Swift, SwiftUI & Environment 🌍

Swift Features behind SwiftUI Interface

Ratnesh Jain

UIKit

Imperative Syntax

UIKit

Imperative Syntax

- Explicitly tell UIKit how each UI element step by step.
- Need to implement UI element for specific target platform.
  - iPadOS, watchOS, tvOS, macOS, visionOS …
- Need to layout manually (Autolayout)

SwiftUI

Declarative Syntax

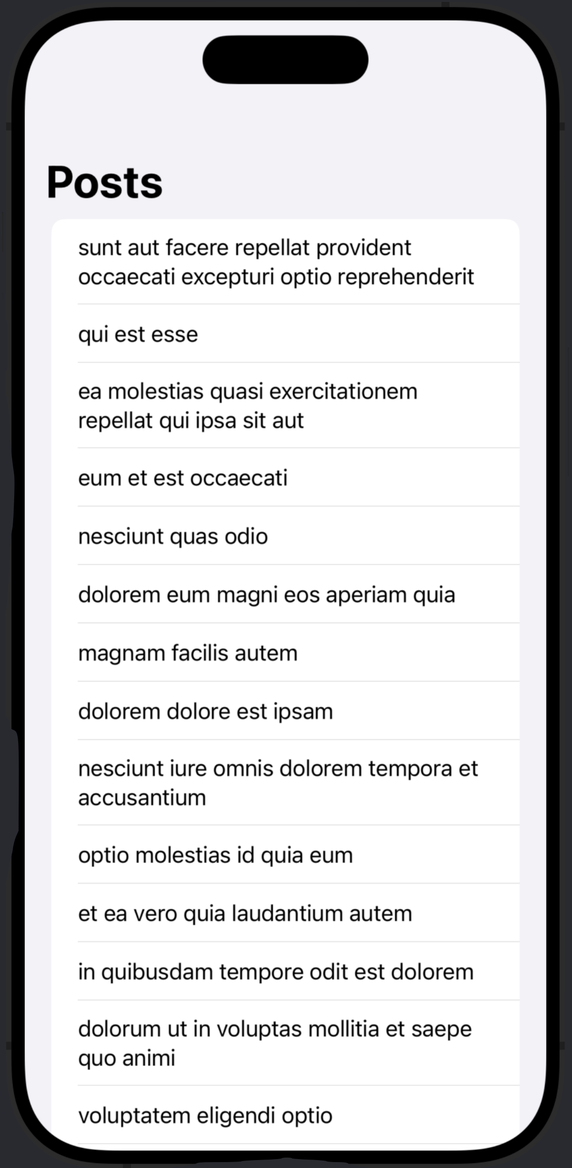
Ratnesh Jain

SwiftUI

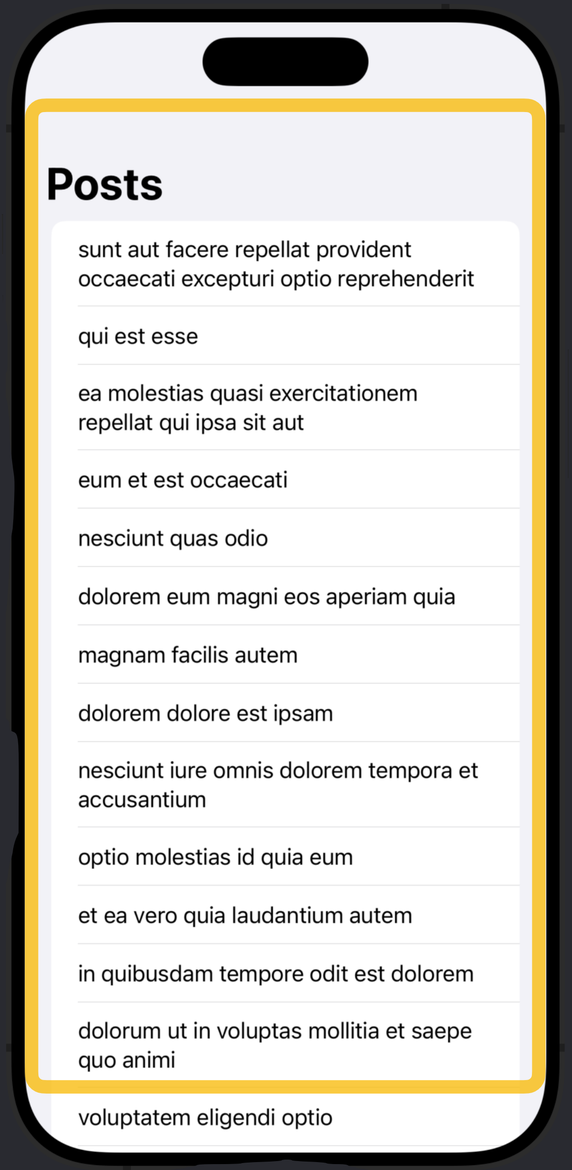
Declarative Syntax (DSL)

- We only need to declare
- UI is function of State
  - To change UI, must change State
  - If state is changed from external event, UI will be in sync 🪄
- SwiftUI supports Multi-platform development (platform)
  - SwiftUI will adjust the rendering of UI for target platform 😎
- No Autolayout 🥺

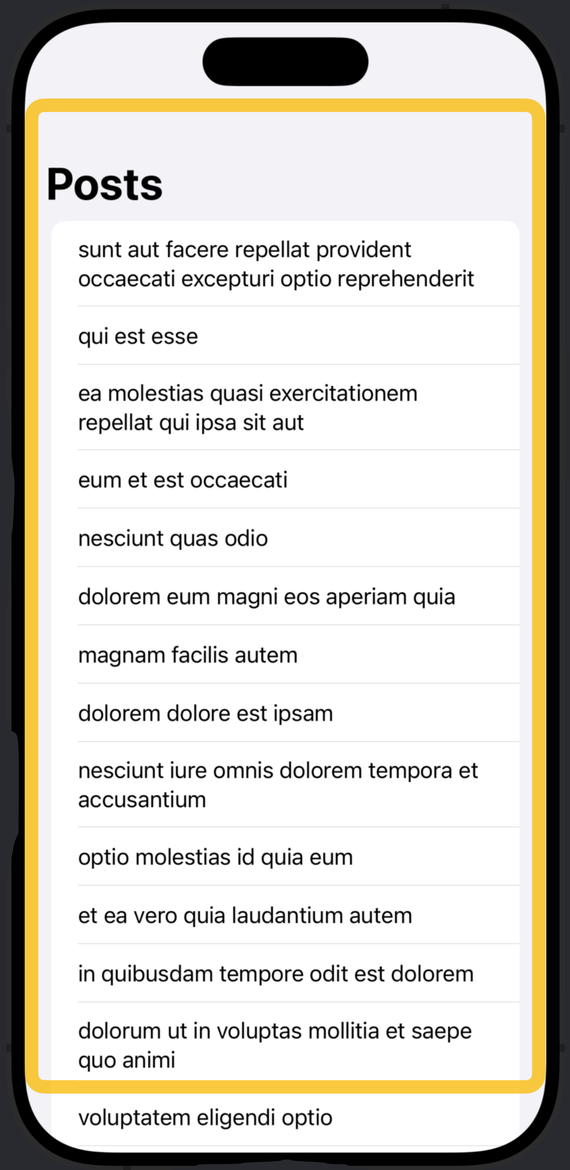
```swift
import SwiftUI

@MainActor
struct ContentView: View {
    @State var viewModel: PostsViewModel = .init()

    var body: some View {
        List {
            ForEach(viewModel.posts, id: \.id) { post in
                Text(post.title)
            }
        }
        .task {
            await viewModel.fetchPosts()
        }
        .navigationTitle("Posts")
    }
}

#Preview {
    NavigationStack {
        ContentView()
    }
}
```



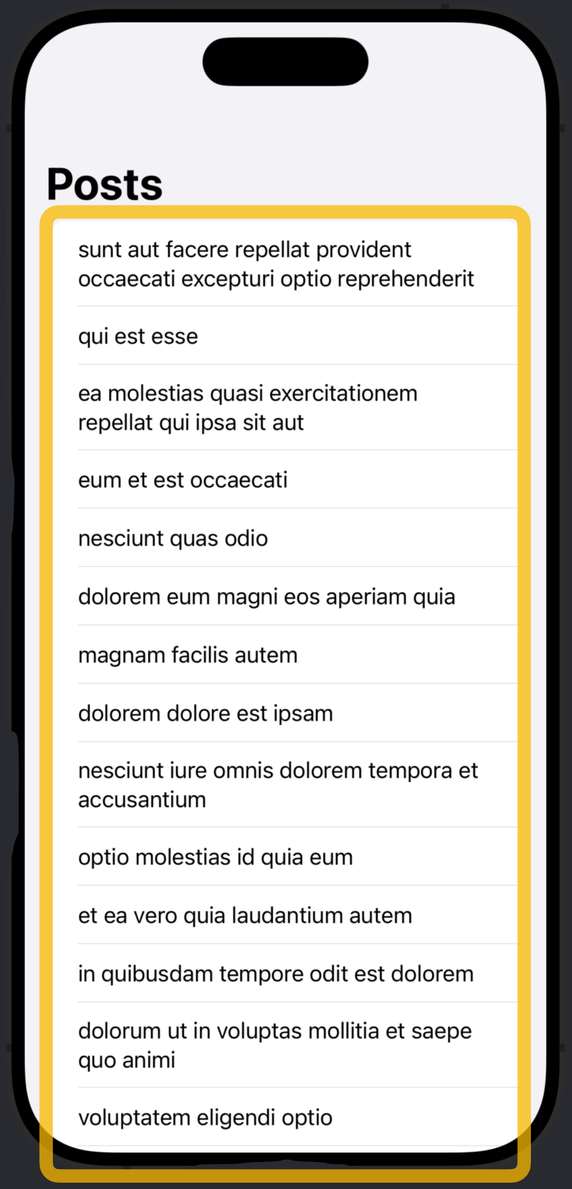
```swift
import SwiftUI

@MainActor
struct ContentView: View {
    @State var viewModel: PostsViewModel = .init()

    var body: some View {
        List {
            ForEach(viewModel.posts, id: \.id) { post in
                Text(post.title)
            }
        }
        .task {
            await viewModel.fetchPosts()
        }
        .navigationTitle("Posts")
    }
}

#Preview {
    NavigationStack {
        ContentView()
    }
}
```



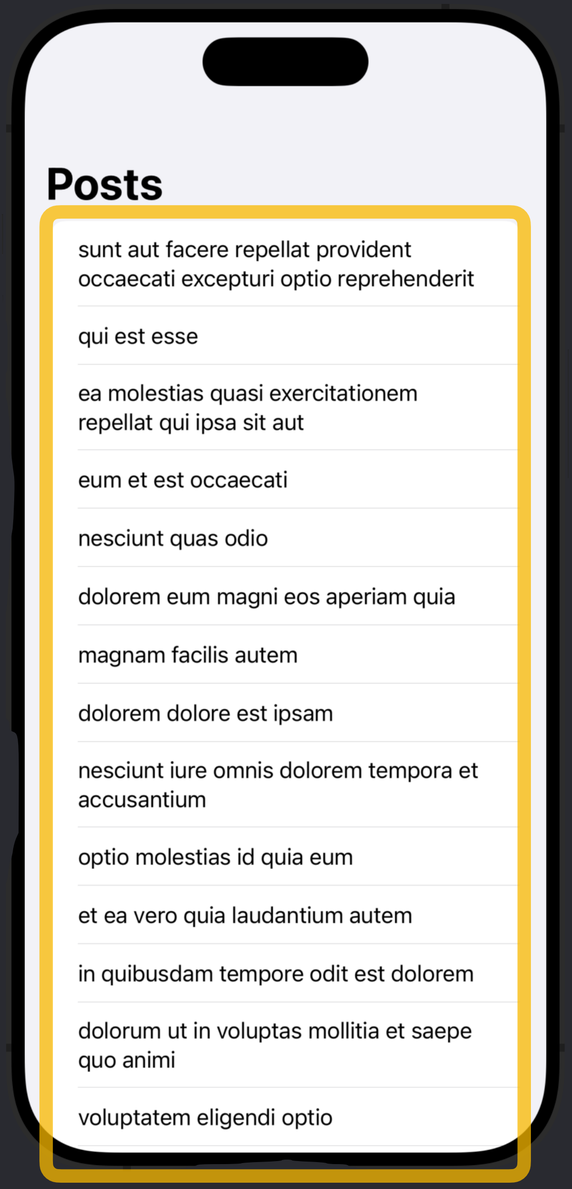
```swift
import SwiftUI

@MainActor
struct ContentView: View {
    @State var viewModel: PostsViewModel = .init()

    var body: some View {
        List {
            ForEach(viewModel.posts, id: \.id) { post in
                Text(post.title)
            }
        }
        .task {
            await viewModel.fetchPosts()
        }
        .navigationTitle("Posts")
    }
}

#Preview {
    NavigationStack {
        ContentView()
    }
}
```



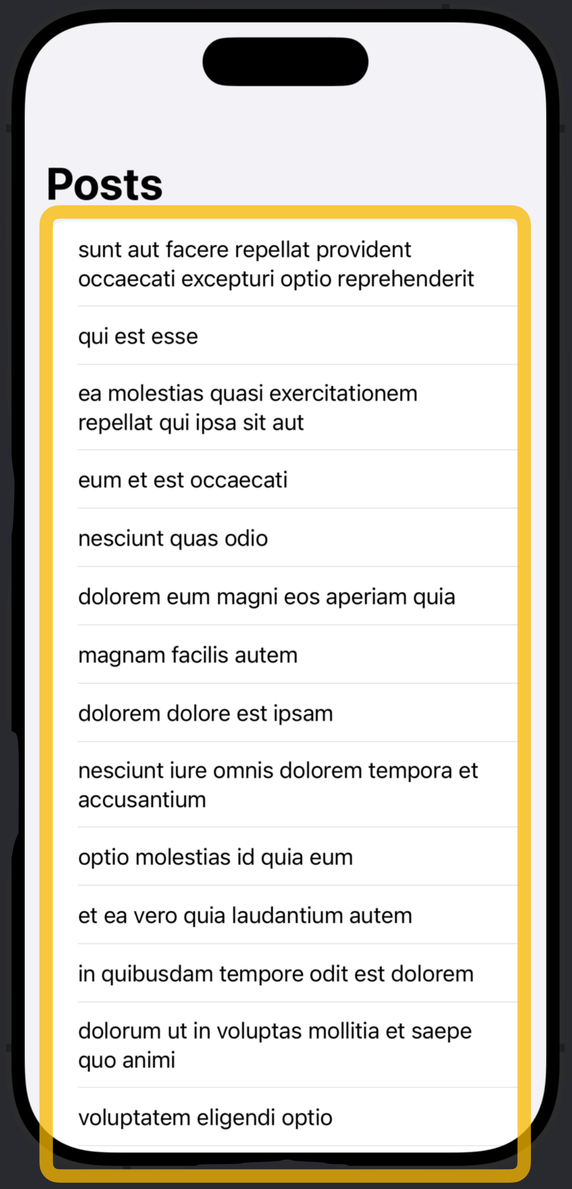
```swift
import SwiftUI

@MainActor
struct ContentView: View {
    @State var viewModel: PostsViewModel = .init()

    var body: some View {
        List {
            ForEach(viewModel.posts, id: \.id) { post in
                Text(post.title)
            }
        }
        .task {
            await viewModel.fetchPosts()
        }
        .navigationTitle("Posts")
    }
}

#Preview {
    NavigationStack {
        ContentView()
    }
}
```



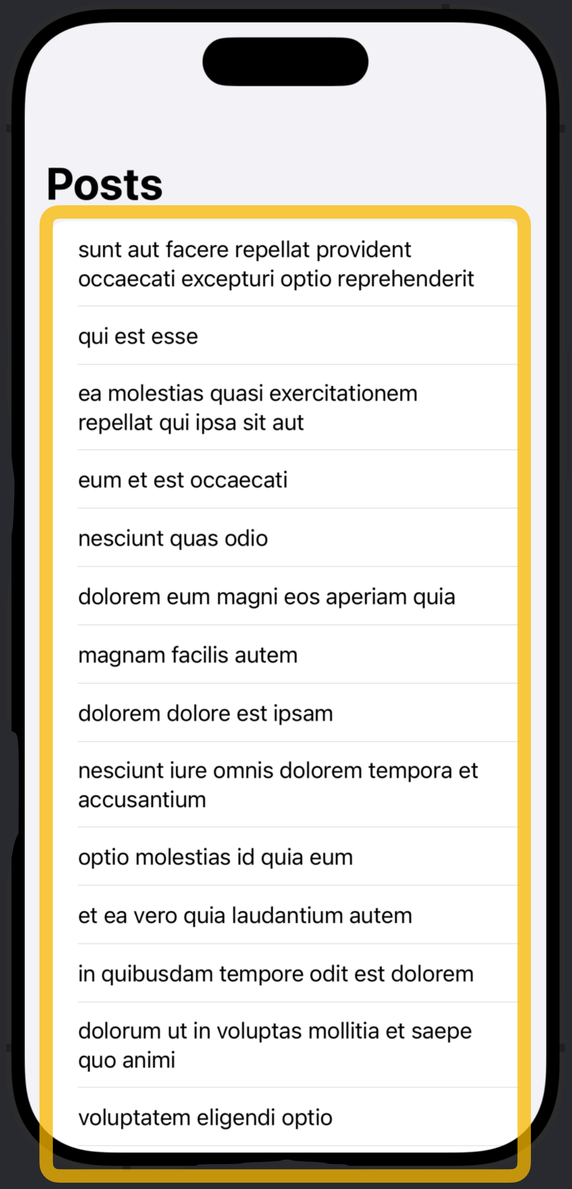
```swift
import SwiftUI

@MainActor
struct ContentView: View {
    @State var viewModel: PostsViewModel = .init()

    var body: some View {
        List {
            ForEach(viewModel.posts, id: \.id) { post in
                Text(post.title)
            }
        }
        .task {
            await viewModel.fetchPosts()
        }
        .navigationTitle("Posts")
    }
}

#Preview {
    NavigationStack {
        ContentView()
    }
}
```



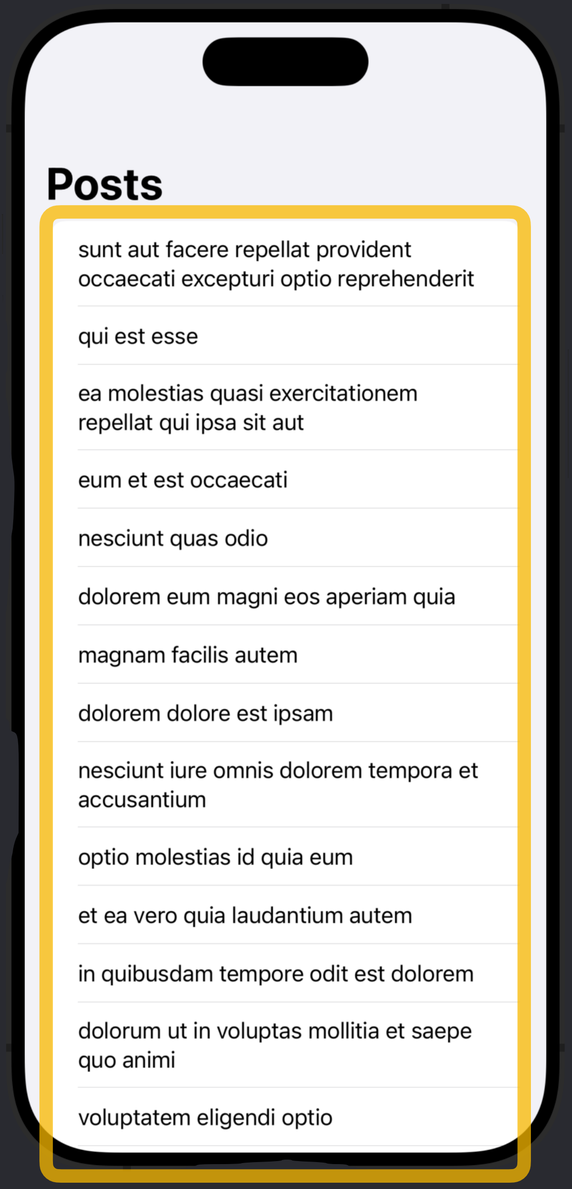
```swift
import SwiftUI

@MainActor
struct ContentView: View {
    @State var viewModel: PostsViewModel = .init()

    var body: some View {
        List {
            ForEach(viewModel.posts, id: \.id) { post in
                Text(post.title)
            }
        }
        .task {
            await viewModel.fetchPosts()
        }
        .navigationTitle("Posts")
    }
}

#Preview {
    NavigationStack {
        ContentView()
    }
}
```



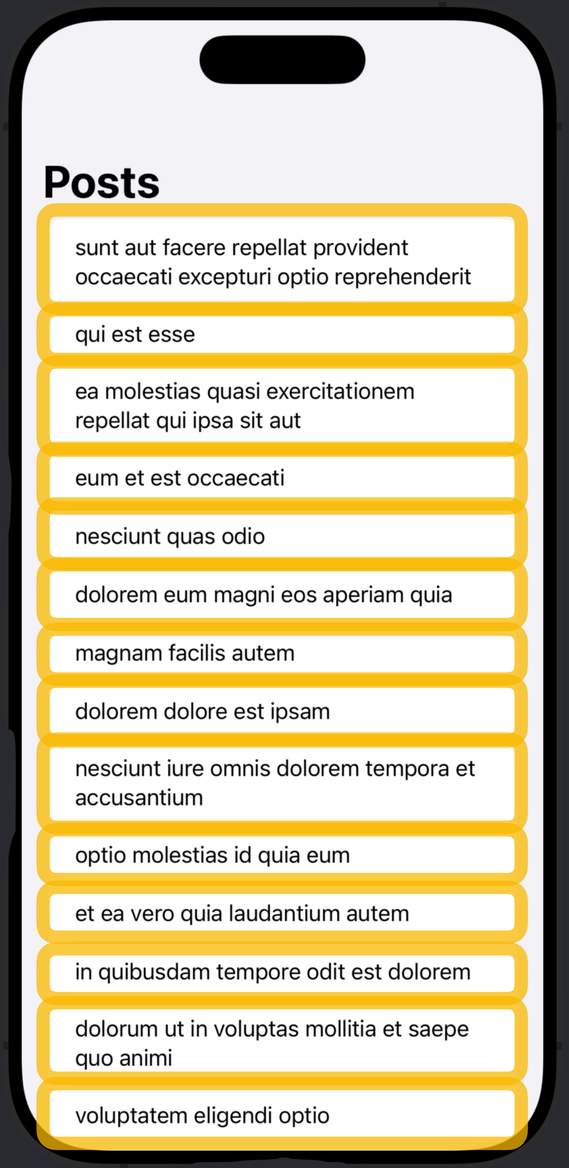
```swift
import SwiftUI

@MainActor
struct ContentView: View {
    @State var viewModel: PostsViewModel = .init()

    var body: some View {
        List {
            ForEach(viewModel.posts, id: \.id) { post in
                Text(post.title)
            }
        }
        .task {
            await viewModel.fetchPosts()
        }
        .navigationTitle("Posts")
    }
}

#Preview {
    NavigationStack {
        ContentView()
    }
}
```



