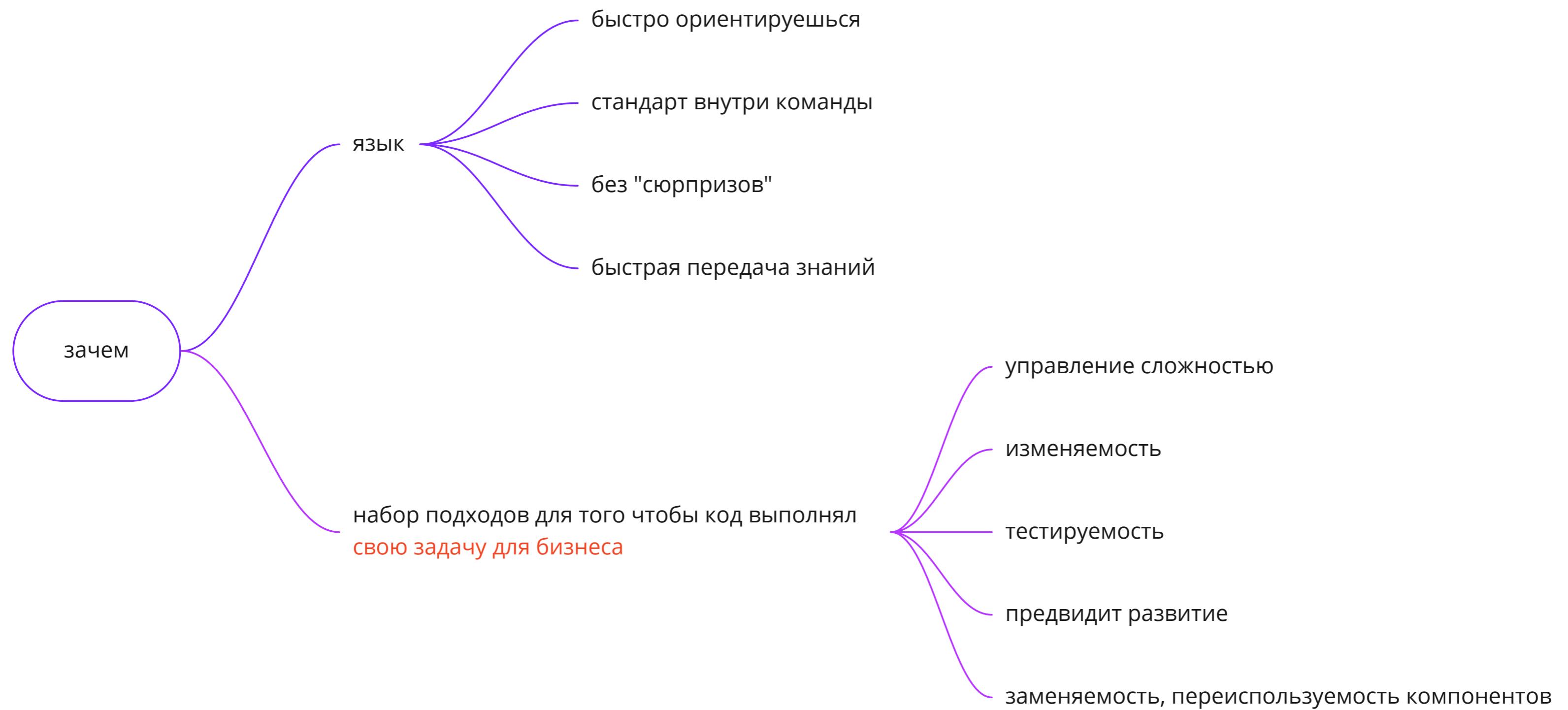


АРХИТЕКТУРА





SOA
(архитектура
приложения)

Архитектуры
презентационного
слоя

Способы управления
элементами
презентационного
слоя (координаторы)

