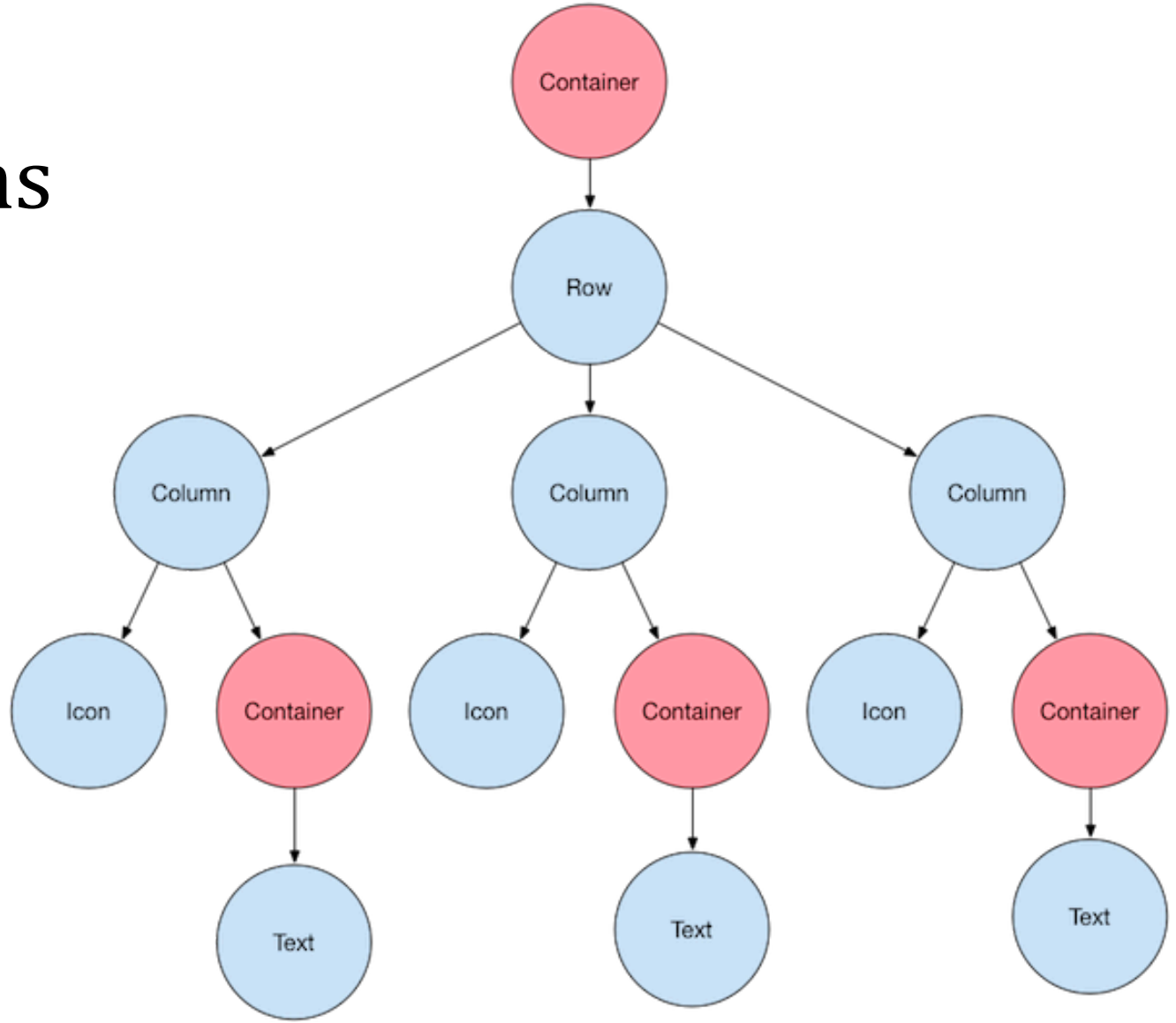
Building Layouts

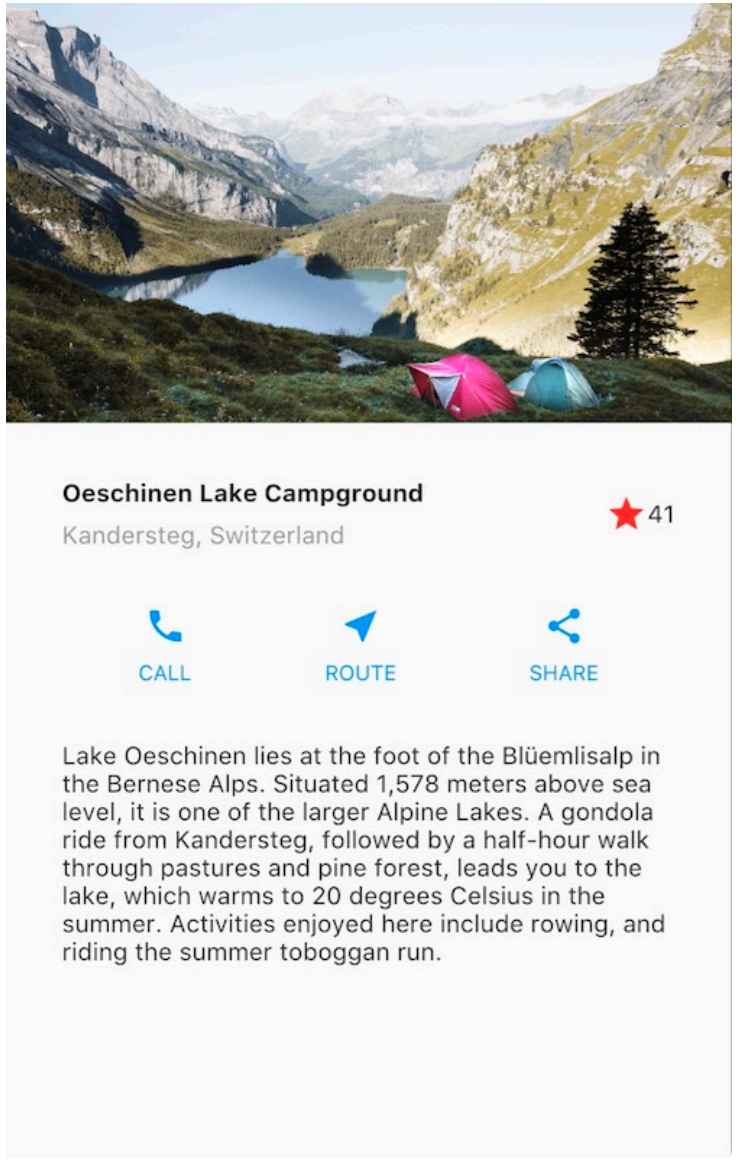
Contents

- Layouts in Flutter
- Nesting rows and columns
- Sizing widgets
- Summary



Layouts in Flutter (1)

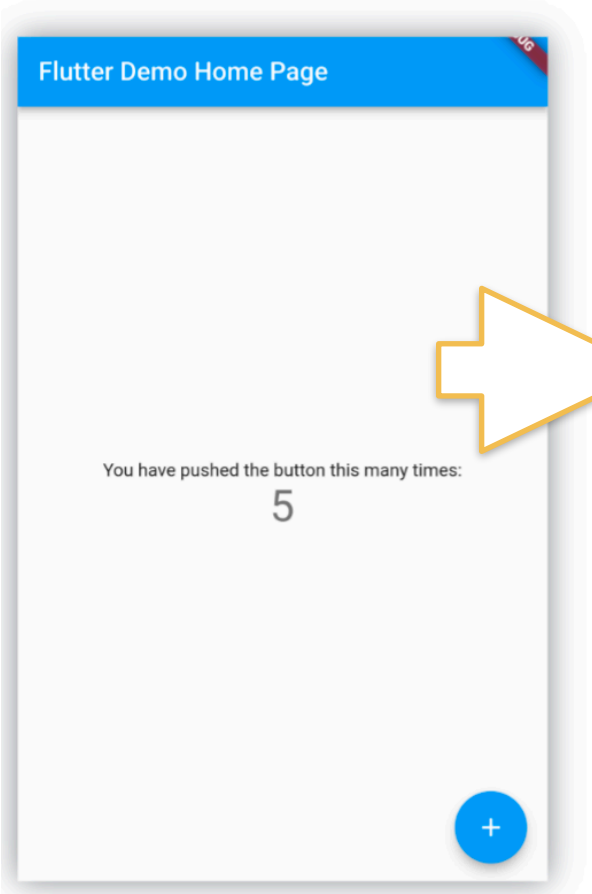
- Widgets are classes used to **build UIs**.
- Widgets are used for both **layout** and **UI elements**
  - The images, icons, text and etc… that you **see** in a Flutter app are all widgets (**UI elements**).
  - But things you **don't see** are also widgets, such as the rows, columns, and grids that **arrange**, **constraint**, and **align** the visible widgets (**layout**).



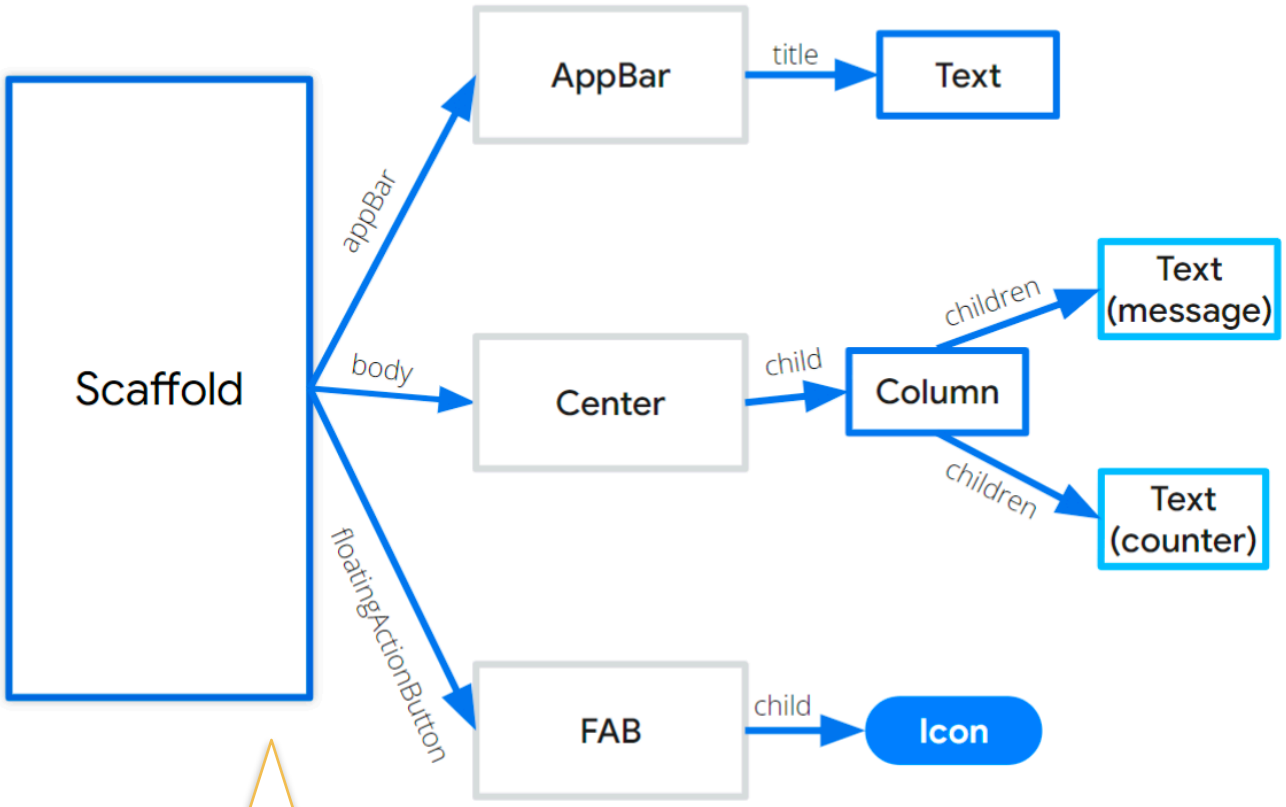
Layouts in Flutter (2)

