



Chapter 2

Flutter Basics

Flutter Basics

- Flutter Widgets
- Flutter Layouts
- Flutter Gesture
- State Management

Widgets

- Widgets are the primary component of any **flutter application**.
- It acts as a **UI** for the user to interact with the application.
- Any flutter application is itself a widget that is made up of a ***combination*** of widgets.
- In a standard application, the **root** defines the **structure** of the application followed by a ***MaterialApp widget*** which basically holds its internal components in place.
- This is where the properties of the **UI** and the **application** itself is set.

Cont..

- **MaterialApp Widget** is a predefined class in a flutter. It is likely the main or core component of flutter. we can access all the other components and widgets provided by **Flutter SDK**

MaterialApp and Scaffold

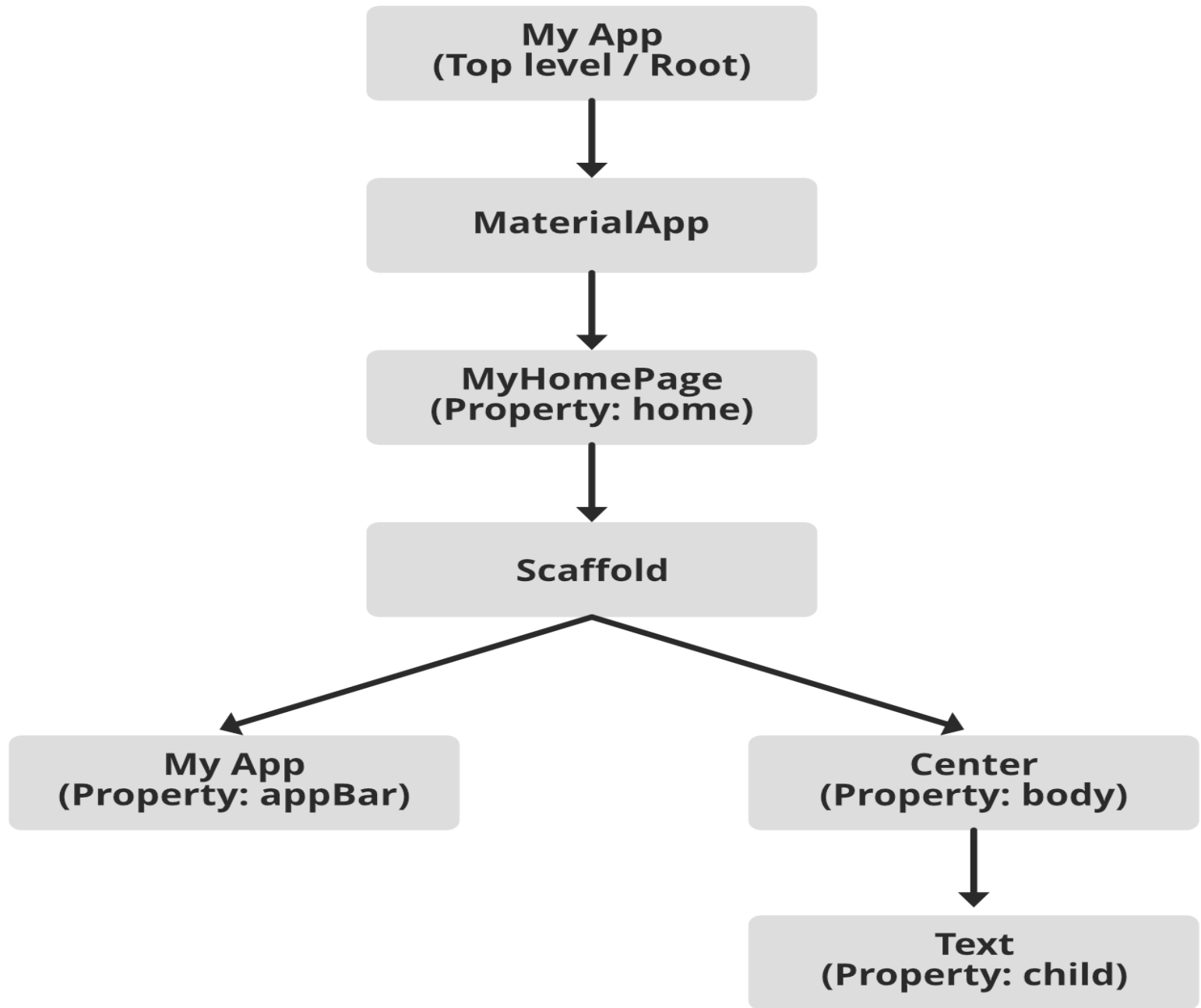
- ***MaterialApp*** is a widget that introduces a number of widgets Navigator, Theme that are required to build a material design app.
- ***Scaffold Widget*** is used under MaterialApp, it gives you many basic functionalities, like
 - AppBar,
 - BottomNavigationBar,
 - Drawer,
 - FloatingActionButton, etc.

Difference MaterialApp and Scaffold

- **MaterialApp Widget** is the starting point of your app, it tells Flutter that you are going to use Material components and follow the material design in your app.
- MaterialApp is a widget that introduces a number of widgets Navigator, Theme that are required to build a material design app.
- **Scaffold Widget** is used under MaterialApp, it gives you many basic functionalities, like AppBar, BottomNavigationBar, Drawer, FloatingActionButton, etc.
- The **Scaffold** is designed to be the single top-level container for a MaterialApp although it is not necessary to nest a Scaffold.

Example

```
void main() {  
  runApp(MaterialApp(  
    home: Scaffold(  
      appBar: AppBar(),  
      body: YourWidget(),  
    ),  
  ));  
}
```

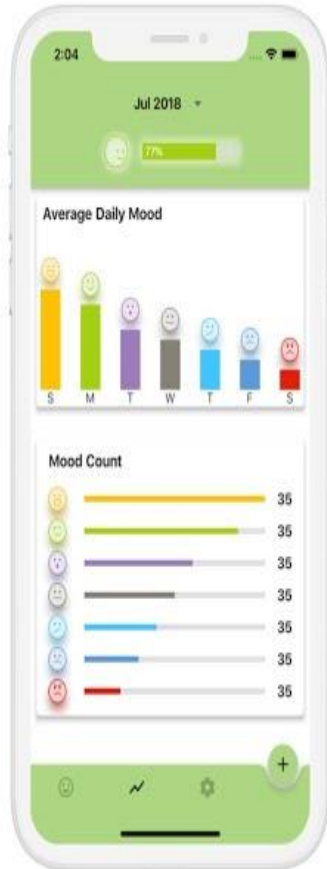


Cont..

