
BONJOUR

IMAGINE . . .

A man with brown hair and a beard, wearing a blue and white plaid shirt, is standing in the middle of a crowded street. He is looking over his shoulder towards two women. One woman, on the left, is wearing a red sleeveless top and has long brown hair. The other woman, on the right, is wearing a light blue sleeveless top and also has long brown hair. They appear to be engaged in a conversation with the man.

SWIFTUI

DEVELOPER

UIKIT

SWIFTUI

SWIFTUI

- ▶ One of the first Swift-only frameworks
- ▶ Structs based on Protocols
- ▶ Simple to read and write
- ▶ Elegant Relationship with data
- ▶ Very lightweight views
- ▶ No Auto Layout or Storyboards
- ▶ Live Preview Canvas
- ▶ Fewer bugs

PROTOCOL BASED

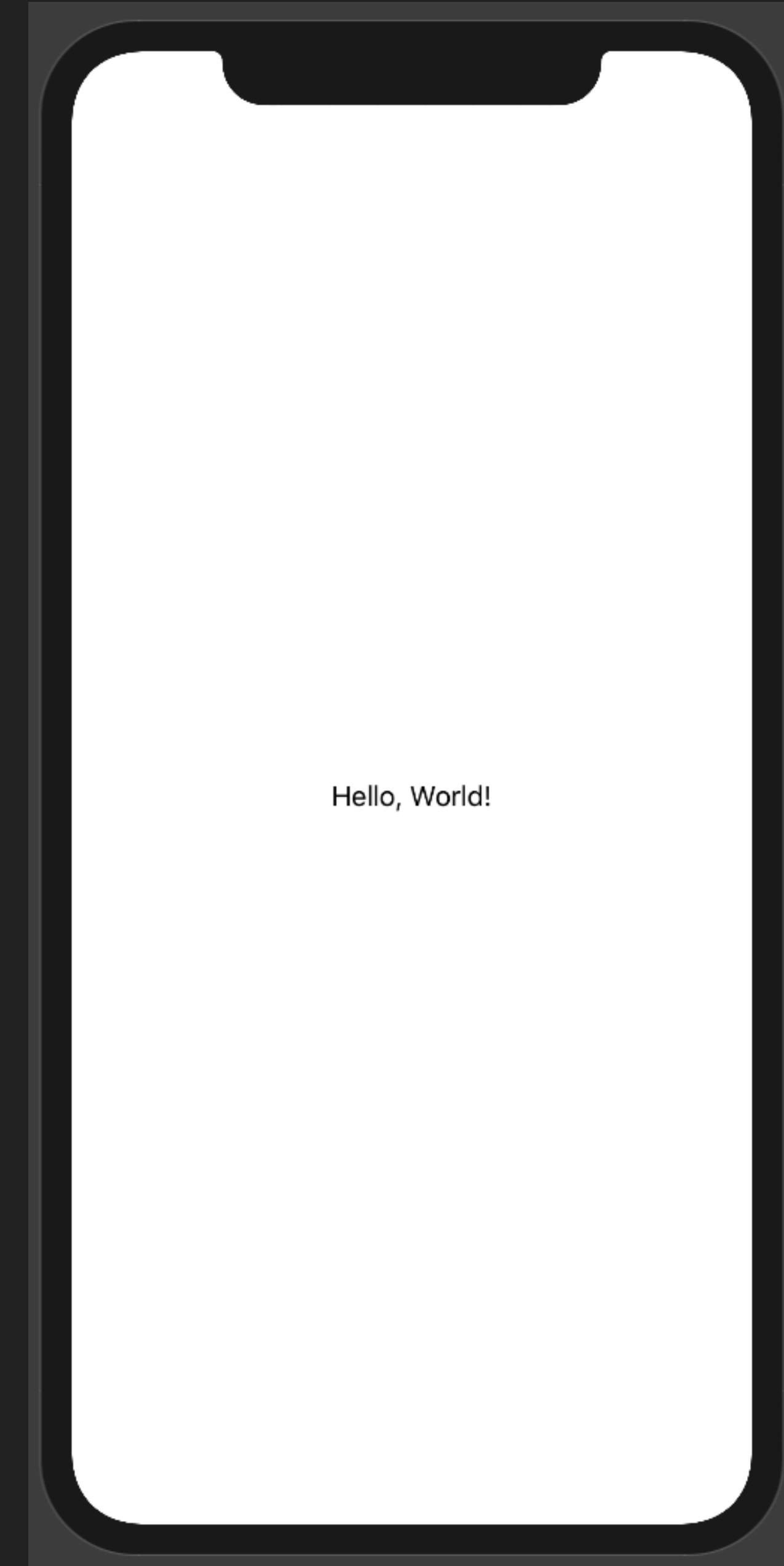
```
public protocol View {  
  
    /// The type of view representing the body of this view.  
    ///  
    /// When you create a custom view, Swift infers this type from your  
    /// implementation of the required `body` property.  
    associatedtype Body : View  
  
    /// Declares the content and behavior of this view.  
    var body: Self.Body { get }  
}
```

PRIMITIVE VIEWS

- ▶ Text
- ▶ Image
- ▶ Color
- ▶ Shape
- ▶ Spacer
- ▶ Divider

SIMPLE TO WRITE & READ

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



ELEGANT RELATIONSHIP WITH DATA

- ▶ Every piece of data has a single source of truth
- ▶ Views are re-drawn based on the state of the data
- ▶ Views are created and destroyed quickly so that the UI always represents the data.

WORKING WITH DATA

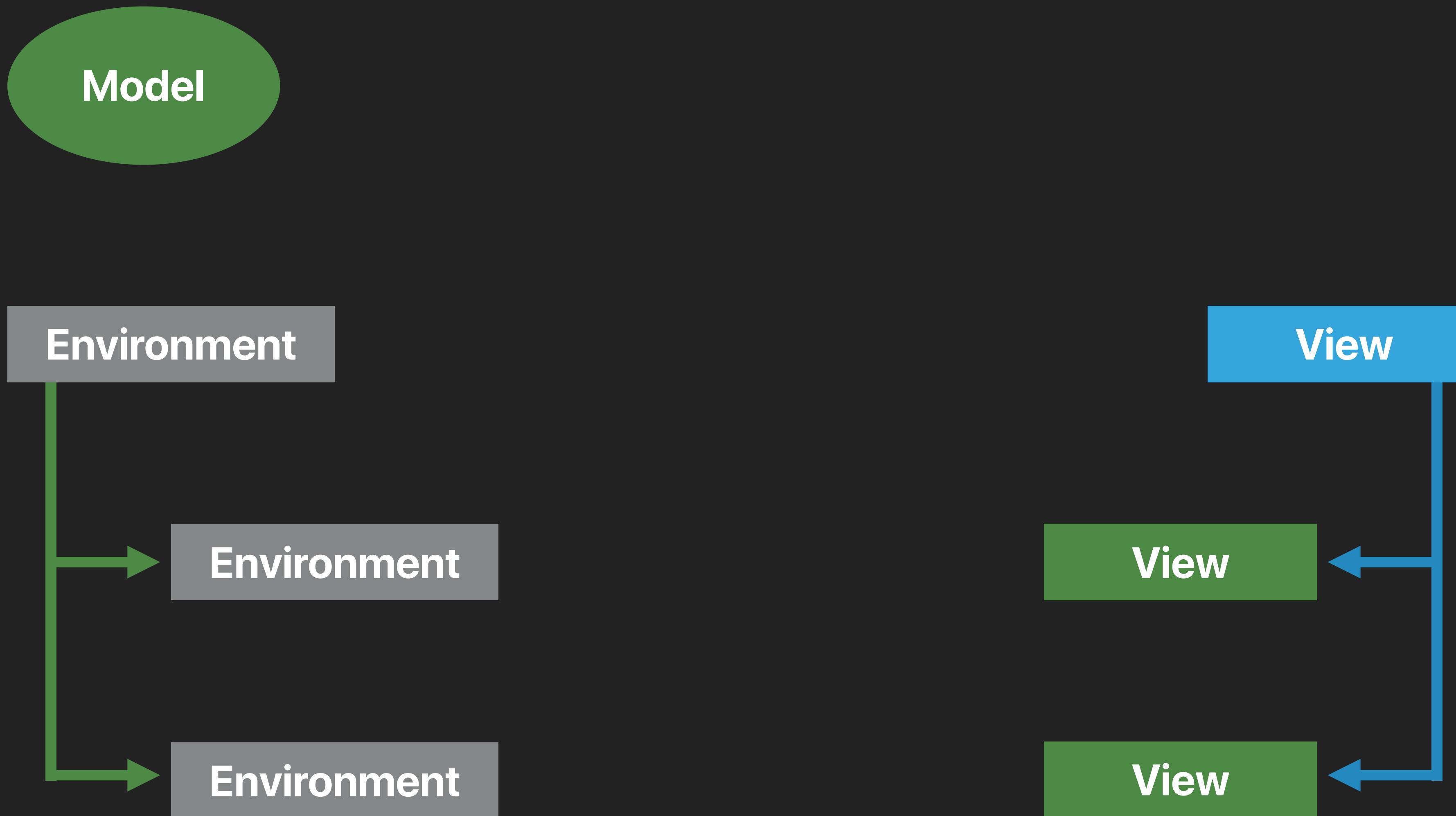
- ▶ `@State` - internal UI state only
- ▶ `@Binding` - used to observe an external `@State` as the single source of truth
- ▶ `@Environment` - environment values applicable to an entire heirarchy
- ▶ `@ObservedObject` - custom classes
- ▶ `@EnvironmentObject` - custom classes injected into the environment

