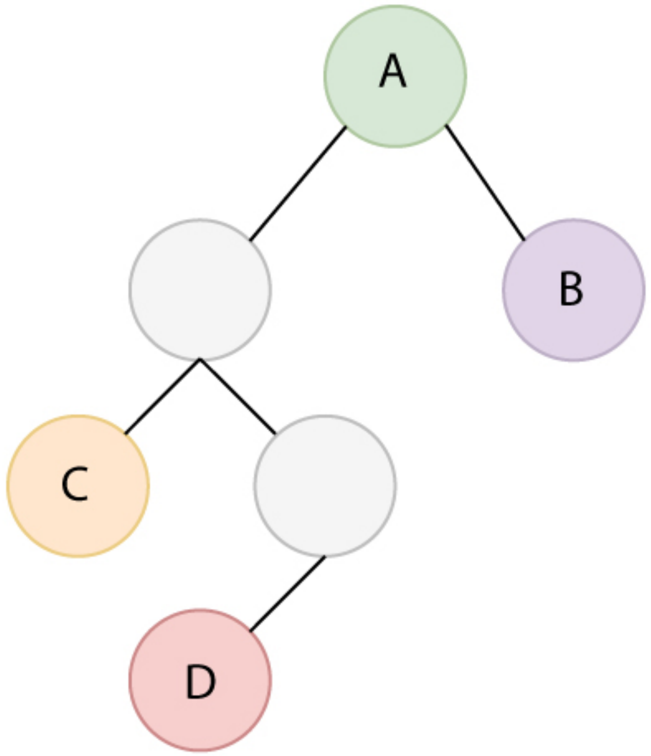
Resource Software Solution

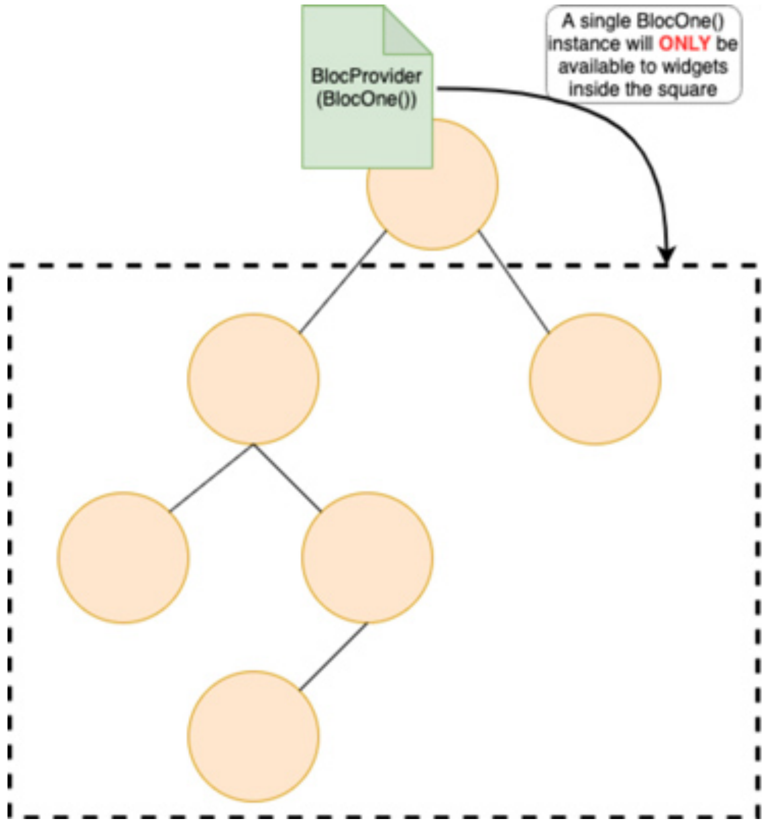
Lightweight Bloc with Cubit

Introduction (1)

- BloC makes it easy to implement the Business Logic Component design pattern, which separates presentation from business logic.



