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

Visão geral

- 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**
- **Controle manual**
- **Mais complexo**

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

Imperativo

Código

```
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**
- **Fácil de entender**

“Crie um botão”

Declarativo

Código

```
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

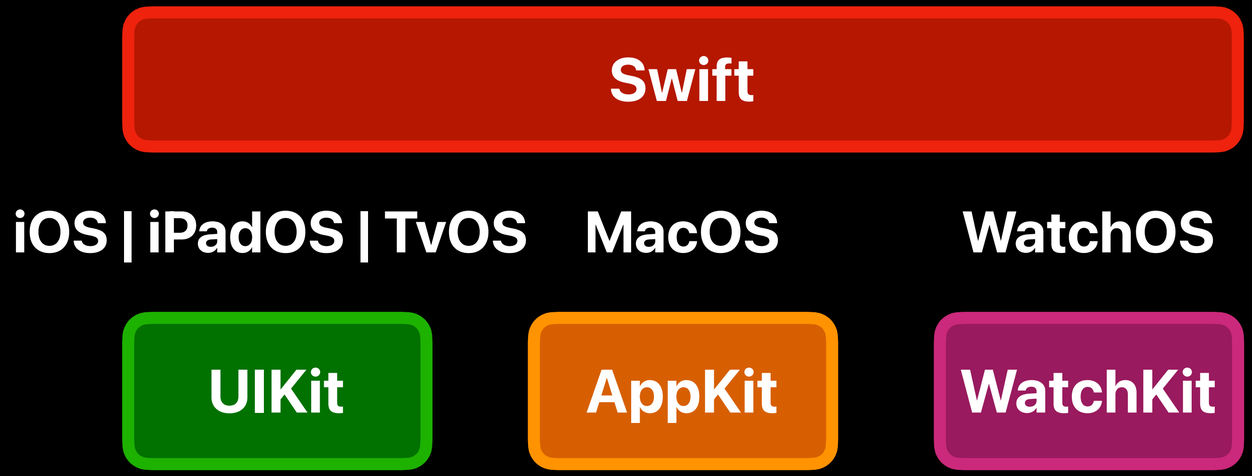
