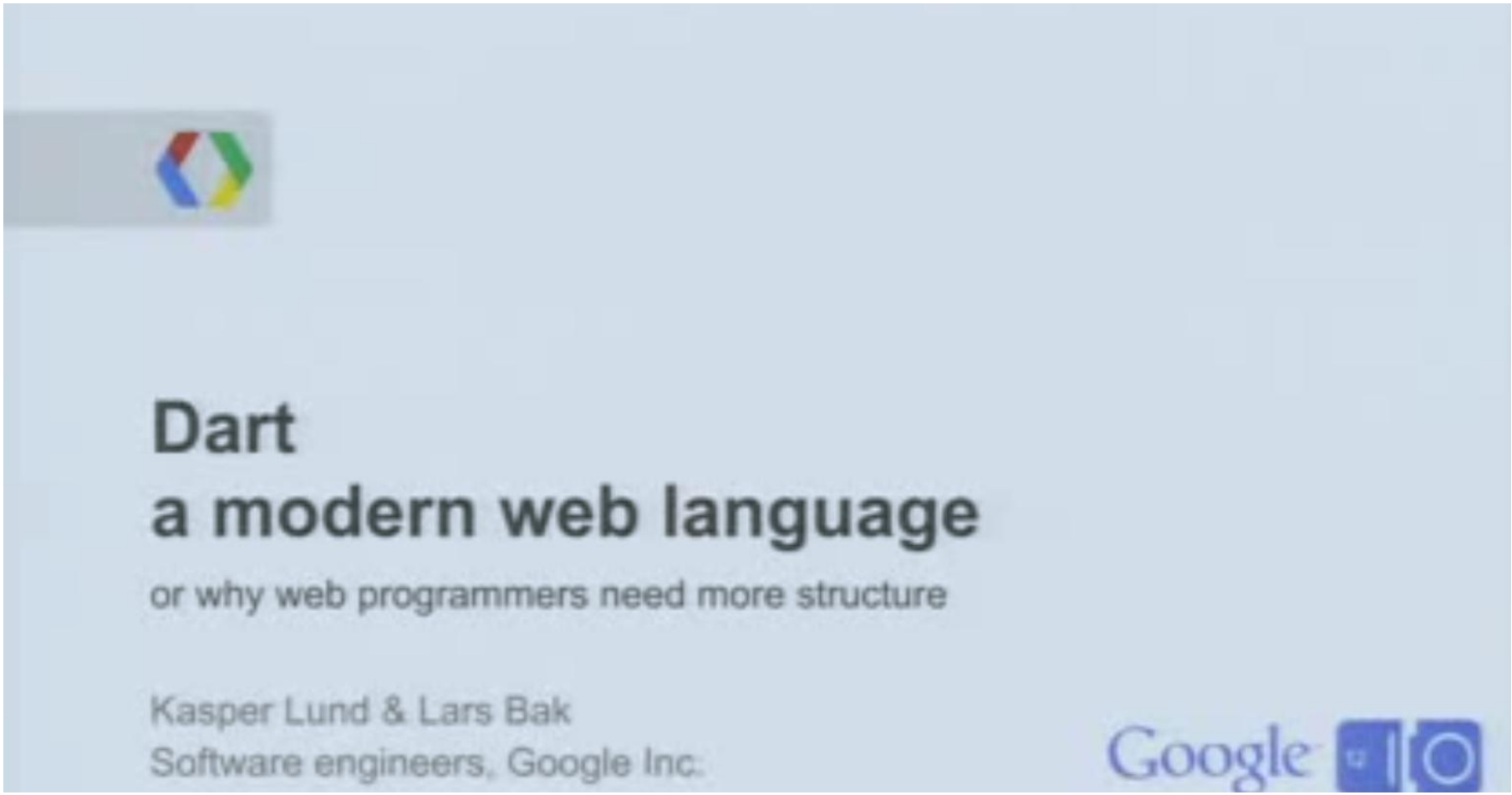

Dart
a modern web language
or why web programmers need more structure
Kasper Lund & Lars Bak
Software engineers, Google Inc.
Google I/O

The two creators of Dart will discuss the rationale behind Dart's design and its impact on web scalability and performance. They'll also present how Dart helps developers innovate by increasing their productivity without breaking backwards compatibility.

