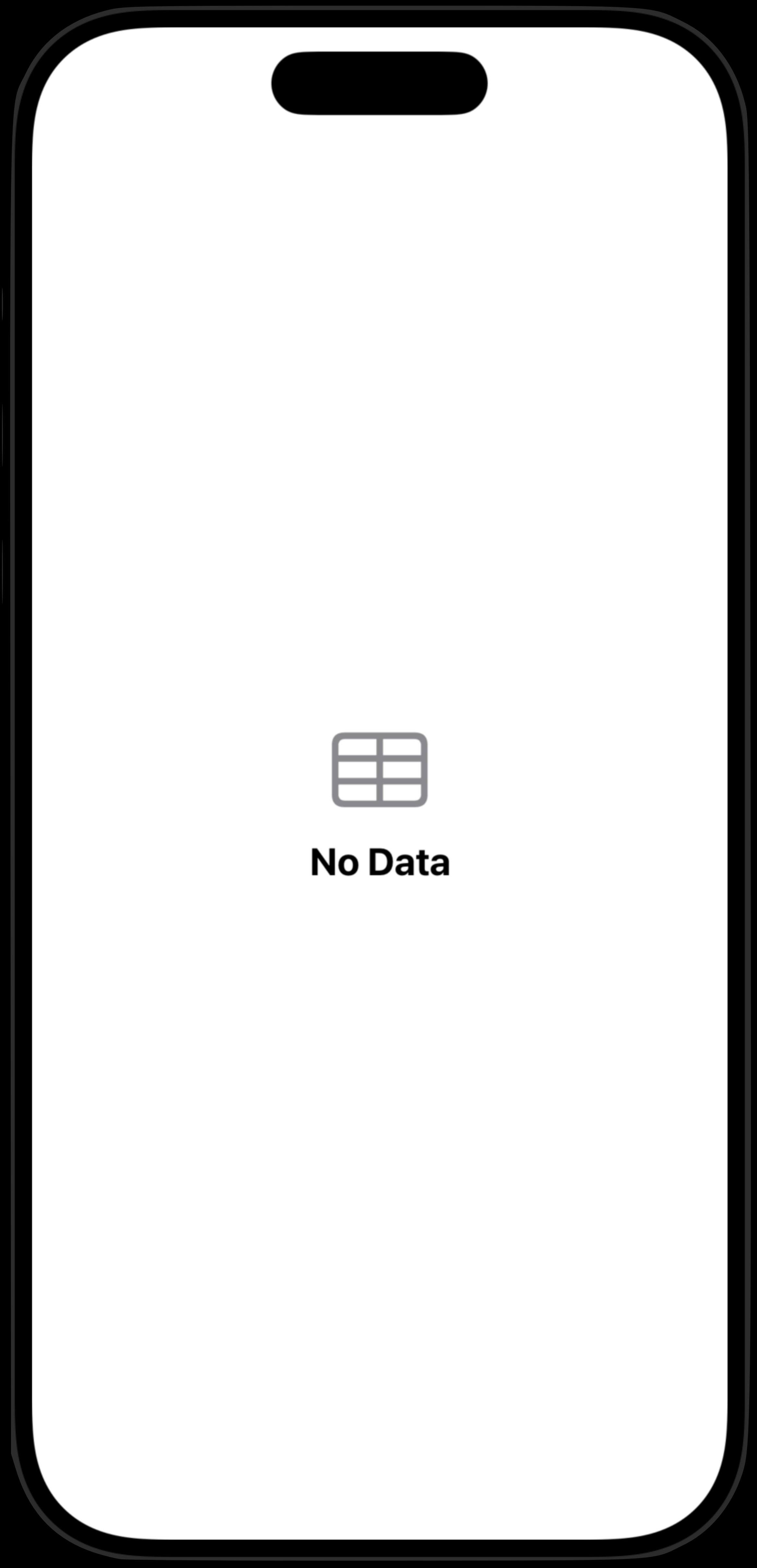
SwiftUI Style API Design

some Swift in SwiftUI

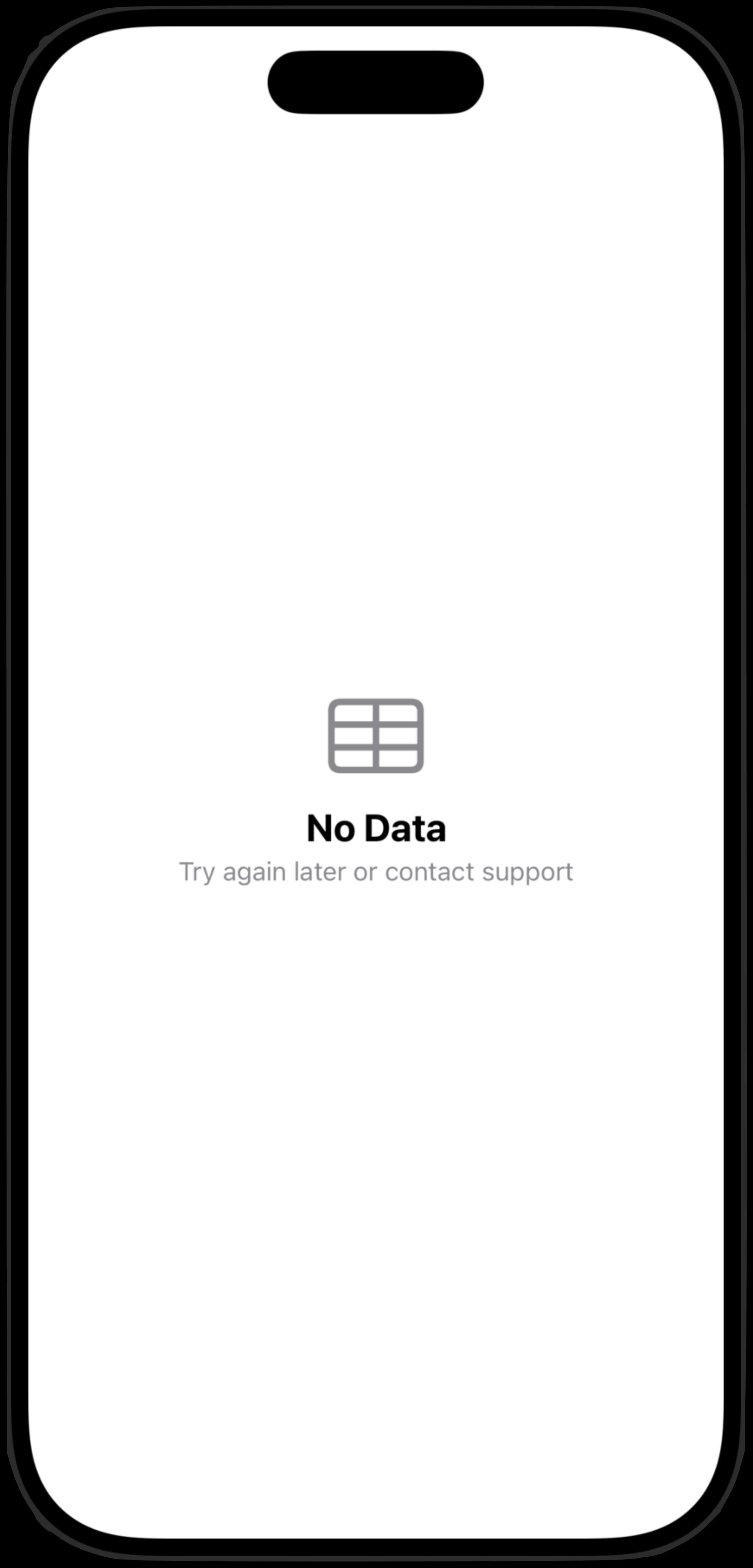
Ratnesh Jain

What SwiftUI Style API?

```
ContentUnavailableView("No Data", systemImage: "tablecells")
```

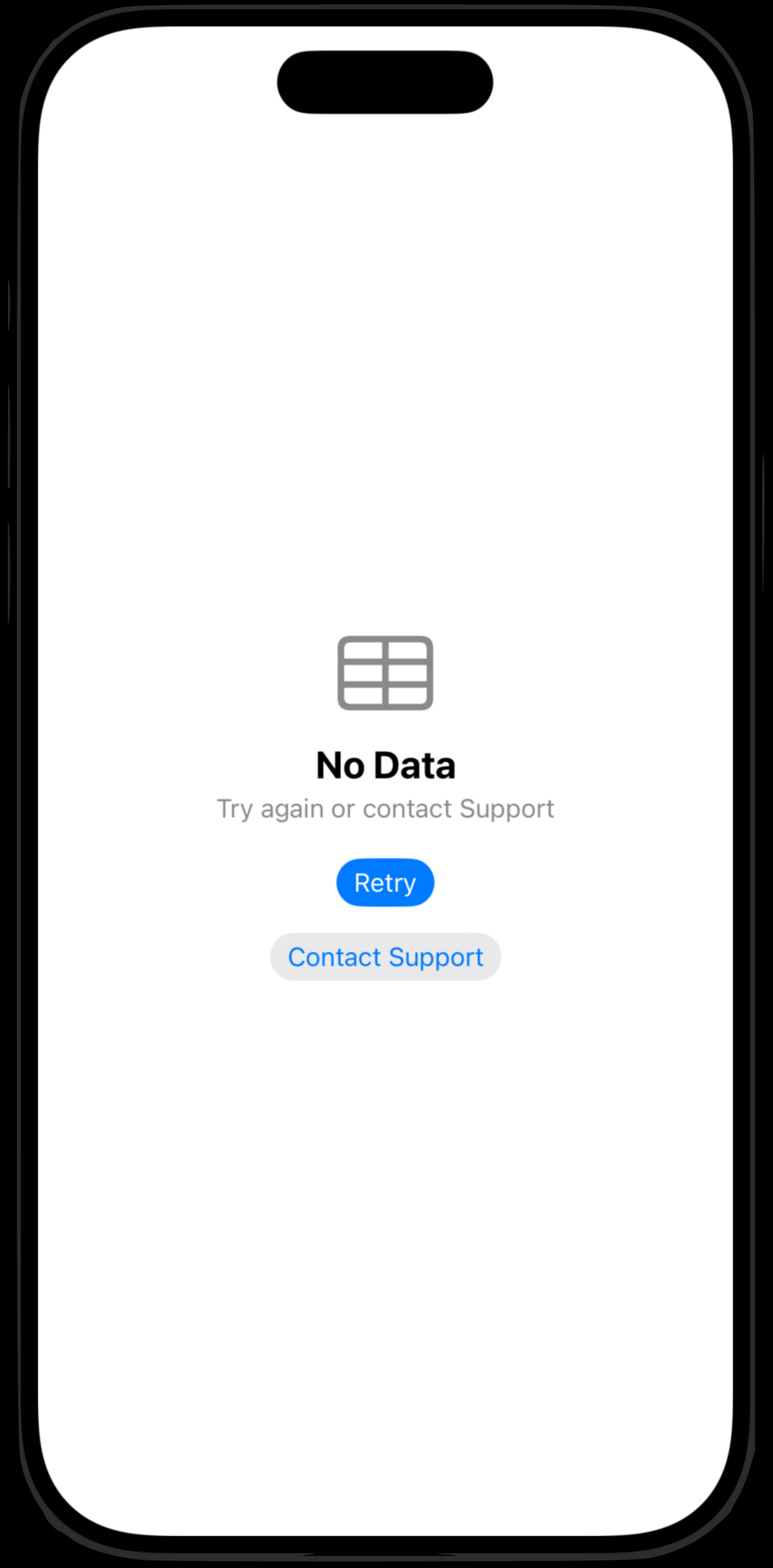


```
ContentUnavailableView(
    "No Data",
    systemImage: "tablecells",
    description: Text("Try again later or contact support")
)
```


No Data

Try again later or contact support

```swift
ContentUnavailableView {
    Label("No Data", systemImage: "tablecells")
} description: {
    Text("Try again or contact Support")
} actions: {
    Button("Retry") {}
        .buttonStyle(.borderedProminent)
    Button("Contact Support") {}
        .buttonStyle(.bordered)
}
```



Today

Swift in SwiftUI

- Enum
- Enum with Associated Value
- Generics
- Generic Constraints

Today

some SwiftUI

- View
- ViewModifier
- Environment

Today

Swift in SwiftUI

- Enum
- Enum with Associated Value
- Generics
- Generic Constraints

Enum

```
enum FetchingState {
    case fetching
    case fetched
    case error
}
```

Enum

```
enum FetchingState {
    case fetching
    case fetched
    case error

    var isFetching: Bool {
        guard self == .fetching else {
            return false
        }
        return true
    }

}
```

Enum

```
enum FetchingState {
    case fetching
    case fetched
    case error

    var isFetching: Bool {
        guard self == .fetching else {
            return false
        }
        return true
    }

}
```

Enum

```swift
enum FetchingState {
    case fetching
    case fetched
    case error

    var isFetching: Bool {
        guard self == .fetching else {
            return false
        }
        return true
    }

}
```

Enum

```
enum FetchingState {
    case fetching
    case fetched
    case error

    var isFetching: Bool {
        guard self == .fetching else {
            return false
        }
        return true
    }

    var isError: Bool {
        guard self == .error else {
            return false
        }
        return true
    }

}
```

Enum

```
enum FetchingState {
    case fetching
    case fetched
    case error

    var isFetching: Bool {
        guard self == .fetching else {
            return false
        }
        return true
    }

    var isError: Bool {
        guard self == .error else {
            return false
        }
        return true
    }

}
```

