SwiftUI

Better apps. Less code.

Pedro Ullmann | Fuze

SwiftUI

Definição

“Declare a interface do usuário e o comportamento do seu aplicativo em todas as plataformas.”

SwiftUI

O inicio

- WWDC | 2019
- Uma nova maneira de pensar
- Foco nas funcionalidades de UI e não nas especificações de layout
- Framework de UI declarativa

UIKit

Framework imperativo

- **Detalhado e complexo**
- **Controle manual**
- **Foco nas especificações de layout**

"Instancie um UIButton, posicione-o na coordenada (X, Y), defina sua altura como Z e sua largura como W e adicione como SubView."

Imperativo

```
let button = UIButton(type: .system)
button.frame = CGRect(x: X, y: Y, width: W, height: Z)
button.setTitle("Botão", for: .normal)
button.addTarget(self, action: #selector(action), for: .touchUpInside)

view.addSubview(button)

@objc func action() {
    // Ação
}
```

SwiftUI

Framework declarativo

- **Conciso e Descritivo**
- **Orientado a resultados**
- **Não precisa focar nas especificações de layout**

"Crie um botão"

Declarativo

```
struct ContentView: View {
    var body: some View {
        Button(
            action: { /* Ação */ },
            label: { Text("Botão") }
        )
    }
}
```

SwiftUI

Multiplataforma

iOS | iPadOS | TvOS

UIKit

SwiftUI

Multiplataforma

iOS | iPadOS | TvOS

UIKit

MacOS

AppKit

SwiftUI

Multiplataforma

iOS | iPadOS | TvOS	MacOS	WatchOS
UIKit	AppKit	WatchKit

SwiftUI

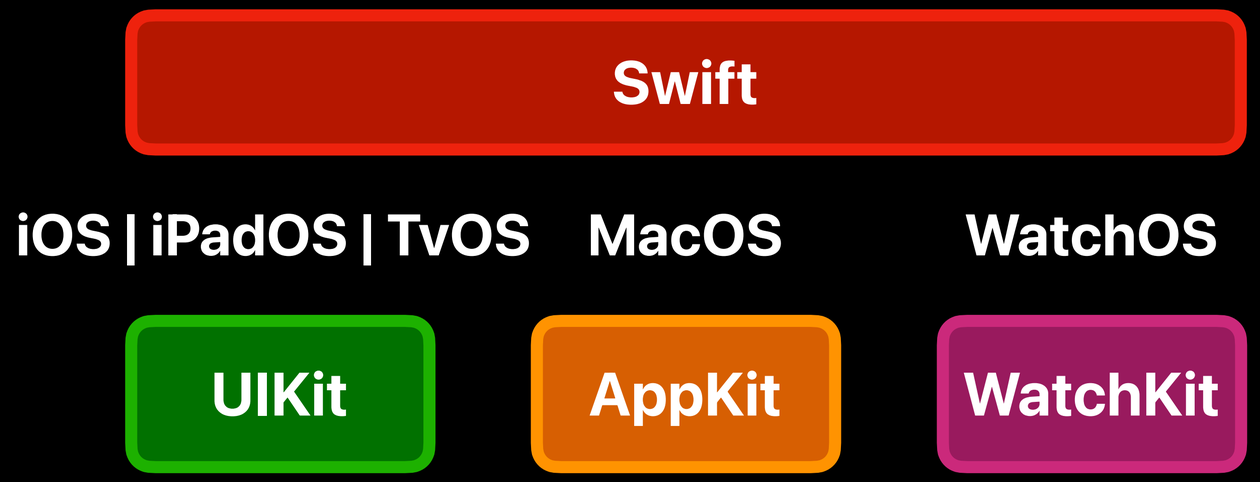
Multiplataforma

Objective-C 💀

iOS | iPadOS | TvOS	MacOS	WatchOS
UIKit	AppKit	WatchKit

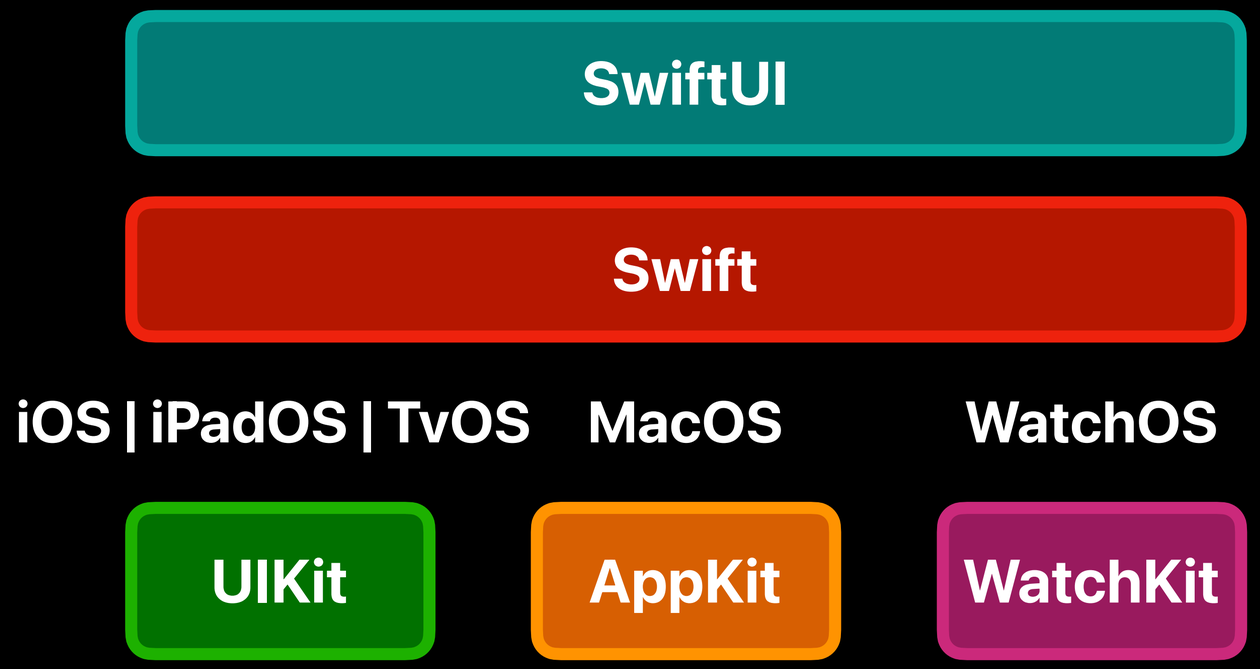
SwiftUI

Multiplataforma



SwiftUI

Multiplataforma



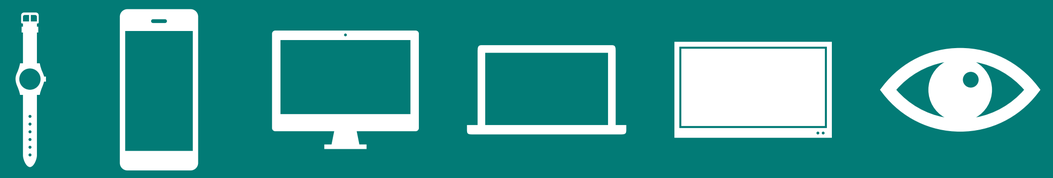
SwiftUI

Multiplataforma

SwiftUI

Swift

iOS | iPadOS | TvOS | MacOS | WatchOS | VisionOS



SwiftUI

E quais são as vantagens?

- Fácil de aprender
- O código é simples e limpo
- Pode ser usado junto ao UIKit
- Live Preview
- Sem mais @IBOutlet ou constraints 🥺

Hype

Hold your horses!

UIKit

🗑️ 🔥 🤢

Hype Hold your horses!

UIKit

- Padrão de mercado
- Mais estável
- Comunidade mais engajada
- Mais documentação

There is no silver bullet

SwiftUI

E quais são as desvantagens?

- Disponível apenas para iOS > 13
- Recursos limitados (com ressalvas)
- Depuração
- Personalização avançada (nativa)
- Anomalias com navegação

UIKit vs SwiftUI

UIKit & SwiftUI

SwiftUI

Quais apps já usam?

Books
Maps
Notes
Weather
Music
Podcasts

Disney+
Spotify
Adidas
Duolingo
OLX
Warren

SwiftUI Basics

```
import SwiftUI
```

```
struct ContentView: View {
    var body: some View {
        Text("Hello, world!")
    }
}
```

Performance, Value Type

```
struct ContentView: View {
    var body: some View {
        Text("Hello, world!")
    }
}
```

View Name

```
struct ContentView: View {
    var body: some View {
        Text("Hello, world!")
    }
}
```

Protocol

```
struct ContentView: View {
    var body: some View {
        Text("Hello, world!")
    }
}
```

```
struct ContentView: View {
    var body: some View {
        Text("Hello, world!")
    }
}
```

