RxKotlin

Reactive Streams for the JVM (and Android)

Problem: React to asynchronous events

- Async reactions to events
- Today done with callbacks
- But what happens when the reactions requires another callback?
  - … and this reaction still another callback
    - … and still another one??

Callback Hell - Pyramid of Doom

```
func1(param, function(err, res) {
  func2(param, function(err, res) {
    func3(param, function(err, res) {
      func4(param, function(err, res) {
        func5(param, function(err, res) {
          func6(param, function(err, res) {
            func7(param, function(err, res) {
              func8(param, function(err, res) {
                func9(param, function(err, res) {
          // Do something...
          });
        });
      });
    });
        });
  });
  });
});
```

Solution proposal: RxJava

- Has its roots in .NET
- Brought to JVM by Netflix: RxJava
- Version 2.0 introduces the concept of Reactive Streams
- Basic elements are Observables and Flowables

Observables – asynchrone Iteratoren

Synchron

Einzelner Wert

```
val value: T
```

Mehrere Werte

```
val values: Iterator<T>
```

Asynchron

Einzelner, asynchoner Wert

```
val futureValue: Future<T>
```

Mehrere, asynchrone Werte

```
val futureValues: Observable<T>
```

Observable

- Variation of the GOF observable pattern
- Actually only an interface:

```
interface Observer<T> {
    fun onNext(t: T)
    fun onError(e: Throwable)
    fun onComplete()
}
```

- There exists also a Single variant – without “onNext”, but with a parameter for “onComplete”.

