Workshop de Programação Reativa com ReactiveX

Fabio Galuppo, M.Sc.
http://member.acm.org/~fabiogaluppo
fabiogaluppo@acm.org

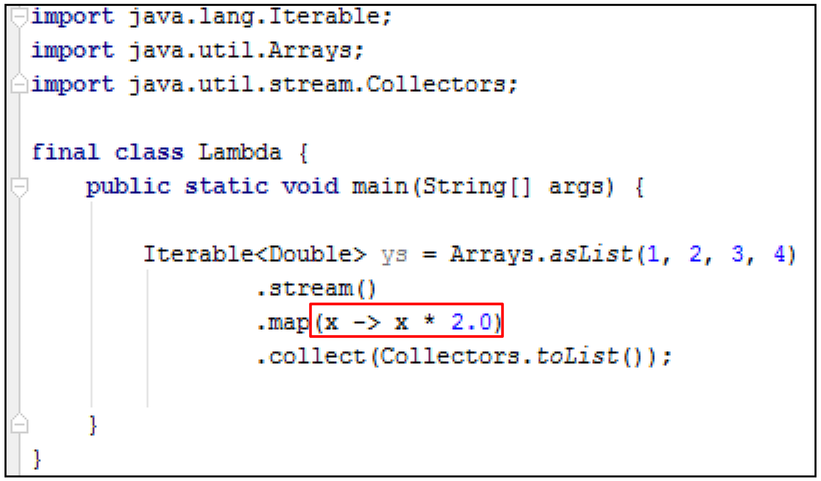
Junho 2015

http://bit.ly/rxworkshop

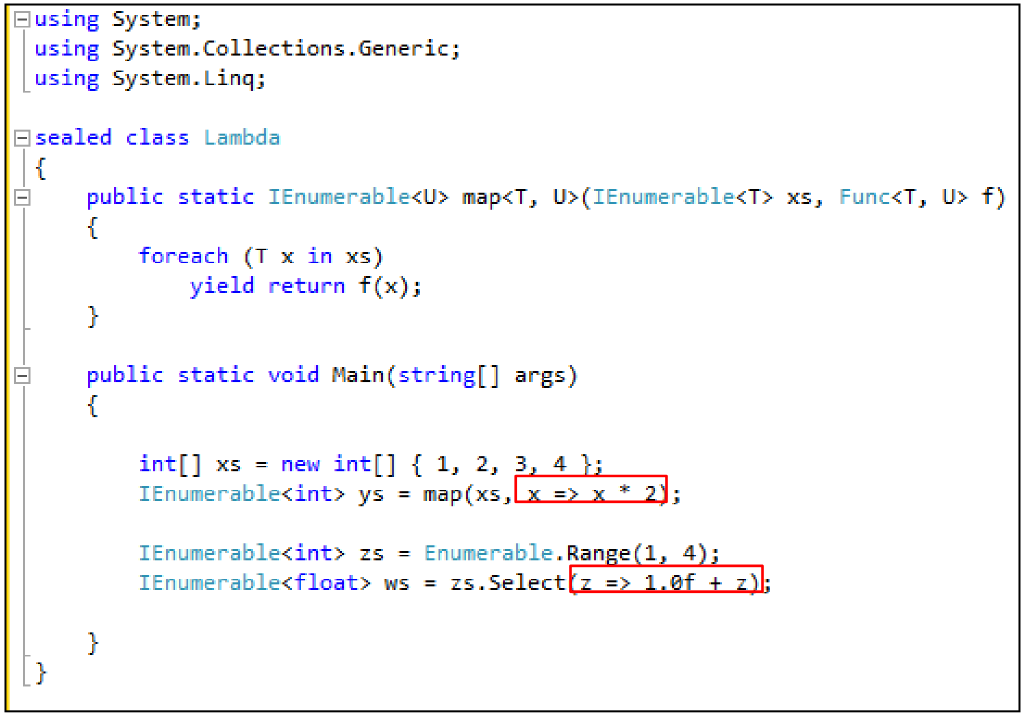
Warm up

- Lambda
- Composição
- Futures

Lambda (λ)