About Lars Bak

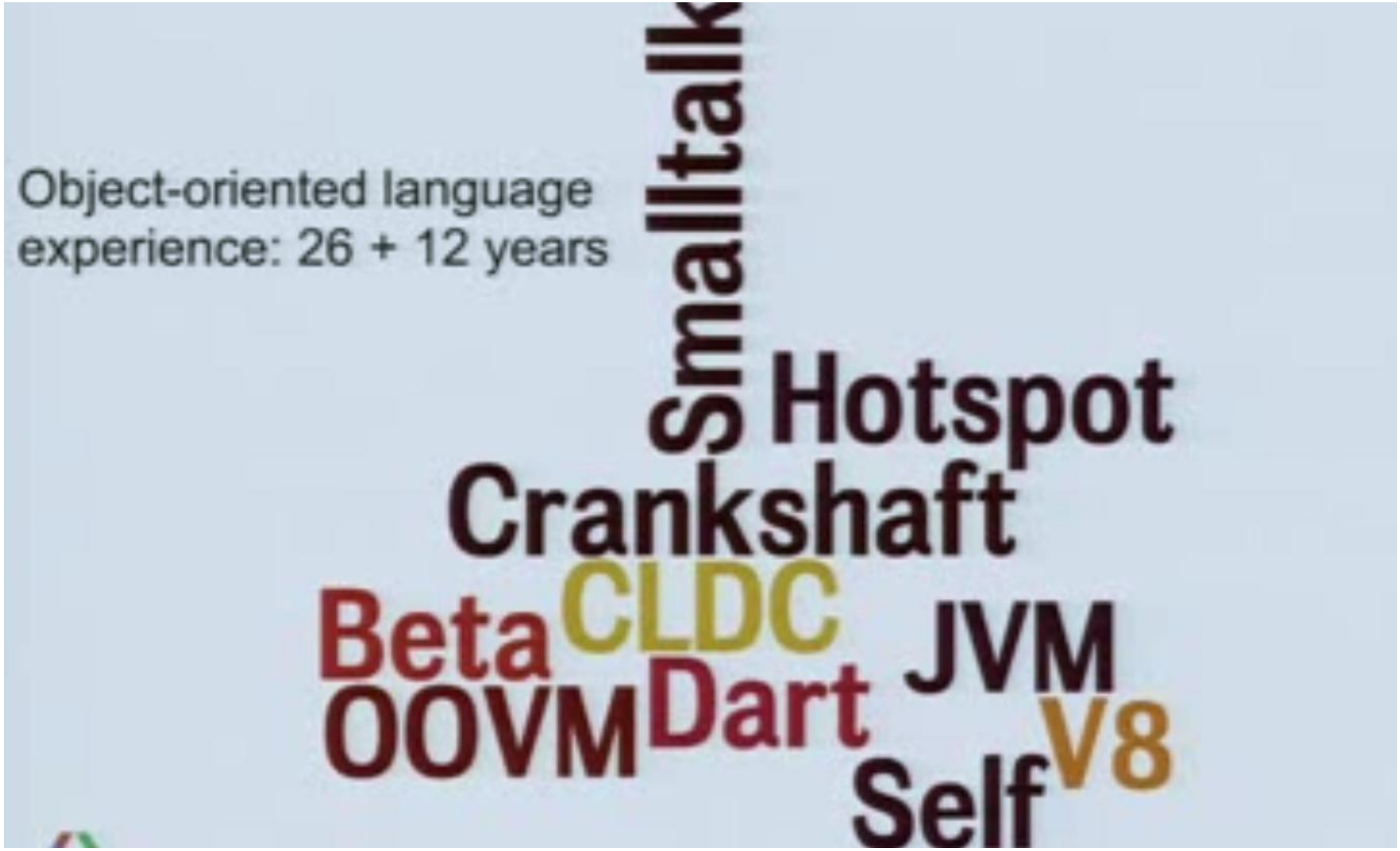
View full profile

Lars is an engineer at Google. After spending 25 years implementing various object-oriented programming languages, he initiated the Dart project with Kasper Lund to improved the state of web programming.

About Kasper Lund

View full profile

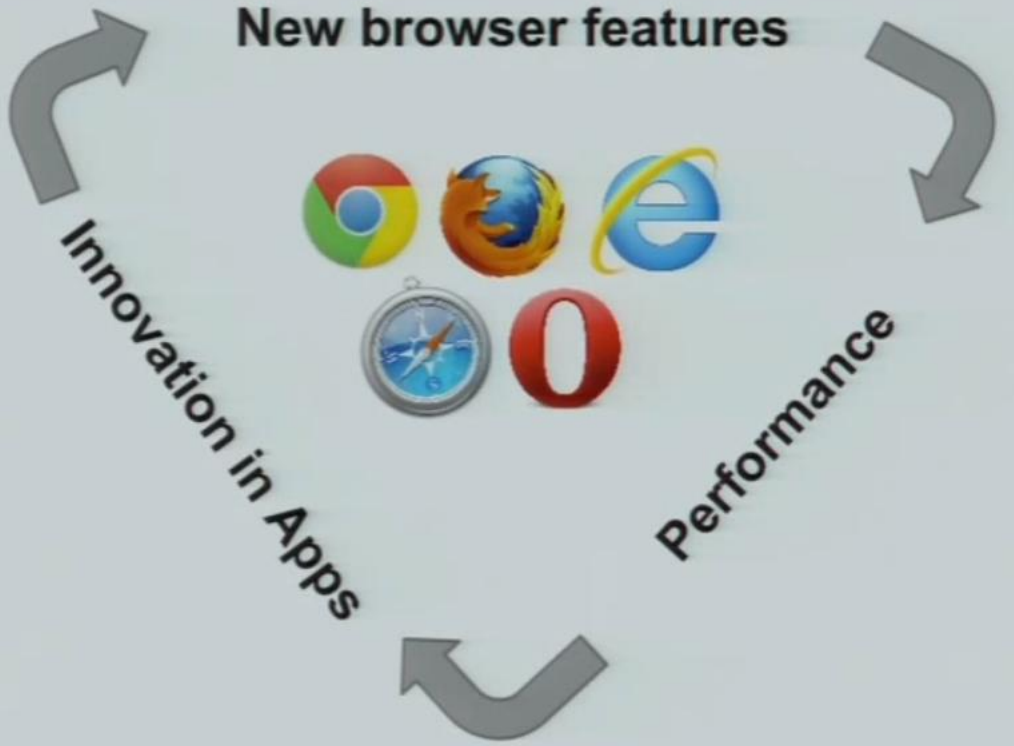
Kasper Lund is a software engineer at Google working on the design and implementation of Dart. Before starting the Dart project, Kasper spent a few years implementing the V8 JavaScript engine.


