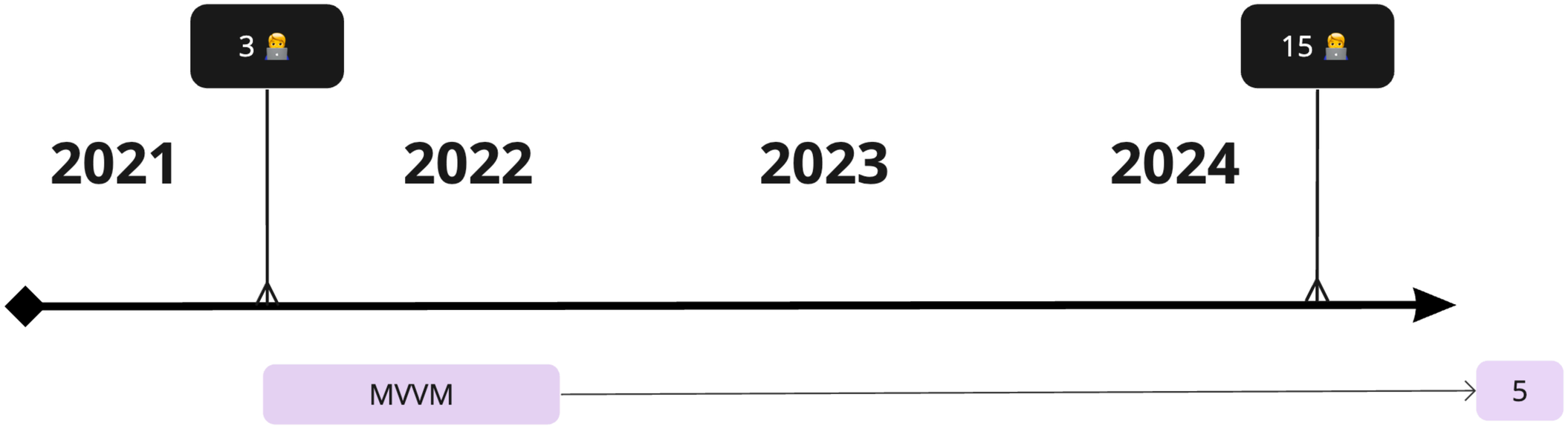
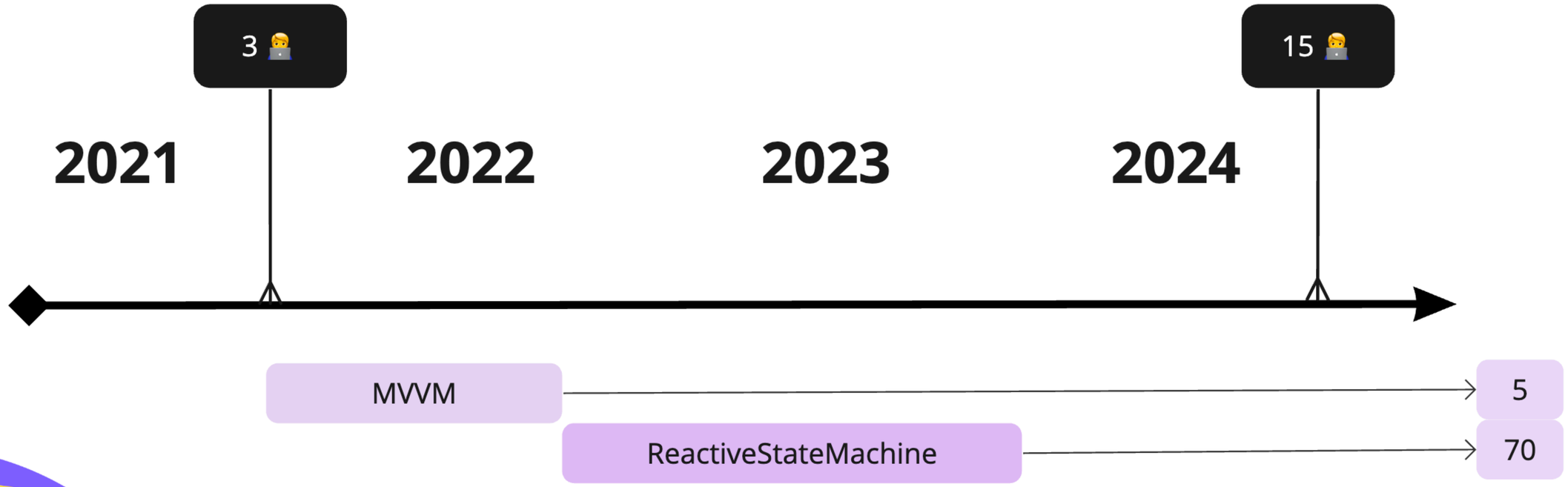
Views are a function of state, not a sequence of events (c) WWDC19

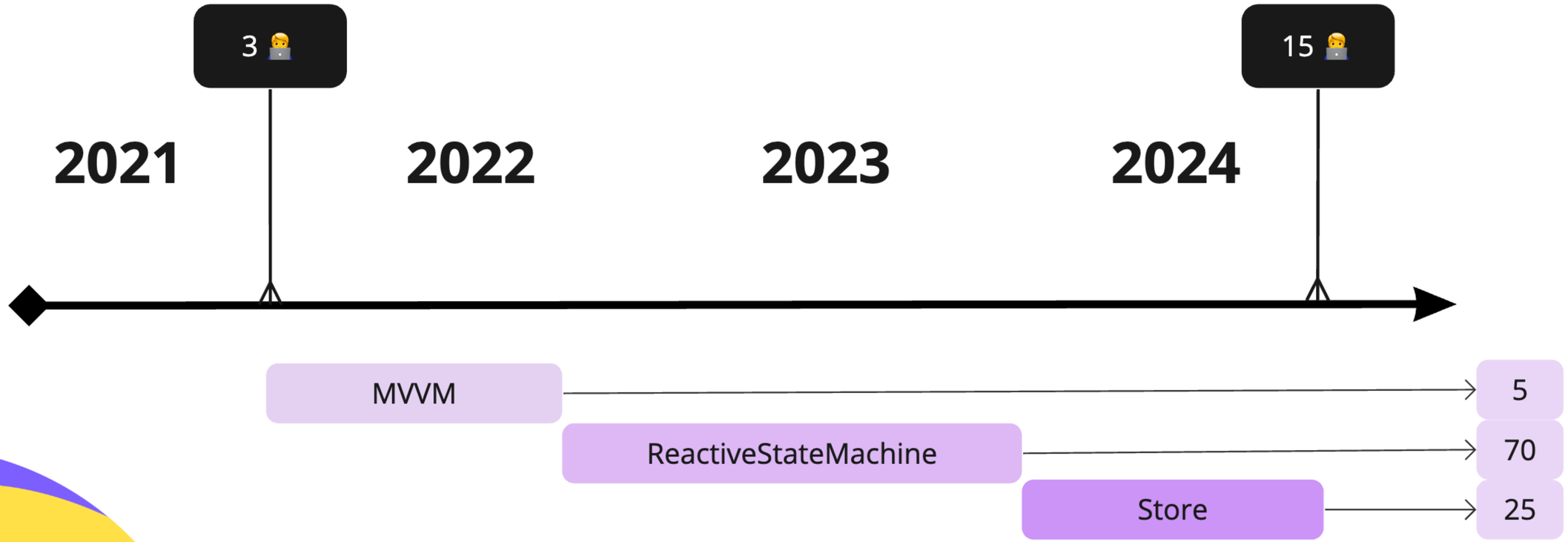
Views are a function of **state**, not a sequence of events (c) WWDC19

