SHORTCUT

Copenhagen Cocoa — November 26

Data sources in Combine

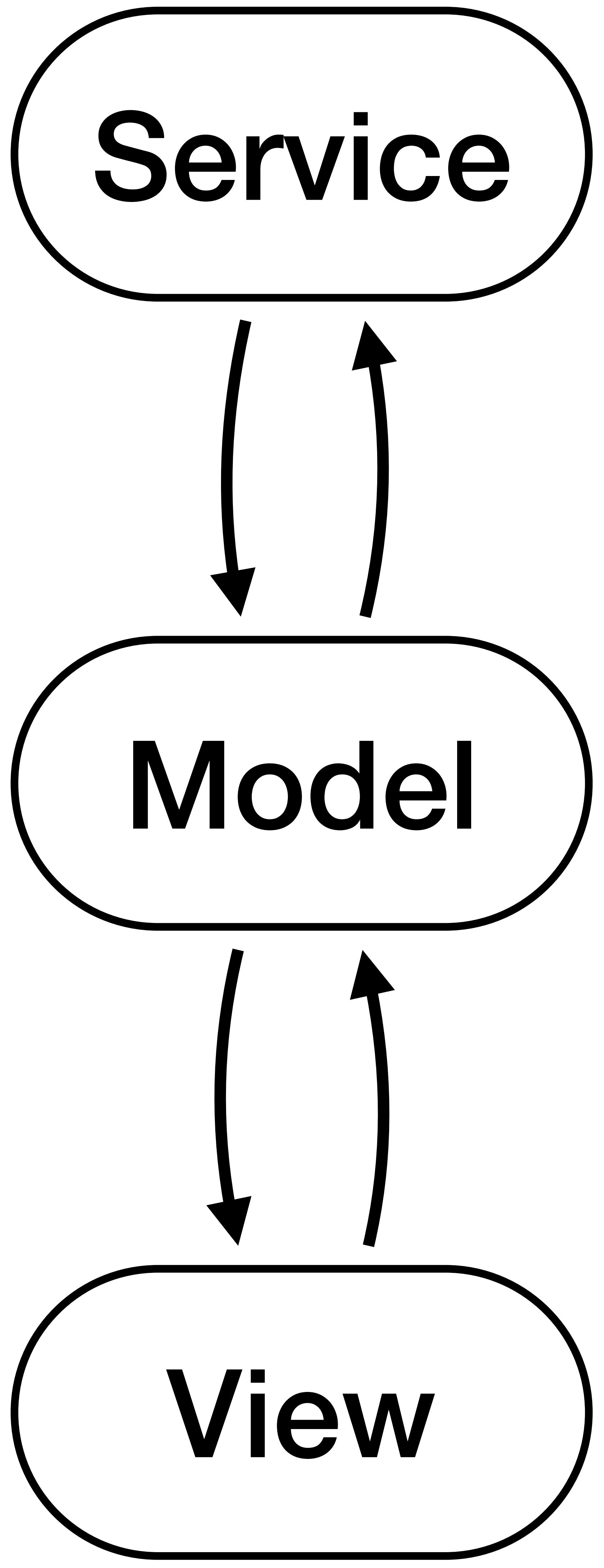
Michael Skiba

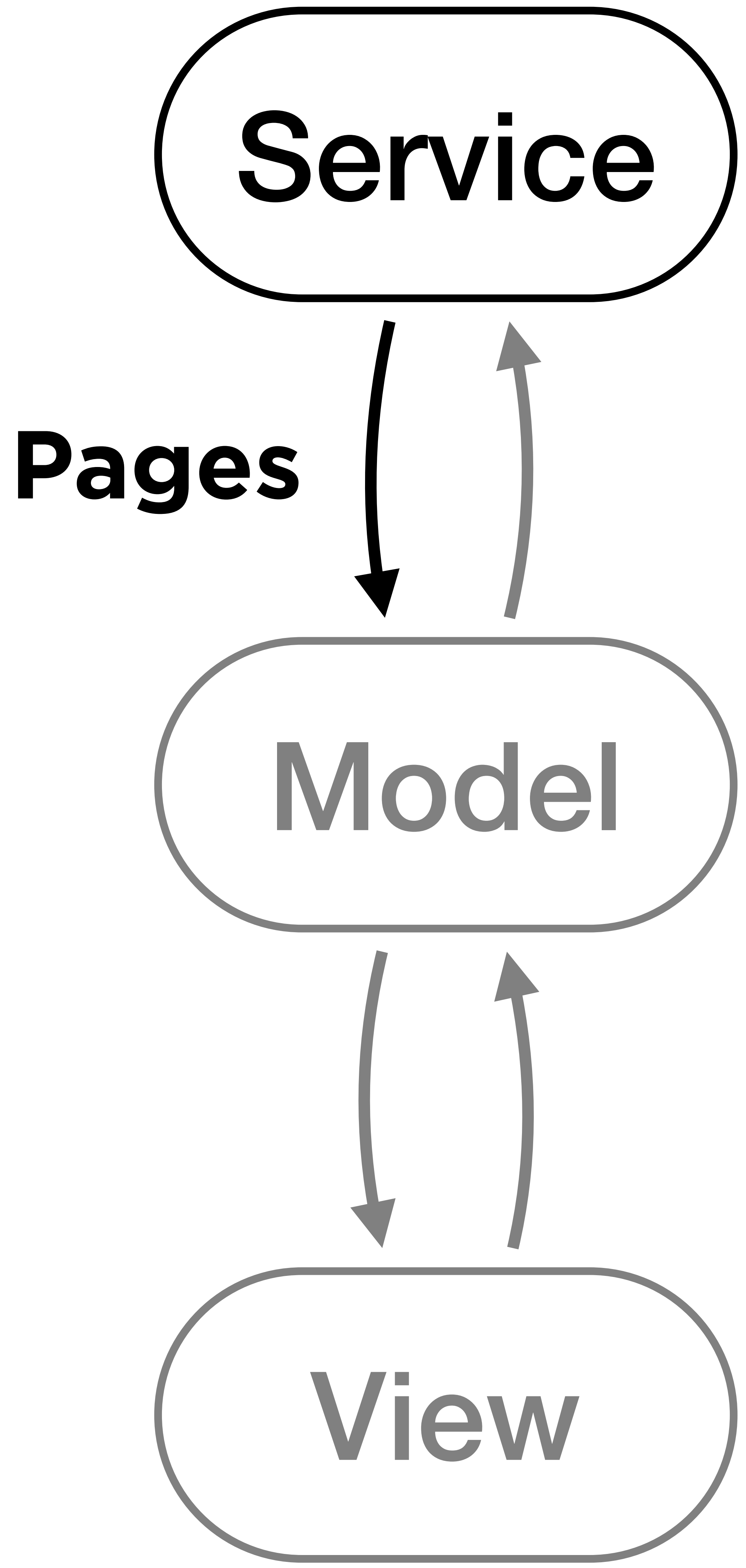
