SwiftUI for Production

try! Swift 2019 | Lea Marolt Sonnenschein | @hellosunschein

🥰 Like SwiftUI?

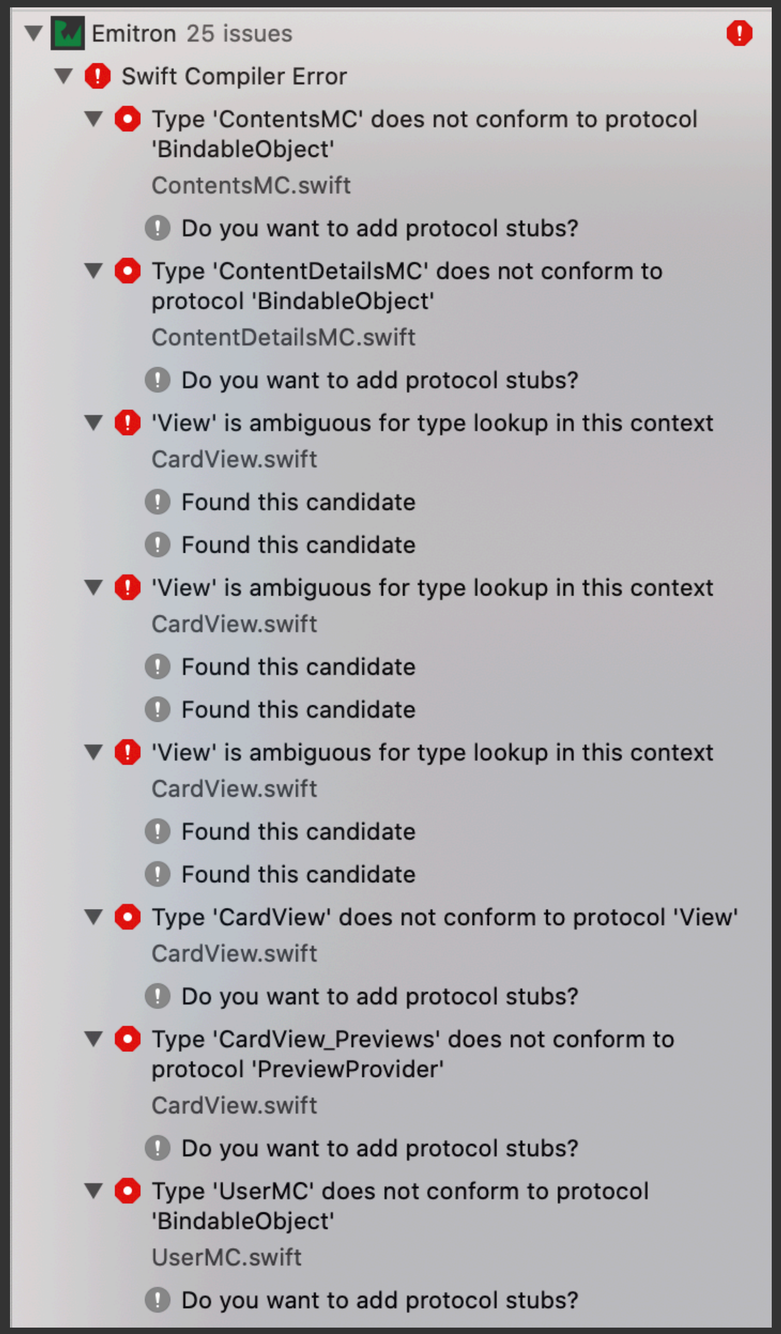
🛠 Used SwiftUI?

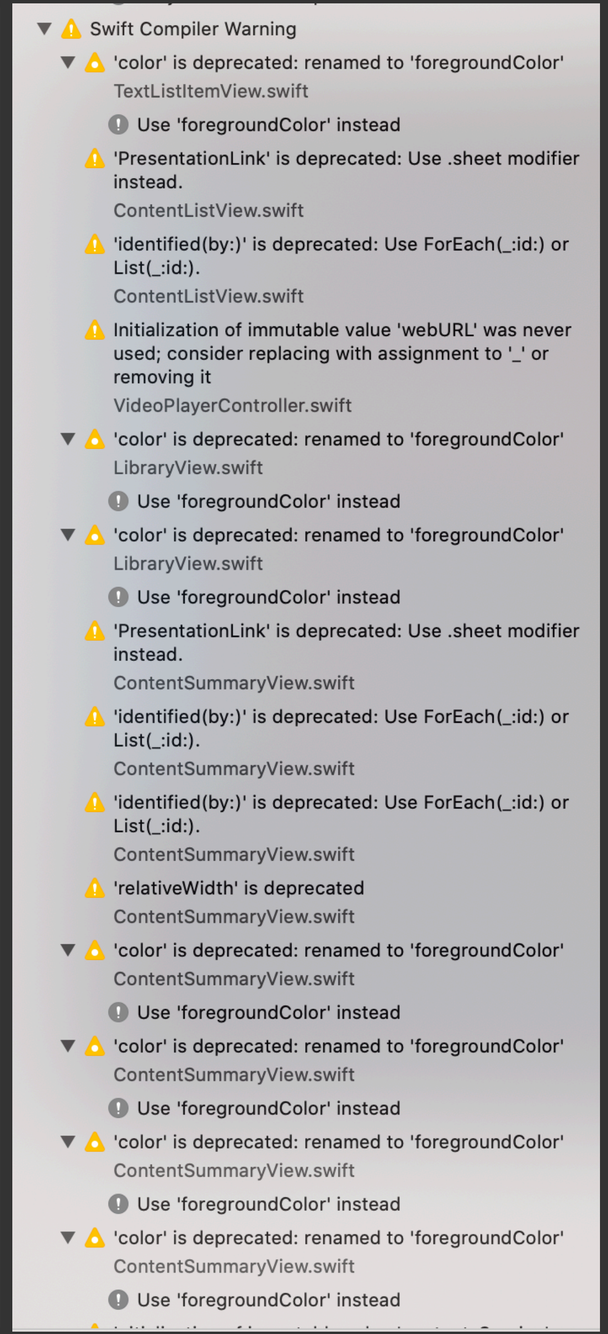
🥰

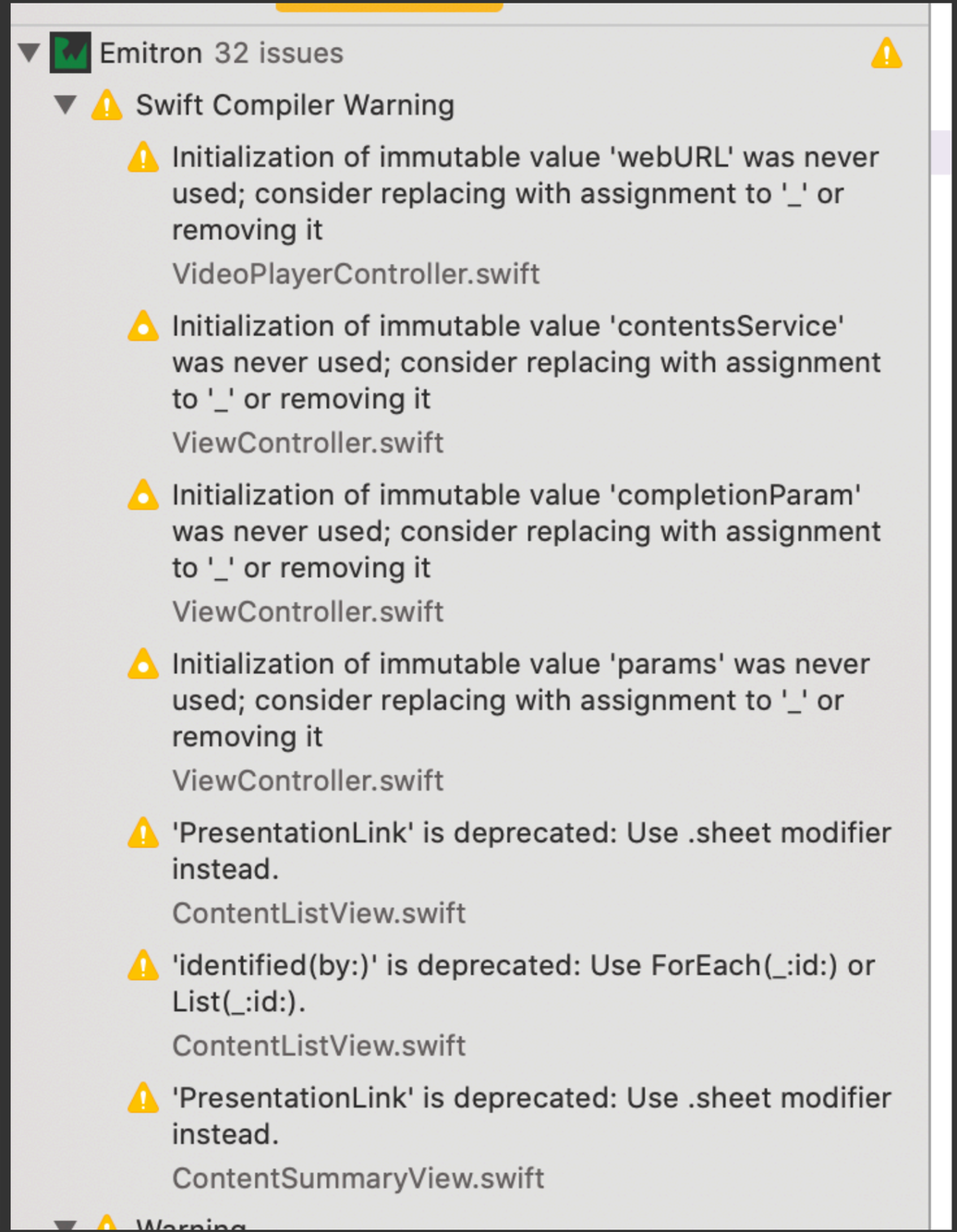
+

raywenderlich.com

SwiftUI

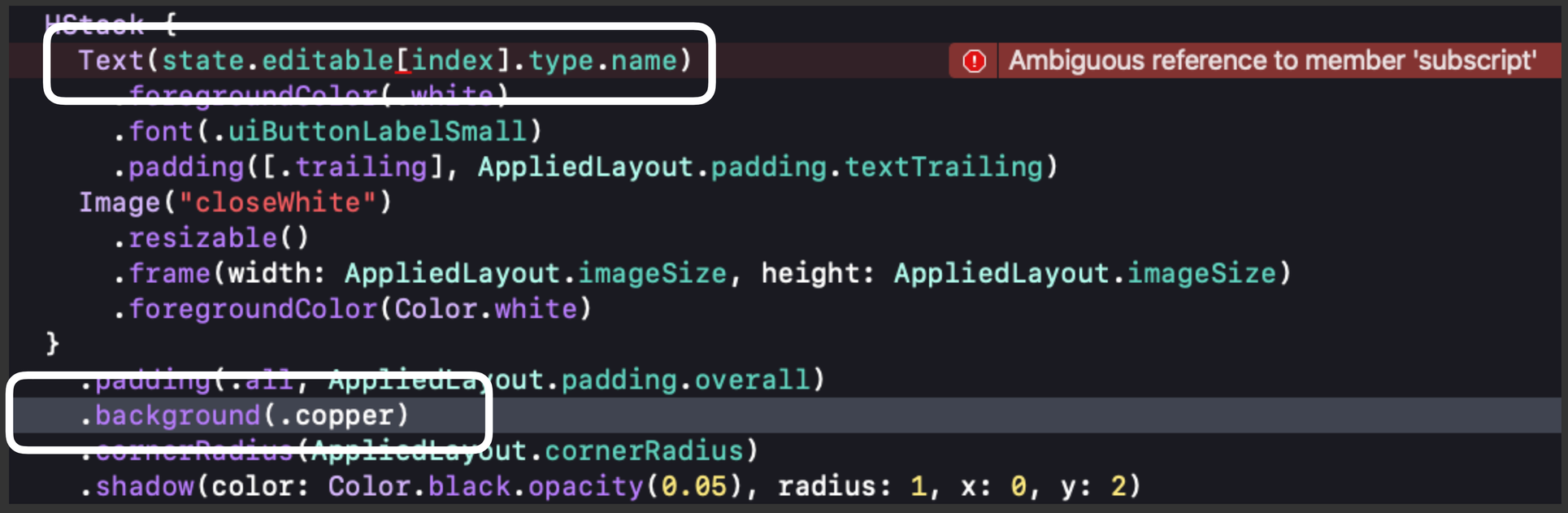

Emitron 25 issues
Swift Compiler Error
Type 'ContentsMC' does not conform to protocol 'BindableObject'
ContentsMC.swift
Do you want to add protocol stubs?
Type 'ContentDetailsMC' does not conform to protocol 'BindableObject'
ContentDetailsMC.swift
Do you want to add protocol stubs?
'View' is ambiguous for type lookup in this context
CardView.swift
Found this candidate
Found this candidate
'View' is ambiguous for type lookup in this context
CardView.swift
Found this candidate
Found this candidate
'View' is ambiguous for type lookup in this context
CardView.swift
Found this candidate
Found this candidate
Type 'CardView' does not conform to protocol 'View'
CardView.swift
Do you want to add protocol stubs?
Type 'CardView_Previews' does not conform to protocol 'PreviewProvider'
CardView.swift
Do you want to add protocol stubs?
Type 'UserMC' does not conform to protocol 'BindableObject'
UserMC.swift
Do you want to add protocol stubs?


Swift Compiler Warning
'color' is deprecated: renamed to 'foregroundColor'
TextListItemView.swift
Use 'foregroundColor' instead
'PresentationLink' is deprecated: Use .sheet modifier instead.
ContentListView.swift
'identified(by:)' is deprecated: Use ForEach(_:id:) or List(_:id:).
ContentListView.swift
Initialization of immutable value 'webURL' was never used; consider replacing with assignment to '_' or removing it
VideoPlayerController.swift
'color' is deprecated: renamed to 'foregroundColor'
LibraryView.swift
Use 'foregroundColor' instead
'color' is deprecated: renamed to 'foregroundColor'
LibraryView.swift
Use 'foregroundColor' instead
'PresentationLink' is deprecated: Use .sheet modifier instead.
ContentSummaryView.swift
'identified(by:)' is deprecated: Use ForEach(_:id:) or List(_:id:).
ContentSummaryView.swift
'identified(by:)' is deprecated: Use ForEach(_:id:) or List(_:id:).
ContentSummaryView.swift
'relativeWidth' is deprecated
ContentSummaryView.swift
'color' is deprecated: renamed to 'foregroundColor'
ContentSummaryView.swift
Use 'foregroundColor' instead
'color' is deprecated: renamed to 'foregroundColor'
ContentSummaryView.swift
Use 'foregroundColor' instead
'color' is deprecated: renamed to 'foregroundColor'
ContentSummaryView.swift
Use 'foregroundColor' instead
'color' is deprecated: renamed to 'foregroundColor'
ContentSummaryView.swift
Use 'foregroundColor' instead


Emitron 32 issues
Swift Compiler Warning
Initialization of immutable value 'webURL' was never used; consider replacing with assignment to '_' or removing it
VideoPlayerController.swift
Initialization of immutable value 'contentsService' was never used; consider replacing with assignment to '_' or removing it
ViewController.swift
Initialization of immutable value 'completionParam' was never used; consider replacing with assignment to '_' or removing it
ViewController.swift
Initialization of immutable value 'params' was never used; consider replacing with assignment to '_' or removing it
ViewController.swift
'PresentationLink' is deprecated: Use .sheet modifier instead.
ContentListView.swift
'identified(by:)' is deprecated: Use ForEach(_:id:) or List(_:id:).
ContentListView.swift
'PresentationLink' is deprecated: Use .sheet modifier instead.
ContentSummaryView.swift

SwiftUI/

Where the compiler thinks the problem is.



Where the problem actually is.

SwiftUI/

I have a ContentView written in swiftUI as simple as below. "" var body: some View {

```
    NavigationView {
        List {
            Section {
                PresentationLink(destination: Text("new Profile")) {
                    Text("new Profile")
                }
            }
        }
    }
```

""

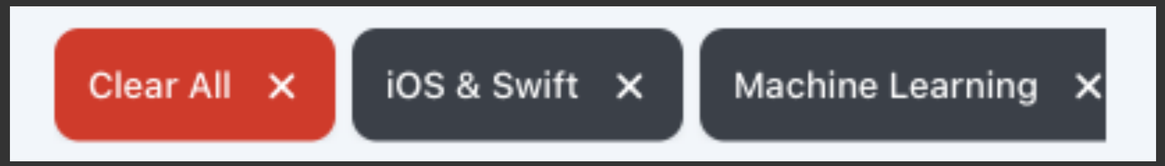
everything is good first time I tap on new profile but when I close the modal and try to tap again, it does not work. is it a bug or a feature?

How to !@#$ change the BG color of a table view?!

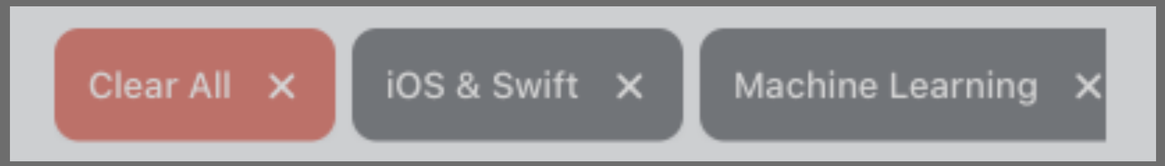
```
List {
  ForEach(contents, id: \.id) { partialContent in
    NavigationLink(destination: ContentListingView() {
      CardView(model: CardViewModel.transform(partialContent))
        .listRowBackground(self.bgColor)
        .background(self.bgColor)
    }
    .listRowBackground(self.bgColor)
    .background(self.bgColor)
  }
  .listRowBackground(self.bgColor)
  .background(self.bgColor)
```

???

```swift
private func filtersView() -> some View {
  ScrollView(.horizontal, showsIndicators: false) {
    HStack(alignment: .top, spacing: .filterSpacing) {

      ForEach(filters.appliedFilters, id: \.self) { filter in
        AppliedFilterView(filter: filter, type: .default) {
          self.contentsMC.updateFilters(newFilters: self.filters)
        }
      }
    }
  }
  .padding([.top], .filtersPaddingTop)
}
```



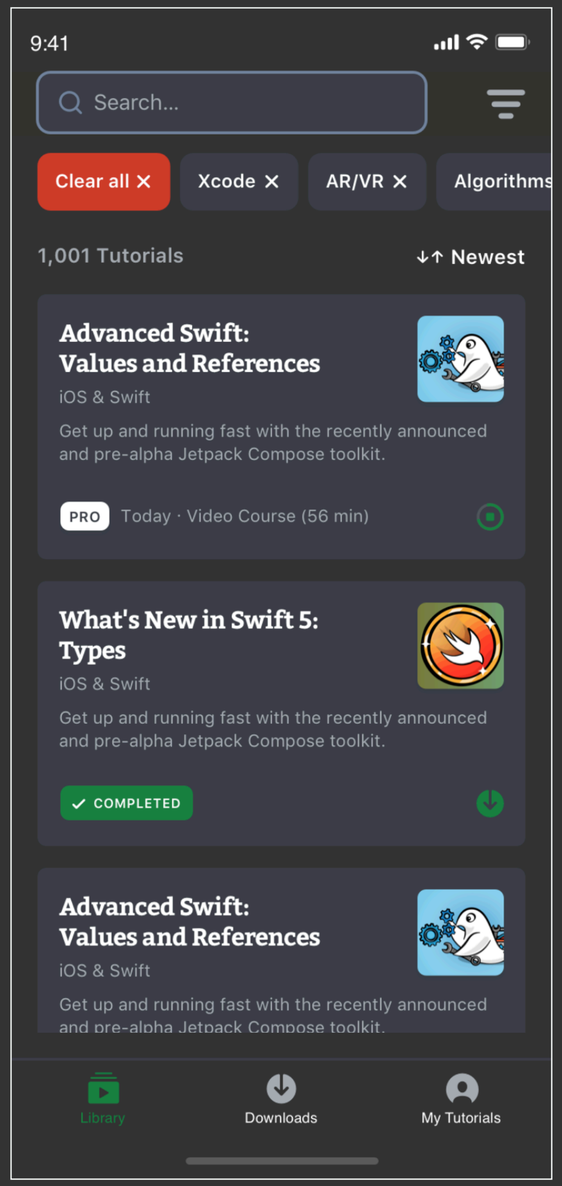
```
private func filtersView() -> some View {
  ScrollView(.horizontal, showsIndicators: false) {
    HStack(alignment: .top, spacing: .filterSpacing) {

      ForEach(filters.appliedFilters, id: \.self) { filter in
        AppliedFilterView(filter: filter, type: .default) {
          self.contentsMC.updateFilters(newFilters: self.filters)
        }
      }
    }
  }
  .padding([.top], .filtersPaddingTop)
}
```



