Maximiliano Firtman
@firt firt.dev

Introduction to Flutter with Dart

firtman.github.io/intro-flutter

Frontend Masters

mobile+web developer & trainer

Let’s Start!

What we'll cover

- Basics of Dart and Flutter
- Android Studio
- Testing and Debugging
- User Interface
- Screen Navigation
- Working with Data
- Web Services
- Deploying Apps

Pre-requisites

firtman.github.io/intro-flutter

Questions?

Frontend Map

Native Clients

Web / PWA

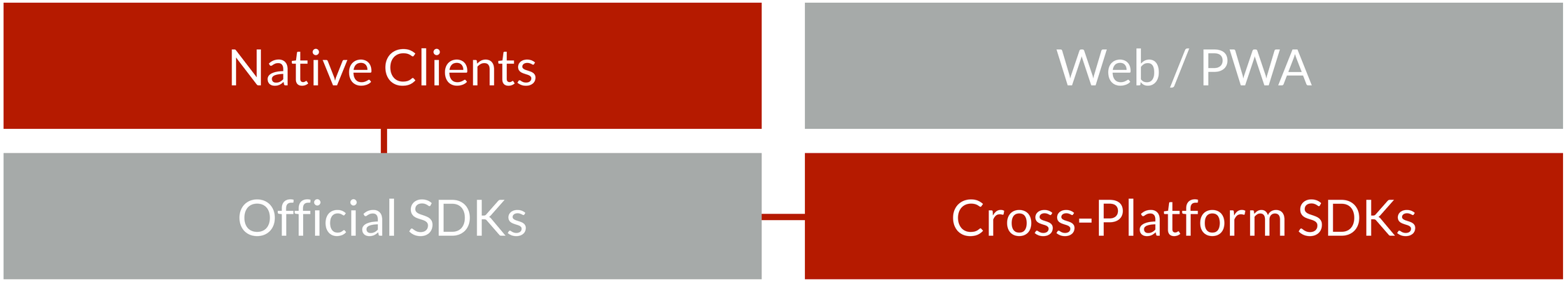
Frontend Map

Native Clients

Web / PWA

Frontend Map

Frontend Map



Frontend Map

Native Clients

Web / PWA

Official SDKs

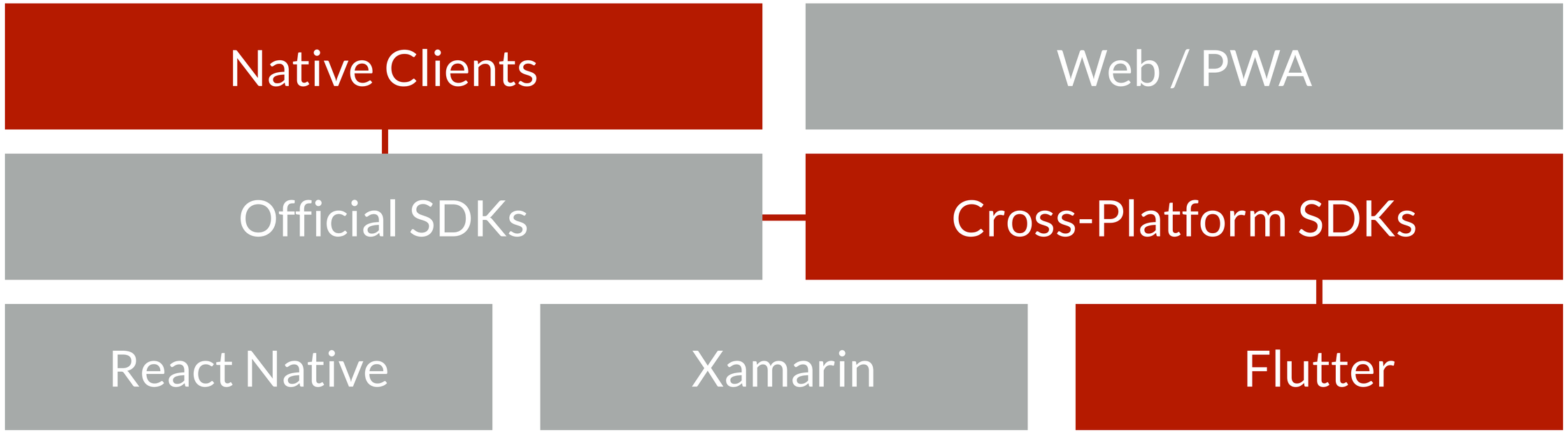
Cross-Platform SDKs

React Native

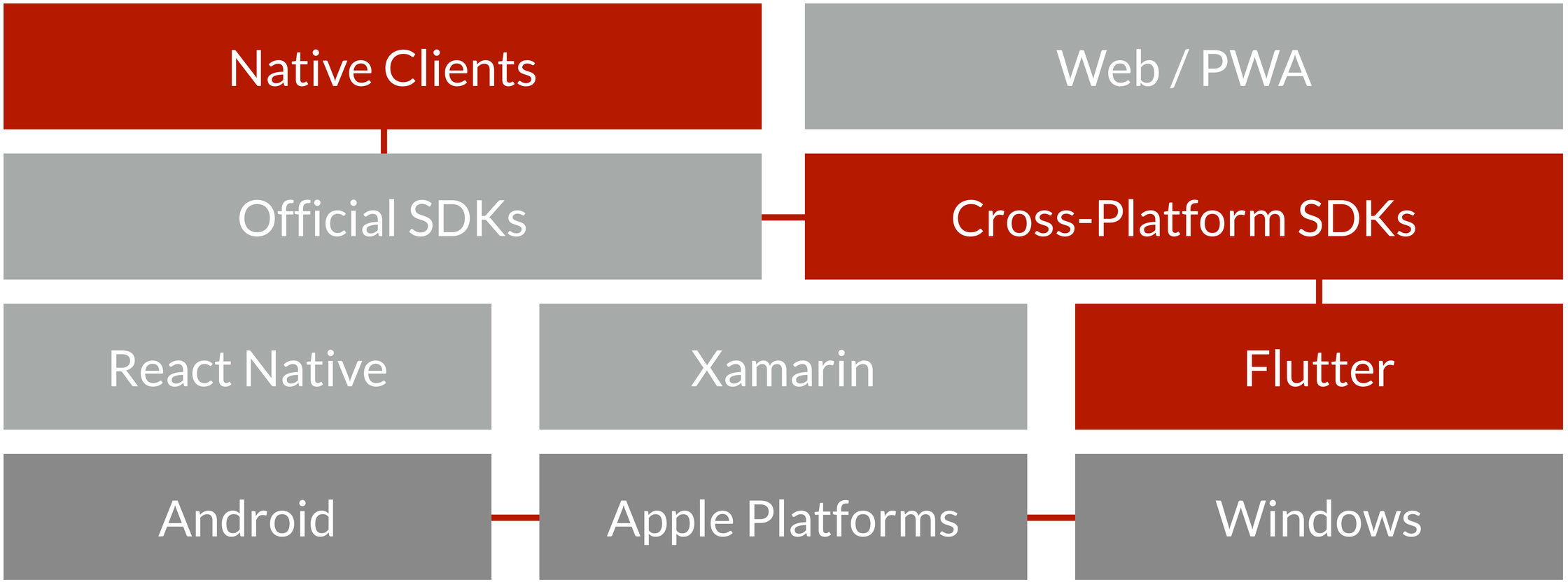
Xamarin

Flutter