mvp

mvvm

контроллер 1

контроллер 2

контроллер ввода
имейла

контроллер ввода
пароля

контроллер
подписки

достаёт
данные

работа с
сетью

работа с
БД

кэш

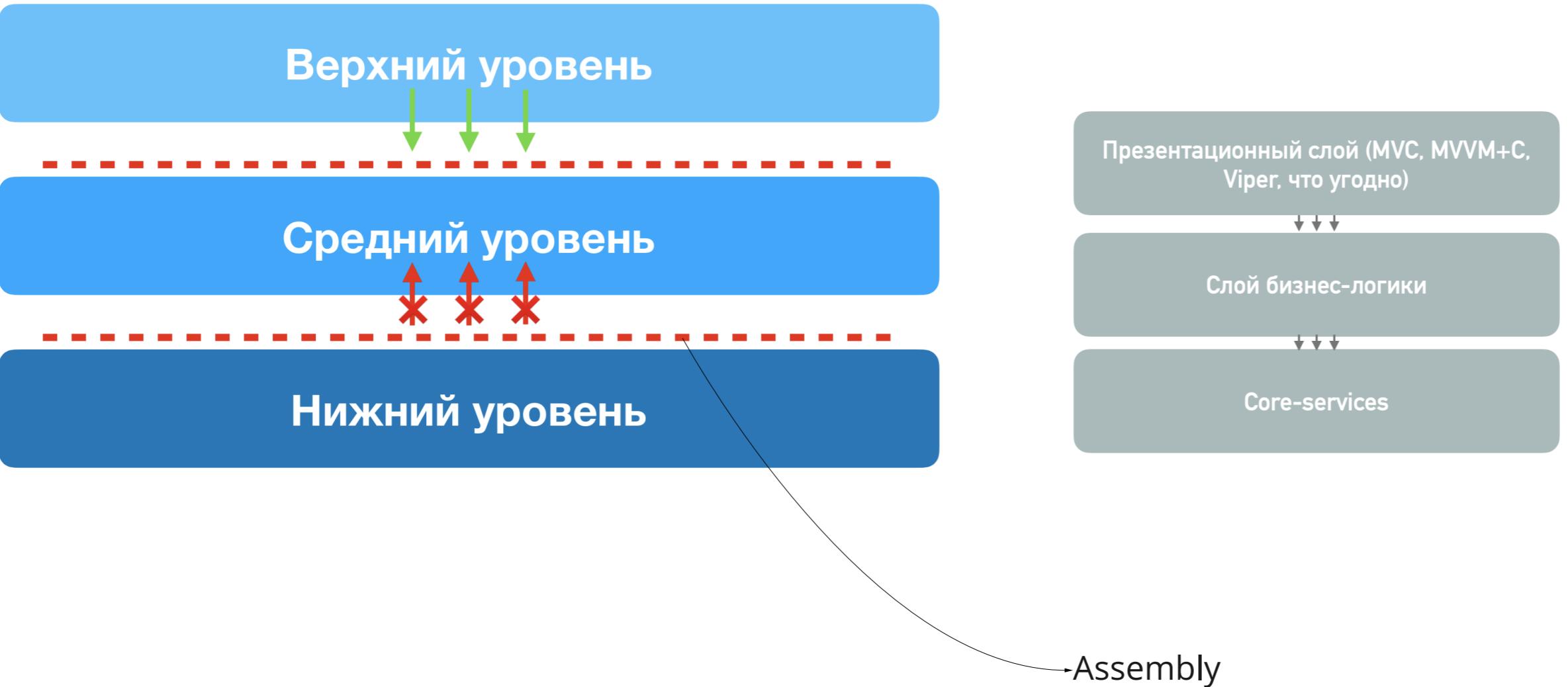
обрезаем
картинки

логика
логина

сервис
логина

КАКИМ ДОЛЖЕН БЫТЬ ХОРОШИЙ СЕРВИС

- Переиспользуемый
 - Начинается с описания его протокола (Input, Output) *Пример*
 - ~~Не привязывается к UI~~
 - Работает со своими модельками
- Поддерживаемый / расширяемый
- Заменяемый
 - Нет плотной связи с другими сервисами того же уровня
(только через протоколы)
 - Не хранит внешнее состояние

