BONJOUR

IMAGINE...


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

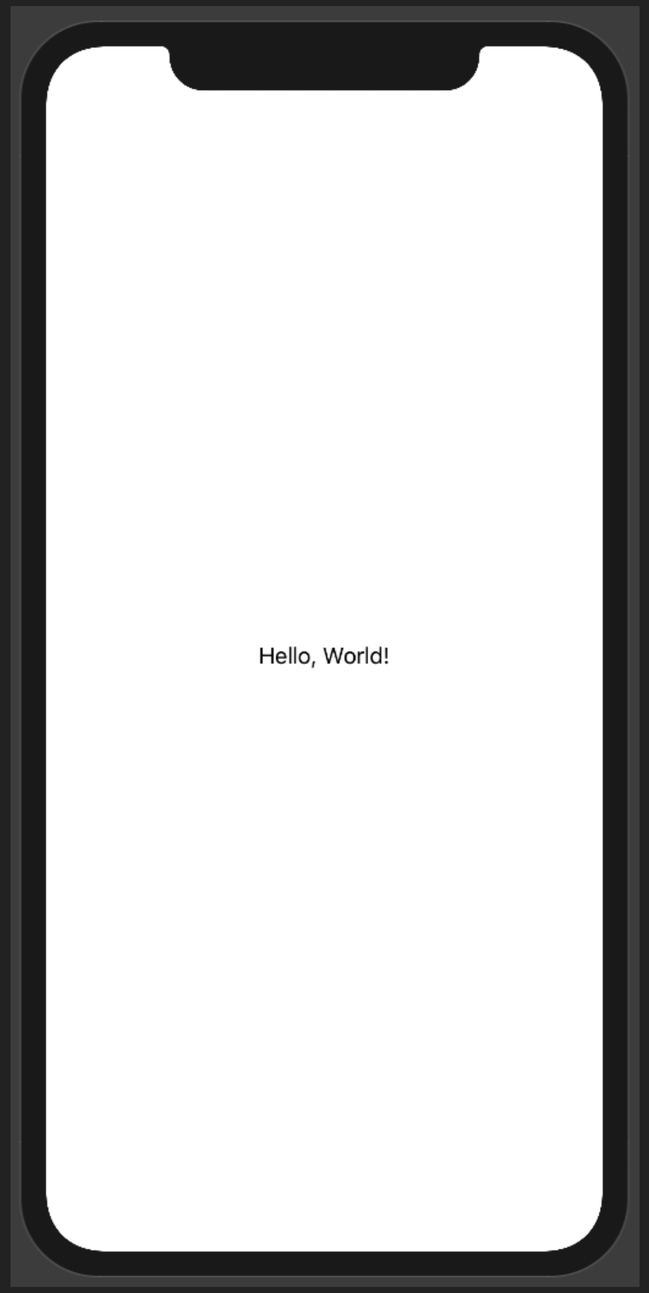
```
public protocol View {

    /// 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")
    }
}
```


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

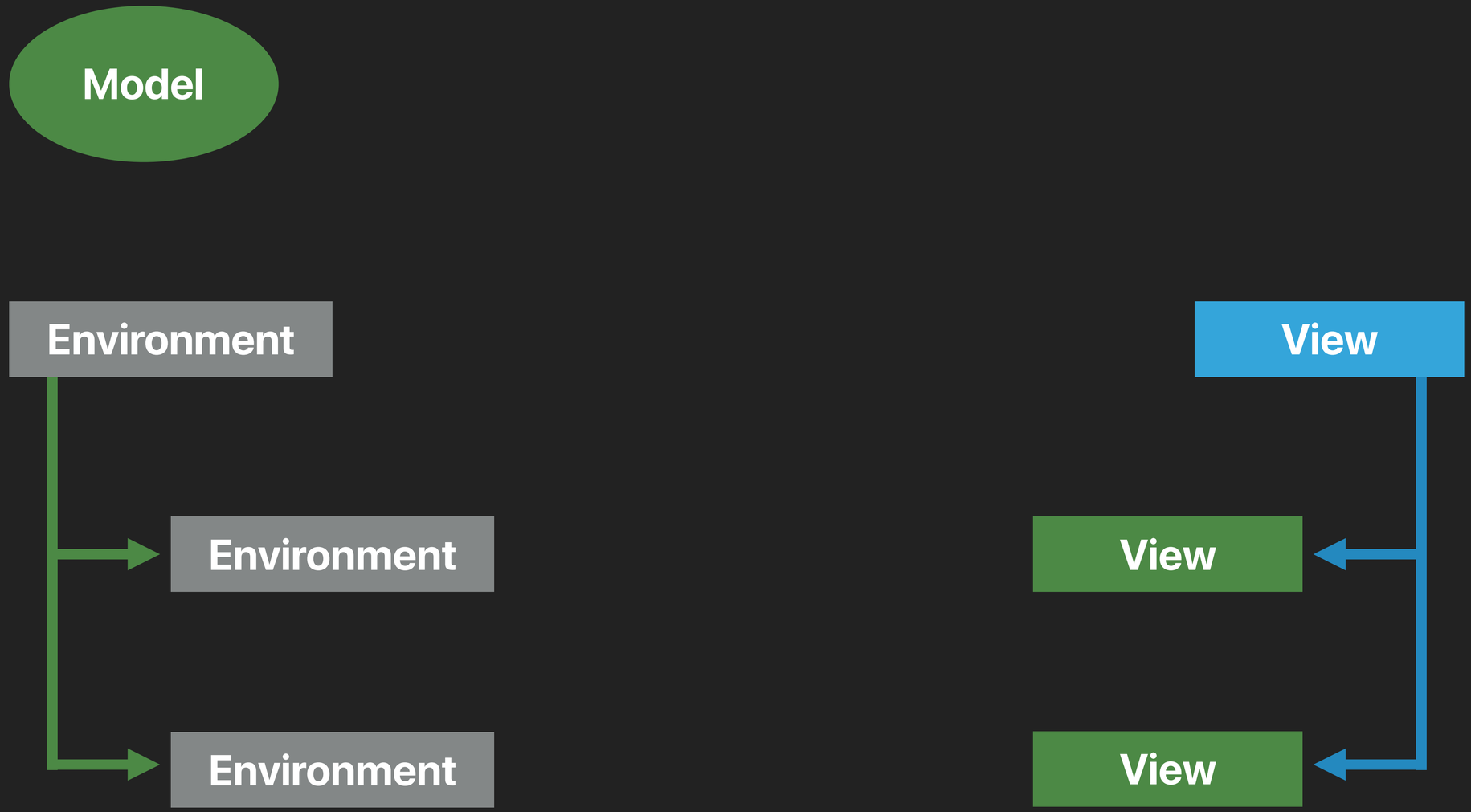
- @State - internal UI state only
- @Binding - used to observe an external @State as the single source of truth
- @Environment - environment values applicable to an entire heirary
- @ObservedObject - custom classes
- @EnvironmentObject - custom classes injected into the environment