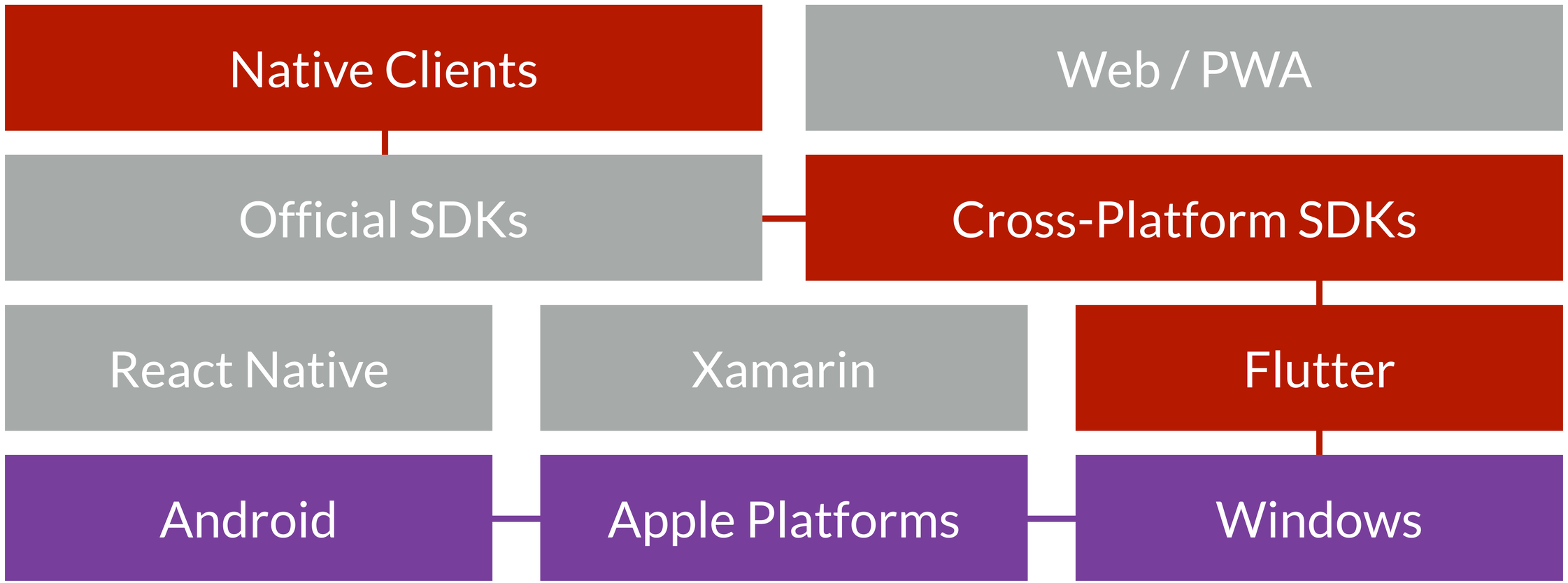
Frontend Map



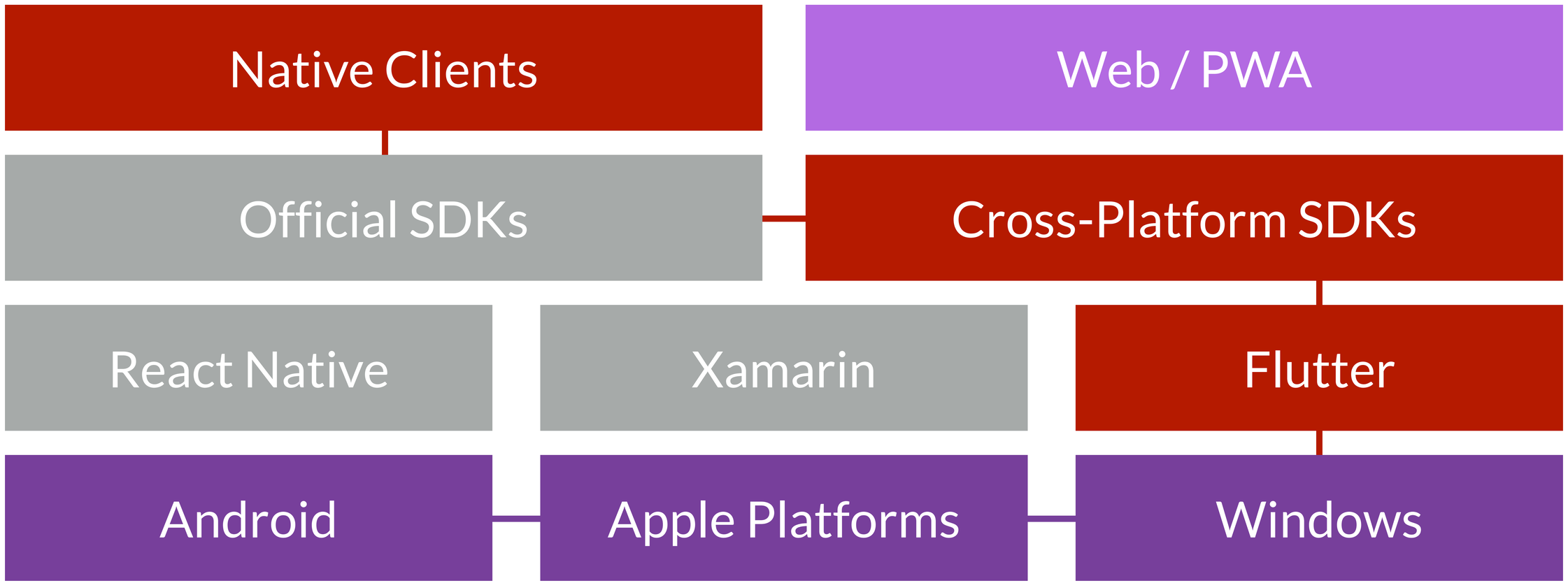
Frontend Map



Frontend Map



Frontend Map



Our Project

- App from scratch
- Coffee Store
- Focus on Material Design
- Download assets and coding help

Dart

- By Google
- Open Source
- Statically typed language
- Type Inference
- Multi-platform
- Easy to Learn
- Concise - less ceremony
- Modern Ideas
- Null-safety

Dart

Originally for internal web apps

Original intention: replace JavaScript

Current intention: front-end apps, AngularDart, and Flutter

Inspired by C, Java, JavaScript, Erlang, Smalltalk, Swift/Kotlin (Dart 2.0)

Compiles to native, IL, JavaScript

We'll be using Dart 2.x

Play with it at dartpad.dev

Dart

Every Dart app has a *main* function

