SWIFTUI I.

Jan Kodeš, iOS Developer @ STRV

STRV

CO JE SWIFTUI?

01

CO JE SWIFTUI

- UI framework a nástroje vytvořené Applem
- Představeno v 2019 (SwiftUI 2.0 - 2020)
- Nové = stále se vyvíjí
- Deklarativní programování
- Combine
- Crossplatform (MacOS + iOS)
- Žádné Storyboardy ani Interface Builder

```
struct ContentView: View {
    var body: some View {
        HStack(alignment: .top) {
            VStack {
                CalendarView()
                Spacer()
            }
            VStack(alignment: .leading) {
                Text("Event title").font(.title)
                Text("Location")
            }
            Spacer()
        }.padding()
    }
}
```

SWIFTUI vs UIKIT

SwiftUI

- Lze kombinovat s UIKitem
- iOS 13+
- Změny mezi verzemi (1.0 vs 2.0)
- Rychlé prototypování
- Některé věci nelze vyřešit jinak než UIKitem

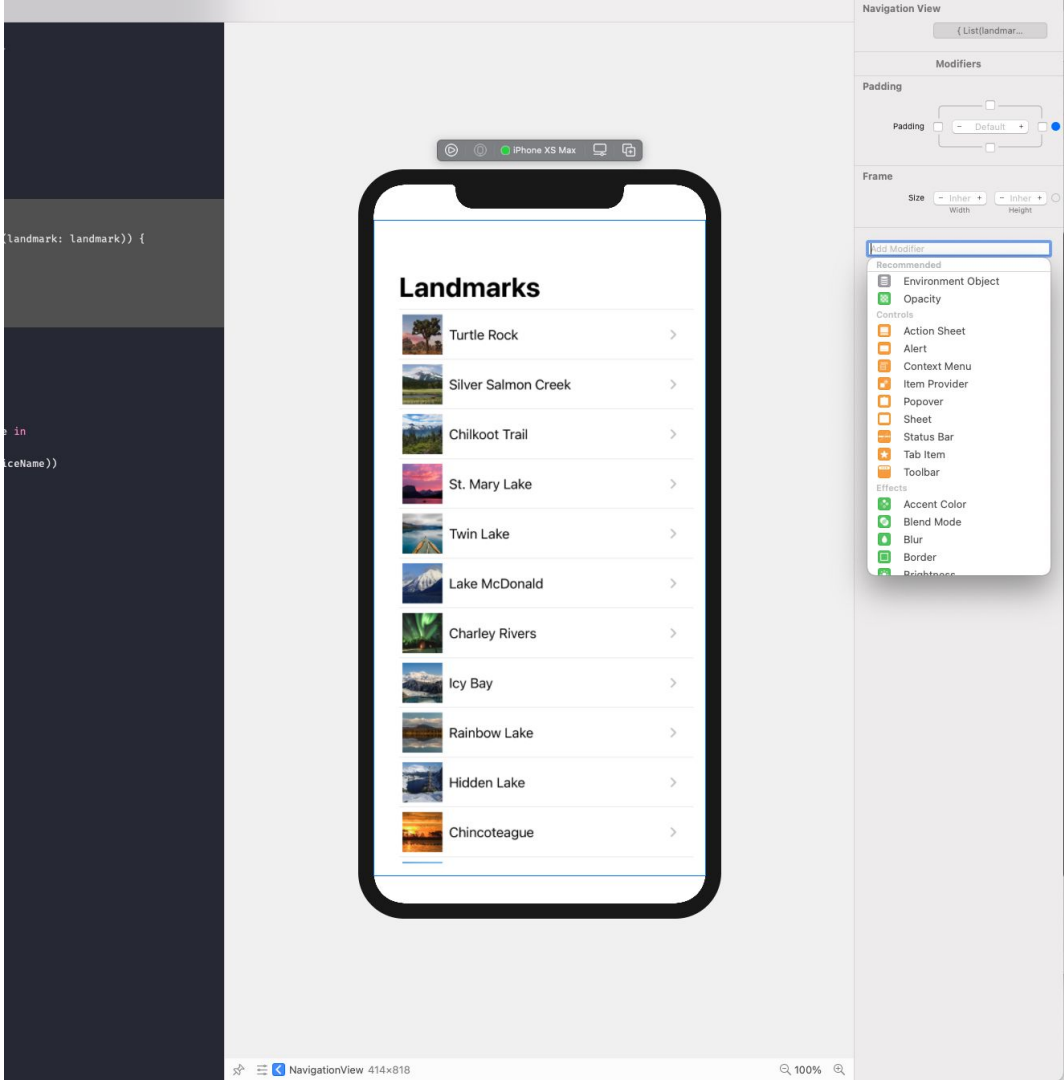
UIKit

- Zůstane i nadále podporován
- Vývoj je pocitově pomalejší oproti SwiftUI
- Autolayout
- Horší podpora Combine


IOS DEVELOPERS
WHEN SWIFTUI IS RELEASED
UIKIT
"I don't want to play with you anymore"
imgflip.com