Enum

```
enum FetchingState {
    case fetching
    case fetched
    case error

    var isFetching: Bool {
        guard self == .fetching else {
            return false
        }
        return true
    }

    var isError: Bool {
        guard self == .error else {
            return false
        }
        return true
    }

}
```

Enum

```swift
enum FetchingState {
    case fetching
    case fetched
    case error
}


let state: FetchingState = .fetching

switch state {
case .fetching:
    print("Fetching")
case .fetched:
    print("Fetched")
case .error:
    print("Error")
}
```

Enum

```swift
enum FetchingState {
    case fetching
    case fetched
    case error
}


let state: FetchingState = .fetching

switch state {
case .fetching:
    print("Fetching")
case .fetched:
    print("Fetched")
case .error:
    print("Error")
}
```

Enum

```swift
enum FetchingState {
    case fetching
    case fetched
    case error
}


let state: FetchingState = .fetching

switch state {
case .fetching:
    print("Fetching")
case .fetched:
    print("Fetched")
case .error:
    print("Error")
}
```

Enum

```swift
enum FetchingState {
    case fetching
    case fetched
    case error
}


let state: FetchingState = .fetching

switch state {
case .fetching:
    print("Fetching")
case .fetched:
    print("Fetched")
case .error:
    print("Error")
}
```

Enum

```
enum FetchingState {
    case fetching
    case fetched
    case error
}


let state: FetchingState = .fetching

switch state {
case .fetching:
    print("Fetching")
case .fetched:
    print("Fetched")
case .error:
    print("Error")
}
```

Enum

```swift
enum FetchingState {
    case fetching
    case fetched
    case error
}


let state: FetchingState = .fetched

switch state {
case .fetching:
    print("Fetching")
case .fetched:
    print("Fetched")
case .error:
    print("Error")
}
```

Enum

```swift
enum FetchingState {
    case fetching
    case fetched
    case error
}


let state: FetchingState = .error

switch state {
case .fetching:
    print("Fetching")
case .fetched:
    print("Fetched")
case .error:
    print("Error")
}
```

Today

Swift in SwiftUI

- Enum
- Enum with Associated Value
- Generics
- Generic Constraints

Enum with Associated Value

```
enum FetchingState {
    case fetching
    case fetched
    case error(Error)
}
```

Enum with Associated Value

```
enum FetchingState {
    case fetching
    case fetched
    case error(Error)
}


let state: FetchingState = .error

switch state {
case .fetching:
    print("Fetching")
case .fetched:
    print("Fetched")
case .error:
    print("Error")
}
```

Enum with Associated Value

```swift
enum FetchingState {
    case fetching
    case fetched
    case error(Error)
}

let state: FetchingState = .error(AppError.somethingWentWrong)

switch state {
case .fetching:
    print("Fetching")
case .fetched(let error):
    print("Fetched")
case .error:
    print("Error")
}
```

Enum with Associated Value

```swift
enum FetchingState {
    case fetching
    case fetched
    case error(Error)

    var isFetching: Bool {
        guard case .fetching = self else {
            return false
        }
        return true
    }

}
```

Enum with Associated Value

```swift
enum FetchingState {
    case fetching
    case fetched
    case error(Error)

    var isFetching: Bool {
        guard case .fetching = self else {
            return false
        }
        return true
    }

}
```

Enum with Associated Value

```swift
enum FetchingState {
    case fetching
    case fetched
    case error(Error)

    var isFetching: Bool {
        guard case .fetching = self else {
            return false
        }
        return true
    }

    var errorValue: Error? {
        guard case .error(let error) = self else {
            return nil
        }
        return error
    }

}
```

Enum with Associated Value

```
enum FetchingState {
    case fetching
    case fetched
    case error(Error)

    var isFetching: Bool {
        guard case .fetching = self else {
            return false
        }
        return true
    }

    var errorValue: Error? {
        guard case .error(let error) = self else {
            return nil
        }
        return error
    }

}
```

Today

Swift in SwiftUI

- Enum
- Enum with Associated Value
- Generics
- Generic Constraints

Generics

```swift
enum FetchingState {
    case fetching
    case fetched(T)
    case error(Error)
}
```

Generics

```swift
enum FetchingState<T> {
    case fetching
    case fetched(T)
    case error(Error)
}
```

Generics

```
enum FetchingState<Value> {
    case fetching
    case fetched(Value)
    case error(Error)
}
```

Generics

```swift
enum FetchingState<Value> {
    case fetching
    case fetched(Value)
    case error(Error)
}

func perform() {
    let state: FetchingState = .fetching

    switch state {
    case .fetching:
        print("Fetching")
    case .fetched:
        print("Fetched")
    case .error(let error):
        print("Error")
    }
}
```

Generics

```swift
enum FetchingState<Value> {
    case fetching
    case fetched(Value)
    case error(Error)
}

func perform() {
    let state: FetchingState<Int> = .fetching

    switch state {
    case .fetching:
        print("Fetching")
    case .fetched(let value):
        print(“Fetched")
    case .error(let error):
        print("Error")
    }
}
```

Generics

```swift
enum FetchingState<Value> {
    case fetching
    case fetched(Value)
    case error(Error)
}

func perform() {
    let state: FetchingState<[User]> = .fetching

    switch state {
    case .fetching:
        print("Fetching")
    case .fetched(let value):
        print("Fetched")
    case .error(let error):
        print("Error")
    }
}
```

Generics

```swift
enum FetchingState<Value> {
    case fetching
    case fetched(Value)
    case error(Error)
}

func perform() {
    let state: FetchingState<[User]> = .fetching

    switch state {
    case .fetching:
        print("Fetching")
    case .fetched(let value):
        print("Fetched")
    case .error(let error):
        print("Error")
    }
}
```

Free Generic

No Condition for Type

Any type can be used for <Value>

Generics

```
enum FetchingState<Value> {
    case fetching
    case fetched(Value)
    case error(Error)
}

func perform() {
    let state: FetchingState<[User]> = .fetching

    switch state {
    case .fetching:
        print("Fetching")
    case .fetched(let value):
        print("Fetched")
    case .error(let error):
        print("Error")
    }
}
```

Additing a Condition

To < Value > is called

Applying Type Constraint

Generics

```swift
enum FetchingState<Value> {
    case fetching
    case fetched(Value)
    case error(Error)
}

func perform() {
    let state: FetchingState<[User]> = .fetching

    switch state {
    case .fetching:
        print("Fetching")
    case .fetched(let value):
        print("Fetched")
    case .error(let error):
        print("Error")
    }
}
```

Type Constraint

By Protocol

By Concrete Type

Today

Swift in SwiftUI

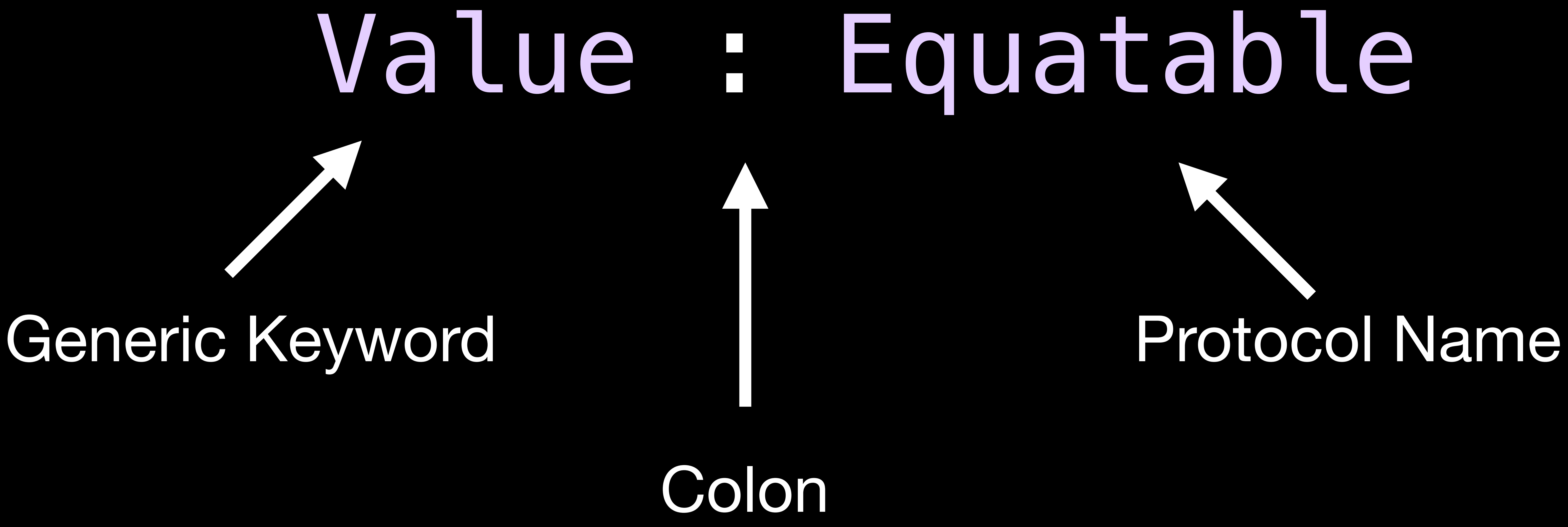
- Enum
- Enum with Associated Value
- Generics
- Generic Constraints

Generics Constraint

```swift
enum FetchingState<Value: Equatable> {
    case fetching
    case fetched(Value)
    case error(Error)
}
```

Generics Constraint

```swift
enum FetchingState<Value: Equatable> {
    case fetching
    case fetched(Value)
    case error(Error)
}
```



Generics Constraint

```swift
enum FetchingState<Value> where Value : Equatable {
    case fetching
    case fetched(Value)
    case error(Error)
}
```

Generics Constraint

```swift
enum FetchingState<Value> where Value : Equatable & Sendable {
    case fetching
    case fetched(Value)
    case error(Error)
}
```

Generics Constraint

```swift
enum FetchingState<Value> where Value : Equatable & Sendable {
    case fetching
    case fetched(Value)
    case error(Error)
}
```

& Sendable

Protocol Composition

Generics Constraint

```swift
enum FetchingState<Value> {
    case fetching
    case fetched(Value)
    case error(Error)
}
```

Generics Constraint

```swift
enum FetchingState<Value> {
    case fetching
    case fetched(Value)
    case error(Error)
}

extension FetchingState where Value == Int {
    var count: Int {
        switch self {
        case .fetching:
            return 0
        case .fetched(let value):
            return value
        case .error:
            return 0
        }
    }
}
```

Generics Constraint

```swift
enum FetchingState<Value> {
    case fetching
    case fetched(Value)
    case error(Error)
}

extension FetchingState where Value == Int {
    var count: Int {
        switch self {
        case .fetching:
            return 0
        case .fetched(let value):
            return value
        case .error:
            return 0
        }
    }
}
```

Generics Constraint

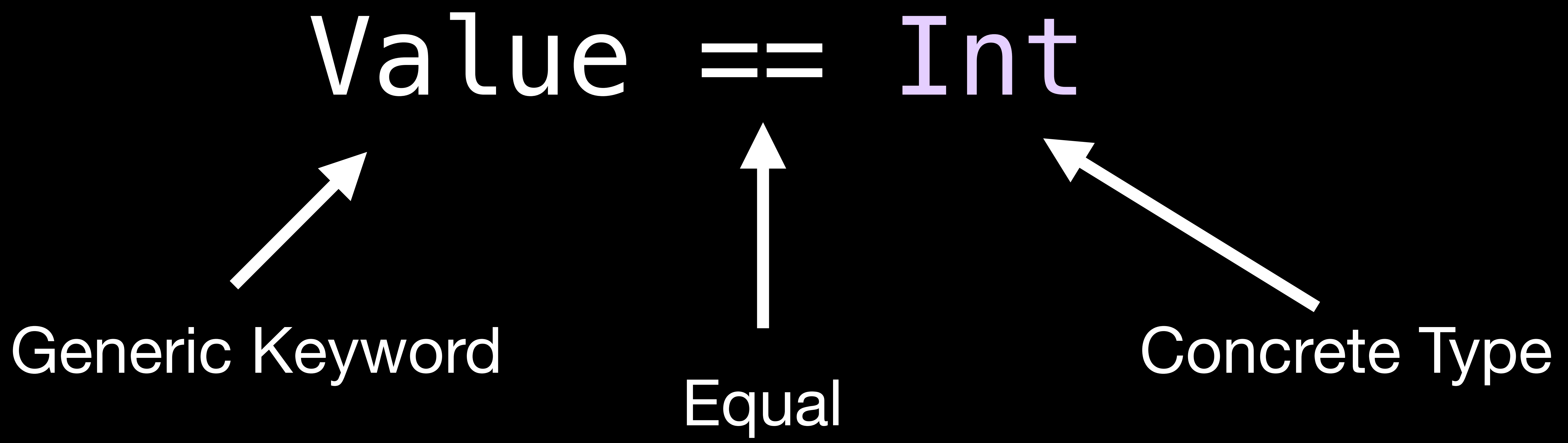
```swift
enum FetchingState<Value> {
    case fetching
    case fetched(Value)
    case error(Error)
}

extension FetchingState where Value == Int {
    var count: Int {
        switch self {
        case .fetching:
            return 0
        case .fetched(let value):
            return value
        case .error:
            return 0
        }
    }
}
```

Generics Constraint

```swift
enum FetchingState<Value> {
    case fetching
    case fetched(Value)
    case error(Error)
}

extension FetchingState where Value == Int {
    var count: Int {
        switch self {
        case .fetching:
            return 0
        case .fetched(let value):
            return value
        case .error:
            return 0
        }
    }
}
```