- Inside Scaffold, there is usually an AppBar widget, which as the name suggests defines the appbar of the application.
- The scaffold also has a body where all the component widgets are placed. This is where these widget's properties are set.
- All these widgets combined to form the Homepage of the application itself.
- The Center widget has a property, Child, which refers to the actual content and it is built using the Text widget.

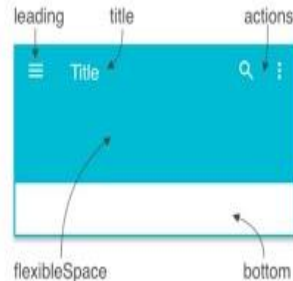
INTRODUCTION - What is covered

Scaffold



UI / UX

AppBar



Text

```
onPanUpdate:  
DragUpdateDetails(Offset(0.3, 0.0))
```

RichText

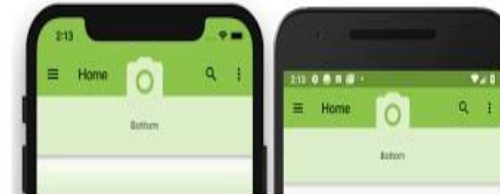
[Flutter World](#) for **Mobile**

SafeArea

No SafeArea



With SafeArea



Column

Column

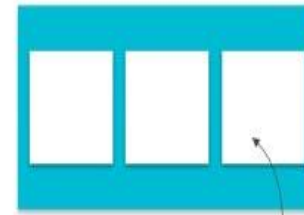


