DART

Dart Today and Beyond

Gilad Bracha
Google

1: Overview

- Dart: Civilized Programming for the Web

1: Overview

- Dart: Civilized Programming for the Web
  - Class based

```
class Point {
  var x, y;
  Point(this.x, this.y);
  operator +(a) => new Point(x + a.x, y + a.y);
}
```

1: Overview

- Dart: Civilized Programming for the Web
  - Class based, Purely Object Oriented

Everything is an Object

- As in Smalltalk, Ruby, Scala and others
- No autoboxing, no hidden coercions

1: Overview

- Dart: Civilized Programming for the Web
- Class based, Purely Object Oriented, Optionally Typed

```
class Point {
  int x, y;
  Point(this.x, this.y);
  Point operator +(Point a) =>
      new Point(x + a.x, y + a.y);
}
```

1: Overview

- Dart: Civilized Programming for the Web
- Class based, Purely Object Oriented, Optionally Typed, Single Inheritance

1: Overview

- Dart: Civilized Programming for the Web
- Class based, Purely Object Oriented, Optionally Typed, **Mixin-based** Inheritance

1: Overview

- Dart: Civilized Programming for the Web
- Class based, Purely Object Oriented, Optionally Typed, Mixin-based Inheritance

1: Overview

- Dart: Civilized Programming for the Web
- Class based, Purely Object Oriented, Optionally Typed, Mixin-based Inheritance, Message-passing Concurrency

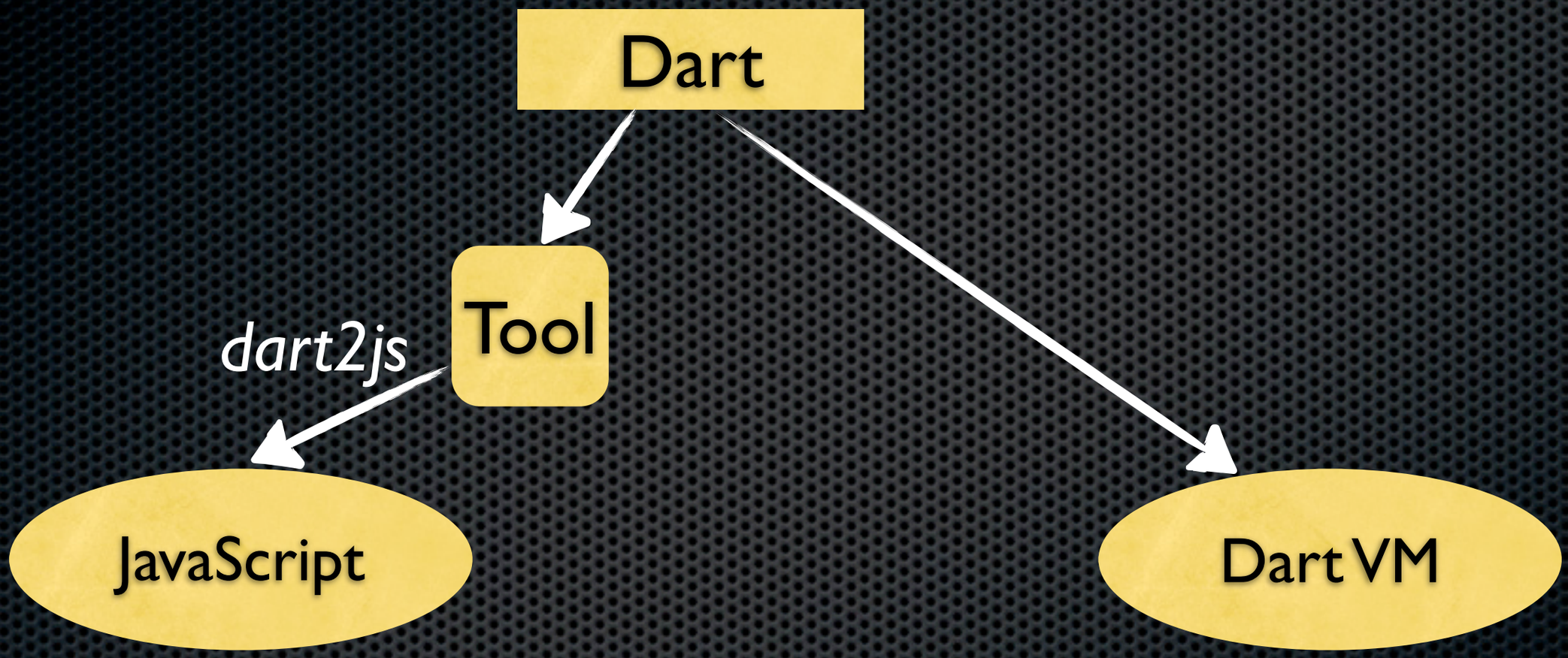
1: Overview

- Dart: Civilized Programming for the Web
- Class based, Purely Object Oriented, Optionally Typed, Mixin-based Inheritance, Message-passing Concurrency, Mirror-based Reflection

1: Overview

- Dart compiles to JS on the client
  - Dart VM provides better performance and functionality
  - For development
  - On the Server
  - At some point, in the client as well