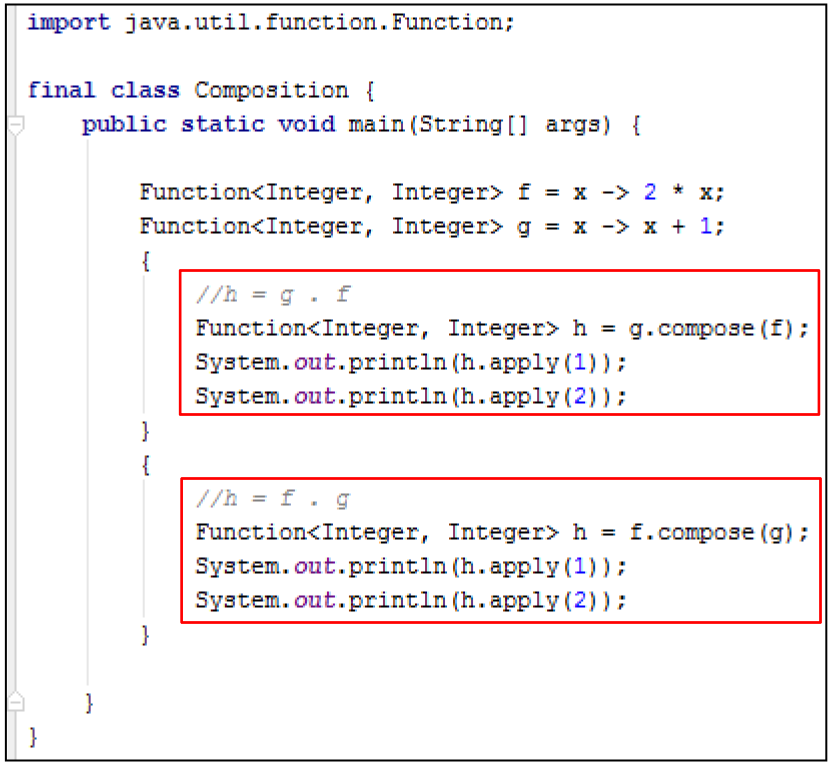


```java
import java.lang.Iterable;
import java.util.Arrays;
import java.util.stream.Collectors;

final class Lambda {
    public static void main(String[] args) {

        Iterable<Double> ys = Arrays.asList(1, 2, 3, 4)
                .stream()
                .map(x -> x * 2.0)
                .collect(Collectors.toList());

    }
}
```

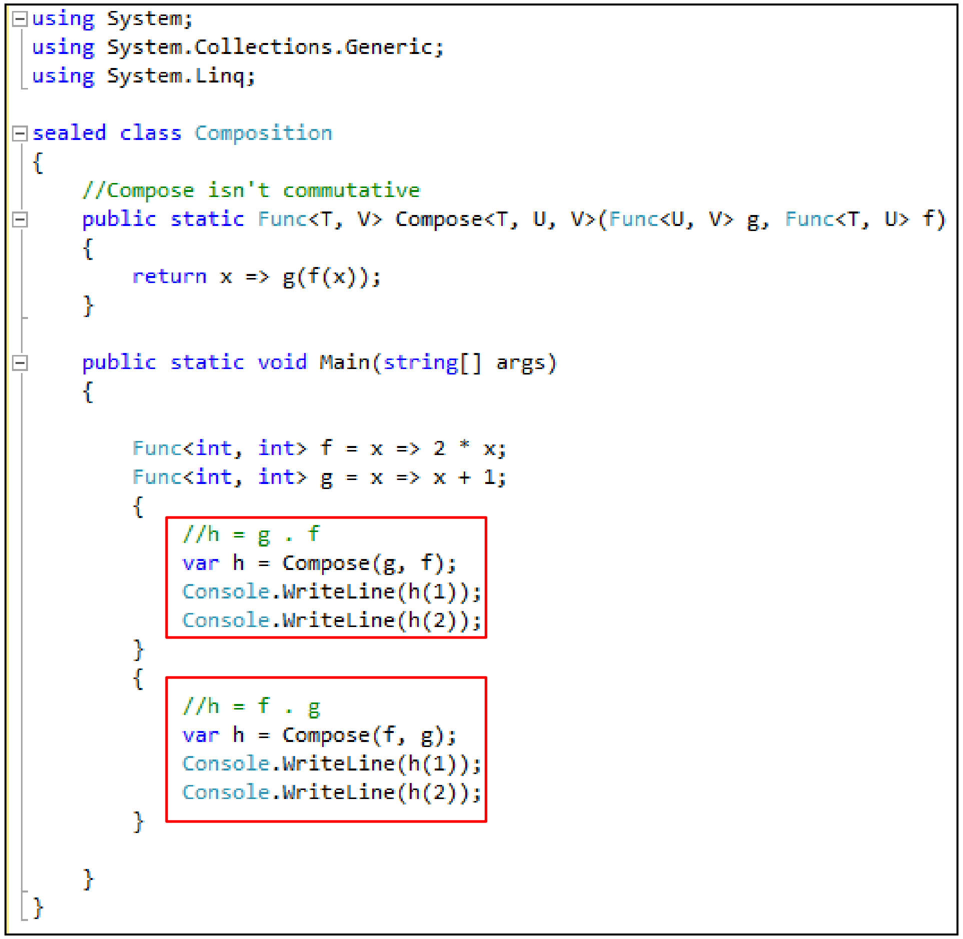


```csharp
using System;
using System.Collections.Generic;
using System.Linq;

sealed class Lambda
{
    public static IEnumerable<U> map<T, U>(IEnumerable<T> xs, Func<T, U> f)
    {
        foreach (T x in xs)
            yield return f(x);
    }

    public static void Main(string[] args)
    {

        int[] xs = new int[] { 1, 2, 3, 4 };
        IEnumerable<int> ys = map(xs, x => x * 2);

        IEnumerable<int> zs = Enumerable.Range(1, 4);
        IEnumerable<float> ws = zs.Select(z => 1.0f + z);

    }
}
```