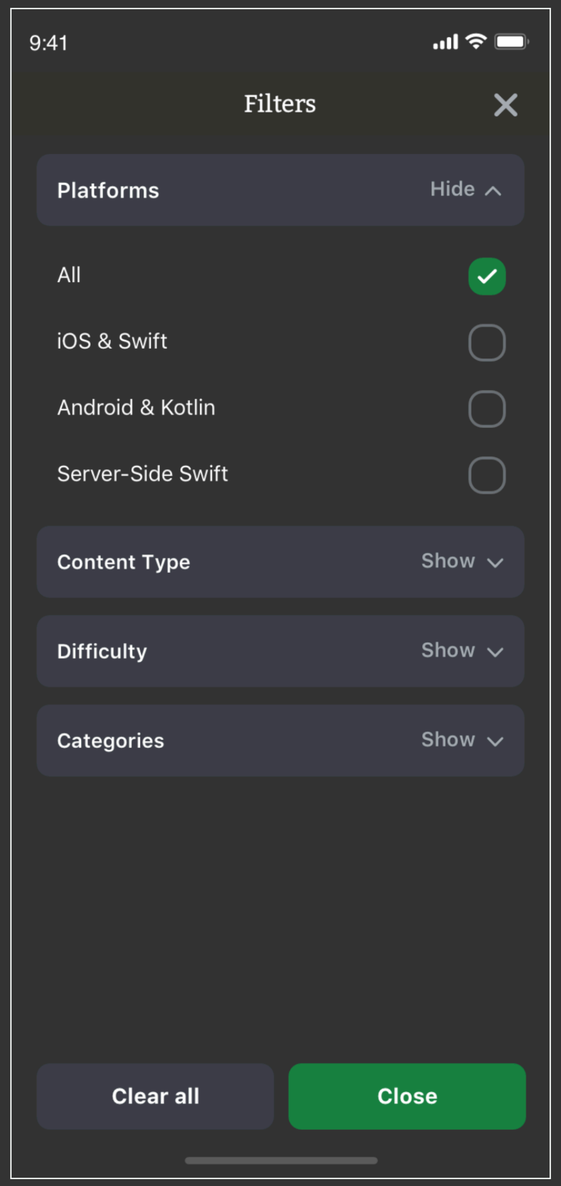
SwiftUI

SwiftUI/

Content List

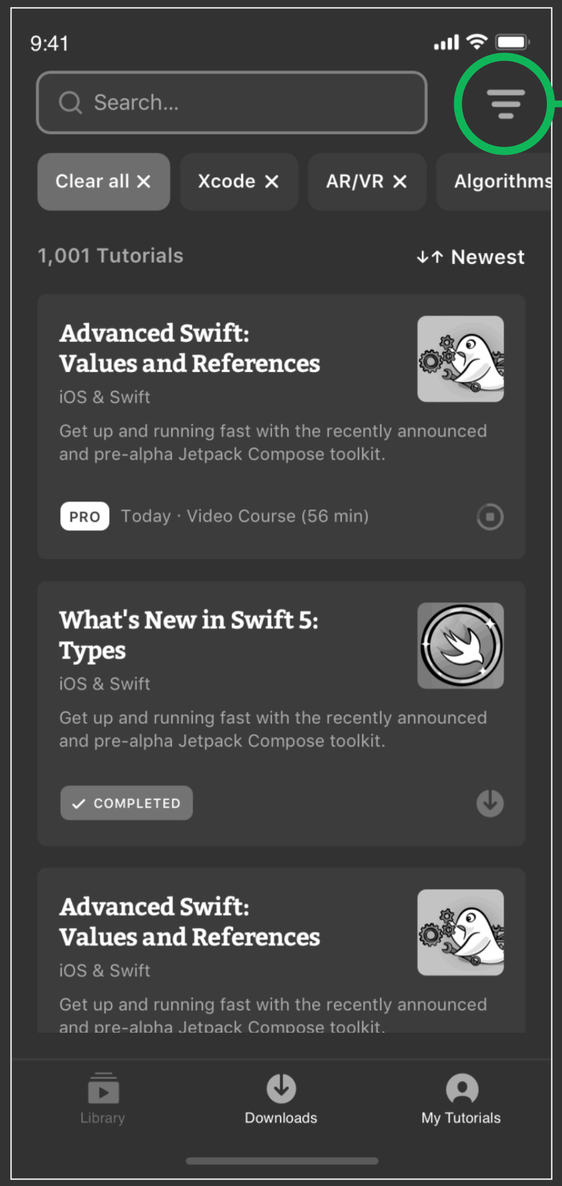


Filters

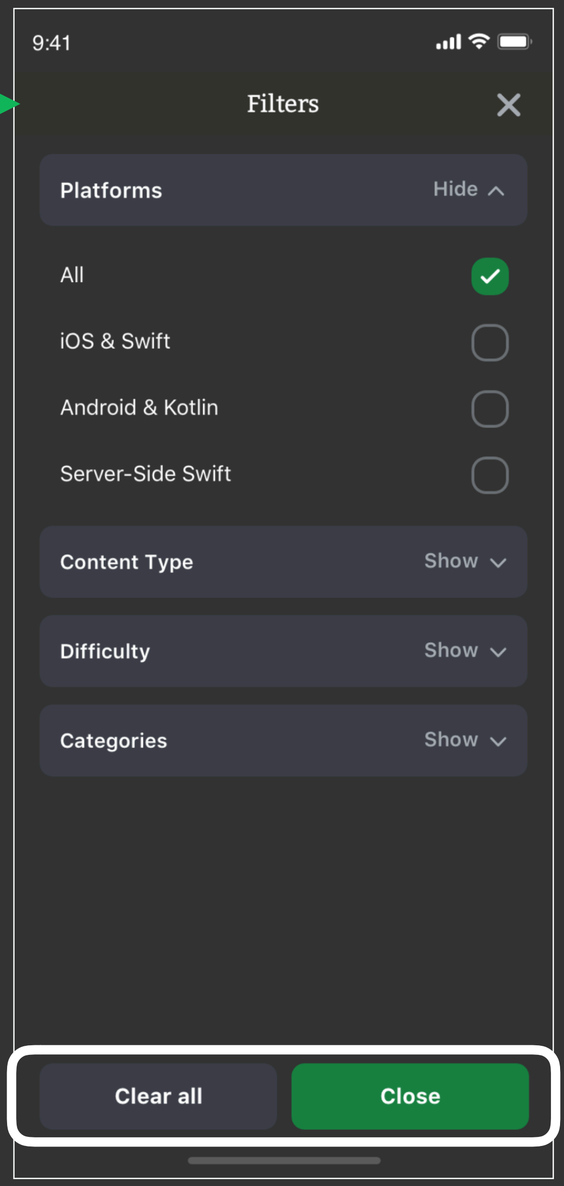


SwiftUI/

Content List

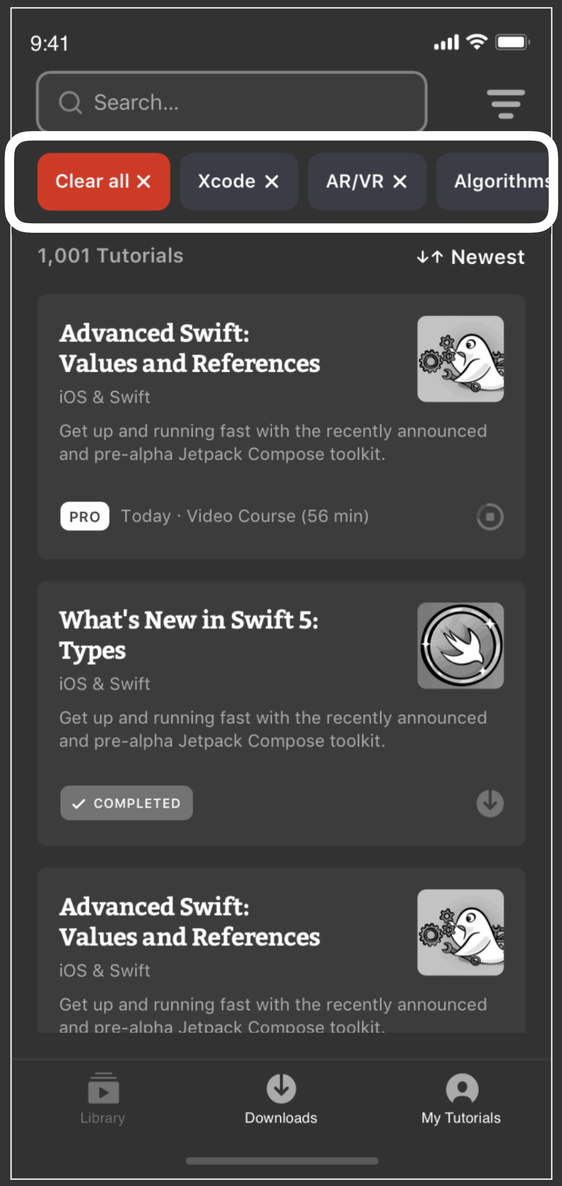


Filters

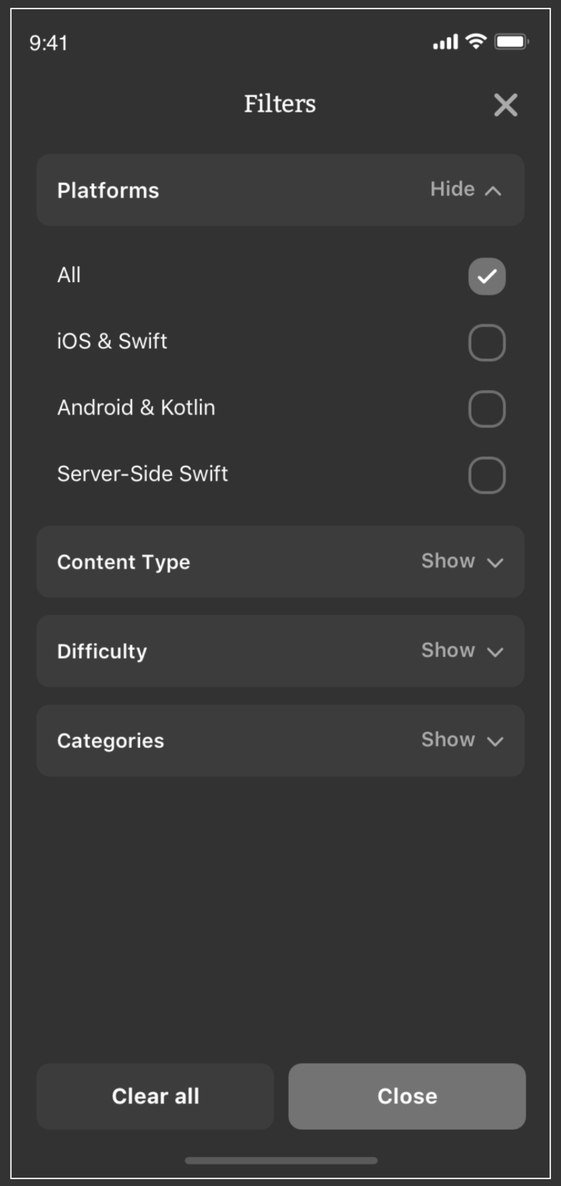


SwiftUI/

Content List



Filters



SwiftUI

SwiftUI/

Declarative UI framework.

“View is a **function of state**, not a sequence of events.”

SwiftUI/

On

Off

SwiftUI/

1. Create

```
class CheckmarkBox: UIView {

  private var isOn: Bool

  init(isOn: Bool) {
    self.isOn = isOn
    super.init(frame: .zero)
    setupViews()
  }
}
```

SwiftUI/

1. Create

```
class CheckmarkBox: UIView {

  private var isOn: Bool

  init(isOn: Bool) {
    self.isOn = isOn
    super.init(frame: .zero)
    setupViews()
  }
}
```

2. Configure

On

Off

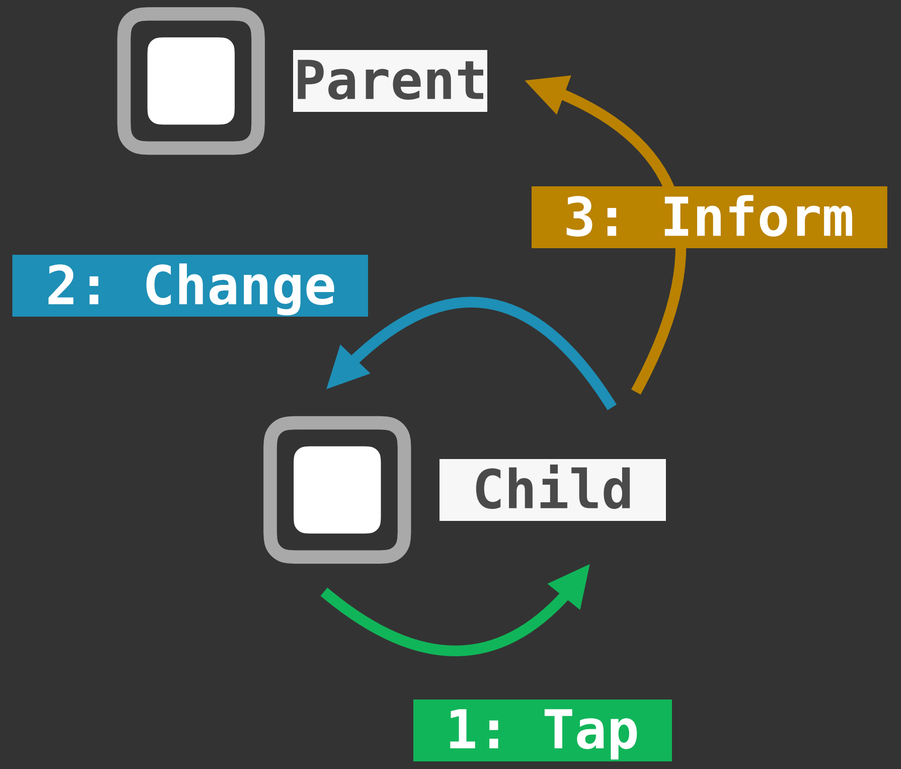
SwiftUI/

1. Create

```
class CheckmarkBox: UIView {

  private var isOn: Bool

  init(isOn: Bool) {
    self.isOn = isOn
    super.init(frame: .zero)
    setupViews()
  }
}
```

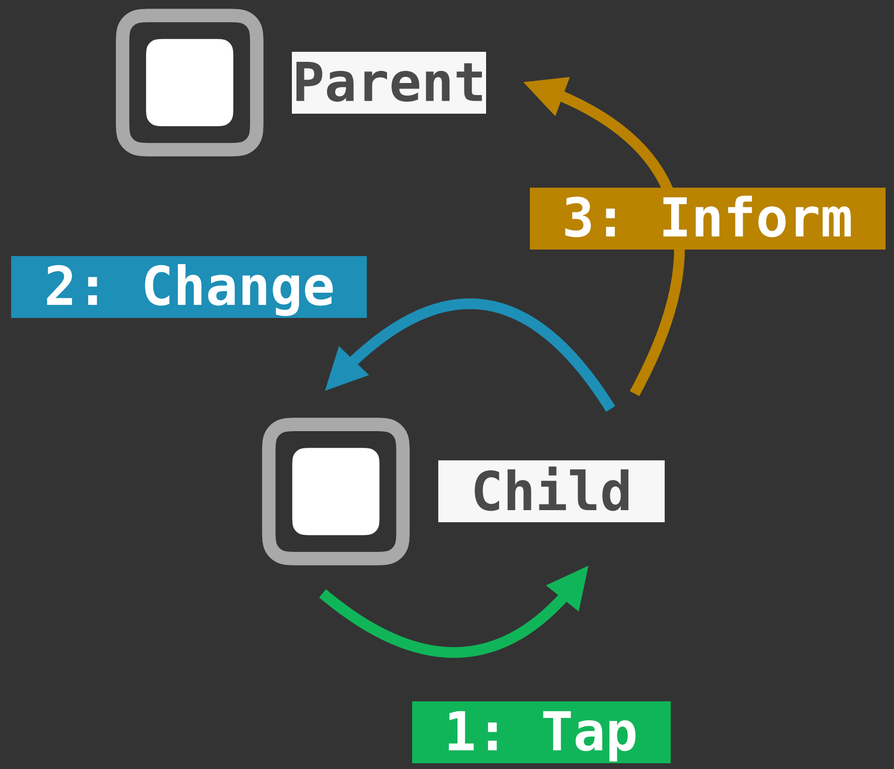
2. Configure

3. Respond



SwiftUI/

UIKit



😩

Multidirectional
data flow.

SwiftUI/

1. Create

```
struct CheckmarkView: View {
  @State var isOn: Bool = false

  var body: some View {

    Button(action: {
      self.isOn.toggle()
    }) {
      if isOn { onView()
      } else { offView()
      }
    }
  }
}
```