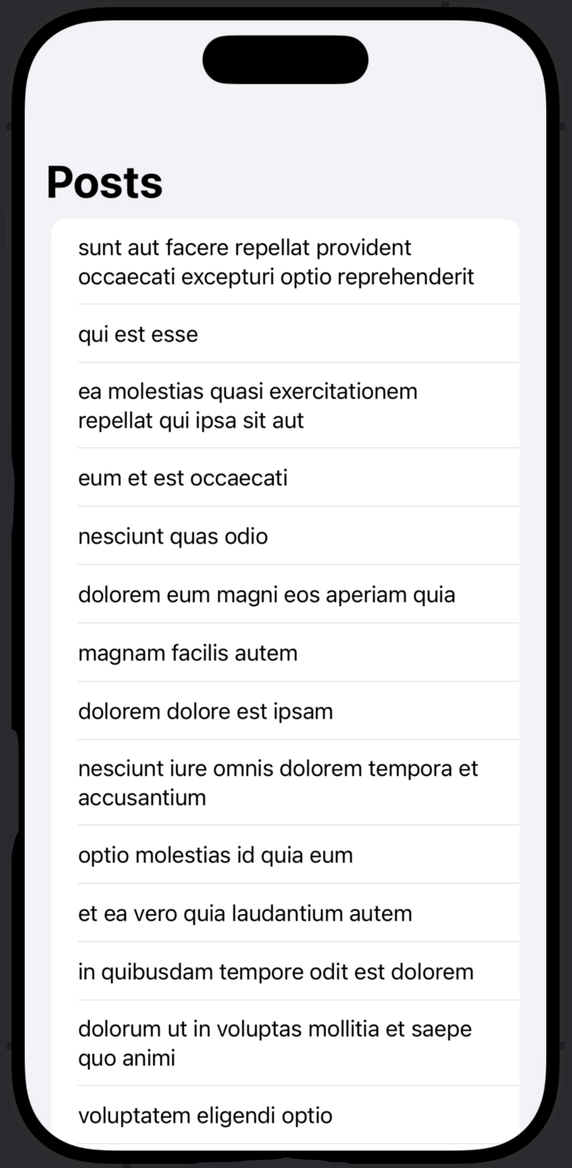
```swift
import SwiftUI

@MainActor
struct ContentView: View {
    @State var viewModel: PostsViewModel = .init()

    var body: some View {
        List {
            ForEach(viewModel.posts, id: \.id) { post in
                Text(post.title)
            }
        }
        .task {
            await viewModel.fetchPosts()
        }
        .navigationTitle("Posts")
    }
}

#Preview {
    NavigationStack {
        ContentView()
    }
}
```



```swift
import SwiftUI

@MainActor
struct ContentView: View {
    @State var viewModel: PostsViewModel = .init()

    var body: some View {
        List {
            ForEach(viewModel.posts, id: \.id) { post in
                Text(post.title)
            }
        }
        .task {
            await viewModel.fetchPosts()
        }
        .navigationTitle("Posts")
    }
}

#Preview {
    NavigationStack {
        ContentView()
    }
}
```



```swift
import SwiftUI

@MainActor
struct ContentView: View {
    @State var viewModel: PostsViewModel = .init()

    var body: some View {
        List {
            ForEach(viewModel.posts, id: \.id) { post in
                Text(post.title)
            }
        }
        .task {
            await viewModel.fetchPosts()
        }
        .navigationTitle("Posts")
    }
}

#Preview {
    NavigationStack {
        ContentView()
    }
}
```



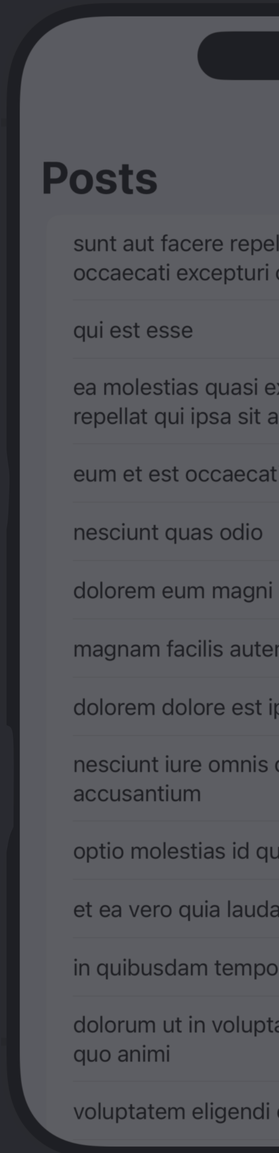
```swift
import SwiftUI

@MainActor
struct ContentView: View {
    @State var viewModel: PostsViewModel = .init()

    var body: some View {
        List {
            ForEach(viewModel.posts, id: \.id) { post in
                Text(post.title)
            }
        }
        .task {
            await viewModel.fetchPosts()
        }
        .navigationTitle("Posts")
    }
}

#Preview {
    NavigationStack {
        ContentView()
    }
}
```

```swift
import SwiftUI

@MainActor
struct ContentView: View {
    @State var viewModel: PostsViewModel = .init()

    var body: some View {
        List {
            ForEach(viewModel.posts, id: \.id) { post in
                Text(post.title)
            }
        }
        .task {
            await viewModel.fetchPosts()
        }
        .navigationTitle("Posts")
    }
}

#Preview {
    NavigationStack {
        ContentView()
    }
}
```

```swift
import SwiftUI

@MainActor
struct ContentView: View {
    @State var viewModel: PostsViewModel = .init()

    var body: some View {
        List {
            ForEach(viewModel.posts, id: \.id) { post in
                Text(post.title)
            }
        }
        .task {
            await viewModel.fetchPosts()
        }
        .navigationTitle("Posts")
    }
}

#Preview {
    NavigationStack {
        ContentView()
    }
}
```

Struct

- Value Semantic
- Uses Stack Memory
- Can not mutate in place
- Requires mutating keyword to functions, property setters



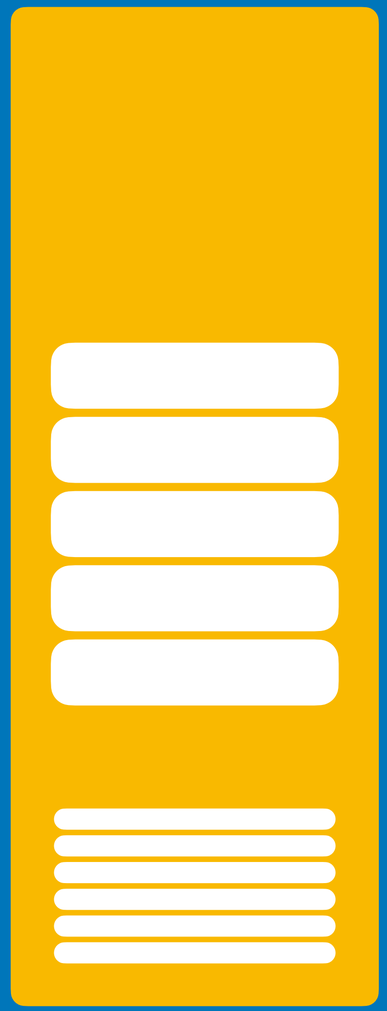
Struct

- Value Semantic
- Uses Stack Memory
- Can not mutate in place
- Requires mutating keyword to functions, property setters

```
func testValueSemantics() {
    var a: Int = 42
    var b = a
    b += 1
    print(b) // 43
    print(a) // 42
}
```

Struct

- Value Semantic
- Uses Stack Memory
- Can not mutate in place
- Requires mutating keyword to functions, property setters



Struct

- Value Semantic
- Uses Stack Memory
- Can not mutate in place
- Requires mutating keyword to functions, property setters

```
func testInPlaceMutation() {
    let a: Int = 42
    a += 1   ⊗ Left side of mutating operator isn't mutable: 'a' is a 'let' constant
}
```

Struct

- Value Semantic
- Uses Stack Memory
- Can not mutate in place
- Requires mutating keyword to functions, property setters

```
func testInPlaceMutation() {
    var a: Int = 42
    a += 1
}
```

⊗ Left side of mutating operator isn't mutable: 'a' is a 'let' constant

Struct

- Value Semantic
- Uses Stack Memory
- Can not mutate in place
- Requires mutating keyword to functions, property setters

```
struct Item {
    var name: String
    var price: Double
}

struct Bag {
    var items: [Item]

    func add(item: Item) {
        self.items.append(item)
    }
}
```

Cannot use mutating member on immutable value: 'self' is immutable
Mark method 'mutating' to make 'self' mutable

Struct

- Value Semantic
- Uses Stack Memory
- Can not mutate in place
- Requires mutating keyword to functions, property setters

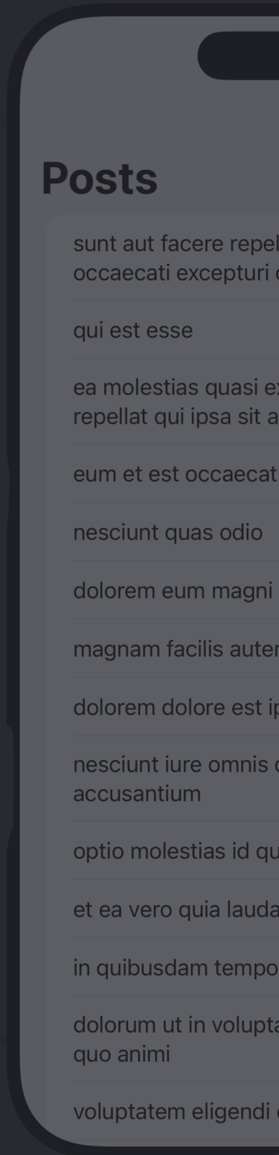
```
struct Item {
    var name: String
    var price: Double
}

struct Bag {
    var items: [Item]

    mutating func add(item: Item) {
        self.items.append(item)
    }
}
```

Cannot use mutating member on immutable value: 'self' is immutable
Mark method 'mutating' to make 'self' mutable

```swift
import SwiftUI

@MainActor
struct ContentView: View {
    @State var viewModel: PostsViewModel = .init()

    var body: some View {
        List {
            ForEach(viewModel.posts, id: \.id) { post in
                Text(post.title)
            }
        }
        .task {
            await viewModel.fetchPosts()
        }
        .navigationTitle("Posts")
    }
}

#Preview {
    NavigationStack {
        ContentView()
    }
}
```

Struct

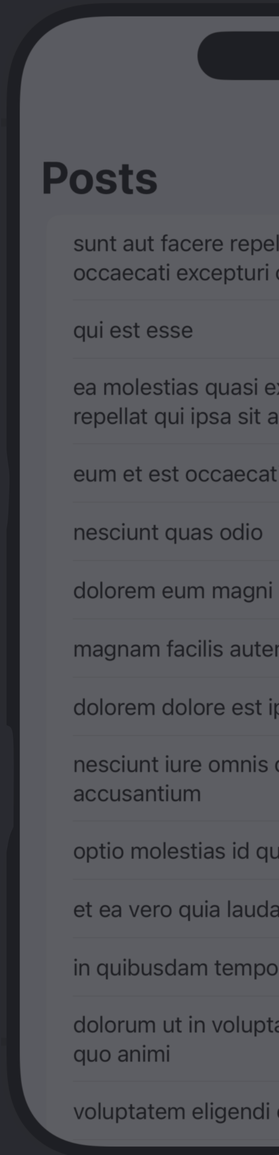
- Value Semantic
- Uses Stack Memory
- Can not mutate in place
- Requires mutating keyword to functions, property setters



```swift
import SwiftUI

@MainActor
struct ContentView: View {
    @State var viewModel: PostsViewModel = .init()

    var body: some View {
        List {
            ForEach(viewModel.posts, id: \.id) { post in
                Text(post.title)
            }
        }
        .task {
            await viewModel.fetchPosts()
        }
        .navigationTitle("Posts")
    }
}

#Preview {
    NavigationStack {
        ContentView()
    }
}
```

Protocol

```swift
public protocol View {
  associatedtype Body : View

  @ViewBuilder
  @MainActor
  var body: Self.Body { get }
}
```



Protocol

```
public protocol View {
  associatedtype Body : View

  @ViewBuilder
  @MainActor
  var body: Self.Body { get }
}
```

Makes Protocol Generic

Protocol

```swift
public protocol View {
    associatedtype Body : View

    @ViewBuilder
    @MainActor
    var body: Self.Body { get }
}
```

Recursion in protocol conformance

```swift
List {
    ForEach(viewModel.posts, id: \.id) { post in
        Text(post.title)
    }
}
```

```swift
extension Text : View {

    /// The type of view representing the body of this view.
    ///
    /// When you create a custom view, Swift infers this type from your
    /// implementation of the required ``View/body-swift.property``
property.
    public typealias Body = Never
}

extension Color, Image, ModifiedContent, Gesture : View {
    public typealias Body = Never
}

@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
extension Never : View {
}
```

Protocol

```
public protocol View {
   associatedtype Body : View

   @ViewBuilder
   @MainActor
   var body: Self.Body { get }
}
```

- Get Only
- Should not set/change/mutate
- Variable of the struct

```
struct CounterView: View {
    var count: Int = 0

    var body: some View {
        Text("Count: \(count)")
        Button("Increment") {
            count += 1
        }
    }
}
```

⊗ Left side of mutating operator isn't mutable: 'self' is immutable

Protocol

```
public protocol View {
   associatedtype Body : View

   @ViewBuilder
   @MainActor
   var body: Self.Body { get }
}
```

- Get Only
- Should not set/change/mutate
- Variable of the struct

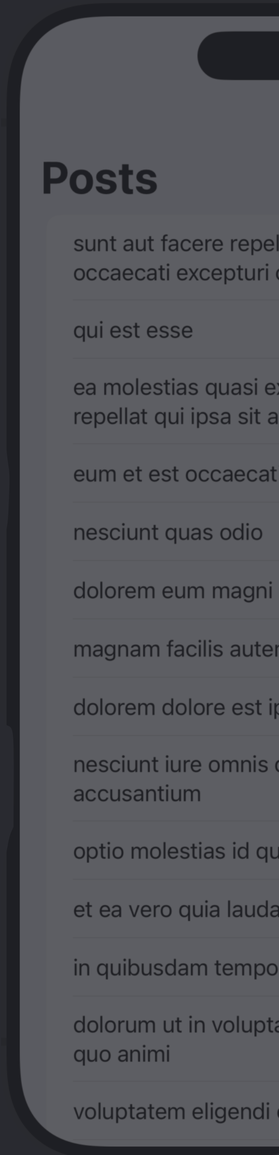
```
struct CounterView: View {
    @State var count: Int = 0

    var body: some View {
        Text("Count: \(count)")
        Button("Increment") {
            count += 1
        }
    }
}
```

⊗ Left side of mutating operator isn't mutable: 'self' is immutable

```swift
import SwiftUI

@MainActor
struct ContentView: View {
    @State var viewModel: PostsViewModel = .init()

    var body: some View {
        List {
            ForEach(viewModel.posts, id: \.id) { post in
                Text(post.title)
            }
        }
        .task {
            await viewModel.fetchPosts()
        }
        .navigationTitle("Posts")
    }
}

#Preview {
    NavigationStack {
        ContentView()
    }
}
```

Protocol

```swift
public protocol View {
    associatedtype Body : View

    @ViewBuilder
    @MainActor
    var body: Self.Body { get }
}
```

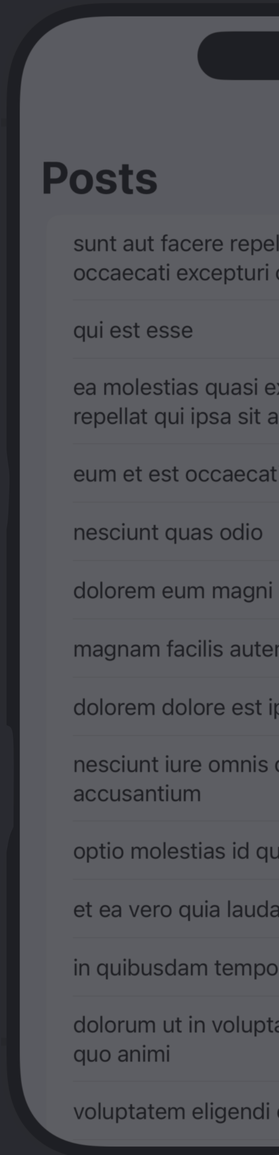


```swift
import SwiftUI

@MainActor
struct ContentView: View {
    @State var viewModel: PostsViewModel = .init()

    var body: some View {
        List {
            ForEach(viewModel.posts, id: \.id) { post in
                Text(post.title)
            }
        }
        .task {
            await viewModel.fetchPosts()
        }
        .navigationTitle("Posts")
    }
}

#Preview {
    NavigationStack {
        ContentView()
    }
}
```

Actor Isolation

Property Wrapper

Result Builder



```swift
import SwiftUI

@MainActor
struct ContentView: View {
    @State var viewModel: PostsViewModel = .init()

    var body: some View {
        List {
            ForEach(viewModel.posts, id: \.id) { post in
                Text(post.title)
            }
        }
        .task {
            await viewModel.fetchPosts()
        }
        .navigationTitle("Posts")
    }
}

#Preview {
    NavigationStack {
        ContentView()
    }
}
```

Actor Isolation

Property Wrapper

State | Binding | FocusState

ObservedObject | StateObject

Environment | AppStorage

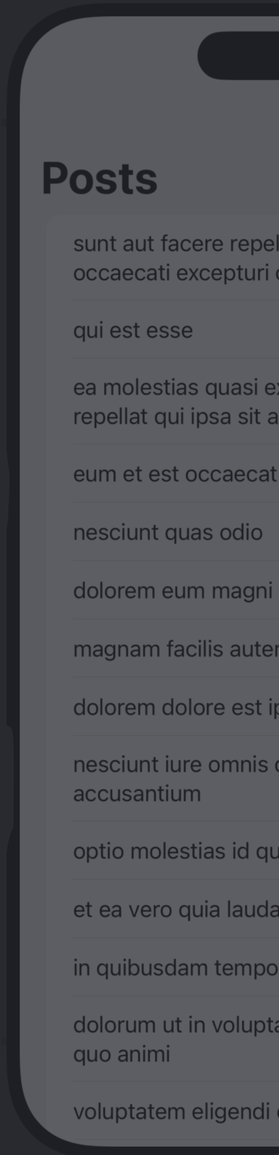
EnvironmentObject | Bindable

Result Builder

ViewBuilder

TableBuilder

SceneBuilder



Property Wrapper

Property Delegates (SE-0258)

@State, @ObservedObject, @Environment, @FocusState etc.

```
@propertyWrapper
struct InMemoryPropertyWrapper {

}
```

Property wrapper type 'InMemoryPropertyWrapper' does not contain a non-static property named 'wrappedValue'

Property Wrapper

Property Delegates (SE-0258)

@State, @ObservedObject, @Environment, @FocusState etc.

```
@propertyWrapper
struct InMemoryPropertyWrapper<T> {
    var wrappedValue: T
}
```

Property wrapper type 'InMemoryPropertyWrapper' does not contain a non-static property named 'wrappedValue'

Property Wrapper

Property Delegates (SE-0258)

@State, @ObservedObject, @Environment, @FocusState etc.

```
@propertyWrapper
struct InMemoryPropertyWrapper<Value> {
    var wrappedValue: Value
}
```

Property Wrapper

Property Delegates (SE-0258)

@State, @ObservedObject, @Environment, @FocusState etc.

```
@propertyWrapper
struct InMemoryPropertyWrapper<Value> {
    var wrappedValue: Value
}
```

But Why?

Why Property Wrapper

```
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []

    var pageLimit: Int = 10
    var pageOffset: Int = 0

    init() {}

    func fetchPosts() async {
        ...
    }
}
```

Why Property Wrapper

```
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []

    var pageLimit: Int = 10
    var pageOffset: Int = 0

    init() {}

    func fetchPosts() async {
        ...
    }
}
```

Why Property Wrapper

```
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []

    var pageLimit: Int {
        get {
            (UserDefaults.standard.value(forKey: "pageLimit") as? Int) ?? 10
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: "pageLimit")
        }
    }
    var pageOffset: Int {
        get {
            (UserDefaults.standard.value(forKey: "pageOffset") as? Int) ?? 0
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: "pageOffset")
        }
    }

    func fetchPosts() async {
      ...
    }
}
```

Why Property Wrapper

```
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []

    var pageLimit: Int {
        get {
            (UserDefaults.standard.value(forKey: "pageLimit") as? Int) ?? 10
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: "pageLimit")
        }
    }
    var pageOffset: Int {
        get {
            (UserDefaults.standard.value(forKey: "pageOffset") as? Int) ?? 0
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: "pageOffset")
        }
    }

    func fetchPosts() async {
      ...
    }
}
```

Why Property Wrapper

```
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []

    var pageLimit: Int {
        get {
            return (UserDefaults.standard.value(forKey: "pageLimit") as? Int) ?? 10
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: "pageLimit")
        }
    }
    var pageOffset: Int {
        get {
            return (UserDefaults.standard.value(forKey: "pageOffset") as? Int) ?? 0
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: "pageOffset")
        }
    }

    func fetchPosts() async {
      ...
    }
}
```

Why Property Wrapper

```
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []

    var pageLimit: Int {
        get {
            return (UserDefaults.standard.value(forKey: "pageLimit") as? Int) ?? 10
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: "pageLimit")
        }
    }
    var pageOffset: Int {
        get {
            return (UserDefaults.standard.value(forKey: "pageOffset") as? Int) ?? 0
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: "pageOffset")
        }
    }

    func fetchPosts() async {
      ...
    }
}
```

Why Property Wrapper

```
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []

    var pageLimit: Int {
        get {
            (UserDefaults.standard.value(forKey: "pageLimit") as? Int) ?? 10
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: "pageLimit")
        }
    }
    var pageOffset: Int {
        get {
            (UserDefaults.standard.value(forKey: "pageOffset") as? Int) ?? 0
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: "pageOffset")
        }
    }

    func fetchPosts() async {
      ...
    }
}
```

Why Property Wrapper

```
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []

    var pageLimit: Int {
        get {
            (UserDefaults.standard.value(forKey: "pageLimit") as? Int) ?? 10
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: "pageLimit")
        }
    }
    var pageOffset: Int {
        get {
            (UserDefaults.standard.value(forKey: "pageOffset") as? Int) ?? 0
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: "pageOffset")
        }
    }

    func fetchPosts() async {
      ...
    }
}
```

How to Property Wrapper

```
struct UserDefaultStore {

    var propertyValue: Int {
        get {
            (UserDefaults.standard.value(forKey: “pageLimit") as? Int) ?? 10
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: “pageLimit")
        }
    }
}
```

How to Property Wrapper

```
@propertyWrapper struct UserDefaultStore {

    var propertyValue: Int {
        get {
            (UserDefaults.standard.value(forKey: “pageLimit") as? Int) ?? 10
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: “pageLimit")
        }
    }
}
```

How to Property Wrapper

```
@propertyWrapper struct UserDefaultStore {

    var wrappedValue: Int {
        get {
            (UserDefaults.standard.value(forKey: “pageLimit") as? Int) ?? 10
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: “pageLimit")
        }
    }
}
```

How to Property Wrapper

```
@propertyWrapper struct UserDefaultStore {

    var wrappedValue: Int {
        get {
            (UserDefaults.standard.value(forKey: “pageLimit") as? Int) ?? 10
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: “pageLimit")
        }
    }
}
```