Full OOP language with type inference

Null-safety is available as an optional feature in Dart 2.12

It feels easy to understand

It has features from many languages:

- Extensions
- Mixins
- Futures (async programming)

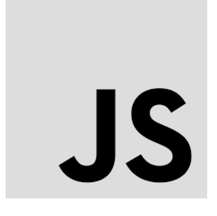
When compiling to Web: remember the engine is the JavaScript VM

Language Types

Developers write code in

Then ship

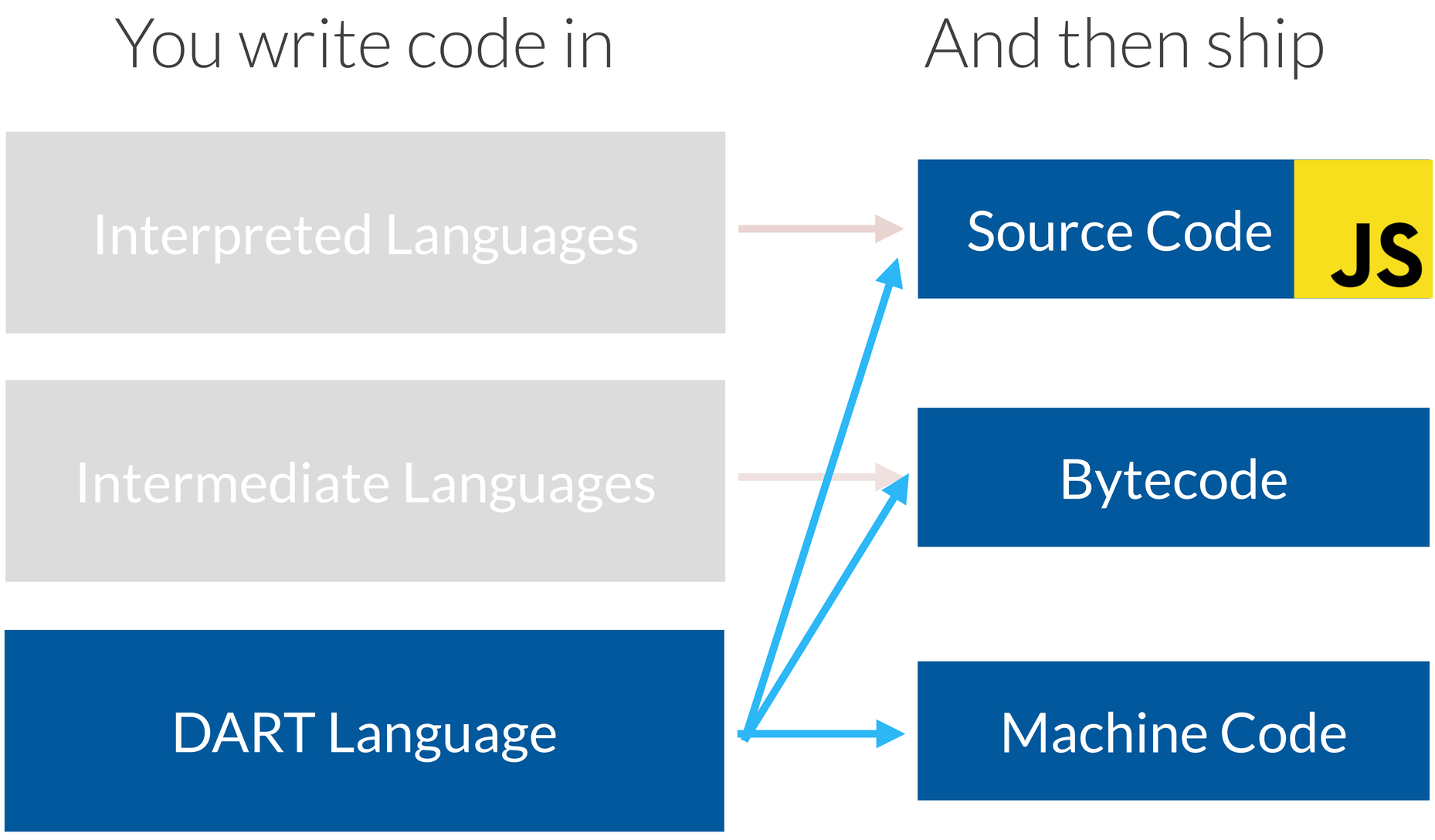
Interpreted Languages → Source Code



Intermediate Languages → Bytecode

Compiled Languages → Machine Code

Language Types



Time to write some Dart code

Flutter

Made by Google

Flutter is Google's UI toolkit for building beautiful, natively compiled applications for mobile, web, desktop, and embedded devices from a single codebase.

