# Flutter Framework

Dr. Harshad Prajapati
29 July 2023

## What is Flutter?

- Flutter is Mobile UI framework for creating native apps for Android and iOS.
- Using Flutter framework, we do not need to write separate code for Android and iOS.
    - Single code base written in dart can work for both the platforms.

Flutter is a
Framework

Dart is a
Programming Language

# Overview of Flutter Architecture

## Overview of Flutter Architecture

- Flutter is cross-platform UI toolkit.
  - A single codebase can create app for both OSes
    - iOS,
    - Android.
- Flutter is open source.

## Overview of Flutter Architecture

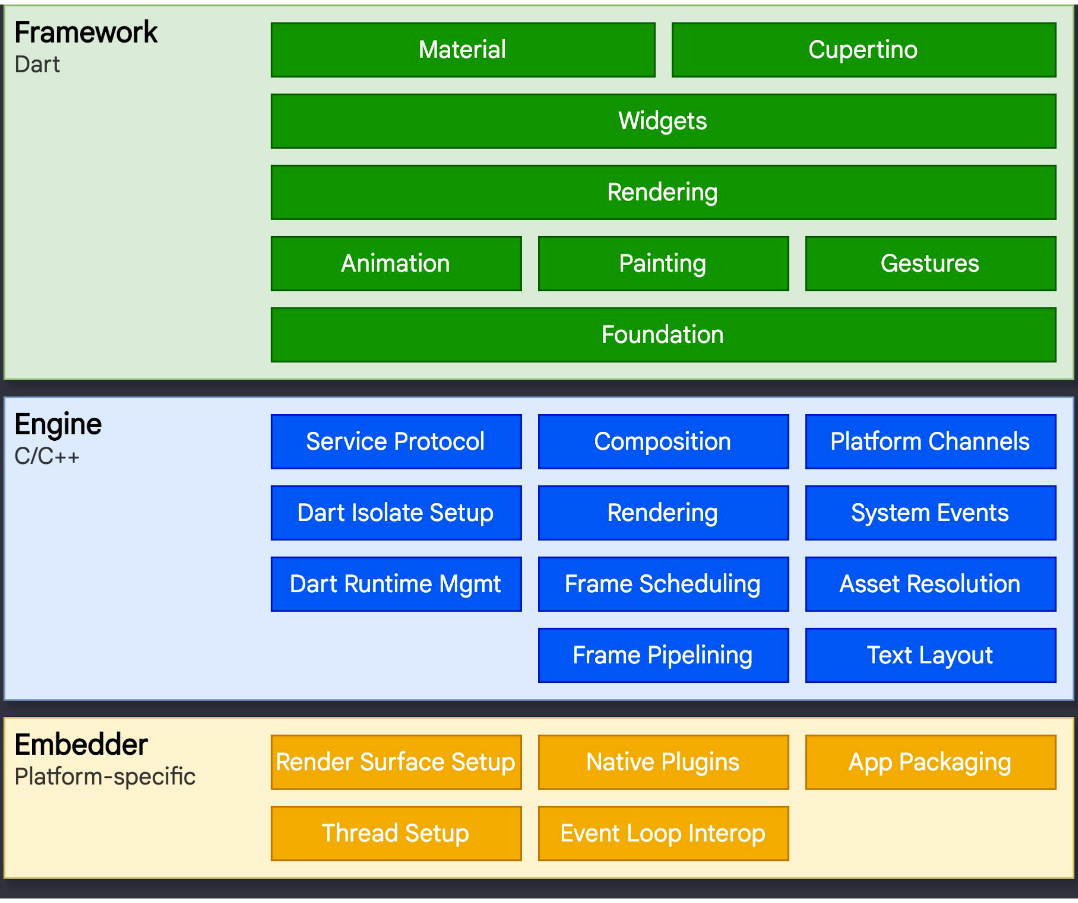
- Running Flutter Apps
  - During development: apps run in a VM with stateful hot reloads, without needing a full compile.
  - For release: apps are compiled directly to machine code:
    - Intel x64,
    - ARM,
    - JavaScript (if targeting the web)

## Overview of Flutter Architecture

- Layered model
- Reactive user interfaces
- Widgets
- Rendering process
- Platform embedders

# Architectural Layers

- Flutter is designed as an extensible, layered system.
- Flutter has a series of independent libraries.
  - Each depend on the underlying layer.
- Layers of Flutter from top to bottom are as follows:
  - Flutter Framework (Used using Dart Language)
  - Flutter Engine (Written in C/C++)
  - Embedder (Platform Specific)