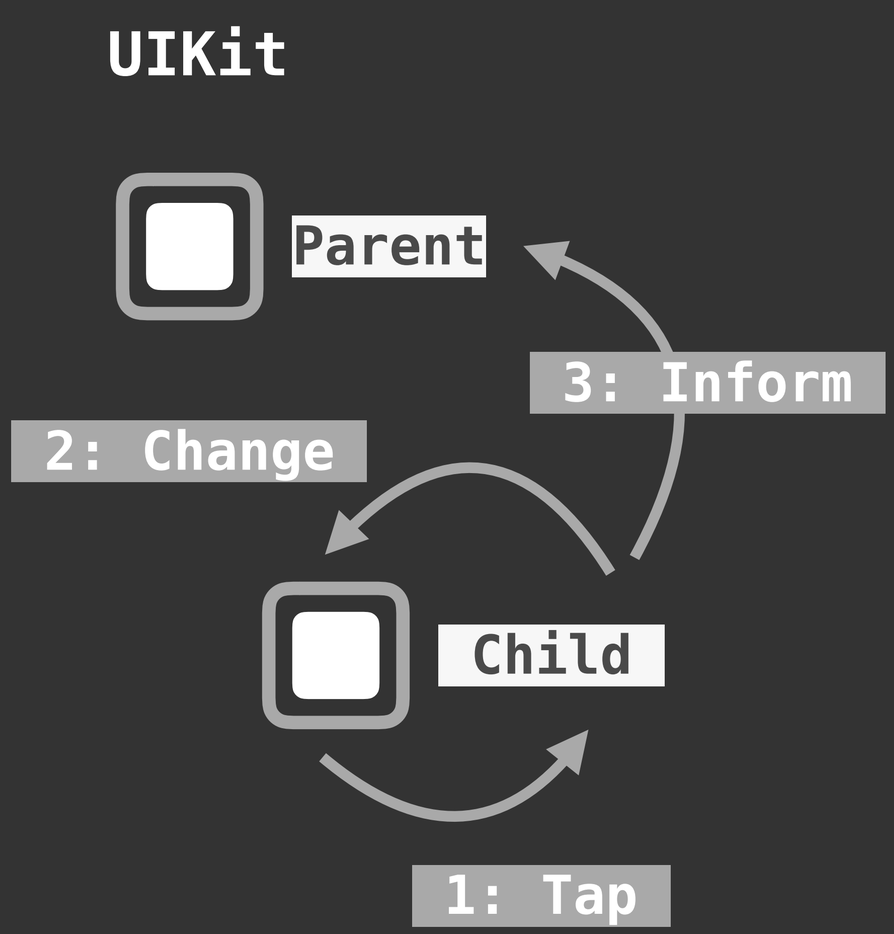
2. Configure

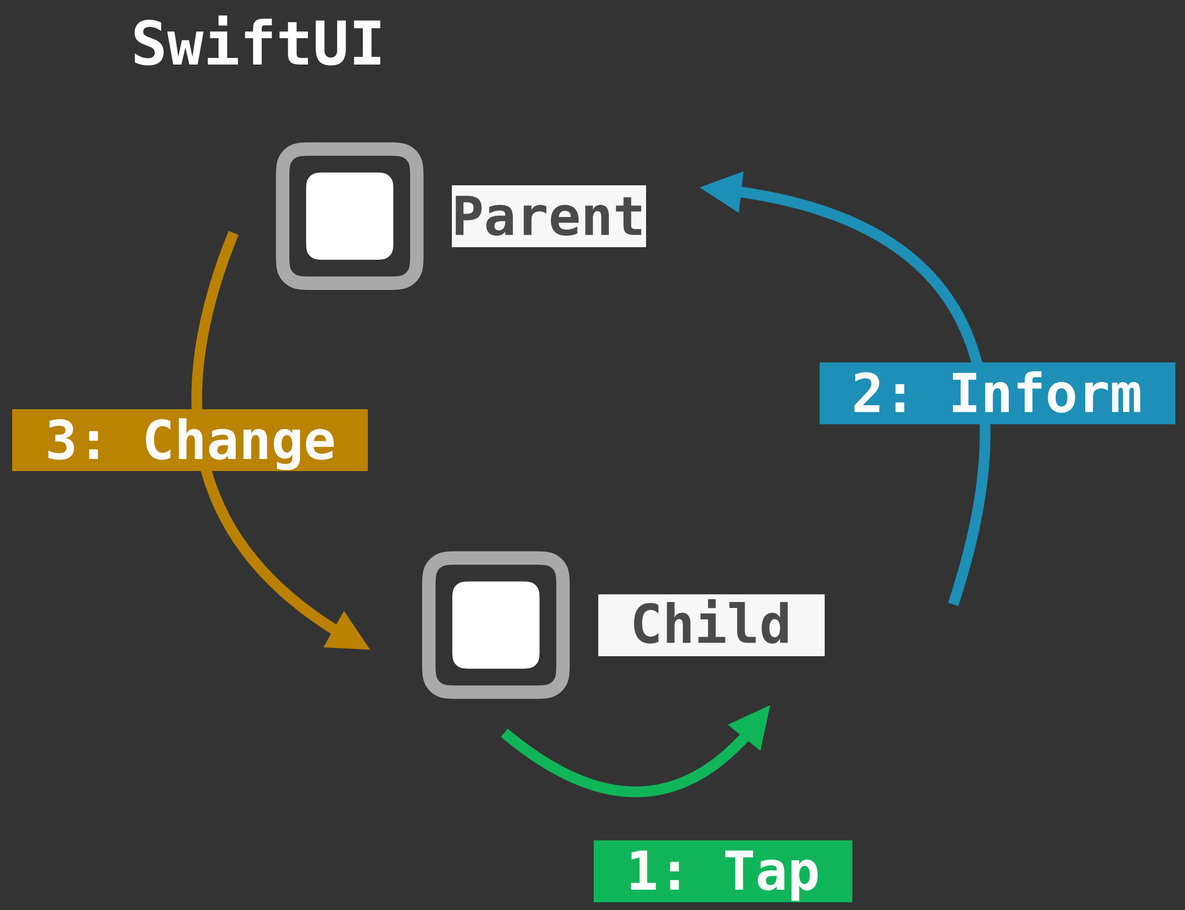
On

Off

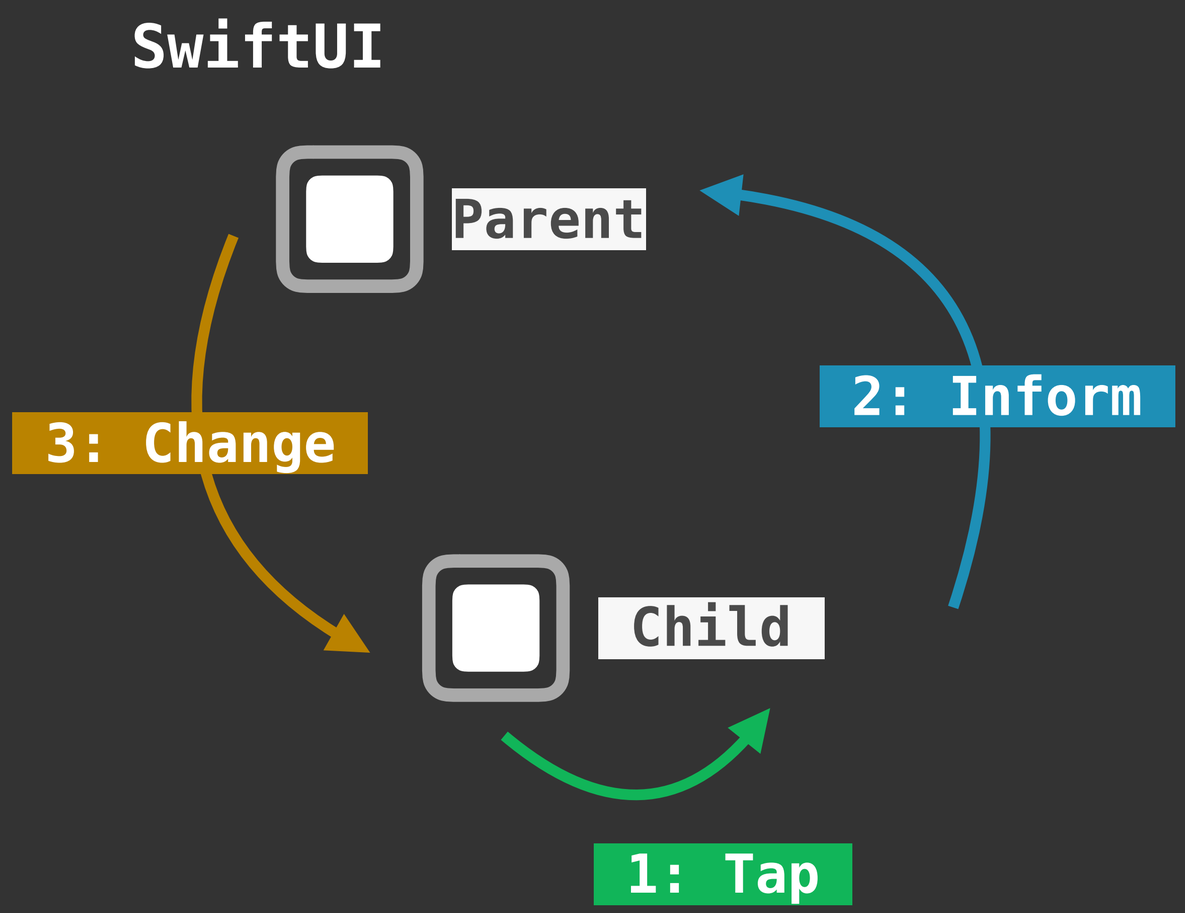
3. Respond

SwiftUI/


UIKit
Parent
3: Inform
2: Change
Child
1: Tap


SwiftUI
Parent
2: Inform
3: Change
Child
1: Tap

Unidirectional data flow.



Data Flow~

Data Flow/

mutable

immutable

a View is a struct

is a value type

is immutable

Data Flow/

```
@State

@ObservedObject

@EnvironmentObject
```

Data Flow/

1. `@State`, to manage own state

Data Flow/

Data Flow/

onView()

```
var body: some View {

    if isOn { onView() }

  }
}
```

Data Flow/

onView()

offView()

```
var body: some View {

    if isOn { onView() }
    else { offView() }
  }
}
```

Data Flow/

onView()

offView()

```
var body: some View {

  Button(action: {
    //toggle between on/off
  }) {
    if isOn { onView() }
    else { offView() }
  }
}
```

Data Flow/

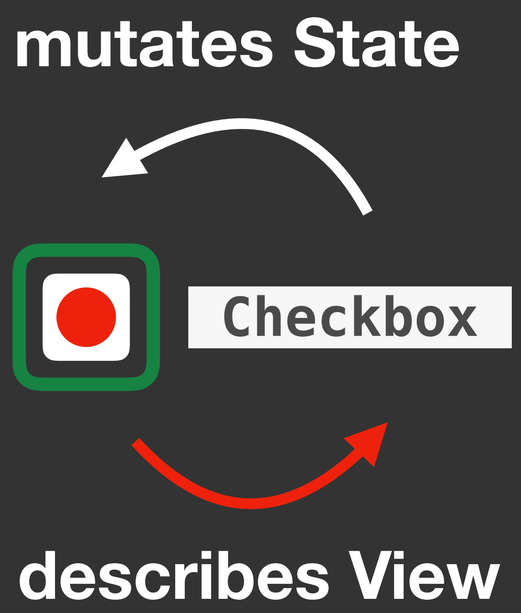
onView()

offView()

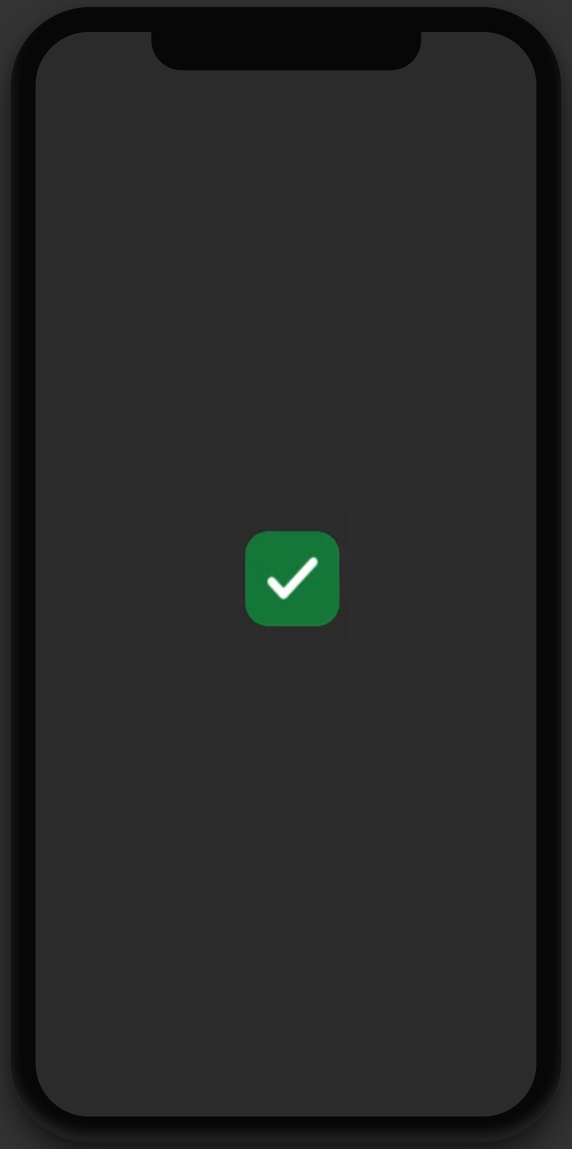
```
@State var isOn: Bool = false
var body: some View {

  Button(action: {
    self.isOn.toggle()
  }) {
    if isOn { onView() }
    else { offView() }
  }
}
```

Data Flow/

- = state; source of truth

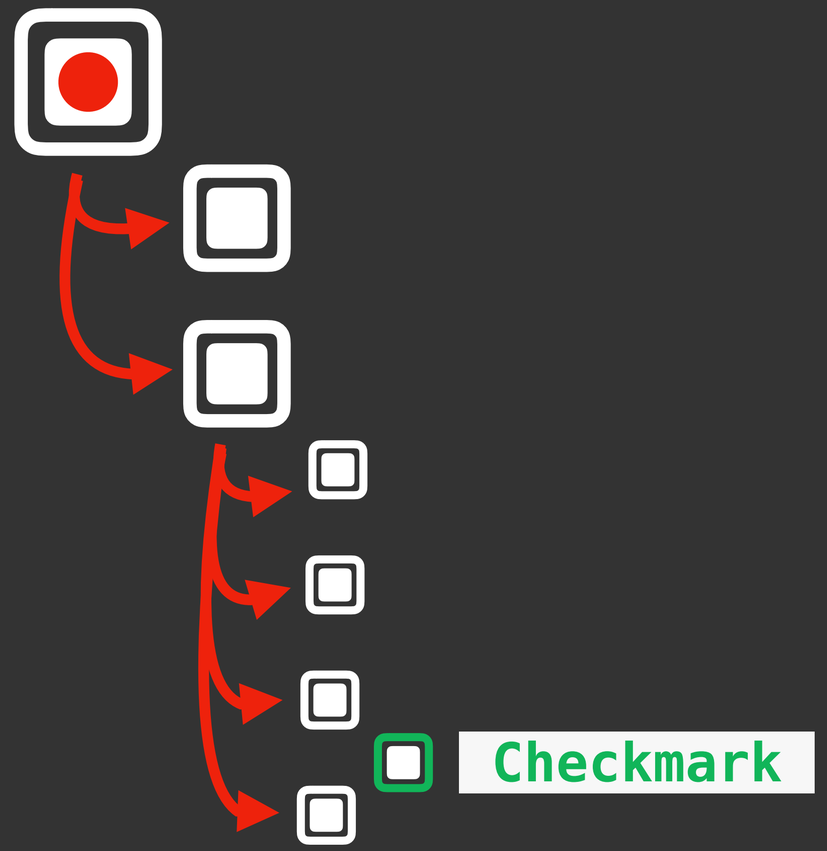


Data Flow/



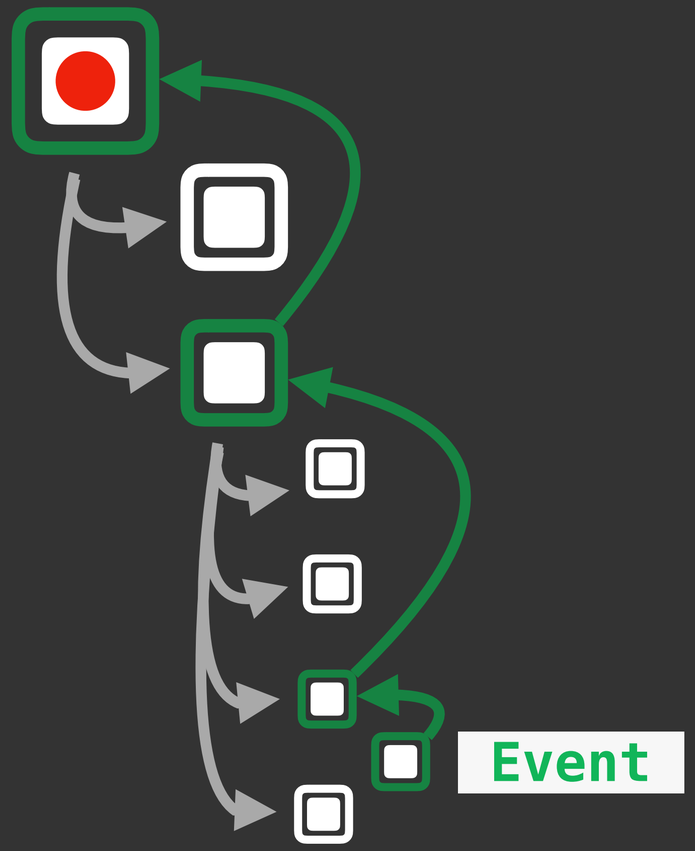
Data Flow/

- = state owner; source of truth



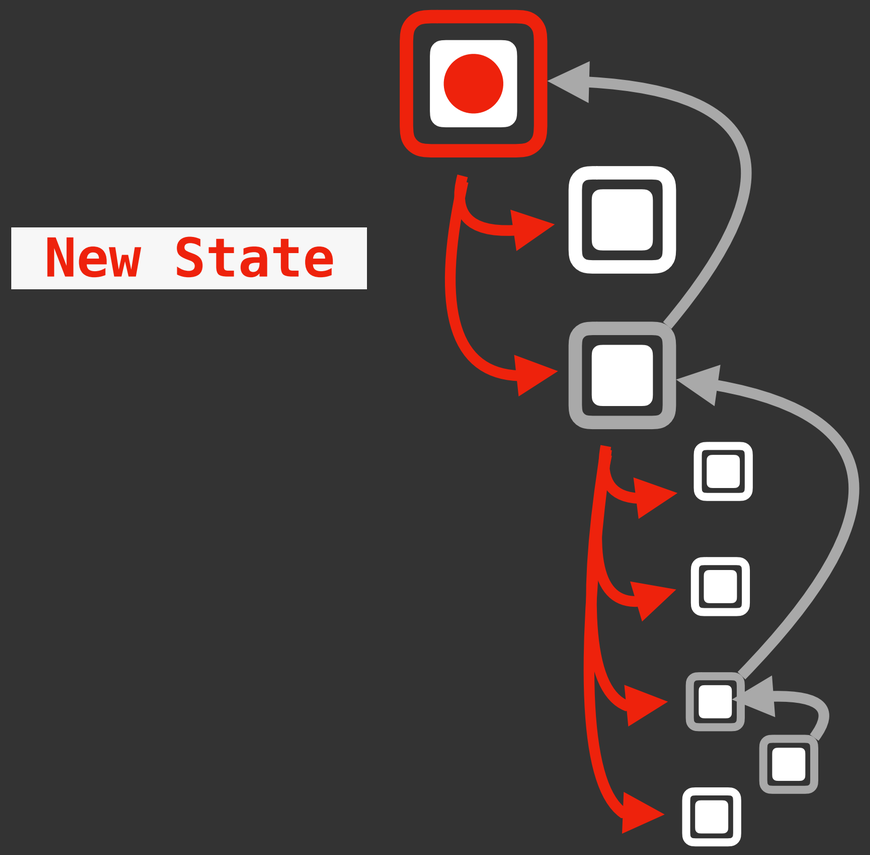
Data Flow/

- = state owner; source of truth



Data Flow/

- = state; source of truth



Data Flow/

1. `@State`, to manage your own state

2. `@State` <> `@Binding`, to communicate between views

Data Flow/

1. `@State`, to manage your own state

2. `@State` <> `@Binding`, to communicate between views

```
struct Room: View {
  var isLampOn: Bool

  var body: some View {
    Lamp(isOn: isLampOn)
  }
}
```

```
struct Lamp: View {
  var isOn: Bool
}
```

Data Flow/

1. `@State`, to manage your own state

2. `@State` <> `@Binding`, to communicate between views

```
struct Room: View {
  @State var isLampOn: Bool

  var body: some View {
    Lamp(isOn: isLampOn)
  }
}
```

```
struct Lamp: View {
  var isOn: Bool
}
```

Data Flow/

1. `@State`, to manage your own state

2. `@State` <> `@Binding`, to communicate between views

```
struct Room: View {
  @State var isLampOn: Bool

  var body: some View {
    Lamp(isOn: isLampOn)
  }
}
```

```
struct Lamp: View {
  @Binding var isOn: Bool
}
```

Data Flow/

1. `@State`, to manage your own state

2. `@State` <> `@Binding`, to communicate between views

```
struct Room: View {
  @State var isLampOn: Bool

  var body: some View {
    Lamp(isOn: $isLampOn)
  }
}
```

```
struct Lamp: View {
  @Binding var isOn: Bool
}
```

Data Flow/

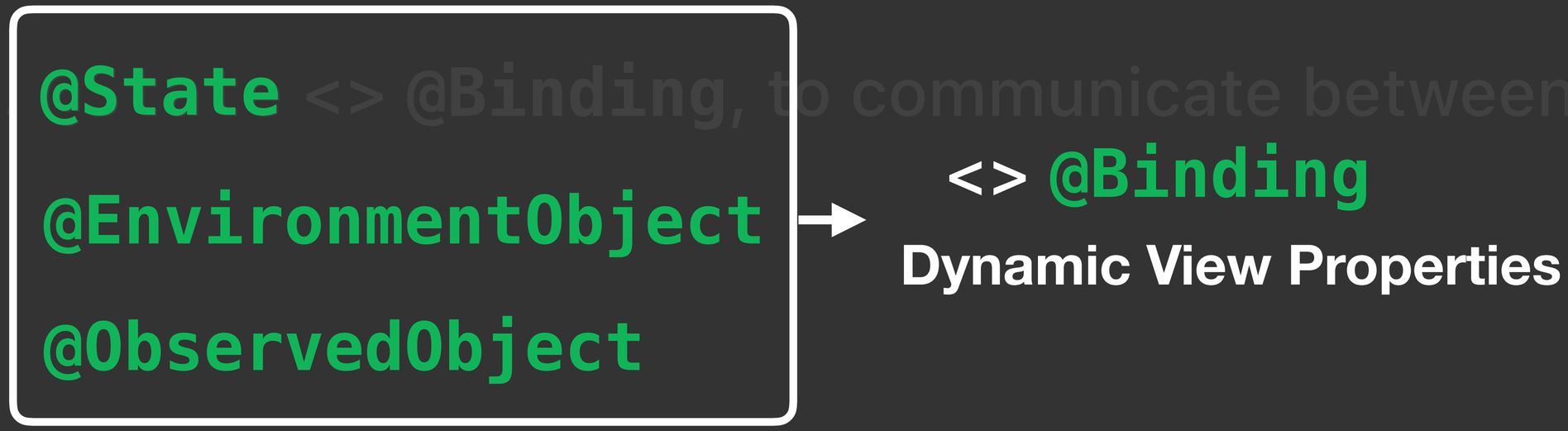
1. `@State`, to manage your own state
2. `@State` <> `@Binding`, to communicate between views

   `@EnvironmentObject`

   `@ObservedObject`

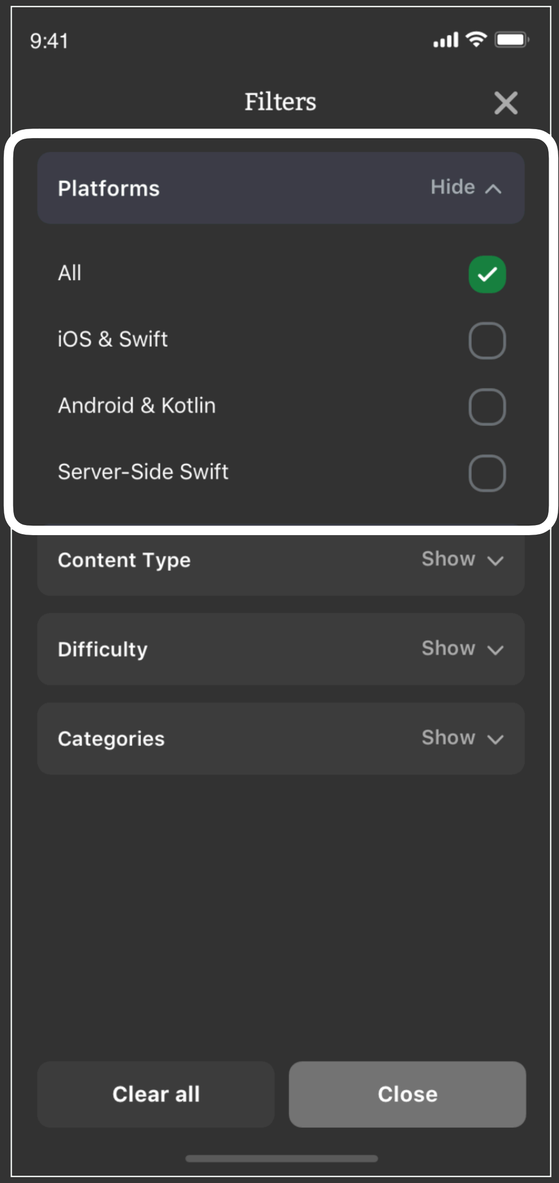
Data Flow/

1. `@State`, to manage your own state
2. `@State` <> `@Binding`, to communicate between views