Vertically Aligned

widgets

Row

Row



Horizontally Aligned

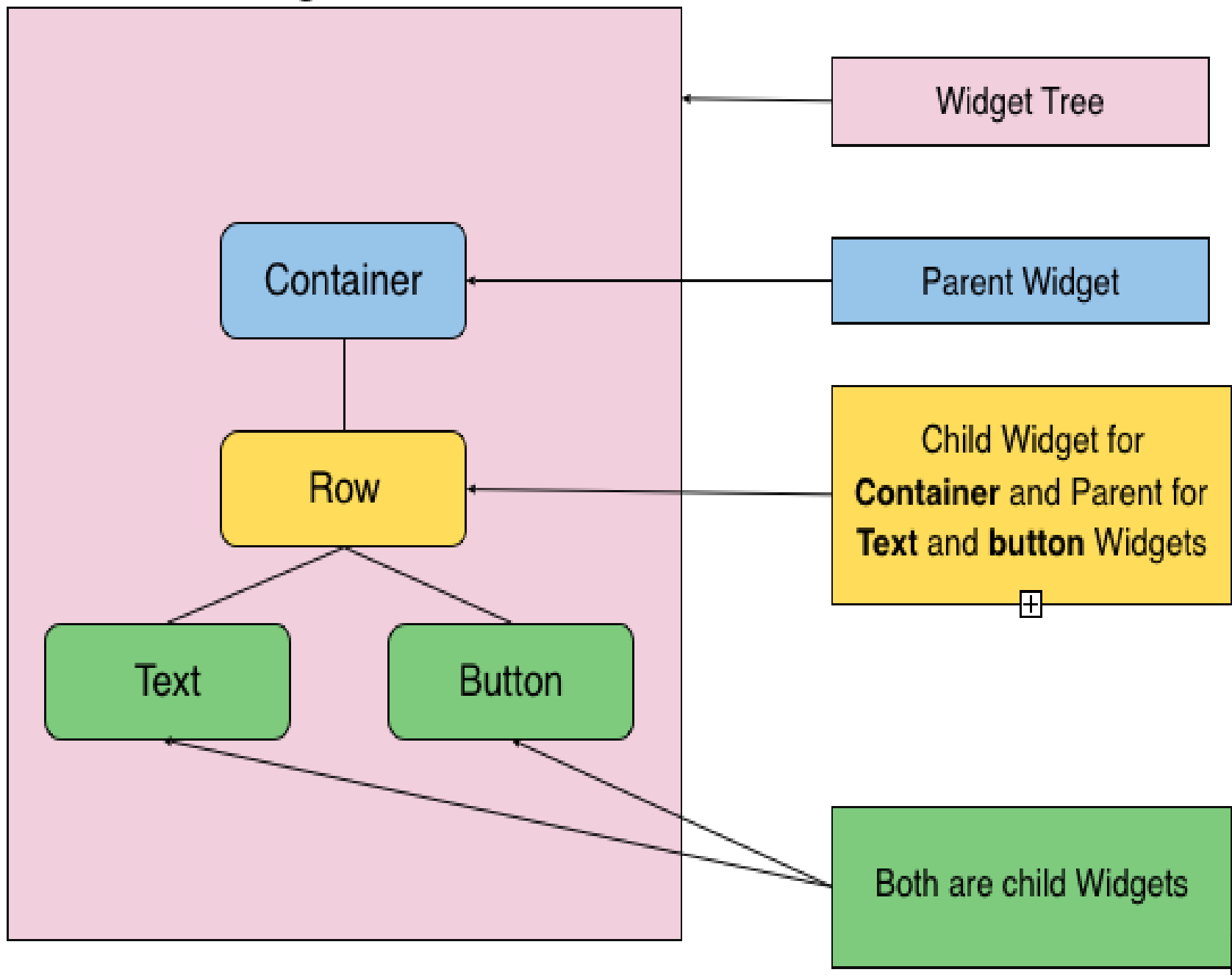
widgets

Container



Button



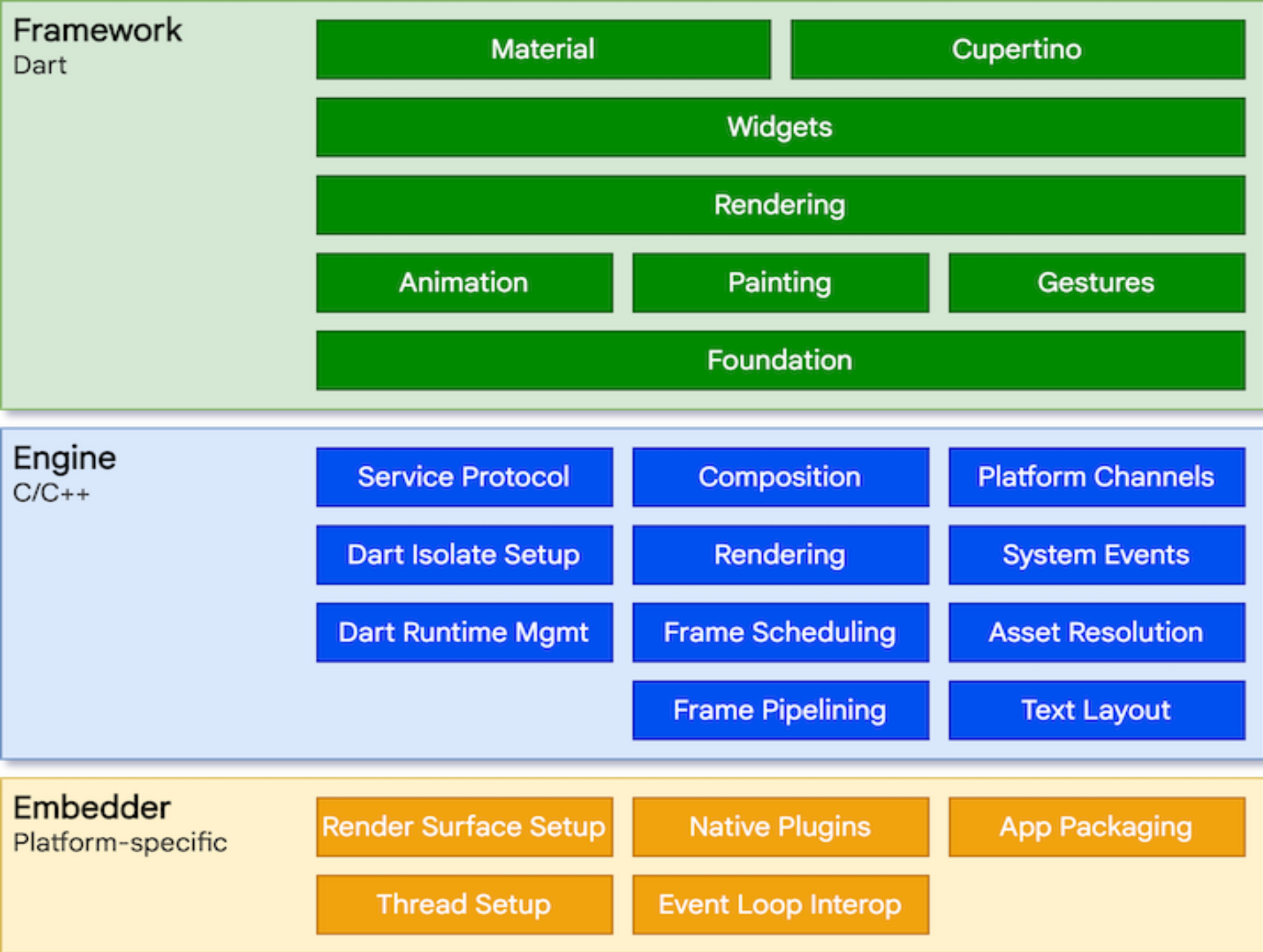


Layers



Flutter is packaged with three layers:

- Embedder (lowest layer)
- Engine
- Framework (highest layer)



Embedder layer



- An entry point is provided by a platform-specific embedder, which coordinates with the underlying operating system to access services such as accessibility, rendering surfaces, and input.
- The embedder is written in a platform-specific language, such as Java and C++ for Android, Objective-C/Objective-C++ for iOS and macOS, and C++ for Windows and Linux.
- Flutter code can be embedded into an existing application as a module or as the complete application's content using the embedder.