@ENVIRONMENT

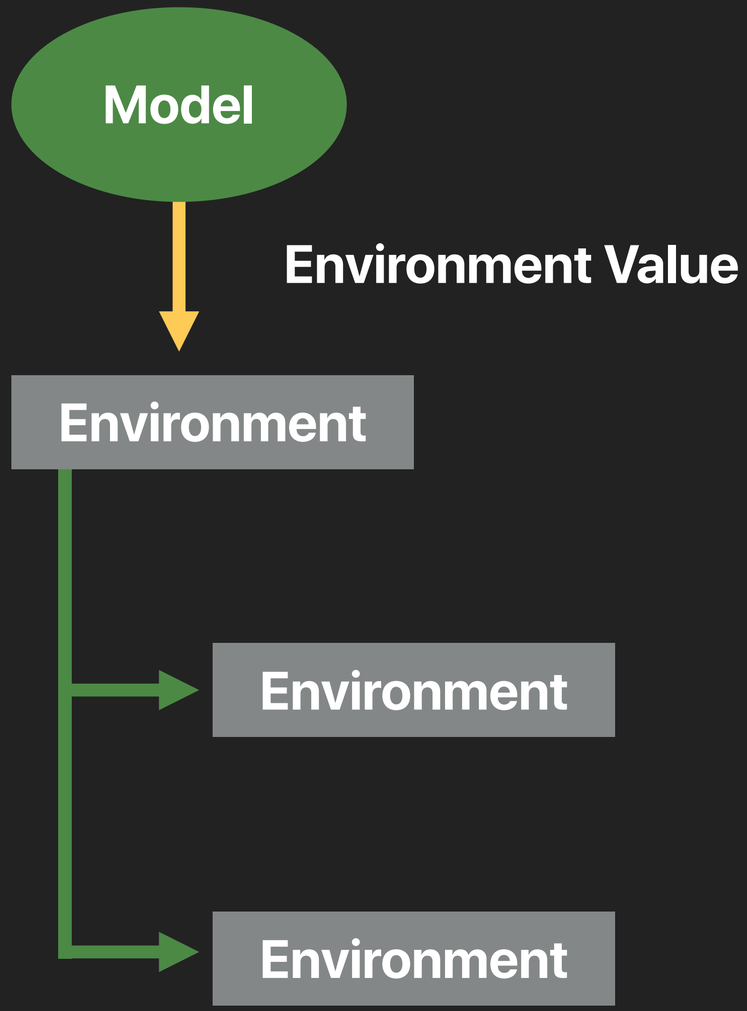
- Data applicable to an entire hierarchy
- Convenient for indirect dependence
- Locale, Color Scheme, Calendar, Font, etc.

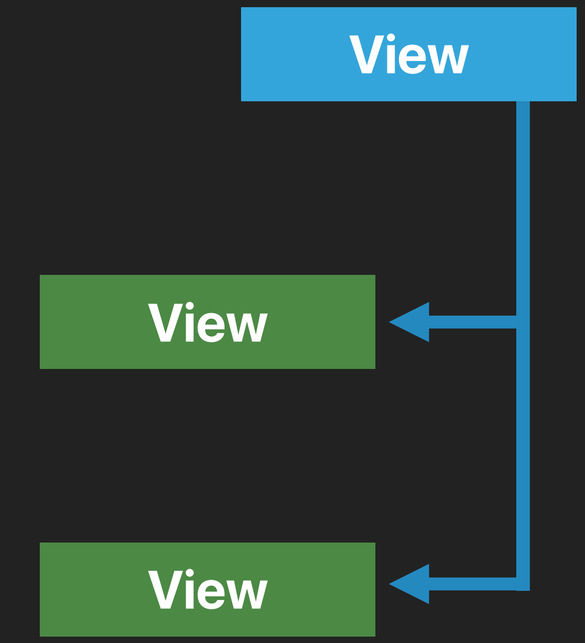
```
@Environment(\.font) var font
```

@ENVIRONMENT

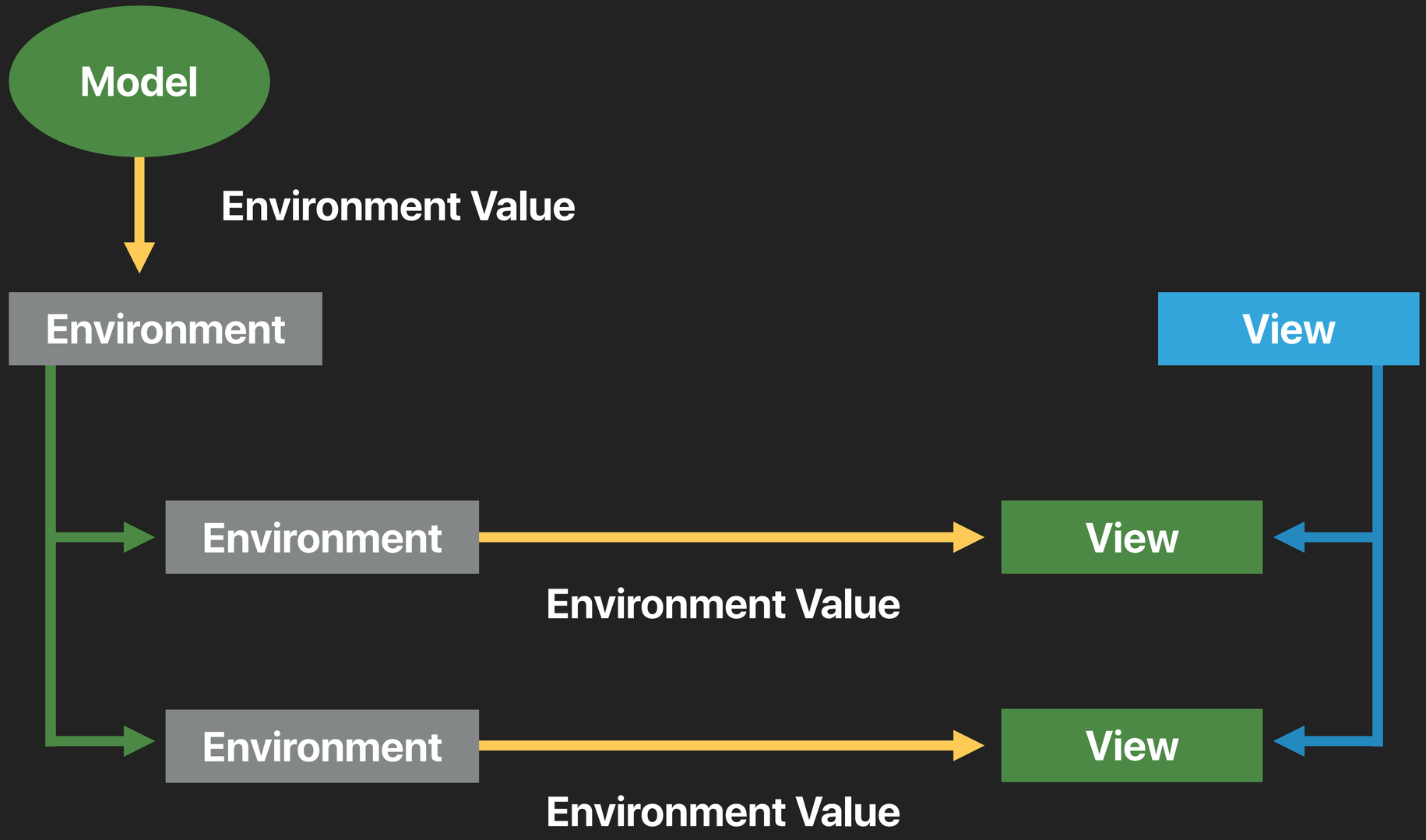


@ENVIRONMENT





@ENVIRONMENT



OBSERVABLE OBJECT

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

```
class NotebookSelectionController:ObservableObject{

    @Published var selectedIndex:Int?

}
```