Multiplataforma

SwiftUI

Multiplataforma

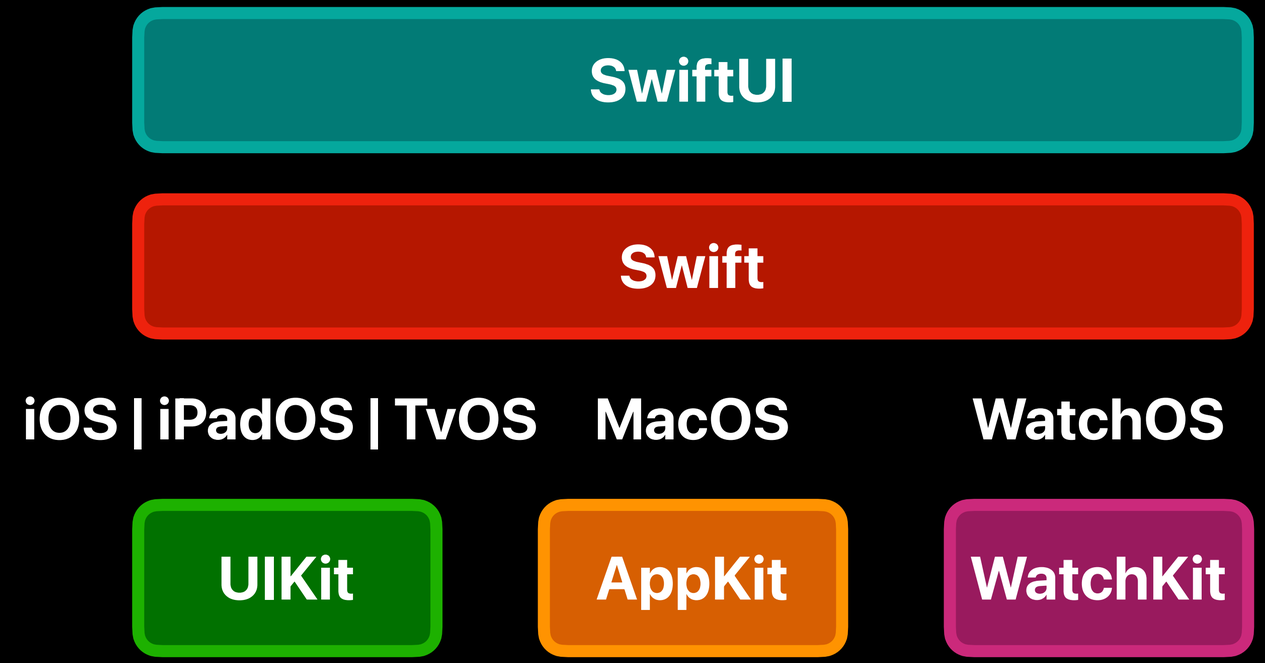
SwiftUI

Multiplataforma



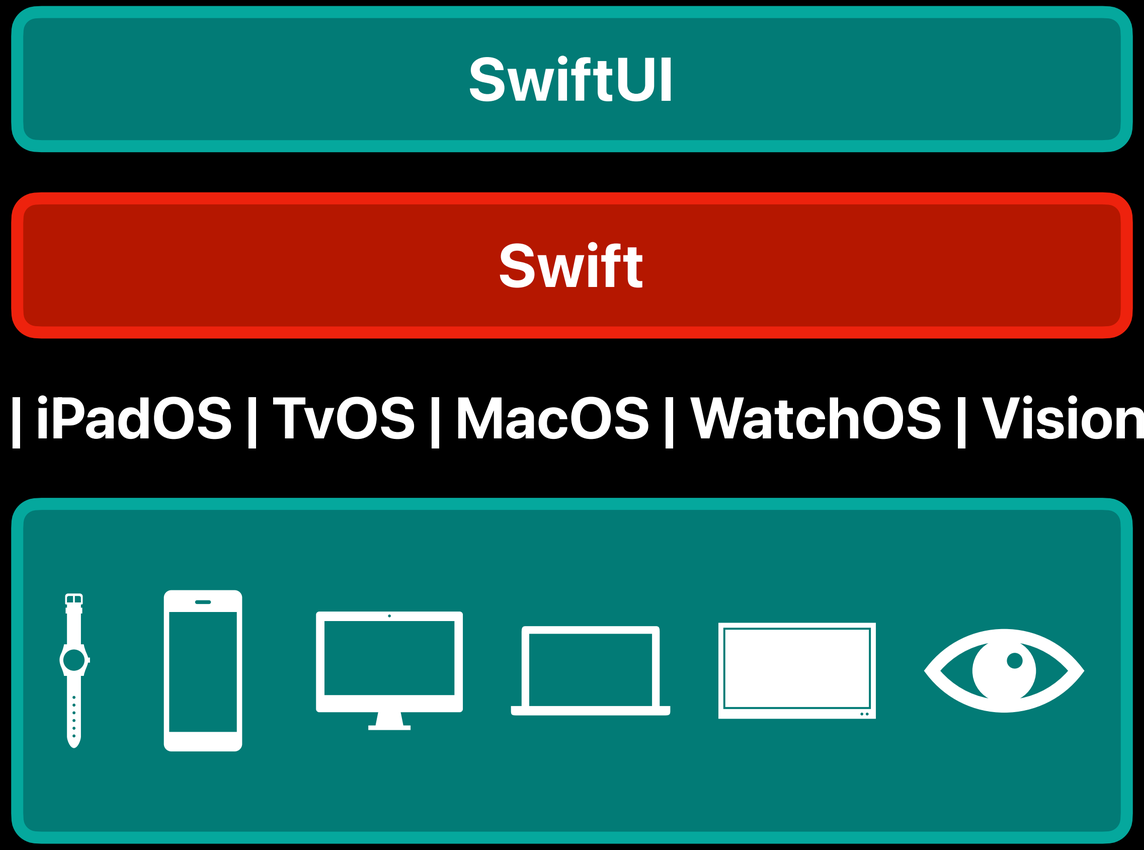
SwiftUI

Multiplataforma



SwiftUI

Multiplataforma



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**
- **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

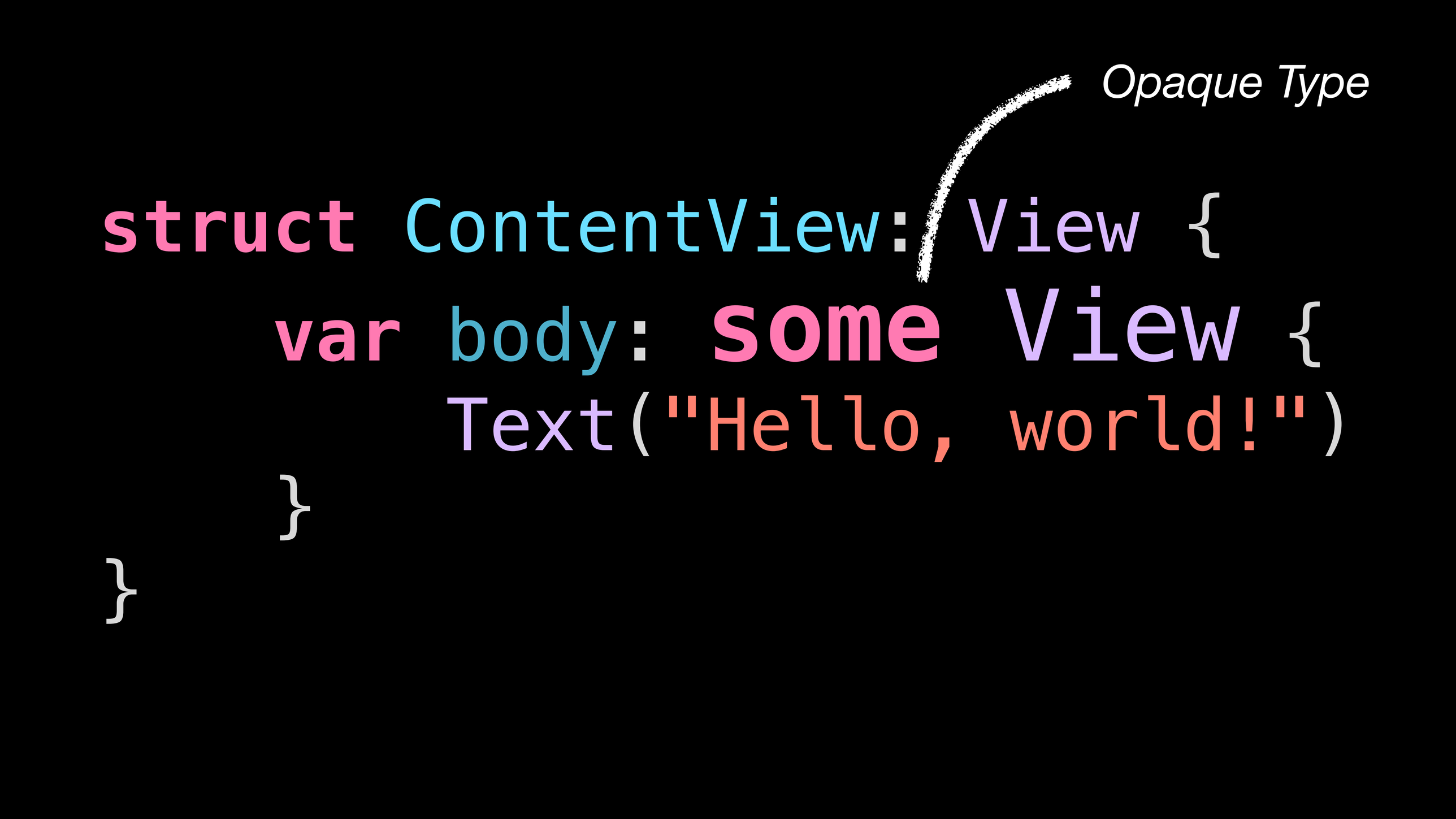
```
struct ContentView: View {
    var body: some View {
        Text("Hello, world!")
    }
}
```


