Rx Seminar

© 2020 du.nguyenhuy@vti.com.vn aka fshdn19

1. About Rx

1.1 Rx là gì?

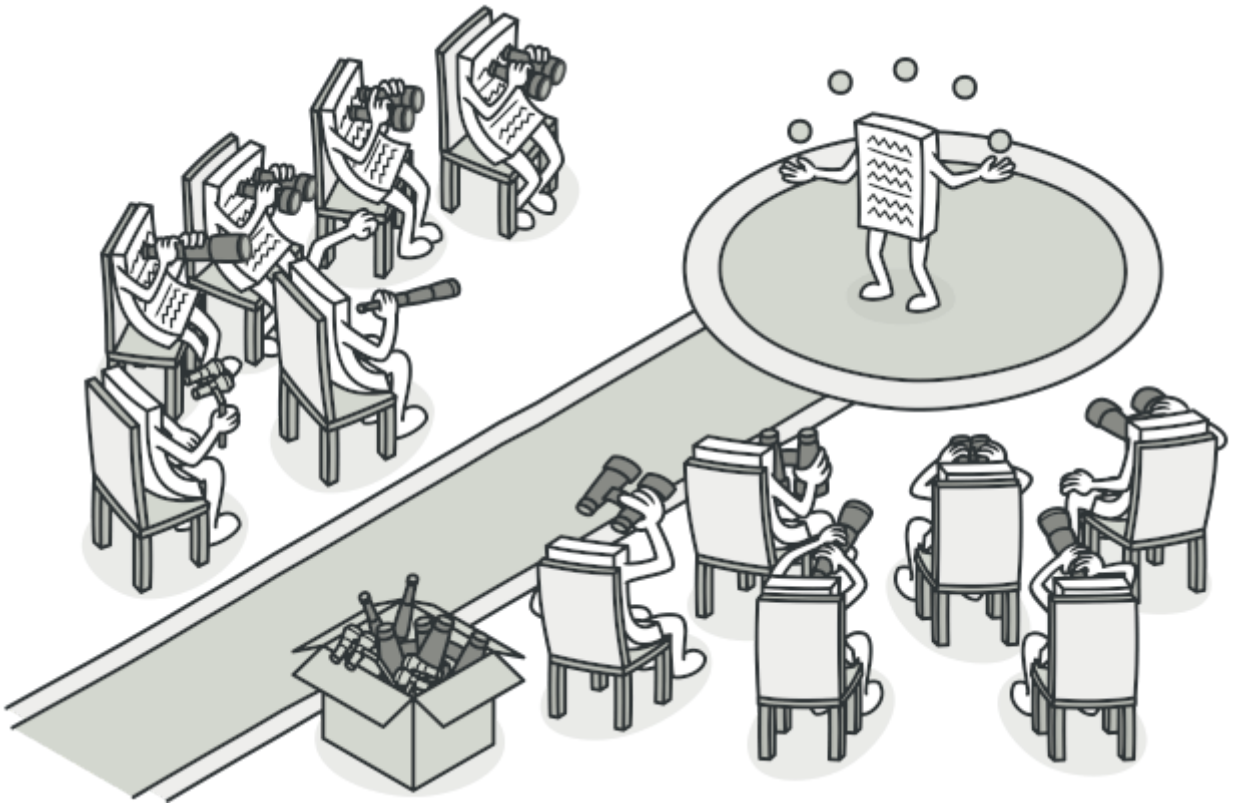
RxDrugs, React, ReactNative and ReactiveX

1.2 Observer pattern

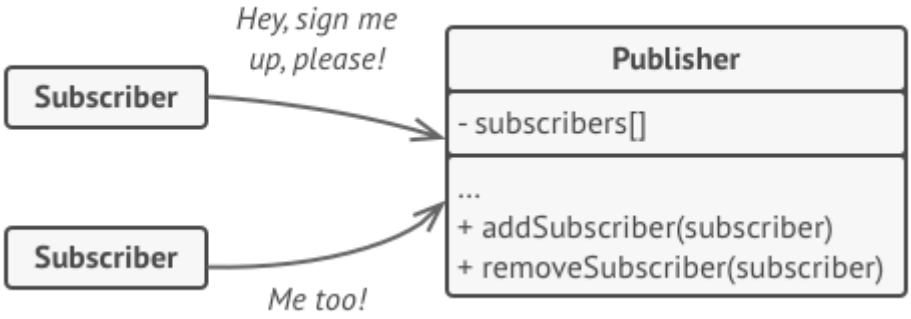
a. Vấn đề


CHỈ CẦN CÓ THAY ĐỔI DỮ
LIỆU TRONG DATABASE, GIAO
DIỆN SẼ LẬP TỨC ĐƯỢC CẬP NHẬT
HOW?
5S QUERY 1 LẦN
imgflip.com

b. Giải pháp



- Tạo ra cơ chế để theo dõi cho nhiều đối tượng khác nhau về một (vài) sự kiện nào đó xảy đến với một chủ thể mà chúng đang để ý tới
- Minh họa thực tế giống như cơ chế theo dõi cá nhân, theo dõi nhóm trên các mạng xã hội hiện tại hoặc người giao báo



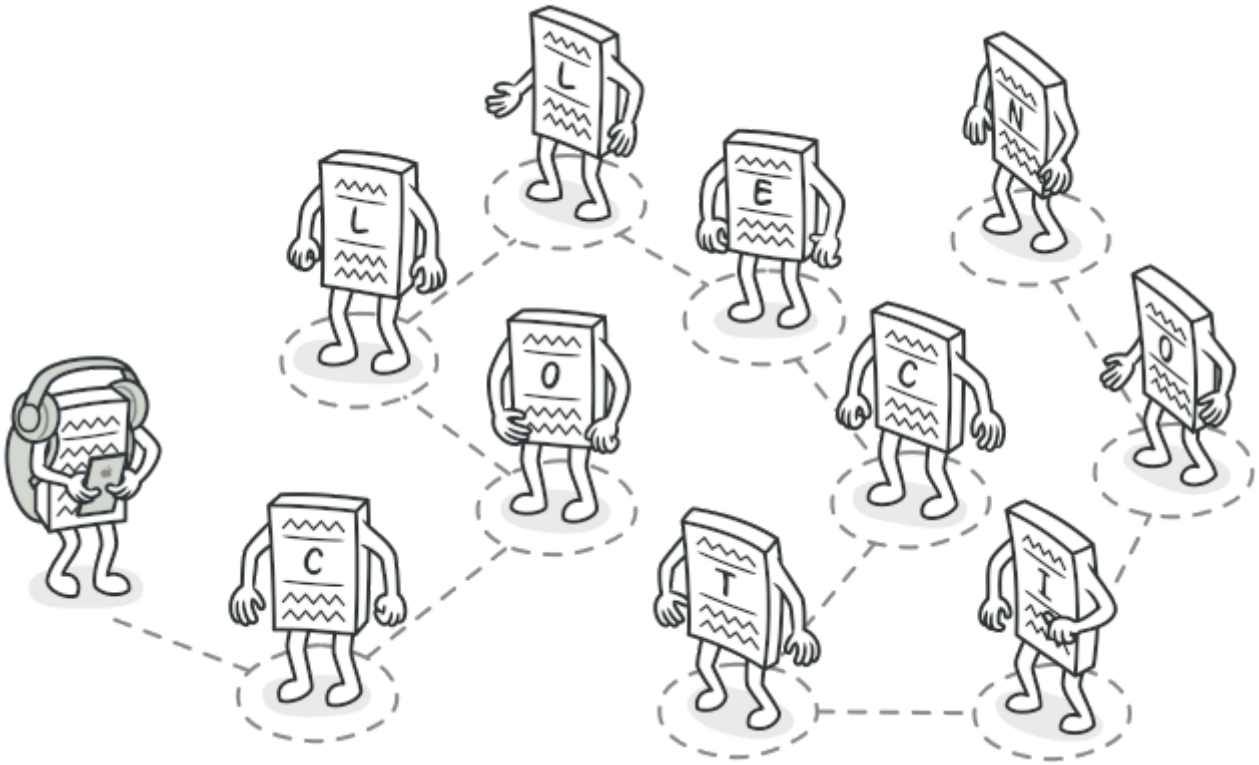
c. Ưu điểm:

