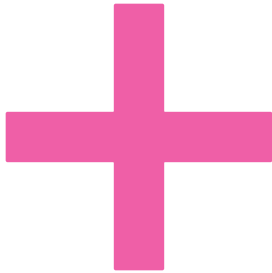


# Subject

05

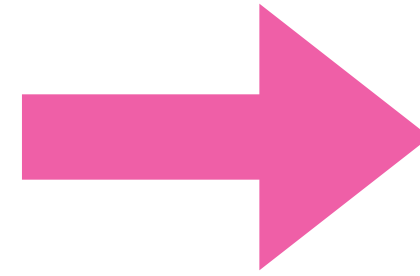
# Subject?

**Observable**  **Observer**

Observer인 동시에 Observable

# Subject?

**Cold**

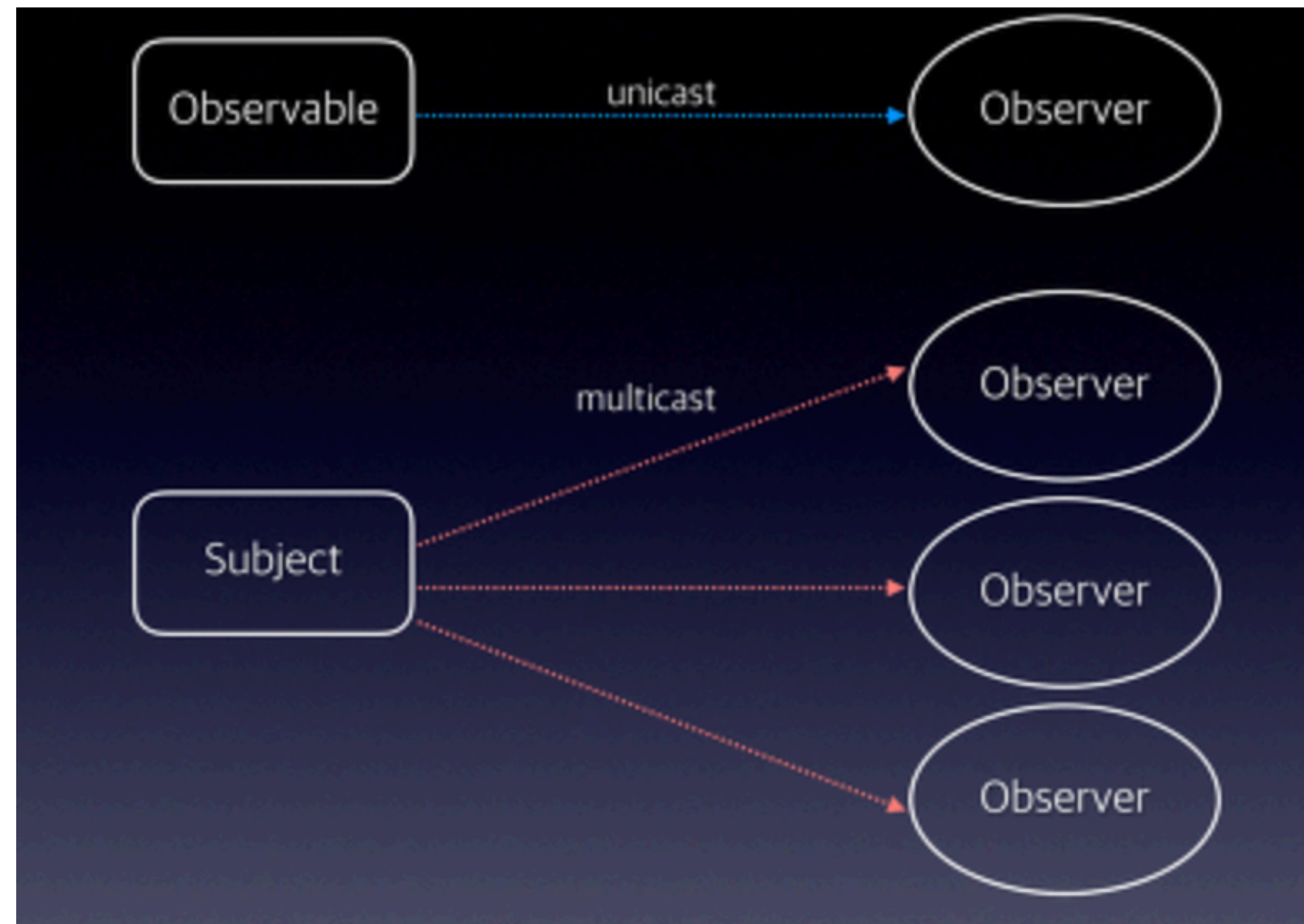


**Hot**

?

Observable이랑  
다른게 뭐야?

