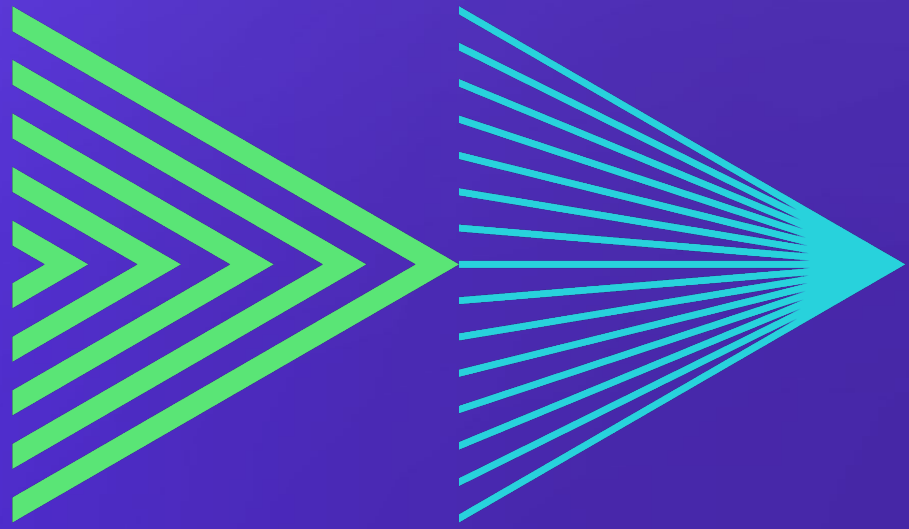
NHN FORWARD

ReactorKit을 이용한 MVI 적용 과정

NHN Dooray 모바일서비스개발팀
임덕현



다룰 내용

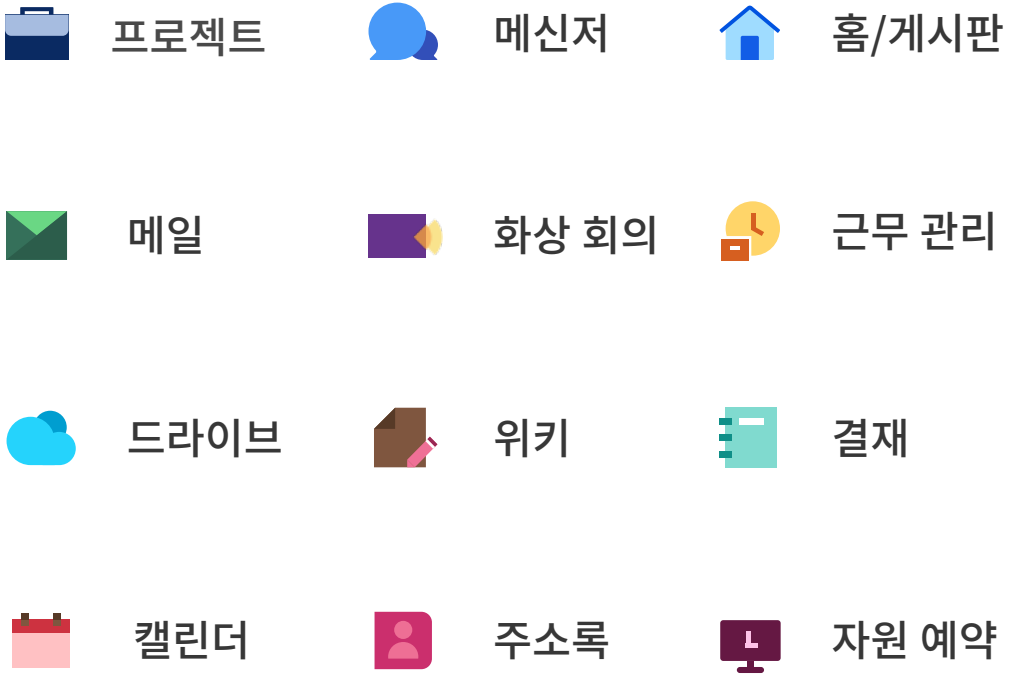
1. 기존 Dooray! 앱 구조 및 문제점
2. MVI란?
3. MVI, ReactorKit 적용 과정
4. 도입 후 개선 사항
5. 단점 및 고려 사항
6. Q&A

두레이 Dooray!

NHN FORWARD

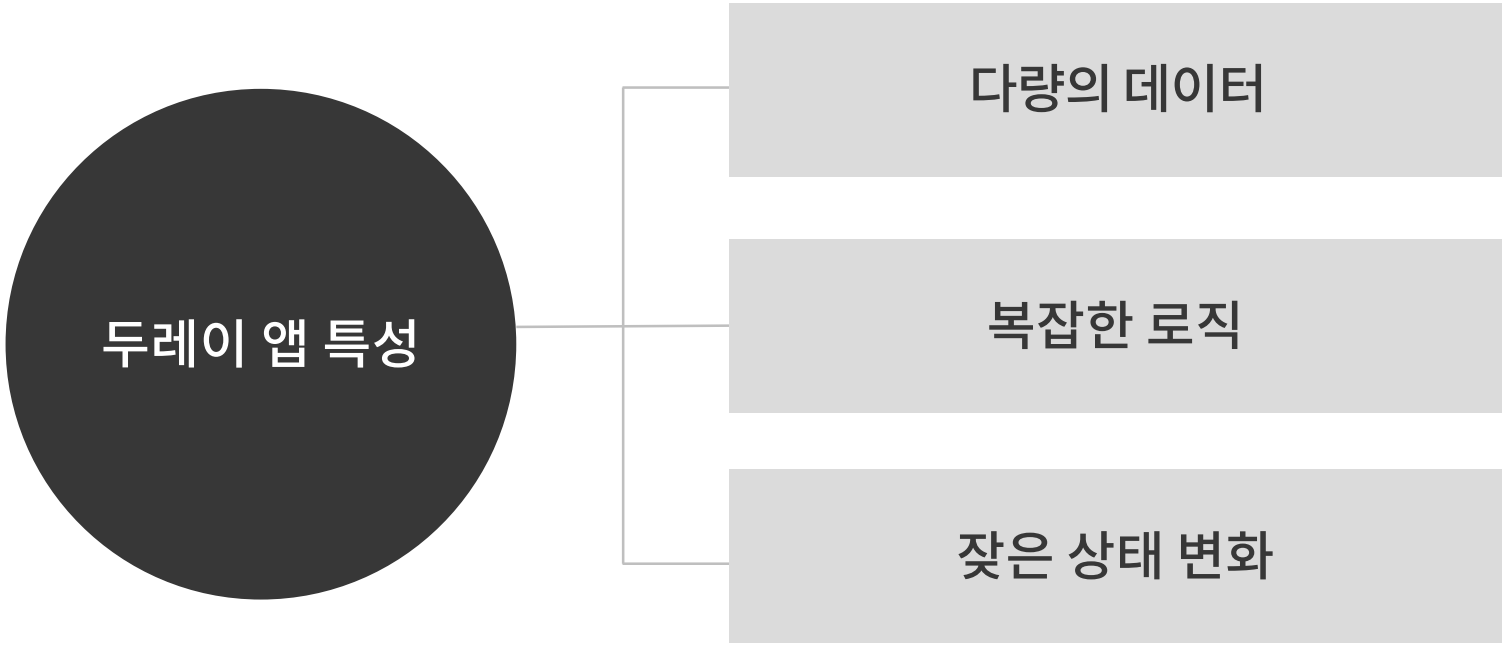
Dooray! 소개

- 업무용 올인원 협업 툴 ‘두레이(Dooray)!’
- 프로젝트, 메신저, 메일, 위키, 주소록, 결재 등 협업 시 필요한 도구를 통합하여 제공
- Dooray! 기능을 모바일에서 제공



두레이 Dooray!

Dooray! 앱 특성

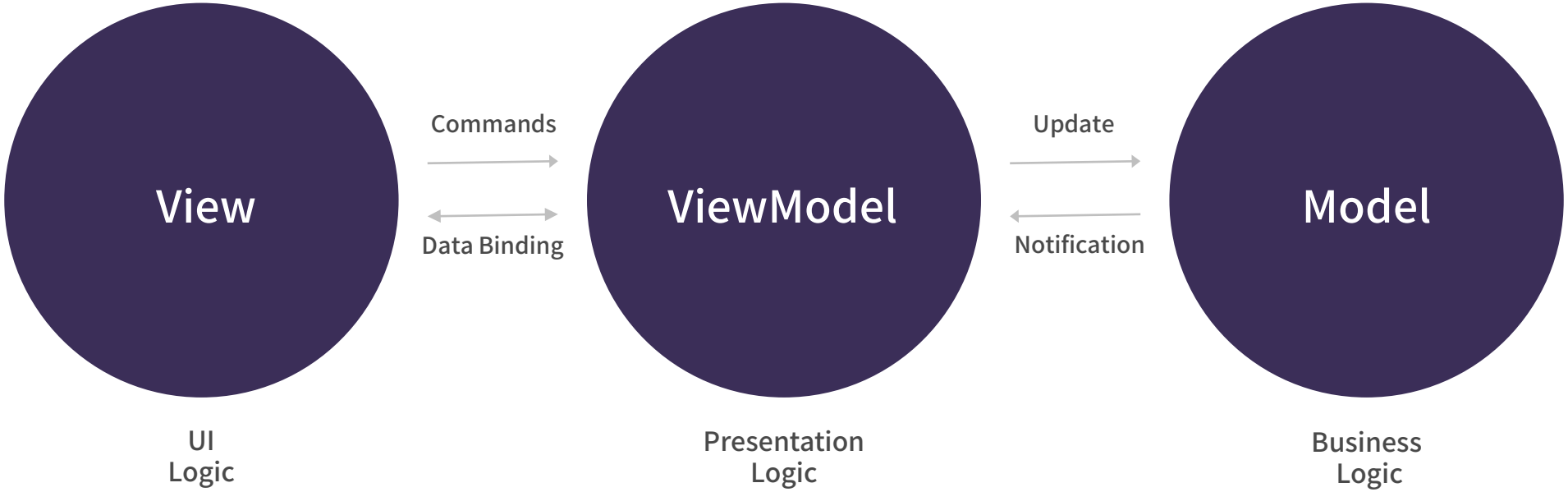


기존 Dooray! 앱 구조와 문제점

기존 사용 패턴 | MVVM

MVVM

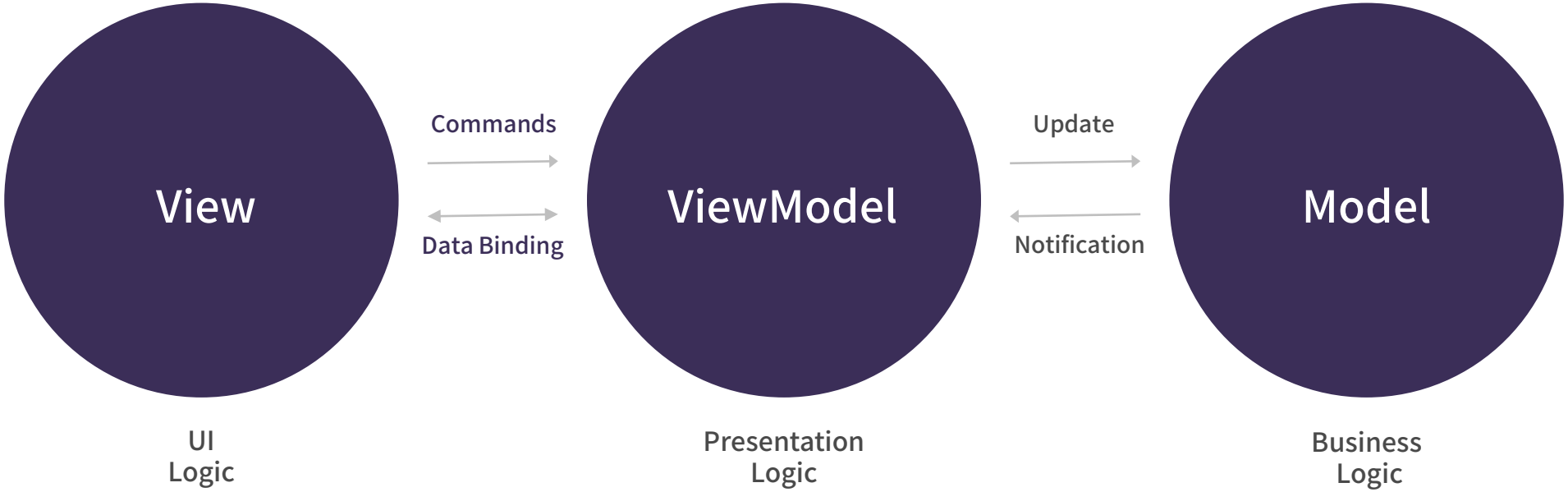
- Model, View, ViewModel
- View와 ViewModel의 명확한 역할 분리
- 독립성 유지, 의존성 감소



