Mobile Application Development

Introduction to Flutter & Dart

Flutter

App Development

Instructor: Engr. Farhan Ghafoor

Associate Professor, Software Engineering Dept, CoE

Learning OUTCOMES

At the end of this lesson, learners should be able to:

- Describe what is Flutter
- Explain what is a Flutter widget
- Understanding of Stateless and Stateful widgets
- Design a Flutter mobile app using Scaffold widgets

What is Flutter?

Flutter is a user interface toolkit developed by Google that can be used for developing mobile applications.

- Developers use Dart programming language to create a Flutter mobile app.
- Dart is an object-oriented, open-source, and general-purpose programming language
- Dart is a general-purpose, object-oriented, open-source programming language with C-style syntax that was also developed by Google.

Technical Overview

What is flutter SDK?

- An app SDK capable of providing high performance applications for
  - iOS & Android,
  - Web (Hummingbird – work in progress)
  - Even Desktop (Work in progress).
- High Performance Applications that will have same natural feel as if it was written on the same platform
- Apps are written in Dart language

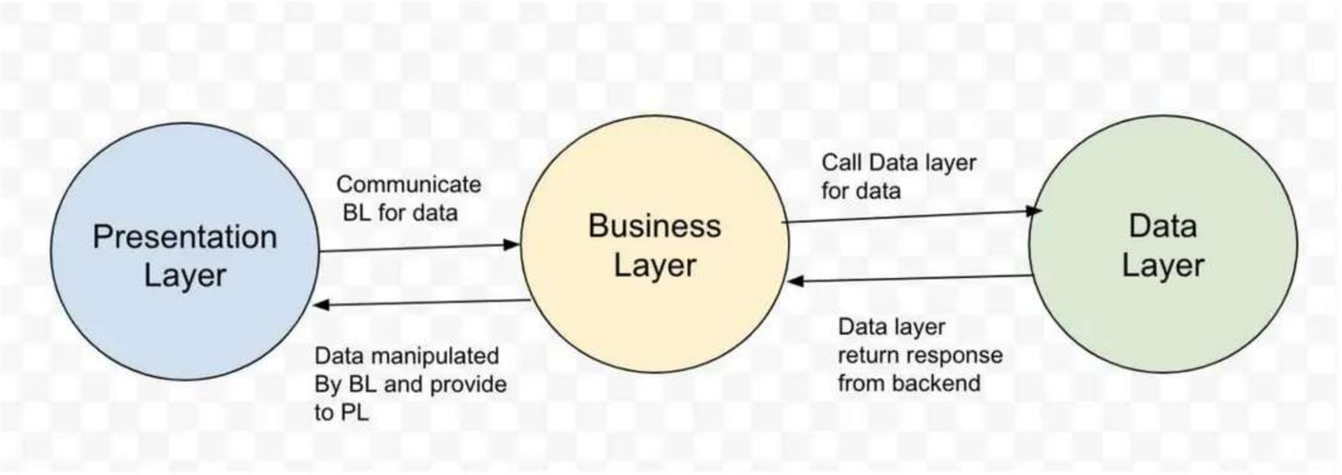
What is Dart?

- An object-oriented language.
- Supports Ahead-of-Time (AOT) and Just-in-Time (JIT) compilation
- C-style syntax
- Statically Type

Dart Basics

- Dart Pad Editor:
  - https://dartpad.dev/?
- Functions
- Variables and Values
- Types
- Classes

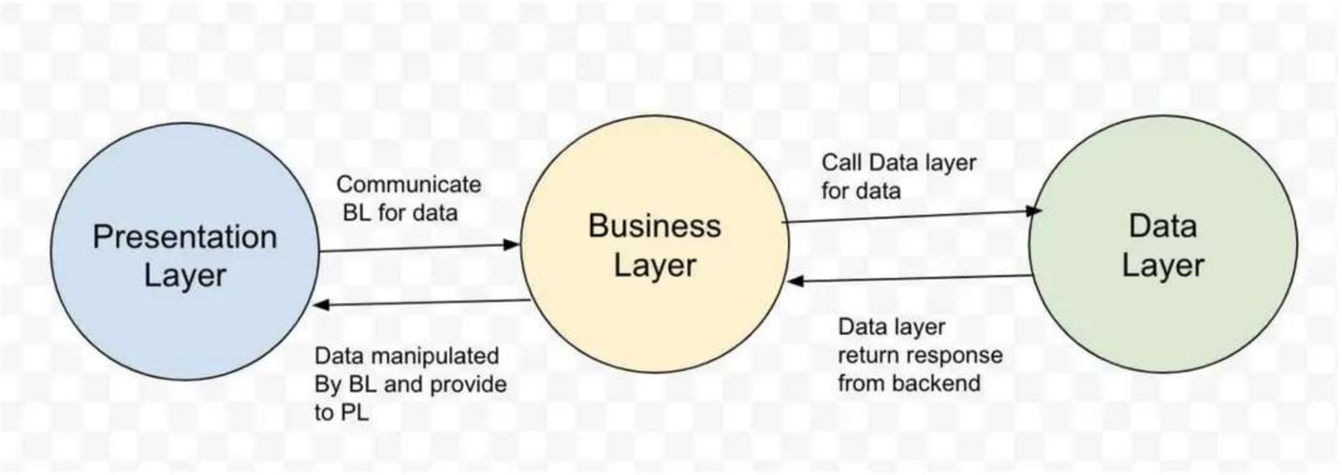
Mobile App Architecture



Presentation Layer (PL)

- This Layer focuses on UI Components (View and Controller)
- Defines the way, the mobile app will present itself in front of the end user
- On this Layer, just focus on features and their locations

Mobile App Architecture



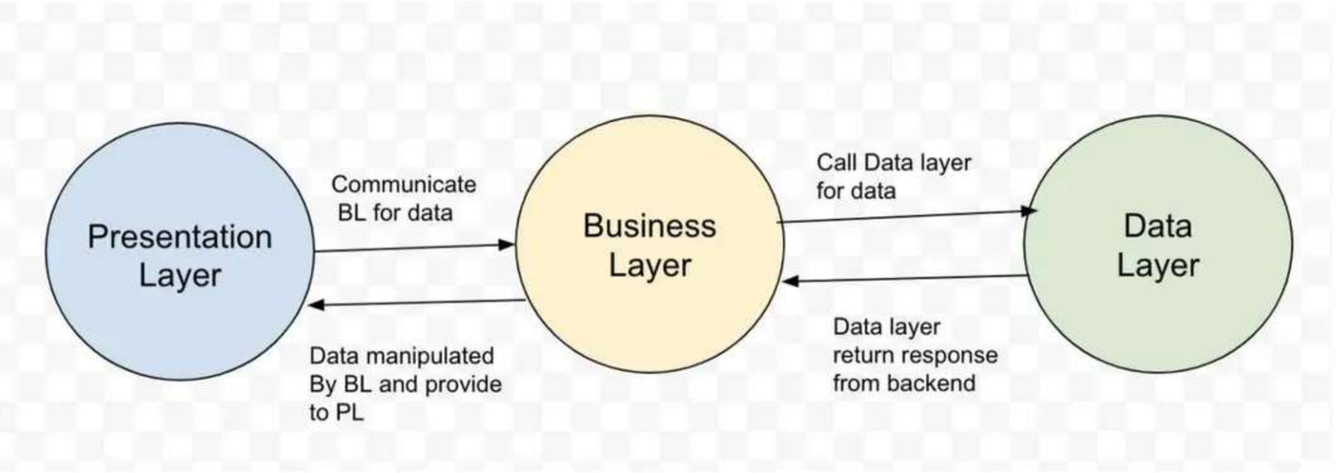
Business Layer (BL)

- Focuses on Business logic
- Includes workflow, Business components
- Service Layer and Domain Layer
  - **Service Layer**: Defines a common set of application functions that will be available to the client and end users
  - **Domain Layer**: Focuses on the domain specific problems

Mobile App Architecture

Data Layer (DL)

- It's the final layer!
- data access components (DAC),
- Data Helpers/Utilities
- Service Agents
- These three components control the — Persister



➢ **Since now we know the basic architecture, here comes the question!**

➢ **Why we want to use flutter ?**

➢ **Or why we should use it ?**

Why Learn Flutter and Dart?

Benefits associated with:

- It's highly productive! It is Perfect for an MVP!
- Develops iOS and Android applications from single code base
- Prototype and iterate easily: — Experimental feature can reflect instantly by hot reload, fix crashes and continue debugging from where the app left off
- Creates beautiful, highly customized user experiences (has its own rendering engine)
- Consistent UIs across devices and manufacturers
- Superb performance

Why Learn Flutter and Dart?

