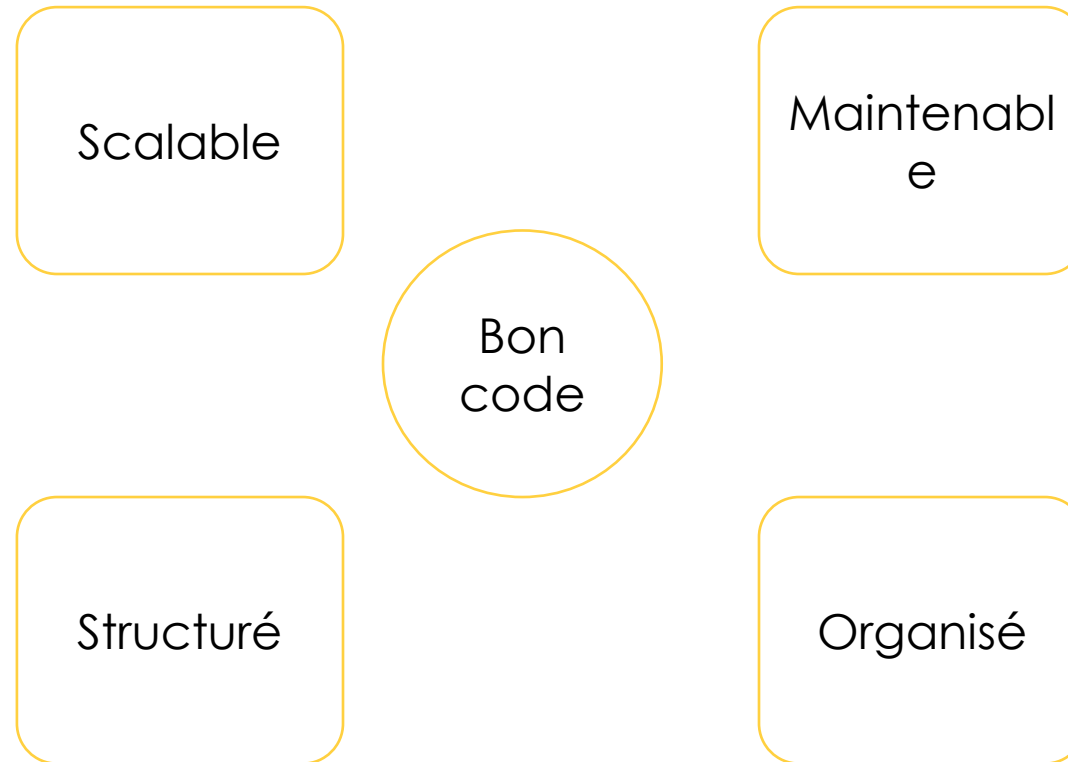
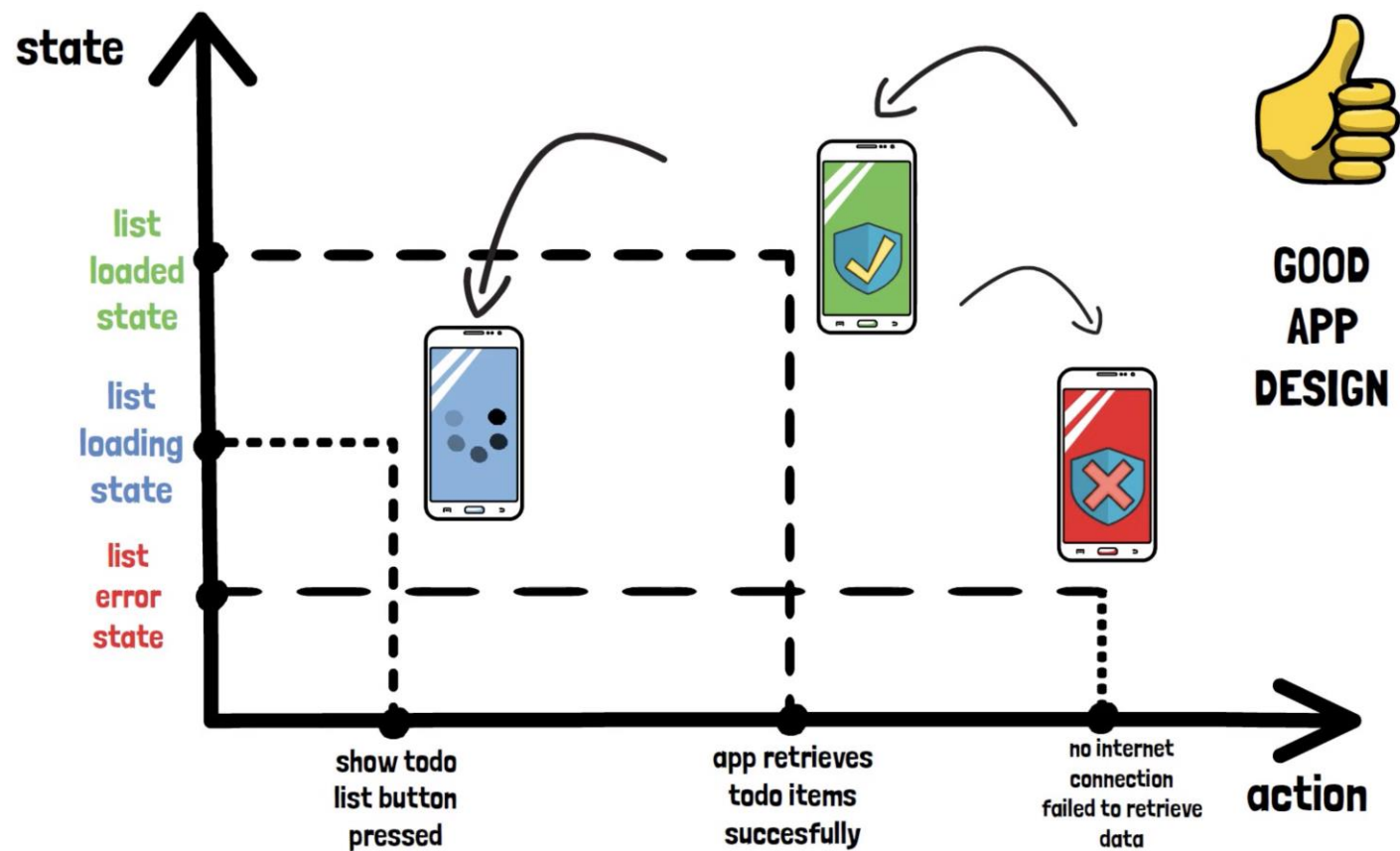