**Framework**
Dart

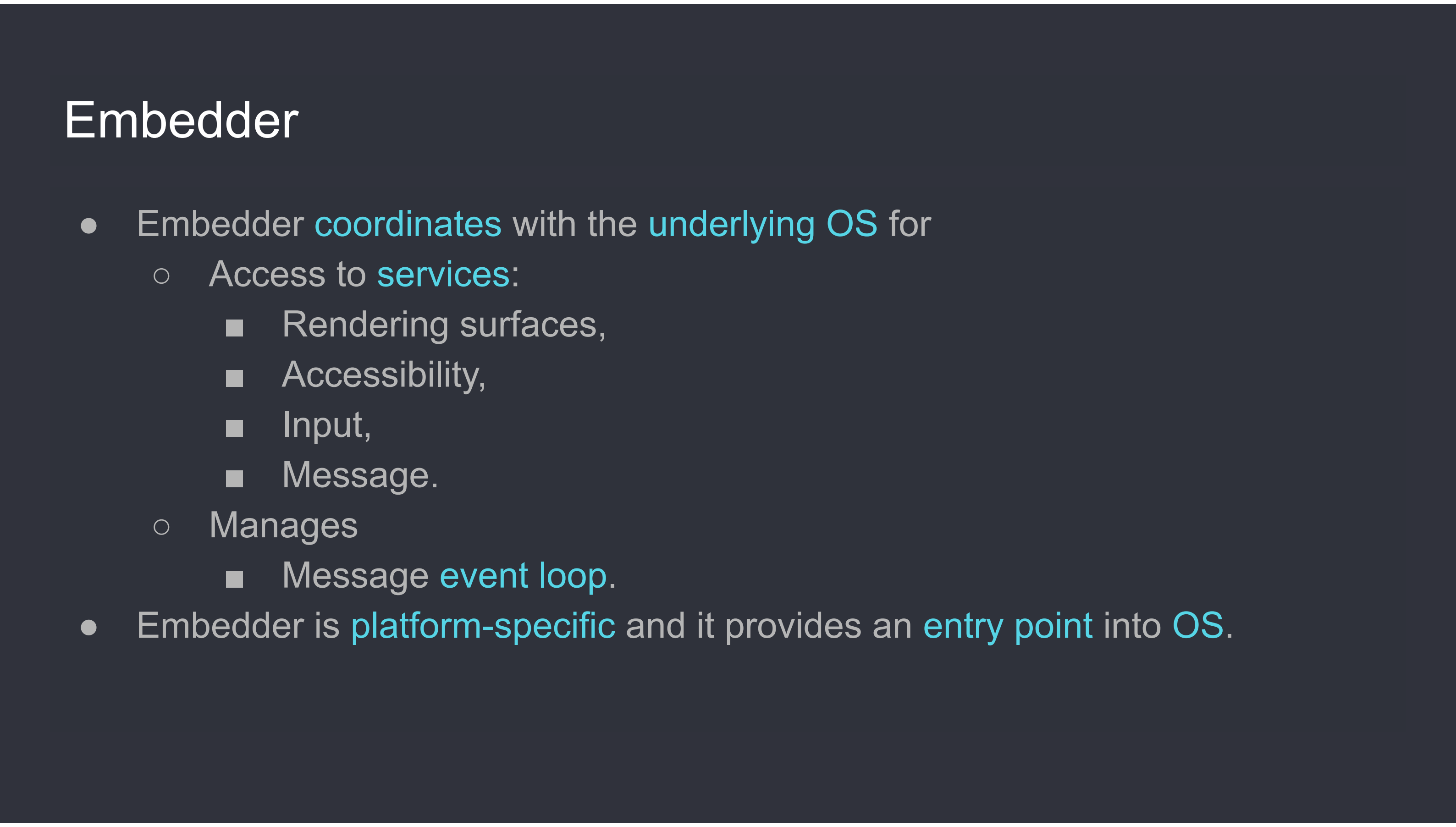
Material | Cupertino
Widgets
Rendering
Animation | Painting | Gestures
Foundation

**Engine**
C/C++

Service Protocol | Composition | Platform Channels
Dart Isolate Setup | Rendering | System Events
Dart Runtime Mgmt | Frame Scheduling | Asset Resolution
Frame Pipelining | Text Layout

