

#WWDC19

Data Flow Through SwiftUI

Luca Bernardi, SwiftUI Engineer

Raj Ramamurthy, SwiftUI Engineer

Principles of Data Flow

Anatomy of an Update

Understanding Your Data

Principles of Data Flow

Anatomy of an Update

Understanding Your Data

Principles of Data Flow

Anatomy of an Update

Understanding Your Data

What is data?

Tools for Data Flow

@Environment

Property



@Binding

BindableObject

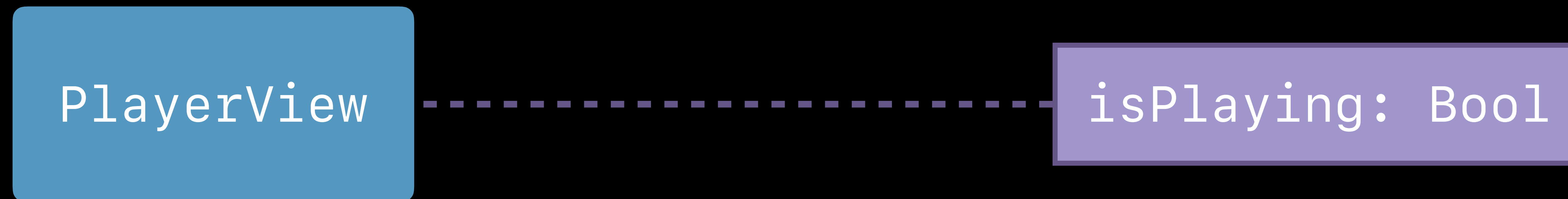
@State

Data Access as a Dependency

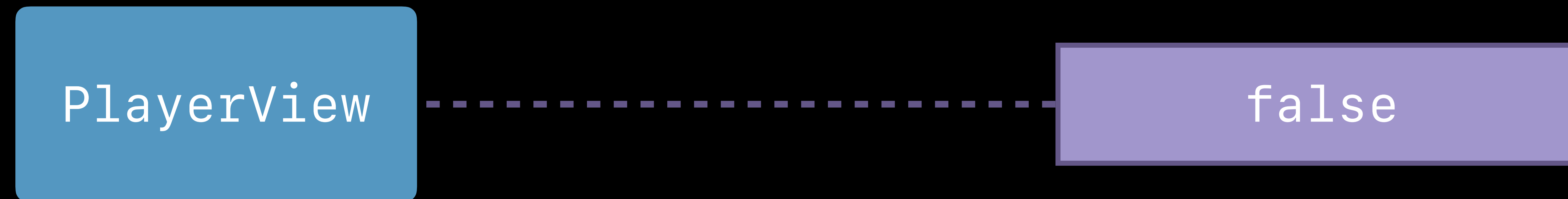
PlayerView

isPlaying: Bool

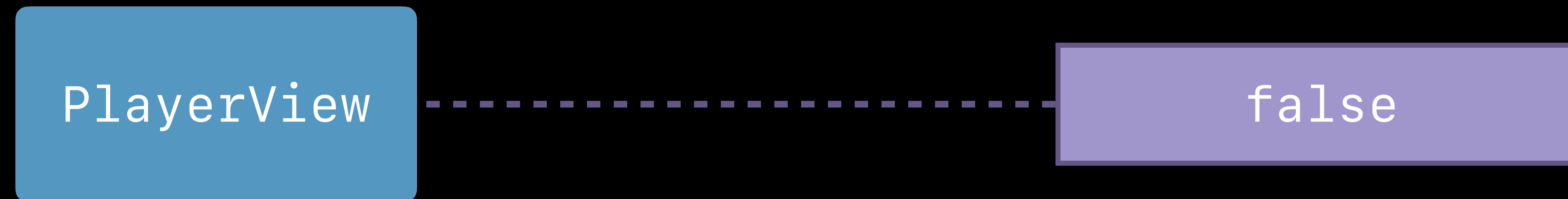
Data Access as a Dependency



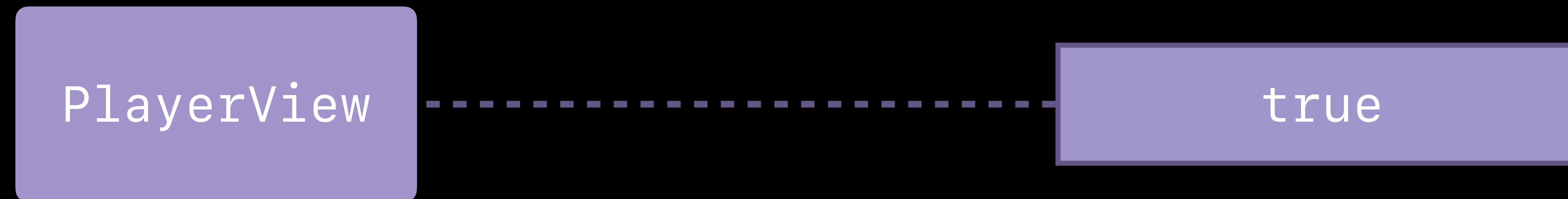
Data Access as a Dependency



Data Access as a Dependency



Data Access as a Dependency



Source of Truth

Source of Truth

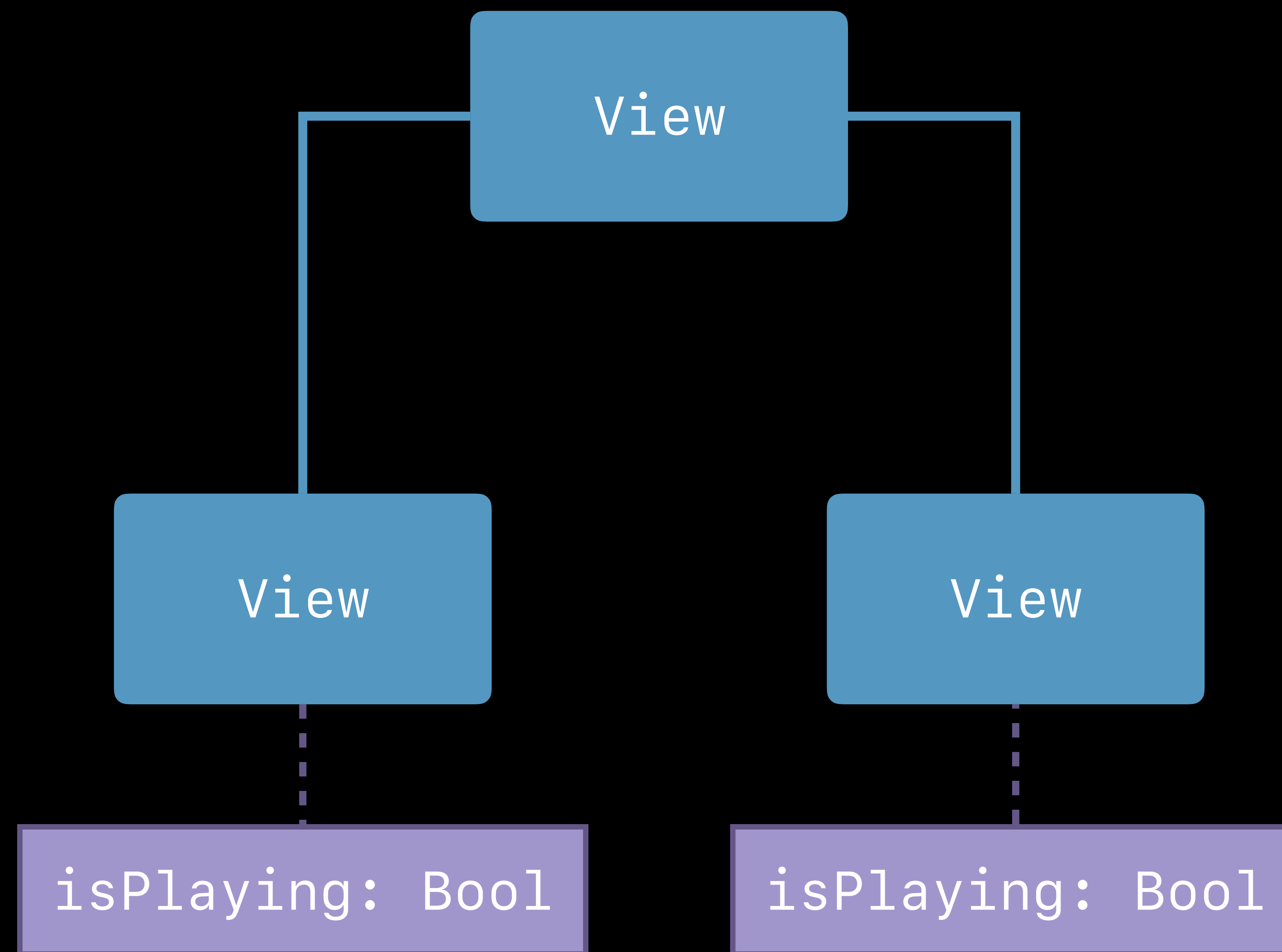
post: PhotoPost

isPlaying: Bool

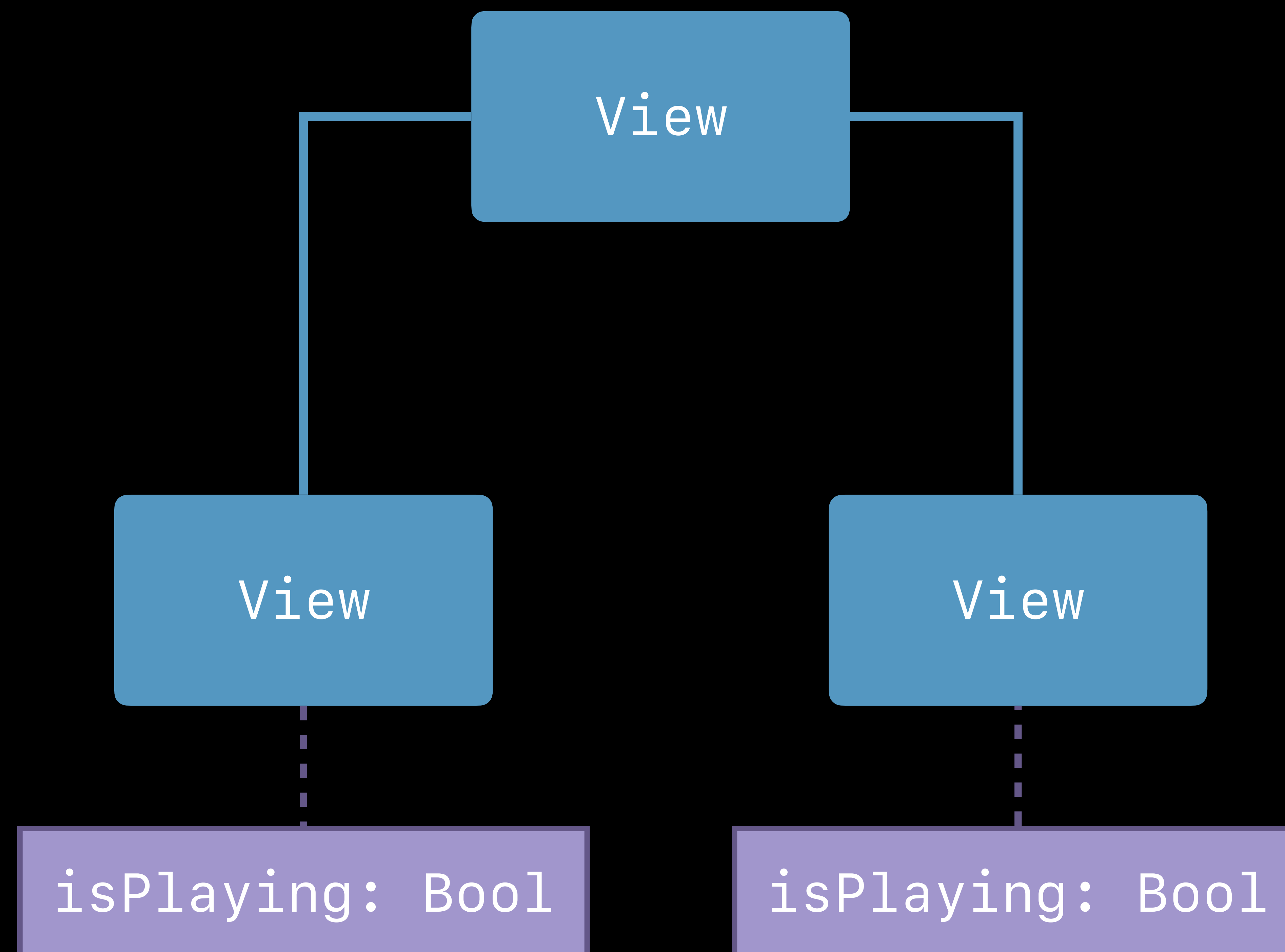
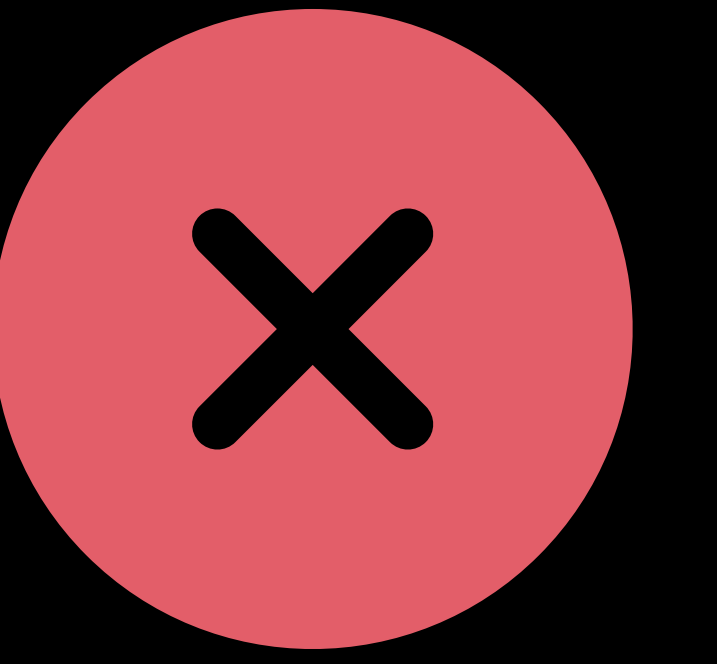
accentColor: Color

store: MessageStore

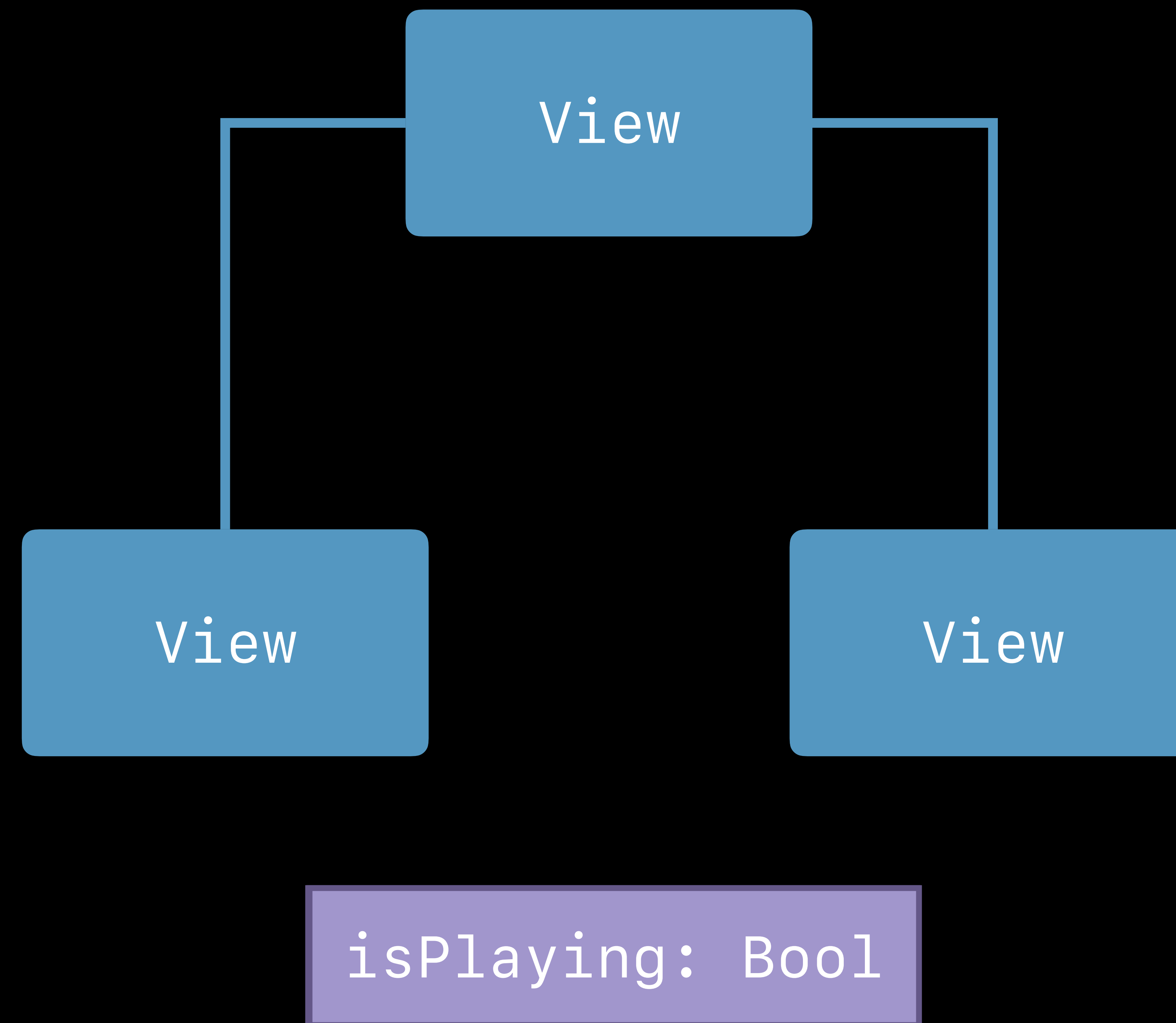
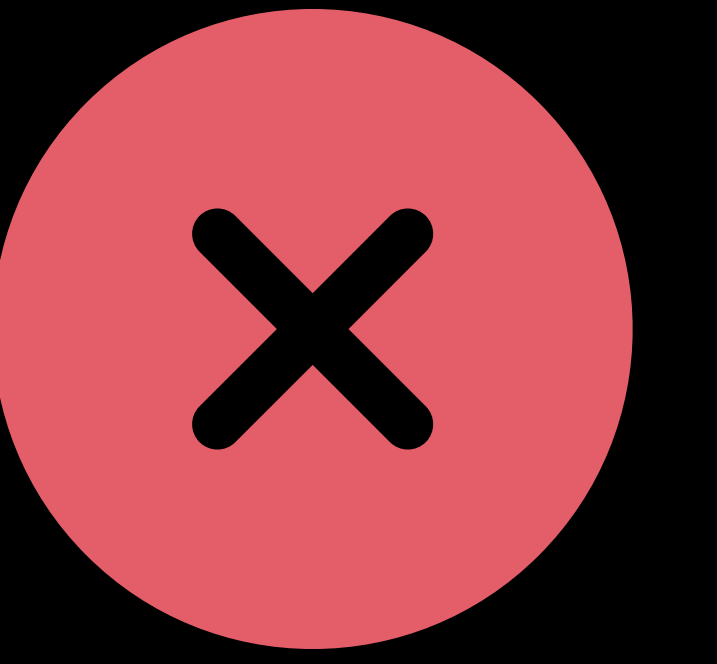
Duplicated Source of Truth



