Flux-архитектура в iOS

Денис Кириллов kirillov@mamba.ru

План

- История
- Концепция и реализация Flux
- Реализация в iOS
- Проблемы и решения
- Итоги

План

- История
- Концепция и реализация Flux
- Реализация в iOS
- Проблемы и решения
- Итоги

План

- История
- Концепция и реализация Flux
- Реализация в iOS
- Проблемы и решения
- Итоги

План

- История
- Концепция и реализация Flux
- Реализация в iOS
- Проблемы и решения
- Итоги

План

- История
- Концепция и реализация Flux
- Реализация в iOS
- Проблемы и решения
- Итоги

История

В поисках серебряной пули

MVC | Reactive | VIPER | MVVM | MVVM+C

В поисках серебряной пули

MVC | Reactive | VIPER | MVVM | MVVM+C

В поисках серебряной пули

MVC | Reactive | **VIPER** | MVVM | MVVM+C

В поисках серебряной пули

MVC | Reactive | VIPER | **MVVM** | MVVM+C

В поисках серебряной пули

MVC | Reactive | VIPER | MVVM | MVVM+C

...что дальше?

Привет frontend!

Flux + React

Flux - концепция архитектуры

React - framework на основе Flux для frontend

Почему FB придумали Flux

- Требовалось повысить качество и сократить время
- Решение на основе MVC плохо масштабировалось
- Разработчики плохо адаптировались в чужом коде
- Возникали проблемы каскадных обновлений

Источник: F8 2014 - Hacker Way: Rethinking Web App Development at Facebook

Почему FB придумали Flux

- Требовалось повысить качество и сократить время
- Решение на основе MVC плохо масштабировалось
- Разработчики плохо адаптировались в чужом коде
- Возникали проблемы каскадных обновлений

Источник: F8 2014 - Hacker Way: Rethinking Web App Development at Facebook

Почему FB придумали Flux

- Требовалось повысить качество и сократить время
- Решение на основе MVC плохо масштабировалось
- Разработчики плохо адаптировались в чужом коде
- Возникали проблемы каскадных обновлений

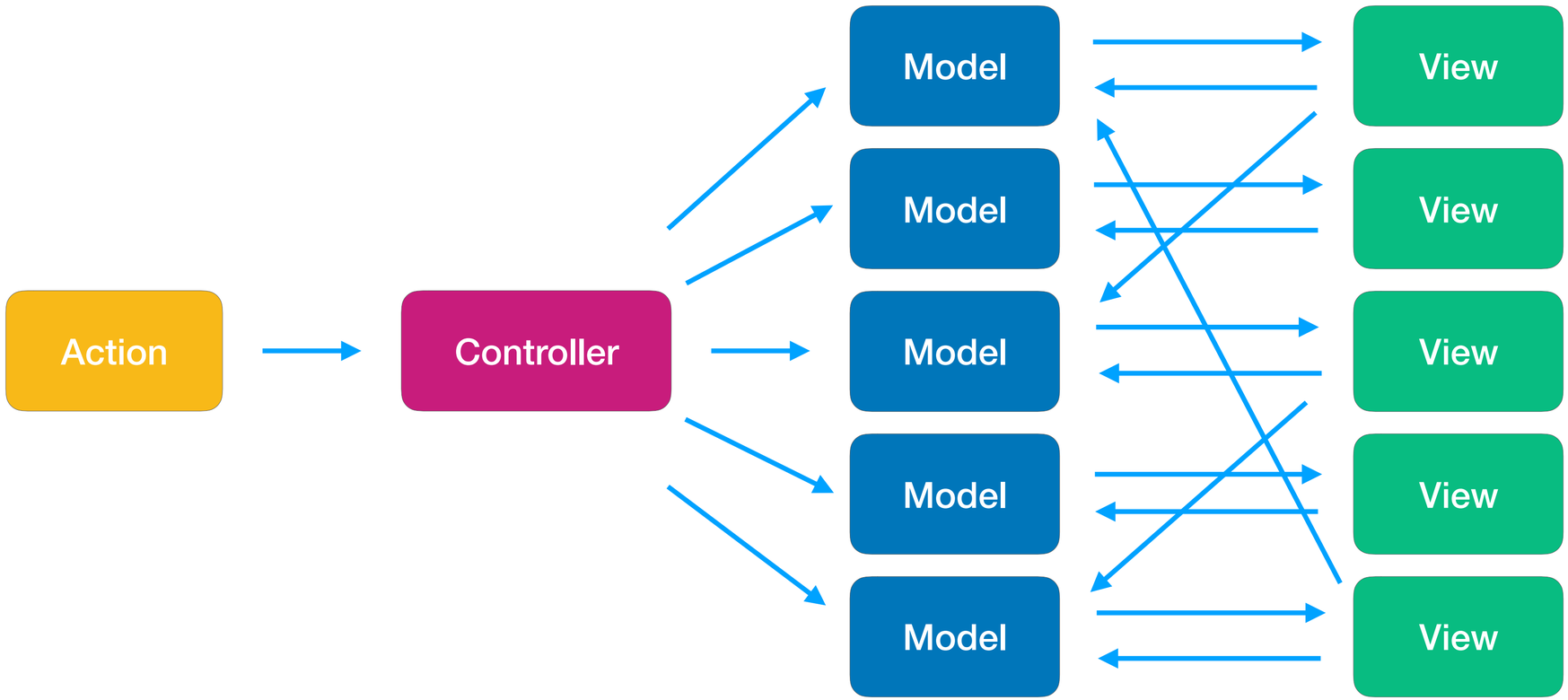
Источник: F8 2014 - Hacker Way: Rethinking Web App Development at Facebook

Почему FB придумали Flux

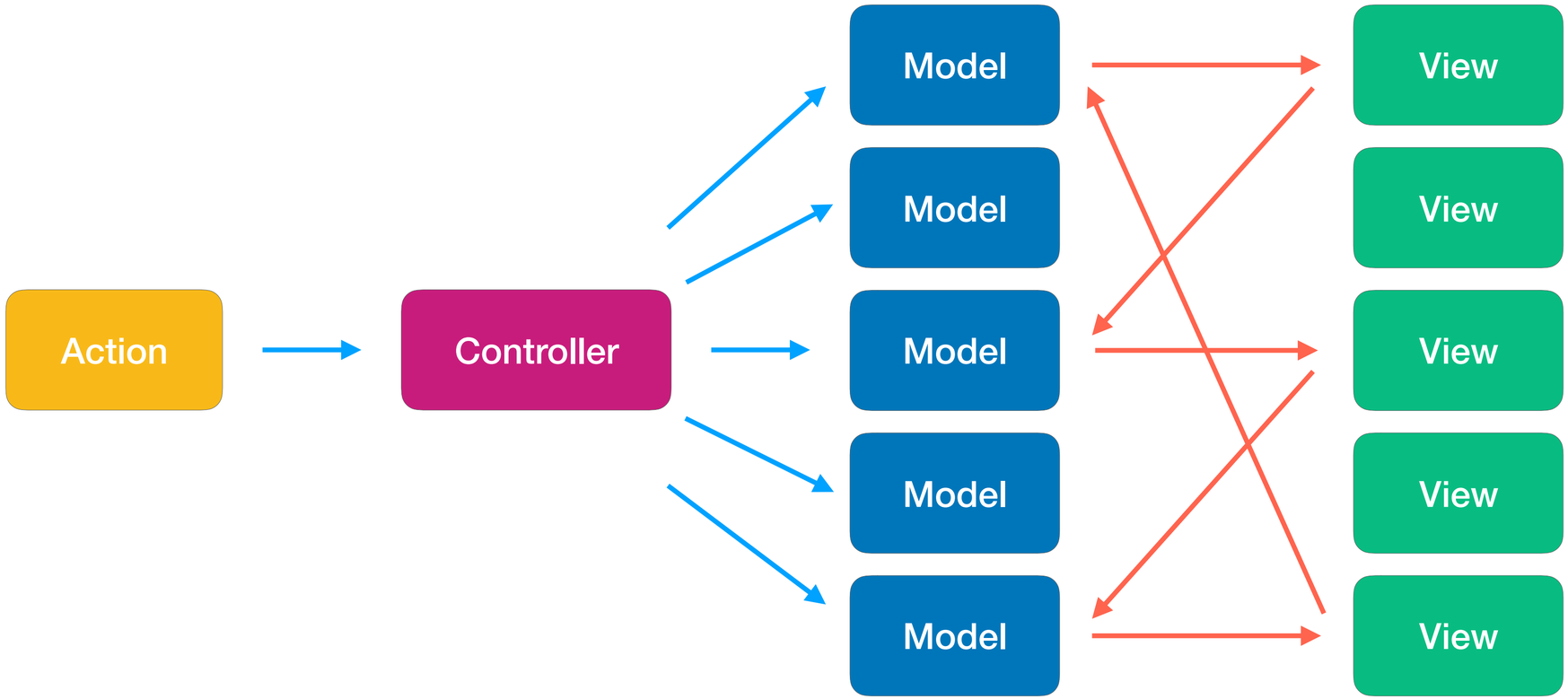
- Требовалось повысить качество и сократить время
- Решение на основе MVC плохо масштабировалось
- Разработчики плохо адаптировались в чужом коде
- Возникали проблемы каскадных обновлений