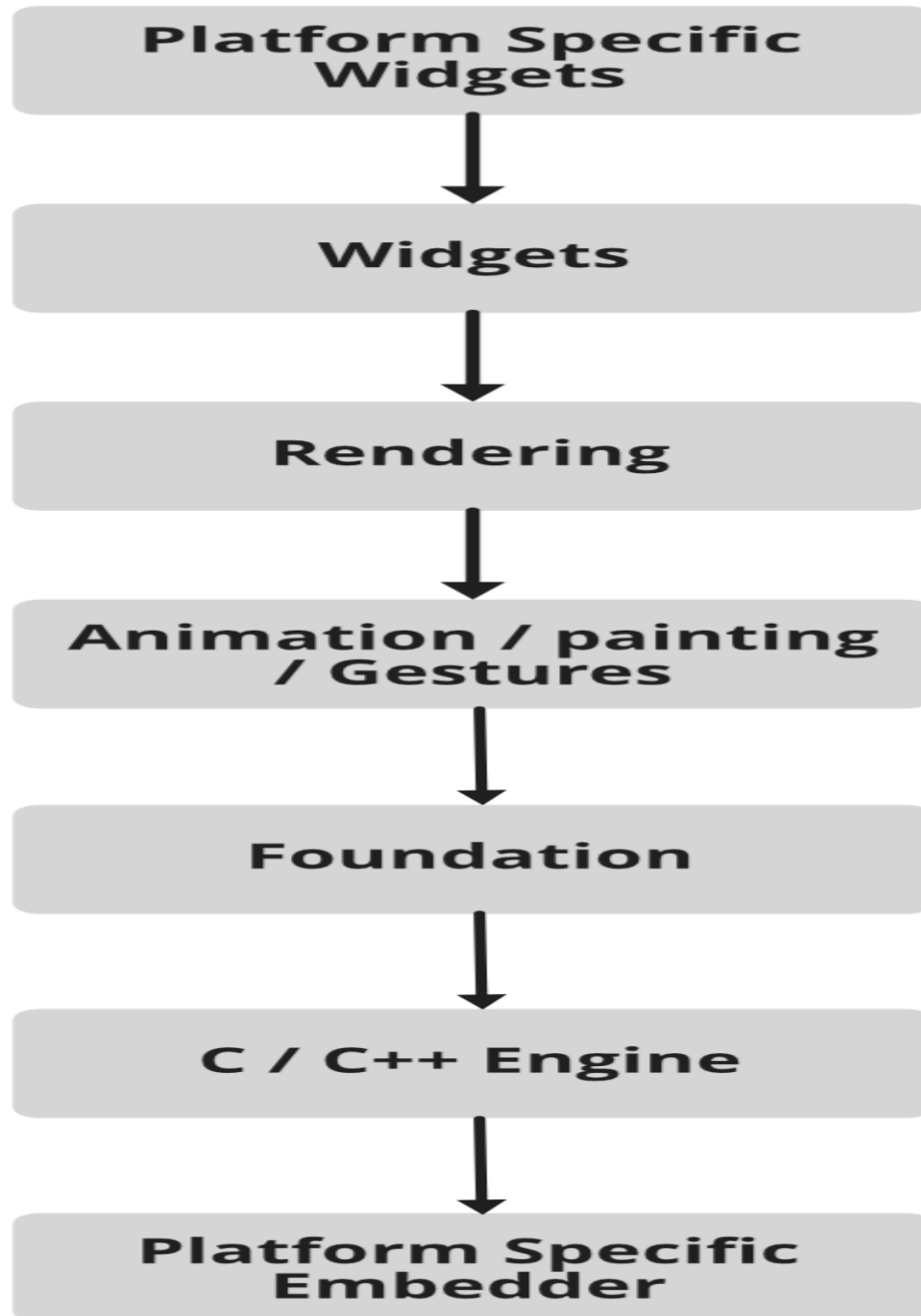
Engine Layer



- The engine layer is written in C/C++, and it takes care of the input, output, network requests, and handles the difficult translation of rendering whenever a frame needs to be painted.
- Flutter use **skia** as its rendering engine and it is revealed to the Flutter framework through the [dart : ui](#) which wraps the principal C++ code in Dart classes.

Framework Layer

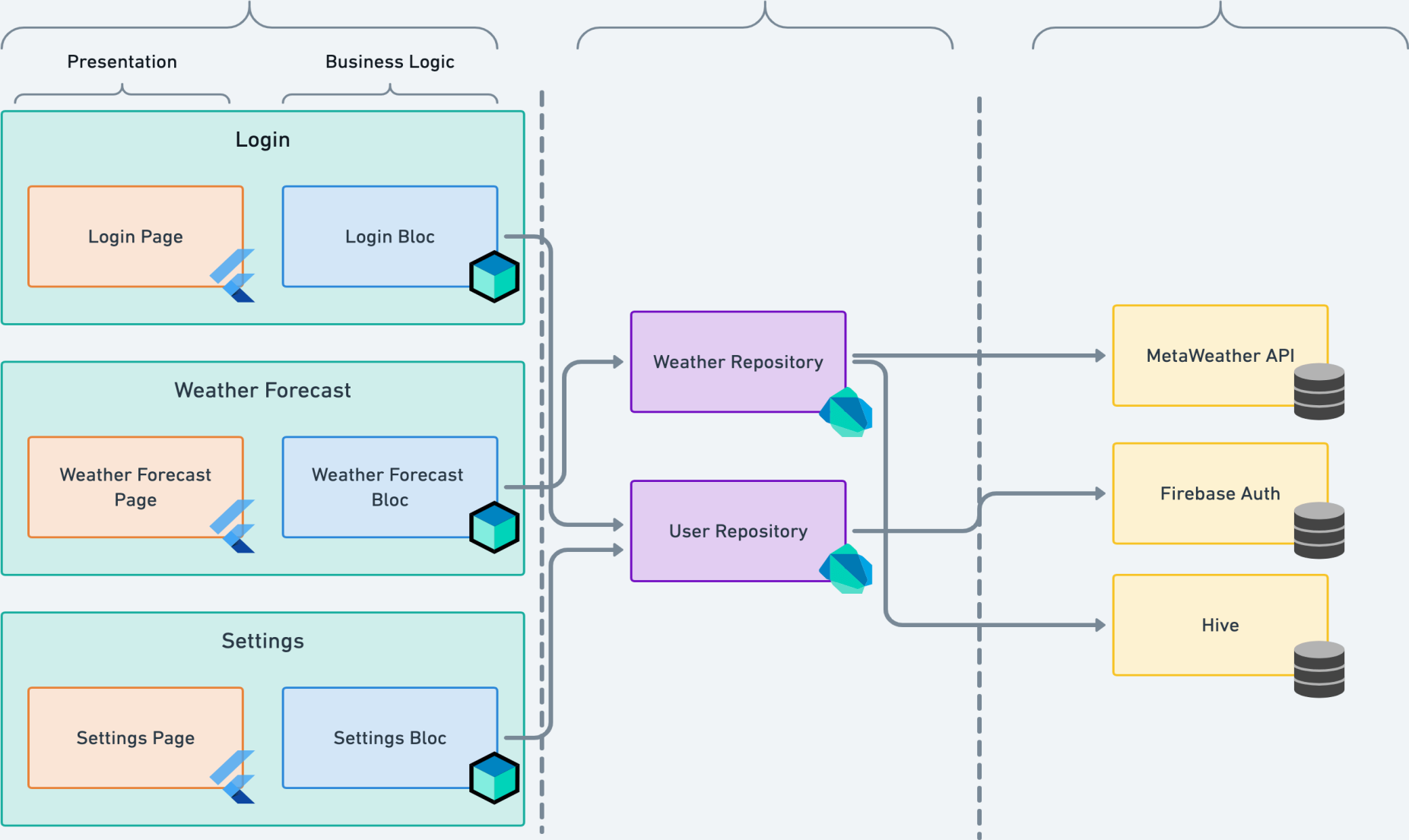
- The framework layer is the part where most developers can interact with Flutter. The Flutter framework provides a reactive and modern framework that is written in Dart.
- Within the framework layer, it comprises of the following:
 - Rendering
 - Widgets
 - Material and Cupertino
- It also has foundational classes and building block services like animation, drawing, and gestures, which are required for writing a Flutter application.




