아키텍처와 Composition

객체를 조립해서 쓰는 방식을 Composition이라고 합니다.
로직을 여러 곳으로 분산시킨 후 재조립해 쓰면 많은 장점을 얻을 수 있습니다.
iOS와 Swift, 그리고 잘 알려진 여러 아키텍처에서는
어떻게 Composition을 활용하고 있는지 알아봅니다.

Composition (합성, 조립)

A way to combine objects and data types
into more complex ones

- *Wikipedia*

“

Favor object composition over class inheritance

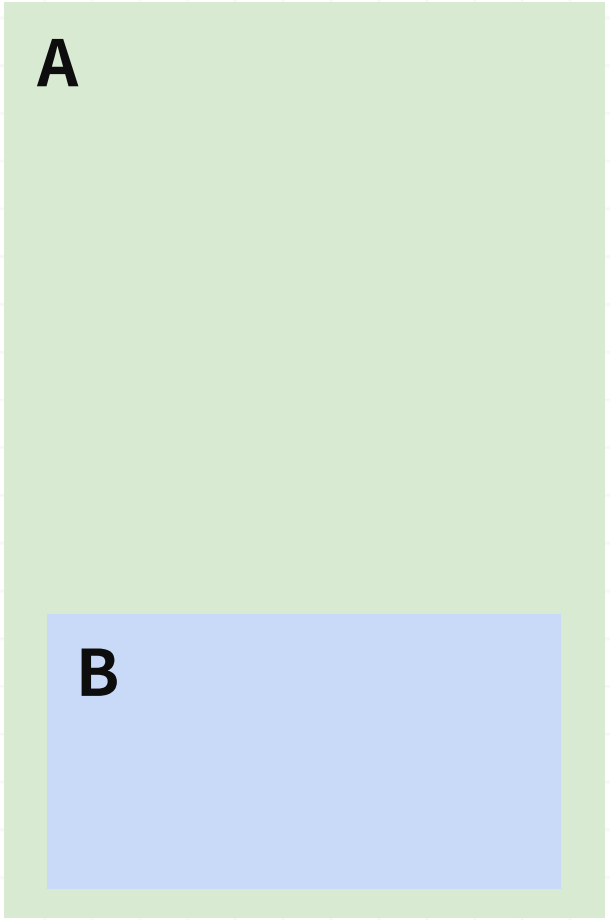
- Gang of Four, Design Patterns (1994)

객체 Composition

A

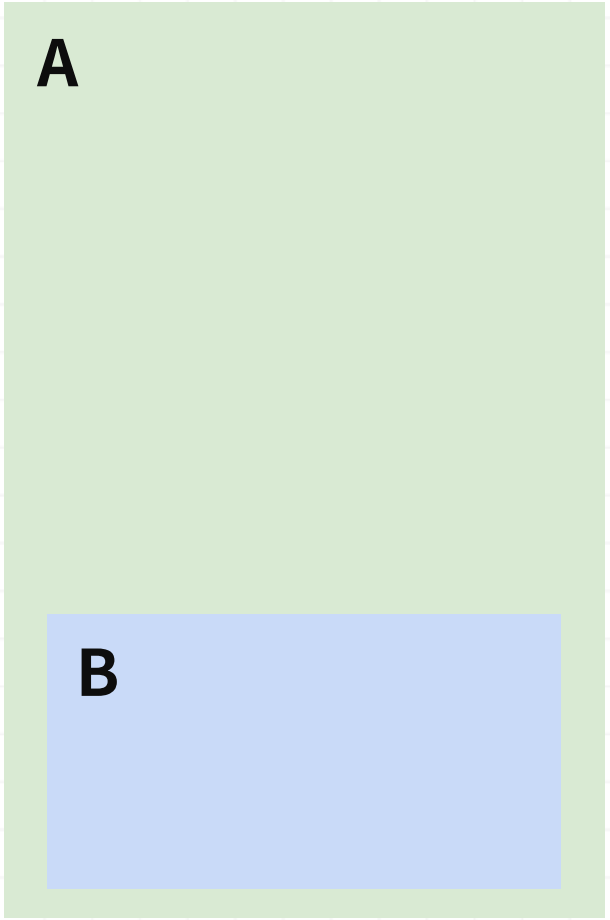
```
class A: UIViewController {
  func setupViews()
    // add subviews
  }
}
```

객체 Composition




```
class A: UIViewController {
  func setupViews()
    // add subviews
  }
}
```

