Finita la State Machine — MVI в iOS приложении

Александр Цыбулько

Александр Цыбулько

iOS-разработчик в hh.ru

Работаю в продуктовой команде

Люблю затереть про архитектуру

@alextsybulko

О чём сегодня будем говорить?

Finita la State Machine — MVI в iOS приложении

Ссылка на слайды

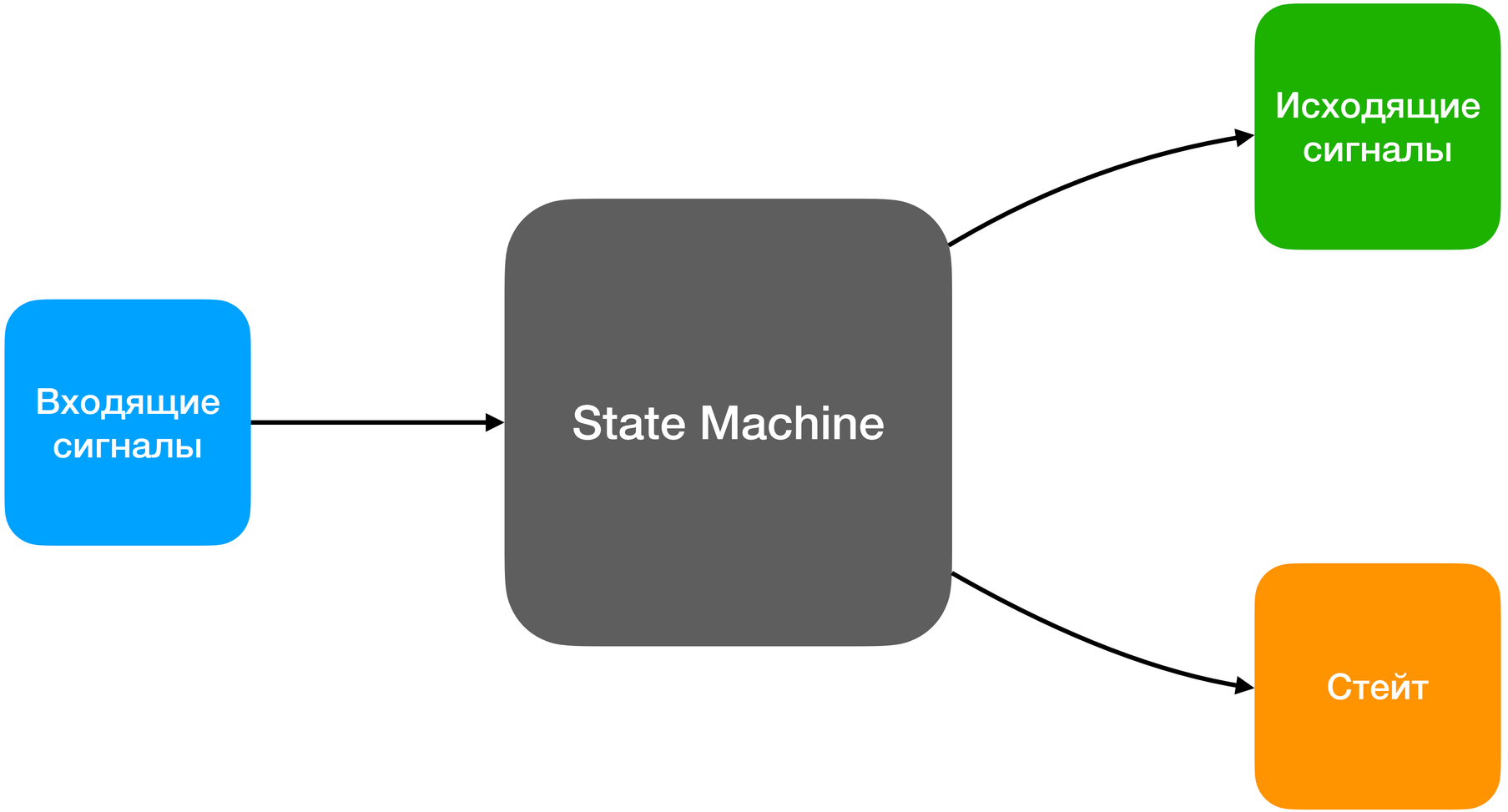
О чём сегодня будем говорить?

1. Что такое стейт-машина
2. Анализ реализаций стейт-машин
3. Схема работы MVI
4. Пример реализации с кодом

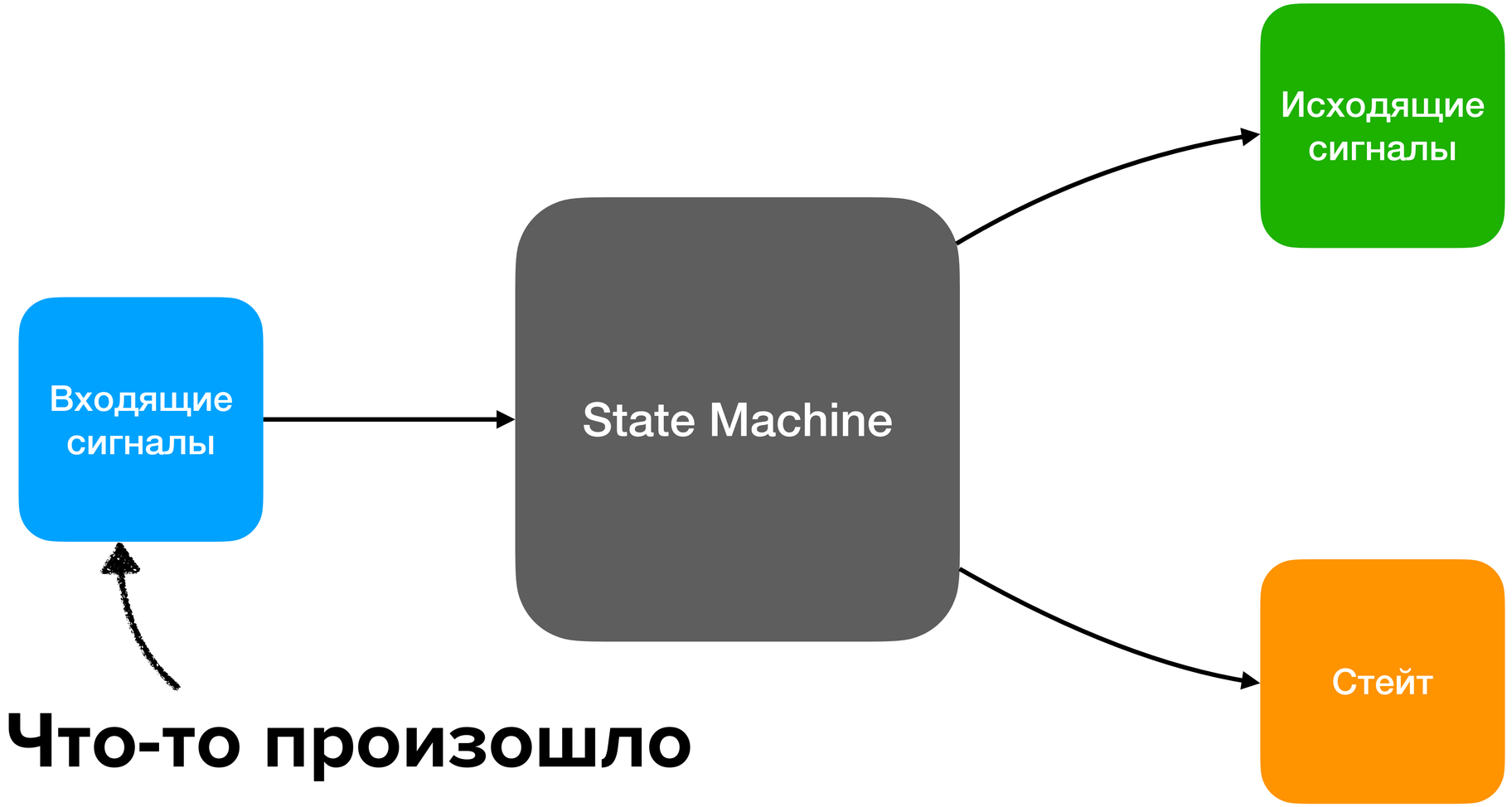
Ссылка на слайды


Что такое
State Machine?

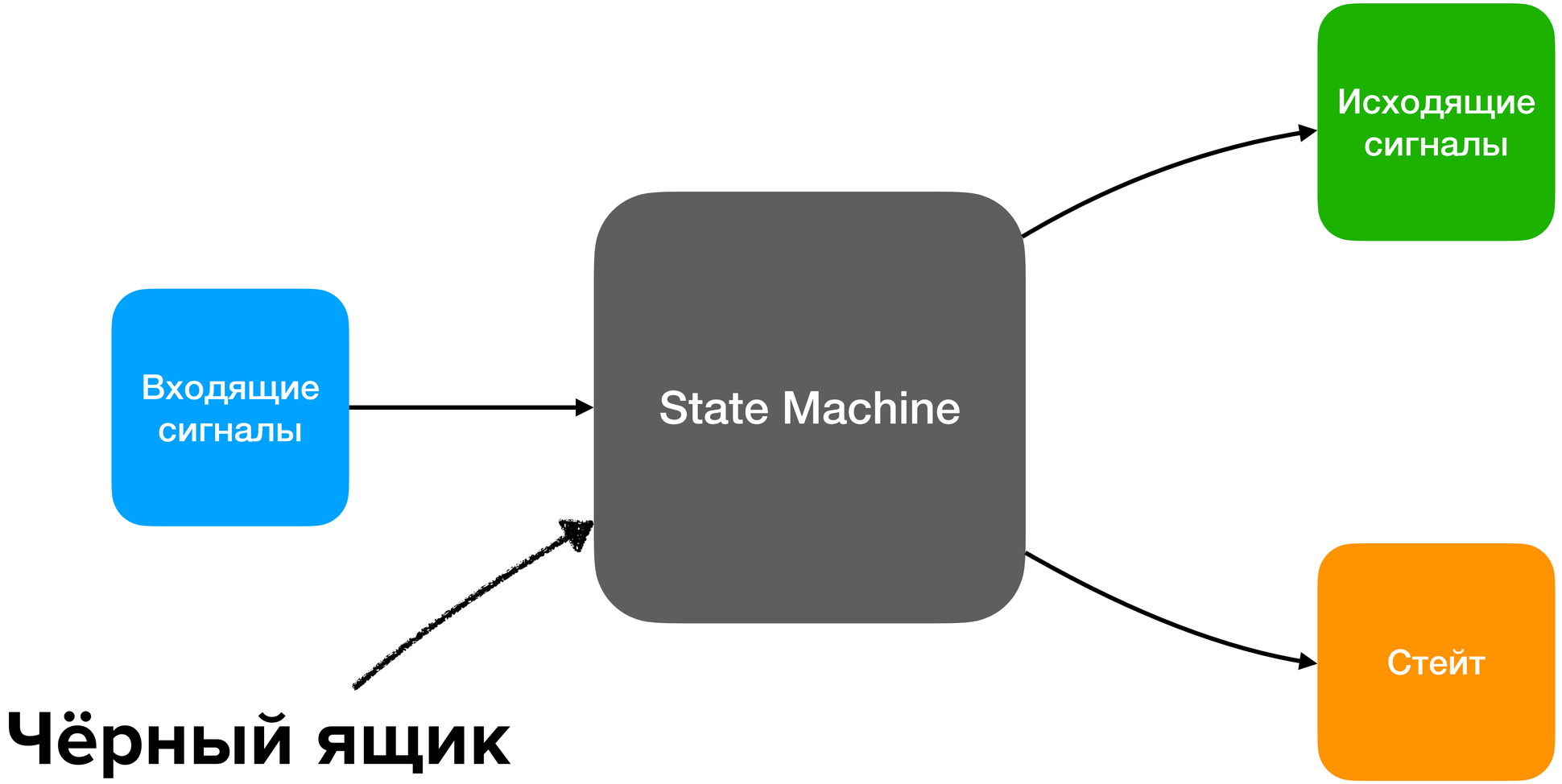
Типичная State Machine



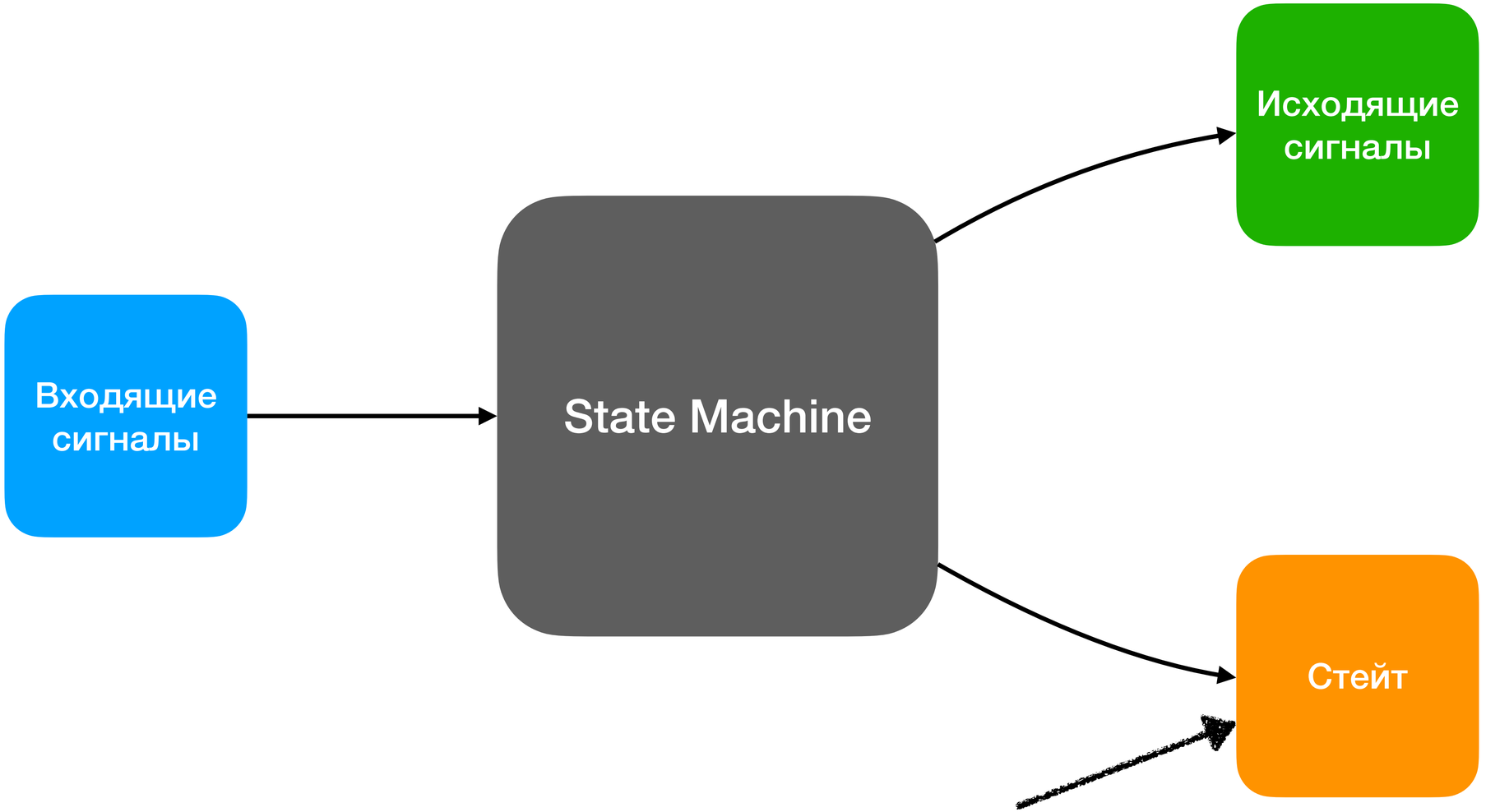
Типичная State Machine



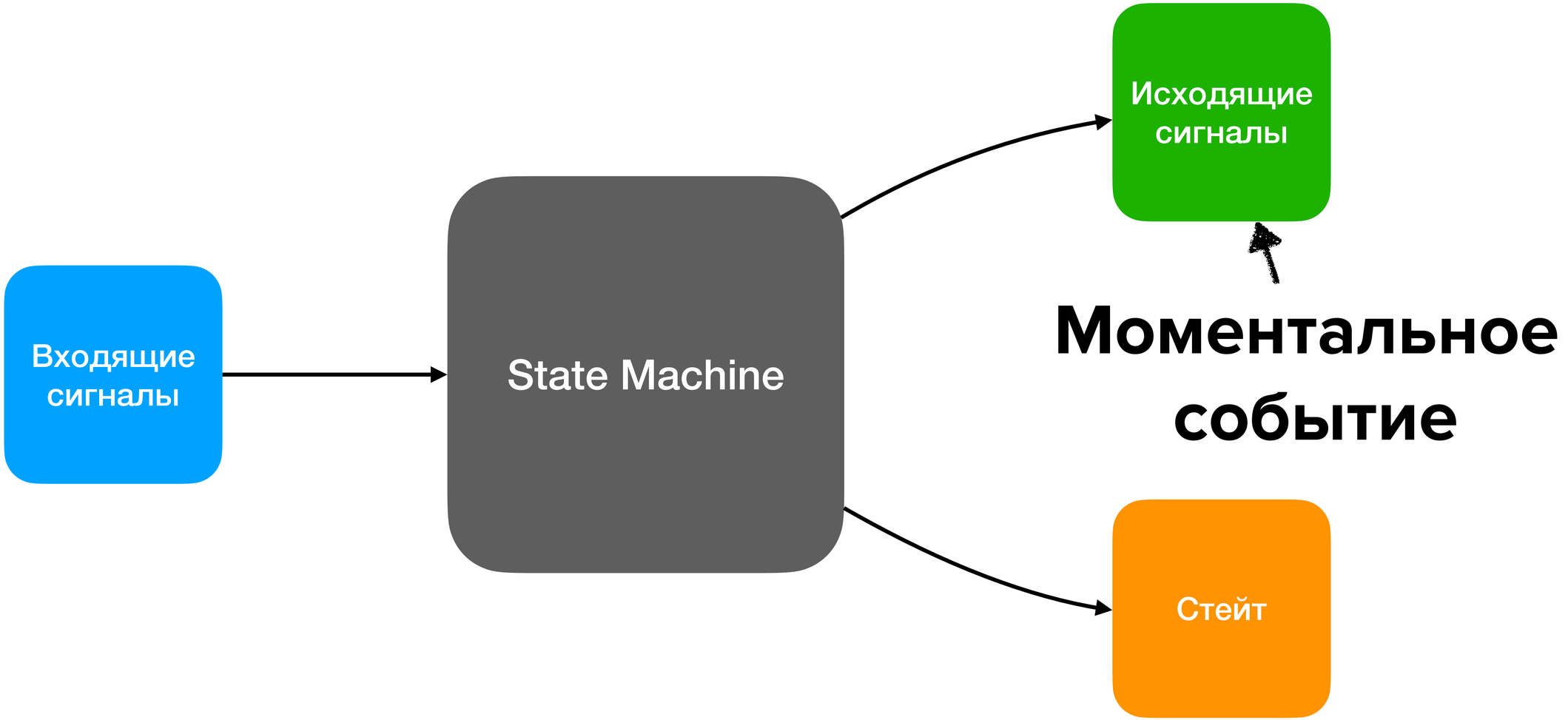
Типичная State Machine



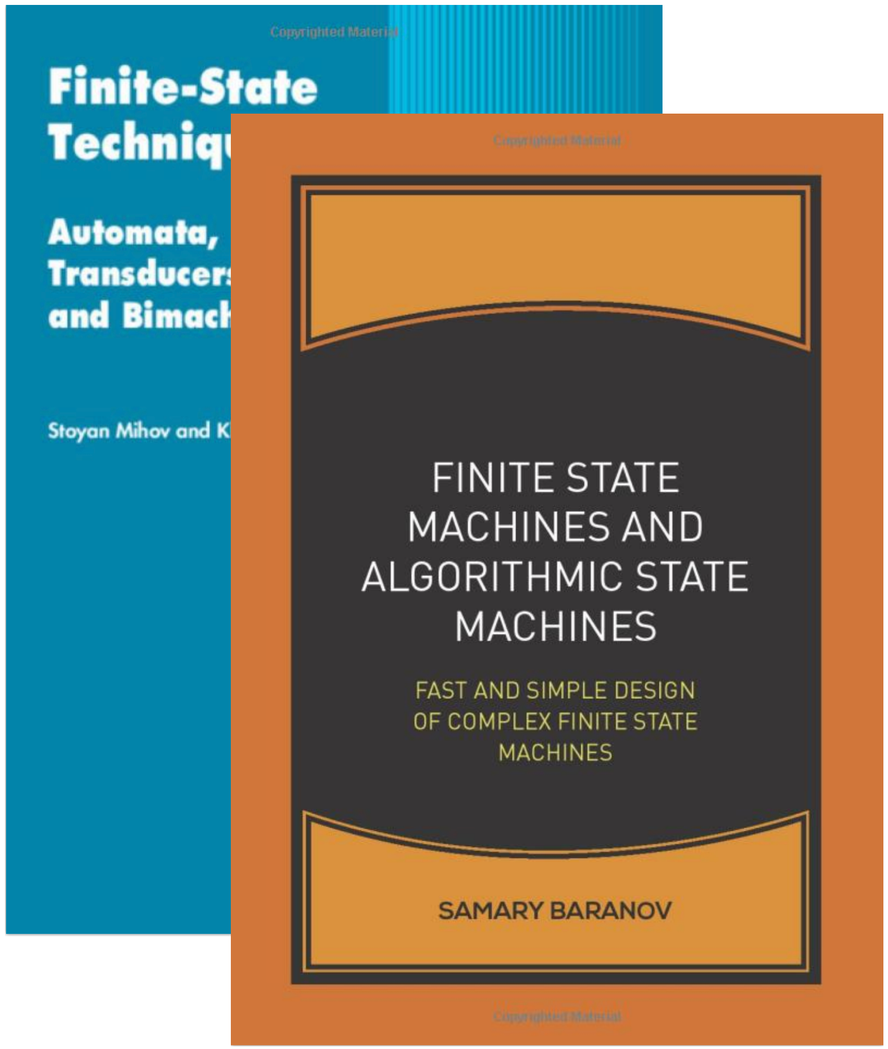
Типичная State Machine



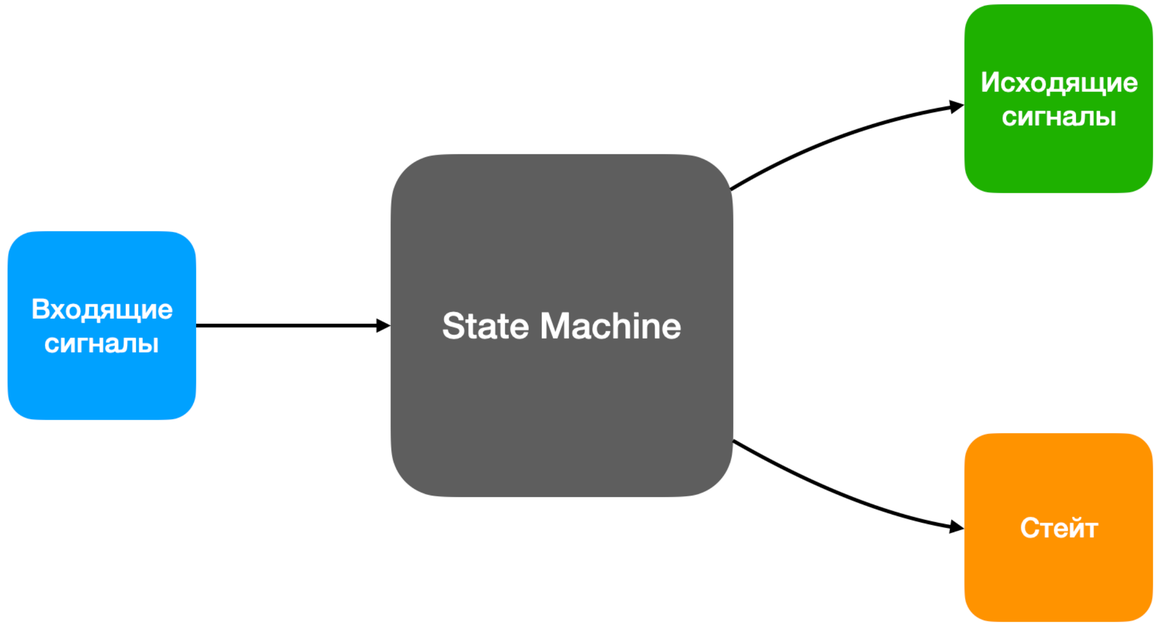
Типичная State Machine



Не сегодня

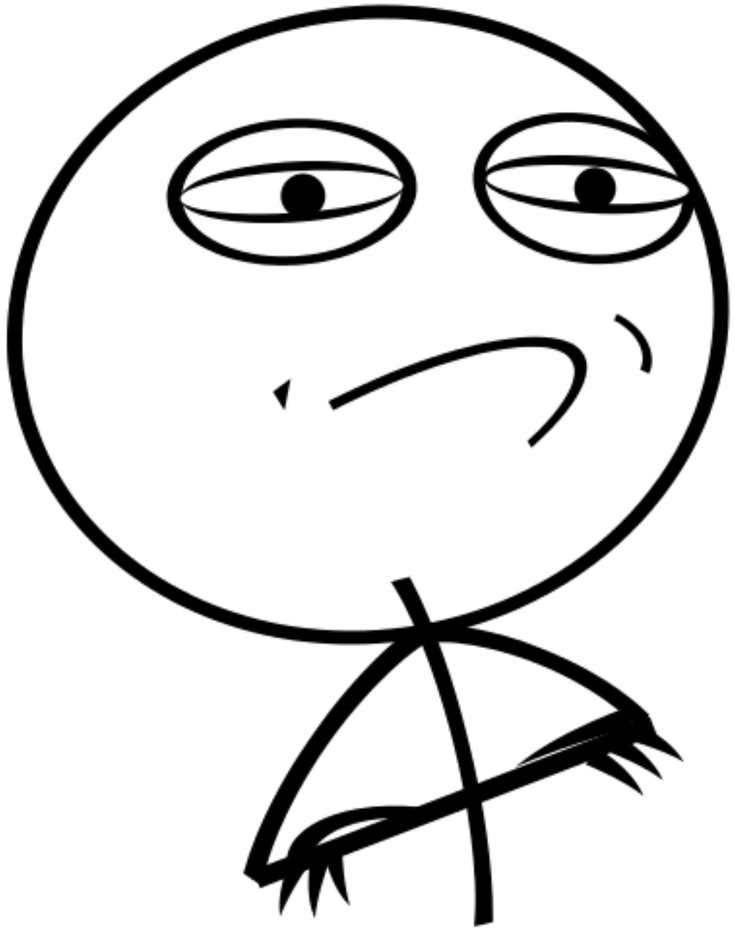


- Построение UML-диаграмм
- Конечные автоматы Мили и Мура



https://en.wikipedia.org/wiki/Finite-state_machine

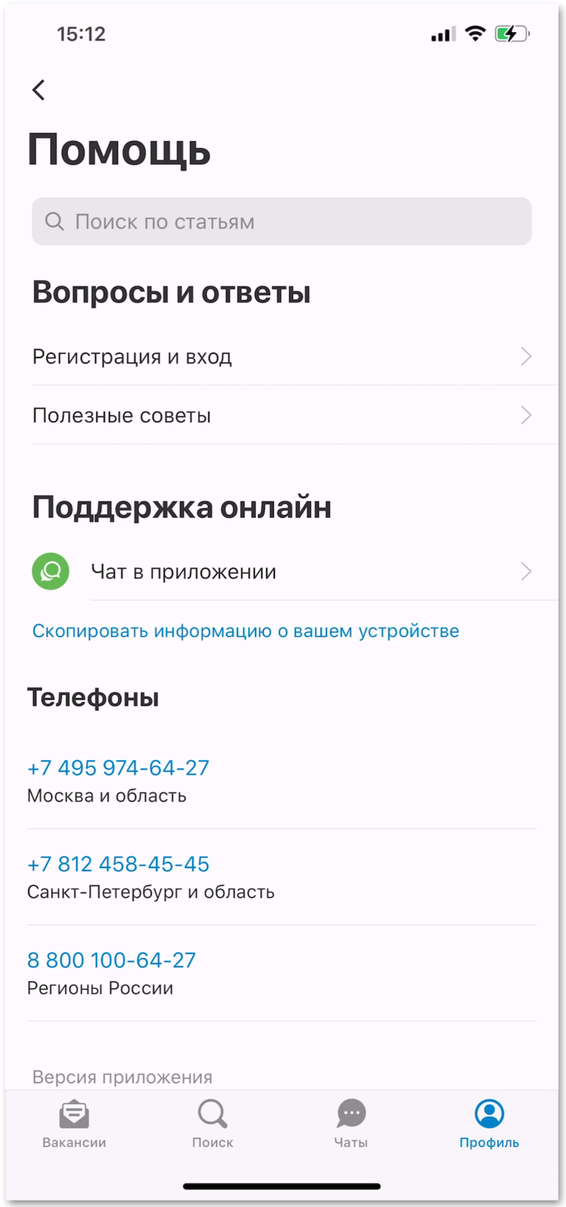
А давай сразу пример?



Опишем

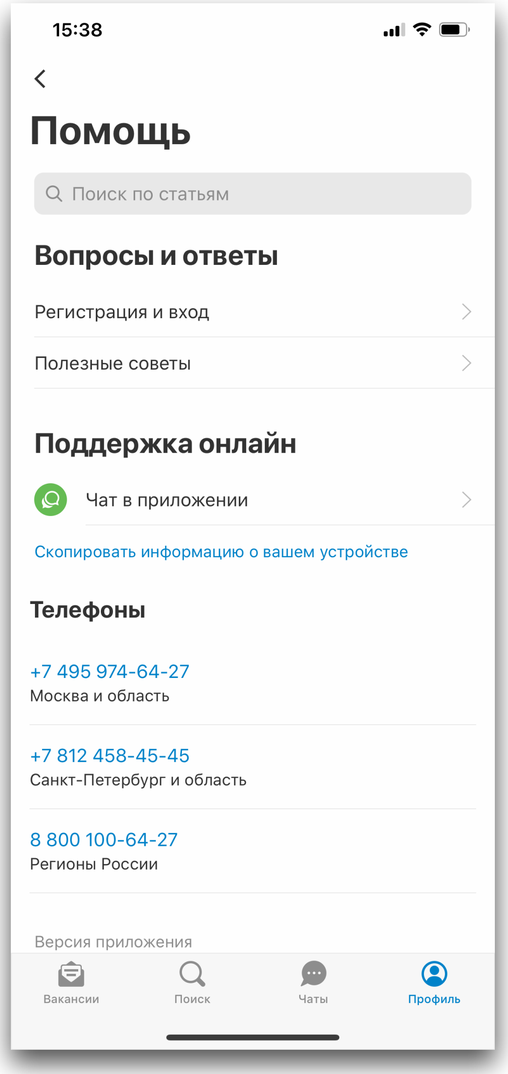
CallTracker

CallTracker



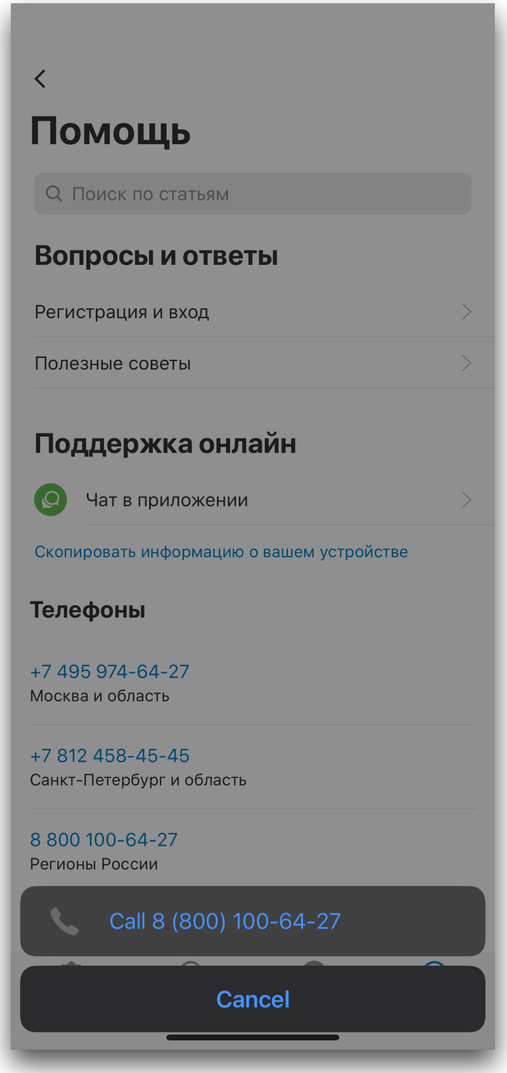
Звонки в iOS

1. **Вызываем URL вида tel://**
2. Система спрашивает подтверждение
3. Возврат в приложение
4. Начало звонка
5. Возврат в приложение



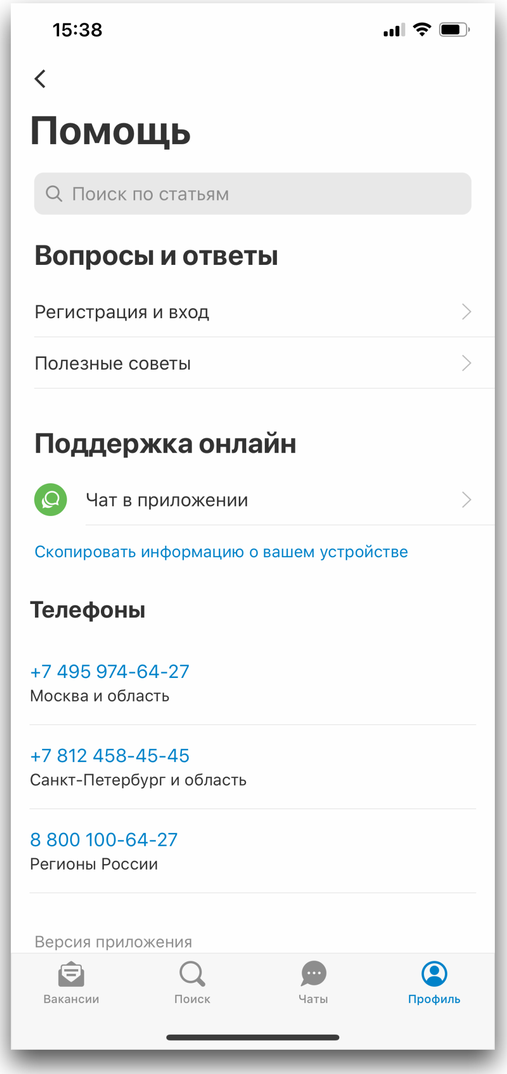
Звонки в iOS

1. Вызываем URL вида tel://
2. **Система спрашивает подтверждение**
3. Возврат в приложение
4. Начало звонка
5. Возврат в приложение



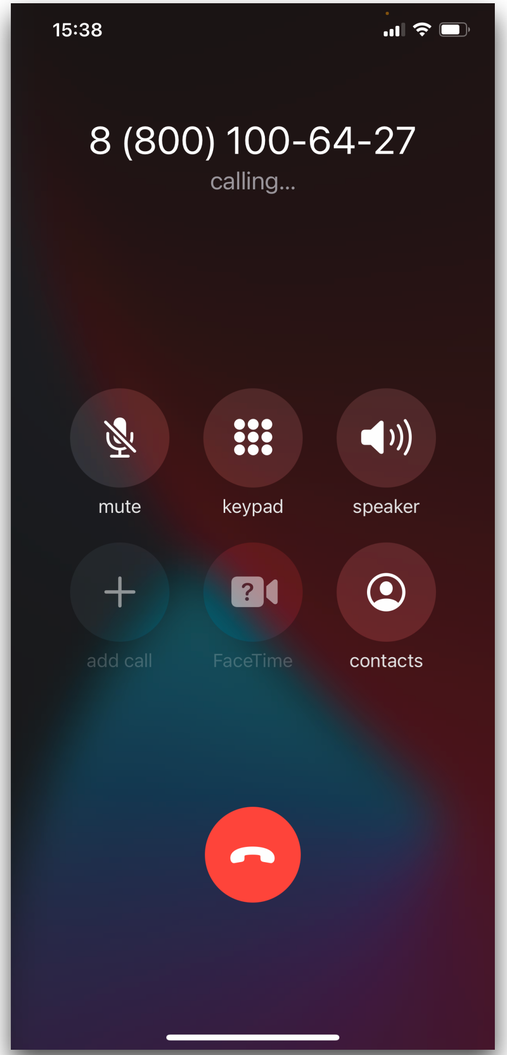
Звонки в iOS

1. Вызываем URL вида tel://
2. Система спрашивает подтверждение
3. **Возврат в приложение**
4. Начало звонка
5. Возврат в приложение



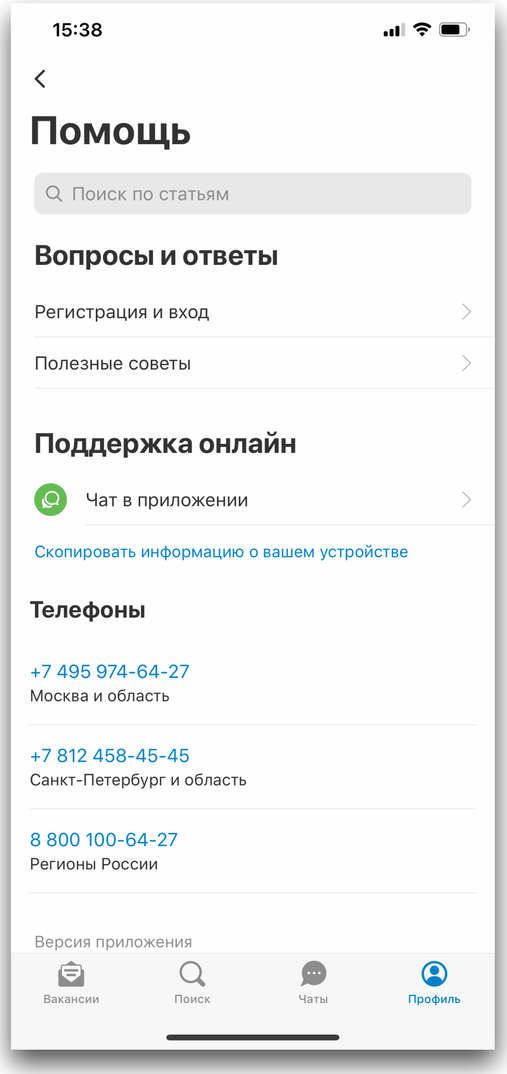
Звонки в iOS

1. Вызываем URL вида tel://
2. Система спрашивает подтверждение
3. Возврат в приложение
4. **Начало звонка**
5. Возврат в приложение

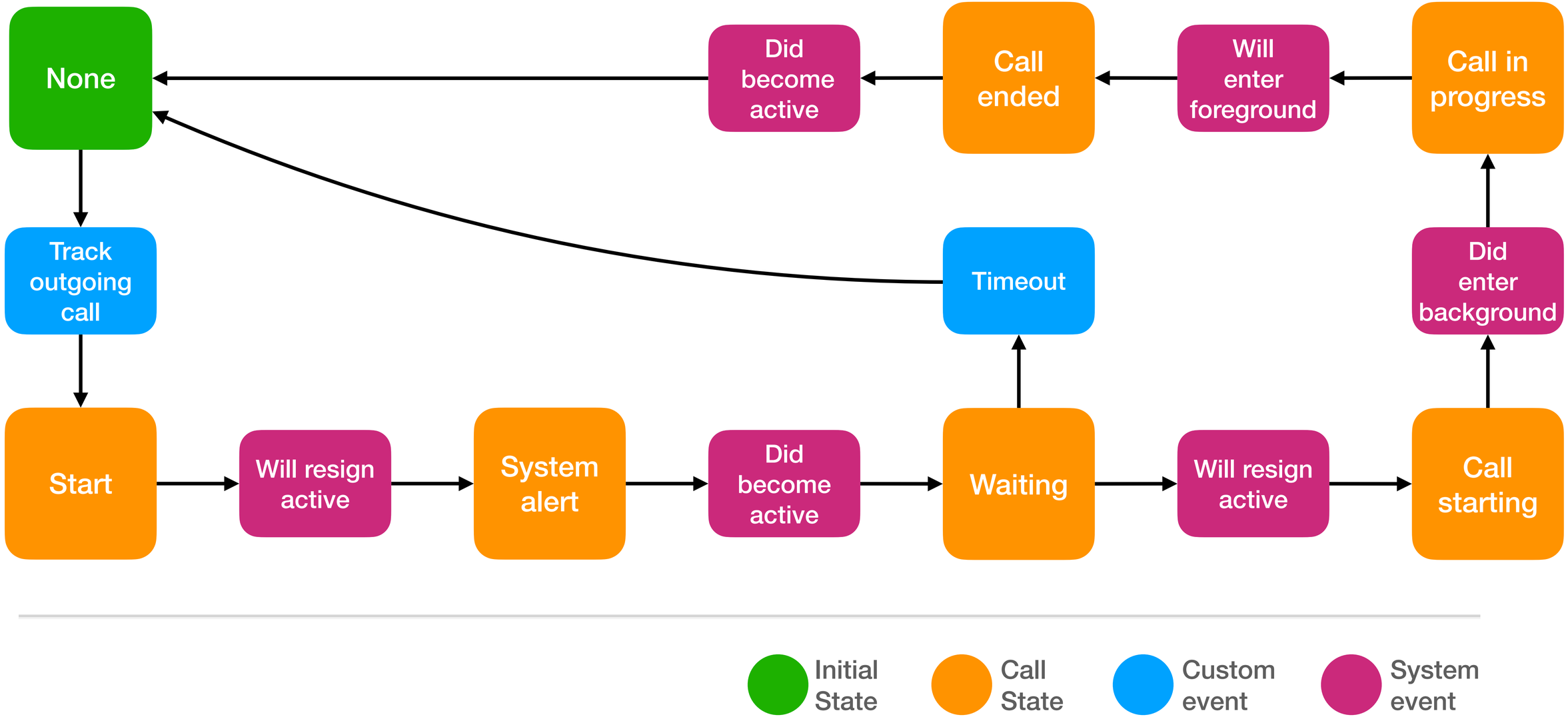


Звонки в iOS

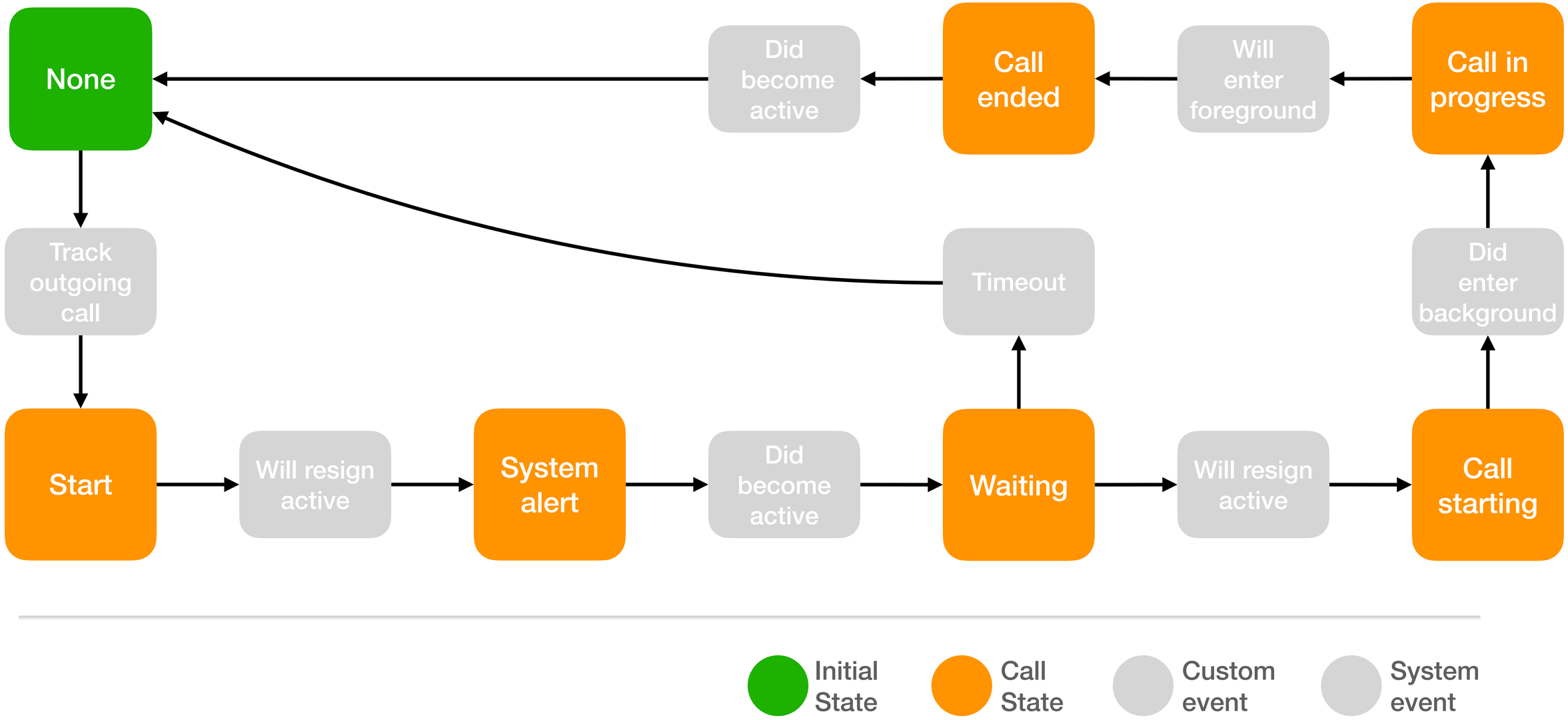
1. Вызываем URL вида tel://
2. Система спрашивает подтверждение
3. Возврат в приложение
4. Начало звонка
5. **Возврат в приложение**



