Wprowadzenie do RxJava

Mikołaj Fejzer

Agenda

- Programowanie reaktywne
  - historia, założenia
  - związek z programowaniem funkcyjnym
  - Iterator vs Observer
- RxJava
  - backpressure (przeciwciśnienie ?)
  - testowanie, debugowanie
- Źródła, referencje

Historia

Próba odpowiedzi na:

- zbyt niski poziom abstrakcji (callback hell)
- brak kontraktów
- brak naturalnego modelu komunikacji
- pojawienie sie koncepcji wielu rdzeni/procesorów

Założenia

- The Reactive Manifesto
  - Responsive
  - Resilient
  - Elastic
  - Message Driven
- Odwrócenie kontroli

Programowanie funkcyjne?

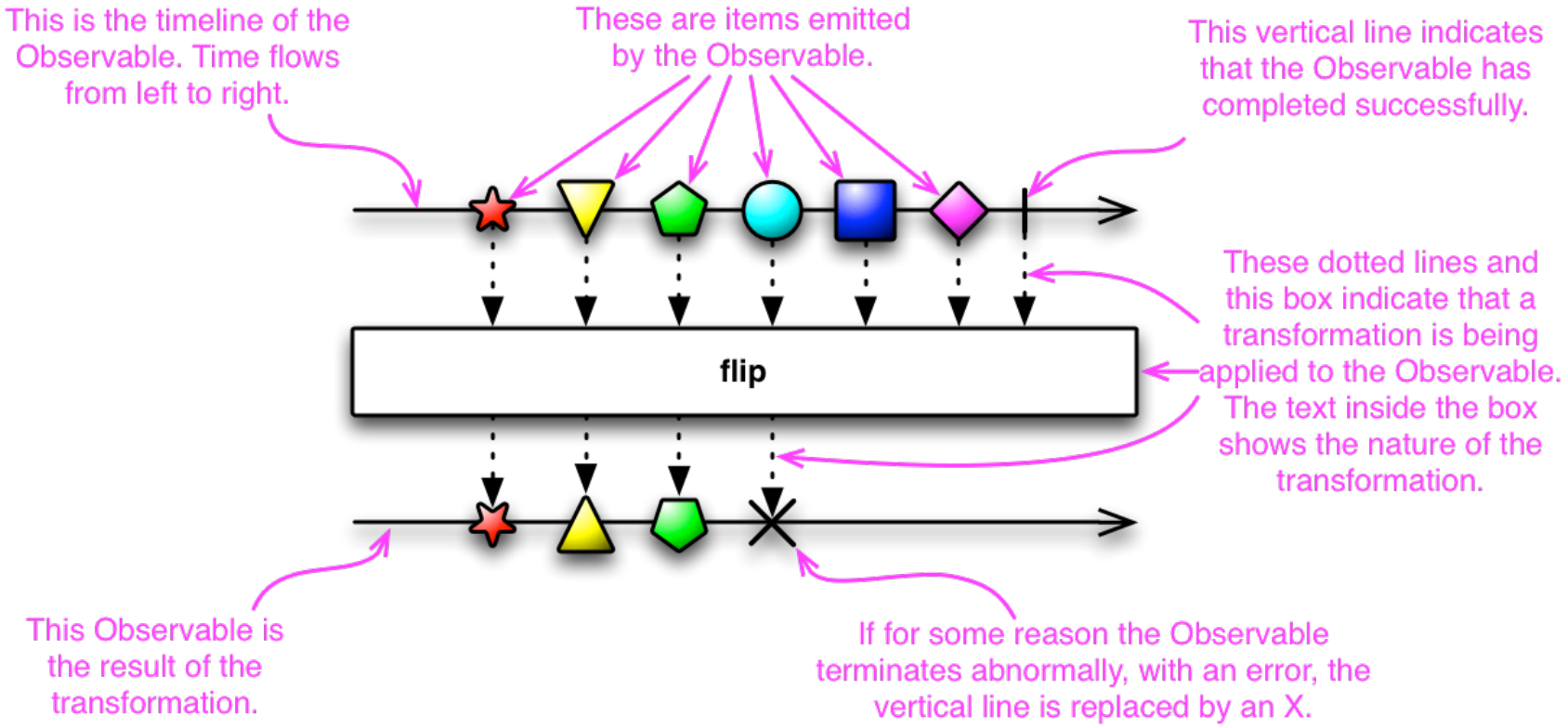
- Obsevable
  - map, flatMap
  - reduce, groupBy
  - filter, take, skip, last
  - zip, merge
- Kompozytowalność strumieni
- Modelowanie zachowania
- Niezmienność zdarzen