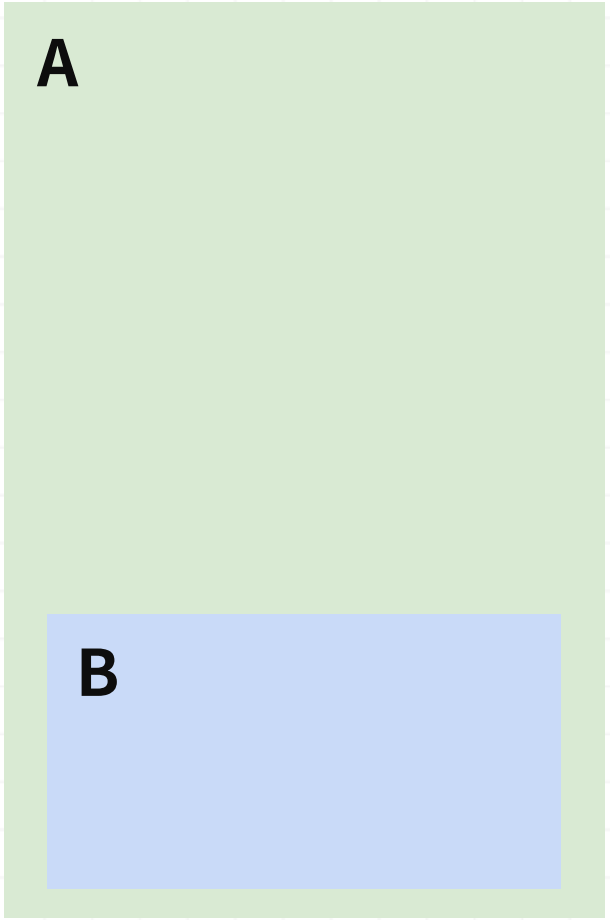
객체 Composition




```
class A: UIViewController {
  func setupViews()
    // add subviews
  }
}

 class B: A {
   override func setupViews() {
     super.setupViews()
     // add B View
   }
 }
```

객체 Composition




```
class A: UIViewController {

}
```

```
class B: UIViewController {

}
```

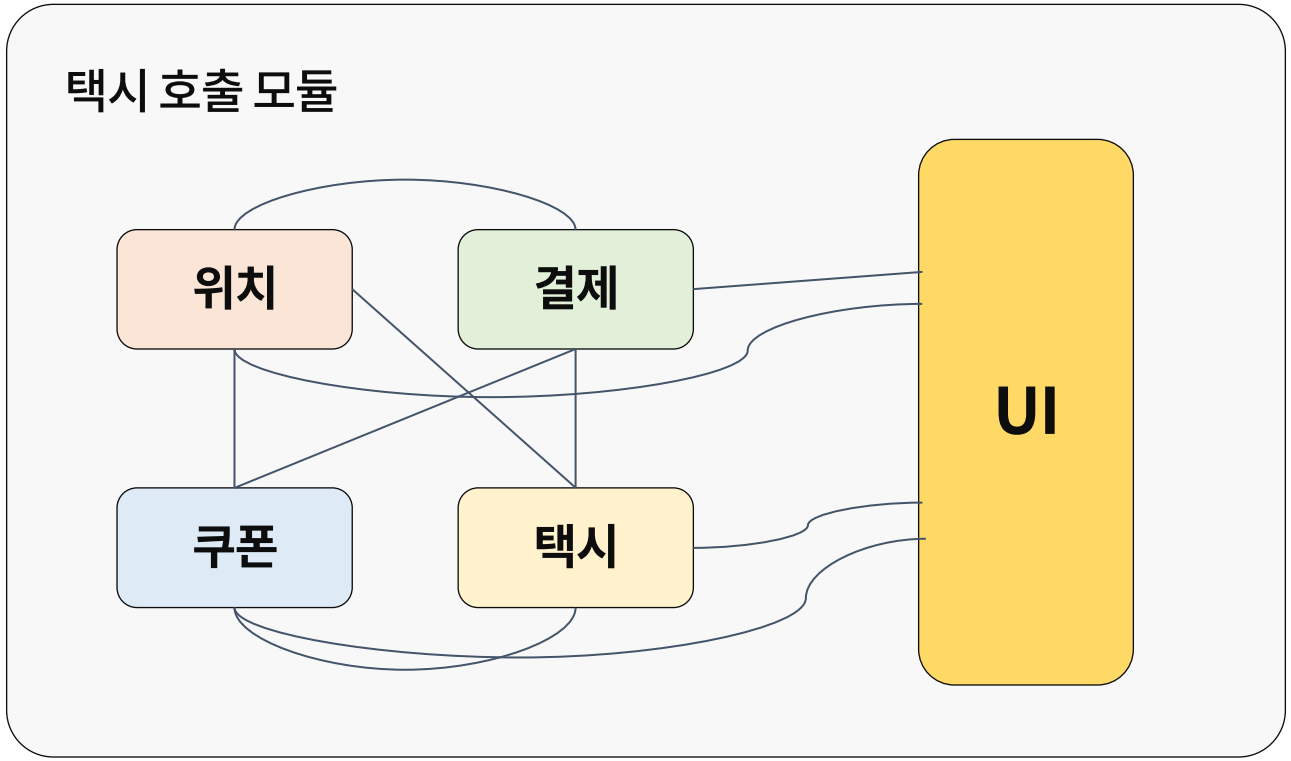
```
class C: UIViewController {
  private let a: UIViewController
  private let b: UIViewController

  // add b to child viewcontroller
}
```