# Flutter – gestion du state : pourquoi

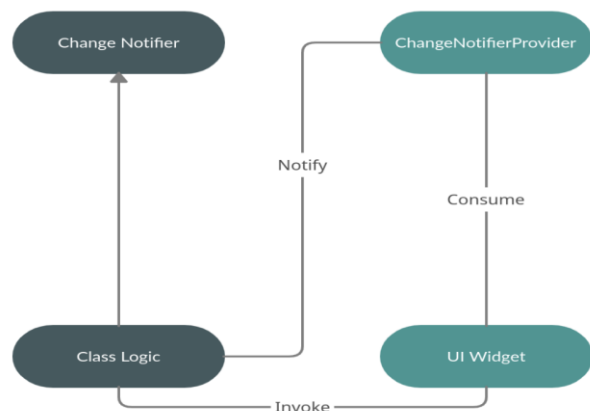


# Flutter – gestion du state : pourquoi

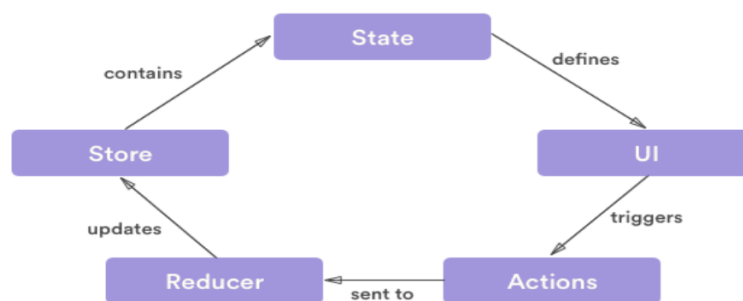


# Flutter – gestion du state : plusieurs patterns

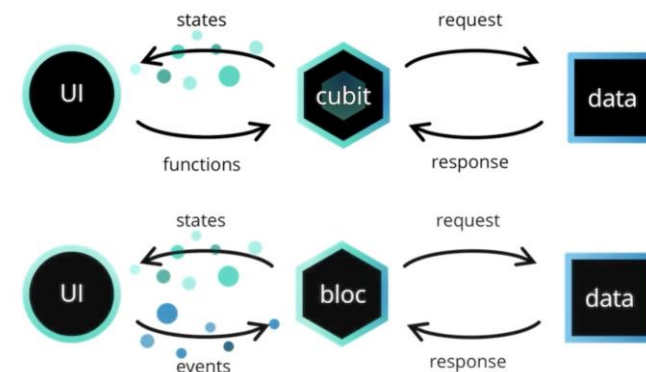
Provider



Redux

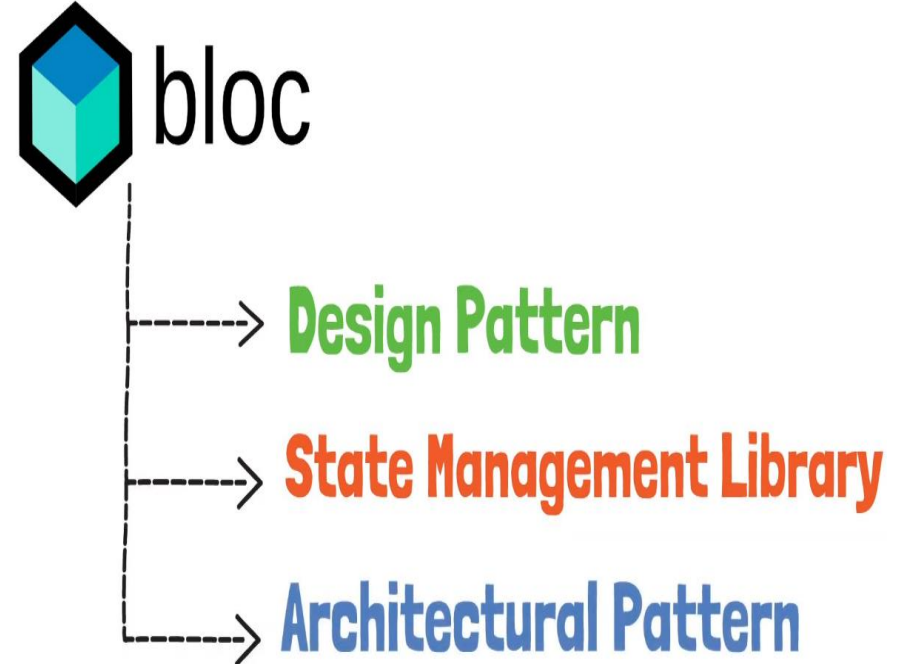


BLoC(Business logic Component)



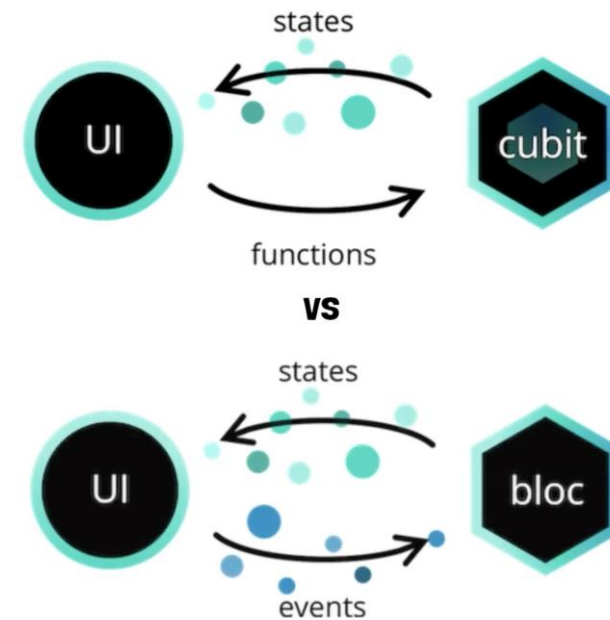
# Flutter – gestion du state : pourquoi utiliser BLoC

1. Il est structurellement organisé et facile à utiliser
2. Recommandé partout dans les petits, moyens et grands projets
3. le code est facilement testable afin que vous puissiez itérer en toute confiance
4. Communauté grandissante qui facilite la courbe d'apprentissage



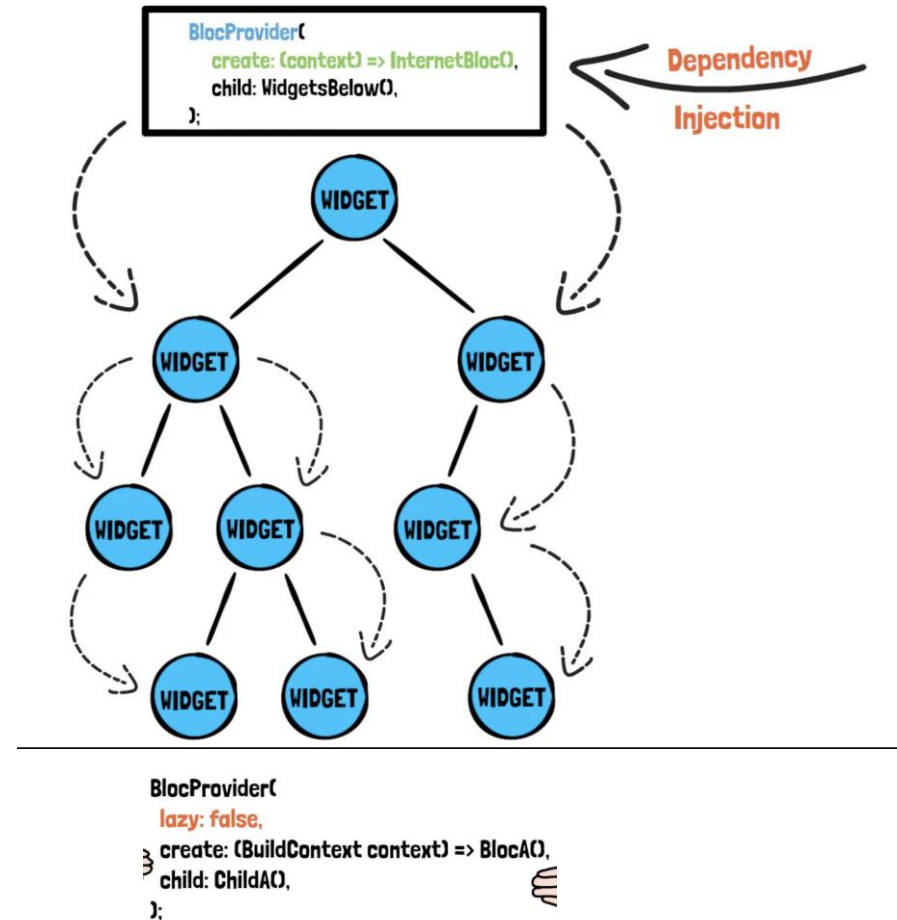
# Flutter – gestion du state : BLoC

1. Quel est le state initial du widget
2. Quelles sont les interactions possibles avec l'application
3. Quelles sont les fonctions à l'intérieur du cubit qui décrivent les différentes actions



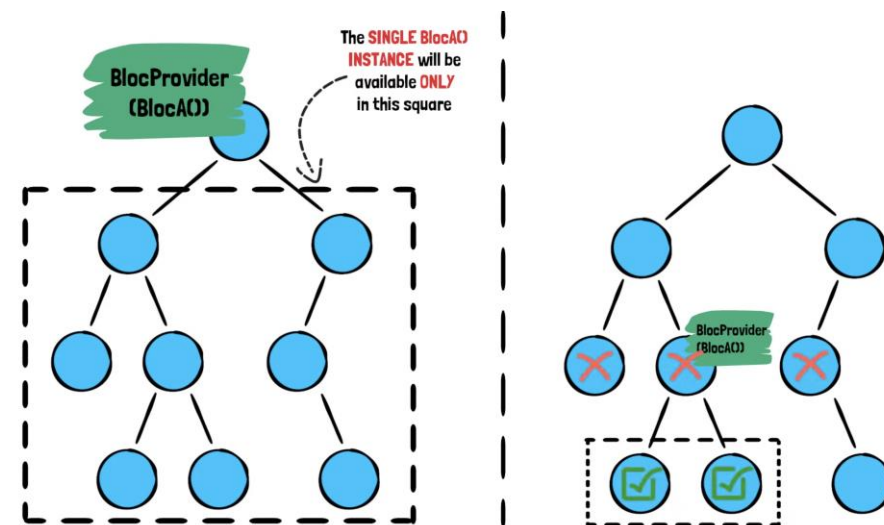
## Flutter – gestion du state : BLoC, BlocProvider, blocBuilder, BlocListner, BlocConsumer, RepositoryProvider, Multi...

1. Est un widget Flutter qui fournit un bloc à ses enfants
2. Il est utilisé comme widget d'injection de dépendance (DI) afin qu'une seule instance d'un bloc puisse être fournie à plusieurs widgets dans un sous-arbre
3. La même instance est partagée entre tous les nœuds du sous arbre
4. Le sous arbre est dépendent du bloc fourni
5. Le bloc est fourni au sous arbre en « Lazy loading »



## Flutter – gestion du state : BLoC, BlocProvider, blocBuilder, BlocListner, BlocConsumer, RepositoryProvider, Multi...

1. Est un widget Flutter qui fournit un bloc à ses enfants
2. Il est utilisé comme widget d'injection de dépendance (DI) afin qu'une seule instance d'un bloc puisse être fournie à plusieurs widgets dans un sous-arbre
3. La même instance est partagée entre tous les nœuds du sous arbre
4. Le sous arbre est dépendent du bloc fourni
5. Le bloc est fourni au sous arbre en « Lazy loading »



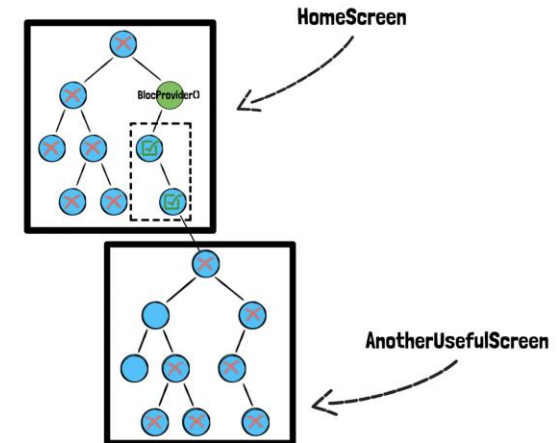
## Flutter – gestion du state : BLoC, BlocProvider, blocBuilder, BlocListner, BlocConsumer, RepositoryProvider, Multi...