CallTracker

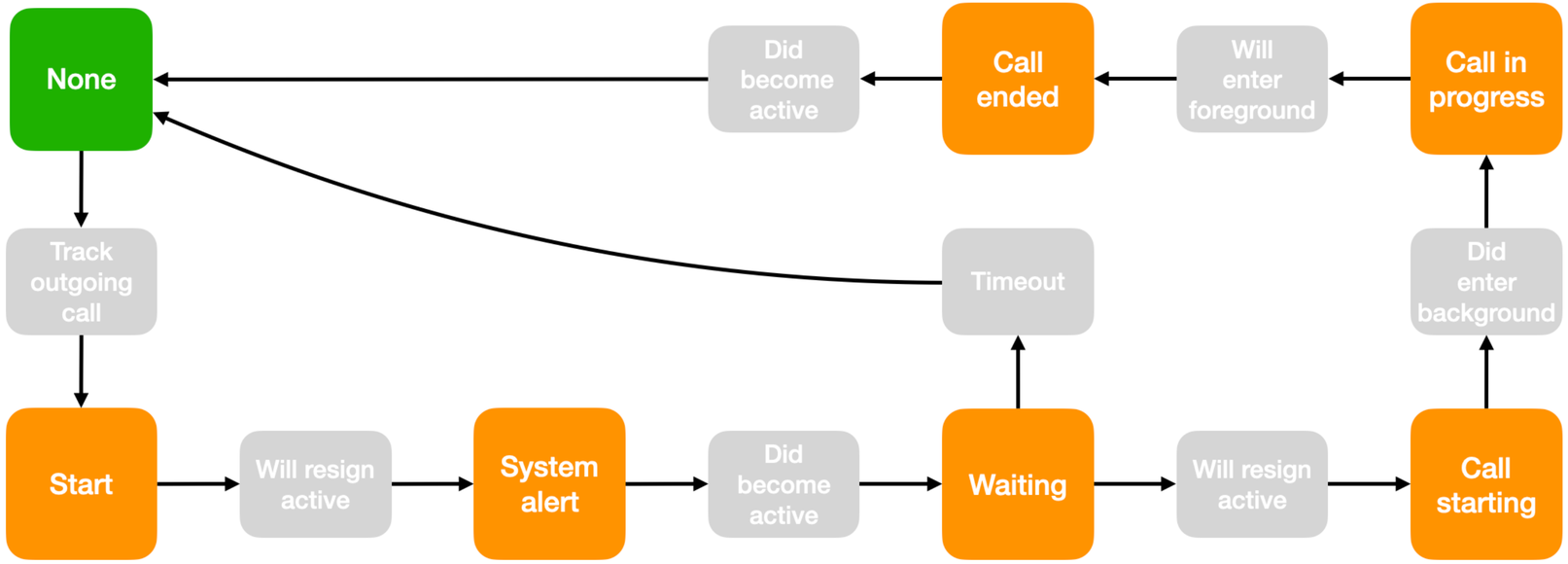


Выделяем состояния

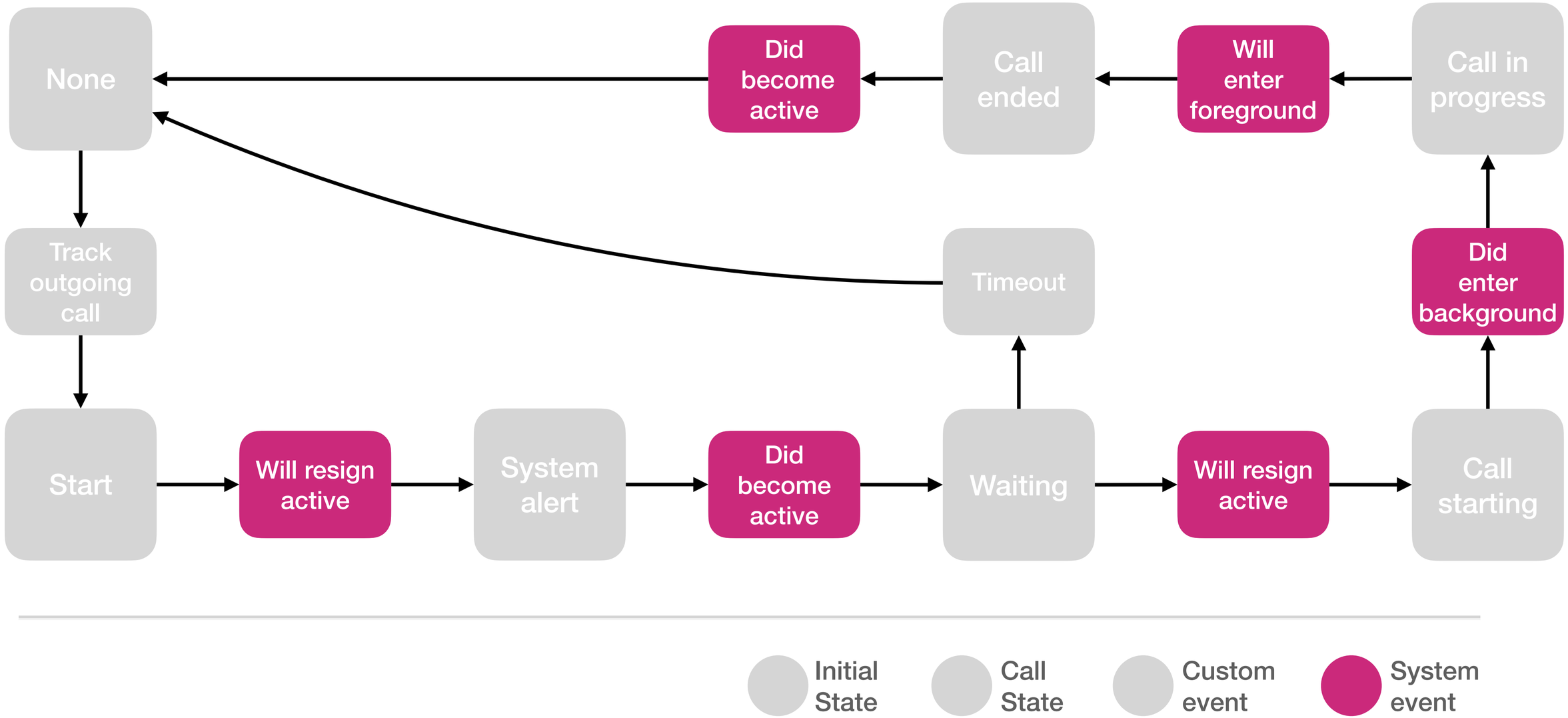


Их легко формализовать

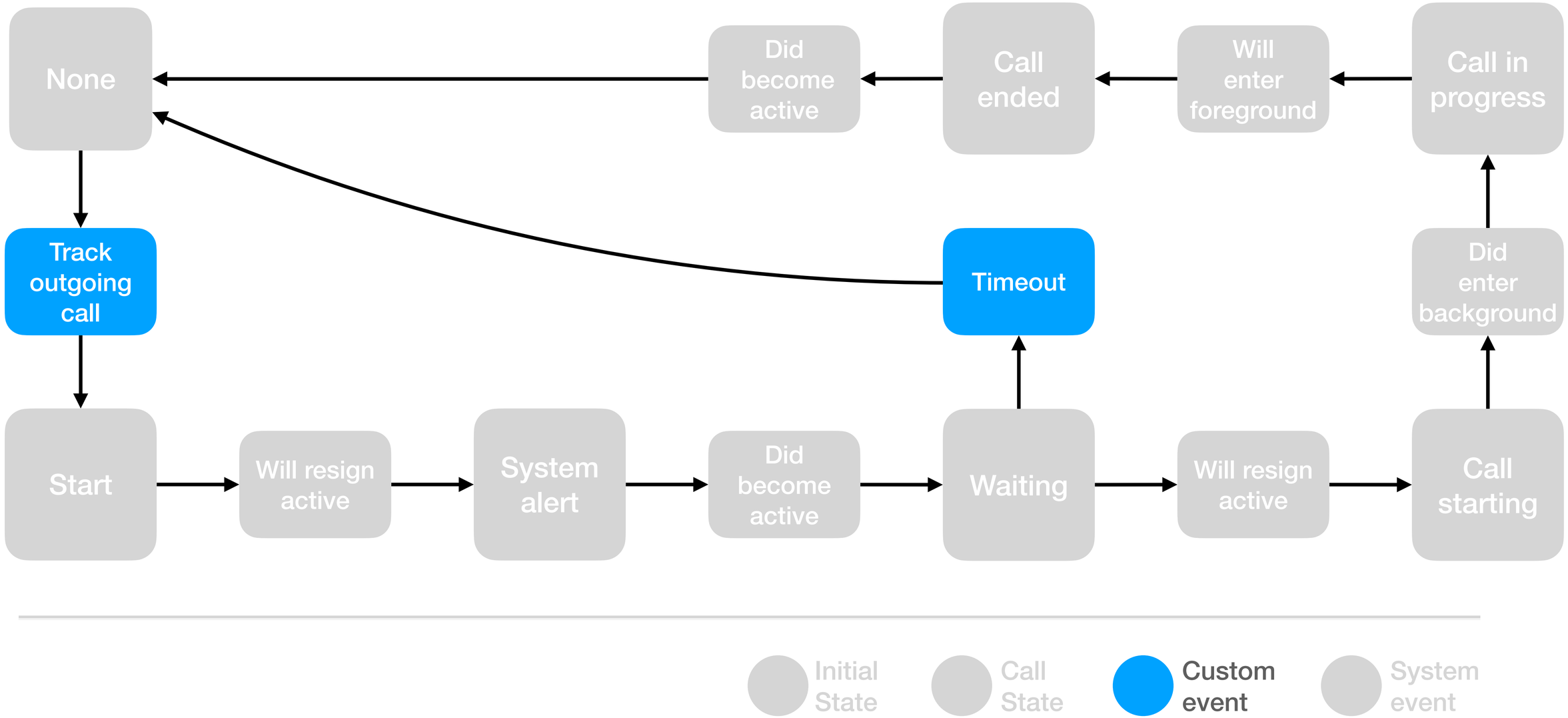
```
enum OutgoingCallState {
    case none
    case start
    case systemAlert
    case waiting
    case callStarting
    case callInProgress
    case callEnded
}
```



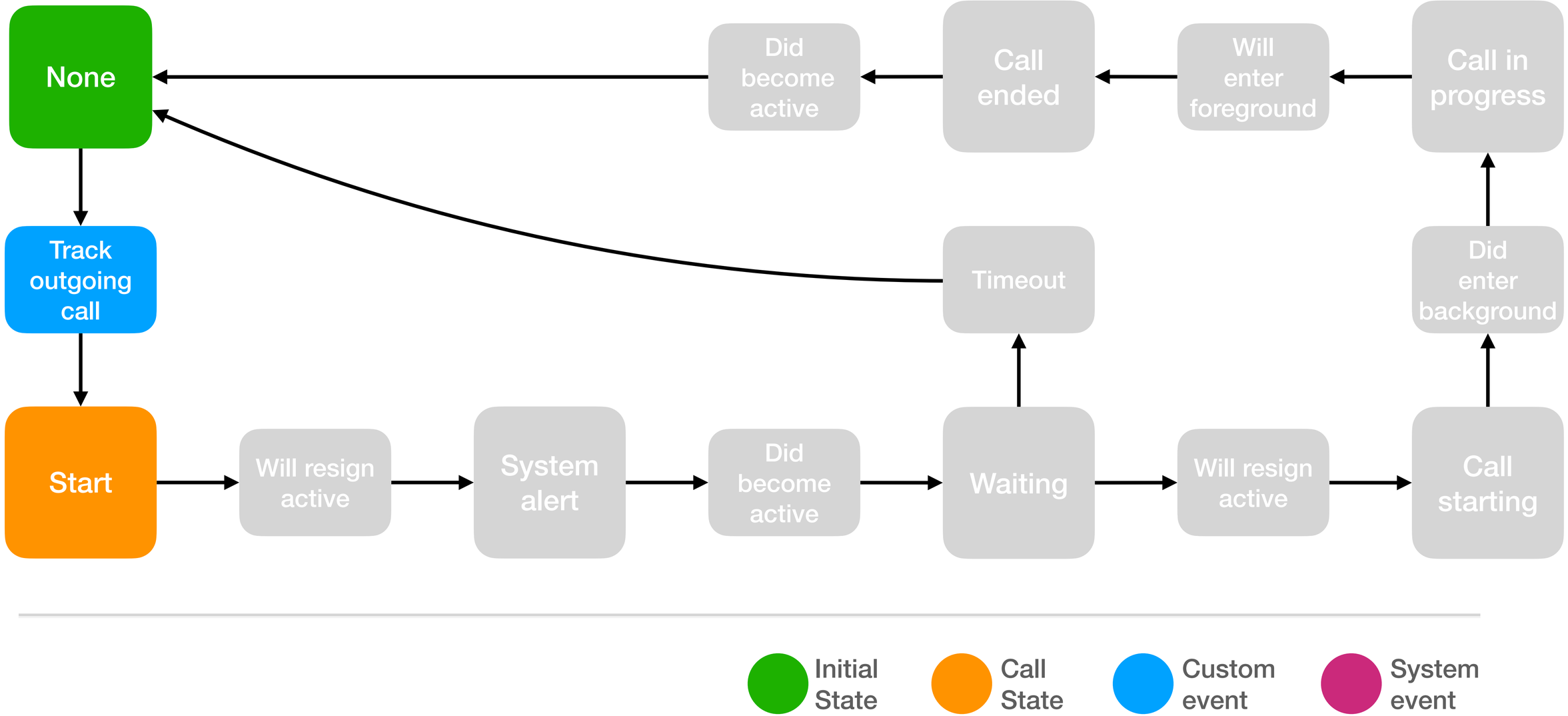
События ЖЦ системы



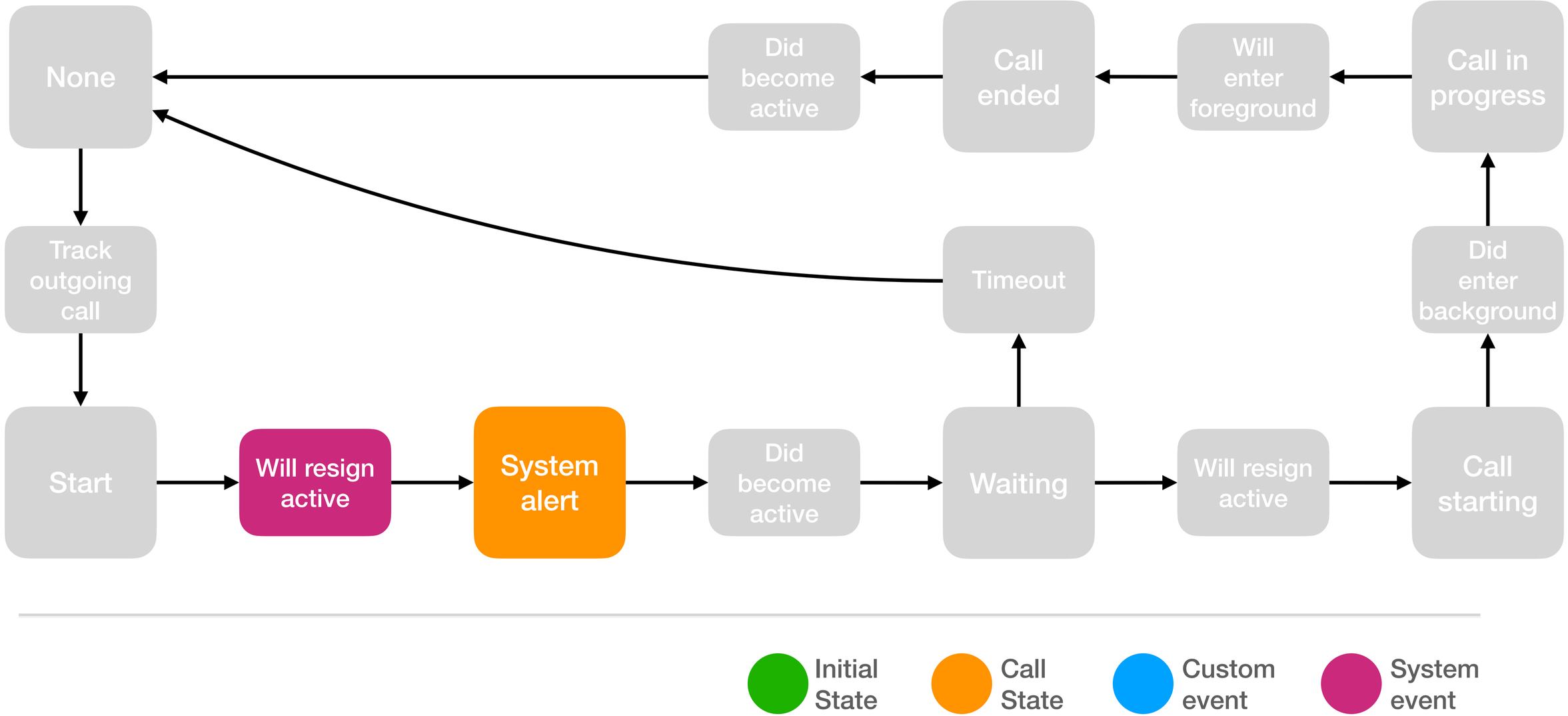
Пользовательские события



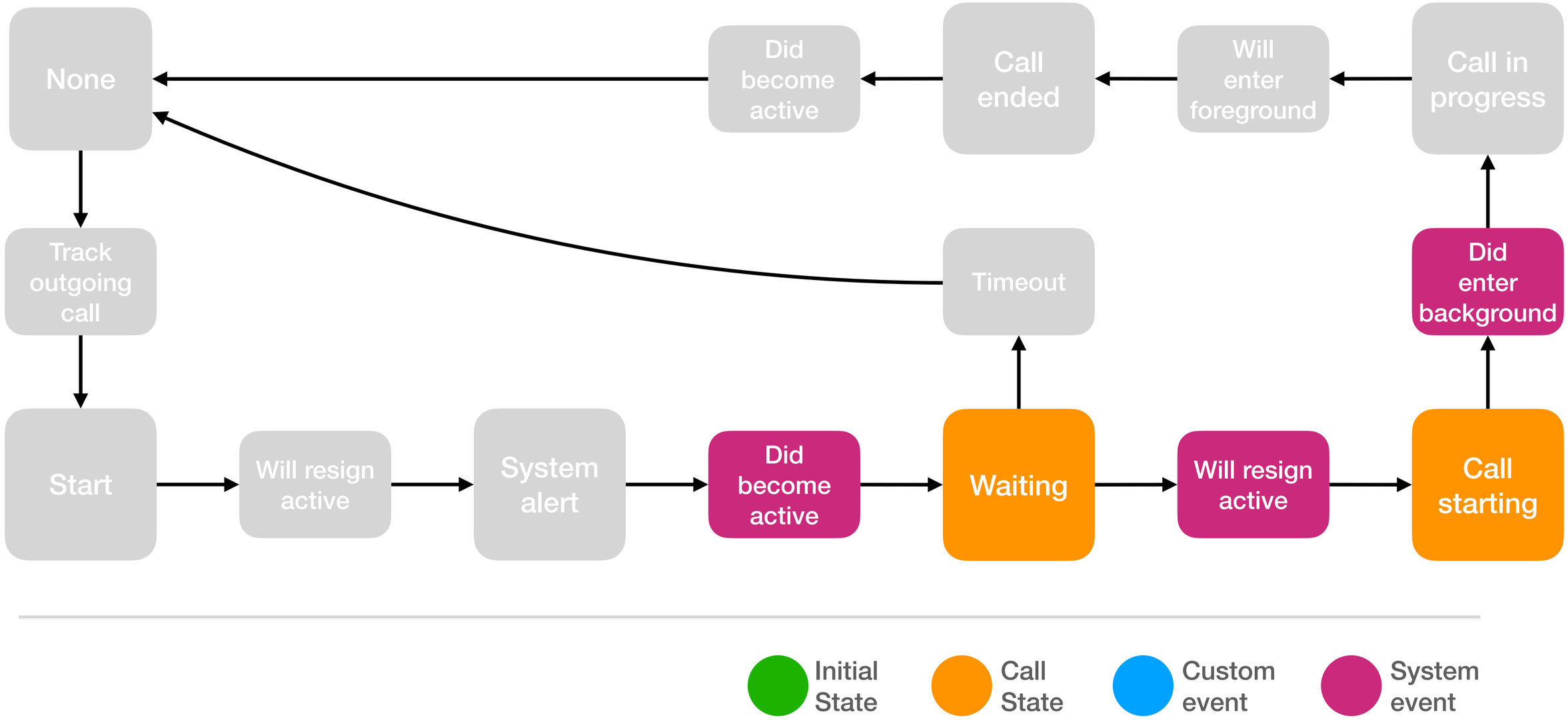
Жмем на телефончик



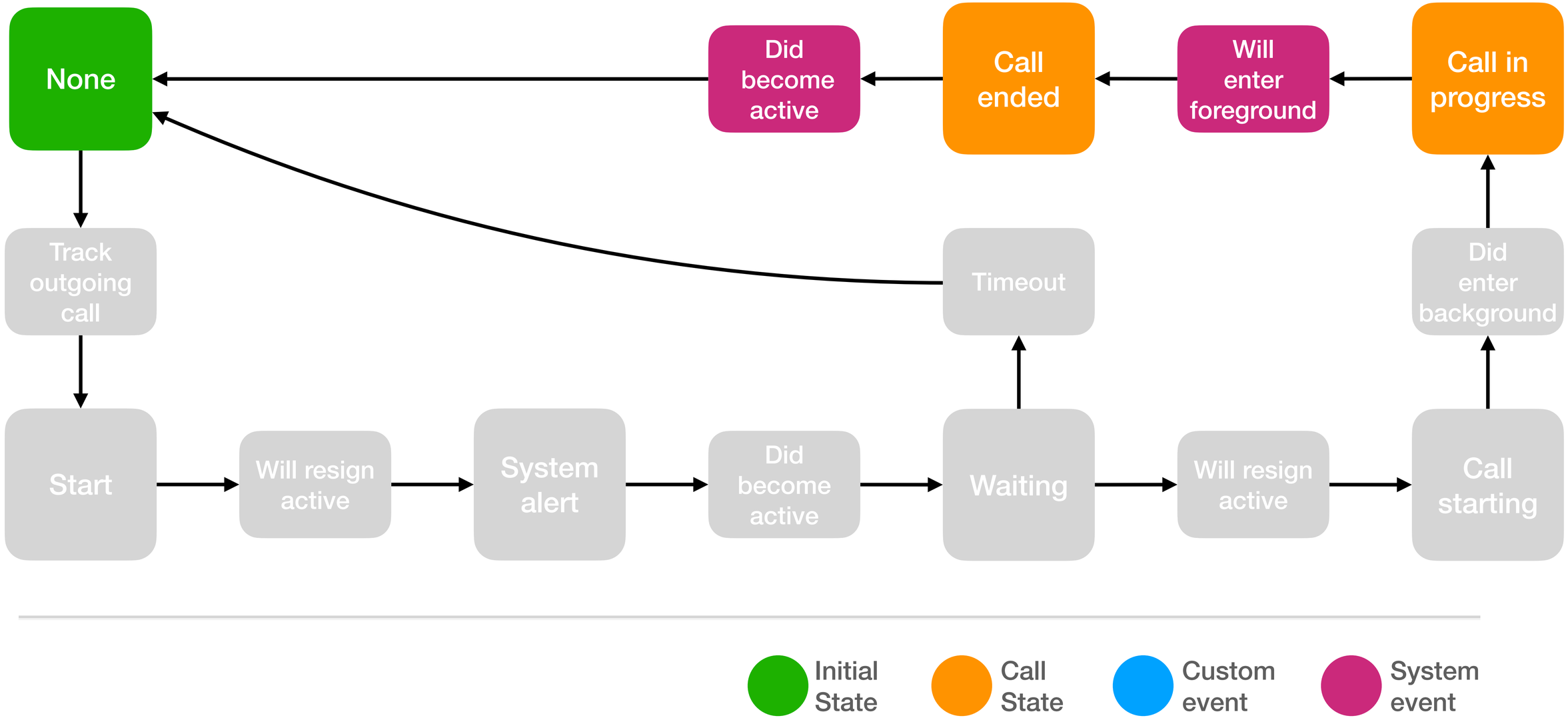
iOS переспрашивает



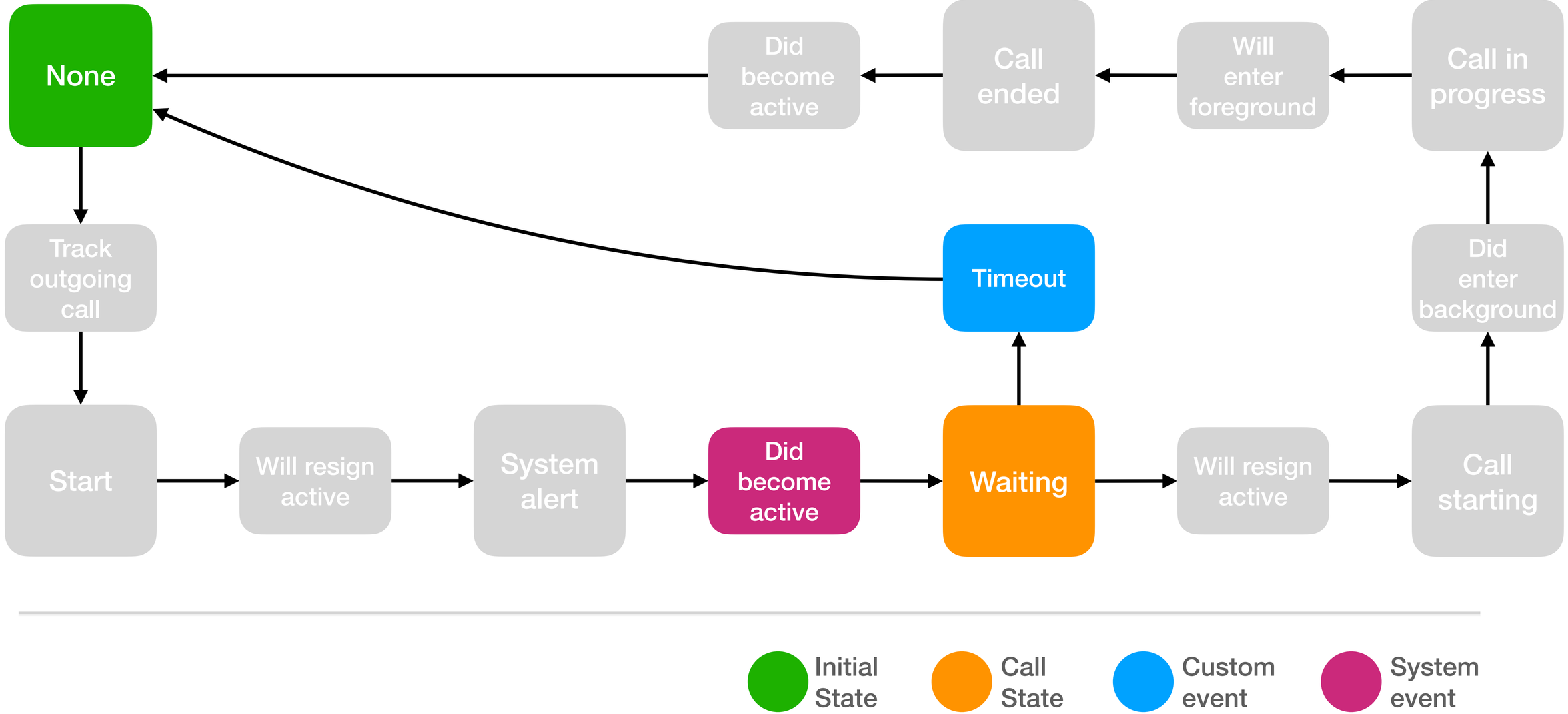
Звонок начинается



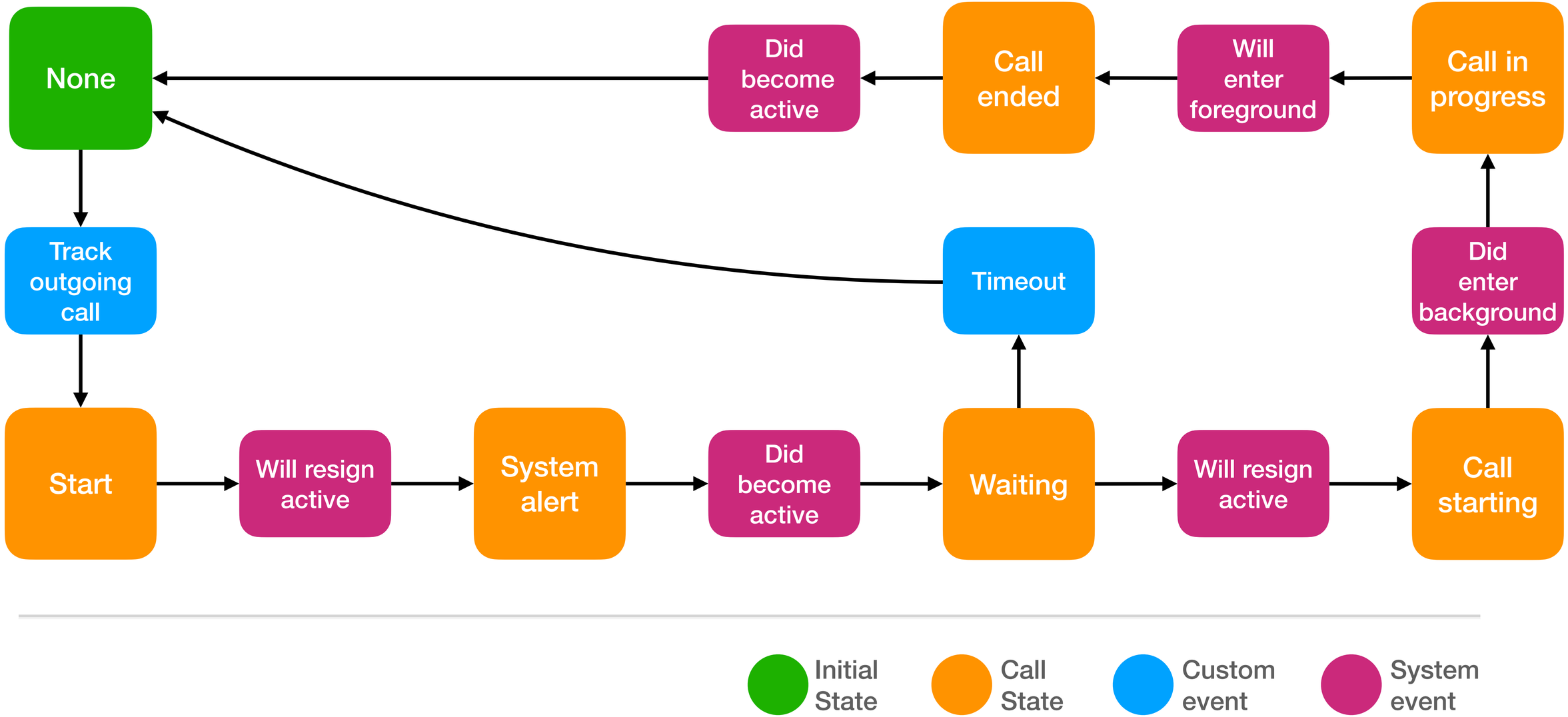
Успешно поговорили



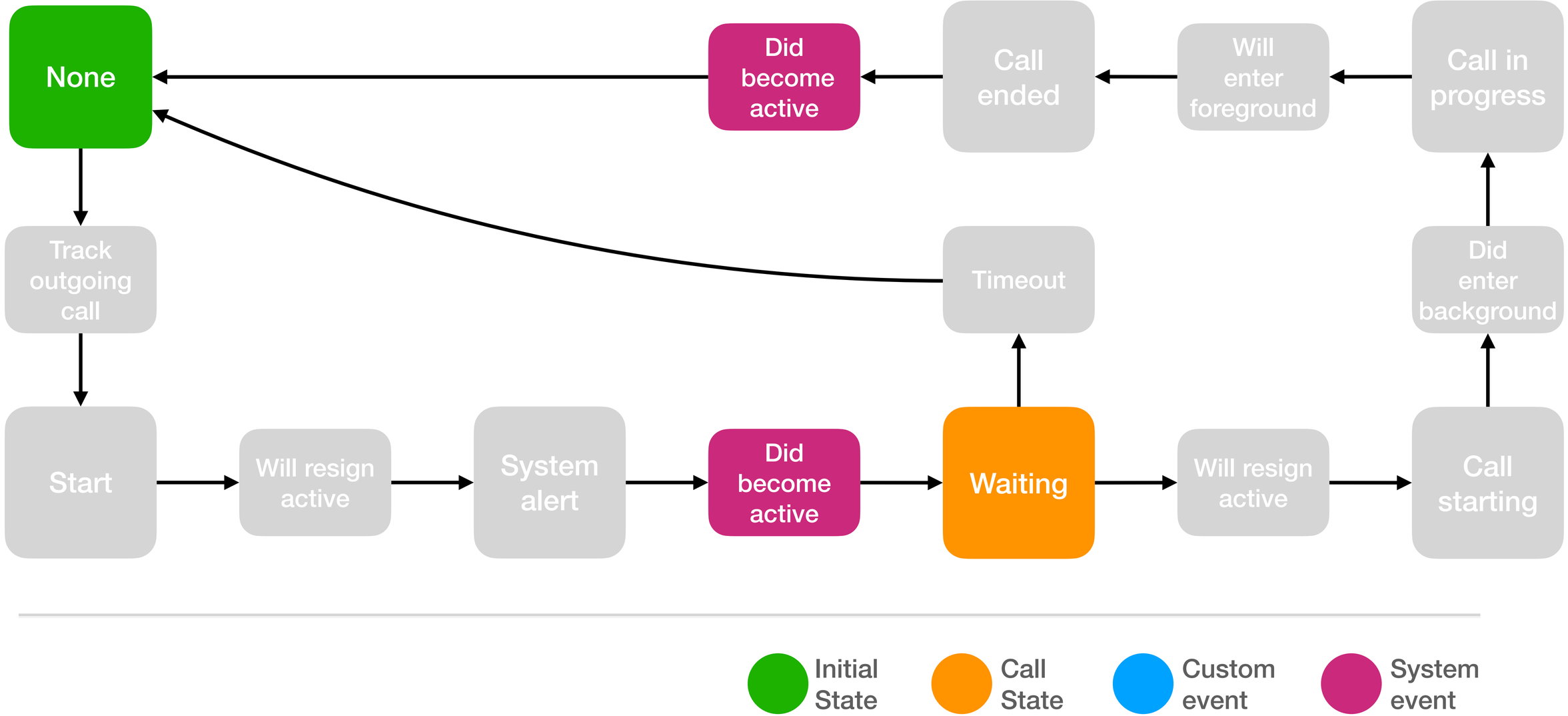
Пользователь отменил



CallTracker



Двойной didBecomeActive



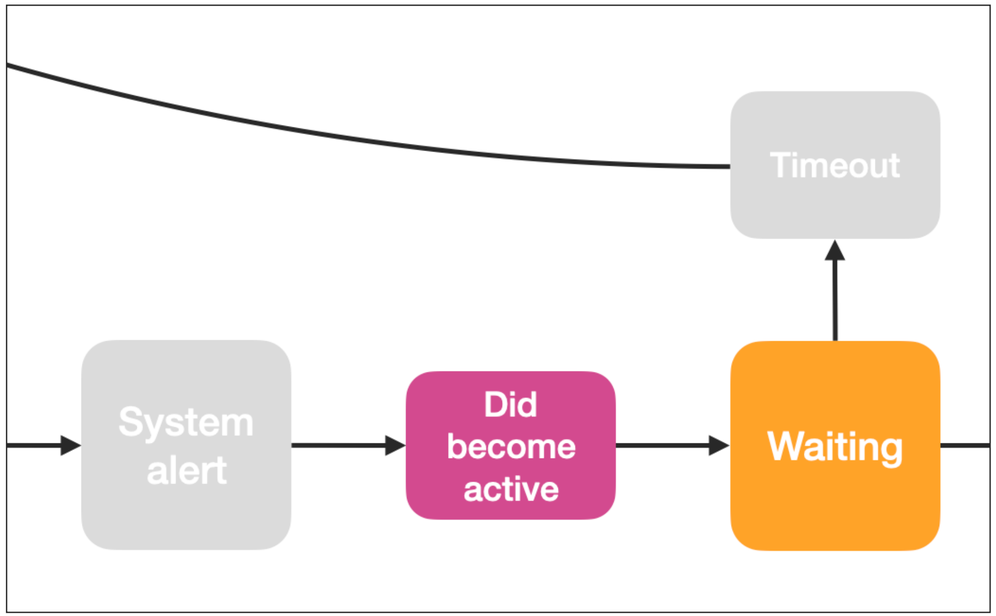
Ждем начало звонка системой

```
private func applicationDidBecomeActiveHandler() {
    switch state {
    case .systemAlert:
        startWaitingTimer()
        state = .waiting

    case .callEnded: // трекаем окончание звонка при запущенном приложении
        state = .none
        trackOutgoingCallEndIfPossible()

    case .none: // штатная ситуация
        break

    case .start,
         .waiting,
         .callStarting,
         .callInProgress:
       state = .none
    }
}
```



Поговорили

```
private func applicationDidBecomeActiveHandler() {
    switch state {
    case .systemAlert:
        startWaitingTimer()
        state = .waiting

    case .callEnded: // трекаем окончание звонка при запущенном приложении
        state = .none
        trackOutgoingCallEndIfPossible()

    case .none: // штатная ситуация
        break

    case .start,
         .waiting,
         .callStarting,
         .callInProgress:
       state = .none
    }
}
```

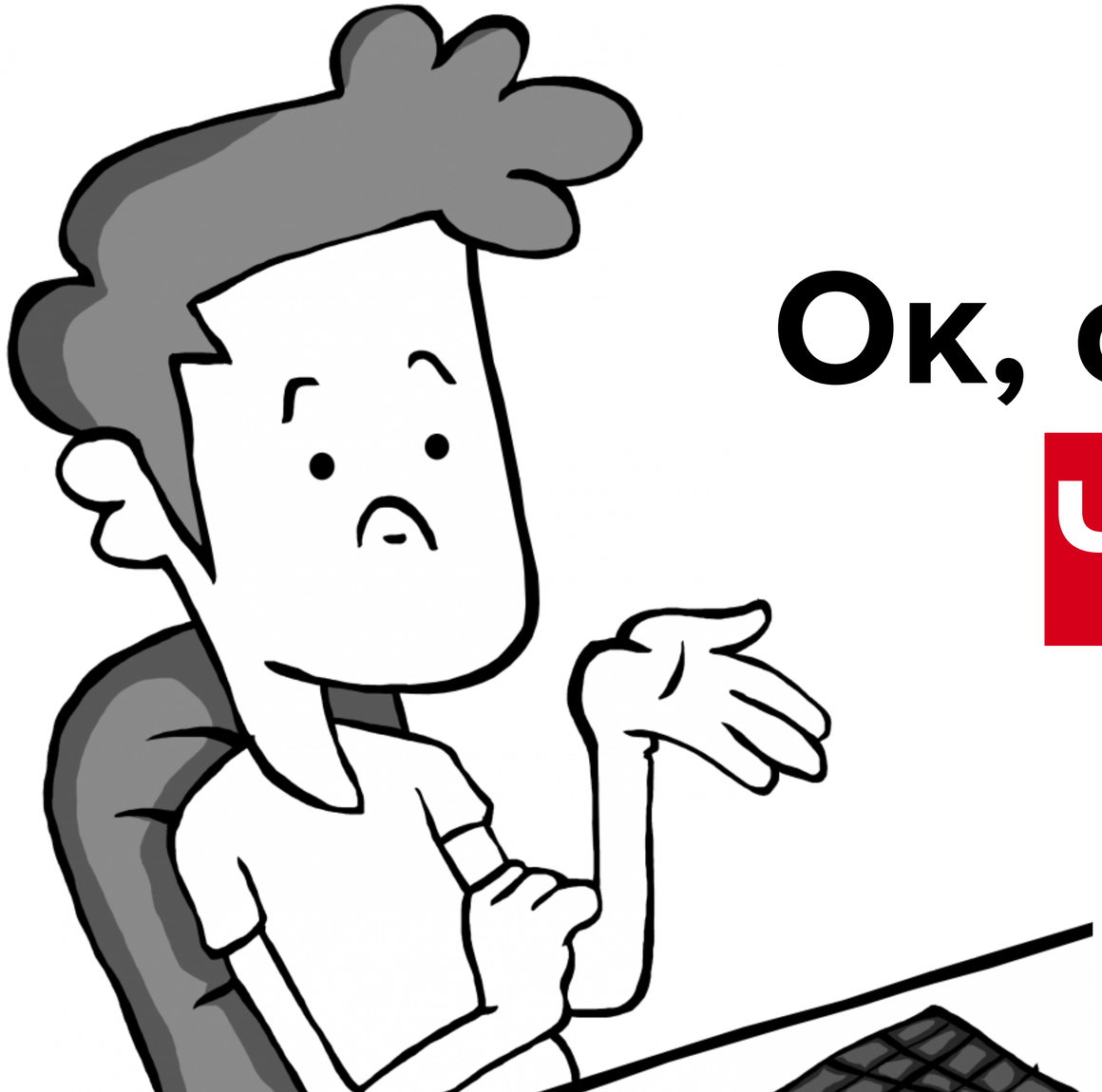
Что-то другое

```
private func applicationDidBecomeActiveHandler() {
    switch state {
    case .systemAlert:
        startWaitingTimer()
        state = .waiting

    case .callEnded: // трекаем окончание звонка при запущенном приложении
        state = .none
        trackOutgoingCallEndIfPossible()

    case .none: // штатная ситуация
        break

    case .start,
         .waiting,
         .callStarting,
         .callInProgress:
       state = .none
    }
}
```



Ок, стейты — круто.

Что дальше?


Анализируем
подходы

Критерии оценки

1. **Взаимодействие между фичами**
2. Исходящие сигналы
3. Итеративное внедрение
4. Схожесть с Android-реализацией
5. Роутинг
6. Поддержка SwiftUI

Критерии оценки

1. Взаимодействие между фичами
2. **Исходящие сигналы**
3. Итеративное внедрение
4. Схожесть с Android-реализацией
5. Роутинг
6. Поддержка SwiftUI

Критерии оценки

1. Взаимодействие между фичами
2. Исходящие сигналы
3. **Итеративное внедрение**
4. Схожесть с Android-реализацией
5. Роутинг
6. Поддержка SwiftUI

Критерии оценки

1. Взаимодействие между фичами
2. Исходящие сигналы
3. Итеративное внедрение
4. **Схожесть с Android-реализацией**
5. Роутинг
6. Поддержка SwiftUI

Критерии оценки

1. Взаимодействие между фичами
2. Исходящие сигналы
3. Итеративное внедрение
4. Схожесть с Android-реализацией
5. **Роутинг**
6. Поддержка SwiftUI

https://github.com/hhru/Nivelir/

Критерии оценки

1. Взаимодействие между фичами
2. Исходящие сигналы
3. Итеративное внедрение
4. Схожесть с Android-реализацией
5. Роутинг
6. **Поддержка SwiftUI**

Какие варианты рассматривали?