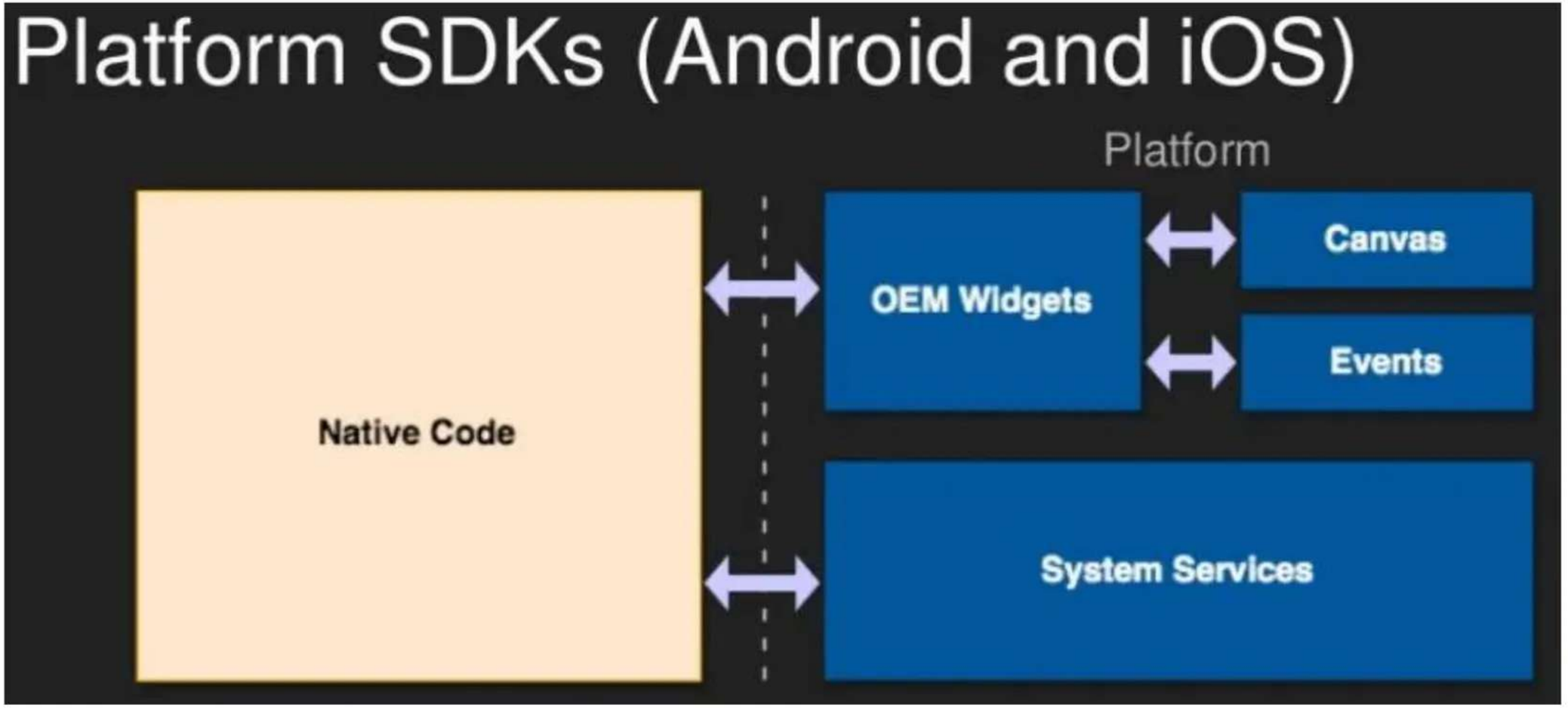
Benefits associated with Flutter

- Material Design and Cupertino available for iOS widgets
- Resembles React Native
- No need to use a JavaScript bridge, which improves app startup times and overall performance.
- Once your UI works, it just works. And keeps working.
  - Manufacturers / OS versions / different devices can't break it
- Dart achieves this thanks to Ahead-of-Time (AOT), compilation. Dart also makes use of Just-in-Time, or JIT, compilation.

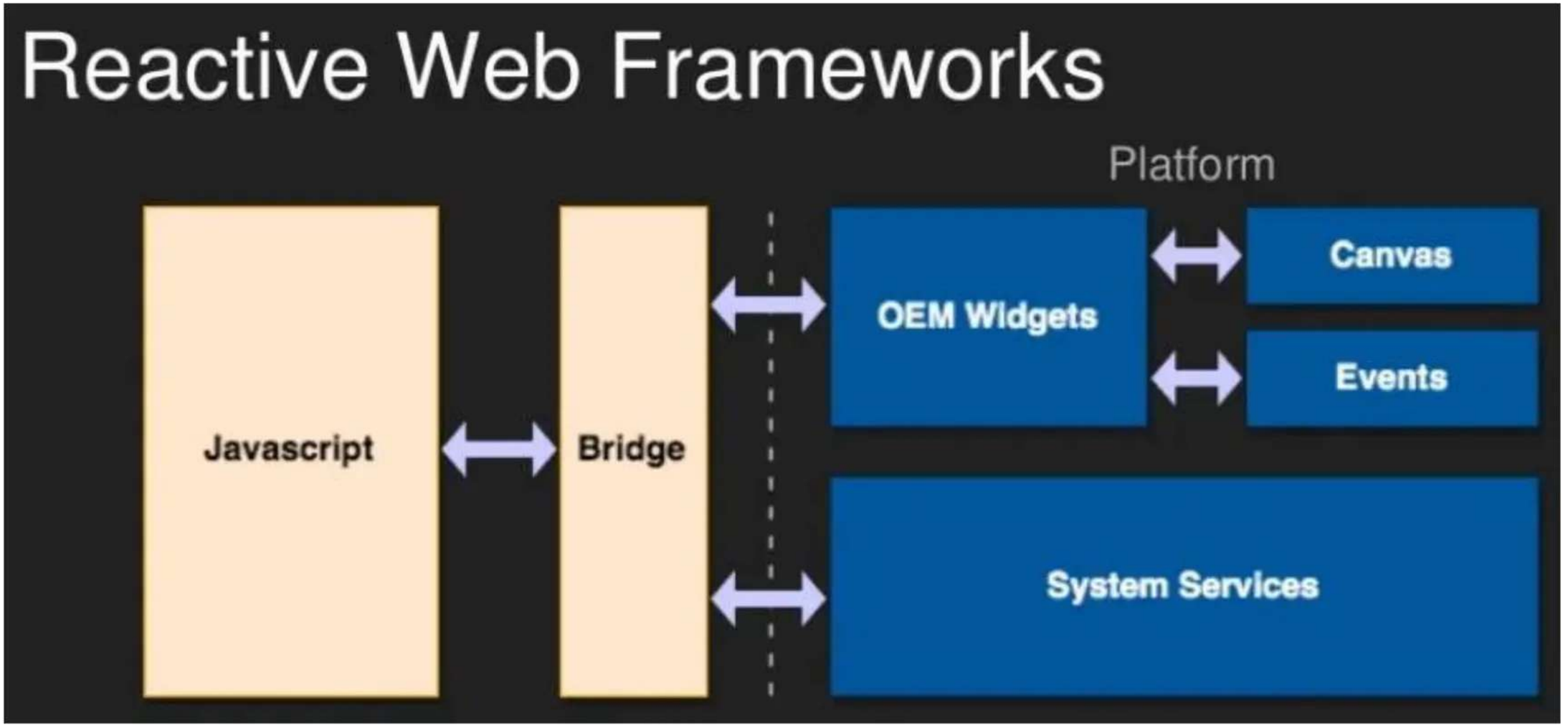
Why Learn Flutter and Dart?

- Flutter includes:
  - A modern react-style framework,
  - A 2-D rendering engine,
  - Ready-to use widgets, and
  - Development tools.
- Flutter basically provides all those components which are required to build a
  - Highly interactive and
  - Personalized application for fast development
- Especially Android APIs require a lot of ceremony for simple things
  - Flutter was able to start from scratch and avoid previously made pitfalls

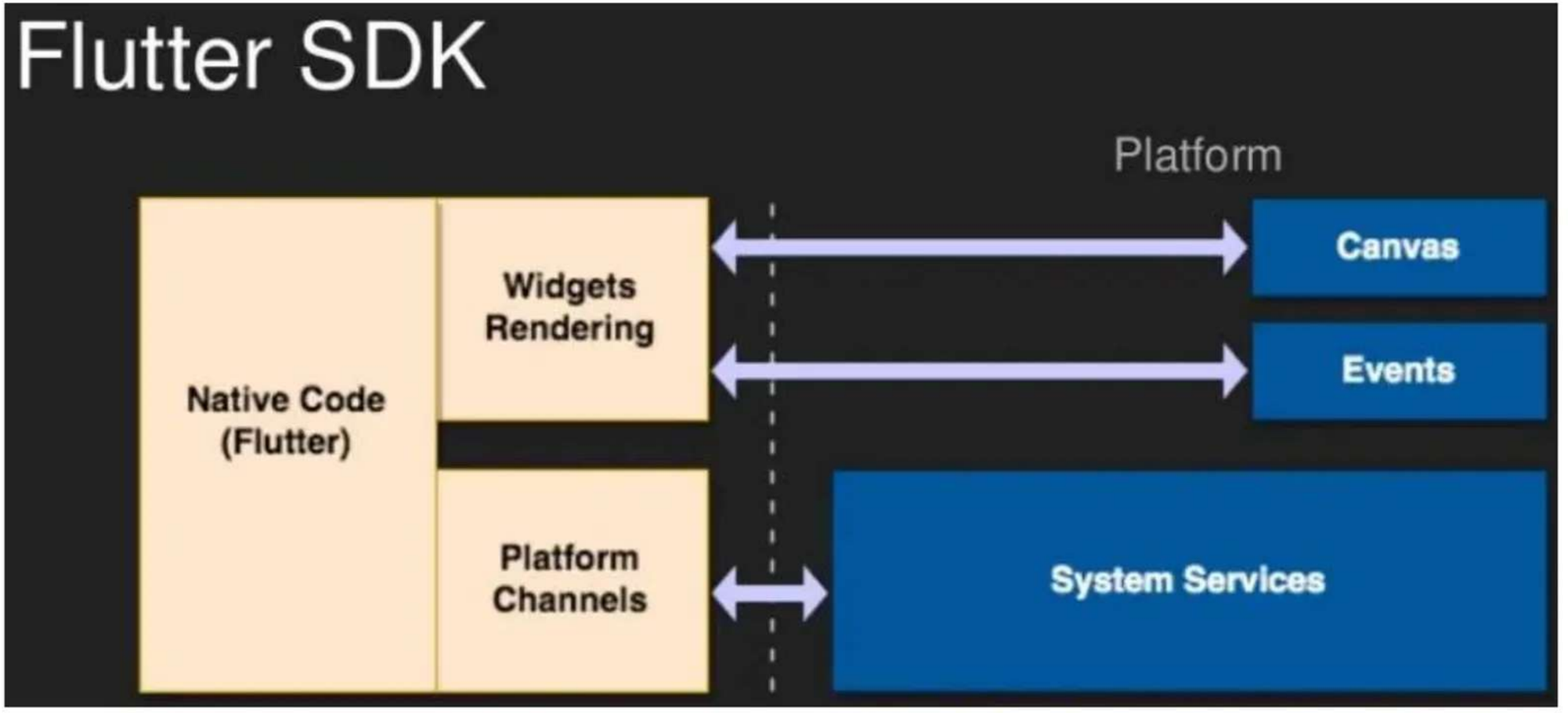
Platform SDKs (Android and IOS)



