Data Flow in SwiftUI from @State to @Observable

Pragma October, 2023

Daniel H Steinberg

dimsumthinking.com

(1) There's a lot of code

The code isn't important

The ideas are

(2) Combine is not dead

It hasn't been deprecated

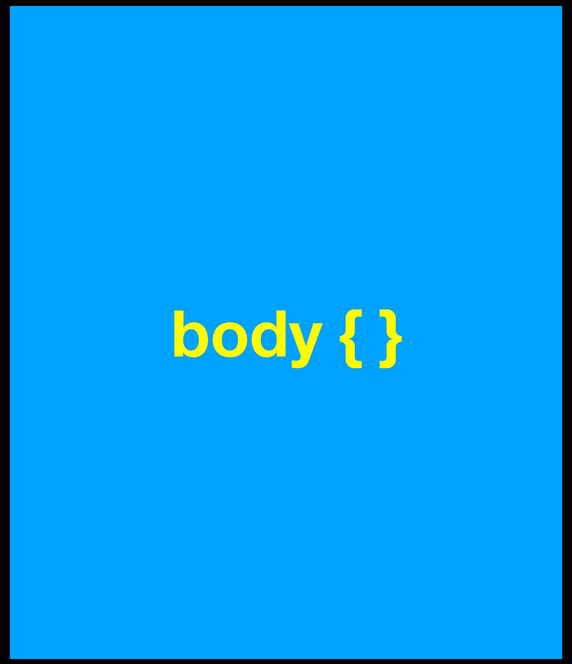
But

For iOS 17 projects I would use @Observable

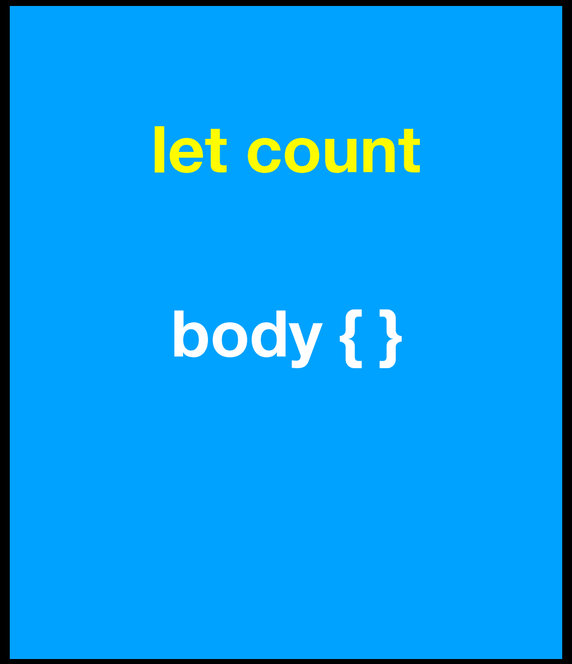
For existing projects that can target iOS 17 I would move to @Observable

Let's start simple

ContentView



ContentView



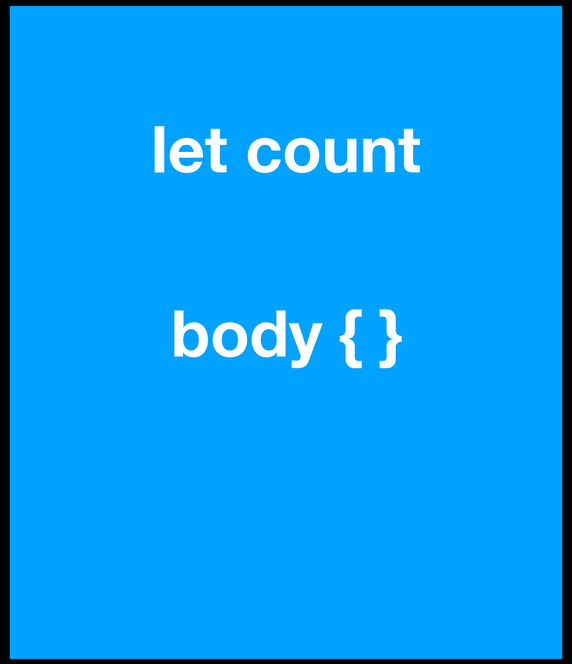
Local state

```
import SwiftUI

struct ContentView: View {

}
```

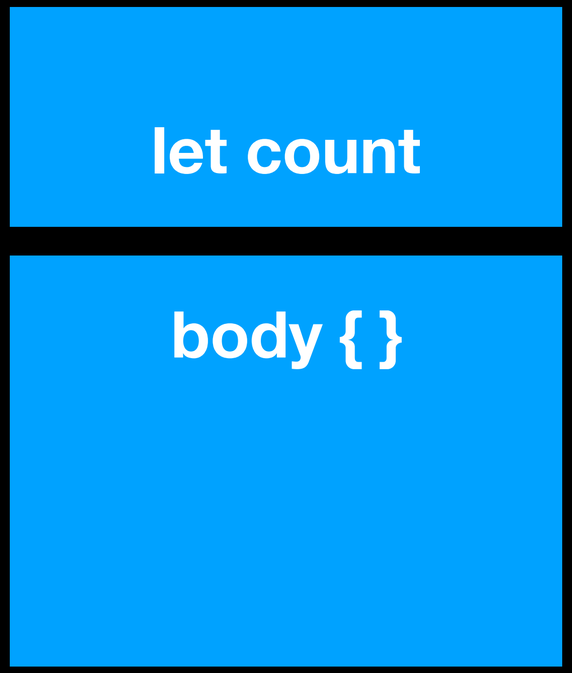
```
import SwiftUI

struct ContentView: View {

}
```

```
import SwiftUI

struct ContentView {

}

extension ContentView: View {




}
```

ContentView

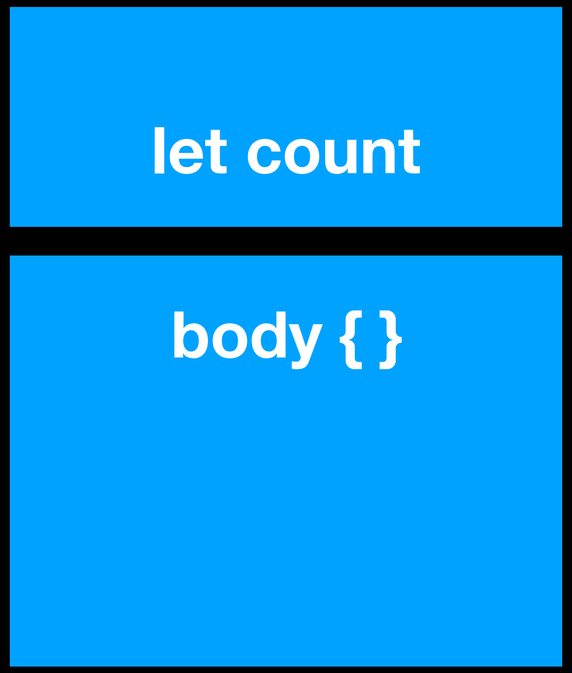


ContentView



ContentView

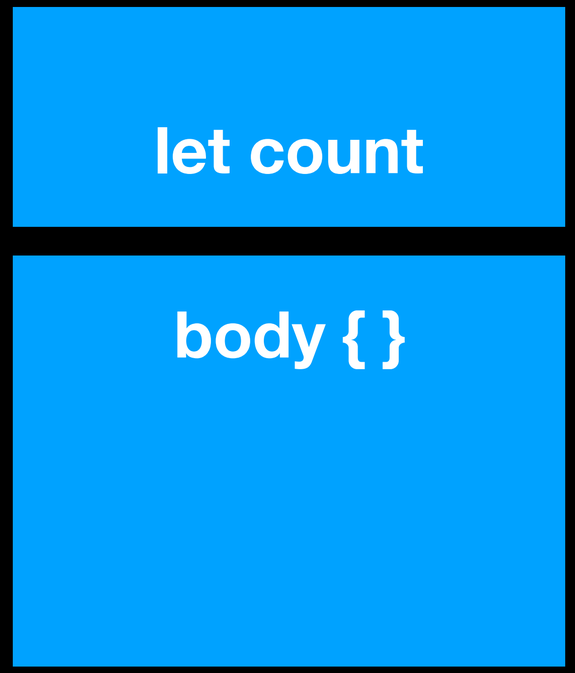
memory footprint



ContentView

memory footprint

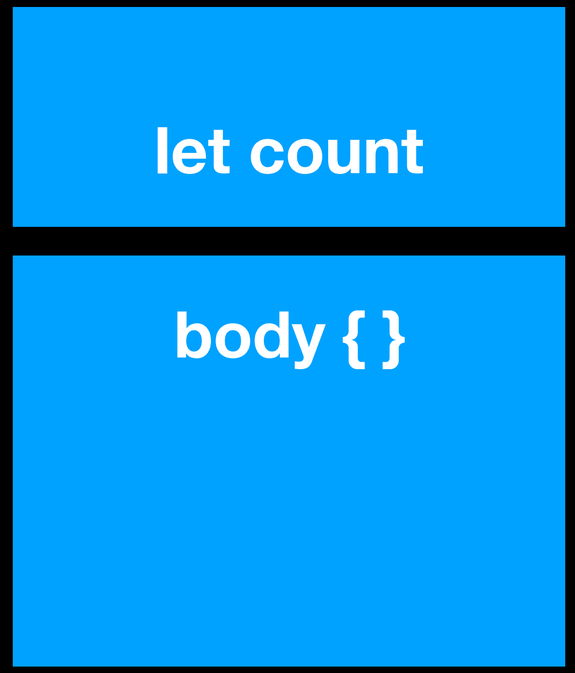
: View



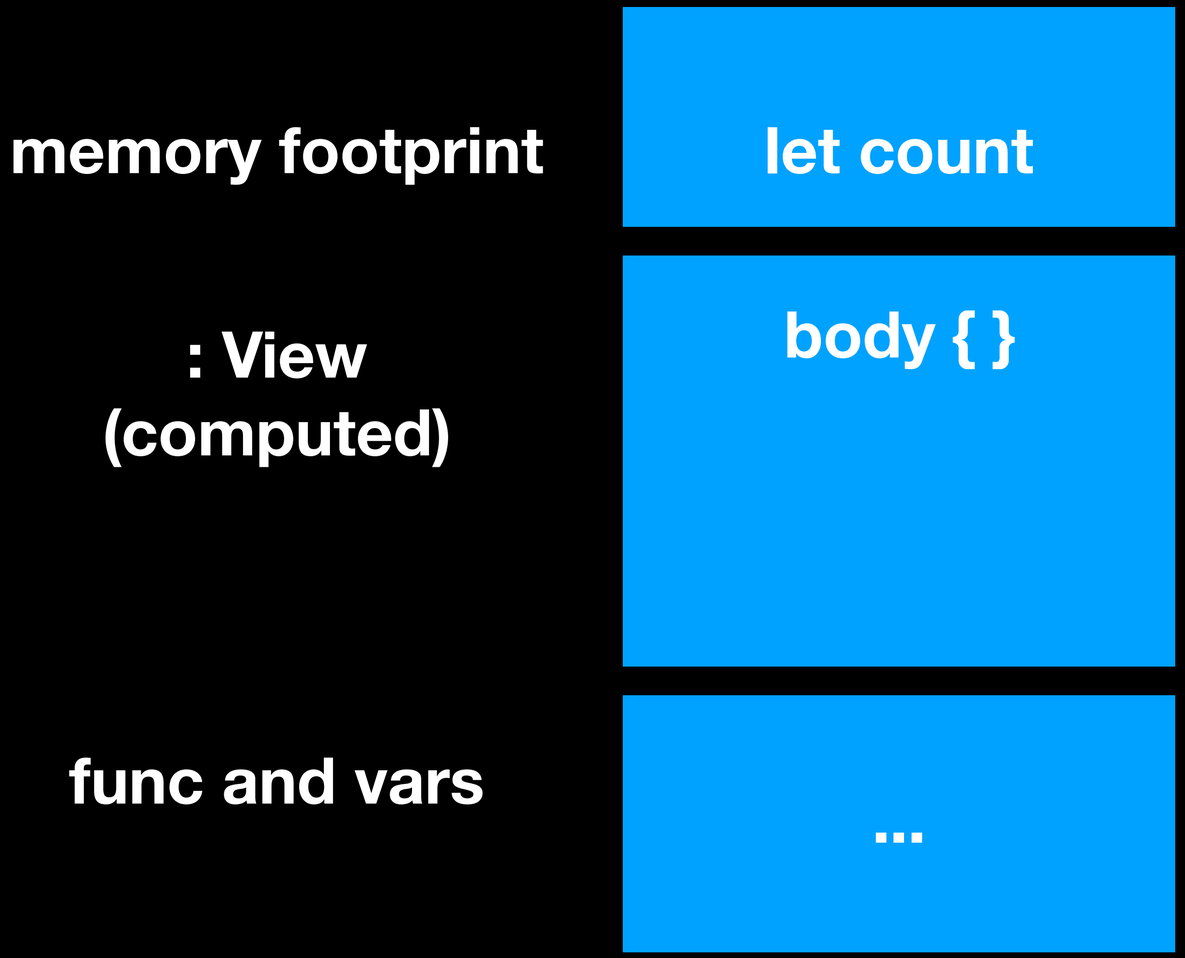
ContentView

memory footprint

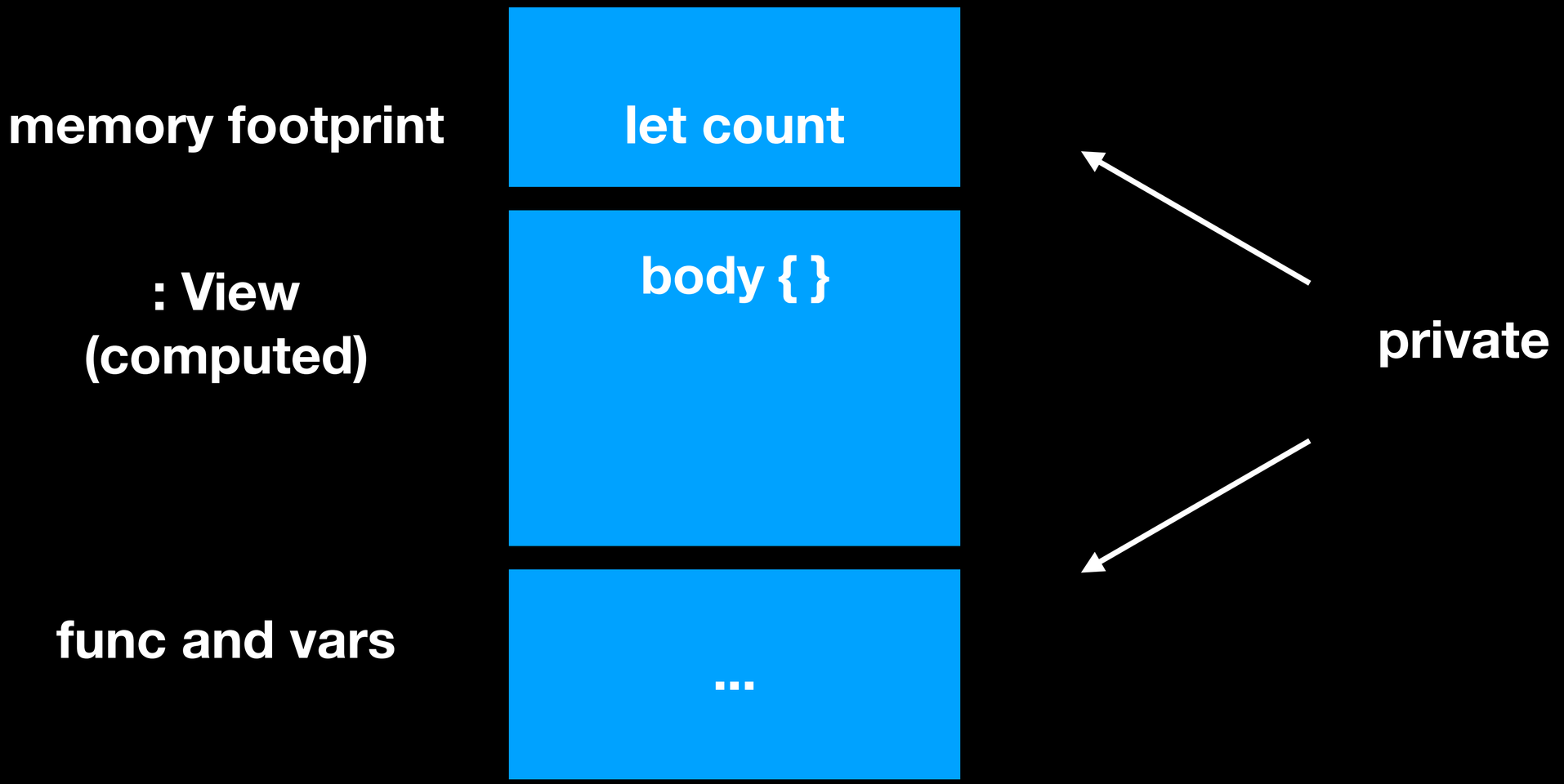
: View
(computed)



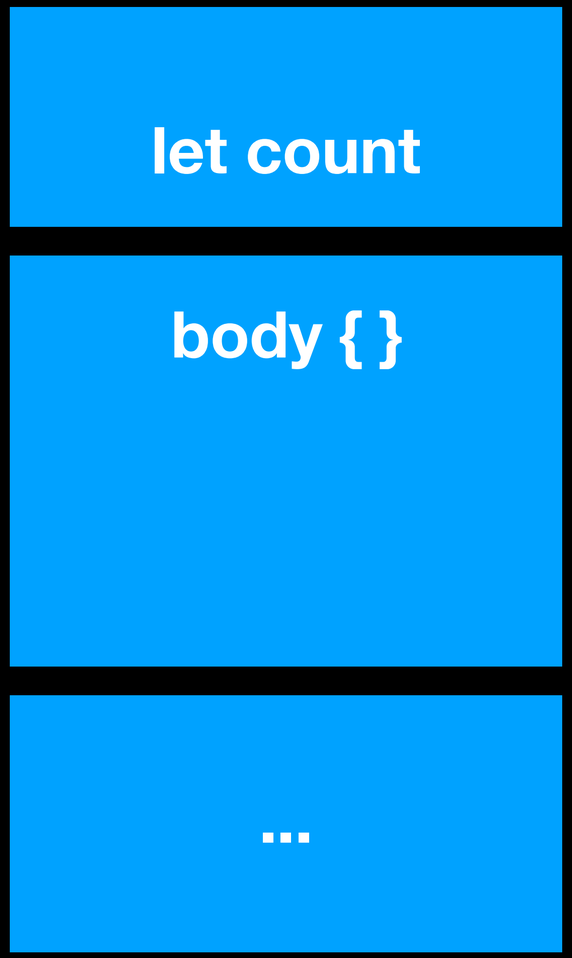
ContentView



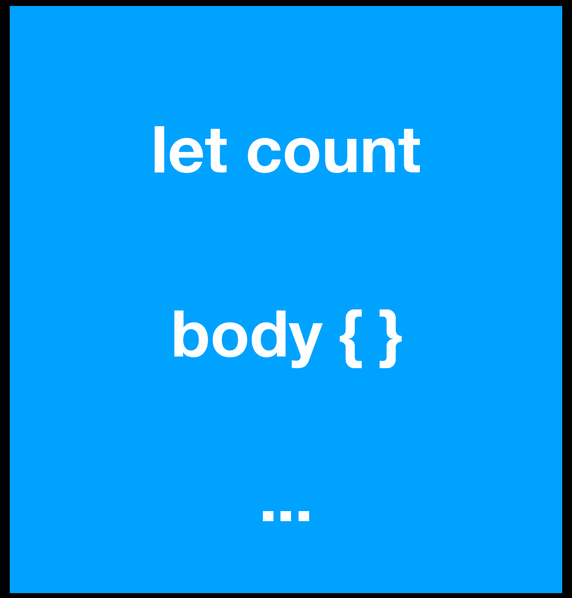
ContentView



ContentView



ContentView



```
import SwiftUI

struct ContentView {

}

extension ContentView: View {




}
```

```
import SwiftUI

struct ContentView {

}

extension ContentView: View {




}
```

```
import SwiftUI

struct ContentView {
  let count = 0
}

extension ContentView: View {




}
```

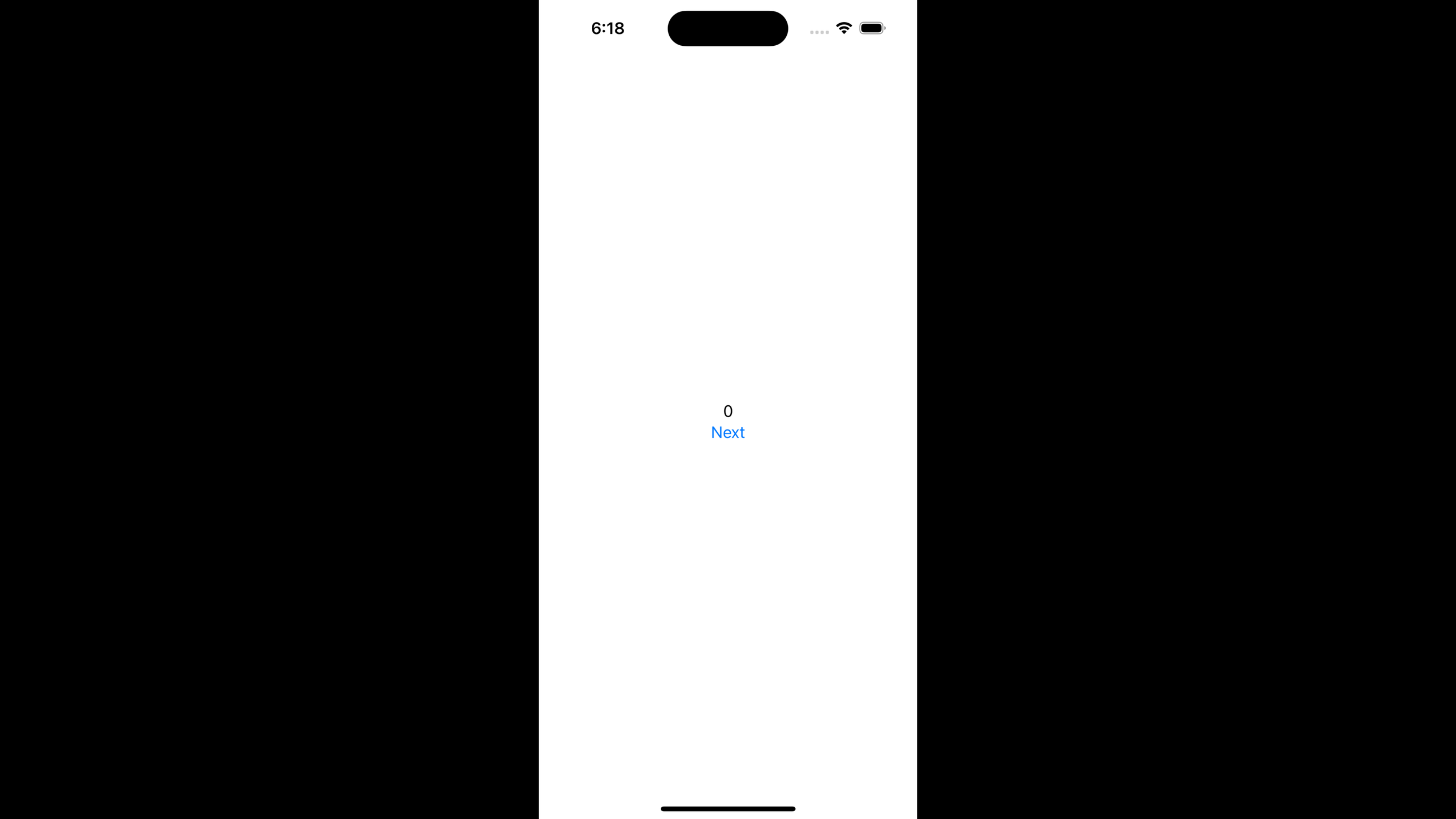
```
import SwiftUI

struct ContentView {
  let count = 0
}

extension ContentView: View {




}
```

```
import SwiftUI

struct ContentView {
  let count = 0
}

extension ContentView: View {
  var body: some View {




  }
}
```

```
import SwiftUI

struct ContentView {
  let count = 0
}

extension ContentView: View {
  var body: some View {
    VStack {




    }
  }
}
```

```
import SwiftUI

struct ContentView {
  let count = 0
}

extension ContentView: View {
  var body: some View {
    VStack {
      Text(count.description)



    }
  }
}
```

```
import SwiftUI

struct ContentView {
  let count = 0
}

extension ContentView: View {
  var body: some View {
    VStack {
      Text(count.description)
        Button("Next") {

        }
    }
  }
}
```