Application Layer

Domain Layer

Data Layer



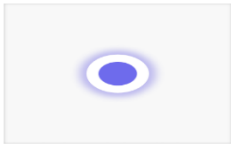
- 
- **Data layer:** This layer is the one in charge of interacting with APIs.
 - **Domain layer:** This is the one in charge of transforming the data that comes from the data layer.
 - finally, we want to manage the state of that data and present it on our user interface, that's why we split the presentation layer in two:
 - **Business logic layer:** This layer manages the state (usually using `flutter_bloc`).
 - **Presentation layer:** Renders UI components based on state.

Gesture

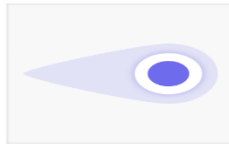
- The gesture system in Flutter has two separate layers. The first layer has raw pointer events that describe the location and movement of pointers (for example, touches, mice, and styli) across the screen. The second layer has gestures that **describe semantic actions that consist of one or more pointer movements**

Gestures

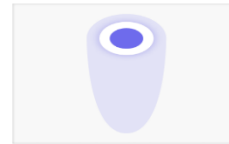
Basic



Tap



Swipe

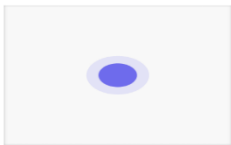


Pan

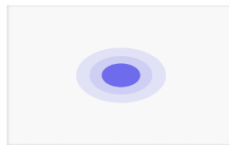


Scale

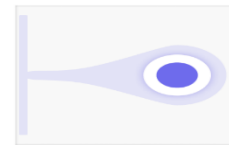
Complementary



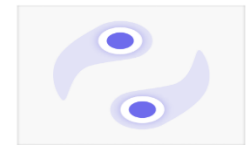
Long Press



Double Tap



Screen-Edge Swipe



Rotate

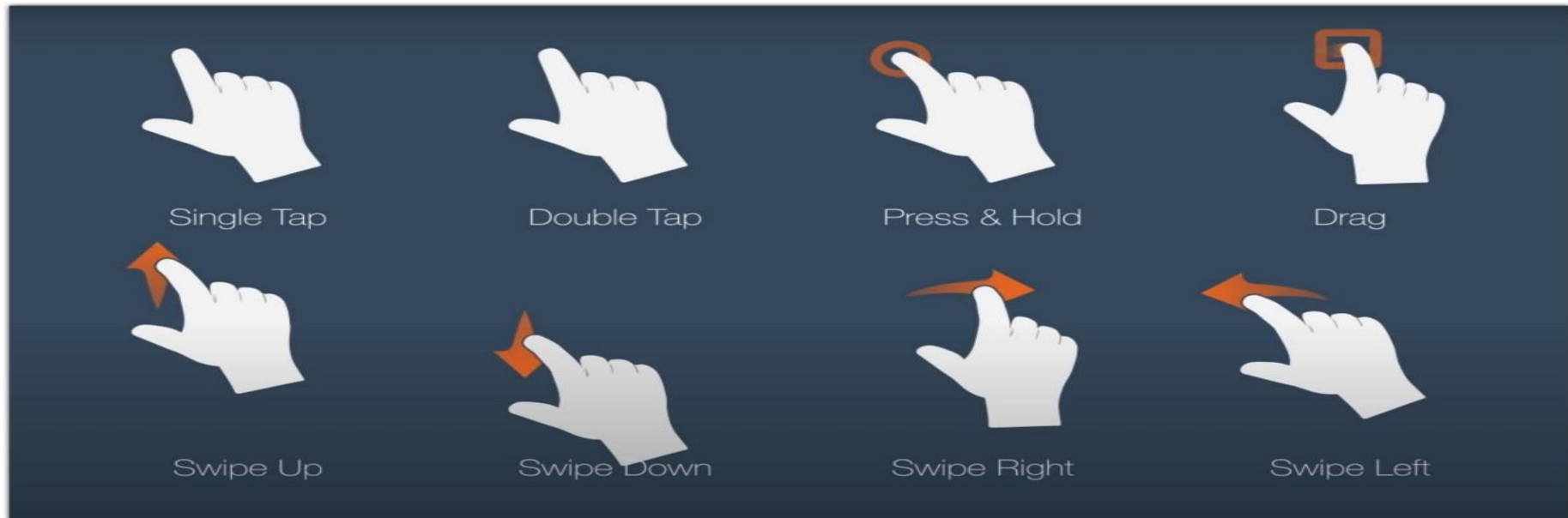
Gestures



- All physical form of interaction with a flutter application is done through pre-defined gestures. `GestureDetectors` are used for the same. It is an invisible widget that is used to process physical interaction with the flutter application. The interaction includes gestures like tapping, dragging, and swiping, etc. These features can be used to creatively enhance the user experiences of the app by making it perform desired actions based on simple gestures.

Gesture Detect

- Gesture Detector in Flutter is **used to detect the user's gestures on the application**. It is a non-visual widget. Inside the gesture detector, another widget is placed and the gesture detector will capture all these events (gestures) and execute the tasks accordingly.



