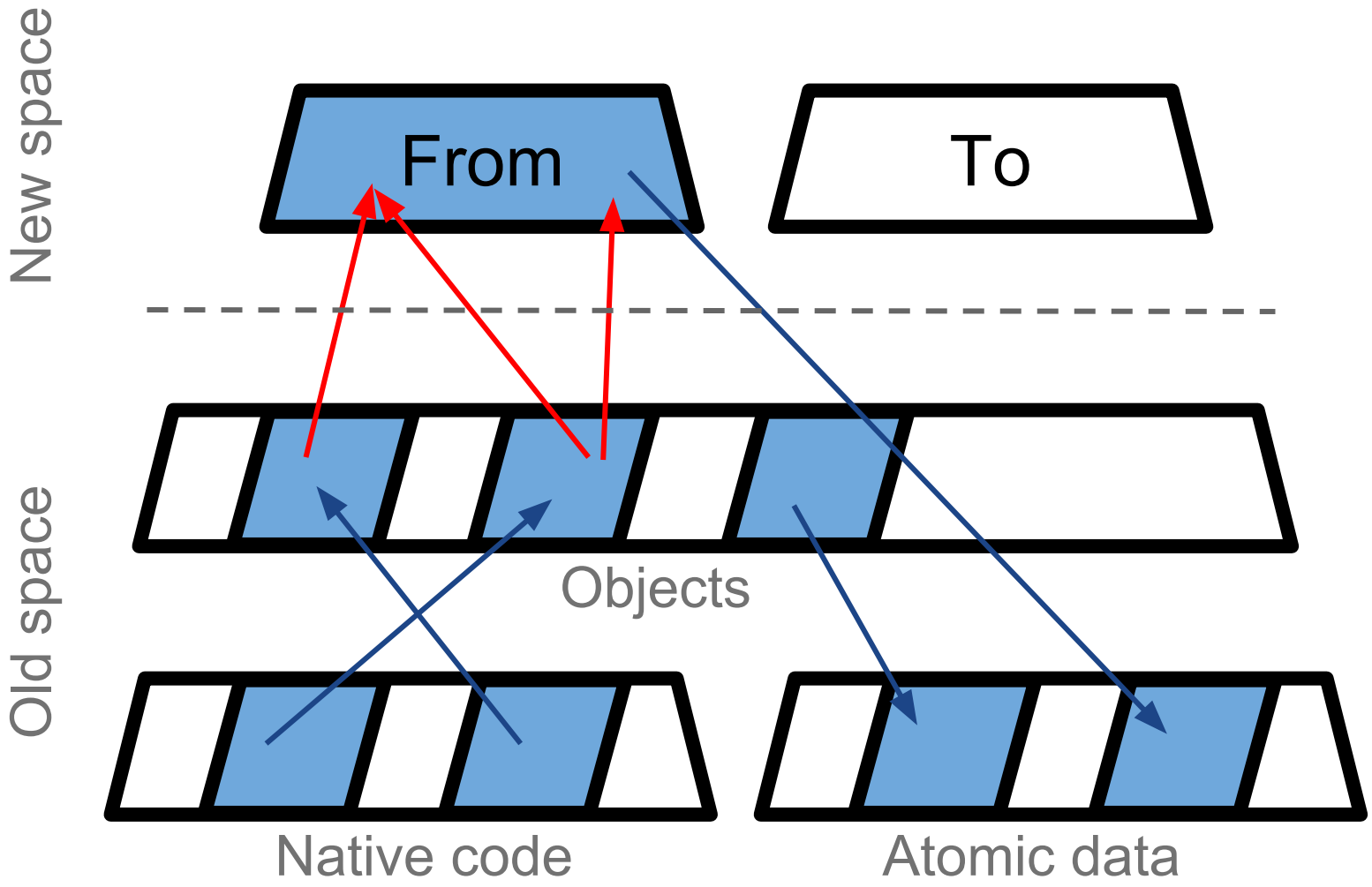
## No

**History of dynamic language execution**

- Interpretation, 59-62: Lisp
- Dynamic compilation, 70: Smalltalk 80
  - Inline caching check in callee methods
- Adaptive compilation, 90: Self
  - Mixed-mode execution, runtime type feedback, adaptive optimizations
  - Deoptimization and on-stack replacement
  - First appeared commercially in the Hotspot JVM

