

Dart

Gilad Bracha for the Dart Team

Part I:

Overview

Why Dart?

We want to improve the state of the art of Web Programming

Web Programming

Web Programming has huge Advantages:

1. Zero-install
2. Automatic update
3. Ubiquity

Web Programming

Web Programming has huge Advantages:

1. Zero-install
2. Automatic update
3. Ubiquity

But ...

Web Programming

Web Programming has huge Advantages:

1. Zero-install
2. Automatic update
3. Ubiquity

But ...

Web Programming

Web Programming has huge Advantages:

1. Zero-install
2. Automatic update
3. Ubiquity

But only works well for
small programs

Web Programming

Web Programming has huge Disadvantages:

- Lack of program structure
 - Unpredictable performance
 - Infinite Install (every run is an install) => slow startup
 - Poor tooling
 - Very poorly suited for software engineering
- Lack of standard libraries -
 - collections
 - MVC
 - Low level communication between client and server

Very hard to create large applications

Web Programming

If you want to appreciate how much fun it is to build large web applications, I recommend:

- wtfjs.com or, for even more fun, watch [Wat](#)
- More seriously, read [Steve Souders](#) who will tell you how the masters do it, e.g., put code in comments to control when it loads

Enter Dart

A new language and platform for web programming

The Goal

Web applications should be competitive with native applications, in terms of:

- Functionality
- Performance
- Ease of Development

The Goal

Web applications should be competitive with native applications, in terms of:

- Functionality
- Performance
- Ease of Development

For example, [swarm](#)

Constraints

Instantly familiar to the mainstream programmer

Efficiently compile to Javascript

Dart in a Nutshell

Purely Object-Oriented, Class-based,
Single Inheritance

Dart in a Nutshell

Purely Object-Oriented, Class-based,
Single Inheritance with Optional Typing,
Mirror-based Reflection and Actor-
based Concurrency

Dart in a Nutshell

Purely Object-Oriented, Class-based,
Single Inheritance with **Optional**
Typing, Mirror-based Reflection and
Actor-based Concurrency

Roadmap

Two paths for executing Dart:

dart2js - Compile Dart to Javascript; should perform competitively with handwritten JS. Runnable in any modern browser.

VM: Higher performance, access to native platform on server side. Embeddable, V8 style

Eclipse based Dart Editor for development

What's it like?

You can try out dartboard at try.dartlang.org

Part II: Optional Types

Mandatory Types

Static type System regarded as mandatory

Maltyped program are illegal

Mandatory Types: Cons

Expressiveness curtailed

Imposes workflow

Brittleness

A History of Non-Mandatory Types

Common Lisp

Scheme (soft typing)

Cecil

Erlang

Strongtalk

BabyJ

Gradual Typing

A History of Non-Mandatory Types

Common Lisp

Scheme (soft typing)

Cecil

Erlang

[Strongtalk](#)

