Как заваривать декларативный чай

Что там опять эти хипстеры придумали и почему стоит этому радоваться

DELIVERY CLUB TECH

Структура доклада

1

Как выбрать правильный сервиз для чайной церемонии

2

Дружим TEA **вместе с** Compose

3

Сессия лайвкодинга

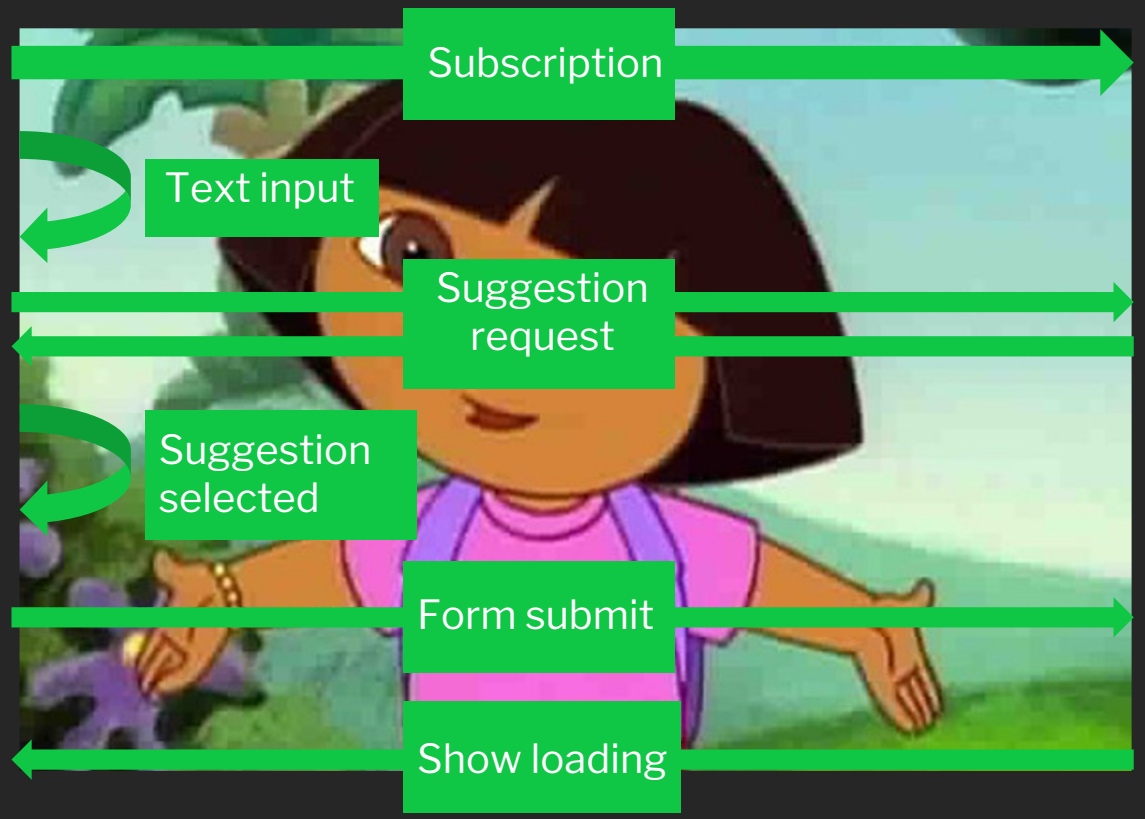
Обо мне

- Андроид разработчик в Delivery Club
- 3 года андроид разработки
- Люблю экспериментировать с различными технолгиями

@murоming

Флоу данных в приложении

Частично тут



Где находится состояние?

View Model

Suggestions

LoadingStatus

onTextChanged()

onSuggestionSelected()

onFormSubmitted()

Частично тут

Почему это плохо

1

Раздробленность состояния

2

Множество способов его изменения

3

Тяжело поддерживать и расширять

TEA **спешит на помощь**

TEA – The Elm Architecture

Unidirectional Data Flow + Functional Programming = 

Unidirectional Data Flow

1

Состояние хранится только в 1 **месте, формируя** SSoT

2

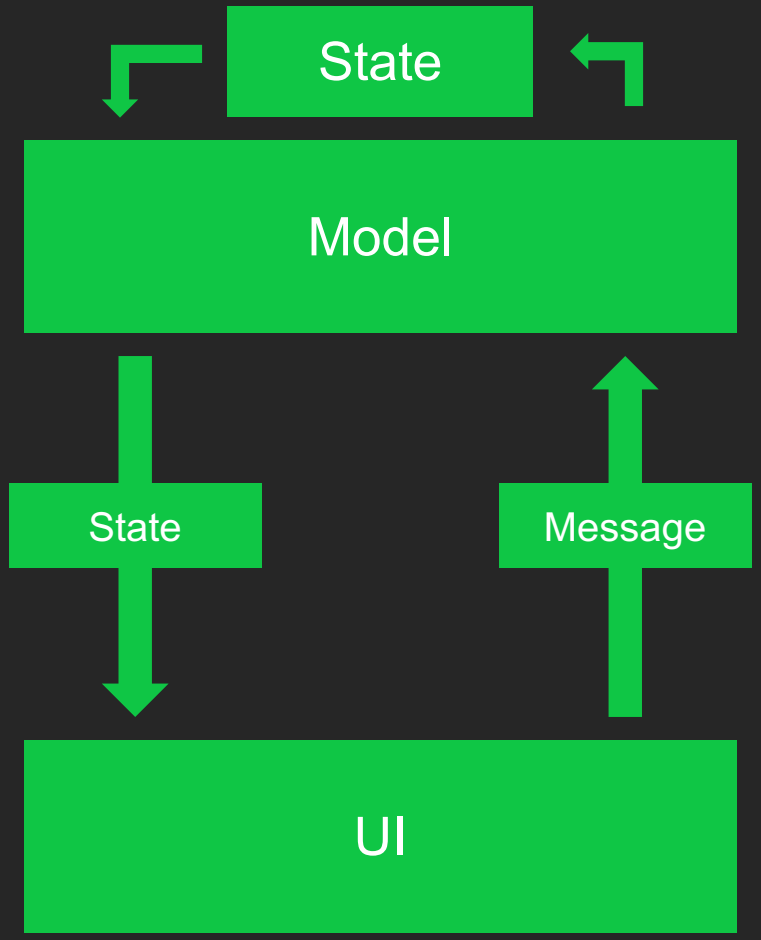
Состояние может быть изменено только ограниченным набором способов