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

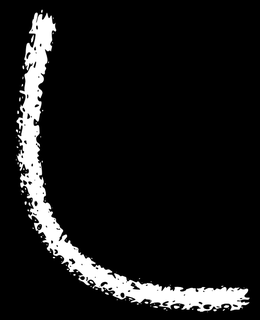

```
struct ContentView: View {
    var body: some View {
        Text("Hello, world!")
    }
}
```

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

Computed Property

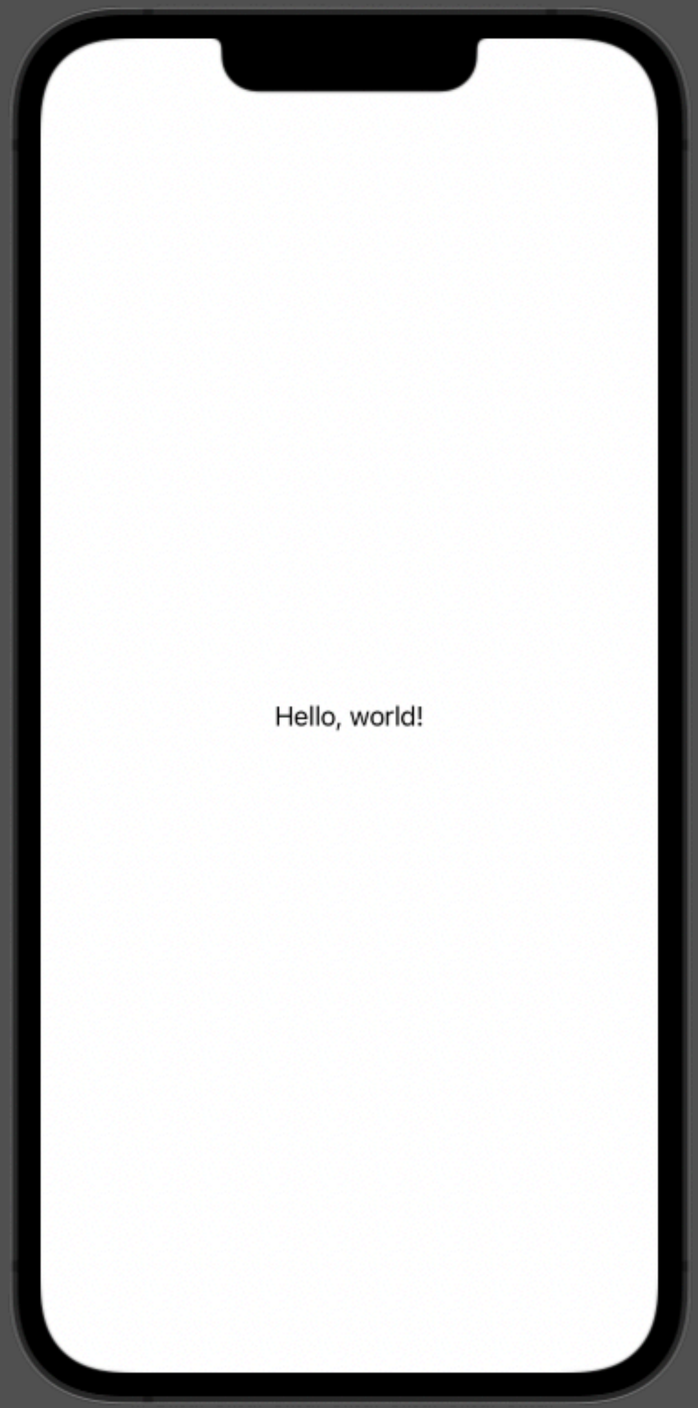

Opaque Type
struct ContentView: View {
var body: some View {
Text("Hello, world!")
}
}

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



View

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



SwiftUI Layout Process

"Layout é decidir o tamanho das coisas na tela"

