

inter



PADRÕES DE ARQUITETURA DE DESENVOLVIMENTO iOS



Antonio Netto

Formado em ciência da computação pela Universidade Católica de Pernambuco.

12 anos de experiência em desenvolvimento de software e desenvolvedor iOS desde 2011.

Atualmente Arquiteto de Soluções Digitais no squad de Investimentos do **Inter**.

PAI Investimentos

Total investido
comece a investir

Qual seu momento de investidor?
descubra

Vamos começar?

Prefixado

Mais Procurados
Renda Fixa

Ativos

Amplie seus investimentos

Renda Fixa

Um portfólio de produtos para investir com segurança e mais rentabilidade.

Fundos de Investimento

Encontre o fundo ideal para você e diversifique seus investimentos.

Home Broker

Negocie ações de forma simples
● MERCADO FECHADO

Busca 20:30 8% 🔋

Amplie seus investimentos

Renda Fixa
Um portfólio de produtos para investir com segurança e mais rentabilidade.

Fundos de Investimento
Encontre o fundo ideal para você e diversifique seus investimentos.

Home Broker
Negocie ações de forma simples
● MERCADO FECHADO

100% GRATUITO

Ofertas Públicas
Ações, cotas de fundos de investimento e outros valores mobiliários para o mercado.

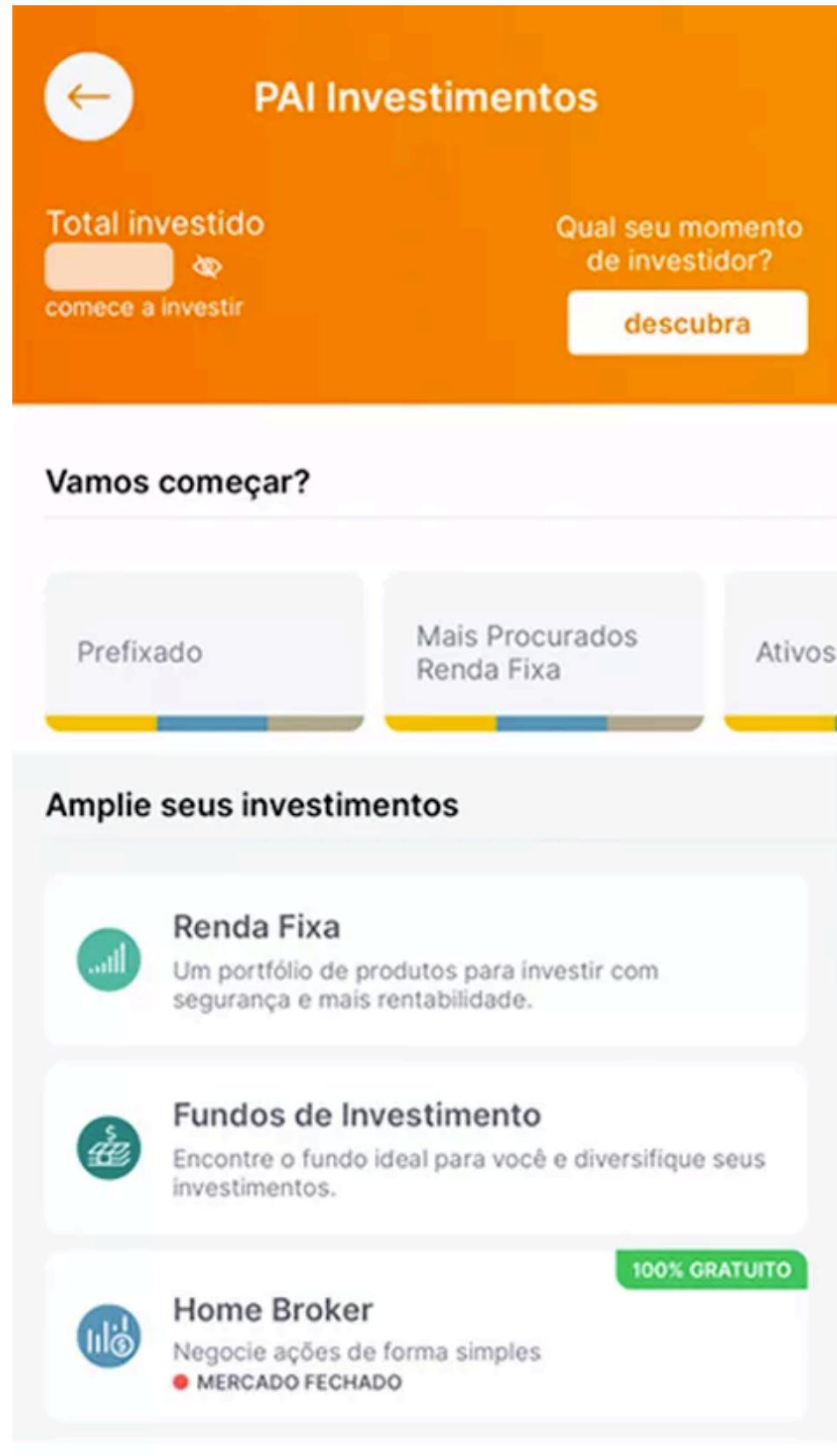
Previdência Privada
Investindo na previdência você garante a sua tranquilidade no futuro.

Poupança
Investimento fácil e seguro para quem quer começar uma reserva financeira.