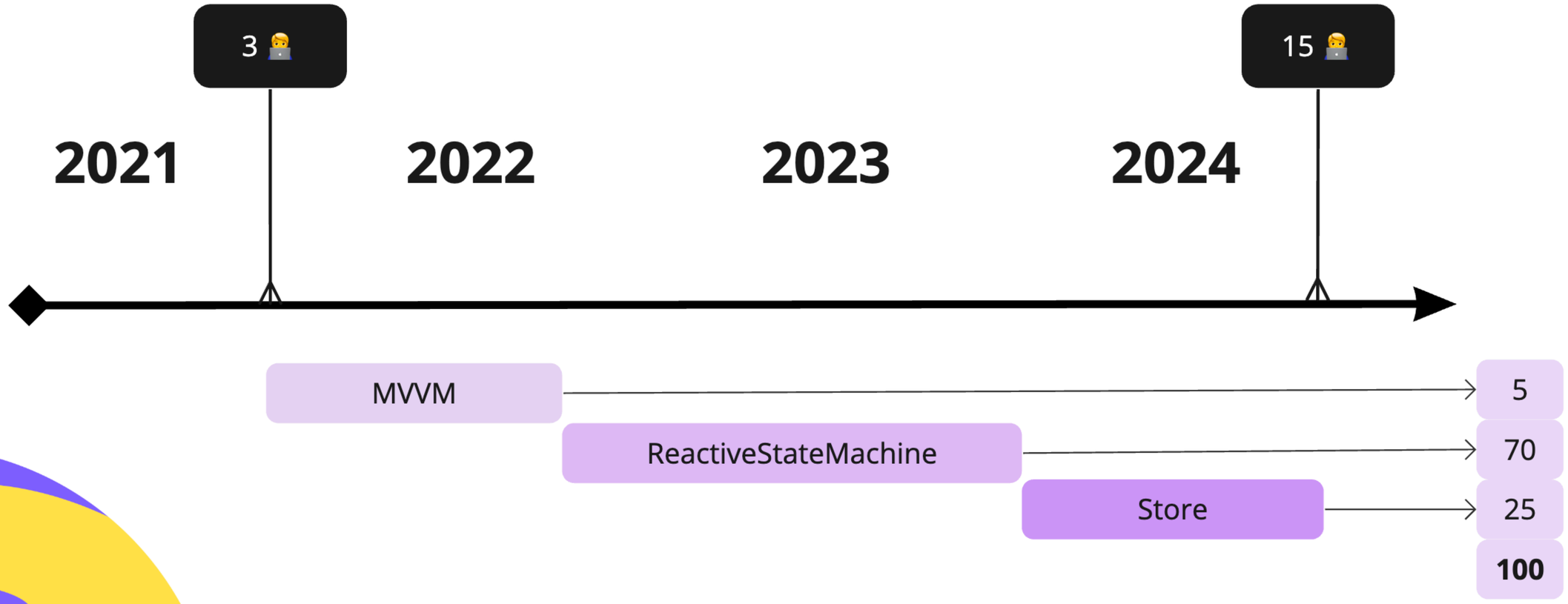
2021 2022 2023 2024

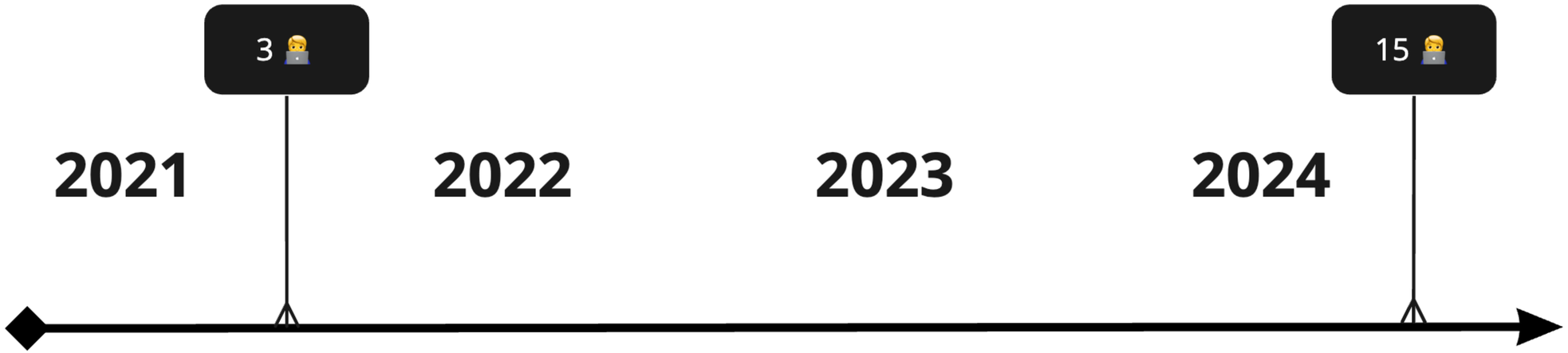

3 👨‍💻
15 👨‍💻
2021
2022
2023
2024


3 👨‍💻
15 👨‍💻
2021
2022
2023
2024
MVVM
5


3 👨‍💻
15 👨‍💻
2021
2022
2023
2024
MVVM
5
ReactiveStateMachine
70


3 👨‍💻
15 👨‍💻
2021
2022
2023
2024
MVVM
5
ReactiveStateMachine
70
Store
25


3 👨‍💻
15 👨‍💻
2021
2022
2023
2024
MVVM
ReactiveStateMachine
Store
5
70
25
100



MVVM → 5

ReactiveStateMachine → 70

Store → 25

100

250k LOC

@State

@State

ObservableObject

@State

👌

@Observable

ObservableObject

🤔

ObservableObject здорового человека

ObservableObject здорового человека

 Согласованность состояния

 Тестируемость

 Удобство отладки

 Потокобезопасность

ObservableObject здорового человека

 Согласованность состояния

 Тестируемость

 Удобство отладки

 Потокобезопасность

ObservableObject здорового человека

⚙️ Согласованность состояния

🧪 Тестируемость

🐞 Удобство отладки

🛡️ Потокобезопасность

ObservableObject здорового человека

⚙️ Согласованность состояния

🧪 Тестируемость

🐞 Удобство отладки

🛡️ Потокобезопасность

ObservableObject здорового человека

⚙️ Согласованность состояния

🧪 Тестируемость

🐞 Удобство отладки

🛡️ Потокобезопасность

⚛️ Декомпозиция

🏭 Унификация

ObservableObject здорового человека

⚙️ Согласованность состояния

🧪 Тестируемость

🐞 Удобство отладки

🛡️ Потокобезопасность

⚛️ Декомпозиция

🏭 Унификация

ObservableObject здорового человека

⚙️ Согласованность состояния

🧪 Тестируемость

🐞 Удобство отладки

 Потокобезопасность

⚛️ Декомпозиция

🏭 **Унификация**

Ниф-Ниф

Ниф-Ниф

MVVM*

Ниф-Ниф

Нуф-Нуф

MVVM*

Ниф-Ниф

MVVM*

Нуф-Нуф

RSM*

Ниф-Ниф

MVVM*

Нуф-Нуф

RSM*

Наф-Наф

Ниф-Ниф	Нуф-Нуф	Наф-Наф
		
MVVM*	RSM*	Store*

MVVM

View


View
ViewModel


View
ViewModel

```
final class ViewModel: ObservableObject {
  @Published private(set) var state: State
  private let dependency: Dependency

  func onAppear() {...}
  func onRefresh() {...}
  func promptUpdated(_ prompt: String) {...}
}
```

```
final class ViewModel: ObservableObject {
  @Published private(set) var state: State
  private let dependency: Dependency

  func onAppear() {...}
  func onRefresh() {...}
  func promptUpdated(_ prompt: String) {...}
}
```

```
final class ViewModel: ObservableObject {
  @Published private(set) var state: State
  private let dependency: Dependency

  func onAppear() {...}
  func onRefresh() {...}
  func promptUpdated(_ prompt: String) {...}
}
```

```
final class ViewModel: ObservableObject {
  @Published private(set) var state: State
  private let dependency: Dependency

  func onAppear() {...}
  func onRefresh() {...}
  func promptUpdated(_ prompt: String) {...}
}
```