3

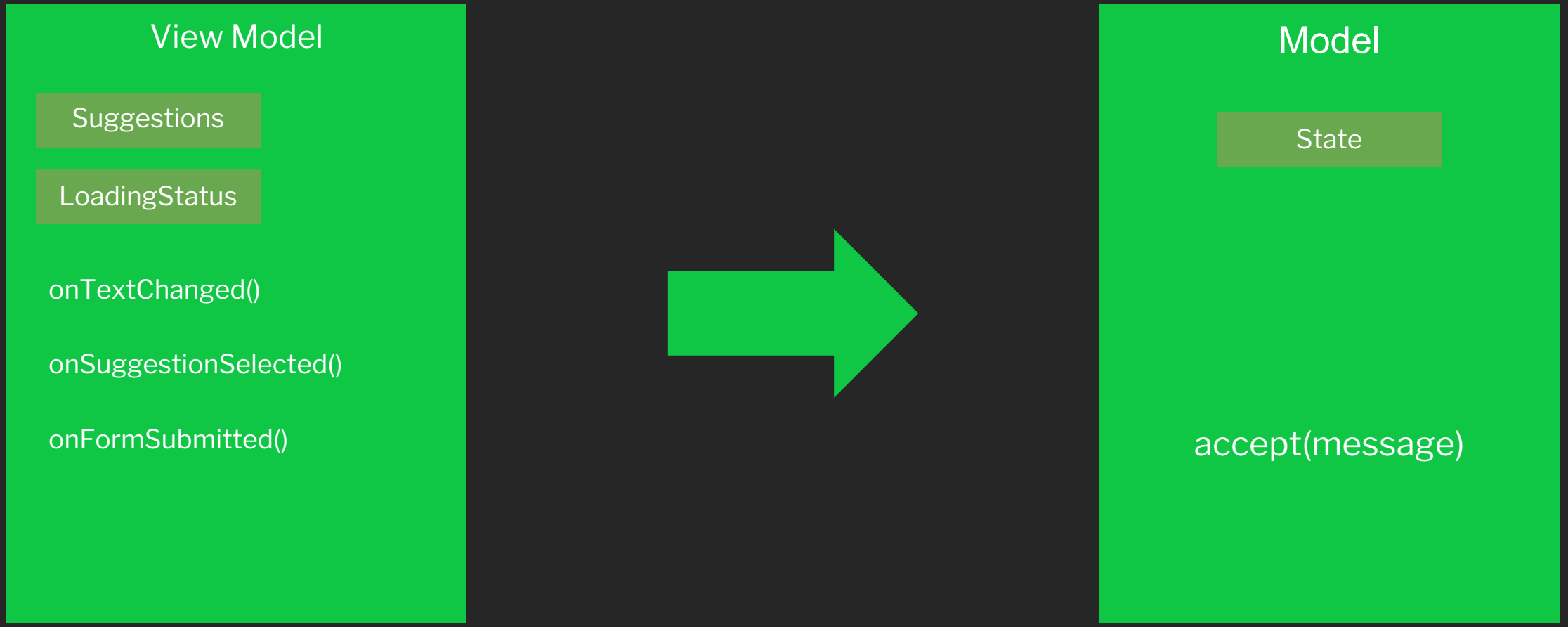
После изменения состояние передается обзерверам

Unidirectional Data Flow

- Model
  Хранит в себе состояние и изменяет его в соответствии со входящими сообщениями
- UI
  Выполняет безусловную отрисовку состояния
- Message
  Событие от UI слоя о взаимодействии



Unidirectional Data Flow



Функциональное программирование

1

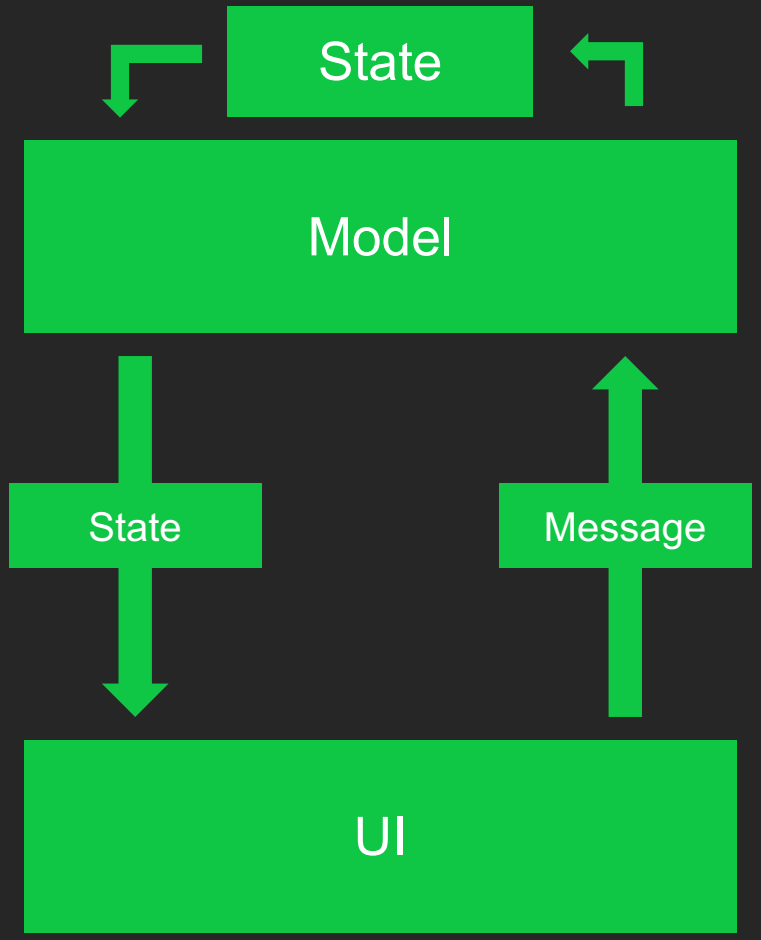
Объекты неизменяемы

2

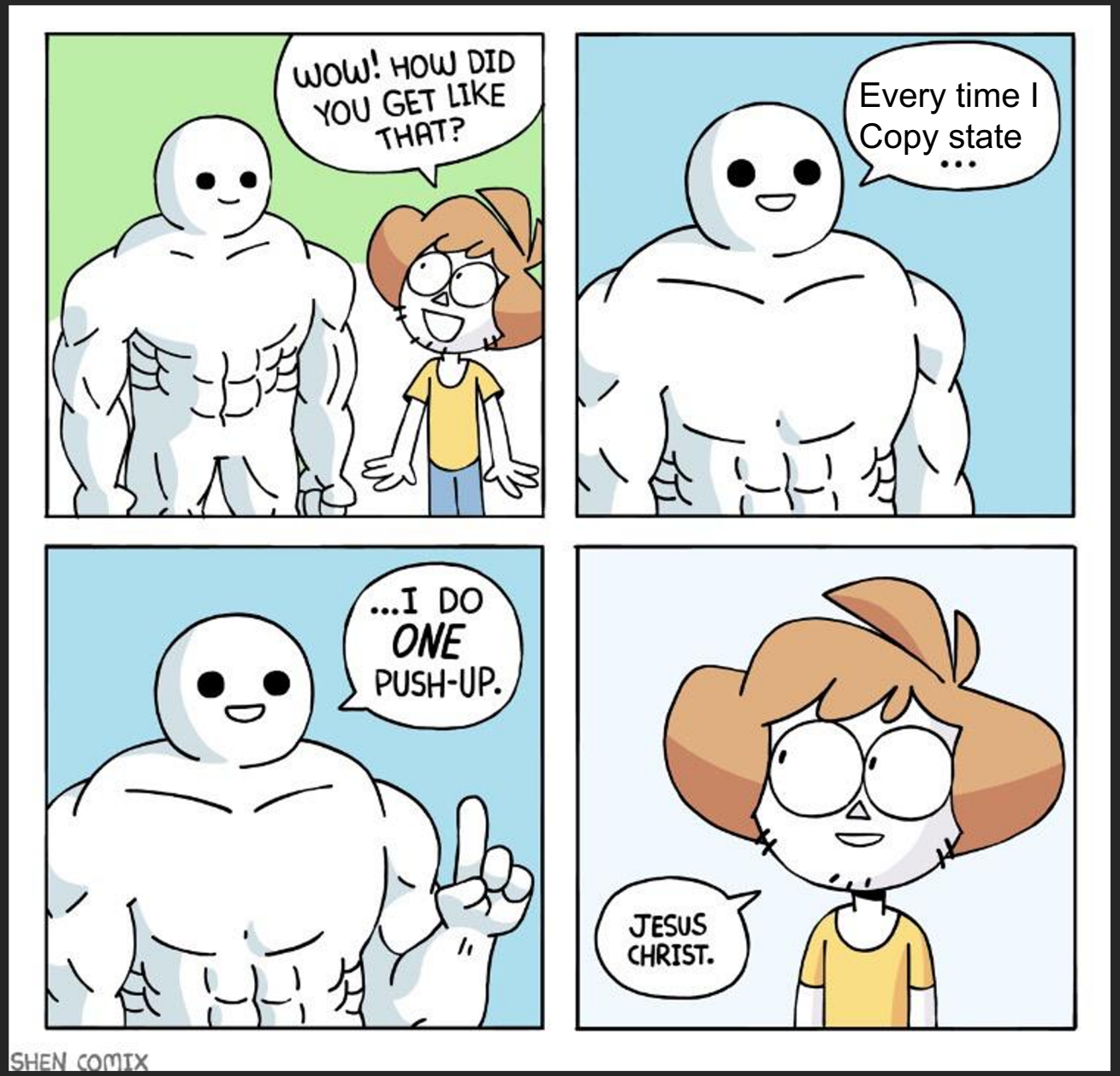
Чистые функции без сторонних эффектов

Неизменяемые объекты

- **State**
  Представляет собой полное состояние функционала
- **Message**
  Транзакция по изменению стейта



Как изменить неизменяемое



Что такое чистая функция?