- Example

UI



Breakdown

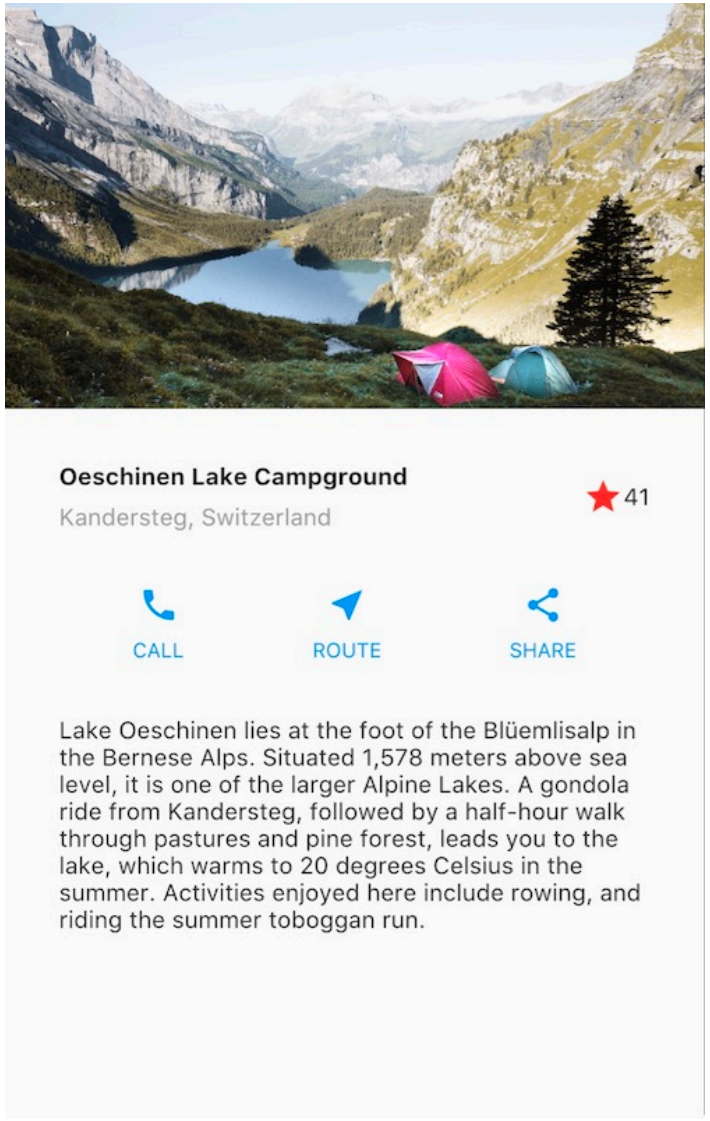


UI elements: Text, Icon

Layout: Scaffold, AppBar, Center, Column, FAB

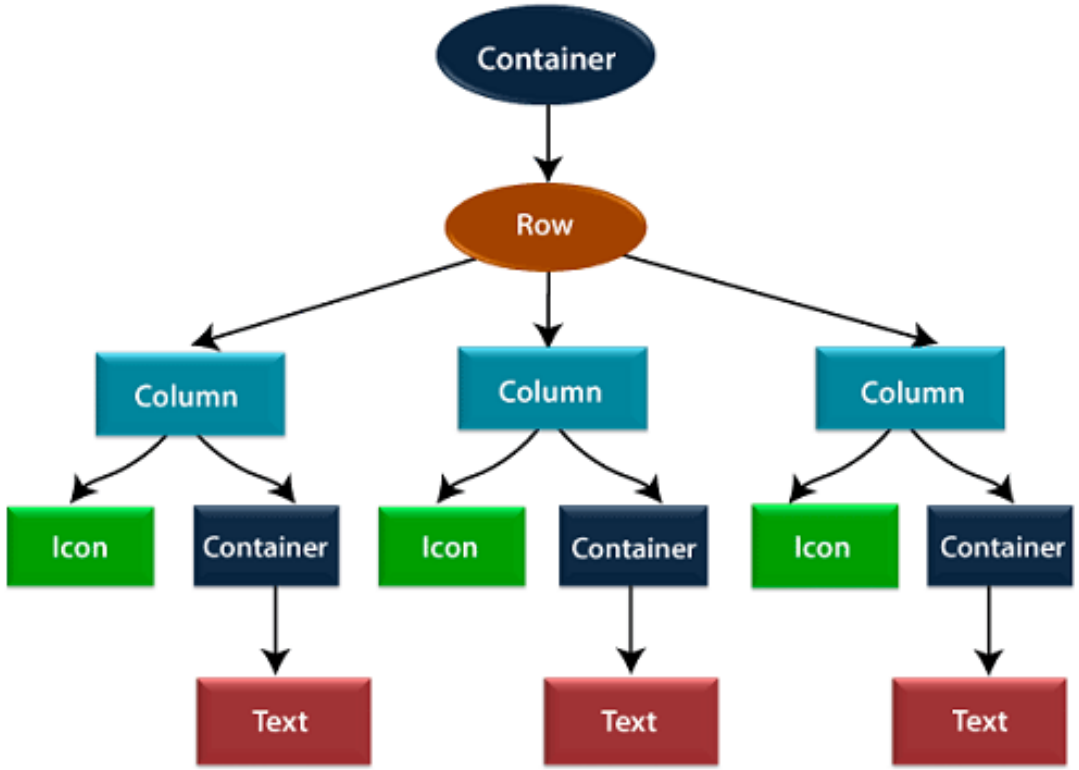
How to Build an UI (1)

- Step 1: First, you need to select a **Layout** widget.
- Step 2: Create a visible widget (**UI elements**).
- Step 3: Add the **visible widget** to the **layout widget.**
- Step 4: Finally, **add the layout widget to the page** where you want to display.



How to Build an UI (2)

- Example:



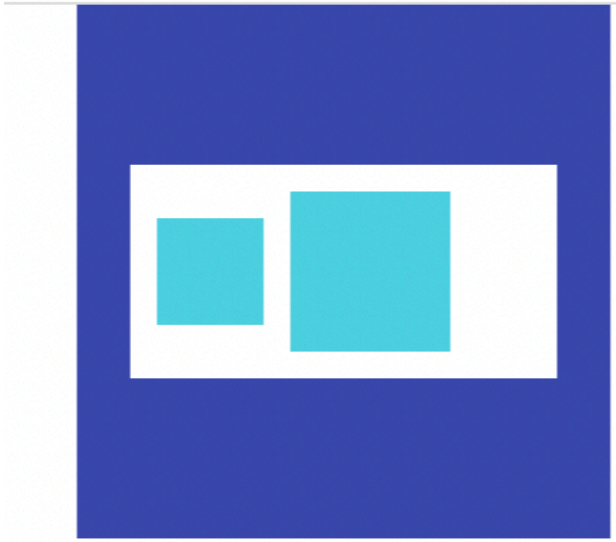
- Layout widget: **Container, Row, Column**
- UI elements: **Icon, Text**

How to Build an UI (3)

- **Step 1: Select a layout widget**
  - Based on how you **want to align**

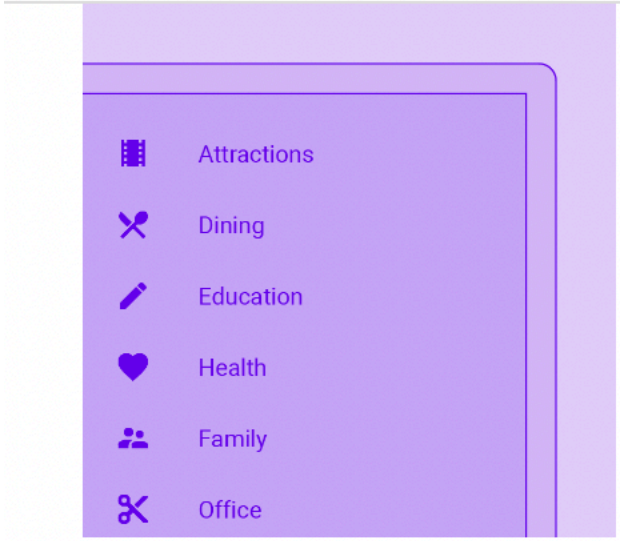
Column

Layout a list of child widgets in the vertical direction.



Row

Layout a list of child widgets in the horizontal direction.