RxJava – Observable Implemenation

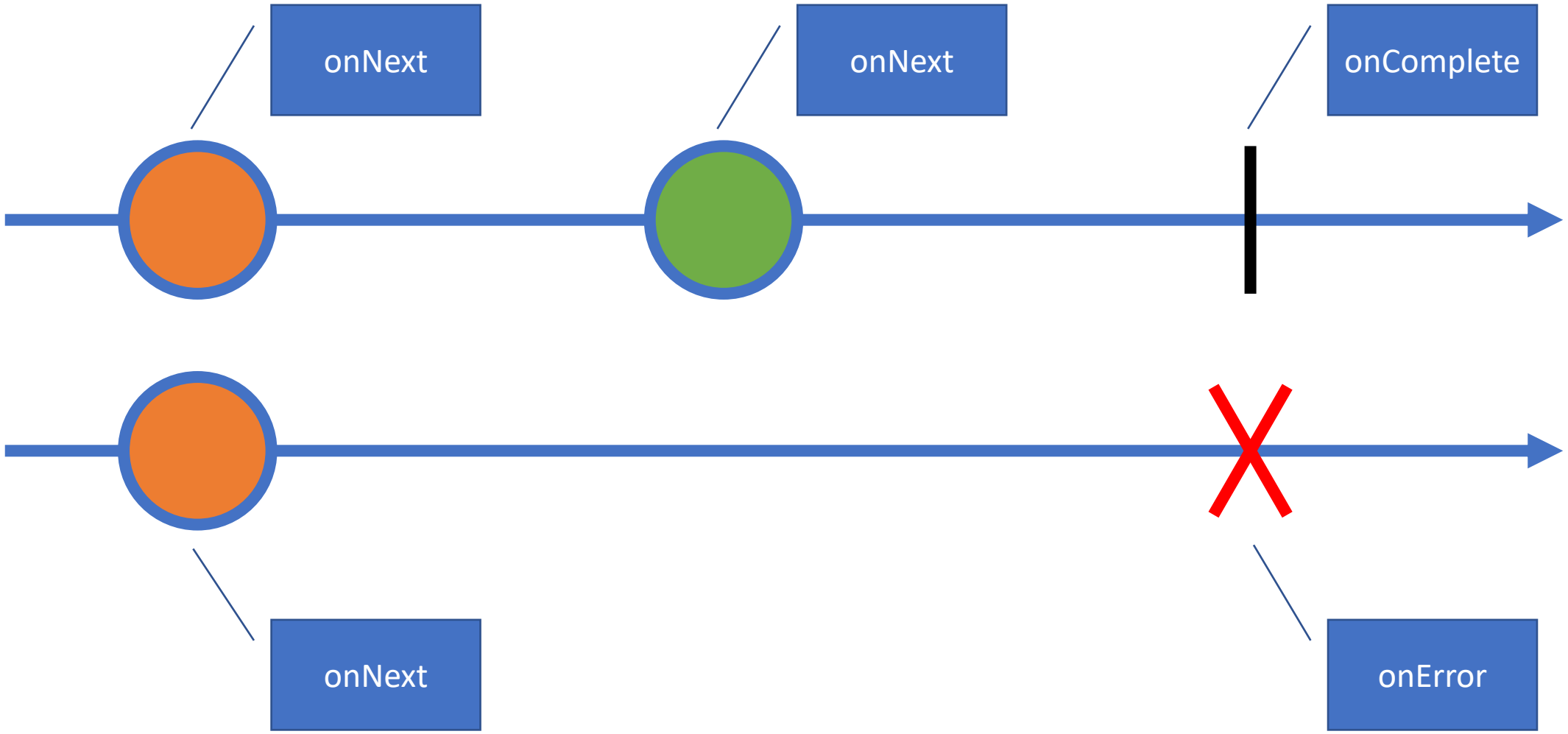
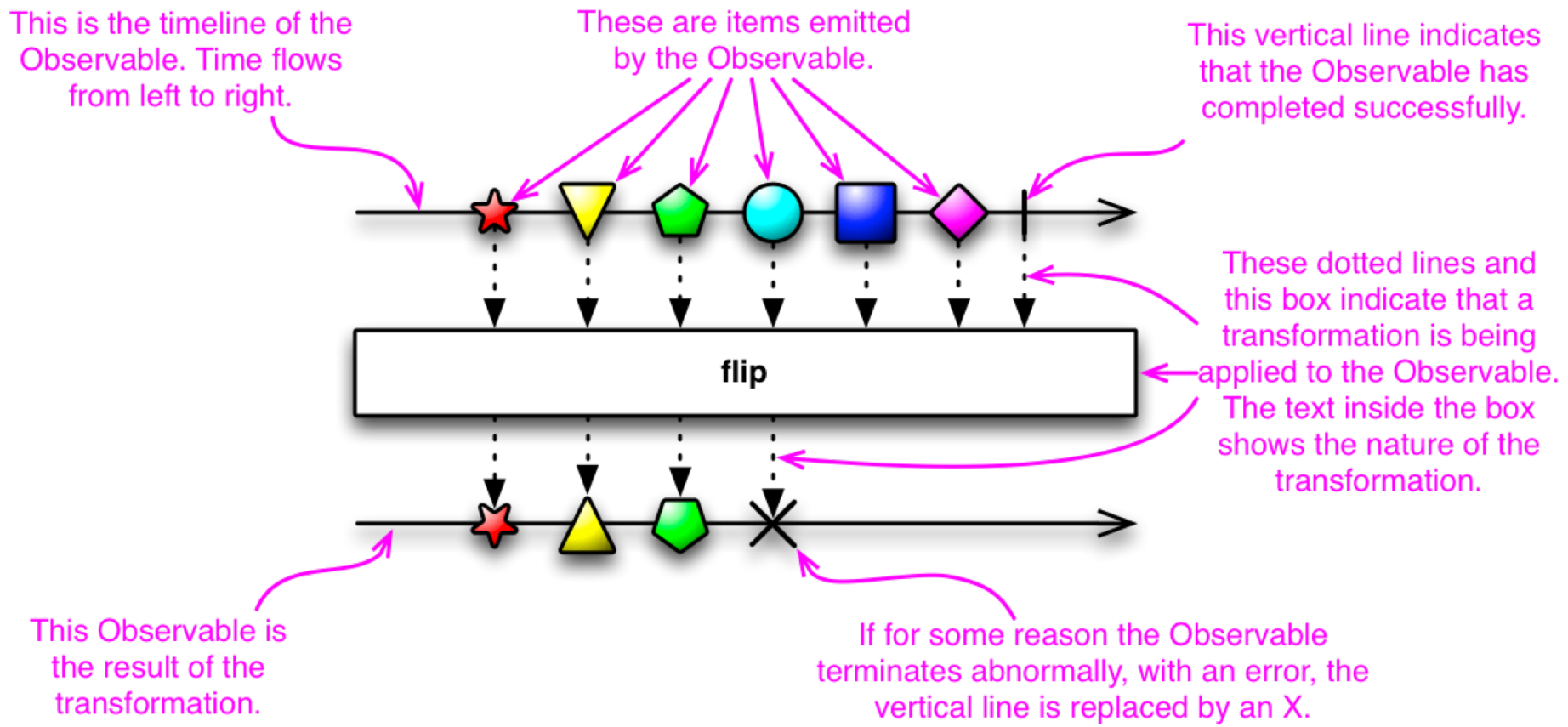