Iterator vs Observer

```
Iterator<E>{
boolean hasNext()
E next()
}
```

```
Observer<T>{
void onCompleted()
void onError
(Throwable e)
void onNext(T t)
}
```

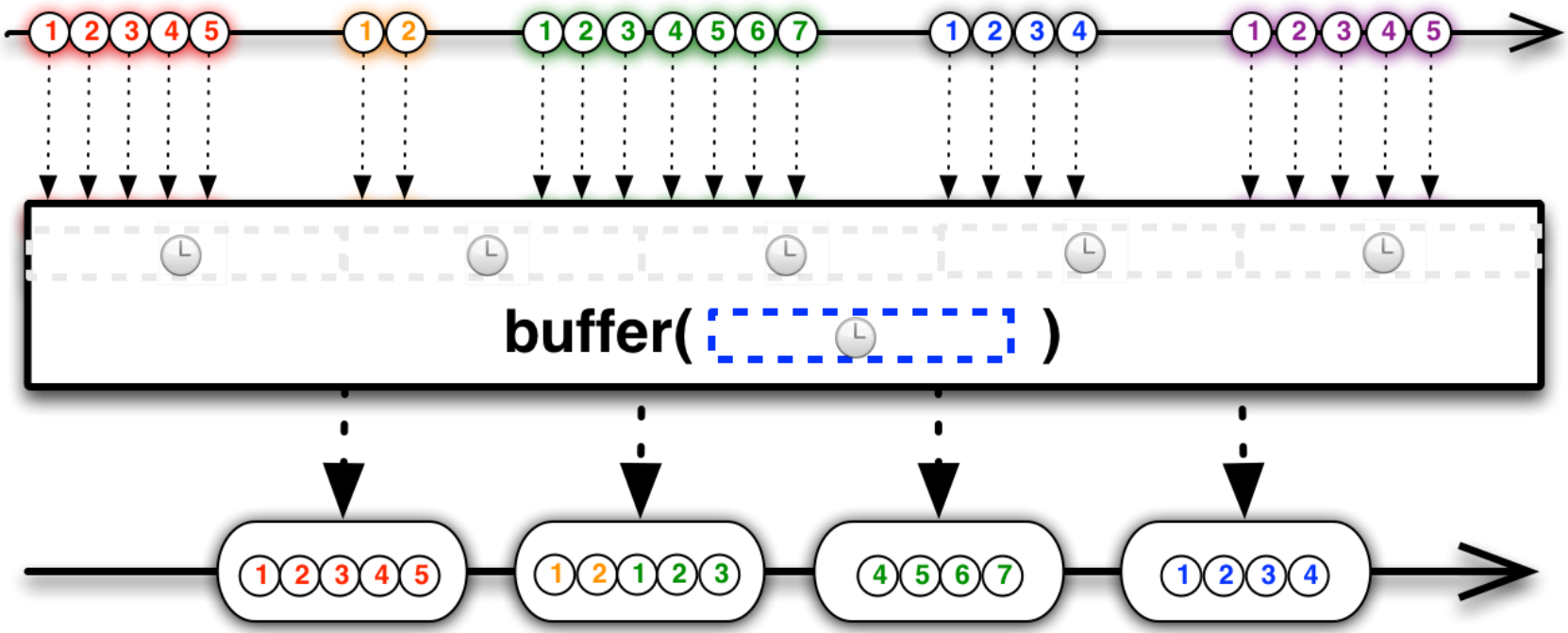
Observable


This is the timeline of the Observable. Time flows from left to right.
These are items emitted by the Observable.
This vertical line indicates that the Observable has completed successfully.
flip
These dotted lines and this box indicate that a transformation is being applied to the Observable. The text inside the box shows the nature of the transformation.
This Observable is the result of the transformation.
If for some reason the Observable terminates abnormally, with an error, the vertical line is replaced by an X.

