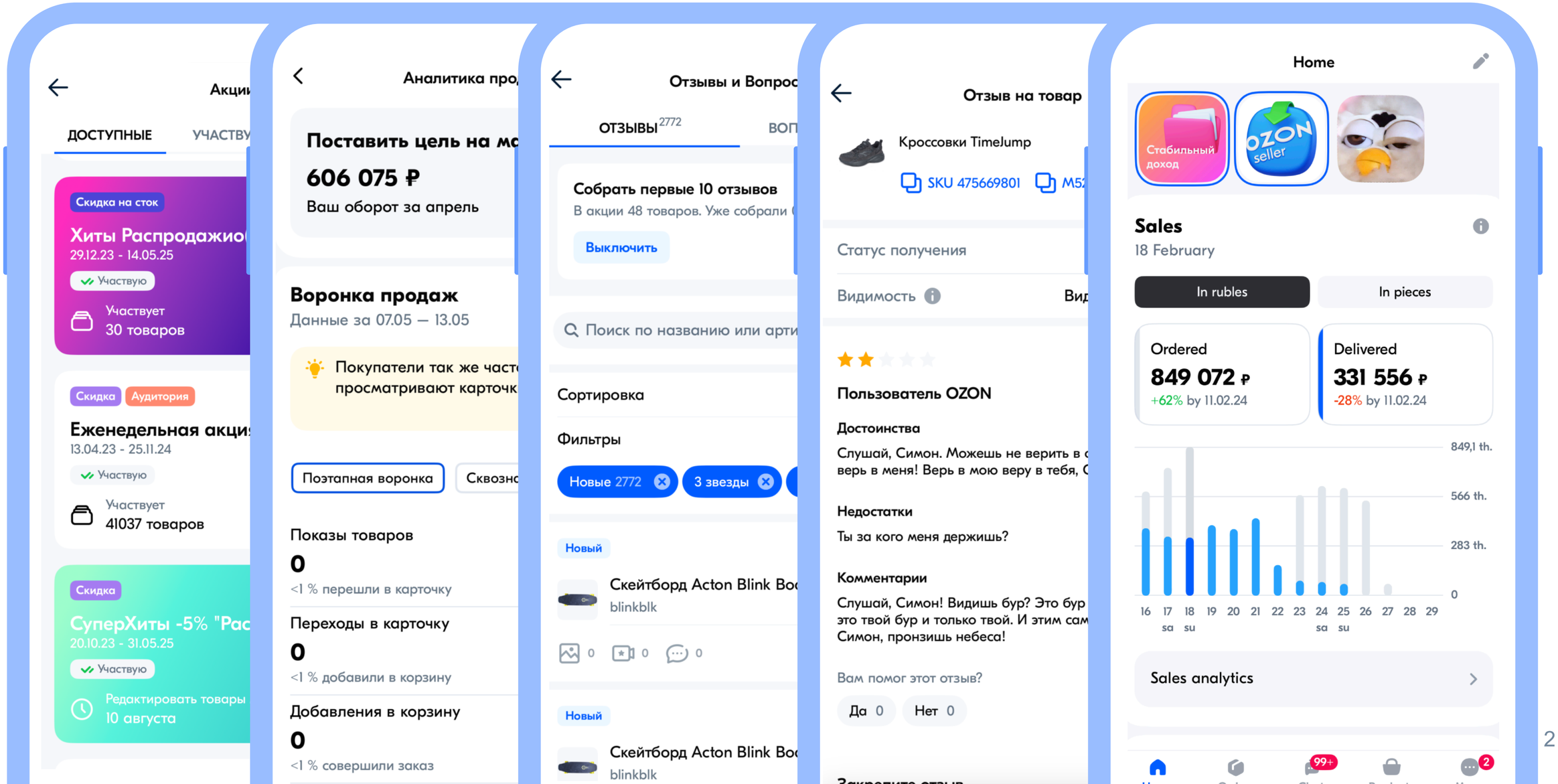
OZONtech

Ускоряем SwiftUI с Observable

Чернов Дмитрий
Руководитель группы в OzonTech

Немного о себе



AGENDA

01 **Что такое Observable object**

02 **Проблемы Observable object**

03 **Знакомство с @Observable**

04 **Результаты использования @Observable**

Паттерн «наблюдатель»

Паблишер

Сабскрайберы

Наблюдатель в SwiftUI и Combine

```
struct OzonProductsView: View {
    @ObservedObject
    var viewModel: OzonProductsViewModel

    var body: some View {
        VStack {
            List(viewModel.productList) { product in
                HStack {
                    nameLabel(name: product.name)
. . .
```

```
final class OzonProductsViewModel: ObservableObject {
    @Published
     var productList: [Product]
    @Published
     var image = ImageResource(name: "pepe", bundle: .main)

    init(productList: [Product] = []) {
        self.productList = productList
    }

    func addMorePoducts() {
        let newOzonProduct = Product(
                                name: .getRandomName(),
                                price: .getRandomPrice())
        productList.append(newOzonProduct)
    }
. . .
```

Наблюдатель в SwiftUI и Combine

```
struct OzonProductsView: View {
    @ObservedObject
    var viewModel: OzonProductsViewModel

    var body: some View {
        VStack {
            List(viewModel.productList) { product in
                HStack {
                    nameLabel(name: product.name)
. . .
```

```
final class OzonProductsViewModel: ObservableObject {
    @Published
     var productList: [Product]
    @Published
     var image = ImageResource(name: "pepe", bundle: .main)

    init(productList: [Product] = []) {
        self.productList = productList
    }

    func addMorePoducts() {
        let newOzonProduct = Product(
                                name: .getRandomName(),
                                price: .getRandomPrice())
        productList.append(newOzonProduct)
    }
. . .
```