Illustration of events streams – Marble Diagramme



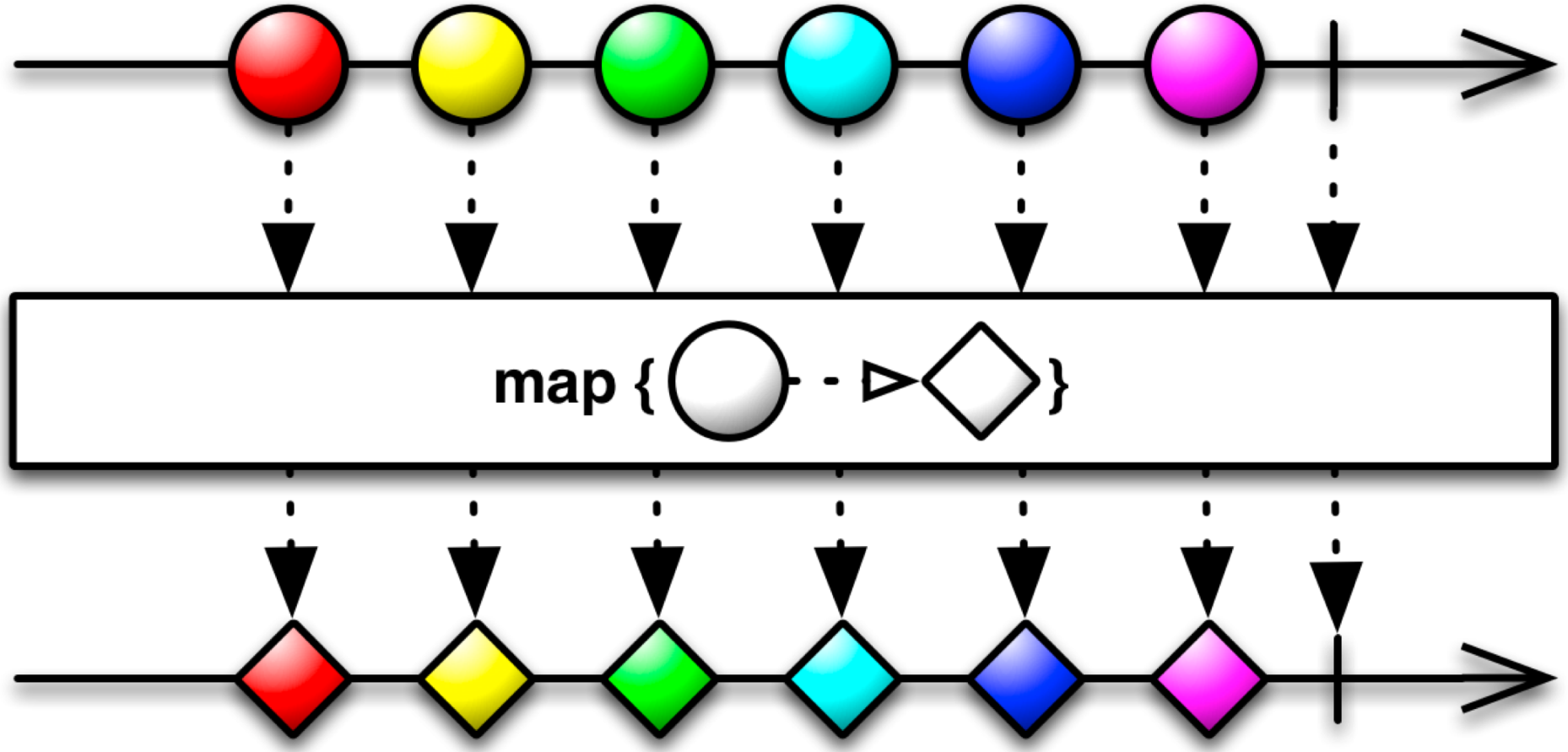
RxJava and functional programming

Higher-Order Functions can be used with many containers:

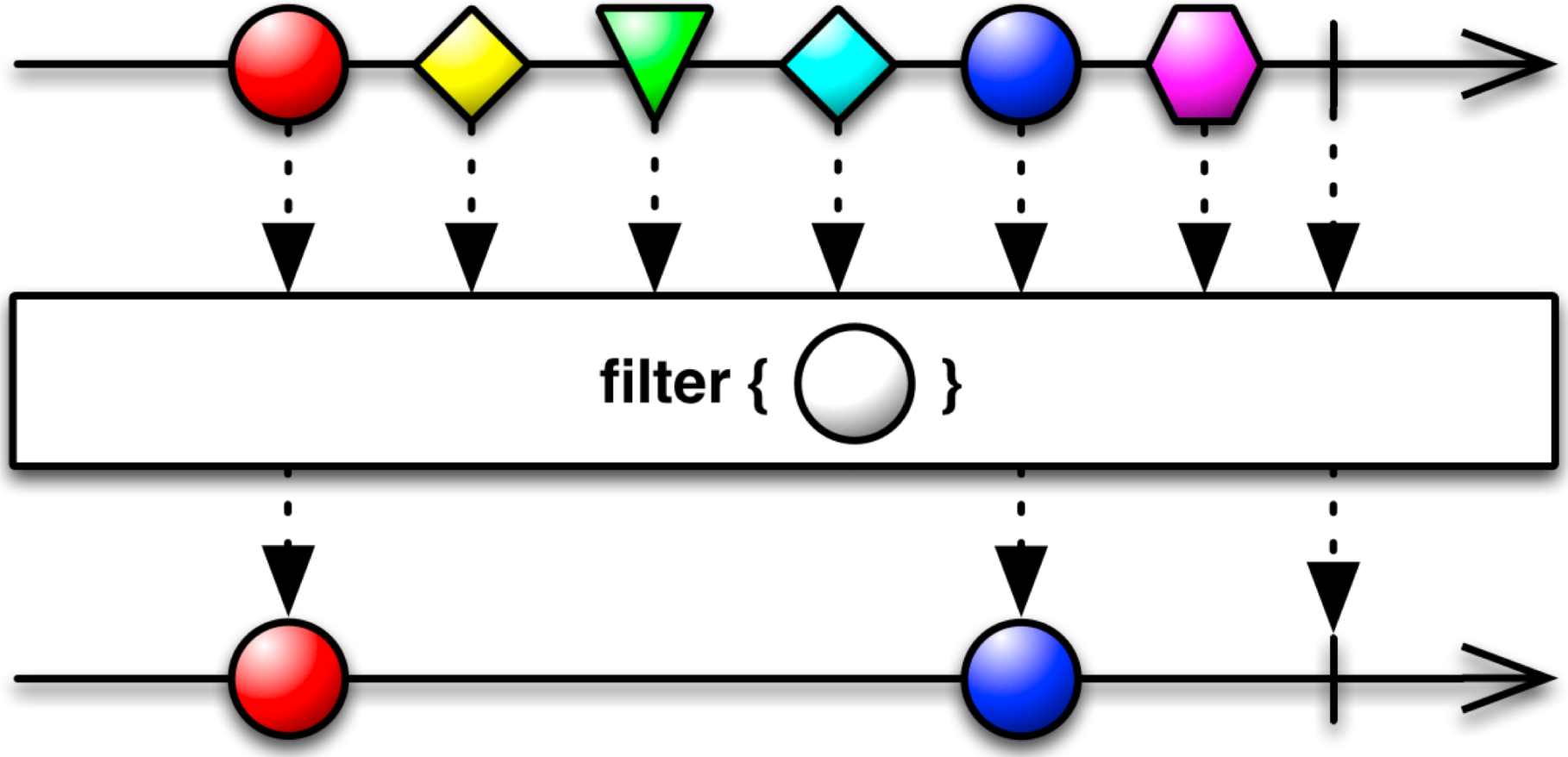
→ Observables can be treated as a container

→ HOF can be used with RxJava!