How to Property Wrapper

```
@propertyWrapper struct UserDefaultStore {
    let key: String

    var wrappedValue: Int {
        get {
            (UserDefaults.standard.value(forKey: key) as? Int) ?? 10
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: key)
        }
    }
}
```

How to Property Wrapper

```
@propertyWrapper struct UserDefaultStore {
    let defaultValue: Int
    let key: String

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

How to Property Wrapper

```
@propertyWrapper struct UserDefaultStore<T> {
    let defaultValue: T
    let key: String

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

How to Property Wrapper

```
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []

    var pageLimit: Int = 10
    var pageOffset: Int = 0

    init() {}

    func fetchPosts() async {
        ...
    }
}
```

How to Property Wrapper

```
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []

    @ObservationIgnored
    @UserDefaultStore(key: "pageLimit", defaultValue: 10) var pageLimit: Int

    @ObservationIgnored
    @UserDefaultStore(key: "pageOffset", defaultValue: 0) var pageOffset: Int

    func fetchPosts() async {
       ...
    }
}
```

How to Property Wrapper

```
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []

    @ObservationIgnored
    @UserDefaultStore(key: "pageLimit") var pageLimit: Int = 10

    @ObservationIgnored
    @UserDefaultStore(key: "pageOffset", defaultValue: 0) var p

    func fetchPosts() async {
       ...
    }
}
```

Extra argument 'wrappedValue' in call

Missing argument for parameter 'defaultValue' in property wrapper initializer; add 'wrappedValue' and 'defaultValue' arguments in '@UserDefaultStore(...)'

```
...
@AppStorage("item") var item: String = ""
```

How to Property Wrapper

```
@propertyWrapper struct UserDefaultStore<T> {
    let defaultValue: T
    let key: String

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

How to Property Wrapper

```
@propertyWrapper struct UserDefaultStore<T> {
    let key: String
    let defaultValue: T

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

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

How to Property Wrapper

```
@propertyWrapper struct UserDefaultStore<T> {
    let key: String
    let defaultValue: T

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

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

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

How to Property Wrapper

```
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []

    @ObservationIgnored
    @UserDefaultStore(key: "pageLimit") var pageLimit: Int = 10

    @ObservationIgnored
    @UserDefaultStore(key: "pageOffset", defaultValue: 0) var p

    func fetchPosts() async {
        ...
    }
}
```

Extra argument 'wrappedValue' in call

Missing argument for parameter 'defaultValue' in property wrapper initializer; add 'wrappedValue' and 'defaultValue' arguments in '@UserDefaultStore(...)'

How to Property Wrapper

```
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []

    @ObservationIgnored
    @UserDefaultStore(key: "pageLimit") var pageLimit: Int = 10

    @ObservationIgnored
    @UserDefaultStore(key: "pageOffset") var pageOffset: Int = 0

    func fetchPosts() async {
       ...
    }
}
```

How to Property Wrapper

```
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []


    @UserDefaultStore(key: "pageLimit") var pageLimit: Int = 10

    @ObservationIgnored
    @UserDefaultStore(key: "pageOffset") var pageOffset: Int = 0

    func fetchPosts() async {
       ...
    }
}
```

Property wrapper cannot be applied to a computed property

How to Property Wrapper

```
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []


    @UserDefaultStore(key: "pageLimit") var pageLimit: Int = 10

    @ObservationIgnored
    @UserDefaultStore(key: "pageOffset") var pageOffset: Int = 0

    func fetchPosts() async {
        ...
    }
}
```

Macro does not play well with property wrappers yet

How to Property Wrapper

```
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []

    @ObservationIgnored
    @UserDefaultStore(key: "pageLimit") var pageLimit: Int = 10

    @ObservationIgnored
    @UserDefaultStore(key: "pageOffset") var pageOffset: Int = 0

    func fetchPosts() async {
       ...
    }
}
```

How to Property Wrapper

```
@propertyWrapper struct UserDefaultStore<T> {
    let key: String
    let defaultValue: T

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

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

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

How to Property Wrapper

```
@propertyWrapper @Observable class UserDefaultStore<T> {
    let key: String
    let defaultValue: T

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

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

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

How to Property Wrapper

```
@propertyWrapper struct UserDefaultStore<T> {
    let key: String
    let defaultValue: T

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

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

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

How to Property Wrapper

```
@propertyWrapper struct UserDefaultStore<T> {
    let key: String
    let defaultValue: T

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

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

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

⚠️ Note

User Defaults only supports plist compatible types.

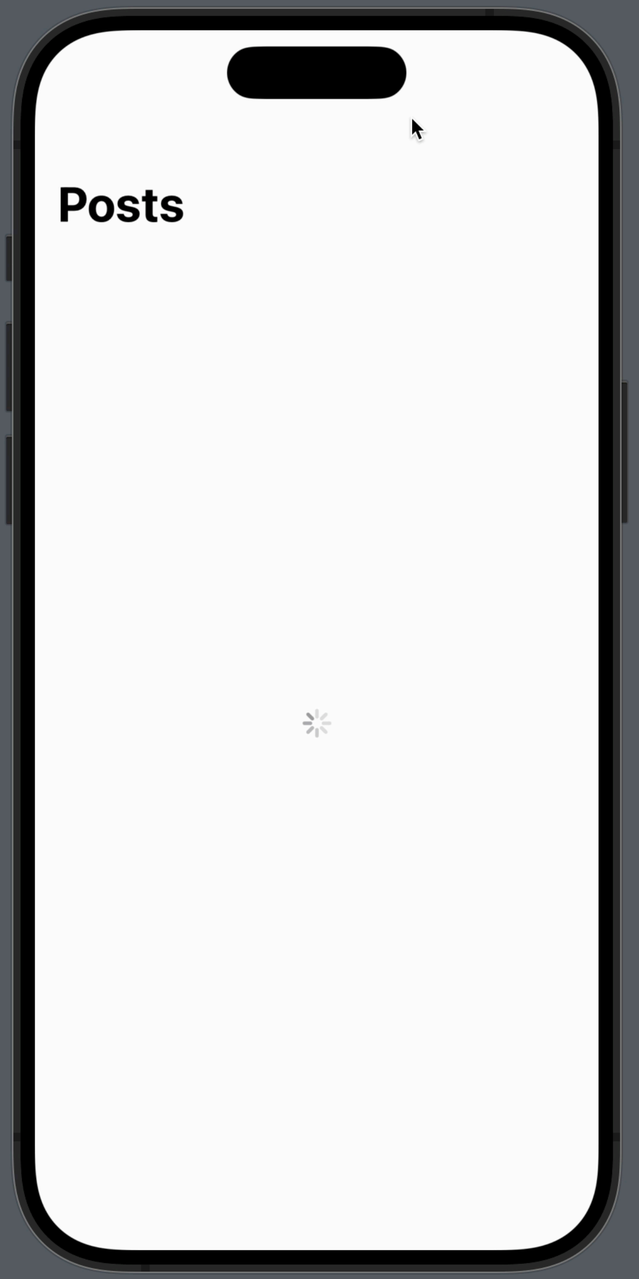
SwiftUI Lifecycle

```
import SwiftUI

@MainActor
struct ContentView: View {
    @StateObject var viewModel = PostViewModel()

    var body: some View {
        List {
            ForEach(viewModel.posts) { post in
                Text(post.title)
            }
        }
        .task {
            await viewModel.fetchPosts()
        }
        .navigationTitle("Posts")
    }
}
```

Time for which an **object** **stays** **in** **memory** during the **runtime** of the program

“**Life-time**” of an object

```swift
@MainActor
struct ContentView: View {
    @State var viewModel: PostViewModel = .init()
    var count: Int = 0

    var body: some View {
        Group {
            if viewModel.isFetching {
                ProgressView()
            } else {
                List {
                    ForEach(viewModel.posts, id: \.id) { post in
                        NavigationLink(destination: {
                            PostDetailView(post: post)
                        }) {
                            Text(post.title)
                        }
                    }
                }
            }
        }
        .task { await viewModel.fetchPosts() }
        .navigationTitle("Posts")
        .onAppear { print("On Appear Called") }
        .onDisappear { print("On Disappear Called") }
    }
}
```



```swift
@MainActor
struct ContentView: View {
    @State var viewModel: PostViewModel = .init()
    var count: Int = 0

    var body: some View {
        Group {
            if viewModel.isFetching {
                ProgressView()
            } else {
                List {
                    ForEach(viewModel.posts, id: \.id) { post in
                        NavigationLink(destination: {
                            PostDetailView(post: post)
                        }) {
                            Text(post.title)
                        }
                    }
                }
            }
        }
        .task { await viewModel.fetchPosts() }
        .navigationTitle("Posts")
        .onAppear { print("On Appear Called") }
        .onDisappear { print("On Disappear Called") }
    }
}
```

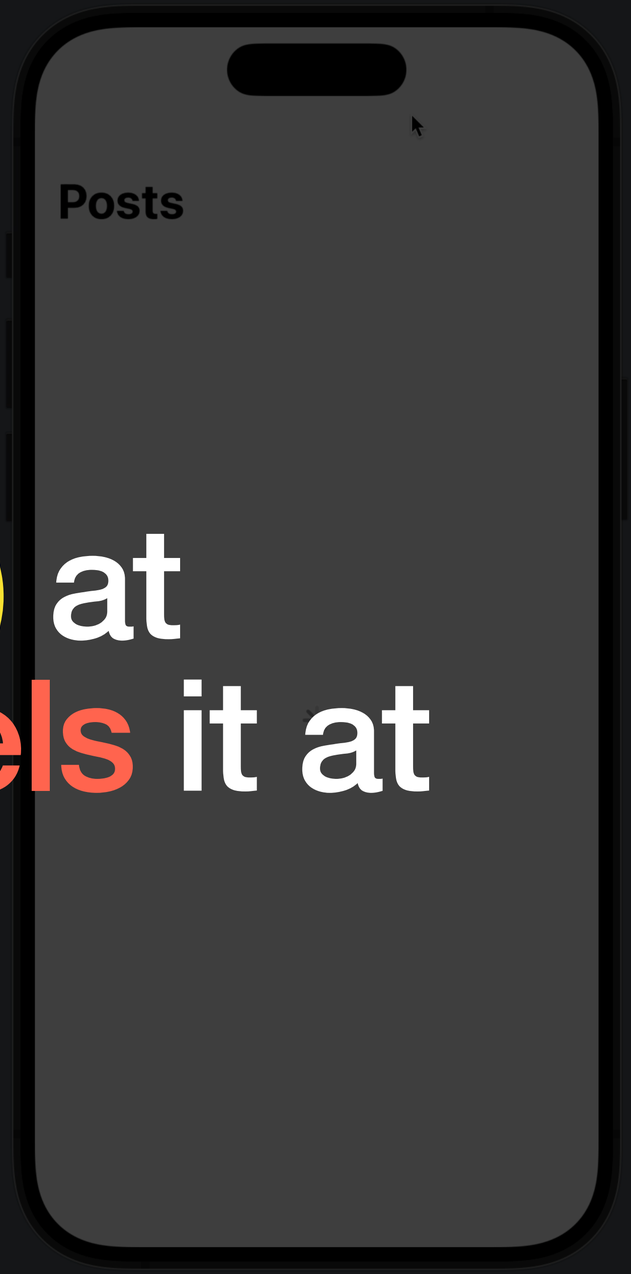
.onAppear, .onDisappear are ViewModifiers, that are the hooks for SwiftUI View's Life-cycle

Ratnesh Jain

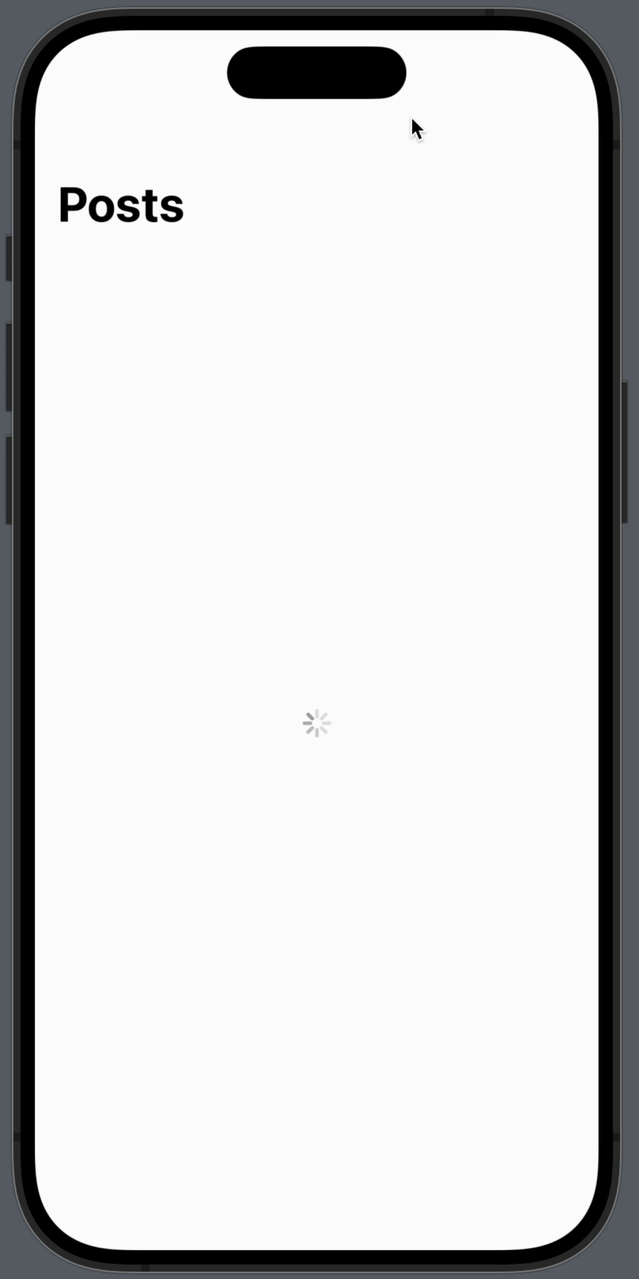
```swift
@MainActor
struct ContentView: View {
    @State var viewModel: PostViewModel = .init()
    var count: Int = 0

    var body: some View {
        Group {
            if viewModel.isFetching {
                ProgressView()
            } else {
                List {
                    ForEach(viewModel.posts, id: \.id) { post in
                        NavigationLink(destination:
                            PostDetailView(post: post)
                        ) {
                            Text(post.title)
                        }
                    }
                }
            }
        }
        .task { await viewModel.fetchPosts() }
        .navigationTitle("Posts")
        .onAppear { print("On Appear Called") }
        .onDisappear { print("On Disappear Called") }
    }
}
```

.task

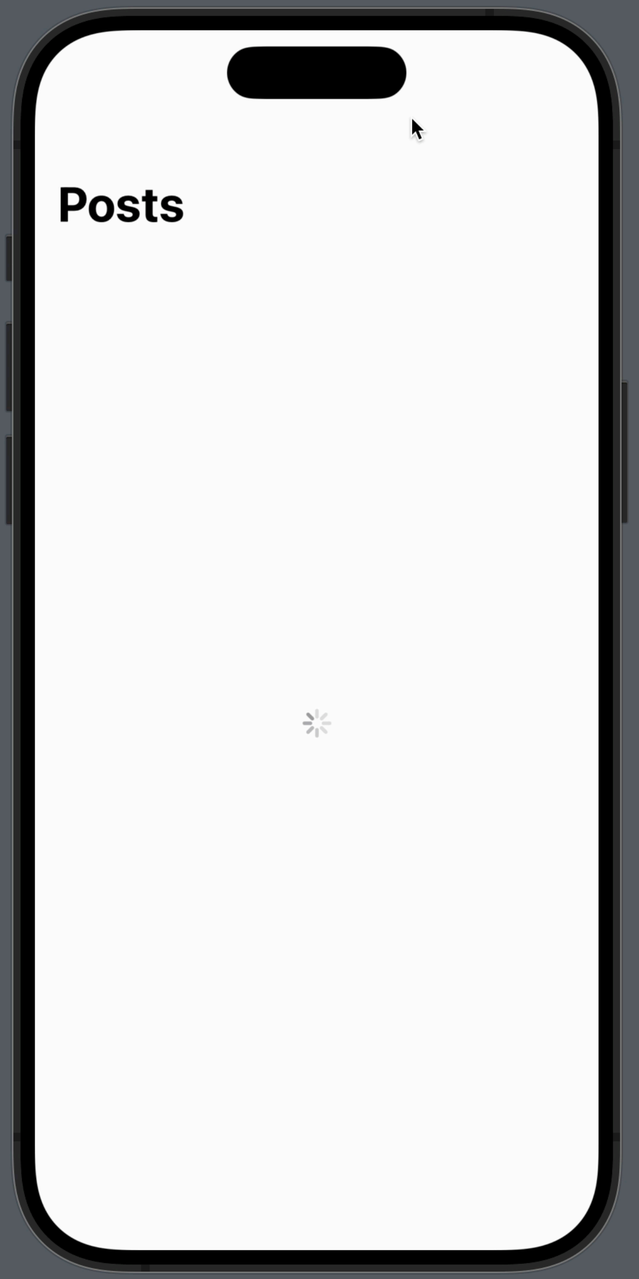
creats Task (async) at “onAppear” and cancels it at “onDisappear”



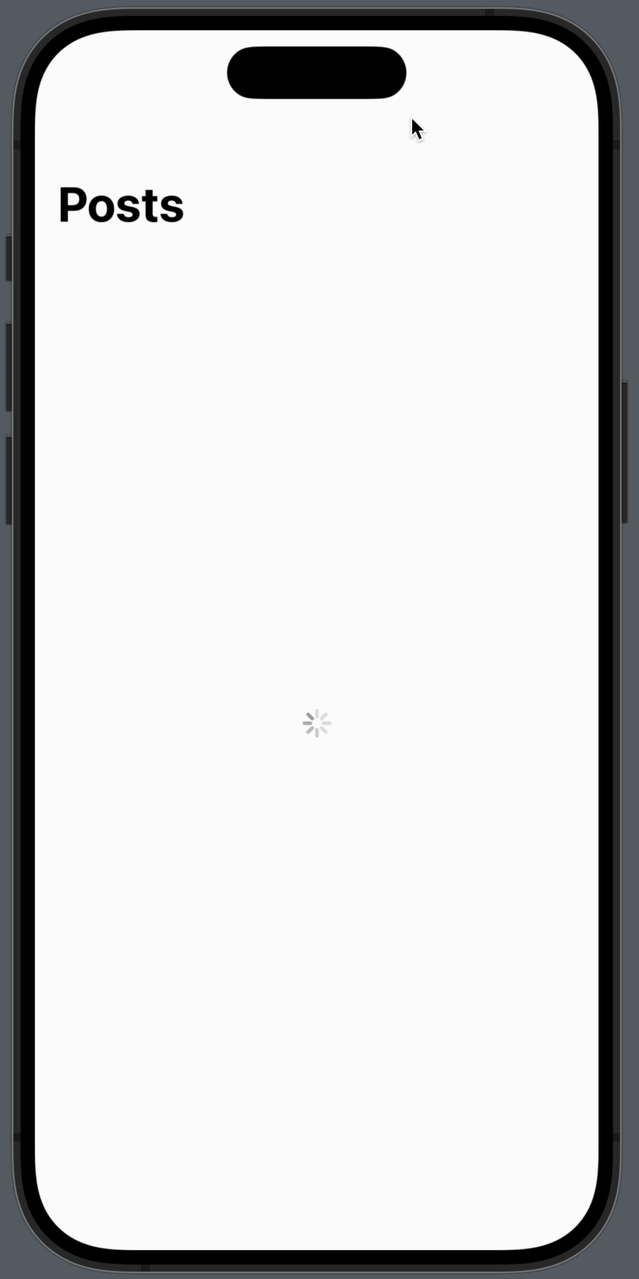
```swift
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []
    var isFetching: Bool = false

    @ObservationIgnored
    @UserDefaultStore(key: "pageLimit") var pageLimit: Int = 10

    @ObservationIgnored
    @UserDefaultStore(key: "pageOffset") var pageOffset: Int = 0

    func fetchPosts() async {
        do {
            self.isFetching = true
            try await Task.sleep(for: .seconds(2))
            let (data, _) = try await URLSession.shared.data(
                from: URL(string: "https://jsonplaceholder.typicode.com/posts")!
            )
            let decoded = try JSONDecoder().decode([Post].self, from: data)
            self.posts = decoded
            self.isFetching = false
        } catch {
            print(error)
            self.isFetching = false
        }
    }
}
```



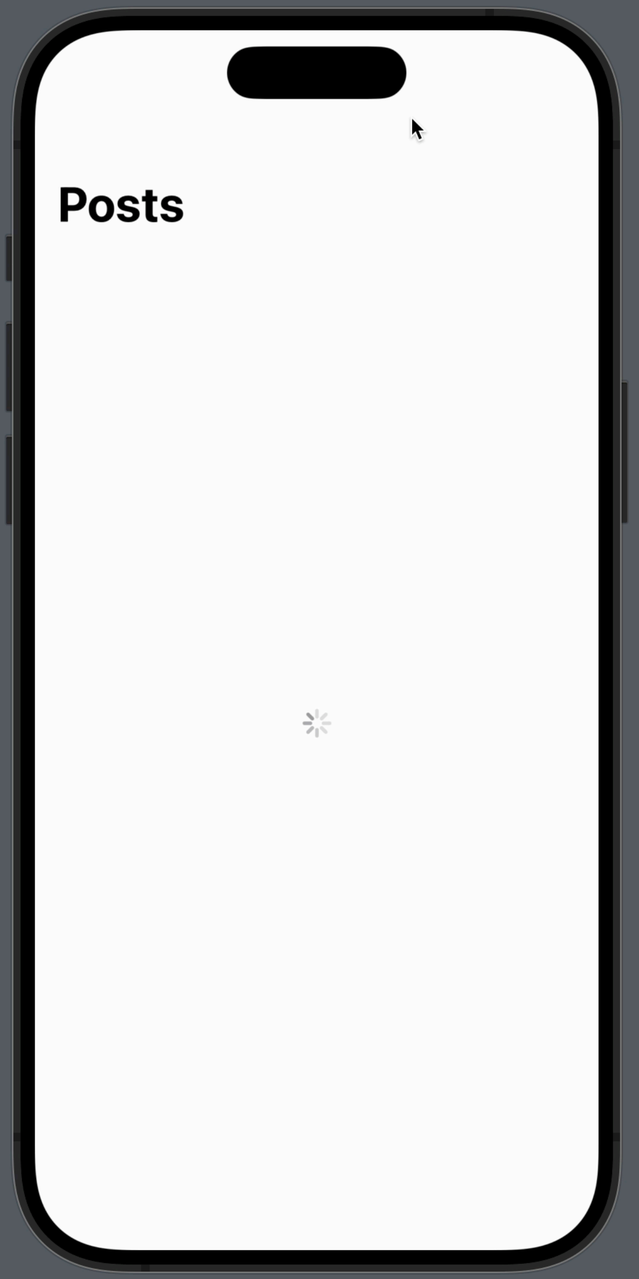
```swift
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []
    var isFetching: Bool = false

    @ObservationIgnored
    @UserDefaultStore(key: "pageLimit") var pageLimit: Int = 10