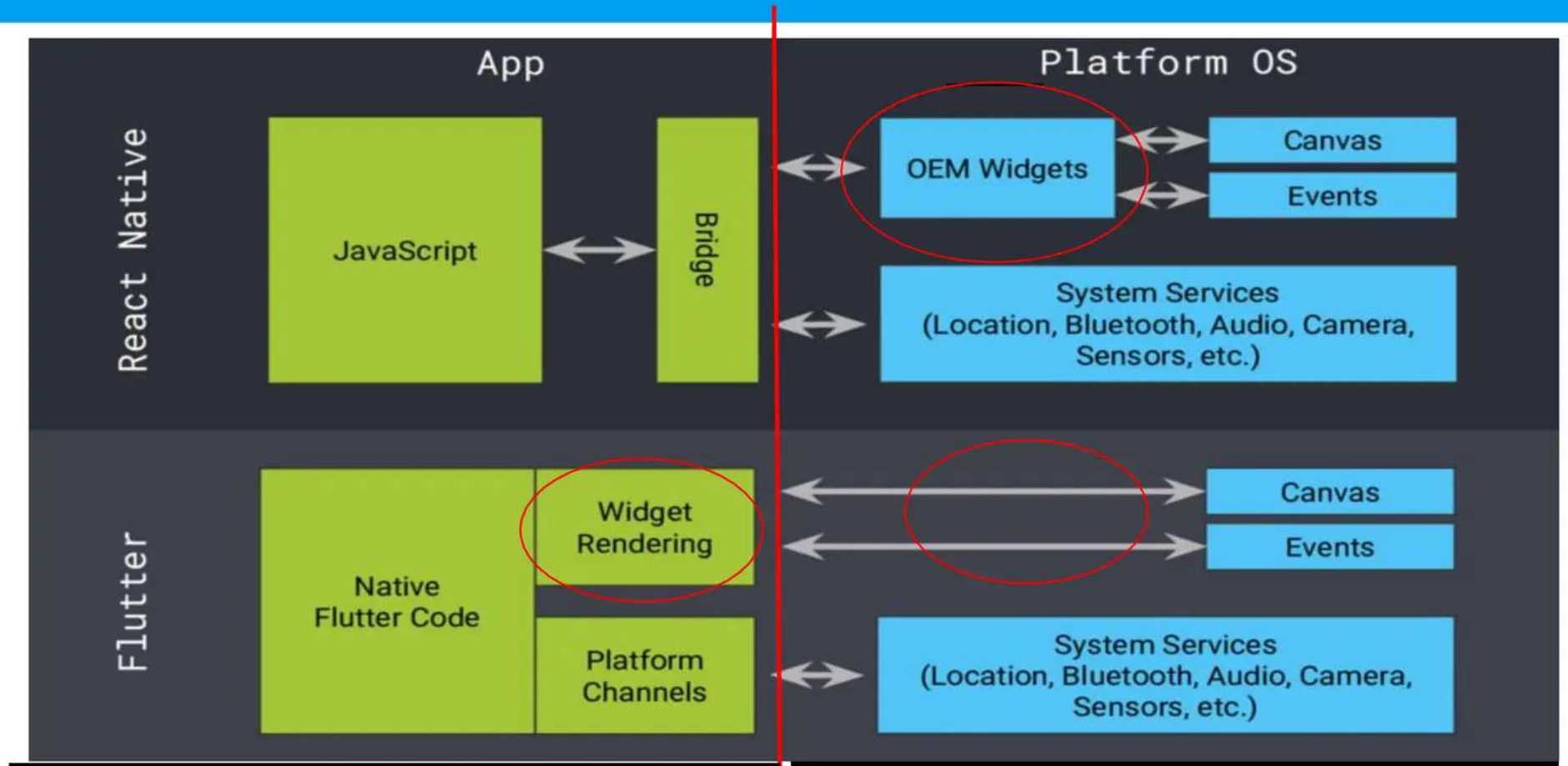
Reactive Web Frameworks



Flutter SDK


Flutter SDK
Platform
Native Code
(Flutter)
Widgets
Rendering
Platform
Channels
Canvas
Events
System Services

Flutter vs React Native



Our cross-platform code

- Everything in our control
- Things we do here have fantastic performance and are cheap
- We should stay here as much as possible

Native land

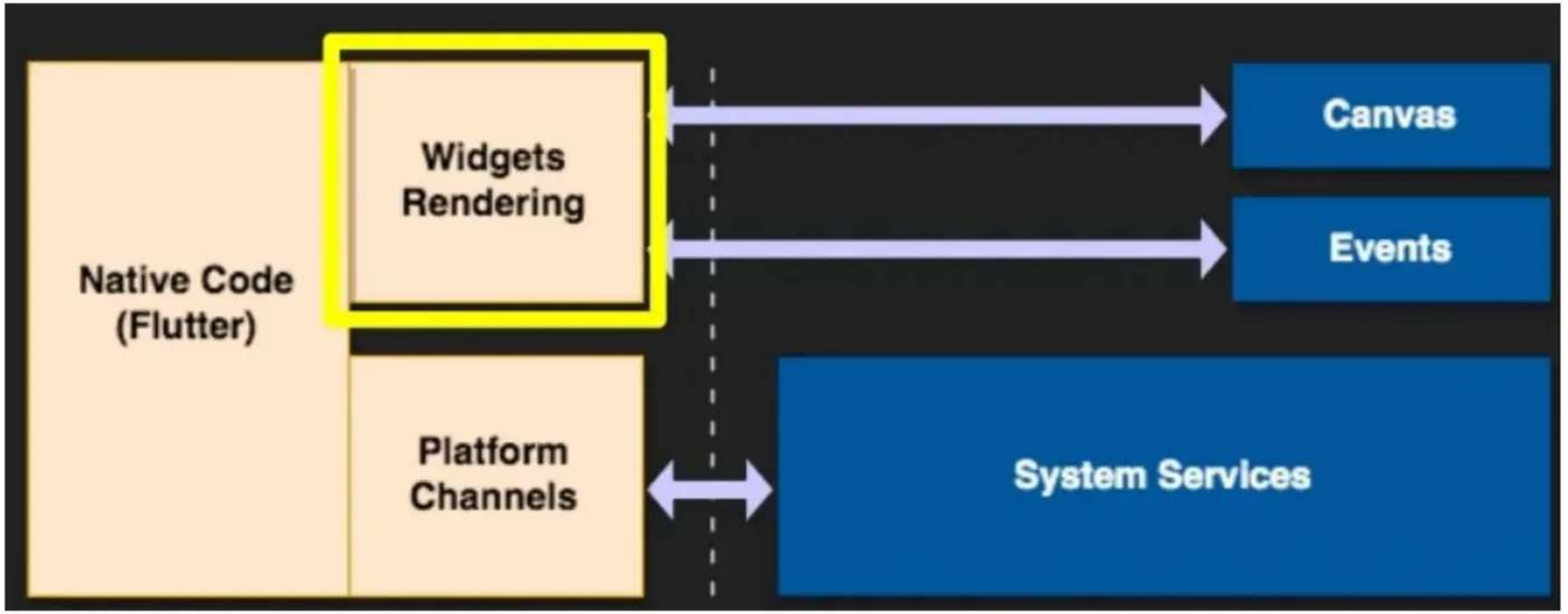
- Also fast performance here, however:
  - Expensive to travel to
- We can't afford to go here too often, just like we can't afford beach vacations every week

Base image from: https://speakerdeck.com/passsy/flutter-60-fps-ui-of-the-future

Flutter - Native Plugins

- Allow access to every native platform API
  - Bluetooth, geolocation, sensors, fingerprint, camera, etc.
- Both official and community-driven plugins available
- Some plugins missing or in early stages
  - There's a community-driven geolocation plugin with really limited API
  - There's a community-driven Bluetooth plugin that doesn't work with iOS just yet
- If a plugin for your use case doesn't exist, you'll have to make it yourself
- This is where other frameworks like React Native & Xamarin currently shine and Flutter takes the loss
  - Situation expected to be solved with time

Flutter SDK Concepts



Widgets

Introduction to Widgets

- Flutter widgets are built using a modern framework that takes inspiration from React.
- Building blocks of UI in Flutter
- The whole app is a Widget. A screen is a Widget that contains Widgets.
- What is a widget? Widgets can be compared to **LEGO blocks**.
- Advanced widgets are made by combining basic widgets.
- Widgets describe what their view should look like given their current configuration and state.

Widgets

- The whole app is a Widget. A screen is a Widget that contains Widgets.
- Can represent
  - UI element, such as Text, Button, BottomNavigationBar, TextField, etc.
  - Layout element, such as Padding, Center, Stack, Column, etc.
  - Completely new screen (Activity/ViewController equivalent)

Widgets

- Each widget is an immutable declaration of the user interface.
- Widgets are configurations for different parts of the UI.
- Placing the widgets together creates the widget tree
  - Like an Architect draws a blue-print of a house.