100% GRATUITO

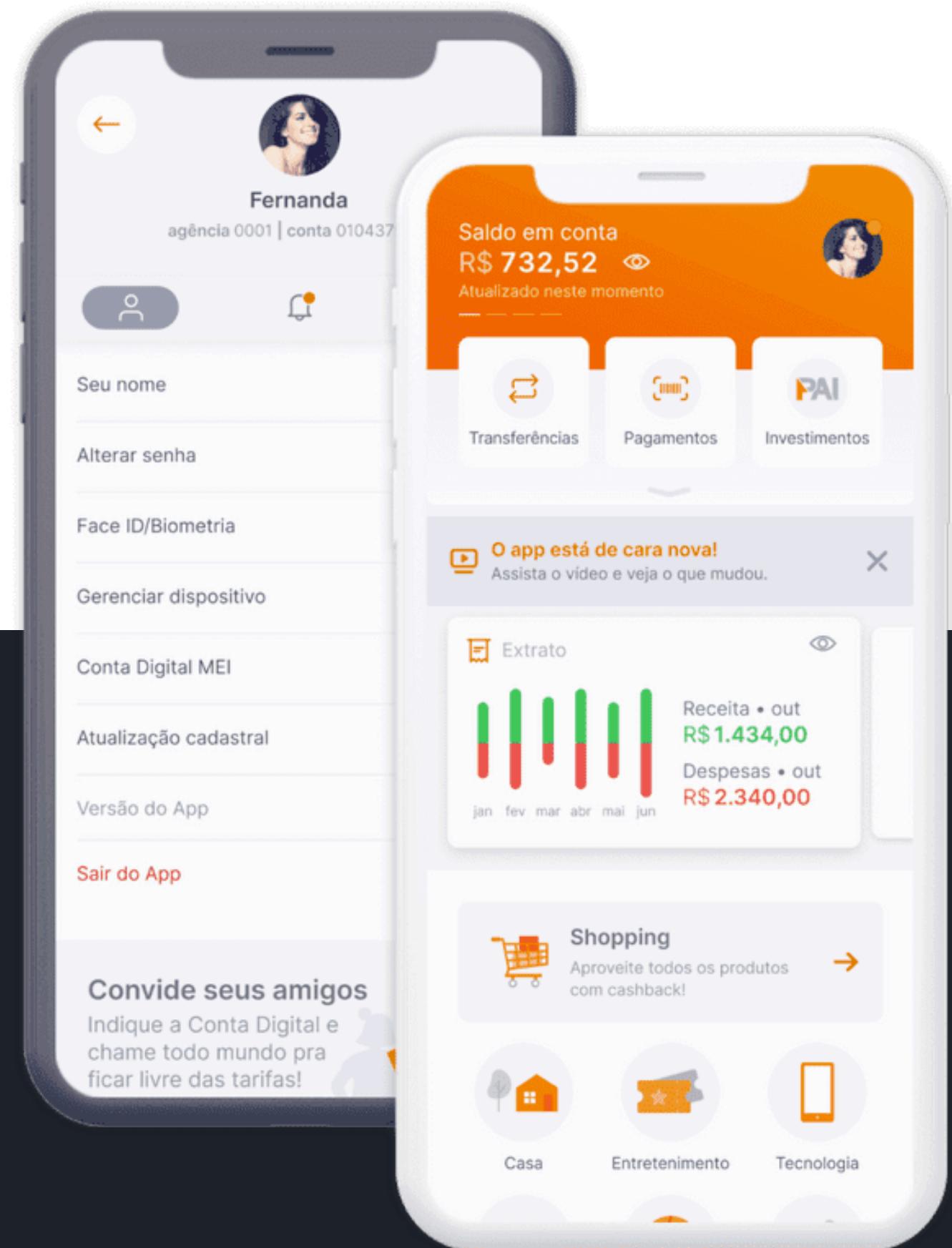
100% GRATUITO

PAI 1.0



PAI 2.0





Primeira Regra para desenvolver um app **SENSACIONAL**



Por que devo me preocupar com a escolha da Arquitetura de

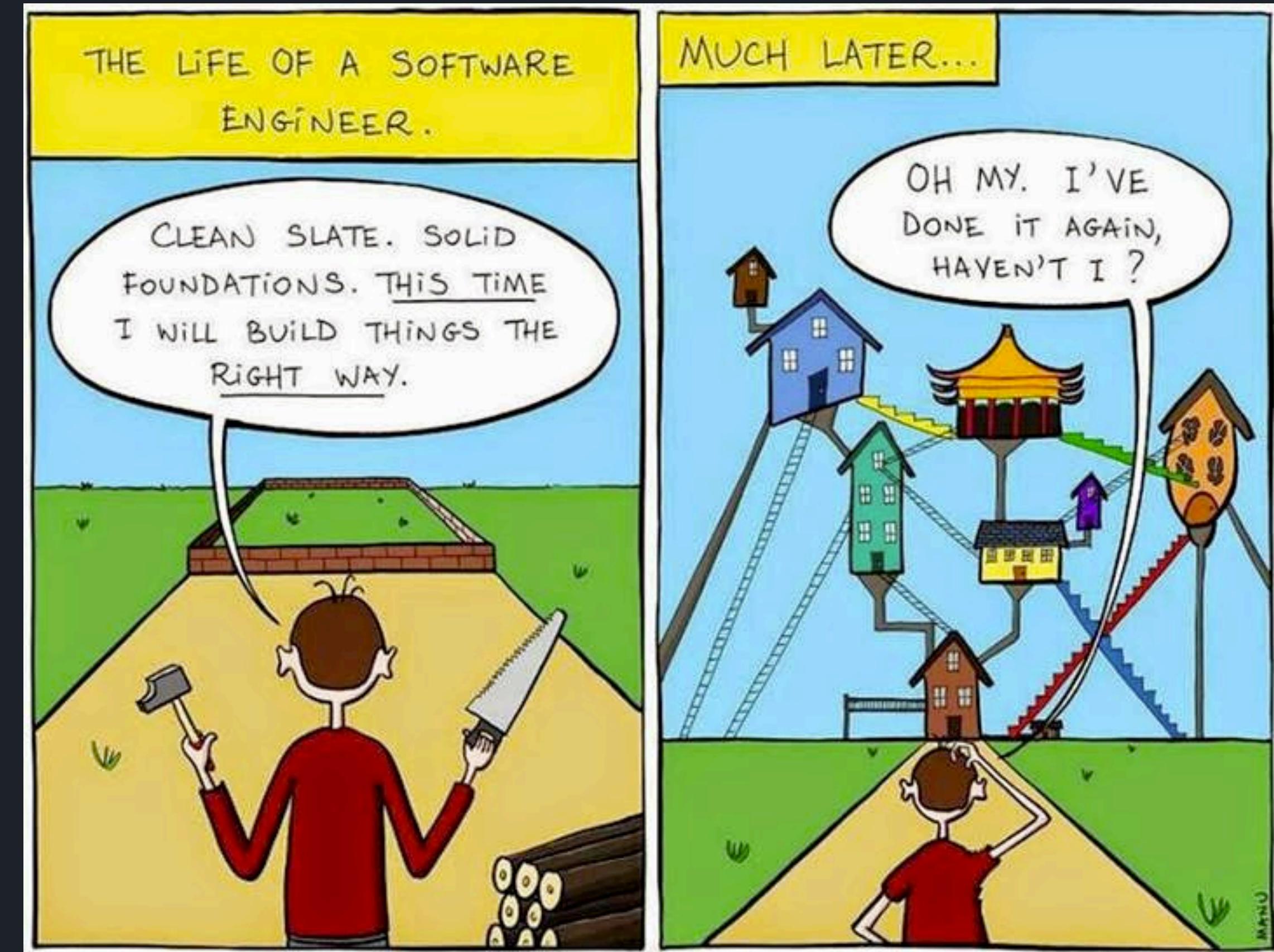
Se você não se importar, um dia, debugando uma classe com centenas de linhas de código e com diferentes coisas, você vai se encontrar perdido e incapaz de achar aquele bug.



Qual é a definição de uma boa arquitetura de software?

1. **Balanceamento de responsabilidades** através de entidades e regras bem definidas.
2. **Testabilidade** das diferentes funcionalidades do app.
3. **Fácil de usar** e baixo custo de manutenção

O que é Arquitetura de Desenvolvimento iOS?



Referência: swifting.io



O que é Arquitetura de Desenvolvimento iOS?

Resposta rápida: É o nível mais alto de design de sistema.

Design de Sistema?

Uma forma de facilitar a construção de um APP

Como detectar uma arquitetura ruim?

Pergunta: Por que o código do app é tão feio?

Resposta: Razões históricas...

Desafios de criar um APP

O maior desafio para um desenvolvedor é criar uma aplicação ROBUSTA que seja fácil de MANTER, TESTAR e ESCALAR.

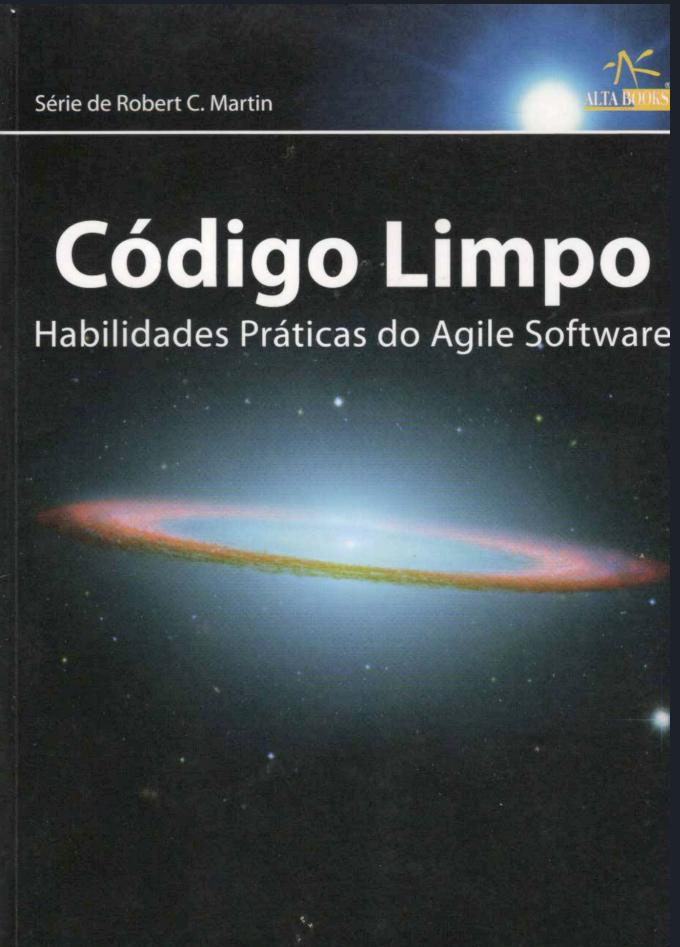
Mas e aquela frase de que “Código bom é código comentado” ?



O Código deve falar por si só!

“O único comentário verdadeiramente bom é aquele em que você encontrou uma forma para não escrevê-lo”.

Robert C. Martin (Uncle Bob)



@unclebobmartin

Principais Padrões de Arquitetura iOS

Apple MVC

A arquitetura mais famosa e utilizada MASSIVAMENTE por diversos desenvolvedores e projetos.

Inicialmente, foi a arquitetura oficial da guideline de desenvolvimento de apps da Apple.

RIP 2018

MVP

Introduzida em 1996 por **Mike Potel** e inicialmente usada pela IBM em aplicações C++ e Java.

Não existe ViewController. É apenas View e toda lógica é feita dentro de um Presenter.

VIPER

Baseada na Arquitetura Limpa de Robert C. Martin, VIPER se tornou uma das arquiteturas mais comentadas recentemente.

Trata-se das 5 camadas mais amadas (haters gonna hate) do clean architecture:

View
Interactor
Presenter
Entity
Router

VIP

Proposta por Raymond Law, VIP (a.k.a Clean Swift), é uma variação ou uma redução do VIPER.

VViewController
Interactor
Presenter

MVVM

Atualmente é a arquitetura oficial da guideline de desenvolvimento da Apple.

Trata-se de uma melhoria do MVP e a principal arquitetura a implementar a ideia de programação reativa funcional.

RXLovers <3

MVVM-C

MVVM é poderosa porém lhe faltava algo. Faltava um roteador ou coordenador para fazer as classes e o modulo se comunicarem com o mundo a fora.

E ninguém melhor que um coordinator.

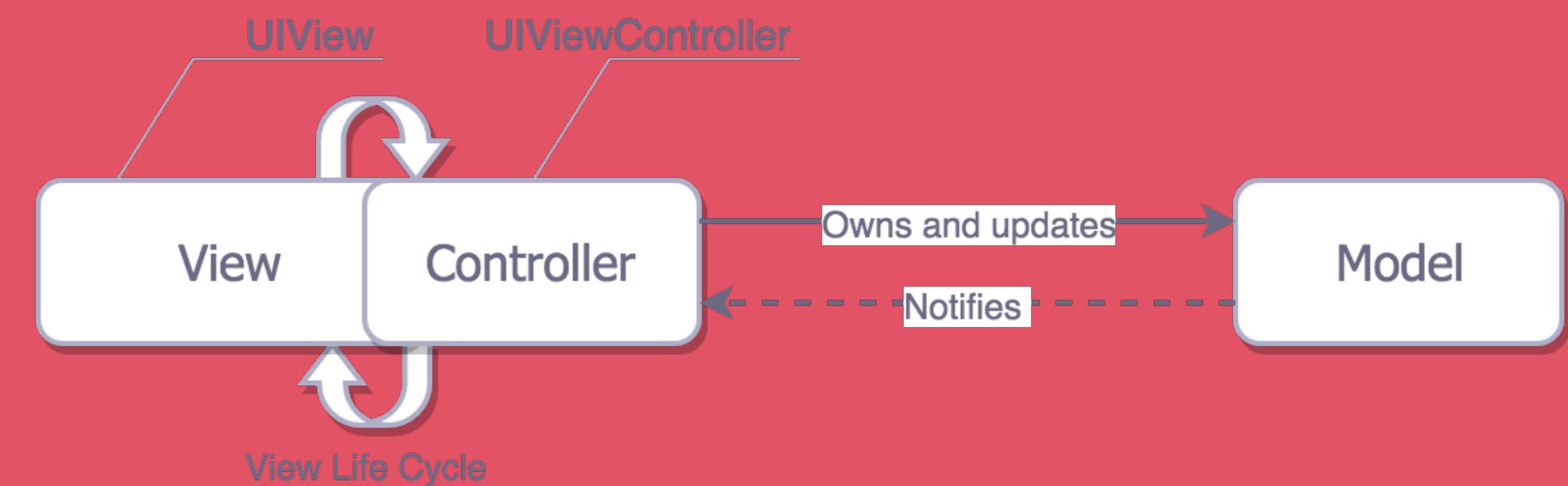
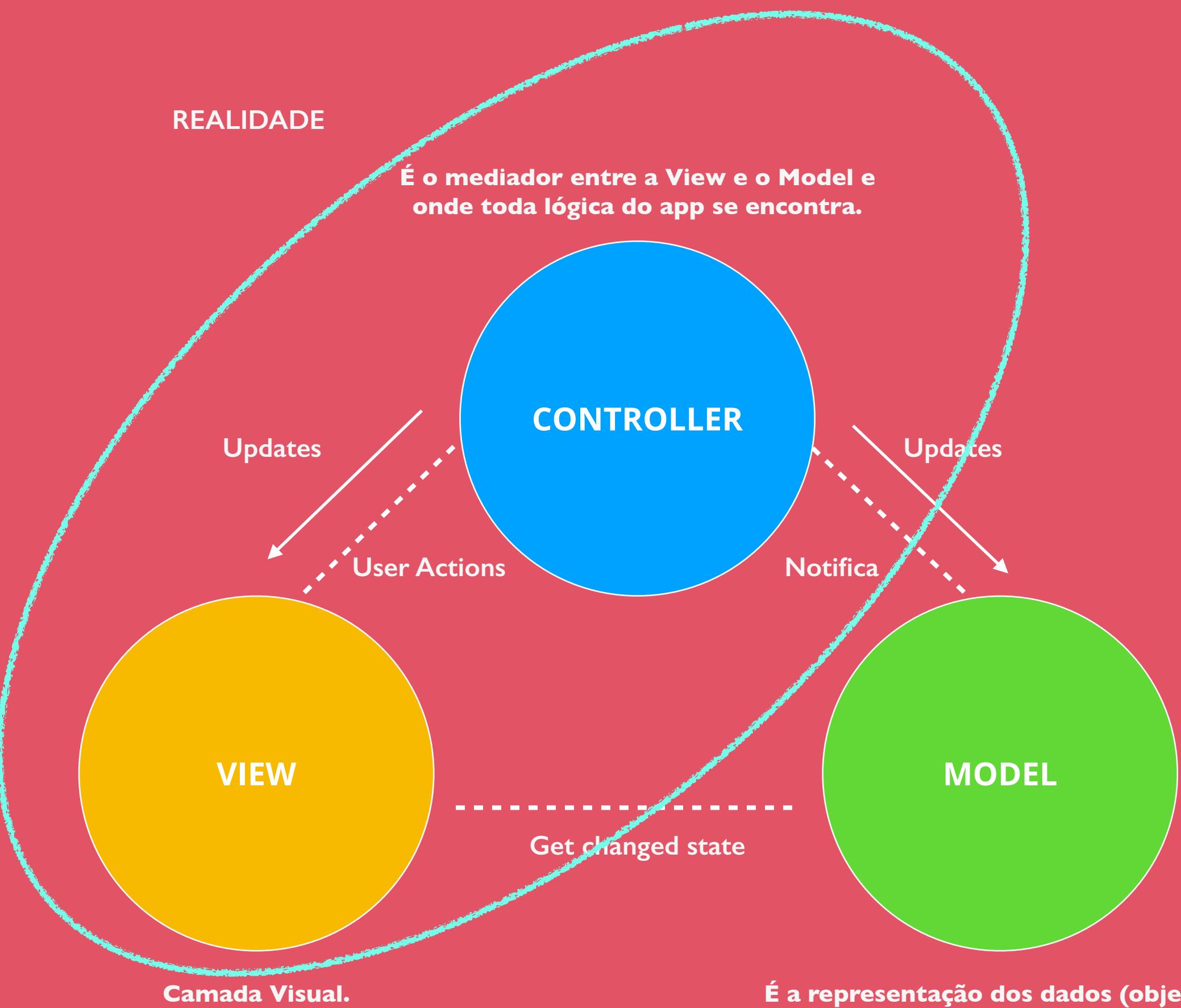
Esta é a principal arquitetura usada no SuperApp do **Inter**.