BabyJ

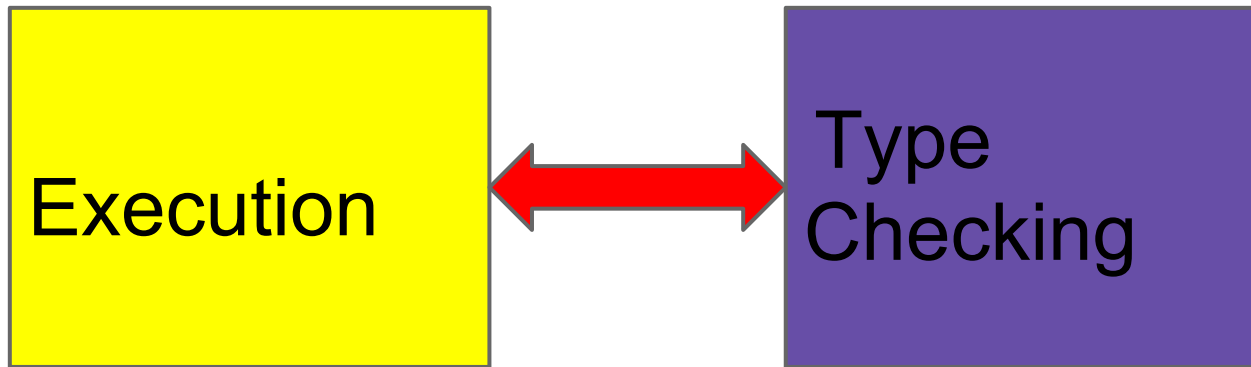
Gradual Typing

Optional Types

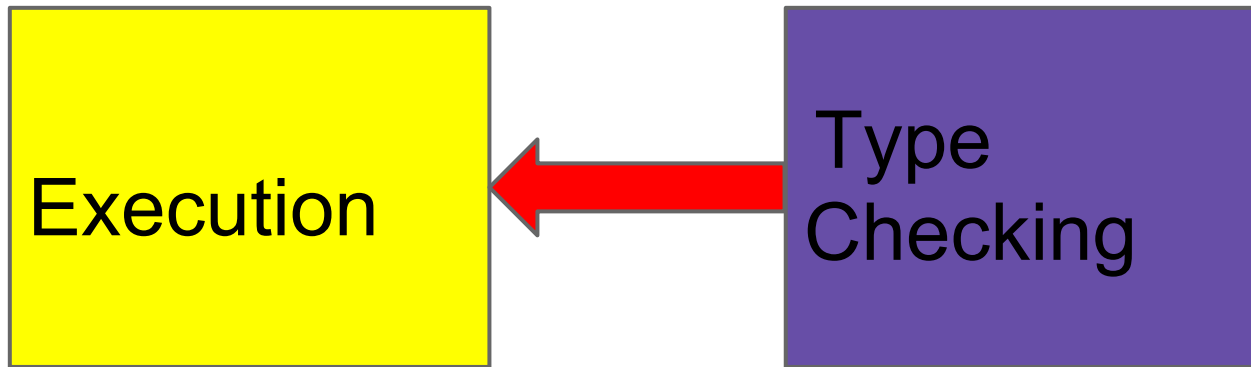
Syntactically optional

Do not affect run-time semantics

Execution depends on Type Checking



Execution independent of Type Checking



Optional Types in Dart

Syntactically optional

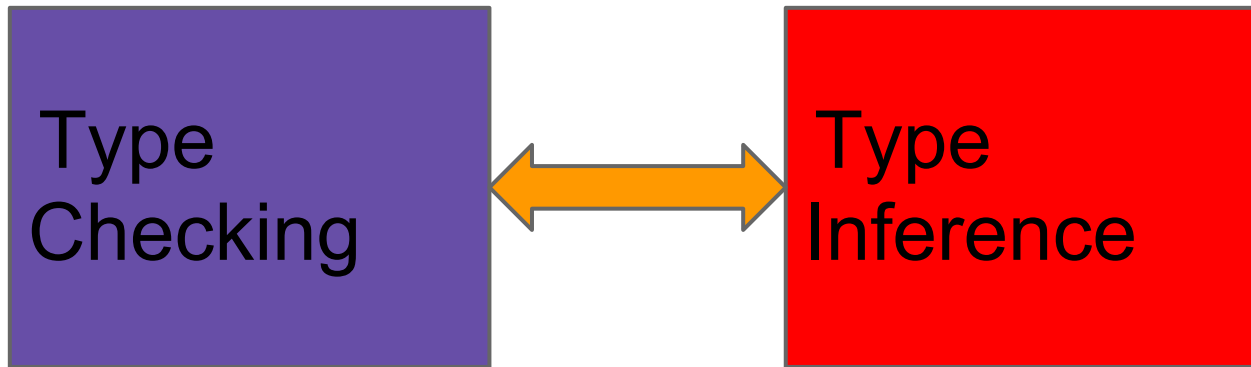