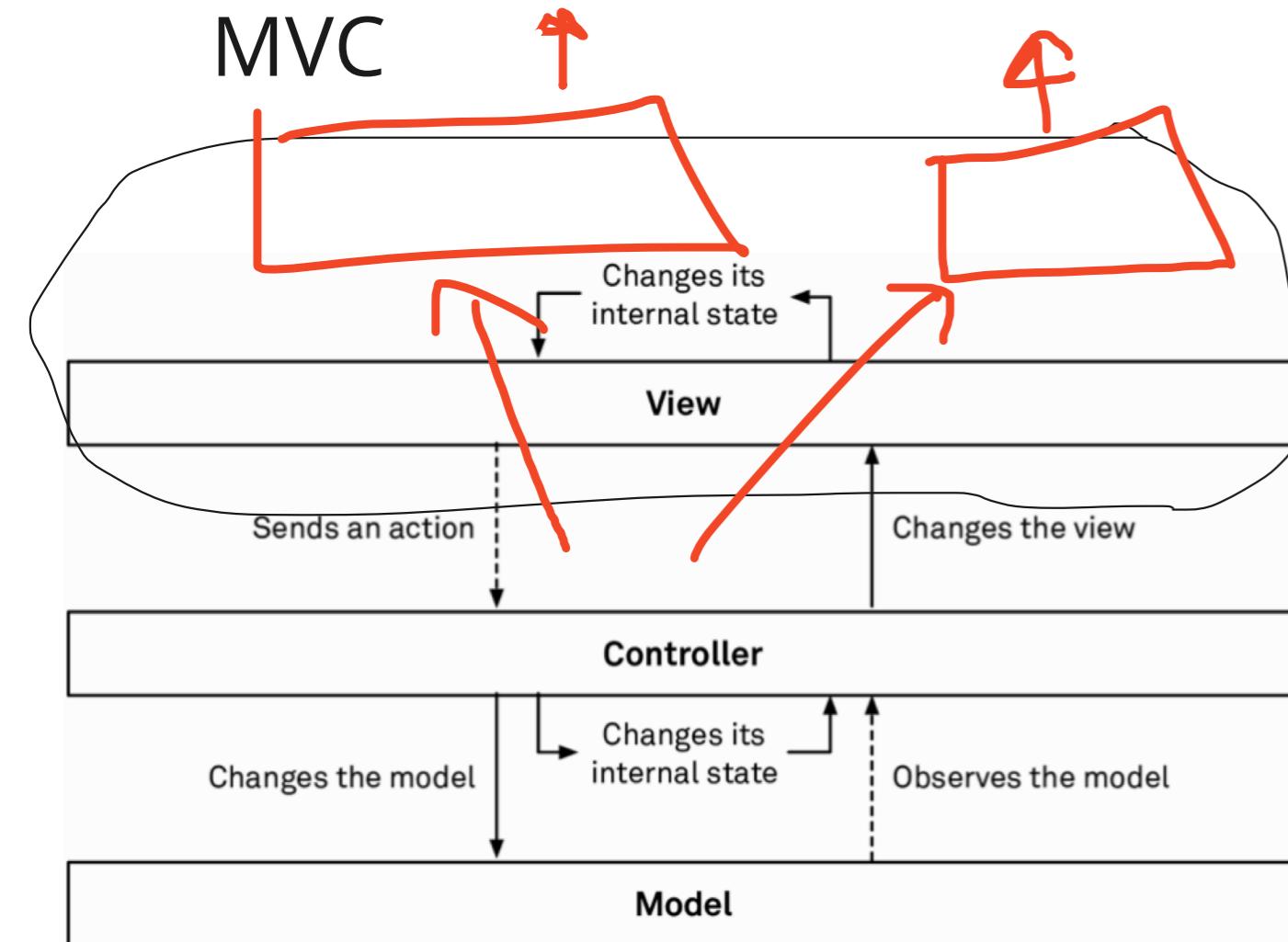


например:

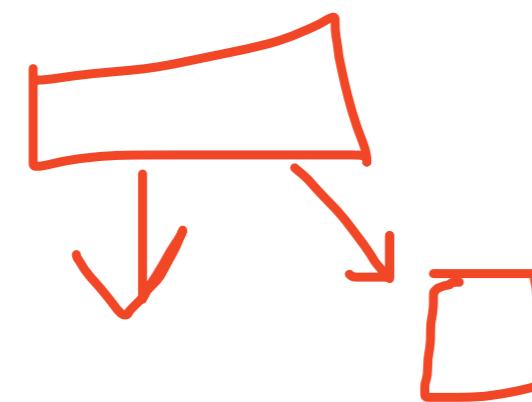
контроллеры и их модели, роутинг

обрабатывают данные

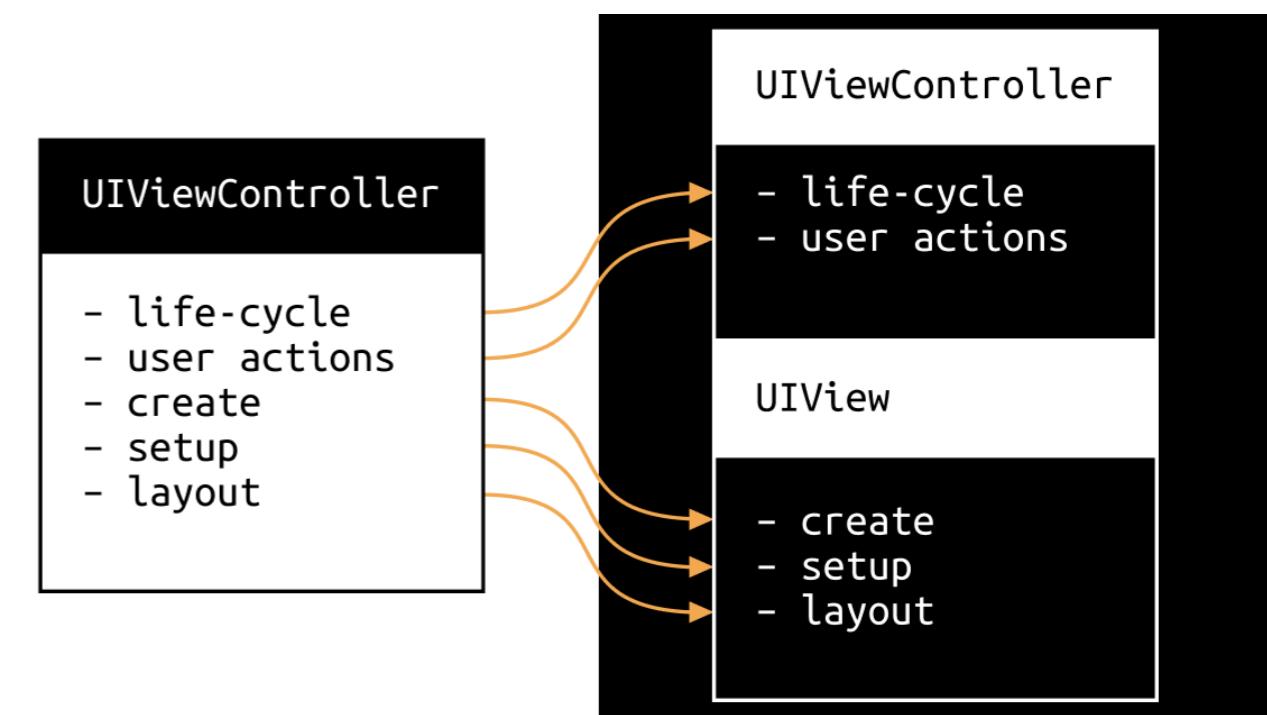
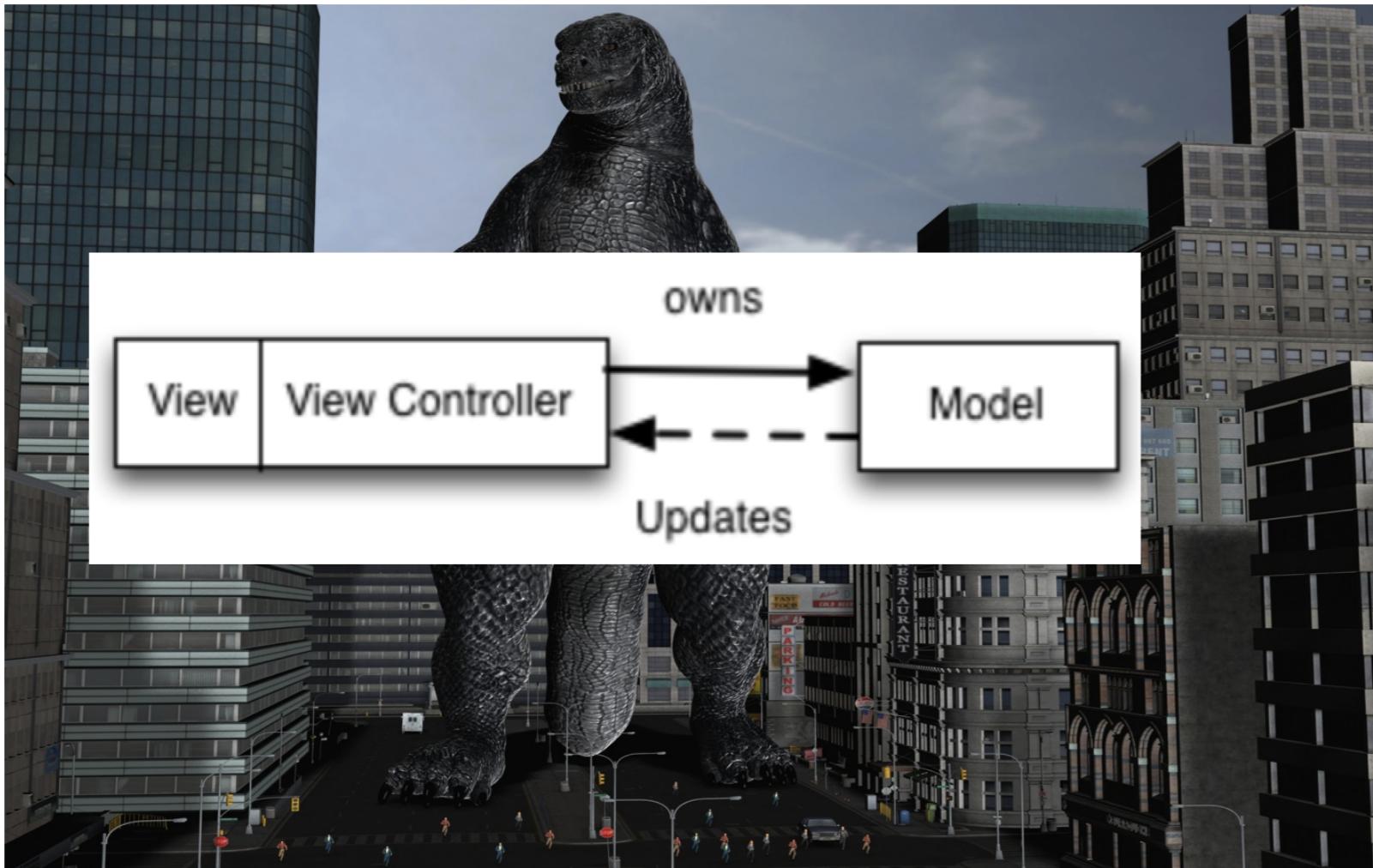
лазают в сеть, в базу данных и тд