TCA

** BONUS **

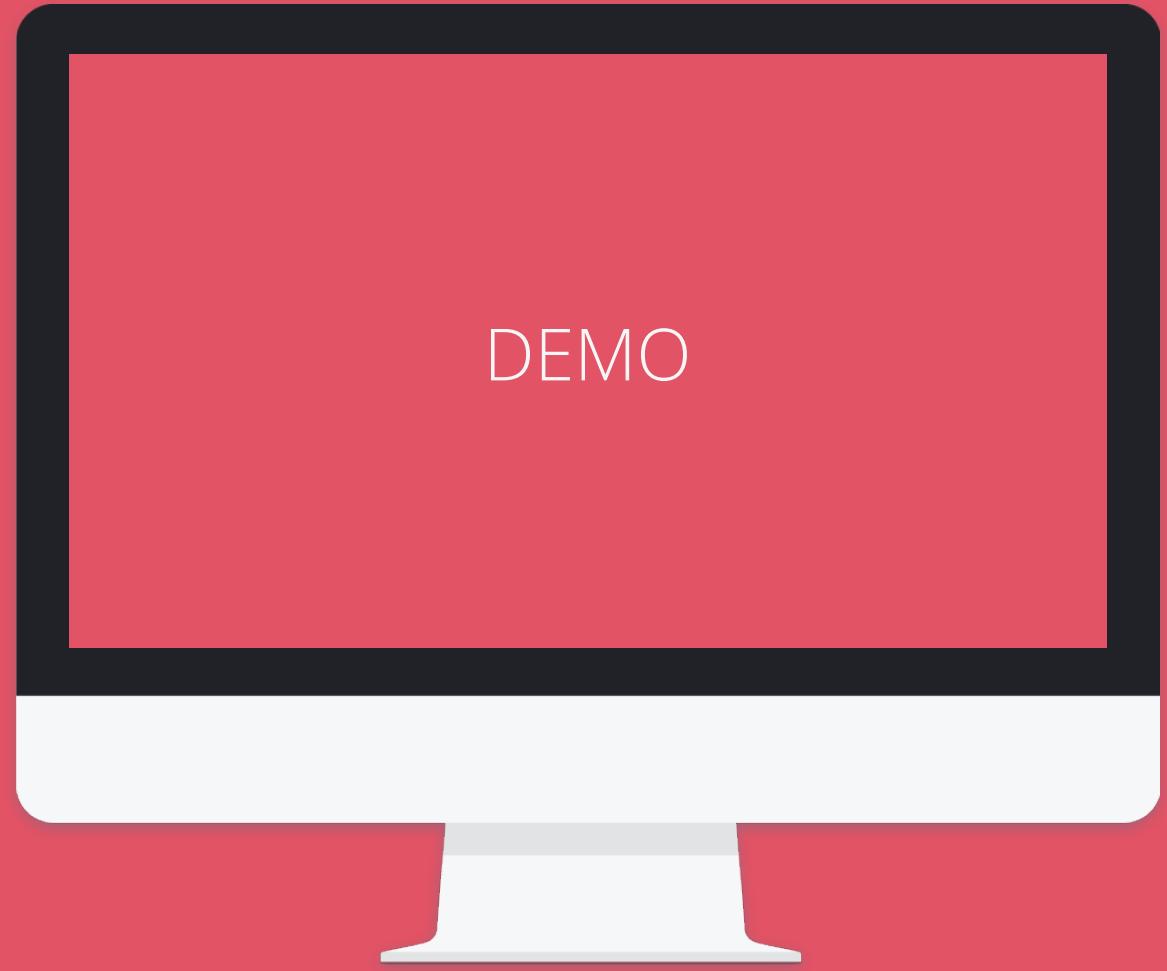
The Composable Architecture é uma proposta feita pela PointFree para implementações com SwiftUI, UIKit e Combine/RXSwift.

Apple MVC



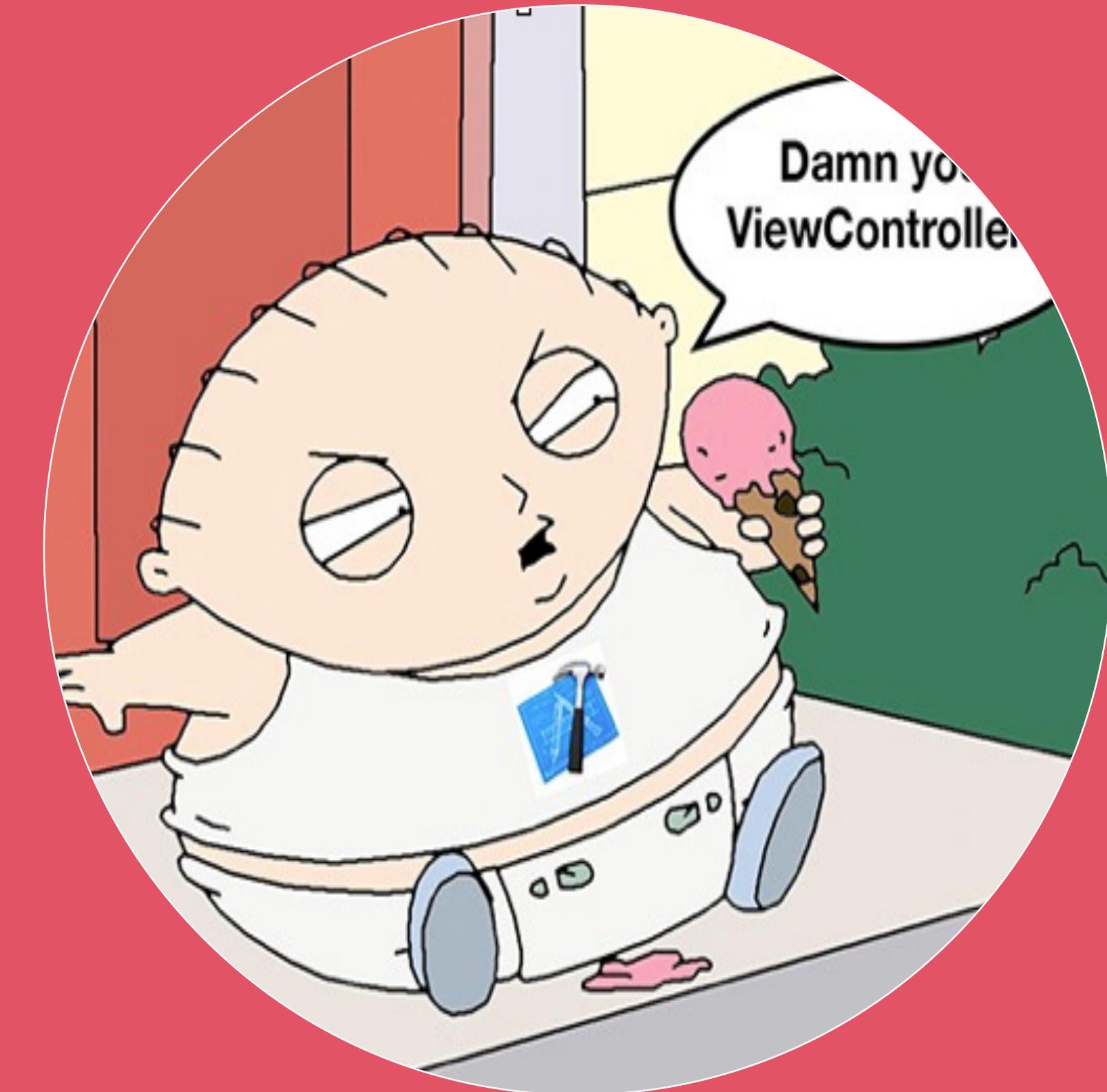
Apple MVC

```
1 import UIKit
2
3 struct Person { // Model
4     let firstName: String
5     let lastName: String
6 }
7
8 class GreetingViewController : UIViewController { // View + Controller
9     var person: Person!
10    let showGreetingButton = UIButton()
11    let greetingLabel = UILabel()
12
13    override func viewDidLoad() {
14        super.viewDidLoad()
15        self.showGreetingButton.addTarget(self, action: "didTapButton:", forControlEvents:
16    }
17
18    func didTapButton(button: UIButton) {
19        let greeting = "Hello" + " " + self.person.firstName + " " + self.person.lastName
20        self.greetingLabel.text = greeting
21    }
22
23    // layout code goes here
24 }
25 // Assembling of MVC
26 let model = Person(firstName: "David", lastName: "Blaine")
27 let view = GreetingViewController()
28 view.person = model;
```



