QUEM BOTOU SWIFT NA MINHA UI?

Txai Wieser

Desenvolvedor iOS desde 2013

Comunidades:

- iOSDevBR
- CocoaHeads Porto Alegre
- TDC Porto Alegre

Experiências:

- Ciência de Computação na UFRGS
- Developer Academy
- Warren Brasil
- Apple CA

Entusiasta de novas tecnologias.

SWIFT + UI

SWIFTUI

O **menor** caminho para construir **ótimos** apps para **todos** dispositivos.

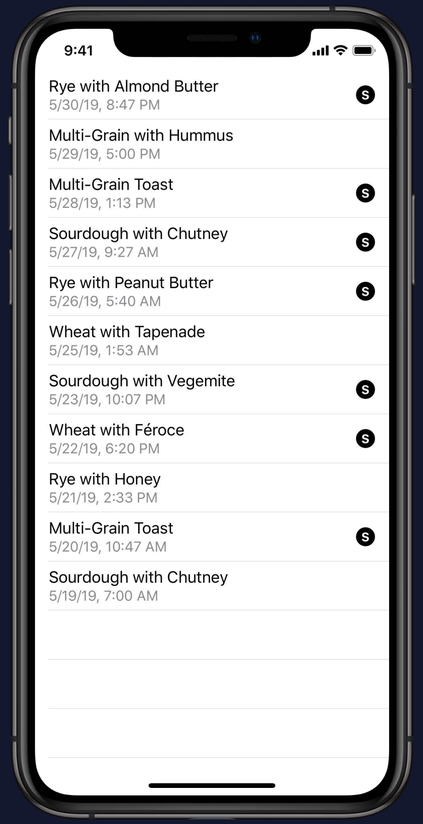
- Apple

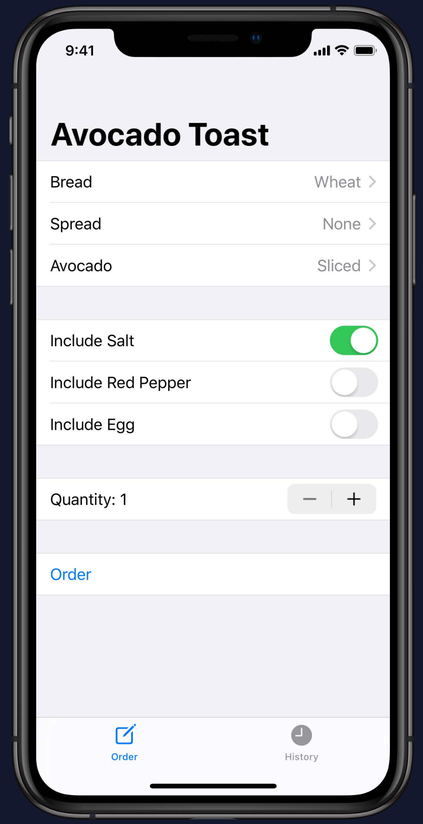
SWIFTUI

- Uma nova maneira de pensar
- Framework de UI Declarativa (e não Imperativa)
- Foco nas funcionalidades da UI (e não nas especificidades do layout)

```
Form {
    Section {
        TextField(
            "Title",
            text: $event.title
        )
        TextField(
            "Location",
            text: $event.location
        )
    }

    Section {
        Toggle(
            "All-Day",
            isOn: $event.isAllDay
        )
        DatePicker(
            "Starts",
            selection: $event.starts
        )
    }
}
```

```
OrderHistory : View {
previousOrders: [CompletedOrder]

body: some View {
ist(previousOrders) { order in
  HStack {
    VStack(alignment: .leading) {
      Text(order.summary)
      Text(order.purchaseDate)
        .font(.subheadline)
        .foregroundColor(.secondary)
    }
    Spacer()
    if order.includeSalt {
      SaltIcon()
    }
  }
```






```
struct ContentView : View {
    var body: some View {
        NavigationView {
            TabbedView {
                OrderForm()
                    .tabItemLabel {
                        Image(systemName: "squa
                        Text("New Order")
                    }
                OrderHistory()
                    .tabItemLabel {
                        Image(systemName: "cloc
                        Text("History")
                    }
            }
        }
    }
}
```


Global
HD

ALEM DO BASICO

- Layout System
- Geometry Reader
- Environment
- Preferences

LAYOUT SYSTEM

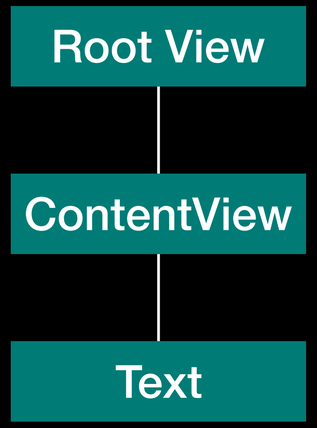
Layout é decidir o **tamanho** das coisas na **tela**

LAYOUT SYSTEM

```
struct ContentView: View {

    var body: some View {
        Text("Olá, Mundo!")
   }
}
```

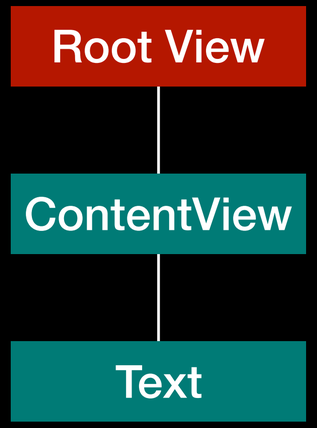
LAYOUT SYSTEM

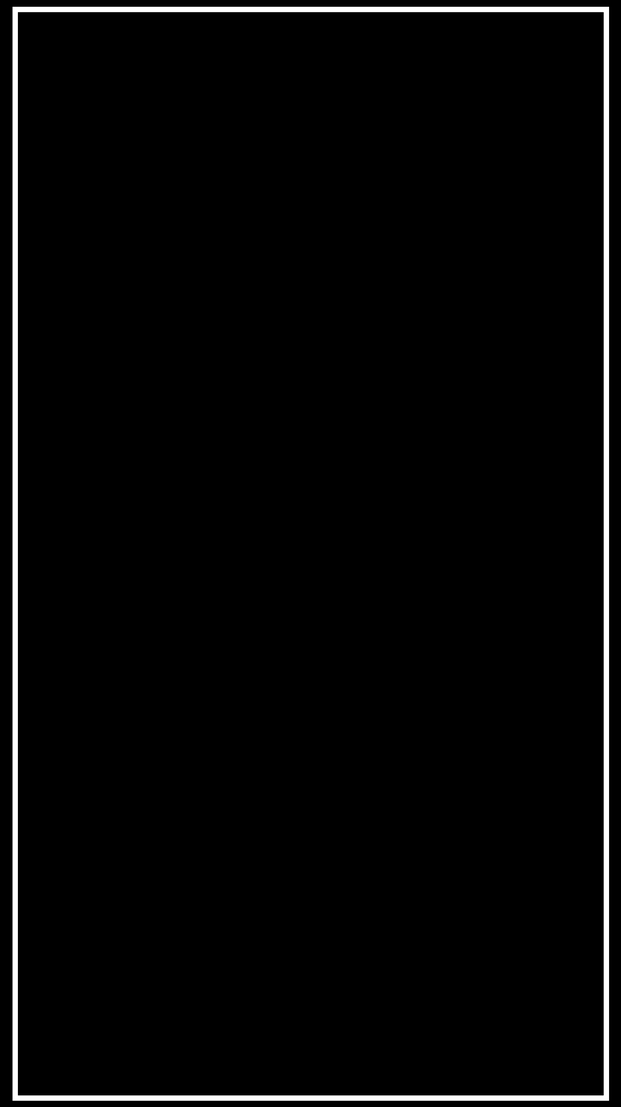
```
struct ContentView: View {

    var body: some View {
        Text("Olá, Mundo!")
    }
}
```



LAYOUT SYSTEM

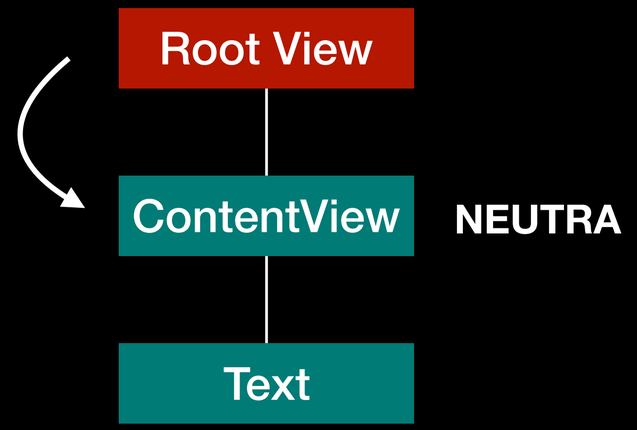
```
struct ContentView: View {

    var body: some View {
        Text("Olá, Mundo!")
    }
}
```

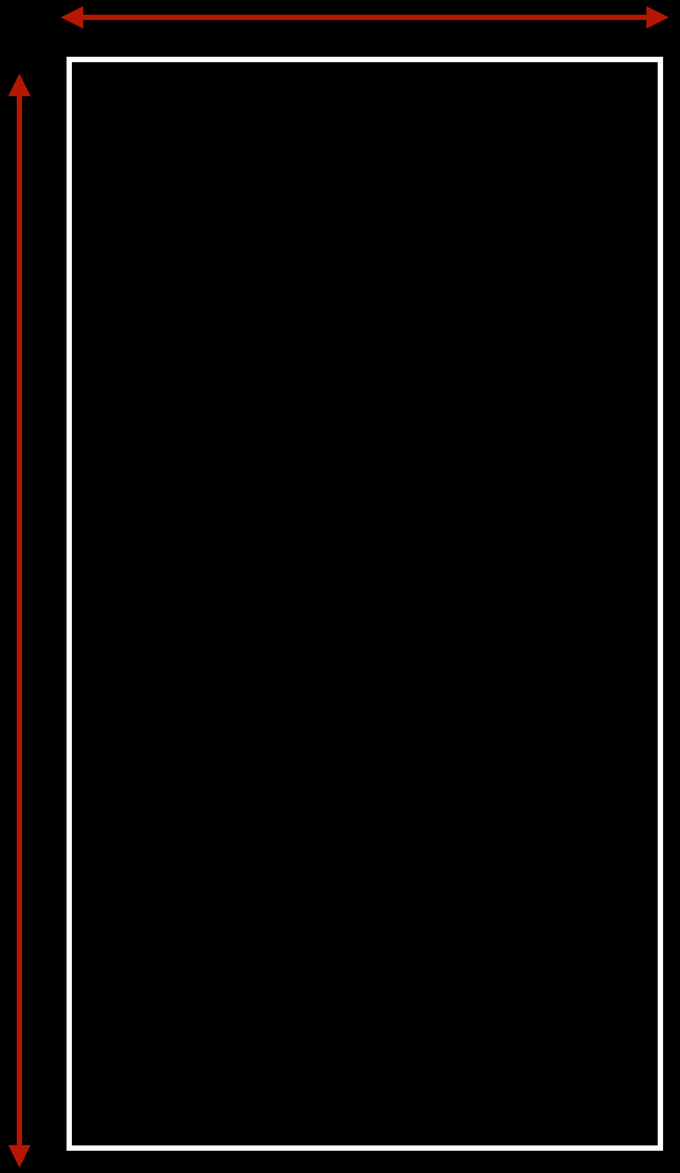




LAYOUT SYSTEM

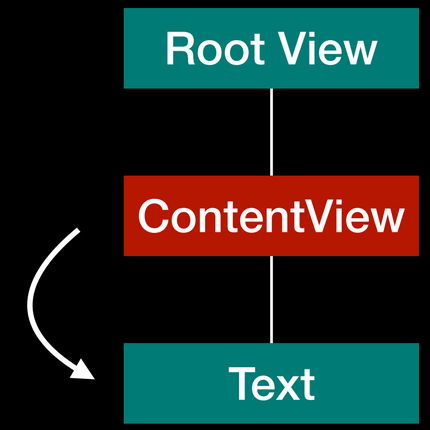
```
struct ContentView: View {

    var body: some View {
        Text("Olá, Mundo!")
    }
}
```

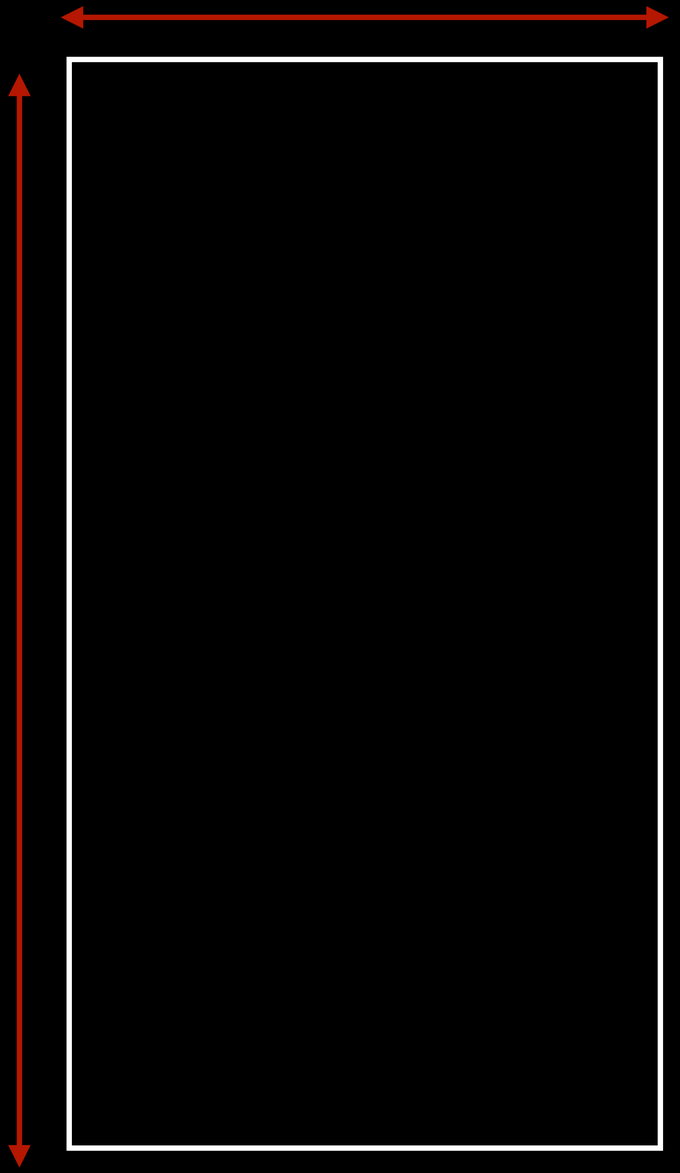




LAYOUT SYSTEM

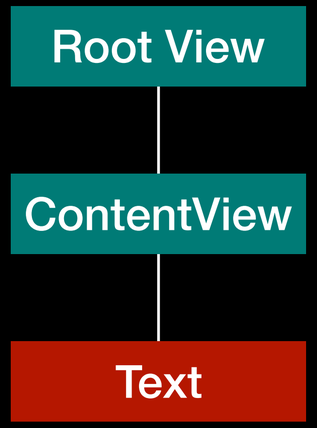
```
struct ContentView: View {

    var body: some View {
        Text("Olá, Mundo!")
    }
}
```





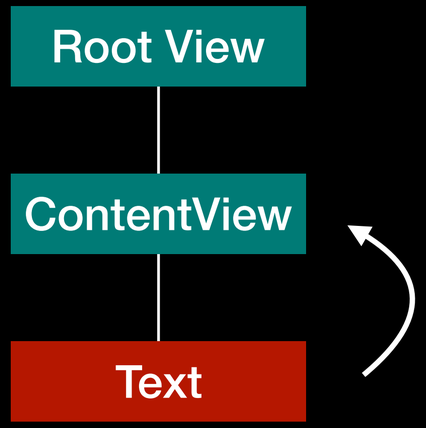
LAYOUT SYSTEM

```
struct ContentView: View {

    var body: some View {
        Text("Olá, Mundo!")
    }
}
```



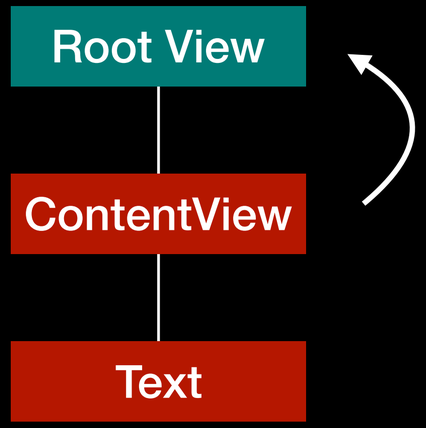
LAYOUT SYSTEM

```
struct ContentView: View {

    var body: some View {
        Text("Olá, Mundo!")
    }
}
```



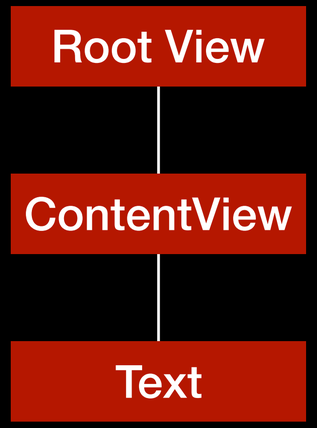
LAYOUT SYSTEM

```
struct ContentView: View {

    var body: some View {
        Text("Olá, Mundo!")
    }
}
```



LAYOUT SYSTEM

```
struct ContentView: View {

    var body: some View {
        Text("Olá, Mundo!")
    }
}
```



LAYOUT SYSTEM

1. Pai propõem um tamanho ao filho
2. Filho escolhe seu próprio tamanho
3. Pai posiciona filho na coordenada do pai

LAYOUT SYSTEM

1. Pai propõem um tamanho ao filho
2. Filho escolhe seu próprio tamanho
3. Pai posiciona filho na coordenada do pai

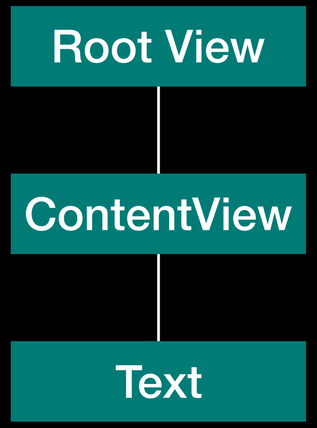
Não existe maneira de **forçar** um tamanho no seu **filho.**

Não existe maneira de **forçar** um tamanho no seu **filho.**

Pense em guias, sugestões para os filhos

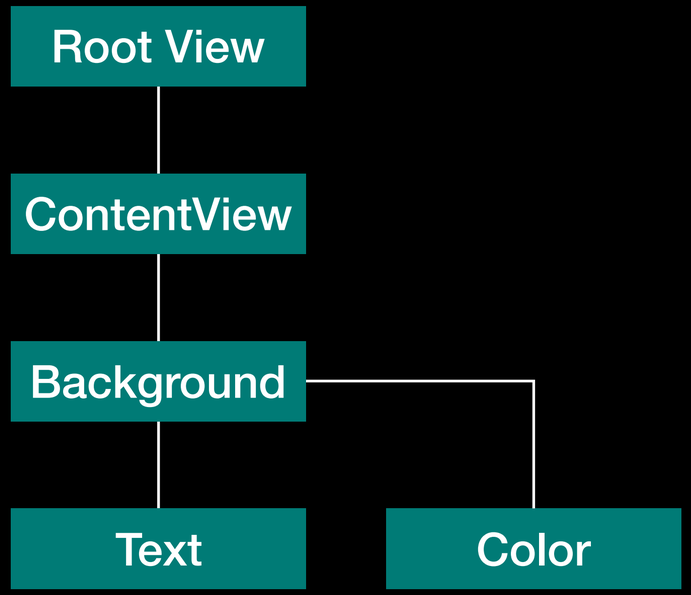
LAYOUT SYSTEM

```
struct ContentView: View {

    var body: some View {
        Text("Olá, Mundo!")
    }
}
```