@ENVIRONMENT

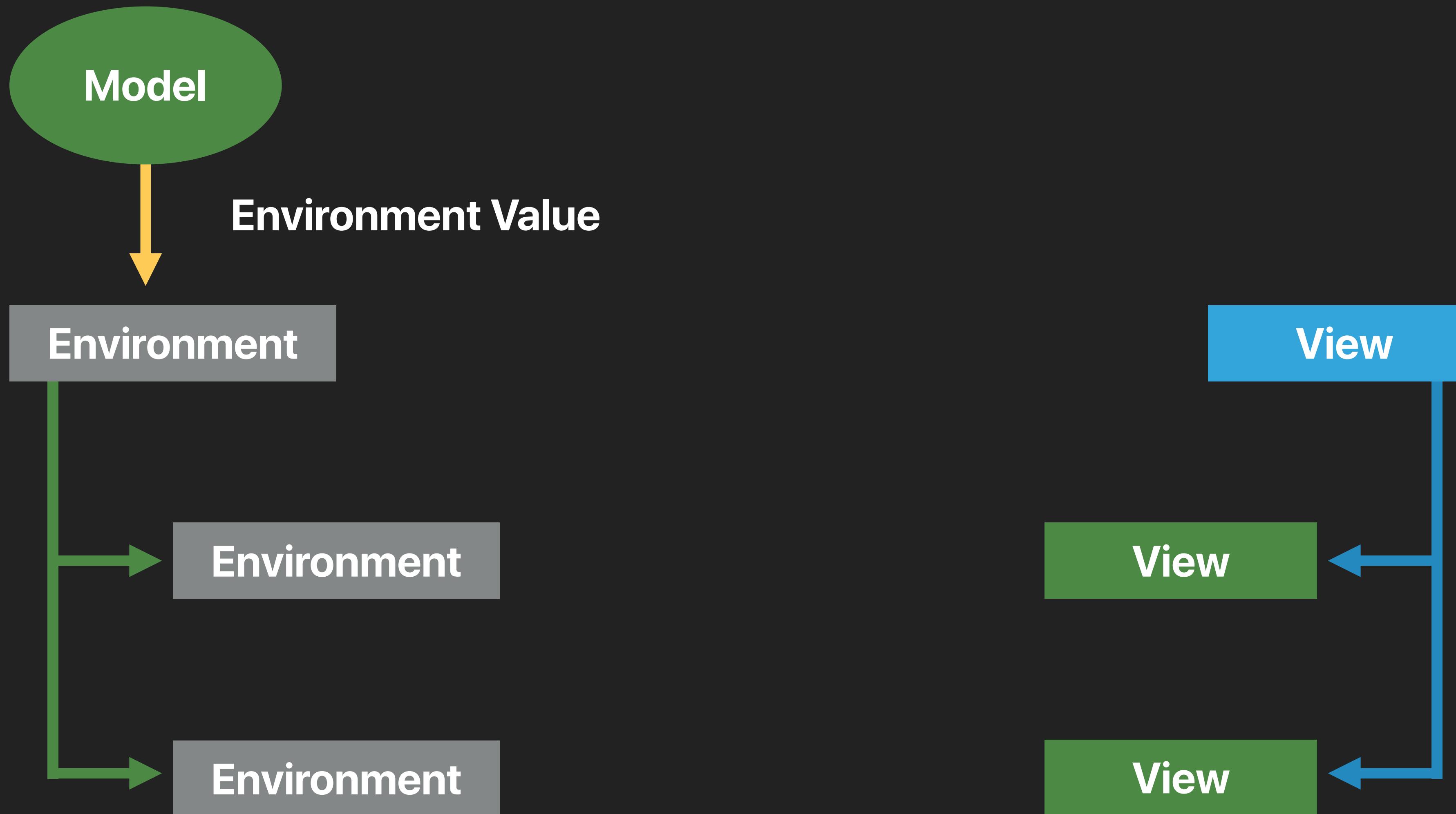
- ▶ Data applicable to an entire hierarchy
- ▶ Convenient for indirect dependence
- ▶ Locale, Color Scheme, Calendar, Font, etc.

```
@Environment(\.font) var font
```

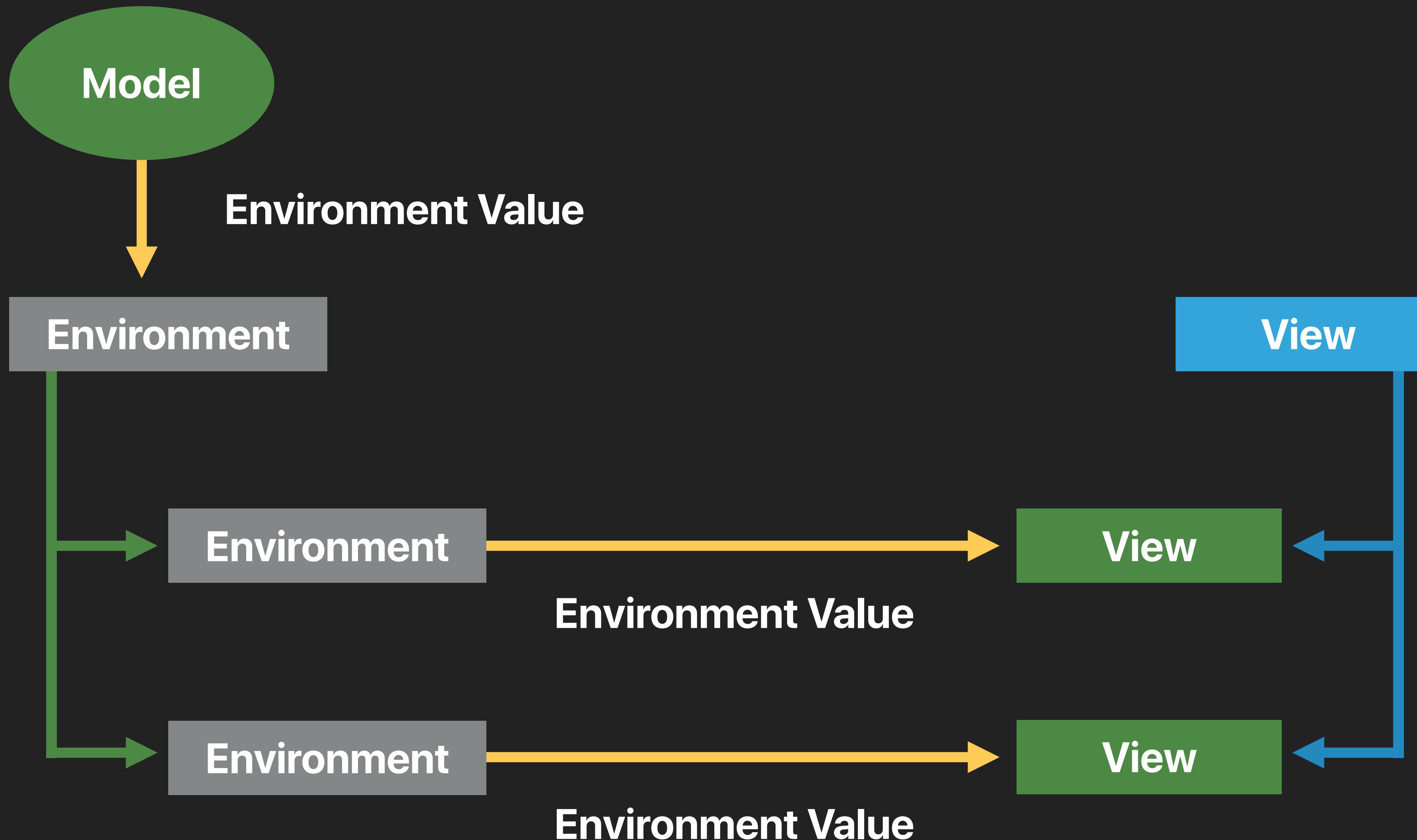
@ENVIRONMENT



@ENVIRONMENT



@ENVIRONMENT



OBSERVABLE OBJECT

- ▶ Custom Class
- ▶ `@Published` properties
- ▶ Changes in published properties are observed by subscribers

```
class NotebookSelectionController: ObservableObject{  
    @Published var selectedIndex:Int?  
}
```

**TWO WAYS TO LISTEN TO
OBSERVABLE OBJECTS**

@OBSERVEDOBJECT

- ▶ Observes changes in an instance of an Observable Object
- ▶ Changes cause body to refresh

```
@ObservedObject var controller = NotebookSelectionController()
```

