

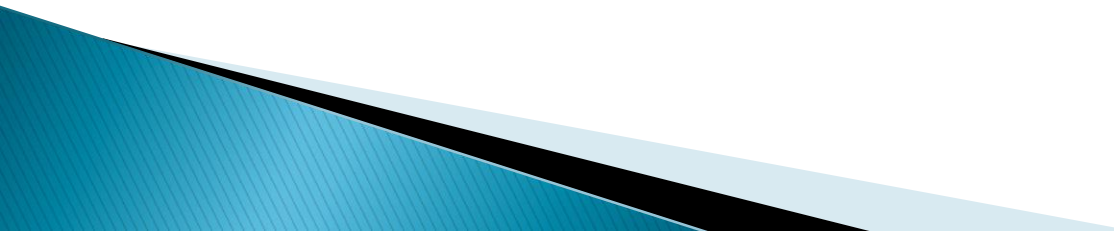
Chapter 5

Scrollable Widgets

Widget Theory. We'll explore:

- ❖ How to use ListView
- ❖ How to nest scroll views
- ❖ How to leverage the power of GridView

What is a ListView?

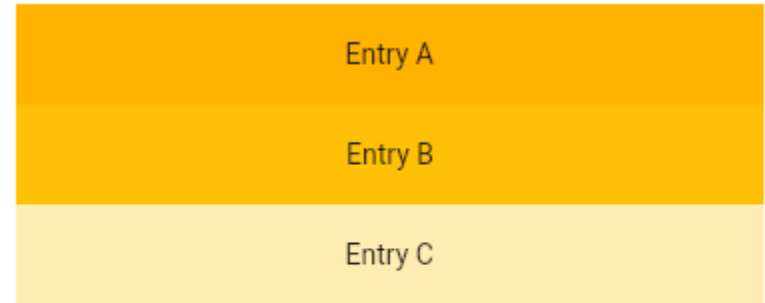
- Is the most commonly used scrolling widget.
 - It displays its children one after another in the scroll direction, i.e. vertical or horizontal.
 - There are different types of ListViews:
 - `ListView`
 - `ListView.builder`
 - `ListView.separated`
 - `ListView.custom`
- 

4 options for constructing a ListView

- The default constructor takes an explicit **List<Widget>** of children.
- This constructor is appropriate for list views with a **small** number of **children** because constructing the **List** requires doing work for every child that could possibly be displayed in the list view **instead** of just those children that are actually visible.

4 options for constructing a ListView

```
ListView(  
  padding: const EdgeInsets.all(8),  
  children: <Widget>[  
    Container(  
      height: 50,  
      color: Colors.amber[600],  
      child: const Center(child: Text('Entry A')),  
    ),  
    Container(  
      height: 50,  
      color: Colors.amber[500],  
      child: const Center(child: Text('Entry B')),  
    ),  
    Container(  
      height: 50,  
      color: Colors.amber[100],  
      child: const Center(child: Text('Entry C')),  
    ),  
  ],  
)
```



4 options for constructing a ListView

- The **ListView.builder** constructor takes an **IndexedWidgetBuilder**, which builds the children on demand.
- This constructor is appropriate for list views with a **large** (or **infinite**) number of children because the builder is called **only** for those children that are **actually visible**.

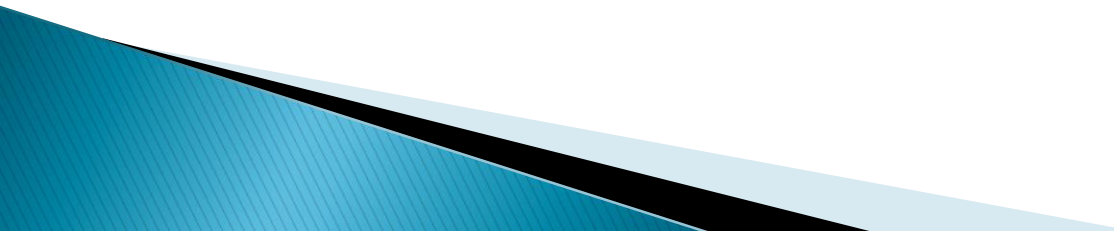
4 options for constructing a ListView

```
final List<String> entries = <String>['A', 'B', 'C'];  
final List<int> colorCodes = <int>[600, 500, 100];
```

```
ListView.builder(  
  padding: const EdgeInsets.all(8),  
  itemCount: entries.length,  
  itemBuilder: (BuildContext context, int index) {  
    return Container(  
      height: 50,  
      color: Colors.amber[colorCodes[index]],  
      child: Center(child: Text('Entry ${entries[index]}'  
    ));  
  }  
);
```