Do not affect run-time semantics

What about Type Inference

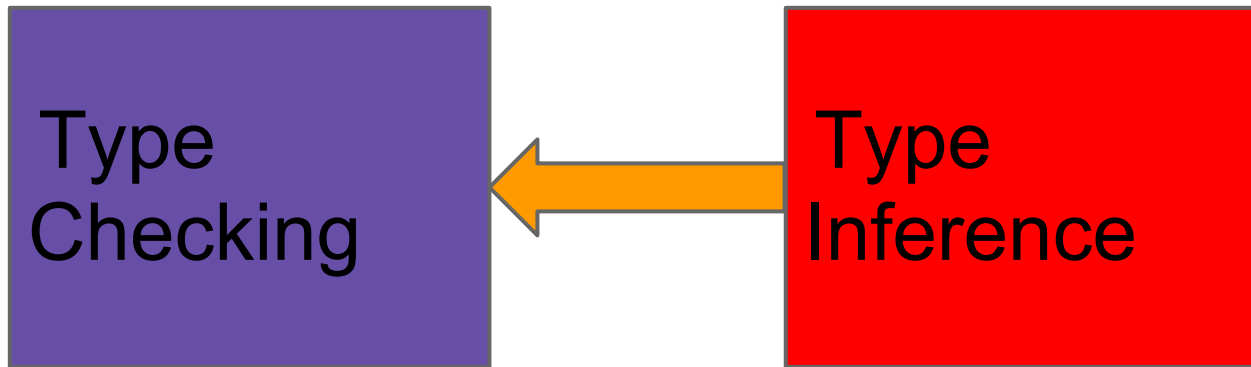
Type Inference relates to Type Checking as
Type Checking relates to Execution.

Type Inference best left to tools

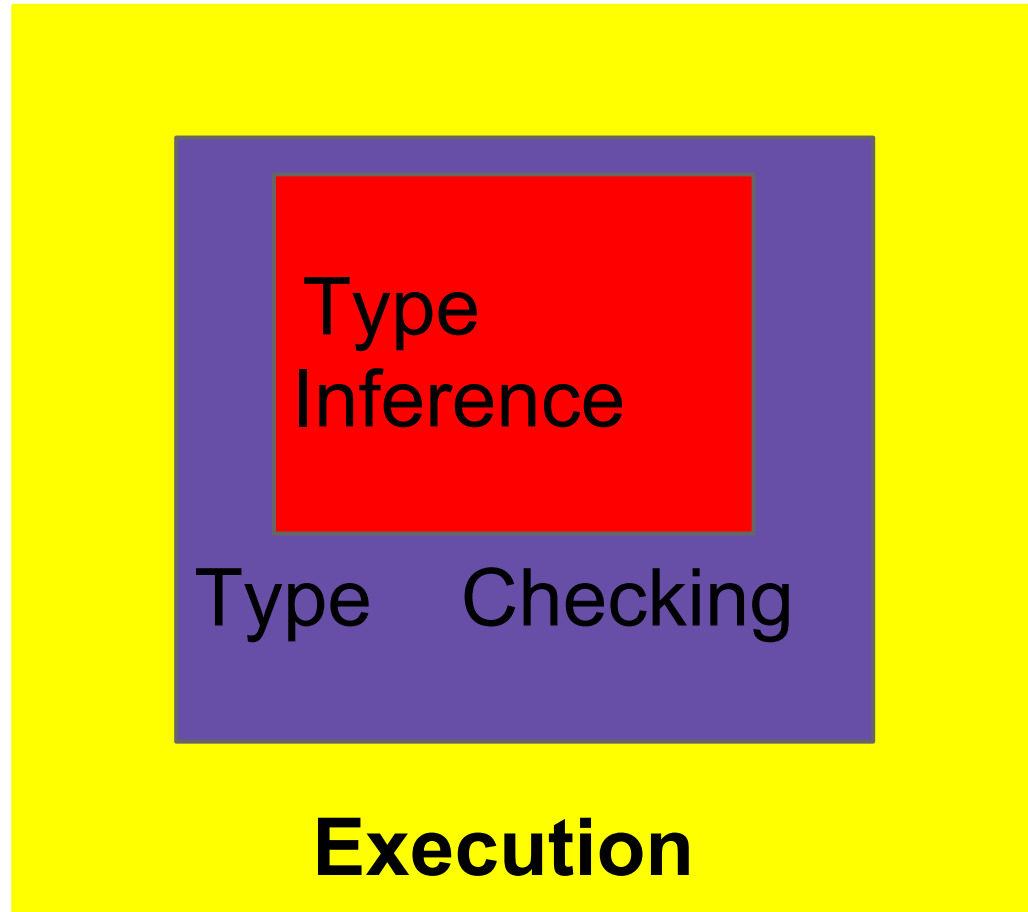
Type Checking depends on Type Inference



Type Checking independent of Type Inference



Don't Get Boxed In!