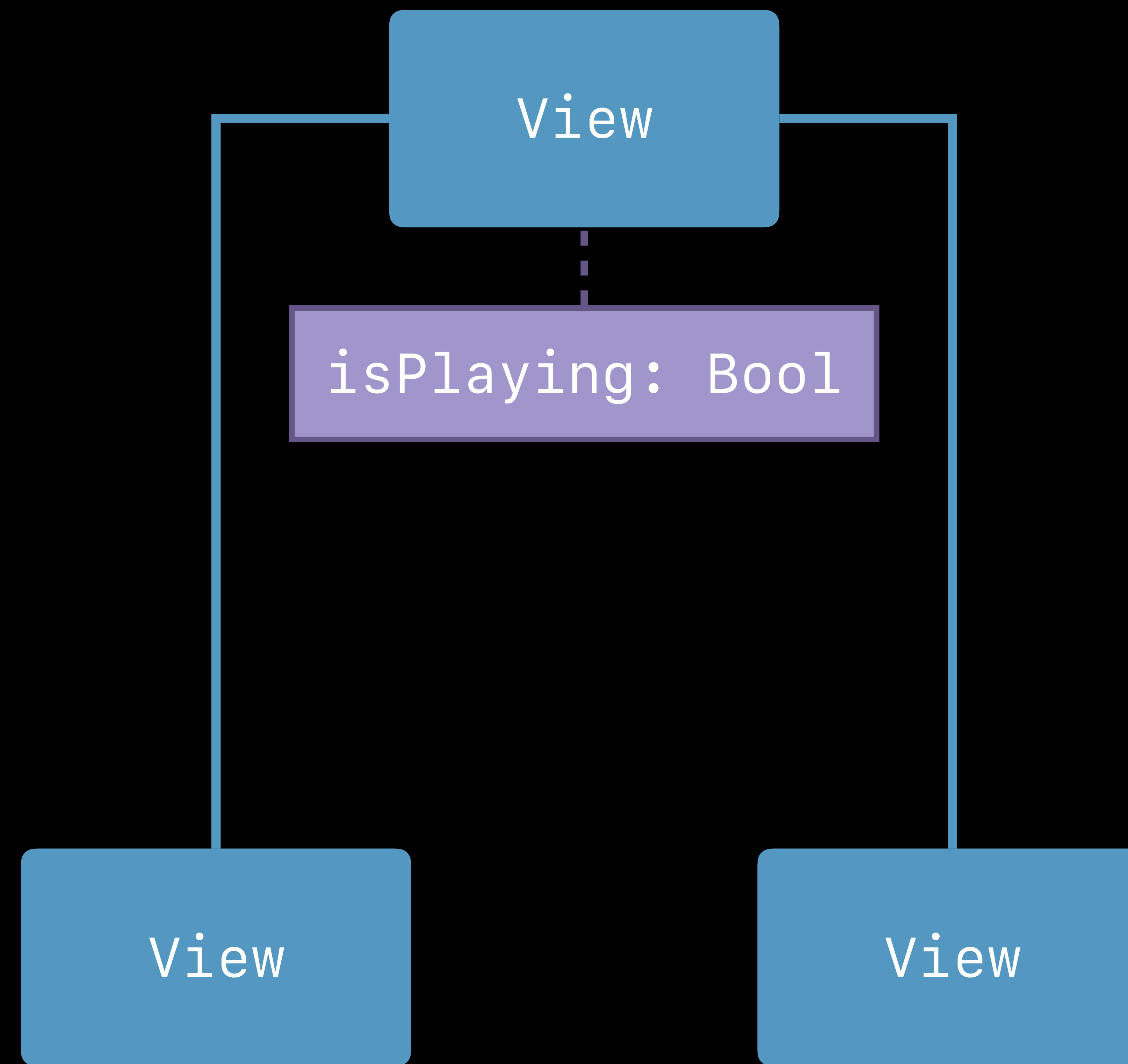
Duplicated Source of Truth



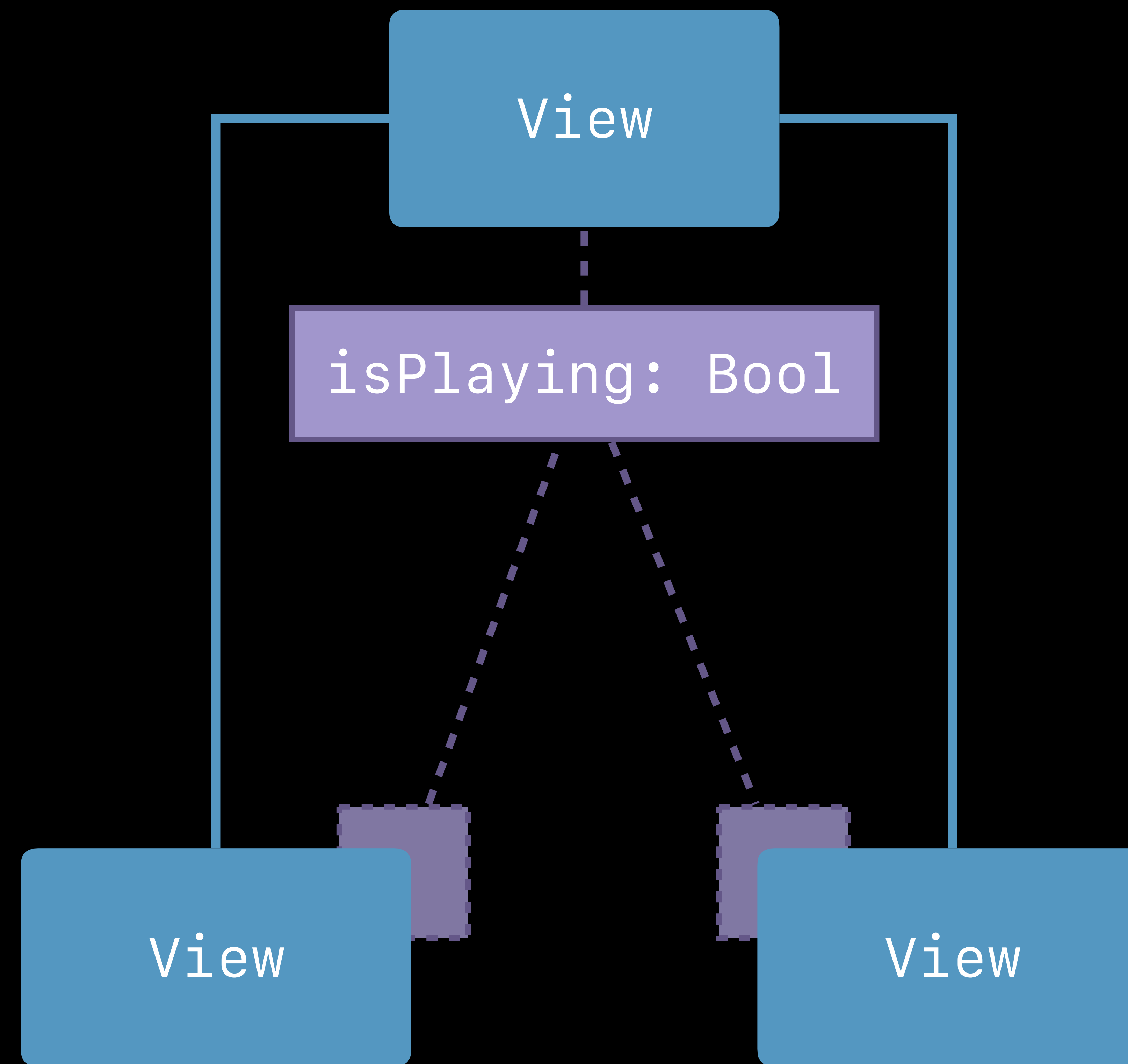
Duplicated Source of Truth



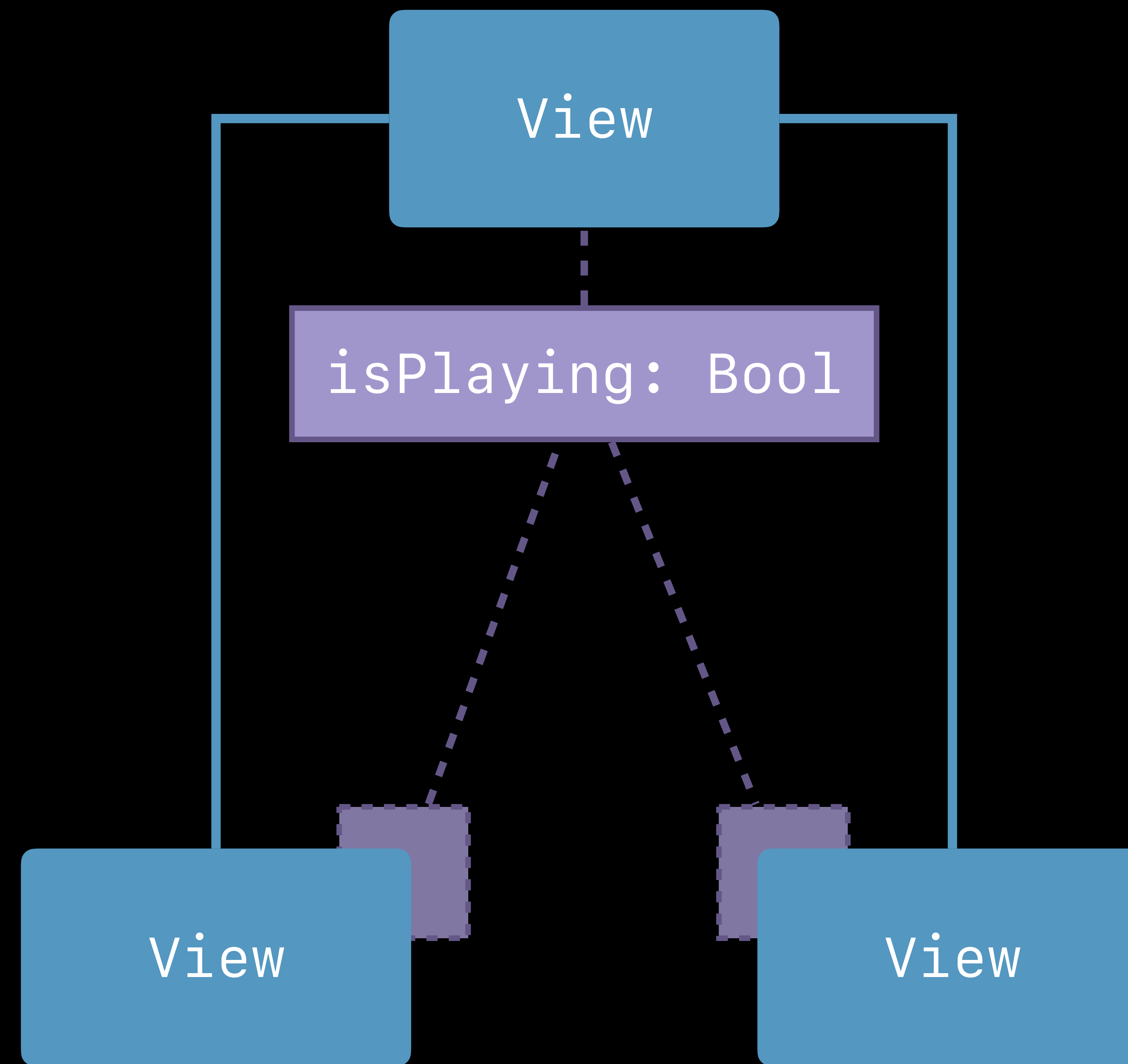
Single Source of Truth



Single Source of Truth



Single Source of Truth





10:09

Amatriciana
Caffè Macs Radio



1:32:00

```
struct PlayerView : View {
    let episode: Episode

    var body: some View {
        VStack {
            Text(episode.title)
            Text(episode.showTitle).font(.caption).foregroundColor(.gray)
        }
    }
}
```



```
struct PlayerView : View {
```

```
    let episode: Episode
```

```
    var body: some View {
```

```
        VStack {
```

```
            Text(episode.title)
```

```
            Text(episode.showTitle).font(.caption).foregroundColor(.gray)
```

```
        }
```

```
    }
```

```
}
```

10:09

Amatriciana
Caffè Macs Radio


```
struct PlayerView : View {
    let episode: Episode

    var body: some View {
        VStack {
            Text(episode.title)
            Text(episode.showTitle).font(.caption).foregroundColor(.gray)
        }
    }
}
```

10:09

Amatriciana
Caffè Macs Radio

```
struct PlayerView : View {
    let episode: Episode

    var body: some View {
        VStack {
            Text(episode.title)
            Text(episode.showTitle).font(.caption).foregroundColor(.gray)
        }
    }
}
```

10:09

Amatriciana
Caffè Macs Radio


```
struct PlayerView : View {
    let episode: Episode
    var isPlaying: Bool

    var body: some View {
        VStack {
            Text(episode.title)
            Text(episode.showTitle).font(.caption).foregroundColor(.gray)

            Image(systemName: isPlaying ? "pause.circle" : "play.circle")
        }
    }
}
```



```
struct PlayerView : View {
    let episode: Episode
    var isPlaying: Bool

    var body: some View {
        VStack {
            Text(episode.title)
            Text(episode.showTitle).font(.caption).foregroundColor(.gray)
            Image(systemName: isPlaying ? "pause.circle" : "play.circle")
        }
    }
}
```



```
struct PlayerView : View {
    let episode: Episode
    var isPlaying: Bool

    var body: some View {
        VStack {
            Text(episode.title)
            Text(episode.showTitle).font(.caption).foregroundColor(.gray)

            Button(action: {
                self.isPlaying.toggle()
            }) {
                Image(systemName: isPlaying ? "pause.circle" : "play.circle")
            }
        }
    }
}
```



```
struct PlayerView : View {
    let episode: Episode
    var isPlaying: Bool

    var body: some View {
        VStack {
            Text(episode.title)
            Text(episode.showTitle).font(.caption).foregroundColor(.gray)

            Button(action: {
                self.isPlaying.toggle()
            }) {
                Image(systemName: isPlaying ? "pause.circle" : "play.circle")
            }
        }
    }
}
```




```

struct PlayerView : View {
    let episode: Episode
    var isPlaying: Bool

    var body: some View {
        VStack {
            Text(episode.title)
            Text(episode.showTitle).font(.caption).foregroundColor(.gray)

            Button(action: {
                self.isPlaying.toggle()
            }) {
                Image(systemName: isPlaying ? "pause.circle" : "play.circle")
            }
        }
    }
}

```



Cannot use mutating member on immutable value: 'self' is immutable

```
struct PlayerView : View {
  let episode: Episode
  @State private var isPlaying: Bool = false

  var body: some View {
    VStack {
      Text(episode.title)
      Text(episode.showTitle).font(.caption).foregroundColor(.gray)

      Button(action: {
        self.isPlaying.toggle()
      }) {
        Image(systemName: isPlaying ? "pause.circle" : "play.circle")
      }
    }
  }
}
```



```
struct PlayerView : View {
  let episode: Episode
  @State private var isPlaying: Bool = false

  var body: some View {
    VStack {
      Text(episode.title)
      Text(episode.showTitle).font(.caption).foregroundColor(.gray)

      Button(action: {
        self.isPlaying.toggle()
      }) {
        Image(systemName: isPlaying ? "pause.circle" : "play.circle")
      }
    }
  }
}
```

false




```
struct PlayerView : View {
  let episode: Episode
  @State private var isPlaying: Bool = false

  var body: some View {
    VStack {
      Text(episode.title)
      Text(episode.showTitle).font(.caption).foregroundColor(.gray)

      Button(action: {
        self.isPlaying.toggle()
      }) {
        Image(systemName: isPlaying ? "pause.circle" : "play.circle")
      }
    }
  }
}
```

false

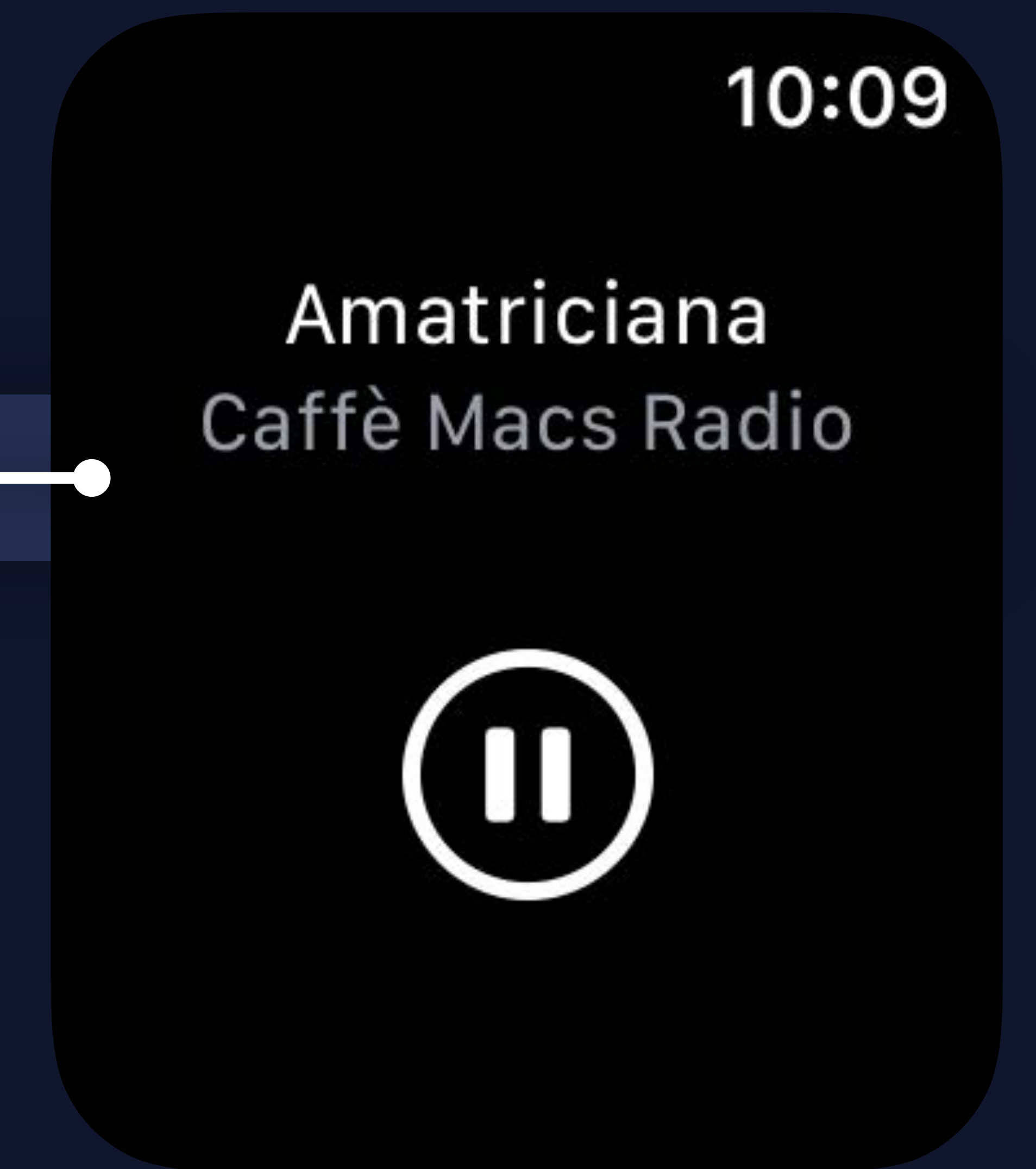



```
struct PlayerView : View {
  let episode: Episode
  @State private var isPlaying: Bool = false

  var body: some View {
    VStack {
      Text(episode.title)
      Text(episode.showTitle).font(.caption).foregroundColor(.gray)

      Button(action: {
        self.isPlaying.toggle()
      }) {
        Image(systemName: isPlaying ? "pause.circle" : "play.circle")
      }
    }
  }
}
```

true



Property Wrapper

Wraps property access with additional behavior

What's New in Swift

WWDC 2019

Modern Swift API Design

Thursday, 2:00

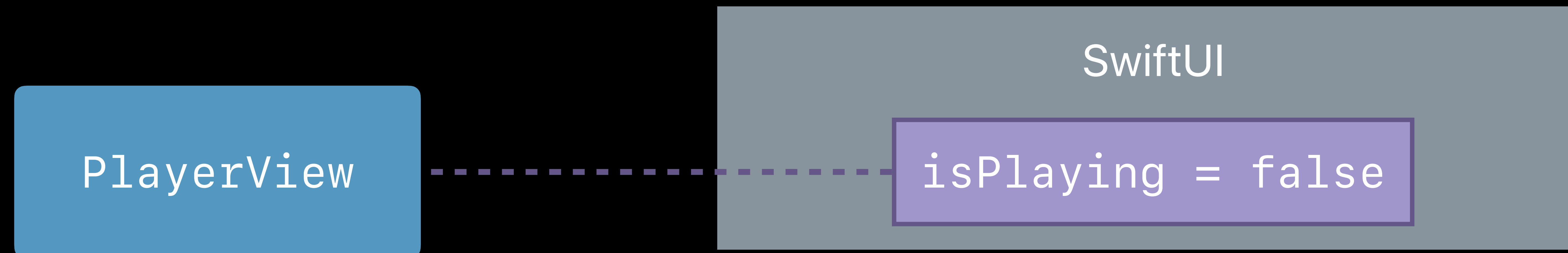
@State Property Wrapper

PlayerView

SwiftUI

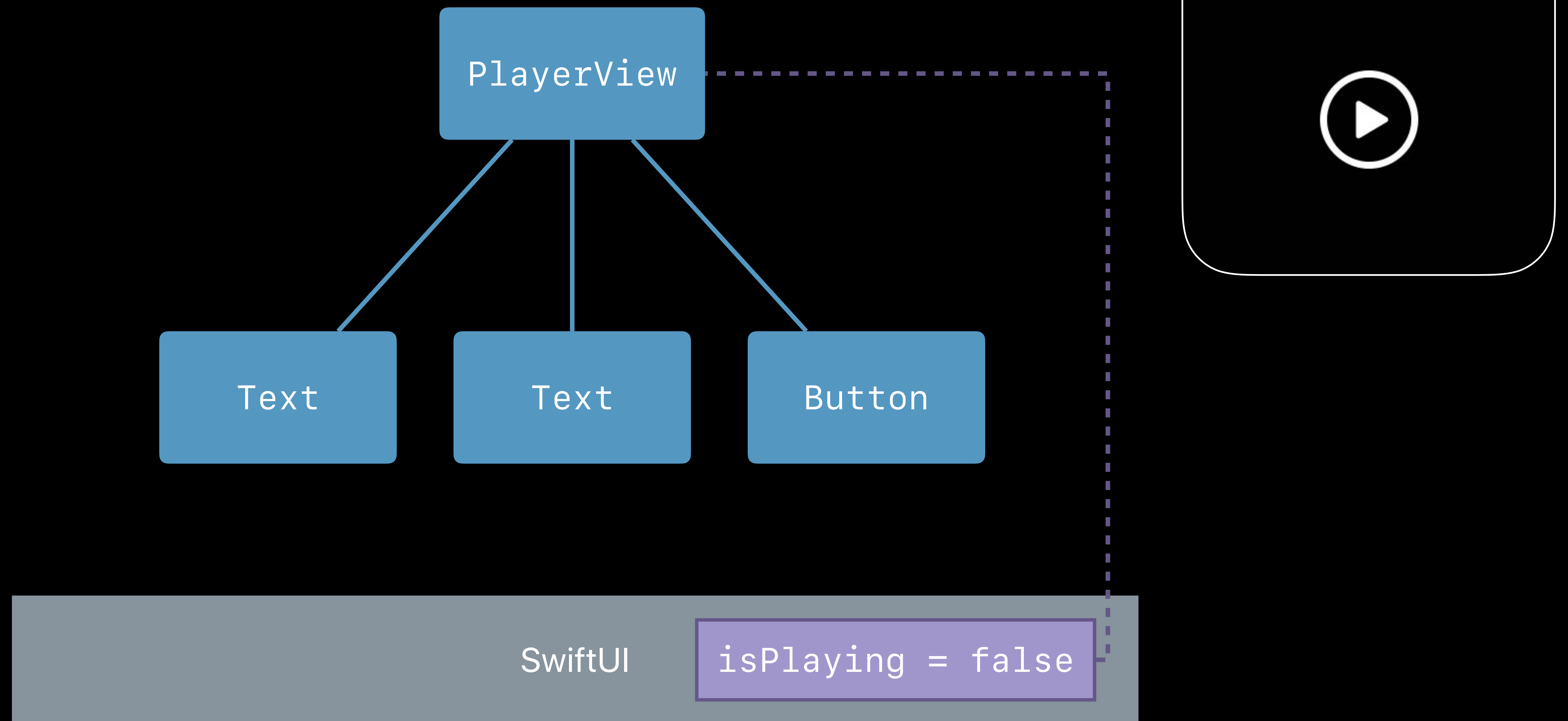
```
@State private var isPlaying: Bool = false
```

@State Property Wrapper

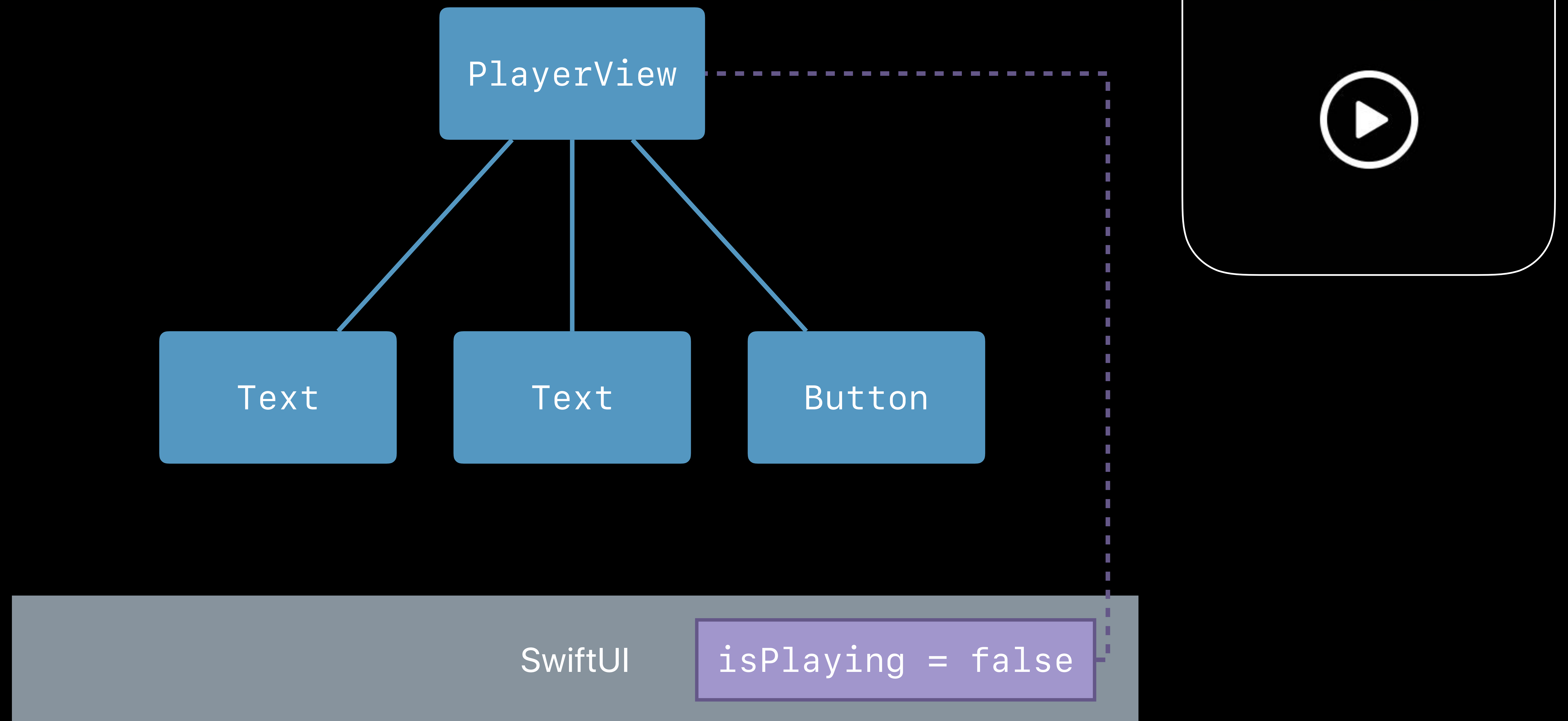


```
@State private var isPlaying: Bool = false
```