Composição

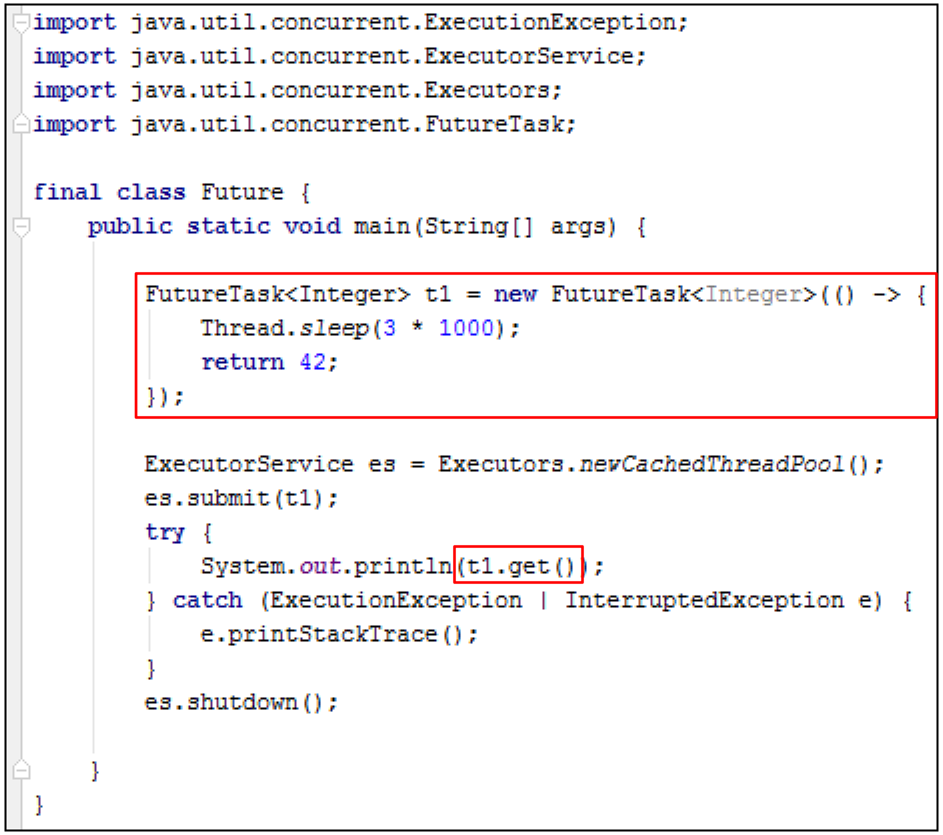


```java
import java.util.function.Function;

final class Composition {
    public static void main(String[] args) {

        Function<Integer, Integer> f = x -> 2 * x;
        Function<Integer, Integer> g = x -> x + 1;
        {
            //h = g . f
            Function<Integer, Integer> h = g.compose(f);
            System.out.println(h.apply(1));
            System.out.println(h.apply(2));
        }
        {
            //h = f . g
            Function<Integer, Integer> h = f.compose(g);
            System.out.println(h.apply(1));
            System.out.println(h.apply(2));
        }

    }
}
```