XCODE + SWIFTUI

- Canvas
- Previews
- Založení projektu - UIKit nebo SwiftUI



STRV

SWIFTUI VIEWS

02

SWIFTUI VIEWS

- Jednoduché struktury (Value type)
- Odlišný lifecycle oproti UIKitu
- body { }
- some View (Opaque Type)
- Využívají Result Builders (@Viewbuilder)
- Modifikátory
- Pro renderování se částečně využívá UIKit

```
struct ContentView: View {
    var body: some View {
        Text("Hello World")
    }
}
```

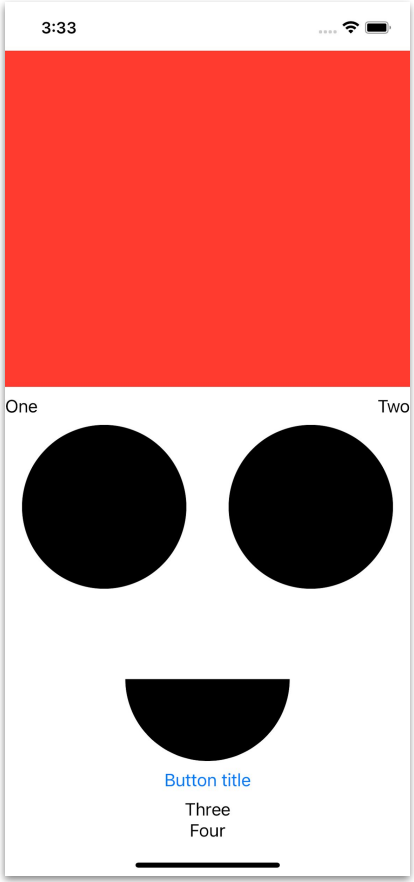
```
VStack {
    Text("Hello")
    Text("World")
    Button("I'm a button") {}
}
// 'VStack<TupleView<(Text, Text, Button<Text>)>>'

// Bez ResultBuilder
var builder = VStackBuilder()
builder.add(Text("Hello")
builder.add(Text("World"))
builder.add(Button("I'm a button"))
return builder.build()
```

SWIFTUI X UIKIT

- View x UIViewController
- HStack, VStack, ZStack x UIStackView
- Button x UIButton
- Text, Label x UILabel
- Image x UIImageView
- ScrollView x UIScrollView
- LazyVGrid, LazyHGrid x UICollectionView
- TabView x UITabBarController

SwiftUI Cheatsheet

```
struct ContentView: View {
    var body: some View {
        VStack {
            Color.red

            HStack {
                Text("One")
                Spacer()
                Text("Two")
            }

            Spacer()

            VStack {
                HStack {
                    Circle()
                    Circle()
                }
                Circle()
                    .trim(from: 0, to: 0.5)
            }

            Button("Button title") {}

            Spacer()

            VStack {
                Text("Three")
                Text("Four")
            }
        }
    }
}
```



VIEWMODIFIER

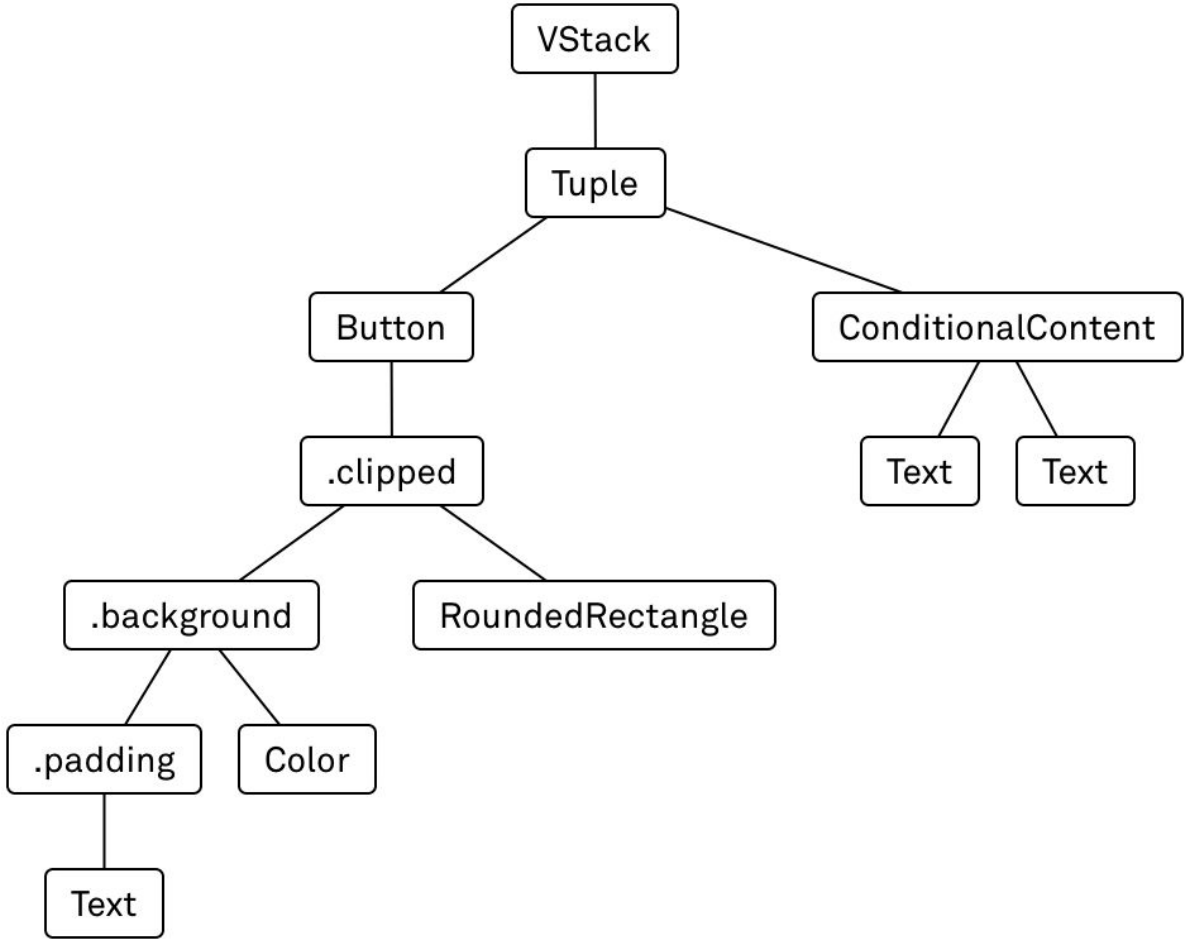
VIEWMODIFIER

```
func body(content: Self.Content) -> Self.Body
```