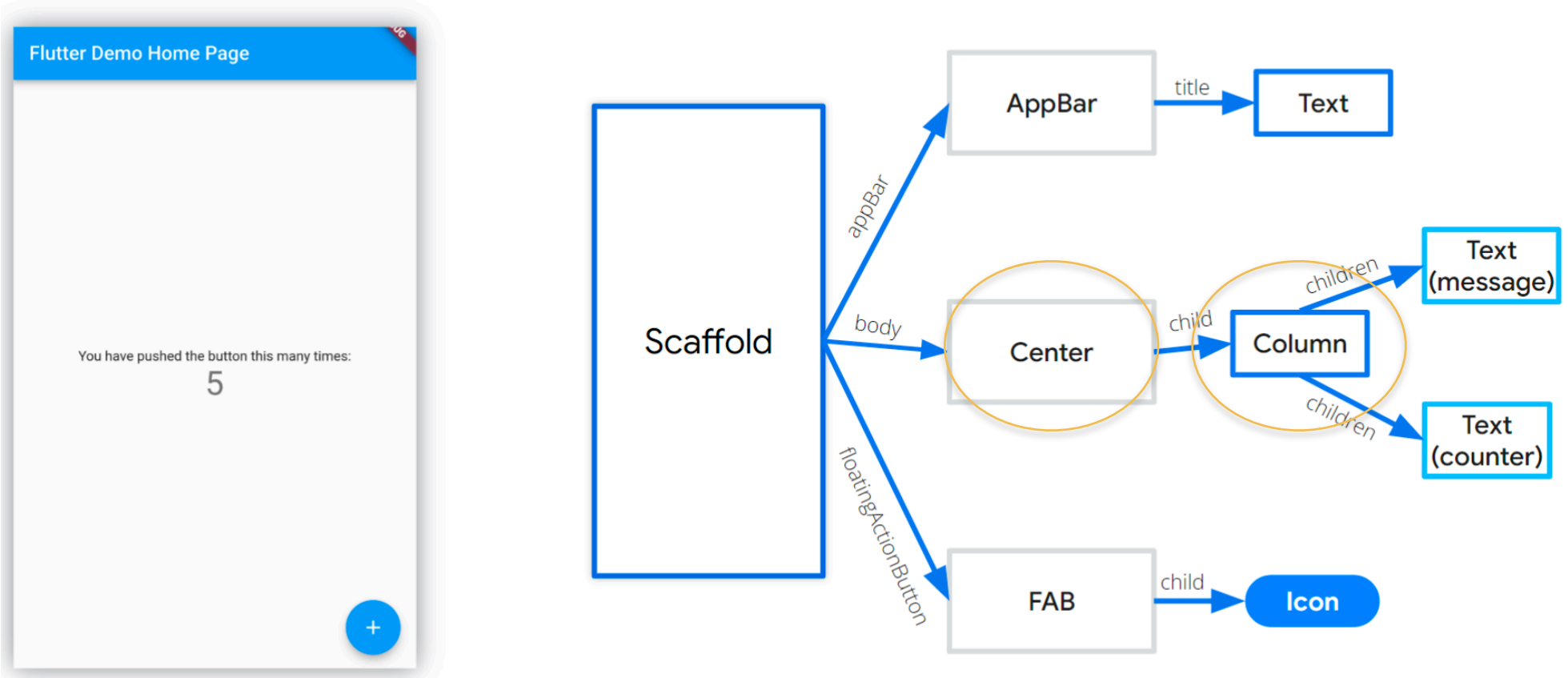


ListView

A scrollable, linear list of widgets. ListView is the most commonly used scrolling widget. It displays its children one after another in the scroll direction....

How to Build an UI (4)

- **Step 1: Select a layout widget** **=> Column**



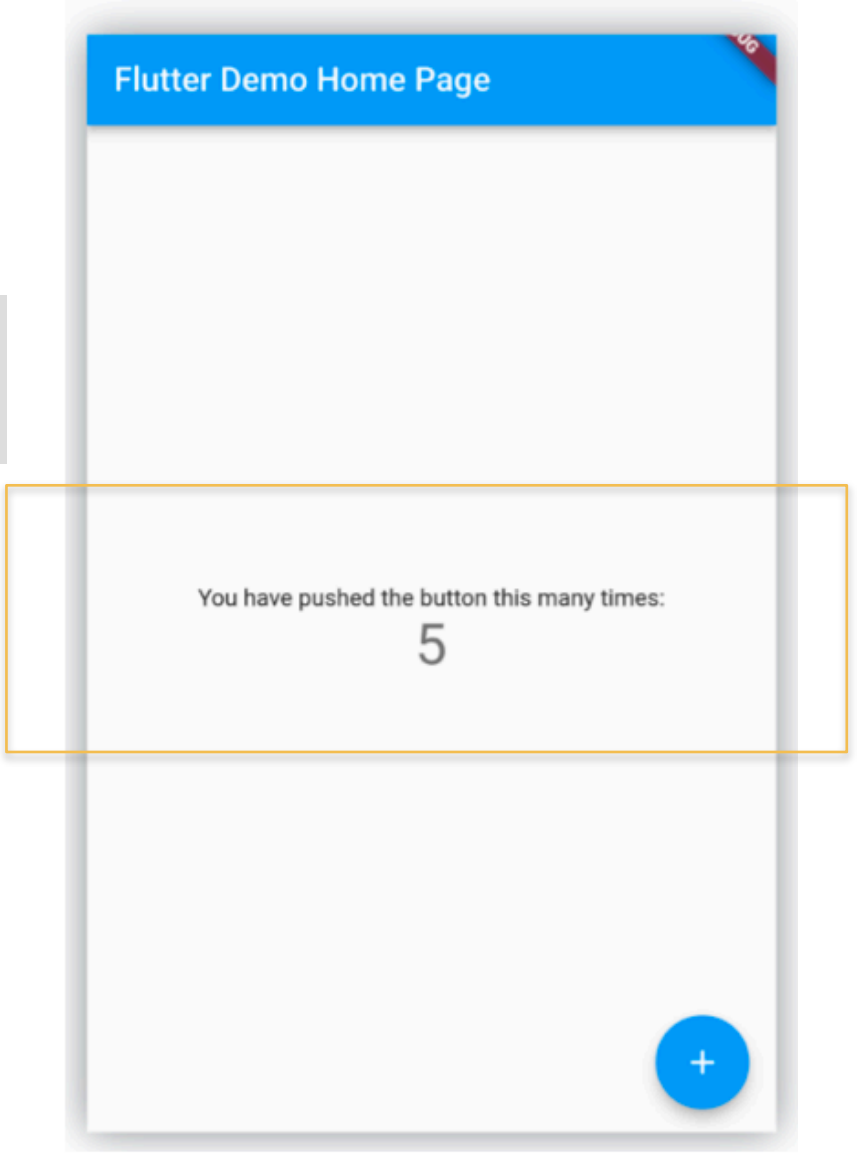
How to Build an UI (5)

- **Step 2: Create a visible widget**
  - For example, create a **Text** widget:

```
Text('You have pushed the button this many times'),
Text('5')
```

  - For example, create an **Icon** widget:

```
Icon(
  Icons.add,
),
```



How to Build an UI (6)

- **Step 3: Add the visible widget to the layout widget**

- All layout widgets have either of the following:

- A **child property** if they take a **single child,** for example: **Center, Padding, Container, etc**
- A **children property** if they take a **multiple child widget,** for example: **Row, Column, ListView**

- For example:

```
const Center(
  child: Text('Hello World'),
),
```

```
return Column(
  mainAxisAlignment: MainAxisAlignment.spaceEvenly,
  children: [
    Image.asset('images/pic1.jpeg'),
    Image.asset('images/pic2.jpeg'),
    Image.asset('images/pic3.jpeg'),
  ],
);  // Column
```

How to Build an UI (7)

- **Step 4: Add the layout widget to the page**
  - Instantiating and returning a widget in the app's **build()** method

Page

Layout

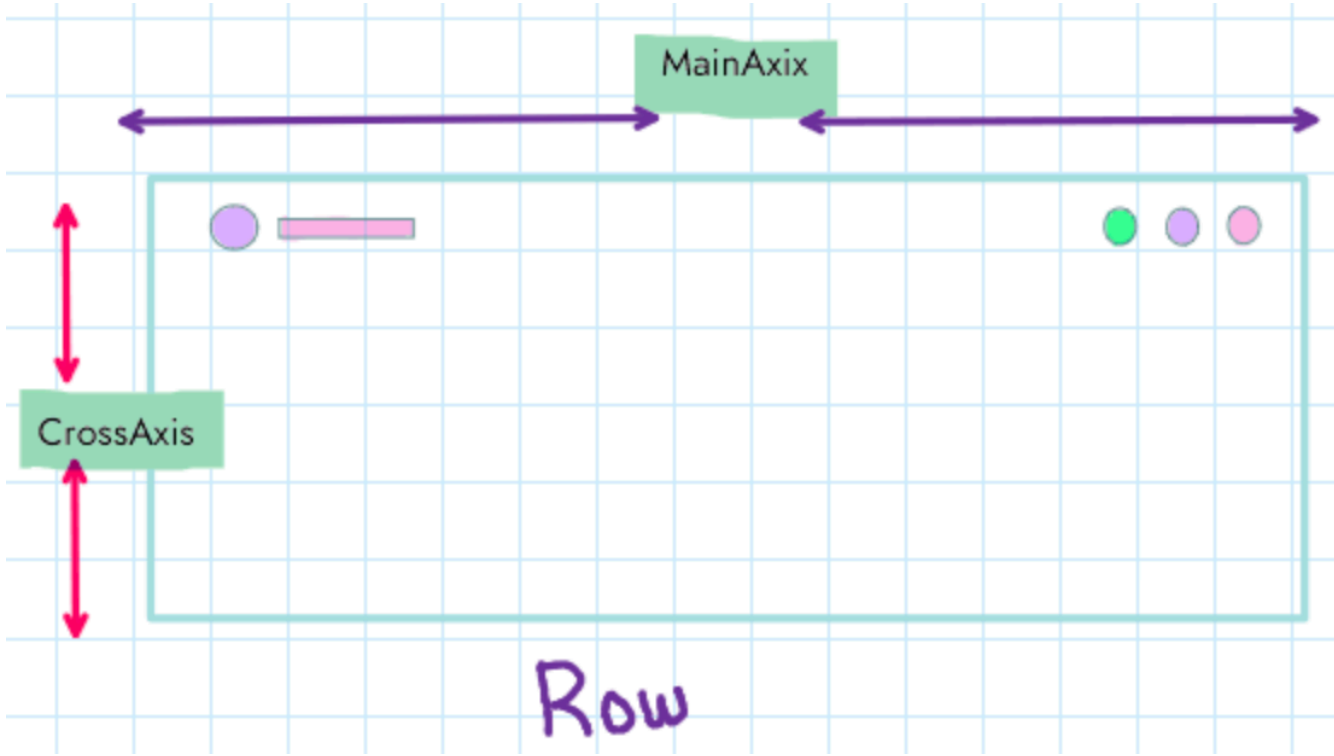
```
class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter layout demo',
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Flutter layout demo'),
        ),
        body: const Center(
          child: Text('Hello World'),
        ),
      ),
    );
  }
}
```

Row (1)

- A widget that displays its children in a **horizontal** array.
- To center (or align) **horizontally**, **mainAxisAlignment** is used.
- To center (or align) **vertically**, **crossAxisAlignment** is used.

