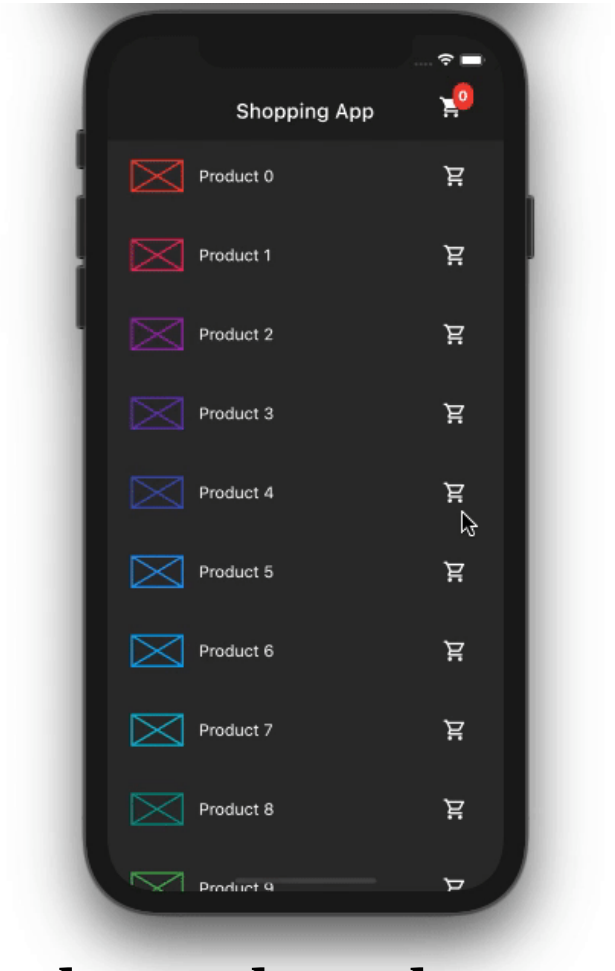
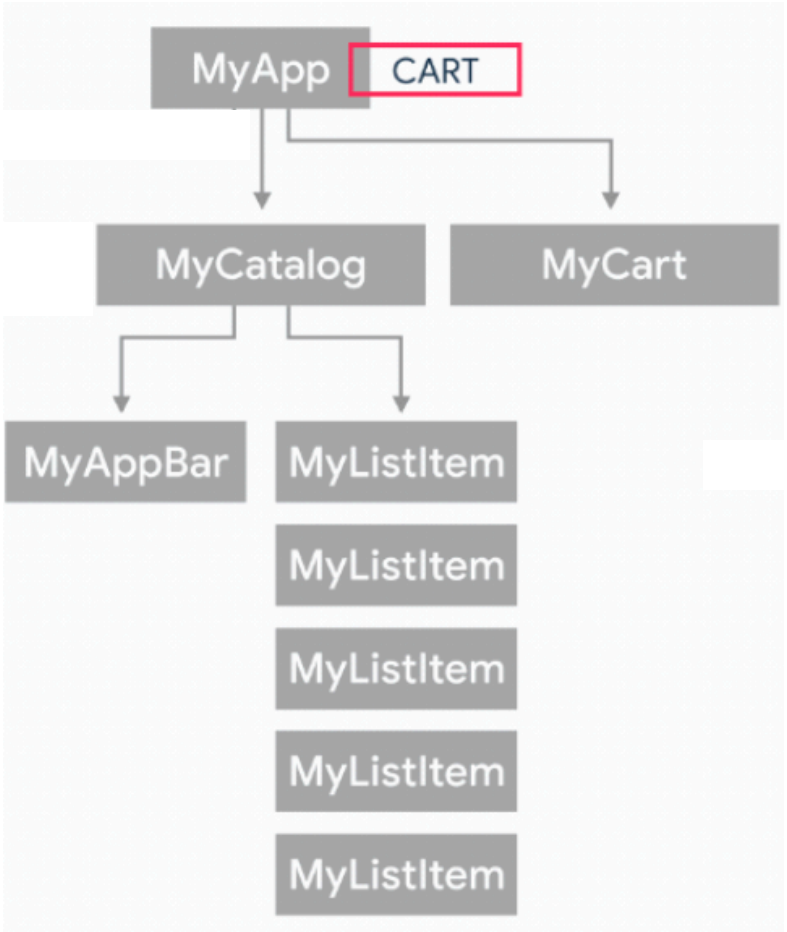
Resource Software Solution

State Management using Bloc

Introduction (1)

- Challenge: **Share application state between screens** across your app.