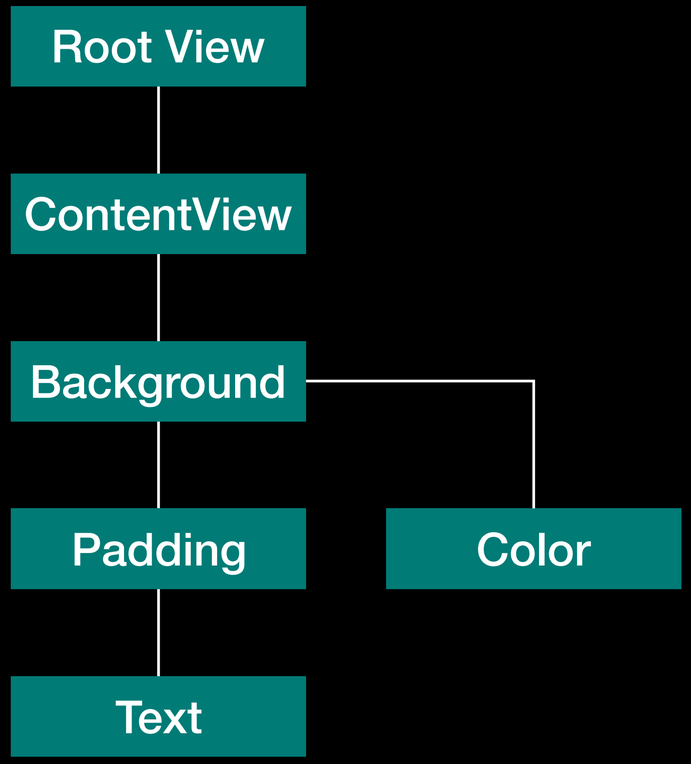
LAYOUT SYSTEM

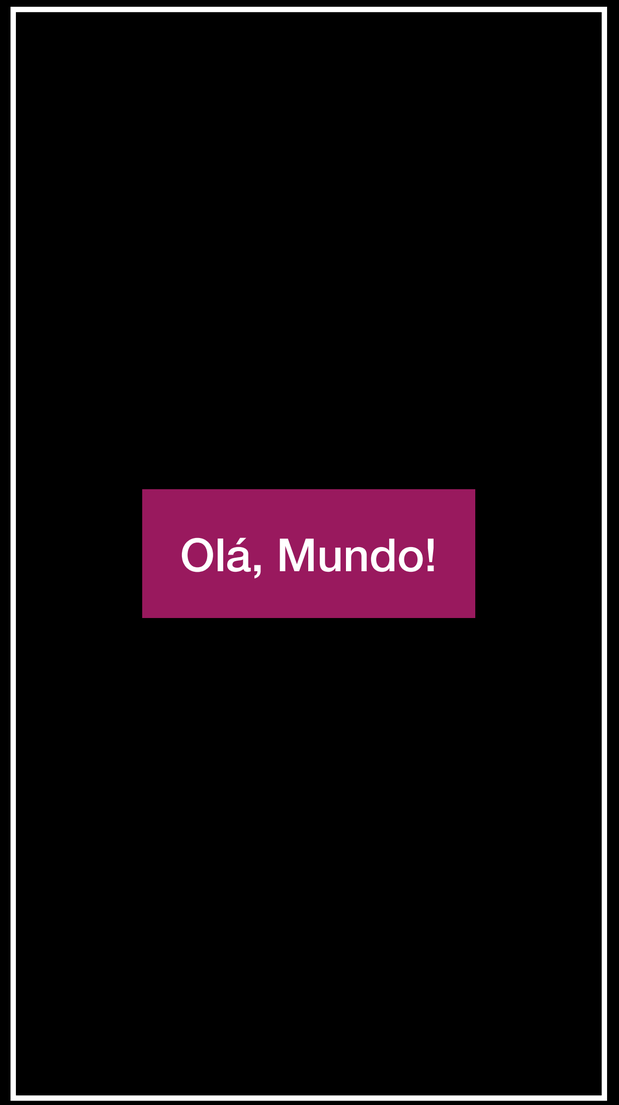
```
struct ContentView: View {

    var body: some View {
        Text("Olá, Mundo!")
            .background(Color.purple)
    }
}
```



LAYOUT SYSTEM

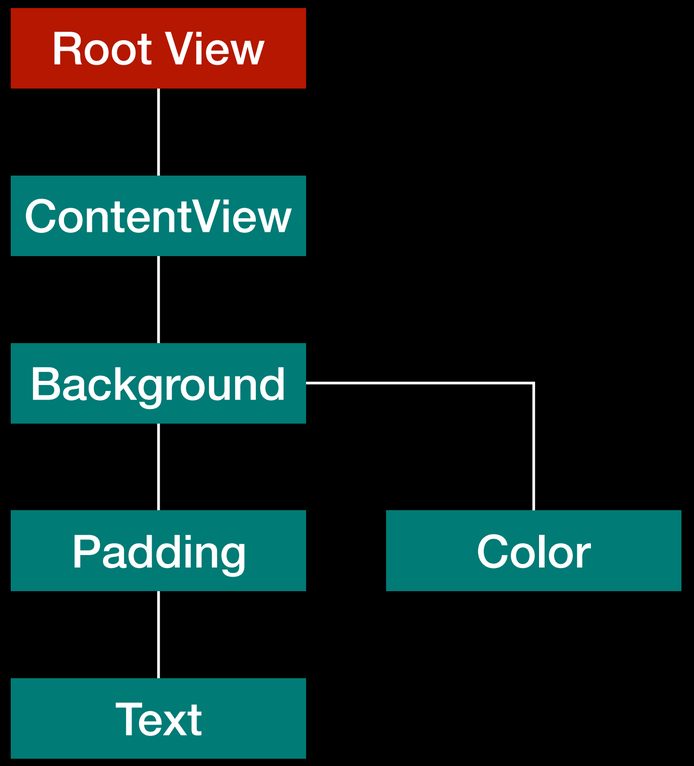
```
struct ContentView: View {

    var body: some View {
        Text("Olá, Mundo!")
            .padding(16)
            .background(Color.purple)
    }
}
```

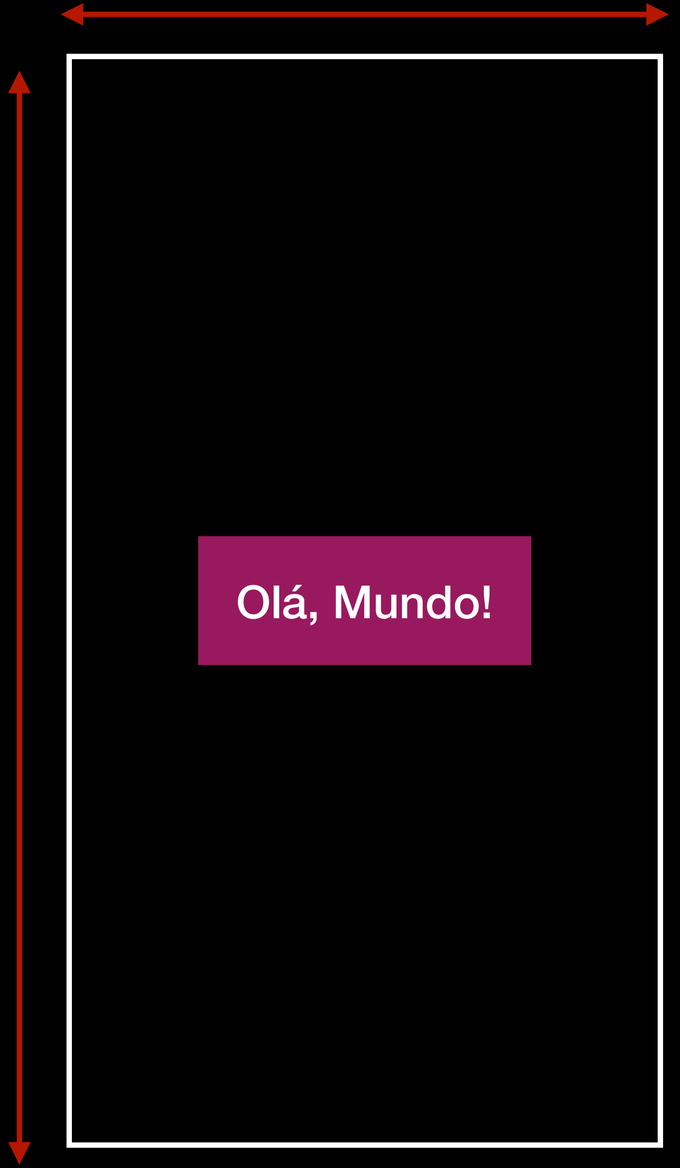




LAYOUT SYSTEM

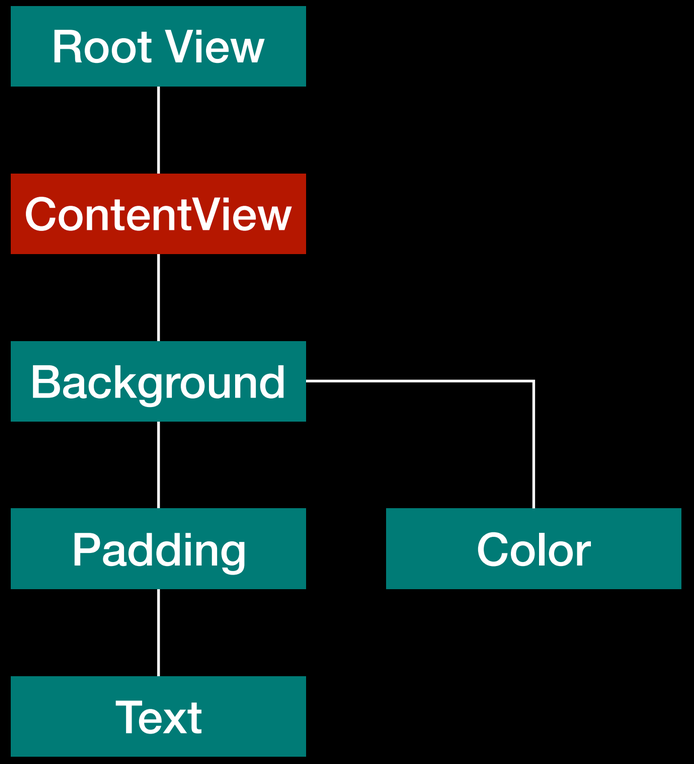
```
struct ContentView: View {

    var body: some View {
        Text("Olá, Mundo!")
            .padding(16)
            .background(Color.purple)
    }
}
```

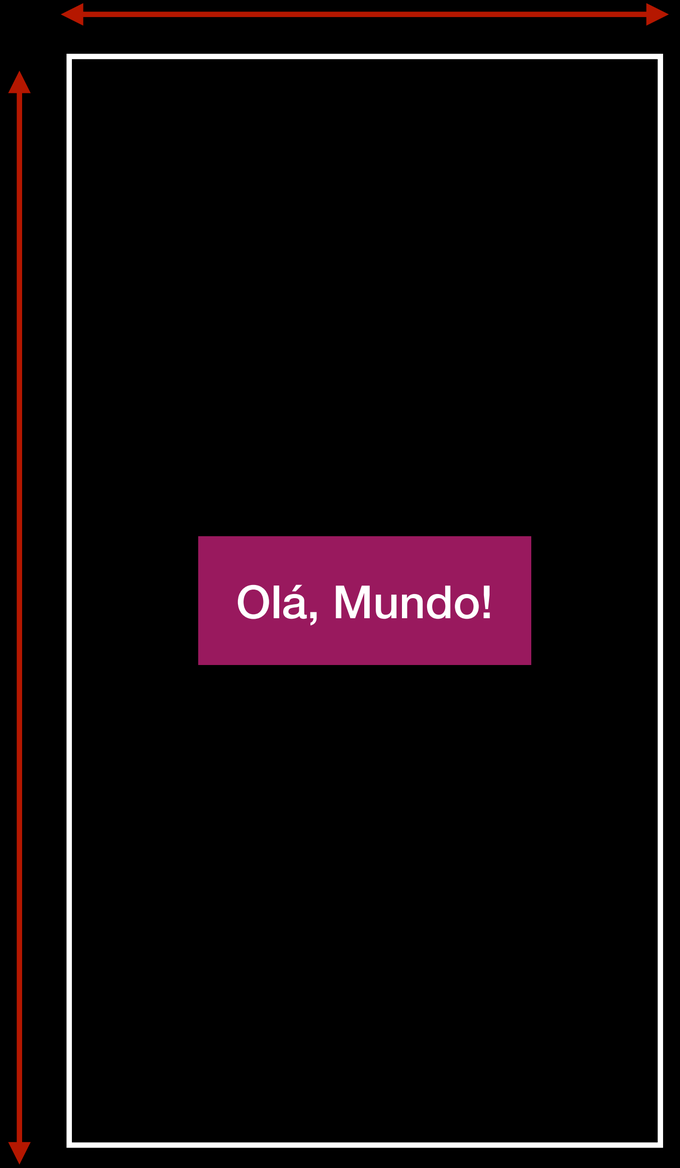




LAYOUT SYSTEM

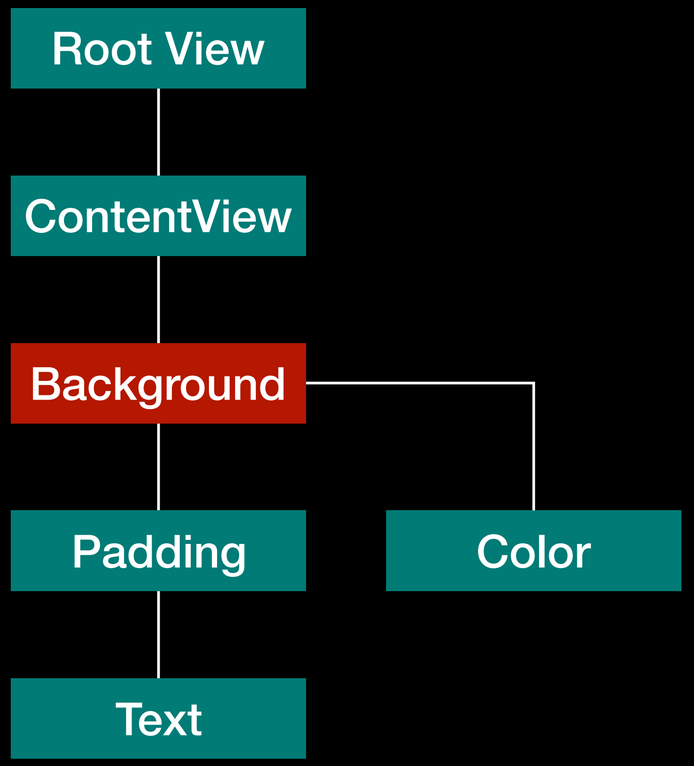
```
struct ContentView: View {

    var body: some View {
        Text("Olá, Mundo!")
            .padding(16)
            .background(Color.purple)
    }
}
```

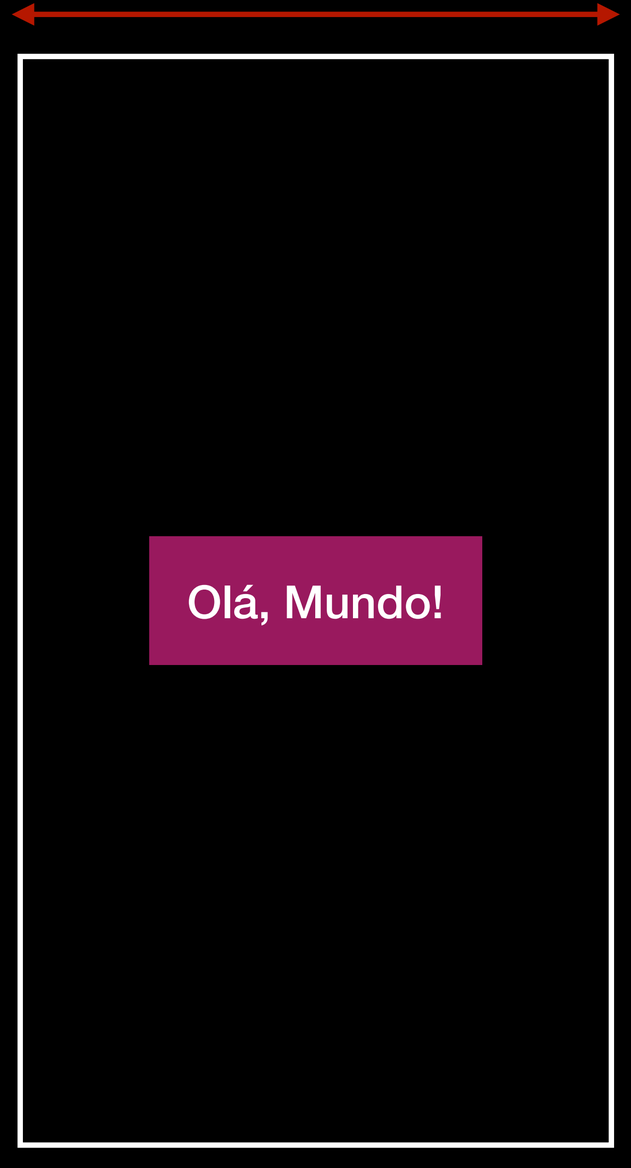




LAYOUT SYSTEM

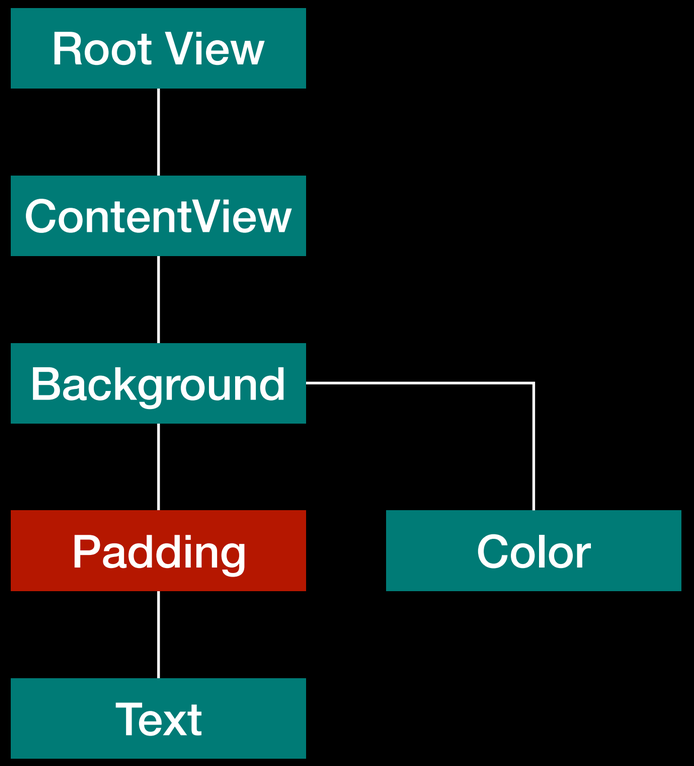
```
struct ContentView: View {

    var body: some View {
        Text("Olá, Mundo!")
            .padding(16)
            .background(Color.purple)
    }
}
```