AGENDA

01 Что такое Observable object

02 Проблемы Observable object

03 Знакомство с @Observable

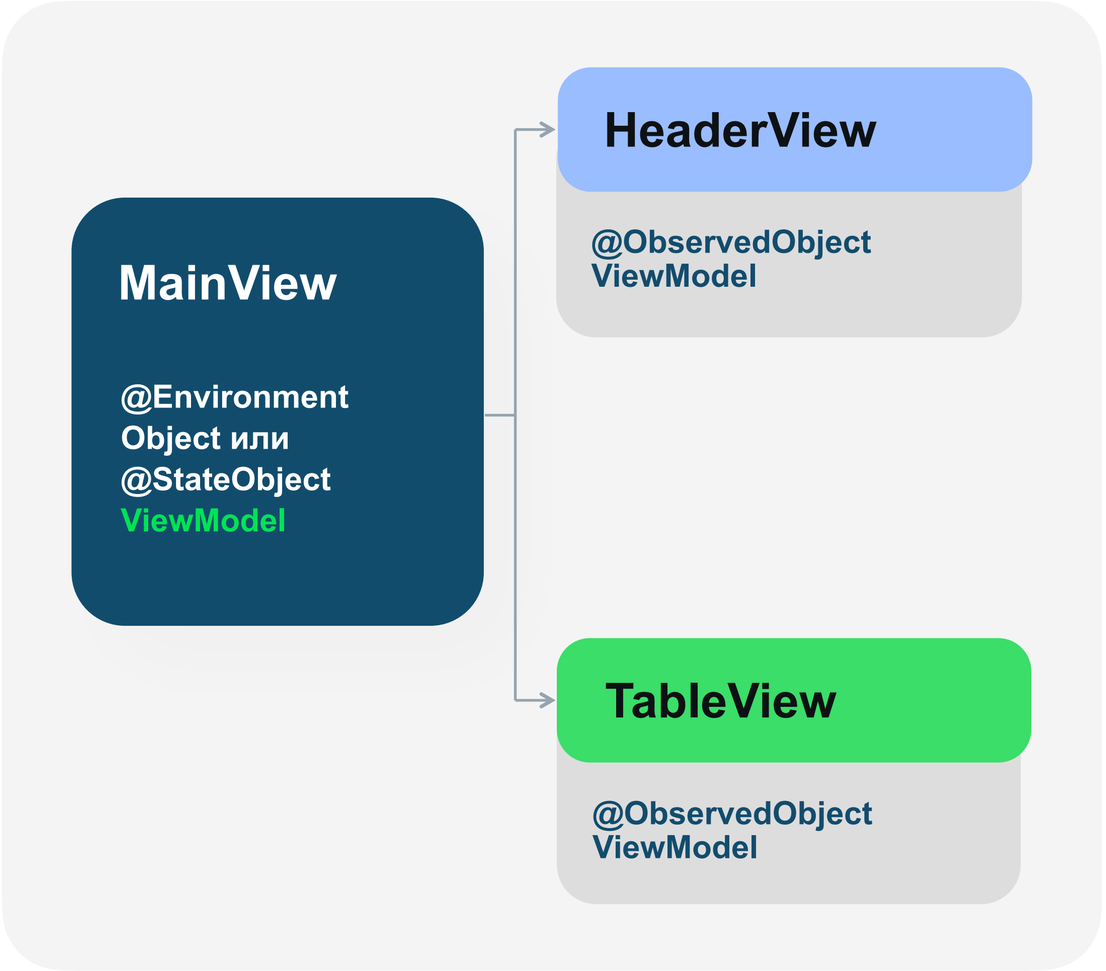
04 Результаты использования @Observable

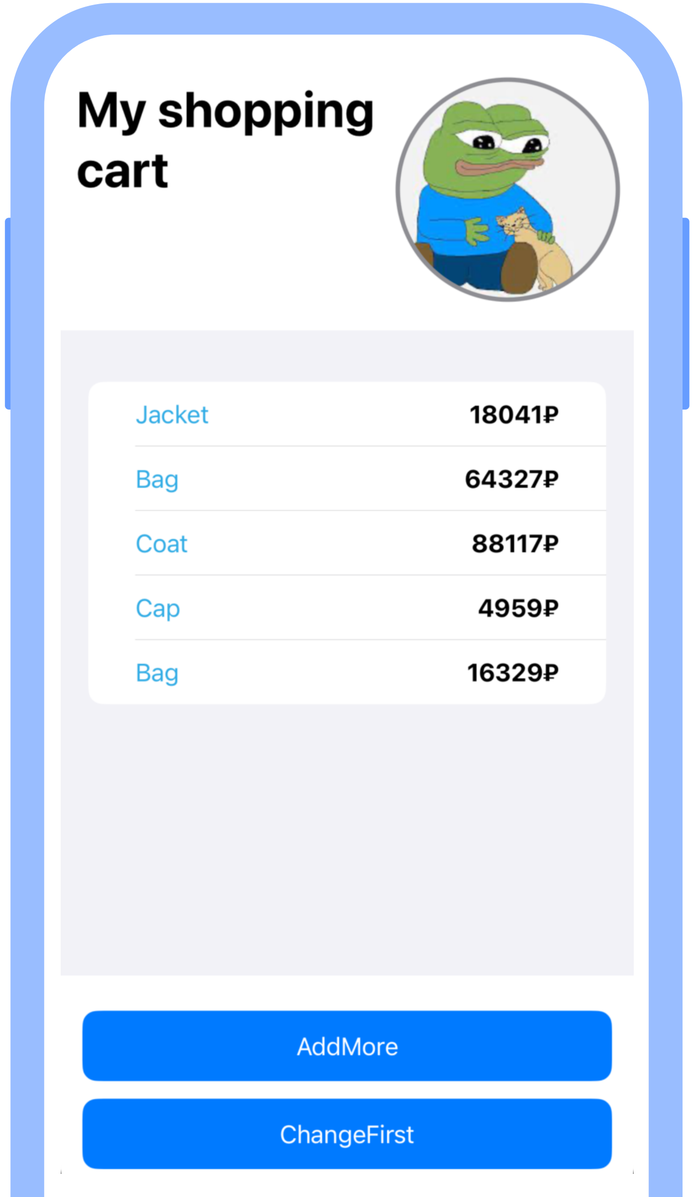
ozontech

И тут начались проблемы

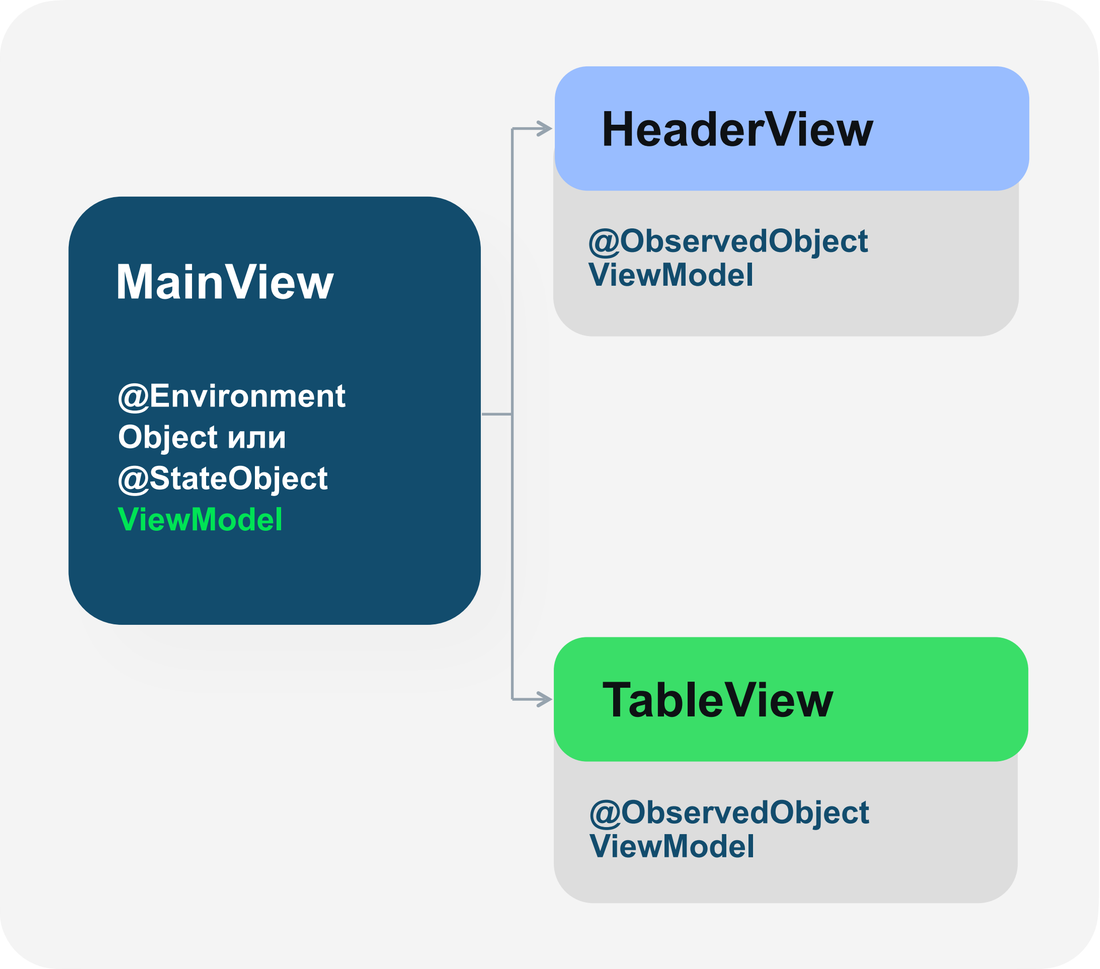
- **Проблема частого перерасчета body** 
- **Тесная интеграция Combine**
- **Проблема вложенных объектов**
- **Проблемы с протоколами**
- **Проблема наблюдений за коллекциями**