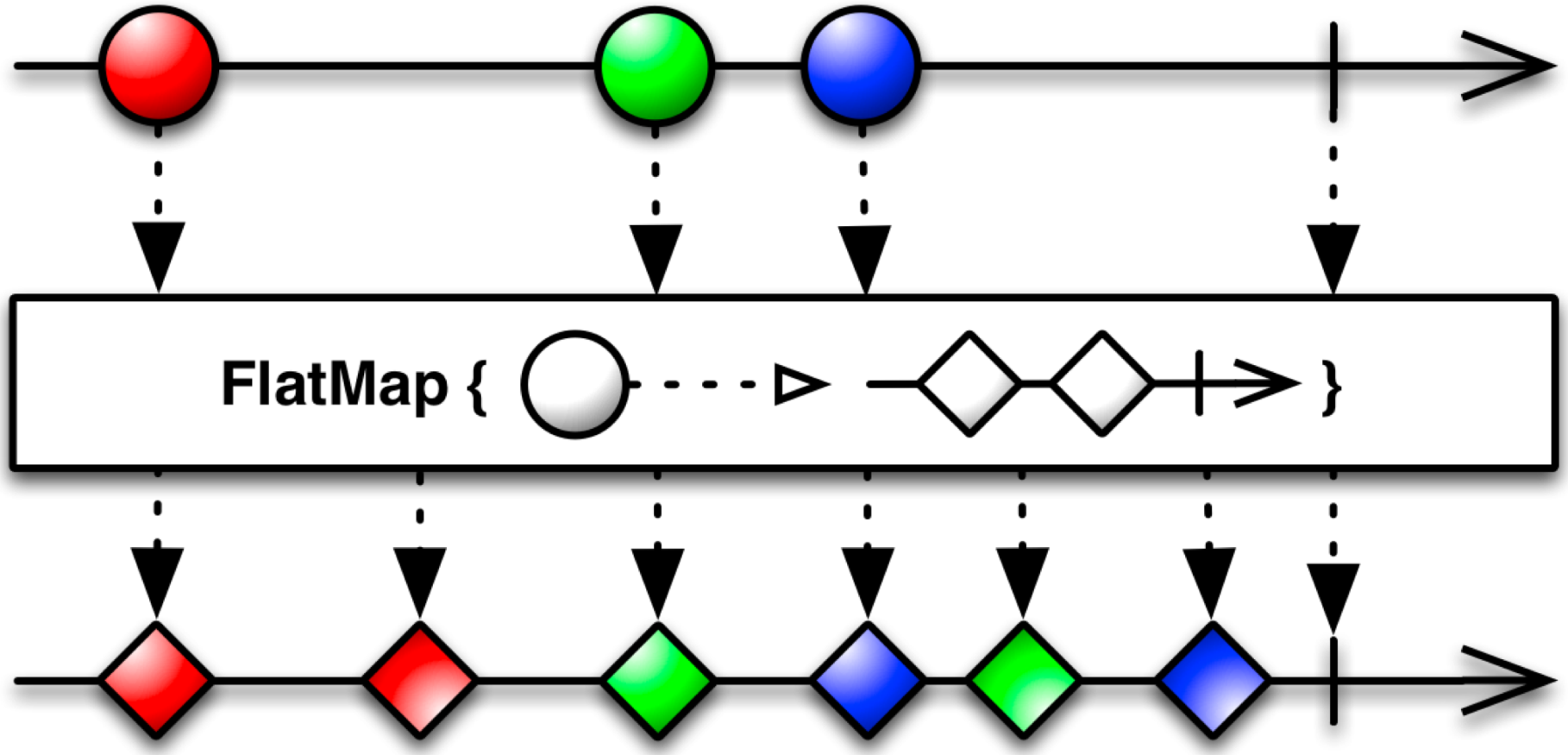
RxJava: Map



RxJava: Filter



RxJava: FlatMap



RxJava: Operators

- Time-orientierte operations
  - Timer
  - Delay
  - Debounce
- Mathematical operations
  - Max, Min, Avg
- Etc. etc.

Example: Username changes

- Model changes in an EditText as an Observable
- Registering the callback when subscribing
- Deregistriering of the callbacks when cancelling

```
val usernameObservable = Observable.create(ObservableOnSubscribe<String> {
emitter ->
    val usernameListener = object : AbstractTextWatcher() {
         override fun afterTextChanged(editable: Editable?) {
             emitter.onNext(editable.toString())
         }
    }
    username.addTextChangedListener(usernameListener)
    emitter.setCancellable {
        username.removeTextChangedListener(usernameListener)
    }
})
```