@ObservedObject

Model

View

View

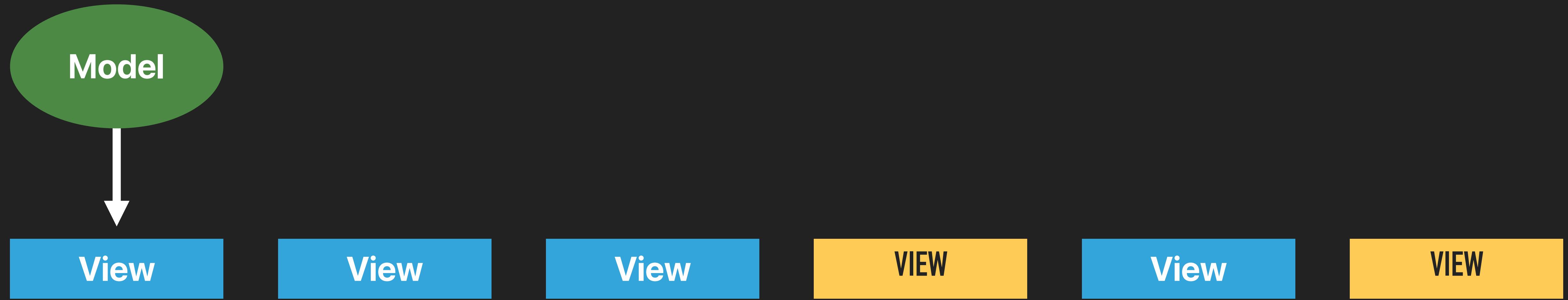
View

VIEW

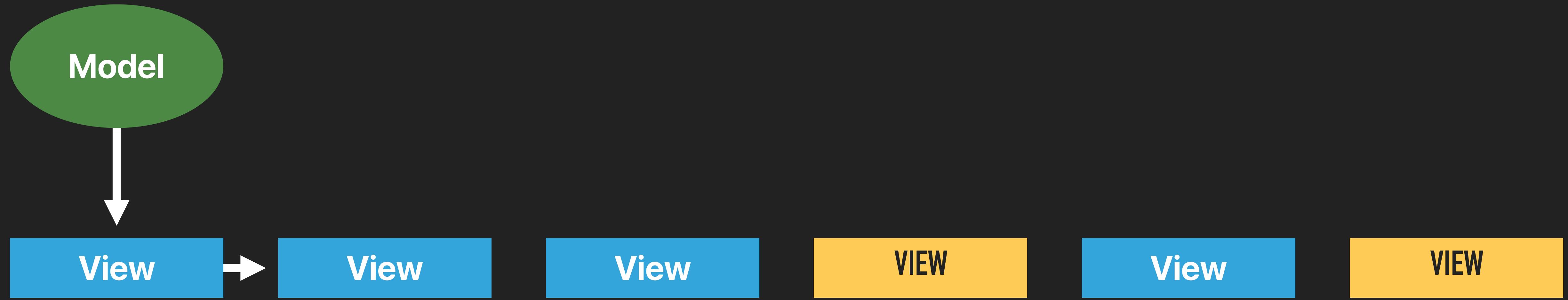
View

VIEW

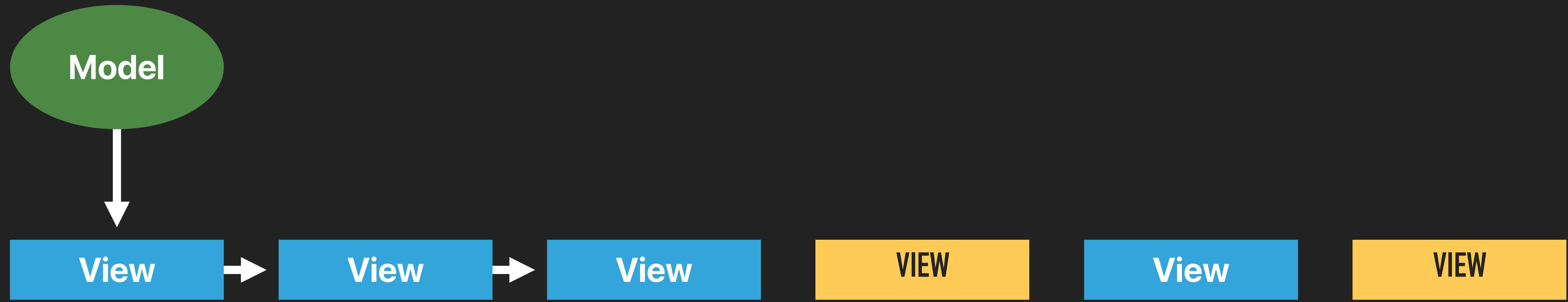
@ObservedObject



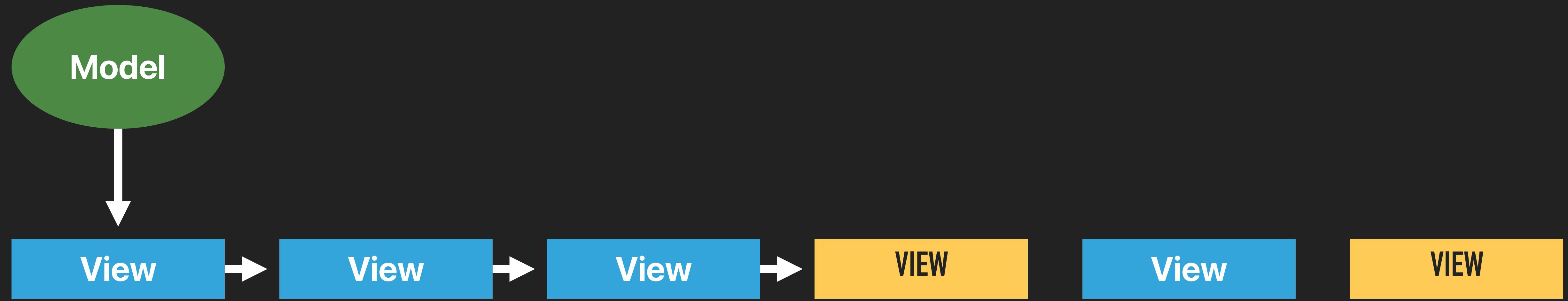
@ObservedObject



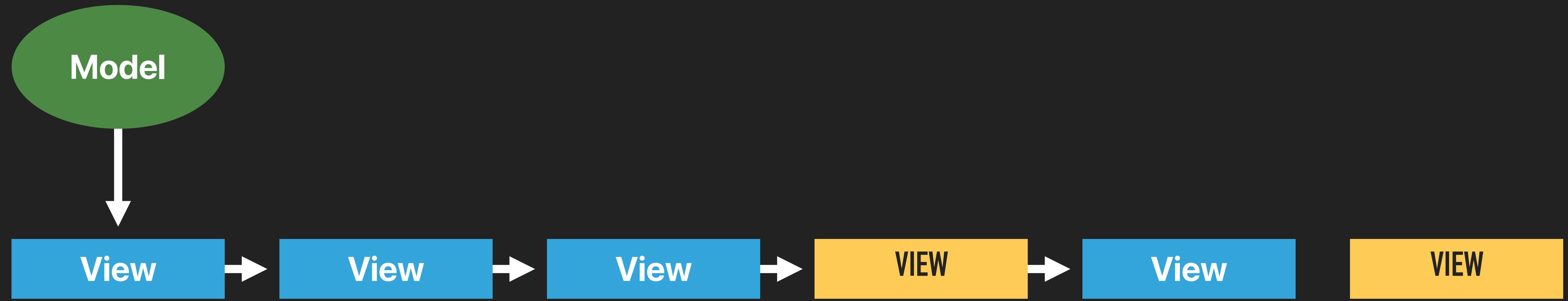
@ObservedObject



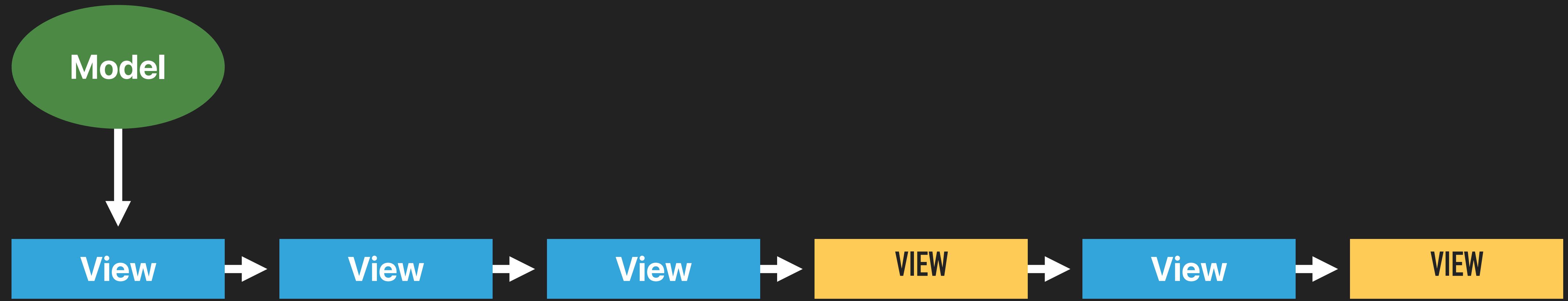
@ObservedObject



@ObservedObject



@ObservedObject



@ENVIRONMENTOBJECT

- ▶ Gives us the ability to inject our custom objects into the environment
- ▶ Very Similar to ObservedObject with the elegance of the environment
- ▶ Promises the existence of objects in the environment