Flutter

- Declarative UI framework
- Launched in 2018
- Focus on modern UI patterns:
  - Single Source of Truth
  - Composable components
- Multiplatform
- Dependency Rendering
- Data Binding & Reactive programming
- UI expressed in Widgets in Dart classes
- Visual editor tooling in Android Studio

Flutter

Mobile devices

-Android

-iOS

-iPadOS

Embedded devices

-Fuchsia OS

Desktop devices

-Windows (win32 y UWP)

-macOS

-Linux

Web platform

-PWAs

Flutter

Google Play Store
- AAB to Android and Chromebook

Amazon AppStore
- APK to FireOS and Windows 11

Huawei AppGallery
- APK to HarmonyOS

Apple AppStore
- IPA to iOS and iPadOS

Microsoft Store
- UWP to Windows 10/11

Flutter

Not using OS SDKs

Your own "widgets" and design

It comes with two widget sets ready to use:

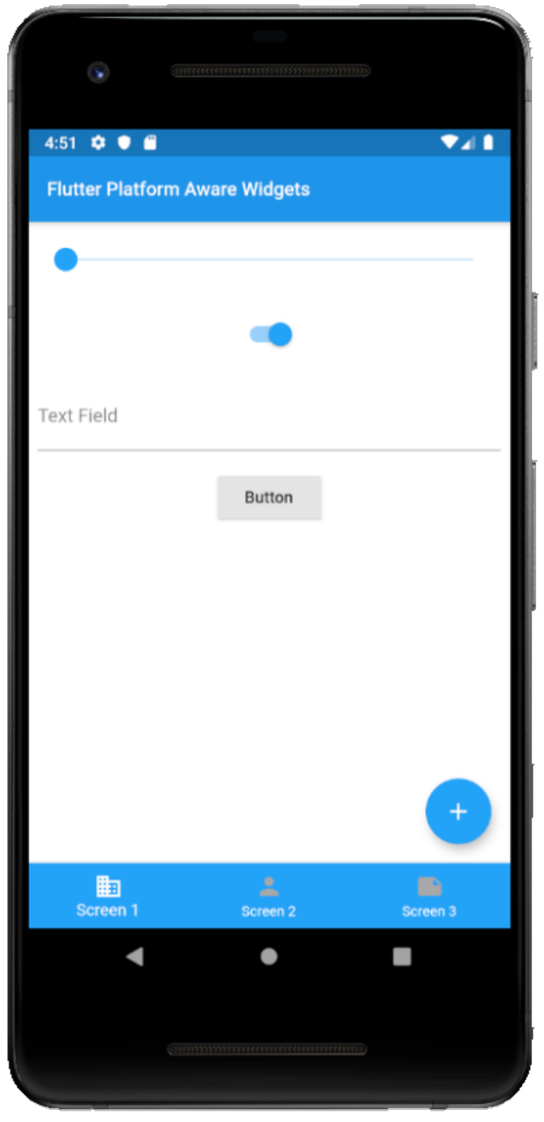
- Material
- Cupertino

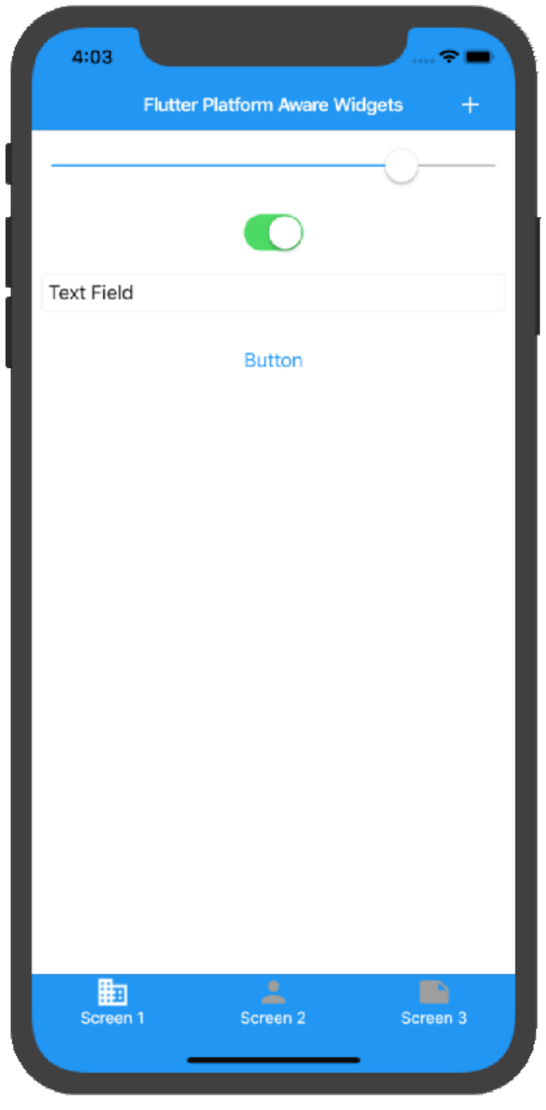
They clone Android Material Design and Apple Human Interface guidelines