- Giảm thiểu thao tác không cần thiết (5s gọi 1 lần để kiểm tra thay đổi hoặc chủ thể liên tục gửi thông báo định kỳ cho tất cả đối tượng,...)
- Thêm mới, gỡ bỏ đối tượng theo dõi mà không ảnh hưởng tới mã nguồn của chủ thể và ngược lại
- Tạo liên kết giữa chủ thể và đối tượng lắng nghe bất cứ khi nào (runtime)

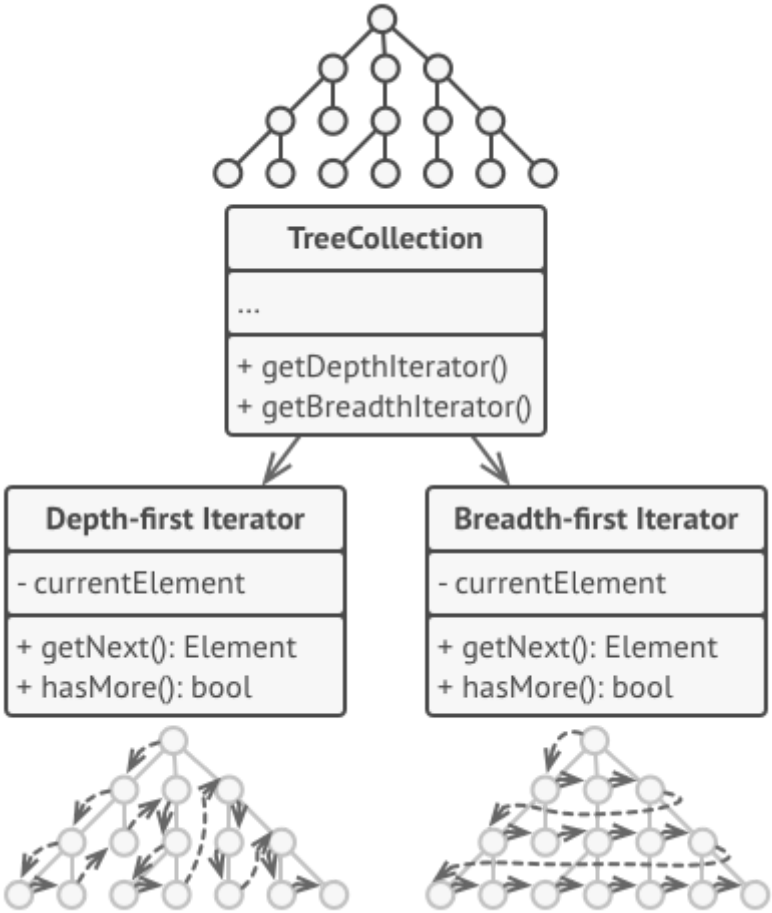
d. Nhược điểm:

- Các đối tượng lắng nghe sẽ nhận được thông báo theo thứ tự ngẫu nhiên

1.3 Iterator pattern



Tạo ra cơ chế để duyệt qua một mảng đối tượng mà không bị phụ thuộc vào lớp kiểu của đối tượng đó



1.4 Functional programing

Rất phổ biến trong JavaScript

- Pure function:
  Pure functions have lots of properties that are important in functional programming, including referential transparency (you can replace a function call with its resulting value without changing the meaning of the program)
  - Đầu vào như nhau thì đầu ra cũng phải như nhau
  - Không có tác dụng phụ (`Changing something somewhere`)
- Avoid shared state: Tránh tồn tại các đối tượng lưu giữ trạng thái giữa các function
- Mutable data: Dữ liệu truyền vào thường ở dạng reference, việc thay đổi thuộc tính nào đó của chúng gây ra việc thay đổi trạng thái của đối tượng ở bên ngoài phạm vi của hàm gọi (chính là tạo ra 1 trong các yếu tố của side-effects)
- Declarative instead of imperative
  - Declarative:

```
List<Integer> result = new ArrayList<>();
for (int number : collection) {
        if (number & 1 != 0)
                result.add(number)
}
```

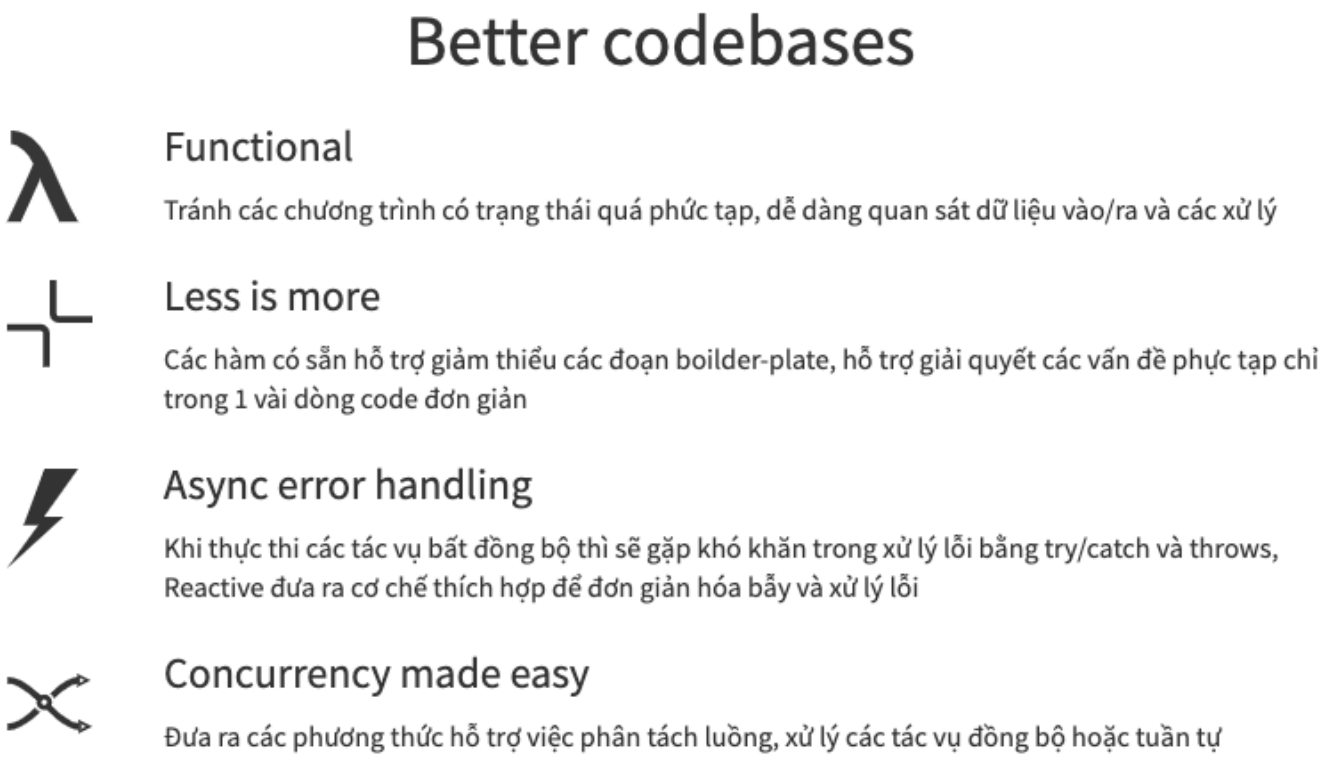
- Imperative

```
val result = collection.filter{number and 1 != 0}
```

Imperative	**Declarative**
- Khởi tạo mảng kết quả - Duyệt giá trị đầu vào - Kiểm tra giá trị, nếu là số lẻ thì thêm vào mảng kết qủa	- Tôi muốn kết quả là tất cả các số lẻ trong mảng đầu vào này

1.5 Why Rx?

- Phổ biến
  - Frontend:
    Thao tác với các sự kiện UI, phản hồi từ API với `RxJS` , `Rx.NET` và `RxJava` ,...
  - Crossplatform
    Đã hỗ trợ cho `Java` `Scala` `C#` `C++` `Clojure` `JavaScript` `Python` `Groovy` `...`
  - Backend
    Tập trung vào tính bất đồng bộ của `ReactiveX` để cho phép thực hiện các tác vụ `đồng thời` hoặc `độc lập`
- Better codebase



2. Understanding about Rx

2.1 A Simple case

```
Observable.fromArray(1,2,3,4,5,6,7,8,9,10)
                .filter{it and 1 != 0}
                .subscribe{println("Get $it")}
```

`CREATE` : Dễ dàng khởi tạo 1 luồng dữ liệu hoặc 1 luồng sự kiện

```
        Observable.fromArray(1,2,3,4,5,6,7,8,9,10)
```