@EnvironmentObject

Model

Environment

View

View

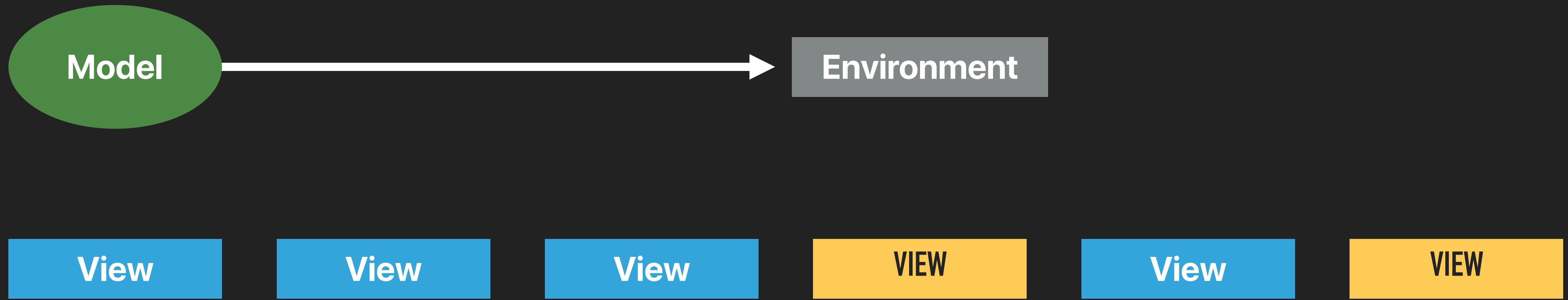
View

VIEW

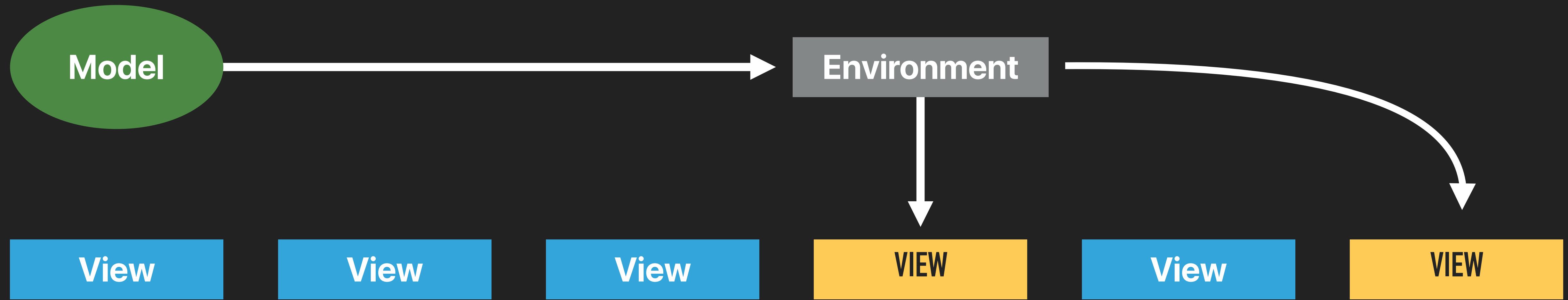
View

VIEW

@EnvironmentObject



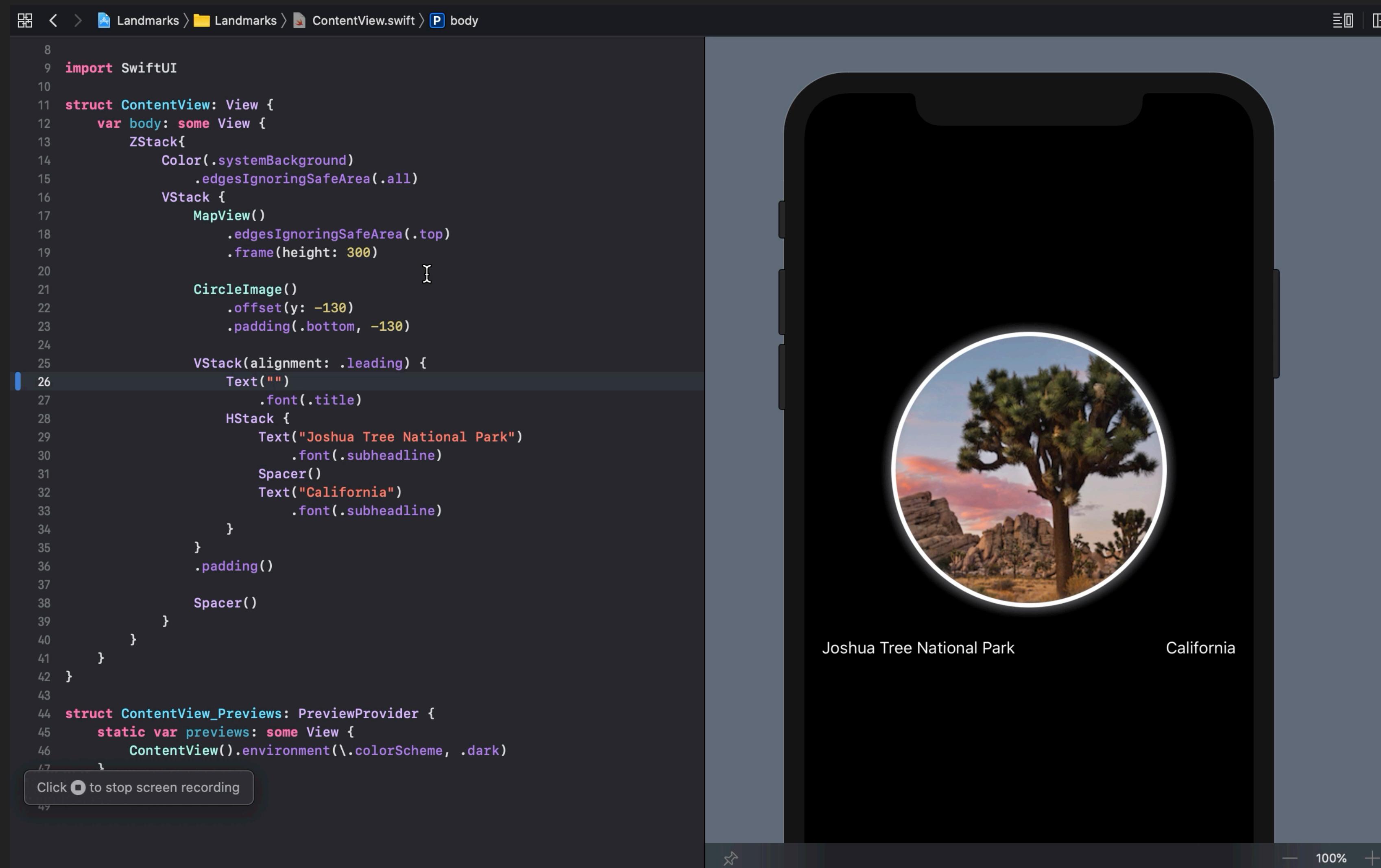
@EnvironmentObject



NO AUTO LAYOUT OR STORYBOARDS

- ▶ No Constraints
- ▶ No Conflicting Constraints
- ▶ No Unsatisfiable Constraints
- ▶ No Storyboard Merge Conflicts

LIVE PREVIEW CANVAS



The screenshot shows the Xcode interface with a live preview canvas. On the left, the code editor displays `ContentView.swift` with Swift code for a SwiftUI view. On the right, a large iPhone X-shaped window shows a live preview of the app's interface. The preview features a circular image of a Joshua tree in a desert landscape at sunset. Below the image, the text "Joshua Tree National Park" and "California" is displayed. A status bar at the bottom of the preview window shows "100%". In the bottom-left corner of the code editor, there is a tooltip that says "Click ⏹ to stop screen recording".

```
8
9 import SwiftUI
10
11 struct ContentView: View {
12     var body: some View {
13         ZStack{
14             Color(.systemBackground)
15                 .edgesIgnoringSafeArea(.all)
16             VStack {
17                 MapView()
18                     .edgesIgnoringSafeArea(.top)
19                     .frame(height: 300)
20
21                 CircleImage()
22                     .offset(y: -130)
23                     .padding(.bottom, -130)
24
25                 VStack(alignment: .leading) {
26                     Text("")
27                         .font(.title)
28                     HStack {
29                         Text("Joshua Tree National Park")
30                             .font(.subheadline)
31                         Spacer()
32                         Text("California")
33                             .font(.subheadline)
34                     }
35                 }
36                 .padding()
37
38                 Spacer()
39             }
40         }
41     }
42 }
43
44 struct ContentView_Previews: PreviewProvider {
45     static var previews: some View {
46         ContentView().environment(\.colorScheme, .dark)
47     }
}
```

Click ⏹ to stop screen recording

ELIMINATES TWO TYPES OF BUGS

- ▶ UI and Model are always in sync
- ▶ No Auto Layout constraints, so no Auto Layout bugs

SWIFTUI



ALL THE THINGS!