기존 사용 패턴 | MVVM

MVVM 패턴의 한계점

- 복잡한 데이터 흐름
- 상태 충돌 – 스레드 안전성

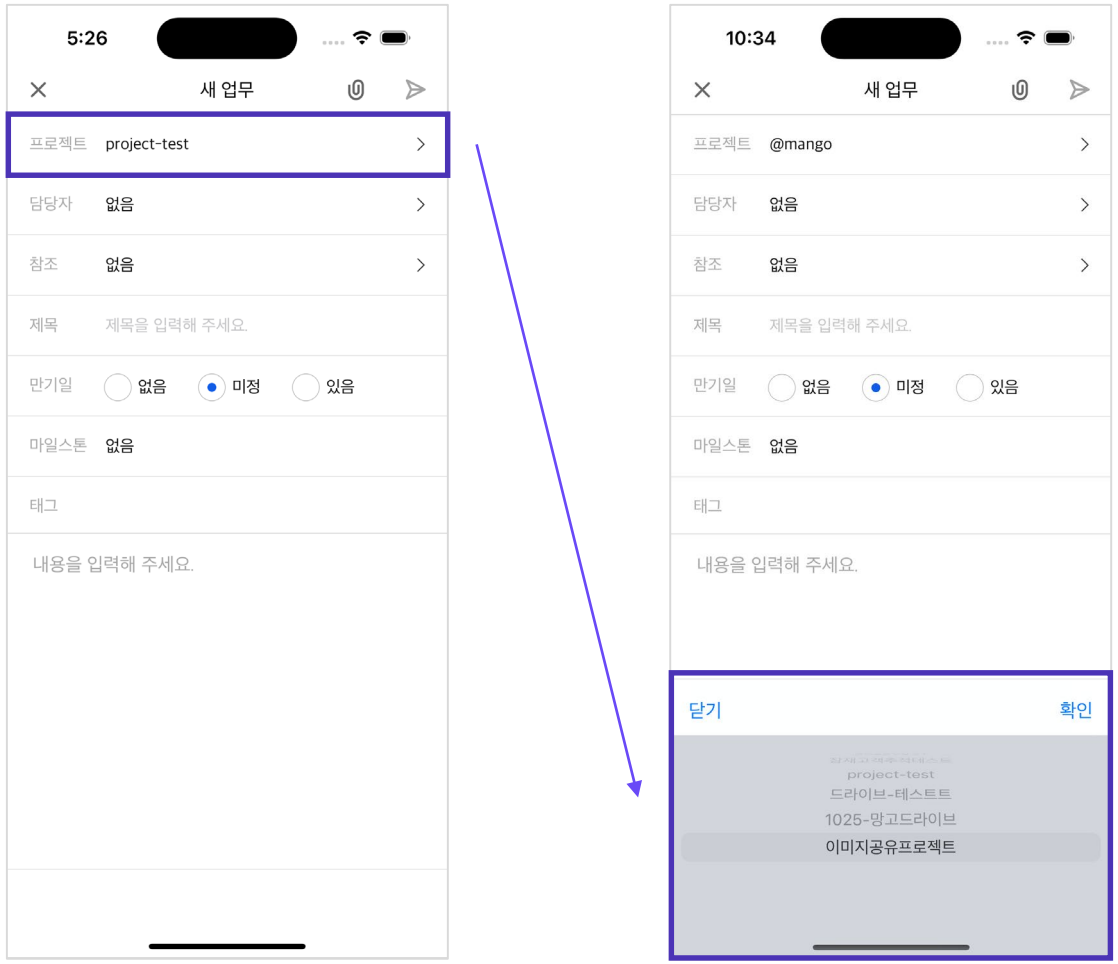


문제점

NHN FORWARD

예시: 업무 등록 > 프로젝트 변경

- 기대 동작
  1. View 업데이트

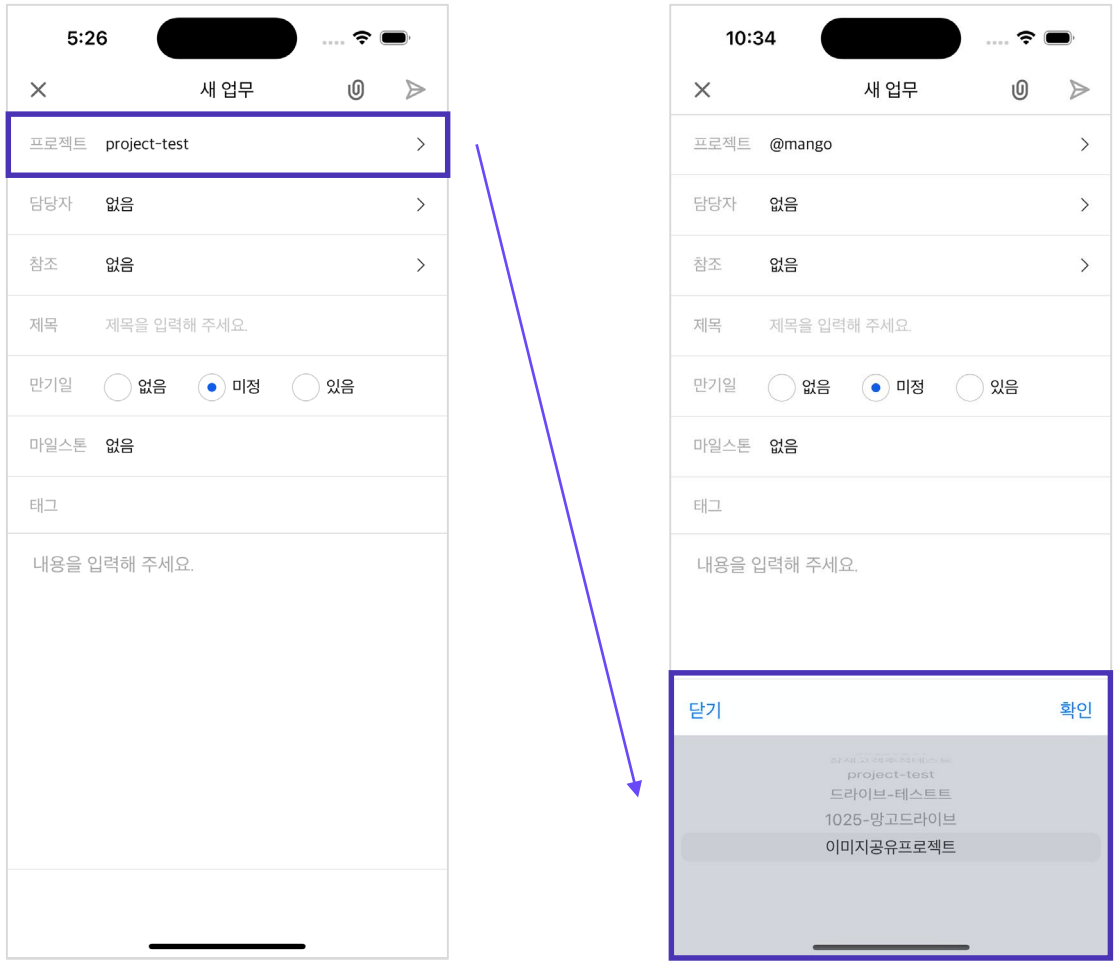


문제점

NHN FORWARD

예시: 업무 등록 > 프로젝트 변경

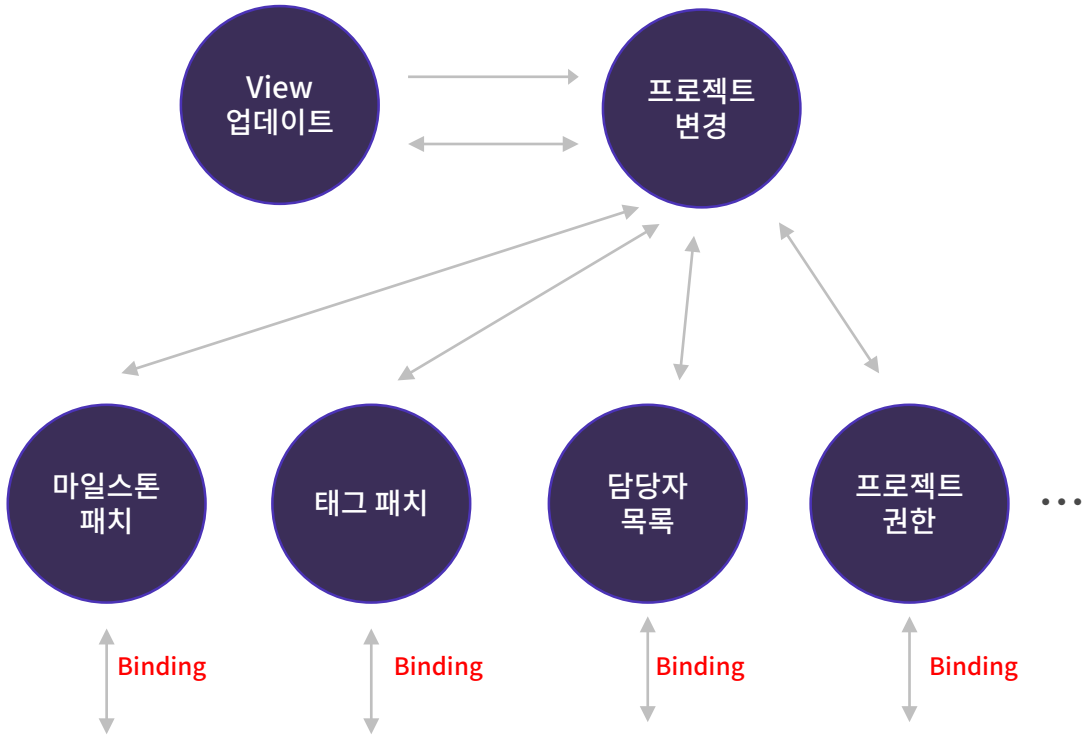
- 기대 동작
  1. View 업데이트
  2. 마일스톤 목록 패치
  3. 태그 목록 패치
  4. 프로젝트 권한 업데이트
  5. 담당자, 참조 업데이트
  …

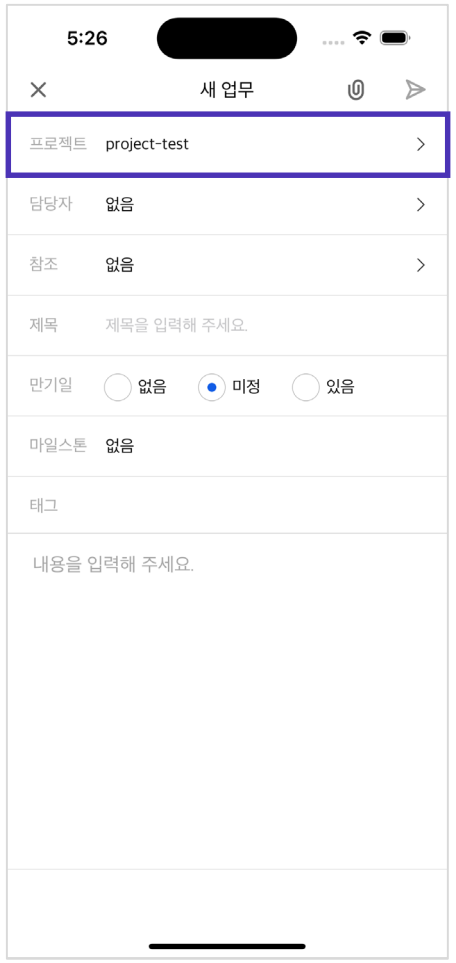


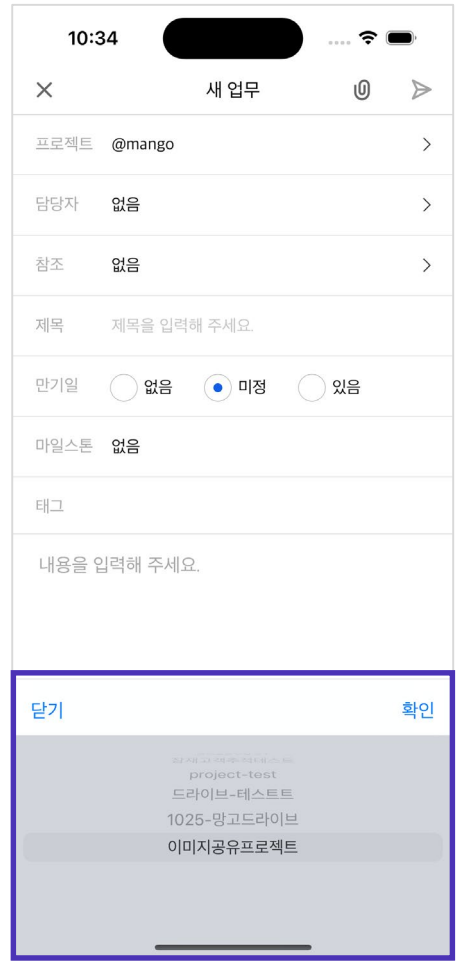
문제점

NHN FORWARD

예시: 업무 등록 > 프로젝트 변경

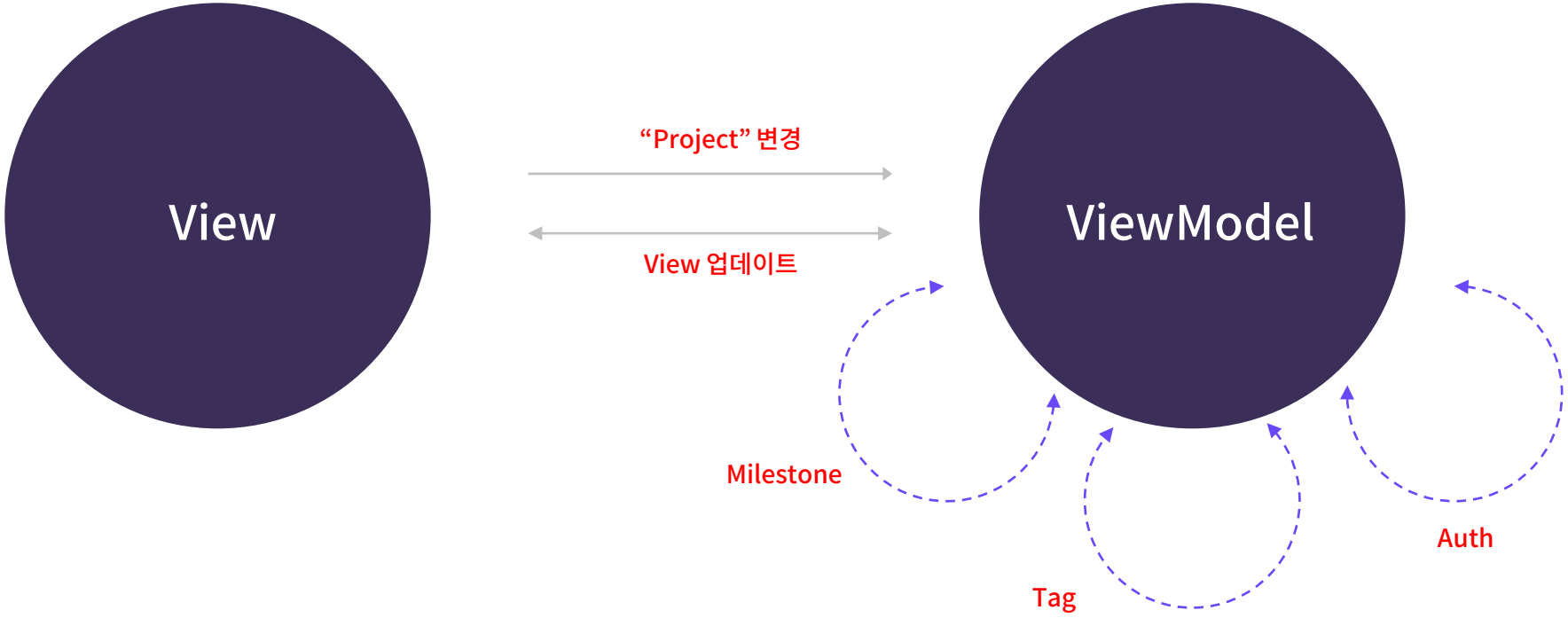






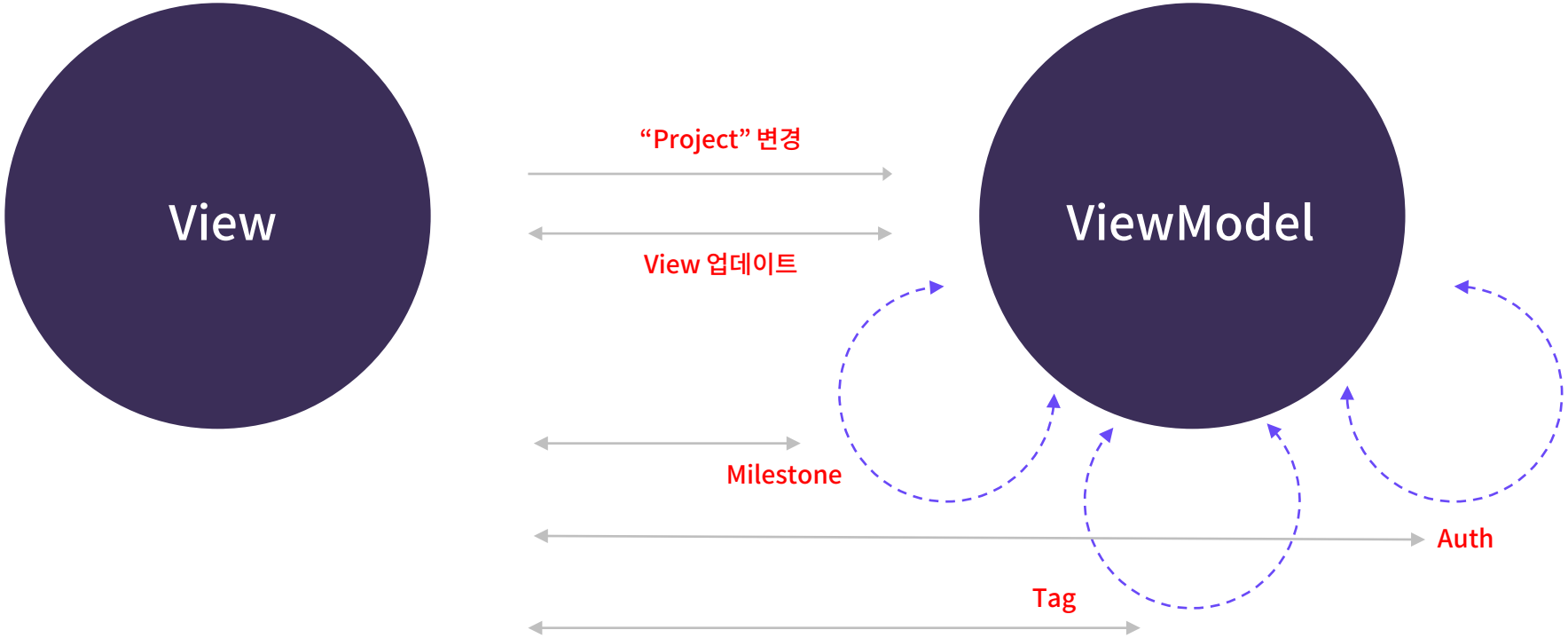
문제점

예시: 업무 등록 > 프로젝트 변경



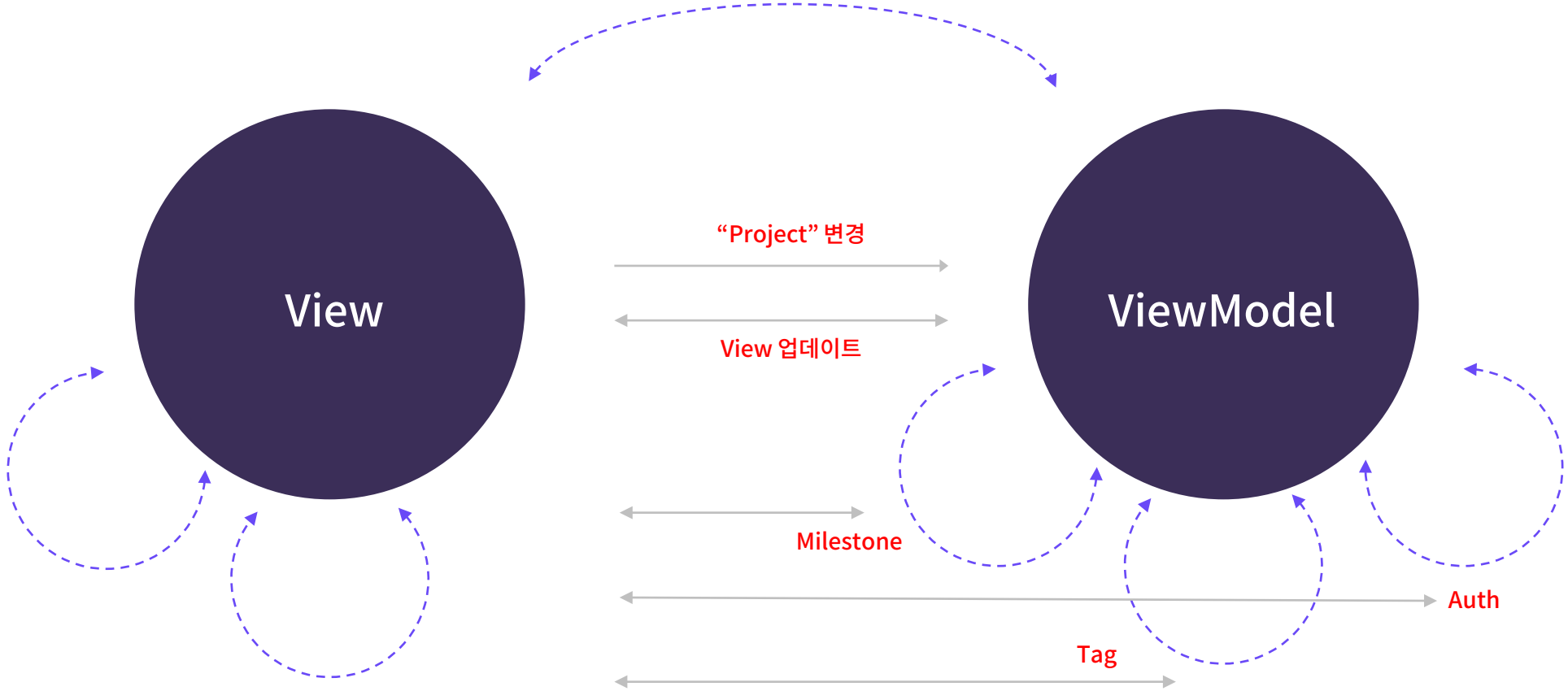
문제점

예시: 업무 등록 > 프로젝트 변경



문제점

예시: 업무 등록 > 프로젝트 변경



새로운 패턴의 필요성

Dooray! 앱 특성 고려 시, 새로운 패턴 활용 불가피

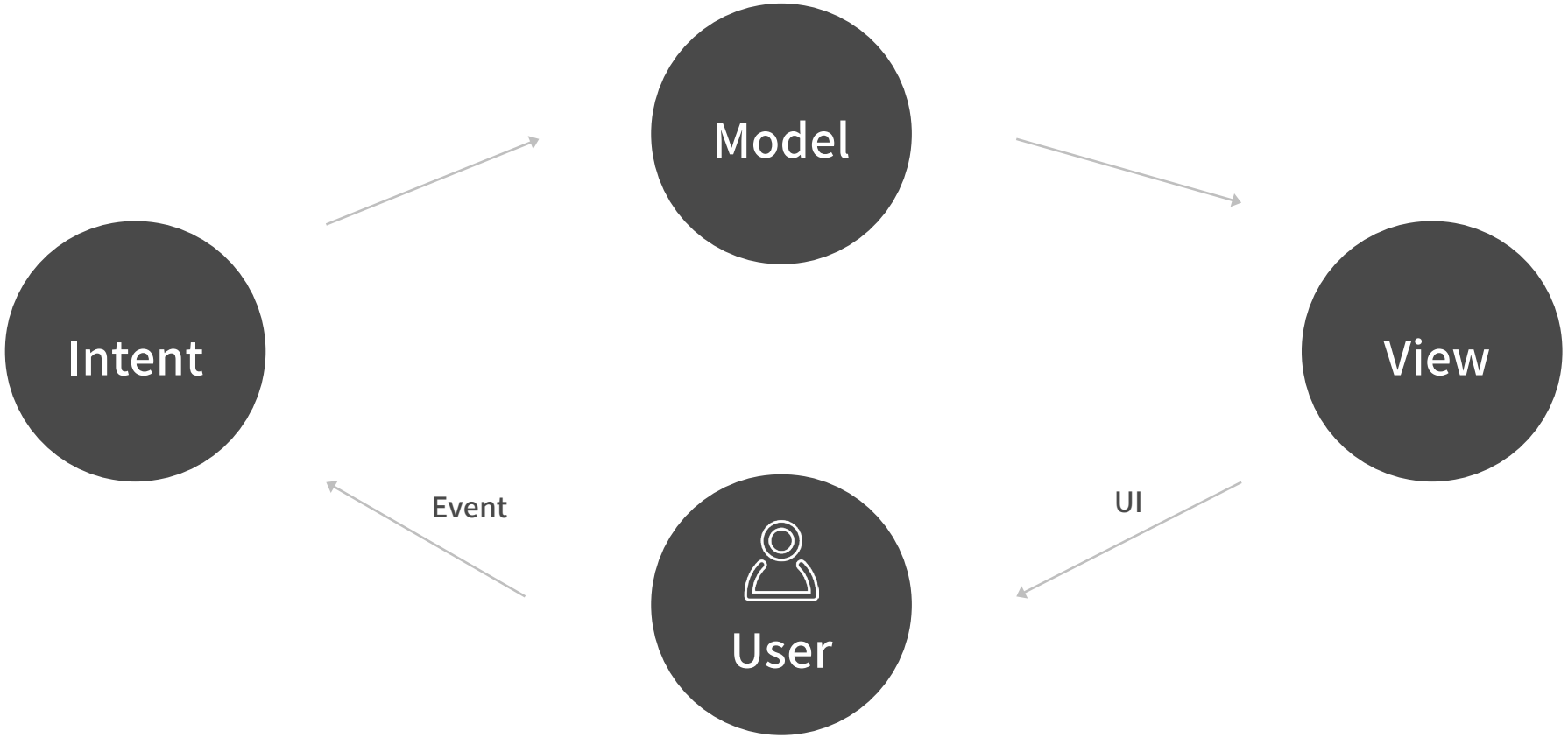
MVVM의 한계점		Dooray! 앱의 특성
복잡한 데이터 흐름		복잡한 로직
데이터 관리 어려움		다량의 데이터
상태 관리 어려움		잦은 상태 변화

MVI란?