1. Est un widget Flutter qui fournit un bloc à ses enfants
2. Il est utilisé comme widget d'injection de dépendance (DI) afin qu'une seule instance d'un bloc puisse être fournie à plusieurs widgets dans un sous-arbre
3. La même instance est partagée entre tous les nœuds du sous arbre
4. Le sous arbre est dépendent du bloc fourni
5. Le bloc est fourni au sous arbre en « Lazy loading »

**BlocProvider.of<BlocA>(context);**

**or**

**context.bloc<BlocA>();**





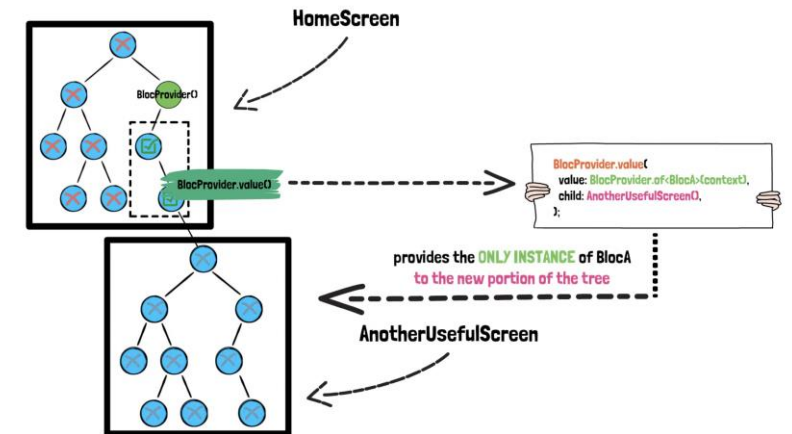
## Flutter – gestion du state : BLoC, **BlocProvider**, blocBuilder, BlocListner, BlocConsumer, RepositoryProvider, Multi...

1. Est un widget Flutter qui fournit un bloc à ses enfants
2. Il est utilisé comme widget d'injection de dépendance (DI) afin qu'une seule instance d'un bloc puisse être fournie à plusieurs widgets dans un sous-arbre
3. La même instance est partagée entre tous les nœuds du sous arbre
4. Le sous arbre est dépendent du bloc fourni
5. Le bloc est fourni au sous arbre en « Lazy loading »

**BlocProvider.of<BlocA>(context);**

or

**context.bloc<BlocA>();**

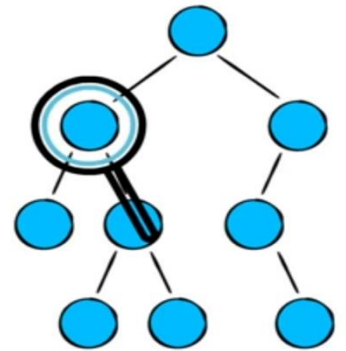


## Flutter – gestion du state : BLoC, BlocProvider, **blocBuilder**, BlocListner, BlocConsumer, RepositoryProvider, Multi...

1. Est un widget Flutter qui nécessite un bloc et une fonction builder
2. Il gère la construction du widget en réponse aux nouveaux états
3. Il est très similaire à **StreamBuilder** mais possède une API plus simple pour réduire la quantité de code nécessaire
4. La fonction **builder** sera potentiellement appelée plusieurs fois et devrait être une fonction pure qui renvoie un widget en réponse à l'état.

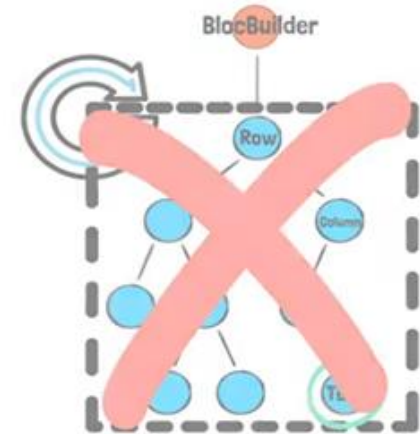
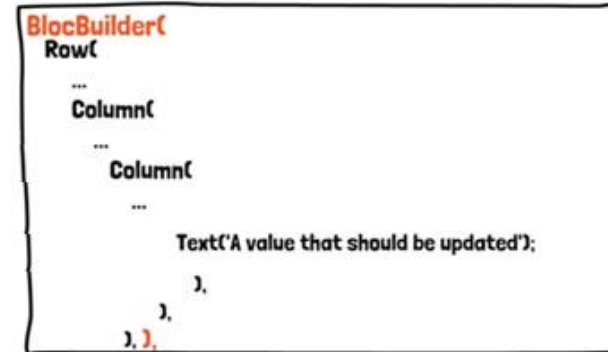
```
BlocBuilder<BlocA, BlocAState>{
  //cubit: BlocProvider.of<BlocA>(context)
  builder: (context, state) {
    // return widget here based on BlocA's state
  }
  buildWhen: (previousState, state) {
    // return true/false to determine whether or not
    // to rebuild the widget with state
  },
}
```

example: if(previousState.value < state.value) return false, this translates as "when the previous value was smaller than the current one (aka the increment function was called), then don't rebuild the ui (aka don't call the builder function above)"



## Flutter – gestion du state : BLoC, BlocProvider, **blocBuilder**, BlocListner, BlocConsumer, RepositoryProvider, Multi...

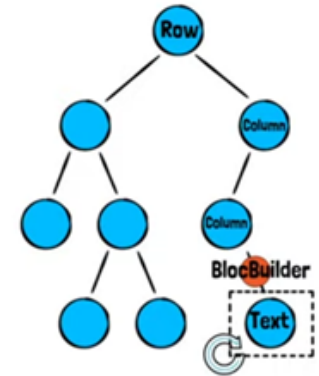
1. Est un widget Flutter qui nécessite un bloc et une fonction builder
2. Il gère la construction du widget en réponse aux nouveaux états
3. Il est très similaire à **StreamBuilder** mais possède une API plus simple pour réduire la quantité de code nécessaire
4. La fonction **builder** sera potentiellement appelée plusieurs fois et devrait être une fonction pure qui renvoie un widget en réponse à l'état.



## Flutter – gestion du state : BLoC, BlocProvider, blocBuilder, BlocListner, BlocConsumer, RepositoryProvider, Multi...

1. Est un widget Flutter qui nécessite un bloc et une fonction builder
2. Il gère la construction du widget en réponse aux nouveaux états
3. Il est très similaire à **StreamBuilder** mais possède une API plus simple pour réduire la quantité de code nécessaire
4. La fonction **builder** sera potentiellement appelée plusieurs fois et devrait être une fonction pure qui renvoie un widget en réponse à l'état.

```
Row(  
  ...  
  Column(  
    ...  
    Column(  
      ...  
      BlocBuilder(  
        Text('A value that should be updated');  
      ),  
    ),  
  ),  
),
```



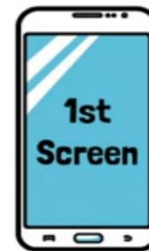
## Flutter – gestion du state : BLoC, BlocProvider, blocBuilder, BlocListener, BlocConsumer, RepositoryProvider, Multi...

1. Est un widget Flutter qui prend un **BlocWidgetListener** et un bloc facultatif et appelle l'écouteur en réponse aux changements d'état dans le bloc.
2. Il doit être utilisé pour les fonctionnalités qui doivent se produire une fois par changement d'état, telles que la navigation, l'affichage d'un snack-bar, l'affichage d'une boîte de dialogue, etc.
3. **listener** n'est appelé qu'une seule fois pour chaque changement d'état (N'incluant PAS l'état initial) contrairement à builder dans BlocBuilder et est une fonction void.
4. Si le paramètre bloc est omis, **BlocListener** effectuera automatiquement une recherche en utilisant **BlocProvider** et le **BuildContext** actuel

**BlocBuilder**

```
BlocBuilder<BlocA, BlocAState>{
  builder: (context, state) {
    Navigator.push(MaterialPageRoute(2NDScreen))
  }
}
```

The builder function CAN BE CALLED MULTIPLE TIMES, as I have previously stated before. (Flutter Engine Quirks)



builder function called 5X

pushing the 2ndScreen into the Navigation Stack 5X

## Flutter – gestion du state : BLoC, BlocProvider, blocBuilder, **BlocListener**, **BlocConsumer**, RepositoryProvider, Multi...