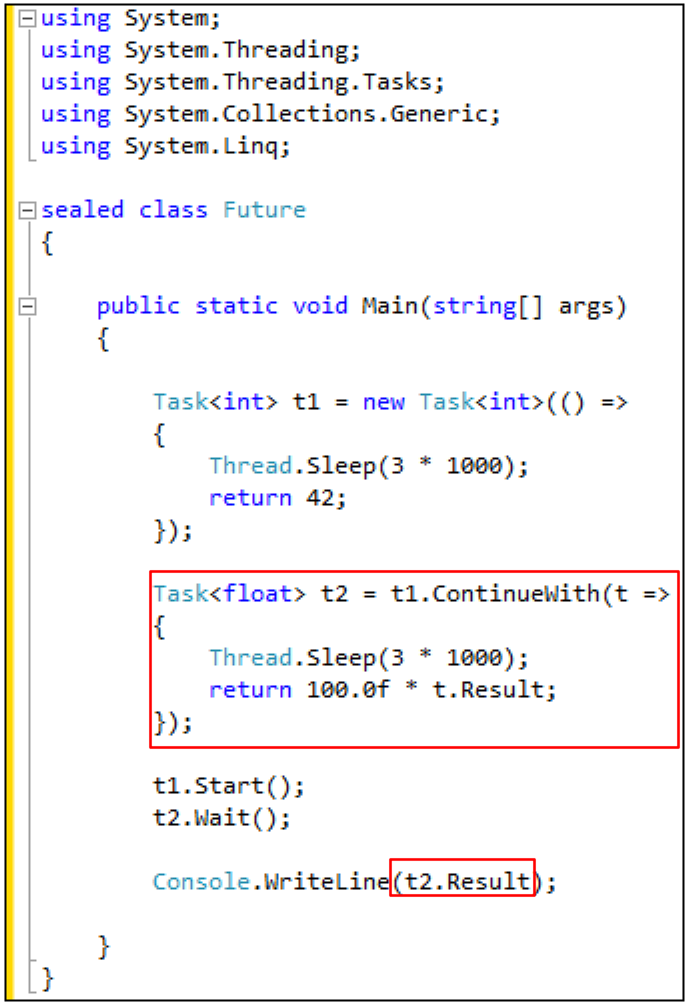


```csharp
using System;
using System.Collections.Generic;
using System.Linq;

sealed class Composition
{
    //Compose isn't commutative
    public static Func<T, V> Compose<T, U, V>(Func<U, V> g, Func<T, U> f)
    {
        return x => g(f(x));
    }

    public static void Main(string[] args)
    {

        Func<int, int> f = x => 2 * x;
        Func<int, int> g = x => x + 1;
        {
            //h = g . f
            var h = Compose(g, f);
            Console.WriteLine(h(1));
            Console.WriteLine(h(2));
        }
        {
            //h = f . g
            var h = Compose(f, g);
            Console.WriteLine(h(1));
            Console.WriteLine(h(2));
        }

    }
}
```

Futures



```java
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.FutureTask;

final class Future {
    public static void main(String[] args) {

        FutureTask<Integer> t1 = new FutureTask<Integer>(() -> {
            Thread.sleep(3 * 1000);
            return 42;
        });

        ExecutorService es = Executors.newCachedThreadPool();
        es.submit(t1);
        try {
            System.out.println(t1.get());
        } catch (ExecutionException | InterruptedException e) {
            e.printStackTrace();
        }
        es.shutdown();

    }
}
```



