Workshop Resources

- Repo Location: **https://github.com/C1-SoftwareEngineeringSummit/NewsfeedUI**
- Get an API key here: **https://newsapi.org/docs/get-started**

Intro to Swift & iOS Development

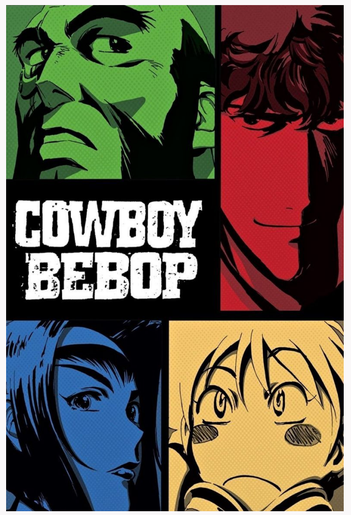
Chris Longe
Connor Nelson
Merlin Levine

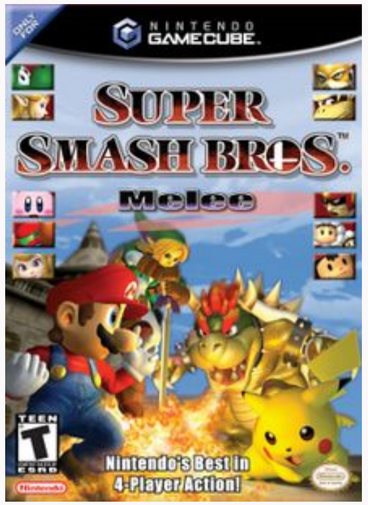
About Chris

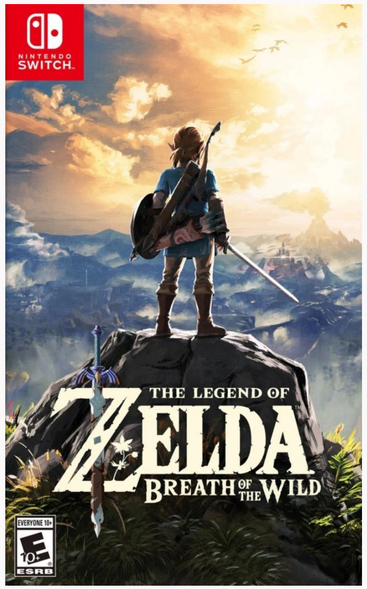
- iOS Tech Lead at Capital One
- University of Texas at Austin 👨‍🎓
- C1 TDP graduate 👨‍🎓
- Love watching/playing futbol ⚽
- What a save…
- Dad of a two year old toddler…

About Connor

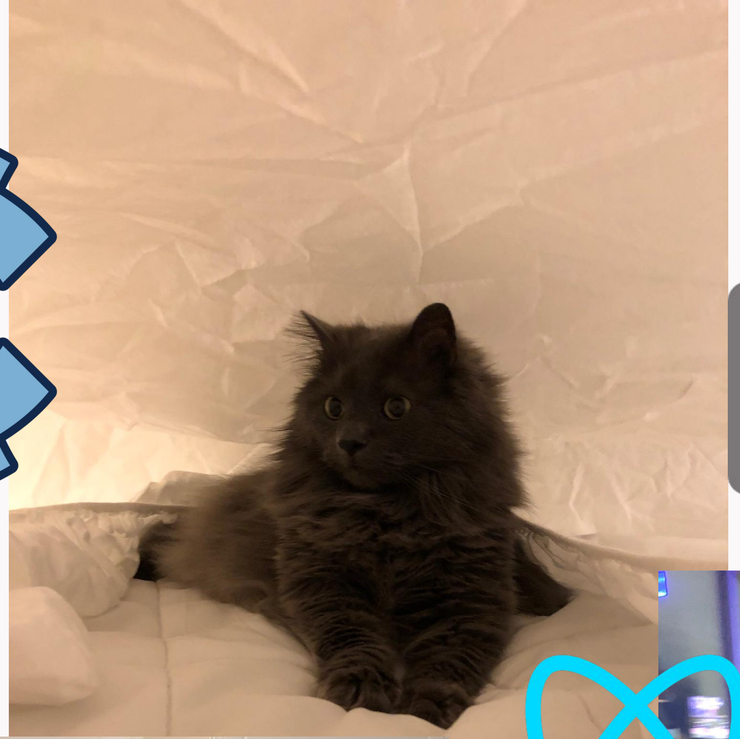
- iOS Developer at Capital One
- TDP graduate
- University of Maryland
- I love anime and video games