При условии x = x, будут ли равны два последовательных вызова функции f(x)?

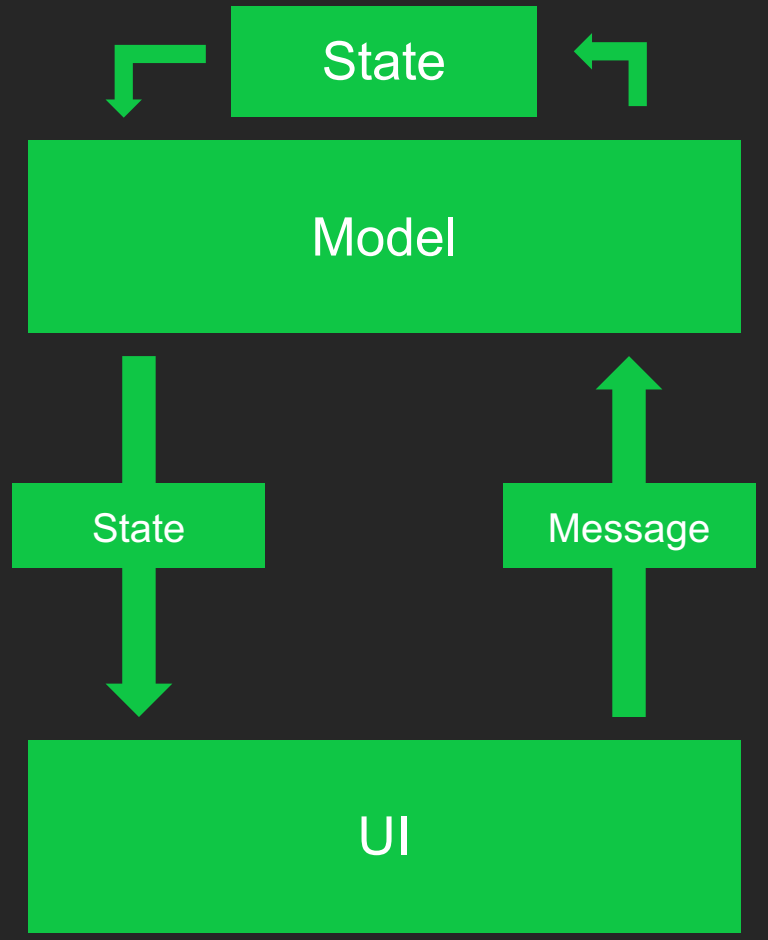
Да, если это чистая функция

- sin(x)
- round(x)
- hash(x)
- x + 2

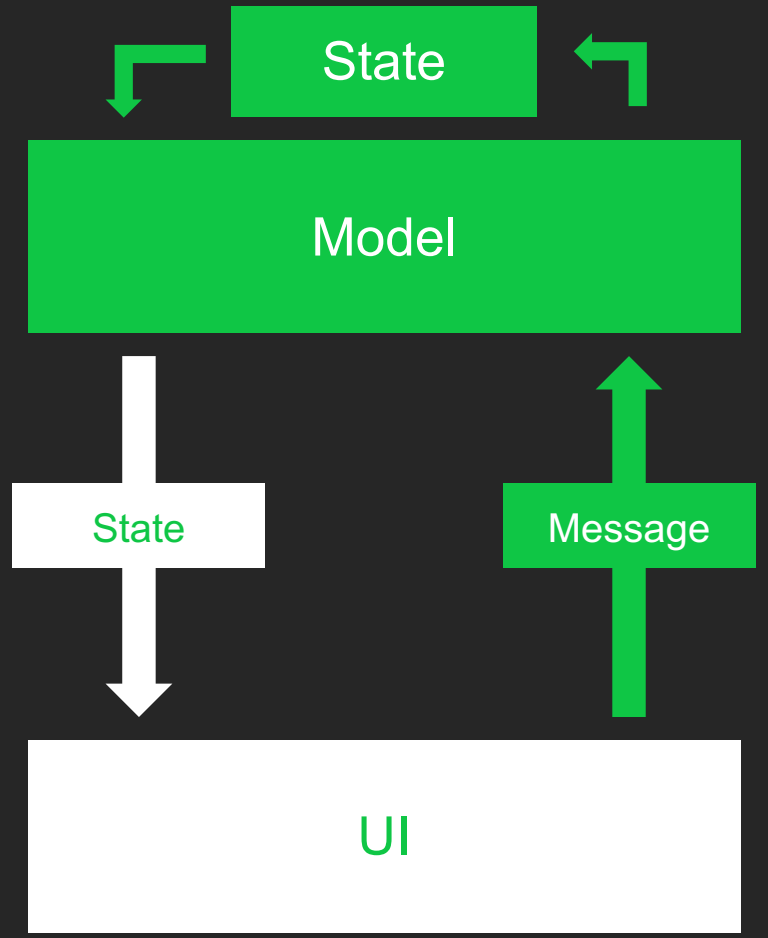
Нет, если функция содержит сайд эффекты

- sendXtoServer(x)
- x + randInt()
- saveXinDatabase(x)