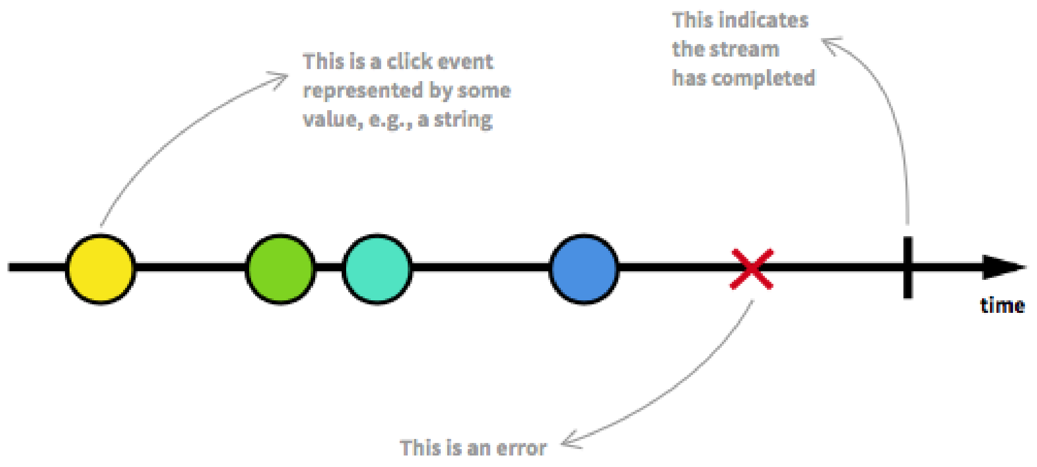
```csharp
using System;
using System.Threading;
using System.Threading.Tasks;
using System.Collections.Generic;
using System.Linq;

sealed class Future
{

    public static void Main(string[] args)
    {

        Task<int> t1 = new Task<int>(() =>
        {
            Thread.Sleep(3 * 1000);
            return 42;
        });

        Task<float> t2 = t1.ContinueWith(t =>
        {
            Thread.Sleep(3 * 1000);
            return 100.0f * t.Result;
        });

        t1.Start();
        t2.Wait();

        Console.WriteLine(t2.Result);

    }
}
```

Agenda

- Programação Reativa
- Reactive Manifesto
- Pull vs. Push
- ReactiveX
- Elementos do Rx
- Single vs. Multiple
  Synchronous vs. Asynchronous
- Marble Diagrams
- Rx Operators

Programação Reativa

Reactive programming is programming with asynchronous data streams.

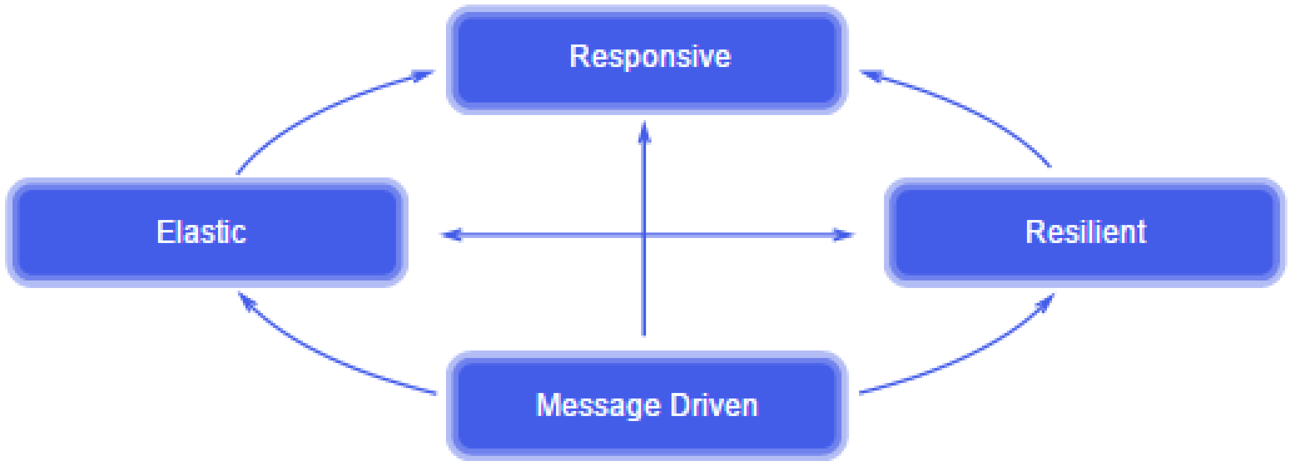

This is a click event represented by some value, e.g., a string
This indicates the stream has completed
time
This is an error