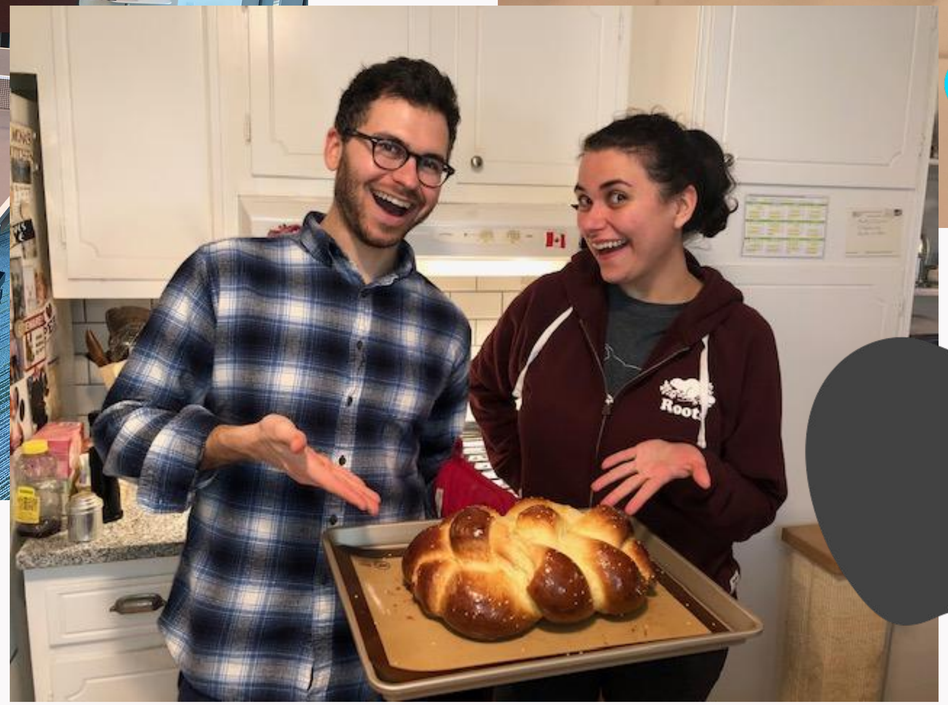


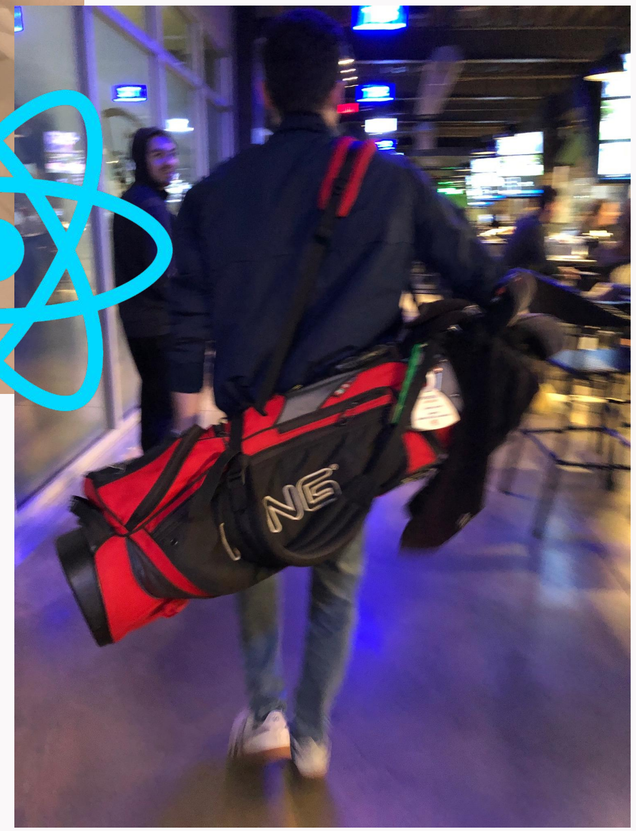




About Merlin







Agenda

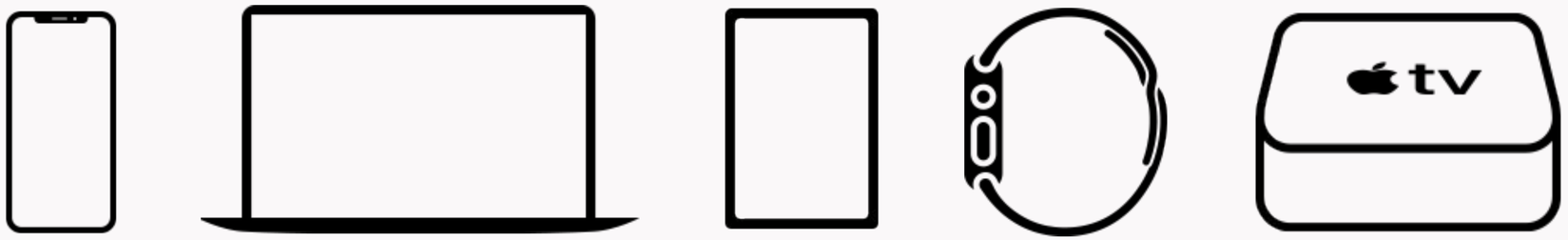
- Brief History
- Intro to the Swift programming language
- Intro to SwiftUI
- ~5-10 minute break