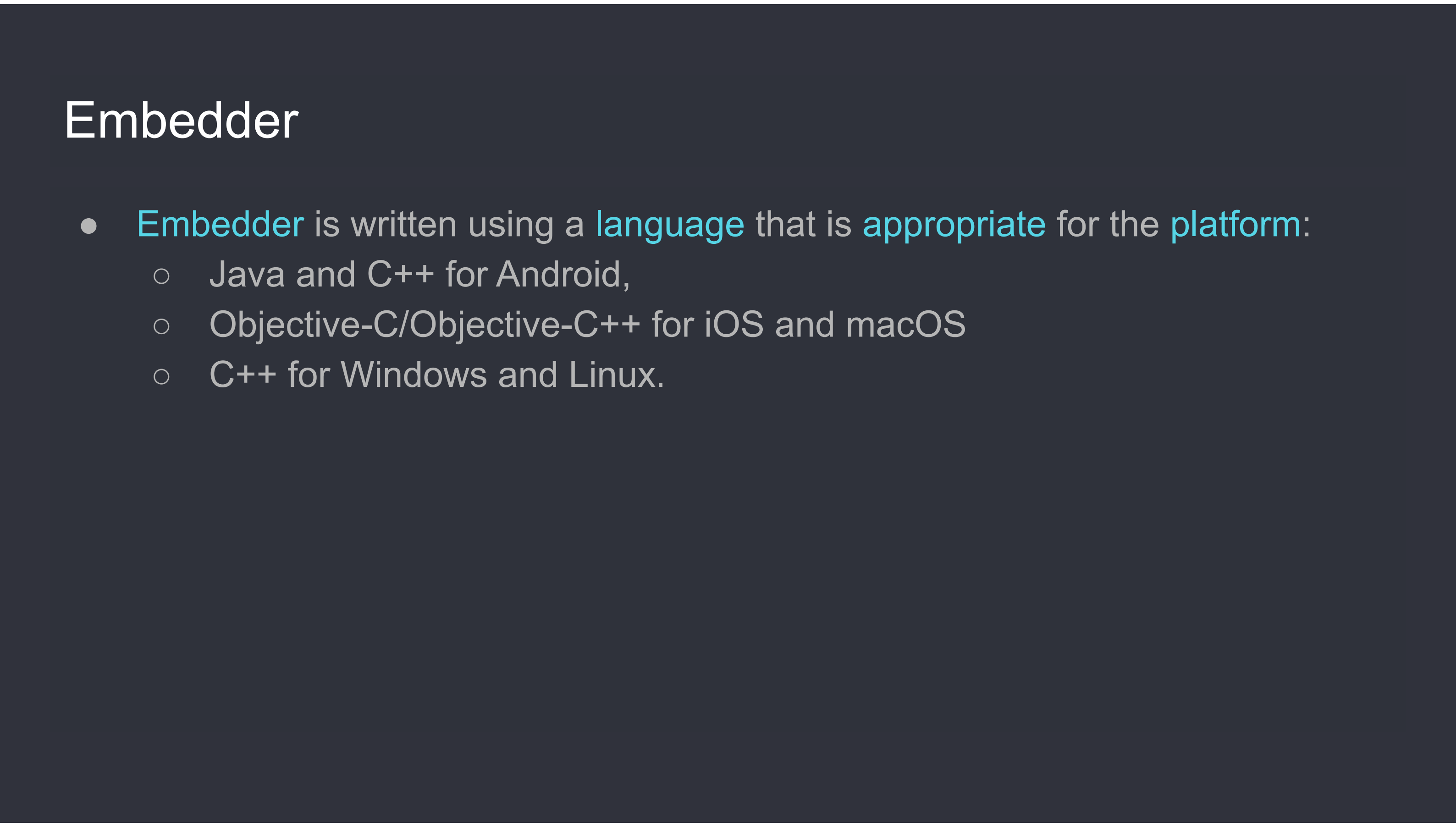
**Embedder**
Platform-specific

Render Surface Setup | Native Plugins | App Packaging
Thread Setup | Event Loop Interop



## Embedder

- Embedder coordinates with the underlying OS for
  - Access to services:
    - Rendering surfaces,
    - Accessibility,
    - Input,
    - Message.
  - Manages
    - Message event loop.
- Embedder is platform-specific and it provides an entry point into OS.



## Embedder

- Embedder is written using 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.

# Flutter Engine

- Flutter engine is mostly written in C++.
- The engine is responsible for rasterizing composited scenes whenever a new frame is to be painted.
- Flutter Engine provides the low-level implementation of Flutter's Core API:
  - Graphics,
  - Text based layout,
  - File I/O,
  - Network I/O,
  - Accessibility support,
  - Plugin architecture,
  - Dart runtime, and
  - Compile toolchain.

# Flutter Engine

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

## Flutter Framework

- Developers interact with Flutter through the Flutter Framework.
  - Flutter framework is modern and reactive framework.
  - It is written in the Dart language.
- Flutter framework includes a series of layers/libraries, which from top to bottom are as follows:
  - Material or Cupertino libraries,
  - Widgets layer,
  - Rendering layer,
  - Foundational classes.

## Flutter Framework

- Material or Cupertino libraries:
  - These libraries offer comprehensive set of controls that use the widget layer’s composition primitives to implement Material or iOS design languages.
- Widgets layer:
  - Each class in widget layer maps to a render object in the underlying layer, rendering layer.
  - In this layer, the reactive programming model is introduced.

