

# Flutter: basic concepts

# Flutter and Dart

Runtime




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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Welcome to Flutter',
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Welcome to Flutter'),
        ),
        body: const Center(
          child: Text('Hello World'),
        ),
      ),
    );
  }
}
```

# Flutter and Dart

Top of the tree: an widget

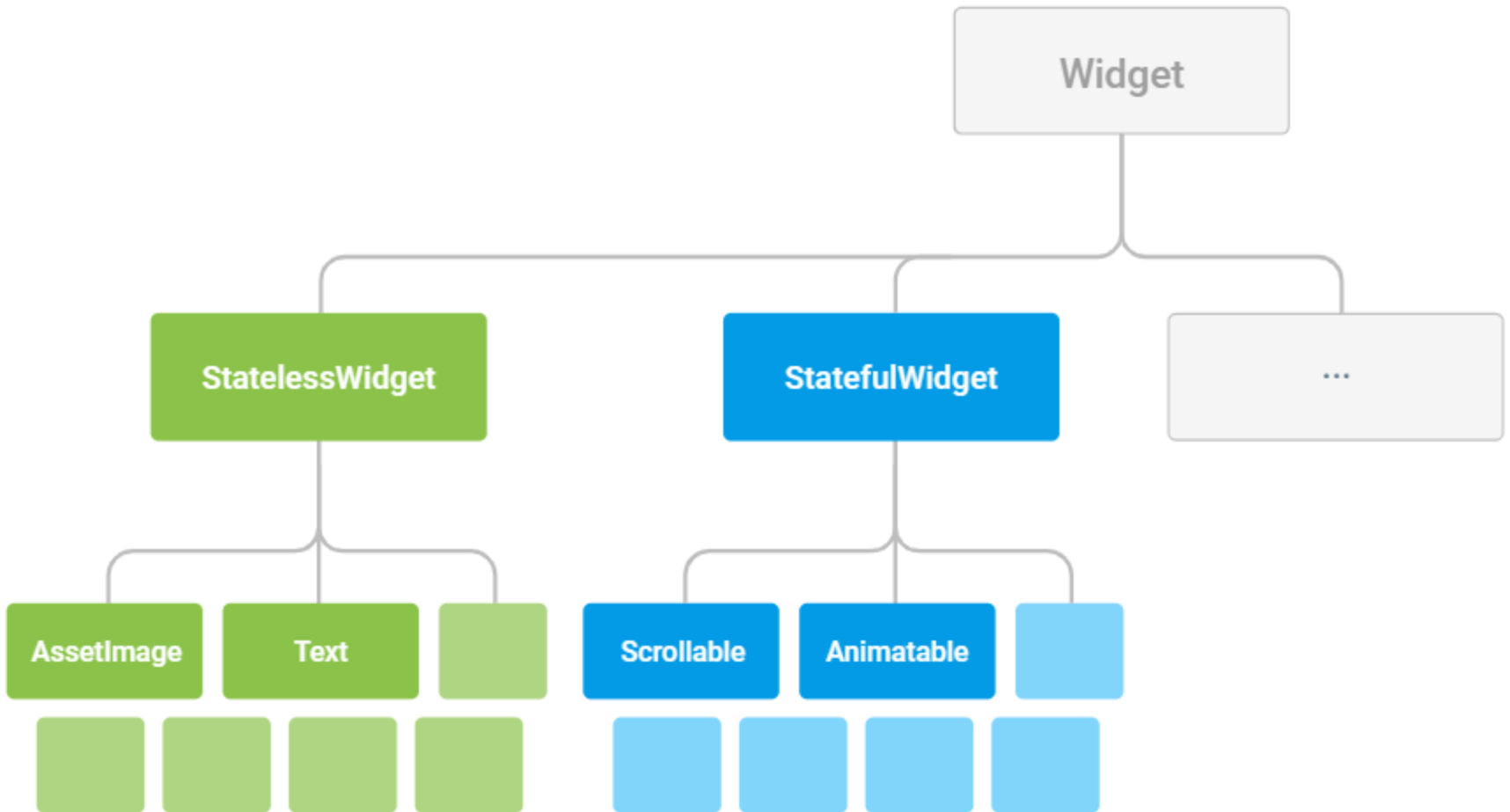


```
import 'package:flutter/material.dart';