    @ObservationIgnored
    @UserDefaultStore(key: "pageOffset") var pageOffset: Int = 0

    func fetchPosts() async {
        do {
            self.isFetching = true
            try await Task.sleep(for: .seconds(2))
            let (data, _) = try await URLSession.shared.data(
                from: URL(string: "https://jsonplaceholder.typicode.com/posts")!
            )
            let decoded = try JSONDecoder().decode([Post].self, from: data)
            self.posts = decoded
            self.isFetching = false
        } catch {
            print(error)
            self.isFetching = false
        }
    }
}
```



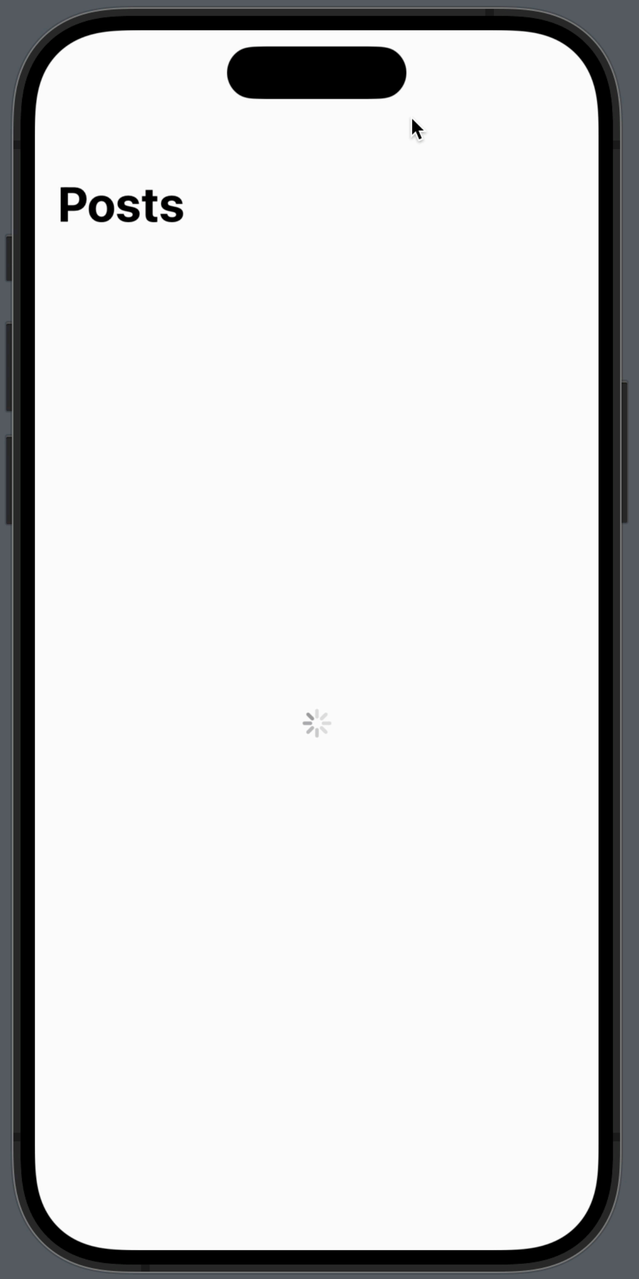
```swift
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []
    var isFetching: Bool = false

    @ObservationIgnored
    @UserDefaultStore(key: "pageLimit") var pageLimit: Int = 10

    @ObservationIgnored
    @UserDefaultStore(key: "pageOffset") var pageOffset: Int = 0

    func fetchPosts() async {
        do {
            self.isFetching = true
            try await Task.sleep(for: .seconds(2))
            let (data, _) = try await URLSession.shared.data(
                from: URL(string: "https://jsonplaceholder.typicode.com/posts")!
            )
            let decoded = try JSONDecoder().decode([Post].self, from: data)
            self.posts = decoded
            self.isFetching = false
        } catch {
            print(error)
            self.isFetching = false
        }
    }
}
```



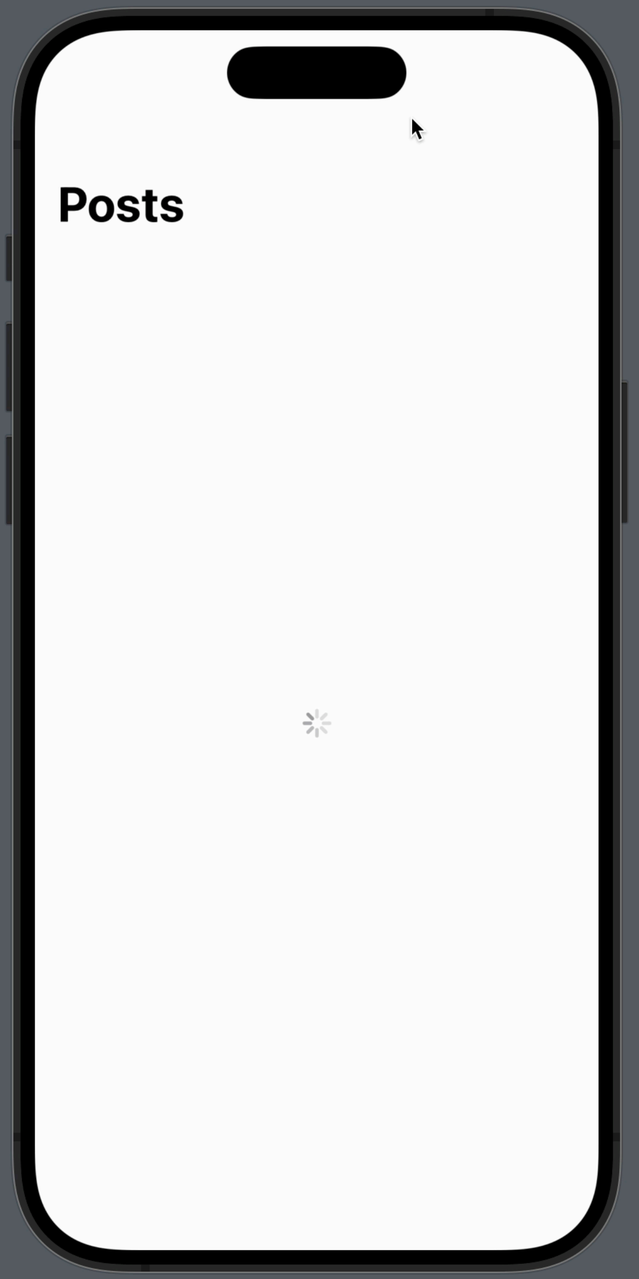
```swift
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []
    var isFetching: Bool = false

    @ObservationIgnored
    @UserDefaultStore(key: "pageLimit") var pageLimit: Int = 10

    @ObservationIgnored
    @UserDefaultStore(key: "pageOffset") var pageOffset: Int = 0

    func fetchPosts() async {
        do {
            self.isFetching = true
            try await Task.sleep(for: .seconds(2))
            let (data, _) = try await URLSession.shared.data(
                from: URL(string: "https://jsonplaceholder.typicode.com/posts")!
            )
            let decoded = try JSONDecoder().decode([Post].self, from: data)
            self.posts = decoded
            self.isFetching = false
        } catch {
            print(error)
            self.isFetching = false
        }
    }
}
```



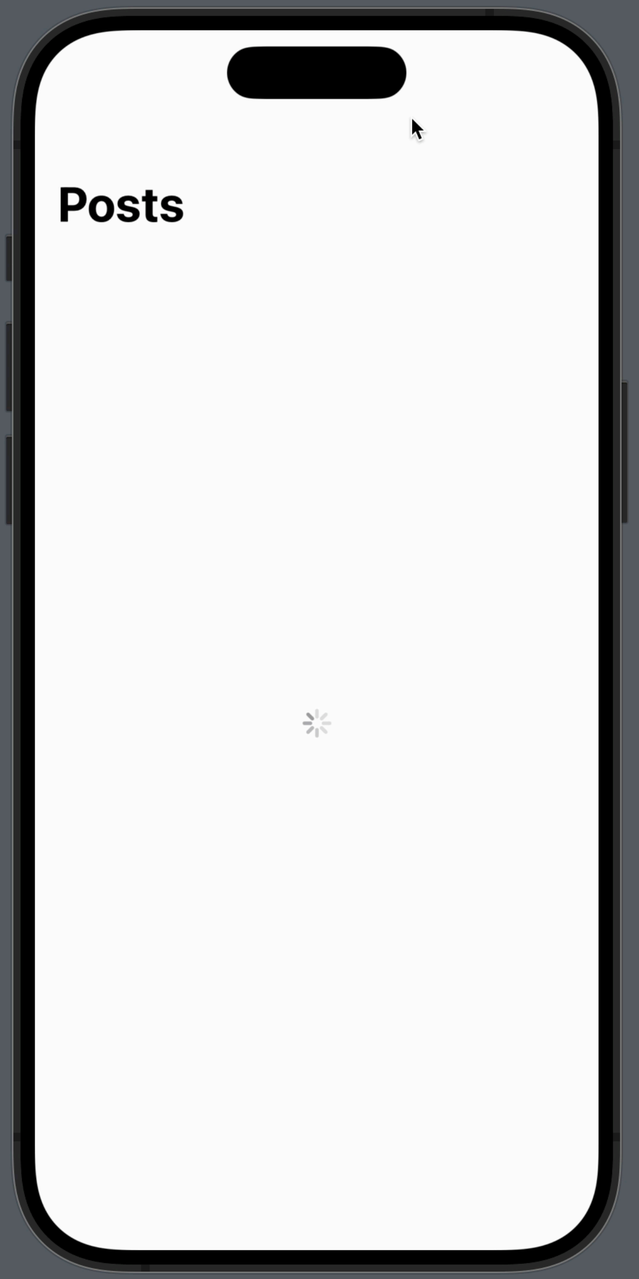
```swift
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []
    var isFetching: Bool = false

    @ObservationIgnored
    @UserDefaultStore(key: "pageLimit") var pageLimit: Int = 10

    @ObservationIgnored
    @UserDefaultStore(key: "pageOffset") var pageOffset: Int = 0

    func fetchPosts() async {
        do {
            self.isFetching = true
            try await Task.sleep(for: .seconds(2))
            let (data, _) = try await URLSession.shared.data(
                from: URL(string: "https://jsonplaceholder.typicode.com/posts")!
            )
            let decoded = try JSONDecoder().decode([Post].self, from: data)
            self.posts = decoded
            self.isFetching = false
        } catch {
            print(error)
            self.isFetching = false
        }
    }
}
```



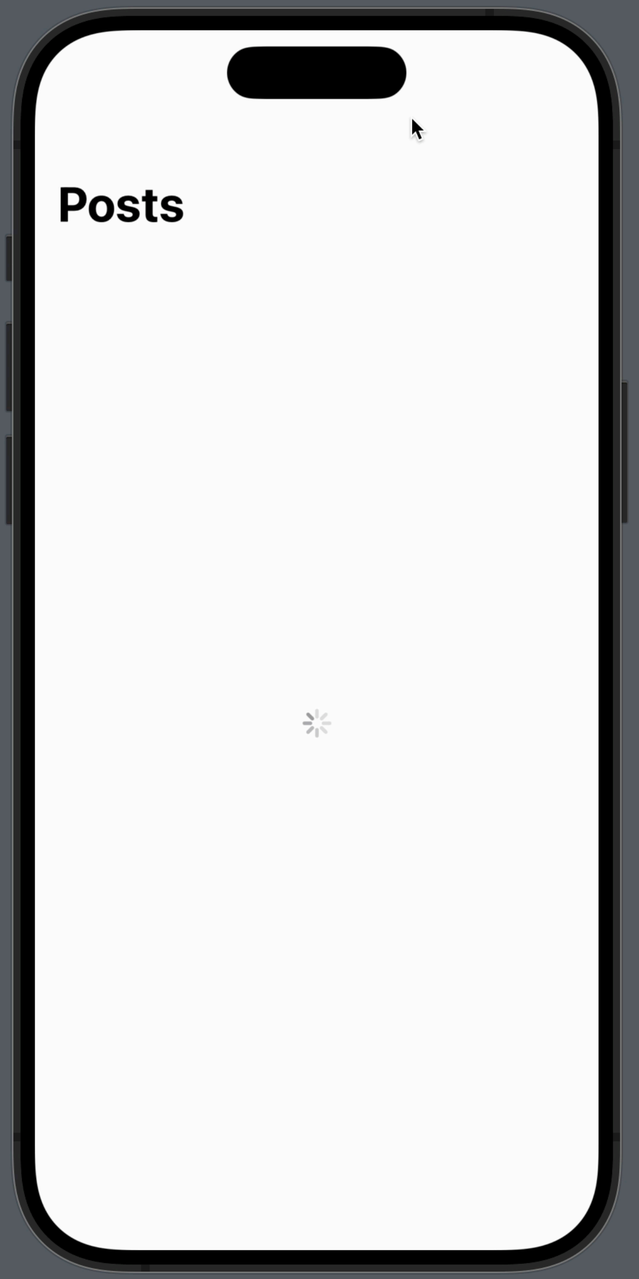
```swift
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []
    var isFetching: Bool = false

    @ObservationIgnored
    @UserDefaultStore(key: "pageLimit") var pageLimit: Int = 10

    @ObservationIgnored
    @UserDefaultStore(key: "pageOffset") var pageOffset: Int = 0

    func fetchPosts() async {
        do {
            self.isFetching = true
            try await Task.sleep(for: .seconds(2))
            let (data, _) = try await URLSession.shared.data(
                from: URL(string: "https://jsonplaceholder.typicode.com/posts")!
            )
            let decoded = try JSONDecoder().decode([Post].self, from: data)
            self.posts = decoded
            self.isFetching = false
        } catch {
            print(error)
            self.isFetching = false
        }
    }
}
```



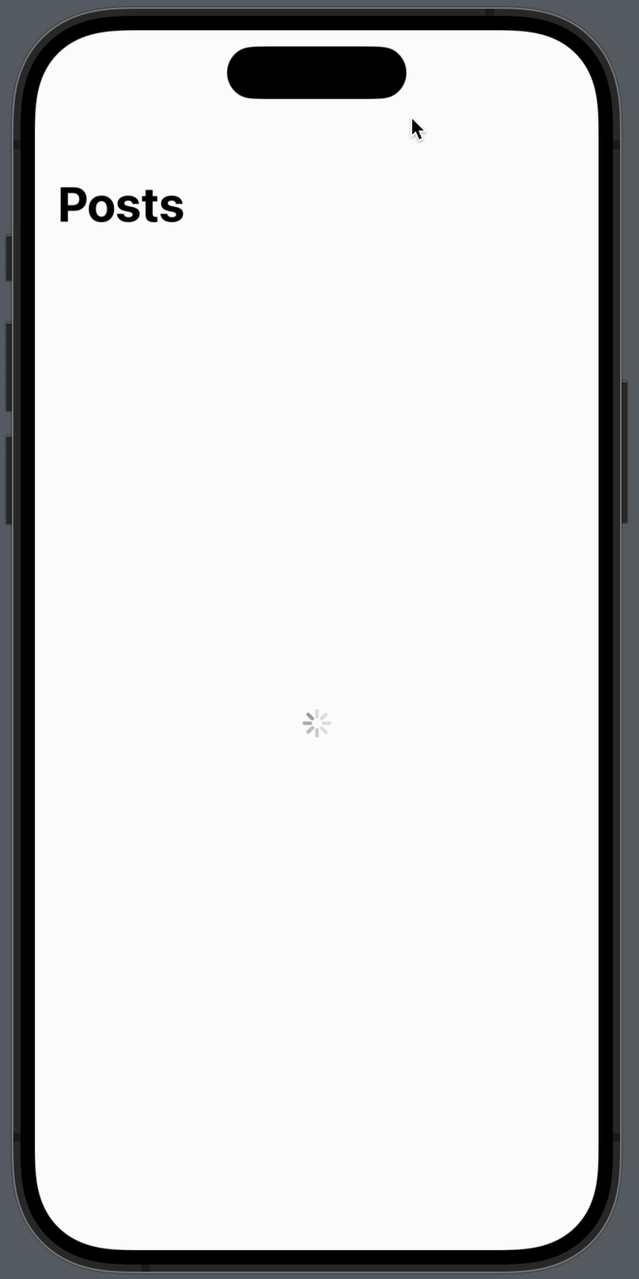
```swift
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []
    var isFetching: Bool = false

    @ObservationIgnored
    @UserDefaultStore(key: "pageLimit") var pageLimit: Int = 10

    @ObservationIgnored
    @UserDefaultStore(key: "pageOffset") var pageOffset: Int = 0

    func fetchPosts() async {
        do {
            self.isFetching = true
            try await Task.sleep(for: .seconds(2))
            let (data, _) = try await URLSession.shared.data(
                from: URL(string: "https://jsonplaceholder.typicode.com/posts")!
            )
            let decoded = try JSONDecoder().decode([Post].self, from: data)
            self.posts = decoded
            self.isFetching = false
        } catch {
            print(error)
            self.isFetching = false
        }
    }
}
```



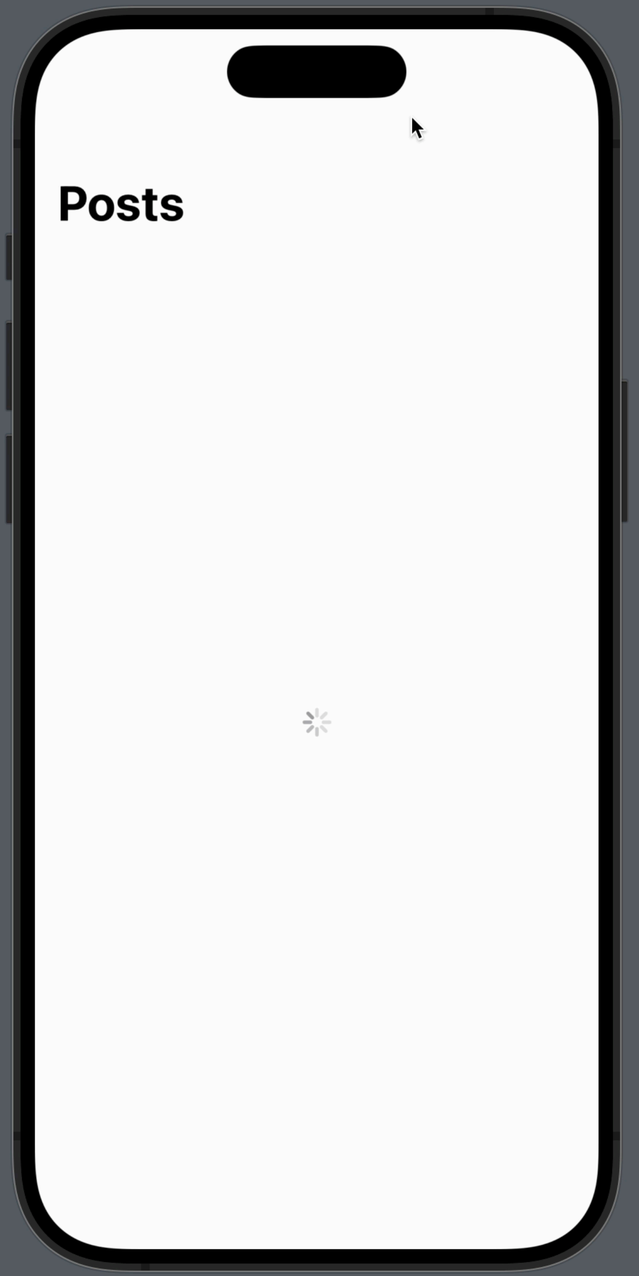
```swift
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []
    var isFetching: Bool = false

    @ObservationIgnored
    @UserDefaultStore(key: "pageLimit") var pageLimit: Int = 10

    @ObservationIgnored
    @UserDefaultStore(key: "pageOffset") var pageOffset: Int = 0

    func fetchPosts() async {
        do {
            self.isFetching = true
            try await Task.sleep(for: .seconds(2))
            let (data, _) = try await URLSession.shared.data(
                from: URL(string: "https://jsonplaceholder.typicode.com/posts")!
            )
            let decoded = try JSONDecoder().decode([Post].self, from: data)
            self.posts = decoded
            self.isFetching = false
        } catch {
            print(error)
            self.isFetching = false
        }
    }
}
```



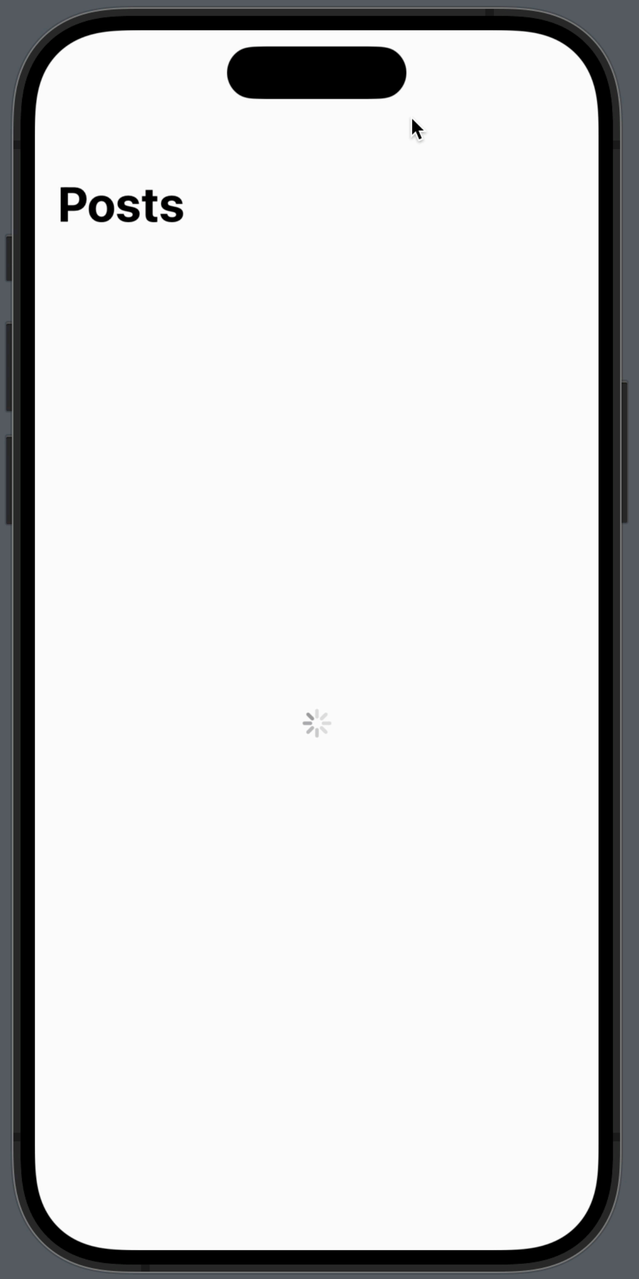
```swift
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []
    var isFetching: Bool = false

    @ObservationIgnored
    @UserDefaultStore(key: "pageLimit") var pageLimit: Int = 10