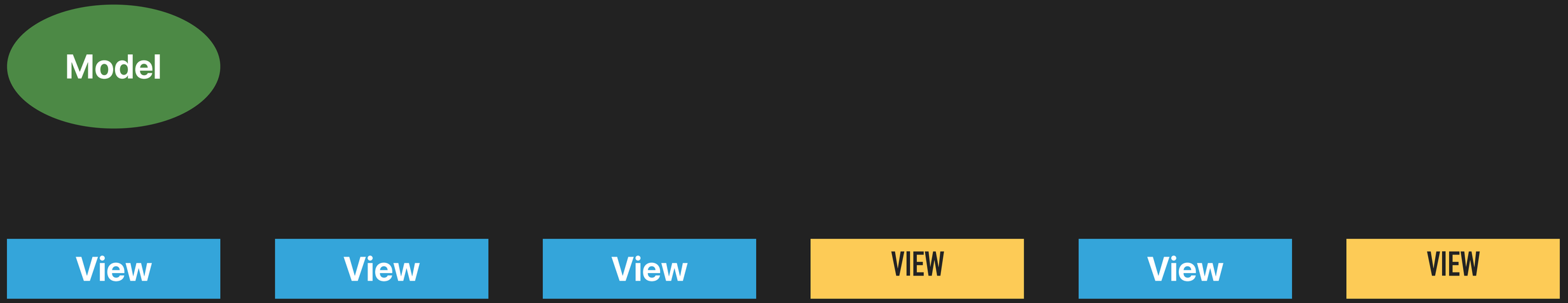
TWO WAYS TO LISTEN TO OBSERVABLE OBJECTS

@OBSERVEDOBJECT

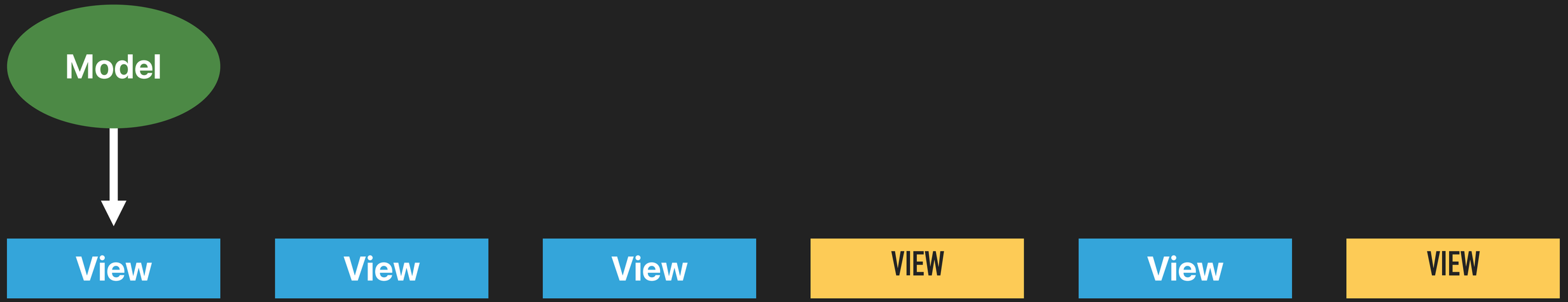
- Observes changes in an instance of an Observable Object
- Changes cause body to refresh