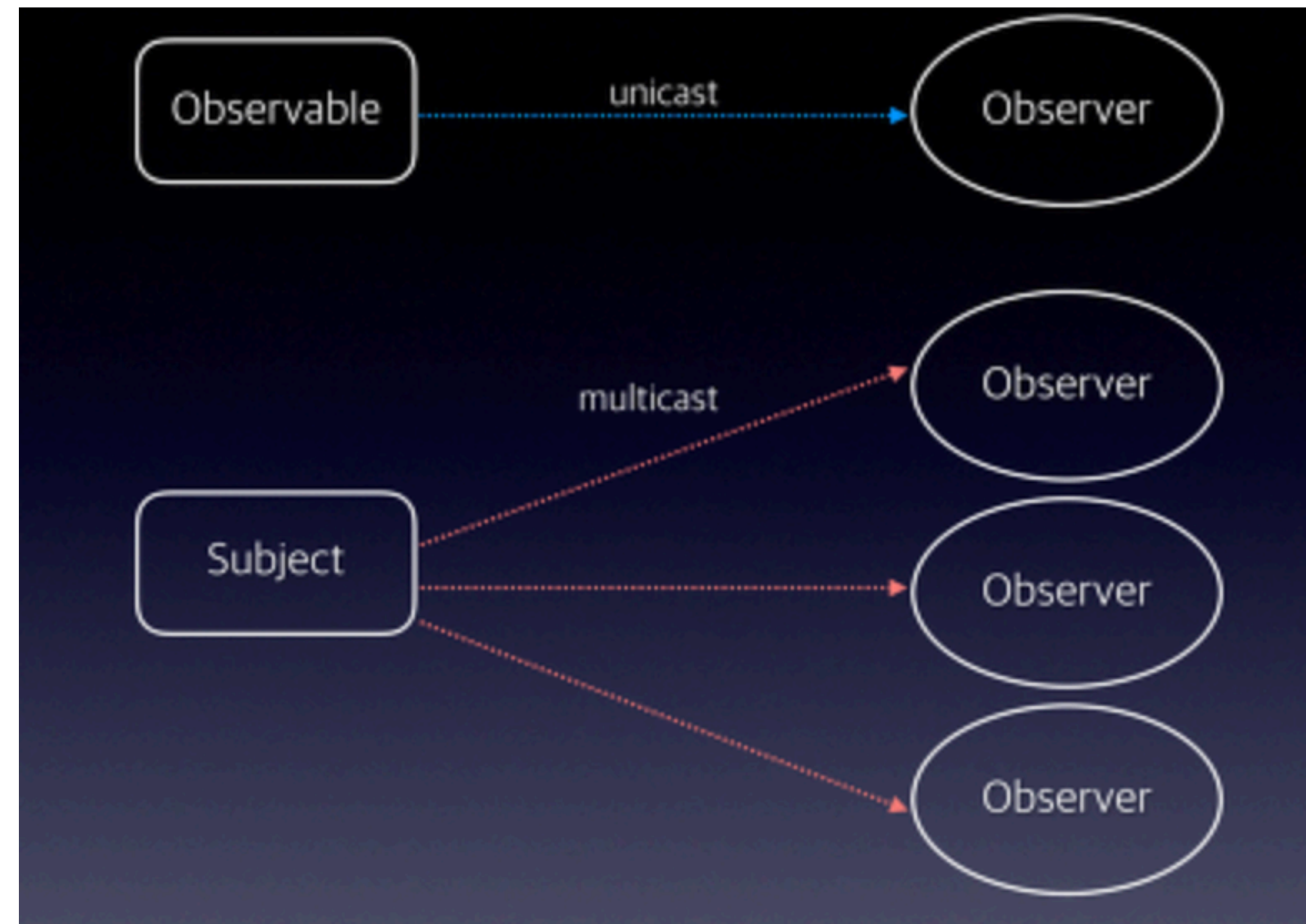
# Subject vs Observable



multicast방식이기 때문에 여러 개의  
Observer에게 이벤트를 발행

반대로 unicast 방식이므로 하나의  
Observer에게만 이벤트를 발행

# Subject vs Observable



Observer들의 정보를 저장하고 발행된 이벤트를 모든 Observer에게 제공

Observer가 구독하게 되면 그때마다 새로 create하여 발행하는 과정

# Subject vs Observable

**State O**

**Data**를 메모리에 저장

**State X**

단지 함수