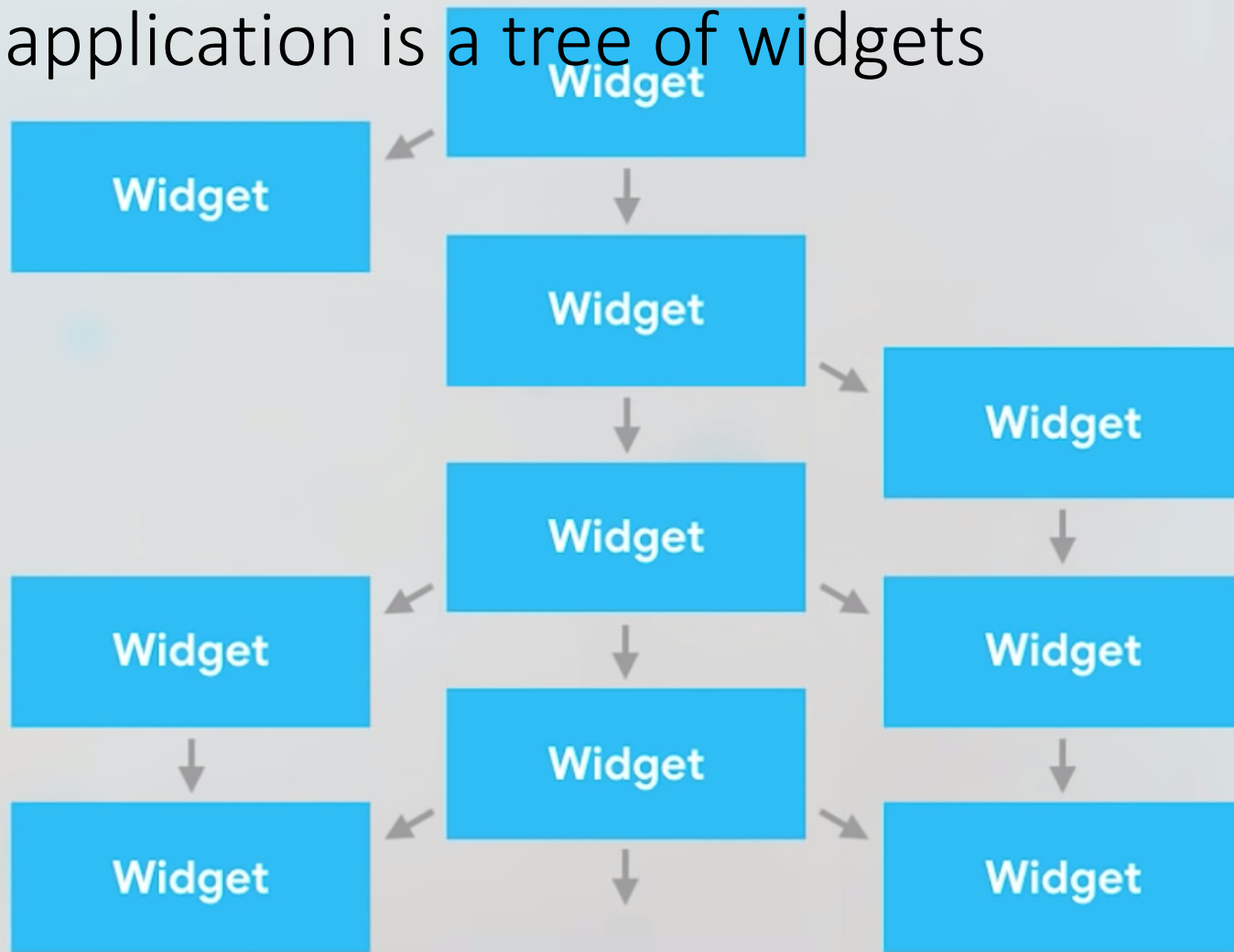
void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Welcome to Flutter',
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Welcome to Flutter'),
        ),
        body: const Center(
          child: Text('Hello World'),
        ),
      ),
    );
  }
}
```

# Everything's a widget

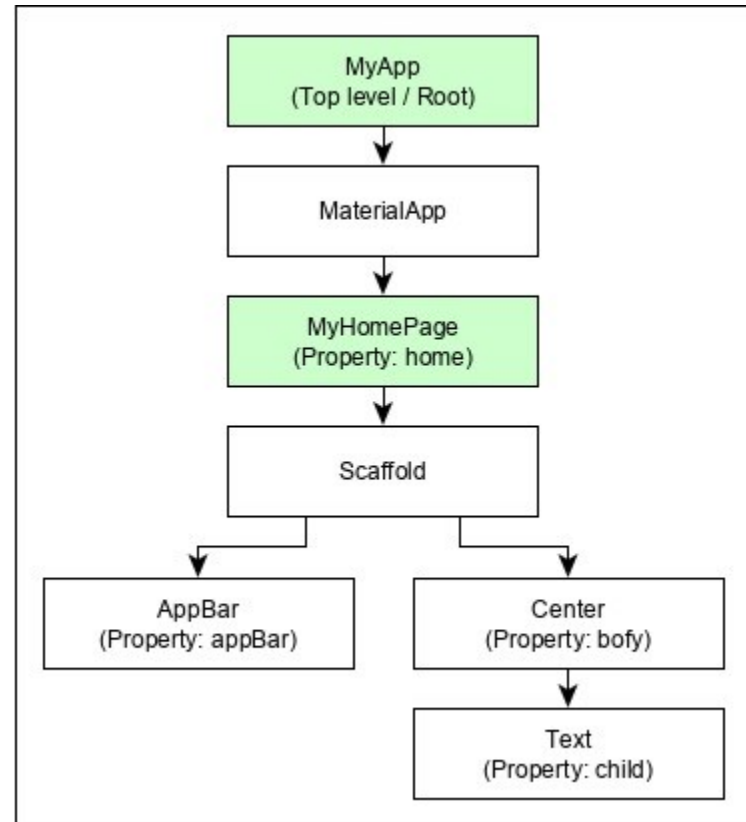


An application is a tree of widgets



# Widget

- user interface components used to create the user interface of the application.
  - Can have children
  - It's UI is build using one or more children (widgets),
  - which again build using its children widgets.
- **Everything is a widget.**
  - the application is itself a widget.
- This **composability** feature helps us to create a user interface of any complexity.



[https://www.tutorialspoint.com/flutter/flutter\\_architecture\\_application.htm](https://www.tutorialspoint.com/flutter/flutter_architecture_application.htm)

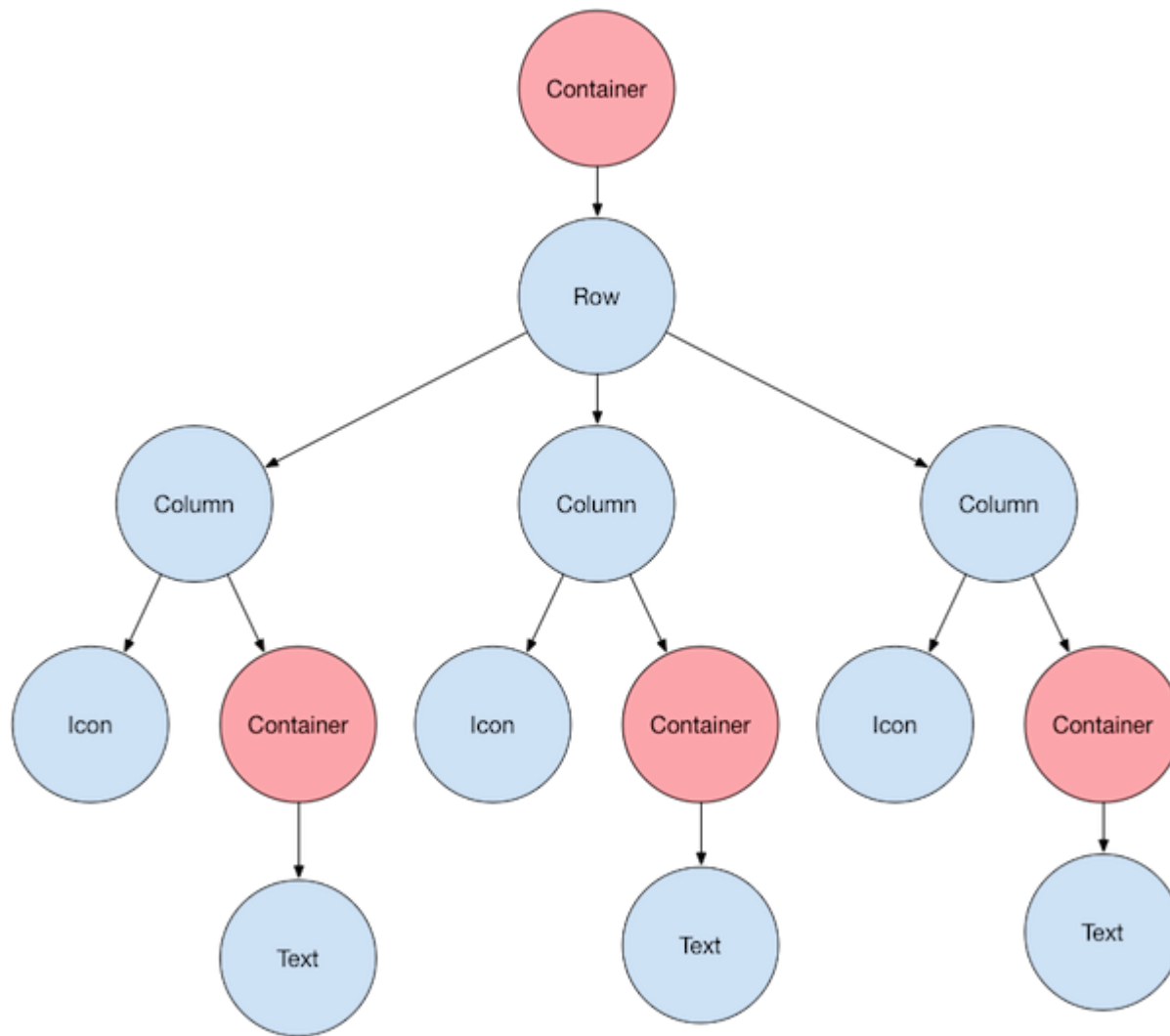
# Types of Widget: UI visibility

## Visible (Output and Input)

- Text
- Button
- Image

## Invisible (Layout and Control)

- Column
  - vertical alignment.
- Row
  - horizontal alignment
- Center
  - center the child widget
- Padding
  - padding in specified directions.
- **Scaffold**
  - common material design elements like AppBar, Floating Action Buttons, Drawers, etc.
- **Stack**
  - mainly used for **overlapping** a widget
- (...)



Layouts in Flutter

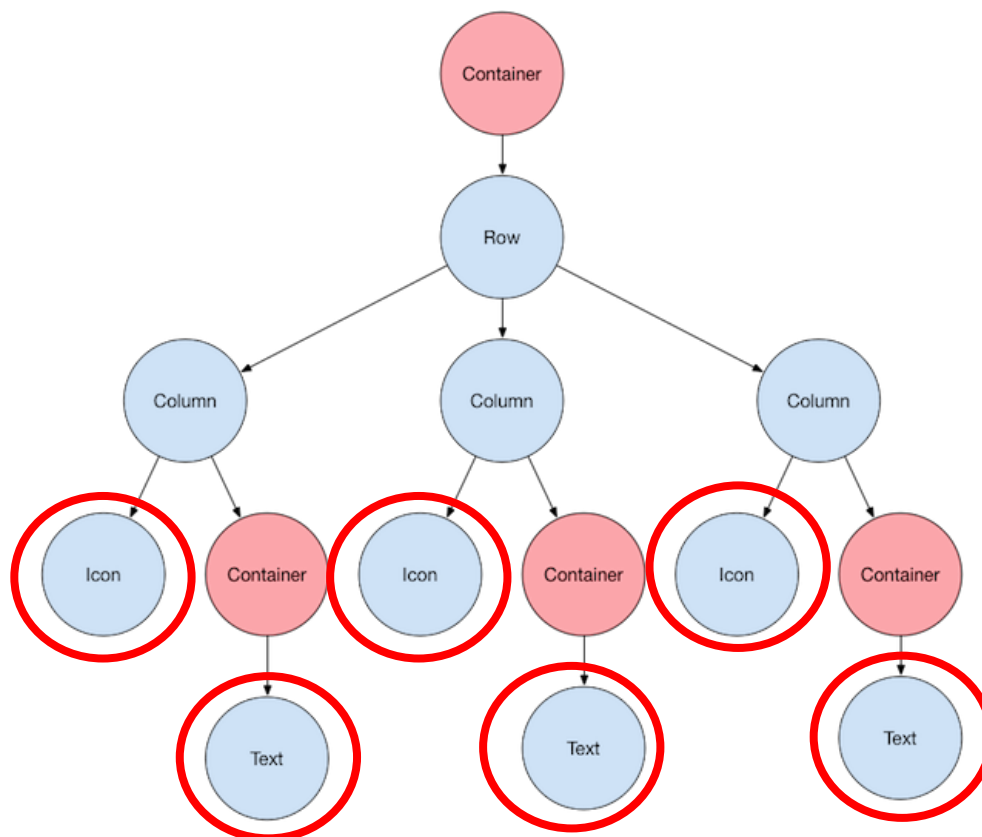
<https://flutter.dev/docs/development/ui/layout>



# Visible and layout widget

**Visible**

**layout**



# All layout widgets have either of the following:

## **Single child**

- A child property if they take a single child – for example, Center or Container

## **List of widgets**

- A children property if they take a list of widgets –
- example, Row, Column, ListView, or Stack.

# Every Widget has its own build() and its context.

- BuildContext is the parent of the widget returned by the build() method.
  - the location of the Widget in the widget tree.
  - a widget of widgets, like nested.wrap <div <div> .html>
  - parent objects

```
class _MyHomePageState extends State<MyHomePage> {  
  _MyHomePageState() {  
    print(context.hashCode);  
    // prints 2011  
  }  
  
  @override  
  Widget build(BuildContext context) {  
    return new Scaffold(  
      appBar: new AppBar(  
        title: new Text(widget.title),  
      ),  
      body: new Container(),  
      floatingActionButton:  
        new FloatingActionButton(onPressed: () => print(context.hashCode)),  
      // prints 63  
    );  
  }  
}
```

# The 'of()' Method

- Allows looking up and down the widget tree, in some cases, to reference other Widgets

```
@override
Widget build(context) {
  return new Text('Hello, World',
    style: new TextStyle(color: Theme.of(context).primaryColor),
  );
}
```

‘of’ method is looking up the tree for the next parent widget that is of type Theme, and grabbing that primary color property. The framework can find the correct Theme object because it knows the tree in relation to this build context.

# The Builders

- Builder is a widget that takes a closure and uses it to build its child widgets.
- can use it to pass the context from a build method directly to children being returned in that build method.

```
@override
Widget build(BuildContext context) {
  return new Scaffold(
    appBar: new AppBar(
      title: new Text(widget.title),
    ),
    body: new Container(),
    /// Builders let you pass context
    /// from your *current* build method
    /// Directly to children returned in this build method
    ///
    /// The 'builder' property accepts a callback
    /// which can be treated exactly as a 'build' method on any
    /// widget
    floatingActionButton: new Builder(builder: (context) {
      return new FloatingActionButton(onPressed: () {
        Scaffold.of(context).showSnackBar(
          new SnackBar(
            backgroundColor: Colors.blue,
            content: new Text('SnackBar'),
          ),
        );
      });
    }),
  );
}
```

# Types of Widget: state



## stateless

- don't store any state.
- don't store values that might change.
- an Icon is stateless;
  - you set the icon image when you create it and then it doesn't change any more.
- A Text widget is also stateless.
  - You might say, "But wait, you can change the text value." True, but if you want to change the text value, you just create a whole new widget with new text.
  - doesn't store a text property that can be changed.

## stateful widget

- can keep track of changes and update the UI based on those changes.
- it creates a State object that keeps track of the changes.
  - When the values in the State object change,
  - it creates a whole new widget with the updated values.
- a checkbox is a stateful widget

# Stateless widget

```
class Frog extends StatelessWidget {  
  const Frog({  
    Key? key,  
    this.color = const Color(0xFF2DBD3A),  
    this.child,  
  }) : super(key: key);  
  
  final Color color;  
  final Widget? child;  
  
  @override  
  Widget build(BuildContext context) {  
    return Container(color: color, child: child);  
  }  
}
```

<https://api.flutter.dev/flutter/widgets/StatelessWidget-class.html>

# Stateful widget

```
class Bird extends StatefulWidget {  
  const Bird({  
    Key? key,  
    this.color = const Color(0xFFFFE306),  
    this.child,  
  }) : super(key: key);  
  
  final Color color;  
  final Widget? child;  
  
  @override  
  State<Bird> createState() => _BirdState();  
}
```

```
class _BirdState extends State<Bird> {  
  double _size = 1.0;  
  
  void grow() {  
    setState(() { _size += 0.1; });  
  }  
  
  @override  
  Widget build(BuildContext context) {  
    return Container(  
      color: widget.color,  
      transform: Matrix4.diagonal3Values(_size, _size, 1.0),  
      child: widget.child,  
    );  
  }  
}
```

<https://api.flutter.dev/flutter/widgets/StatefulWidget-class.html>



# Stateful widget

- StatefulWidget
  - themselves are immutable

```
class Bird extends StatefulWidget {  
  const Bird({  
    Key? key,  
    this.color = const Color(0xFFFFE306),  
    this.child,  
  }) : super(key: key);  
  
  final Color color;  
  final Widget? child;  
  
  @override  
  State<Bird> createState() => _BirdState();  
}
```

- Stateless widget are useful when the part of the user interface you are describing does not depend on anything other than the configuration information in the object itself

# Stateful widget

- StatefulWidget

- themselves are immutable
- store their mutable state either in separate State objects