6:18
0
Next

6:18

0

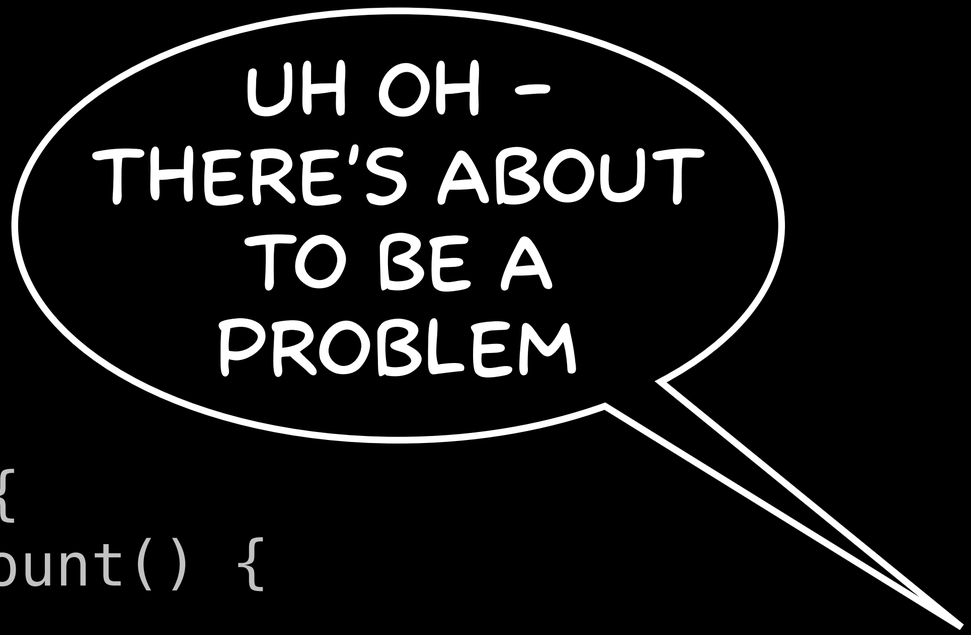
Next

```
struct ContentView {
  let count = 0
}

extension ContentView: View {
  var body: some View {
    VStack {
      Text(count.description)
      Button("Next") {

      }
    }
  }
}

extension ContentView {



}
```

```
struct ContentView {
  let count = 0
}

extension ContentView: View {
  var body: some View {
    VStack {
      Text(count.description)
      Button("Next") {

      }
    }
  }
}

extension ContentView {
  private func changeCount() {

  }
}
```

```
struct ContentView {
  let count = 0
}

extension ContentView: View {
  var body: some View {
    VStack {
      Text(count.description)
      Button("Next") {
        changeCount()
      }
    }
  }
}

extension ContentView {
  private func changeCount() {

  }
}
```

```
struct ContentView {
  let count = 0
}

extension ContentView: View {
  var body: some View {
    VStack {
      Text(count.description)
      Button("Next") {
        changeCount()
      }
    }
  }
}

extension ContentView {
  private func changeCount() {

  }
}
```



```
struct ContentView {
  let count = 0
}

extension ContentView: View {
  var body: some View {
    VStack {
      Text(count.description)
      Button("Next") {
        changeCount()
      }
    }
  }
}

extension ContentView {
  private func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

```
struct ContentView {
  let count = 0
}

extension ContentView: View {
  var body: some View {
    VStack {
      Text(count.description)
      Button("Next") {
        changeCount()
      }
    }
  }
}

extension ContentView {
  private func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

```
struct ContentView {
  let count = 0
}

extension ContentView: View {
  var body: some View {
    VStack {
      Text(count.description)
      Button("Next") {
        changeCount()
      }
    }
  }
}

extension ContentView {
  private func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

```
struct ContentView {
  let count = 0
}

extension ContentView: View {
  var body: some View {
    VStack {
      Text(count.description)
      Button("Next") {
        changeCount()
      }
    }
  }
}

extension ContentView {
  private func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

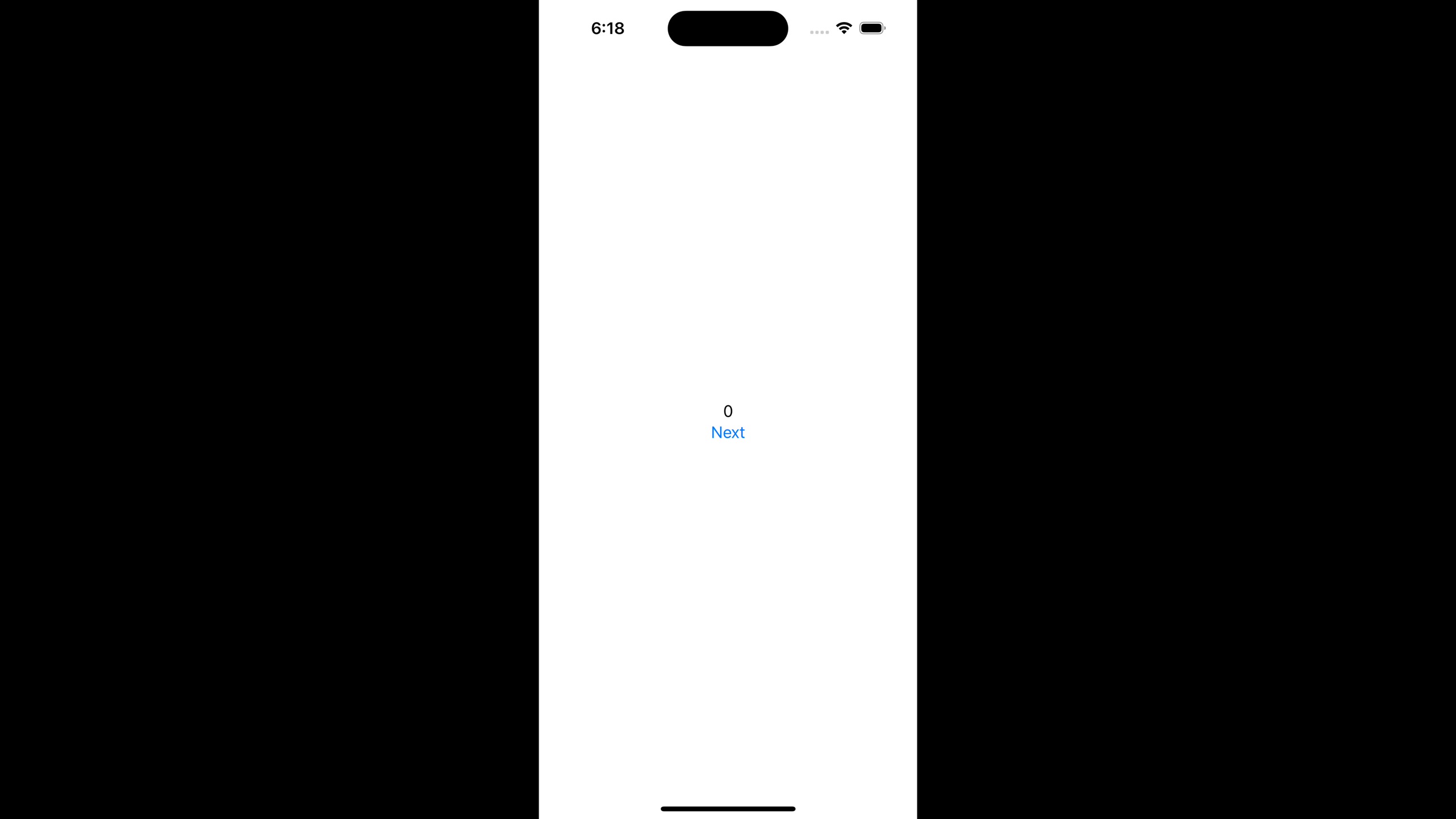
```
struct ContentView {
  var count = 0
}

extension ContentView: View {
  var body: some View {
    VStack {
      Text(count.description)
      Button("Next") {
        changeCount()
      }
    }
  }
}

extension ContentView {
  private func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

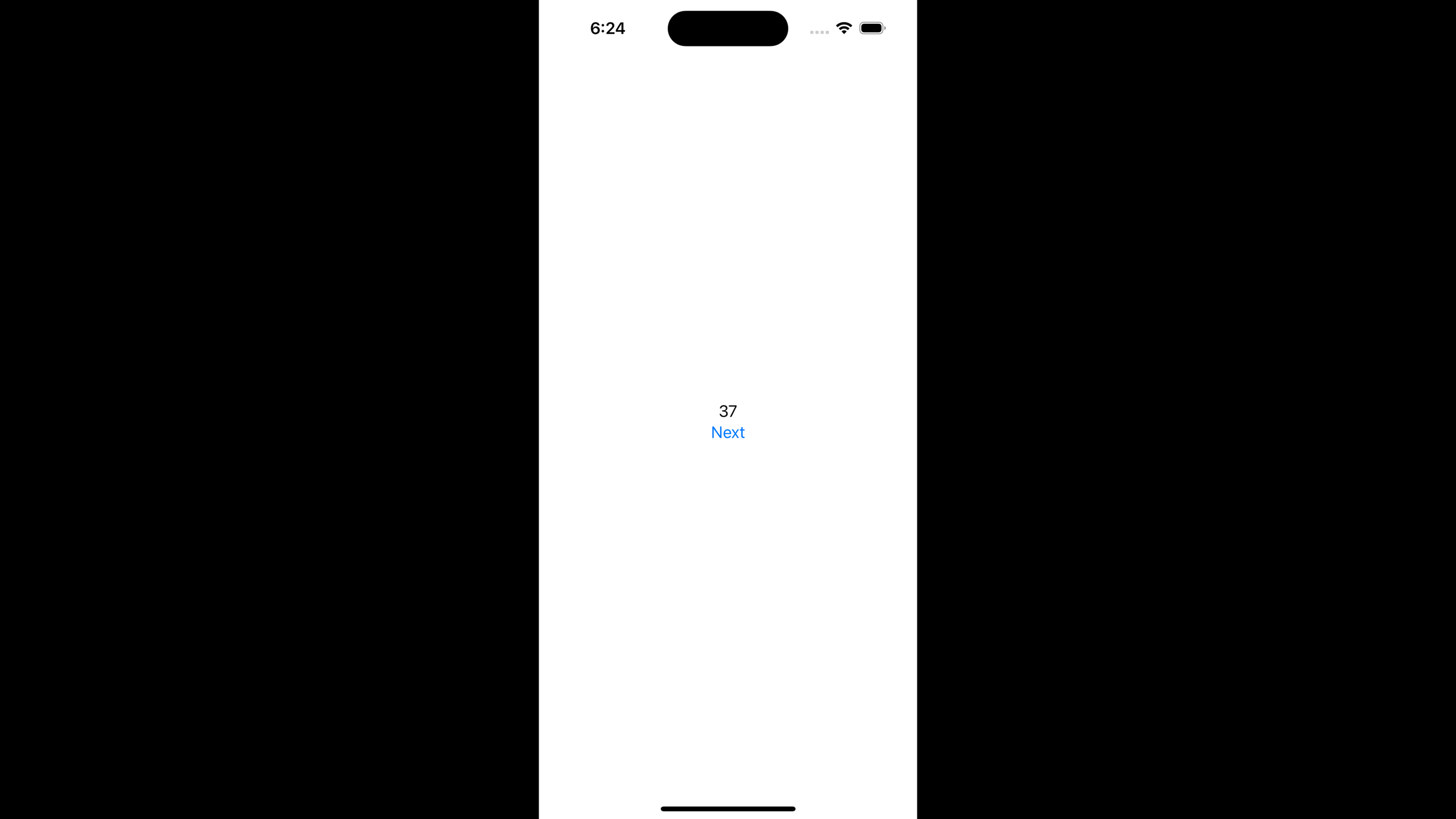
```
struct ContentView {
  var count = 0
}

extension ContentView: View {
  var body: some View {
    VStack {
      Text(count.description)
      Button("Next") {
        changeCount()
      }
    }
  }
}

extension ContentView {
  private func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

```
struct ContentView {
  var count = 0
}

extension ContentView: View {
  var body: some View {
    VStack {
      Text(count.description)
      Button("Next") {
        changeCount()
      }
    }
  }
}

extension ContentView {
  private func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

```
struct ContentView {
  var count = 0
}

extension ContentView: View {
  var body: some View {
    VStack {
      Text(count.description)
      Button("Next") {
        changeCount()
      }
    }
  }
}

extension ContentView {
  private func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

```
struct ContentView {
  @State var count = 0
}

extension ContentView: View {
  var body: some View {
    VStack {
      Text(count.description)
      Button("Next") {
        changeCount()
      }
    }
  }
}

extension ContentView {
  private func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```


6:18
0
Next


6:24
37
Next

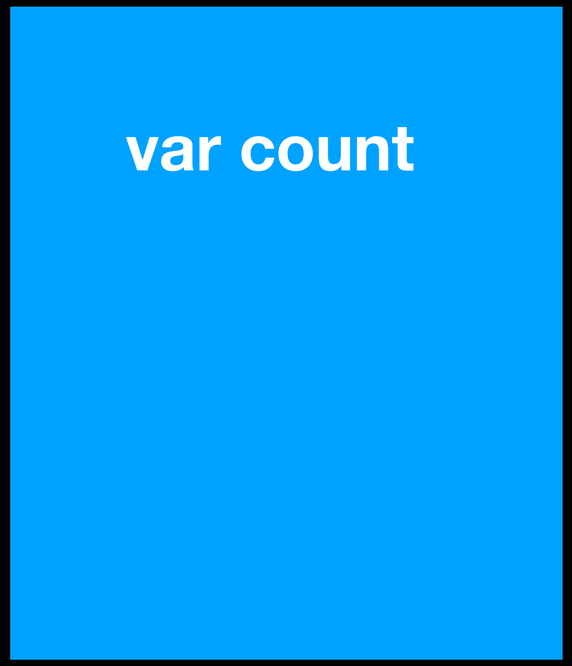
@State is doing a lot

```
struct ContentView {
  @State var count = 0
}

extension ContentView: View {
  var body: some View {
    VStack {
      Text(count.description)
      Button("Next") {
        changeCount()
      }
    }
  }
}

extension ContentView {
  private func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

```
struct ContentView {
  @State var count = 0
}

extension ContentView: View {
  var body: some View {
    VStack {
      Text(count.description)
      Button("Next") {
        changeCount()
      }
    }
  }
}

extension ContentView {
  private func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

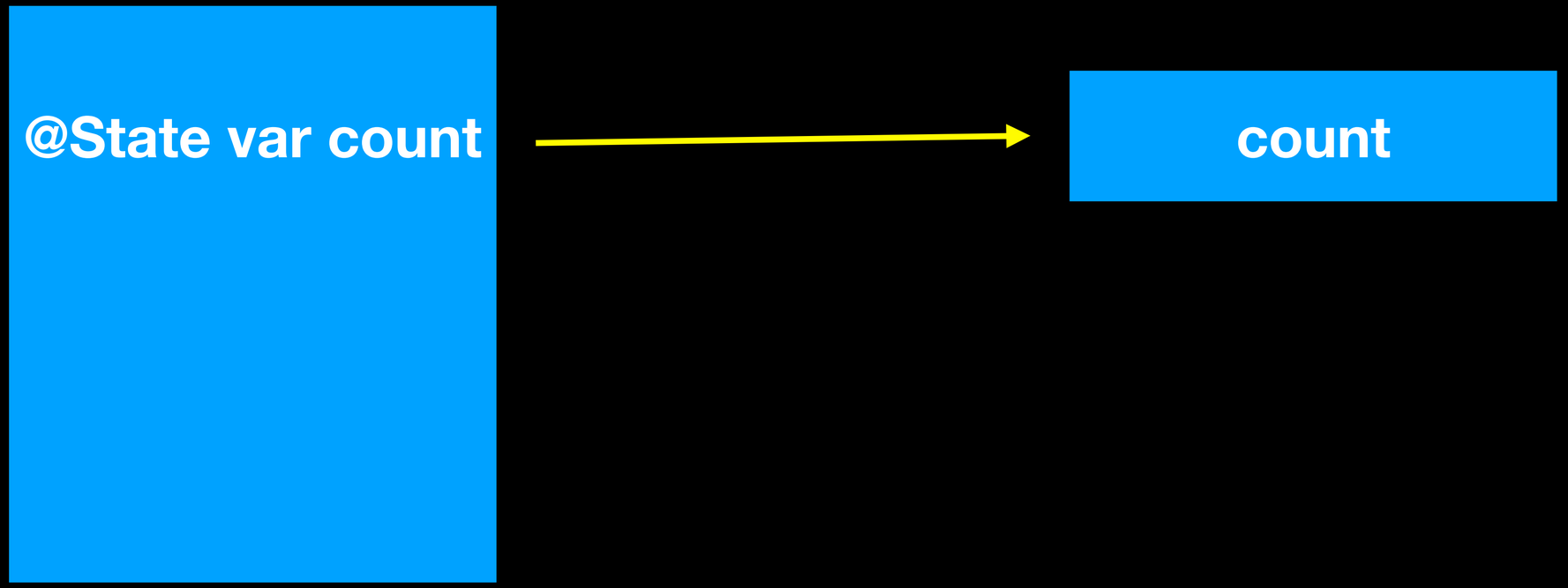
```
struct ContentView {
  @State var count = 0
}

extension ContentView: View {
  var body: some View {
    VStack {
      Text(count.description)
      Button("Next") {
        changeCount()
      }
    }
  }
}

extension ContentView {
  private func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