1. Est un widget Flutter qui prend un **BlocWidgetListener** et un bloc facultatif et appelle l'écouteur en réponse aux changements d'état dans le bloc.
2. Il doit être utilisé pour les fonctionnalités qui doivent se produire une fois par changement d'état, telles que la navigation, l'affichage d'un snack-bar, l'affichage d'une boîte de dialogue, etc.
3. **listener** n'est appelé qu'une seule fois pour chaque changement d'état (N'incluant PAS l'état initial) contrairement à builder dans BlocBuilder et est une fonction void.
4. Si le paramètre bloc est omis, **BlocListener** effectuera automatiquement une recherche en utilisant **BlocProvider** et le **BuildContext** actuel

```
BlocListener<BlocA, BlocAState>(  
  listener: (context, state) {  
    // do stuff here based on BlocA's state  
  },  
  child: Container(),  
)
```

this listener function

**IS CALLED ONLY ONCE PER STATE  
(NOT INCLUDING THE INITIAL STATE)**

optional **listenWhen** function for **BlocListener** as the optional **buildWhen** function was for **BlocBuilder**

```
BlocBuilder<BlocA, BlocAState>(  
  builder: (context, state) {  
    // return widget here based on BlocA's state  
  },  
)
```

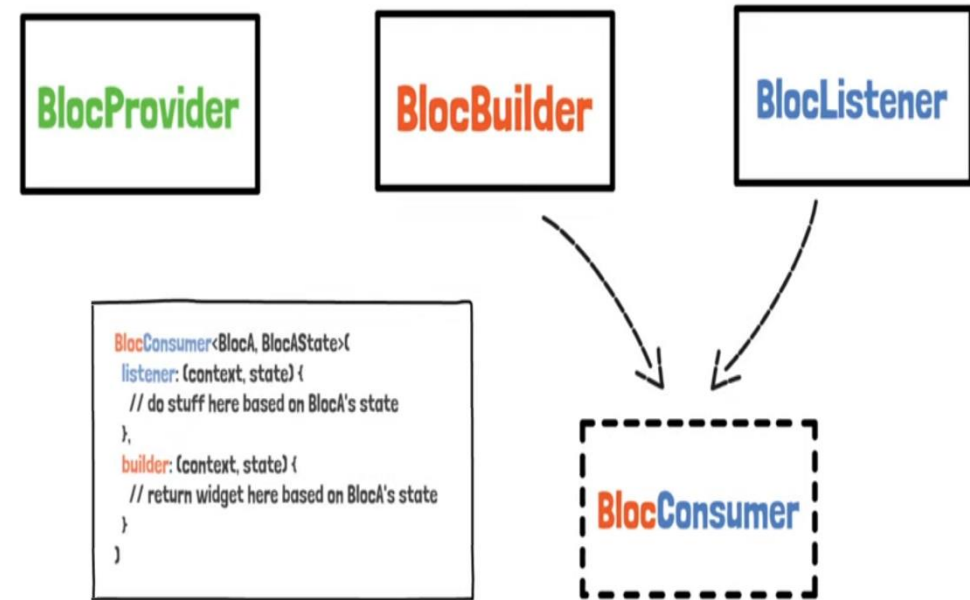


**Rebuilding Widgets**

this builder function  
**MAY BE CALLED MULTIPLE TIMES PER STATE**  
(due to Flutter Engine)

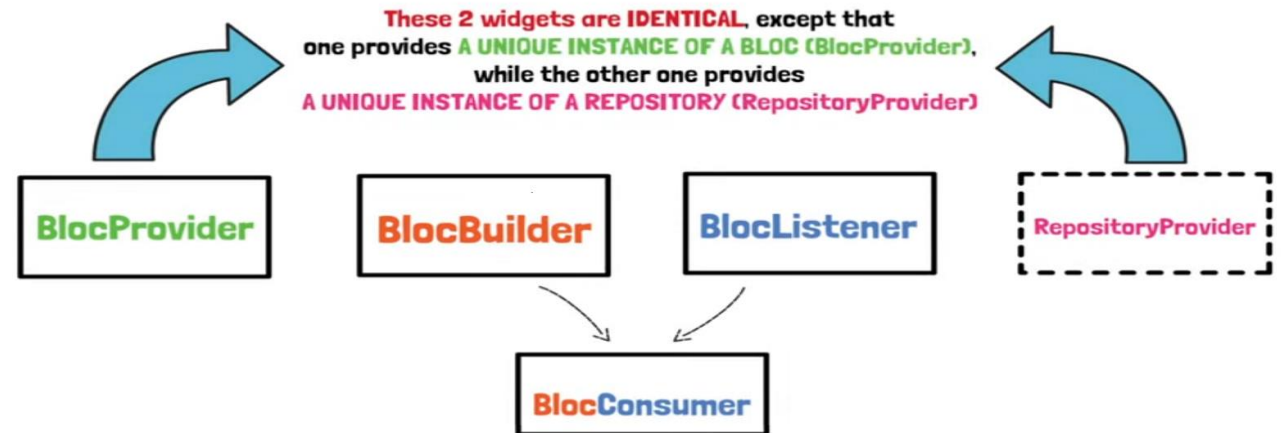


1. Il expose un **builder** et un **listner** afin de réagir aux nouveaux états
2. Il est analogue à un BlocListener et à un BlocBuilder imbriqués mais réduit la quantité de code nécessaire.
3. BlocConsumer ne doit être utilisé que lorsqu'il est nécessaire à la fois de reconstruire l'interface utilisateur et d'exécuter d'autres réactions aux changements d'état dans le bloc



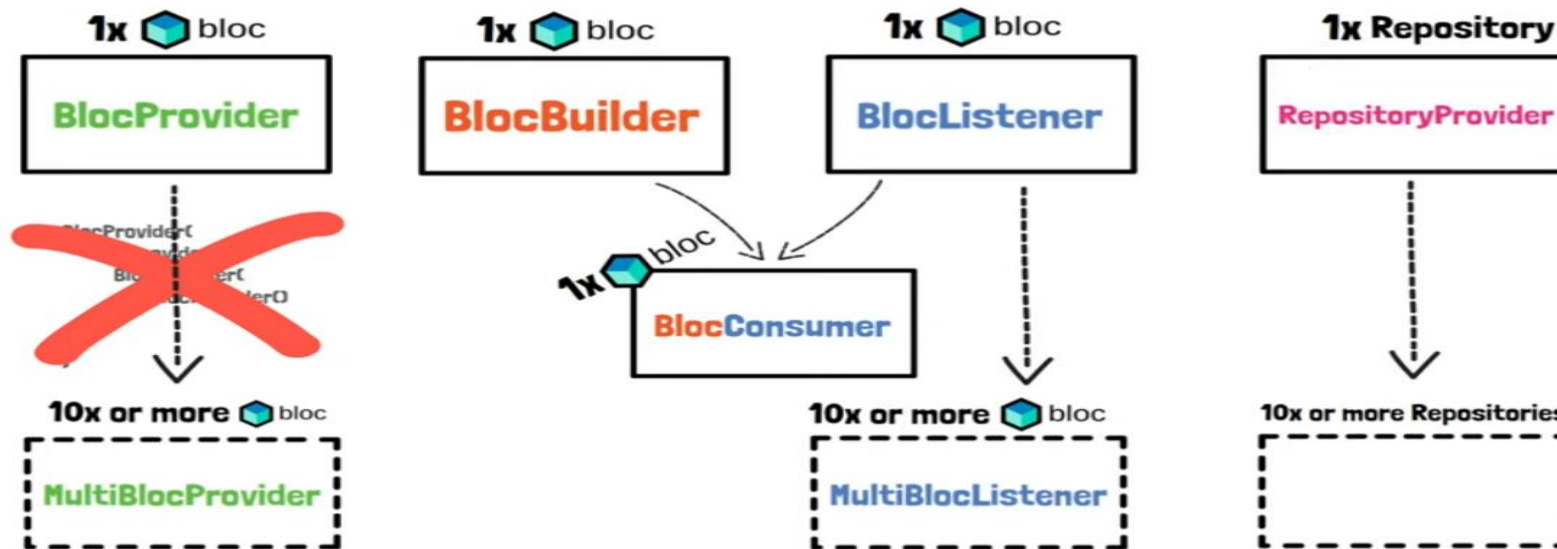
## Flutter – gestion du state : BLoC, BlocProvider, blocBuilder, BlocListner, BlocConsumer, RepositoryProvider, Multi...