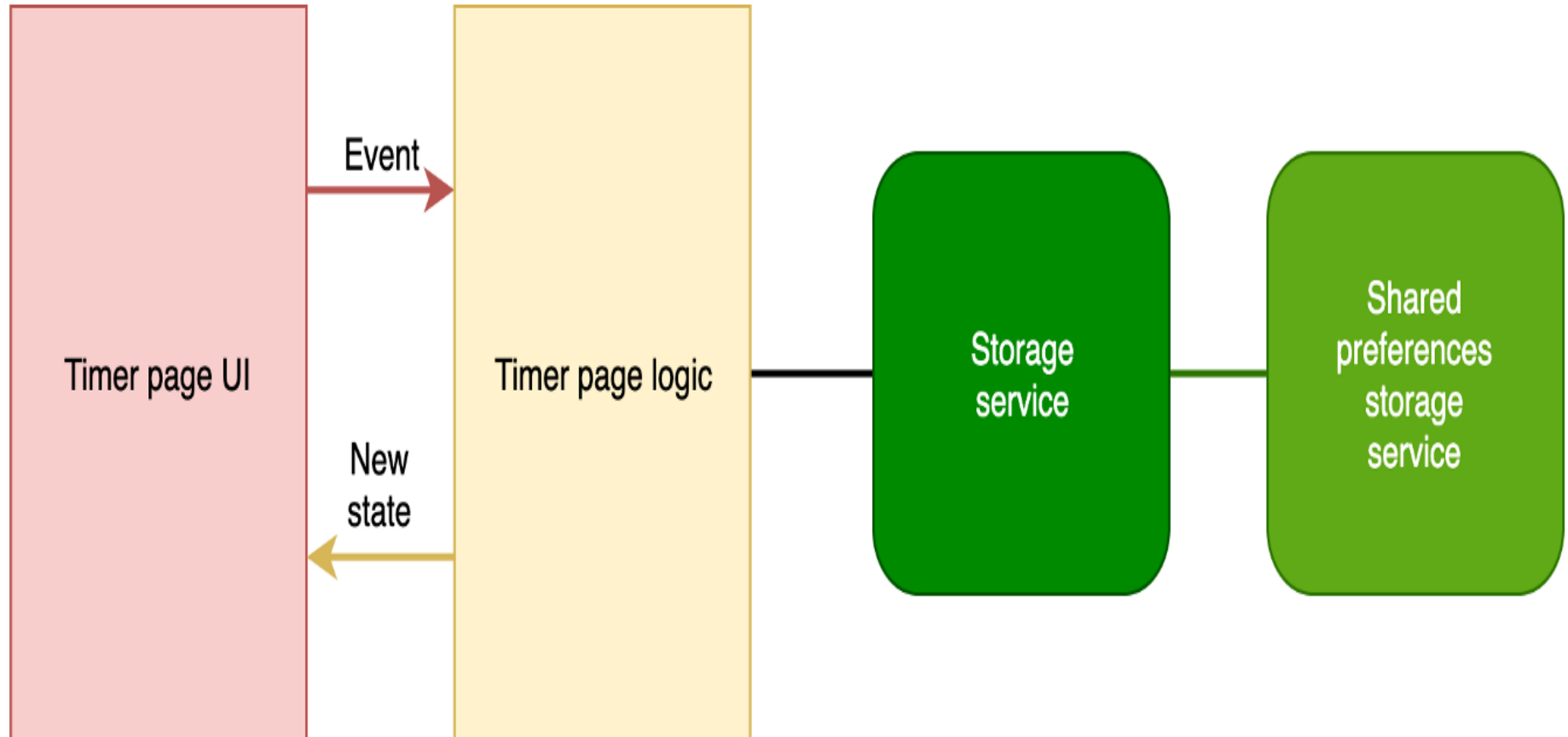
Concept of State

- State is **information that (1) can be read synchronously when the widget is built and (2) might change during the lifetime of the widget**. It is the responsibility of the widget implementer to ensure that the State is promptly notified when such state changes, using State.
- State can be described as "**whatever data you need in order to rebuild your UI at any moment in time**". When the state of your app changes (for example, the user flips a switch in the settings screen), you change the state, and that triggers a redraw of the user interface.

UI Layer

State Management Layer

Service Layer



Concept of State

- If you have ever worked with React js, you might be familiar with the concept of a state. The states are nothing but data objects. Flutter also operates on similar turf. For the management of state in a Flutter application, Stateful Widget is used. Similar to the concept of state in React js, the re-rendering of widgets specific to the state occurs whenever the state changes. This also avoids the re-rendering of the entire application, every time the state of a widget changes.

Stateful and Stateless Widgets

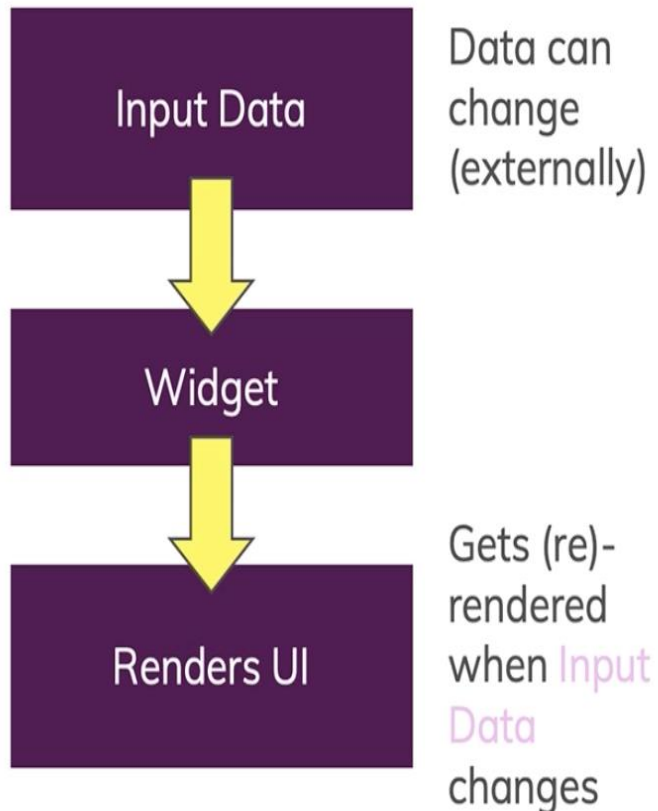
- A widget is either stateful or stateless. **If a widget can change—when a user interacts with it, for example—it's stateful.**
- **A stateless widget never changes.** Icon , IconButton , and Text are examples of stateless widgets.