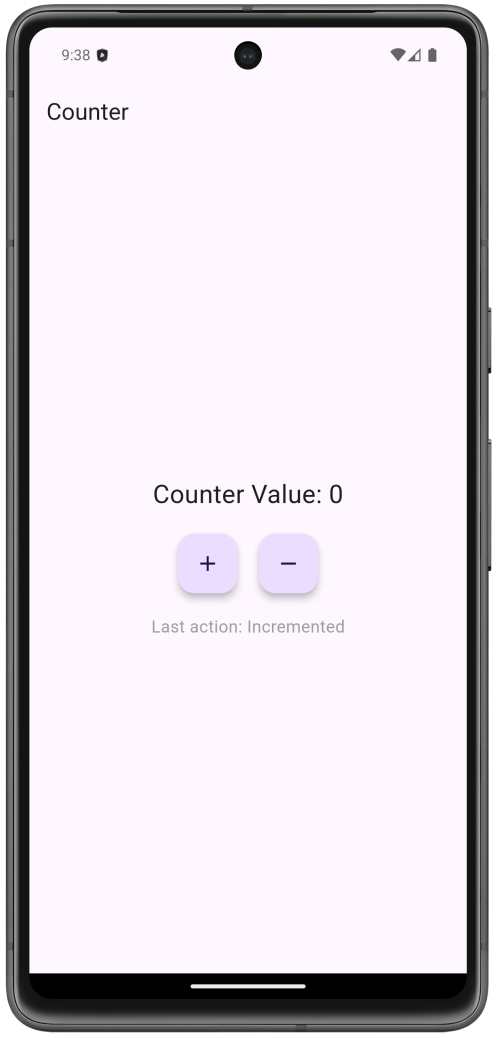
Without Bloc

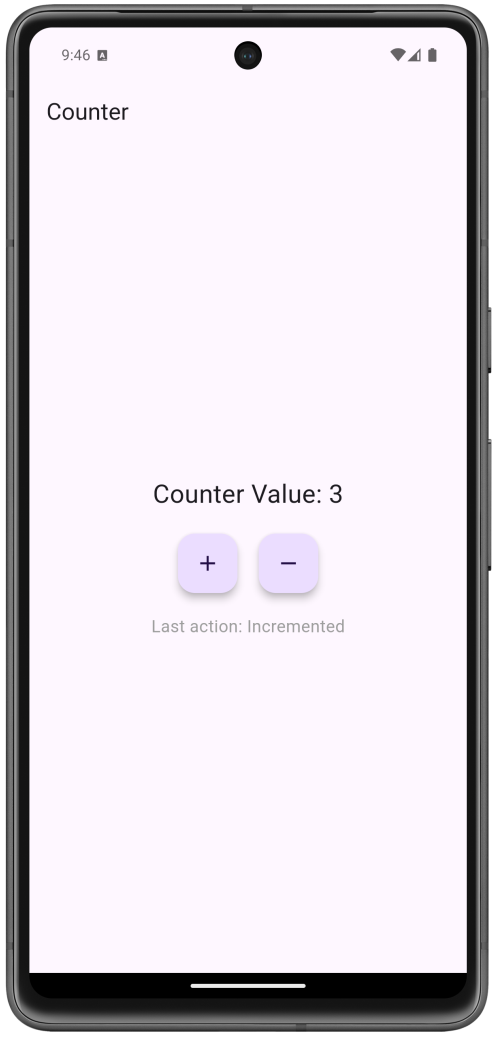

BlocProvider (BlocOne())
A single BlocOne() instance will ONLY be available to widgets inside the square