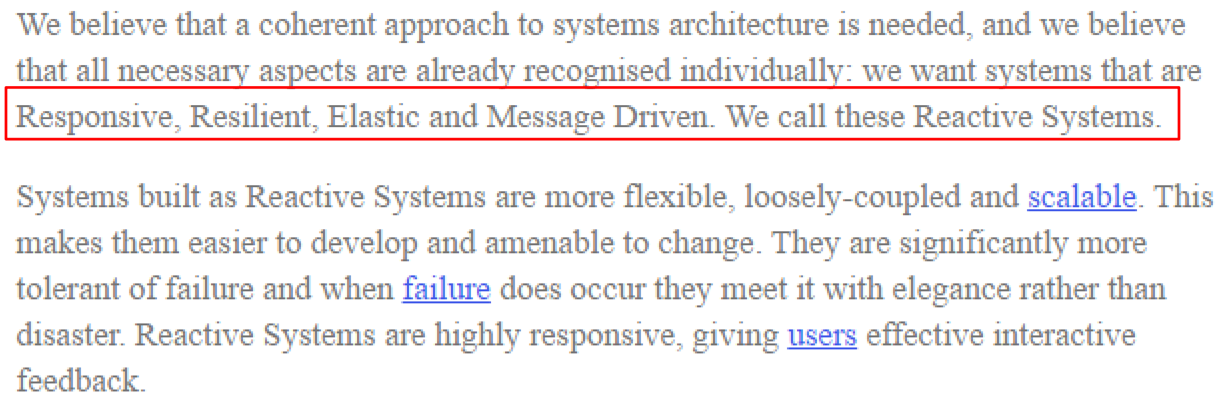
A stream is a sequence of **ongoing events ordered in time**. It can emit three different things: a value (of some type), an error, or a "completed" signal. Consider that the "completed" takes place, for instance, when the current window or view containing that button is closed.

We capture these emitted events only **asynchronously**, by defining a function that will execute when a value is emitted, another function when an error is emitted, and another function when 'completed' is emitted. Sometimes these last two can be omitted and you can just focus on defining the function for values. The "listening" to the stream is called **subscribing**. The functions we are defining are observers. The stream is the subject (or "observable") being observed. This is precisely the Observer Design Pattern.

https://gist.github.com/staltz/868e7e9bc2a7b8c1f754

Reactive Manifesto




We believe that a coherent approach to systems architecture is needed, and we believe that all necessary aspects are already recognised individually: we want systems that are Responsive, Resilient, Elastic and Message Driven. We call these Reactive Systems.

Systems built as Reactive Systems are more flexible, loosely-coupled and scalable. This makes them easier to develop and amenable to change. They are significantly more tolerant of failure and when failure does occur they meet it with elegance rather than disaster. Reactive Systems are highly responsive, giving users effective interactive feedback.

The Reactive Manifesto

11368 people already signed (Go back to the manifesto)

Search: Fabio Galuppo

Fabio Galuppo a year ago (1.0)

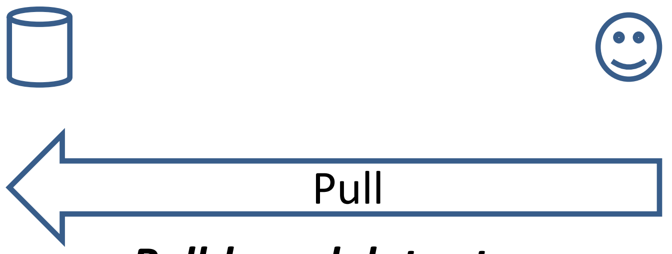
http://www.reactivemanifesto.org/

De um *slide* sobre Programação Reativa. Lembra alguma coisa?

Bid 20, bid 20.2, bid 20.8,...

Pull vs. Push

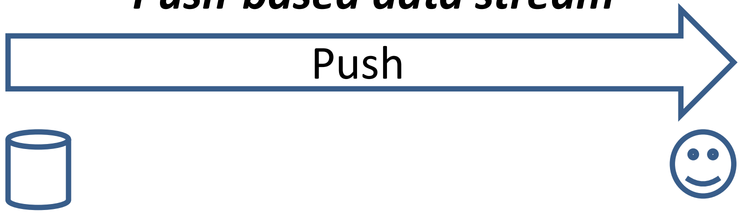
- Ler um arquivo
- Somar números de um *array*
- Iterar sobre o resultado de uma consulta no banco de dados
- Percorrer um diretório