```
class Bird extends StatefulWidget {  
  const Bird({  
    Key? key,  
    this.color = const Color(0xFFFFE306),  
    this.child,  
  }) : super(key: key);  
  
  final Color color;  
  final Widget? child;  
  
  @override  
  State<Bird> createState() => _BirdState();  
}
```

```
class _BirdState extends State<Bird> {  
  double _size = 1.0;  
  
  void grow() {  
    setState(() { _size += 0.1; });  
  }  
  
  @override  
  Widget build(BuildContext context) {  
    return Container(  
      color: widget.color,  
      transform: Matrix4.diagonal3Values(_size, _size, 1.0)  
      child: widget.child,  
    );  
  }  
}
```

<https://api.flutter.dev/flutter/widgets/StatefulWidget-class.html>

# Stateful widget

- StatefulWidget

- themselves are immutable
- store their mutable state either in separate State objects

```
class Bird extends StatefulWidget {  
  const Bird({  
    Key? key,  
    this.color = const Color(0xFFFFE306),  
    this.child,  
  }) : super(key: key);  
  
  final Color color;  
  final Widget? child;  
  
  @override  
  State<Bird> createState() => _BirdState();  
}
```

```
class _BirdState extends State<Bird> {  
  double _size = 1.0;  
  
  void grow() {  
    setState(() { _size += 0.1; });  
  }  
  
  @override  
  Widget build(BuildContext context) {  
    return Container(  
      color: widget.color,  
      transform: Matrix4.diagonal3Values(_size, _size, 1.0)  
      child: widget.child,  
    );  
  }  
}
```

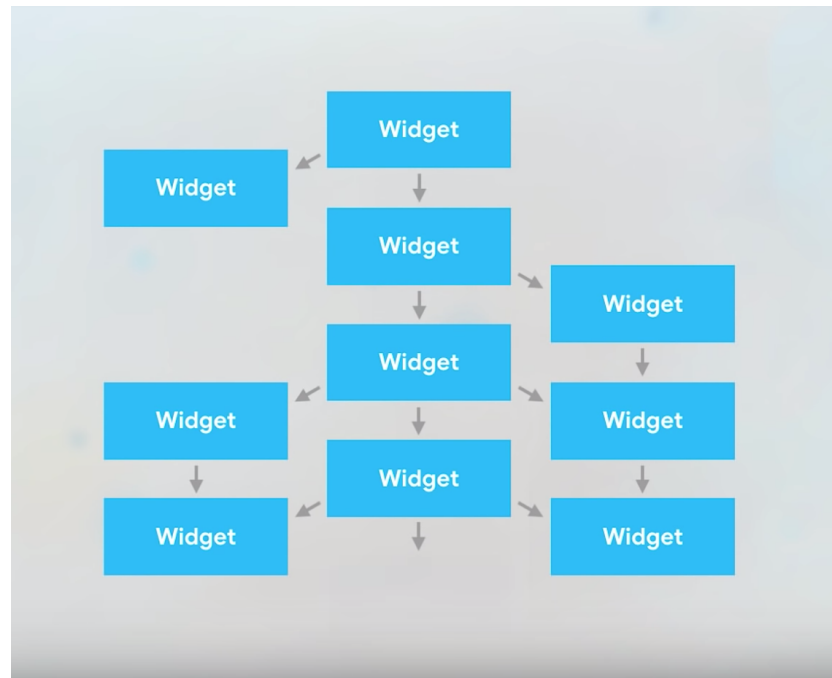
‘widget’ keyword allows access to “parent” widget

<https://api.flutter.dev/flutter/widgets/StatefulWidget-class.html>

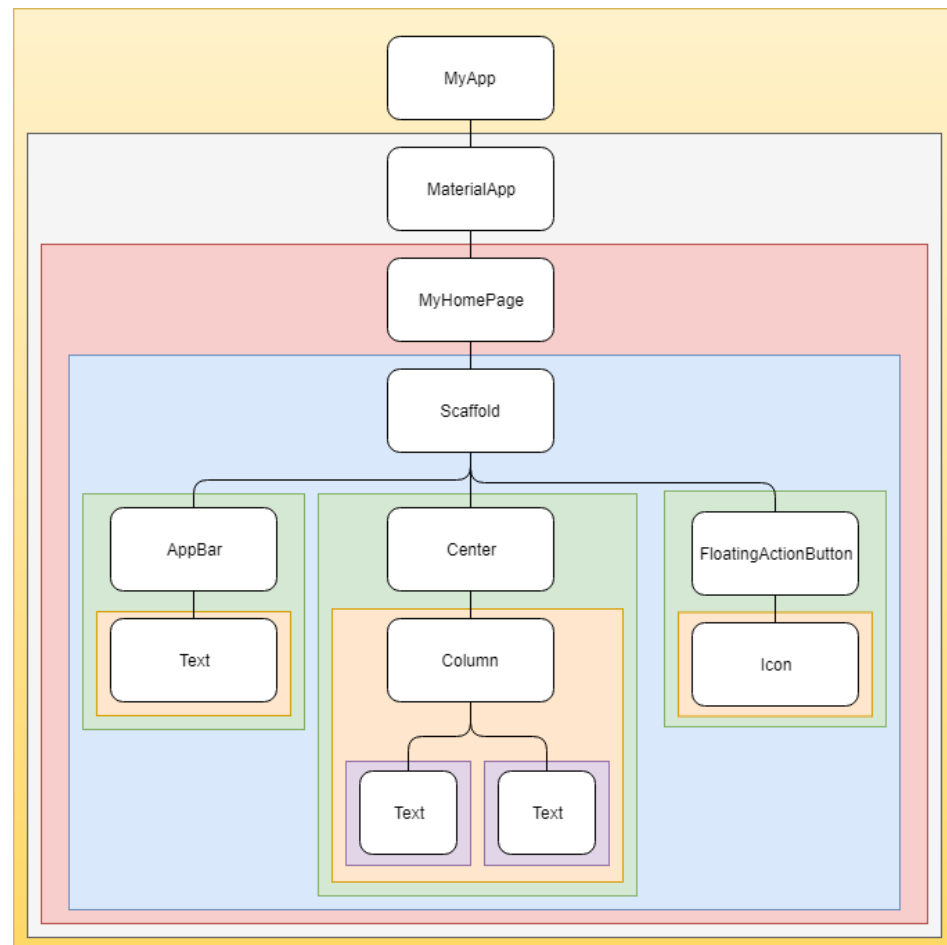
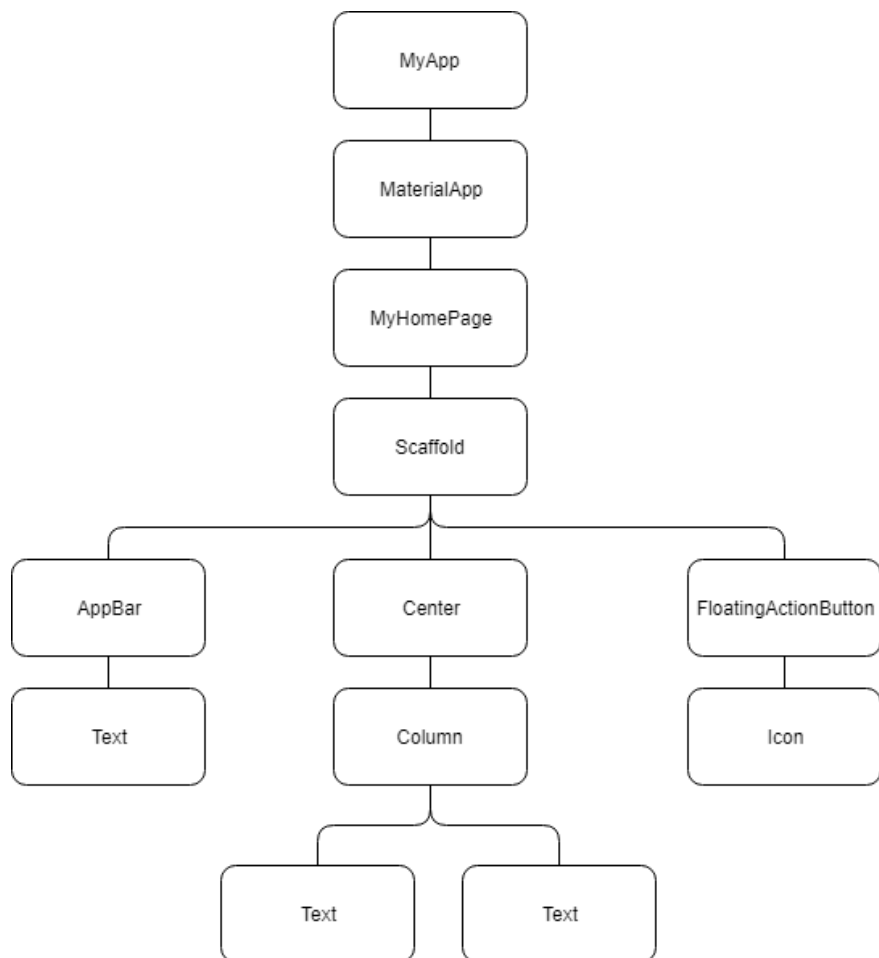
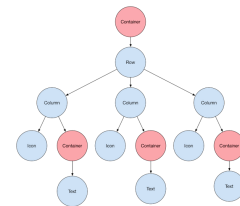
# Recommendation

- Official documentation: docs and videos
- Intro to Widgets
  - <https://flutterbyexample.com/lesson/intro-to-widgets>
  - Informal and clear
- Playlist on youtube
  - Flutter in Focus
  - <https://www.youtube.com/playlist?list=PLjxrf2q8roU2HdJQDjJzOeO6J3FoFLWr2>
  - Flutter Widgets 101
  - [https://www.youtube.com/playlist?list=PLOU2XLYxmsIJyiWUPCou\\_OVTpRIn8UMd](https://www.youtube.com/playlist?list=PLOU2XLYxmsIJyiWUPCou_OVTpRIn8UMd)

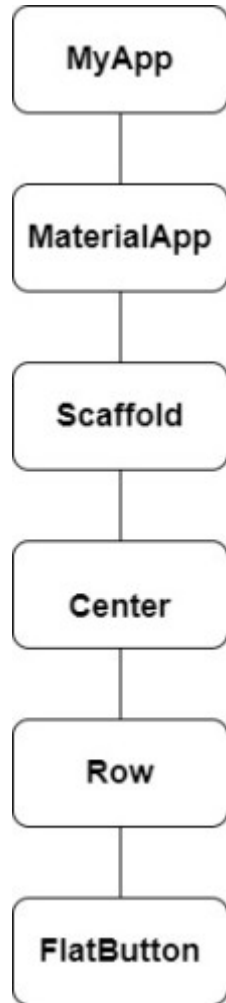
# Tree and Context



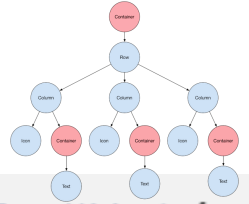
# Tree and context



# Context: parents and child



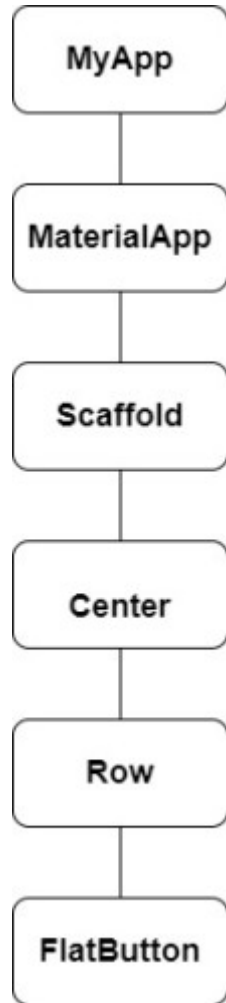
- MaterialApp
  - own BuildContext
  - parent of Scaffold.
- Scaffold
  - own BuildContext
  - parent of Center.
- Center
  - own BuildContext
  - parent of Row.
- Row
  - own BuildContext
  - parent of FlatButton.
- FlatButton
  - own BuildContext.



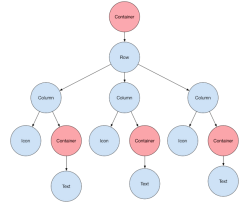
```
class App extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return MaterialApp(  
      // some more code  
    );  
  }  
}
```

<https://blog.mindorks.com/understanding-buildcontext-in-flutter>

# Context visibility

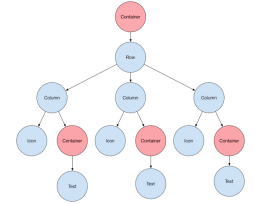


is that widgets are only visible to their BuildContext or to its parent's BuildContext. So, from a child widget, you can locate the parent widget.

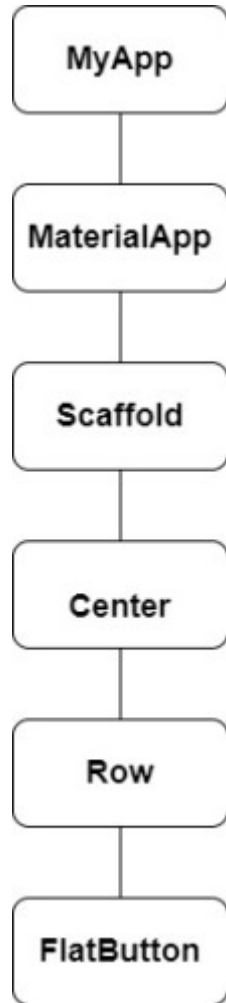


<https://blog.mindorks.com/understanding-buildcontext-in-flutter>





# Navigating the context



For example, if the tree structure is:  
Scaffold > Center > Row > FlatButton, then  
you can get the first Scaffold from  
FlatButton by going up:

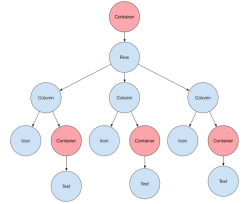
`Theme.of(context)`

Look for Theme in context

`context.ancestorWidgetOfExactType(Scaffold)`

Look for a Scaffold ... picks first.

# Get the “parent”