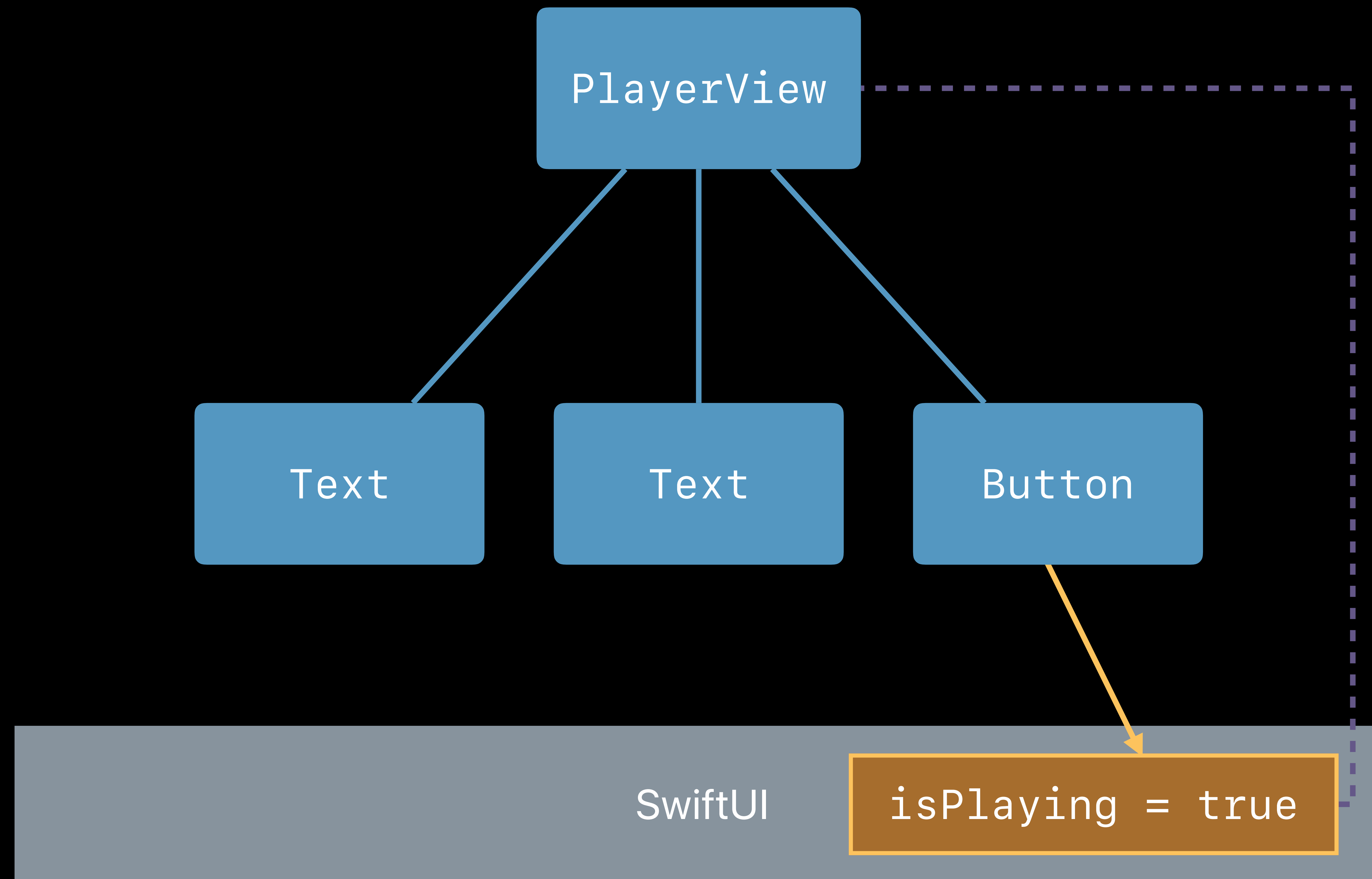
Anatomy of a View Update



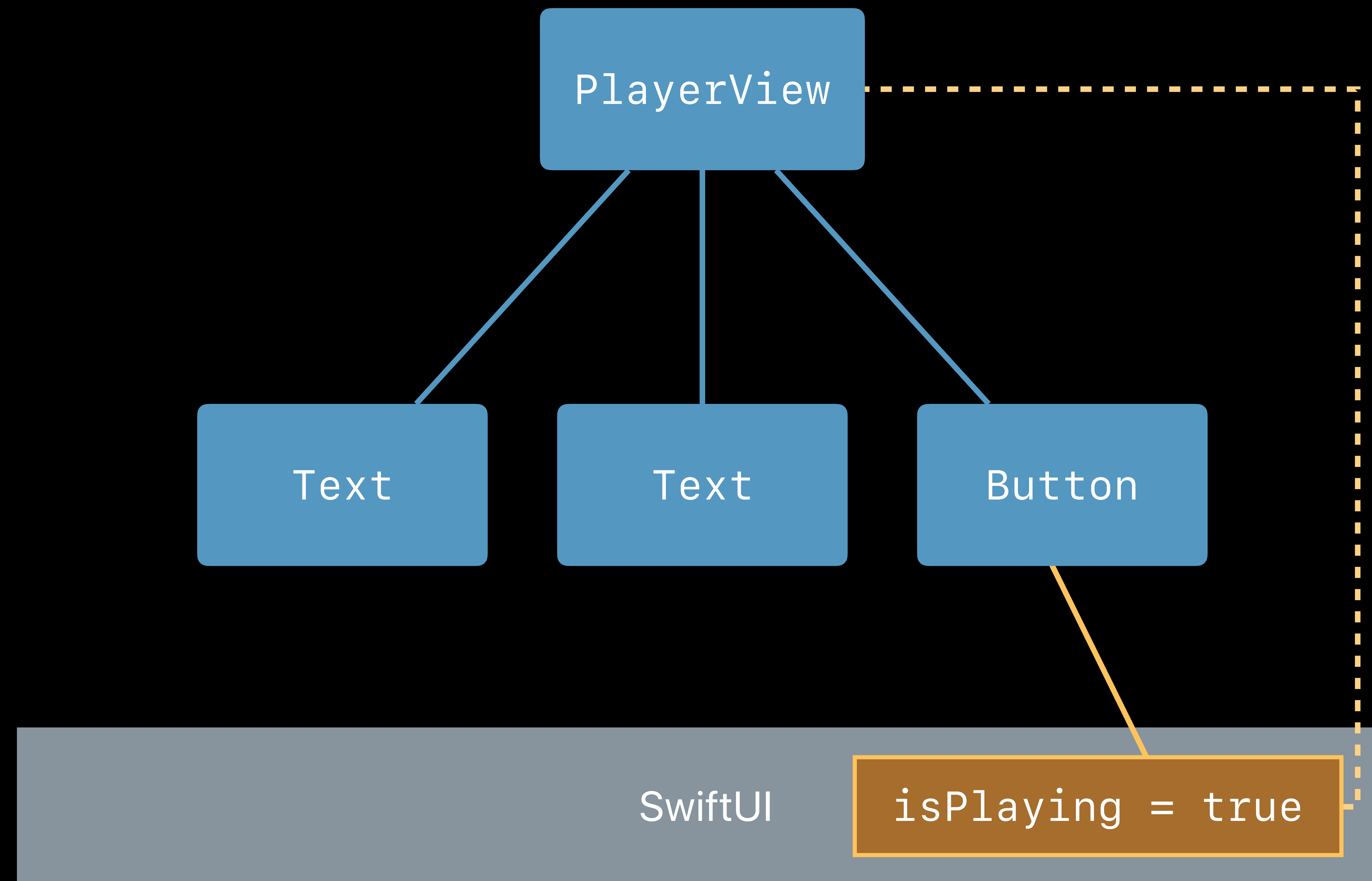
Anatomy of a View Update



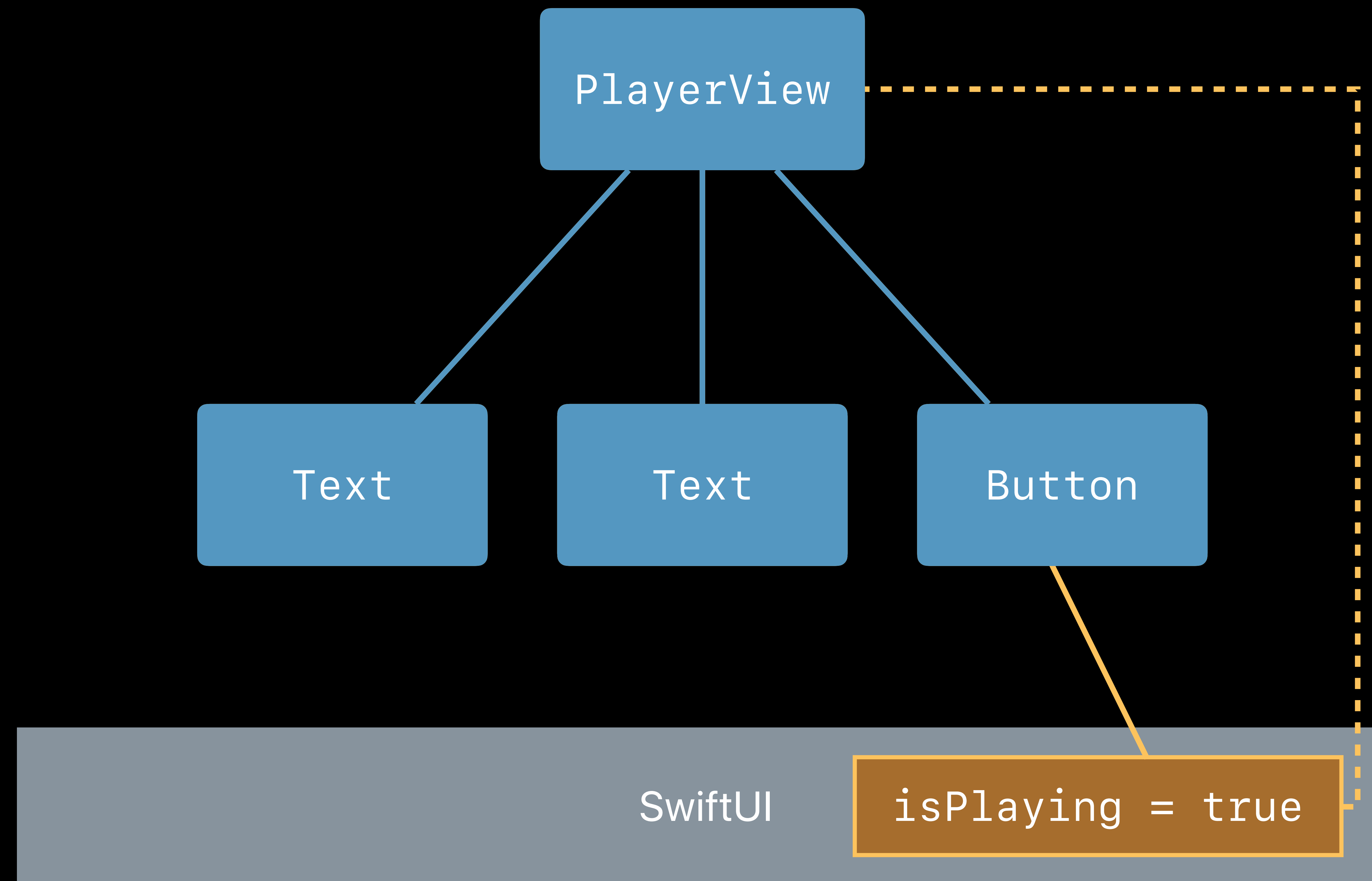
Anatomy of a View Update



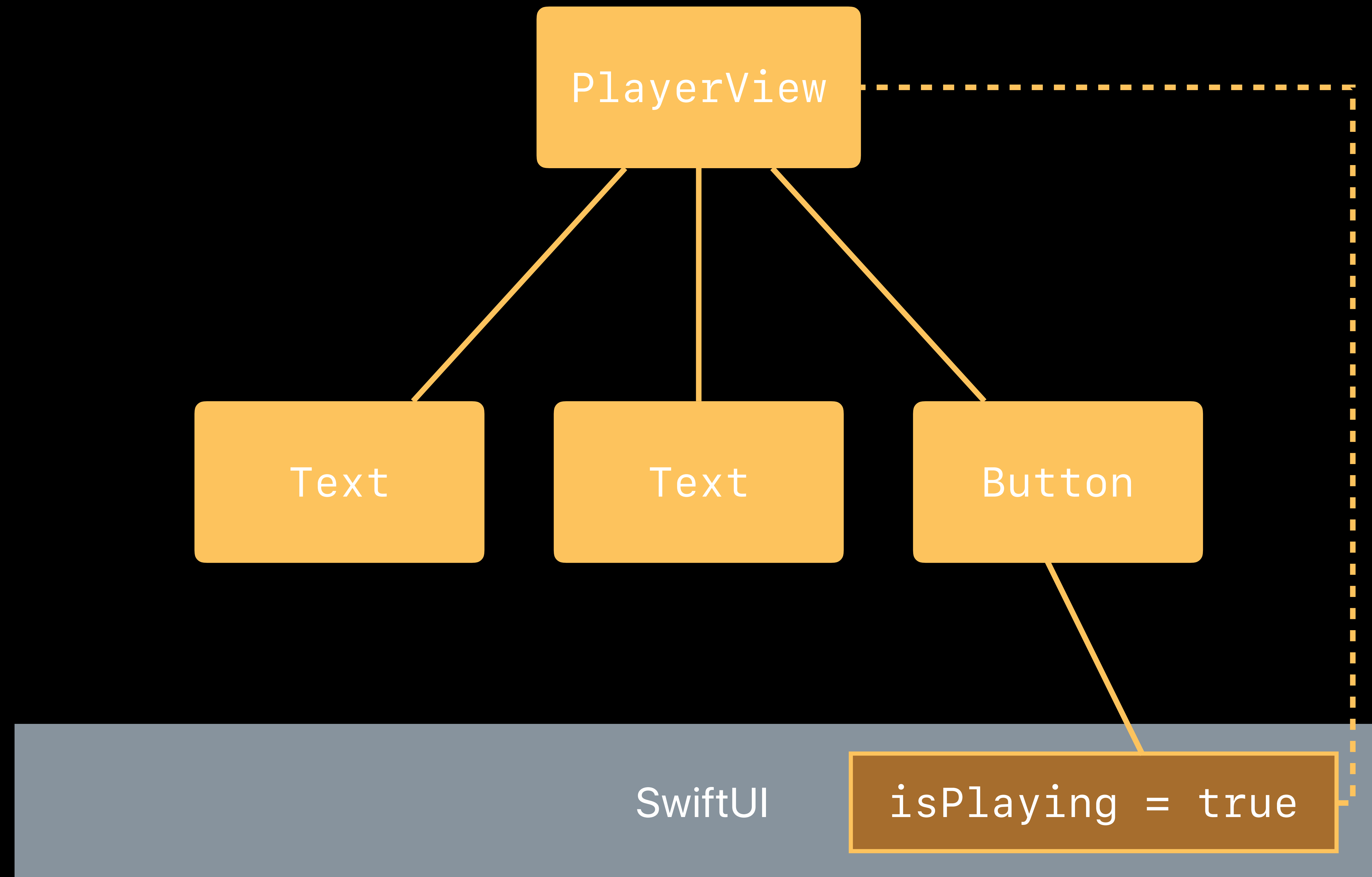
Anatomy of a View Update



Anatomy of a View Update

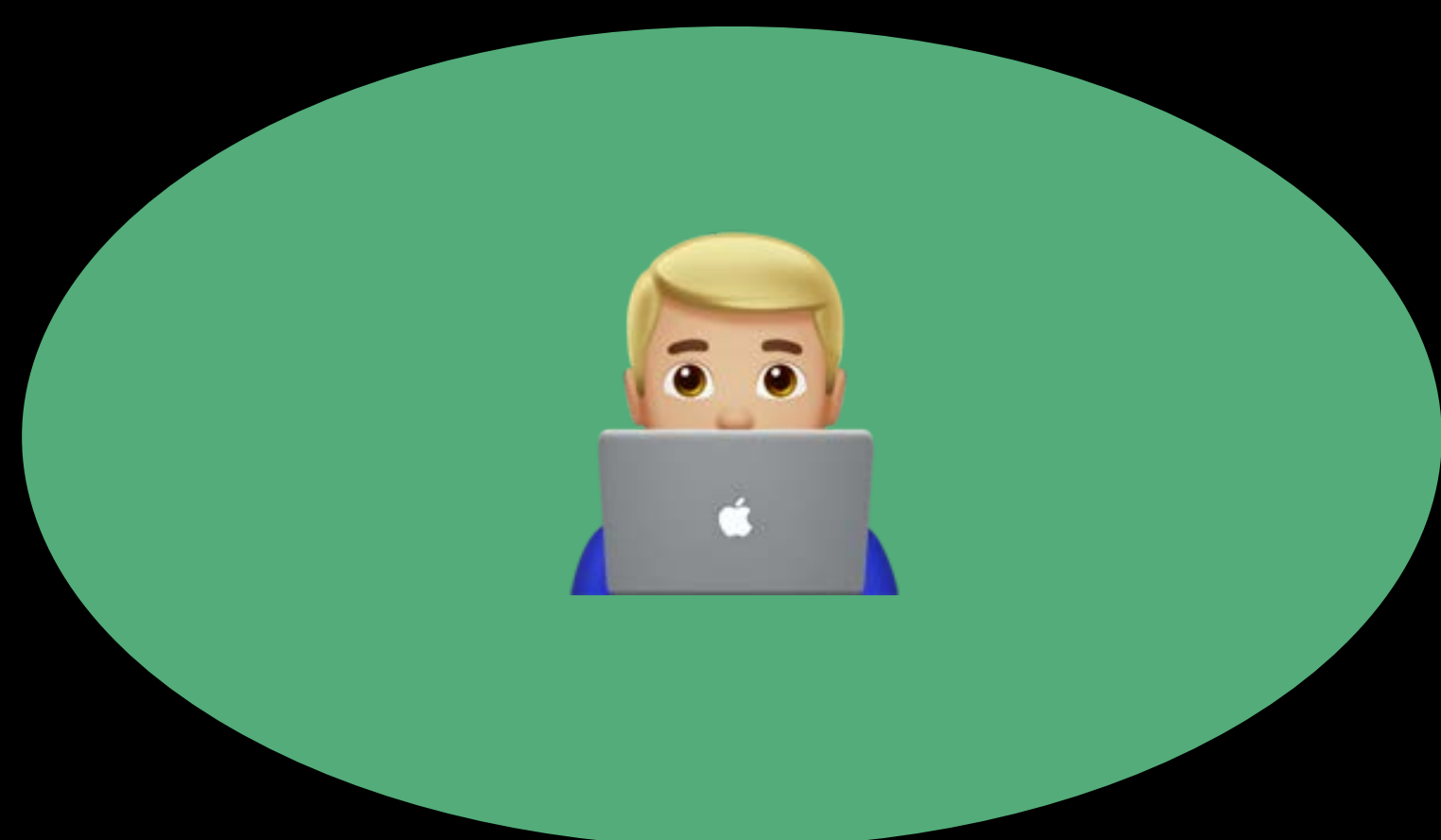


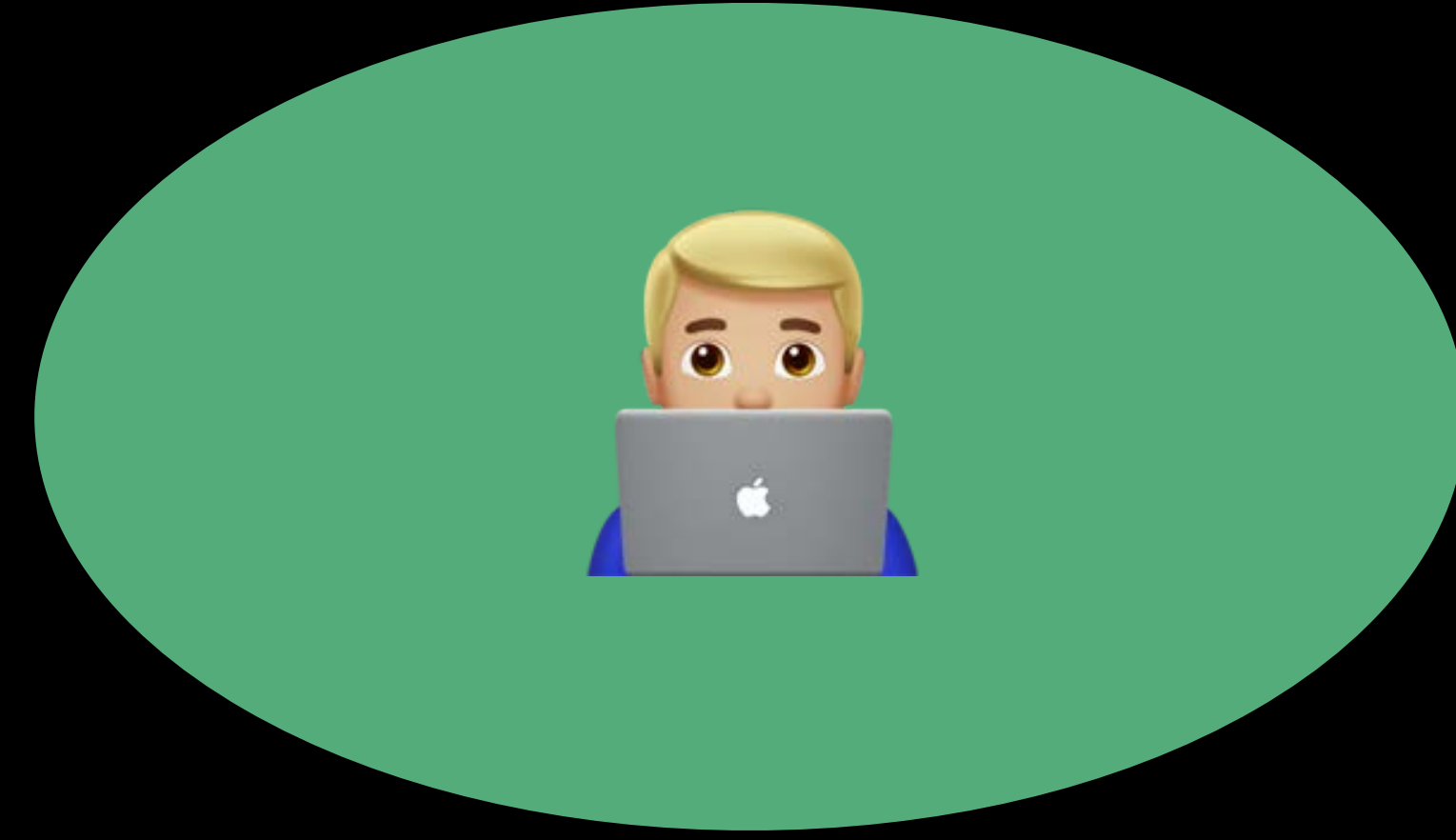
Anatomy of a View Update



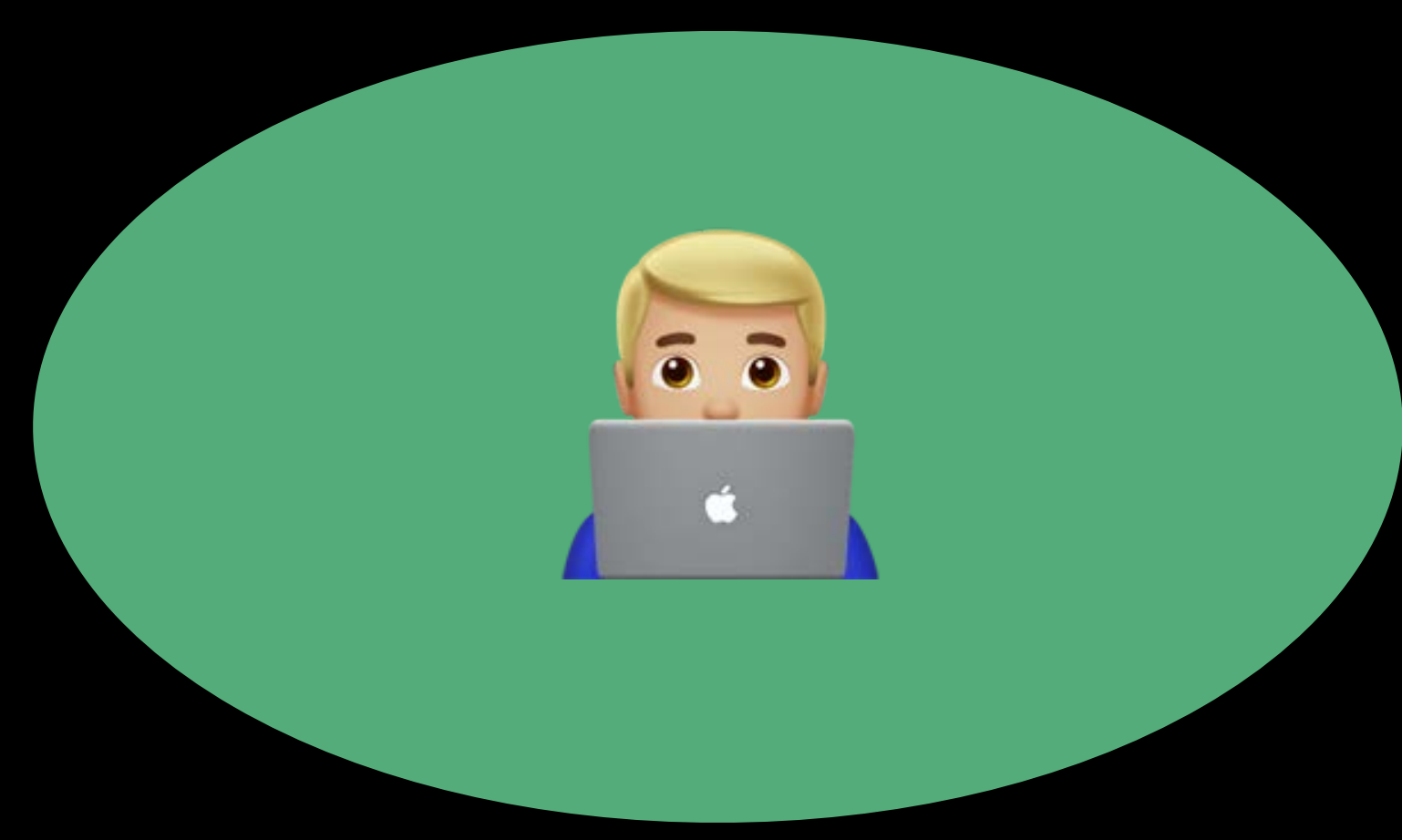
Every @State is a source of truth.

Views are a function of state,
not of a sequence of events.

