

In one line: A view is a cheap **struct** rebuilt constantly; its state lives **outside** it in SwiftUI's storage, keyed by **view identity**. The only question each wrapper answers is **who owns the truth**: this view (@State, @StateObject) or someone else (@Binding, @ObservedObject, @EnvironmentObject).

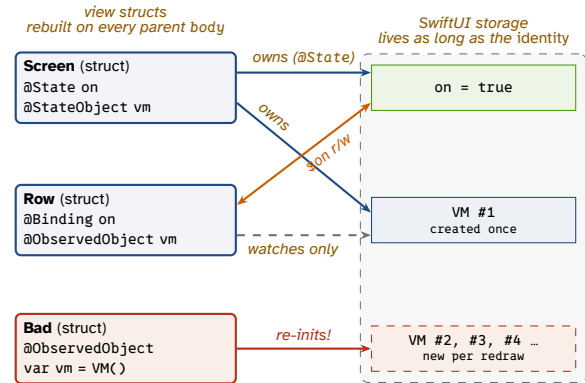
How it works

- View** is a protocol: associatedtype Body: View + @ViewBuilder var body: Self.Body { get }. some View is an **opaque result type**: one concrete type (e.g. VStack<TupleView<(Text, Button<Text>)>>), chosen by the *implementation*, known to the compiler, hidden from the reader. Not a box, not dynamic — unlike AnyView, which erases the type and hurts diffing.
- @State** — view-**owned** source of truth for a *value*; storage lives in SwiftUI, survives body re-runs; the initial value is used *once*. Mark it private. \$x (the projectedValue) is a Binding.
- @Binding** — **not** storage: a two-way get/set *reference* to truth owned elsewhere. Child writes → parent's @State changes → both re-render. Passed as \$name; .constant(x) in previews.
- @StateObject** — view **creates and owns** an ObservableObject; the @autoclosure initialiser runs **once per identity**, the instance survives struct re-creation.
- @ObservedObject** — object is **injected**; view only subscribes, does *not* control lifetime. Initialised inline, it is re-created on every parent redraw.
- @EnvironmentObject** — injected by an ancestor's .environmentObject(obj), looked up by *type* at run time; missing = crash. @Environment(\.colorScheme) reads *values* (\.dismiss, \.scenePhase).
- Notification:** @Published fires objectWillChange *before* the set (whole object). **iOS 17 @Observable** (macro, Observation) tracks **each property body read** — see the table.
- @ViewBuilder** — a *result builder*: buildBlock packs children into TupleView; if/else → _ConditionalContent (buildEither); if alone → Optional (buildOptional). That is why body may branch but a plain helper func → some View may not (all returns must be one type) — mark it @ViewBuilder.
- Identity:** *structural* (position in the tree, which if branch) or *explicit* (.id(_), ForEach ids). New identity = state destroyed, onAppear again, transition played.
- body re-runs** when something it *read* changes: a @State / @Binding value, an observed object publishes (or an @Observable property it read), an @Environment value, or the parent passes different inputs. Then SwiftUI diffs.

Example

```
final class VM: ObservableObject { @Published var n = 0 }
struct Screen: View {
    @StateObject private var vm = VM() // made ONCE, owned
    @State private var on = false      // value, owned
    var body: some View {              // one concrete type
        VStack { Toggle("On", isOn: $on) // $on : Binding<Bool>
            Row(vm: vm, on: $on) }       // hand truth down
    }
}
struct Row: View {
    @ObservedObject var vm: VM // injected, NOT owned
    @Binding var on: Bool      // someone else's @State
    var body: some View { Button("+") { vm.n += 1; on = true } }
}
```

Picture — structs die, storage stays



role	ObservableObject	iOS 17 @Observable
owns it	@StateObject	@State
passed in	@ObservedObject	plain let / var
needs \$x.prop	@ObservedObject	@Bindable
injected	@EnvironmentObject	@Environment(T.self)
provide	.environmentObject(o)	.environment(o)
publishes	@Published props	every stored prop (macro)
re-renders	every observer, any change	only views that read it

Interview traps

- ! @ObservedObject var vm = VM()** — created but not owned: each parent redraw allocates a fresh VM, state resets, *no warning*. Creator: @StateObject.
- ! “some View = any view / like AnyView”** — no: some is *one fixed type*; returning Text in one branch and Image in another fails without @ViewBuilder.
- ! “@Binding is a copy”** — no: it stores nothing; writes go to the owner. Two @State copies of one fact = two truths that drift.
- ! @State holding a plain class:** inner mutation leaves the reference equal → no re-render (fine only for an @Observable class).
- ! if c { MyView() } else { MyView() }** = two identities → state resets on toggle. .id(value) changing often silently nukes state.
- ! Heavy work or network calls in body** — it runs many times; use .task (auto-cancelled on disappear) or .onAppear.

Remember

“Who wrote = VM()? That view writes @StateObject.”

State / StateObject = *mine*; Binding / Observed / Environment = *borrowed*. \$ = a pointer to the truth.

Likely questions

- some View? — opaque: one concrete type, compiler-known, unnamed.
- @StateObject vs @ObservedObject? — owns, made once vs injected.
- @Binding? — read/write reference to an ancestor's state (\$x).
- Why @Observable? — per-property tracking → fewer re-renders.
- When does body run? — a dependency it read changed, or new inputs.