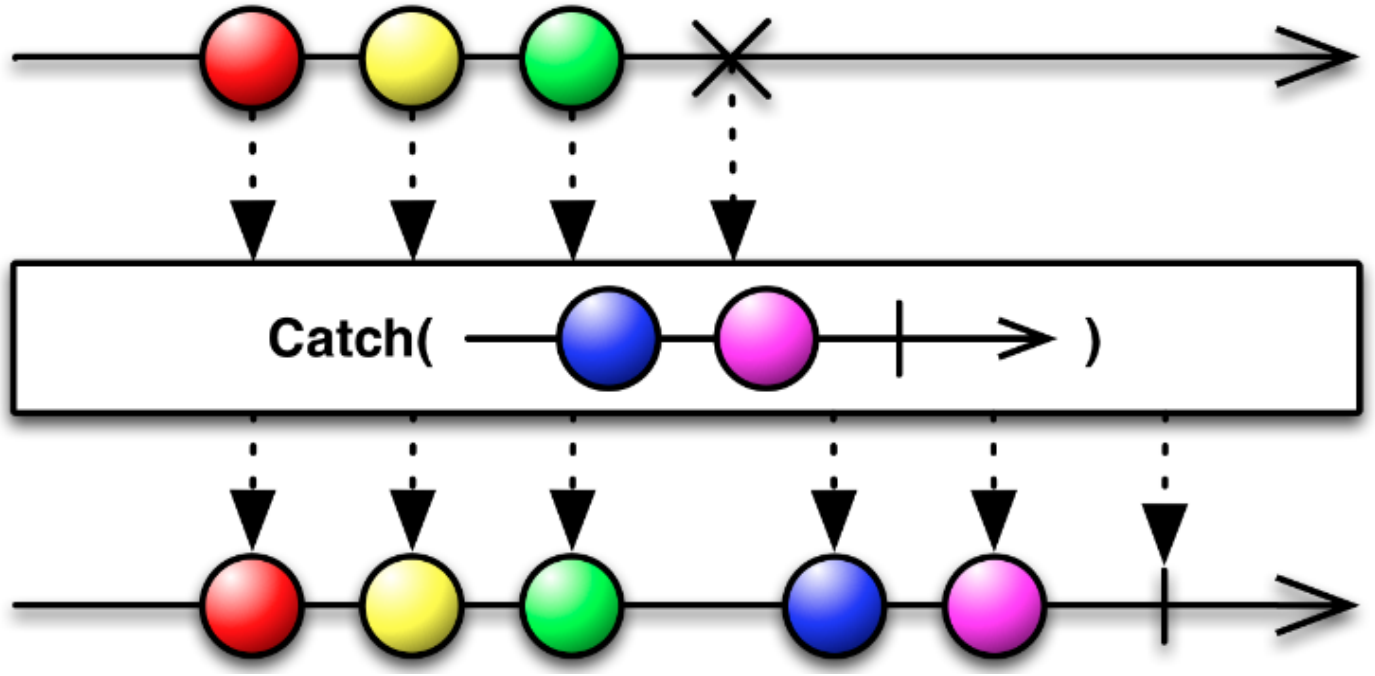
`COMBINE` : Đưa ra các xử lý, biến đổi dữ liệu, sự kiện

```
                .filter{it and 1 != 0}
```

`LISTEN` : Lắng nghe từ các luồng `Observable` để thực thi các hành động khác (hiển thị lên giao diện,...)

```
            .subscribe{println("Get $it")}
```

2.2 Marble diagram



- Mũi tên: dòng thời gian
- Đường line trên cùng: nguồn phát `Observable`
- Khối block: Các xử lý `Operation`
- Đường line cuối cùng: Dữ liệu hoặc sự kiện ở đối tượng lắng nghe `Observer`
- Dấu `X` : Lỗi `Exception`
- Dấu gạch dọc: Tín hiệu kết thúc `Completion`

3. Rx elements

3.1 Observable:

- Được hiểu như 1 đối tượng phát ra dữ liệu hoặc sự kiện (từ đây sẽ gọi là `item`)
- Thường đi với việc phát ra dữ liệu
- Các `Observer` đăng ký theo dõi các `Observable`
- Có thể là 0, 1 hoặc N `item`
- Dừng lại khi phát sinh lỗi

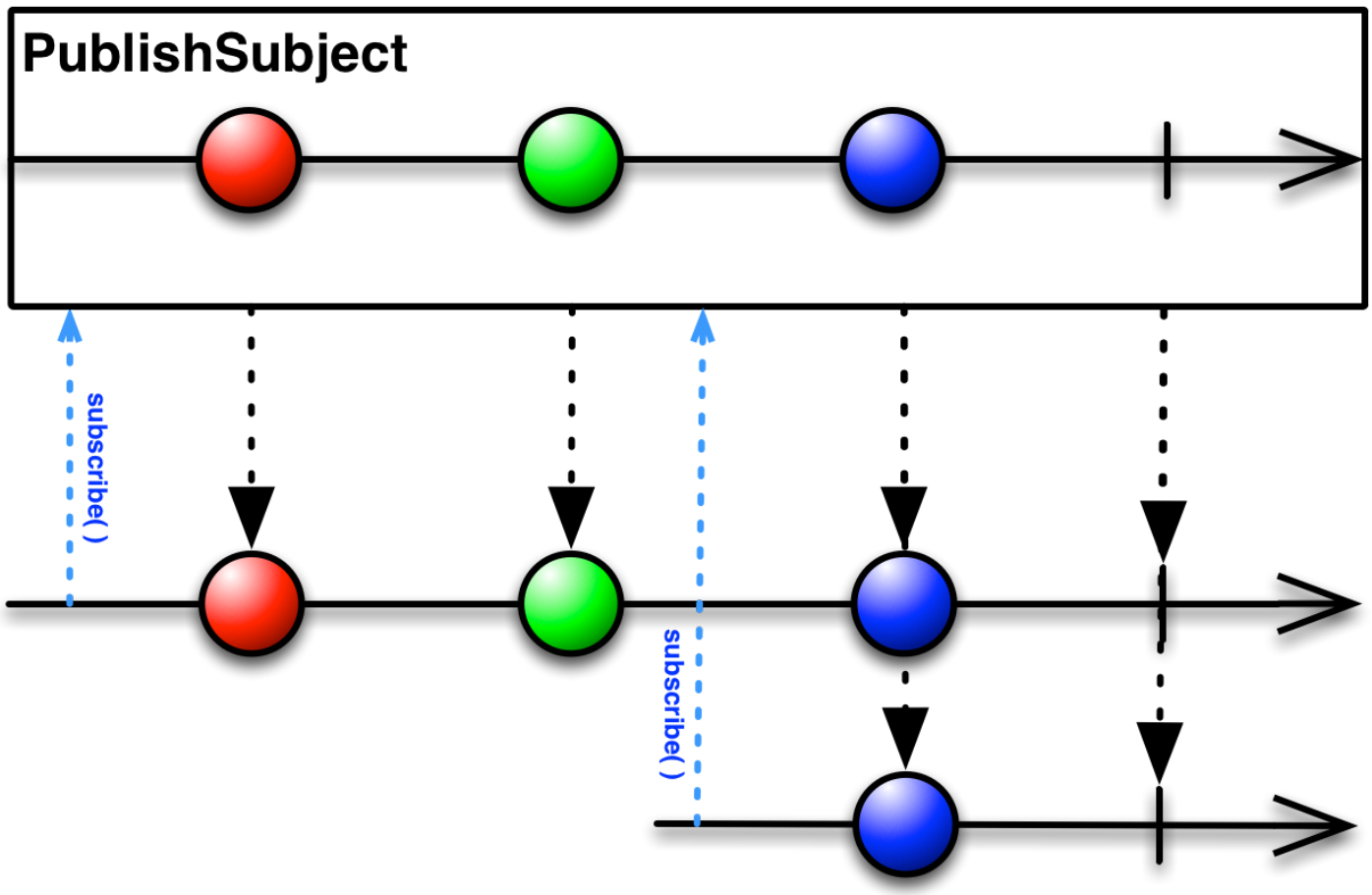
3.2 Operator:

- Được chia các loại cơ bản:

- Khởi tạo: `Create` `Defer` `Just/Empty/Never/Throw` `From...`
- Biến đổi: `Buffer` `Map/FlatMap/ConcatMap` `Window`
- Lược bỏ: `Debounce` `Distict` `Filter` `...`
- Xử lý lỗi: `Retry` `Catch`
- Tiện ích: `Delay` `Do...` `ObserverOn/SubscribeOn`
- `...`

- A Decision Tree of Observable Operators

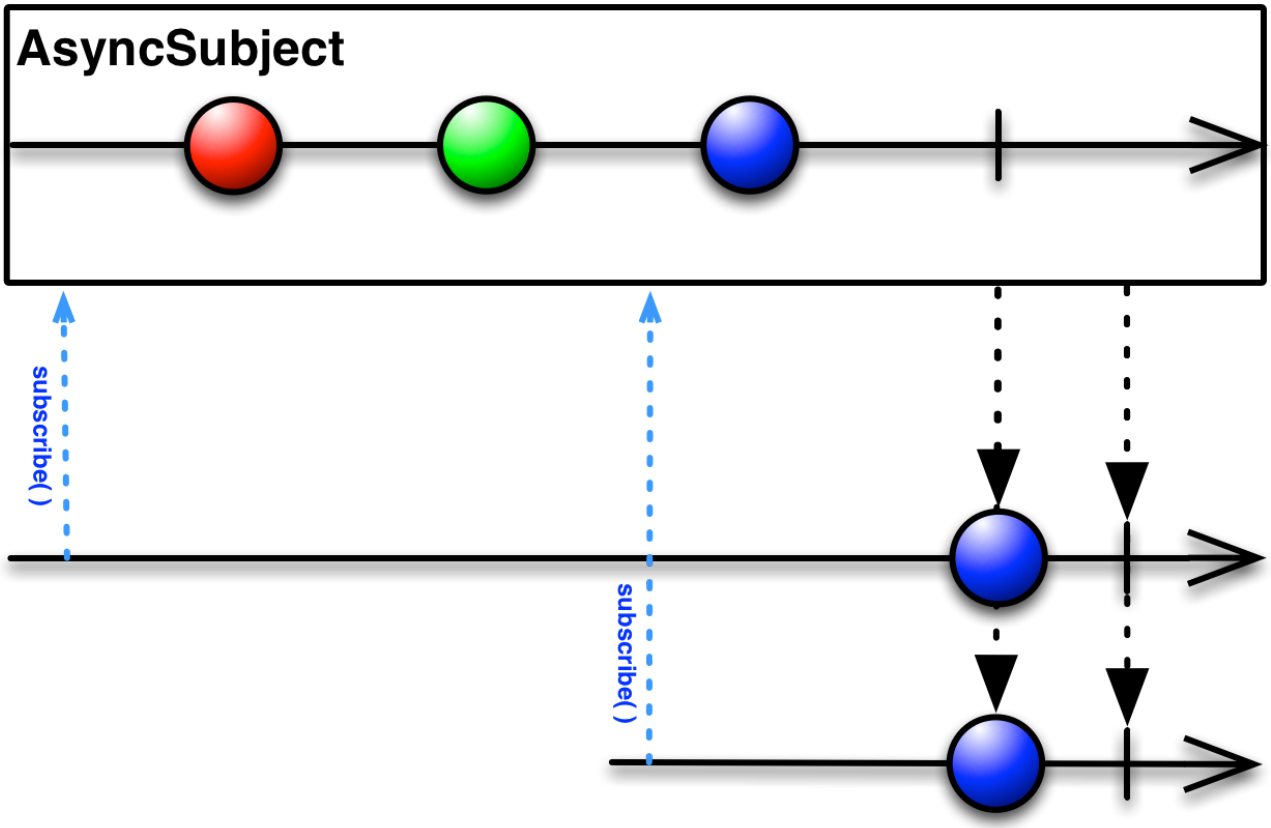
3.3 Subject



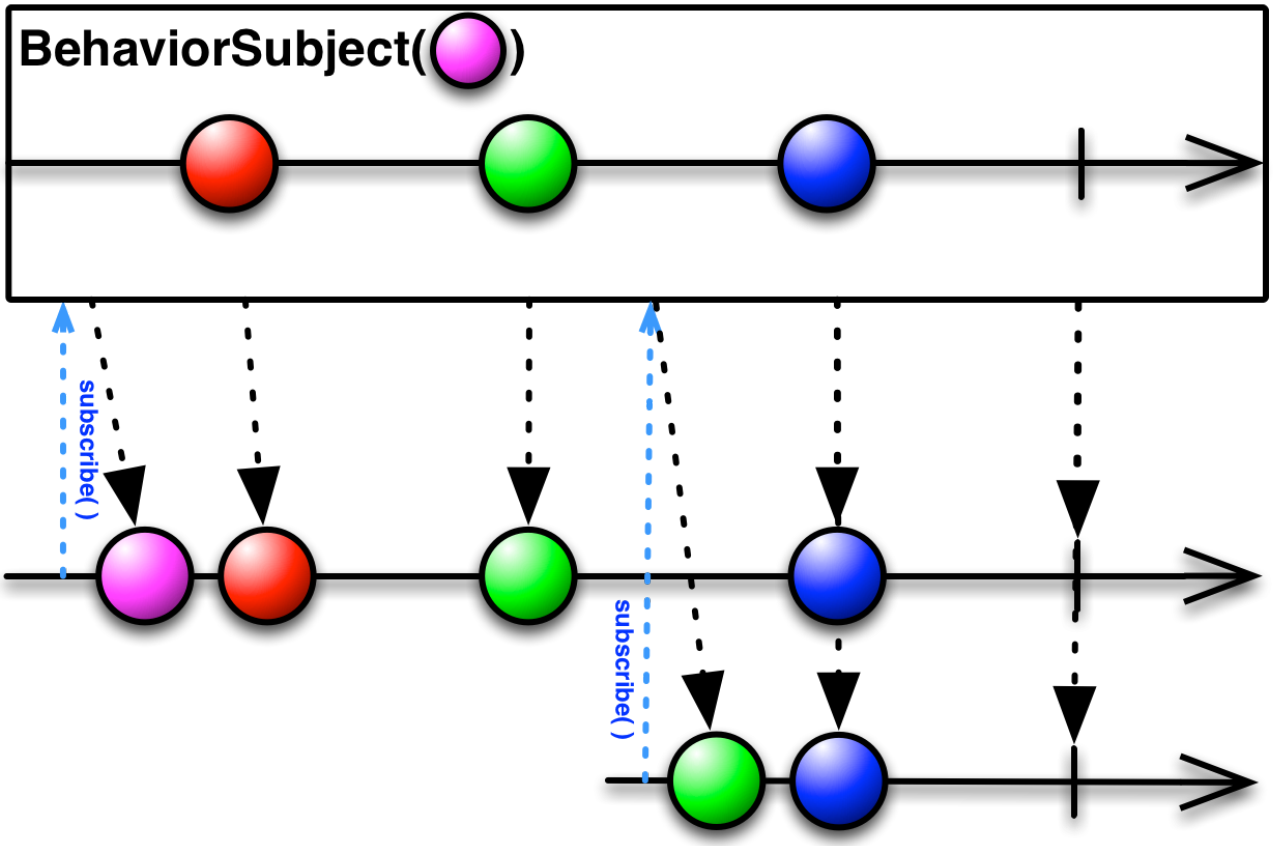
> Khi bắt đầu với Rx, thường sẽ suy nghĩ coi `Observer` như 1 `Consumer` , `Writer` , `Reader` còn `Observable` như 1 `Publisher` hay `Provider`
>
> Vấn đề phát sinh khi bạn tìm cách tự tạo ra 1 `Observable` từ 1 nguồn dữ liệu không có sẵn (các event,...) lúc này bạn sẽ cần 1 phương thức để có thể đẩy ra item `onNext` đẩy ra lỗi `onError` hoặc tuyên bố kết thúc `onComplete` . 3 Methods này thực sự giống với 3 methods của 1 `Observer` . Lúc này thì có vẻ như nó trông giống 1 `Observer` mặc dù ý tưởng khi bắt đầu nó là 1 `Observable` . Lúc này chúng ta cần 1 lớp implement cả 2 interfaces kể trên
>
> **AND WE HAVE** `Subject` . Với 1 `Subject` , bạn hoàn toàn có thể trả ra 1 đối tượng kiểu `Observable` để đối tượng khác lắng nghe. Nhưng đồng thời trong đó, bạn có thể gọi các phương

