Firebase + Combine + SwiftUI

소개

Swift (UIkit)

RxSwift

SwiftUI
+
Combine

반응형, 선언형

RxSwift

SwiftUI
+
Combine

Combine

Why

iOS 및 iPadOS 사용 현황

2022년 1월 11일에 App Store에서 처리된 결과를 디바이스들에서 측정한 수치입니다.

iPhone

지난 4년 동안 도입된 기기의 72%가 iOS 15를 사용하고 있습니다.

iOS 15

- 72% iOS 15
- 26% iOS 14
- 2% 이전 버전

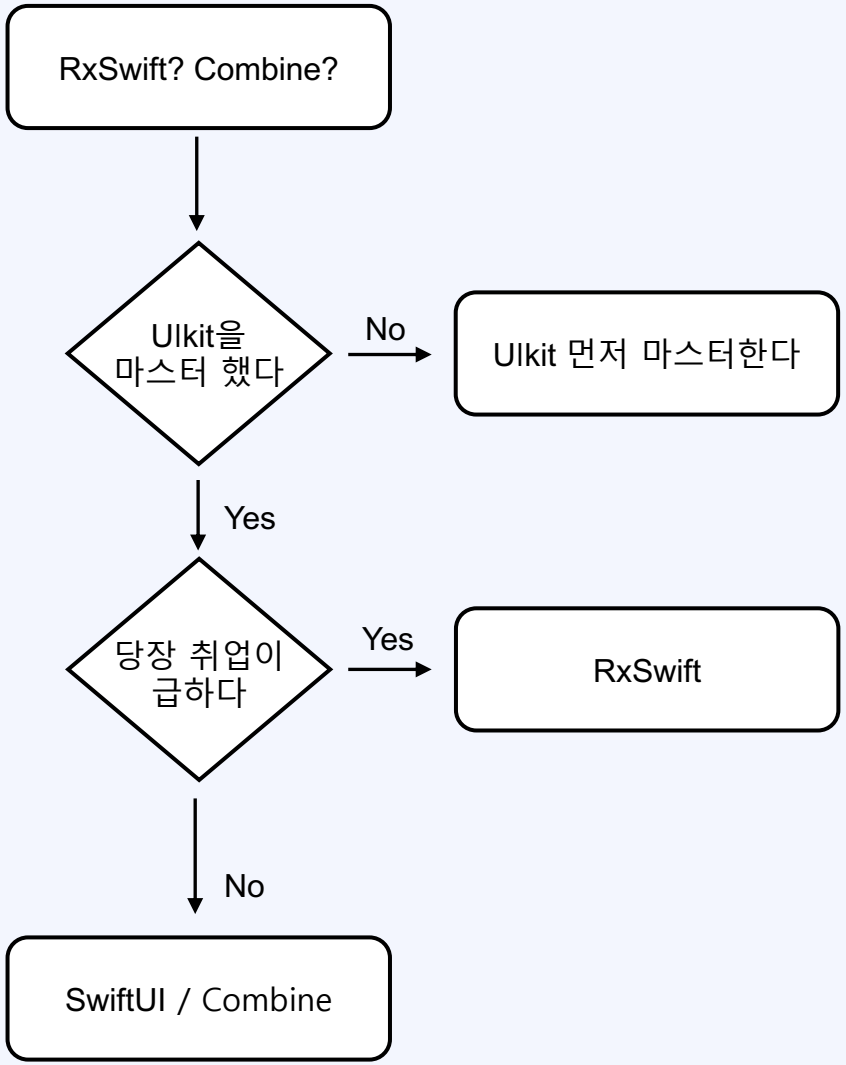
63%의 기기가 iOS 15를 사용하고 있습니다.

iOS 15

- 63% iOS 15
- 30% iOS 14
- 7% 이전 버전

iOS 점유율 : https://developer.apple.com/kr/support/app-store/

정리


RxSwift? Combine?
UIkit을
마스터 했다
No
UIkit 먼저 마스터한다
Yes
당장 취업이
급하다
Yes
RxSwift
No
SwiftUI / Combine

SwiftUI

Combine

Combine

Customize handling

of **asynchronous events**

by **combining event-processing operators**

Combine

Combine declares

Publishers **to expose values that can change over time,** and

Subscribers **to receive those values from the** publishers.