Data Sources

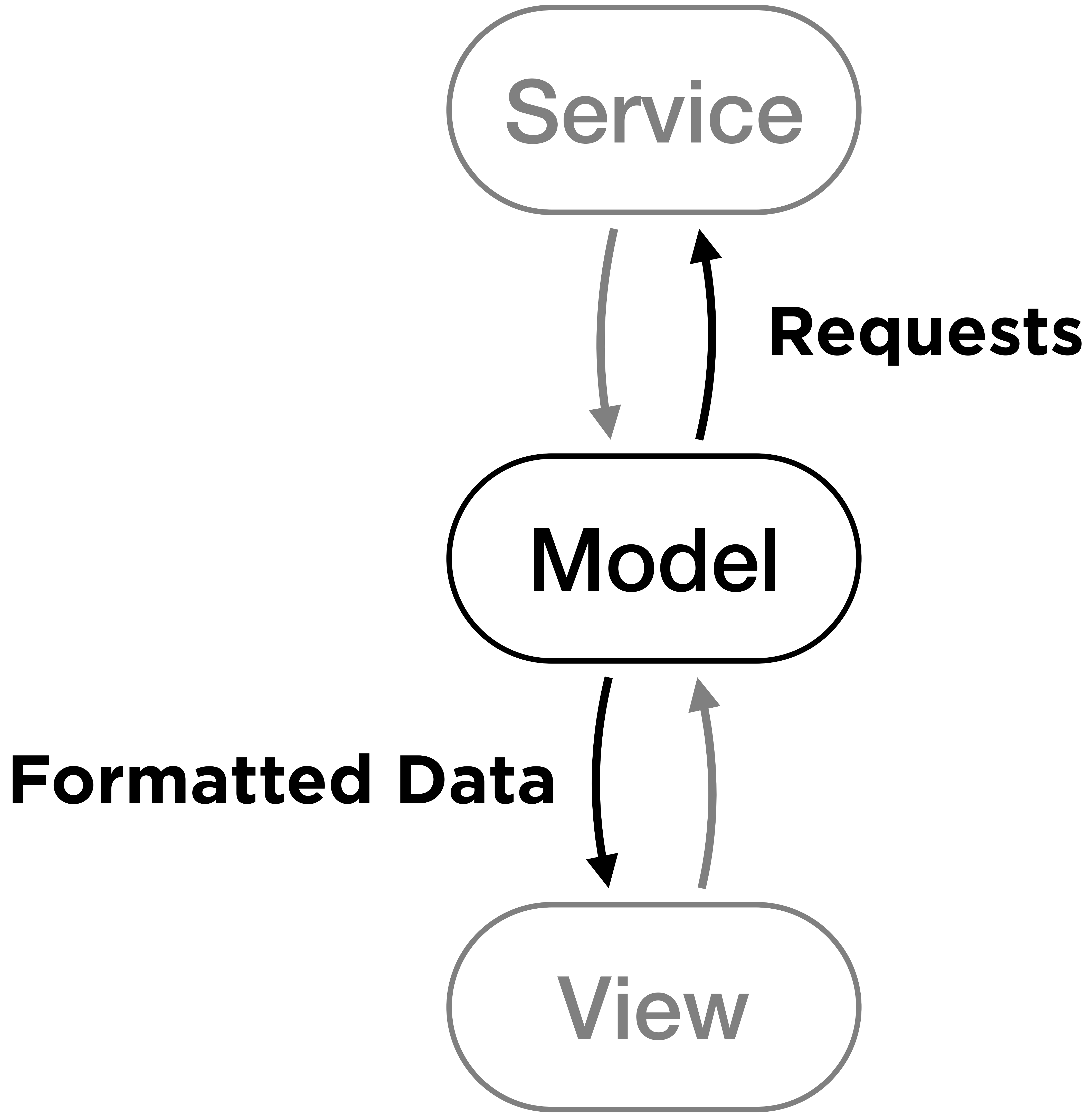
The plan:

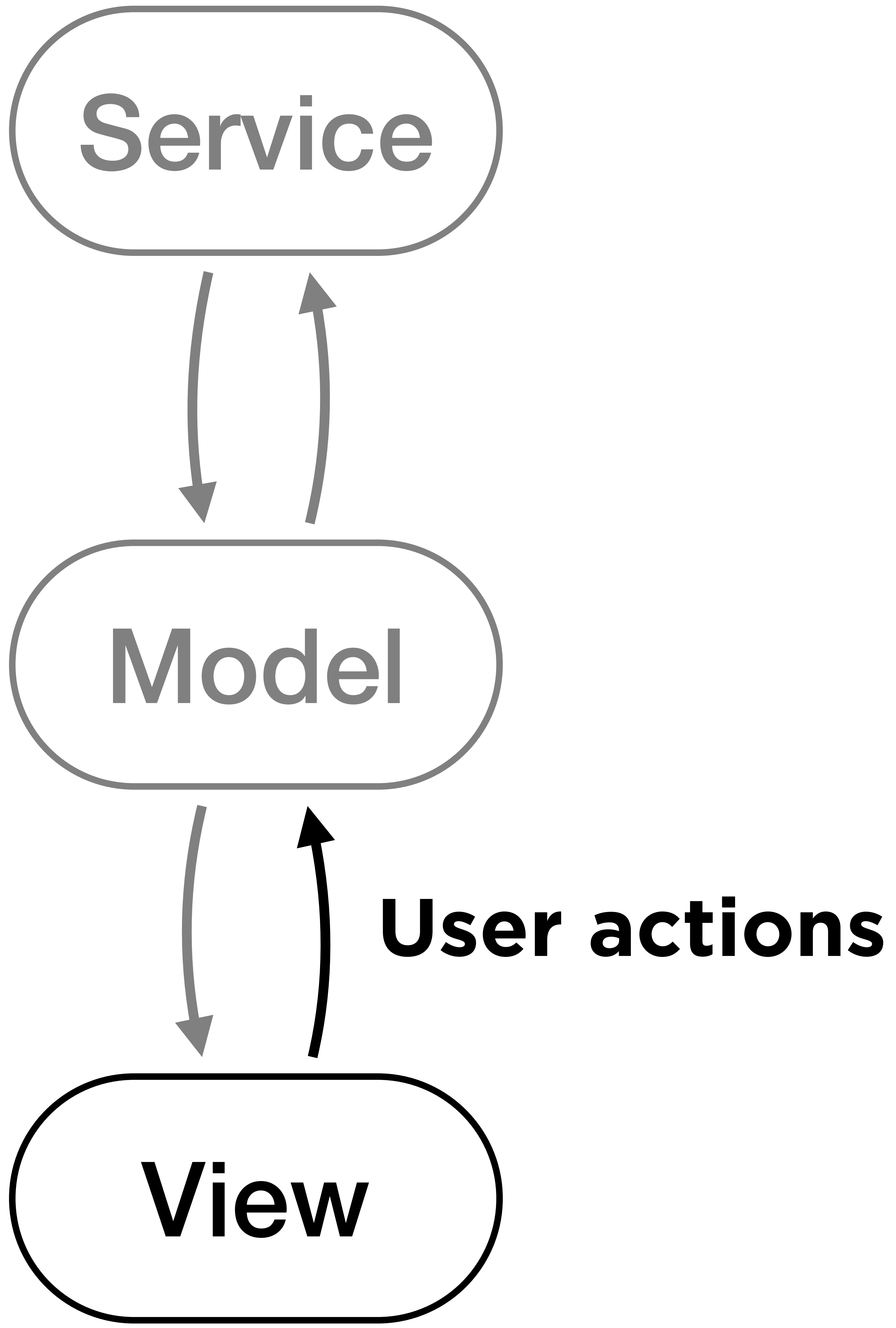
- The idea
- The service
- The model
- The views
- Demo
- Questions

The Idea


Service
Model
View


Service
Pages
Model
View


Service
Requests
Model
Formatted Data
View


Service
Model
User actions
View

The service

```swift
enum TypeService {
    static func info(type: PageType, page: Int) ->
                                AnyPublisher<Page<[Item]>, Error>
}

struct Page<Content> {
    let content: Content
    let hasMoreData: Bool
}

struct Item: Identifiable, Hashable {
    let id: UUID
    let text: String
}
```

```swift
enum PageType: CaseIterable, Identifiable, Hashable {
    case loading
    case error
    case empty
    case single
    case paged
    case stream

    var id: Self { self }
}
```

Publishers

- Just
- Fail
- Empty
- Future
- Deferred
- Record

Error

```
Fail<Page<[Item]>, Error>(error:
                        TypeService.ServiceError.generic)
                        .delay(for: .seconds(0.5),
                               scheduler: DispatchQueue.main)
                        .eraseToAnyPublisher()
```

Loading

```swift
Empty(completeImmediately: false,
      outputType: Page<[Item]>.self,
      failureType: Error.self)
     .eraseToAnyPublisher()
```

Single

```
Just(Page<[Item]>.simple)
    .setFailureType(to: Error.self)
    .delay(for: .seconds(0.5),
           scheduler: DispatchQueue.main)
    .eraseToAnyPublisher()
```

Paged

```swift
Just(Page<[Item]>.paged(page: page))
    .setFailureType(to: Error.self)
    .delay(for: .seconds(0.5),
           scheduler: DispatchQueue.main)
    .eraseToAnyPublisher()

extension Page where Content == [Item] {
    static func paged(page: Int) -> Page<[Item]> {
        let tail = [Item].neuromancer
                         .dropFirst(page * .pageSize)
        return Page(content: Array(tail.prefix(.pageSize)),
                    hasMoreData: tail.count > .pageSize)
    }
}
```