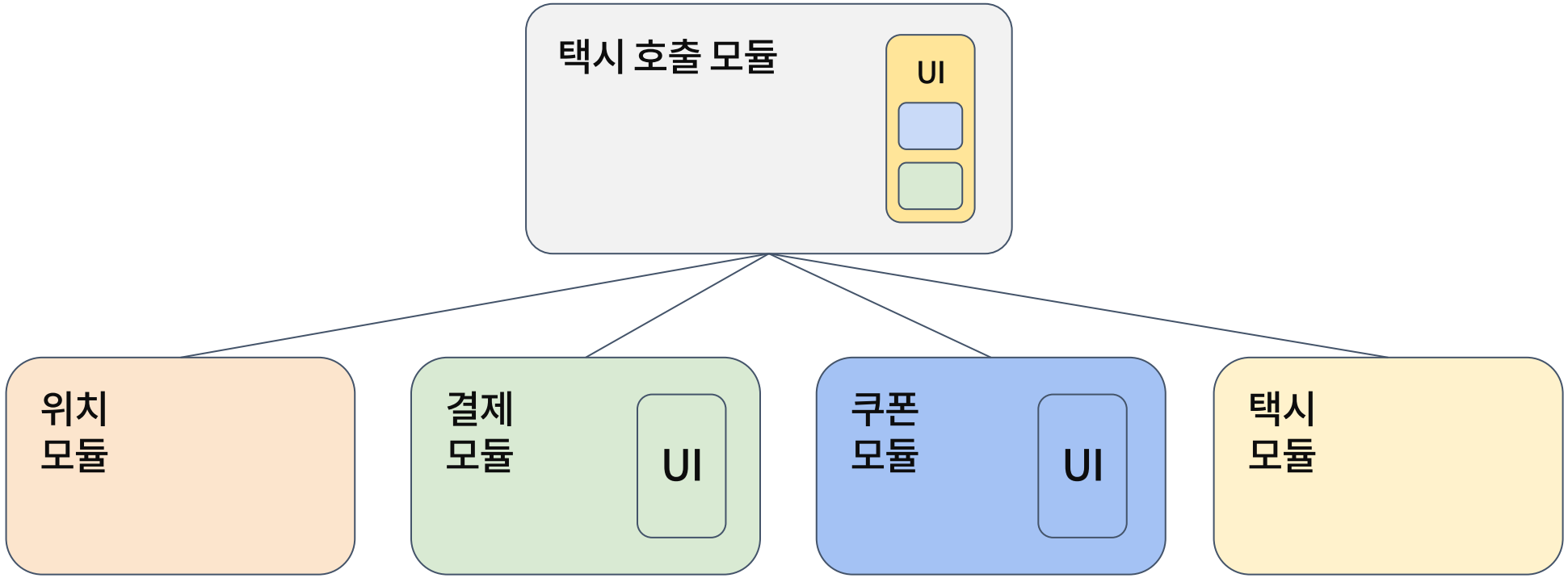
함수 Composition

현실적인 예제

모듈 Composition

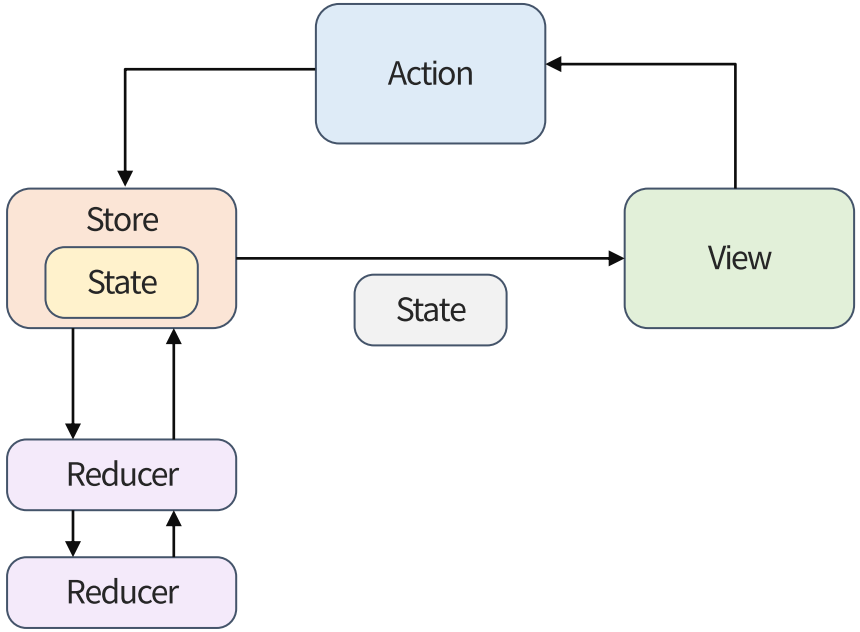


모듈 Composition



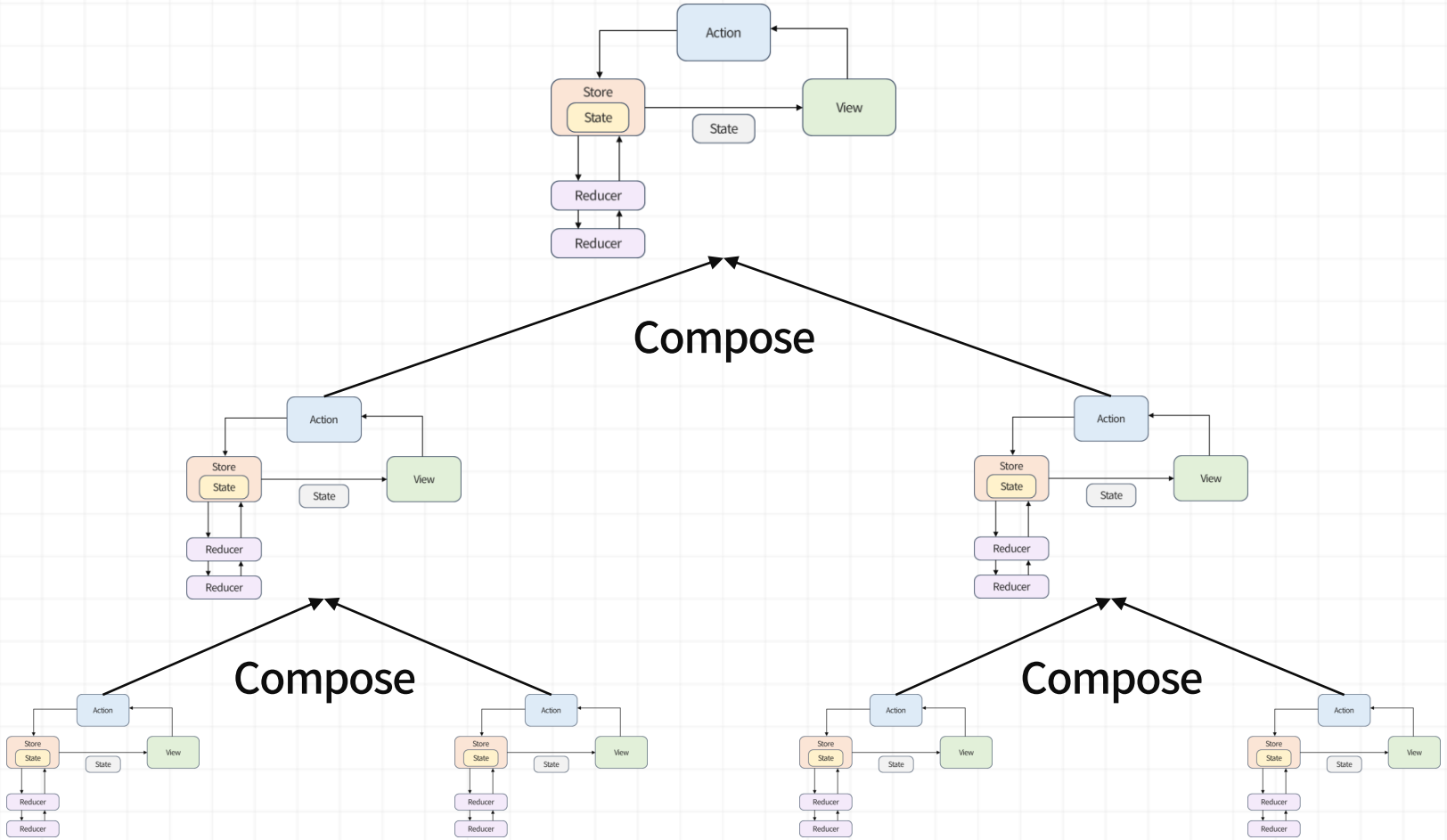
Redux기반 아키텍처의 Composition

The Composable Architecture, ReSwift, etc