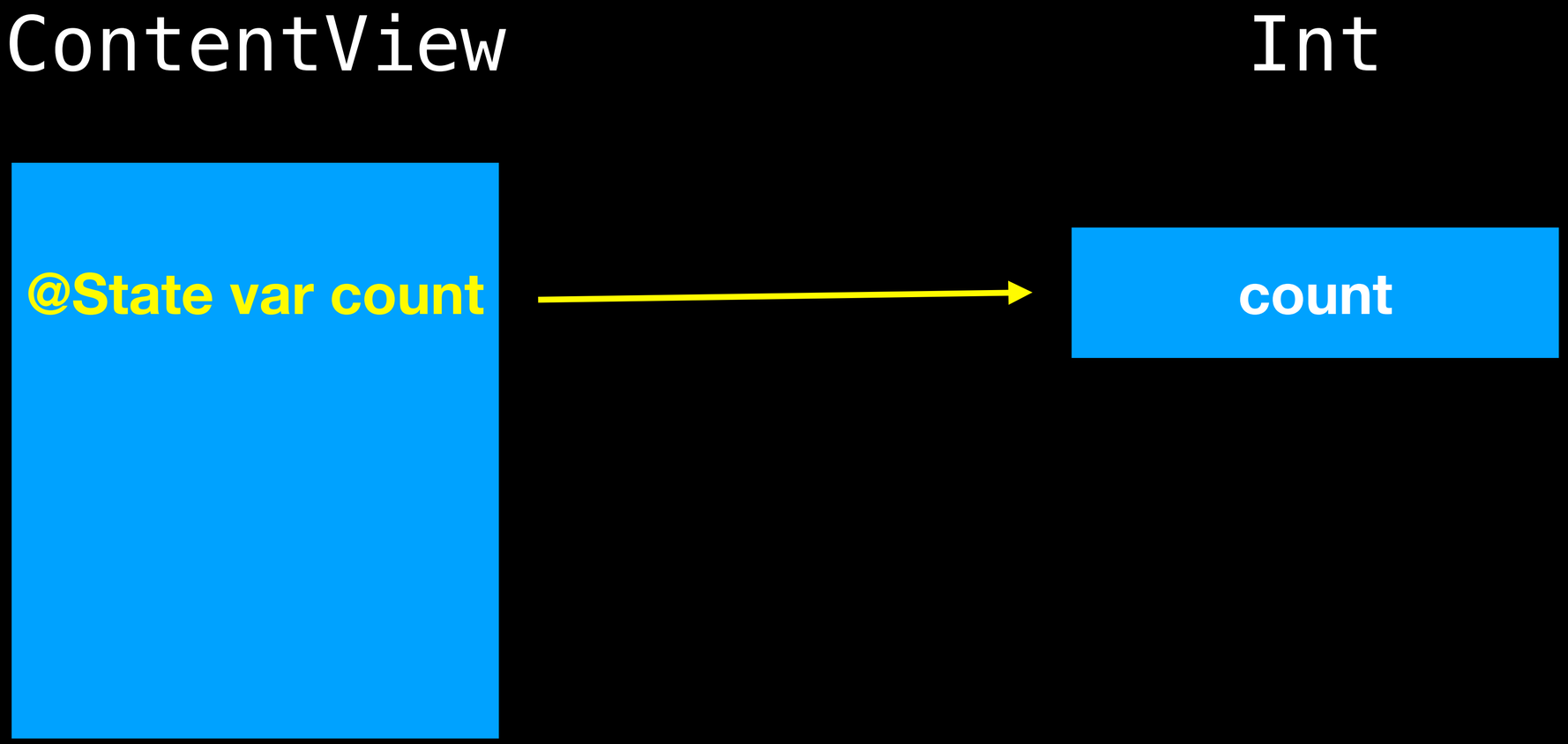
ContentView



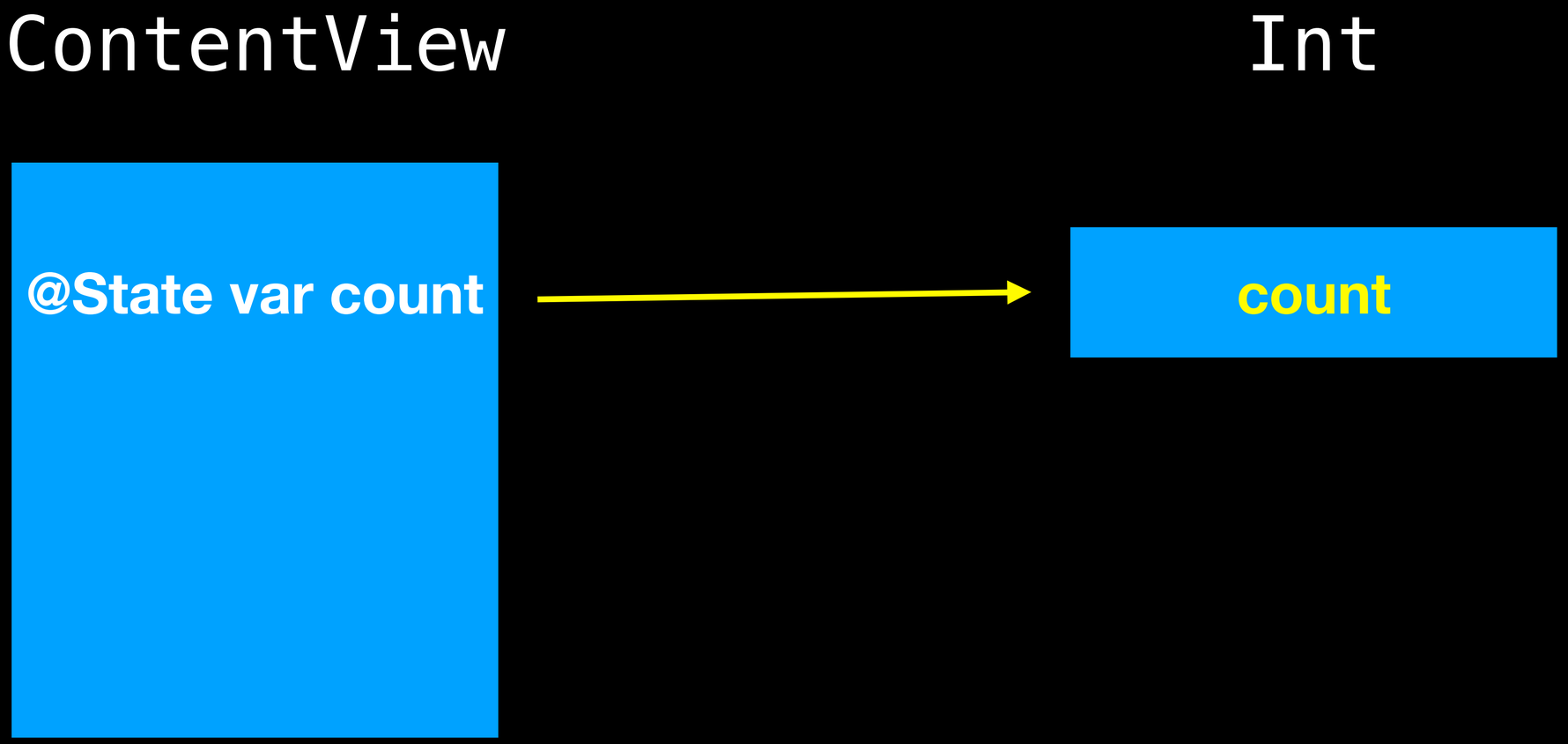
ContentView

Int





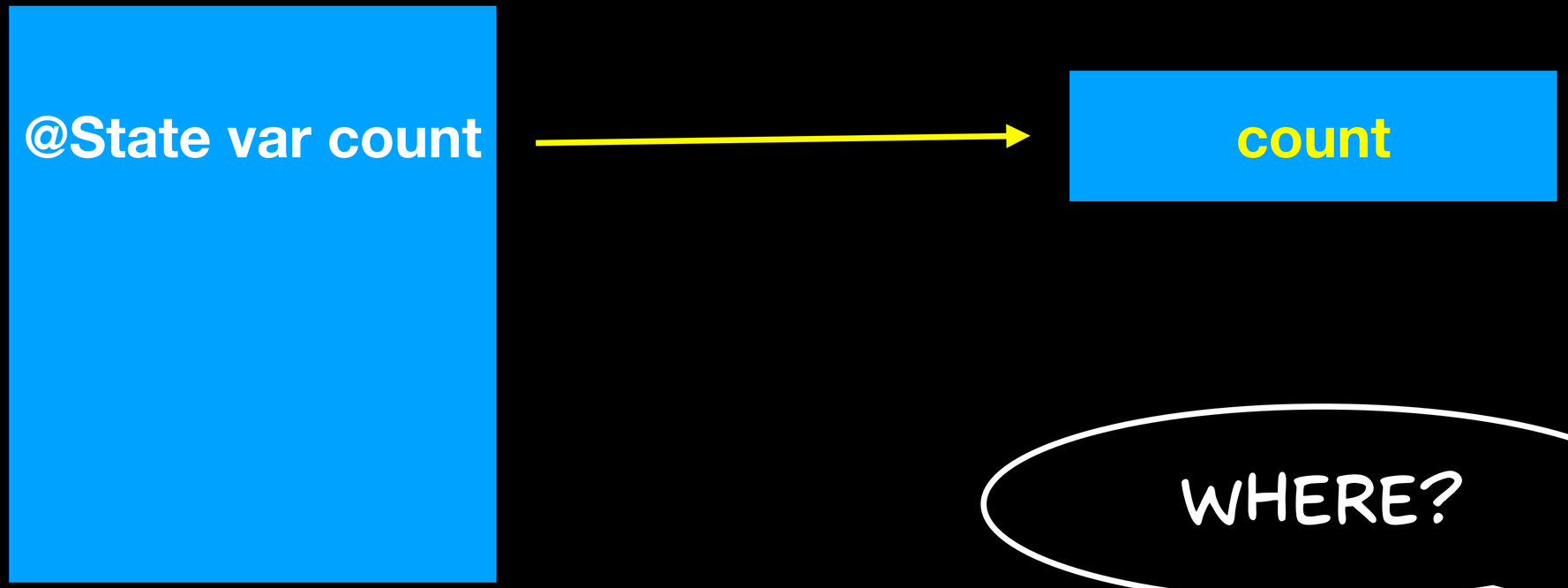
this doesn't change



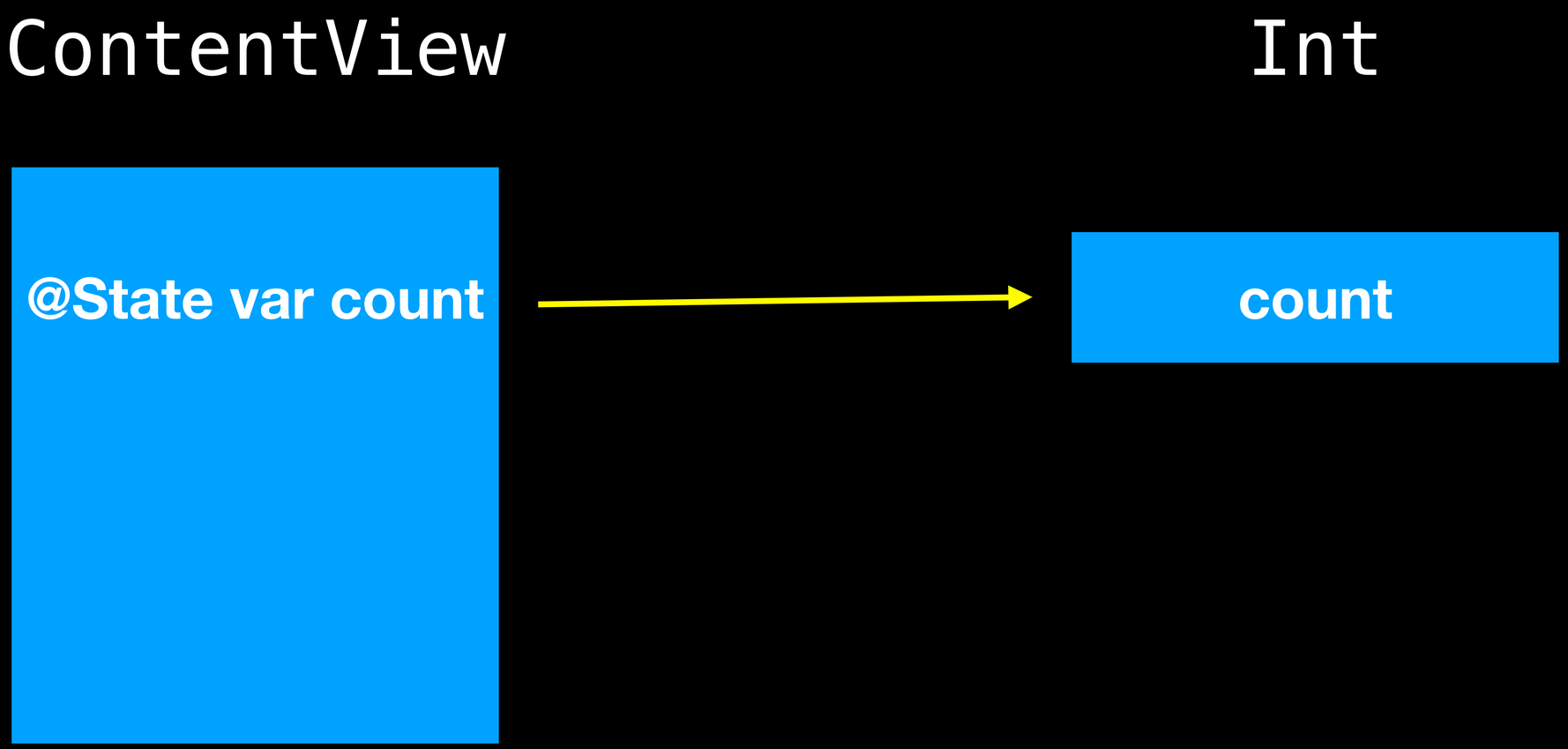
this is stored elsewhere

ContentView

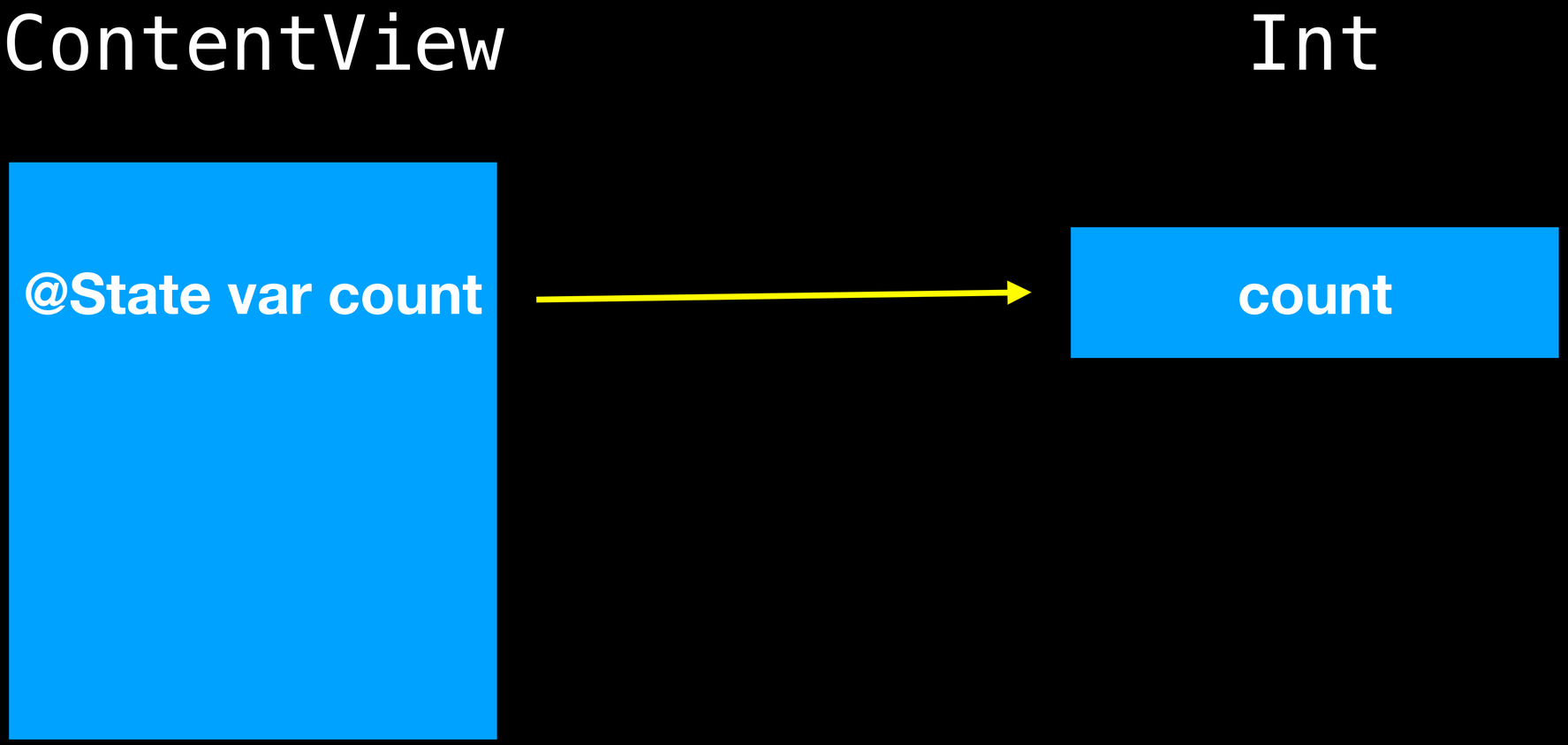
Int



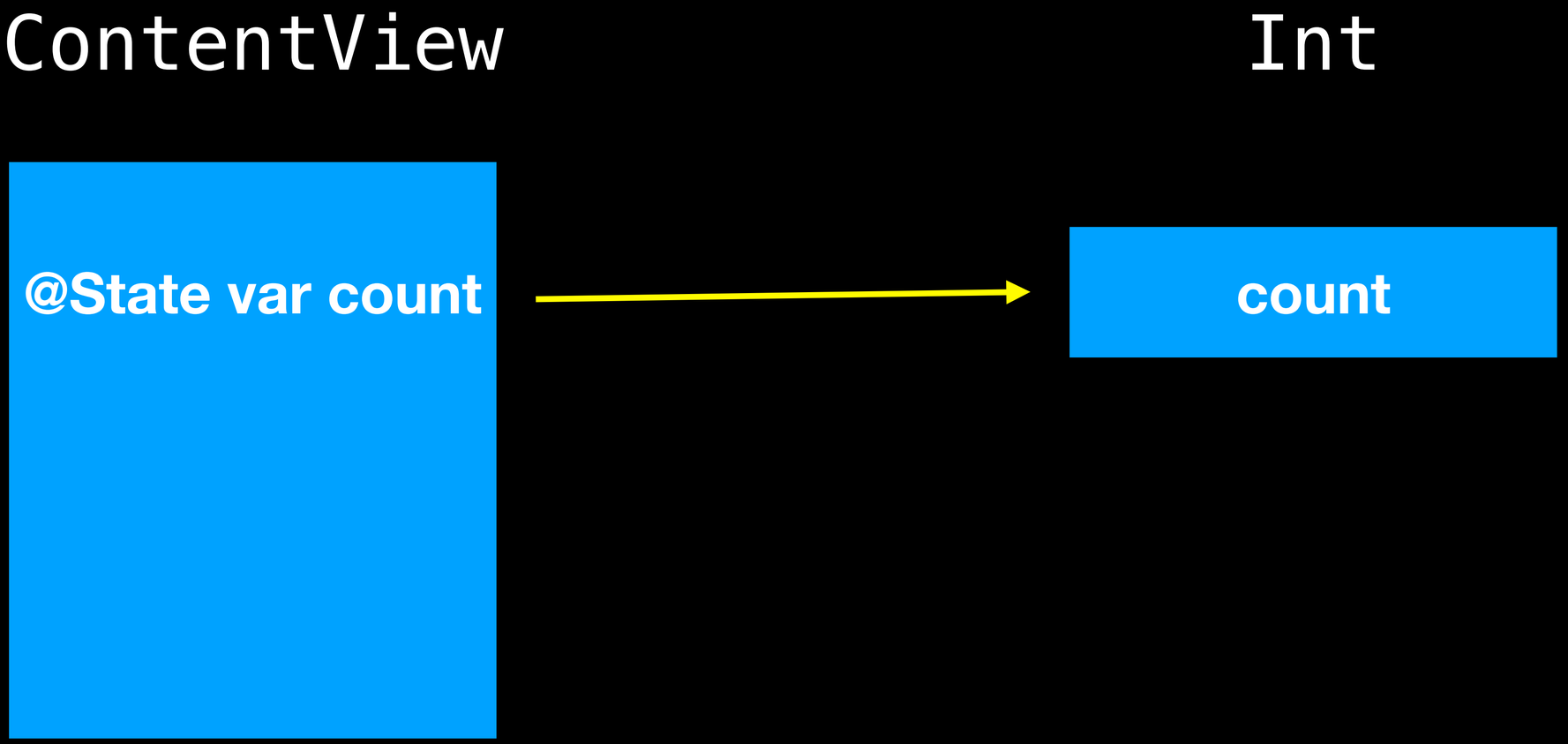
this is stored elsewhere



this is stored elsewhere



Remember - Views are cheap

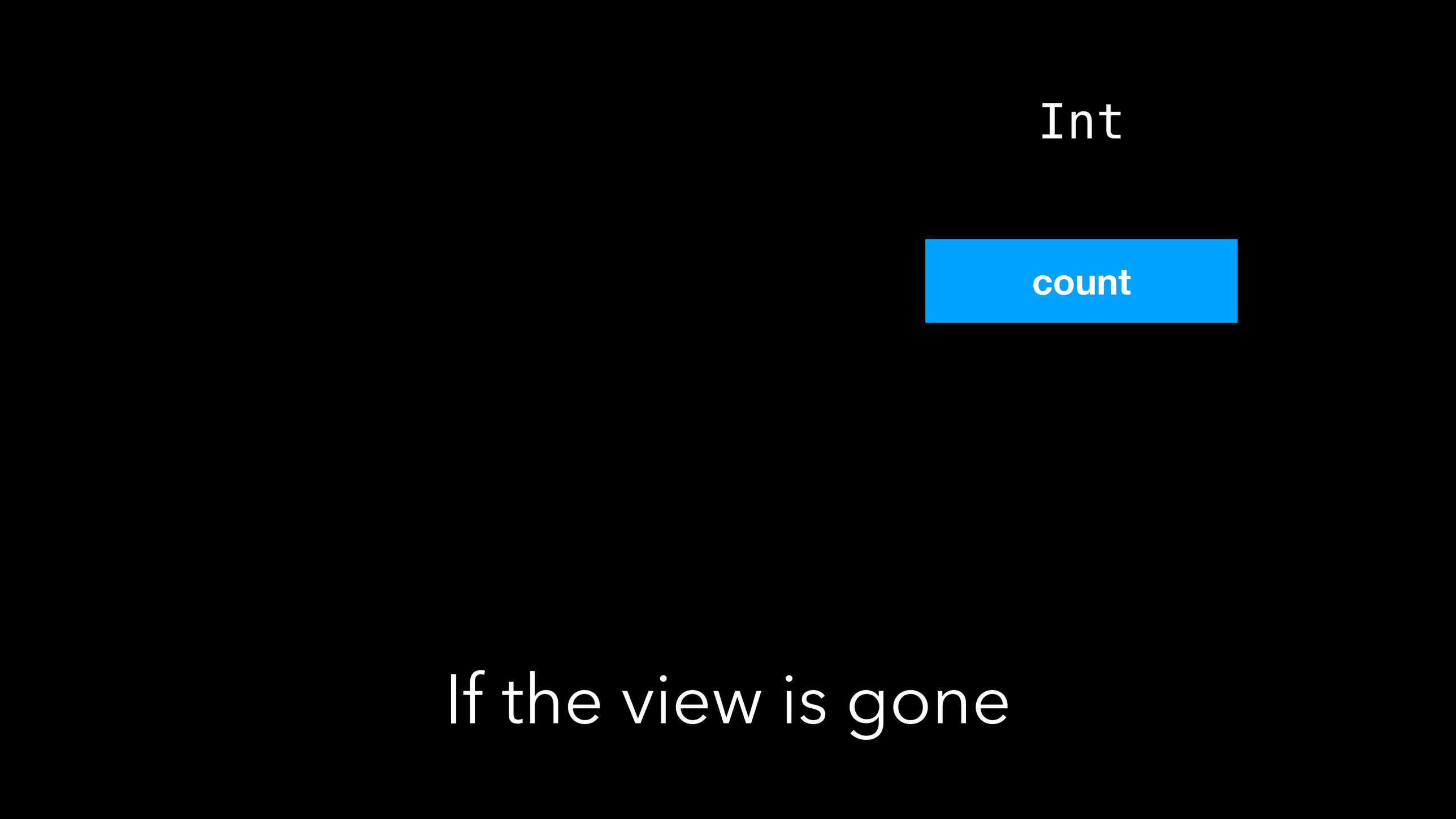


Remember - Views go away

Int

count

You can think of it this way...