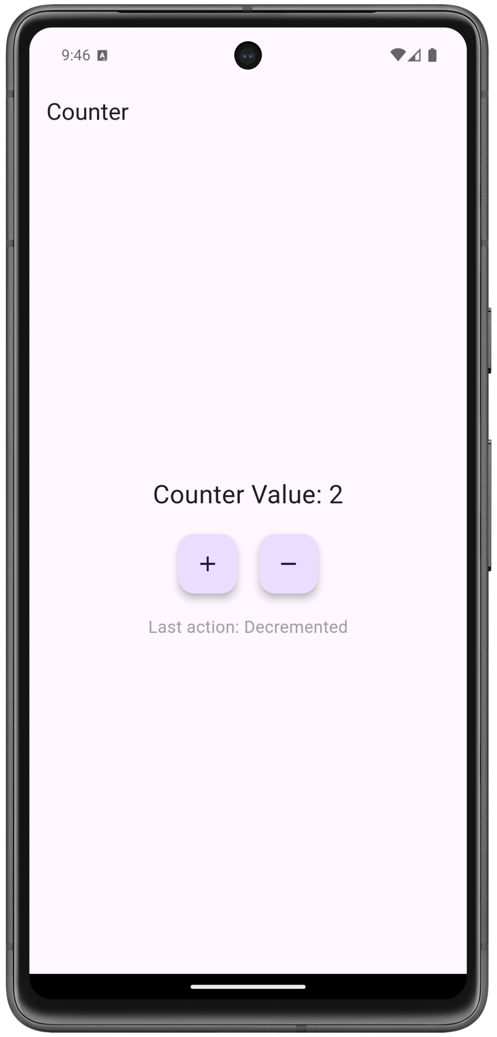
With Bloc

Introduction (2)

- Example: Flutter Counter App







Introduction (3)

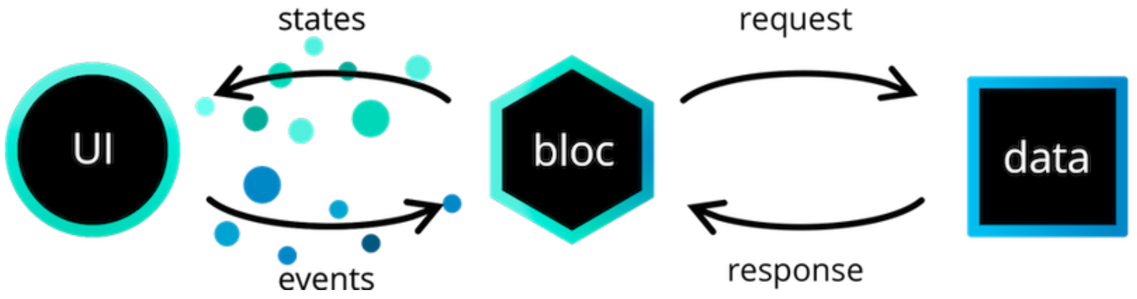
- Code: Flutter Counter App

```
abstract class CounterEvent {}

class Increment extends CounterEvent {}
class Decrement extends CounterEvent {}

class _CounterBloc extends Bloc<CounterEvent, int> {

  _CounterBloc() : super(0) {
    on<CounterEvent>(((event, emit) {
      if (event is Increment) {
        emit(state + 1);
      } else if (event is Decrement) {
        emit(state - 1);
      }
    }));
  }
}
```



VS

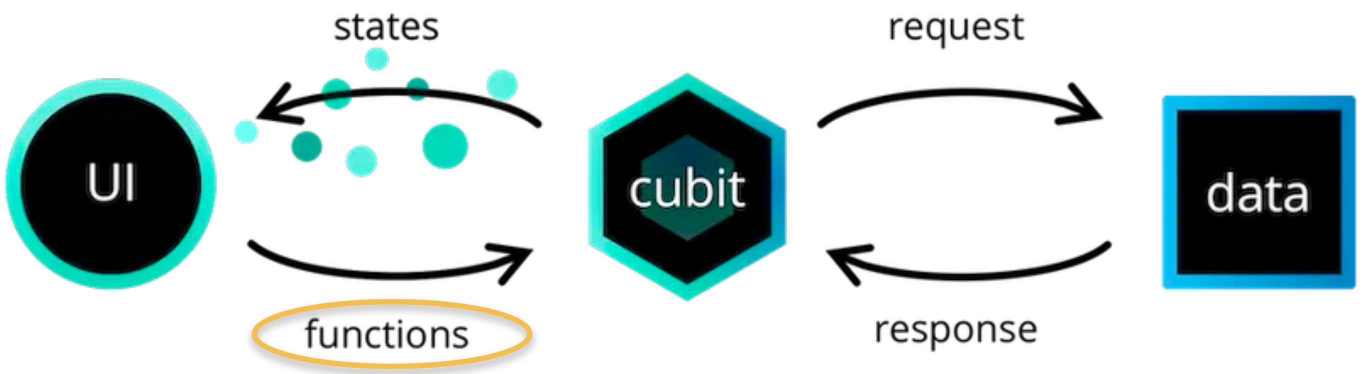
```
class CounterCubit extends Cubit<int> {
  CounterCubit() : super(0);

  void increase() => emit(state + 1);
  void decrease() => emit(state - 1);
}
```