# Flutter Framework

- Rendering layer:
  - Rendering layer provides an abstraction for dealing with layout.
  - We can build a tree of renderable objects.
  - We can manipulate these objects dynamically.
  - Tree automatically updates the layout to reflect our changes.
- Foundational classes and building block services:
  - Animation
  - Painting
  - Gestures

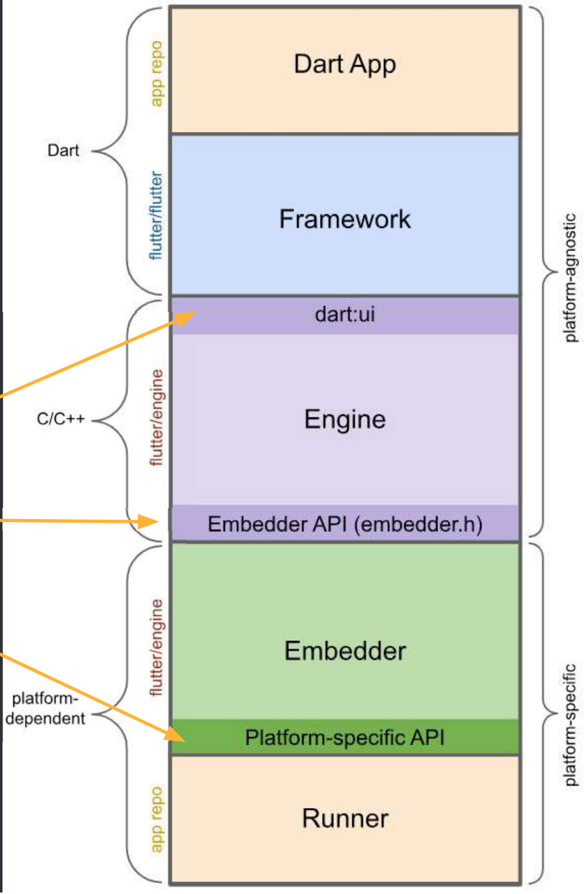
# Flutter Framework

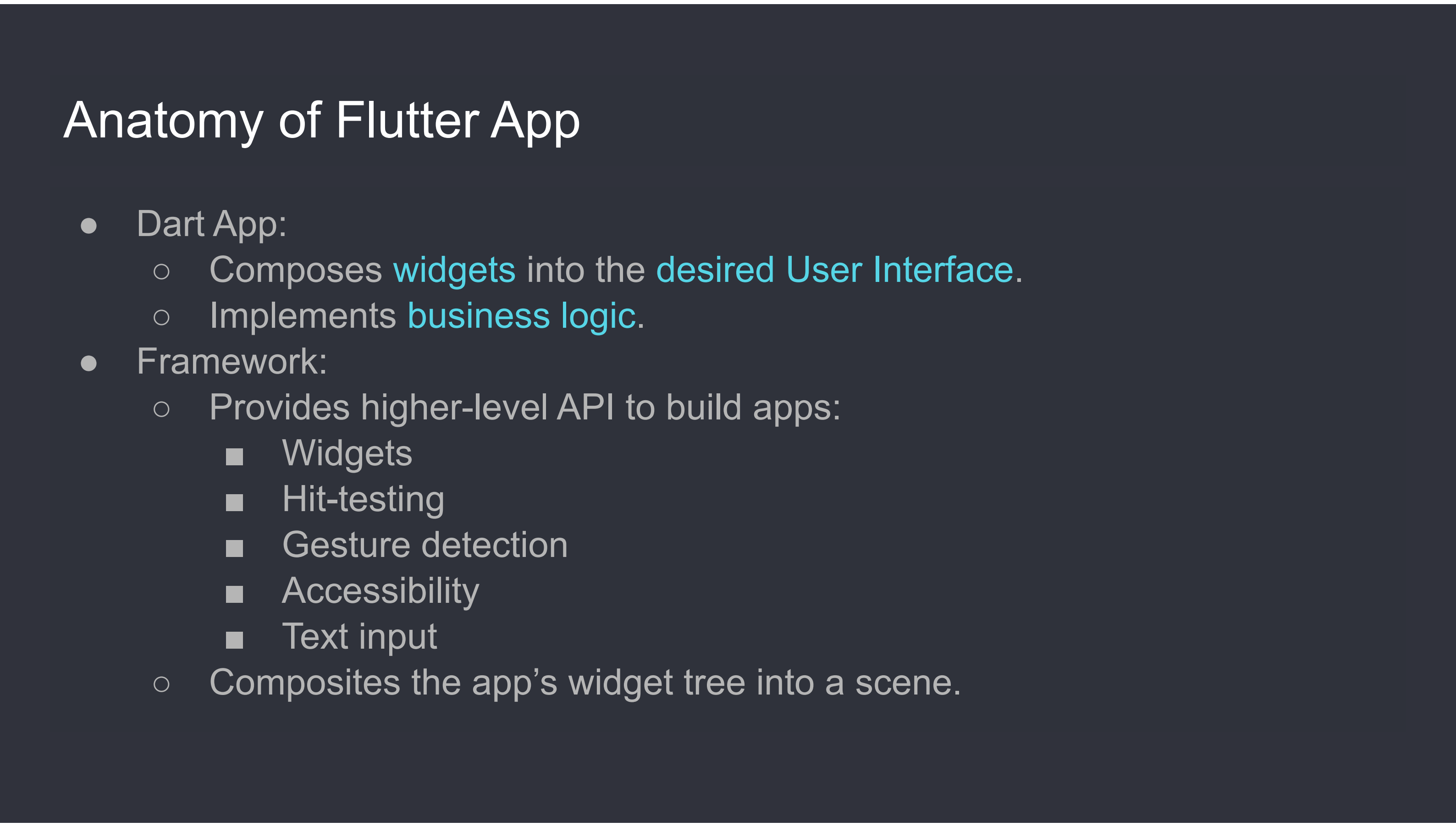
- Flutter framework is relatively small.
- Many higher-level features are implemented as packages.
  - Platform plugins like camera and webview.
  - Platform-agnostic features are built upon core Dart and Flutter libraries:
    - Characters
    - Http
    - Animations