Data Flow/

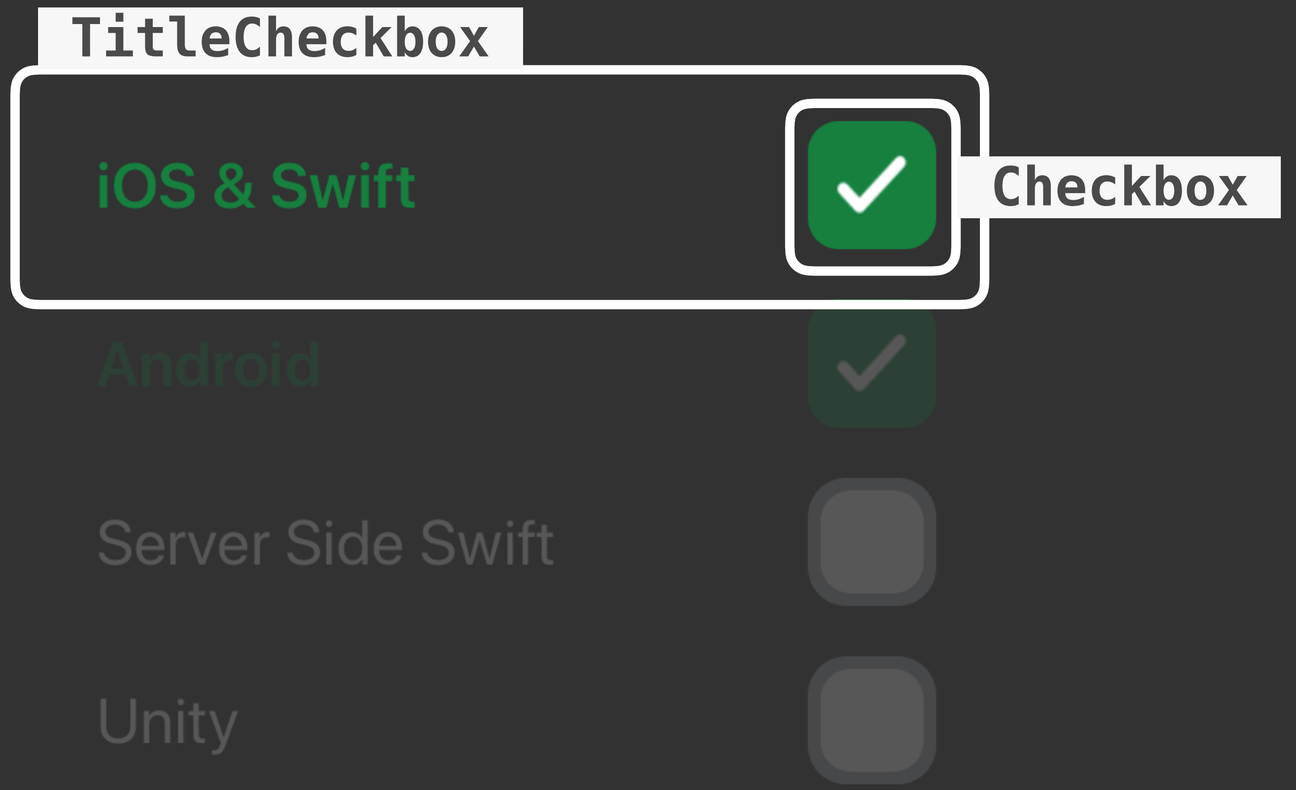
Filters



Data Flow/

- [x] iOS & Swift
- [x] Android
- [ ] Server Side Swift
- [ ] Unity

Data Flow/


TitleCheckbox
iOS & Swift
Checkbox
Android
Server Side Swift
Unity

Data Flow/


```
struct TitleCheckbox: View {
  @State var isOn: Bool
  var name: String
}
```

Data Flow/


```
struct TitleCheckbox: View {
  @State var isOn: Bool
  var name: String
}
```

Data Flow/


```
struct TitleCheckbox: View {
  @State var isOn: Bool
  var name: String
}
```

```
struct Checkbox: View {
  @Binding var isOn: Bool
}
```

Data Flow/

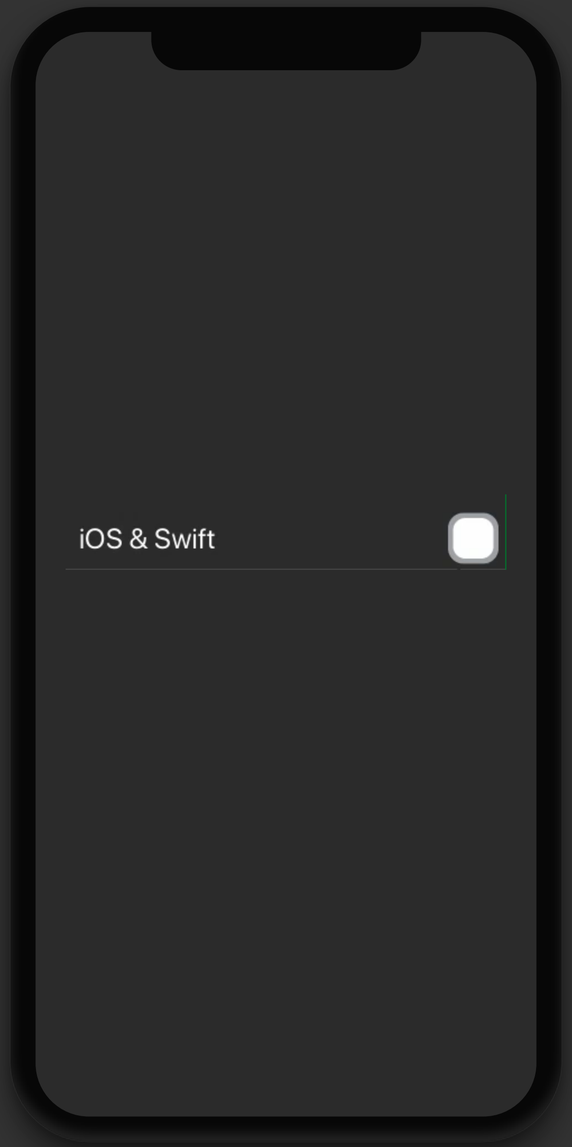

```
struct TitleCheckbox: View {
  @State var isOn: Bool
  var name: String

  var body: some View {
    CheckboxView(isOn: $isOn)
  }
}
```

```
struct Checkbox: View {
  @Binding var isOn: Bool
}
```

Data Flow/

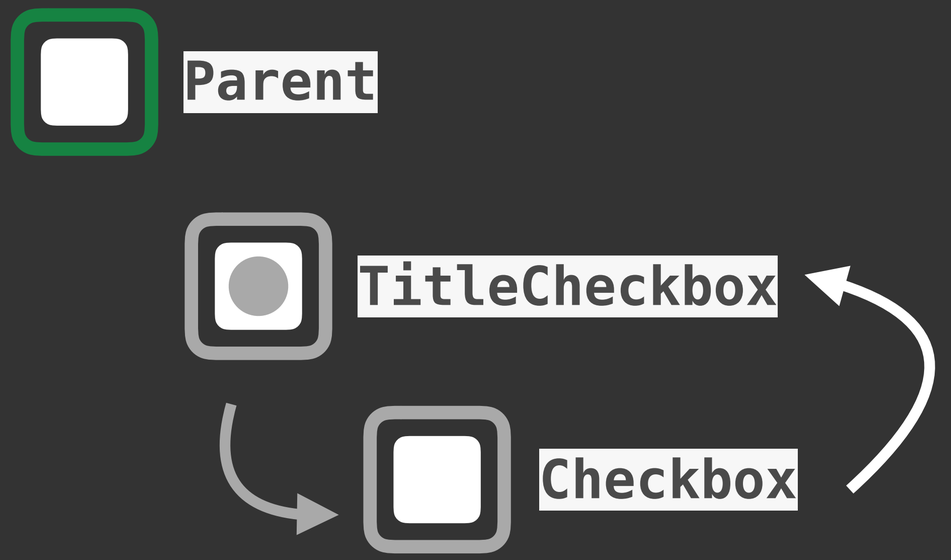
- = state; source of truth

Data Flow/



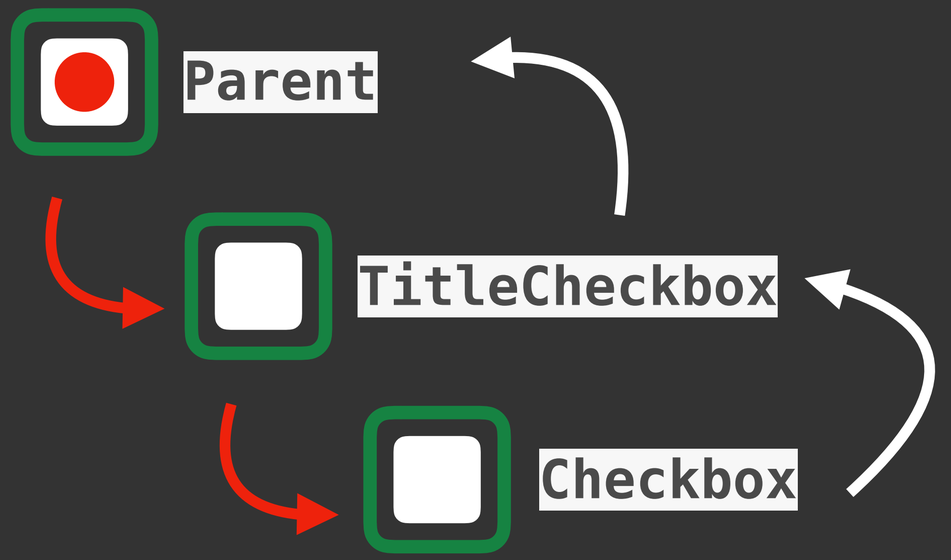
Data Flow/

- = state; source of truth



Data Flow/

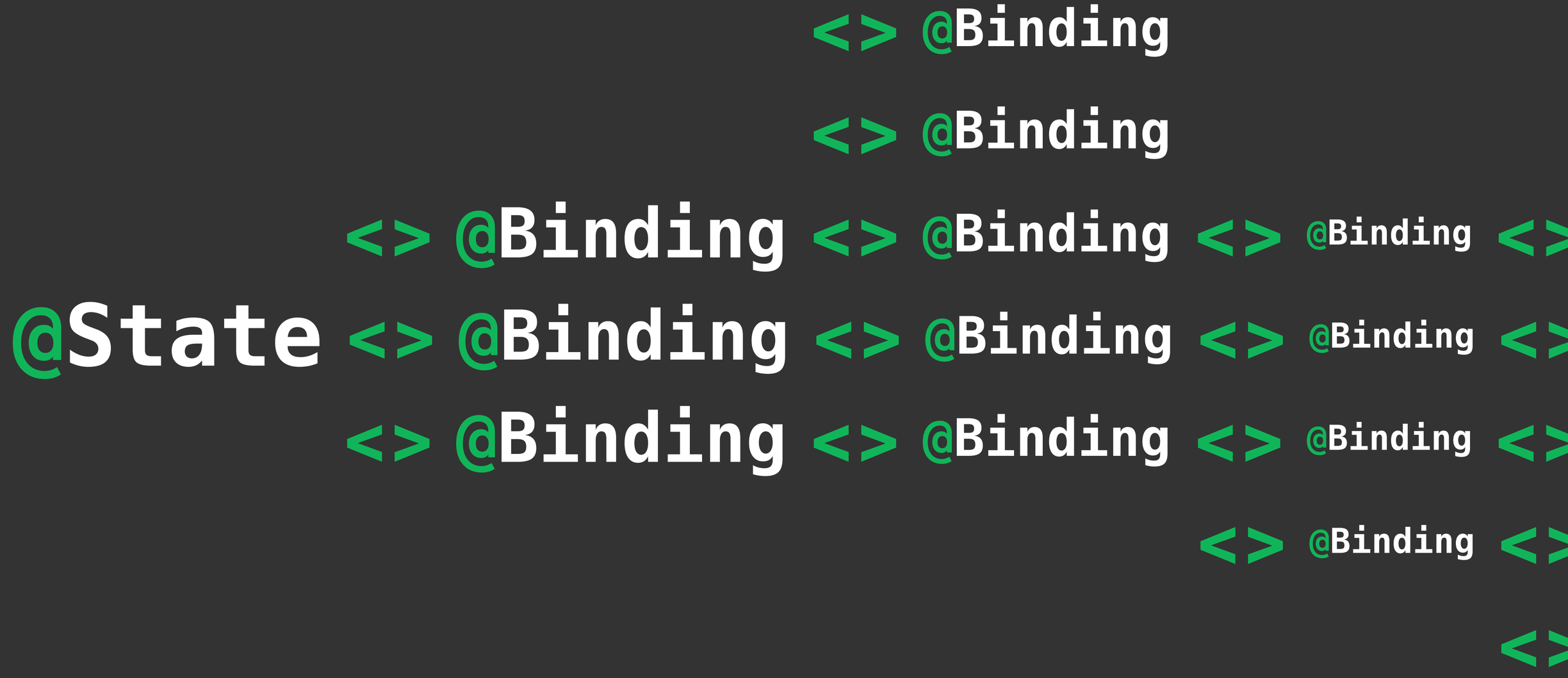
- = state; source of truth



Data Flow/

@State <> @Binding <> @Binding <> @Binding <>

Data Flow/


@State
<> @Binding
<> @Binding
<> @Binding
<> @Binding
<> @Binding
<> @Binding
<> @Binding
<> @Binding
<> @Binding
<> @Binding
<> @Binding
<>

Custom controls shouldn’t be using `@State`.

Data Flow Through SwiftUI

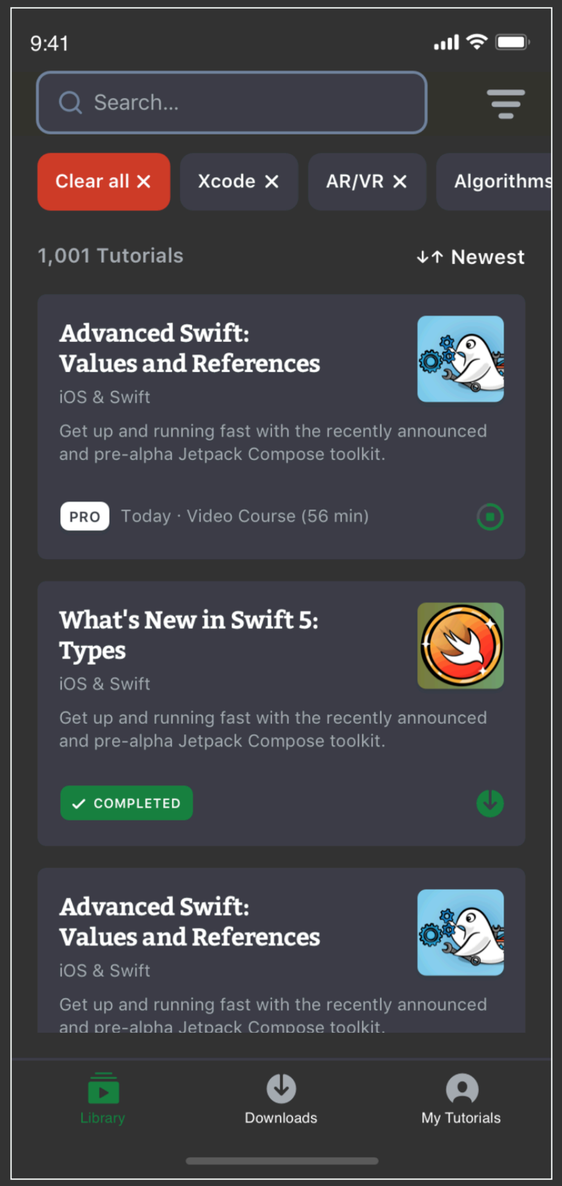
Custom controls shouldn’t be using `@State`. *

Data Flow Through SwiftUI

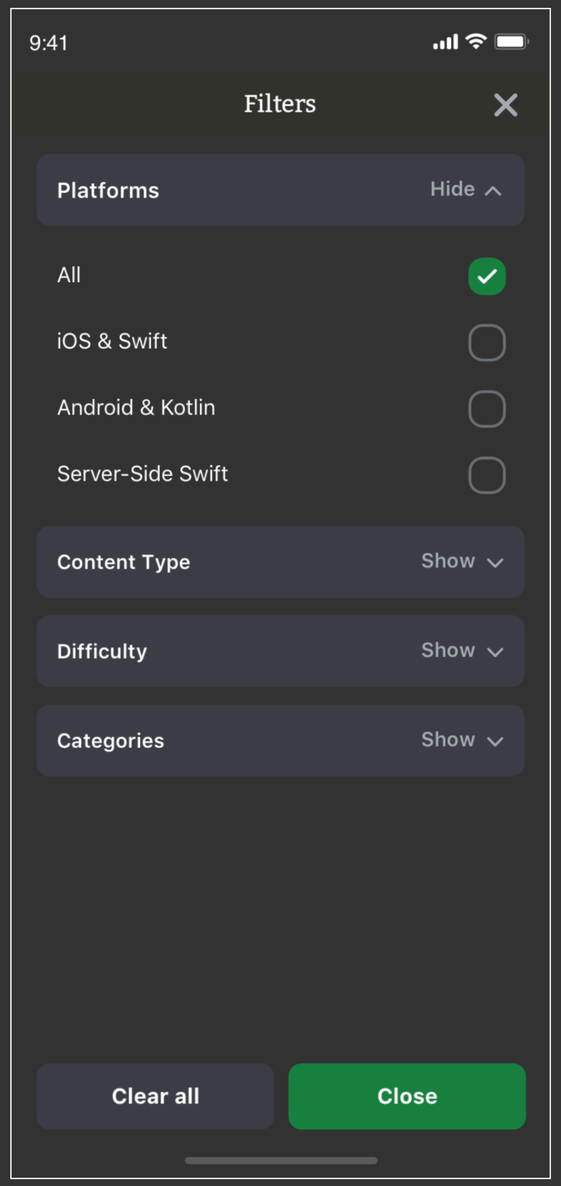
(* or very, very sparingly)