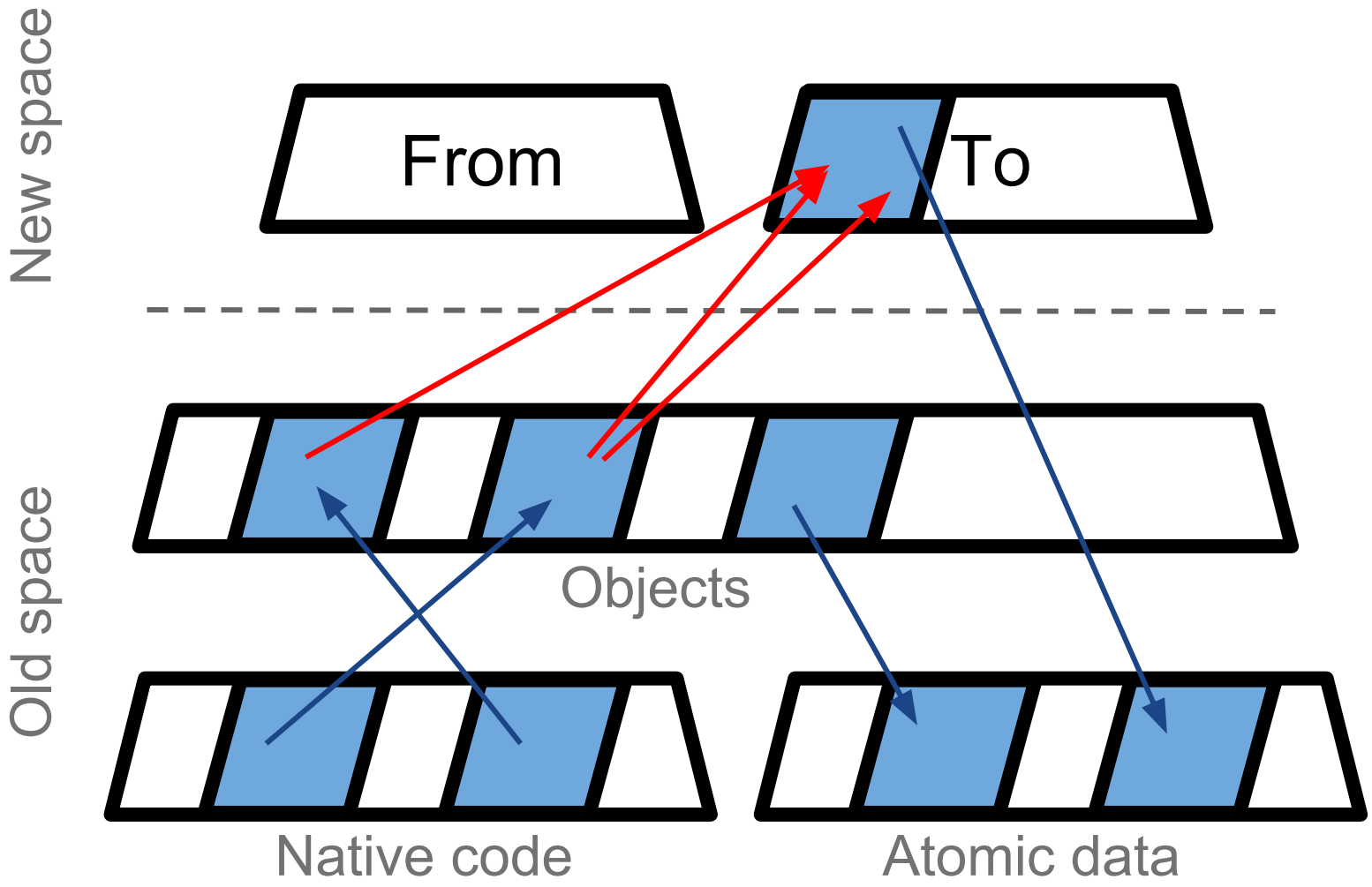
*Introducing behind-the-scene classes in V8 allowed us to use all this!*

# Generational Garbage Collection with a Twist




- Stop-the-world single threaded
- Store buffer enables generations
- Age-based promotion
- Incremental marking in old space

# Generational Garbage Collection with a Twist




- Stop-the-world single threaded
- Store buffer enables generations
- Age-based promotion
- Incremental marking in old space

# Did We Invent Generational Garbage Collection?

## No

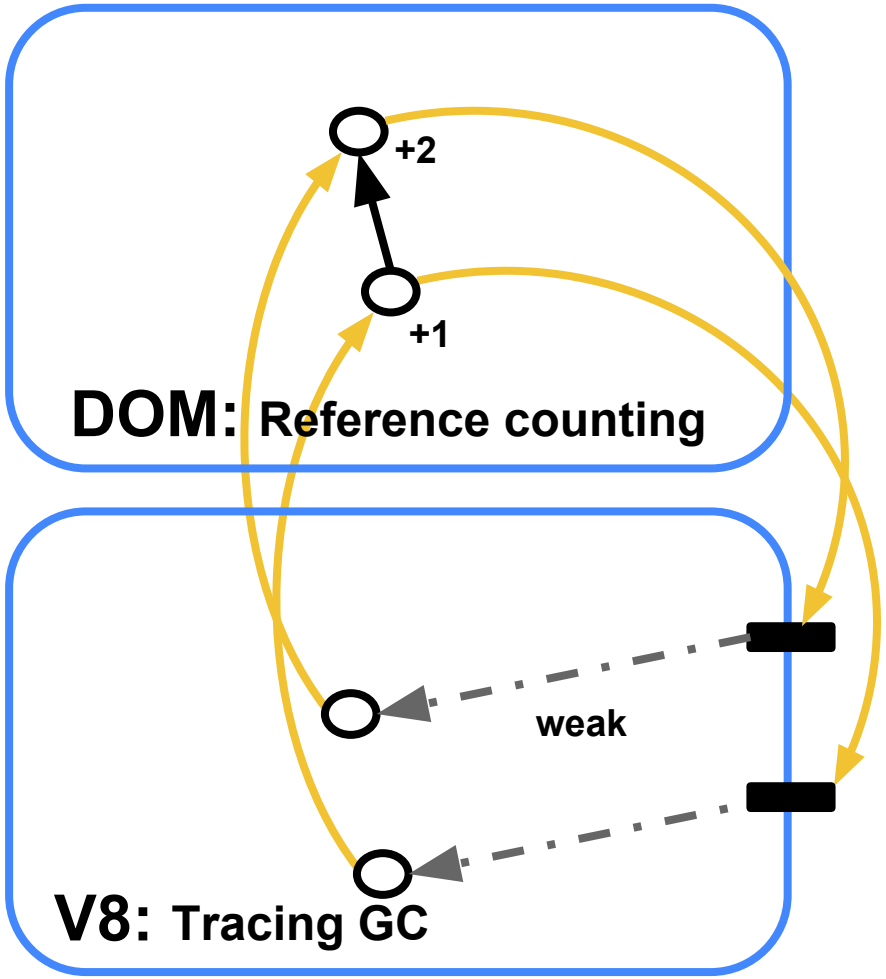
**History of automatic memory management**

