REACTIVEX

AGENDA

1 REACTIVEX ?

- What is Rx?
- Observer & Iterator pattern
- Functional programming

2 HELLO REACTIVE X

- A simple case
- Marble diagram

3 RX ELEMENTS

- Observable and Observer
- Operator
- Subject
- Scheduler

4 USECASE

- Array operation
- Dodge rapid clicking
- Query with Search View
- Presign uploading
- Optimize mail's body loading

5 ADVANCED TOPICS

REACTIVE X ?

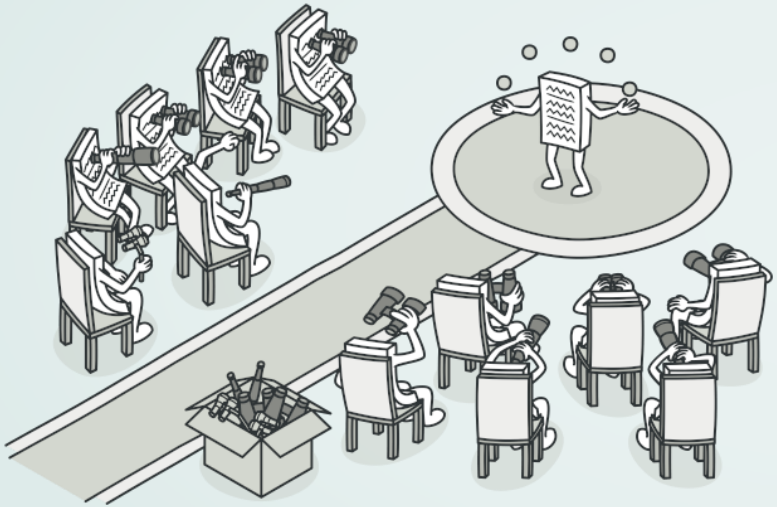
- An API for **asynchronous** programming with **observable** streams
- The combination of **observer** pattern, **iterator** pattern and **functional** programming

1.2. Observer pattern

Vấn đề

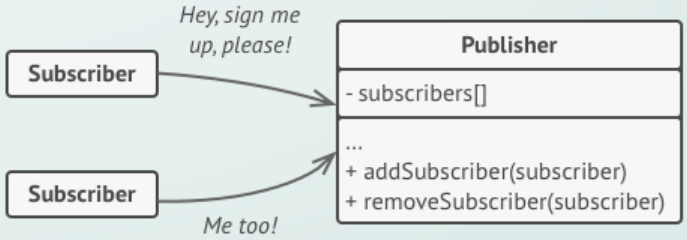
- Cập nhật giao diện khi có thay đổi
- Thực hiện hành động mỗi khi một sự kiện nào đó phát sinh