Проблема частого перерасчета body

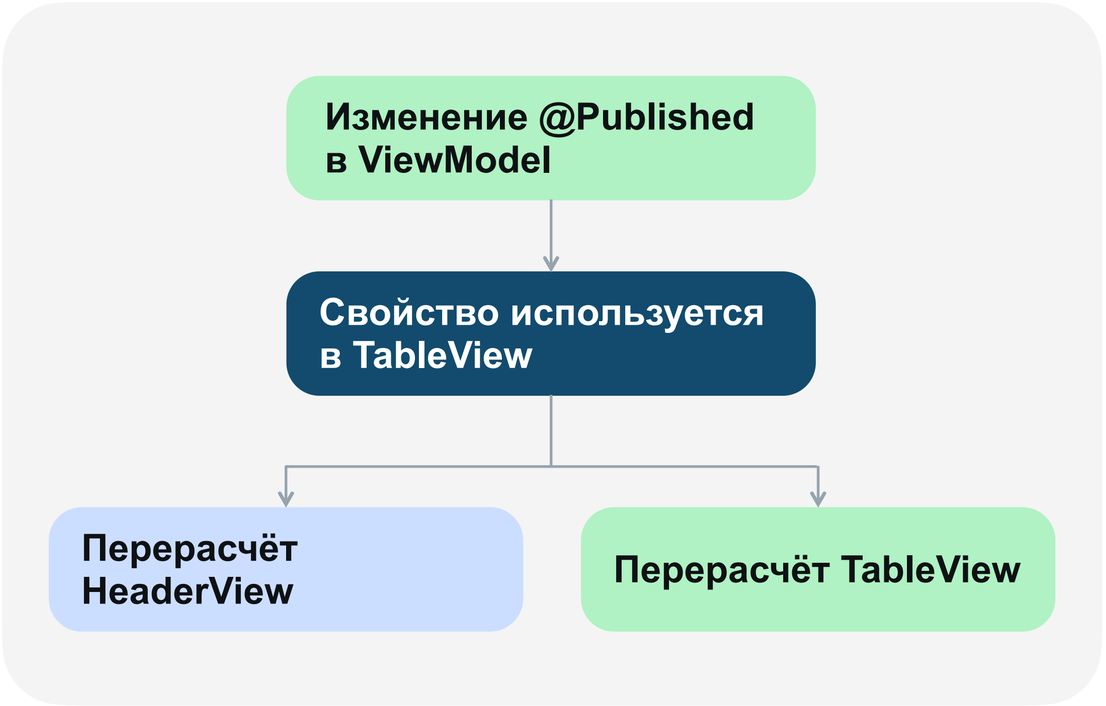




Проблема частого перерасчета body



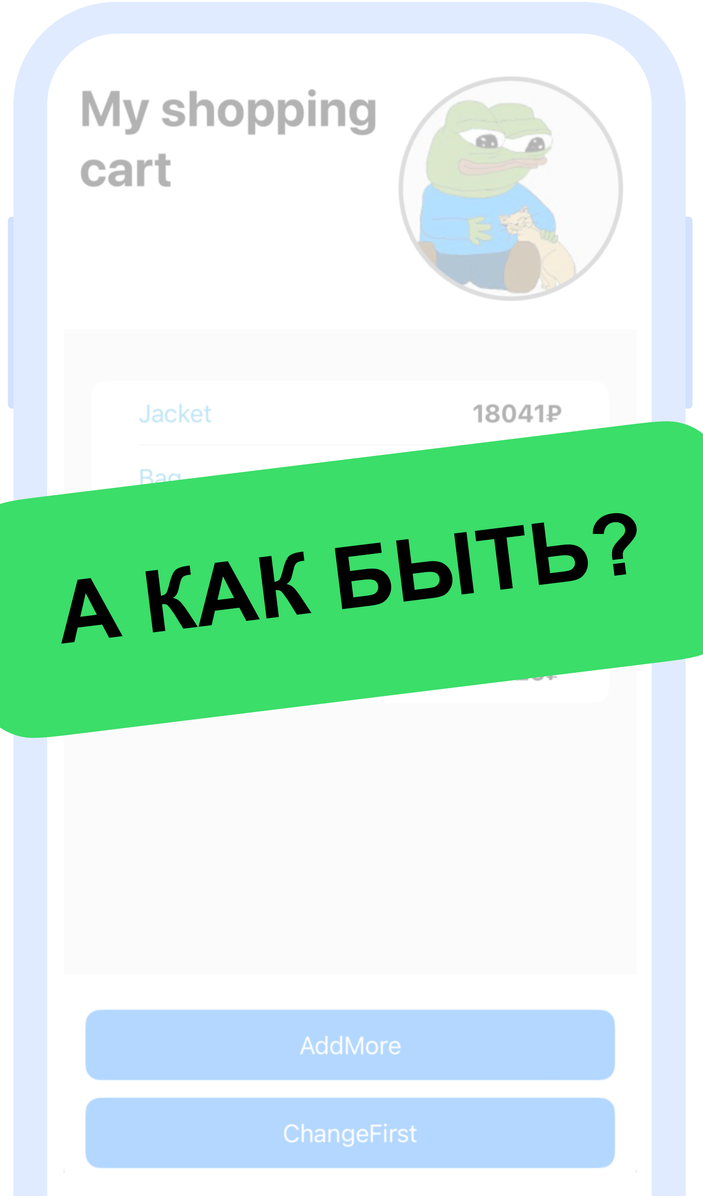
Меняем ячейку таблицы



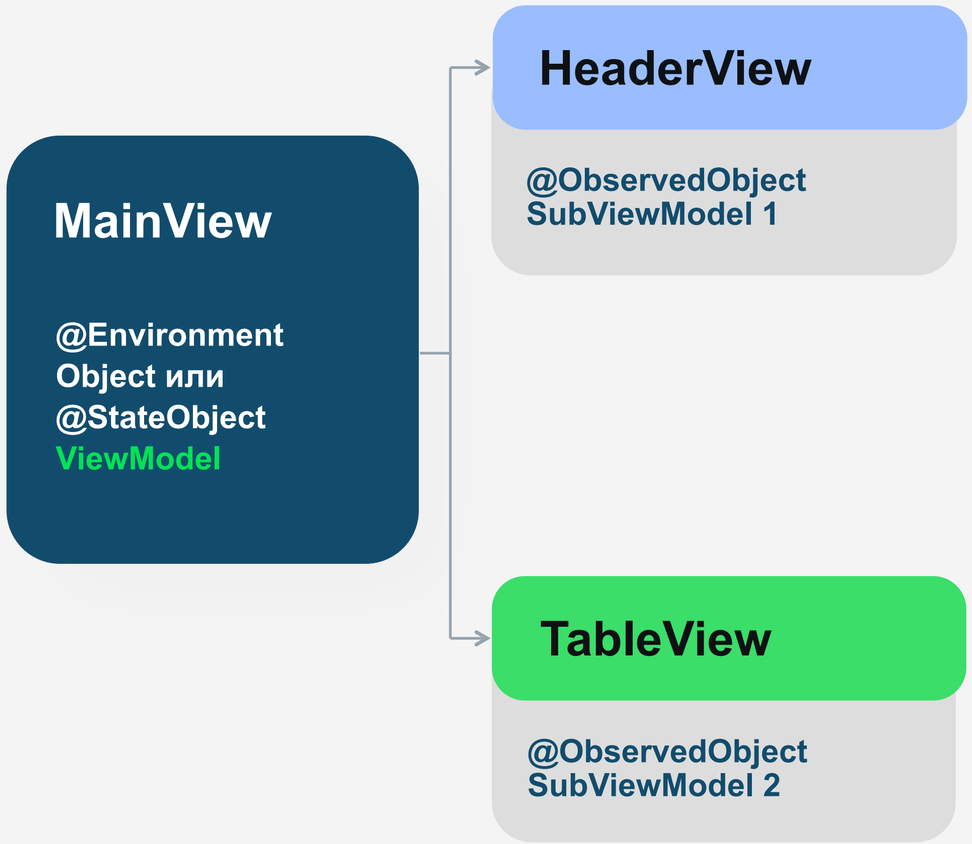
Проблема частого перерасчета body

```
OzonImage: _ozonProductsViewModel changed.
OzonProductsView: _viewModel changed.
```