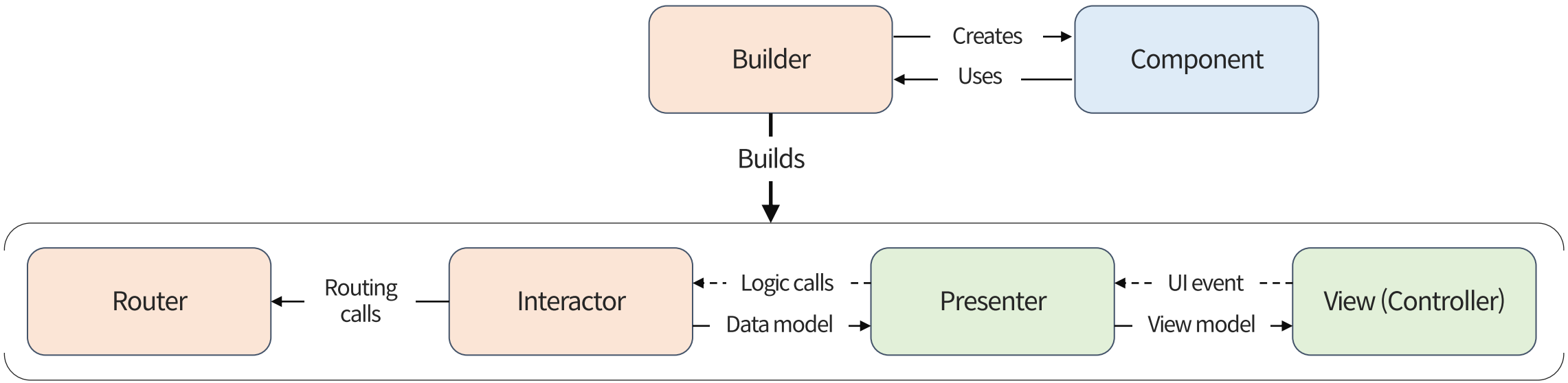
- State는 sub-states를 멤버 변수로 가진다.
- Reducer는 sub-reducers로 분리할 수 있다.

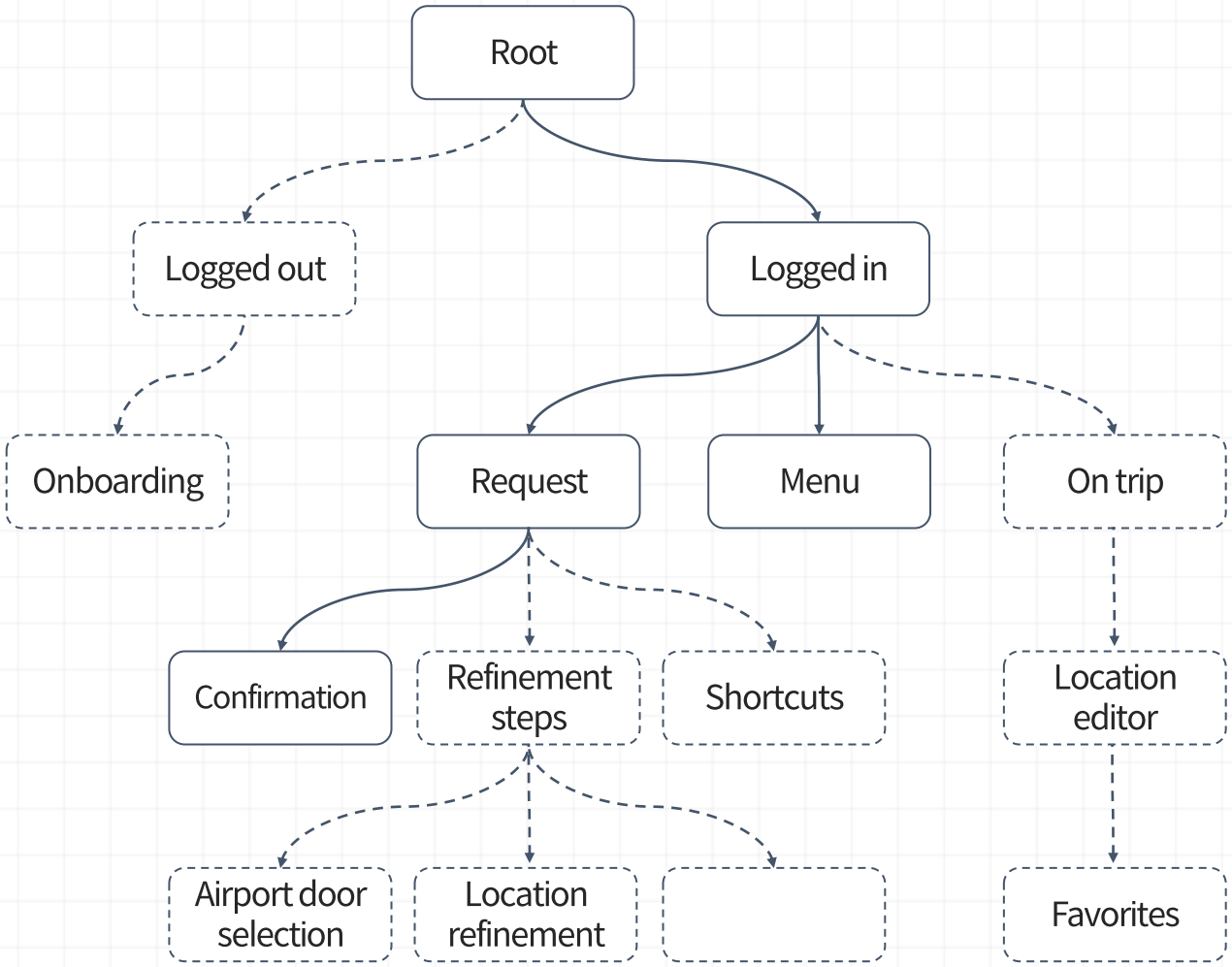
```
func appReducer(action: Action, state: State?) -> State {
    return State(
      navigationState: navigationReducer(action, state?.navigationState),
      authenticationState: authenticationReducer(action, state?.authenticationState),
      repositories: repositoriesReducer(action, state?.repositories),
      bookmarks: bookmarksReducer(action, state?.bookmarks)
    )
}
```