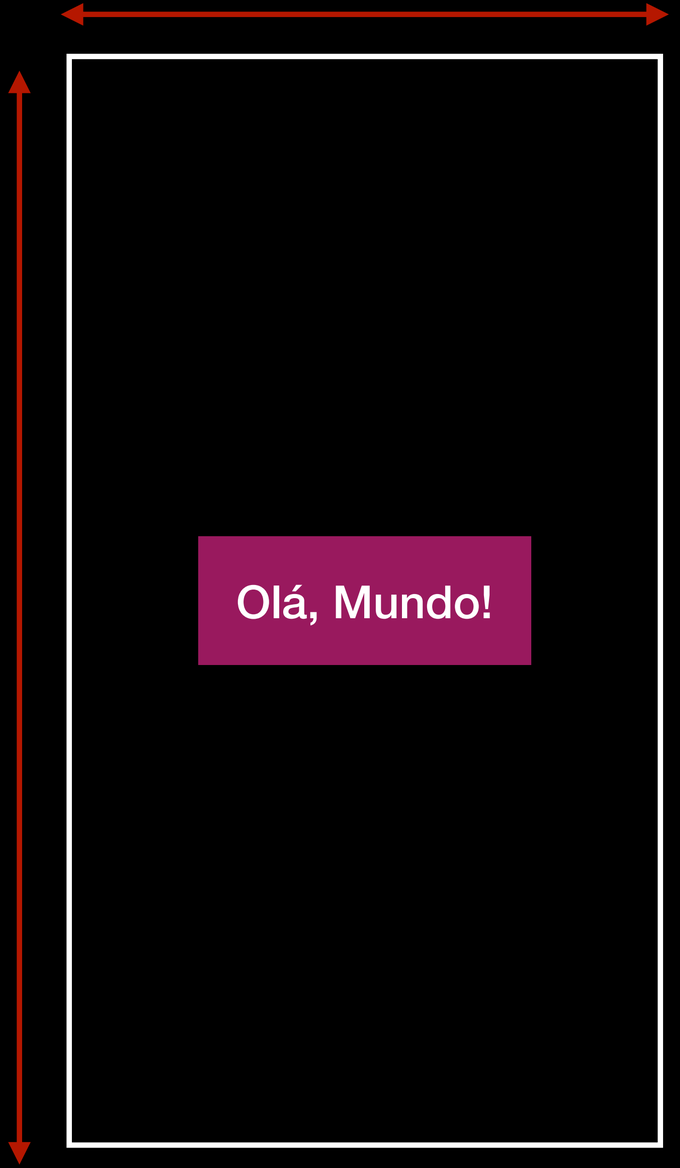




LAYOUT SYSTEM

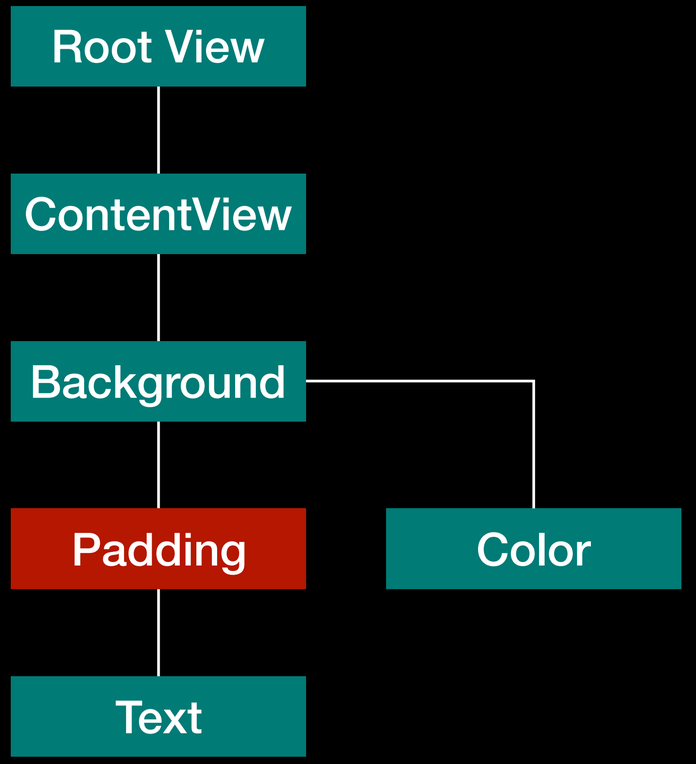
```
struct ContentView: View {

    var body: some View {
        Text("Olá, Mundo!")
            .padding(16)
            .background(Color.purple)
    }
}
```





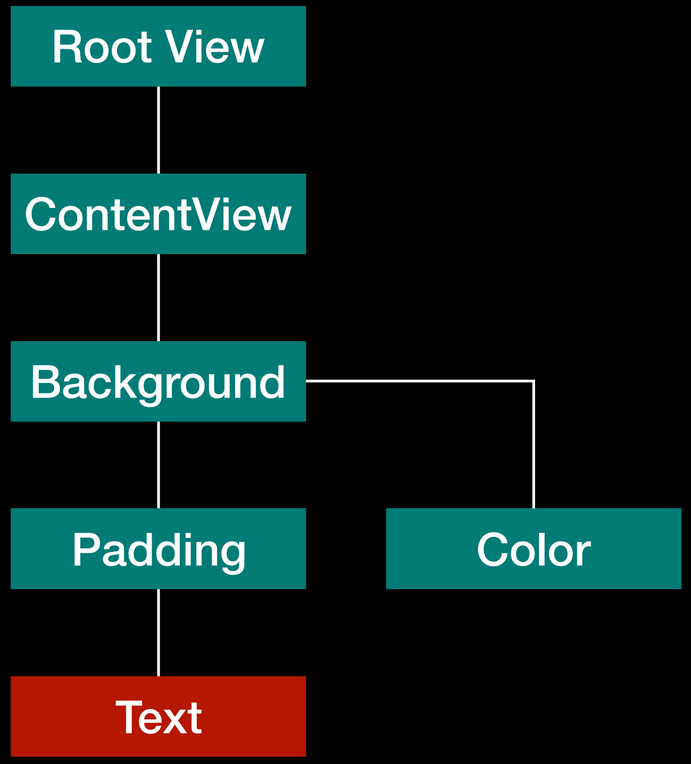
LAYOUT SYSTEM

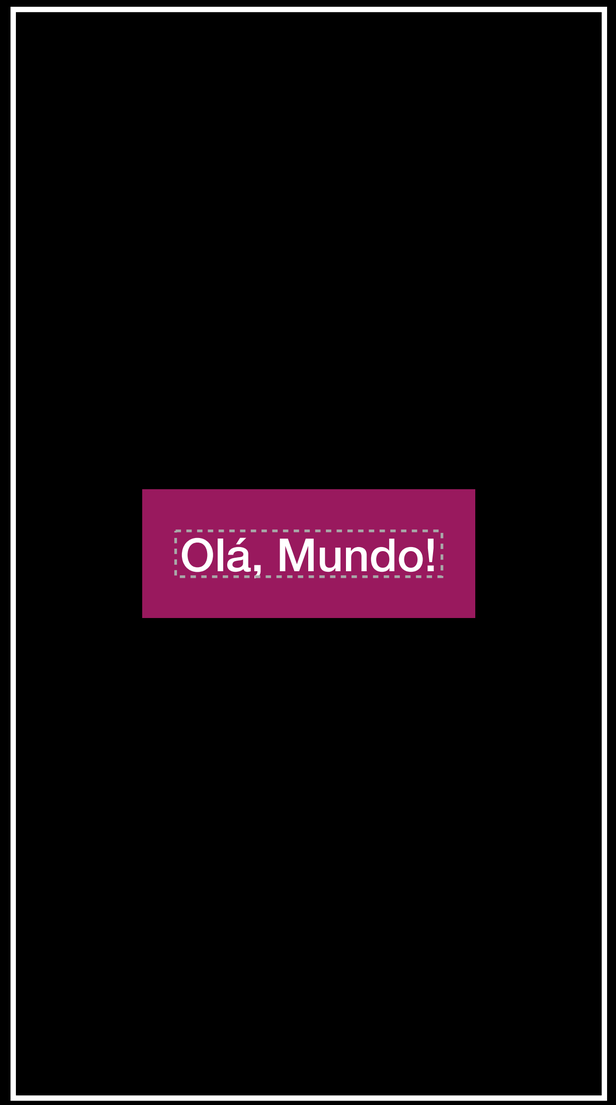
```
struct ContentView: View {

    var body: some View {
        Text("Olá, Mundo!")
            .padding(16)
            .background(Color.purple)
    }
}
```



LAYOUT SYSTEM

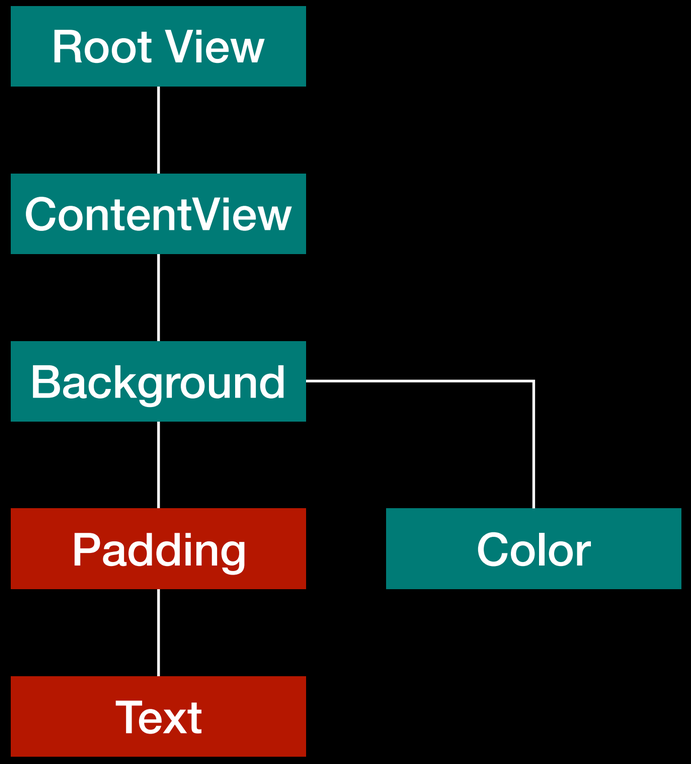
```
struct ContentView: View {

    var body: some View {
        Text("Olá, Mundo!")
            .padding(16)
            .background(Color.purple)
    }
}
```

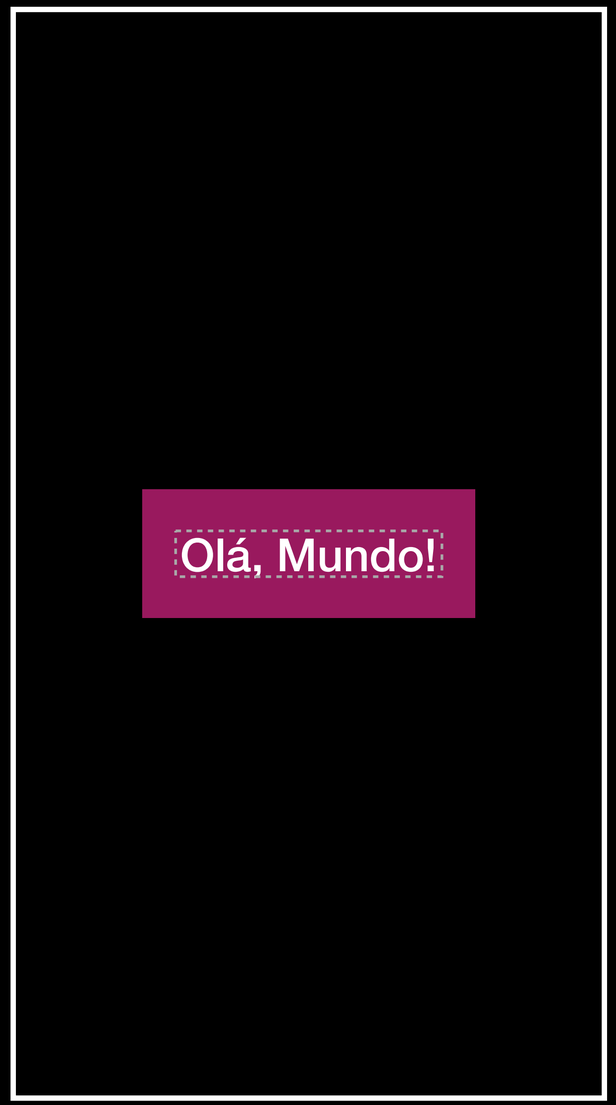




LAYOUT SYSTEM

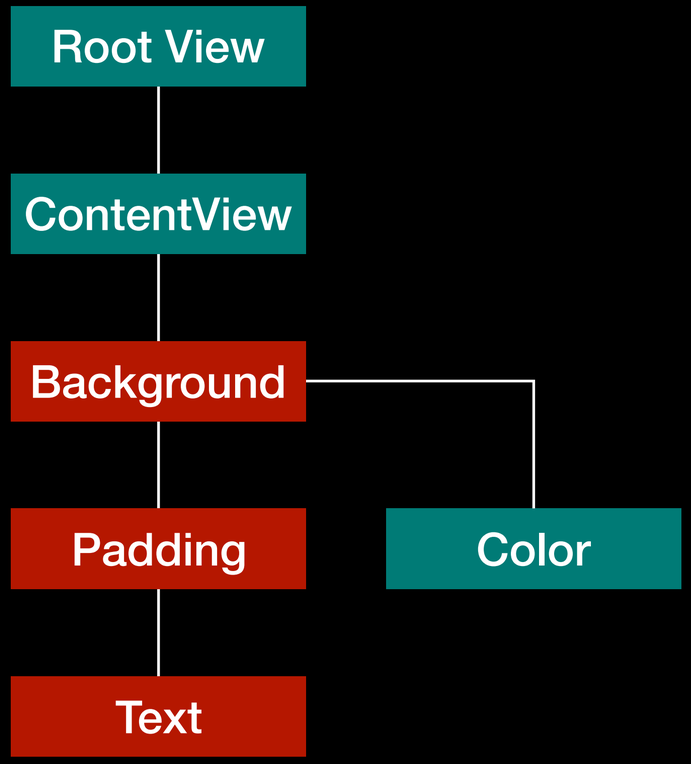
```
struct ContentView: View {

    var body: some View {
        Text("Olá, Mundo!")
            .padding(16)
            .background(Color.purple)
    }
}
```

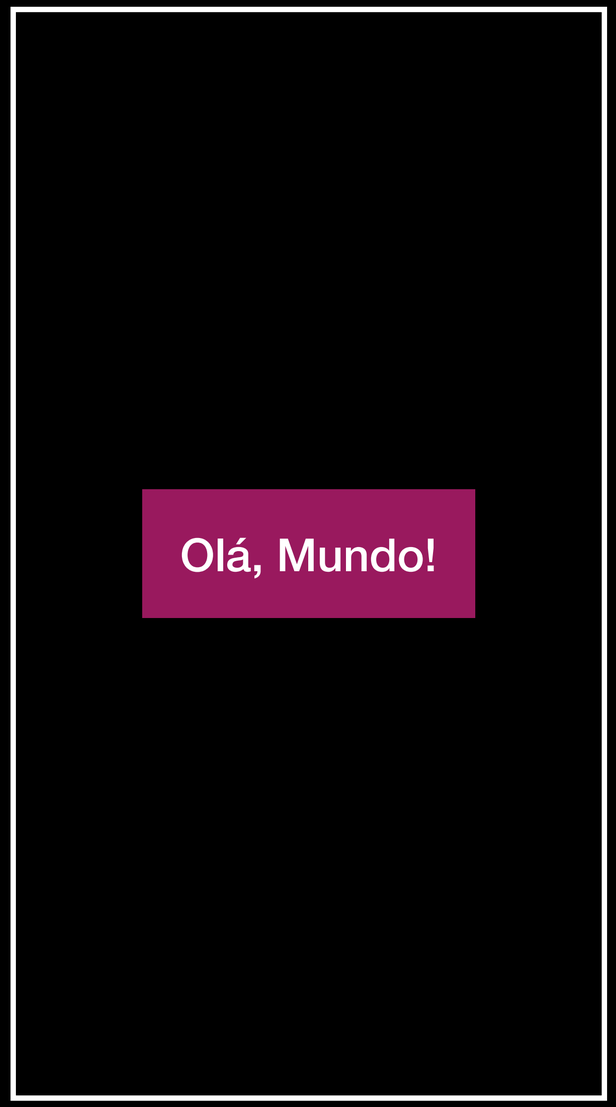




LAYOUT SYSTEM

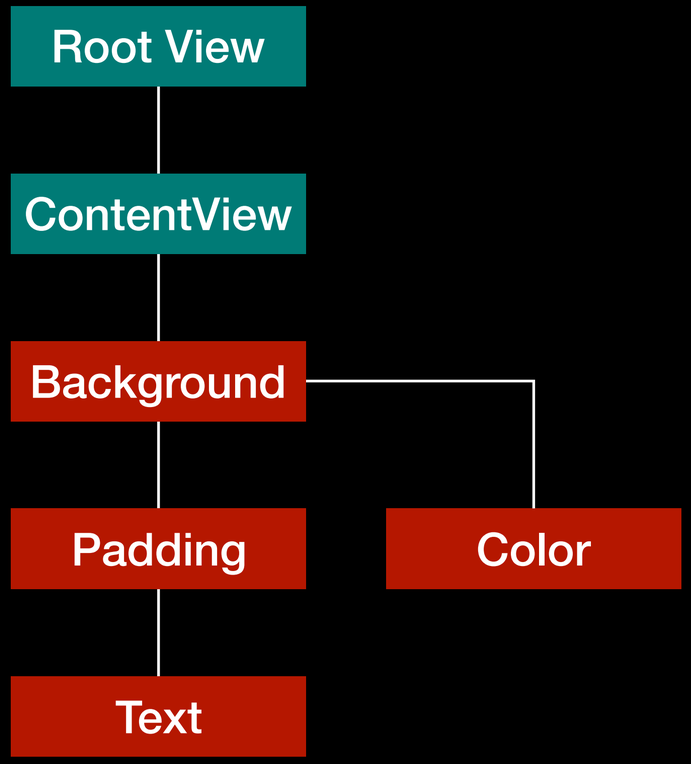
```
struct ContentView: View {

    var body: some View {
        Text("Olá, Mundo!")
            .padding(16)
            .background(Color.purple)
    }
}
```

