Google's Solution to Cross-Platform Development

Cross-Platform Development Approaches 1

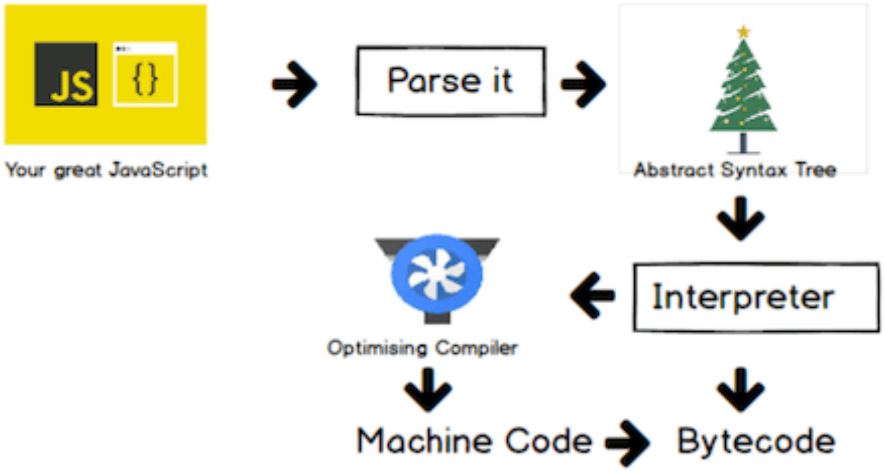
There are two ways to build cross-platform development.

- The first approach is using **web apps**: make a web browser as the platform to run web apps.
    - React Native (Mobile devices)
    - Electron (Visual Stuido Code and many others)

V8 JavaScript Engine

This technology is based on V8 JavaScript engine developed by Google.

- Google introduced this engine when they developed Chrome.
- V8 translates the JavaScript code into machine code using **JIT compiler** technology.



Cross-Platform Development Approaches 2

- The second approach is using **programming languages/compilers** such as Flutter (Dart) or C# (.NET Maui).

Compile-to-native solutions

- **Pros**: Real apps that can be found in the stores and run fast
- **Cons**: Learning a framework may be difficult. Mastering the toolchain definitely is

Skia Graphics Library (Flutter)