1. MVI
2. ReactorKit
3. ReSwift (Redux)
4. The Composable Architecture

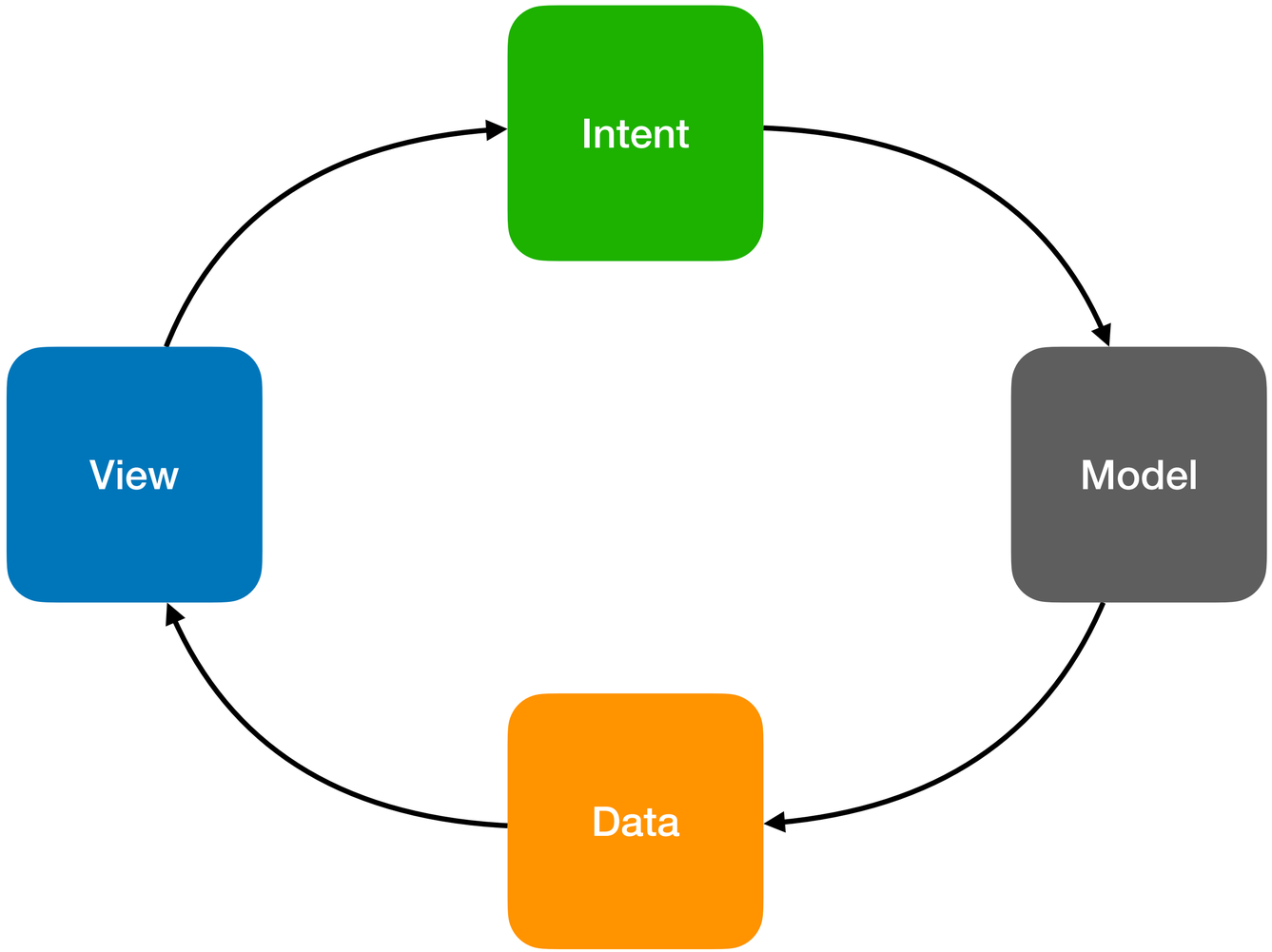
Таблица результатов

	Уже готово	Android	Взаимодействие между фичами	Исходящие сигналы	Итеративное внедрение	Роутинг
MVI	-	+	+	news	не навязывает архитектуру приложения	через сигналы
ReactorKit	+/- RxSwift	+/-	+	-	не навязывает глобальный стейт	за пределами фреймворка
ReSwift (Redux)	+	-	единый стор	-	-	отдельный стейт для роутинга
TCA	+	-	выделение скоупа, композиция	-	-	открытие экрана это стейт

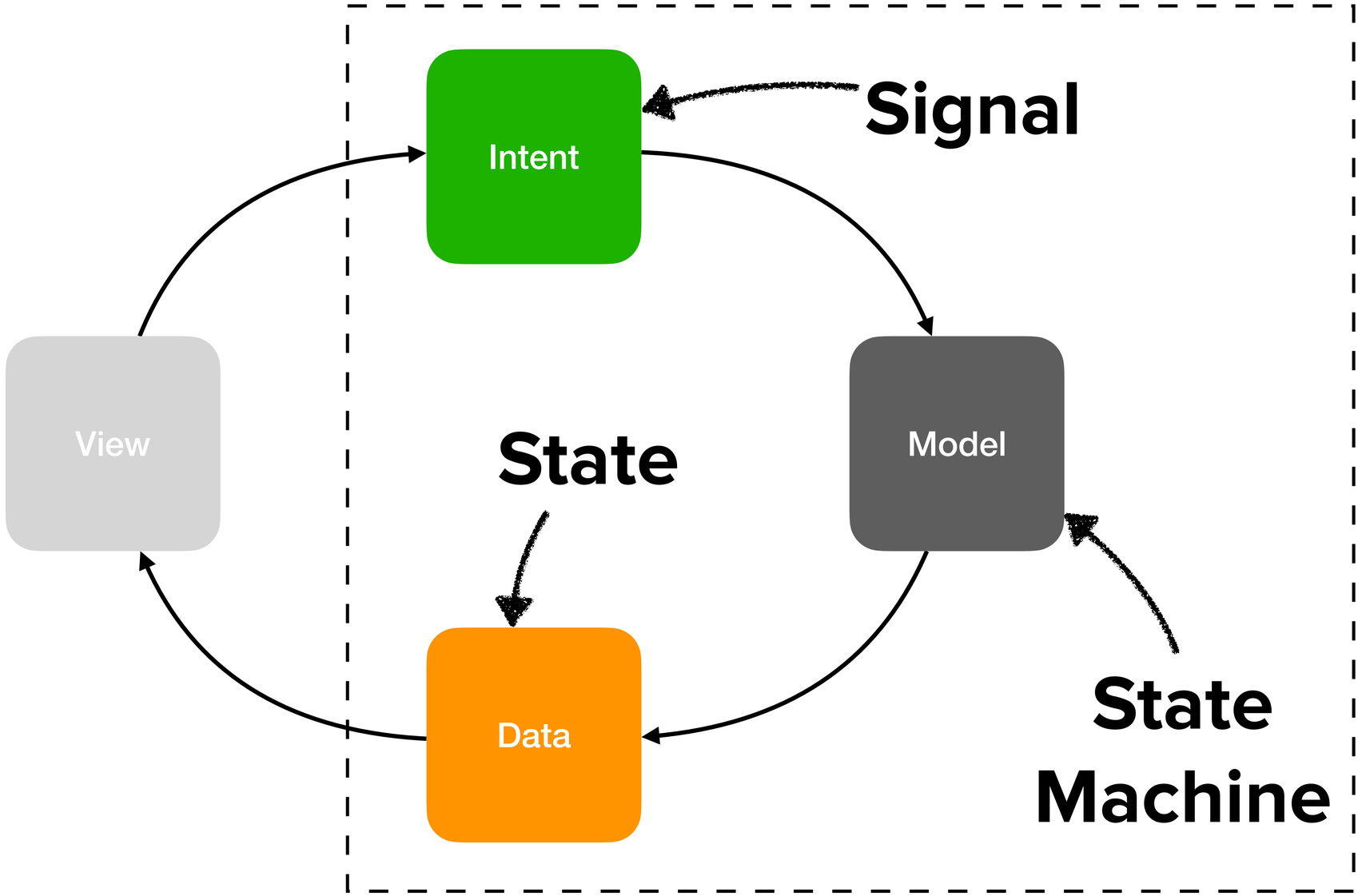
Наш выбор

архитектура MVI

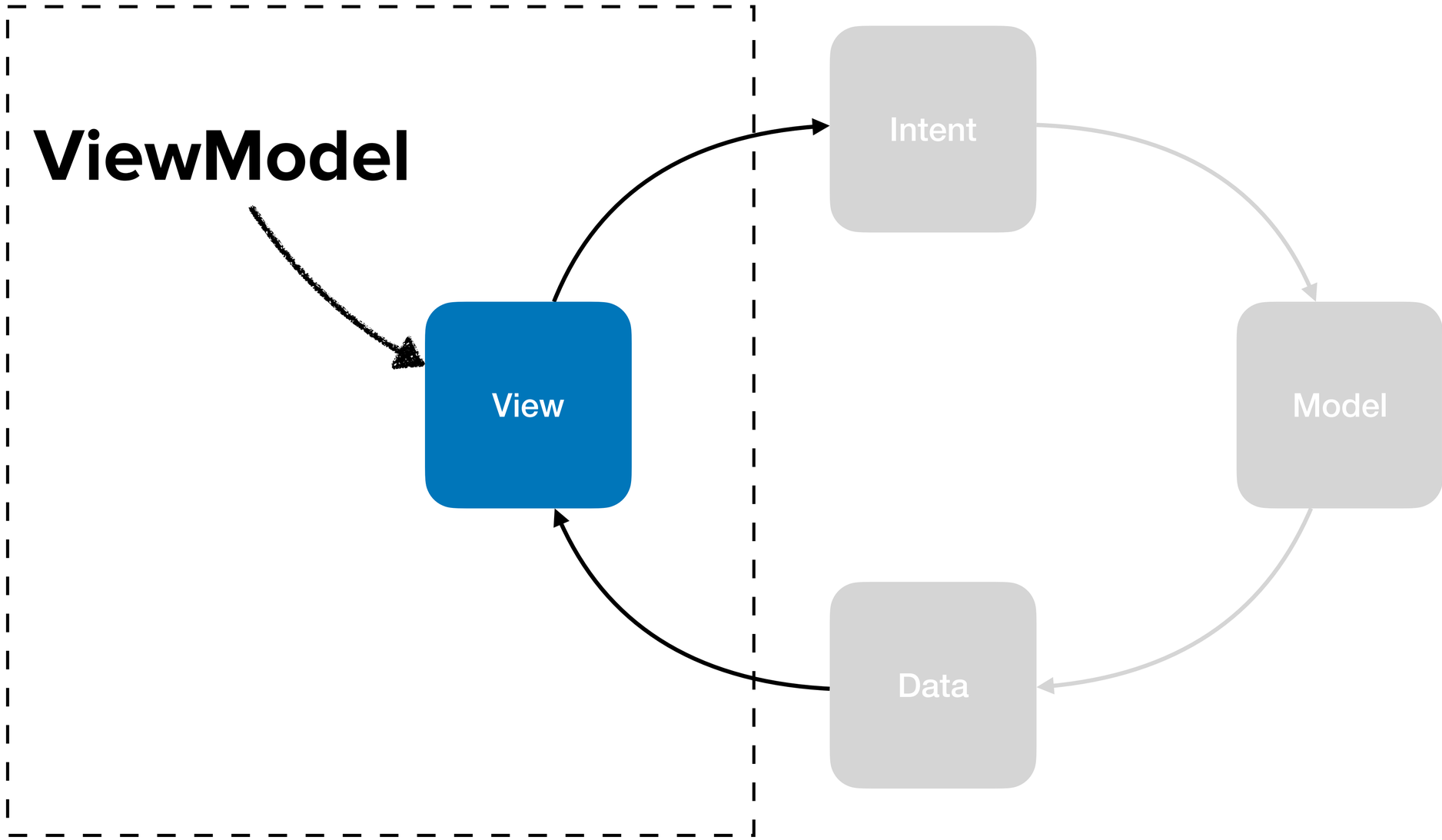
Model-View-Intent



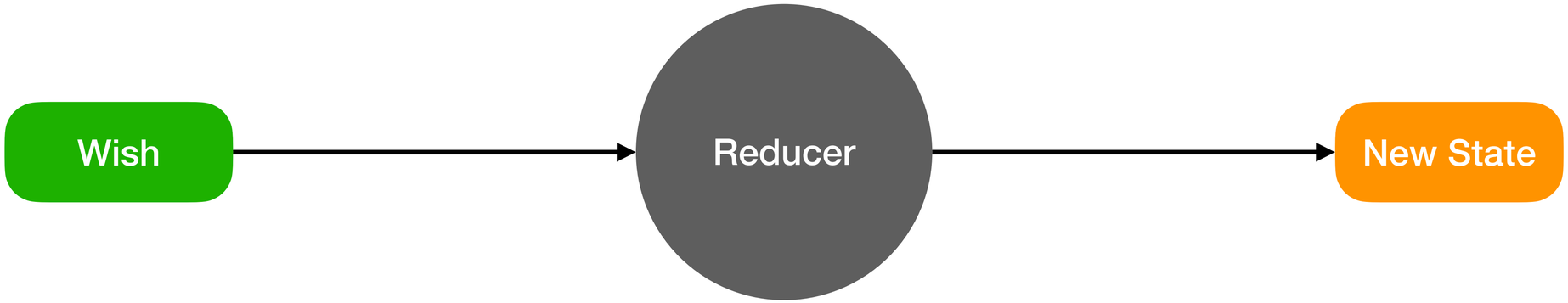
Чуть подробнее



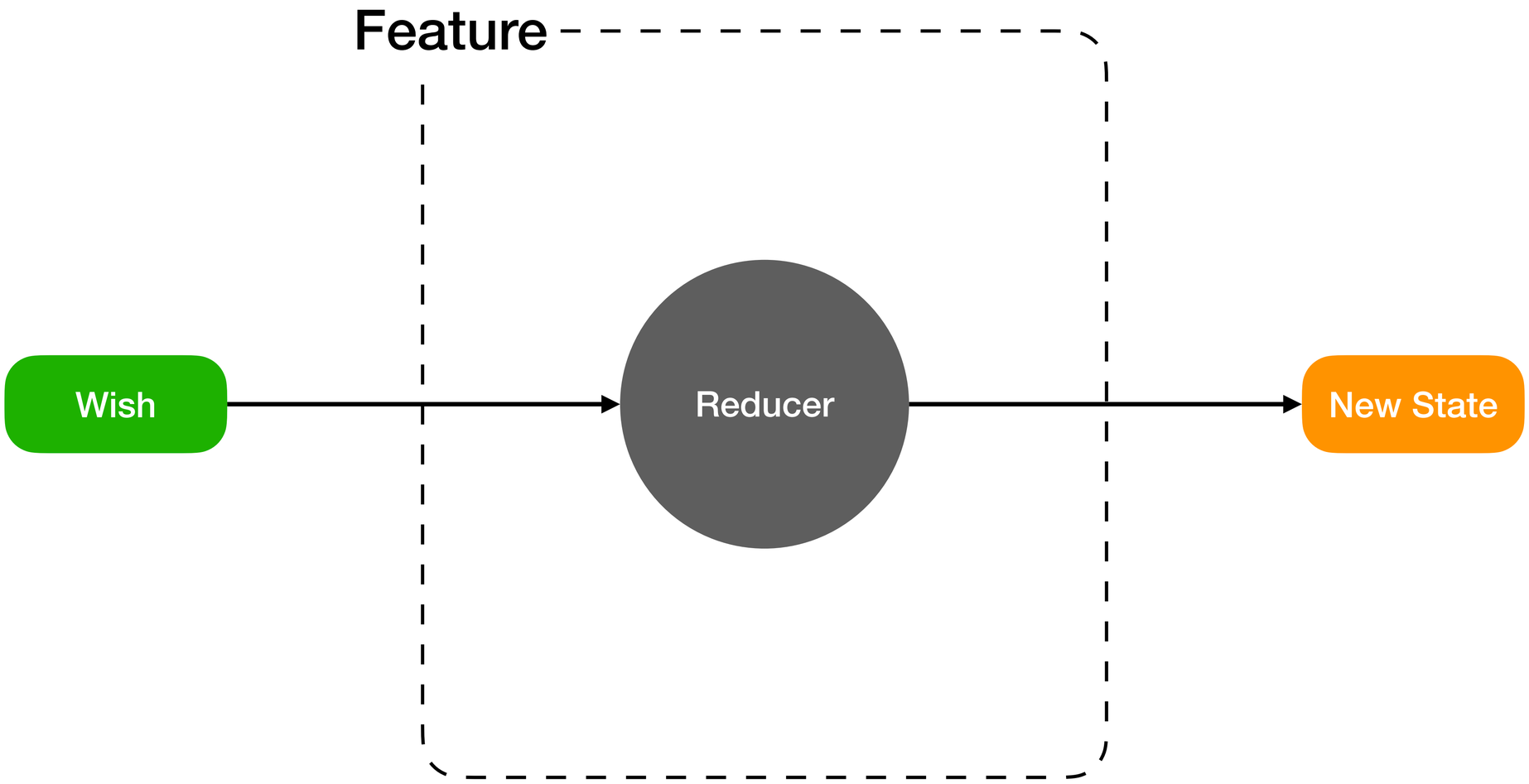
Не обязательно View



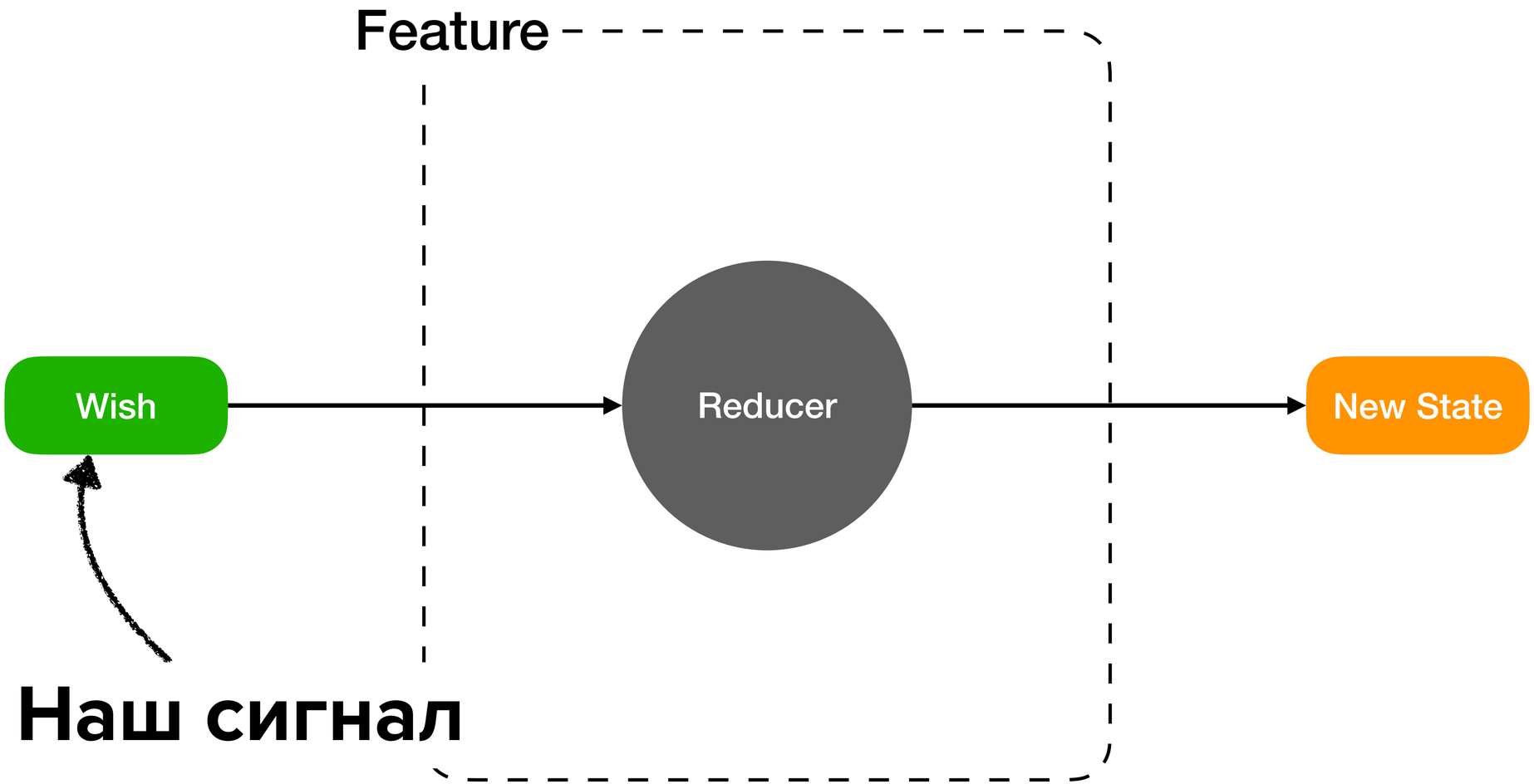
Простейшая стейт-машина



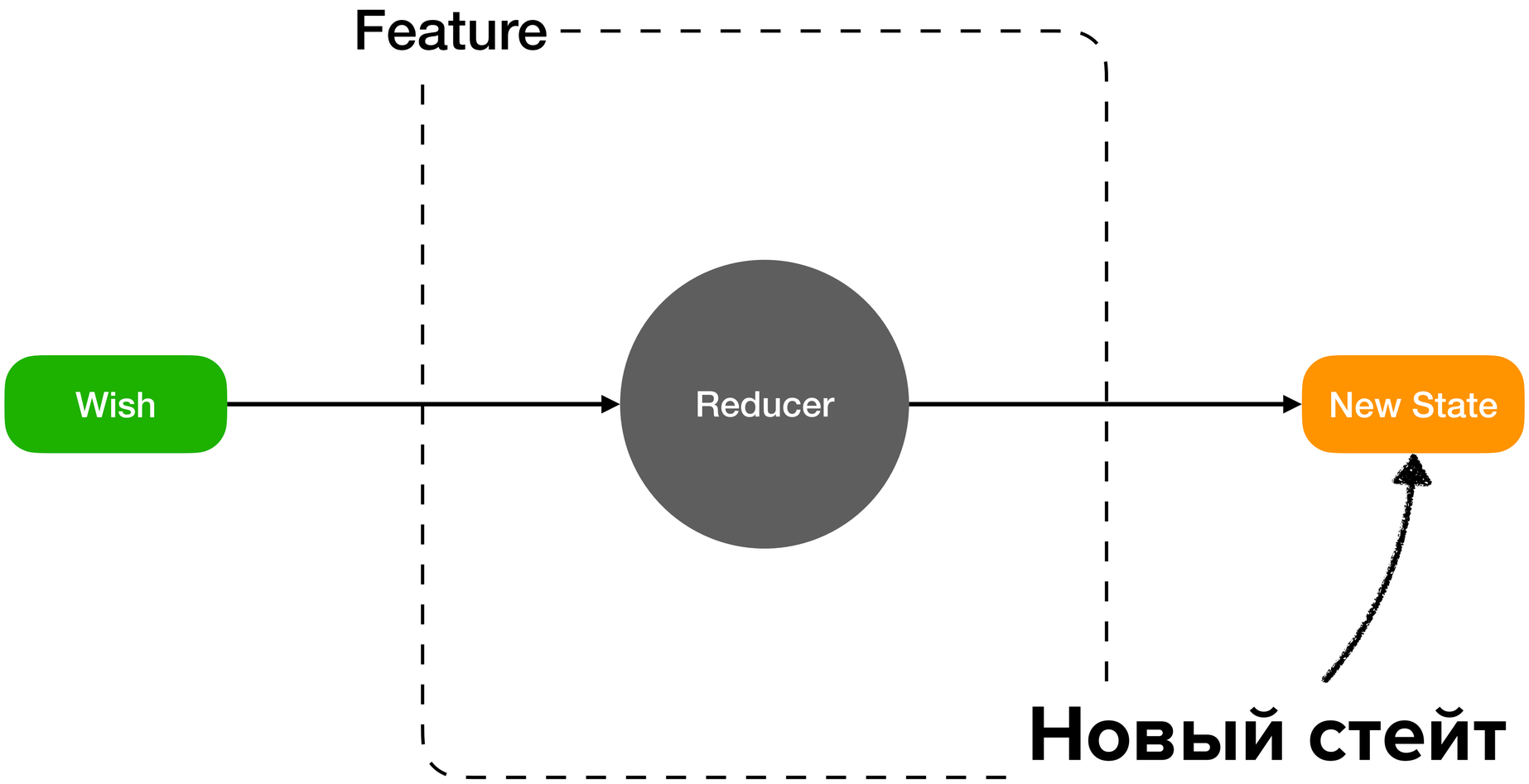
Простейшая стейт-машина



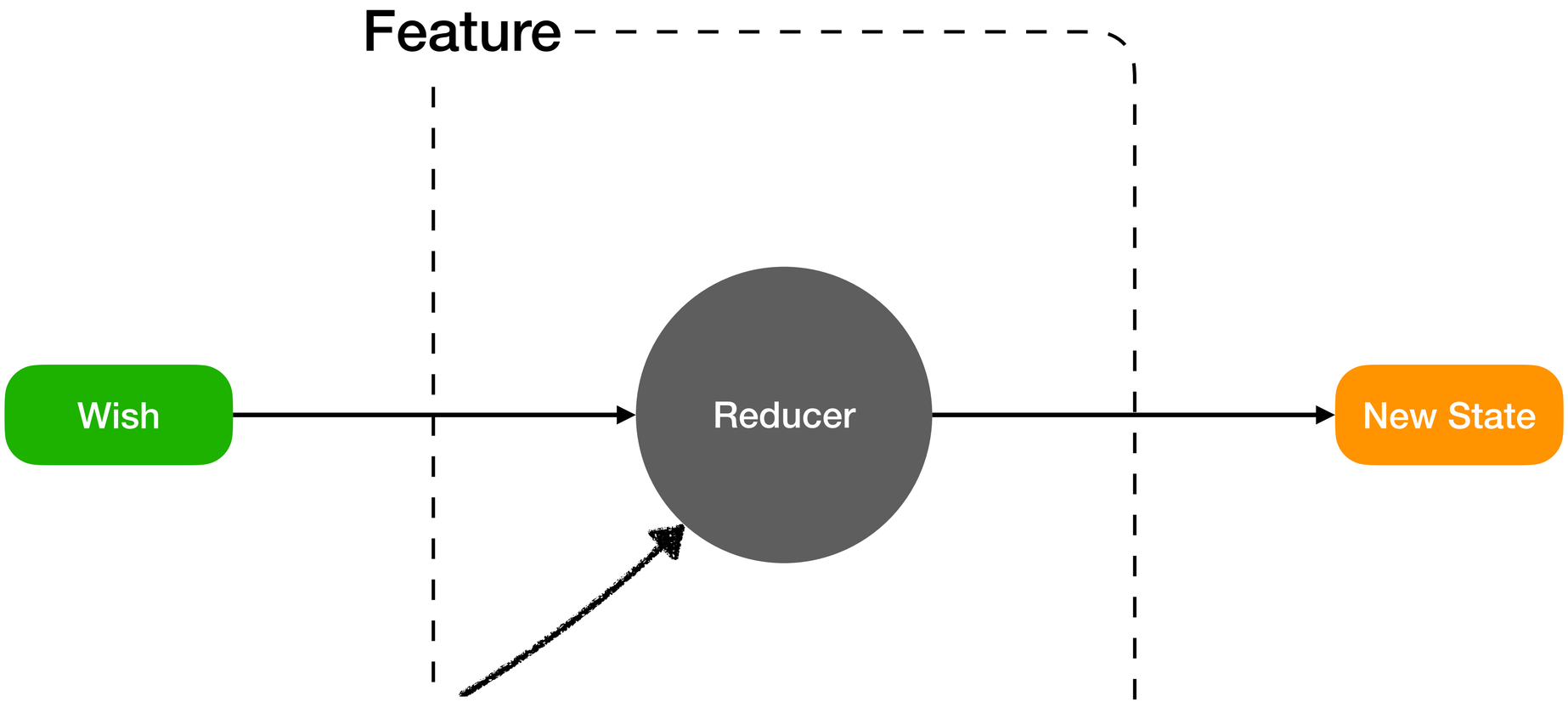
Простейшая стейт-машина



Простейшая стейт-машина



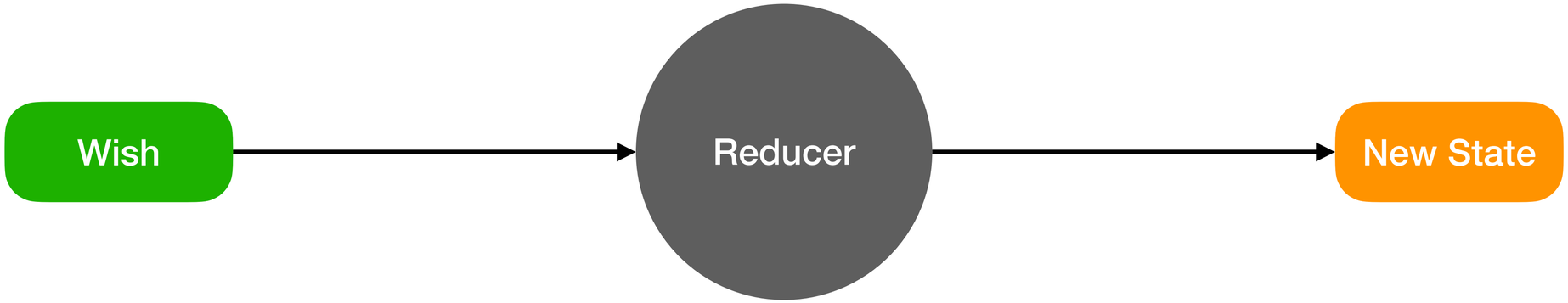
Простейшая стейт-машина



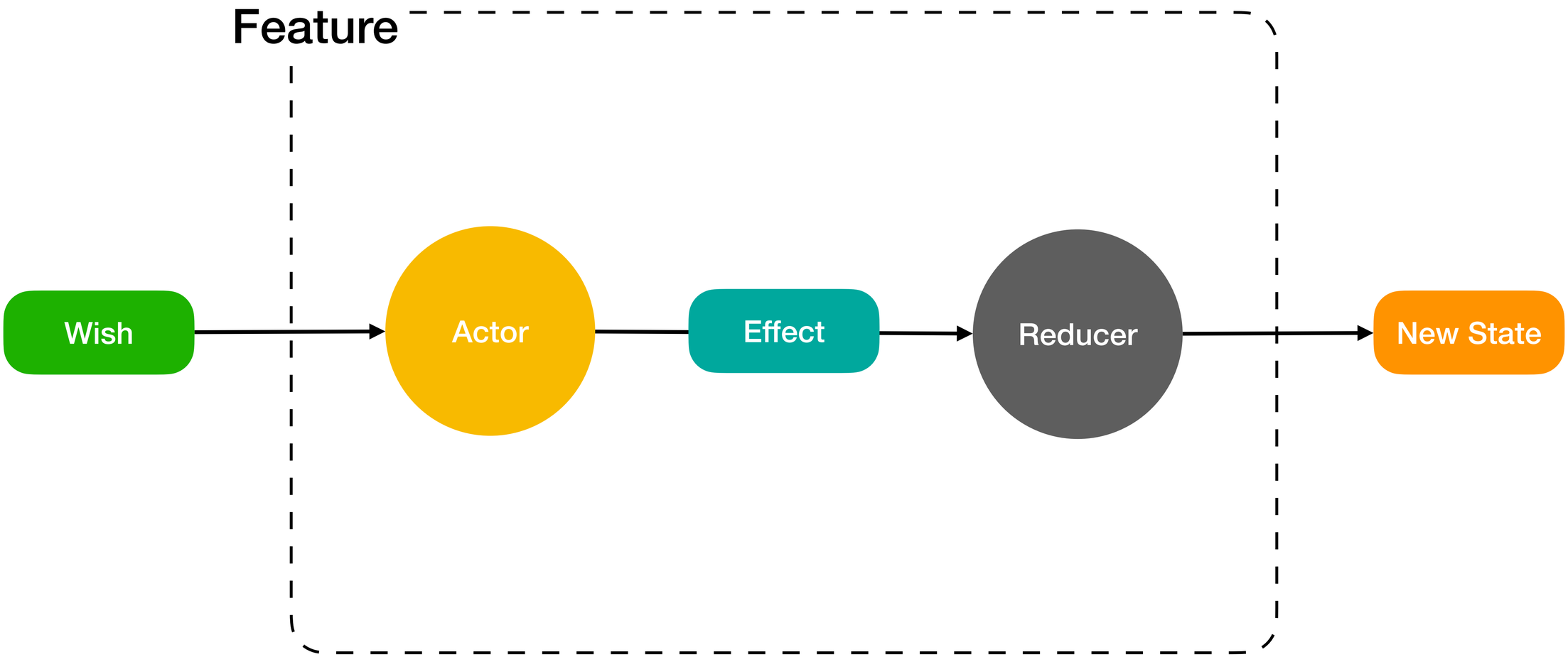
Чистая функция
для изменения стейта

Reducer

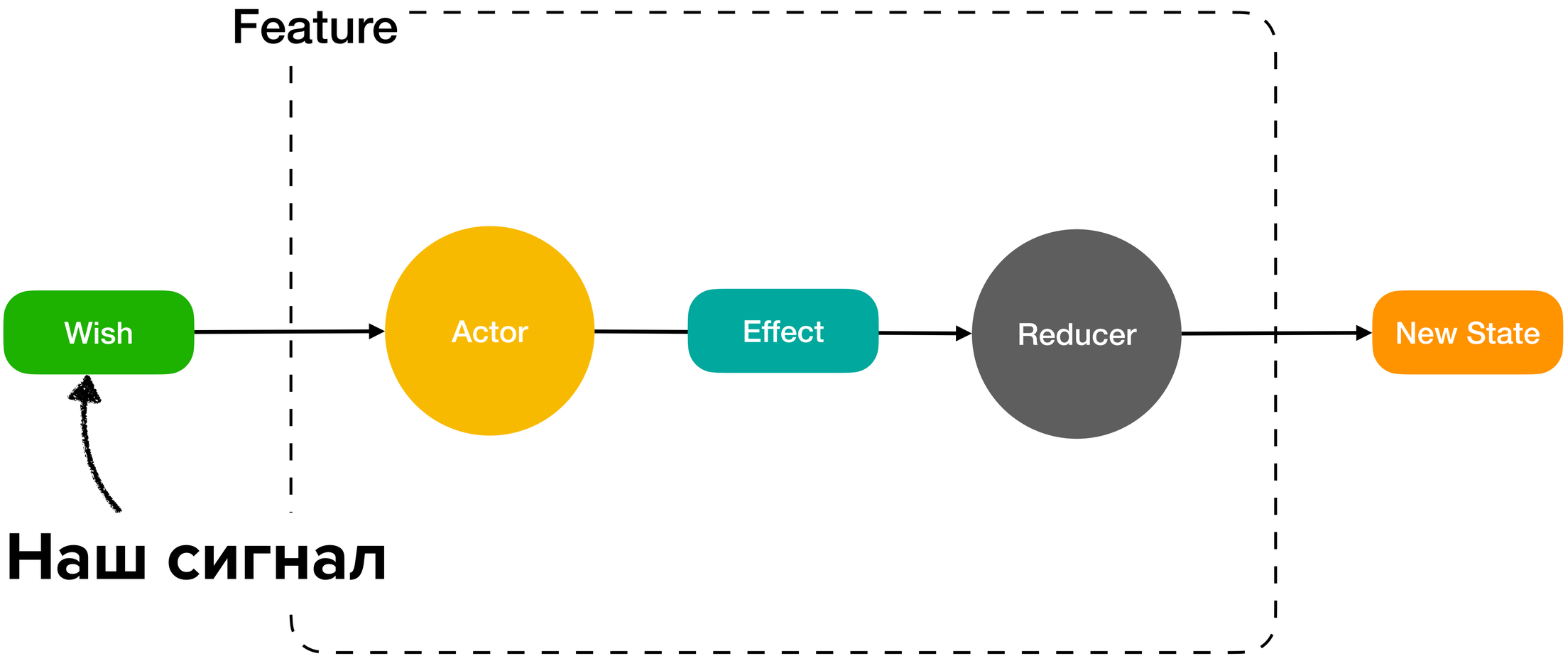
1. Изменяет текущий стейт с учетом *Wish*
2. Работает синхронно