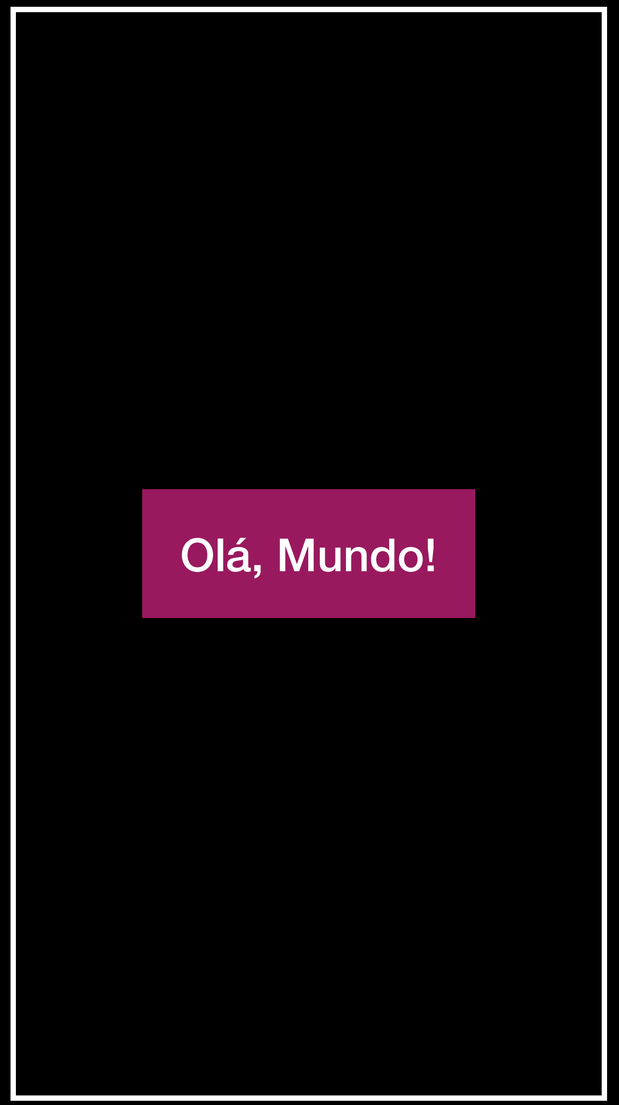




LAYOUT SYSTEM

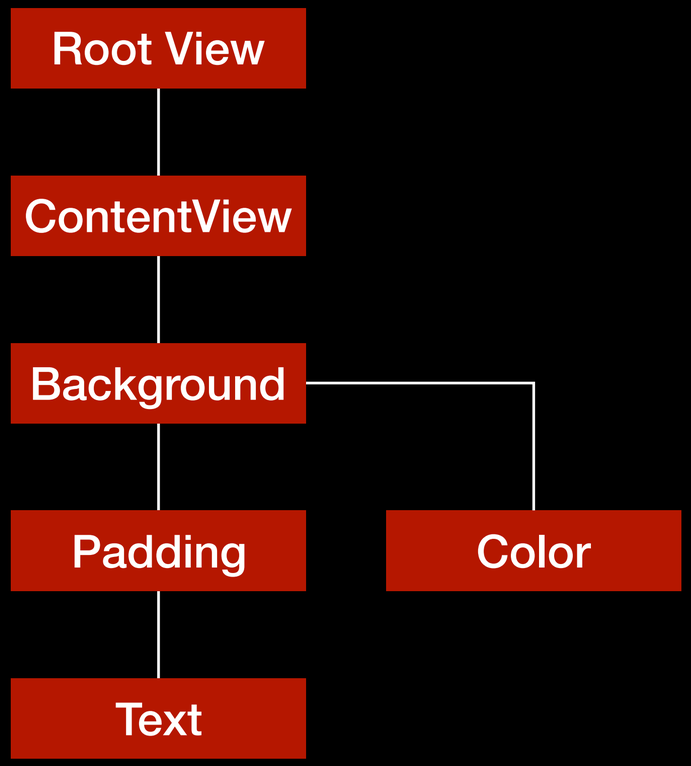
```
struct ContentView: View {

    var body: some View {
        Text("Olá, Mundo!")
            .padding(16)
            .background(Color.purple)
    }
}
```





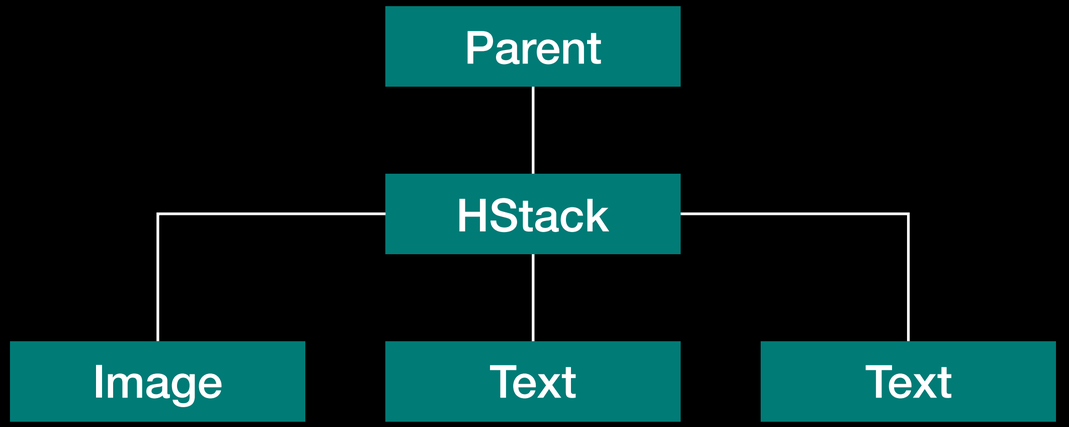
LAYOUT SYSTEM

```
struct ContentView: View {

    var body: some View {
        Text("Olá, Mundo!")
            .padding(16)
            .background(Color.purple)
    }
}
```



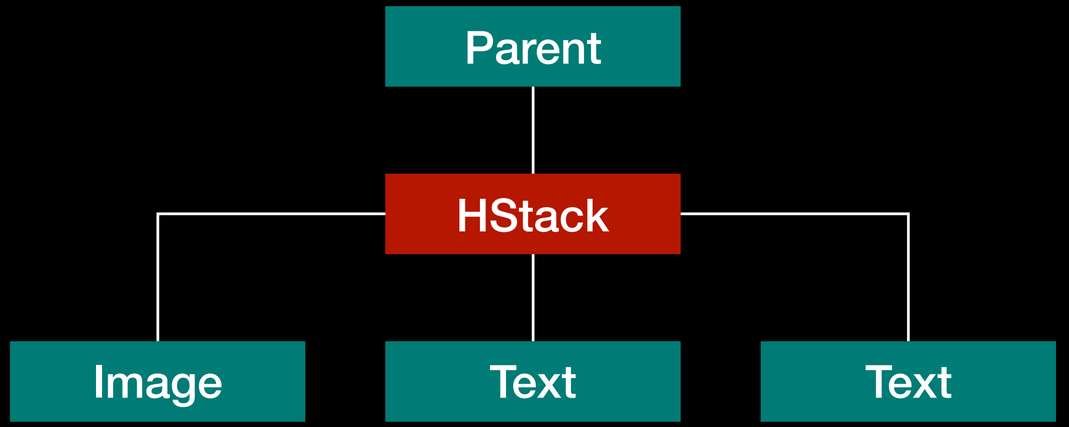
LAYOUT SYSTEM

```
HStack {
    Image(“icon”)
    Text("Nome Completo")
    Text("Txai Wieser")
}
    .lineLimit(1)
```



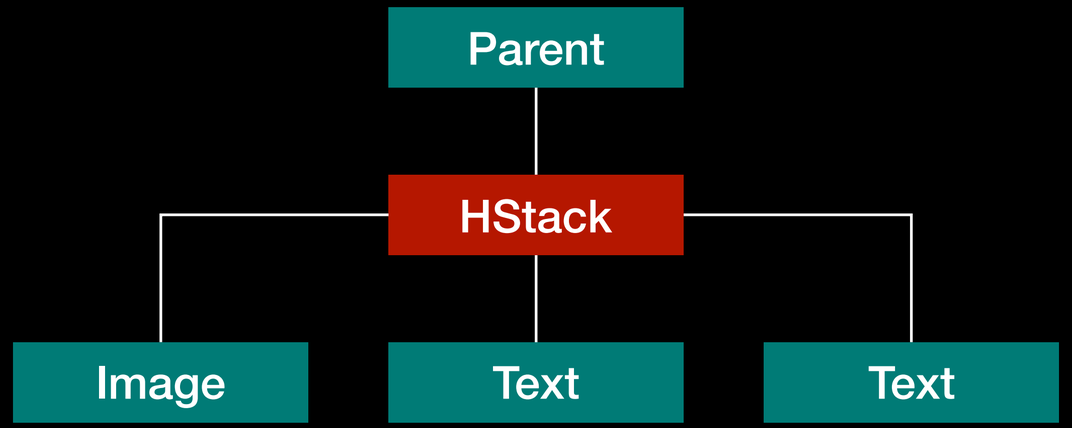
LAYOUT SYSTEM

```
HStack {
    Image(“icon”)
    Text("Nome Completo")
    Text("Txai Wieser")
}
    .lineLimit(1)
```



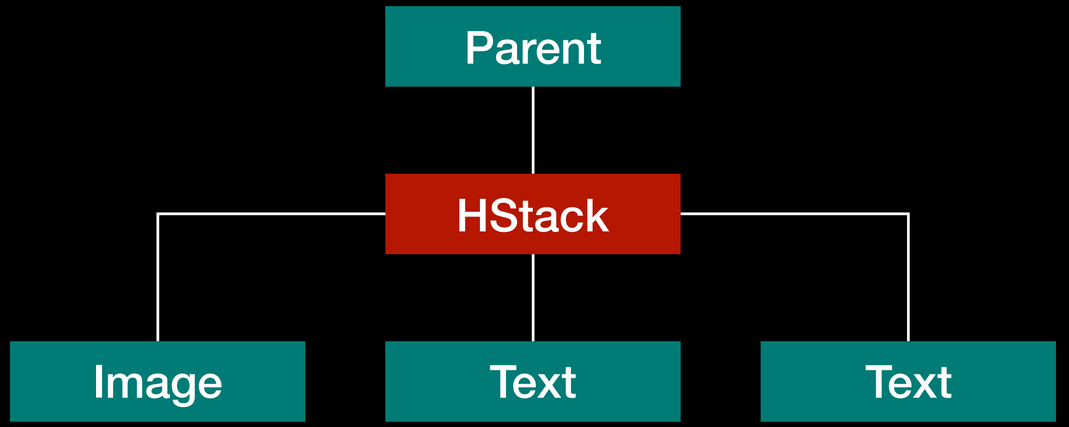
LAYOUT SYSTEM

```
HStack {
    Image(“icon”)
    Text("Nome Completo")
    Text("Txai Wieser")
}
    .lineLimit(1)
```



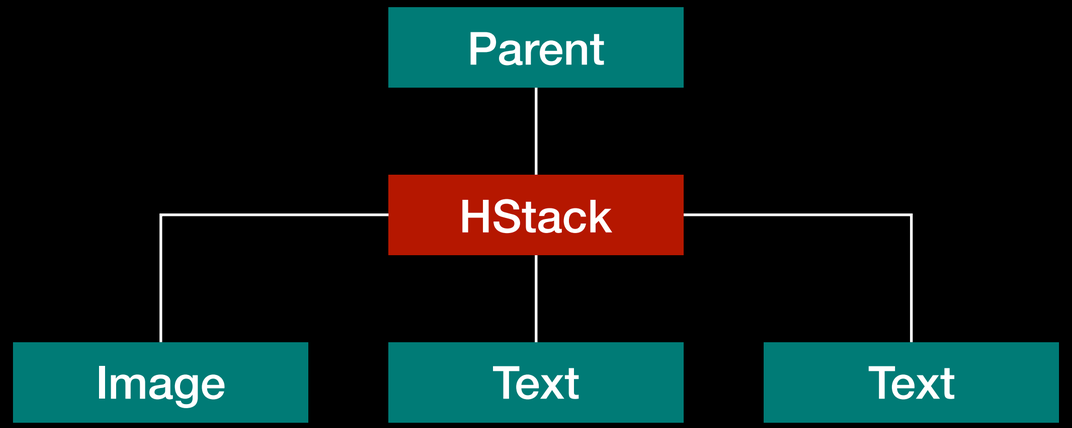
LAYOUT SYSTEM

```
HStack {
    Image(“icon”)
    Text("Nome Completo")
    Text("Txai Wieser")
}
    .lineLimit(1)
```



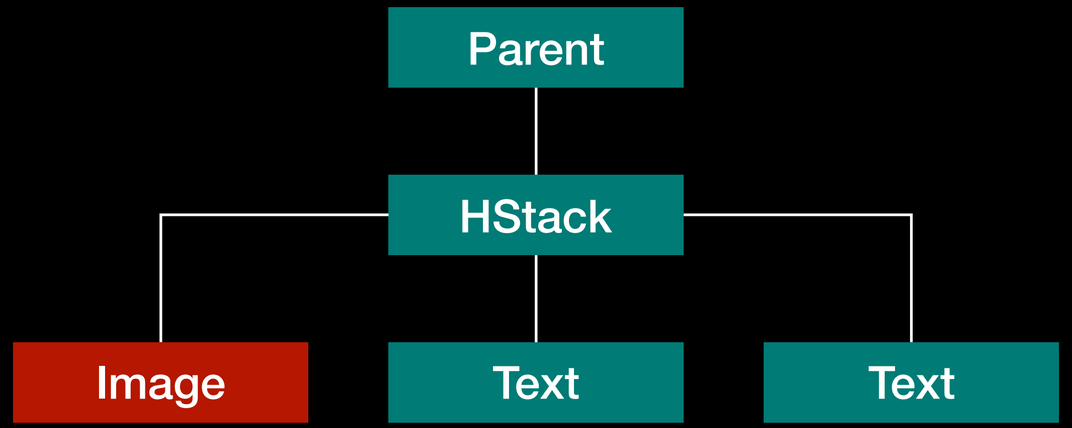
LAYOUT SYSTEM

```
HStack {
    Image(“icon”)
    Text("Nome Completo")
    Text("Txai Wieser")
}
    .lineLimit(1)
```



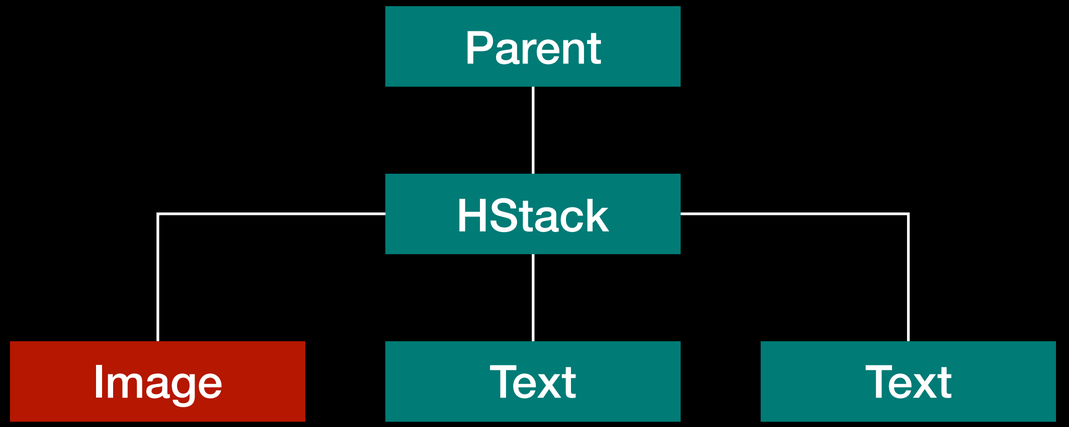
LAYOUT SYSTEM

```
HStack {
    Image(“icon”) // 30x30
    Text("Nome Completo")
    Text("Txai Wieser")
}
    .lineLimit(1)
```



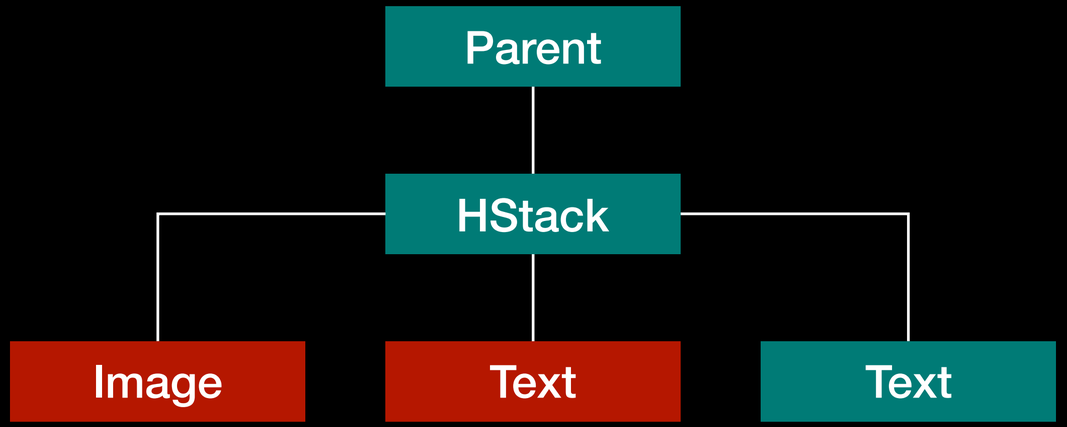
LAYOUT SYSTEM

```
HStack {
    Image(“icon”) // 30x30
    Text("Nome Completo")
    Text("Txai Wieser")
}
    .lineLimit(1)
```



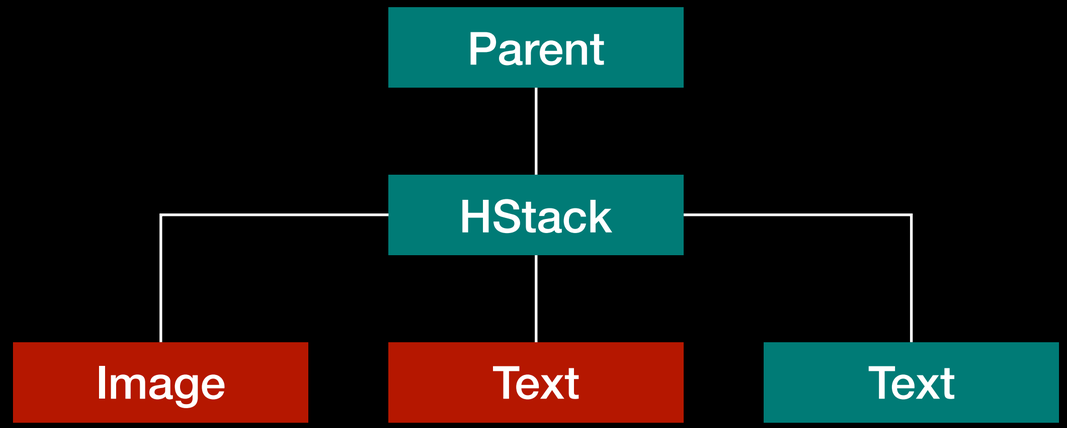
LAYOUT SYSTEM

```
HStack {
    Image(“icon”) // 30x30
    Text("Nome Completo")
    Text("Txai Wieser")
}
    .lineLimit(1)
```