Generics Constraint

```swift
enum FetchingState <Value> {
    case fetching
    case fetched(Value)
    case error(Error)
}

extension FetchingState where Value == Int {
   ...
}
```



Today

Swift in SwiftUI

- Enum
- Enum with Associated Value
- Generics
- Generic Constraints

Today

some SwiftUI

- View
- ViewModifier
- Environment

Text Image List ScrollView ForEach Color Shape

TapGesture DragGesture MagnifyGesture RotateGesture SpatialTapGesture

ExclusiveGesture Menu Label ModifiedContent MultiDatePicker NavigationLink

NavigationSplitView NavigationStack NavigationView OffsetShape OutlineGroup

OutlineSubgroupChildren PasteButton PhaseAnimator Picker PlaceholderContentView

PlaceholderTextShapeStyle PresentedWindowContent ProgressView RenameButton

ScrollViewReader Section SecureField ShareLink Slider Spacer Stepper

StrokeBorderShapeView StrokeShapeView SubscriptionView TabView Table Text

TextEditor TextField TimelineView Toggle TupleView UIViewRepresentable VStack

ViewThatFits WindowGroup Optional Never AsyncImage Button EditButton

Ratnesh Jain

Text Image List ScrollView ForEach Color Shape

TapGesture

Menu Label MultiDatePicker NavigationLink

NavigationStack NavigationView

PasteButton Picker

ProgressView

ScrollViewReader Section SecureField ShareLink Slider Spacer Stepper

SubscriptionView TabView Text

TextEditor TextField Toggle UIViewRepresentable VStack

WindowGroup AsyncImage Button

View

```swift
@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
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 ``View/body-swift.property`` property.
    associatedtype Body : View

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

View

```swift
@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
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 ``View/body-swift.property`` property.
    associatedtype Body : View

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

View

- Primitive Views
- Container Views
- View Modifiers
- View Readers

View

- Primitive Views
- Container Views
- View Modifiers
- View Readers

Text

Image

Color

Divider

Spacer

Gestures

Material

View

- Primitive Views
- **Container Views**
- View Modifiers
- View Readers

HStack
VStack
ZStack
LazyHStack
LazyVStack

Grid
LazyHGrid
LazyVGrid

List
ScrollView
ForEach

TabView
NavigationView
NavigationStack
NavigationSplitView

Label
Button
Menu

View

- Primitive Views
- Container Views
- View Modifiers
- View Readers

A modifier that you apply to a view or another view modifier, producing a different version of the original value.

SwiftUI Documentation

https://developer.apple.com/documentation/swiftui/viewmodifier

View

- Primitive Views
- Container Views
- View Modifiers
- View Readers

To

- *Decorate*
- *Pass values via Env / Pref*
- *Read Dimensions*
- *Observe life cycle events*

ViewModifier

- Decoration
- Environment / Preference
- Observe Life Cycles

```
.padding()
.background(Color.yellow)
.clipShape(.rect(cornerRadius: 12, style: .continuous))
```

ViewModifier

- Decoration
- Environment / Preference
- Observe Life Cycles

```
List(selection: $selection) {
    ForEach(1...10, id: \.self) { index in
        Text("\(index)")
    }
}
.environment(\.editMode, .constant(.active))
```

ViewModifier

- Decoration
- Environment / Preference
- Observe Life Cycles

```
List(selection: $selection) {
    ForEach(1...10, id: \.self) { index in
        Text("\(index)")
    }
}
.environment(\.editMode, .constant(.active))
```

ViewModifier

- Decoration
- Environment / Preference
- Observe Life Cycles

```
ContentView()
.onAppear {
    print("On Appear")
}
.onDisappear {
    print("On Disappear Called")
}
.task {
    print("Creates new task on OnAppear and cancels it on Disappear")
}
```

ViewModifier

- Decoration
- Environment / Preference
- Observe Life Cycles

```
ContentView()
.onAppear {
    print("On Appear")
}
.onDisappear {
    print("On Disappear Called")
}
.task {
    print("Creates new task on OnAppear and cancels it on Disappear")
}
```

ViewModifier

- Decoration
- Environment / Preference
- Observe Life Cycles

```
ContentView()
.onAppear {
    print("On Appear")
}
.onDisappear {
    print("On Disappear Called")
}
.task {
    print("Creates new task on OnAppear and cancels it on Disappear")
}
```

ViewModifier

- Decoration
- Environment / Preference
- Observe Life Cycles

```
ContentView()
.onAppear {
    print("On Appear")
}
.onDisappear {
    print("On Disappear Called")
}
.task {
    print("Creates new task on OnAppear and cancels it on Disappear")
}
```

ViewModifier

- Primitive Views
- Container Views
- View Modifiers
- View Readers

```
.background {
    GeometryReader { proxy in
        Color.clear
            .onChange(of: proxy.size, initial: true) { oldValue, newValue in
                self.size = newValue
            }
            .onChange(of: proxy.frame(in: .global), initial: true) { oldValue, newValue in
                self.position = newValue.origin
            }
    }
}
```

ViewModifier

- Primitive Views
- Container Views
- View Modifiers
- View Readers

```
.background {
    GeometryReader { proxy in
        Color.clear
            .onChange(of: proxy.size, initial: true) { oldValue, newValue in
                self.size = newValue
            }
            .onChange(of: proxy.frame(in: .global), initial: true) { oldValue, newValue in
                self.position = newValue.origin
            }
    }
}
```

ViewModifier

- Primitive Views
- Container Views
- View Modifiers
- View Readers

```
.background {
    GeometryReader { proxy in
        Color.clear
            .onChange(of: proxy.size, initial: true) { oldValue, newValue in
                self.size = newValue
            }
            .onChange(of: proxy.frame(in: .global), initial: true) { oldValue, newValue in
                self.position = newValue.origin
            }
    }
}
```

ViewModifier

- Primitive Views
- Container Views
- View Modifiers
- View Readers

```
.background {
    GeometryReader { proxy in
        Color.clear
            .onChange(of: proxy.size, initial: true) { oldValue, newValue in
                self.size = newValue
            }
            .onChange(of: proxy.frame(in: .global), initial: true) { oldValue, newValue in
                self.position = newValue.origin
            }
    }
}
```

Today

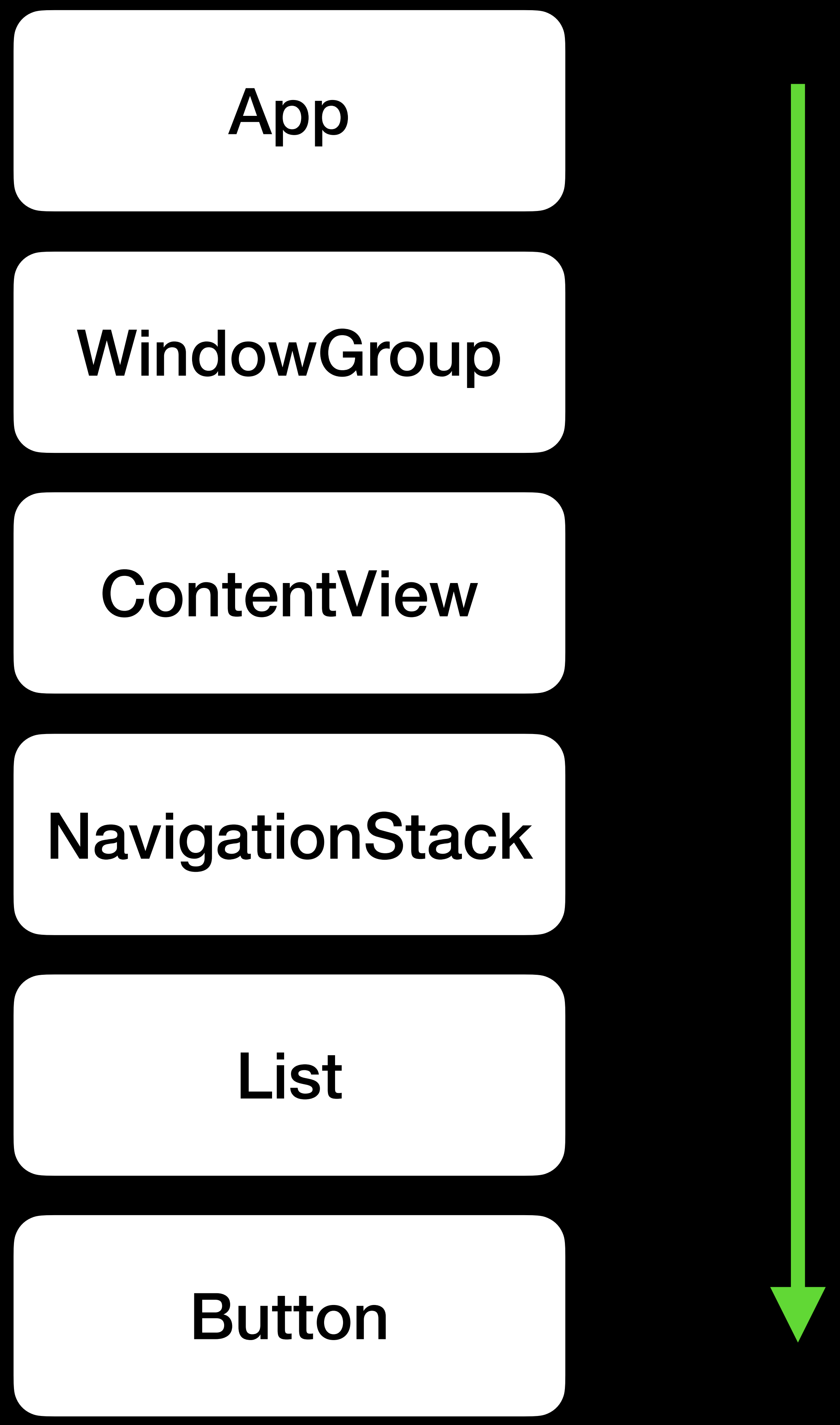
some SwiftUI

- View
- ViewModifier
- Environment

Environment

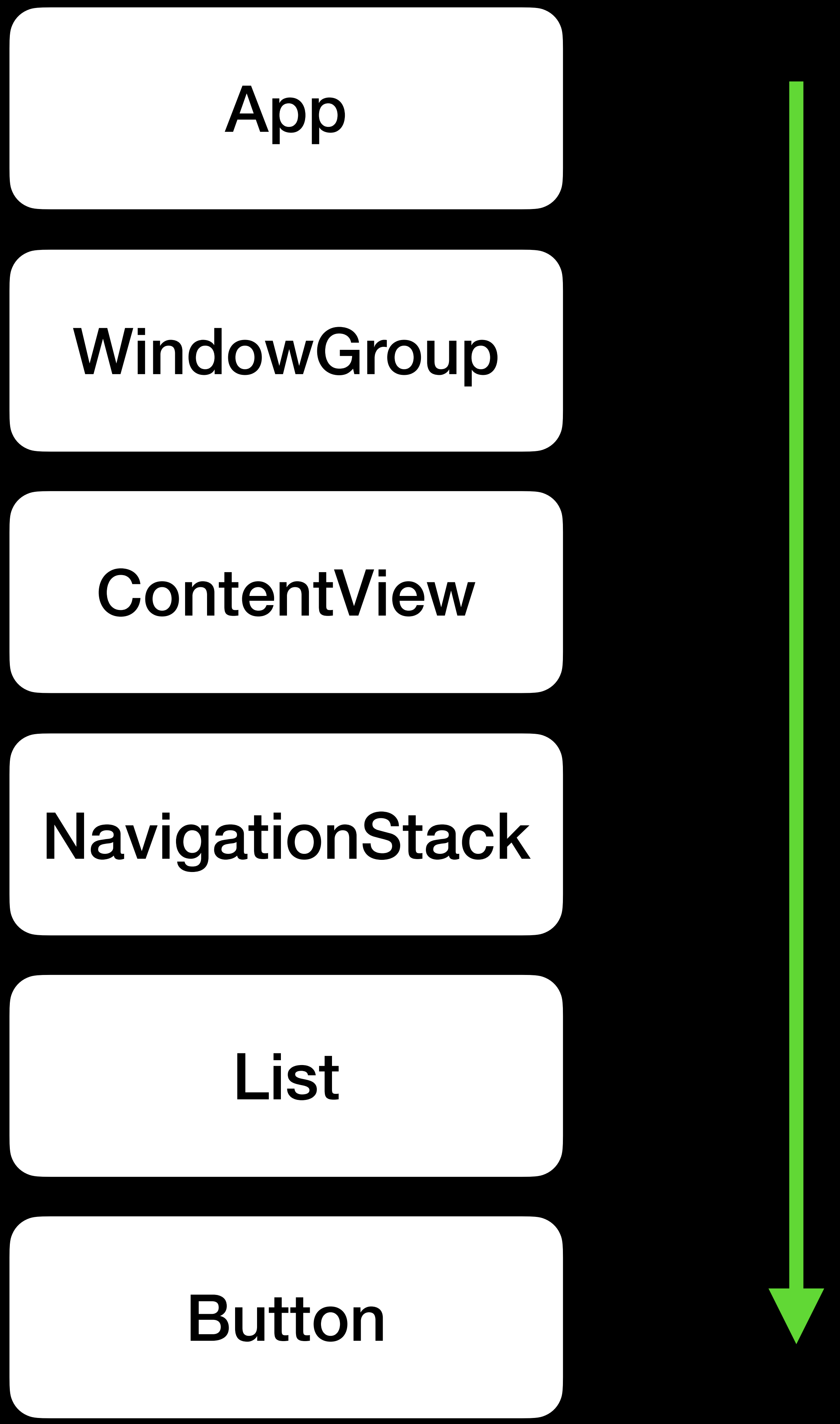
- Like Global variable / Singleton
- Allows hierarchical overriding
- 3 Type System
  - EnvironmentKey
  - EnvironmentValues
  - @Environment property Wrapper