Exercise

Login validation

Exercise

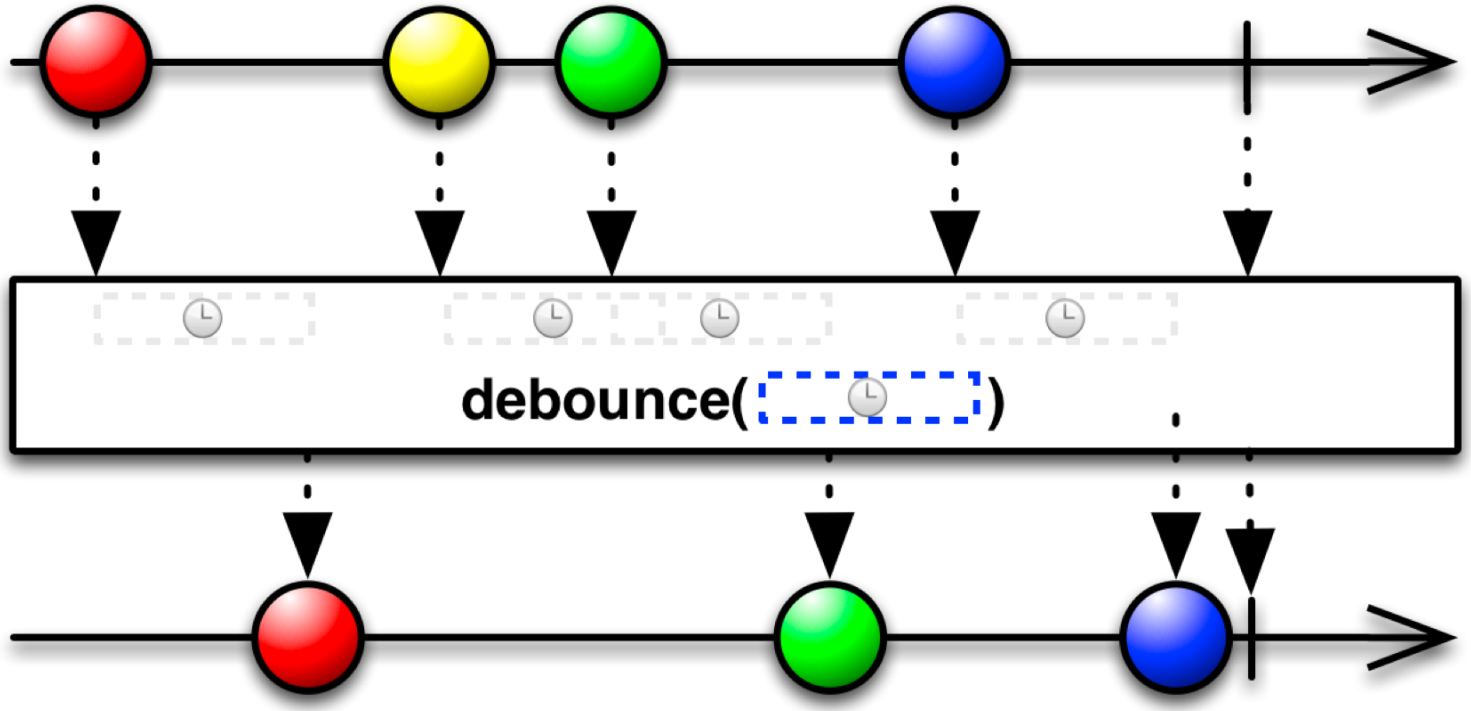
Login validation with a delay

- Checkout branchk ‘*chapter_05_loginValidation*’
- In the class ‘LoginFragment’, delay the Observable ’usernameObservable’
- In the same class, implement the validation of the password as Observable too

Exercise: Login validation

- Input validation is helpful to the user – but it can get on your nerves!
- Only start validation when the user takes a break.
- How?

→ Debouncer



Exercise: Login validation

```
FATAL EXCEPTION: RxComputationThreadPool-
Process: ch.zuehlke.sbb.reddit, PID:
    io.reactivex.exceptions.OnErrorNotImplementedException: Only the original thread that
    created a view hierarchy can touch its views.
    at
    io.reactivex.internal.functions.Functions$OnErrorMissingConsumer.accept(Functions.java:704)
```

- Debounce runs on a background thread. So is the observation → Exception
- Solution: instruct RxJava to observe on the main thread

```
.observeOn(AndroidSchedulers.mainThread())
```

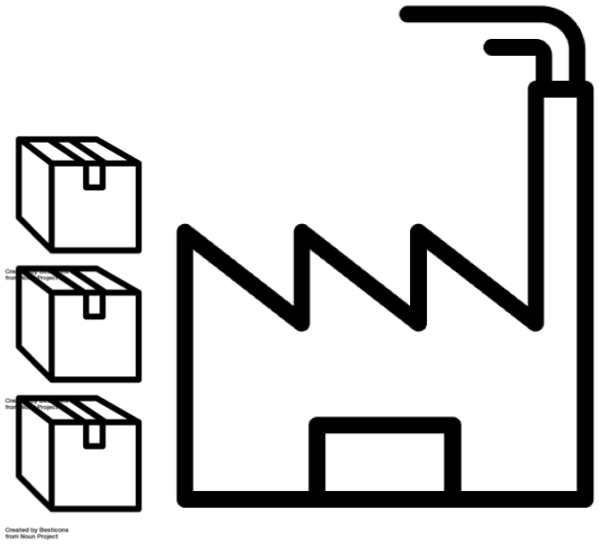
Flowable

What happens when a publisher generates events quicker than the subscriber is able to process them?

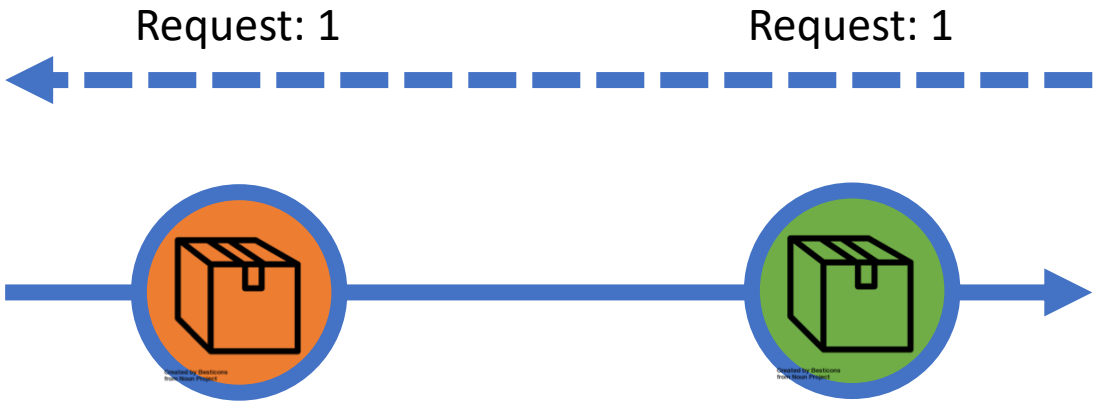
- Observables: Push
  The source decides when to generate a new event
- Java Streams: Pull
  The sink decides when to generate a new event