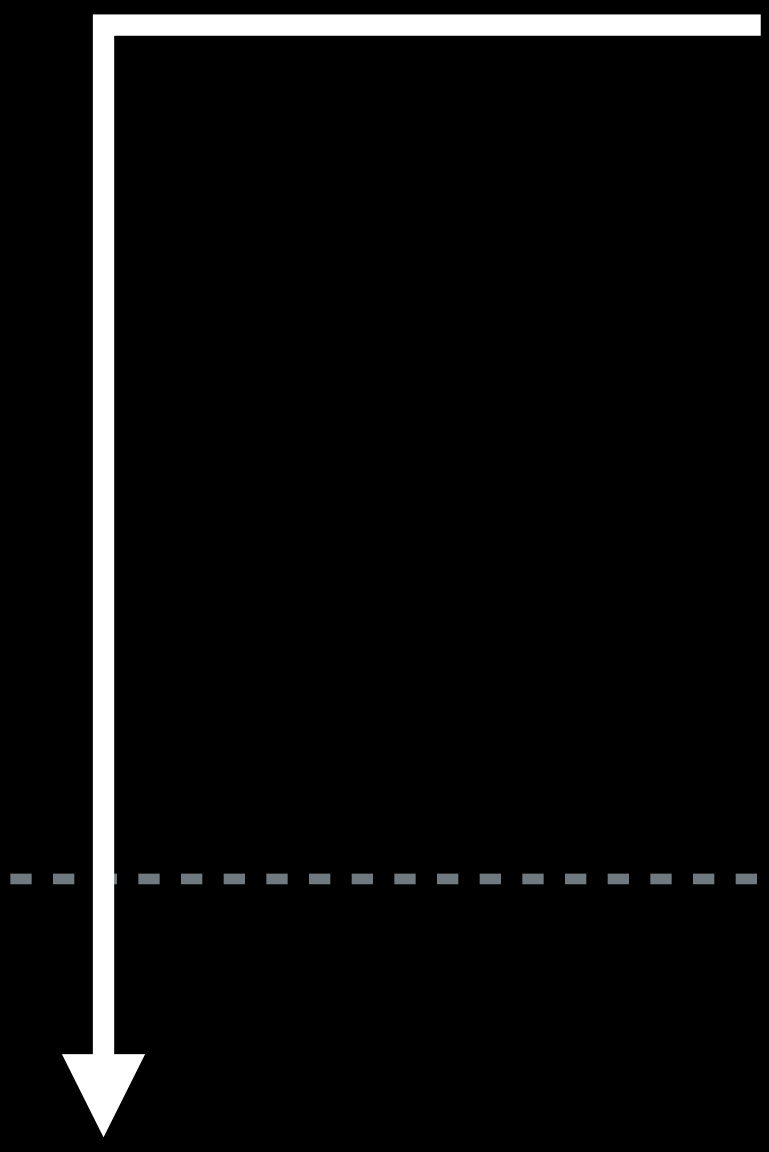




SwiftUI

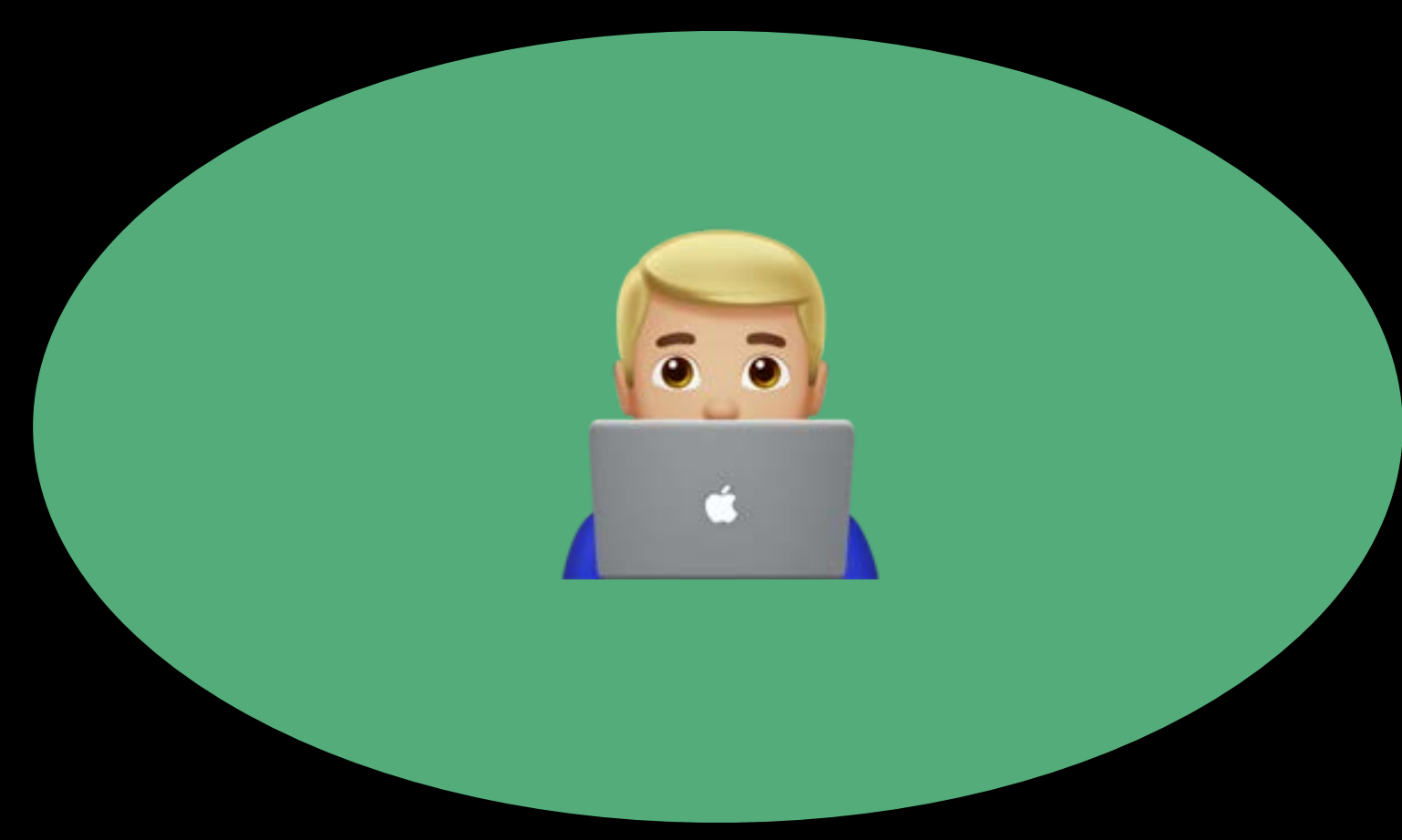


User Interaction

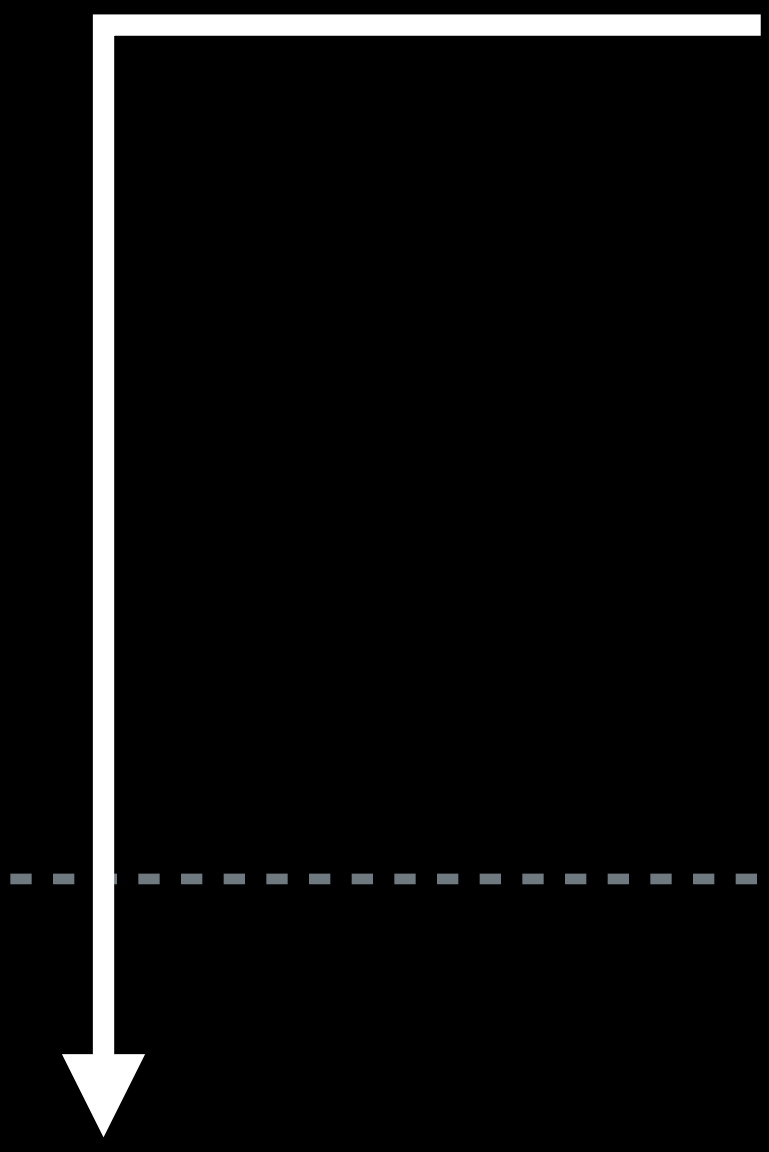


Action

SwiftUI



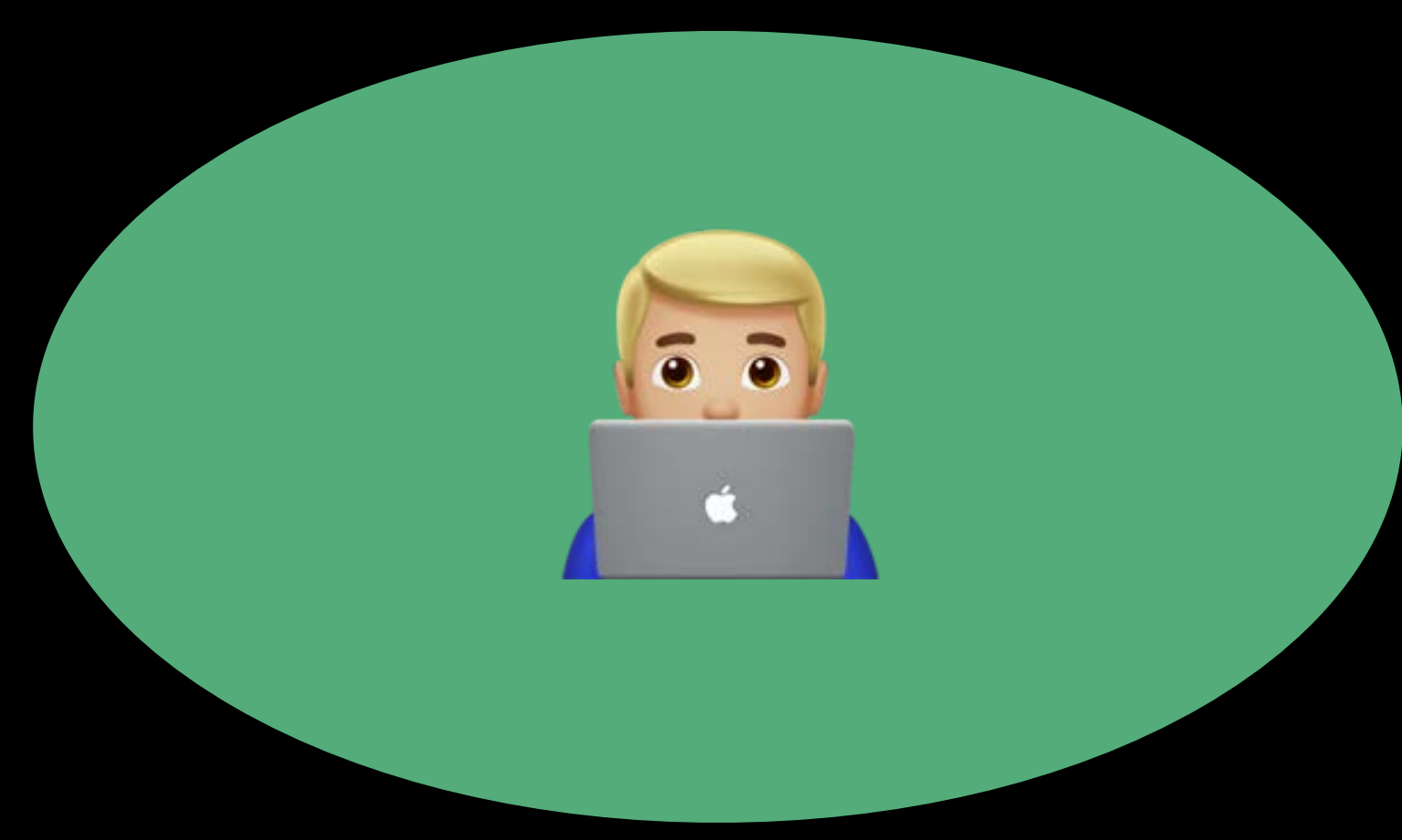
User Interaction



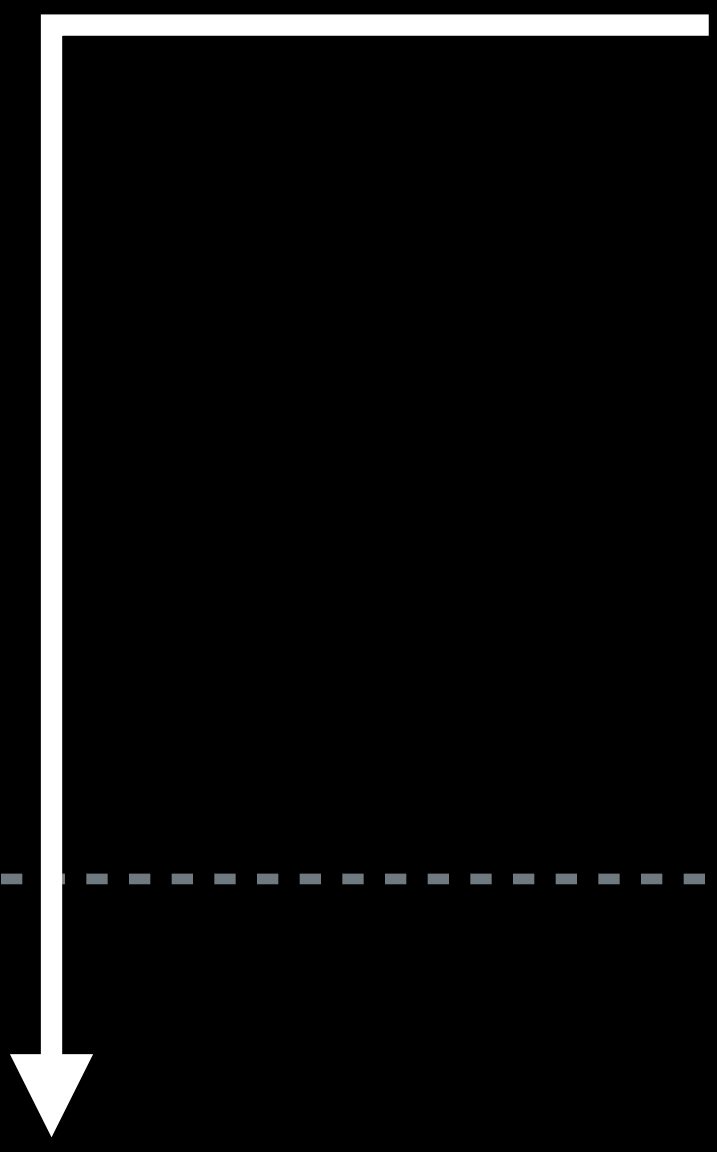
Action

SwiftUI



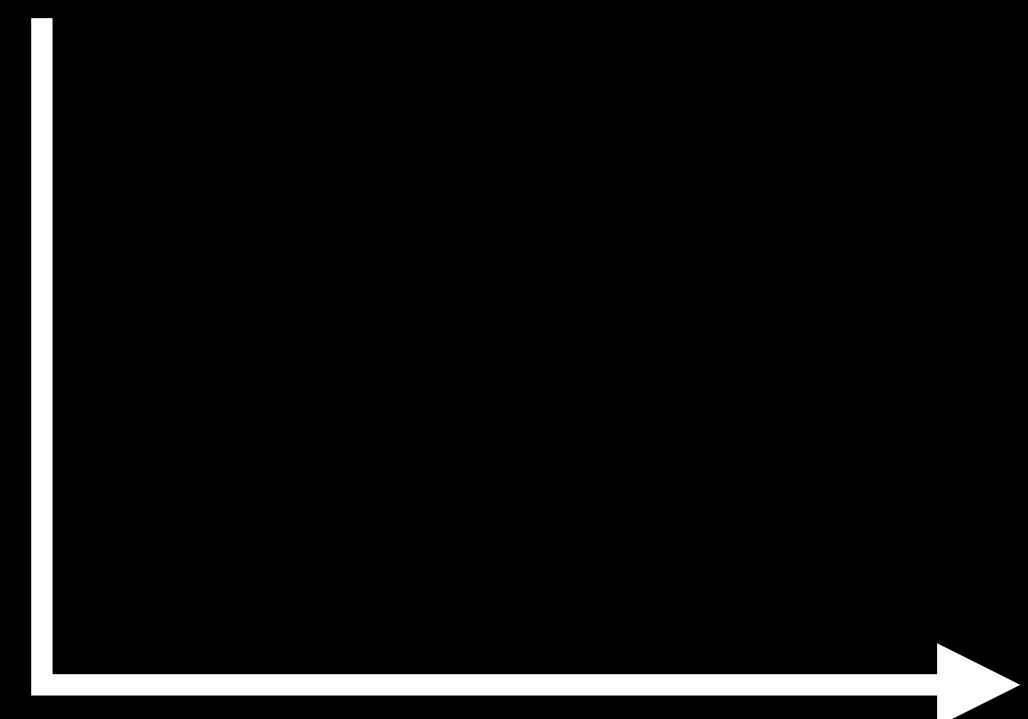


User Interaction



Action

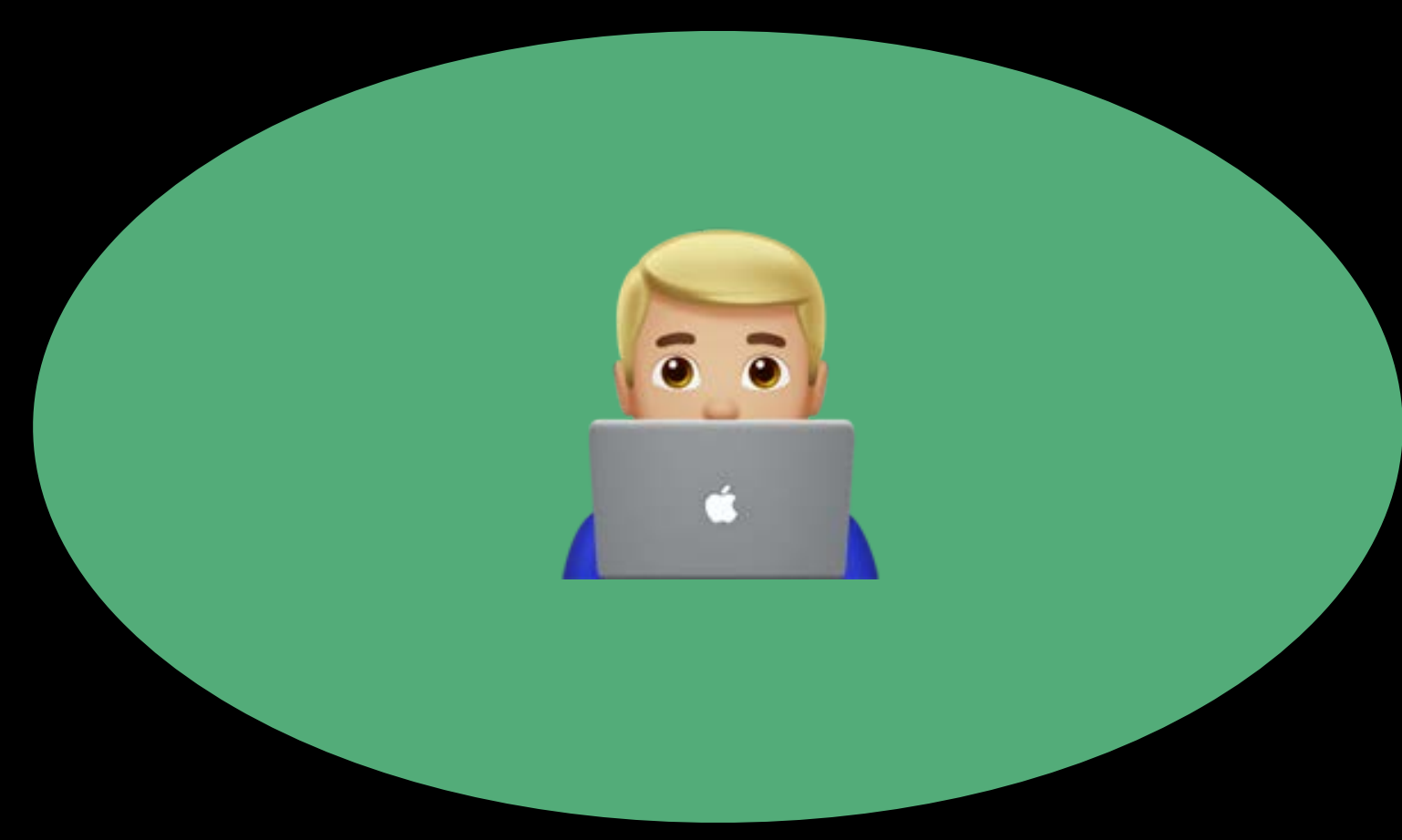
Mutation



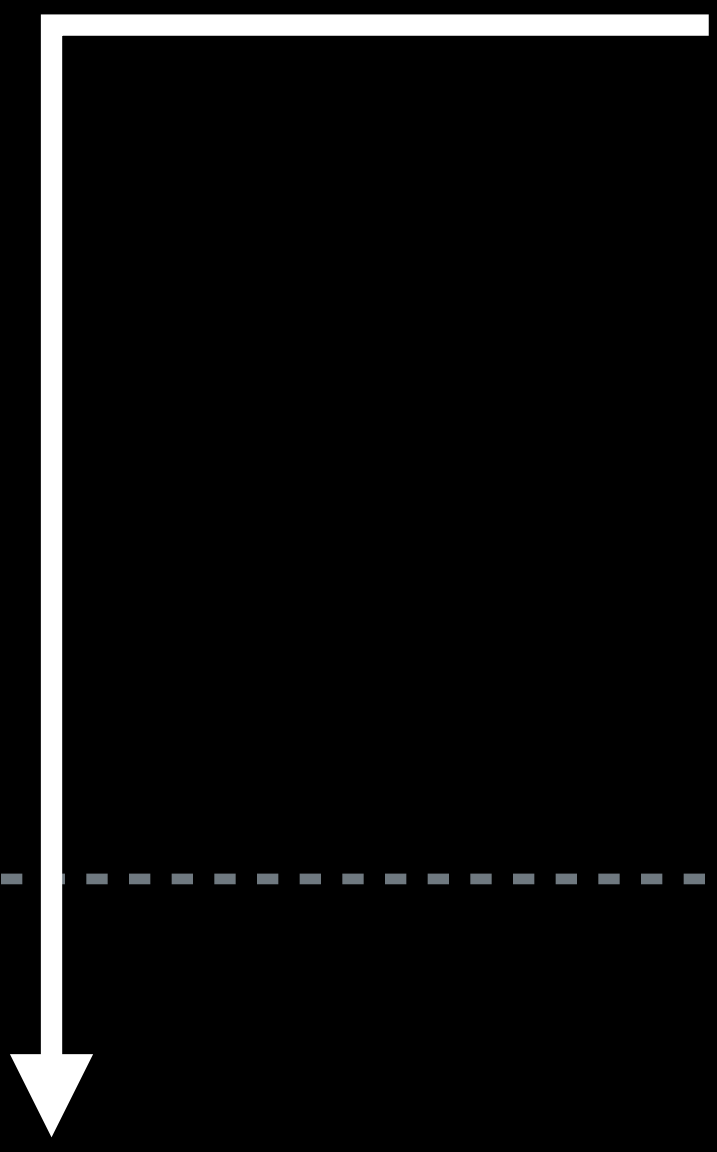
State

SwiftUI





User Interaction



Action

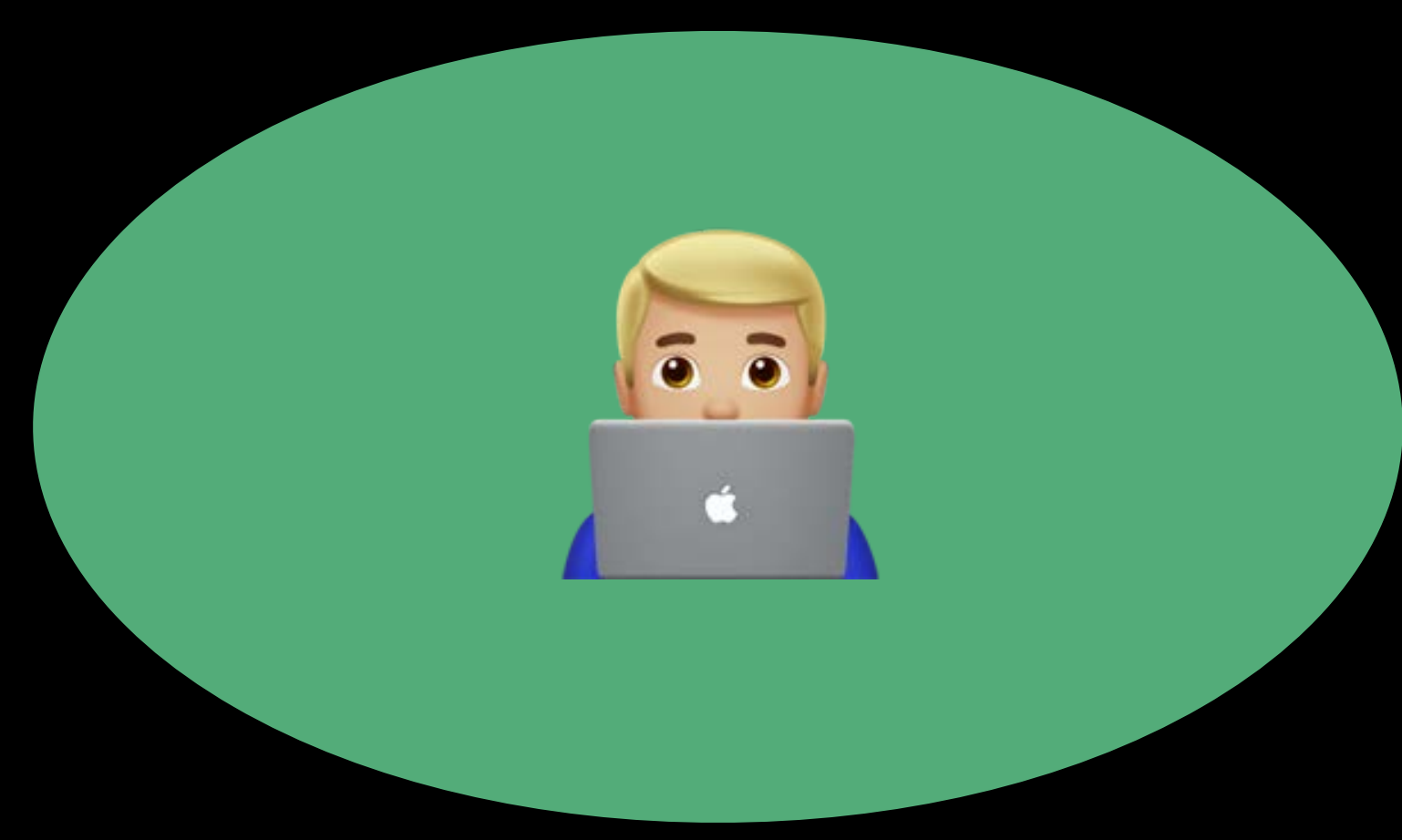
Mutation



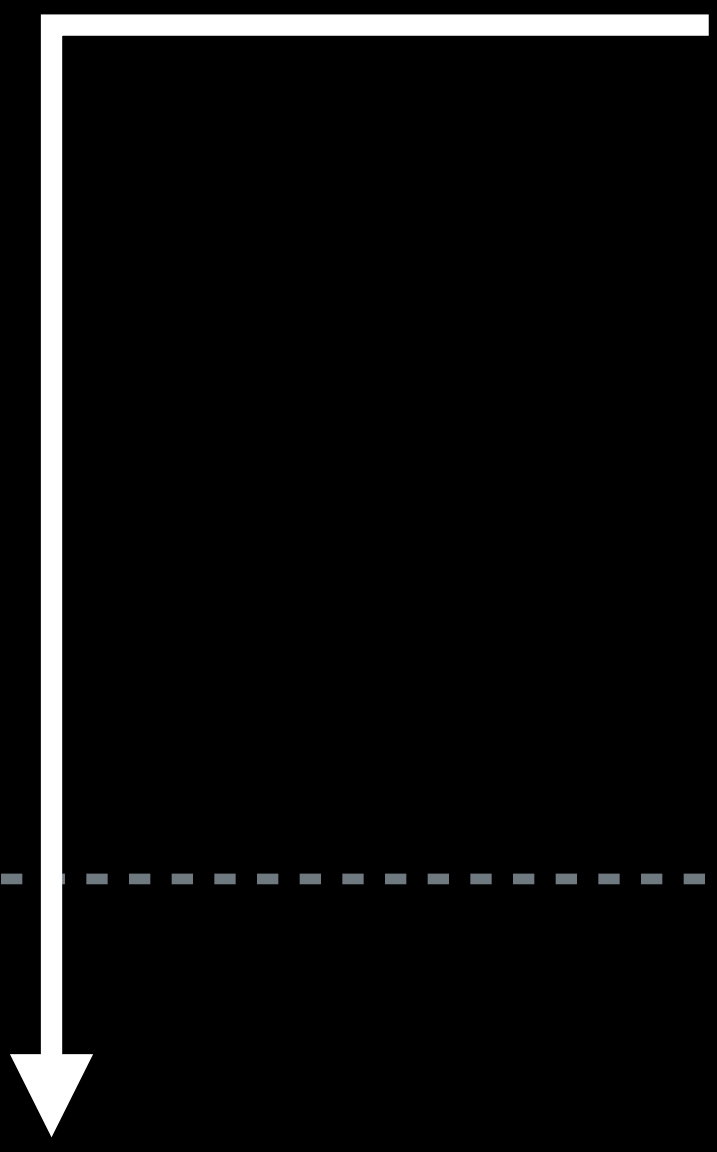
State

SwiftUI





User Interaction



Action

View

Mutation



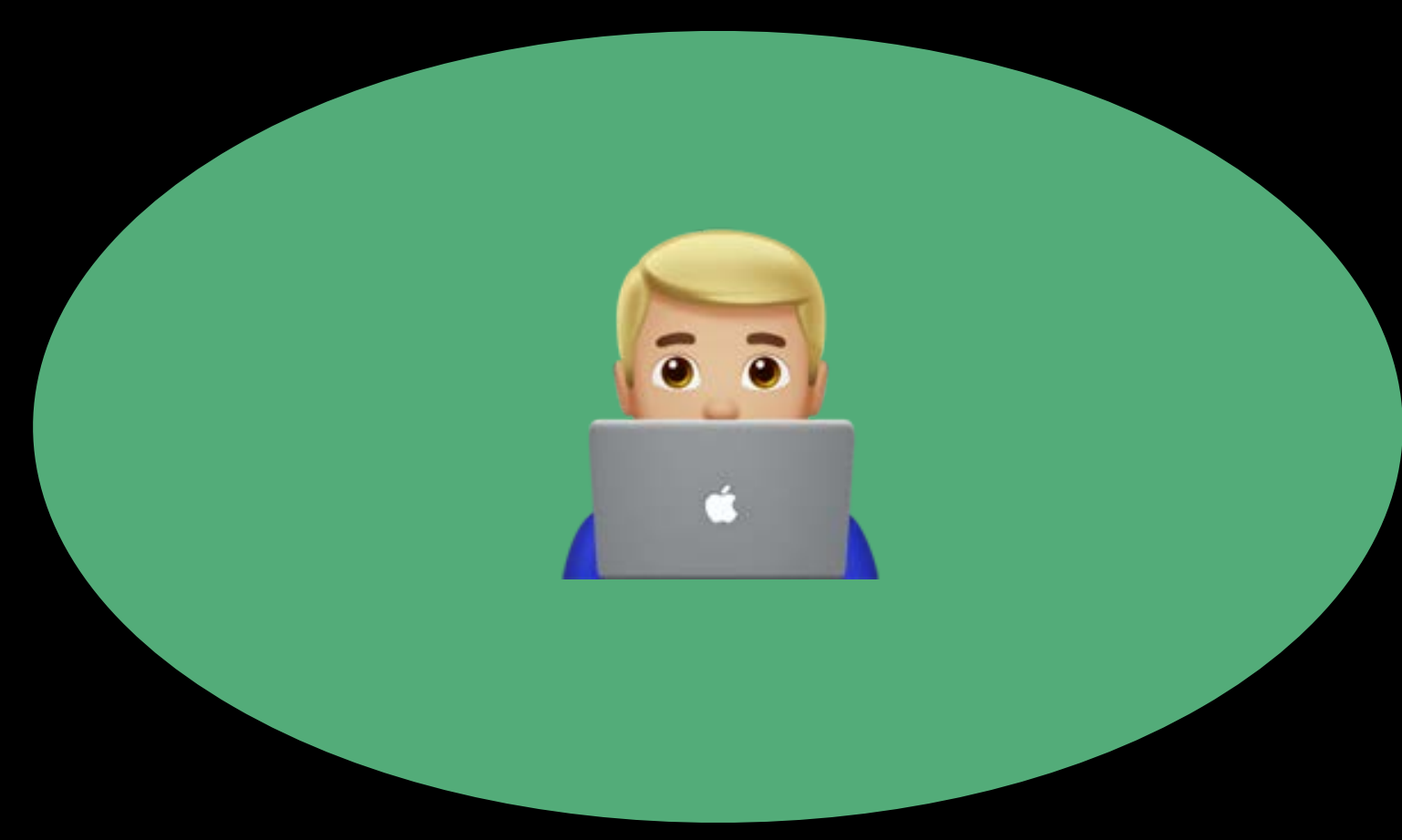
State

Updates

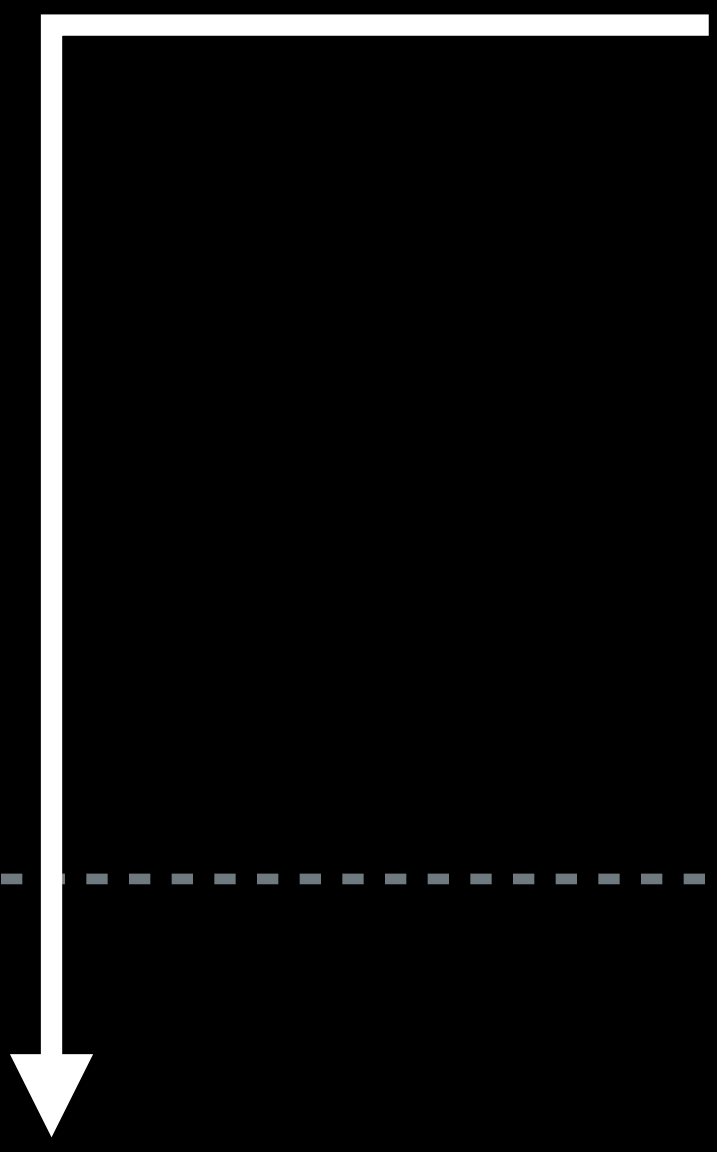


SwiftUI





User Interaction



Action

View

Mutation



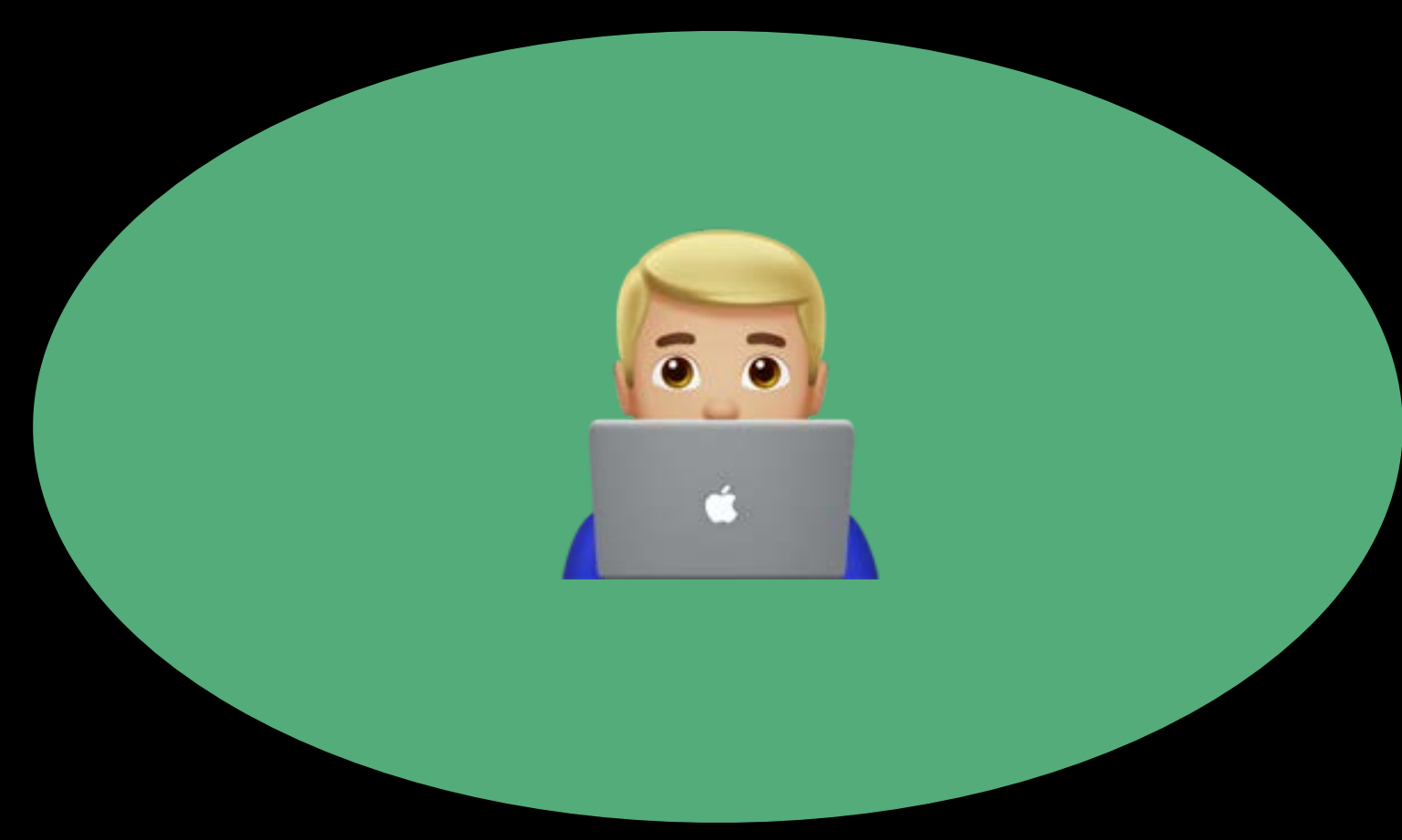
State

Updates



SwiftUI





User Interaction

Render



SwiftUI

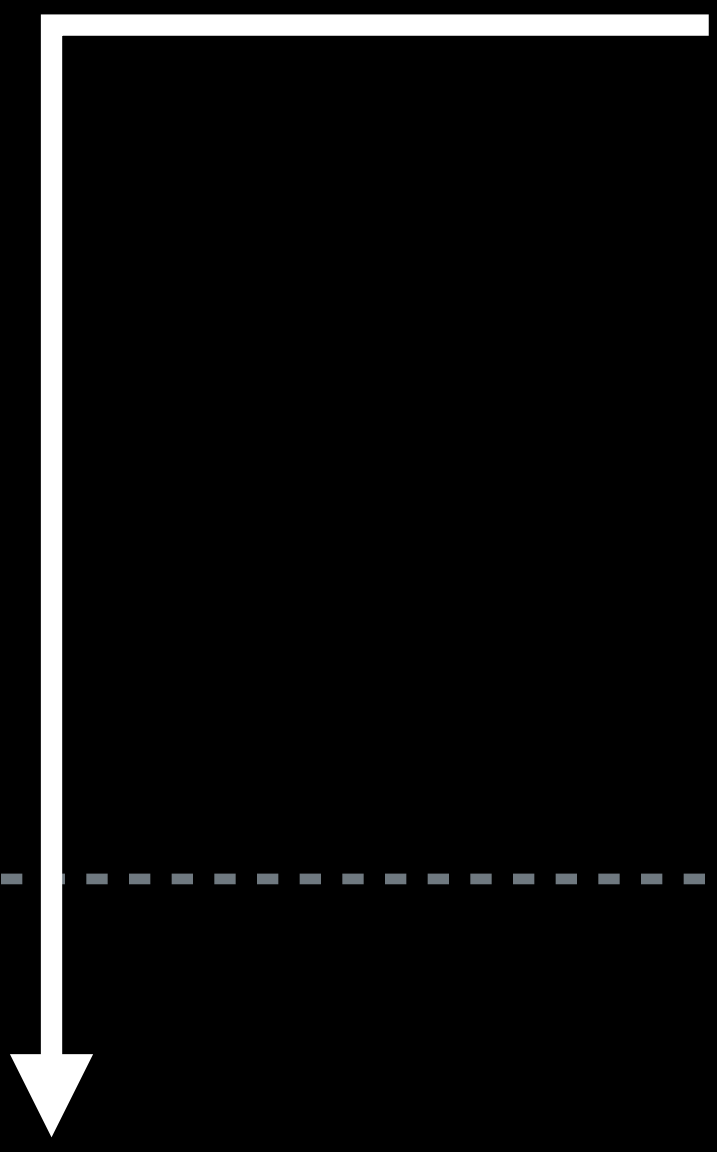
Action

View

Mutation

State

Updates



```
struct PlayerView : View {
  let episode: Episode
  @State private var isPlaying: Bool = false

  var body: some View {
    VStack {
      Text(episode.title).foregroundColor(isPlaying ? .white : .gray)
      Text(episode.showTitle).font(.caption).foregroundColor(.gray)

      Button(action: { self.isPlaying.toggle() }) {
        Image(systemName: isPlaying ? "pause.circle" : "play.circle")
      }
    }
  }
}
```



```
struct PlayerView : View {
  let episode: Episode
  @State private var isPlaying: Bool = false

  var body: some View {
    VStack {
      Text(episode.title).foregroundColor(isPlaying ? .white : .gray)
      Text(episode.showTitle).font(.caption).foregroundColor(.gray)

      Button(action: { self.isPlaying.toggle() }) {
        Image(systemName: isPlaying ? "pause.circle" : "play.circle")
      }
    }
  }
}
```




```
struct PlayerView : View {
  let episode: Episode
  @State private var isPlaying: Bool = false

  var body: some View {
    VStack {
      Text(episode.title).foregroundColor(isPlaying ? .white : .gray)
      Text(episode.showTitle).font(.caption).foregroundColor(.gray)

      Button(action: { self.isPlaying.toggle() }) {
        Image(systemName: isPlaying ? "pause.circle" : "play.circle")
      }
    }
  }
}
```




```

struct PlayerView : View {
    let episode: Episode
    @State private var isPlaying: Bool = false

    var body: some View {
        VStack {
            Text(episode.title).foregroundColor(isPlaying ? .white : .gray)
            Text(episode.showTitle).font(.caption).foregroundColor(.gray)

            Button(action: { self.isPlaying.toggle() }) {
                Image(systemName: isPlaying ? "pause.circle" : "play.circle")
            }
        }
    }
}

```



```
struct PlayerView : View {
  let episode: Episode
  @State private var isPlaying: Bool = false

  var body: some View {
    VStack {
      Text(episode.title).foregroundColor(isPlaying ? .white : .gray)