# Subject vs Observable

자주 데이터를 저장하고 수정할 때

여러 개의 옵저버가 data를 관찰해야할 때

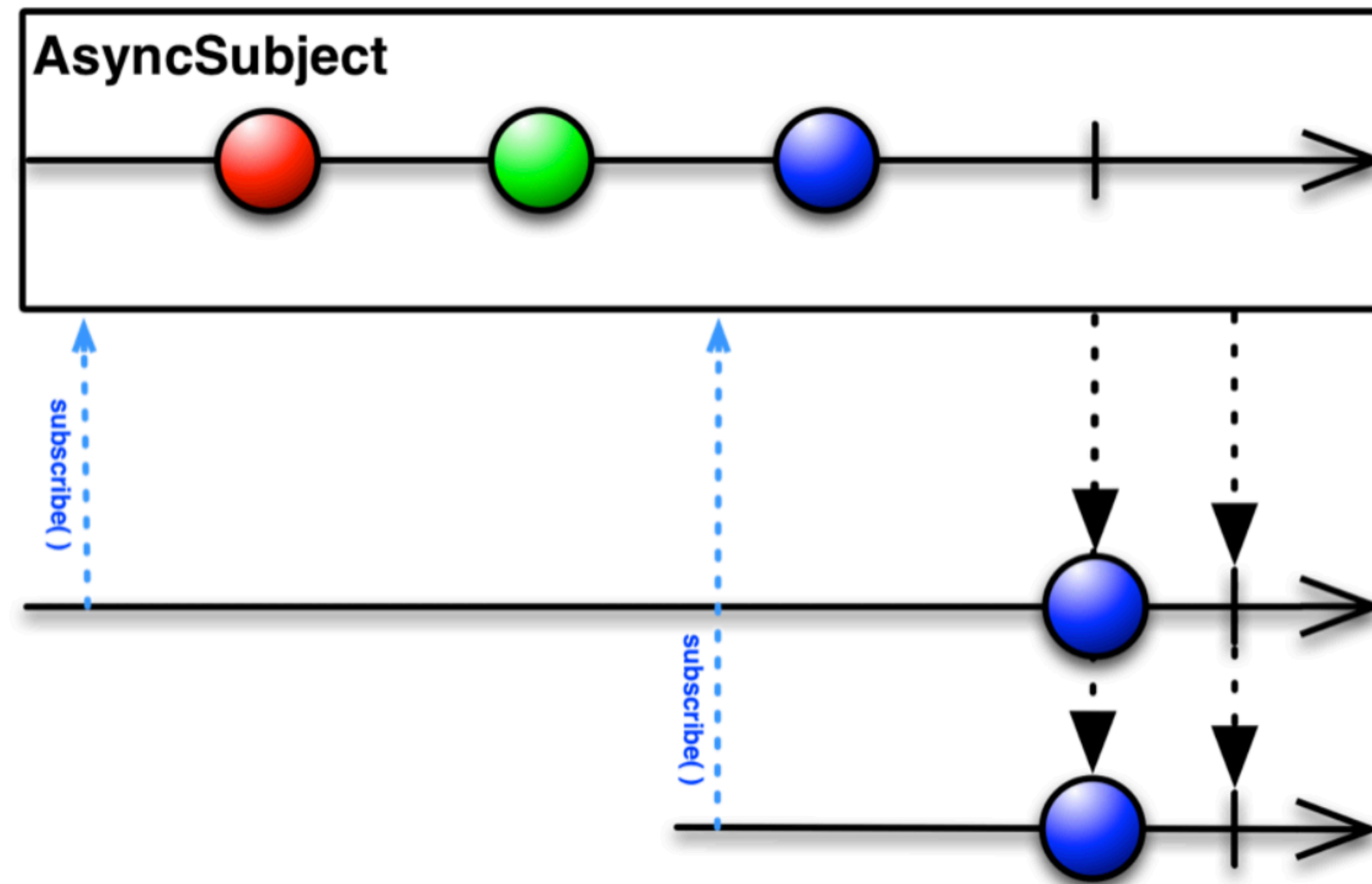
Observer와 Observable 사이의 프록시 역할

하나의 Observer에 대한  
간단한 Observable이 필요할 때



# **Varieties of Subject**

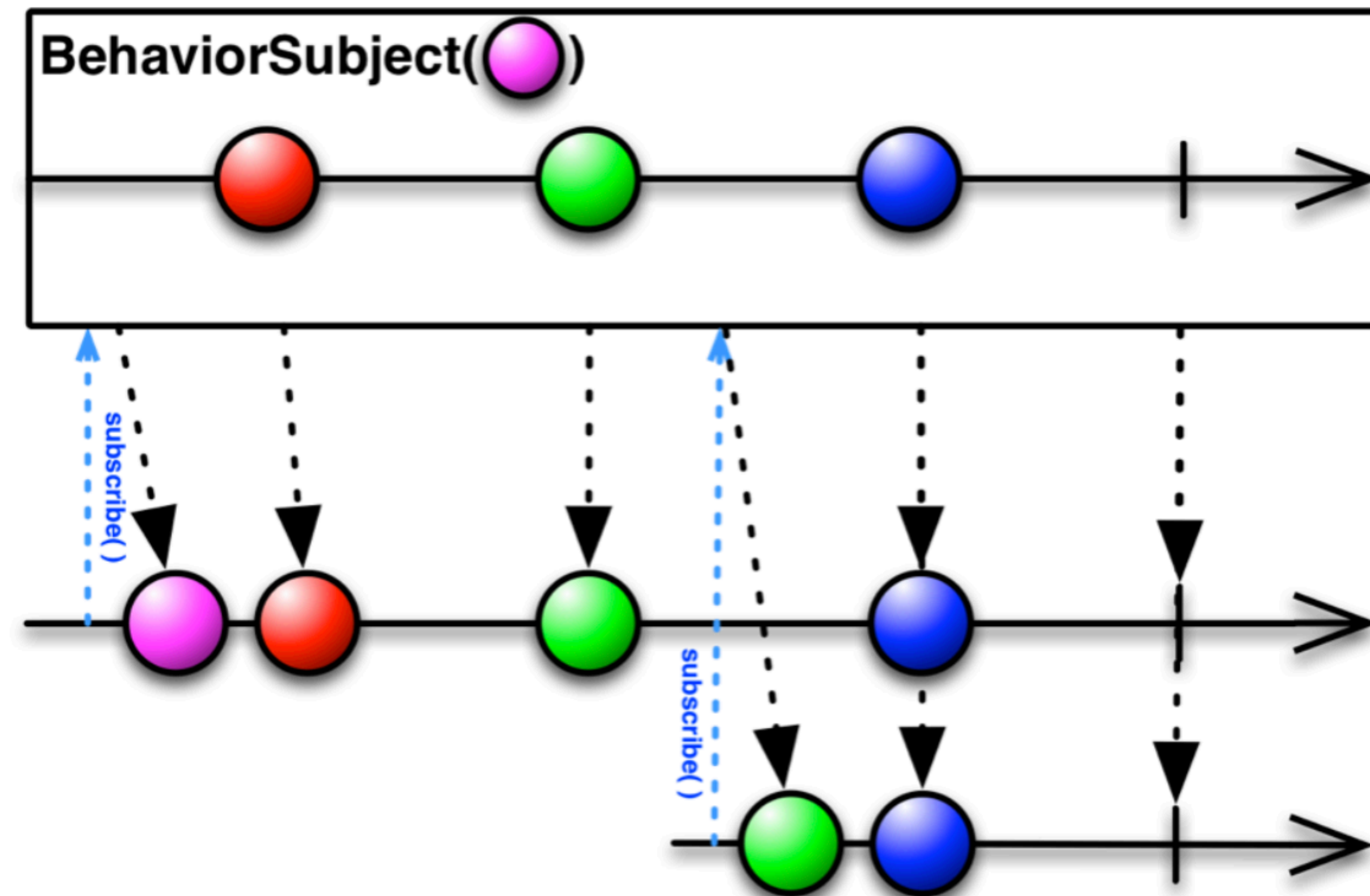
# Varieties of Subject



**Observable로부터 배출된 마지막 값을 배출하고  
소스 Observable의 동작이 완료된 후에야 동작**