# Anatomy of Flutter App

## Anatomy of Flutter App

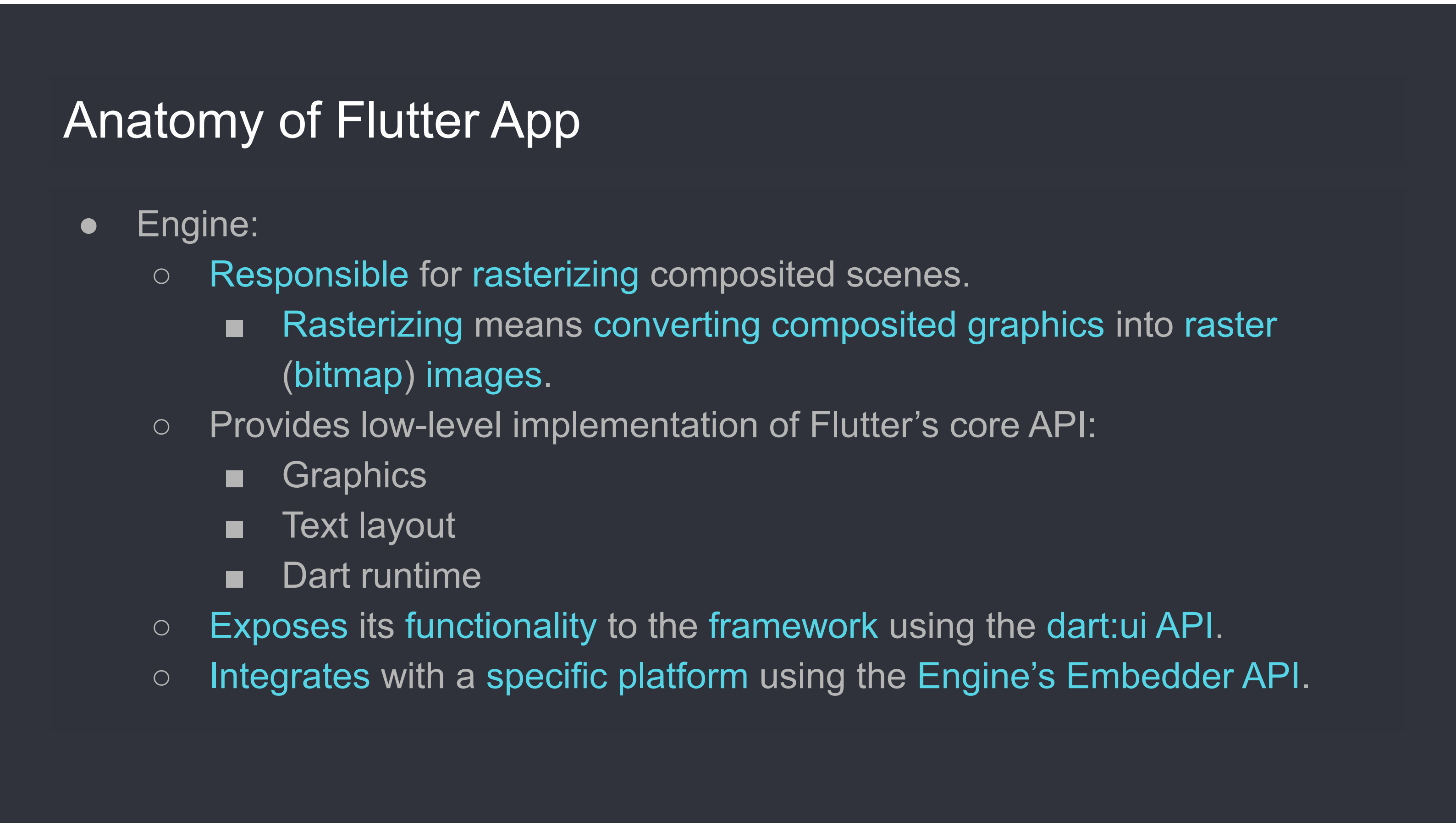
- This diagram shows various pieces that make up a regular Flutter app.
    - Flutter app is generated by flutter create command.
- This diagram
    - highlights API boundaries.
    - shows where the Flutter Engine sits.