If the view is gone

`Int`

count

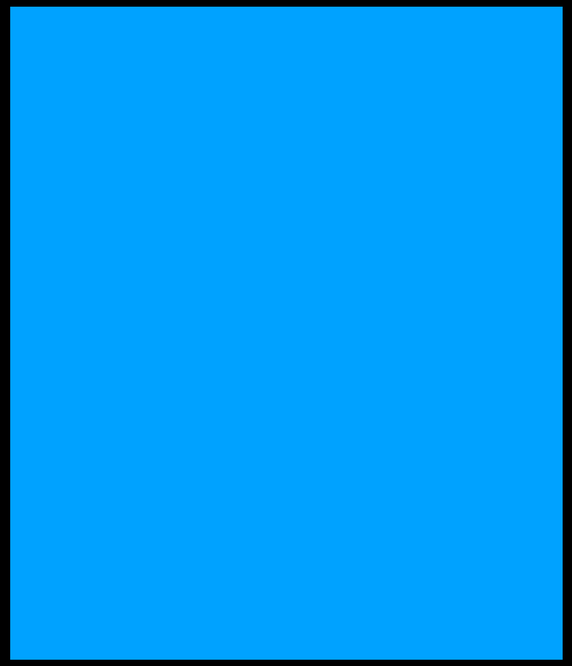
Someone needs to know

Int

count

ContentView's body depends on count

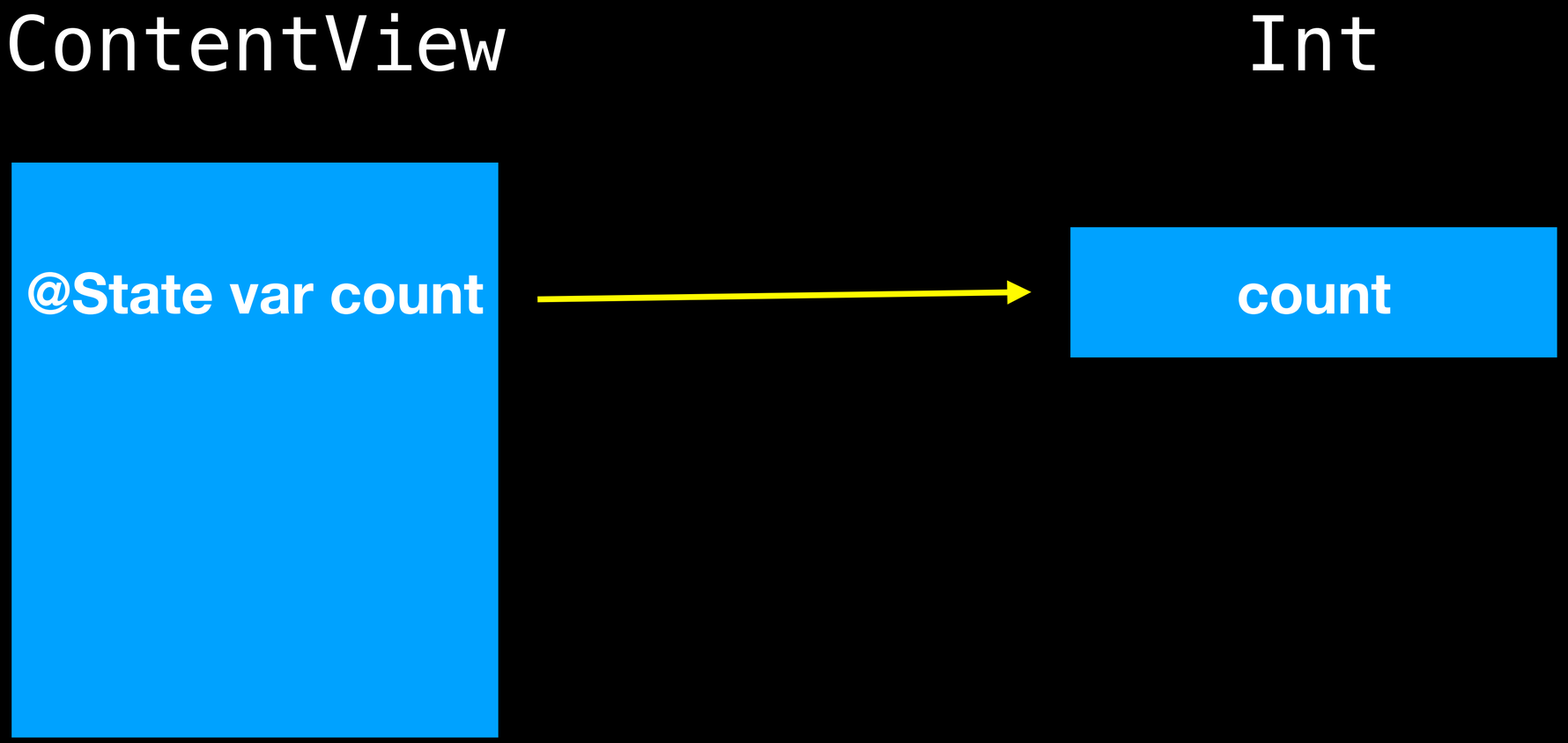
ContentView



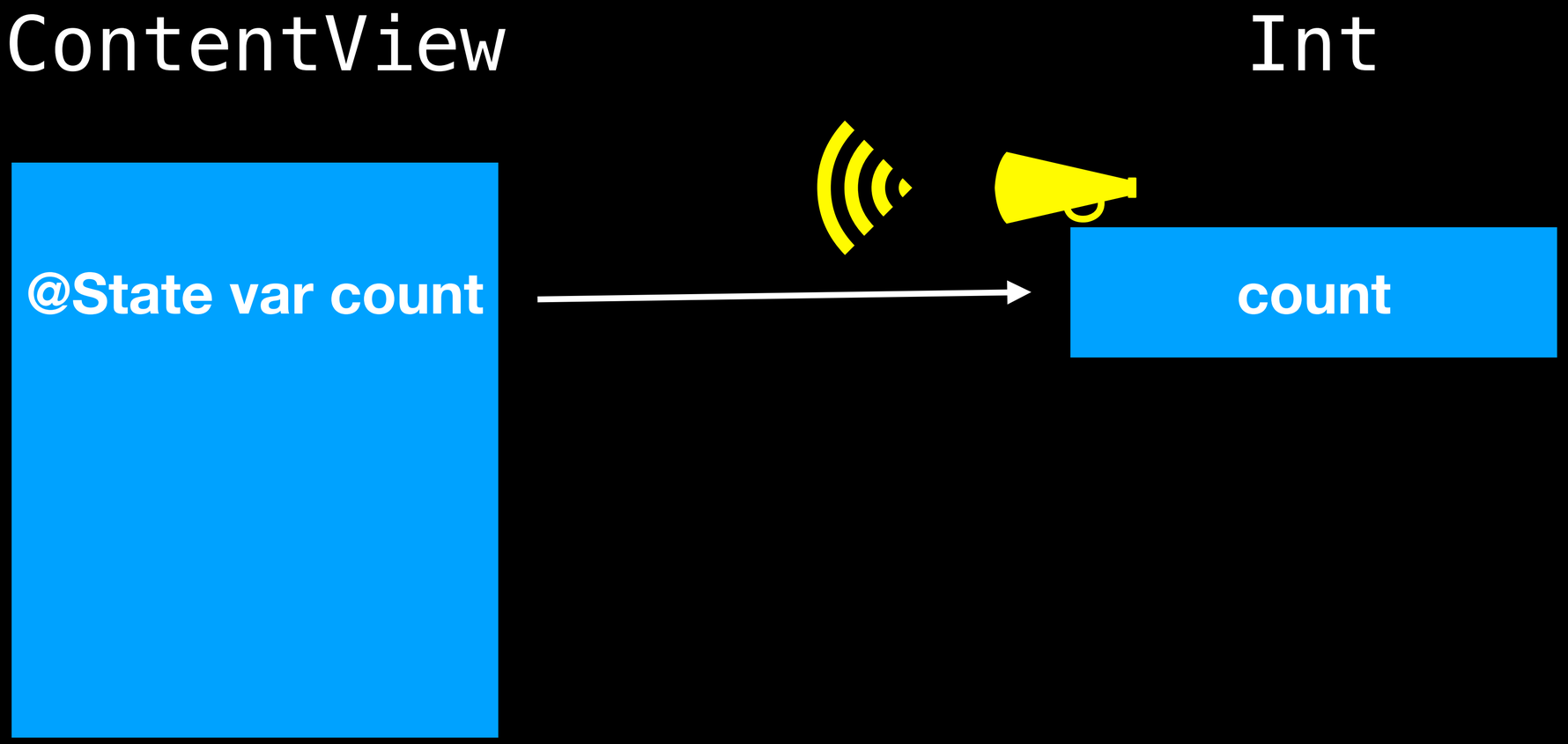
Int

count

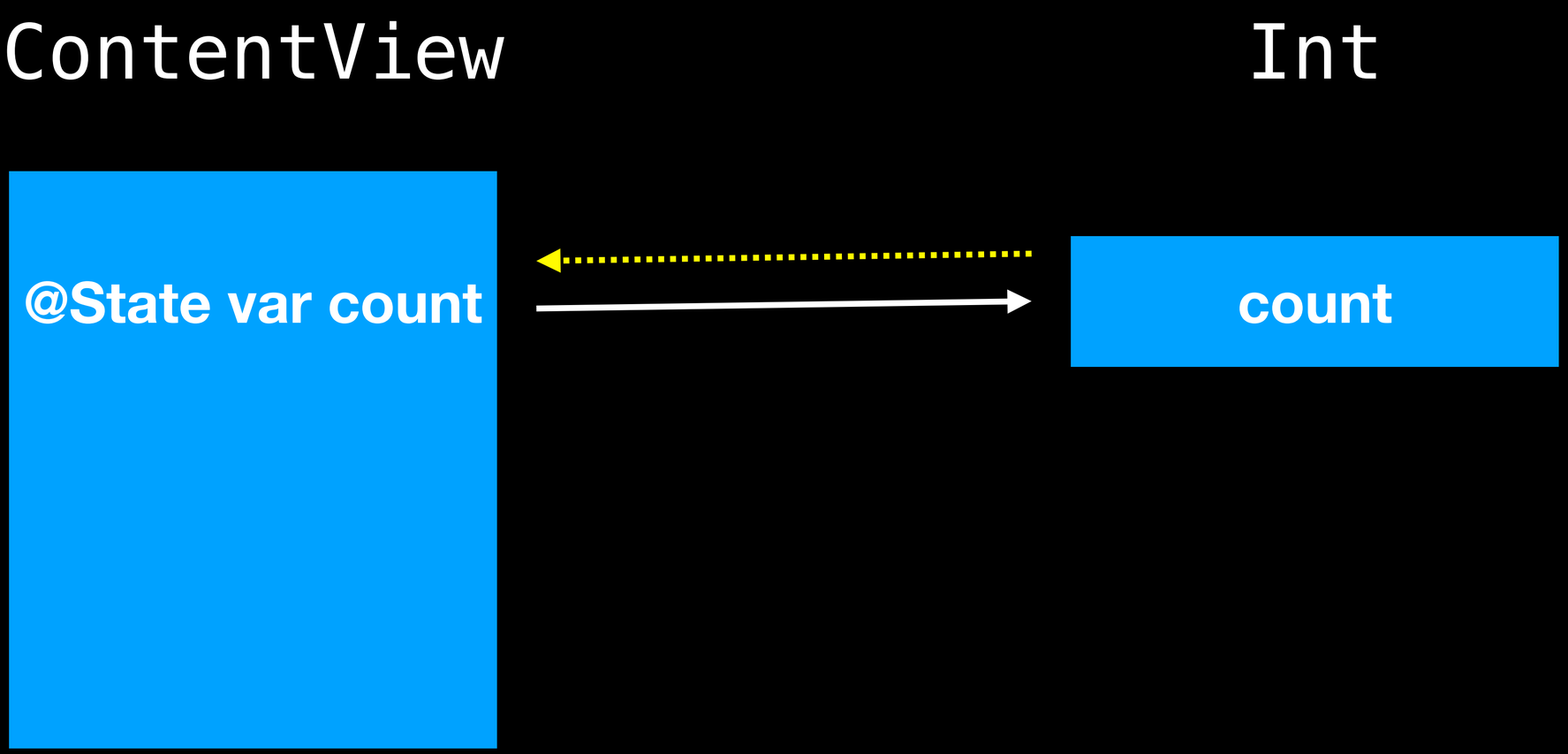
ContentView is created again



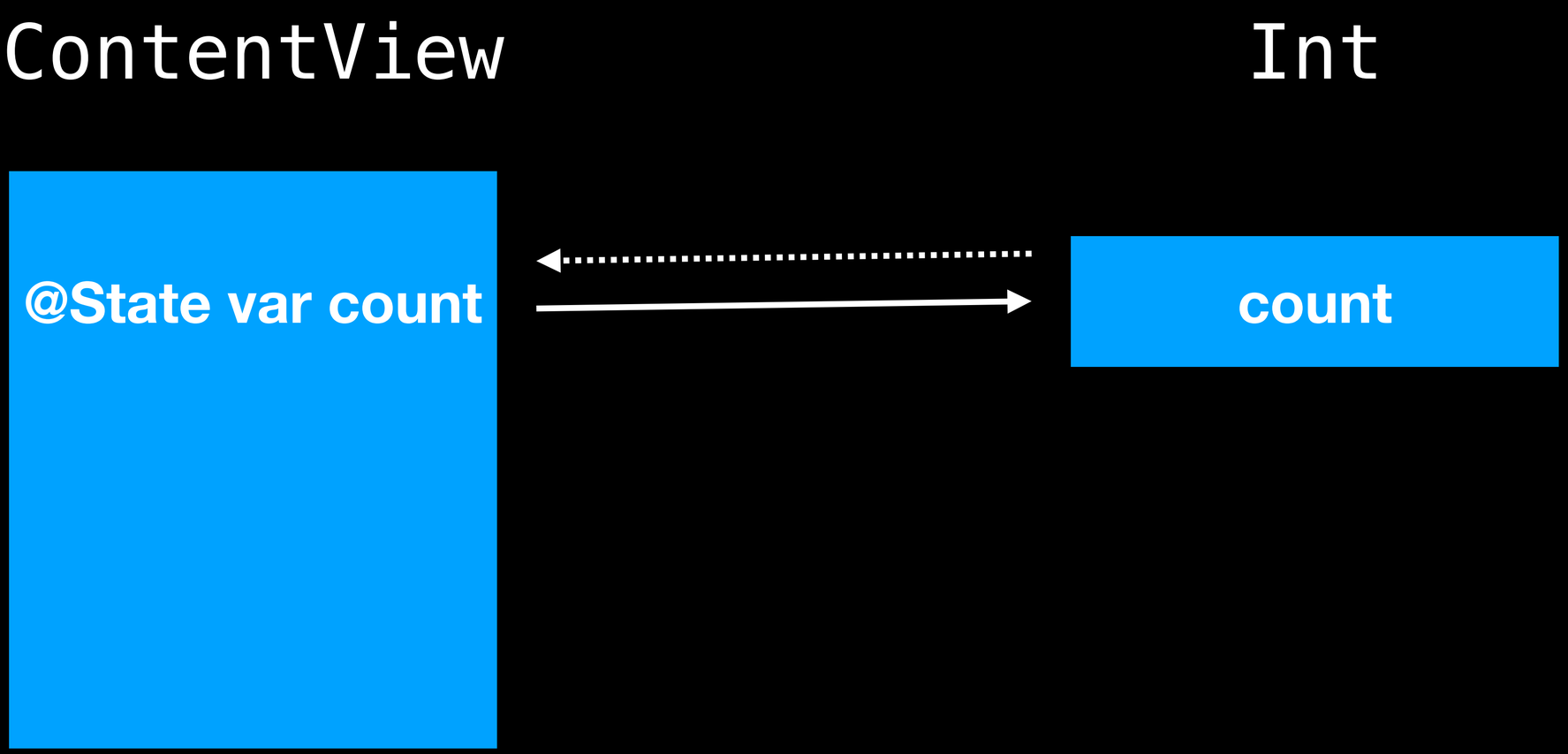
and reconnected



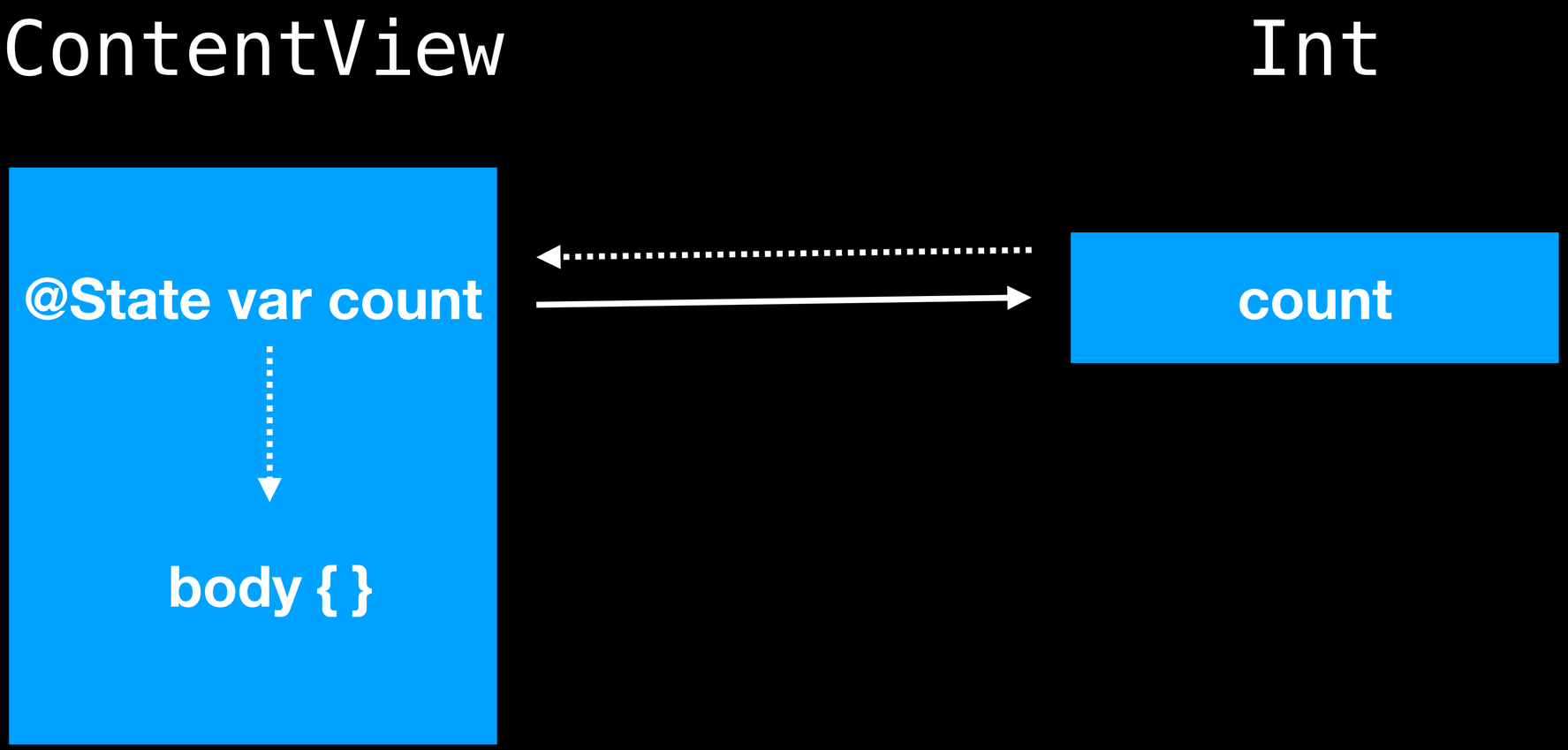
The view is told count will change



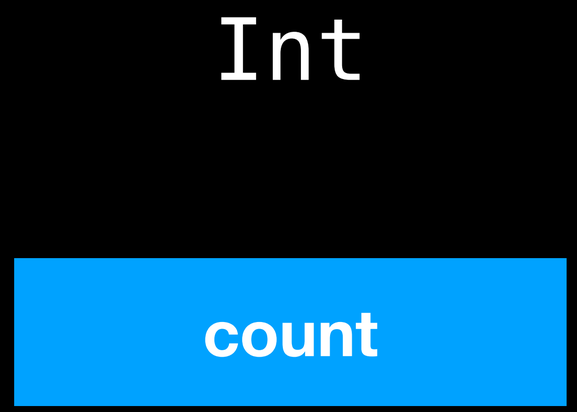
The view is told count will change



The view is told count will change



View updates body



And disappears

@State is doing a lot

Next move the state out of the view

A lot of ContentView isn't View code

```
struct ContentView {
  @State var count = 0
}

extension ContentView: View {
  var body: some View {
    VStack {
      Text(count.description)
      Button("Next") {
        changeCount()
      }
    }
  }
}

extension ContentView {
  private func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

So you create a Controller

So you create a Controller
or ViewModel

So you create a Controller
or ViewModel
or Presenter

```
class Controller {

}
```

```
class Controller {
                    var count = 0
}
```

```
class Controller {
  private(set) var count = 0
}
```

```
class Controller {
  private(set) var count = 0
}

extension Controller {



}
```

```
class Controller {
  private(set) var count = 0
}

extension Controller {
  private func changeCount() {

  }
}
```

```
class Controller {
  private(set) var count = 0
}

extension Controller {
  private func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

We've moved the controller code out of the view

Reconnect the view

```
import SwiftUI

struct ContentView {
        var controller = Controller()
}

extension ContentView: View {
  var body: some View {
    VStack {
      Text(          count.description)
      Button("Next") {
                  changeCount()
      }
    }
  }
}
```

```
import SwiftUI

struct ContentView {
        var controller = Controller()
}

extension ContentView: View {
  var body: some View {
    VStack {
      Text(controller.count.description)
      Button("Next") {
                  changeCount()
      }
    }
  }
}
```

```
import SwiftUI

struct ContentView {
          var controller = Controller()
}

extension ContentView: View {
  var body: some View {
    VStack {
      Text(controller.count.description)
      Button("Next") {
        controller.changeCount()
      }
    }
  }
}
```

```
import SwiftUI

struct ContentView {
          var controller = Controller()
}

extension ContentView: View {
  var body: some View {
    VStack {
      Text(controller.count.description)
      Button("Next") {
        controller.changeCount()
      }
    }
  }
}
```

ContentView

var controller = Controller()

ContentView

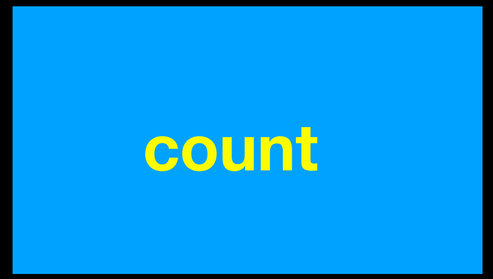
Controller

@State var controller = Controller()

count

Controller

Controller



```
import SwiftUI

struct ContentView {
        var controller = Controller()
}

extension ContentView: View {
  var body: some View {
    VStack {
      Text(controller.count.description)
      Button("Next") {
        controller.changeCount()
      }
    }
  }
}
```

```
import SwiftUI

struct ContentView {
  @State var controller = Controller()
}

extension ContentView: View {
  var body: some View {
    VStack {
      Text(controller.count.description)
      Button("Next") {
        controller.changeCount()
      }
    }
  }
}
```