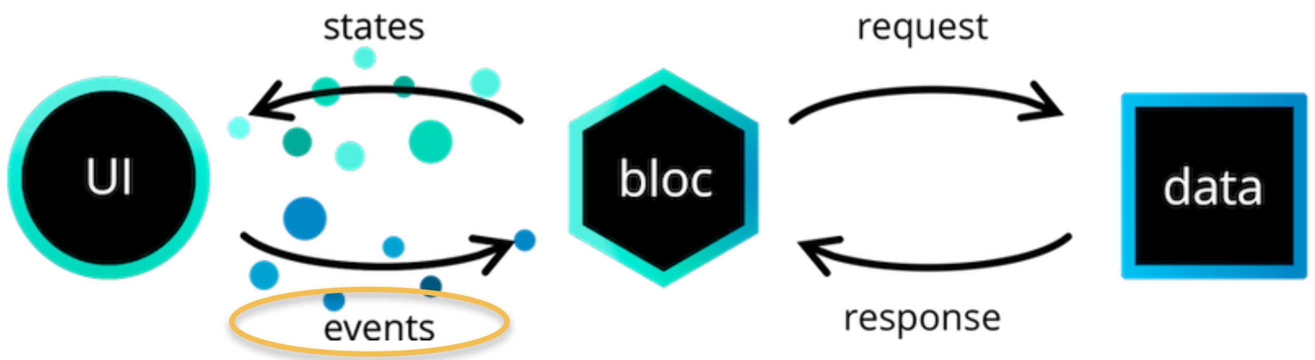
Bloc **Cubit**

Cubit (1)

- Cubit is a **minimal** version or a subset of the **BLoC** design pattern that simplifies the way we manage the state of an application.
- It **substitutes the use of events** (used in Bloc) **with functions** that rebuild the UI by emitting different states on a stream.



VS



Cubit (2)