## Anatomy of Flutter App

- Dart App:
  - Composes widgets into the desired User Interface.
  - Implements business logic.
- Framework:
  - Provides higher-level API to build apps:
    - Widgets
    - Hit-testing
    - Gesture detection
    - Accessibility
    - Text input
  - Composites the app’s widget tree into a scene.



## Anatomy of Flutter App

- Engine:
  - Responsible for rasterizing composited scenes.
    - Rasterizing means converting composited graphics into raster (bitmap) images.
  - Provides low-level implementation of Flutter’s core API:
    - Graphics
    - Text layout
    - Dart runtime
  - Exposes its functionality to the framework using the dart:ui API.
  - Integrates with a specific platform using the Engine’s Embedder API.

# Anatomy of Flutter App

- Embedder:
  - Coordinates with the underlying OS for access to services
    - Rendering surfaces,
    - Accessibility, and
    - Input.
  - Manages the event loop.
  - Exposes platform-specific API to integrate the Embedder into apps.

# Anatomy of Flutter App

- Runner:
  - Composes the pieces exposed by the platform-specific API of the Embedder into an app package runnable on the target platform.
  - It is part of app template generated by flutter create.

## Reactive User Interfaces

- Flutter is a reactive, declarative UI framework:
  - Developers provide mapping from application state to interface state.
  - Framework takes care of updating UI at runtime when application state changes.
- Flutter's approach of updating UI is similar to React (Virtual and Real DOMs)
  - In Flutter, widgets (like components in React) are represented by immutable classes that configure a tree of objects.
  - Flutter efficiently handles and updates modified parts of trees.

## Reactive User Interfaces

- A widget declares its UI by overriding the build() method. (React has render())
  - The build() is a function that converts state to UI.
  - UI = f(state)
- The build() method is fast to execute and free from side effects.
  - The build() method is called by the framework as often as once per rendered frame.

## Widgets and Hierarchy

- Widget is a unit of composition and is a building block of a UI in Flutter’s app.
- Each widget is an immutable declaration of part of the user interface.
- Widgets form hierarchy, from children to the root widget (container that hosts the Flutter app)
    - Each widget nests inside its parent.
    - Each widget can receive context from the parent.

## Widgets and Updating UI

- Apps update their UI in response to events (such as user interaction).
    - Apps tell the framework to replace a widget in the hierarchy with another widget.
    - The framework compares the new widgets and old widgets and efficiently updates the UI.
- Flutter has its own implementation of UI controls, separate for iOS and Android.

## Widget State

- The framework has two major classes of widget:
  - Stateful widgets and
  - Stateless widgets.

## Widget State

- Stateless widgets:
- Components that do not have mutable state (changes over time).
- Example: icon or label.
- These widgets are subclassed from StatelessWidget.

## Widget State

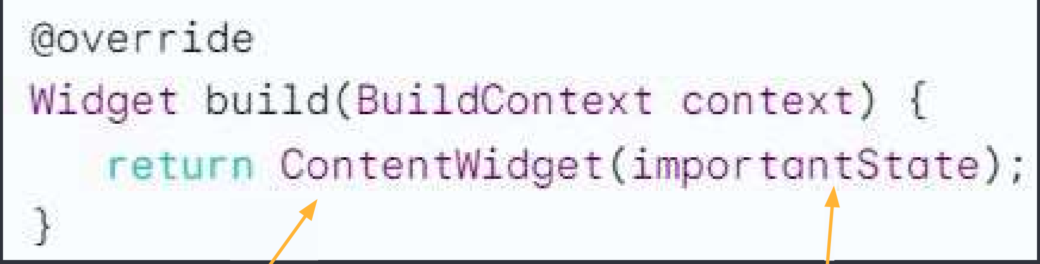
- Stateful widgets:
- The characteristics of a widget change based on user interaction or other interactions.
- For example a widget having counter that change on say tapping a button.
- When the value changes, the widget needs to be rebuilt to update its part of UI.
- These widgets subclass StatefulWidget and do not have build() method.
- As all widgets are immutable, they cannot store state.
- They store their mutable state in a separate class that subclasses State.
  - From this State object, their UI is built instead of using build() method.