Источник: F8 2014 - Hacker Way: Rethinking Web App Development at Facebook

Каскадные обновления



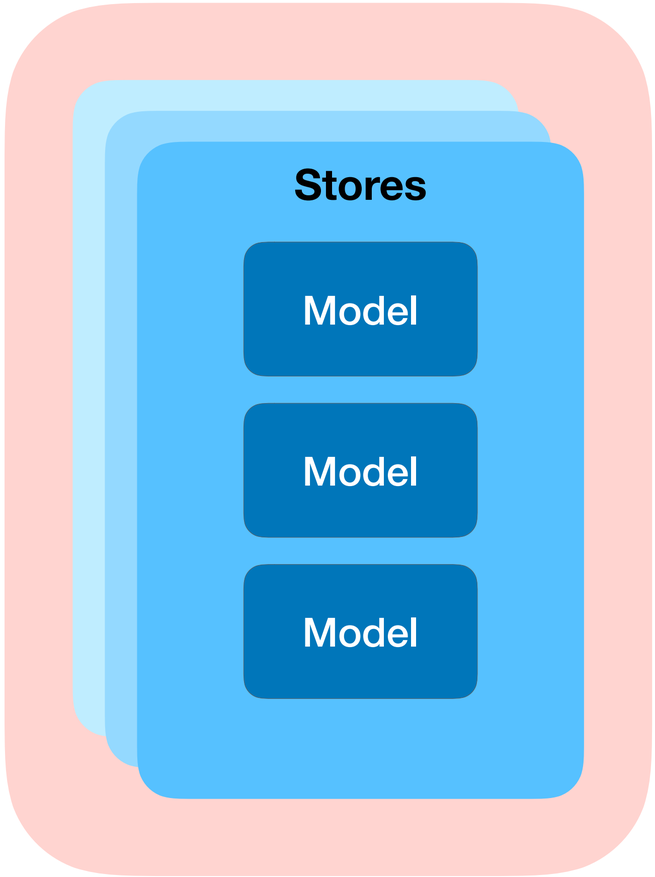
Каскадные обновления



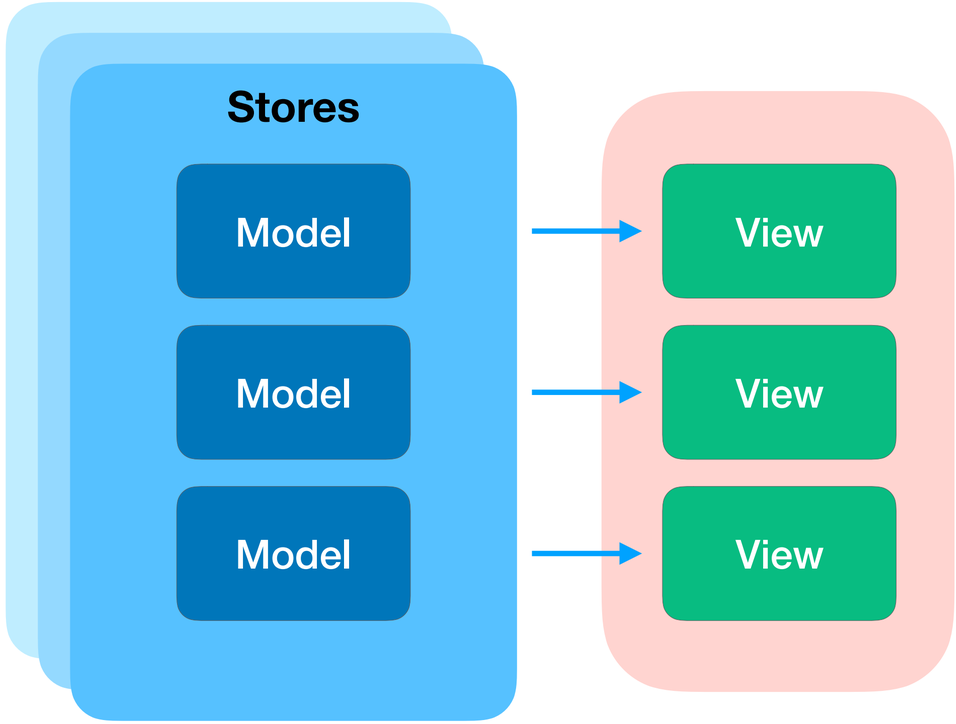
Концепция Flux

Flux архитектура реализует однонаправленный поток данных между компонентами.