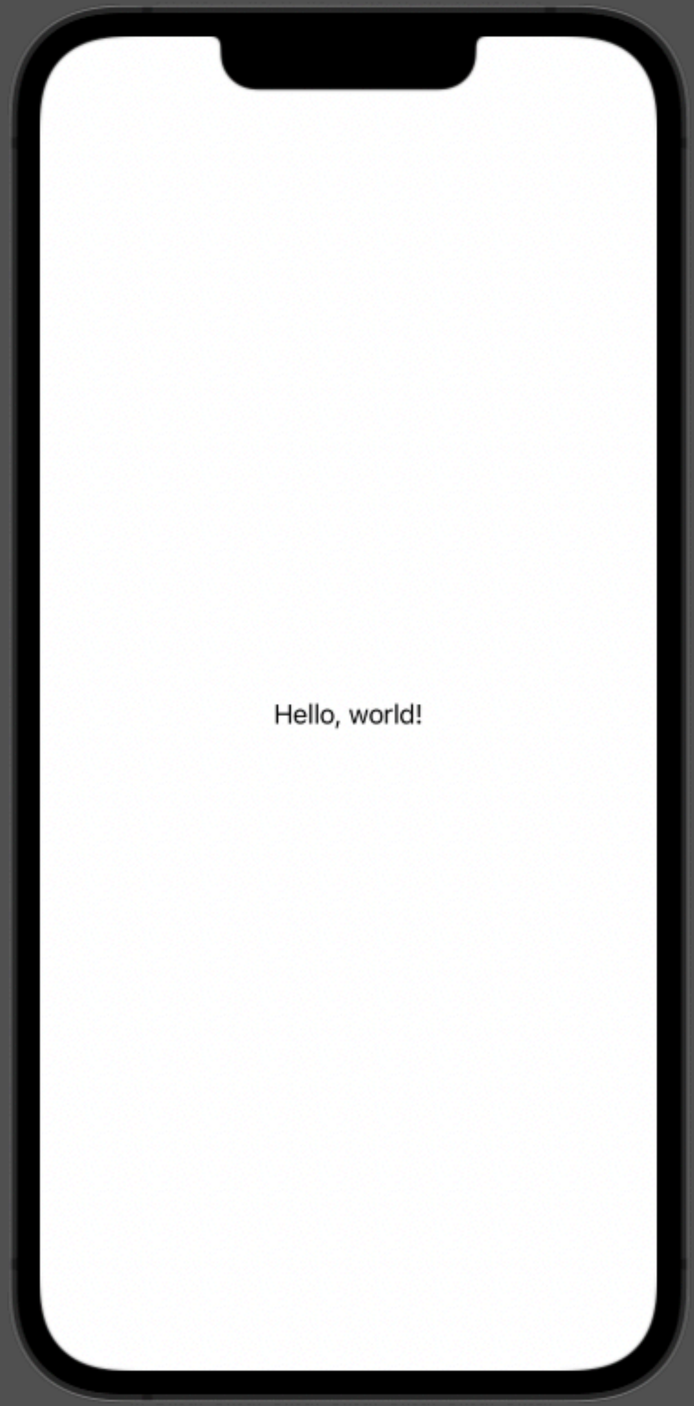
SwiftUI Layout Process

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

RootView

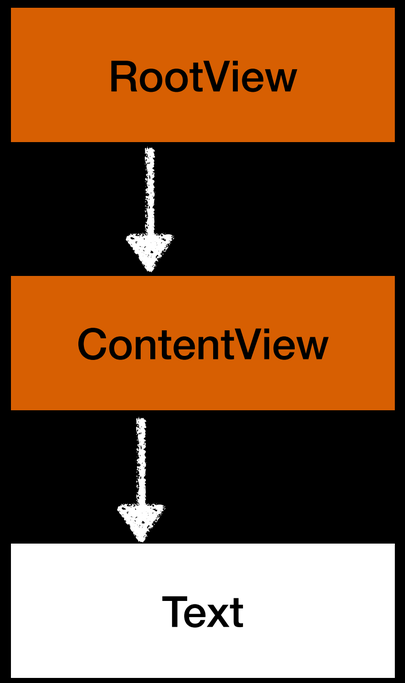
ContentView

Text



SwiftUI Layout Process

1. O pai propõe um tamanho para o filho

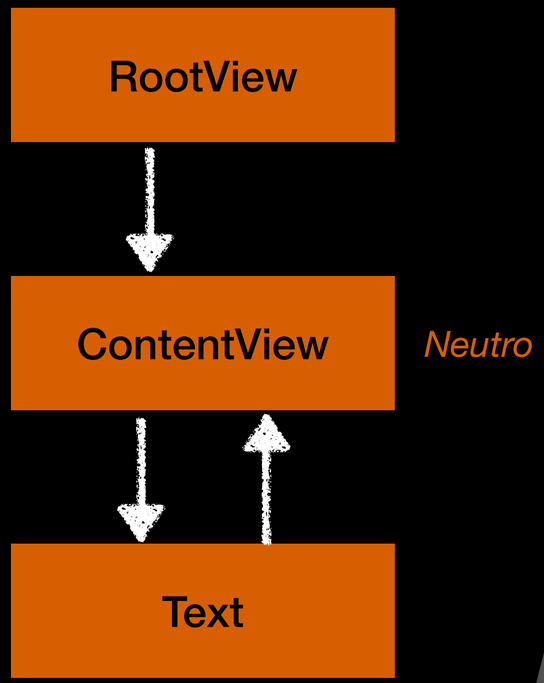


Neutro

SwiftUI Layout Process