1. RepositoryProvider est un widget Flutter qui fournit un repository
2. Il est utilisé comme widget d'injection de dépendance (DI) afin qu'une seule instance de repository puisse être fournie à plusieurs widgets dans un sous-arbre





# Flutter – gestion du state : BLoC, BlocProvider, blocBuilder, BlocListner, BlocConsumer, RepositoryProvider, Multi...



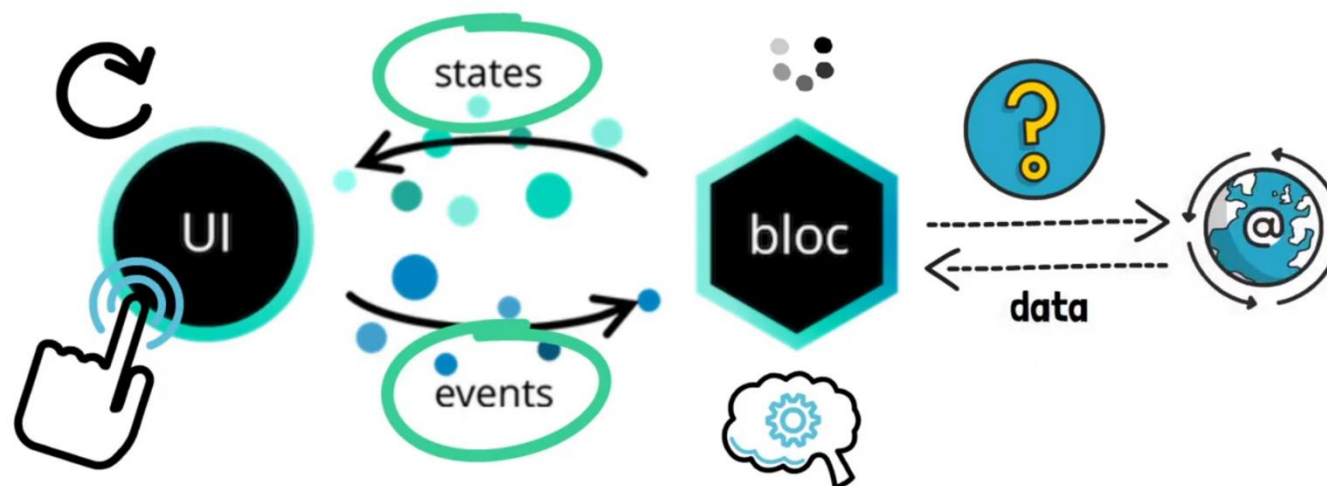
## Flutter – gestion du state : BLoC, BlocProvider, blocBuilder, BlocListner, BlocConsumer, RepositoryProvider, Multi...

```
MultiBlocProvider(
  providers: [
    BlocProvider<BlocA>{
      create: (BuildContext context) => BlocAO,
    },
    BlocProvider<BlocB>{
      create: (BuildContext context) => BlocBO,
    },
    BlocProvider<BlocC>{
      create: (BuildContext context) => BlocCO,
    },
  ],
  child: ChildAO,
)
```

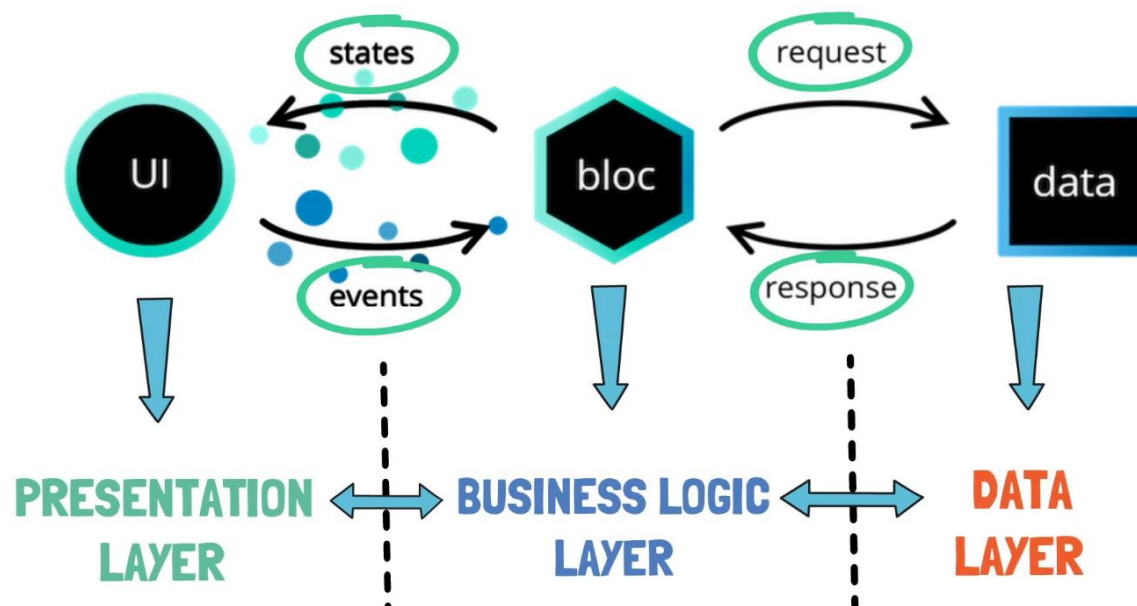
```
MultiBlocListener(
  listeners: [
    BlocListener<BlocA, BlocAState>{
      listener: (context, state) {},
    },
    BlocListener<BlocB, BlocBState>{
      listener: (context, state) {},
    },
    BlocListener<BlocC, BlocCState>{
      listener: (context, state) {},
    },
  ],
  child: ChildAO,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