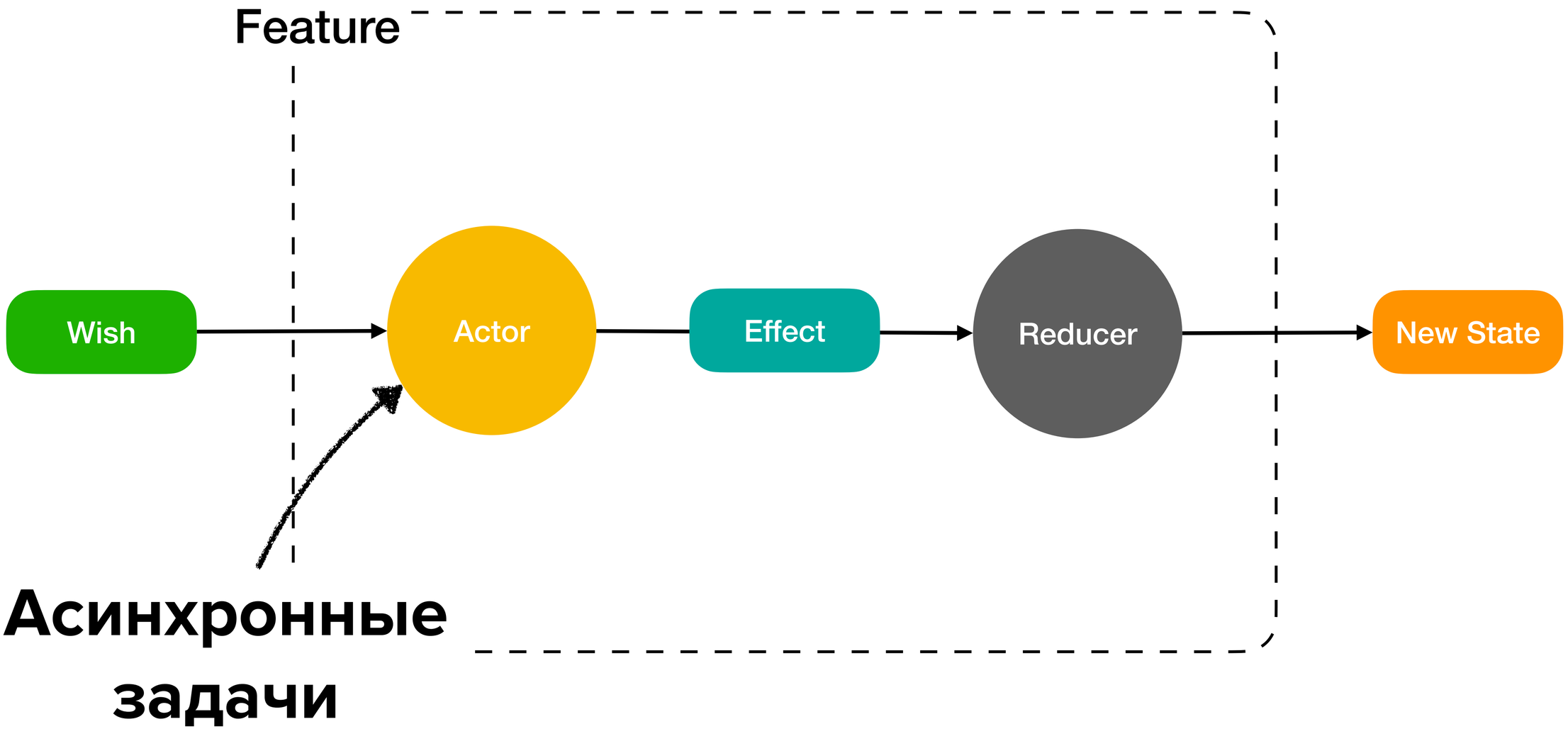
Полноценная MVI-машина



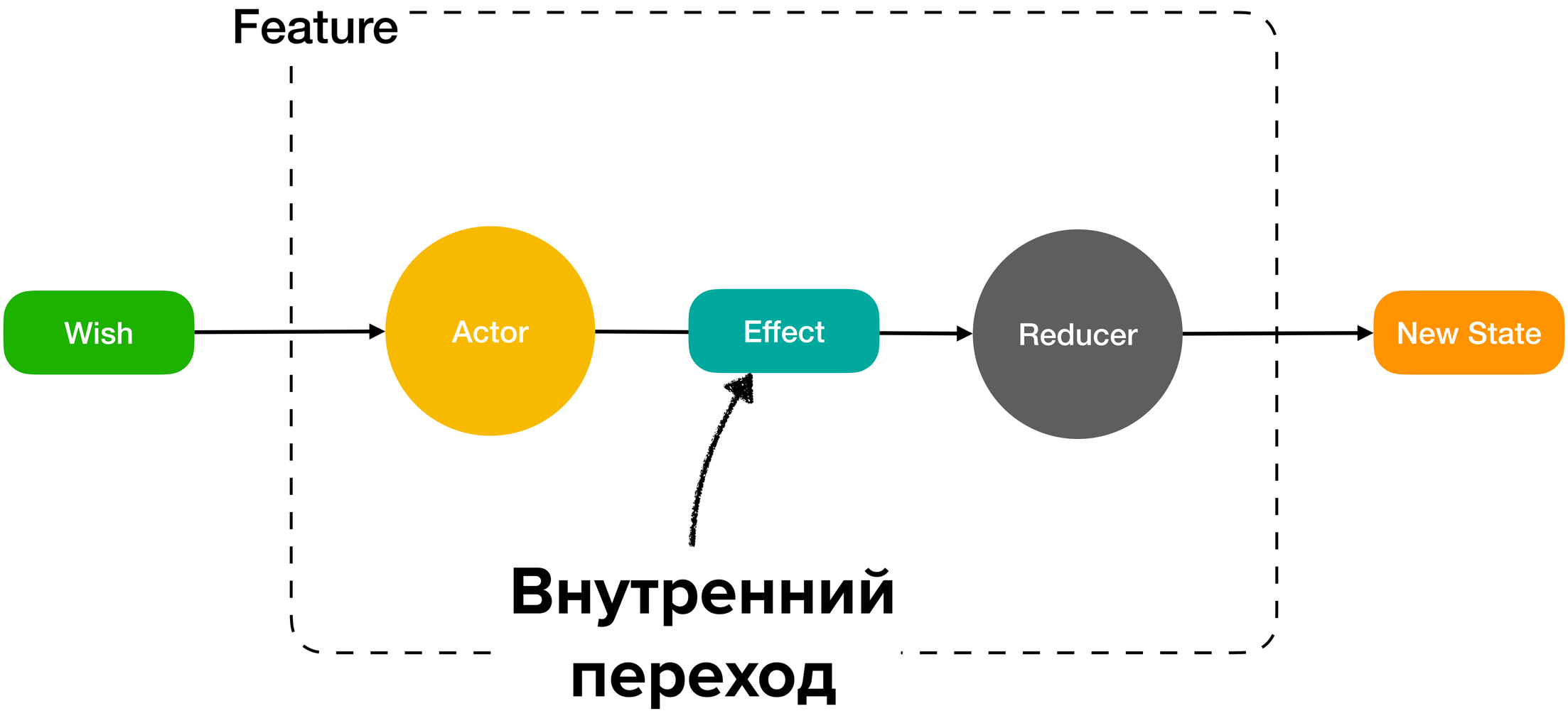
Полноценная MVI-машина



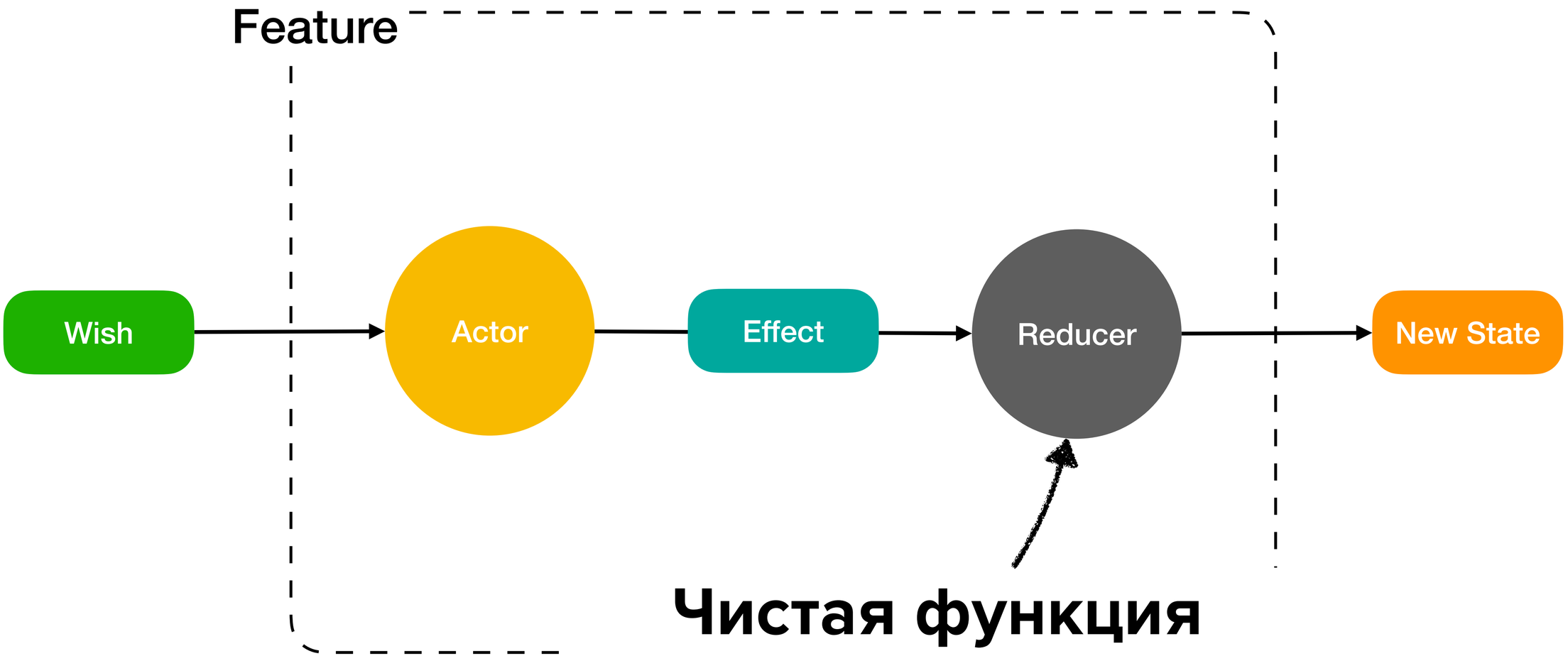
Полноценная MVI-машина



Полноценная MVI-машина

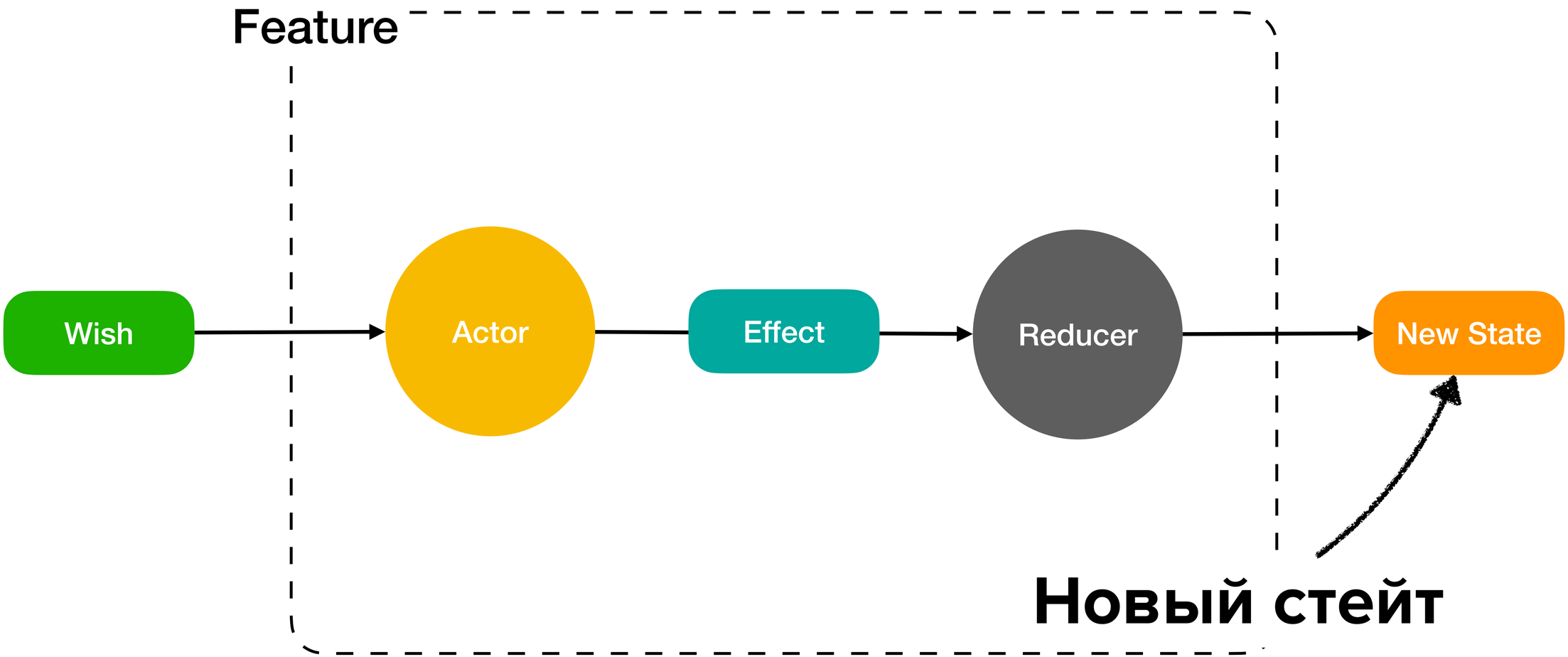


Полноценная MVI-машина



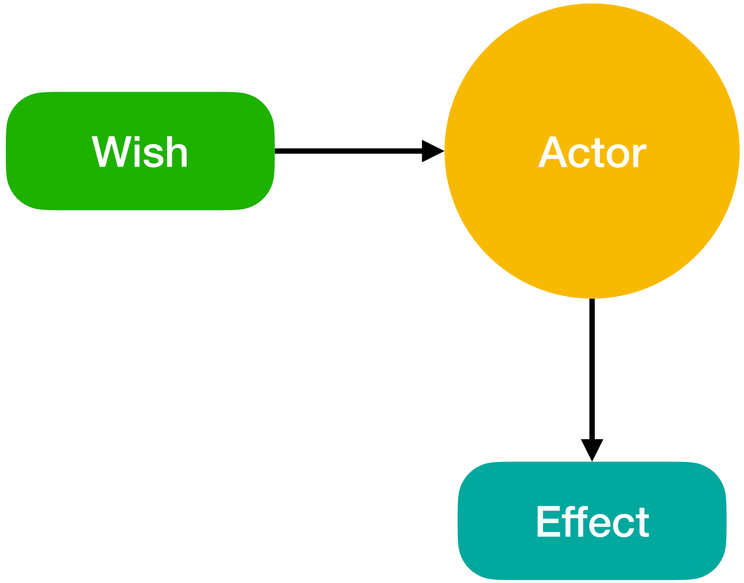
Чистая функция для изменения стейта

Полноценная MVI-машина

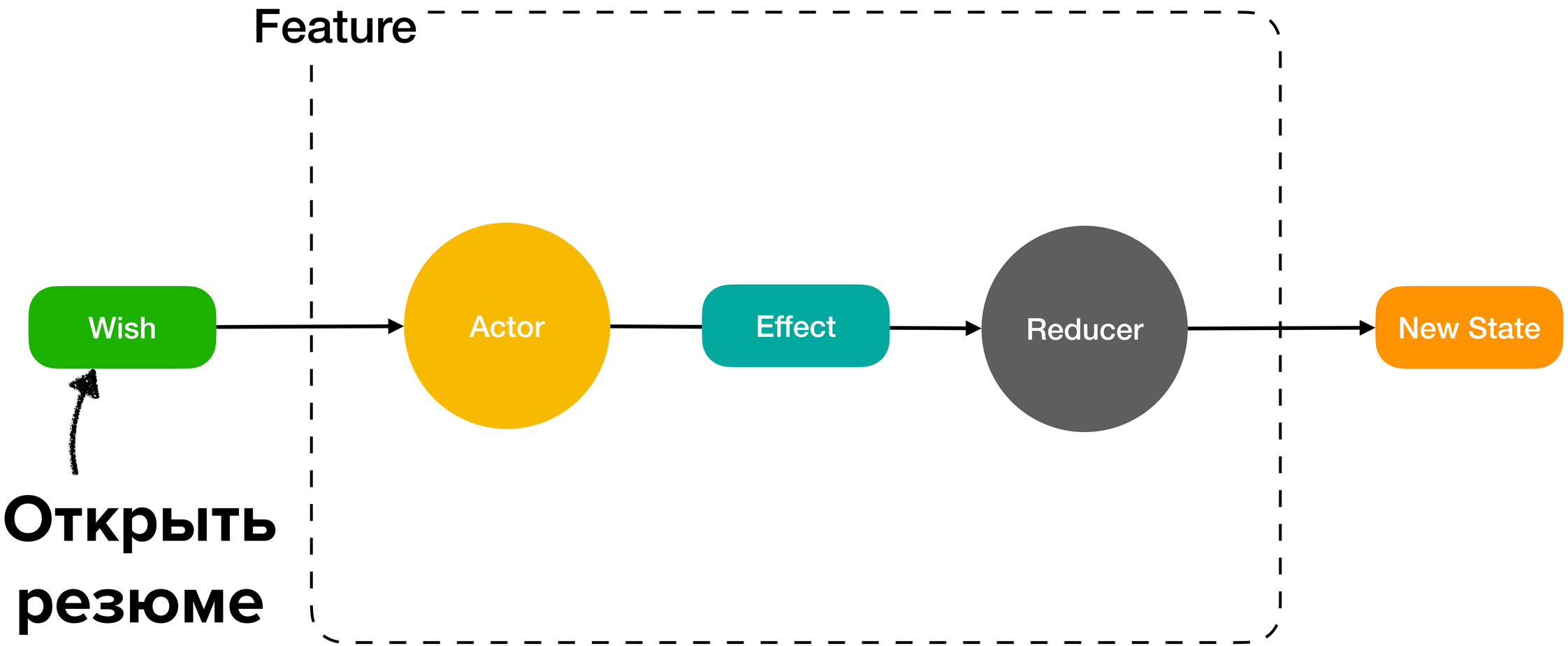


Что ты такое, Actor

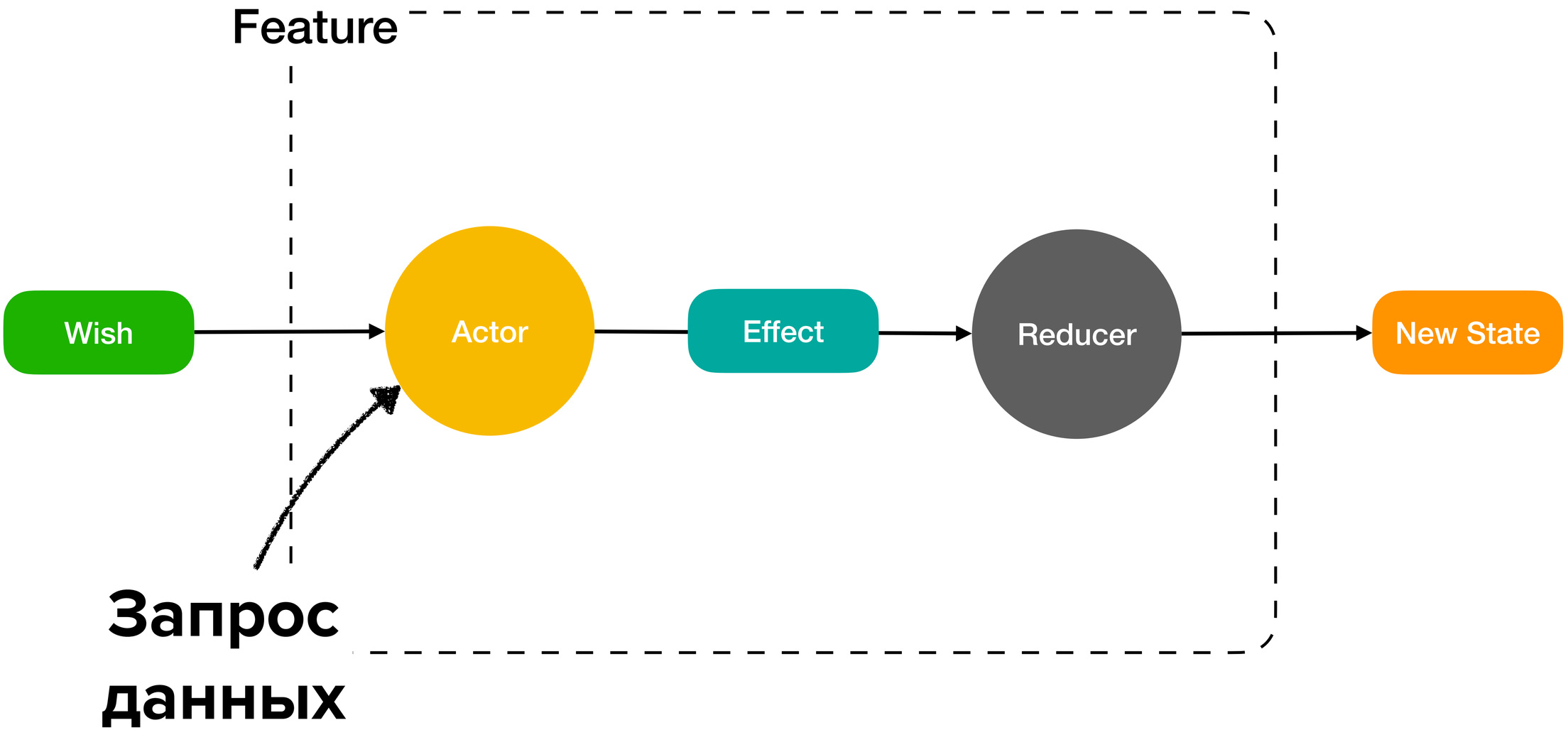
1. Превращает *Wish*-ы в *Effect*-ы
2. Не знает какой стейт следующий
3. Запрашивает данные из сервисов
4. Из одного *Wish* может сделать несколько *Effect*