- Garbage collection, 58: Lisp
- Incremental garbage collection, 75: Lisp
- Generational scavenging, 83-84: Smalltalk

**This is relatively simple compared to interacting with the browser**

- Nodes in the DOM are reference counted
- JavaScript objects are traced

# Tracing GC Tangoing with Reference Counting




Setup

- Three pointer pairing
- JavaScript wrapper objects are referred through handles
- JavaScript wrapper objects are reclaimed in DOM groups to avoid object resurrection

Problems

- DOM code cannot have cycles
- To avoid cycles, raw pointers are used
- Substantial processing overhead

It works but it is brittle and really hard to maintain

Warning: any cycles between the segments results in memory leaks

# Did We Invent This Morass?

## Yes

We are not proud of it and we believe it needs more work
We will get back to what we are doing about this later

# Does This Mean V8 Is As Good As It Gets?

Advanced compilation and runtime tricks have been used

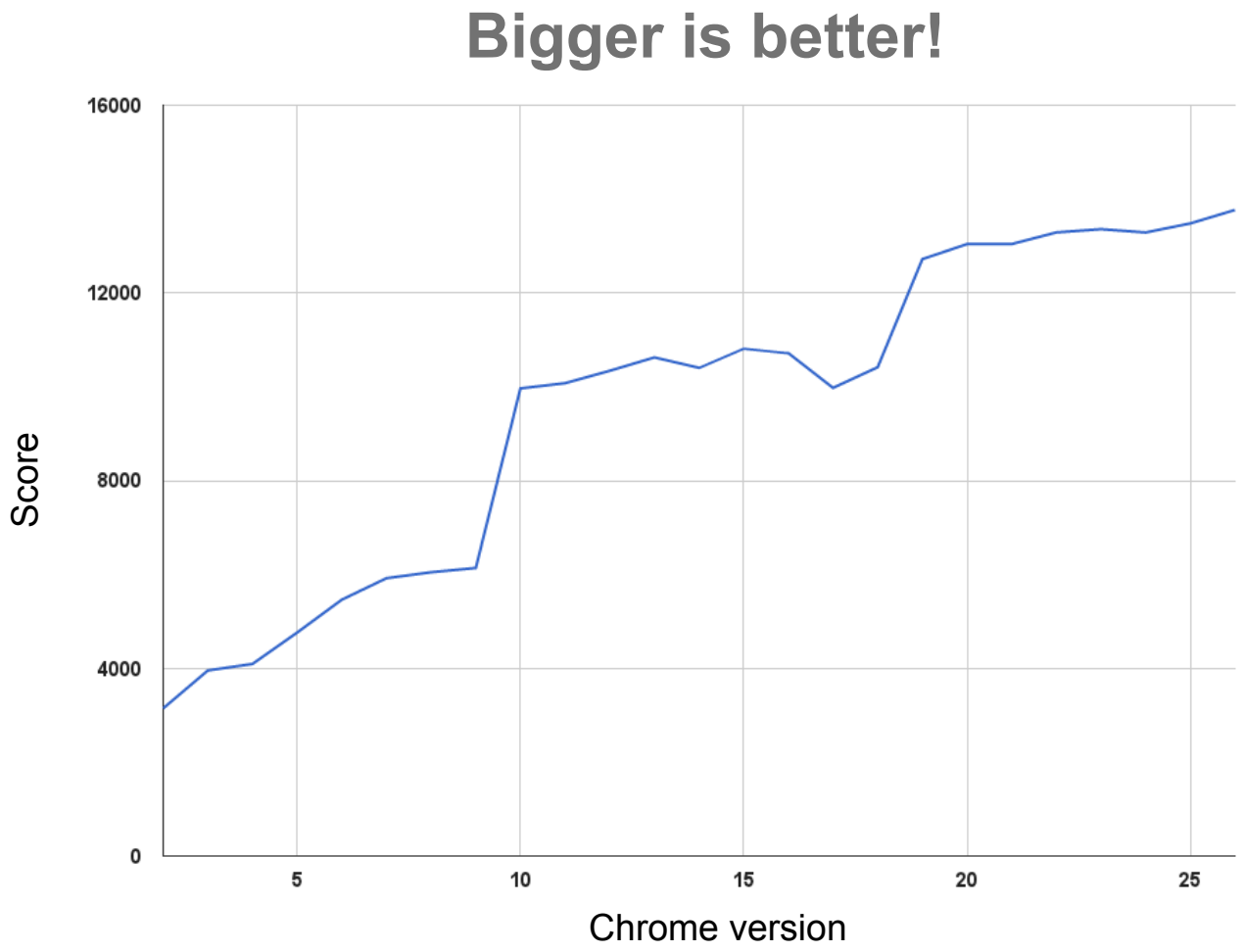
- Hidden classes
- State-of-the-art adaptive compilation
- Sophisticated memory management

