FROM RXSWIFT PERSPECTIVE

WELCOME TO RX WORLD (ELEMENTRY)

INTRODUCTION

RXSWIFT & RXCOCOA

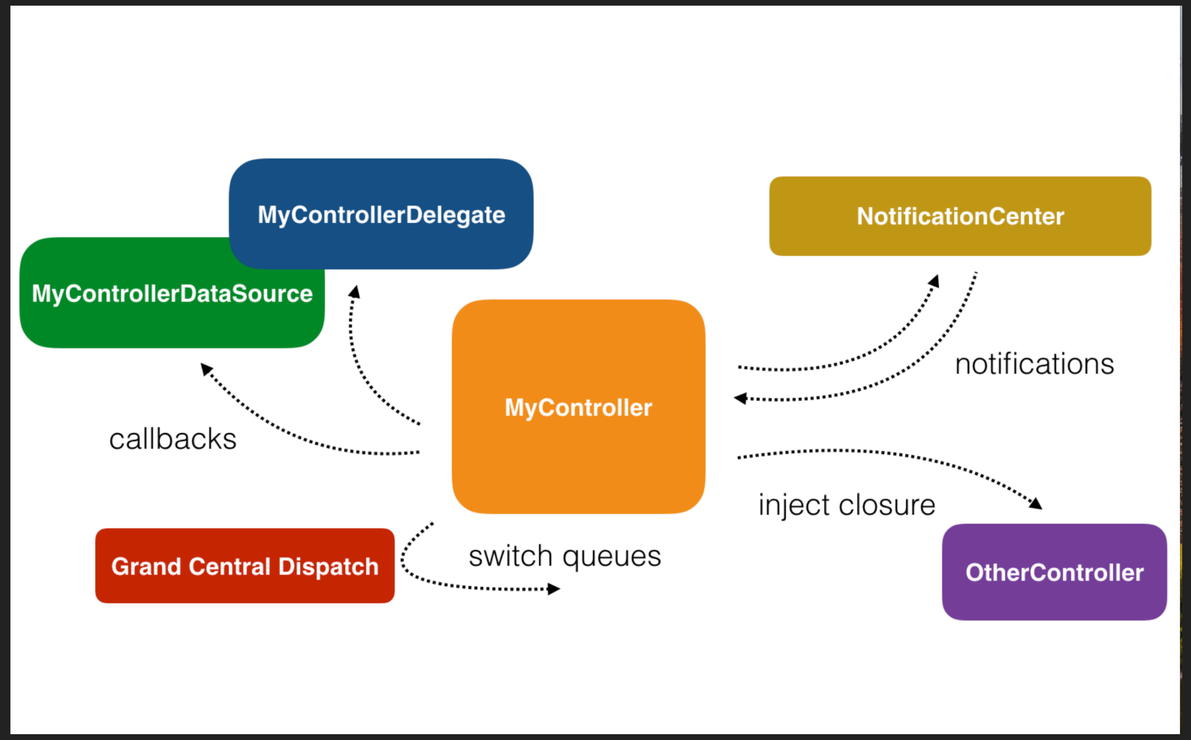
CONTENT

RXSWIFT

RX: REACTIVE EXTENTIONS

HOW TO COMMUNICATE AMONG ASYNC FUNCTIONS

- Closure: 💪[closurehell.swift]
  - closure hell
  - force caller format
- Delegate💪[delegate.swift]
  - interaction
  - continue to return back data
- NotificationCenter
  - so weak connection
  - so, we have to keep notification name and selector same.



Is there any better way to coordinate Async functions?

3 CONCEPTS

- ✨Observable, Subject
- ✨Operators
  - filter
  - transform
  - combine
- Scheduler
  - subscribeOn
  - observeOn

OBSERVABLE

- Key words: Cold, Sequence, Status, Dispose
- Methods: of…, create
- Traits: why need them?

🦐[observable]

There is a trouble…

We can NOT modify Observale after defined.

SO... SUBJECT 🦐

- Types: 4 types(Pub, Beh, Rep, Variable).
  - Variable:
    - only 2 methods: value(change), asObservable(subscribe);
  - Pub, Beh, Rep =>
- ✨Snapshot
- ✨Private subject(∴onNext), expose observable(∴ONLY subscribe).