- Устройство:
- Модели держатся контроллером
- Обновление модели:
Контроллер получает события и обновляет модель.
- Обновление вью:
Контроллер подписан на обновления модели (делегатами либо). Когда ему прилетает событие от модели, он меняет вьюхи
- Состояние вью:
Хранится в самом вью, или в контроллере.



НО...



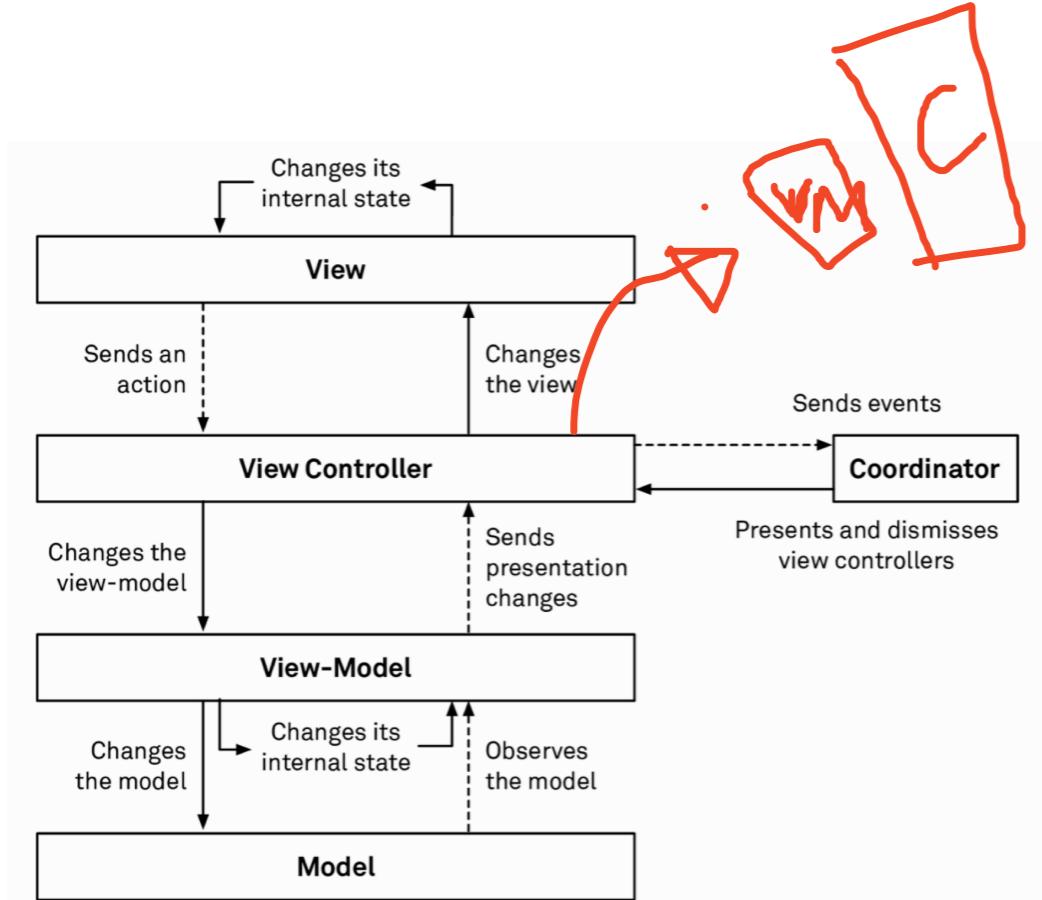
КОНТРОЛЛЕР СТАНОВИТСЯ ГИГАНТСКИМ

MVP

- Как в MVVM
- Но когда надо изменить значения, презентер пушит их вьюхам, потому что видит их интерфейсы.
(MVVM: "эй, делегат, тут поменялось значение, отреагирий плиз").
(MVP: "теперь ты круг").
- Он может быть закрыт протоколом

```
11 protocol MiddleModuleView: AnyObject {
12     func updateName(_ name: String)
13 }
14
15 class MiddleModuleViewController: UIViewController, MiddleModuleView {
16     private var presenter: MiddleModulePresenter!
17
18     private var label = UILabel()
19
20     public convenience init(
21         something: String,
22         somethingElse: [String]?,
23         deps: ServiceLocator,
24         out: @escaping MiddleModuleOut
25     ) {
26         self.init()
27         self.presenter = MiddleModulePresenter(view: self, out: out, deps: deps, something: something)
28     }
29
30     func updateName(_ name: String) {
31         label.text = name
32     }
33 }
34
35 import Foundation
36
37 class MiddleModulePresenter {
38     private weak var view: MiddleModuleView?
39     private let out: MiddleModuleOut
40     private var deps: ServiceLocator
41
42     init(
43         view: MiddleModuleView,
44         out: @escaping MiddleModuleOut,
45         deps: ServiceLocator,
46         something: String
47     ) {
48         self.view = view
49         self.out = out
50         self.deps = deps
51     }
52
53     private var name: String = ""
54
55     public func foo() {
56         deps.networkService.loadData({ model in
57             self.name = model.letters.reduce(into: "", { res, letter in
58                 res += letter
59             })
60             view?.updateName(name)
61         })
62     }
63 }
64
65 }
```

MVVM



- Устройство:
 - Объекты создаются контроллерами
 - Модели держатся вьюмоделями, а не контроллерами
 - Вьюмодель занимается подготовкой данных для вьюшек
- Обновление модели:

Все события вью ничего не меняют ни у вью, ни у контроллера, а сразу передаются вьюмодели. А уже вьюмодель меняет модель (или вью/контроллер)
- Обновление вью:

Контроллер подписан на обновления вьюмодели (делегатами либо биндингами). Когда ему прилетает событие от модели, он меняет вьюхи
- Состояние вью:

Хранится в самом вью, или во вьюмодели.

```
class MMVMViewController: UIViewController {  
    private var viewModel: MVVMViewModel  
  
    private var label = UILabel()  
  
    init(  
        viewModel: MVVMViewModel  
    ) {  
        self.viewModel = viewModel  
        super.init(nibName: nil, bundle: nil)  
        self.viewModel.updated = { [weak self] in  
            self?.label.text = viewModel.name  
        }  
    }  
  
    required init?(coder: NSCoder) {  
        fatalError("init(coder:) has not been implemented")  
    }  
}
```

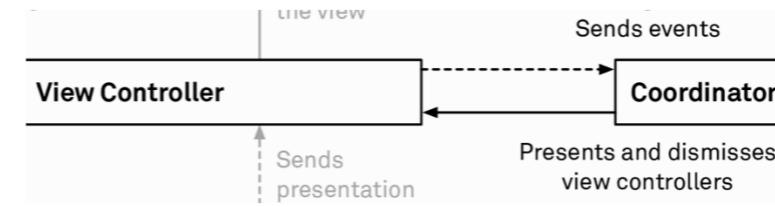
```
9  
10 class MVVMViewModel {  
11     private var deps: ServiceLocator  
12  
13     var updated: (() -> Void)?  
14  
15     init(deps: ServiceLocator) {  
16         self.deps = deps  
17     }  
18  
19     public var name: String = ""  
20  
21     func fetchData() {  
22         deps.networkService.loadData({ model in  
23             self.name = model.letters.reduce(into: "", { res, letter in  
24                 res += letter  
25             })  
26             updated?()  
27         })  
28     }  
29  
30 }  
31  
32 }
```