1.2. Observer pattern

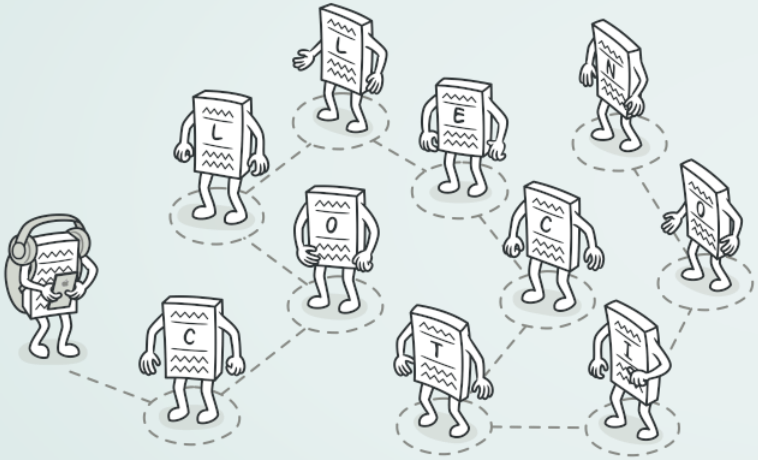


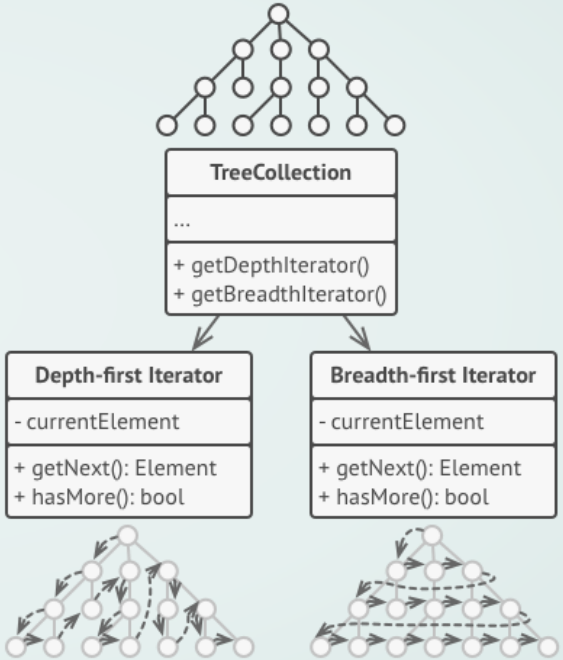
Giải pháp

- Cơ chế theo dõi
- Chỉ nhận được những gì mình quan tâm
- Chỉ gửi cho đối tượng đăng ký lắng nghe



1.3. Iterator pattern





1.4 Functional programming

- Pure function
- Hạn chế các đối tượng lưu giữ trạng thái được chia sẻ trong chương trình
- Hạn chế đối tượng có thể thay đổi (Avoid mutating objects)
- Hạn chế tác dụng phụ (side effects)
- Hướng tới khai báo (Declarative) thay vì ra lệnh (Imperative)

WHY RX? - Ubiquitous

- Cập nhật UI
- Xử lý dữ liệu từ API
- Tác vụ bất đồng bộ

FRONTEND

- RxJava
- RxKotlin
- RxSwipe
- Rx.Net
- RxJS
- ...

CROSSPLATFORM

Tập trung vào tính bất đồng bộ của ReactiveX để cho phép thực hiện các tác vụ đồng thời hoặc độc lập

BACKEND

WHY RX? – Better codebase

Better codebases

Functional

Tránh các chương trình có trạng thái quá phức tạp, dễ dàng quan sát dữ liệu vào/ra và các xử lý

Less is more

Các hàm có sẵn hỗ trợ giảm thiểu các đoạn boilder-plate, hỗ trợ giải quyết các vấn đề phực tạp chỉ trong 1 vài dòng code đơn giản

Async error handling

Khi thực thi các tác vụ bất đồng bộ thì sẽ gặp khó khăn trong xử lý lỗi bằng try/catch và throws, Reactive đưa ra cơ chế thích hợp để đơn giản hóa bẫy và xử lý lỗi

Concurrency made easy

Đưa ra các phương thức hỗ trợ việc phân tách luồng, xử lý các tác vụ đồng bộ hoặc tuần tự

HELLO REACTIVE X

- A Simple case
- Marble diagram

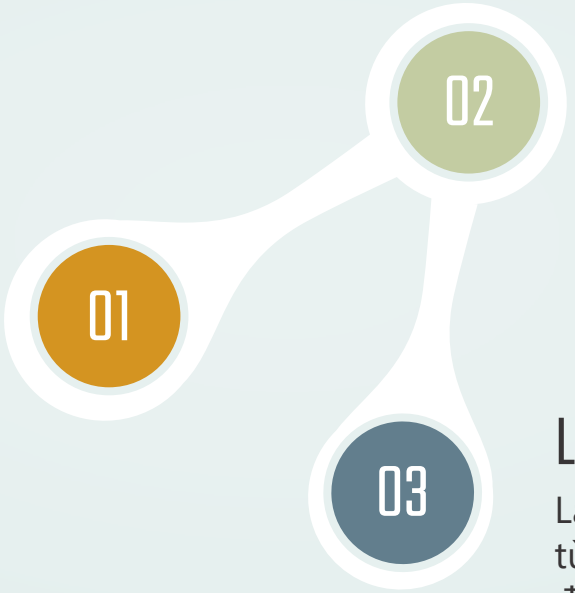
A SIMPLE CASE

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

A SIMPLE CASE

CREATION

Tạo ra nguồn phát từ một mảng, một hàm đồng bộ, bất đồng bộ thông qua các hàm tiện ích có sẵn



