Flutter Architecture Overview

Flutter follows a **layered architecture** where each layer has a specific responsibility. Together, they make Flutter fast, flexible, and truly cross-platform.

CORE CONCEPT

How Flutter Works

1

You Write Dart Code

You build the app using Dart and Flutter's declarative UI.

UI is written using widgets (everything is a widget).

2

Flutter Draws Everything

Flutter does not use native UI components.

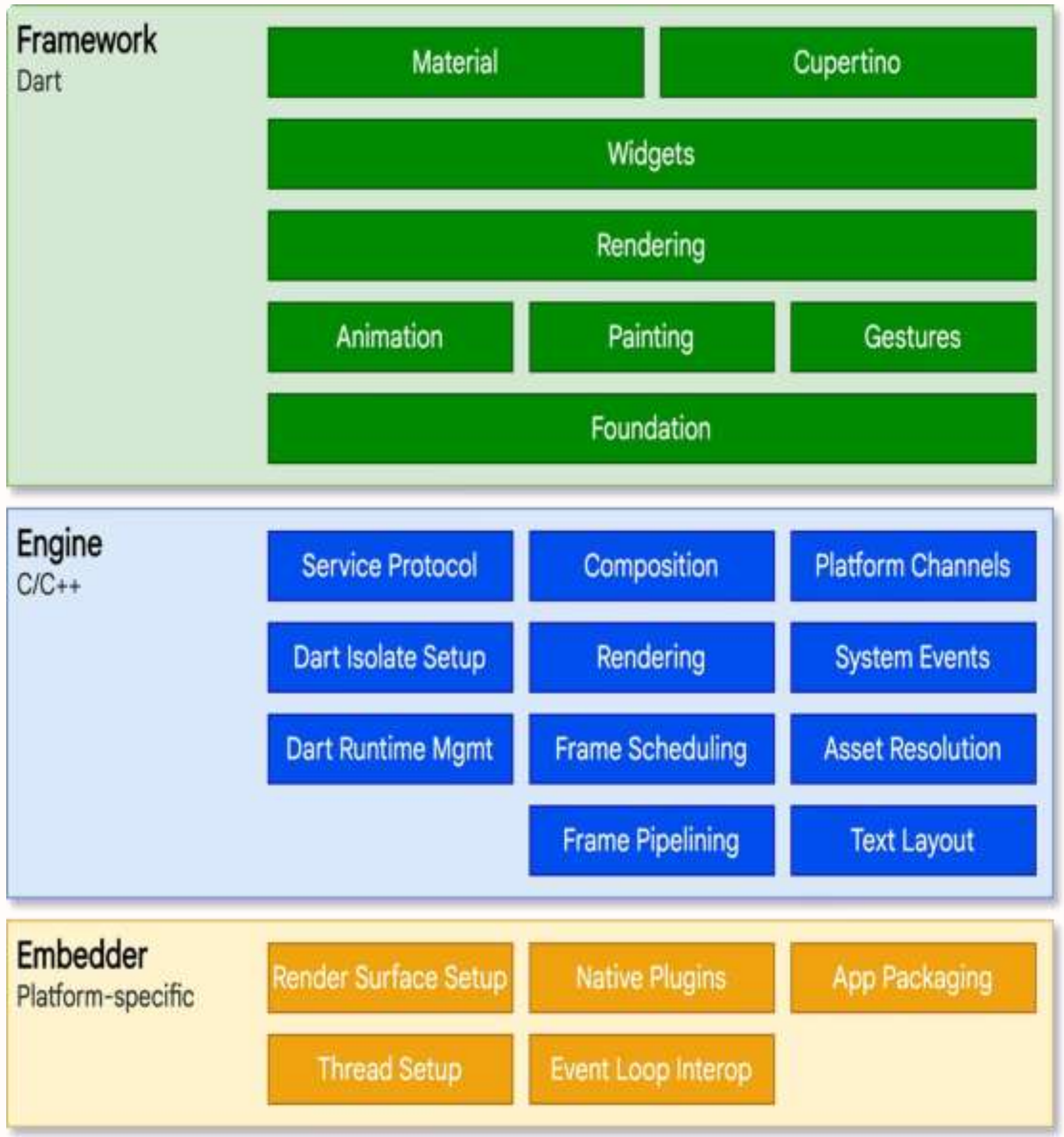
It draws every pixel itself using its own graphics engine (Skia).

This makes the UI fast, consistent, and identical on Android & iOS.

3

Engine Talks to OS

Flutter's C++ Engine communicates with the Android / iOS system.
It accesses: Camera, Sensors, File system, Network, Platform APIs
This happens via platform channels.



• LAYER 1

Dart Programming Language

What It Is

- The **language used to write Flutter apps**
- Created by Google specifically for UI development
- Object-oriented and class-based, similar to Java & JavaScript

Why Dart?