Deliver features faster

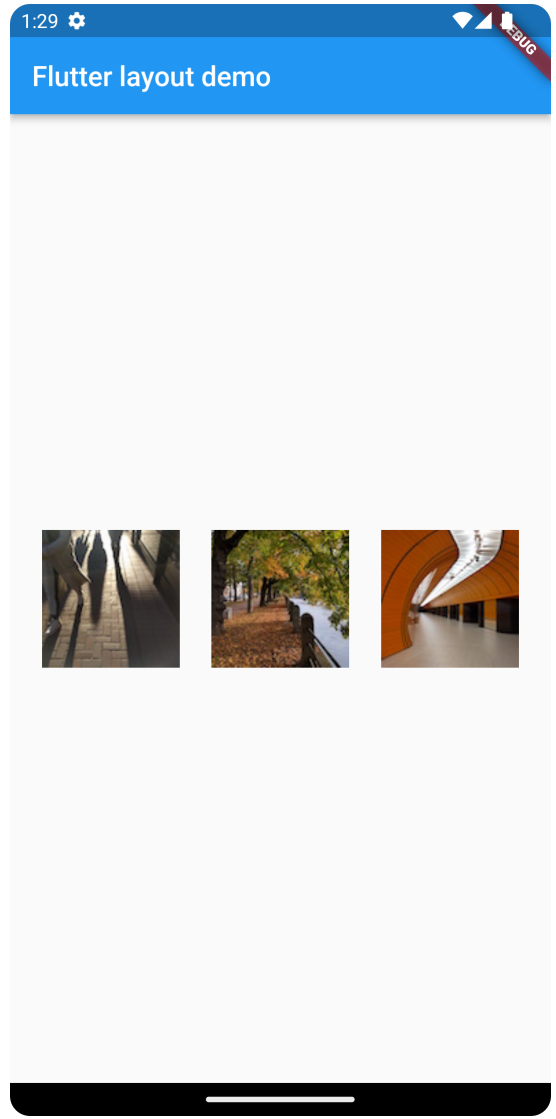
Craft beautiful UIs



Row (2)

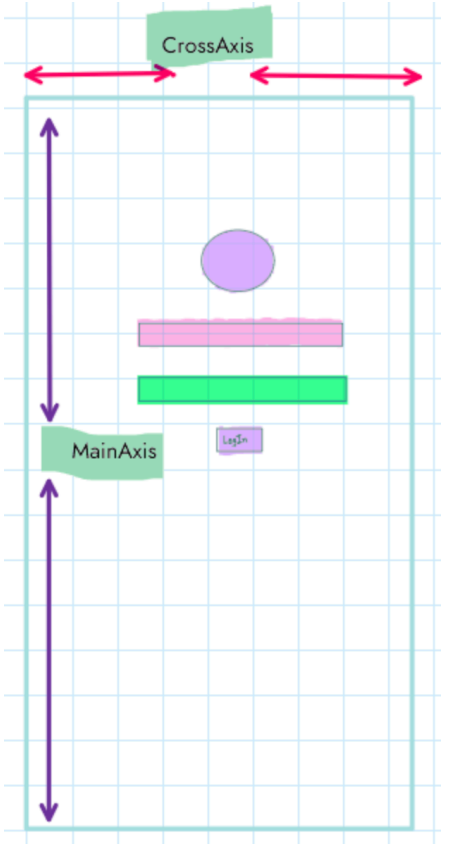
- Example

```
return Row(
  mainAxisAlignment: MainAxisAlignment.spaceEvenly,
  children: [
    Image.asset('images/pic1.jpeg'),
    Image.asset('images/pic2.jpeg'),
    Image.asset('images/pic3.jpeg'),
  ],
);  // Row
```



Column (1)

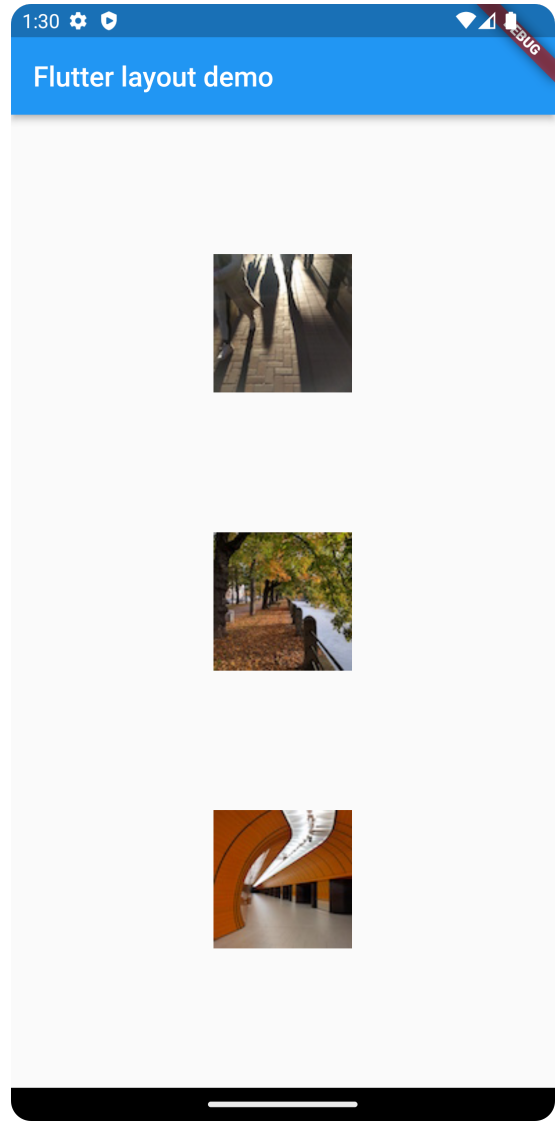
- A widget that displays its children in a **vertical** array.
- To center (or align) **vertically**, **mainAxisAlignment** is used.
- To center (or align) **horizontally**, **crossAxisAlignment** is used.



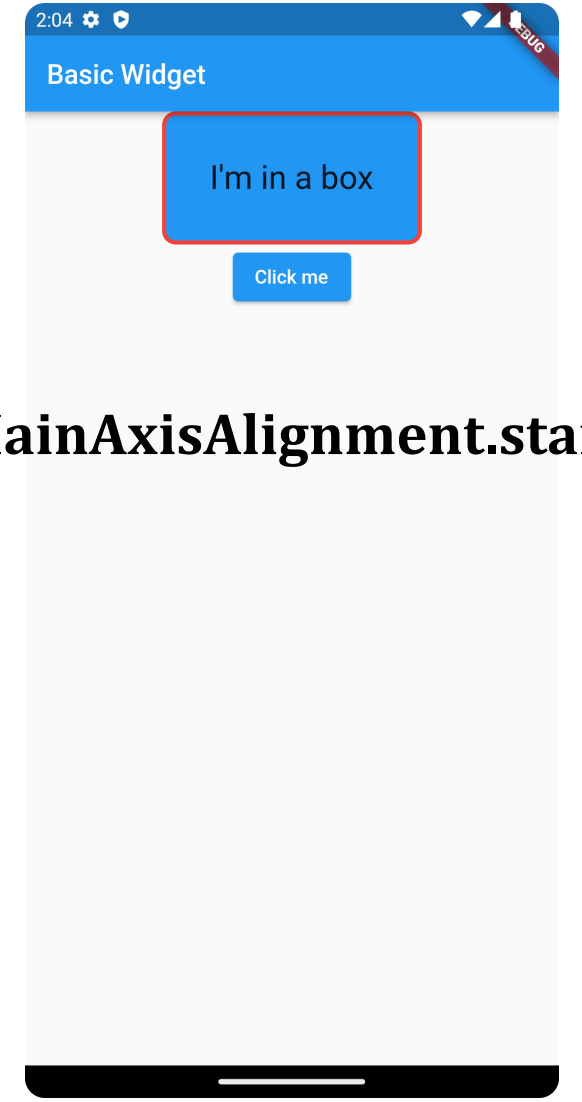
Column (2)

- Example

```
return Column(
  mainAxisAlignment: MainAxisAlignment.spaceEvenly,
  children: [
    Image.asset('images/pic1.jpeg'),
    Image.asset('images/pic2.jpeg'),
    Image.asset('images/pic3.jpeg'),
  ],
);  // Column
```



Column (3)