Combine

By adopting Combine, you'll **make your code easier tor read and maintain,**

by **centralizing** your event-processing code

and **eliminating** troublesome techniques

like nested closures and convention-based callbacks

Why Combine

비동기적인 인터페이스

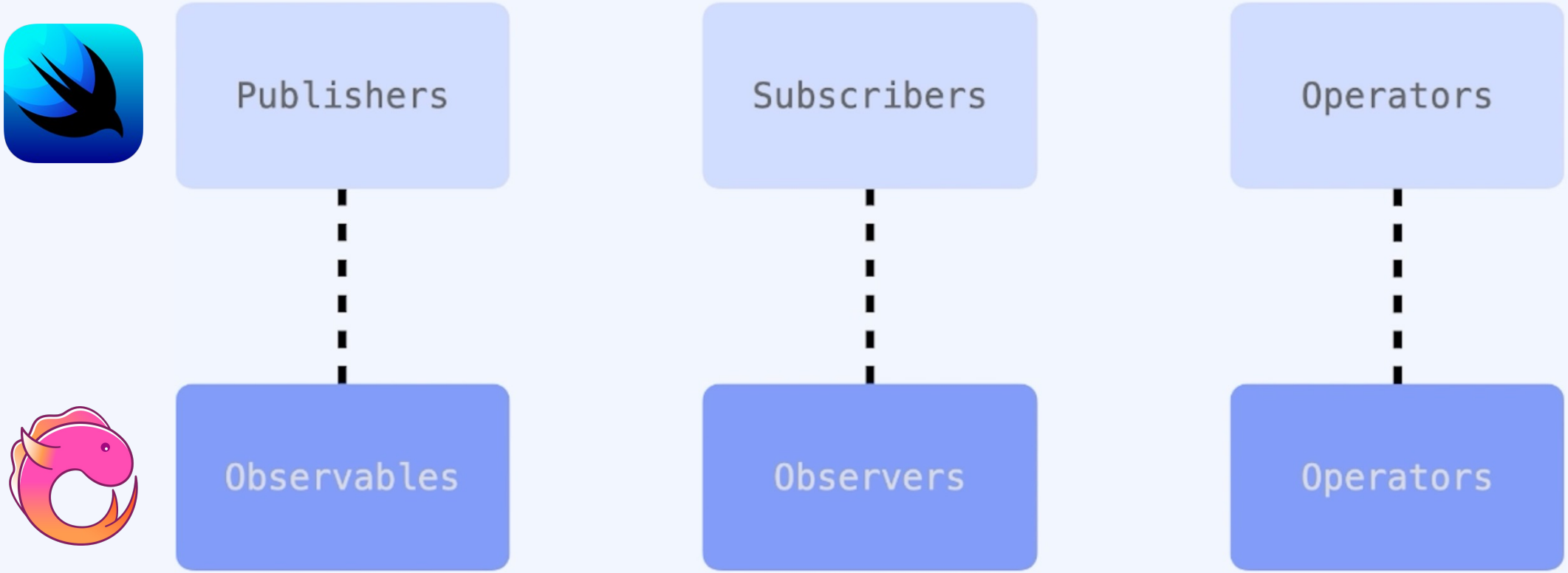
- IBTarget / IBAction
- Notification Center
- URLSession
- KVO
- Ad-hoc callbacks

