RxKotlin

Reactive Streams für die JVM (und Android)

Problem: Auf asynchrone Events reagieren

- Async auf Events reagieren
- Heute mit Callbacks
- Was aber wenn die Reaktion einen weiteren Callback benötigt?
  - … und diese Reaktion noch einen Callback
    - … und noch einen??

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...
          });
        });
      });
    });
      });
  });
  });
});
```

Lösungsvorschlag: RxJava

- Ursprünge in .NET
- Von Netflix auf die JVM gebracht: RxJava
- Mit Version 2.0 um das Konzept von Reactive Streams erweitert
- Grundelemente sind Observables und 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 vom GOF Observable Pattern
- Nur ein Interface:

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

- Existiert auch als Single Variante – ohne “onNext”, dafür mit Parameter beim “onComplete”.

RxJava – Observable Implemenation

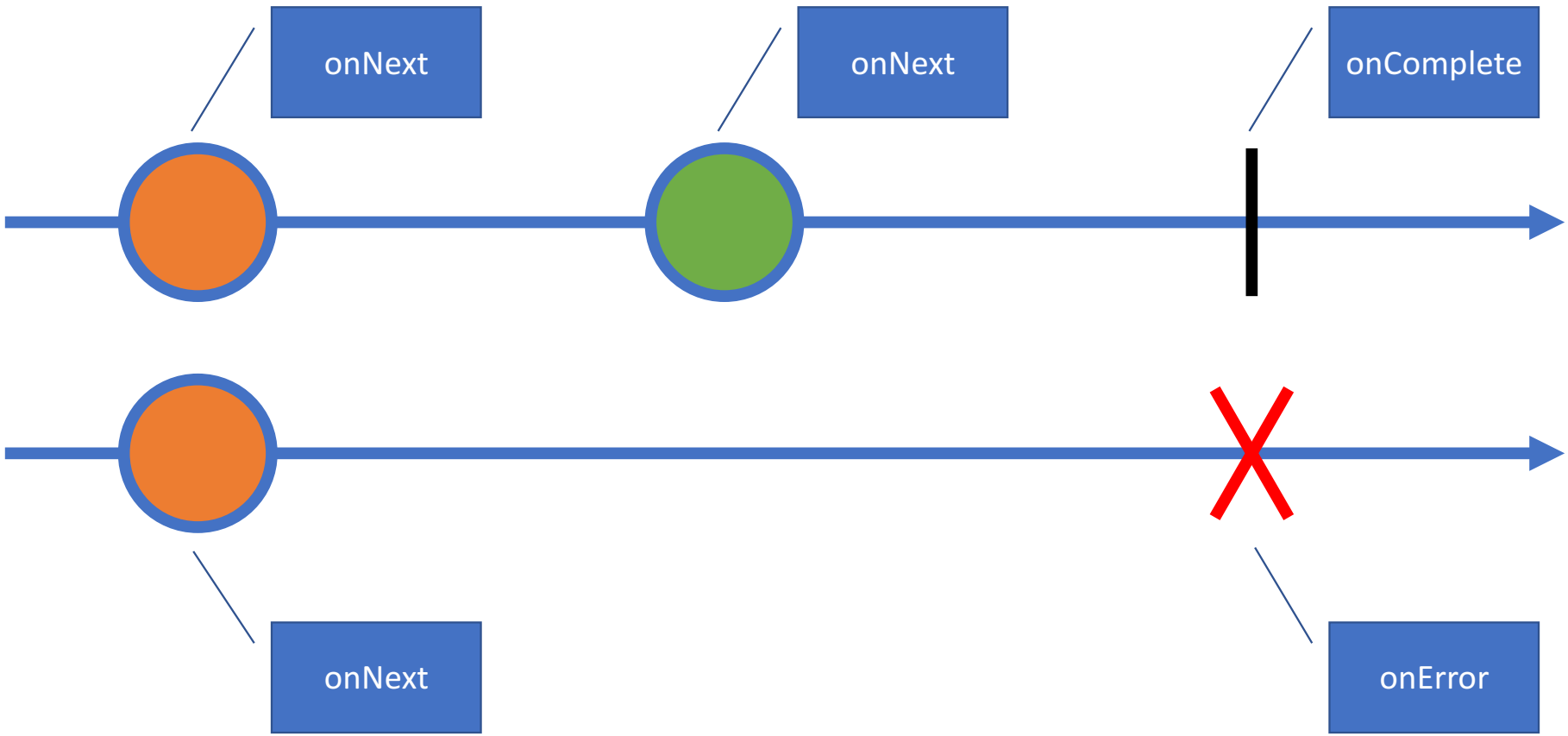
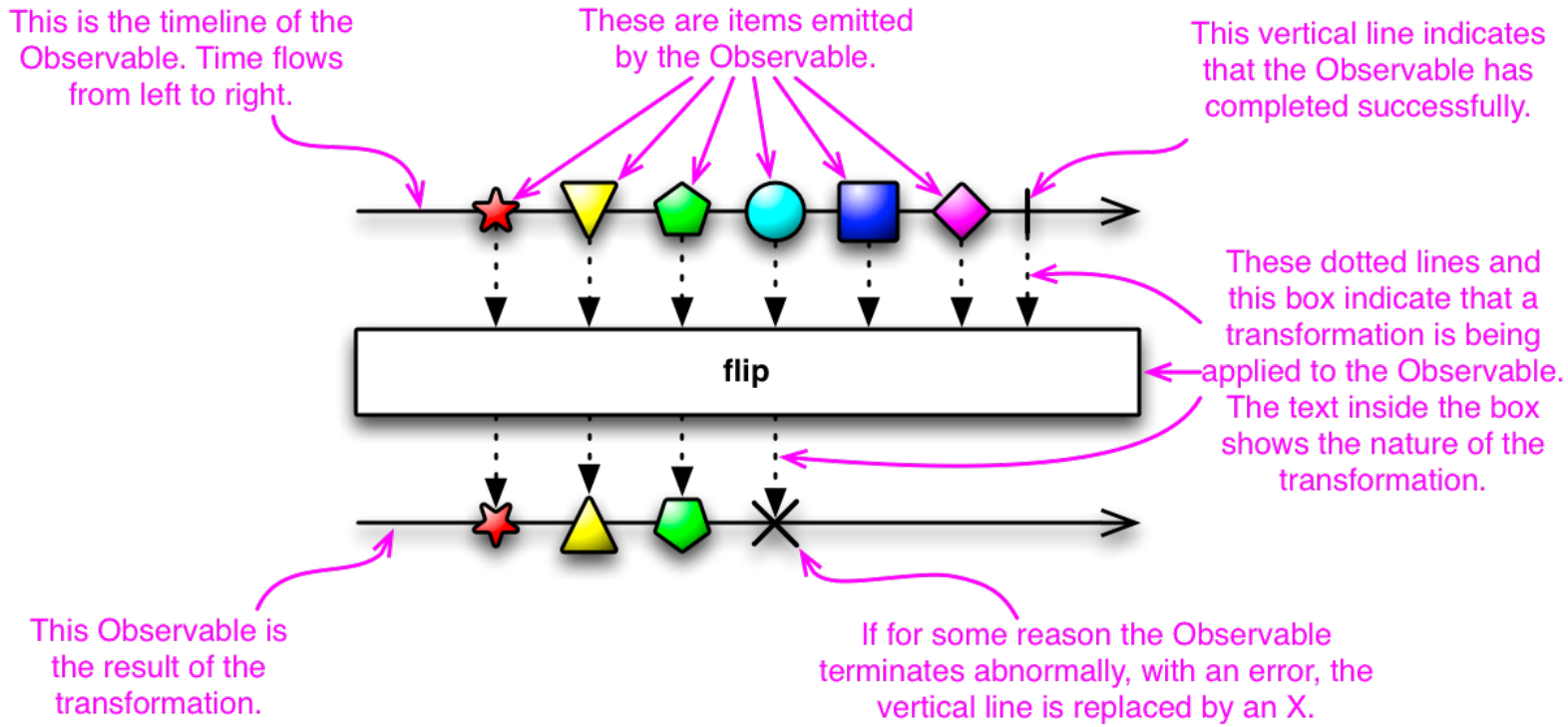


Illustration von Events Streams – Marble Diagramme



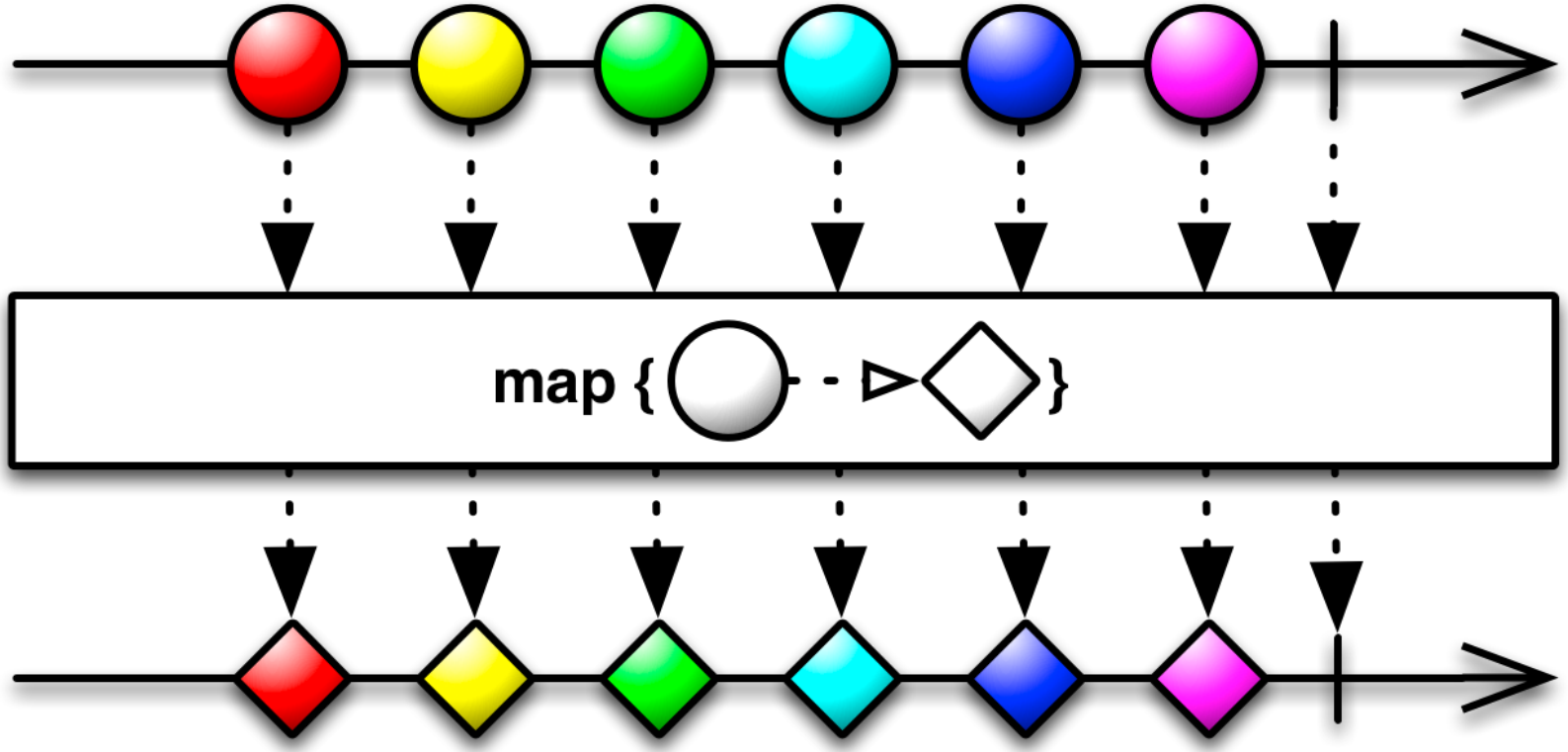