There are other problems

Run the app

Tap the button

```
import SwiftUI

struct ContentView {
  @State var controller = Controller()
}

extension ContentView: View {
  var body: some View {
    VStack {
      Text(controller.count.description)
      Button("Next") {
        controller.changeCount()
      }
    }
  }
}
```

```
class Controller {
  private(set) var count = 0
}

extension Controller {
  func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

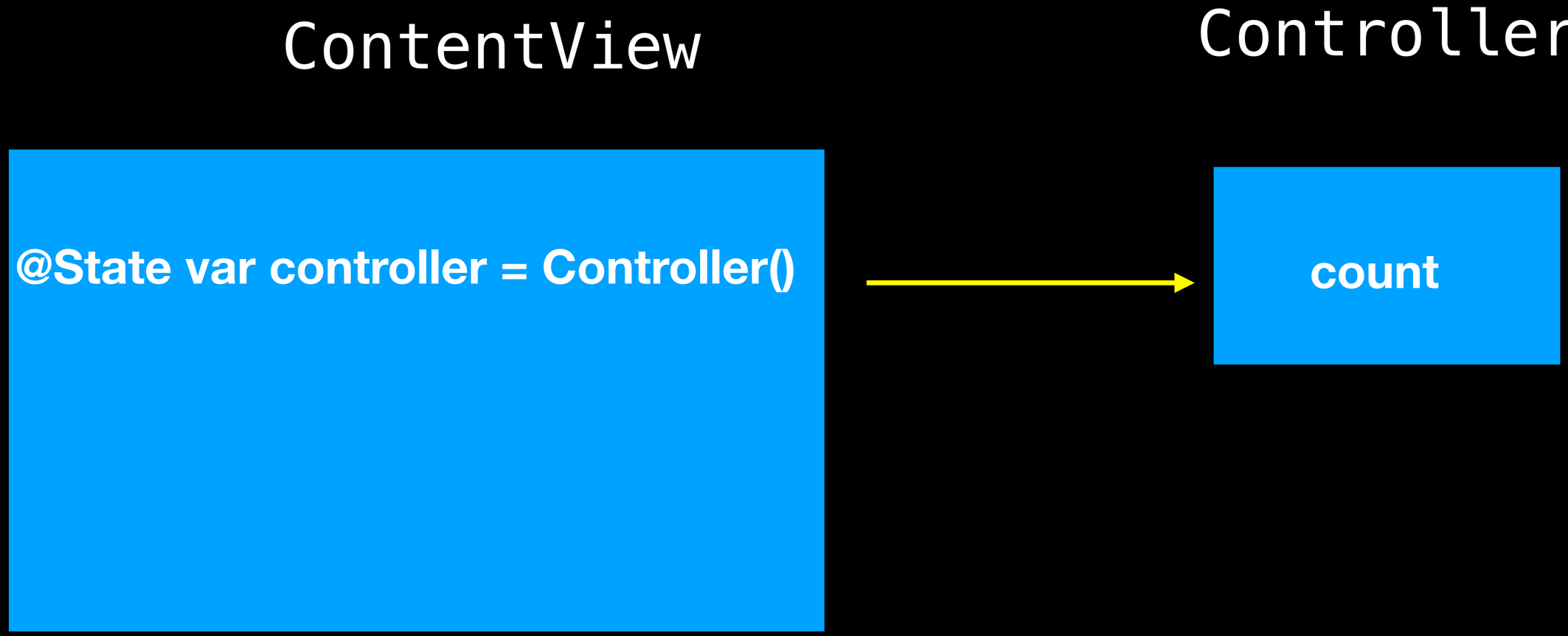
```
class Controller {
  private(set) var count = 0
}

extension Controller {
  func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

```
class Controller {
  private(set) var count = 0
}

extension Controller {
  func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

We don't see updates

We don't see updates

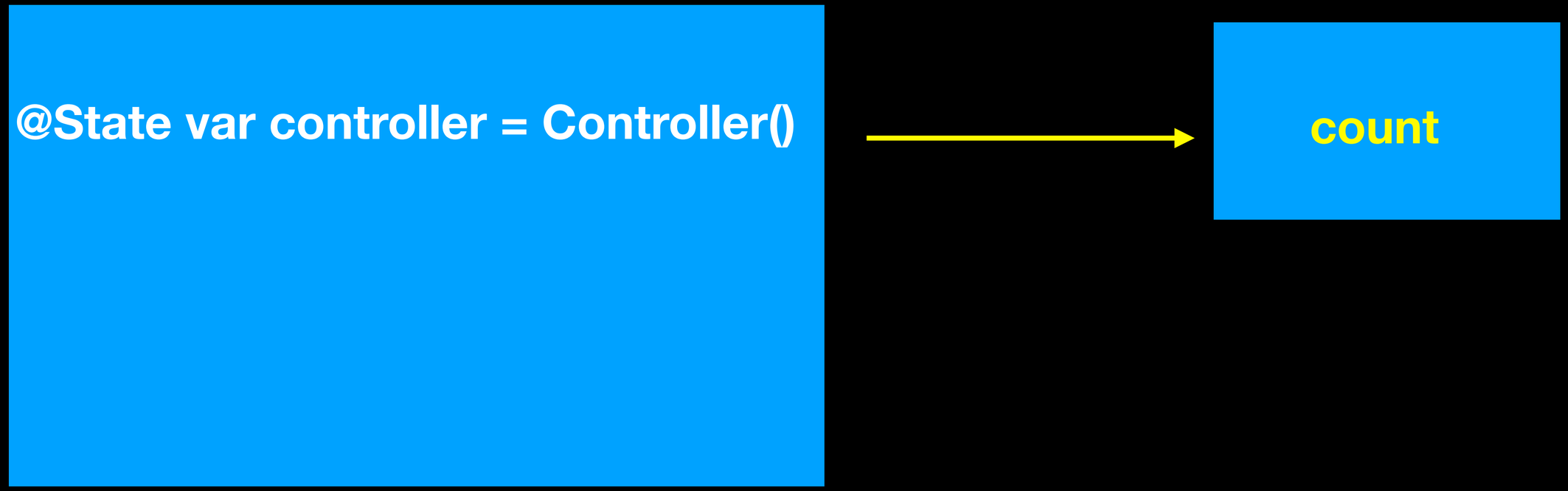
The View has no idea



Controller is a reference type

ContentView

Controller



When count changes

ContentView

Controller

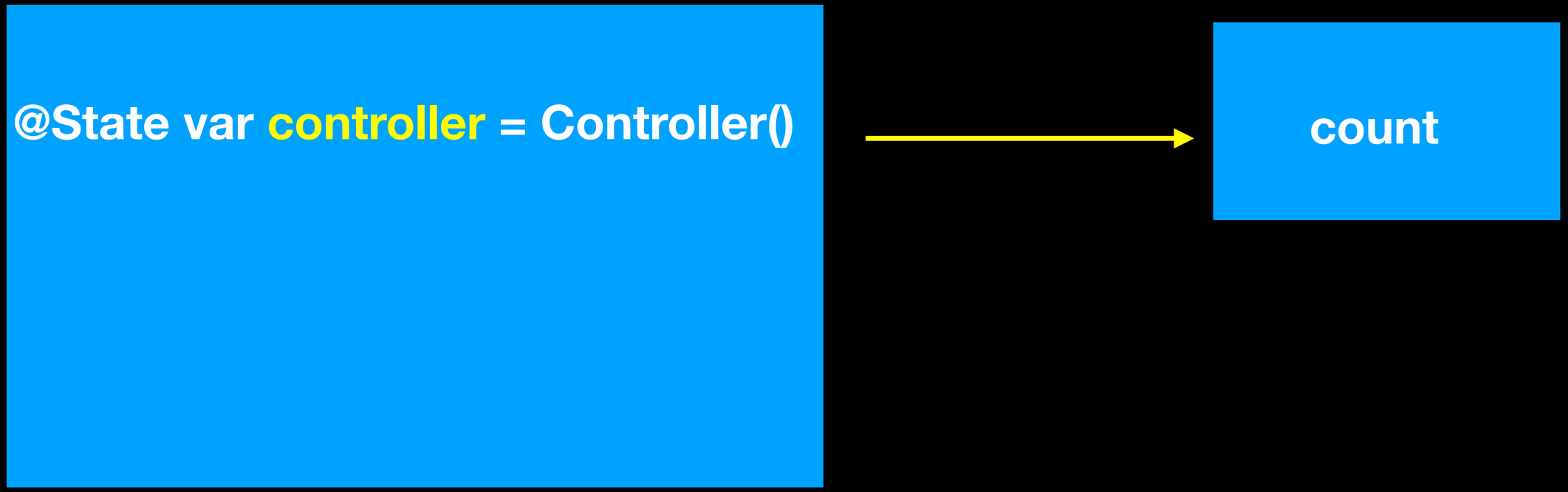
@State var controller = Controller()

count

controller itself doesn't change

ContentView

Controller



controller holds a memory address

ContentView

Controller

@State var controller = Controller()

count

This is where we usually use Combine

ContentView

Controller

@State var controller = Controller()

count

to announce that something will change

```
class Controller                        {
            private(set) var count = 0
}

extension Controller {
  func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

```
class Controller                        {
  @Published private(set) var count = 0
}

extension Controller {
  func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

```
class Controller: ObservableObject {
  @Published private(set) var count = 0
}

extension Controller {
  func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

```
import Combine

class Controller: ObservableObject {
  @Published private(set) var count = 0
}

extension Controller {
  func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

```
import Combine
```

or Foundation or SwiftUI

```
class Controller: ObservableObject {
  @Published private(set) var count = 0
}

extension Controller {
  func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

How does this work?

```
import Combine

class Controller: ObservableObject {
  @Published private(set) var count = 0
}

extension Controller {
  func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

```
import Combine

class Controller: ObservableObject {
  @Published private(set) var count = 0
}

extension Controller {
  func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

```
import Combine

class Controller: ObservableObject {
  @Published private(set) var count = 0
}

extension Controller {
  func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

Controller instance will change

```
import Combine

class Controller: ObservableObject {
  @Published private(set) var count = 0
}

extension Controller {
  func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

Controller instance will change

```
import Combine

class Controller: ObservableObject {
  @Published private(set) var count = 0
}

extension Controller {
  func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

Controller instance will change

```
import Combine

class Controller: ObservableObject {
  @Published private(set) var count = 0
}

extension Controller {
  func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

The change to ContentView is trivial

```
struct ContentView {
  @State       var controller = Controller()
}
```

```
struct ContentView {
  @StateObject var controller = Controller()
}
```

```
struct ContentView {
  @StateObject var controller = Controller()
}
```

ContentView is responsible for controller

```
struct ContentView {
  @StateObject var controller = Controller()
}
```

and receives controller.objectWillChange()

Now the app runs perfectly

Now the app runs perfectly


YOU'RE ABOUT TO RUIN THINGS - AREN'T YOU?

Let's remove Combine

ContentView

Controller

@State var controller = Controller()

count

ContentView

Controller

@State var controller = Controller()

count

```
import Combine

class Controller: ObservableObject {
  @Published private(set) var count = 0
}

extension Controller {
  func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

```
class Controller {
  private(set) var count = 0
}

extension Controller {
  func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

The app is broken again

Introduce Observable

```
class Controller {
  private(set) var count = 0
}

extension Controller {
  func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

```
@Observable
class Controller {
  private(set) var count = 0
}

extension Controller {
  func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

@Observable is a Macro

```
@Observable
class Controller {
  private(set) var count = 0
}

extension Controller {
  func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

```
import Observation

@Observable
class Controller {
  private(set) var count = 0
}

extension Controller {
  func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

```
import Observation

@Observable
class Controller {
  private(set) var count = 0
}

extension Controller {
  func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

Requires initial value

Let's fix the view

```
struct ContentView {
  @StateObject var controller = Controller()
}
```

```
struct ContentView {
  @State var controller = Controller()
}
```

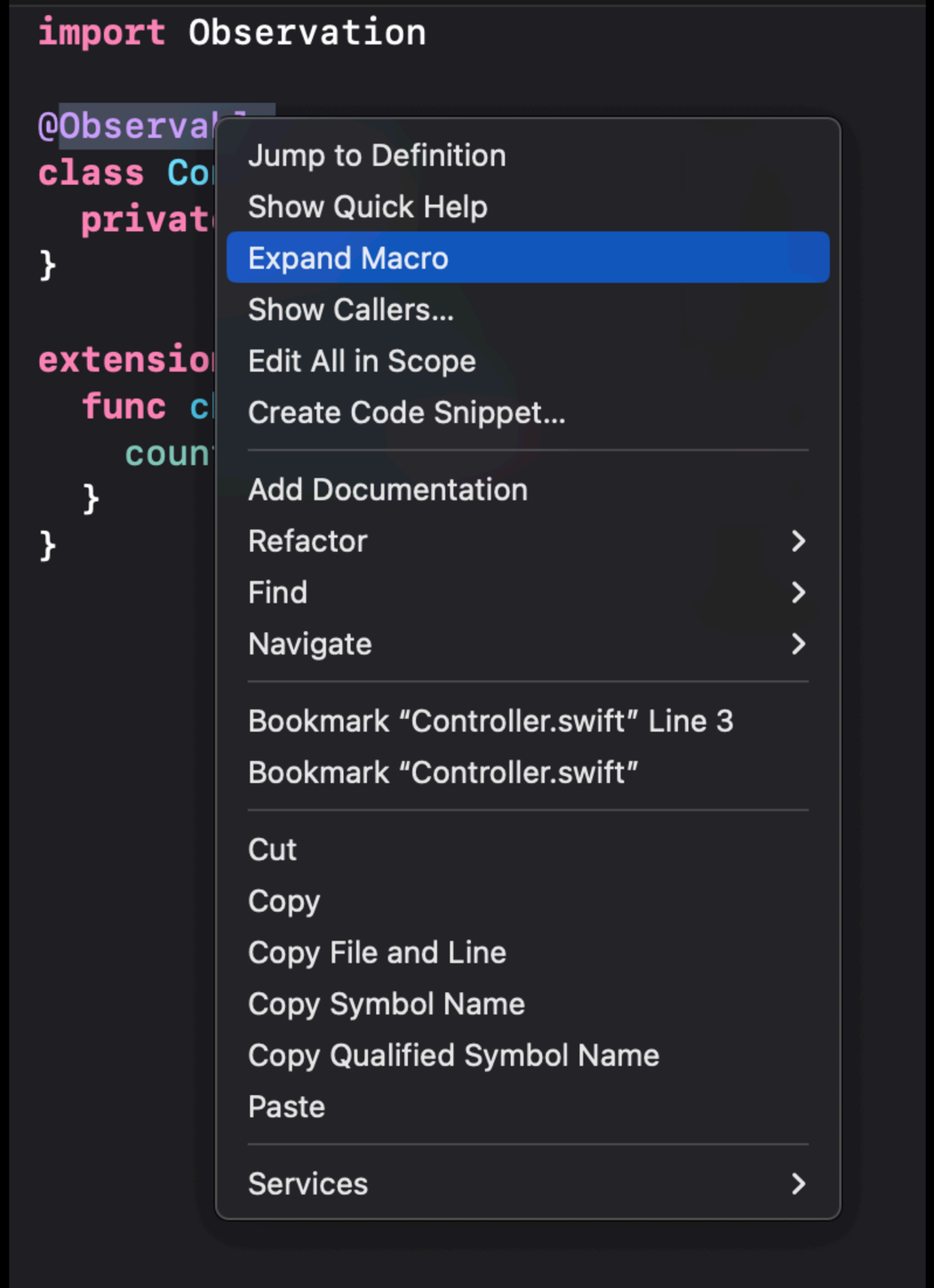
That's it

Run the app

Tap the button

It works perfectly

```
import Observation

@Observable
class Controller {
  private(set) var count = 0
}

extension Controller {
  func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```


import Observation
@Observa
class Co
privat
}
extensio
func c
coun
}
}
Jump to Definition
Show Quick Help
Expand Macro
Show Callers...
Edit All in Scope
Create Code Snippet...
Add Documentation
Refactor
Find
Navigate
Bookmark “Controller.swift” Line 3
Bookmark “Controller.swift”
Cut
Copy
Copy File and Line
Copy Symbol Name
Copy Qualified Symbol Name
Paste
Services

```swift
import Observation

@Observable
class Controller {
  @ObservationTracked
  private(set) var count = 0
  @ObservationIgnored private let _$observationRegistrar = ObservationRegistrar()

  internal nonisolated func access<Member>(
      keyPath: KeyPath<Controller , Member>
  ) {
    _$observationRegistrar.access(self, keyPath: keyPath)
  }

  internal nonisolated func withMutation<Member, T>(
    keyPath: KeyPath<Controller , Member>,
    _ mutation: () throws -> T
  ) rethrows -> T {
    try _$observationRegistrar.withMutation(of: self, keyPath: keyPath, mutation)
  }

  @ObservationIgnored private var _count  = 0
}
extension Controller : Observable  {}

extension Controller {
  func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

```swift
import Observation

@Observable
class Controller {

  private(set) var count = 0

}

extension Controller {
  func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

```swift
import Observation

@Observable
class Controller {
  @ObservationTracked
  private(set) var count = 0
  @ObservationIgnored private let _$observationRegistrar = ObservationRegistrar()

  internal nonisolated func access<Member>(
      keyPath: KeyPath<Controller , Member>
  ) {
    _$observationRegistrar.access(self, keyPath: keyPath)
  }

  internal nonisolated func withMutation<Member, T>(
    keyPath: KeyPath<Controller , Member>,
    _ mutation: () throws -> T
  ) rethrows -> T {
    try _$observationRegistrar.withMutation(of: self, keyPath: keyPath, mutation)
  }

  @ObservationIgnored private var _count  = 0
}
extension Controller : Observable  {}

extension Controller {
  func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

Conformance to the Observable protocol

```
extension Controller: Observation.Observable {
}
```

The count property is marked...

```
@ObservationTracked
private(set) var count = 0
```

The count property is marked...

```
@ObservationTracked
private(set) var count = 0
```

We'll come back to this

A private property is added named _count

```
private var _count = 0
```

A private property is added named _count

```
private var _count = 0
```

Process reminds me a bit of synthesized properties

A private property is added named _count

```
@ObservationIgnored
private var _count = 0
```

_count is not tracked

A registrar is added

```
private let _$observationRegistrar = ObservationRegistrar()
```

It is not tracked

```
@ObservationIgnored
private let _$observationRegistrar = ObservationRegistrar()
```

gets are registered

```
internal nonisolated func access<Member>(
    keyPath: KeyPath<Controller , Member>
) {

}
```

gets are registered

```
internal nonisolated func access<Member>(
    keyPath: KeyPath<Controller , Member>
) {
  _$observationRegistrar.access(self, keyPath: keyPath)
}
```

sets are registered

```
internal nonisolated func withMutation<Member, MutationResult>(
  keyPath: KeyPath<Controller , Member>,
  _ mutation: () throws -> MutationResult
) rethrows -> MutationResult {


}
```

sets are registered

```
internal nonisolated func withMutation<Member, MutationResult>(
  keyPath: KeyPath<Controller , Member>,
  _ mutation: () throws -> MutationResult
) rethrows -> MutationResult {
  try _$observationRegistrar
       .withMutation(of: self, keyPath: keyPath, mutation)
}
```

How?

We come back to this

```
@ObservationTracked
private(set) var count = 0
```

We come back to this

```
@ObservationTracked
private(set) var count = 0
```

@ObservationTracked is a macro

Omitting some details...

```
get {
  access(keyPath: \.count )
  return _count
}
```

Omitting some details...

```
get {
  access(keyPath: \.count )
  return _count
}

set {
  withMutation(keyPath: \.count ) {
    _count  = newValue
  }
}
```

```swift
import Observation

@Observable
class Controller {
  @ObservationTracked
  private(set) var count = 0
  @ObservationIgnored private let _$observationRegistrar = ObservationRegistrar()

  internal nonisolated func access<Member>(
      keyPath: KeyPath<Controller , Member>
  ) {
    _$observationRegistrar.access(self, keyPath: keyPath)
  }

  internal nonisolated func withMutation<Member, T>(
    keyPath: KeyPath<Controller , Member>,
    _ mutation: () throws -> T
  ) rethrows -> T {
    try _$observationRegistrar.withMutation(of: self, keyPath: keyPath, mutation)
  }

  @ObservationIgnored private var _count  = 0
}
extension Controller : Observable  {}

extension Controller {
  func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

```
import Observation

@Observable
class Controller {
  private(set) var count = 0
}

extension Controller {
  func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

stored properties

```
import Observation

@Observable
class Controller {
  private(set) var count = 0
}

extension Controller {
  func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

Computed properties?

```
import Observation

@Observable
class Controller {
  private var count = 0
}
```

```
extension Controller {
  func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

```
import Observation

@Observable
class Controller {
  private var count = 0
}

extension Controller {



}

extension Controller {
  func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

```
import Observation

@Observable
class Controller {
  private var count = 0
}

extension Controller {
  var annotatedCount: String {

  }
}

extension Controller {
  func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

```
import Observation

@Observable
class Controller {
  private var count = 0
}

extension Controller {
  var annotatedCount: String {
    "The controller's count is \(count)"
  }
}

extension Controller {
  func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

```
import Observation

@Observable
class Controller {
  private var count = 0
}

extension Controller {
  var annotatedCount: String {
    "The controller's count is \(count)"
  }
}

extension Controller {
  func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

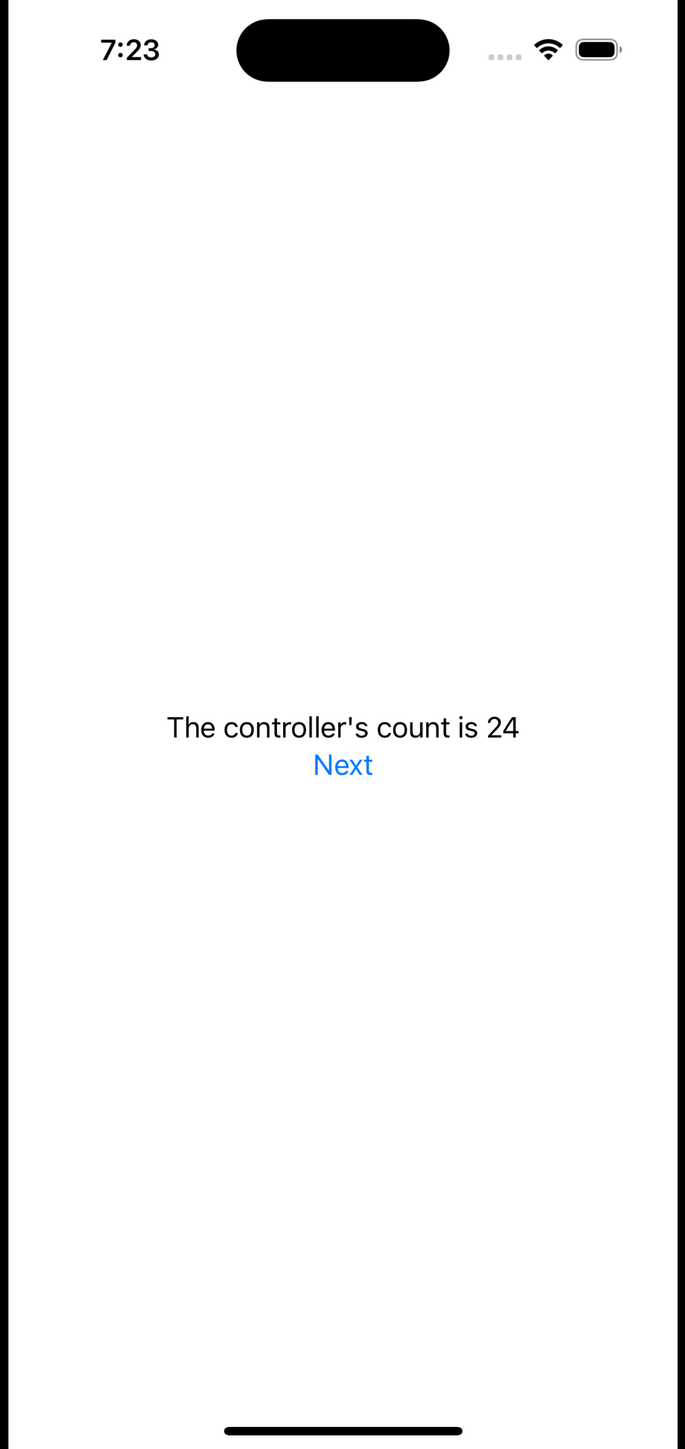
```
import Observation

@Observable
class Controller {
  private var count = 0
}

extension Controller {
  var annotatedCount: String {
    "The controller's count is \(count)"
  }
}

extension Controller {
  func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

```
import Observation

@Observable
class Controller {
  private var count = 0
}

extension Controller {
  var annotatedCount: String {
    "The controller's count is \(count)"
  }
}

extension Controller {
  func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

```
import Observation

@Observable
class Controller {
  private var count = 0
}

extension Controller {
  var annotatedCount: String {
    "The controller's count is \(count)"
  }
}

extension Controller {
  func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

Adjust the view

```
extension ContentView: View {
  var body: some View {
    VStack {
      Text(controller.annotatedCount)
      Button("Next") {
        controller.changeCount()
      }
    }
  }
}
```