- Building a news feed app in SwiftUI with Xcode
- Quiz! (If time permits)

Prerequisites

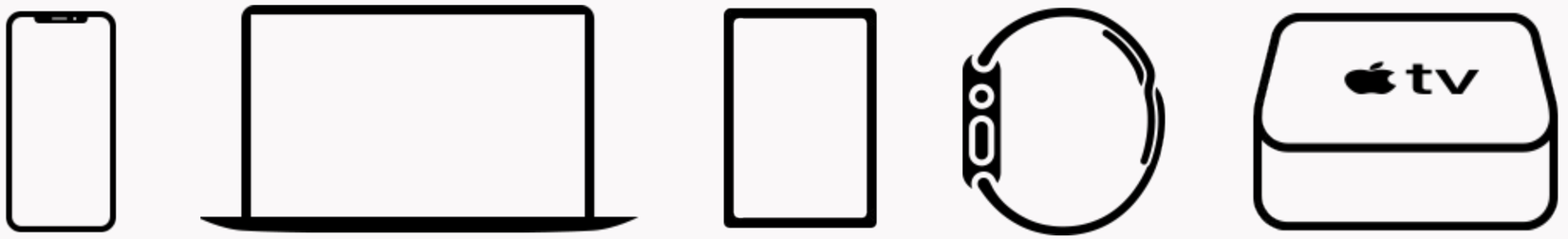
- macOS 10.15.4+ (Catalina or Big Sur)
- Xcode 12.4+
- Swift 5.3

Swift

Swift is a powerful and intuitive programming language for macOS, iOS, watchOS, tvOS and beyond.