Dart Execution



2: Mixins

- Planned for a future Milestone
- Nothing is finalized, let alone implemented

What is a Mixin?

```
class Point {
  var x, y;
  Point(this.x, this.y);
  operator +(a) => new Point(x + a.x, y + a.y);
}
```

A Mixin is the Delta between a Class and its Superclass

```
class Point {
    var x, y;
    Point(this.x, this.y);
    operator +(a) => new Point(x + a.x, y + a.y);
}
```

A Useful Mixin

```
class Collection<E> {

  abstract Collection<E> newInstance();

  Collection<E> map(f) (
      var result = newInstance();

      this.forEach((e){result.add(f(e))});

      return result;
    }
  }
```

A Useful Mixin

```
class Collection<E> {

  abstract Collection<E> newInstance();

  Collection<E> map(f) (
      var result = newInstance();

      this.forEach((e){result.add(f(e))});

      return result;
    }
  }
```

Using a Mixin

```
class DOMElementList<E> mixin Collection<E>
extends DOMList {
  DOMElementList<E> newInstance() {
    return new DOMElementList<E>;
  }
}
```

Using a Mixin (again)

```
class DOMElementSet<E> mixin Collection<E>
extends DOMList {
  DOMElementSet<E> newInstance() {
    return new DOMElementSet<E>;
  }
}
```

Using a Mixin (again)

30 times

Fine Print

- Initial version will restrict the use of fields
  - Possibly no fields allowed
  - Or perhaps no constructor bodies/initializers
- May disallow super calls, non-trivial superclasses

More on Mixins

- Originated in LISP
- OOPSLA 90 paper: bracha.org/oopsla90.pdf
- Implemented in Strongtalk VM
- Restricted version popularized as traits
- Variants found in Ruby, Scala, Newspeak

3: Isolates & Asynchrony

- No Shared State Concurrency
- Programs are built out of *Isolates*
- Isolates are Actor-like: Separate Heap and Stack, Communicate via Messages over *Ports*

Futures

- Sending a Message returns a *Future*
- You can set callbacks on futures
  - Very similar to node.js

Speculating on the Future

- Use of callbacks can get tedious
- Nesting makes code hard to read

```
Future<List<int>> xAxis = db1.fetchData(100);
xAxis.handleException((e) { doSomething();});

Future<List<int>> yAxis = db2.fetchData(100);
yAxis.handleException((e) { doSomething();});
```

```
xAxis.then(
  (xs) {
    yAxis.then(
      (ys) {
        plot(xs, ys);
      })
  });
```

Speculating on the Future

- Use of callbacks can get tedious
- Nesting makes code hard to read
- Can we do better?

- C# style await?
- Async blocks (aka future comprehensions)?
- Promise pipelining?

```
Future<List<int>> xAxis = db1.fetchData(100);
Future<List<int>> yAxis = db2.fetchData(100);
async {
   List<int> xs <- xAxis;
   List<int> ys <- yAxis;
   plot(xs, ys);
} catch (e) {doSomething()};
```

Beyond Ports

- Ports are rather low level
  - Lots of plumbing, serialization
  - Loss of type information
- Can we do better?

From Ports to Objects

```
class IsolateWrapper {
  SendPort _sendPort;
  IsolateWrapper(this._sendPort);
  noSuchMethod(name, args){
    return _sendPort.call([name, args]);
  }
}
```

4: Mirrors

- Mirror-based Reflection
  - Originated in Self
  - Used in Strongtalk, Java (JDI & APT), Newspeak, Scala
  - Now in Dart
  - Caveat Emptor: WIP! Unavailable on dart2js!

Classic OO Reflection

o.getClass().getMethods();

Mirrors are Different

Mirrors are objects that reflect *other* objects.

If you don’t have the right mirror, you cannot reflect, addressing difficulties in deployment, distribution, security

Dart's Mirrors

- In Dart, one isolate can reflect on another
- API is necessarily asynchronous in places
- We seek to minimize asynchrony

Mirror Demo

Beyond Introspection

- Mirror builders for creating new/modified code
- Stack mirrors and Activation mirrors to support debugging

Caveats

- Web apps often optimized for size by eliminating symbols (minification) and unused code (tree shaking)
- Reflecting on code that has been optimized away, or whose name has been minified, is not possible
- Options:
  - ~~Do not optimize? Ouch.~~
  - Provide mechanism to selectively preserve reflective information

More on Mirrors

- Blog:
  - gbracha.blogspot.com/2010/03/through-looking-glass-darkly.html
- OOPSLA 2004 paper: bracha.org/mirrors.pdf
- 2010 Video:
- www.hpi.uni-potsdam.de/hirschfeld/events/past/media/100105_Bracha_2010_LinguisticReflectionViaMirrors_HPI.mp4

Dart

- Class based, Purely Object Oriented, Optionally Typed, Mixin-based Inheritance, Mirror-based Reflection, Message-passing Concurrency
- Supports Software Engineering, Good Performance, Toolability
- Status: Open Source (Apache), Open Development, M1 Released Today

- http://www.dartlang.org