    @ObservationIgnored
    @UserDefaultStore(key: "pageOffset") var pageOffset: Int = 0

    func fetchPosts() async {
        do {
            self.isFetching = true
            try await Task.sleep(for: .seconds(2))
            let (data, _) = try await URLSession.shared.data(
                from: URL(string: "https://jsonplaceholder.typicode.com/posts")!
            )
            let decoded = try JSONDecoder().decode([Post].self, from: data)
            self.posts = decoded
            self.isFetching = false
        } catch {
            print(error)
            self.isFetching = false
        }
    }
}
```



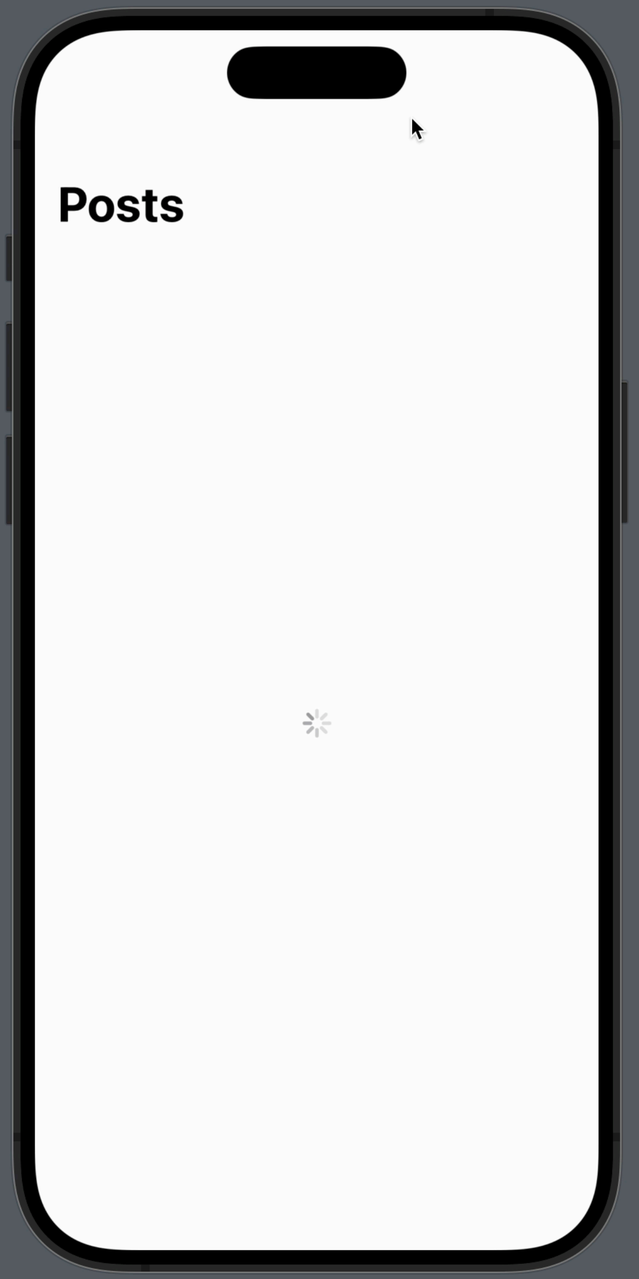
```swift
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []
    var isFetching: Bool = false

    @ObservationIgnored
    @UserDefaultStore(key: "pageLimit") var pageLimit: Int = 10

    @ObservationIgnored
    @UserDefaultStore(key: "pageOffset") var pageOffset: Int = 0

    func fetchPosts() async {
        do {
            self.isFetching = true
            try await Task.sleep(for: .seconds(2))
            let (data, _) = try await URLSession.shared.data(
                from: URL(string: "https://jsonplaceholder.typicode.com/posts")!
            )
            let decoded = try JSONDecoder().decode([Post].self, from: data)
            self.posts = decoded
            self.isFetching = false
        } catch {
            print(error)
            self.isFetching = false
        }
    }
}
```



```swift
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []
    var isFetching: Bool = false

    @ObservationIgnored
    @UserDefaultStore(key: "pageLimit") var pageLimit: Int = 10

    @ObservationIgnored
    @UserDefaultStore(key: "pageOffset") var pageOffset: Int = 0

    func fetchPosts() async {
        do {
            self.isFetching = true
            try await Task.sleep(for: .seconds(2))
            let (data, _) = try await URLSession.shared.data(
                from: URL(string: "https://jsonplaceholder.typicode.com/posts")!
            )
            let decoded = try JSONDecoder().decode([Post].self, from: data)
            self.posts = decoded
            self.isFetching = false
        } catch {
            print(error)
            self.isFetching = false
        }
    }
}
```

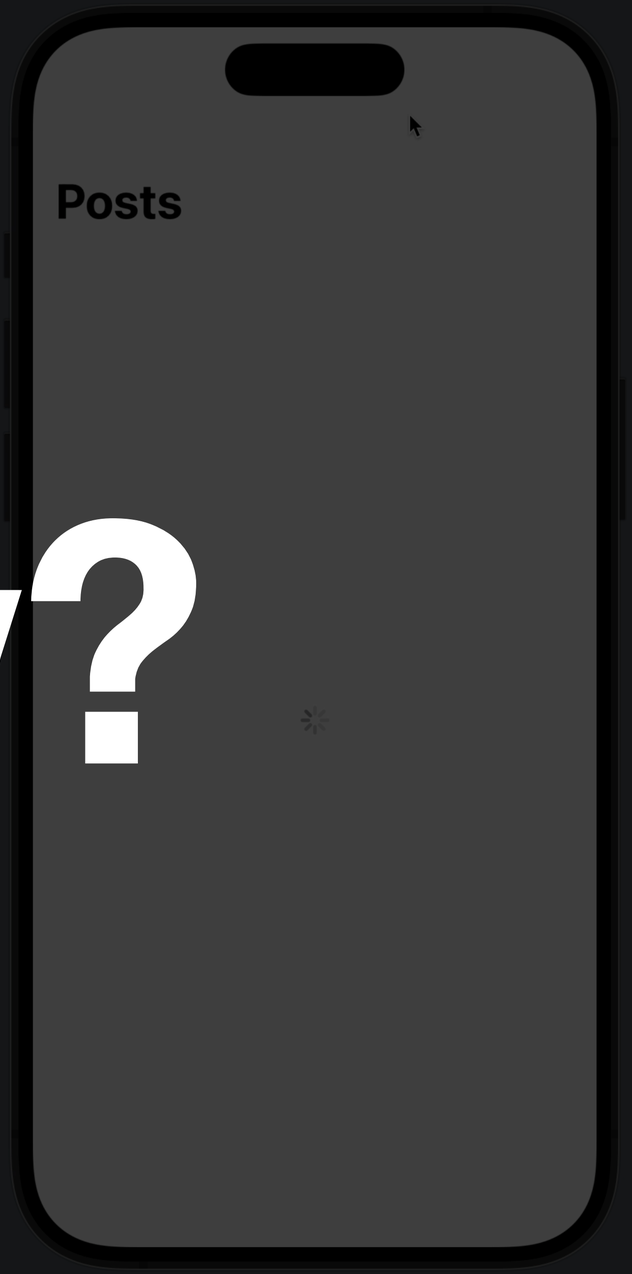


```swift
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []
    var isFetching: Bool = false

    @ObservationIgnored
    @UserDefaultStore(key: "pageLimit") var pageLimit: Int = 10

    @ObservationIgnored
    @UserDefaultStore(key: "pageOffset") var pageOffset: Int = 0

    func fetchPosts() async {
        do {
            self.isFetching = true
            try await Task.sleep(for: .seconds(2))
            let (data, _) = try await URLSession.shared.data(
                from: URL(string: "https://jsonplaceholder.typicode.com/posts")!
            )
            let decoded = try JSONDecoder().decode([Post].self, from: data)
            self.posts = decoded
            self.isFetching = false
        } catch {
            print(error)
            self.isFetching = false
        }
    }
}
```



```swift
@MainActor
struct ContentView: View {
    @State var viewModel: PostViewModel = .init()
    var count: Int = 0

    var body: some View {
        Group {
            if viewModel.isFetching {
                ProgressView()
            } else {
                List {
                    Fo                Model.posts, i \.id) { st in
                        avig onLink(destin on: {
                            tDe lVie pos post)
                        ) {
                            T (po ti )
                        }
                    }
                }
            }
        }
        .task { await viewModel.fetchPosts() }
        .navigationTitle("Posts")
        .onAppear { print("On Appear Called") }
        .onDisappear { print("On Disappear Called") }
    }
}
```

But How?



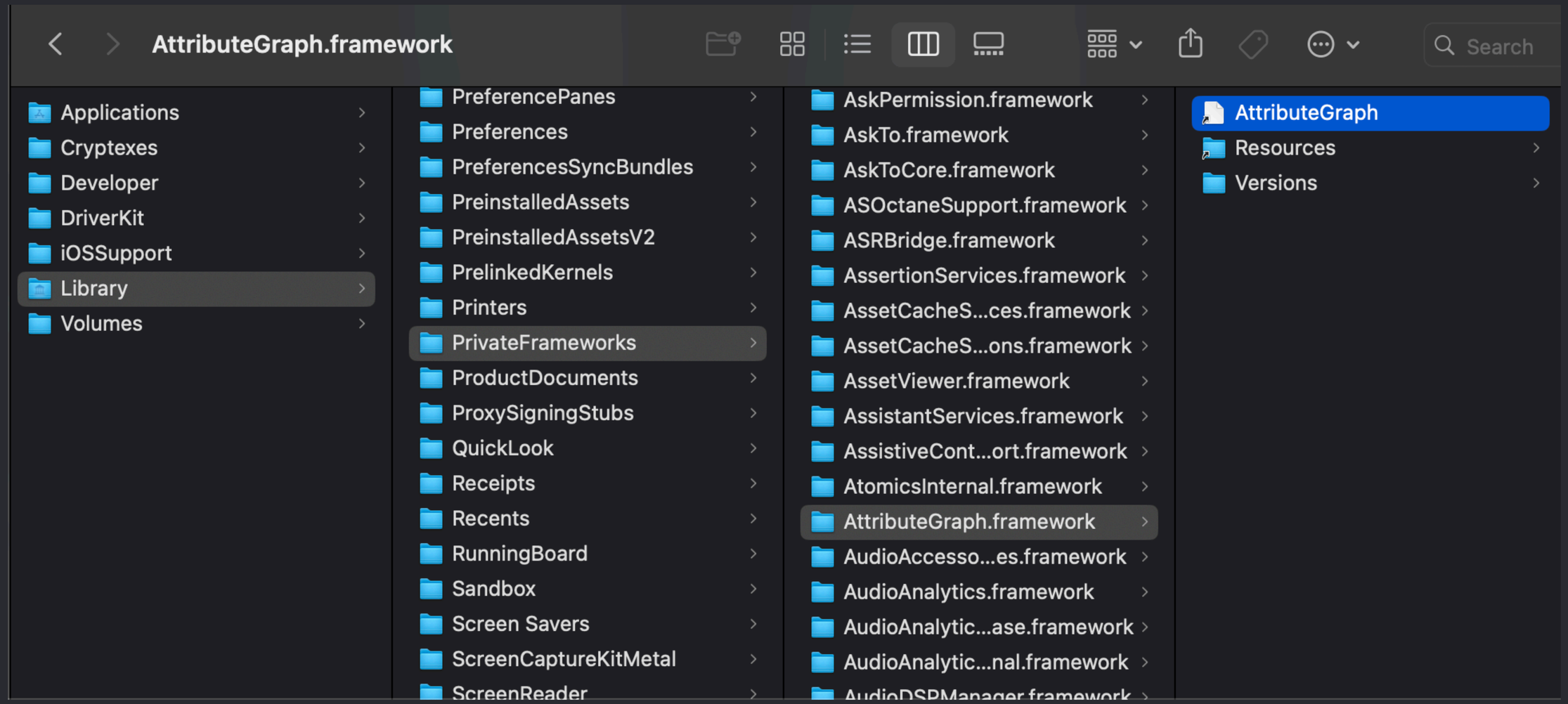
Attribute Graph

Other Side of SwiftUI

Private Framework*

Ratnesh Jain

=== AttributeGraph: cycle detected through attribute 290 ===
=== AttributeGraph: cycle detected through attribute 320 ===
=== AttributeGraph: cycle detected through attribute 360 ===
=== AttributeGraph: cycle detected through attribute 420 ===



/System/Library/PrivateFrameworks/AttributeGraph.framework

Attribute Graph

Other Side of SwiftUI

- Reads the View’s declaration.
- Creates a View Tree
- Keep track of State, Binding and other view dependency
- Creates a Persistent Reader Tree
- Observe the state, dependency changes
  - Throw away existing View Tree
  - Creates a new View tree
  - Re-render the Reader tree with smart difference.

Attribute Graph

Read View’s Declaration

```
struct ContentView: View {
    @State private var isLoading: Bool = false
    var body: some View {
        Group {
            if isLoading {
                ProgressView()
            } else {
                VStack {
                    Image(systemName: "globe")
                        .imageScale(.large)
                        .foregroundStyle(.tint)
                    Text("Hello, world!")
                }
                .padding()
            }
        }
    }
}
```

Attribute Graph

Read View's Declaration

```
struct ContentView: View {
    @State private var isLoading: Bool = false
    var body: some View {
        Group {
            if isLoading {
                ProgressView()
            } else {
                VStack {
                    Image(systemName: "globe")
                        .imageScale(.large)
                        .foregroundStyle(.tint)
                    Text("Hello, world!")
                }
                .padding()
            }
        }
    }
}
```

Opaque Type

Attribute Graph

Read View's Declaration

```
struct ContentView: View {
    @State private var isLoading: Bool = false
    var body: some View {
        Group {
            if isLoading {
                ProgressView()
            } else {
                VStack {
                    Image(systemName: "globe")
                        .imageScale(.large)
                        .foregroundStyle(.tint)
                    Text("Hello, world!")
                }
                .padding()
            }
        }
    }
}
```

Opaque Type

- Type information is not directly visible to user/dev
- Compiler knows all type information

Attribute Graph

Read View’s Declaration

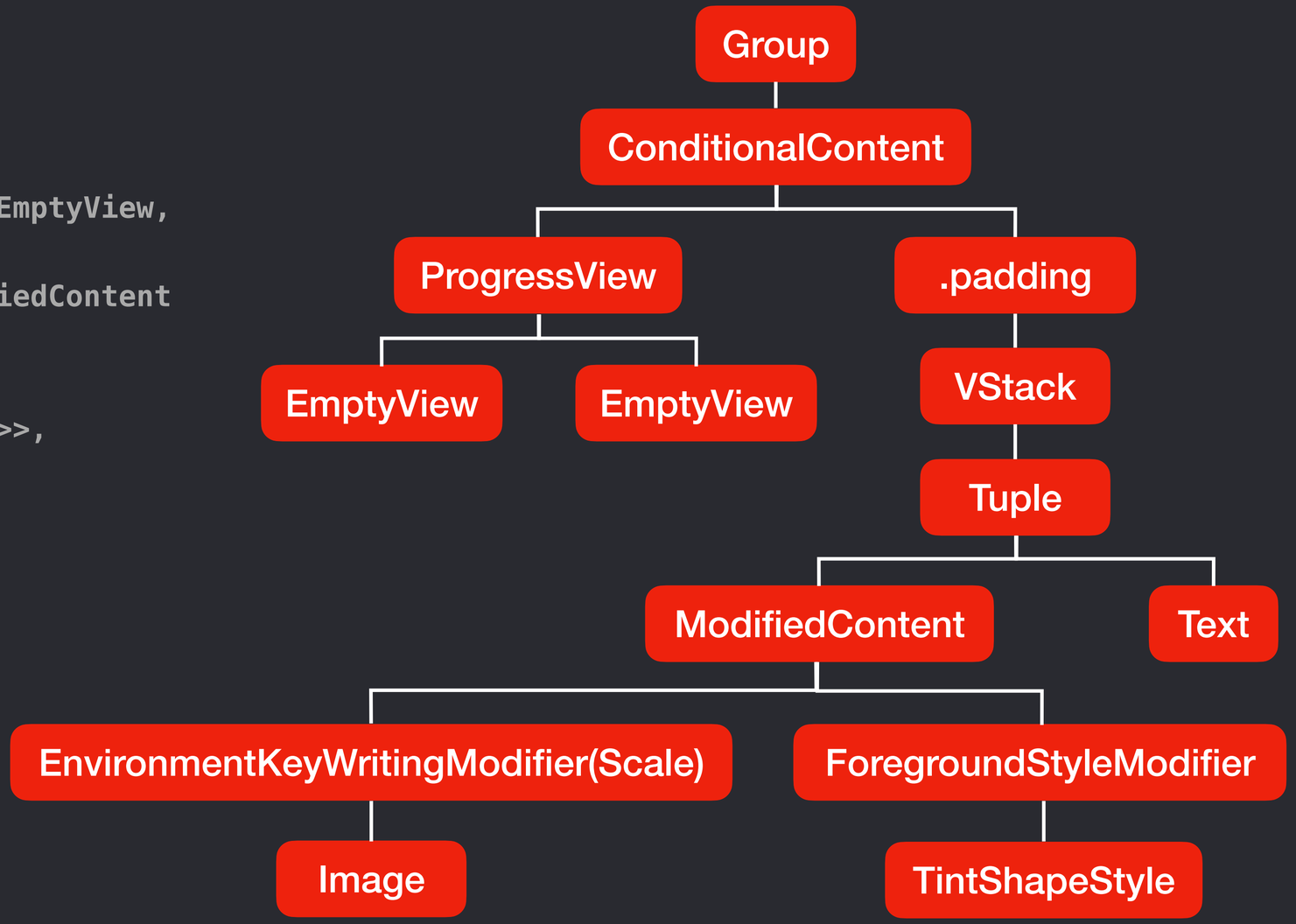
```
extension View {
    func mirror() {
        print(Mirror(reflecting: self).subjectType)
    }
}
```