Environment



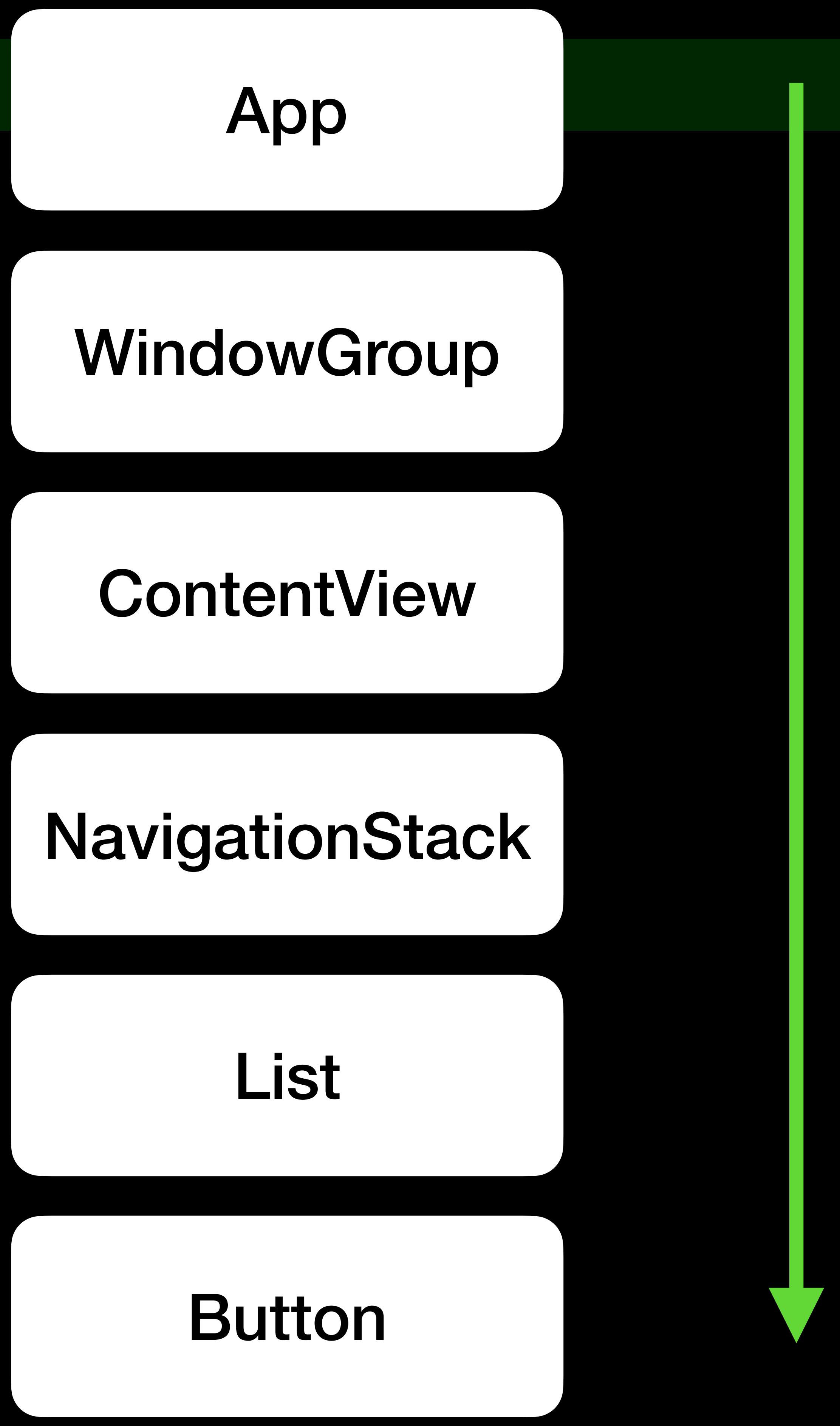
Environment

```swift
enum AppTheme {
    case neon
    case shine
    case dance

    var foregroundColor: Color {
        switch self {
        case .neon: Color.yellow
        case .shine: Color.green
        case .dance: Color.red
        }
    }

    var backgroundColor: Color {
        switch self {
        case .neon: Color.brown
        case .shine: Color.gray
        case .dance: Color.blue
        }
    }
}
```



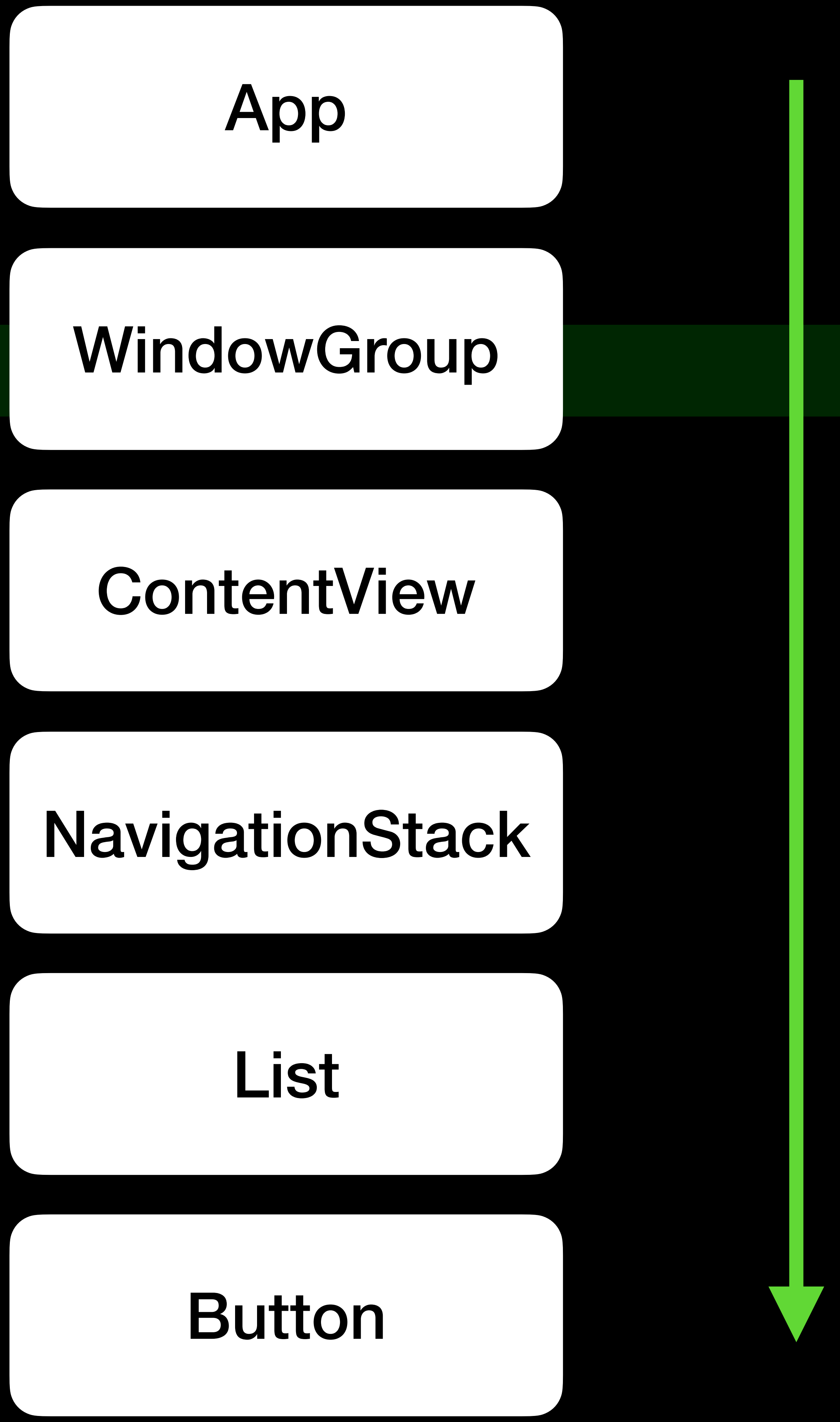
Environment

```
enum AppTheme {
    case neon
    case shine
    case dance

    var foregroundColor: Color {
        switch self {
        case .neon: Color.yellow
        case .shine: Color.green
        case .dance: Color.red
        }
    }

    var backgroundColor: Color {
        switch self {
        case .neon: Color.brown
        case .shine: Color.gray
        case .dance: Color.blue
        }
    }
}
```



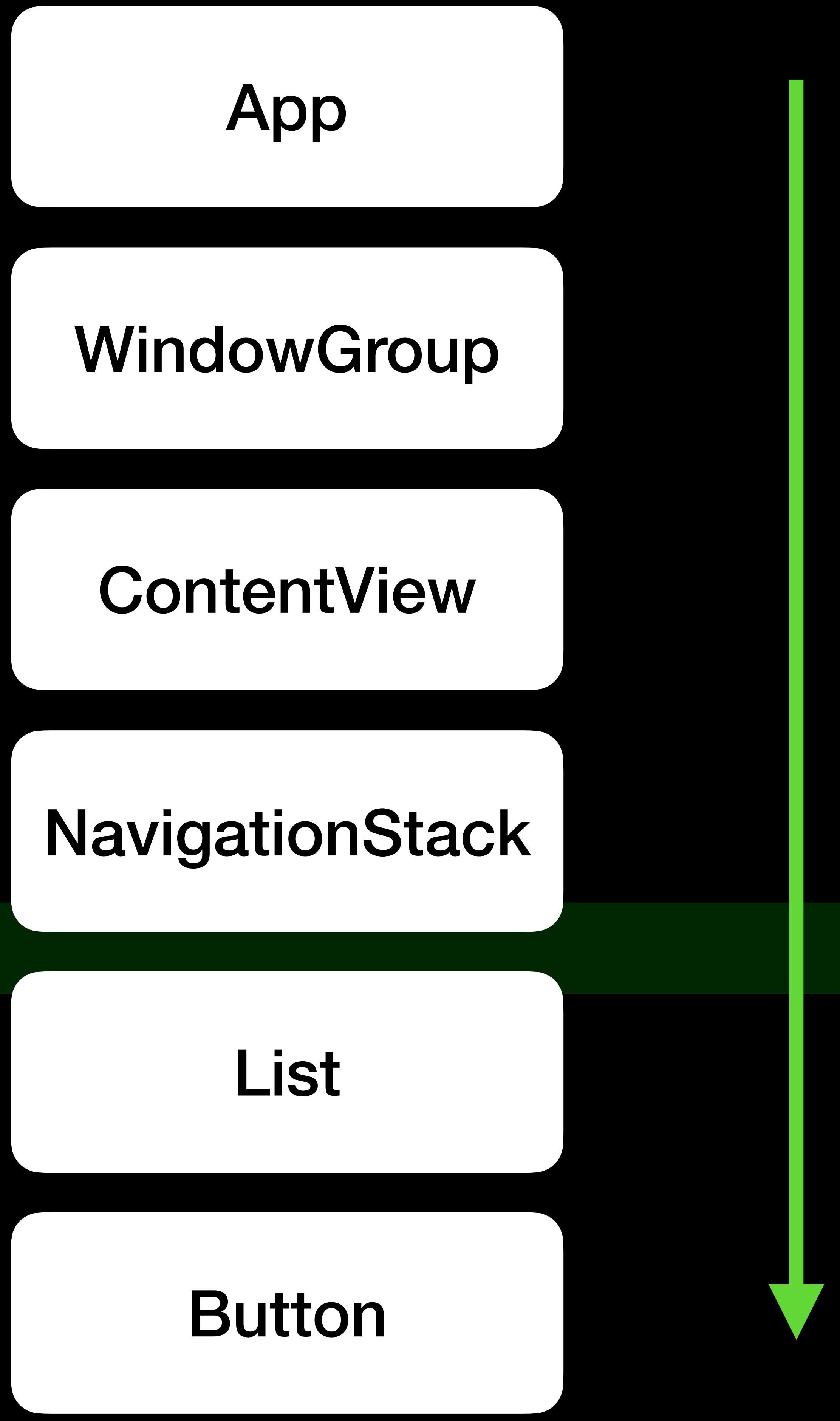
Environment

```
enum AppTheme {
    case neon
    case shine
    case dance

    var foregroundColor: Color {
        switch self {
        case .neon: Color.yellow
        case .shine: Color.green
        case .dance: Color.red
        }
    }

    var backgroundColor: Color {
        switch self {
        case .neon: Color.brown
        case .shine: Color.gray
        case .dance: Color.blue
        }
    }
}
```