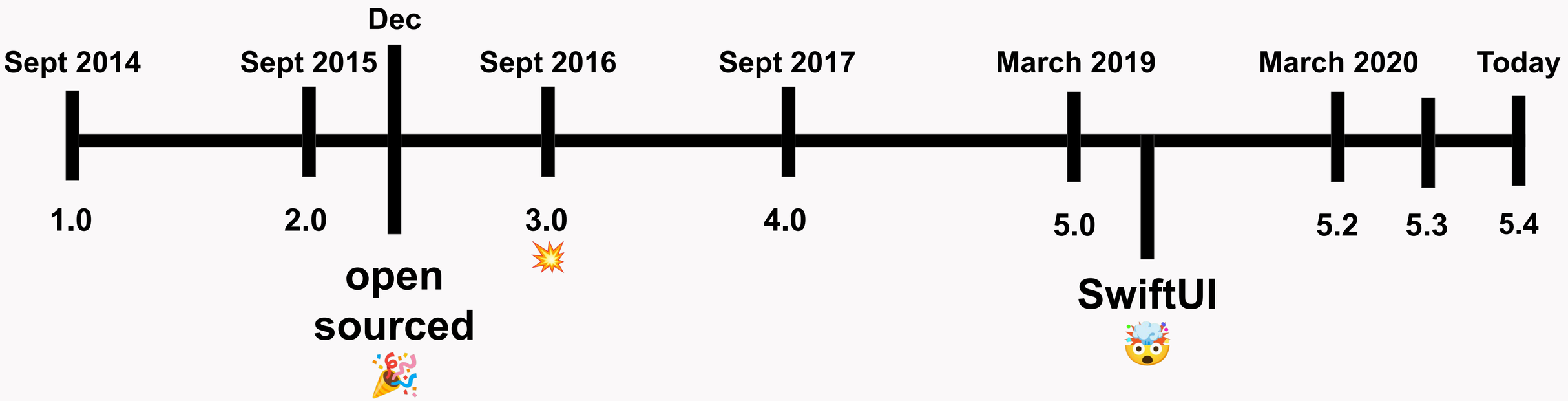


SwiftUI is a brand new user interface toolkit that lets us design apps in a **declarative** way.



Swift Timeline

Evolving Quickly!

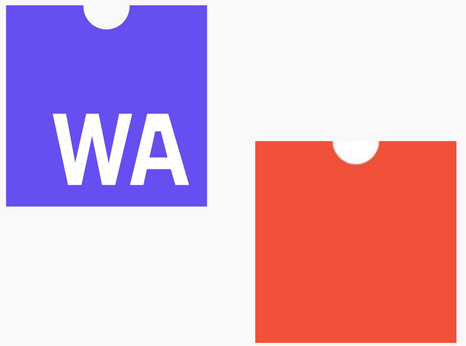


Swift also runs on non-Apple platforms!

- Server Side Swift!
  - Build websites or API's
  - Ubuntu, CentOS, & **Amazon Linux**
- Swift AWS Lambda Runtime (Serverless)
- Swift on Raspberry Pi or Arduino
- Swift for Machine Learning
  - TensorFlow

More places you can use Swift

- swiftwasm.org
  - SwiftWasm compiles your Swift code to WebAssembly!
- Swift 5.3 officially supports Microsoft Windows!



Writing Swift

Variables

```
var bool1: Bool = true          // var defines a variable
let bool2: Bool = false         // let defines a constant

var str1: String = “string”     // You can define your variable’s type
var str2 = “string”             // Swift can also use type inference

var arr1: [String] = []         // An empty array containing Strings
var arr2 = [String]()           // Also an empty array of Strings
var arr3 = [“s1”, “s2”]         // An array with initial values
```

Optionals

```
var str1: String? = “maybe”     // This optional contains a value
var str2: String? = nil         // This one does not

if let unwrappedStr = str1 {    // An optional binding safely
    print(unwrappedStr)         // checks for a value before proceeding
}

var str3 = str2 ?? “default”    // Nil-coalescing: str3 == “default”
var str4: String = str2!        // This will crash if str2 == nil
```

Functions

```
func functionName(argLabel paramName: Type, ...) -> ReturnType {
    ...
}

func greet(_ name: String, from town: String) -> String {
    return "Hello " + name + " from " + town
}

greet("SpongeBob", from: "Bikini Bottom")   // Calling the function
```