- **AOT compilation** → Fast production apps
- **JIT compilation** → Instant hot reload during development
- Familiar syntax for developers from C/Java backgrounds

Critical Foundation: All UI, business logic, and state management is written in Dart. Without Dart, Flutter cannot exist.

• LAYER 1

Flutter Framework: The Widget Layer

1.

Everything is a Widget

Buttons, text, images, padding, and layouts are all widgets that compose together

2.

Stateless Widget

For UI components that don't change after creation

3.

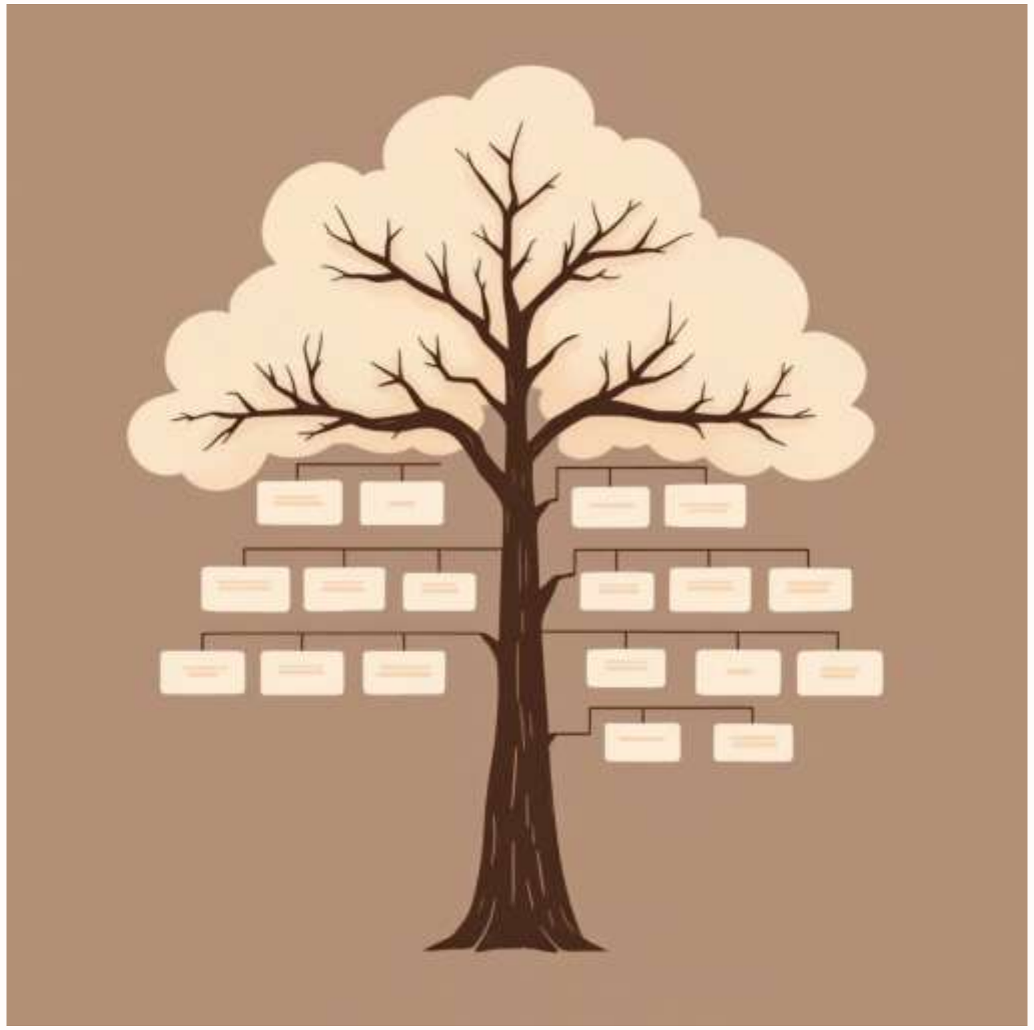
StatefulWidget

For UI components that change dynamically over time

The Widget Tree

Flutter builds UI as a **tree of widgets** with parent-child relationships. This compositional approach makes Flutter incredibly flexible and powerful.

Unlike Android's XML or iOS's Storyboards, Flutter UI is written **completely in Dart code**.



LAYER 1

Flutter Framework

These libraries provide **core services** that power Flutter apps behind the scenes.

1. Foundation Libraries – Core Services & Utilities

Base support system of Flutter

Provide low-level services required by the framework

Handle app lifecycle (start, pause, resume)

Manage frame scheduling

Provide state change notification

Enable communication with native OS

Offer debugging & diagnostics tools

Examples:

ChangeNotifier, Key, BuildContext, SchedulerBinding

2. Animation Library – Animations

Controls how things move on screen

Provides smooth & high-performance animations

Supports:

Implicit animations

Explicit animations

Allows fine-grained control over motion

Used for:

Fade, scale, rotate, slide effects

3. Gestures Library – Touch Handling

Handles user interaction

Detects: Taps, Double taps, Swipes, Drags, Long pressesMulti-touch gestures
Converts raw touch input into meaningful actions

Example widgets:

GestureDetector,

InkWell

4. Rendering Layer – Layout & Painting

Turns widgets into visual structure

Calculates: Size, Position, Layout of UI elements
Converts widget tree into render objects
Prepares data to be drawn as pixels

Uses:

Box model

Constraints-based layout

LAYER 1

Flutter Framework

These libraries provide **core services** that power Flutter apps behind the scenes.