[Exemplo de código MVC - by Wasin Thonkaew](#)

Apple MVC



Massive View Controllers

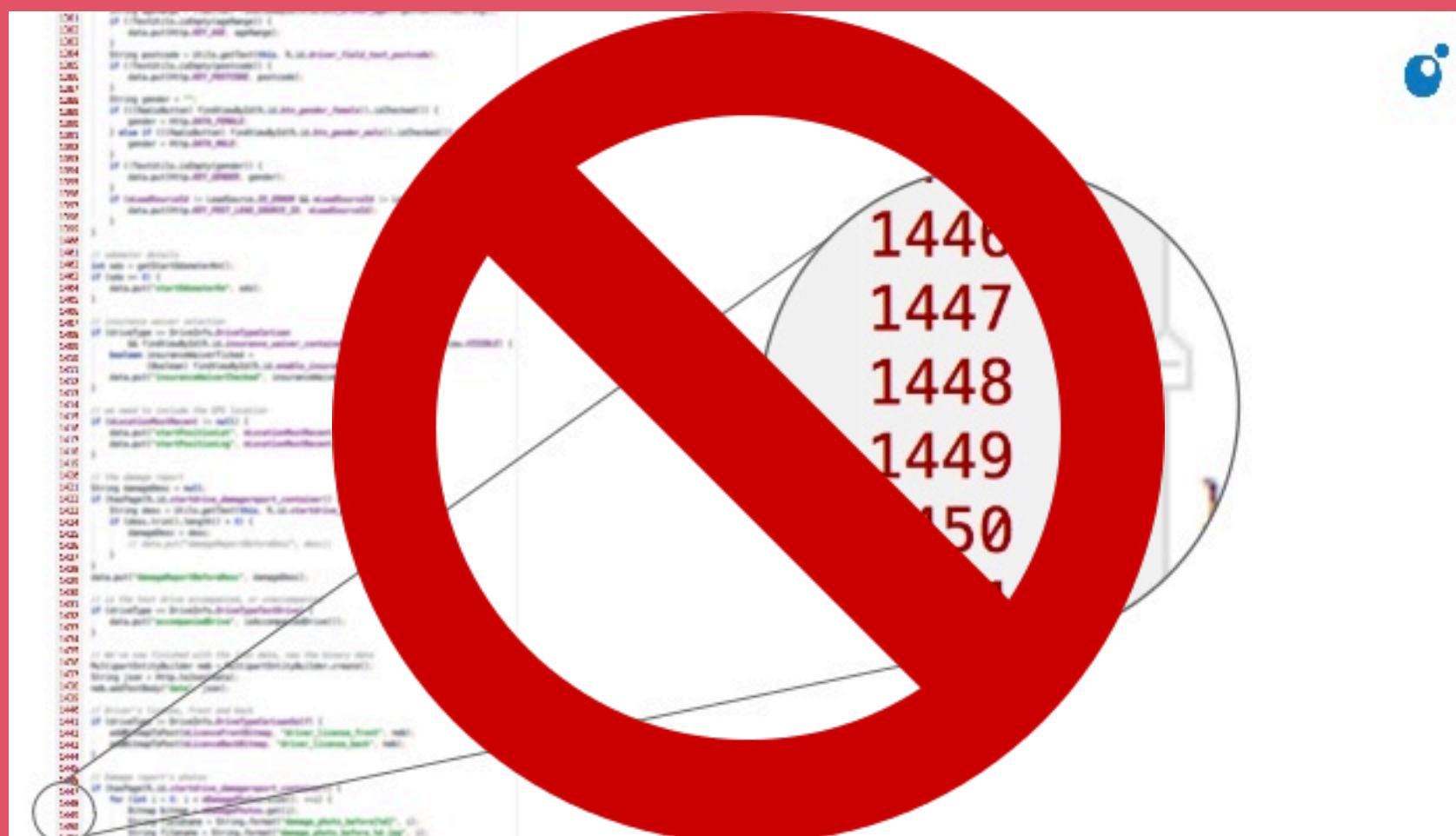
Problemas do Apple MVC

Muita lógica nas ViewControllers

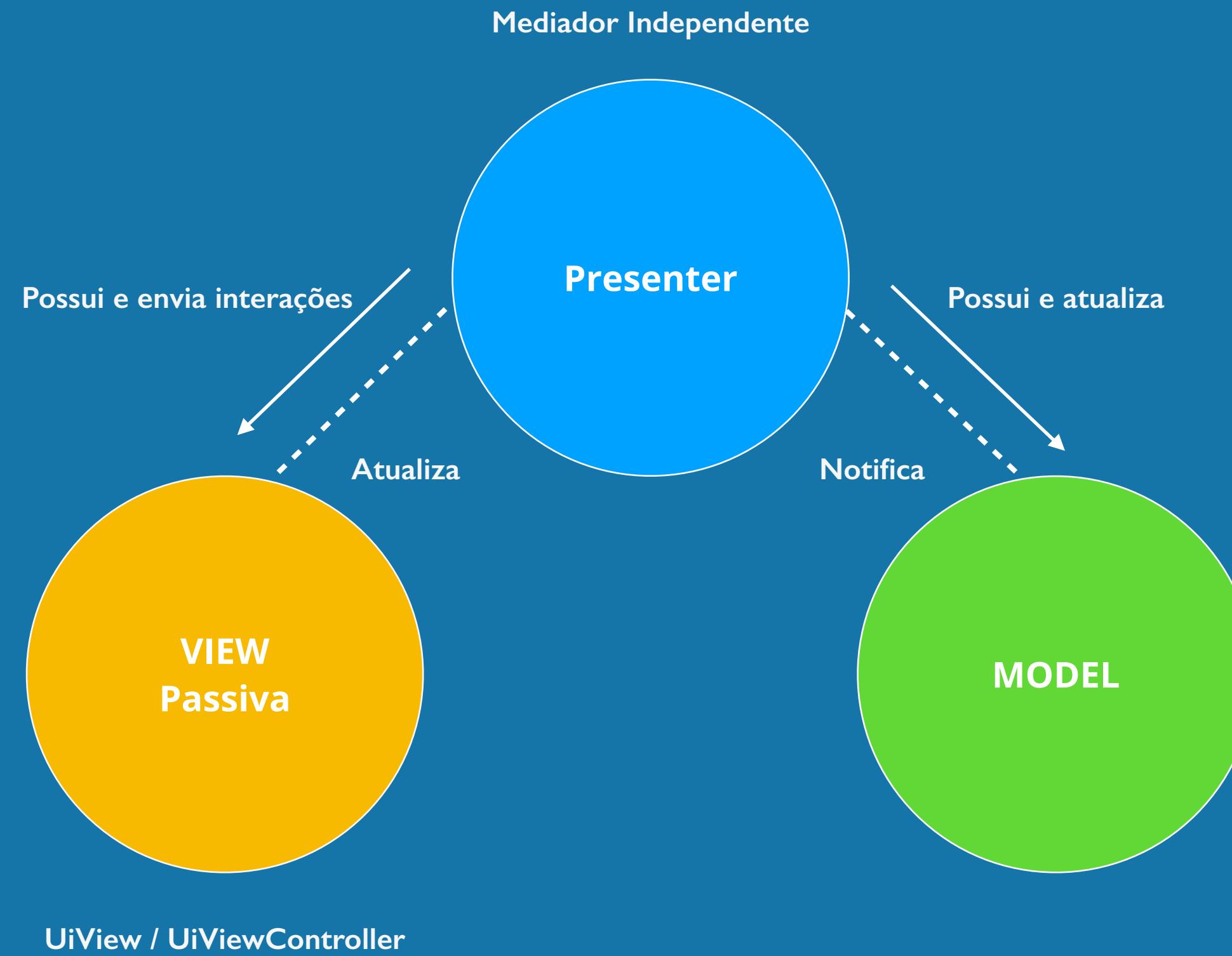
Manipulação da view + tratamento de interação da UI + tratamento das requests + tratamento da persistência de dados + lógica de formatação de dados

Péssima testabilidade

Leitura de código horrível



MVP



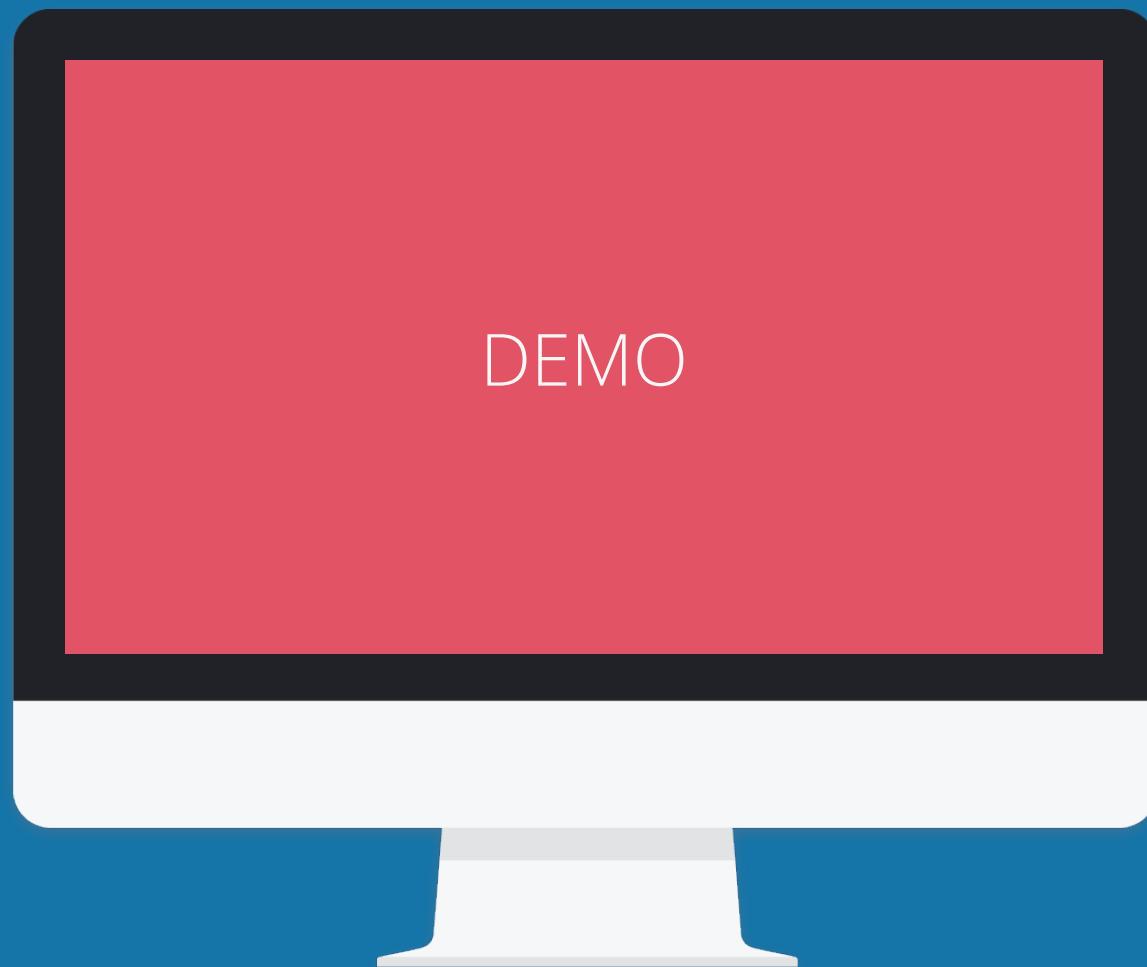
UÉ!? É igual ao MVC?