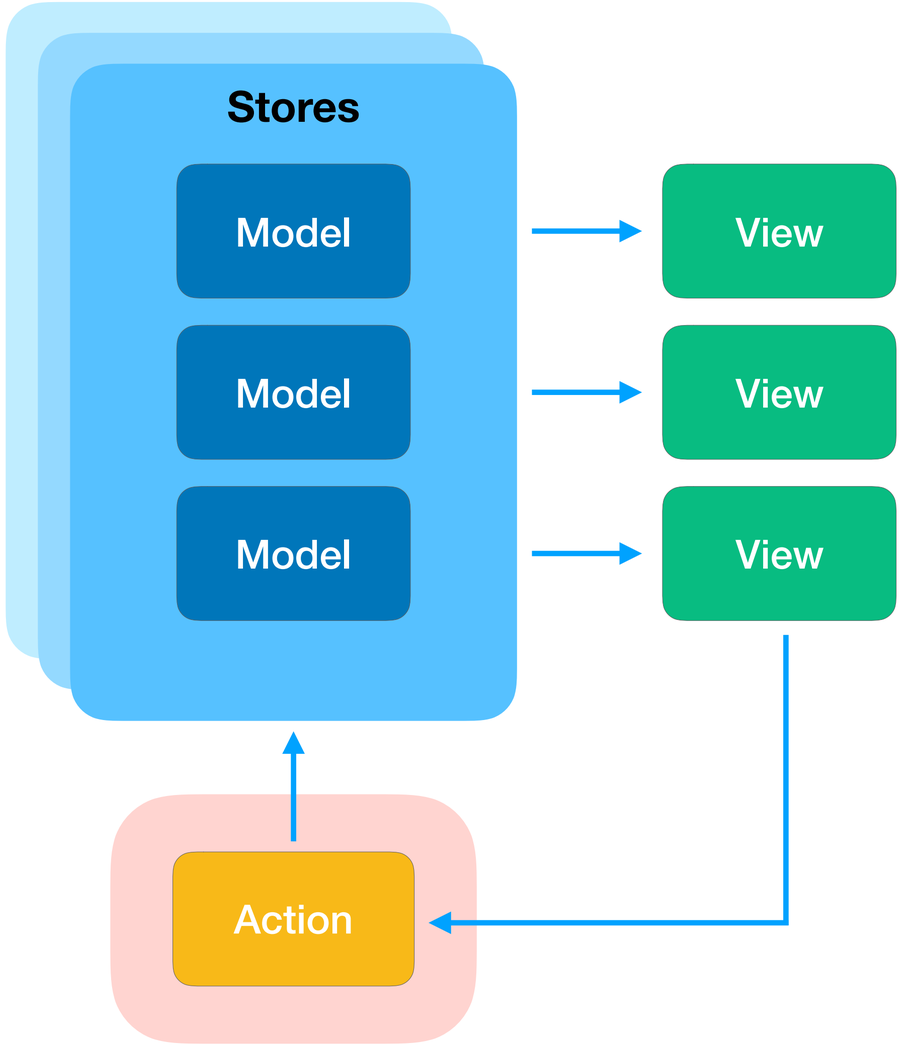
Flux архитектура



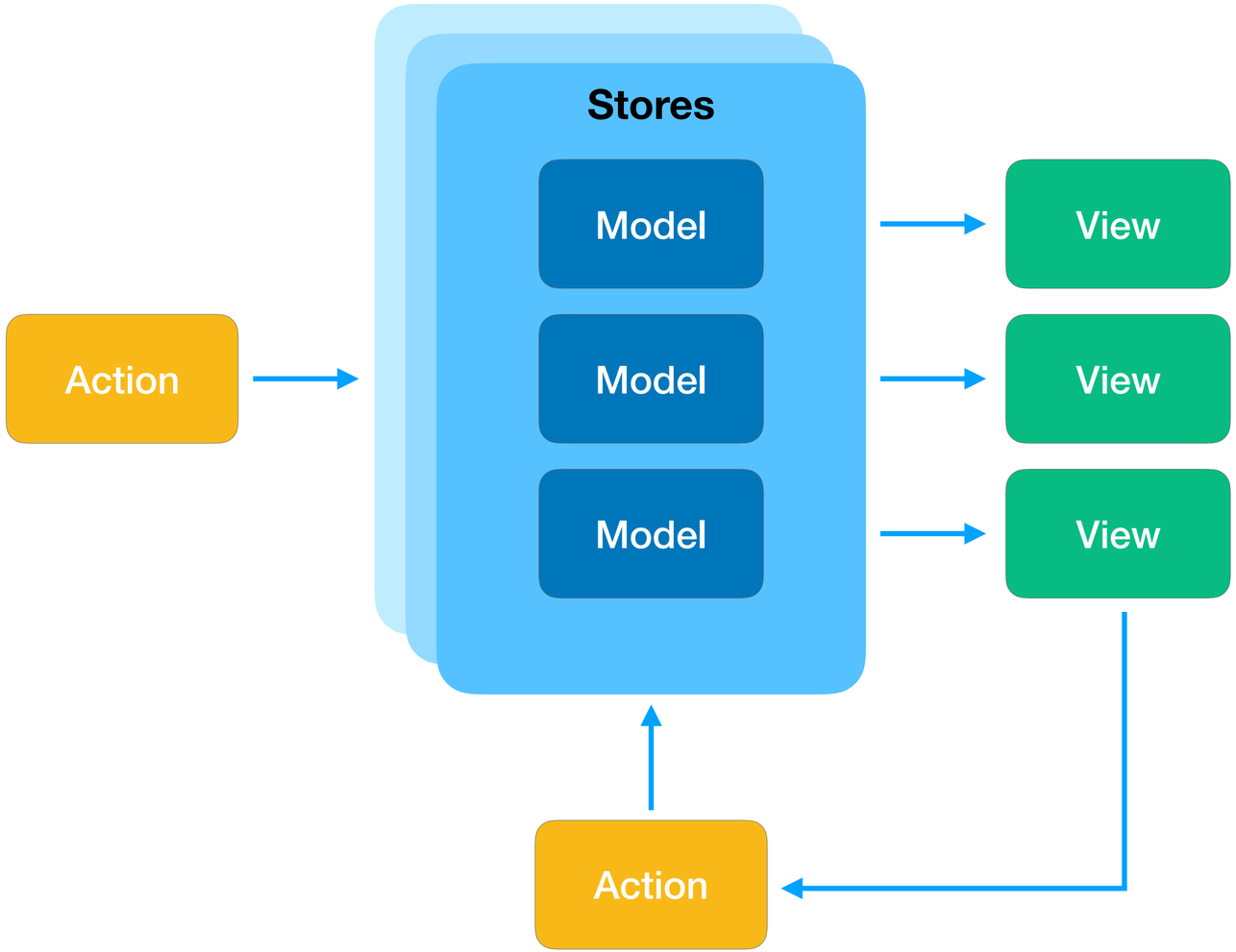
Flux архитектура



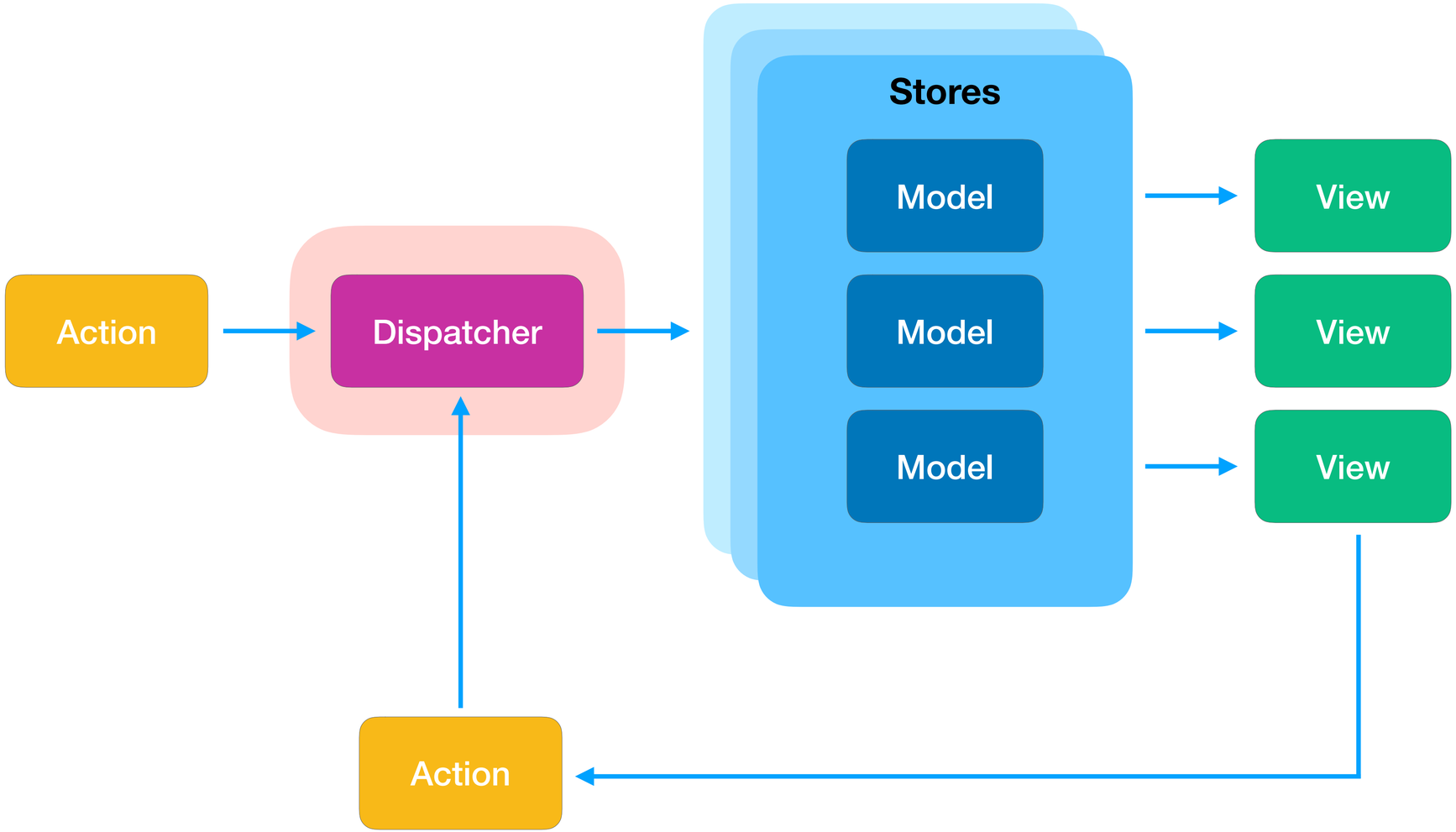
Flux архитектура



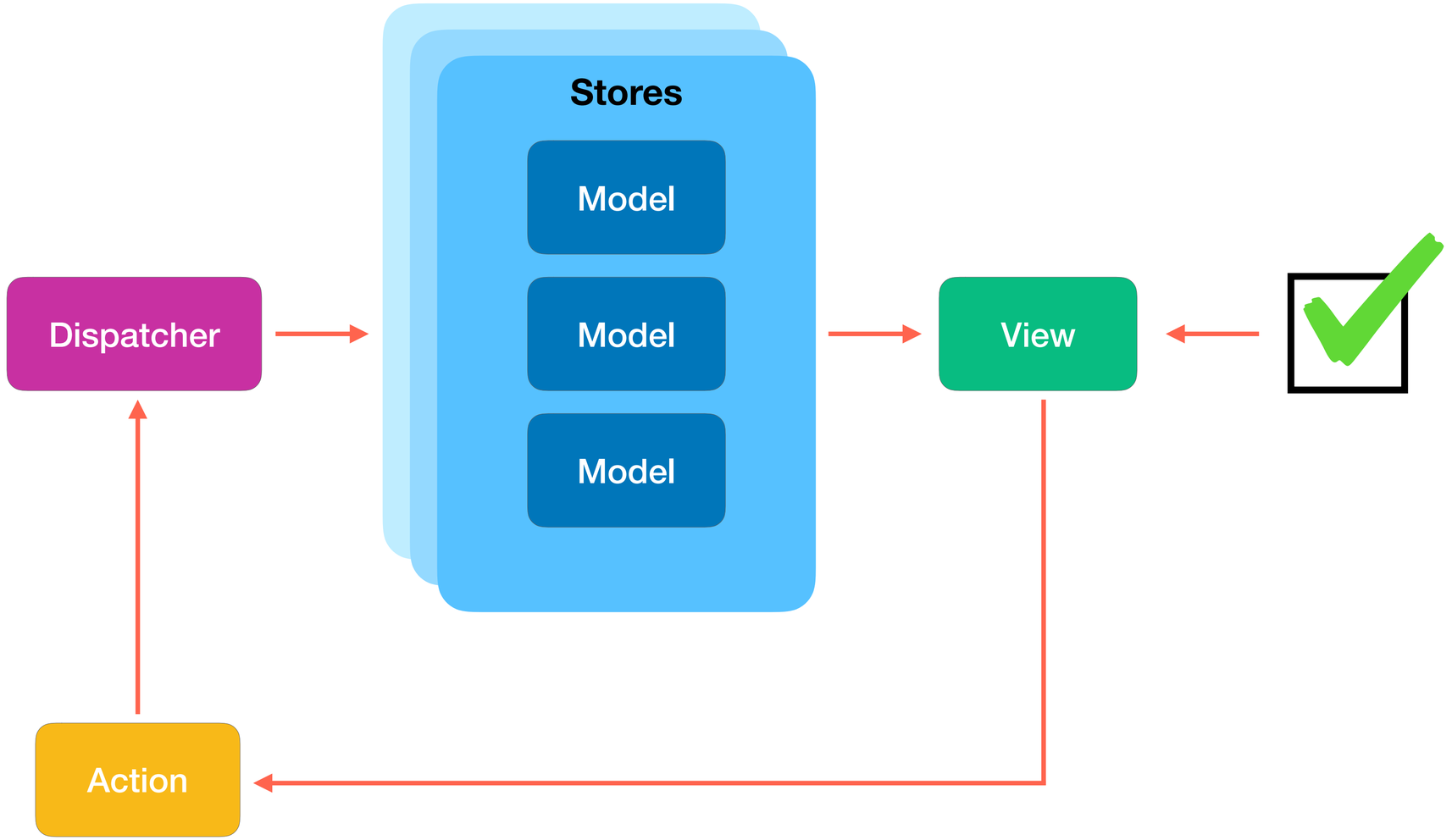
Flux архитектура



Flux архитектура



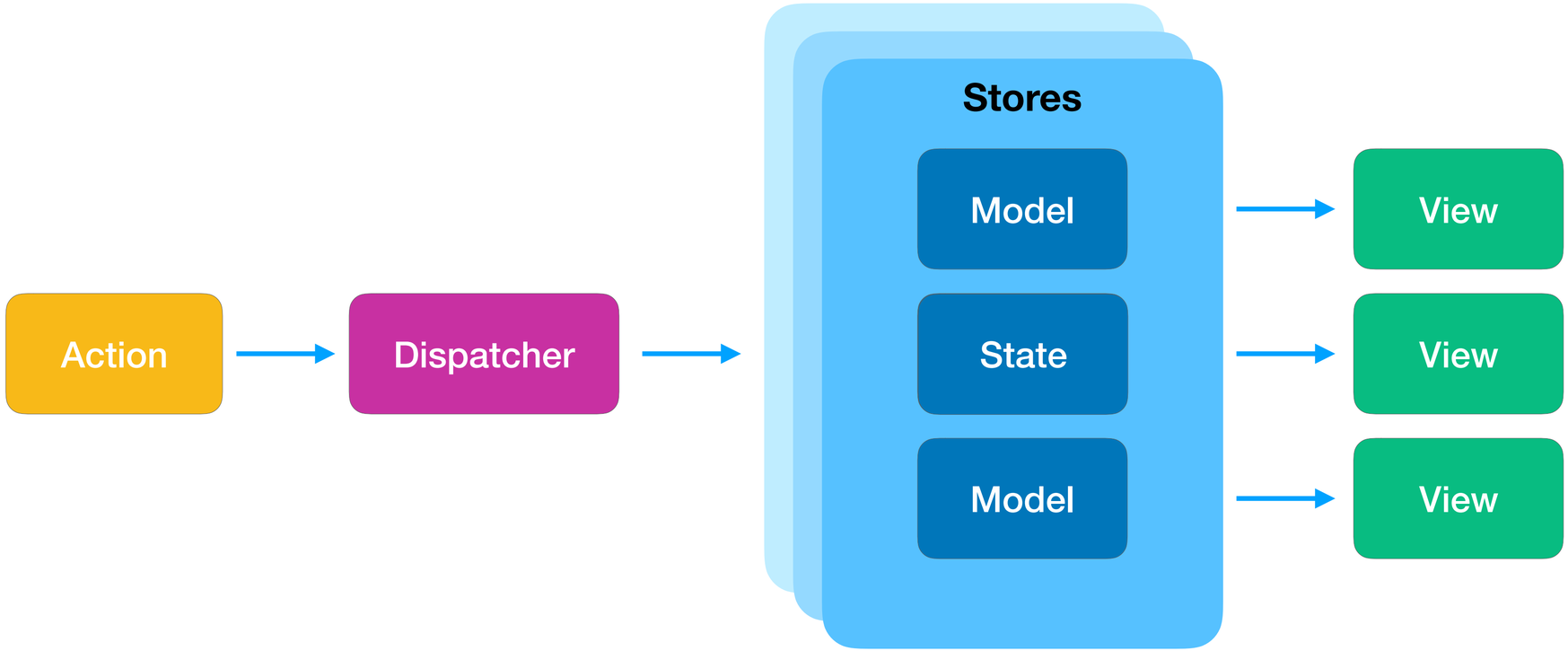
Flux архитектура



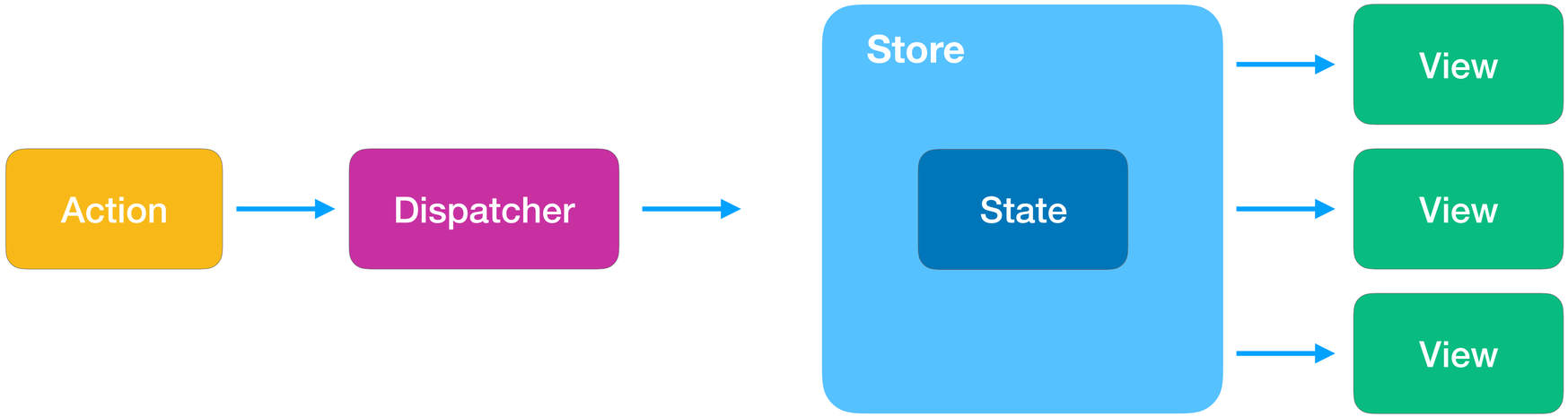
Реализация Flux

- Redux прост в понимании
- Самый популяный framework на основе Flux
- 45 000 звезд и 600 контрибьютеров на github
- Более 1000 расширений и море документации