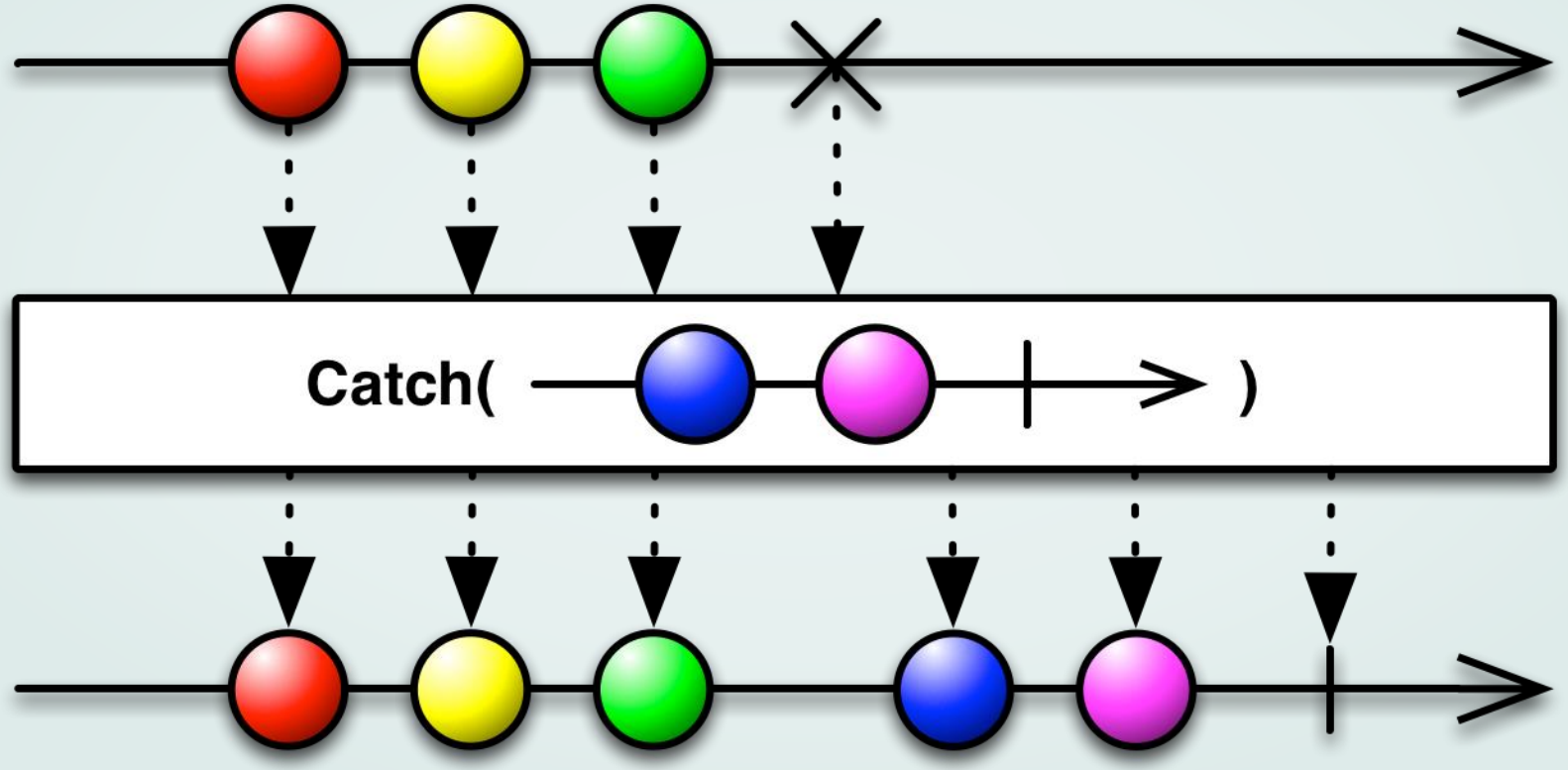
COMBINE

Chuỗi các xử lý, kết hợp, biến đổi dữ liệu

LISTEN

Lắng nghe dữ liệu phát ra từ luồng và thực thi hành động

MARBLE DIAGRAM



RX ELEMENTS

- Observable
- Operator
- Subject
- Scheduler

OBSERVABLE

	OBSERVABLE	SINGLE	FLOWABLE	COMPLETABLE
ITEMs	0,1..N	1	0,1..N	0
BACKPRESSURE	KHÔNG	KHÔNG	CÓ	KHÔNG
OUTPUT	• onNext • onComplete • onError	• onSuccess • onError	• onNext • onComplete • onError	• onComplete • onError

OPERATION

CREATION

Khởi tạo nguồn phát

FILTERING

Lọc, tổng hợp, lược bỏ dữ liệu

TRANSFORMING

Chuyển đổi định dạng, số lượng dữ liệu,...

ERROR HANDLING

Bắt và xử lý lỗi

AND SO ON...