Entry A
Entry B
Entry C

4 options for constructing a ListView

- The **ListView.separated** constructor takes two **IndexedWidgetBuilders**: **itemBuilder** builds child items on demand, and **separatorBuilder** similarly builds separator children which appear in between the child items.
 - This constructor is appropriate for list views with a fixed number of children.
- 

4 options for constructing a ListView

```
final List<String> entries = <String>['A', 'B', 'C'];  
final List<int> colorCodes = <int>[600, 500, 100];
```

```
ListView.separated(  
  padding: const EdgeInsets.all(8),  
  itemCount: entries.length,  
  itemBuilder: (BuildContext context, int index) {  
    return Container(  
      height: 50,  
      color: Colors.amber[colorCodes[index]],  
      child: Center(child: Text('Entry ${entries[index]}')),  
    );  
  },  
  separatorBuilder: (BuildContext context, int index) =>  
    const Divider(),  
);
```

Entry A

Entry B

Entry C

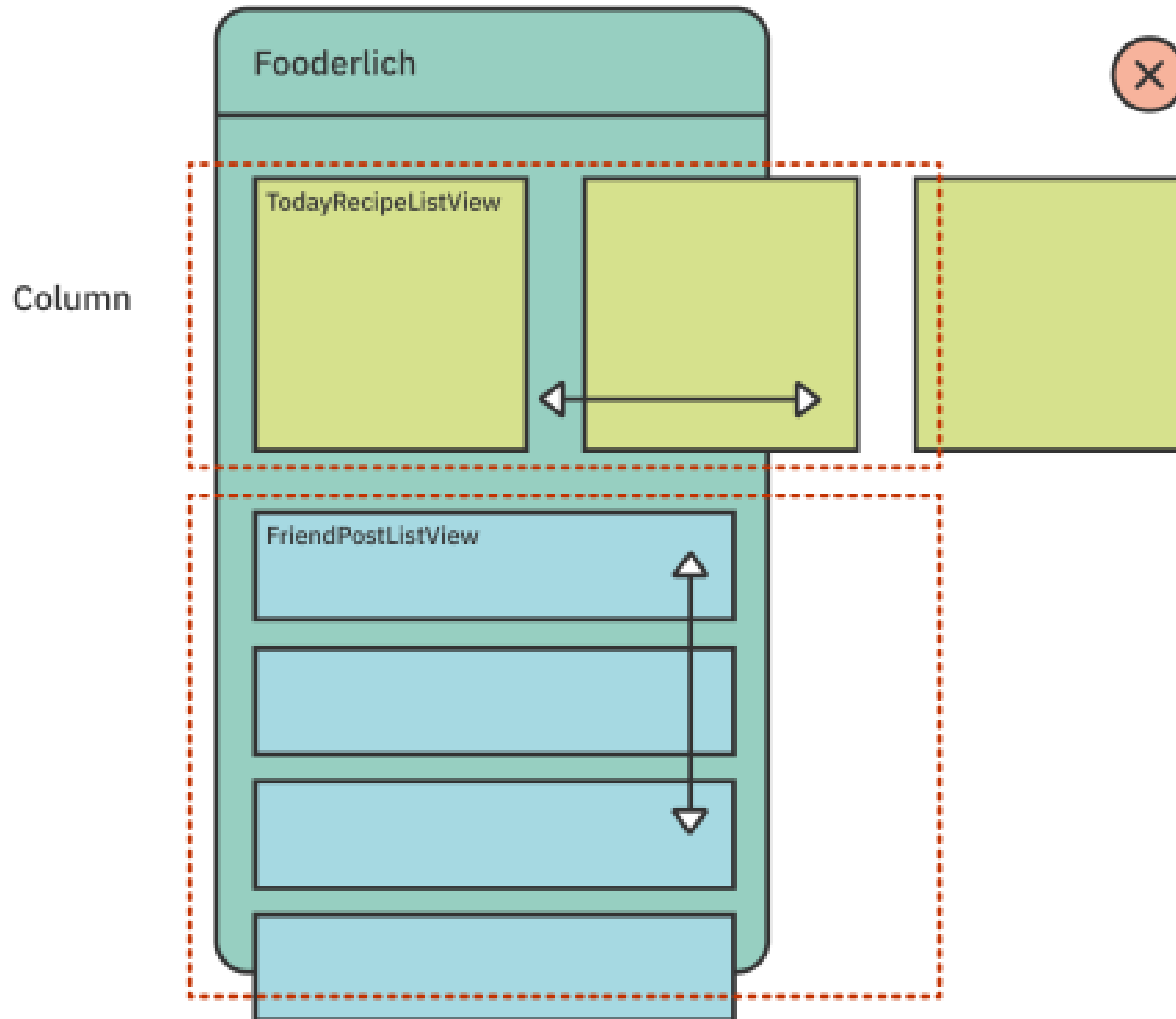
4 options for constructing a ListView

- The **ListView.custom** constructor takes a **SliverChildDelegate**, which provides the ability to customize additional aspects of the child model.
- For example, a **SliverChildDelegate** can control the algorithm used to estimate the size of children that are not actually visible.

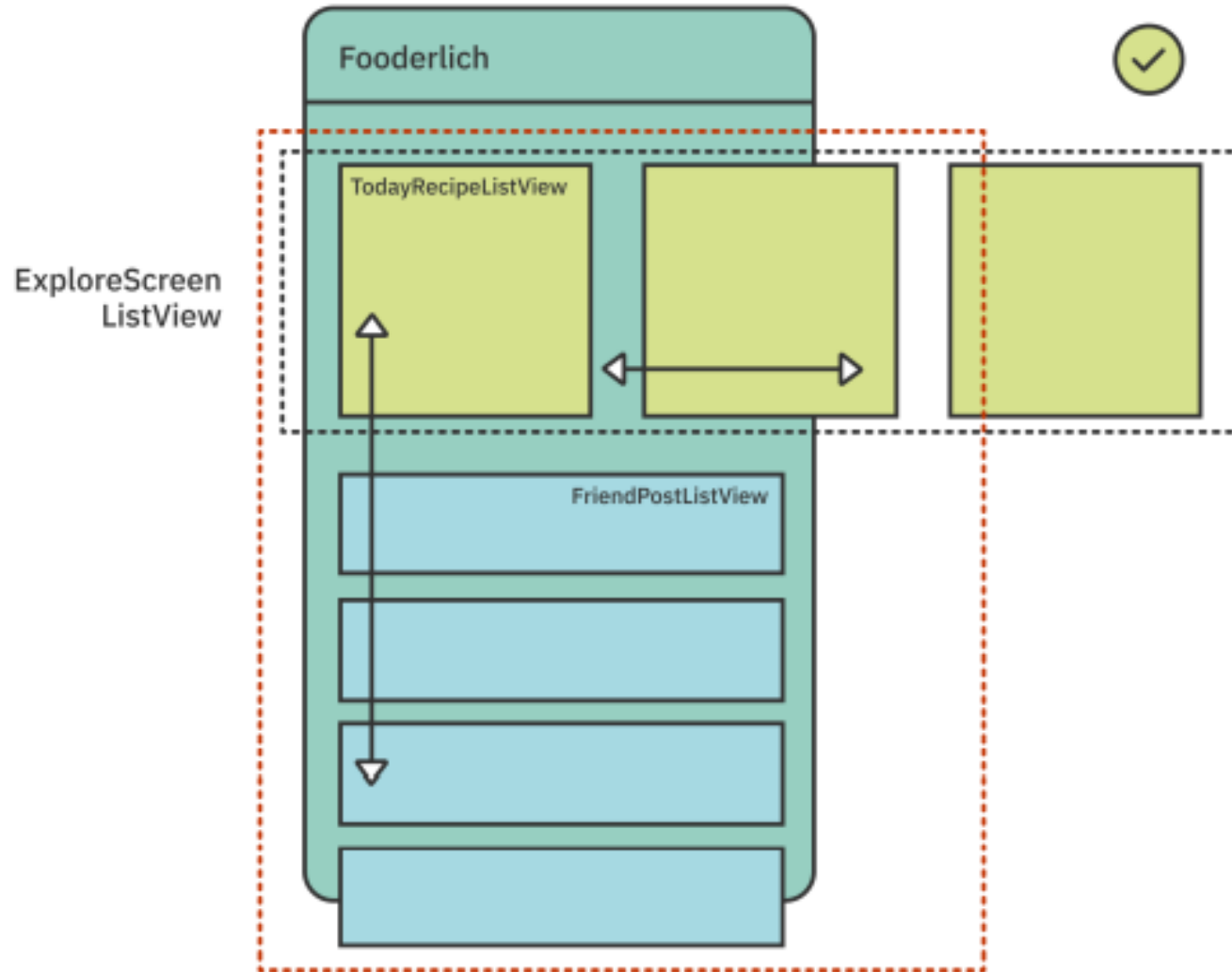
4 options for constructing a ListView

```
CustomScrollView(  
  shrinkWrap: true,  
  slivers: <Widget>[  
    SliverPadding(  
      padding: const EdgeInsets.all(20.0),  
      sliver: SliverList(  
        delegate: SliverChildListDelegate(  
          <Widget>[  
            const Text("I'm dedicating every day to you"),  
            const Text('Domestic life was never quite my style'),  
            const Text('When you smile, you knock me out, I fall apart'),  
            const Text('And I thought I was so smart'),  
          ],  
        ),  
      ),  
    ],  
  ),  
)
```