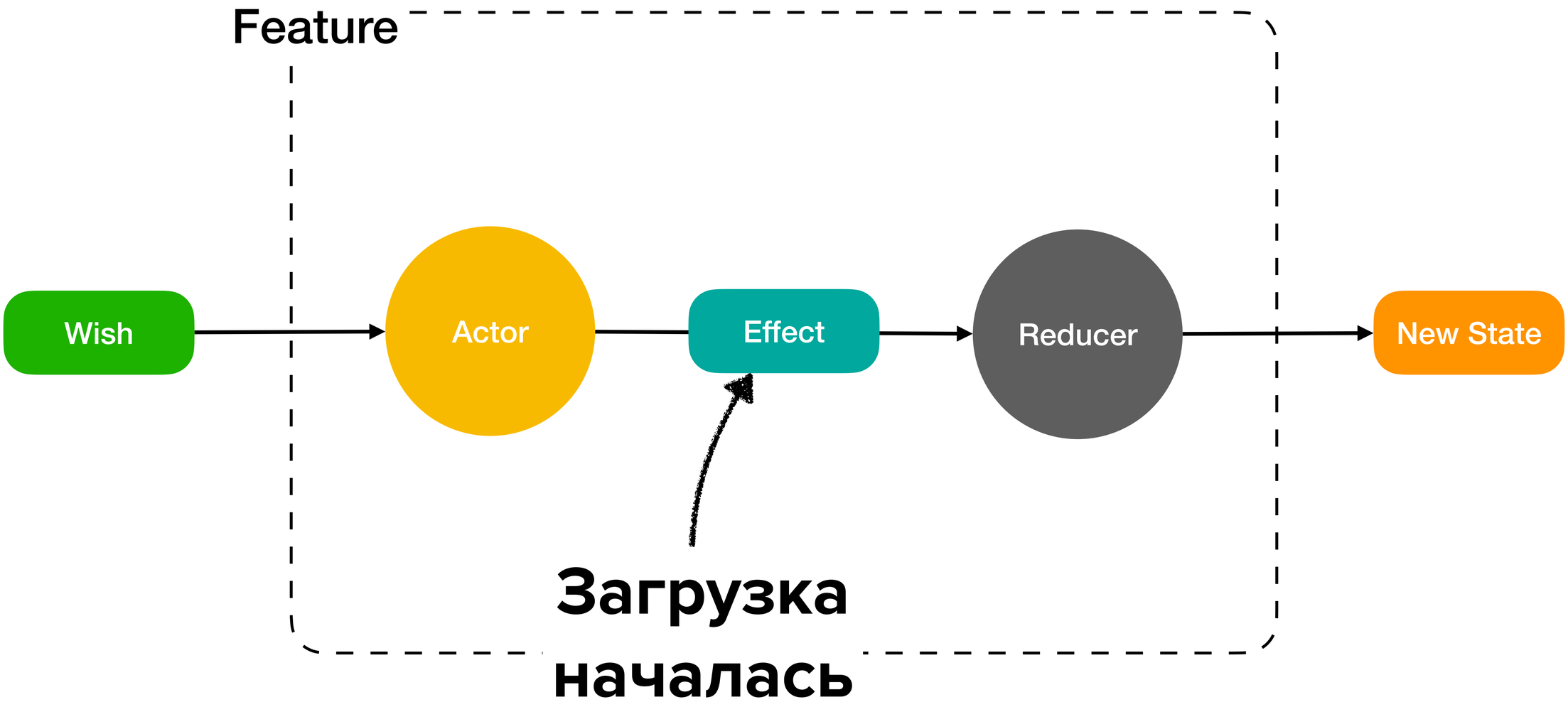
Полноценная MVI-машина



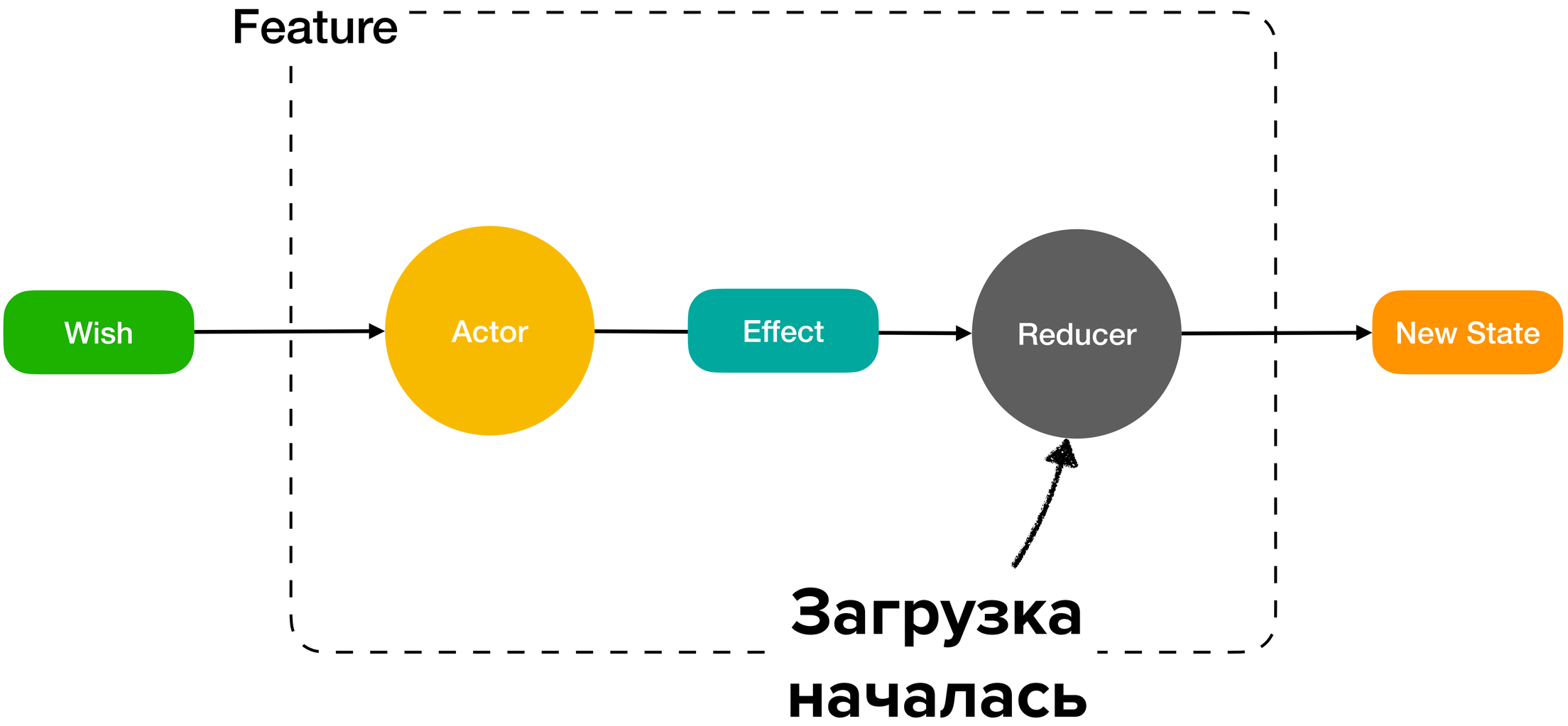
Полноценная MVI-машина



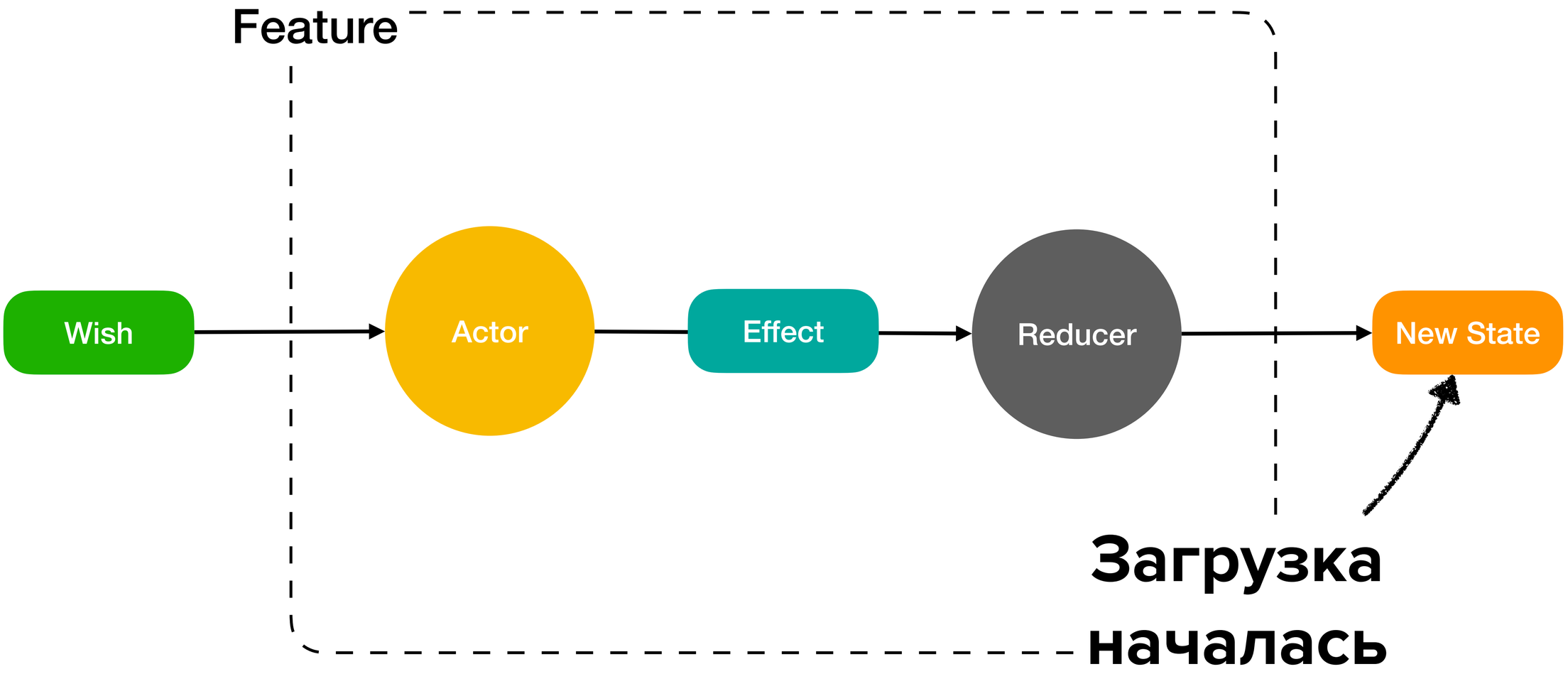
Полноценная MVI-машина



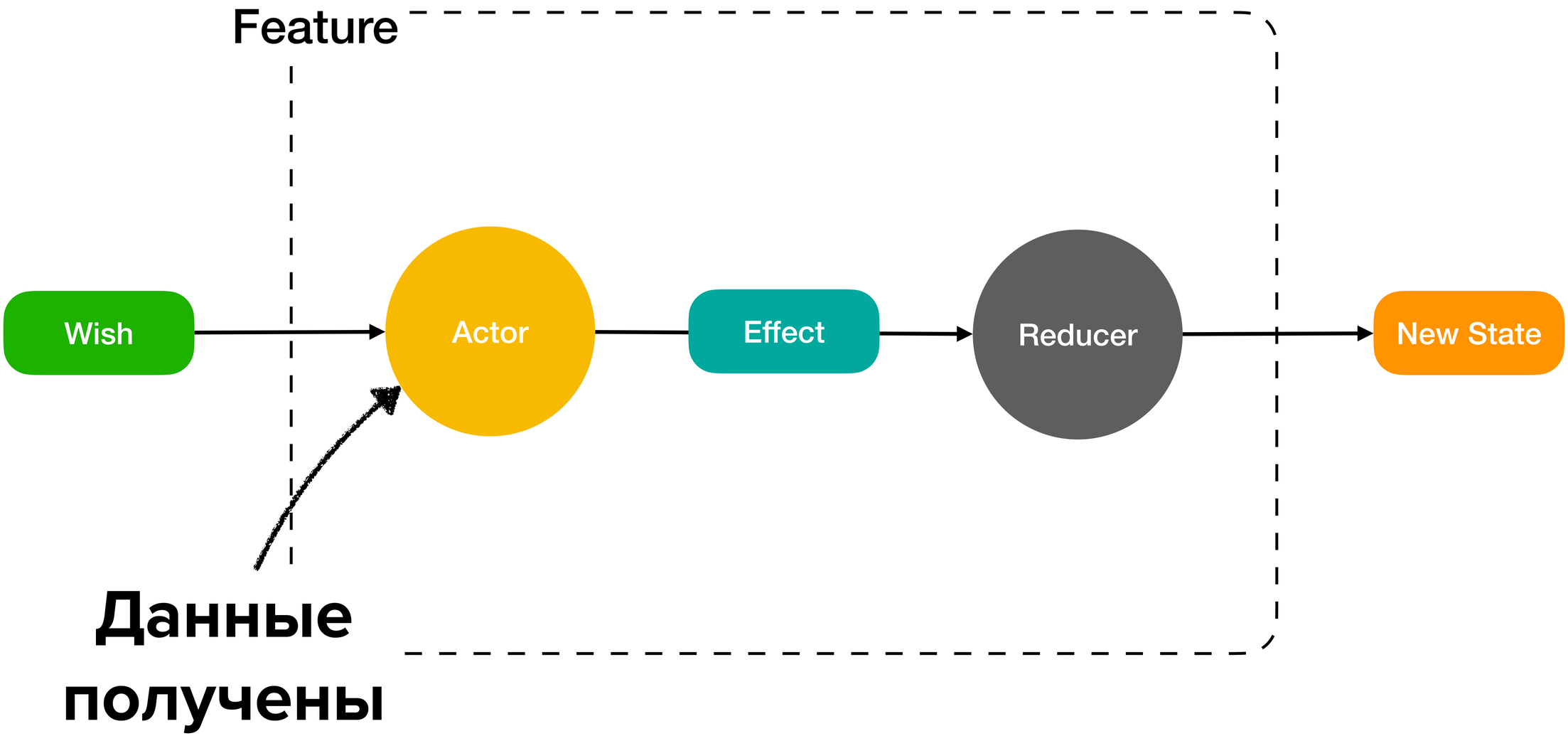
Полноценная MVI-машина



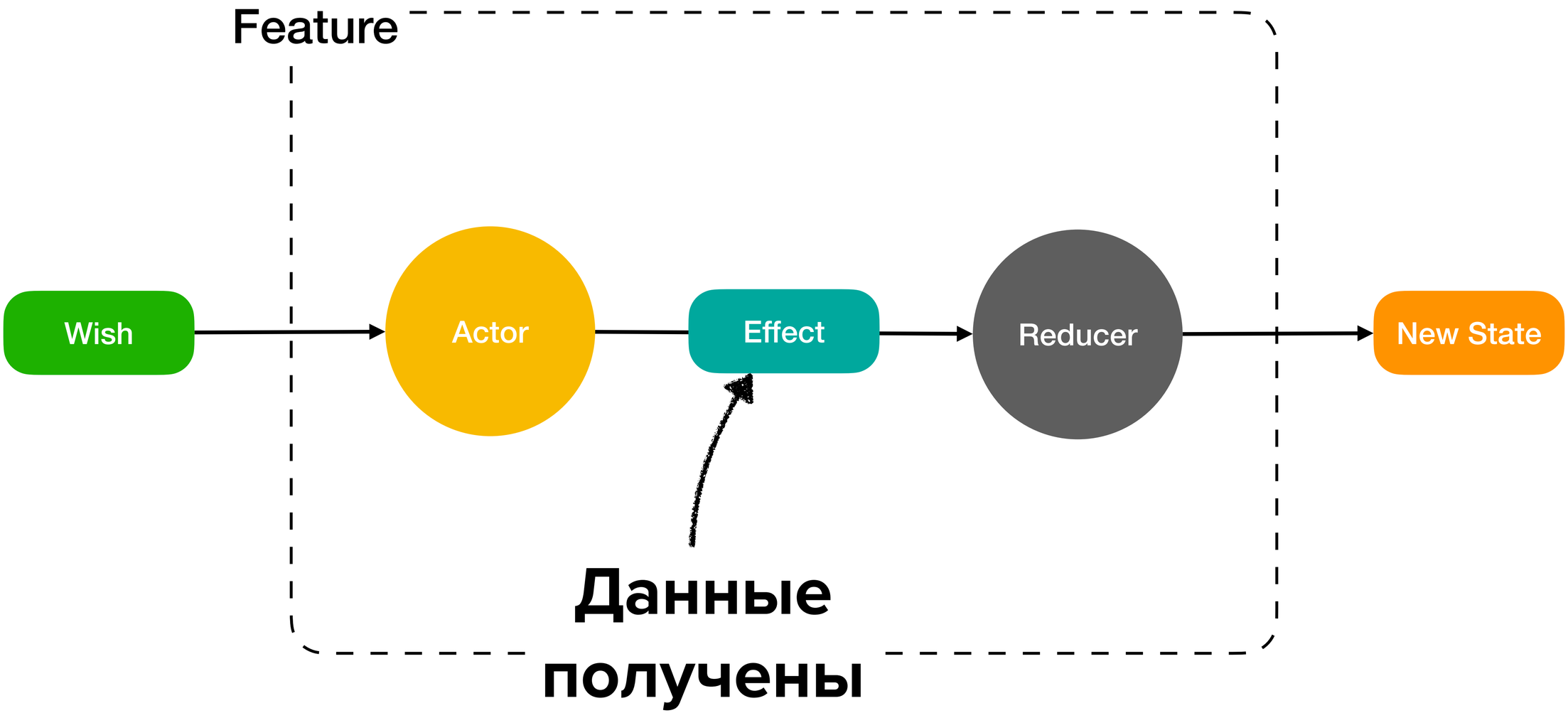
Полноценная MVI-машина



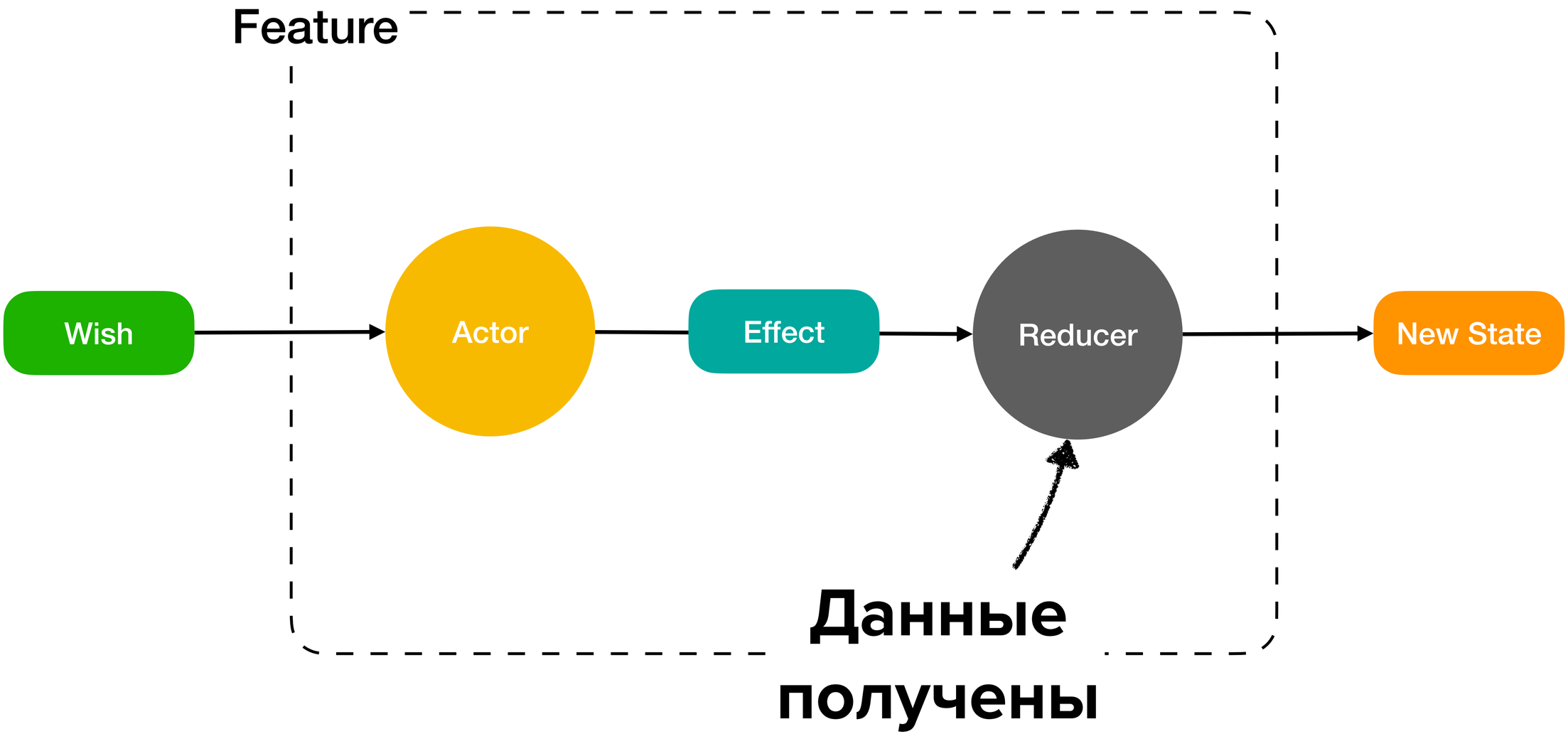
Полноценная MVI-машина



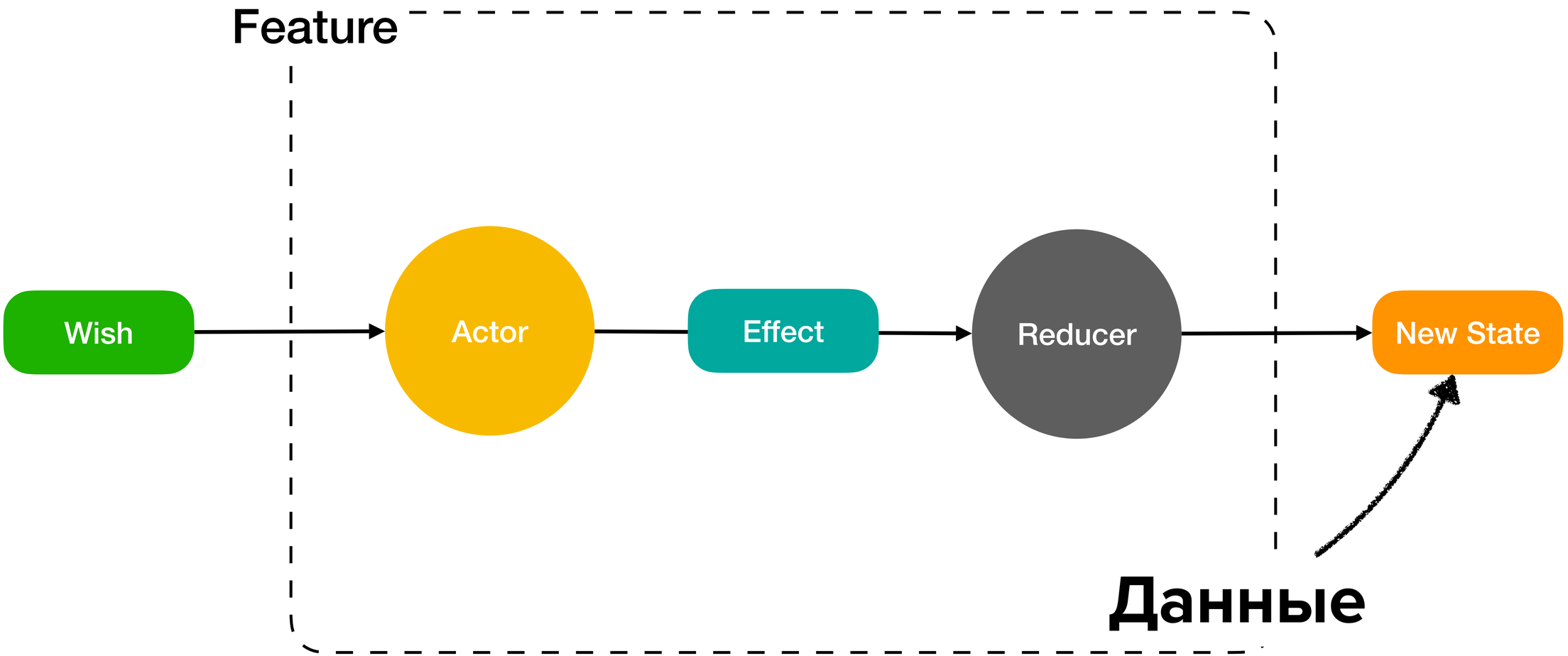
Полноценная MVI-машина



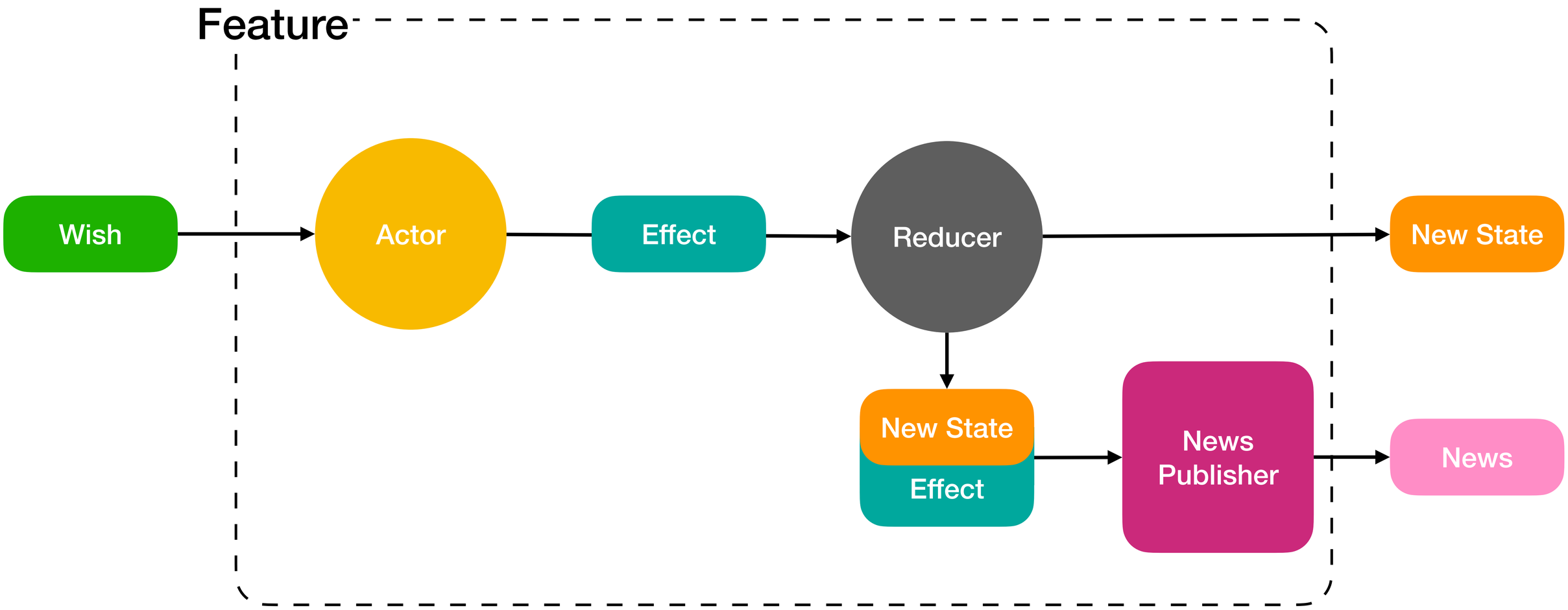
Полноценная MVI-машина



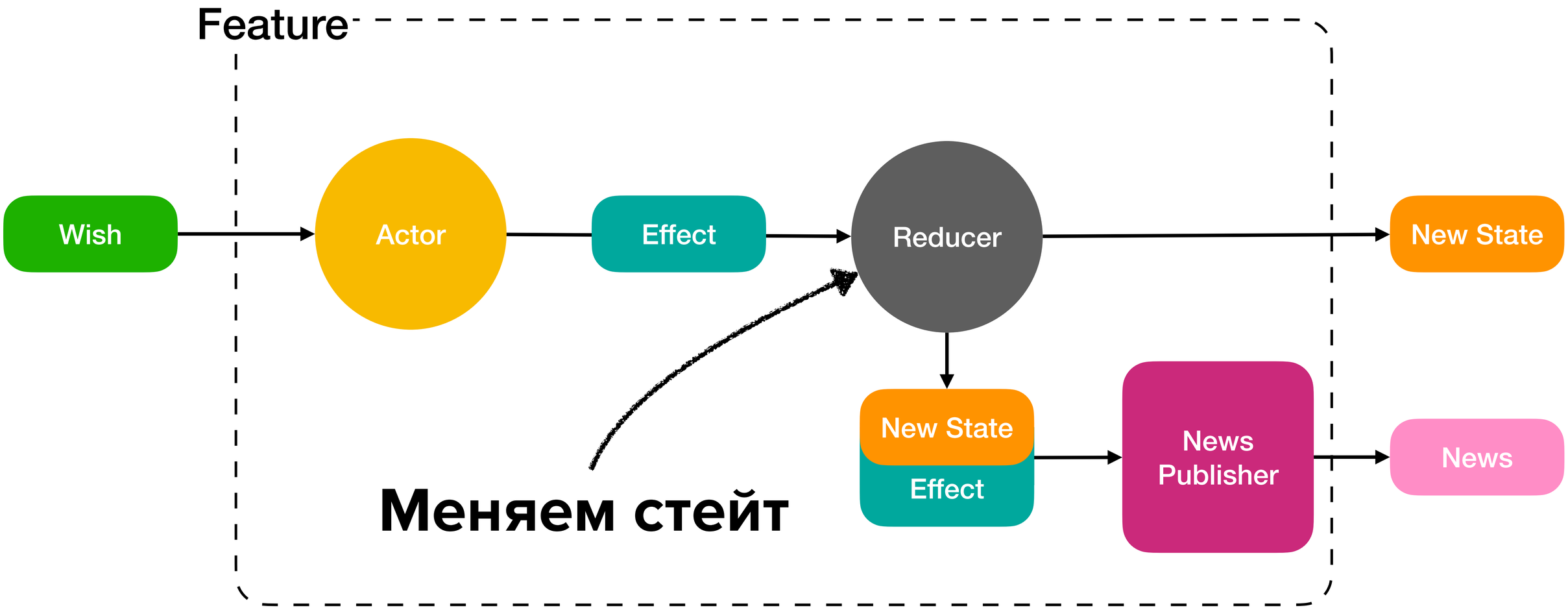
Полноценная MVI-машина



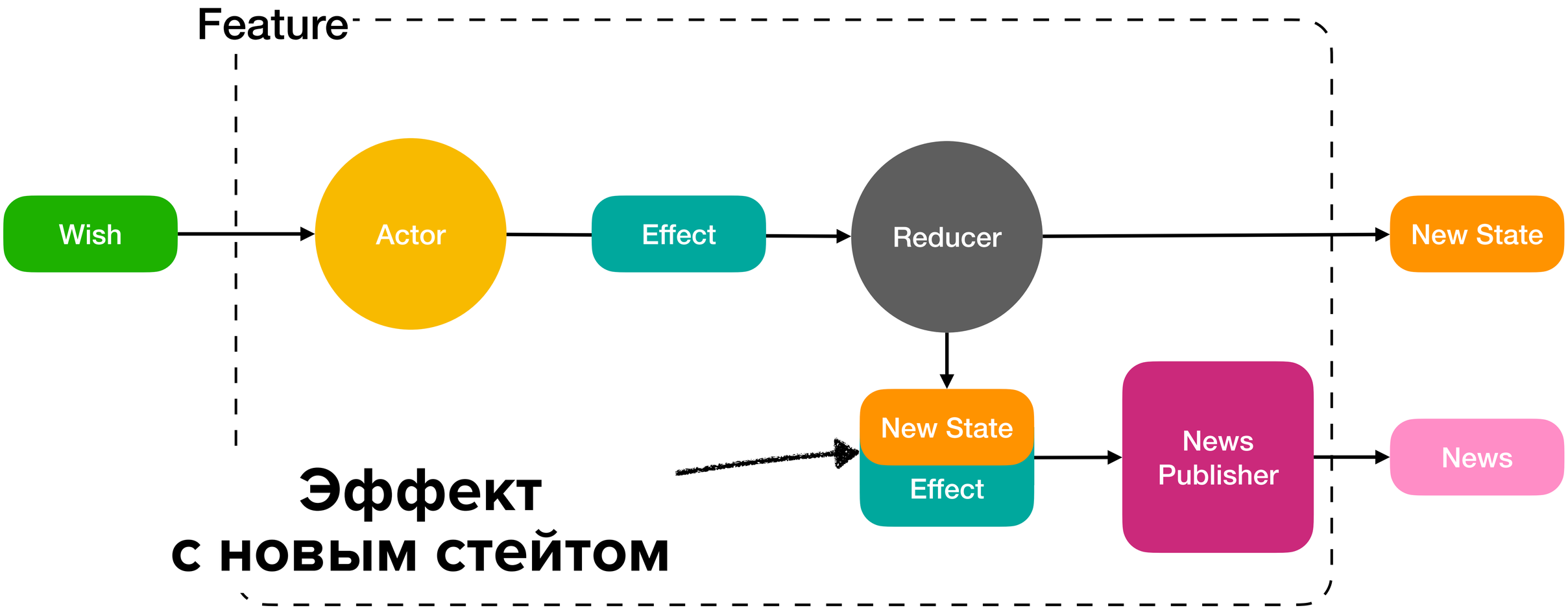
Исходящие сигналы



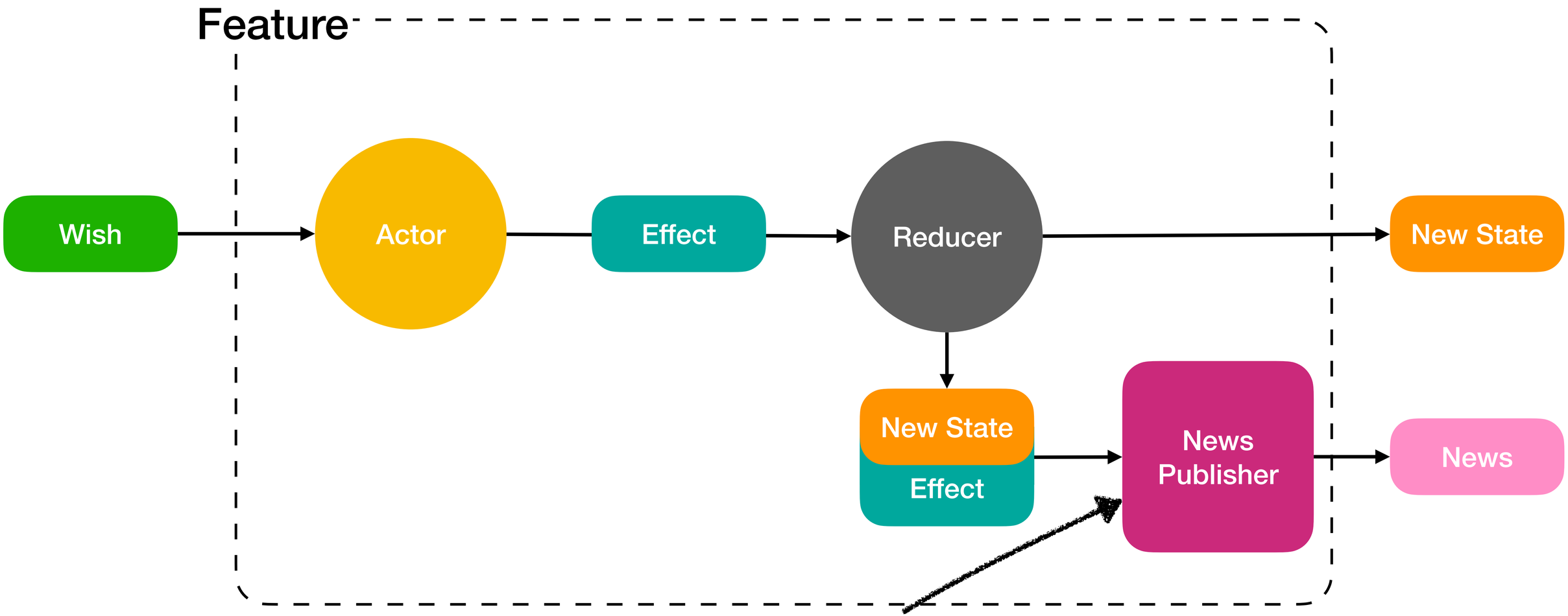
Исходящие сигналы



Исходящие сигналы

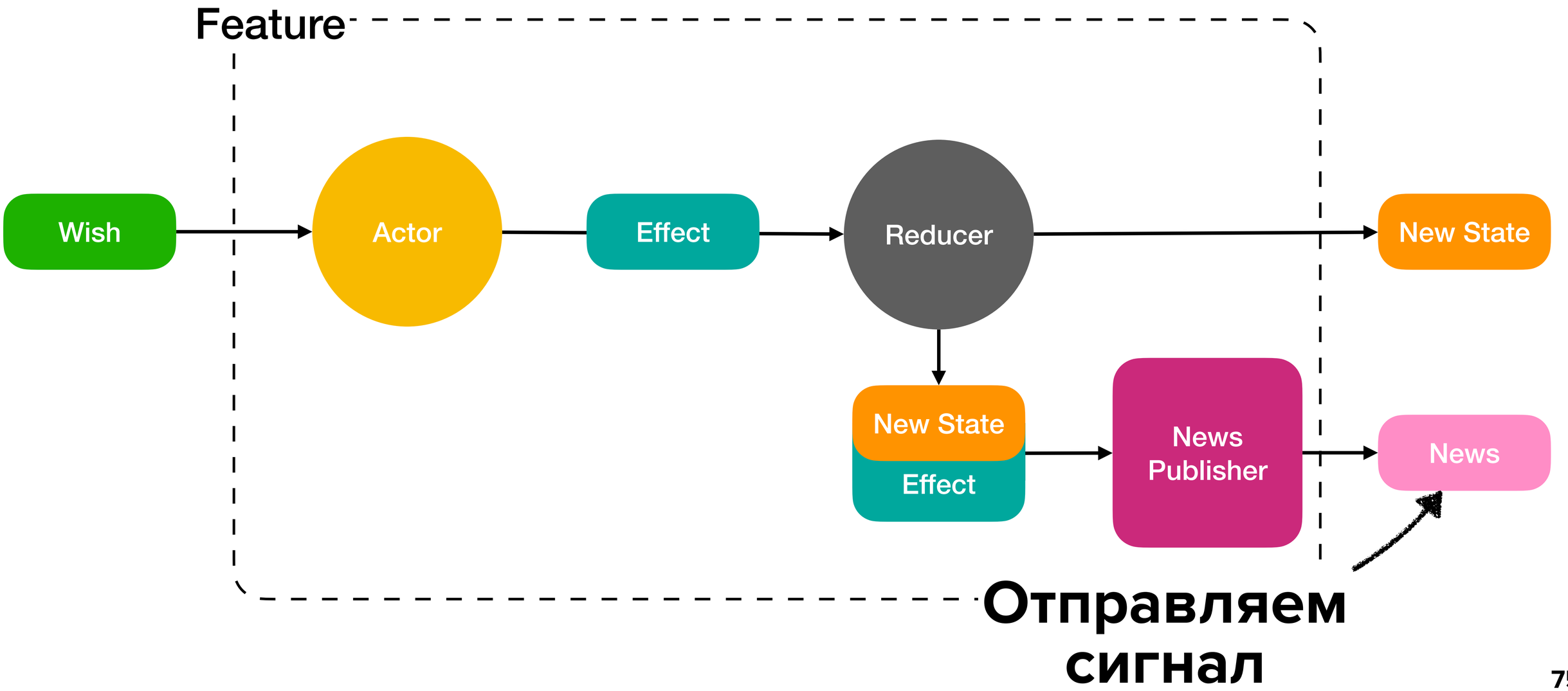


Исходящие сигналы



Решаем, что делать

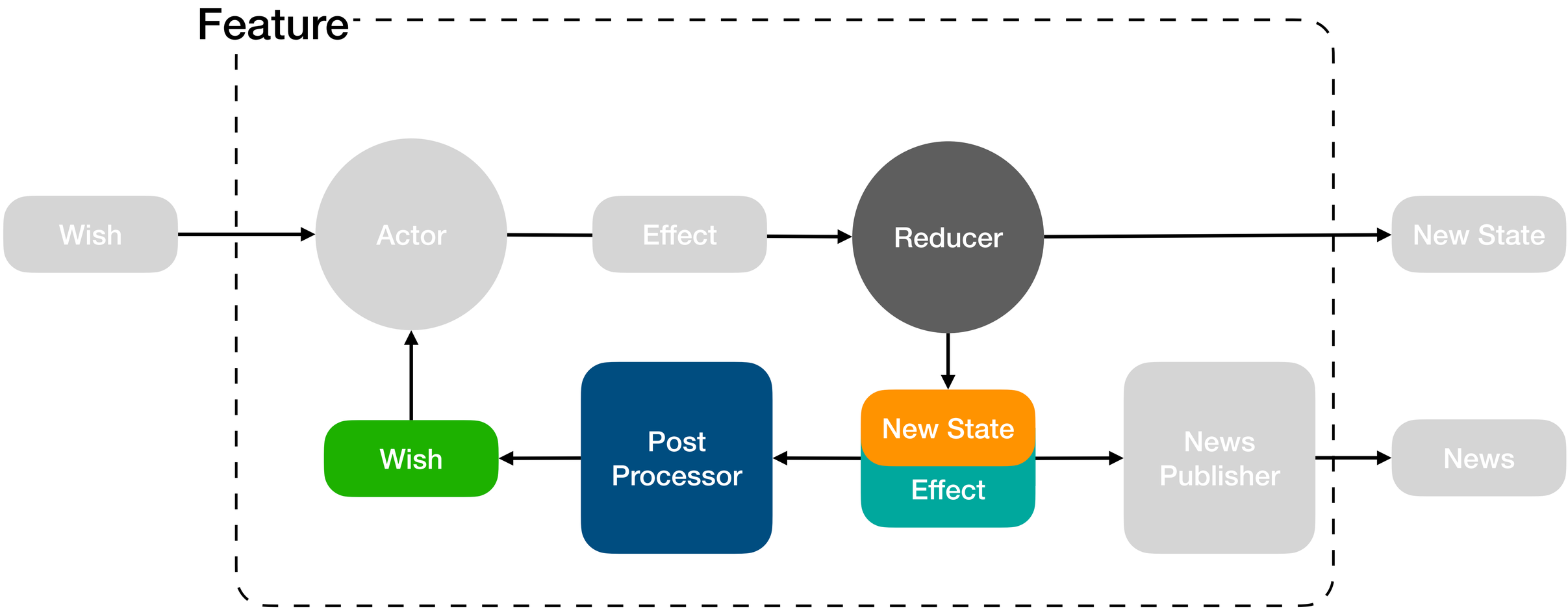
Исходящие сигналы



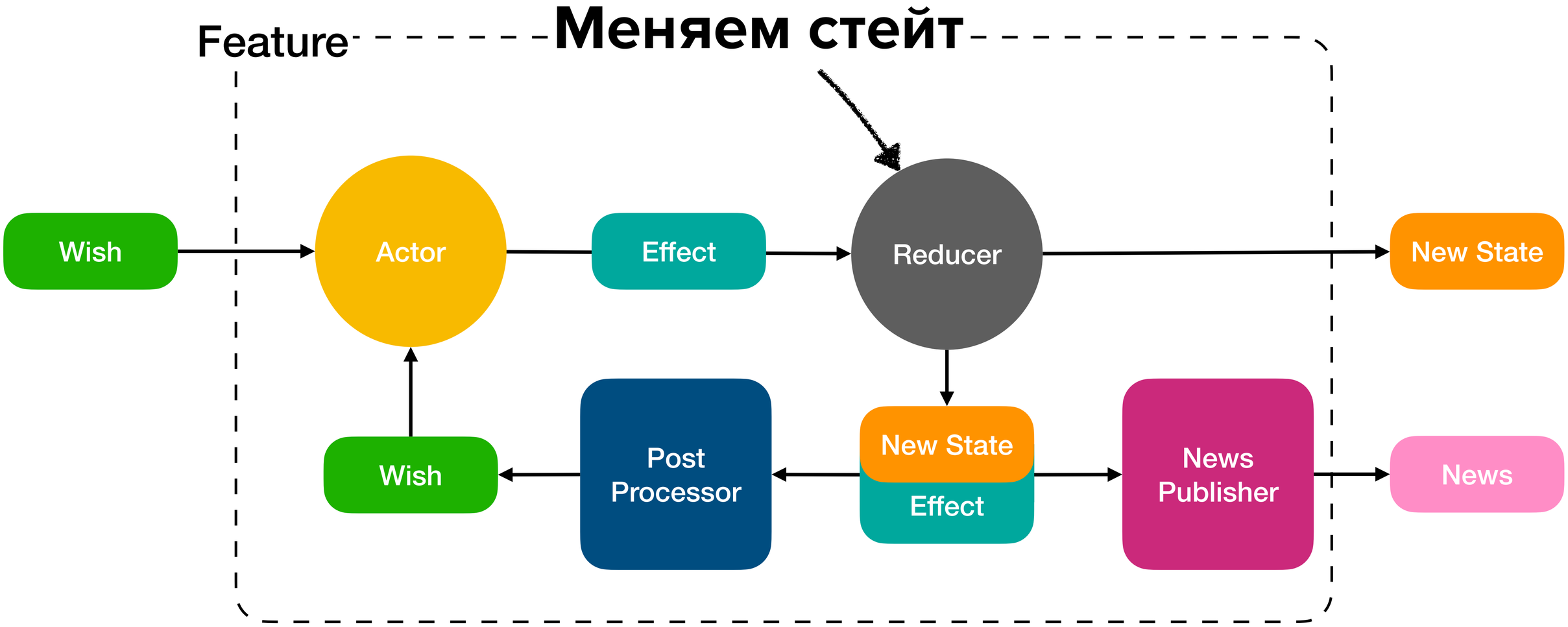
NewsPublisher

1. Отправка сигнала после изменения стейта

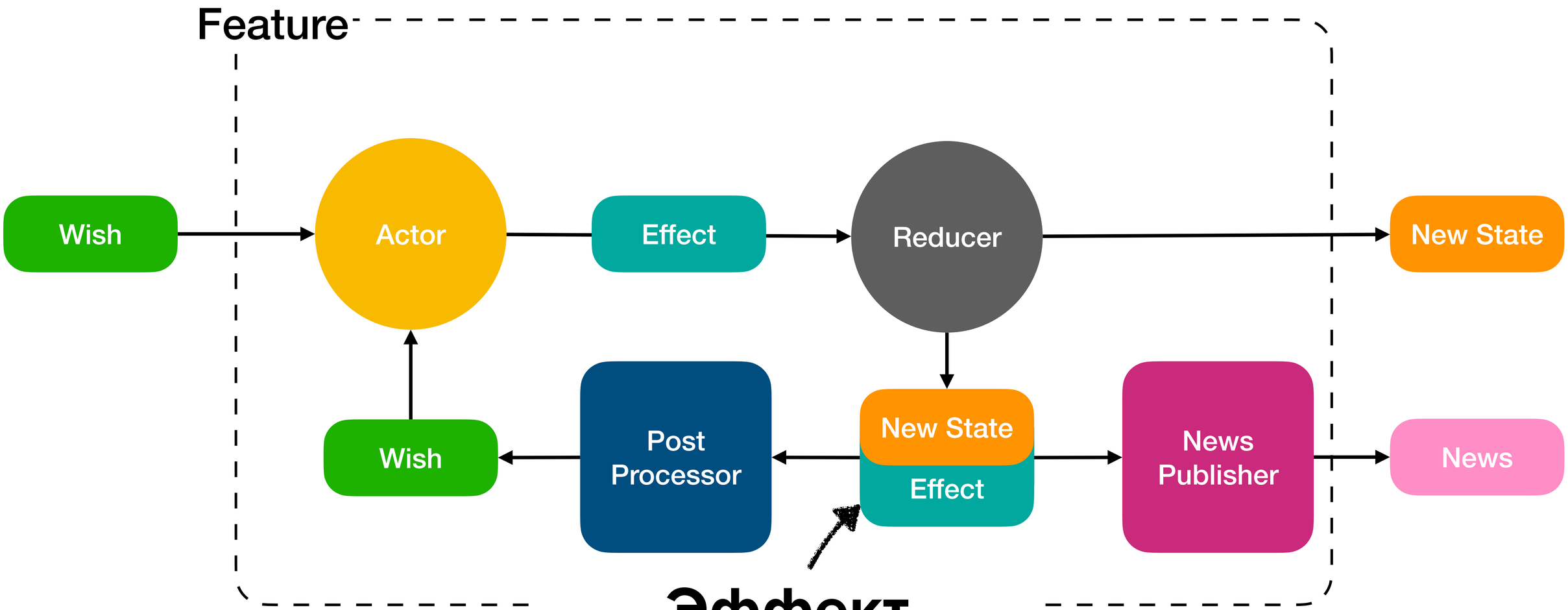
PostProcessor



PostProcessor

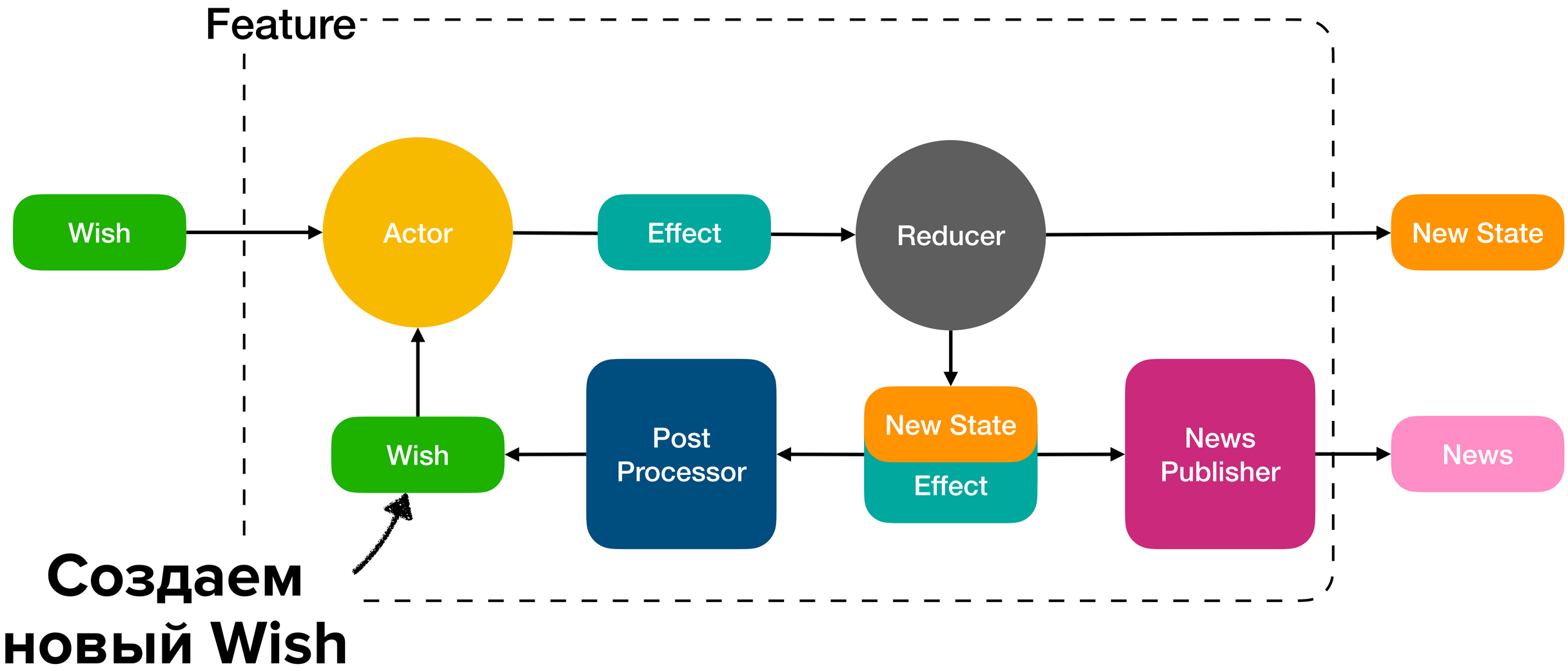


PostProcessor

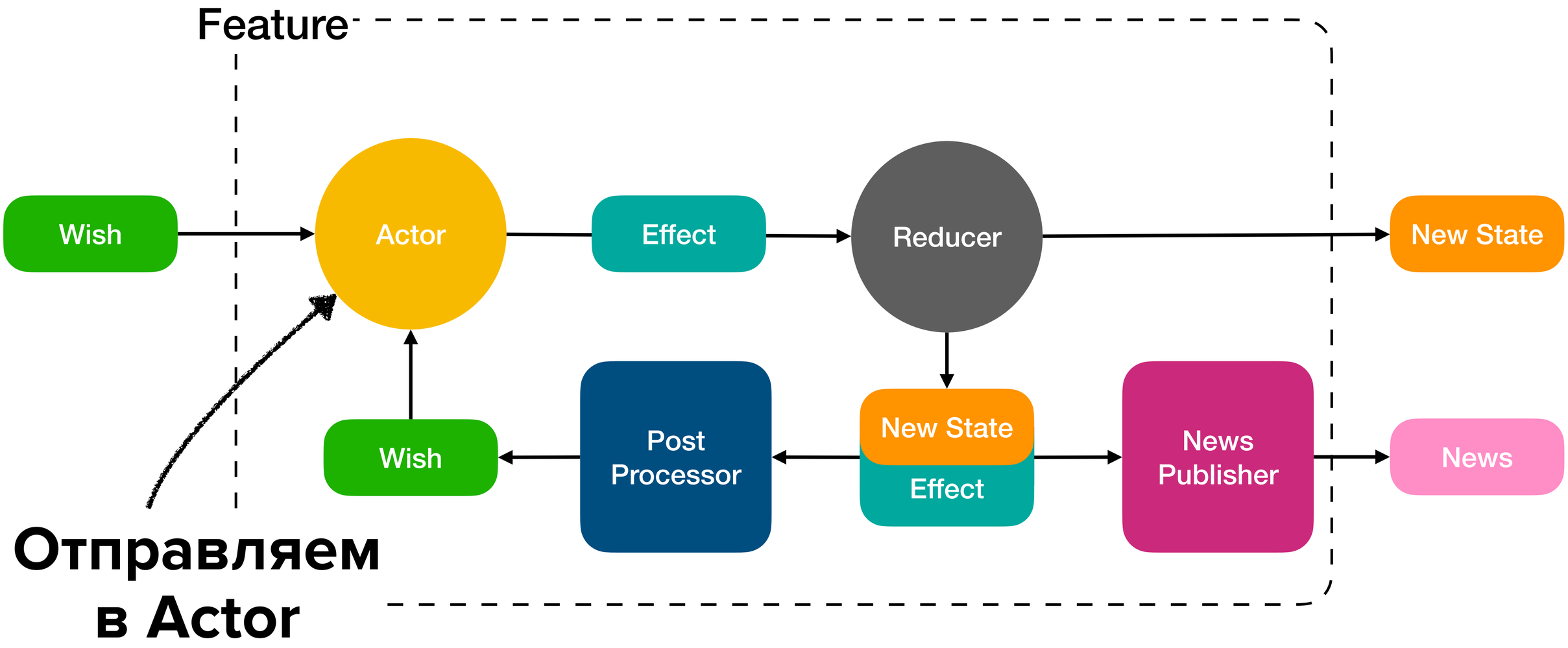


Эффект с новым стейтом

PostProcessor

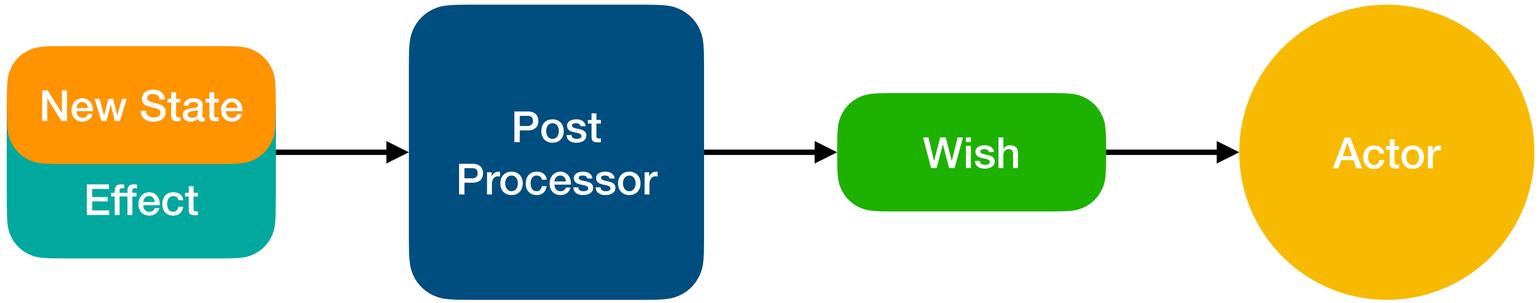


PostProcessor

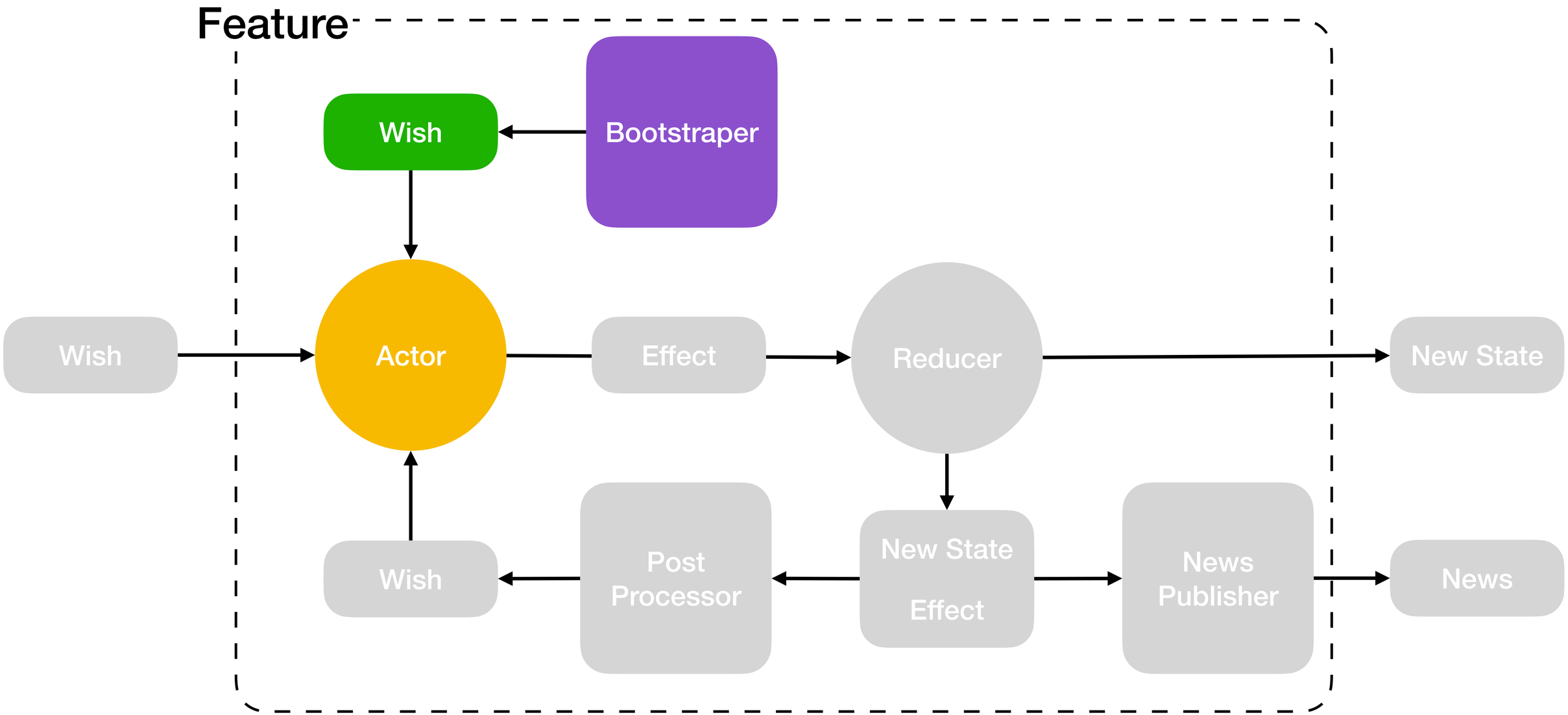


PostProcessor

1. Отправка *Wish* в *Actor* после изменения стейта



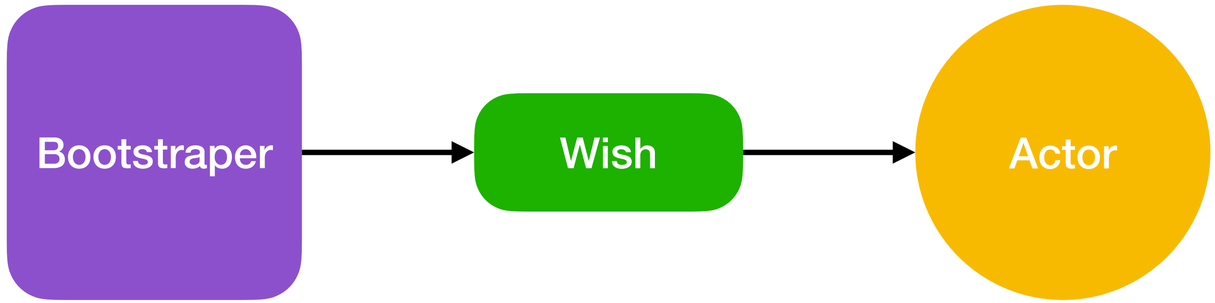
Bootstraper



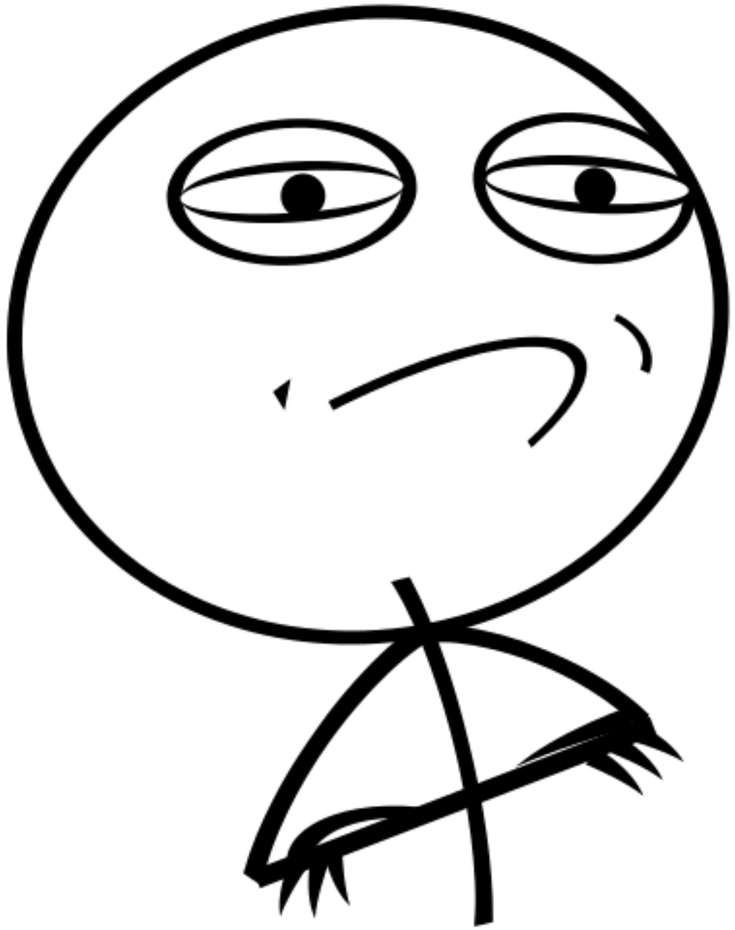
Bootstraper

1. Подписка на внешние сервисы

2. Отправка *Wish* в *Actor* после сигнала из вне



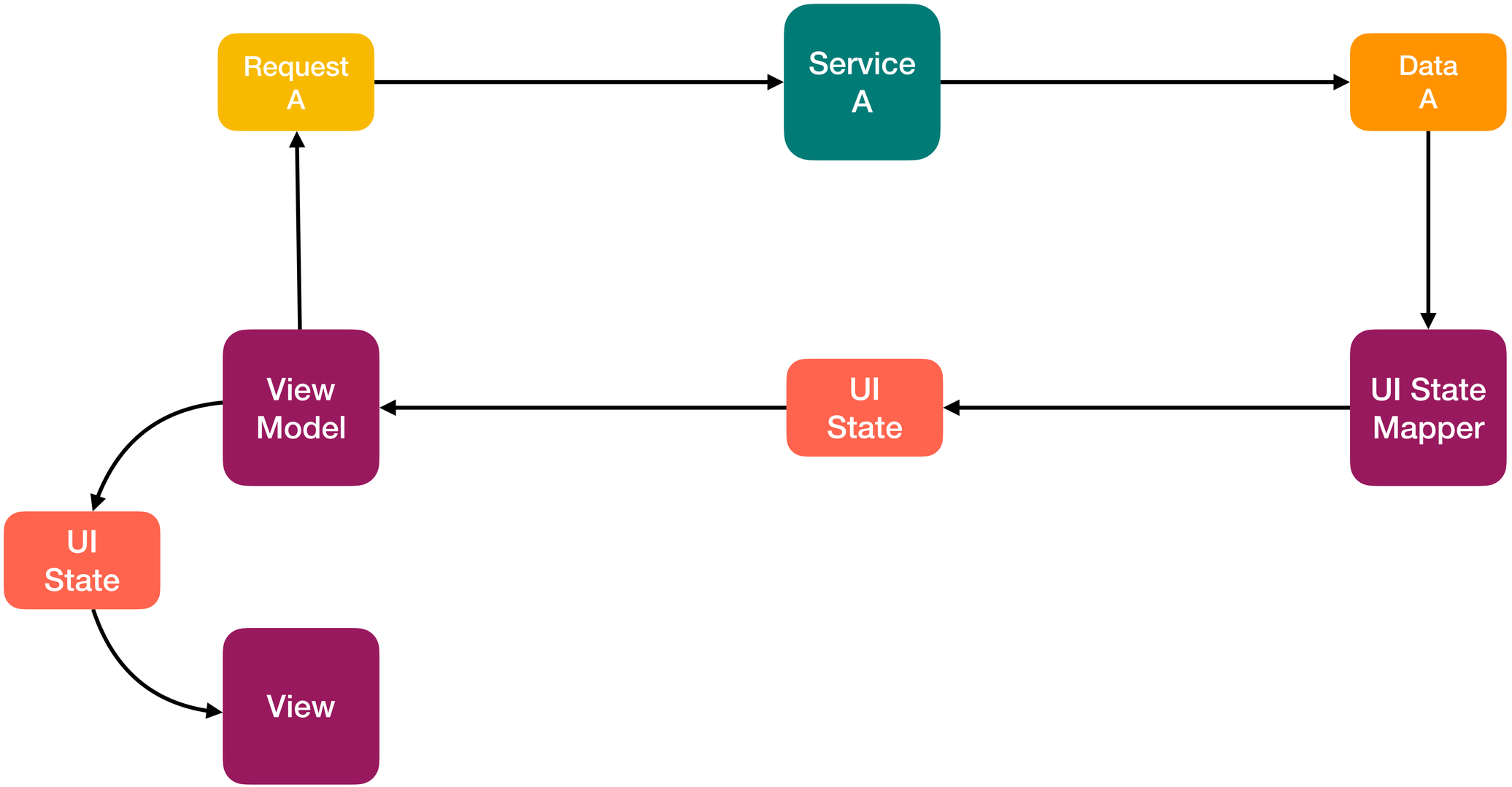
Выглядит сложновато



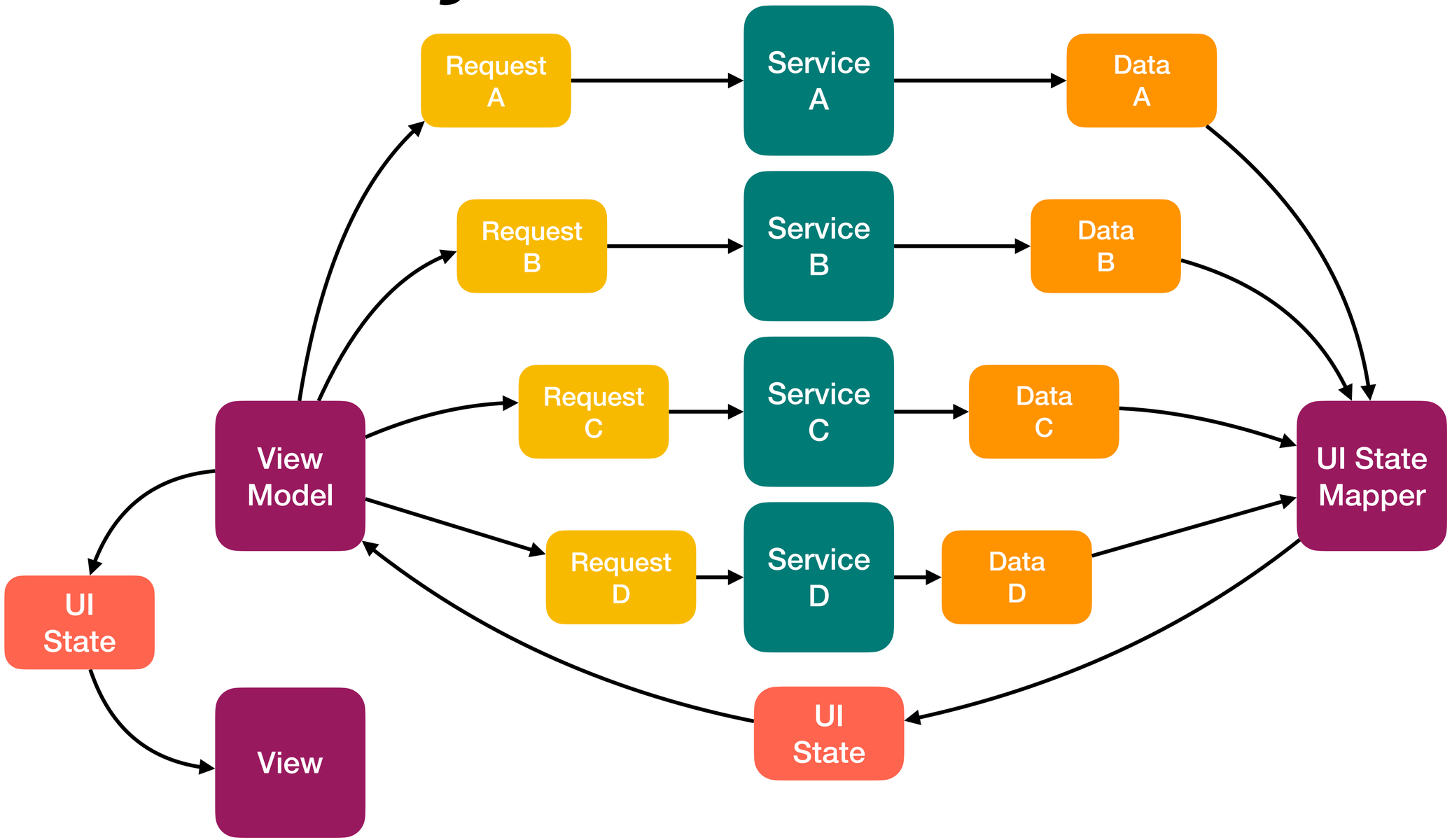
Но мы справимся


Используем
MVI в проде

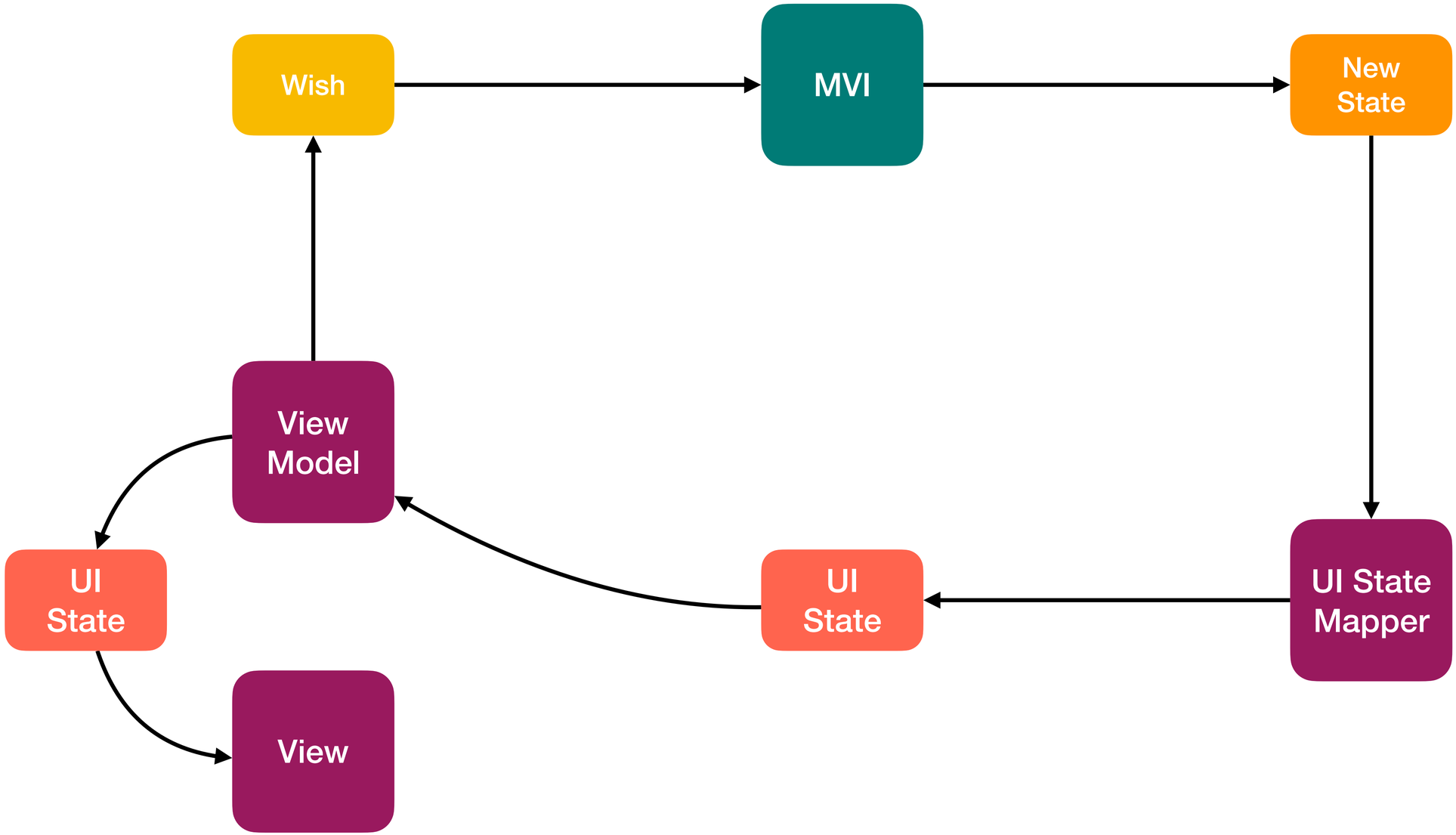
Продуктовые фичи без MVI



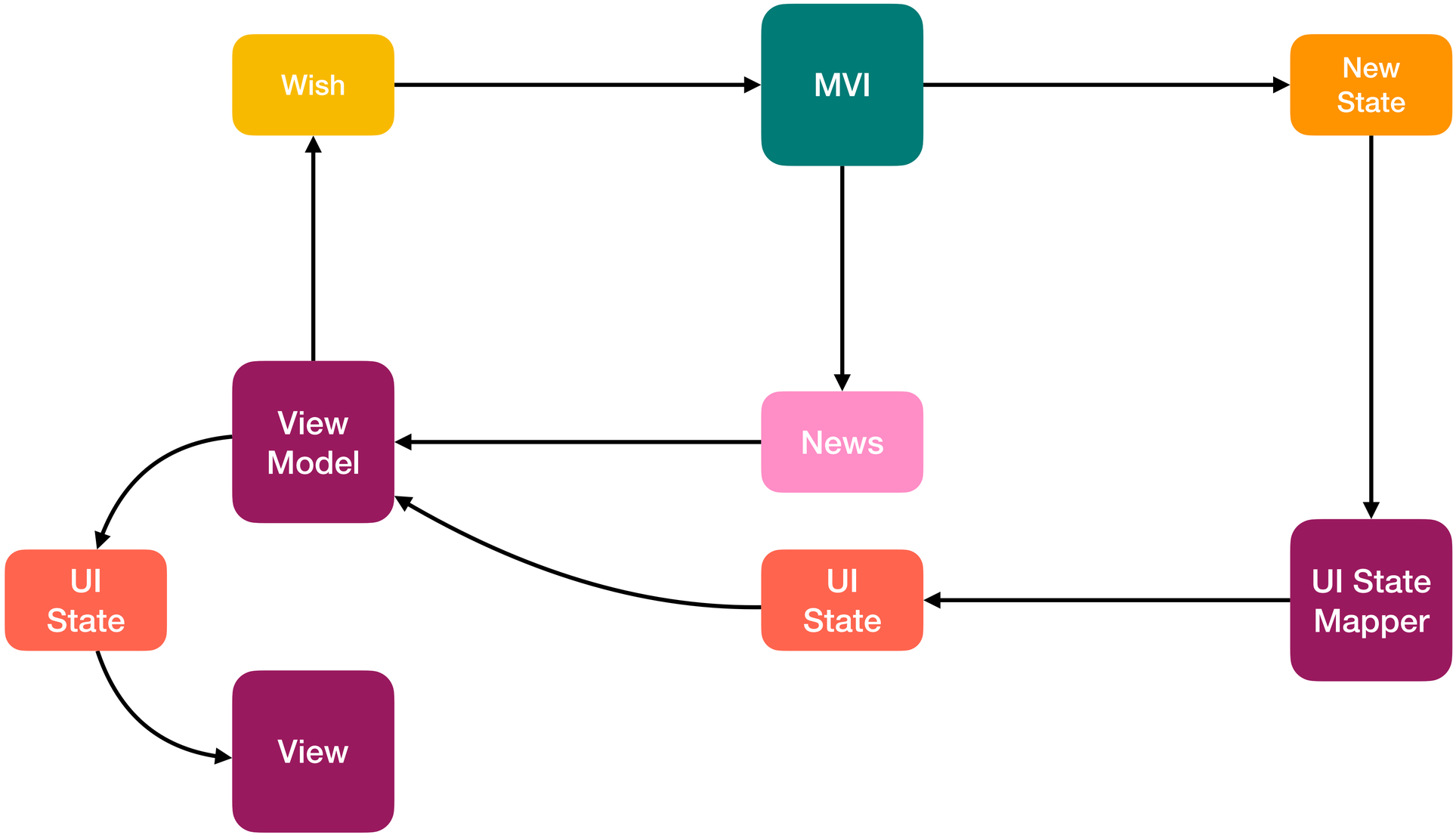
Немного усложним



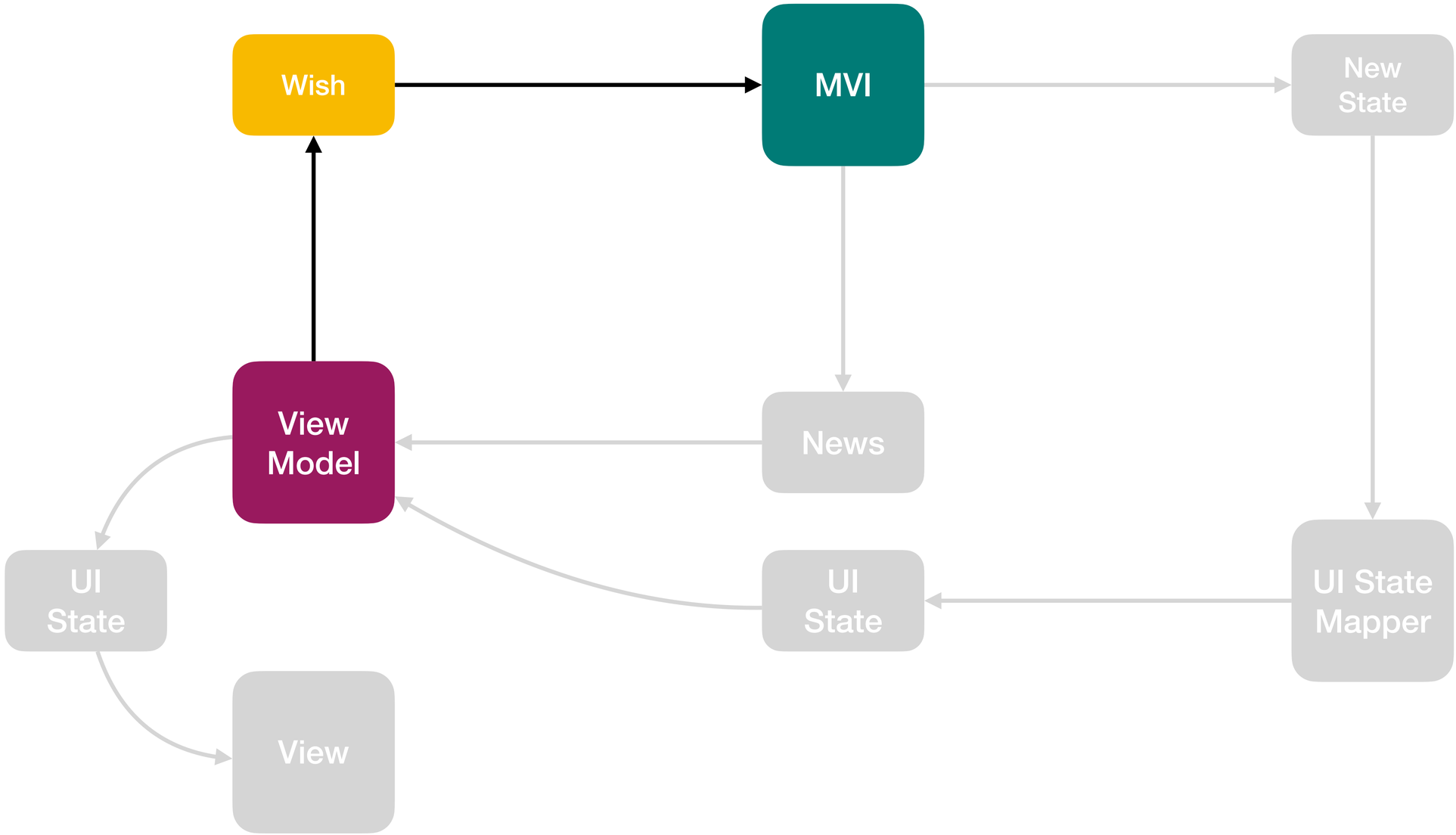
Фича с MVI


Wish
MVI
New State
View Model
UI State
UI State
UI State Mapper
View

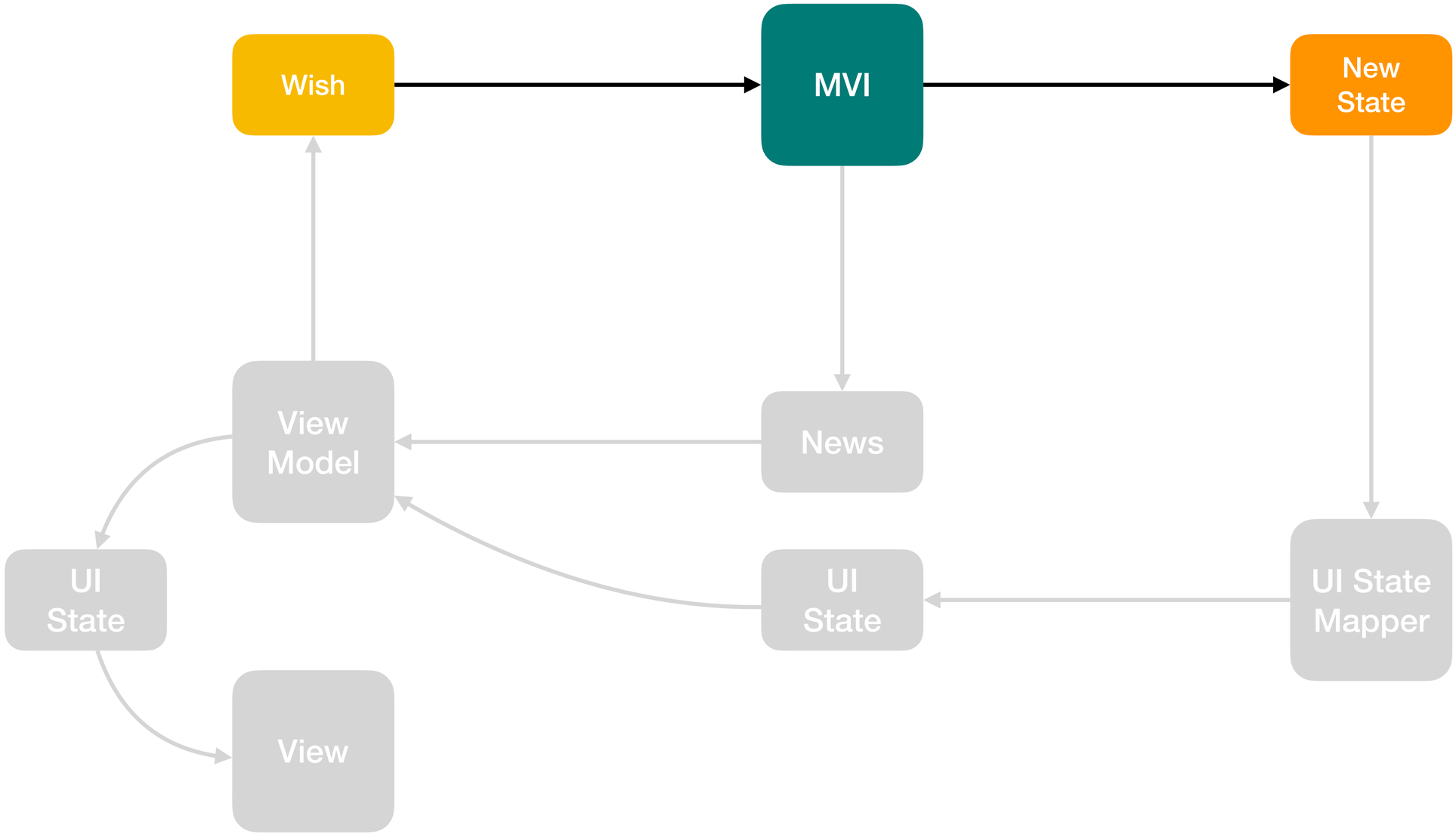
Фича с MVI + News



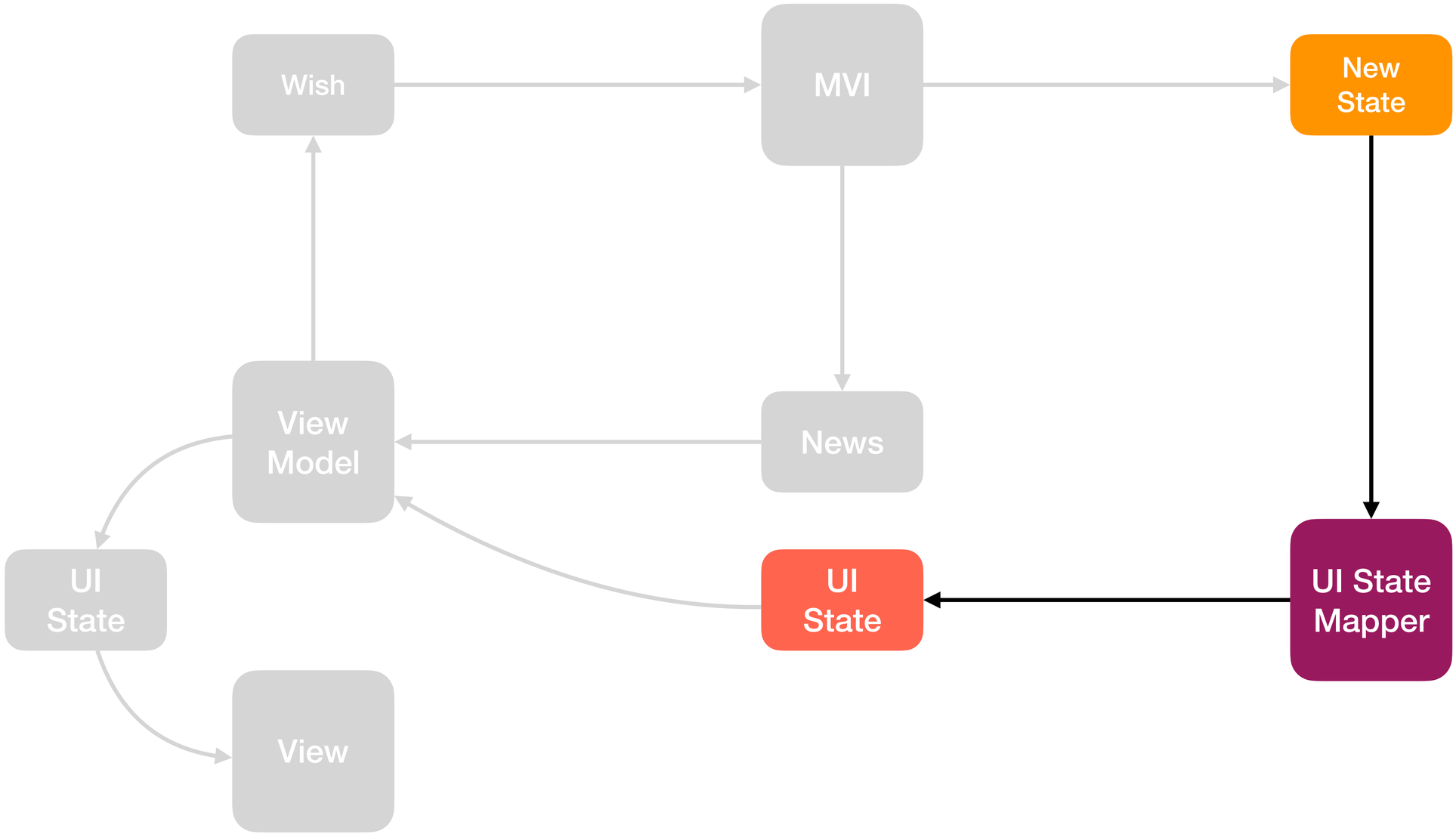
Продуктовые фичи с MVI



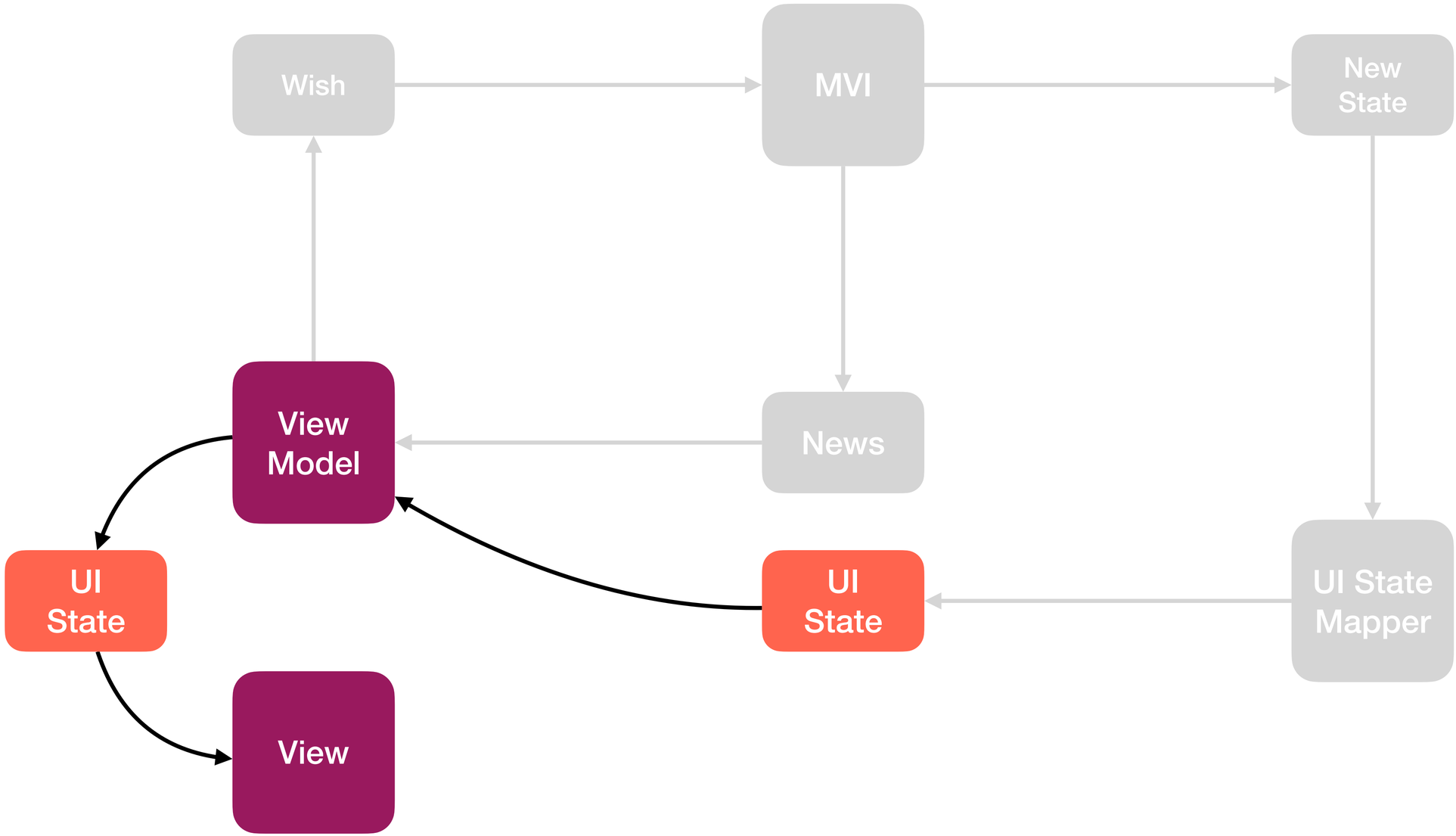
Продуктовые фичи с MVI



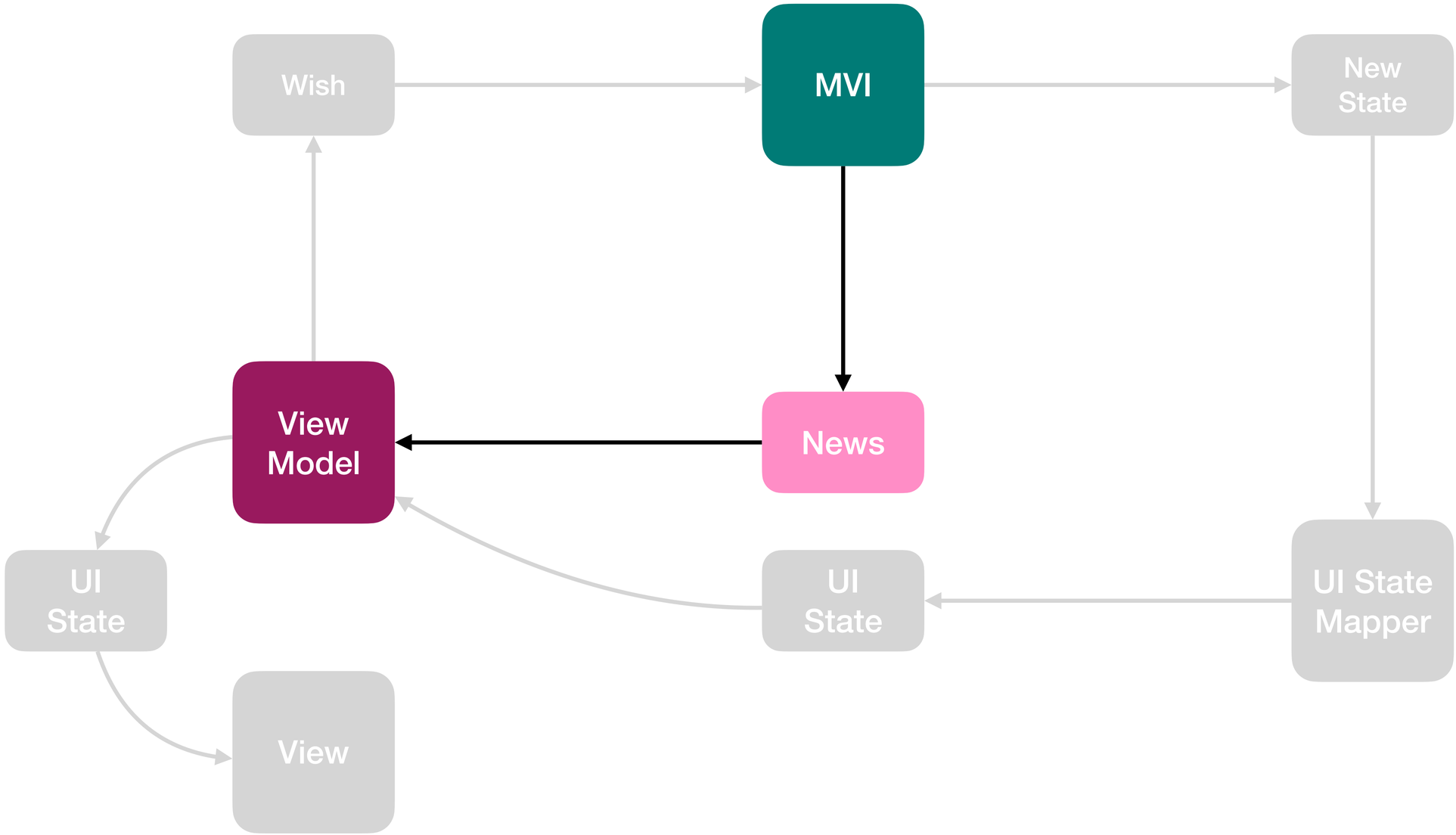
Обновим UI