Action
Store
State
View
State
Reducer
Reducer
Compose
Action
Store
State
View
State
Reducer
Reducer
Action
Store
State
View
State
Reducer
Reducer
Compose
Compose
Action
Store
State
View
State
Reducer
Reducer
Action
Store
State
View
State
Reducer
Reducer
Action
Store
State
View
State
Reducer
Reducer
Action
Store
State
View
State
Reducer
Reducer

Interactor, Router기반 아키텍처의 Composition

RIBs, VIPER





https://github.com/uber/RIBs/wiki

앱과 비즈니스 로직

앱을 구성하고 있는 코드를 로직의 특성에 따라 분류해볼 수 있습니다.
여러 아키텍처는 각 로직을 어디에서 처리하는지 봄으로써
아키텍처의 이해도를 높여봅니다.

앱 로직의 분류

데이터 저장

메모리 캐시, 바이너리,
데이터베이스, 파일 등

내비게이션

화면의 이동
(Present, Dismiss, Push, Pop)

뷰

UIView, UIViewController

서비스

네트워크, 블루투스,
위치 서비스 등

코디네이션

각종 layer를 조합해
앱이 사용자를 위해 하는 일

프레젠테이션

이미지, 색상, 폰트 등
UI 모델 변환

앱 로직의 분류

외부 디펜던시

데이터 저장

메모리 캐시, 바이너리,
데이터베이스, 파일 등

서비스

네트워크, 블루투스,
위치 서비스 등

비즈니스 로직

내비게이션

화면의 이동
(Present, Dismiss, Push, Pop)