1. O pai propõe um tamanho para o filho

2. O filho escolhe seu próprio tamanho



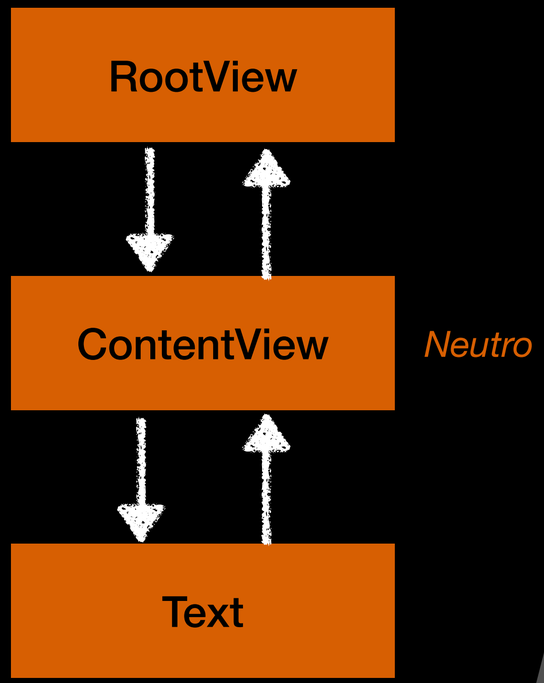
Neutro

SwiftUI Layout Process

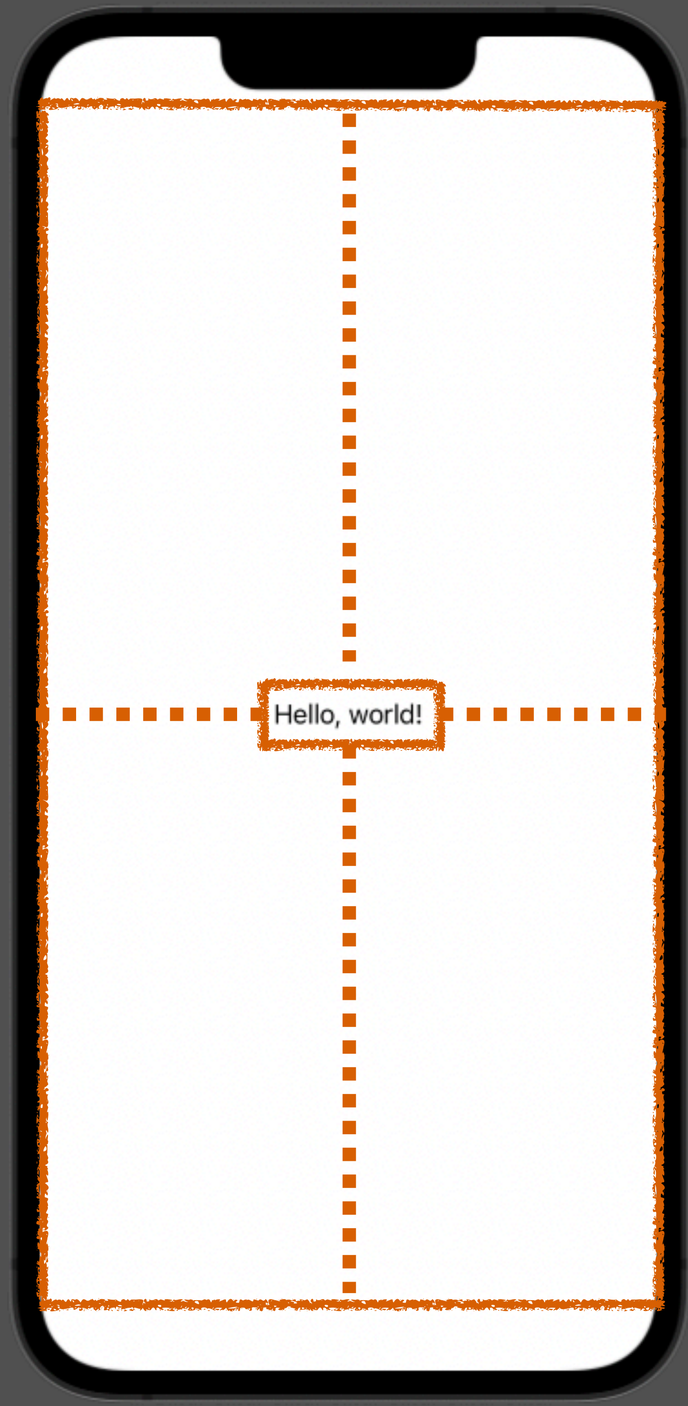
1. O pai propõe um tamanho para o filho

2. O filho escolhe seu próprio tamanho

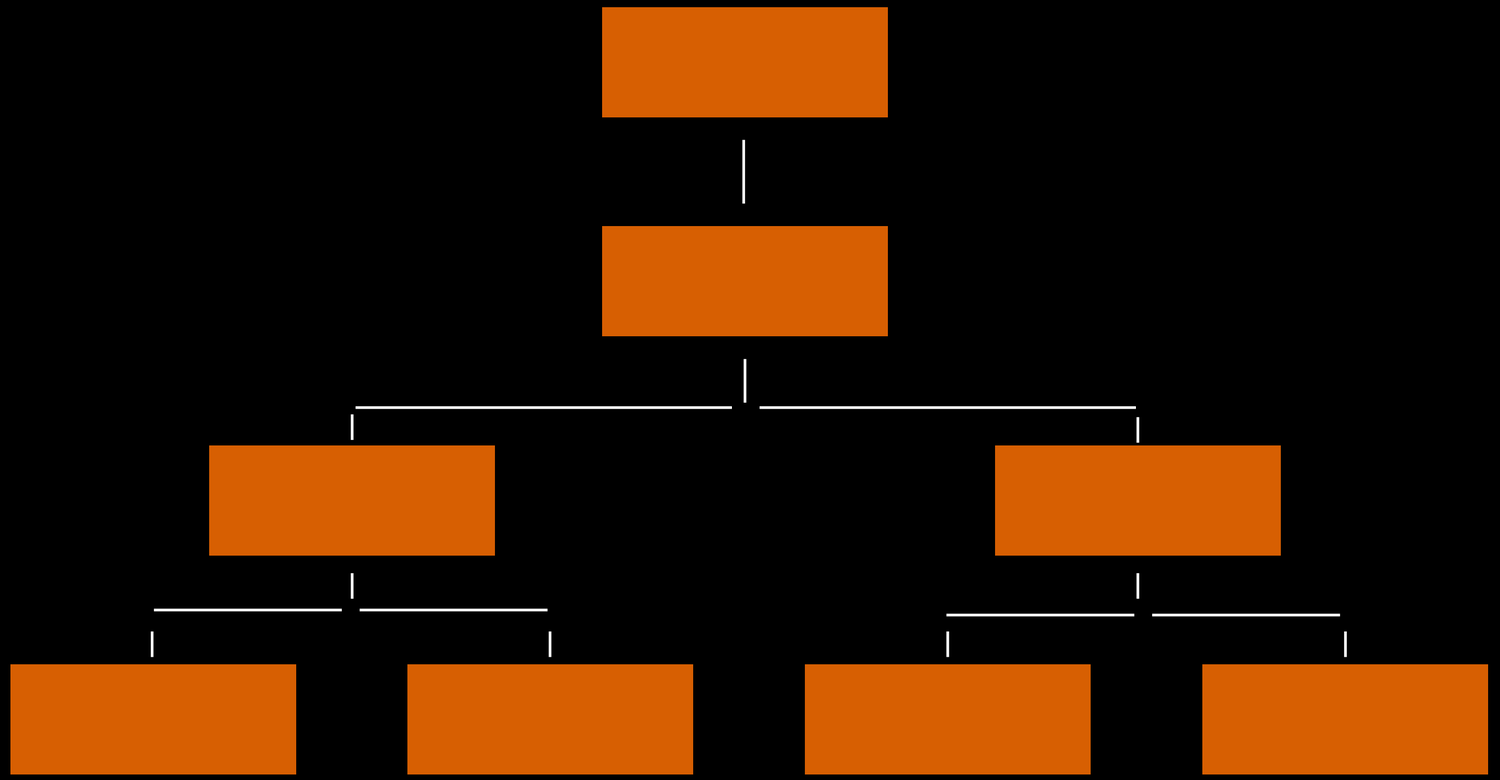
3. O pai posiciona o filho na coordenada do pai