```
MultiRepositoryProvider(
  providers: [
    RepositoryProvider<RepositoryA>{
      create: (context) => RepositoryAO,
    },
    RepositoryProvider<RepositoryB>{
      create: (context) => RepositoryBO,
    },
    RepositoryProvider<RepositoryC>{
      create: (context) => RepositoryCO,
    },
  ],
  child: ChildAO,
)
```

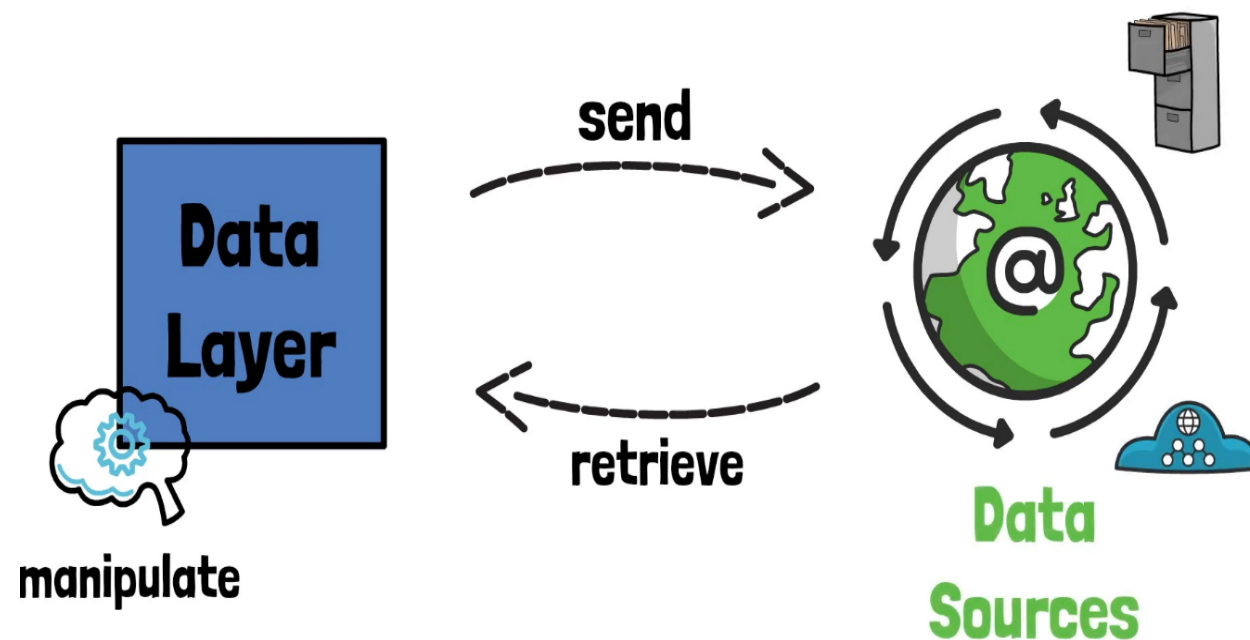
# Flutter – gestion du state : BLoC et Architecture



# Flutter – gestion du state : BLoC et Architecture

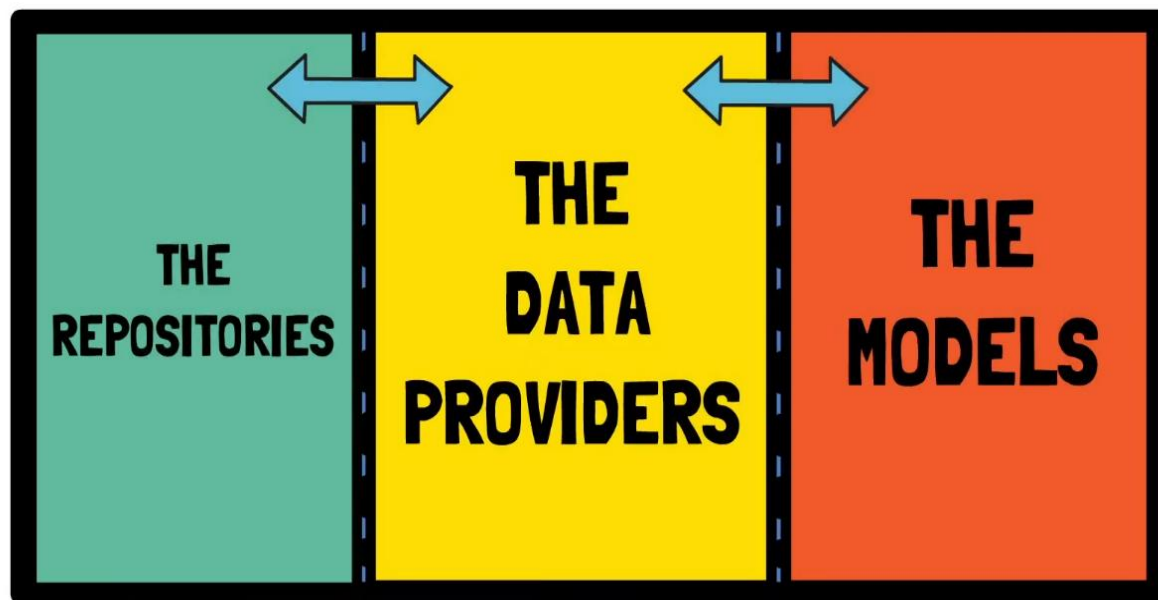


# Flutter – gestion du state : BLoC et Architecture

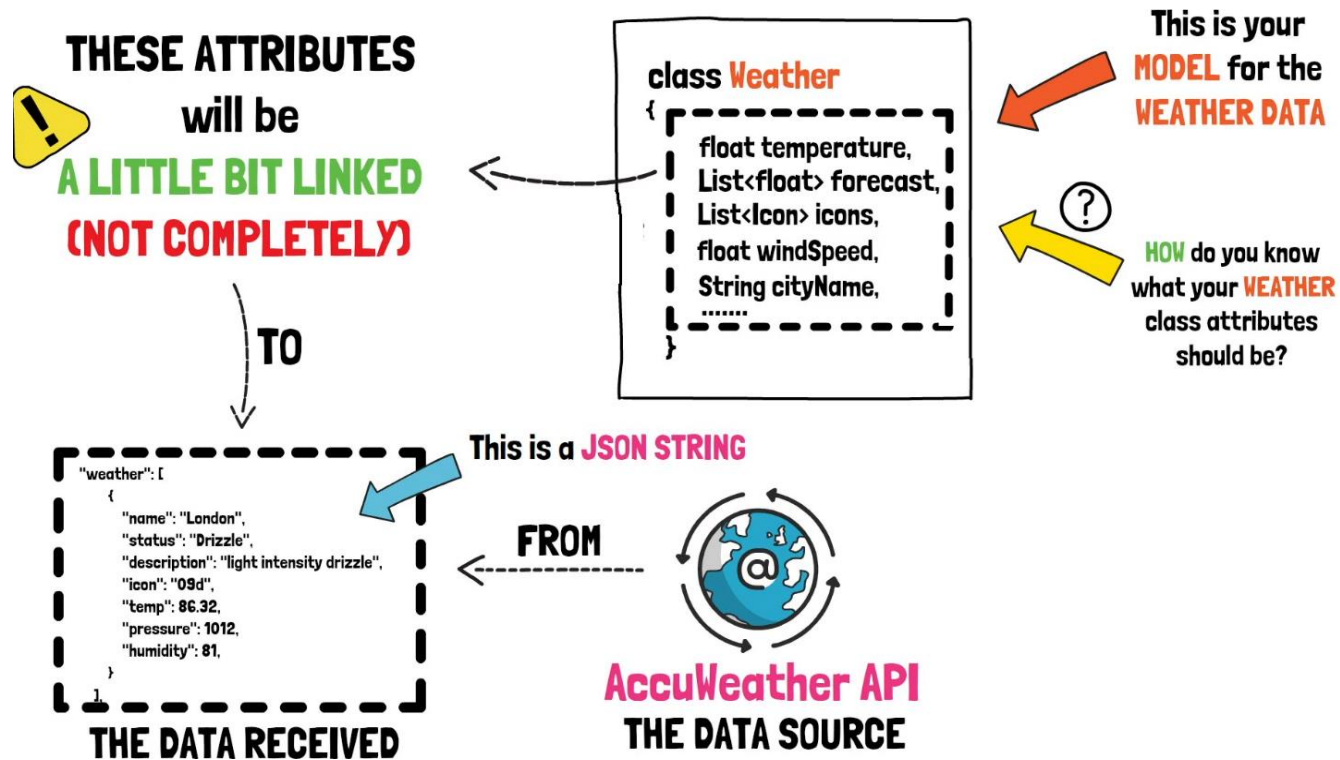


# Flutter – gestion du state : BLoC et Architecture

## THE DATA LAYER

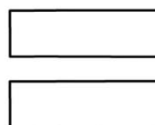


# Flutter – gestion du state : BLoC et Architecture



# Flutter – gestion du state : BLoC et Architecture

**A DATA PROVIDER**



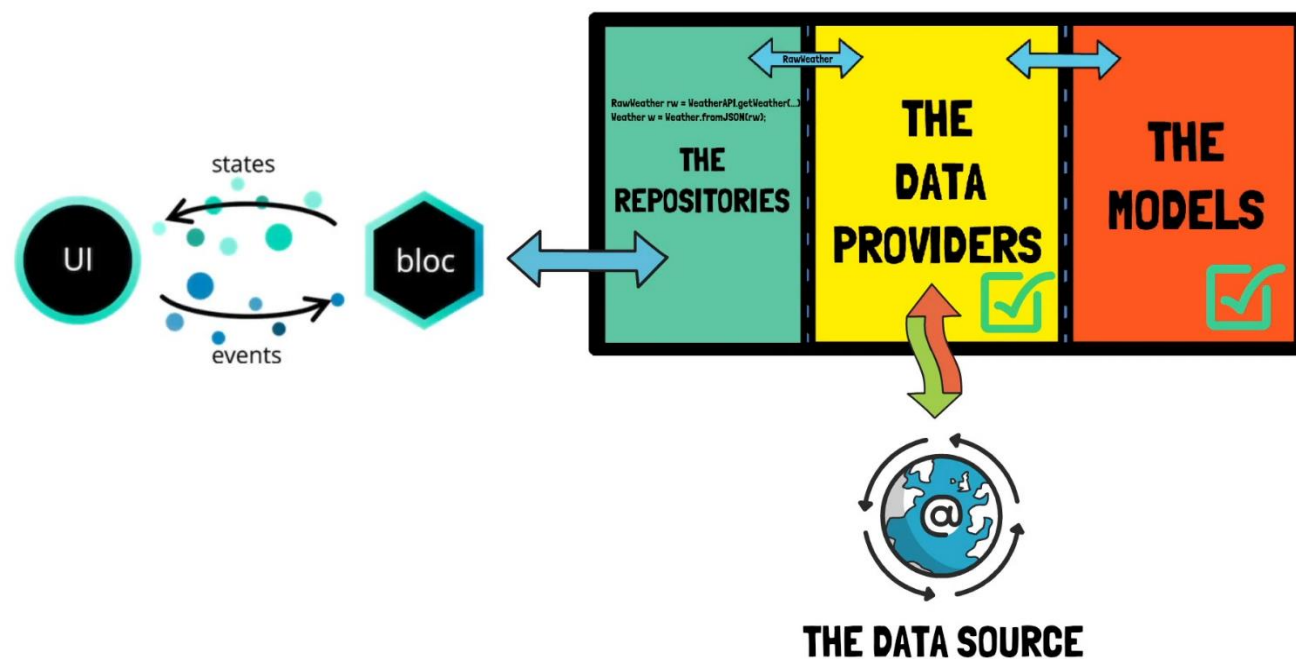
**AN API FOR  
YOUR APP**

```
class WeatherAPI {
  Future<RawWeather> getRawWeather(String city) async {
    RawWeather rawWeather = http.get('www.accuweather.com/${city}');
    return rawWeather;
  }

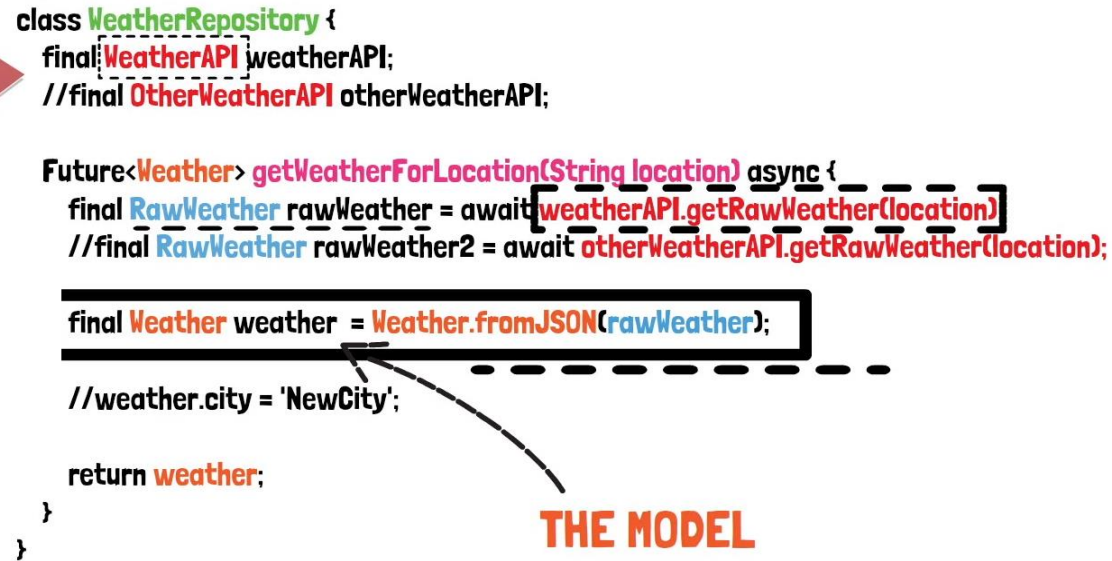
  Future<RawWeather> getRawWeatherForCurrentLocation(Location loc) async {
    RawWeather rawWeather = http.get('www.accuweather.com/${loc.city}');
    return rawWeather;
  }
}
```



# Flutter – gestion du state : BLoC et Architecture



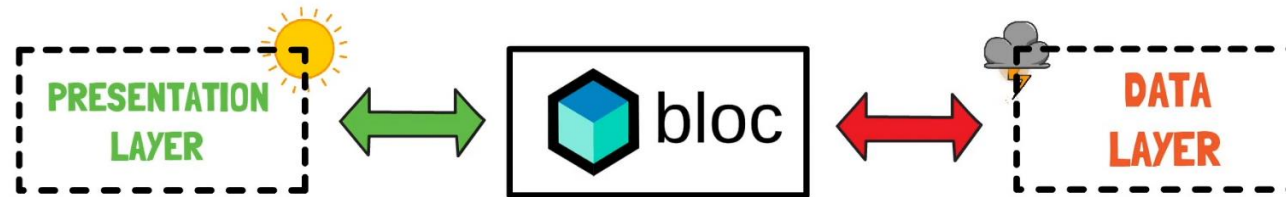
# Flutter – gestion du state : BLoC et Architecture