```
Group<_ConditionalContent<ProgressView<EmptyView, EmptyView>,
ModifiedContent<VStack<TupleView<(ModifiedContent<ModifiedContent<Image,
_EnvironmentKeyWritingModifier<Scale>>, _ForegroundStyleModifier<TintShapeStyle>>, Text)>>, _PaddingLayout>>>
```

Attribute Graph

View Tree

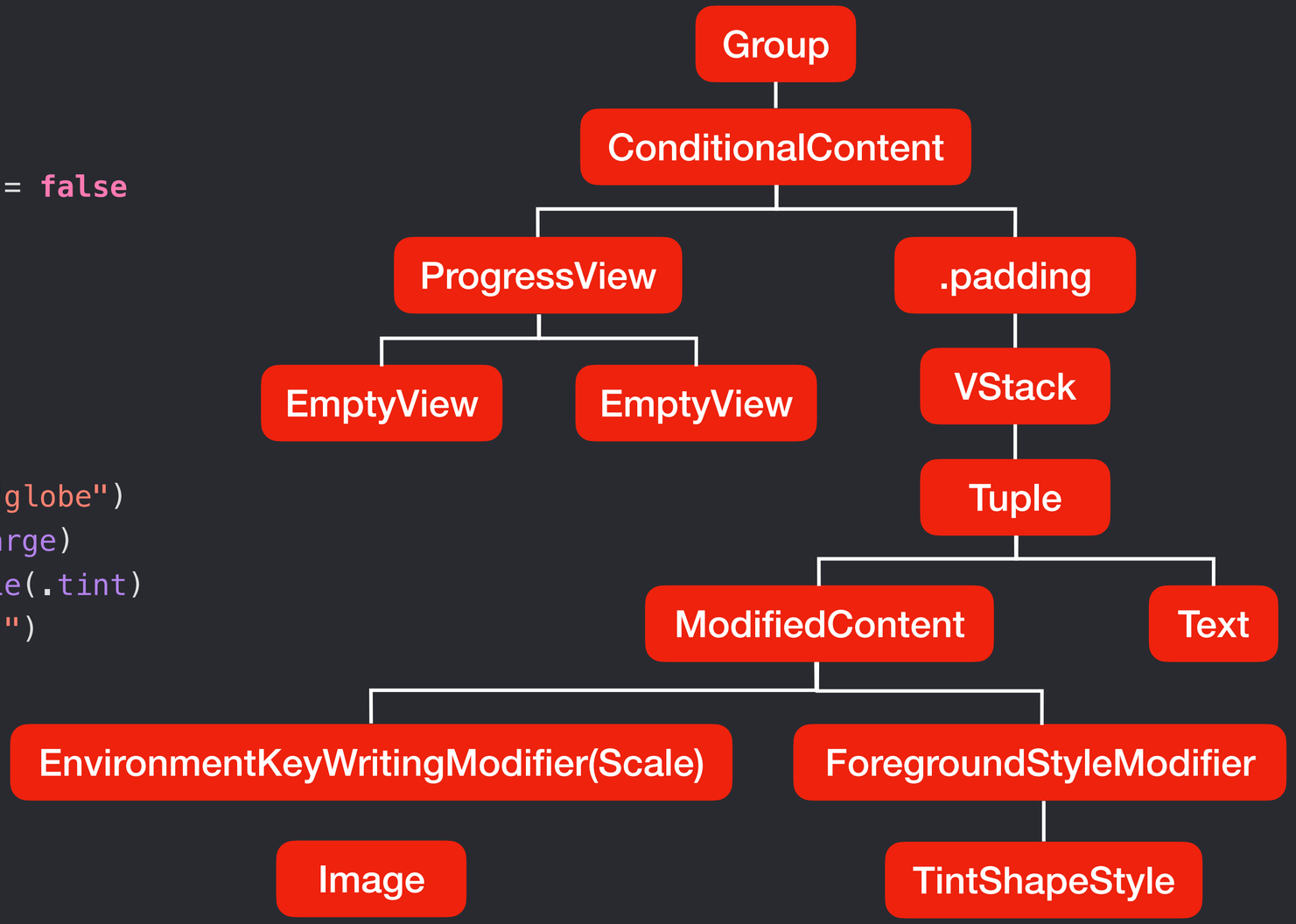
```
Group<_ConditionalContent<ProgressView<EmptyView,
EmptyView>,
ModifiedContent<VStack<TupleView<(ModifiedContent
<ModifiedContent<Image,
_EnvironmentKeyWritingModifier<Scale>>,
_ForegroundStyleModifier<TintShapeStyle>>,
Text)>>, _PaddingLayout>>>
```



Attribute Graph

View Tree

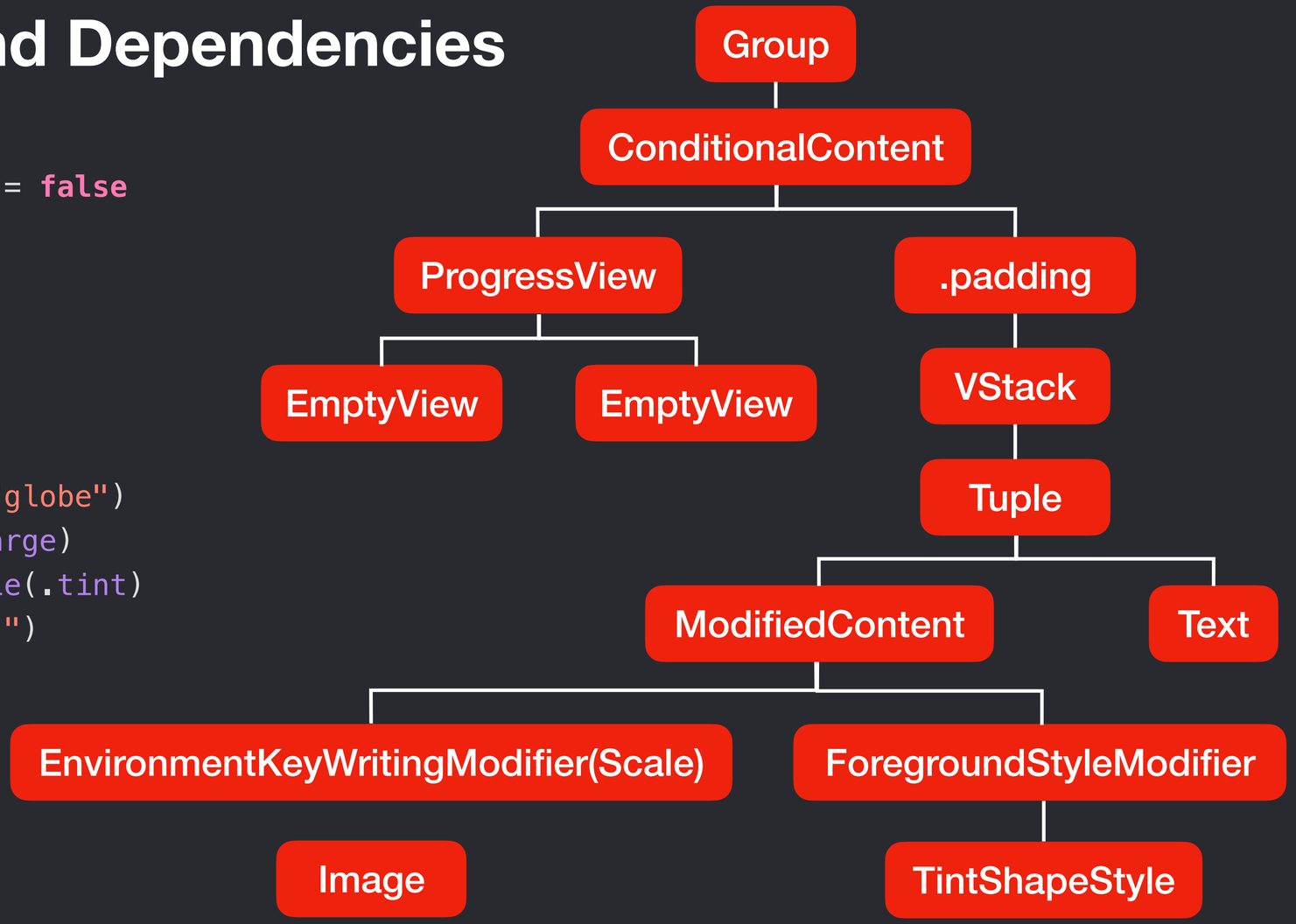
```
struct ContentView: View {
    @State private var isLoading: Bool = false
    var body: some View {
        Group {
            if isLoading {
                ProgressView()
            } else {
                VStack {
                    Image(systemName: "globe")
                        .imageScale(.large)
                        .foregroundStyle(.tint)
                    Text("Hello, world!")
                }
                .padding()
            }
        }
    }
}
```



Attribute Graph

Keep Track of State and Dependencies

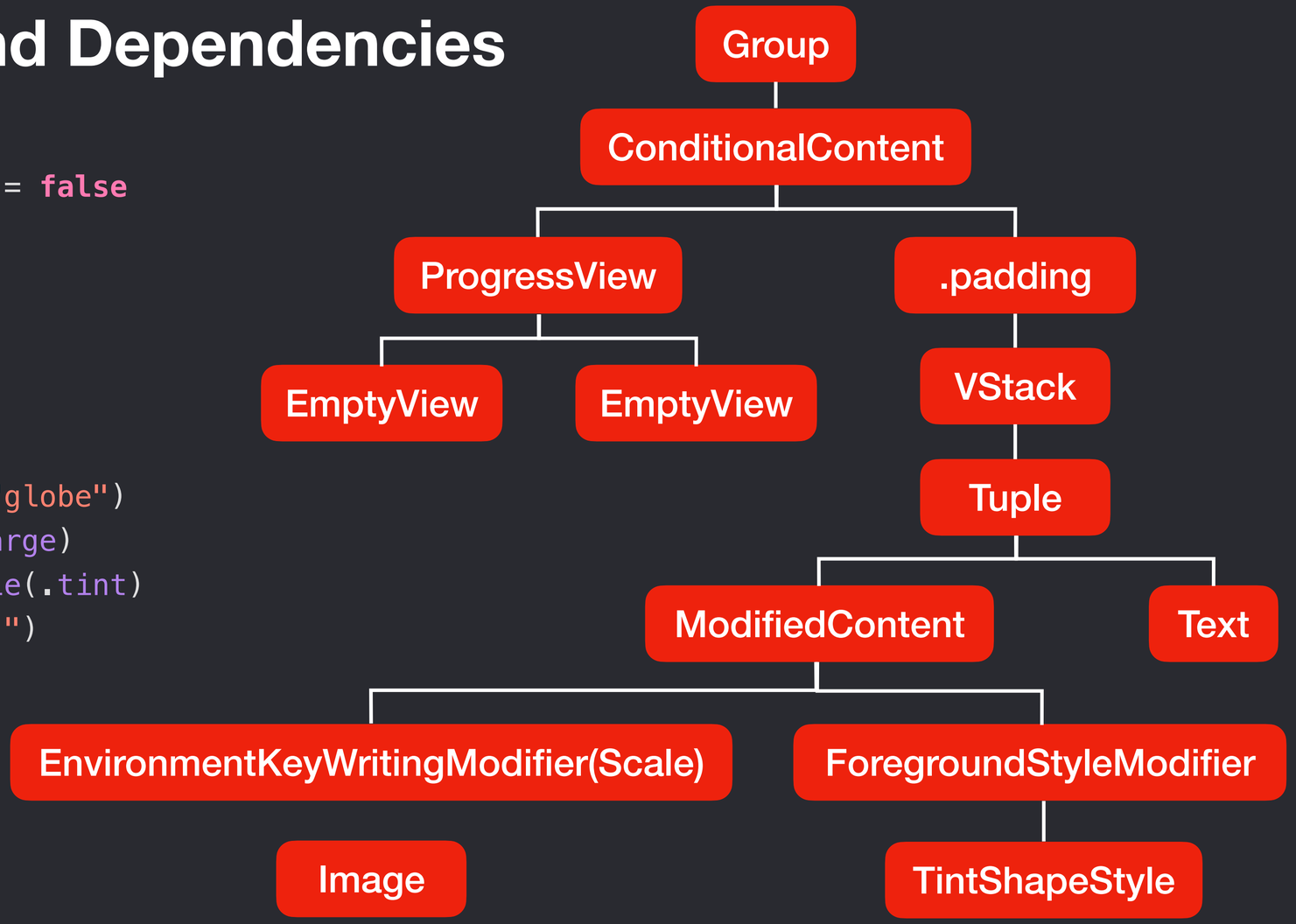
```
struct ContentView: View {
    @State private var isLoading: Bool = false
    var body: some View {
        Group {
            if isLoading {
                ProgressView()
            } else {
                VStack {
                    Image(systemName: "globe")
                        .imageScale(.large)
                        .foregroundStyle(.tint)
                    Text("Hello, world!")
                }
                .padding()
            }
        }
    }
}
```



Attribute Graph

Keep Track of State and Dependencies

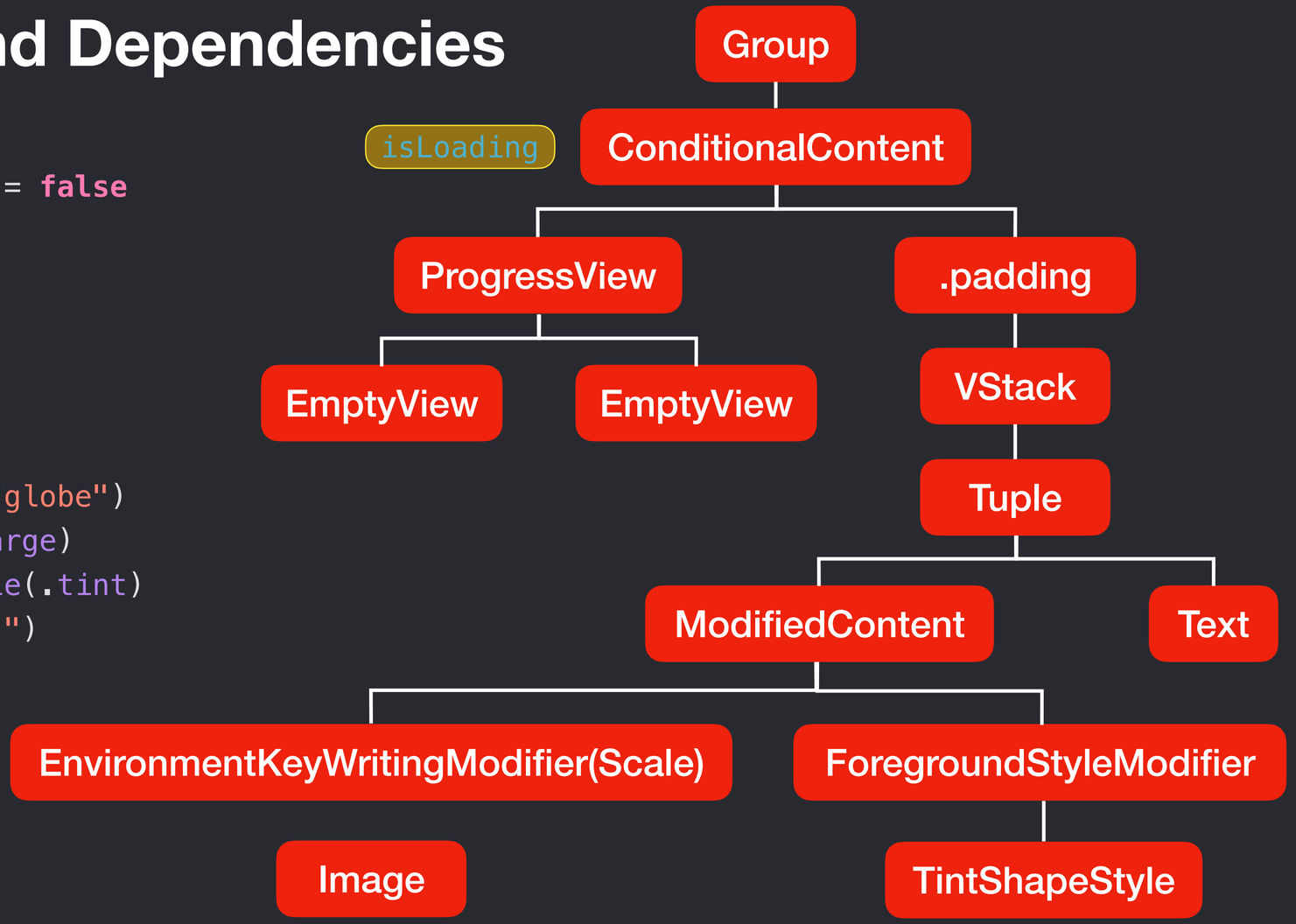
```
struct ContentView: View {
    @State private var isLoading: Bool = false
    var body: some View {
        Group {
            if isLoading {
                ProgressView()
            } else {
                VStack {
                    Image(systemName: "globe")
                        .imageScale(.large)
                        .foregroundStyle(.tint)
                    Text("Hello, world!")
                }
                .padding()
            }
        }
    }
}
```



Attribute Graph

Keep Track of State and Dependencies

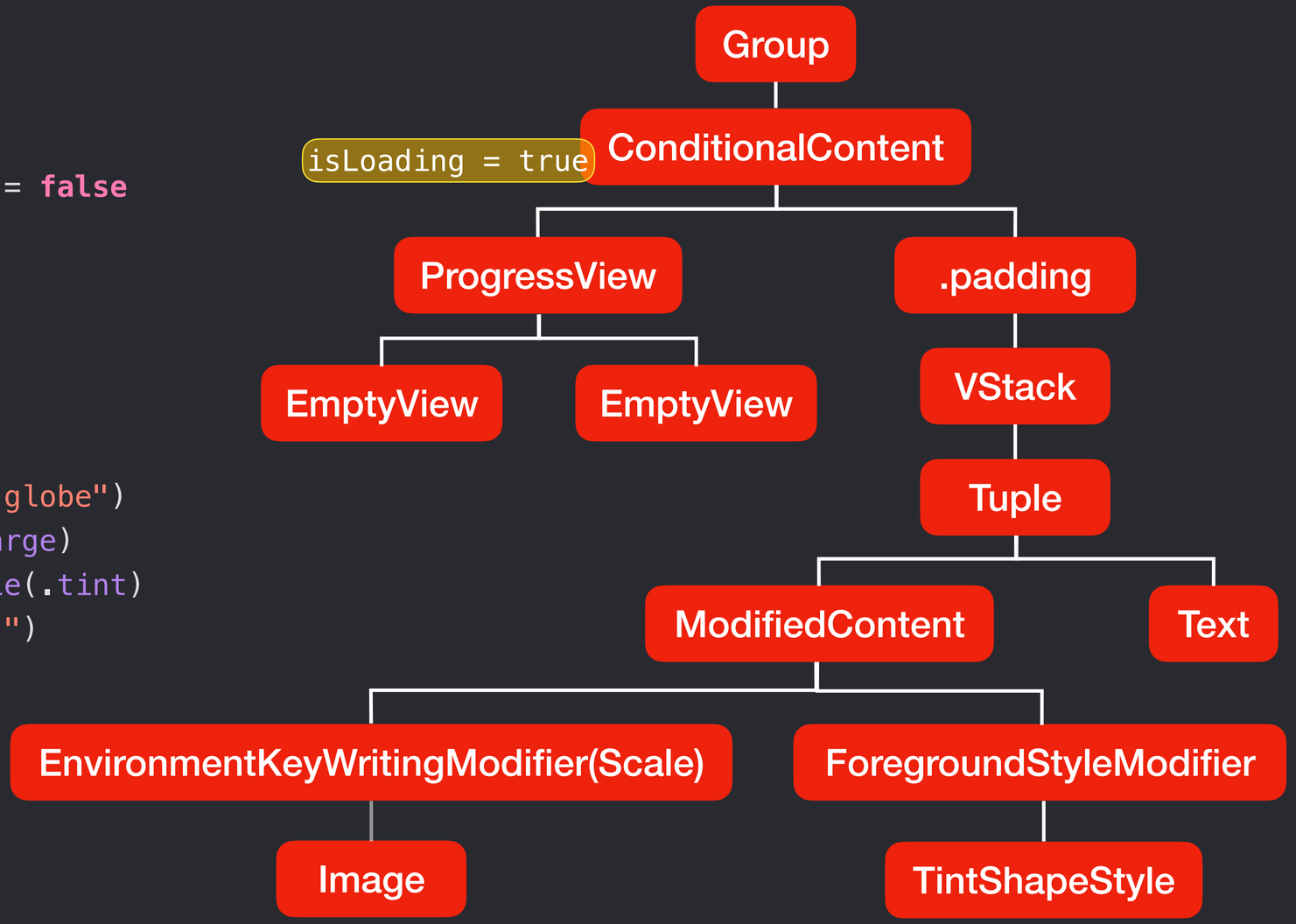
```
struct ContentView: View {
    @State private var isLoading: Bool = false
    var body: some View {
        Group {
            if isLoading {
                ProgressView()
            } else {
                VStack {
                    Image(systemName: "globe")
                        .imageScale(.large)
                        .foregroundStyle(.tint)
                    Text("Hello, world!")
                }
                .padding()
            }
        }
    }
}
```



Attribute Graph

Render Tree*

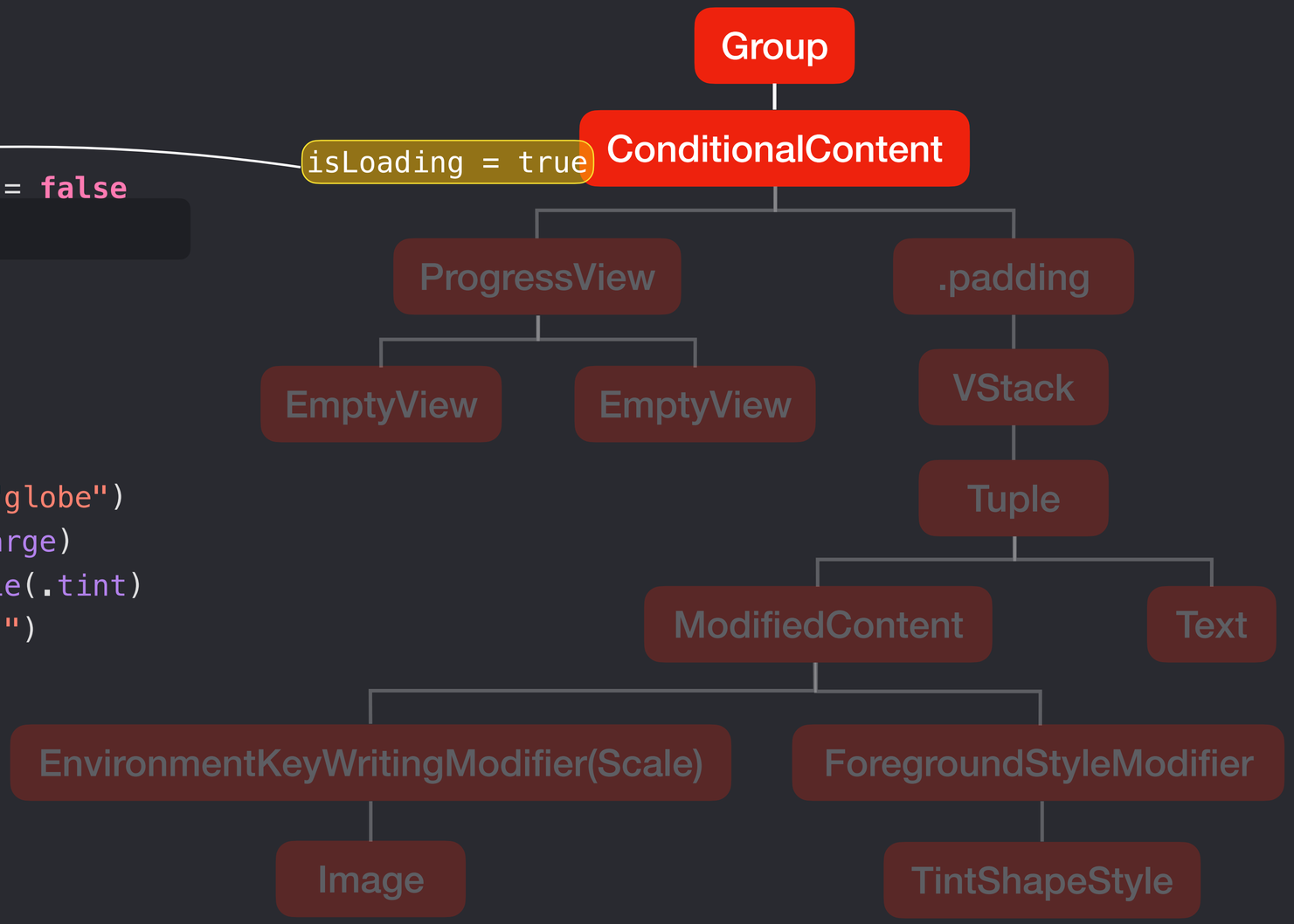
```
struct ContentView: View {
    @State private var isLoading: Bool = false
    var body: some View {
        Group {
            if isLoading {
                ProgressView()
            } else {
                VStack {
                    Image(systemName: "globe")
                        .imageScale(.large)
                        .foregroundStyle(.tint)
                    Text("Hello, world!")
                }
                .padding()
            }
        }
    }
}
```



Attribute Graph

Render Tree*

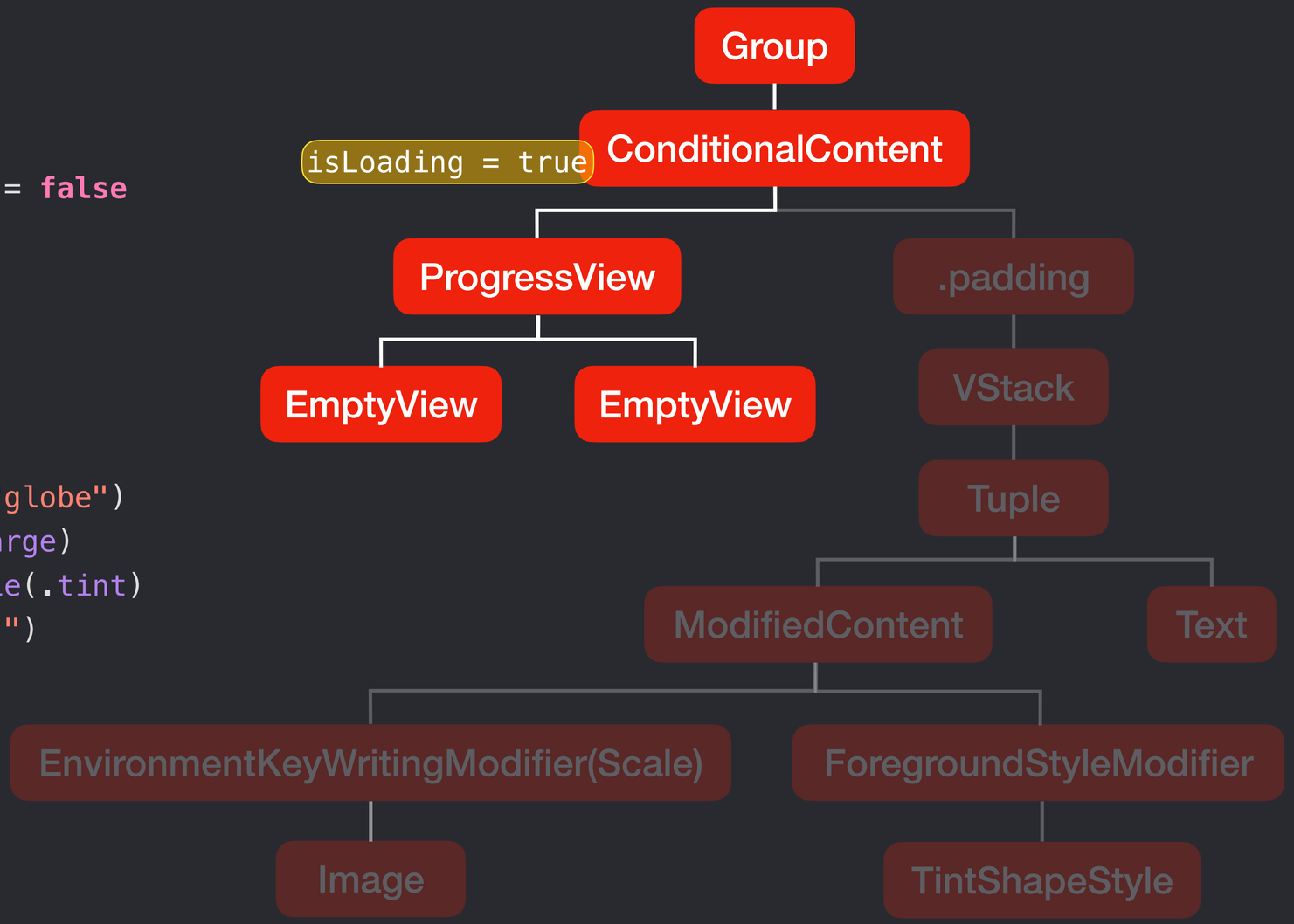
```
struct ContentView: View {
    @State private var isLoading: Bool = false
    var body: some View {
        Group {
            if isLoading {
                ProgressView()
            } else {
                VStack {
                    Image(systemName: "globe")
                        .imageScale(.large)
                        .foregroundStyle(.tint)
                    Text("Hello, world!")
                }
                .padding()
            }
        }
    }
}
```



Attribute Graph

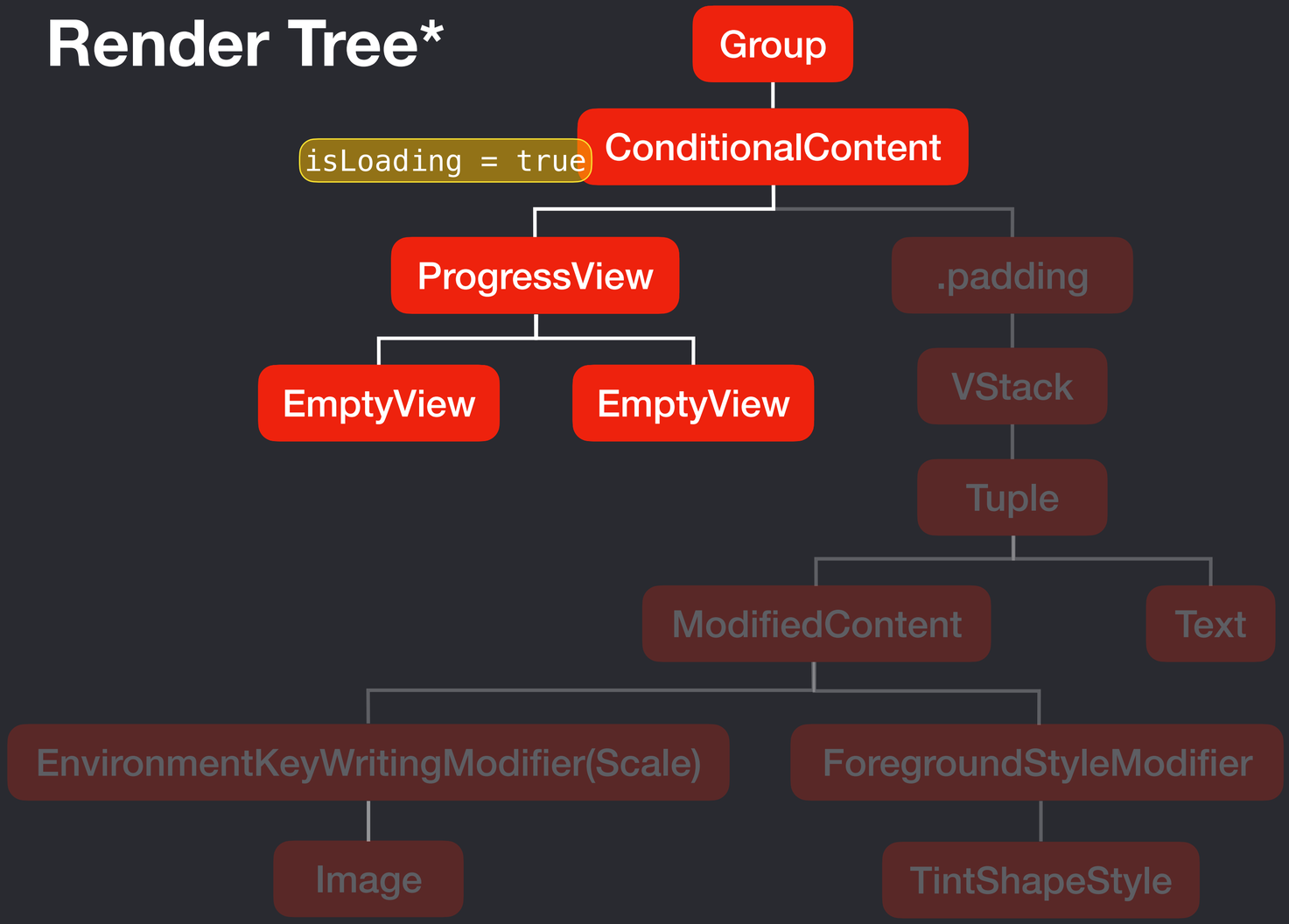
Render Tree*