```
floatingActionButton: new Builder(builder: (context) {  
  return new FloatingActionButton(onPressed: () {  
    Scaffold.of(context).showSnackBar(  
      new SnackBar(  
        backgroundColor: Colors.blue,  
        content: new Text('SnackBar'),  
      ),  
    );  
  });  
}).
```

# Builders



"Most of the time, widgets in Flutter build without looking up any extra information about their parents. **Sometimes, you might write some code where the child widget needs to access the build context of a parent widget in the same build method.** That's what the Builder widget is for!"

# Some concerns

- Build the widget
  - Resources
  - UI
- Access the data needed
  - Local
  - In the tree
  - remote



# Builder class

A platonic widget that calls a closure to obtain its child widget.

See also:

- [StatefulBuilder](#), a platonic widget which also has state.

## Inheritance

[Object](#) > [DiagnosticableTree](#) > [Widget](#) > [StatelessWidget](#) > [Builder](#)

## Constructors

**Builder**({Key key, @required WidgetBuilder builder})

Creates a widget that delegates its build to a callback. [...]

*const*

## Properties

**builder** → [WidgetBuilder](#)

Called to obtain the child widget. [...]

*final*



# Builder class

A platonic widget that calls a closure to obtain its child widget.

See also:

- [StatefulBuilder](#), a platonic widget which also has state.

## Inheritance

[Object](#) > [DiagnosticableTree](#) > [Widget](#) > [StatelessWidget](#) > [Builder](#)

## Constructors

**Builder**({Key key, @required WidgetBuilder builder})

Creates a widget that delegates its build to a callback. [...]

*const*

## Properties

**builder** → [WidgetBuilder](#)

Called to obtain the child widget. [...]

*final*



# Build with the parent context accessible



```
Widget build(BuildContext context) {  
  return Scaffold(  
    body: Builder(  
      builder: (BuildContext context) {  
        return RaisedButton(onPressed: () {  
          Scaffold.of(context).showSnackBar(  
            SnackBar(content: Text('Great!')),  
          );  
        });  
      },  
    );  
  }  
}
```

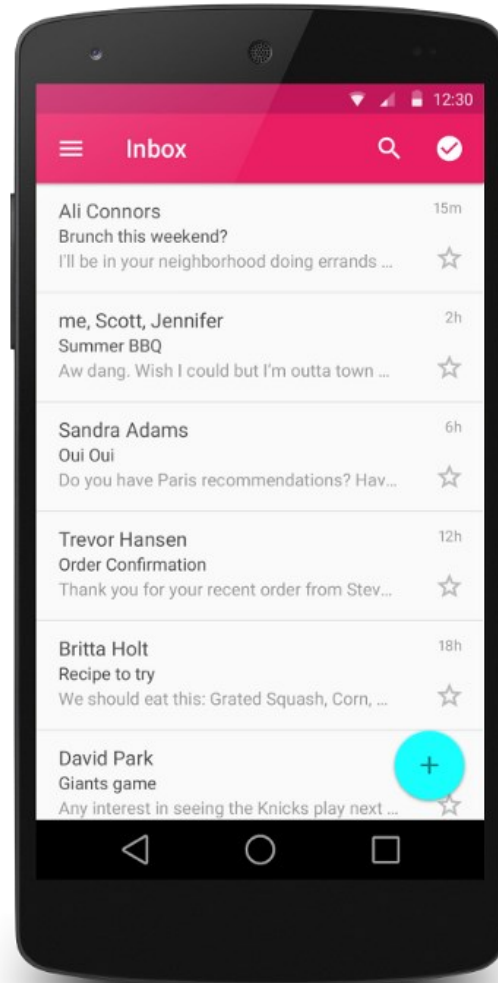
Builder (Flutter Widget of the Week)

<https://www.youtube.com/watch?v=xXNOklusYuA>

# Flutter Lists

Flutter Lists

<https://www.javatpoint.com/flutter-lists>

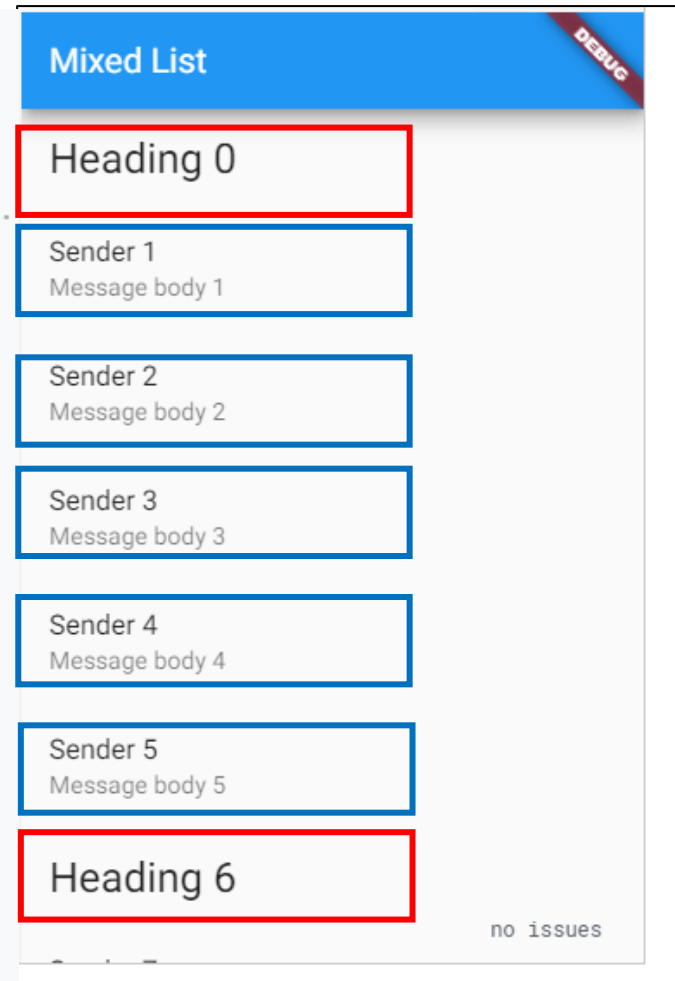




# Create lists with different types of items

[Docs](#) > [Cookbook](#) > [Lists](#) > Create lists with different types of items

```
ListView.builder(  
  // Let the ListView know how many items it needs to build.  
  itemCount: items.length,  
  // Provide a builder function. This is where the magic happens.  
  // Convert each item into a widget based on the type of item it is.  
  itemBuilder: (context, index) {  
    final item = items[index];  
  
    if (item is HeadingItem) {  
      return ListTile(  
        title: Text(  
          item.heading,  
          style: Theme.of(context).textTheme.headline,  
        ),  
      );  
    }  
    else if (item is MessageItem) {  
      return ListTile(  
        title: Text(item.sender),  
        subtitle: Text(item.body),  
      );  
    }  
  },  
);
```



<https://flutter.dev/docs/cookbook/lists/mixed-list>

# Create lists with different types of items

[Docs](#) > [Cookbook](#) > [Lists](#) > Create lists with different types of items

Dart

Format Reset Run