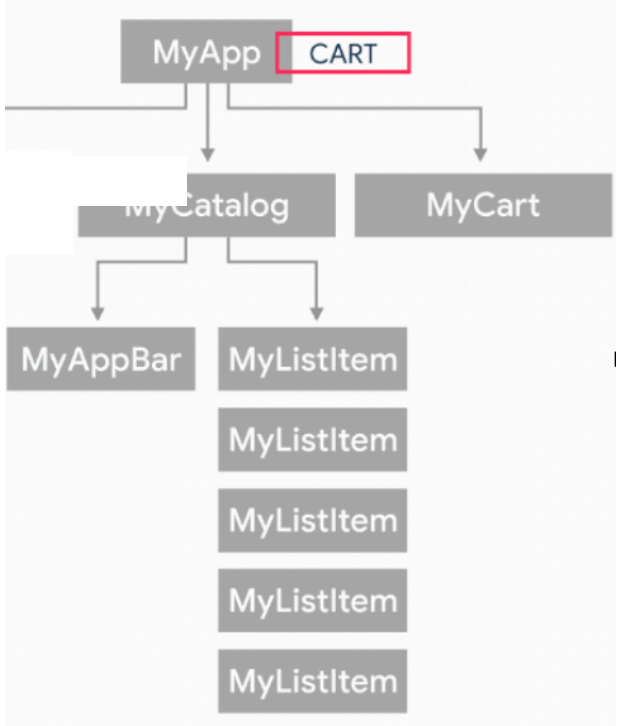


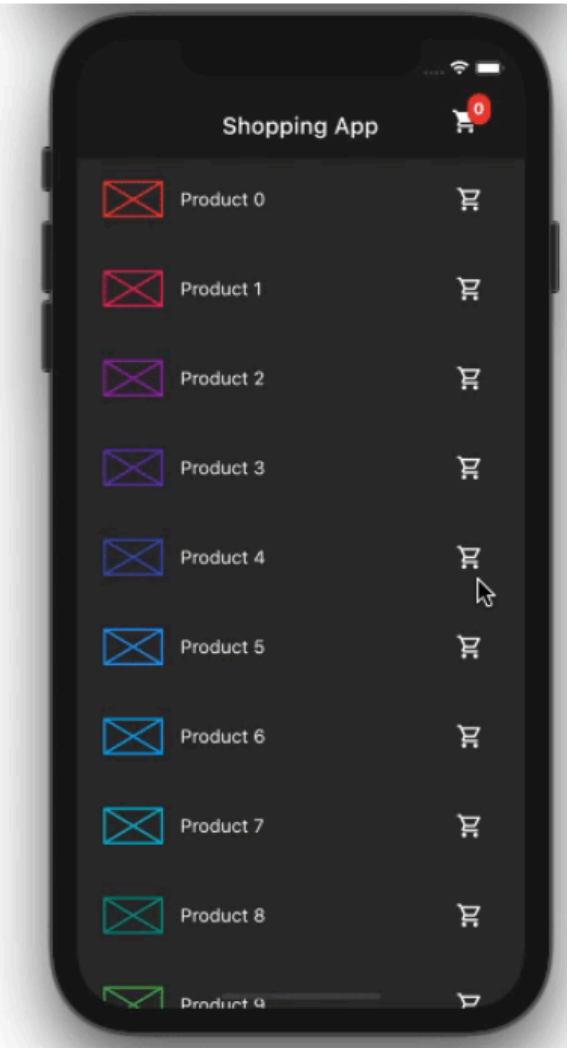


- "State" refers to the **data values that can change over time.**

Introduction (2)

- The entire UI is broken down into three classes:
  1. **home.dart** is the main file holding the scaffold and AppBar. AppBar contains the **cart icon** widget
  2. **product_list.dart** shows the list of products
  3. **product_tile.dart** shows the individual product item.





With setState (1)

Resource Software Solution

shopping_home.dart

```
No usages
@override
Widget build(BuildConte
  return Scaffold(
    appBar: AppBar(...)
    body: ProductList(
      cart: cart,
      refresh: () {
        setState(() {})
      },
    ),  // ProductList
  );  // Scaffold
}
}
```

product_list.dart

```
2 usages
final Cart cart;
2 usages
final VoidCallback refresh;
1 usage
const ProductList({super.key, requ

No usages
@override
Widget build(BuildContext context)
  return ListView.builder(
    itemCount: 20,
    itemBuilder: ( BuildContext co
      child: ProductTile(
        itemNo: index,
        cart: cart,
        refresh: refresh,
      ),  // ProductTile
    ));  // Card, ListView.
  }
}
```

product_tile.dart

```
trailing: IconButton(
  key: Key( value: 'icon_$itemNo'),
  icon: cart.items.contains( element: itemNo)
      ? const Icon( icon: Icons.shopping_cart)
      : const Icon( icon: Icons.shopping_cart_outlined),
  onPressed: () {
    !cart.items.contains( element: itemNo)
        ? cart.add(itemNo)
        : cart.remove(itemNo);
    debugPrint('cart item count ${cart.items.length}');
    refresh();
    ScaffoldMessenger.of(context).showSnackBar(
      snackBar: SnackBar(
        content: Text( data: cart.items.contains( element: itemNo)
            ? 'Added to cart.'
            : 'Removed from cart.'),  // Text
        duration: const Duration(seconds: 1),
      ),  // SnackBar
    );
  },
),  // IconButton
```

1.**home.dart** 2.**product_list.dart** 3.**product_tile.dart**

With setState (2)

- The problem: **data has to be passed from all widgets** to the bottom widget

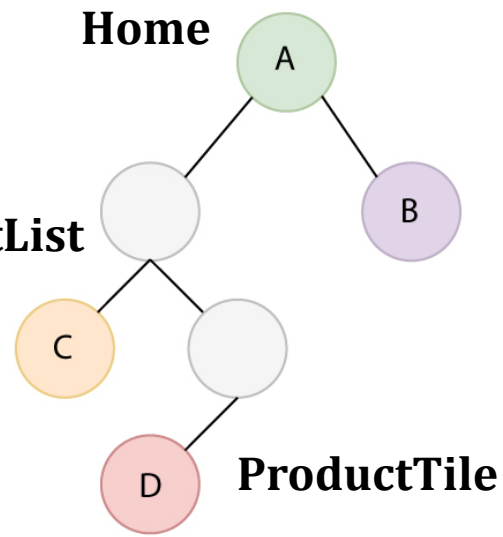
shopping_home.dart

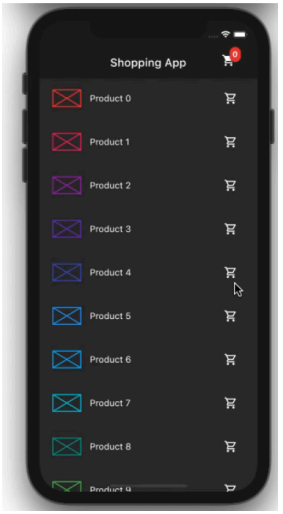
```
No usages
@override
Widget build(BuildCont
  return Scaffold(
    appBar: AppBar(...
    body: ProductList(
      cart: cart,
      refresh: () {
        setState(() {})
      },
    ),  // ProductList
  );  // Scaffold
}
}
```

product_list.dart

```
2 usages
final Cart cart;
2 usages
final VoidCallback refresh;
1 usage
const ProductList({super.key, required this.cart,

No usages
@override
Widget build(BuildContext context) {
  return ListView.builder(
    itemCount: 20,
    itemBuilder: (BuildContext context, int index)
      child: ProductTile(
        itemNo: index,
        cart: cart,
        refresh: refresh,
      ),  // ProductTile
    ));  // Card, ListView.builder
}
}
```

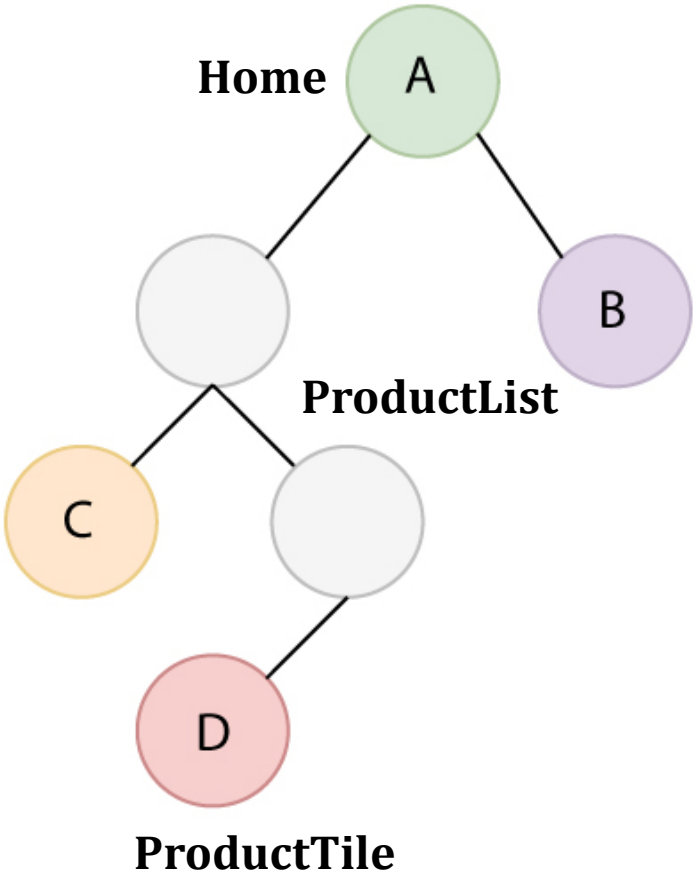
```
class ProductTile extends StatelessWidget {
  10 usages
  final int itemNo;
  7 usages
  final Cart cart;
  2 usages
  final VoidCallback refresh;

  1 usage
  const ProductTile({super.key, required this.itemNo, required this.cart,
      required this.refresh});
```





With setState (3)

- The problems:
  1. To send that data to Widget D, we have to add the data we want to **send down to each constructor of each parent of D** and when more children that need that data are added to the tree, those widgets and their parents would need to also **receive that data in the constructor.** Still, we send data to widgets that don't really need it (Grey nodes).
  2. How can we easily alter the widget C whenever an update in B is made?