```
class WeatherRepository {  
  final WeatherAPI weatherAPI;  
  //final OtherWeatherAPI otherWeatherAPI;  
  
  Future<Weather> getWeatherForLocation(String location) async {  
    final RawWeather rawWeather = await weatherAPI.getRawWeather(location);  
    //final RawWeather rawWeather2 = await otherWeatherAPI.getRawWeather(location);  
  
    final Weather weather = Weather.fromJSON(rawWeather);  
  
    //weather.city = 'NewCity';  
  
    return weather;  
  }  
}
```

THE MODEL

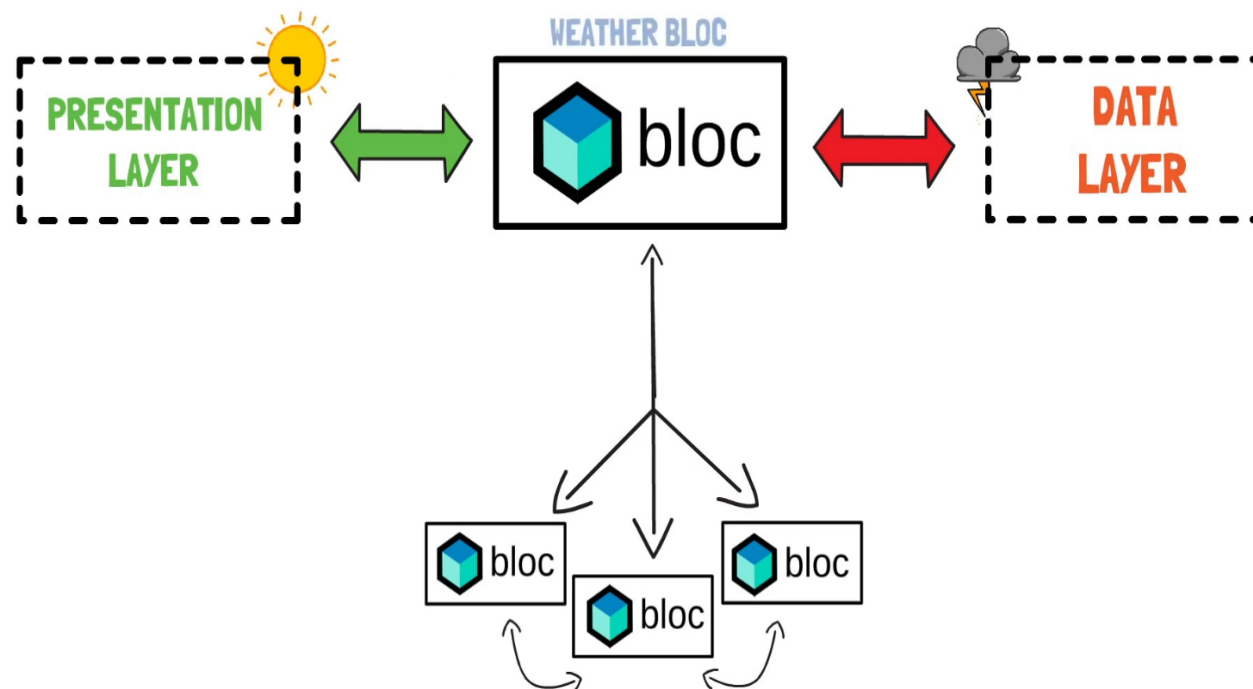
# Flutter – gestion du state : BLoC et Architecture



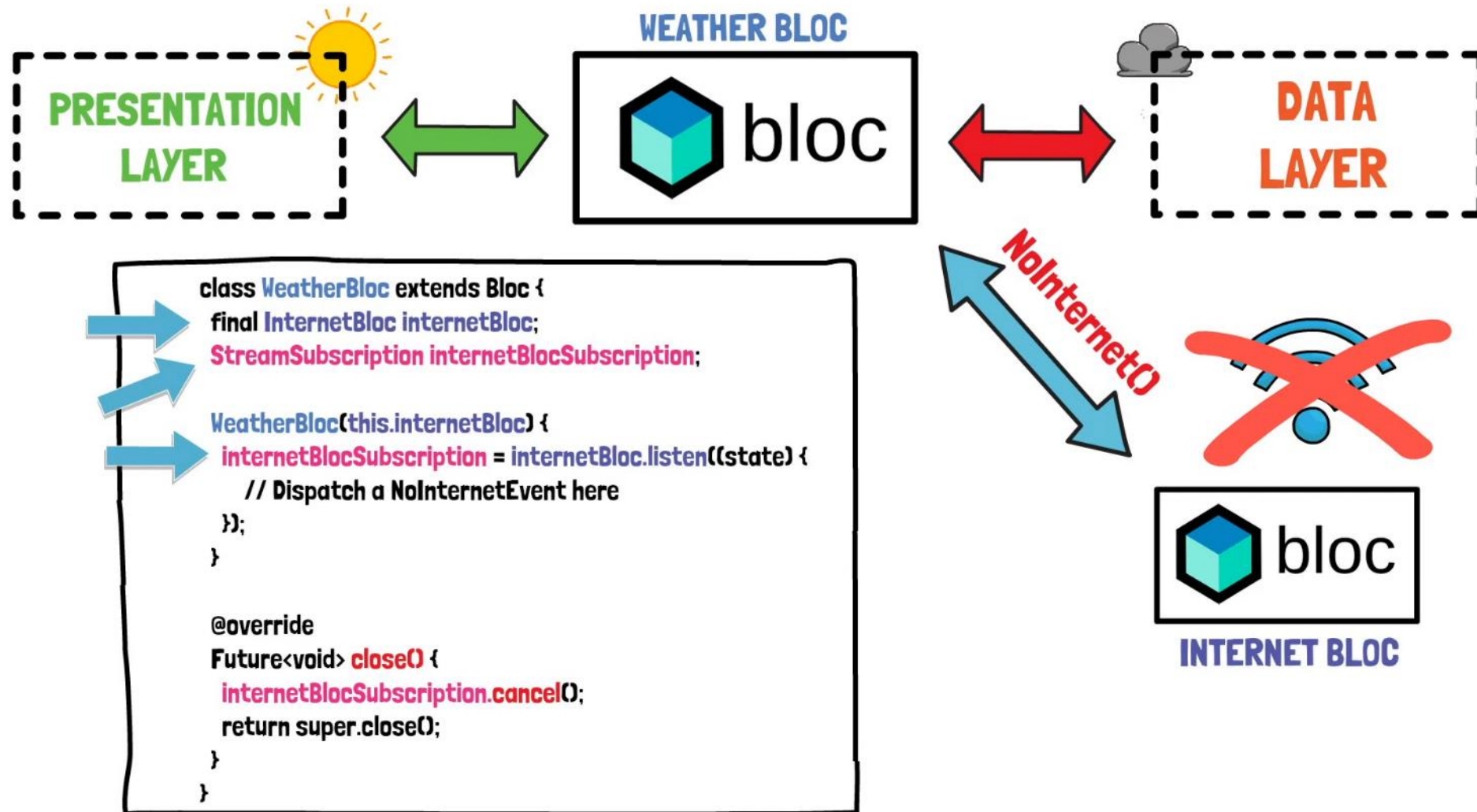
```
class WeatherBloc extends Bloc<WeatherEvent, WeatherState> {  
  final WeatherRepository repository;  
  