Низкий порог входа

Гибкость

Низкий порог входа

Гибкость

Низкий порог входа

Гибкость

Низкий порог входа

Гибкость

Непредсказуемое состояние

Компот из эффектов и мутаций

Страдает тестируемость

Может обернуться в зоопарк

Низкий порог входа

Гибкость

Непредсказуемое состояние

Компот из эффектов и мутаций

Страдает тестируемость

Может обернуться в зоопарк

Низкий порог входа

Гибкость

Непредсказуемое состояние

Компот из эффектов и мутаций

Страдает тестируемость

Может обернуться в зоопарк

Низкий порог входа

Гибкость

Непредсказуемое состояние

Компот из эффектов и мутаций

Страдает тестируемость

Может обернуться в зоопарк

Низкий порог входа

Гибкость

Непредсказуемое состояние

Компот из эффектов и мутаций

Страдает тестируемость

Может обернуться в зоопарк

Непредсказуемое состояние

Finite State Machine

Непредсказуемое состояние

Finite State Machine

Непредсказуемое состояние

Компот из эффектов и мутаций

Finite State Machine

Реактивная обработка эффектов

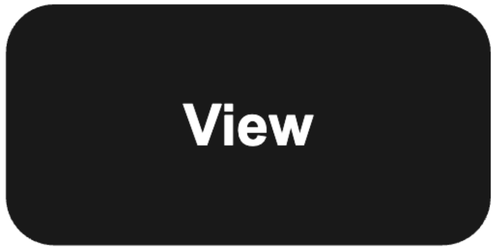
Непредсказуемое состояние

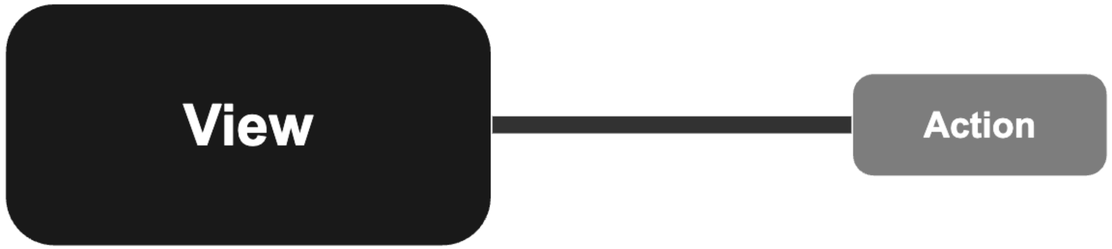
Компот из эффектов и мутаций

Finite State Machine

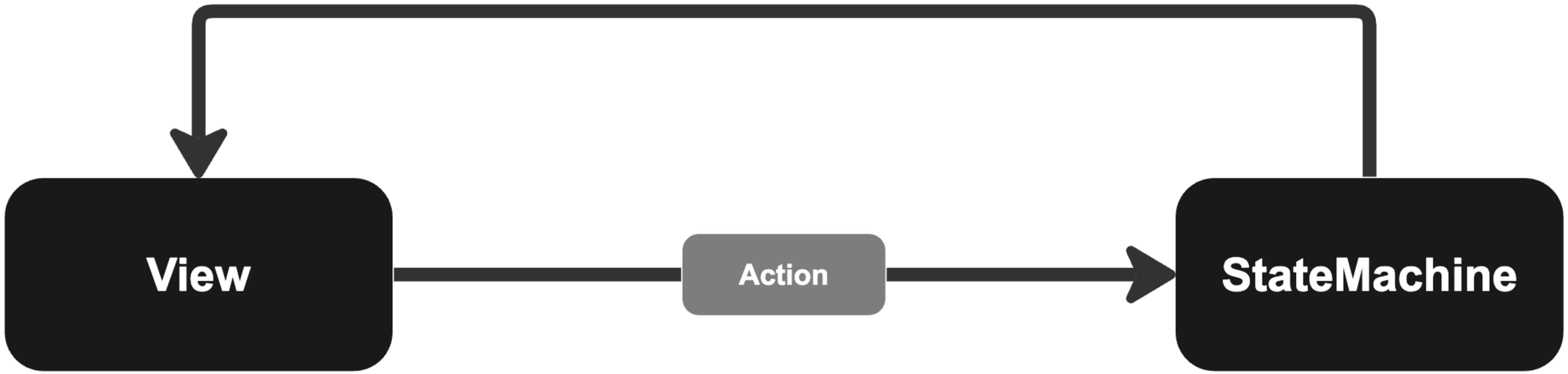
Реактивная обработка эффектов

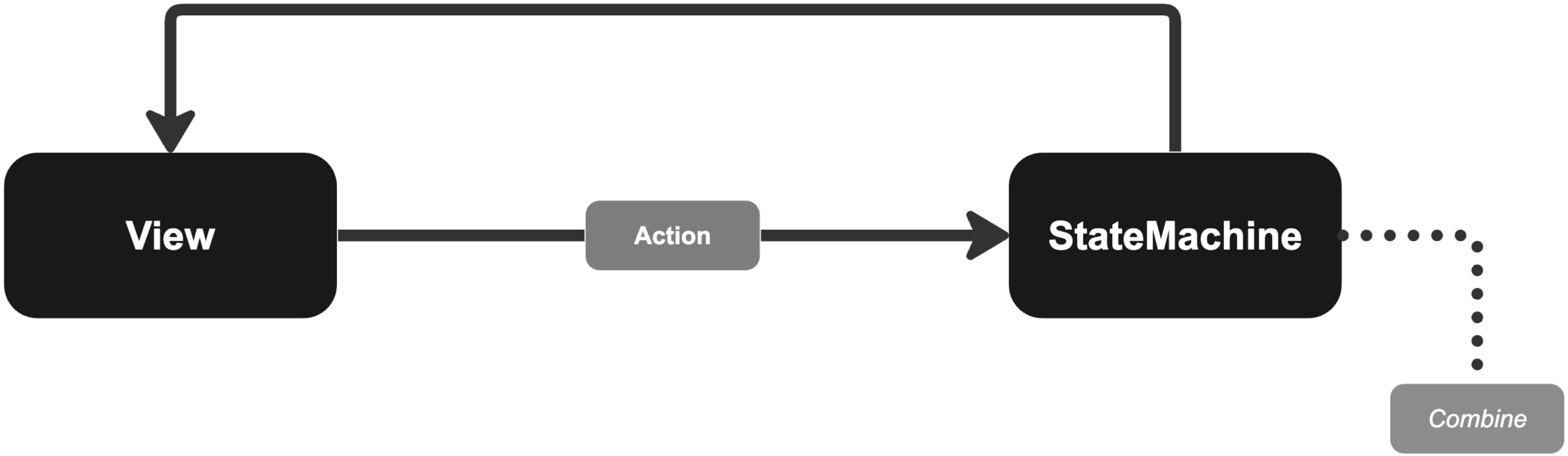
Reactive State Machine

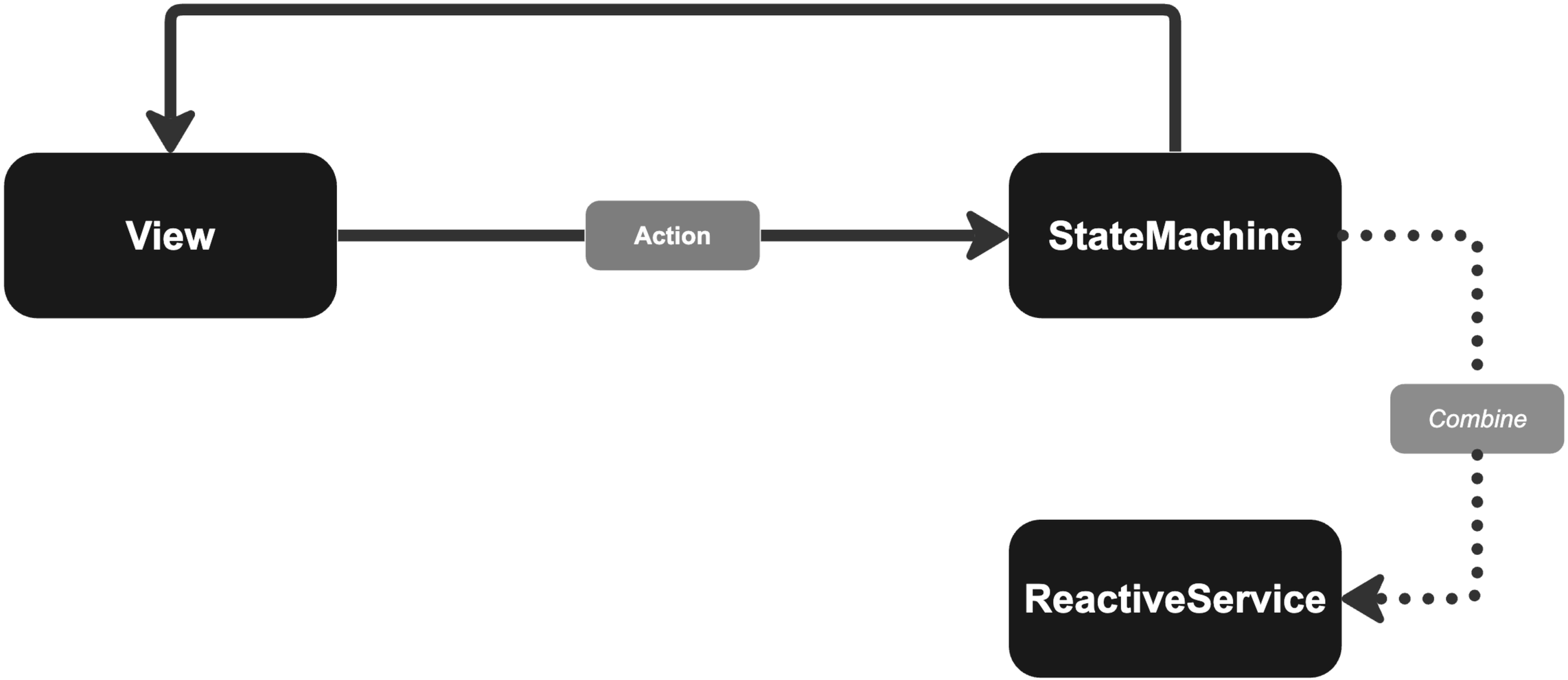

View

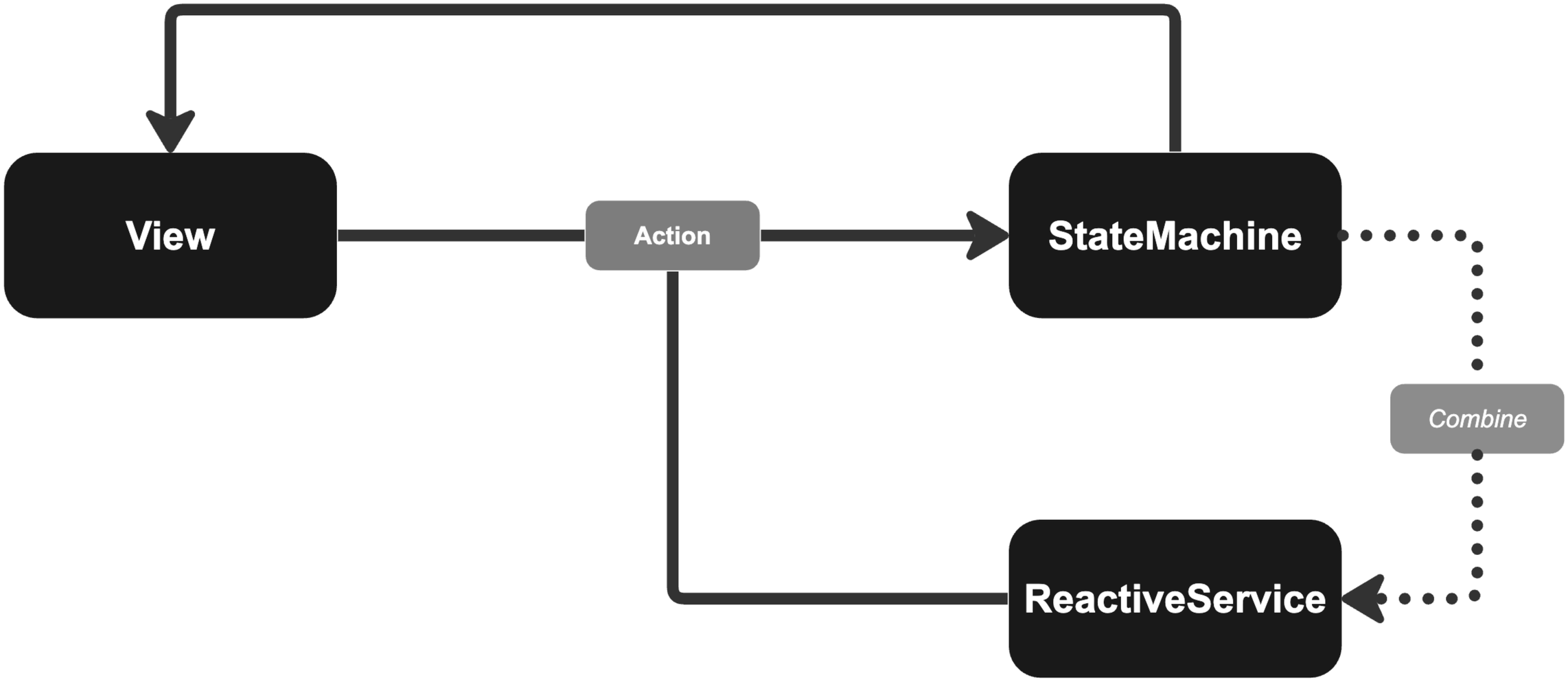

View
Action


View
Action
StateMachine


View
Action
StateMachine


View
Action
StateMachine
Combine


View
Action
StateMachine
Combine
ReactiveService


View
Action
StateMachine
Combine
ReactiveService

```
final class StateMachine: ObservableObject {
  @Published private(set) var state: State

  let fetchRequestSignal = PassthroughSubject<Void, Never>()
  let promptUpdatedSignal = PassthroughSubject<String, Never>()

  enum Action {
    case onAppear
    case onRefresh
    case promptUpdated(String)
  }

  func execute(action: Action) async {
    switch action { ... }
  }
}
```

```
final class StateMachine: ObservableObject {
  @Published private(set) var state: State

  let fetchRequestSignal = PassthroughSubject<Void, Never>()
  let promptUpdatedSignal = PassthroughSubject<String, Never>()

  enum Action {
    case onAppear
    case onRefresh
    case promptUpdated(String)
  }

  func execute(action: Action) async {
    switch action { ... }
  }
}
```

```
final class StateMachine: ObservableObject {
  @Published private(set) var state: State

  let fetchRequestSignal = PassthroughSubject<Void, Never>()
  let promptUpdatedSignal = PassthroughSubject<String, Never>()

  enum Action {
    case onAppear
    case onRefresh
    case promptUpdated(String)
  }

  func execute(action: Action) async {
    switch action { ... }
  }
}
```

```
final class StateMachine: ObservableObject {
  @Published private(set) var state: State

  let fetchRequestSignal = PassthroughSubject<Void, Never>()
  let promptUpdatedSignal = PassthroughSubject<String, Never>()

  enum Action {
    case onAppear
    case onRefresh
    case promptUpdated(String)
  }

  func execute(action: Action) async {
    switch action { ... }
  }
}
```