RxJava und Funktionale Programmierung

Higher-Order Functions können auf verschiedenste Container angewendet werden

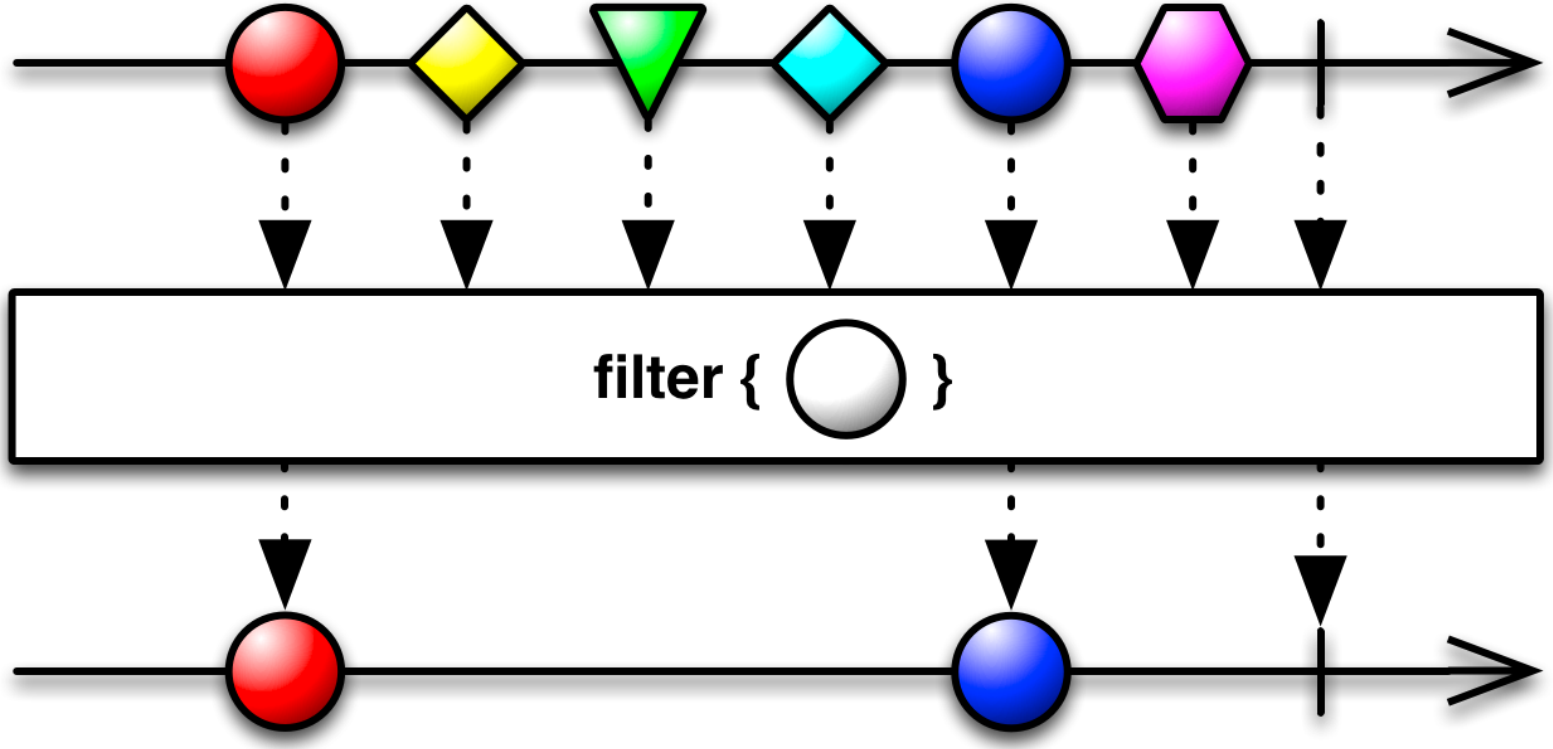
→ Observables können als Container angesehen werden

→ HOF können mit RxJava verwendet werden!

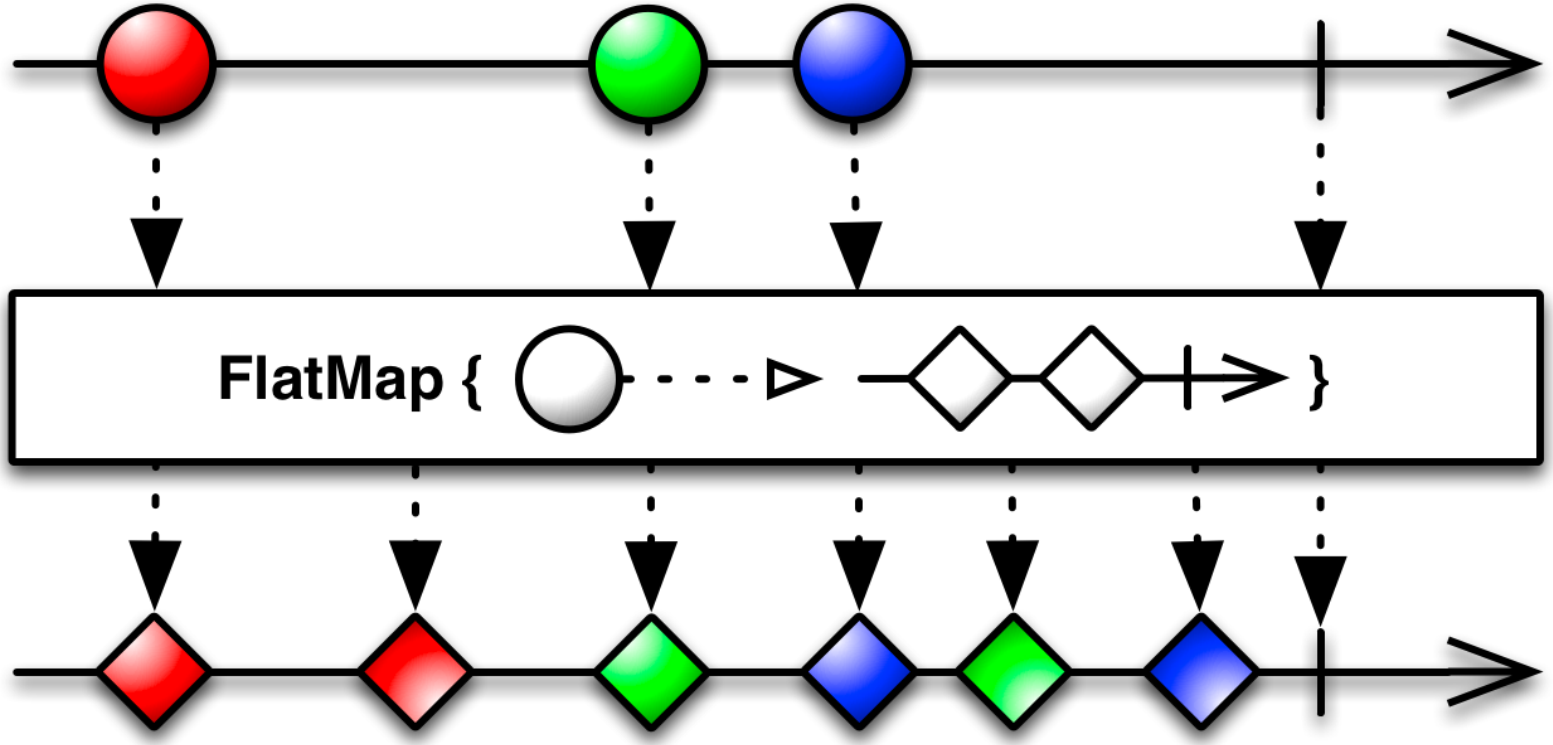
RxJava: Map



RxJava: Filter



RxJava: FlatMap



RxJava: Operationen

- Zeit-orientierte Operationen
  - Timer
  - Delay
  - Debounce
- Mathematische Operationen
  - Max, Min, Avg
- Etc. etc.

Beispiel: Username Changes

- Änderungen eines EditText-Elements als Observable modellieren
- Registrieren des Callbacks bei der Subscription
- Deregistrieren des Callbacks bei der Cancellation