> thức `OnNext/OnError/OnComplete` để kiểm soát dữ liệu được phát ra

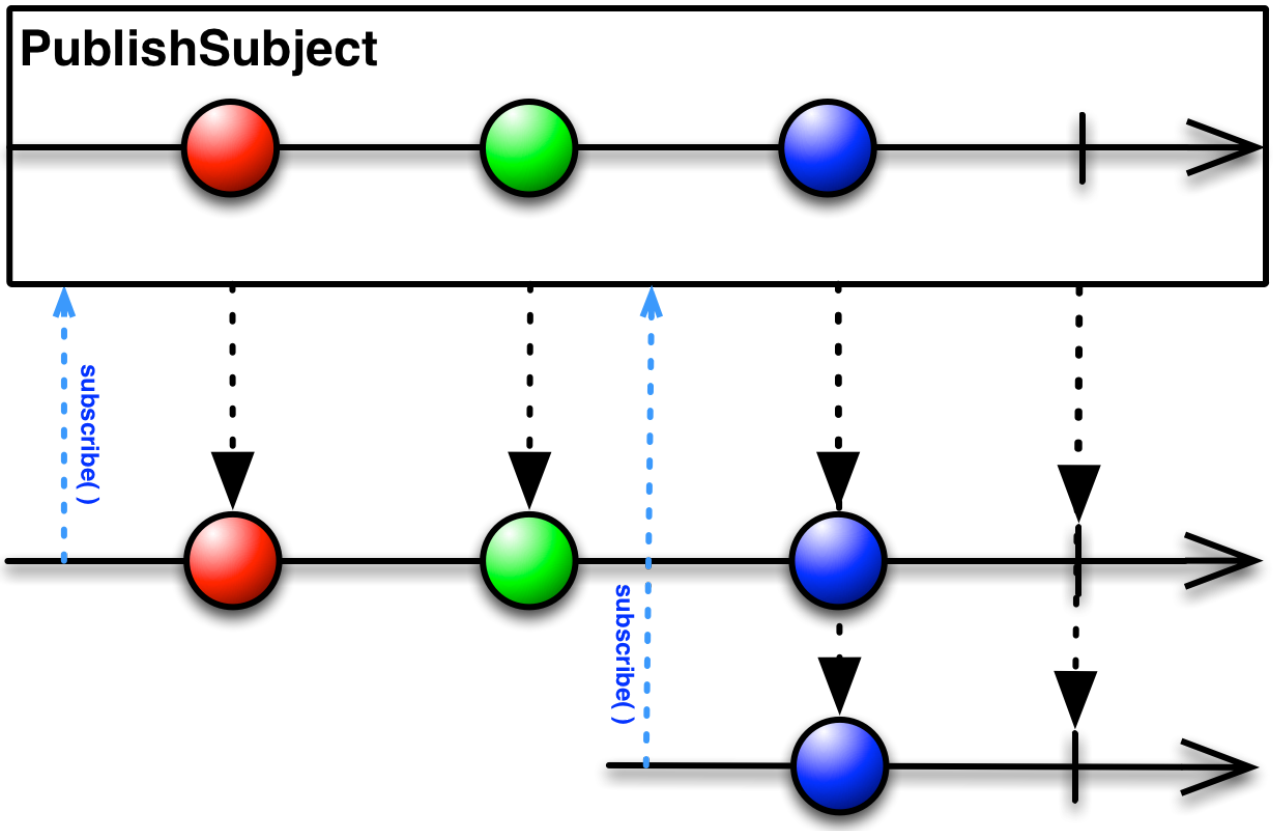
- Phục vụ cho multi-casting
- Có thể coi như 1 `Observable` hoặc 1 `Observer`
  - như 1 `Observer` : Có thể quan sát 1 hoặc nhiều `Observable` khác
  - như 1 `Observable` : Có thể phát lại các `item` cho các `Observer` ...
- Thường ứng dụng để phát ra các `event` (hot observable)
- Có nhiều loại `Subject`
  - `AsyncSubject` : Bài toán về `câu hỏi trắc nghiệm`



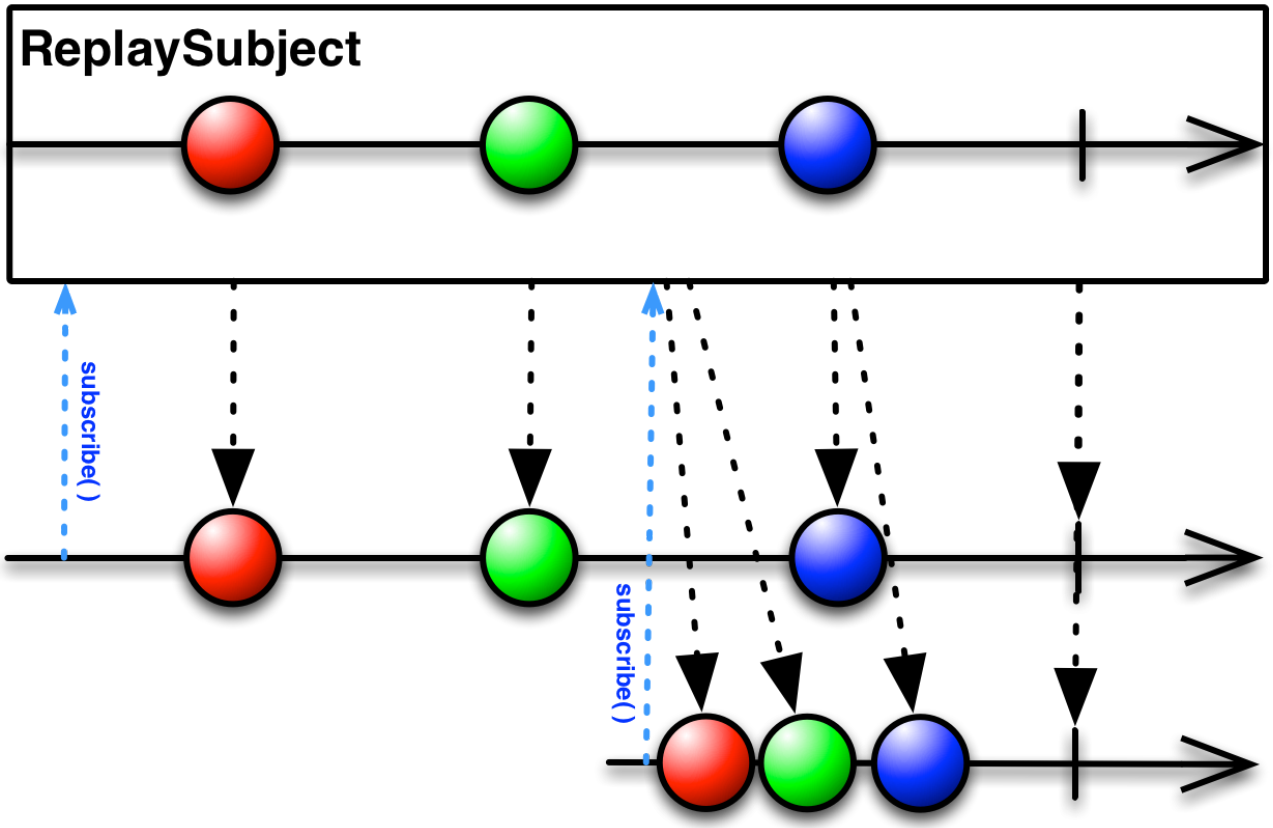
  - `BehaviorSubject` : Bài toán về `ô tìm kiếm` , `ui-switch`



