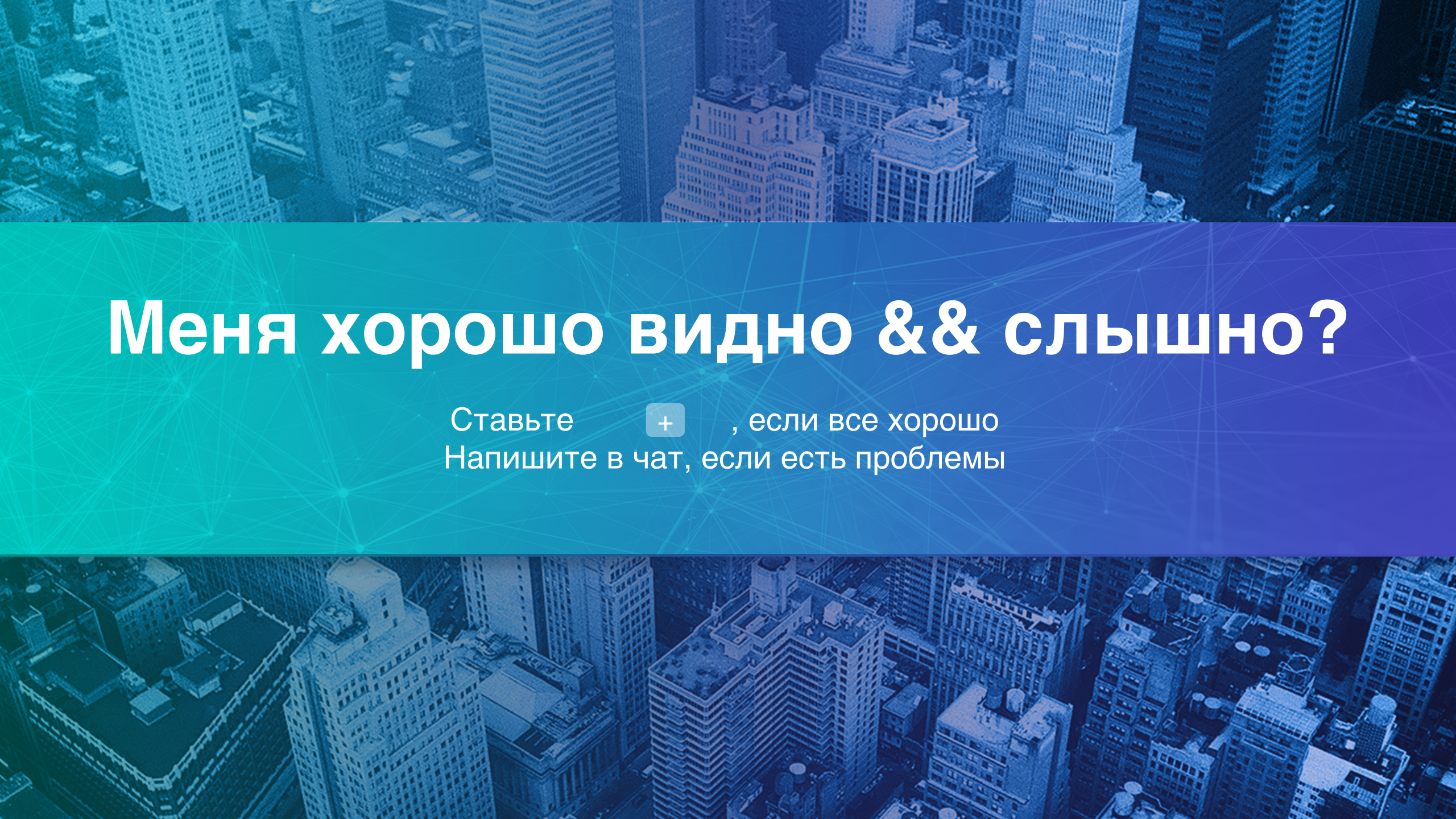

OTUS
ОНЛАЙН-ОБРАЗОВАНИЕ
Онлайн-образование
О С WWW.SW.HELP - ПРИСОЕДИ

Проверить, идет ли запись!



Меня хорошо видно && слышно?

Ставьте + , если все хорошо

Напишите в чат, если есть проблемы


Business Logic Components

Цели вебинара

1. Разберемся что такое BLoC и посмотрим как он работает.

2. Попробуем написать свой собственный bloc

3. Познакомимся с библиотекой flutter_bloc
и реализуем с ее помощью страницу коктейлей.


BLoC

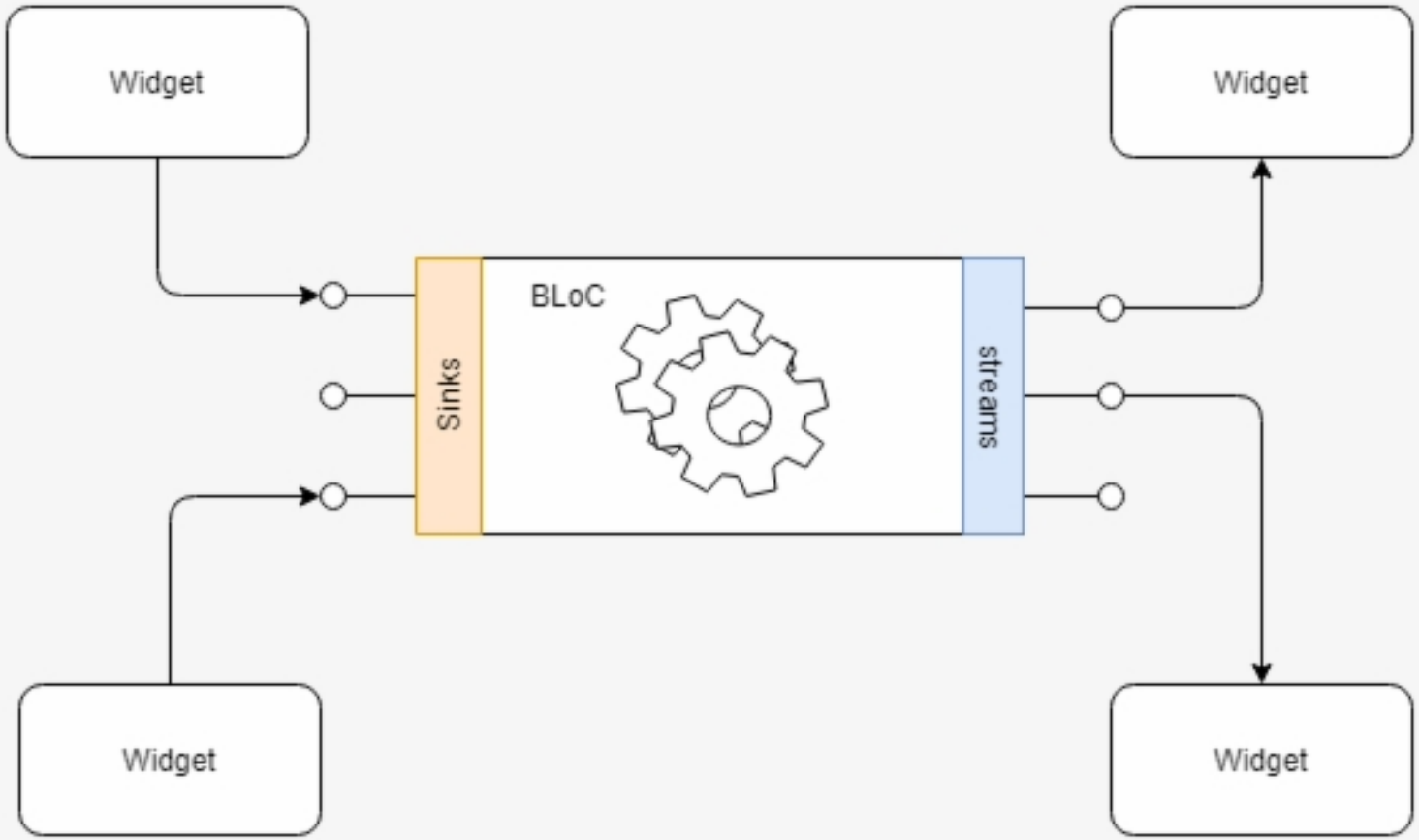
BLoC

BLoC - «Business Logic Component»

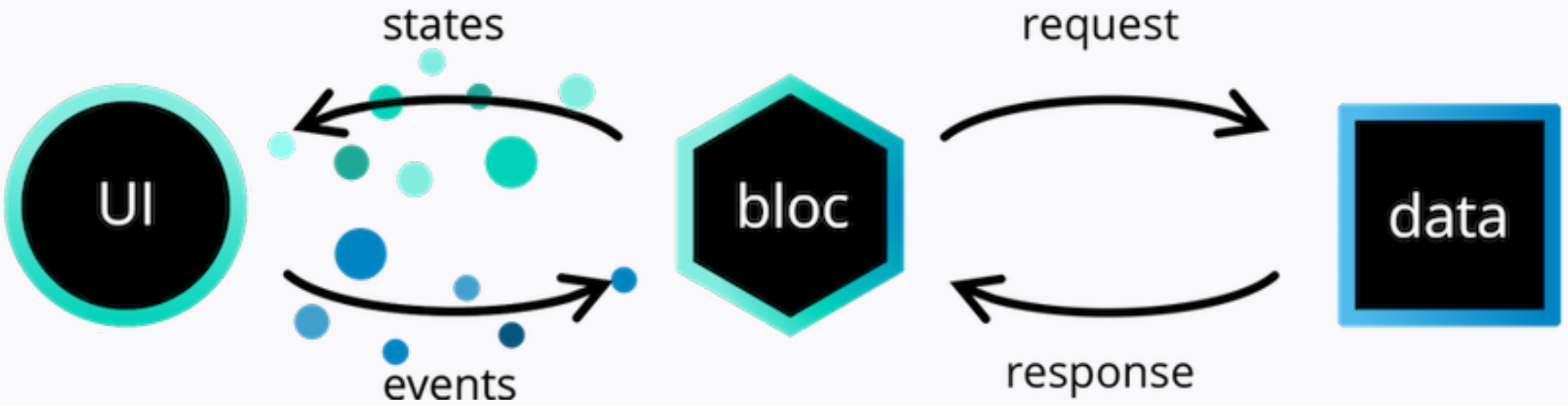
- построен на «потоках»