5. Widgets Layer – UI Composition

Heart of Flutter UI

Everything in Flutter is a widget

Widgets describe what UI should look like

Two main types:

- StatelessWidget
- StatefulWidget

Builds a widget tree

Examples:
Text, Row, Column, Container, Scaffold

3. Gestures Library – Touch Handling

Handles user interaction

Detects: Taps, Double taps, Swipes, Drags, Long pressesMulti-touch gestures
Converts raw touch input into meaningful actions

Example widgets:
GestureDetector,
InkWell

6. Material / Cupertino – Design & Theming

Ready-made UI design systems

Material Design (Android-style)

Buttons, AppBar, FloatingActionButton

Follows Google's Material Design

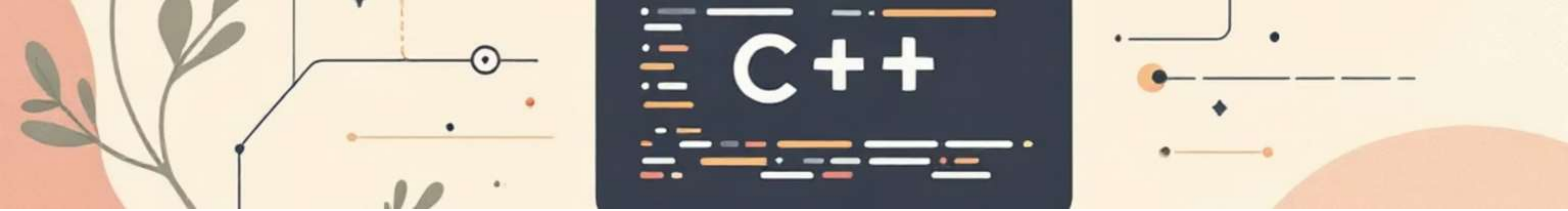
Cupertino (iOS-style)

iOS-looking widgets

Native iOS feel

Supports:
- Themes
- Colors
- Fonts
- Consistent UI

Gives professional look without designing from scratch.



• LAYER 2

Flutter Engine: The C++ Powerhouse

This layer is written in **C++** and works behind the scenes to deliver high performance.

Graphics Rendering

Uses **Skia graphics engine** to draw everything Flutter displays on screen

Core Calculations

- Text rendering and layout
- Animation frame scheduling
- Accessibility tree maintenance

Dart Runtime Management

Manages the Dart VM and handles memory, garbage collection, and isolates

Native Communication

Bridges Flutter code with platform-specific APIs and services

Key Insight: Flutter does NOT use native UI components. It draws **everything itself** using Skia. This is why Flutter UI looks identical on Android, iOS, and all platforms.

• LAYER 3

Platform Embedder: The OS Bridge

This layer serves as the bridge between Flutter and the operating system, enabling true cross-platform functionality.

Platform Support

- Android (Java/Kotlin)
- iOS (Swift/Objective-C)
- Windows, macOS, Linux
- Web (JavaScript)

System Integration

- Touch input and keyboard events
- File system access
- Camera, GPS, and sensors
- Platform channels for native code communication

The embedder layer makes Flutter **truly cross-platform**, allowing a single codebase to run everywhere while maintaining native performance.

DEEP DIVE

Engine Architecture Components

Engine (C/C++)

Core Services

- **Service Protocol:** Debugging and profiling tools
- **Composition:** Layering and blending graphics
- **Platform Channels:** Native code communication
- **Dart Runtime Management:** VM lifecycle control

Rendering Pipeline

- **Frame Scheduling:** Coordinates 60fps rendering
- **Frame Pipelining:** Optimizes render performance
- **Asset Resolution:** Loads images and resources
- **Text Layout:** Advanced typography engine

System Integration

- **Dart Isolate Setup:** Manages concurrent execution
- **System Events:** Handles OS-level events
- **Rendering:** Skia-based graphics

DEEP DIVE

Embedder: Platform Integration

The embedder layer provides platform-specific implementations that make Flutter apps feel native on each operating system.

1

Render Surface Setup

Initializes the drawing canvas where Flutter renders all graphics

2

Thread Setup

Configures UI, GPU, and I/O threads for optimal performance

3

Native Plugins

Provides access to platform-specific APIs like camera, GPS, and sensors

4

Event Loop Interop

Integrates with the OS event system for responsive user interactions

5

App Packaging

Bundles Flutter apps into platform-specific formats (APK, IPA, EXE)

> The embedder is what makes Flutter truly cross-platform—it adapts the same Flutter code to look and feel native on every operating system.