- We start by creating a cubit

Extending Cubit and specifying the type for our state

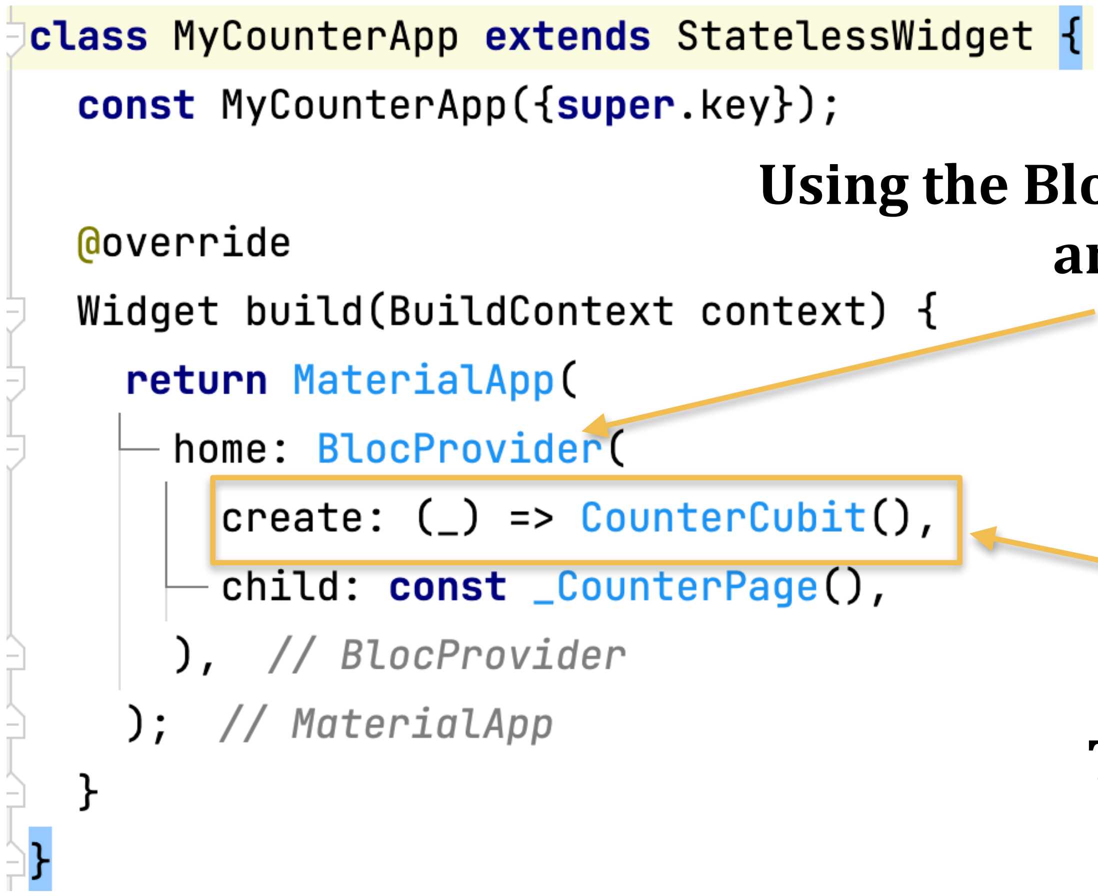
```dart
class CounterCubit extends Cubit<int> {
  CounterCubit() : super(0);

  void increase() => emit(state + 1);
  void decrease() => emit(state - 1);
}
```

Specifying the initial state

Cubit (3)

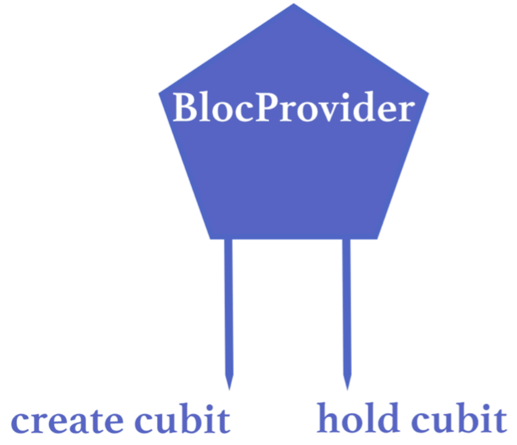
- So putting creating and providing our Cubit in the MyApp Widget means every widget below it will have access to the Cubit.