Yet real bottlenecks still exist

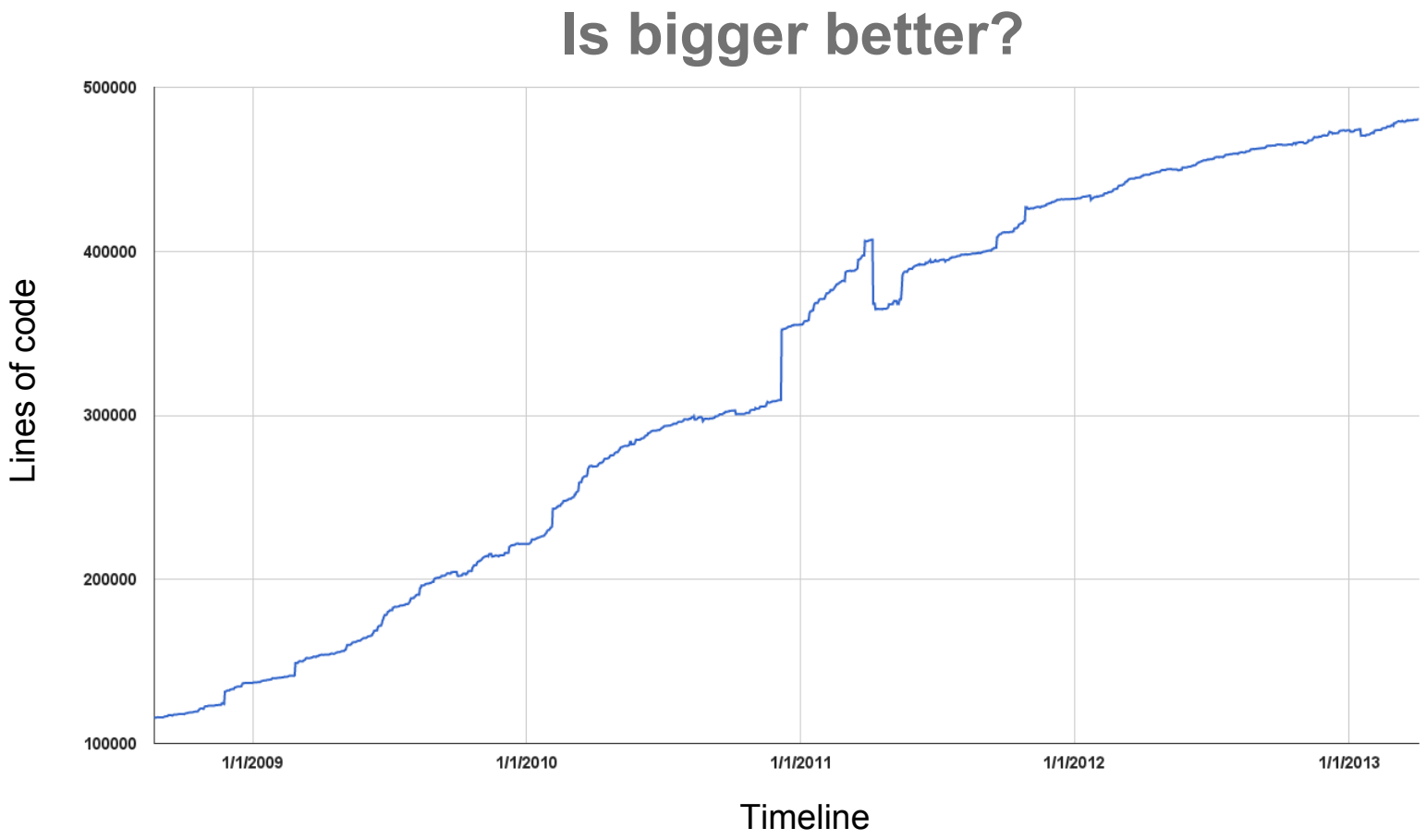
- Performance does not match 'real' languages
- Performance is unpredictable
- Startup is slow
- JavaScript is still ... JavaScript

*The V8 project is still vibrant but innovation is needed for the next level*

# V8: Performance Over Time




# V8: Complexity Over Time



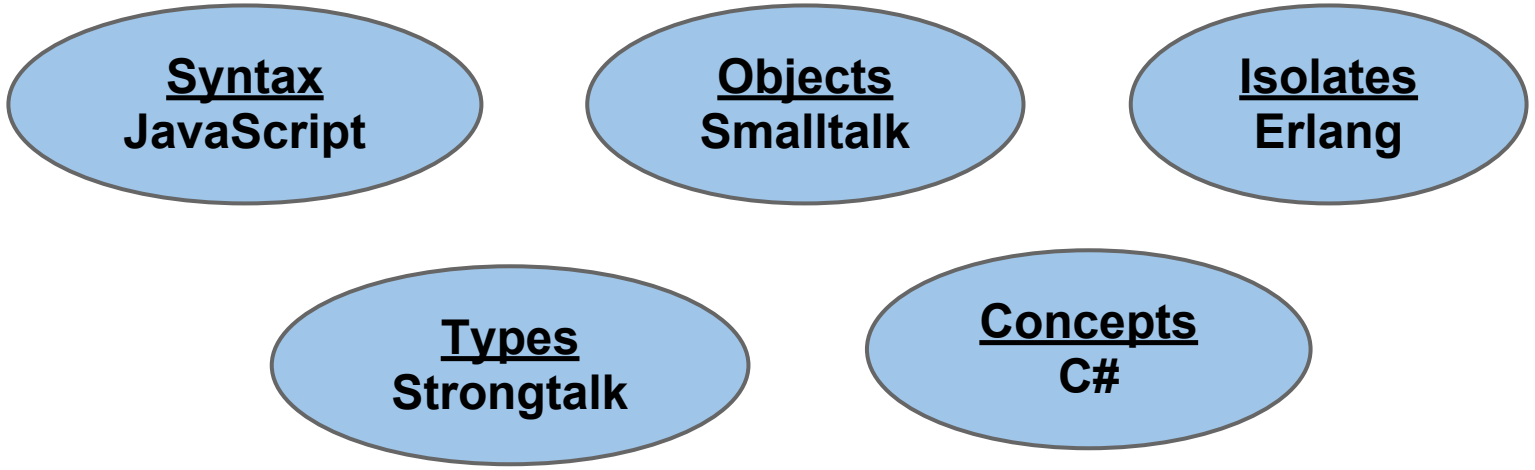

# Welcome to The Dart Side

back to the future

# Dart Programming Language

- Class-based
- Familiar syntax
- Optional types

**Dart =**




# A Taste of Dart

Easy to understand, familiar syntax

Dart

```
import 'dart:html';

main() {
  var button = new ButtonElement();
  button.text = 'Click me!';
  document.body.children.add(button);
}
```