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

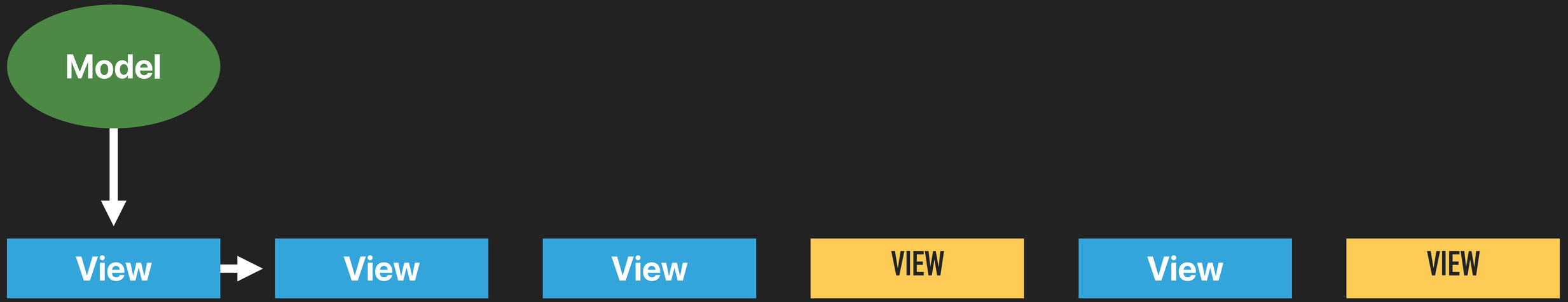
@ObservedObject



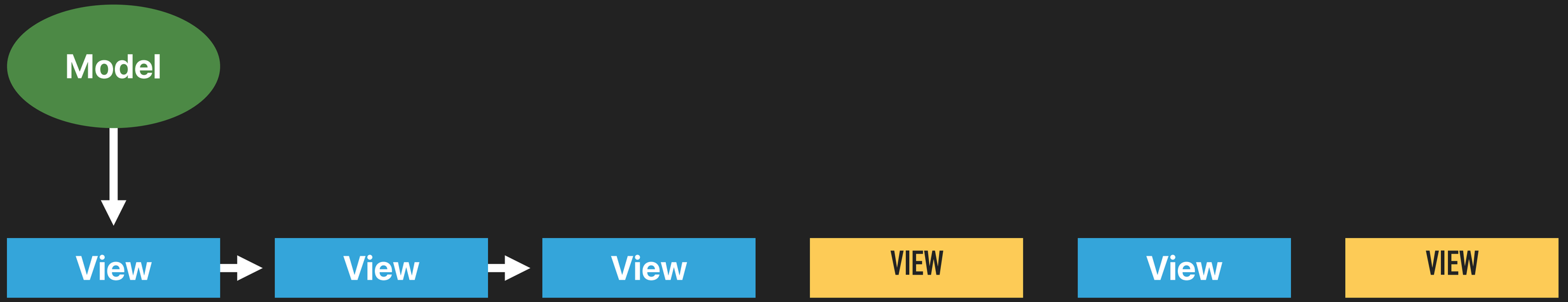
@ObservedObject



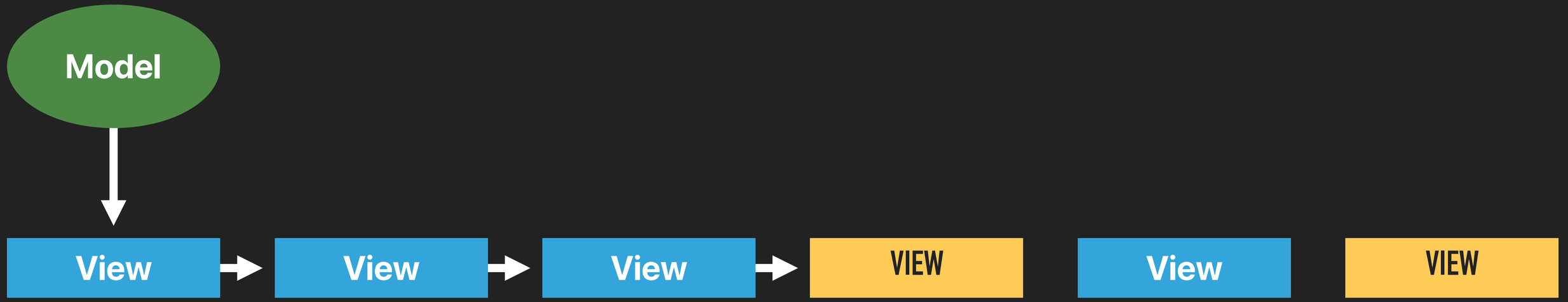
@ObservedObject



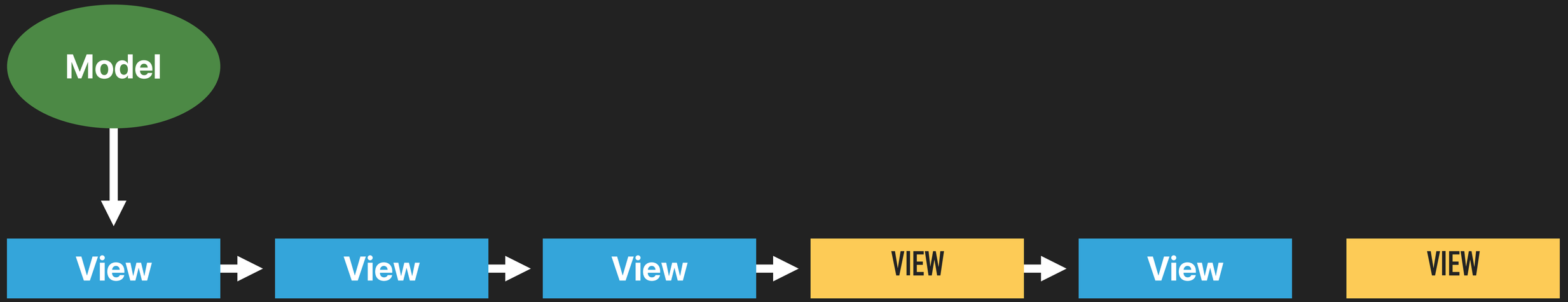
@ObservedObject