SUBJECT

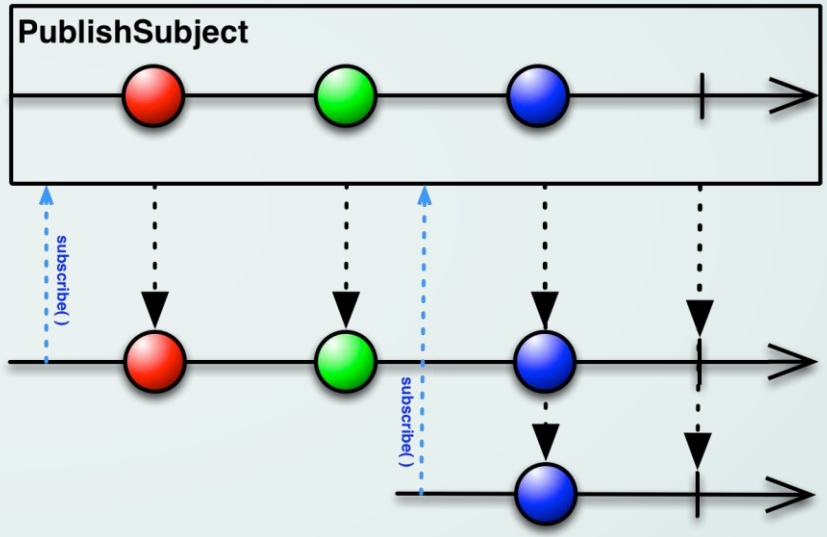
PUBLISH

Nhận item từ khi subscribe

BEHAVIOR

ASYNC

REPLAY



SUBJECT

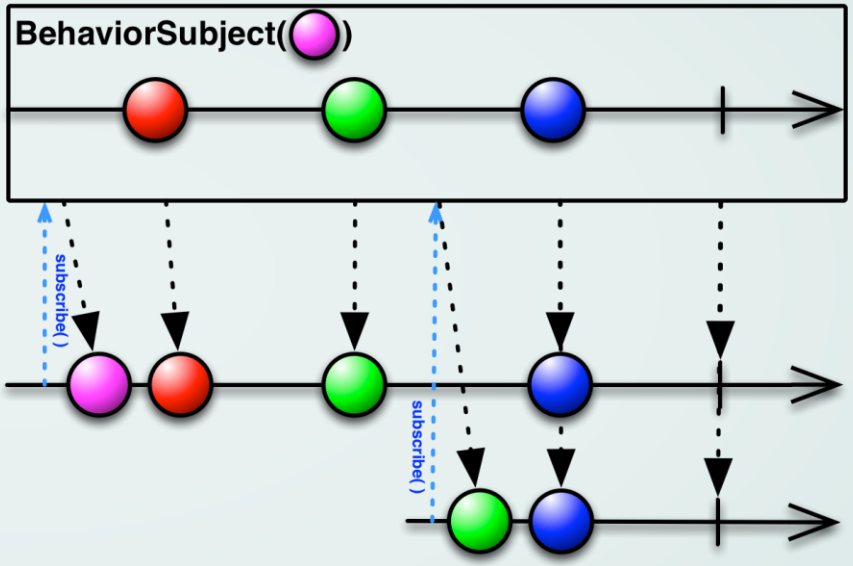
PUBLISH

BEHAVIOR

Nhận item mặc định hoặc item gần nhất và sau đó thì như subject

ASYNC

REPLAY



SUBJECT

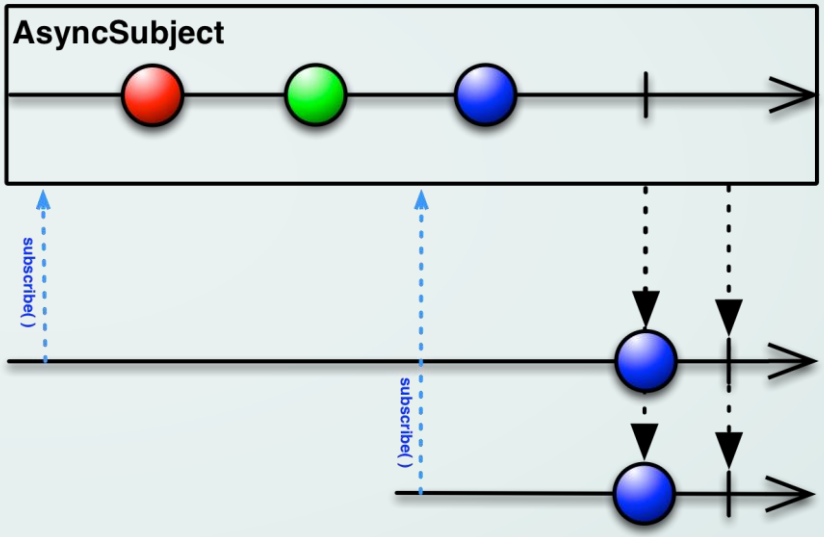
PUBLISH

BEHAVIOR

ASYNC

Chỉ nhận được item cuối cùng

REPLAY



SUBJECT

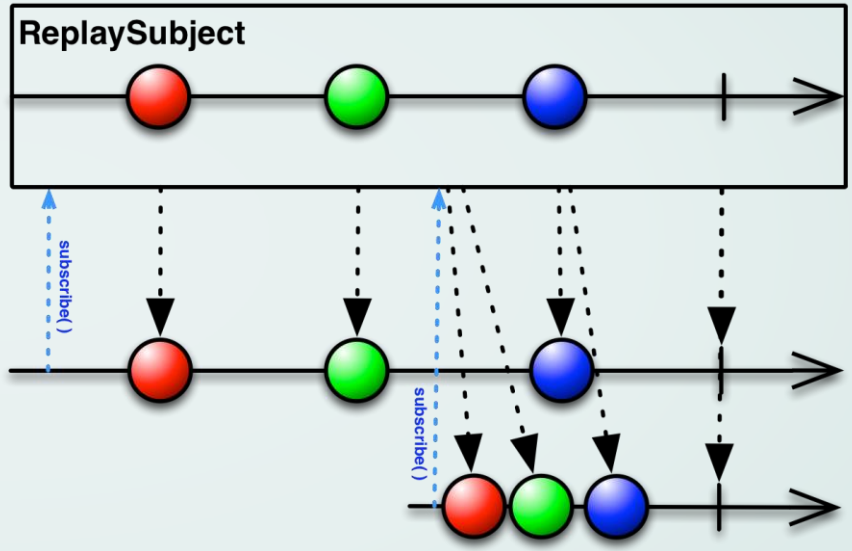
PUBLISH