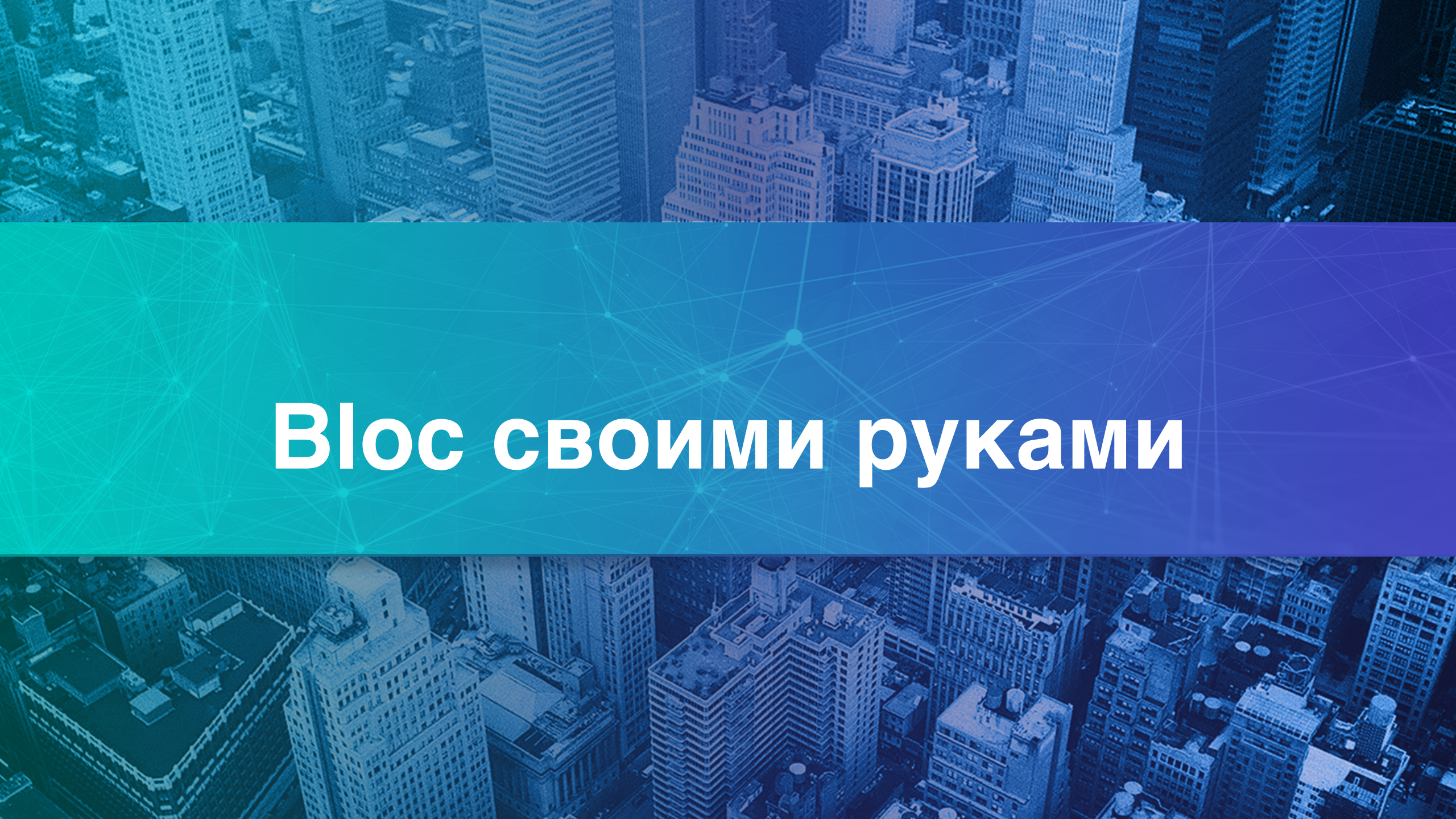
- нужен для разделения UI и «Бизнес логики»