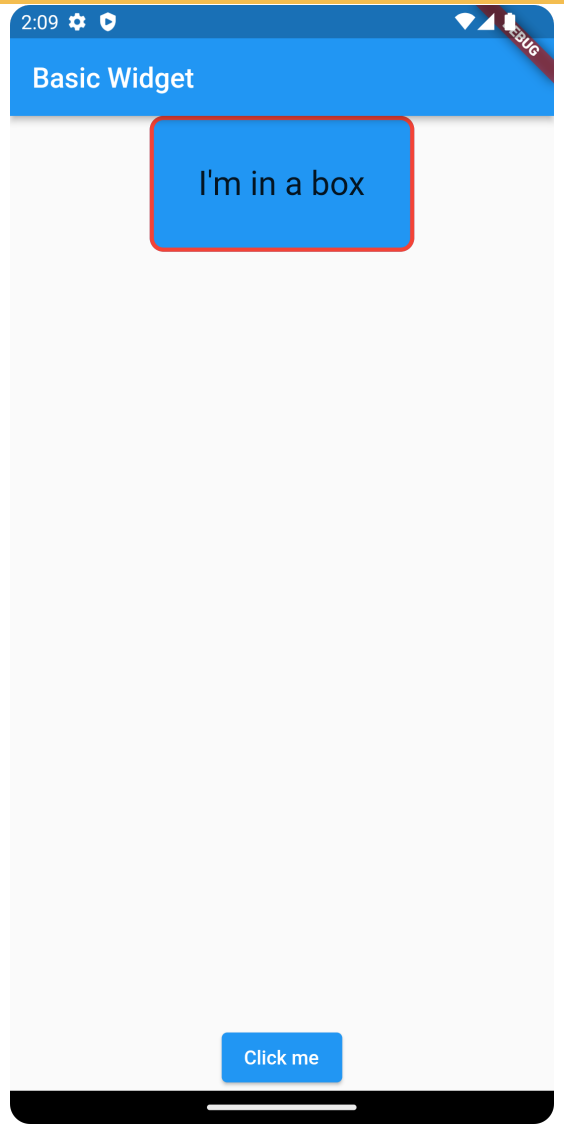


MainAxisAlignment.start

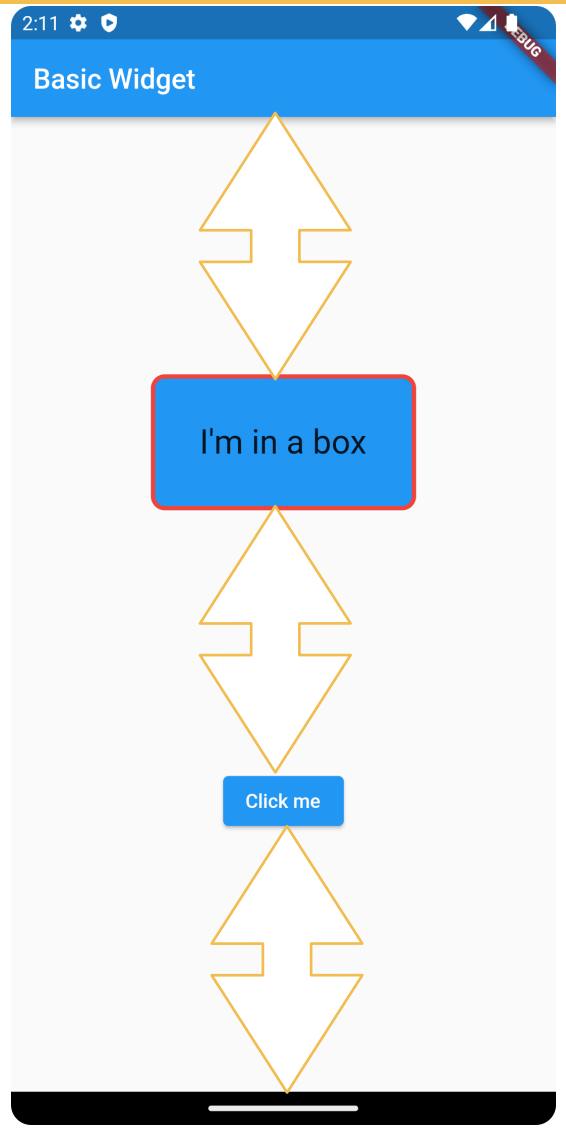
MainAxisAlignment.center

MainAxisAlignment.end

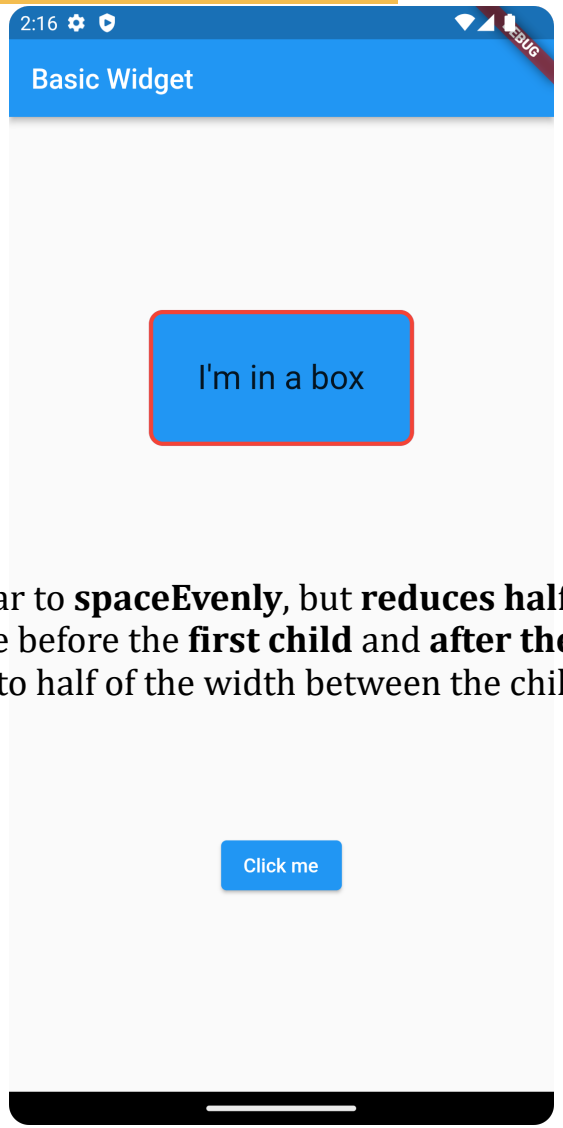
Column (4)



MainAxisAlignment.spaceBetween



MainAxisAlignment.spaceEvenly

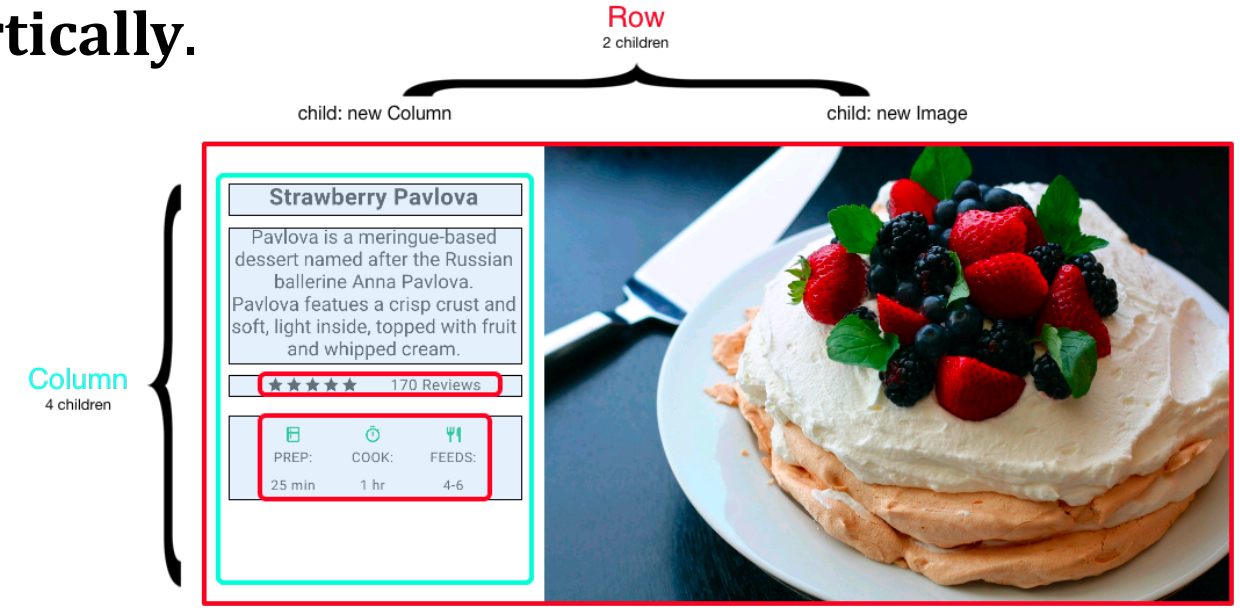


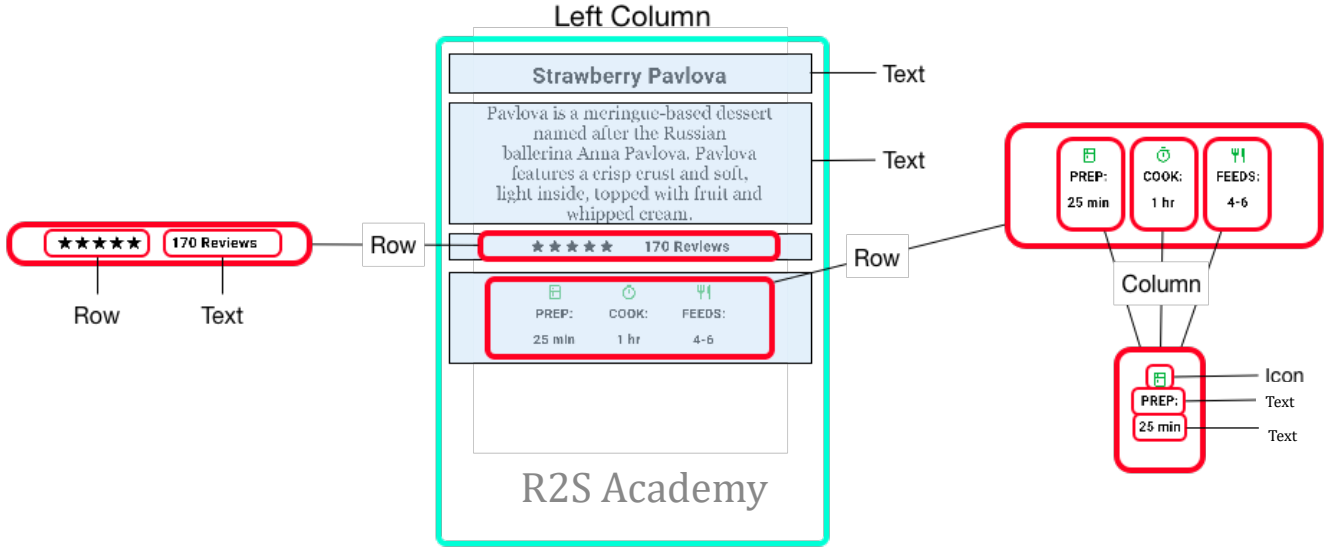
(Similar to **spaceEvenly**, but **reduces half** of the space before the **first child** and **after the last child** to half of the width between the children.)

MainAxisAlignment.spaceAround

Nesting Rows and Columns (1)

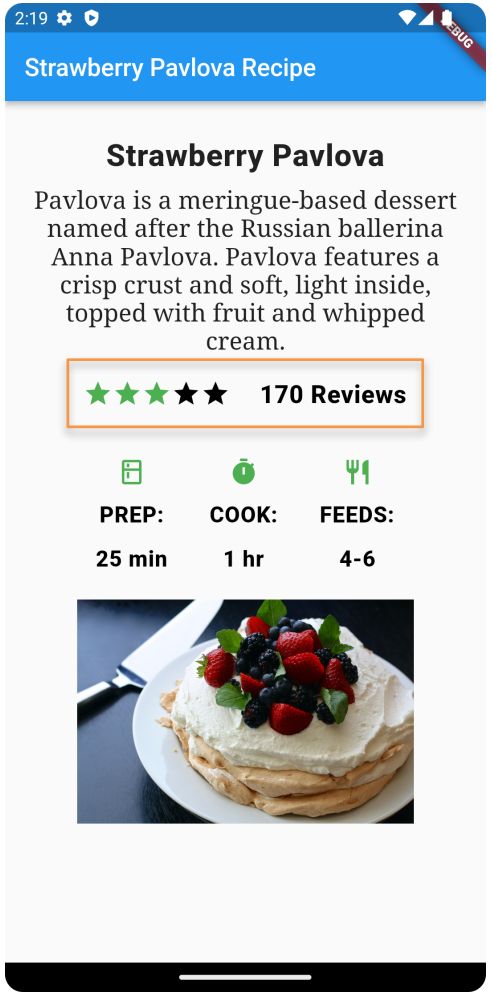
- You can use a **Row** widget to arrange widgets **horizontally**, and a **Column** widget to arrange widgets **vertically.**
- Example