NestedScrollView class



NestedScrollView class



NestedScrollView class

- A scrolling view inside of which can be nested other scrolling views, with their scroll positions being intrinsically linked.
- The most common use case for this widget is a scrollable view with a flexible **SliverAppBar** containing a **TabBar** in the header.

GridView class

He'd have you all
unravel at the

Heed not the
rabble

Sound of
screams but the

Who scream

Revolution is
coming...


Revolution,
they...

GridView class

- A **GridView** is basically a **CustomScrollView** with a single **SliverGrid** in its **CustomScrollView.slivers** property.
- If **GridView** is no longer sufficient, for example **because** the scroll view is to have both a grid and a list, or **because** the grid is to be combined with a **SliverAppBar**.

GridView class

```
GridView.count(  
  primary: false,  
  padding: const EdgeInsets.all(20),  
  crossAxisSpacing: 10,  
  mainAxisSpacing: 10,  
  crossAxisCount: 2,  
  children: <Widget>[  
    Container(  
      padding: const EdgeInsets.all(8),  
      child: const Text("He'd have you all unravel at the"),  
      color: Colors.teal[100],  
    ),  
    Container(  
      padding: const EdgeInsets.all(8),  
      child: const Text('Heed not the rabble'),  
      color: Colors.teal[200],  
    ),  
  ],  
)
```



GridView class

```
Container(  
    padding: const EdgeInsets.all(8),  
    child: const Text('Sound of screams but the'),  
    color: Colors.teal[300],  
),  
Container(  
    padding: const EdgeInsets.all(8),  
    child: const Text('Who scream'),  
    color: Colors.teal[400],  
),  
Container(  
    padding: const EdgeInsets.all(8),  
    child: const Text('Revolution is coming...'),  
    color: Colors.teal[500],  
),  
Container(  
    padding: const EdgeInsets.all(8),  
    child: const Text('Revolution, they...'),  
    color: Colors.teal[600],  
),  
],  
)
```