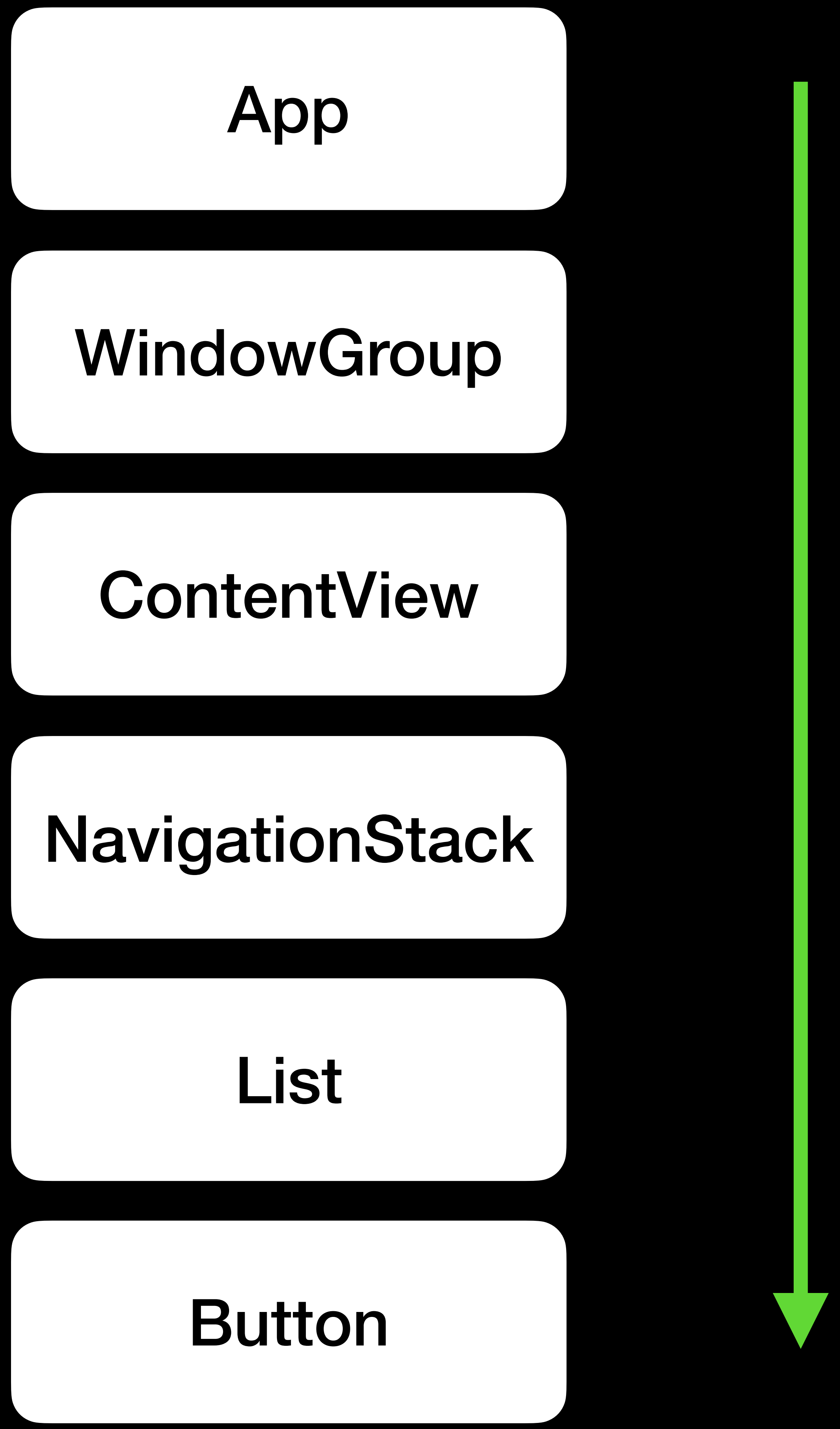
Environment

```swift
enum AppTheme {
    case neon
    case shine
    case dance

    var foregroundColor: Color {
        switch self {
        case .neon: Color.yellow
        case .shine: Color.green
        case .dance: Color.red
        }
    }

    var backgroundColor: Color {
        switch self {
        case .neon: Color.brown
        case .shine: Color.gray
        case .dance: Color.blue
        }
    }
}
```



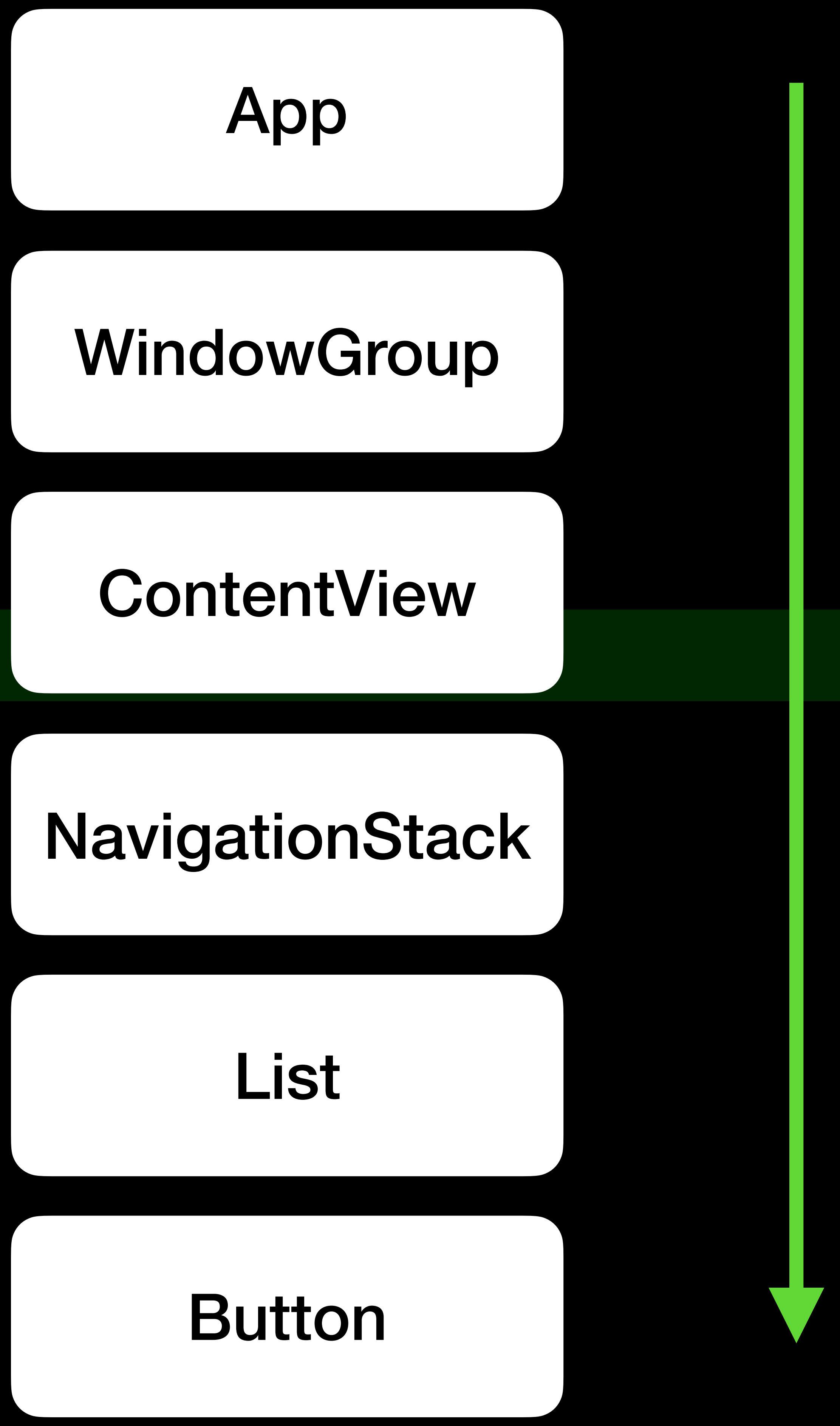
Environment

```
enum AppTheme {
    case neon
    case shine
    case dance

    var foregroundColor: Color {…}
    var backgroundColor: Color {…}
}

extension AppTheme: EnvironmentKey {
    static var defaultValue: AppTheme {
        AppTheme.neon
    }
}
```



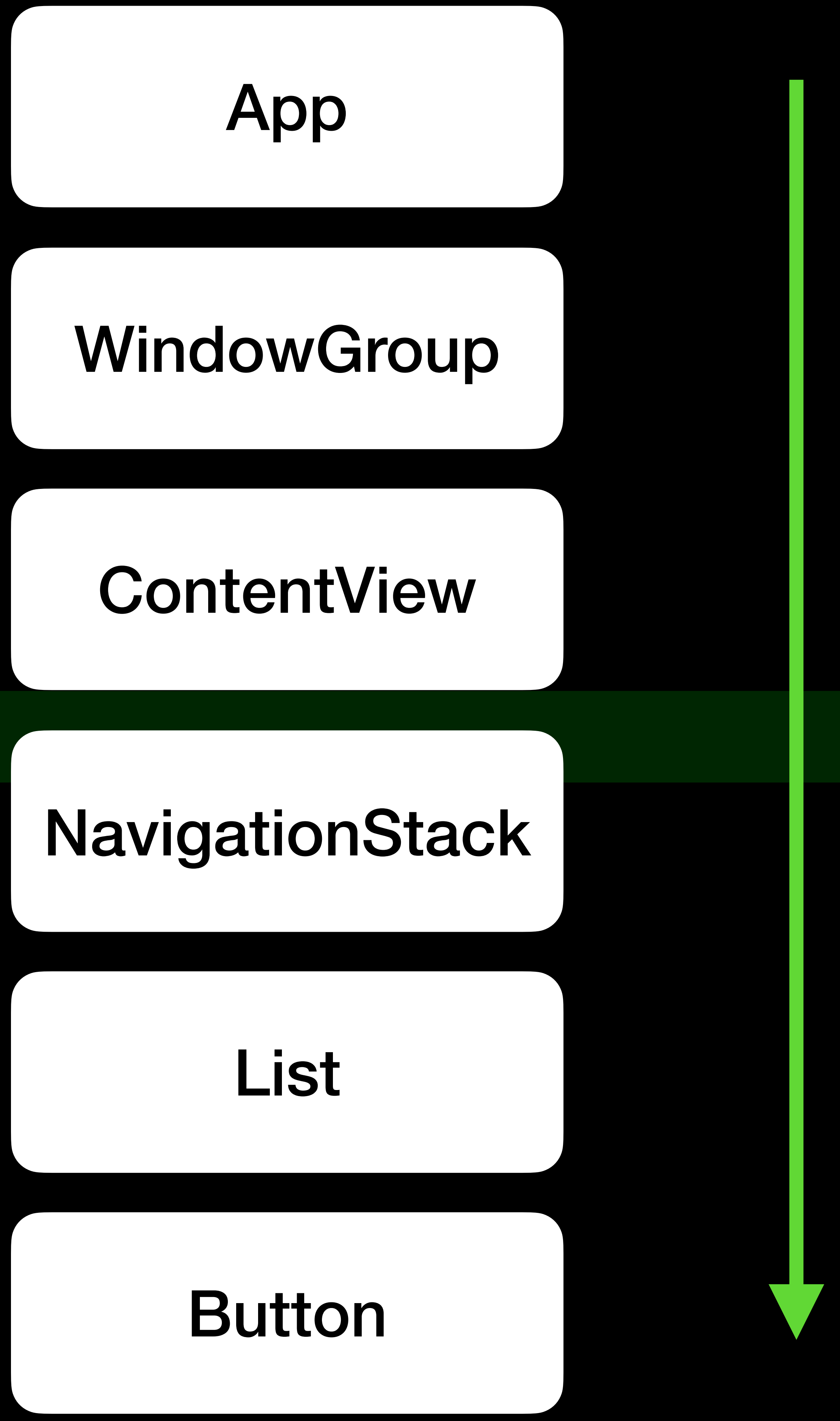
Environment

```
enum AppTheme {
    case neon
    case shine
    case dance

    var foregroundColor: Color {…}
    var backgroundColor: Color {…}
}

extension AppTheme: EnvironmentKey {
    static var defaultValue: AppTheme {
        AppTheme.neon
    }
}
```