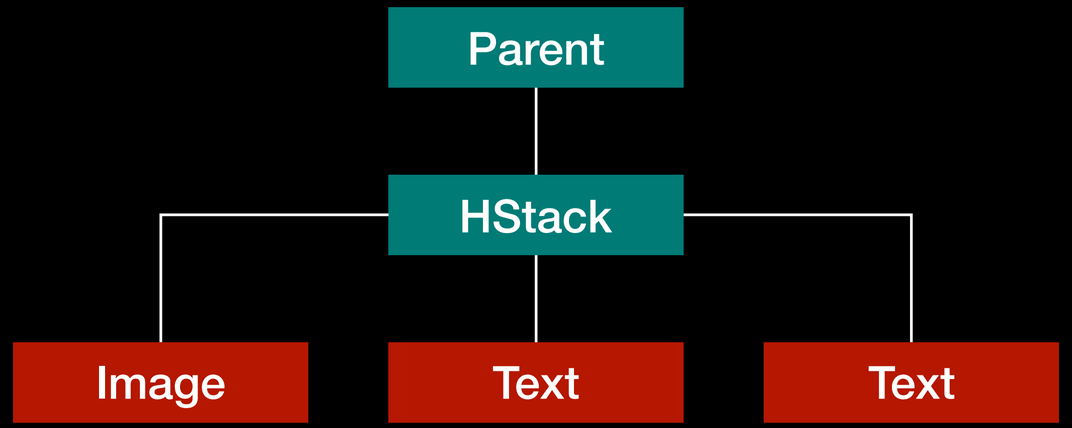
LAYOUT SYSTEM

```
HStack {
    Image(“icon”) // 30x30
    Text("Nome Completo")
    Text("Txai Wieser")
}
    .lineLimit(1)
```



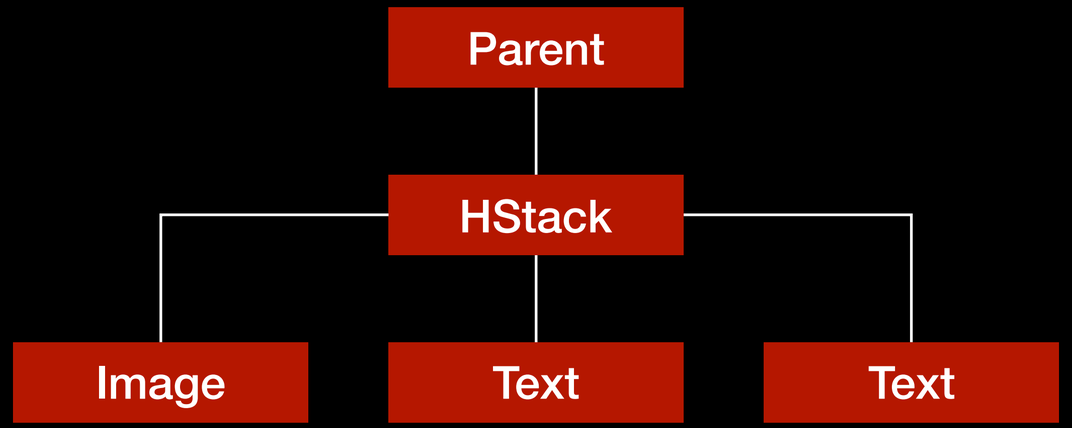
LAYOUT SYSTEM

```
HStack {
    Image(“icon”) // 30x30
    Text("Nome Completo")
    Text("Txai Wieser")
}
    .lineLimit(1)
```



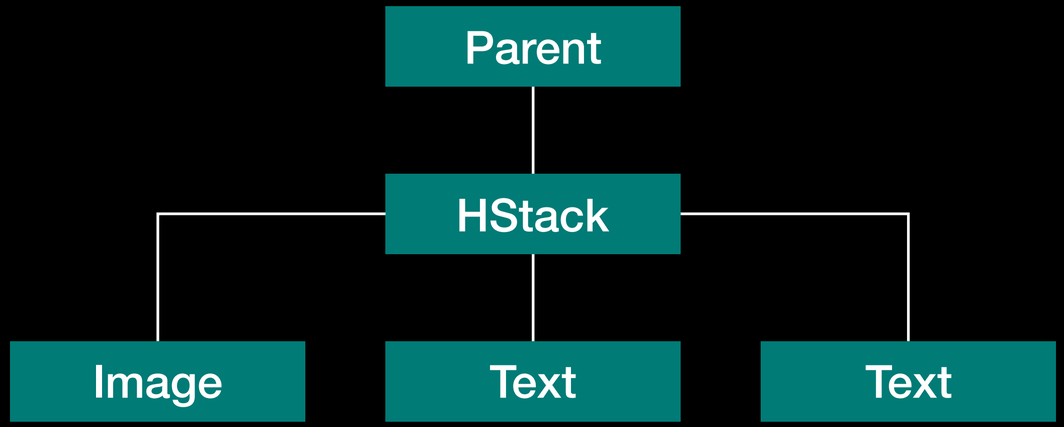
LAYOUT SYSTEM

```
HStack {
    Image(“icon”) // 30x30
    Text("Nome Completo")
    Text("Txai Wieser")
}
    .lineLimit(1)
```



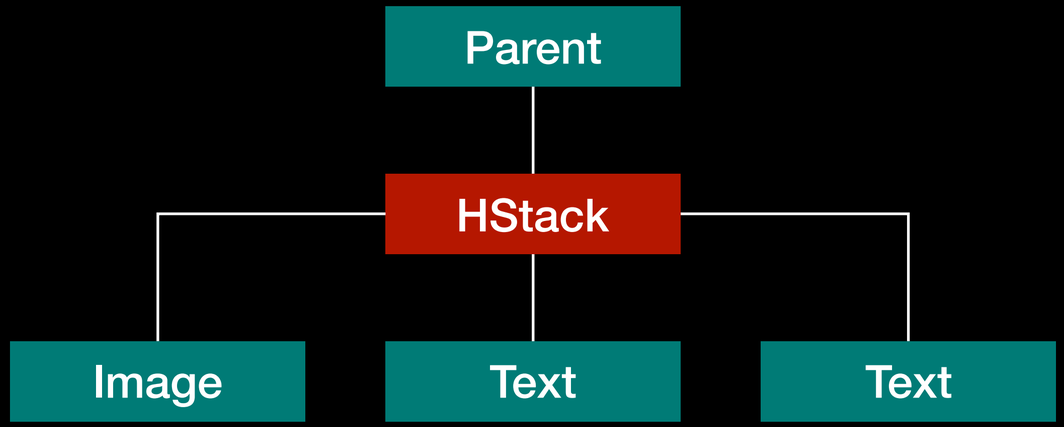
LAYOUT SYSTEM

```
HStack {
    Image(“icon”) // 30x30
    Text("Nome Completo")
    Text("Txai Wieser")
}
    .lineLimit(1)
```



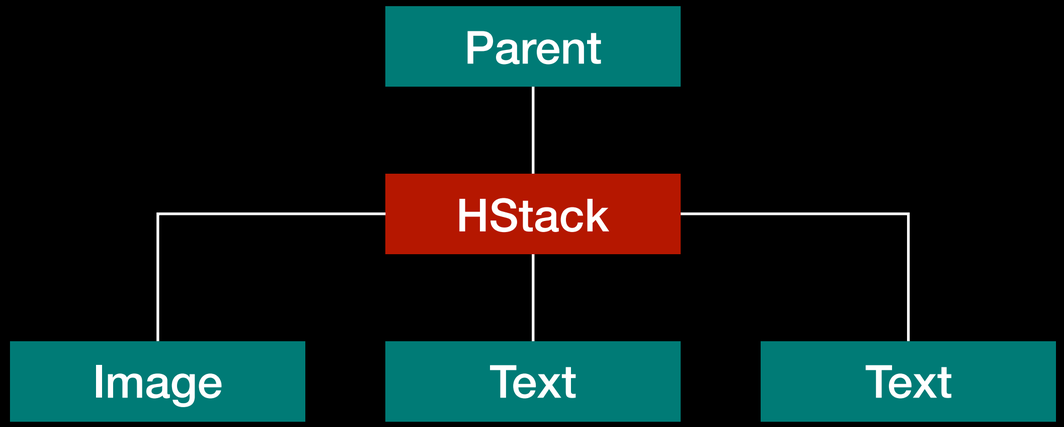
LAYOUT SYSTEM

```
HStack {
    Image(“icon”) // 30x30
    Text("Nome Completo")
    Text("Txai Wieser")
}
    .lineLimit(1)
```



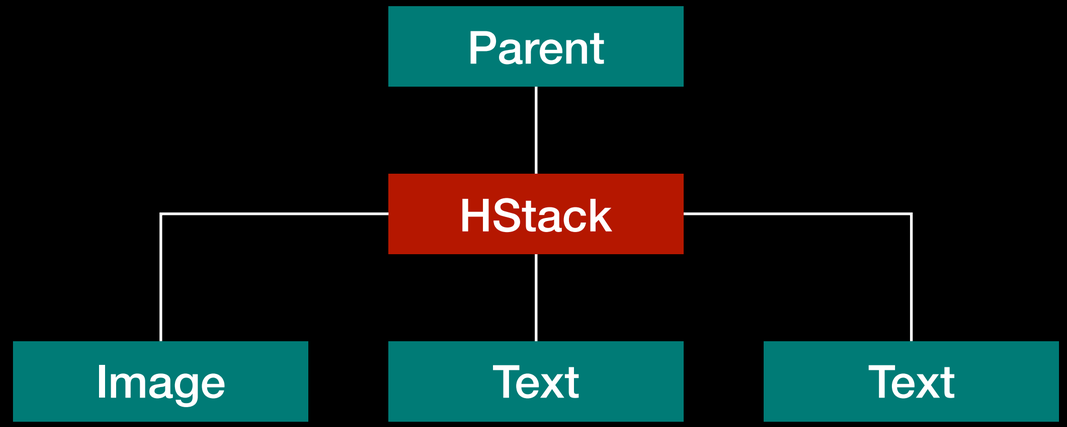
LAYOUT SYSTEM

```
HStack {
    Image(“icon”) // 30x30
    Text("Nome Completo")
    Text("Txai Wieser")
}
    .lineLimit(1)
```



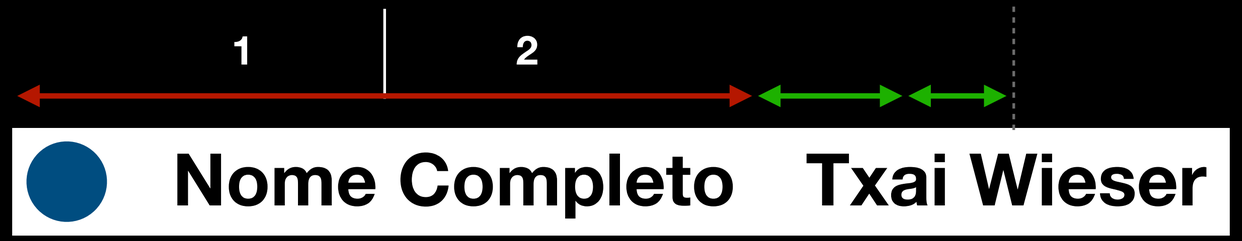
LAYOUT SYSTEM

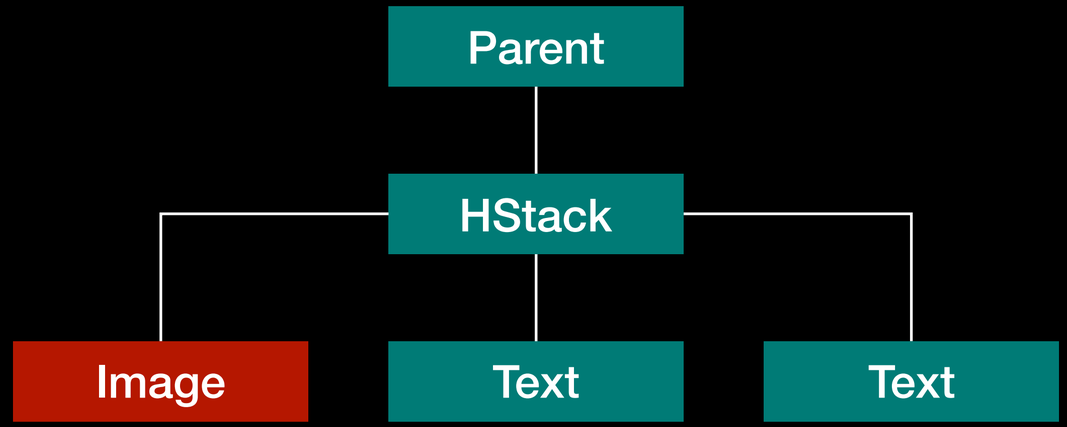
```
HStack {
    Image(“icon”) // 30x30
    Text("Nome Completo")
    Text("Txai Wieser")
}
    .lineLimit(1)
```



LAYOUT SYSTEM

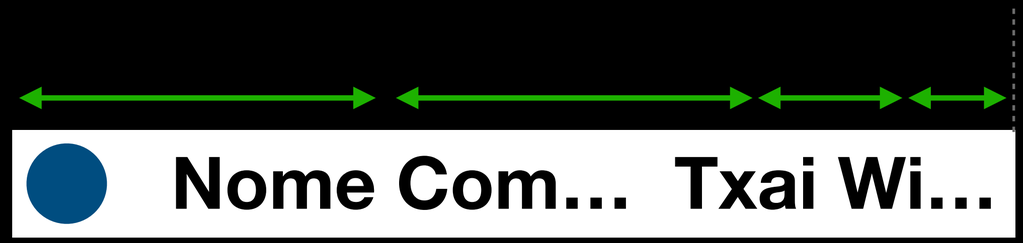
```
HStack {
    Image(“icon”) // 30x30
    Text("Nome Completo")
    Text("Txai Wieser")
}
    .lineLimit(1)
```

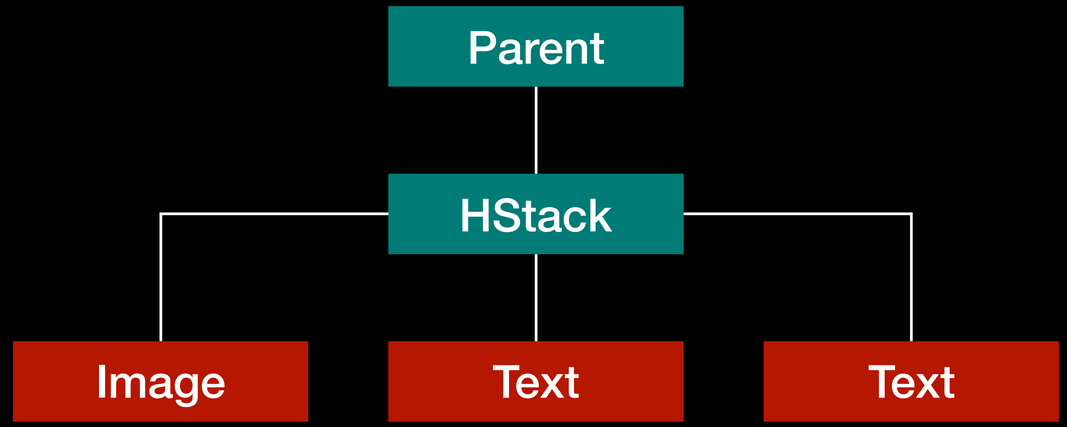




LAYOUT SYSTEM

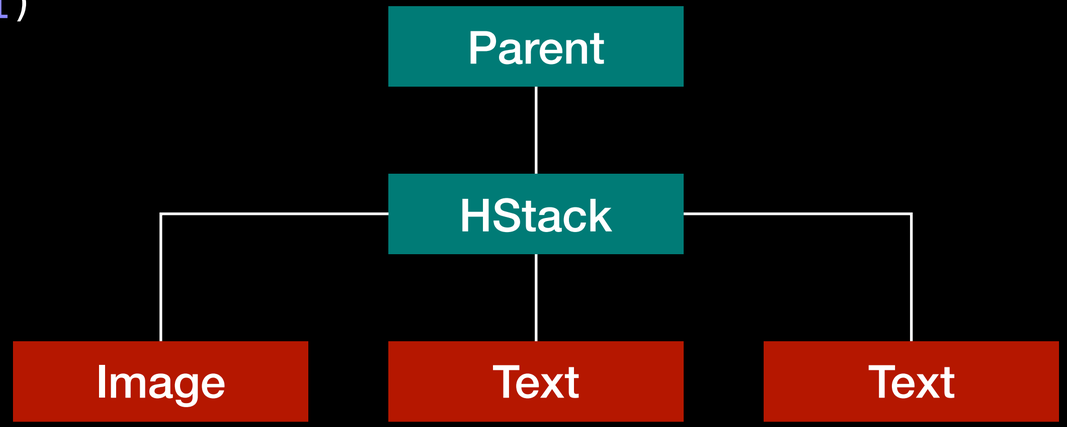
```
HStack {
    Image(“icon”) // 30x30
    Text("Nome Completo")
    Text("Txai Wieser")
}
    .lineLimit(1)
```





LAYOUT SYSTEM


```
HStack {
    Image(“icon”) // 30x30
    Text("Nome Completo")
    Text("Txai Wieser").layoutPriority(1)
}
    .lineLimit(1)
```

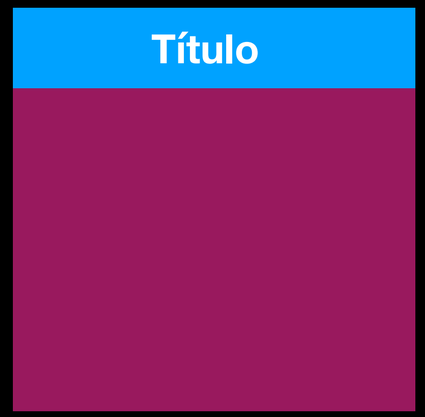


LAYOUT SYSTEM

GEOMETRY READER

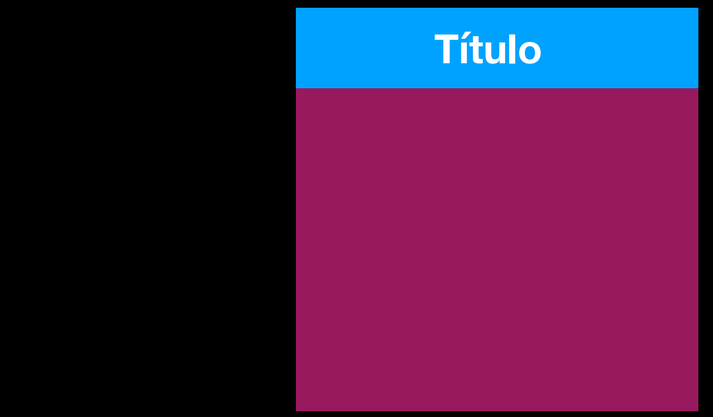
GEOMETRY READER

```
struct ContentView : View {

    var body: some View {
        VStack {
            Text("Header")
            CustomView()
        }
            .frame(width: 300, height: 200)
            .background(Color.blue)
    }
}

struct CustomView: View {

    var body: some View {
       Rectangle()
           .fill(Color.purple)
  }
}
```