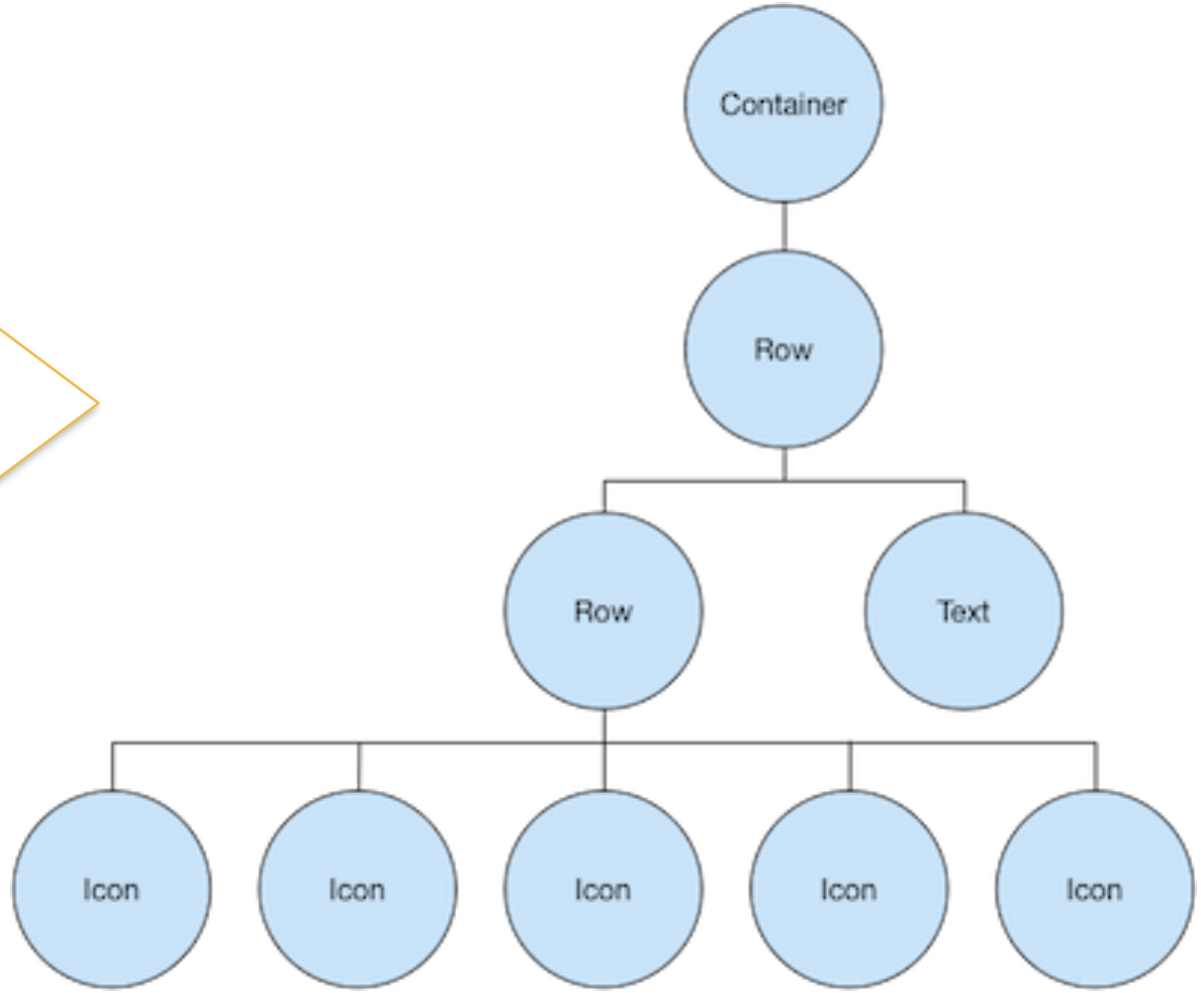




Nesting Rows and Columns (2)

- The outlined section is implemented as two rows. The ratings row contains five stars and the number of reviews. The icons row contains three columns of icons and text.

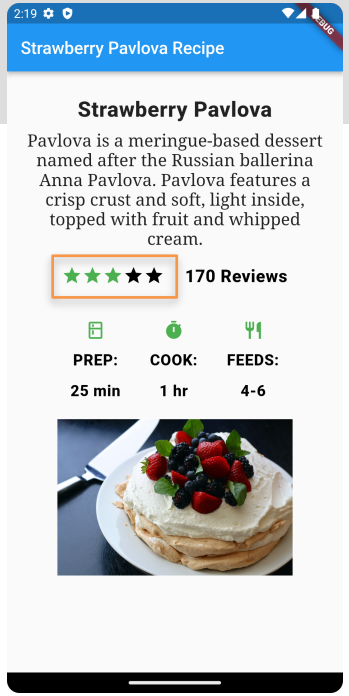




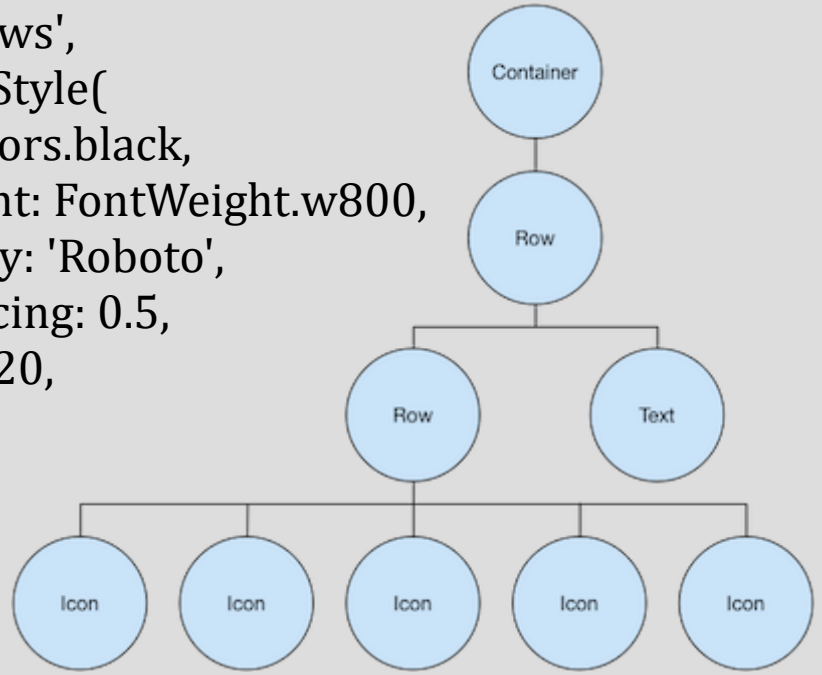
Nesting Rows and Columns (3)

- Creates a row containing a smaller row of 5 star icons, and text:

```
var stars = Row(
  mainAxisSize: MainAxisSize.min,
  children: [
    Icon(Icons.star, color: Colors.green[500]),
    Icon(Icons.star, color: Colors.green[500]),
    Icon(Icons.star, color: Colors.green[500]),
    Icon(Icons.star, color: Colors.black),
    Icon(Icons.star, color: Colors.black),
  ],
);
```

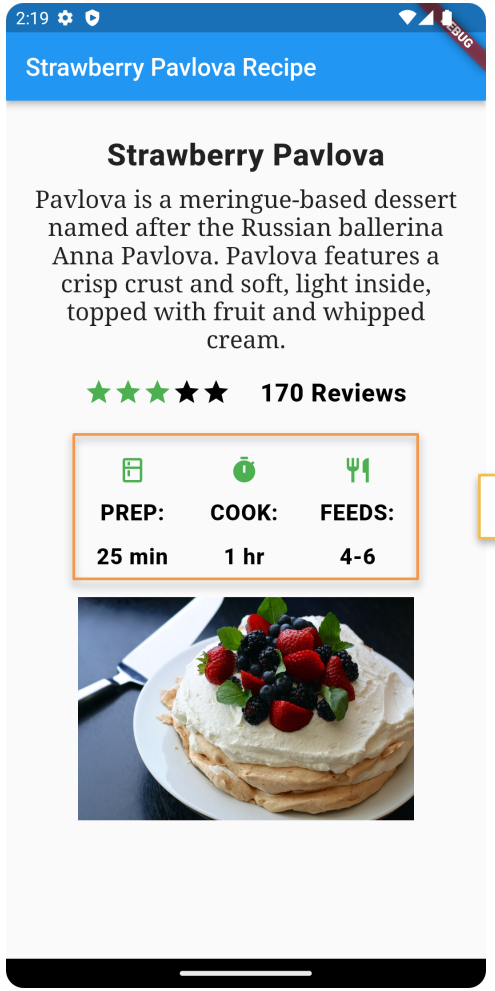


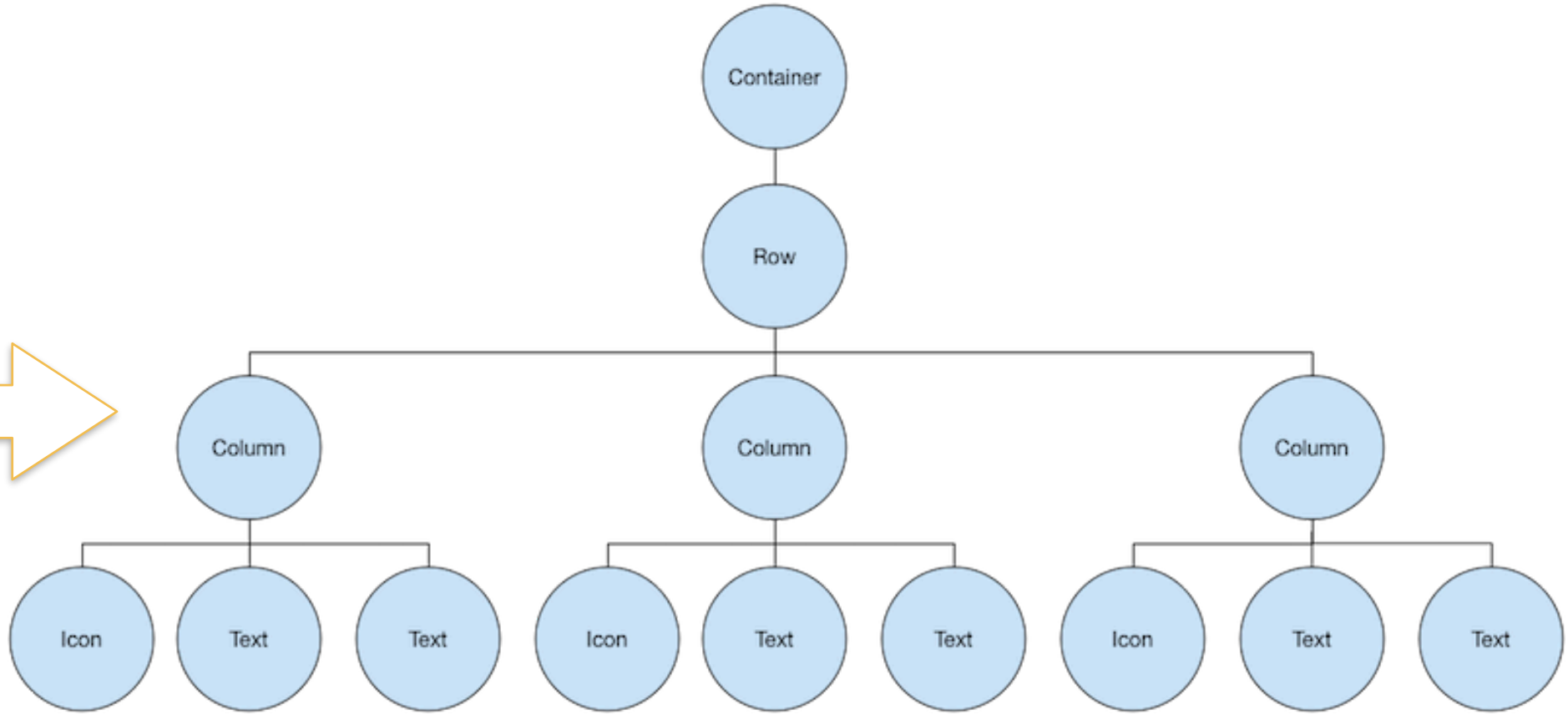

```
final ratings = Container(
  padding: const EdgeInsets.all(20),
  child: Row(
    mainAxisAlignment: MainAxisAlignment.spaceEvenly,
    children: [
      stars,
      const Text(
        '170 Reviews',
        style: TextStyle(
          color: Colors.black,
          fontWeight: FontWeight.w800,
          fontFamily: 'Roboto',
          letterSpacing: 0.5,
          fontSize: 20,
        ),
      ),
    ],
  ),
);
```