Pixel-perfect, high-performance

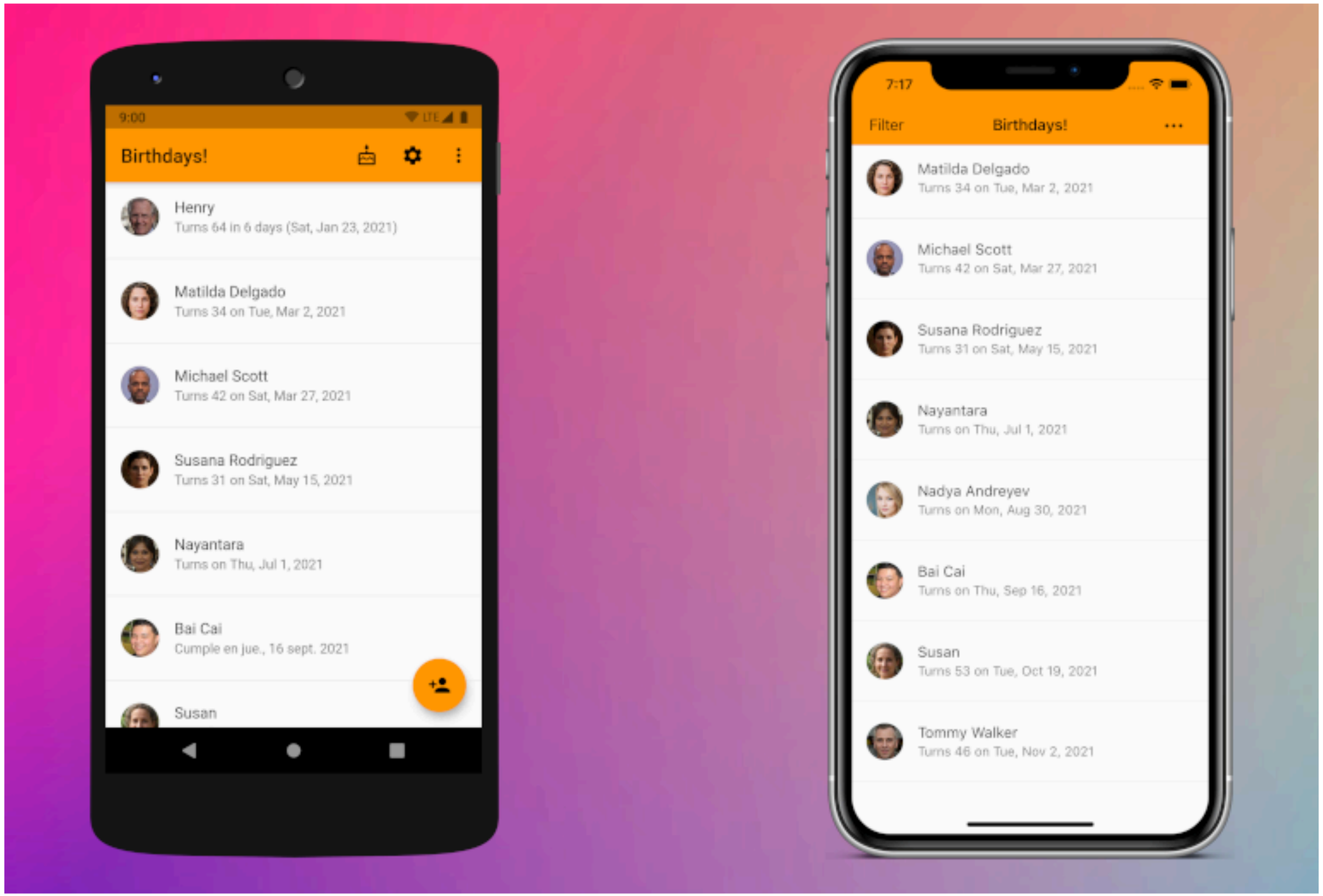
Same gestures and animations

Material vs. Cupertino





Material vs. Cupertino



Decisions to Make

Design your own visual pattern

Use Material for all devices

Multi UI
Cupertino for iOS/iPadOS and Material for other devices

Multi user interface

Doesn't happen automatically

Widgets are not translatable

There are design patterns to reduce coding apps twice

Product from a Flutter project

IPA: iOS App (compatible with iPadOS)

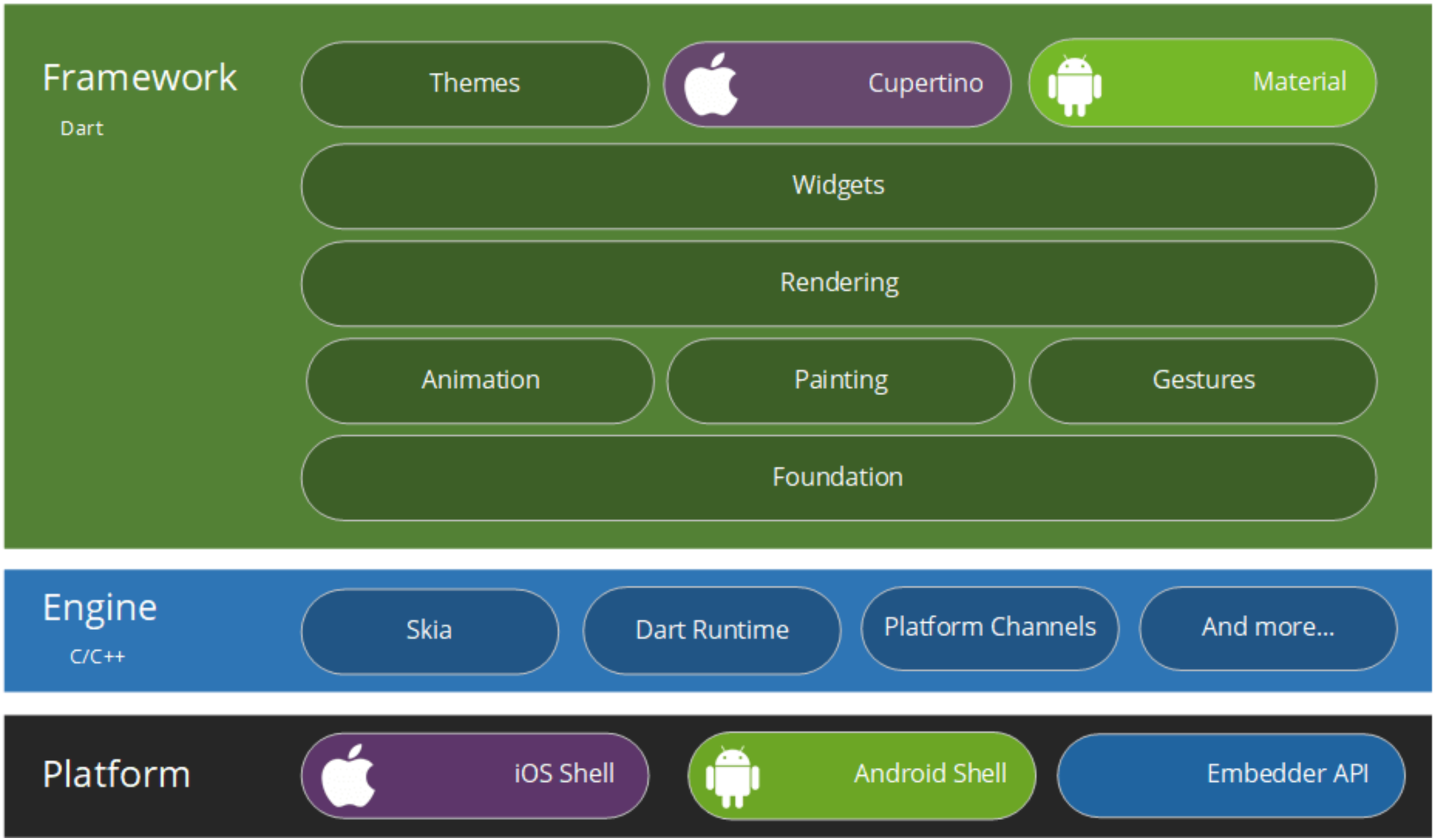
APK and AAB: Android App

Web build folder: ready to deploy - PWA

EXE: Windows (beta)