Object-oriented language
experience: 26 + 12 years
Smalltalk
Hotspot
Crankshaft
CLDC
Beta
JVM
OOVM
Dart
V8
Self

The Web Is Fantastic

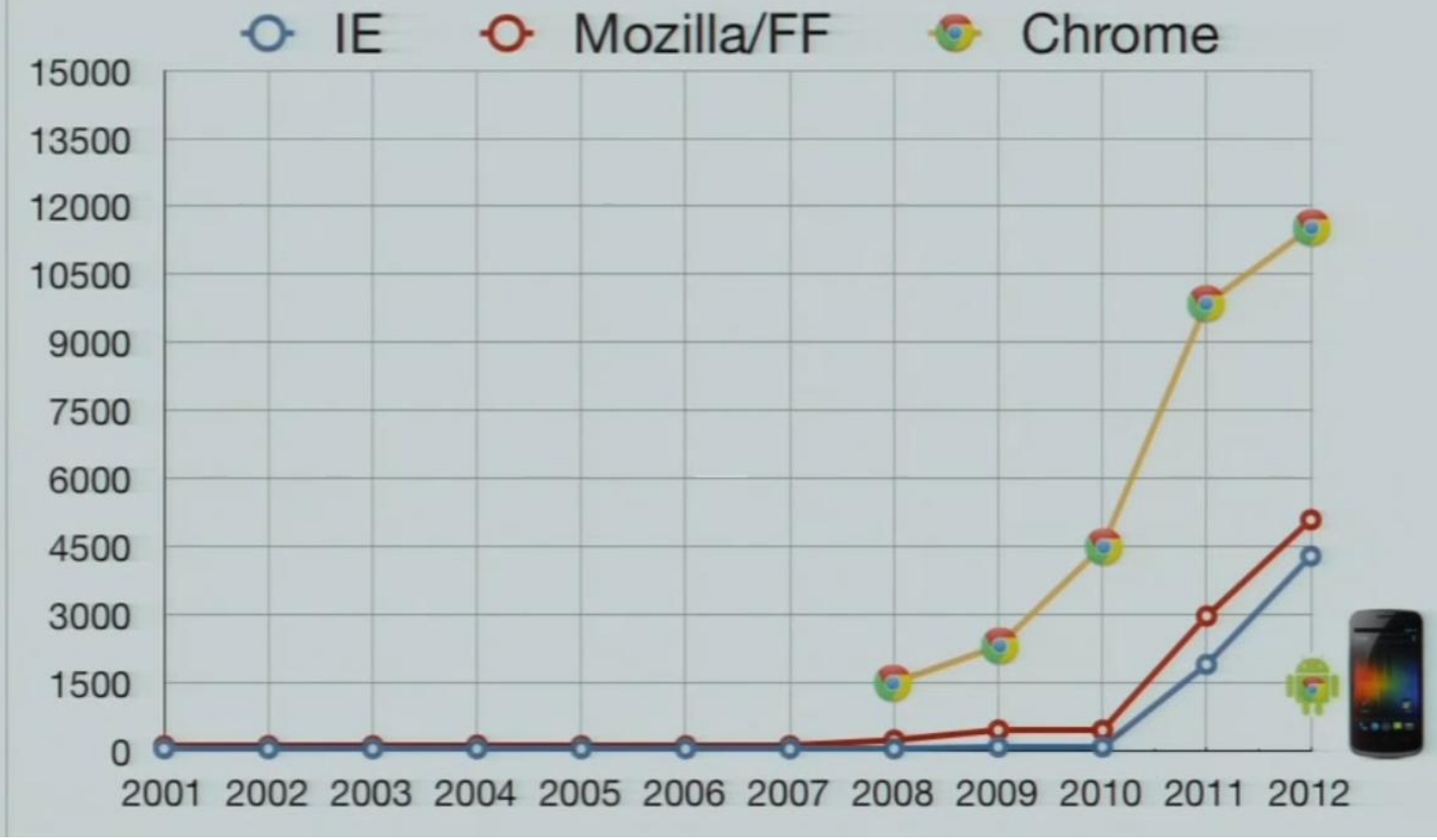
- The web is everywhere
- Developing small applications is easy
- No required installation of applications
- Support for incremental development
- Friendly competition drives the industry

The Web Platform Wheel of Improvements



JavaScript Performance Improvements

V8 Benchmark Suite v3 - higher numbers are better



Challenges for the Web

- Programmer productivity
- Application scalability
- Raw execution speed
- Startup performance

..... if we don't innovate, the web will lose to mobile apps

Welcome to the Dart Platform

A web programming platform that has accepted these challenges

High level goals

- Simple productive programming language
- Support for programming in the large
- Predictable high performance
- Ultra-fast startup
- Compatibility with modern browsers

JavaScript Issue #1: Where Is The Program?