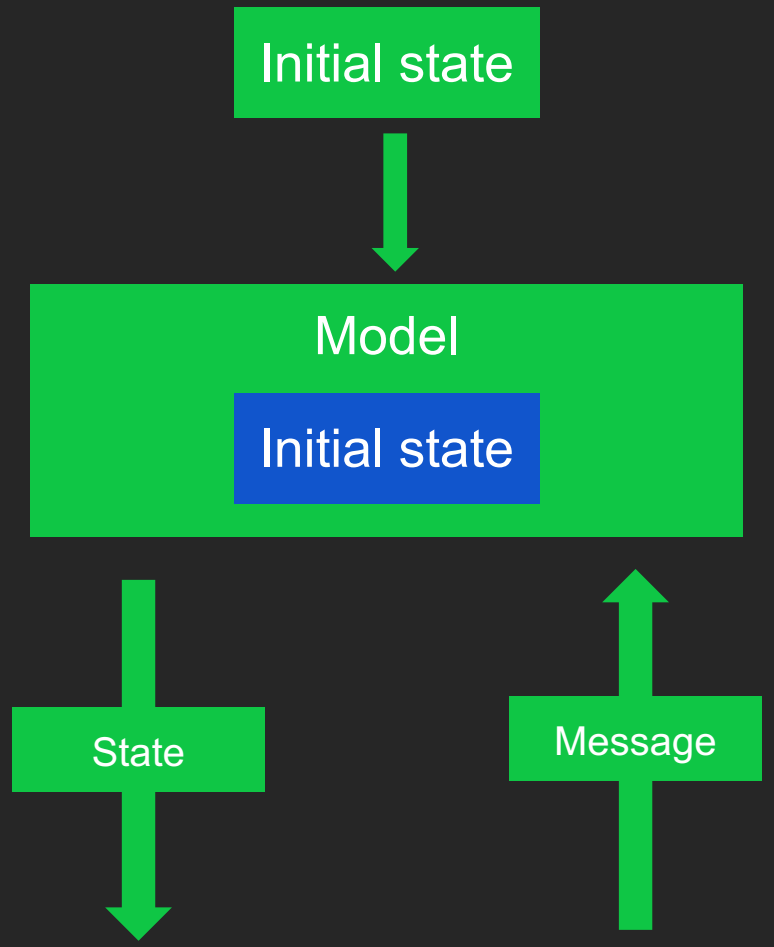
Складываем все вместе



Складываем все вместе



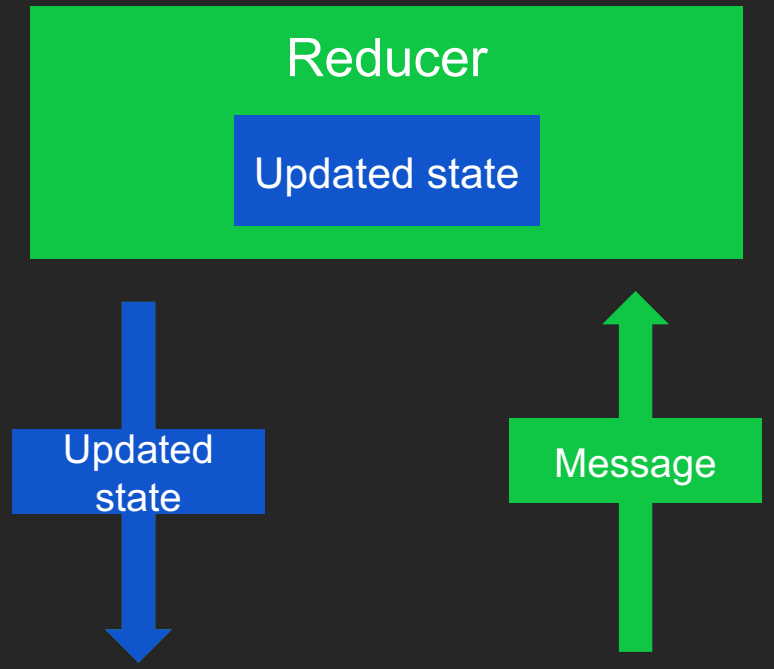
Новый флоу данных



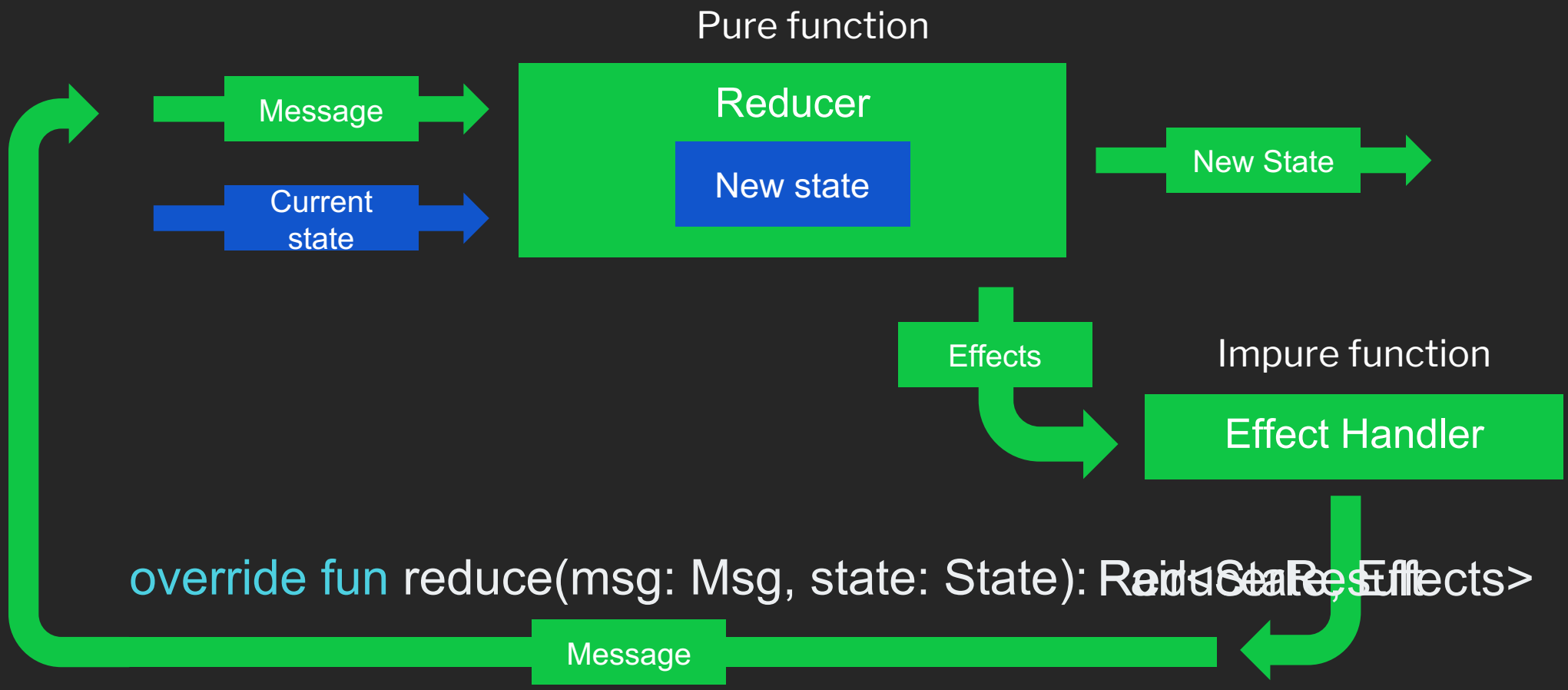
Что такое reducer


```
override fun reduce(msg: Msg, state: State): ReducerResult
```

Новый флоу данных



Добавим щепотку сайд эффектов



Recap

1

Основные компоненты TEA

- State
- Message
- Reducer
- Effect
- EffectHandler

2

Взаимодействие между UI **и состоянием**

UI не имеет собственного состояния. Он только отображает пришедешее ему состояние и делегирует его изменение редюсеру

3

Только редюсер может изменять состояние

Редюсер это чистая функция, которая возвращает новое состояние на входящее сообщение

4

EffectHandler **выполняет функции с сайд эффектами**

Все асинхронные операции выносятся в эффекты и обрабатываются внутри эффект хендлера