MVI란?

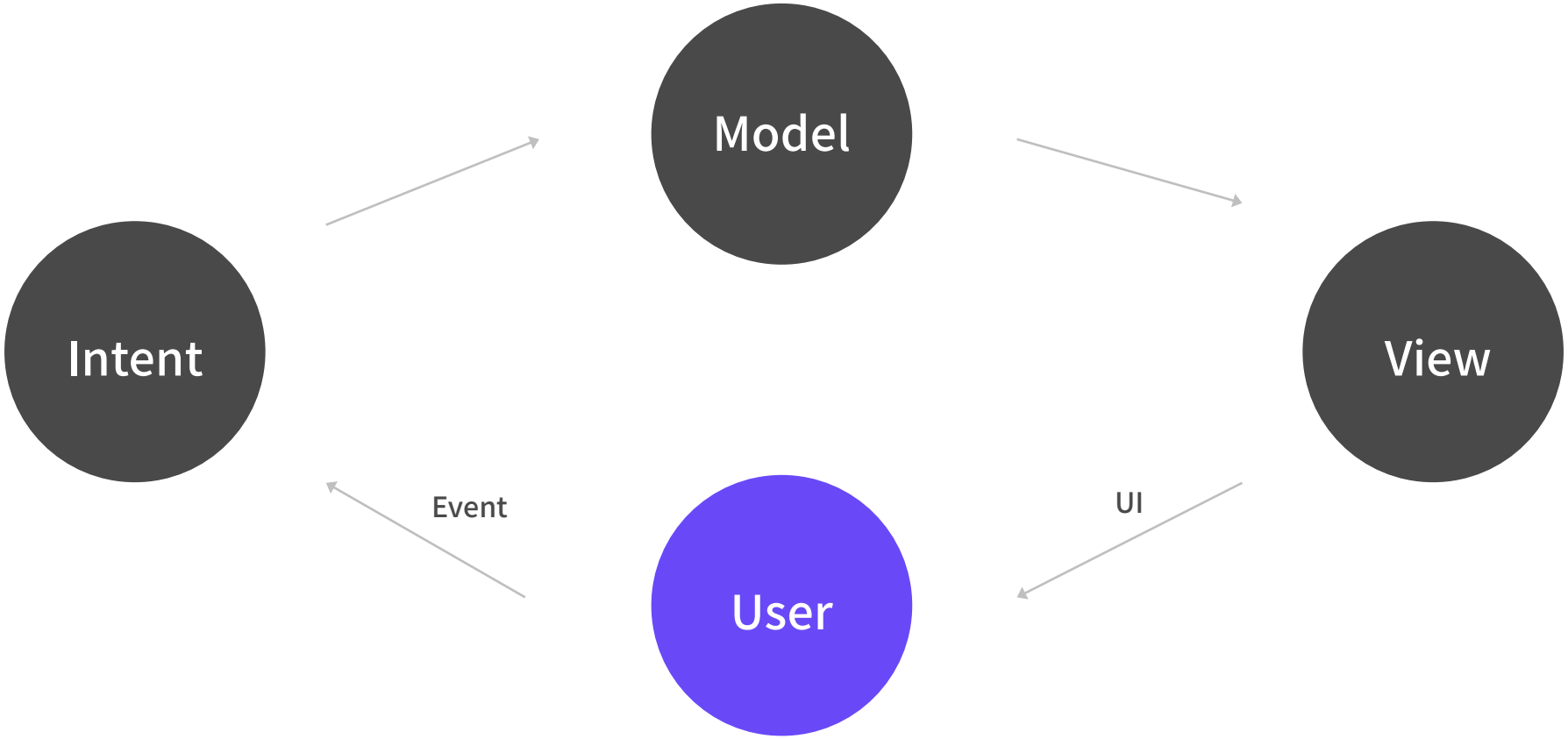
MVI란?

- Model - View – Intent
- 단방향 아키텍처 패턴
- 디버깅 용이
- 상태 관리 용이



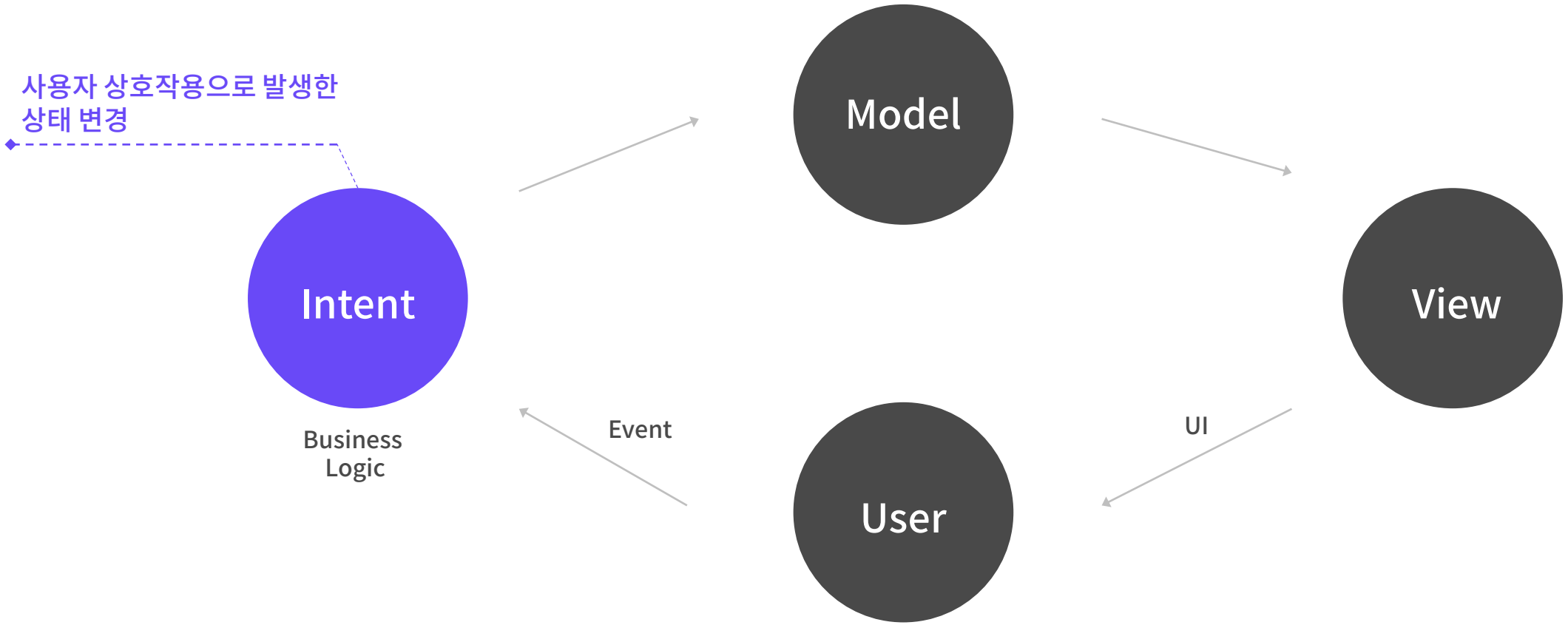
MVI란?

MVI 구조



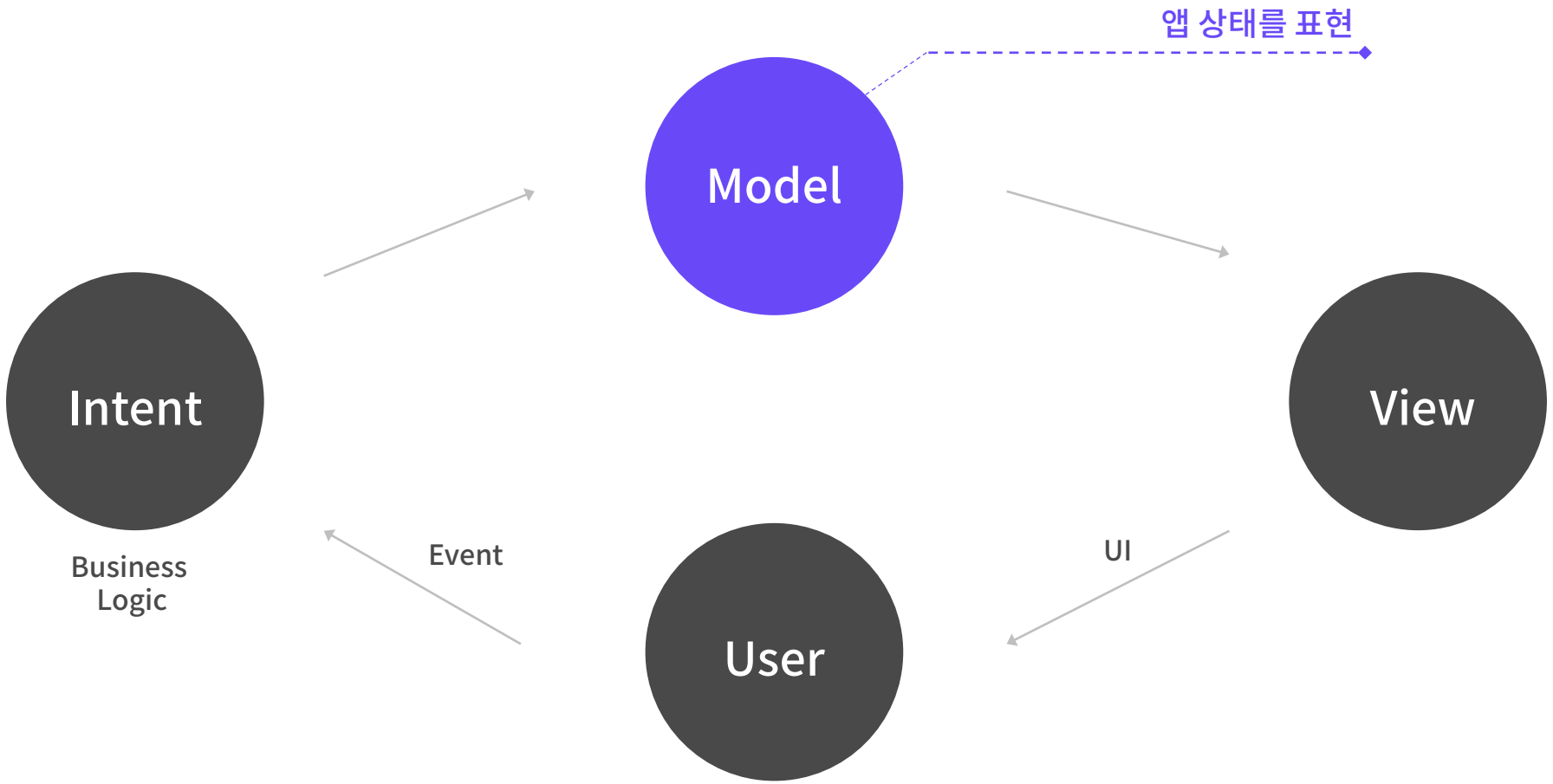
MVI란?

MVI 구조


사용자 상호작용으로 발생한
상태 변경
Intent
Business
Logic
Model
View
User
Event
UI

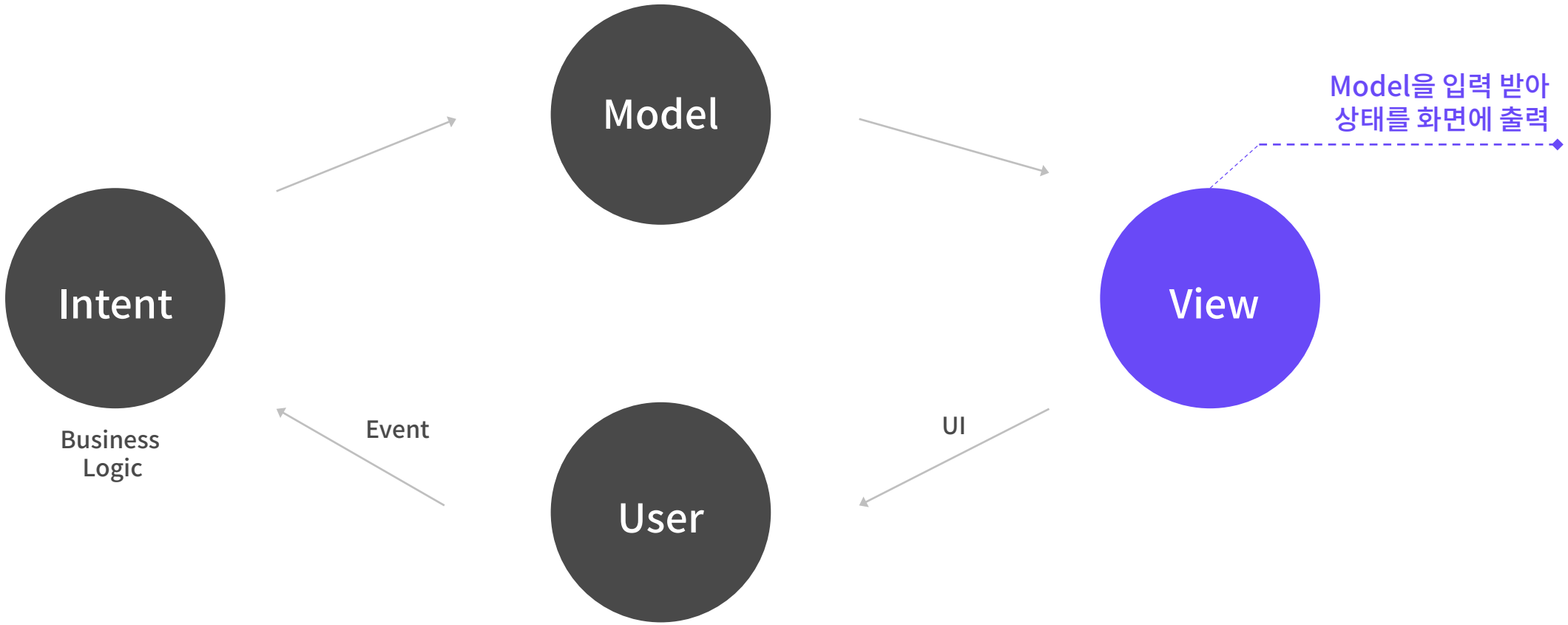
MVI란?

MVI 구조



MVI란?

MVI 구조

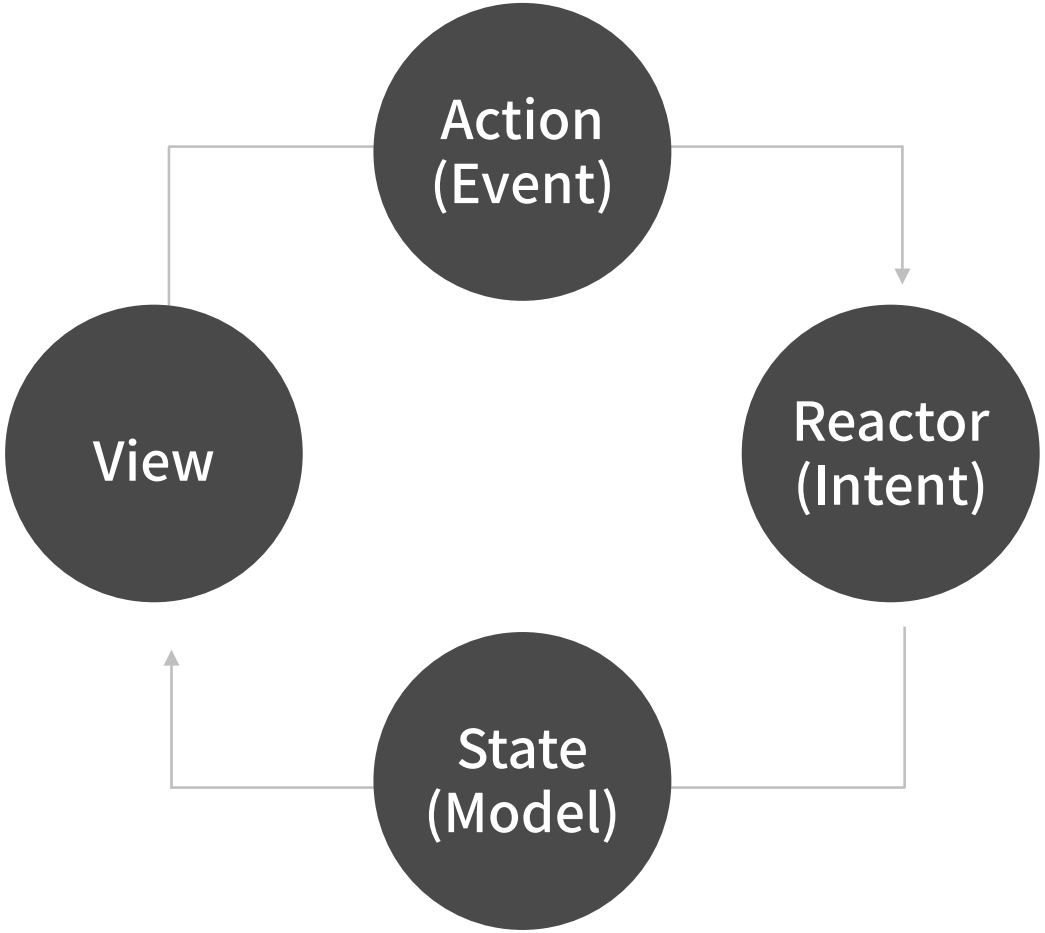

Model
Model을 입력 받아
상태를 화면에 출력
Intent
View
Business
Logic
Event
UI
User

MVI, ReactorKit 적용 과정

MVI, ReactorKit 적용 과정

ReactorKit

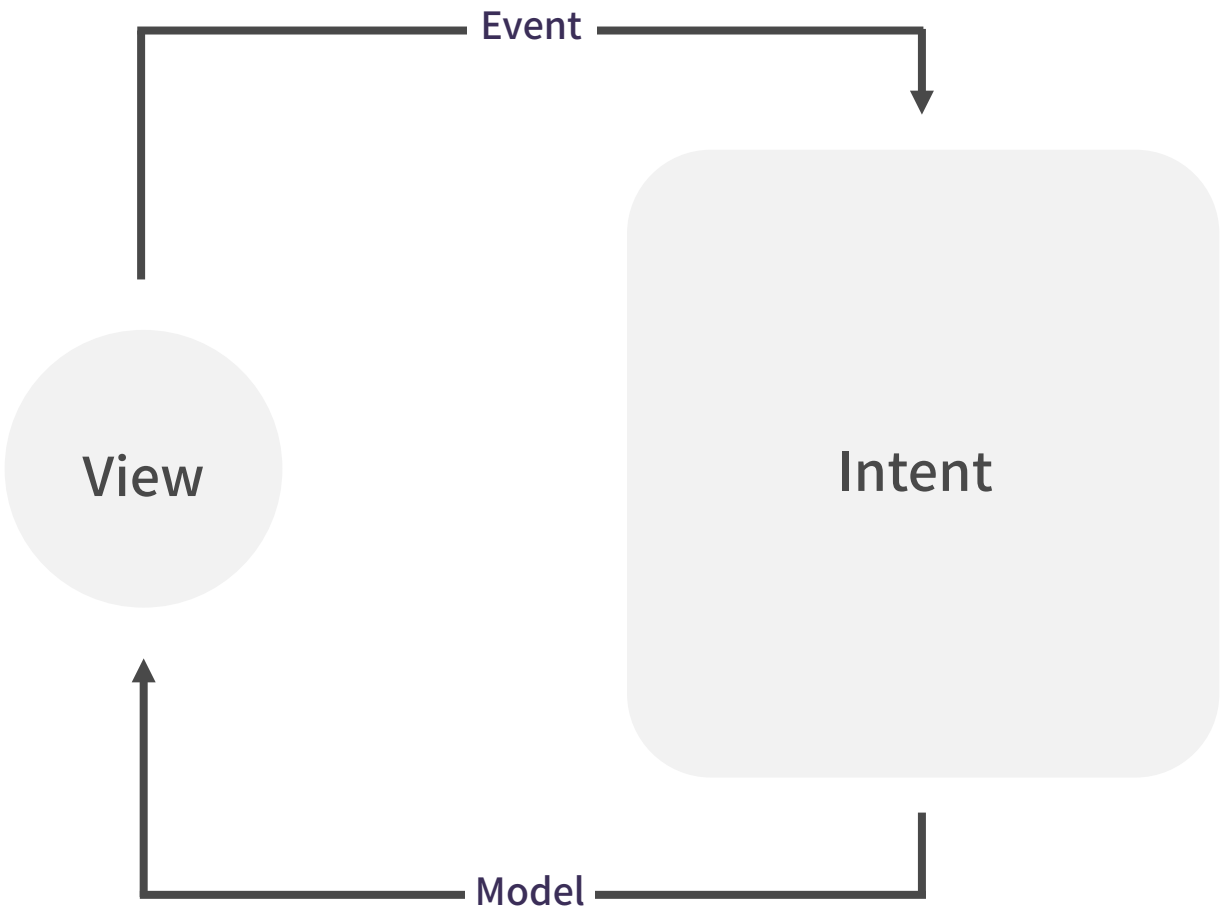
- 앱 아키텍처를 위한 프레임워크
- 단방향, 반응형(RxSwift)
- View, Reactor간의 관계 정의