WHAT'S MISSING

- ▶ Collection Views
- ▶ Page Control
- ▶ Toolbar
- ▶ Search Bars
- ▶ Progress Bars
- ▶ TextViews
- ▶ ImagePicker
- ▶ AVPlayer





COMBINING SWIFTUI & UIKIT

SWIFTUI + UIKIT

REPRESENTABLE PROTOCOL

REPRESENTABLE PROTOCOL

UIView

UIViewController

REPRESENTABLE PROTOCOL

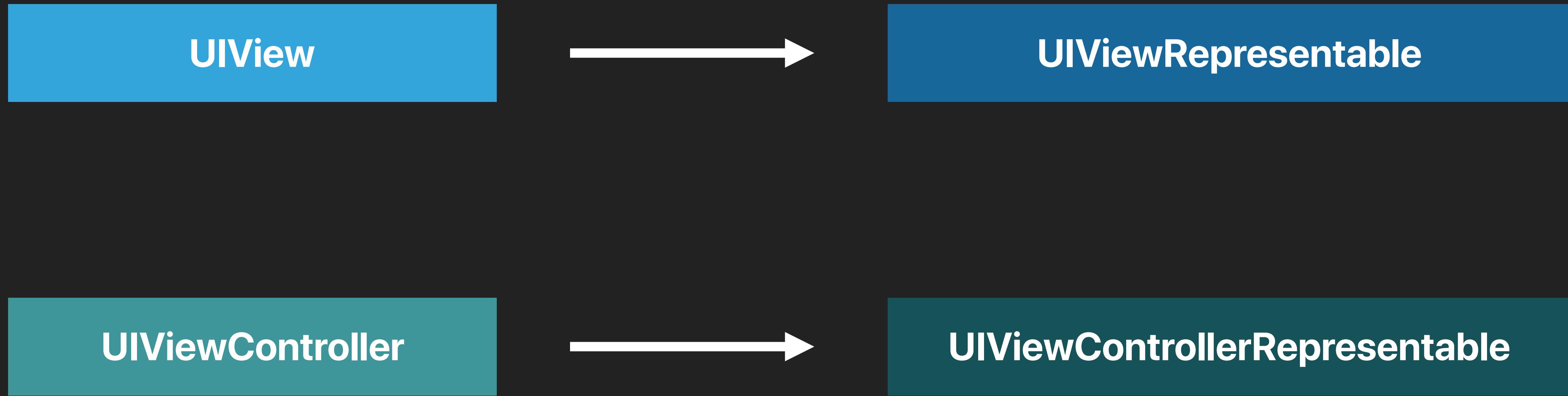
UIView

UIViewRepresentable

UIViewController

UIViewControllerRepresentable

REPRESENTABLE PROTOCOL



REPRESENTABLE PROTOCOL

Make

Update

REPRESENTABLE PROTOCOL

Make Coordinator

Make

Update

REPRESENTABLE PROTOCOL

Make Coordinator

Make

Update

Dismantle

COORDINATOR

REPRESENTABLE PROTOCOL

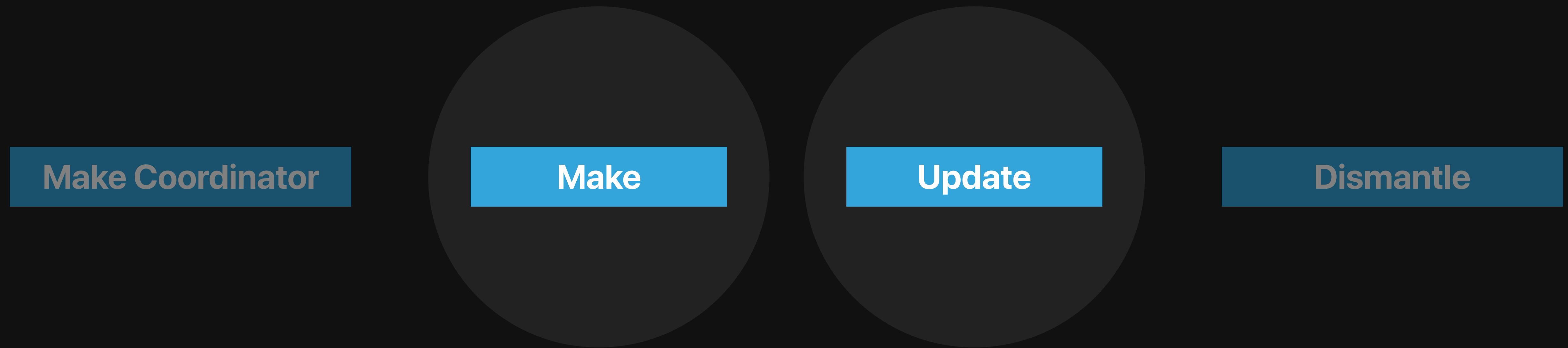
Make Coordinator

Make

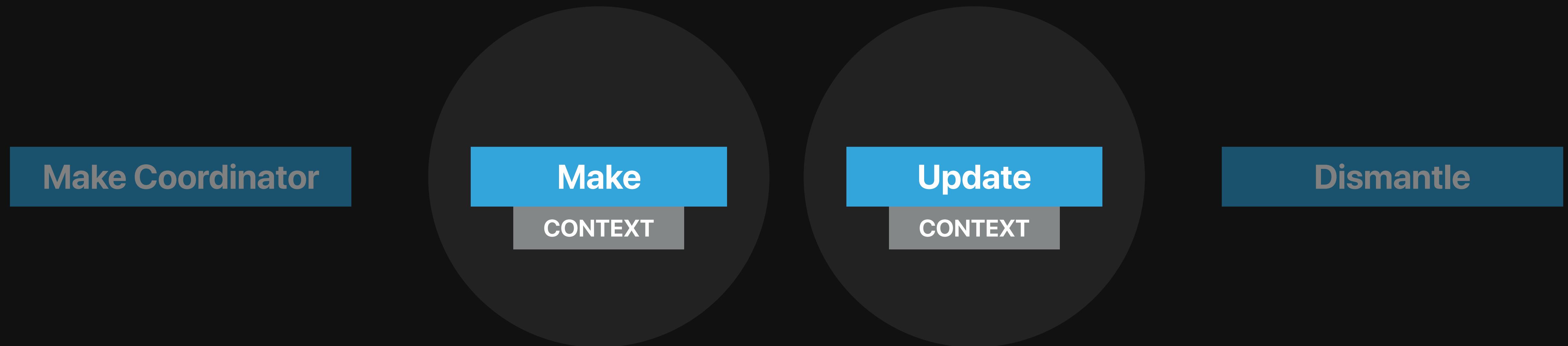
Update