```
1 import 'package:flutter/foundation.dart';
2 import 'package:flutter/material.dart';
3
4 void main() {
5   runApp(MyApp(
6     items: List<ListItem>.generate(
7       1000,
8       (i) => i % 6 == 0
9         ? HeadingItem("Heading $i")
10        : MessageItem("Sender $i", "Message body $i"),
11     ),
12   ));
13 }
14
15 class MyApp extends StatelessWidget {
16   final List<ListItem> items;
17
18   MyApp({Key key, @required this.items}) : super(key: key);
19
20   @override
21   Widget build(BuildContext context) {
22     final title = 'Mixed List';
23
24     return MaterialApp(
25       title: title,
26       home: Scaffold(
27         appBar: AppBar(
28           title: Text(title),
```

Mixed List

DEBUG

Heading 0

Sender 1  
Message body 1

Sender 2  
Message body 2

Sender 3  
Message body 3

Sender 4  
Message body 4

Sender 5  
Message body 5

Heading 6

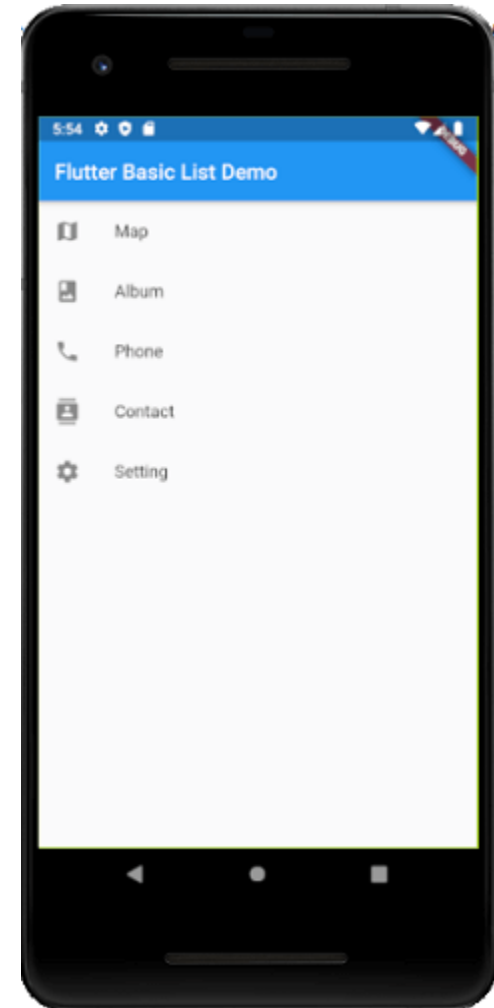
no issues

Console

<https://flutter.dev/docs/cookbook/lists/mixed-list>

# Flutter Lists & ListView

- Lists
  - most popular elements of every web or mobile application.
  - made up of multiple rows of items, which include text, buttons, toggles, icons, thumbnails, and many more.
- ListView
  - for working with Lists, which is the fundamental concept of displaying data in the mobile apps.
  - displaying lists that contains only a few items.
  - includes **ListTitle**



# ListView

## Basic lists / static data

- Display collection of items known during code

ListView creates all items at once,

```
ListView(  
  children: <Widget>[  
    Text("Element 1"),  
    Text("Element 2")  
  ],  
)
```

## Long lists / Dynamic data

- list with repeating blocks with different data in each one.
- `ListView.builder()` constructor creates items when they are scrolled onto the screen.

```
ListView.builder(  
  itemBuilder: (context, position) {  
    return Card(  
      child:  
        Text(androidVersionNames[position]),  
    );  
  },  
  itemCount: androidVersionNames.length,  
)
```

jul2018

The Flutter Series: Lists and Grids (RecyclerViews in Flutter)

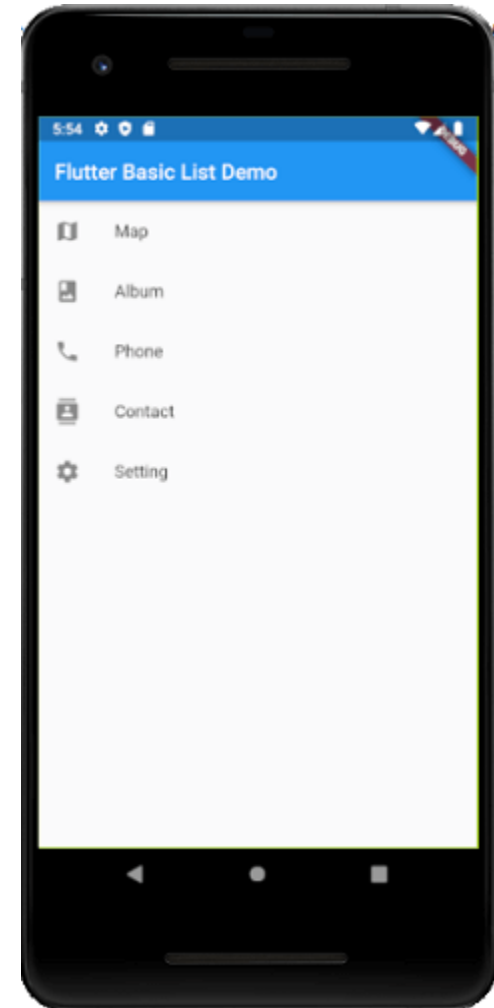
<https://medium.com/@dev.n/the-complete-flutter-series-article-3-lists-and-grids-in-flutter-b20d1a393e39>

# The base MyApp, we are changing the body

```
import 'package:flutter/material.dart';  
void main() => runApp(MyApp());  
class MyApp extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    final appTitle = 'Flutter Basic List Demo';  
    return MaterialApp(  
      title: appTitle,  
      home: Scaffold(  
        appBar: AppBar(  
          title: Text(appTitle),  
        ),
```

# Basic Lists

```
body: ListView(  
  children: <Widget>[  
    ListTile(  
      leading: Icon(Icons.map),  
      title: Text('Map'),  
    ),  
    ListTile(  
      leading: Icon(Icons.photo_album),  
      title: Text('Album'),  
    ),  
    ListTile(  
      leading: Icon(Icons.phone),  
      title: Text('Phone'),  
    ),  
    ListTile(  
      leading: Icon(Icons.contacts),  
      title: Text('Contact'),  
    ),  
    ListTile(  
      leading: Icon(Icons.settings),  
      title: Text('Setting'),  
    ),  
  ],  
)
```



# Listview builder

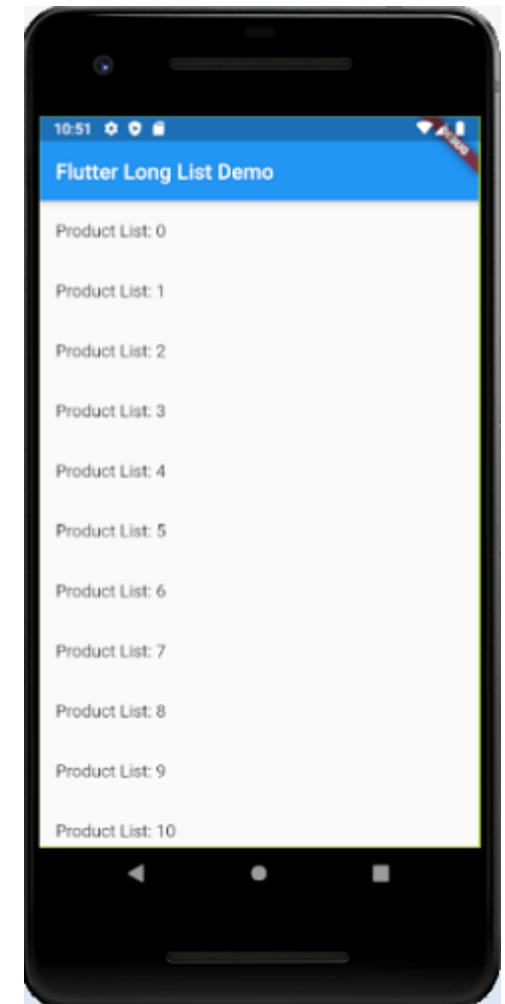

- There are two properties inside the builder
  - **itemCount** is pretty straightforward, it asks how many repeating items you want to display in the list.
  - **itemBuilder** is where you return the item itself that you want to display. Here we made a card with a simple text widget inside. Item builder expects a lambda function which has the parameters of context and position. Position gives you which index of the list it is.

```
ListView.builder(  
    itemBuilder: (context, position) {  
        return Card(  
            child:  
Text(androidVersionNames[position]),  
        );  
    },  
    itemCount: androidVersionNames.length,  
)
```

# The constructor

```
import 'package:flutter/material.dart';  
void main() {  
  runApp(MyApp(  
    products: List<String>.generate(500, (i) => "Product List: $i"),  
  ));  
}  
  
class MyApp extends StatelessWidget {  
  final List<String> products;  
  MyApp({Key key, @required this.products}) : super(key: key);
```

Just passed a list to my app





# Item builder

@override

```
Widget build(BuildContext context) {
```

```
  final appTitle = 'Flutter Long List Demo';
```

```
  return MaterialApp(
```

```
    title: appTitle,
```

```
    home: Scaffold(
```

```
      appBar: AppBar(
```

```
        title: Text(appTitle),
```

```
      ),
```

```
      body: ListView.builder(
```

```
        itemCount: products.length,
```

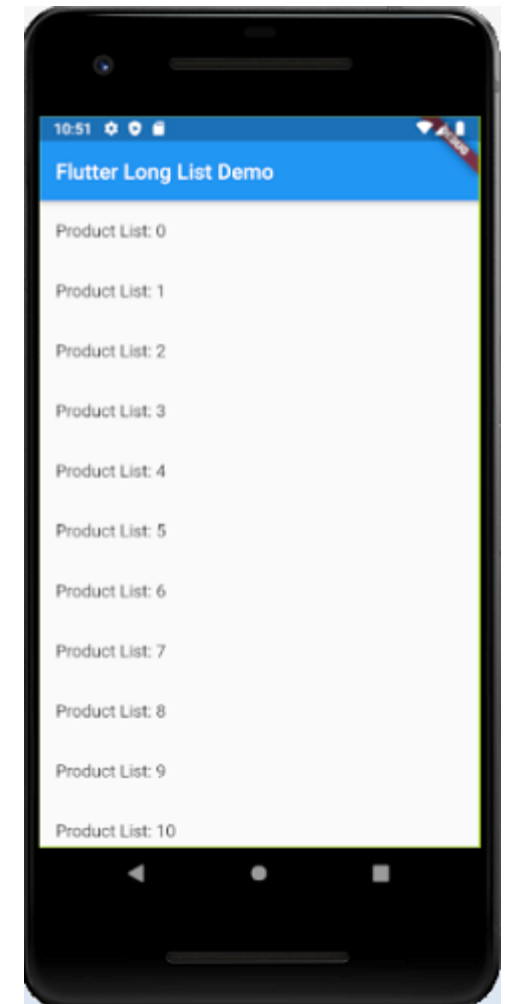
```
        itemBuilder: (context, index) {
```

```
          return ListTile(
```

```
            title: Text('${products[index]}'),
```

```
          );    },    ),    ),    );  } }
```

Using the list to build a list

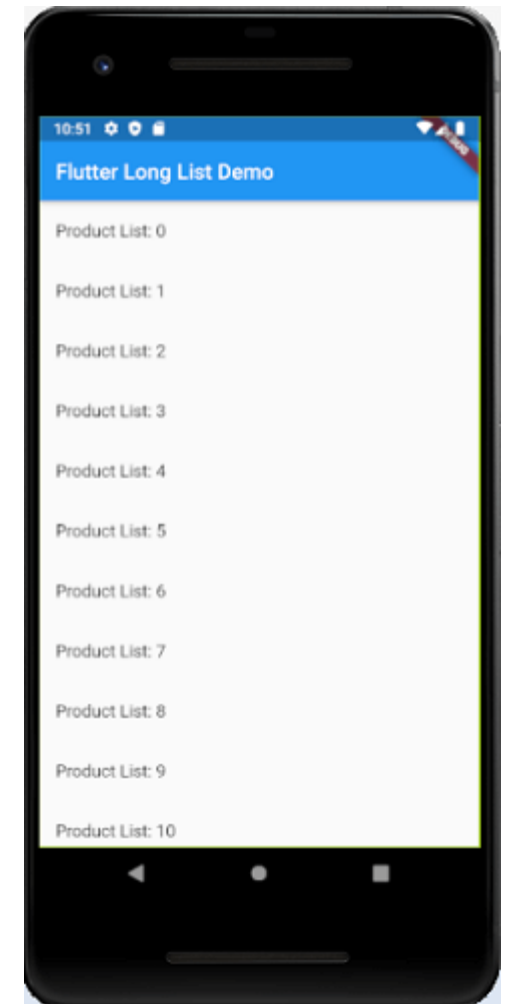


# Item builder

@override

