LeanCode

# State Management

How to wrap it up in my app?

# Ephemeral vs global

- Local
- Has my button been touched?
- Short-living in terms of a feature
- Selected tab in bottom navigation
- Current page in the IndexedStack

- Global
- Am I signed in?
- Long-living in terms of a feature, business process or the whole app
- User Authentication state
- Cart items state

# We already know about ephemeral state

# Ephemeral state

```dart
class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;

  void _incrementCounter() {
    setState(() {
      _counter++;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(widget.title),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            const Text(
              'You have pushed the button this many times:',
            ),
            Text(
              '$_counter',
              style: Theme.of(context).textTheme.headline4,
            ),
```

```dart
class _DraggableCardState extends State<DraggableCard>
    with SingleTickerProviderStateMixin {
  AnimationController? _controller;
  Alignment _dragAlignment = Alignment.center;
  Animation<Alignment>? _animation;

  void _runAnimation(Offset pixelsPerSecond, Size size) {
    _animation = _controller!.drive(
      AlignmentTween(
        begin: _dragAlignment,
        end: Alignment.center,
      ),
    );

    final unitsPerSecondX = pixelsPerSecond.dx / size.width;
    final unitsPerSecondY = pixelsPerSecond.dy / size.height;
    final unitsPerSecond = Offset(unitsPerSecondX, unitsPerSecondY);
    final unitVelocity = unitsPerSecond.distance;

    const spring = SpringDescription(
      mass: 30,
      stiffness: 1,
      damping: 1,
    );

    final simulation = SpringSimulation(spring, 0, 1, -unitVelocity);

    _controller!.animateWith(simulation);
  }

  @override
  void initState() {
    super.initState();
    _controller = AnimationController(vsync: this);

    _controller!.addListener(() {
      setState(() {
        _dragAlignment = _animation!.value;
      });
    });
  }
```

# Things that are tightly coupled with our UI

# What about our current pizza order in the app?

# Do we want to keep it in some widget?

# Is it connected to UI?

# Is it connected to UI? Somehow it is, because we need it as long as we are in the process.

# ChangeNotifier

```dart
class Counter with ChangeNotifier {
  int _count = 0;

  int get count => _count;

  void increment() {
    _count++;
    notifyListeners();
  }
}
```

```dart
MultiProvider(
  providers: [
    ChangeNotifierProvider(create: (context) => Counter()),
  ],
  child: const MyApp(),
),
```

```dart
floatingActionButton: FloatingActionButton(
  onPressed: () => context.read<Counter>().increment(),
  child: const Icon(Icons.add),
),
```

```dart
class Count extends StatelessWidget {
  const Count({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Text(
      '${context.watch<Counter>().count}',
      key: const Key('counterState'),
      style: Theme.of(context).textTheme.headline4,
    );
  }
}
```

# Pretty simple, huh?

# It’s just a class so we can do whatever we like

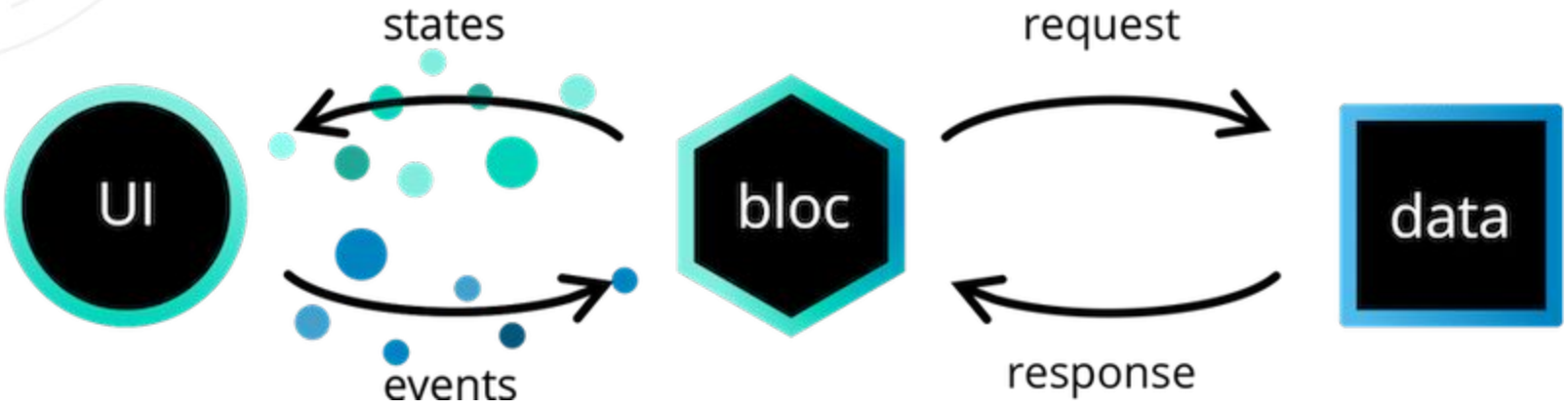
# What if…?

```
class Counter with ChangeNotifier {
  int _count = 0;

  int get count => _count;

  void increment() {
    _count++;
  }
}
```