코디네이션

각종 layer를 조합해
앱이 사용자를 위해 하는 일

UI

뷰

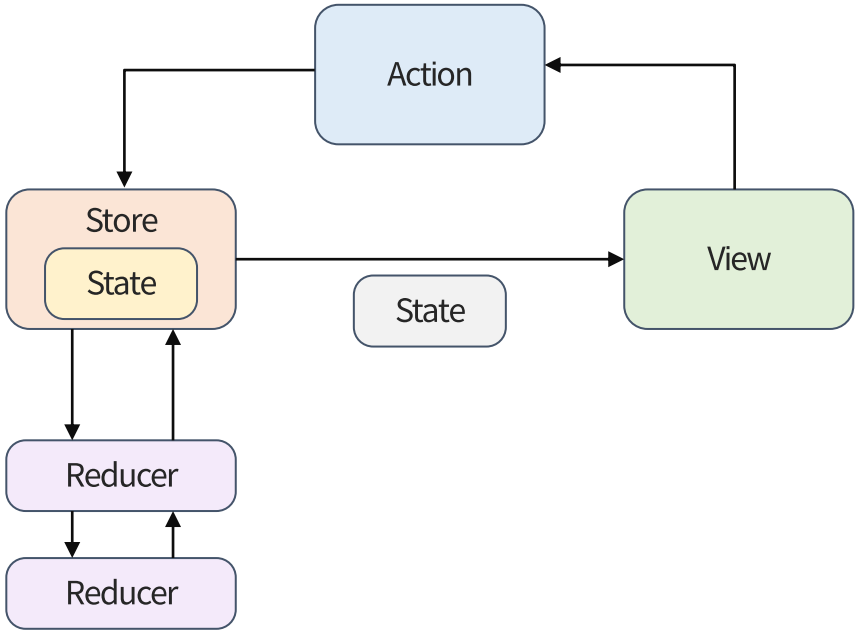
UIView, UIViewController

프레젠테이션

이미지, 색상, 폰트 등
UI 모델 변환

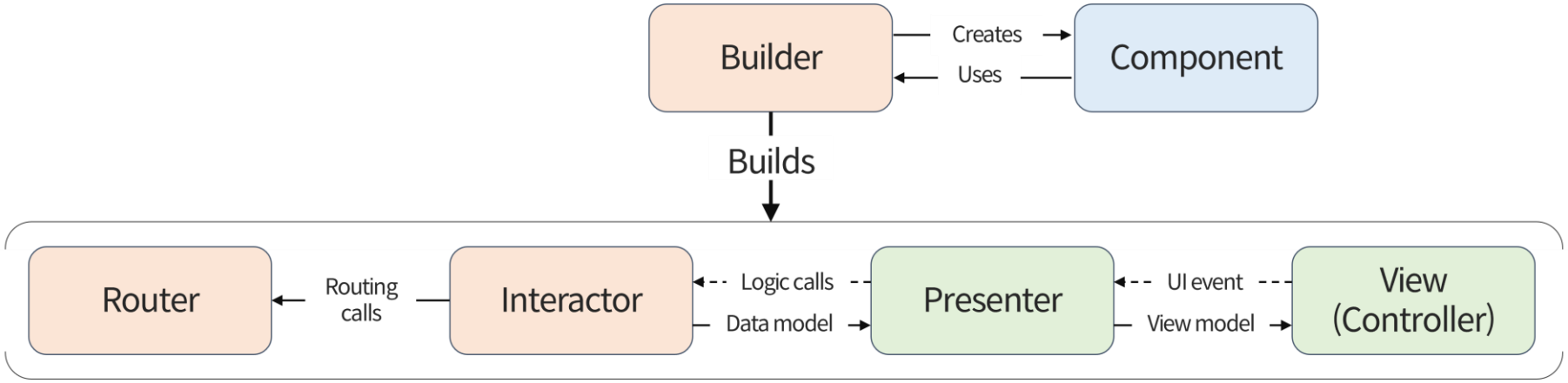
Redux Architecture

The Composable Architecture, ReSwift, etc



UI	View
외부 디펜던시	Environment, Middleware
코디네이션	Reducer, Action
내비게이션	Router

RIBs, VIPER

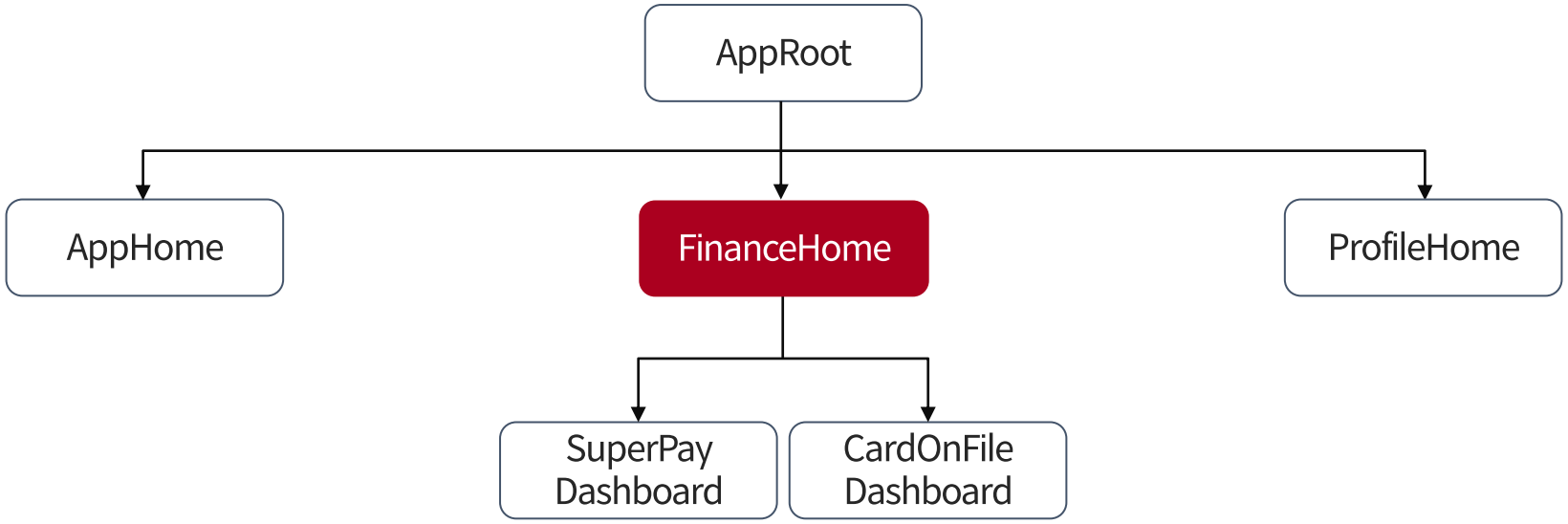


UI	View	코디네이션	Interactor