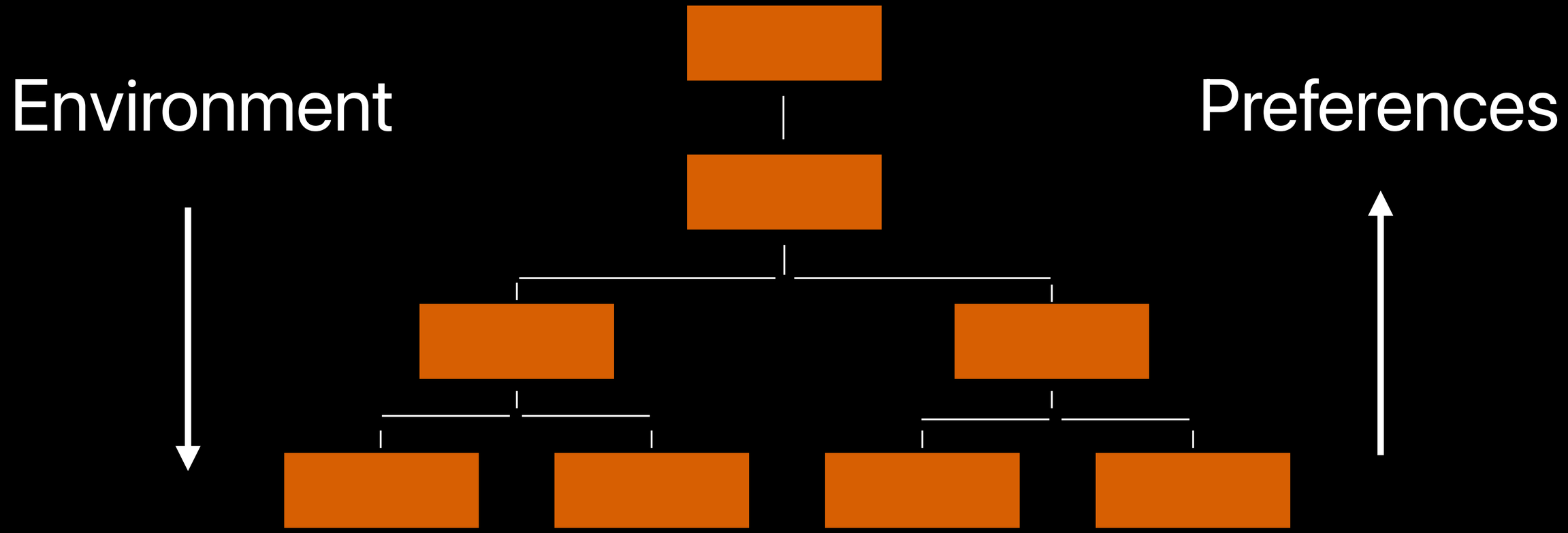
Neutro



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 ↓

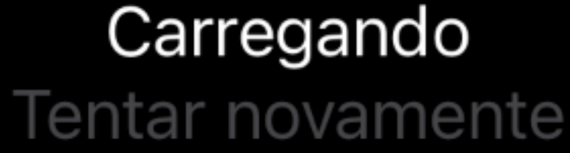
```
.colorScheme
.layoutDirection
.locale
.sizeCategory
.accessibilityDifferentiateWithoutColor
.accessibilityReduceTransparency
.accessibilityReduceMotion
.multilineTextAlignment
.lineSpacing
.truncationMode
.minimumScaleFactor
.isEnabled
.presentationMode
.managedObjectContext
.undoManager
.horizontalSizeClass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

Environment ↓

```
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)
        }
    }
}
```

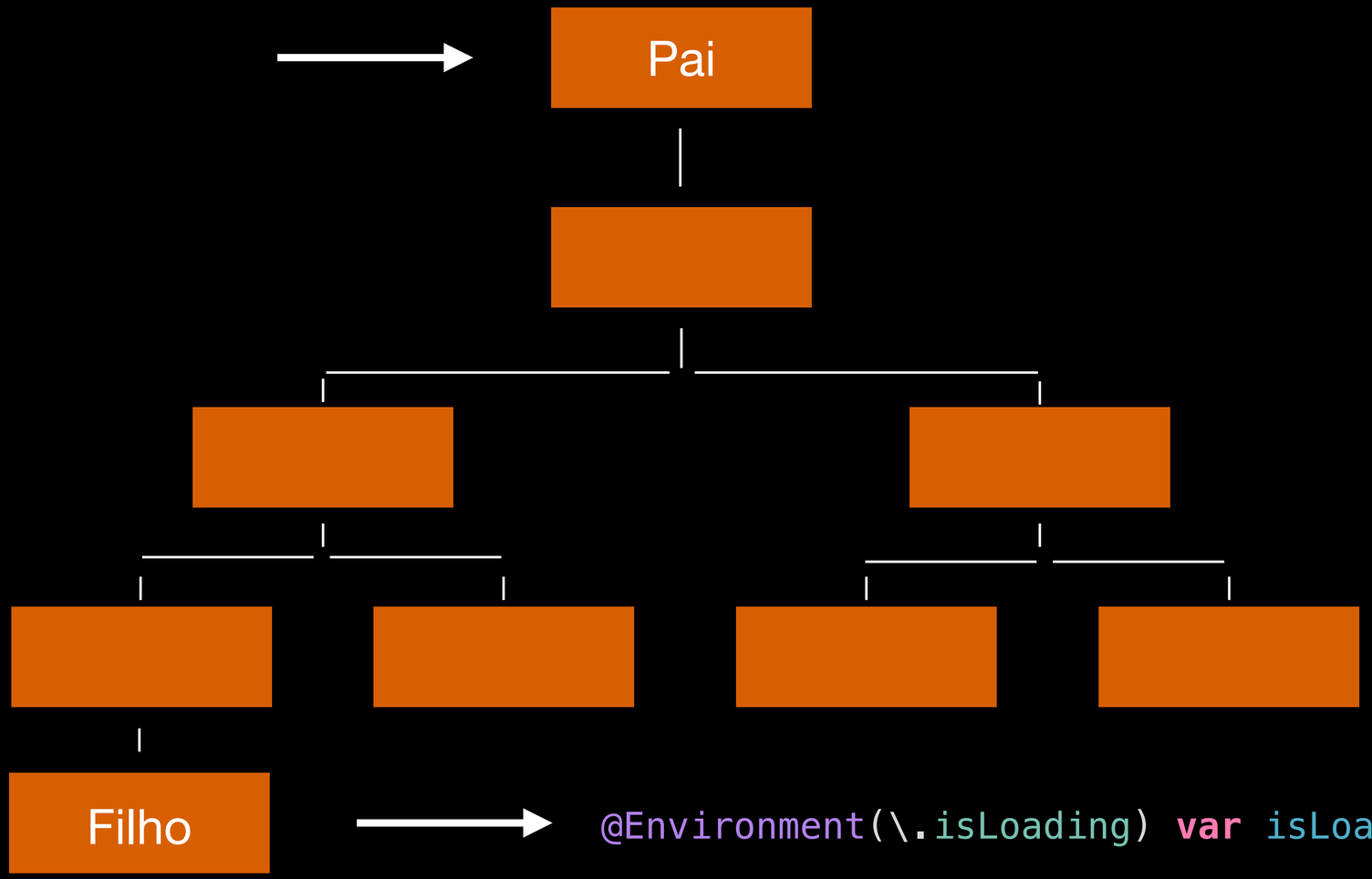


Environment ↓