Parece mas não é ...

Diferente do MVC, onde existe uma única camada para View e Controller, no MVP, a View é uma camada independente (e burra).

Sendo assim, há uma separação completa entre os conceitos e responsabilidades (SoC).

MVP



Exemplo MVP

Você já ouviu falar da iniciativa Arquitetura Limpa?

Maintainable

Easy to refactor on volatile projects

Testable

Be confident that the code is reliable

Scalable

Can increase scope without requiring a rework

Readable

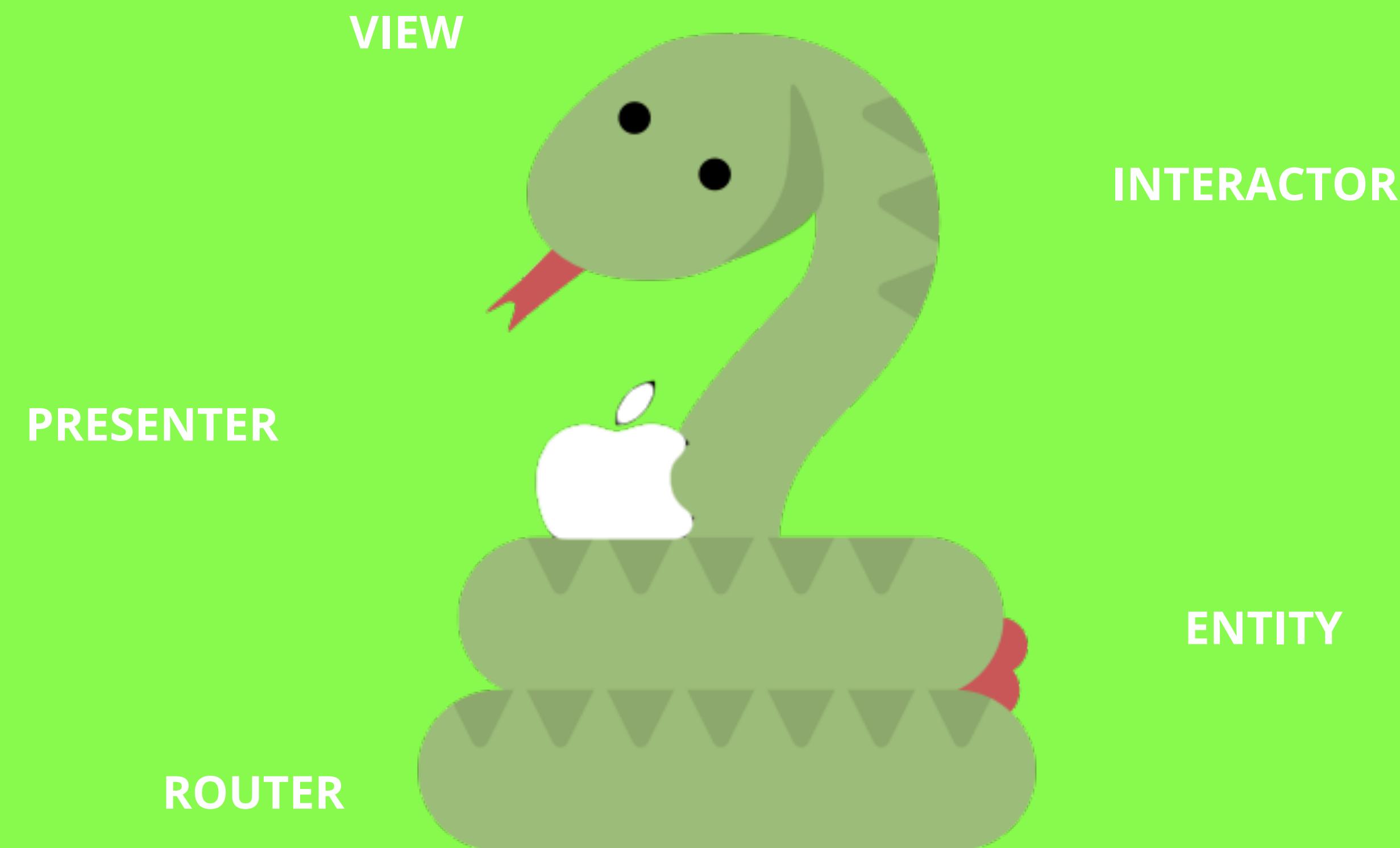
Reduce the difficulty of future development efforts

THERE WAS AN IDEA...