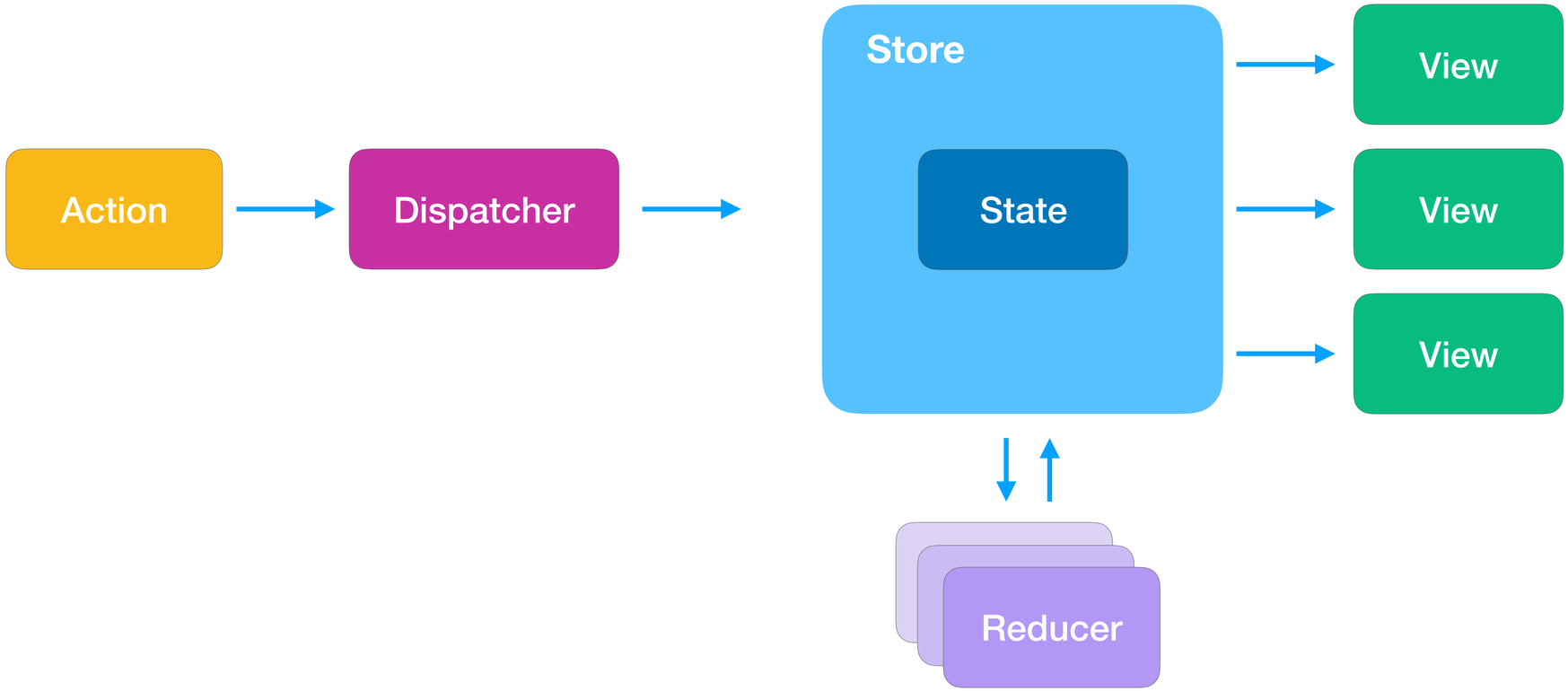
Flux в Redux



Flux в Redux

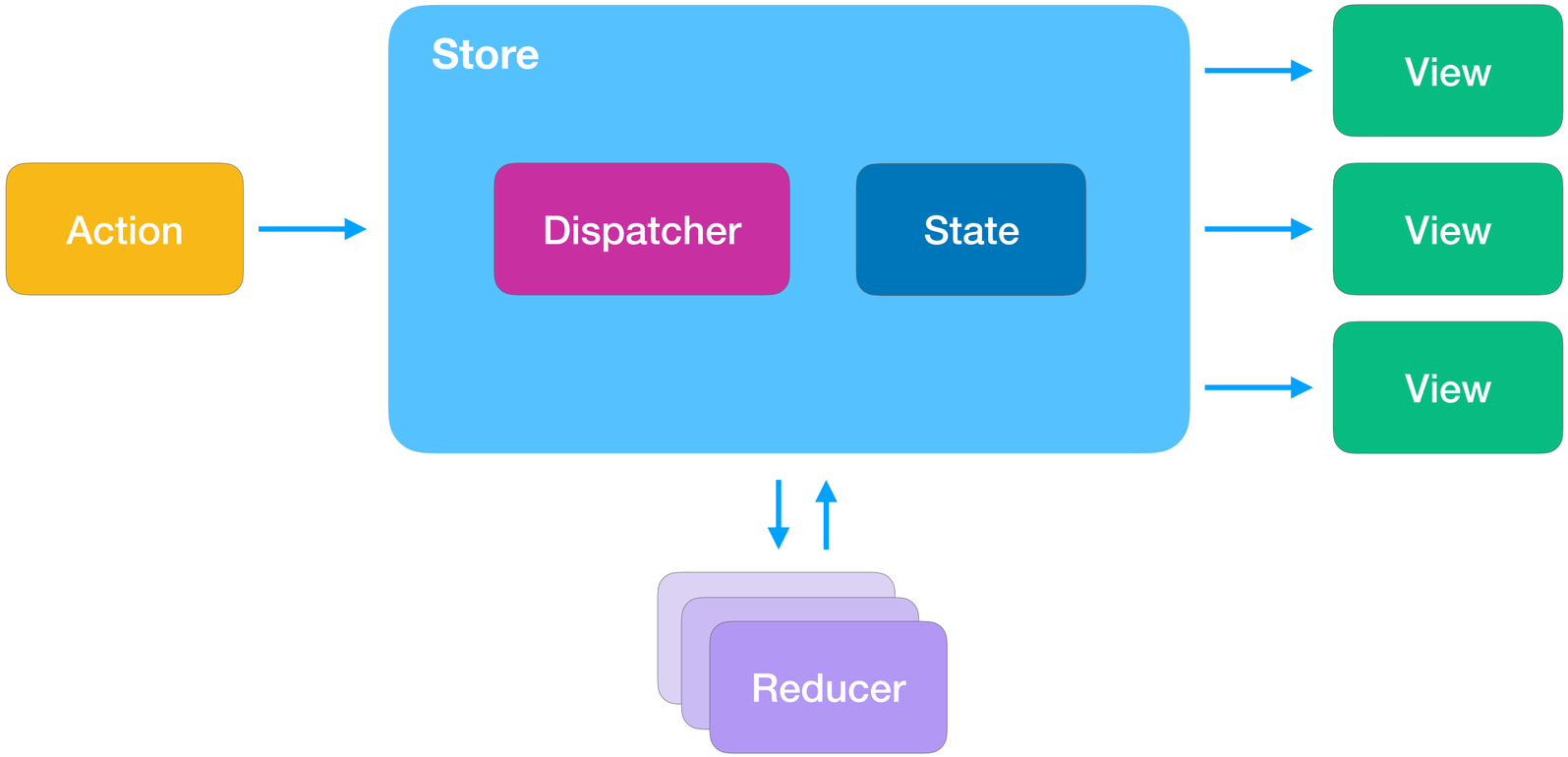


Flux в Redux



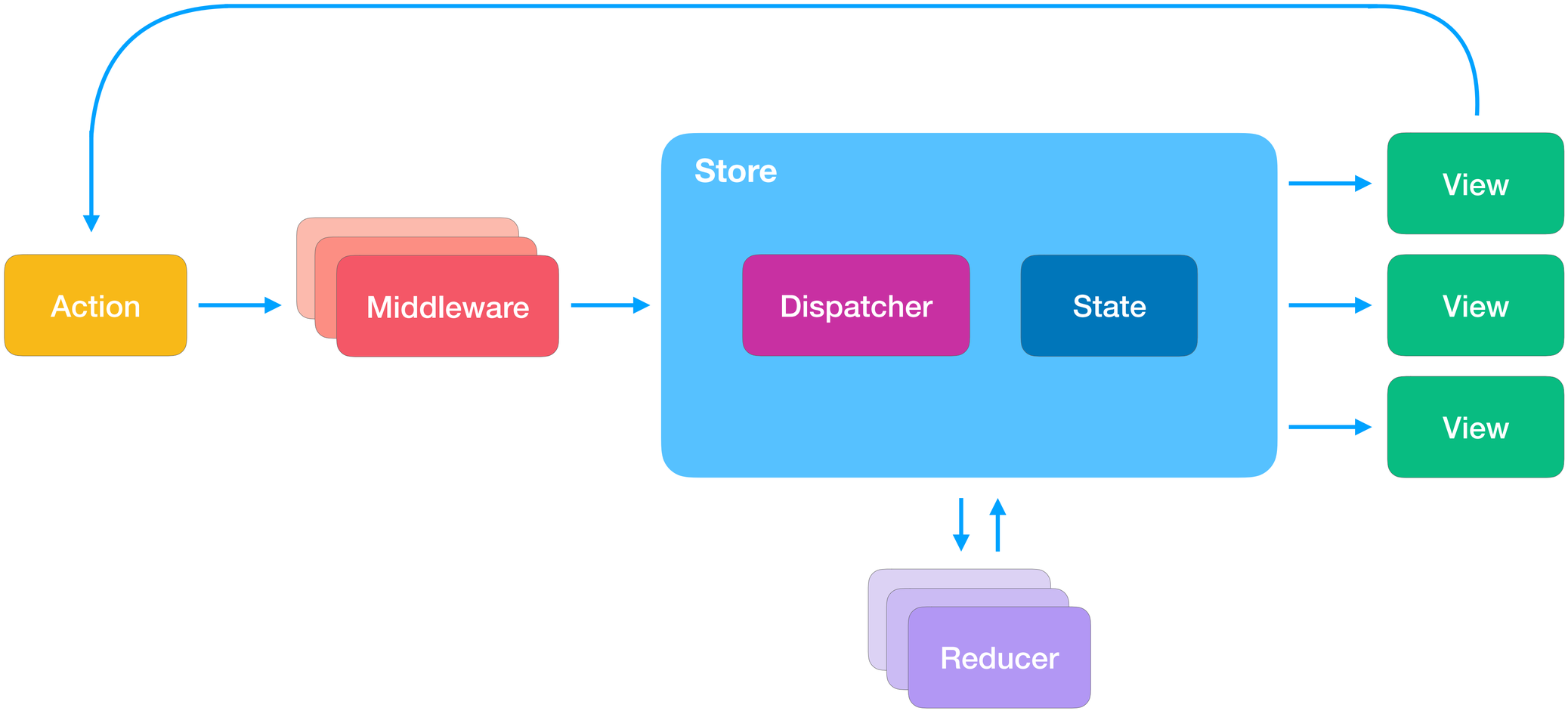
reducer(Action, State?) -> State

Flux в Redux



reducer(Action, State?) -> State

Flux в Redux



reducer(Action, State?) -> State

Что нам это даст?