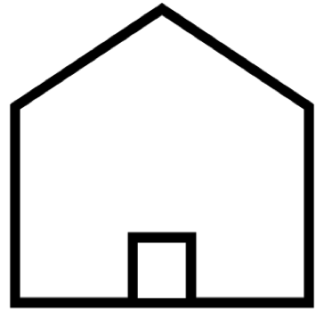
~~You have to choose!~~

Flowable - Pull



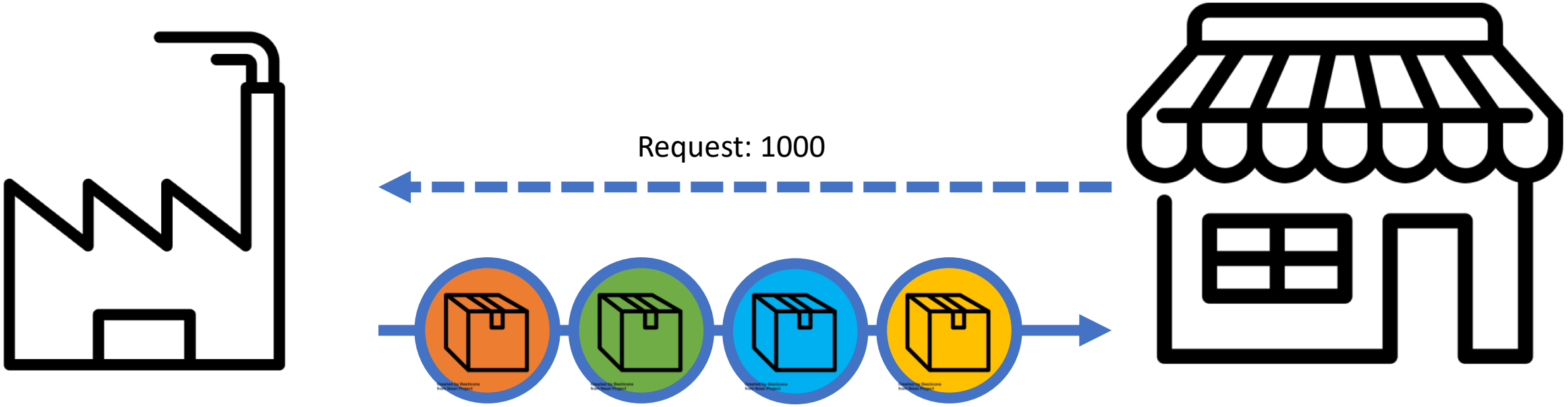
Created by Symbolon
from Noun Project





Created by Pyetro Rapp
from Noun Project

Flowable - Push

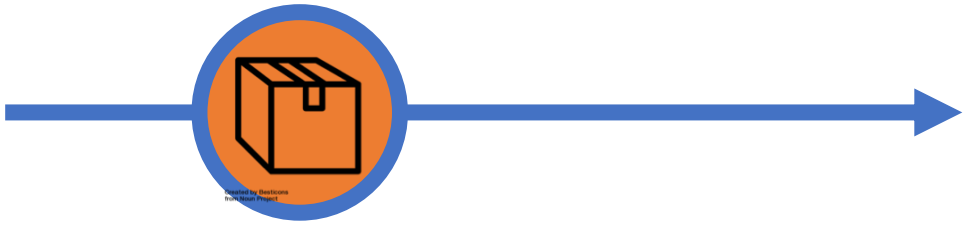


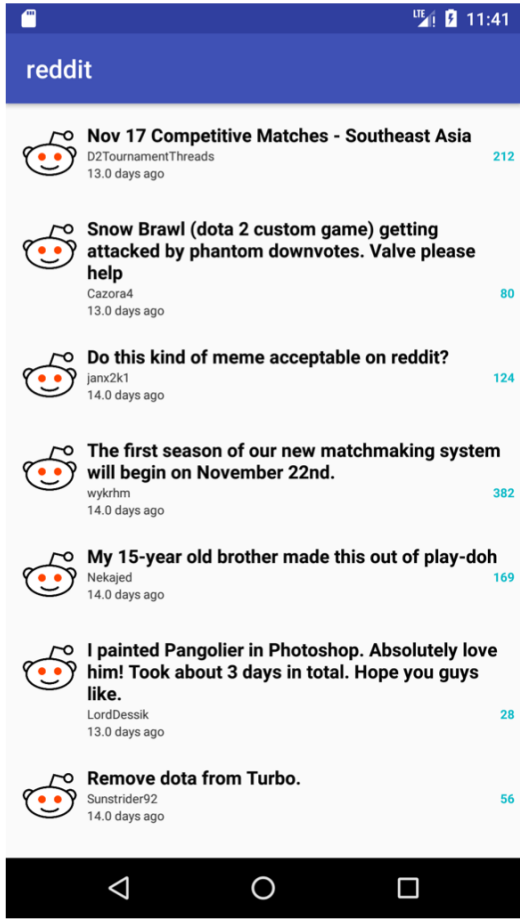
Created by Symbolon
from Noun Project

Created by anbileru adaleru
from Noun Project

Beispiel: Infinite Scroll Anbindung

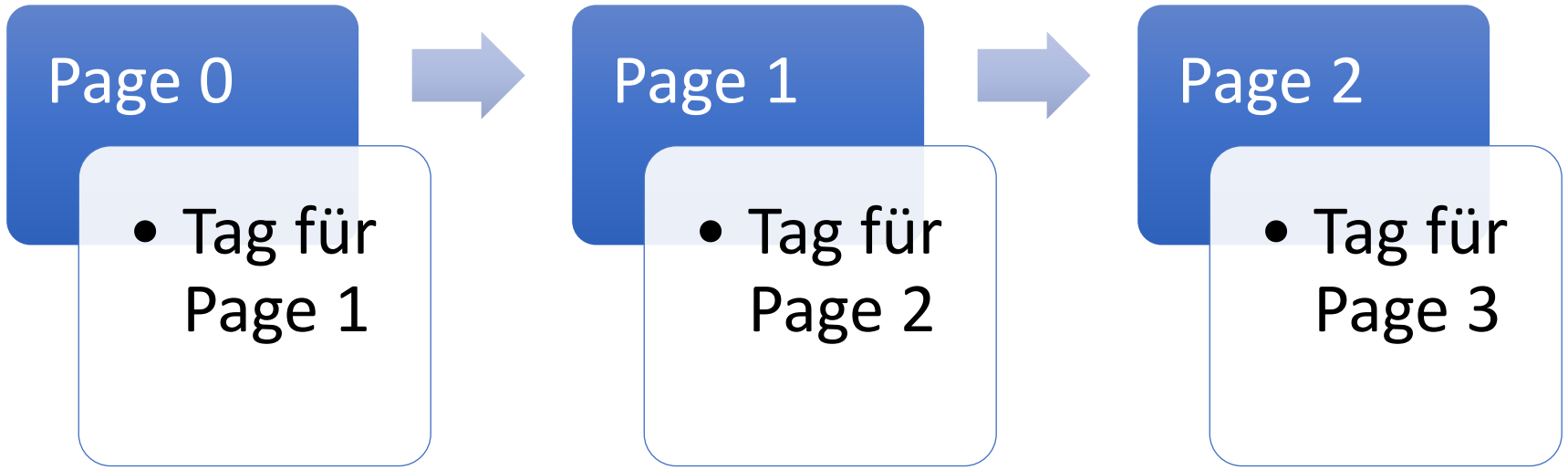
Request: 1





RedditAPI Pagination

- Reddit API uses pagination
- Entries are not numbered
- Response contains a tag for the next page



Example: Infinite Scroll API Side

```
override val news: Flowable<List<RedditNewsData>> = Flowable.generate(
        fun(): String { return "" },
        fun(currentTag: String, emitter: Emitter<List<RedditNewsData>>): String {
            try {
                val (newsList, nextTag) = queryRedditAPI(redditAPI, currentTag)
                emitter.onNext(newsList.mapNotNull { it })
                return nextTag
            } catch (error: Exception) {
                emitter.onError(error);
                return ""
            }
        }
)
```

Example: Infinite Scroll UI Side

```
abstract class PageSubscriber : ResourceSubscriber<List<RedditNewsData>>()
{
    override fun onStart() {
        nextPage()
    }

    fun nextPage() {
        logD("Request next page")
        request(1) // <<- Generate a single request
    }
}
```

Example: Infinite Scroll UI Side

```
fun createRedditSubscriber() = object : PageSubscriber() {

    override fun onNext(news: List<RedditNewsData>) =
            if (news.isEmpty())
                processEmptyTasks()
            else
                mOverviewView.addRedditNews(news)

    override fun onError(t: Throwable?) {
            mOverviewView.showRedditNewsLoadingError()
    }

}
```

Example: NextPage

```
currentSubscription = createRedditSubscriber()
mRedditRepository.news
        .subscribeOn(ioScheduler)
        .observeOn(mainScheduler, false, 1)
        .subscribeWith(currentSubscription)
```

```
currentSubscription.nextPage()
```

Branch '*chapter_06_infiniteScroll*' contains this example

Quellen

- http://reactivex.io/documentation/operators.html
- https://github.com/ReactiveX/RxJava/wiki/Backpressure
- http://reactivex.io/RxJava/javadoc/rx/Observable
- http://www.reactive-streams.org/
- http://reactivex.io/documentation/operators/subscribeon.html
- http://reactivex.io/documentation/operators/observeon.html