Data Flow/

Library

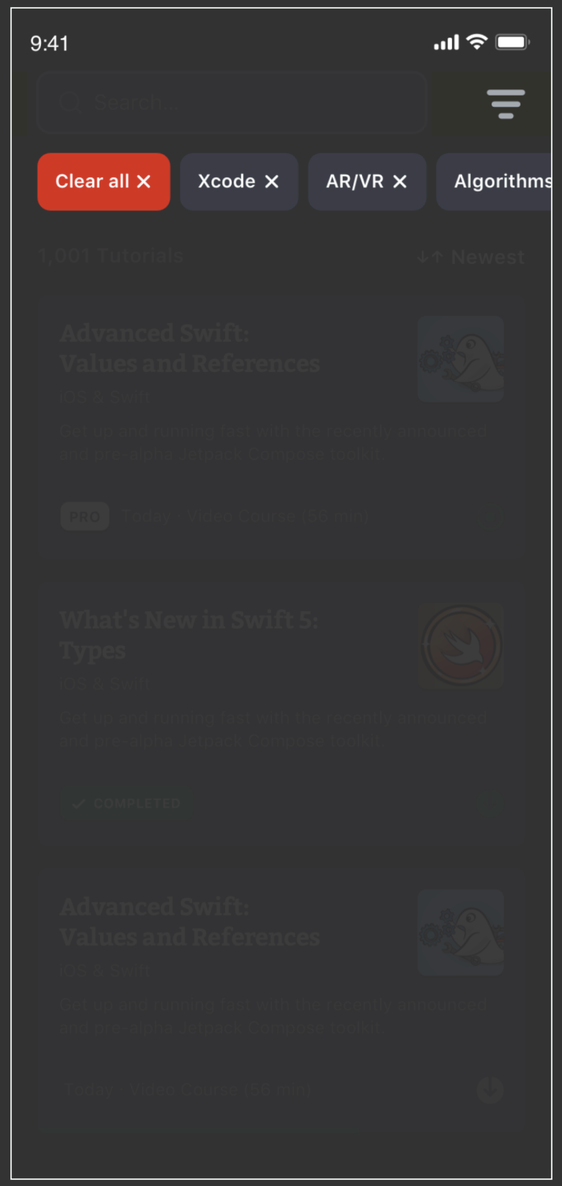


Filters

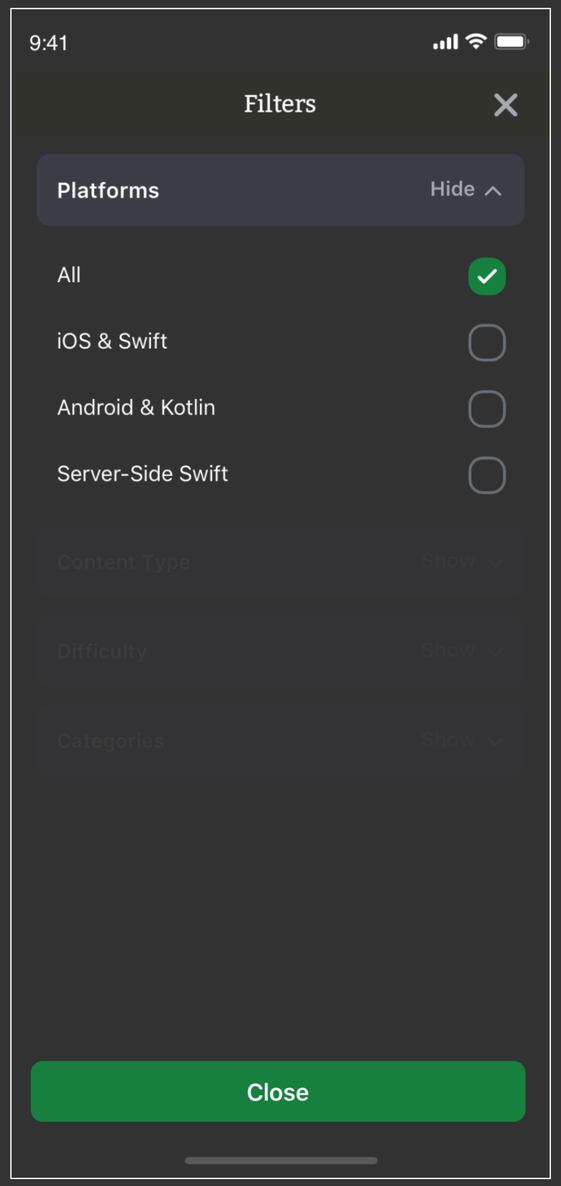


Data Flow/

Library



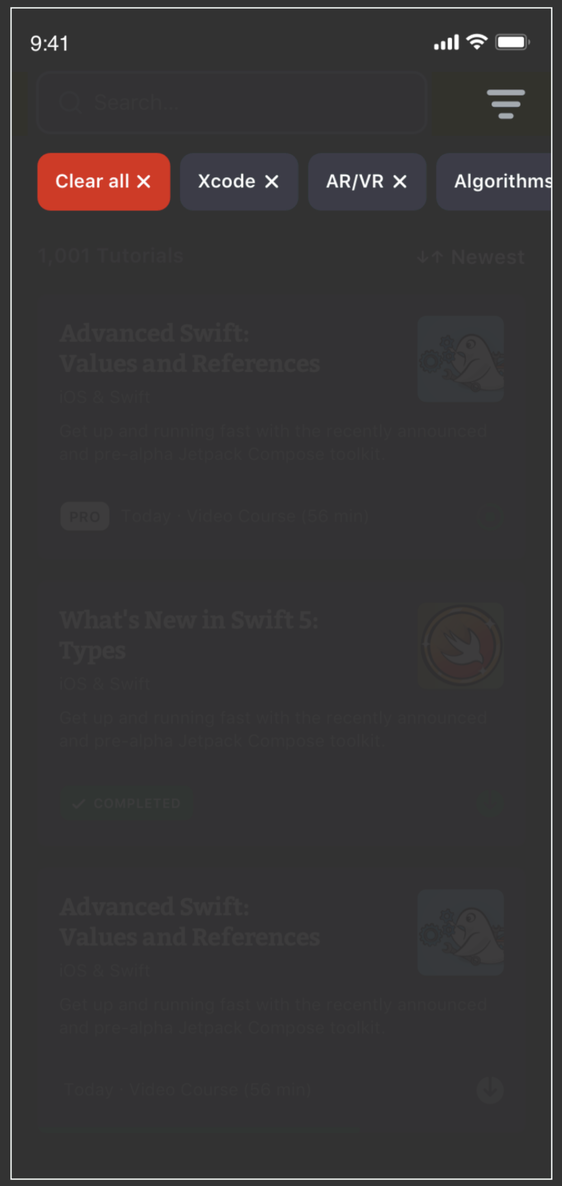
Filters



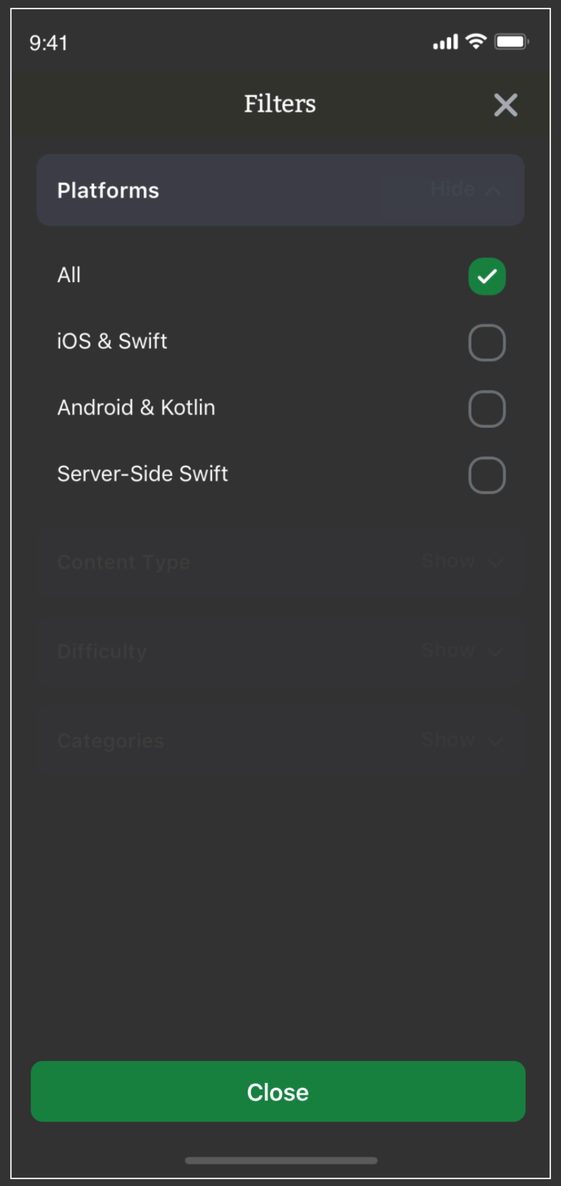
Data Flow/

`FiltersState`

Library



Filters

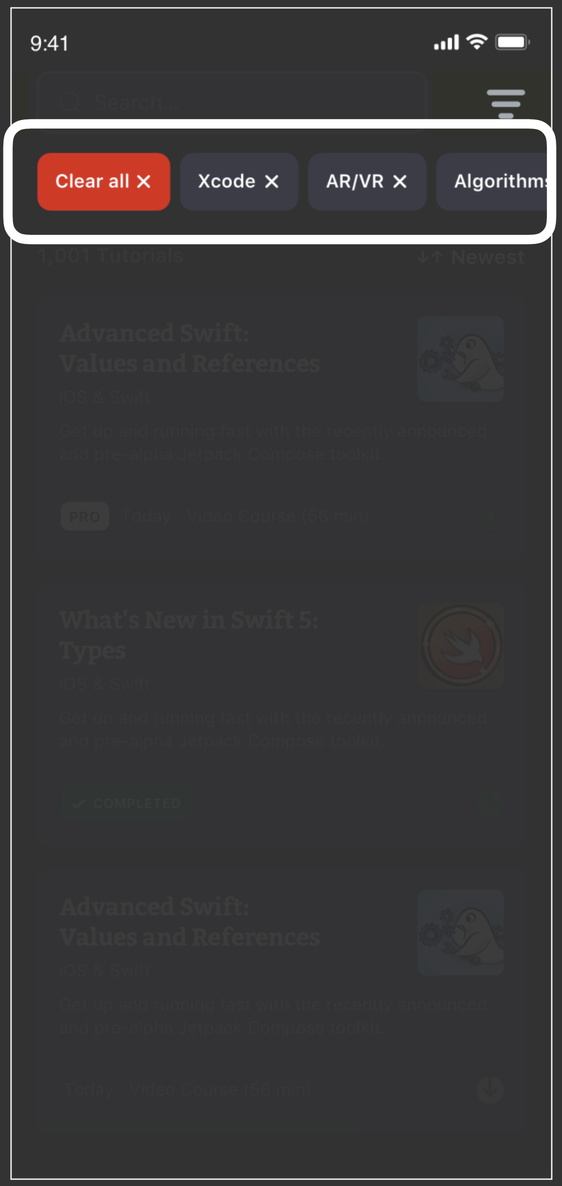


Data Flow/

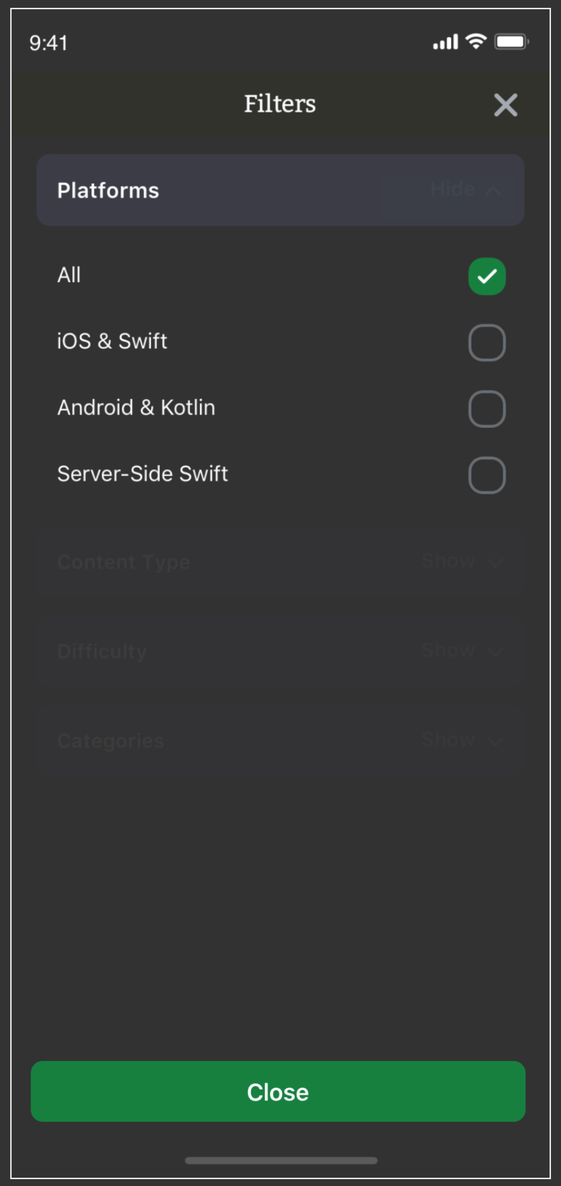
`FiltersState`

`applied`

Library



Filters

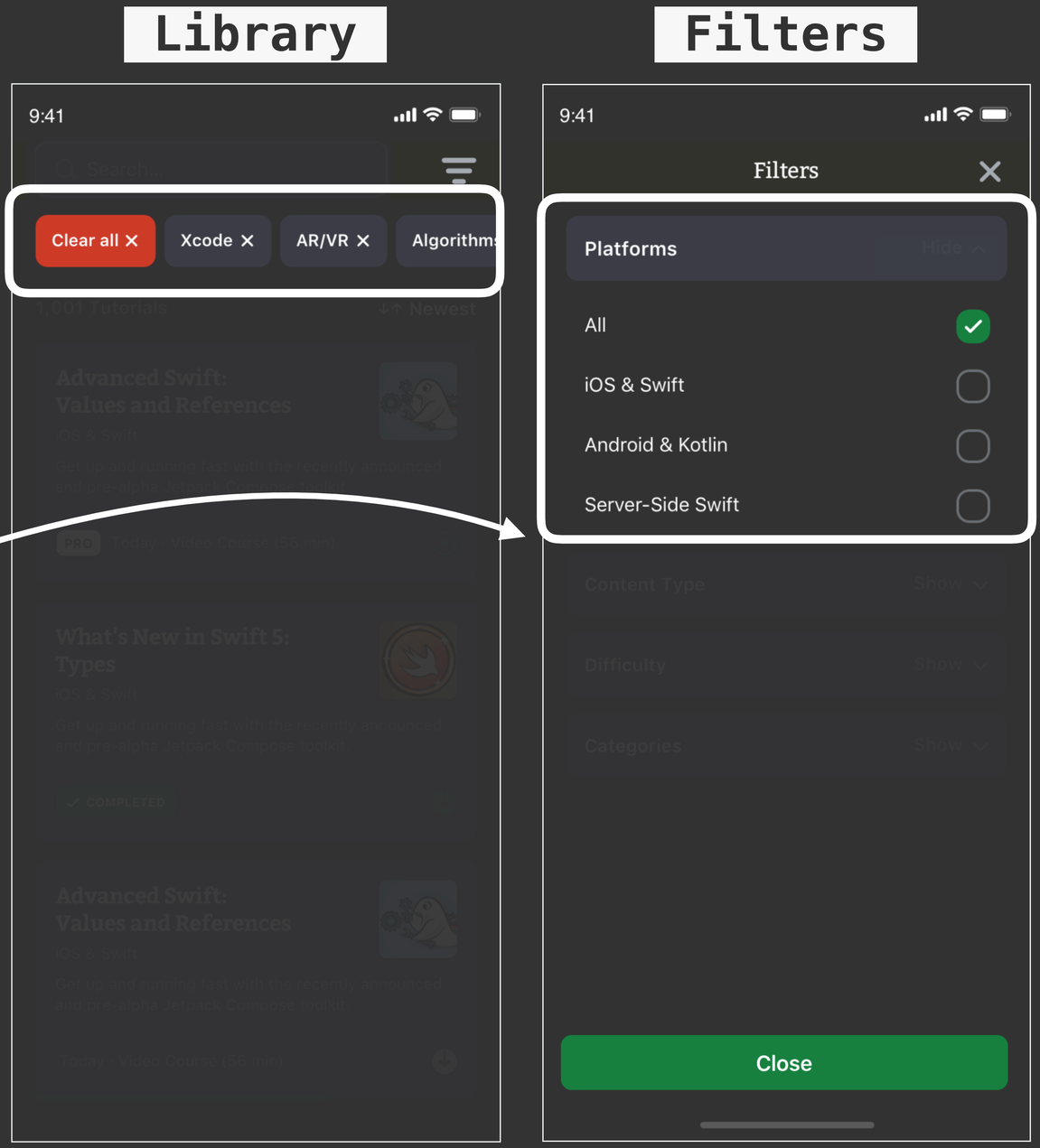


Data Flow/

`FiltersState`

`applied`

`editable`



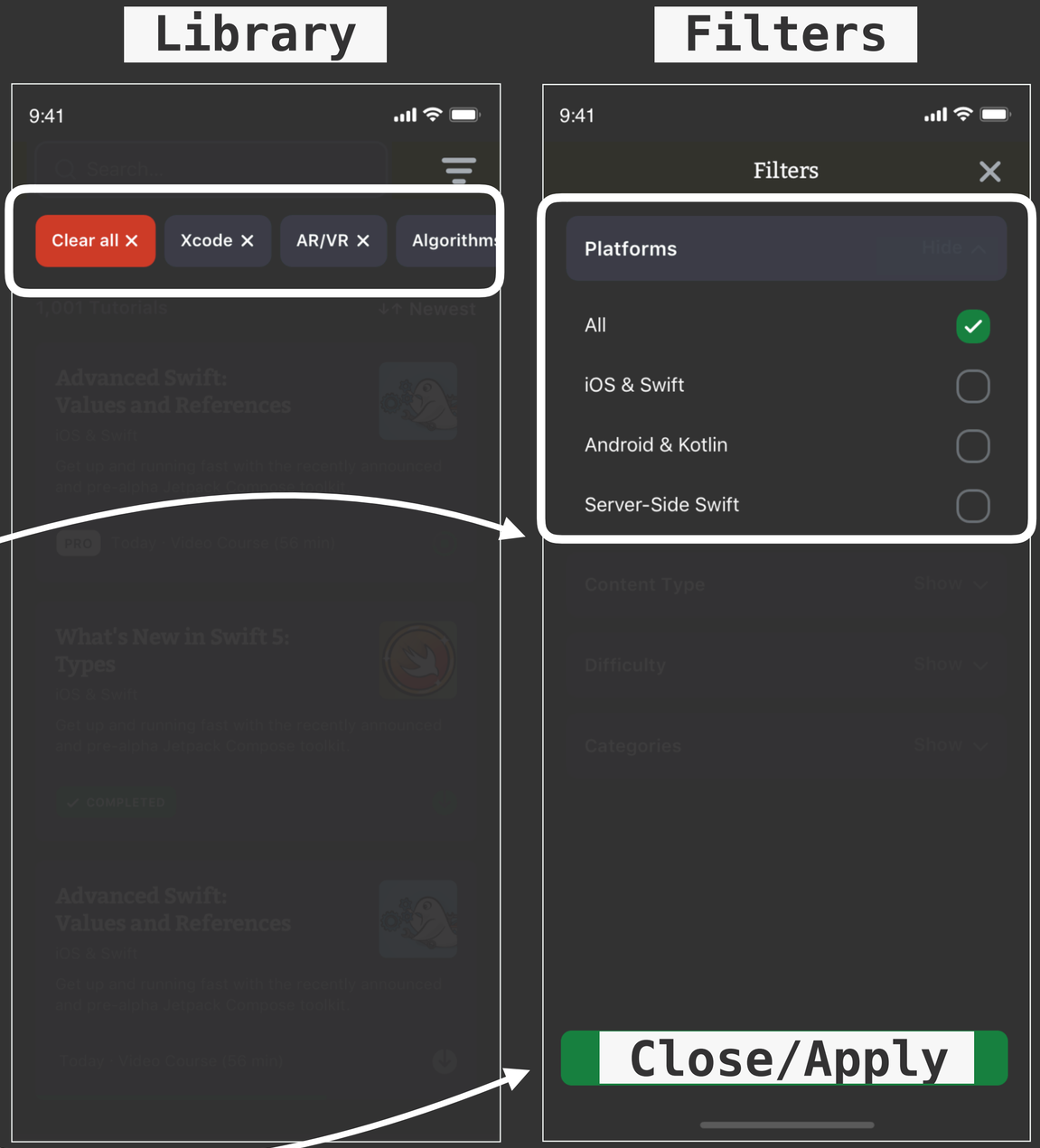
Data Flow/

FiltersState

applied

editable

areFiltersSame



Data Flow/

```
@State

@ObservedObject

@EnvironmentObject
```

Data Flow/

~~@State~~

@ObservedObject

@EnvironmentObject

Data Flow/

1. `@State`, to manage a View's own state
2. `@State` <> `@Binding`, to communicate between Views
3. `@ObservedObject` `@EnvironmentObject` <> `@ObservableObject`

to communicate between a View and its model

Data Flow/

```
class FiltersState {

  var applied: [Filter] = []

  var all: [Filter] = [] {

  var editable: [Filter]

  var areFiltersSame: Bool {
    return all.elementsEqual(editable)
  }

  func clearAll() {...}

  func update(filter: Filter) {...}
}
```

Data Flow/

```
class FiltersState: ObservableObject {

  var applied: [Filter] = []

  var all: [Filter] = []

  var editable: [Filter]

  var areFiltersSame: Bool {
    return all.elementsEqual(editable)
  }

  func clearAll() {...}

  func update(filter: Filter) {...}
}
```

Data Flow/

```
class FiltersState: ObservableObject {

  @Published var applied: [Filter] = []

  @Published var all: [Filter] = []

  @Published var editable: [Filter]

  var areFiltersSame: Bool {
    return all.elementsEqual(editable)
  }

  func clearAll() {...}

  func update(filter: Filter) {...}
}
```

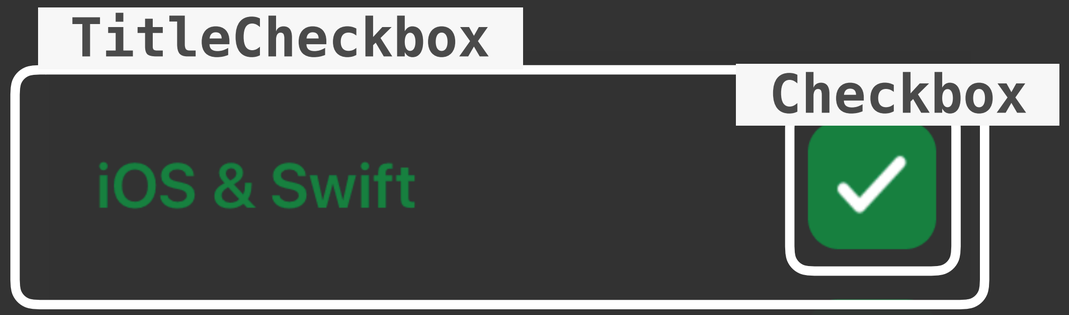
Data Flow/

```
class FiltersState: ObservableObject {

  private(set) var objectWillChange = PassthroughSubject<Void, Never>()

  @Published var applied: [Filter] = []

  @Published var all: [Filter] = []

  @Published var editable: [Filter]

  var areFiltersSame: Bool {
    return all.elementsEqual(editable)
  }

  func clearAll() {...}

  func update(filter: Filter) {...}
}
```

Data Flow/

```
class FiltersState: ObservableObject {

  private(set) var objectWillChange = PassthroughSubject<Void, Never>()

  @Published var applied: [Filter] = []

  @Published var all: [Filter] = []

  @Published var editable: [Filter]

  var areFiltersSame: Bool {
    return all.elementsEqual(editable)
  }

  func clearAll () {
    objectWillChange.send(())
  }

  func update(filter: Filter) {...}
}
```