- Modifikátor, který vrátí upravené View
- Specifické a globální
- Lze definovat vlastní
- Pořadí je důležité!

```
struct ContentView: View {
    var body: some View {
        Text("Example")
            .font(.largeTitle)
            .background(Color.black)
            .foregroundColor(.white)
            .cornerRadius(8.0)
            .padding(.top, 50.0)
    }
}
```

VIEW TREE


VStack
Tuple
Button
ConditionalContent
.clipped
Text
Text
.background
RoundedRectangle
.padding
Color
Text

LAYOUT SYSTEM

LAYOUT SYSTEM

1. Rodič nabídne velikost potomkovi

Kořenové View nabídne Text danou velikost.

V tomto příkladu celou obrazovku.

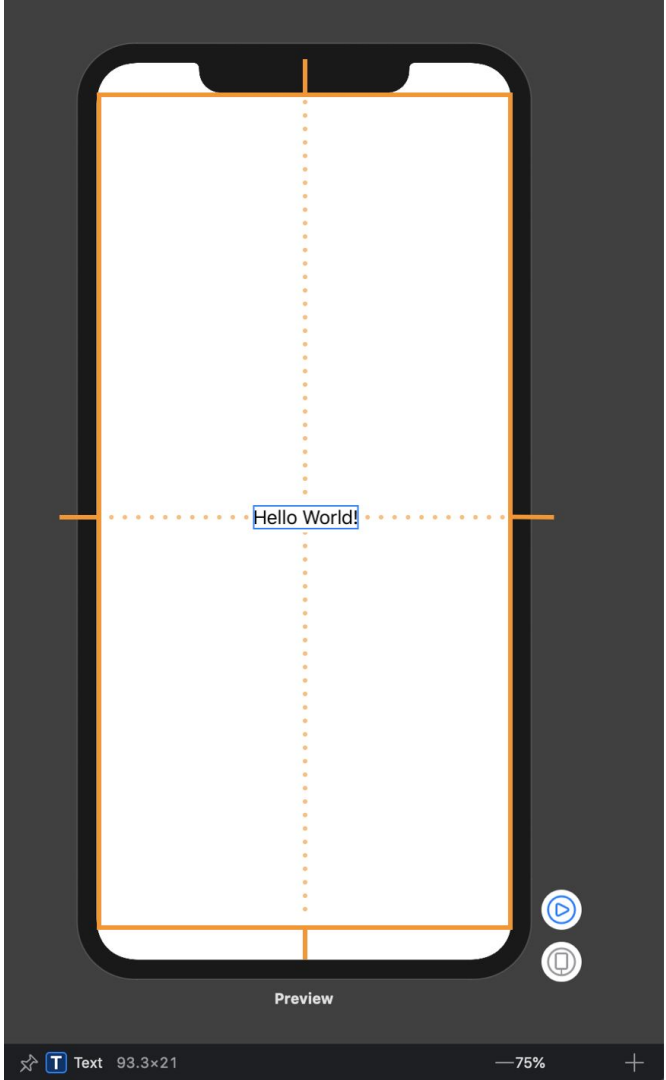
2. Potomek rozhodne o své velikosti

Text určí svou velikost dle svého obsahu.

Rodič respektuje potomka, tzn. že neupravuje jeho velikost.

3. Rodič umístí potomka do hierarchie

Kořenové View musí potomka někam umístit a umístí ho doprostřed.



DEMO

CO SI ZAPAMATOVAT?