Computed Property

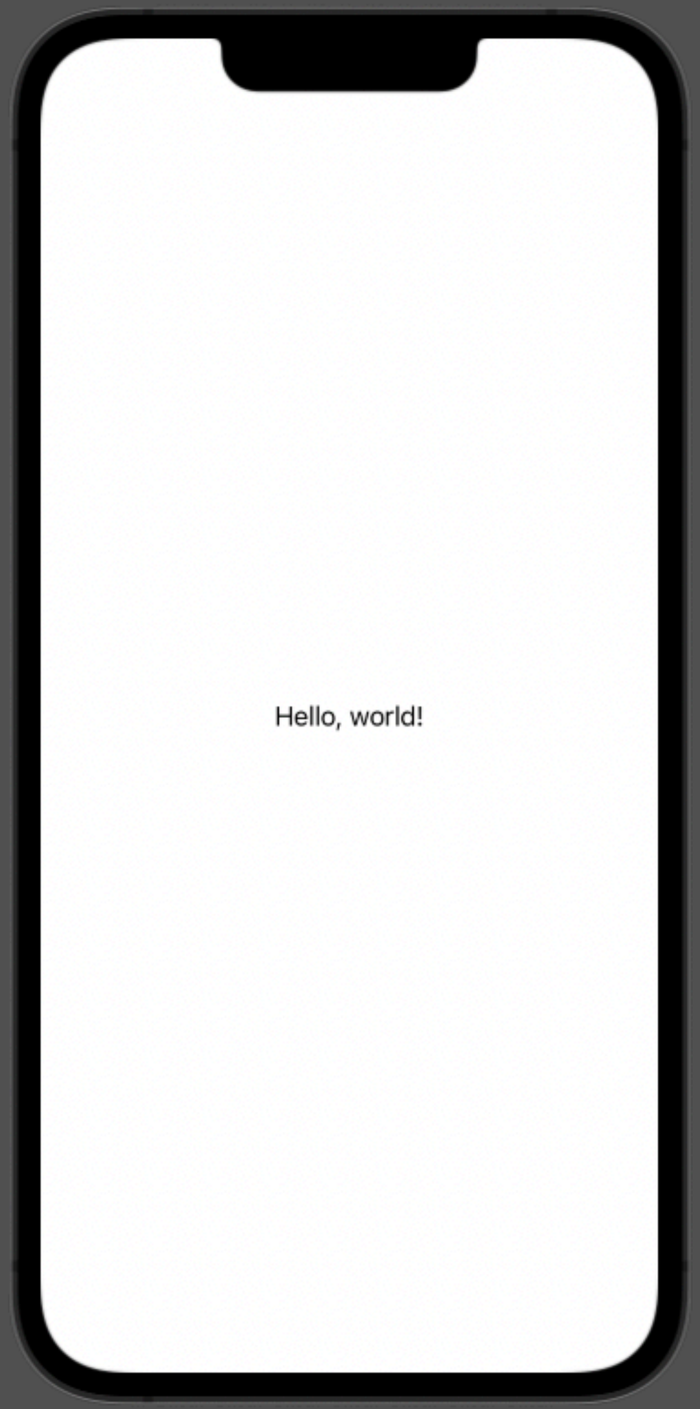
Opaque Type

```
struct ContentView: View {
    var body: some View {
        Text("Hello, world!")
    }
}
```

```
struct ContentView: View {
    var body: some View {
        Text("Hello, world!")
    }
}
```

View

```swift
struct ContentView: View {
    var body: some View {
        Text("Hello, world!")
    }
}
```



SwiftUI Views

```
Text()

Button()

Image()

TextField()

SecureField()

Toggle()

Slider()

Stepper()
```

```
DatePicker()

Picker()

SegmentedControl()

ProgressView()

ActivityIndicator()

VStack {
    /* Views */
}

HStack {
    /* Views */
}
```

```
ZStack {
    /* Views */
}

List {
    /* Views */
}

ScrollView {
    /* Views */
}

TabView {
    /* Views */
}

Section {
    /* Views */
}
```

```
Alert()

ActionSheet()

Group()

Spacer()

Divider()

EmptyView()

AnyView()

Color()

Rectangle()
```

```
RoundedRectangle()

Circle()

Ellipse()

Capsule()

Path()

GeometryReader()

LinearGradient()

RadialGradient()

AngularGradient()
```

E MUITO MAIS...

SwiftUI Layout Process

"Layout é decidir o tamanho das coisas na tela"

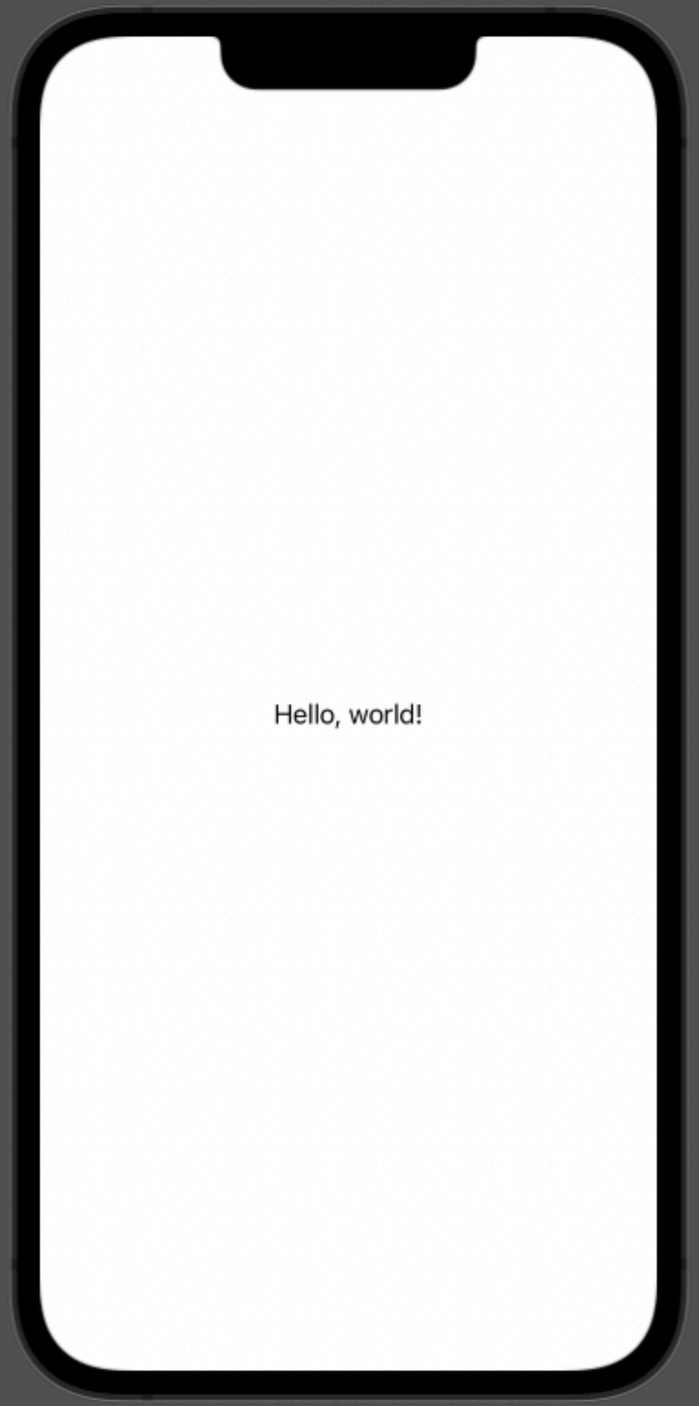
SwiftUI Layout Process

```
struct ContentView: View {
    var body: some View {
        Text("Hello, world!")
    }
}
```