Combine

A unified declarative API for processing values over time

Combine 핵심 요소

Combine 핵심 요소



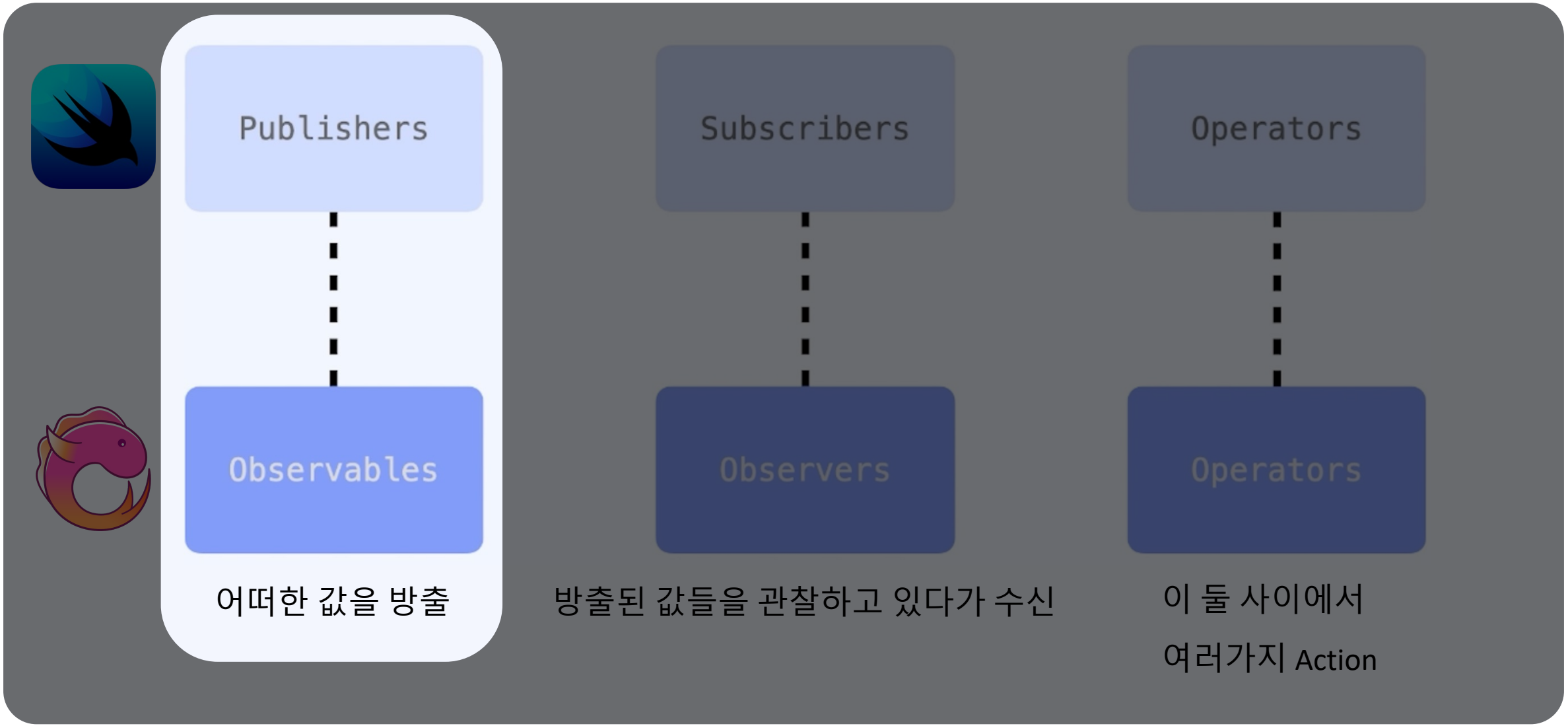
어떠한 값을 방출

방출된 값들을 관찰하고 있다가 수신

이 둘 사이에서

여러가지 Action

Publisher


Publishers
Subscribers
Operators
Observables
Observers
Operators
어떠한 값을 방출
방출된 값들을 관찰하고 있다가 수신
이 둘 사이에서
여러가지 Action

Publisher

- AnyPublisher

```
public protocol Publisher {}

struct AnyPublisher: Publisher {}
```

Publisher

- AnyPublisher
- Value Type

```
public protocol Publisher {}

struct AnyPublisher: Publisher {}
```

Publisher

- AnyPublisher
- Value Type
- Output (Data Type)
- Failure (Error Type)

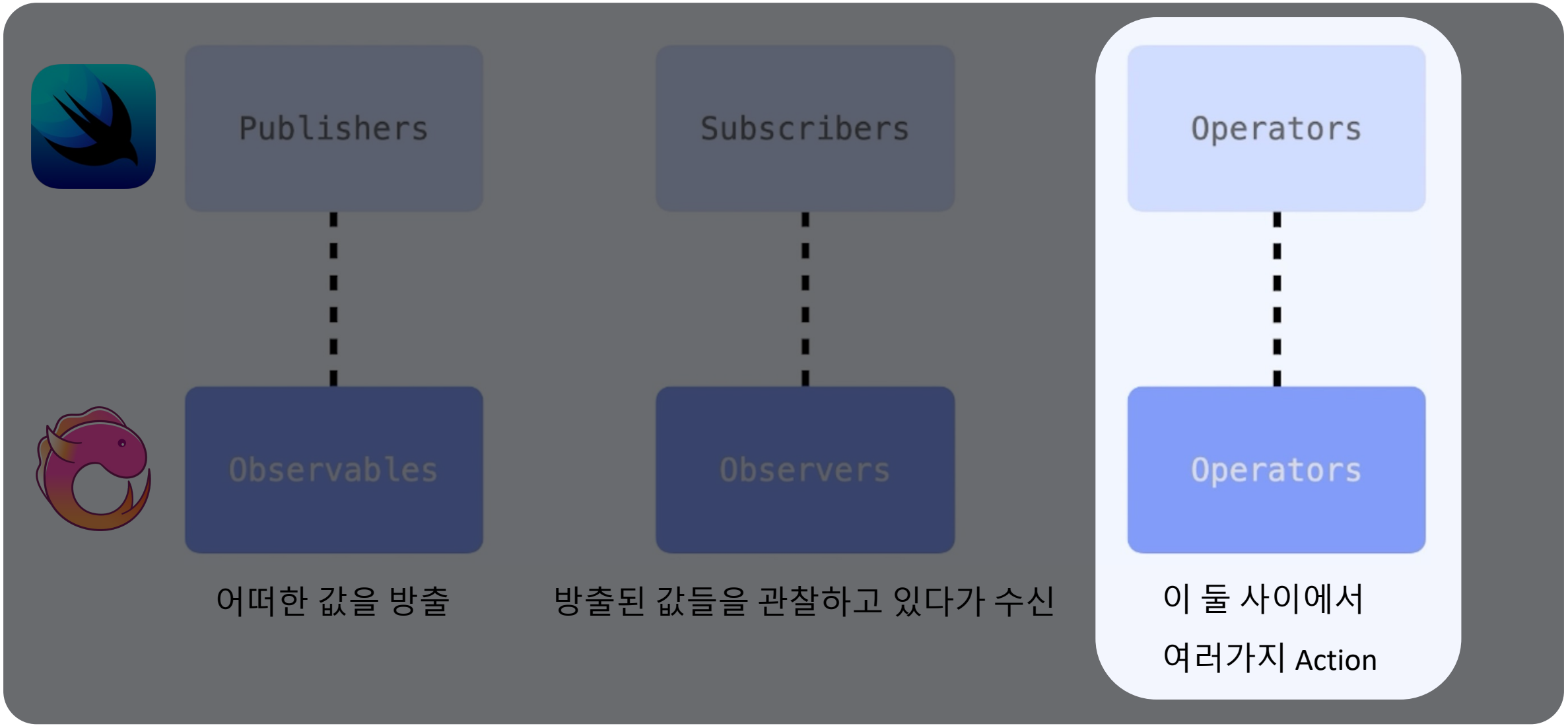
```
associatedtype Output

associatedtype Failure : Error
```