With BloC (1)

Resource Software Solution

shopping_home.dart

```
No usages
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(...),  // AppBar
    body: ProductList(
      cart: cart,
      refresh: () {
        setState(() {});
      },
    ),  // ProductList
  );  // Scaffold
}
}
```

product_list.dart

```
2 usages
final Cart cart;
2 usages
final VoidCallback refresh;
1 usage
const ProductList({super.key, required this.cart, required this.refresh});

No usages
@override
Widget build(BuildContext context) {
  return ListView.builder(
    itemCount: 20,
    itemBuilder: (BuildContext context, int index) => Card(
      child: ProductTile(
        itemNo: index,
        cart: cart,
        refresh: refresh,
      ),  // ProductTile
    ));  // Card, ListView.builder
  }
}
```

```
class ProductTile extends StatelessWidget {
  10 usages
  final int itemNo;
  7 usages
  final Cart cart;
  2 usages
  final VoidCallback refresh;

  1 usage
  const ProductTile({super.key, required this.itemNo, required this.cart,
      required this.refresh});
```

Bloc

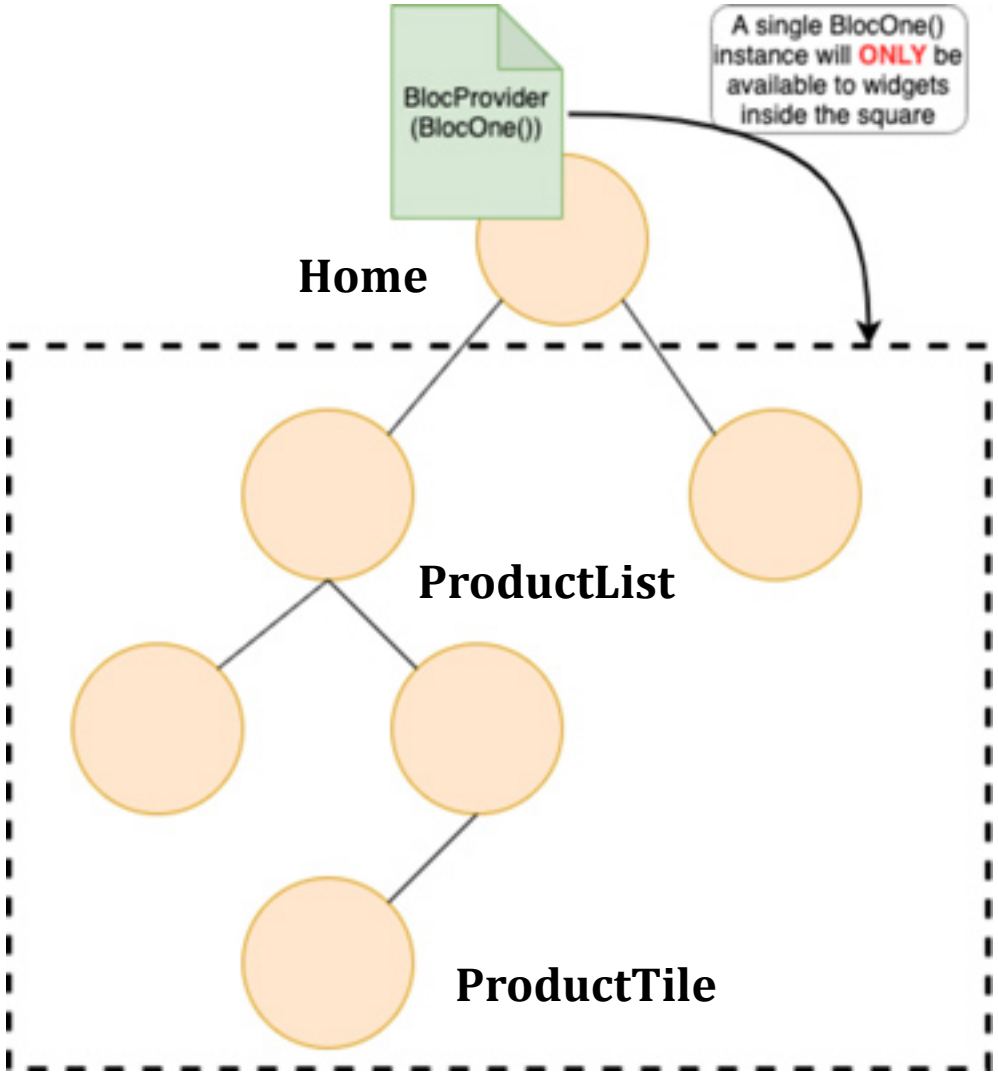
```
Widget build(BuildContext context) {
  return BlocProvider(
      create: (_) => CartBloc(),
      child: Scaffold(
        appBar: AppBar(...),  // AppBar
        body: const ProductList(),
      ));  // Scaffold, BlocProvider
}
```