7:23
The controller's count is 24
Next

Let's separate our code further

This doesn't look like controller code

```
import Observation

@Observable
class Controller {
  private var count = 0
}

extension Controller {
  var annotatedCount: String {
    "The controller's count is \(count)"
  }
}

extension Controller {
  func changeCount() {
    count = Int.random(in: 1...100)
  }
}
```

Maybe we should introduce a model

ContentView

Controller

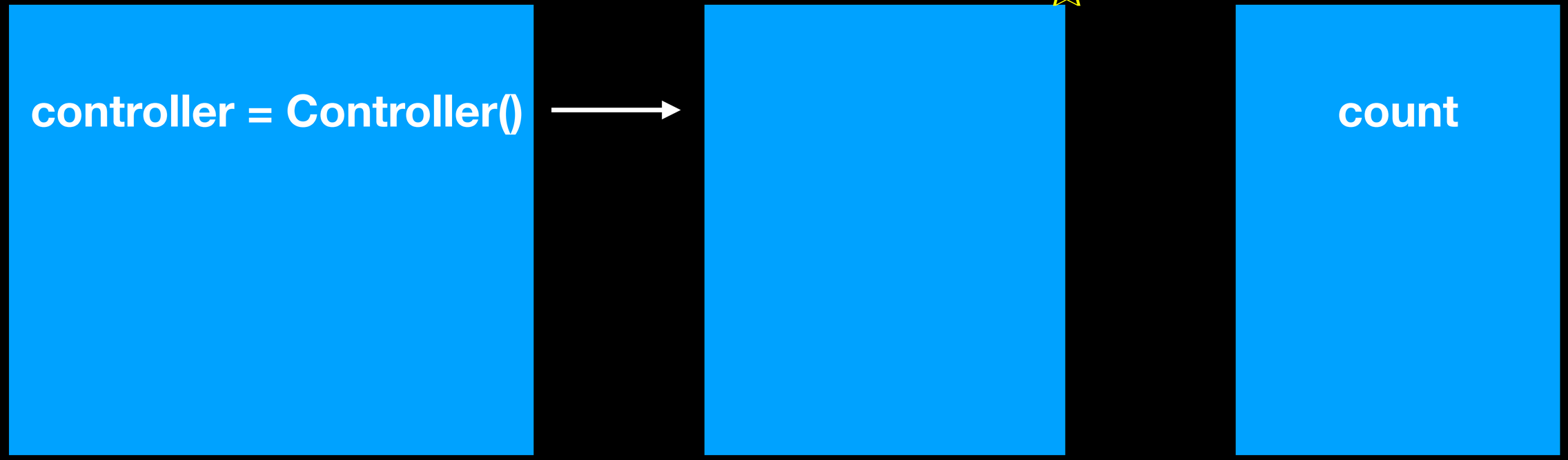
controller = Controller()

count

ContentView

Controller

Model



ContentView

Controller

Model

controller = Controller()

model = Model()

count

```
class Model {

}
```

```
class Model {
  private(set) var count = 0
}
```

```
class Model {
  private(set) var count = 0
}

extension Model {



}
```

```
class Model {
  private(set) var count = 0
}

extension Model {
  func updateCount() {

  }
}
```

```
class Model {
  private(set) var count = 0
}

extension Model {
  func updateCount() {
    count = Int.random(in: 1...100)
  }
}
```

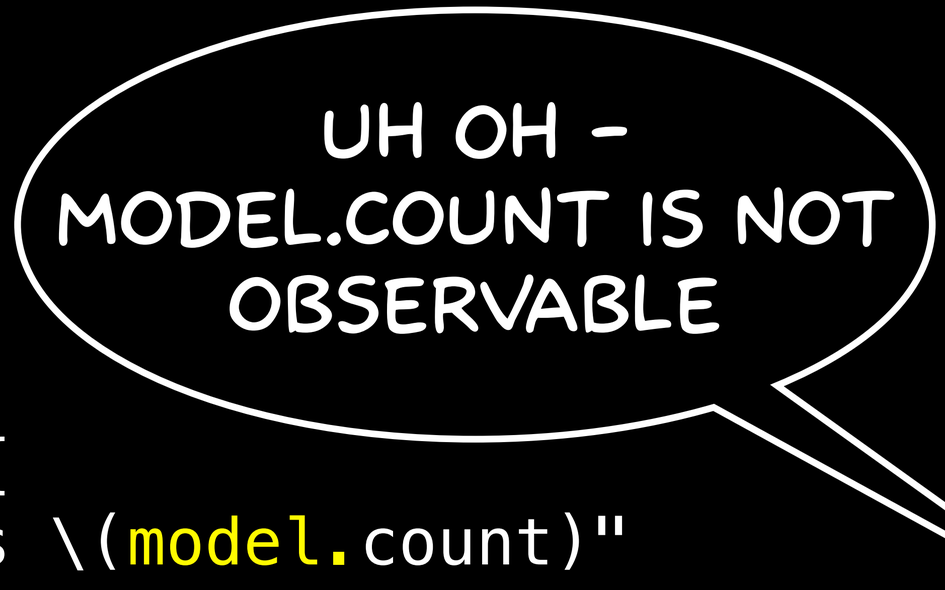
Modify the Controller to use the model

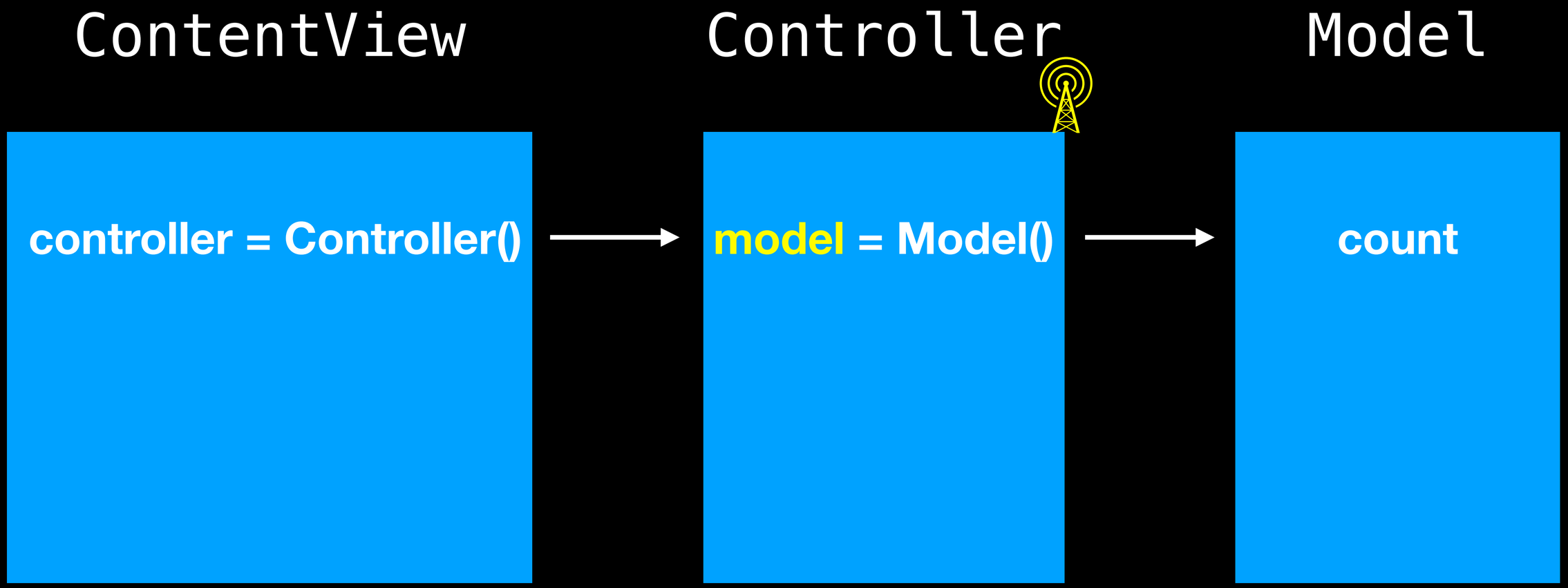
```
import Observation

@Observable
class Controller {
  private var model = Model()
}

extension Controller {
  var annotatedCount: String {
    "The controller's count is \(       count)"
  }
}

extension Controller {
  func changeCount() {

  }
}
```

```
import Observation

@Observable
class Controller {
  private var model = Model()
}

extension Controller {
  var annotatedCount: String {
    "The controller's count is \(       count)"
  }
}

extension Controller {
  func changeCount() {
    model.updateCount()
  }
}
```

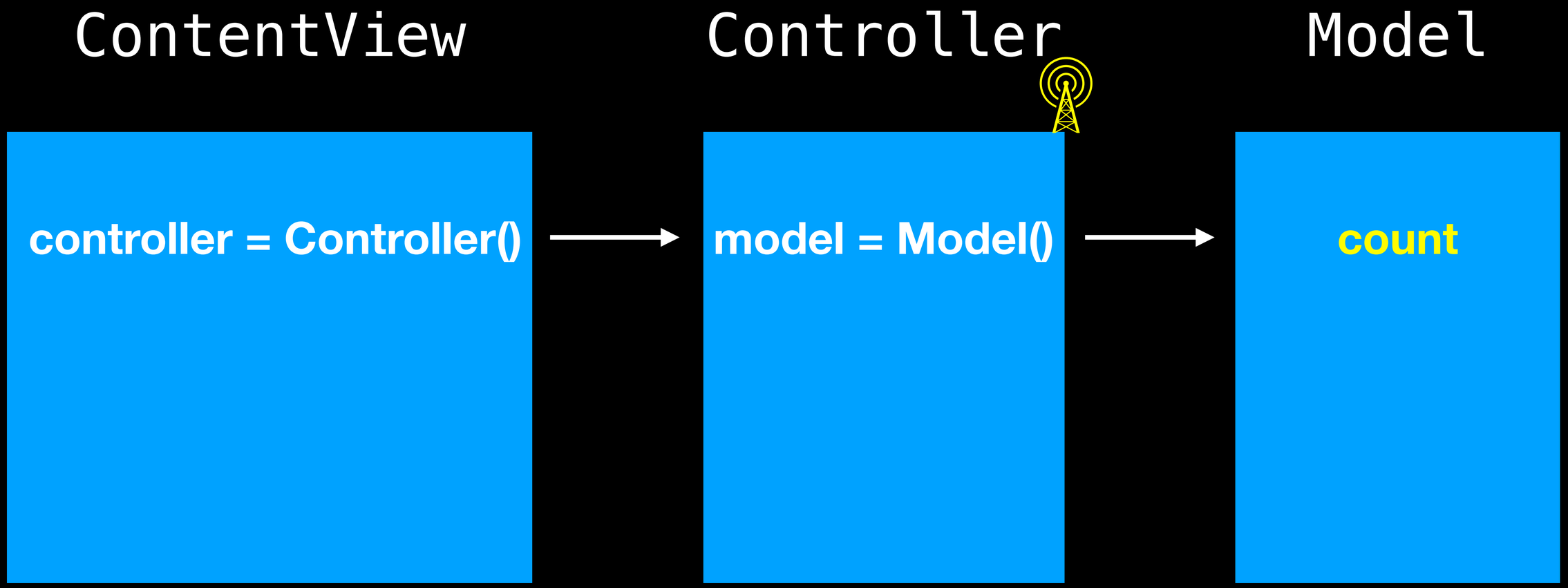
```
import Observation

@Observable
class Controller {
  private var model = Model()
}

extension Controller {
  var annotatedCount: String {
    "The controller's count is \(model.count)"
  }
}

extension Controller {
  func changeCount() {
    model.updateCount()
  }
}
```

```
import Observation

@Observable
class Controller {
  private var model = Model()
}

extension Controller {
  var annotatedCount: String {
    "The controller's count is \(model.count)"
  }
}

extension Controller {
  func changeCount() {
    model.updateCount()
  }
}
```

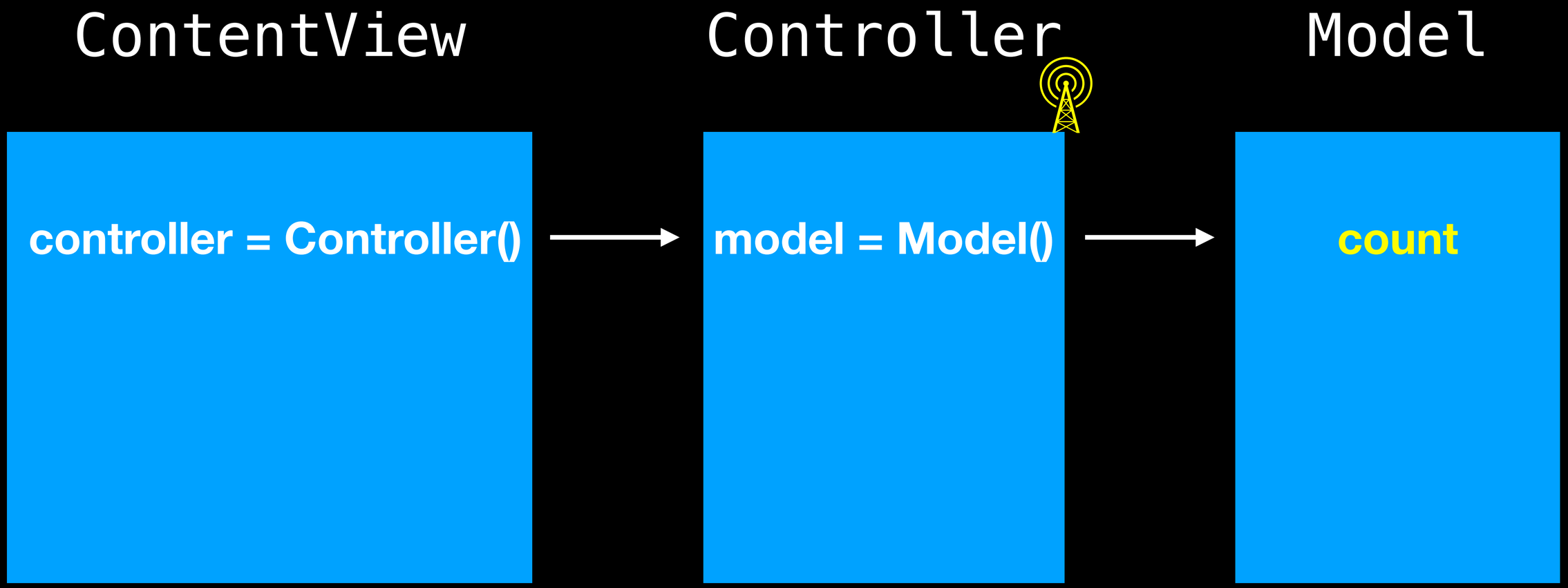




Model is a reference type

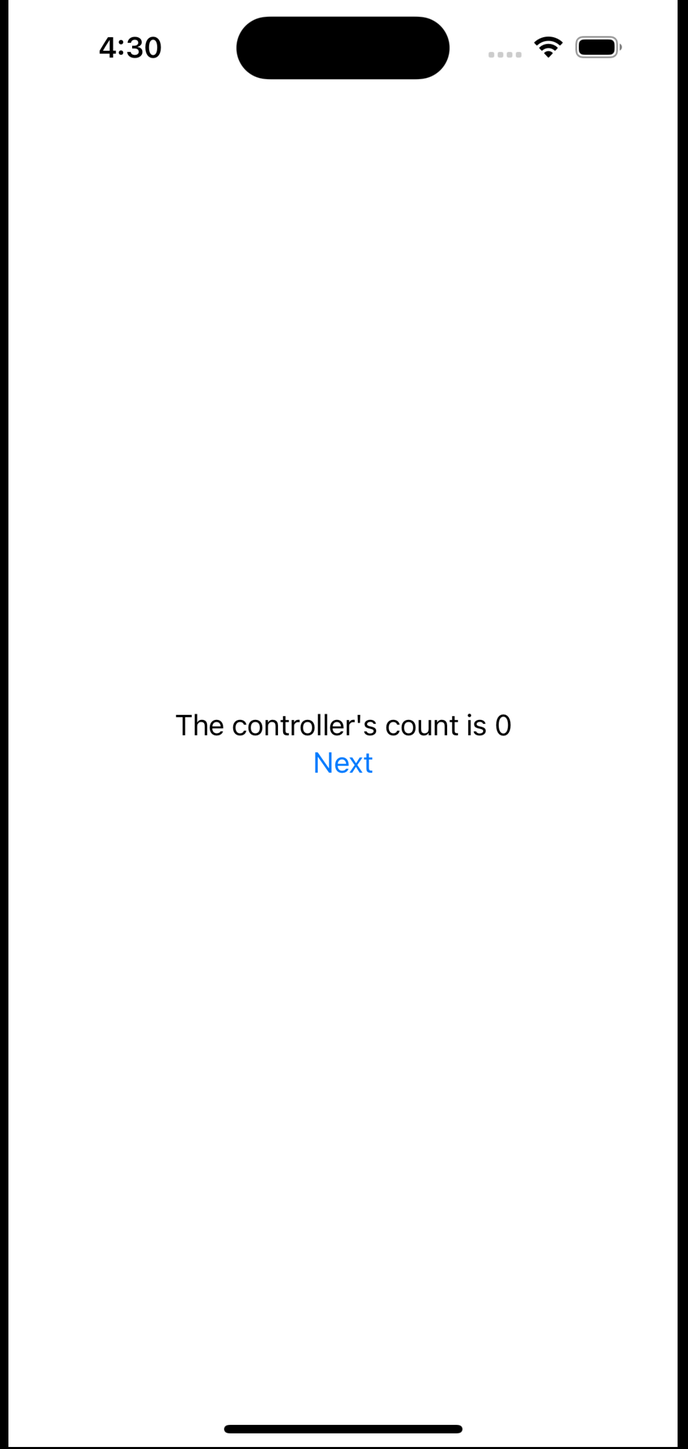


it doesn't change when count changes



we aren't announcing changes to count

The app launches fine


4:30
The controller's count is 0
Next

It never updates

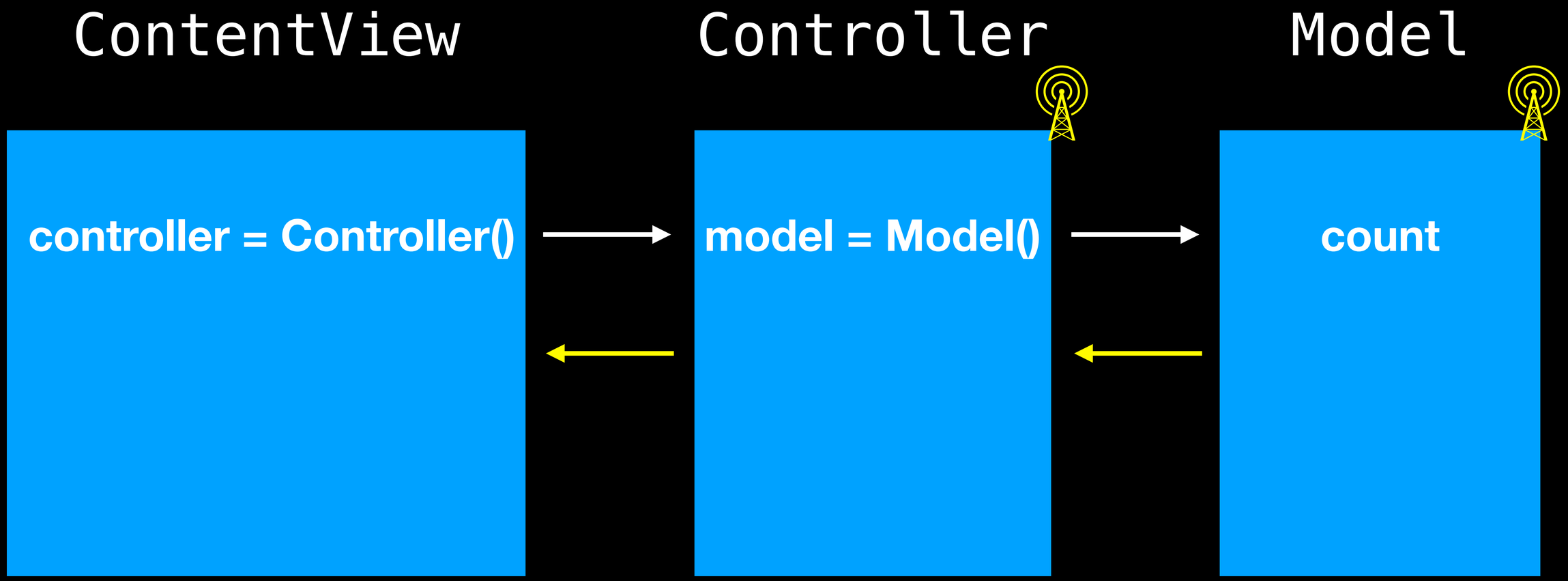
How could it?