Flutter is based on the Skia Graphic Library

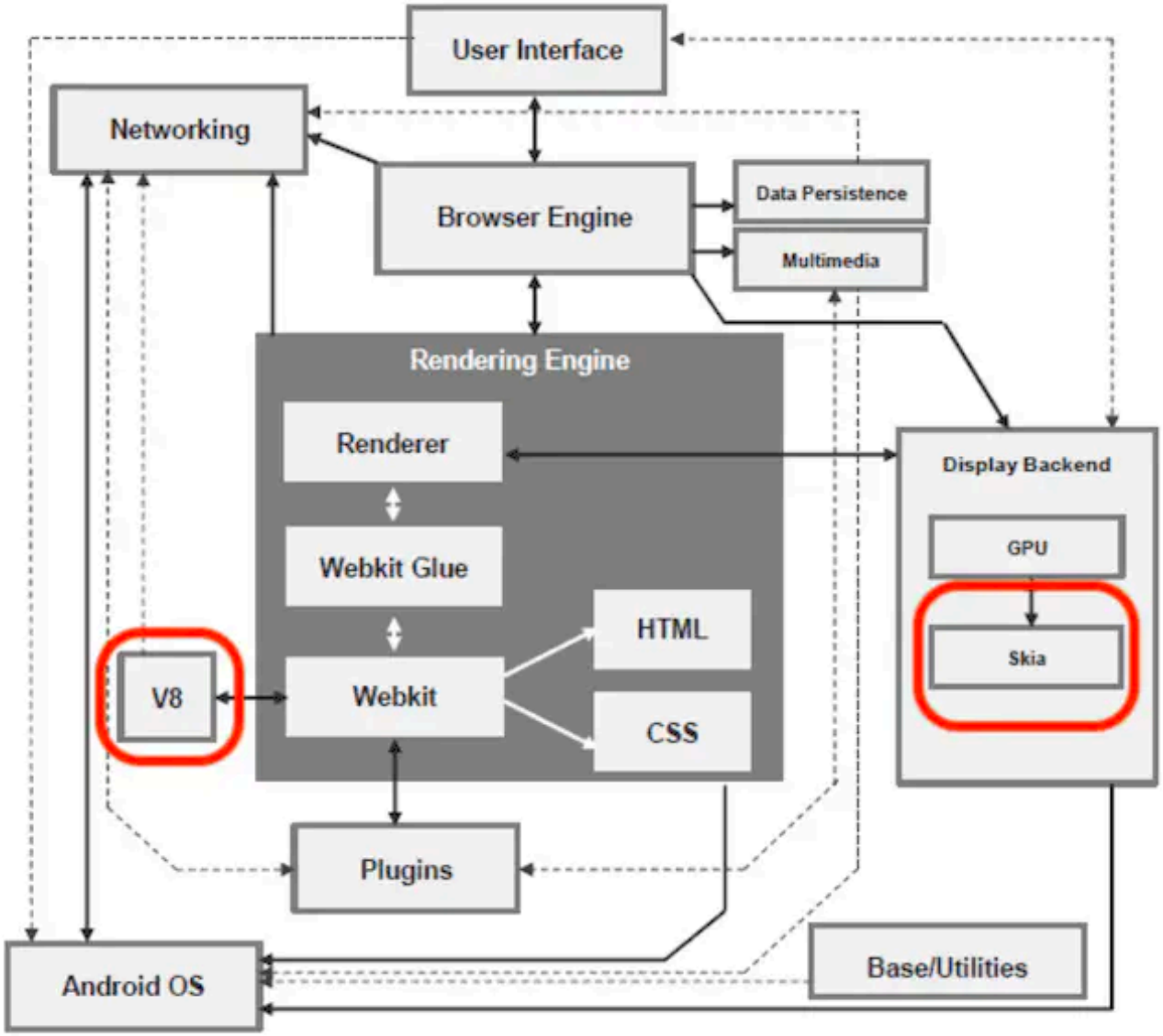
- Google invented **Skia** when they developed Chrome.
- Skia is an open-source **2D graphics library** written in C++ to provide common APIs that work across a variety of hardware and software platforms.

Chrome Architecture

- Google Chrome uses **Skia** and **V8** as the GUI cross-platform application.

Key Components:

- **V8**: JavaScript execution engine
- **Skia**: 2D graphics rendering
- **Webkit**: Browser engine for HTML/CSS
- Platform-specific embedders (Android OS, etc.)


User Interface
Networking
Browser Engine
Data Persistence
Multimedia
Rendering Engine
Renderer
Webkit Glue
V8
Webkit
HTML
CSS
Display Backend
GPU
Skia
Plugins
Android OS
Base/Utilities

Dart

The Programming Language for Cross-platform

Static (Java) or Dynamic (JavaScript)?

Google's choice between the two different language types was supporting **both** - statically and dynamically typed.

Static typing: Types are checked at compile time
Dynamic typing: Types are checked on the fly, during execution (at runtime)

In other words, in Dart, programmers choose any type that can be **static or dynamic**.

Dart Type System Examples

We can program using Dart in different ways:

```
int x = 10;       // we specify type (old fashioned)
var x = 10;       // x is int type (type inference)
dynamic x = 10;   // the type of x can be modified at runtime
```

- The Dart compiler is smart enough to produce the correct code from type inference (line 2).
- Lines 1 and 2 will produce the same code, and line 3 will generate slow but flexible code.

System vs Virtual Machine (VM)?

Systems programming languages produce low-level machine code to speed up the execution time (AOT).

VM languages produce byte code that is executed on VM and translated into machine language at the runtime (JIT).

Dart chooses **both**, so we can use Dart VM or produce machine code directly.

Mode	Compiler	Compile time	Execution time
Debug	Just-in-time (JIT)	✅ FAST	❌ SLOW
Release	Ahead-of-time (AOT)	❌ SLOW	✅ FAST

Web (JavaScript) vs Native (Machine code)?

- Dart can produce **JavaScript code**.
- Dart also can produce **native machine code** for mobile devices (iOS and Android) and desktop computers (Mac and Windows).
- When we make code using Dart, it can be executed on **practically all platforms**.