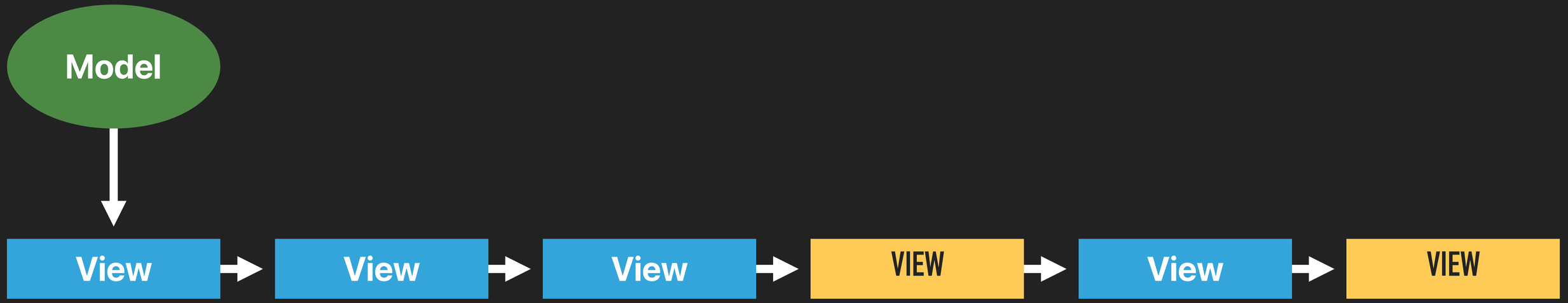
@ObservedObject



@ObservedObject



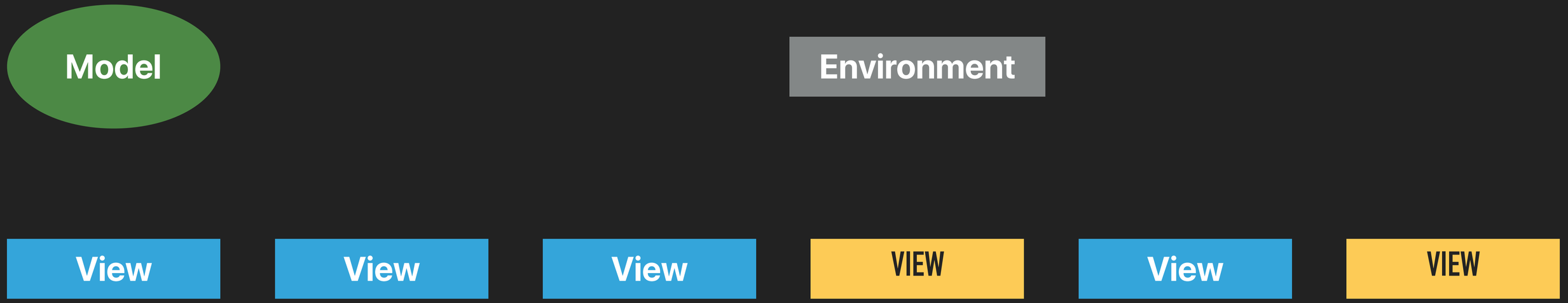
@ObservedObject



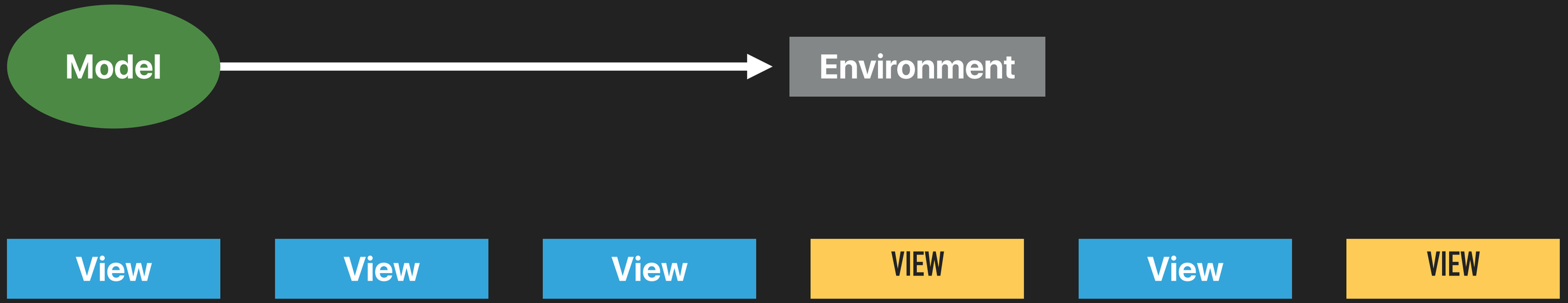
@ENVIRONMENTOBJECT

- 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

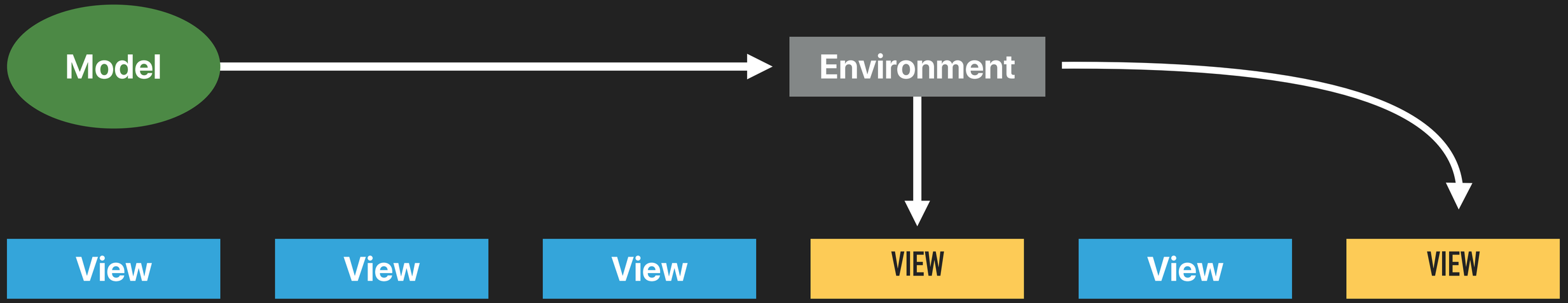