Mandatory Types: Pros

In order of importance:

- Machine-checkable documentation
- Types provide conceptual framework
- Early error detection
- Performance advantages

Optional Types

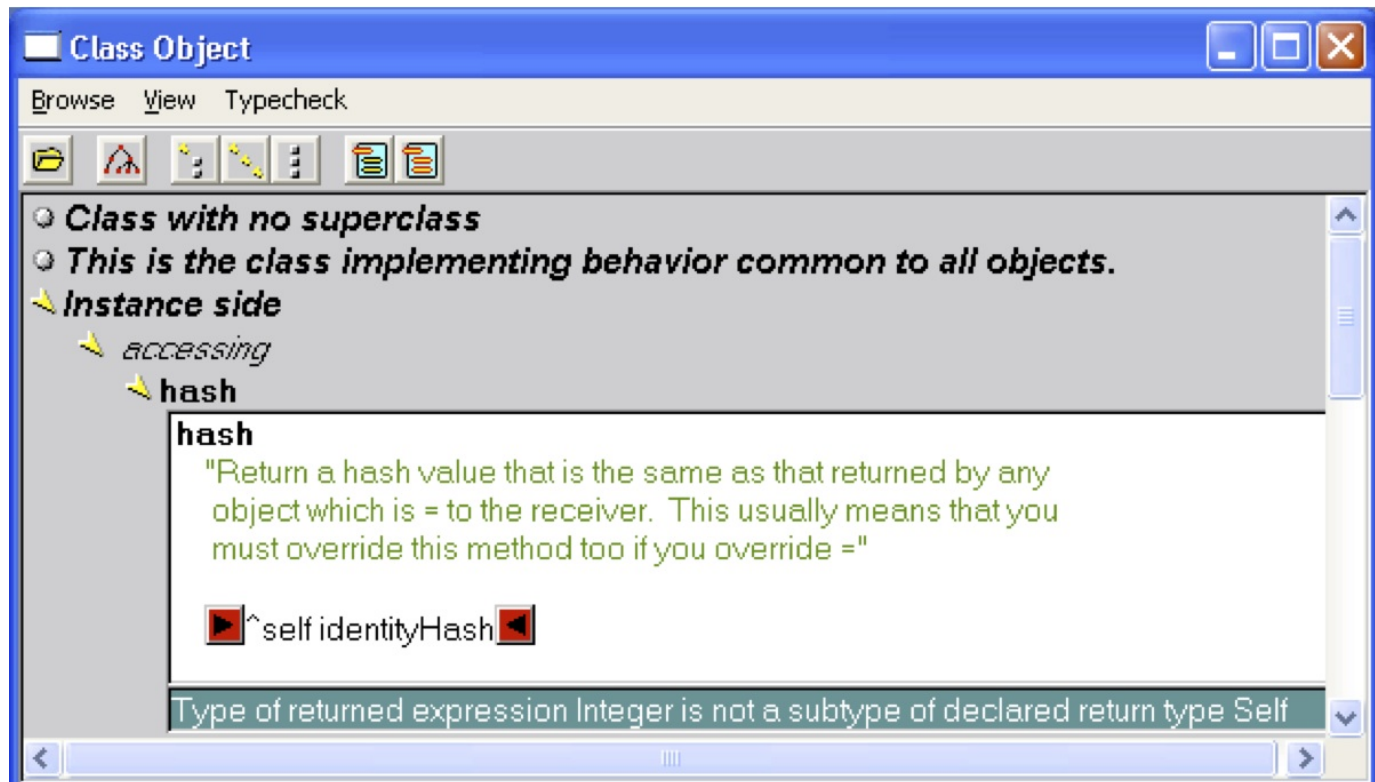
Syntactically optional

Do not affect run-time semantics

Optional Types in Dart

So, what's actually new?