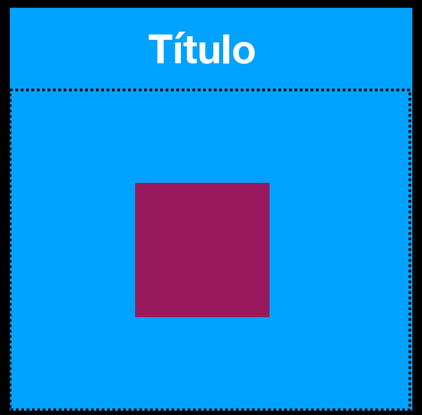
GEOMETRY READER

```
struct ContentView : View {

    var body: some View {
        VStack {
            Text("Header")
            CustomView()
        }
            .frame(width: 300, height: 200)
            .background(Color.blue)
    }
}

struct CustomView: View {

    var body: some View {
       Rectangle()
           .fill(Color.purple)
           .frame(width: CGFloat?, height: CGFloat?)
   }
}
```



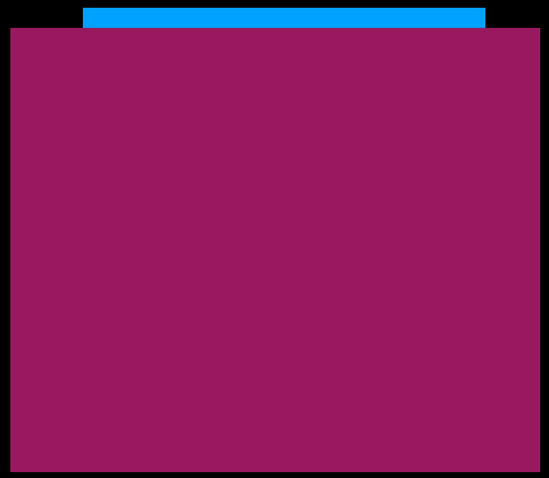
GEOMETRY READER

```
struct ContentView : View {

    var body: some View {
        VStack {
            Text("Header")
            CustomView()
        }
            .frame(width: 300, height: 200)
            .background(Color.blue)
    }
}

struct CustomView: View {

    var body: some View {
       Rectangle()
           .fill(Color.purple)
           .frame(width: 50, height: 50)
   }
}
```



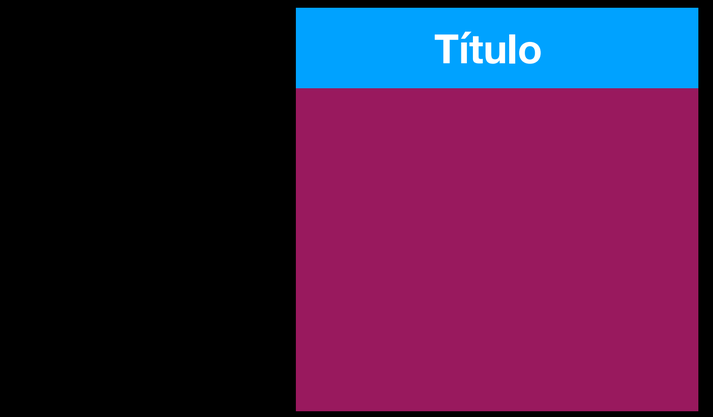
GEOMETRY READER

```
struct ContentView : View {

    var body: some View {
        VStack {
            Text("Header")
            CustomView()
        }
            .frame(width: 300, height: 200)
            .background(Color.blue)
    }
}

struct CustomView: View {

    var body: some View {
       Rectangle()
           .fill(Color.purple)
           .frame(width: 400, height: 200)
   }
}
```



GEOMETRY READER

```
struct ContentView : View {

    var body: some View {
        VStack {
            Text("Header")
            CustomView()
        }
            .frame(width: 300, height: 200)
            .background(Color.blue)
    }
}

struct CustomView: View {

    var body: some View {
       Rectangle()
           .fill(Color.purple)
           .frame(width: CGFloat?, height: CGFloat?)
   }
}
```



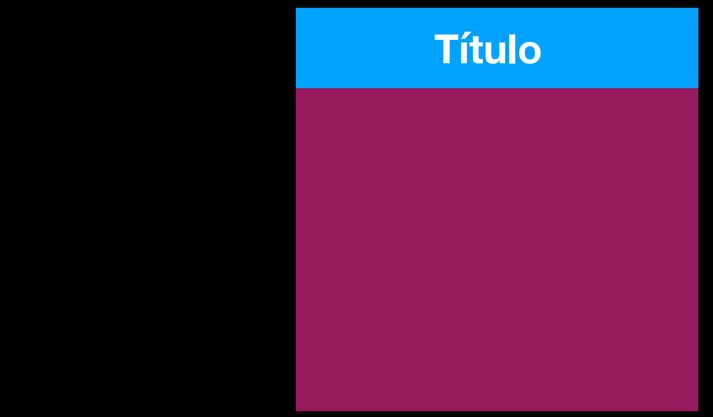
GEOMETRY READER

```
/// A container view that defines its content as a function of its own
/// size and coordinate space. Returns a flexible preferred size to its
/// parent layout.
public struct GeometryReader<Content>: View where Content : View {

    public var content: (GeometryProxy) -> Content

    @inlinable public init(@ViewBuilder content: @escaping (GeometryProxy) -> Content)
    …
}
```

```
/// Acts as a proxy for access to the size and coordinate space (for
/// anchor resolution) of the container view.
public struct GeometryProxy {
    /// The size of the container view.
    public var size: CGSize { get }
    …
    /// The container view's bounds rectangle converted to a defined
    /// coordinate space.
    public func frame(in coordinateSpace: CoordinateSpace) -> CGRect
}
```

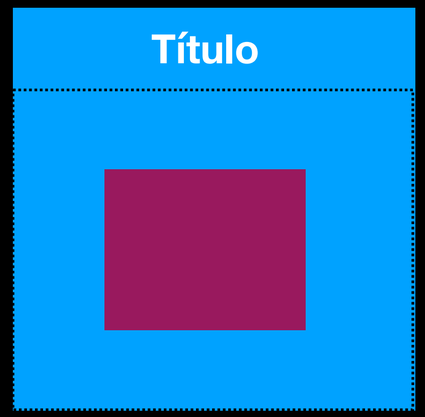
GEOMETRY READER

```
struct ContentView : View {

    var body: some View {
        VStack {
            Text("Header")
            CustomView()
        }
            .frame(width: 300, height: 200)
            .background(Color.blue)
    }
}

struct CustomView: View {

    var body: some View {
       Rectangle()
           .fill(Color.purple)
           .frame(width: CGFloat?, height: CGFloat?)
   }
}
```



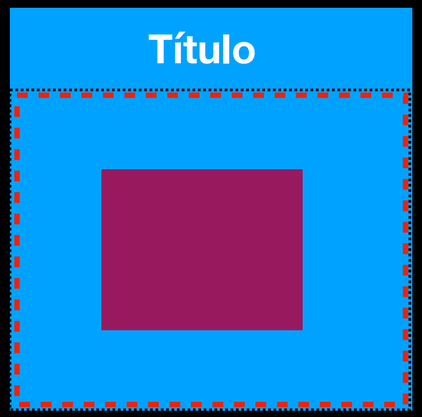
GEOMETRY READER

```
struct ContentView : View {

    var body: some View {
        VStack {
            Text("Header")
            CustomView()
        }
            .frame(width: 300, height: 200)
            .background(Color.blue)
    }
}

struct CustomView: View {

    var body: some View {
        GeometryReader { geometry in
            Rectangle()
                .fill(Color.purple)
                .frame(
                    width: geometry.size.width/2,
                    height: geometry.size.height/2
                )
        }
    }
}
```



GEOMETRY READER

```
struct ContentView : View {

    var body: some View {
        VStack {
            Text("Header")
            CustomView()
        }
            .frame(width: 300, height: 200)
            .background(Color.blue)
    }
}

struct CustomView: View {

    var body: some View {
        GeometryReader { geometry in
            Rectangle()
                .fill(Color.purple)
                .frame(
                    width: geometry.size.width/2,
                    height: geometry.size.height/2
                )
        }
    }
}
```



Views em SwiftUI são como **receitas**

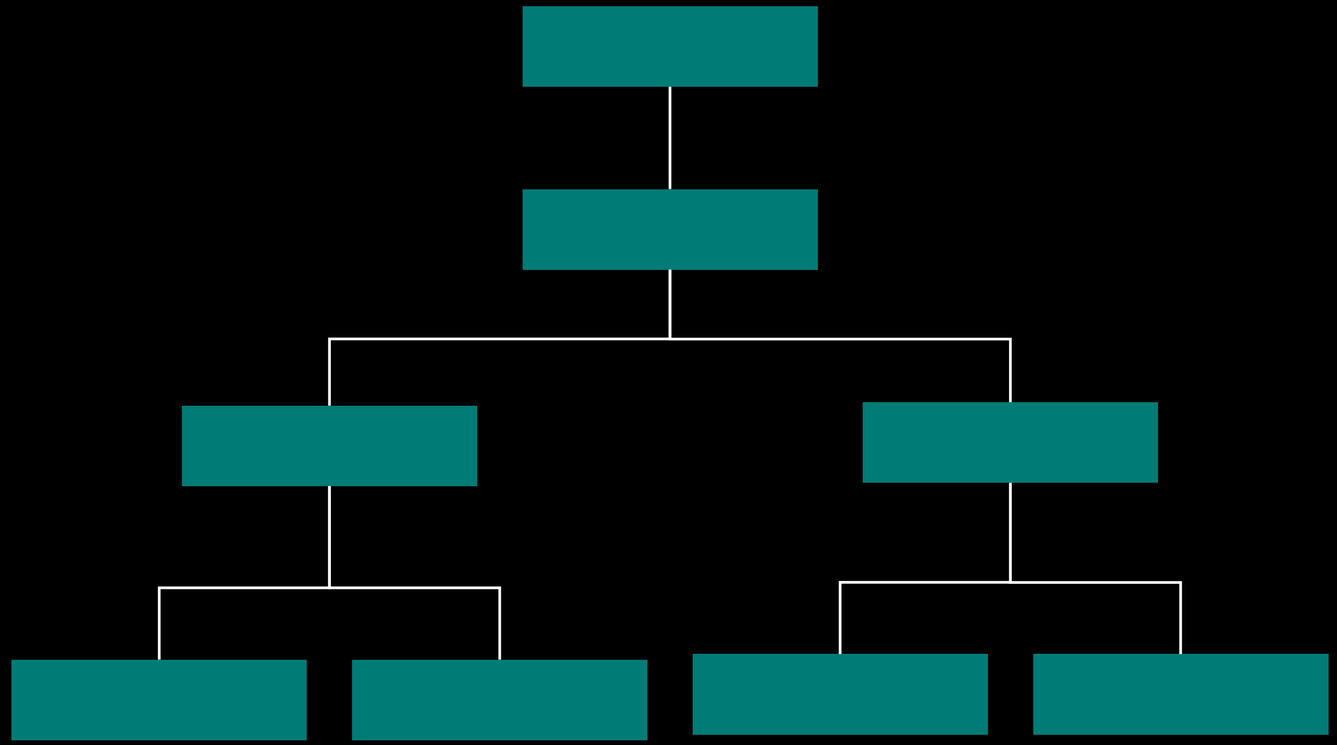
Views em SwiftUI são como **receitas**

Não meça o **tamanho** de uma view, mas sim inclua um **medidor** na receita.

GEOMETRY READER

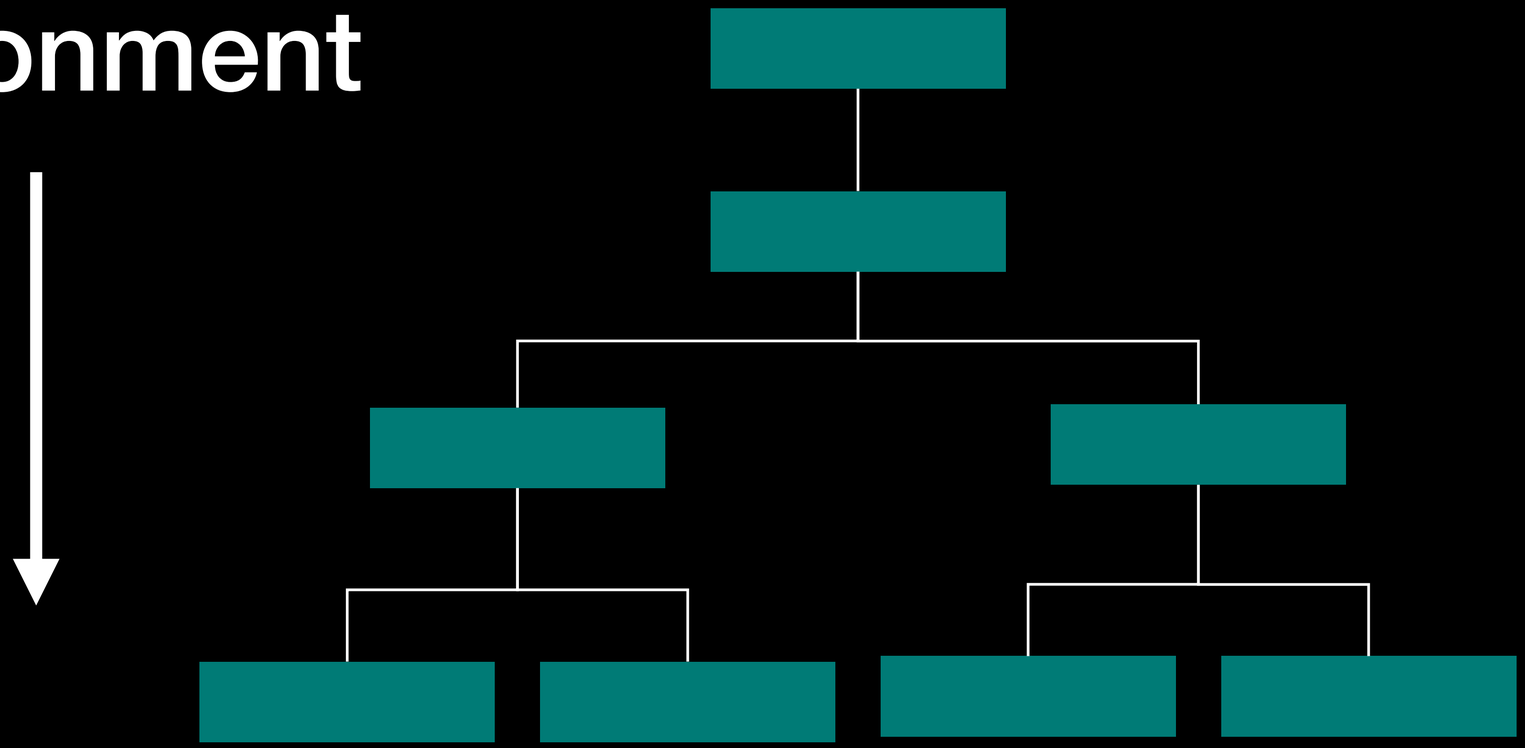
FLUXO DE DADOS

FLUXO DE DADOS



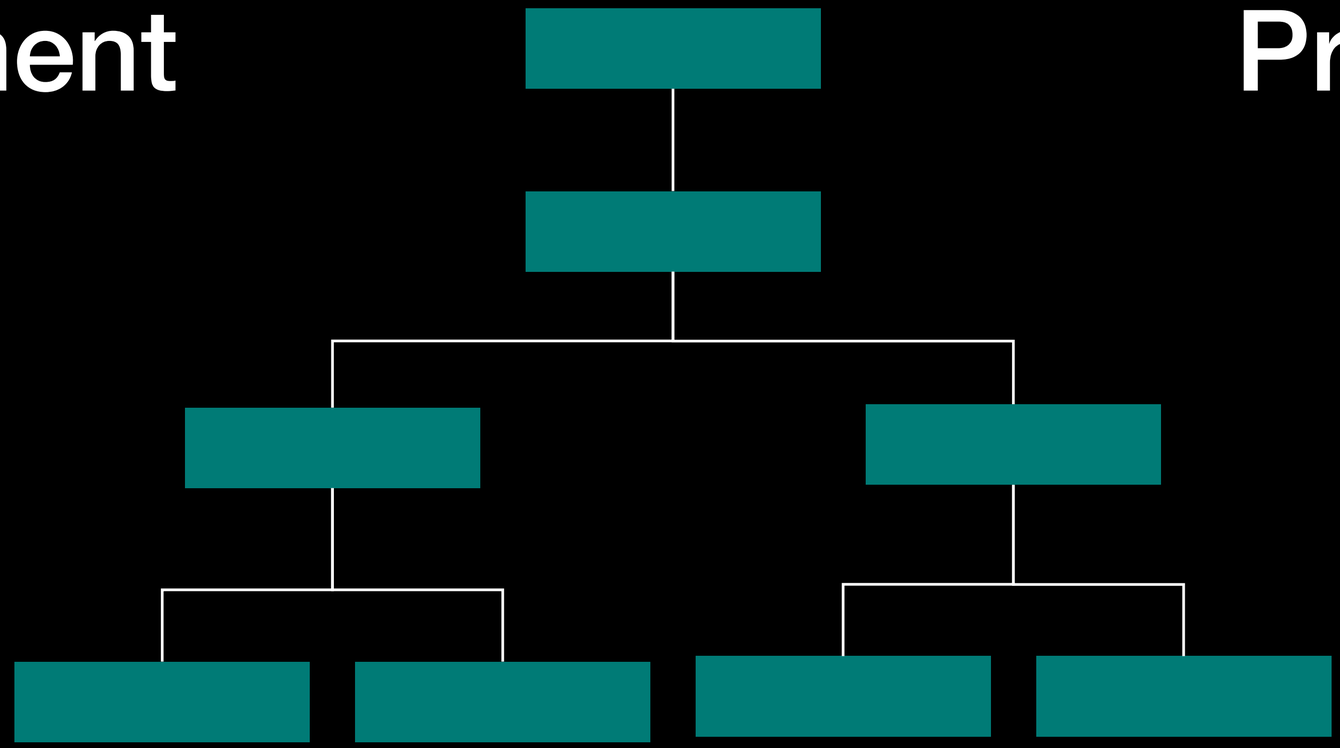
FLUXO DE DADOS

Environment



FLUXO DE DADOS

Environment



Preferences

ENVIRONMENT

ENVIRONMENT

Conjunto de **informações** passadas de uma view para seus **filhos.**

ENVIRONMENT

```
struct ContentView: View {
    var body: some View {
        VStack {
            HStack {
                Text("Title")
                Spacer()
                Text("Detail")
            }
                .font(.headline)

            HStack {
                Text("Subtitle")
                Spacer()
                Text("Info")
            }
                .font(.subheadline)
        }
            .lineLimit(1)
    }
}
```

Title **Detail**

Subtitle Info

ENVIRONMENT

```
struct ContentView: View {
    var body: some View {
        VStack {
            HStack {
                Text("Title")
                Spacer()
                Text("Detail")
            }
                .font(.headline)

            HStack {
                Text("Subtitle")
                Spacer()
                Text("Info")
            }
                .font(.subheadline)
        }
            .lineLimit(1)
    }
}
```