Filter

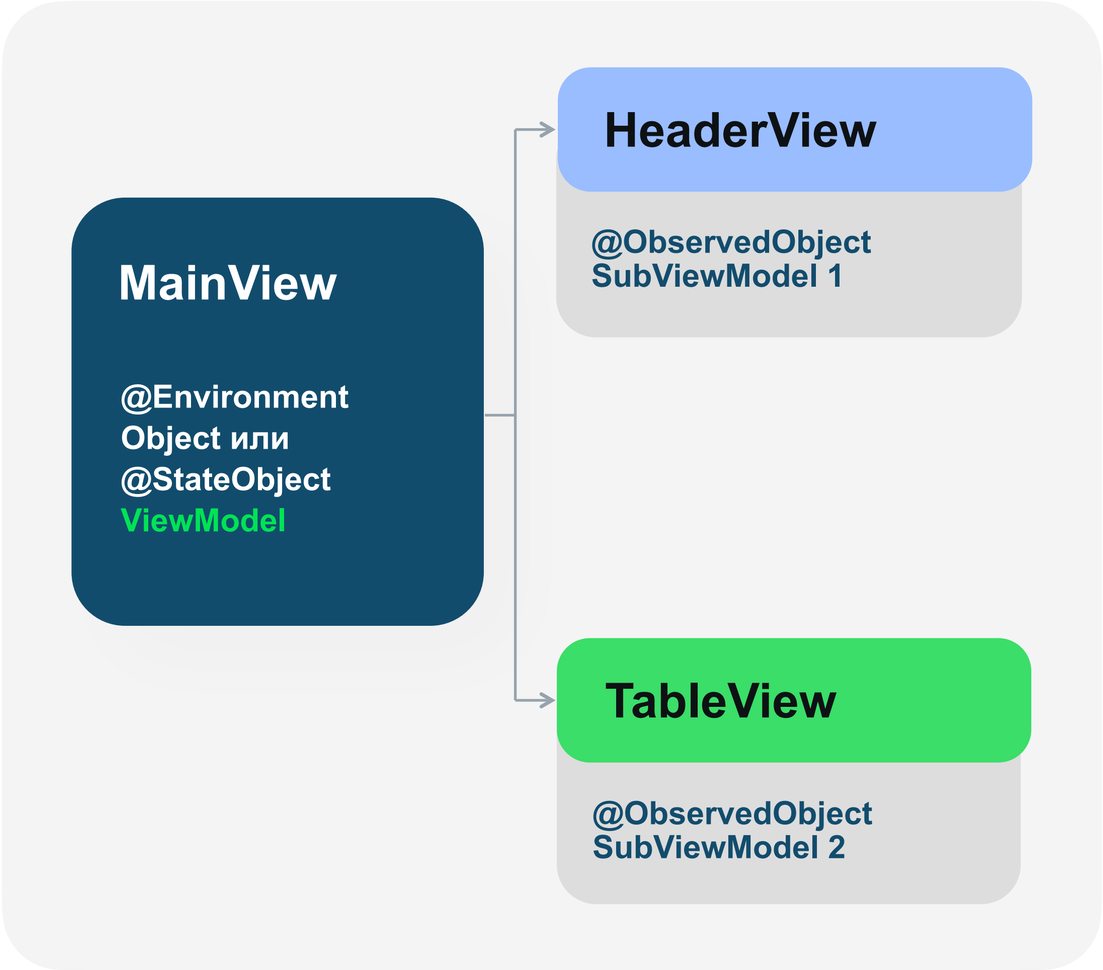


Проблема частого перерасчета body

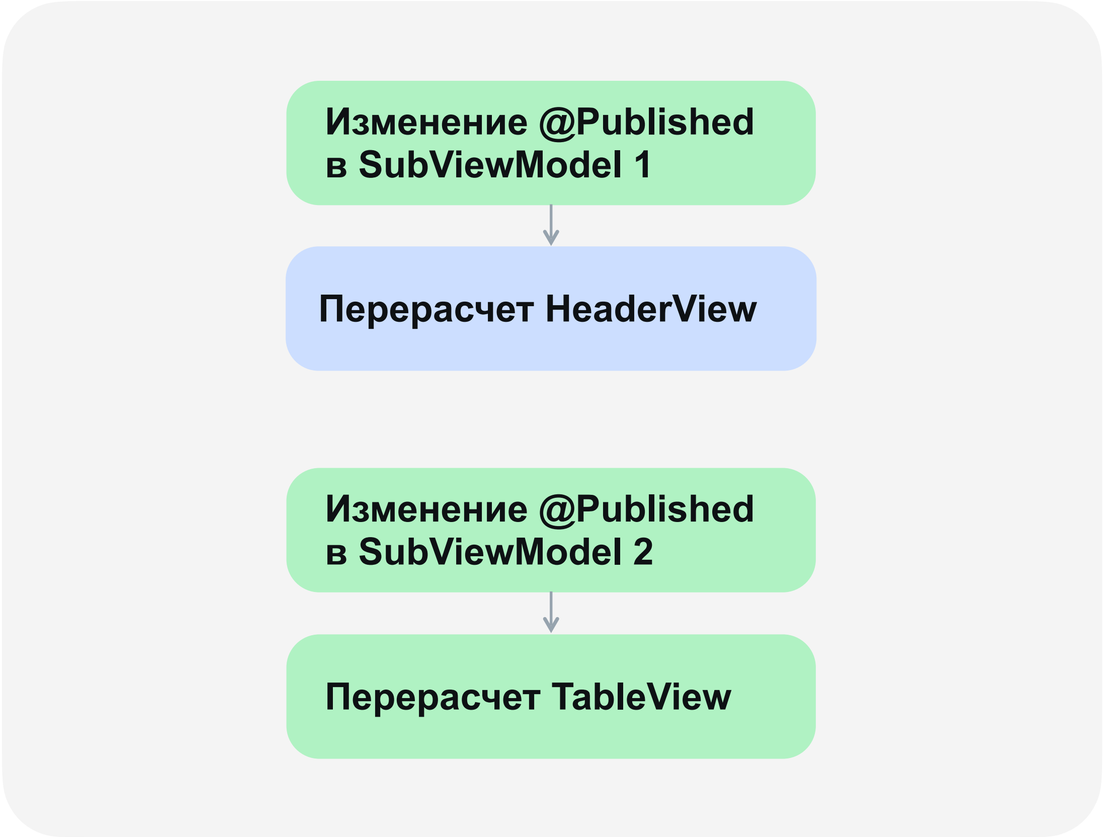



```
final class OzonProductsViewModel: ObservableObject {

    var headerViewModel = SubViewModel1()

    var productsViewModel = SubViewModel2()

  func addMorePoducts() {
    let newOzonProduct = Product(
                          name: .getRandomName(),
                          price: .getRandomPrice())
    productsViewModel.add(newOzonProduct)
  }
...
```

Проблема частого перерасчета body



Меняем ячейку таблицы



Проблема частого перерасчета body

Нужны механизмы для отслеживания и предотвращения лишних перерасчётов View

Метрики

Код-ревью

И тут начались проблемы

- Проблема частого перерасчета body
- Тесная интеграция Combine
- Проблема вложенных объектов 
- Проблемы с протоколами
- Проблема наблюдений за коллекциями

Проблема частого перерасчета body

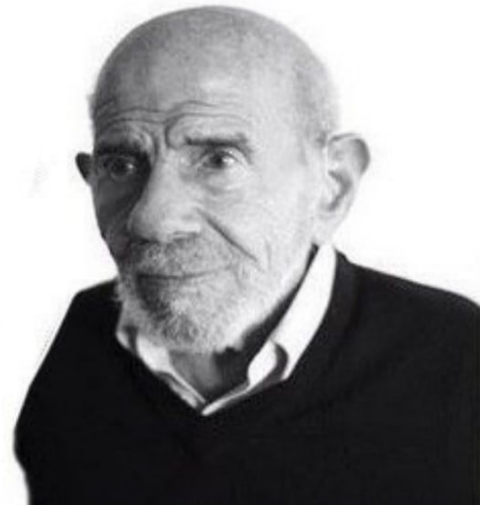
«Почему моя View не обновляется?»

«Почему SwiftUI не реагирует на изменения @StateObject или @ObservableObject?»

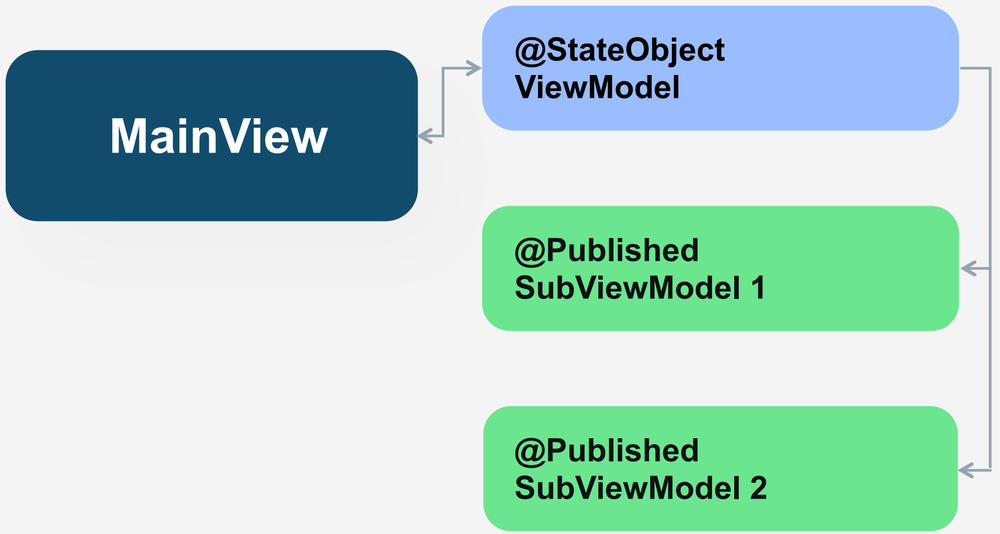
«ObservableObject не триггерит обновления View, что это такое?»