BEHAVIOR

ASYNC

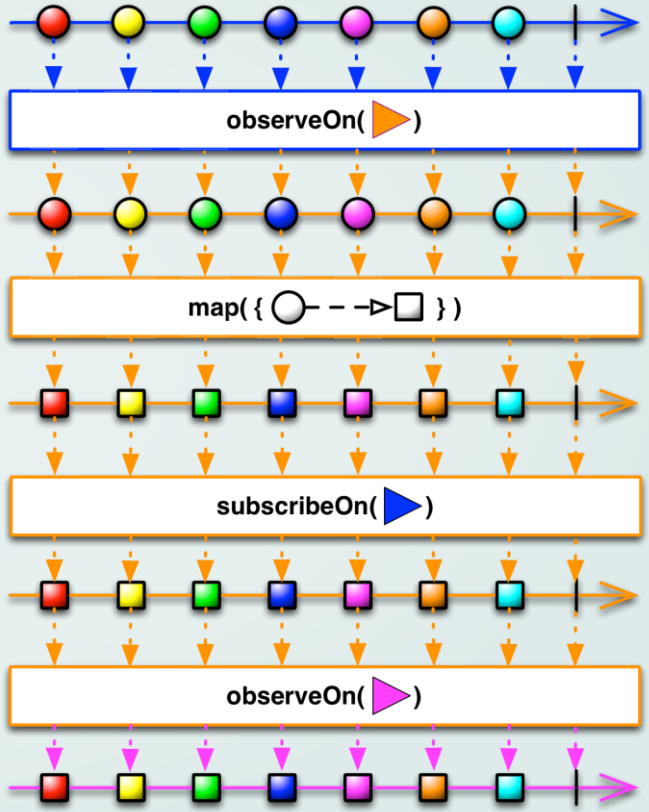
REPLAY

Tương đồng với behavior nhưng có thêm buffer, có khả năng phát lại item



SCHEDULER

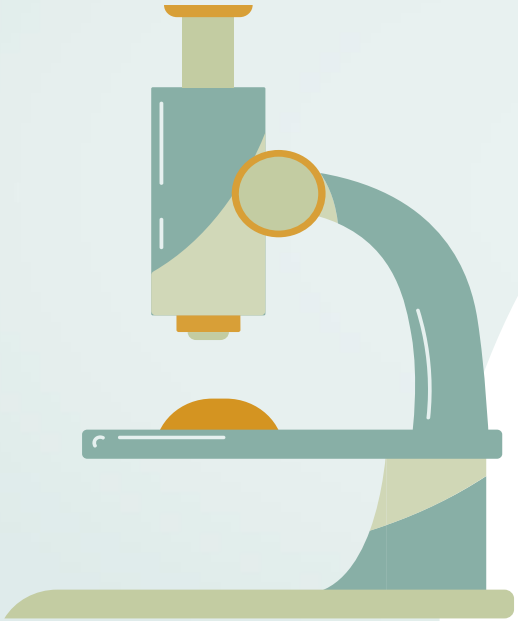
Tiêu chí	ObserverOn	SubscribeOn
Mặc định	Tương đương SubscribeOn	Luồng nơi hàm subscribe được gọi
Tác dụng	Downstream	Upstream
Phạm vi	Tất cả operation cho tới khi có hàm ObserverOn mới được gọi	Chỉ có tác dụng ở lần gọi đầu tiên
Số lần gọi	0..N	0..1