  Stream mapEventToState(event) async* {  
    if (event is GetWeatherEvent) {  
      yield WeatherLoading();  
      try {  
        final Weather weather = await repository.getWeather(event.cityName);  
        yield WeatherLoaded(weather);  
      } catch (error) {  
        yield Failure(error);  
      }  
    }  
  }  
}
```

Here we're sending the user a **Failure state with an error** to let him know that something went wrong, instead of crashing the application unexpectedly

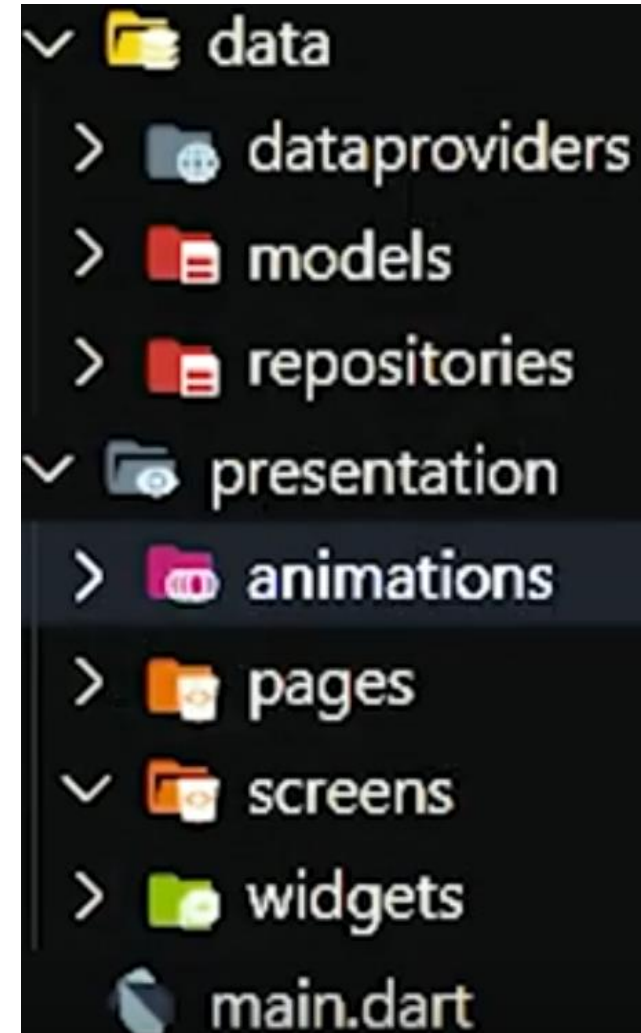
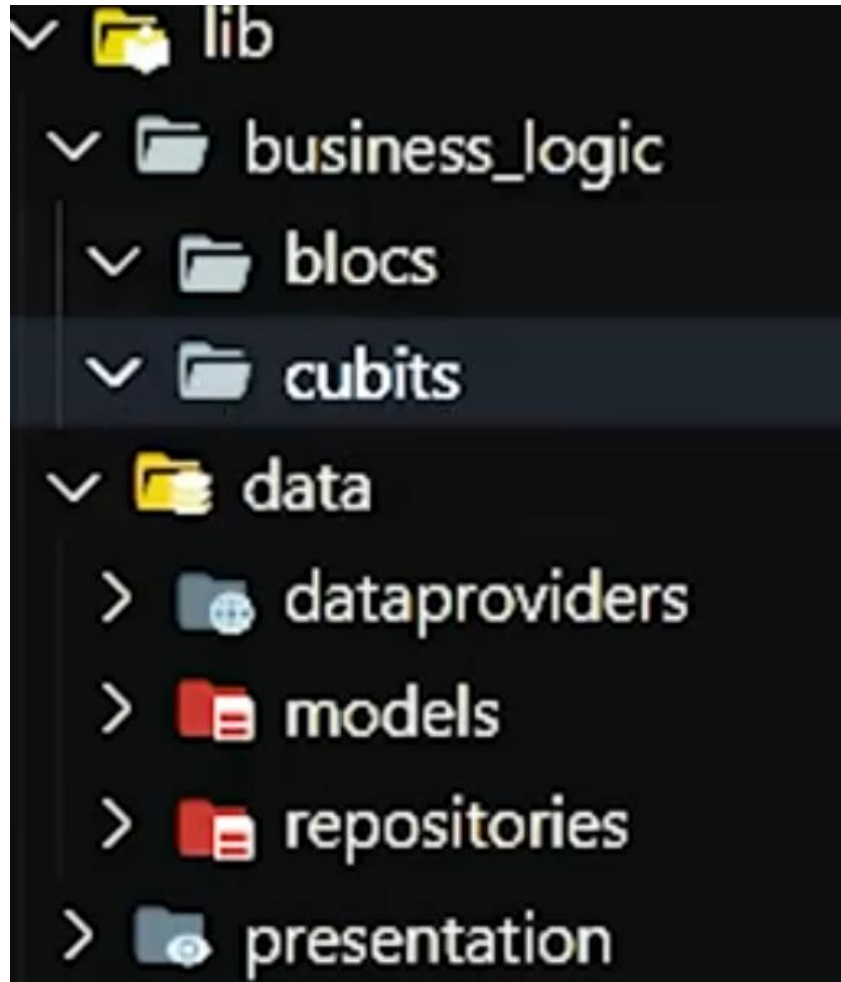
# Flutter – gestion du state : BLoC et Architecture



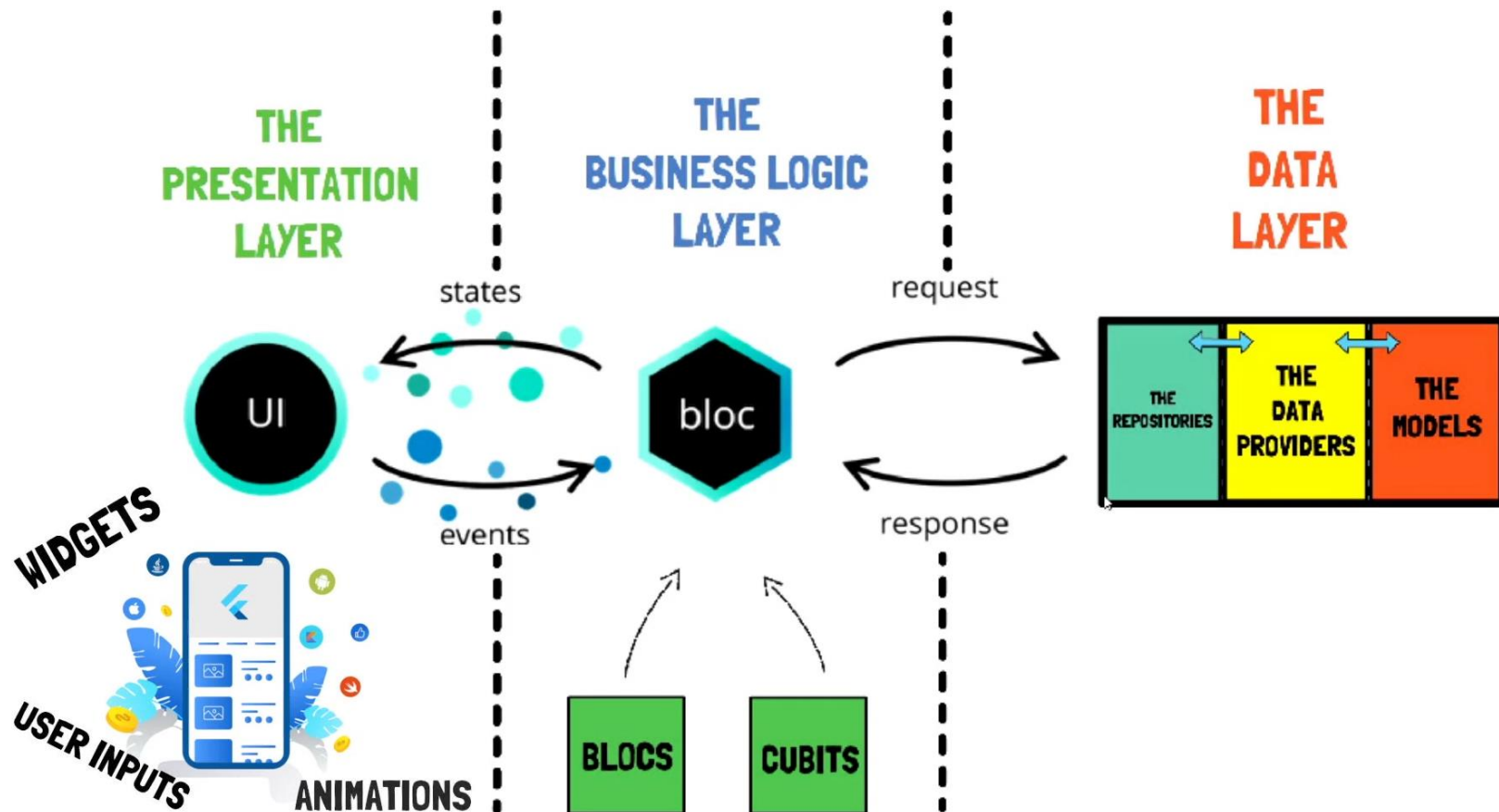
# Flutter – gestion du state : BLoC et Architecture



# Flutter – gestion du state : BLoC et Architecture



# Flutter – gestion du state : BLoC et Architecture





# Flutter – gestion du state : BLoC et Architecture

## BLOC ARCHITECTURE WORKFLOW (FOR A BASIC WEATHER APP)

Scenario: The user wants to know the weather for Chicago