- View je Struct a skládá se z dalších View => View hierarchy
- Můžeme je upravovat pomocí ViewModifier
- Pořadí ViewModifier je důležité!
- Ne všechny elementy UIKitu mají svou paralelu ve SwiftUI
- View rozhoduje o své velikosti
- View by mělo být jednoduché => kompozice

DATA FLOW

03

@ENVIRONMENTOBJECT
@OBSERVEDOBJECT
@ENVIRONMENT
@STATEOBJECT
@PUBLISHED
@BINDING
@STATE

@PROPERTYWRAPPER

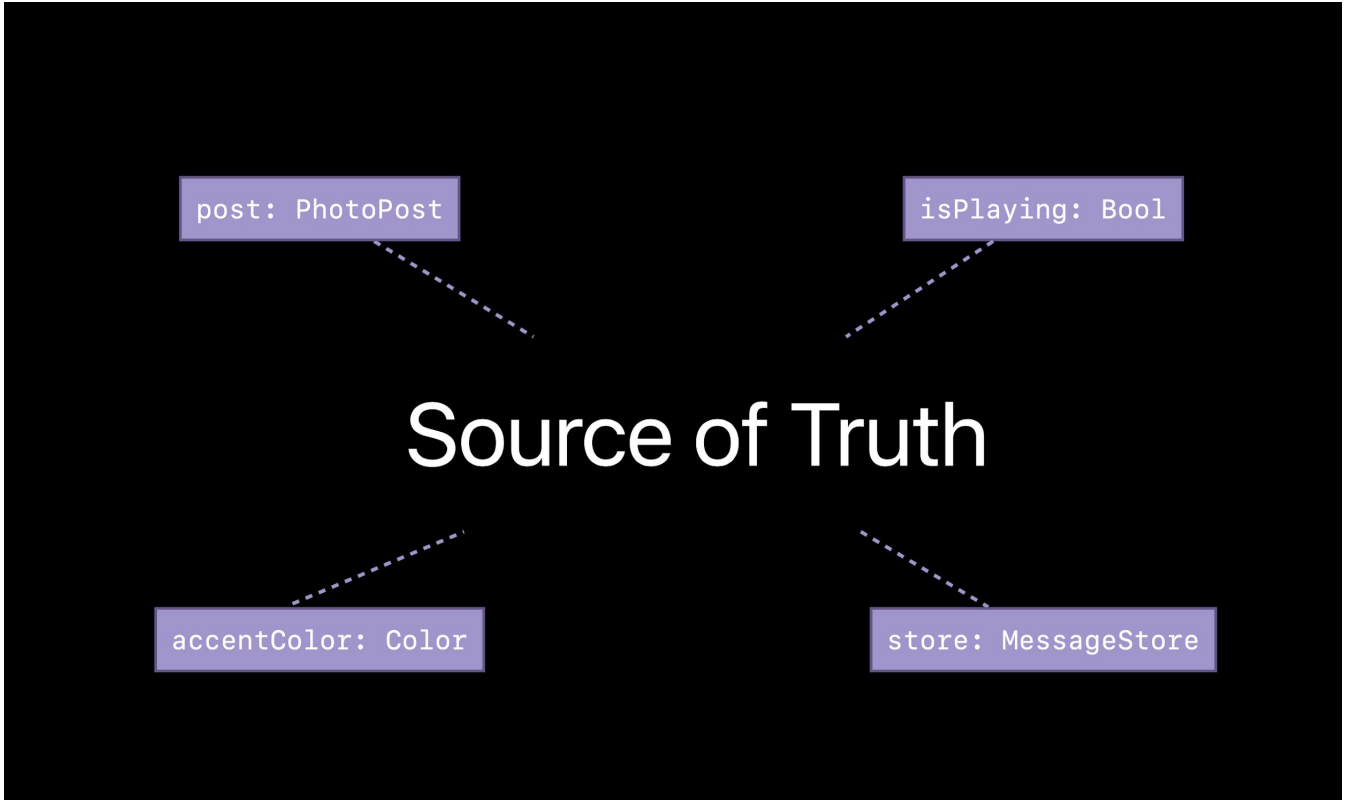
PROPERTY WRAPPER

- Rozšiřuje proměnné o námi definovanou logiku
- Přidává přehlednost a znovupoužitelnost kódu
- Při každé změně hodnoty se provede definovaná operace
- Např. @AppStorage, @SceneStorage...