! 만약 해당 Observable이 오류로 인해 종료될 경우엔 아무 항목도 배출하지 않고 발생한 오류를 그대로 전달합니다.

# Varieties of Subject



구독하기 시작하면 초기값을 가진 형태의 Subject  
가장 최근에 발행한 항목의 발행을 시작함

! 만약 해당 Observable이 오류로 인해 종료될 경우엔 아무 항목도 배출하지 않고 발생한 오류를 그대로 전달합니다.

```
let disposeBag = DisposeBag()
let subject = BahaviorSubject<String>(value: "init")
subject.onNext("Last Issue")

subject.subscribe { event in
    print(event)
}

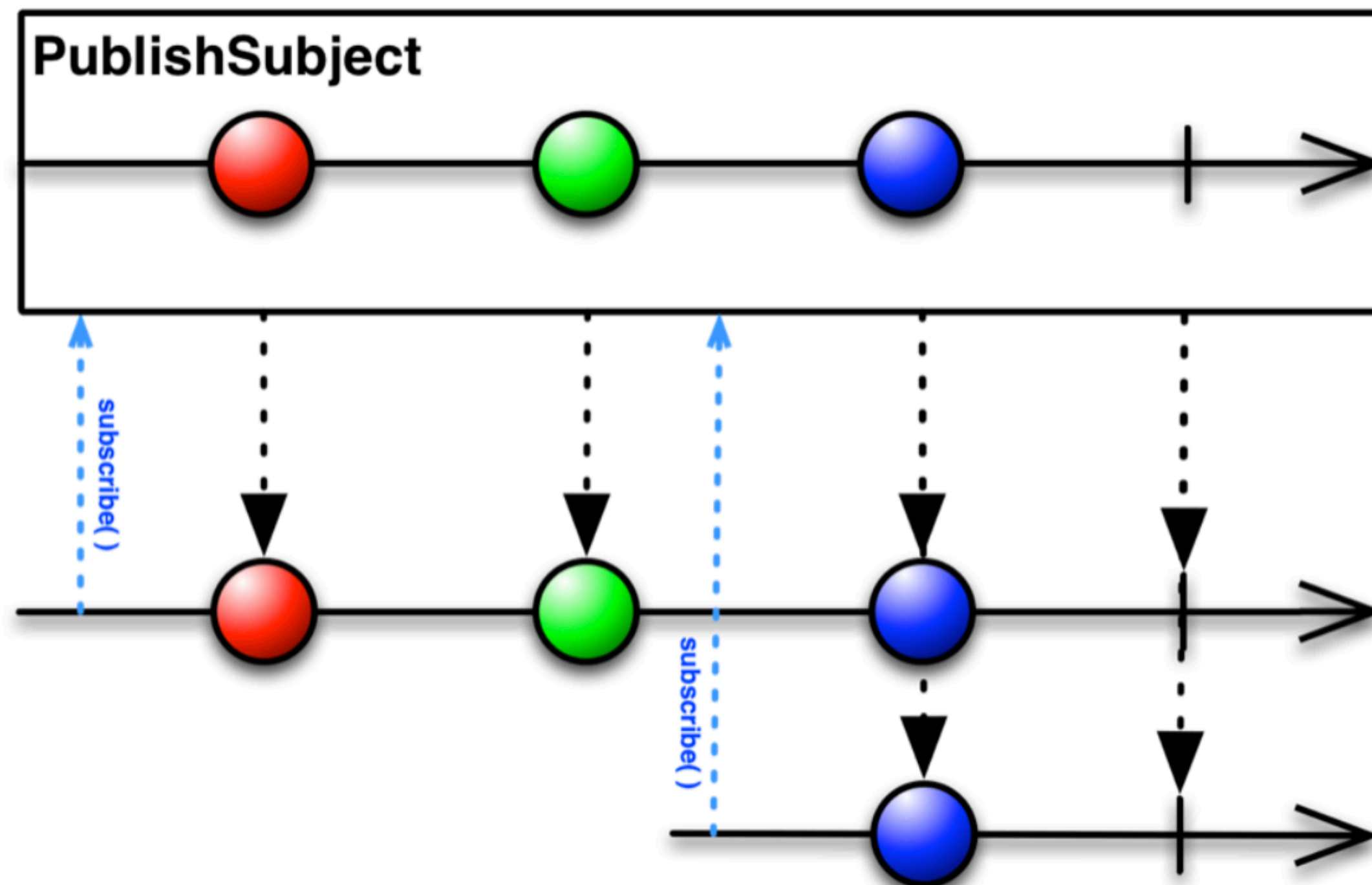
subject.onNext("Issue 1")
//Last Issue
//Issue 1
```

```
BehaviorSubject<Object> subject = BehaviorSubject.create("default");

subject.subscribe(observer);

subject.onNext("one");
subject.onNext("two");
subject.onNext("three");
//default one two three
```