출처: https://github.com/ReactorKit/ReactorKit

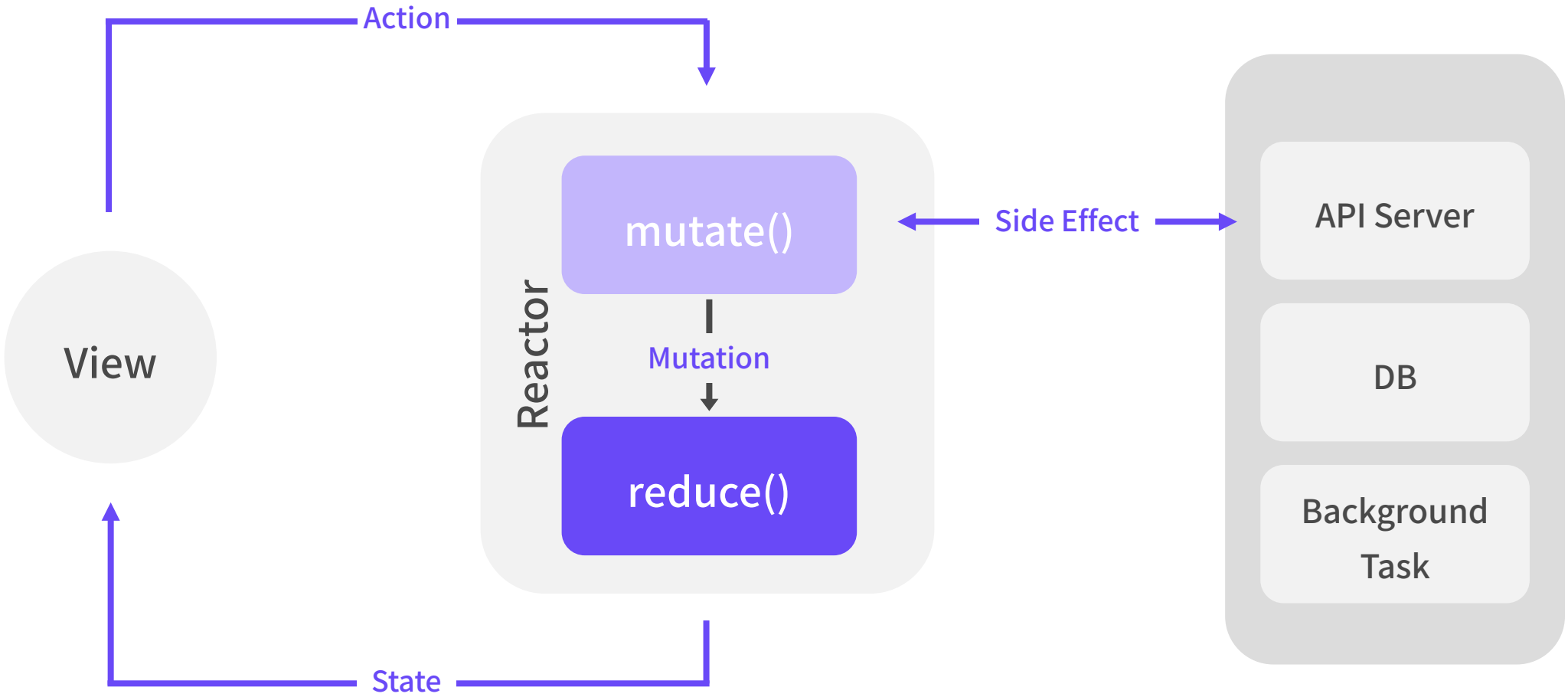
MVI, ReactorKit 적용 과정

MVI



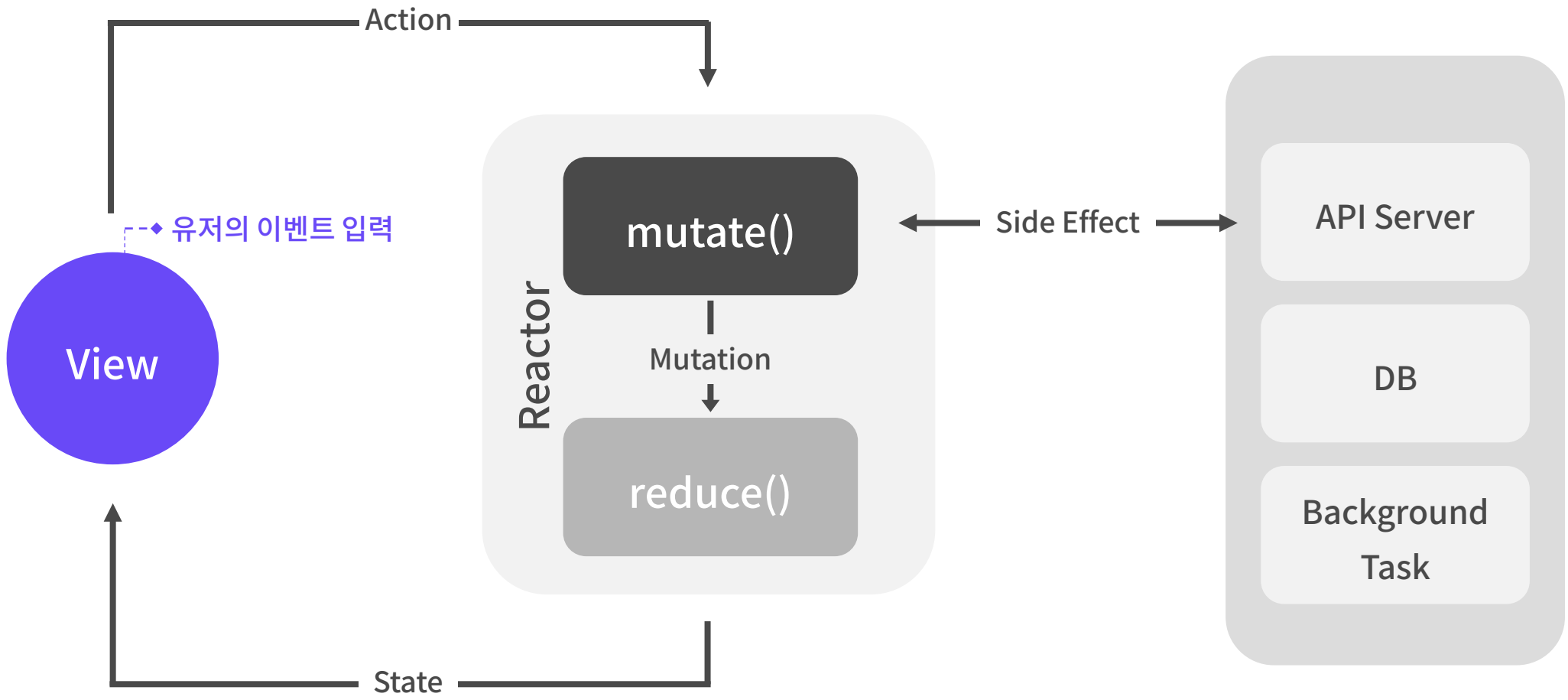
MVI, ReactorKit 적용 과정

ReactorKit



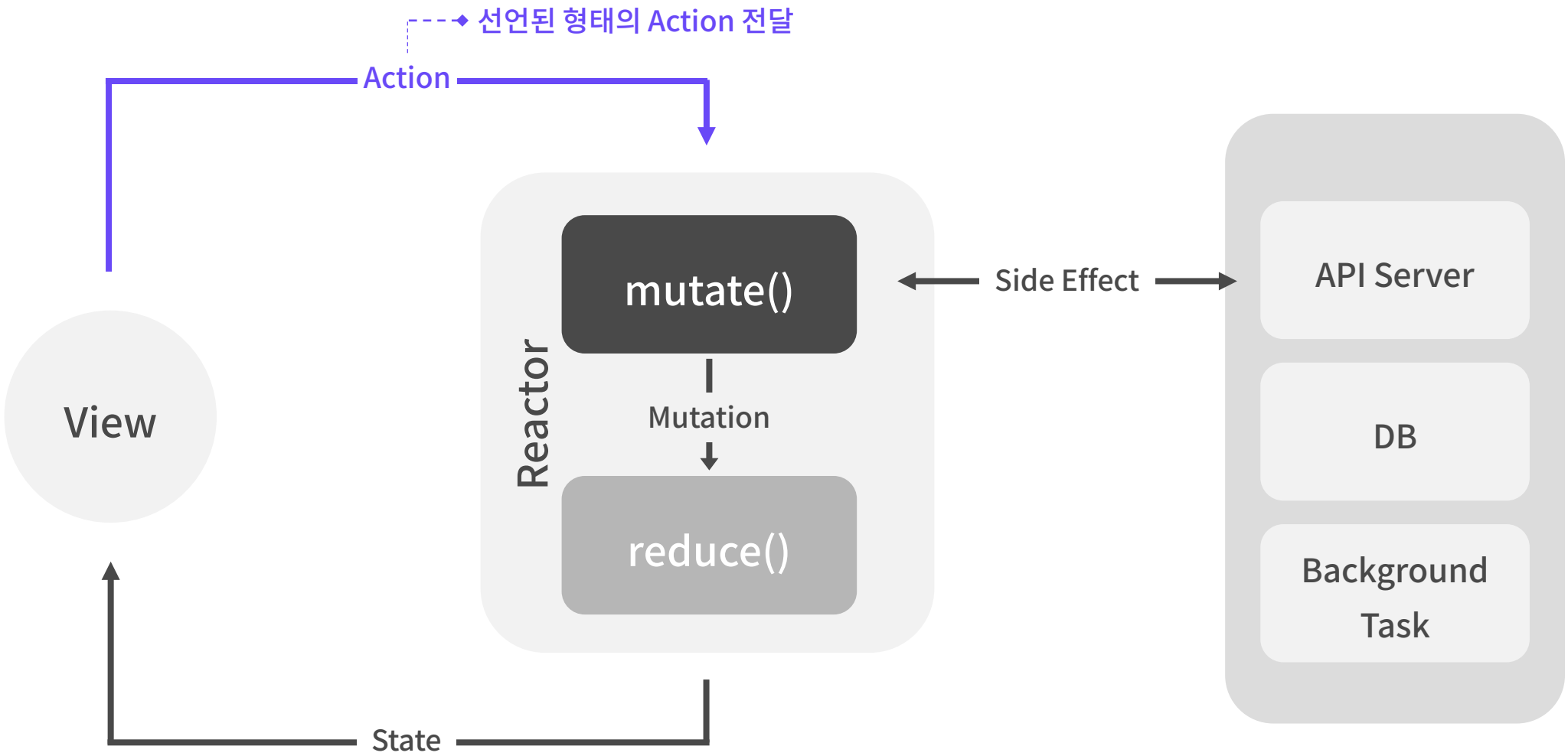
MVI, ReactorKit 적용 과정

ReactorKit



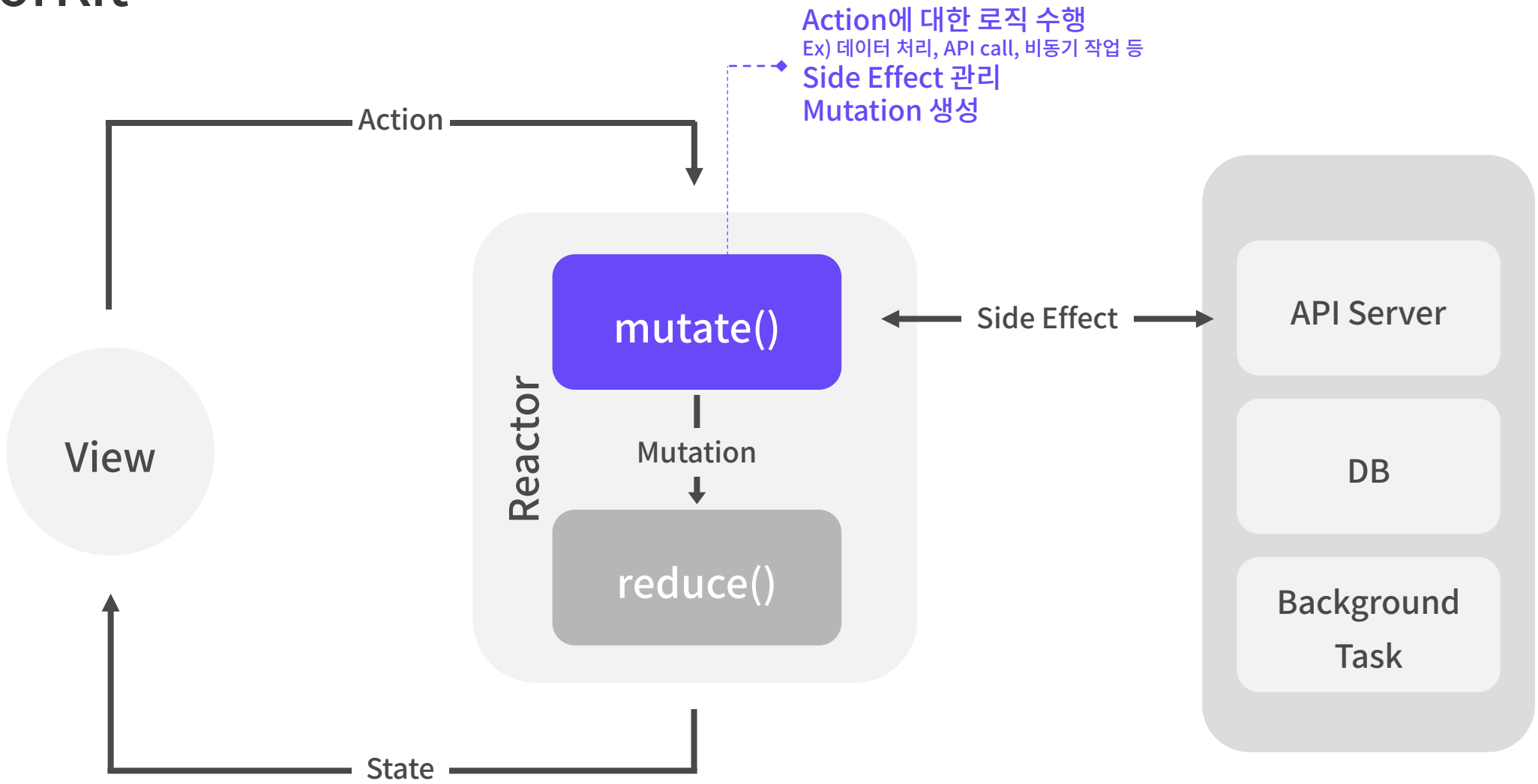
MVI, ReactorKit 적용 과정

ReactorKit



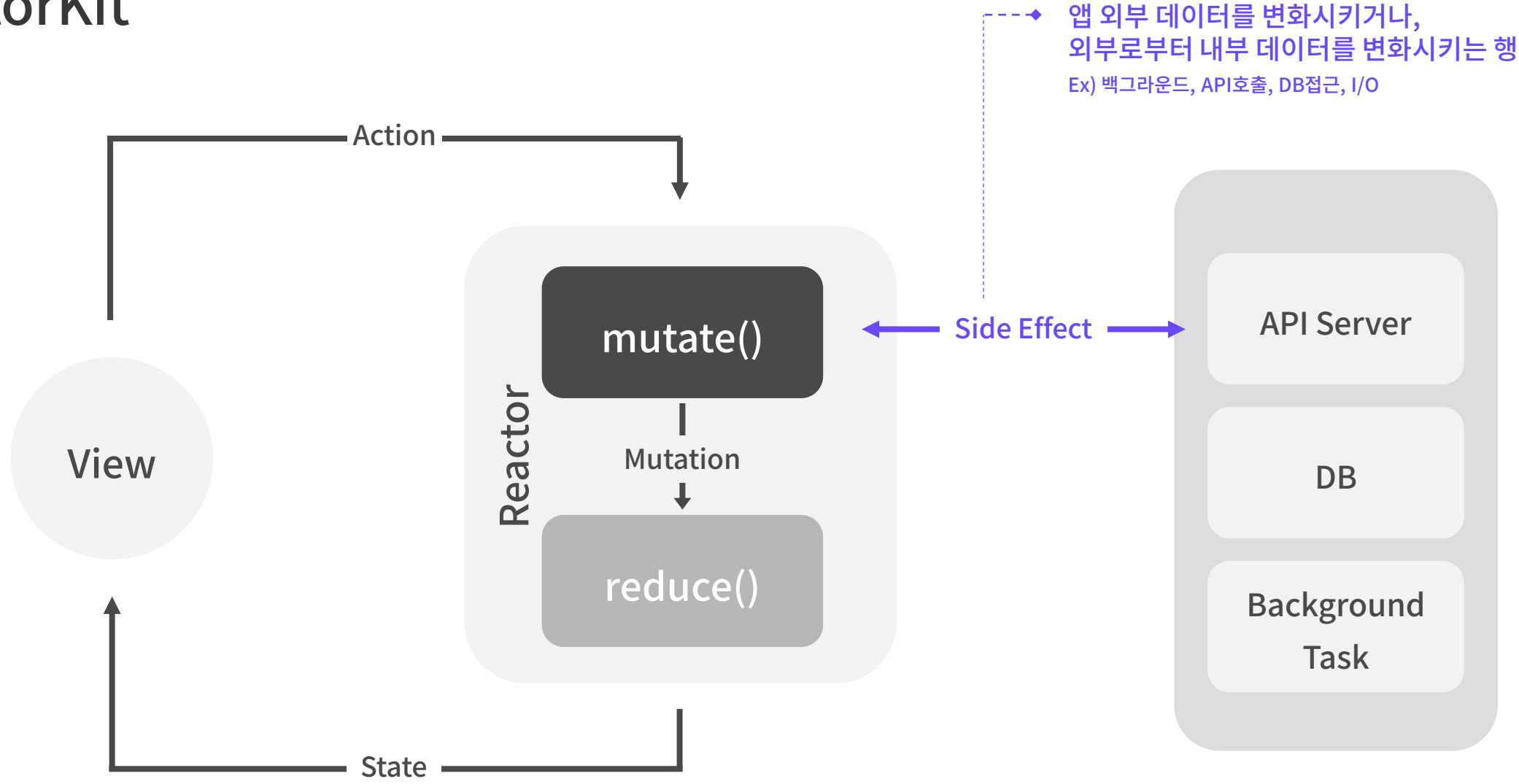
MVI, ReactorKit 적용 과정

ReactorKit



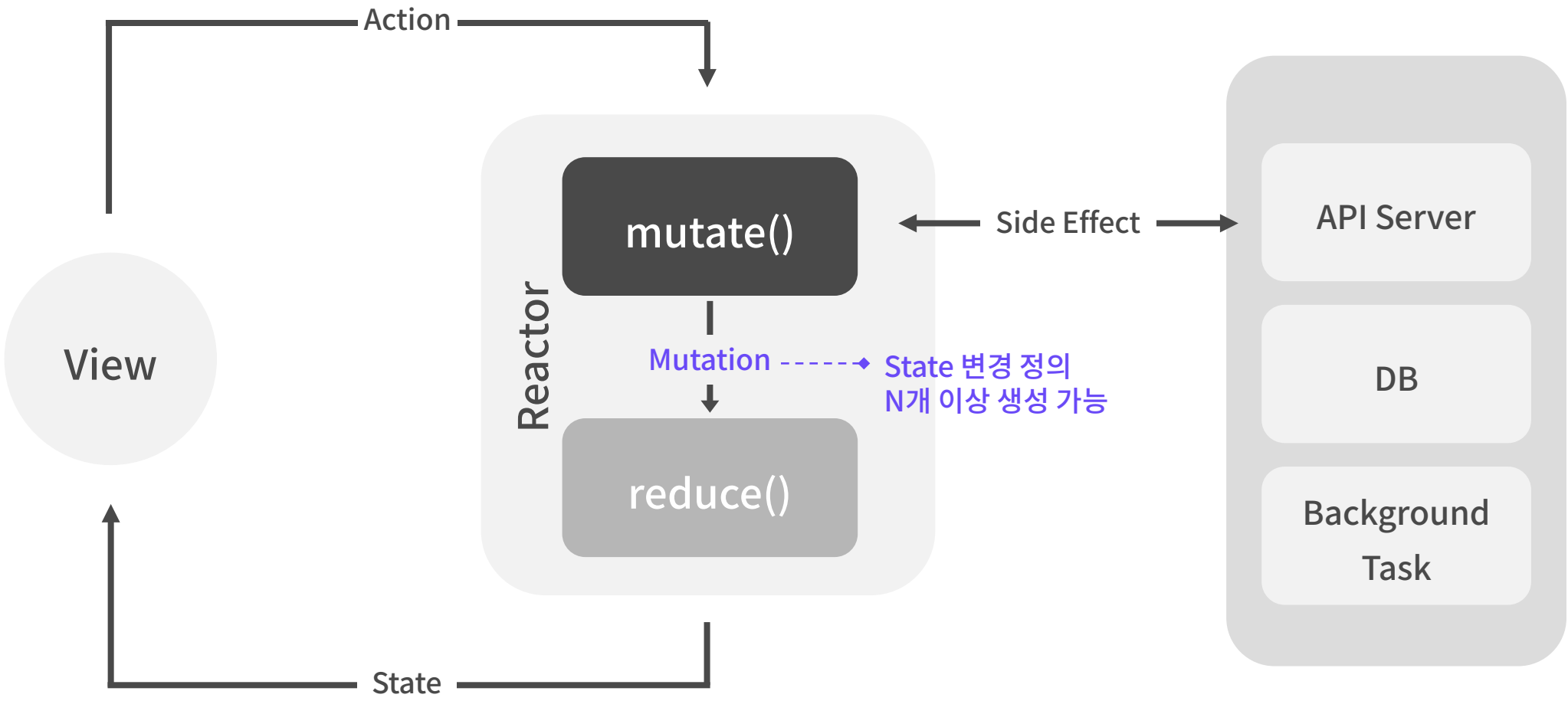
MVI, ReactorKit 적용 과정

ReactorKit



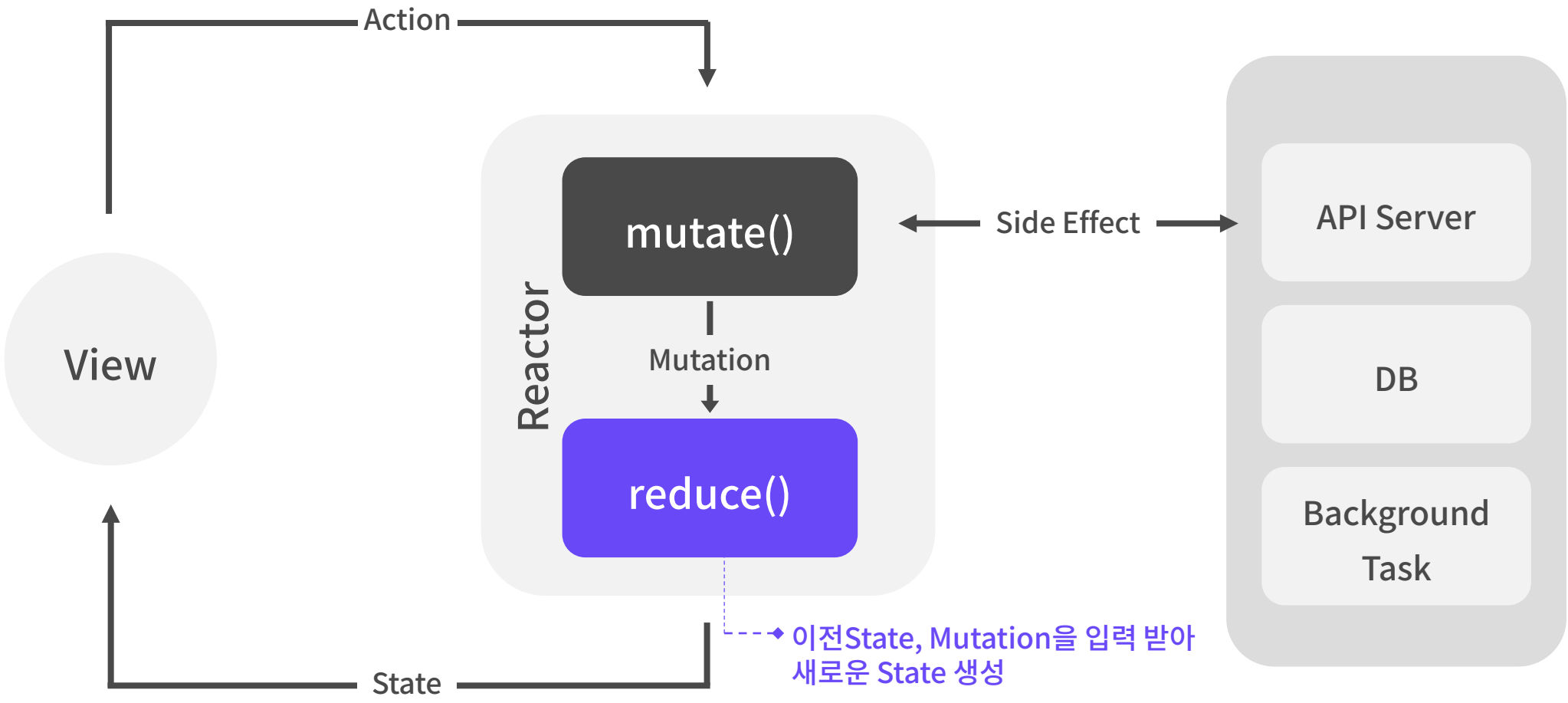