Title **Detail**

Subtitle Info

ENVIRONMENT

```
struct ContentView: View {
    var body: some View {
        VStack {
            HStack {
                Text("Title")
                Spacer()
                Text("Detail")
            }
                .environment(\.font, .headline)

            HStack {
                Text("Subtitle")
                Spacer()
                Text("Info")
            }
                .environment(\.font, .subheadline)
        }
            .environment(\.lineLimit, 1)
    }
}
```

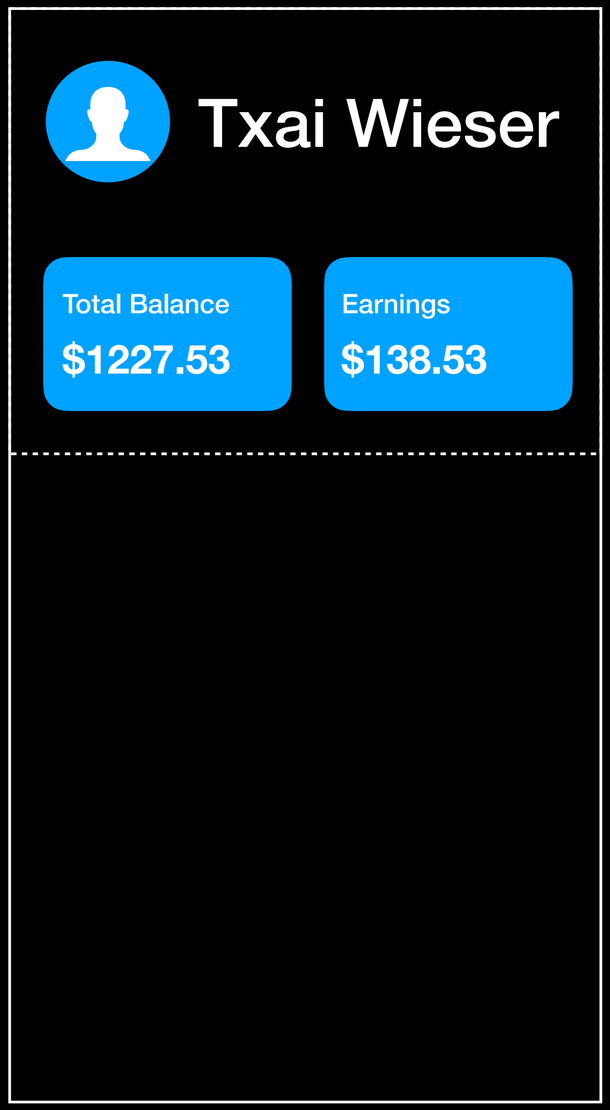
Title **Detail**

Subtitle Info

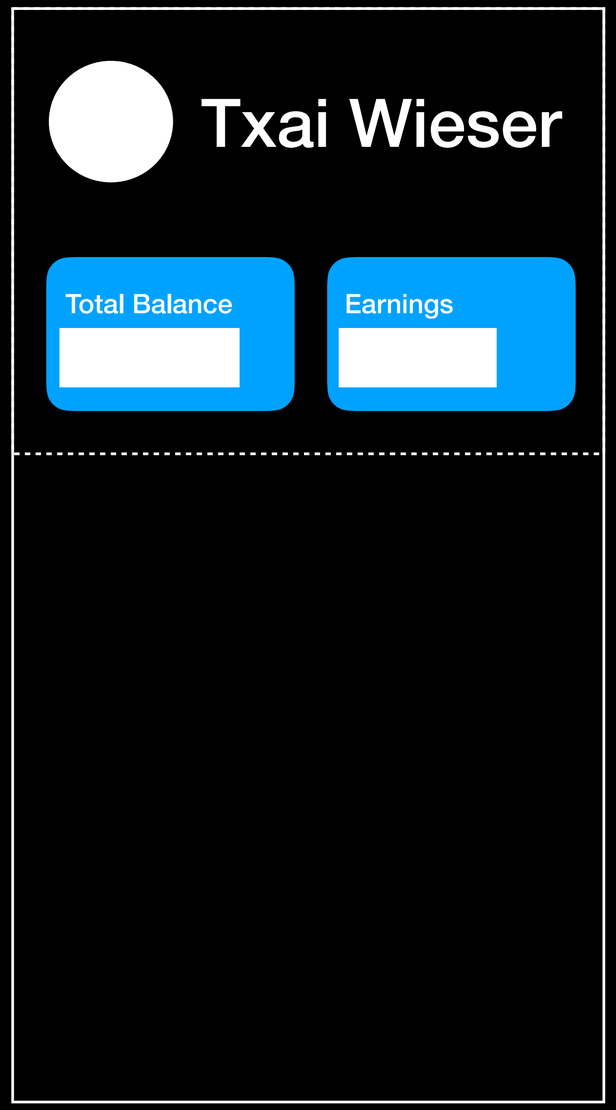
ENVIRONMENT

```
extension EnvironmentValues {
    public var isEnabled: Bool
    public var font: Font?
    public var lineLimit: Int?
    public var layoutDirection: LayoutDirection
    public var colorScheme: ColorScheme

    public var disableAutocorrection: Bool?
    public var sizeCategory: ContentSizeCategory
    public var managedObjectContext: NSManagedObjectContext

    public var displayScale: CGFloat
    public var imageScale: Image.Scale
    public var locale: Locale
    public var calendar: Calendar
    public var timeZone: TimeZone

    public var horizontalSizeClass: UserInterfaceSizeClass?
    public var verticalSizeClass: UserInterfaceSizeClass?

    public var editMode: Binding<EditMode>?
    public var presentationMode: Binding<PresentationMode>
    …
}
```

ENVIRONMENT

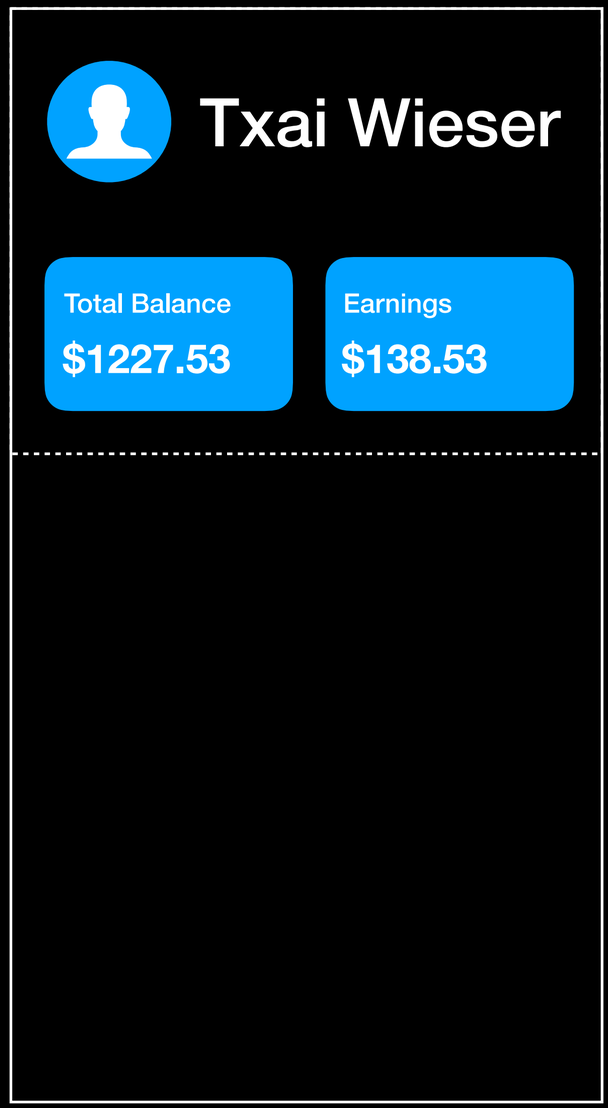


ENVIRONMENT



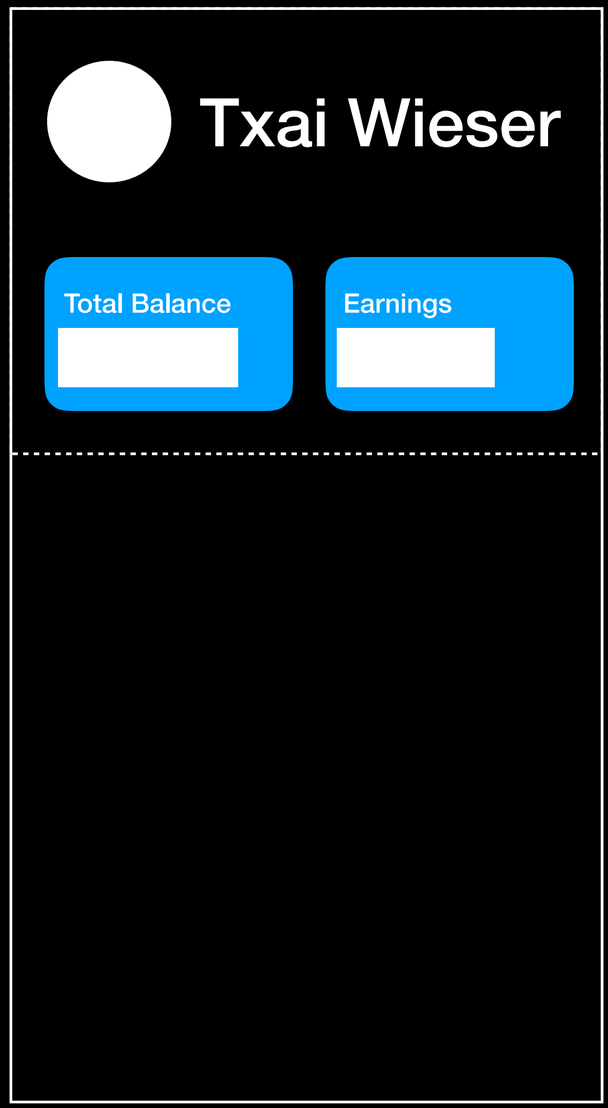
ENVIRONMENT

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

extension EnvironmentValues {
    var isRedacted: Bool {
        get { return self[RedactInfo.self] }
        set { self[RedactInfo.self] = newValue }
    }
}
```



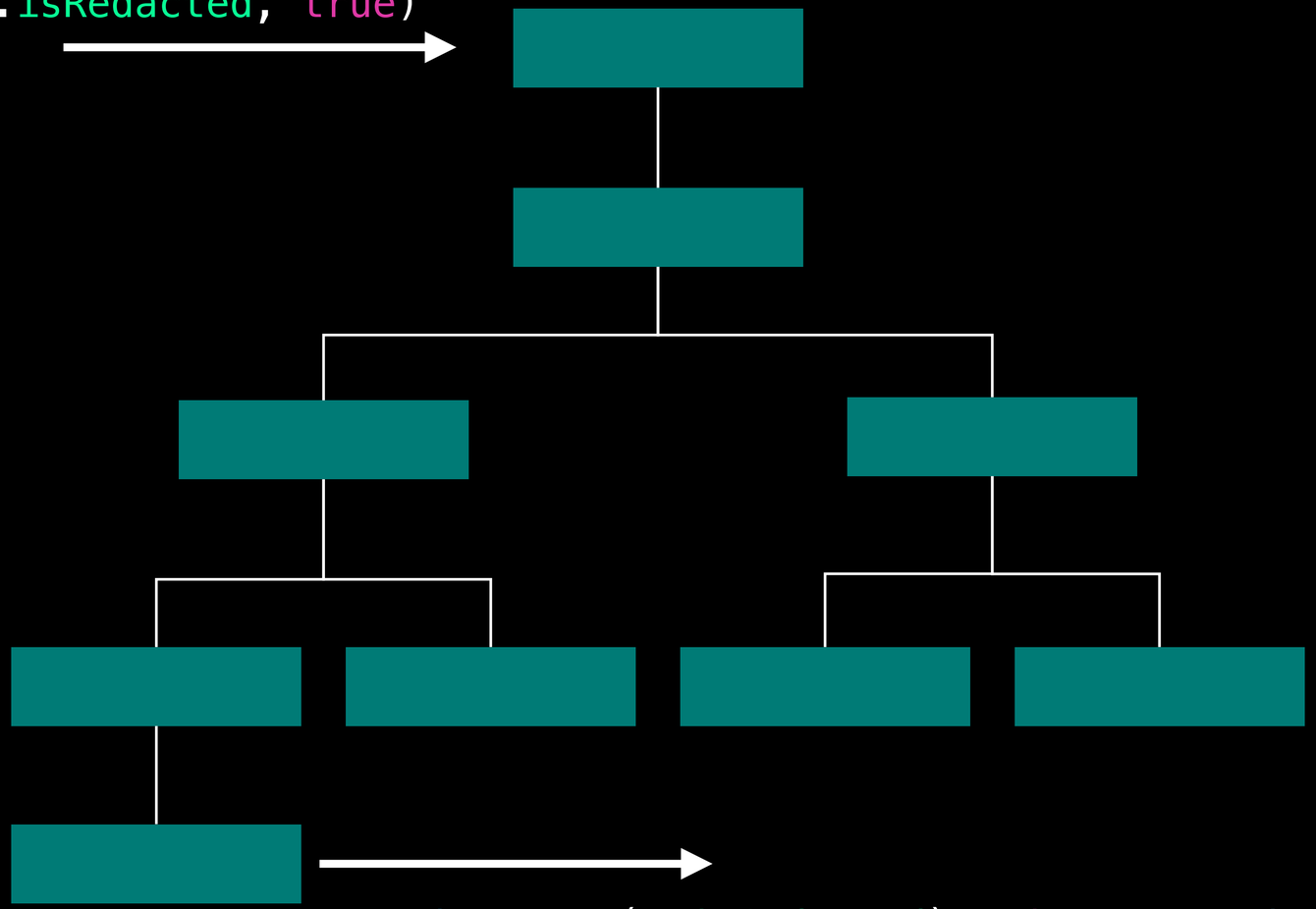
ENVIRONMENT

```swift
//  SceneDelegate.swift

let appView = AppView()

UIHostingController(
    rootView: appView.environment(\.isRedacted, true)
)



//  CurrencyLabel.swift

struct CurrencyLabel: View {
    var value: String
    @Environment(\.isRedacted) private var isRedacted

    var body: some View {
        Text(value)
            .overlay(isRedacted ? Color.primary : Color.clear)
    }
}
```



ENVIRONMENT

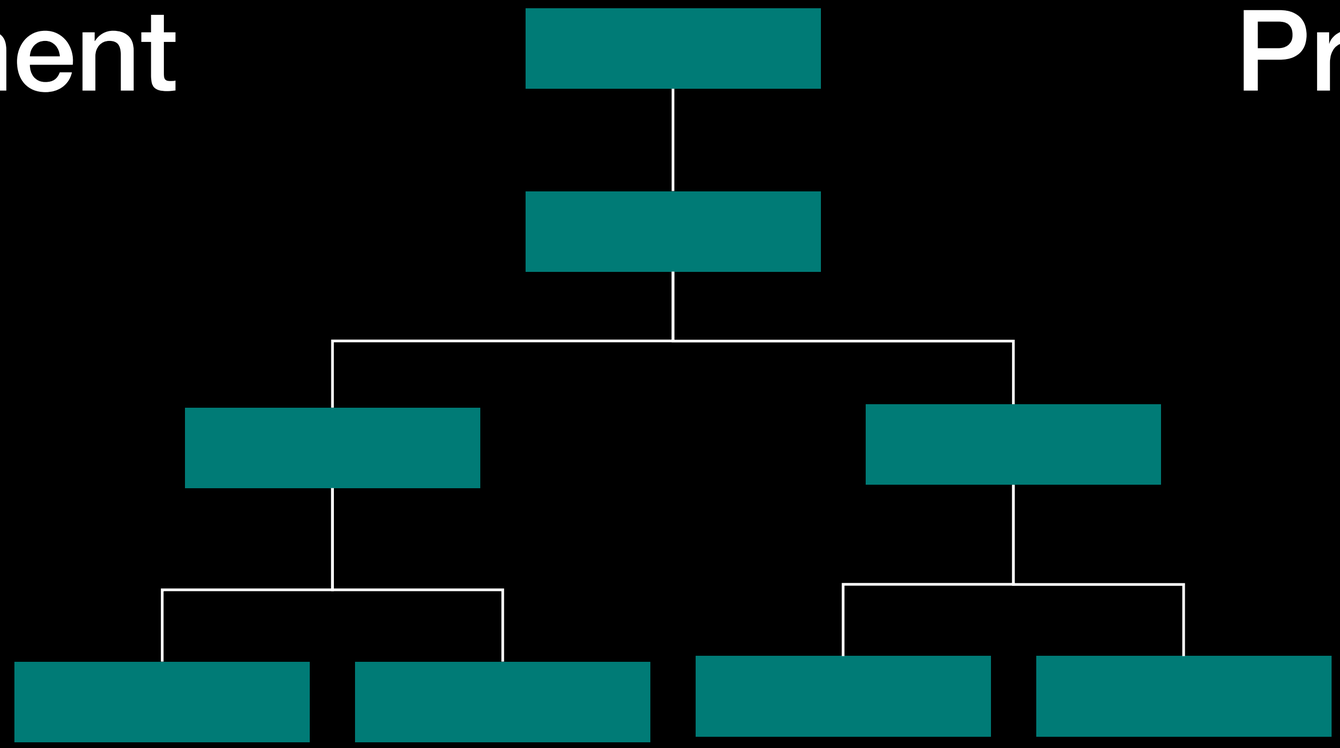
```
appView.environment(\.isRedacted, true)
```