      Text(episode.showTitle).font(.caption).foregroundColor(.gray)

      PlayButton()
    }
  }
}
```



```
struct PlayerView : View {
  let episode: Episode
  @State private var isPlaying: Bool = false

  var body: some View {
    VStack {
      Text(episode.title).foregroundColor(isPlaying ? .white : .gray)
      Text(episode.showTitle).font(.caption).foregroundColor(.gray)
      PlayButton()
    }
  }
}
```



```
struct PlayButton : View {
    @State private var isPlaying: Bool = false

    var body: some View {
        Button(action: {
            self.isPlaying.toggle()
        }) {
            Image(systemName: isPlaying ? "pause.circle" : "play.circle")
        }
    }
}
```



```
struct PlayButton : View {
    @State private var isPlaying: Bool = false

    var body: some View {
        Button(action: {
            self.isPlaying.toggle()
        }) {
            Image(systemName: isPlaying ? "pause.circle" : "play.circle")
        }
    }
}
```

```
struct PlayButton : View {
  @State private var isPlaying: Bool = false

  var body: some View {
    Button(action: {
      self.isPlaying.toggle()
    }) {
      Image(systemName: isPlaying ? "pause.circle" : "play.circle")
    }
  }
}
```



@Binding Property Wrapper

Read and write without ownership

Derivable from @State

```
@Binding var isPlaying: Bool
```

```
struct PlayButton : View {
    @Binding var isPlaying: Bool

    var body: some View {
        Button(action: {
            self.isPlaying.toggle()
        }) {
            Image(systemName: isPlaying ? "pause.circle" : "play.circle")
        }
    }
}
```



```
struct PlayButton : View {
  @Binding var isPlaying: Bool

  var body: some View {
    Button(action: {
      self.isPlaying.toggle()
    }) {
      Image(systemName: isPlaying ? "pause.circle" : "play.circle")
    }
  }
}
```



```
struct PlayerView : View {
  let episode: Episode
  @State private var isPlaying: Bool = false

  var body: some View {
    VStack {
      Text(episode.title).foregroundColor(isPlaying ? .white : .gray)
      Text(episode.showTitle).font(.caption).foregroundColor(.gray)

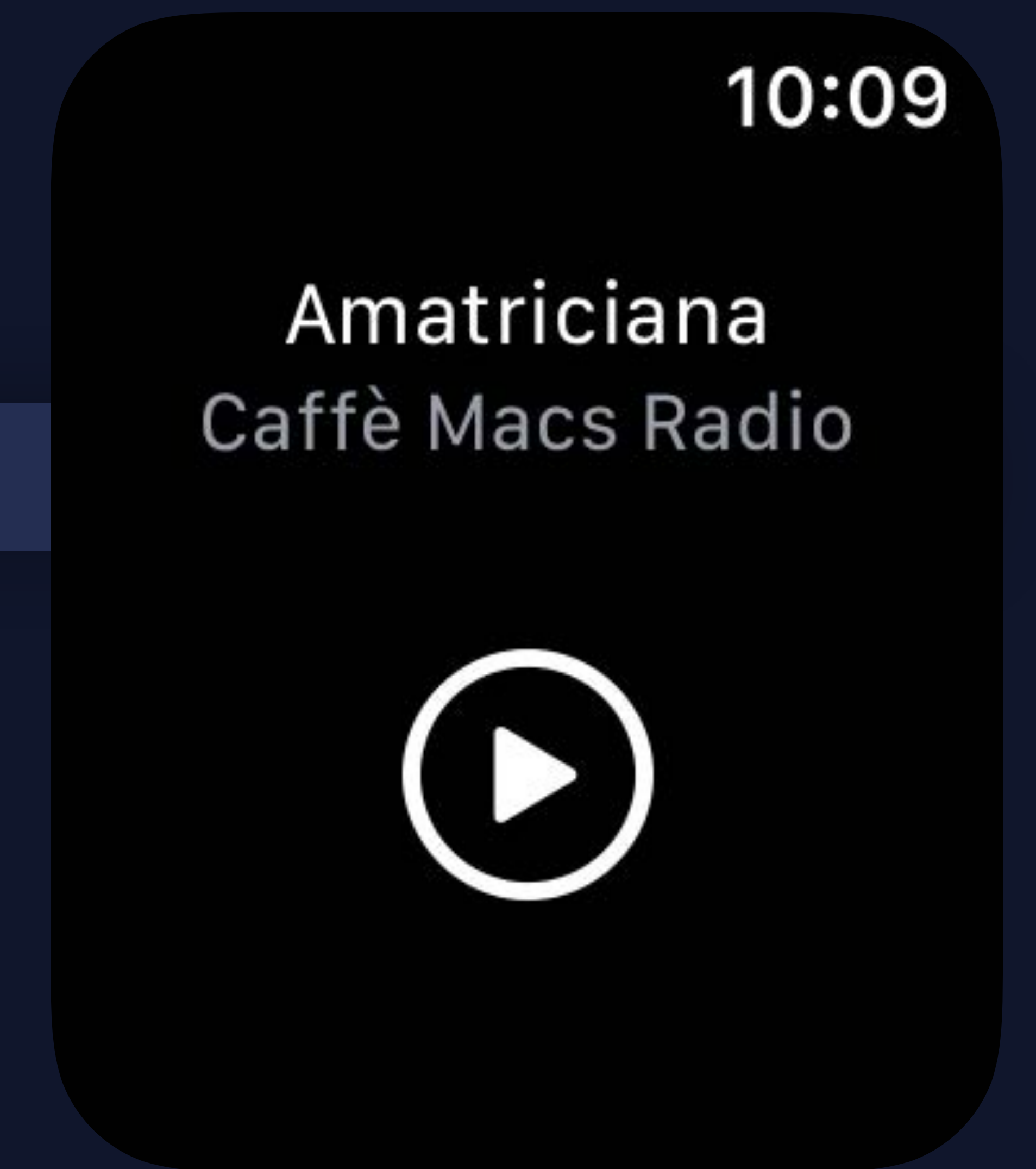
      PlayButton(isPlaying: $isPlaying)
    }
  }
}
```



```
struct PlayerView : View {
    let episode: Episode
    @State private var isPlaying: Bool = false

    var body: some View {
        VStack {
            Text(episode.title).foregroundColor(isPlaying ? .white : .gray)
            Text(episode.showTitle).font(.caption).foregroundColor(.gray)

            PlayButton(isPlaying: $isPlaying)
        }
    }
}
```

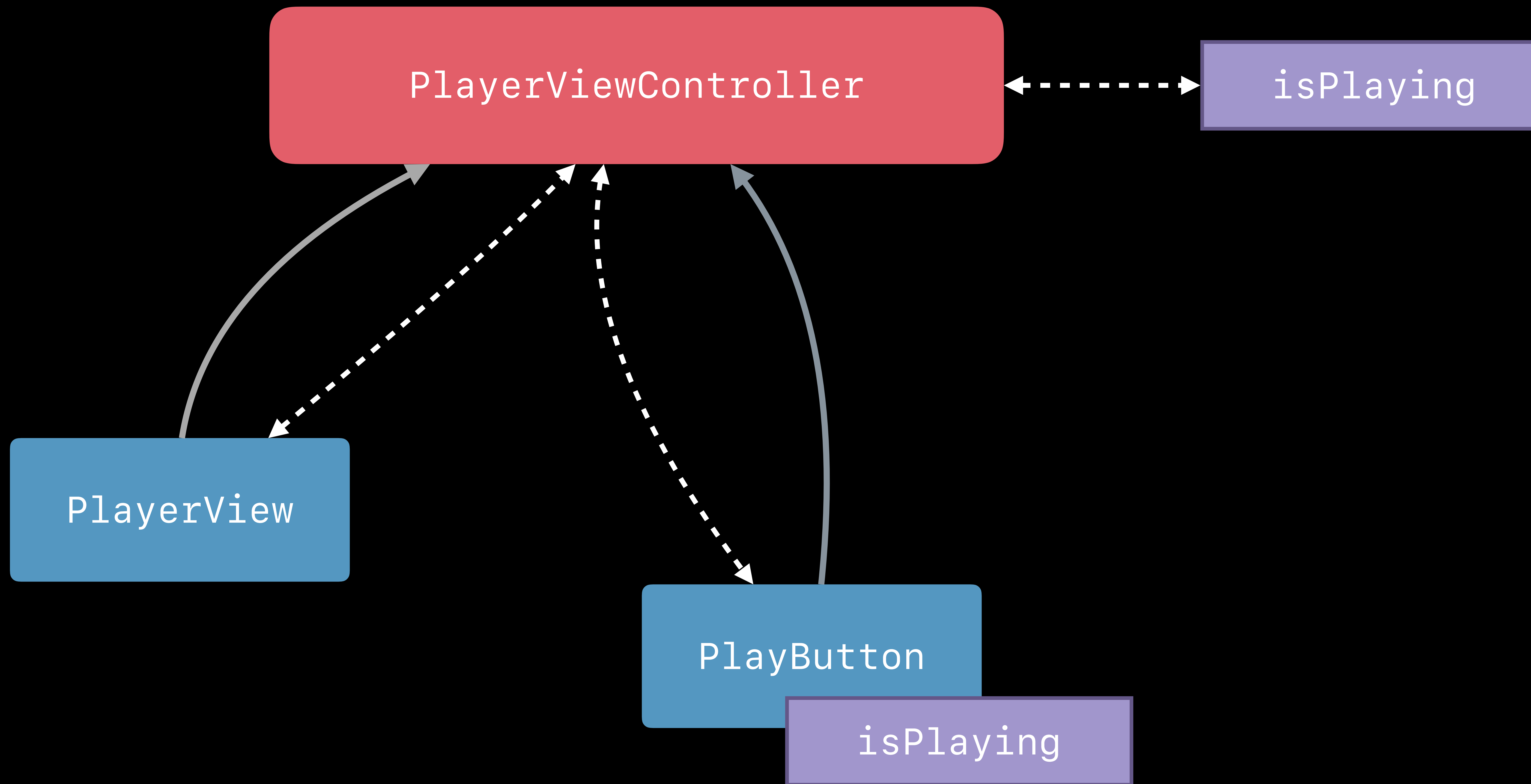


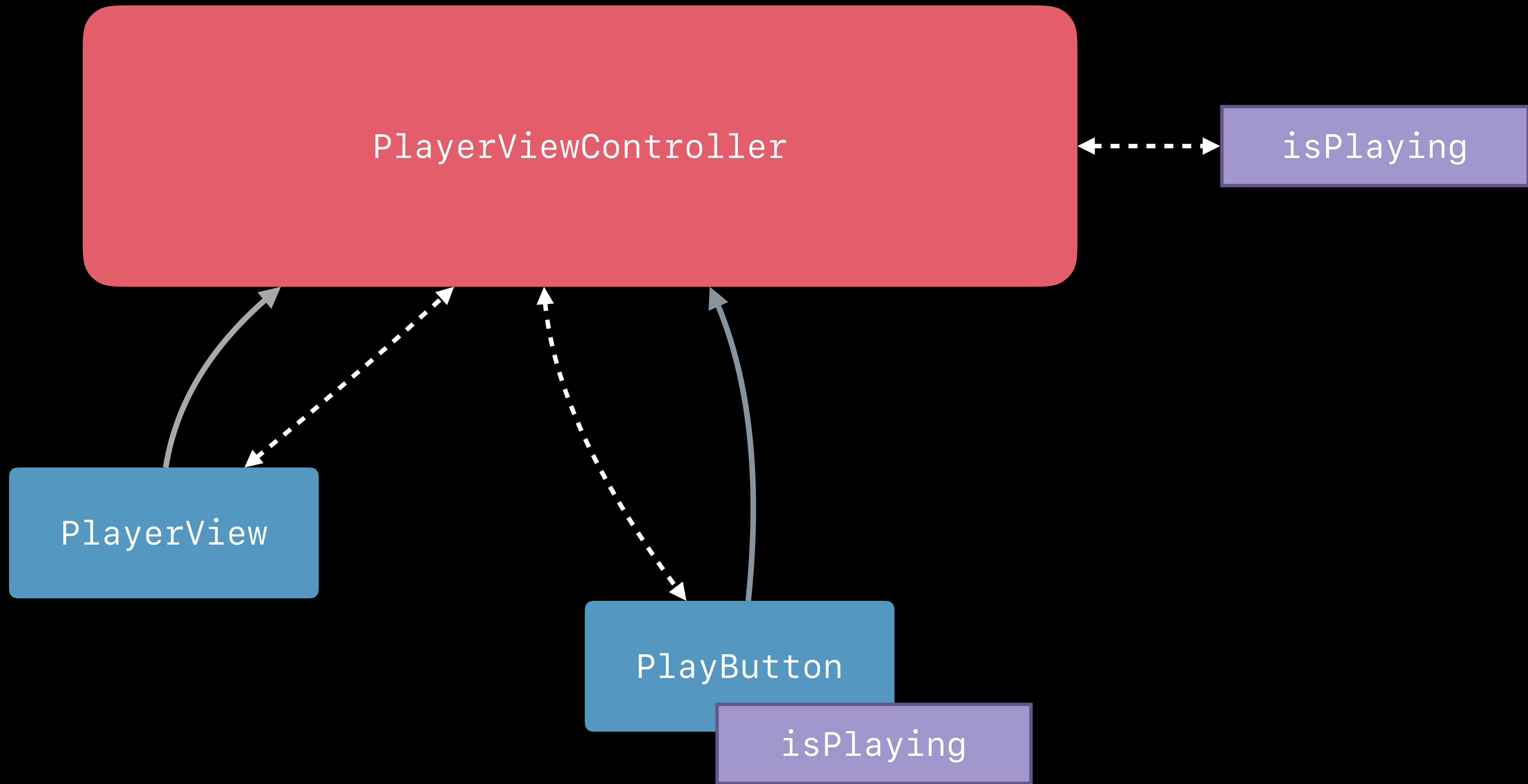
```
struct PlayerView : View {
  let episode: Episode
  @State private var isPlaying: Bool = false

  var body: some View {
    VStack {
      Text(episode.title).foregroundColor(isPlaying ? .white : .gray)
      Text(episode.showTitle).font(.caption).foregroundColor(.gray)
      PlayButton(isPlaying: $isPlaying)
    }
  }
}
```