RootView

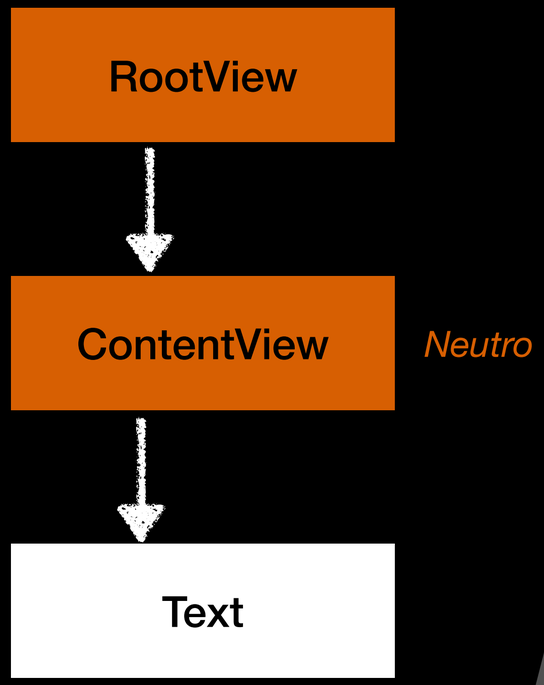
ContentView

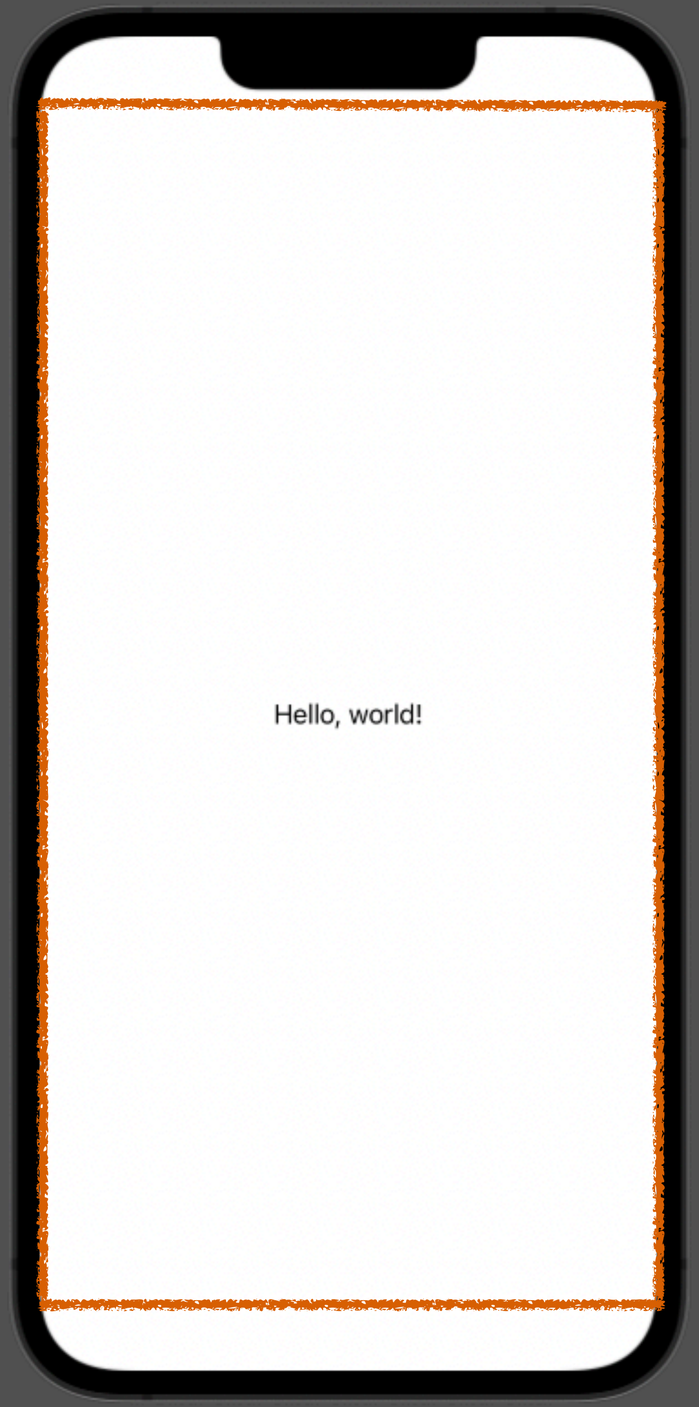
Text



SwiftUI Layout Process

1. O pai propõe um tamanho para o filho

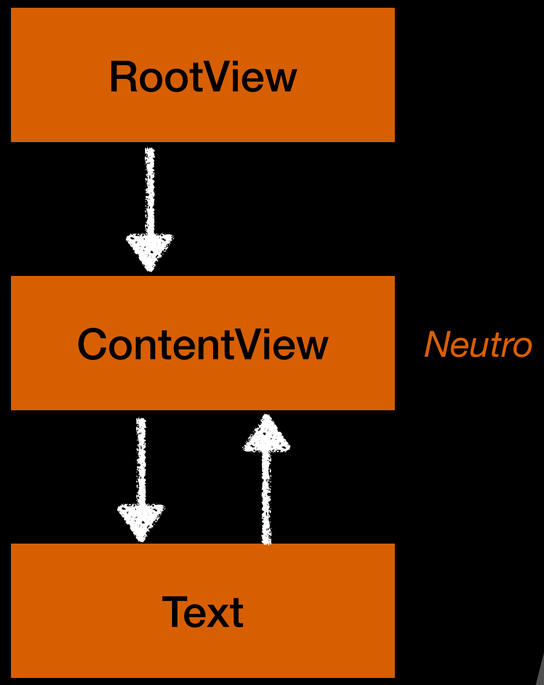




SwiftUI Layout Process

1. O pai propõe um tamanho para o filho

2. O filho escolhe seu próprio tamanho

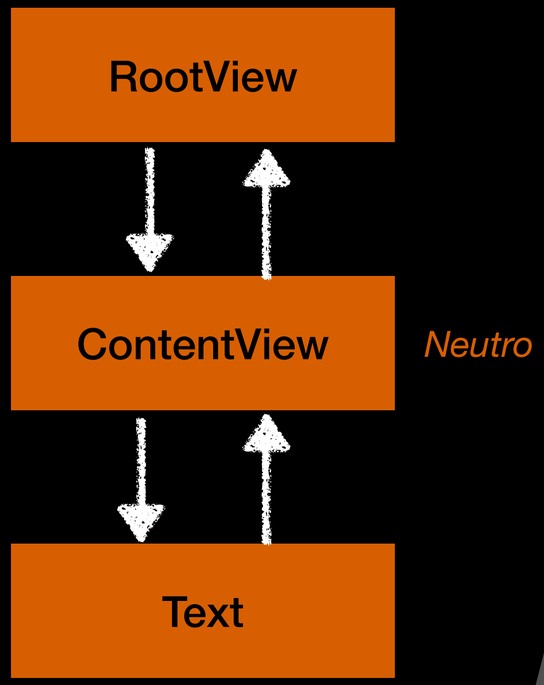


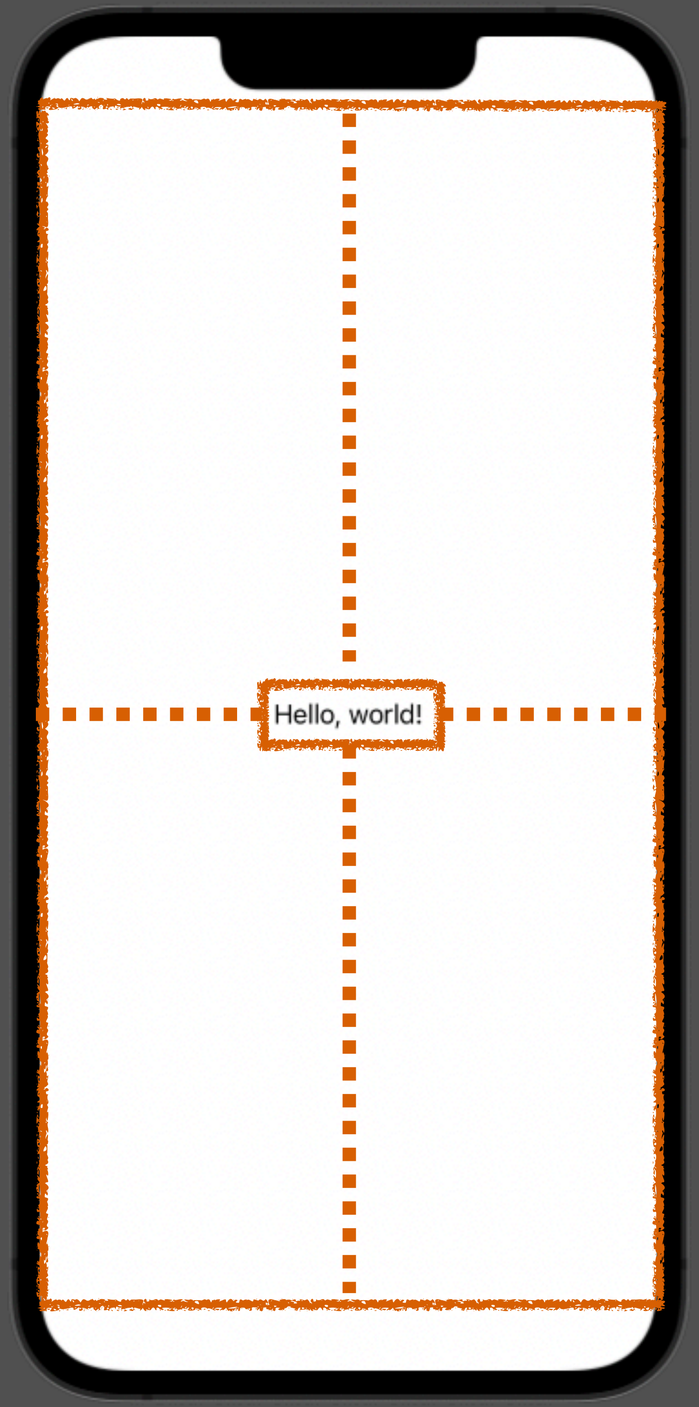
SwiftUI Layout Process

1. O pai propõe um tamanho para o filho

2. O filho escolhe seu próprio tamanho

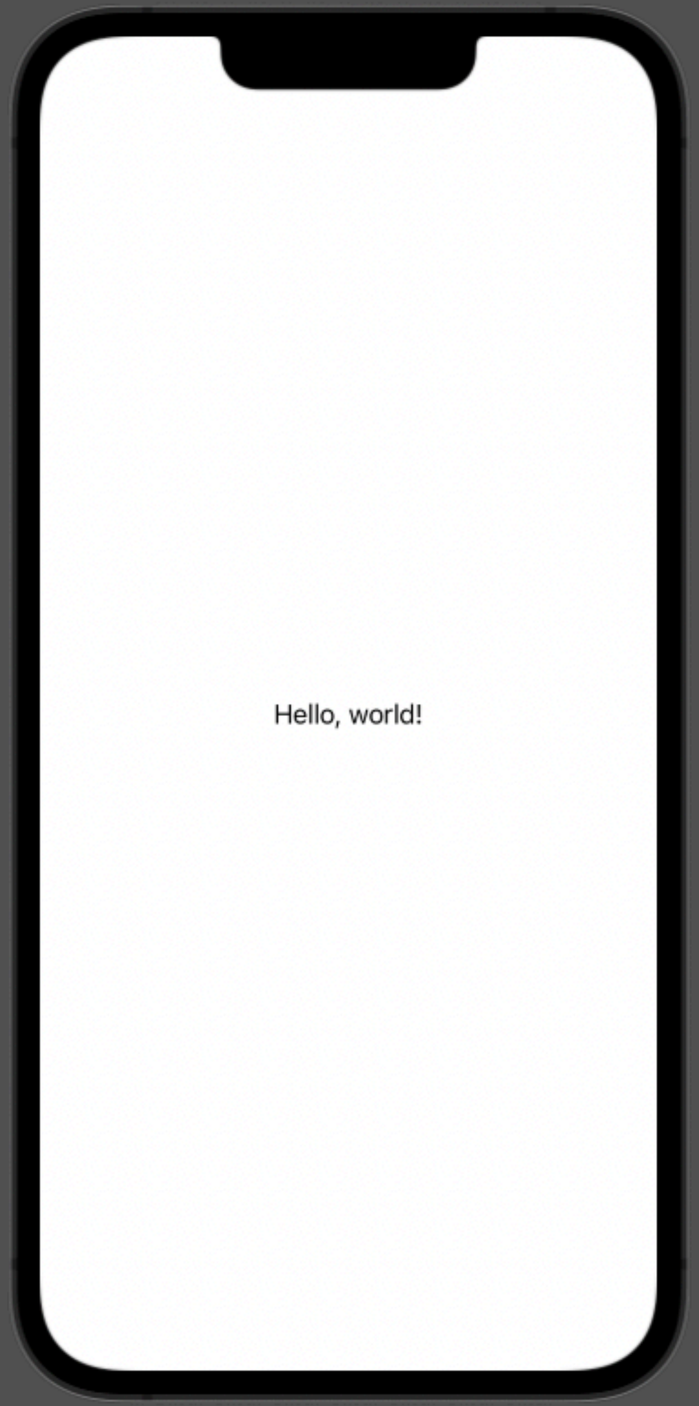
3. O pai posiciona o filho na coordenada do pai