JavaScript code is hard to reason about

- Almost no declarative syntax
- Borderline impossible to find dependencies
- Monkey patching makes it even worse

Understanding the program structure is crucial

- Easier code maintenance and refactoring
- Better debuggability and navigation of code

Where Is The Program?

All declarations are actually statements that must be executed

JavaScript

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

Fields are "declared" implicitly by the statements in the constructor.

Where Is The Program?

It can be pretty hard to analyze what actually gets declared

JavaScript

```
function Point(x, y) {
  if (Object.defineProperty) {
    Object.defineProperty(this, 'x', { value: x, writable: false });
    Object.defineProperty(this, 'y', { value: y, writable: false });
  } else {
    this.x = x;
    this.y = y;
  }
}
```

Control flow makes it difficult to statically tell which fields you end up with

JavaScript Issue #2: Keep On Truckin'

JavaScript has a *keep on truckin'* mentality

- Mistakes are tolerated
- Wrong types lead to unusable results
- Almost anything goes...

Throwing errors eagerly is better

- Easier to locate the source of errors
- Forces more errors during testing
- Gives confidence in deployed apps

Keep On Truckin': Constructors Are Just Functions

Seems nice and simple, but leads to lots of issues

JavaScript

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

var point = Point(2, 3);      // Whoops, forgot to write new but
assert(point == undefined);   // that's okay because we get undefined back
assert(x == 2 && y == 3);     // and now we have more global variables!
```

Keep On Truckin': Accessing Non-existing Properties

Typos lead to code that runs but does that wrong thing

JavaScript

```
var request = new XMLHttpRequest();
...
request.onreadystatechange = function() {
  if (request.readystate == 4) {
    console.log('Request done: ' + request.responseText);
  }
};
```

Did you mean:
readyState?

Keep On Truckin': Implicit Conversions

Mixing objects and immutable values

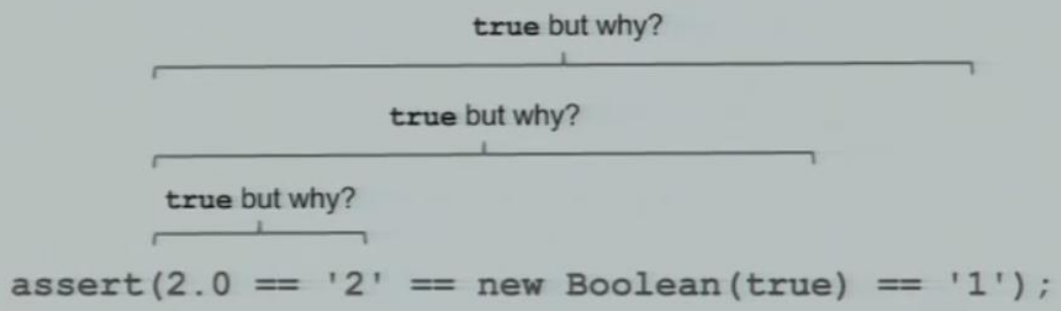
JavaScript

```
var string = 'the quick brown fox';    // strings are values, not objects
string.hash = md5(string);             // string -> object
assert(string.hash == undefined);      // string -> object (a different one)
```

Keep On Truckin': Implicit Conversions

This just gets worse and worse

JavaScript



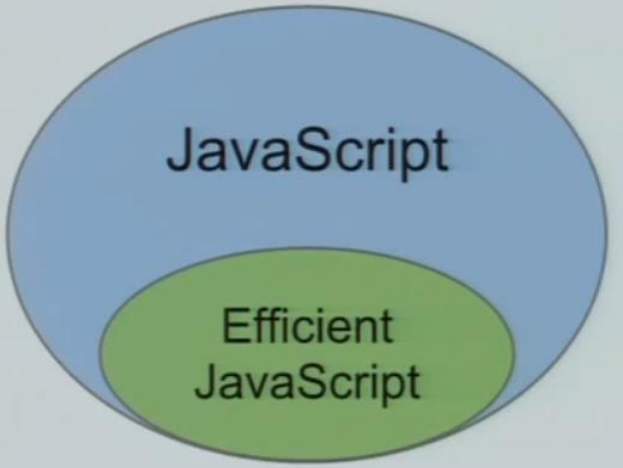