## Widget State Update

- Whenever we mutate State object (for example, incrementing or decrementing a counter), we must call setState().
  - Calling setState() informs to the framework that the UI needs to be updated.

## State Management

- Many widgets can contain state.
  - How is state managed and passed around the system?
- We can use a constructor in a widget to initialize its data.
- The build() method can ensure that any child widget gets required data.



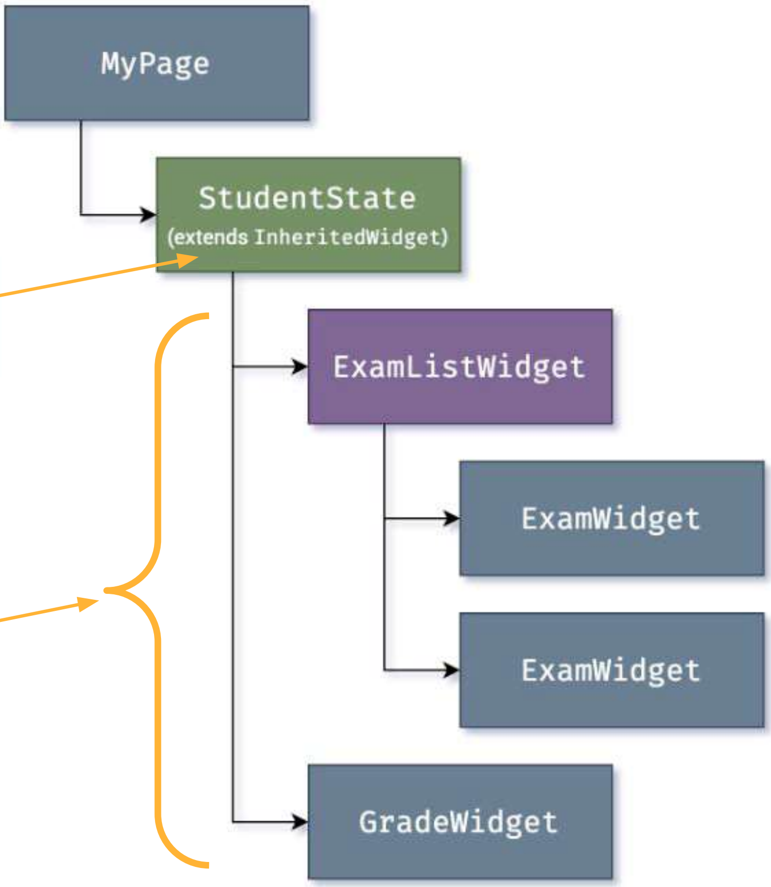
```
@override
Widget build(BuildContext context) {
   return ContentWidget(importantState);
}
```

Calling child widget's constructor.

Passing data.

## State Management

- When a widget trees get deeper, passing state information becomes difficult.
- InheritedWidget (third widget type) provides a way to get data from a shared ancestor.



All child widgets get data from StudentState, subclassed from InheritedWidget.

## State Management

- Whenever any ExamWidget or GradeWidget objects need data from StudentState, it can access it using
  - final studentState = StudentState.of(context);
- The of(context) method takes the build context as input and returns the nearest ancestor in the tree that matches the StudentState type.
- Flutter itself uses InheritedWidget extensively for shared state.
  - Theme is inserted in the tree and child widgets can access it using of() method.
  - Navigator also uses for page routing.
  - MediaQuery also uses it to provide access of screen metrics.
- Flutter apps also use utility packages like provider, wrapper on InheritedWidget.