BLoC. Схема 1

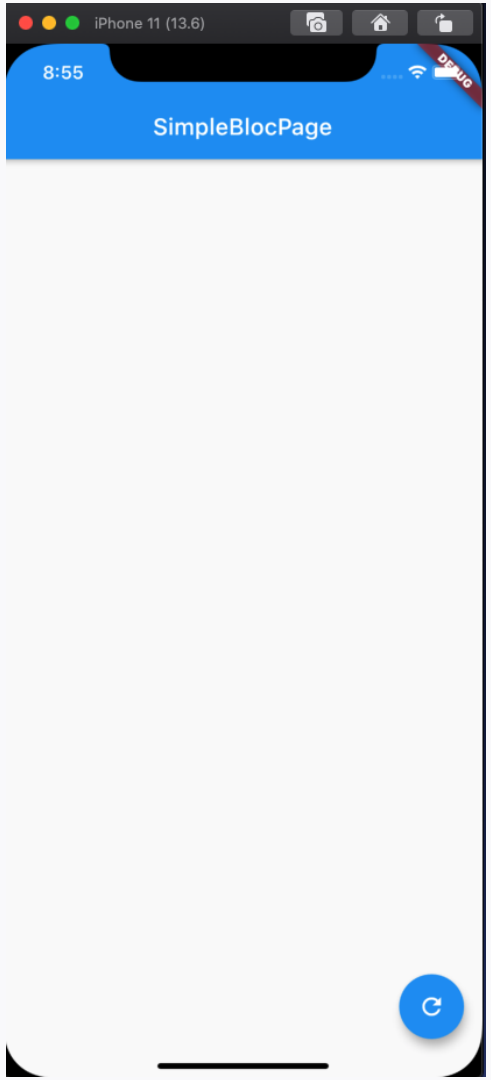


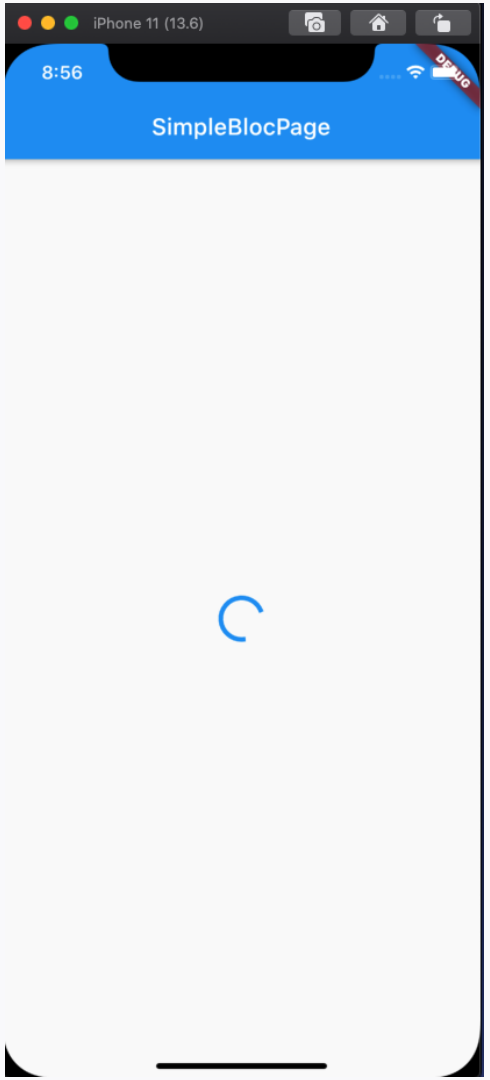
BLoC. Схема 2

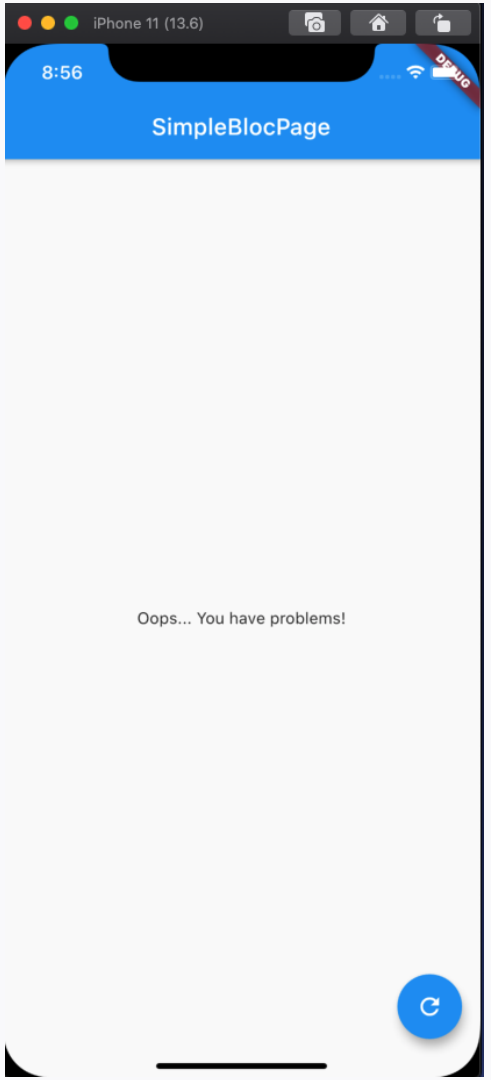


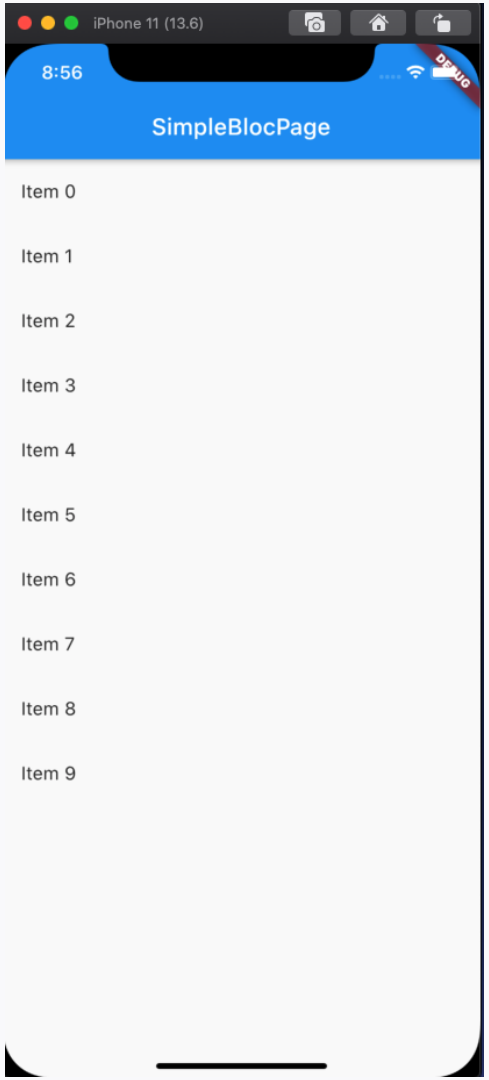

Bloc своими руками

Bloc своими руками



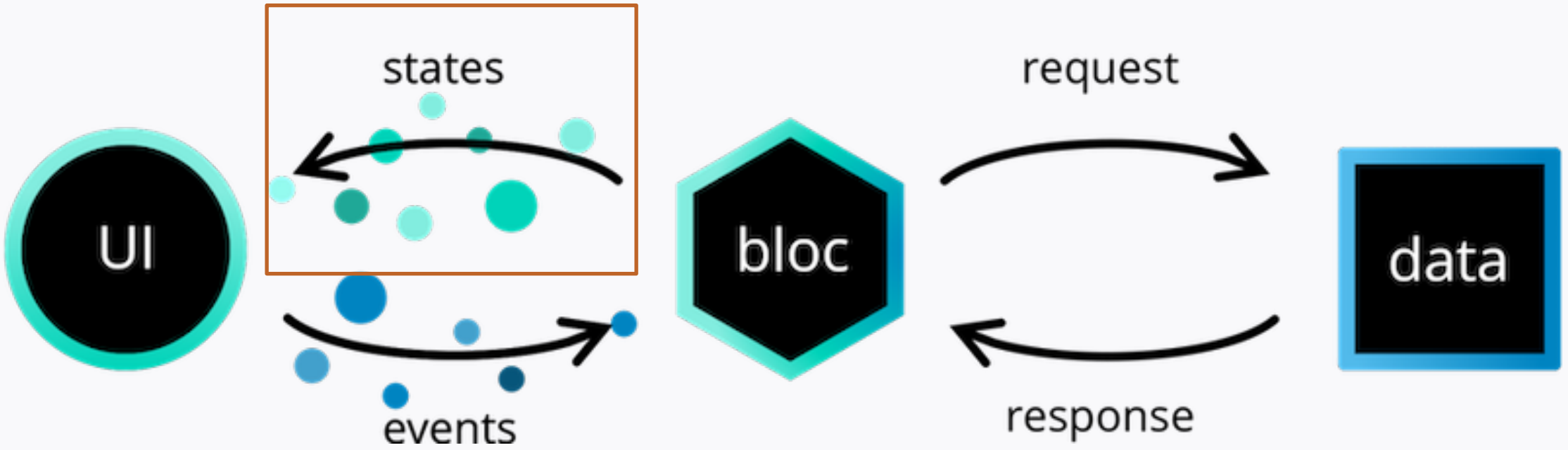






Bloc. Шаг 1 - описываем состояния

Bloc. Шаг 1 - описываем состояния



Bloc. Шаг 1 - описываем состояния

Anatomy

BlocSubject + Глагол (action) + Состояние

Examples

✅ Good

CounterInitial CounterLoadInProgress CounterLoadSuccess CounterLoadFailure

❌ Bad

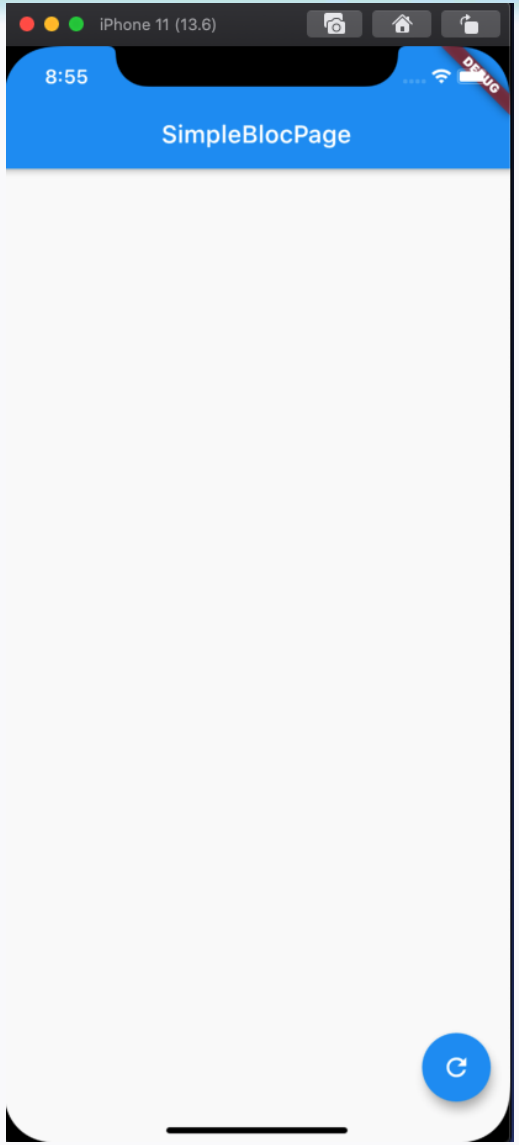
Initial Loading Success Succeeded Loaded Failure Failed

Bloc. Шаг 1 - описываем состояния

```
@immutable
abstract class SimpleBlocState {}

class SimpleBlocInitial extends SimpleBlocState {}

class SimpleBlocLoadInProgress extends SimpleBlocState {}

class SimpleBlocLoadSuccess<T> extends SimpleBlocState {
  SimpleBlocSuccess(this.items);
  final Iterable<T> items;
}

class SimpleBlocLoadFailure extends SimpleBlocState {
  SimpleBlocLoadFailure(this.errorMessage);
  final String errorMessage;
}
```

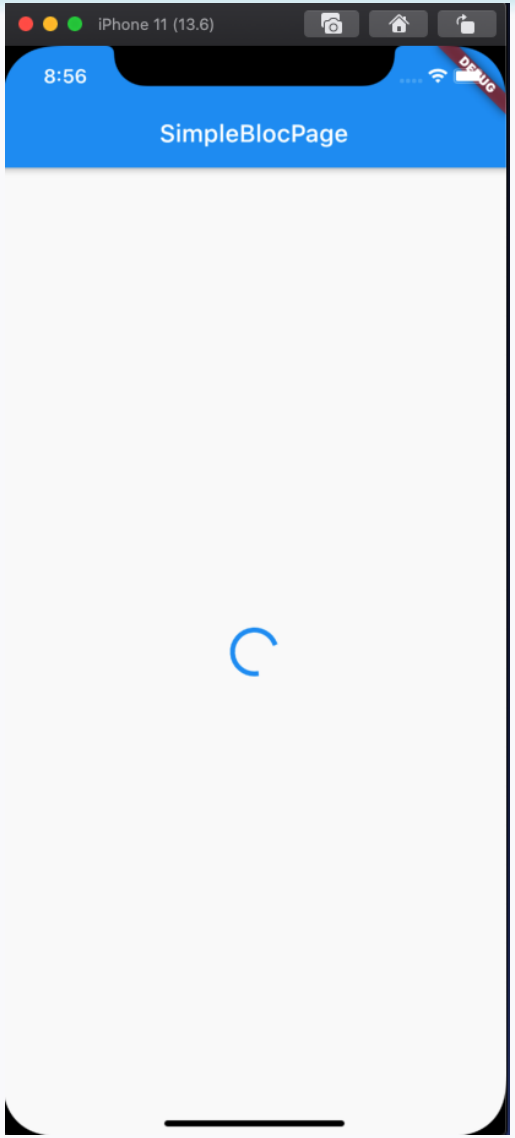
Bloc. Шаг 1 - описываем состояния

```
@immutable
abstract class SimpleBlocState {}

class SimpleBlocInitial extends SimpleBlocState {}

class SimpleBlocLoadInProgress extends SimpleBlocState {}

class SimpleBlocLoadSuccess<T> extends SimpleBlocState {
  SimpleBlocLoadSuccess(this.items);
  final Iterable<T> items;
}

class SimpleBlocLoadFailure extends SimpleBlocState {
  SimpleBlocLoadFailure(this.errorMessage);
  final String errorMessage;
}
```



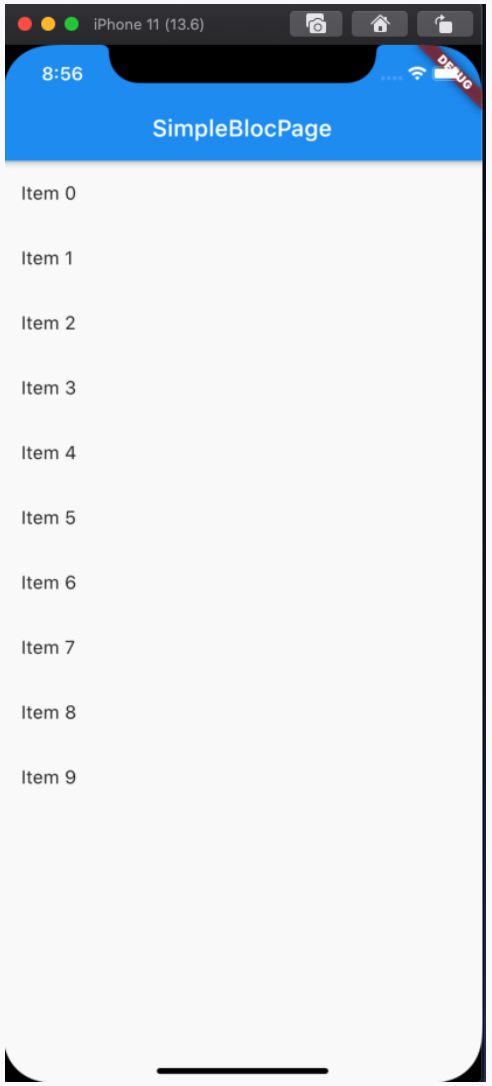
Bloc. Шаг 1 - описываем состояния

```
@immutable
abstract class SimpleBlocState {}

class SimpleBlocInitial extends SimpleBlocState {}

class SimpleBlocLoadInProgress extends SimpleBlocState {}

class SimpleBlocLoadSuccess<T> extends SimpleBlocState {
  SimpleBlocSuccess(this.items);
  final Iterable<T> items;
}

class SimpleBlocLoadFailure extends SimpleBlocState {
  SimpleBlocFailure(this.errorMessage);
  final String errorMessage;
}
```