```
assert(2.0 == '2' == new Boolean(true) == '1');
```

Enough implicit conversions to make your head explode?

JavaScript Issue #3: Unpredictable Performance

Advice: Use the efficient subset of JavaScript

- VMs have been optimized for certain patterns
- Benefit from the performance improvements



Harder than it sounds

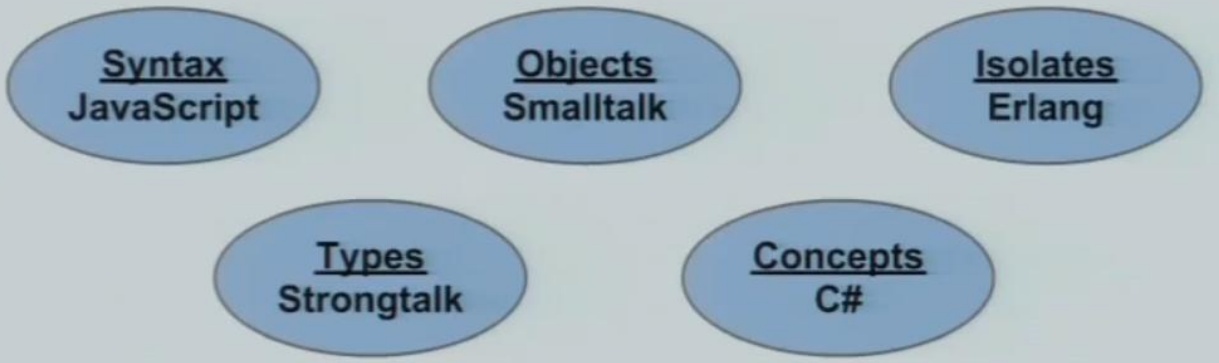
- Lots of differences between browsers
- Not just about syntactic constructs

JavaScript performance is very unpredictable

Building high-performance web apps in JavaScript is tricky

Dart in a Nutshell

- Unsurprising simple object-oriented programming language
- Class-based single inheritance with interfaces
- Familiar syntax with proper lexical scoping
- Single-threaded with isolate-based concurrency
- Optional static types



First Dart Example

Point example that computes distance to origin

Dart

```
class Point {
  var x, y;
  Point(this.x, this.y);
  operator +(other) => new Point(x + other.x, y + other.y);
  toString() => "($x,$y)";
}

main() {
  var p = new Point(2, 3);
  print(p + new Point(4, 5));
}
```

Optional Static Types

Static type annotations convey the intent of the programmer

- They act as checkable documentation for code and interfaces
- They can be generic which makes them very useful for collections
- They have no effect on runtime semantics

The Dart type system is considered **unsound** since downcasts are allowed
However, these downcasts can be validated at runtime

First Dart Example

Now with static types

Dart

```
class Point {
  num x, y;
  Point(this.x, this.y);
  Point operator +(Point other) => new Point(x + other.x, y + other.y);
  String toString() => "($x,$y)";
}

main() {
  Point p = new Point(2, 3);
  print(p + new Point(4, 5));
}
```

Dart Has Covariant Generic Types

An apple is a fruit, so a list of apples is clearly a list of fruits?

Dart

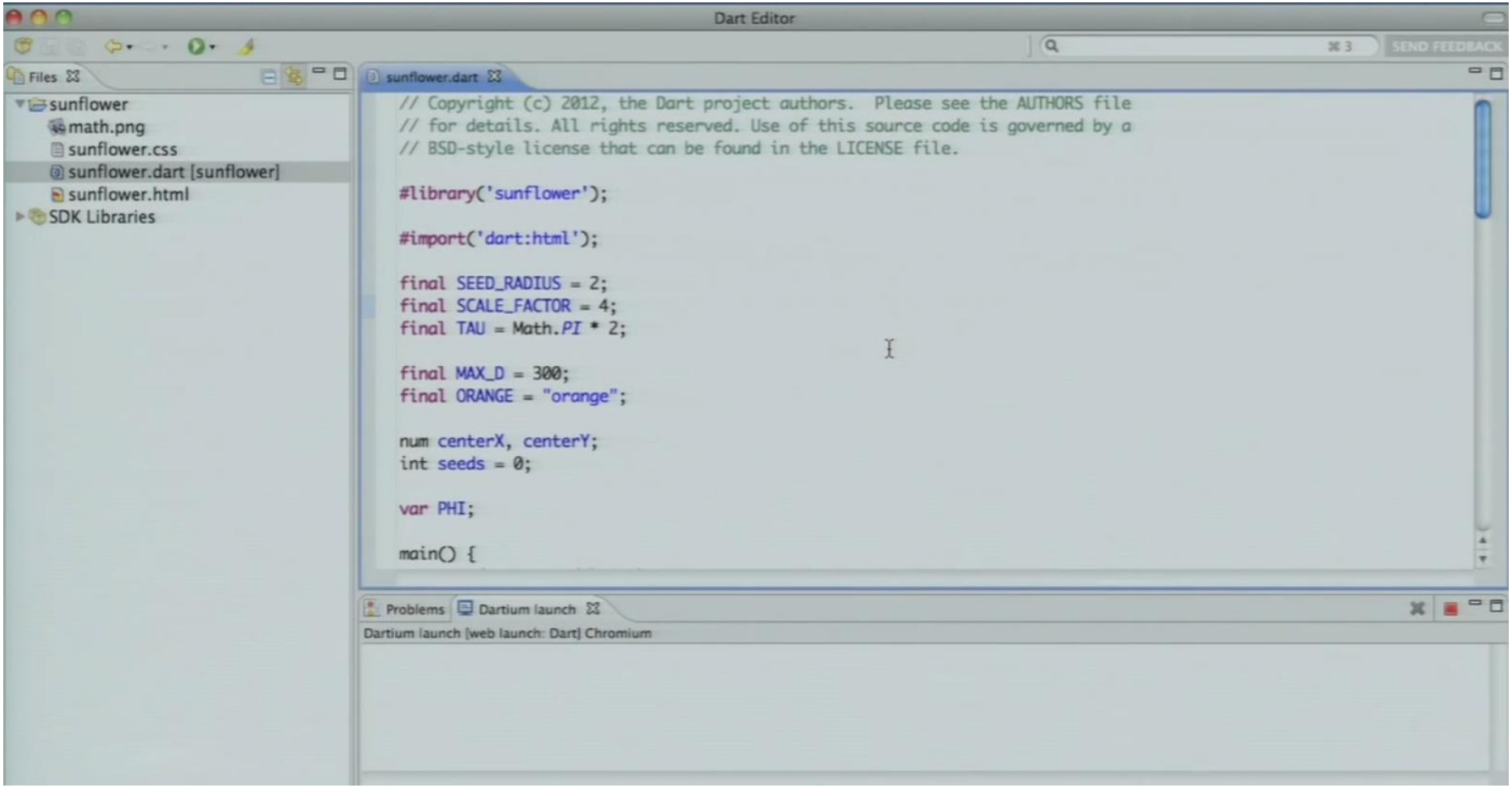
```
main() {
  List<Apple> apples = tree.pickApples();
  printFruits(apples);
}