Widgets

➢ **Widgets create a widget tree.**

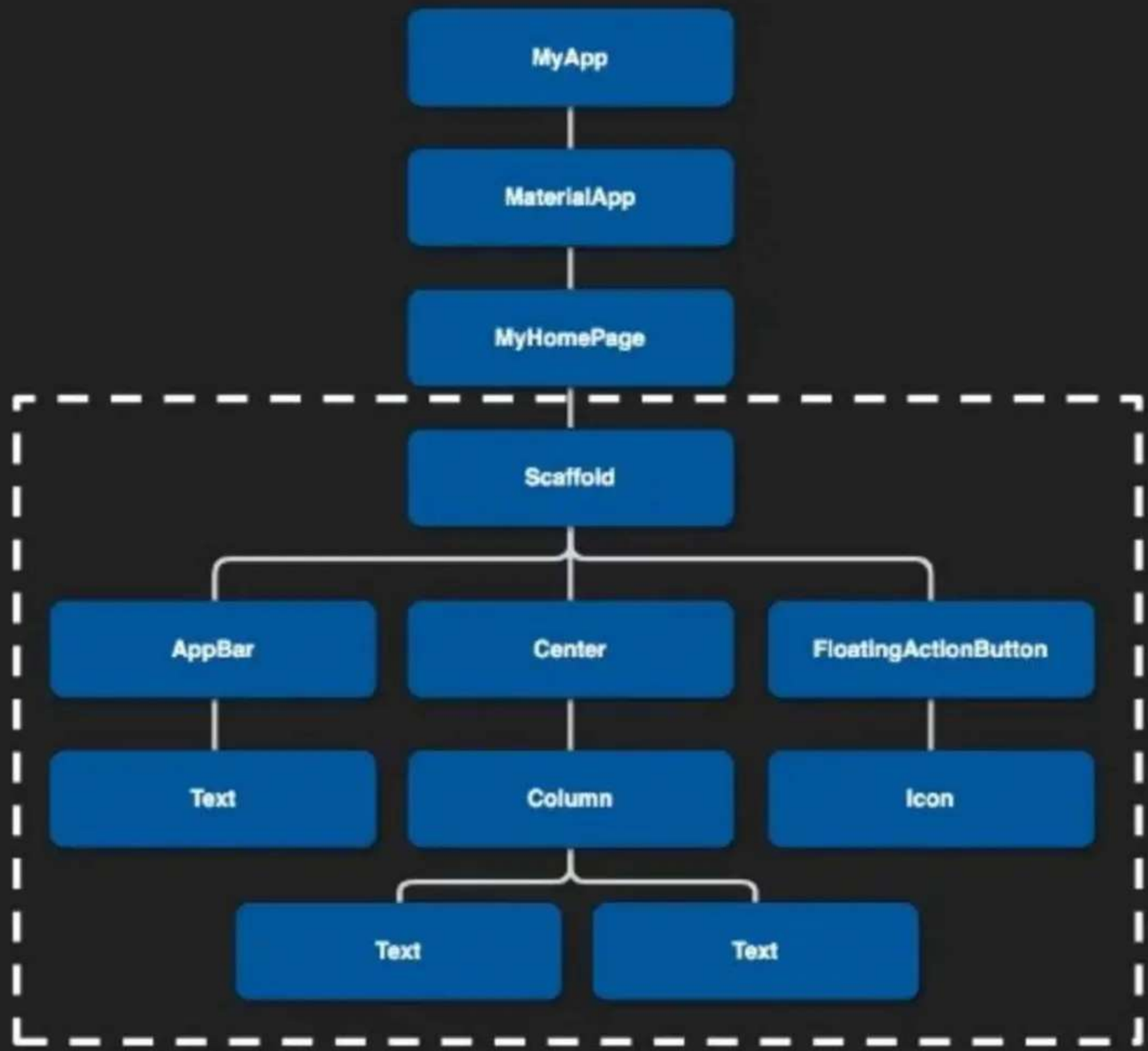
➢ When a widget's state changes,

- the widget rebuilds its description,
- Which the framework differentiates against the previous description
- Why?
  - To determine the minimal changes needed in the underlying render tree to transition from one state to the next.

Widget Tree

Widget Tree

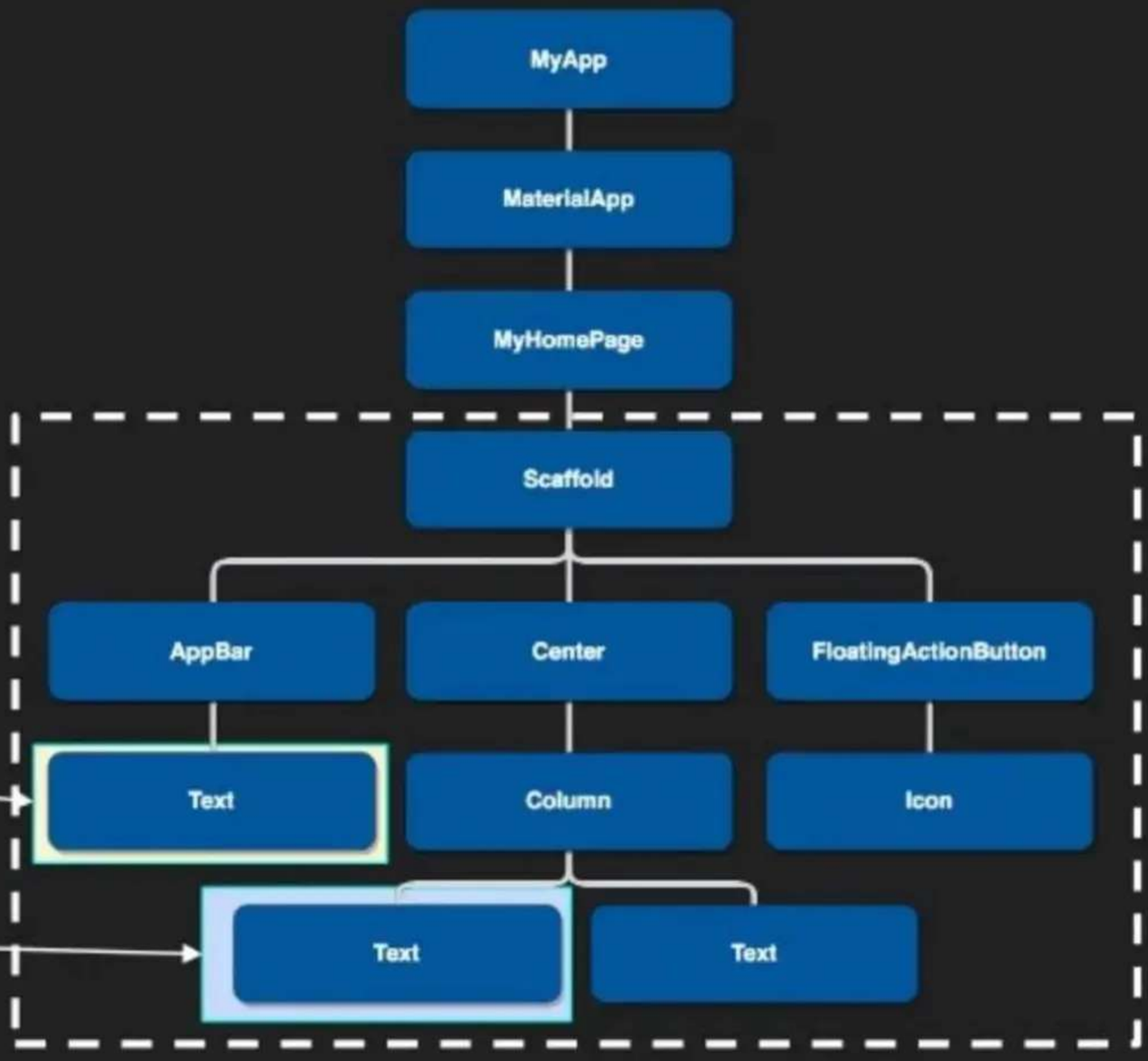
```dart
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text(widget.title),
    ),
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          Text(
            'You have pushed the button this many times:',
          ),
          Text(
            '$_counter',
            style: Theme.of(context).textTheme.display1,
          ),
        ],
      ),
    ),
    floatingActionButton: FloatingActionButton(
      onPressed: _incrementCounter,
      tooltip: 'Increment',
      child: Icon(Icons.add),
    ),
  );
}
```



BuildContext

- reference to the location of a Widget within the Widget Tree structure
- Belongs to one widget

