초격차 패키지 Online.

TS를 활용한 함수형 프로그래밍 온보딩

PART1 | 함수와 맥락

함수와 부수효과, 함수와 타입, Array 의 map

PART2 | 함수형 프로그래밍과 에러처리

Option 으로 null 처리, 여러 모양의 함수와 map, Try로 Error 처리

PART3 | 함수형 비동기 반응형 프로그래밍

Promise와 async await, Observable 과 Reactive, 더 깊은 함수형의 세계로

Observable: 반응형으로 가는 길

1. 돌아보기

Continuation-Passing Style

결괏값을 인자로 전달받은
callback 함수를 통해 전달

콜백 지옥?

```
f("test", (a)=>{
    g(a, (b) => {
        h(b, (c) => {
            program(c);
        });
    });
});
```

Async


```ts
type Async<A> = (ret: (x: A) => void) => void;

const flatMap = <A, B>(a: Async<A>, f: (a: A) => Async<B>): Async<B> => {
  return (ret) => {
    a((a_) => {
      const b = f(a_);
      b((b_) => ret(b_));
    });
  };
}

const map = <A, B>(a: Async<A>, f: (a: A) => B): Async<B> => {
  return flatMap(a, (a_) => resolve(f(a_)));
}
```

Async vs Promise

Async map, flatMap

```
const a = f("test");
const b = flatMap(a, (a_) => g(a_));
const c = flatMap(b, (b_) => h(b_));
const result = map(c, (c_) => program(c_));
run(result);
run(result);
```

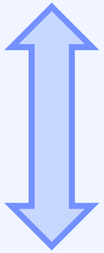
Promise then

```
export const main = async () => {
  const a = f("abc");
  const b = a.then(g);
  const c = b.then(h);
  const result = c.then(program);
  result.catch(handleError);

  f("abc").then(g).then(h).then(program).catch(handleError);
```

async await 구문

```
f("abc").then(g).then(h).then(program).catch(handleError);
```



```
xport async function main() {
 const a = await f("test");
 const b = await g(a);
 const c = await h(b);
 program(c);
```

핵심 개념 2가지

함수를 합성해서 복잡한 프로그램을 쉽게 만들기

부수효과를 공통적인 방법으로 추상화

함수형은 어디까지?

Array : 값이 몇 개인지 모른다

Option : 값이 없을 수 있다
Try : 예외가 발생할 수 있다

Promise: 값의 전달을 미룰 수 있다
Observable : ???