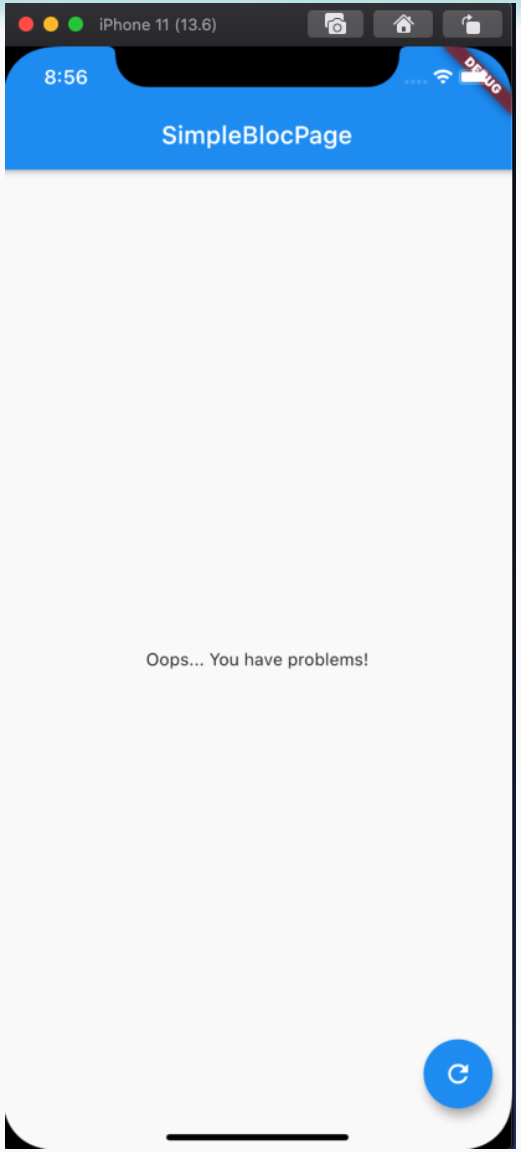
Bloc. Шаг 1 - описываем состояния

```
@immutable
abstract class SimpleBlocState {}

class SimpleBlocLoadInitial extends SimpleBlocState {}

class SimpleBlocLoadInProgress extends SimpleBlocState {}

class SimpleBlocLoadSuccess<T> extends SimpleBlocState {
  SimpleBlocLoadSuccess(this.items);
  final Iterable<T> items;
}

class SimpleBlocLoadFailure extends SimpleBlocState {
  SimpleBlocFailure(this.errorMessage);
  final String errorMessage;
}
```



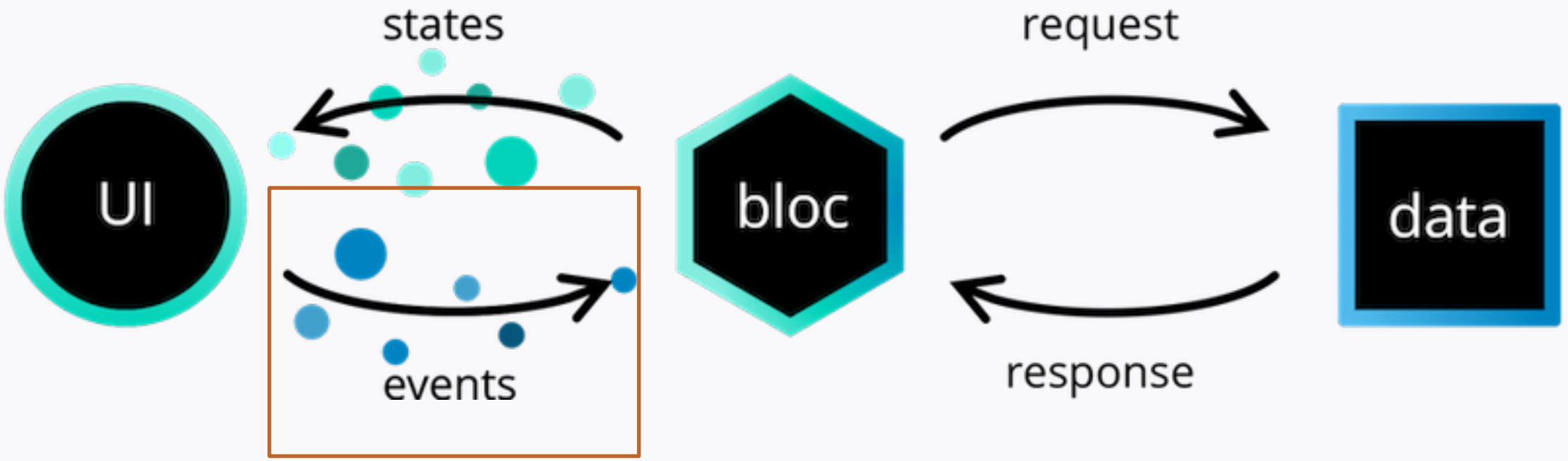
Bloc. Шаг 1 - описываем состояния

```
@immutable
abstract class SimpleBlocState {}

class SimpleBlocInitial extends SimpleBlocState {}

class SimpleBlocLoadInProgress extends SimpleBlocState {}

class SimpleBlocLoadSuccess<T> extends SimpleBlocState {
  SimpleBlocLoadSuccess(this.items);
  final Iterable<T> items;
}

class SimpleBlocLoadFailure extends SimpleBlocState {
  SimpleBlocLoadFailure(this.errorMessage);
  final String errorMessage;
}
```



Bloc. Шаг 2 - описываем события

Bloc. Шаг 2 - описываем события



Bloc. Шаг 2 - описываем события

Anatomy

BlocSubject + Существительное (optional) + Глагол (event)

Examples

✅ Good

CounterStarted CounterIncremented CounterDecremented CounterIncrementRetried

❌ Bad

Initial CounterInitialized Increment DoIncrement IncrementCounter

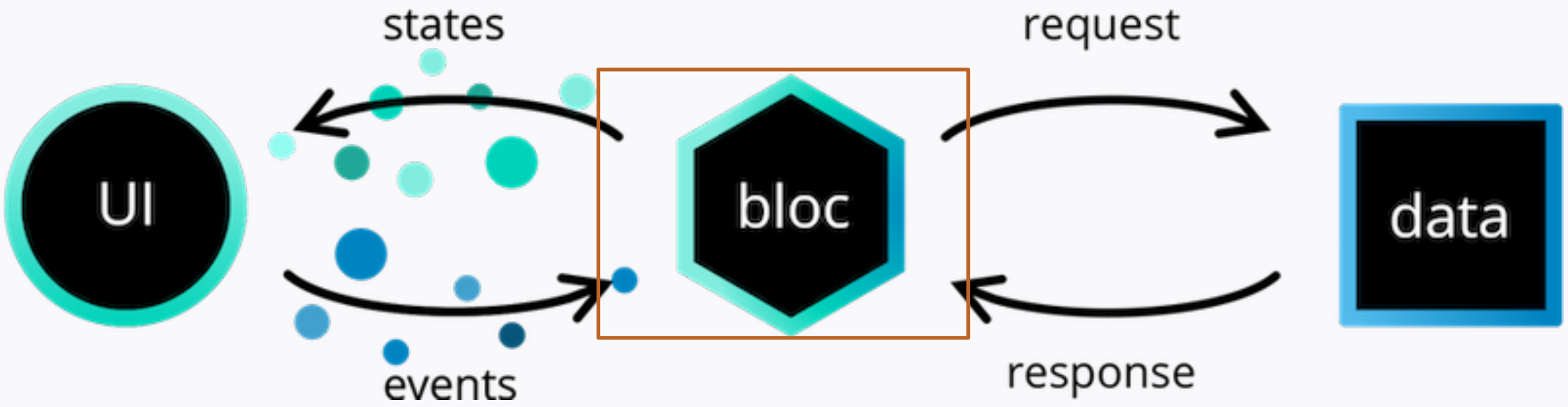
* События следует называть в прошедшем времени, потому что события - это вещи, которые уже произошли с точки зрения блока.

Bloc. Шаг 3 - создаем свой bloc

```
@immutable
abstract class SimpleBlocEvent {}

class SimpleBlocFetched extends SimpleBlocEvent {}
```

Bloc. Шаг 3 - создаем свой bloc

Bloc. Шаг 3 - создаем свой bloc



Bloc. Шаг 3 - создаем свой bloc

```
class SimpleBloc {

  final _inputController = StreamController<SimpleBlocEvent>();

  final _outputController = StreamController<SimpleBlocState>.broadcast();

  Sink<SimpleBlocEvent> get sink => _inputController.sink;

  Stream<SimpleBlocState> get stream => _outputController.stream;

  SimpleBloc() {
    _inputController.stream.listen(_onEvent);
    _outputController.add(SimpleBlocInitial());
  }

  void _onEvent(event) {
    ...
  }

  void close() {
    ...
  }
}
```

Bloc. Шаг 3 - создаем свой bloc

```
class SimpleBloc {
  final _inputController = StreamController<SimpleBlocEvent>();

  final _outputController = StreamController<SimpleBlocState>.broadcast();

 Sink<SimpleBlocEvent> get sink => _inputController.sink;

  Stream<SimpleBlocState> get stream => _outputController.stream;

  SimpleBloc() {
    _inputController.stream.listen(_onEvent);
    _outputController.add(SimpleBlocInitial());
  }

  void _onEvent(event) {
    ...
  }

  void close() {
  ...
  }

}
```

Bloc. Шаг 3 - создаем свой bloc

```
class SimpleBloc {
  final _inputController = StreamController<SimpleBlocEvent>();

  final _outputController = StreamController<SimpleBlocState>.broadcast();

  Sink<SimpleBlocEvent> get sink => _inputController.sink;

  Stream<SimpleBlocState> get stream => _outputController.stream;

  SimpleBloc() {
    _inputController.stream.listen(_onEvent);
    _outputController.add(SimpleBlocInitial());
  }

  void _onEvent(event) {
    ...
  }

  void close() {
    ...
  }

}
```

Bloc. Шаг 3 - создаем свой bloc

```
class SimpleBloc {
  final _inputController = StreamController<SimpleBlocEvent>();

  final _outputController = StreamController<SimpleBlocState>.broadcast();

  Sink<SimpleBlocEvent> get sink => _inputController.sink;

  Stream<SimpleBlocState> get stream => _outputController.stream;

  SimpleBloc() {
    _inputController.stream.listen(_onEvent);
    _outputController.add(SimpleBlocInitial());
  }

  void _onEvent(event) {
    ...
  }

  void close() {
    ...
  }

}
```

Bloc. Шаг 3 - создаем свой bloc

```
class SimpleBloc {

  ...

  void _onEvent(event) {

    if (event is SimpleBlocFetched) {
      _outputController.add(SimpleBlocLoadInProgress());
      _fetchItems().then((items) {
        _outputController.add(SimpleBlocLoadSuccess(items));
      }).catchError((error) {
        _outputController.add(SimpleBlocLoadFailure(error?.toString()));
      });
    }

  }

}
```

Bloc. Шаг 3 - создаем свой bloc

```
class SimpleBloc {

  ...

  void _onEvent(event) {

    if (event is SimpleBlocFetched) {
     _outputController.add(SimpleBlocLoadInProgress());
      _fetchItems().then((items) {
        _outputController.add(SimpleBlocLoadSuccess(items));
      }).catchError((error) {
        _outputController.add(SimpleBlocLoadFailure(error?.toString()));
      });
    }

  }

}
```