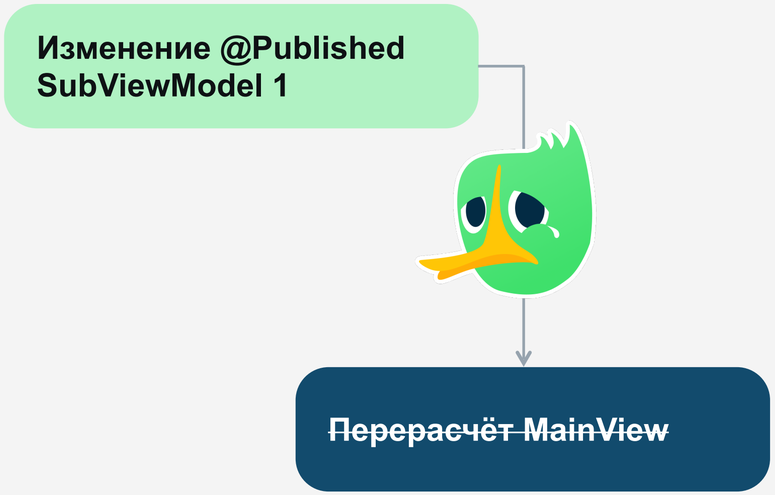
ладно

— Жак Фреско —



Проблема вложенных объектов



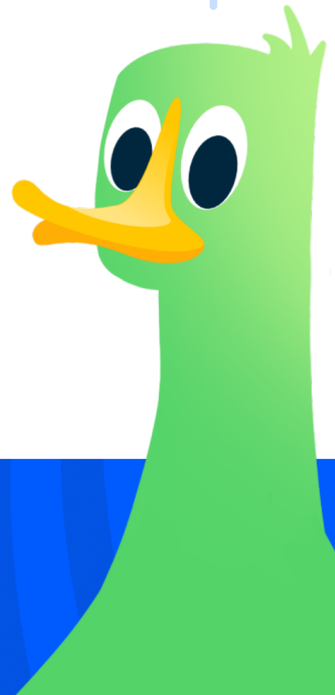


А ДЕЛО В ТОМ…

Проблема частого перерасчёта body

ObservableObject

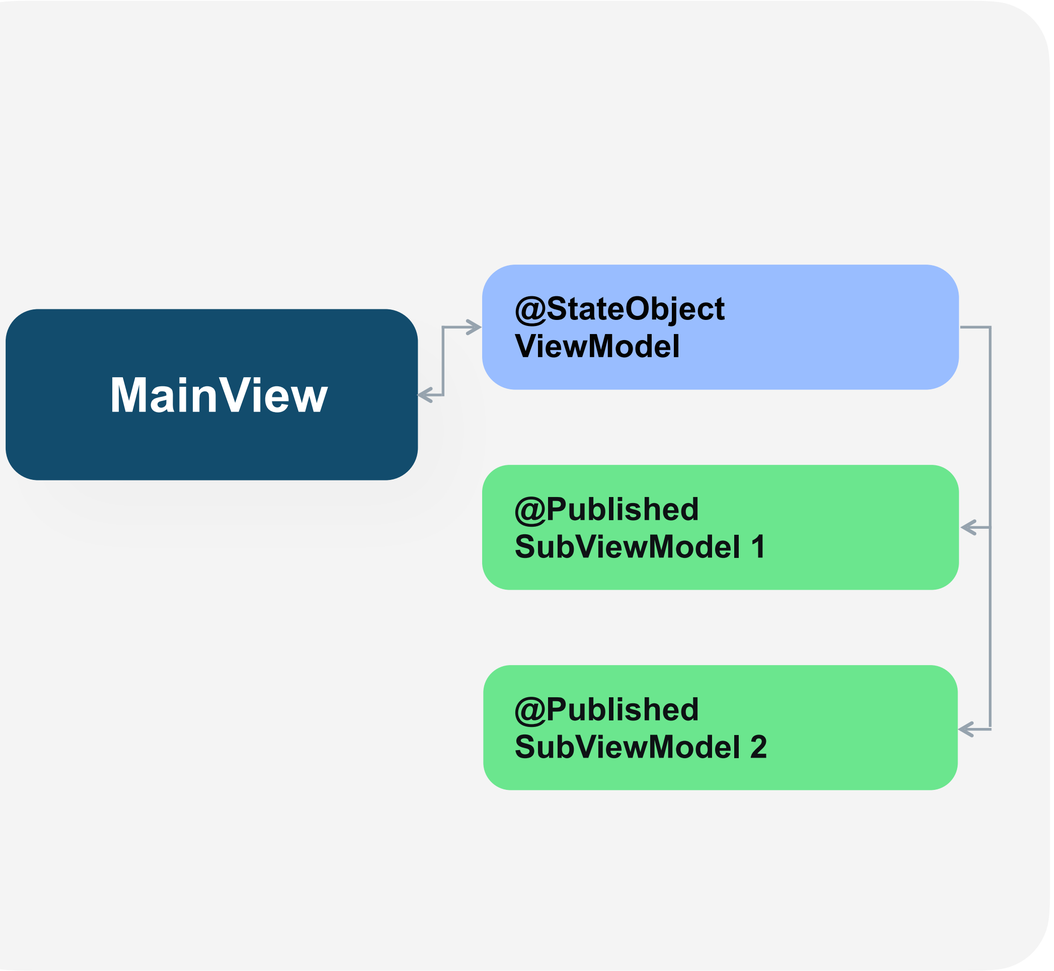
создаёт паблишер objectWillChange

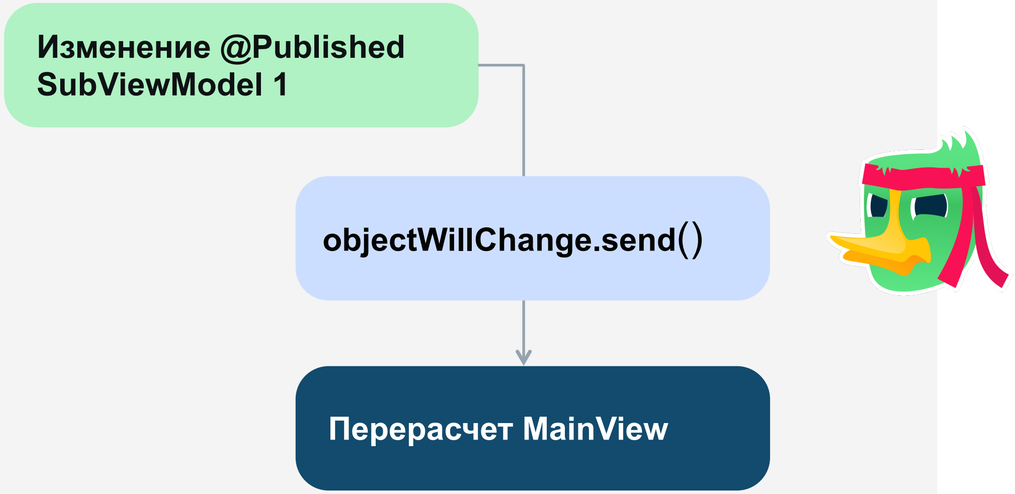


ObjectWillChange

не проверяет изменения и не знает, **ЧТО** изменилось

Проблема вложенных объектов





Проблема вложенных объектов

Несколько вариантов решений

- Вынос изменяемых свойств в стейт View
- Вызов ObjectWillChange.send() мануально
- Подписывать View на множество объектов сразу

И тут начались проблемы

- **Проблема частого перерасчета body**
- **Тесная интеграция Combine**
- **Проблема вложенных объектов**
- **Проблемы с протоколами**
- **Проблема наблюдений за коллекциями**

Проблема наблюдений за коллекциями

```
import Combine
import SwiftUI

class Model: ObservableObject {
    @Published var str = "Outer"
    @Published var innerModels: [InnerModel] = [InnerModel(), InnerModel(), InnerModel()]
}

class InnerModel {
    @Published var str = "inner"
}

struct ContentView: View {
    @ObservedObject var model: Model
    var body: some View {
        VStack {
            List(model.array, id: \.id) { element in
                Text(element.str)
            }
            Button("Hit me") {
                model.innerModels[1].str = "KABOOM"
            }
        }
        .padding()
    }
}
```

И тут начались проблемы

- Проблема частого перерасчета body
- Проблема вложенных объектов
- Проблема наблюдений за коллекциями

- Тесная интеграция Combine

- Проблемы с протоколами

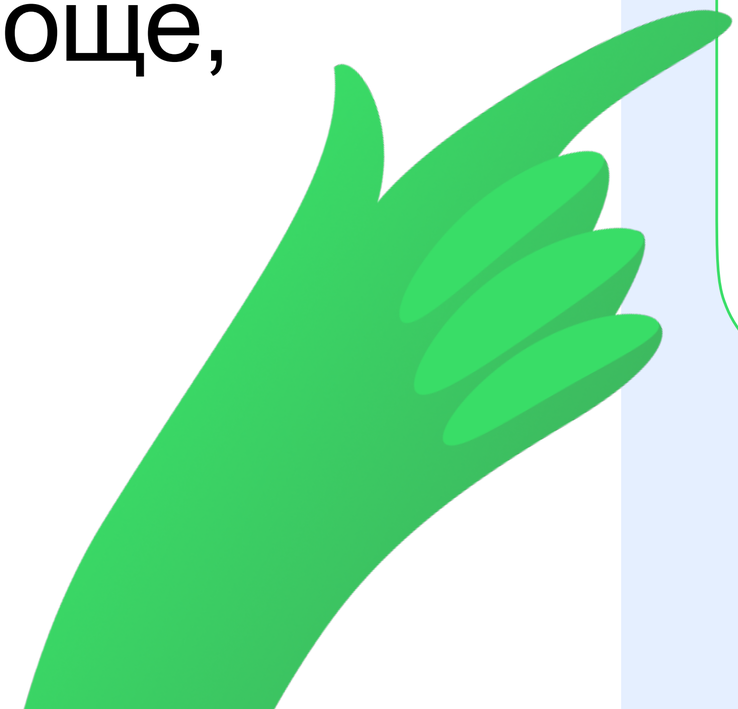
Макрос Observable

Oh hai Macro!