to bring together a group of remarkable **developer goals**



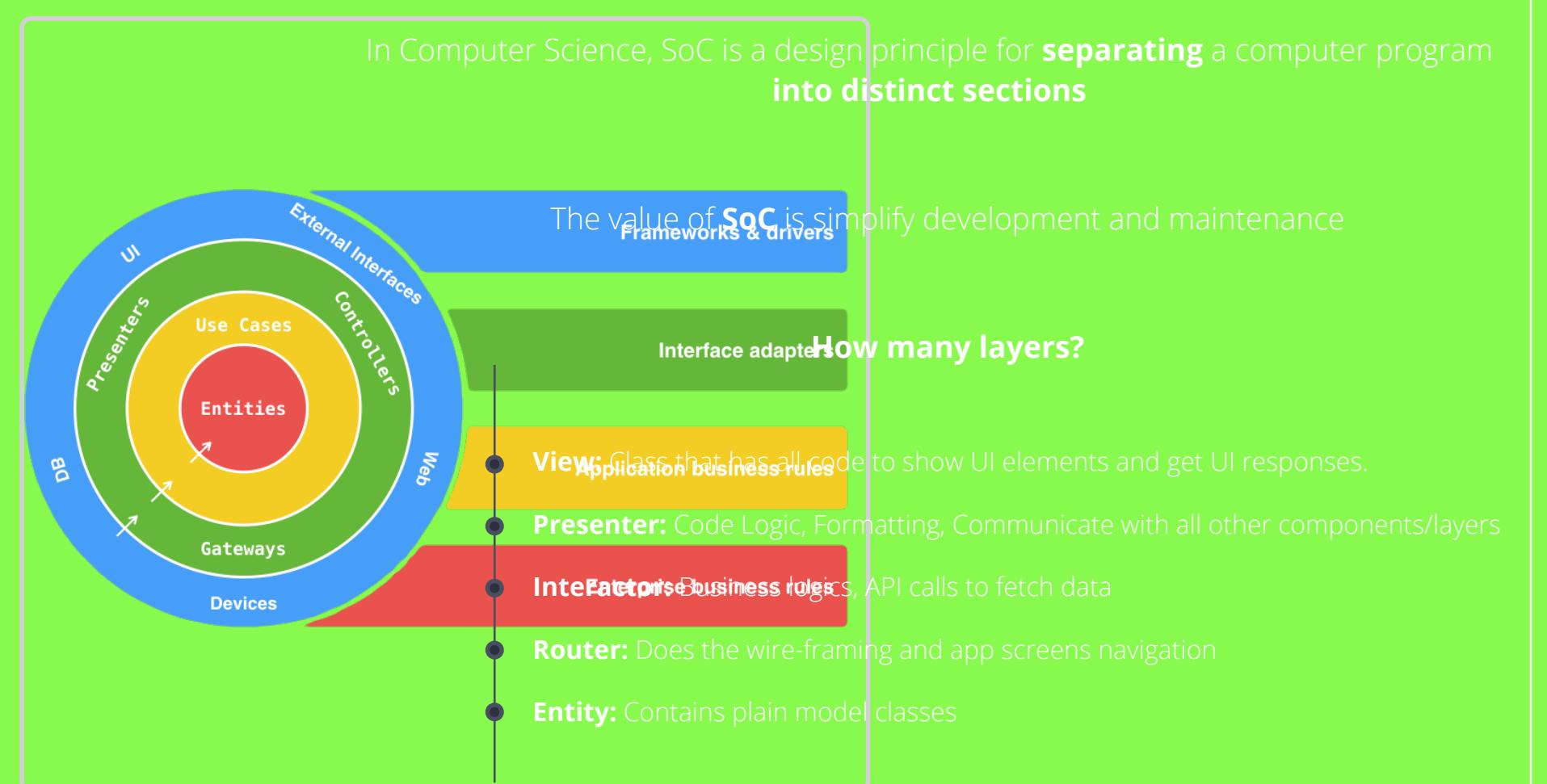
VIPER



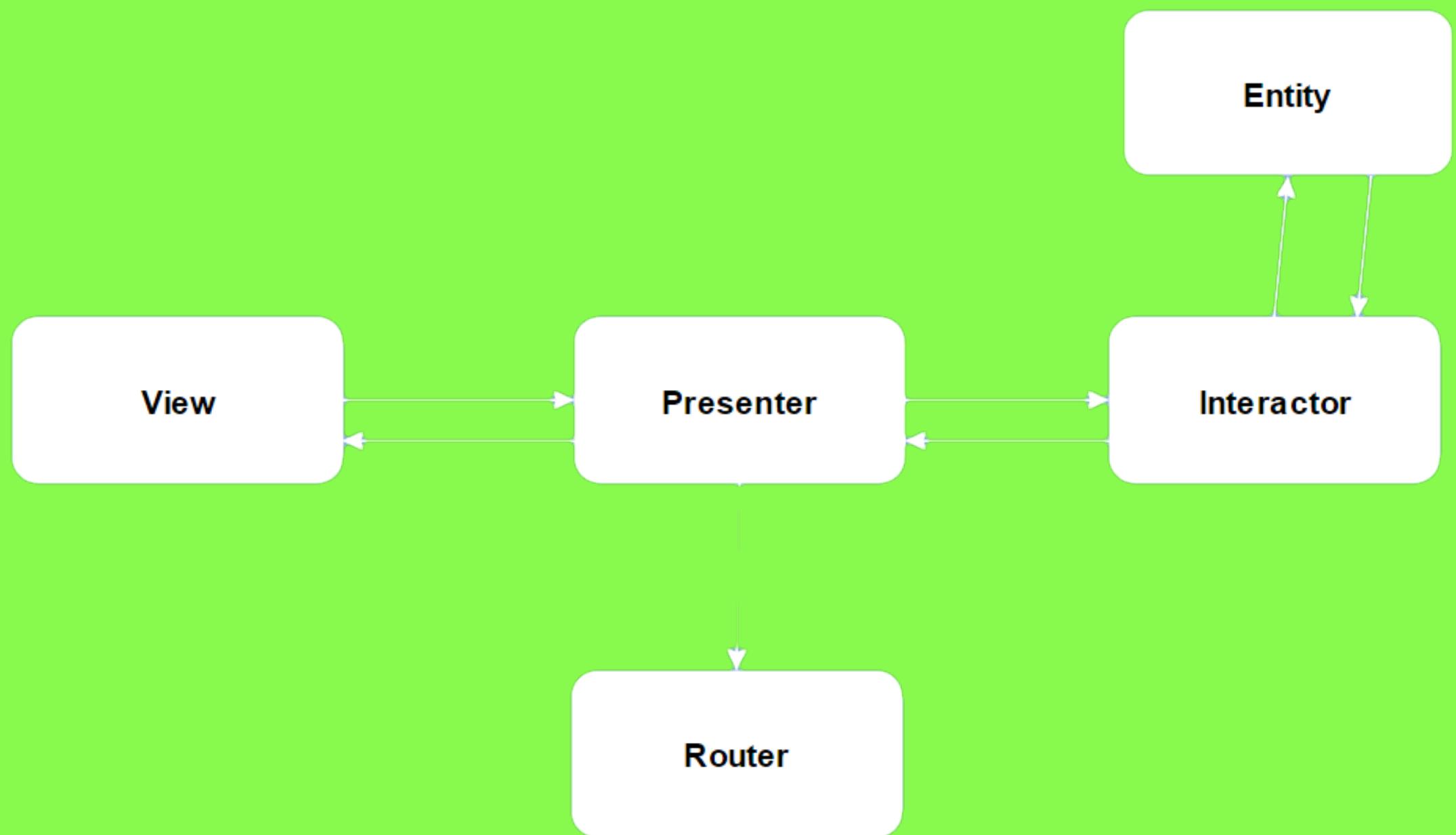
VIPER

SoC

Separation of Concern

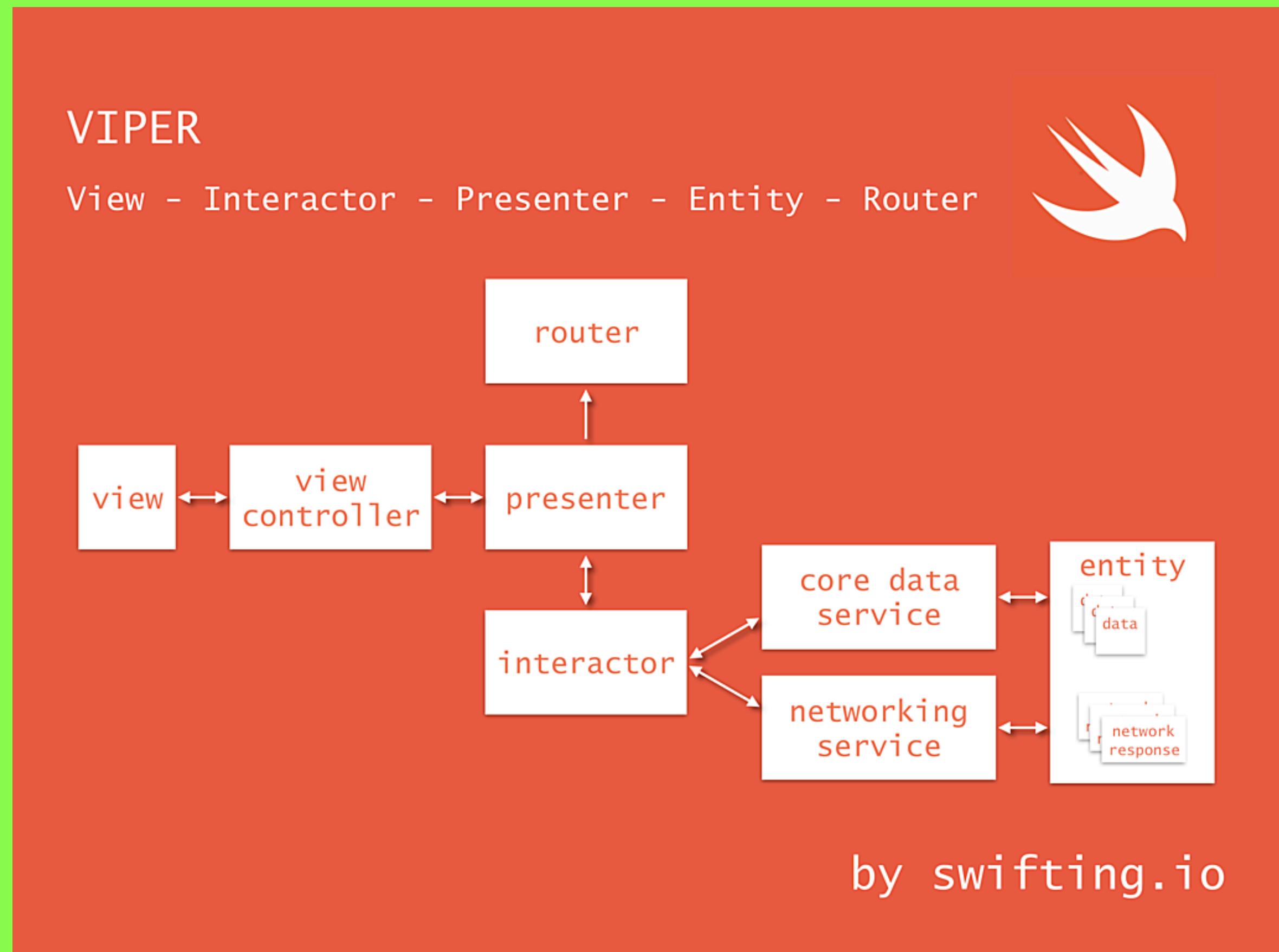


Viper Flow



VIPER

Viper Flow

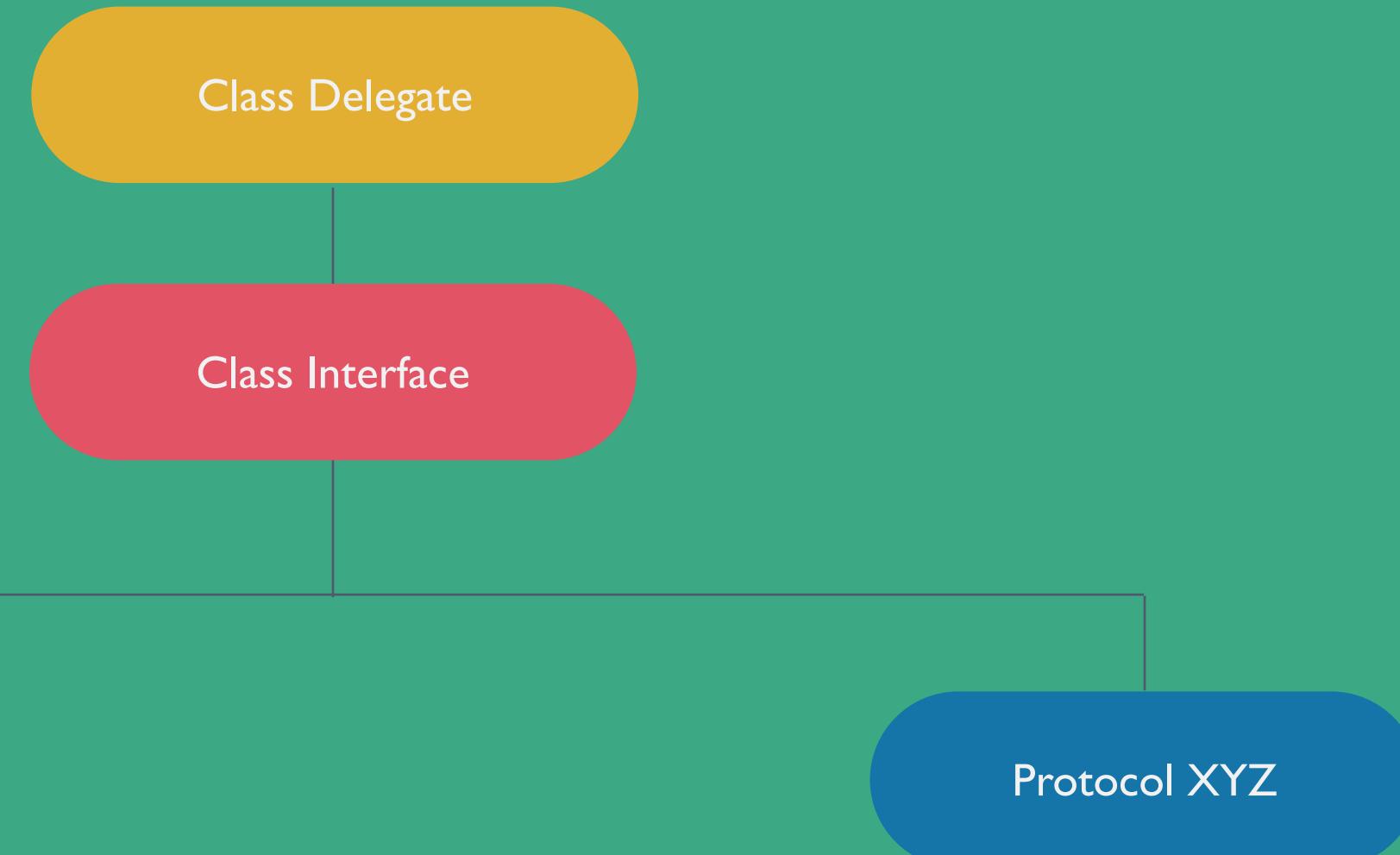


Referência: swifting.io

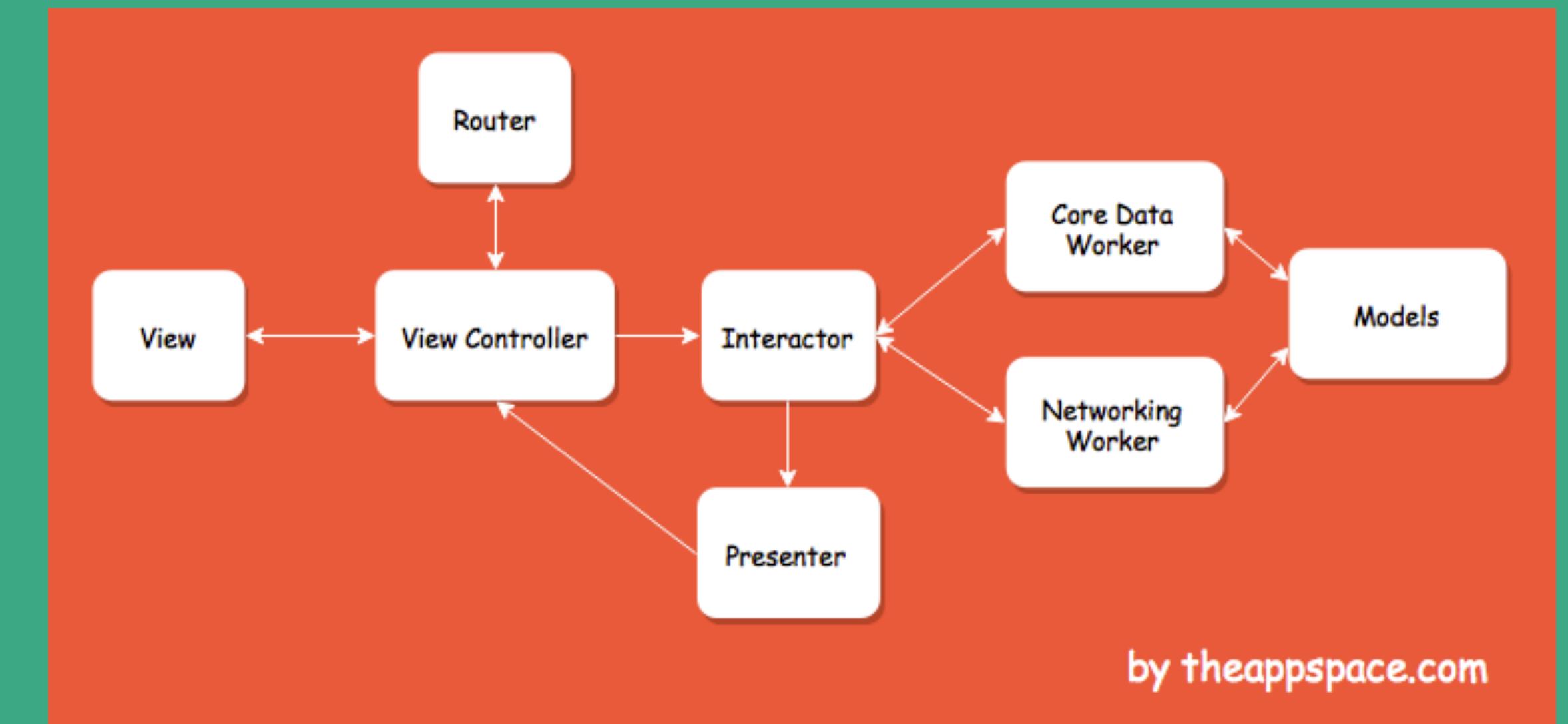
Clean Swift

Arquitetura proposta por Uncle Bob. Na verdade, trata-se de uma coleção de templates para gerar componentes semelhantes ao VIPER.

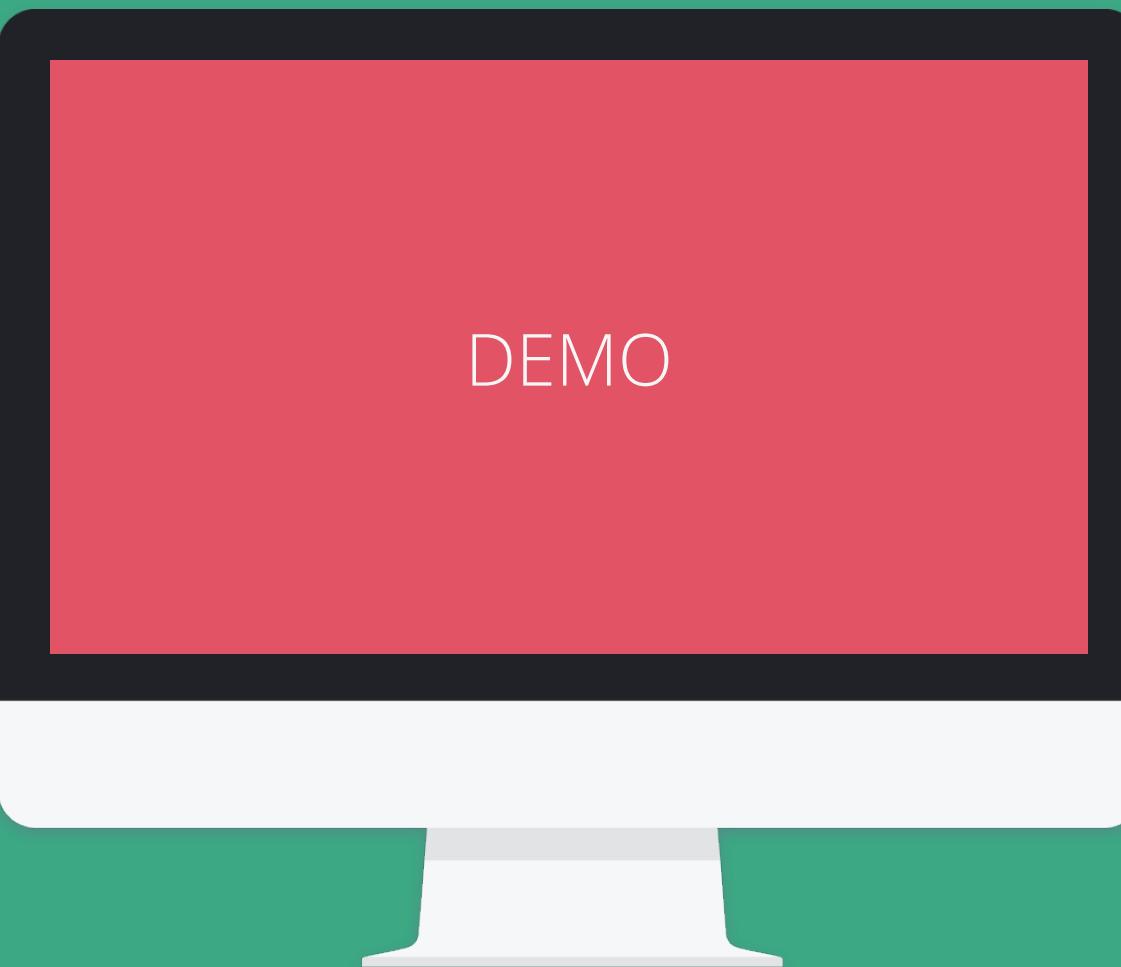
Protocols Everywhere



Comunicação entre componentes feita via Protocol-Oriented Programming (PoP)