SwiftUI Layout Process

```
struct ContentView: View {
    var body: some View {
        Text("Hello, world!")
    }
}
```



SwiftUI Property Wrappers

“É uma forma de encapsular o acesso e a modificação de uma propriedade, fornecendo uma camada de funcionalidade extra”

```swift
struct ContentView: View {
    @State var counter: Int = 0

    var body: some View {
        VStack {
            Text("\(counter)")

            Button(
                action: { counter += 1 },
                label: { Text("Incrementar") }
            )
        }
    }
}
```

Atributo

```
@propertyWrapper
public struct State<Value>: DynamicProperty
```

```swift
@propertyWrapper
public struct State<Value>: DynamicProperty
```

Valor genérico

Protocolo

```
@propertyWrapper
public struct State<Value>: DynamicProperty
```

```swift
public protocol DynamicProperty {

    /// Updates the underlying value of the stored value.
    ///
    /// SwiftUI calls this function before rendering a view's
    /// ``View/body-swift.property`` to ensure the view has the most recent
    /// value.
    mutating func update()
}
```

@State: *Armazena um estado mutável dentro de uma struct*

```swift
@propertyWrapper
struct UserDefaultsWrapper<Value: Codable> {
    private let key: String
    private let defaultValue: Value

    init(key: String, defaultValue: Value) {
        self.key = key
        self.defaultValue = defaultValue
    }

    var wrappedValue: Value {
        get { UserDefaults.standard.object(forKey: key) as? Value ?? defaultValue }
        set { UserDefaults.standard.set(newValue, forKey: key) }
    }
}
```

```swift
struct ContentView: View {
    @UserDefaultsWrapper(key: "usuario", defaultValue: "")
    var username: String

    var body: some View {
        Text(username)
    }
}
```

```swift
@propertyWrapper
struct UserDefaults
```

Permite armazenar e recuperar valores no UserDefaults

```swift
@propertyWrapper
struct Capitalized
```

Converte automaticamente uma string para maiúsculas na primeira letra de cada palavra.

```swift
@propertyWrapper
struct NonNegative
```

Garante que um valor numérico seja sempre não negativo, ajustando automaticamente para zero se um valor negativo for atribuído.

```swift
@propertyWrapper
struct Localized
```

Localiza automaticamente uma string

```swift
@propertyWrapper
struct Dependency
```

Service Locator (Container)

```swift
@propertyWrapper
public struct Dependency<T> {

    // MARK: - Properties
    private let value: T

    // MARK: - Initialization
    public init(resolvedValue: T? = nil) {
        if let resolved = resolvedValue {
            self.value = resolved
        } else if let instance = ServiceContainer.shared.resolve(T.self) {
            self.value = instance
        } else {
            fatalError("Service '\(T.self)' not registered")
        }
    }

    // MARK: - Public methods
    public var wrappedValue: T { value }

    public static func resolved(_ instance: T) -> Self {
        .init(resolvedValue: instance)
    }
}
```

Registrando a dependência

```
let container = ServiceContainer.shared

container.register(
    APIServiceProtocol.self, /// Interface
    factory: { APIService() } /// Concreto
)
```

Usando a dependência

```
final class ControladorLogico {
    @Dependency var apiService: APIServiceProtocol
}
```