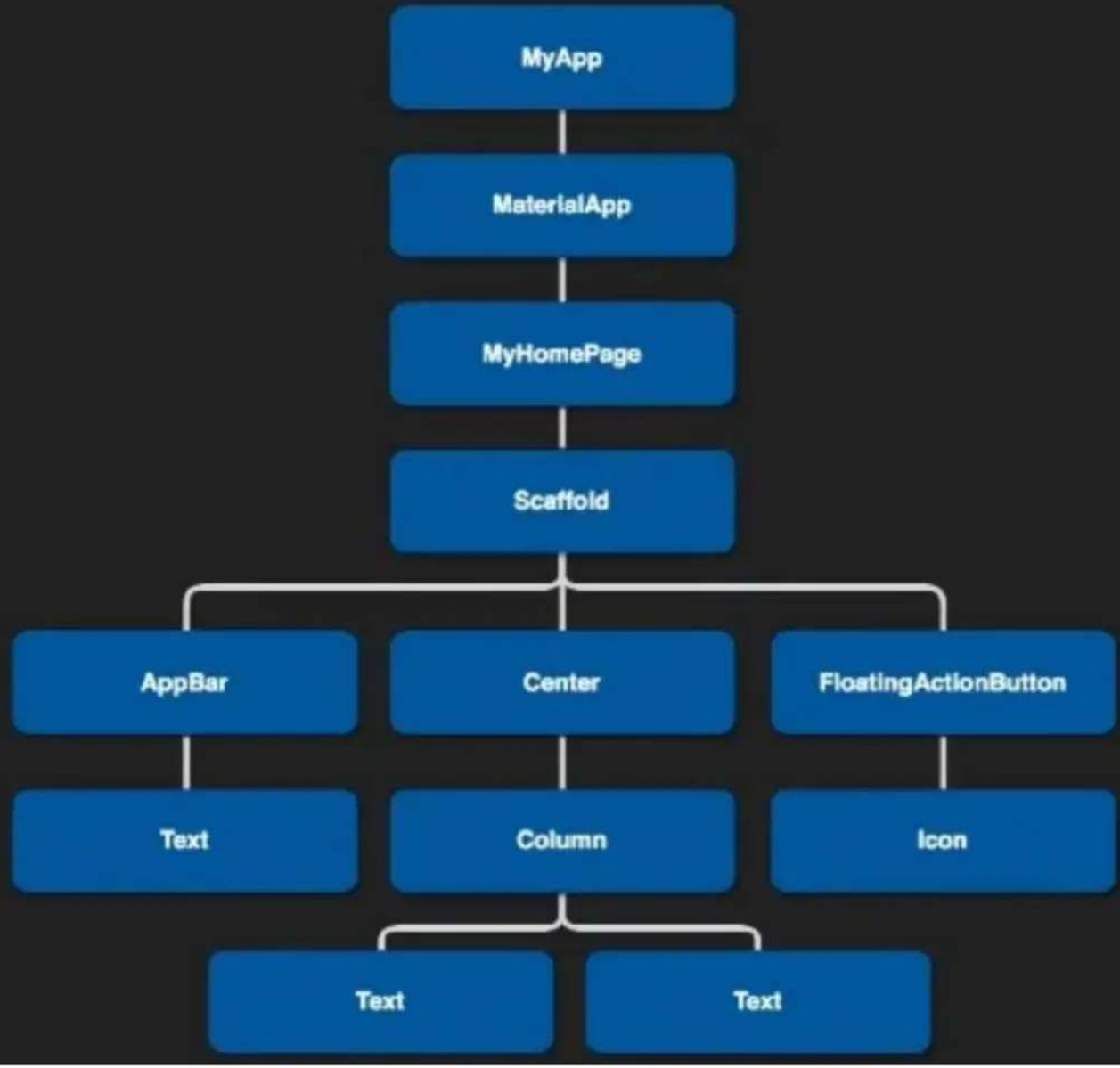
```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text(widget.title),
    ),
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          Text(
            'You have pushed the button this many times:',
          ),
          Text(
            '$_counter',
            style: Theme.of(context).textTheme.display1,
```



BuildContext Visibility

- ‘Something’ is only visible within its own BuildContext or in the BuildContext of its parent(s) BuildContext
- An example:

```
Theme.of(context)
```



Widget Catalog https://docs.flutter.dev/development/ui/widgets

Accessibility

Make your app accessible.

Visit

Animation and Motion

Bring animations to your app.

Visit

Assets, Images, and Icons

Manage assets, display images, and show icons.

Visit

Async

Async patterns to your Flutter application.

Visit

Basics

Widgets you absolutely need to know before building your first Flutter app.

Visit

Cupertino (iOS-style widgets)

Beautiful and high-fidelity widgets for current iOS design language.

Visit

Input

Take user input in addition to input widgets in Material Components and Cupertino.

Interaction Models

Respond to touch events and route users to different views.

Layout

Arrange other widgets columns, rows, grids, and many other layouts.

Widget Catalog – Material Design Widgets

- https://docs.flutter.dev/development/ui/widgets/material

Appbar

A Material Design app bar. An app bar consists of a toolbar and potentially other widgets, such as a TabBar and a FlexibleSpaceBar.

BottomNavigationBar

Bottom navigation bars make it easy to explore and switch between top-level views in a single tap. The BottomNavigationBar widget implements this component.

Drawer

A Material Design panel that slides in horizontally from the edge of a Scaffold to show navigation links in an application.

MaterialApp

A convenience widget that wraps a number of widgets that are commonly required for

Scaffold

Implements the basic Material Design visual layout structure. This class provides APIs for showing

SliverAppBar

A material design app bar that integrates with a CustomScrollView.

Widgets

Flutter has widgets with:

- Structuring elements such as a list, grid, text, and button
- Input elements such as a form, form fields, and keyboard listeners
- Styling elements such as font type, size, weight, color, border, and shadow
- to lay out the UI such as row, column, stack, centering, and padding
- Interactive elements that respond to touch, gestures, dragging, and dismissible
- Animation and motion elements such as hero animation, animated container,
- Animated crossfade, fade transition, rotation, scale, size, slide, and opacity
- Elements like assets, images, and icons
- Widgets that can be nested together to create the UI needed
- **Custom widgets you can create yourself**

Widgets - Stateless

```
class HelloWorldScreen extends StatelessWidget {
  final String message = 'Hello world!';

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Text(message),
      ),
    );
  }
}
```

- Immutable
- Once created, it doesn't change

Stateless Widgets - Lifecyle

```
class GreetingsScreen extends StatelessWidget {
  GreetingsScreen(this.message);

  final String message;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Text(message),
      ),
    );
  }
}
```

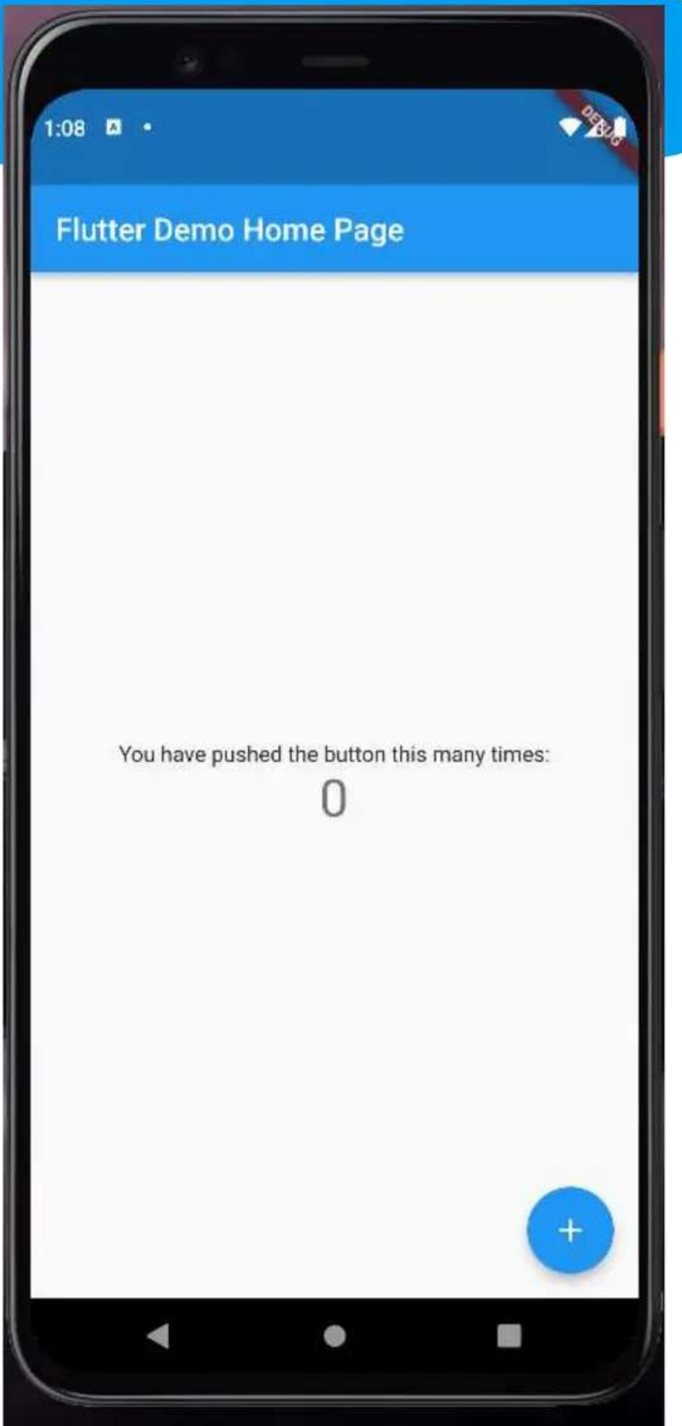
- Initialization
- build()

Stateful Widgets

```dart
class CounterScreen extends StatefulWidget {
  @override
  _CounterScreenState createState() => _CounterScreenState();
}

class _CounterScreenState extends State<CounterScreen> {
  int _counter = 0;

  void _increment() {
    setState(() {
      _counter++;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Counter Screen'),
      ),
      body: Column(
        children: <Widget>[
          Text('Counter is $_counter'),
          RaisedButton(
            child: Text('Increment'),
            onPressed: _increment,
          ),
        ],
      ),
    );
  }
}
```

- Have a “State”
- State - set of data held by a widget that can change in its lifetime

Stateful Widgets



Stateful Widgets - State

- State defines the information on how to interact with the Widget in terms of:
  - Behaviour
  - Layout
- Any changes to State will trigger the Widget to rebuild
- State is associated with BuildContext
- A State is considered **mounted** only when the State is associated with the BuildContext

Flutter architectural overview

- Flutter is a cross-platform UI toolkit
- It is designed to allow code reuse across operating systems such as iOS and Android, while also allowing applications to interface directly with underlying platform services.
- To enable developers to deliver high-performance apps that feel natural on different platforms, embracing differences where they exist while sharing as much code as possible.

Flutter architectural overview

During development:

- Flutter apps run in a VM that offers stateful hot reload of changes without needing a full recompile.