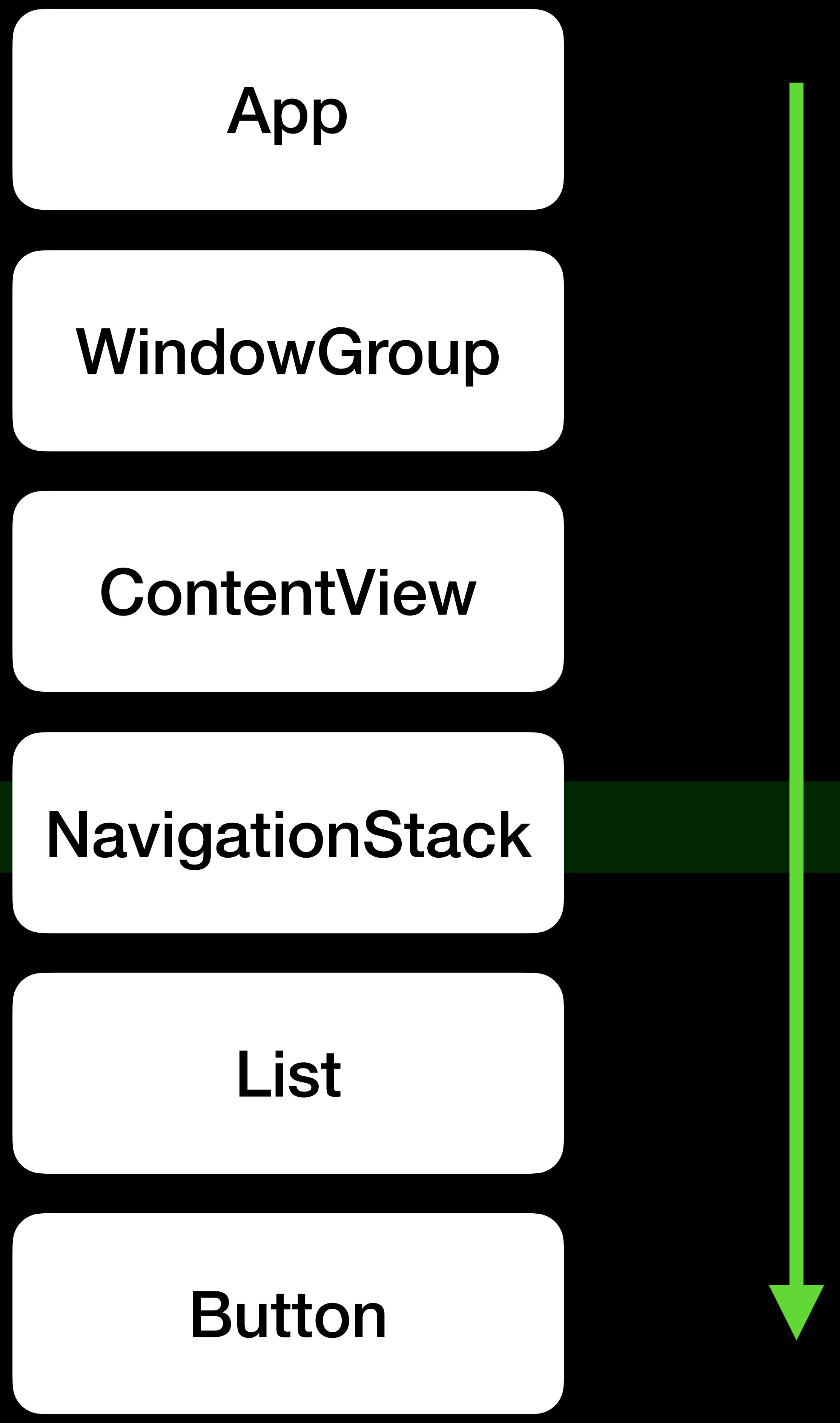
Environment

```
enum AppTheme {
    case neon
    case shine
    case dance

    var foregroundColor: Color {…}
    var backgroundColor: Color {…}
}

extension AppTheme: EnvironmentKey {
    static var defaultValue: AppTheme {
        AppTheme.neon
    }
}
```



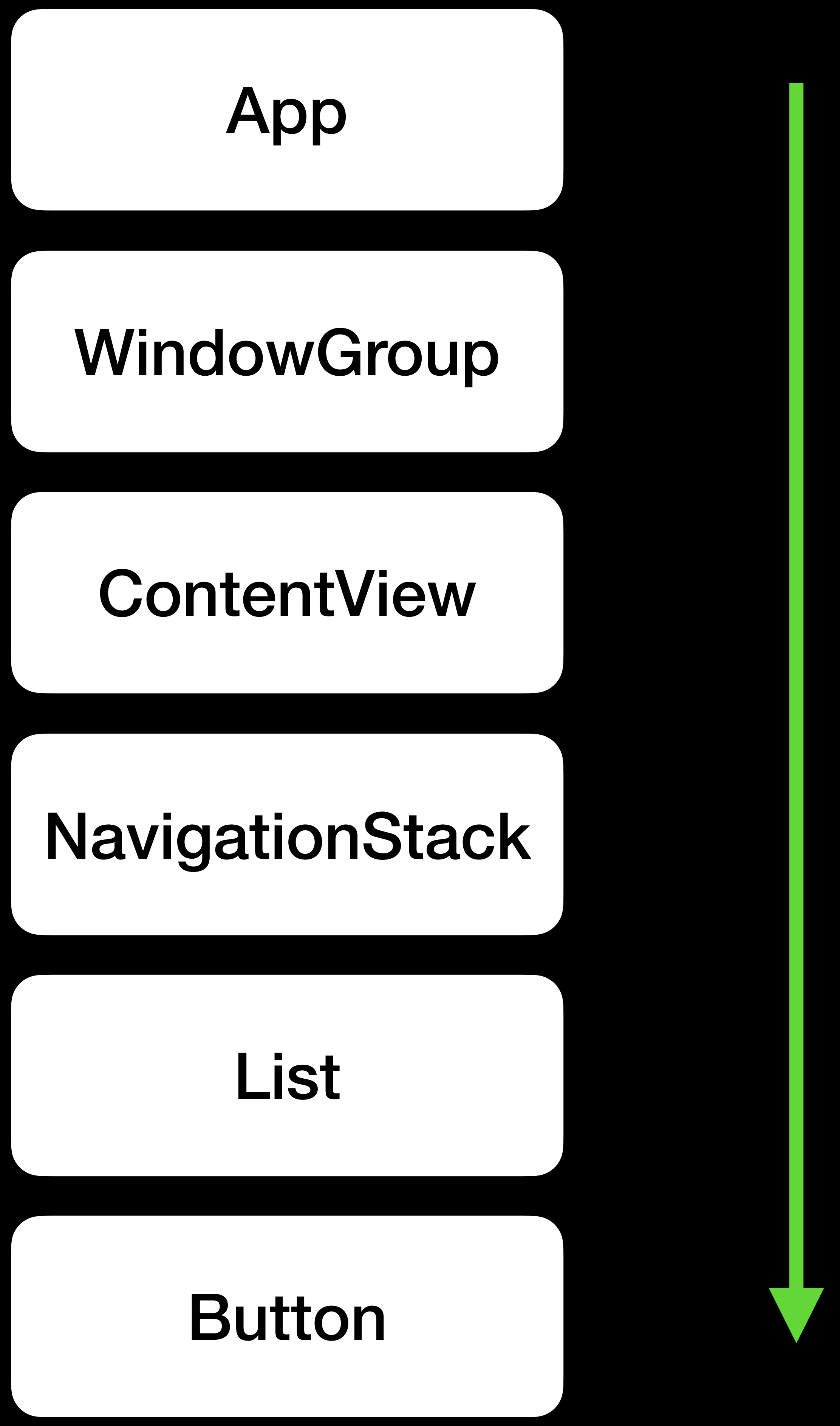
Environment

```
enum AppTheme {
    case neon
    case shine
    case dance

    var foregroundColor: Color {…}
    var backgroundColor: Color {…}
}

extension AppTheme: EnvironmentKey {
    static var defaultValue: AppTheme {
        AppTheme.neon
    }
}
```



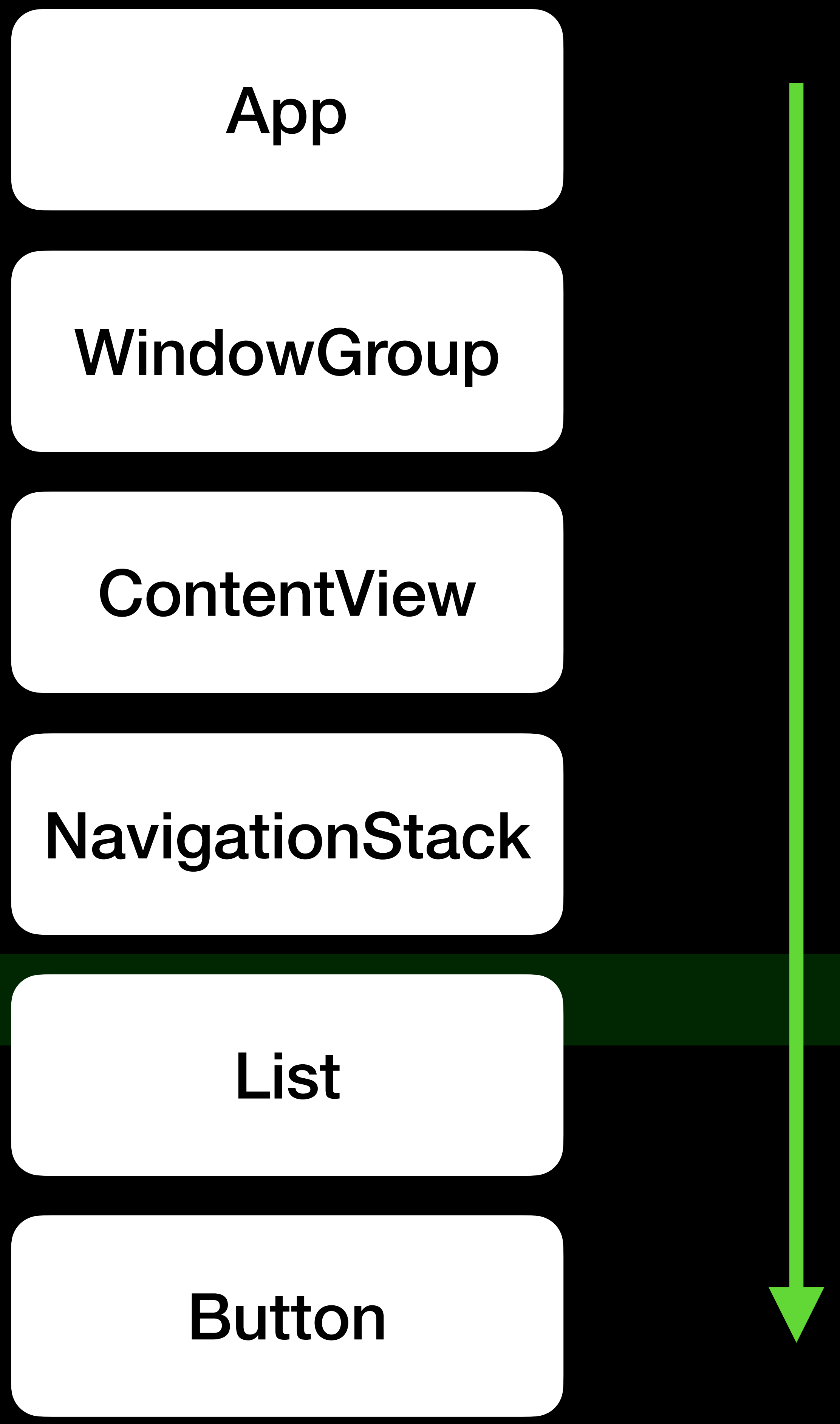
Environment

```
enum AppTheme {
    case neon, shine, dance

    var foregroundColor: Color {…}
    var backgroundColor: Color {…}
}

extension AppTheme: EnvironmentKey {
    static var defaultValue: AppTheme {
        AppTheme.neon
    }
}

extension EnvironmentValues {
    var appTheme: AppTheme {
        get { self[AppTheme.self] }
        set { self[AppTheme.self] = newValue }
    }
}
```



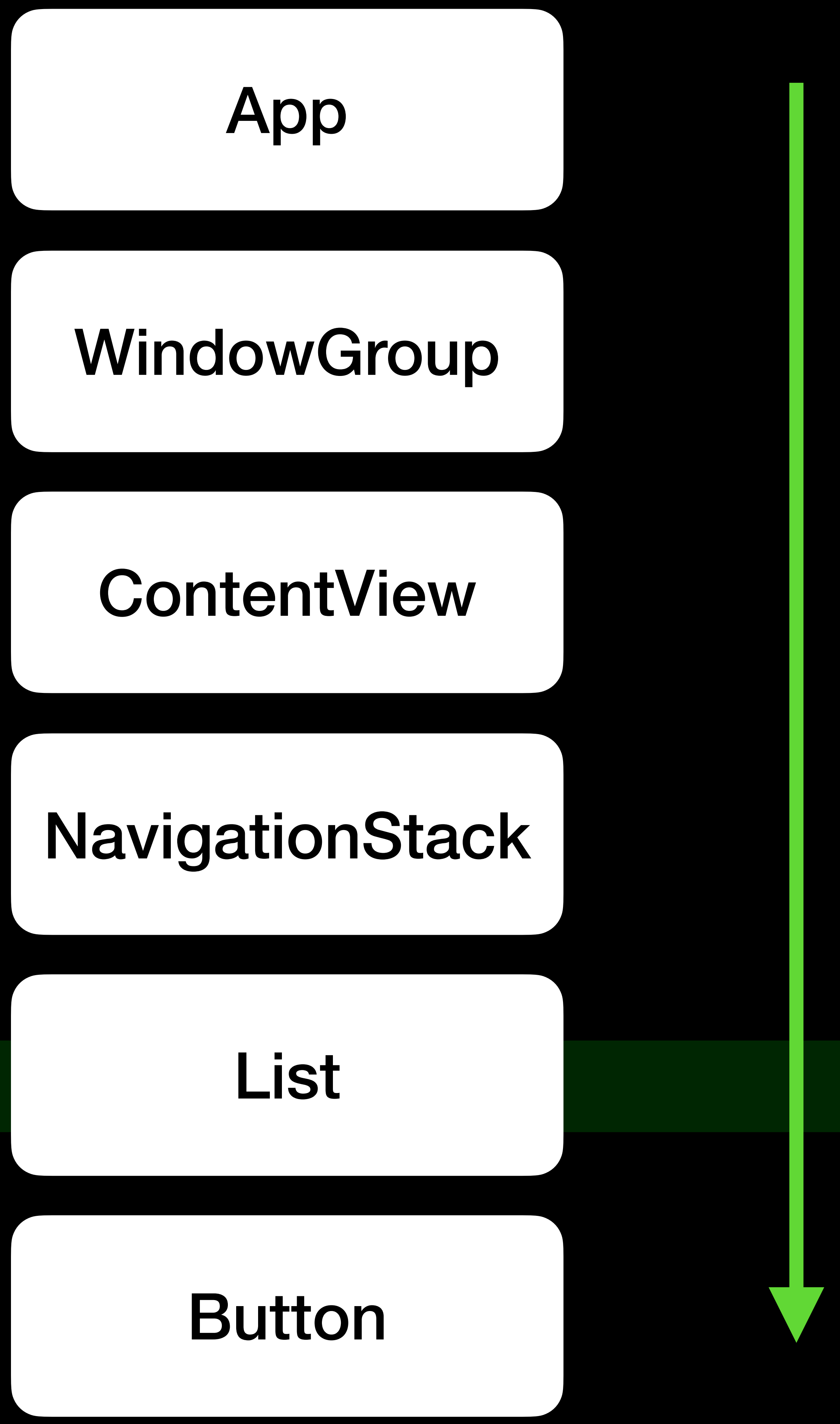
Environment

```
enum AppTheme {
    case neon, shine, dance

    var foregroundColor: Color {…}
    var backgroundColor: Color {…}
}

extension AppTheme: EnvironmentKey {
    static var defaultValue: AppTheme {
        AppTheme.neon
    }
}

extension EnvironmentValues {
    var appTheme: AppTheme {
        get { self[AppTheme.self] }
        set { self[AppTheme.self] = newValue }
    }
}
```