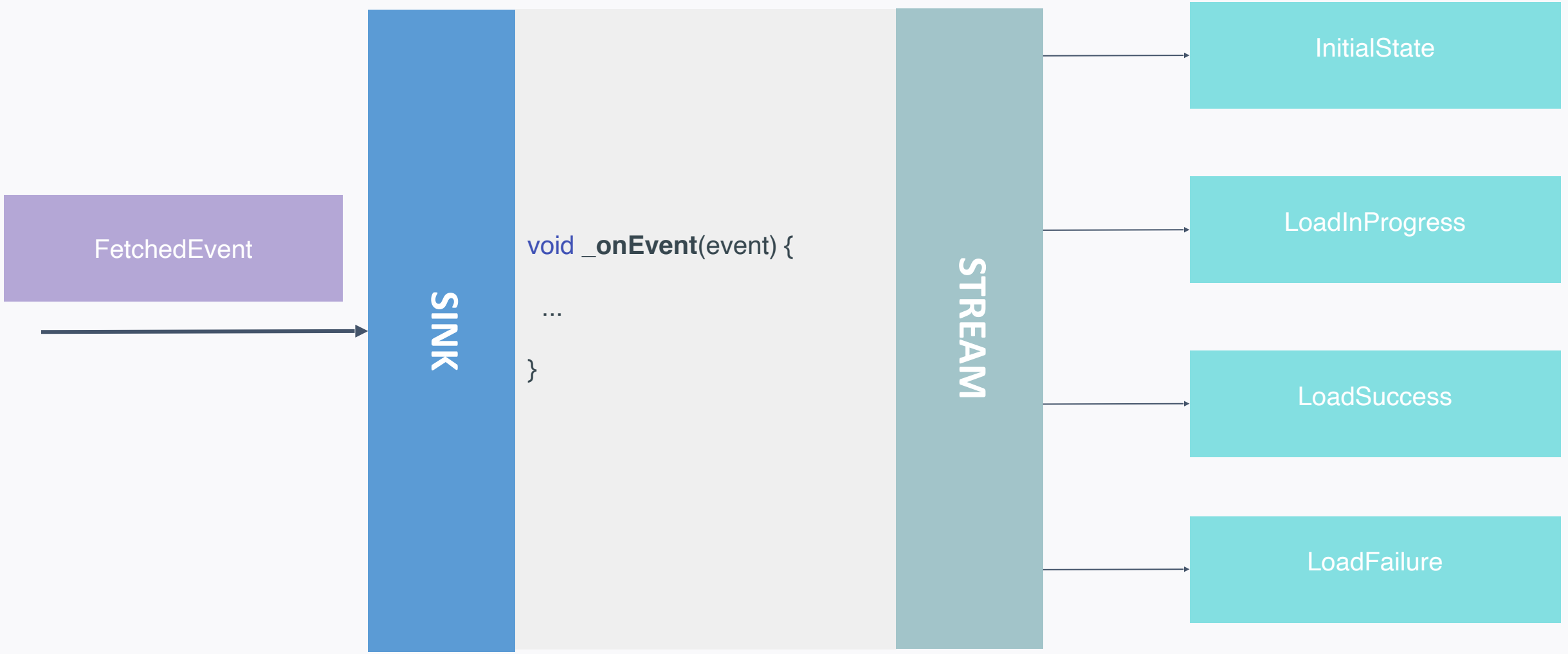
Bloc. Шаг 3 - создаем свой bloc

```
class SimpleBloc {

  ...

  void _onEvent(event) {

    if (event is SimpleBlocFetched) {
      _outputController.add(SimpleBlocLoadInProgress());
      _fetchItems().then((items) {
        _outputController.add(SimpleBlocLoadSuccess(items));
      }).catchError((error) {
        _outputController.add(SimpleBlocLoadFailure(error?.toString()));
      });
    }

  }

}
```

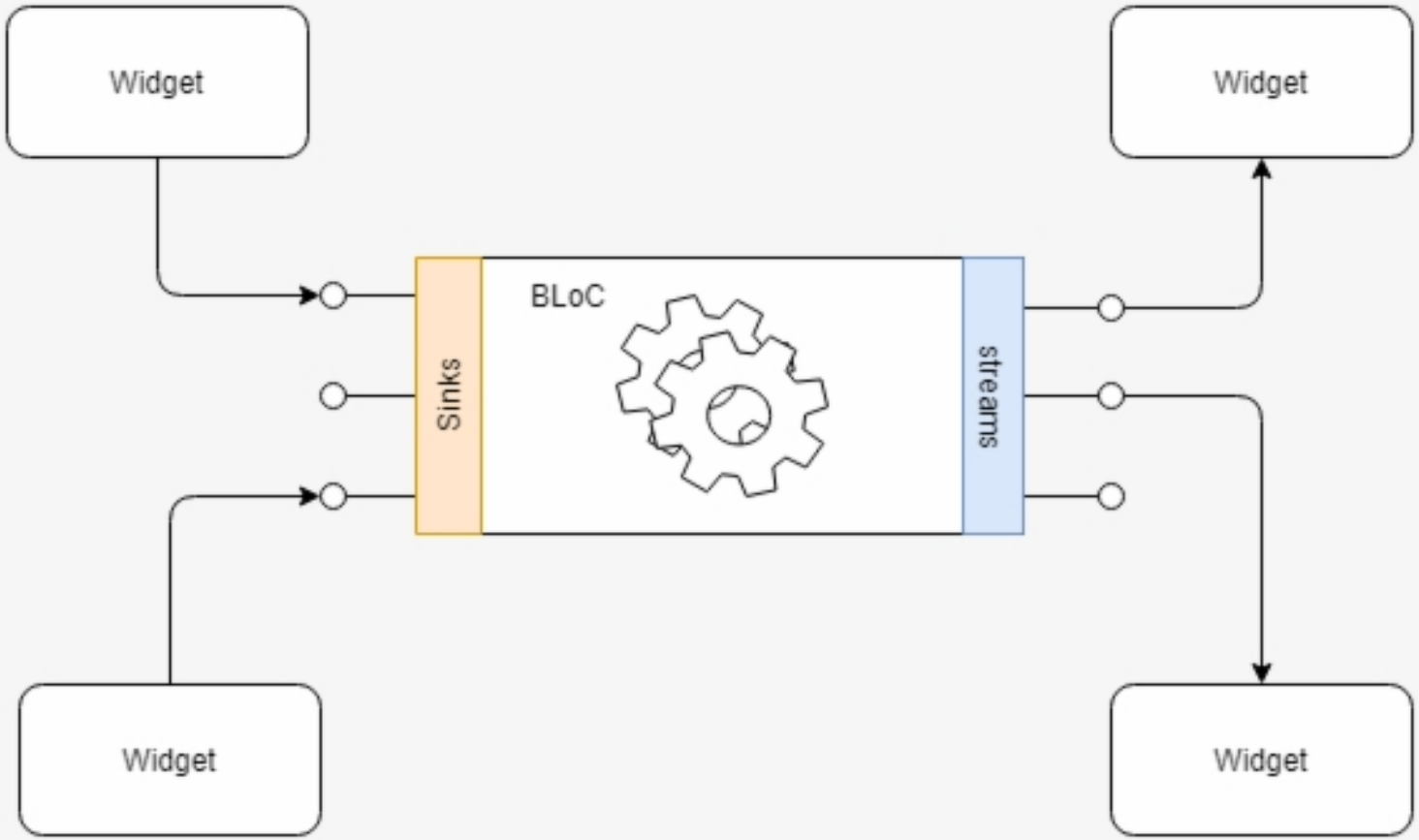
Bloc. Шаг 3 - создаем свой bloc

```
class SimpleBloc {

  ...

  void _onEvent(event) {

    if (event is SimpleBlocFetched) {
      _outputController.add(SimpleBlocLoadInProgress());
      _fetchItems().then((items) {
        _outputController.add(SimpleBlocLoadSuccess(items));
      }).catchError((error) {
        _outputController.add(SimpleBlocLoadFailure(error?.toString()));
      });
    }

  }

}
```

SimpleBloc схема



BLoC. Схема 1



Bloc. Шаг 4 - создаем UI

Bloc. Шаг 4 - создаем UI

Bloc. Шаг 4 - создаем UI

```
class SimpleBlocWidget extends StatelessWidget {
  const SimpleBlocWidget(this.bloc, {Key key}) : super(key: key);

  final SimpleBloc bloc;

  @override
  Widget build(BuildContext context) {
    return StreamBuilder<SimpleBlocState>(
      stream: bloc.stream,
      builder: (context, snapshot) {
          ...
      },
    );
  }
}
```

Bloc. Шаг 4 - создаем UI

1. Bloc передает свое состояние через stream - соответственно для отрисовки состояния нужно использовать StreamBuilder.

2. Для передачи Bloc по дереву виджетов можно (лучше всего) использовать Provider.

3. Не забываем освободить ресурсы, в том месте, где мы создали bloc

Bloc. Шаг 4 - создаем UI

```
class SimpleBlocWidget extends StatelessWidget {
  const SimpleBlocWidget(this.bloc, {Key key}) : super(key: key);

  final SimpleBloc bloc;


  @override
  Widget build(BuildContext context) {
    return StreamBuilder<SimpleBlocState>(
      stream: bloc.stream,
      builder: (context, snapshot) {
          ...
      },
    );
  }
}
```

Bloc. Шаг 4 - создаем UI

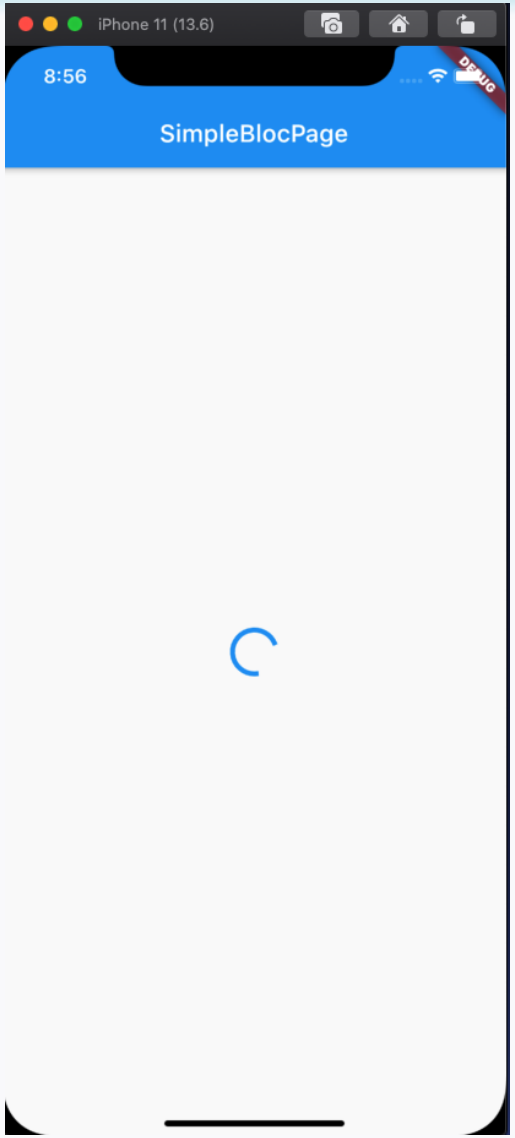
```
class SimpleBlocWidget extends StatelessWidget {
  const SimpleBlocWidget(this.bloc, {Key key}) : super(key: key);

  final SimpleBloc bloc;


 @override
  Widget build(BuildContext context) {
    return StreamBuilder<SimpleBlocState>(
      stream: bloc.stream,
      builder: (context, snapshot) {
          ...
      },
    );
  }
}
```