SwiftUI Property Wrappers

@EnvironmentObject

@Binding

@State

@Environment

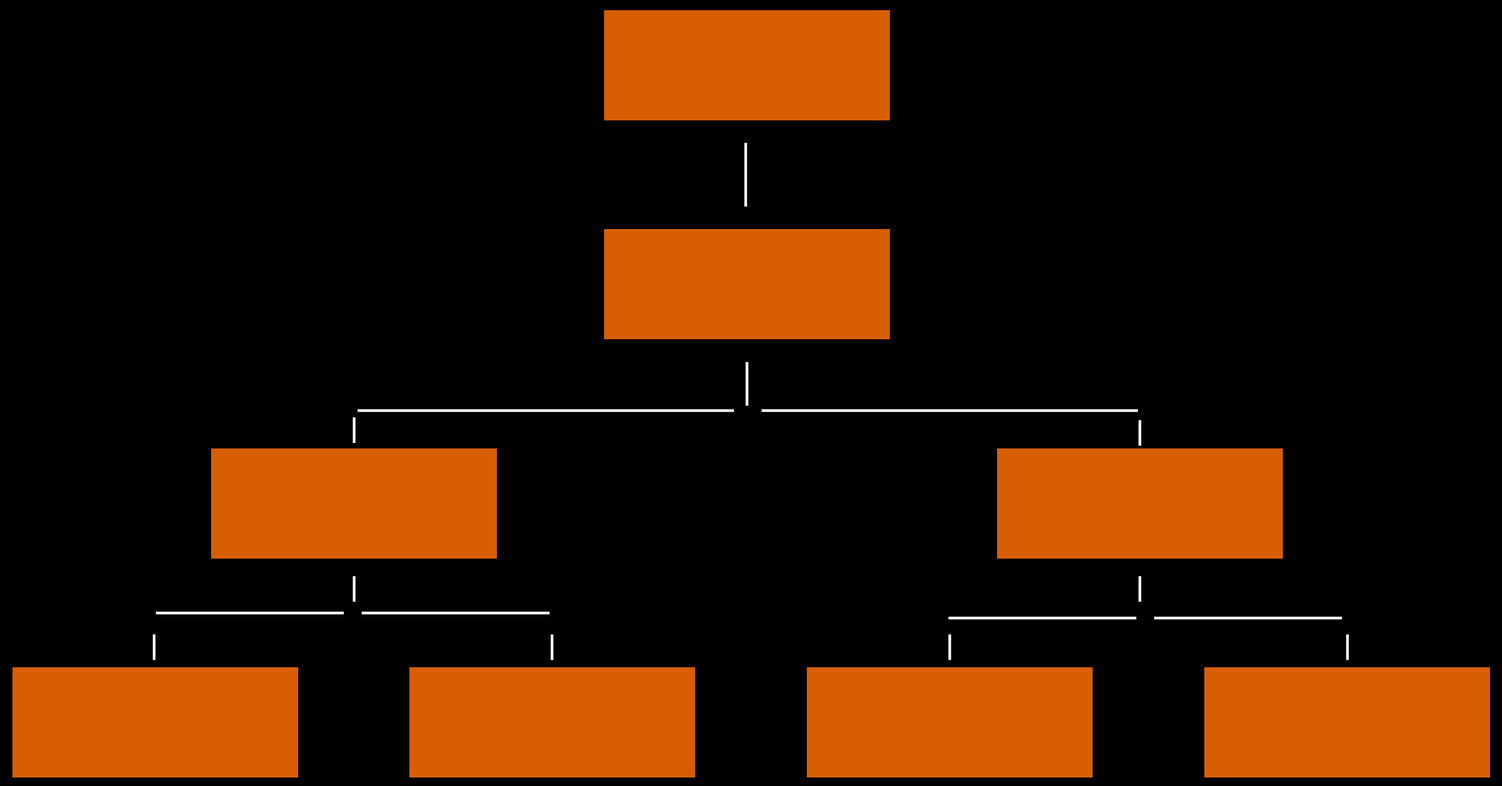
@StateObject

@AppStorage

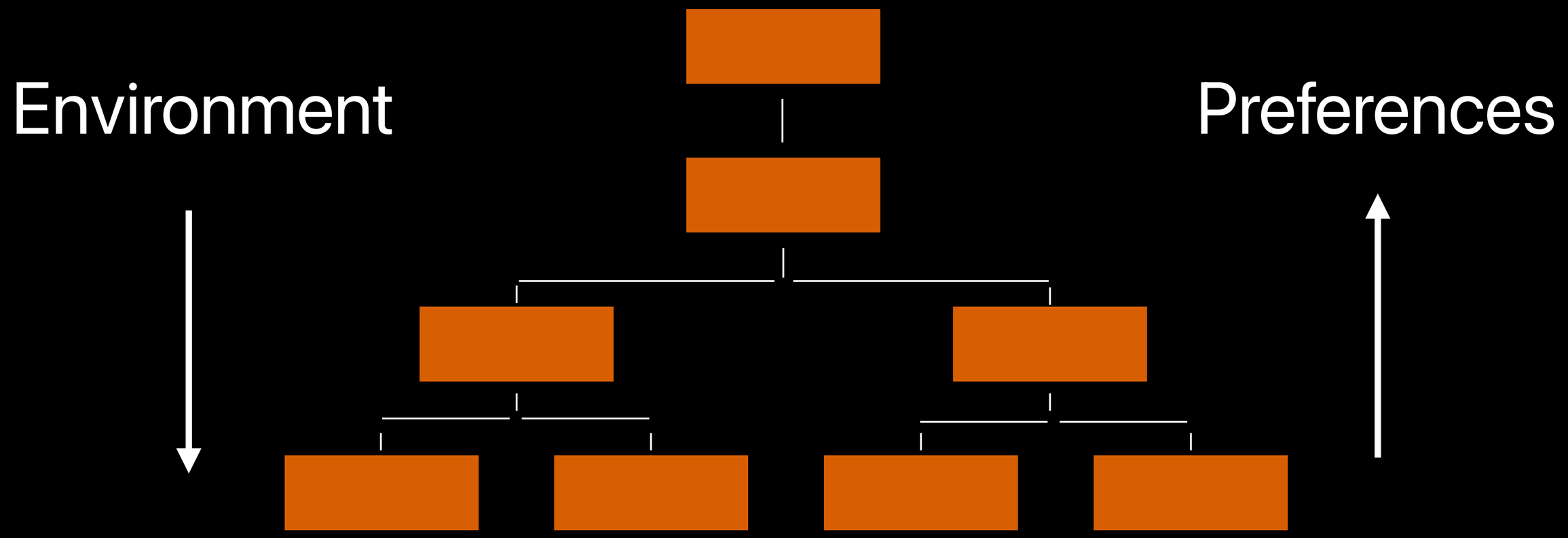
@ObservedObject

@GestureState

SwiftUI Data Flow



SwiftUI Data Flow



Environment ↓

Conjunto de informações passadas de uma view para seus filhos

Environment ↓

```
struct ContentView: View {
    var body: some View {
        VStack {
            HStack {
                Text("Pedro")
                Text("Ullmann")
            }

            HStack {
                Text("Desenvolvedor")
                Text("iOS")
            }
        }
    }
}
```

Pedro Ullmann
Desenvolvedor iOS

Environment ↓

```
struct ContentView: View {
    var body: some View {
        VStack {
            HStack {
                Text("Pedro")
                Text("Ullmann")
            }
            .environment(\.font, .title)

            HStack {
                Text("Desenvolvedor")
                Text("iOS")
            }
            .environment(\.font, .subheadline)
        }
    }
}
```

Pedro Ullmann

Desenvolvedor iOS

Environment ↓

```
.colorScheme
.layoutDirection
.locale
.sizeCategory
.accessibilityDifferentiateWithoutColor
.accessibilityReduceTransparency
.accessibilityReduceMotion
.multilineTextAlignment
.lineSpacing
.truncationMode
.minimumScaleFactor
.isEnabled
.presentationMode
.managedObjectContext
.undoManager
.horizontalSizeClass
.verticalSizeClass
.legibilityWeight
.layoutPriority
.defaultMinListRowHeight
.ignoresSafeArea
.presentationStyle
.allowsTightening
.lineLimit
.allowsHitTesting
.statusBarStyle
.menuButtonStyle
```

Environment ↓

.colorScheme
.layoutDirection
.locale
.sizeCategory
.accessibilityDifferentiateWithoutColor
.accessibilityReduceTransparency
.accessibilityReduceMotion
.multilineTextAlignment
.lineSpacing
.truncationMode
.minimumScaleFactor
.isEnabled
.presentationMode
.managedObjectContext
.undoManager
.horizontalSizeClass
.verticalSizeClass
.legibilityWeight
.layoutPriority
.defaultMinListRowHeight
.ignoresSafeArea
.presentationStyle
.allowsTightening
.lineLimit
.allowsHitTesting
.statusBarStyle
.menuButtonStyle

Environment ↓

```
struct LoadingEnvironment: EnvironmentKey {
    typealias Value = Bool
    static let defaultValue: Bool = false
}

extension EnvironmentValues {
    var isLoading: Bool {
        get { self[LoadingEnvironment.self] }
        set { self[LoadingEnvironment.self] = newValue }
    }
}
```

Environment ↓

```
struct ContentView: View {
    var body: some View {
        Component()
            .environment(\.isLoading, true)
    }
}

struct Component: View {
    @Environment(\.isLoading) var isLoading: Bool

    var body: some View {
        VStack {
            Text(isLoading ? "Carregando": "Terminou")

            Button(
                action: { /* Ação */ },
                label: { Text("Tentar novamente") }
            )
            .disabled(isLoading)
        }
    }
}
```

Carregando

Tentar novamente

Environment ↓

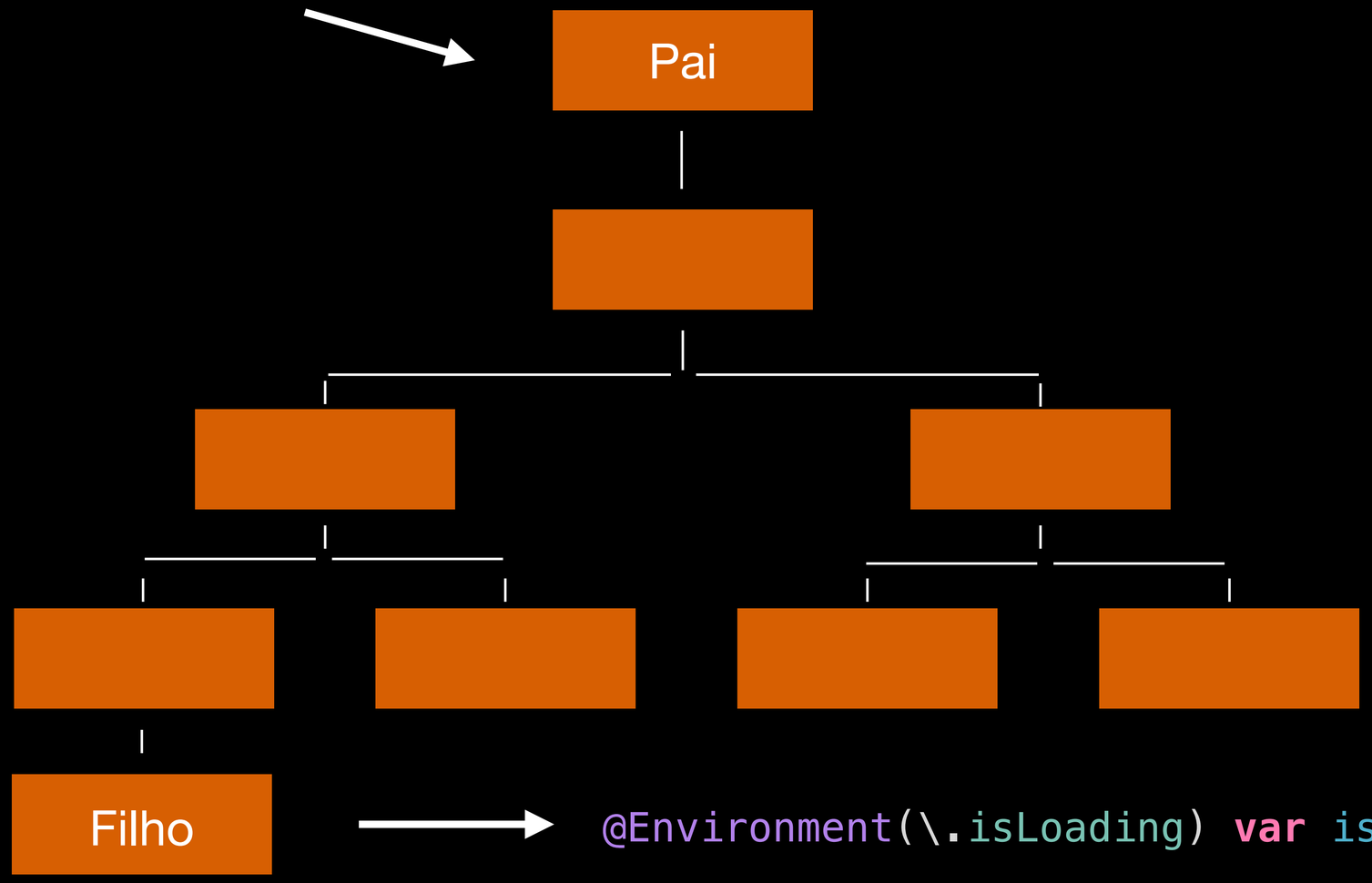
```
struct ContentView: View {
    var body: some View {
        Component()
            .environment(\.isLoading, false)
    }
}

struct Component: View {
    @Environment(\.isLoading) var isLoading: Bool

    var body: some View {
        VStack {
            Text(isLoading ? "Carregando": "Terminou")

            Button(
                action: { /* Ação */ },
                label: { Text("Tentar novamente") }
            )
            .disabled(isLoading)
        }
    }
}
```

Terminou

Tentar novamente

Environment ↓

```
.environment(\.isLoading, true)
```




```
@Environment(\.isLoading) var isLoading: Bool
```

Environment ↓

```
extension View {
    func isLoading(_ value: Bool) -> some View {
        self.environment(\.isLoading, value)
    }
}

struct ContentView: View {
    var body: some View {
        Component()
            .isLoading(true)
    }
}
```