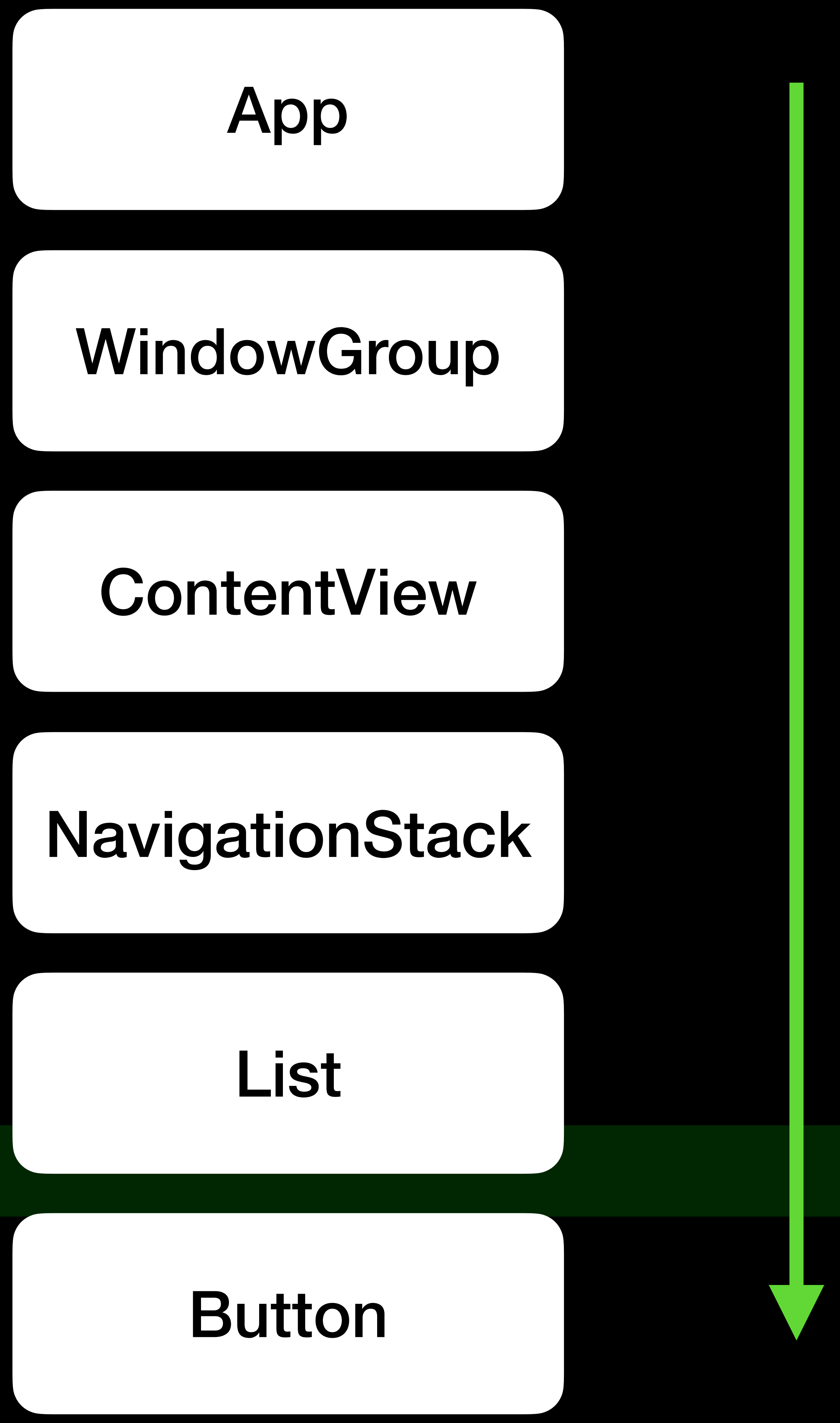
Environment

```
enum AppTheme {
    case neon, shine, dance

    var foregroundColor: Color {…}
    var backgroundColor: Color {…}
}

extension AppTheme: EnvironmentKey {
    static var defaultValue: AppTheme {
        AppTheme.neon
    }
}

extension EnvironmentValues {
    var appTheme: AppTheme {
        get { self[AppTheme.self] }
        set { self[AppTheme.self] = newValue }
    }
}
```



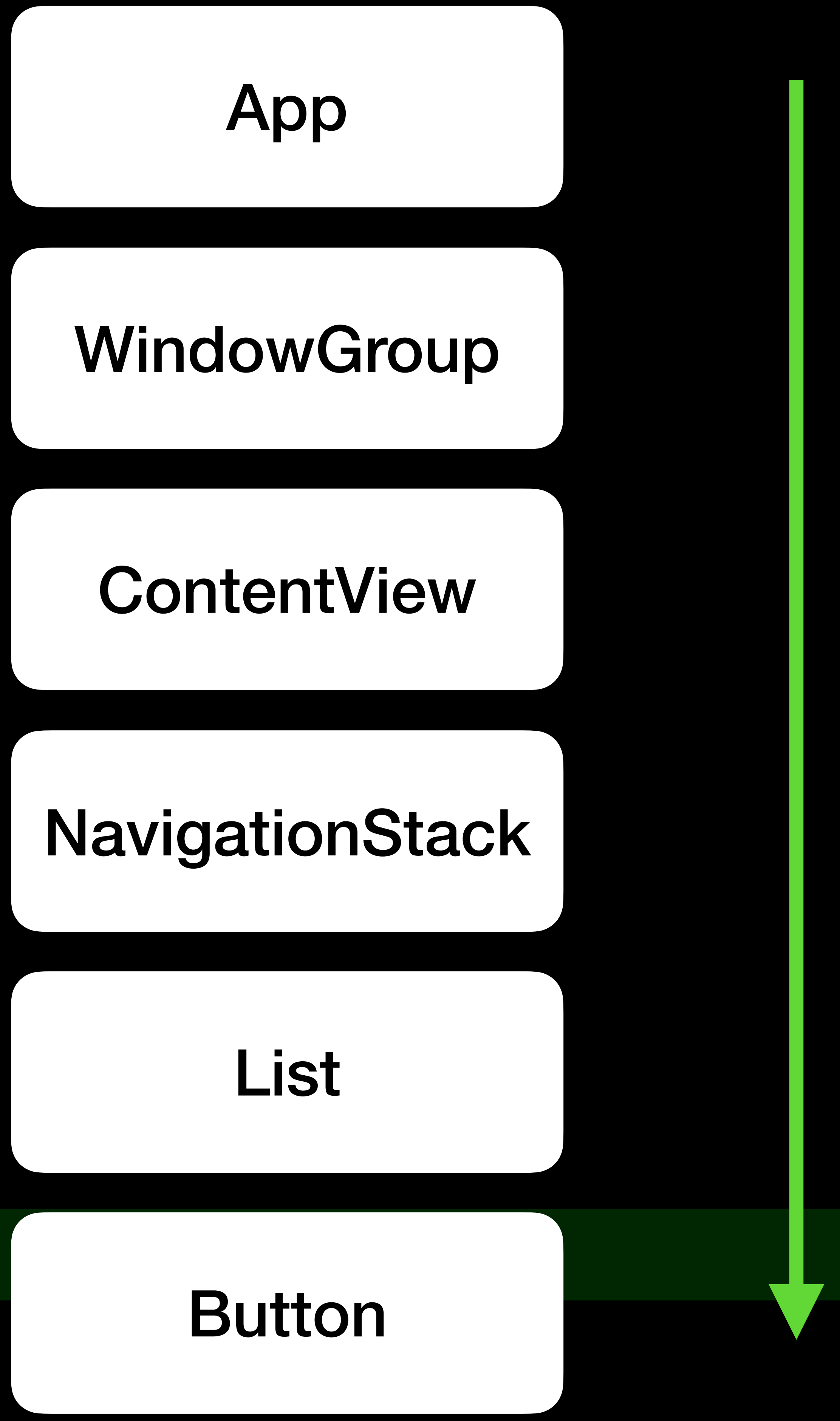
Environment

```
enum AppTheme {
    case neon, shine, dance

    var foregroundColor: Color {…}
    var backgroundColor: Color {…}
}

extension AppTheme: EnvironmentKey {
    static var defaultValue: AppTheme {
        AppTheme.neon
    }
}

extension EnvironmentValues {
    var appTheme: AppTheme {
        get { self[AppTheme.self] }
        set { self[AppTheme.self] = newValue }
    }
}
```



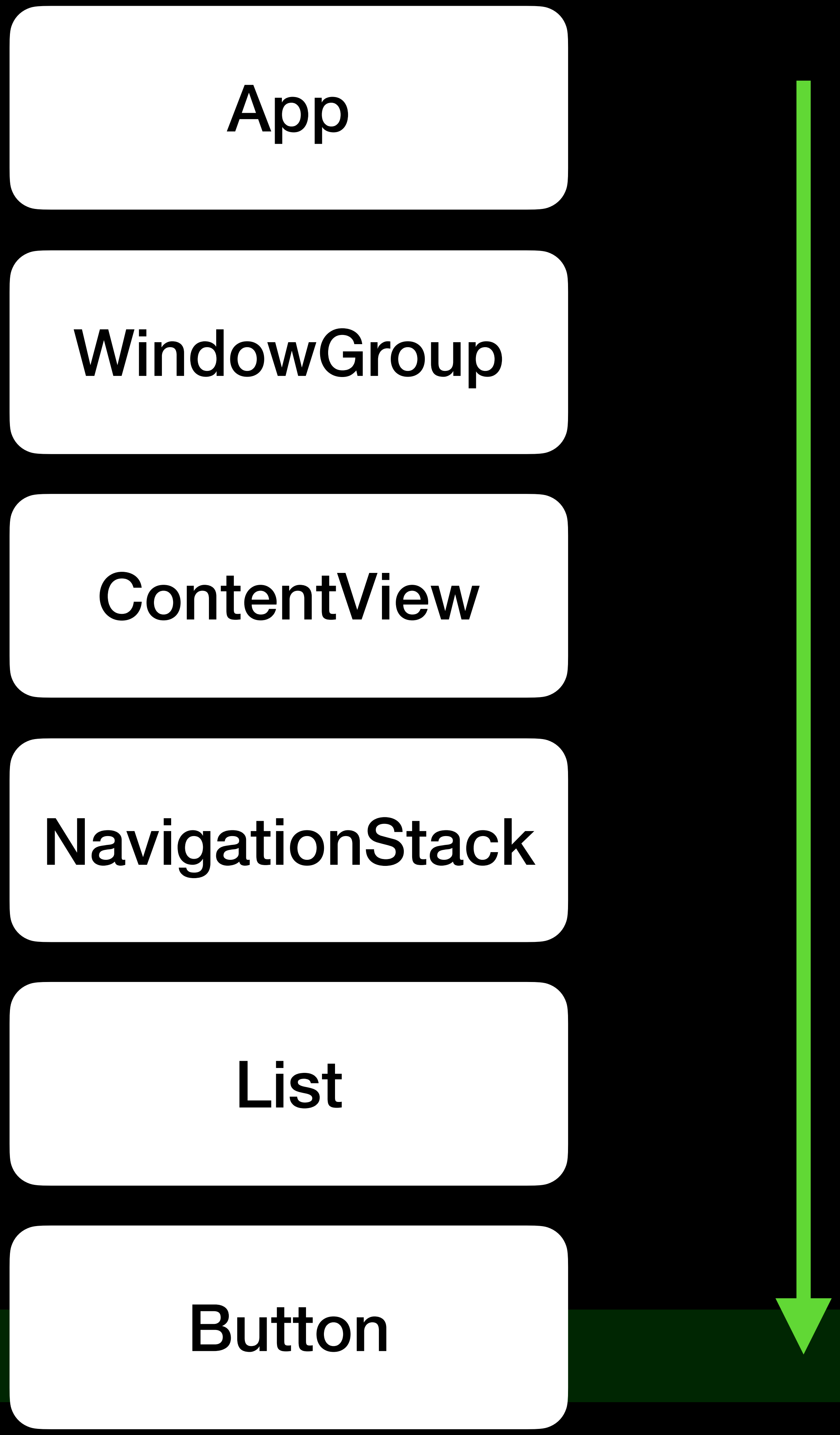
Environment

```swift
enum AppTheme {
    case neon, shine, dance

    var foregroundColor: Color {…}
    var backgroundColor: Color {…}
}

extension AppTheme: EnvironmentKey {
    static var defaultValue: AppTheme {
        AppTheme.neon
    }
}

extension EnvironmentValues {
    var appTheme: AppTheme {
        get { self[AppTheme.self] }
        set { self[AppTheme.self] = newValue }
    }
}
```