Dart Compiler Targets:

- **Native Code (ARM, x86)**: Android, iOS, desktop
- **JS/HTML/CSS**: web
- **Dart VM**: command-line

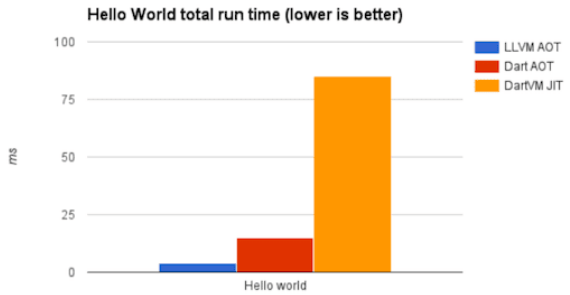
Flexible

Productive

Fast

Dart Compiler

Dart Performance Comparison



- When we **deploy**, we can choose LLVM AOT (Native AOT) to speed up runtime performance.
- When we **develop**, we can choose DartVM JIT or even JavaScript to speed up development time.

Performance: LLVM AOT ≈ Dart AOT >> DartVM JIT

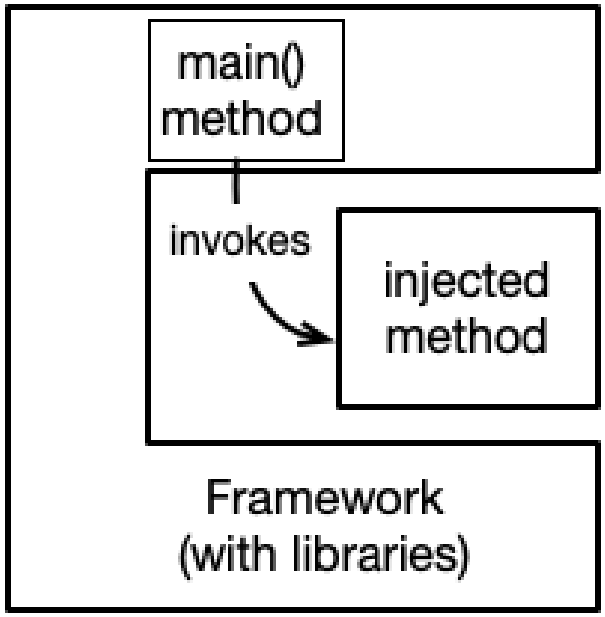
Dart Language Features

- Dart is an **OOP language**, so it supports all of the OOP principles.
- Dart is a **static language**, so the Dart compiler checks variable types to prevent runtime errors, but it also supports **dynamic language features** too.
- We can program using Dart both **Java-like** or **JavaScript-like** (Python-like if you will).

Flutter

The Cross-platform GUI framework

Framework (Dependency Injection)



- GUI frameworks provide a main() method in the generated skeleton code.
- The framework's main() calls our defined methods.
- The framework controls the flow, while we inject custom features.

Flutter Framework

Flutter is an open source framework by Google for building beautiful, natively compiled, multi-platform applications from a single codebase.

Key Benefits:

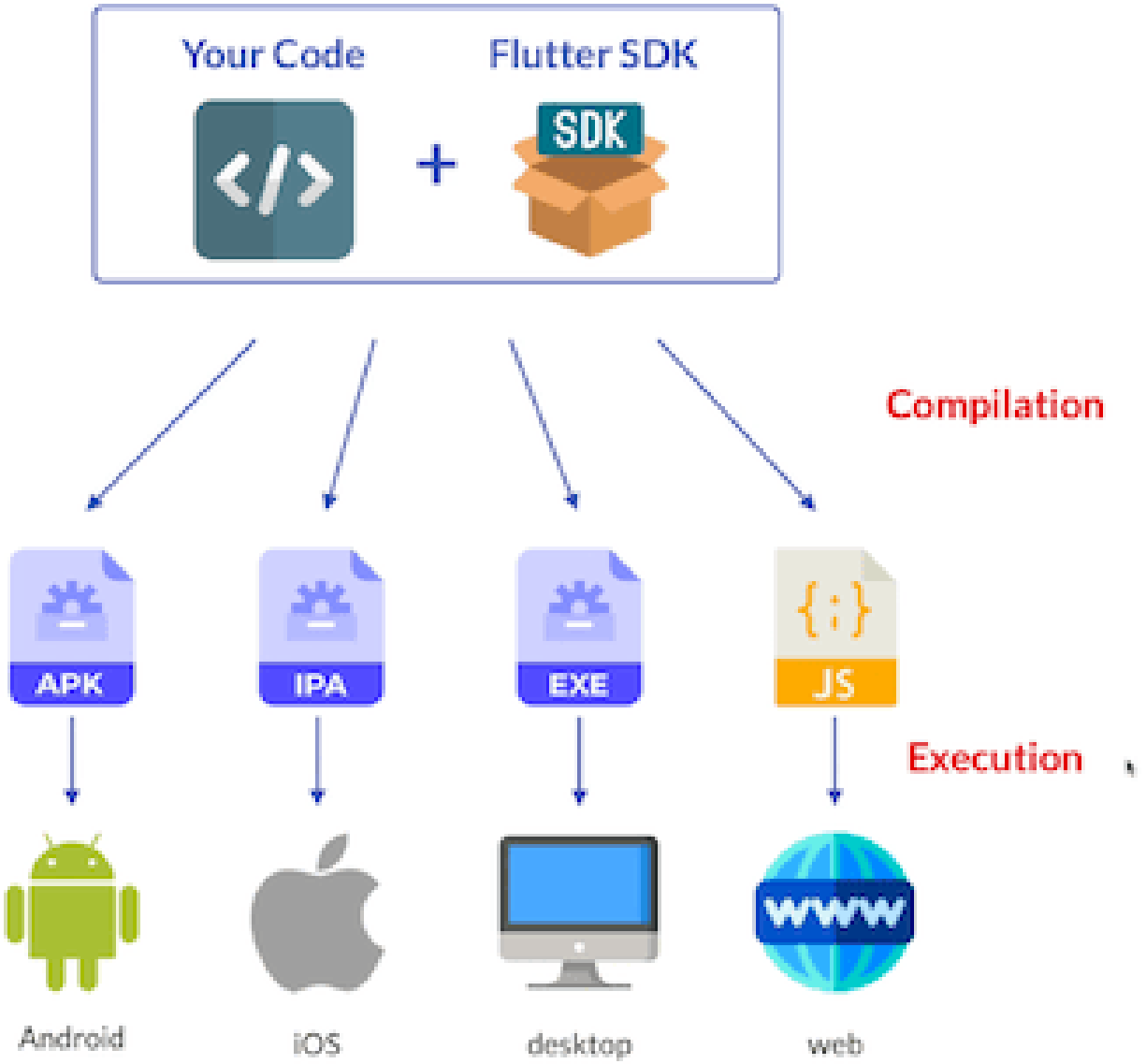
- **Fast**: Native performance
- **Productive**: Hot reload
- **Flexible**: Customizable widgets

Flutter SDK Architecture

We make applications in Dart and use Flutter SDK libraries.

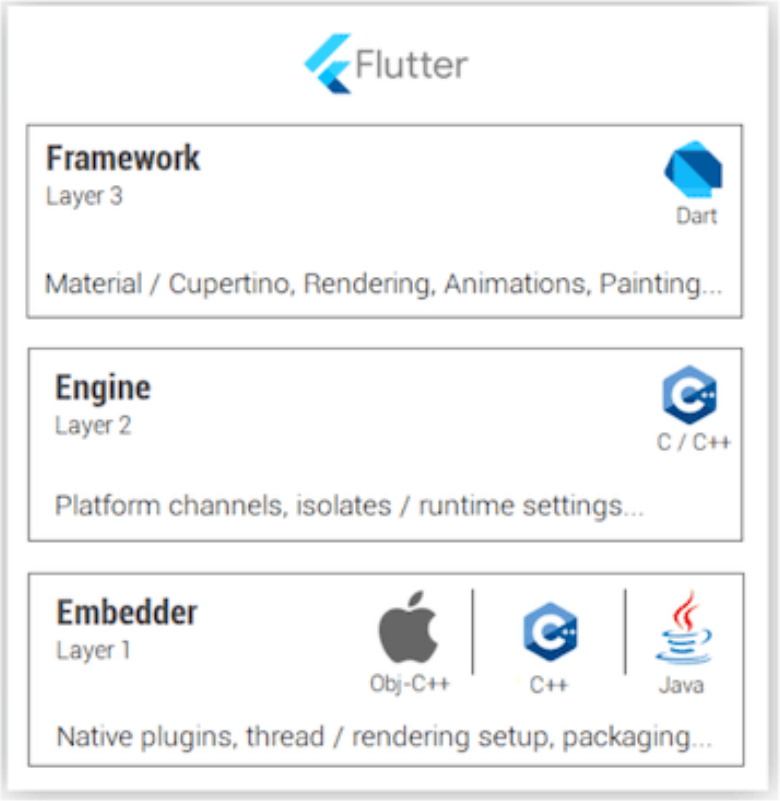
Depending on our target, we can generate:

- **Native Android application** (compiled by Android development tools)
- **Xcode applications** (for iOS)
- **Windows applications**
- **JavaScript web applications**


Your Code
Flutter SDK
SDK
Compilation
APK
IPA
EXE
JS
Execution
Android
iOS
desktop
web

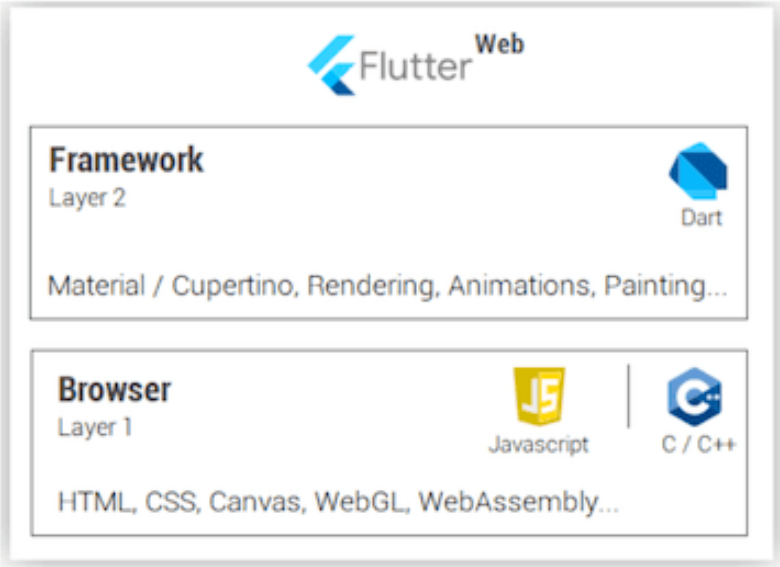
Flutter Framework Layers

Flutter separates **framework** (layer 3), **engine** (layer 2), and **embedder** (layer 1) to generate different platform development files.



Flutter for Web

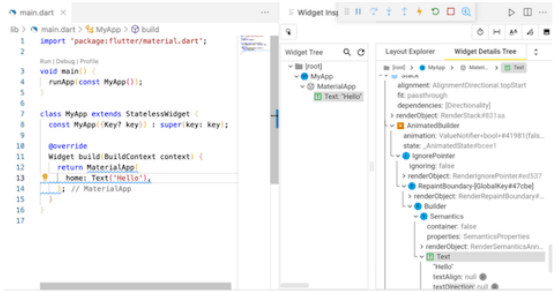
For the **web**, it has only two layers, as the layer 2 functions are delegated to web technology such as **JavaScript** or **WebAssembly**.



Widget Tree

The GUI components are written in **Dart**.

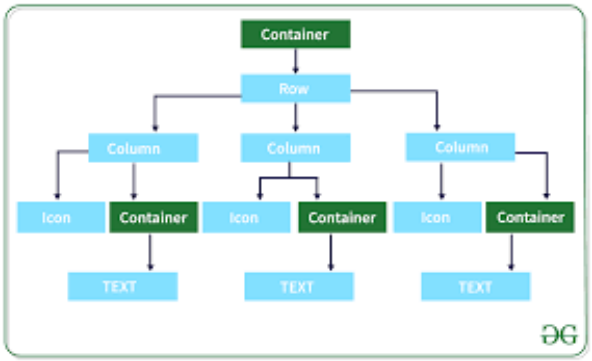
- The GUI components are read and decomposed into a **widget tree**.
- The Dart code is analyzed and converted into a widget tree structure for rendering.