Jetpack compose

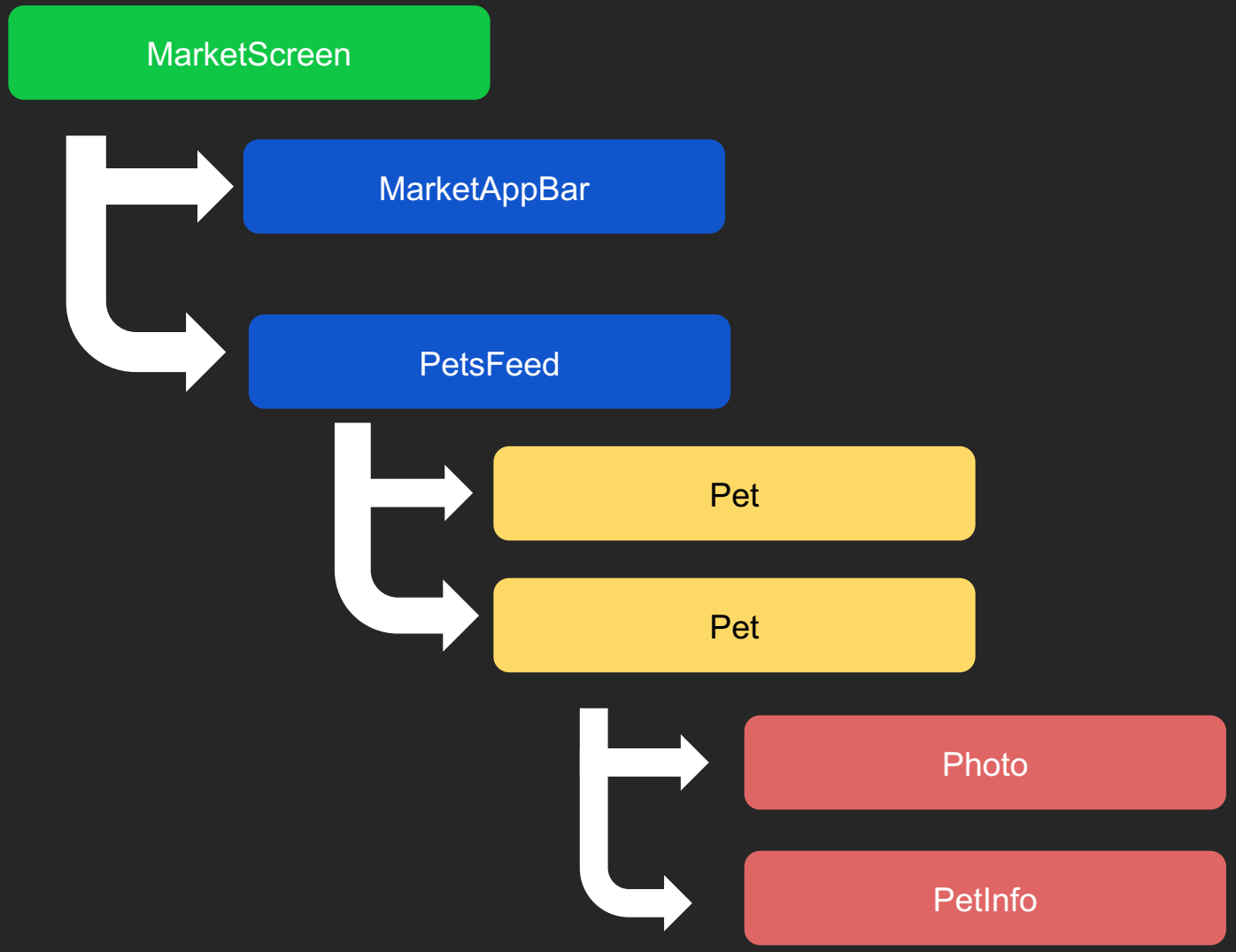
Новый декларативный UI фреймворк

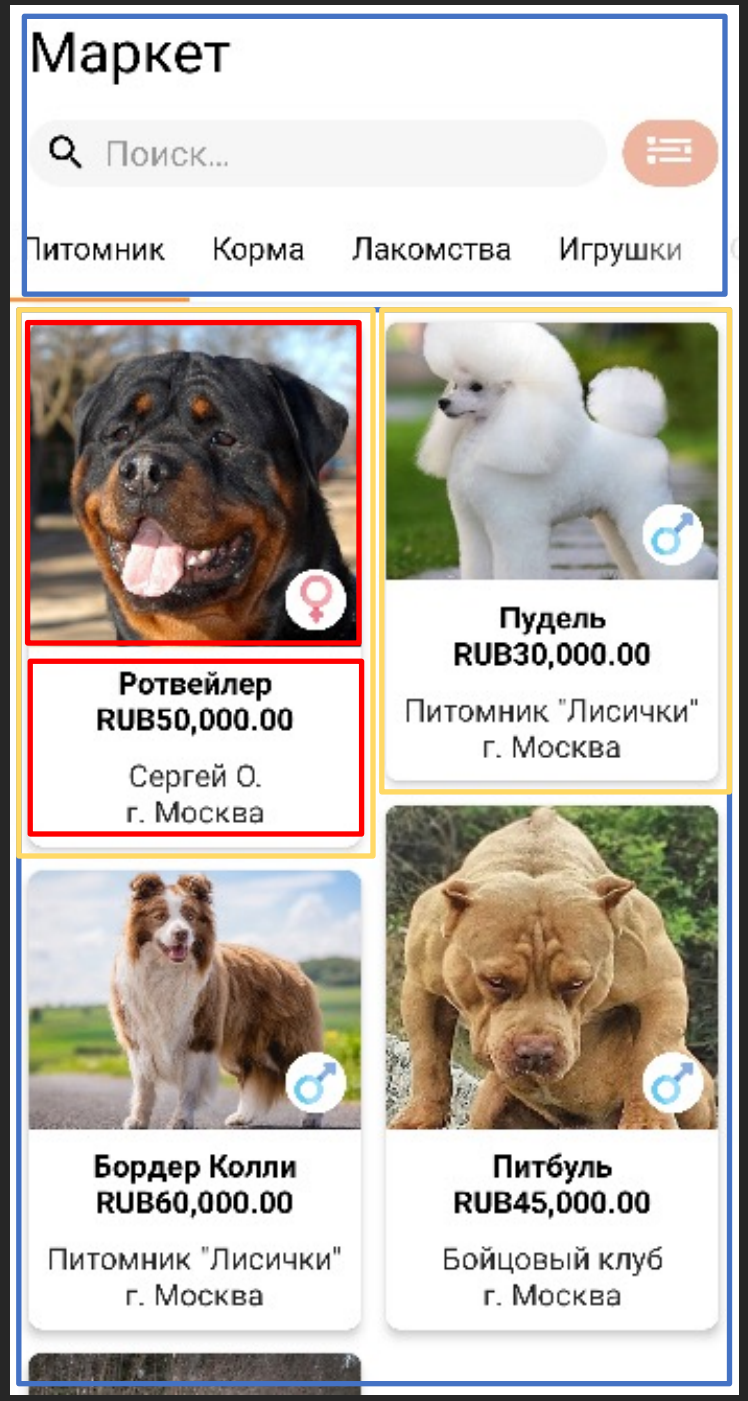
- Декларативный
- Откреплен от Андроид фреймворка
- Имеет древовидную структуру

Основы compose

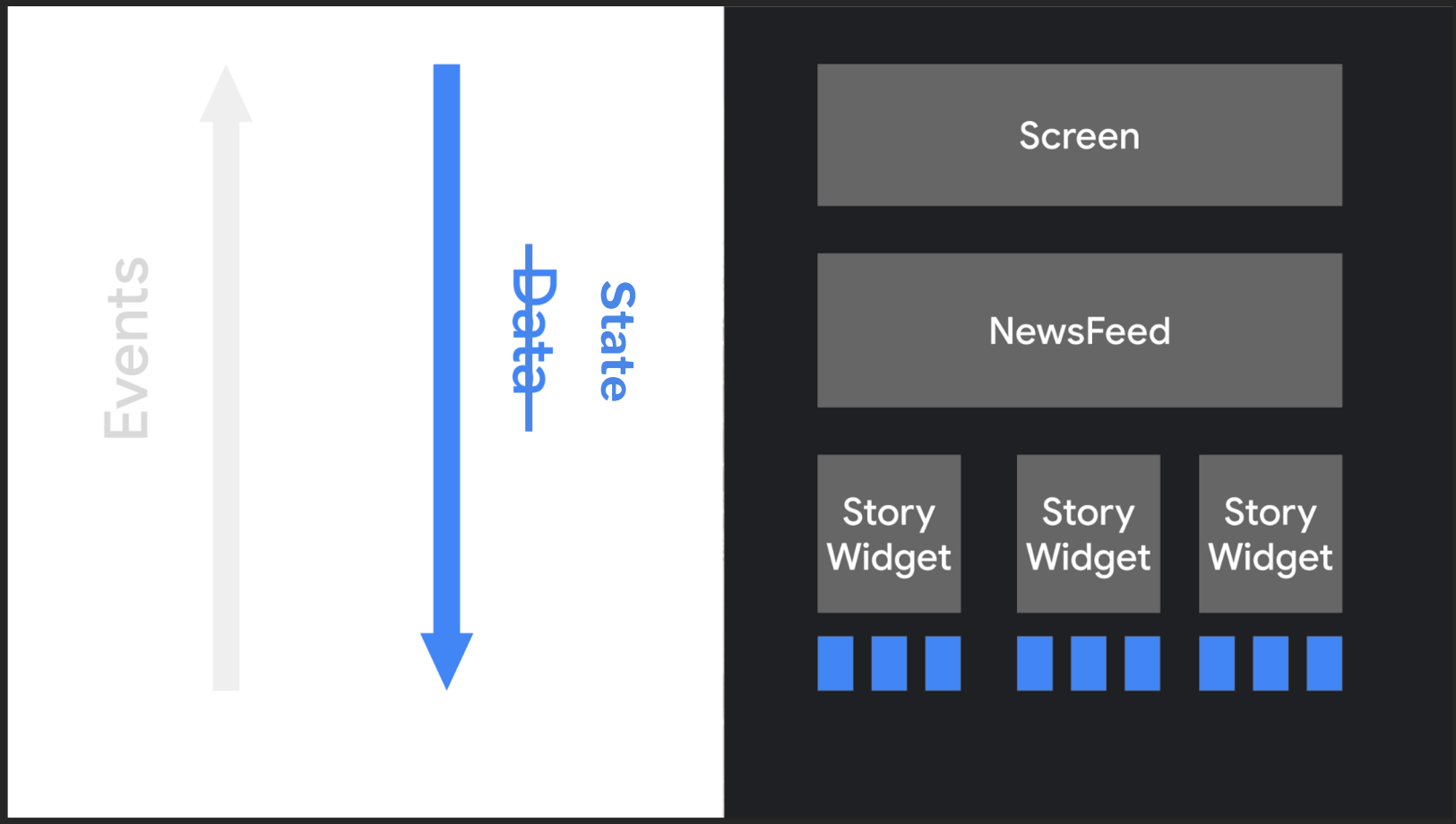
```
@Composable
fun MyCoolScreen(state: ScreenState) {
...
}
```