Гибкость

Изоляция эффектов

Лаконичность FSM

Гибкость

Изоляция эффектов

Лаконичность FSM

Гибкость

Изоляция эффектов

Лаконичность FSM

Гибкость

Изоляция эффектов

Лаконичность FSM

Гибкость

Изоляция эффектов

Лаконичность FSM

Опасная реактивность

Масса бойлерплейта

Болезненная отладка

Может обернуться в зоопарк

Гибкость

Изоляция эффектов

Лаконичность FSM

Опасная реактивность

Масса бойлерплейта

Болезненная отладка

Может обернуться в зоопарк

Гибкость

Изоляция эффектов

Лаконичность FSM

Опасная реактивность

Масса бойлерплейта

Болезненная отладка

Может обернуться в зоопарк

Гибкость

Изоляция эффектов

Лаконичность FSM

Опасная реактивность

Масса бойлерплейта

Болезненная отладка

Может обернуться в зоопарк

Гибкость

Изоляция эффектов

Лаконичность FSM

Опасная реактивность

Масса бойлерплейта

Болезненная отладка

Может обернуться в зоопарк

Опасная реактивность

Старые добрые делегаты

Опасная реактивность

Старые добрые делегаты

Опасная реактивность

Масса бойлерплейта

Старые добрые делегаты

Progressive disclosure API

Опасная реактивность

Масса бойлерплейта

Старые добрые делегаты

Progressive disclosure API

Опасная реактивность