## Rendering and Layout

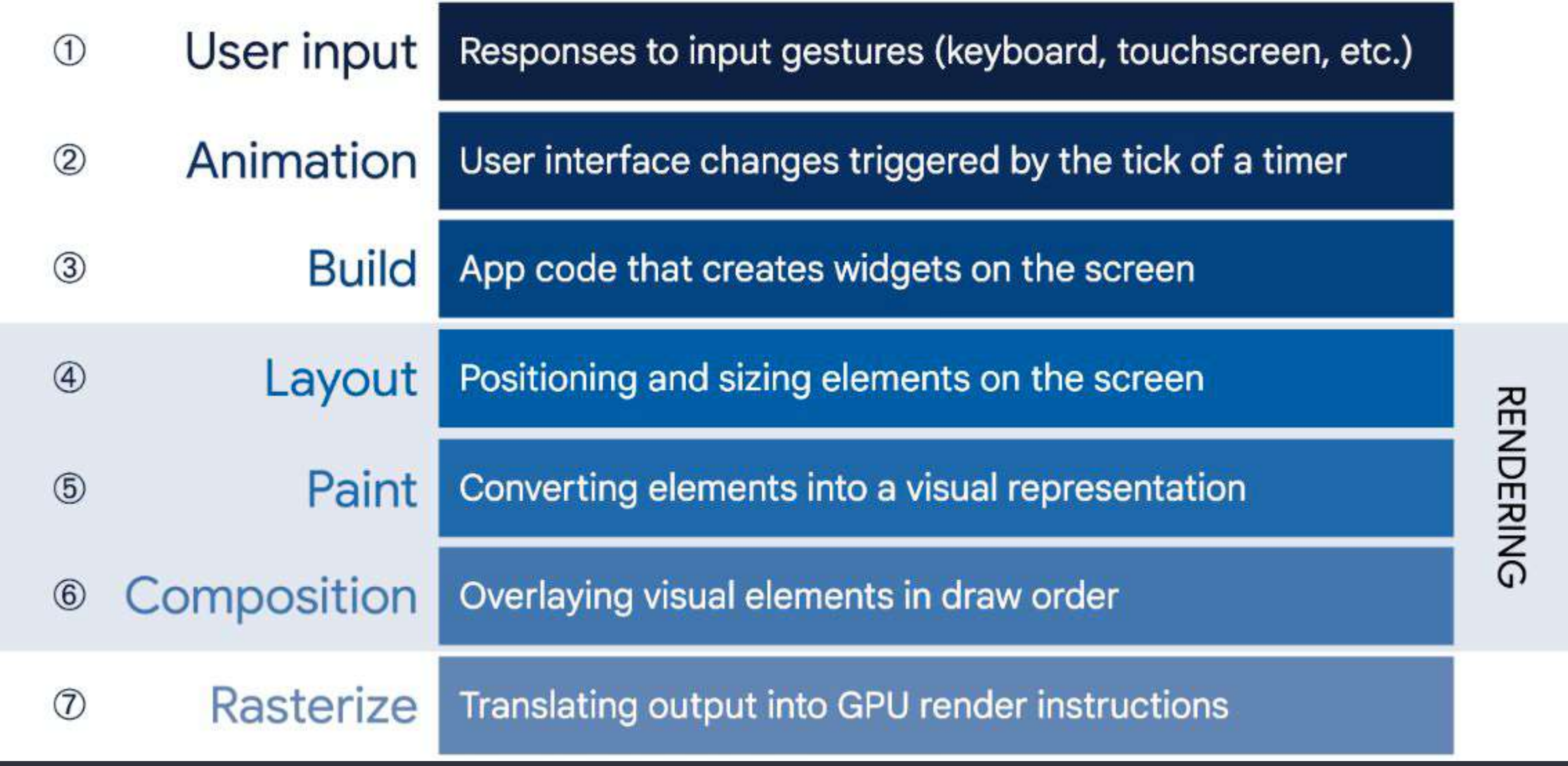
- In rendering pipeline, Flutter takes a series of steps to convert a hierarchy of widgets into the actual pixels that we see on the screen.
- Rendering model of Android:
  - Java code of Android framework is called.
  - Android system libraries provide components that draw themselves on a Canvas object.
  - Android render Canvas object with Skia (graphics engine written in C/C++)

# Rendering and Layout

- Rendering model of (JavaScript based) Cross-platform frameworks:
  - Have abstraction layer over the underlying native Android or iOS UI libraries.
  - App code is written in an interpreted language like JavaScript.
  - Code in interpreted language interact with Java-based Android or Objective-C based iOS system libraries to display UI.
  - This model adds overhead when there is a lot of interaction between the UI and the app logic.

# Rendering and Layout

- Rendering model of Flutter:
  - Flutter minimizes abstractions as compared to other cross-platform frameworks.
  - Flutter bypasses the system UI widget libraries of the platform in favor of its own widget set.
  - The Flutter’s visual, created using Dart code, gets compiled into native code.
  - This native code is rendered using native engine. (Skia for Android and Impeller for iOS)



① User input — Responses to input gestures (keyboard, touchscreen, etc.)

② Animation — User interface changes triggered by the tick of a timer

③ Build — App code that creates widgets on the screen

④ Layout — Positioning and sizing elements on the screen

⑤ Paint — Converting elements into a visual representation

⑥ Composition — Overlaying visual elements in draw order

⑦ Rasterize — Translating output into GPU render instructions

## References

- https://docs.flutter.dev/resources/architectural-overview
- https://docs.flutter.dev/