Управление элементами презентационного слоя

Coordinator

Controller
of
controllers

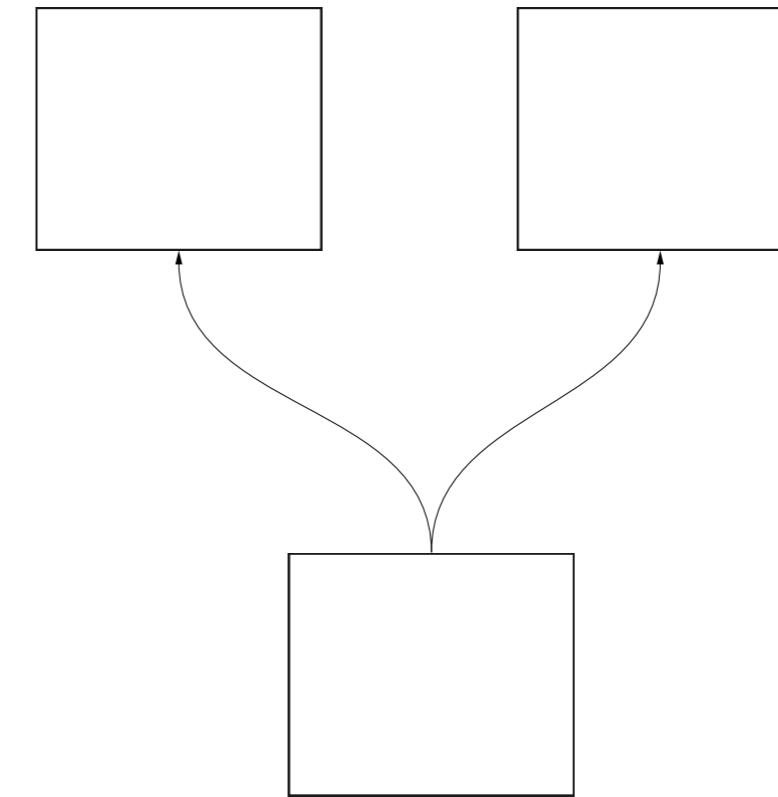
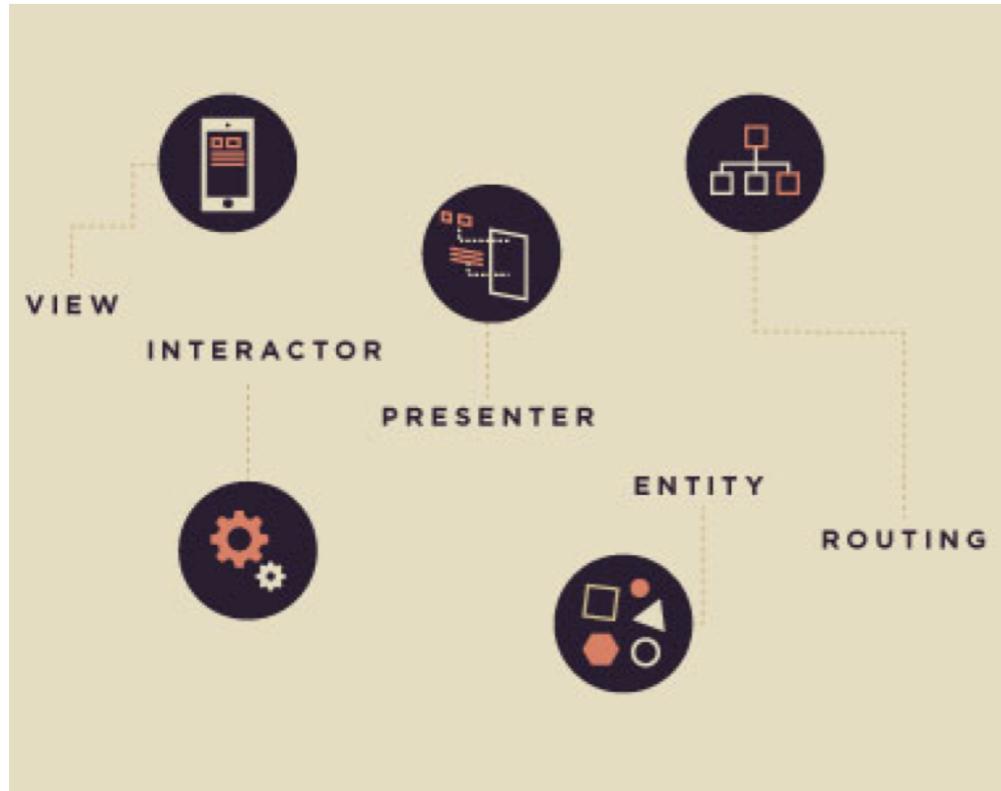
КООРДИНАТОР



```
self.delegate?.didSelect($0) —→ extension Coordinator: FolderViewControllerDelegate {  
    func didSelect(_ item: Item) {  
        switch item {  
            case let folder as Folder:  
                let folderVC = storyboard.instantiateViewController(  
                    with: folder, delegate: self)  
                folderNavigationController.pushViewController(folderVC,  
                                                animated: true)  
        // ...  
    }  
}
```