PlayerViewController

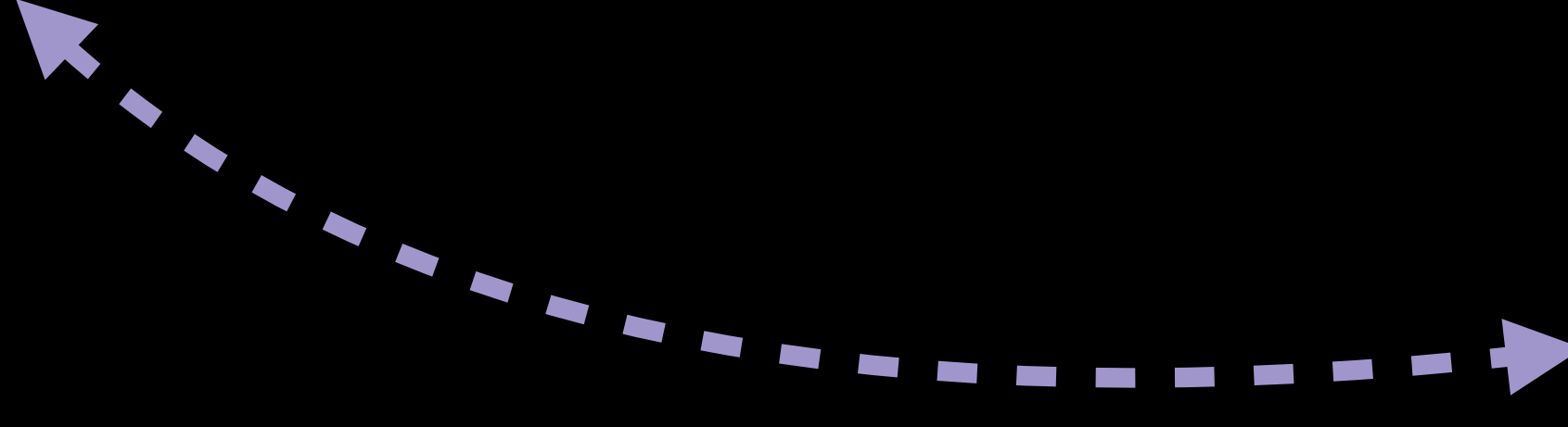
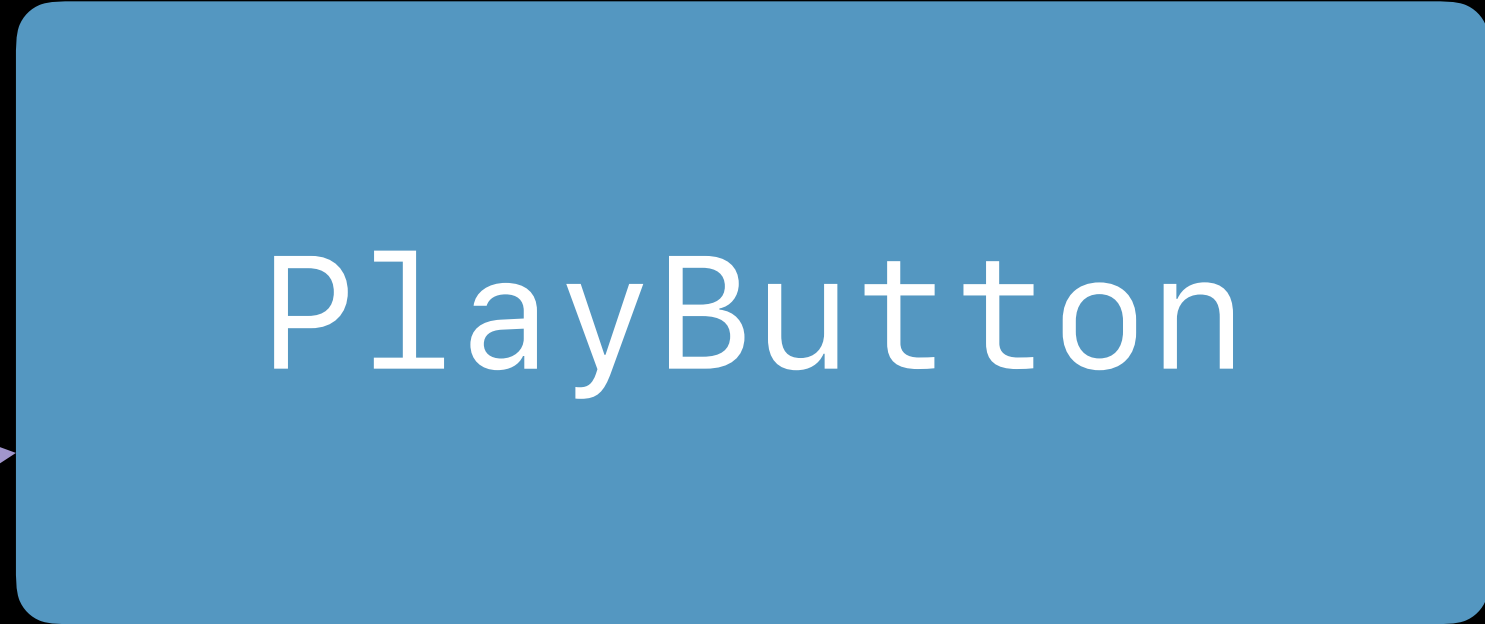
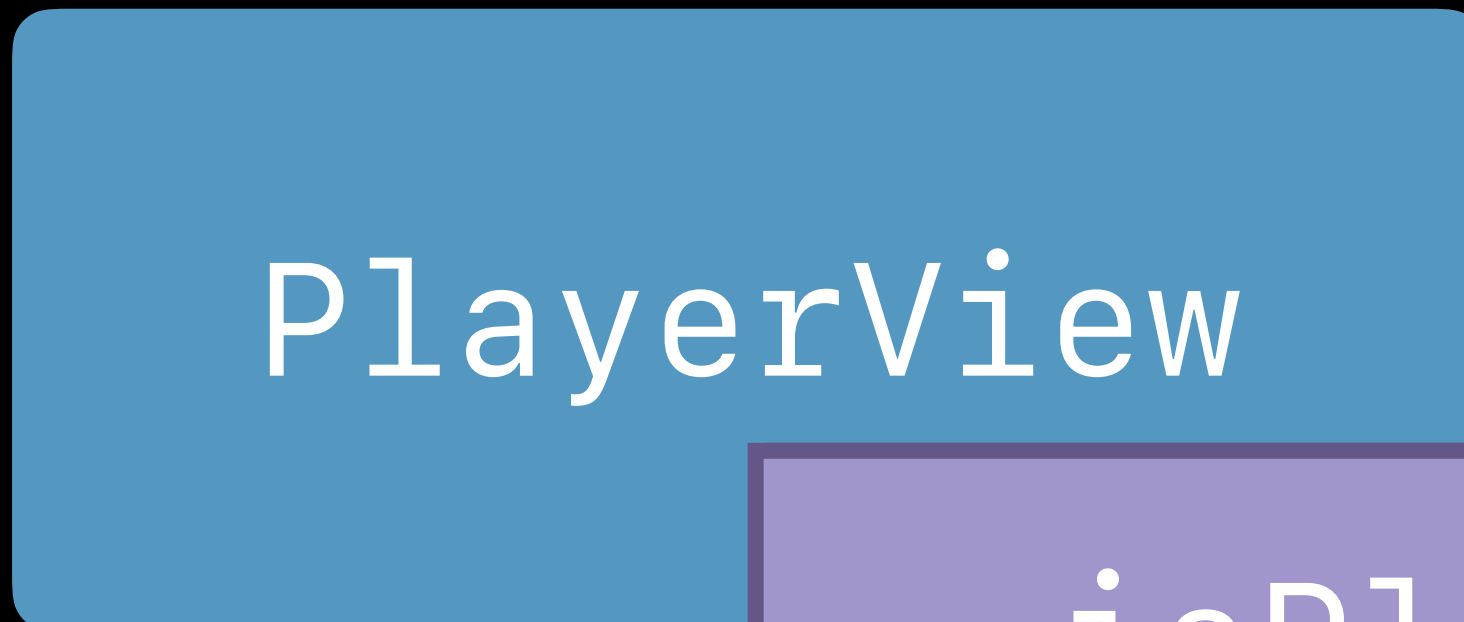


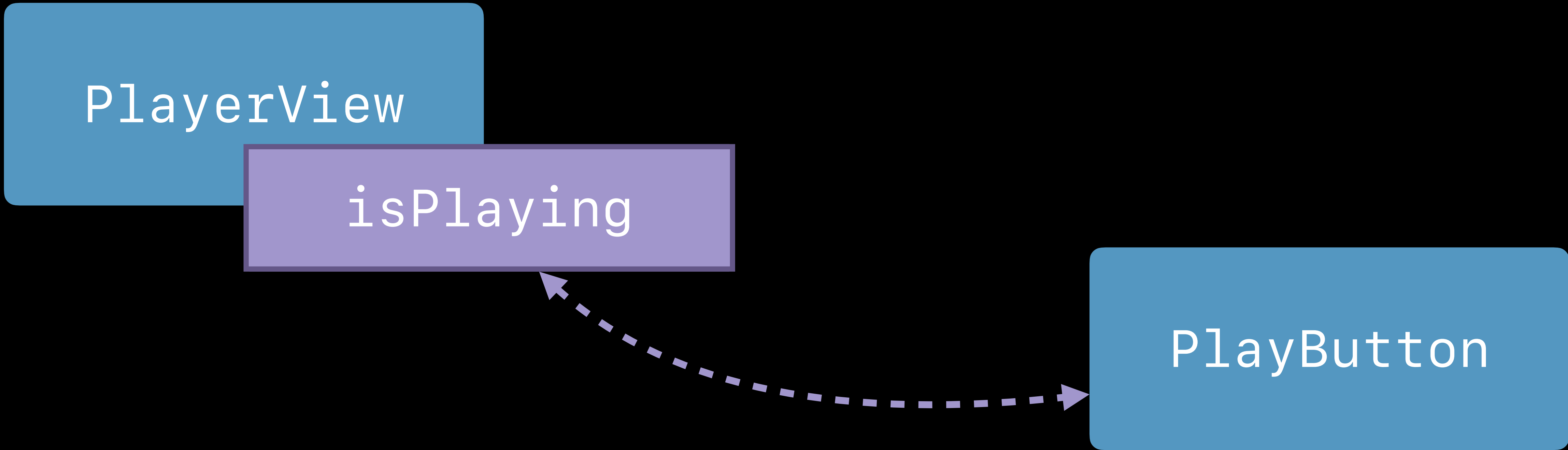


PlayerViewController

PlayerView

isPlaying







API



API

```
public struct Toggle<Label> : View {  
    public init(  
        isOn: Binding<Bool>,  
        label: () -> Label  
    )  
}
```

```
public struct TextField : View {  
    init(  
        _ text: Binding<String>  
    )  
}
```

```
public struct Slider : View {  
    public init<V : BinaryFloatingPoint>(  
        value: Binding<V>  
    )  
}
```

Layout

Visual Effects

Navigation

Views are for Data

Gestures

Drawing

Animations


```
// Animated Changes

struct PlayButton : View {
    @Binding var isPlaying: Bool

    var body: some View {
        Button(action: {
            self.isPlaying.toggle()
        }) {
            Image(systemName: isPlaying ? "pause.circle" : "play.circle")
        }
    }
}
```



```
// Animated Changes

struct PlayButton : View {
    @Binding var isPlaying: Bool

    var body: some View {
        Button(action: {
            withAnimation { self.isPlaying.toggle() }
        }) {
            Image(systemName: isPlaying ? "pause.circle" : "play.circle")
        }
    }
}
```



```
// Animated Changes
```

```
struct PlayButton : View {  
    @Binding var isPlaying: Bool  
  
    var body: some View {  
        Button(action: {  
            withAnimation { self.isPlaying.toggle() }  
        }) {  
            Image(systemName: isPlaying ? "pause.circle" : "play.circle")  
        }  
    }  
}
```