Dismantle

REPRESENTABLE PROTOCOL



REPRESENTABLE PROTOCOL



CONTEXT

CONTEXT

- ▶ Environment
- ▶ Transaction
- ▶ Coordinator

REPRESENTABLE PROTOCOL

Make Coordinator

Make

CONTEXT

Update

CONTEXT

Dismantle



REPRESENTABLE PROTOCOL

Make Coordinator

Make

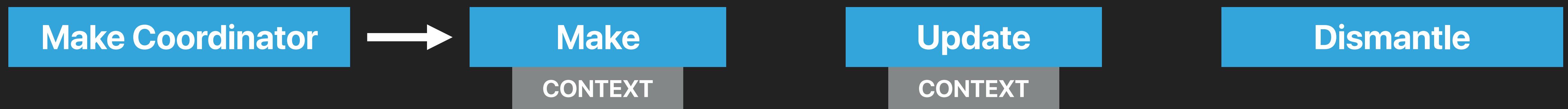
CONTEXT

Update

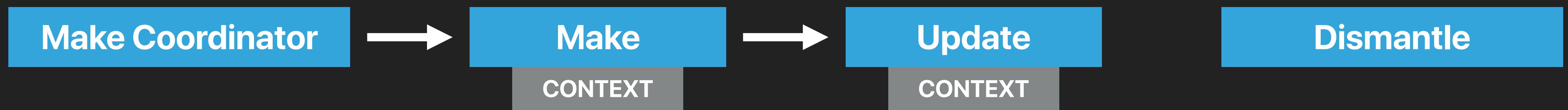
CONTEXT

Dismantle

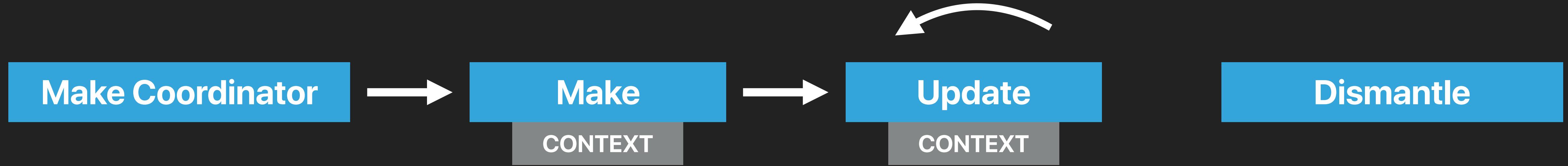
REPRESENTABLE PROTOCOL



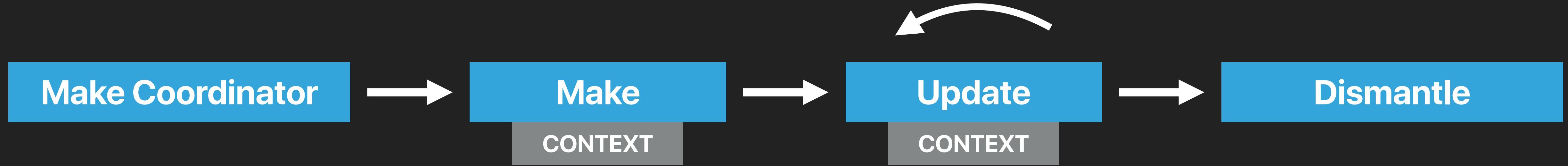
REPRESENTABLE PROTOCOL



REPRESENTABLE PROTOCOL



REPRESENTABLE PROTOCOL



INJECTING DATA

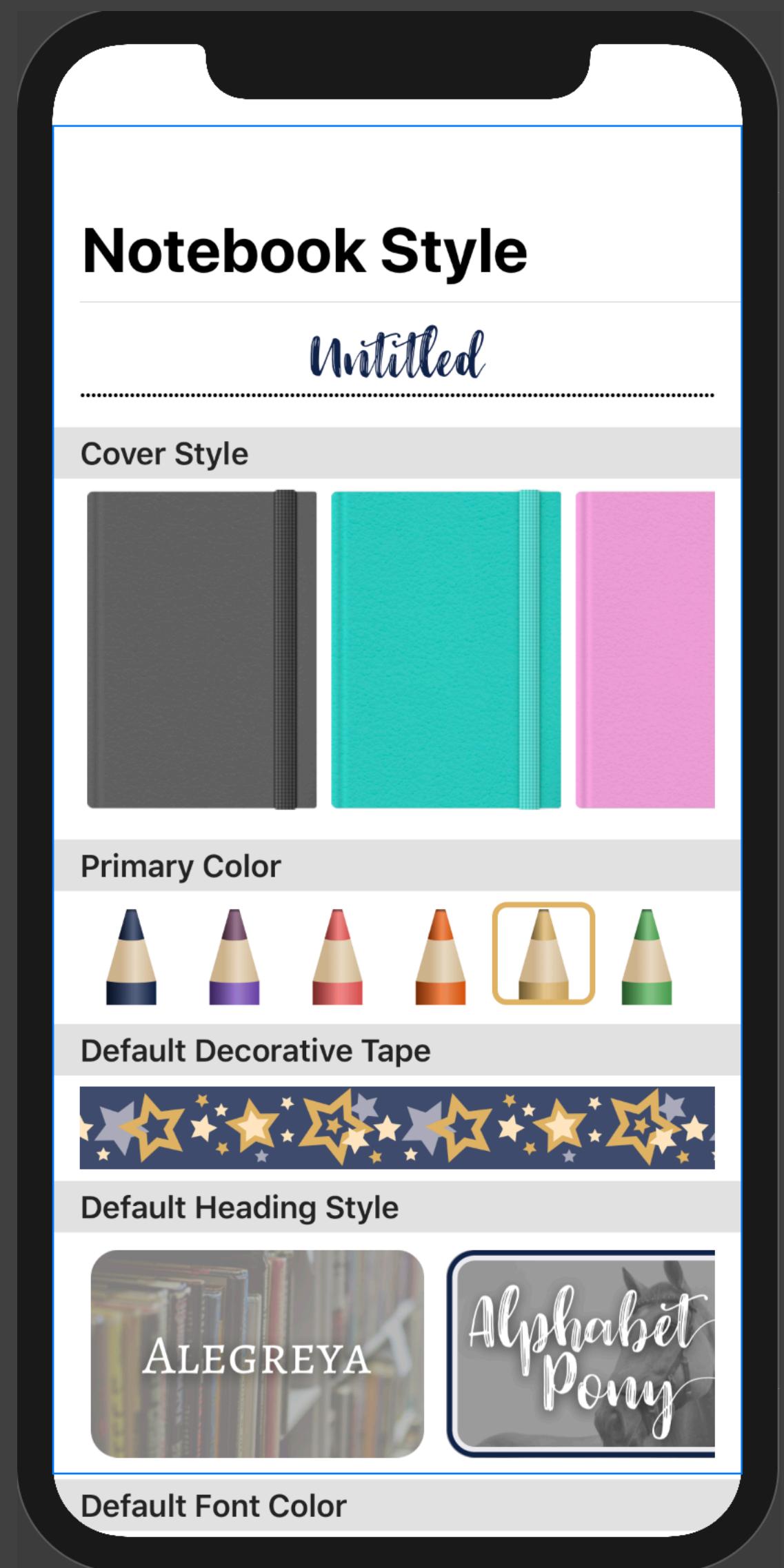
INJECTING DATA

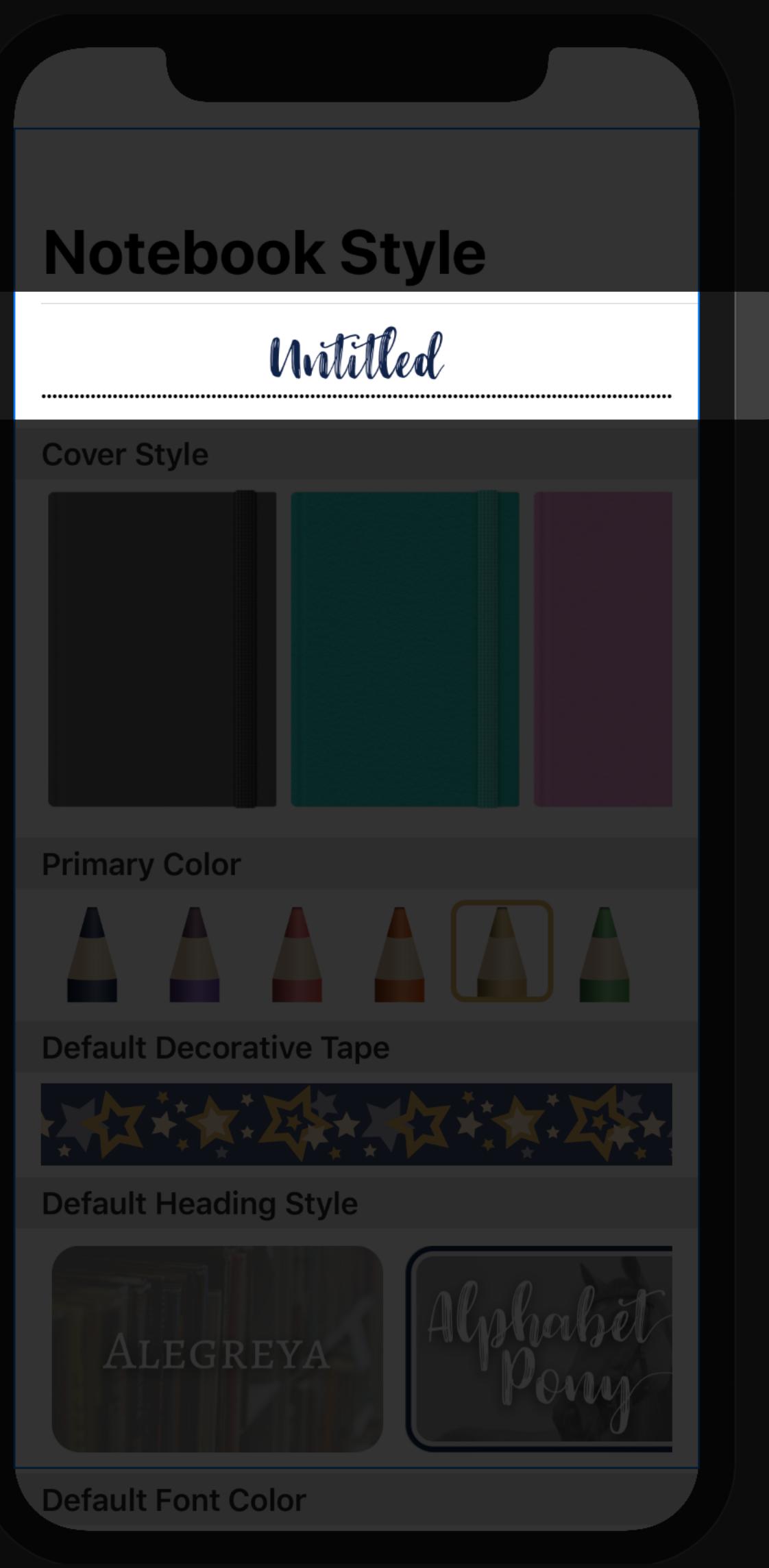
- ▶ `@EnvironmentObjects`
- ▶ Passing data with Initializers