Examples

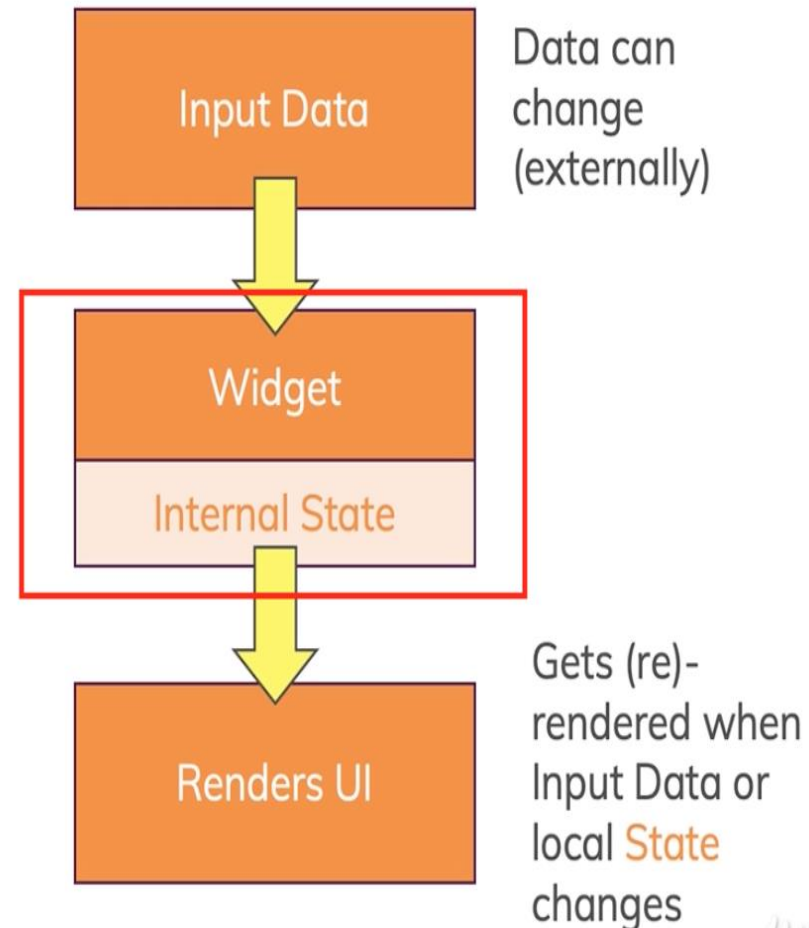
- Stateless widgets are **text, icons, icon buttons, and raised buttons.**
- A stateful widget is dynamic: for example, it can change its appearance in response to events triggered by user interactions or when it receives data. **Checkbox , Radio , Slider , InkWell , Form , and TextField** are examples of stateful widgets.

Stateless vs Stateful

Stateless



Stateful



Stateful Widget	Stateless Widget
<p>when Widget changes its value, that's Stateful.</p> <p>e.g. Checkbox, Radio button, Textfield</p>	<p>No change in widget value, that's Stateless.</p> <p>e.g. Text, Icon, Icon button, Raised button</p>
<p>Override the createState() and return State.</p>	<p>Override the build() and return Widget.</p>
<p>Use when user want to change UI dynamically.</p>	<p>Use when UI remains constant during runtime.</p>
<p>When Widget's state changes, the State object calls setState(), telling framework to redraw widget.</p>	

Stateless

Does not require the server to retain information about the state.

Server design, implementation and architecture is simple.

Handles crashes well, as we can fail over to a completely new server. Servers are regarded as cheap commodity machines.

Scaling architecture is easy.

Stateful

Requires a server to save information about a session.

Server design, implementation and architecture is complicated.

Does not handle crashes well. Servers are regarded as valuable and long-living. The user would probably be logged out and have to start from the beginning.

Scaling architectures is difficult and complex.