MVI, ReactorKit 적용 과정

ReactorKit



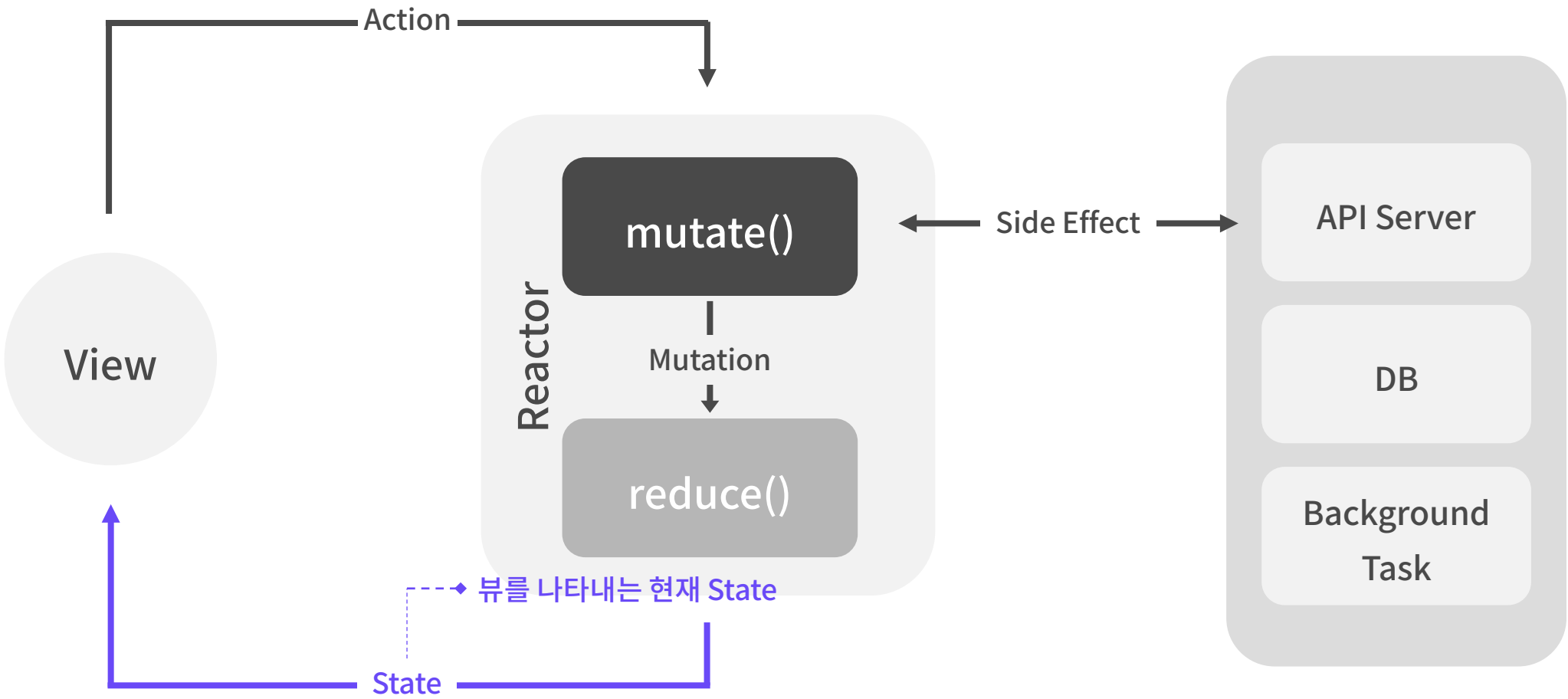
MVI, ReactorKit 적용 과정

ReactorKit



MVI, ReactorKit 적용 과정

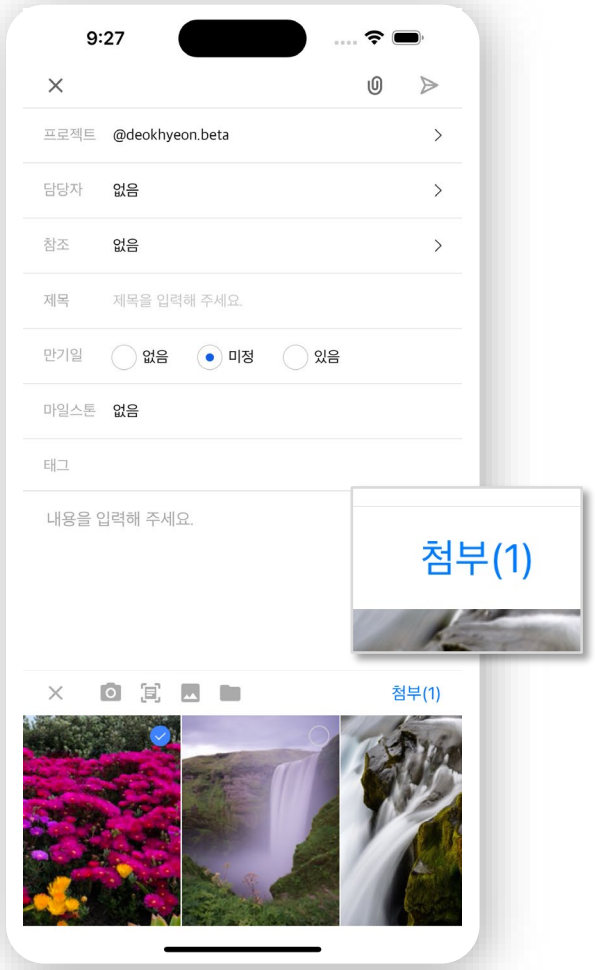
ReactorKit

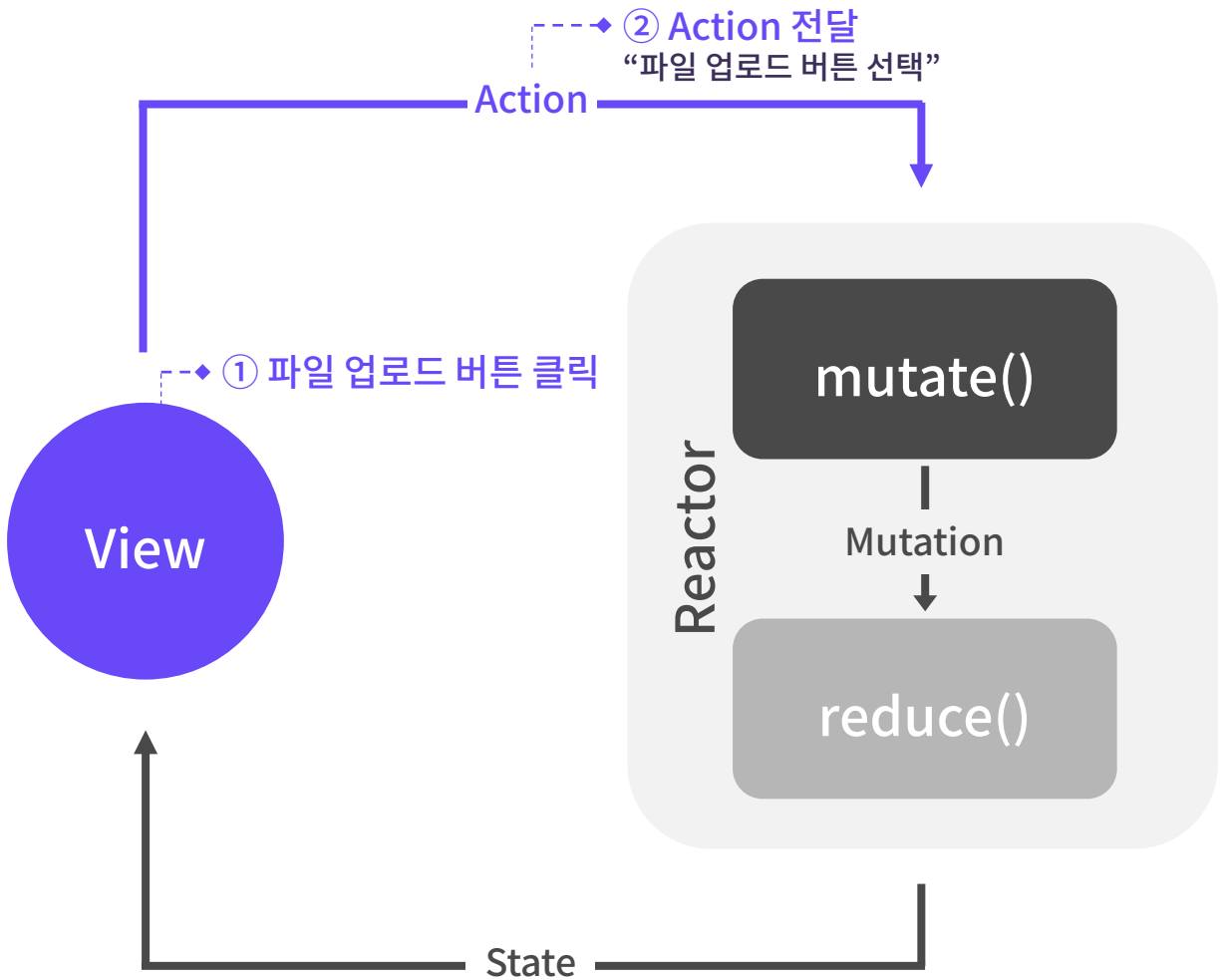


MVI, ReactorKit 적용 과정

NHN FORWARD

ReactorKit – 예시 > 파일 업로드

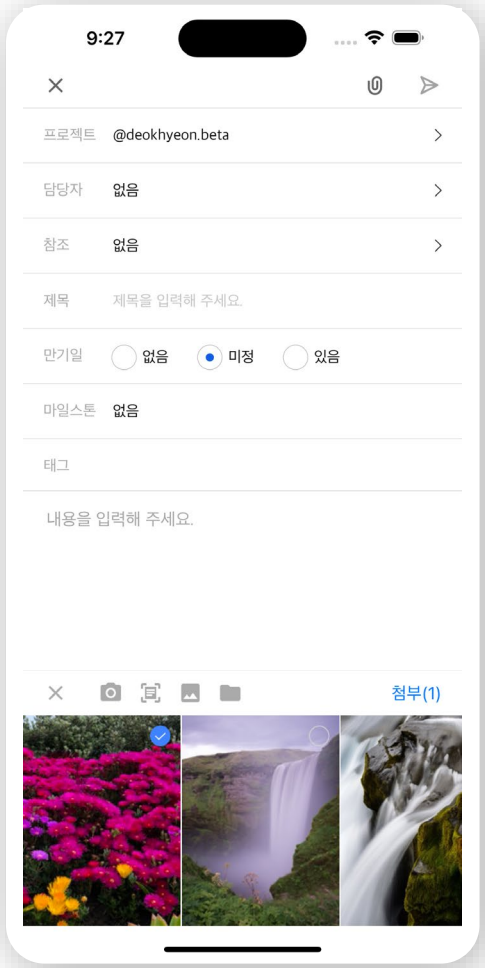


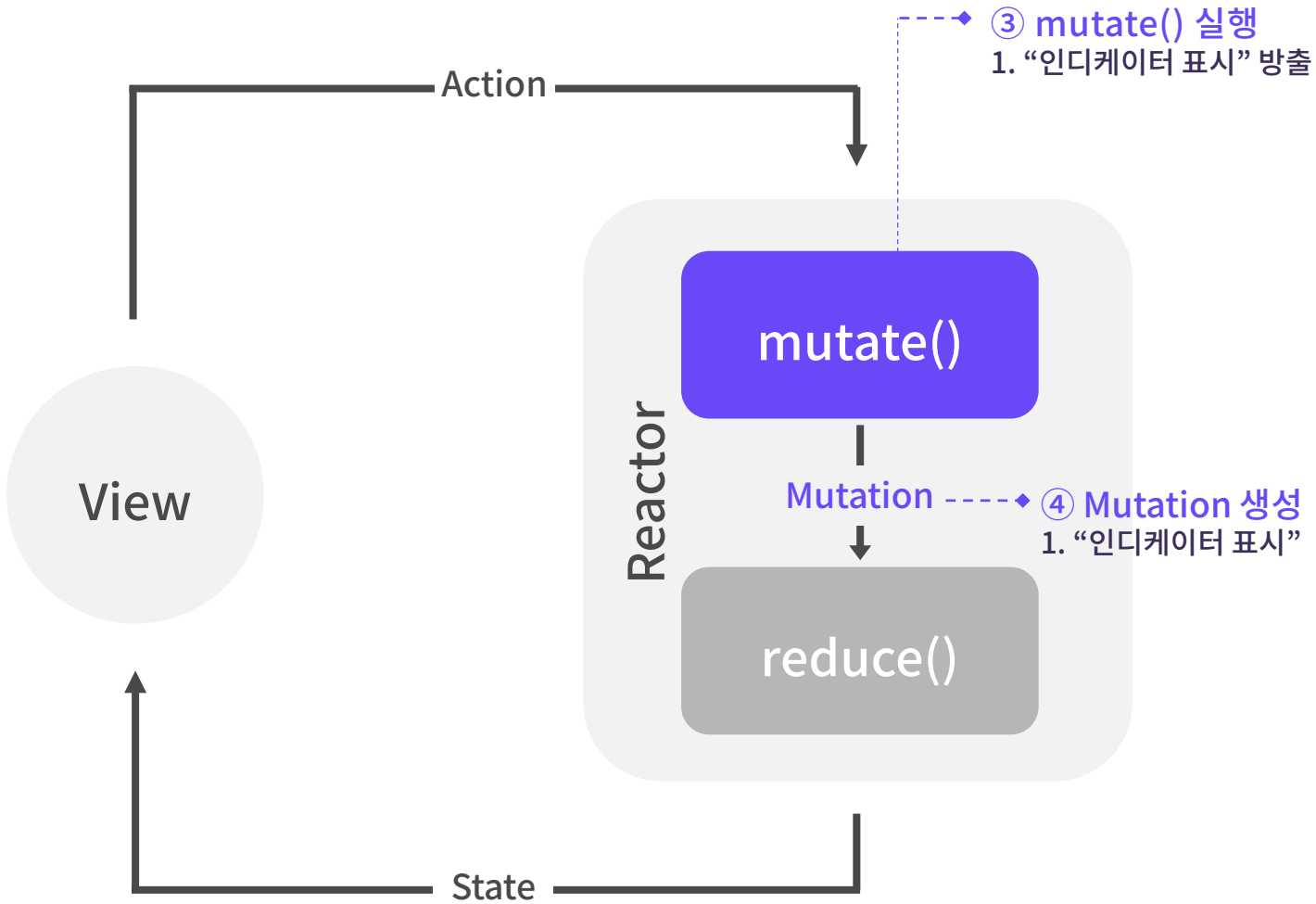


MVI, ReactorKit 적용 과정

NHN FORWARD

ReactorKit – 예시 > 파일 업로드

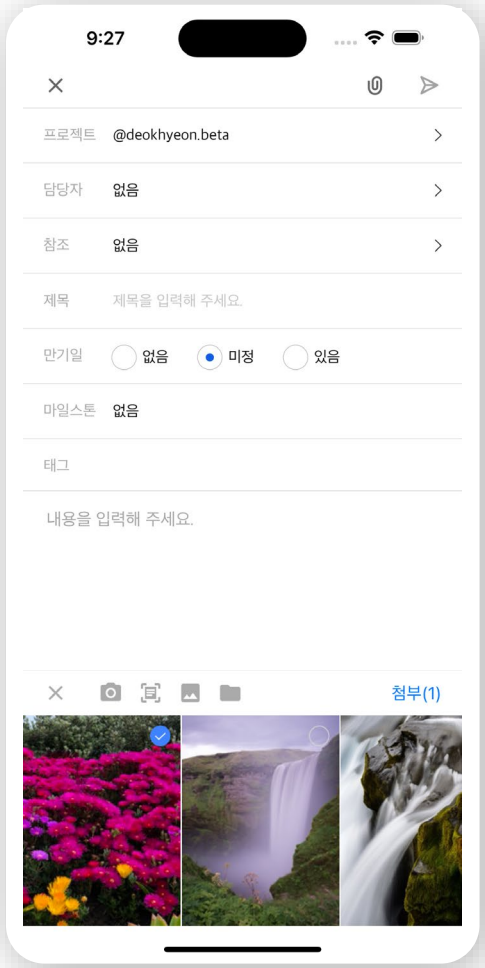


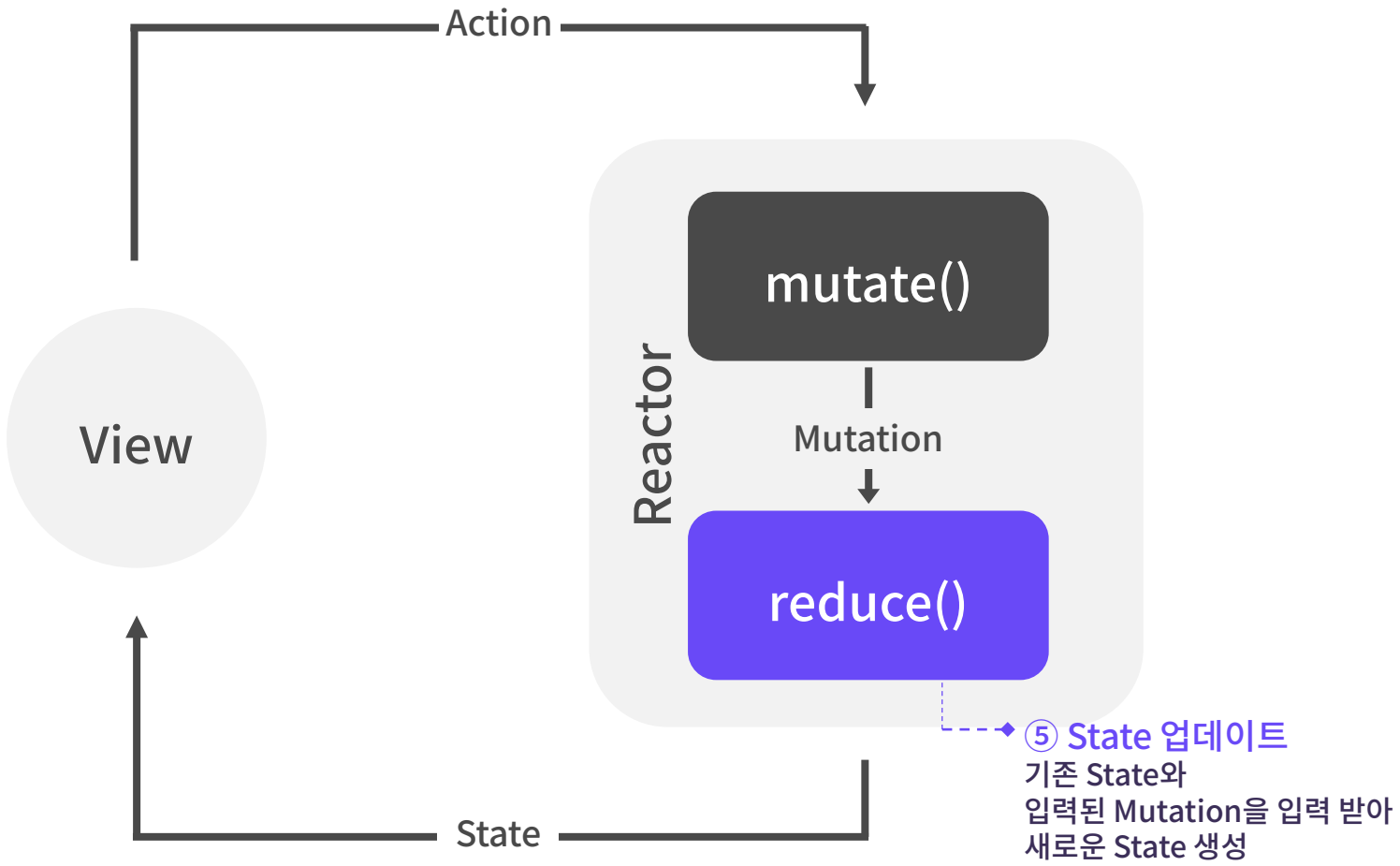