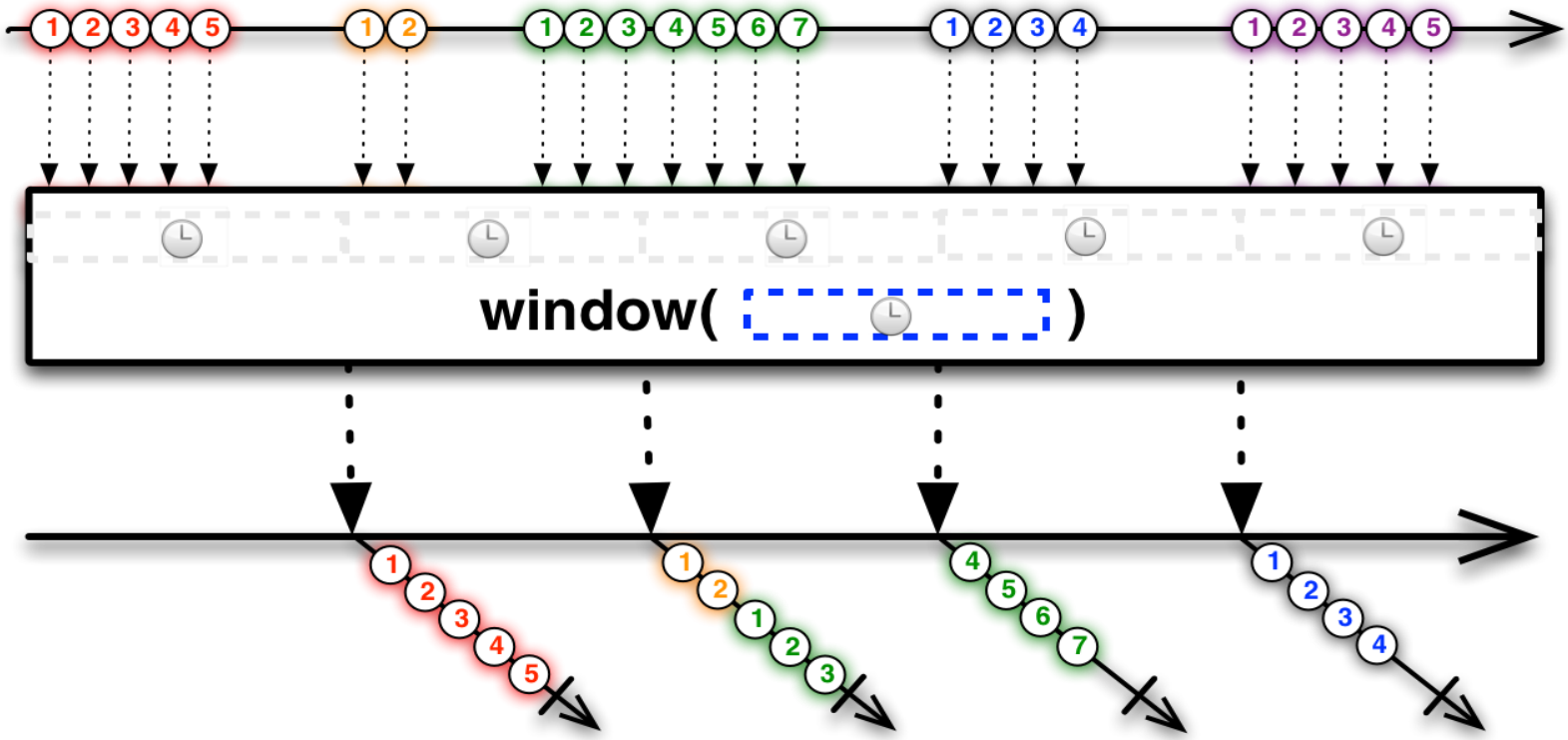
Rx na JVM

- Dlaczego na JVM?
  - Rx.NET
- Dlaczego RxJava?
  - RxScala
  - RxGroovy
- Integracja
  - Reactive Streams

Buffer



Window



Źródła, referencje

- http://www.reactivemanifesto.org/
- http://en.wikipedia.org/wiki/Reactive_programming
- https://github.com/ReactiveX/RxJava
- http://java.dzone.com/articles/whats-wrong-java-8-part-iv

Dziękuję za uwagę

Q&A