외부 디펜던시	Builder, Component	내비게이션	Router

복잡한 뷰

인터랙션이 복잡한 뷰를
작은 단위로 분해한 후 조립해봅니다

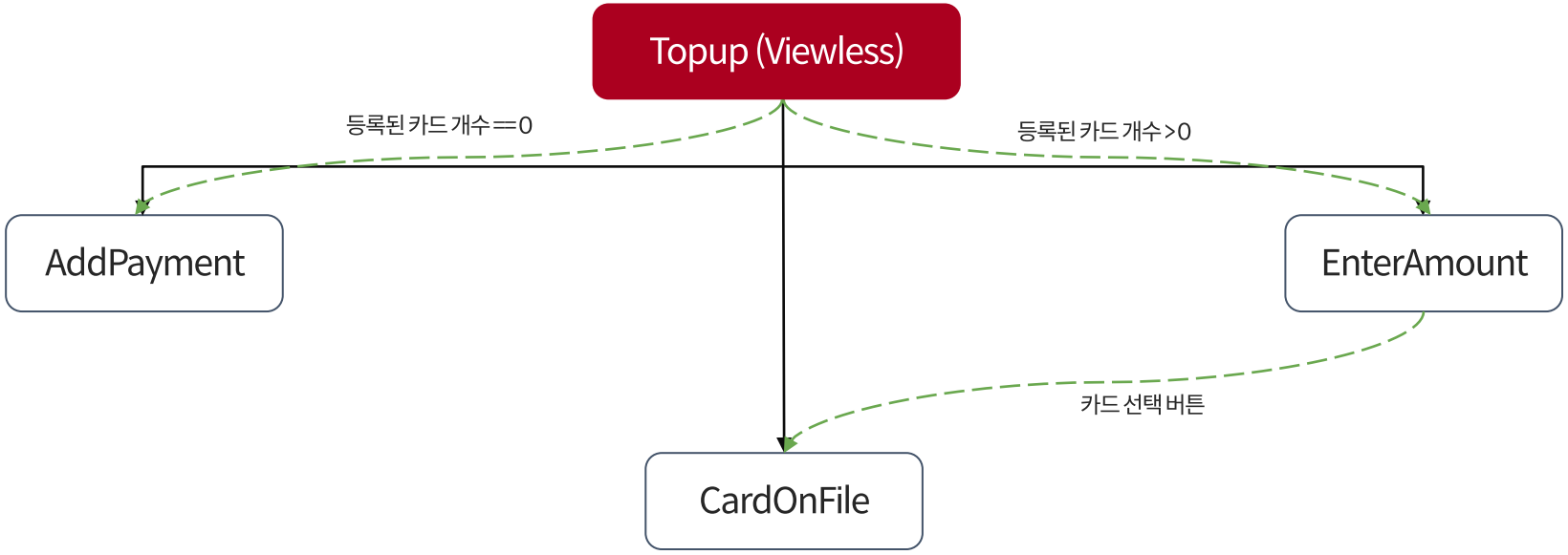
슈퍼페이 홈 화면


AppRoot
AppHome
FinanceHome
ProfileHome
SuperPay Dashboard
CardOnFile Dashboard

복잡한 플로우

RIBs의 장점 중 하나는 비즈니스 로직과
뷰 계층을 분리할 수 있다는 점입니다.
뷰에 얽매이지 않고 비즈니스 로직을 구현해봅니다.

슈퍼페이 충전 플로우


Topup (Viewless)
등록된 카드 개수 == 0
등록된 카드 개수 > 0
AddPayment
EnterAmount
카드 선택 버튼
CardOnFile