Publisher

- AnyPublisher
- Value Type
- Output (Data Type)
- Failure (Error Type)

```
AnyPublisher<String, Error>

AnyPublisher<String, Never>
```

Operators


Publishers
Subscribers
Operators
Observables
Observers
Operators
어떠한 값을 방출
방출된 값들을 관찰하고 있다가 수신
이 둘 사이에서
여러가지 Action

Operators

- `tryMap`
- `tryScan`
- `tryFilter`
- `tryCompactMap`
- `tryRemoveDuplicates(by:)`
- `tryReduce`
- `tryMax(by:)`
- `tryMin(by:)`
- `tryContains(where:)`
- `tryAllSatisfy`
- `tryDrop(while:)`
- `tryPrefix(while:)`
- `tryFirst(where:)`
- `tryLast(where:)`
- `tryCatch`

Map vs TryMap

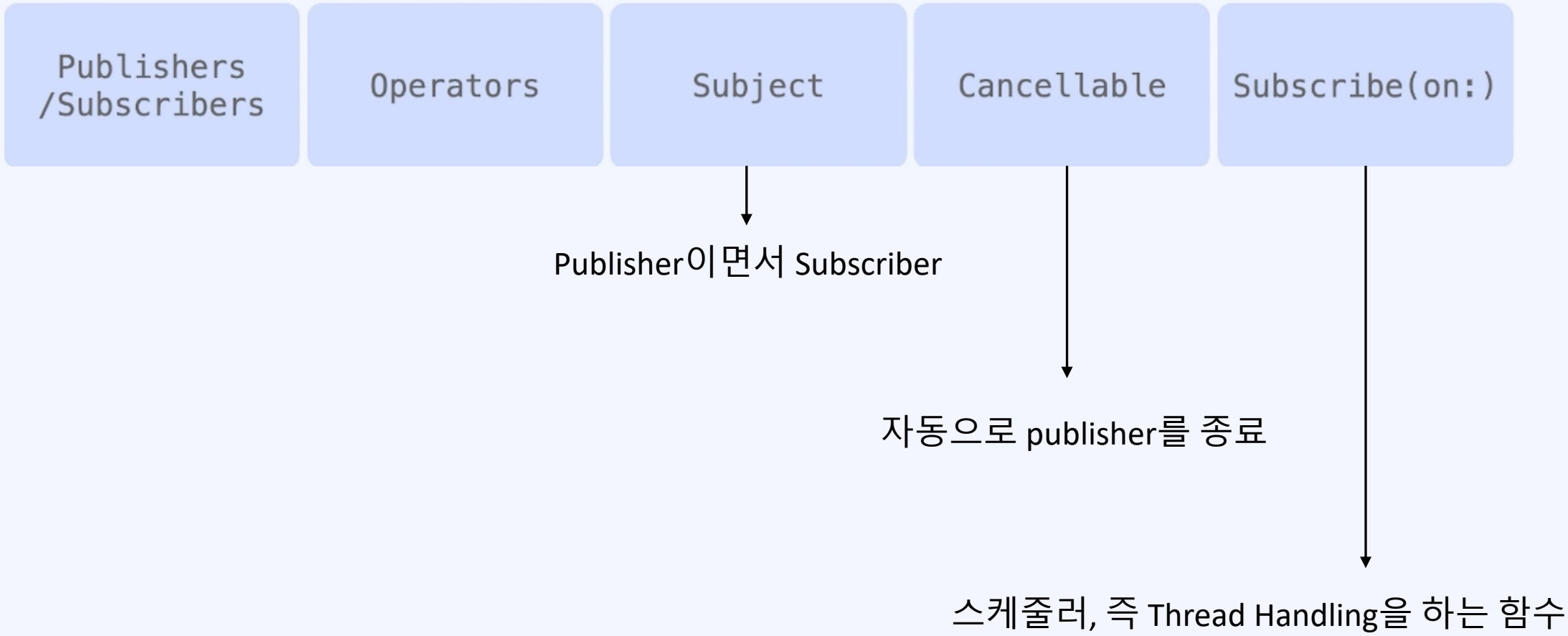
```
func map<T>(_ transform: (Output) -> T) -> Just<T>
```

```
func tryMap<T>(_ transform: (Output) throws -> T) -> Result<T, Error>.Publisher
```