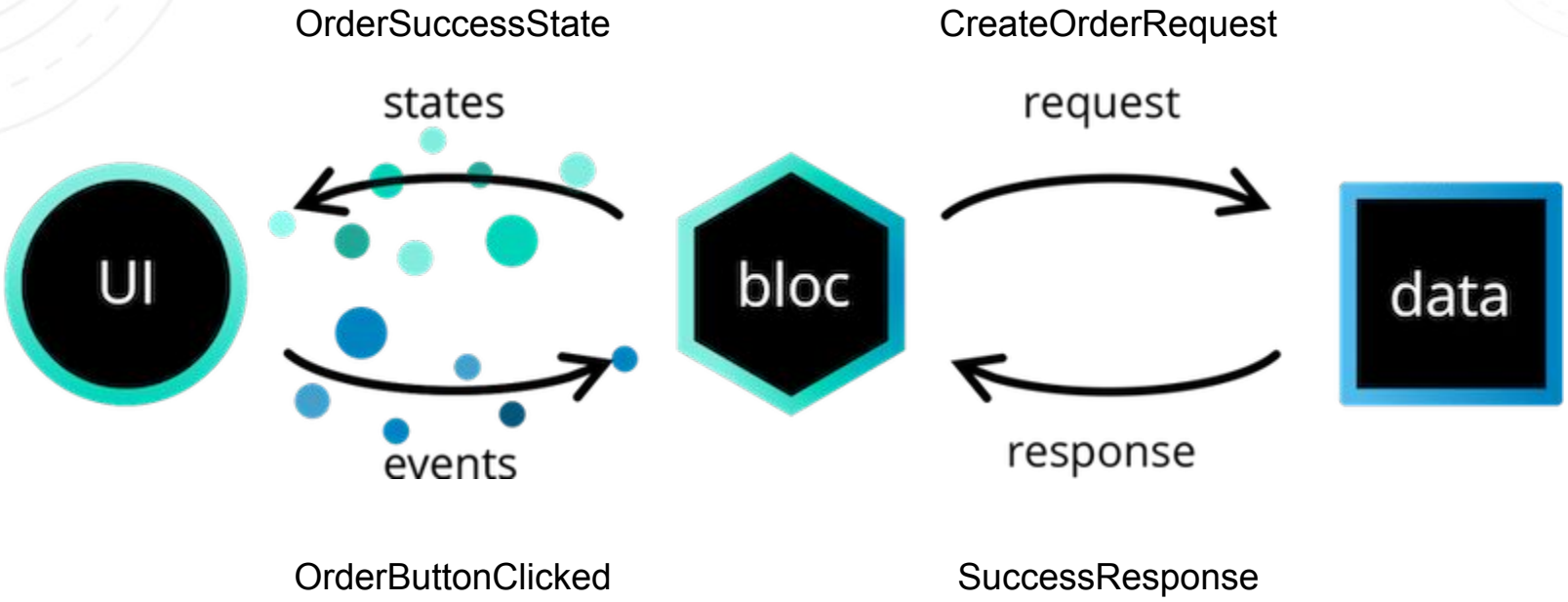
# Doesn’t standardize

```dart
class ValueNotifier<T> extends ChangeNotifier implements ValueListenable<T> {
  /// Creates a [ChangeNotifier] that wraps this value.
  ValueNotifier(this._value);

  /// The current value stored in this notifier.
  ///
  /// When the value is replaced with something that is not equal to the old
  /// value as evaluated by the equality operator ==, this class notifies its
  /// listeners.
  @override
  T get value => _value;
  T _value;
  set value(T newValue) {
    if (_value == newValue)
      return;
    _value = newValue;
    notifyListeners();
  }

  @override
  String toString() => '${describeIdentity(this)}($value)';
}
```