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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: BlocProvider(
        create: (_) => CounterCubit(),
        child: const _CounterPage(),
      ), // BlocProvider
    ); // MaterialApp
  }
}
```

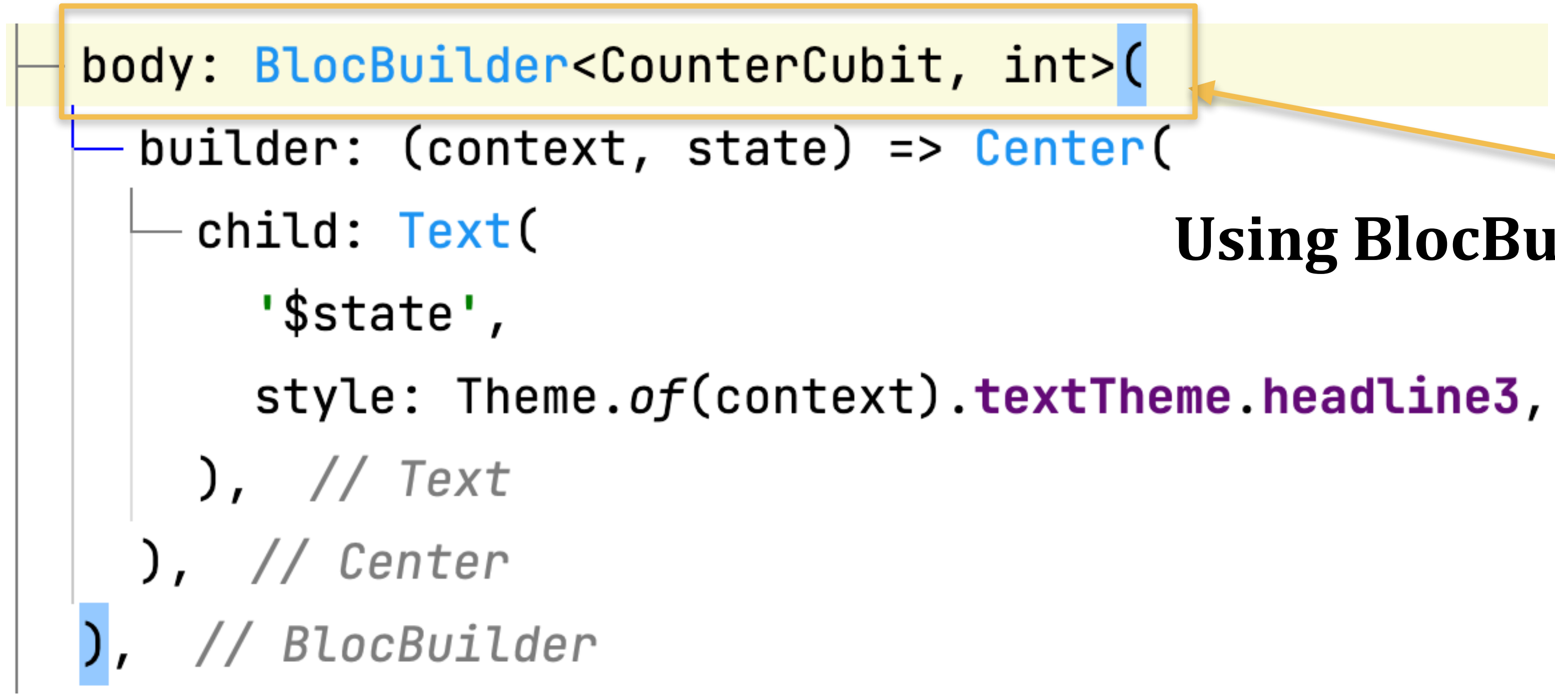
Using the BlocProvider as a dependency injector to create and provide to the child widget tree



The create method, creates the cubit and makes it accessible in the context

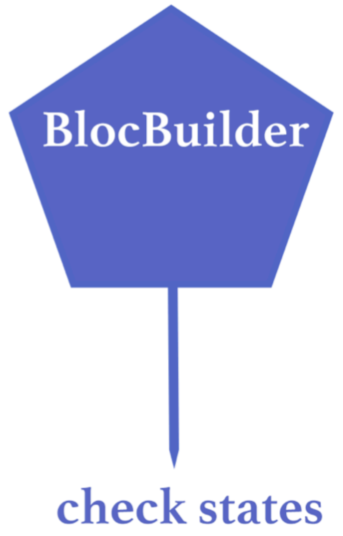
Cubit (4)

- We can use it either using the widget BlocBuilder as such