```
Widget build(BuildContext context) {  
  final appTitle = 'Flutter Long List Demo';  
  return MaterialApp(  
    title: appTitle,  
    home: Scaffold(  
      appBar: AppBar(  
        title: Text(appTitle),  
      ),  
      body: ListView.builder(  
        itemCount: products.length,  
        itemBuilder: (context, index) {  
          return ListTile(  
            title: Text('${products[index]}'),  
            );    },    ),    );  } }
```

Using the list to build a list  
With products.length elements



# Item builder

@override

```
Widget build(BuildContext context) {
```

```
  final appTitle = 'Flutter Long List Demo';
```

```
  return MaterialApp(
```

```
    title: appTitle,
```

```
    home: Scaffold(
```

```
      appBar: AppBar(
```

```
        title: Text(appTitle),
```

```
      ),
```

```
      body: ListView.builder(
```

```
        itemCount: products.length,
```

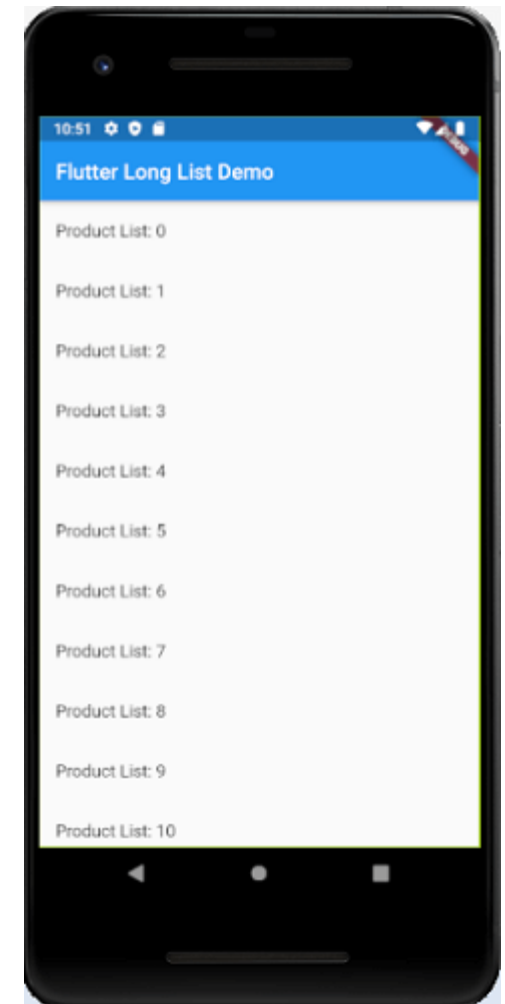
```
        itemBuilder: (context, index) {
```

```
          return ListTile(
```

```
            title: Text('${products[index]}'),
```

```
          );    },    ),    ),    );  } }
```

Using the list to build a list  
With products.length elements  
& a build entry in a given index



# Item builder

@override

```
Widget build(BuildContext context) {
```

```
  final appTitle = 'Flutter Long List Demo';
```

```
  return MaterialApp(
```

```
    title: appTitle,
```

```
    home: Scaffold(
```

```
      appBar: AppBar(
```

```
        title: Text(appTitle),
```

```
      ),
```

```
      body: ListView.builder(
```

```
        itemCount: products.length, Using products[index]
```

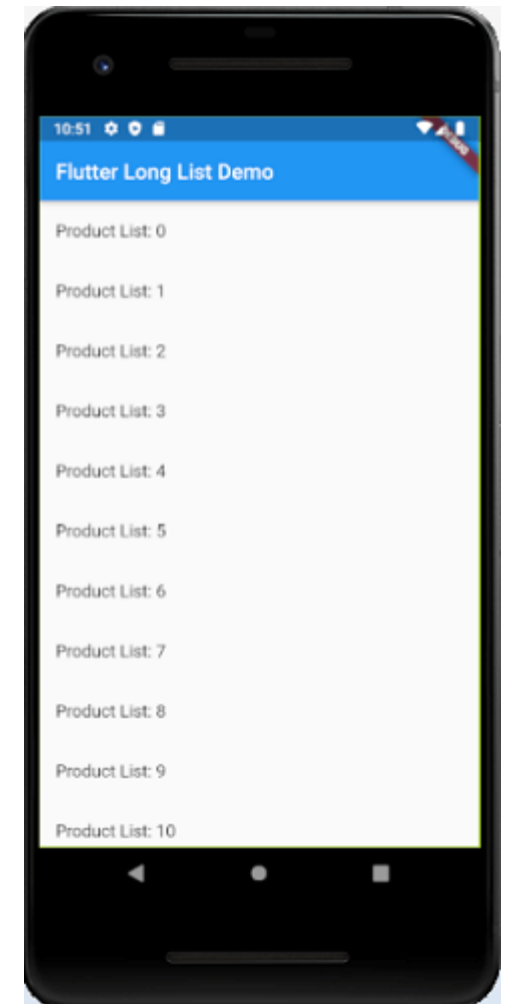
```
        itemBuilder: (context, index) {
```

```
          return ListTile(
```

```
            title: Text('${products[index]}'),
```

```
          );    },    ),    ),    );  }
```

Using the list to build a list  
With products.length elements  
& a build entry in a given index



# FutureBuilder<T> class



```
class _MyStatefulWidgetState extends State<MyStatefulWidget> {  
  Future<String> _calculation = Future<String>.delayed(  
    Duration(seconds: 2),  
    () => 'Data Loaded',  
  );  
  
  Widget build(BuildContext context) {  
    return DefaultTextStyle(  
      style: Theme.of(context).textTheme.headline2,  
      textAlign: TextAlign.center,  
      child: FutureBuilder<String>(  
        future: _calculation, // a previously-obtained Future<String> or null  
        builder: (BuildContext context, AsyncSnapshot<String> snapshot) {  
          List<Widget> children;  
          if (snapshot.hasData) {  
            children = <Widget>[  
              Icon(  
                Icons.check_circle_outline,  
                color: Colors.green,  
                size: 60,  
              ),  
            ];  
          }  
        },  
      ),  
    );  
  }  
}
```



Widget that builds itself based on the latest snapshot of interaction with a Future.

<https://api.flutter.dev/flutter/widgets/FutureBuilder-class.html>

[https://www.youtube.com/watch?v=ek8ZPdWj4Qo&feature=emb\\_logo](https://www.youtube.com/watch?v=ek8ZPdWj4Qo&feature=emb_logo)



# FutureBuilder<T> class

```
class _MyStatefulWidgetState extends State<MyStatefulWidget> {  
  Future<String> _calculation = Future<String>.delayed(  
    Duration(seconds: 2)  
    () => 'Data Loaded',  
  );  
  
  Widget build(BuildContext context) {  
    return DefaultTextStyle(  
      style: Theme.of(context).textTheme.headline2,  
      textAlign: TextAlign.center,  
      child: FutureBuilder<String>(  
        future: _calculation, // a previously-obtained Future<String> or null  
        builder: (BuildContext context, AsyncSnapshot<String> snapshot) {  
          List<Widget> children;  
          if (snapshot.hasData) {  
            children = <Widget>[  
              Icon(  
                Icons.check_circle_outline,  
                color: Colors.green,  
                size: 60,  
              ),  
            ];  
          }  
        }  
      ),  
    );  
  }  
}
```

Expect one result ( the Future)

incoming

Do I have data?

Widget that builds itself based on the latest snapshot of interaction with a Future.

<https://api.flutter.dev/flutter/widgets/FutureBuilder-class.html>

[https://www.youtube.com/watch?v=ek8ZPdWj4Qo&feature=emb\\_logo](https://www.youtube.com/watch?v=ek8ZPdWj4Qo&feature=emb_logo)

# StreamBuilder<T> class



```
class _MyStatefulWidgetState extends State<MyStatefulWidget> {  
  Stream<int> _bids = (() async* {  
    await Future<void>.delayed(Duration(seconds: 1));  
    yield 1;  
    await Future<void>.delayed(Duration(seconds: 1));  
  })();  
  
  Widget build(BuildContext context) {  
    return DefaultTextStyle(  
      style: Theme.of(context).textTheme.headline2,  
      textAlign: TextAlign.center,  
      child: Container(  
        alignment: FractionalOffset.center,  
        color: Colors.white,  
        child: StreamBuilder<int>(  
          stream: _bids,  
          builder: (BuildContext context, AsyncSnapshot<int> snapshot) {  
            List<Widget> children;  
            if (snapshot.hasError) {  
              children = <Widget>[  
                Icon(  
                  Icons.error_outline,  

```

Widget that builds itself based on the latest snapshot of interaction with a Stream.

<https://api.flutter.dev/flutter/widgets/StreamBuilder-class.html>

[https://www.youtube.com/watch?time\\_continue=1&v=MkKEWHfy99Y&feature=emb\\_logo](https://www.youtube.com/watch?time_continue=1&v=MkKEWHfy99Y&feature=emb_logo)

# StreamBuilder<T> class



```
class _MyStatefulWidgetState extends State<MyStatefulWidget> {  
  Stream<int> _bids = (() async* {  
    await Future<void>.delayed(Duration(seconds: 1));  
    yield 1;  
    await Future<void>.delayed(Duration(seconds: 1));  
  })();  
  