Closures

```
// Closures are self-contained, executable blocks of code

let customGreeting: (String) -> (String) = { name in

    return “Hello “ + name

}


print(customGreeting("SpongeBob"))    // “Hello SpongeBob”
```

Classes and Structs

```
// Classes are reference types
class Person {
    var name: String
    init(name: String) {
        self.name = name
    }
}

var p1 = Person(name: “asdf”)
```

```
// Structs are value types
struct Person {
    var name: String
    init(name: String) {
        self.name = name
    }
}

var p1 = Person(name: “asdf”)
```

Protocols

```
// Like an Interface in Java
protocol Animal {
    var sound: String { get }
}
```

```
struct Person: Animal {
    var name: String
    var sound: String {
        return "I'm \(name)"
    }
    init(name: String) {
        self.name = name
    }
}
```

Dot Syntax Shorthand

```
// Imagine we have this function

func setColor(name color: Color) -> Void { ... }


// We could call it like this

setColor(name: Color.purple)


// But we can also call it like this

setColor(name: .purple)
```

Swift is Value-Oriented

Values 😍

Swift is an aggressively value-oriented language, it uses value types rather than reference types for nearly everything.

- structs, enums, bools, integers, floats, strings, arrays, dictionaries, and sets are all implemented using value semantics
- A **value** type is something that only ever has one unique owner
- In contrast, classes are **reference** types in Swift
- This means that the same object can have multiple owners, and if any of them change the value then it changes everywhere
- Always remember, **structs** pass by **value** and **classes** pass by **reference**