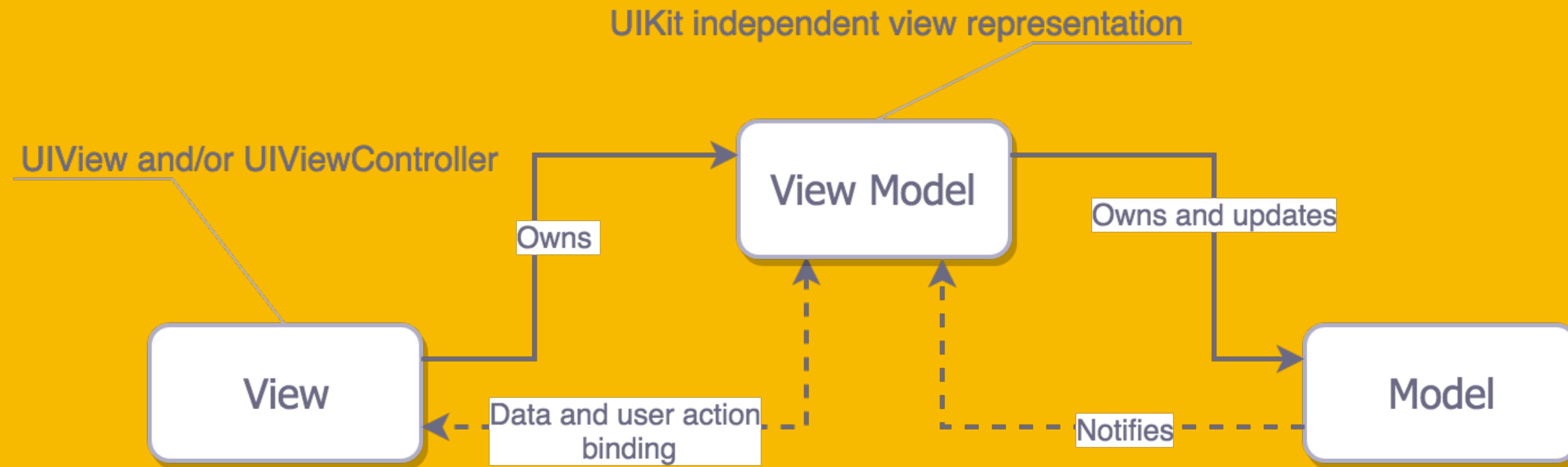


VIPER PROJECT



Exemplo VIPER

MVVM



Semelhante ao MVP, a **ViewController** é tratada como **VIEW** (passiva).
Porém, ao invés do data binding ocorrer entre a **View** e o **Model**, ocorre entre a **View** e o **ViewModel**.

COMBINE (> iOS 13)

O toolbox do iOS não possui um Data Binding nativo.
Sim KVO/Notificações existem porém não são muito convenientes de usar.

Assim, grande parte da comunidade adotou essas opções:

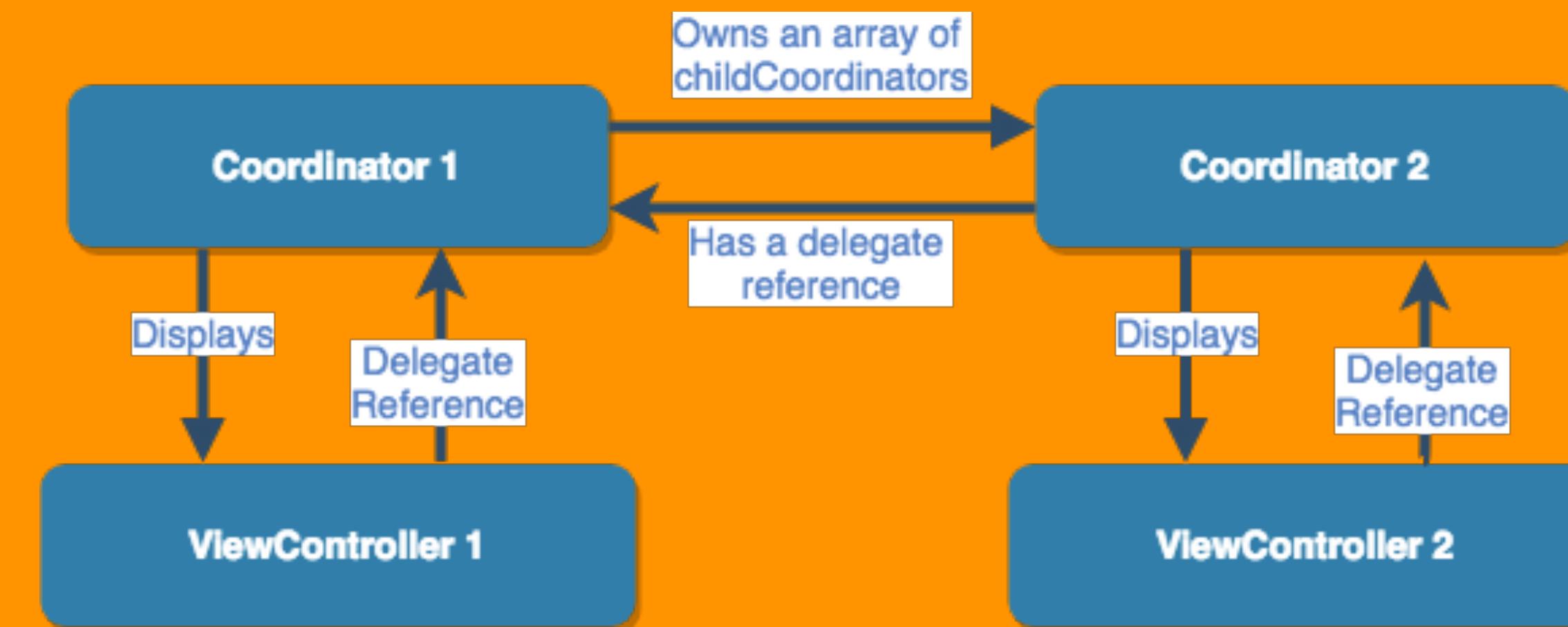
- RXSwift
- PromiseKit
- ReactiveCocoa

Paradigma de Programação Reativa Funcional

O padrão do coordenador fornece um encapsulamento da lógica de navegação.