Environment

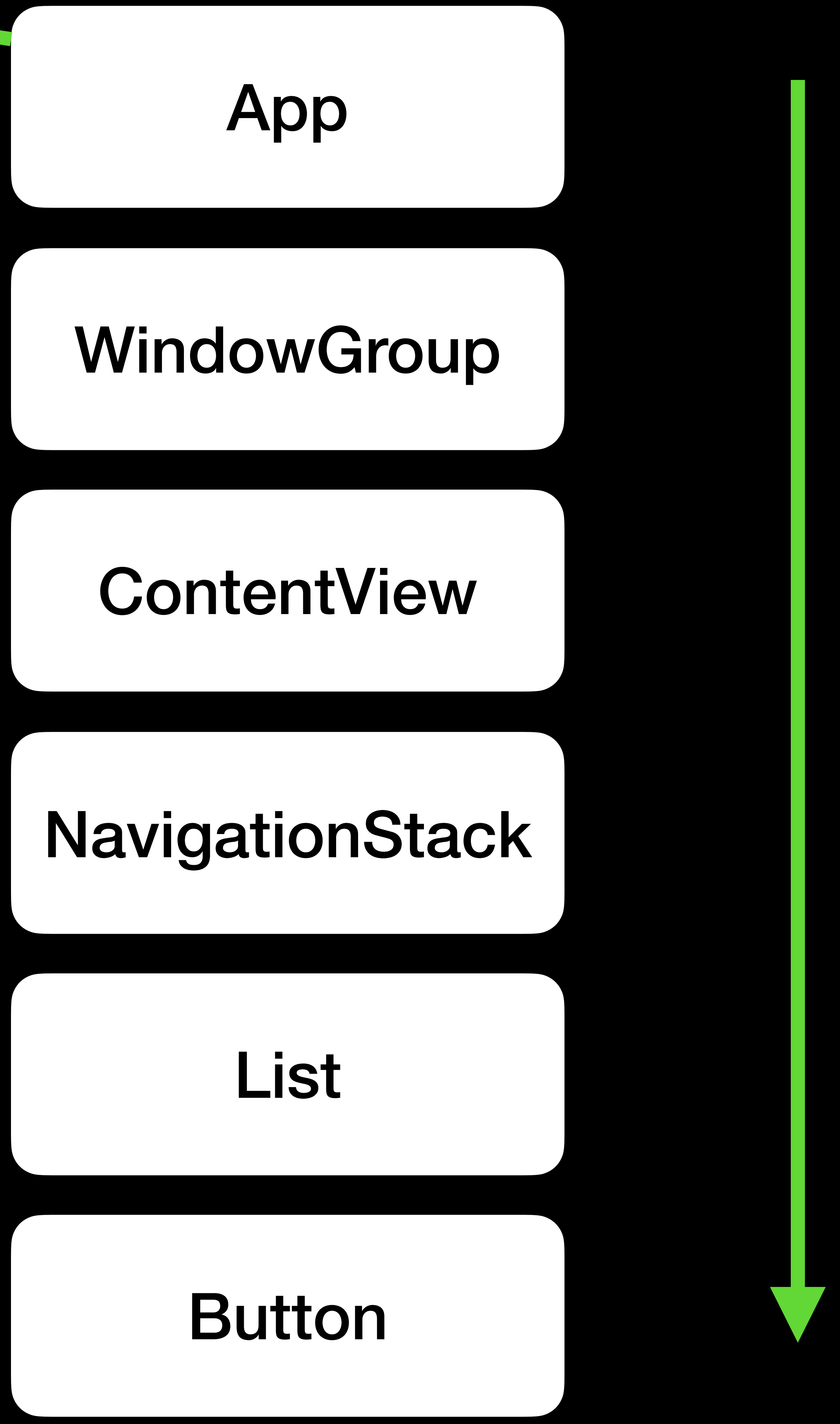
```
enum AppTheme {
    case neon, shine, dance

    var foregroundColor: Color {…}
    var backgroundColor: Color {…}
}

extension AppTheme: EnvironmentKey {
    static var defaultValue: AppTheme {
        AppTheme.neon
    }
}

extension EnvironmentValues {
    var appTheme: AppTheme {…}
}

@Environment(\.appTheme) var theme
```



Environment

```
@Environment(\.appTheme) var theme
```

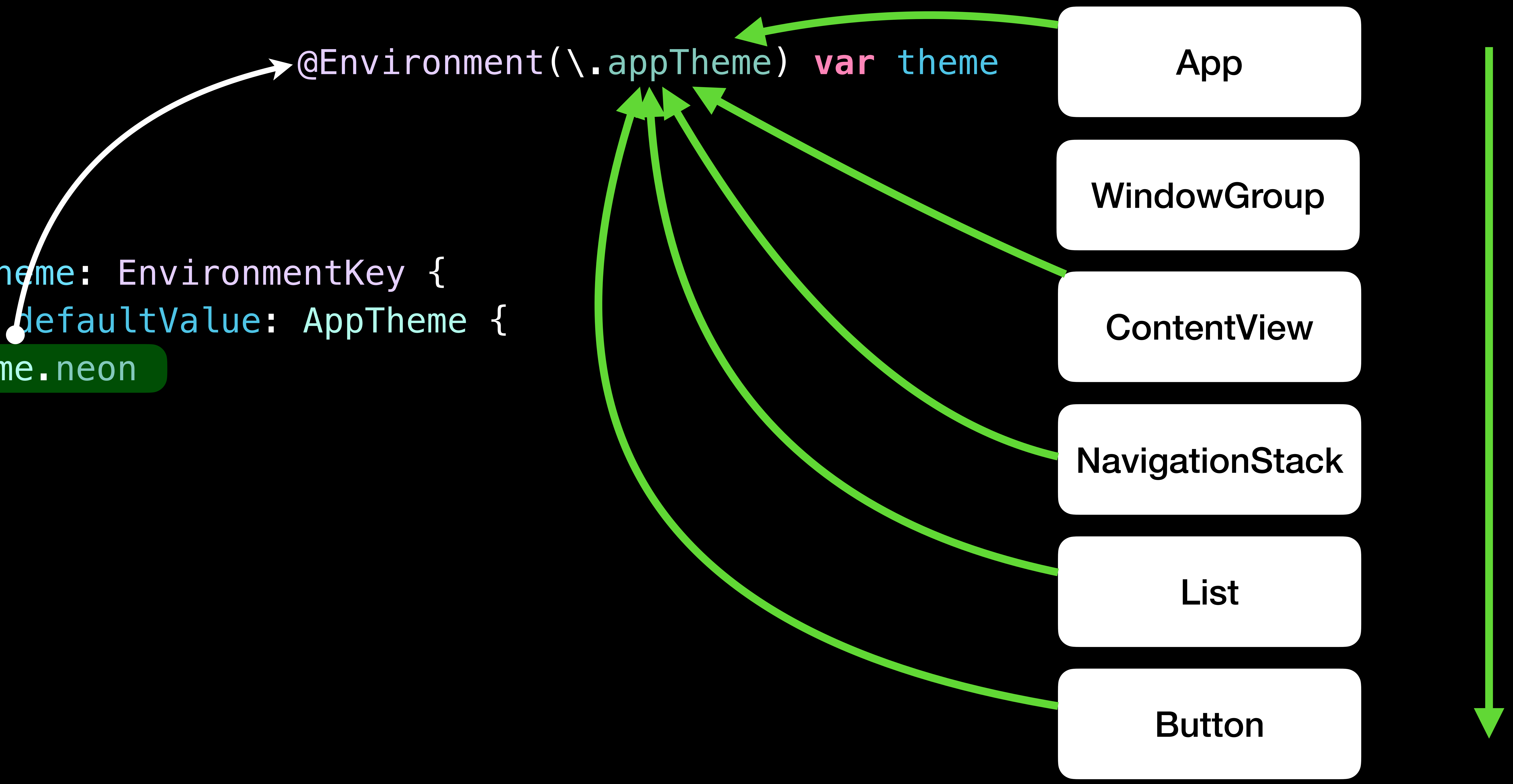
```
extension AppTheme: EnvironmentKey {
    static var defaultValue: AppTheme {
        AppTheme.neon
    }
}
```



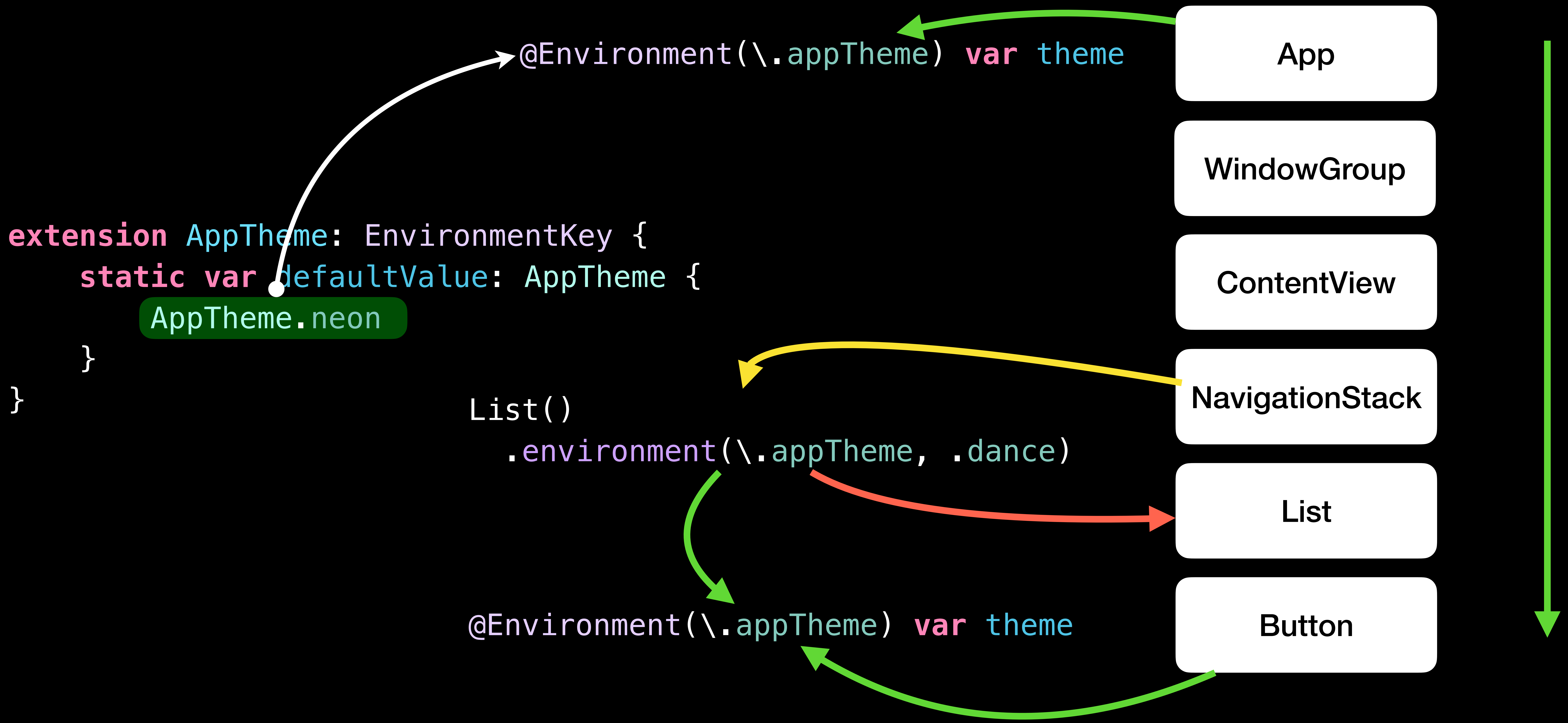
Environment

```swift
@Environment(\.appTheme) var theme
```

```swift
extension AppTheme: EnvironmentKey {
    static var defaultValue: AppTheme {
        AppTheme.neon
    }
}
```



Environment



```
@Environment(\.appTheme) var theme
```

```
extension AppTheme: EnvironmentKey {
    static var defaultValue: AppTheme {
        AppTheme.neon
    }
}
```

```
List()
  .environment(\.appTheme, .dance)
```

```
@Environment(\.appTheme) var theme
```

Today

some SwiftUI

- View
- ViewModifier
- Environment

Let’s Study in

Ratnesh Jain

Reference

- A Day in a Life of SwiftUI View
  - By Chris Eidhof https://chris.eidhof.nl/presentations/day-in-the-life/

Reference

- Building Reusable SwiftUI Components
  - by Peter Friese (from Firebase) https://www.youtube.com/watch?v=YjSxPxT5V40

Thank You 🙇

Ratnesh Jain

Sr. iOS Engineer

- Open Source Contribution
  - @ratnesh-jain/swiftui-fetching-view/
  - @ratnesh-jain/swift-image-downloader
  - @ratnesh-jain/AssetPluginLibrary
  - @ratnesh-jain/swiftui-status-reporting
- Blog posts
  - ratnesh-jain.github.io
- Social Media
  - https://linktr.ee/ratneshjain1993

Ratnesh Jain