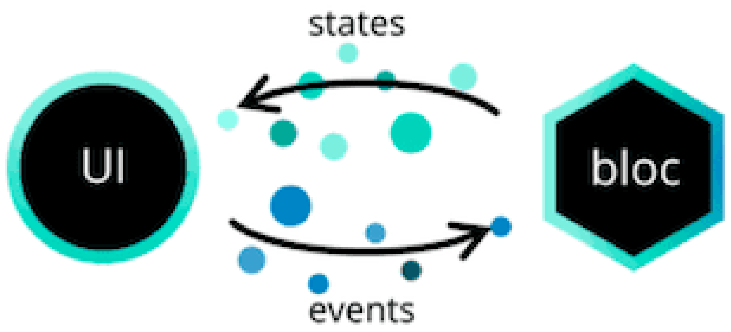
Bloc Stage Management

- The BLoC (Business Logic Component) enables us to manage complex Flutter states.

BLoC Stream Core Idea

BLoC (Business Logic Component) separates logic from UI.

- It receives `events`, processes logic, and emits `states`.
- The UI subscribes to `states` and dispatches `events`.



Events

- Represent **user actions or triggers**.
- Flow **from UI → BLoC**.
- Example:

```
abstract class CounterEvent {}
class Increment extends CounterEvent {}
class Decrement extends CounterEvent {}
```

- The UI adds events through a stream sink (e.g., bloc.add(Increment())).

States

- Represent the data snapshot at any moment.
- Flow from BLoC → UI.
- Example:

```dart
abstract class CounterState {}
class CounterInitial extends CounterState {}
class CounterValue extends CounterState {
  final int count;
  CounterValue(this.count);
}
```

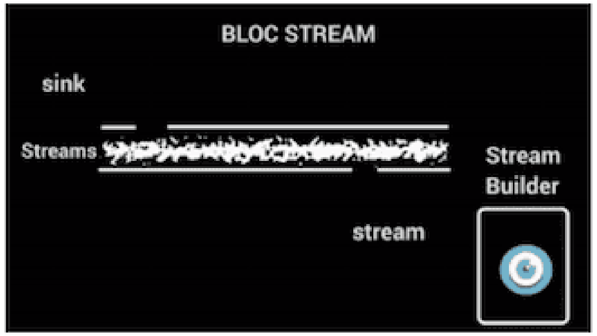
Streams

- Stream<Event> and Stream<State> are used to communicate asynchronously:
    - UI → BLoC via event stream
    - BLoC → UI via state stream

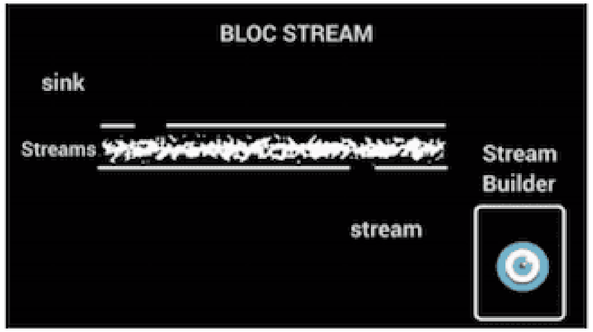
Example:

```
bloc.stream.listen((state) {
  // UI updates here
});
```

- When a Flutter Widget (UI) receives user input, it **adds an `event` to the BLoC** through the **event stream's `sink`**.
- The **BLoC listens to the event stream**, processes the event (business logic), and then **emits a new `state`** to the **state stream**.



In short:



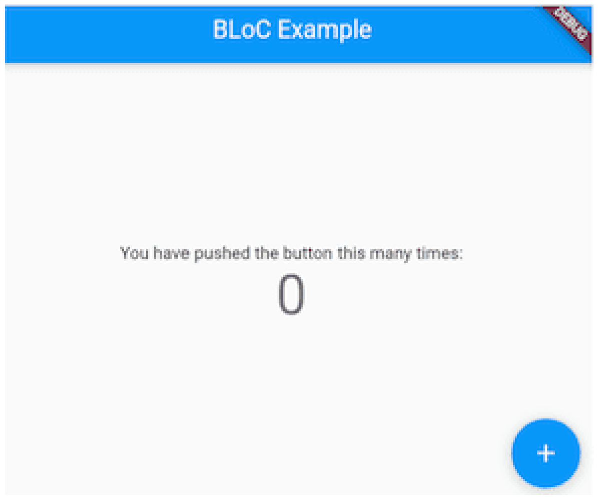
```
UI (input): User gives input
—-> add(Event): UI generates an event
—-> BLoC processes
—-> emit(State): BLoC emits a state
—-> UI rebuilds: UI Rebuilds itself accordingly
```

- The **UI uses a `StreamBuilder` or `BlocBuilder`** to rebuild itself **whenever a new state** is emitted.

Three BLoC Programming Components

We need three components to use BLoC in Flutter.

1. Events
2. Class that uses BLoC
3. StreamBuilder



- We rewrite the Flutter stateful counter example using BLoC.
- In the application, we have a _counter state that is updated with setState() and in Scaffold Widget.

```dart
int _counter = 0;
void _incrementCounter() {
   // Notify Dart UI to update screen
  setState(() { _counter++;});
}

return Scaffold(
  body: Text('$_counter'),
  ...
```

Events

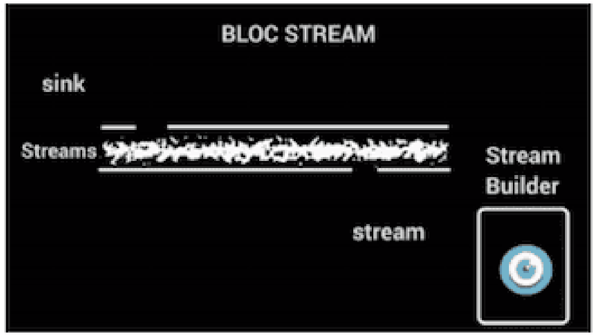
- We need to make a function that has setState() to update UI.
- For BLoC, we create an event.

```
// No BLoC
void _incrementCounter() {
  setState(...);
}
// BLoC
abstract class CounterEvent {}
class IncrementEvent extends CounterEvent {}
```

BLoC Class

- We make the state (_counter).
- We make a stream controller (StreamController).
- From the controller, we get the sink (StreamSink) and the stream (Stream).

```
class CounterBloc {
  int _counter = 0;
  final _counterStateController = StreamController<int>();
  StreamSink<int> get _inCounter => _counterStateController.sink;
  Stream<int> get counter => _counterStateController.stream;
  ...
```



- The stream (_counterEventController.stream) listens to the event and updates states in the_mapEventToState method.

```
class CounterBloc {
  CounterBloc() {
    _counterEventController.stream.listen(_mapEventToState);
  }
  ...
  void _mapEventToState(CounterEvent event) {
    if (event is IncrementEvent) {
      _counter++;
    }
    _inCounter.add(_counter);
  }
```

StreamBuilder

- The StreamBuilder gets the state using the BLoC object.

```
final _bloc = CounterBloc();
...
body: Center(
  child: StreamBuilder(
    stream: _bloc.counter,
    initialData: 0,
    ...
```

- The builder is invoked to redraw itself using the new state in the snapshot.data (_bloc.counter).

```
...
body: Center(
  child: StreamBuilder(
    ...
    builder: (BuildContext context, AsyncSnapshot<int> snapshot)
        => {
      return Column(
        children: <Widget>[
          const Text(...),
          Text(
            '${snapshot.data}',
          ), ...
```

Diagrams to Understand BLoC