# What Are We Trying to Achieve with Dart?

- More scalable development platform
- Higher performance and faster startup
- Toolable with static type checking
- Useful and consistent libraries

*A translator from Dart to JavaScript makes it run in all modern browsers...*

“The parts fit together and make sense. They are pragmatic and unsurprising. Exactly what I need to be productive. I can’t describe in words how liberating it feels to have a consistent web development experience that makes sense.”

Thomas Schranz

Founder of Blossom

# Why is Dart Already Faster than JavaScript?

- Straightforward language semantics
- Much simpler object model
- Programs are declared, not constructed at runtime
- Fewer special corner cases to worry about

# Example Demonstrating Why Dart Is Fast

Modifying the program structure at runtime is costly

JS

```
function A() {}
A.prototype.foo = function() { print("foo") }

function B() { A.call(this) }
B.prototype = new A()

var b = new B()
b.foo()
```

```
B.prototype.foo = function() { print("new foo") }
b.foo()
```

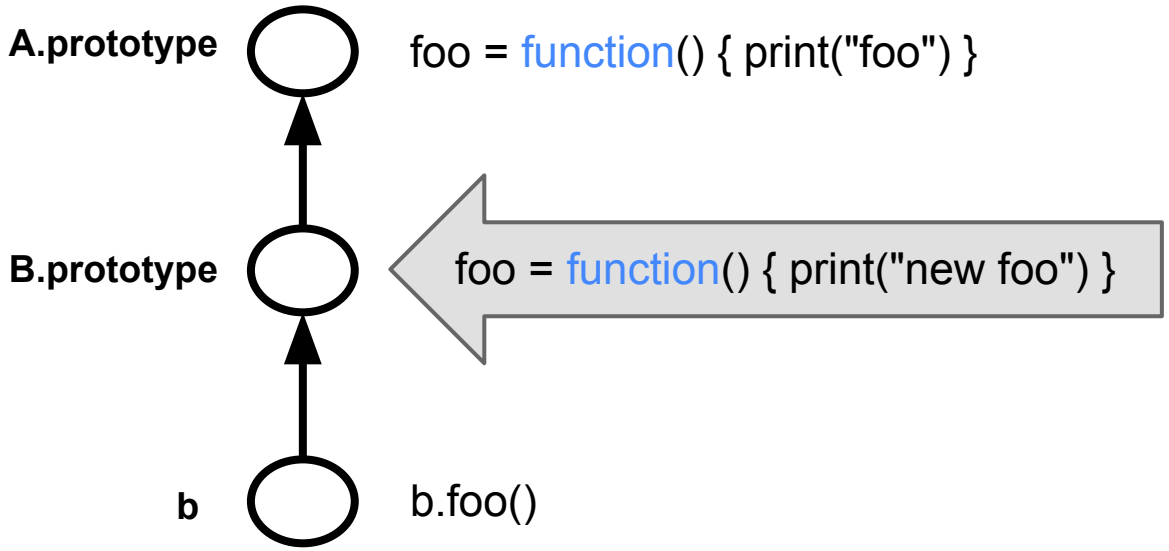
Dart

```
class A {
  foo() => print("foo");
}

class B extends A {}

var b = new B();
main() { b.foo(); }
```

# Detecting Changes in the Prototype Chain




In JavaScript, you have to either validate or keep track of dependencies

*None of this is needed in Dart!*

# Simpler Language Results in Less Generated Code

JS

```
function bench() {
  for (var i = 0; i < 100000; i++) b.foo()
}
```

Dart

```
bench() {
  for (var i = 0; i < 100000; i++) b.foo();
}
```

V8 compiler generates

- 281 bytes of common code +
- 239 bytes of stubs

Dart VM compiler generates:

- 103 bytes of common code +
- 57 bytes of stubs

```
mov edx,(nil)
test esp,0x4
jnz 44
push 0x0
mov ebx,esp
mov edx,0x2
mov ecx,0x3
mov eax,[ebx+0x4]
mov [ebx],eax
add ebx,0x4
dec ecx
jnz 27
mov [ebx],0x12345678
push ebp
mov ebp,esp
push esi
push edi
sub esp,0x18
mov [ebp+0xf4],edx
mov eax,[ebp+0xfc]
mov [ebp+0xe0],eax
mov esi,eax
cmp esp,[0x9cff7f4]
jnc 76
call 0x328131a0
jmp 110
mov eax,[ebp+0xfc]
mov ecx,[ebp+0xe4]
test_b cl,0x1
jnz 284
sar ecx,1
mov ebx,[ebp+0xf0]
mov edx,[ebp+0xec]
xchg eax, ecx0x3282c329
mov ebx,[ebp+0xc]
mov edx,[ebp+0x8]
mov ecx,[ebp+0xe0]
xor eax,eax
test edx,0x1
jz 498
cmp [edx+0xfff],0x44a108c1
jnz 503
mov esi,[0x2c00a6fc]
cmp eax, 0x186a0
jnl 258
cmp esp,[0x9cff7f4]
jc 331
mov esi,[edx+0xb]
test esi,0x1
jnz 353
sar esi,1
add esi,0x1
jo 508
add esi,esi
jo 400
mov [edx+0xb],esi
test esi,0x1
jz 253
lea edi,[edx+0xb]
and esi,0xfff00000
test_b [esi+0xc],0x4
jz 253
mov esi,0xfff00000
and esi,edx
test_b [esi+0xc],0x8
jz 253
call 0x3282bbc0
add eax,0x1
jmp 152
mov eax,0x3d808091
mov edx,[ebp+0xf4]
mov esp,ebp
pop ebp
cmp edx,0x0
jz 281
ret 0xc
ret 0x8
```