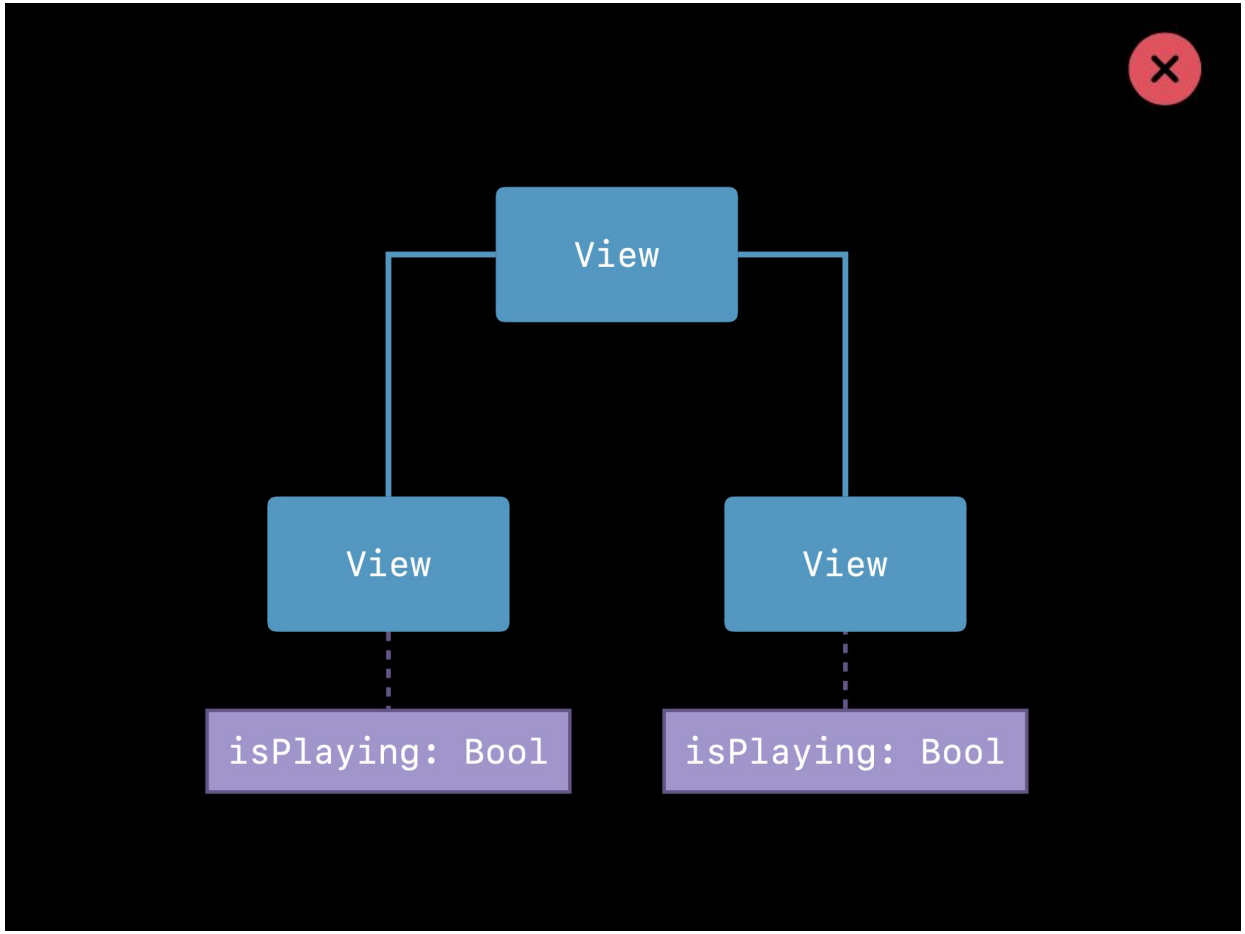
Seznam PropertyWrappers

```
// Definition
@propertyWrapper
struct Capitalized {
    var wrappedValue: String {
        didSet { wrappedValue = wrappedValue.capitalized }
    }

    init(wrappedValue: String) {
        self.wrappedValue = wrappedValue.capitalized
    }
}

// Usage
struct User {
    @Capitalized var firstName: String
    @Capitalized var lastName: String
}
```