OPERATORS

- Types:
  - Filter 🍤[filter.swift]: filter, throttle, take(_:schedular) …
  - Transform 🍤[tranform.swift]: map, flatMap
  - Combine Video
- ✨Life cycle

RXCOCOA

STEPING INTO UI . . .

DEAD UI: LABLE

ACTIVE UI: SWITCH BUTTON

ABOUT RXCOCOA

- Active
  - ControlProperty<Bool>: UISwitch+Rx.swift [✨Observable]
  - ControlEvent
- Dead
  - Binder<String?>: UILable+Rx.swift [✨Observer]
  - asDriver, drive

IN PRACTICE

Functional & MVVM

FUNCTIONAL

CONCAT & FLATMAP & RETRYWHEN

CONCAT: HANPPEN IN LINE

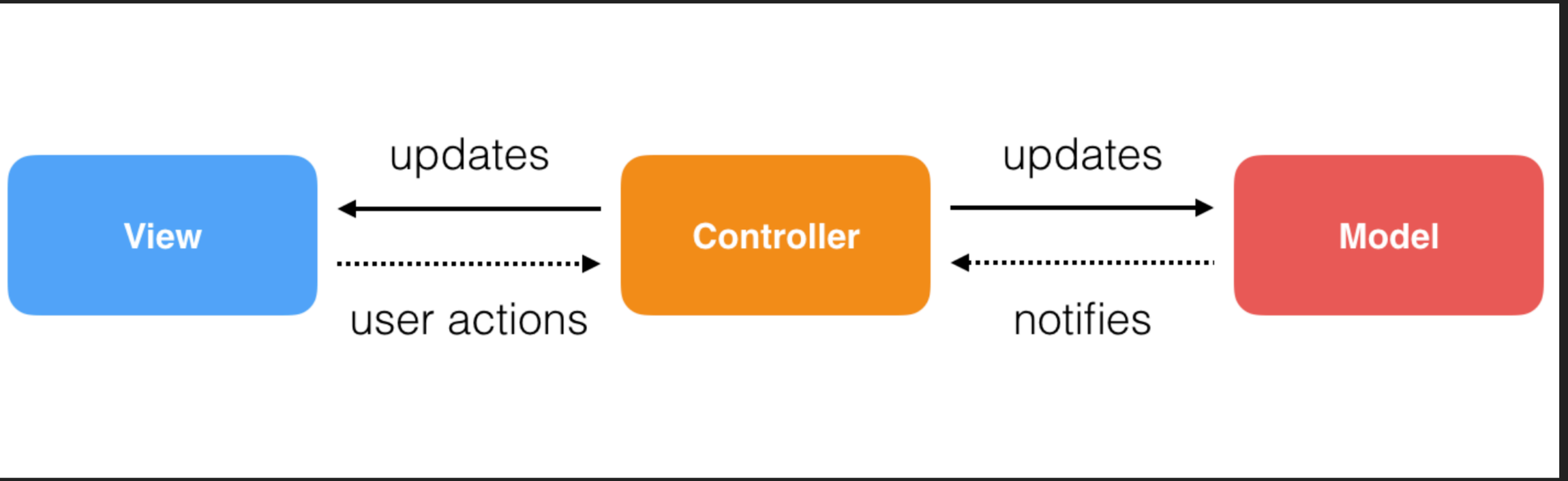
FLATMAP: PRODUCER – CONSUMER

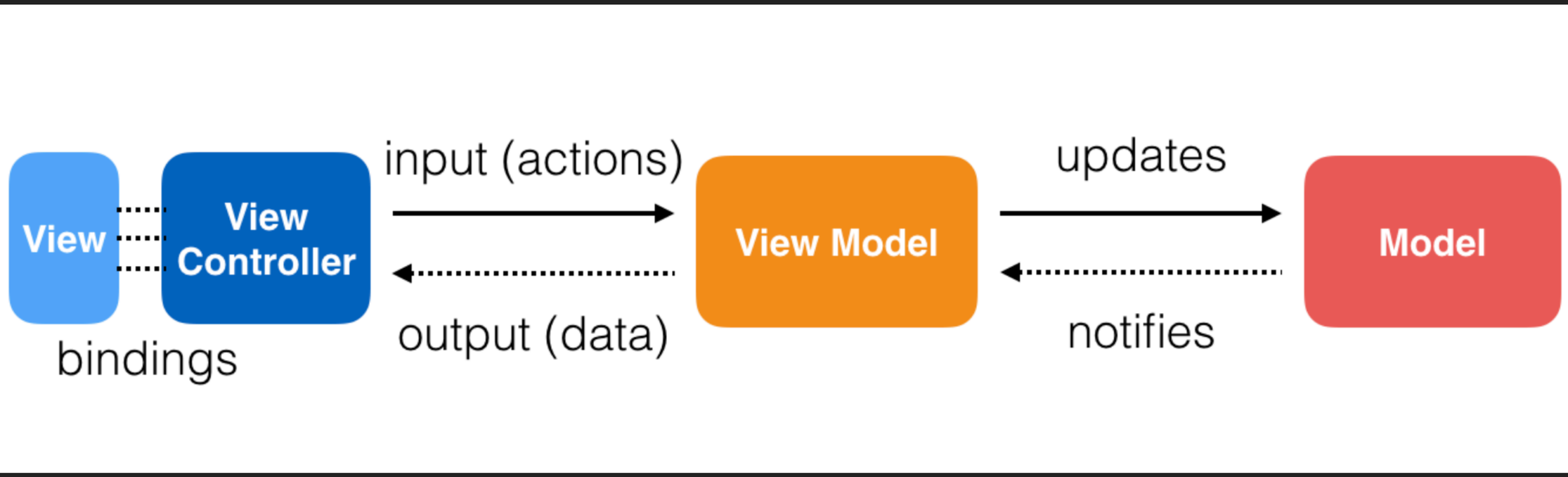
RETRYWHEN: RETRY LOGIC IS BELONGED TO USER

MVVM

MVC = MASSIVE VIEW CONTROLLER

A Joke


View
updates
user actions
Controller
updates
notifies
Model


View
View Controller
bindings
input (actions)
output (data)
View Model
updates
notifies
Model

CONCLUDE

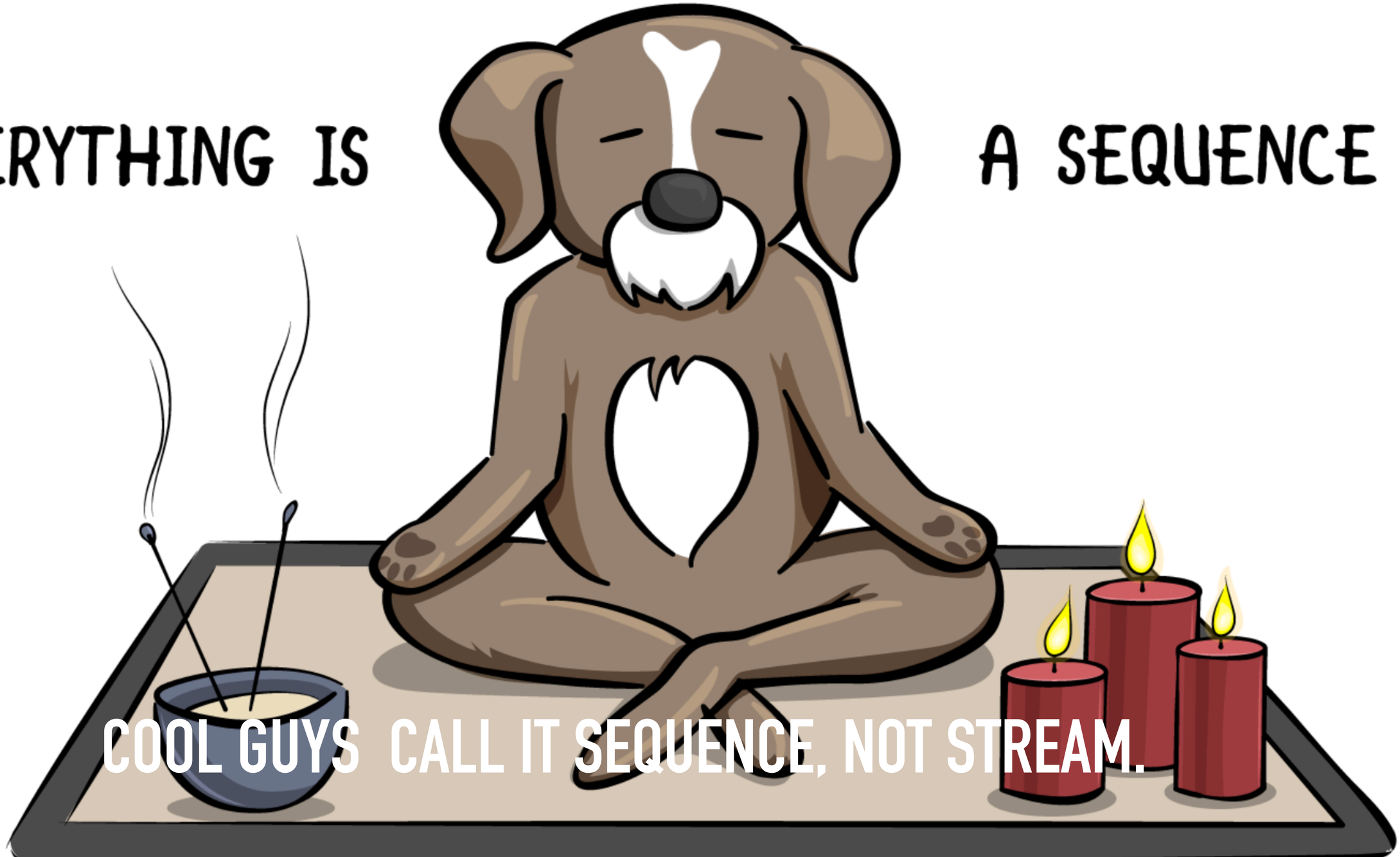
TODO:

1. USE MVVM(TEST)
2. MONAD(F)
3. SCHEDULER – THREAD(DEBUG)
4. CUSTOM DATASOURCE
5. COORDINATE

SUMMERIZE

- Notion about Rx: logo, name, origin
- RxSwift
  - ✨Observable(Subject) ✨Operators ✨Scheduler
- RxCocoa
  - ✨ControlProperty ✨Binder ✨Drive
- IN PRACTICE
  - ✨Functional ✨MVVM

EVERYTHING IS A SEQUENCE



COOL GUYS CALL IT SEQUENCE, NOT STREAM.

REFERENCES

REFERENCES

- Book:
  RxSwif Reactive Programming with Swift
  (second)

- App:
  RxMarbles

CODE SEGMENT

```
// MARK: - observable

//      Observable<String>.create({ (<#AnyObserver<String>#>) -
> Disposable in
//        <#code#>
//      })


// MARK: - control property & drive

let search =
searchCityName.rx.controlEvent(.editingDidEndOnExit).asObservab
le()
      .map { self.searchCityName.text }
      .filter { ($0 ?? "").count > 0 }
      .flatMap { text in
        return ApiController.shared.currentWeather(city:
text ?? "Error")
          .catchErrorJustReturn(ApiController.Weather.empty)
      }
      .asDriver(onErrorJustReturn: ApiController.Weather.empty)

    search.map { "\($0.temperature)° C" }
      .drive(tempLabel.rx.text)
      .disposed(by:bag)


// MARK: - pipeline

__async🏐Signal()
        .concat(__async😴Signal())
        .concat(__async打🐠Signal())
        .subscribe(onNext: { print("=> \($0)") })
        .disposed(by: bag)


// MARK: - produce & consume

__produceFoods()
        .flatMap { food in return __consumeFood(food) }
        .subscribe(onNext: { print($0) })
        .disposed(by: bag)
```

```
// MARK: - closure hell

  func reflectOnReal1() {
    example(of: "reflectOnReal1") {
      __async🏐 {_ in }
      __async😴 {_ in }
      __async打🐠 {_ in }
    }
  }

  func reflectOnReal2() {
    example(of: "reflectOnReal2") {
      __async🏐 {_ in
        __async😴 {_ in
          __async打🐠() { _ in
          }
        }
      }
    }
  }

// MARK: - delegate

protocol DataSourceDelegate {
  func didReceive(_ data: Int)
}

class Producer {
  var delegate: DataSourceDelegate?

  func produce() {
    let data = 1 // produce data
    delegate?.didReceive(data) // feed back data
  }
}

var producer = Producer()

class Consumer: DataSourceDelegate {

  func start() { producer.delegate = self }

  // MARK: - DataSourceDelegate
  func didReceive(_ data: Int) {
    // consume data
  }
}
```