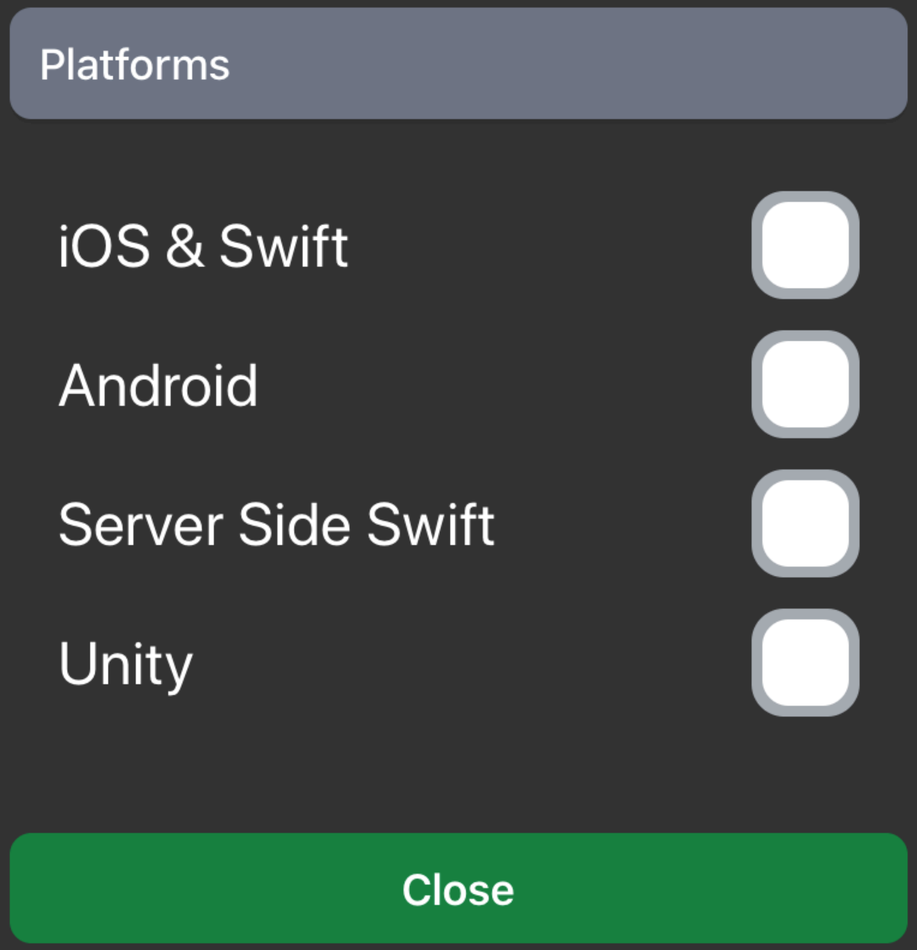
Data Flow/



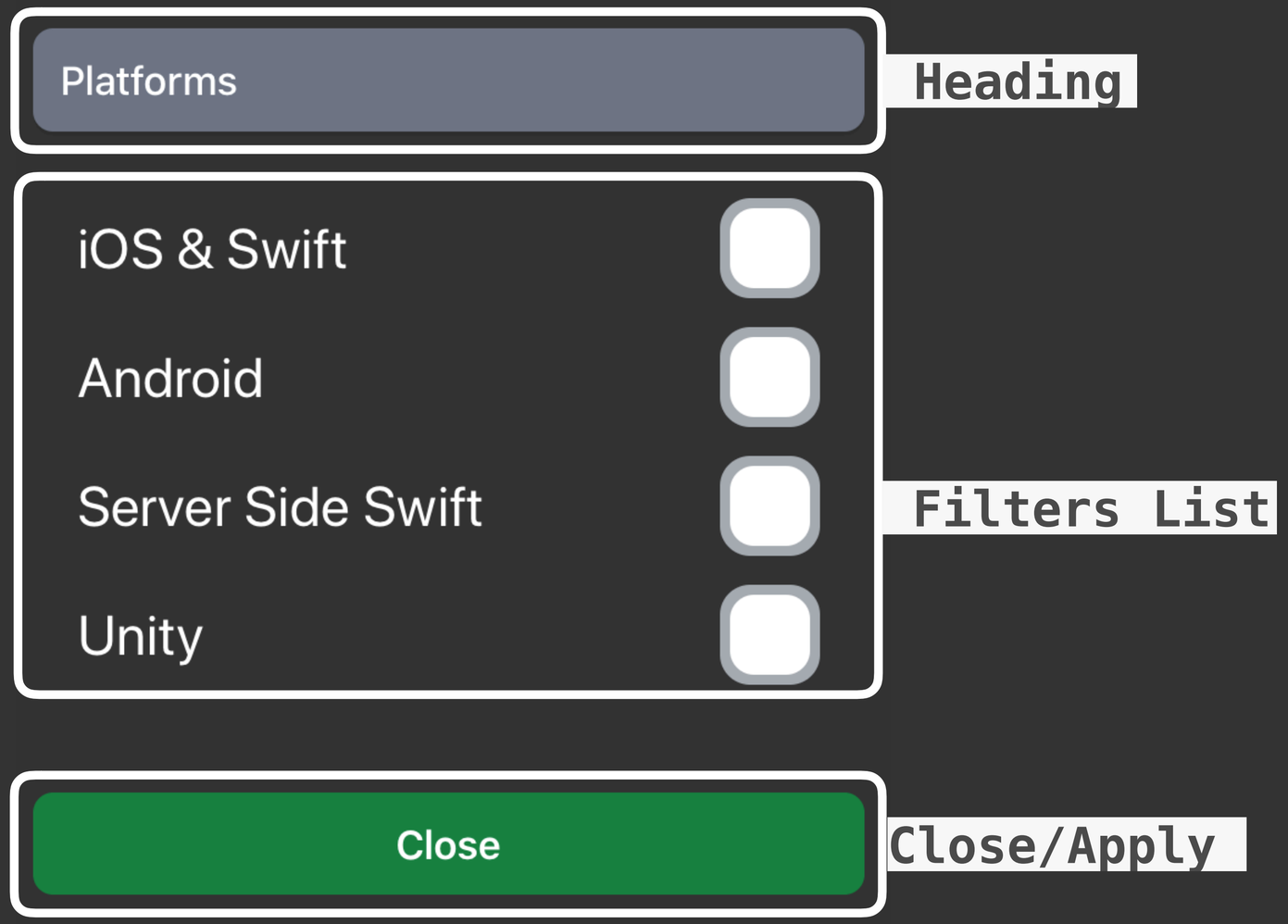

```
struct TitleCheckbox: View {
  @State var isOn: Bool = false
  var name: String

  var body: some View {
    CheckboxBindingView(isOn: $isOn)
  }
}
```

```
struct Checkbox: View {
  @Binding var isOn: Bool

  var body: some View {…}

}
```

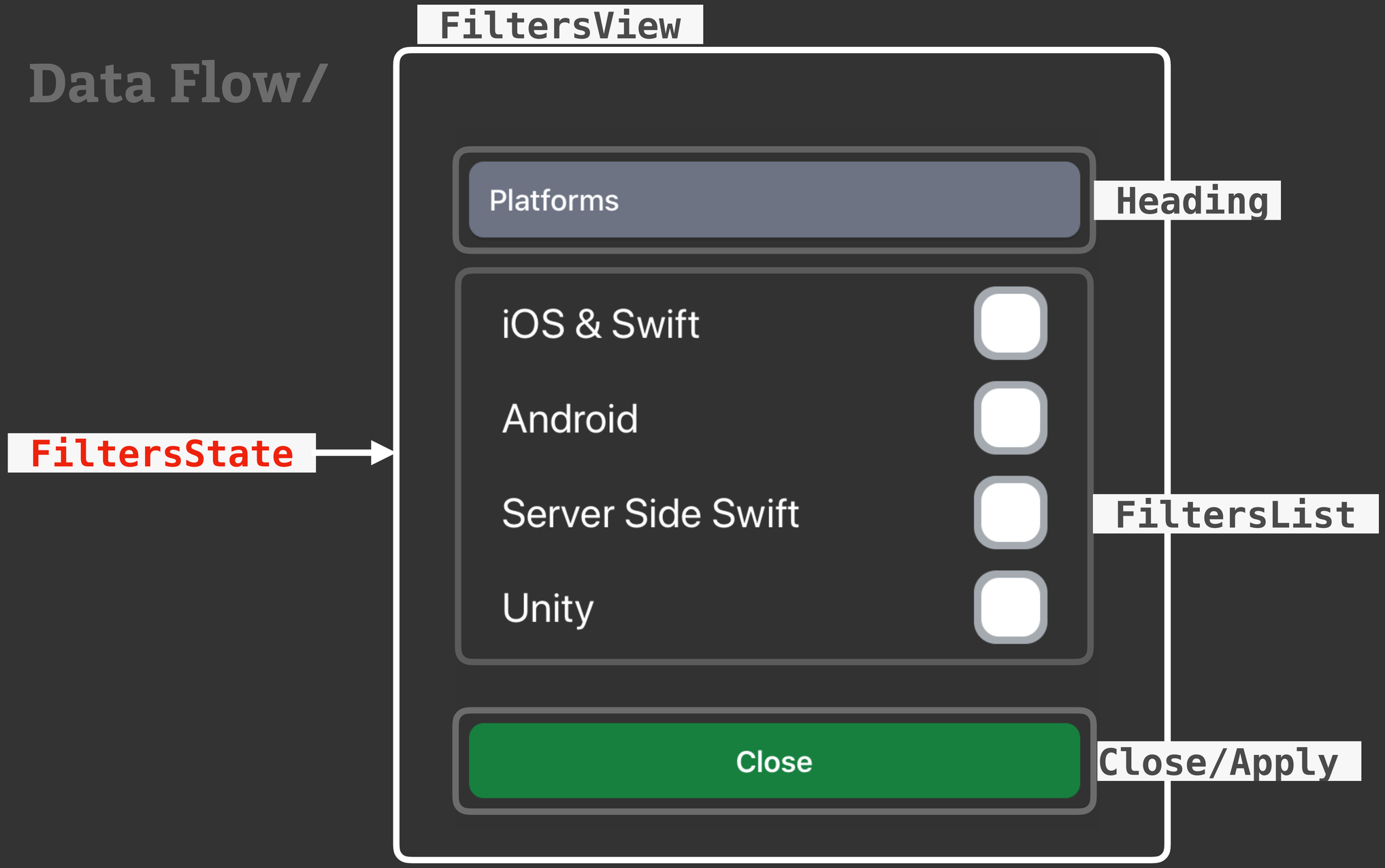
Data Flow/



Data Flow/


Platforms
Heading
iOS & Swift
Android
Server Side Swift
Unity
Filters List
Close
Close/Apply

Data Flow/


FiltersView
Platforms
Heading
iOS & Swift
Android
Server Side Swift
Unity
FiltersList
FiltersState
Close
Close/Apply

Data Flow/

```
@ObservedObject
```

```
FiltersView(state: state)
```

```
@EnvironmentObject
```

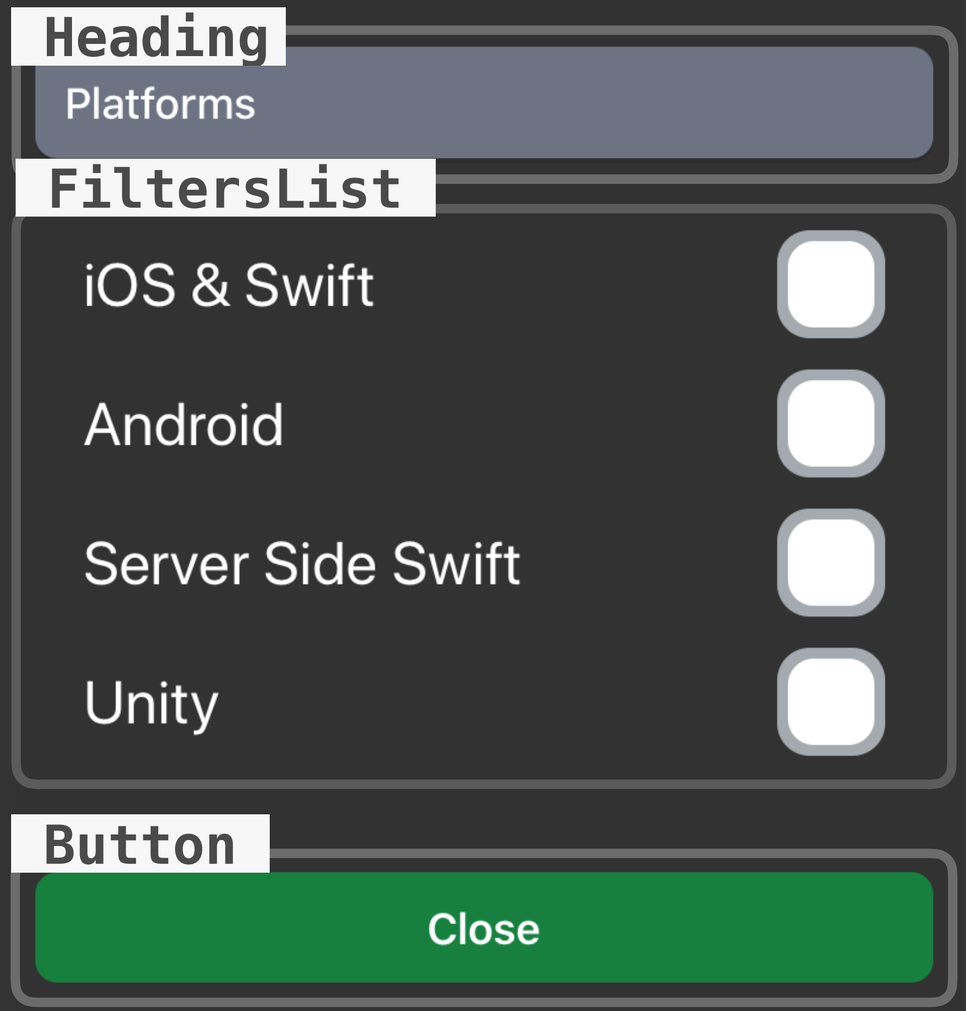
```
FiltersView().environmentObject(state)
```

Singleton!?

```
FiltersView(state: state)
```

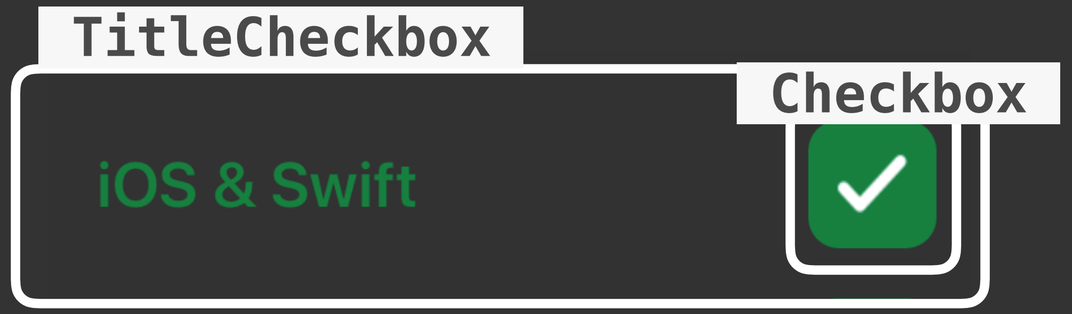
```
@EnvironmentObject

FiltersView().environmentObject(state)
```

Data Flow/



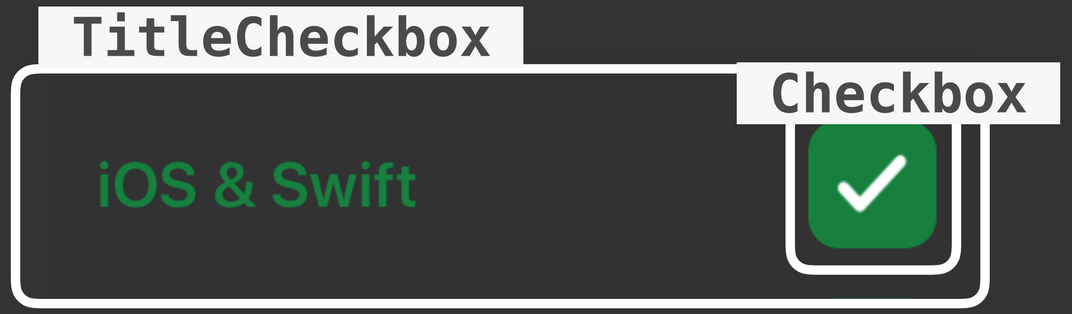

```
struct FiltersView: View {

  @EnvironmentObject var state: FiltersState

  var body: some View {

    return VStack {
      Heading()
      filters()
      button()
   }
  }

  func button() -> ButtonView {...}

  func filters() -> some View {...}
}
```

Data Flow/




```
struct TitleCheckbox: View {

  @State var isOn: Bool
  var name: String

  var body: some View {...}
}
```

Data Flow/




```
struct TitleCheckbox: View {

  @Binding var isOn: Bool
  var name: String

  var body: some View {...}
}
```

Data Flow/

```
struct FiltersView: View {

  @EnvironmentObject var state: FiltersState

  var body: some View {...}
}
```

```
struct TitleCheckbox: View {

  @Binding var isOn: Bool
  var name: String

  var body: some View {...}
}
```

Data Flow/

~~`TitleCheckbox(name: name)`~~

```
TitleCheckbox(isOn: $isOn, name: name)
```

Data Flow/

```
struct FiltersView2: View {

  @EnvironmentObject var state: FiltersState

  var body: some View {...}

  private func filters() -> some View {
    VStack() {

      ForEach(state.editable, id: \.self) { filter in
        TitleCheckbox(isOn:  Binding<Bool> , name:  String )
      }
    }
  }
}
```

Data Flow/

```
struct FiltersView2: View {

  @EnvironmentObject var state: FiltersState

  var body: some View {...}

  private func filtersList() -> some View {
    VStack() {

      ForEach(state.editable, id: \.self) { filter in
        TitleCheckbox(isOn: Binding<Bool>, name: String)
      }
    }
  }
}
```

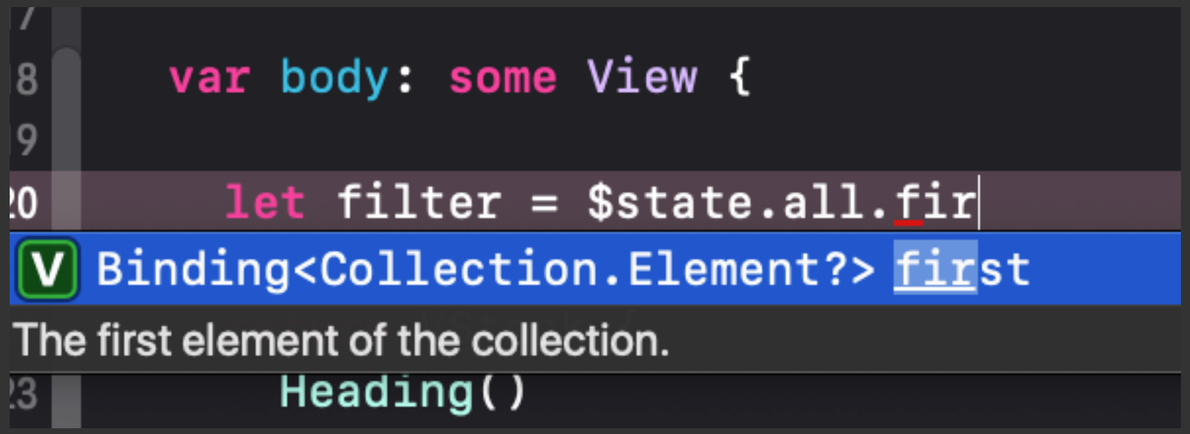
Data Flow/

```
@EnvironmentObject var state: FiltersState
```

```
var body: some View {

    let filter = $state.
        Binding<[Filter]> all
        Binding<[Filter]> applied
            Binding<Bool> areFiltersSame
```

Data Flow/

```
@EnvironmentObject var state: FiltersState
```



Data Flow/

```
struct FiltersView2: View {

  @EnvironmentObject var state: FiltersState

  var body: some View {...}

  private func filtersList() -> some View {
    VStack() {

      ForEach($state.editable, id:\.self) { filter in
        TitleCheckbox(isOn: filter.isOn, name: filter.name)
      }
    }
  }
}
```

Type of expression is ambiguous without more context

Data Flow/

```
struct FiltersView2: View {

  @EnvironmentObject var state: FiltersState

  var body: some View {...}

  private func filtersList() -> some View {
    VStack() {

      ForEach(0..<state.editable.count) { index in
        TitleCheckbox(isOn: self.$filters.editable[index].isOn,
                      name: self.filters.editable[index].name)
      }
    }
  }
}
```

Data Flow/

```
Fatal error: Accessing State<Array<Filter>> outside View.body:
file /BuildRoot/Library/Caches/com.apple.xbs/Sources/
Monoceros_Sim/Monoceros-24.4/Core/State.swift, line 44
```

Data Flow/

```
Fatal error: Accessing State<Array<Filter>> outside View.body:
file /BuildRoot/Library/Caches/com.apple.xbs/Sources/
Monoceros_Sim/Monoceros-24.4/Core/State.swift, line 44
```


```swift
ForEach(0..<state.editable.count) { index in
  TitleCheckbox(isOn: self.$state.editable[index].isOn,
                name: self.state.editable[index].name)
}
```

Do not access `@State` outride of View.body!