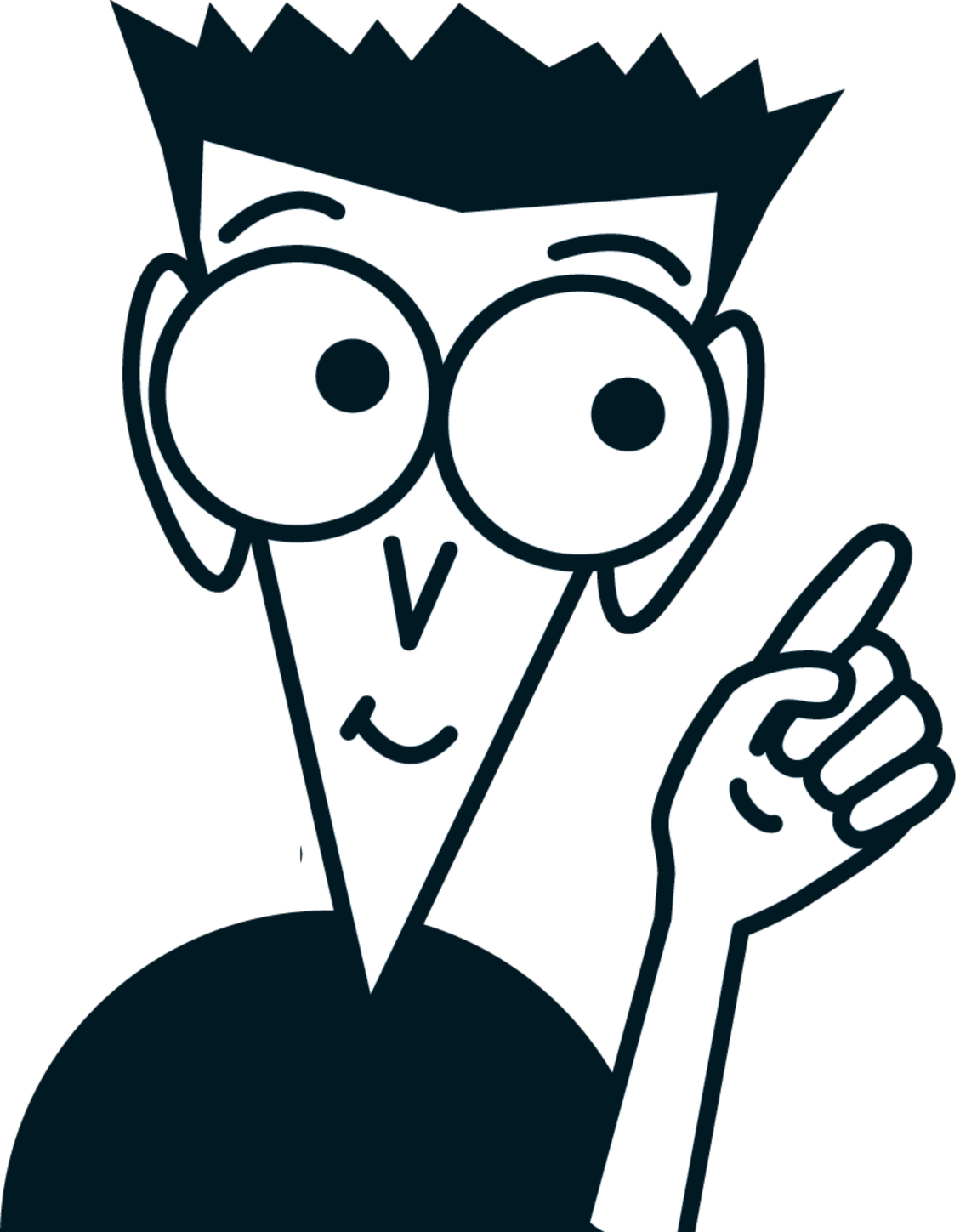

Wish
MVI
New State
View Model
News
UI State
UI State
UI State Mapper
View

Обновим UI

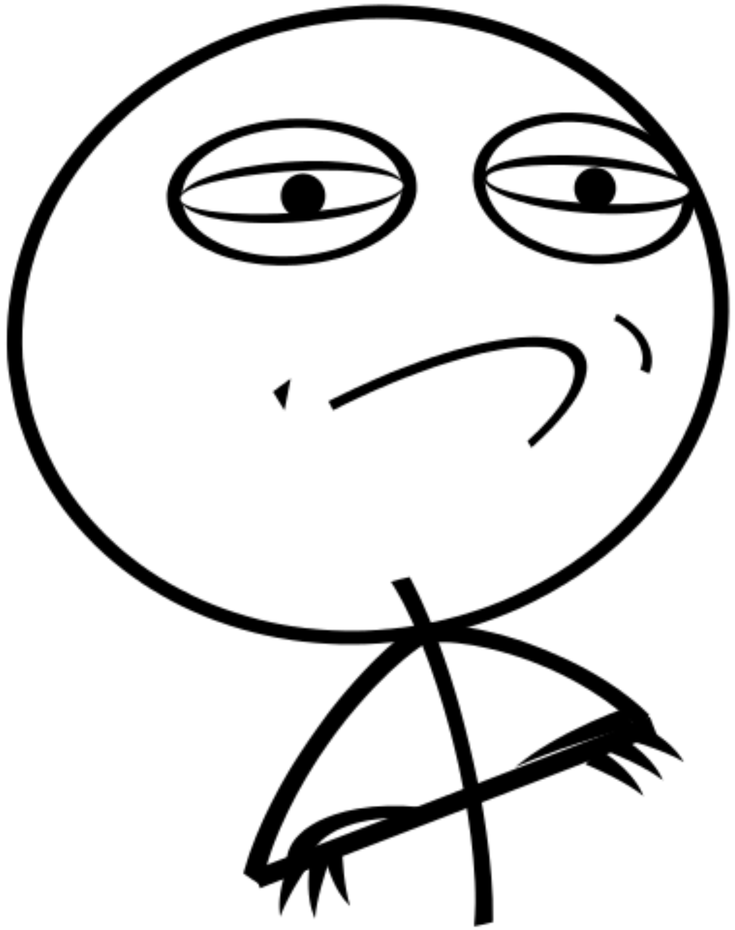

Wish
MVI
New State
View Model
News
UI State
UI State
UI State Mapper
View

Отправим News





Немного кода

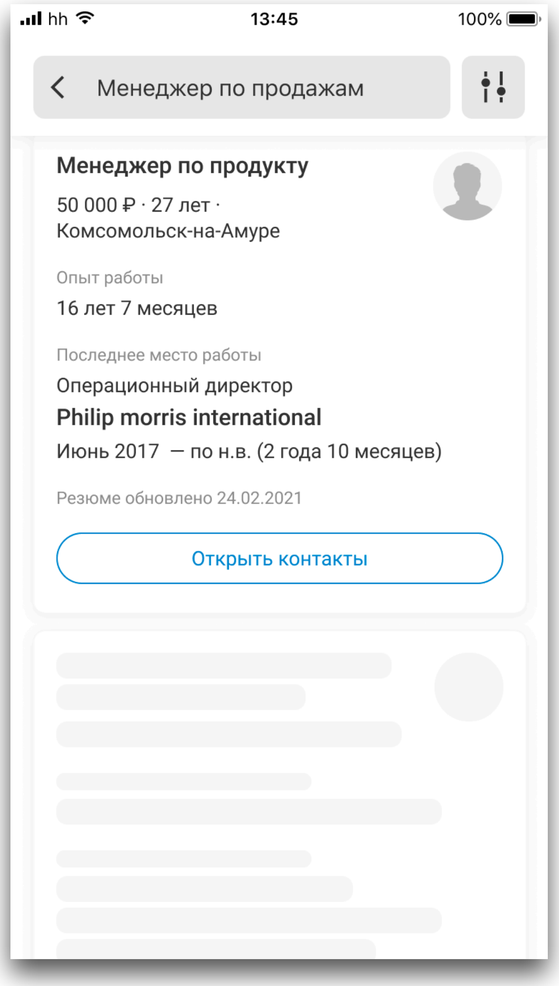


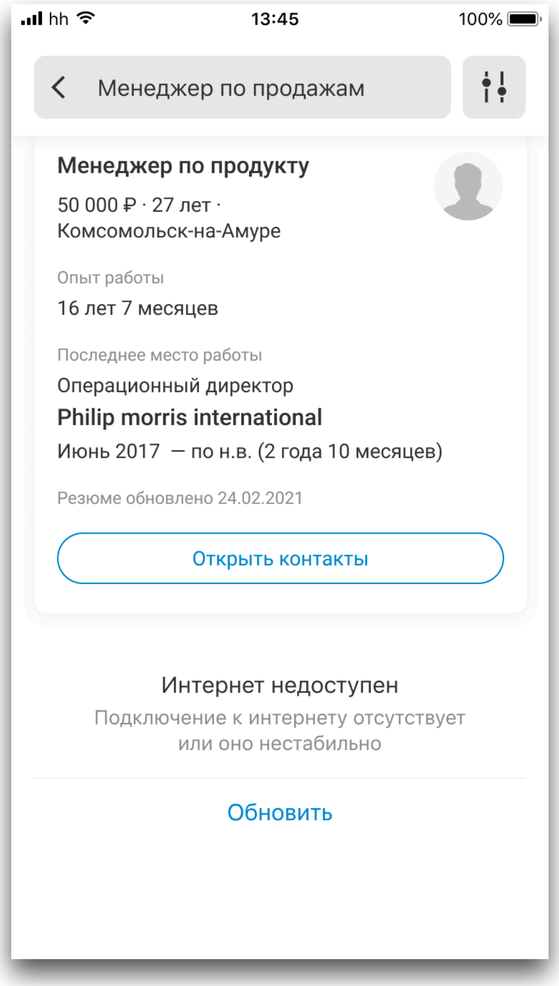
Сделаем как мы

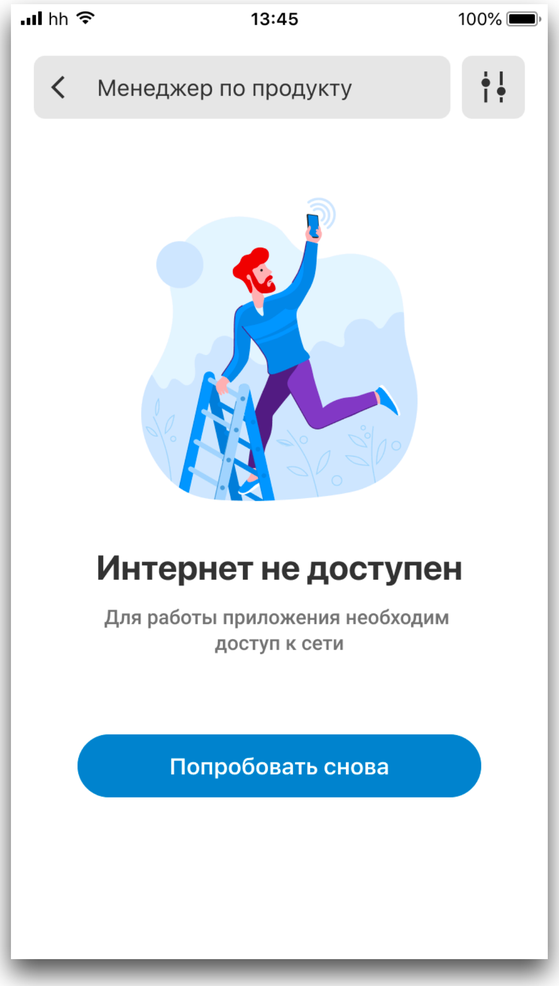
Pagination

Paginator


hh
13:45
100%
Менеджер по продажам


hh
13:45
100%
Менеджер по продажам
Менеджер по продукту
50 000 ₽ · 27 лет ·
Комсомольск-на-Амуре
Опыт работы
16 лет 7 месяцев
Последнее место работы
Операционный директор
Philip morris international
Июнь 2017 — по н.в. (2 года 10 месяцев)
Резюме обновлено 24.02.2021
Открыть контакты


hh
13:45
100%
Менеджер по продажам
Менеджер по продукту
50 000 ₽ · 27 лет ·
Комсомольск-на-Амуре
Опыт работы
16 лет 7 месяцев
Последнее место работы
Операционный директор
Philip morris international
Июнь 2017 — по н.в. (2 года 10 месяцев)
Резюме обновлено 24.02.2021
Открыть контакты
Интернет недоступен
Подключение к интернету отсутствует
или оно нестабильно
Обновить

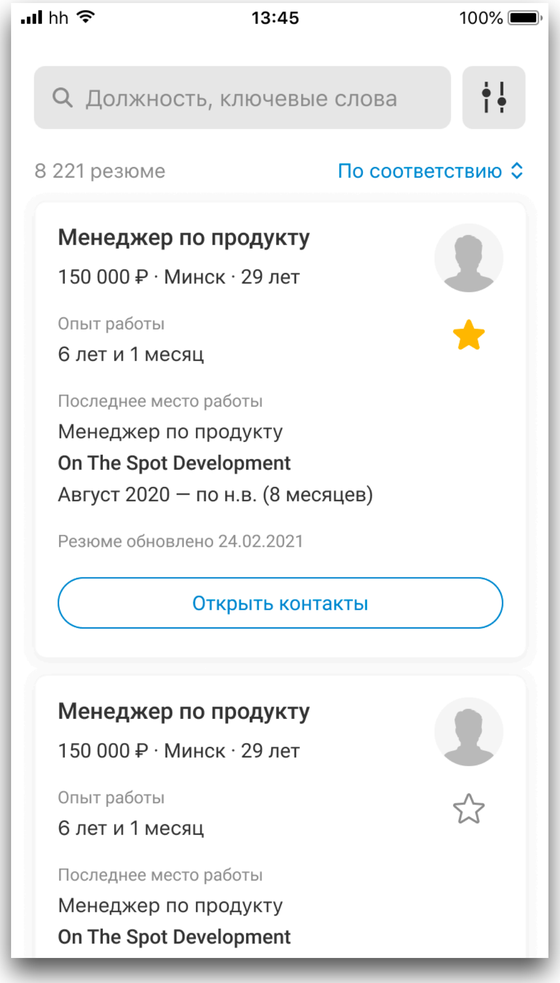

hh
13:45
100%
Менеджер по продукту
Интернет не доступен
Для работы приложения необходим
доступ к сети
Попробовать снова

Что мы хотим

1. **Загружать начальные данные**
2. Отображать эти данных
3. Догружать по страницам
4. Обновлять загруженные данные
5. Отображать ошибки

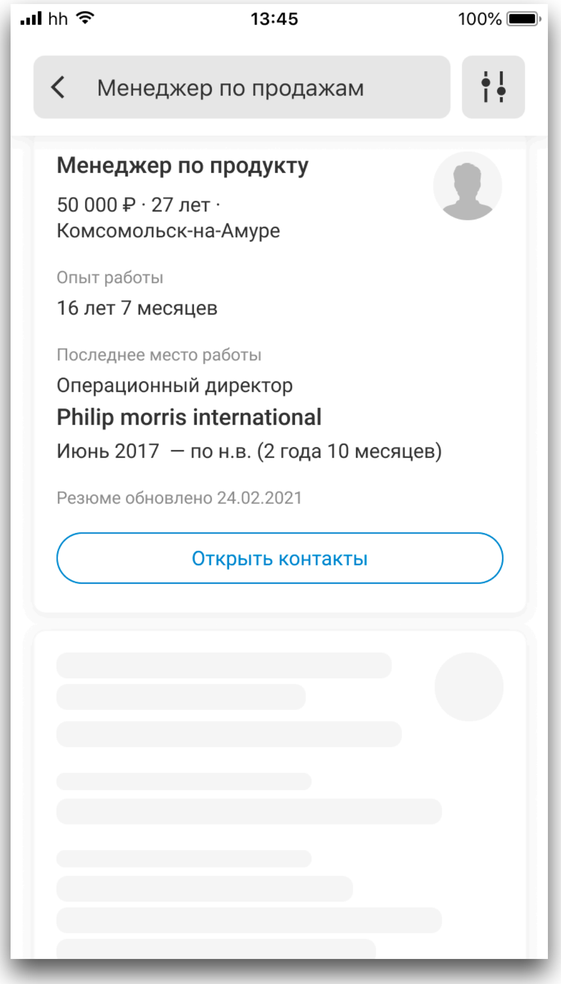
Что мы хотим

1. Загружать начальные данные
2. **Отображать эти данных**
3. Догружать по страницам
4. Обновлять загруженные данные
5. Отображать ошибки



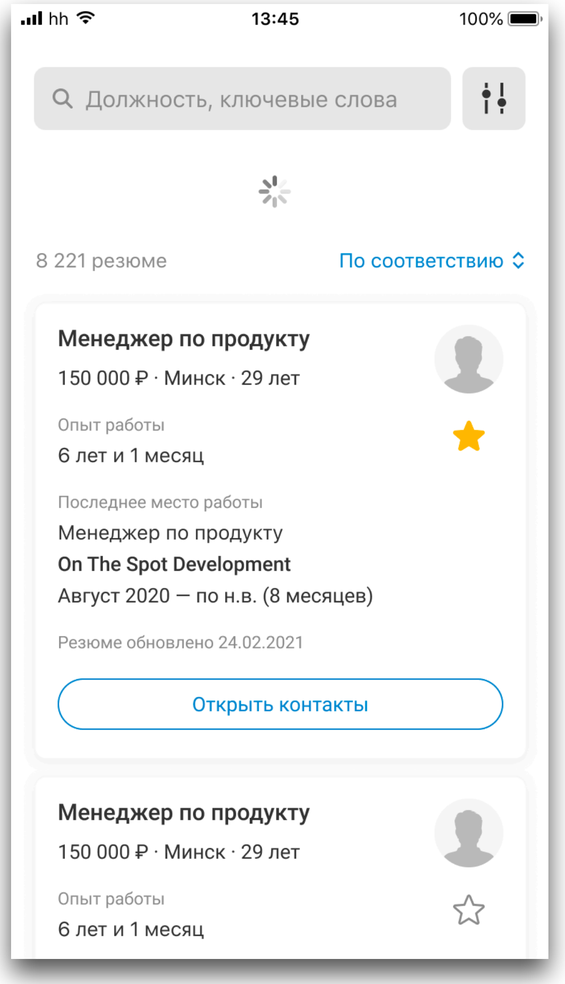
Что мы хотим

1. Загружать начальные данные
2. Отображать эти данных
3. **Догружать по страницам**
4. Обновлять загруженные данные
5. Отображать ошибки



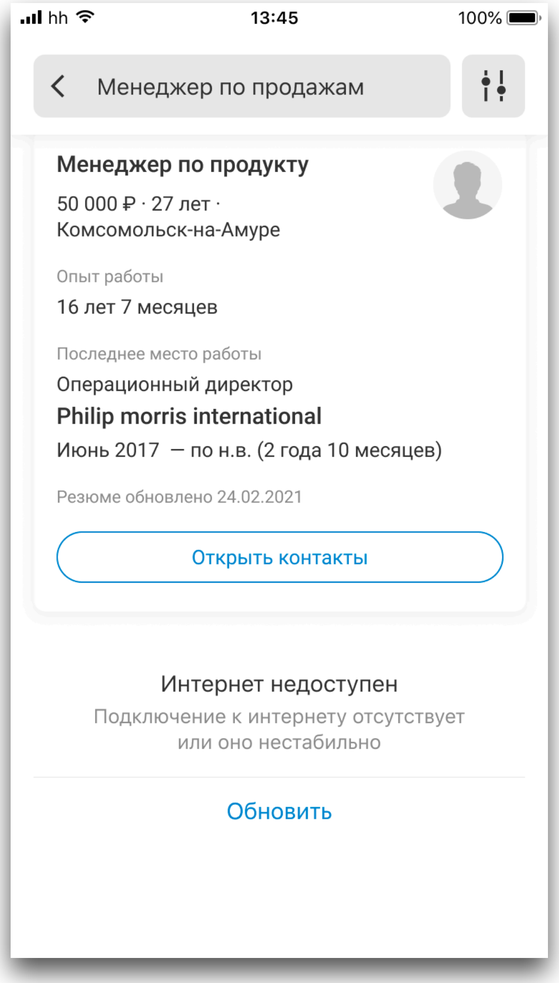
Что мы хотим

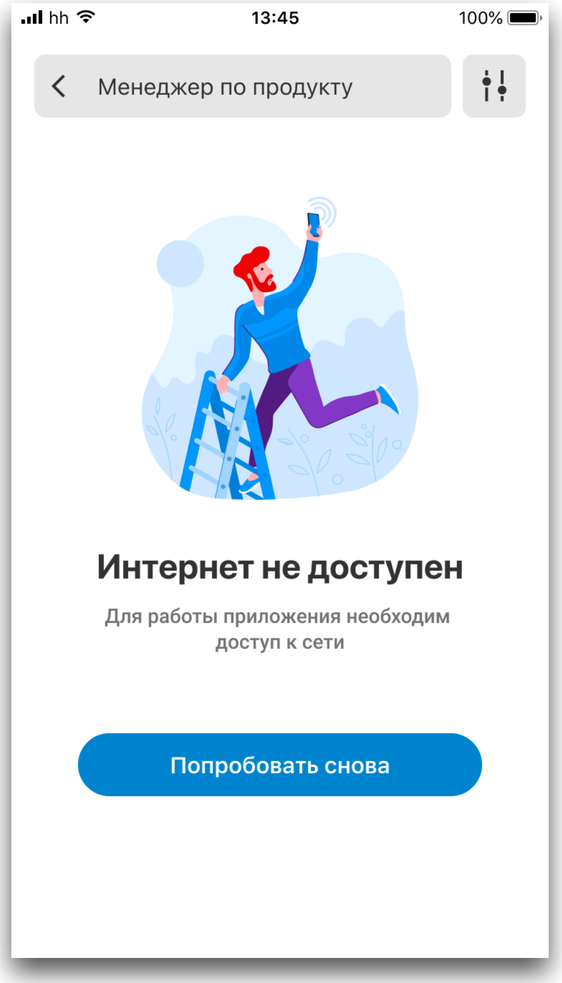
1. Загружать начальные данные
2. Отображать эти данных
3. Догружать по страницам
4. **Обновлять загруженные данные**
5. Отображать ошибки



Что мы хотим

1. Загружать начальные данные
2. Отображать эти данных
3. Догружать по страницам
4. Обновлять загруженные данные
5. **Отображать ошибки**