@EnvironmentObject



@EnvironmentObject



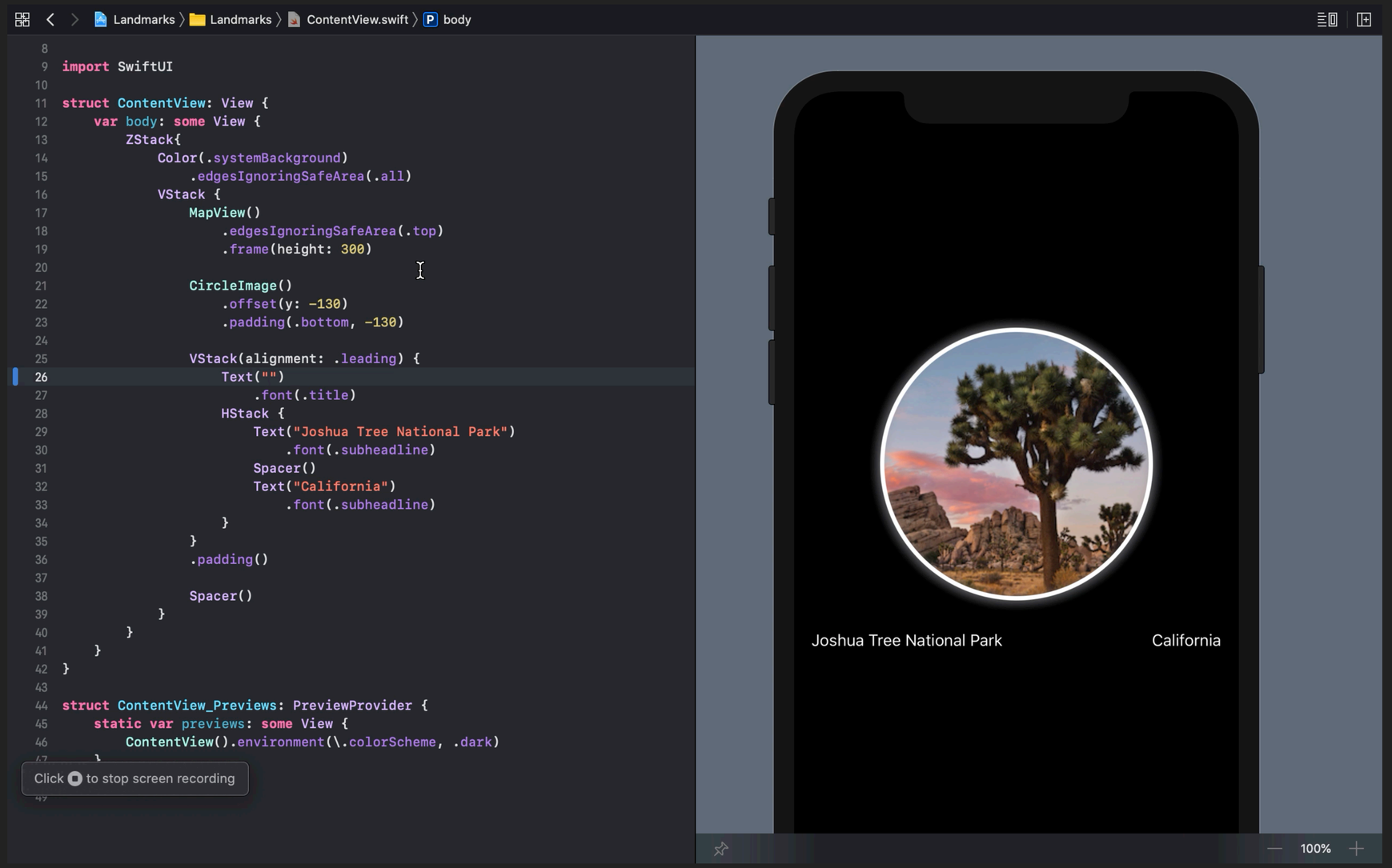
@EnvironmentObject



NO AUTO LAYOUT OR STORYBOARDS

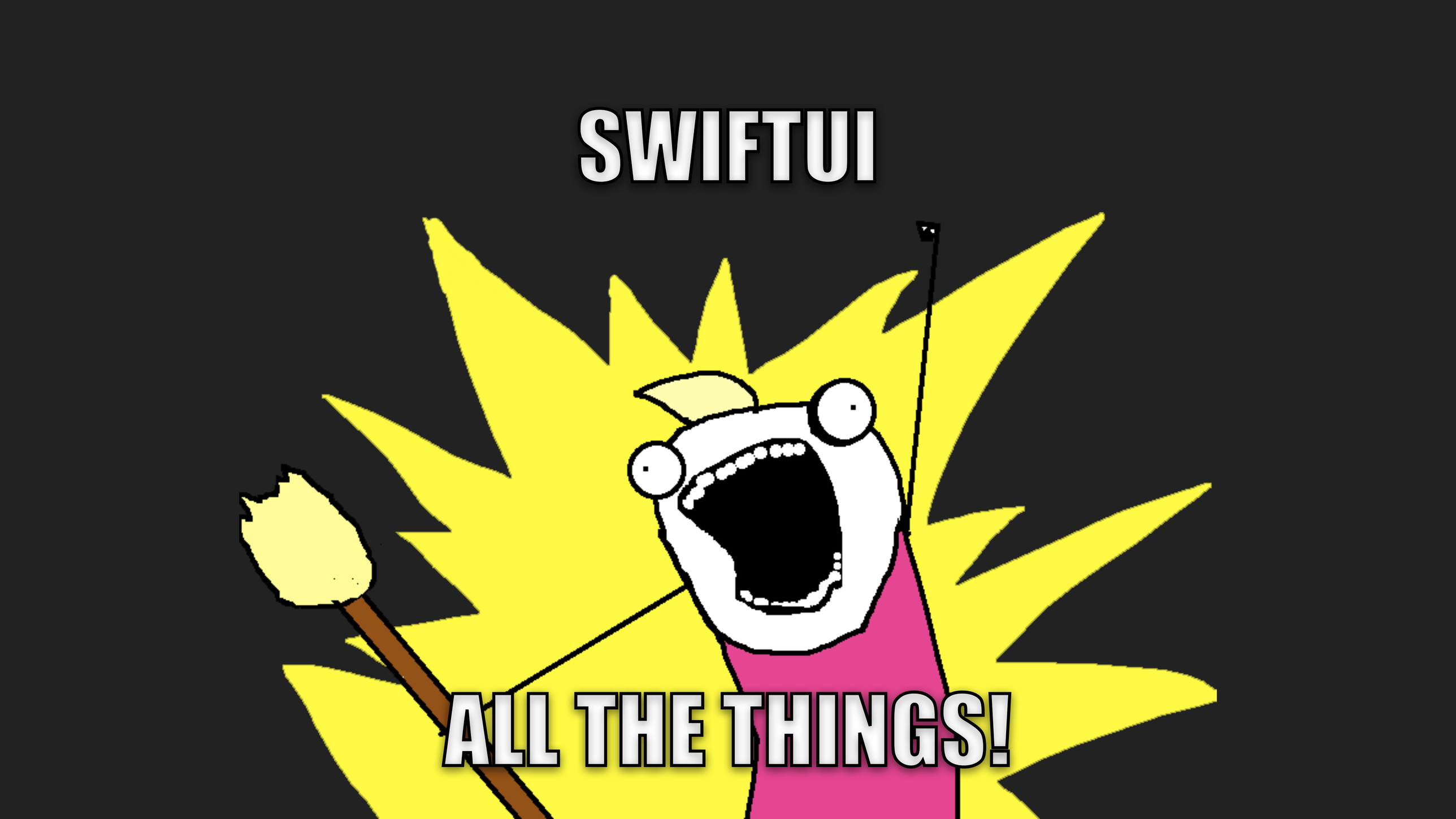
- No Constraints
- No Conflicting Constraints
- No Unsatisfiable Constraints
- No Storyboard Merge Conflicts