USECASES

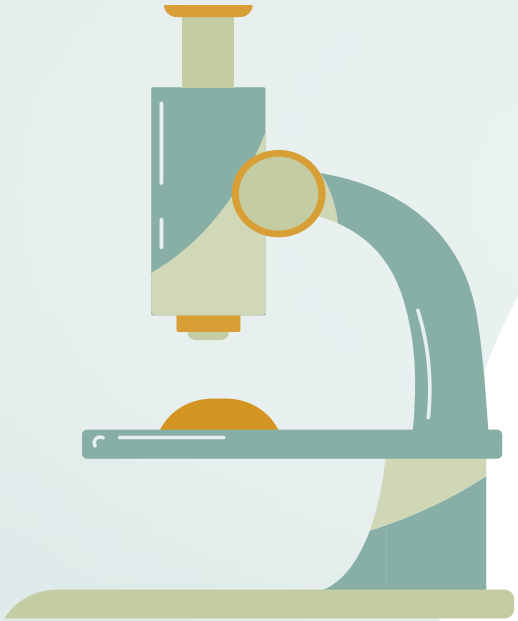
Some issues which **ReactiveX** can give you a hand

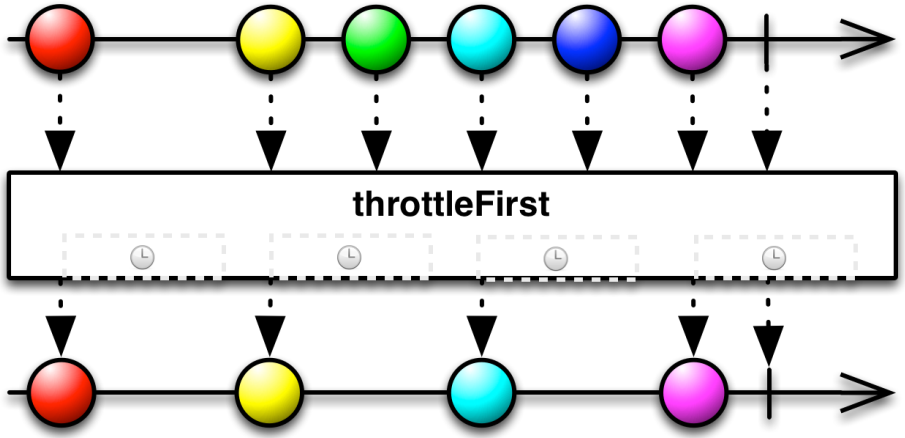
DODGE RAPID CLICK WITH THROTLEFIRST



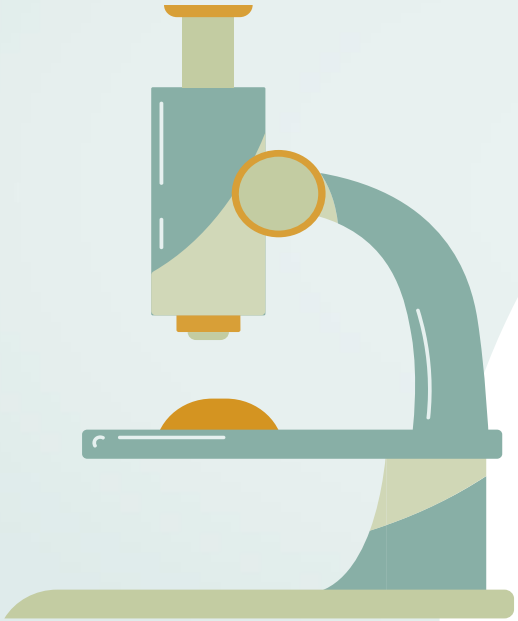
- Tránh mở nhiều màn hình
- Tránh gửi form trùng lặp nhiều lần