# Simpler Language Just Makes Sense

- Easier to make fast
- Less generated code
- Predictable performance
- Better memory utilization

Keep it simple!

# Let's Benchmark This Thing!

**Object-oriented performance predictors**

- Richards
- DeltaBlue

**Used in the past for**

- Self
- Strongtalk
- Hotspot JVM
- CLDC HI
- V8

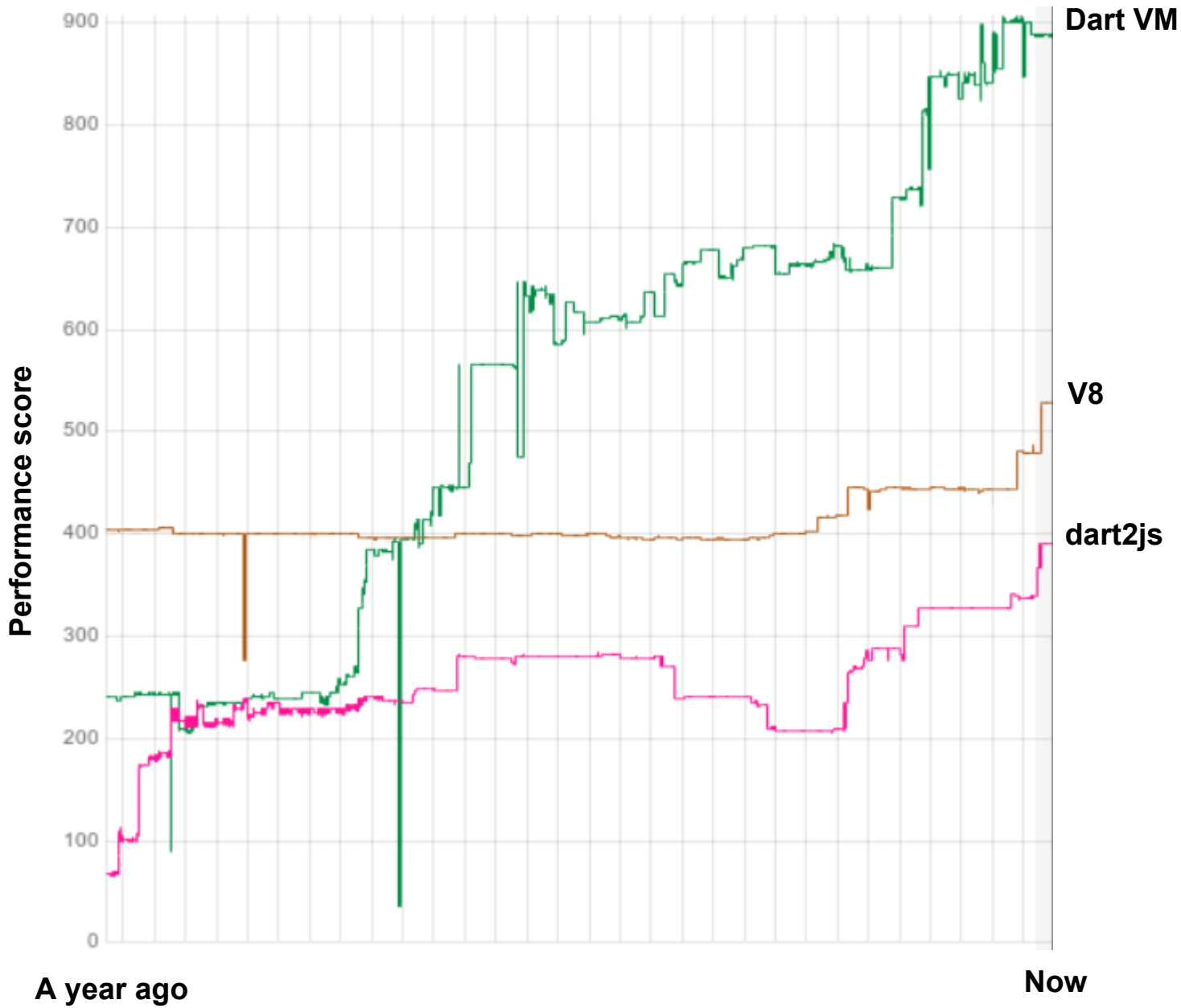
... and now Dart!

# Richards

OS kernel simulation benchmark, originally written in BCPL by Martin Richards.