Combining Operators

• Merge, Merge3, Merge4, Merge5, Merge6, Merge7, Merge8, MergeMany	• merge
• CombineLatest, CombineLatest3, CombineLatest4	• combineLatest
• Zip, Zip3, Zip4	• zip

Combine 핵심 요소



Subject

- `PassthroughSubject`
- `CurrentValueSubject`

Subject

- PassthroughSubject
- CurrentValueSubject

```
class PassthroughSubject<Output, Failure> {

    public init()

}
```

Subject

- PassthroughSubject
- CurrentValueSubject

```
class CurrentValueSubject<Output, Failure> {

    public init(_ value: Output)

}
```

Cancellable

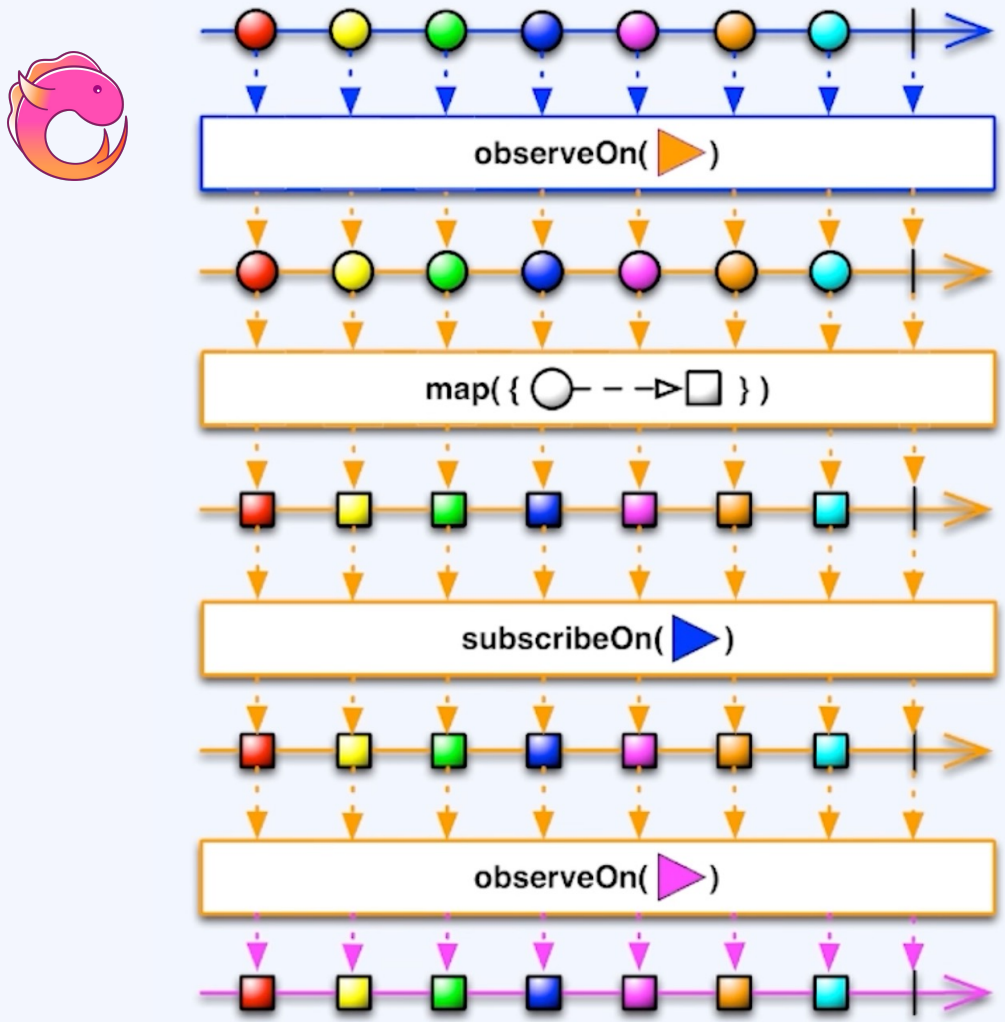
`Cancellable`

`AnyCancellable`

Cancellable

```
let cancellables = Set<Cancellable>()

Just(1)
    .sink {
        print($0)
    }
    .store(in: &cancellables)
```