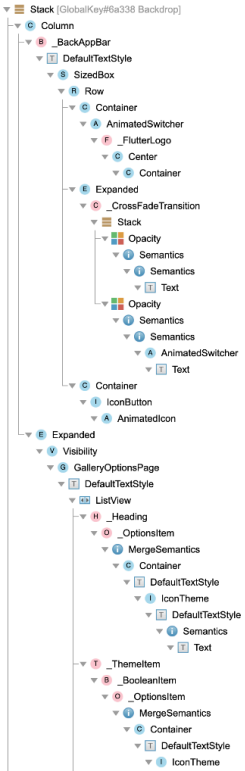


```dart
void main() {
  runApp(MyApp());
}
```

Flutter renders all UI itself, not using native libraries.

- It controls all on-screen visuals through its own rendering engine.
- Using the **Decorator pattern**, widgets wrap other widgets, forming a **Widget tree** that defines layout and behavior.



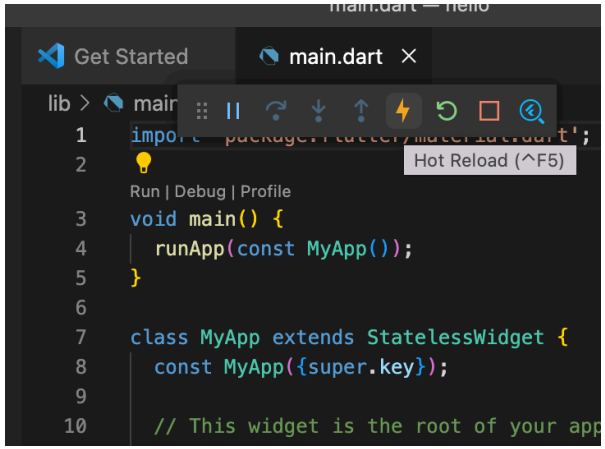


Widget relationships form a **Widget tree** using the **Decorator pattern**.

- Each wrapper decorates the widget beneath it, forming a Widget tree of nested responsibilities.

```
Center(
  child: Padding(
    padding: EdgeInsets.all(8),
    child: DecoratedBox(
      decoration: BoxDecoration(color: Colors.blue),
      child: Text('Hello'),
    ),
  ),
)
```

Hot Reload

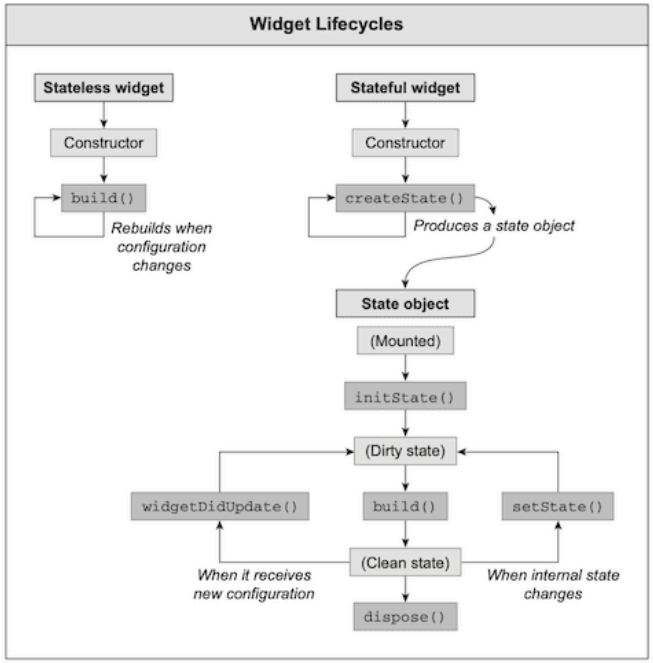


- This enables **hot reload**, letting Flutter update the UI instantly.
- Widget changes apply without restarting the app, speeding up development.

Widget Lifecycle

Flutter has a simple and straightforward design:

- For simplicity, **whatever we see is a widget**.
- All the widgets follow a **simple life cycle**.



Hot Reload