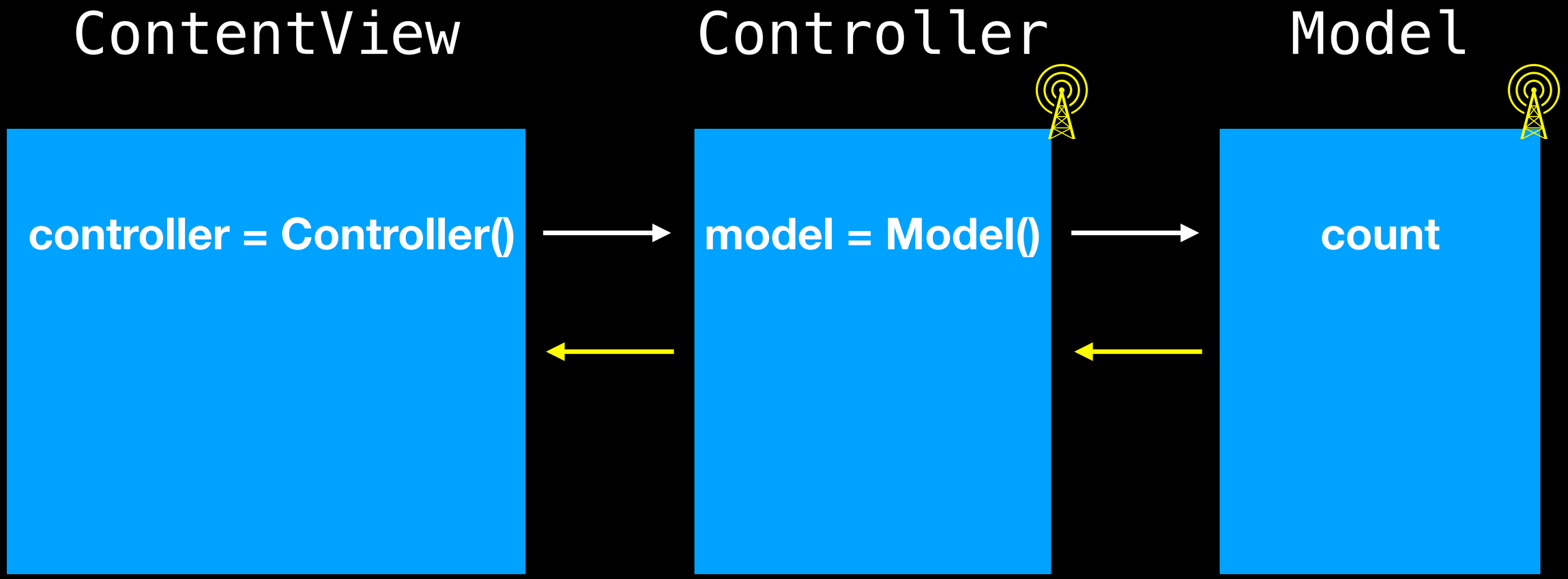
Let's make Model @Observable

```
@Observable
class Model {
  private(set) var count = 0
}

extension Model {
  func updateCount() {
    count = Int.random(in: 1...100)
  }
}
```

```
import Observation

@Observable
class Model {
  private(set) var count = 0
}

extension Model {
  func updateCount() {
    count = Int.random(in: 1...100)
  }
}
```

Now it updates fine


ContentView
Controller
Model
controller = Controller()
model = Model()
count



There's more going on than meets the eye

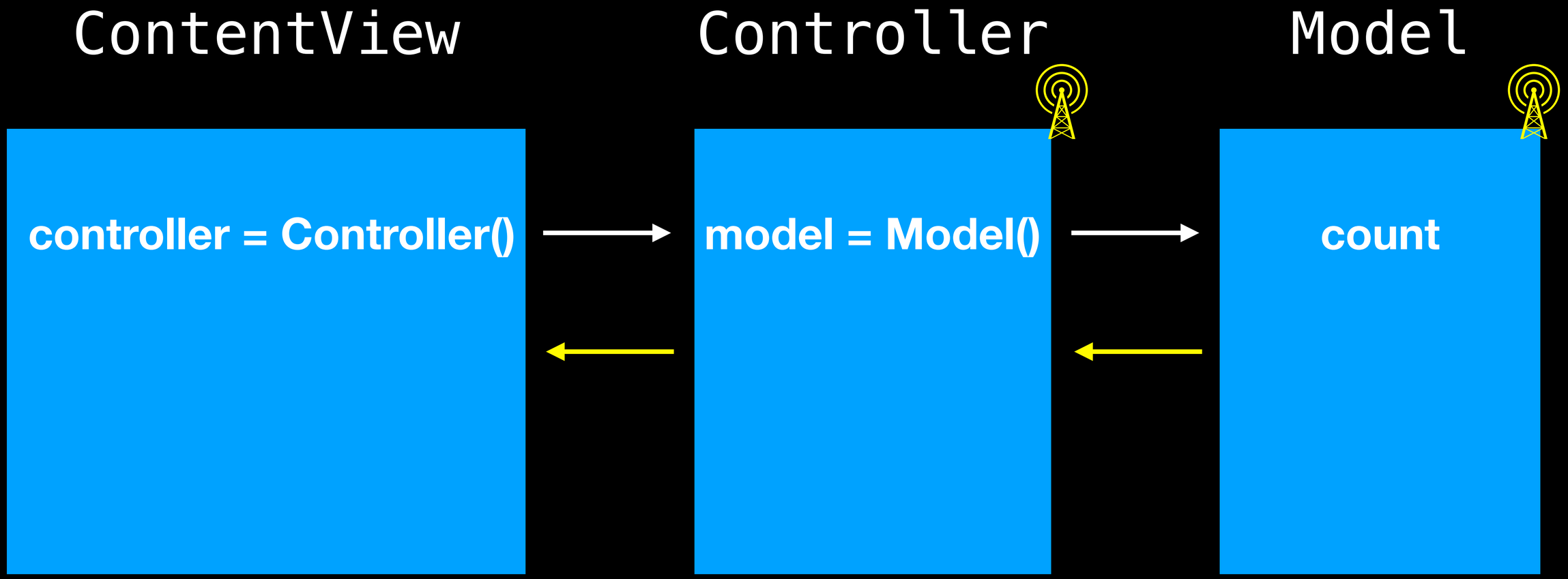
In fact...

This might blow your mind

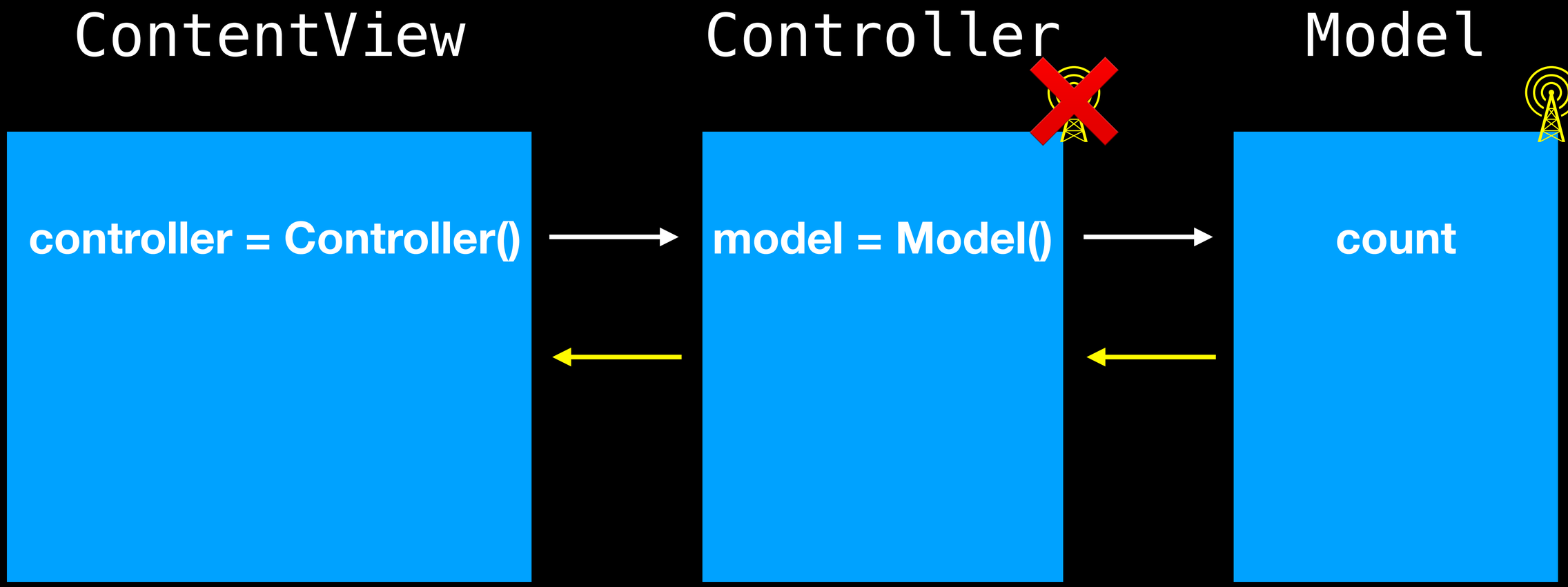
If Model is Observable

We get updates to
controller's computed property

Even if Controller isn't Observable



There's more going on than meets the eye



There's more going on than meets the eye

```
~~import Observation~~

~~@Observable~~
class Controller {
  private var model = Model()
}

extension Controller {
  var annotatedCount: String {
    "The controller's count is \(model.count)"
  }
}
```

```
class Controller {
  private var model = Model()
}

extension Controller {
  var annotatedCount: String {
    "The controller's count is \(model.count)"
  }
}
```

Even if model is a let

```
class Controller {
  let model = Model()
}

extension Controller {
  var annotatedCount: String {
    "The controller's count is \(model.count)"
  }
}
```

Even if Controller is a struct

```
struct Controller {
  let model = Model()
}

extension Controller {
  var annotatedCount: String {
    "The controller's count is \(model.count)"
  }
}
```

It still works

Let's run an experiment to investigate why

```
struct Controller {
  let model = Model()

}

extension Controller {
  var annotatedCount: String {
    "The controller's count is \(model.count)"
  }









}
```

```
struct Controller {
  let model = Model()

}

extension Controller {
  var annotatedCount: String {
    "The controller's count is \(model.count)"
  }

  private func investigation() {




  }
}
```

```
import Observation

struct Controller {
  let model = Model()

}

extension Controller {
  var annotatedCount: String {
    "The controller's count is \(model.count)"
  }

  private func investigation() {




  }
}
```

```
import Observation

struct Controller {
  let model = Model()

}

extension Controller {
  var annotatedCount: String {
    "The controller's count is \(model.count)"
  }

  private func investigation() {
    withObservationTracking {

    } onChange: {

    }
  }
}
```

```
withObservationTracking {

} onChange: {

}
```

```
withObservationTracking {
  // monitor these things for changes
} onChange: {

}
```

```
withObservationTracking {
  // monitor these things for changes
} onChange: {
  // perform these things
}
```

```
import Observation

struct Controller {
  let model = Model()

}

extension Controller {
  var annotatedCount: String {
    "The controller's count is \(model.count)"
  }

  private func investigation() {
    withObservationTracking {
              model.count
    } onChange: {

    }
  }
}
```

```
import Observation

struct Controller {
  let model = Model()

}

extension Controller {
  var annotatedCount: String {
    "The controller's count is \(model.count)"
  }

  private func investigation() {
    withObservationTracking {
              model.count
    } onChange: {

    }
  }
}
```

```
import Observation

struct Controller {
  let model = Model()

}

extension Controller {
  var annotatedCount: String {
    "The controller's count is \(model.count)"
  }

  private func investigation() {
    withObservationTracking {
      let _ = model.count
    } onChange: {

    }
  }
}
```

```
import Observation

struct Controller {
  let model = Model()

}

extension Controller {
  var annotatedCount: String {
    "The controller's count is \(model.count)"
  }

  private func investigation() {
    withObservationTracking {
      let _ = model.count
    } onChange: {
      print("count = \(model.count)")
    }
  }
}
```

```
import Observation

struct Controller {
  let model = Model()
  init() { investigation()}
}

extension Controller {
  var annotatedCount: String {
    "The controller's count is \(model.count)"
  }

  private func investigation() {
    withObservationTracking {
      let _ = model.count
    } onChange: {
      print("count = \(model.count)")
    }
  }
}
```

```
import Observation

struct Controller {
  let model = Model()
  init() { investigation()}
}

extension Controller {
  var annotatedCount: String {
    "The controller's count is \(model.count)"
  }

  private func investigation() {
    withObservationTracking {
      let _ = model.count
    } onChange: {
      print("count = \(model.count)")
    }
  }
}
```

On Screen

In Console

The controller's count is 0

On Screen

In Console

The controller's count is 0

The controller's count is 39

```
count = 0
```

On Screen

In Console

The controller's count is 0

The controller's count is 39

```
count = 0
```

The controller's count is 65

On Screen

In Console

The controller's count is 0

The controller's count is 39

```
count = 0
```

The controller's count is 65

The controller's count is 84

On Screen

In Console

The controller's count is 0

The controller's count is 39

```
count = 0
```

The controller's count is 65

The controller's count is 84

The controller's count is 27

```
extension Controller {
  var annotatedCount: String {
    "The controller's count is \(model.count)"
  }

  private func investigation() {
    withObservationTracking {
      let _ = model.count
    } onChange: {
      print("count = \(model.count)")

    }
  }
}
```

On Screen	In Console
The controller's count is 0	prints once
The controller's count is 39	`count = 0`
The controller's count is 65	
The controller's count is 84	
The controller's count is 27	

On Screen

In Console

The controller's count is 0

prints once

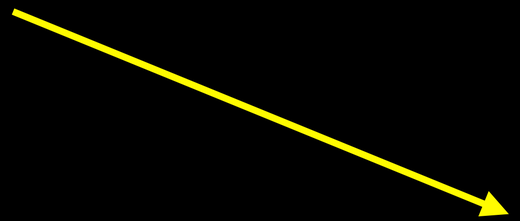
The controller's count is 39

`count = 0`

The controller's count is 65

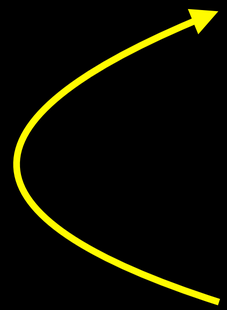
The controller's count is 84

The controller's count is 27



```
extension Controller {
  var annotatedCount: String {
    "The controller's count is \(model.count)"
  }

  private func investigation() {
    withObservationTracking {
      let _ = model.count
    } onChange: {
      print("count = \(model.count)")

    }
  }
}
```

```
extension Controller {
  var annotatedCount: String {
    "The controller's count is \(model.count)"
  }

  private func investigation() {
    withObservationTracking {
      let _ = model.count
    } onChange: {
      print("count = \(model.count)")
      investigation()
    }
  }
}
```



```
extension Controller {
  var annotatedCount: String {
    "The controller's count is \(model.count)"
  }

  private func investigation() {
    withObservationTracking {
      let _ = model.count
    } onChange: {
      print("count = \(model.count)")
      investigation()
    }
  }
}
```

On Screen

In Console

The controller's count is 0

On Screen

In Console

The controller's count is 0

The controller's count is 39

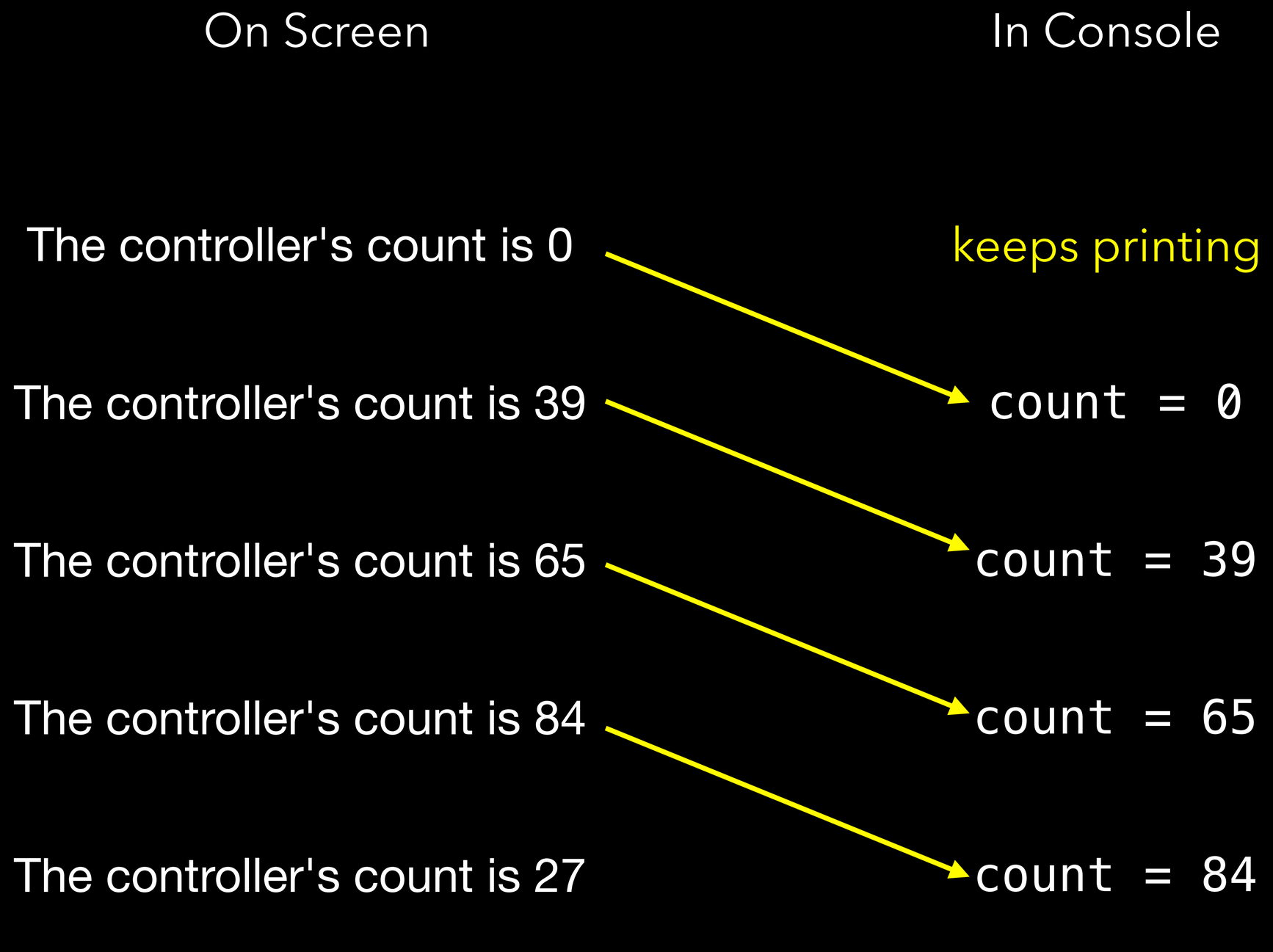
```
count = 0
```

On Screen	In Console
The controller's count is 0	
The controller's count is 39	`count = 0`
The controller's count is 65	`count = 39`

On Screen	In Console
The controller's count is 0	
The controller's count is 39	`count = 0`
The controller's count is 65	`count = 39`
The controller's count is 84	`count = 65`

On Screen	In Console
The controller's count is 0	
The controller's count is 39	`count = 0`
The controller's count is 65	`count = 39`
The controller's count is 84	`count = 65`
The controller's count is 27	`count = 84`

On Screen	In Console
The controller's count is 0	keeps printing
The controller's count is 39	`count = 0`
The controller's count is 65	`count = 39`
The controller's count is 84	`count = 65`
The controller's count is 27	`count = 84`


On Screen
In Console
The controller's count is 0
keeps printing
The controller's count is 39
count = 0
The controller's count is 65
count = 39
The controller's count is 84
count = 65
The controller's count is 27
count = 84

Generally, we shouldn't use withObservationTracking onChange

Generally, we shouldn't use
withObservationTracking onChange
it's always behind

You know who likes to know when something will change?

SwiftUI

```
extension ContentView: View {
  var body: some View {
    VStack {
      Text(controller.annotatedCount)
      Button("Next") {
        controller.model.updateCount()
      }
    }
  }
}
```

How does this update?

```
extension ContentView: View {
  var body: some View {
    VStack {
      Text(controller.annotatedCount)
      Button("Next") {
        controller.model.updateCount()
      }
    }
  }
}
```

```
Text(controller.annotatedCount)
```

```
Text(controller.annotatedCount)
```

```
var annotatedCount: String {
  "The controller's count is \(model.count)"
}
```

```
Text(controller.annotatedCount)
```

```
var annotatedCount: String {
  "The controller's count is \(model.count)"
}
```

```
withObservationTracking {
  let _ = model.count
}
```

```
withObservationTracking {
  let _ = model.count
}
```

Even though ContentView can't see model
or model's count

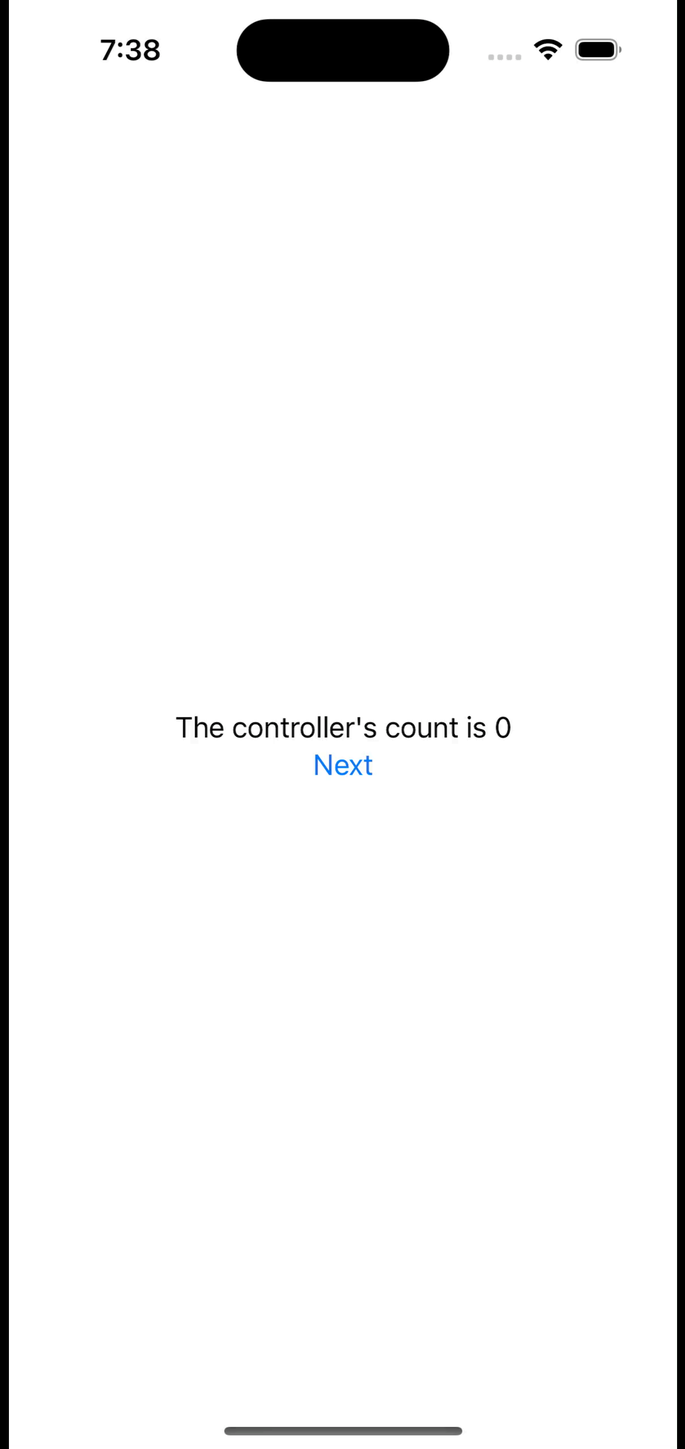
```
withObservationTracking {
  let _ = model.count
}
```

SwiftUI checks a global registry of Observable objects

```
withObservationTracking {
  let _ = model.count
} onChange: {
   Text(controller.annotatedCount)
}
```

because annotatedCount depends on model.count

Magic


7:38
The controller's count is 0
Next

More magic

```
import SwiftUI

struct ContentView {
  @State var controller = Controller()
}

extension ContentView: View {
  var body: some View {
    VStack {
      Text(controller.annotatedCount)
      Button("Next") {
        controller.model.updateCount()
      }
    }
  }
}
```

```
import SwiftUI

struct ContentView {
  @State var controller = Controller()
}

extension ContentView: View {
  var body: some View {
    VStack {
      Text(controller.annotatedCount)
      Button("Next") {
        controller.model.updateCount()
      }
    }
  }
}
```

```
import SwiftUI

struct ContentView {
        var controller = Controller()
}

extension ContentView: View {
  var body: some View {
    VStack {
      Text(controller.annotatedCount)
      Button("Next") {
        controller.model.updateCount()
      }
    }
  }
}
```

```
import SwiftUI

struct ContentView {
  let controller: Controller
}

extension ContentView: View {
  var body: some View {
    VStack {
      Text(controller.annotatedCount)
      Button("Next") {
        controller.model.updateCount()
      }
    }
  }
}
```

This controller never changes

This controller never changes

```
struct ContentView {
  let controller: Controller
}
```

This controller never changes
controller is a let

```
struct ContentView {
  let controller: Controller
}
```