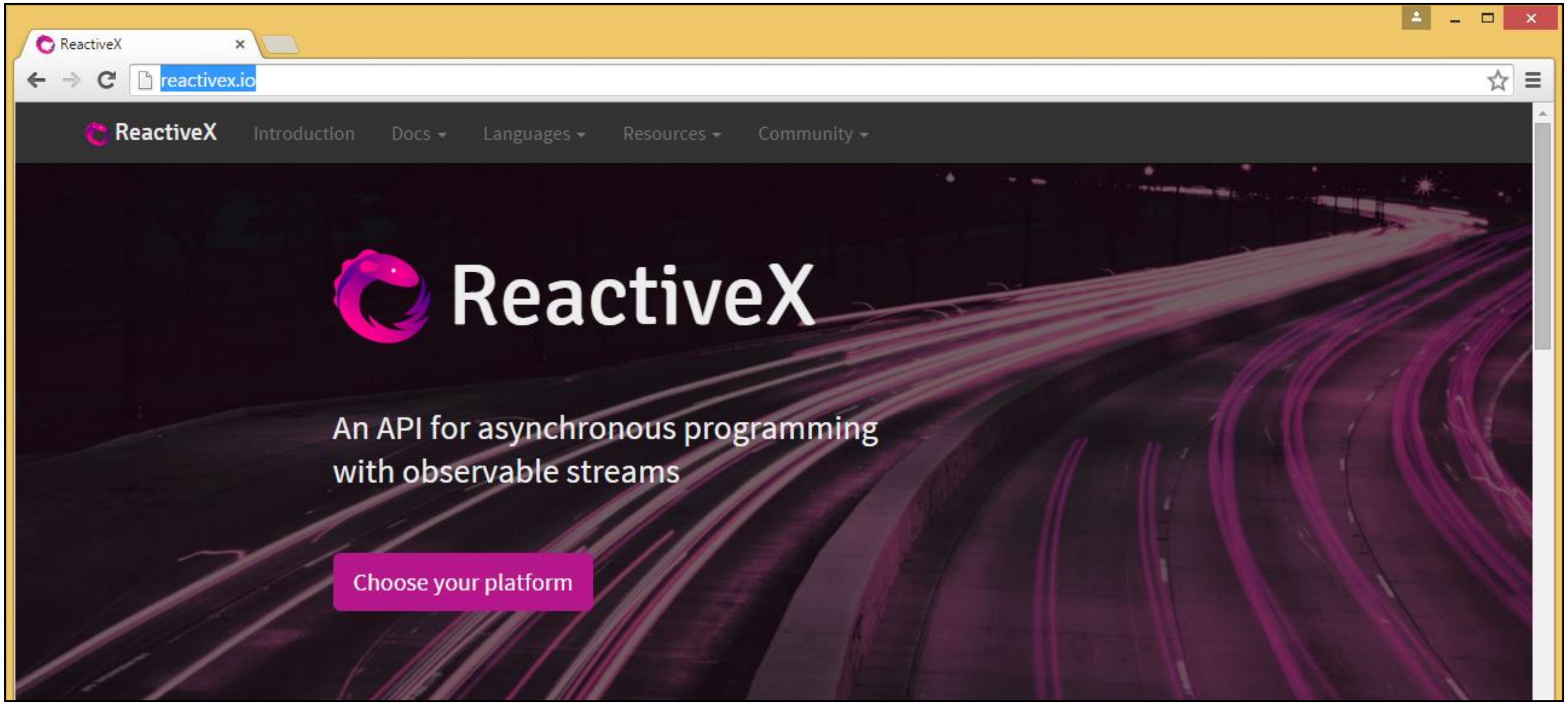
Pull-based data stream

- Dispositivos de medição
  - Tempo, Luz, Calor
- Eventos
  - *Mouse* e Teclado
  - Outros eventos de UI
- *Trigger*
  - Notificação da Inclusão de um Registro
- *Push Notifications* (Alertas)
  - Notícias, Ofertas, Lembretes