O padrão do coordenador fornece um encapsulamento da lógica de navegação.

Em vez de enviar e apresentar seus ViewControllers a parte de outro controlador, toda navegação da tela é gerenciada por coordenadores. Sendo assim, as ViewControllers ficam isoladas e invisíveis entre elas e facilita o reuso.



Regras

- Uma ou mais ViewControllers por coordinator
- Normalmente, cada coordinator apresenta a ViewController através de um método Start()
- Cada ViewController possui uma referência delegate para seu coordinator
- Pode haver uma lista ChildCoordinators que possuem delegates para seu ParentCoordinator

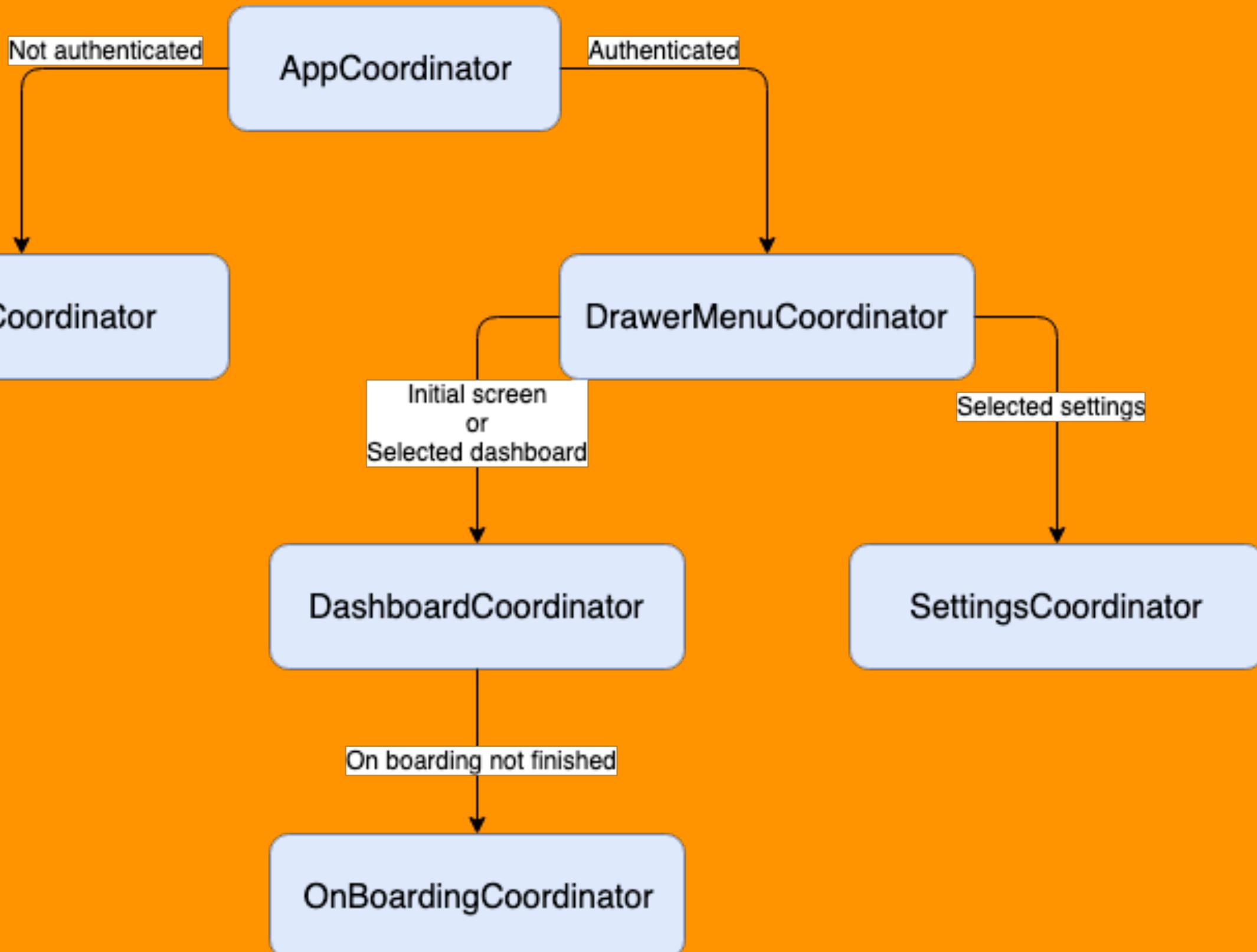
```
class SomeCoordinator {  
    var navigationController = UINavigationController()  
  
    func start() {  
        // TODO: start flow  
    }  
}
```

MVVM-C



```
▼ └── InvestmentsHome
    └── Entity
        ├── InvestmentsHomeViewEntity.swift
        └── TabViewControllerEntity.swift
    └── View
        ├── InvestmentsHomeView.swift
        ├── InvestmentsHomeViewState.swift
        ├── InvestmentsHomeViewType.swift
        ├── InvestmentsHomeProtocols.swift
        └── InvestmentsHomeViewController.swift
    └── ViewModel
        └── InvestmentsHomeViewModel.swift
```

Coordinators



Referência: Wojciech Kulik

COORDINATORS



HOW TO USE THE COORDINATOR PATTERN IN iOS

Referência: [Advanced coordinators in iOS](#)

The Composable Architecture

TCA

Criada por [Stephen Celis](#) e [Brandon Williams](#) do site [PointFree](#) baseada em pilares já existentes e pode ser usada com UIKit ou com SwiftUI (iOS > 5.1).

- Dependência do Framework Combine (iOS 13). Existem Forks que permitem o uso de ReactiveSwift e RXSwift.



The Composable Architecture

Pilares da TCA

Gerenciamento de estado

Uso de máquina de estados usando tipos de valor para compartilhar estados entre várias telas permitindo que mutações sejam facilmente observadas.

Testes

Capacidade de escrever testes de integração de recursos que foram compostos por várias partes e escrever testes ponta a ponta afim de entender os efeitos colaterais que regem sua aplicação. Isso vai te proporcionar que tua lógica de negócios esteja da maneira esperada.

Composição

Capacidade de componentizar recursos grandes em menores, possibilitando o isolamento em módulos para que possam facilmente ser montados afim de formar novamente a feature.

Ergonomia

Capacidade de realizar os itens citados acima em uma API bem simples e com o mínimo de conceitos e partes mais maleáveis possível.

Efeitos colaterais

Capacidade de permitir que algumas partes da aplicação possam conversar com o mundo externo de forma mais testável e comprehensível possível.

Referência: [PointFree Github](#)

Referência: [PointFree Website](#)

Referência: [@alcidesjuniorbr](#)

The Composable Architecture

Estrutura da TCA

State

Tipo que descreve os dados que sua feature precisa para executar sua lógica e renderizar a interface de usuário.

Action

Tipo que representa todas as ações que podem acontecer na sua feature que vão desde ações do usuário como transitar de uma tela para outra enviando determinada informação até notificações, fontes de eventos etc.

Environment

Tipo que mantém todas as dependências que a feature precisa, como clientes de API, clientes de análise etc.

Reducer

Função que descreve como o estado atual da aplicação irá evoluir para outro estado a partir de uma determinada ação. O papel do Reducer também é ser responsável por retornar quaisquer efeitos que devem ser executados como solicitações de API que podem ser feitos retornando um valor do *Efeito*.

Store

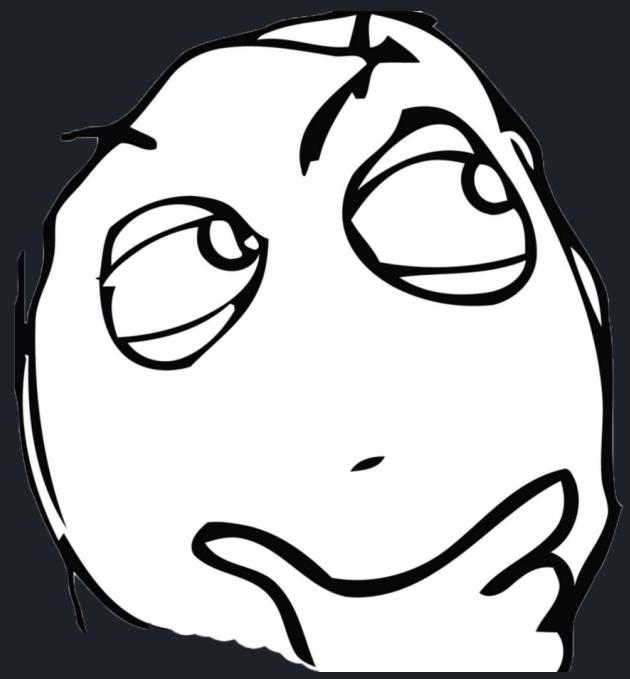
Aqui você envia todas as ações do usuário para a *Store* para que a *Store* possa executar o *Reducer* e os efeitos e pode observar alterações de estado na *Store* para poder atualizar a interface de usuário.

Referência: [PointFree Github](#)

Referência: [PointFree Website](#)

Referência: [@alcidesjuniorbr](#)

Dúvidas



Guia: [Awesome iOS Architecture](#)



Obrigado!

Para em contatar, siga-me aqui:

 tonis.04@gmail.com

 linkedin.com/in/acalbuquerque/

 [@acalbuquerque](https://twitter.com/acalbuquerque)