void printFruits(List<Fruit> fruits) {
  for (Fruit each in fruits) print(each);
}
```

Demo: Runtime Type Checking In Dart

DART

It's demo time...


Dart Editor
SEND FEEDBACK
Files
sunflower
math.png
sunflower.css
sunflower.dart [sunflower]
sunflower.html
SDK Libraries
sunflower.dart
// Copyright (c) 2012, the Dart project authors. Please see the AUTHORS file
// for details. All rights reserved. Use of this source code is governed by a
// BSD-style license that can be found in the LICENSE file.
#library('sunflower');
#import('dart:html');
final SEED_RADIUS = 2;
final SCALE_FACTOR = 4;
final TAU = Math.PI * 2;
final MAX_D = 300;
final ORANGE = "orange";
num centerX, centerY;
int seeds = 0;
var PHI;
main() {
Problems
Dartium launch
Dartium launch [web launch: Dart] Chromium

Sunflower Demo

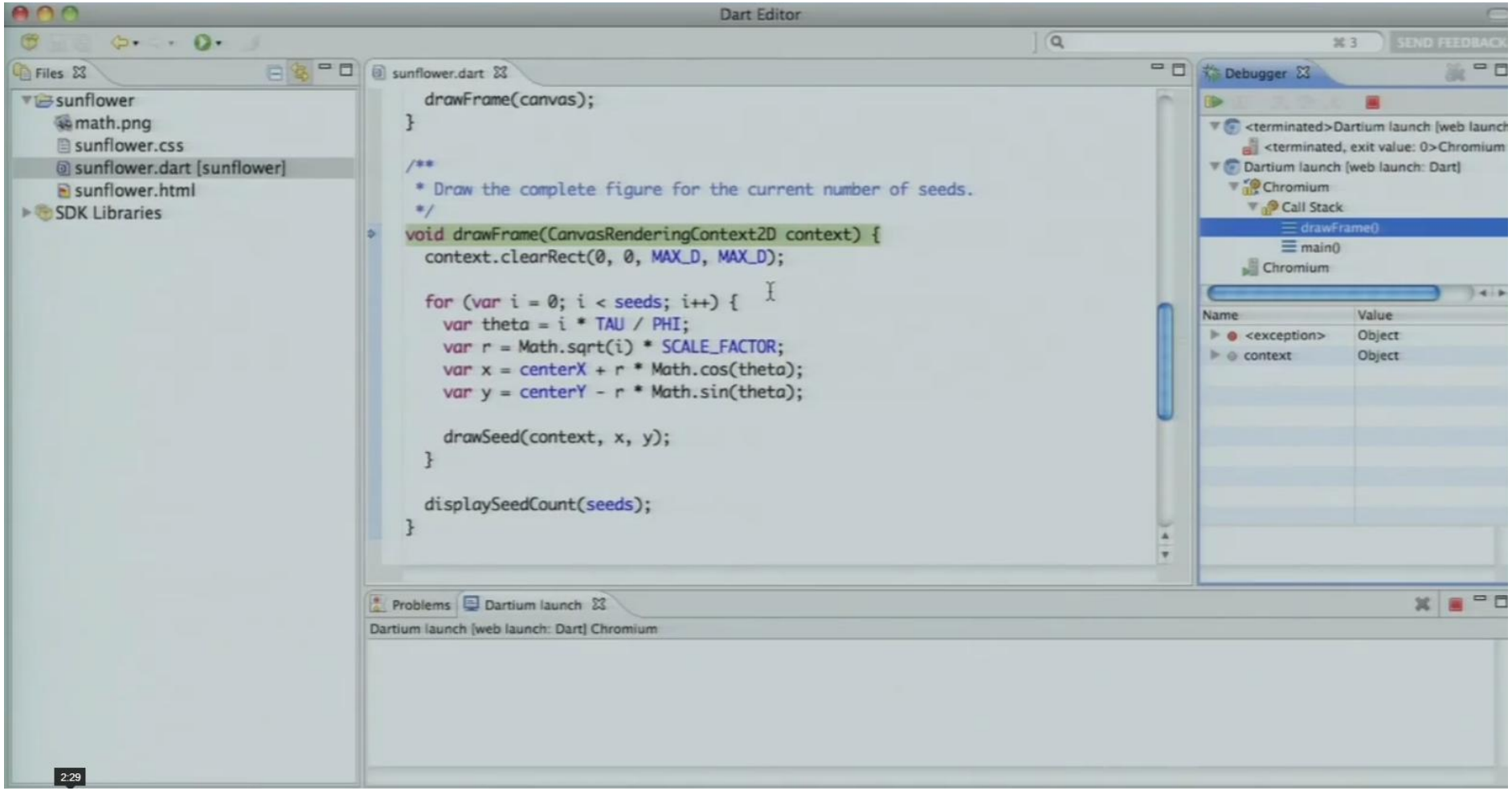
127.0.0.1:3030/Users/bak/editor/samples/sunflower/sunflower.html

drfibonacci's Sunflower Spectacular

A canvas 2D demo.

$$r(n) = \sqrt{n} \quad \theta(n) = \frac{2\Pi n}{\Phi} \quad \Phi = \frac{1+\sqrt{5}}{2}$$

667 seeds


Dart Editor
Files
sunflower
math.png
sunflower.css
sunflower.dart [sunflower]
sunflower.html
SDK Libraries
sunflower.dart
drawFrame(canvas);
}
/**
* Draw the complete figure for the current number of seeds.
*/
void drawFrame(CanvasRenderingContext2D context) {
context.clearRect(0, 0, MAX_D, MAX_D);
for (var i = 0; i < seeds; i++) {
var theta = i * TAU / PHI;
var r = Math.sqrt(i) * SCALE_FACTOR;
var x = centerX + r * Math.cos(theta);
var y = centerY - r * Math.sin(theta);
drawSeed(context, x, y);
}
displaySeedCount(seeds);
}
Debugger
<terminated>Dartium launch [web launch
<terminated, exit value: 0>Chromium
Dartium launch [web launch: Dart]
Chromium
Call Stack
drawFrame()
main()
Chromium
Name
Value
<exception>
Object
context
Object
Problems
Dartium launch
Dartium launch [web launch: Dart] Chromium
2:29

Potpourri of Cool Dart Language Features

Named Constructors

No overloading based on argument types

Dart