- Более предсказуемый код
- Контроль на состоянием
- Удобно тестировать

Что нам это даст?

- Более предсказуемый код
- Контроль на состоянием
- Удобно тестировать

Что нам это даст?

- Более предсказуемый код
- Контроль на состоянием
- Удобно тестировать

Когда использовать?

- Много взаимосвязанных компонентов
- Нужно восстанавливать State
- Требуется отмена действий

Когда использовать?

- Много взаимосвязанных компонентов
- Нужно восстанавливать State
- Требуется отмена действий

Когда использовать?

- Много взаимосвязанных компонентов
- Нужно восстанавливать State
- Требуется отмена действий

Flux в iOS

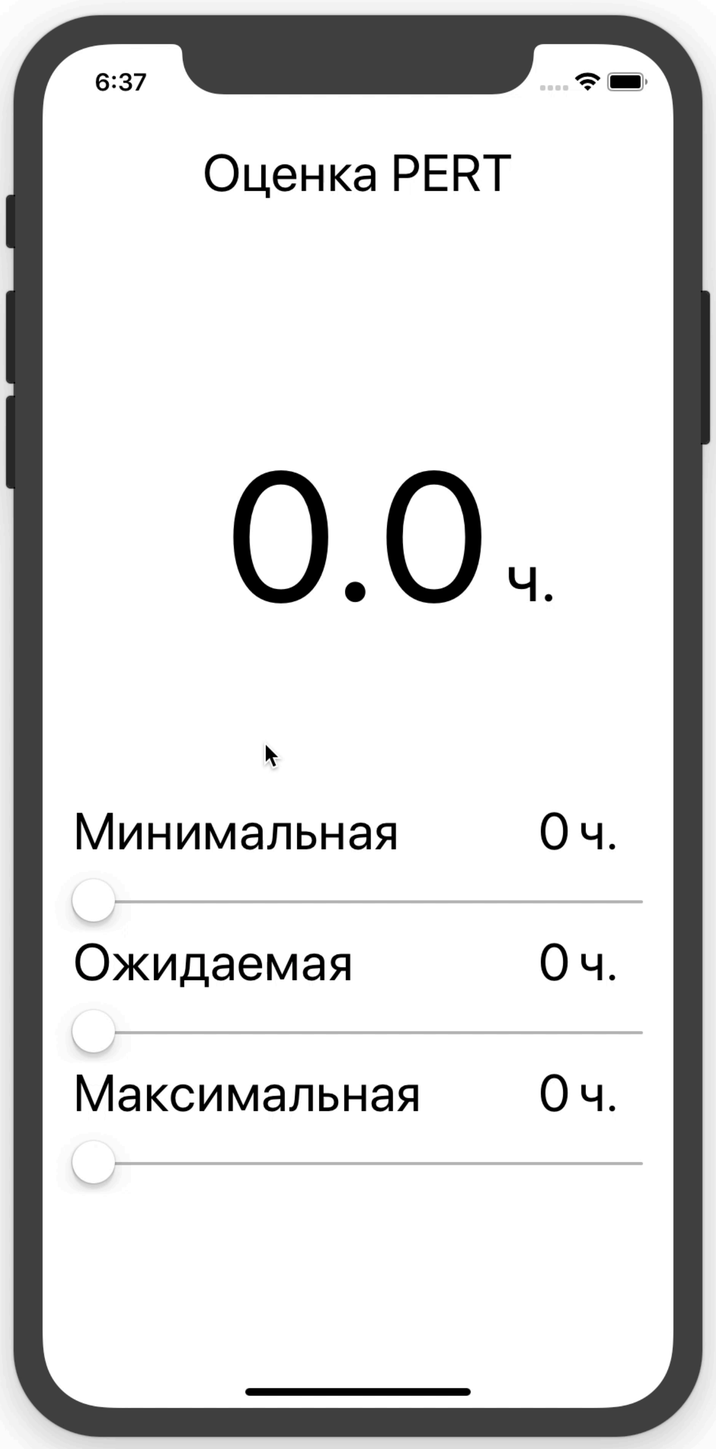
ReSwift

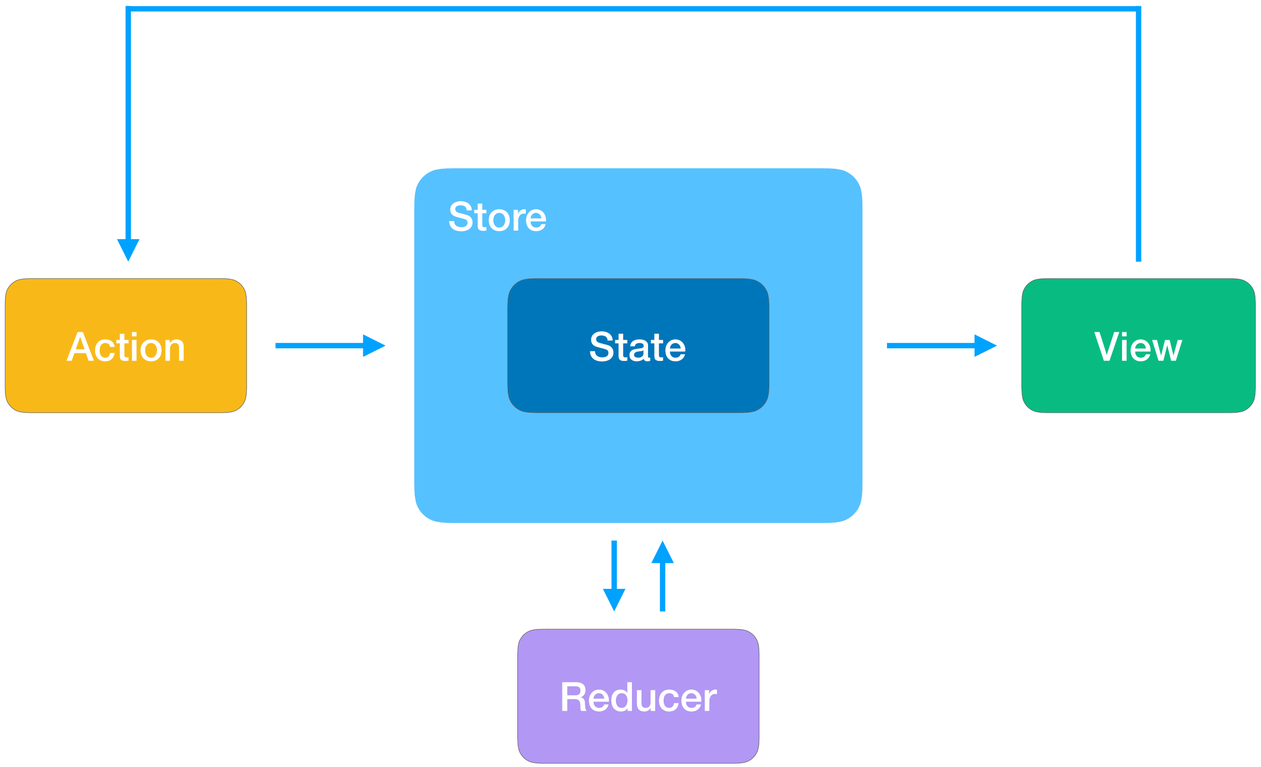
- Реализована по канонам Redux
- Имеет множество примеров и документацию
- Есть свой Router и Timetraveler
- Поддерживает Swift 4.0 и обновляется