# Varieties of Subject



## 구독 이후에 Observable이 배출한 이벤트들만 Observer에게 배출하는 형태

**Subject 생성 시점과 Observer가 구독하기 시작하는 시점  
사이의 항목은 잃어버릴 수 있음,,**

**!** 만약 해당 Observable이 오류로 인해 종료될 경우엔 아무 항목도 배출하지 않고 발생한 오류를 그대로 전달합니다.

```
let disposeBag = DisposeBag()
let subject = PublishSubject<String>()
subject.onNext("Issue 1")

subject.subscribe { event in
    print(event)
}

subject.onNext("Issue 2")
subject.onNext("Issue 3")
//Issue 2
//Issue 3
```

```
PublishSubject<Integer> source = PublishSubject.create();

source.subscribe(getFirstObserver());

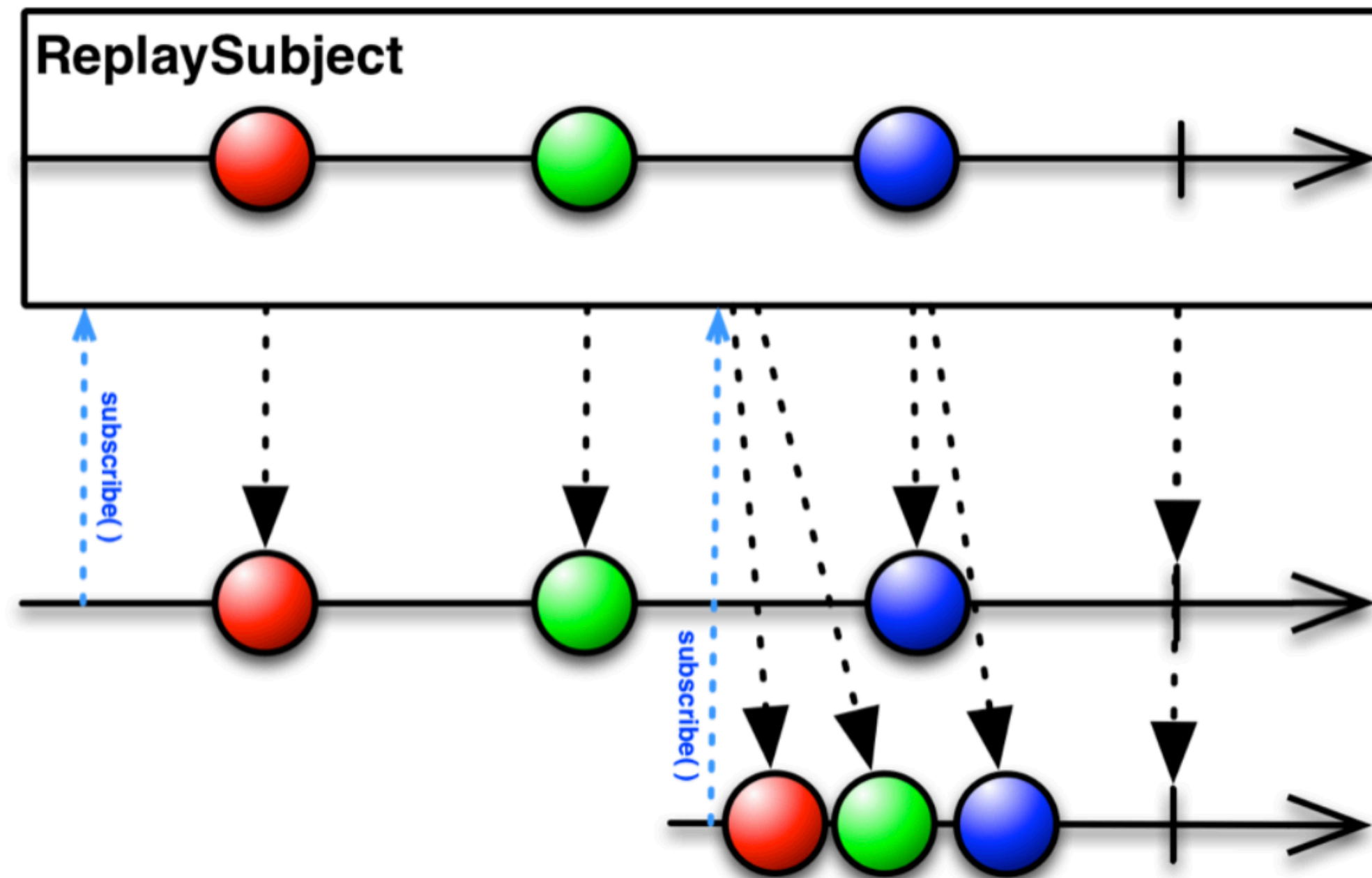
source.onNext(1);
source.onNext(2);
source.onNext(3);
//1 2 3

source.subscribe(getSecondObserver());

//4
source.onNext(4);
source.onComplete();
```



# Varieties of Subject



**Observer가 구독을 시작한 시점과 관계없이  
Observable들이 배출한 모든 항목들을 배출**

! 멀티스레드 환경에서는 Observable 계약 위반과 Subject에서 어느 항목 또는 알림을 먼저 재생해야하는지 모호함이 발생하기 때문에 호출을 유발시키는 onNext 메소드는 사용하지 않도록 주의해야합니다.

```
let disposeBag = DisposeBag()
let subject = ReplaySubject<String>.create(bufferSize: 2)

subject.onNext("Issue 1")
subject.onNext("Issue 2")
subject.onNext("Issue 3")

subject.subscribe { event in
    print(event)
}

subject.onNext("Issue 4")
//Issue 2
//Issue 3
//Issue 4
```

```
ReplaySubject<Integer> source = ReplaySubject.create();

source.subscribe(getFirstObserver());

source.onNext(1);
source.onNext(2);
source.onNext(3);
source.onNext(4);
source.onComplete();

source.subscribe(getSecondObserver());
```