Nesting Rows and Columns (4)

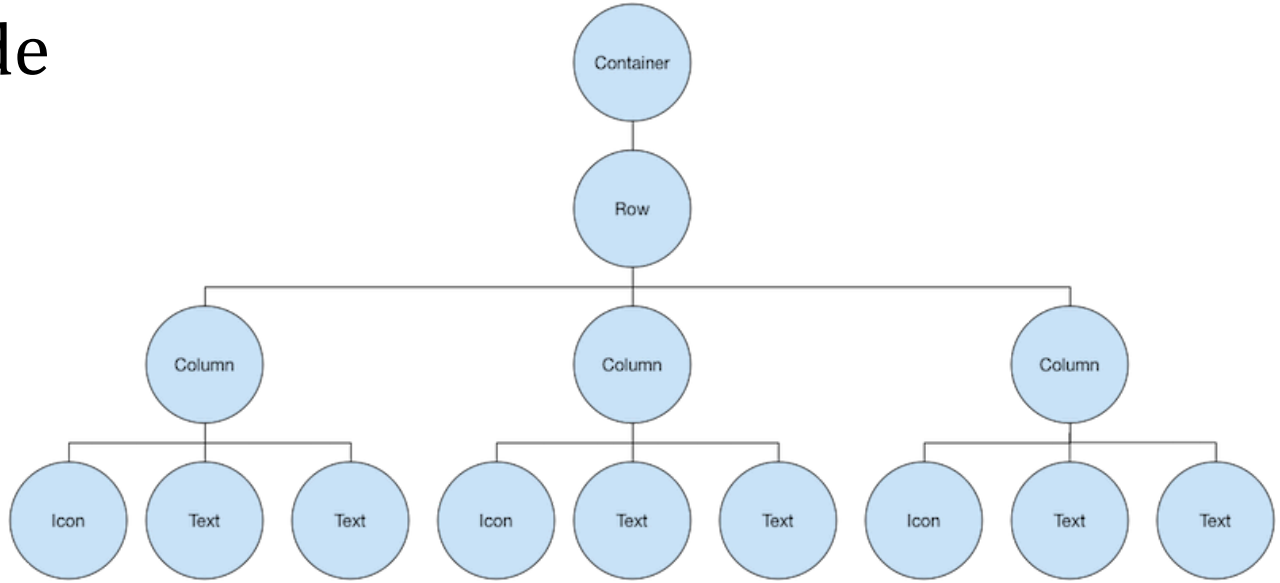
- The icons row, below the ratings row, contains 3 columns; each column contains an icon and two lines of text, as you can see in its widget tree:

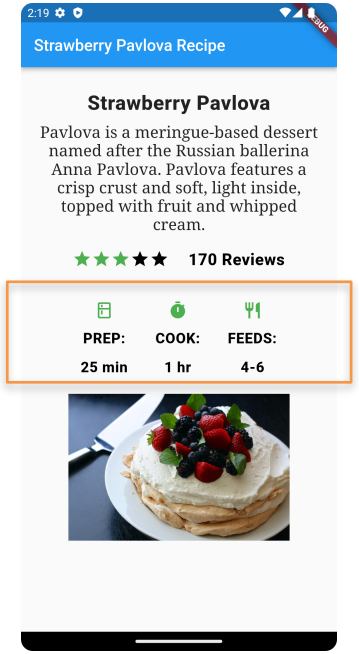




Nesting Rows and Columns (5)

- Code



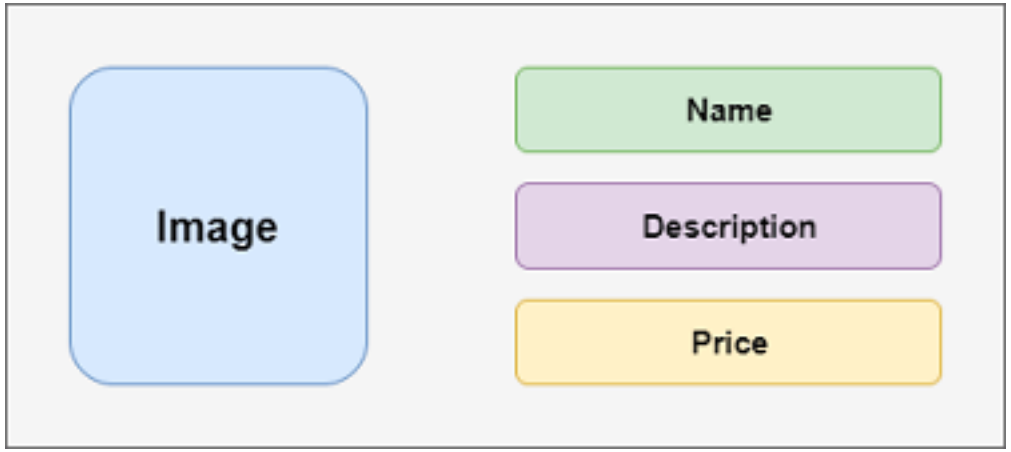



```
child: Row(
  mainAxisAlignment: MainAxisAlignment.spaceEvenly,
  children: [
    Column(
      children: [
        Icon(Icons.kitchen, color: Colors.green[500]),
        const Text('PREP:'),
        const Text('25 min'),
      ],
    ),  // Column
    Column(
      children: [
        Icon(Icons.timer, color: Colors.green[500]),
        const Text('COOK:'),
        const Text('1 hr'),
      ],
    ),  // Column
    Column(
      children: [
        Icon(Icons.restaurant, color: Colors.green[500]),
        const Text('FEEDS:'),
        const Text('4-6'),
      ],
    ),  // Column
  ],
),  // Row
```

Nesting Rows and Columns (6)

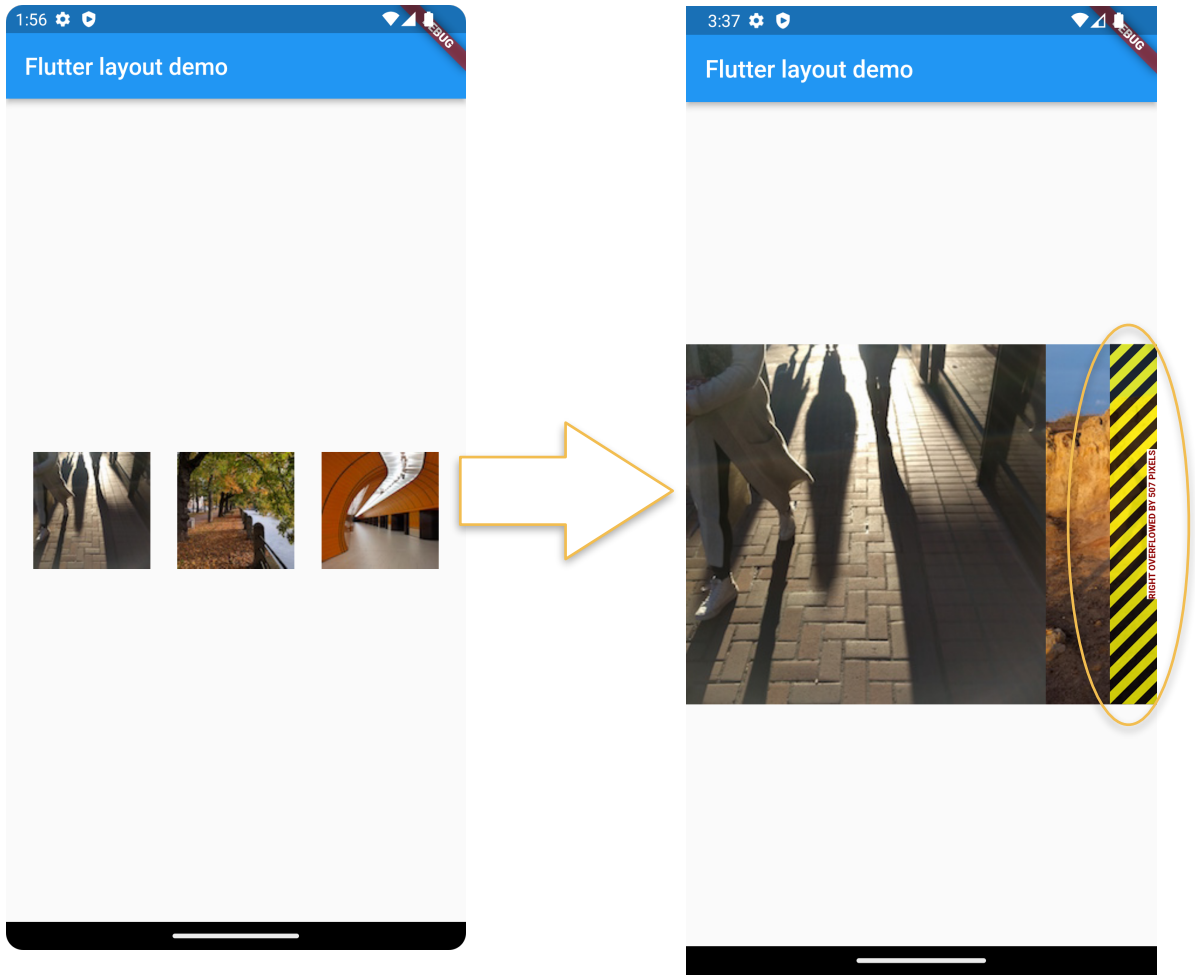
- Demo: Building complex layout

Widget tree:
- Column
  - Row
    - Image
    - Column
      - Text
      - Text
      - Text



Sizing Widgets (1)

- When a layout is **too large** to fit a device, a yellow and black striped pattern appears along the affected edge.