github.com/ReSwift/ReSwift

Пример

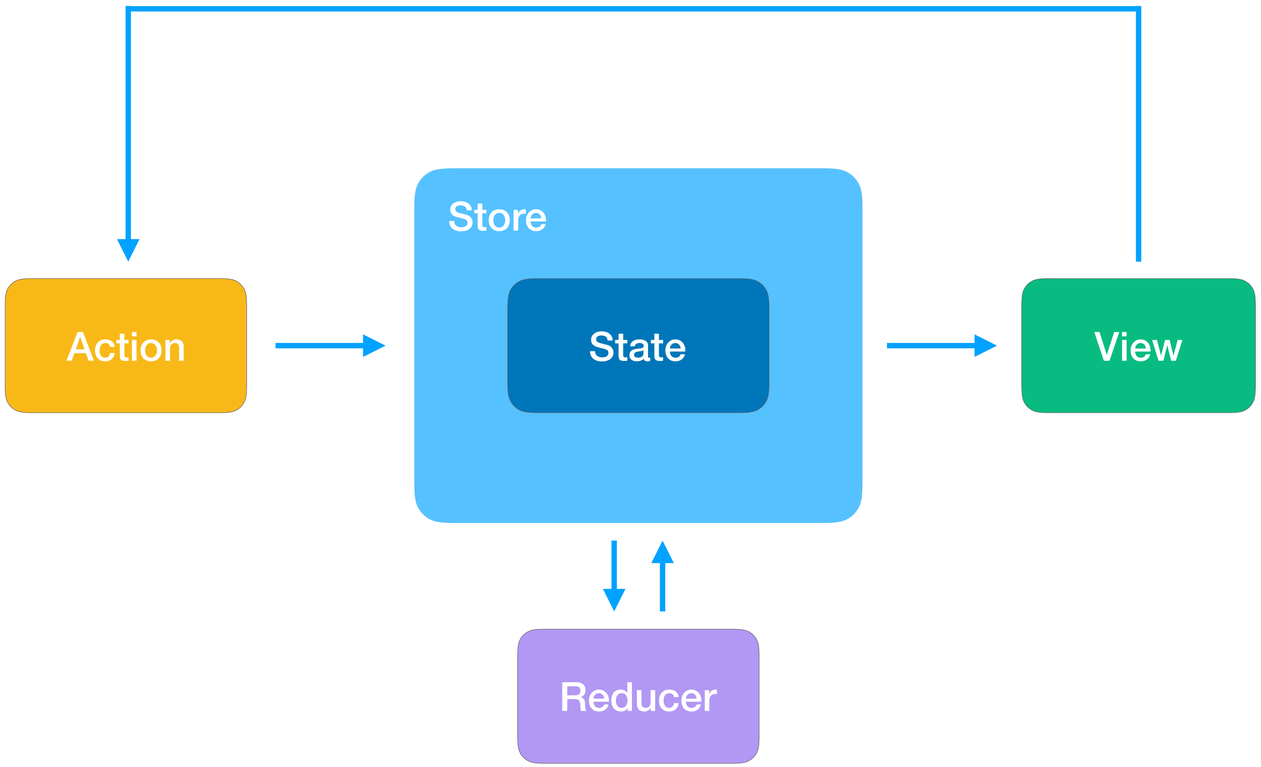
github.com/deniskirillov/repert




Store
Action
State
View
Reducer

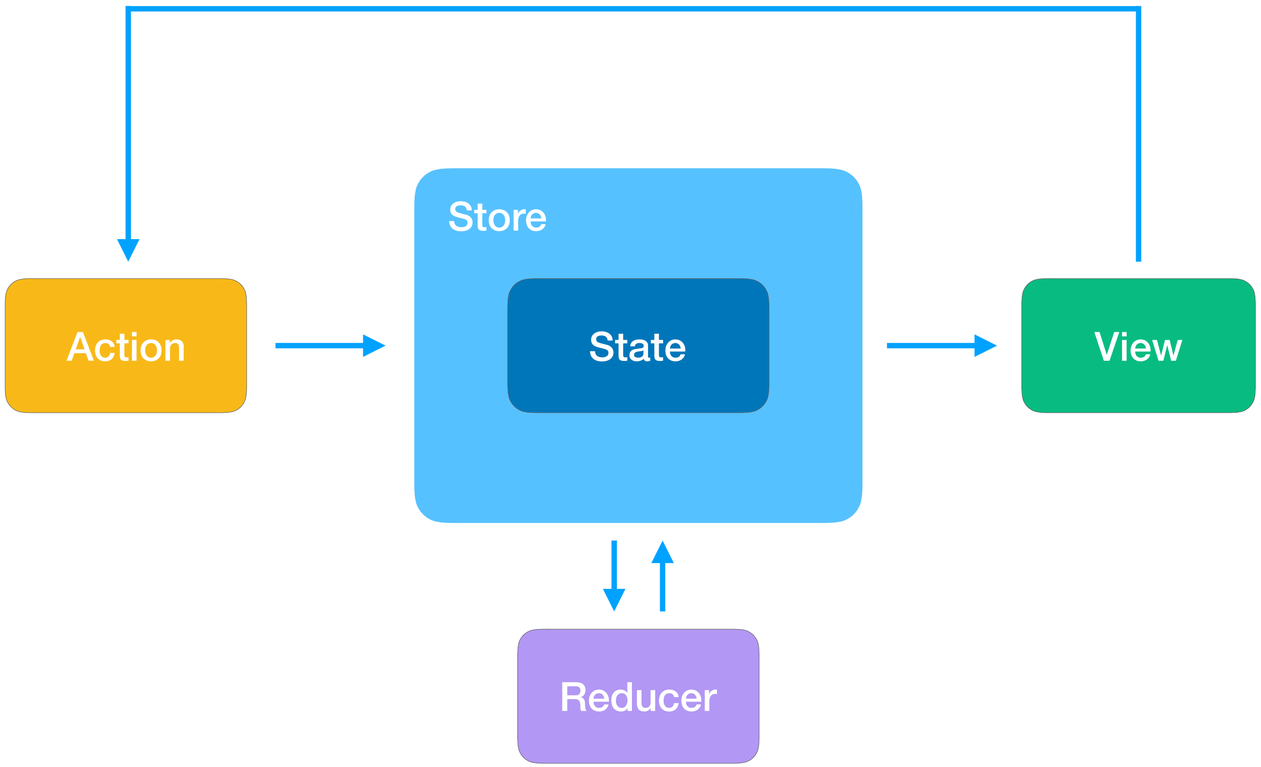
State

```
import ReSwift

struct AppState: StateType {

    // Входные данные
    var max:Float = 0
    var min:Float = 0
    var exp:Float = 0

    // Результат вычислений
    var result:Float = 0
}
```


Store
Action
State
View
Reducer

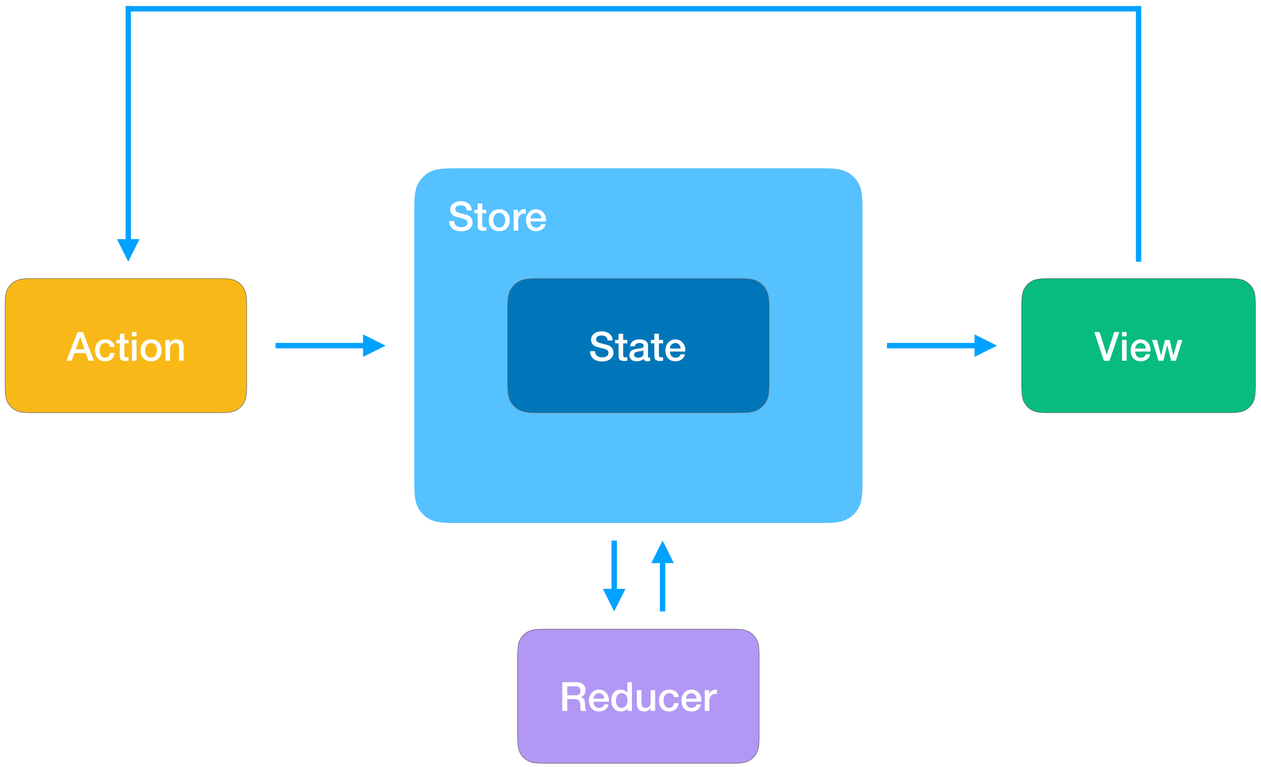
Action

```
import ReSwift

enum Actions : Action {

    // Обновление значений
    case updateMax(Float)
    case updateMin(Float)
    case updateExp(Float)

}
```


Store
Action
State
View
Reducer

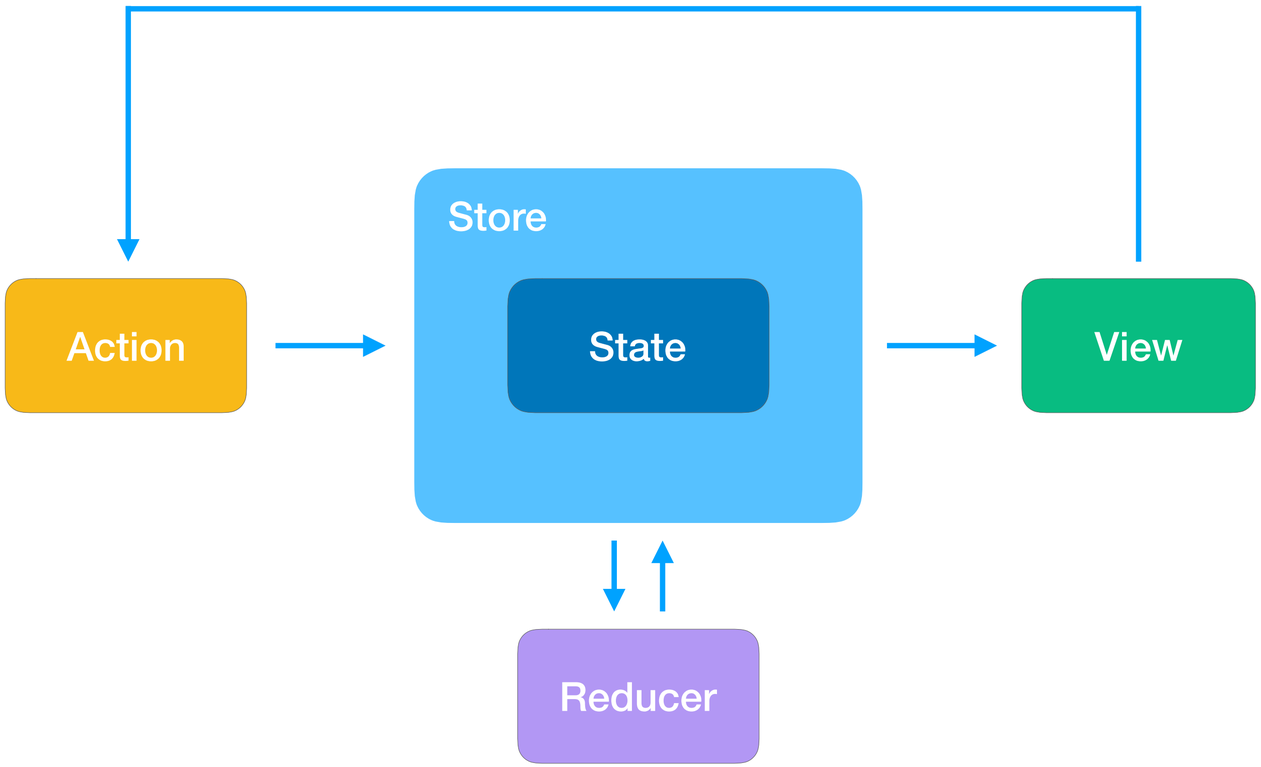
Reducer

```
import ReSwift

func rootReducer(_ action: Action, _ state: AppState?) -> AppState {

    var state = state ?? AppState()
    guard let action = action as? Actions else { return state }

    switch action {
    case .updateMax(let max):
        state.max = max
    case .updateMin(let min):
        state.min = min
    case .updateExp(let exp):
        state.exp = exp
    }

    state.result = (state.max + state.min + 4 * state.exp) / 6.0
    return state
}
```