*V8 has tuned for this benchmark the last six years.*

*Intel Core i5, ia32, Linux*

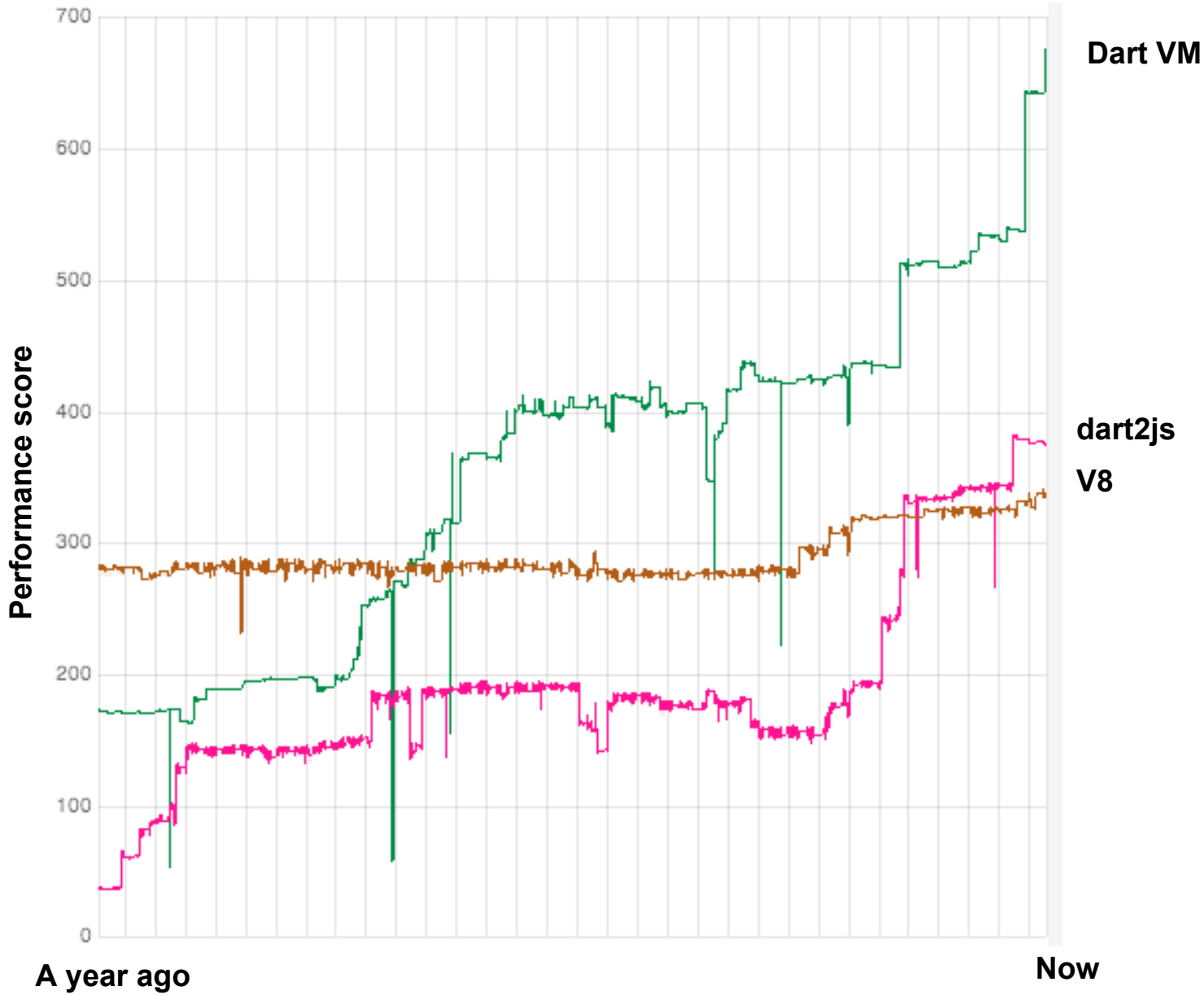



# DeltaBlue

One-way constraint solver, originally written in Smalltalk by John Maloney and Mario Wolczko.