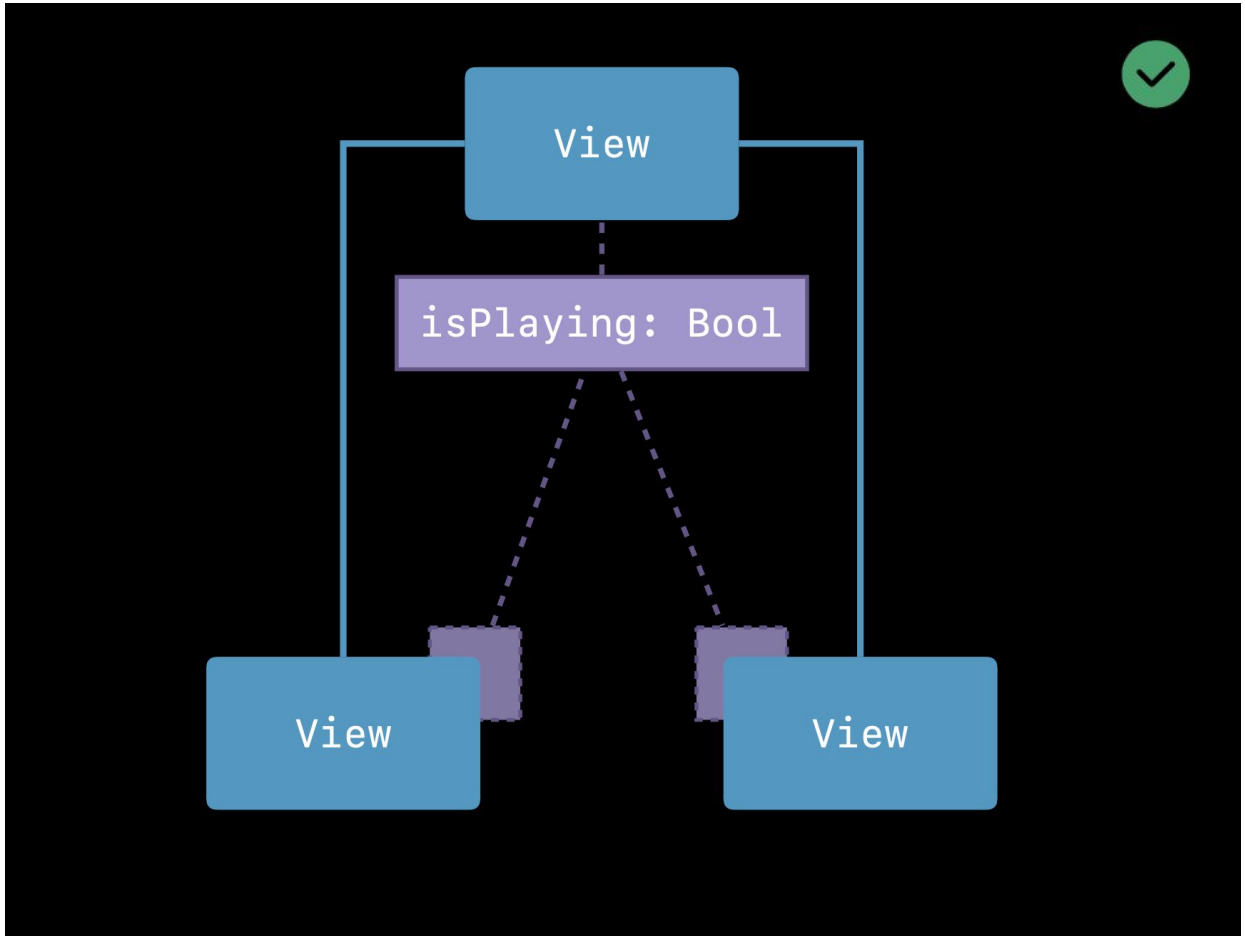
SINGLE SOURCE OF TRUTH



Zdroj: developer.apple.com



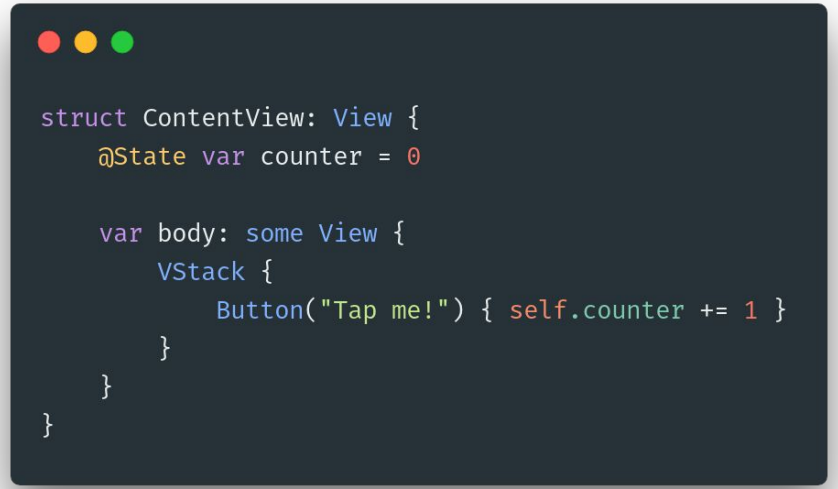
Zdroj: developer.apple.com



Zdroj: developer.apple.com

@STATE

- Lze použít pouze uvnitř View
- Reprezentuje interní stav View
- Ideálně pro jednodušší datové typy (Value types)
- private