```
@Environment(\.isRedacted) private var isRedacted
```

FLUXO DE DADOS

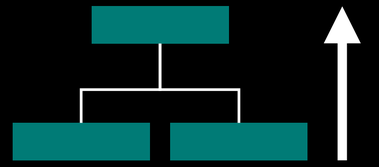
Environment



Preferences

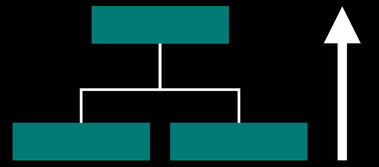
PREFERENCES

PREFERENCES

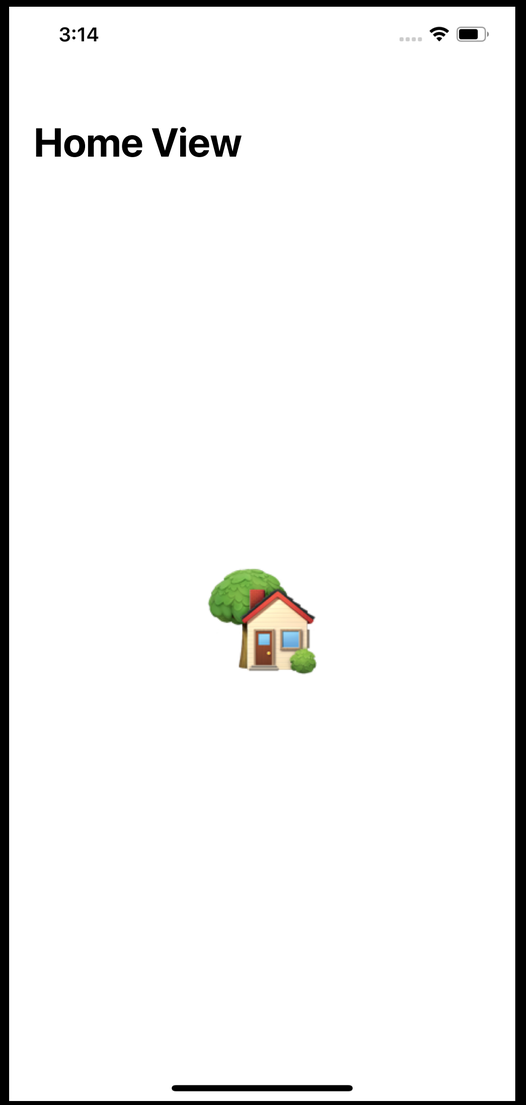


Mecanismo para uma view expressar suas **preferencias** para seus **ascendentes**.

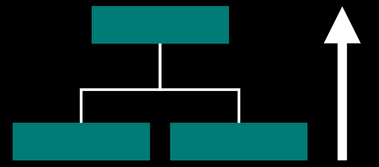
PREFERENCES



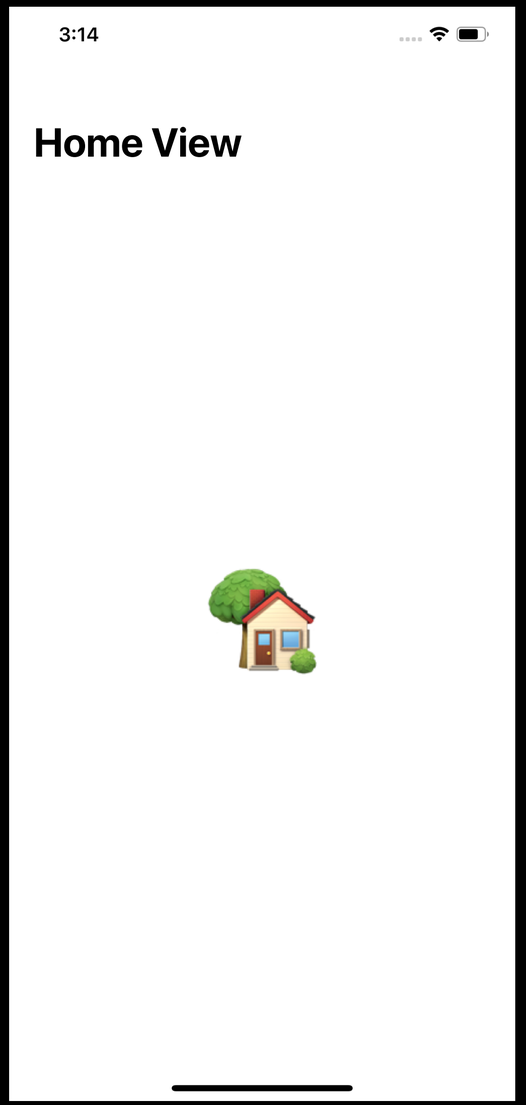
```
struct ContentView: View {
    var body: some View {
        NavigationView {
            HomeView()
        }
    }
}

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



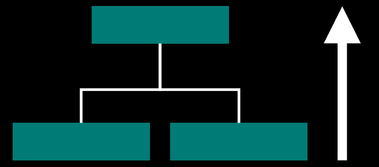
PREFERENCES



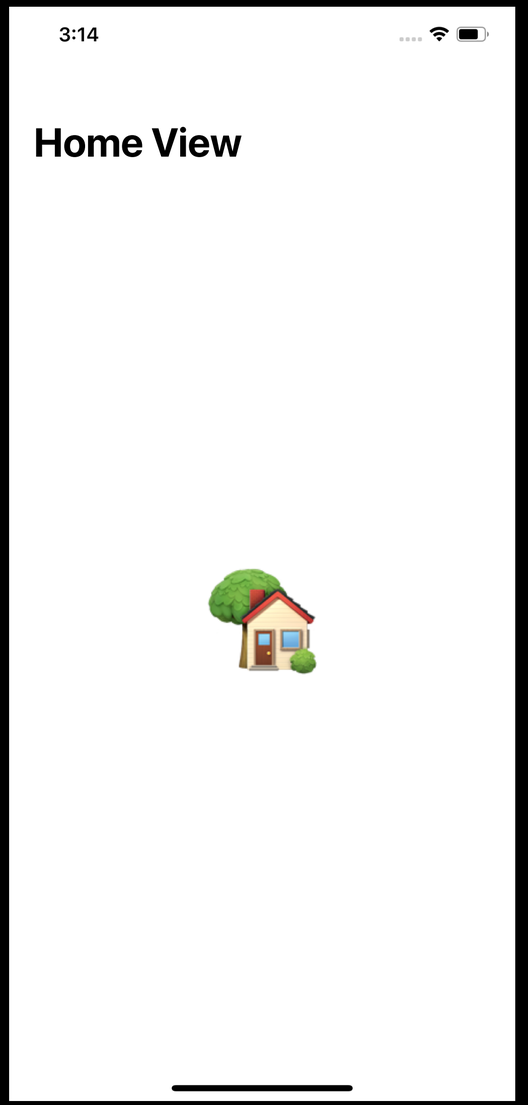
```
struct ContentView: View {
    var body: some View {
        NavigationView {
            HomeView()
        }
    }
}

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



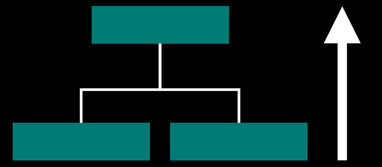
PREFERENCES

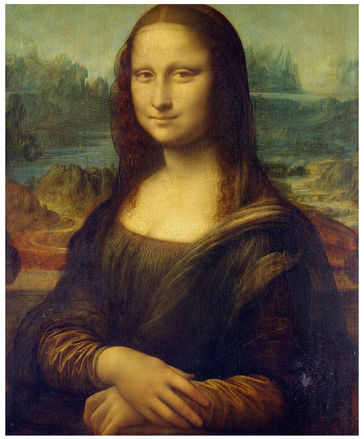


```
struct ContentView: View {
    var body: some View {
        NavigationView {
            HomeView()
        }
    }
}

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



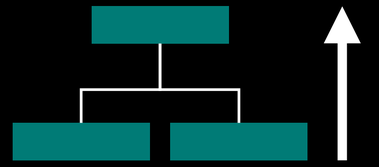
PREFERENCES





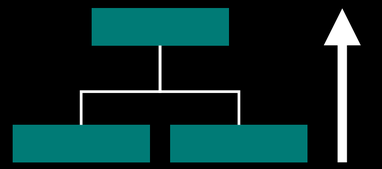
Mona Lisa

PREFERENCES



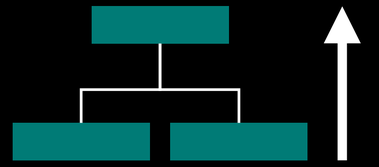
```
struct ContentView: View {
    var body: some View {
        PictureFrame {
            Monalisa()
        }
    }
}

struct Monalisa: View {
    var body: some View {
        Image("monalisa.png")
            .pictureTitle("Mona Lisa")
    }
}
```

PREFERENCES



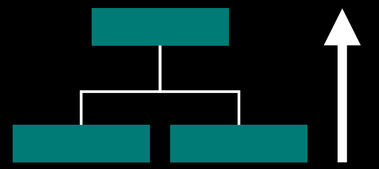
```
struct ContentView: View {
    var body: some View {
        PictureFrame {
            Monalisa()
        }
    }
}

struct Monalisa: View {
    var body: some View {
        Image("monalisa.png")
           .preference(
                key: TitlePrefKey.self,
                value: "Mona Lisa"
            )
    }
}
```

PREFERENCES



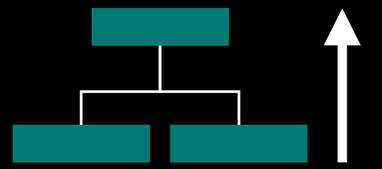
```
extension View {

    /// Returns a view producing `value` as the value of preference
    /// `Key` seen by its ancestors.
    @inlinable public func preference<K>(key _: K.Type = K.self, value: K.Value)
    -> some View where K : PreferenceKey

}
```

PREFERENCES



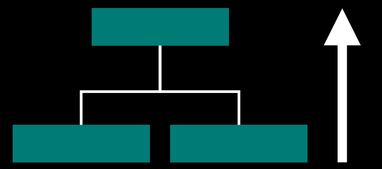
```
struct TitlePrefKey: PreferenceKey {
    typealias Value = String?

    static var defaultValue: Value = nil

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

PREFERENCES



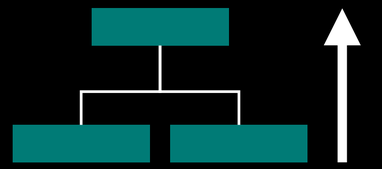
```
struct ContentView: View {
    var body: some View {
        PictureFrame {
            Monalisa()
        }
    }
}

struct Monalisa: View {
    var body: some View {
        Image("monalisa.png")
           .preference(
                key: TitlePrefKey.self,
                value: "Mona Lisa"
            )
    }
}
```

PREFERENCES



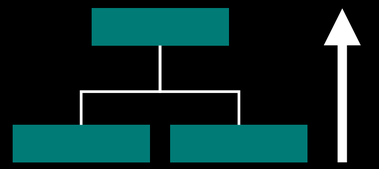
```
struct ContentView: View {
    var body: some View {
        PictureFrame {
            Monalisa()
        }
    }
}

struct Monalisa: View {
    var body: some View {
        Image("monalisa.png")
           .preference(
                key: TitlePrefKey.self,
                value: "Mona Lisa"
            )
    }
}
```

PREFERENCES

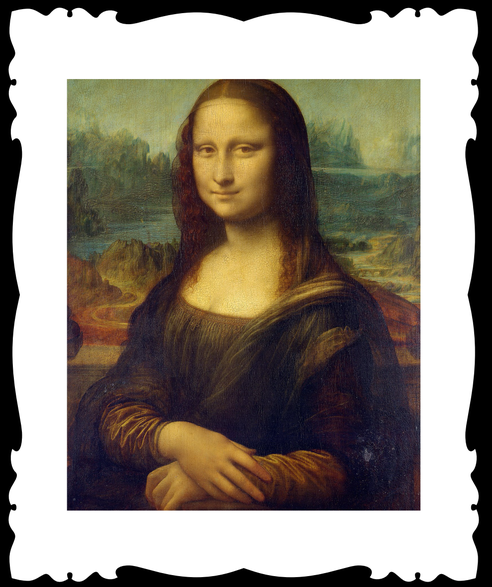


```
struct PictureFrame<Content: View>: View {
    let content: () -> Content

    var body: some View {
        VStack {
            content()
        }
    }
}
```

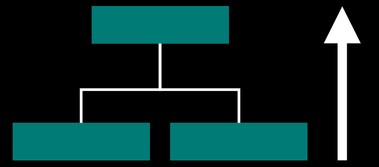
PREFERENCES



```
struct PictureFrame<Content: View>: View {
    let content: () -> Content

    var body: some View {
        VStack {
            content()
                .overlay(
                    Image("frame.png")
                )
        }
    }
}
```

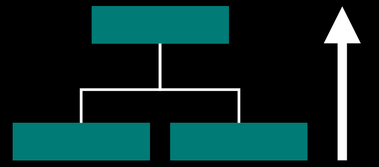


PREFERENCES



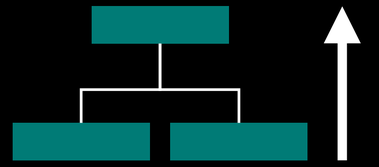
```
struct PictureFrame<Content: View>: View {
    let content: () -> Content

    var body: some View {
        VStack {
            content()
                .overlay(
                    Image("frame.png")
                )
                .onPreferenceChange(TitlePrefKey.self) { preference in

                }
        }
    }
}
```

PREFERENCES



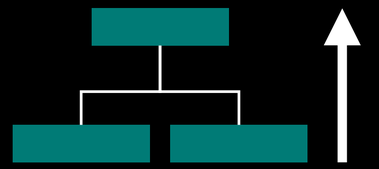
```
struct PictureFrame<Content: View>: View {
    let content: () -> Content

    @State private var title: String?

    var body: some View {
        VStack {
            content()
                .overlay(
                    Image("frame.png")
                )
                .onPreferenceChange(TitlePrefKey.self) { preference in
                    self.title = preference
                }
        }
    }
}
```

PREFERENCES



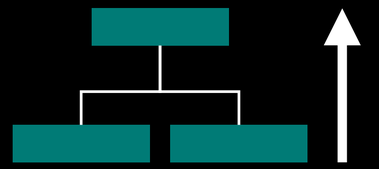
```
struct PictureFrame<Content: View>: View {
    let content: () -> Content

    @State private var title: String?

    var body: some View {
        VStack {
            content()
                .overlay(
                    Image("frame.png")
                )
                .onPreferenceChange(TitlePrefKey.self) { preference in
                    self.title = preference
                }
                .overlay(
                    VStack {
                        Spacer()
                        Text(title ?? "Obra sem nome")
                    }
                )
        }
    }
}
```

PREFERENCES



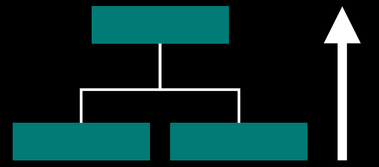
```
struct PictureFrame<Content: View>: View {
    let content: () -> Content

    @State private var title: String?

    var body: some View {
        VStack {
            content()
                .overlay(
                    Image("frame.png")
                )
                .onPreferenceChange(TitlePrefKey.self) { preference in
                    self.title = preference
                }
                .overlay(
                    VStack {
                        Spacer()
                        Text(title ?? "Obra sem nome")
                    }
                )
        }
    }
}
```

PREFERENCES



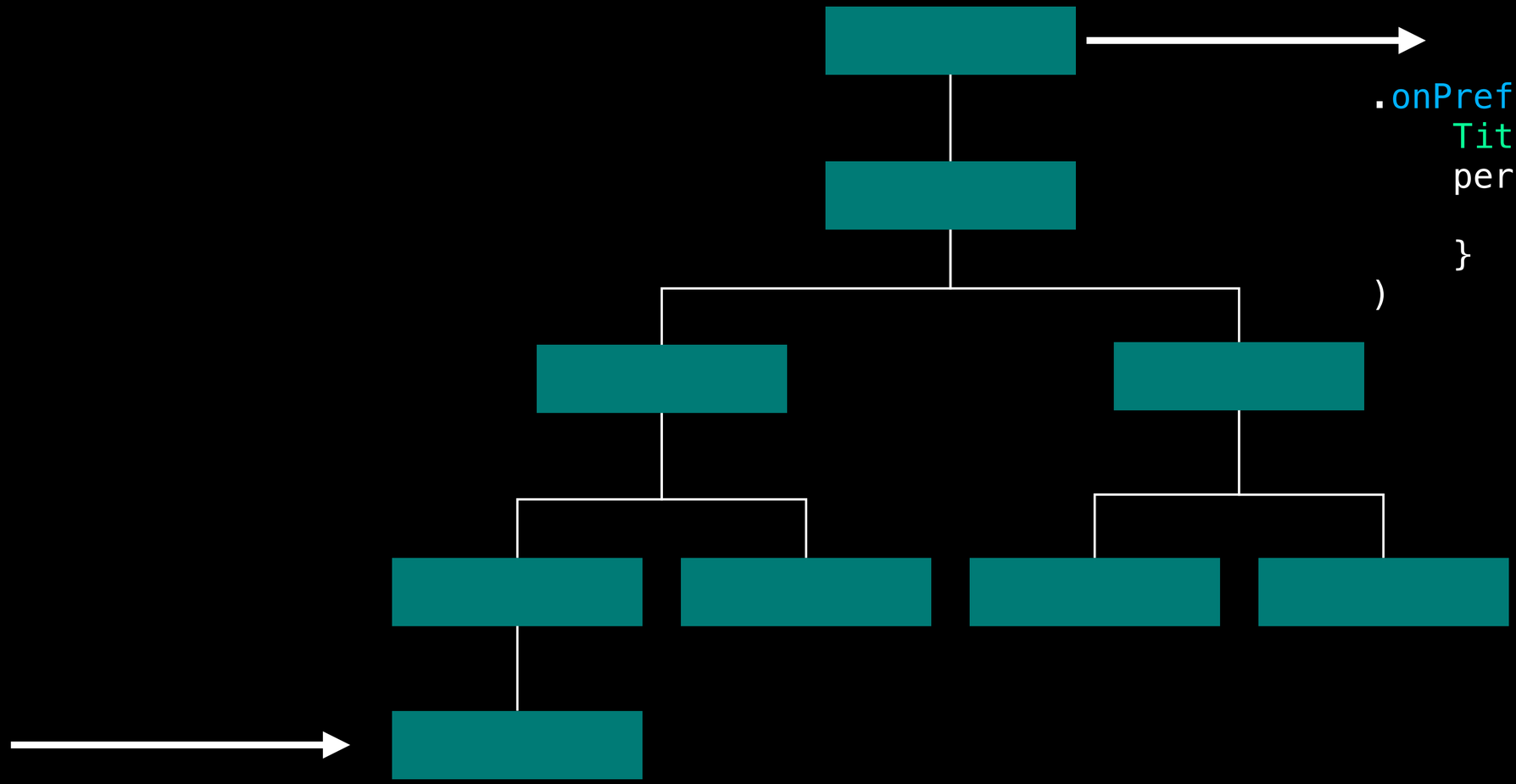
```
struct PictureFrame<Content: View>: View {
    let content: () -> Content

    @State private var title: String?

    var body: some View {
        VStack {
            content()
                .overlay(
                    Image("frame.png")
                )
                .overlayPreferenceValue(TitlePrefKey.self) { title in
                    VStack {
                        Spacer()
                        Text(title ?? "Obra sem nome")
                    }
                }
        }
    }
}
```

PREFERENCES



```
struct PictureFrame<Content: View>: View {
    let content: () -> Content

    var body: some View {
        VStack {
            content()
                .overlay(
                    Image("frame.png")
                )
                .overlayPreferenceValue(TitlePrefKey.self) { title in
                    VStack {
                        Spacer()
                        Text(title ?? "Obra sem nome")
                    }
                }
        }
    }
}
```

PREFERENCES



```
.onPreferenceChange(
    TitlePrefKey.self,
    perform: { preference in
        self.title = preference
    }
)
```

```
.preference(key: TitlePrefKey.self, value: "Mona Lisa")
```

PREFERENCES

RESUMO

RESUMO

✦ O **menor** caminho para construir **ótimos** apps para **todos** dispositivos.

✦ **Layout** é decidir o **tamanho** das coisas na tela

✦ Não existe maneira de **forçar** um tamanho no seu **filho**.

1. Pai **propõem** um tamanho ao filho
2. Filho **escolhe** seu próprio tamanho
3. Pai posiciona filho na **coordenada** do pai

✦ **Views** são como **receitas**, use as ferramentas para expressar o que deve ser feito durante o layout.

✦ **Environment** é o mecanismo para uma view passar informações para seus filhos.

✦ **Preferences** é o mecanismo para uma view passar informações para seus pais.

Muito Obrigado!

@txaiwieser

Twitter | Github | LinkedIn | Slack

Muito Obrigado!

Perguntas, dicas e sugestões!

@txaiwieser

Twitter | Github | LinkedIn | Slack