```
body: BlocBuilder<CounterCubit, int>(
  builder: (context, state) => Center(
    child: Text(
      '$state',
      style: Theme.of(context).textTheme.headline3,
    ), // Text
  ), // Center
), // BlocBuilder
```

Using BlocBuilder to get access to our Cubit



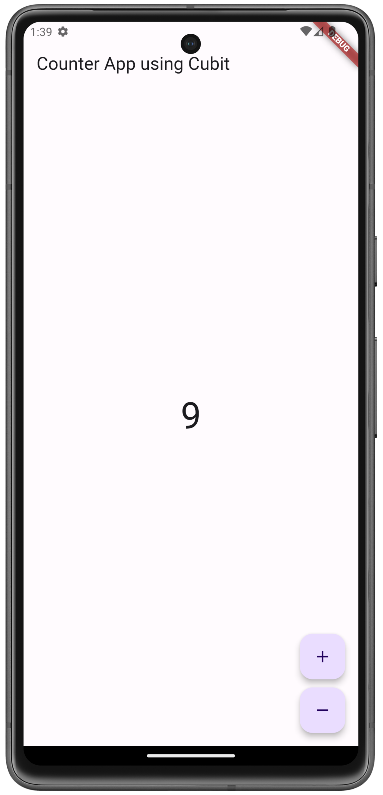
Cubit (5)

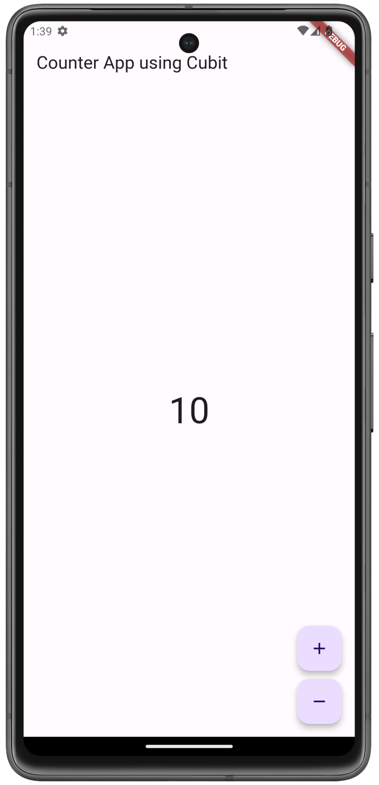
- To call the method in Cubit

```dart
floatingActionButton: Column(
  mainAxisAlignment: MainAxisAlignment.end,
  crossAxisAlignment: CrossAxisAlignment.end,
  children: [
    FloatingActionButton(
      child: const Icon(Icons.add),
      onPressed: () => context.read<CounterCubit>().increase()),
    const SizedBox(height: 10,),
    FloatingActionButton(
      child: const Icon(Icons.remove),
      onPressed: () => context.read<CounterCubit>().decrease())
  ],
), // Column
```

```dart
class CounterCubit extends Cubit<int> {
  CounterCubit() : super(0);

  void increase() => emit(state + 1);
  void decrease() => emit(state - 1);
}
```

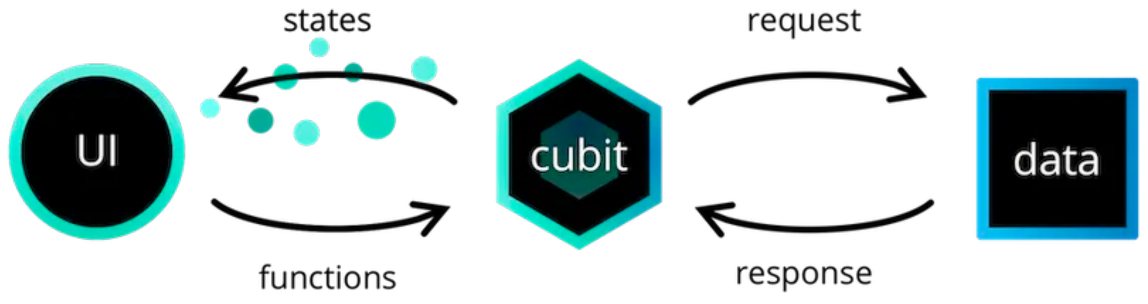


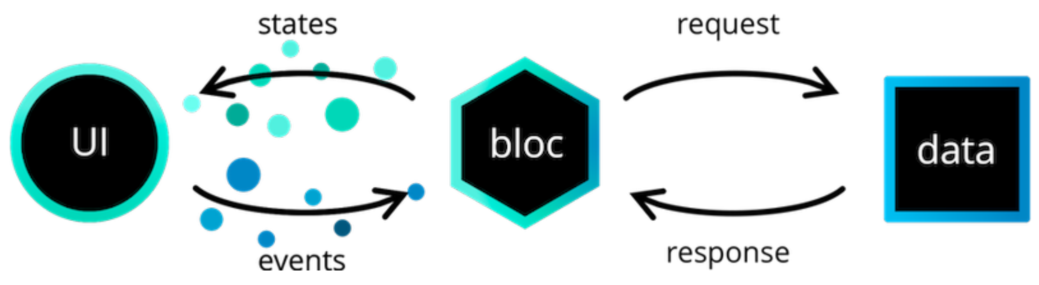


Summary (1)

- Key Differences Between Cubit and Bloc

Feature	Cubit	Bloc
Complexity	Simpler, less boilerplate	More structured, more boilerplate
Usage	Direct method calls for state changes	Events are added, and Bloc reacts by emitting states
Recommended for	Small to medium applications	Medium to large applications
Code Overhead	Lower	Higher due to event-to-state mapping





Summary (2)

- The **flutter_bloc** package provides several core widgets to help manage and build UIs based on **Bloc** states
- Here's an introduction to each of the main widgets, including
  1. **BlocProvider**
  2. **BlocBuilder**
  3. **BlocConsumer**
  4. **BlocListener**
  5. **BlocSelector**

Summary (3)

- **BlocProvider** is a widget that provides an instance of a **Bloc** or **Cubit** to the widget tree, making it available to any widget in the subtree. It’s typically used to create a **Bloc** or **Cubit** instance and inject it into the widget tree.