APPX: Windows Universal (alpha)

App: macOS (beta)


Framework
Dart
Themes
Cupertino
Material
Widgets
Rendering
Animation
Painting
Gestures
Foundation
Engine
C/C++
Skia
Dart Runtime
Platform Channels
And more...
Platform
iOS Shell
Android Shell
Embedder API

Flutter and Native Projects

Flutter generates:

- Android Studio native SDK project
- Xcode project for iOS
- Web standard files

We can have a hybrid Flutter app, part Flutter, part native SDK or JavaScript

Flutter SDK

Flutter Console

Dart SDK

Flutter CLI

Flutter DevTools

Android Studio

- IDE from JetBrains (IntelliJ)
- User Interface Designer (native)
- Android SDK
- Android NDK
- Flutter and Dart support (plugin)
- Profiler and other tools
- AVD - Android Virtual Devices
- ADB - Android Debug Bridge
- Gradle

Visual Studio Code

- Code Editor

- Flutter and Dart plugins

- Integration with emulators and DevTools

- Plenty of additional plugins and services

Xcode

- IDE
- User Interface Designer (native)
- SwiftUI
- Profiler and other tools
- iOS Simulator
- Xcode CLI
- NO Flutter or Dart support
- **macOS only!**

To test and/or compile
iOS or iPadOS
applications we need a
macOS device

Time to create our project

Dart vs other languages

Dart uses AOT compilation to create native code for Android and iOS

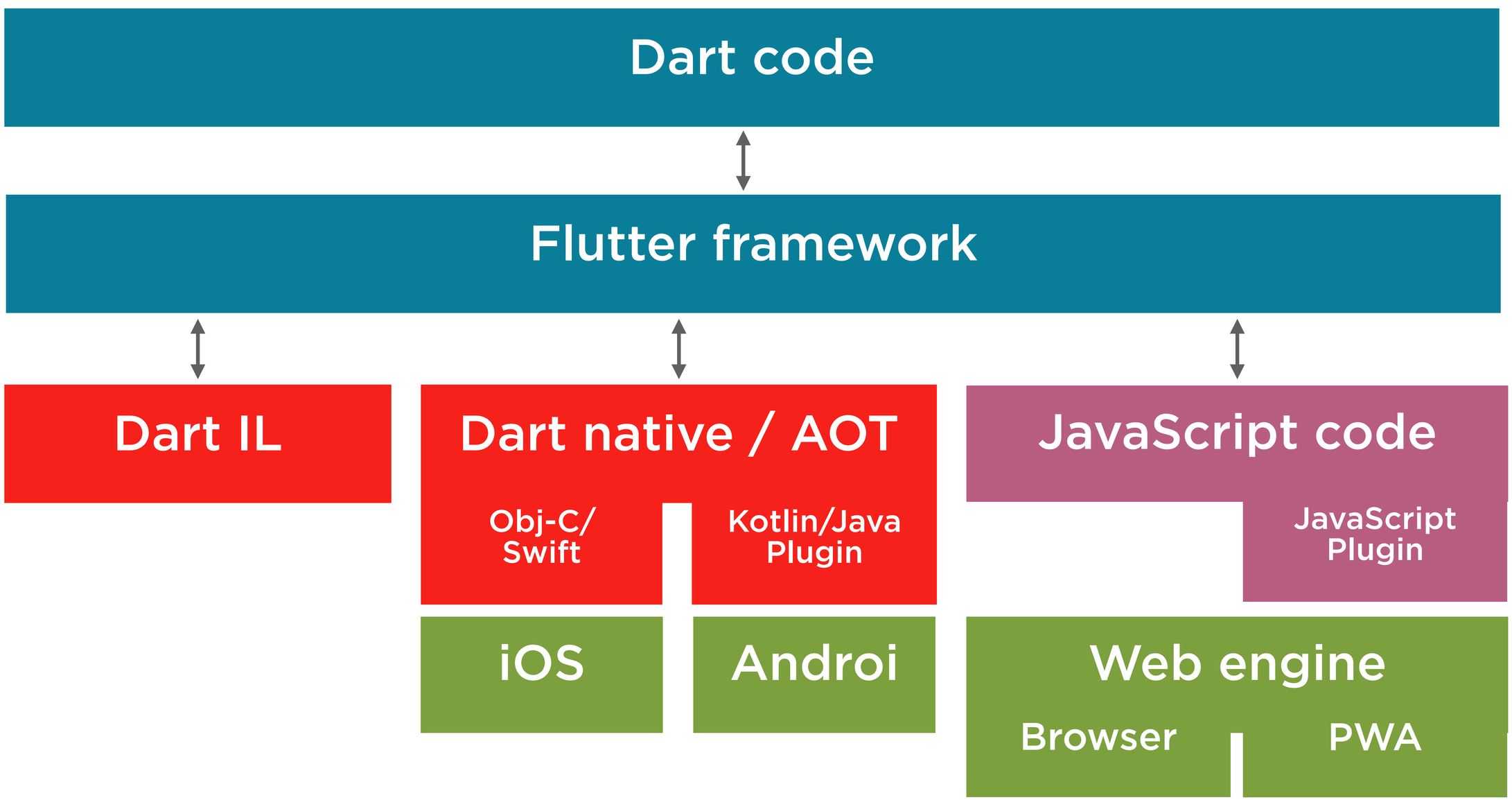
Android Runtime VM is not used

Dart compiles to VM-code for desktop

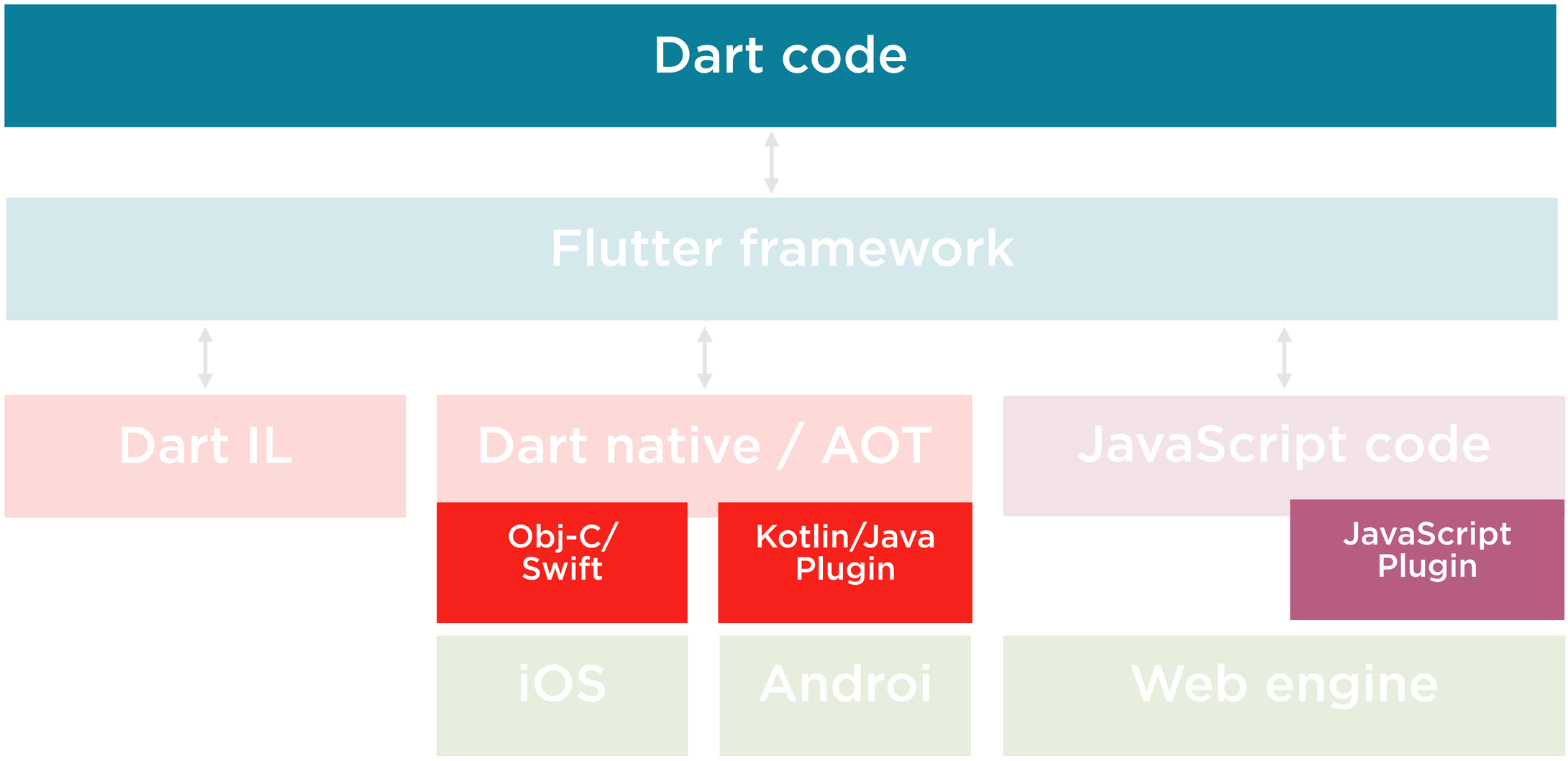
Dart transpiles to JavaScript for the Web

We can create plugins to connect Dart with
- Android and iOS SDK
- Web APIs
- Native SDKs

Flutter Code



What we write



When using third-party plugins we need to check platform compatibility

Flutter

Main Concepts

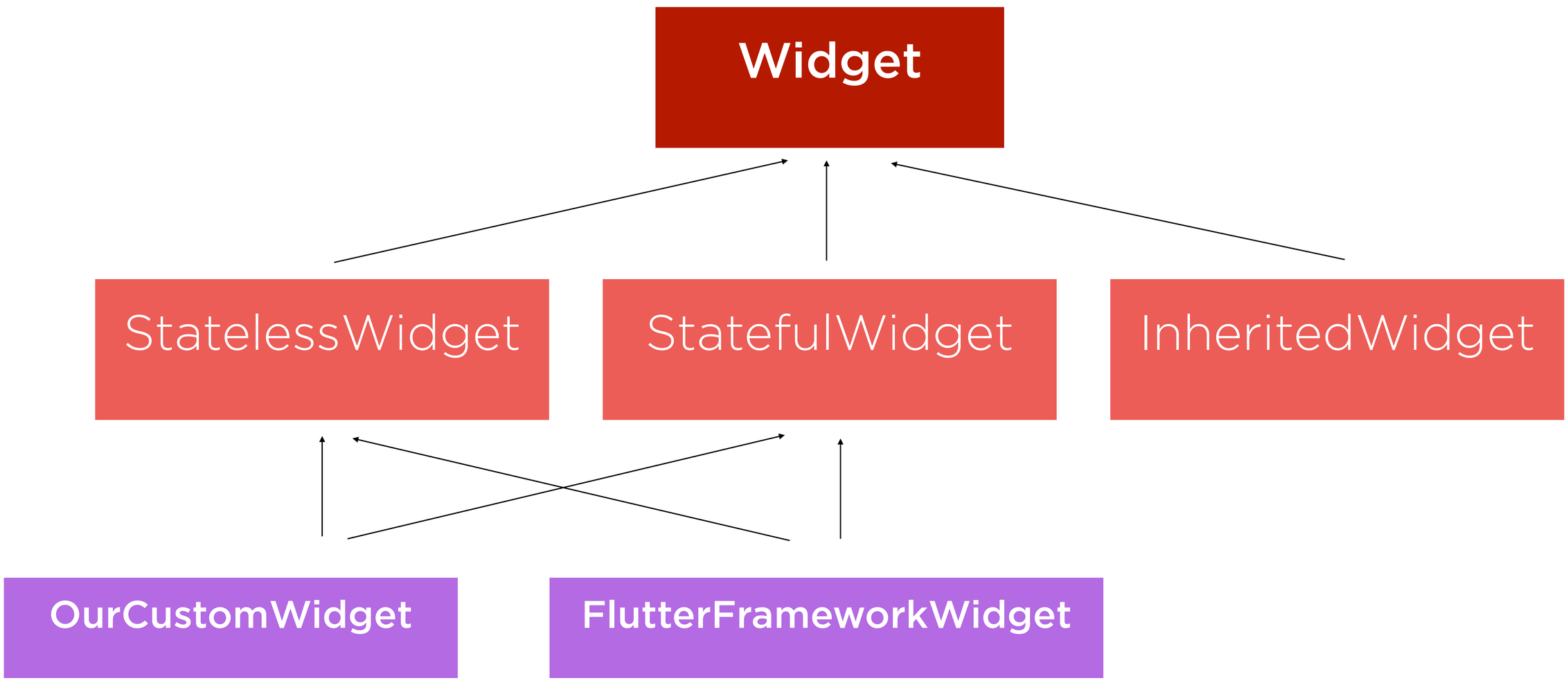
```dart
import 'package:flutter/material.dart';

void main() {
  runApp(          );
}
```