Tread Handling



Tread Handling

```
Just(1)
  .subscribe(on: DispatchQueue.main)
  .map { _ in
     implements()
  }
  .sink { … }
```

```swift
//MARK: - Read
// Note Data Fetch
///비동기 통신 2번을 통해 [Note]를 가져옵니다.
func fetchNote() async throws -> [Note] {
    guard let userId = UserInfo.token else { return [] }

    var data = [Note]() // 비동기 통신으로 받아올 Property

    let userSnapshot = try await database.collection("User").document(userId).getDocument() // 첫번째 비동기 통신
    let docData = userSnapshot.data()

    let noteIdList: [String] = docData?["noteId"] as? [String] ?? []

    for noteId in noteIdList {
        // 두번째 비동기 통신
        let noteSnapshot = try await database.collection("Note").document(noteId).getDocument()

        let docData = noteSnapshot.data()!

        let id: String = docData["id"] as? String ?? ""
        let userId: String = docData["userId"] as? String ?? ""
        let title: String = docData["title"] as? String ?? ""
        let contents: String = docData["contents"] as? String ?? ""
        let image: [String] = docData["image"] as? [String] ?? []
        let isFixed: Bool = docData["isFixed"] as? Bool ?? false
        let geoPoint: GeoPoint = docData["geoPoint"] as! GeoPoint
        let updatedAt: Timestamp = docData["updatedAt"] as! Timestamp
        let createdAt: Timestamp = docData["createdAt"] as! Timestamp

        data.append(Note(id: id, userId: userId, title: title, contents: contents, image: image, isFixed: isFixed, updatedAt: upda
            createdAt.dateValue(), geoPoint: Note.Coordinates(latitude: geoPoint.latitude, longitude: geoPoint.longitude)))
    }

    return data.sorted {$0.updatedAt > $1.updatedAt} // 내림차순 정렬
}
```

```swift
struct FirebaseManager {
    //MARK: Property
    static let database = Firestore.firestore()

    static func fetchNote(userId: String) -> AnyPublisher<[Note], Error> {
        Future<[Note], Error> { promise in
            self.database.collection("User").document(userId).getDocument { snapshot, error in // 첫번째 비동기 통신
                if let error = error {
                    promise(.failure(error))
                    return
                }

                guard let snapshot = snapshot else {
                    promise(.failure(NetworkResult<Int>.pathErr))
                    return
                }

                var data = [Note]()

                let docData = snapshot.data()
                let noteIdList: [String] = docData?["noteId"] as? [String] ?? []

                let group = DispatchGroup()

                for noteId in noteIdList {
                    // 두번째 비동기 통신
                    group.enter()
                    self.database.collection("Note").document(noteId).getDocument { snapshot2, error2 in
                        if let error2 = error2 {
                            promise(.failure(error2))
                            return
                        }

                        guard let snapshot2 = snapshot2 else {
                            promise(.failure(NetworkResult<Int>.pathErr))
                            print("error")
                            return
                        }

                        let docData = snapshot2.data()!

                        let id: String = docData["id"] as? String ?? ""
                        let userId: String = docData["userId"] as? String ?? ""
                        let title: String = docData["title"] as? String ?? ""
                        let contents: String = docData["contents"] as? String ?? ""
```

```swift
        let group = DispatchGroup()

        for noteId in noteIdList {
            // 두번째 비동기 통신
            group.enter()
            self.database.collection("Note").document(noteId).getDocument { snapshot2, error2 in
                if let error2 = error2 {
                    promise(.failure(error2))
                    return
                }

                guard let snapshot2 = snapshot2 else {
                    promise(.failure(NetworkResult<Int>.pathErr))
                    print("error")
                    return
                }

                let docData = snapshot2.data()!

                let id: String = docData["id"] as? String ?? ""
                let userId: String = docData["userId"] as? String ?? ""
                let title: String = docData["title"] as? String ?? ""
                let contents: String = docData["contents"] as? String ?? ""
                let image: [String] = docData["image"] as? [String] ?? []
                let isFixed: Bool = docData["isFixed"] as? Bool ?? false
                let geoPoint: GeoPoint = docData["geoPoint"] as! GeoPoint
                let updatedAt: Timestamp = docData["updatedAt"] as! Timestamp
                let createdAt: Timestamp = docData["createdAt"] as! Timestamp

                data.append(Note(id: id, userId: userId, title: title, contents: contents, image: image, isFixed: isFixed, updatedAt: updatedAt.dateValue(),
                    createdAt: createdAt.dateValue(), geoPoint: Note.Coordinates(latitude: geoPoint.latitude, longitude: geoPoint.longitude)))
                group.leave()
            }
        }
        // 모든 task들이 완료되었을 경우 .notify 속에 있는 코드가 실행
        group.notify(queue: .main) {
            data = data.sorted {$0.updatedAt > $1.updatedAt} // 내림차순 정렬
            promise(.success(data))
        }
    }
}
.eraseToAnyPublisher()
```

```swift
//MARK: - 여기부터 설명
private var cancellables = Set<AnyCancellable>()    // cancellables
@Published var networkError: NetworkResult<Int>
@Published var showErrorAlertMessage = "오류"

func fetchNoteByCombine() {
    guard let userId = UserInfo.token else { return }

    FirebaseManager.fetchNote(userId: userId)
        .receive(on: DispatchQueue.main)                // Main Thread
        .sink { [self] completion in
            switch completion {
            case .finished:
                return
            case .failure(let error):                   // 에러 발생
                NSLog(error.localizedDescription)
                self.networkError = .pathErr
                self.showErrorAlertMessage = self.networkError.errorDescription!    // 에러 출력
            }
        } receiveValue: { [weak self] (data) in
            self?.noteList = data                       // 성공적으로 통신했을때
        }
        .store(in: &cancellables)                       // cancellables : 자동으로 publisher(AnyPublisher)를 종료
}
```