```
class CounterApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return BlocProvider(
      create: (context) => CounterCubit(), // Create an instance of CounterCubit
      child: MaterialApp(
        home: CounterPage(),
      ),
    );
  }
}
```

Summary (4)

- **BlocBuilder** is a widget that **rebuilds the UI in response to state changes** from a specific Bloc or Cubit. It listens to state changes and rebuilds whenever the state changes.

```
class CounterPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Counter')),
      body: Center(
        child: BlocBuilder<CounterCubit, int>(
          builder: (context, count) {
            return Text(
              'Counter Value: $count',
              style: TextStyle(fontSize: 24),
            );
          },
        ),
      )
    );
  }
}
```

Summary (5)

- **BlocListener** is used to **"listen" for state changes without rebuilding the UI**. It's ideal for triggering side effects like **showing a dialog, navigating, or showing a Snackbar based on a specific state change**.

```
class CounterPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Counter')),
      body: Center(
        child: BlocListener<CounterCubit, int>(
          listener: (context, count) {
            if (count < 0) {
              ScaffoldMessenger.of(context).showSnackBar(SnackBar(content: Text('Counter cannot go below zero!')),
              );
            }
          },
        ),
      ),
    );
  }
}
```

Summary (6)

- **BlocConsumer combines the functionalities of BlocBuilder and BlocListener**. It rebuilds the UI when the state changes and also triggers side effects in response to state changes.

```
class CounterPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: BlocConsumer<CounterCubit, int>(
          listener: (context, count) {
            if (count < 0) {
              ScaffoldMessenger.of(context).showSnackBar(
                SnackBar(content: Text('Counter cannot go below zero!')),
              );
            }
          },
          builder: (context, count) {
            return Text(
              'Counter Value: $count',style: TextStyle(fontSize: 24),
            );
          },
        ),
    );
  }
}
```

Summary (7)

- **BlocSelector** is a specialized widget that allows you to **filter specific parts of the state and only rebuild the widget when that part changes**. This can help **improve performance** by preventing unnecessary rebuilds when only a small part of the state is relevant.

```
BlocSelector<CounterCubit, CounterState, bool>(
        selector: (state) => state.wasIncremented,
        builder: (context, wasIncremented) {
         return Text(
          wasIncremented
            ? 'Last action: Incremented'
            : 'Last action: Decremented',
          style: const TextStyle(fontSize: 16, color: Colors.grey),
         );
        })
```

Summary (8)

Widget	**Purpose**
BlocProvider	Provides a Bloc or Cubit to the widget tree, allowing any widget within its subtree to access it.
BlocBuilder	Rebuilds UI in response to state changes, typically used for displaying data based on the current state.
BlocListener	Listens to state changes and performs side effects (e.g., showing dialogs, navigating), but does not rebuild the UI.
BlocConsumer	Combines BlocBuilder and BlocListener for cases where both state-based UI rebuilding and side effects are needed.
BlocSelector	Selects a specific part of the state and rebuilds only when that part changes, improving performance by avoiding unnecessary rebuilds.

Resource Software Solution

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