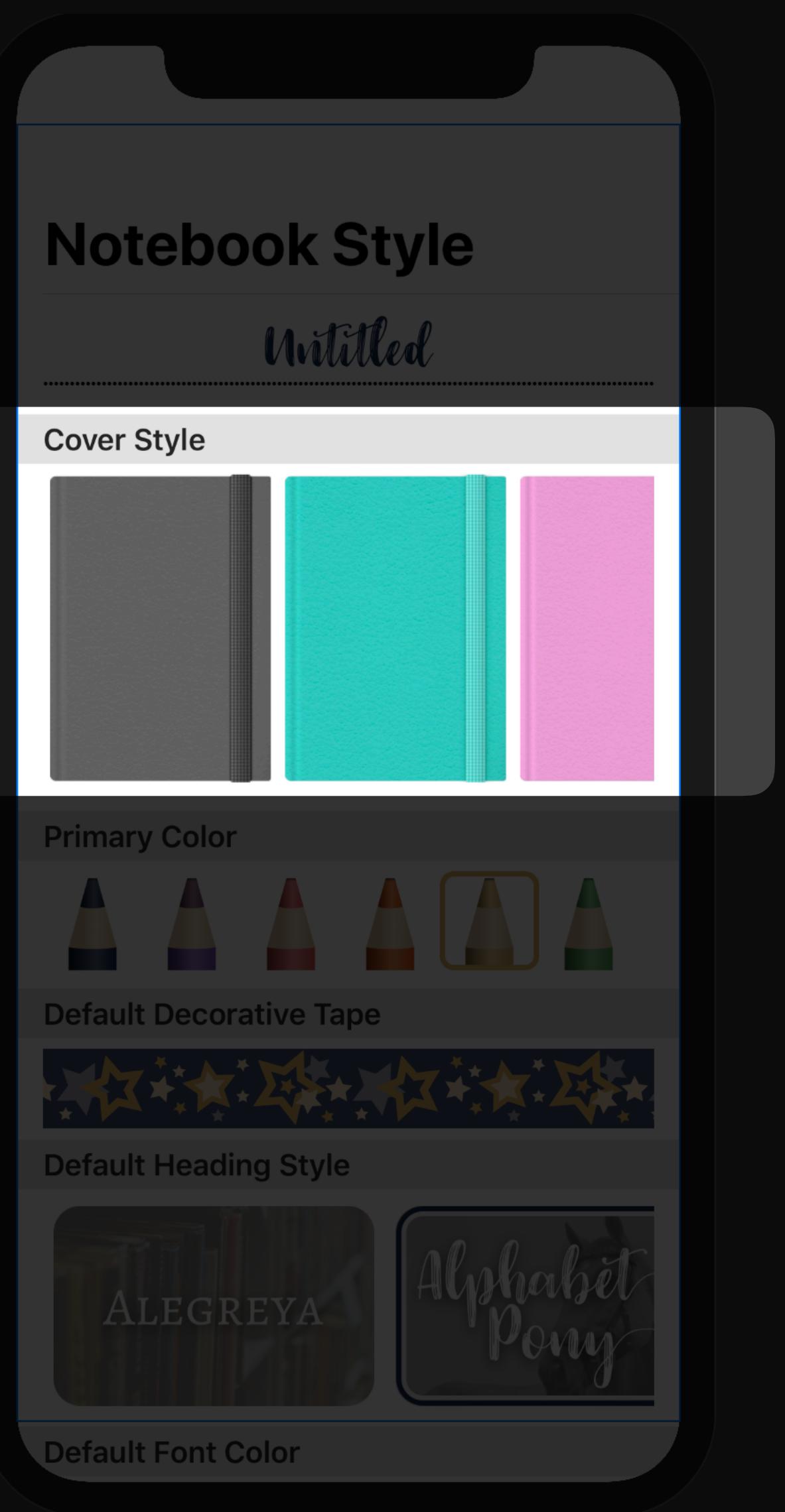
INJECTING DATA

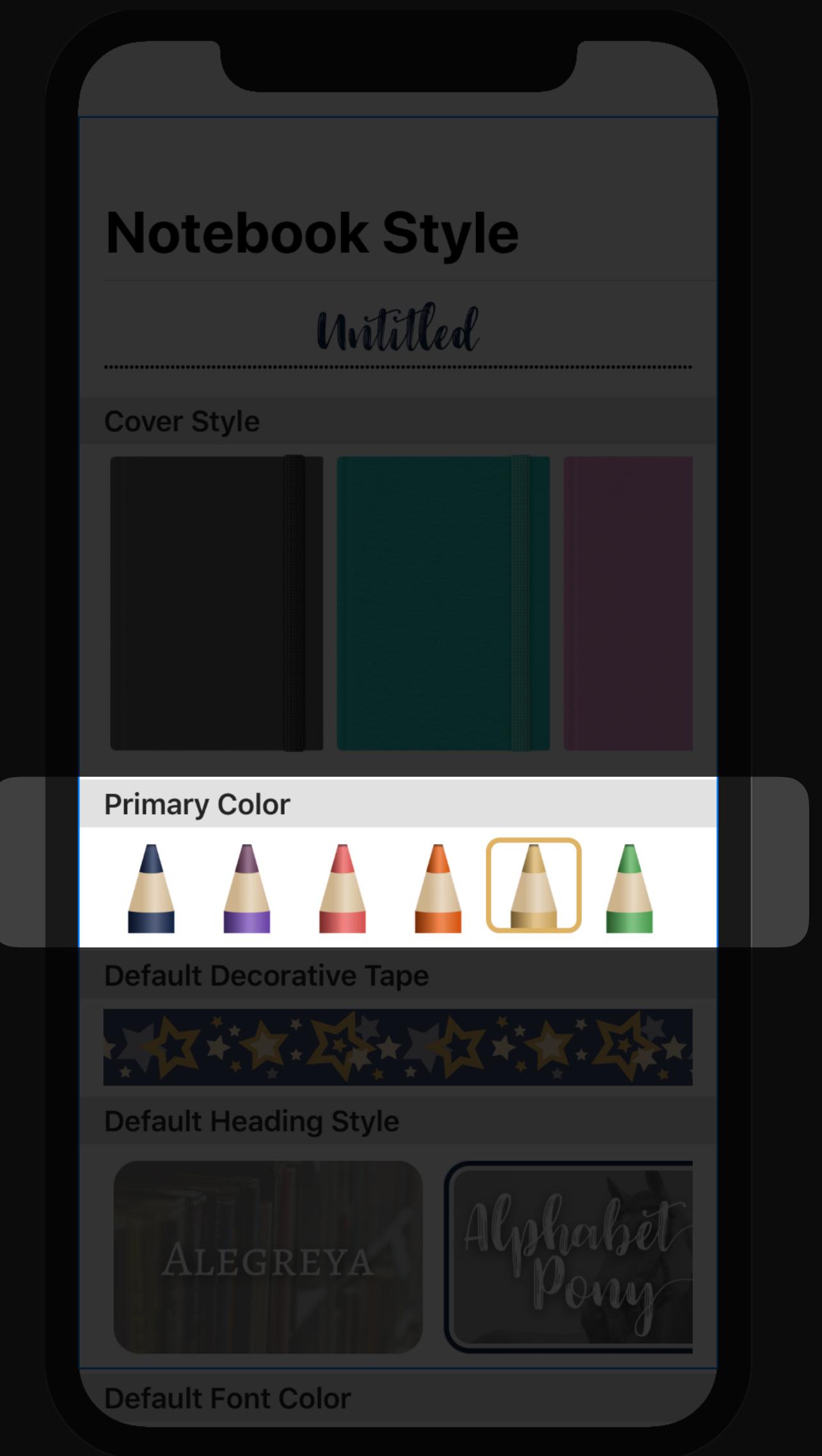
- ▶ `@EnvironmentObject`
- ▶ Pass data with explicitly using initializers