Data Flow/

```
ForEach<Range<Int>, Int, AppliedFilterView> count (2) != its
initial count (0). `ForEach(_:content:)` should only be used
for *constant* data. Instead conform data to `Identifiable` or
use `ForEach(_:id:content:)` and provide an explicit `id`!
```

```
ForEach(0..<state.editable.count) { index in
  TitleCheckbox(isOn: self.$state.editable[index].isOn,
                name: self.state.editable[index].name)
}
```

Data Flow/

```
Fatal error: Accessing State<Array<Filter>> outside View.body:
file /BuildRoot/Library/Caches/com.apple.xbs/Sources/
Monoceros_Sim/Monoceros-24.4/Core/State.swift, line 44
```

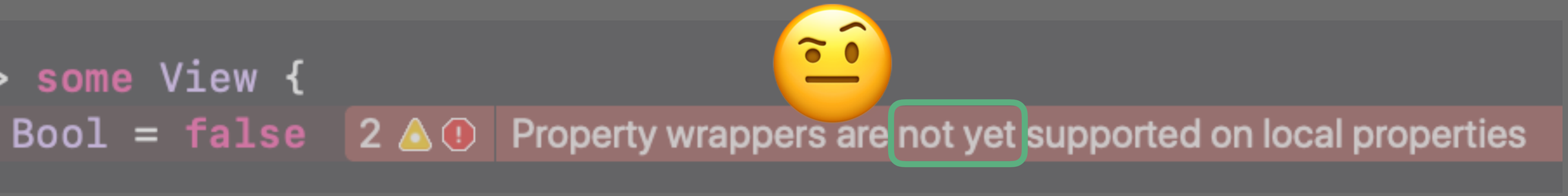
```
private func filtersList() -> some View {
  @State var willYouLetMeBe: Bool = false   2 ⚠ ⓘ  Property wrappers are not yet supported on local properties
```

Data Flow/

```
Fatal error: Accessing State<Array<Filter>> outside View.body:
file /BuildRoot/Library/Caches/com.apple.xbs/Sources/
Monoceros_Sim/Monoceros-24.4/Core/State.swift, line 44
```

```
> some View {
Bool = false   2 ⚠ ⛔ | Property wrappers are not yet supported on local properties
```

```
Fatal error: Accessing State<Array<Filter>> outside View.body:
file /BuildRoot/Library/Caches/com.apple.xbs/Sources/
Monoceros_Sim/Monoceros-24.4/Core/State.swift, line 44
```



Data Flow/

```
struct TitleCheckbox: View {
  @Binding var isOn: Bool
  var name: String

  var body: some View {
    HStack {
      TitleText(text: name)
      Spacer()
      Checkbox(isOn: $isOn)
    }
  }
}
```

```
struct Checkbox: View {
  @Binding var isOn: Bool

  var body: some View {

    Button(action: {
      self.isOn.toggle()
    }) {
      if isOn { OnView() }
      else { OffView() }
    }
  }
}
```

Data Flow/

```
struct TitleCheckbox: View {
  @Binding var isOn: Bool
  var name: String

  var body: some View {
    HStack {
      TitleText(text: name)
      Spacer()
      Checkbox(isOn: $isOn)
    }
  }
}
```

```
struct Checkbox: View {
  @Binding var isOn: Bool

  var body: some View {

    Button(action: {
      self.isOn.toggle()
    }) {
      if isOn { OnView() }
      else { OffView() }
    }
  }
}
```

Data Flow/

```
struct TitleCheckbox: View {
 @State var filter: Filter

  var body: some View {
    HStack {
      TitleText(text: name)
      Spacer()
      Checkbox(isOn: $isOn)
    }
  }
}
```

```
struct Checkbox: View {
  @Binding var filter: Filter

  var body: some View {

    Button(action: {
      self.isOn.toggle()
    }) {
      if isOn { OnView() }
      else { OffView() }
    }
  }
}
```

Data Flow/

Step 1

Create a new custom view called `CategoryItem` alongside `CategoryRow`, and replace the `Text` that holds the landmark name text with the new view.

```
26 struct CategoryItem: View {
27     var landmark: Landmark
28     var body: some View {
29         VStack(alignment: .leading) {
30             landmark.image
31                 .resizable()
32                 .frame(width: 155, height: 155)
33                 .cornerRadius(5)
34             Text(landmark.name)
35                 .font(.caption)
36         }
37         .padding(.leading, 15)
38     }
```

Composing complex interfaces

Data Flow/

```
private func filtersList() -> some View {
  return VStack {
    ForEach(state.editable, id:\.self) { filter in
      TitleCheckbox(filter: filter)
    }
  }
}
```

```
struct TitleCheckbox: View {
  @State var filter: Filter

  var body: some View {
    Checkbox(filter: $filter)
  }
}
```

Data Flow/

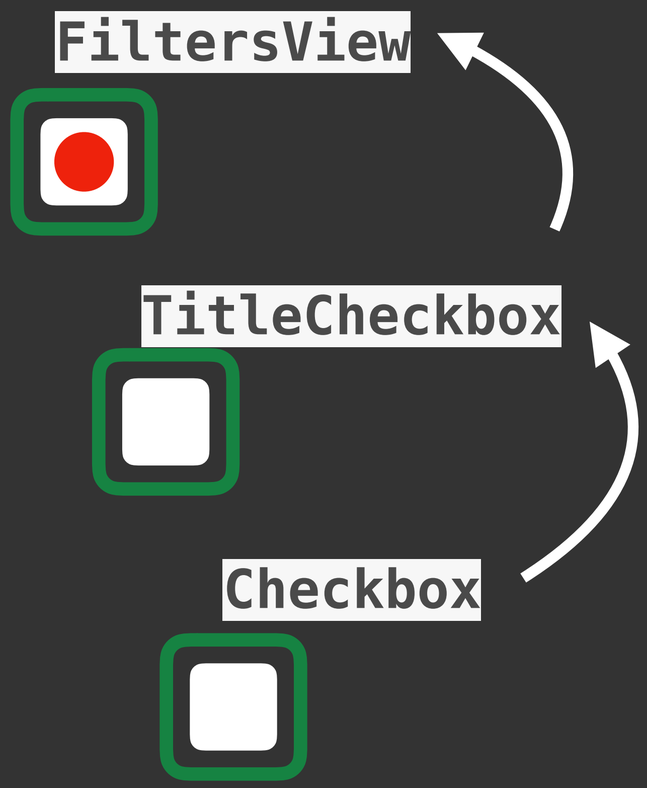
~~`TitleCheckbox(isOn: $isOn, name: name)`~~

```
TitleCheckbox(filter: filter)
```

Data Flow/

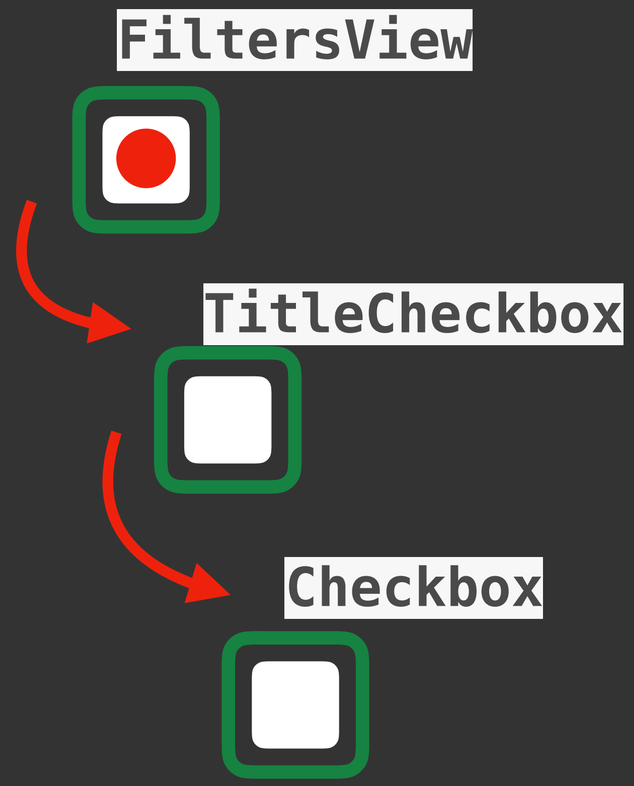
```
struct Checkbox: View {
  @Binding var filter: Filter

  var body: some View {

    Button(action: {
      self.isOn.toggle()
    }) {
      if isOn { OnView() }
      else { OffView() }
    }
  }
}
```

Data Flow/

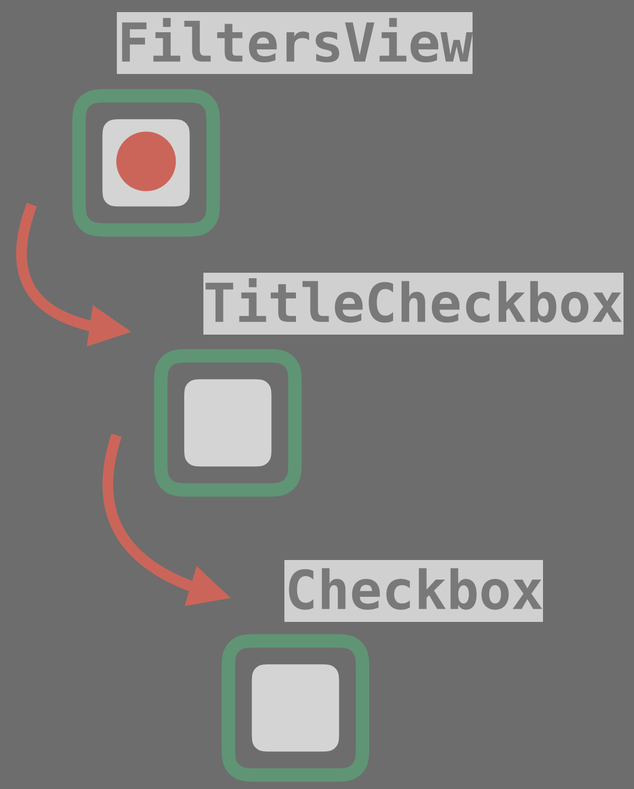



```
struct Checkbox: View {
  @Binding var filter: Filter

  var body: some View {

    Button(action: {
      self.filter.isOn.toggle()
    }) {
      if isOn { OnView() }
      else { OffView() }
    }
  }
}
```

Data Flow/

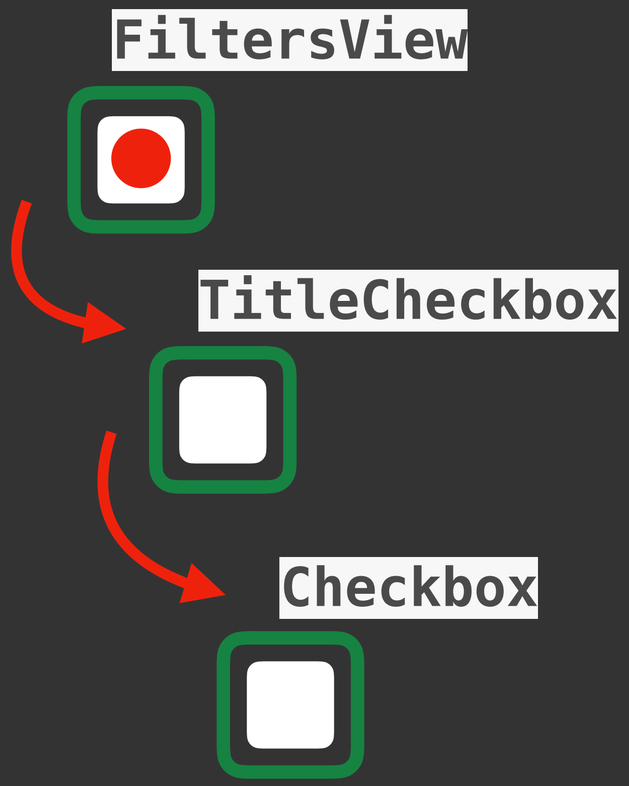



```
struct Checkbox: View {
  @Binding var filter: Filter

  var body: some View {

    Button(action: {
      self.filter.isOn.toggle()
    }) {
      if filter.isOn { OnView() }
      else { OffView() }
    }
  }
}
```



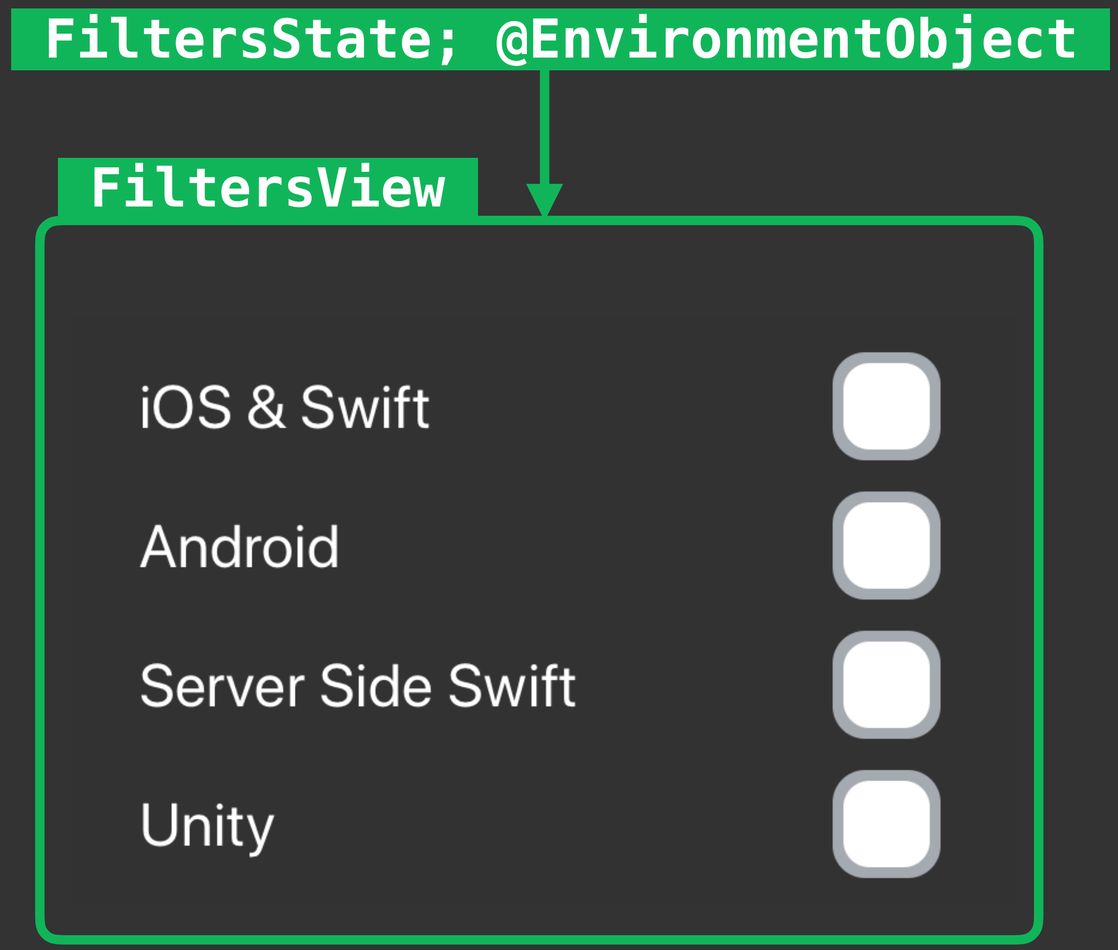

```
struct Checkbox: View {
  @Binding var filter: Filter

  var body: some View {

    Button(
      self.
      isOn.toggle()
    }) {
      if filter.isOn { OnView() }
      else { OffView() }
    }
  }
}
```

Data Flow/



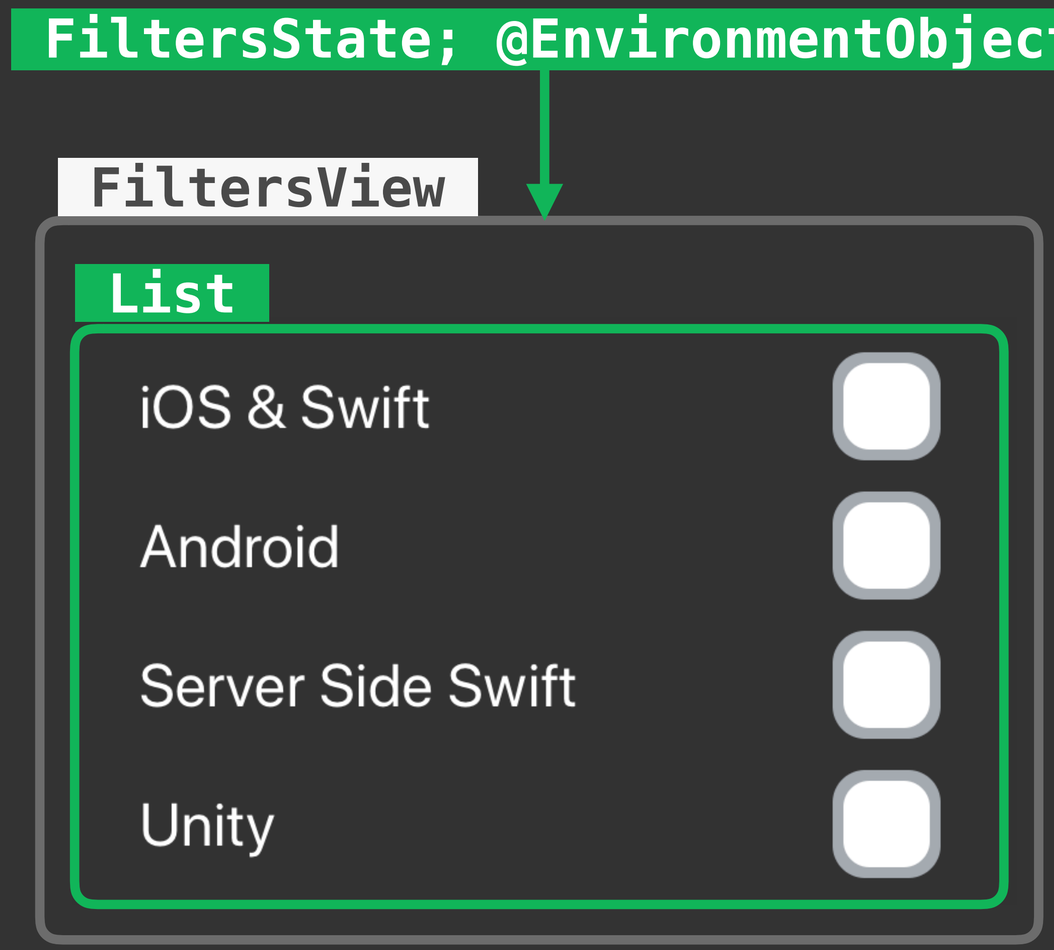

```
struct Checkbox: View {
  @Binding var filter: Filter

  var body: some View {

    Button(action: {
      self.filter.isOn.toggle()
    }) {
      if filter.isOn { OnView() }
      else { OffView() }
    }
  }
}
```

Data Flow/



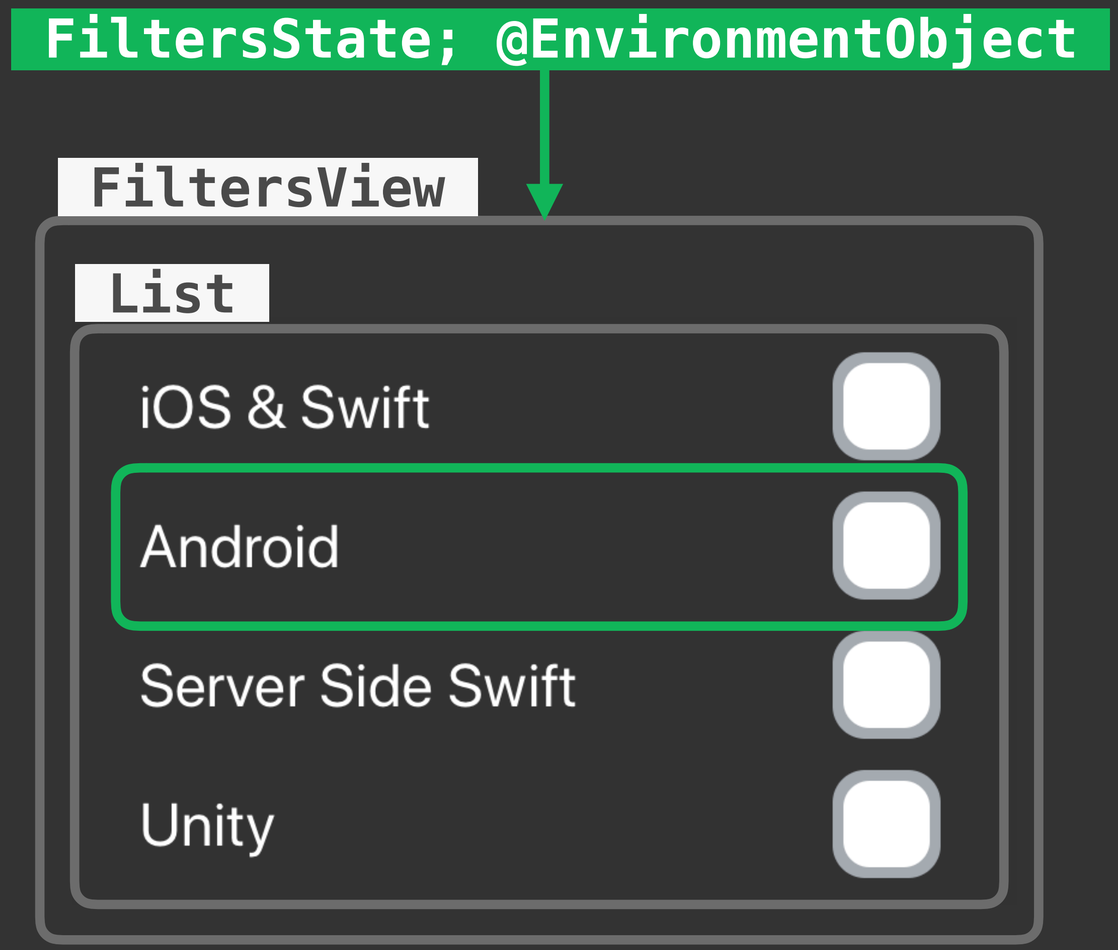

```
struct Checkbox: View {
  @Binding var filter: Filter

  var body: some View {

    Button(action: {
      self.filter.isOn.toggle()
    }) {
      if filter.isOn { OnView() }
      else { OffView() }
    }
  }
}
```