MVI, ReactorKit 적용 과정

NHN FORWARD

ReactorKit – 예시 > 파일 업로드

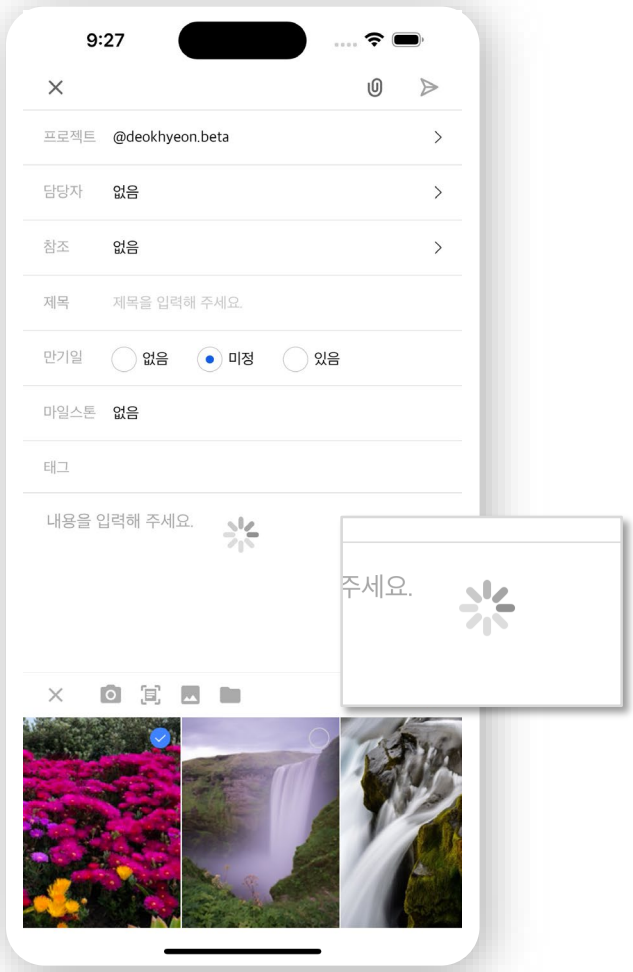


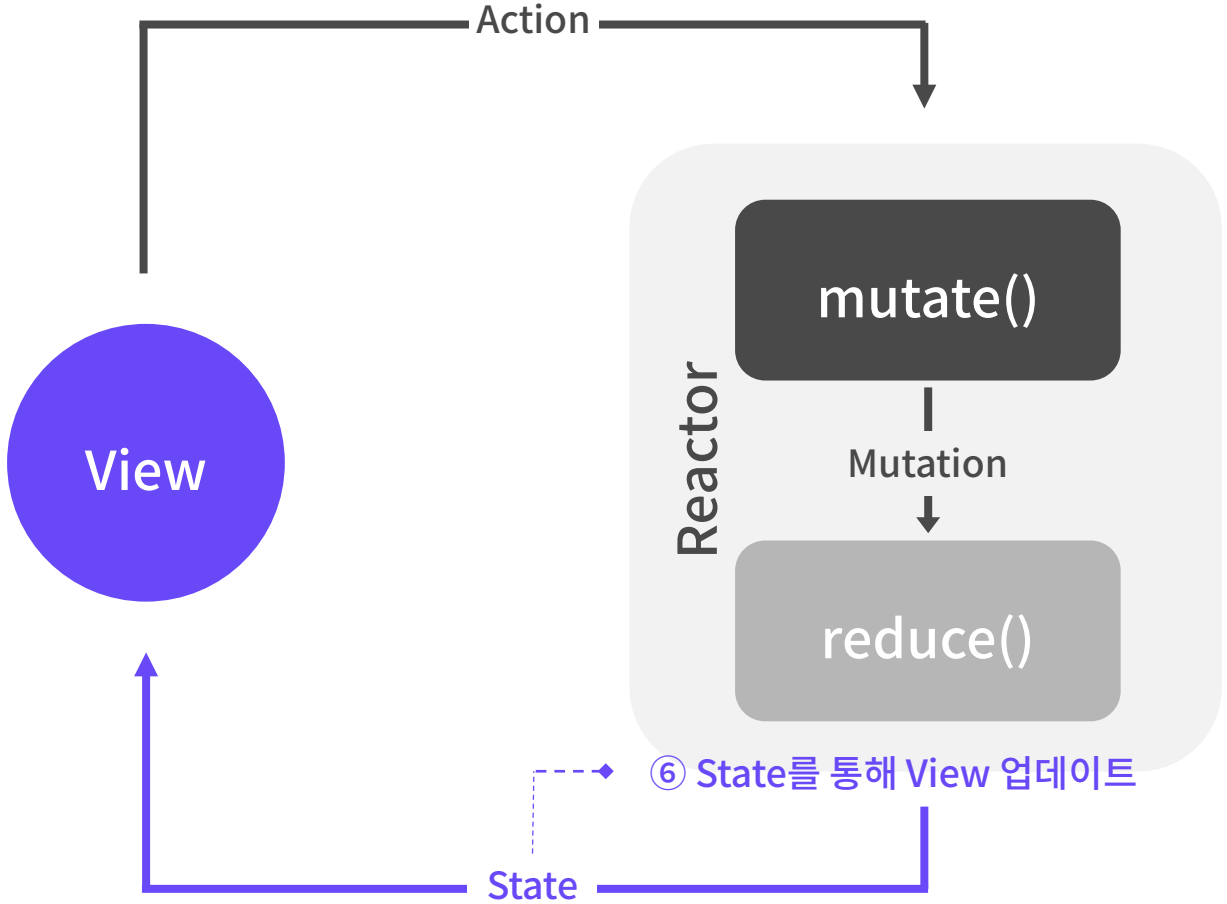


MVI, ReactorKit 적용 과정

NHN FORWARD

ReactorKit – 예시 > 파일 업로드

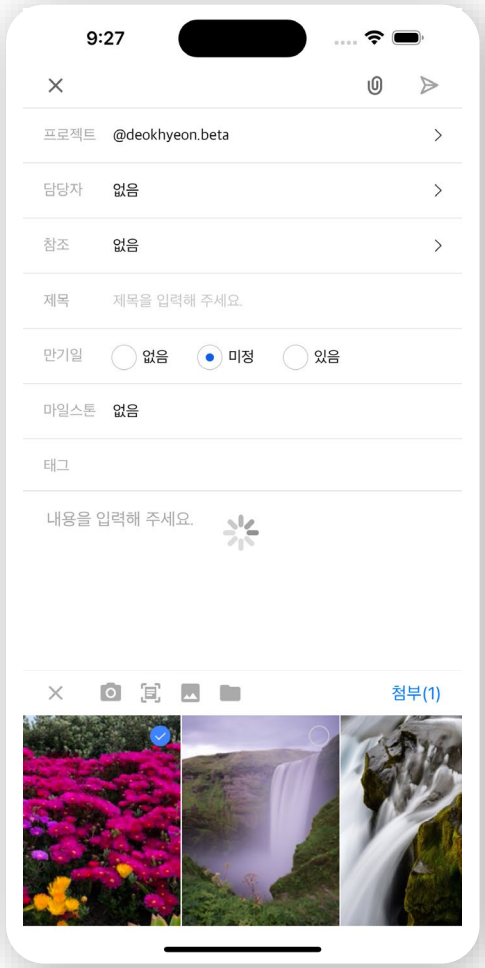


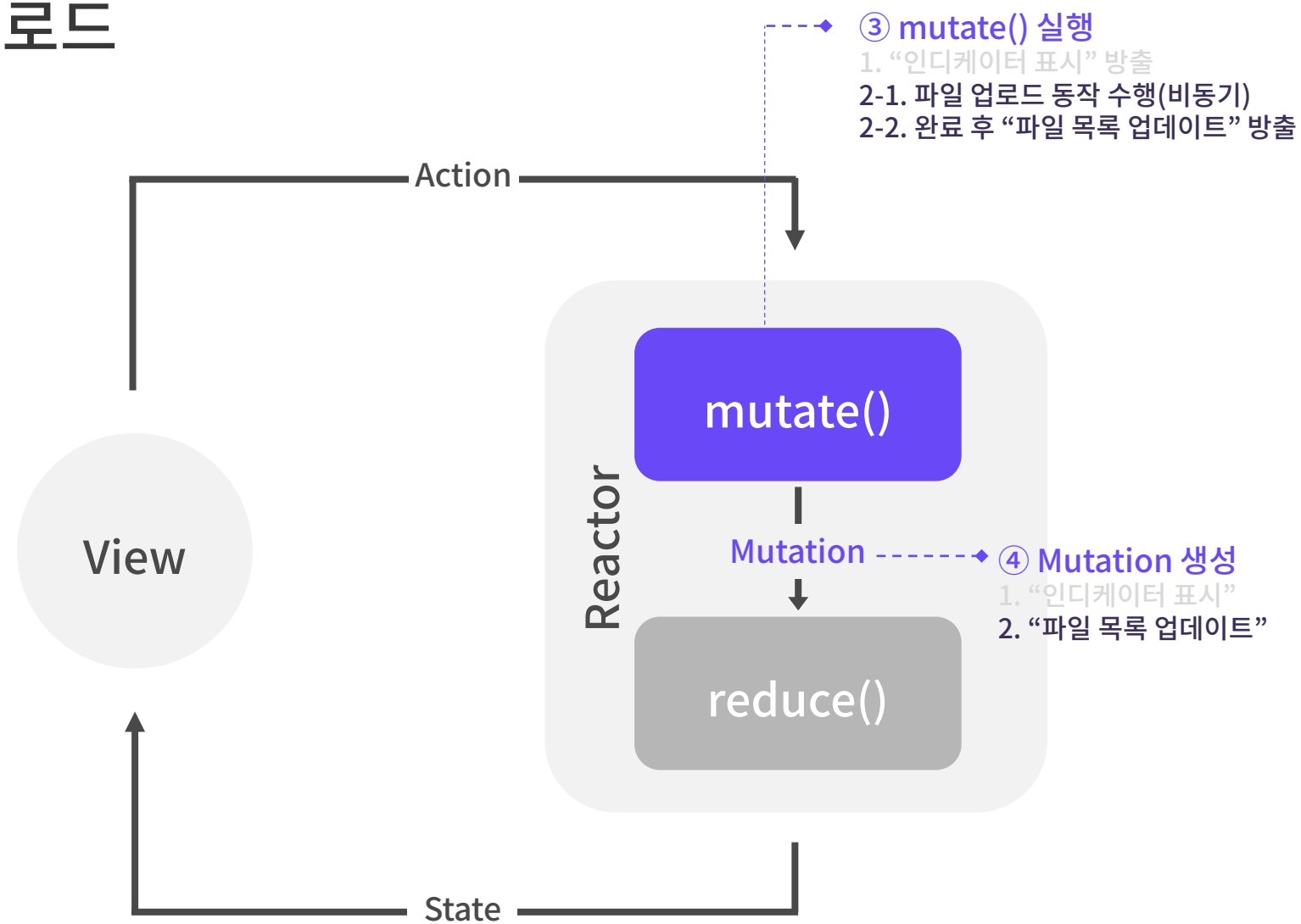


MVI, ReactorKit 적용 과정

NHN FORWARD

ReactorKit – 예시 > 파일 업로드

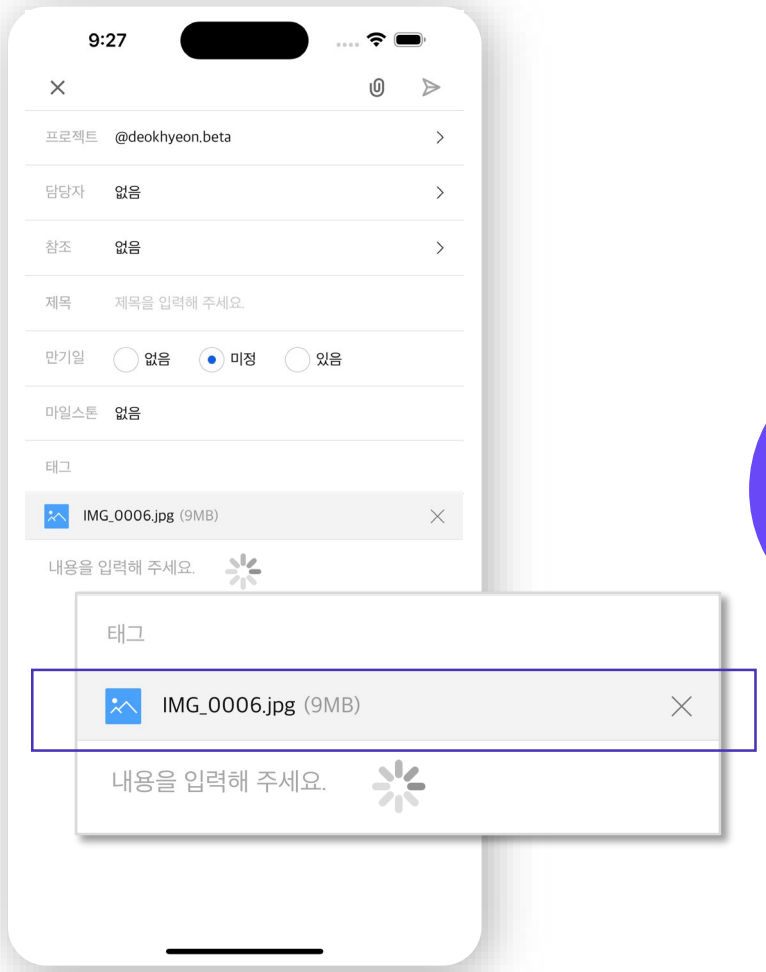


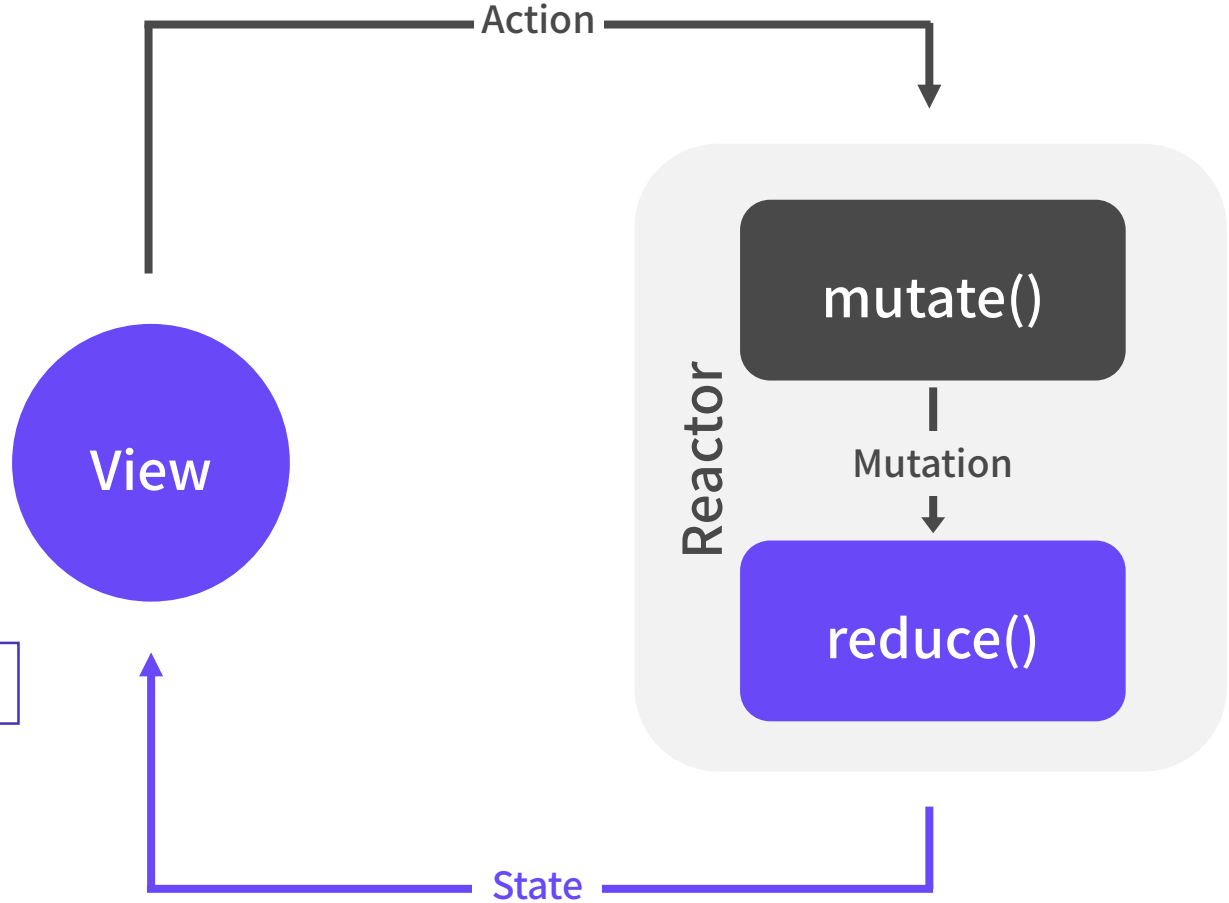


MVI, ReactorKit 적용 과정

NHN FORWARD

ReactorKit – 예시 > 파일 업로드