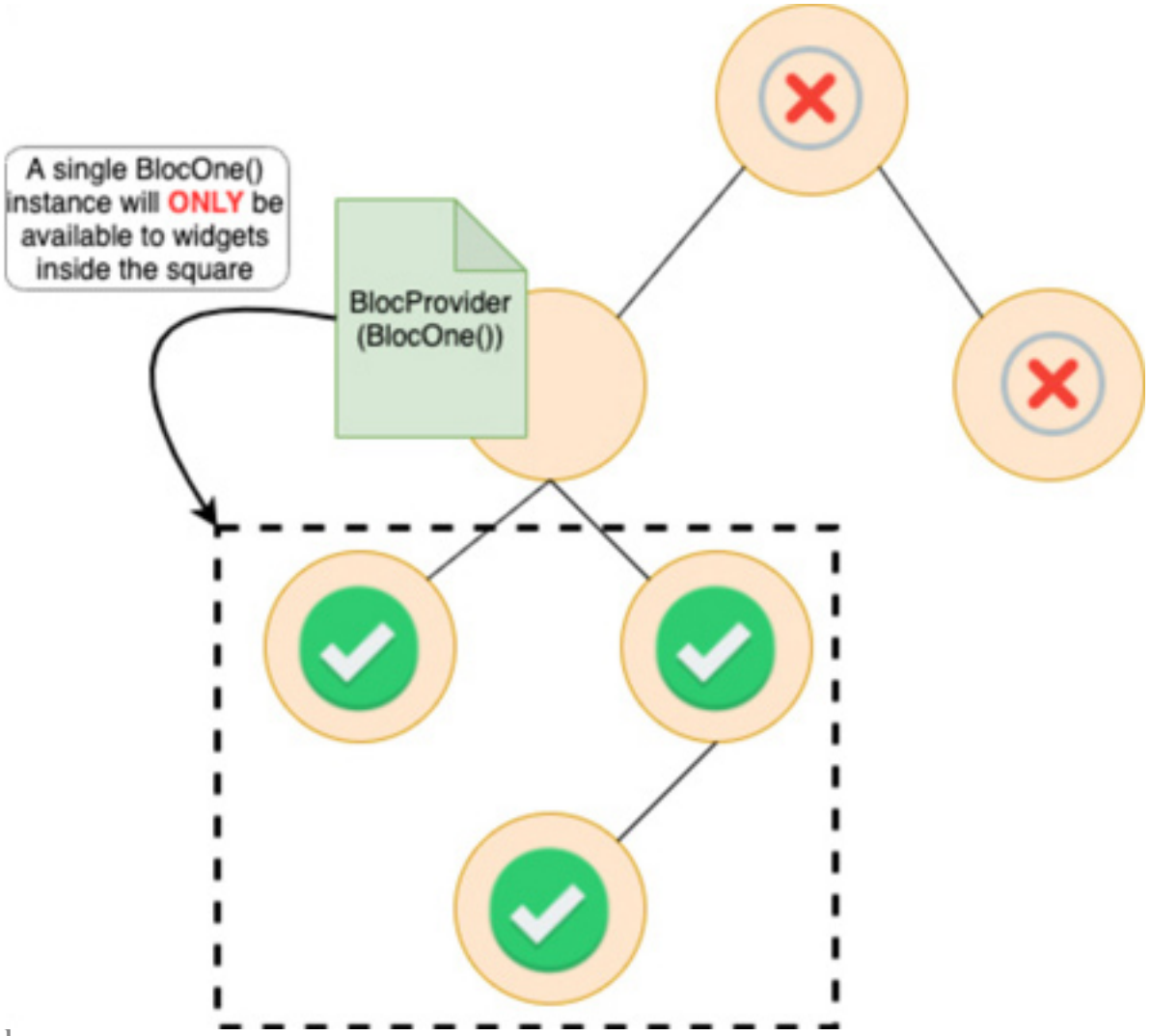
```
Widget build(BuildContext context) {
  return BlocBuilder<CartBloc, CartState>(builder: (_, __) {
    //List<int> cart = cartState.cartItem;
    return ListView.builder(
        itemCount: 20,
        itemBuilder: (context, index) => ProductTile(itemNo: index));
  });  // BlocBuilder
}
```

```
var cartBloc = BlocProvider.of<CartBloc>(context);
var cart = cartBloc.items;
```

With BloC (2)

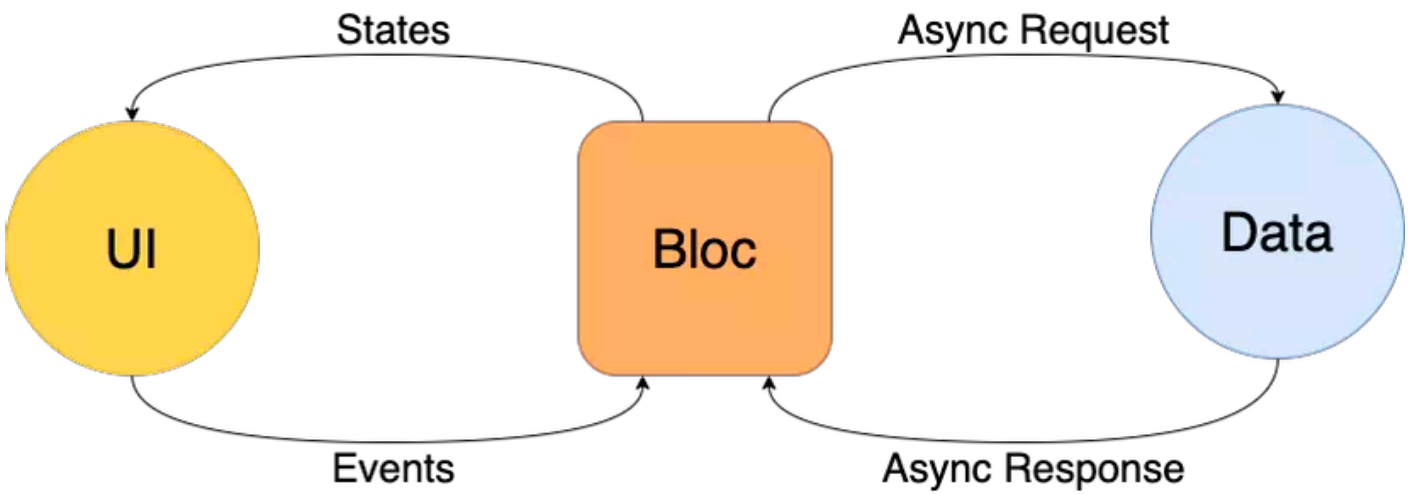
- BlocProvider is a Flutter Widget that builds and provides a Bloc to all its