LIVE PREVIEW CANVAS



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

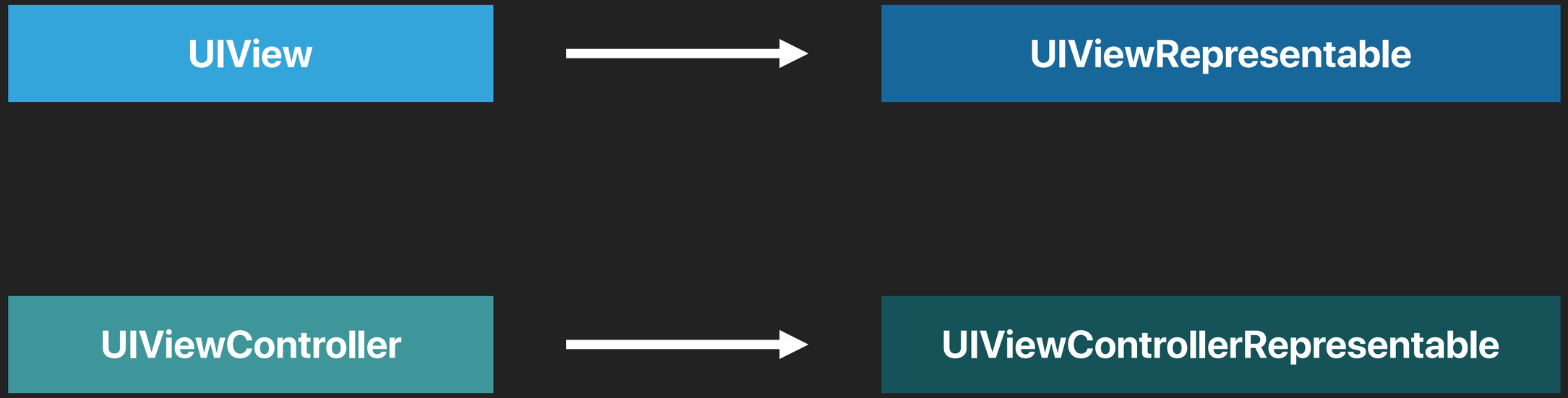
UIView

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

Update

Dismantle

REPRESENTABLE PROTOCOL

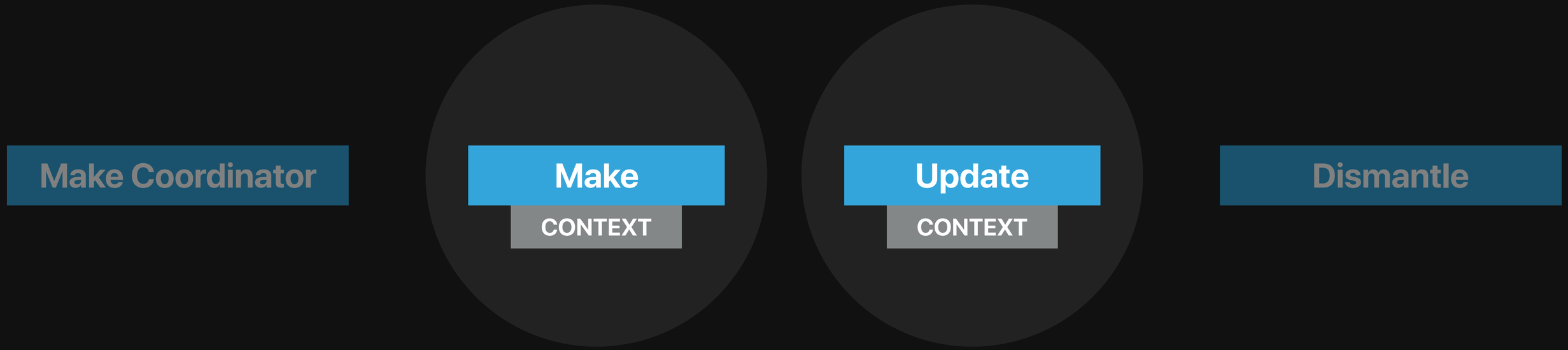
Make Coordinator

Make

Update

Dismantle

REPRESENTABLE PROTOCOL



CONTEXT

CONTEXT

- Environment
- Transaction
- Coordinator

REPRESENTABLE PROTOCOL

Make Coordinator

Make

CONTEXT

Update

CONTEXT

Dismantle

REPRESENTABLE PROTOCOL

Make Coordinator

Make

CONTEXT

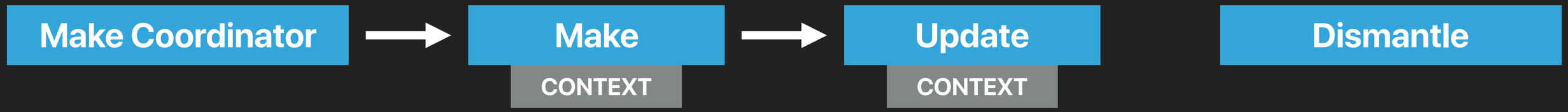
Update

CONTEXT

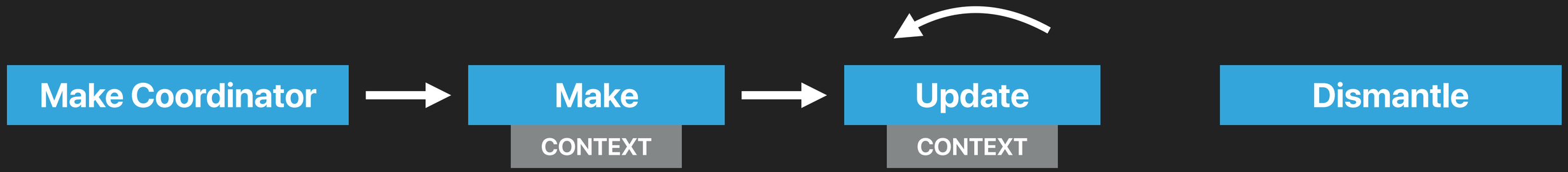
Dismantle

REPRESENTABLE PROTOCOL

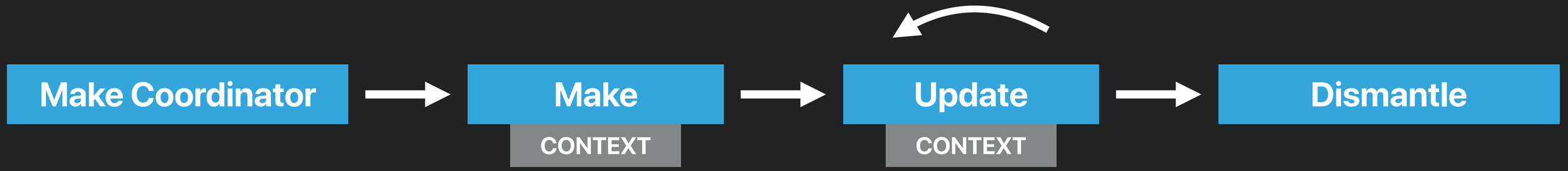
REPRESENTABLE PROTOCOL



REPRESENTABLE PROTOCOL



REPRESENTABLE PROTOCOL



INJECTING DATA

INJECTING DATA

- @EnvironmentObjects
- Passing data with Initializers

INJECTING DATA

- ~~@EnvironmentObjects~~
- Pass data with explicitly using initializers

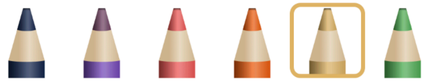
AUTOMATIC UI UPDATES

Notebook Style

Untitled

Cover Style

Primary Color



Default Decorative Tape

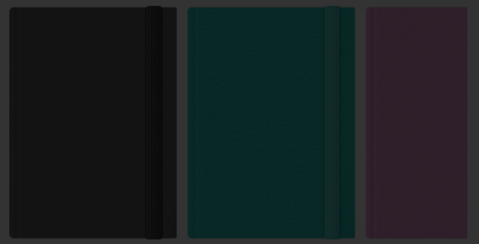
Default Heading Style

Default Font Color

Notebook Style

Untitled

Cover Style



Primary Color

Default Decorative Tape

Default Heading Style

Default Font Color

Notebook Style

Untitled

Cover Style

Primary Color

Default Decorative Tape

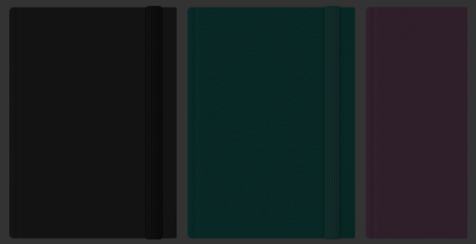
Default Heading Style

Default Font Color

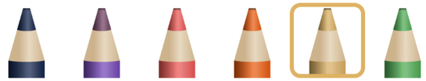
Notebook Style

Untitled

Cover Style



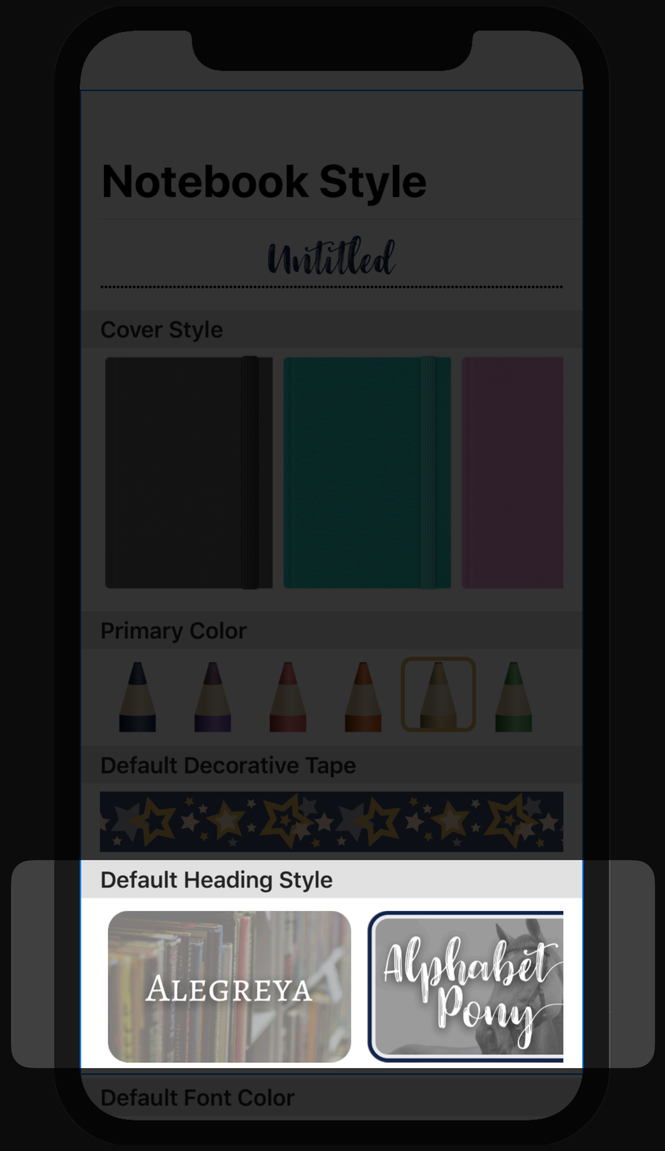
Primary Color



Default Decorative Tape

Default Heading Style

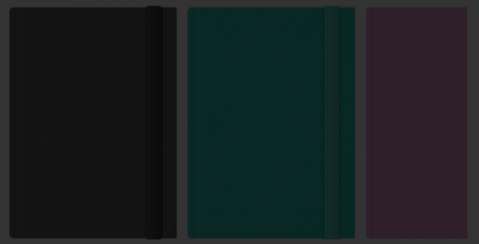
Default Font Color


Notebook Style
Untitled
Cover Style
Primary Color
Default Decorative Tape
Default Heading Style
ALEGREYA
Alphabet Pony
Default Font Color