Didn't we have all this in Strongtalk in 1993?




Type Assertion Support

Dart's optional types are best thought of as a type assertion mechanism, **not** a static type system

Dart types at Runtime

During development one can choose to validate types

- $T x = o;$  ***assert(o === null || o is T)***
- By default, type annotations have no effect and no cost
- Code runs free

Not your Grandfather's Type System

Not a type system at all;

Rather, a static analysis tool based on heuristics, coupled to a type assertion system

What about a Real, Sound Type System?

There is no privileged type system, but pluggable types are possible.

For example, one can write a tool that interprets existing type annotations statically

Part III:

Mirrors

Mirrors in Dart

Mirror-based Reflection

- Originated in [Self](#)
- Used in Strongtalk, Java (JDI & APT), [Newspeak](#)
- Caveat Emptor: WIP!

Mirrors Overview

Classic OO reflection:

```
o.getClass().getMethods();
```

Mirrors are separate objects that reflect other objects.

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

Mirrors Overview

If you want to know more, check out:

[My blog](#)

[OOPSLA 2004 paper](#)

[2010 video](#)

A Look at Dart's Mirrors

In Dart, one isolate can reflect on another.

API is necessarily asynchronous in places.

We seek to minimize asynchrony

Synchronous Mirrors

```
ObjectMirror om = ... ;
```

```
ClassMirror cm = om.getClass();
```

```
for (var p in  
        cm.fields.getKeys().map((f) => [f, om.invoke  
        (f, [], {})])  
) print('${p[0]}: ${p[1]}');
```

Synchronous Mirrors

```
ObjectMirror om = ... ;
```

```
ClassMirror cm = om.getClass();
```

```
for (va Map<String, VariableMirror>)  
    cm.fields.getKeys().map((f) => [f, om.invoke  
(f, [], {})]  
) print('${p[0]}: ${p[1]}');
```

Synchronous Mirrors

```
ObjectMirror om = ... ;
```

```
ClassMirror cm = om.getClass();
```

```
for (var p in  
    cm.fields.getKeys().map((f) => [f, om.invoke  
(f, [], {})]  
) print('${p[0]}: ${p[1]}');
```

ObjectMirror

Asynchronous Mirrors

```
ObjectMirror om = ...;
```

```
Future<ClassMirror> cmf = om.getClass();  
cmf.then(  
  (ClassMirror cm) {  
    for (var p in  
      cm.fields.getKeys().map((f) => [f, om.invoke  
(f, [], {})]  
    ) p[1].then( (v) => print('${p[0]}: $v'));  
  })
```


Asynchronous Mirrors

```
ObjectMirror om = ...;
```

```
Future<ClassMirror> cmf = om.getClass();
```

```
cmf.then(
```

```
  (ClassMirror cm) {
```

```
    for (var p in
```

```
      cm.fields.getKeys().map((f) => [f, om.invoke  
(f, [], {})])
```

```
    ) p[1].then( (v) => print('${p[0]}: $v'));
```

```
  }
```

```
)
```

Future<ObjectMirror>

Asynchronous Mirrors

```
ObjectMirror om = ...;
```

```
ClassMirror cm = om.getClass();
```

```
for (var p in
```

```
    cm.fields.getKeys().map((f) => [f, om.invoke  
(f, [], {})])
```

```
    ) p[1].then( (v) => print('${p[0]}: $v');
```

Reflection and Minification

Web apps often optimized for size by eliminating unused code and symbols

Reflecting on code that has been optimized away, or whose name has been minified, is not possible.

Options:

- Do not minify if program uses reflection
- Provide mechanism to selectively protect against minification

Reflection and Minification

Web apps often optimized for size by eliminating unused code and symbols

Reflecting on code that has been optimized away, or whose name has been minified, is not possible.

Options:

- ~~Do not minify if program uses reflection~~
- Provide mechanism to selectively protect against minification

Summary

Dart is a new language for web programming

- Deployable to any modern web browser via compilation to Javascript
- Designed for large projects: emphasis on performance and software engineering
- Open source (BSD)
- Currently a **technology preview** - **everything is subject to change**

Check it out at dartlang.org