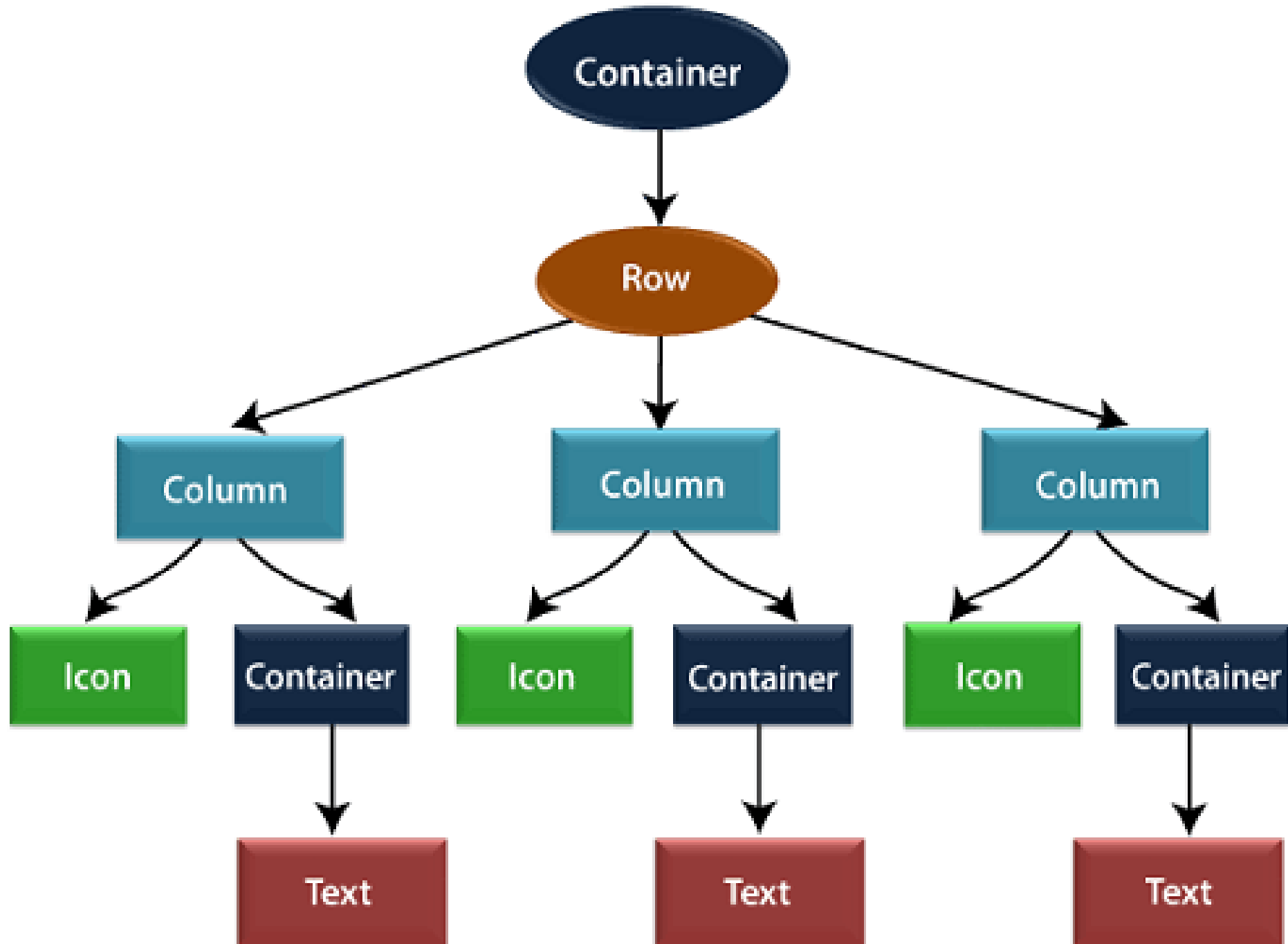
Flutter Layouts

- The main concept of the layout mechanism is the widget. We know that flutter assume everything as a widget. So the image, icon, text, and even the layout of your app are all widgets. Here, some of the things you do not see on your app UI, such as rows, columns, and grids that arrange, constrain, and align the visible widgets are also the widgets.
- Flutter allows us to create a layout by composing multiple widgets to build more complex widgets. **For example**, we can see the below image that shows three icons with a label under each one.

Cont..



In the second image, we can see the visual layout of the above image. This image shows a row of three columns, and these columns contain an icon and label.



Cont.

- In the above **image**, the **container** is a **widget class** that allows us to customize the child widget. It is mainly used to add borders, padding, margins, background color, and many more. Here, the text widget comes under the container for adding margins. The entire row is also placed in a container for adding margin and padding around the row. Also, the rest of the UI is controlled by properties such as color, text. Style, etc.

Layout a Widget

- Let us learn how we can create and display a simple widget. The following steps show how to layout a widget:
- **Step 1:** First, you need to select a Layout widget.
- **Step 2:** Next, create a visible widget.
- **Step 3:** Then, add the visible widget to the layout widget.
- **Step 4:** Finally, add the layout widget to the page where you want to display.

Types of Layout Widgets

- We can categories the layout widget into two types:
 1. Single Child Widget
 2. Multiple Child Widget

Single Child Widgets

- The single child layout widget is a type of widget, which can have only **one child widget** inside the parent layout widget. These widgets can also contain special **layout functionality**. Flutter provides us many *single child widgets* to make the *app UI* attractive. If we use these widgets appropriately, it can **save** our time and makes the app code more readable. The list of different types of *single child widgets* are:

Example

- **Container:** It is the most popular layout widget that provides customizable options for painting, positioning, and sizing of widgets.

```
Center(  
  child: Container(  
    margin: const EdgeInsets.all(15.0),  
    color: Colors.blue,  
    width: 42.0,  
    height: 42.0,  
  ),  
)
```

Padding

- It is a widget that is used to arrange its child widget by the given padding.
- It contains **EdgeInsets** and **EdgeInsets.fromLTRB** for the desired side where you want to provide padding.

```
const Greetings(  
  child: Padding(  
    padding: EdgeInsets.all(14.0),  
    child: Text('Hello JavaTpoint!'),  
  ),  
)
```

Cont..

- **Center:** This widget allows you to center the child widget within itself.
- **Align:** It is a widget, which aligns its child widget within itself and sizes it based on the child's size. It provides more control to place the ***child widget*** in the ***exact position*** where you need it.

Example

```
Center(  
  child: Container(  
    height: 110.0,  
    width: 110.0,  
    color: Colors.blue,  
    child: Align(  
      alignment: Alignment.topLeft,  
      child: FlutterLogo(  
        size: 50,  
      ),  
    ),  
  ),  
)
```

SizedBox:



This widget allows you to give the specified size to the child widget through all screens.

```
SizedBox(  
  width: 300.0,  
  height: 450.0,  
  child: const Card(child: Text('Hello JavaTpoint!'),  
)
```

Aspect Ratio



- This widget allows you to keep the size of the child widget to a specified aspect ratio.

```
AspectRatio(  
  aspectRatio: 5/3,  
  child: Container(  
    color: Colors.blue,  
  ),  
)
```

Baseline

- This widget shifts the child widget according to the child's baseline.

```
child: Baseline(  
  baseline: 30.0,  
  baselineType: TextBaseline.alphabetic,  
  child: Container(  
    height: 60,  
    width: 50,  
    color: Colors.blue,  
  ),  
)
```

ConstrainedBox



- It is a widget that allows you to force the additional constraints on its child widget. It means you can force the child widget to have a specific constraint without changing the properties of the child widget.

Example

```
ConstrainedBox(  
  constraints: new BoxConstraints(  
    minHeight: 150.0,  
    minWidth: 150.0,  
    maxHeight: 300.0,  
    maxWidth: 300.0,  
  ),  
  child: new DecoratedBox(  
    decoration: new BoxDecoration(color: Colors.red,  
  ),  
),
```

CustomSingleChildLayout

- It is a widget, which defers from the layout of the single child to a delegate. The delegate decides to position the child widget and also used to determine the size of the parent widget.
- **FittedBox:** It **scales** and **positions** the child widget according to the specified **fit**.

Example

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

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

class MyApp extends StatelessWidget {
  // It is the root widget of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Multiple Layout Widget',
      debugShowCheckedModeBanner: false,
      theme: ThemeData(
        // This is the theme of your application.
        primarySwatch: Colors.green,
      ),
      home: MyHomePage(),
    );
  }
}
```


Cont..

```
class MyHomePage extends StatelessWidget {  
  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(title: Text("FittedBox Widget")),  
      body: Center(  
        child: FittedBox(child: Row(  
          children: <Widget>[  
            Container(  
              child: Image.asset('assets/computer.png'),  
            ),  
            Container(  
              child: Text("This is a widget"),  
            )  
          ],  
        ),  
        fit: BoxFit.contain,  
      )  
    ),  
  );  
}
```

Output



THANK YOU