Stream

```
[Item].neuromancer.map { item in
    Page(content: [item], hasMoreData: false)
}
.publisher
.zip(Timer.publish(every: 0.2, on: .main, in: .default)
                   .autoconnect())
.map(\.0)
.setFailureType(to: Error.self)
.eraseToAnyPublisher()
```

The view model

```swift
class TypeViewModel : ObservableObject {
    private let type: PageType
    @Published private(set) var state: TypeViewModel.CurrentState
    @Published private(set) var items: [Item]
    @Published private var page
    init(type: PageType)

    func load()
}

extension TypeViewModel {
    enum CurrentState {
        case loading
        case loadedContent
        case hasMoreData
        case error
    }
}
```

Loading

```
let shareable = TypeService.info(type: type, page: page)
    .map(Result<Page<[Item]>, Error>.success)
    .catch {
        Just(Result<Page<[Item]>, Error>.failure($0))
    }
    .share()
```

Loading

```
shareable.map { result -> TypeViewModel.CurrentState in
    switch result {
    case .success(let page):
        return page.hasMoreData ?
            .hasMoreData : .loadedContent
    case .failure: return .error
    }
}
.prepend(.loading)
.assign(to: &$state)
```

Assign VS Assign

```
.assign(to: &$state)

.assign(to: \.state, on: self)
```

Loading

```
shareable.map { result -> [Item] in
    switch result {
    case .success(let page): return page.content
    case .failure: return []
    }
}
.scan(items) { items, pageItems in
    items + pageItems
}
.assign(to: &$items)
```

Loading

```
shareable.map { result in
    switch result {
    case .failure: return 0
    case .success: return 1
    }
}
.scan(page) { page, advance in
    page + advance
}
.assign(to: &$page)
```

Loading

```swift
shareable.map { result in
    switch result {
    case .failure: return 0
    case .success: return 1
    }
}
.scan(page) { page, advance in
    page + advance
}
.assign(to: &$page)
```

The views

Root View

```swift
struct RootView: View {
    @State private var type: PageType = .loading

    var body: some View {
        VStack {
            TypePicker(type: $type)
            TypeView(viewModel: TypeViewModel(type: type))
        }
        .padding()
    }
}
```

Type Picker

```swift
struct TypePicker: View {
    @Binding var type: PageType

    var body: some View {
        Picker(selection: $type, label: EmptyView()) {
            ForEach(PageType.allCases) { type in
                Text(type.description)
            }
        }
        .pickerStyle(SegmentedPickerStyle())
    }
}
```

Type View

```swift
struct TypeView: View {
    @ObservedObject private(set) var viewModel: TypeViewModel

    var body: some View {
        ScrollView {
            LazyVStack(alignment: .center) {
                ...
            }
        }
        .frame(minWidth: 200, minHeight: 200)
    }
}
```

Type View

```
LazyVStack(alignment: .center) {
    itemsView
    …
}

var itemsView: some View {
    ForEach(viewModel.items) { item in
        Text(item.text)
    }
    .frame(maxWidth: .infinity, alignment: .leading)
}
```

Type View

```
LazyVStack(alignment: .center) {
    …
    switch viewModel.state {
    case .error:
         errorView
    …
    }
}

var errorView: some View {
    VStack {
        Text("An error has occurred")
        Button("Retry", action: viewModel.load)
    }
}
```

Type View

```
LazyVStack(alignment: .center) {
    …
    switch viewModel.state {
    …
    case .loading:
        ProgressView()
    case .hasMoreData:
        ProgressView()
            .onAppear(perform: viewModel.load)
    …
    }
}
```

Type View

```
LazyVStack(alignment: .center) {
    …
    switch viewModel.state {
    …
    case .loadedContent where viewModel.items.isEmpty:
        Text("Nothing to see here")
    case .loadedContent:
        EmptyView()
    }
}
```

Demo

Conclusions

- 241 lines of Swift (including swiftUI previews)
- Opportunities for Generics
- All data is routed and transformed within publishers
- Publishers can be used track a single event, or a series of events interchangeably
- Code is available in: https://github.com/shortcut/combine-in-practice

Questions?