Environment ↓

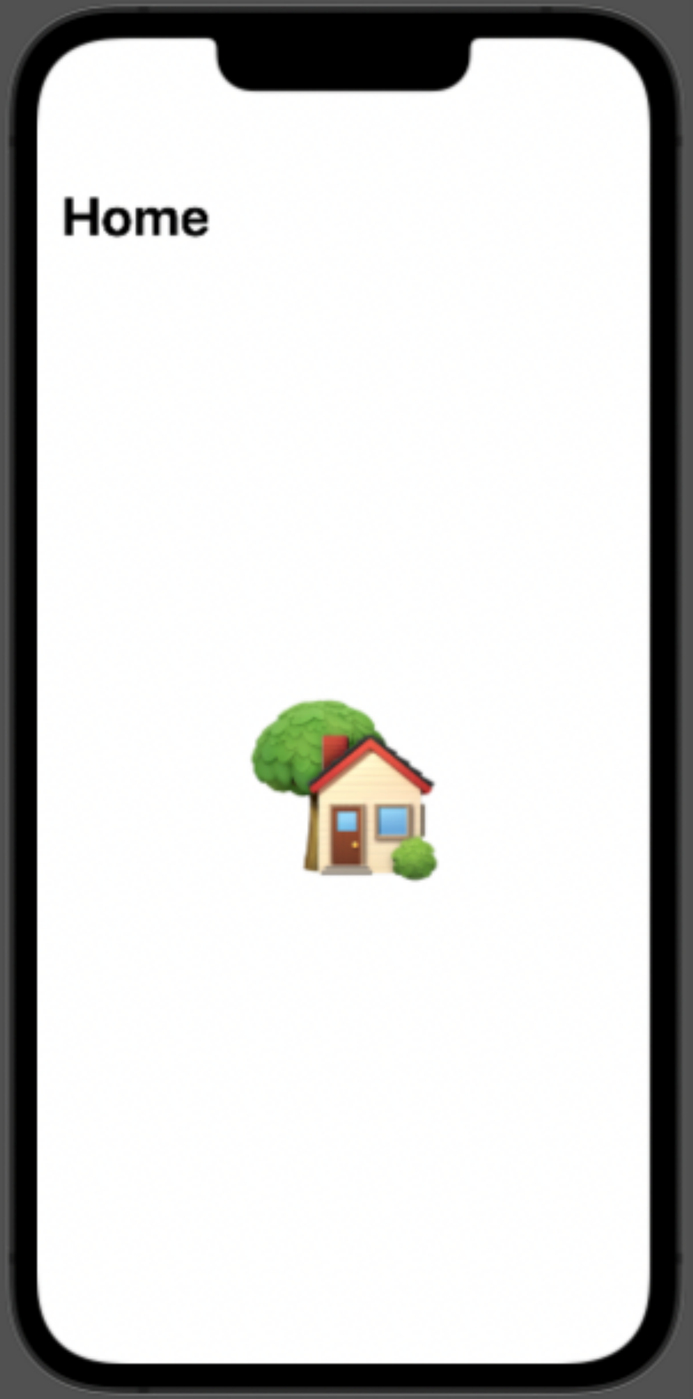
```
struct ContentView: View {
    var body: some View {
        Button(
            action: { /* Ação */ },
            label: { Text("Botão") }
        )
        .disabled(true)
    }
}

extension View {
    func disabled(_ value: Bool) -> some View {
        self.environment(\.isEnabled, !value)
    }
}
```

Preferences ↑

Mecanismo para uma view expressar suas preferencias para seus ascendentes

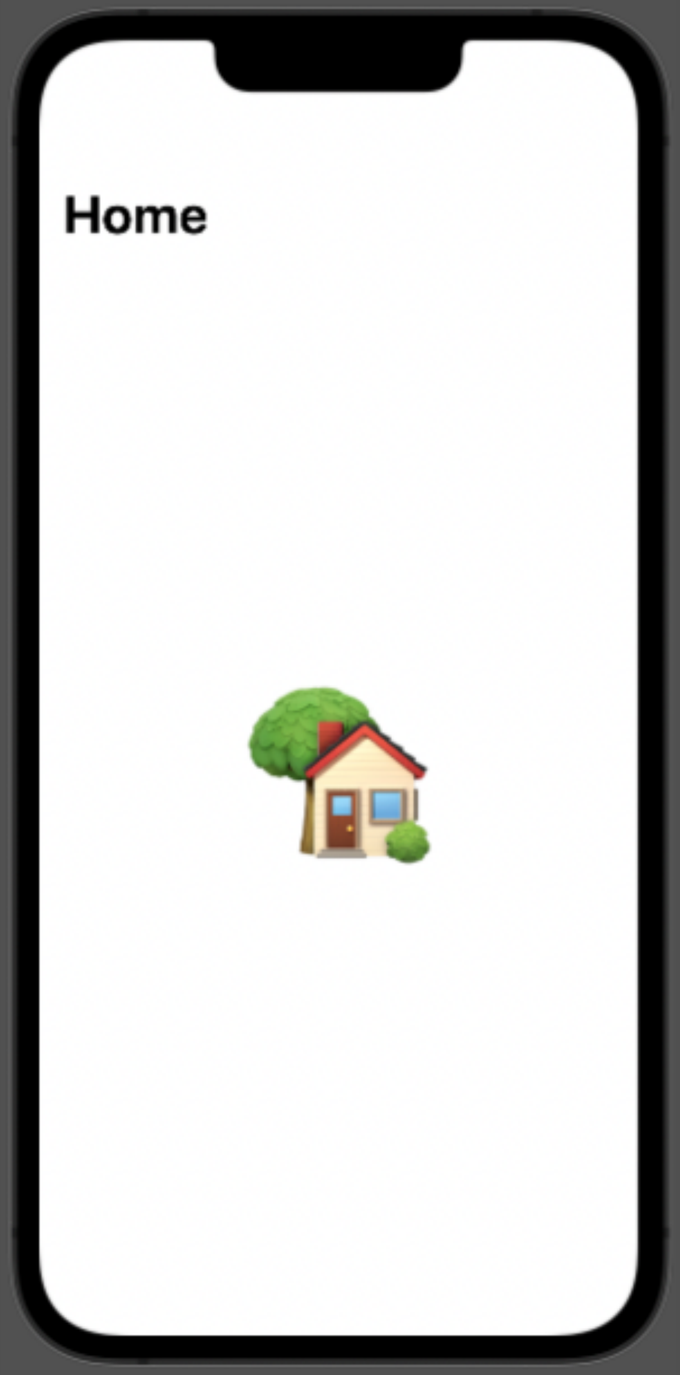
Preferences

```
struct ContentView: View {
    var body: some View {
        NavigationView {
            HomeView()
        }
    }
}

struct HomeView: View {
    var body: some View {
        Text("🏡")
            .navigationTitle("Home")
    }
}
```



Preferences ↑

```
struct ContentView: View {
    var body: some View {
        NavigationView {
            HomeView()
        }
    }
}

struct HomeView: View {
    var body: some View {
        Text("🏡")
            .preference(
                key: NavigationTitlePreferenceKey.self,
                value: "Home"
            )
    }
}
```



Preferences ↑

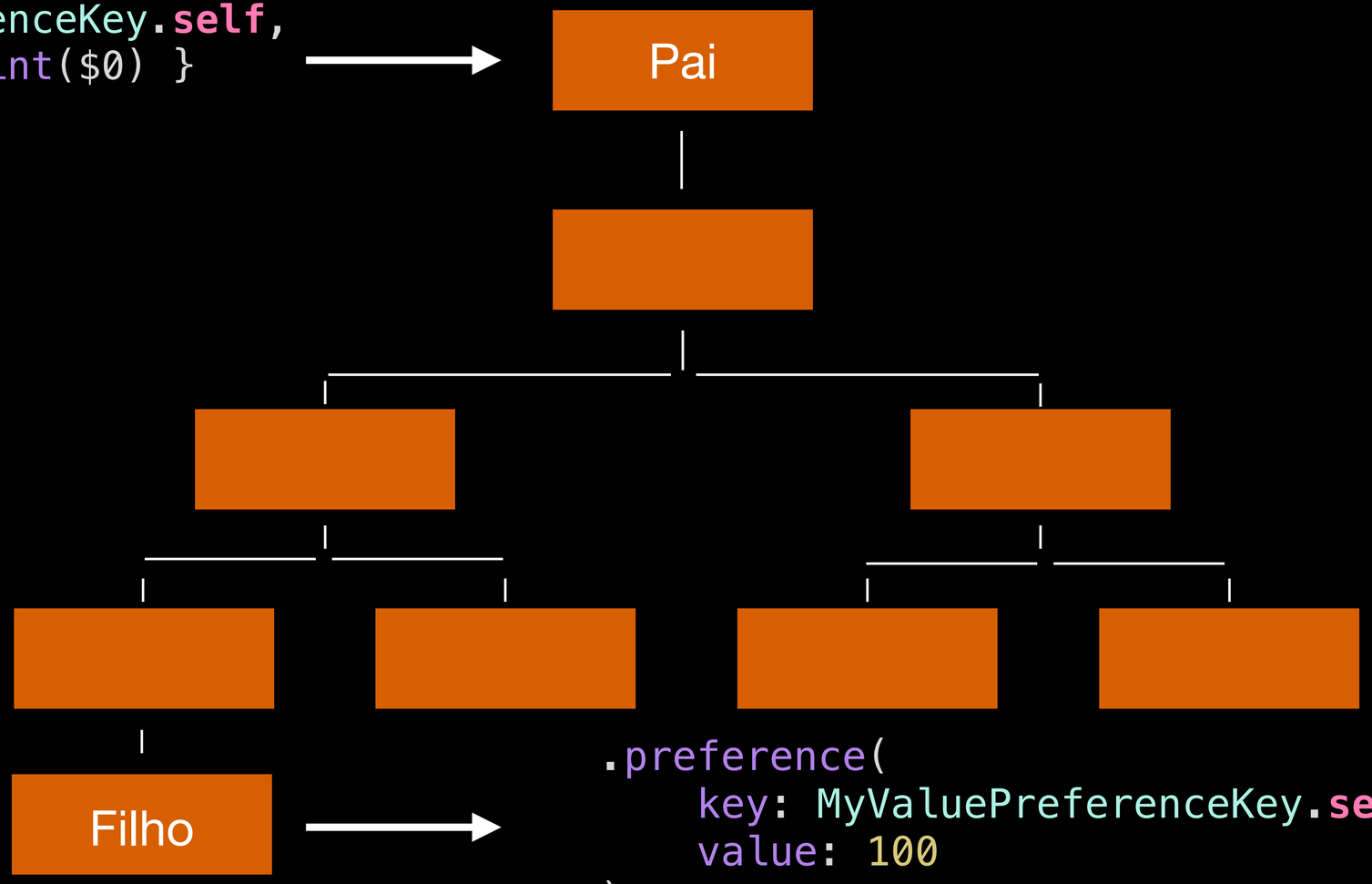
```
struct MyValuePreferenceKey: PreferenceKey {
    typealias Value = Int

    static var defaultValue: Value = .zero

    static func reduce(value: inout Int, nextValue: () -> Int) {
        value = nextValue()
    }
}
```