명령형 구문

↕

값으로 타입 정의하기

map과 flatMap으로 추상화

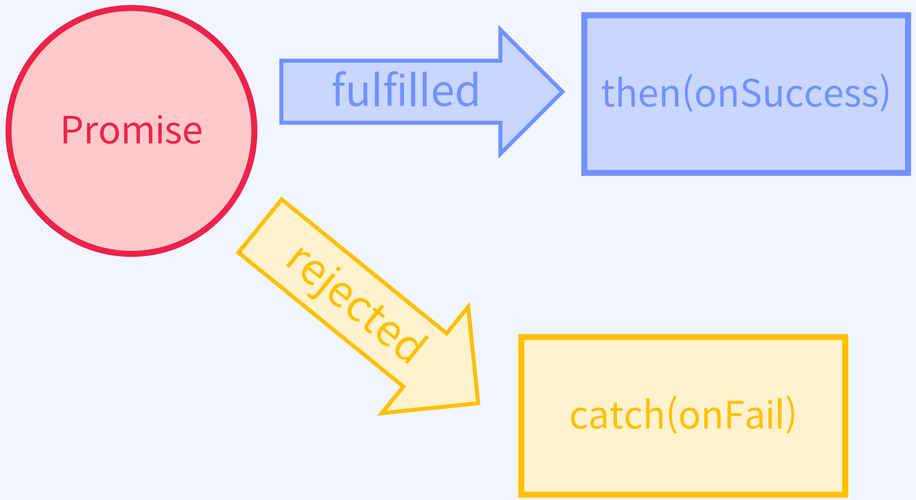
함수를 합성해서 리팩토링

Observable: 반응형으로 가는 길

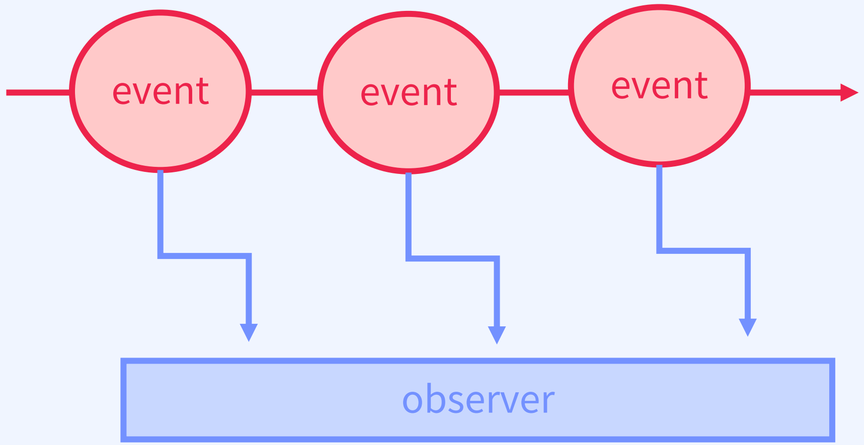
2. 미리보기

Promise vs Observable

Promise
값이 한 번만 전달된다



Observable
값이 여러 번 전달된다



Observable 타입

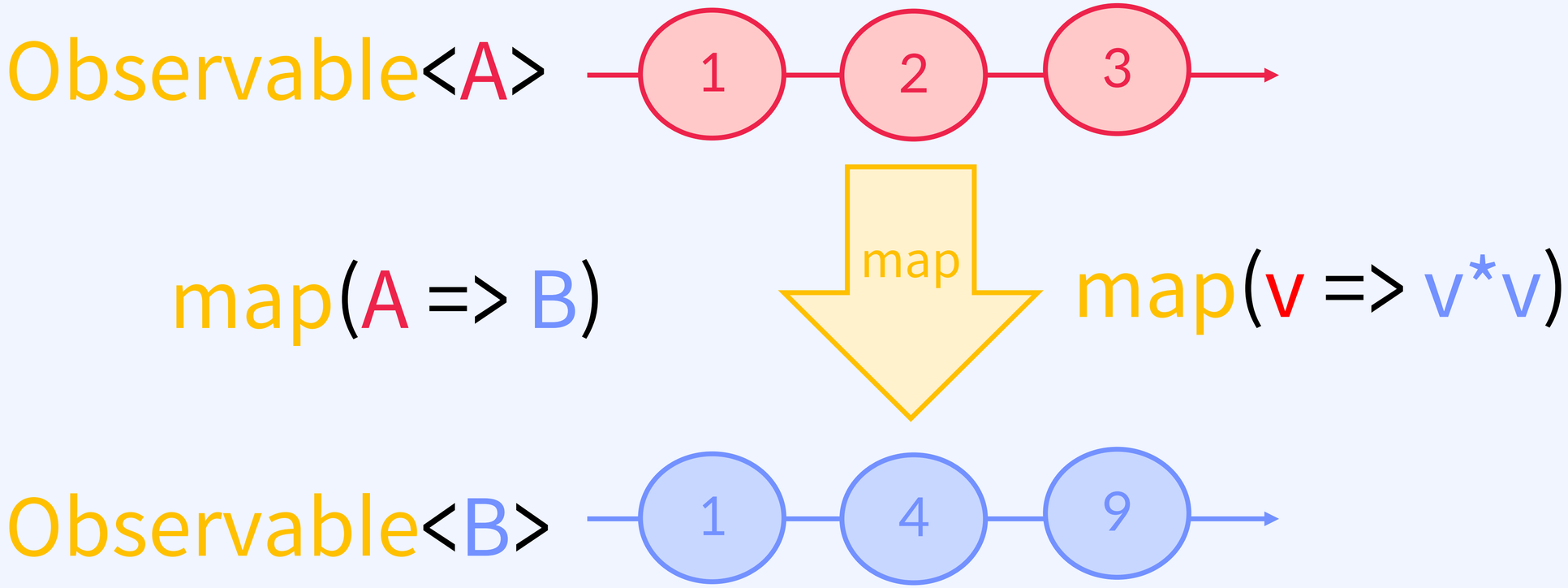
type Observer<T> = (v: T)=> void

비동기로 발생하는 값을 구독하기 위해 외부에서 전달하는 함수

type Observable<T> => (o: Observer<T>) => void

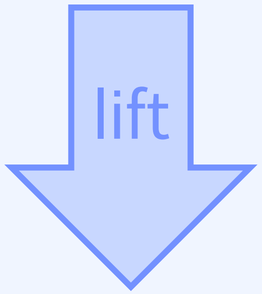
값이 생성되는 비동기 작업에 Observer를 전달하고
비동기 작업을 실행하는 함수