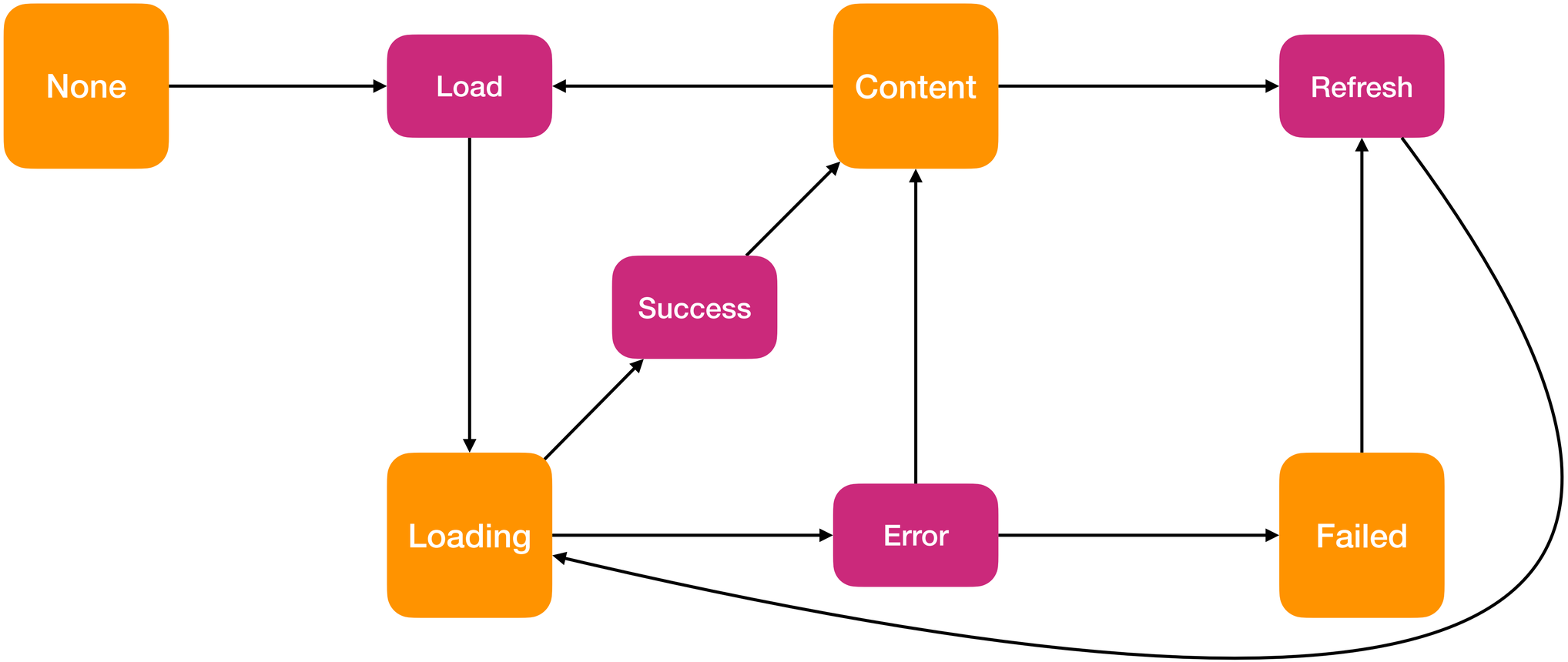




Что мы хотим

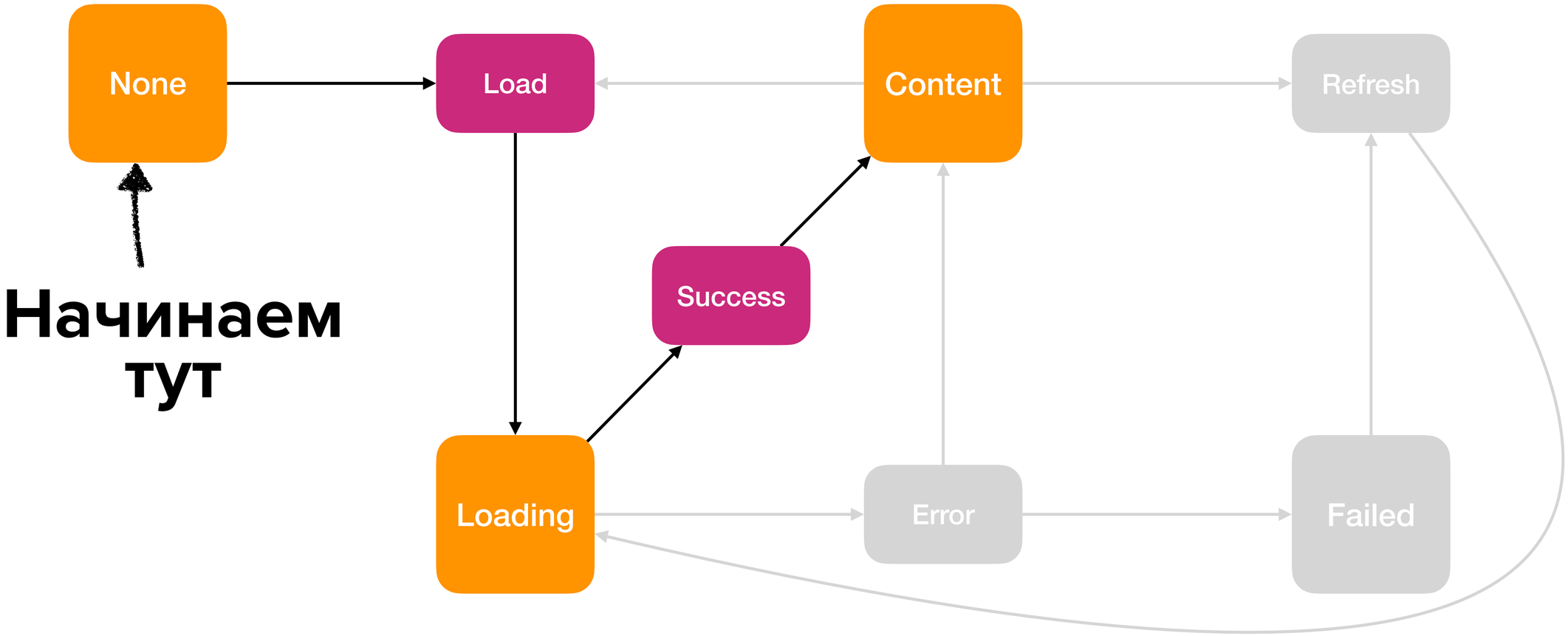
1. Загружать начальные данные
2. Отображать эти данных
3. Догружать по страницам
4. Обновлять загруженные данные
5. Отображать ошибки