DODGE RAPID CLICK WITH THROTLEFIRST



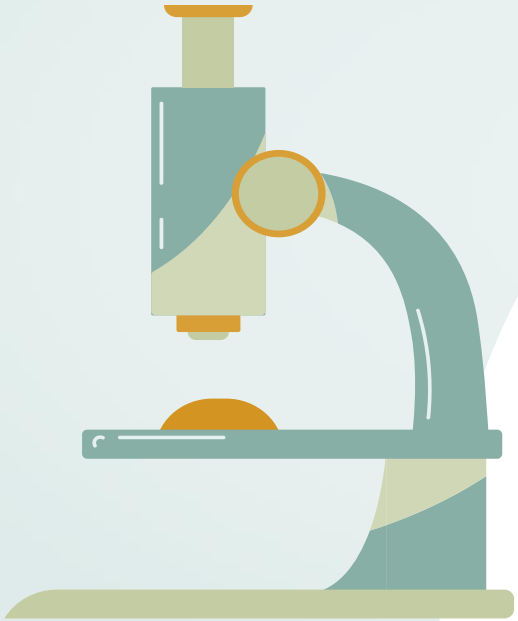


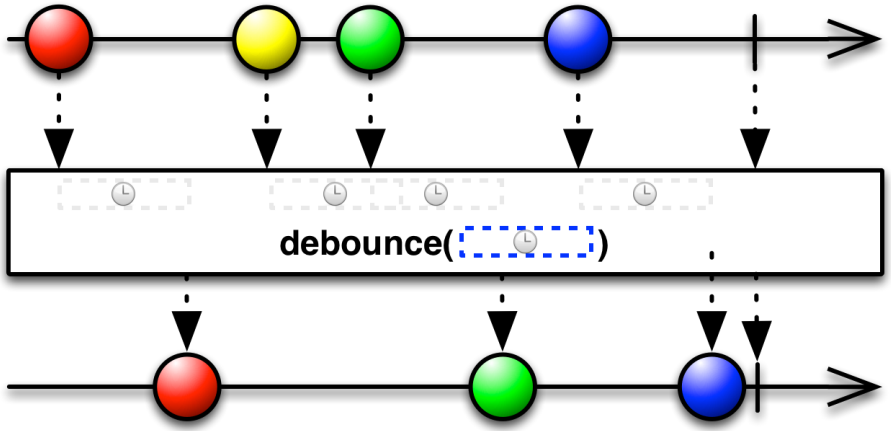
SEARCH-VIEW QUERY WITH DEBOUNCE



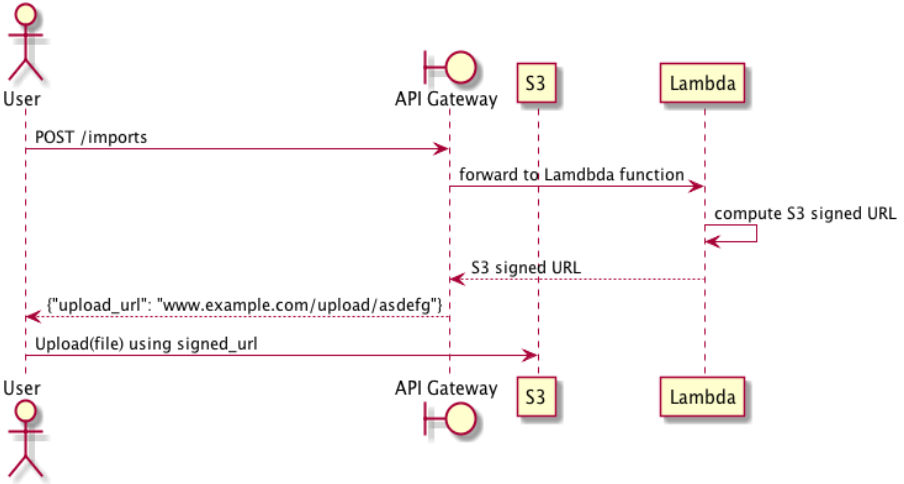
- Tìm kiếm dựa trên dữ liệu nhập vào từ người dùng
- Tối ưu hóa số lượng request bằng cách hạn chế số lần query (chỉ gọi khi người dùng dừng nhập keyword)