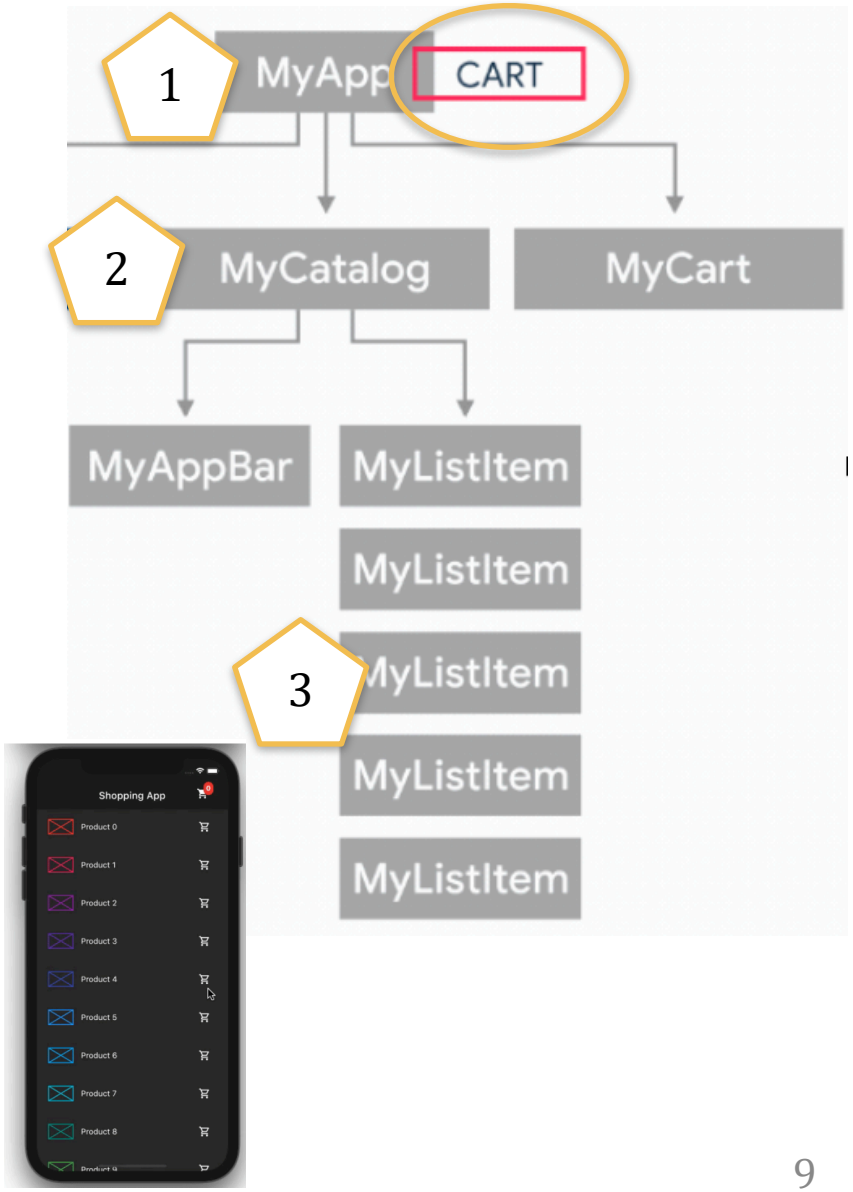


BloC (1)

- BloC stands for **B**usiness **L**ogic **C**omponent

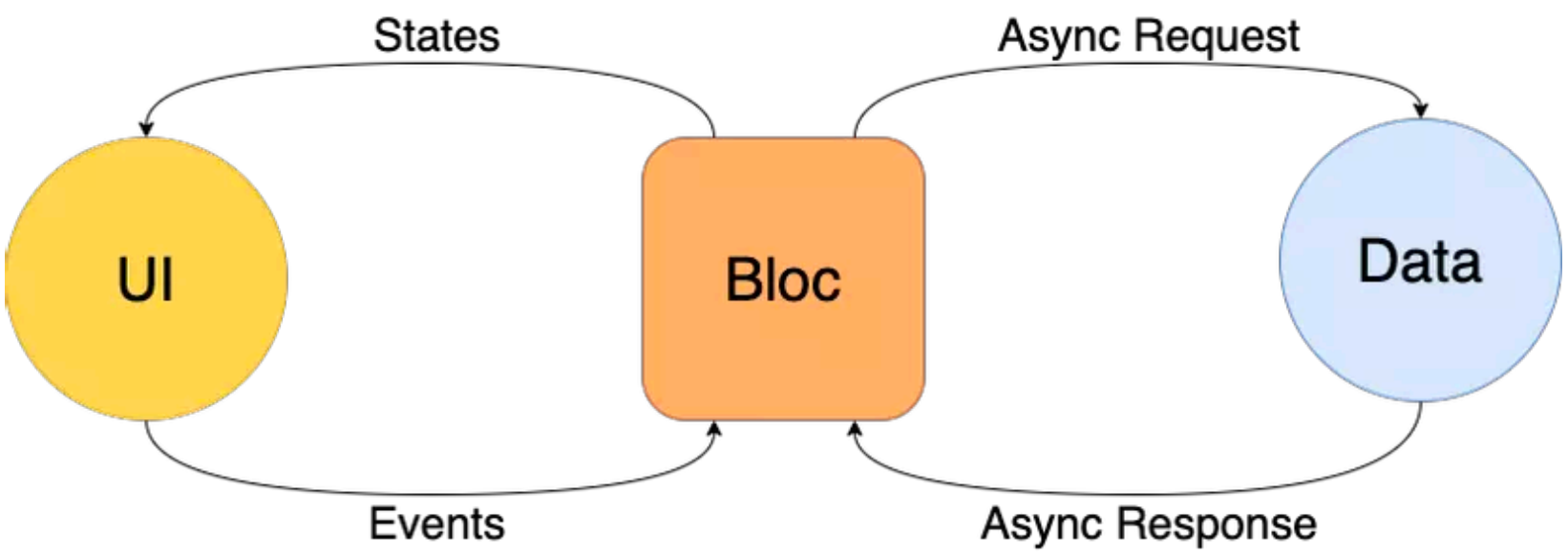


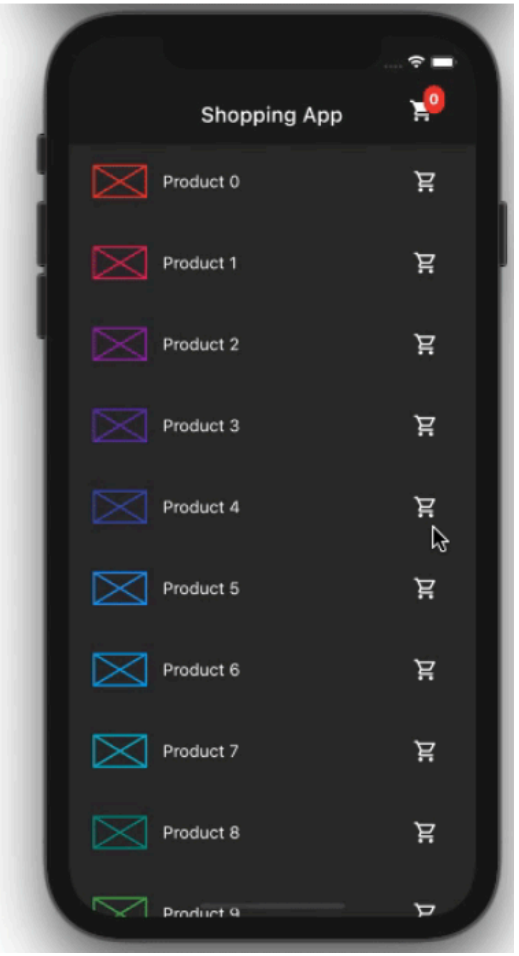
- This library is to make it easy to separate ***presentation*** from ***business logic*** *(get data, update data, ...)* facilitating **testability** and **reusability**.