```
struct ContentView: View {
    @State private var isLoading: Bool = false
    var body: some View {
        Group {
            if isLoading {
                ProgressView()
            } else {
                VStack {
                    Image(systemName: "globe")
                        .imageScale(.large)
                        .foregroundStyle(.tint)
                    Text("Hello, world!")
                }
                .padding()
            }
        }
    }
}
```



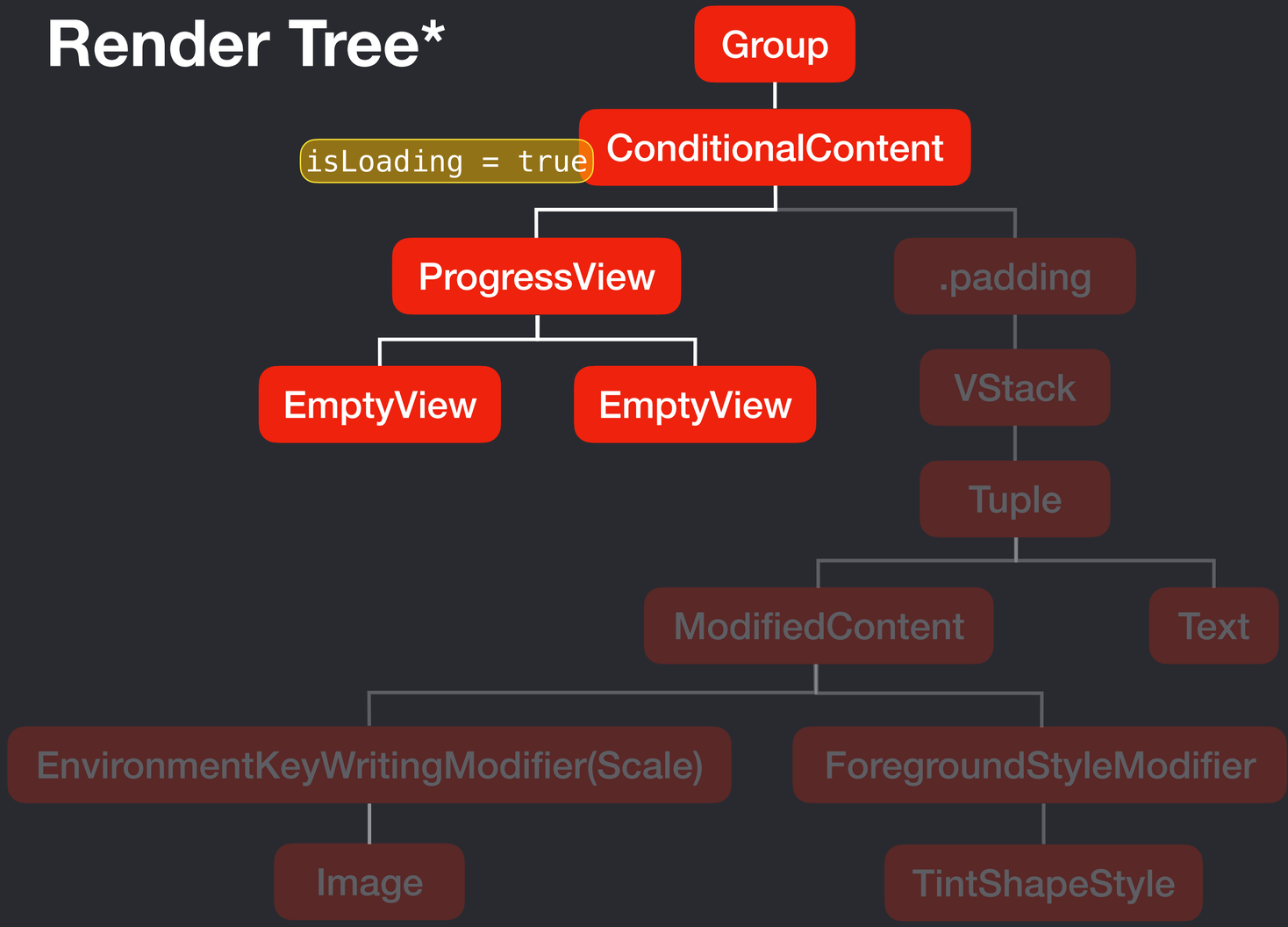
Attribute Graph

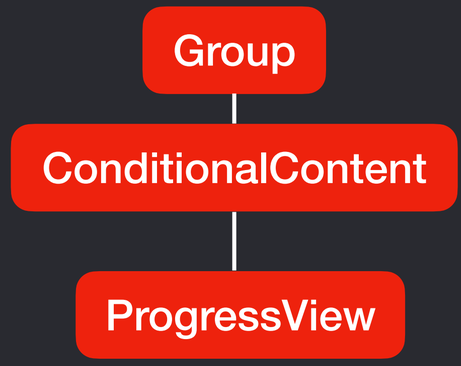
Render Tree*



Attribute Graph

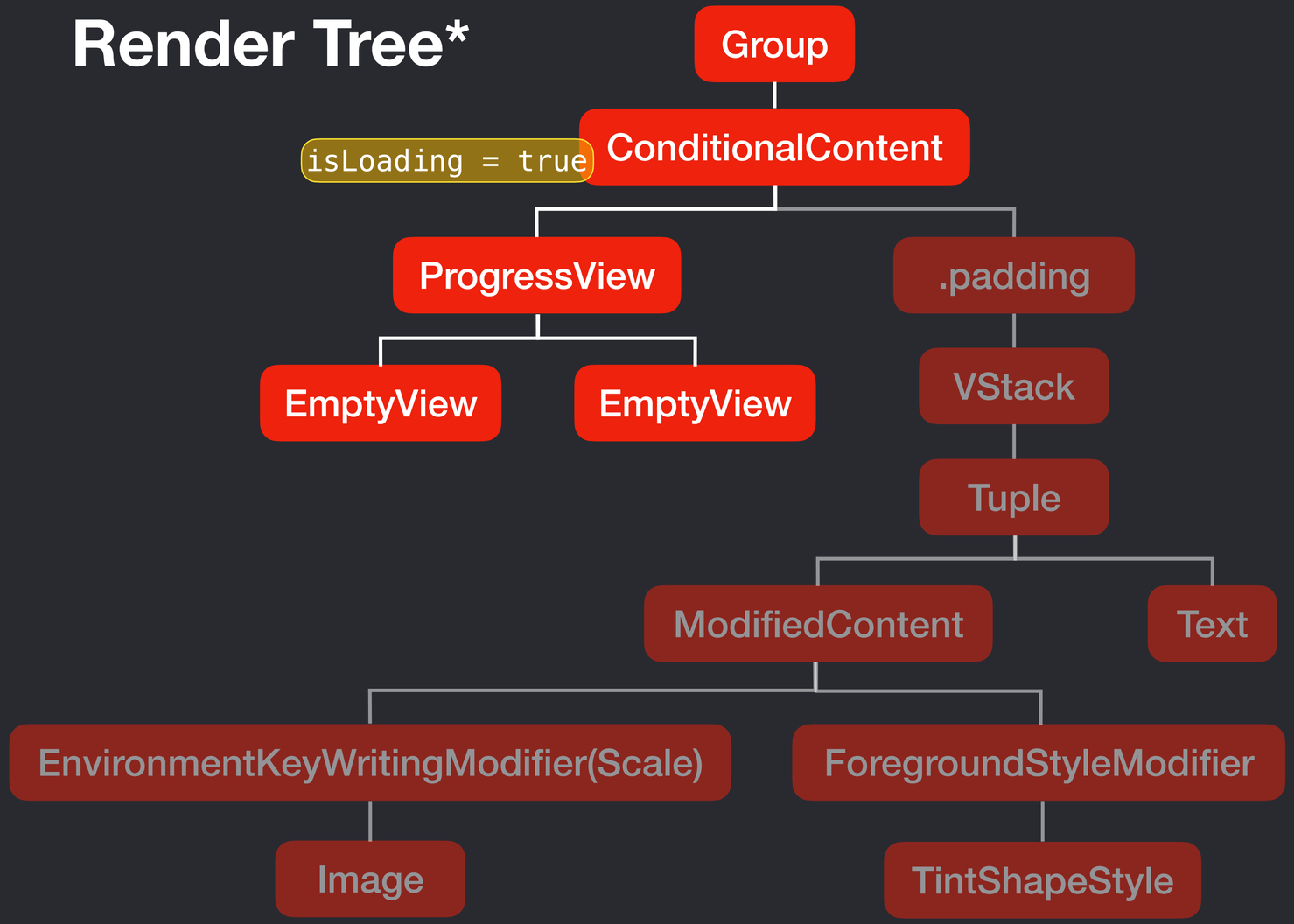
Render Tree*

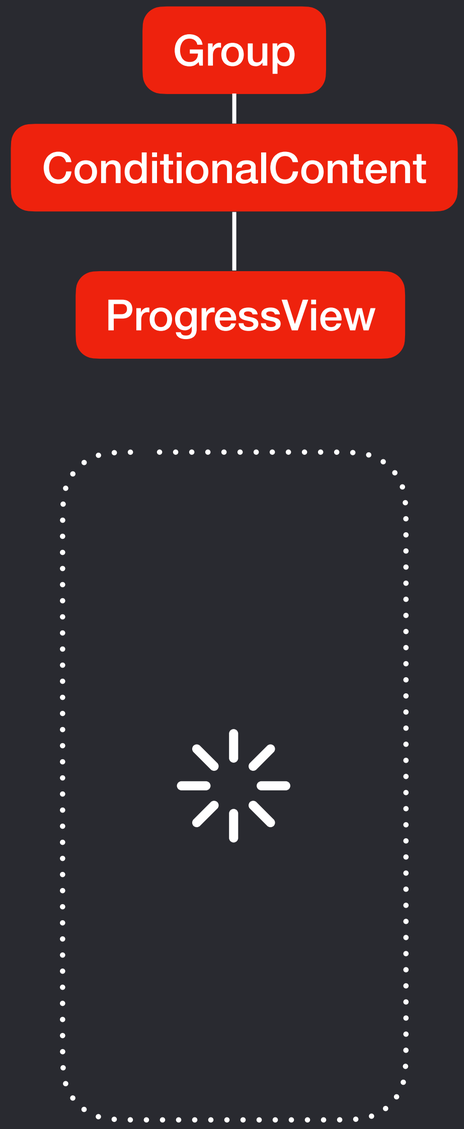




Attribute Graph

Render Tree*

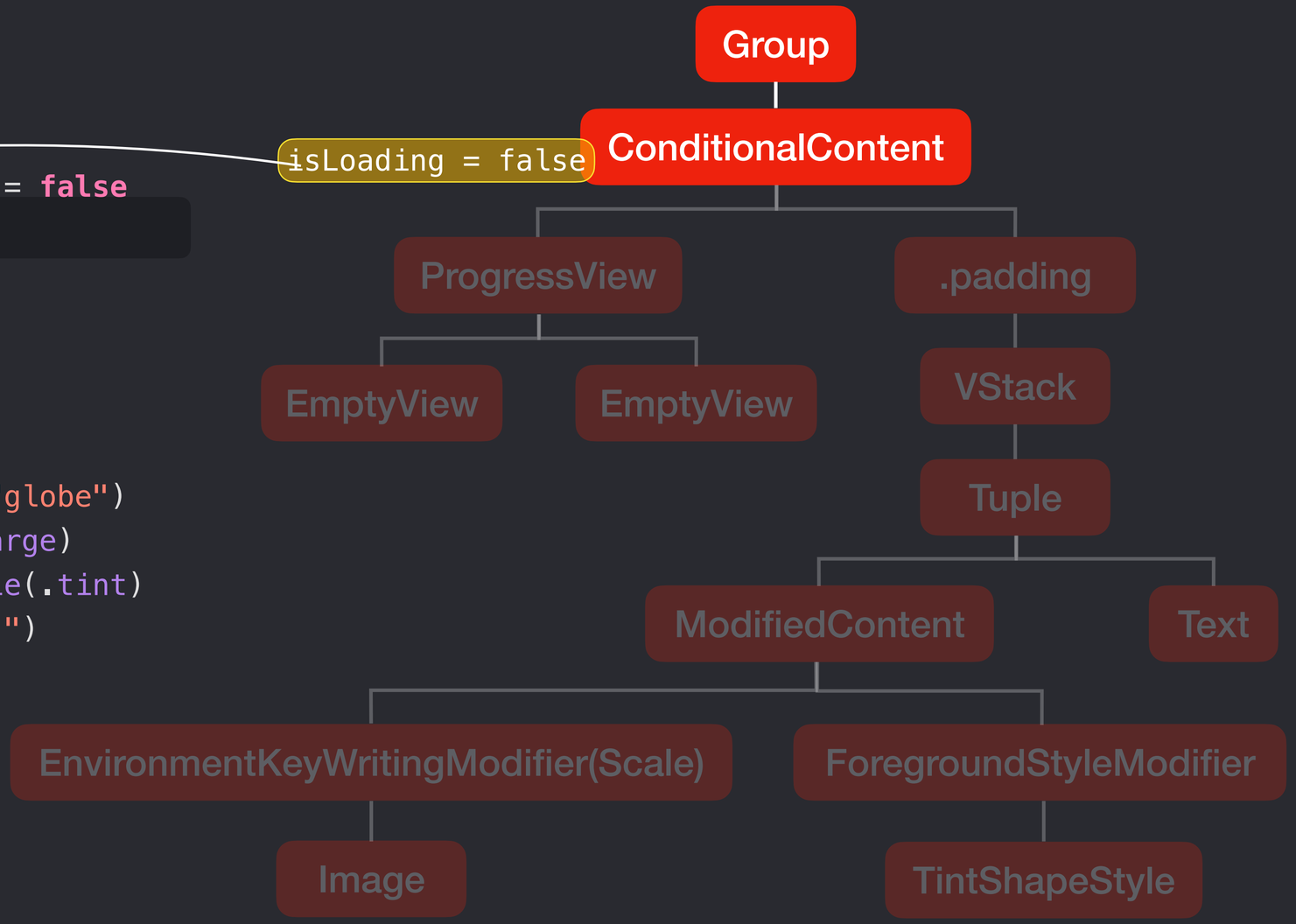




Attribute Graph

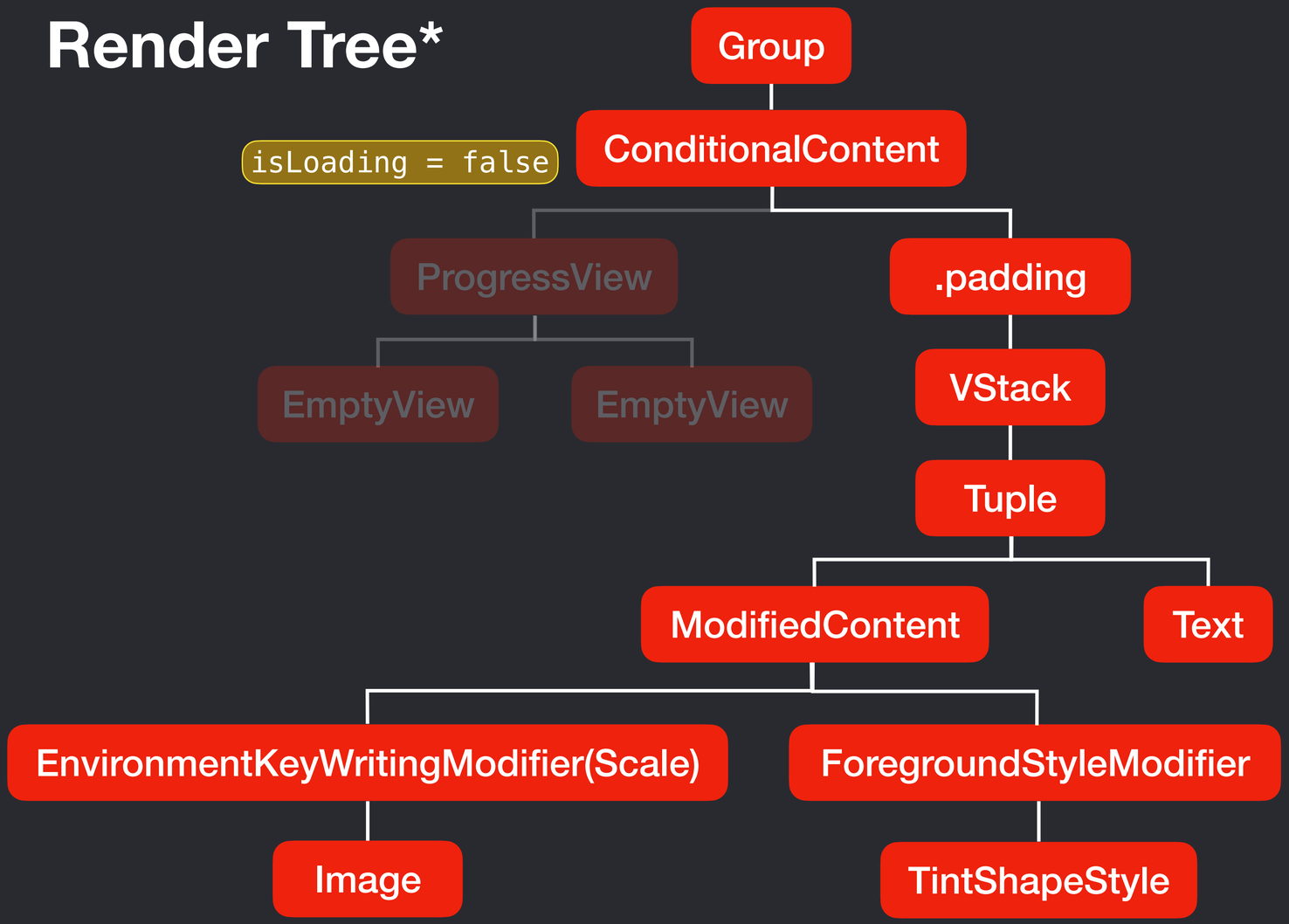
Render Tree*