SEARCH-VIEW QUERY WITH DEBOUNCE





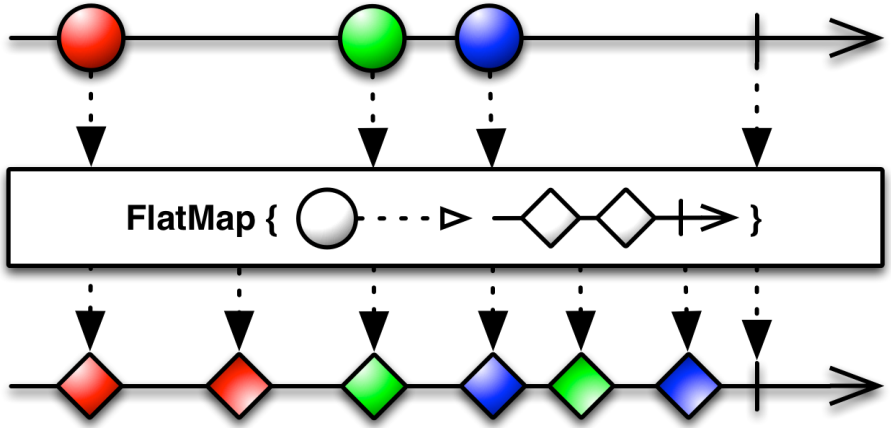
UPLOAD PRESIGN URL



UPLOAD PRESIGN URL

1. Ưu điểm:
   - Dữ liệu đi thẳng tới nơi cần đến
   - Tối ưu hóa hiệu suất cho server
   - Client có thể tiếp tục hoặc bắt đầu lại nếu tiến trình lỗi
2. Khó khăn
   - Chuỗi API
   - Khó kiểm soát luồng khi có nhiều file
   - Thời gian gọi API dài, khó cập nhật tiến trình

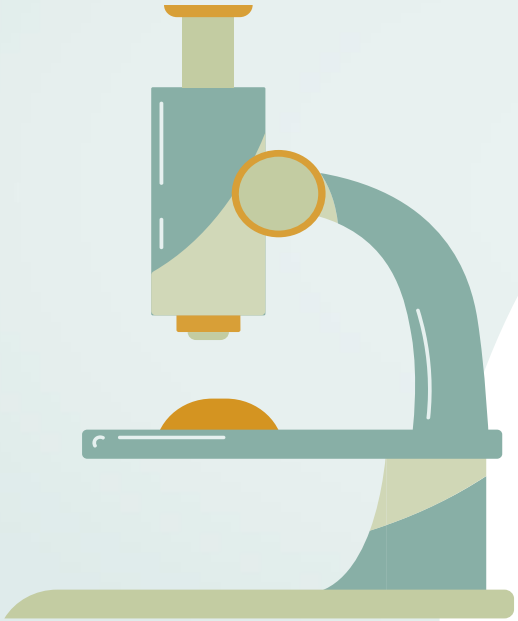
UPLOAD PRESIGN URL

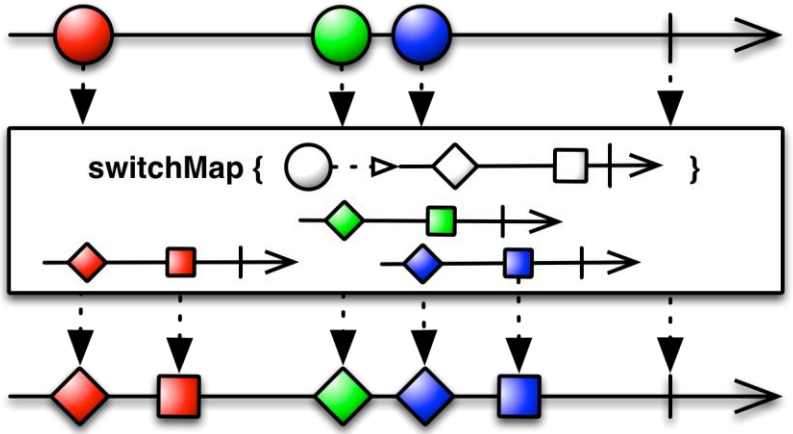


OPTIMIZE LOADING MAIL'S BODY

1. Vấn đề:
   - Phải call 2 api để có dữ liệu đầy đủ
   - Tối ưu hóa số lần gọi lấy mail body, vì không có local-database
   - Tối ưu hóa trải nghiệm người dùng
2. Giải quyết
   - Hiển thị dữ liệu cơ bản ngay khi lấy xong mail-list
   - Cập nhật item ngay khi lấy được mailbody của item đó
   - Chỉ load mailbody của item trên màn hình

OPTIMIZE LOADING MAIL'S BODY





Some

ADVANCED TOPICS

05

You may find interesting

- Hot vs Cold Observables
- Subject, when to use, when not to use?
- Transformations: Map vs FlatMap vs SwitchMap vs ConcatMap
- Filtering: Throttle vs Debounce
- Error handling

THANKS

References:

- ReactiveX: reactivex.io
- Design pattern: refactoring.guru/