Working With External Data

Raj Ramamurthy, SwiftUI Engineer

Tools for Data Flow

@Environment

Property



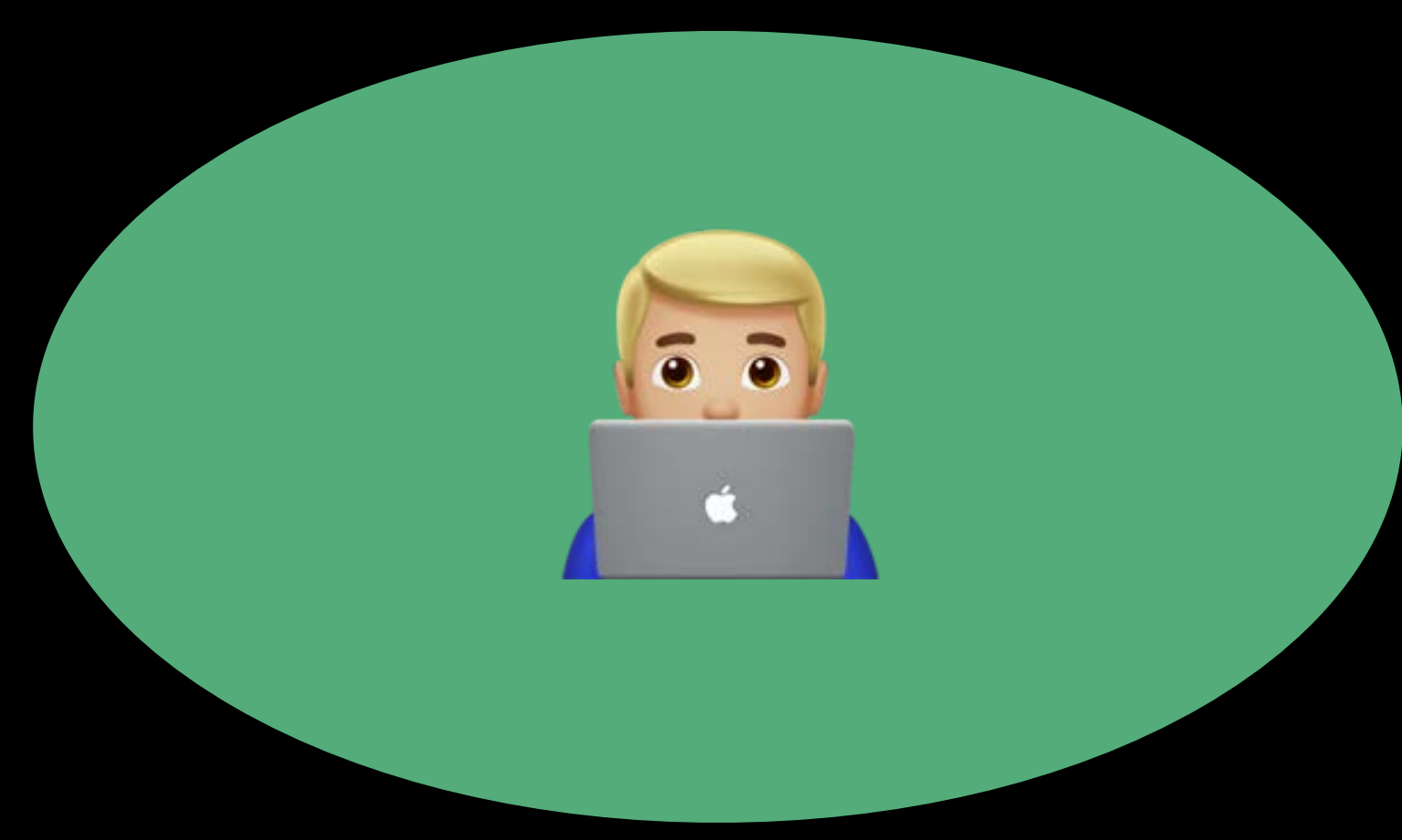
@Binding

BindableObject

@State



SwiftUI



User Interaction

Render

SwiftUI

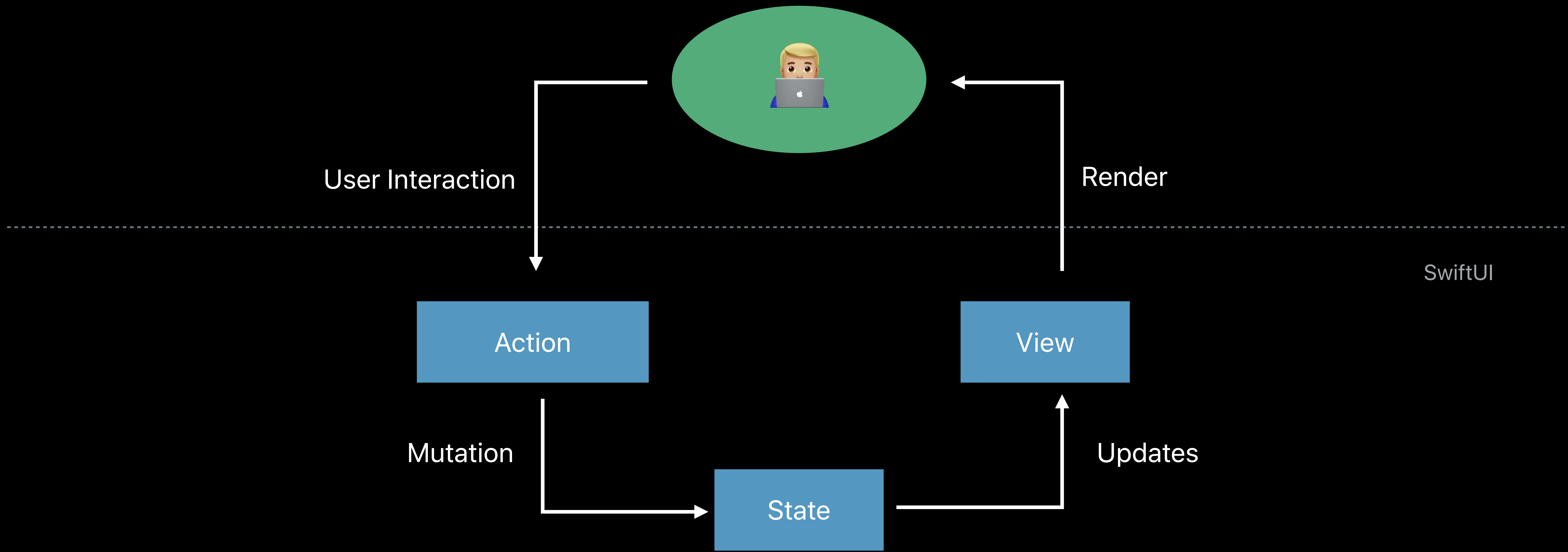
Action

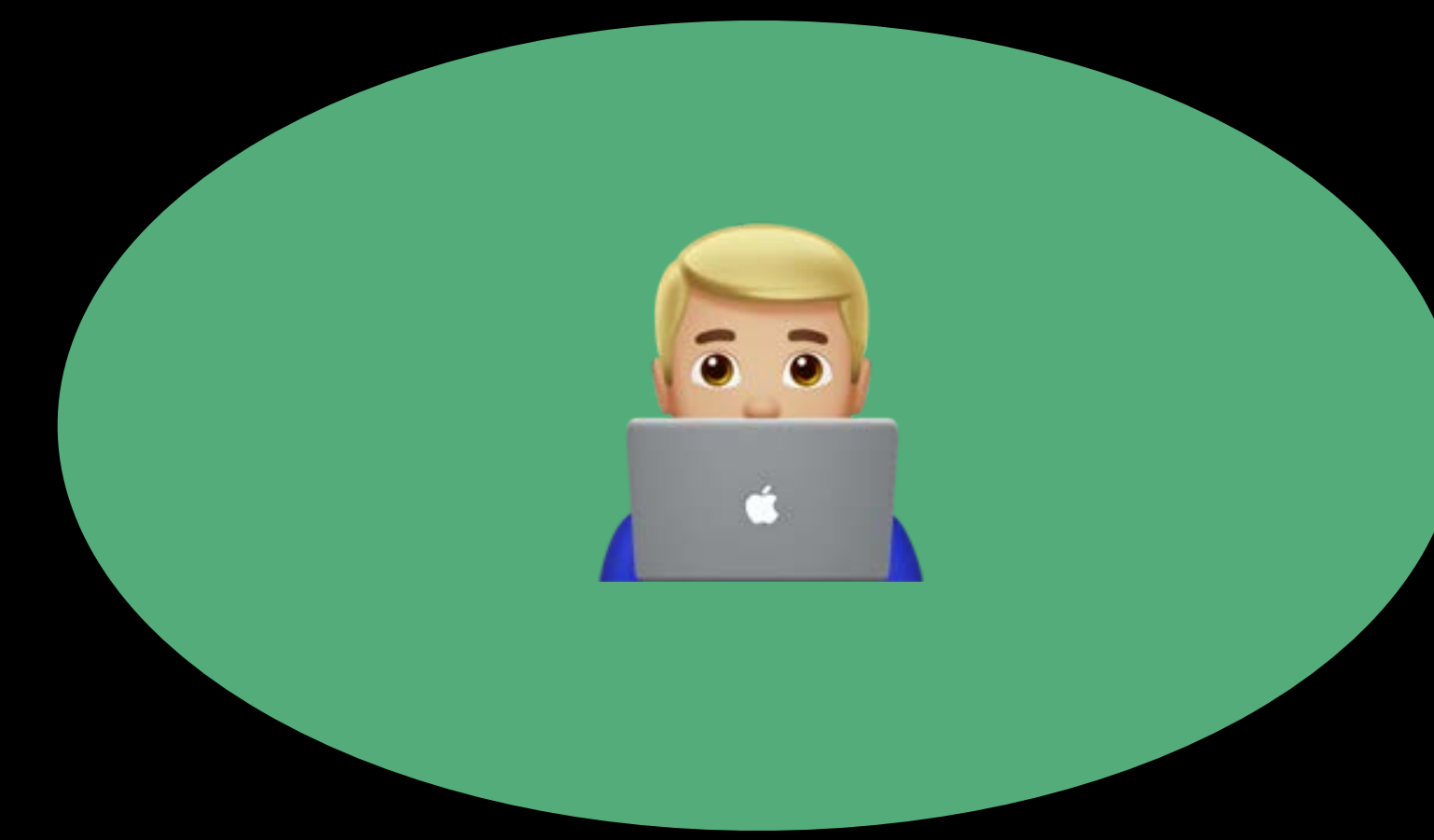
View

Mutation

State

Updates





User Interaction

Render



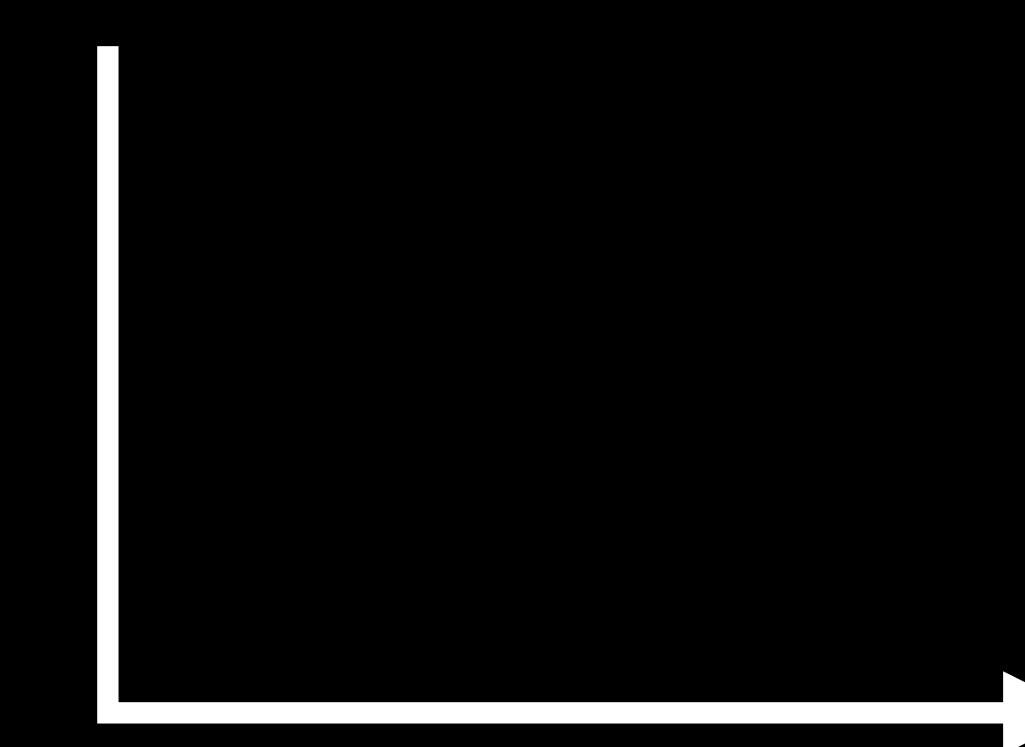
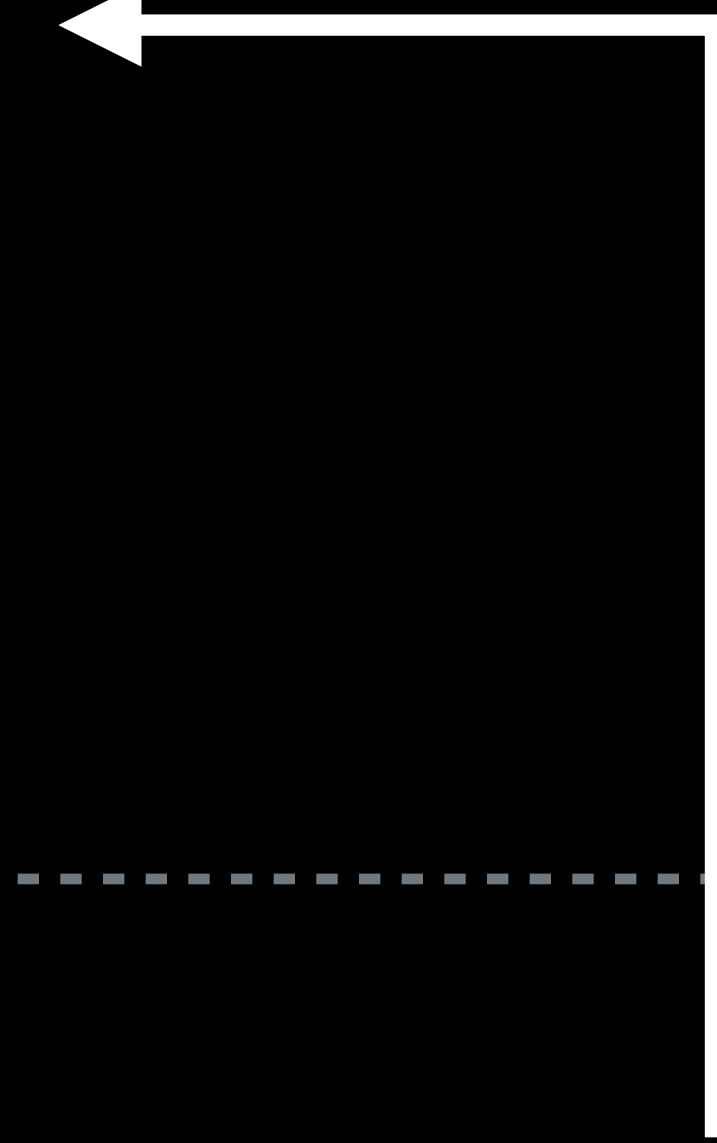
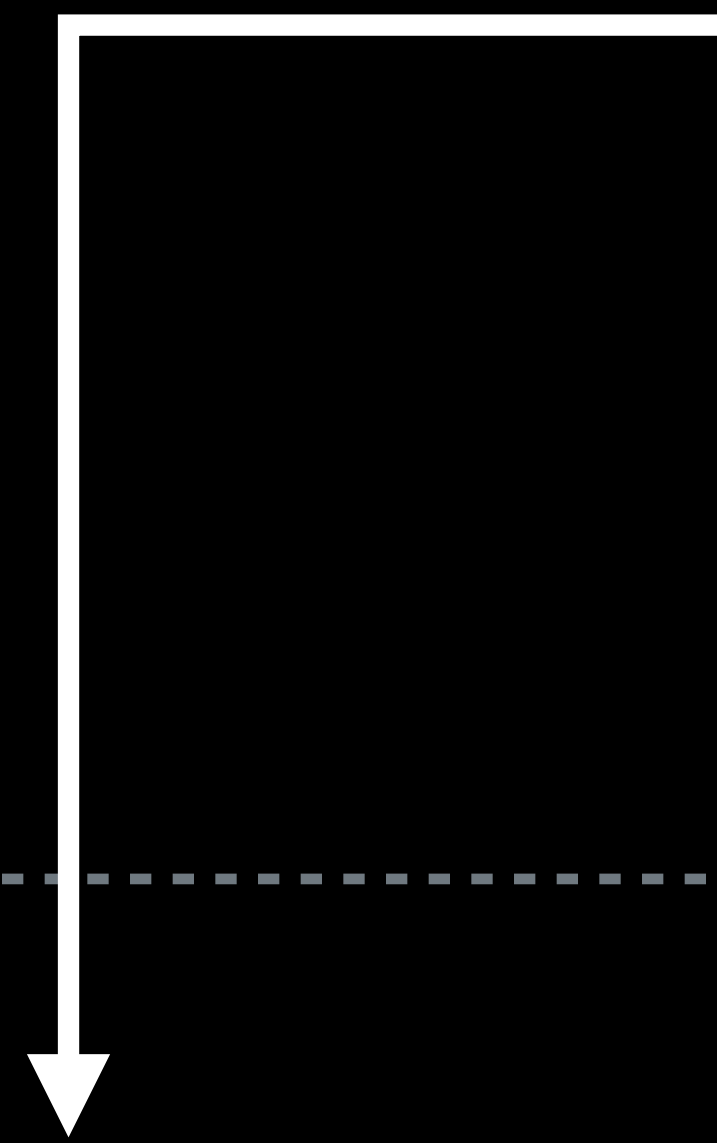
SwiftUI

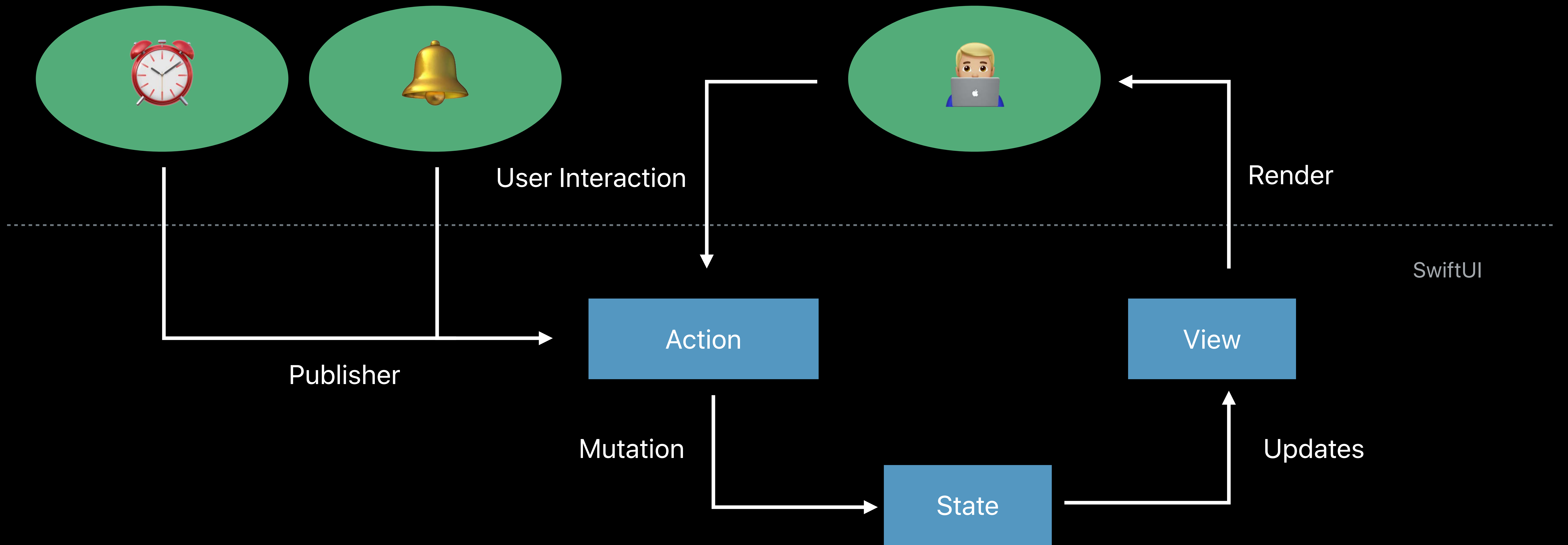


Mutation



Updates





Combine Publisher

Single abstraction

Main thread: use `.receive(on:)`

Introducing Combine and Advances in Foundation

Thursday, 10:00

Combine in Practice

Thursday, 2:00

```
struct PlayerView : View {
  let episode: Episode
  @State private var isPlaying: Bool = true
  @State private var currentTime: TimeInterval = 0.0

  var body: some View {
    VStack {
      Text(episode.title).foregroundColor(isPlaying ? .white : .gray)
      Text(episode.showTitle).font(.caption).foregroundColor(.gray)

      PlayButton(isPlaying: $isPlaying)

      Text("\(currentTime, formatter: currentTimeFormatter)")
    }
  }
}
```




```

struct PlayerView : View {
    let episode: Episode
    @State private var isPlaying: Bool = true
    @State private var currentTime: TimeInterval = 0.0

    var body: some View {
        VStack {
            Text(episode.title).foregroundColor(isPlaying ? .white : .gray)
            Text(episode.showTitle).font(.caption).foregroundColor(.gray)

            PlayButton(isPlaying: $isPlaying)

            Text("\(currentTime, formatter: currentTimeFormatter)")
        }
    }
}

```




```
struct PlayerView : View {
    let episode: Episode
    @State private var isPlaying: Bool = true
    @State private var currentTime: TimeInterval = 0.0

    var body: some View {
        VStack {
            // ...
            Text("\(playhead, formatter: currentTimeFormatter)")
        }
        .onReceive(PodcastPlayer.currentTimePublisher) { newCurrentTime in
            self.currentTime = newCurrentTime
        }
    }
}
```



```

struct PlayerView : View {
    let episode: Episode
    @State private var isPlaying: Bool = true
    @State private var currentTime: TimeInterval = 0.0

    var body: some View {
        VStack {
            // ...
            Text("\(playhead, formatter: currentTimeFormatter)")
        }
        .onReceive(PodcastPlayer.currentTimePublisher) { newCurrentTime in
            self.currentTime = newCurrentTime
        }
    }
}

```



BindableObject Protocol

External

Reference type

Great for the model you already have

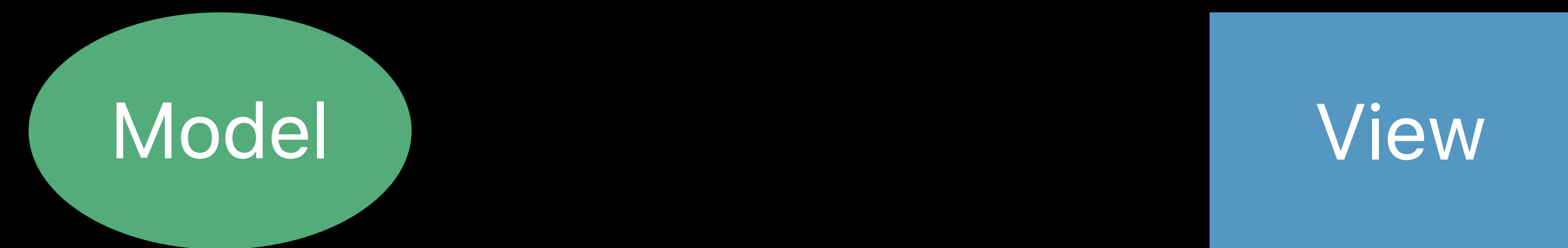
```
class PodcastPlayerStore {  
    var currentTime: TimeInterval  
    var isPlaying: Bool  
    var currentEpisode: Episode  
  
    func advance() { ... }  
    func skipForward() { ... }  
    func skipBackward() { ... }  
}
```

```
class PodcastPlayerStore : BindableObject {  
  
    var didChange = PassthroughSubject<Void, Never>()  
  
    // ...  
  
    func advance() {  
        currentEpisode = nextEpisode  
        currentTime = 0.0  
        // Notify subscribers that the player changed  
        didChange.send()  
    }  
}
```



```
class PodcastPlayerStore : BindableObject {  
  
    var didChange = PassthroughSubject<Void, Never>()  
  
    // ...  
  
    func advance() {  
        currentEpisode = nextEpisode  
        currentTime = 0.0  
        // Notify subscribers that the player changed  
        didChange.send()  
    }  
}
```


Creating Dependencies on BindableObject



Creating Dependencies on BindableObject



Creating Dependencies on BindableObject



```
struct MyView : View {  
    @ObjectBinding var model: MyModelObject  
    ...  
}
```

```
MyView(model: modelInstance)
```

Creating Dependencies on BindableObject



```
struct MyView : View {  
    @ObjectBinding var model: MyModelObject  
    ...  
}
```

```
MyView(model: modelInstance)
```

Creating Dependencies on BindableObject



```
struct MyView : View {  
    @ObjectBinding var model: MyModelObject  
    ...  
}
```

```
MyView(model: modelInstance)
```

Creating Dependencies on BindableObject



```
struct MyView : View {  
    @ObjectBinding var model: MyModelObject  
    ...  
}
```

```
MyView(model: modelInstance)
```


Creating Dependencies on BindableObject

Pass directly with `@ObjectBinding`

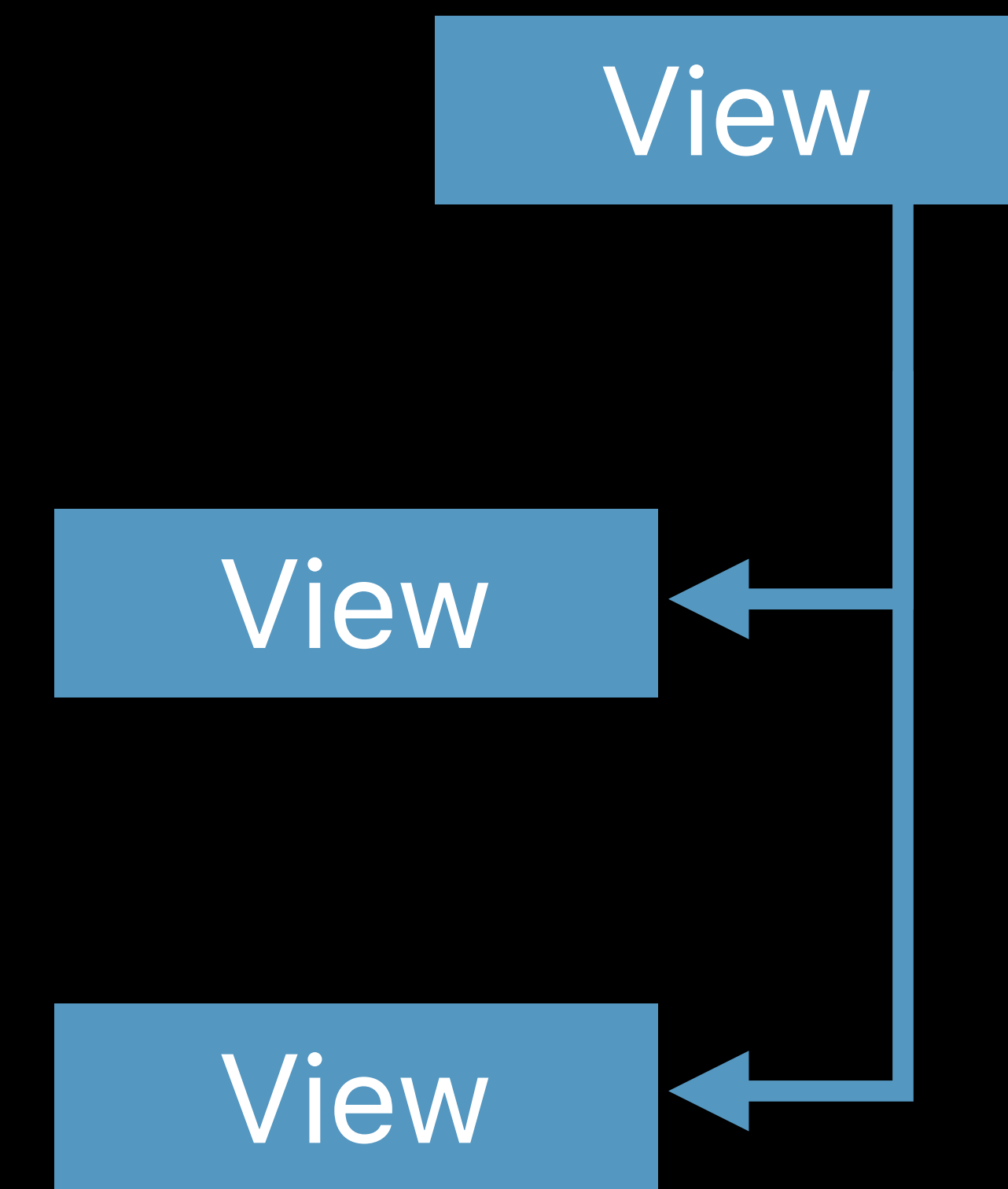
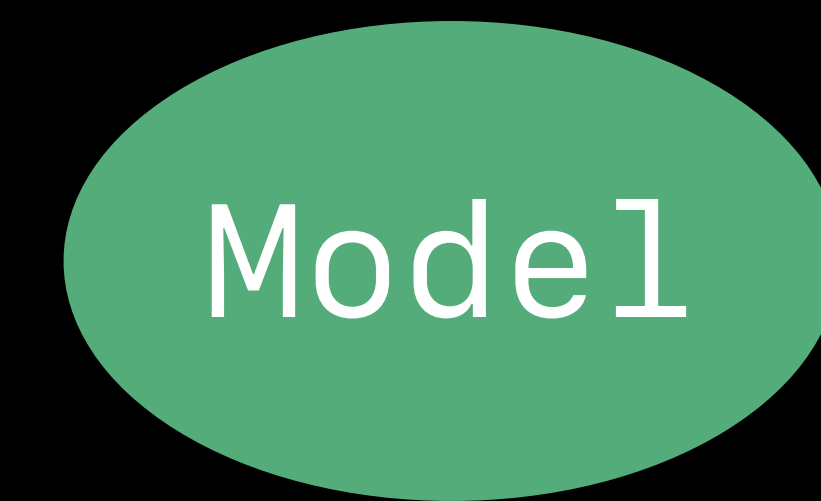
Automatic dependency tracking



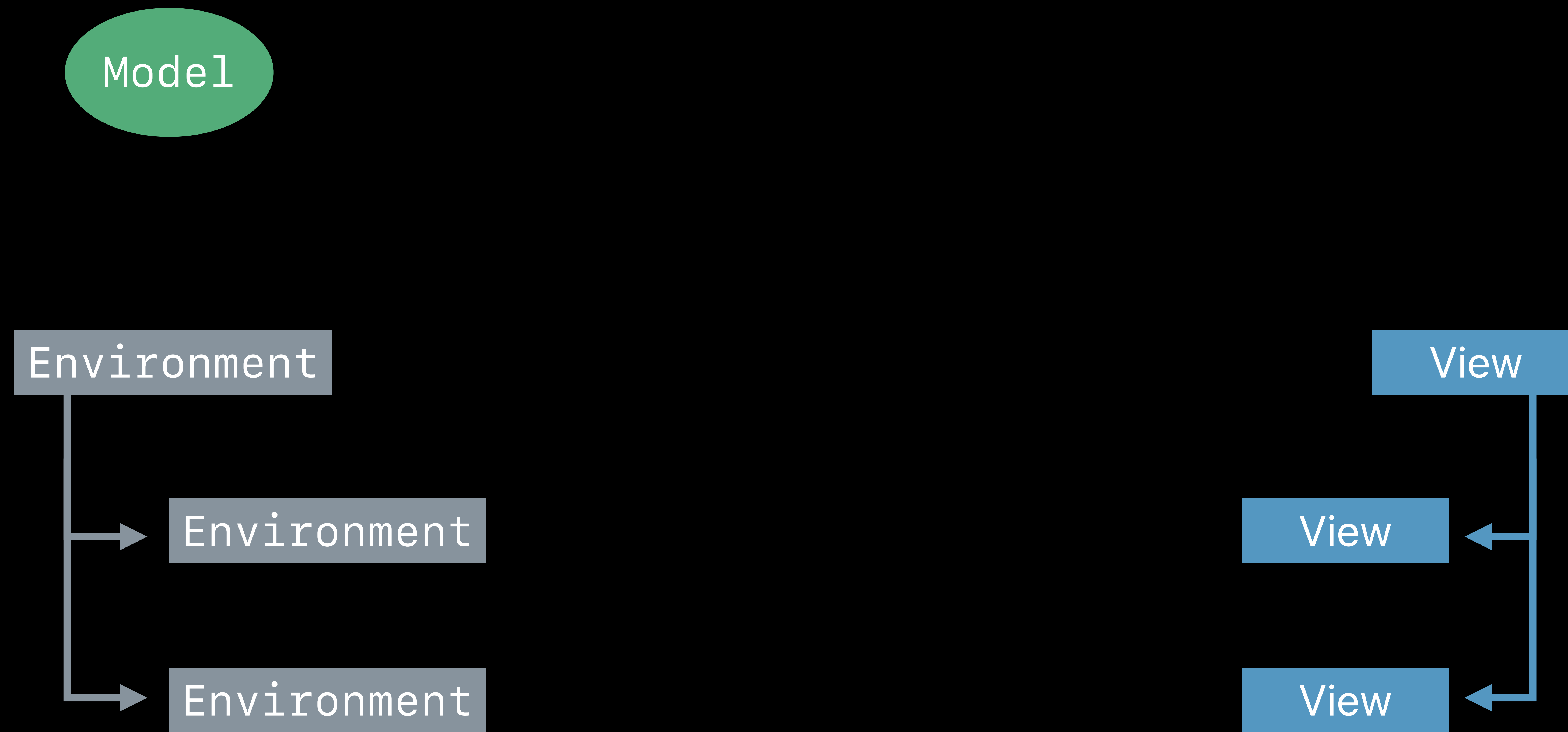
```
struct MyView : View {  
    @ObjectBinding var model: MyModelObject  
    ...  
}
```

```
MyView(model: modelInstance)
```

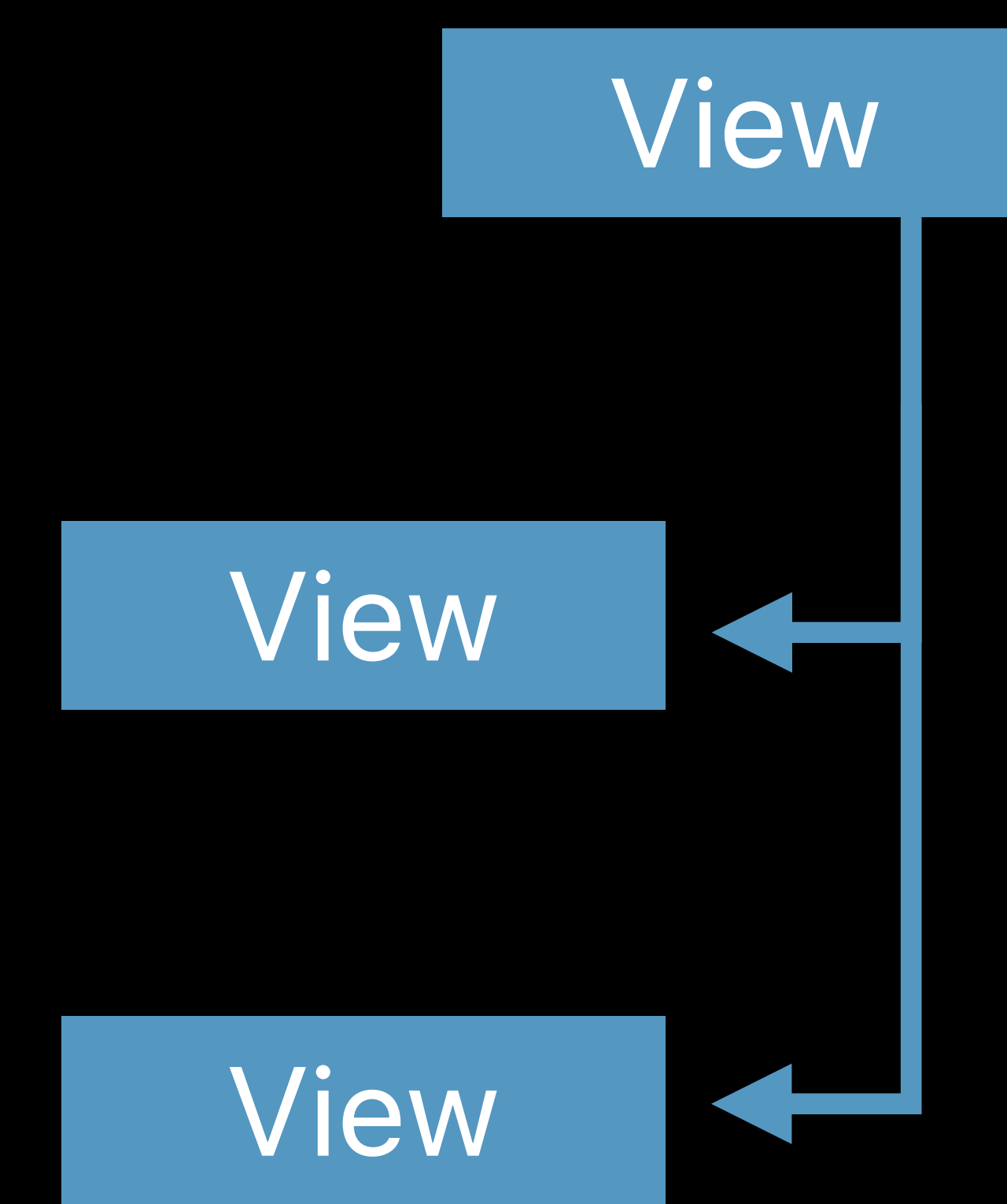
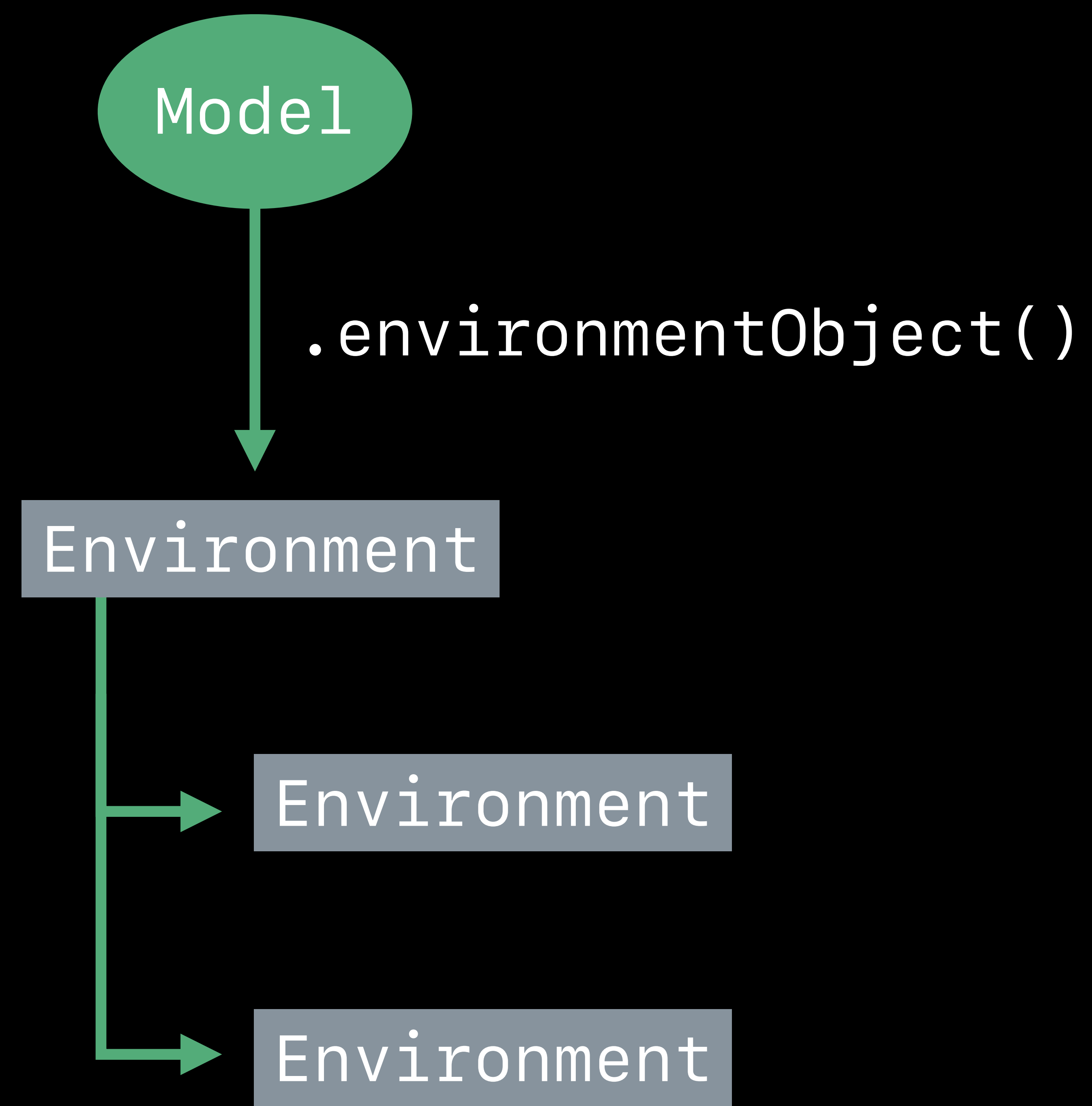
Creating Dependencies Indirectly