For release:

- Flutter apps are compiled directly to machine code, whether Intel x64 or ARM instructions,
- Or to JavaScript if targeting the web.
- The framework is open source, with a permissive BSD license,
- Has a thriving ecosystem of third-party packages that supplement the core library functionality.

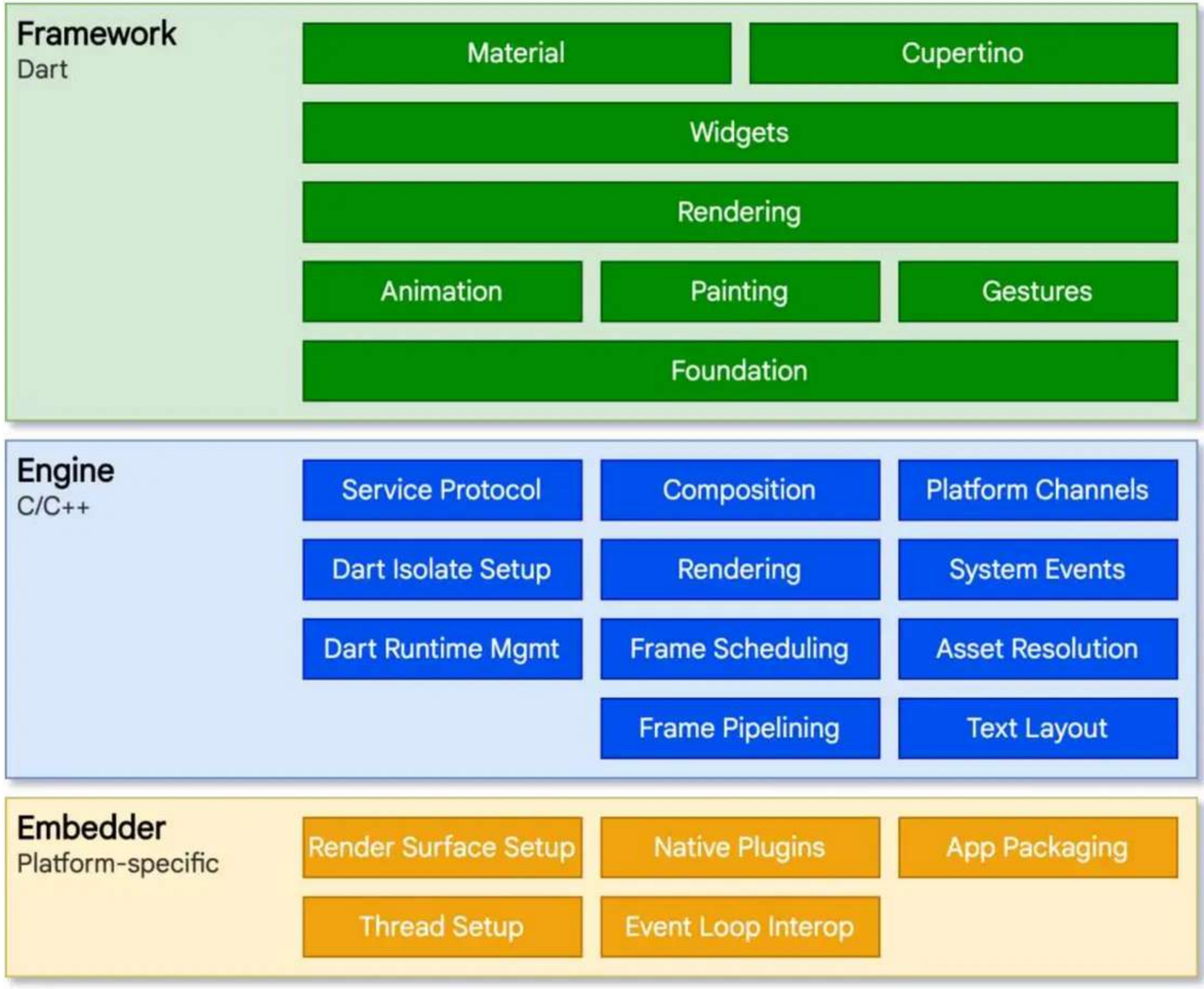
Flutter architectural overview

This overview is divided into a number of sections:

1. The **layer model**: The pieces from which Flutter is constructed.
2. **Reactive user interfaces**: A core concept for Flutter user interface development.
3. An introduction to **widgets**: The fundamental building blocks of Flutter user interfaces.
4. The **rendering process**: How Flutter turns UI code into pixels.
5. An overview of the **platform embedders**: The code that lets mobile and desktop OSes execute Flutter apps.
6. **Integrating Flutter with other code**: Information about different techniques available to Flutter apps.
7. **Support for the web**: Concluding remarks about the characteristics of Flutter in a browser environment.

1. Architectural layers

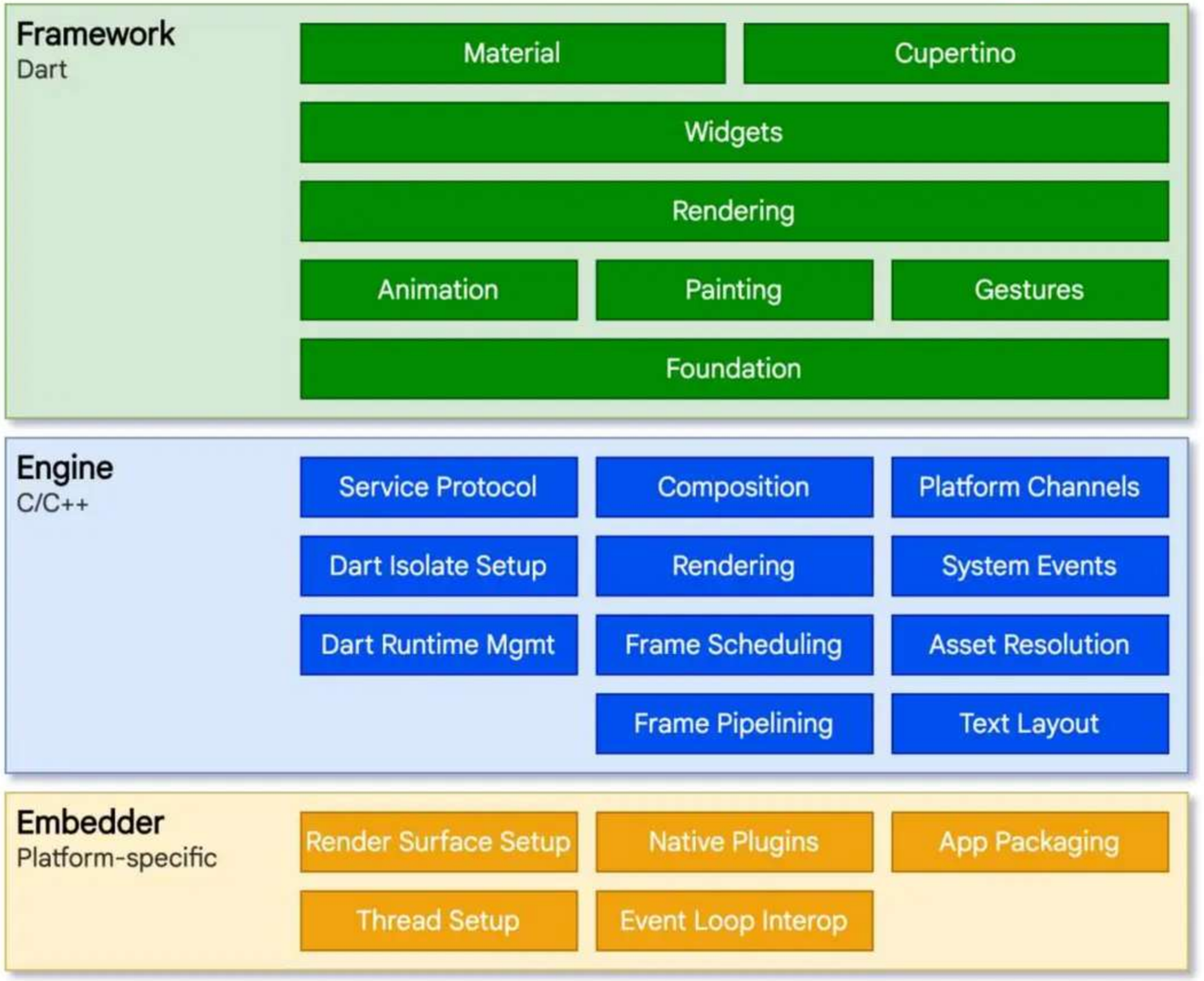
- Designed as an extensible, layered system.
- Exists as a series of independent libraries that each depend on the underlying layer.
- No layer has privileged access to the layer below
- Every part of the framework level is designed to be optional and replaceable.



