

In one line: A **representable** struct wraps a long-lived UIKit object: **make** runs **once per identity**, **update** **on every change**, a **Coordinator** carries callbacks back. The other way, **UIHostingController** is a real view controller showing a SwiftUI view.

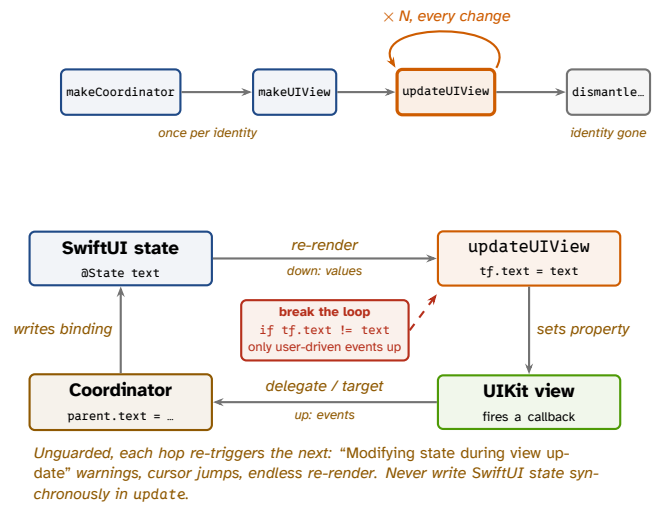
How it works

- UIViewRepresentable** (UIViewControllerRepresentable when you need VC lifecycle — pickers, SFSafariViewController). Order: `makeCoordinator()` → `makeUIView(context:) → updateUIView(_ context:)` × N → `static dismantleUIView(_ coordinator:)` when the identity goes away.
- make** = one-time setup (view, delegates, targets, gestures, observers). **update** = **idempotent sync** of SwiftUI values onto the view, only when they differ — it runs whenever inputs or anything it reads change.
- Context:** coordinator, environment (apply `colorScheme`, `isEnabled...`), `transaction(.animation != nil ⇒ animate the UIKit change)`.
- Coordinator** — NSObject subclass: delegate, data source, @objc target. *Down:* properties / @Binding applied in update; *up:* the coordinator writes the binding or calls a closure. Made once, but the struct is new each update ⇒ refresh `context.coordinator.parent = self`.
- Sizing** (16): `sizeThatFits(_ uiView: context:) → CGSize?` answers the `ProposedViewSize`; nil = default (intrinsic size + hugging/compression priorities, which is all you had before 16).
- UIHostingController(rootView:)** — push, present, or embed as a **child VC**: `addChild` → `add host.view + constraints` → `didMove(toParent:)`. `sizingOptions` (16): `.intrinsicContentSize / .preferredContentSize`. Update via an observable model (or `host.rootView = ...`).
- UIHostingConfiguration** (16) — SwiftUI in a collection/table cell: `cell.contentConfiguration = UIHostingConfiguration { Row(item) }; self-sizing, .margins, no host VC per cell`.
- Environment:** a hosting controller starts a *new* environment — traits (colour scheme, Dynamic Type) flow in, your `.environment(model) / .environmentObject` do **not**: re-inject on `rootView`. `UITraitBridgedEnvironmentKey` (17) bridges custom traits.

Example — two-way text field

```
struct Field: UIViewRepresentable {
    @Binding var text: String
    func makeCoordinator() -> Coordinator { Coordinator(self) }
    func makeUIView(context: Context) -> UITextField {
        let tf = UITextField() // once per identity
        tf.delegate = context.coordinator; return tf
    }
    func updateUIView(_ tf: UITextField, context: Context) {
        context.coordinator.parent = self // fresh binding
        if tf.text != text { tf.text = text } // guard: no echo
    }
    final class Coordinator: NSObject, UITextFieldDelegate {
        var parent: Field
        init(_ p: Field) { parent = p }
        func textFieldDidChangeSelection(_ tf: UITextField) {
            parent.text = tf.text ?? "" } } // up: UIKit -> binding
```

Picture — lifecycle and the update loop



The other direction — SwiftUI as a child VC inside a UIViewController:

```
let host = UIHostingController(rootView: Card().environment(model))
host.sizingOptions = .intrinsicContentSize // iOS 16
addChild(host); view.addSubview(host.view) // 1. contain
host.view.translatesAutoresizingMaskIntoConstraints = false
NSLayoutConstraint.activate([ /* pin edges */ ]) // 2. lay out
host.didMove(toParent: self) // 3. finish
```

Interview traps

- ! Setup in `updateUIView` (adding gestures, subviews, observers) — duplicated on every update. `make` sets up, `update` syncs.
- ! Representable behind a changing `.id` or unstable `ForEach` id — `makeUIView` runs again: scroll position, first responder, player state lost.
- ! `UIHostingController` in a cell — wrong/zero height; use `UIHostingConfiguration`, or `sizingOptions = .intrinsicContentSize + pinned edges`.
- ! Retain cycles: a closure stored on a long-lived UIKit object capturing the coordinator (and it the view). UIKit delegate properties are weak, so plain delegation is safe.

Remember

“Make once, update often, coordinate back.” update must be safe to call 100 times in a row.

Likely questions

- Why a Coordinator? — structs cannot be delegates / @objc targets; it is the reference type.
- make vs update? — once per identity vs every change; update is idempotent.
- Size a wrapped UILabel? — `sizeThatFits` (16) or hugging priorities.
- SwiftUI inside UIKit? — `UIHostingController` (child VC) / `UIHostingConfiguration` (cells).
- Match a SwiftUI animation? — `context.transaction.animation ⇒ UIView.animate`.