```
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)
    }
})
```

Aufgabe

Login Validierung

Aufgabe

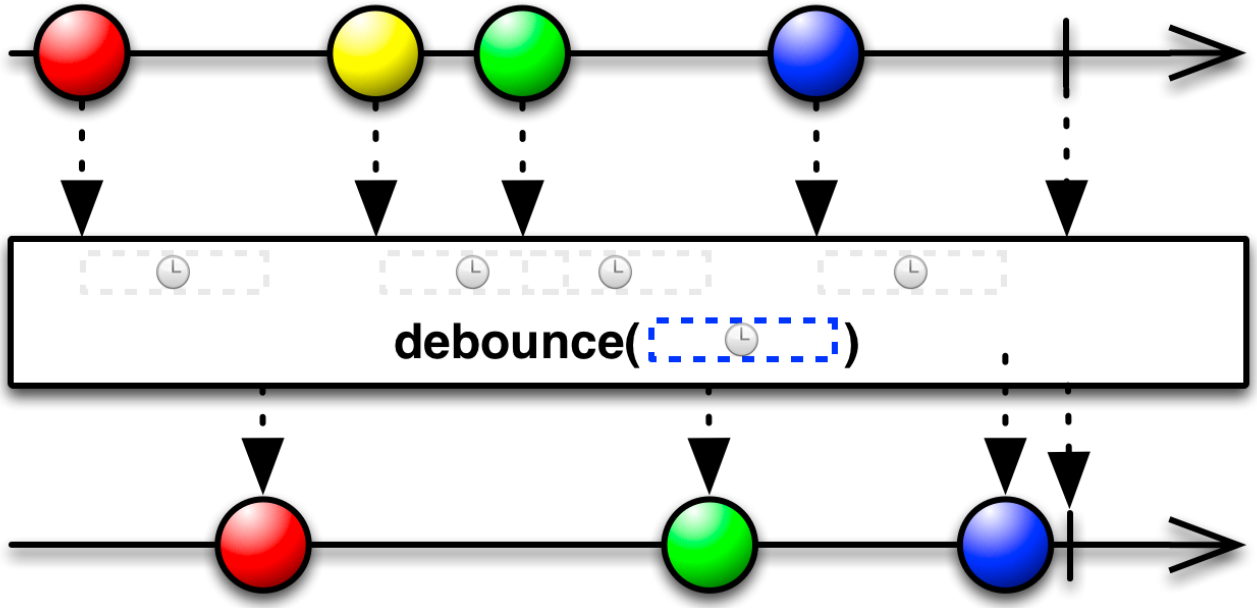
Login Validierung mit Verzögerung auswerten

- Branch ‘`step_07_debounce_exercise`’ auschecken
- In der Klasse ‘LoginFragment’ das Observable ’usernameObservable’ verzögern
- In derselben Klasse die Validierung des Passworts auch als Observable implementieren

Aufgabe: Login Validierung

- Validierung der Eingabe ist hilfreich für den User – kann aber auch nerven!
- Validierung erst starten wenn der User eine Pause macht – Wie?

→ Debouncer



Aufgabe: Login Validierung