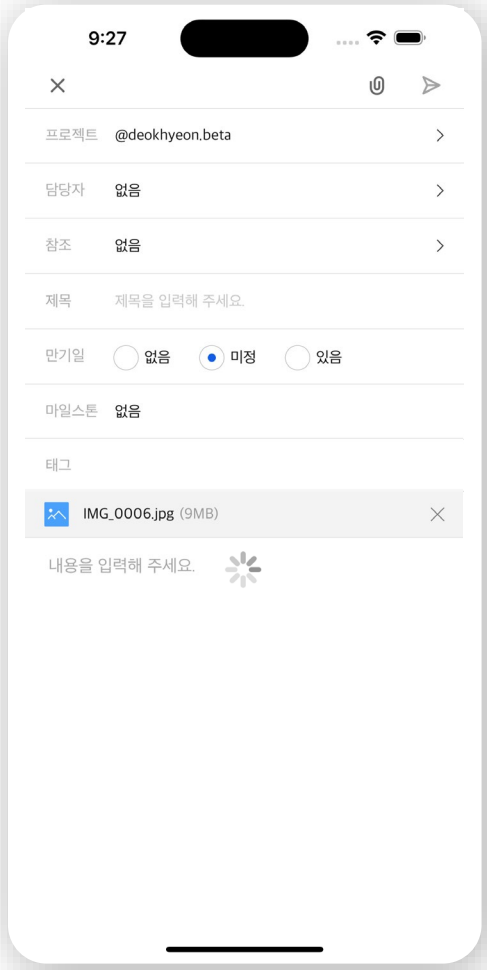


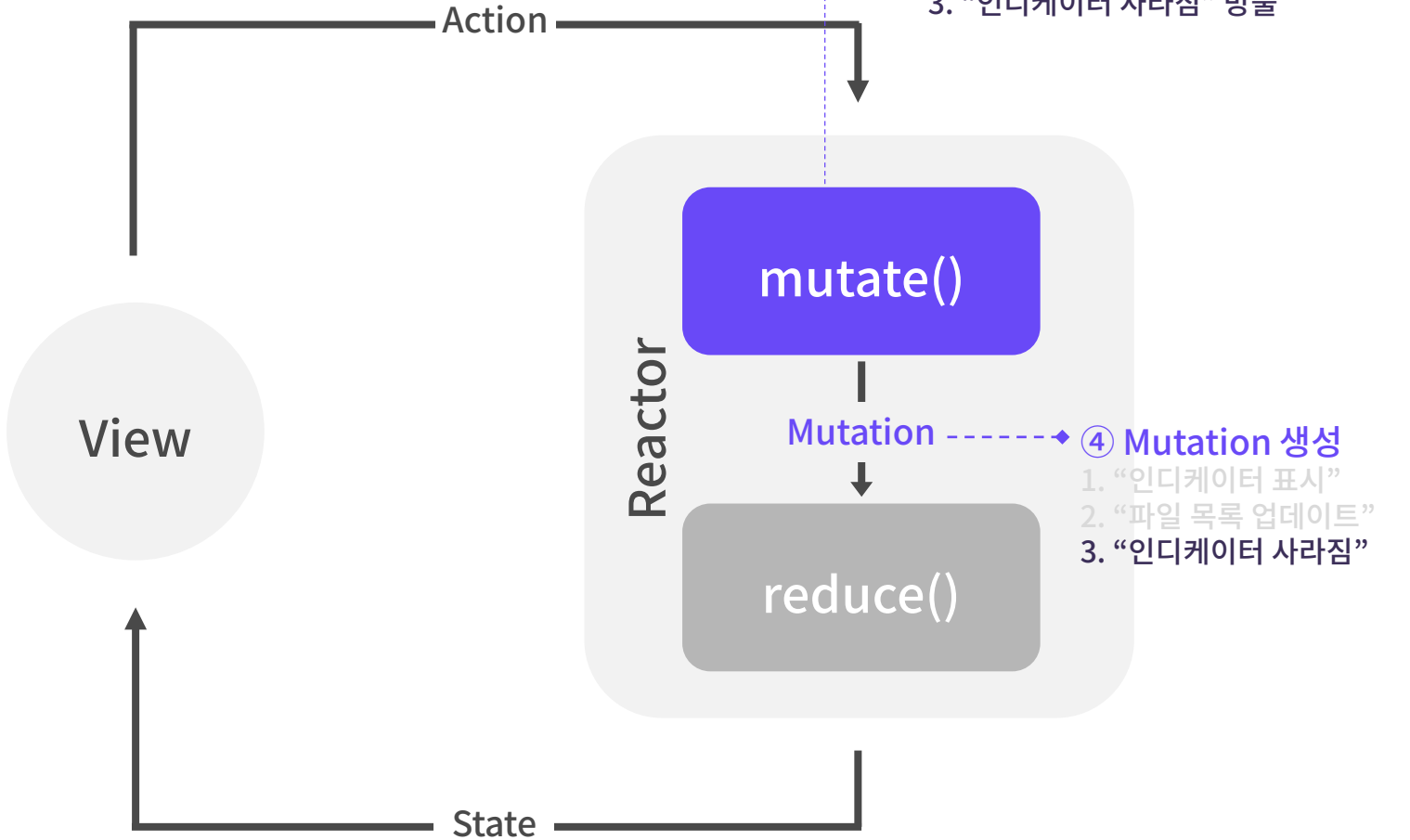


MVI, ReactorKit 적용 과정

NHN FORWARD

ReactorKit – 예시 > 파일 업로드

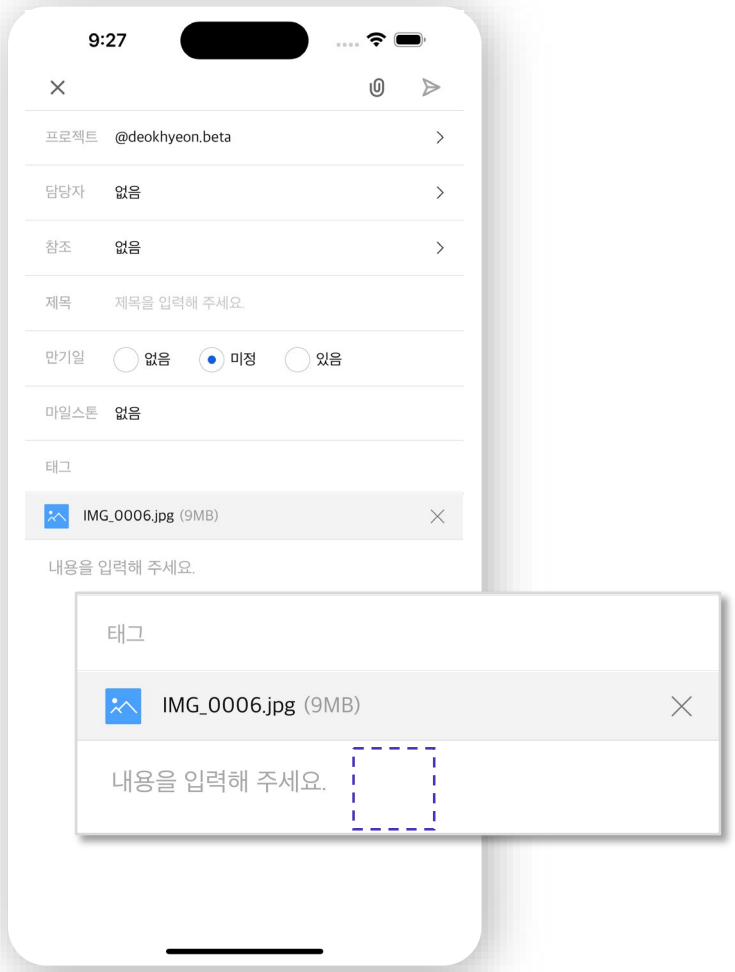


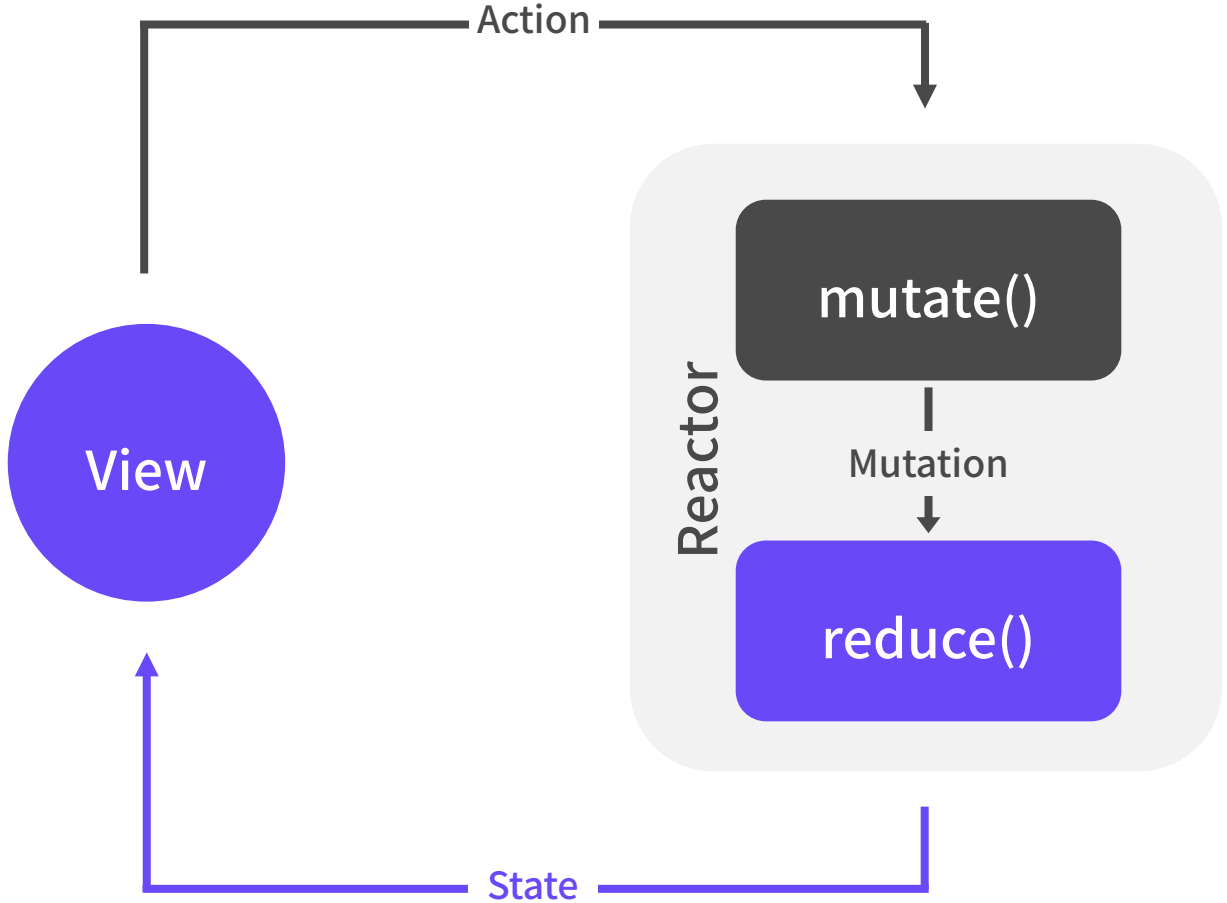


MVI, ReactorKit 적용 과정

NHN FORWARD

ReactorKit – 예시 > 파일 업로드





MVI, ReactorKit 적용 과정

ReactorKit

1. 파일 업로드 버튼 클릭
2. Action 전달

View.swift

```
func fileUpload(file: File) {
    reactor?.action.onNext(.selectedFileUpload(file))
}
```

Reactor.swift

```
enum Action {
    case selectedFileUpload(File) //파일 업로드
}
```