Store
Action
State
View
Reducer

Store

```
import ReSwift

let store = Store<AppState>(
    reducer: rootReducer,
    state: nil,
    middleware:[]
)
```

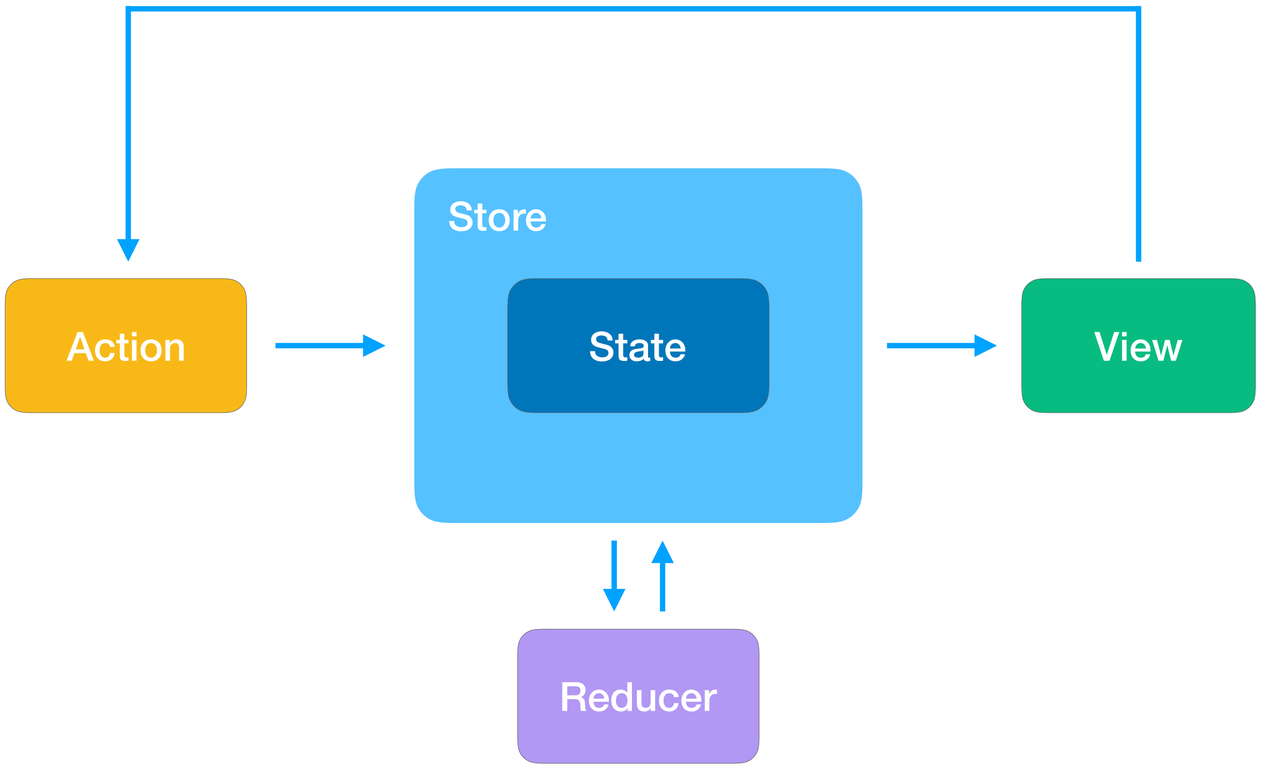

Store
Action
State
View
Reducer

View

```
class ViewController: UIViewController, StoreSubscriber {

    override func viewDidAppear(_ animated: Bool) {
        store.subscribe(self)
    }

    func newState(state: AppState) {

        minLabel.text = String(Int(state.min))
        maxLabel.text = String(Int(state.max))
        expLabel.text = String(Int(state.exp))

        minSlider.value = state.min
        maxSlider.value = state.max
        expSlider.value = state.exp

        resultLabel.text = String(format: "%.1f", state.result)
    }
```

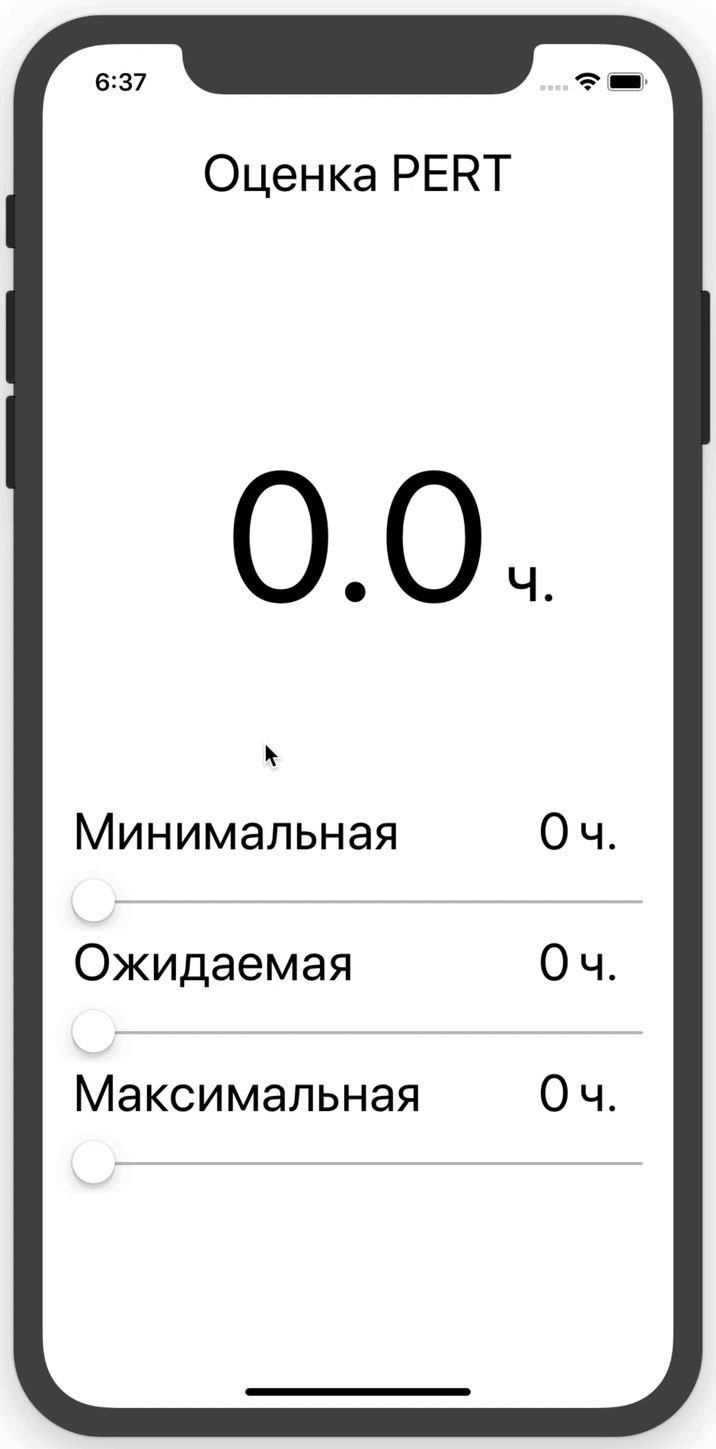
View

```
class ViewController: UIViewController, StoreSubscriber {

    @IBAction func onMaxChanged(_ sender: Any) {
        let value = maxSlider.value.rounded()
        store.dispatch(Actions.updateMax(value))
    }

    @IBAction func onMinChanged(_ sender: Any) {
        let value = minSlider.value.rounded()
        store.dispatch(Actions.updateMin(value))
    }

    @IBAction func onExpChanged(_ sender: Any) {
        let value = expSlider.value.rounded()
        store.dispatch(Actions.updateExp(value))
    }
```


Store
Action
State
View
Reducer

Пример

github.com/deniskirillov/repert



Проблемы и решения

1. Реализация State

(Shape of State)

Shape of State

```
struct AppState: StateType {

    // Data domain
    let users:Fetchable<User>
    let comments:Fetchable<Comment>

    // Service domain
    let routerState:RouterState
    let sessionState:SessionState

    // View domain
    let feedViewState:FeedViewState?
    let userViewState:UserViewState?
}
```

Shape of State

```
struct AppState: StateType {

    // Data domain
    let users:Fetchable<User>
    let comments:Fetchable<Comment>

    // Service domain
    let routerState:RouterState
    let sessionState:SessionState

    // View domain
    let feedViewState:FeedViewState?
    let userViewState:UserViewState?
}
```

Shape of State

```
struct AppState: StateType {

    // Data domain
    let users:Fetchable<User>
    let comments:Fetchable<Comment>

    // Service domain
    let routerState:RouterState
    let sessionState:SessionState

    // View domain
    let feedViewState:FeedViewState?
    let userViewState:UserViewState?
}
```

Shape of State

```
struct AppState: StateType {

    // Data domain
    let users:Fetchable<User>
    let comments:Fetchable<Comment>

    // Service domain
    let routerState:RouterState
    let sessionState:SessionState

    // View domain
    let feedViewState:FeedViewState?
    let userViewState:UserViewState?
}
```

Shape of State

```
func rootReducer(_ action: Action, _ state: AppState?) -> AppState {
    let state = state ?? AppState()
    return AppState(









    )
}
```

Shape of State, рекомендации

- Выполнять декомпозицию
- Реализовать протокол Equatable
- Удалять ненужные объекты
- Выполнять нормализацию

Shape of State, рекомендации

- Выполнять декомпозицию
- Реализовать протокол Equatable
- Удалять ненужные объекты
- Выполнять нормализацию

Shape of State, рекомендации

- Выполнять декомпозицию
- Реализовать протокол Equatable
- Удалять ненужные объекты
- Выполнять нормализацию