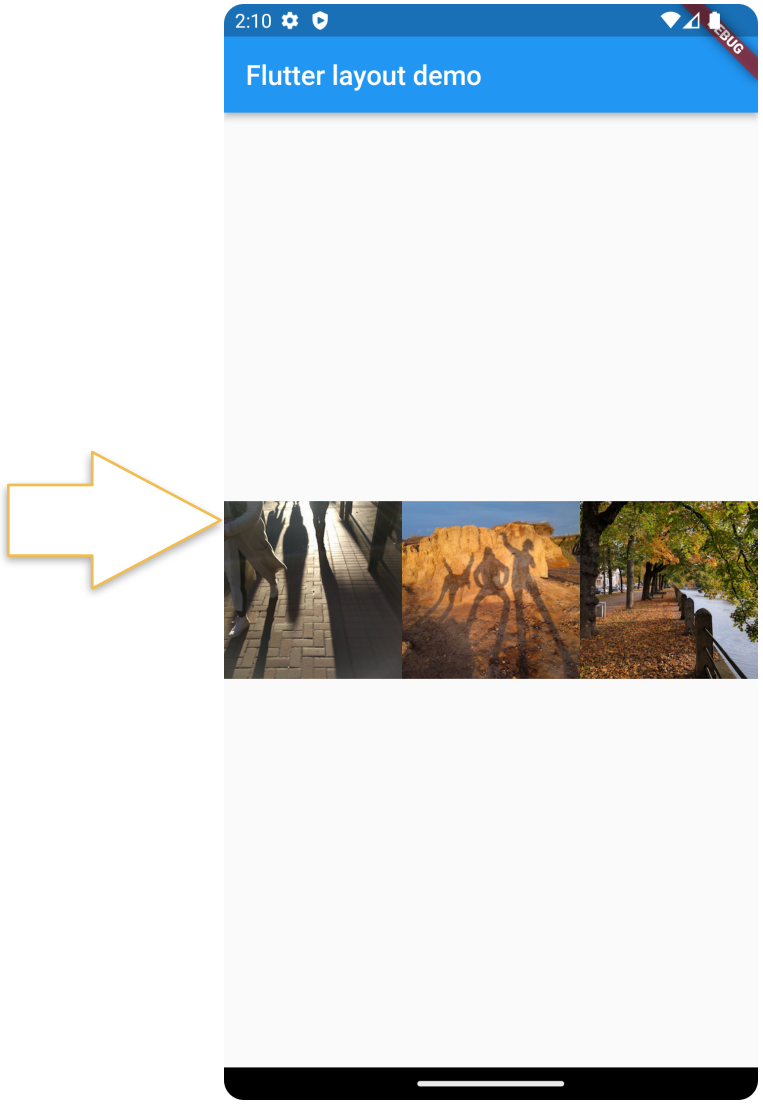


Sizing Widgets (2)

- Widgets can be sized to fit within a row or column by using the **Expanded** widget.

Fixed

```
Row(
  crossAxisAlignment: CrossAxisAlignment.center,
  children: [
    Expanded(child: Image.asset('images/pic1.jpg'),),
    Expanded(child: Image.asset('images/pic2.jpg'),),
    Expanded(child: Image.asset('images/pic3.jpg'),),
  ],
);
```



- Others: SizeBox widget

Summary (1)

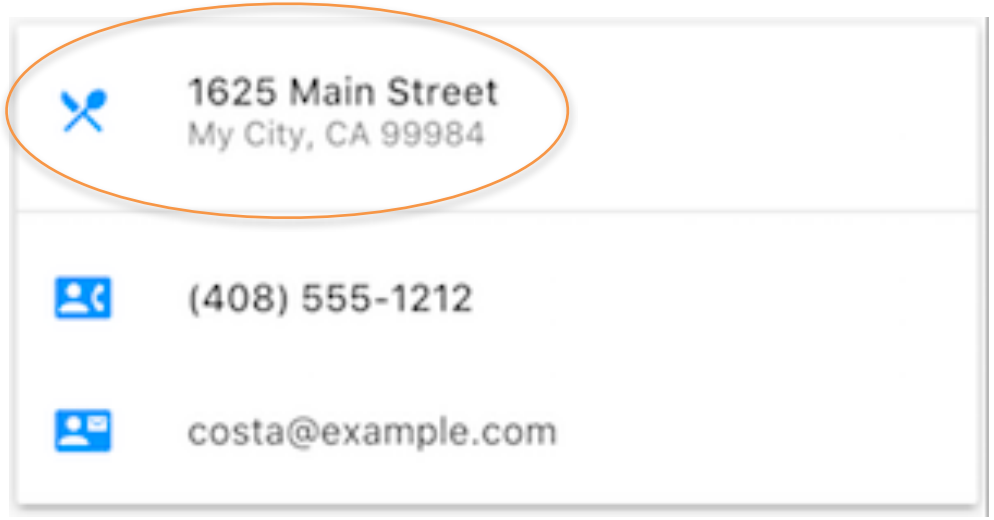
- **Row** is **basic** primitive widgets for **horizontal** (Allow for **maximum customization**).

Summary (2)

- **Column** is **basic** primitive widgets for **vertical** layouts (allow for **maximum customization**).

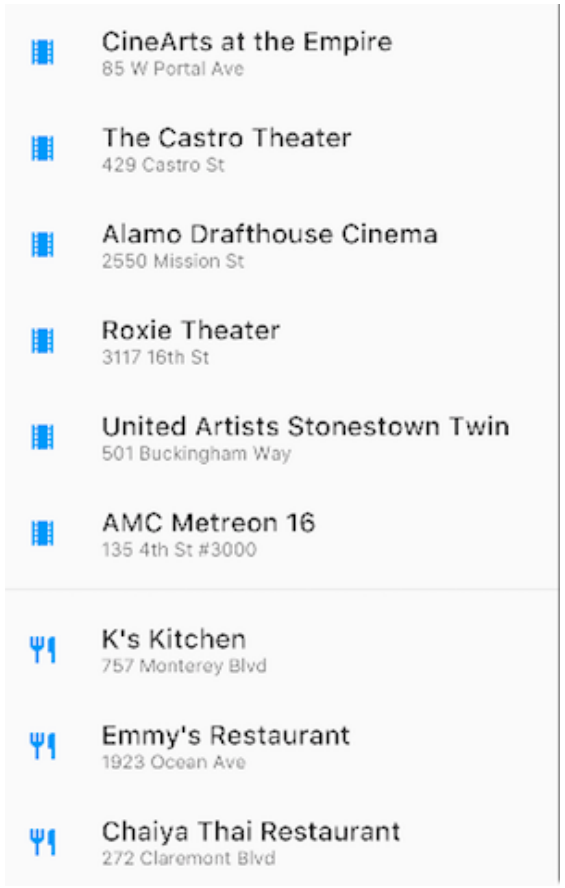
Summary (3)

- Instead of **Row** you might prefer **ListTile**, an easy-to-use widget with properties for **leading, title, subtitle** and **trailing**.
- ListTile (Instead of **Row)**



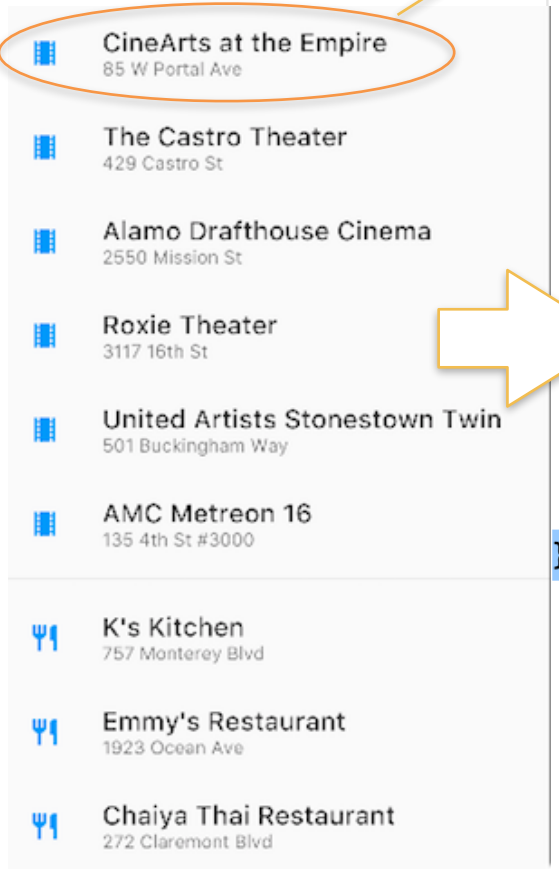
Summary (4)

- Instead of **Column**, you might prefer **ListView**, a column-like layout that **automatically scrolls** if its content is too long to fit the available space.
- ListView (Instead of **Column)**



Summary (5)

- Example



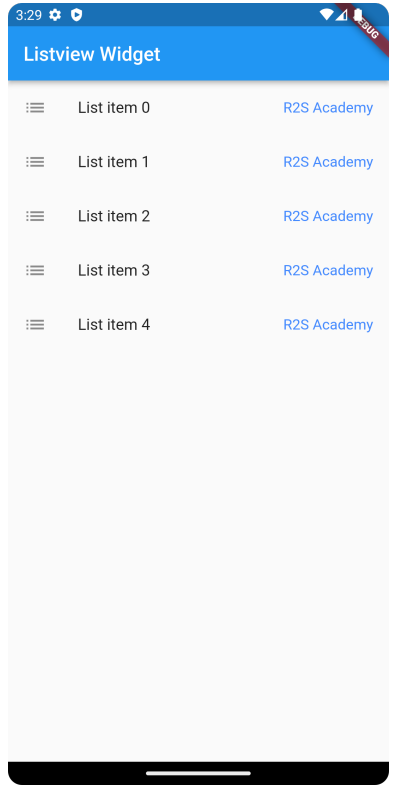

```
ListTile _tile(String title, String subtitle, IconData icon) {
  return ListTile(
    title: Text(title,
        style: const TextStyle(
          fontWeight: FontWeight.w500,
          fontSize: 20,
        )),  // TextStyle, Text
    subtitle: Text(subtitle),
    leading: Icon(
      icon,
      color: Colors.blue[500],
    ),  // Icon
  );  // ListTile
}
```

```
Widget _buildList() {
  return ListView(
    children: [
      _tile('CineArts at the Empire', '85 W Portal Ave', Icons.theaters),
      _tile('The Castro Theater', '429 Castro St', Icons.theaters),
      _tile('Alamo Drafthouse Cinema', '2550 Mission St', Icons.theaters),
      _tile('Roxie Theater', '3117 16th St', Icons.theaters),
      _tile('United Artists Stonestown Twin', '501 Buckingham Way',
          Icons.theaters),
      _tile('AMC Metreon 16', '135 4th St #3000', Icons.theaters),
      const Divider(),
      _tile('K\'s Kitchen', '757 Monterey Blvd', Icons.restaurant),
      _tile('Emmy\'s Restaurant', '1923 Ocean Ave', Icons.restaurant),
      _tile(
          'Chaiya Thai Restaurant', '272 Claremont Blvd', Icons.restaurant),
      _tile('La Ciccia', '291 30th St', Icons.restaurant),
    ],
  );  // ListView
}
```

Summary (6)

- When we want to create a list **recursively without writing code again and again** then **ListView.builder** is used.

```
Widget _buildListViewBuilder() {
  return ListView.builder(
      itemCount: 5,
      itemBuilder: (BuildContext context, int index) {
        return ListTile(
            leading: const Icon(Icons.list),
            trailing: const Text(
              "R2S Academy",
              style: TextStyle(color: Colors.blueAccent, fontSize: 15),
            ),  // Text
            title: Text("List item $index"));  // ListTile
      }
      );  // ListView.builder
}
```



- We have *ListView.builder* with itemCount and *itemBuilder* which will create a new widget **again and again up to 5 times** because we have *itemCount = 5*
- The *itemBuilder* returns ***ListTile*** which has **leading**, **title** and **trailing**. This *ListTile* will build again and again up to 5 times.

*Keeping up those **inspiration** and the **enthusiasm** in the **learning path**. Let confidence to bring it into **your career path** for getting gain the **success** as your expectation.*

Thank you

Contact

- Name: R2S Academy
- Email: daotao@r2s.edu.vn
- Phone/Zalo: 0919 365 363
- FB: https://www.facebook.com/r2s.tuyendung
- Website: www.r2s.edu.vn

Questions and Answers