```
class Point {
  var x, y;
  Point(this.x, this.y);
  Point.polar(r, t) : x = r * cos(t), y = r * sin(t);
}

main() {
  var p = new Point(2, 3);
  var q = new Point.polar(3, 0.21);
}
```

Interfaces With Default Implementations

Instances can be constructed from interfaces

Dart

```
interface List<E> implements Collection<E> default ListFactory<E> {
  List([int length]);
  ...
}

main() {
  var untyped = new List();          // any object can be added
  var typed = new List<Point>(12);   // only points can be added
}
```

Cascaded Calls

Multiple calls to the same object

Dart

```
void drawCircle(CanvasElement canvas, int x, int y, int size) {
  canvas.context..beginPath()
                ..arc(x, y, size, 0, Math.PI * 2, false)
                ..fill()
                ..closePath()
                ..stroke();
}
```

Proper Capturing of Loop Variables

Prints 01234567 rather than 88888888 as in JavaScript

Dart

```
main() {
  var closures = [];
  for (var i = 0; i < 8; i++) closures.add(() => i);  // collect
  for (var c in closures) print(c());                 // evaluate
}
```

For each iteration of the loop, a fresh copy of the variable `i` is captured.

Dart Is Beautiful

Dart programs are declared and easy to read in the source code

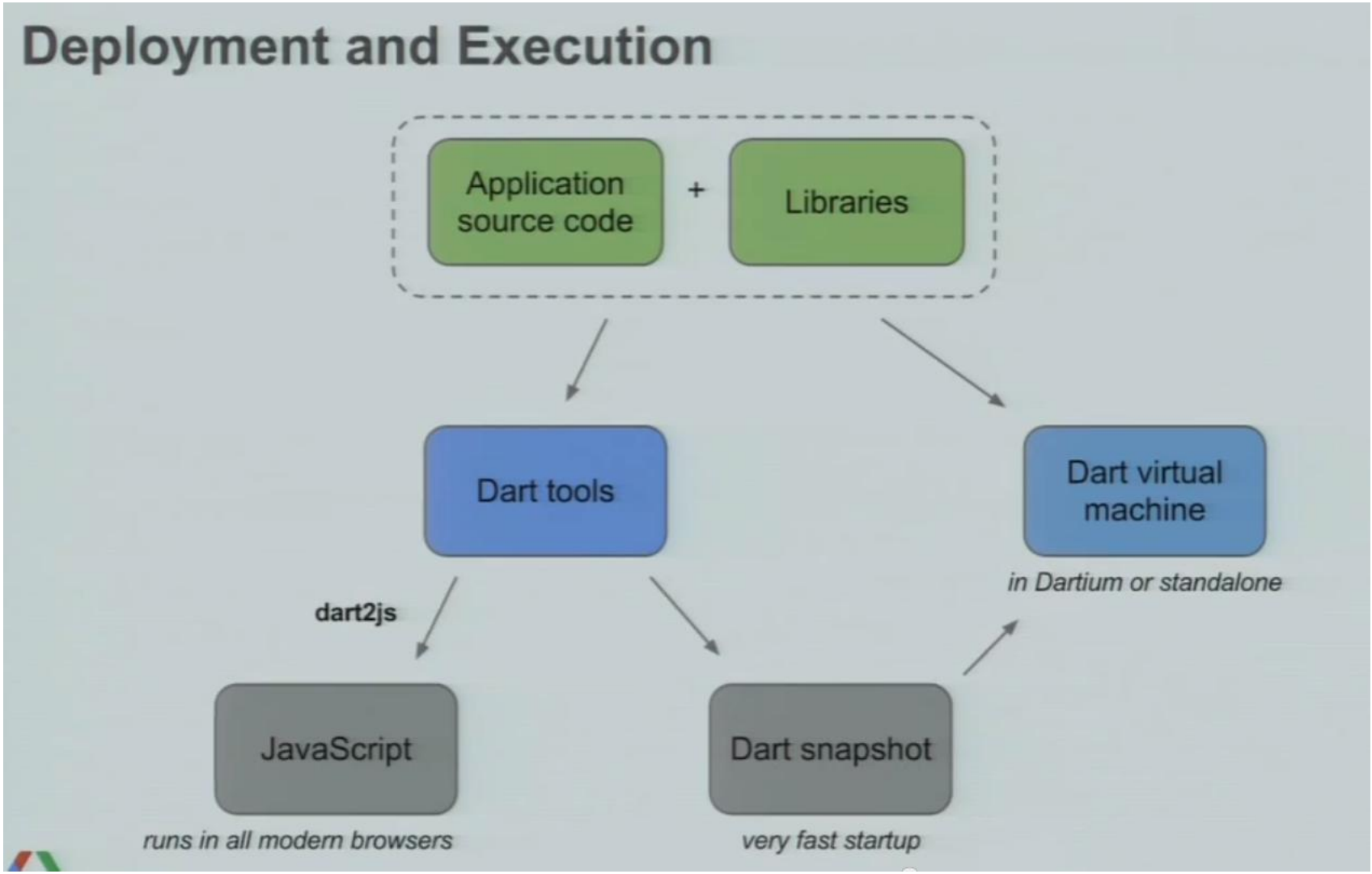
Clean semantics without surprises

- No strange implicit conversions are performed
- Libraries cannot be monkey patched at runtime

Java, C#, and JavaScript programmers:

You can be productive in Dart within a few hours

Deployment and Execution



Dart Is Compatible With Modern Web Browsers

http://buildbot.dartlang.org/

Translation To JavaScript

There is a compiler that translates Dart to JavaScript (**dart2js**)
The entire compiler is implemented in Dart

Dart

```
class Point {
  var x, y;
  Point(this.x, this.y);
  toString() => "($x,$y)";
}
```

JavaScript

```
Point = {
  super: "Object",
  fields: ["x", "y"],
  toString$0: function() {
    return '(' + str(this.x) +