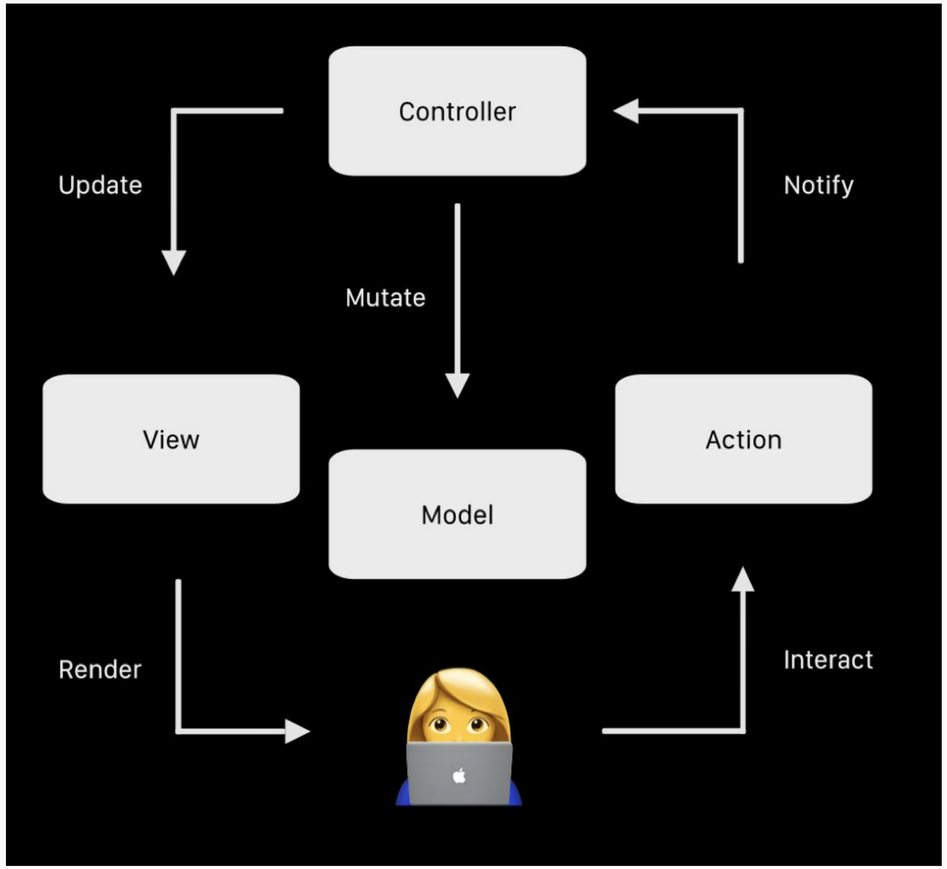
SwiftUI

SwiftUI

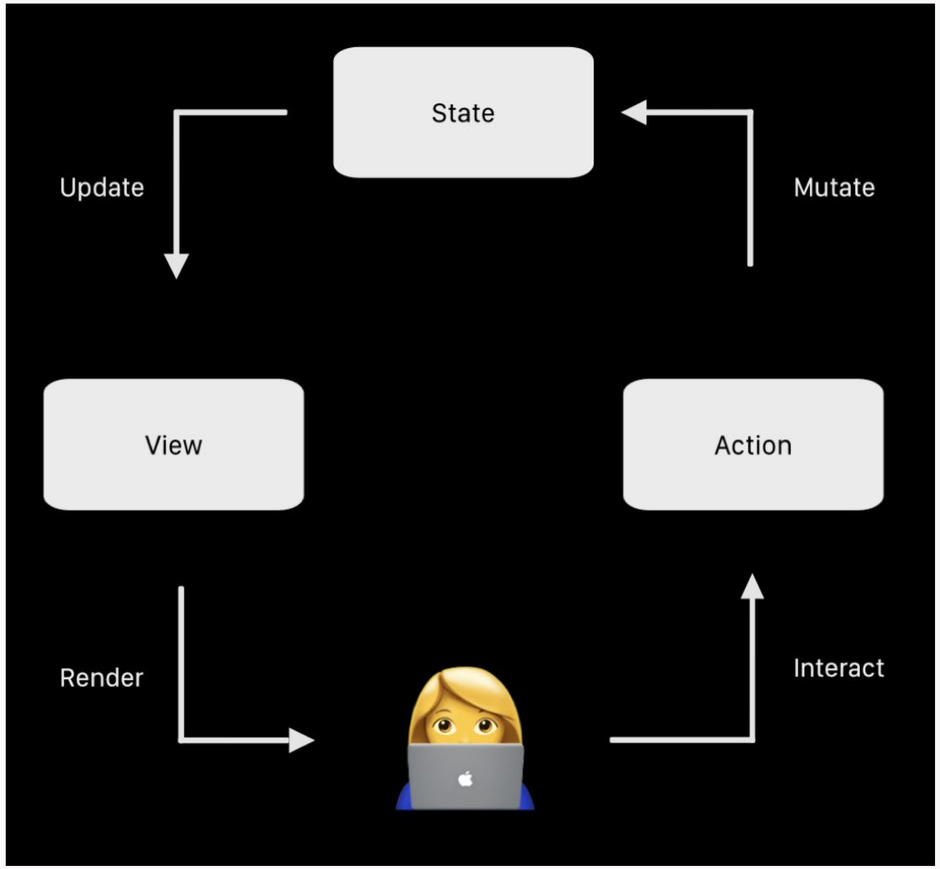
A new paradigm in Apple development

- A new framework using **declarative** syntax to build a user interface
- Automatic support for **Dynamic Type**, **Dark Mode**, **localization**, and **accessibility**
- Design tools, like a **canvas/preview** that stays in sync with your adjacent code
- SwiftUI views are a **function of app state**
- View hierarchies are **rebuilt automatically** when state changes