```
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 wird auf einem Background Thread ausgeführt
  Folglich auch das Observing → Exception
- Lösung: RxJava anweisen, Observationen auf dem Main Thread durchzuführen

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

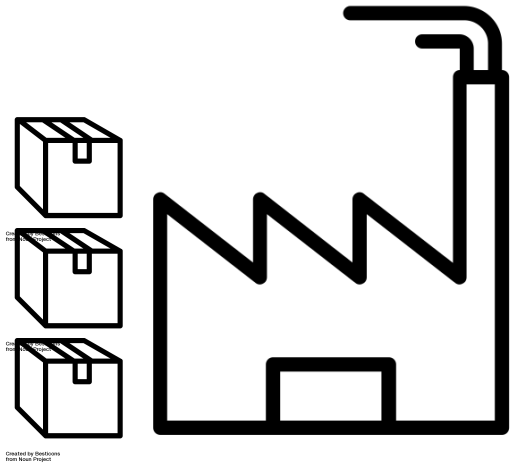
Flowable

Was passiert, wenn ein Subscriber Events nicht so schnell verarbeiten kann wie sie generiert werden?

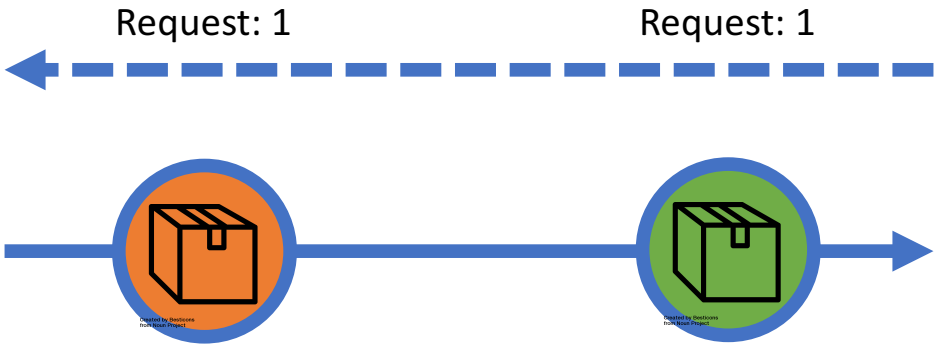
- Observables: Push
  Die Quelle entscheidet, wann ein neuer Event generiert wird
- Java Streams: Pull
  Die Senke entscheidet, wann ein neues Element generiert wird