- `PublishSubject` : Bài toán về `sự kiện click`



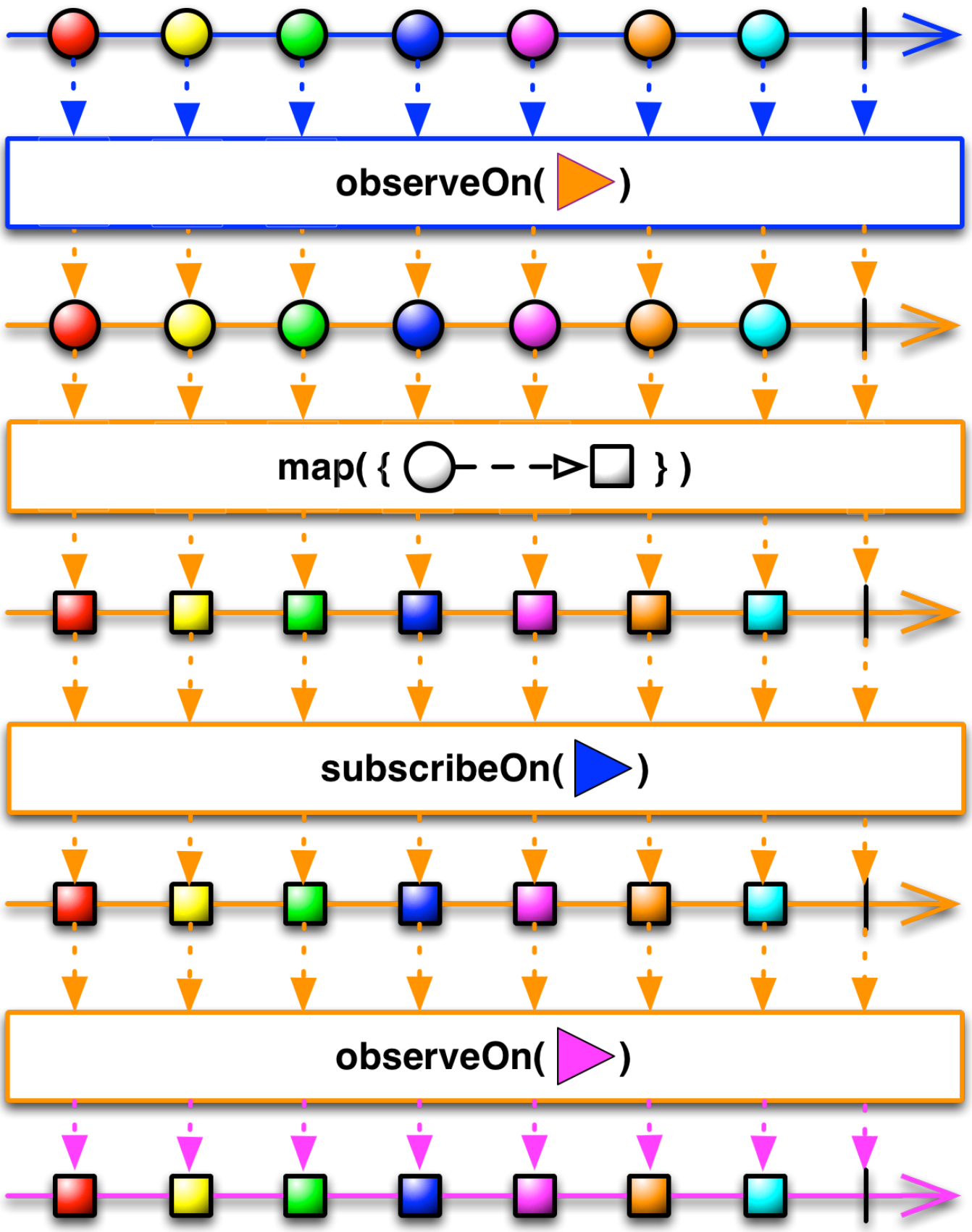
- `ReplaySubject` : Tương đồng với `BehaviorSubject` nhưng kèm theo tham số `buffer` (thời gian hoặc số lượng item), có thể nhận item kể cả khi nguồn đã completed nhưng bù lại sẽ không thể có giá trị khởi tạo như `BehaviorSubject`



Ekko ultimate shadow

3.4 Scheduler:

- Phục vụ cho việc multithreading
- Khai báo `làm cái gì ở đâu`



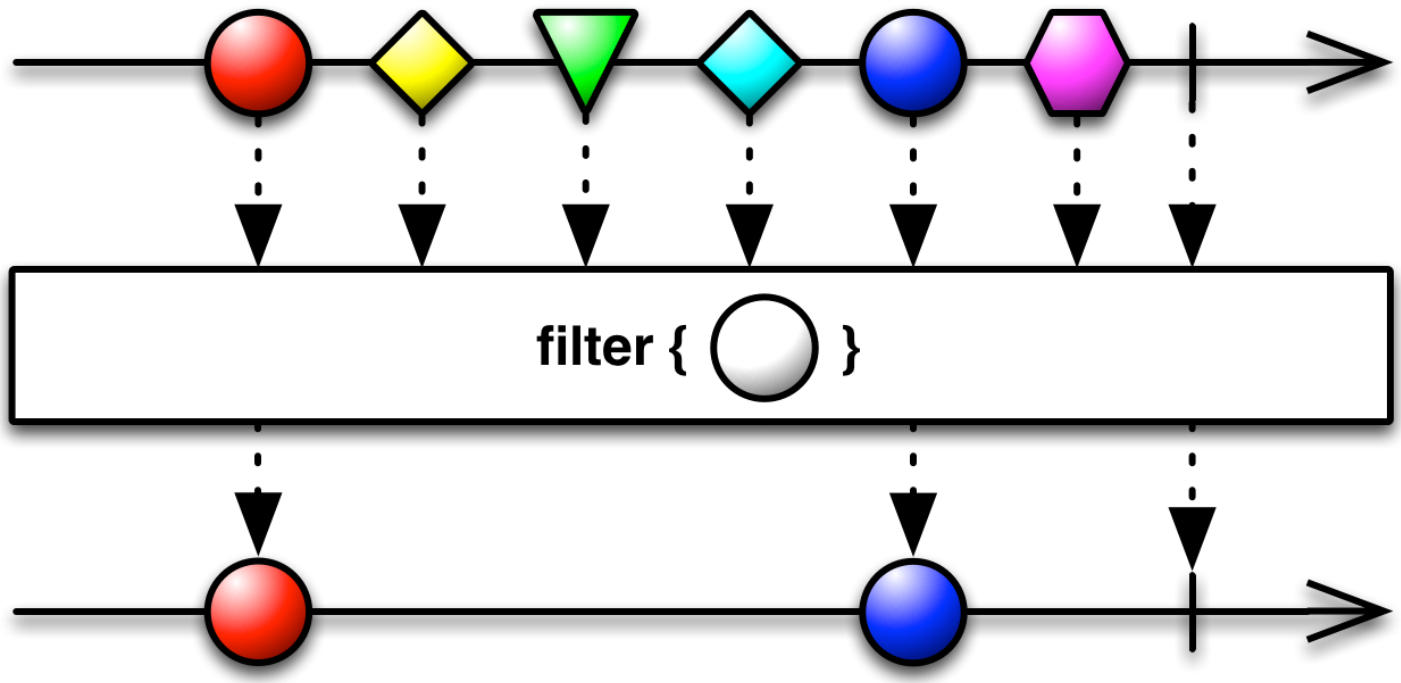
- So sánh

Tiêu chí	ObserverOn	SubscribeOn
Mặc định	Tương đương SubscribeOn	Luồng nơi hàm `subscribe` được gọi

Tiêu chí	ObserverOn	SubscribeOn
Tác dụng	Downstream	Upstream
Phạm vi	Tất cả operation cho tới khi xuất hiện `ObserverOn` khác	Chỉ có tác dụng ở lần gọi đầu tiên những lần gọi sau sẽ không có tác dụng
Số lần gọi	0..N	0..1