Макрос Observable

Доклад Анны Жарковой

Макросы Swift: проще, чище, быстрее



Почему Observable?

01

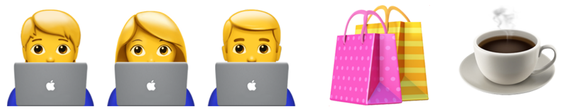
Все ссылочные типы

+

02

Кросс-платформенное использование преимуществ async/await

Правило ☕



☕ ObservableObject

Бариста говорит
всем 👨‍💻👩‍💻👨‍💻:
«Ваш ☕ готов»

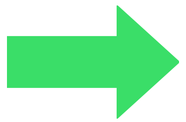
☕ @Observable

Бариста говорит
конкретному 👩‍💻:
«Ваш яблочный ☕ готов»

Простота перехода

ObservableObject ViewModel

```
final class OzonProductsViewModel: ObservableObject {
    @Published var productList: [Product]
    @Published var image = ImageResource(name: "pepe",
bundle: .main)
}
```



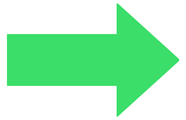
@Observable ViewModel

```
@Observable final class OzonProductsViewModel
{
    var productList: [Product]
    var image = ImageResource(name: "pepe", bundle: .main)
}
```

Простота перехода

ObservableObject View

```
struct OzonProductsView: View {
    @ObservedObject
  var viewModel: OzonProductsViewModel

    var body: some View {
        VStack {
            List(viewModel.productList) { product in
                HStack {
                    nameLabel(name: product.name)
                    Spacer()
                    priceLabel(price: product.price)
                }
                .padding(.horizontal, 12)
            }
            Spacer()
            buttons
        }
        .onAppear {
            Task {
                await viewModel.onAppear()
            }
        }
    }
}
```



@Observable View

```
struct OzonProductsView: View {
    var viewModel: OzonProductsViewModel

    var body: some View {
        VStack {
            List(viewModel.productList) { product in
                HStack {
                    nameLabel(name: product.name)
                    Spacer()
                    priceLabel(price: product.price)
                }
                .padding(.horizontal, 12)
            }
            Spacer()
            buttons
        }
        .onAppear {
            Task {
                await viewModel.onAppear()
            }
        }
    }
}
```

Макрос Observable

🏛 **@State**	Хранение стейта View
🌏 **@Environment**	Глобальный стейт
⚖️ **@Bindable**	Позволяет создавать binding к любому свойству

А что под капотом?

- **Расширяет наблюдаемый тип**
- **Вешает трекинг на свойства нашего типа**
- **Пересчитывает UI, только при изменении используемых свойств**

Наблюдаемые свойства 🔍

```
final class MyObject {
    var someProperty: String = ""
    var someOtherProperty: Int = 0
}
```

Наблюдаемые свойства 🔍

```
// Добавляем макрос
@Observable
final class MyObject {
    var someProperty: String = ""
    var someOtherProperty: Int = 0
}
```

Наблюдаемые свойства 🔍

```swift
// Добавляем макрос
@Observable
final class MyObject {
    var someProperty: String = ""
    var someOtherProperty: Int = 0
}
```

```swift
@Observable
final class MyObject {
    @ObservationTracked
    var someProperty: String = ""

    @ObservationTracked
    var someOtherProperty: Int = 0

    @ObservationIgnored
    private let _$observationRegistrar = Observation.ObservationRegistrar()

    internal nonisolated func access<Member>(
        keyPath: KeyPath<MyObject , Member>
    ) {
    _ $observationRegistrar.access(self, keyPath: keyPath)
    }

    internal nonisolated func withMutation<Member, MutationResult>(
        keyPath: KeyPath<MyObject , Member>,
        _ mutation: () throws -> MutationResult
    ) rethrows -> MutationResult {
        try _$observationRegistrar.withMutation(of: self, keyPath: keyPath, mutation)
    }
}

extension MyObject: Observation.Observable {}
```

Наблюдаемые свойства 🔍

Помечает каждое свойство макросом **@ObservationTracked**

Меняет все хранимые свойства в вычисляемые

Создает регистратор и добавляет туда свойства

Добавляет internal методы для отслеживания

Подписывает под протокол **Observable**

```
@Observable
final class MyObject {
    @ObservationTracked
    var someProperty: String = ""

    @ObservationTracked
    var someOtherProperty: Int = 0

    @ObservationIgnored
    private let _$observationRegistrar = Observation.ObservationRegistrar()

    internal nonisolated func access<Member>(
        keyPath: KeyPath<MyObject , Member>
    ) {
    _ $observationRegistrar.access(self, keyPath: keyPath)
    }

    internal nonisolated func withMutation<Member, MutationResult>(
        keyPath: KeyPath<MyObject , Member>,
        _ mutation: () throws -> MutationResult
    ) rethrows -> MutationResult {
        try _$observationRegistrar.withMutation(of: self, keyPath: keyPath, mutation)
    }
}

extension MyObject: Observation.Observable {}
```

Наблюдаемые свойства 🔍

```swift
// Добавляем макрос для наблюдаемого свойства
@Observable
final class MyObject {
  @ObservationTracked
    var someProperty: String = ""
    var someOtherProperty: Int = 0
}
```

```swift
@Observable
final class MyObject {
    @ObservationTracked
    var someProperty: String = ""
```

```swift
{
    @storageRestrictions(initializes: _someProperty)
    init(initialValue) {
      _someProperty = initialValue
    }

    get {
      access(keyPath: \.someProperty)
      return _someProperty
    }

    set {
      withMutation(keyPath: \.someProperty) {
        _someProperty = newValue
      }
    }
}
```

```swift
var someOtherProperty: Int = 0
```

willSet и didSet свойств

```
@Observable final class MyObject {
    var someProperty: String = "" {
        willSet {
            print("Значение будет установлено")
        }
        didSet {
            print("Значение было установлено")
        }
    }
    var someOtherProperty = 0
    private var somePrivateProperty = 1
}
```

Observation Registrar

Дефолтный сторадж для трекинга и отслеживания изменений свойств

Генерится макросом @Observable

Потокобезопасен

А ЧТО ВНУТРИ?

ObservationRegistrar

- **Идентификация транзакции и обновление View**

```
public struct ObservationRegistrar : Sendable {

    /// Регистрируем свойство для наблюдения
    public func access<Subject, Member>(
        _ subject: Subject, keyPath: KeyPath<Subject, Member>
    ) where Subject : Observable

    /// Метод, вызываемый перед установкой значения.
    public func willSet<Subject, Member>(
        _ subject: Subject, keyPath: KeyPath<Subject, Member>
    ) where Subject : Observable

    /// Метод, вызываемый после установки значения.
    public func didSet<Subject, Member>(
        _ subject: Subject, keyPath: KeyPath<Subject, Member>
    ) where Subject : Observable

    /// Идентифицирует изменения в транзакциях
    зарегистрированных для наблюдателей.
    public func withMutation<Subject, Member, T>(
        of subject: Subject,
        keyPath: KeyPath<Subject, Member>, _ mutation: () throws -> T
    ) rethrows -> T where Subject : Observable
}
```