Построим схему

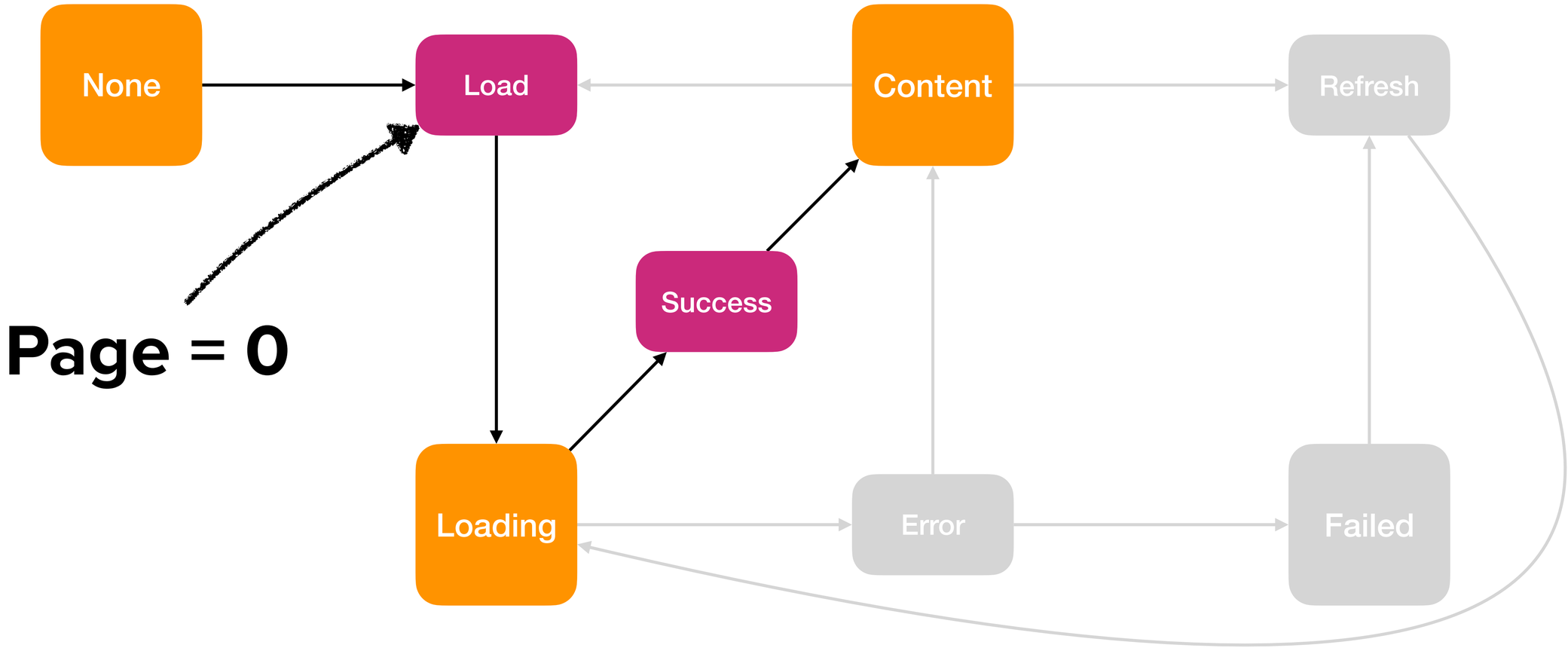


State Event

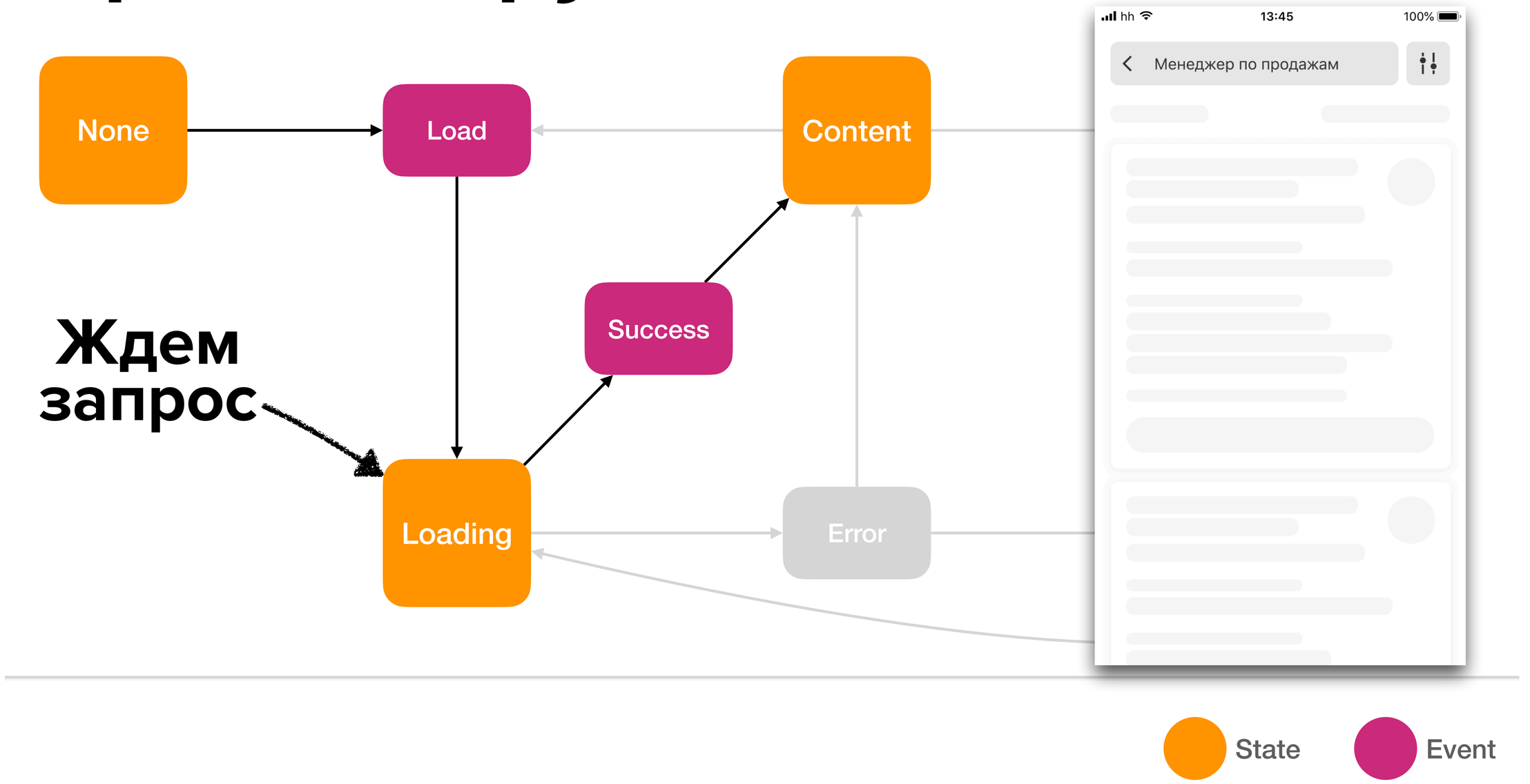
Первая загрузка



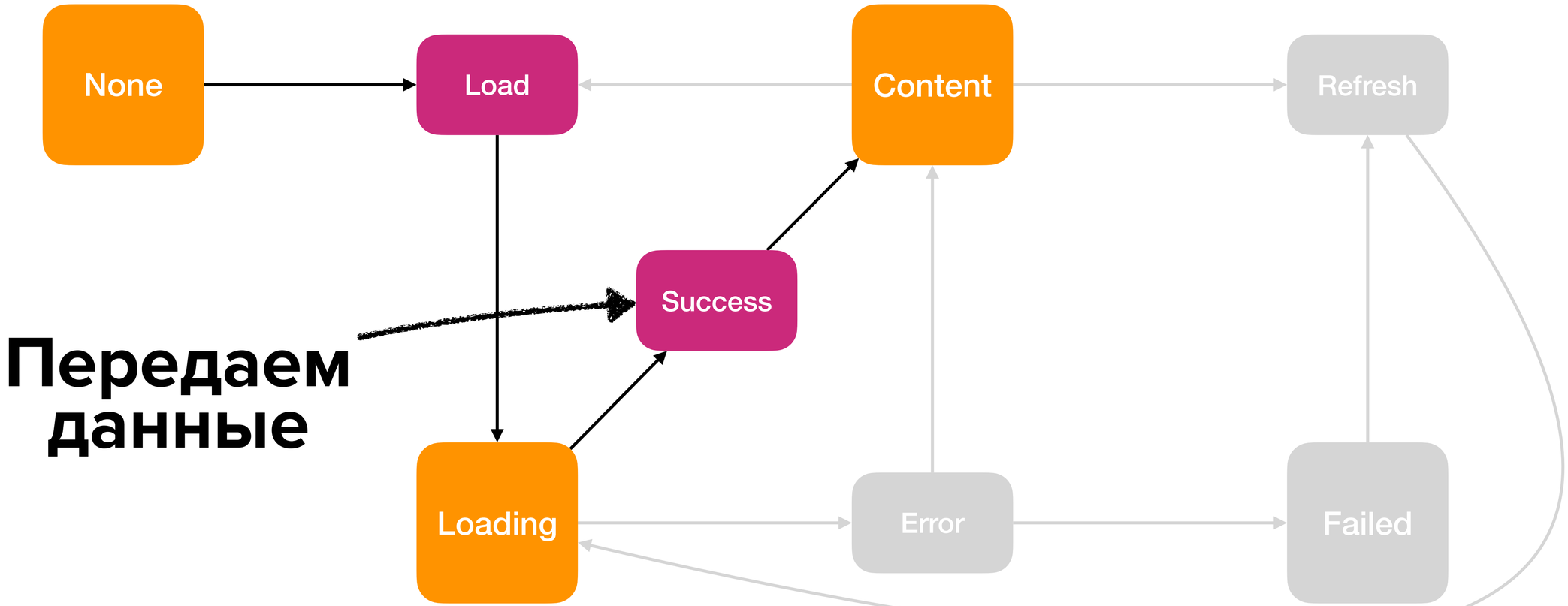
Первая загрузка



Первая загрузка

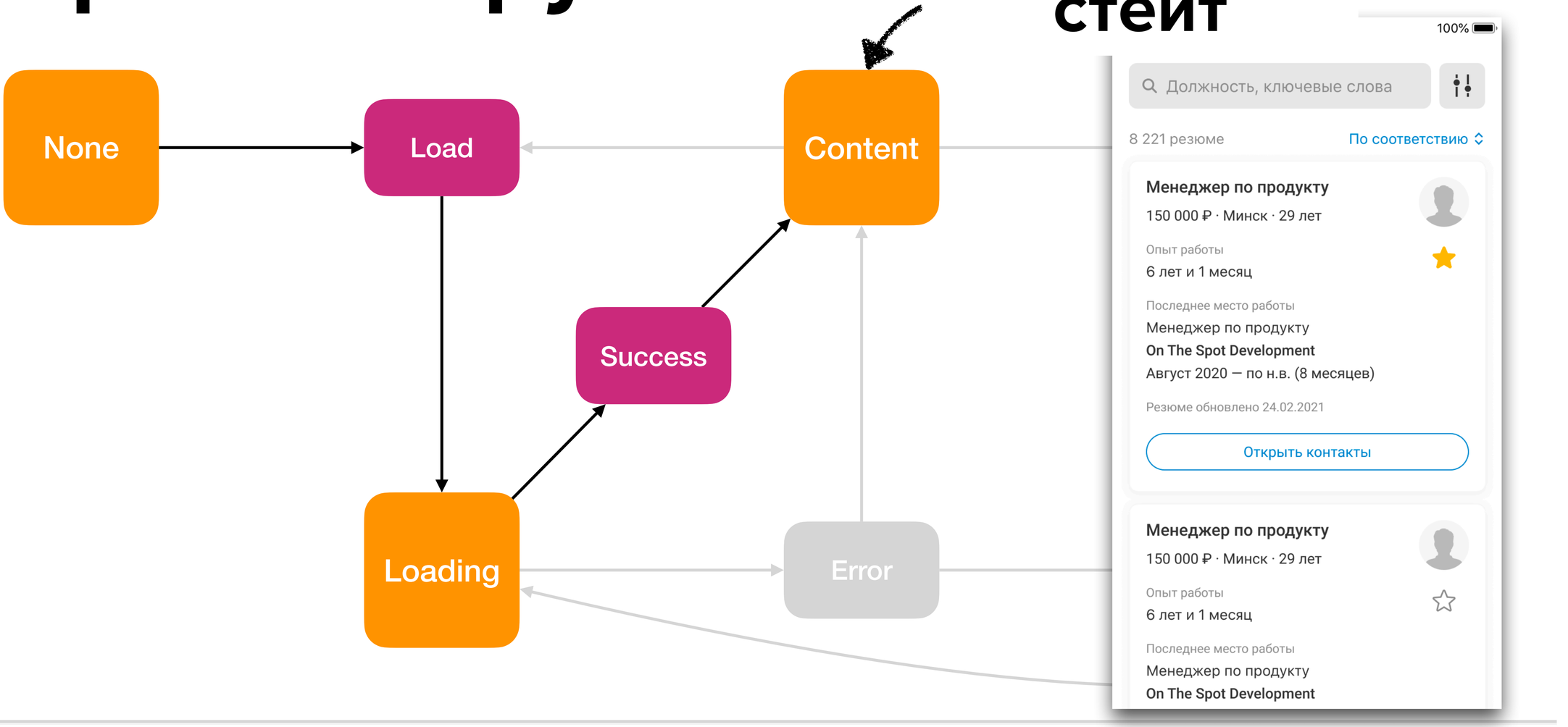


Первая загрузка



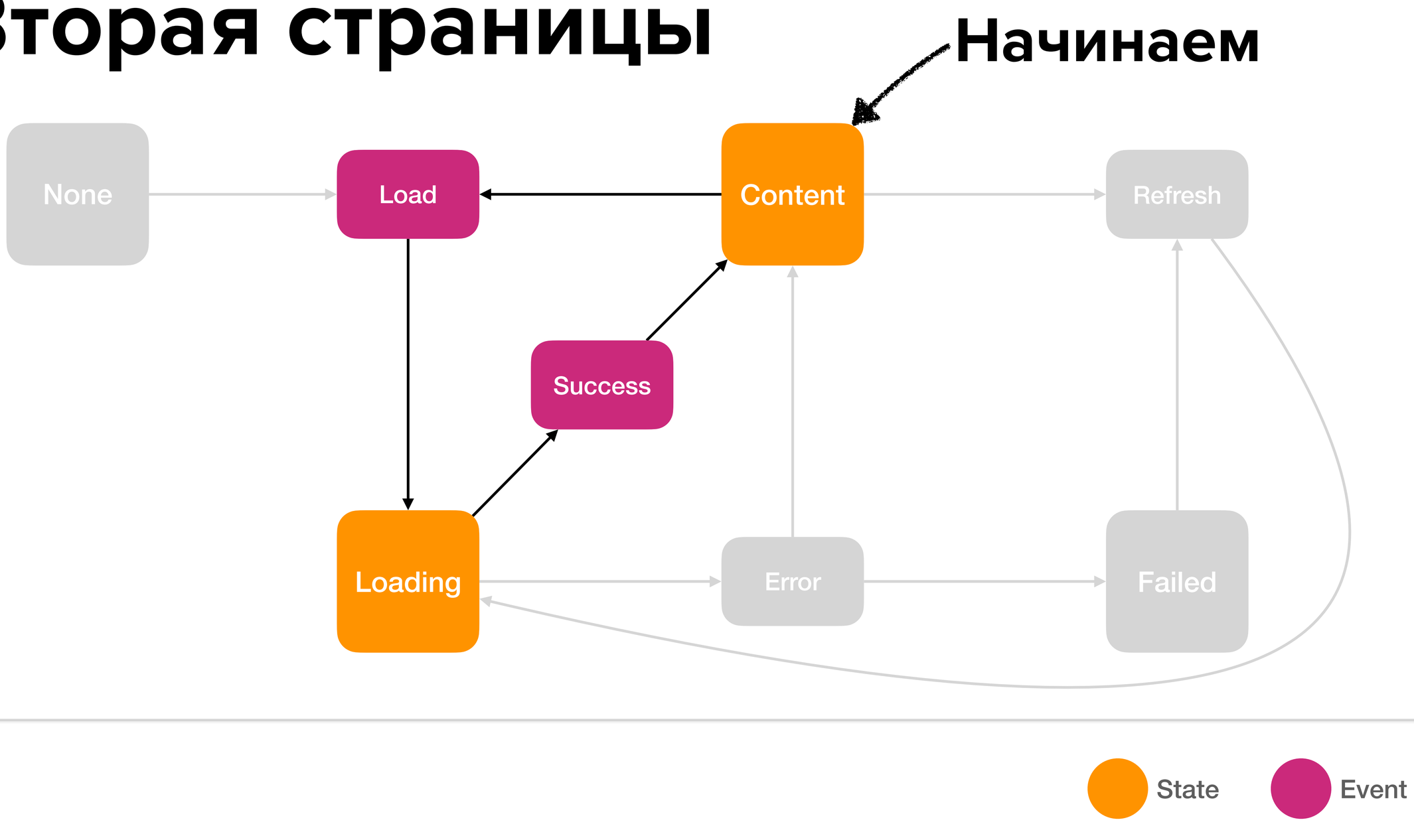
Первая загрузка

Сохраняем в стейт

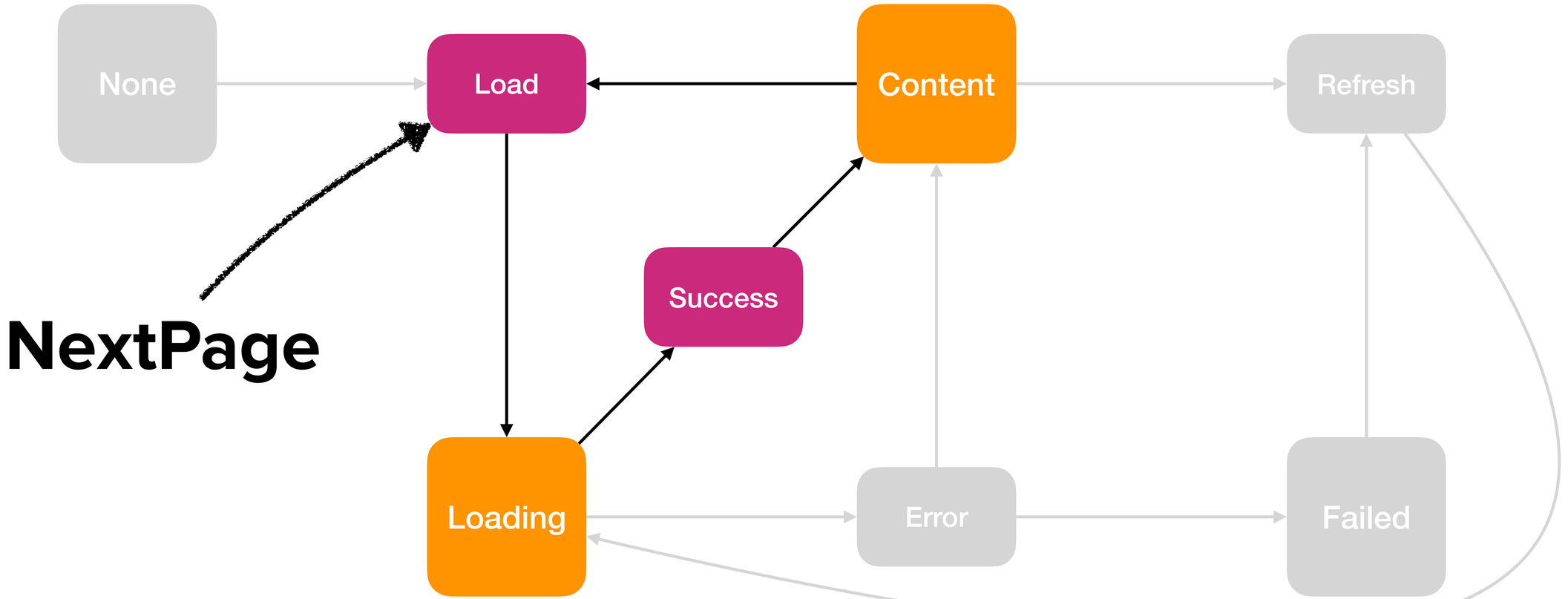


State Event

Вторая страницы

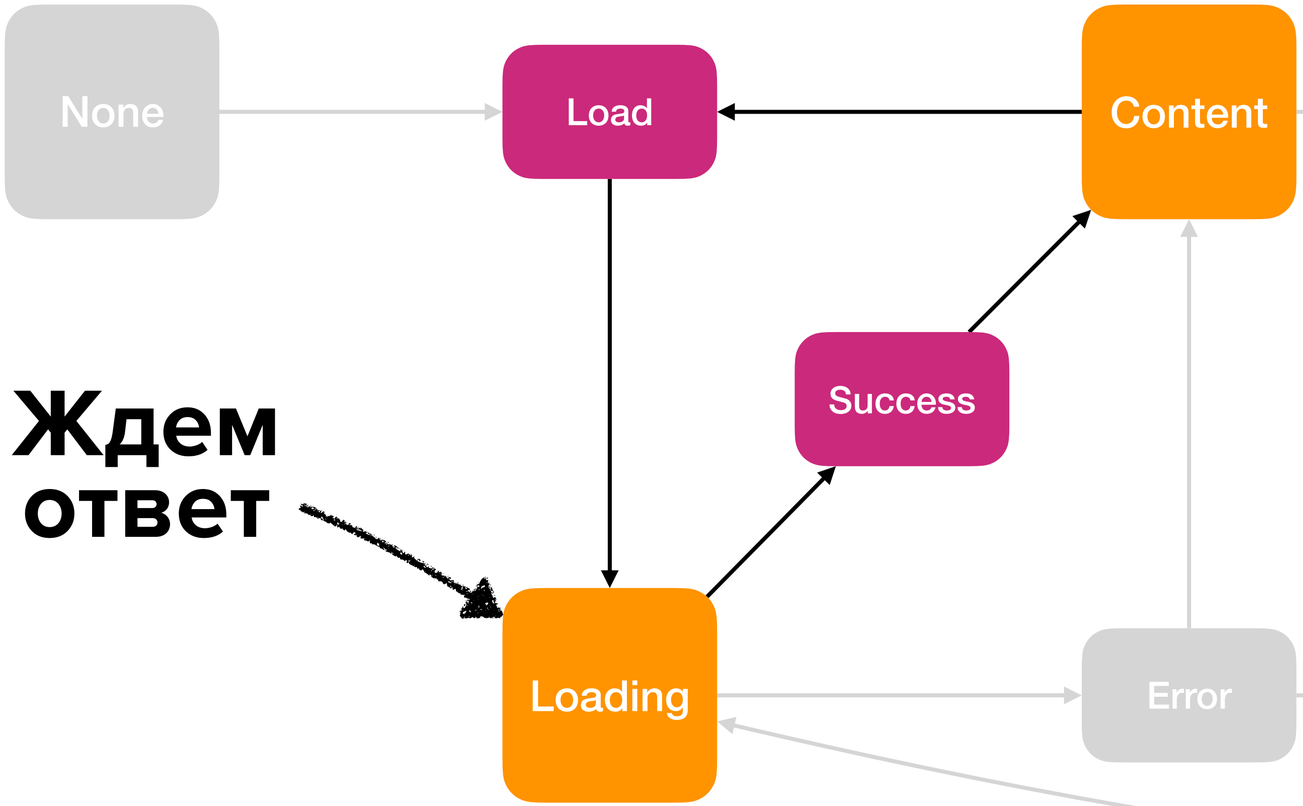


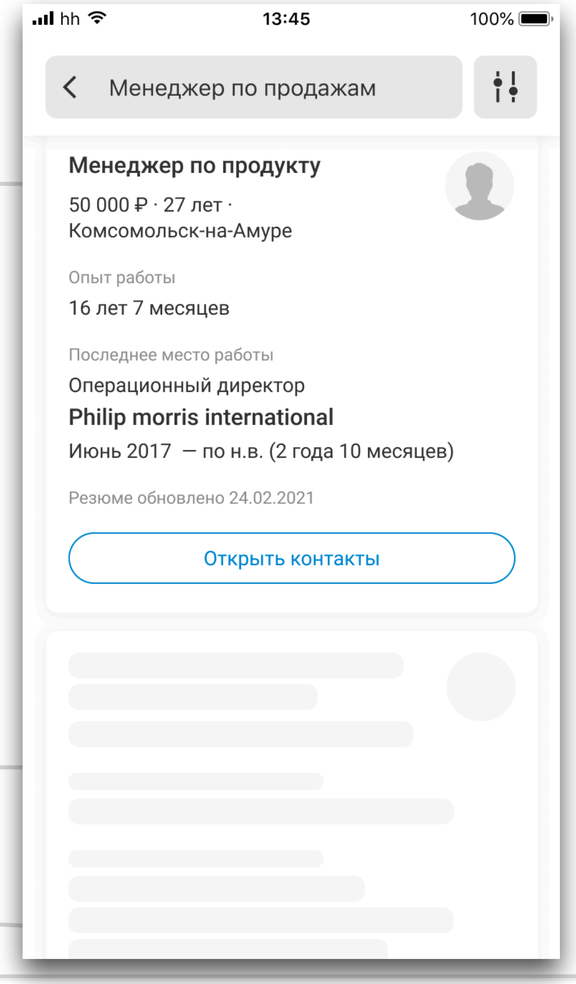
Вторая страницы



State Event

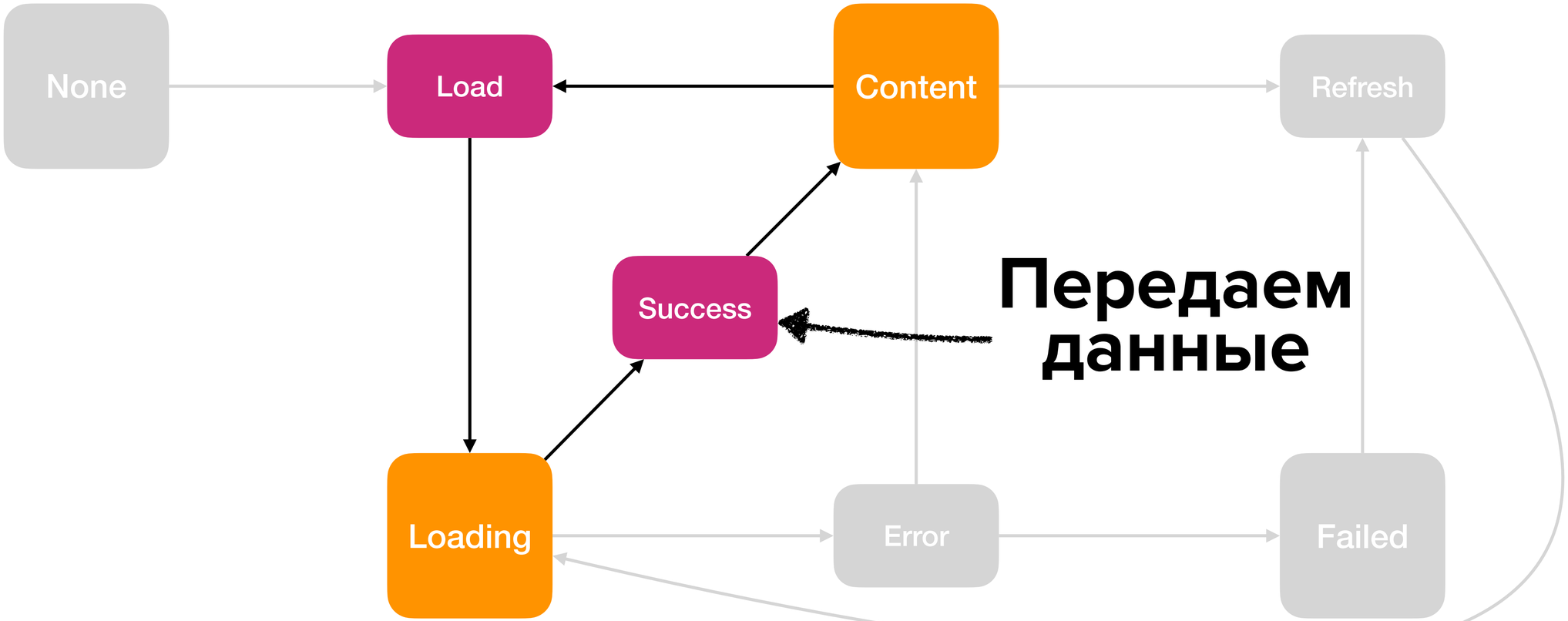
Вторая страницы





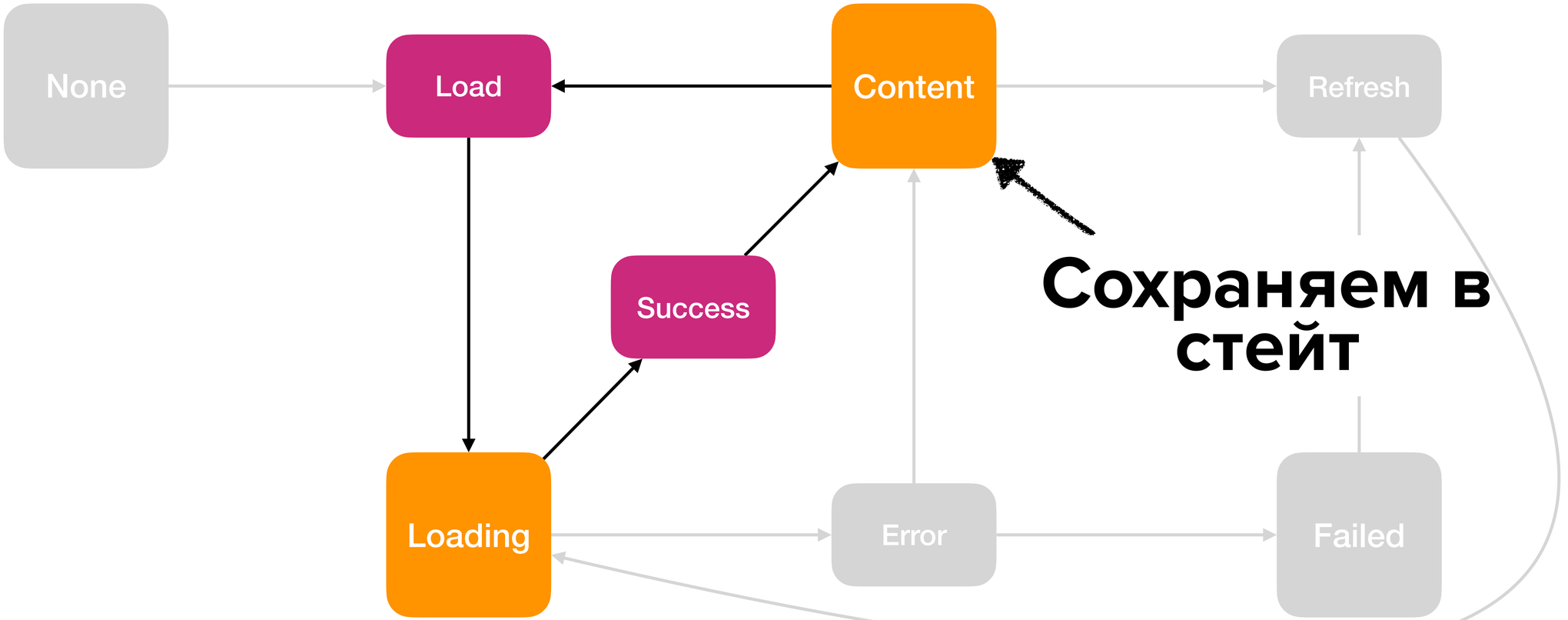
State Event

Вторая страницы



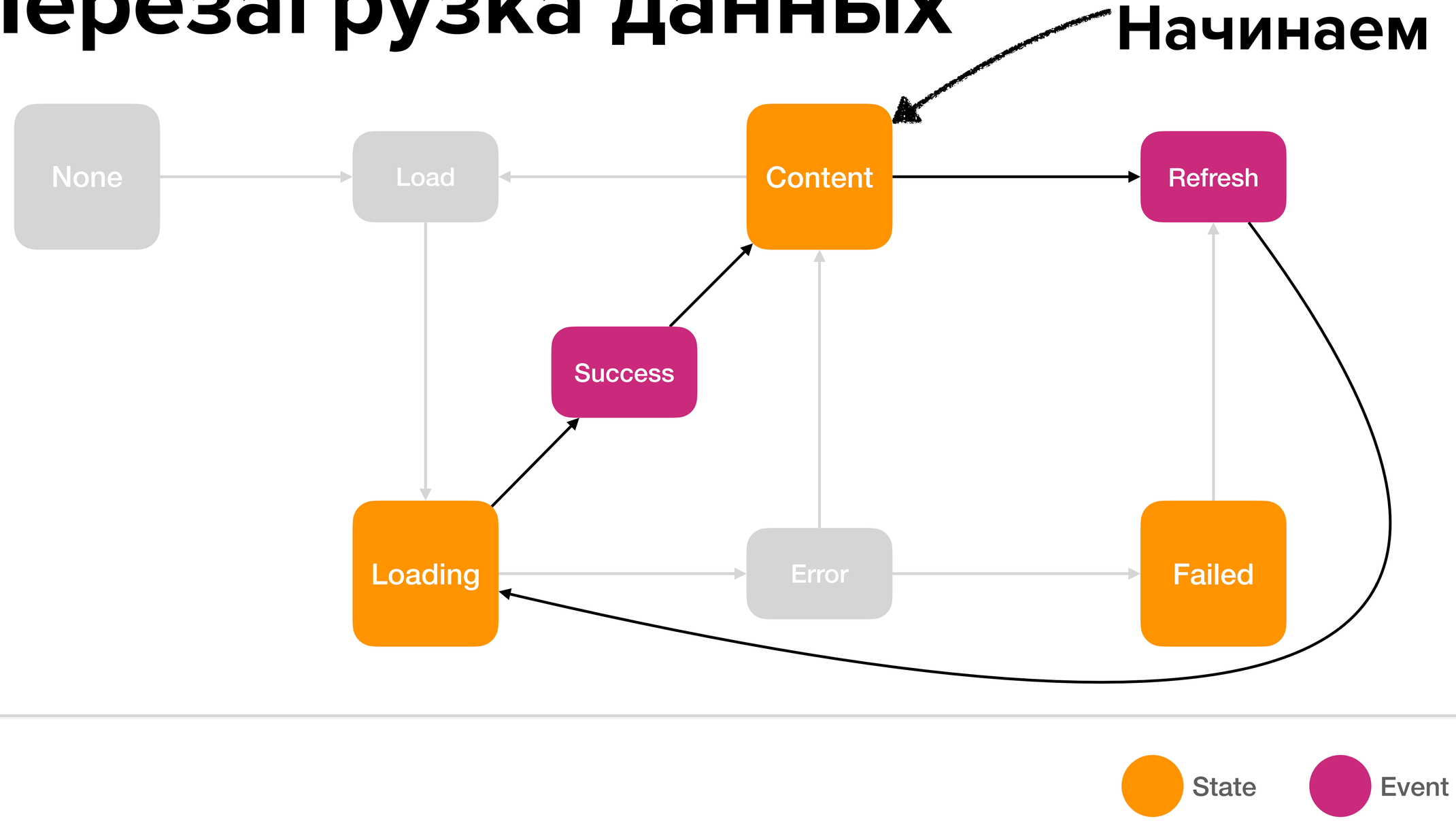
State Event

Вторая страницы

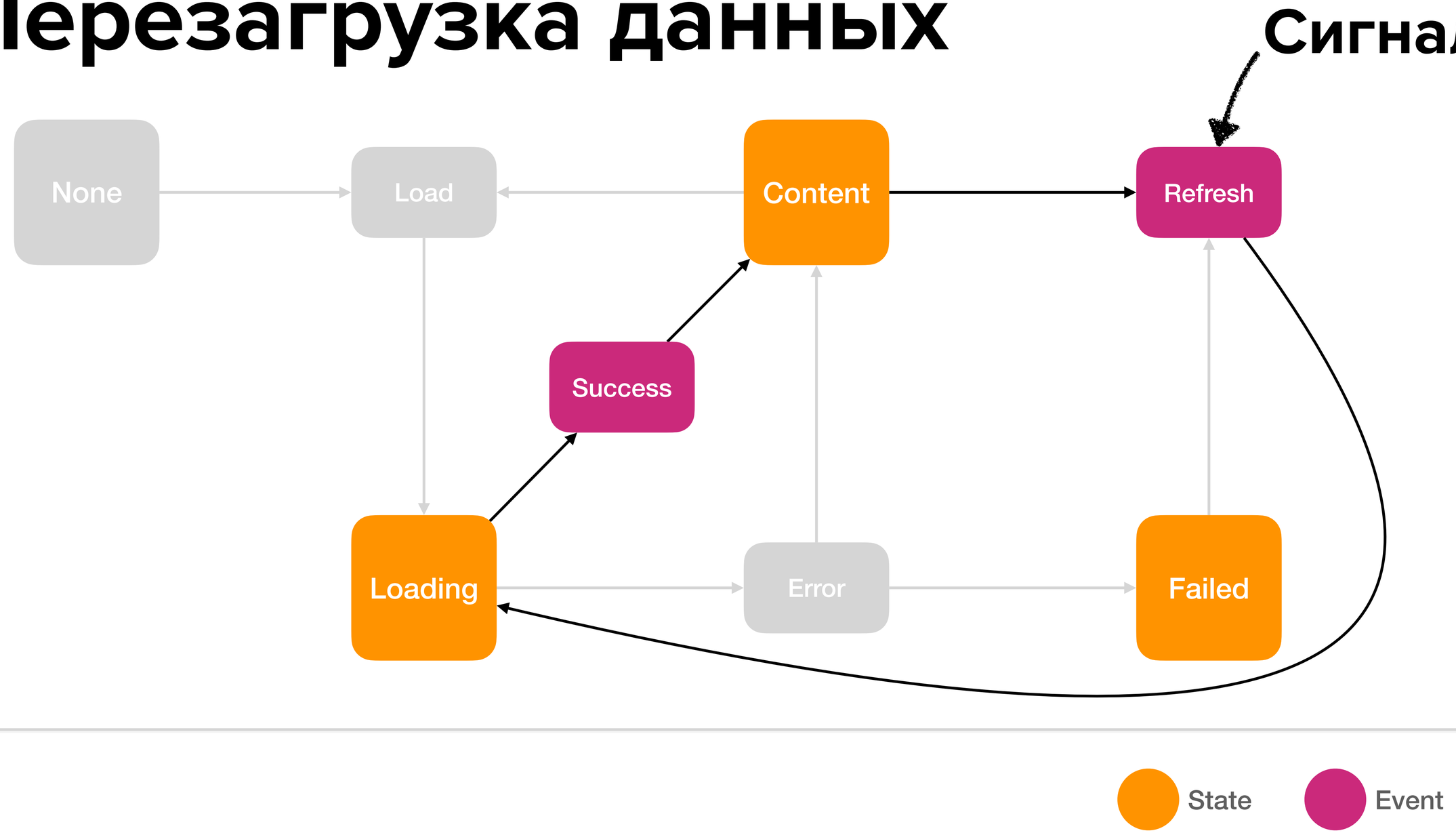


State Event

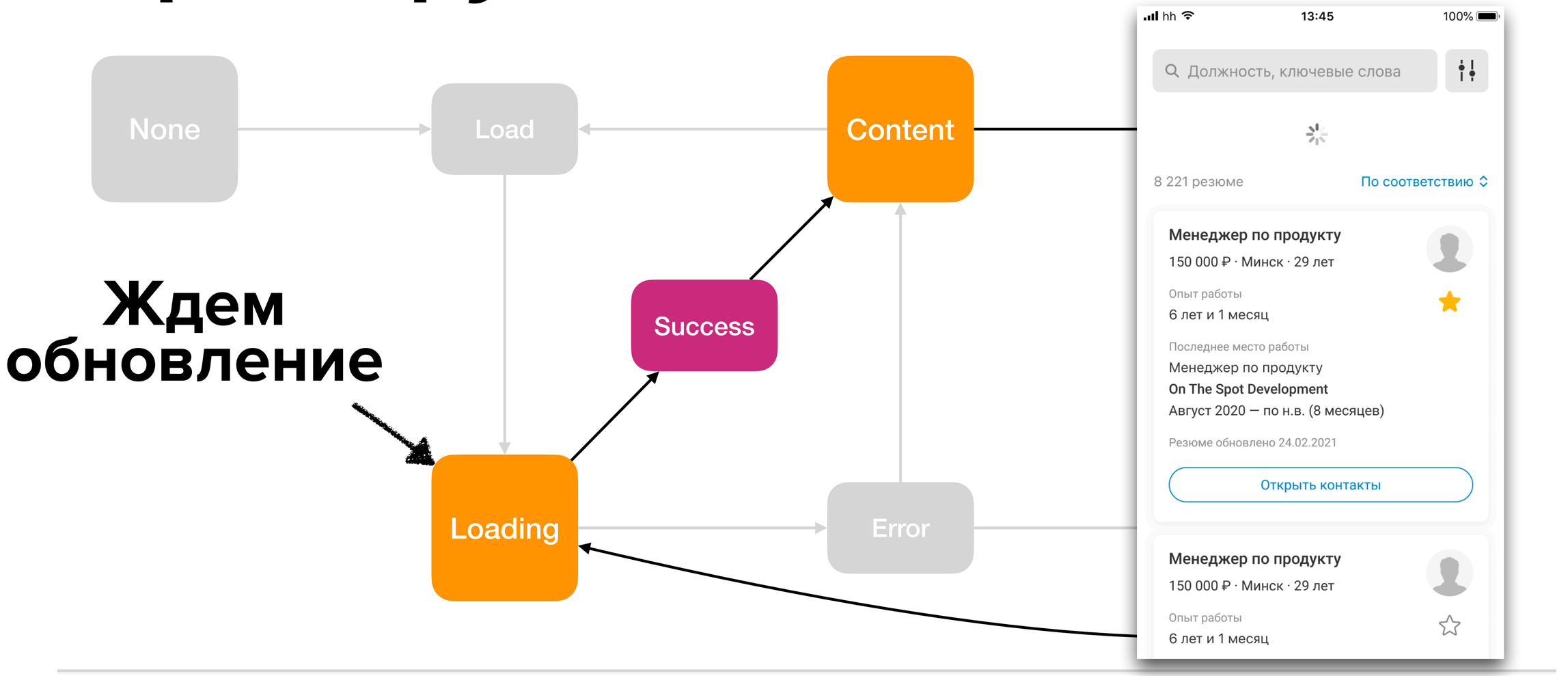
Перезагрузка данных



Перезагрузка данных

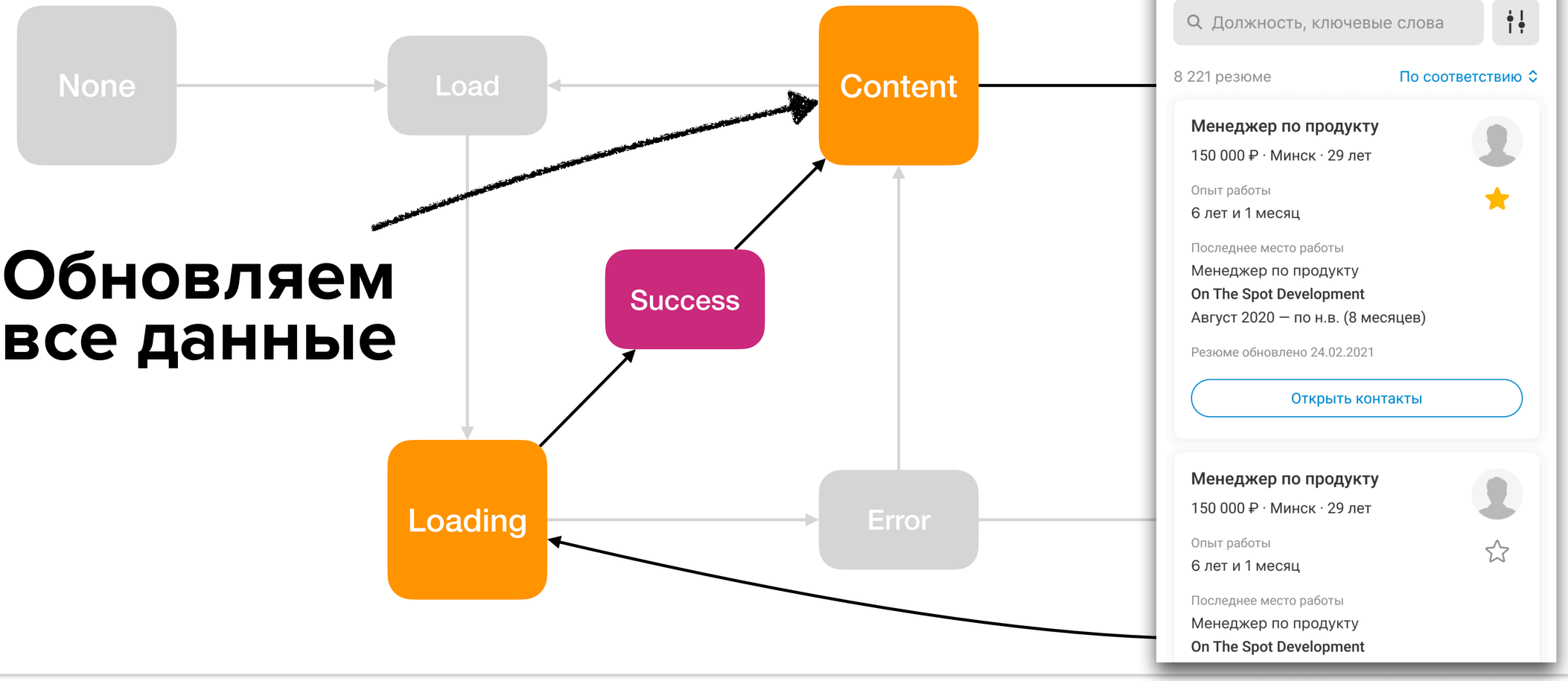


Перезагрузка данных



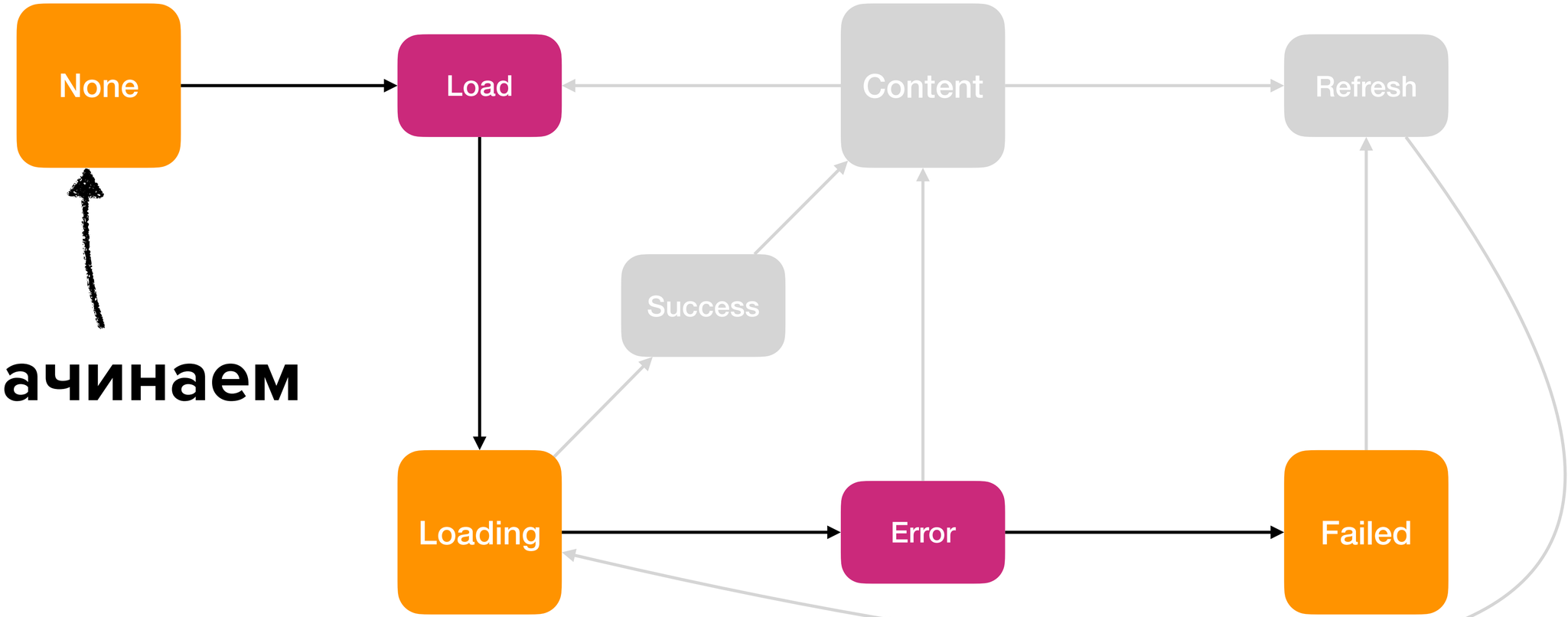
State Event

Перезагрузка данных



State Event

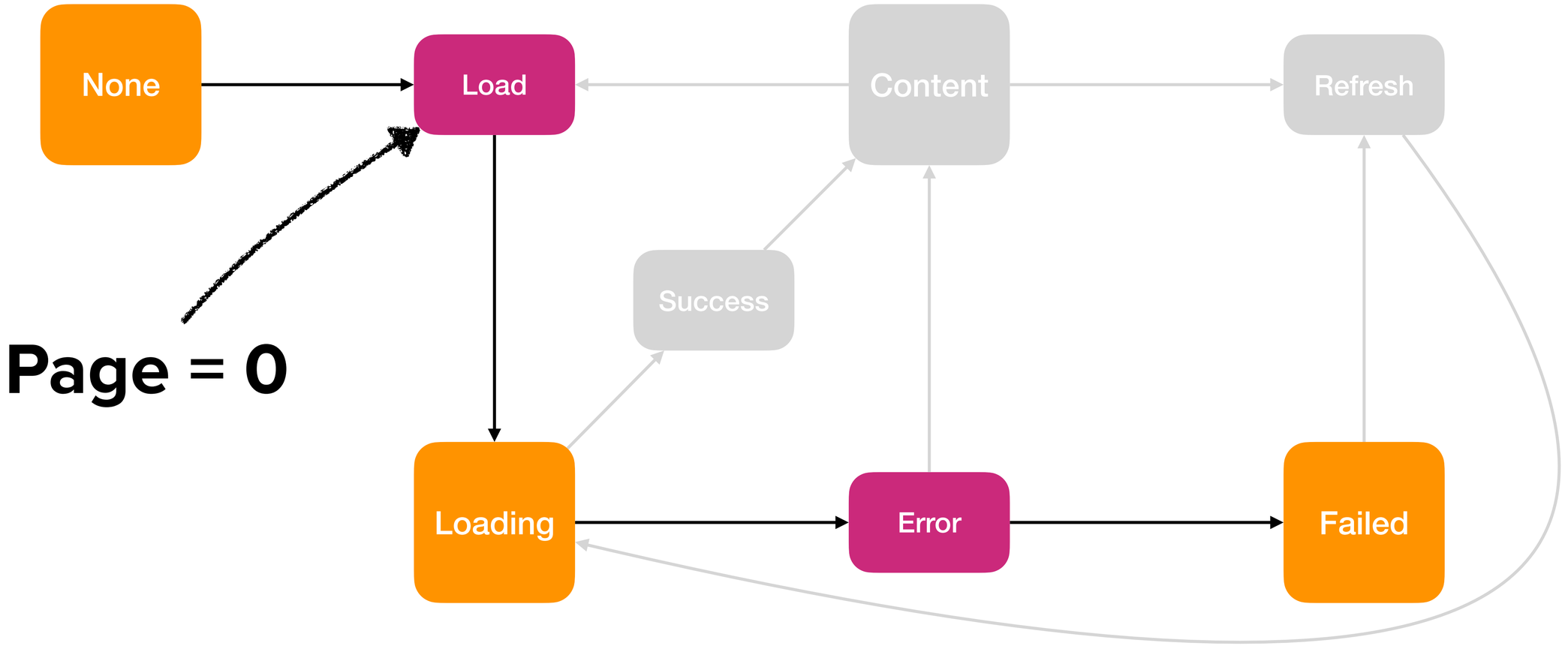
Ошибка



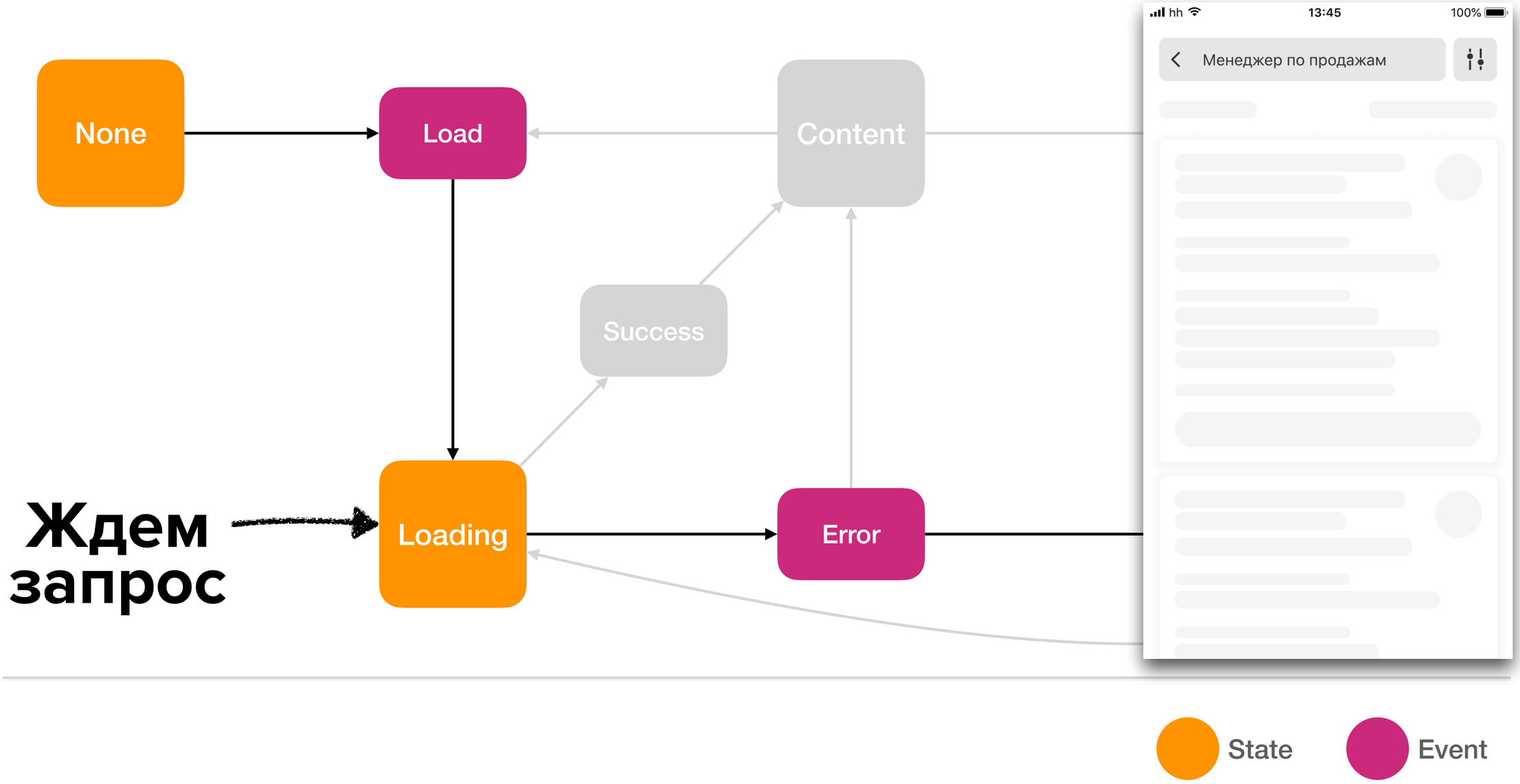
Начинаем

State Event

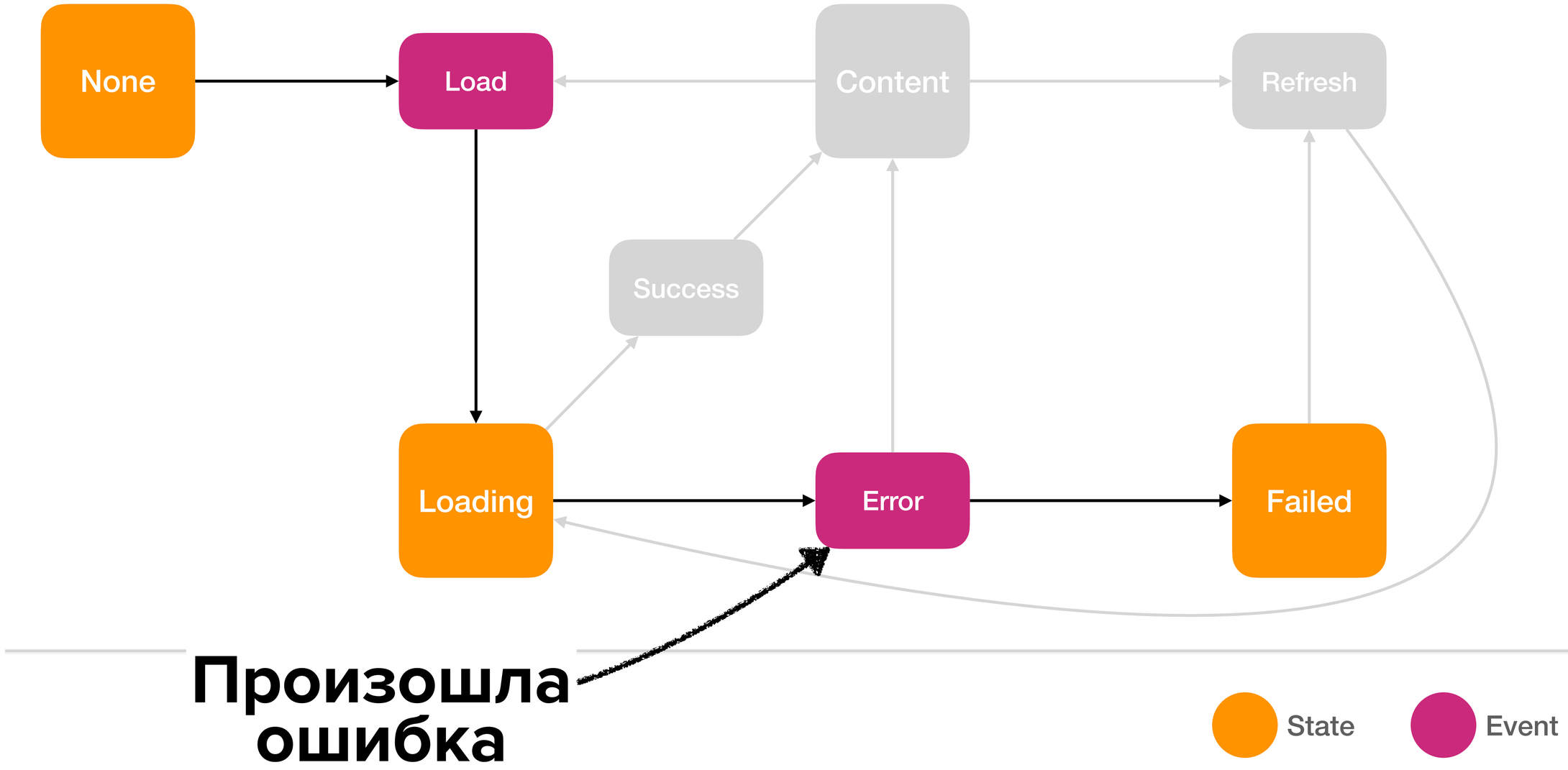
Ошибка


None
Load
Content
Refresh
Success
Page = 0
Loading
Error
Failed
State
Event

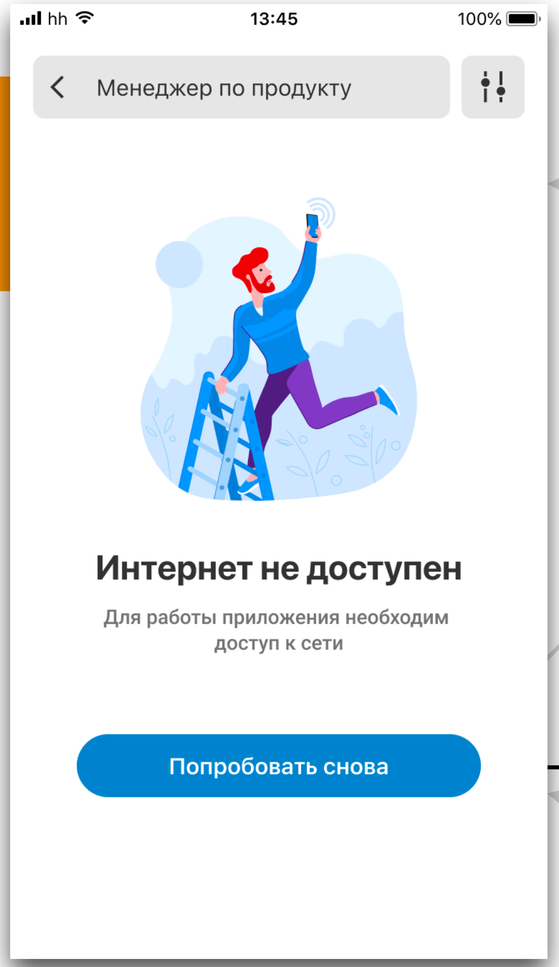
Ошибка


None
Load
Content
Success
Loading
Error
Ждем запрос
hh
13:45
100%
Менеджер по продажам
State
Event

Ошибка


None
Load
Content
Refresh
Success
Loading
Error
Failed
Произошла ошибка
State
Event

Ошибка



None

Content

Refresh

Success

Error

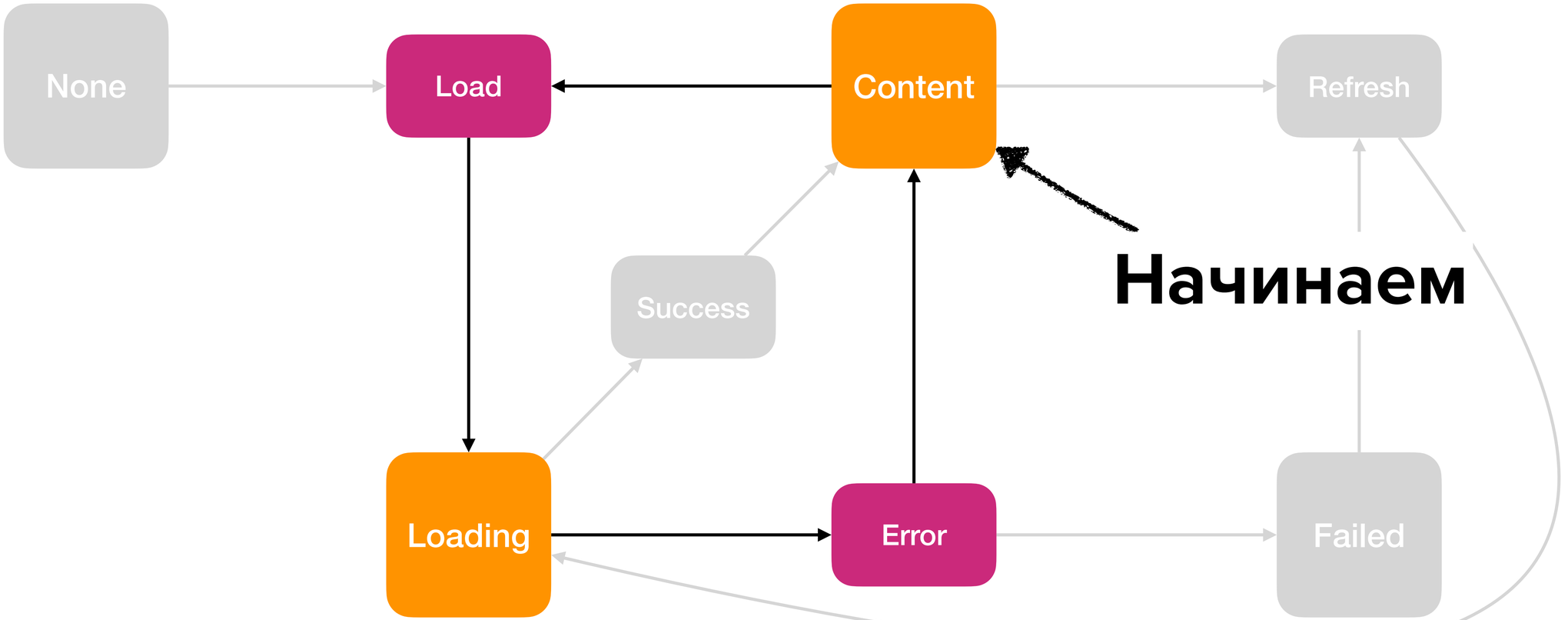
Failed

Покажем ее

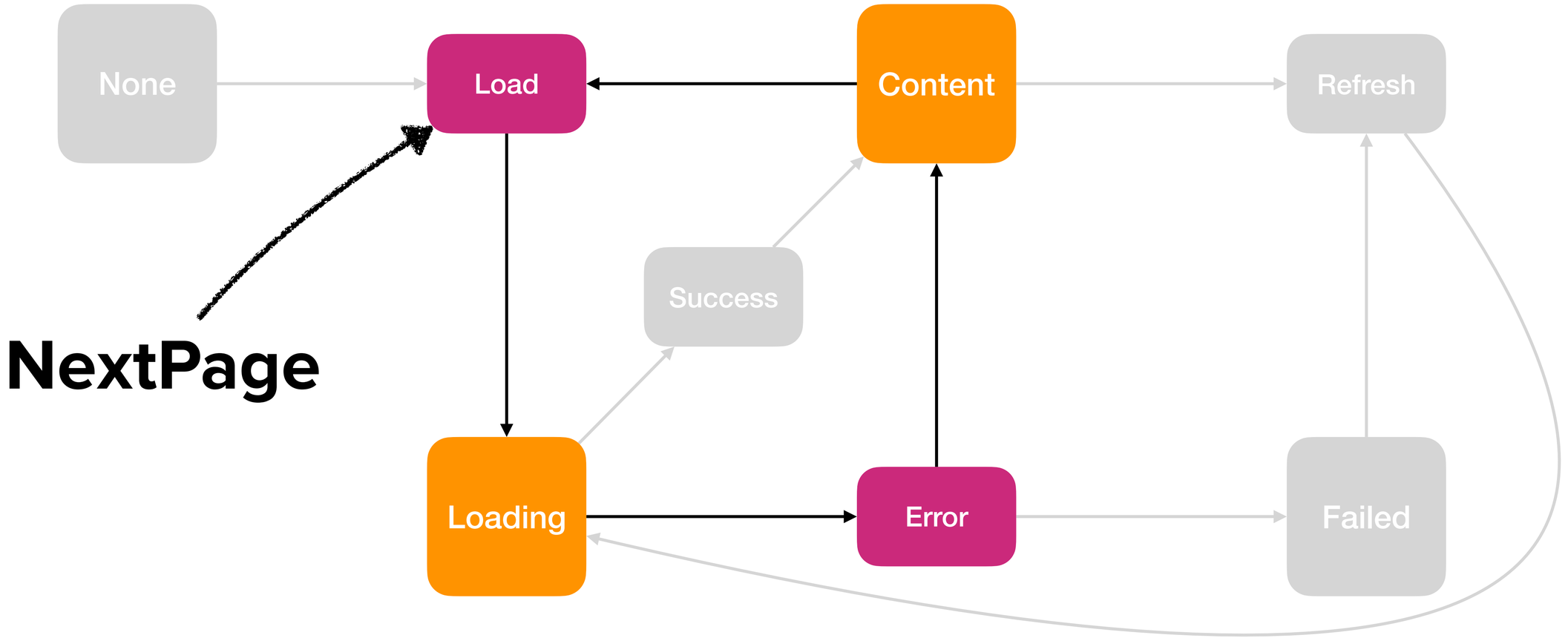
State

Event

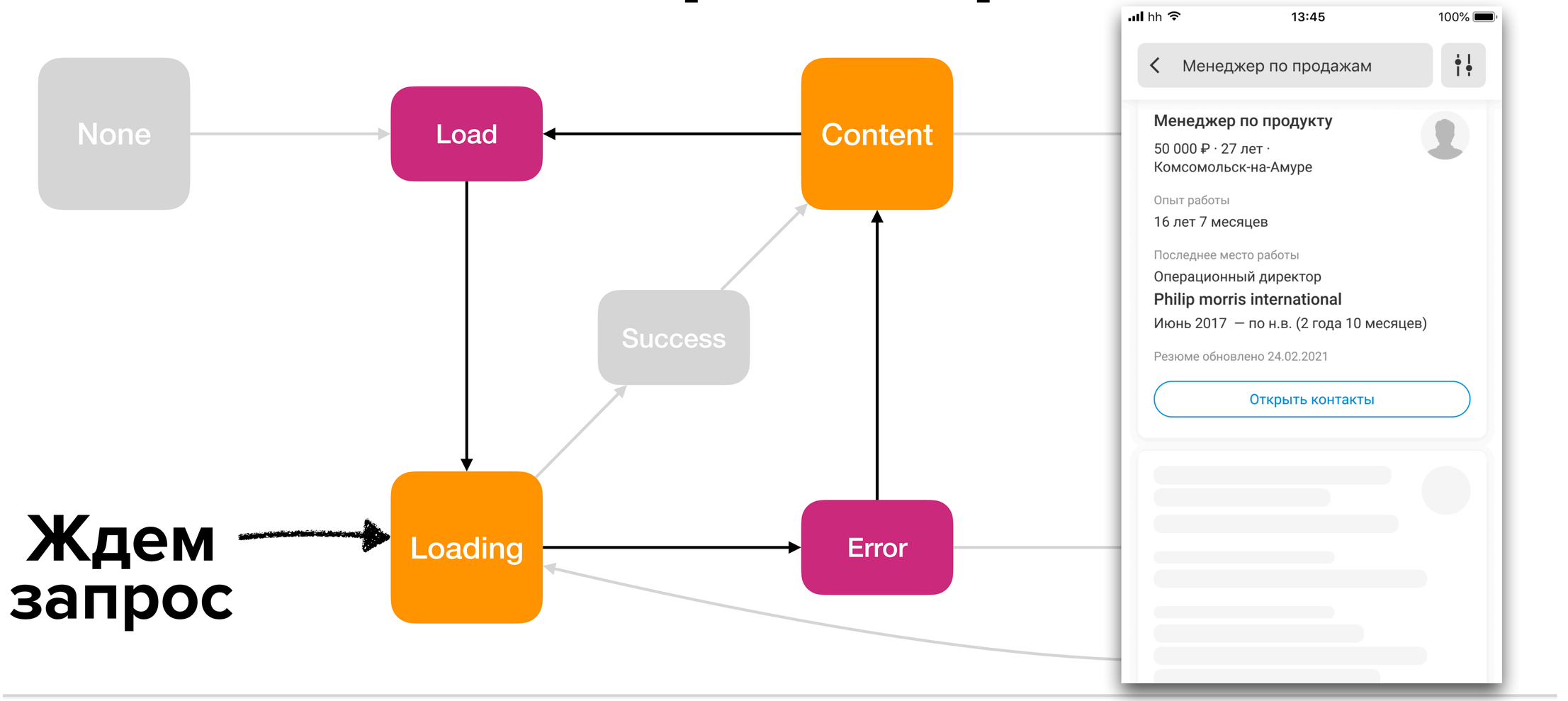
Ошибка на второй странице



Ошибка на второй странице

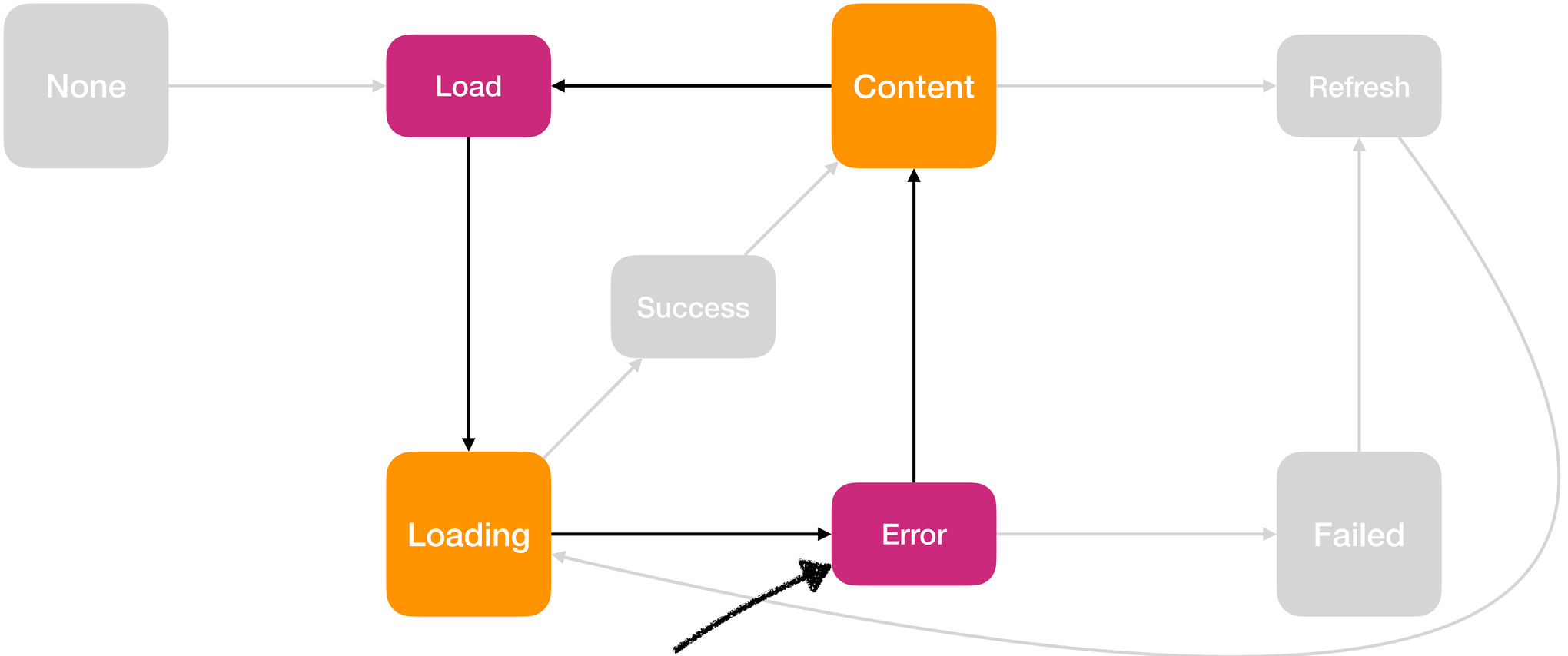


Ошибка на второй странице



State Event

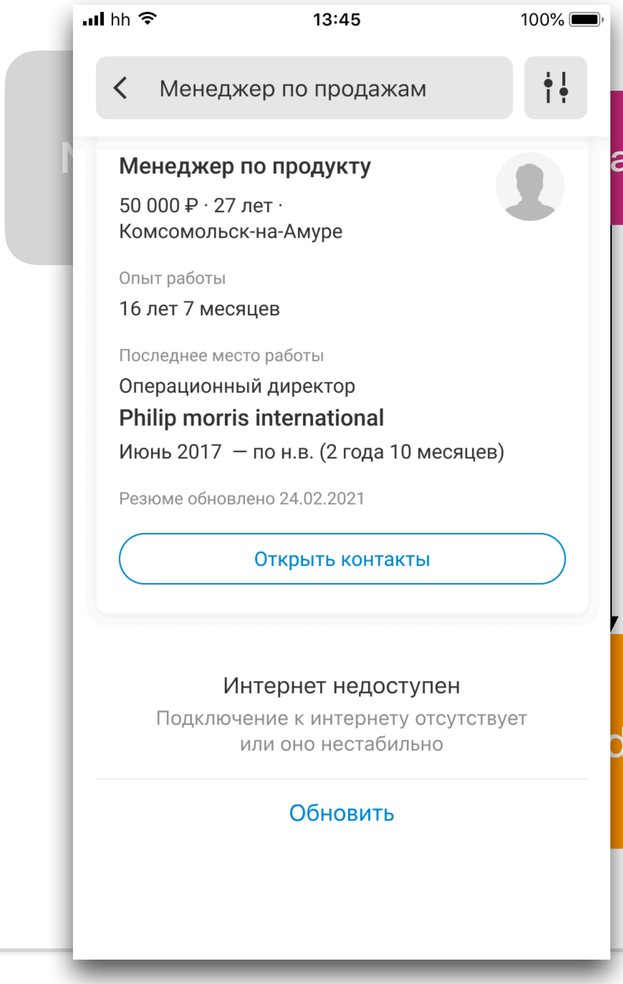
Ошибка на второй странице



Но тут ошибка пришла

State Event

Ошибка на второй странице



PaginationFeature — State

```
struct PaginationFeatureState {

    enum LoadingOption: Equatable {
        case refreshing
        case nextPageLoading
        case nextPageLoadingError(_ error: Error)
    }

    enum DataState: Equatable {
        case initial
        case loading
        case content(paginationItems: PaginationItems, loadingOption: LoadingOption?)
        case error(_ error: Error)
    }

    let data: DataState
}
```

Немного кода

```
struct PaginationFeatureState {

    enum LoadingOption: Equatable {
        case refreshing
        case nextPageLoading
        case nextPageLoadingError(_ error: Error)
    }

    enum DataState: Equatable {
        case initial
        case loading
        case content(paginationItems: PaginationItems, loadingOption: LoadingOption?)
        case error(_ error: Error)
    }

    let data: DataState
}
```

LCE-состояния экрана

```
struct PaginationFeatureState {

    enum LoadingOption: Equatable {
        case refreshing
        case nextPageLoading
        case nextPageLoadingError(_ error: Error)
    }

    enum DataState: Equatable {
        case initial
        case loading
        case content(paginationItems: PaginationItems, loadingOption: LoadingOption?)
        case error(_ error: Error)
    }

    let data: DataState
}
```

Флаги о загрузке для Content-a

```
struct PaginationFeatureState {

    enum LoadingOption: Equatable {
        case refreshing
        case nextPageLoading
        case nextPageLoadingError(_ error: Error)
    }

    enum DataState: Equatable {
        case initial
        case loading
        case content(paginationItems: PaginationItems, loadingOption: LoadingOption?)
        case error(_ error: Error)
    }

    let data: DataState
}
```

PaginationFeature — Wish

```
public enum PaginationWish {
    case load(isNextPage: Bool)
    case refresh
}

public enum PaginationEffect {
    case itemsDidLoad(paginationItems: PaginationItems)
    case itemsLoadingDidFail(error: Error, isPaginationError: Bool)
    case itemsLoadingDidStart(isNextPage: Bool)
    case itemsRefreshingDidStart
}
```

PaginationFeature — Effect

```
public enum PaginationWish {
    case load(isNextPage: Bool)
    case refresh
}

public enum PaginationEffect {
    case itemsLoadingDidStart(isNextPage: Bool)
    case itemsDidLoad(paginationItems: PaginationItems)
    case itemsLoadingDidFail(error: Error, isPaginationError: Bool)
    case itemsRefreshingDidStart
}
```

PaginationFeature — Actor

```
public func process(
    state: PaginationFeatureState,
    wish: PaginationWish
) -> AnyPublisher<PaginationEffect> {
    switch wish {
    case let .load(isNextPage):
        guard
            !state.isProcessing,
            state.canLoadNextPage || !isNextPage,
            state.loadingError == nil
        else {
            return .none
        }
        return fetch(
                for: state,
                page: isNextPage ? state.paginationItems?.nextPageIndex : 0
            )
            .prepend(.itemsLoadingDidStart(isNextPage: isNextPage))
            .eraseToAnyPublisher()
```

.. .. ..

PaginationFeature — Actor

```
public func process(
    state: PaginationFeatureState,
    wish: PaginationWish
) -> AnyPublisher<PaginationEffect> {
    switch wish {
    case let .load(isNextPage):
        guard
            !state.isProcessing,
            state.canLoadNextPage || !isNextPage,
            state.loadingError == nil
        else {
            return .none
        }
        return fetch(
                for: state,
                page: isNextPage ? state.paginationItems?.nextPageIndex : 0
            )
            .prepend(.itemsLoadingDidStart(isNextPage: isNextPage))
            .eraseToAnyPublisher()
```

.. .. ..

PaginationFeature — Actor

```
public func process(
    state: PaginationFeatureState,
    wish: PaginationWish
) -> AnyPublisher<PaginationEffect> {
    switch wish {
    case let .load(isNextPage):
        guard
            !state.isProcessing,
            state.canLoadNextPage || !isNextPage,
            state.loadingError == nil
        else {
            return .none
        }
        return fetch(
                for: state,
                page: isNextPage ? state.paginationItems?.nextPageIndex : 0
            )
            .prepend(.itemsLoadingDidStart(isNextPage: isNextPage))
            .eraseToAnyPublisher()
```

.. .. ..

PaginationFeature — Actor

```
public func process(
    state: PaginationFeatureState,
    wish: PaginationWish
) -> AnyPublisher<PaginationEffect> {
    switch wish {
    case let .load(isNextPage):
        guard
            !state.isProcessing,
            state.canLoadNextPage || !isNextPage,
            state.loadingError == nil
        else {
            return .none
        }
        return fetch(
                for: state,
                page: isNextPage ? state.paginationItems?.nextPageIndex : 0
            )
            .prepend(.itemsLoadingDidStart(isNextPage: isNextPage))
            .eraseToAnyPublisher()
```

.. .. ..

PaginationFeature — Actor

```
public func process(
    state: PaginationFeatureState,
    wish: PaginationWish
) -> AnyPublisher<PaginationEffect> {
    switch wish {
    case let .load(isNextPage):
        guard
            !state.isProcessing,
            state.canLoadNextPage || !isNextPage,
            state.loadingError == nil
        else {
            return .none
        }
        return fetch(
                for: state,
                page: isNextPage ? state.paginationItems?.nextPageIndex : 0
            )
            .prepend(.itemsLoadingDidStart(isNextPage: isNextPage))
            .eraseToAnyPublisher()
```

.. .. ..

PaginationFeature — Reducer

```
public func process(
    state: PaginationFeatureState,
    effect: PaginationEffect
) -> PaginationFeatureState {
    switch effect {
    case let .itemsDidLoad(paginationItems):
        return state.changing(
            \.state,
            to: .items(paginationItems: paginationItems, loadingStatus: .none)
        )
    case let .itemsLoadingDidFail(error, isPaginationError):
        let errorModel = error.mapToHHSDKErrorModel()
        return state.changing(
            \.state,
            to: .failed(
                error: isPaginationError
                    ? .paginationError(paginationItems: state.paginationItems, error: errorModel)
                    : .loadingError(error: errorModel)
            )
        )
.. .. ..
```

PaginationFeature — Reducer

```
public func process(
    state: PaginationFeatureState,
    effect: PaginationEffect
) -> PaginationFeatureState {
    switch effect {
    case let .itemsDidLoad(paginationItems):
        return state.changing(
            \.state,
            to: .items(paginationItems: paginationItems, loadingStatus: .none)
        )
    case let .itemsLoadingDidFail(error, isPaginationError):
        let errorModel = error.mapToHHSDKErrorModel()
        return state.changing(
            \.state,
            to: .failed(
                error: isPaginationError
                    ? .paginationError(paginationItems: state.paginationItems, error: errorModel)
                    : .loadingError(error: errorModel)
            )
        )
.. .. ..
```


Подведём
ИТОГИ

Что мы получили

\+ Общий подход в модулях (и между платформами)

\+ Легко расширять и менять логику

\+ Генерация кода

\- Иногда оверхед

\- Сложный онбординг новых разработчиков

Спросите меня о чём-нибудь ;)

1. Что такое стейт-машина
2. Анализ реализаций стейт-машин
3. Схема работы MVI
4. Пример реализации с кодом

Ссылка на слайды