```
struct ContentView: View {
    @State private var isLoading: Bool = false
    var body: some View {
        Group {
            if isLoading {
                ProgressView()
            } else {
                VStack {
                    Image(systemName: "globe")
                        .imageScale(.large)
                        .foregroundStyle(.tint)
                    Text("Hello, world!")
                }
                .padding()
            }
        }
    }
}
```



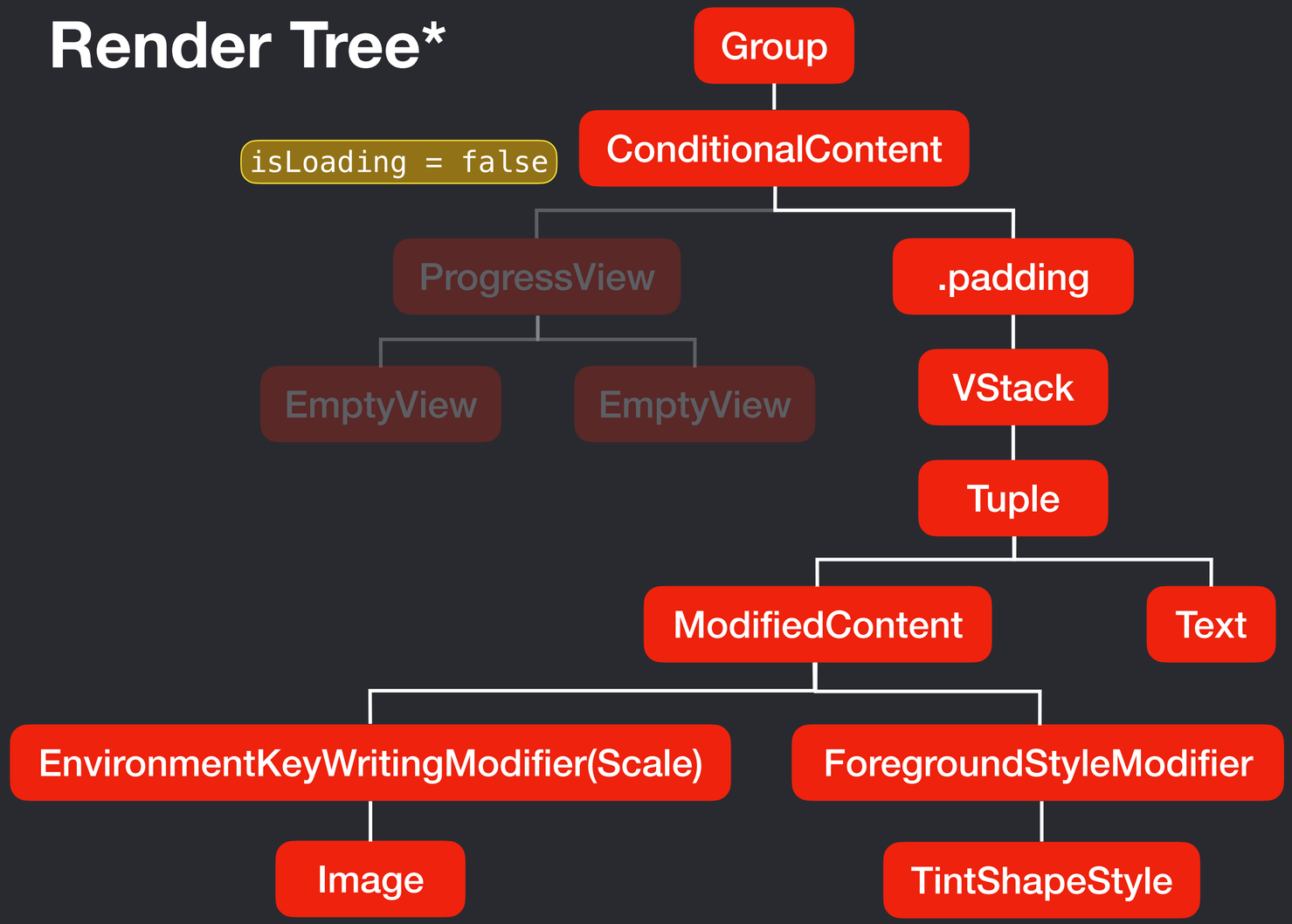
Attribute Graph

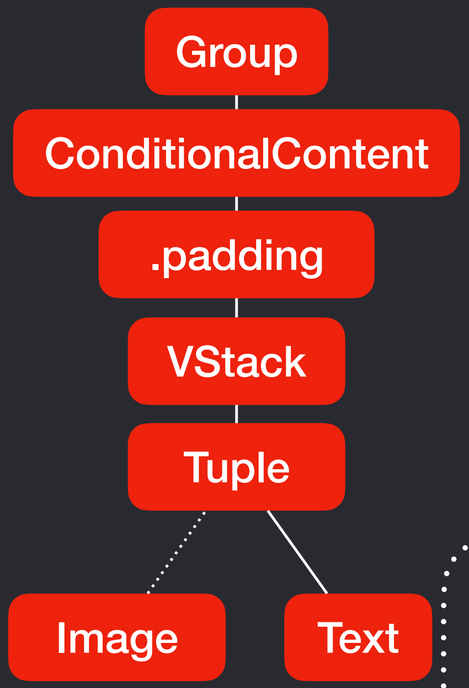
Render Tree*



Attribute Graph

Render Tree*





Attribute Graph

Other Side of SwiftUI

- Reads the View’s declaration.
- Creates a View Tree
- Keep track of State, Binding and other view dependency
- Creates a Persistent Reader Tree
- Observe the state, dependency changes
  - Throw away existing View Tree
  - Creates a new View tree
  - Re-render the Reader tree with smart difference.

Attribute Graph

For more details, please visit

https://chris.eidhof.nl/presentations/day-in-the-life/

Attribute Graph

For more details, please visit

A Day in a Life of SwiftUI View

https://chris.eidhof.nl/presentations/day-in-the-life/

```swift
import SwiftUI

@MainActor
struct ContentView: View {
    @State var viewModel: PostsViewModel = .init()

    var body: some View {
        List {
            ForEach(viewModel.posts, id: \.id) { post in
                Text(post.title)
            }
        }
        .task {
            await viewModel.fetchPosts()
        }
        .navigationTitle("Posts")
    }
}

#Preview {
    NavigationStack {
        ContentView()
    }
}
```

Actor Isolation

Property Wrapper

State | Binding | FocusState

ObservedObject | StateObject

Environment | AppStorage

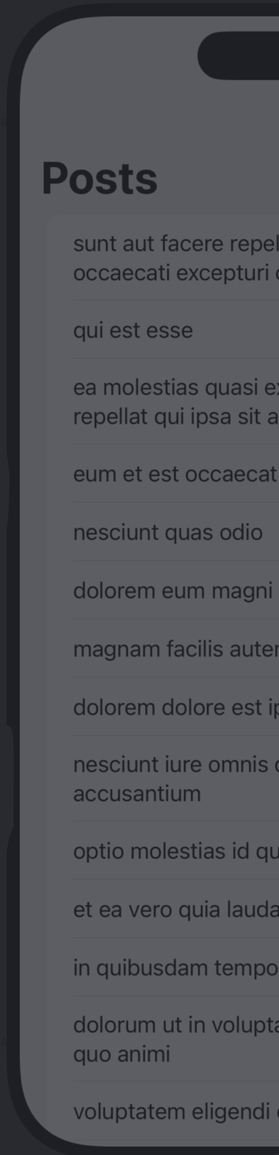
EnvironmentObject | Bindable

Result Builder

ViewBuilder

TableBuilder

SceneBuilder



Result Builder

@_functionBuilder SE-0289

Result Builder

@_functionBuilder SE-0289

- Allows to write in DSL syntax
- Easy to apply condition

Demo

Array Builder

Environment

An Modern Dependency System

Ratnesh Jain

How to Pass Information around

By any means

- Global Variables
- Pass as arguments
- Delegate
- Post a Notification
- Store in UserDefaults/FileSystem and access it somewhere else.
- Use Thread Dictionary
- Task Locals

Environment

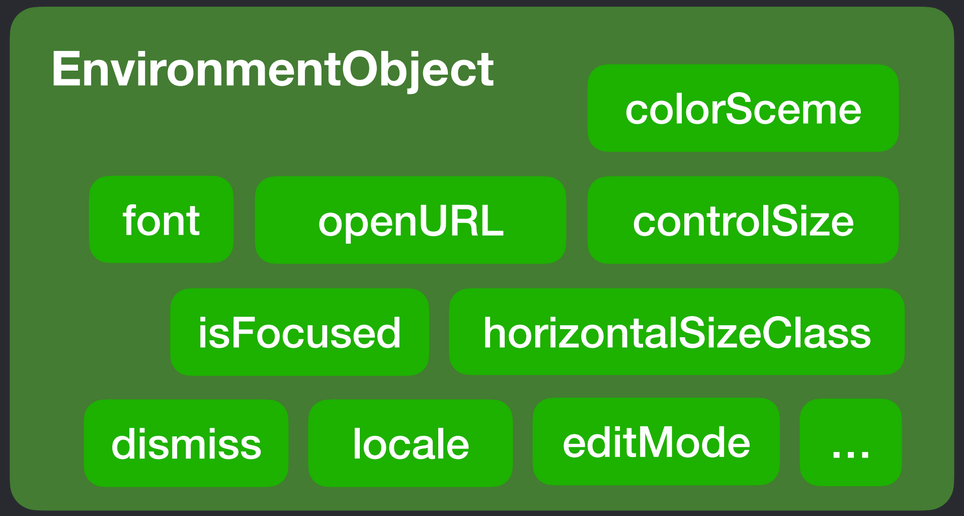
An Modern Dependency System

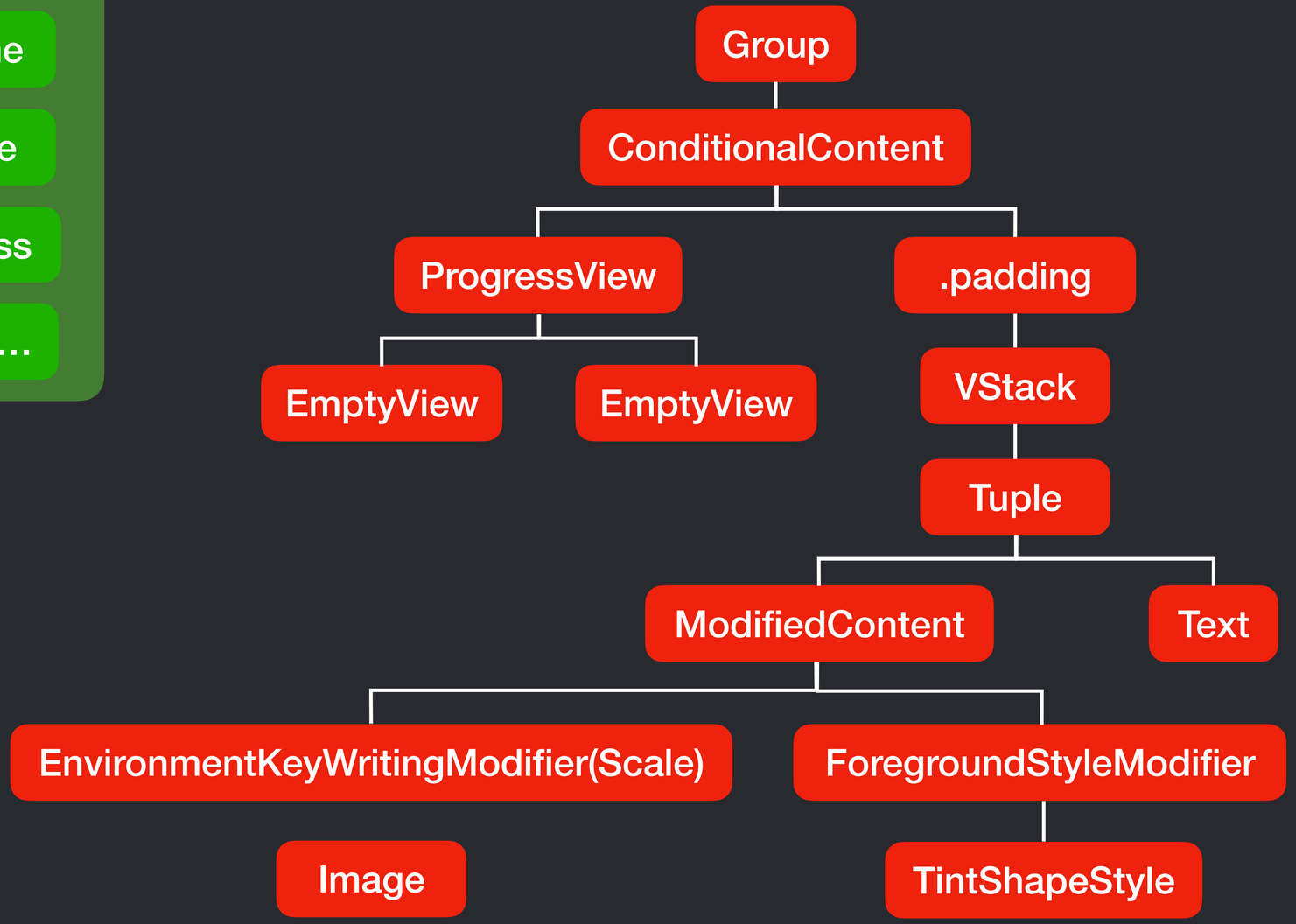
Kind of Global Variable

Thread Safe

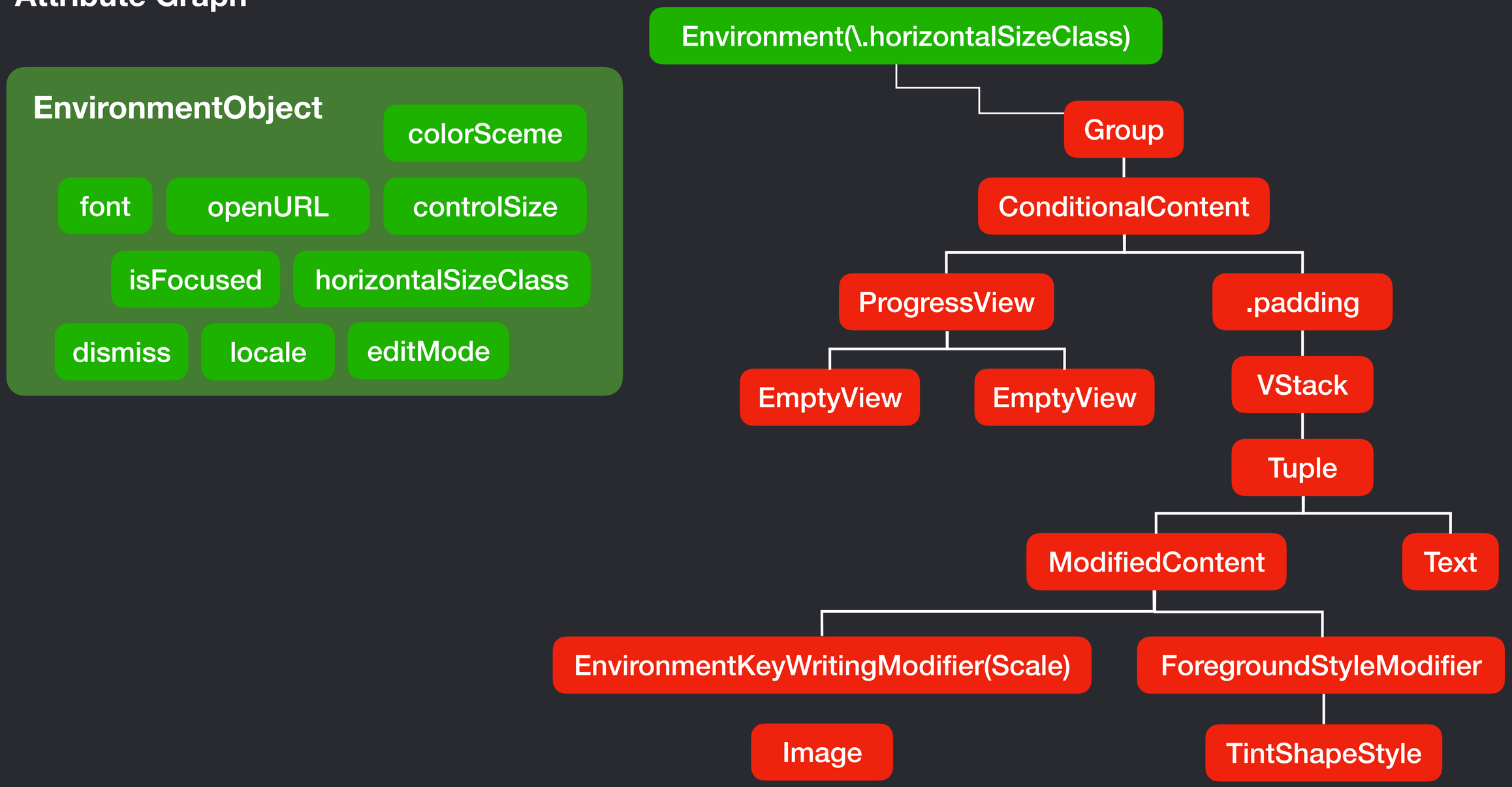
Task Locals

Attribute Graph

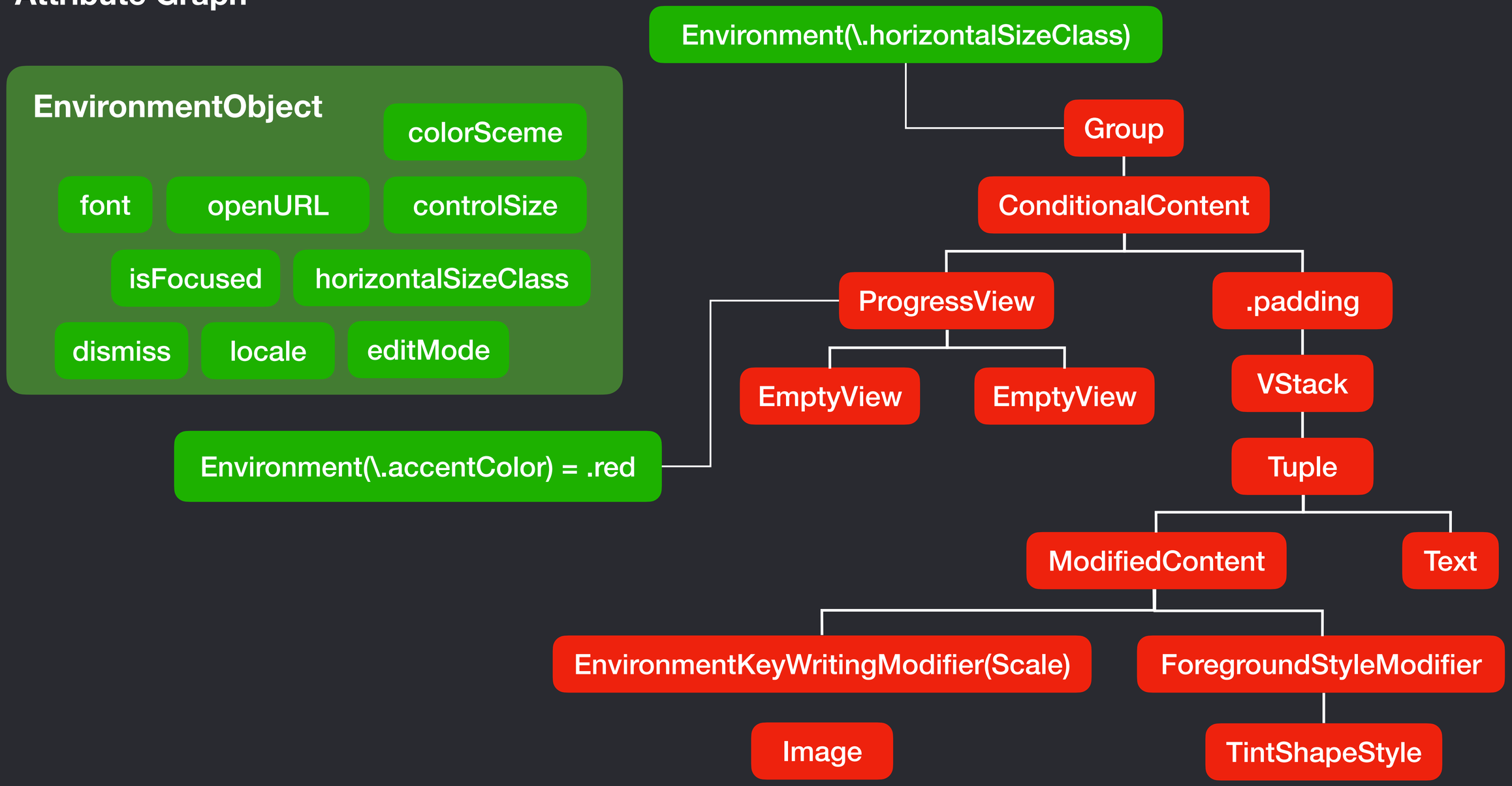




Attribute Graph



Attribute Graph



Demo

Task Local and Environment

```swift
import SwiftUI

@MainActor
struct ContentView: View {
    @State var viewModel: PostsViewModel = .init()

    var body: some View {
        List {
            ForEach(viewModel.posts, id: \.id) { post in
                Text(post.title)
            }
        }
        .task {
            await viewModel.fetchPosts()
        }
        .navigationTitle("Posts")
    }
}

#Preview {
    NavigationStack {
        ContentView()
    }
}
```

Actor Isolation

Property Wrapper

State | Binding | FocusState

ObservedObject | StateObject

Environment | AppStorage

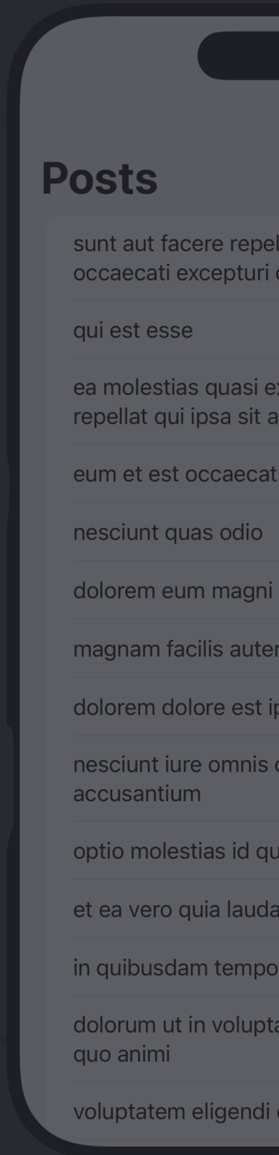
EnvironmentObject | Bindable

Result Builder

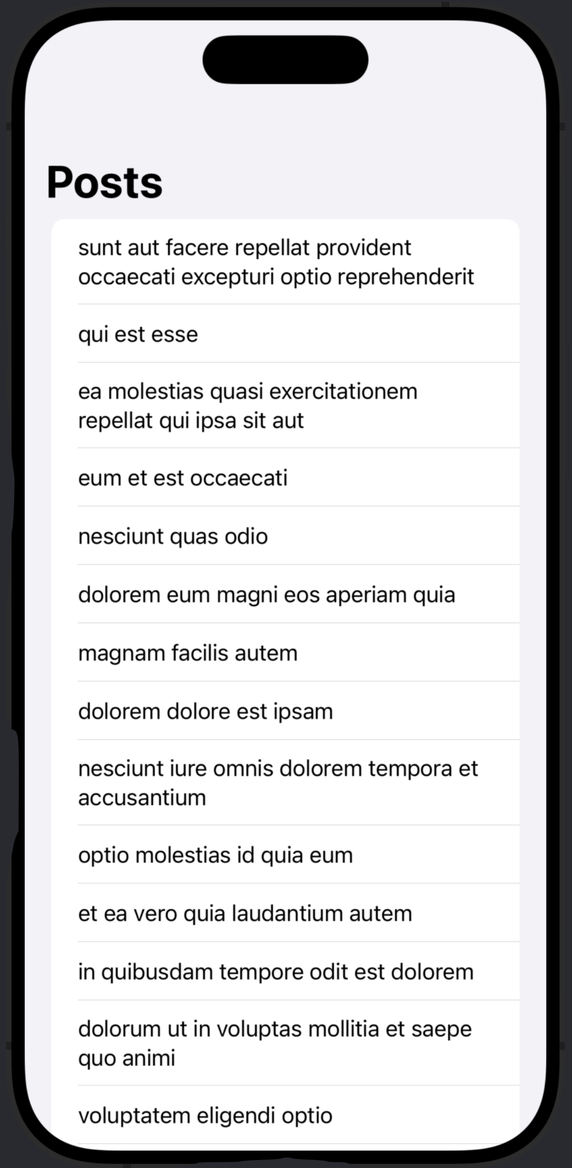
ViewBuilder

TableBuilder

SceneBuilder



```swift
import SwiftUI

@MainActor
struct ContentView: View {
    @State var viewModel: PostsViewModel = .init()

    var body: some View {
        List {
            ForEach(viewModel.posts, id: \.id) { post in
                Text(post.title)
            }
        }
        .task {
            await viewModel.fetchPosts()
        }
        .navigationTitle("Posts")
    }
}

#Preview {
    NavigationStack {
        ContentView()
    }
}
```



Thank You