Bloc. Шаг 4 - создаем UI

```
return StreamBuilder<SimpleBlocState>(
    stream: bloc.stream,
    builder: (context, AsyncSnapshot<SimpleBlocState> snapshot) {
      final state = snapshot.data;
      if (state is SimpleBlocLoadInProgress) {
        return Center(child: CircularProgressIndicator());
      }
      if (state is SimpleBlocLoadSuccess) {
        final items = state.items;
        return ListView.builder(
            itemBuilder: (context, index) {
              final item = items.elementAt(index);
              return ListTile(title: Text(item));
            }, itemCount: items.length);
      }

      if (state is SimpleBlocLoadFailure) {
        return Center(child: Text(state.errorMessage));
      }

      return SizedBox();
    },
  );
```

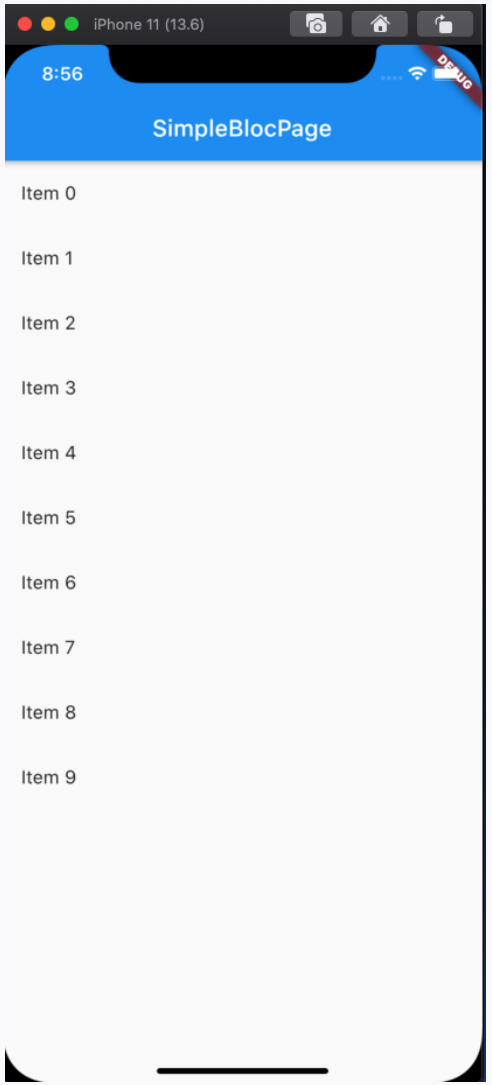
Bloc. Шаг 4 - создаем UI

```
return StreamBuilder<SimpleBlocState>(
    stream: bloc.stream,
    builder: (context, AsyncSnapshot<SimpleBlocState> snapshot) {
      final state = snapshot.data;
     if (state is SimpleBlocLoadInProgress) {
       return Center(child: CircularProgressIndicator());
     }
      if (state is SimpleBlocLoadSuccess) {
        final items = state.items;
        return ListView.builder(
            itemBuilder: (context, index) {
              final item = items.elementAt(index);
              return ListTile(title: Text(item));
            }, itemCount: items.length);
      }

      if (state is SimpleBlocLoadFailure) {
        return Center(child: Text(state.errorMessage));
      }

      return SizedBox();
    },
  );
```



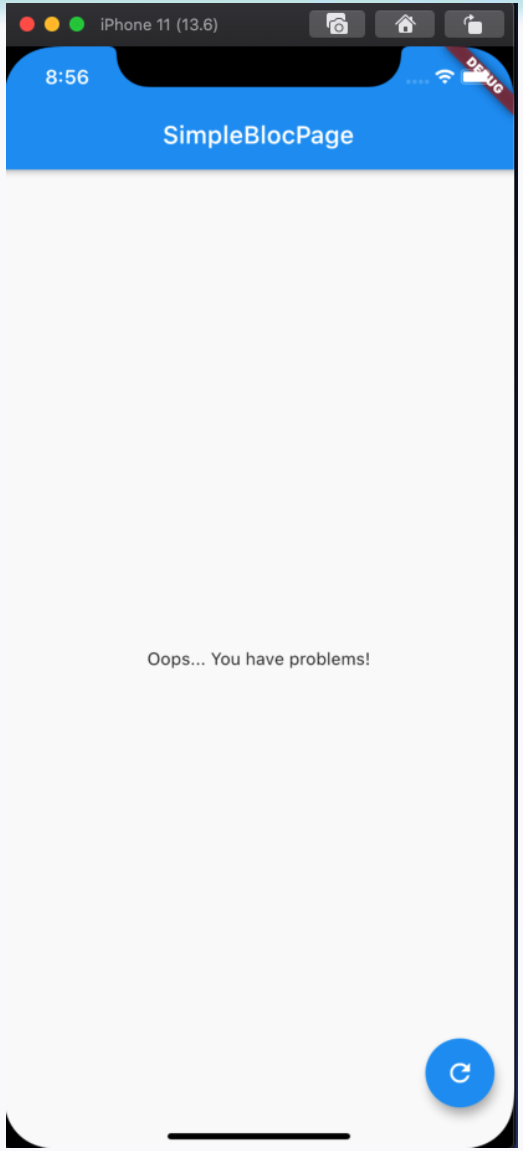
Bloc. Шаг 4 - создаем UI

```
return StreamBuilder<SimpleBlocState>(
    stream: bloc.stream,
    builder: (context, AsyncSnapshot<SimpleBlocState> snapshot) {
      final state = snapshot.data;
      if (state is SimpleBlocLoading) {
        return Center(child: CircularProgressIndicator());
      }
     if (state is SimpleBlocLoadSuccess) {
        final items = state.items;
        return ListView.builder(
            itemBuilder: (context, index) {
              final item = items.elementAt(index);
              return ListTile(title: Text(item));
            }, itemCount: items.length);
      }

      if (state is SimpleBlocLoadFailure) {
        return Center(child: Text(state.errorMessage));
      }

      return SizedBox();
    },
  );
```



Bloc. Шаг 4 - создаем UI

```
return StreamBuilder<SimpleBlocState>(
    stream: bloc.stream,
    builder: (context, AsyncSnapshot<SimpleBlocState> snapshot) {
      final state = snapshot.data;
      if (state is SimpleBlocLoading) {
        return Center(child: CircularProgressIndicator());
      }
      if (state is SimpleBlocLoadSuccess) {
        final items = state.items;
        return ListView.builder(
            itemBuilder: (context, index) {
              final item = items.elementAt(index);
              return ListTile(title: Text(item));
            }, itemCount: items.length);
      }

      if (state is SimpleBlocLoadFailure) {
        return Center(child: Text(state.errorMessage));
      }

      return SizedBox();
    },
  );
```



Bloc. Шаг 4 - создаем UI. Отправка событий

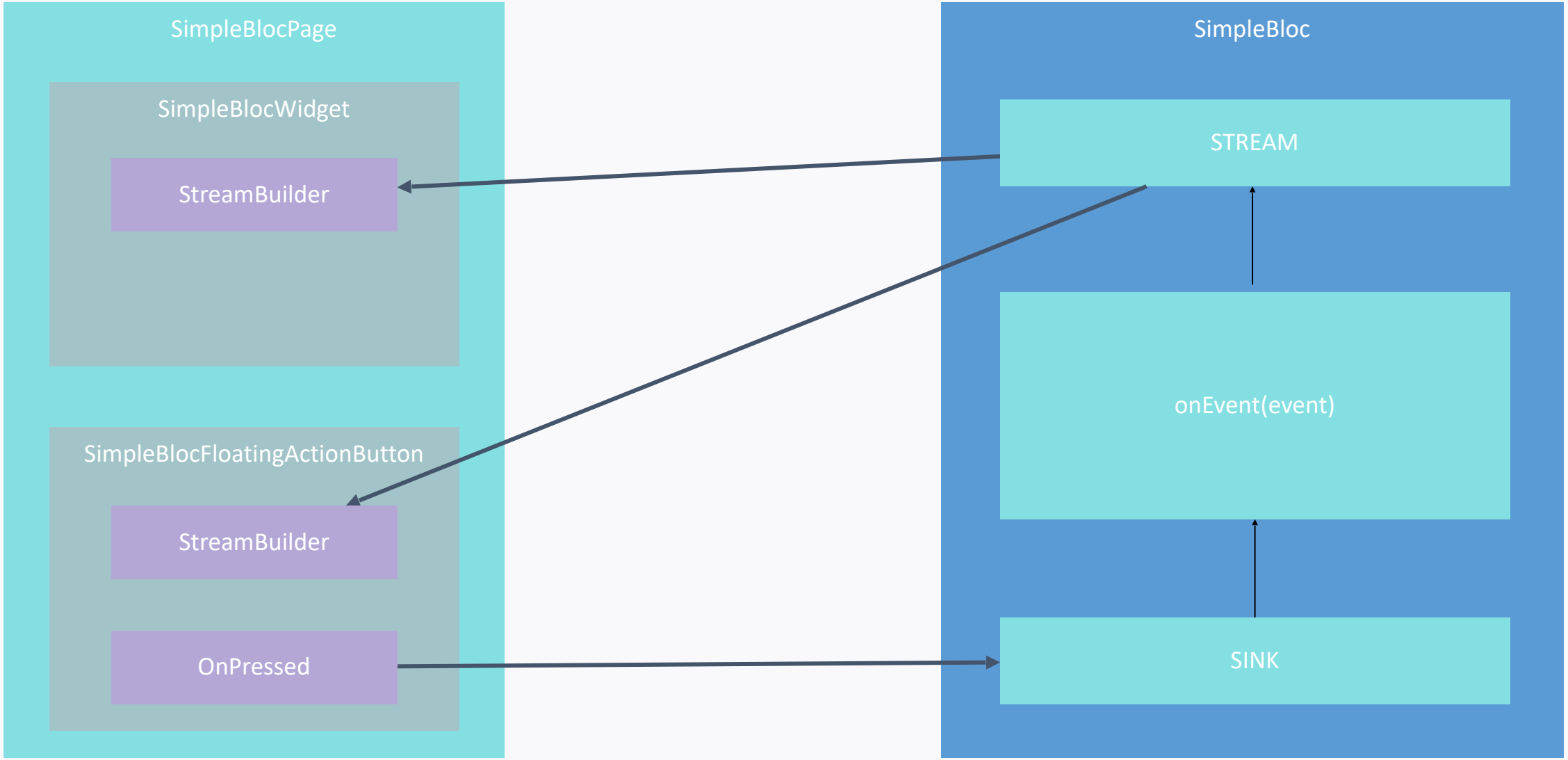
```
class SimpleBlocFetchFloatingActionButton extends StatelessWidget {
  final SimpleBloc bloc;

  const SimpleBlocFetchFloatingActionButton(this.bloc, {Key key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return StreamBuilder<SimpleBlocState>(
      stream: bloc.stream,
      builder: (context, snapshot) {
        final state = snapshot.data;
        final isVisible =  state is SimpleBlocLoadFailure
                        || state is SimpleBlocInitial;
        return Visibility(
          visible: isVisible,
          child: FloatingActionButton(
            onPressed: () => bloc.sink.add(SimpleBlocFetched()),
             child: Icon(Icons.refresh, color: Colors.white)),
        );
      },
    );
  }
}
```