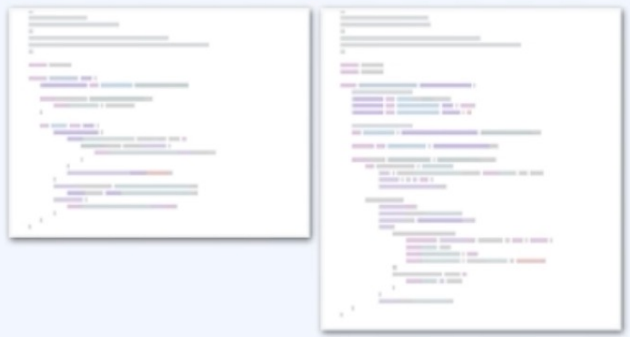
```swift
struct HomeView: View {
    var body: some View {
                    } // Button

                    Button {
                        showMedia = true
                    } label: {
                        Label("미디어", systemImage: "photo.on.rectangle")
                    } // Button
                } header: {
                    Text("자료")
                } // Section
            } // List
            .refreshable {
                do {
                    noteVM.noteList = try await noteVM.fetchNote()
                } catch let(error) {
                    print(error)
                }
            }
        } // VStack
        .onAppear {
            Task {
                // 로그인 작업전 임시
                UserDefaults.standard.set("erXb0BUUhmRmHT7PngBb", forKey: "userIdToken")
//              noteVM.noteList = try await noteVM.fetchNote()
                noteVM.fetchNoteByCombine()
//              noteVM.noteIdList = try await noteVM.fetchNoteIdInUser()
            }
        }
        .toolbar {
            Button {
                showSetting = true
            } label: {
                Image(systemName: "ellipsis.circle")
            } // Button
        }
        .navigationTitle("Notes")
        .navigationDestination(isPresented: $showSetting) {
            Settings()  // 설정
        }
        .navigationDestination(isPresented: $showNote) {
```

```swift
enum NetworkResult<T>: Error, LocalizedError {
    case success(T) // 서버 통신 성공했을 때
    case fail(T) // 서버 통신 성공은 성공했으나, 로직상 fail일때
    case requestErr(T) // 요청 에러 발생했을 때
    case unAuthorized // 인증이 되지 않을때
    case pathErr // 경로 에러 발생했을 때
    case serverErr // 서버의 내부적 에러가 발생했을 때
    case networkFail // 네트워크 연결 실패했을 때

    // error message
    var errorDescription: String? {
        switch self {
        case .success:
            return NSLocalizedString("서버 통신 성공했을 때", comment: "")
        case .fail:
            return NSLocalizedString("서버 통신 성공은 성공했으나, 로직상 fail일때", comment: "")
        case .requestErr:
            return NSLocalizedString("요청 에러 발생했을 때", comment: "")
        case .unAuthorized:
            return NSLocalizedString("인증이 되지 않을때", comment: "")
        case .pathErr:
            return NSLocalizedString("경로 에러 발생했을 때", comment: "")
        case .serverErr:
            return NSLocalizedString("서버의 내부적 에러가 발생했을 때", comment: "")
        case .networkFail:
            return NSLocalizedString("네트워크 연결 실패했을 때", comment: "")
        }
    }
}
```