Data Flow in SwiftUI



Old Paradigm



SwiftUI

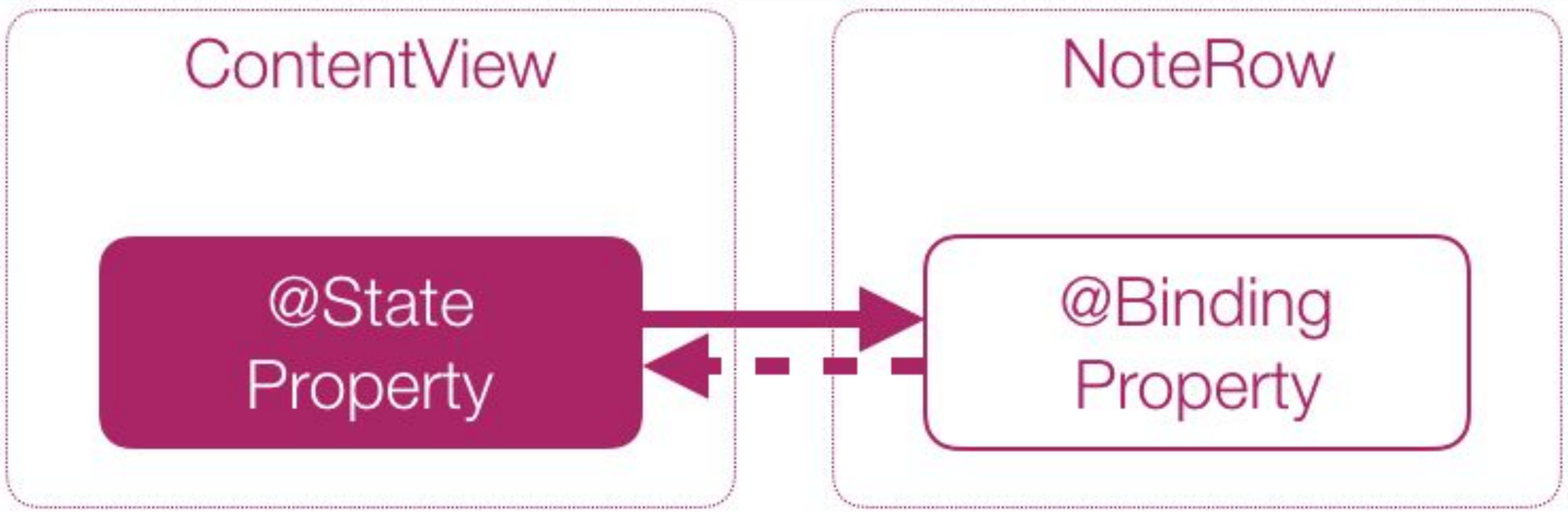
Property Wrappers

- A type that **wraps** a given value to provide **additional functionality**
- **Prepended** to a variable definition; you can apply **only one per variable**
  - `@Capitalized var firstName: String`
- Property wrappers are **transparent**
  - `firstName = “bob”  // firstName is now “Bob”`
  - `var name2 = firstName  // name2 is now “Bob”`

@State and @Binding

- @State causes a property to live in **shared storage managed by SwiftUI**
  - When a @State variable changes, its view will update as well
  - @State variables should be used as the **source of truth** for a view
- @Binding declares a **shared property** where the **source of truth is stored elsewhere**
  - Creates a **2-way connection** between the source of truth and a view
  - We use the `$` prefix on a @State variable to create a @Binding

@State and @Binding



“Home Owner” (@State)
- **Builds the home**
  (initializes the property)

- **Does not require a tenant**
  (@State does not require there to be a @Binding with access)

“Tenant” (@Binding)
- **Does not build the home, but has access**
  (cannot initialize the property, but can access the reference from a @State)
- **Requires a landlord to rent home from**
  (@Binding requires a @State to reference)

@ObservedObject and @Published

- @ObservedObject allows a view to **watch the state of an external object**
  - When an @ObservedObject changes, **SwiftUI will refresh the view**
  - @ObservedObjects are specifically used for external data, or **global data in your app**
- @Published declares that **a property will cause views to refresh**
  - @Published properties **exist inside of @ObservedObjects**
  - When a @Published property changes, **SwiftUI reloads any views that rely on it**

Break (~5-10 min)

Get help from assistants in **#ses-aug21help-xg**

Building an iOS App 🛠️

- Hands-on workshop
- Questions in **#ses-aug21help-xg**

Workshop Resources

- Repo Location: **https://github.com/C1-SoftwareEngineeringSummit/NewsfeedUI**
- Get an API key here: **https://newsapi.org/docs/get-started**