Preferences ↑

```
struct ContentView: View {
    @State var valorDaCasa: Int = .zero

    var body: some View {
        VStack {
            HomeView()
            Text("Valor da casa é: \(valorDaCasa)")
        }
        .onPreferenceChange(
            MyValuePreferenceKey.self,
            perform: { valorDaCasa = $0 }
        )
    }
}

struct HomeView: View {
    var body: some View {
        Text("🏡")
            .preference(
                key: MyValuePreferenceKey.self,
                value: 100
            )
    }
}
```

Preferences ↑

```
.onPreferenceChange(
    MyValuePreferenceKey.self,
    perform: { print($0) }
)
```

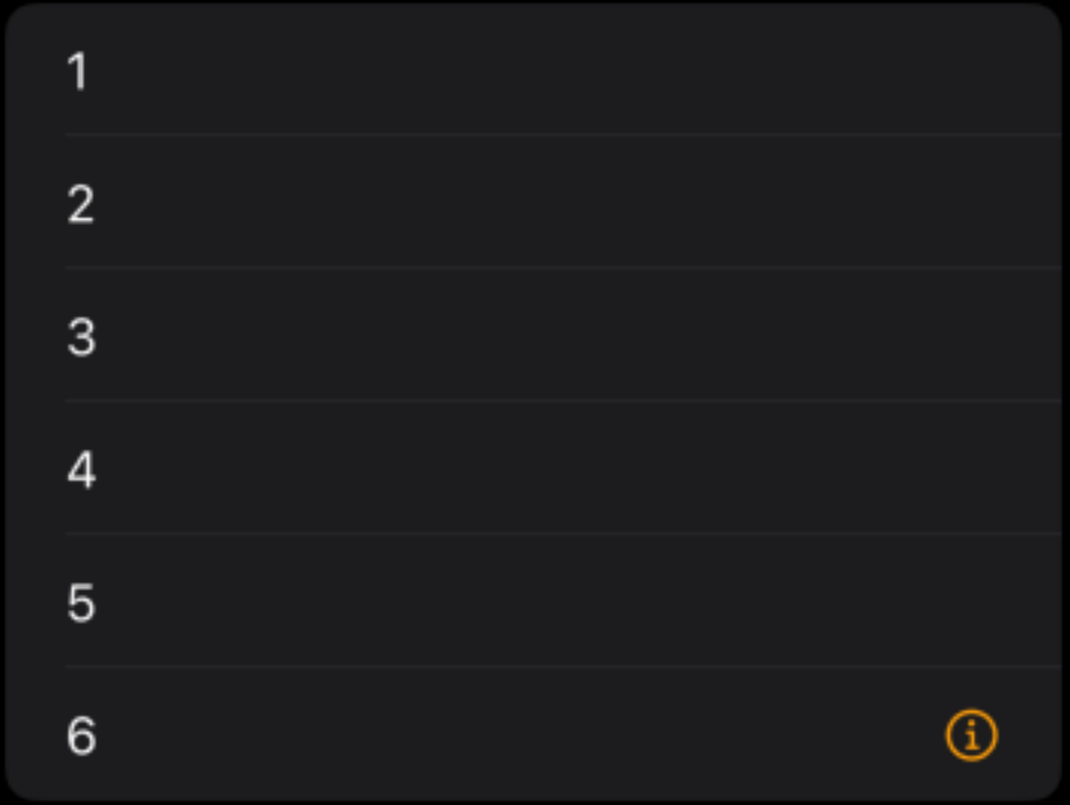



```
.preference(
    key: MyValuePreferenceKey.self,
    value: 100
)
```

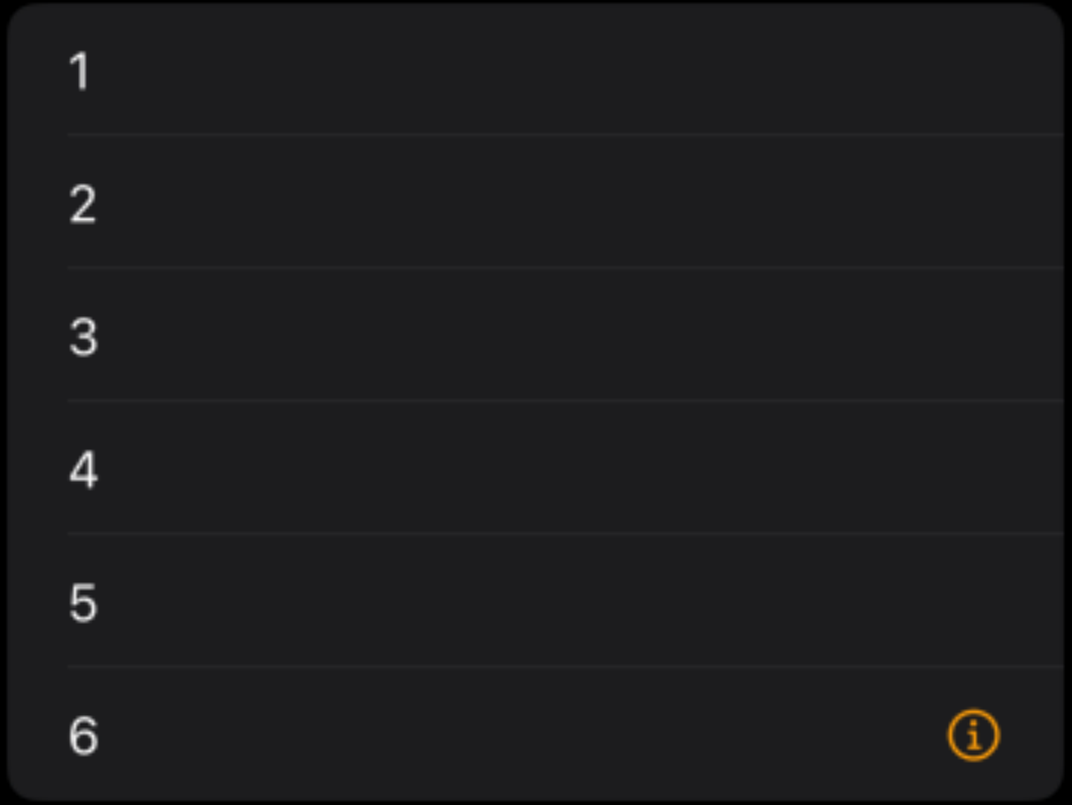
SwiftUI ViewModifier

“Encapsula modificações de estilo e comportamento, permitindo a reutilização dessas modificações.”

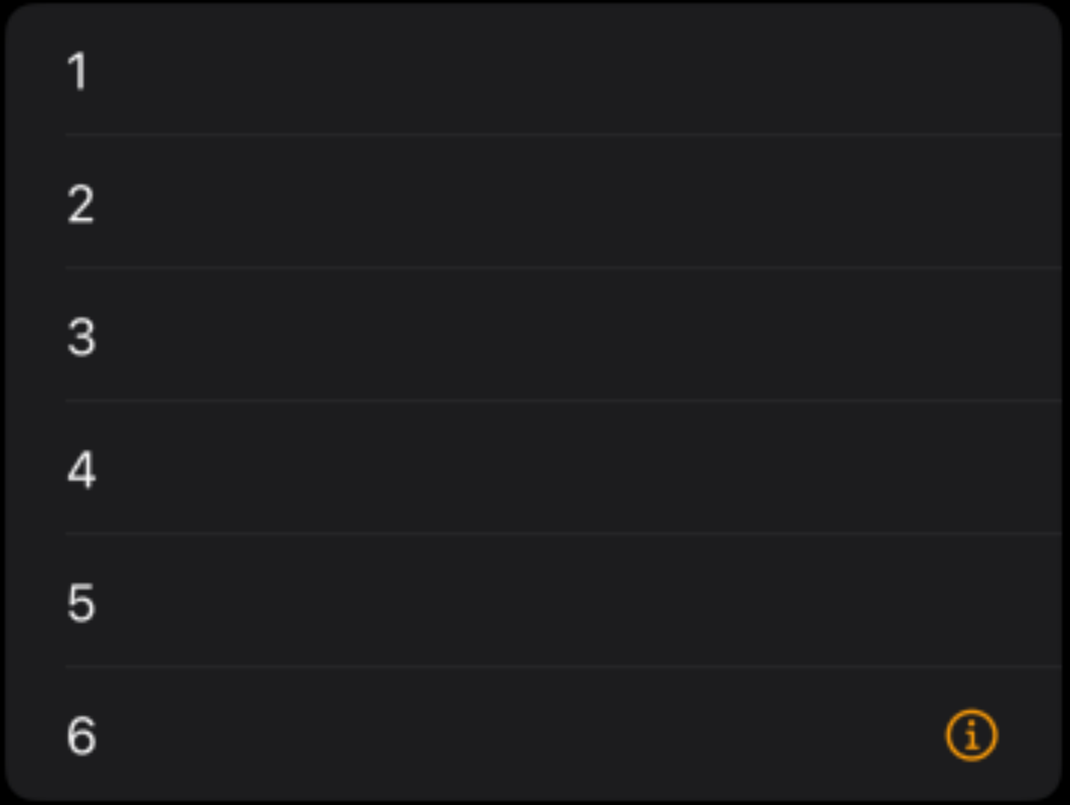
```swift
struct ContentView: View {
    var body: some View {
        List {
            Text("1")
            Text("2")
            Text("3")
            Text("4")
            Text("5")
            Text("6")
        }
    }
}
```



```swift
struct ContentView: View {
    var body: some View {
        List {
            Text("1")
            Text("2")
            Text("3")
            Text("4")
            Text("5")

            HStack {
                Text("6")
                Spacer()

                Button(
                    action: { /* Ação */ },
                    label: {
                        Image(systemName: "info.circle")
                            .foregroundColor(.orange)
                    }
                )
            }
        }
    }
}
```



```swift
struct ContentView: View {
    var body: some View {
        List {
            Text("1")
            Text("2")
            Text("3")
            Text("4")
            Text("5")
            Text("6")
                .info {
                    /* Action*/
                }
        }
    }
}
```



```
struct InfoModifier: ViewModifier {
    let action: () -> Void

    func body(content: Content) -> some View {
        HStack {
            content
            Spacer()

            Button(
                action: action,
                label: {
                    Image(systemName: "info.circle")
                        .foregroundColor(.orange)
                }
            )
        }
    }
}
```