BloC (2)

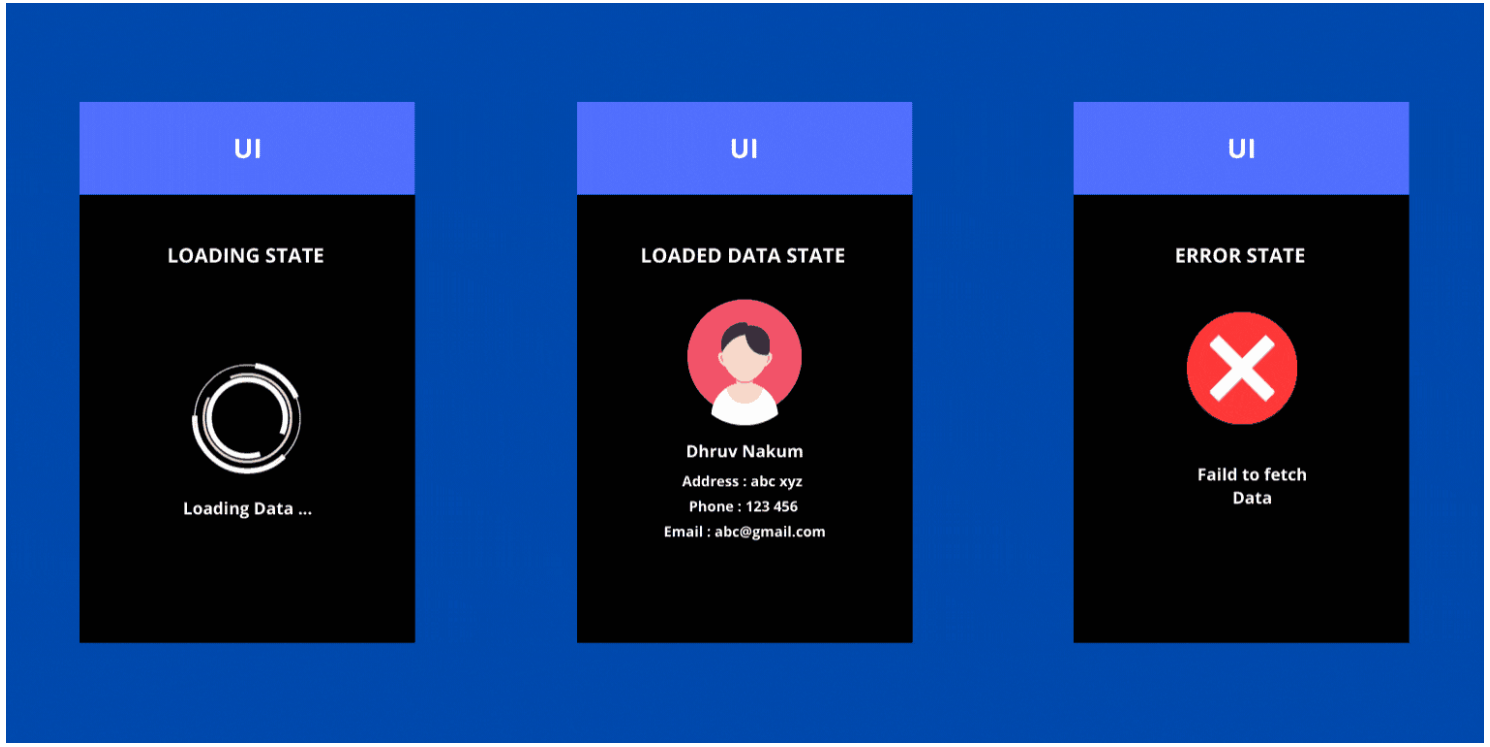
- **Event** is nothing but different **actions** (button click, submit, etc) **triggered by the user from UI**. It contains information about the action and gives it to the Bloc to handle.
- The UI will update according to the **State** it receives from the Bloc.





BloC (3)

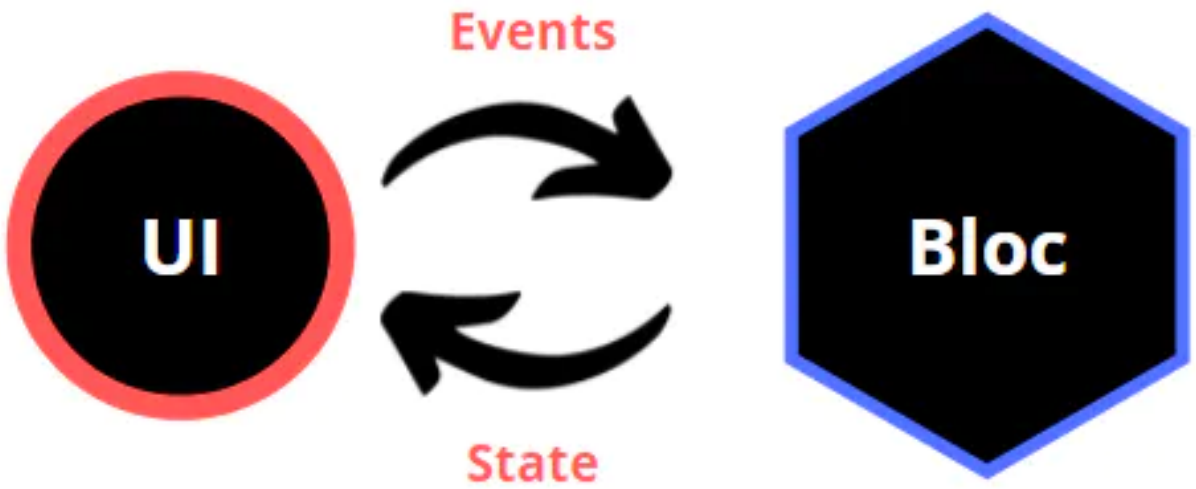
- For example, there could be different kinds of states
  - LoadingState - Will Show Progress Indicator (**reading data**)
  - LoadedState - Will Show Actual widget with data (**show data**)
  - ErrorState - Will show an error that something went wrong.
  - Etc



BloC (4)

- Installation: add the flutter_bloc package to our pubspec.yaml as a dependency.

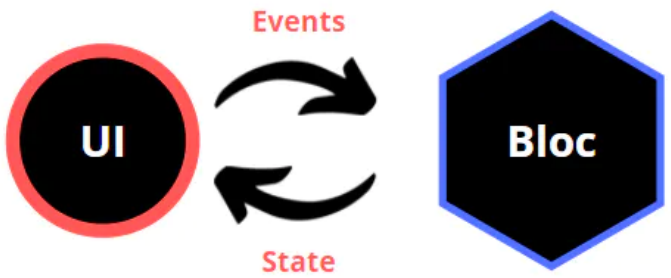
```
flutter_bloc: ^8.0.0
```



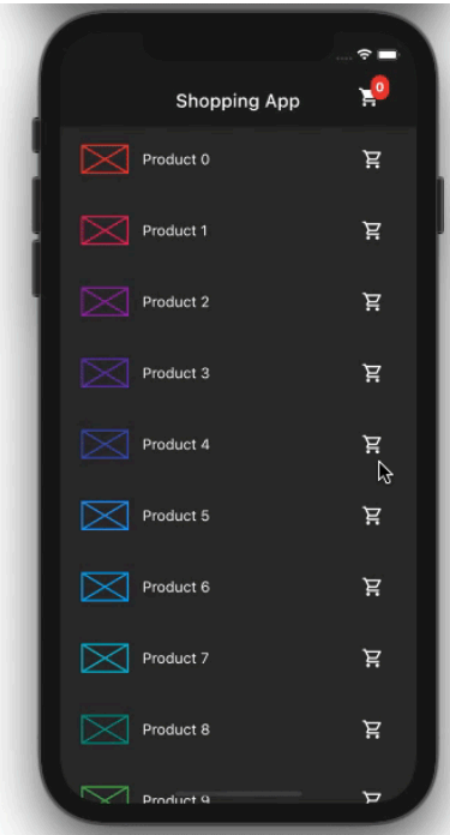
BloC (5)