  Widget build(BuildContext context) {  
    return DefaultTextStyle(  
      style: Theme.of(context).textTheme.headline2,  
      textAlign: TextAlign.center,  
      child: Container(  
        alignment: FractionalOffset.center,  
        color: Colors.white,  
        child: StreamBuilder<int>(  
          stream: _bids,  
          builder: (BuildContext context, AsyncSnapshot<int> snapshot) {  
            List<Widget> children:  
            if (snapshot.hasError) {  
              children = <Widget>[  
                Icon(  
                  Icons.error_outline,  

```

Expect a flow of data – stream of int

incoming

Do I have error?  
Do I have data?

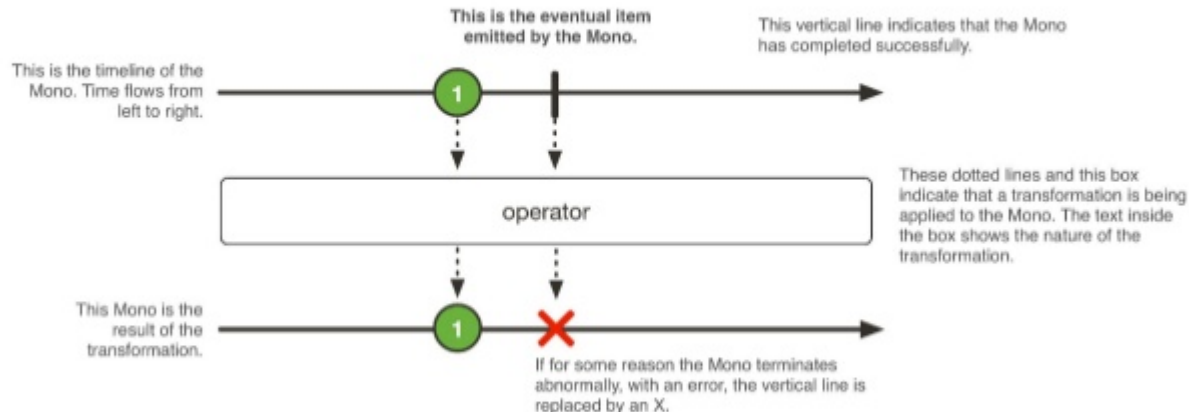
Widget that builds itself based on the latest snapshot of interaction with a Stream.

<https://api.flutter.dev/flutter/widgets/StreamBuilder-class.html>