Data Flow/



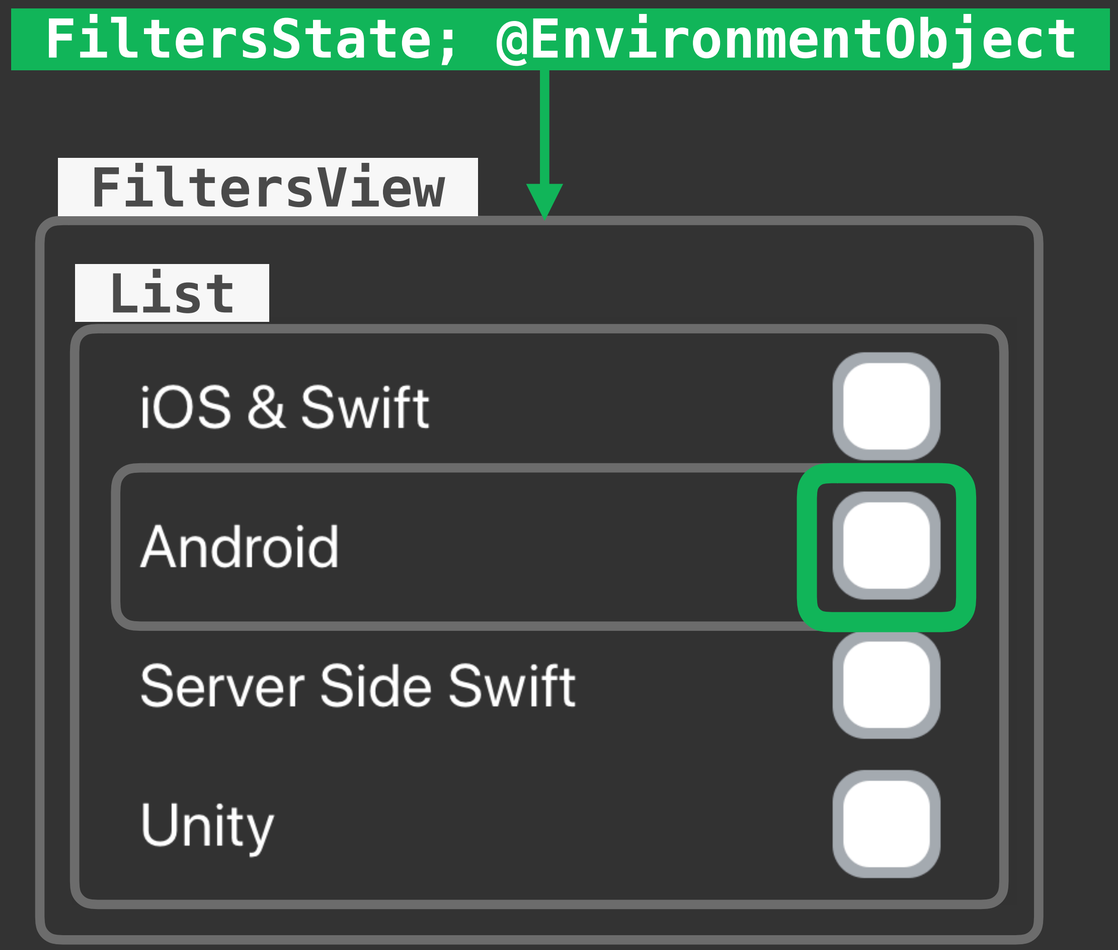

```
struct Checkbox: View {
  @Binding var filter: Filter

  var body: some View {

    Button(action: {
      self.filter.isOn.toggle()
    }) {
      if filter.isOn { OnView() }
      else { OffView() }
    }
  }
}
```

Data Flow/




```
struct Checkbox: View {
  @Binding var filter: Filter

  var body: some View {

    Button(action: {
      self.filter.isOn.toggle()
    }) {
      if filter.isOn { OnView() }
      else { OffView() }
    }
  }
}
```

Data Flow/




```
struct Checkbox: View {
  @Binding var filter: Filter

  var body: some View {

    Button(action: {
      self.filter.isOn.toggle()
    }) {
      if filter.isOn { OnView() }
      else { OffView() }
    }
  }
}
```

Data Flow/

```
struct Checkbox: View {
  @Binding var filter: Filter

  var body: some View {

    Button(action: {
      self.filter.isOn.toggle()
    }) {
      if filter.isOn { OnView() }
      else { OffView() }
    }
  }
}
```

Data Flow/

```
struct Checkbox: View {
  @Binding var filter: Filter
  @EnvironmentObject var state: FiltersState

  var body: some View {

    Button(action: {
      self.filter.isOn.toggle()
    }) {
      if filter.isOn { OnView() }
      else { OffView() }
    }
  }
}
```

Data Flow/

```
struct Checkbox: View {
  @Binding var filter: Filter
  @EnvironmentObject var state: FiltersState

  var body: some View {

    Button(action: {
      self.filter.isOn.toggle()
    }) {
      if filter.isOn { OnView() }
      else { OffView() }
    }
  }
}
```

Data Flow/

```
struct Checkbox: View {
  @Binding var filter: Filter
  @EnvironmentObject var state: FiltersState

  var body: some View {

    Button(action: {
      let index = state.editable.firstIndex(of: filter)!
    }) {
      if filter.isOn { OnView() }
      else { OffView() }
    }
  }
}
```

Data Flow/

```
struct Checkbox: View {
  @Binding var filter: Filter
  @EnvironmentObject var state: FiltersState

  var body: some View {

    Button(action: {
      let index = state.editable.firstIndex(of: filter)!
      self.state.editable[self.index].isOn.toggle()
    }) {
      if filter.isOn { OnView() }
      else { OffView() }
    }
  }
}
```

Data Flow/

```
struct Checkbox: View {
  @Binding var filter: Filter
  @EnvironmentObject var state: FiltersState

  var body: some View {

    Button(action: {
      let index = state.editable.firstIndex(of: filter)!
      self.state.editable[self.index].isOn.toggle()
    }) {
      if filter.isOn { OnView() }
      else { OffView() }
    }
  }
}
```

Data Flow/

```
struct LandmarkDetail: View {
    @EnvironmentObject var userData: UserData
    var landmark: Landmark

    var landmarkIndex: Int {
        userData.landmarks.firstIndex(where: { $0.id == landmark.id })!
    }

    var body: some View {
      Button(action: {
        self.userData.landmarks[self.landmarkIndex].isFavorite.toggle()
      }) {
      if self.userData.landmarks[self.landmarkIndex].isFavorite {
        Image(systemName: "star.fill")
          .foregroundColor(Color.yellow)
      } else {
        Image(systemName: "star")
          .foregroundColor(Color.gray)
      }
    }

}
```

Data Flow/

```
struct Checkbox: View {
  @Binding var filter: Filter
  @EnvironmentObject var state: FiltersState

  var body: some View {

    Button(action: {
      let index = state.editable.firstIndex(of: filter)!
      self.state.editable[self.index].isOn.toggle()
    }) {
      if filter.isOn { OnView() }
      else { OffView() }
    }
  }
}
```

Data Flow/

```
struct Checkbox: View {
  @Binding var filter: Filter
  @EnvironmentObject var state: FiltersState

  var body: some View {

    Button(action: {
      let index = state.editable.firstIndex(of: filter)!
      self.state.editable[self.index].isOn.toggle()
    }) {
      if filter.isOn { OnView() }
      else { OffView() }
    }
  }
}
```

Data Flow/

```
struct Checkbox: View {
  @Binding var filter: Filter
  @EnvironmentObject var state: FiltersState

  var body: some View {

    Button(action: {
      let index = state.editable.firstIndex(of: filter)!
      self.state.editable[self.index].isOn.toggle()
    }) {
      if filter.isOn { OnView() }
      else { OffView() }
    }
  }
}
```

Data Flow/

```
struct Checkbox: View {
  @Binding var filter: Filter
  @EnvironmentObject var state: FiltersState

  var body: some View {

    Button(action: {
      let index = state.editable.firstIndex(of: filter)!
      self.state.editable[self.index].isOn.toggle()
    }) {
      if filter.isOn { OnView() }
      else { OffView() }
    }
  }
}
```

Data Flow/

```
struct Checkbox: View {
  @Binding var filter: Filter
  @EnvironmentObject var state: FiltersState

  var body: some View {

    Button(action: {
      let filtered = state.editable.filter { $0.name == filter.name }.first!
      let index = state.editable.firstIndex(of: filtered)!
      self.state.editable[self.index].isOn.toggle()
    }) {
      if filter.isOn { OnView() }
      else { OffView() }
    }
  }
}
```

Data Flow/

Data Flow/

Not reusable.

Data Flow/

Not reusable.

Code smells...

Checkbox

🤔

```
let filtered = state.editable.filter { $0.name == filter.name }.first!
let index = state.editable.firstIndex(of: filtered)!
self.state.editable[self.index].isOn.toggle()
```

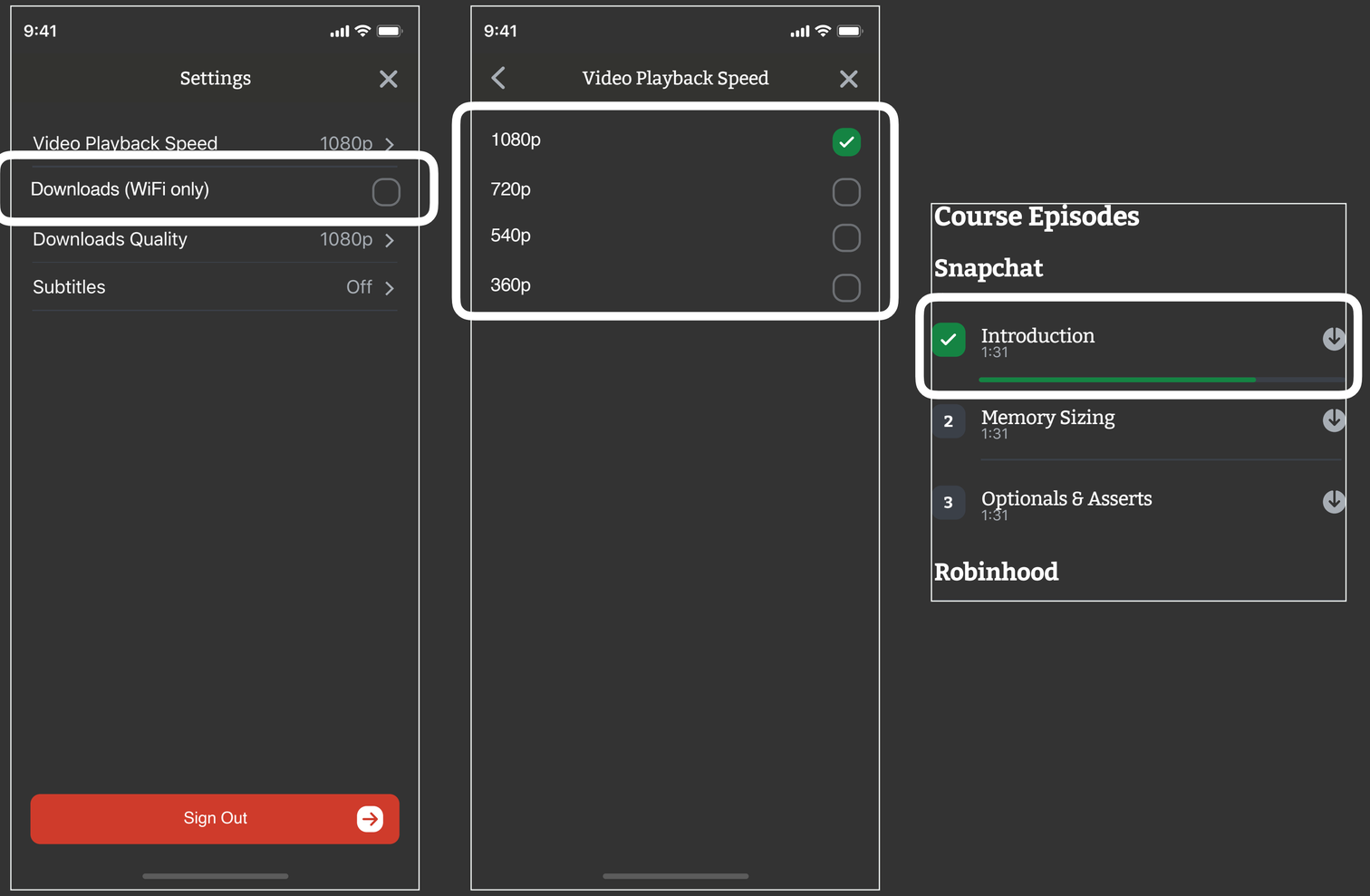
Not reusable.

Code smells...

Checkbox

```
let filtered = state.editable.filter { $0.name == filter.name }.first!
let index = state.editable.firstIndex(of: filtered)!
self.state.editable[self.index].isOn.toggle()
```

Data Flow/


9:41
Settings
Video Playback Speed
1080p
Downloads (WiFi only)
Downloads Quality
1080p
Subtitles
Off
Sign Out
9:41
Video Playback Speed
1080p
720p
540p
360p
Course Episodes
Snapchat
Introduction
1:31
2
Memory Sizing
1:31
3
Optionals & Asserts
1:31
Robinhood

Data Flow/

```
struct TitleCheckbox: View {
  @Binding var isOn: Bool
  var name: String

  var body: some View {
    HStack {
      TitleText(text: name)
      Spacer()
      Checkbox(isOn: $isOn)
    }
  }
}
```

```
struct Checkbox: View {
  @Binding var isOn: Bool

  var body: some View {

    Button(action: {
      self.isOn.toggle()
    }) {
      if isOn { OnView() }
      else { OffView() }
    }
  }
}
```

Data Flow/

```
struct TitleCheckbox: View {
  var isOn: Bool
  var name: String

  var body: some View {
    HStack {
      TitleText(text: name)
      Spacer()
      Checkbox(isOn: $isOn)
    }
  }
}
```

```
struct Checkbox: View {
  var isOn: Bool

  var body: some View {

    Button(action: {
      self.isOn.toggle()
    }) {
      if isOn { OnView() }
      else { OffView() }
    }
  }
}
```

Data Flow/

```
struct TitleCheckbox: View {
  var isOn: Bool
  var name: String
  var onChange: (Bool) -> Void


  var body: some View {
    HStack {
      TitleText(text: name)
      Spacer()
      Checkbox(isOn: $isOn)
    }
  }
}
```

```
struct Checkbox: View {
  var isOn: Bool
  var onChange: (Bool) -> Void

  var body: some View {

    Button(action: {
      self.isOn.toggle()
    }) {
      if isOn { OnView() }
      else { OffView() }
    }
  }
}
```

Data Flow/

```
struct TitleCheckbox: View {
  var isOn: Bool
  var name: String
  var onChange: (Bool) -> Void


  var body: some View {
    HStack {
      TitleText(text: name)
      Spacer()
      Checkbox(isOn: isOn, onChange: onChange)
    }
  }
}
```

```
struct Checkbox: View {
  var isOn: Bool
  var onChange: (Bool) -> Void

  var body: some View {

    Button(action: {
      self.onChange(!self.isOn)
    }) {
      if isOn { OnView() }
      else { OffView() }
    }
  }
}
```

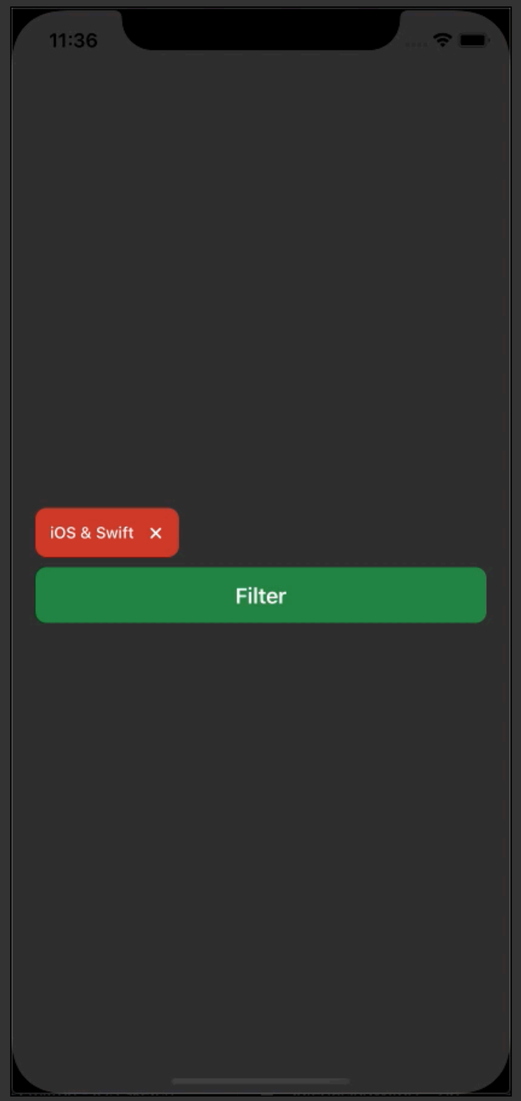
Data Flow/

```
func filtersList() -> some View {

  VStack {
    ForEach(state.editable, id:\.self) { filter in
      TitleCheckbox(isOn: filter.isOn, name: filter.name) { isOn in
        guard let index = self.state.editable.firstIndex(of: filter)
        else { return }

        self.state.editable[index].isOn = isOn
      }
    }
  }
}
```

Data Flow/

```
func filtersList() -> some View {

  VStack {
    ForEach(state.editable, id:\.self) { filter in
      TitleCheckbox(isOn: filter.isOn, name: filter.name) { isOn in
        guard let index = self.state.editable.firstIndex(of: filter)
        else { return }

        self.state.editable[index].isOn = isOn
      }
    }
  }
}
```

Data Flow/



Recap

Recap

1. **What SwiftUI is**

Recap

1. **What SwiftUI is**

2. **SwiftUI VS UIKit**

Recap

1. **What SwiftUI is**
2. **SwiftUI VS UIKit**
3. **Passing data between Views (and Models)**

Recap

1. **What SwiftUI is**
2. **SwiftUI VS UIKit**
3. **Passing data between Views (and Models)**
4. **Challenges of data-binding for Lists**

Recap

1. **What SwiftUI is**
2. **SwiftUI VS UIKit**
3. **Passing data between Views**
4. **Challenges of data-binding for Lists**
5. **Approach the unknown by experimenting**

Recap

1. **What SwiftUI is**
2. **SwiftUI VS UIKit**
3. **Passing data between Views**
4. **Challenges of data-binding for Lists**
5. **Approach the unknown by experimenting**

Recap

	Views	
Data	**Reusable**	**Specific**
Constant	Enumerated + Binding	Identifiable + Binding
Dynamic	Identifiable + Callbacks	Identifiable + Binding

Recap

```
ForEach(state.editable, id:\.self) { filter in
  TitleCheckbox(isOn: $filter.isOn, name: filter.name)
}
```

```
struct TitleCheckbox: View {
  @Binding var isOn: Bool
  var name: String

  var body: some View {
    HStack {
      TitleText(text: name)
      Spacer()
      Checkbox(isOn: $isOn)
    }
  }
}
```

```
struct Checkbox: View {
  @Binding var isOn: Bool

  var body: some View {

    Button(action: {
      self.isOn.toggle()
    }) {
      if isOn { OnView() }
      else { OffView() }
    }
  }
}
```

Recap

```
ForEach(state.editable, id:\.self) { filter in
  TitleCheckbox(isOn: $filter.isOn, name: filter.name)
}
```

```
struct TitleCheckbox: View {
  @Binding var isOn: Bool
  var name: String

  var body: some View {
    HStack {
      TitleText(text: name)
      Spacer()
      Checkbox(isOn: $isOn)
    }
  }
}
```

```
struct Checkbox: View {
  @Binding var isOn: Bool

  var body: some View {

    Button(action: {
      self.isOn.toggle()
    }) {
      if isOn { OnView() }
      else { OffView() }
    }
  }
}
```

Recap

```
ForEach(state.editable, id:\.self) { filter in
  TitleCheckbox(isOn: $filter.isOn, name: filter.name)
}
```

```
struct TitleCheckbox: View {
  @Binding var isOn: Bool
  var name: String

  var body: some View {
    HStack {
      TitleText(text: name)
      Spacer()
      Checkbox(isOn: $isOn)
    }
  }
}
```

```
struct Checkbox: View {
  @Binding var isOn: Bool

  var body: some View {

    Button(action: {
      self.isOn.toggle()
    }) {
      if isOn { OnView() }
      else { OffView() }
    }
  }
}
```

What Now?

What Now

1. Conference repo https://github.com/leamars/trySwift2019
2. UIKit translation to SwiftUI https://goshdarnswiftui.com
3. Facebook iOS Components: https://youtu.be/XhXC4SKOGfQ
4. Jetpack Compose: https://youtu.be/VsStyq4Lzxo
5. @hellosunschein on Twitter

Thank You!