*V8 has tuned for this benchmark the last six years.*

*Intel Core i5, ia32, Linux*

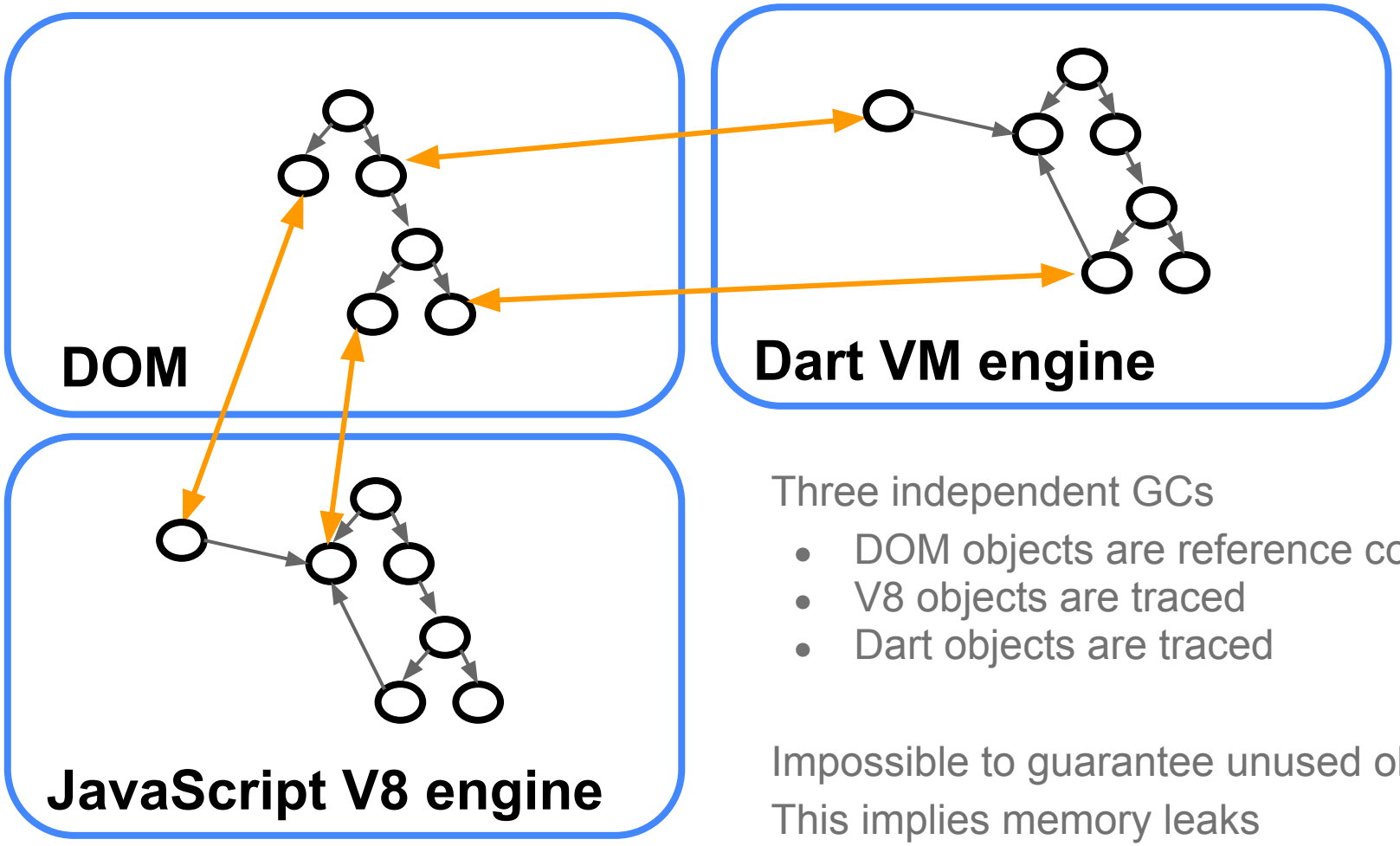


“I've been using Dart for the past few months and have seen my productivity increase.  The Dart platform, along with its excellent tooling, made Glyph3D development a pleasant experience.”

Ali Akbar

Author of Glyph3D

# Including Dart Makes Memory Management Worse




Three independent GCs

- DOM objects are reference counted
- V8 objects are traced
- Dart objects are traced

Impossible to guarantee unused objects are reclaimed

This implies memory leaks

# Oilpan: A Unified Memory Manager for Blink

## Make all objects subject to tracing

- Eliminate reference counting in the DOM
- Eliminate three pointer pairing and DOM grouping
- Unused objects are guaranteed to be reclaimed

## Additional benefits

- Smaller memory footprint
- Entire web application can be serialized
- Concurrent manipulation of the DOM is possible

*Oilpan development just started*

# Accelerating Dart Using SIMD

Modern CPUs support SIMD - Single Instruction Multiple Data

- Dart VM generates code that uses the instructions
- New addition to the web platform!

Where can it be used?

- 3D calculations
- Image processing
- Audio processing

**Demo:** Google Chrome with Dart VM running 3D skeleton animation

# Summary

Performance is always in fashion

- Dart takes web performance to the next level
- Dart on the VM is now faster than JavaScript
- Higher performance means more innovation headroom for web apps

Dart's core platform is stable, and you can start using it today

- Dart works across all modern browsers by compiling to JavaScript
- Dart makes you more productive when working with large apps

Google is committed to Dart!

# More Dart @ Google IO

**What's New in Dart: Your First-class Upgrade to Web Development**

Today, 12:45 PM - 1:25 PM in room 6 (Seth Ladd, Justin Fagnani)

**Dart's DOM of the Future, Today!**

Today, 3:30 PM - 4:10 PM in room 6 (Sigmund Cherem, Emily Fortuna)

**Code Lab: Mobile Web Apps with Dart and Web Components**

Friday, May 17, 9:00 AM - 11:00 AM in room 1 (Andrei Mouravski)

# Questions?

"Few years back GWT was heavily promoted by Google. Now Google is promoting Dart for the future web development. How could I convince my company towards those technology for the longer run? and what are benefits would I get if I choose?"

Thamizharasu, Chennai Share ▼

Flag as inappropriate

---

"Today Google introduced new Android Studio. Is there a plan for releasing something similar for Dart?"

Valeriy, Ukraine Share ▼

Flag as inappropriate

---

"In some benchmarks the DartVM is now outperforming the JVM. Do you you think it will be possible for the DartVM to outperform the JVM for most code? Are there specific areas where the JVM's design will allow it to perform better than the DartVM?"

Greg, Wellingtron Share ▼

Flag as inappropriate

# Thank You!

# References to Research Papers


- E. W. Dijkstra, Leslie Lamport, A. J. Martin, C. S. Scholten, E. F. M. Steffens. 1976. On-the-fly Garbage Collection: An Exercise in Cooperation.
- Dave Ungar. 1984. Generation Scavenging: A Non-disruptive High Performance Storage Reclamation Algorithm.
- Urs Hölzle, Craig Chambers, David Ungar. 1992. Debugging Optimized Code with Dynamic Deoptimization.
- Urs Hölzle. 1994. Adaptive Optimization for Self: Reconciling High Performance with Exploratory Programming.
- Richard E. Jones, Rafael Lins. 1996. Garbage Collection: Algorithms for Automatic Dynamic Memory Management.
- Kentaro Hara. March, 2013. What Percentages of Real-world JavaScript Execution are Charged on What.



Google
Developers