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

    var body: some View {
        VStack {
            Button("Tap me!") { self.counter += 1 }
        }
    }
}
```

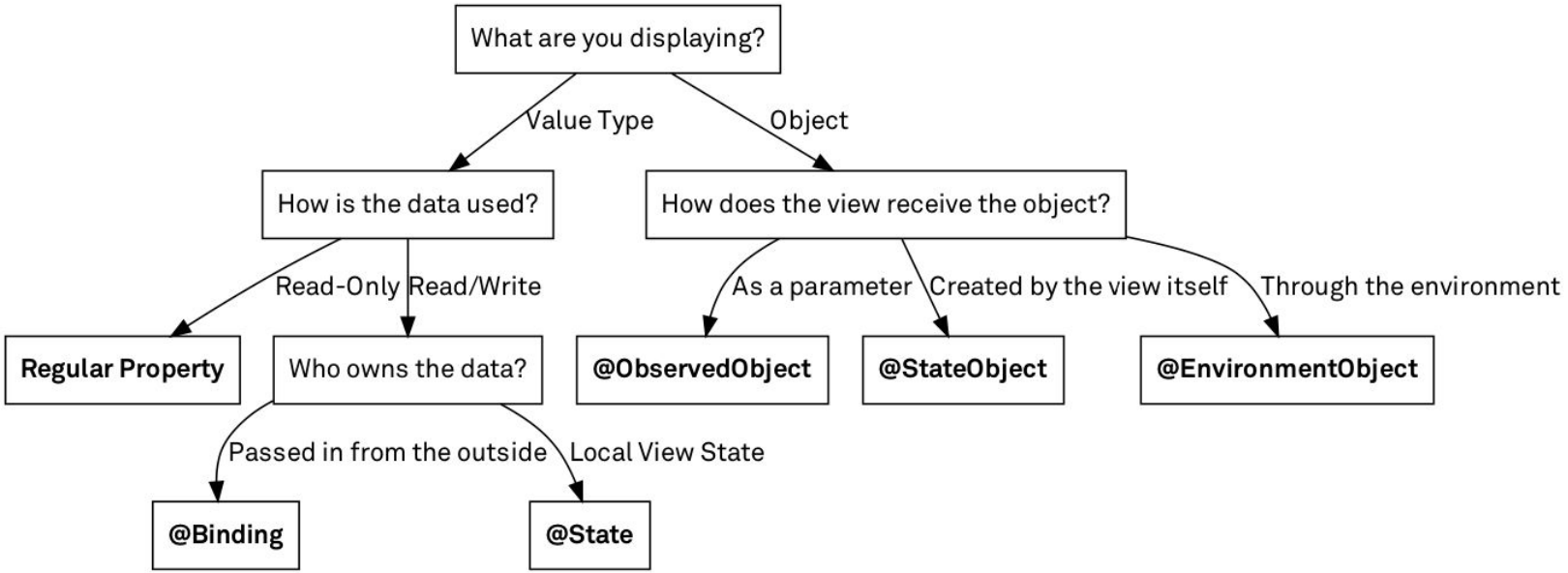
@BINDING

- Předávání hodnot (hlavně @State)
- Source of Truth
- Slider, TextField…
- Two-way binding

```
$counter -> Binding<Int>
counter  -> Int
```

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

    var body: some View {
        VStack {
            Button("Tap me!") { self.counter += 1 }
            LabelView(number: $counter)
        }
    }
}

struct LabelView: View {
    @Binding var counter: Int

    ...
}
```

JAKÝ WRAPPER POUŽÍT?



https://swiftuipropertywrappers.com/

DEMO

CO SI ZAPAMATOVAT?

- Single source of truth
- Data ve View měním pomocí **@State**
- **@State** předávám pomocí **@Binding** ($stateProperty)
- .sheet() = .present() / .show()

VIEW LIFECYCLE

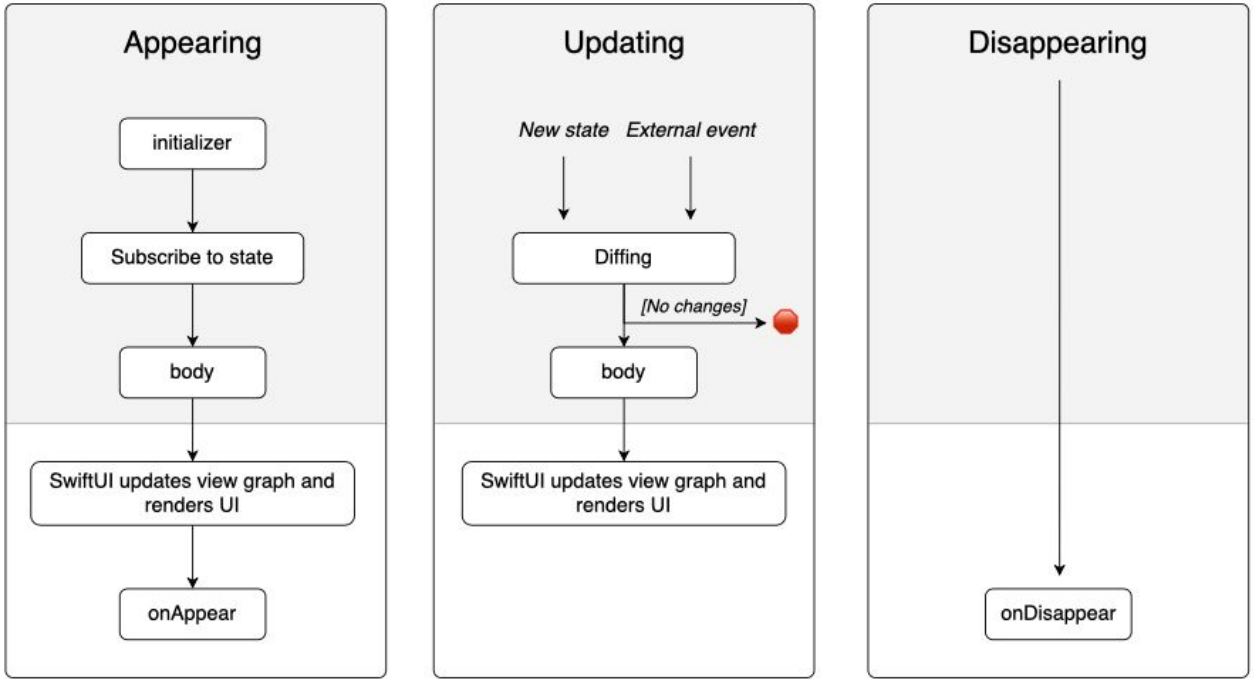
04

"Layout phase"

Must be pure and without side effects. May be restarted or aborted by SwiftUI.