Масса бойлерплейта

Болезненная отладка

Старые добрые делегаты

Progressive disclosure API

Императивность

Опасная реактивность

Масса бойлерплейта

Болезненная отладка

Старые добрые делегаты

Progressive disclosure API

Императивность

Опасная реактивность

Масса бойлерплейта

Болезненная отладка

Может обернуться в зоопарк

Старые добрые делегаты

Progressive disclosure API

Императивность

Generic ObservableObject

Опасная реактивность

Масса бойлерплейта

Болезненная отладка

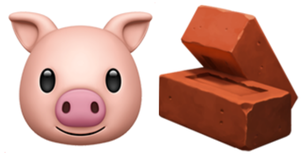
Может обернуться в зоопарк

Старые добрые делегаты

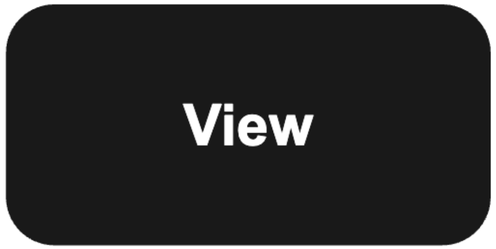
Progressive disclosure API

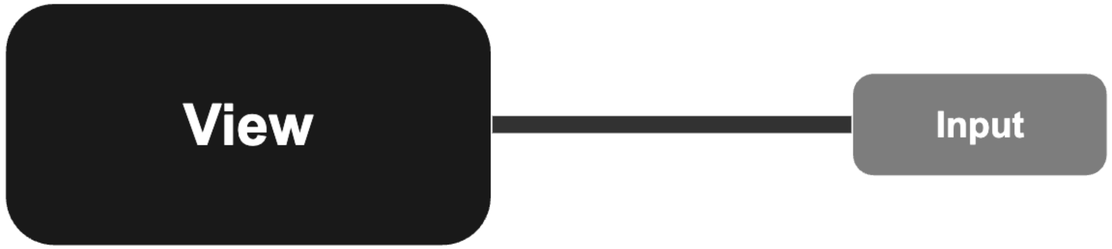
Императивность

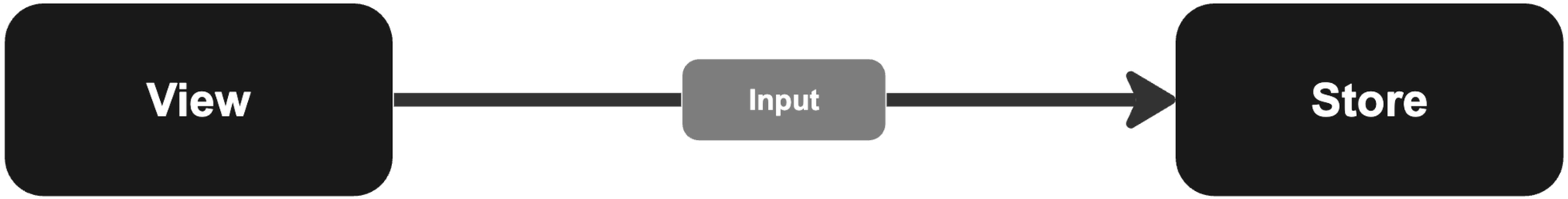
Generic ObservableObject

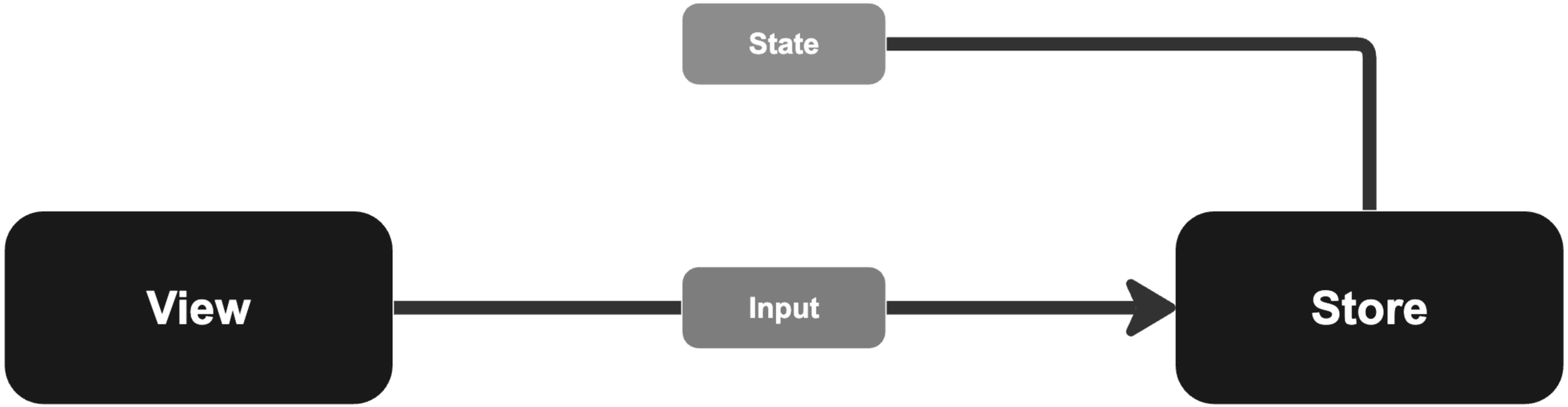


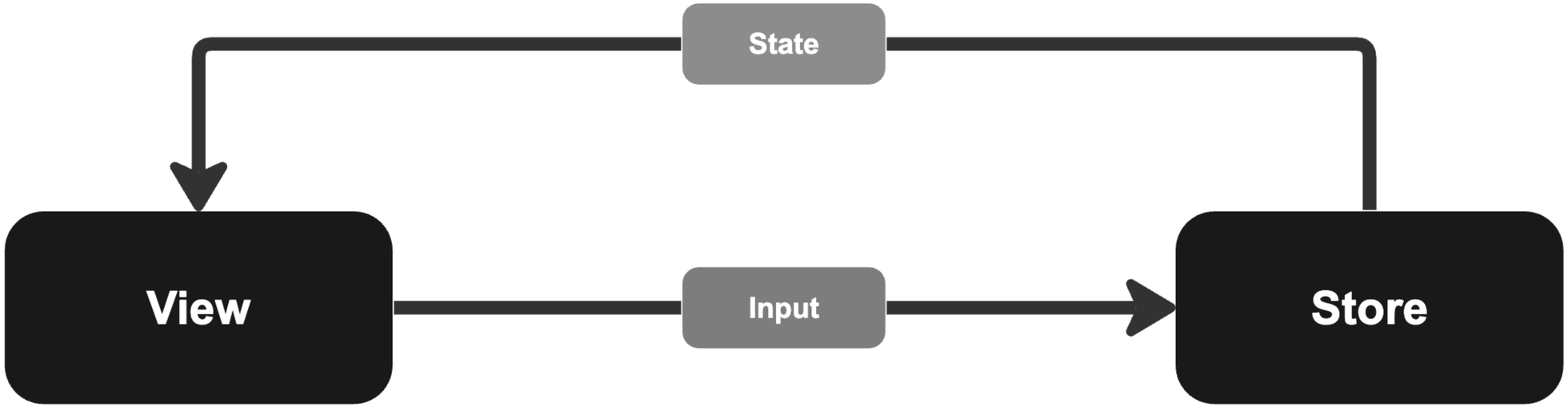
Store

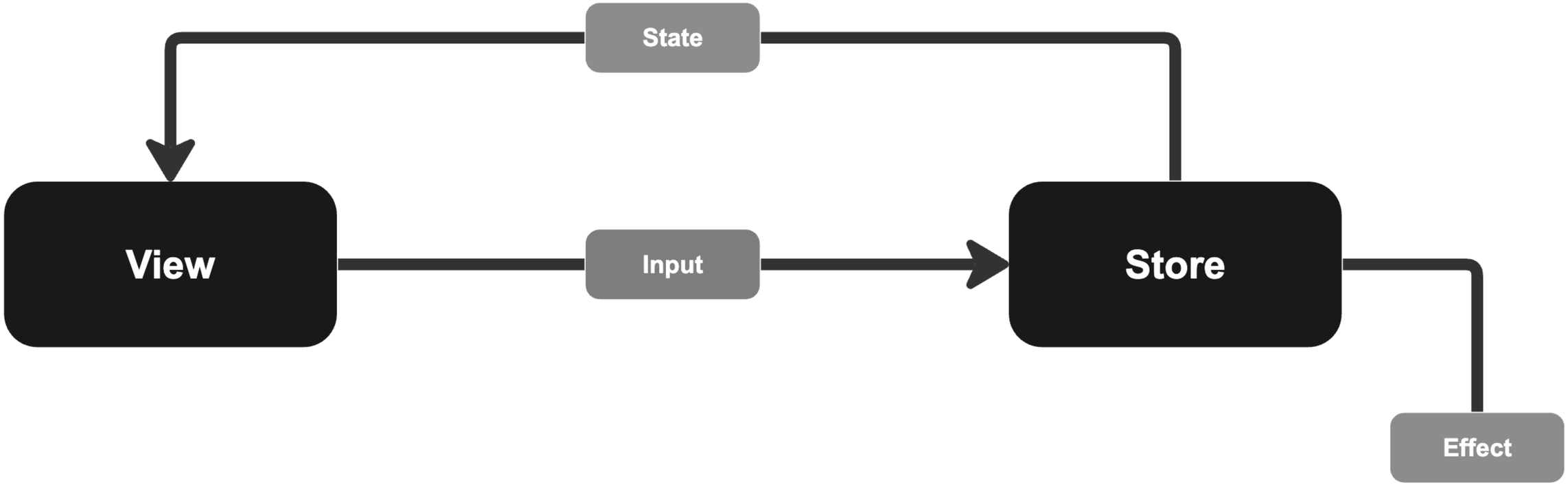

View

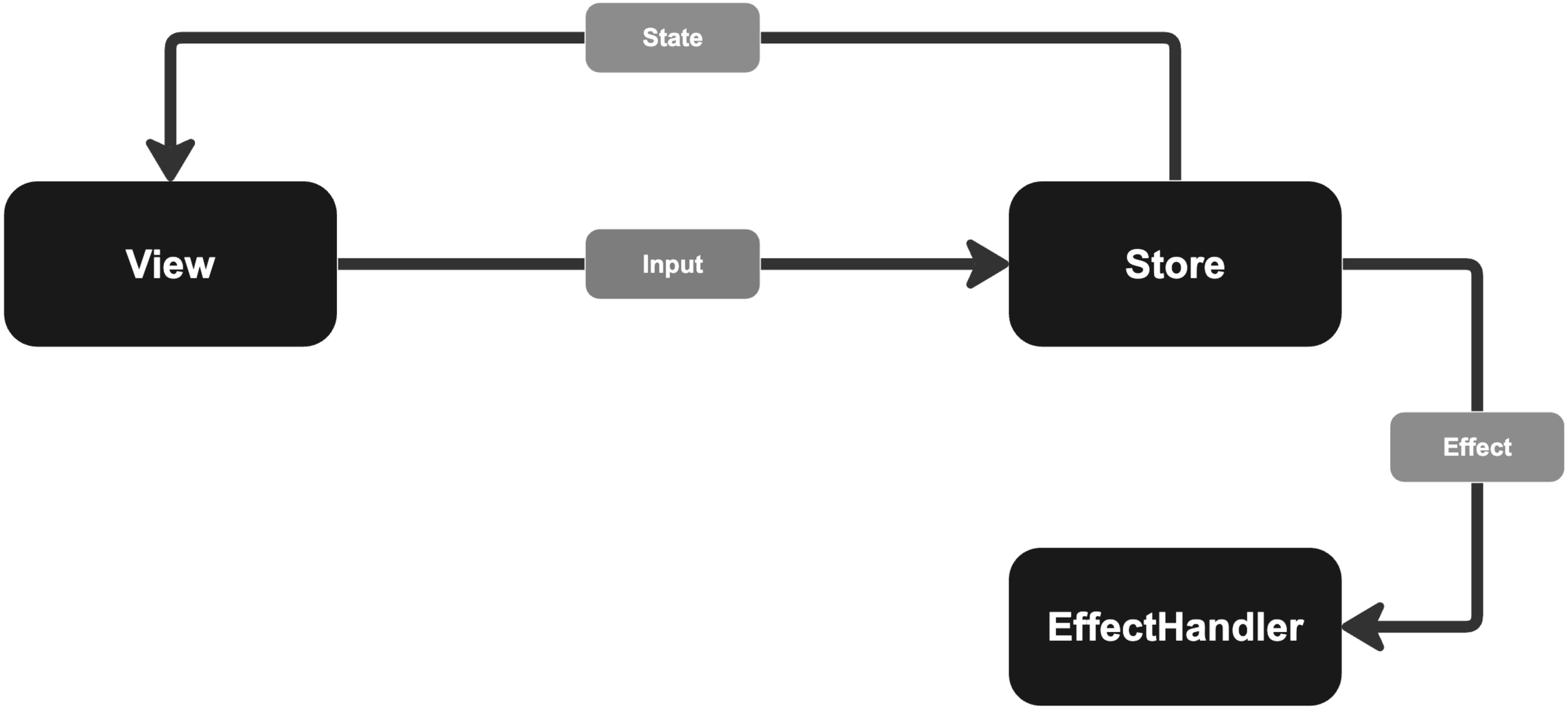

View
Input

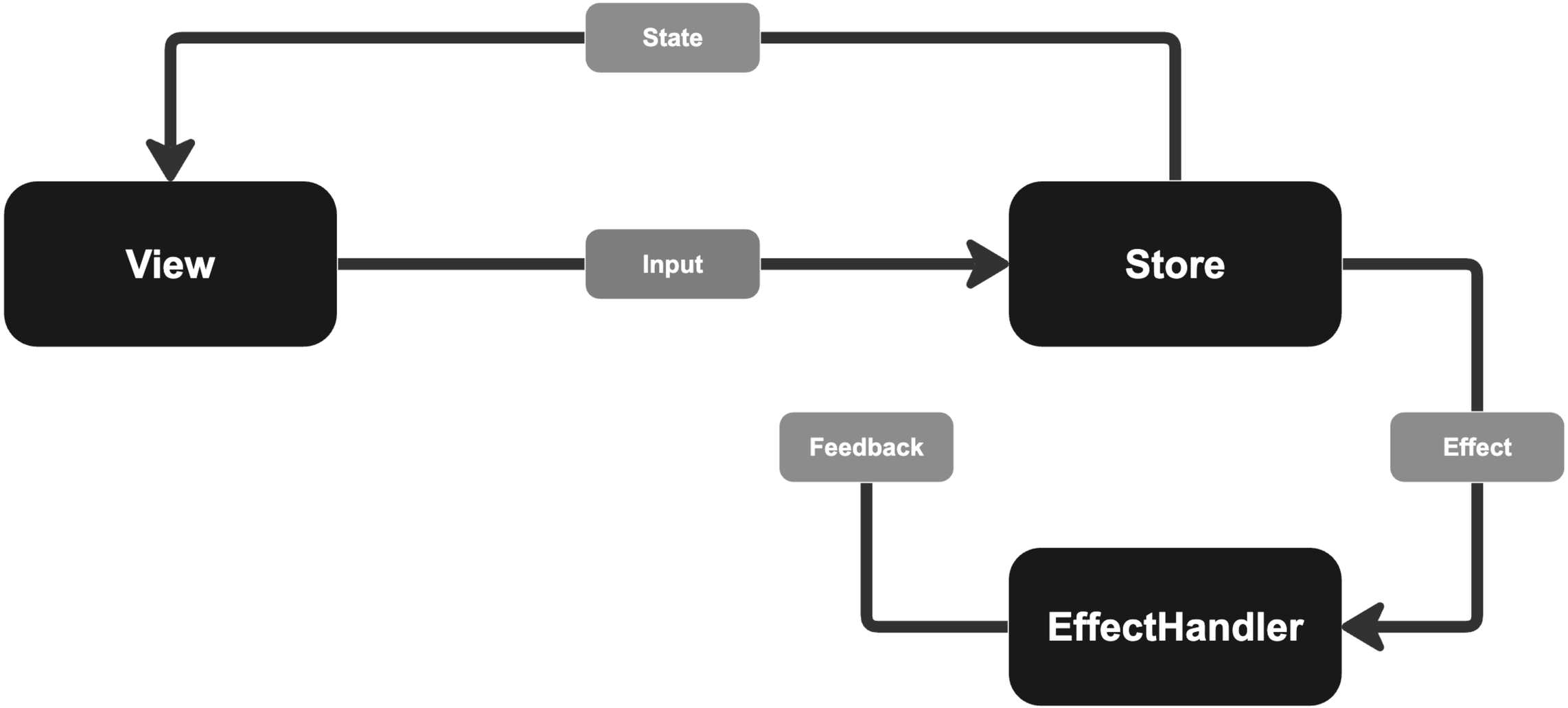

View
Input
Store

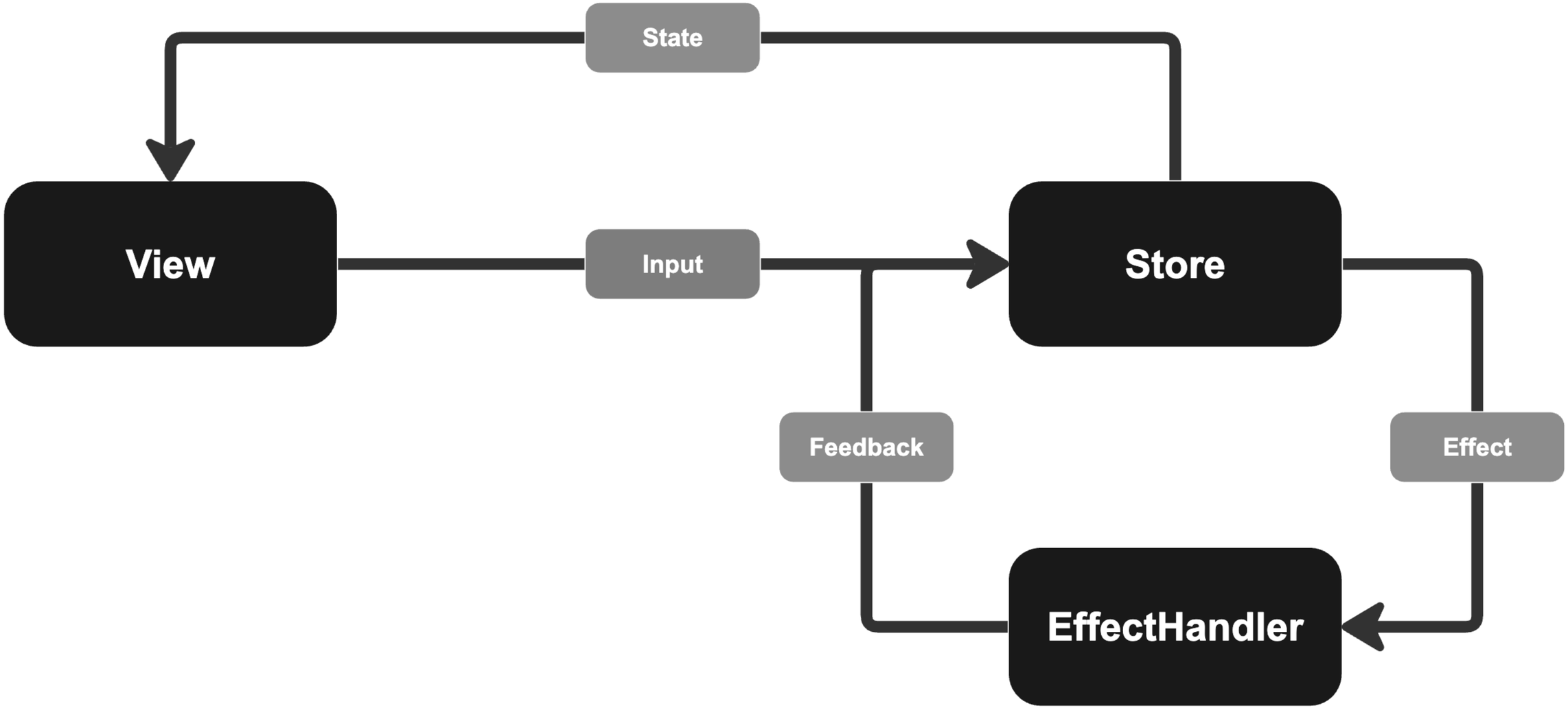

State
View
Input
Store

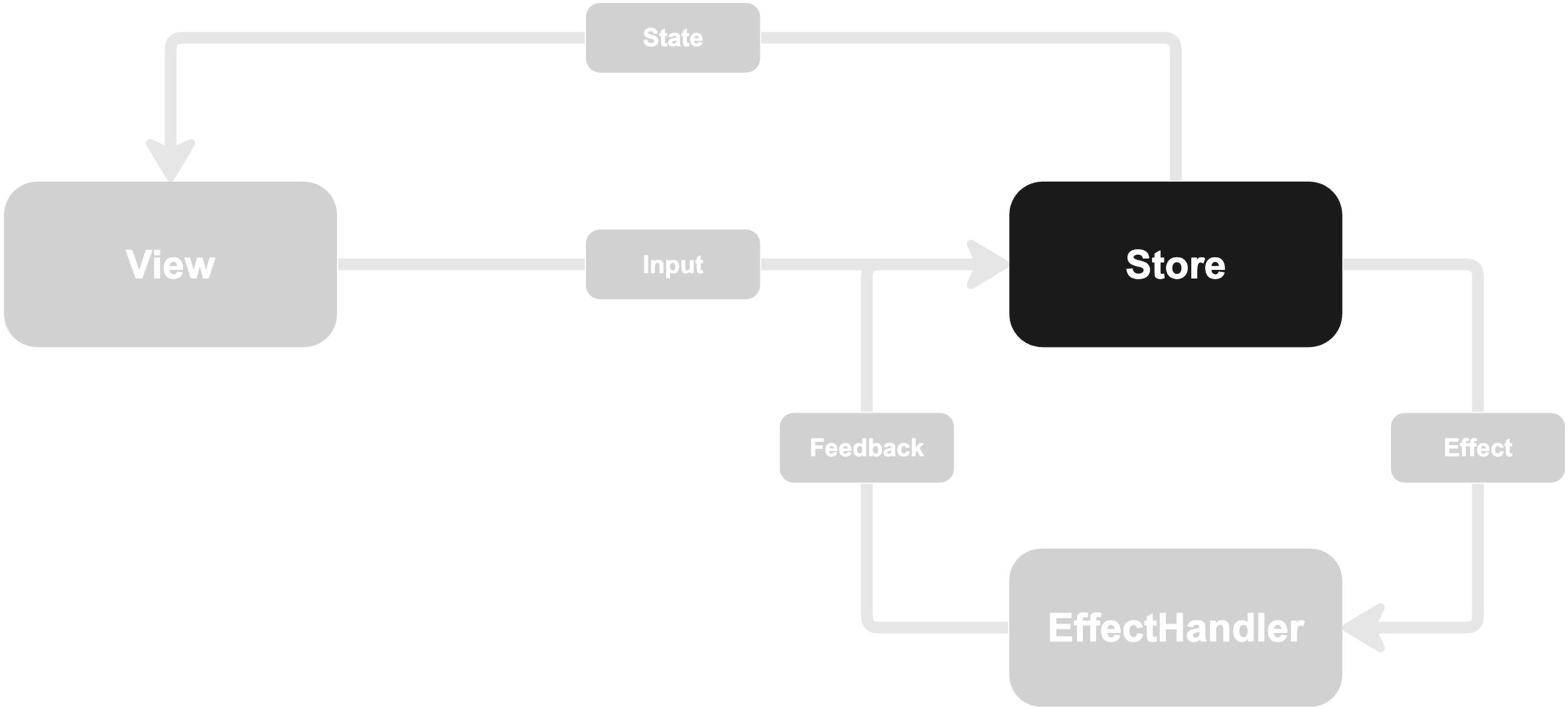

State
View
Input
Store

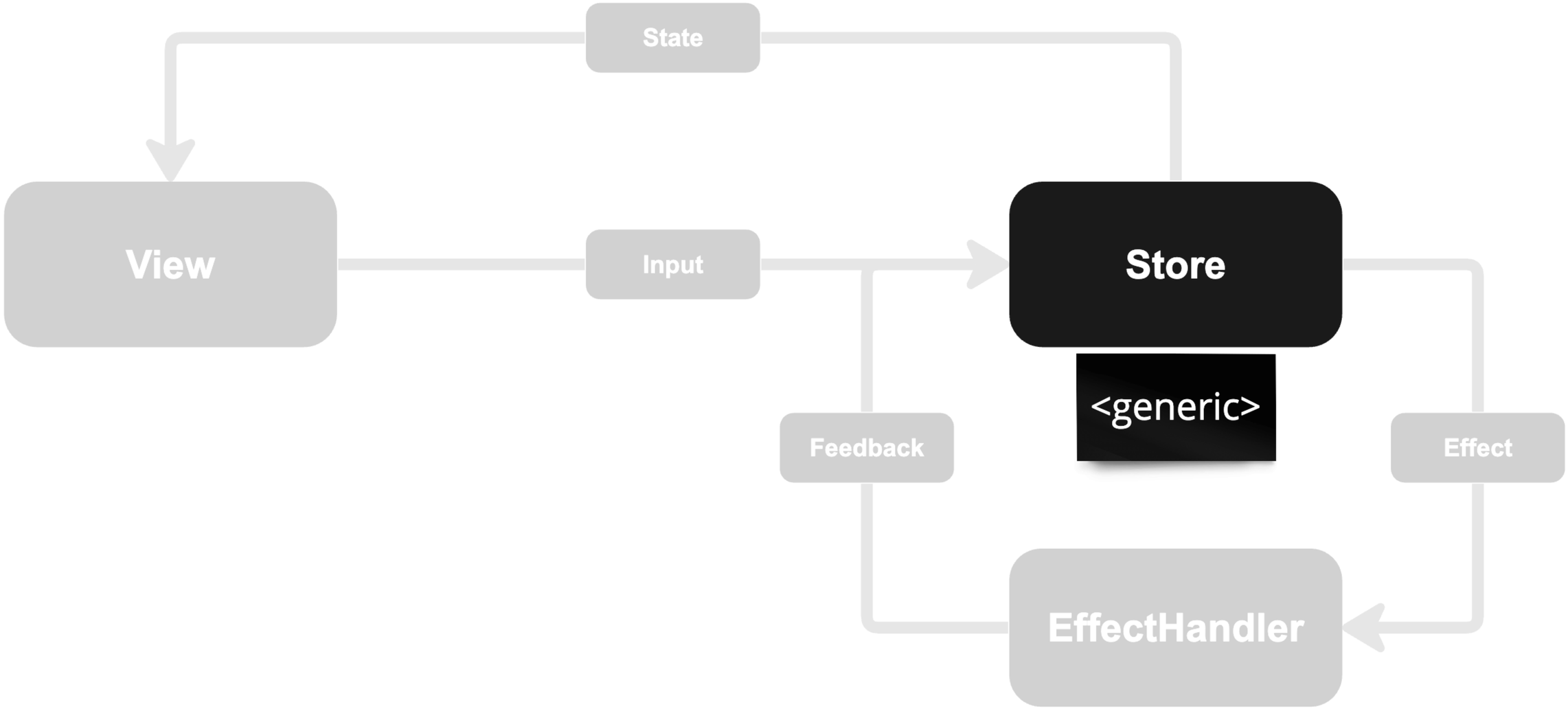