Notebook Style

Untitled

Cover Style



Primary Color

Default Decorative Tape

Default Heading Style

Default Font Color

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

UIKIT + SWIFTUI

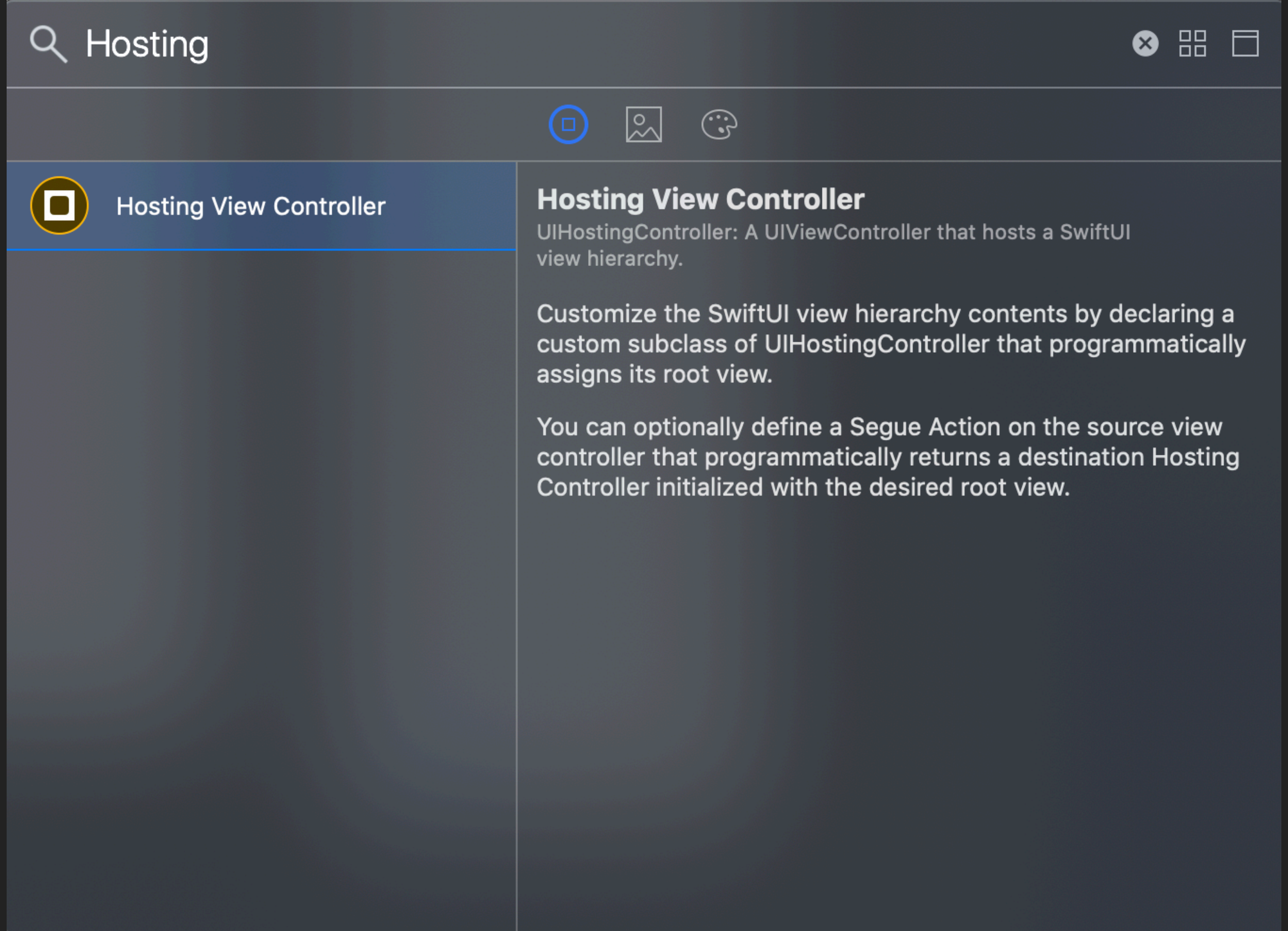
IN CODE

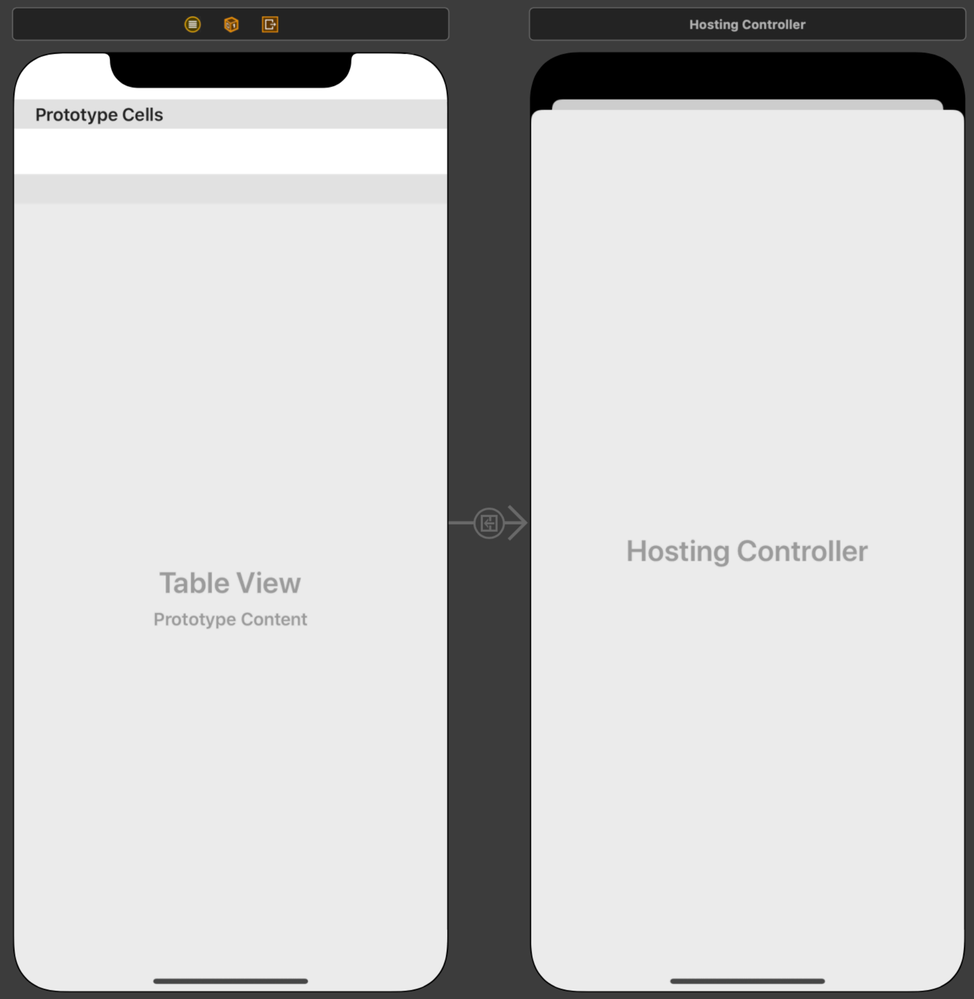
UIHostingController

- Subclass of UIViewController
- rootView property represents a SwiftUI view

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

STORYBOARD


Hosting
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.


Prototype Cells
Table View
Prototype Content
Hosting Controller
Hosting Controller

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 Representable View as Subview to Cell ContentView
    self.contentView.addSubview(view)

    //Add Contriants 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