Дерево compose функций





Thinking in compose



Основы compose

```
@Composable
fun CheckoutContent(
 screen: Screen.Checkout.Loaded,
 priceFormatter: PriceFormatter,
 modifier: Modifier = Modifier,
 action: (CheckoutFeature.Msg) -> Unit
) {
        when (screen) {
        Column {is Screen.Checkout.Loading -> Loading(Modifier.fillMaxSize())
                is.Screen.Checkout.Error -> GenericLoadingError {...}
                Order Screen.Checkout.Loaded -> CheckoutContent(
                        screen,
                        priceFormatter.formatPrice(screen.orderPrice),
                        priceFormatter,formatPrice(screen.deliveryPrice),
                        Modifier.padding(0.dp),formatPrice(screen.totalPrice)
                )       action
                )..
        }
}
```

Основы compose

```
@Composable
fun CheckoutContent(
 screen: Screen.Checkout.Loaded,
 priceFormatter: PriceFormatter,
 modifier: Modifier = Modifier,
 action: (CheckoutFeature.Msg) -> Unit
) {
        Column {
                ...
                OrderPricing(
                        priceFormatter.formatPrice(screen.orderPrice),
                        priceFormatter.formatPrice(screen.deliveryPrice),
                        priceFormatter.formatPrice(screen.totalPrice)
                )
                ...
        }
}
```

Основы compose

```
@Composable
fun CheckoutContent(
 screen: Screen.Checkout.Loaded,
 priceFormatter: PriceFormatter,
 modifier: Modifier = Modifier,
 action: (CheckoutFeature.Msg) -> Unit
) {
        Column {
                ...
                OrderPricing(
                        priceFormatter.formatPrice(screen.orderPrice),
                        priceFormatter.formatPrice(screen.deliveryPrice),
                        priceFormatter.formatPrice(screen.totalPrice)
                )
                ...
        }
}
```

Основы compose

```
@Composable
fun OrderPricing(
 orderPrice: String,
 deliveryPrice: String?,
 totalPrice: String
) {
        Column {
                Row {
                        Text(text = "Цена", fontSize = 14.sp)
                        Text(text = orderPrice, fontSize = 14.sp)
                }
                Row {
                        Text(text = "Цена доставки", fontSize = 14.sp)
                        Text(text = deliveryPrice ?: "Бесплатно", fontSize = 14.sp)
                }
                Row {
                        Text(text = "Итого", style = MaterialTheme.typography.h6)
                        Text(text = totalPrice, style = MaterialTheme.typography.h6)
                }
        }
}
```

Основы compose

```
@Composable
fun OrderPricing(
 orderPrice: String,
 deliveryPrice: String?,
 totalPrice: String
) {
        Column {
                Row {
                        Text(text = "Цена", fontSize = 14.sp)
                        Text(text = orderPrice, fontSize = 14.sp)
                }
                Row {
                        Text(text = "Цена доставки", fontSize = 14.sp)
                        Text(text = deliveryPrice ?: "Бесплатно", fontSize = 14.sp)
                }
                Row {
                        Text(text = "Итого", style = MaterialTheme.typography.h6)
                        Text(text = totalPrice, style = MaterialTheme.typography.h6)
                }
        }
}
```

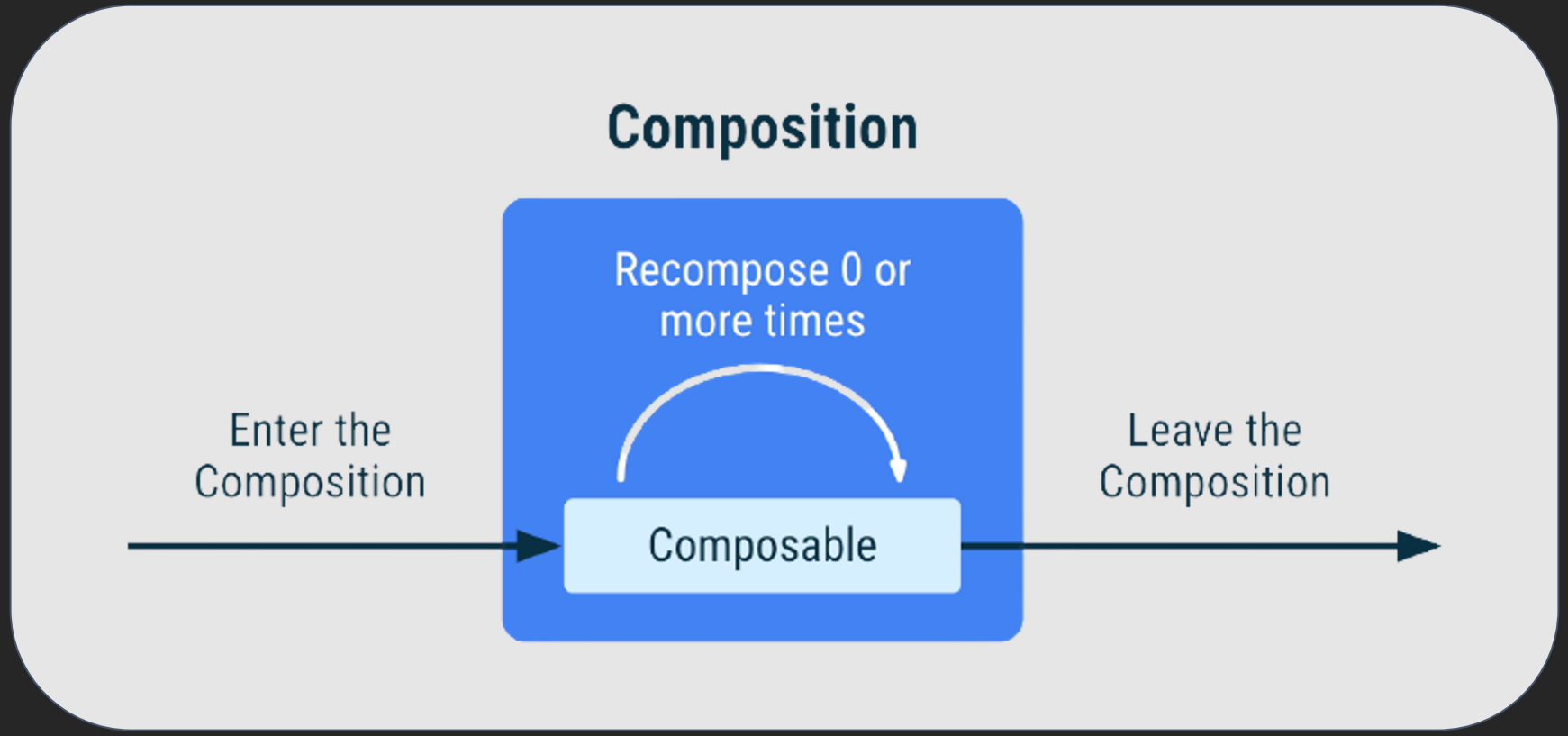
Основы compose

```
@Composable
fun OrderPricing(
 orderPrice: String,
 deliveryPrice: String?,
 totalPrice: String
) {
        Column {
                Row {
                        Text(text = "Цена", fontSize = 14.sp)
                        Text(text = orderPrice, fontSize = 14.sp)
                }
                Row {
                        Text(text = "Цена доставки", fontSize = 14.sp)
                        Text(text = deliveryPrice ?: "Бесплатно", fontSize = 14.sp)
                }
                Row {
                        Text(text = "Итого", style = MaterialTheme.typography.h6)
                        Text(text = totalPrice, style = MaterialTheme.typography.h6)
                }
        }
}
```