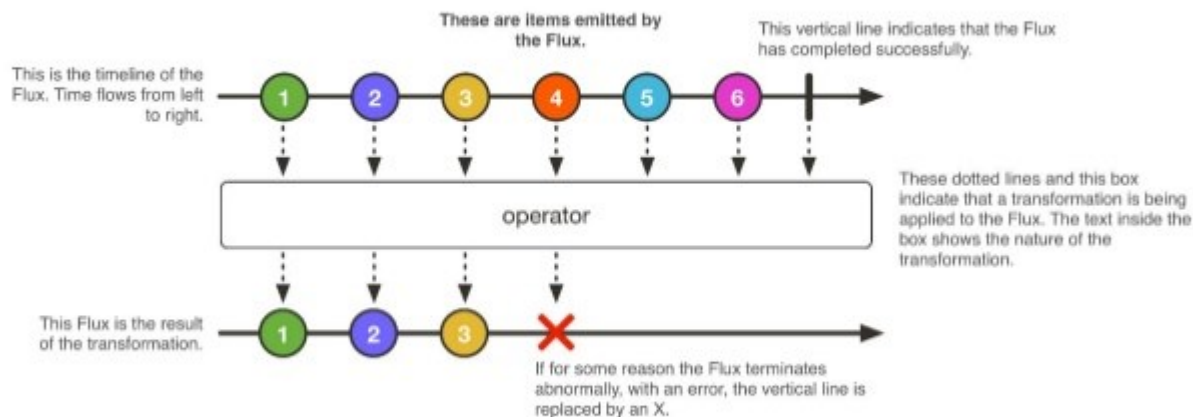
[https://www.youtube.com/watch?time\\_continue=1&v=MkKEWHfy99Y&feature=emb\\_logo](https://www.youtube.com/watch?time_continue=1&v=MkKEWHfy99Y&feature=emb_logo)



# Mono is a reactive type for 0..1 element

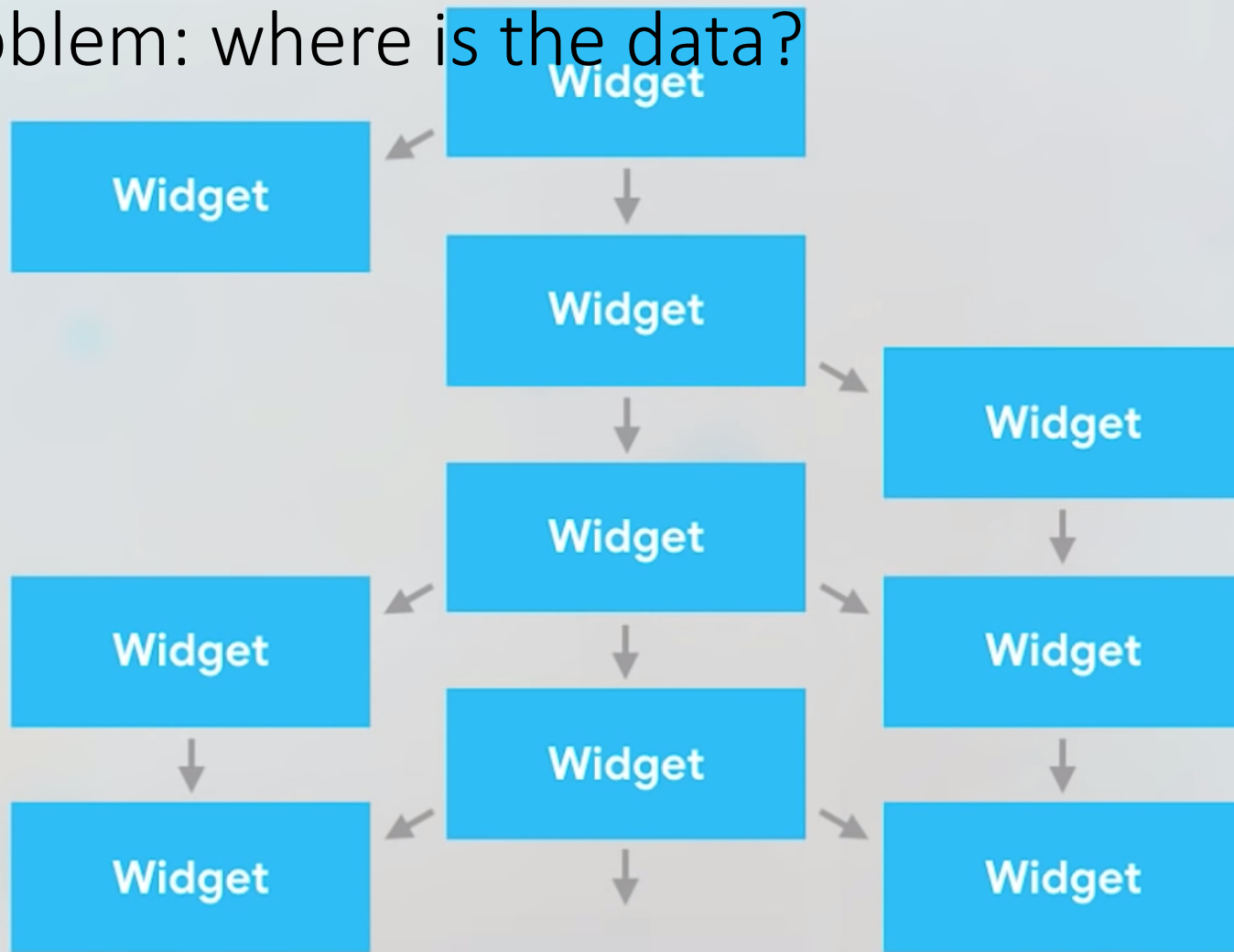


# Flux is for reactive collection and stream

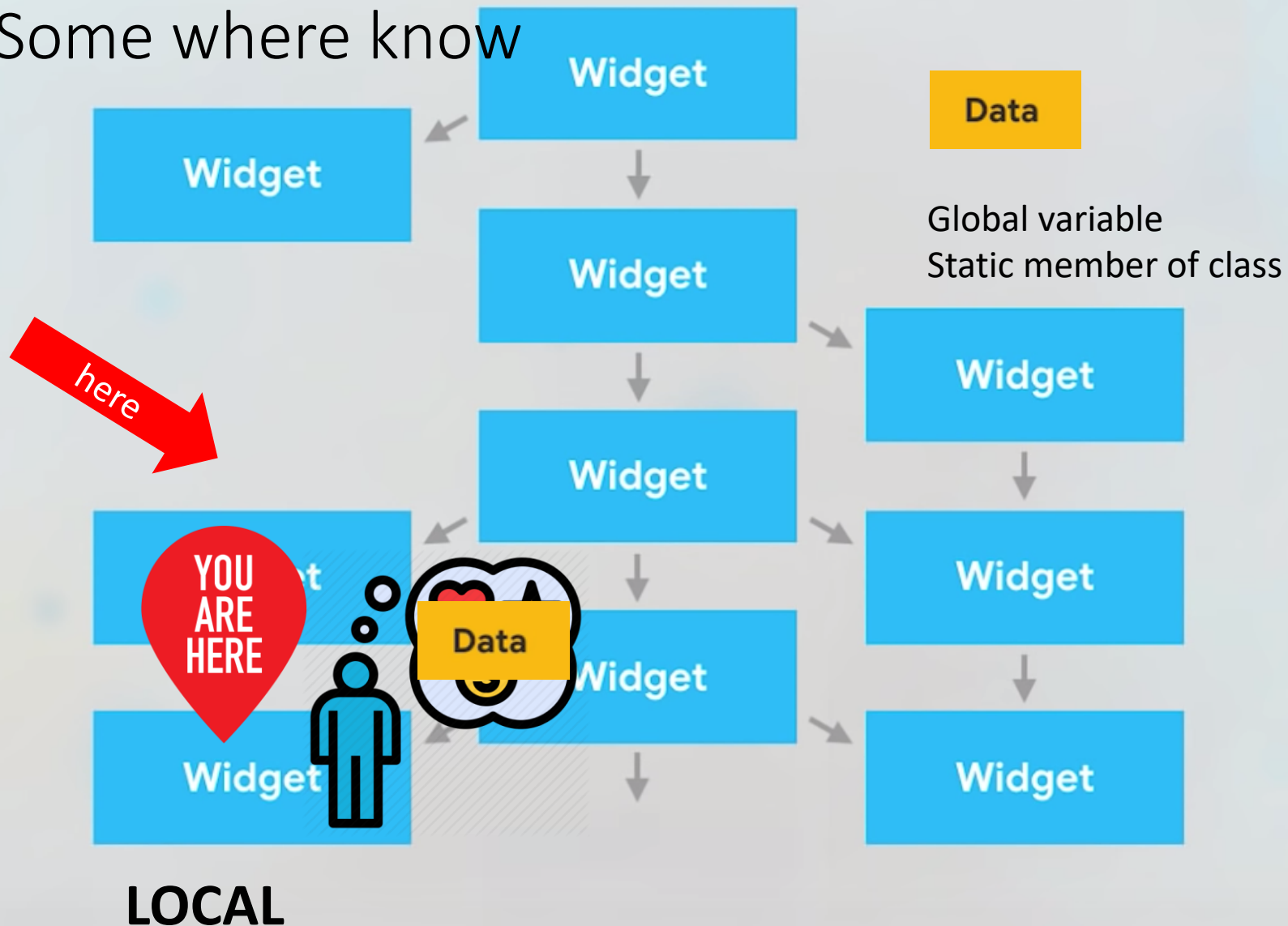


Next step:  
Where is the data?

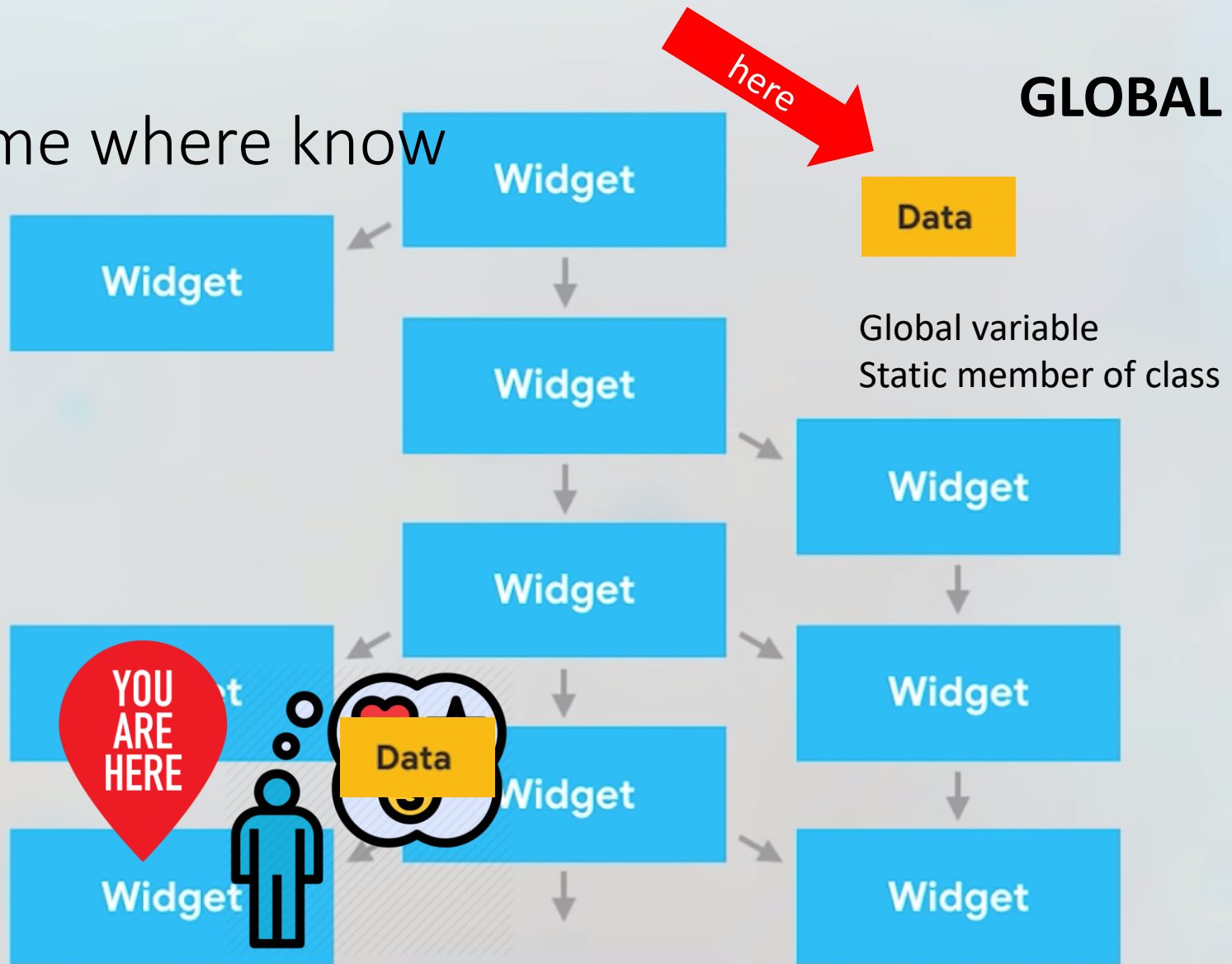
Problem: where is the data?



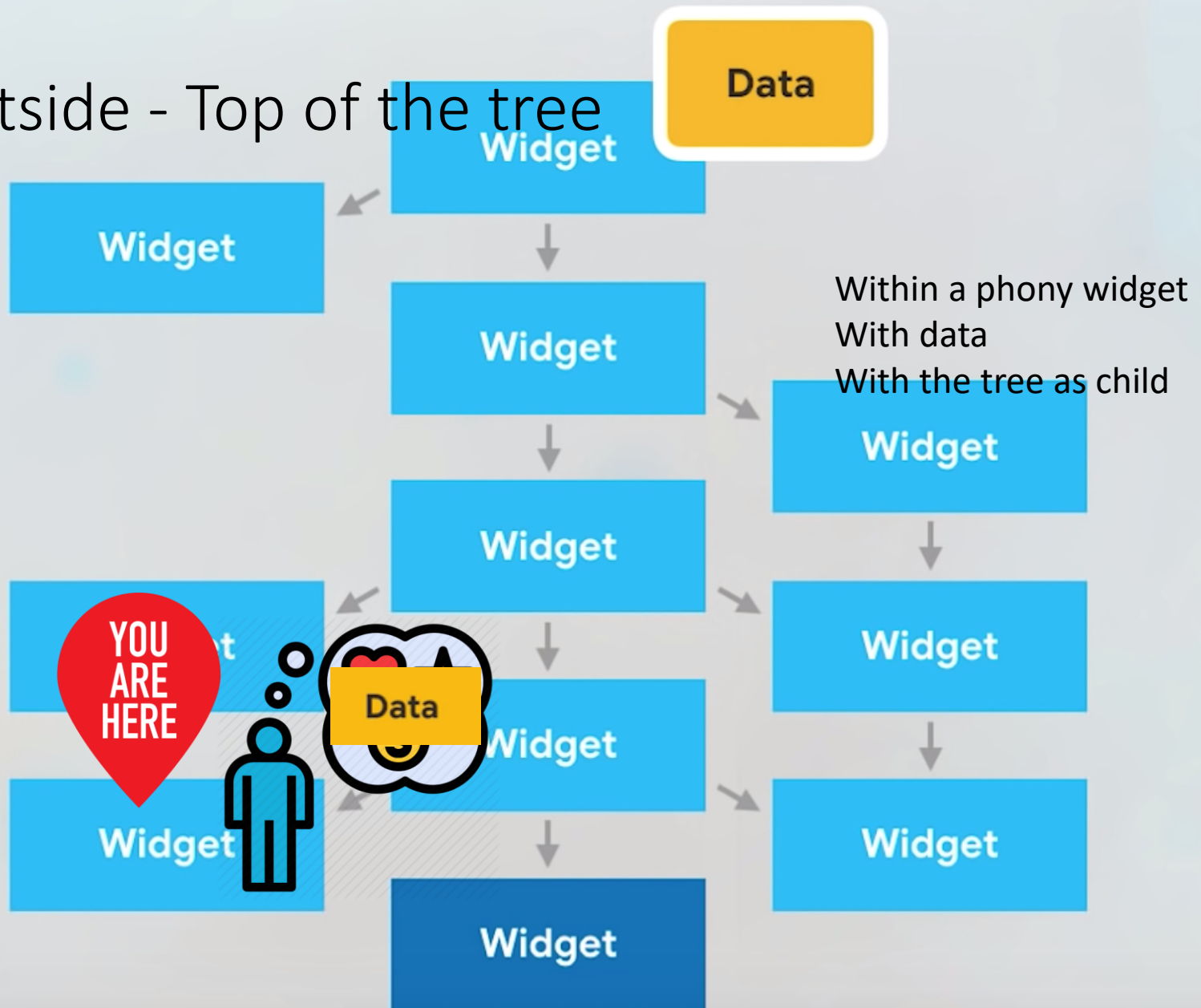
Some where know



Some where know



# Outside - Top of the tree



# State management in Flutter

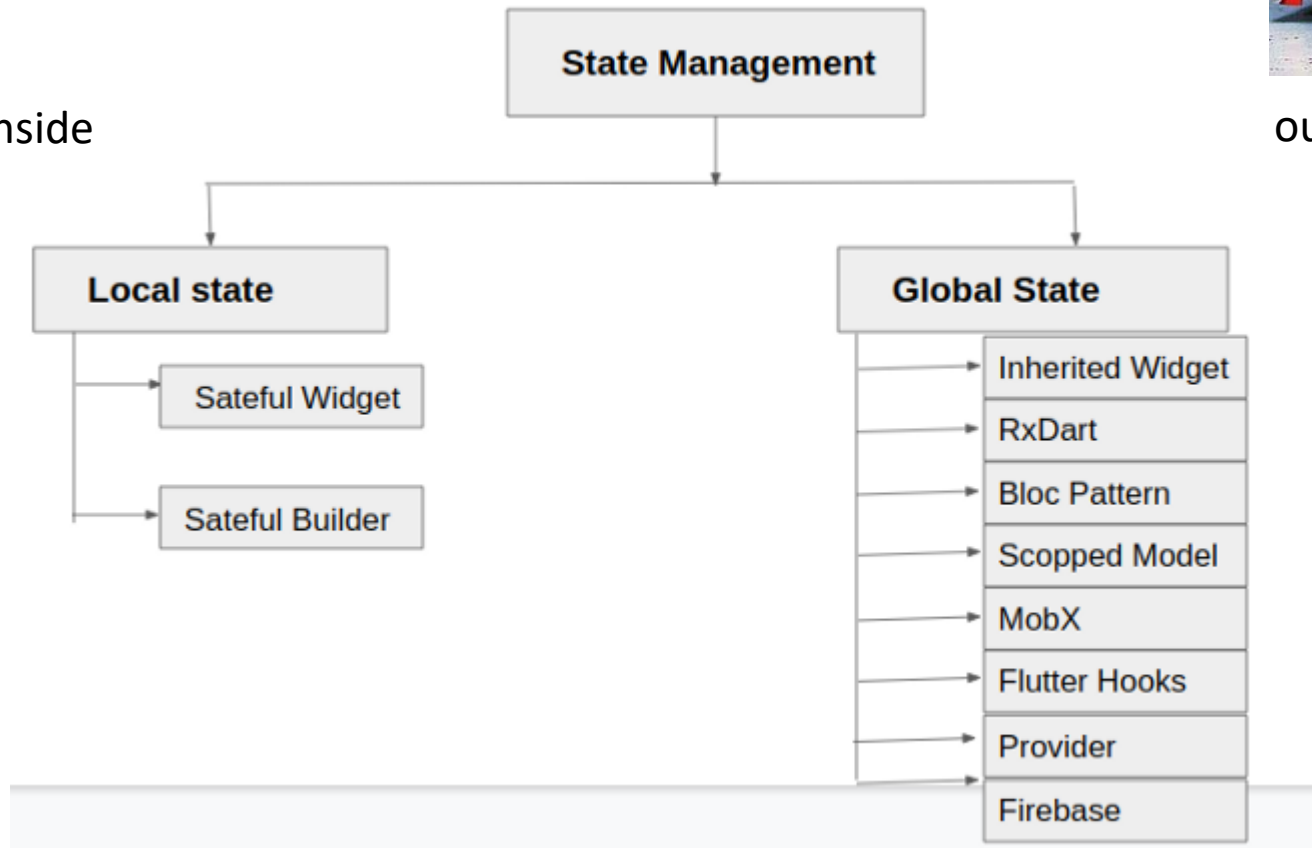


Kendi J. [Follow](#)  
Feb 8 · 3 min read



Inside

outside



<https://medium.com/@kendyjaky/state-management-in-flutter-fe2641981ae5>

# The END