- **Creating a Bloc:** We must also define the event and state. Events are the input to a Bloc.

```
class CartBloc extends Bloc<CartEvent, CartState> {
  final List<int> _cart = [];

  CartBloc() : super(CartInitialState(cart: [])) {
    on<CartEvent>((event, emit) {
      if (event is AddProductEvent) {
        _cart.add(event.productIndex);
        emit(ProductAddedState(cart: _cart));
      } else if (event is RemoveProductEvent) {
        _cart.remove(event.productIndex);
        emit(ProductRemovedState(cart: _cart));
      }
    });
  }

  List<int> get cart => _cart;
}
```



State changes

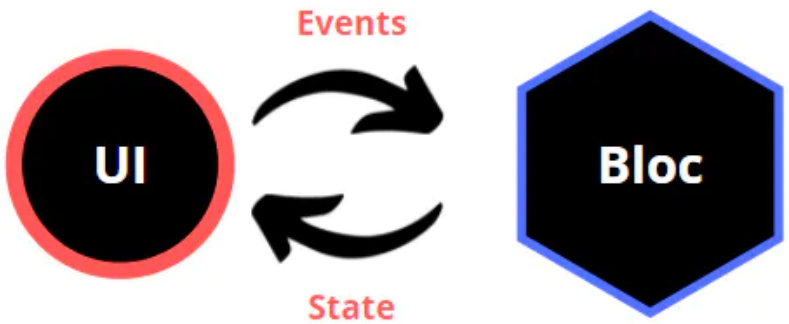


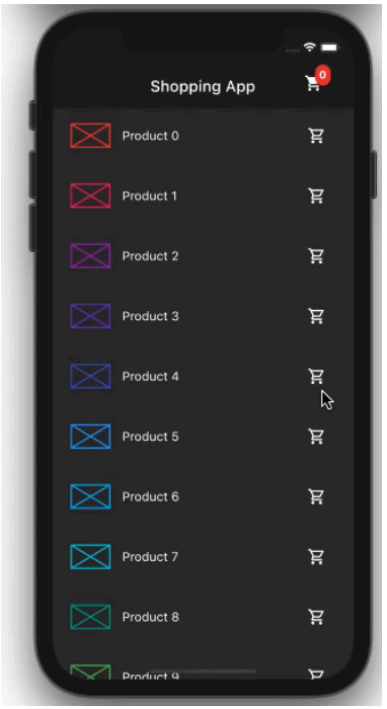
The UI will update according to the **State** it receives from the Bloc.

BloC (6)

- Creating a Bloc: **Events** are the input to a Bloc.

```
abstract class CartEvent {
  const CartEvent();
}

class AddProductEvent extends CartEvent {
  final int productIndex;
  const AddProductEvent(this.productIndex);
}

class RemoveProductEvent extends CartEvent {
  final int productIndex;
  const RemoveProductEvent(this.productIndex);
}
```

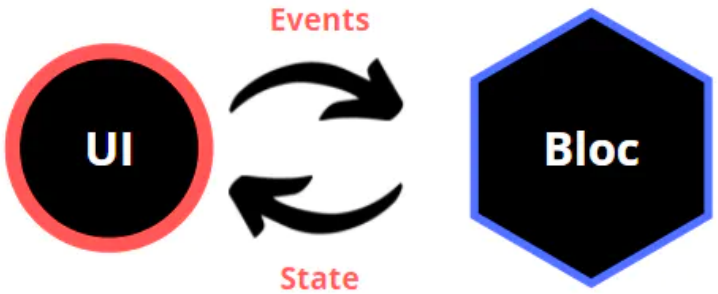




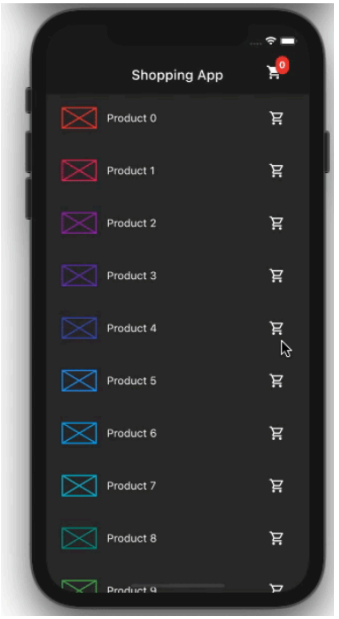
BloC (7)

- Creating a Bloc: The UI will update according to the **State** it receives from the Bloc

```
abstract class CartState {
  final List<int> cart;

  const CartState({required this.cart});
}

class CartInitialState extends CartState {
  CartInitialState({required super.cart});
}

class ProductAddedState extends CartState {
  const ProductAddedState({required super.cart});
}

class ProductRemovedState extends CartState {
  const ProductRemovedState({required super.cart});
}
```

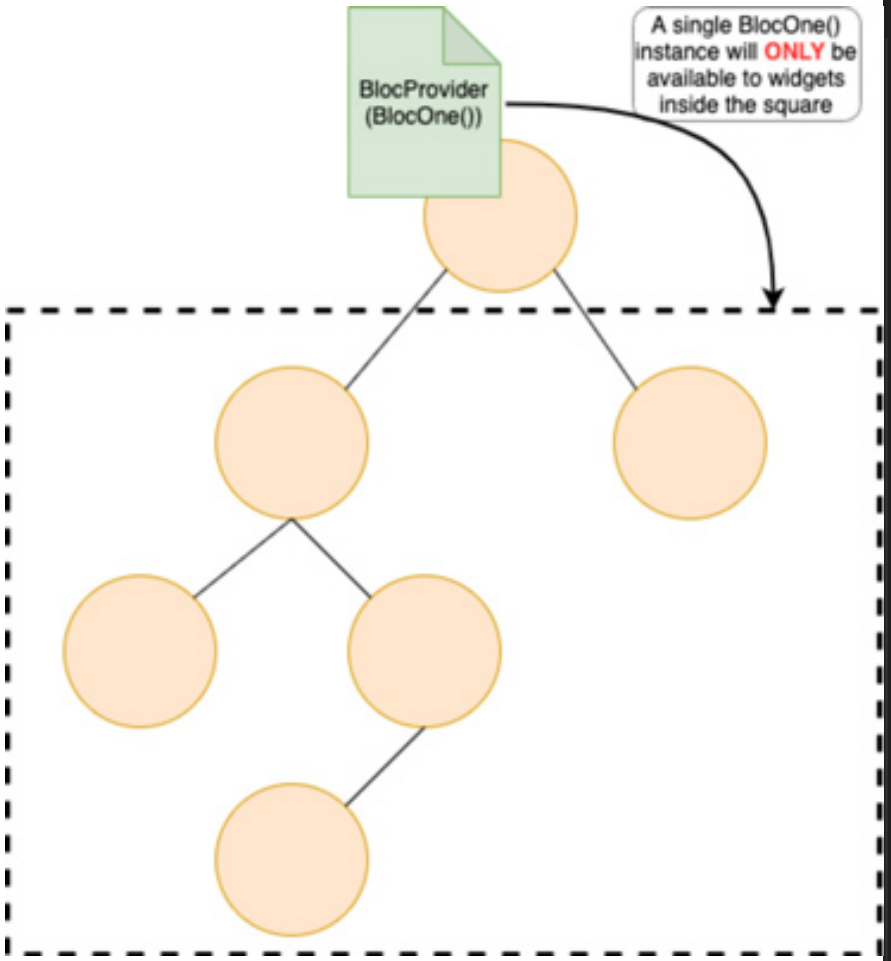



```
BlocBuilder<CartBloc, CartState>(builder: (_, state) {
  List<int> cart = state.cart;
```