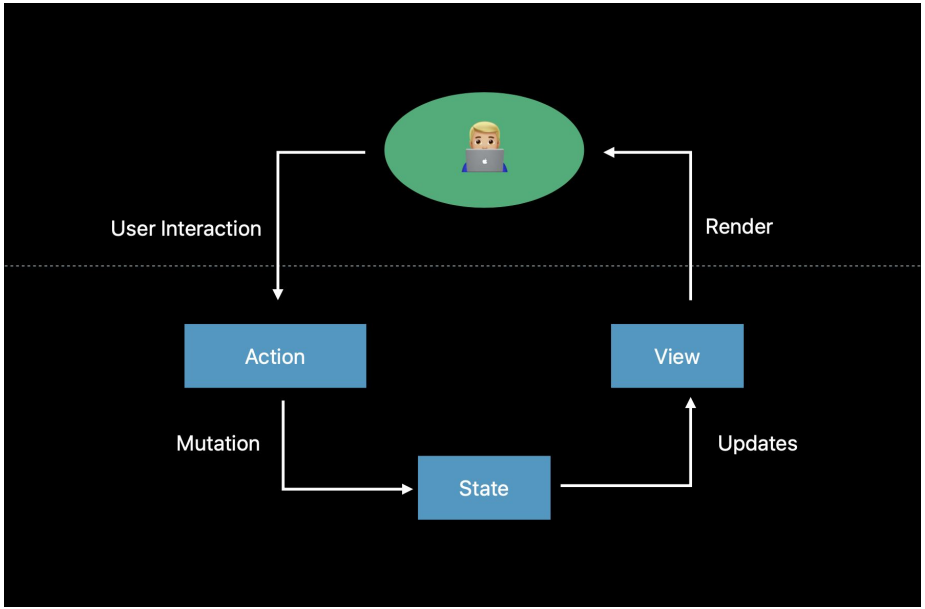
"Commit phase"

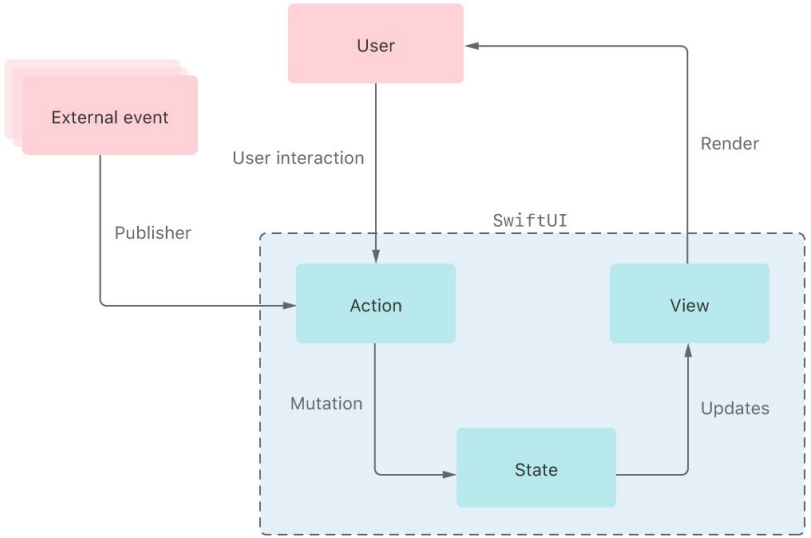
Can run side effects and trigger view graph updates.



VIEW UPDATES

- SwiftUI renderuje obsah body { }
- Výsledkem je virtuální strom View + jeho potomků
- Při každé změně dat ve View dochází k přepočítání změn celého stromu (tzv. diffing)
- Nedochází k překreslení celého stromu, ale pouze tam, kde je to potřeba
- Renderování je levné díky hodnotovému typu (Struct)
- Úpravu View lze udělat pouze pomocí @State (@ObservedObject, @EnvironmentObject…)





Zdroj: developer.apple.com

CO SI ZAPAMATOVAT?

- Přerenderování probíhá na základě výpočtu změn
- Je rychlé i díky tomu, že SwiftUI zná celou hierarchii již z kompilace
- *View* nemá klasickou metodu *viewDidLoad*

DĚKUJI ZA POZORNOST!

Jan Kodeš / jan.kodes@strv.com

STRV

ZAJÍMAVÉ ZDROJE

- https://developer.apple.com/tutorials/swiftui/
- https://www.hackingwithswift.com/books/ios-swiftui
- https://developer.apple.com/videos/play/wwdc2019/226/
- https://developer.apple.com/videos/play/wwdc2020/10040/
- https://swiftwithmajid.com/
- https://www.vadimbulavin.com/tag/swiftui/
- https://github.com/ygit/swiftui