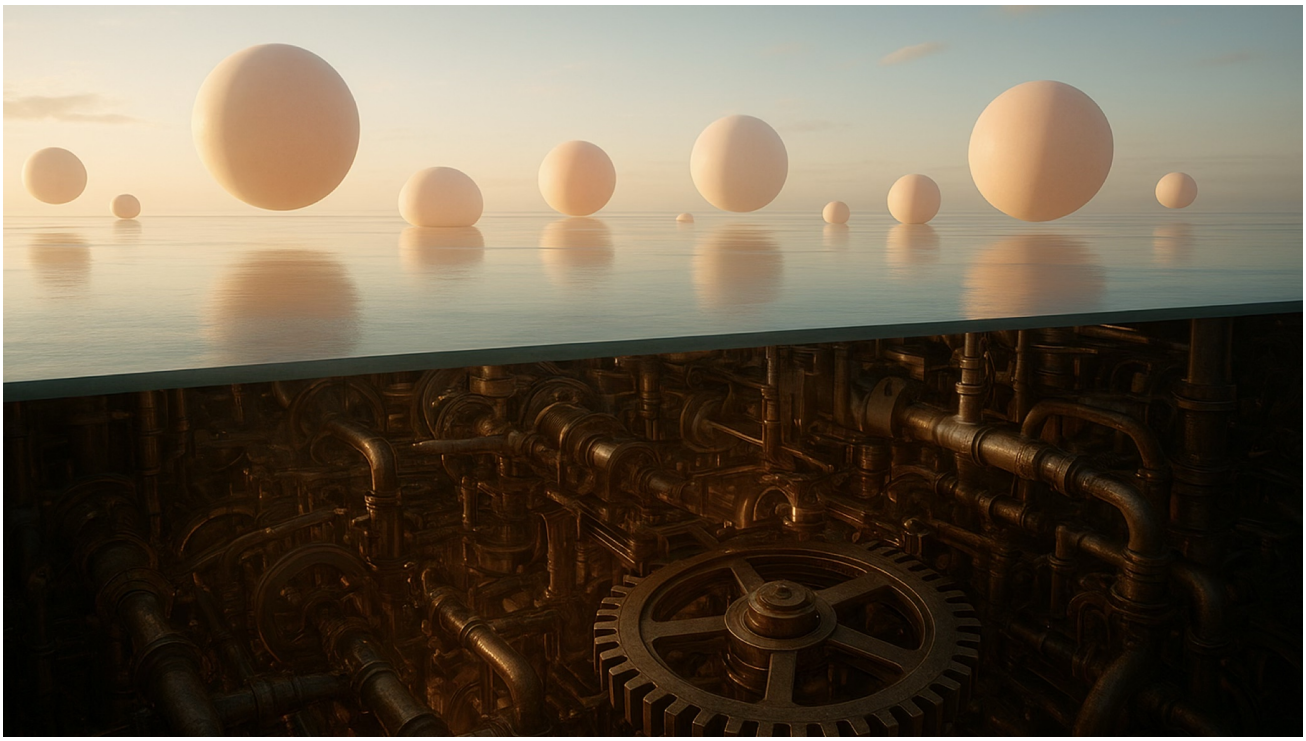

The Morlocks Built SwiftUI

Swift is for the Eloi.

That is the cynical reading. There is a ladder here, and every rung hides more than the last:

> **C**: too dangerous. Pointers, manual memory, undefined behavior.
> **Objective-C**: too dynamic. A runtime that will happily message a deallocated object and crash you in production.
> **Swift**: safe by default, but the pointers are still there if you go looking.
> **Swift 6 strict concurrency**: the compiler now walks you across every actor boundary so you never have to reason about a data race yourself.
> **SwiftUI**: declarative. Stop thinking about the render loop.
> **SwiftData**: persistence without ever learning what an object graph is.

Most developers shipping apps today have never typed `CALayer`, because SwiftUI just works. Until it doesn't.

It is tempting to call this condescension: the toolmakers kept the real tools and handed us the Fisher-Price version. It is a satisfying story. It is also wrong.

The man who built both ends

Core Animation was created by an Apple engineer named John Harper, and it is older than the device that made it famous. He built it under the codename "LayerKit", and Apple was already showing it running on the Mac at WWDC 2006, months before the iPhone was announced. The foundational patent, US 8,130,226 ("Framework for Graphics Animation and Compositing Operations"), traces its priority to an application filed on August 4, 2006, and its own text still notes that "Core Animation" is just the "LayerKit" of that earlier filing. It reached the public in 2007: in the first iPhone OS, and later that year on the Mac in OS X 10.5 Leopard.

He left for Facebook in 2014. He came back to Apple and co-built SwiftUI: at WWDC 2019 he was

on stage in "Building Custom Views with SwiftUI", explaining how the view system actually works.

The person who built the rendering foundation that shipped in 2007 was, twelve years later, on the team building the declarative layer that sits on top of it. The same hands on both ends of the abstraction.

A Morlock in the daylight.

SwiftUI is not what happens when the serious engineers go build silicon and leave the toy framework to someone junior. It is what happens when the people who understand the foundation best decide what you should usually not have to think about. The implicit animation system, the idea that a state change should animate by default, that you declare what the UI is and let the framework diff and interpolate, is not a dumbing-down of Core Animation. It is Core Animation's own philosophy, made the default instead of the opt-in.

The foundations never left

Of all of Apple's low-level frameworks, almost everything named `Core` is written in C and meant to be called from anywhere: Core Graphics, Core Text, Core Audio, Core Foundation. Close to the metal, language-agnostic, no object model in the way.

Two are different. Core Animation and Core Data are written in Objective-C, object-model frameworks built around a tree of objects rather than a bag of C functions: Core Animation with its layer tree, Core Data with its managed-object graph. They were designed object-first because the problems they solve are object-shaped.

Those two still do not have Swift-native replacements. SwiftUI does not replace Core Animation. It sits on top of it. Your `.animation()` modifier drives a `CAAnimation`. Your view hierarchy resolves to a layer tree and a render server that has been doing this job, more or less unchanged, since 2007. SwiftData does the same thing to Core Data. The friendly declarative surface is new. The foundation underneath is the same machine it always was.

Core Animation did not need to change, because it was right the first time. The layer tree, the split between the model tree and the presentation tree, the separate render process, the implicit transactions: all still there, all still doing the work. What SwiftUI replaced was not Core Animation. It was UIKit's interface to Core Animation: the manual animation blocks, the animator APIs, the layout system that had accreted cruft for a decade. The part that aged badly was how we talked to the machine, not the machine.

So what is the truth we can't handle?

It is not that we are too stupid for pointers. Plenty of us could learn the layer tree in a weekend.

The truth is quieter: the foundation is still there, you have just stopped being able to see it, and that is a debt, not a gift. It comes due on the specific day SwiftUI does something you cannot explain. A frame that drops. An animation that fights itself. A view that will not redraw when every rule you know says it should. On that day the abstraction has failed, and the only people who can fix it are the ones who know what is underneath: the layer tree, implicit versus explicit animation, transforms, compositing, the render server.

Apple did not give us Swift and SwiftUI because we are too soft for the real thing. They gave them to us because the people who built the real thing learned, over fifteen years, which parts you genuinely should not have to touch most days. That is not contempt. It is the opposite. It is mastery deciding the default.

A good enough abstraction does not insult you. It just makes you comfortable on the surface, and

helpless the day the machinery underneath asks whether you were paying attention. The layer tree is a weekend's study. Learn it before the day you need it, because the framework will not tell you, in advance, which kind of day today is.