BloC (8)

- **Using a Bloc**



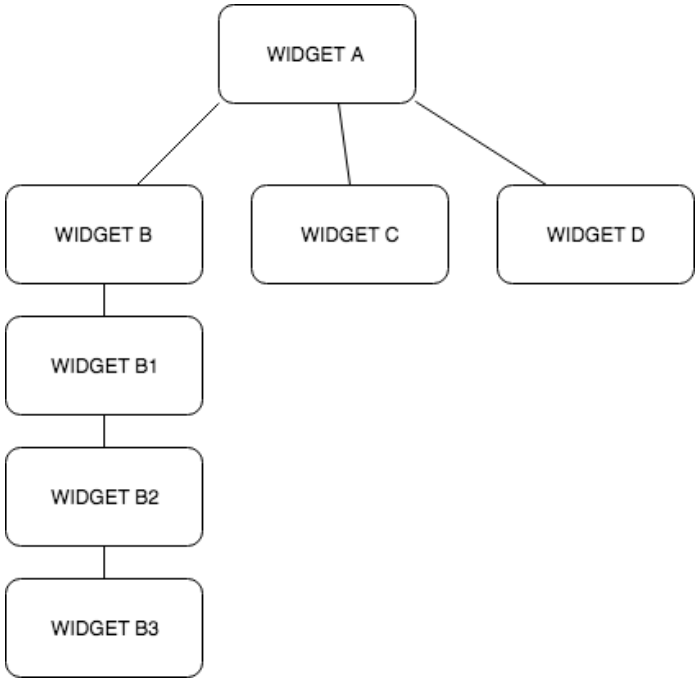

```
return BlocProvider(
    create: (_) => CartBloc(),
    child: Scaffold(
      appBar: AppBar(
        title: const Text('Shopping App'),
        actions: <Widget>[
          Stack(
            children: [
              Padding(...),  // Padding
              BlocBuilder<CartBloc, CartState>(builder: (_, state) {...}),
            ],
          ),  // Stack
        ],  // <Widget>[]
      ),  // AppBar
      body: const ProductList(),
    ));  // Scaffold, BlocProvider
```

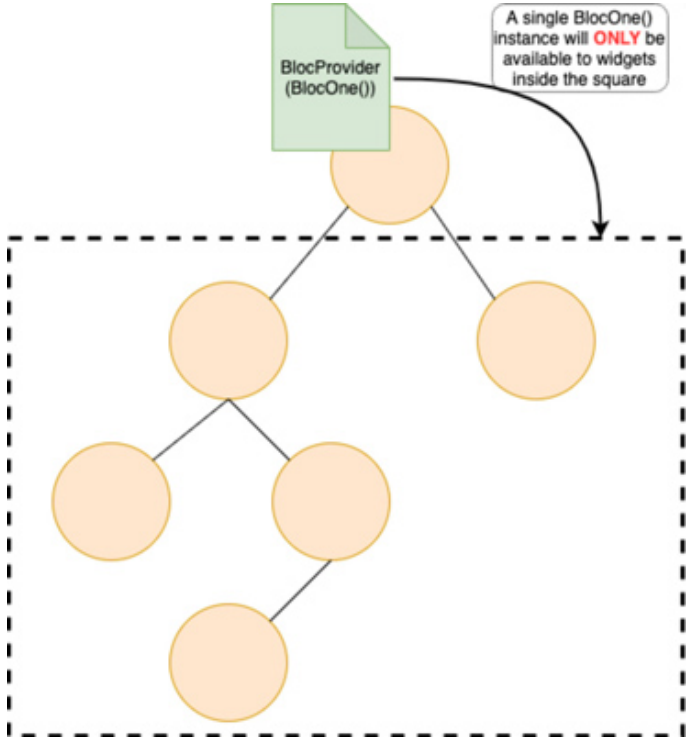
BlocProvider widget provides a bloc to its children (i.e Widgets)

BlocBuilder is a widget that helps **Re-building the UI** based on **State changes**

Summary (1)

- The setState(){} is used to manage local state in the same StatefulWidget and it's child .
- BLoC pattern is used to manage global state.
- if you want to pass data from B2 to A?
  - Using StatefulWidget you should pass data from B2 to B1 to B to A.
  - Using BLoC pattern to manage global state you pass data from B2 to A directly.





Summary (2)

- **BlocBuilder** is a widget that helps **Re-building the UI** based on **State changes**

```
BlocBuilder<CartBloc, CartState>(builder: (_, state) {
  List<int> cart = state.cart;

  Widget animatedText =
      _buildDefaultAnimatedText(cart.length);

  if (state is ProductAdded) {
    animatedText =
        _buildProductAddedAnimatedText(cart.length);
  } else if (state is ProductRemoved) {
    animatedText =
        _buildProductRemovedAnimatedText(cart.length);
  }

  return Positioned(
    left: 30,
    child: Container(
      padding: const EdgeInsets.all(5),
      decoration: BoxDecoration(
          borderRadius: BorderRadius.circular(10),
          color: Colors.red),  // BoxDecoration
      child: animatedText,
    ),  // Container
  );  // Positioned
}),  // BlocBuilder
```

Home page

```
return BlocBuilder<CartBloc, CartState>(builder: (_, state) {
  return ListView.builder(
      itemCount: 20,
      itemBuilder: (context, index) => ProductTile(itemNo: index));
});  // BlocBuilder
```

ProductList page

- Get data from Bloc

```
var cartBloc = BlocProvider.of<CartBloc>(context);
var cart = cartBloc.cart;
```

ProductTile page

*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