Наблюдаем вычисляемые свойства

Реализуем подписку
на вычисляемые свойства

01 **Реализуем стор**

02 **Реализуем геттер и сеттер**

03 **Добавляем обращения к методам макроса**

```
@Observable final class MyObject {
    @ObservationIgnored
   fileprivate let _observableOzonProductsStore =
   ObservableOzonProductsViewModel()

    var firstProductName: String {
        get {
            self.access(keyPath: \.firstProductName)
            return _observableOzonProductsStore.productList[0].name
        }
        set {
            self.withMutation(keyPath: \.firstProductName) {
                _observableOzonProductsStore.productList[0].name =
newValue
            }
        }
    }

    private var somePrivateProperty = 1
    var someProperty: String = ""
}
```

withObservation Tracking

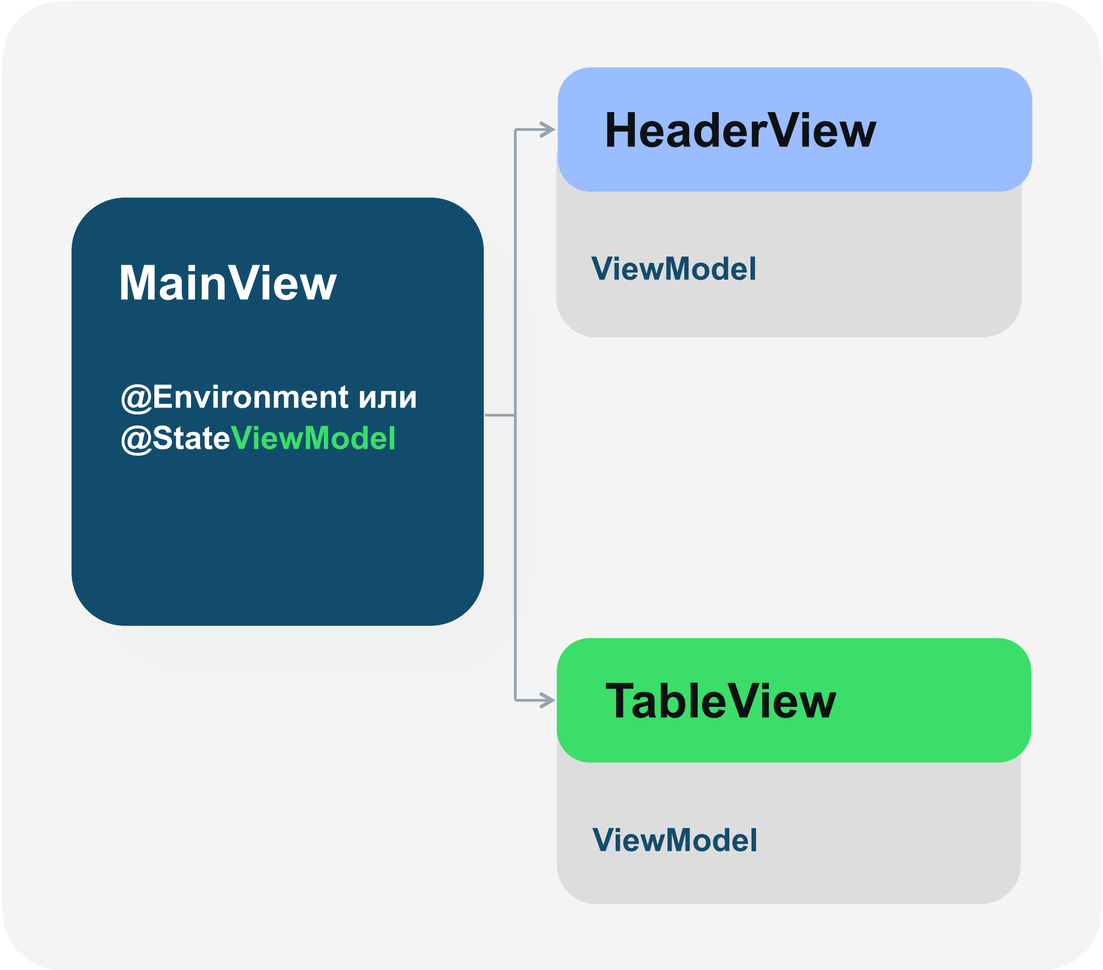
- **Ограничиваем отслеживание в конкретном скоупе**

```
func printProducts() {
      withObservationTracking {
          for product in productList {
              print(product.name)
          }
      } onChange: {
          Task { @MainActor in
              printProducts()
          }
      }
 }
```

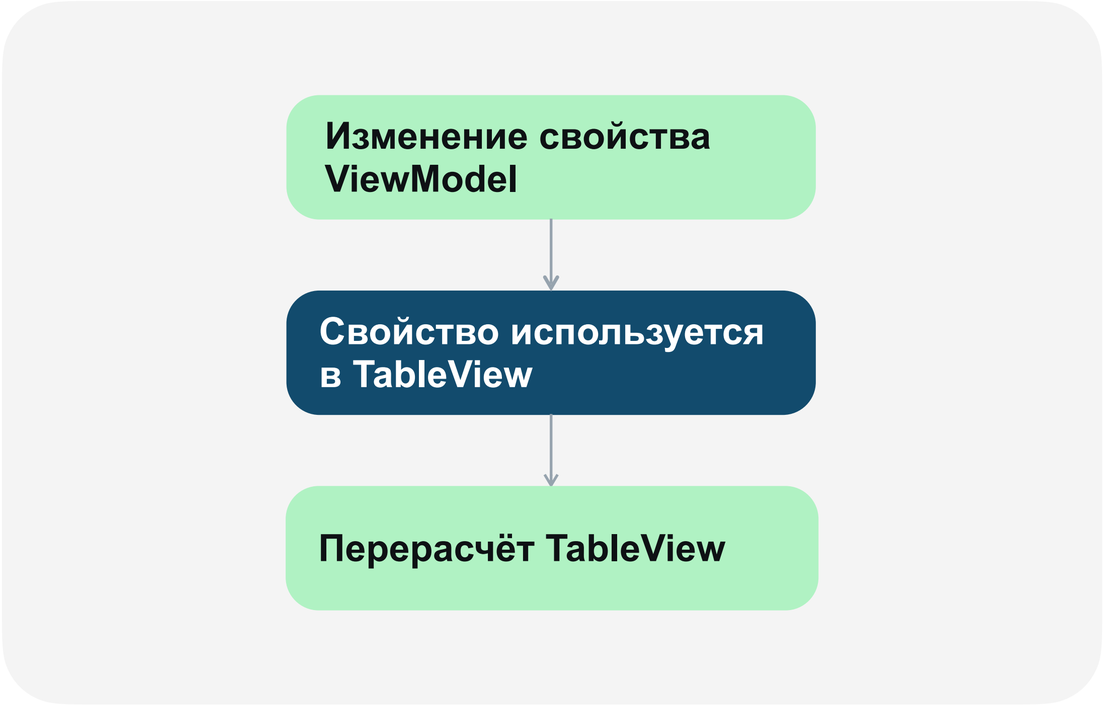
Макрос Observable

- ~~Проблема частого перерасчета body~~
- ~~Тесная интеграция Combine~~
- ~~Проблема вложенных объектов~~
- ~~Проблемы с протоколами~~
- ~~Проблема наблюдений за коллекциями~~

Проблема частого перерасчета body



Меняем ячейку таблицы



Проблемы @Observable

И снова о проблемах

- **Ограничение по версии оси**
- **Более точные манипуляции с анимациями**
- **Все наблюдаемые свойства должны иметь дефолтные значения**

И снова о проблемах

- **Ограничение по версии оси**
- **Более точные манипуляции с анимациями**
- **Все наблюдаемые свойства должны иметь дефолтные значения**

А КАК?

Ограничение по версии оси


```
- @Observable
@Perceptible

final class ReviewsViewModel: IReviewsViewModel
{
    var state: State?
    var isBottomSheetPresented = false
}
```

Perceptible-библиотека

- **А так же @Bindable и @Environment**

```
struct FeatureView: View {
    let model: FeatureModel

    var body: some View {
        WithPerceptionTracking {
            Form {
                Text(model.count.description)
                Button("Increment")
{ model.count += 1 }
            }
        }
    }
}
```

И снова о проблемах

Ограничение по версии оси

Более точные манипуляции с анимациями

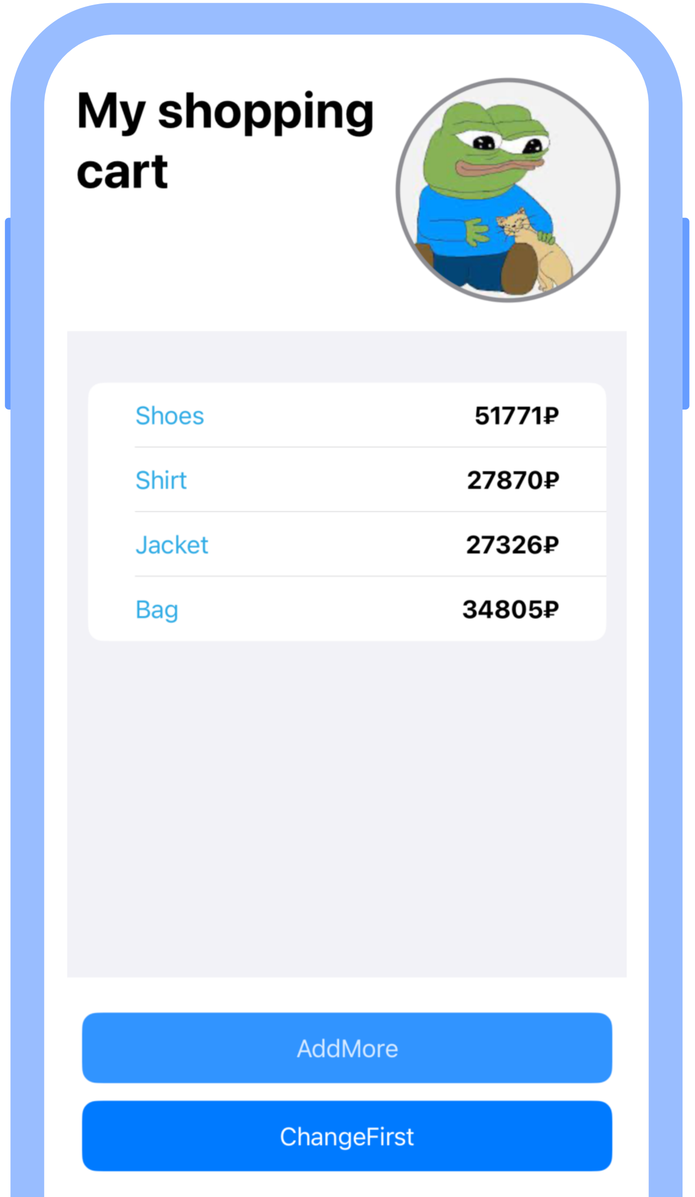
Все наблюдаемые свойства должны иметь дефолтные значения