This controller never changes

controller is a let

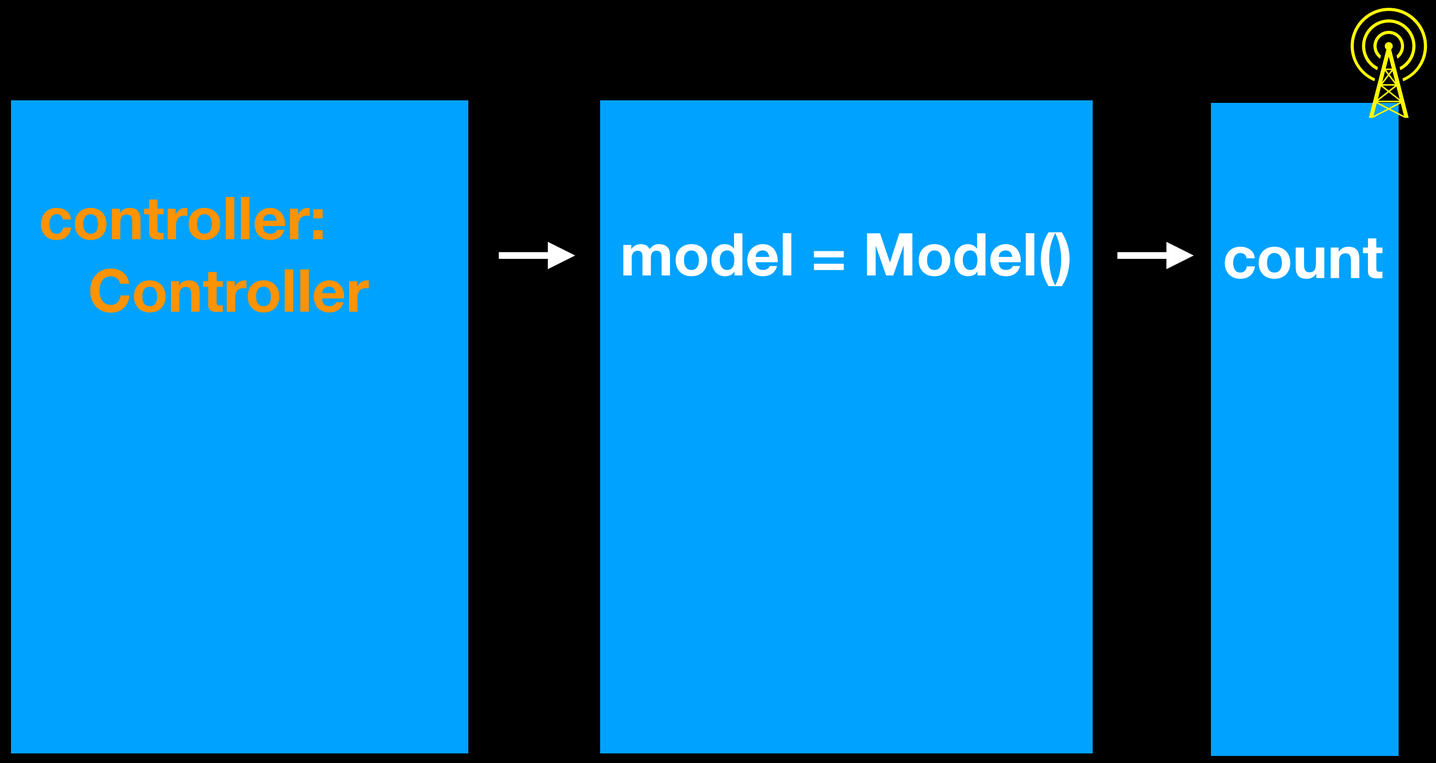
Controller is a struct

```
struct ContentView {
  let controller: Controller
}
```

Controller must be passed in

```
struct ContentView {
  let controller: Controller
}
```

ContentView Controller Model



TestingApp

ContentView

Controller

Model

contentView
= ContentView()

controller:
Controller

model = Model()

count

TestingApp

ContentView

Controller

Model

controller = Controller()

contentView
= ContentView()

controller:
Controller

model = Model()

count

TestingApp ContentView Controller Model

```
controller = Controller()

contentView
  = ContentView(controller:
                controller)
```

→

```
controller:
  Controller
```

→

```
model = Model()
```

→

```
count
```

```
import SwiftUI

@main
struct TestingApp {

}

extension TestingApp: App {
  var body: some Scene {
    WindowGroup {
      ContentView()
    }
  }
}
```

```
import SwiftUI

@main
struct TestingApp {
  @State private var controller = Controller()
}

extension TestingApp: App {
  var body: some Scene {
    WindowGroup {
      ContentView()
    }
  }
}
```

```
import SwiftUI

@main
struct TestingApp {
  @State private var controller = Controller()
}

extension TestingApp: App {
  var body: some Scene {
    WindowGroup {
      ContentView(controller: controller)
    }
  }
}
```

I find this magical

TestingApp ContentView Controller Model

controller = Controller()

contentView
= ContentView(controller:
controller)

controller
= Controller()

model = Model()

count

controller never changes

```
import SwiftUI

struct ContentView {
  let controller: Controller
}

extension ContentView: View {
  var body: some View {
    VStack {
      Text(controller.annotatedCount)
      Button("Next") {
        controller.model.updateCount()
      }
    }
  }
}
```

controller.model never changes

```
import SwiftUI

struct ContentView {
  let controller: Controller
}

extension ContentView: View {
  var body: some View {
    VStack {
      Text(controller.annotatedCount)
      Button("Next") {
        controller.model.updateCount()
      }
    }
  }
}
```

controller.model is observable

```
import SwiftUI

struct ContentView {
  let controller: Controller
}

extension ContentView: View {
  var body: some View {
    VStack {
      Text(controller.annotatedCount)
      Button("Next") {
        controller.model.updateCount()
      }
    }
  }
}
```

controller.model.count changes

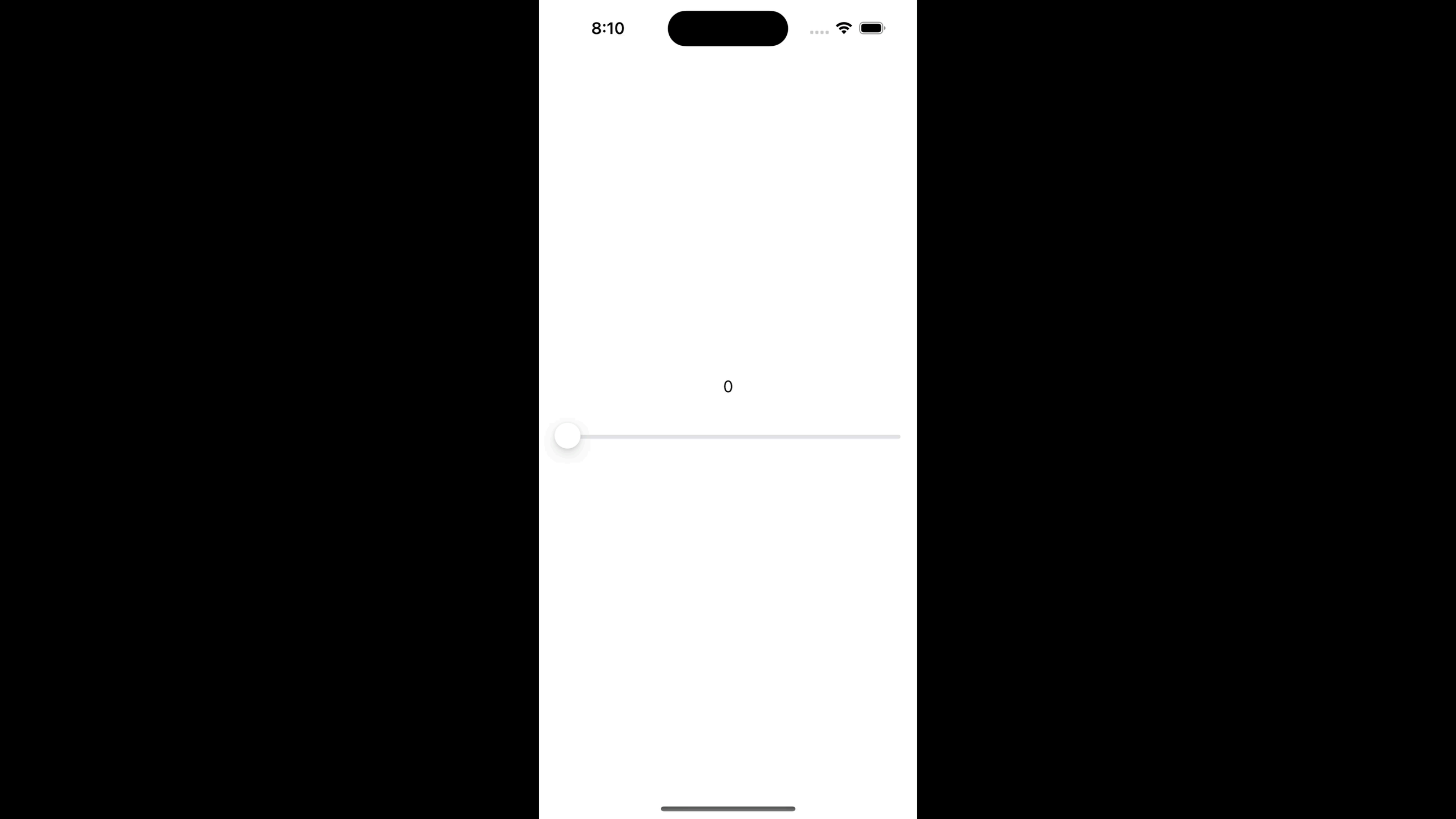
```
import SwiftUI

struct ContentView {
  let controller: Controller
}

extension ContentView: View {
  var body: some View {
    VStack {
      Text(controller.annotatedCount)
      Button("Next") {
        controller.model.updateCount()
      }
    }
  }
}
```

controller.model.count changes

```
import SwiftUI

struct ContentView {
  let controller: Controller
}

extension ContentView: View {
  var body: some View {
    VStack {
      Text(controller.annotatedCount)
      Button("Next") {
        controller.model.updateCount()
      }
    }
  }
}
```

This slide intentionally left blank

For our last trick -
count will be a Double


8:10
0

Simplify Model

```
import Observation

@Observable
class Model {
  var count = 0.0
}
```

Simplify Controller

```
struct Controller {
  let model = Model()
}

extension Controller {
  var annotatedCount: String {

  }
}
```

```
struct Controller {
  let model = Model()
}

extension Controller {
  var annotatedCount: String {
        model.count
  }
}
```

```
struct Controller {
  let model = Model()
}

extension Controller {
  var annotatedCount: String {
        model.count.rounded()
  }
}
```

```
struct Controller {
  let model = Model()
}

extension Controller {
  var annotatedCount: String {
    Int(model.count.rounded()).description
  }
}
```

Create a slider view

```
import SwiftUI

struct ValueSlider {

}

extension ValueSlider: View {
  var body: some View {
    Slider(
            in: 0...100)
    .padding()
  }
}
```

```
import SwiftUI

struct ValueSlider {

}

extension ValueSlider: View {
  var body: some View {
    Slider(value:              ,
           in: 0...100)
    .padding()
  }
}
```

```
import SwiftUI

struct ValueSlider {

}

extension ValueSlider: View {
  var body: some View {
    Slider(value:                ,
           in: 0...100)
    .padding()
  }
}
```

TestingApp ContentView Controller Model

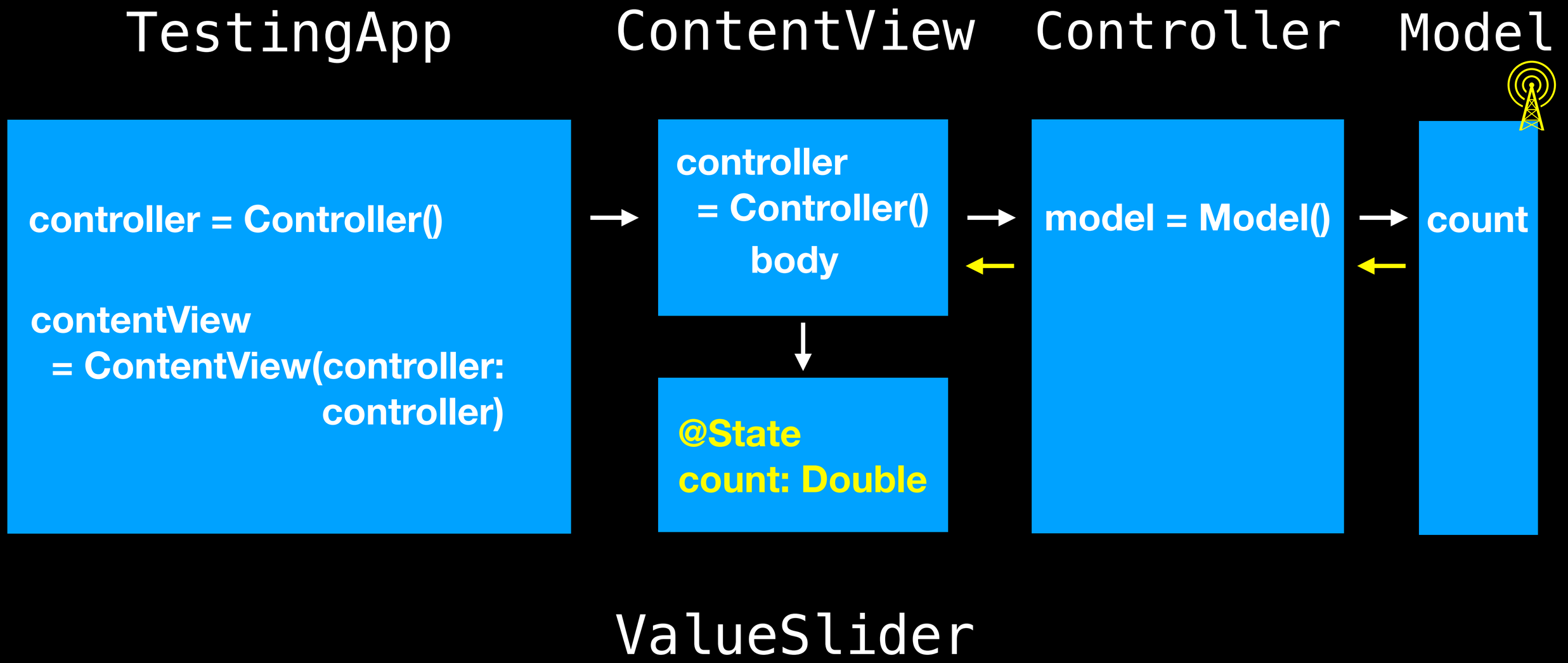
controller = Controller()

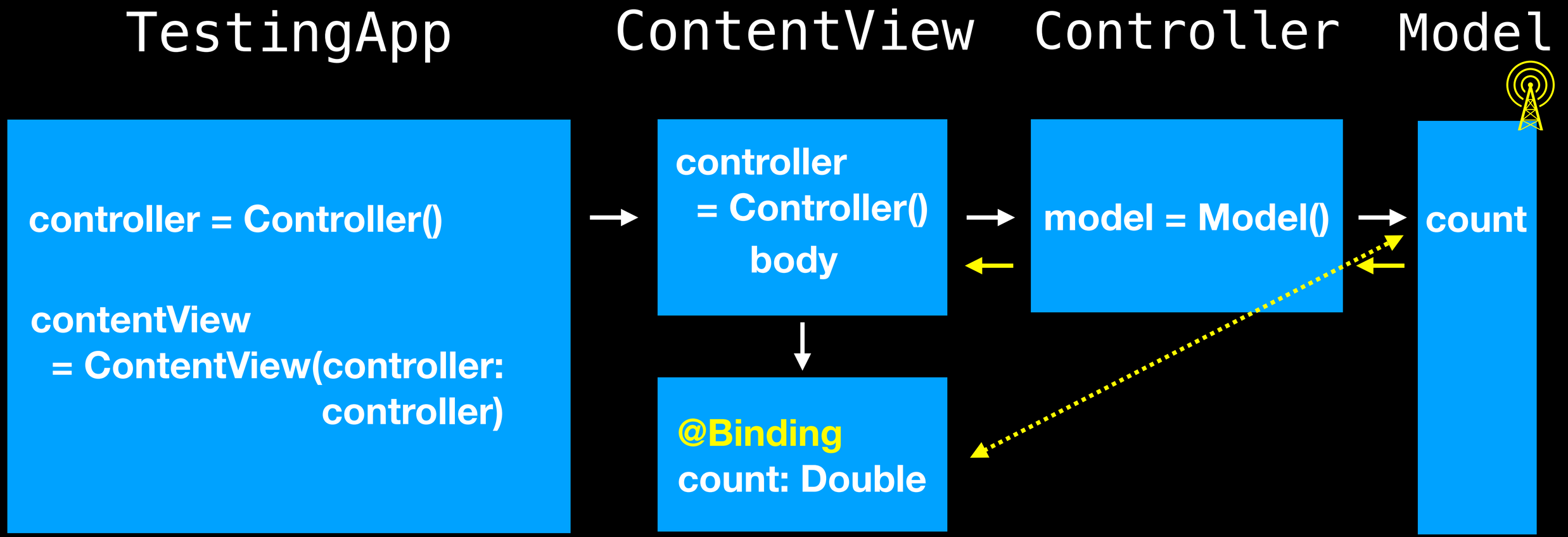
contentView
= ContentView(controller:
controller)

controller
= Controller()
body

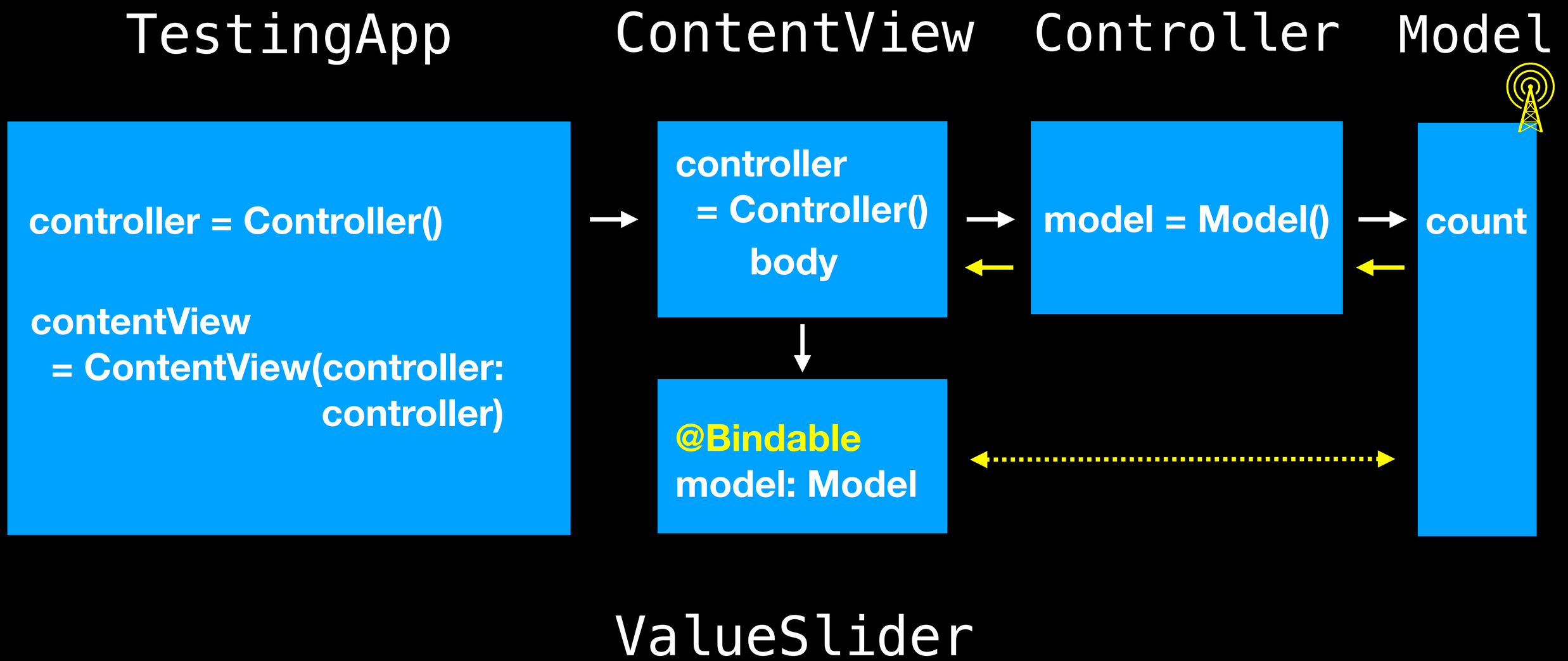
model = Model()

count


TestingApp
ContentView
Controller
Model
controller = Controller()
contentView
= ContentView(controller:
controller)
controller
= Controller()
body
@State
count: Double
model = Model()
count
ValueSlider



ValueSlider


TestingApp
ContentView
Controller
Model
controller = Controller()
contentView
= ContentView(controller:
controller)
controller
= Controller()
body
@Bindable
model: Model
model = Model()
count
ValueSlider

@State is for local only

```
import SwiftUI

struct ValueSlider {
  @State var count = 0.0
}

extension ValueSlider: View {
  var body: some View {
    Slider(value:                ,
           in: 0...100)
    .padding()
  }
}
```

Pass in model.count

```
import SwiftUI

struct ValueSlider {
  @Binding var count: Double
}

extension ValueSlider: View {
  var body: some View {
    Slider(value:                  ,
           in: 0...100)
    .padding()
  }
}
```

Pass in model.count

```
import SwiftUI

struct ValueSlider {
  @Binding var count: Double
}

extension ValueSlider: View {
  var body: some View {
    Slider(value:                  ,
           in: 0...100)
    .padding()
  }
}
```

Pass in model.count

```
import SwiftUI

struct ValueSlider {
  @Bindable var count: Double
}

extension ValueSlider: View {
  var body: some View {
    Slider(value:                 ,
           in: 0...100)
    .padding()
  }
}
```

Can't pass in model.count

```
import SwiftUI

struct ValueSlider {
  @Bindable var count: Double
}

extension ValueSlider: View {
  var body: some View {
    Slider(value:                ,
           in: 0...100)
    .padding()
  }
}
```

Bindable applies to an Observable object

```
import SwiftUI

struct ValueSlider {
  @Bindable var count: Double
}

extension ValueSlider: View {
  var body: some View {
    Slider(value:                 ,
           in: 0...100)
    .padding()
  }
}
```

Pass in an Observable object

```
import SwiftUI

struct ValueSlider {
  @Bindable var model: Model
}

extension ValueSlider: View {
  var body: some View {
    Slider(value:                 ,
           in: 0...100)
    .padding()
  }
}
```

```
import SwiftUI

struct ValueSlider {
  @Bindable var model: Model
}

extension ValueSlider: View {
  var body: some View {
    Slider(value: $model.count,
           in: 0...100)
    .padding()
  }
}
```

Even more magical

Take a breath

We moved the state

From the view

to the controller using Combine

to the controller using Observable

to the model using Observable

to the model using Observable

to a persisted model using SwiftData

to a persisted model using SwiftData
which uses Observable

to a persisted model using SwiftData
which uses Observable

The Curious Case of the Mutating Model

dimsumthinking.com

Data Flow in SwiftUI from @State to @Observable

Pragma October, 2023

Daniel H Steinberg

dimsumthinking.com