Observable의 map



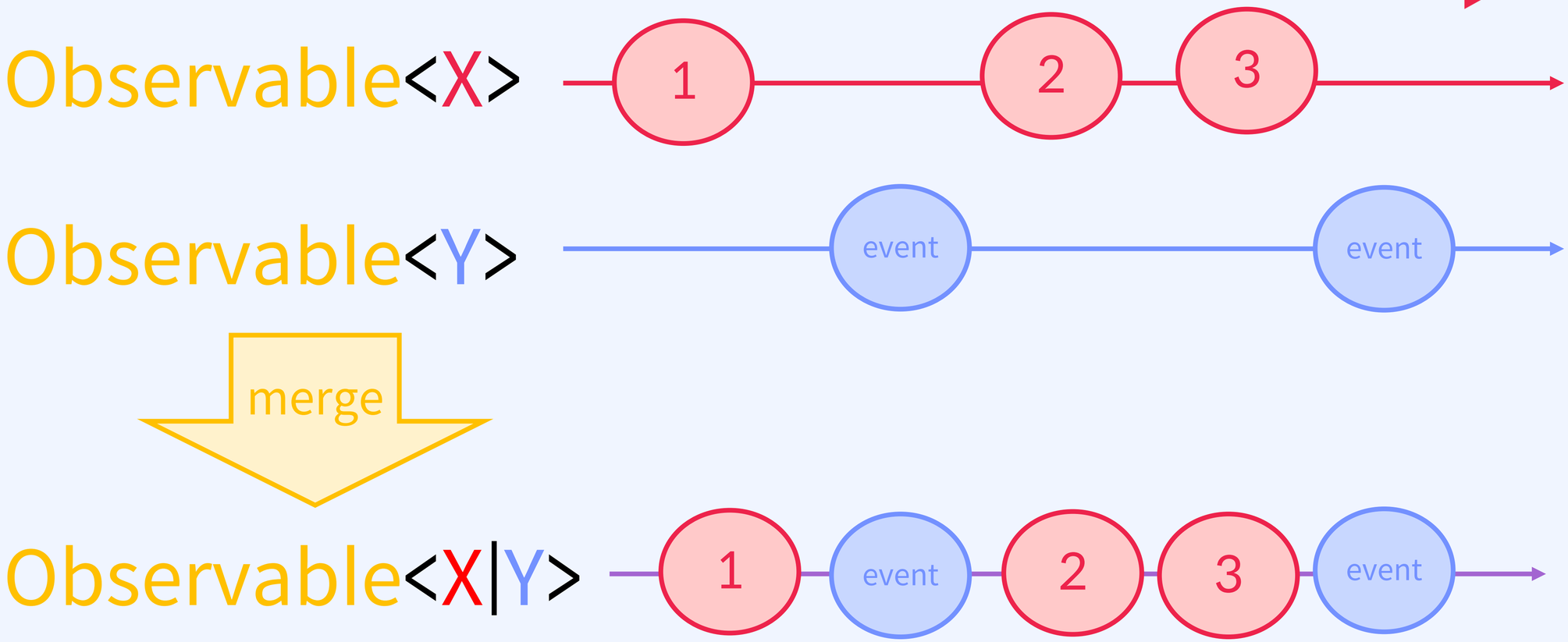
lift

Observer<B> => Observer<A>



Observable<A> => Observable<B>

merge 하나로 합치기



Observable: 반응형으로 가는 길

3. 여러 번 발생하는 비동기

Observable: 반응형으로 가는 길

4. Observable의 타입

Observable: 반응형으로 가는 길

5. Observer와 Observable

Observable: 반응형으로 가는 길

6. lift: Observer를 Observable로 만들기

Observable: 반응형으로 가는 길

7. Observable의 map

Observable: 반응형으로 가는 길

8. merge : 여러 개의 Observable을 하나로