           ',' + str(this.y) +
           ')';
  }
};
```

Dart Tree Shaking

Tree shaking is a pre-deployment step that eliminates unused code

- Unused classes and methods are eliminated
- Applications are only penalized for what they use
- Easy process since Dart apps are declared

Tree shaking is used by **dart2js**

- Reduces the download size
- Improves startup times

Dart Snapshots

Binary form of an entire Dart app

- Speeds up loading by **10x**

Created by loading the app and serializing the heap

- Snapshots contain classes and methods
- Only works with the Dart VM

Benefits

- No need to scan and parse source code
- Crucial on slower mobile devices

The Future of the Language

The Dart language is frozen for the first version

Except these few remaining issues:

- Eliminate interfaces and rely on abstract classes
- Introduce mixins for better sharing
- Add support for annotations

The Dart Timeline

Dart Developer Release in Fall 2012

We are serious about making Dart the next generation web app platform!

The fall release will include:

- Language specification
- Libraries for web and server development
- Programming environment
- Virtual machine + browser integration
- Translator to JavaScript

Dart Is Open Source

Dart is available under a BSD license

Developed in the open (code reviews, build bots, etc.)

Excited and active community

Online resources

- **Primary site** - http://www.dartlang.org/
- **Code** - http://dart.googlecode.com/
- **Libraries** - http://api.dartlang.org/
- **Specification** - http://www.dartlang.org/docs/spec/

Please try out Dart

Dart Summary

Dart is an unsurprising object-oriented language, instantly familiar to most
Dart allows you to write code that tools and programmers can reason about
Dart applications work in all modern browsers by translating to JavaScript

Making fast virtual machines is easy ... try making a new language

Thank You!

That is it folks...

kasperl@google.com & bak@google.com