1. Architectural layers - Embedder

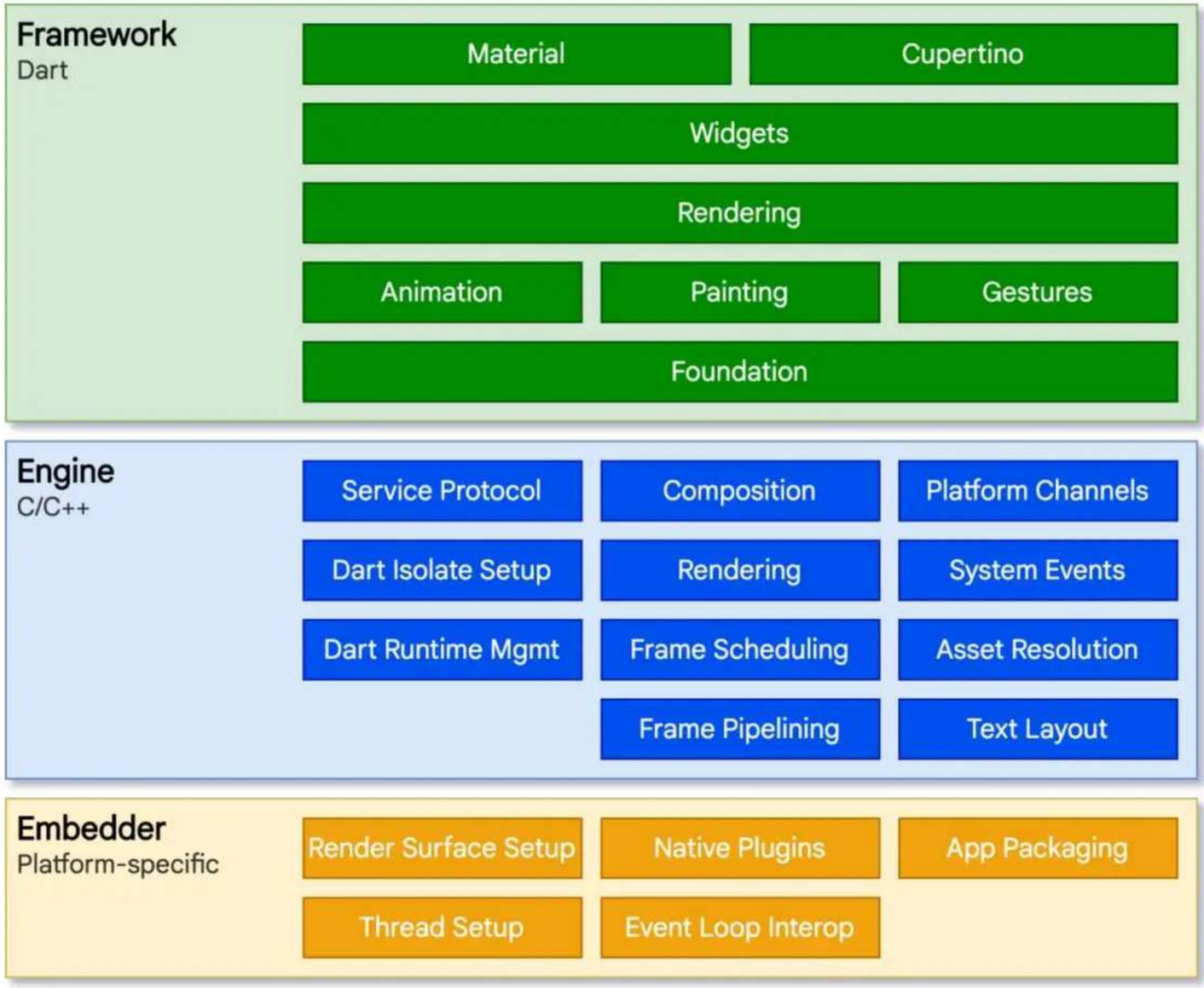
- To the underlying OS, Flutter applications are packaged in the same way as any other native application.

- A platform-specific embedder provides an entry point; coordinates with the underlying operating system for access to services like
  - rendering surfaces,
  - accessibility, and
  - Input
  - manages the message event loop.
  - The embedder is written in a language that is appropriate for the platform:
    - Java and C++ for Android,
    - Objective-C/Objective-C++ for iOS and macOS,
    - C++ for Windows and Linux.



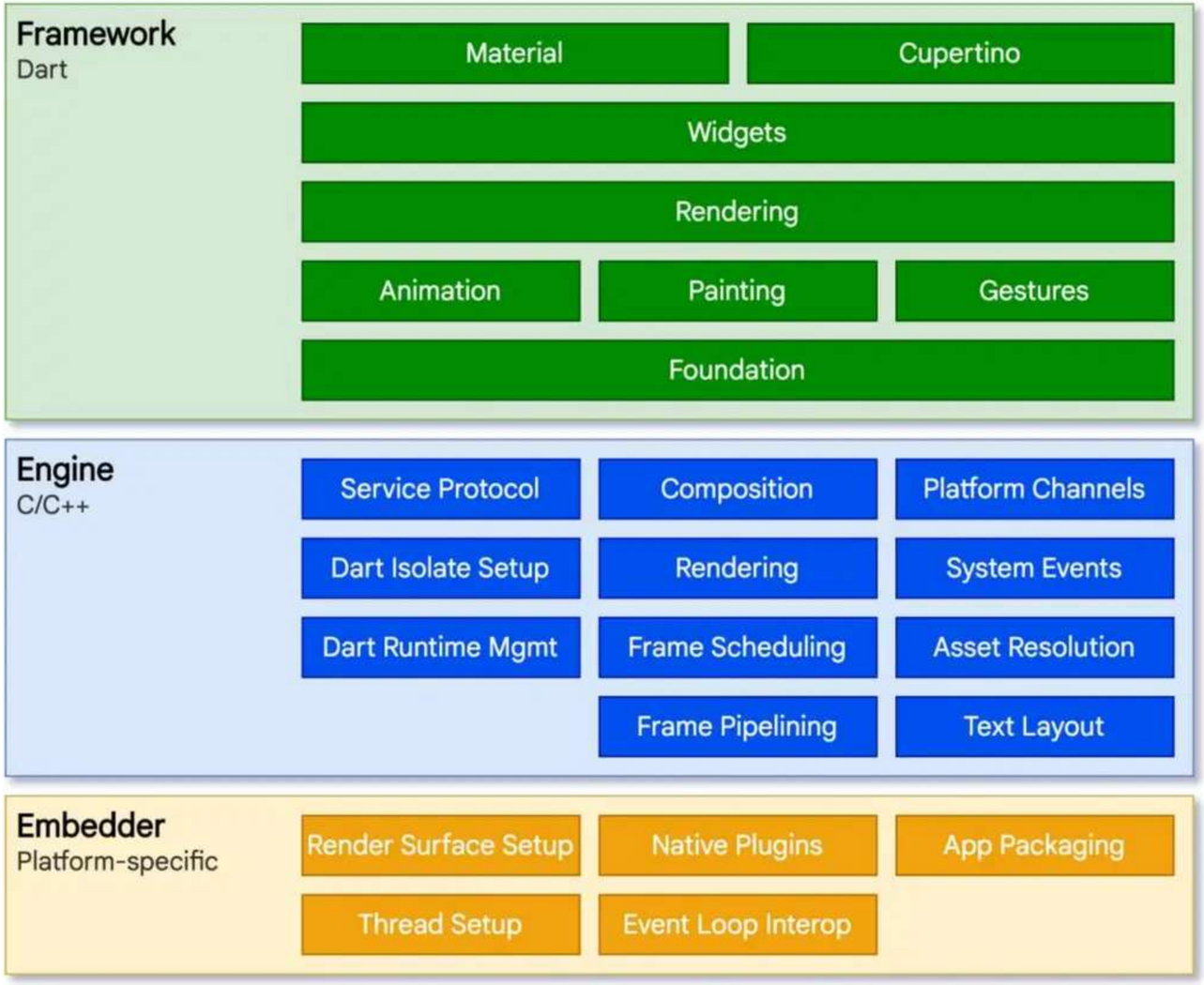
1. Architectural layers - Embedder

- Using the **embedder**,
  - Flutter code can be integrated into an existing application as a module,
  - Or the code may be the entire content of the application.
- Flutter includes a number of embedders for common target platforms, but other embedders also exist.



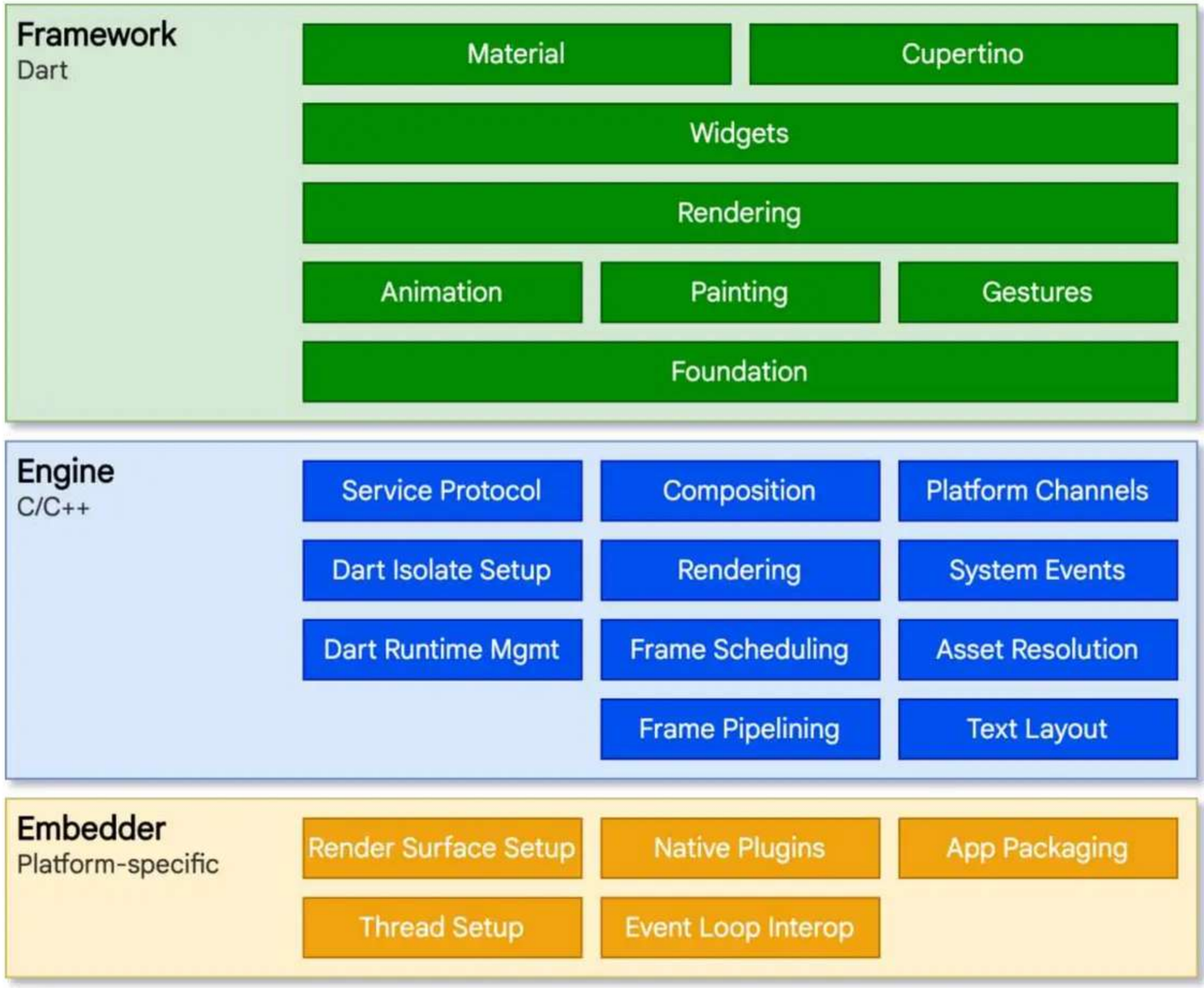
1. Architectural layers – Flutter Engine