```
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)
        }
    }
}
```

Environment ↓

```
Component().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 ↓

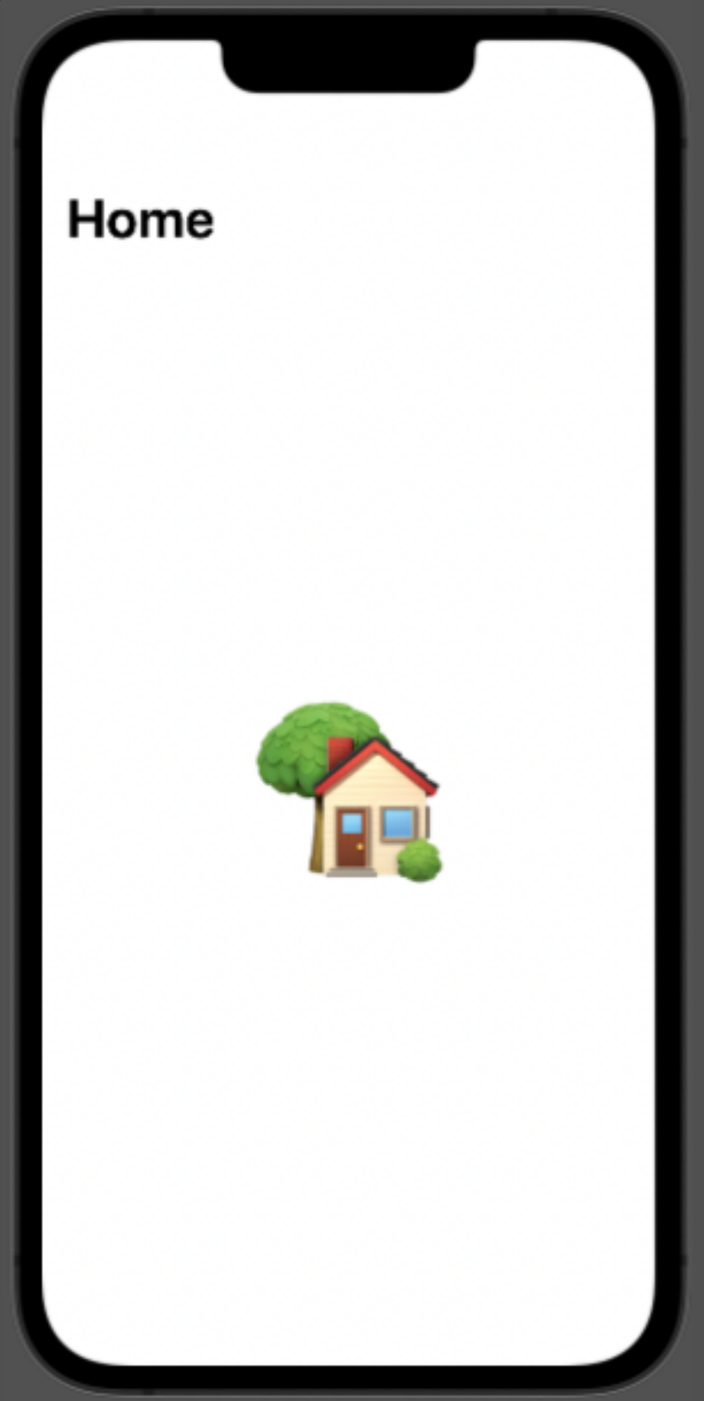
```
struct ContentView: View {
    var body: some View {
        Button(
            action: { /* Ação */ },
            label: { Text("Botão") }
        )
        .disabled(true)
    }
}

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

Preferences 

Mecanismo para uma view expressar suas preferencias para seus ascendentes

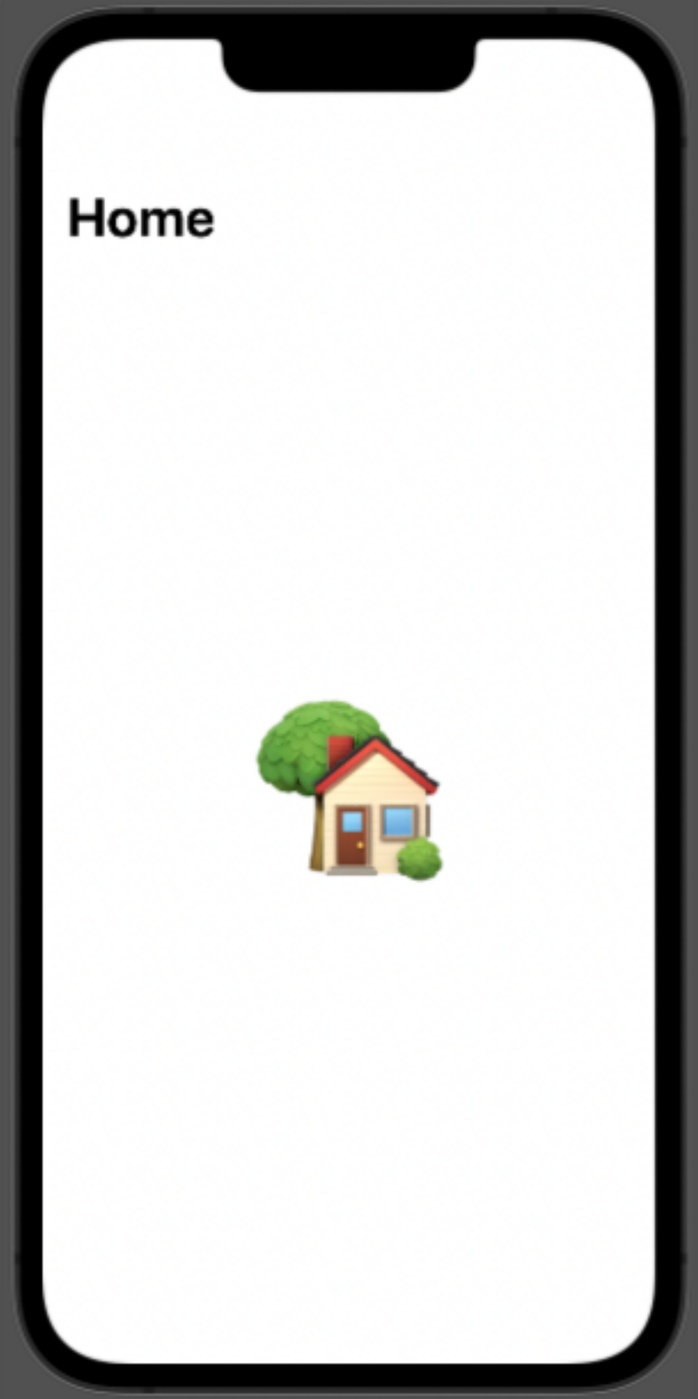
Preferences ↑

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

struct HomeView: View {
    var body: some View {
        Text("🏠")
            .font(.system(size: 120))
            .navigationTitle("Home")
    }
}
```