Bloc. Шаг 4 - создаем UI

Не забываем закрывать bloc в том же месте где мы его создавали!!!

```
@override
 void dispose() {
   bloc.close();
   super.dispose();
 }
```

Bloc. Конечная схема

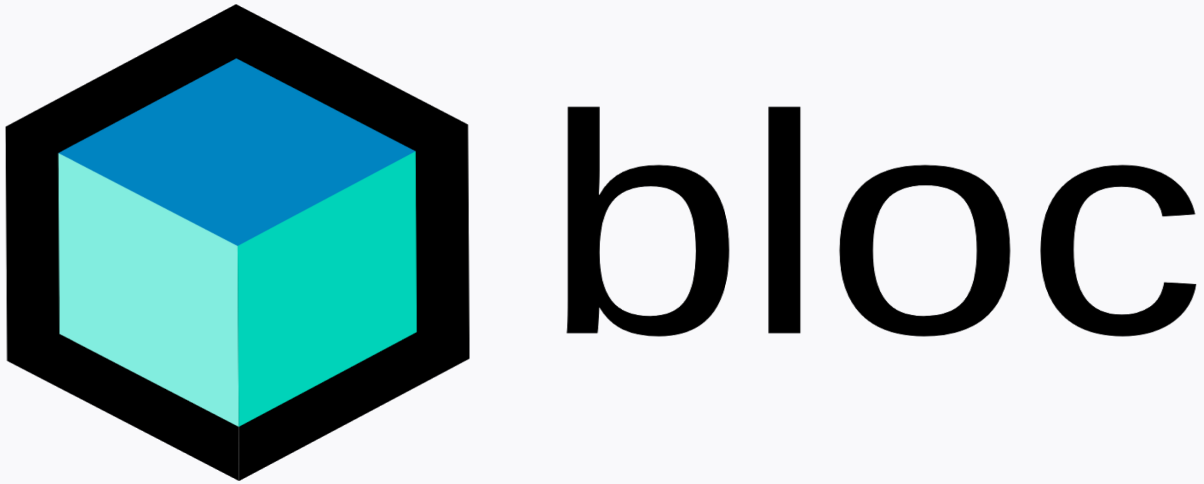


BLoC.Как еще может быть?

```
abstract class SimpleBloc {

  // Inputs
   Sink get addItem;
   Sink get deleteItem;
   Sink get editItem;
   Sink get fetch;
   Sink get clear;

  // Outputs
   Stream get items;
   Stream get progressStatus;

}
```

BLoC и библиотека bloc

BLoC. Библиотека Bloc



https://bloclibrary.dev/#/

BLoC Lib. Что есть в библиотеке?

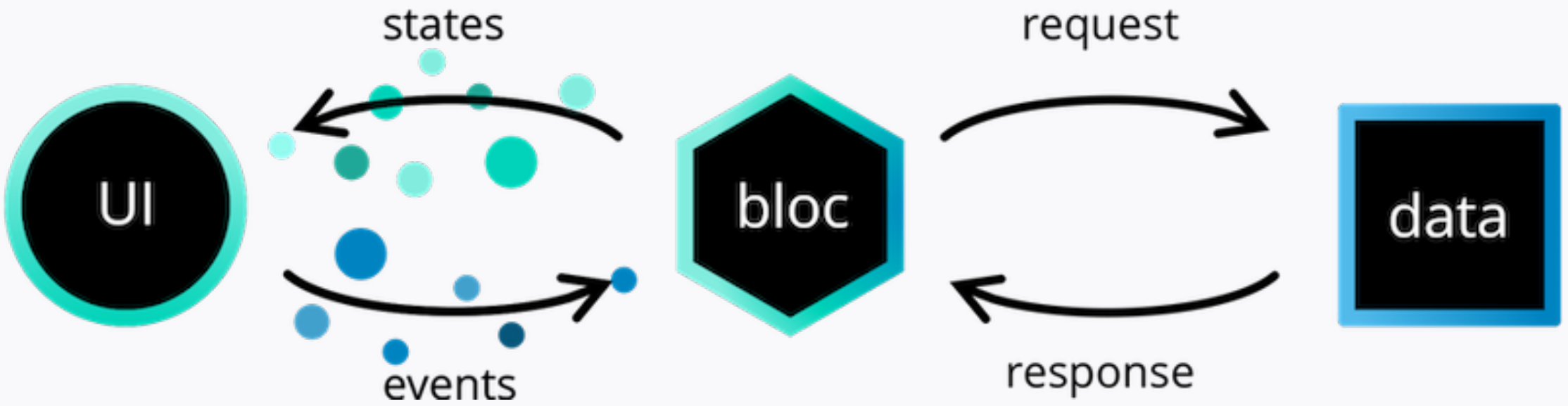
Cubit

Bloc

BlocWidgets


BLoC - Bloc

BLoC Lib. Bloc



BLoC Lib. Bloc

```
class SampleBloc extends Bloc<Event, State> {
  SampleBloc() : super(//initialState);

  @override
  Stream<State> mapEventToState(Event event) async* {



  }
}
```

BLoC Lib. Bloc

```
class SampleBloc extends Bloc<Event, State> {
  SampleBloc() : super(//initialState);

  @override
  Stream<State> mapEventToState(Event event) async* {

    // Преобразуем event в state и возвращаем с помощью yield

  }
}
```

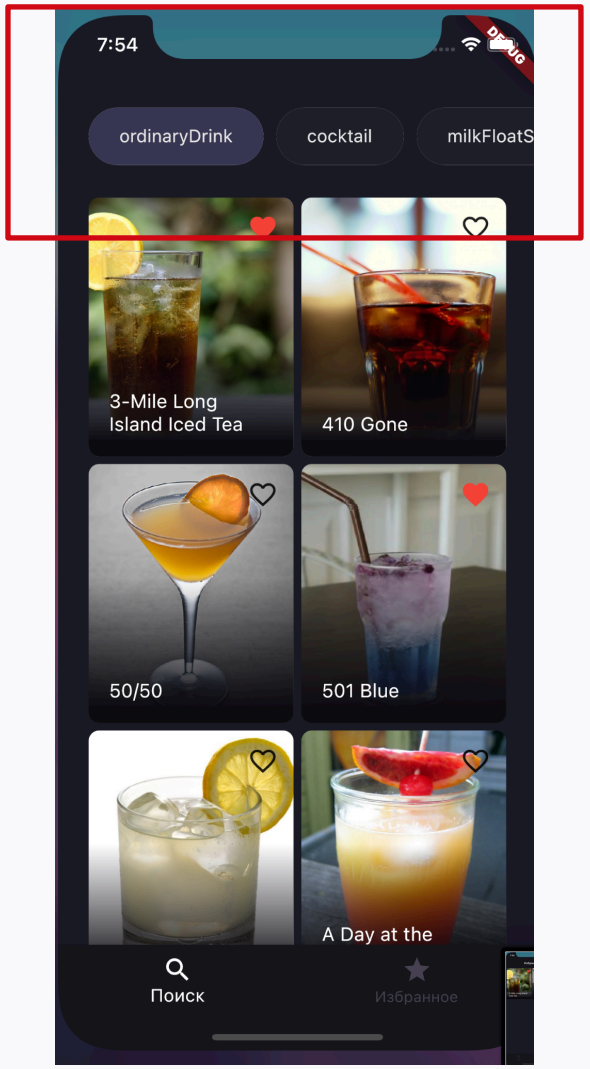
BLoC Lib. Bloc

Задание:

Мы хотим создать Bloc для фильтра категорий.

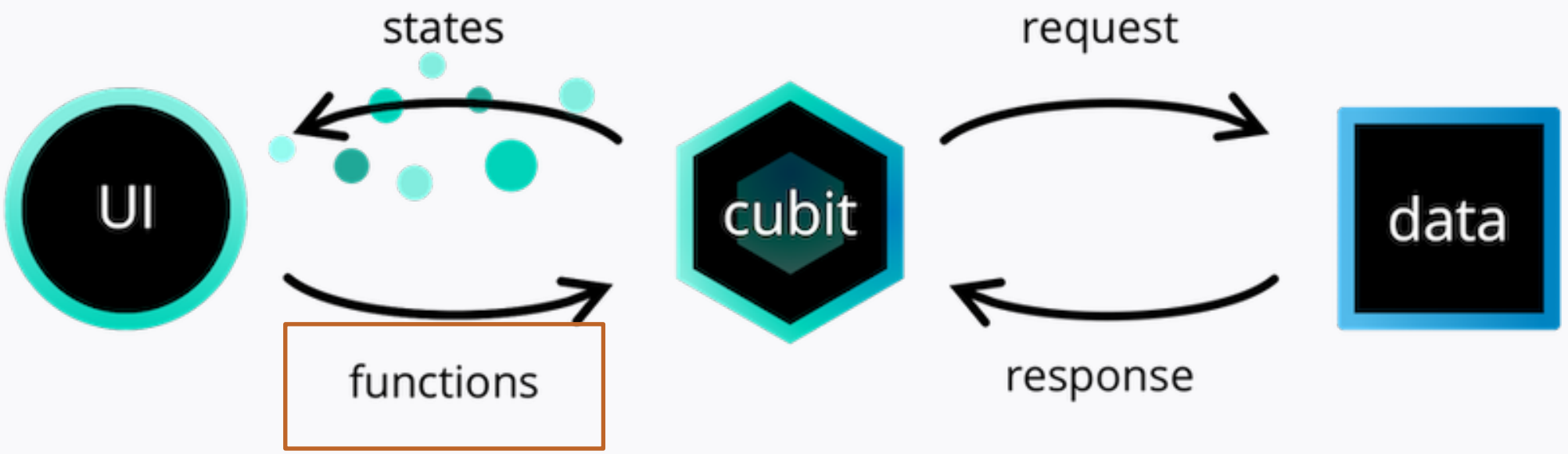
В нем отображается список всех категорий и мы можем выбирать текущую категорию.