- At the core of Flutter is the **Flutter engine, m**ostly written in C++
- Supports the primitives necessary to support all Flutter applications.
- Responsible for rasterizing composited scenes whenever a new frame needs to be painted.
- Provides the low-level implementation of Flutter's core API, including:
  - Graphics (through Skia),
  - Text layout, File and network I/O,
  - Accessibility support,
  - Plugin architecture, and
  - A Dart runtime and compile toolchain.



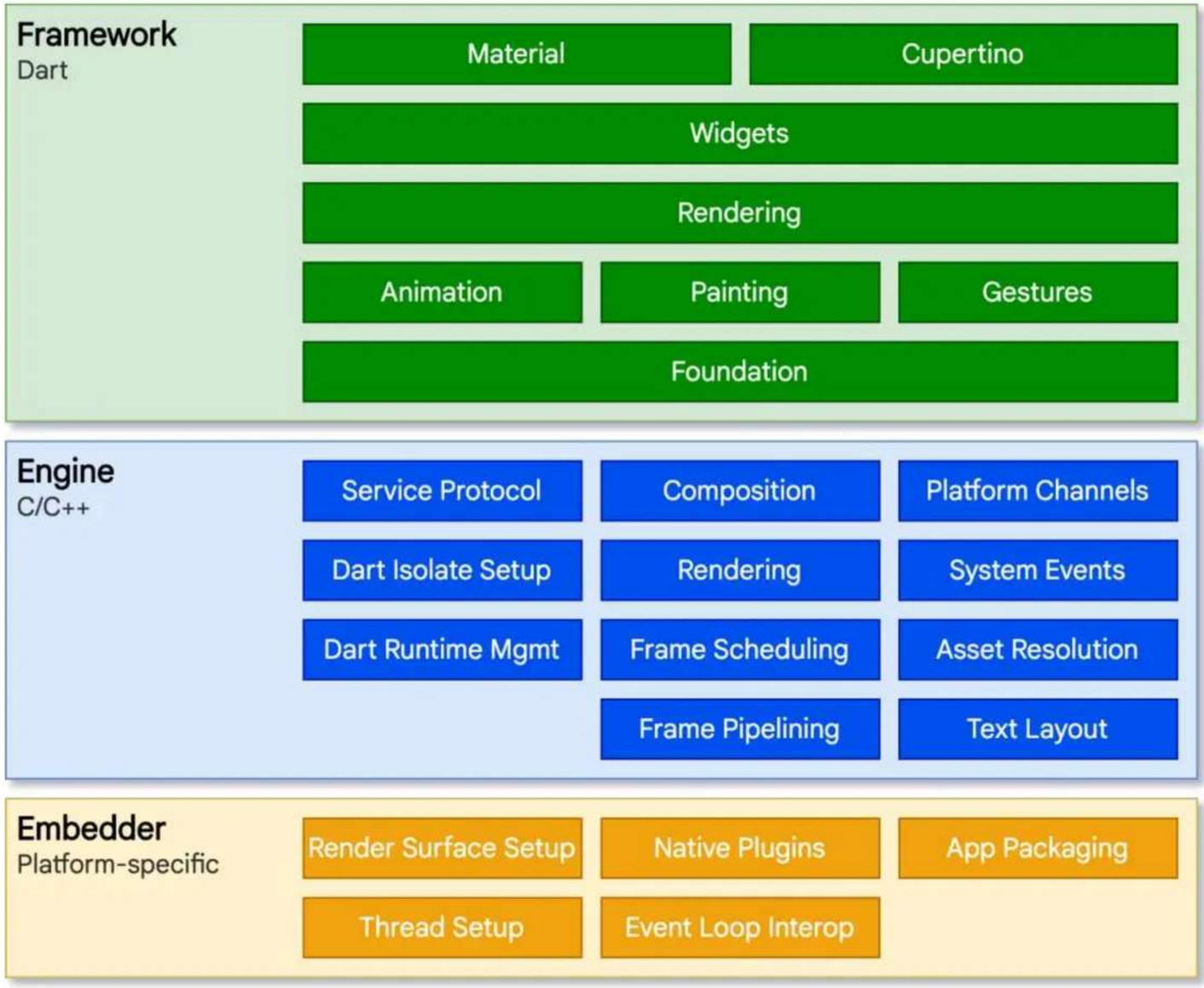
1. Architectural layers – Flutter Engine

- The **engine** is exposed to the Flutter framework through dart:ui
- **dart:ui** wraps the underlying C++ code in Dart classes.
- This library exposes the lowest-level primitives, such as:
  - Classes for driving input
  - Graphics
  - Text rendering subsystems



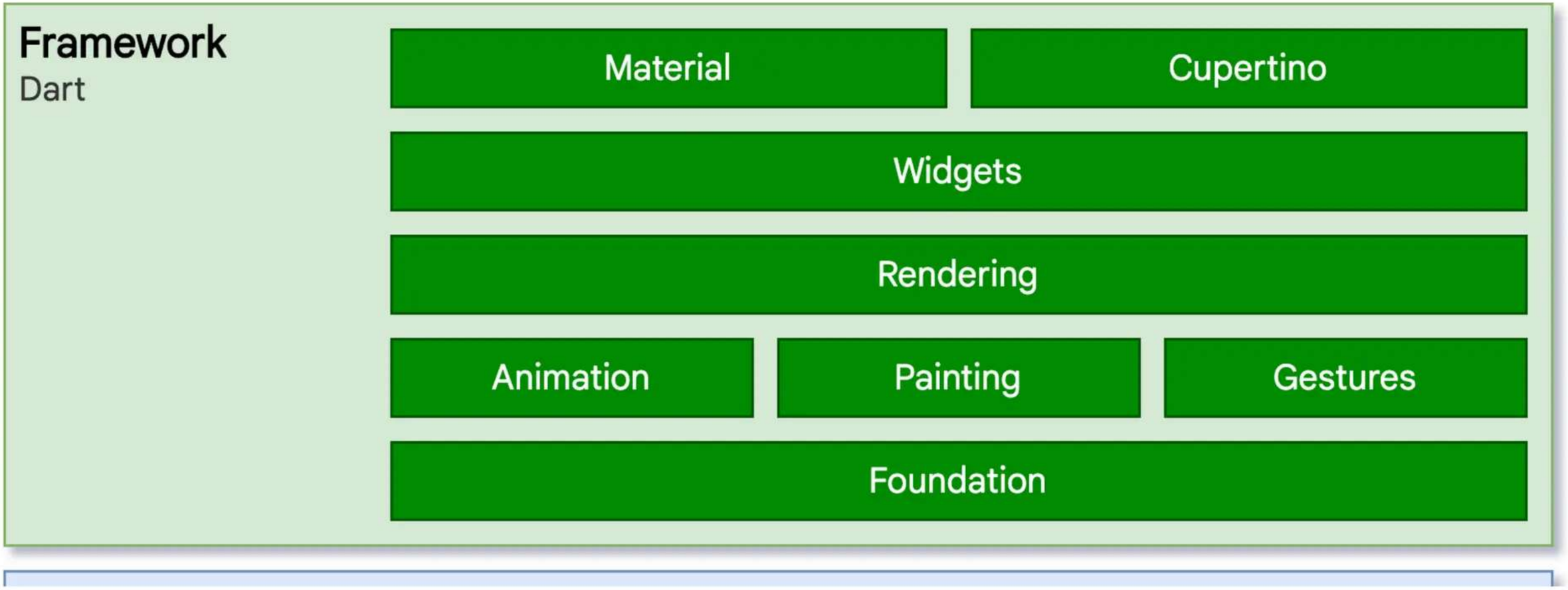
1. Architectural layers – Flutter Framework

- Developers interact with Flutter through the **Flutter framework**,
- Provides a modern, reactive framework written in the Dart language.
- Includes a rich set of platform, layout, and foundational libraries, composed of a series of layers.



1. Architectural layers – Flutter Framework

- Working from the bottom to the top, we have:



1. Architectural layers – Conclusion

- Flutter framework is relatively small

- Many higher-level features are implemented as packages, including
  - Platform plugins like camera and webview,
  - Platform-agnostic features like characters, http, and animations that build upon the core Dart and Flutter libraries.

- Some of these packages come from the broader ecosystem, covering services like
  - in-app payments,
  - Apple authentication, and
  - Animations.

Q&A

Thank you