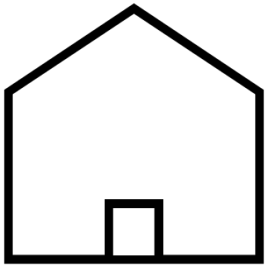
~~Man muss sich entscheiden!~~

Flowable - Pull



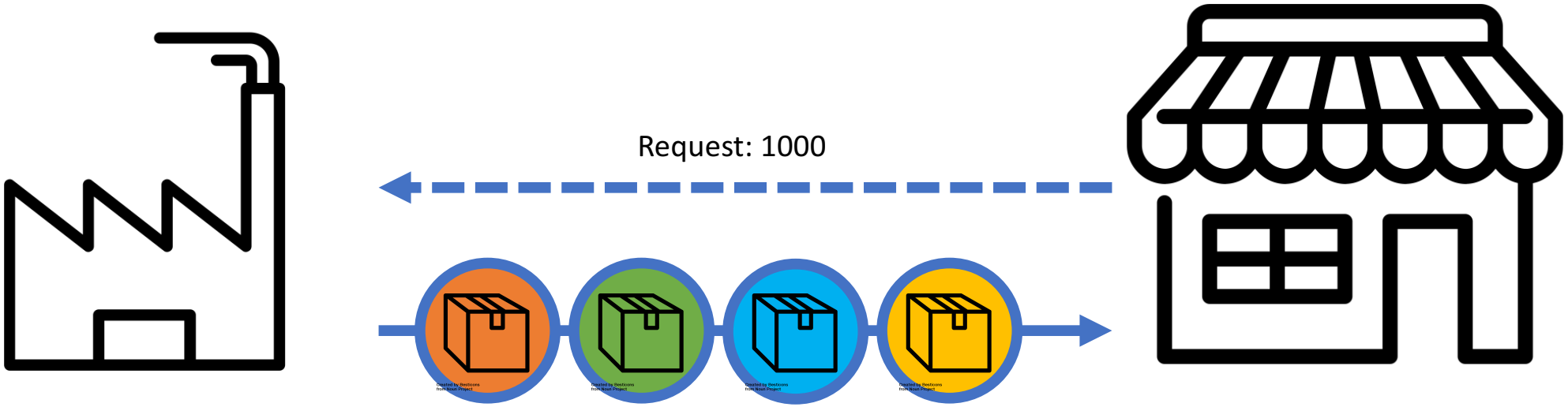
Created by Symbolon
from Noun Project





Created by Pyetro Rapp
from Noun Project

Flowable - Push

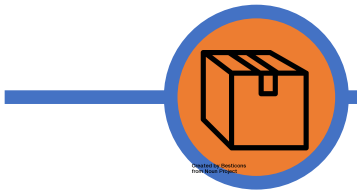


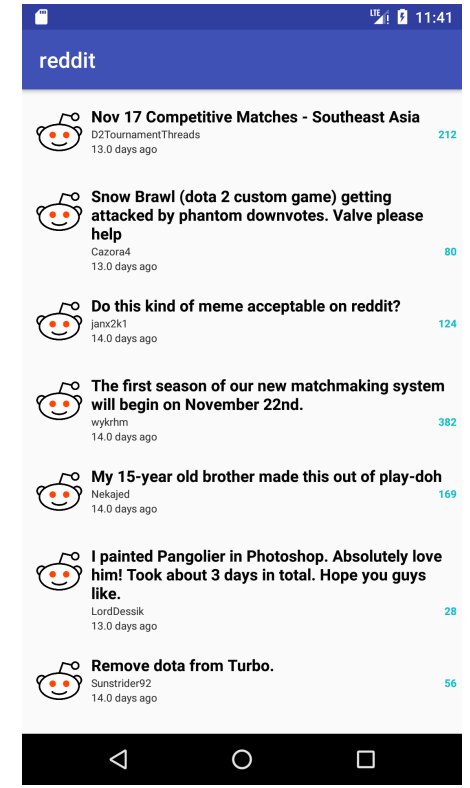
Created by Symbolon
from Noun Project

Created by anbileru adaleru
from Noun Project

Beispiel: Infinite Scroll Anbindung

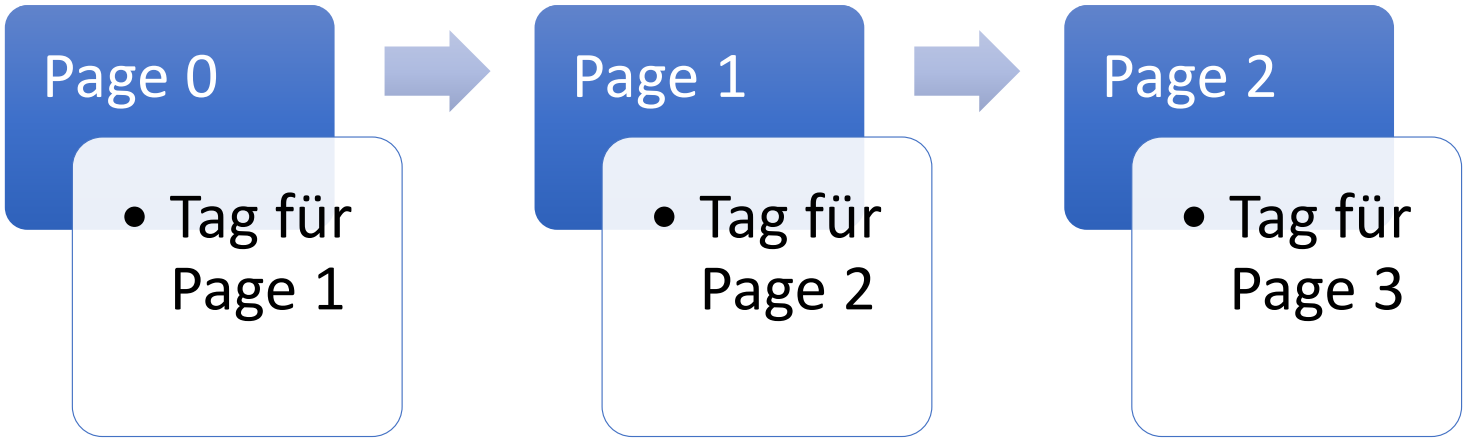
Request: 1





RedditAPI Pagination

- Reddit API nutzt Pagination
- Einträge sind nicht nummeriert
- Antwort enthält ein Tag für die nächste Page



Beispiel: 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 ""
            }
        }
)
```

Beispiel: 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
    }
}
```

Beispiel: 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()
    }

}
```

Beispiel: NextPage

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

```
currentSubscription.nextPage()
```

Branch ‘step_07_infinite_scroll’ enthält dieses Beispiel

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