А в чём скорость?

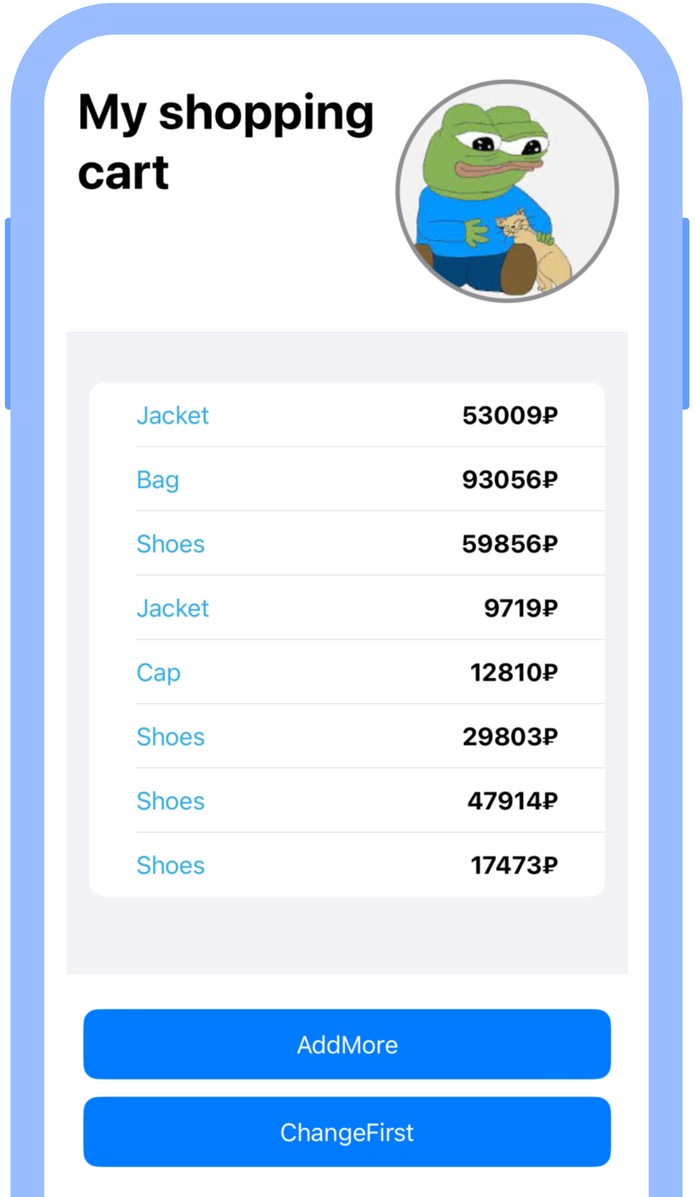
А в чём скорость?

```
OzonImage: _ozonProductsViewModel changed.
OzonProductsView: _viewModel changed.
```

Filter



А в чём скорость?



А в чём скорость?

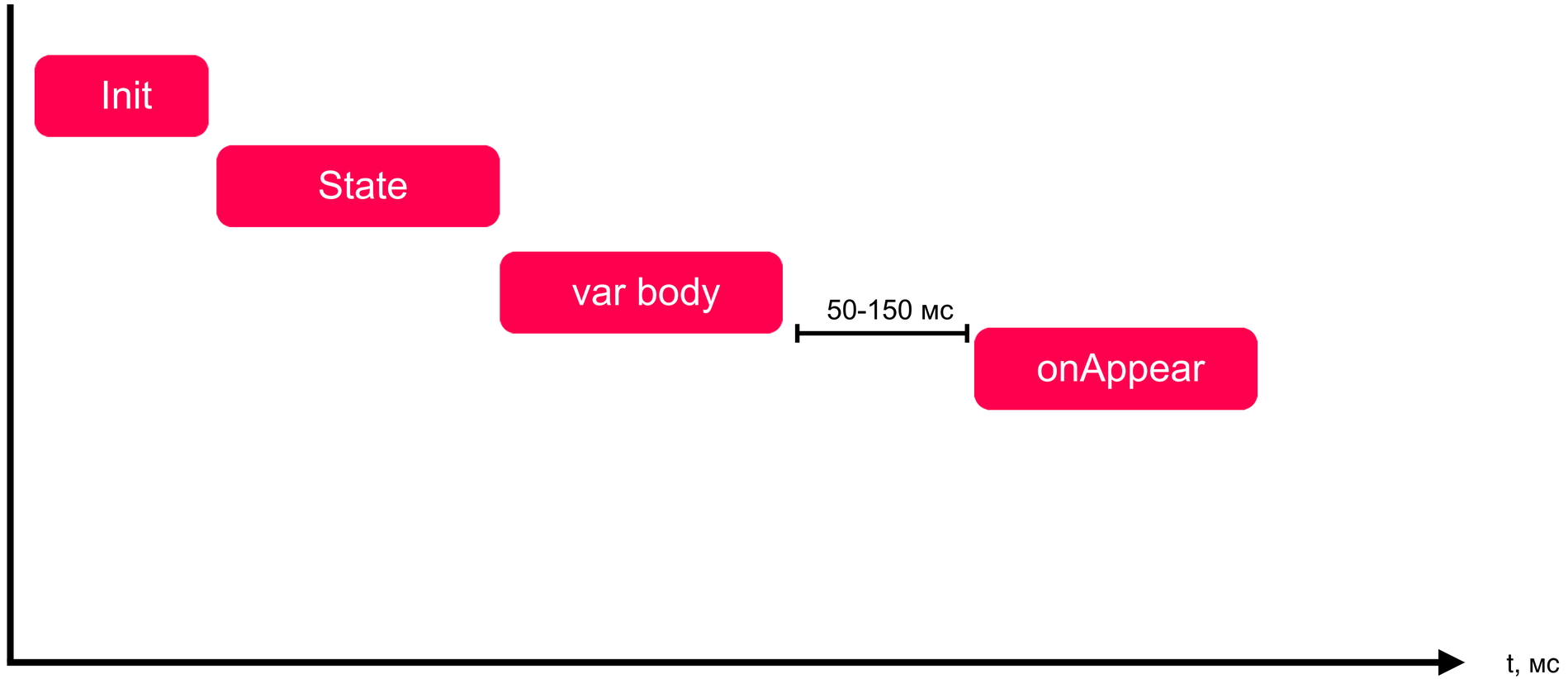
Количество изменений x
Количество View

При 1 изменении и 30 элементов —
буст в 30 вызовов перерасчетов body дочерних View

Оверхэд всего семейства @Object

```
struct ObservableOzonProductsView: View {
    var viewModel: IObservableOzonProductsViewModel

    var body: some View {
        VStack {
            List(viewModel.productList) { currentItem in
                HStack {
                    ProductName(product: currentItem)
                    Spacer()
                    ProductPrice(product: currentItem)
                }
                .padding(.horizontal, 12)
            }
            Spacer()
            buttons
        }
        .onAppear {
                    Task {
                        await viewModel.onAppear()
                    }
                }
            }
          }
```

Оверхэд всего семейства @Object



Разница в вызове методов onAppear около 50мс-150мс

А что дальше?

А что дальше?

- Все наблюдаемые свойства должны иметь дефолтные значения
- Поддержка наблюдаемых акторов
- Работа с asyncSequences

OZON tech

Спасибо за внимание!

Чернов Дмитрий
Руководитель группы в OzonTech
@NSFlood
dmchernov