Within the main function, we call runApp and pass a **widget** instance as an argument

That creates a **Widget Tree**

Widget class



Flutter

Widgets

Basic unit for user interface

Only property: *key*

They have a build method that returnsother Widgets

They typically have a box within the canvas but there are invisible widgets as well

There are mainly two kinds:

* **Stateless widgets** (literal or parametrized)
* **State-full widgets**

Flutter includes +150 widgets! 😱

Widget Constructors

```
var widget = WidgetName()

var widget = WidgetName(property: value,
                        property: value)

var widget = WidgetName.builder()
```

Widgets

Standalone

Containers for one Child

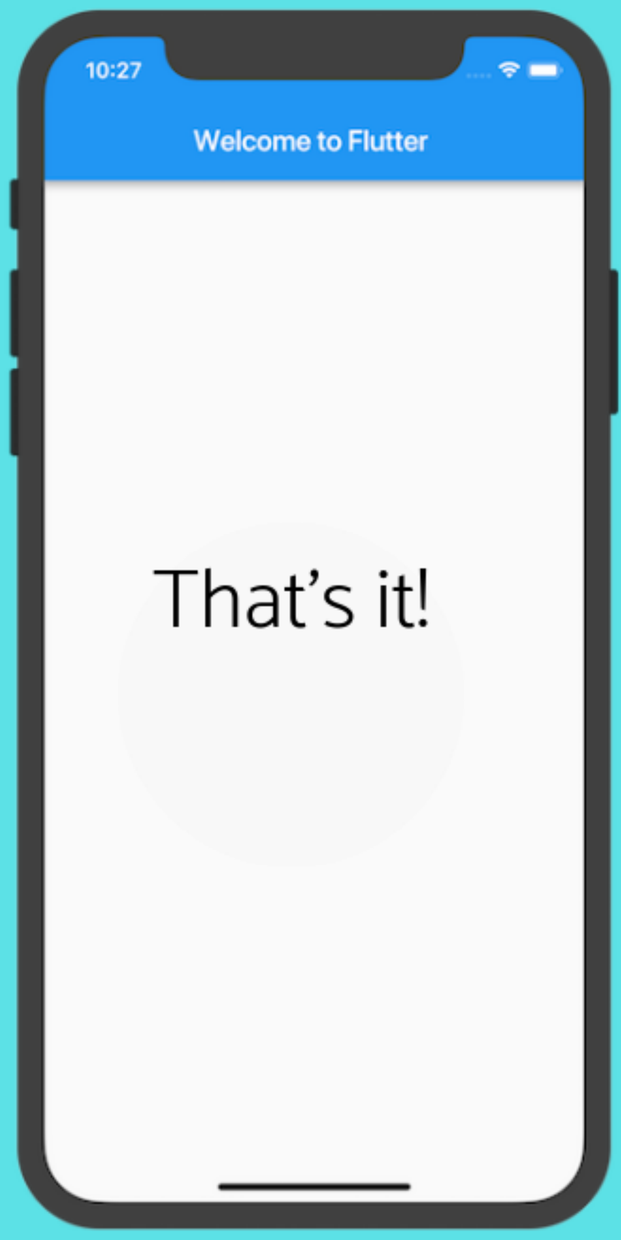
child property Widget

Containers for Children

children property <Widget>[]

Complex Containers

different properties Widget




```dart
import 'package:flutter/material.dart';

void main() => runApp(MyApp());


class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Welcome to Flutter',
      home: Scaffold(
        appBar: AppBar(
          title: Text('Welcome to Flutter'),
        ),
        body: Center(
          child: Text('That's it!'),
        ),
      ),
    );
  }
}
```

Flutter

Common Widgets

Scaffold

AppBar

Container, Expanded

ElevatedButton, TextButton, IconButton

Text, RichText

Column, Row, Wrap

Center

Padding

Image, Icon

Creating Widgets

Stateful

Stateless

Stateless widget

```dart
class Name extends StatelessWidget {
  const Name({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Container();
  }
}
```

TIP: Use stless in Android Studio/VS Code for snippet

Stateful widget

```dart
class Name extends StatefulWidget {
  const Name({Key? key}) : super(key: key);

  @override
  _NameState createState() => _NameState();
}
```

```dart
class _NameState extends State<Name> {
  @override
  Widget build(BuildContext context) {
    return Container();
  }
}
```

TIP: Use stful in Android Studio/VS Code for snippet

Flutter

Stateful Widgets

Most of the time, we don't touch the Widget class

We use the build method of the State class

State properties are set in the State class

State properties MUST NOT be changed directly

We change state values by calling:
setState(() { })

setState receives a Function as an argument; that function should update the state

The lifecycle will call build again and the new state should render an updated UI

Stateful widgets

Most of the time, we don't touch the Widget class

We use the build method of the State class

State properties are set in the State class

State properties MUST NOT be changed directly

We change state values by calling:
setState(() { })

setState receives a Function as an argument;
that function should update the state

The lifecycle will call build again and the new state should render an updated UI

hi@firt.dev
@firt