Основы compose

```
@Composable
fun OrderPricing(
 orderPrice: String,
 deliveryPrice: String?,
 totalPrice: String
) {
        Column {
                Row {
                        Text(text = "Цена”, fontSize = 14.sp)
                        Text(text = orderPrice, fontSize = 14.sp)
                }
                Row {
                        Text(text = "Цена доставки", fontSize = 14.sp)
                        Text(text = deliveryPrice ?: "Бесплатно", fontSize = 14.sp)
                }
                Row {
                        Text(text = "Итого", style = MaterialTheme.typography.h6)
                        Text(text = totalPrice, style = MaterialTheme.typography.h6)
                }
        }
}
```

Compose lifecycle

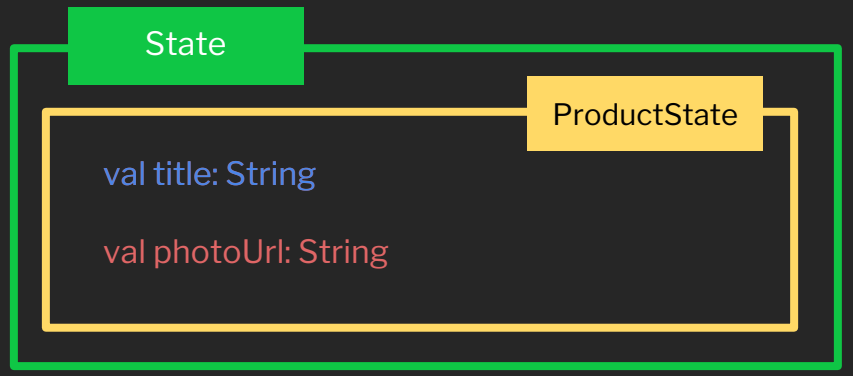


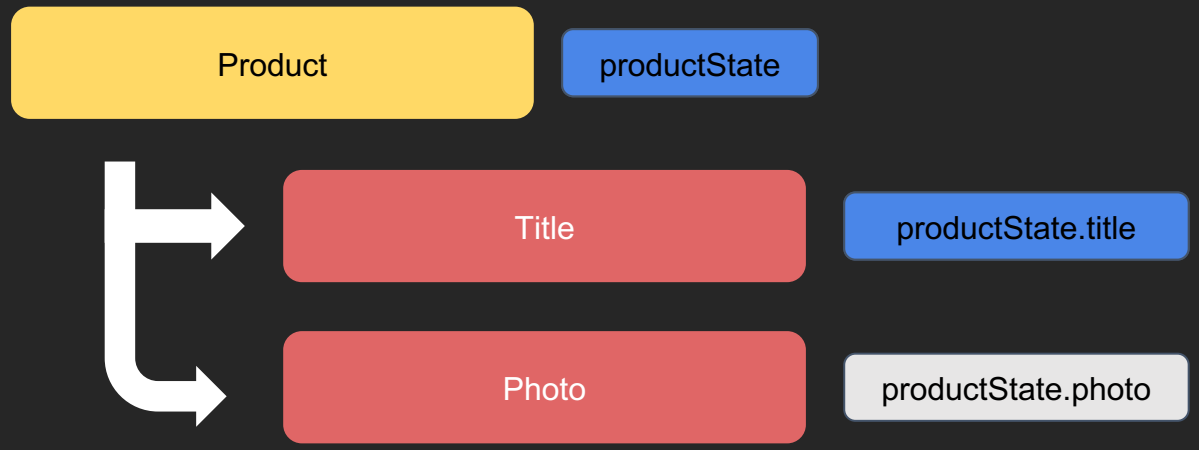
Использование посредника

```
@Stable
interface State<out T> {
    val value: T
}
```

```
@Stable
interface MutableState<T> : State<T> {
    override var value: T
    operator fun component1(): T
    operator fun component2(): (T) -> Unit
}
```