State
View
Input
Store
Effect


State
View
Input
Store
Effect
EffectHandler


State
View
Input
Store
Feedback
Effect
EffectHandler


State
View
Input
Store
Feedback
Effect
EffectHandler


State
View
Input
Store
Feedback
Effect
EffectHandler

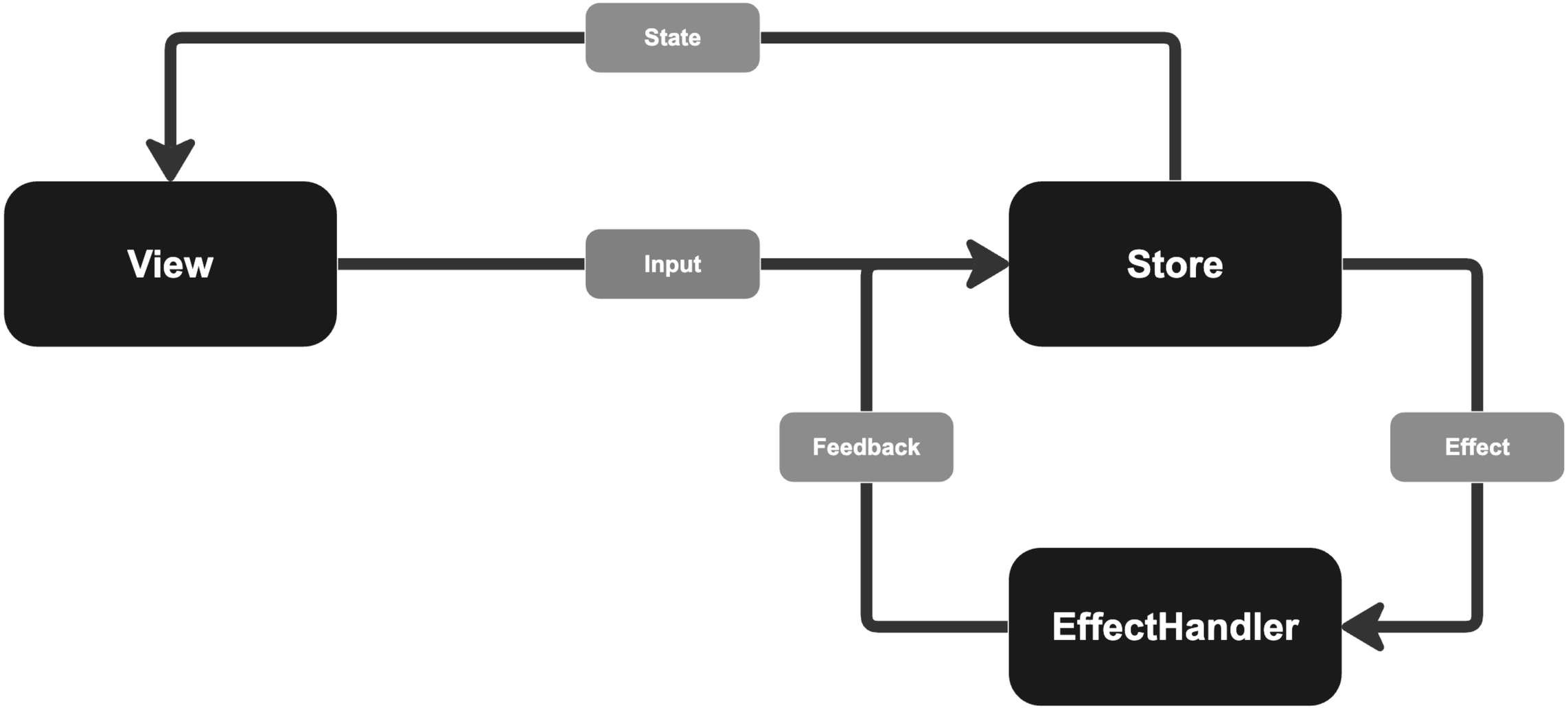

State
View
Input
Store
<generic>
Feedback
Effect
EffectHandler

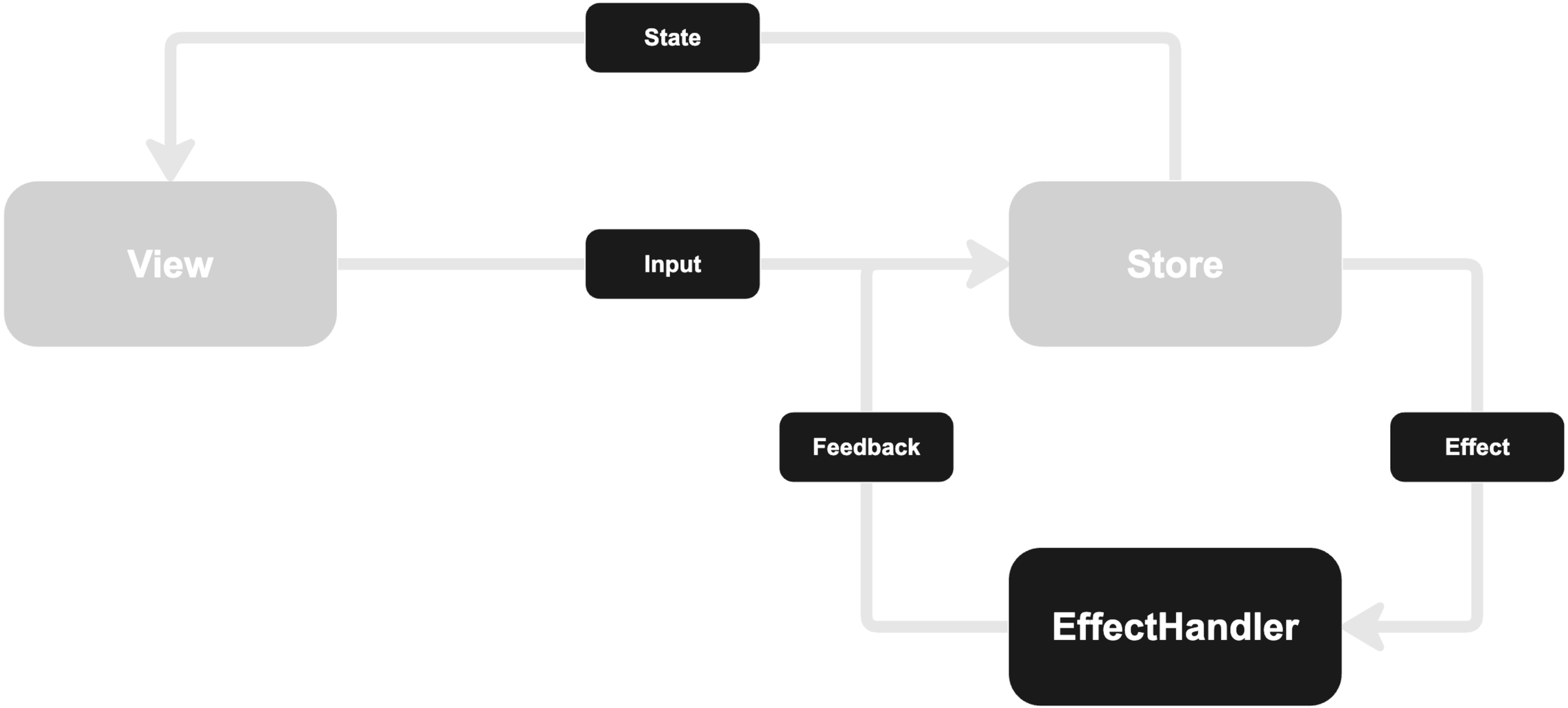
```
final class Store<S, EH>: ObservableObject where
  S: State,
  EH: EffectHandler<S> {
  @Published private(set) var state: S
  let effectHandler: EH

  func send(_ message: Message<S.Input, S.Feedback>) {
    var updatedState = state
    let effect = S.reduce(state: &updatedState, with: message)
    state = updatedState

    Task {
      if let effect,
         let feedback = await effectHandler.handle(effect: effect) {
        send(.feedback(feedback))
      }
    }
  }
}
```

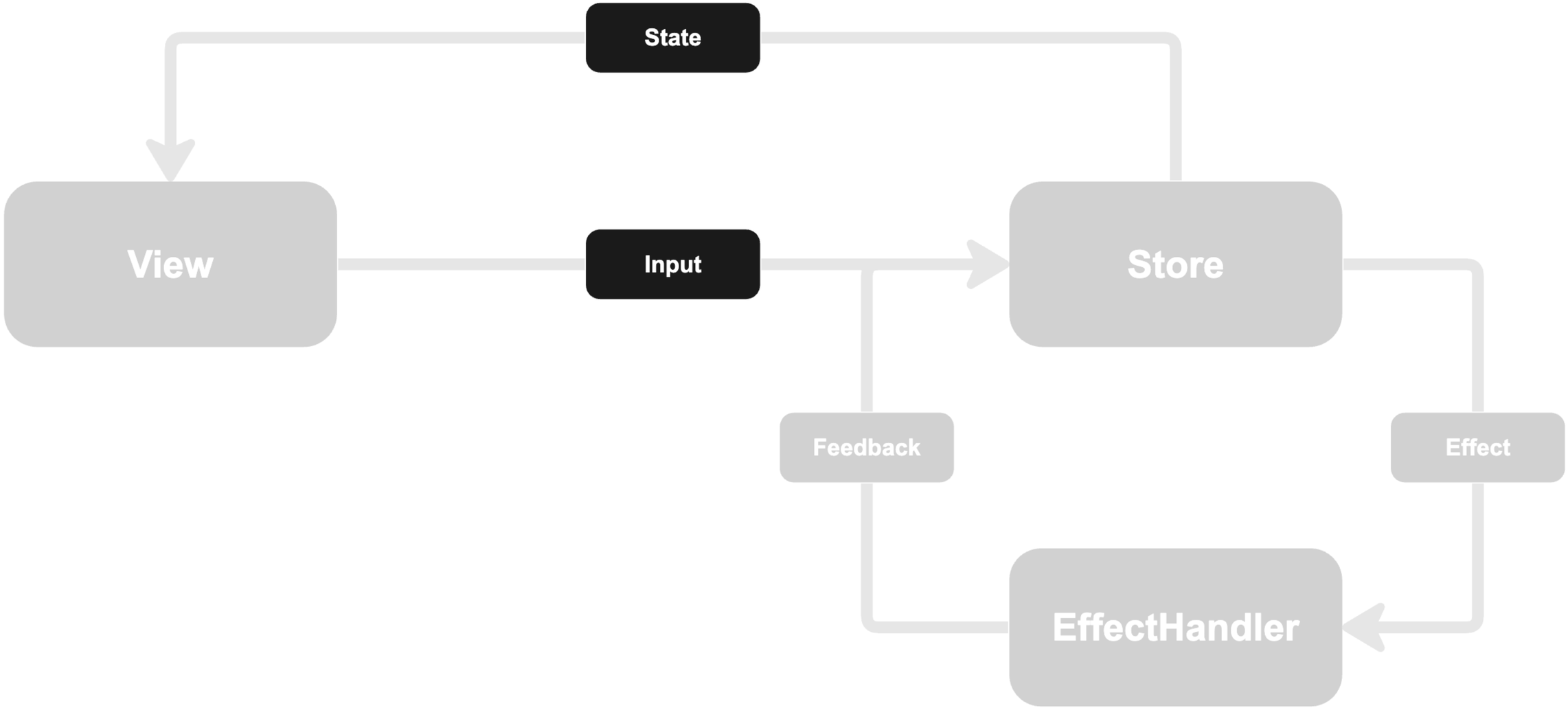
```
final class Store<S, EH>: ObservableObject where
  S: State,
  EH: EffectHandler<S> {
  @Published private(set) var state: S
  let effectHandler: EH

  func send(_ message: Message<S.Input, S.Feedback>) {
    var updatedState = state
    let effect = S.reduce(state: &updatedState, with: message)
    state = updatedState

    Task {
      if let effect,
         let feedback = await effectHandler.handle(effect: effect) {
        send(.feedback(feedback))
      }
    }
  }
}
```

```
final class Store<S, EH>: ObservableObject where
  S: State,
  EH: EffectHandler<S> {
  @Published private(set) var state: S
  let effectHandler: EH

  func send(_ message: Message<S.Input, S.Feedback>) {
    var updatedState = state
    let effect = S.reduce(state: &updatedState, with: message)
    state = updatedState

    Task {
      if let effect,
         let feedback = await effectHandler.handle(effect: effect) {
        send(.feedback(feedback))
      }
    }
  }
}
```

```
final class Store<S, EH>: ObservableObject where
  S: State,
  EH: EffectHandler<S> {
  @Published private(set) var state: S
  let effectHandler: EH

  func send(_ message: Message<S.Input, S.Feedback>) {
    var updatedState = state
    let effect = S.reduce(state: &updatedState, with: message)
    state = updatedState

    Task {
      if let effect,
         let feedback = await effectHandler.handle(effect: effect) {
        send(.feedback(feedback))
      }
    }
  }
}
```