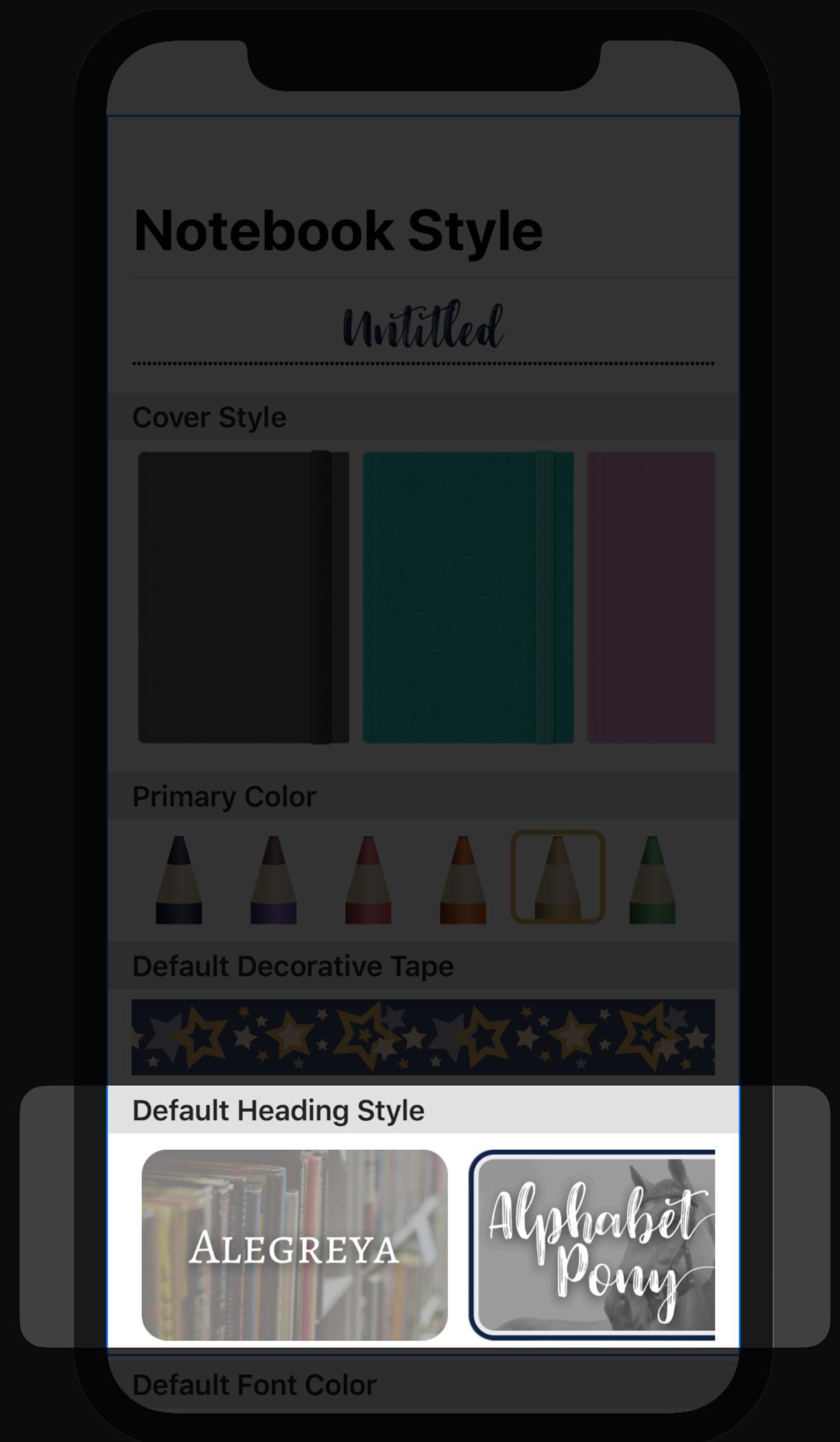
AUTOMATIC UI UPDATES

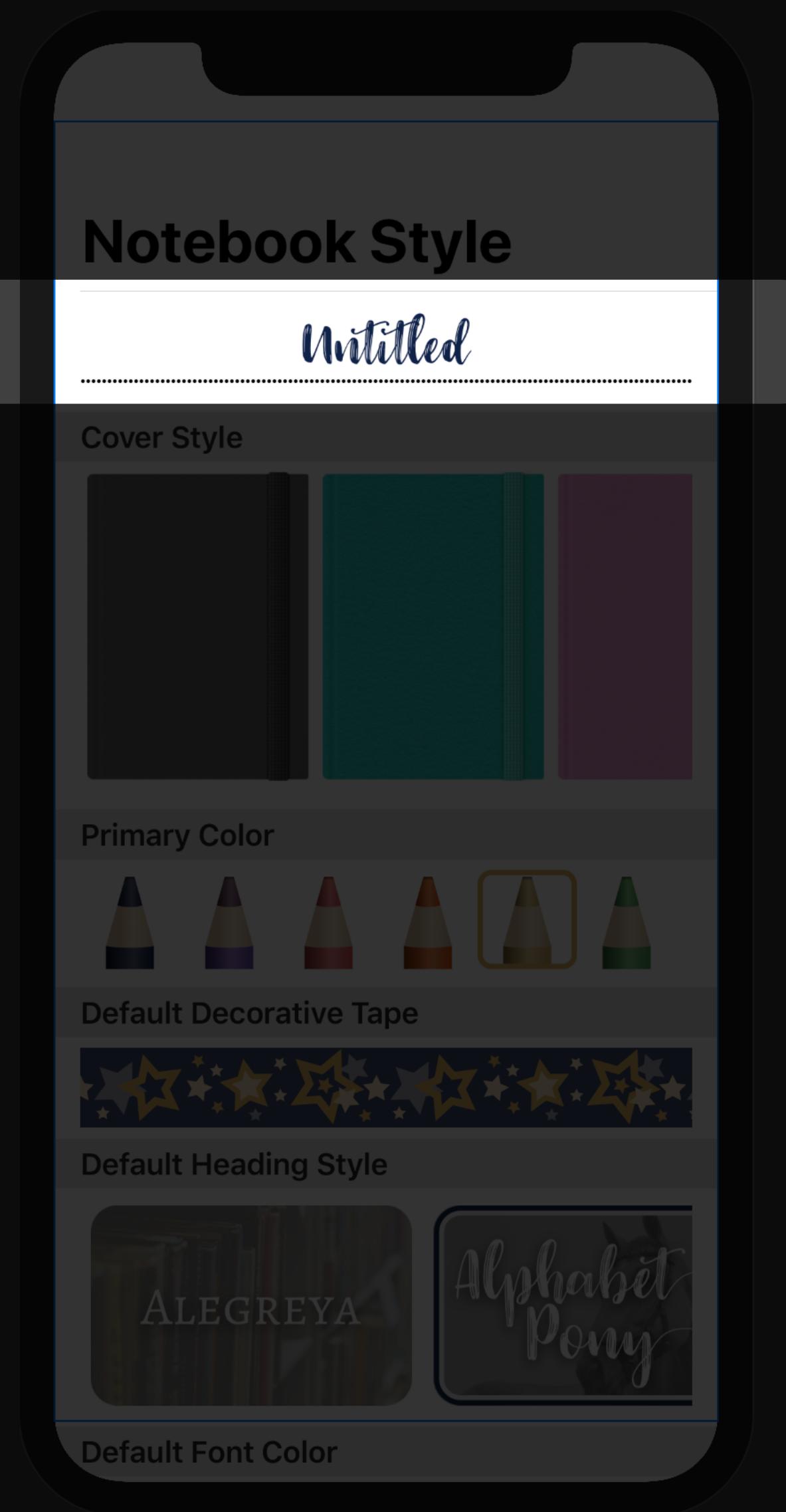












Avoid using UIKit objects in places
where they will be created and
destroyed frequently.

UIKIT + SWIFTUI

IN CODE

UIHostingController

- ▶ Subclass of UIViewController
- ▶ rootView property represents a SwiftUI view

```
let hostingController = UIHostingController(rootView: MyView())
```

STORYBOARD



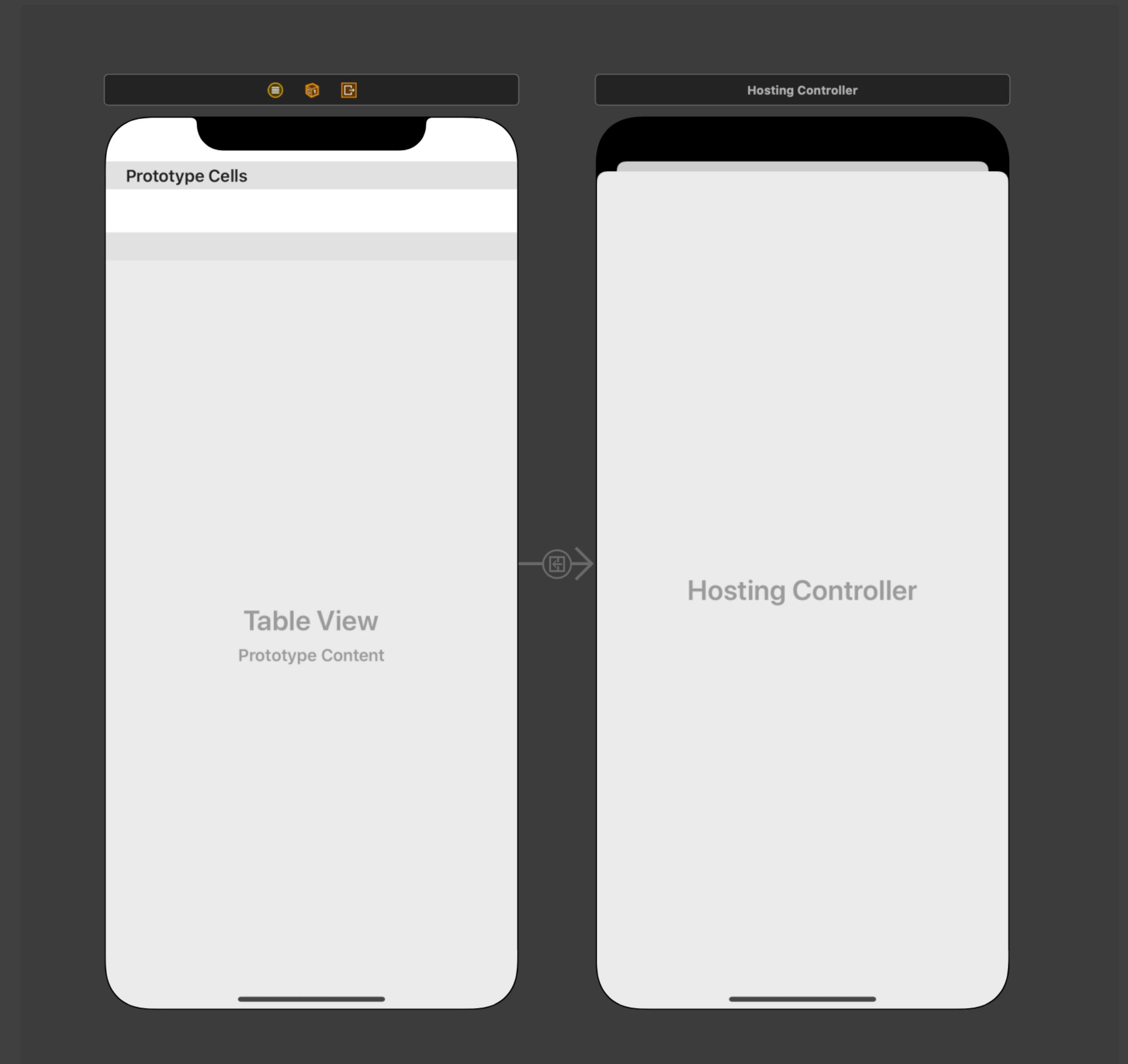
Hosting View Controller

Hosting View Controller

`UIHostingController`: A `UIViewController` that hosts a SwiftUI view hierarchy.

Customize the SwiftUI view hierarchy contents by declaring a custom subclass of `UIHostingController` that programmatically assigns its root view.

You can optionally define a Segue Action on the source view controller that programmatically returns a destination Hosting Controller initialized with the desired root view.



SEGUE

```
@IBSegueAction func showDetail(_ coder: NSCoder) -> UIViewController?{
    let view = MyView()
    return UIHostingController(rootView: view.foregroundColor(.blue))
}
```

UIKIT CELLS

CONVERT SWIFTUI VIEW TO UIKIT VIEW

```
extension NotebookCoverCell{
    func uiview() -> UIView{
        guard let view = UIHostingController(rootView: self).view else {return ErrorView()}
        view.backgroundColor = .clear
        return view
    }
}
```

CONVERT SWIFTUI VIEW TO UIKIT VIEW

```
func setup(){
    //Grab a reference to the local SwiftUI Representable View
    let view = innerCell.uiview()

    //Remove All Subviews
    self.cleanUpCell()

    // Add Represtentable View as Subview to Cell ContentView
    self.contentView.addSubview(view)

    //Add Contraints to top, bottom, leading, trailing
    view.leadingAnchor.constraint(equalTo: contentView.leadingAnchor).isActive = true
    view.trailingAnchor.constraint(equalTo: contentView.trailingAnchor).isActive = true
    view.topAnchor.constraint(equalTo: contentView.topAnchor).isActive = true
    view.bottomAnchor.constraint(equalTo: contentView.bottomAnchor).isActive = true
    view.translatesAutoresizingMaskIntoConstraints = false
}
```

TRY NEW THINGS

**WHERE DOES THAT
LEAVE US?**

MERCI