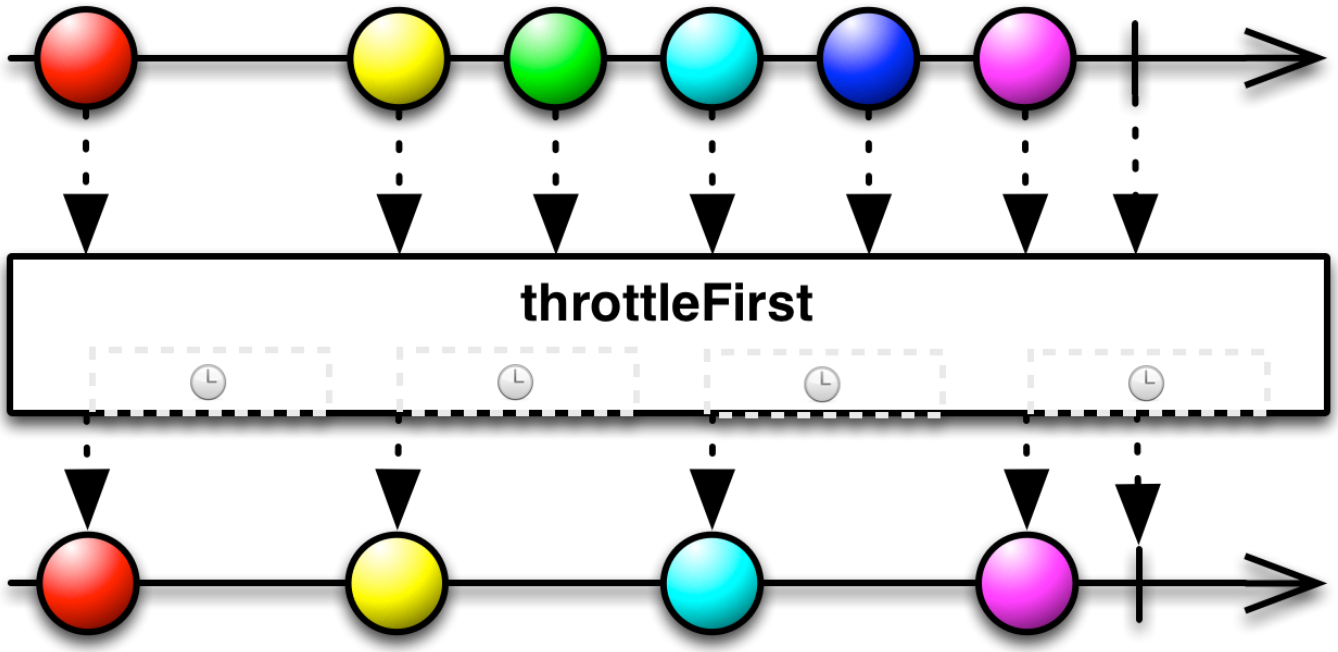
4. Usecase

4.1 Array `filter` :



4.2 Dodge multiclick with `throtleFirst`

- Vấn đề: Người dùng có thể ấn liên tục vào 1 button trong thời gian ngắn dẫn tới luồng xử lý bị sai:
  - Gửi form 2 lần liên tiếp
  - Duplicate chuyển màn hình (chồng chéo màn hình, backpress nhiều màn hình)
- Giải pháp: Khi nhận được sự kiện `onClick` từ 1 `Widget` , xử lý event đồng thời tạm dừng nhận `onClick` event từ `Widget` đó trong một khoảng thời gian ngắn



- Thực thi:
  - Tạo nguồn phát sự kiện

```
val onClickSubject : PublishSubject<Long> =  PublishSubject.create()
anyView.setOnClickListener{onClickSubject.onNext(System.currentTimeMillis())}
```

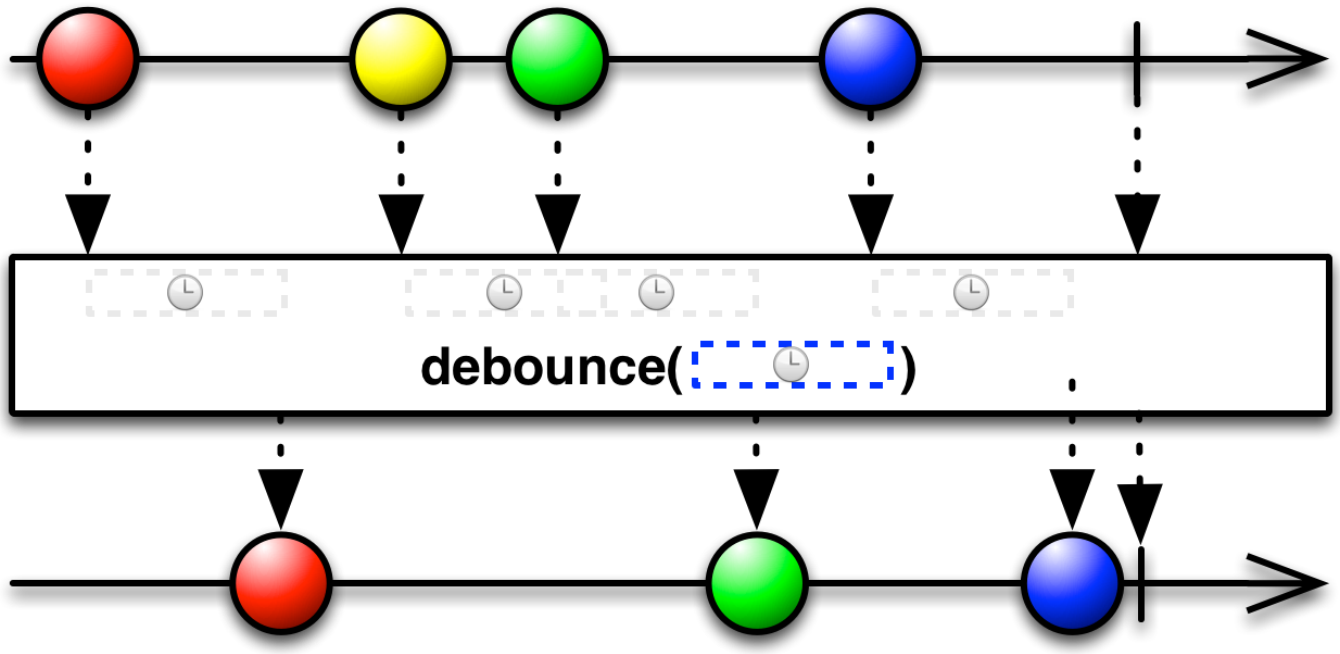
  - Xử lý tạm dừng nhận và đăng ký lắng nghe

```
var disposable = onClickSubject.throtleFirst(500, TimeUnit.MILLISECONDS)
                .subscribe{
                        //do onClickEvent
                }
```

  - **Tips**: Android có thể dùng thư viện RxBindings cho phần tạo nguồn phát

4.3 SearchView textchanges event with `debounce`

- Vấn đề: Người dùng nhập text vào 1 ô `SearchView` . Khi nào thì nên gửi query tới repository để lấy dữ liệu về mà vẫn đảm bảo
  - Không gửi quá nhiều (Người dùng cứ nhập text mới thì lại gửi lên) nhằm tối ưu hóa sử dụng pin, network
  - Tự động tìm kiếm theo từ khóa được nhập vào (giảm bớt thao tác ấn nút search) nhằm tối ưu hóa UI/UX
- Giải pháp: Sau khi người dùng nhập xong thì mới gửi query tới repository. Sự kiện người dùng nhập xong được tính là sau một khoảng thời gian ngắn không phát sinh thêm sự kiện `textChanges`



- Thực thi:
  - Tạo nguồn phát sự kiện:

```
val keywordSubject = BehaviorSubject.create("")
searchView.textChangesEvent = {
    if (keyword == null || keyword.isEmpty) return
    keywordSubject.onNext(keyword)
}
```

  - Xử lý sự kiện:

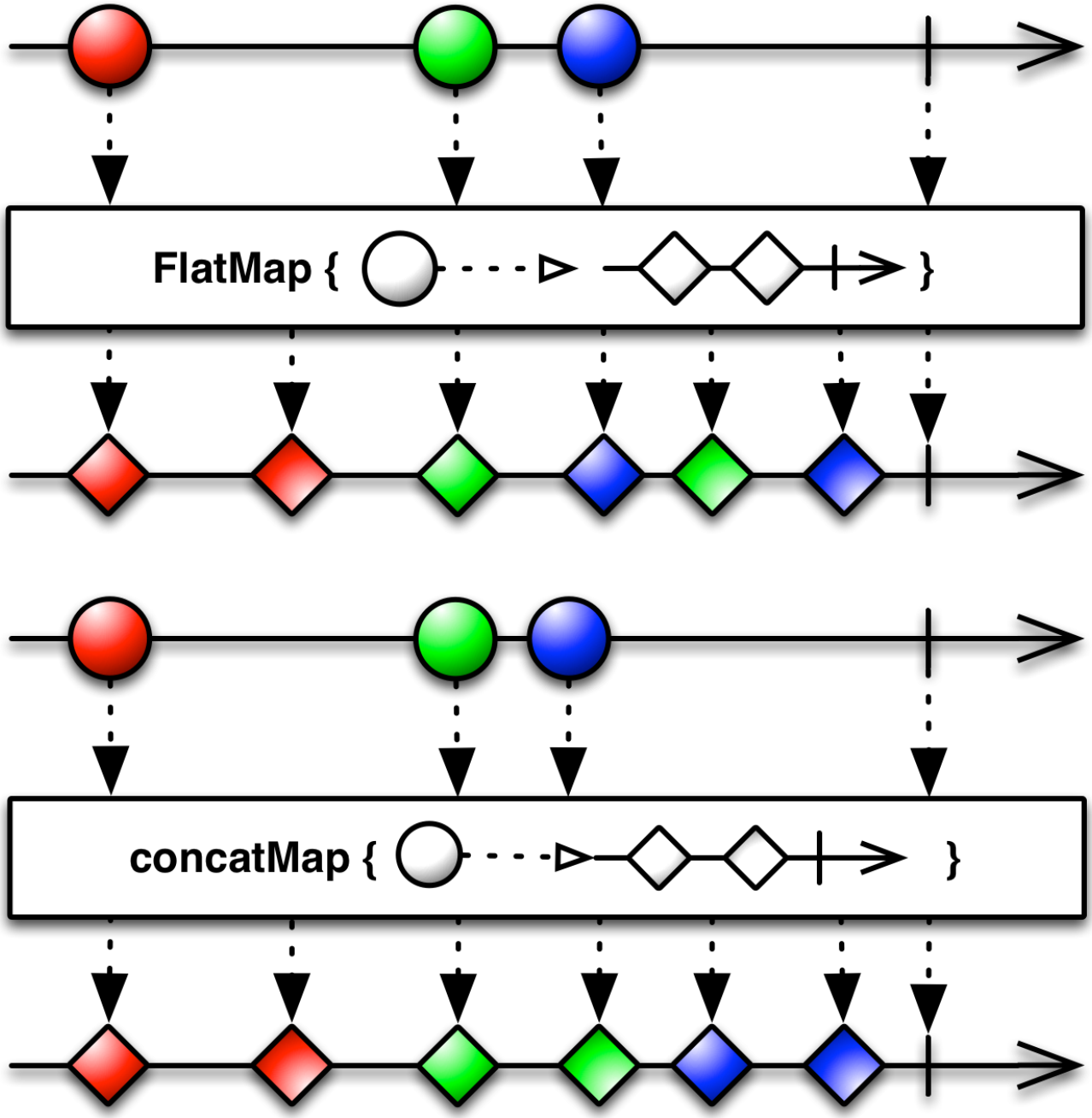
```
var disposable = keywordSubject.debounce(500, TimeUnit.MILLISECONDS)
                .subscribe{
                        //query to repository
                }
```

  - **Tips**:
    - Android có thể sử dụng thư viện RxBindings cho phần tạo nguồn phát
    - Đối với `SearchView` có suggestion, thì phần query suggestion có thể sử dụng `ThrotleFirst` hoặc `ThrotleLast`

4.4 Upload Amazon S3 presign url with `flatMap` and `concatMap`

- Vấn đề:
  - Upload ảnh không chỉ là gọi 1 api mà phải gọi tới 3 api
  - Độ phức tạp tăng lên nếu việc upload không phải 1 mà là nhiều file
  - Thường được giải quyết ở phía server hiệu quả không cao (luồng byte data phải đi lòng vòng)
- Giải pháp:

- Đưa ra cơ chế chaining api (tuần tự: lấy url presign -> thực hiện upload -> lấy thông tin upload được, cập nhật lên server)
- Tuần từ upload từng file hoặc đồng thời nhiều file (có giới hạn số file đồng thời)



- Thực thi:
  - Tạo dữ liệu file cần upload:

```
val uploadObservable = Observable.fromArray(...)
```

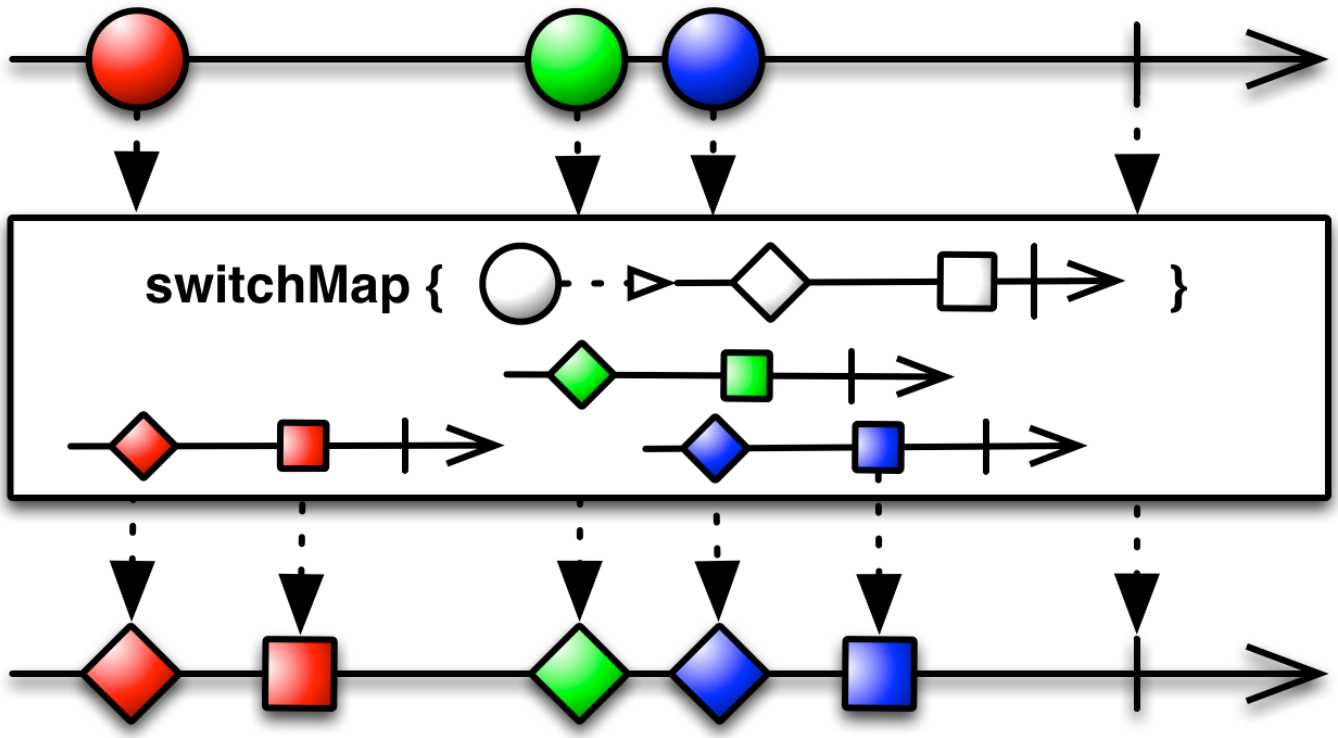
  - Chaining api với flatMap và lắng nghe kết quả

```
uploadObservable.subscribeOn(Schedulers.io)
  .flatMap(this::doGetPresignUrlAndUpload, this::updateModelWithUploadResult)
  .flatMap(this::doSubmitCompletedMode)
  .observerOn(AndroidSchedulers.mainThread())
  .subscribe{
        //update UI
  }
```

- **Tips**
  - Tuần tự upload với `concatMap`
  - Nếu upload đồng thời nên giới hạn số lượng upload đồng thời với limit của `flatMap`

4.5 Optimize loading mail's body with `switchMap`

- Yêu cầu:
  - Dữ liệu danh sách từ repository chỉ là dữ liệu cơ bản (Title,...) mà chưa có nội dung
  - Yêu cầu hiển thị có title và 1 dòng ngắn gọn nội dung
  - Api lấy nội dung đơn lẻ với đầu vào là 1 `item` và đầu ra `item có body`
- Vấn đề:
  - Tối ưu hóa tải dữ liệu chi tiết để tránh dư thừa
  - Hiển thị dữ liệu ở mức độ nhanh nhất có thể
- Giải pháp:



  - Lấy dữ liệu cơ bản từ server
  - Hiển thị dữ liệu cơ bản
  - Khi người dùng scroll tới đâu

    - Thực hiện load body dựa trên item đang hiển thị trên màn hình
    - Dừng load với các item khác (nếu có)
    - Load tới đâu, cập nhật giao diện tới đó
- Thực thi:

5. References:

- Design pattern: https://refactoring.guru/design-patterns/catalog
- About functional programing: https://medium.com/javascript-scene/master-the-javascript-interview-what-is-functional-programming-7f218c68b3a0
- Meme: https://imgflip.com/memegenerator/129315248/No---Yes

6. Advanced topics

- Hot and Cold observables
- Subject, when to use, when not to use?
- Map vs FlatMap vs ConcatMap vs SwitchMap
- Throtle vs Debounce
- Error Handling