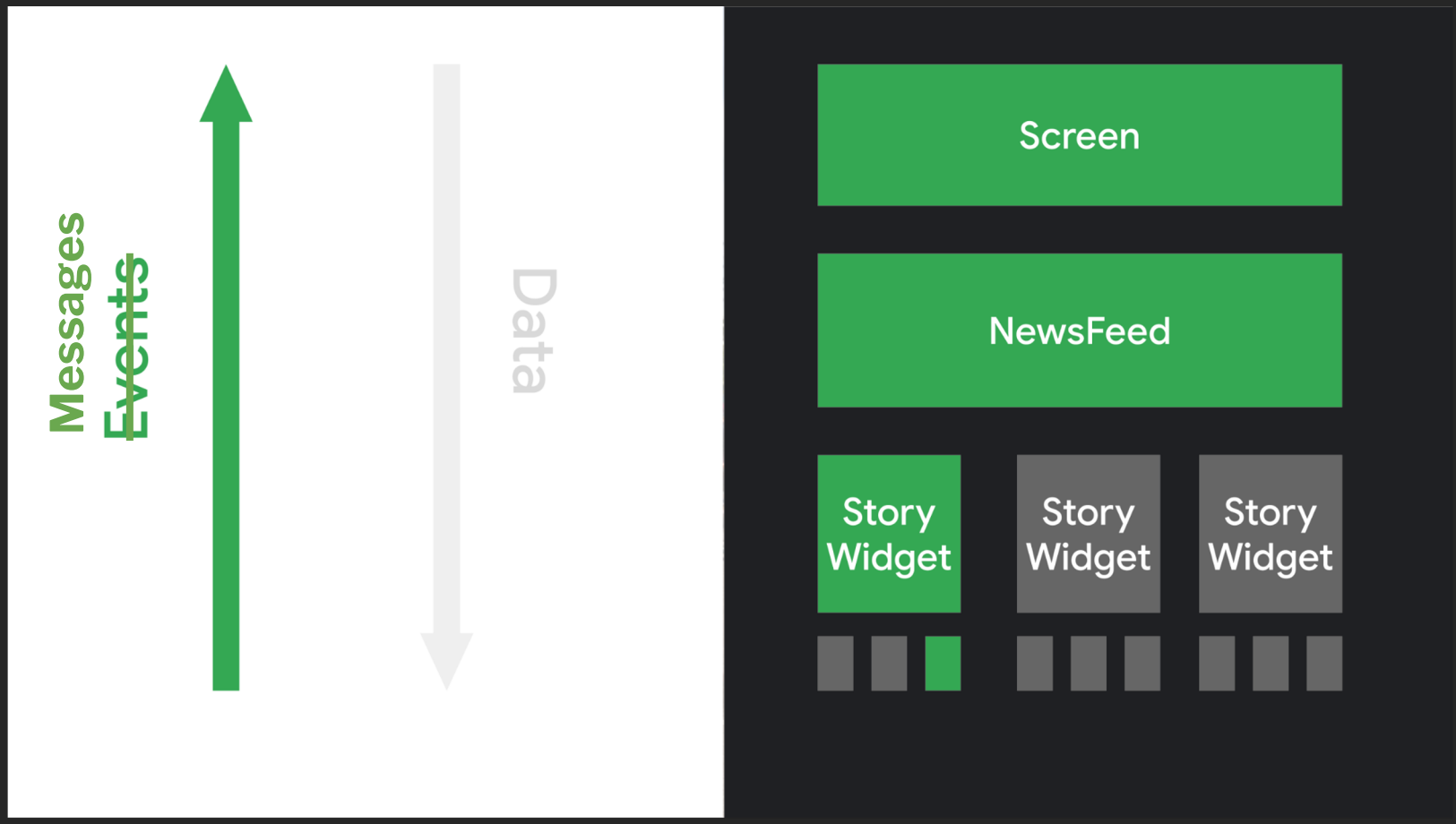
Наблюдаем за состоянием



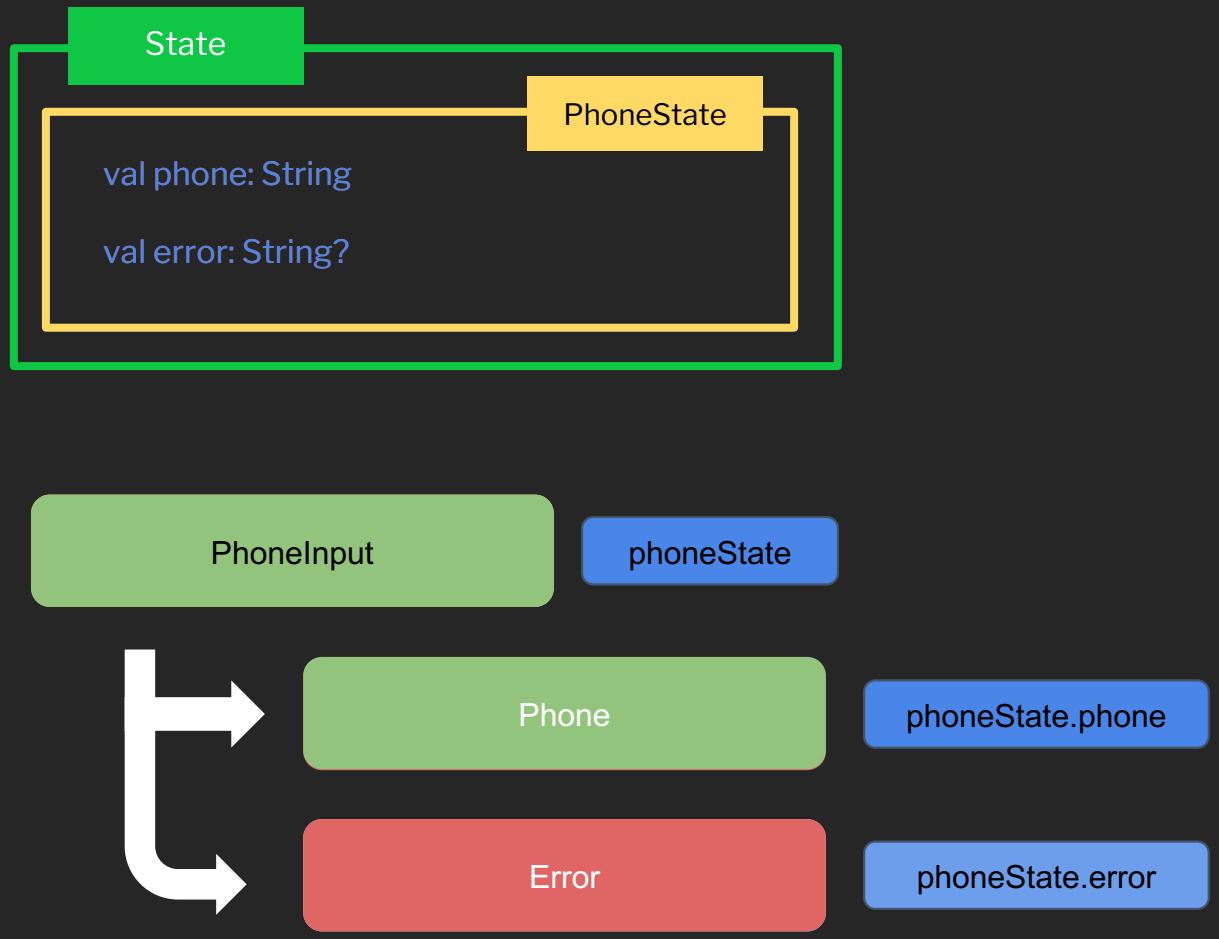


Обновлением состояние после редюсера

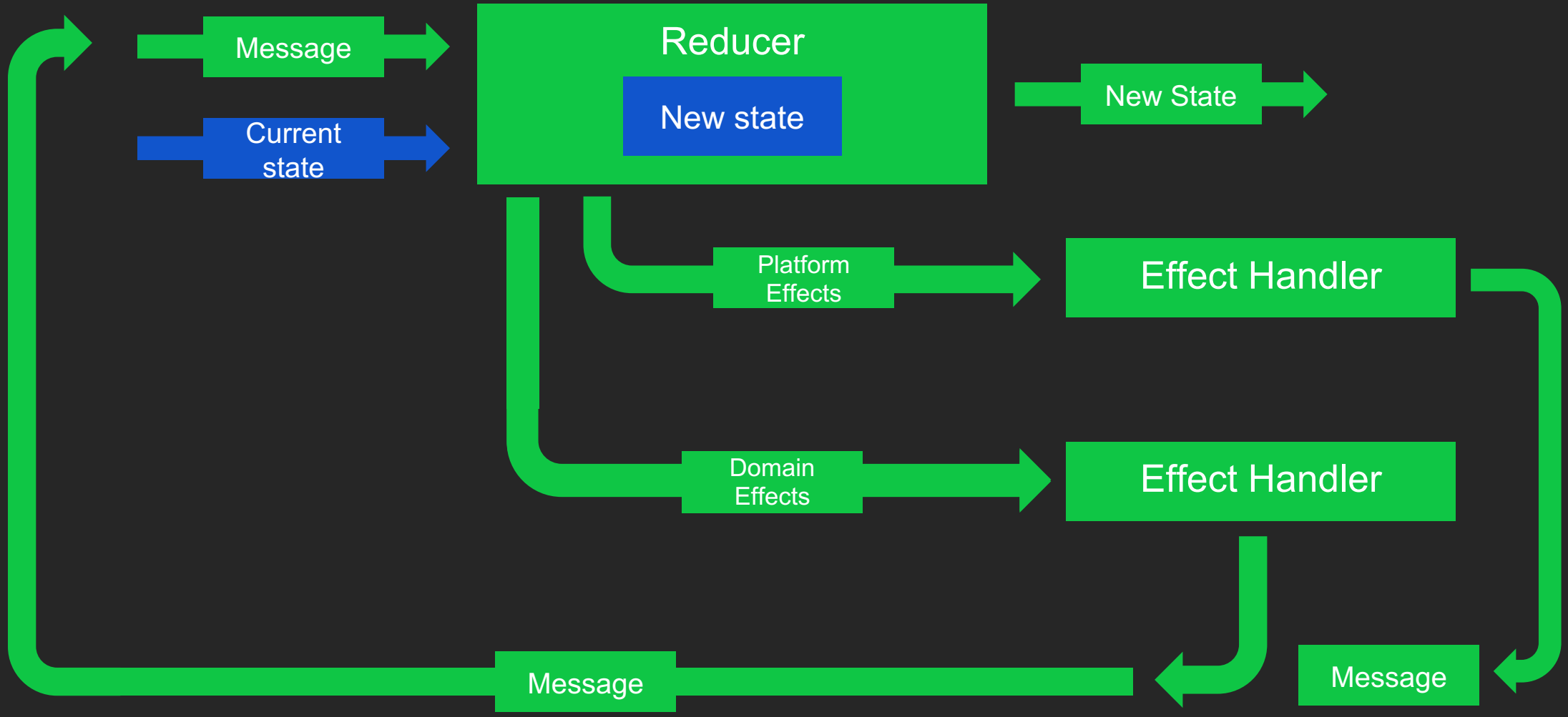
Thinking in compose



Всплывающие сообщения



Обработка эффектов



Integration recap

1

Состояние TEA является SSoT для построения UI

2

Всё взаимодействие пользователя всплывает вверх по дереву в виде сообщений

3

Вся асинхронная логика реализована в виде эффектов

Сессия лайвкондинга