# Bloc



# Bloc



# UI = f(state)

# Bloc

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

```dart
class CounterBloc extends Bloc<CounterEvent, int> {
  CounterBloc() : super(0) {
    on<Increment>((event, emit) => emit(state + 1));
    on<Decrement>((event, emit) => emit(state - 1));
  }
}
```

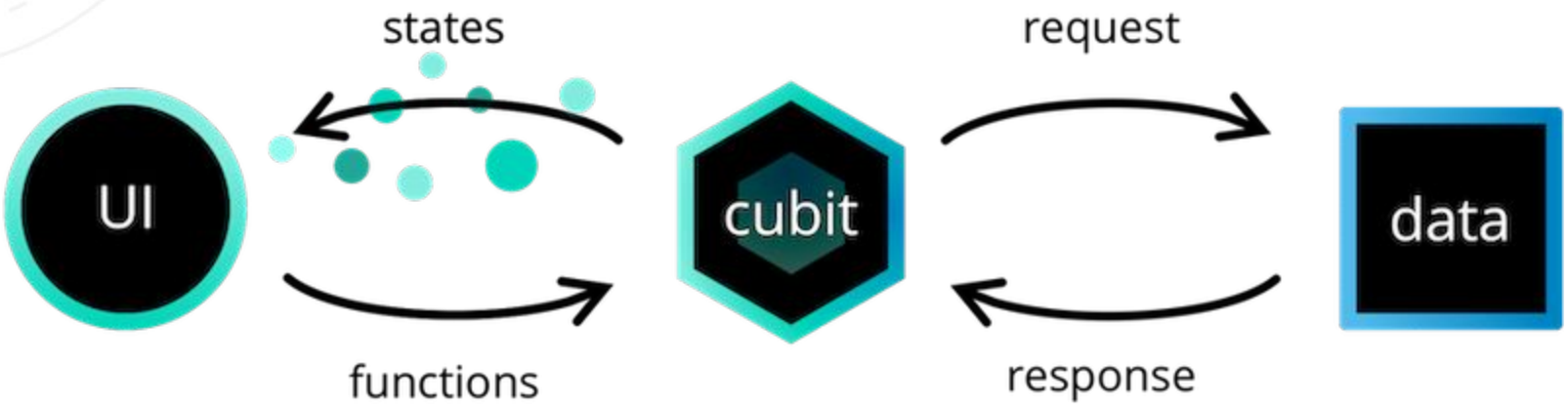
# Bloc

```
Padding(
  padding: const EdgeInsets.symmetric(vertical: 5.0),
  child: FloatingActionButton(
    child: const Icon(Icons.add),
    onPressed: () =>
        context.read<CounterBloc>().add(Increment()),
  ),
),
```

```
return MaterialApp(
  theme: theme,
  home: BlocProvider(
    create: (_) => CounterBloc(),
    child: CounterPage(),
  ),
);
```

```
child: BlocBuilder<CounterBloc, int>(
  builder: (context, count) {
    return Text(
      '$count',
      style: Theme.of(context).textTheme.headline1,
    );
  },
),
```

# Cubit



# Cubit

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

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

# Sources

https://flutter.dev
https://dartpad.dev
https://bloclibrary.dev