Preferences ↑

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

struct HomeView: View {
    var body: some View {
        Text("🏡")
            .font(.system(size: 120))
            .preference(
                key: NavigationTitlePreferenceKey.self,
                value: "Home"
            )
    }
}
```



Preferences 

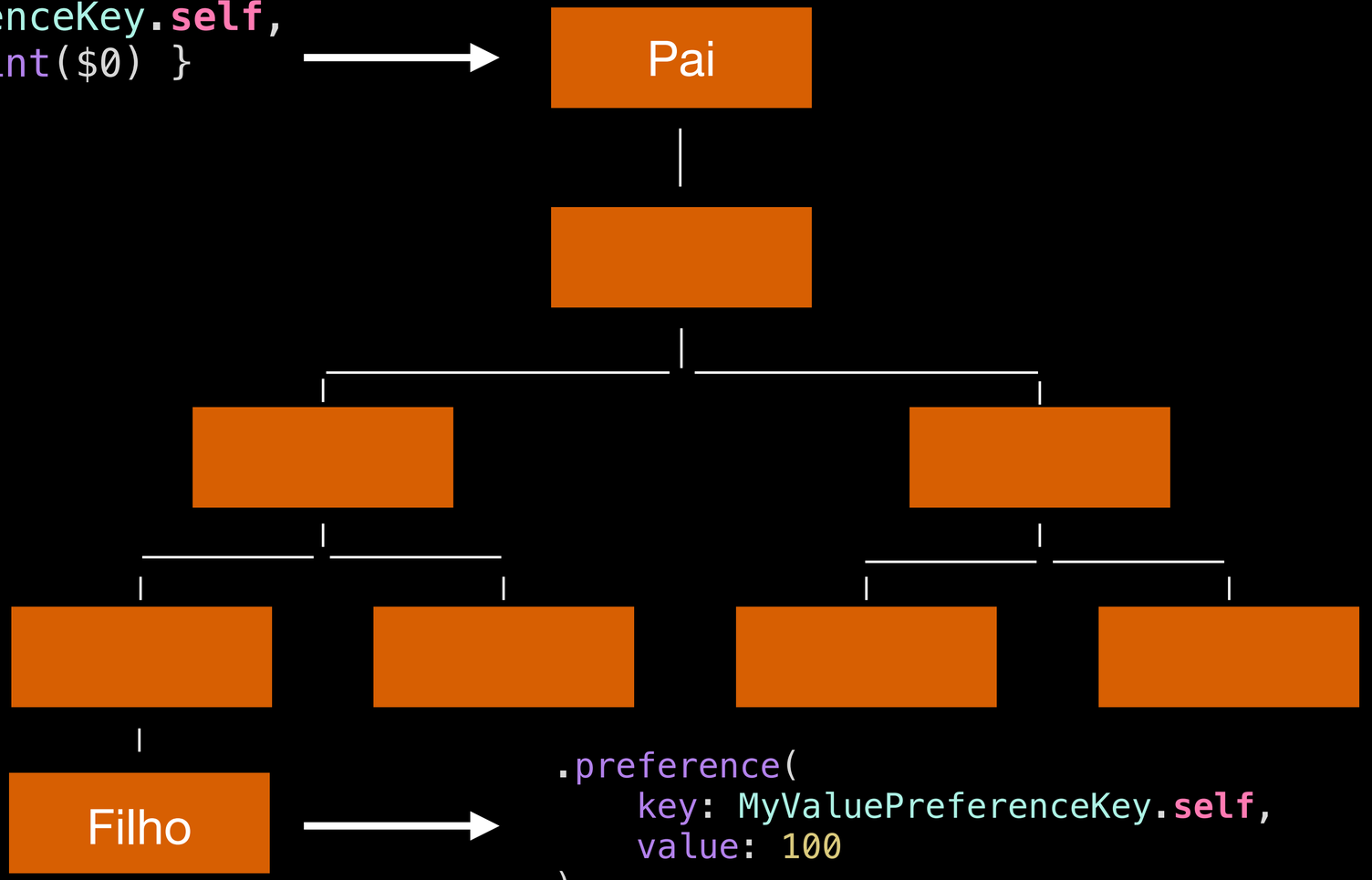
```
struct MyValuePreferenceKey: PreferenceKey {
    typealias Value = Int

    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

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

Swift Package Manager

Isolamos a parte nova

- Modularização
- Migração (Bridges)
- Acabou os conflitos
- Integração nativa
- Documentação e suporte

Outras mudanças

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