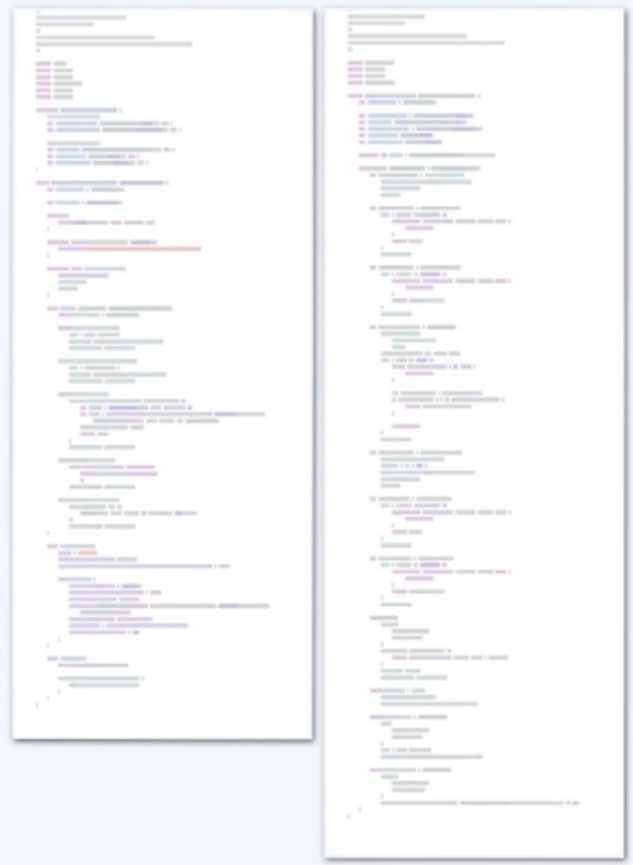
```swift
    //MARK: - 여기부터 설명
    private var cancellables = Set<AnyCancellable>()    // disposeBag
    @Published var networkError: NetworkResult<Int> = .success(200)
    @Published var showErrorAlertMessage = "오류"

    func fetchNoteByCombine() {
        guard let userId = UserInfo.token else { return }

        FirebaseManager.fetchNote(userId: userId)
            .receive(on: DispatchQueue.main)
            .sink { completion in
//                switch error.responseCode {
//                case 400: // 요청 에러 발생했을 때
//                  break
//                case 500: // 서버의 내부적 에러가 발생했을 때
//                  break
//                default:
//                  break
//                }
            } receiveValue: { [weak self] (data) in
                self?.noteList = data
            }
            .store(in: &cancellables)
    }
}
```

정리

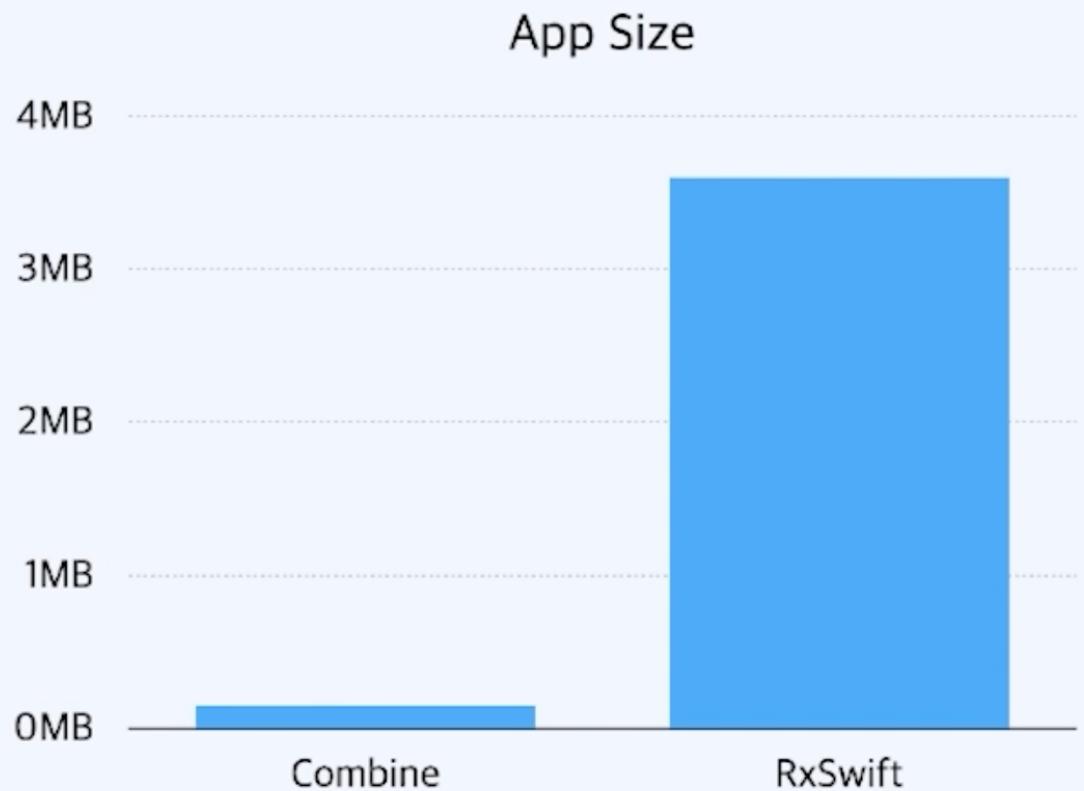
Combine/SwiftUI



RxSwift/RxCocoa



정리



마무리

The memory models of RxSwift and Combine are very different.

Combine is really made for performance.