Опишите все State и Event для CategoriesBloc




BLoC - Cubit

BLoC Lib. Cubit



BLoC Lib. Cubit.

```
class SampleCubit extends Cubit<State> {
  SampleCubit() : super(//initialState);

  void someFunction() => emit(//new state);
}
```

emit - используется для обновления state

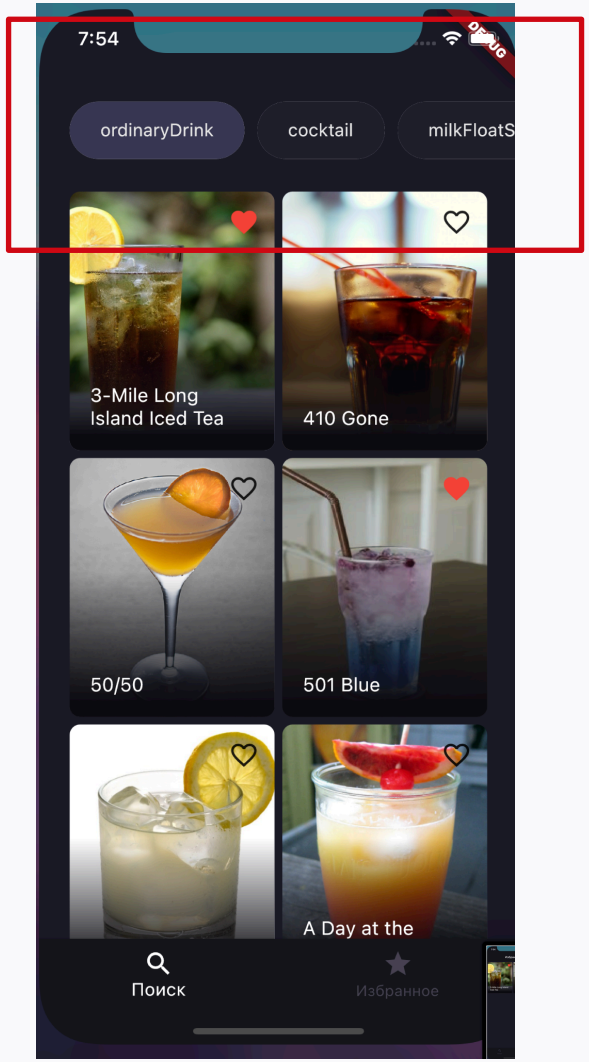
BLoC Lib. Cubit

Задание:

Предположим что для работы с фильтром категорий мы все же решили использовать Cubit.

Заполните содержимое

```
class CategoriesCubit extends Cubit<CategoriesState> {

...

}
```



BLoC. Что еще есть у Bloc и Cubit

Возможность отслеживать изменения state

```
@override
void onChange(Change<State> change) {
  print(change);
  super.onChange(change);
}
```

```
Change {
  currentState: ...,
  nextState: ...
}
```

BLoC. Что еще есть у Bloc и Cubit

Возможность отслеживать изменения state и event который вызвал это изменение.

Есть только у Bloc !!!

```
@override
void onTransition(Transition<Event, State> transition) {
  //print(transition);
  super.onTransition(transition);
}
```

```
Transition {
  currentState: ...,
  event: ...,
  nextState: ...
}
```

BLoC. Что еще есть у Bloc и Cubit

Возможность накладывать различные трансформации на Events

Есть только у Bloc !!!

```
@override
Stream<Transition<Event, State>> transformEvents(Stream<Event> events,
  TransitionFunction<Event, State> transitionFn,
) {

  return super.transformEvents(
    events.debounceTime(const Duration(milliseconds: 300)),
    transitionFn,
  );
}
```

BLoC Lib. Сравнение.

Bloc	Cubit
+ Наличие Transition	+ Меньше кода
+ Возможность трансформации входящих событий	


BLoC - BlocWidgets

BLoC Lib. BlocWidgets

Presentation

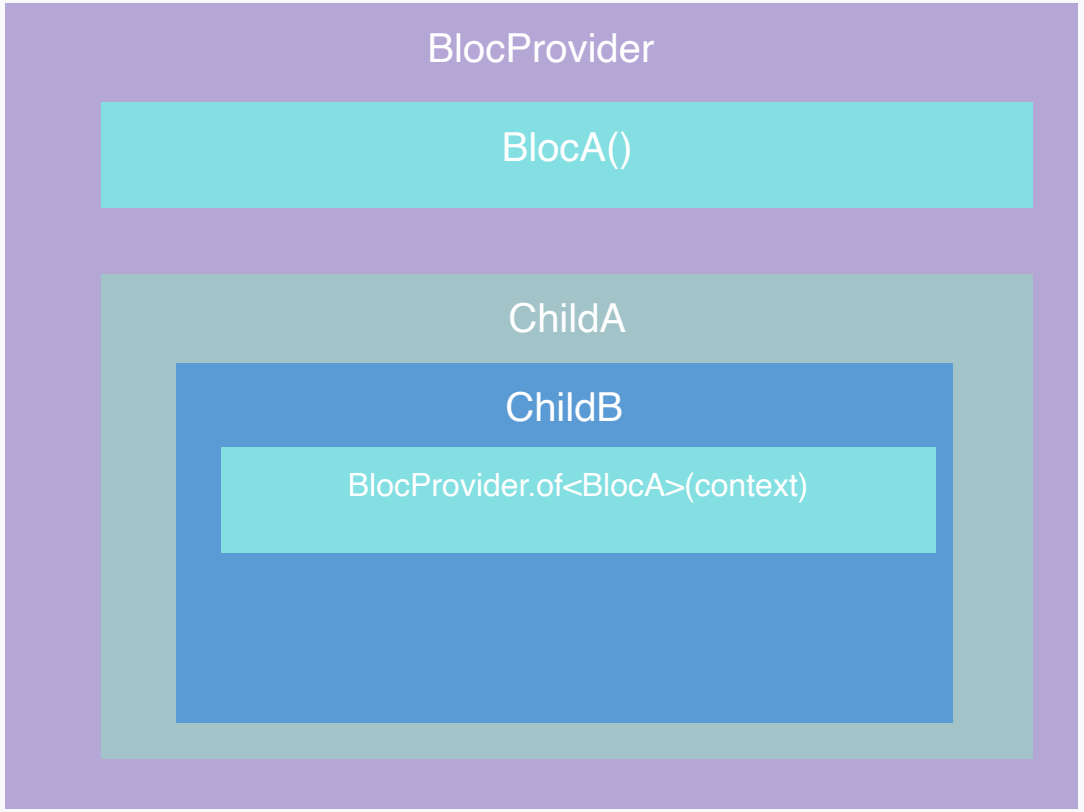
- BlocBuilder
- BlocListener
- MultiBlocListener
- BlocConsumer

Providers

- BlocProvider
- MultiBlocProvider
- RepositoryProvider
- MultiRepositoryProvider

BLoC Lib. Providers

```
BlocProvider(
  create: (BuildContext context) => BlocA(),
  child: ChildA(),
);

class ChildA extends StatelessWidget{
  @override
  Widget build(BuildContext context) => ChildB();
}

class ChildB extends StatelessWidget{
@override
  Widget build(BuildContext context) {
  var blocA = BlocProvider.of<BlocA>(context);
  ...
  }
}
```



BLoC Lib. Presentation Widgets

BlocBuilder

```
BlocBuilder<BlocA, BlocAState>(
  buildWhen: (previous,next) {
   //позволяет проверять изменения состояний и не вызывать метод build когда это не нужно
  }
  builder: (context, state) {
    // создаем widget который зависит от state
  }
)
```

BLoC Lib. Presentation Widgets

BlocListener

```
BlocListener<BlocA, BlocAState>(
  listenWhen: (previousState, state) {
    // return true/false to determine whether or not
    // to call listener with state
  },
  listener: (context, state) {
    // do stuff here based on BlocA's state
  },
  child: Container(),
)
```

BLoC Lib. Presentation Widgets

MultiBlocListener

```
MultiBlocListener(
  listeners: [
    BlocListener<BlocA, BlocAState>(
      listener: (context, state) {},
    ),
    BlocListener<BlocB, BlocBState>(
      listener: (context, state) {},
    ),
    BlocListener<BlocC, BlocCState>(
      listener: (context, state) {},
    ),
  ],
  child: ChildA(),
)
```

BLoC Lib. Presentation Widgets

BlocConsumer

Объединяет внутри себя BlocBuilder и BlocListener

```
BlocConsumer<BlocA, BlocAState>(
  listenWhen: (previous, current) {
    // return true/false to determine whether or not
    // to invoke listener with state
  },
  listener: (context, state) {
    // do stuff here based on BlocA's state
  },
  buildWhen: (previous, current) {
    // return true/false to determine whether or not
    // to rebuild the widget with state
  },
  builder: (context, state) {
    // return widget here based on BlocA's state
  }
)
```

Bloc. Дополнения

hydrated_bloc

https://pub.dev/packages/hydrated_bloc

Позволяет сохранять текущее состояние в storage.

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

  void increment() => emit(state + 1);

  @override
  int fromJson(Map<String, dynamic> json) => json['value'] as int;

  @override
  Map<String, int> toJson(int state) => { 'value': state };
}
```

replay_bloc

https://pub.dev/packages/replay_bloc

Добавляет поддержку автоматической отмены и повтора для состояний bloc и cubit.

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

  void increment() => emit(state + 1);
}
```

```dart
void main() {
  final cubit = CounterCubit();

  // trigger a state change
  cubit.increment();
  print(cubit.state); // 1

  // undo the change
  cubit.undo();
  print(cubit.state); // 0

  // redo the change
  cubit.redo();
  print(cubit.state); // 1
}
```

Заполните, пожалуйста,
опрос о занятии по ссылке в чате