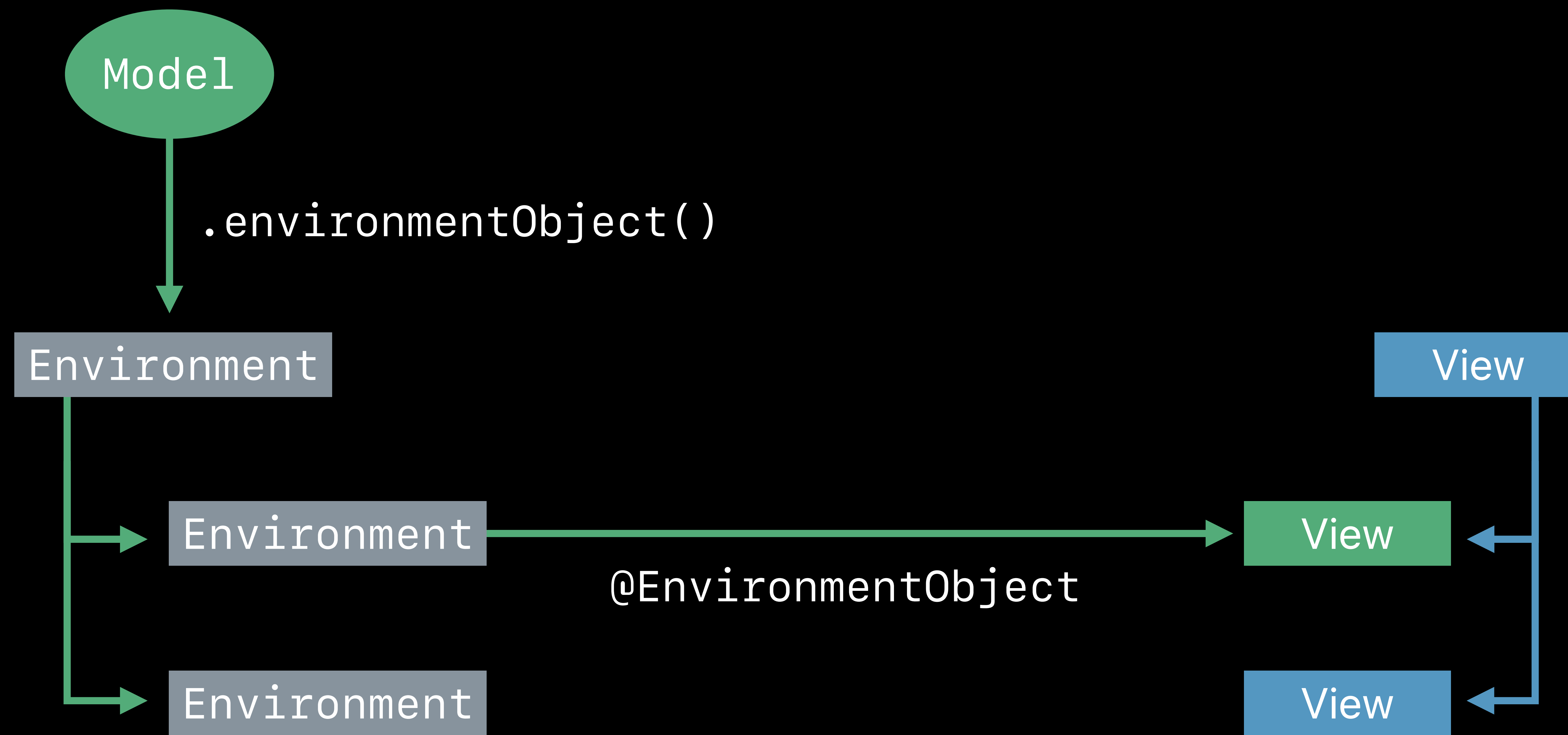
Creating Dependencies Indirectly



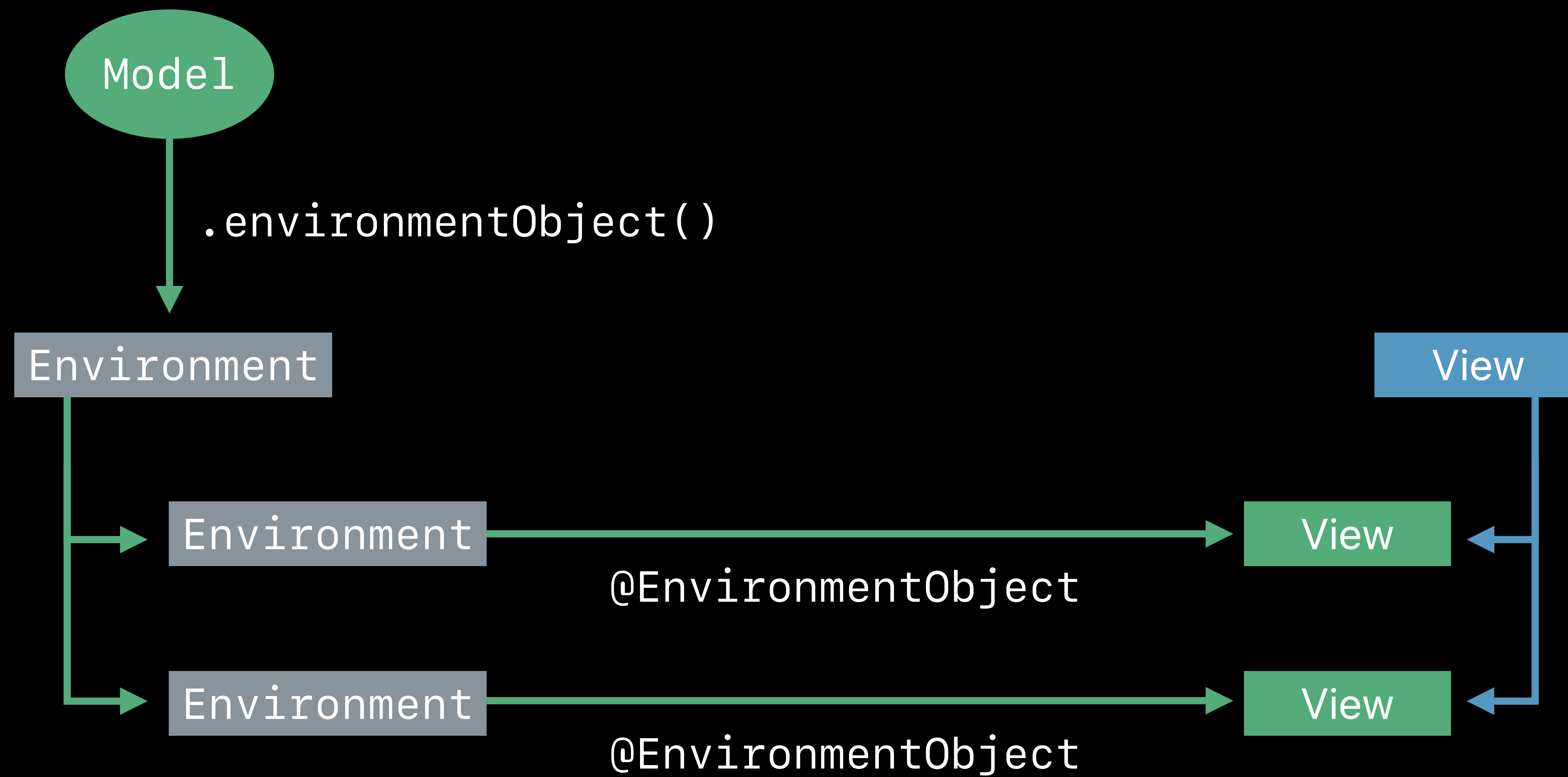
Creating Dependencies Indirectly



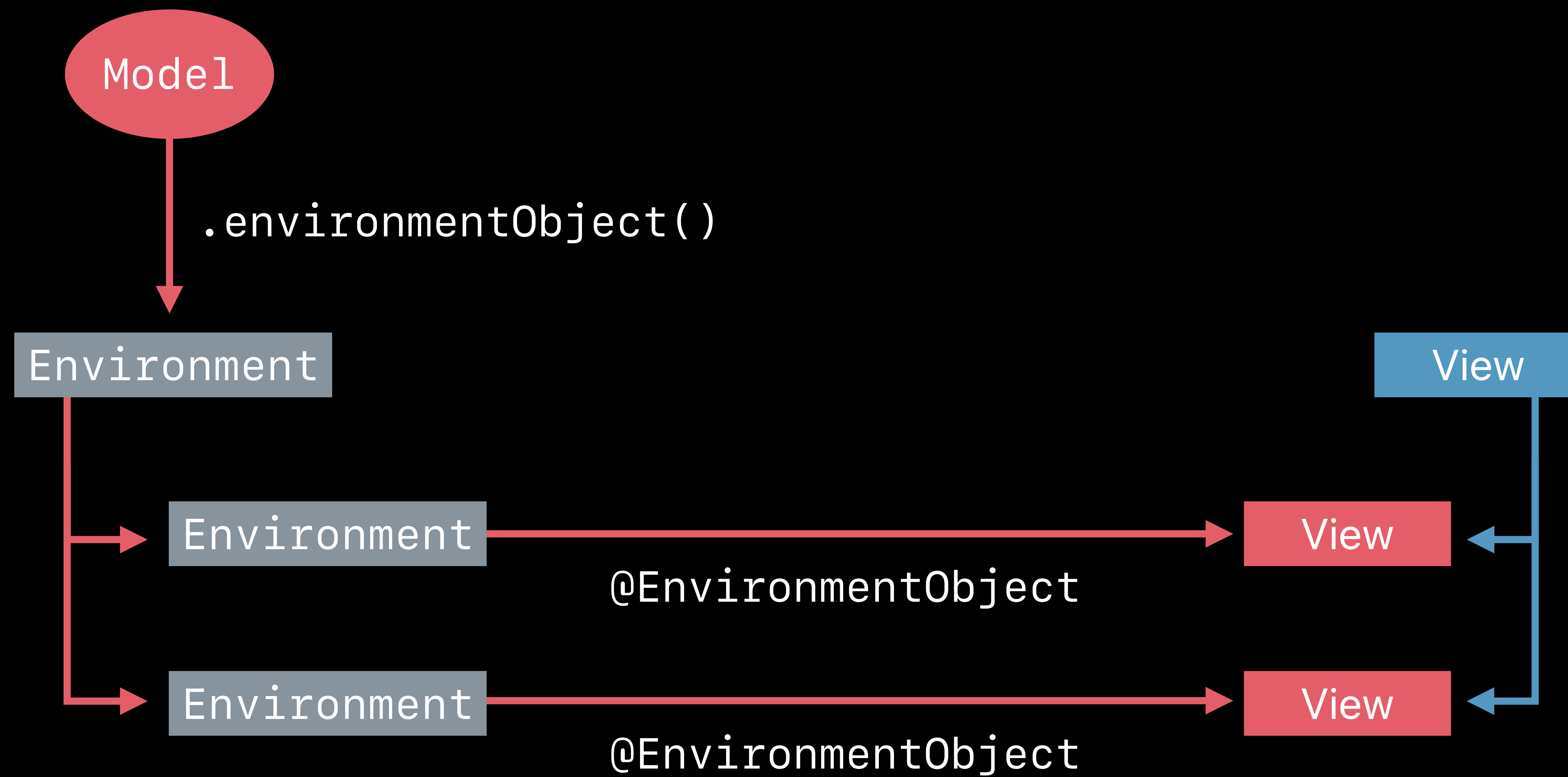
Creating Dependencies Indirectly



Creating Dependencies Indirectly



Creating Dependencies Indirectly



```
struct PlayerView : View {
    @EnvironmentObject var player: PodcastPlayerStore

    var body: some View {
        VStack {
            Text(player.currentEpisode.title)
                .foregroundColor(isPlaying ? .white : .gray)
            Text(player.currentEpisode.showTitle)
                .font(.caption).foregroundColor(.gray)

            PlayButton(isPlaying: $player.isPlaying)

            Text("\(player.currentTime, formatter: playheadTimeFormatter)")
        }
    }
}
```



```
struct PlayerView : View {
  @EnvironmentObject var player: PodcastPlayerStore

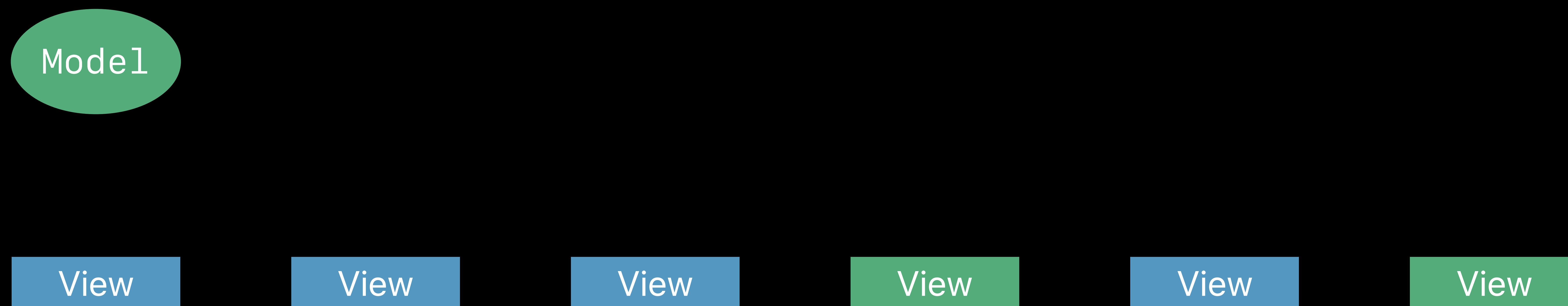
  var body: some View {
    VStack {
      Text(player.currentEpisode.title)
        .foregroundColor(isPlaying ? .white : .gray)
      Text(player.currentEpisode.showTitle)
        .font(.caption).foregroundColor(.gray)

      PlayButton(isPlaying: $player.isPlaying)

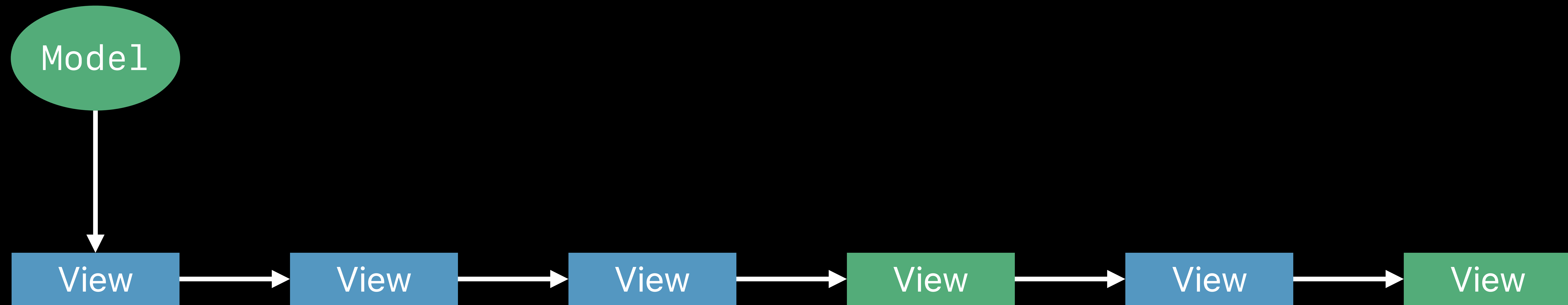
      Text("\(player.currentTime, formatter: playheadTimeFormatter)")
    }
  }
}
```



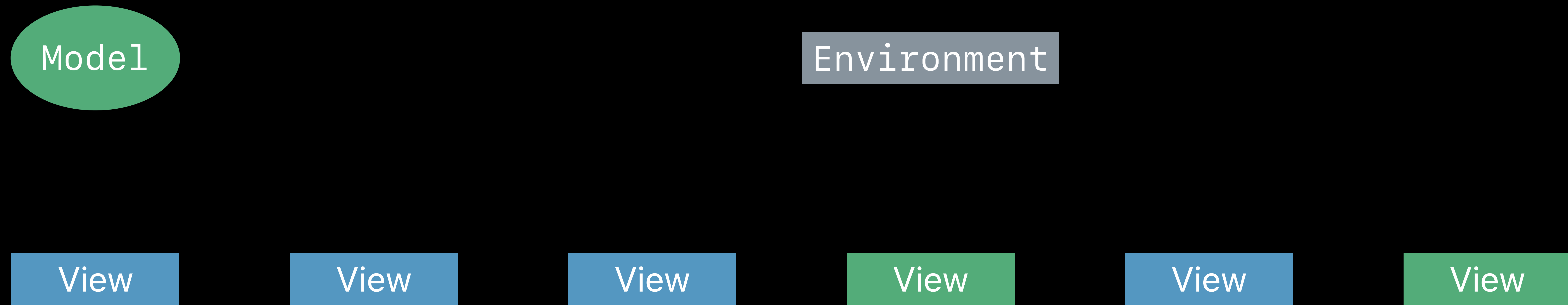
@ObjectBinding



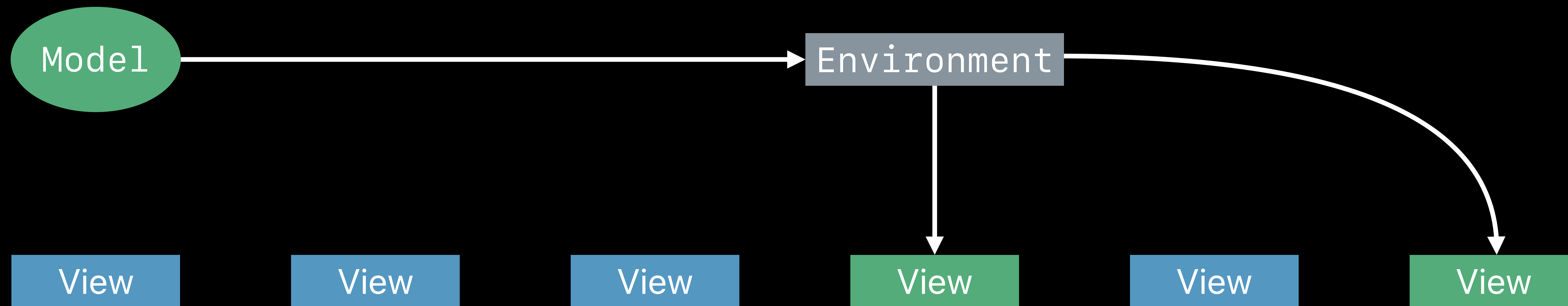
@ObjectBinding



@EnvironmentObject



@EnvironmentObject



Environment

Data applicable to an entire hierarchy

Convenience for indirection

Accent color, right-to-left, and more

Sources of Truth

@State

View-local
Value

Framework Managed

BindableObject

External
Reference

Developer Managed

Building Reusable Components

Read-only: Swift property, Environment

Read-write: `@Binding`

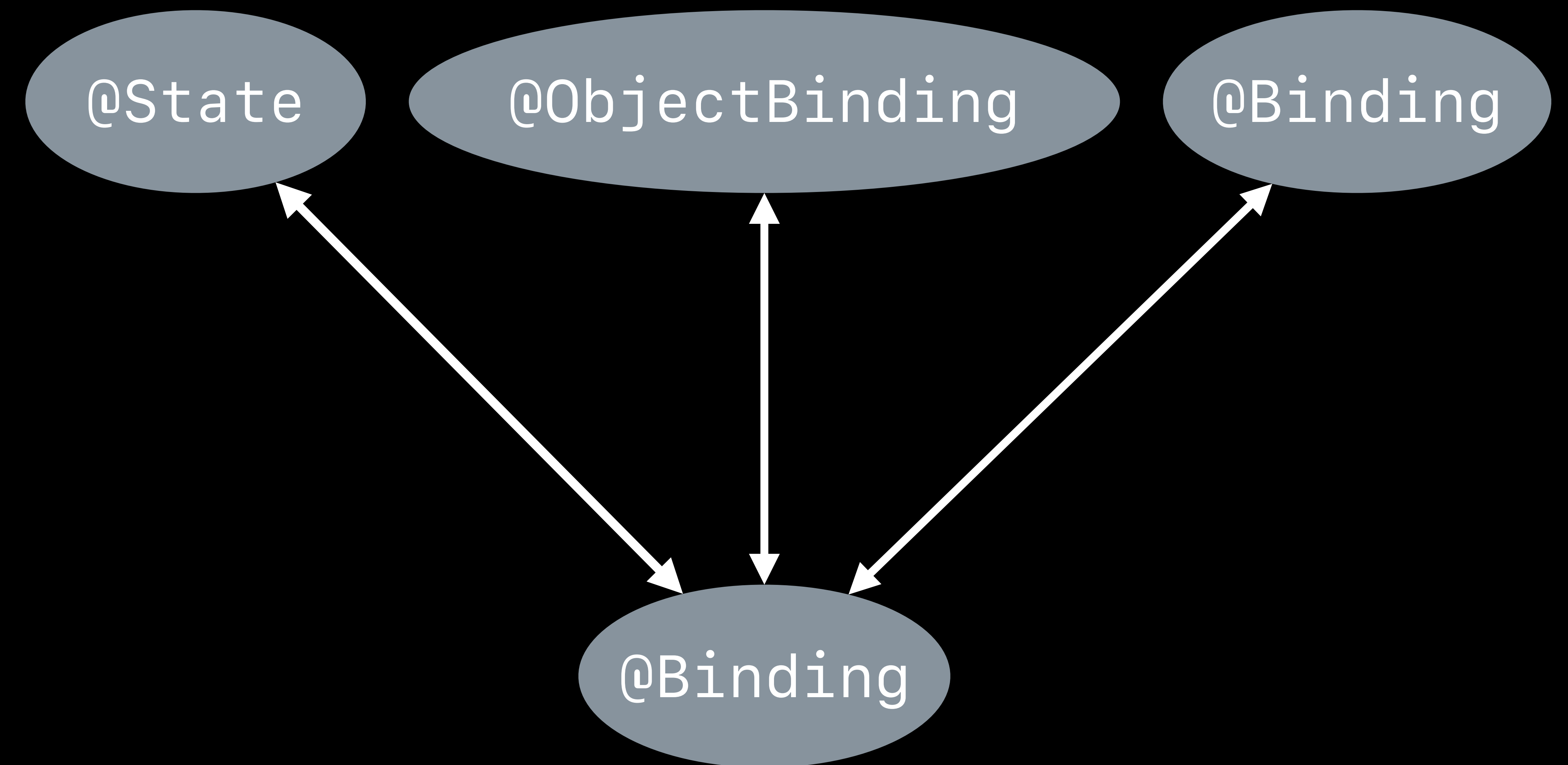
Prefer immutable access

@Binding

First class reference to data

Great for reusability

Use \$ to derive from source



Using State Effectively

Limit use if possible

Use derived `Binding` or value

Prefer `BindableObject` for persistence

Example: Button highlighting

Next Steps

Minimize sources of truth

Understand your data

Build a great app!

More Information

developer.apple.com/wwdc19/226

SwiftUI Essentials

WWDC 2019

Integrating SwiftUI

Thursday, 3:00

Building Custom Views with SwiftUI

Friday, 9:00

 WWDC19