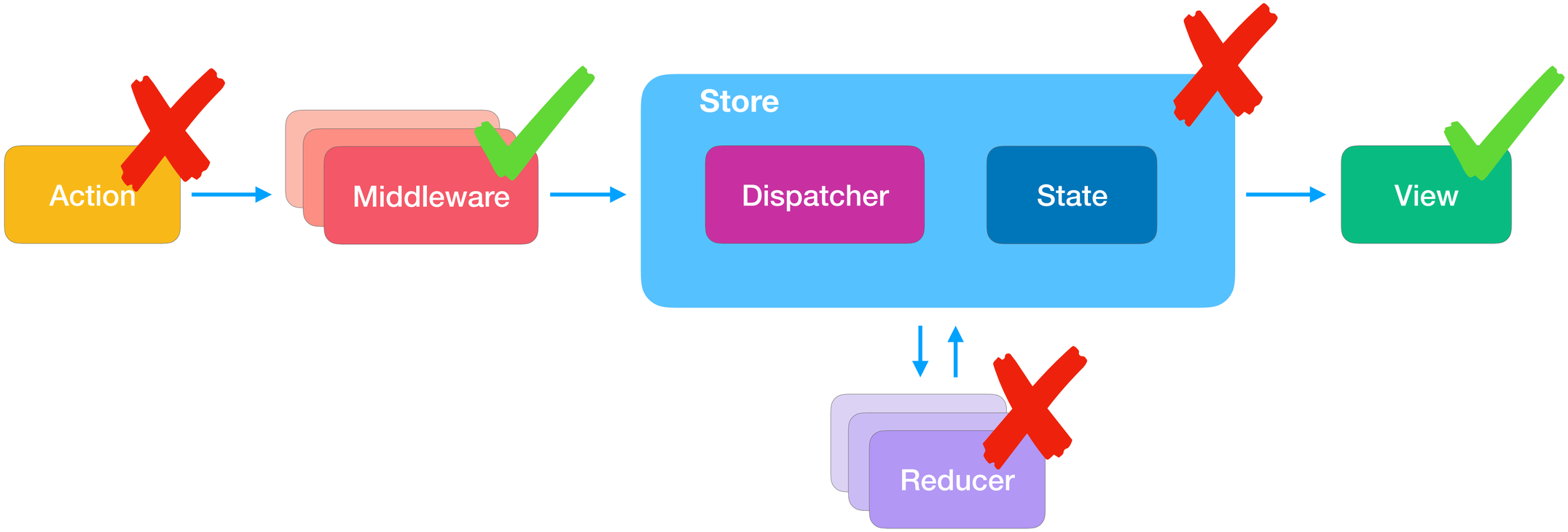
Shape of State, рекомендации

- Выполнять декомпозицию
- Реализовать протокол Equatable
- Удалять ненужные объекты
- Выполнять нормализацию

2. Асинхронные вызовы

(Async call)

Где делать Async call?



Async call из View

```
func actionCreator(_ state: State, _ store: Store) -> Action?
```

Action Creator

```
// ActionCreator
func fetchUsers(state:AppState, store: Store<AppState>) -> Action? {

    NetworkService.shared.fetchUsers { users, error in
        if error == nil {
            store.dispatch(UsersAction.update(users))
        } else {
            store.dispatch(UsersAction.markAsFailedToFetch(error))
        }
    }

    return UsersAction.didStartFetching
}
```

Используем Action Creator

```
class SomeViewController : UIViewController, StoreSubscriber {

    override func viewDidAppear(_ animated: Bool) {
        //...
        switch mainStore.state.users {
        case .notFetched, .failed(_):
            mainStore.dispatch(fetchUsers)
        default: break
        }
    }

    func newState(state: AppState) {
        // Do something with users
    }
}
```

Async call из Middleware

```
// Print any action to console
let logMiddleware: Middleware<AppState> = { dispatch, getState in
    return { next in
        return { action in
            print("Action > \(action)")
            next(action)
        }
    }
}
```

Async call из Middleware

```
func fetchUsersMiddleware(_ api:APIProtocol) -> Middleware<AppState> {
    return { dispatch, getState in
        return { next in
            return { action in // Action processing starts here
                if let action = action as? UsersAction, case .fetch = action {
                    api.fetchUsers { users, error in
                        if error == nil {
                            dispatch(UsersAction.update(users))
                        } else {
                            dispatch(UsersAction.markAsFailedToFetch(error))
                        }
                    }
                }
                next(action) // Call next middleware or reducer
            }
        }
    }
}
```

Async call из Middleware

```
let mainStore = Store<AppState>(
    reducer: appReducer,
    state: nil,
    middleware:[
        fetchUsersMiddleware(NetworkService()),
        logMiddleware
    ]
)
```

Async call из Middleware

```
mainStore.dispatch(UsersAction.fetch)
```

Async call, рекомендации

ActionCreator удобен в локальном контексте View

- Анимация, таймеры, преобразования

Middleware удобен в глобальном контексте

- Обращения к API и БД
- Логирование, статистика и т.п.

3. Flux + UIKit роутинг

(Against the framework)

Flux не дружит с UIKit

- Некоторые переходы в UIKit работают из коробки
- UIKit не разрешает одновременные переходы
- Универсального решения нет

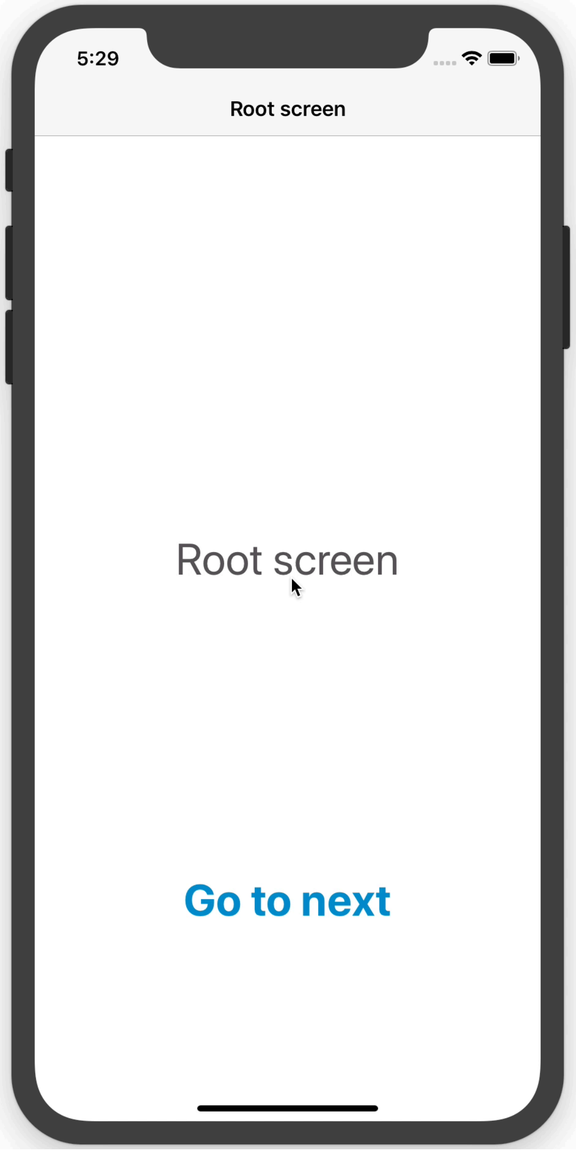
Flux не дружит с UIKit

- Некоторые переходы в UIKit работают из коробки
- UIKit не разрешает одновременные переходы
- Универсального решения нет

Flux не дружит с UIKit

- Некоторые переходы в UIKit работают из коробки
- UIKit не разрешает одновременные переходы
- Универсального решения нет

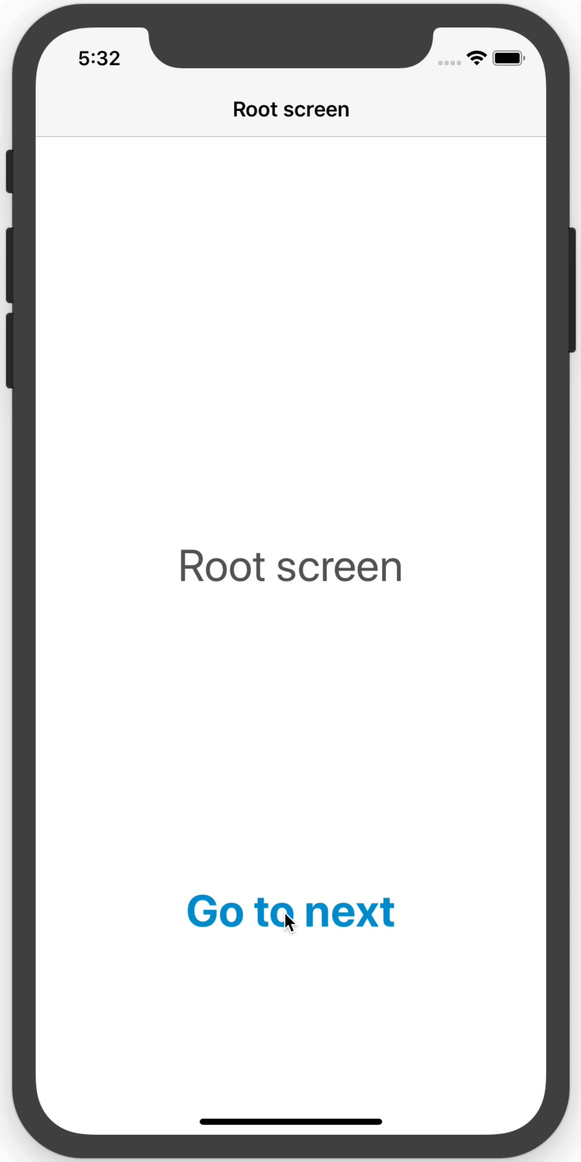
UINavigationController



UINavigationController

```
class ChildScreenViewController : UIViewController {

    var goBack:(() -> Void)?

    override func viewDidLoad() {
        super.viewDidLoad()
        self.navigationItem.leftBarButtonItem = buildBackButton()
    }

    private func buildBackButton() -> UIBarButtonItem {
        return UIBarButtonItem(title: "Back", style: UIBarButtonItem.Style.plain,
            target: self, action: #selector(internalGoBack))
    }

    @objc func internalGoBack() {
        goBack?()
    }
}
```

UIInteractiveGesture



UIInteractiveGesture

```
extension CustomNavigationController : UINavigationControllerDelegate {

    func navigationController(_ navigationController: UINavigationController,
        willShow viewController: UIViewController, animated: Bool) {

        if let topViewController = navigationController.topViewController?,
            coordinator = topViewController.transitionCoordinator {

            coordinator.notifyWhenInteractionChanges { (context) in
                if !context.isCancelled {
                    // Send router action NotifyCtrlDissmisedByGesture
                }
            }
        }
    }
}
```

Роутинг во Flux, рекомендации

- Не пытайтесь реализовать универсальный роутер
- Меньше анимации при переходах
- Изучить готовые решения

Роутинг во Flux, рекомендации

- Не пытайтесь реализовать универсальный роутер
- Меньше анимации при переходах
- Изучить готовые решения

Роутинг во Flux, рекомендации

- Не пытайтесь реализовать универсальный роутер
- Меньше анимации при переходах
- Изучить готовые решения

github.com/ReSwift/ReSwift-Router

Итоги

\+ Упрощает связи между компонентами

\+ Упрощает контроль над состоянием

\- Не интегрируется с другими паттернами

\- Плохо адаптируется с модульной архитектурой

\- Сложная реализация роутинга

\- Возможны проблемы с производительностью

Вывод

Знакомство с Flux лучше начинать в PET-проекте.

Q&A

facebook.github.io/flux/

github.com/ReSwift/ReSwift

academy.realm.io/posts/benji-encz-unidirectional-data-flow-swift/

blog.benjamin-encz.de/

github.com/deniskirillov/RePERT

medium.com/intive-developers/reswift-in-practice-1512e0f59eb5

karlbowden.com/reswift-prezzo/

Denis Kirillov
Head of iOS dev at Mamba
kirillov@mamba.ru

mamba
nice to meet you