Обычно координатор "висит" на какой-то долгоживущей сущности, например на UIApplicationDelegate.
Поэтому можно через него передавать зависимости в контроллеры.

How did it come to VIPER



And what to do about it

НЕ Я ОДИН ТАК СЧИТАЮ

To enforce the rules about referencing in one direction only, these patterns require a *lot* of protocols, classes, and passing of data between layers. For this reason, many developers using these patterns use code generators. Our feeling is that code generators — and any patterns verbose enough to imply their need — are misguided. Attempts to bring “Clean Architecture” to Cocoa usually claim to manage “massive view controllers,” but ironically, do so by making the code base even larger.

While interface decomposition is a valid approach for managing code size, we feel it should be performed as needed, rather than methodically and per view controller. Decomposition should be performed along with knowledge of the data and tasks involved so that the best abstraction — and hence the best reduction in complexity — can be achieved.

Время на
обеспечение
качества =
Общее время на
задачу - время на
создание кода.



Теперь у тебя нет огромного
контроллера... Зато теперь у
тебя огромное приложение
ахахахаха!

И, пока ты это писал,
твой старта разорился))

Советы

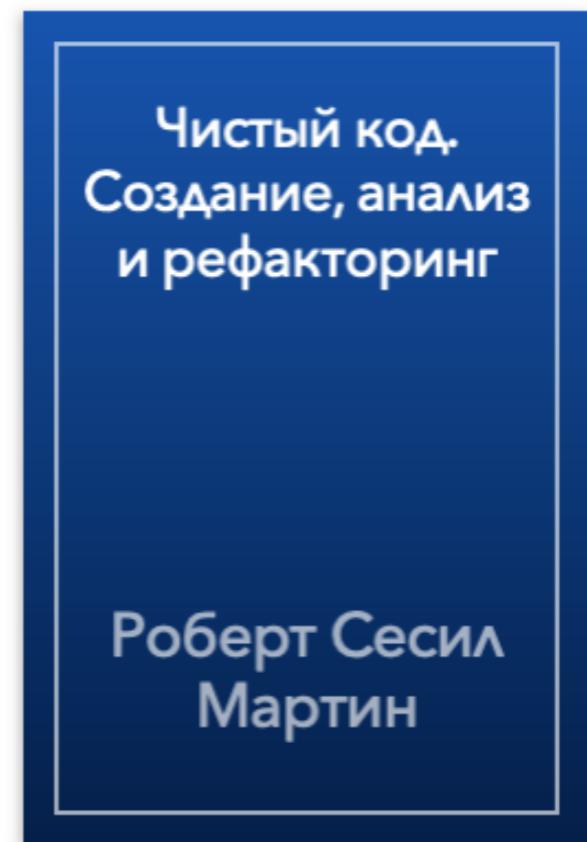
РИСОВАТЬ

шаблоны



ДЗ

Читать



Чистый

Код

Литература

- Прикольный MVC <https://habr.com/en/company/dododev/blog/432718/>
- Однонаправленные и двунаправленные архитектуры
<https://habr.com/en/post/466591/>
- □Про то, как в скаенге используют координаторы
<https://www.youtube.com/watch?v=Tvtgijhcmwk>
- Подлодка №5 - SOA и CleanSwift
- RedMadRobot - Архитектура iOS приложений
- Miro - что такое архитектура <https://t.me/miroengineering/21>
- □Чистый Код
- □ [objc.io](#) - книга архитектура iOS приложений (хайли рекоммендед)
- Референсный проект red mad robot <https://t.me/redmadnewsios/144>
- Серия архитектурных статей, в которых ребята из Big Nerd Ranch рассказывают об архитектурных подходах и паттернах, которые позволяют им оставаться гибкими и легко перестраивать свой продукт.
<https://www.bignerdranch.com/blog/agile-software-development-architecture-patterns-for-responding-to-change-part-1/>
- Подход к построению модульного сетевого слоя. <https://medium.com/flawless-app-stories/generic-network-layer-in-ios-development-2bffff780832>