Protocolo

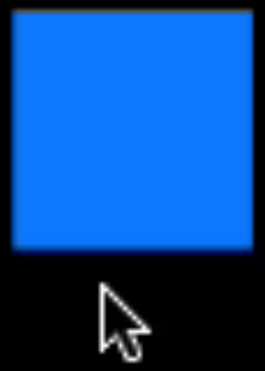
View pai

```swift
extension View {
    func info(_ action: @escaping () -> Void) -> some View {
        modifier(InfoModifier(action: action))
    }
}
```

```swift
struct ContentView: View {
    var body: some View {
        CardView()
            .info { /* Ação */ }
    }
}
```

SwiftUI Animação

```
struct ContentView: View {
    @State var isAnimating: Bool = false
    @State var offset: CGFloat = .zero

    var body: some View {
        Rectangle()
            .foregroundColor(.blue)
            .offset(y: offset)
            .animation(.spring(), value: offset)
            .onTapGesture {
                isAnimating.toggle()
                offset = isAnimating ? 120: 0
            }
    }
}
```



SwiftUI Backports

swipeActions(edge:allowsFullSwipe:content:)

Adds custom swipe actions to a row in a list.

iOS 15.0+ | iPadOS 15.0+ | macOS 12.0+ | Mac Catalyst 15.0+ | watchOS 8.0+ | visionOS 1.0+ Beta

Mas meu projeto é iOS 14 e agora? 😕

SwiftUI

🗑️ 🔥 🤢

SwiftUI Backports alternativas

6) Desistir

1) Criar o componente do zero no SwiftUI

2) Criar uma UIViewRepresentable

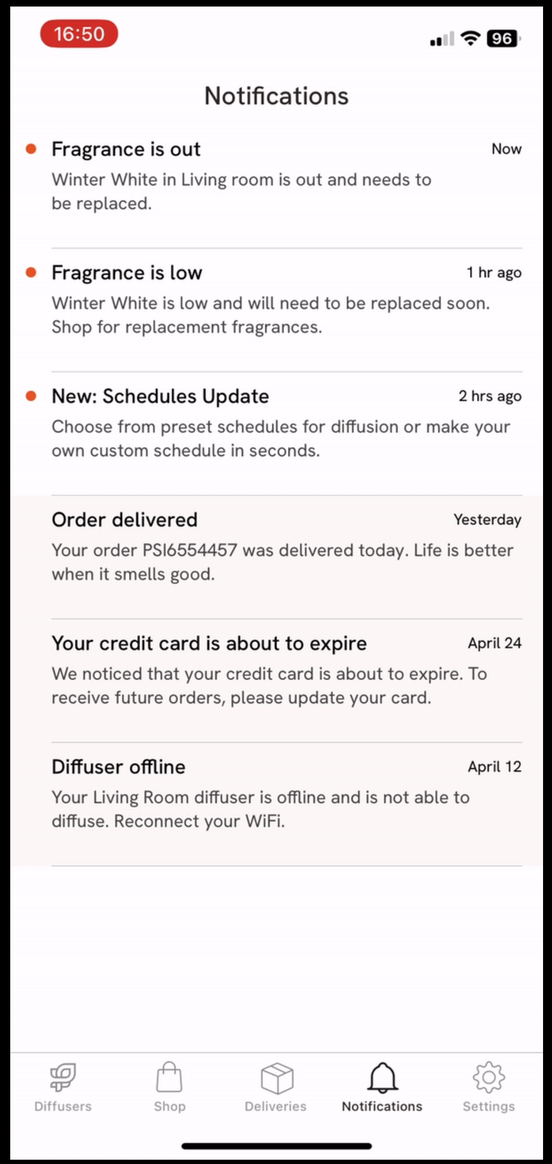
3) Acessar a UIView por baixo do SwiftUI

4) Usar o componente só para usuários iOS 15+

5) Negociar um novo design da feature

1) Criar o componente do zero no SwiftUI

```
extension View {
    func onSwipe(swipeRowID: Binding<UUID?>, slots: [Slot]) -> some View {
        modifier(SwipeAction(swipeRowID: swipeRowID, slots: slots))
    }
}

.onSwipe(
    swipeRowID: $swipeRowID,
    slots: [
        Slot(
            icon: Image(systemName: "checkmark"),
            text: "Mark read",
            action: { /* Action */ },
            backgroundColor: .blue
        ),
        Slot(
            icon: Image(systemName: "clock"),
            text: "Snooze",
            action: { /* Action */ },
            backgroundColor: .brown
        )
    ]
)
```



2) Criar uma UIViewRepresentable

```
struct ActivityIndicator: UIViewRepresentable {
    let style: UIActivityIndicatorView.Style

    func makeUIView(context: Context) -> some UIView {
        let view = UIActivityIndicatorView(style: style)
        view.color = .white
        view.startAnimating()
        return view
    }

    func updateUIView(_ uiView: UIViewType, context: Context) {
        //
    }
}
```

```
struct ContentView: View {
    var body: some View {
        ActivityIndicator(style: .large)
    }
}
```

3) Acessar a UIView por baixo do SwiftUI

refreshable(action:)

Marks this view as refreshable.

iOS 15.0+ | iPadOS 15.0+ | macOS 12.0+ | Mac Catalyst 15.0+ | tvOS 15.0+ | watchOS 8.0+ | visionOS 1.0+ Beta

E lá vamos nós de novo!

3) Acessar a UIView por baixo do SwiftUI

```
struct ContentView: View {
    @State var isRefreshing: Bool = false

    var body: some View {
        ScrollView {
            /* Views */
        }
        .introspectScrollView { scrollView in
            let refreshControl = MyCustomRefreshControl($isRefreshing)
            scrollView.refreshControl = refreshControl
        }
    }
}
```

3) Acessar a UIView por baixo do SwiftUI

The Power of the Hosting+Representable Combo

March 4, 2020 by javier

If you are allergic to hacks, you should probably stay away from the code in this article. However, if you continue, know that we will explore the powerful effects of combining Hosting Views with View Representables. Many times I found myself with a SwiftUI view and wishing I could access the AppKit/UIKit stuff behind it. … Read more

10 Comments

4) Usar o componente só para usuários iOS 15+

```
extension View {
    func swipeActions() -> some View {
        if #available(iOS 15.0, *) {
            return self.swipeActions {
                /* Implementação */
            }
        } else {
            return self
        }
    }
}
```

SwiftUI Backports alternativas

6) Desistir

1) Criar o componente do zero no SwiftUI

2) Criar uma UIViewRepresentable

3) Acessar a UIView por baixo do SwiftUI

4) Usar o componente só para usuários iOS 15+

5) Negociar um novo design da feature

SwiftUI Como usar no UIKit?

```
let hosting = UIHostingController(rootView: ContentView())

/// The root view of the SwiftUI view hierarchy managed by this view
/// controller.
hosting.rootView
```

SwiftUI

Em grande escala

SwiftUI

Quais apps já usam?

Books
Maps
Notes
Weather
Music
Podcasts

Disney+
Spotify
Adidas
Duolingo
OLX
Warren

Uma breve história

- Tudo começou em Janeiro | 2020
- Hype do lançamento
- 90% da base já usava iOS 13
- Criamos uma plataforma do zero
- Coragem
- Desenvolvimento exponencial

Swift Package Manager

Isolamos a parte nova

- Modularização
- Migração (Bridges)
- Integração nativa
- Documentação e suporte

Outras mudanças

- RxSwift -> Combine
- UIKit -> SwiftUI
- Coordinator -> Navegação (SwiftUI)
- MVVM -> The Composable Architecture

O mundo perfeito

- Combine
- SwiftUI
- Coordinator (UIKit)
- The Composable Architecture

Q&A

Não tenha medo de questionar!
As melhores respostas estão escondidas nas perguntas que você faz