State
View
Input
Store
Feedback
Effect
EffectHandler


State
View
Input
Store
Feedback
Effect
EffectHandler


State
View
Input
Store
Feedback
Effect
EffectHandler

```
struct CitiesSearchState: State {
  ...

  enum Input { ... }

  enum Feedback { ... }

  enum Effect { ... }

  static func reduce(
    state: inout Self,
    with message: Message<Input, Feedback>
  ) -> Effect? {
    switch message { ... }
  }
}
```

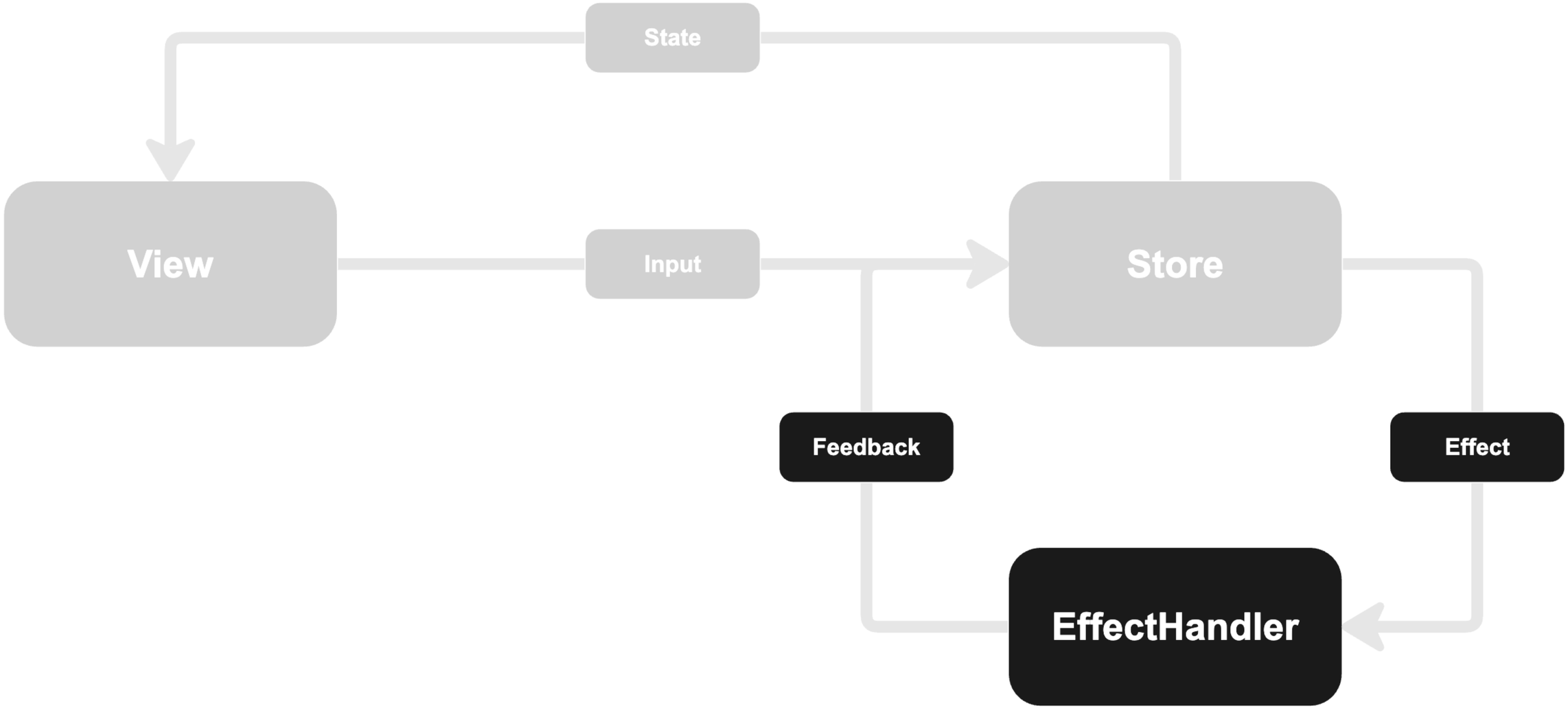
```
struct CitiesSearchState: State {
  ...

  enum Input { ... }

  enum Feedback { ... }

  enum Effect { ... }

  static func reduce(
    state: inout Self,
    with message: Message<Input, Feedback>
  ) -> Effect? {
    switch message { ... }
  }
}
```

```
struct CitiesSearchState: State {
  var listState: CitiesListState

  enum Input {
    case onAppear
  }

  enum Feedback {
    case suggestsFetchingFailed
  }

  enum Effect {
    case fetchRequest
  }

  static func reduce(
    state: inout Self,
    with message: Message<Input, Feedback>
  ) -> Effect? {
    switch message {
    case .input(.onAppear):
      state.listState = .loading
      return .fetchRequest
    case .feedback(.suggestsFetchingFailed):
      state.listState = .error
      return nil
    }
  }
}
```

```
struct CitiesSearchState: State {
  var listState: CitiesListState

  enum Input {
    case onAppear
  }

  enum Feedback {
    case suggestsFetchingFailed
  }

  enum Effect {
    case fetchRequest
  }

  static func reduce(
    state: inout Self,
    with message: Message<Input, Feedback>
  ) -> Effect? {
    switch message {
    case .input(.onAppear):
      state.listState = .loading
      return .fetchRequest
    case .feedback(.suggestsFetchingFailed):
      state.listState = .error
      return nil
    }
  }
}
```

```
struct CitiesSearchState: State {
  var listState: CitiesListState

  enum Input {
    case onAppear
  }

  enum Feedback {
    case suggestsFetchingFailed
  }

  enum Effect {
    case fetchRequest
  }

  static func reduce(
    state: inout Self,
    with message: Message<Input, Feedback>
  ) -> Effect? {
    switch message {
    case .input(.onAppear):
      state.listState = .loading
      return .fetchRequest
    case .feedback(.suggestsFetchingFailed):
      state.listState = .error
      return nil
    }
  }
}
```

```
struct CitiesSearchState: State {
  var listState: CitiesListState

  enum Input {
    case onAppear
  }

  enum Feedback {
    case suggestsFetchingFailed
  }

  enum Effect {
    case fetchRequest
  }

  static func reduce(
    state: inout Self,
    with message: Message<Input, Feedback>
  ) -> Effect? {
    switch message {
    case .input(.onAppear):
      state.listState = .loading
      return .fetchRequest
    case .feedback(.suggestsFetchingFailed):
      state.listState = .error
      return nil
    }
  }
}
```


State
View
Input
Store
Feedback
Effect
EffectHandler

```
final class CitiesSearchEffectHandler: EffectHandler {
  func handle(effect: S.Effect) async -> S.Feedback? {
    switch effect {
    case .fetchRequest:
      // async networking
      return .suggestsFetchingFailed
  }
}
```

Старые добрые делегаты

Progressive disclosure API

Императивность

Generic ObservableObject

Старые добрые делегаты

Progressive disclosure API

Императивность

Generic ObservableObject

Старые добрые делегаты

Progressive disclosure API

Императивность

Generic ObservableObject

Старые добрые делегаты

Progressive disclosure API

Императивность

Generic ObservableObject

Старые добрые делегаты

Progressive disclosure API

Императивность

Generic ObservableObject

Старые добрые делегаты

Progressive disclosure API

Императивность

Generic ObservableObject

Требует вложений в унификацию

Меньше гибкости

Старые добрые делегаты

Progressive disclosure API

Императивность

Generic ObservableObject

Требует вложений в унификацию

Меньше гибкости

MVVM

~~MVVM~~

~~MVVM~~

Reactive programming

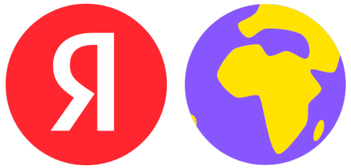
~~MVVM~~

~~Reactive programming~~

~~MVVM~~

~~Reactive programming~~

Generic ObservableObject



~~MVVM~~

~~Reactive programming~~

Generic ObservableObject

Николай Пучко

nick-puchko

NikolaiPuchko