MVI, ReactorKit 적용 과정

ReactorKit

3-1. mutate() 실행

a. “인디케이터 표시” 방출

b. 파일 업로드 동작 수행(비동기)

c. 완료 후 “파일 목록 업데이트” 방출

d. “인디케이터 사라짐” 방출

Reactor.swift

```
func mutate(action: Action) -> Observable<Mutation> {
    switch action {
    case .selectFileUpload(let file):
        return .concat(.just(.setIndicator(true)),
                       uploadFile(file: file),
                       .just(.appendUploadedFiles(file)),
                       .just(.setIndicator(false)))
        .catch { error in
                .concat(.just(.showAlertMessage(error)),
                        .just(.setIndicator(false)))
        }
    }
}
```

MVI, ReactorKit 적용 과정

ReactorKit

3-2. mutate() 실행 – 에러 발생

a. “인디케이터 실행” 방출

b. 파일 업로드 동작 수행 - 에러

c. “얼럿 실행” 방출

d. “인디케이터 중지” 방출

Reactor.swift

```
func mutate(action: Action) -> Observable<Mutation> {
    switch action {
    case .selectFileUpload(let file):
        return .concat(.just(.setIndicator(true)),
                       uploadFile(file: file),
                       .just(.appendUploadedFiles(file)),
                       .just(.setIndicator(false)))
        .catch { error in
                .concat(.just(.showAlertMessage(error)),
                        .just(.setIndicator(false)))
        }
    }
}
```

MVI, ReactorKit 적용 과정

ReactorKit

4. 상태 업데이트

기존 State와 입력된 Mutation을
입력 받아 새로운 State 생성

Reactor.swift

```swift
func reduce(state: State, mutation: Mutation) -> State {
    var newState = state
    switch mutation {
    case .setIndicator(let show):
        newState.showIndicator = show
    case .showAlertMessage(let show):
        newState.showAlertMessage = show
    case .appendUploadedFiles(let file):
        var uploadedFiles = state.uploadedFiles ?? []
        uploadedFiles.append(file)
        newState.uploadedFiles = uploadedFiles
    }
    return newState
}
```

MVI, ReactorKit 적용 과정

ReactorKit

5. State를 통해 View 업데이트

Reactor.swift

```
struct State {
    var showIndicator: Bool?
    var showAlertMessage: String?
    var uploadedFiles: [File]?
}
```

MVI, ReactorKit 적용 과정

ReactorKit

5. State를 통해 View 업데이트

View.swift

```
func bind(reactor: Reactor) {
    reactor.state.map { $0.uploadedFiles }
        .distinctUntilChanged()
        .subscribe(onNext: { files in
            // Render
        }).disposed(by: disposeBag)

    reactor.state.map { $0.showIndicator }
        .distinctUntilChanged()
        .compactMap { $0 }
        .bind(to: indicator.rx.isAnimating)
        .disposed(by: disposeBag)
}
```

도입 후 개선 사항

도입 후 개선 사항

Dooray! 앱에서 MVI 패턴 사용을 통한 개선

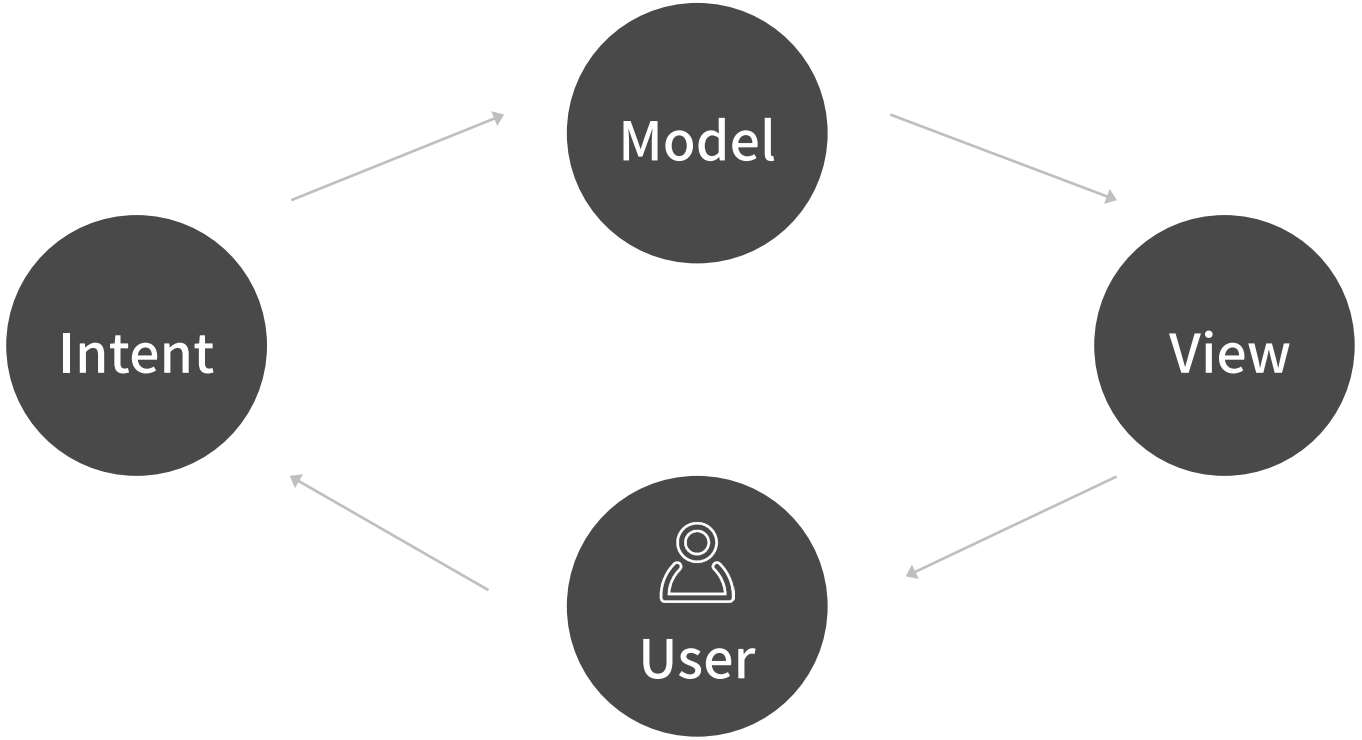
Dooray! 앱의 특성	MVI 사용을 통한 개선
복잡한 로직	단방향 데이터 흐름
다량의 데이터	단일 상태 관리
잦은 상태 변화	스레드 안정성

도입 후 개선 사항

NHN FORWARD

단방향 데이터 흐름

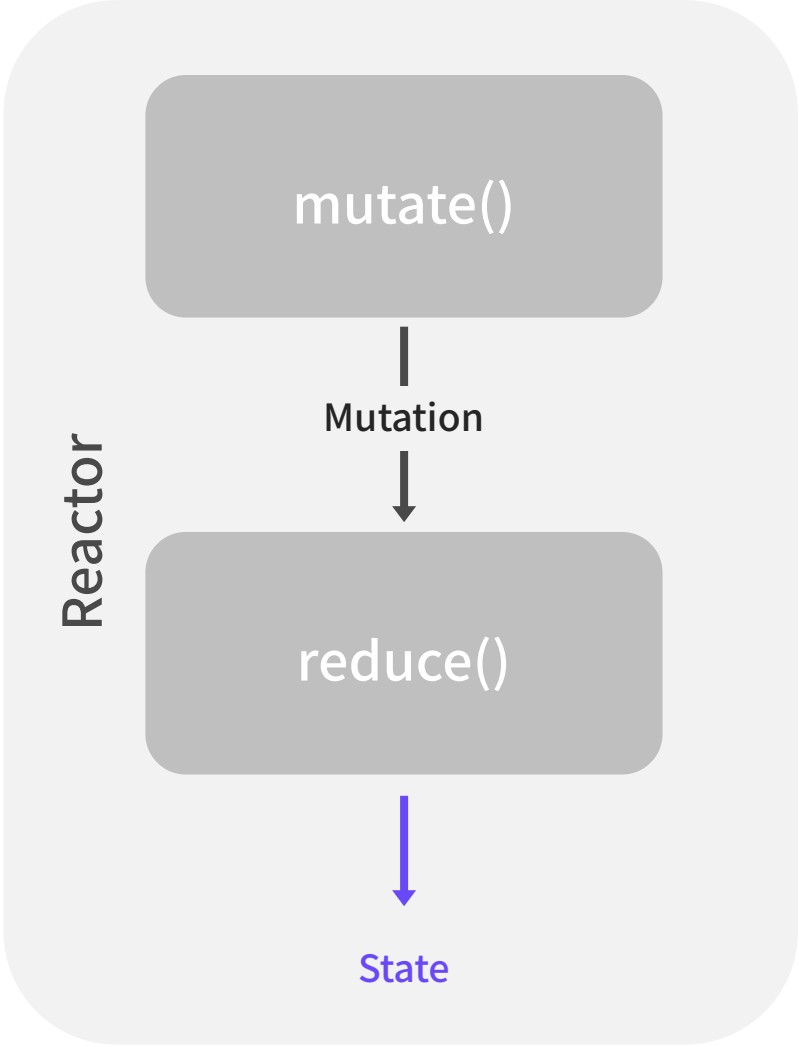
- Intent – Model - View
- 명확한 흐름
- 복잡한 로직의 단순화



도입 후 개선 사항

단일 상태 관리

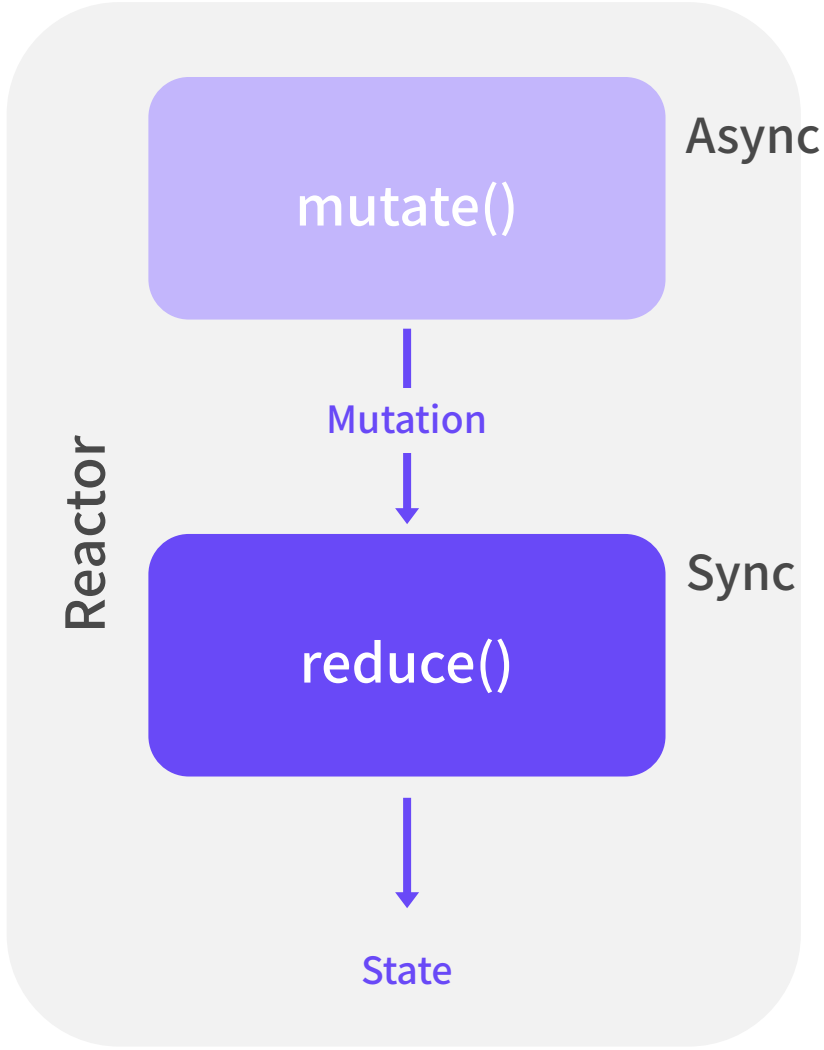
- View의 표현 명확
- View에서 State 직접 변경 불가
- 디버깅 용이
- 테스트 코드 작성 용이



도입 후 개선 사항

스레드 안정성

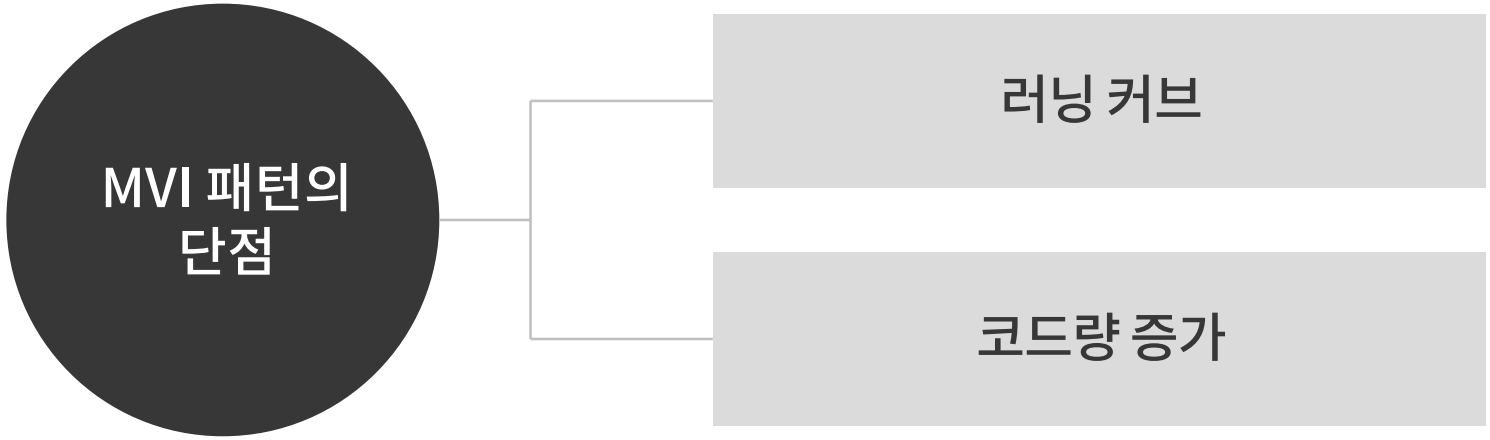
- mutate() 에서만 비동기 처리 가능
- reduce() 동기 처리
- reduce() 에서만 State 변경 가능



MVI 단점 및 고려 사항

MVI 단점

MVI 패턴의 단점

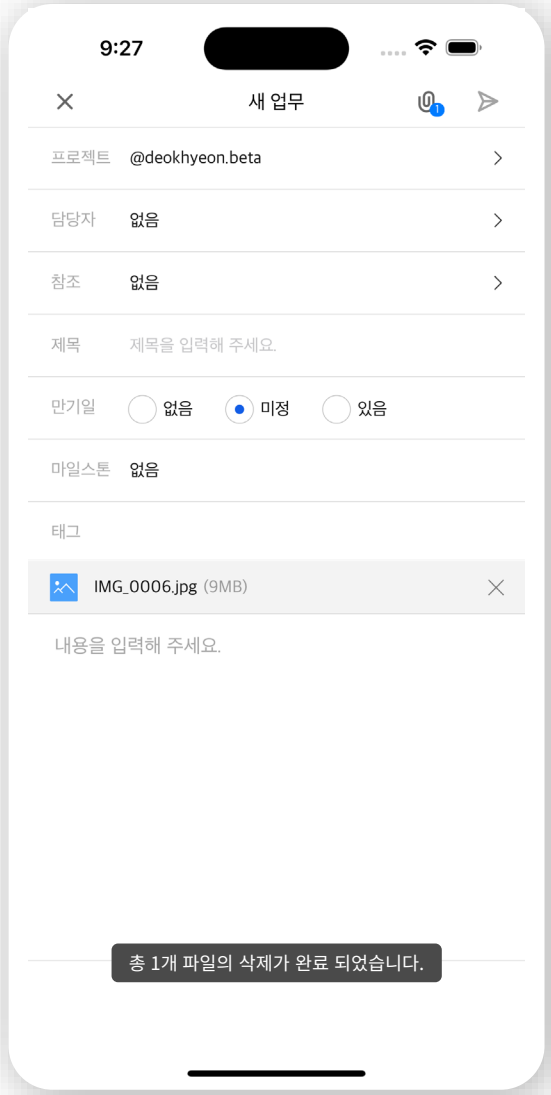


MVI 패턴 적용 시 고려 사항

NHN FORWARD

일회성 이벤트 처리

- 일회성 이벤트 처리 방법 모호함
  ex) Toast, Alert, Navigation



MVI 패턴 적용 시 고려 사항

일회성 이벤트 처리

- 일회성 이벤트 처리 방법 모호함
  ex) Toast, Alert, Navigation

```
func mutate(action: Action) -> Observable<Mutation
> {
    switch action {
    case .selectDeleteFile(let file):
        return .concat(deleteFile(file: file)
                       //Show toast
        )
    default:
        return .empty()
    }
}
```

MVI 패턴 적용 시 고려 사항

일회성 이벤트 처리

- 일회성 이벤트 처리 방법 모호함
  ex) Toast, Alert, Navigation
- State, Side Effect 로 관리 …

```
func mutate(action: Action) -> Observable<Mutation
> {
    switch action {
    case .selectDeleteFile(let file):
        return .concat(deleteFile(file: file)
                       //Show toast
        )
    default:
        return .empty()
    }
}
```

MVI 패턴 적용 시 고려 사항

일회성 이벤트 처리 – State로 관리

- 일회성 이벤트 처리 방법 모호함
  ex) Toast, Alert, Navigation
- State, Side Effect 로 관리 …

```
func mutate(action: Action) -> Observable<Mutation
> {
    switch action {
    case .selectDeleteFile(let file):
        return .concat(deleteFile(file: file),
                       .just(.showToast(true))
                       .delay(5),
                       .just(.showToast(false)))
    default:
        return .empty()
    }
}
```

```
struct State {
    var showToast: Bool?
}
```

MVI 패턴 적용 시 고려 사항

일회성 이벤트 처리 – Side Effect로 관리

- 일회성 이벤트 처리 방법 모호함
  ex) Toast, Alert, Navigation
- State, Side Effect 로 관리 …
- Dooray! 앱에서는 Side Effect 로 관리

```
func mutate(action: Action) -> Observable<Mutation
> {
    switch action {
    case .selectDeleteFile(let file):
        return .concat(deleteFile(file: file),
                       showToast()
        )
    default:
        return .empty()
    }
}
```

```
func showToast() -> Observable<Mutation> {
    Toaster.show(sec: toastDelay)
    return .empty()
}
```

MVI 패턴 적용 시 고려 사항

일회성 이벤트 처리 – Side Effect로 관리

- 일회성 이벤트 처리 방법 모호함
  ex) Toast, Alert, Navigation
- State, Side Effect 로 관리 …
- Dooray! 앱에서는 Side Effect 로 관리

```
func mutate(action: Action) -> Observable<Mutation
> {
    switch action {
    case .selectDeleteFile(let file):
        return .concat(deleteFile(file: file),
                       showToast()
        )
    case .selectedReceiver:
        coordinator?.navigation(type: .selectRecei
verView)
        return .empty()
    default:
        return .empty()
    }
}
```

Q & A

NHN FORWARD

고맙습니다.