Push-based data stream



Por onde começar?
ReactiveX



An API for asynchronous programming with observable streams

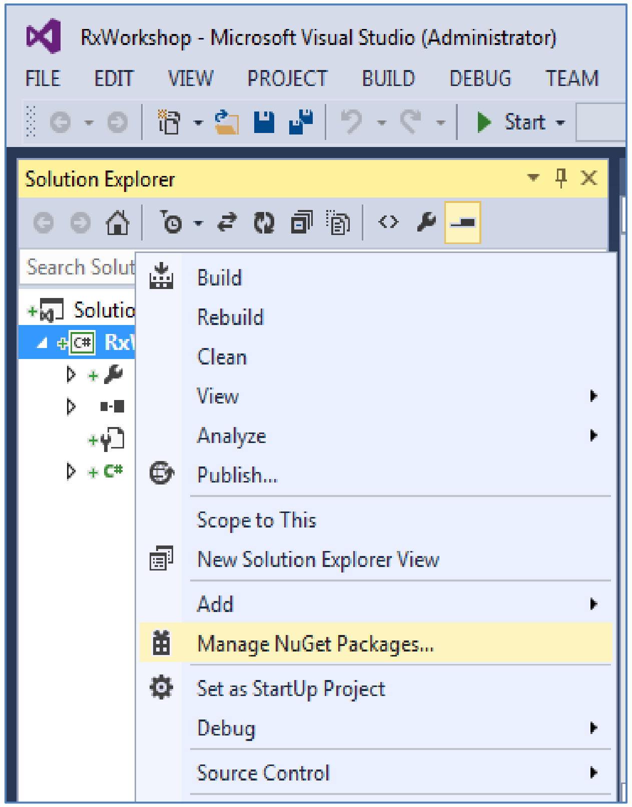
The Observer pattern done right

ReactiveX is a combination of the best ideas from
the Observer pattern, the Iterator pattern, and functional programming

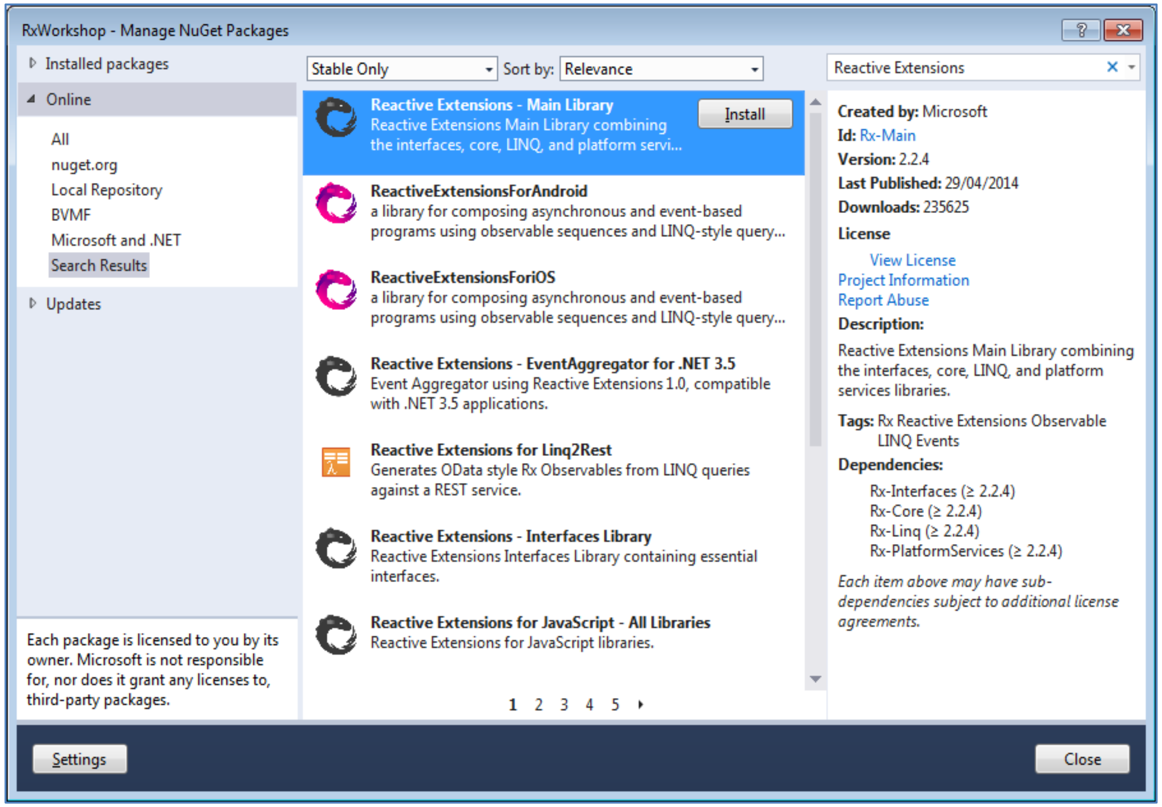
http://reactivex.io/

Reactive Extensions (Rx) é para .NET!

Adicionando pacote via NuGet



(1)



Each package is licensed to you by its owner. Microsoft is not responsible for, nor does it grant any licenses to, third-party packages.

Reactive Extensions - Main Library
Reactive Extensions Main Library combining the interfaces, core, LINQ, and platform servi...

ReactiveExtensionsForAndroid
a library for composing asynchronous and event-based programs using observable sequences and LINQ-style query...

ReactiveExtensionsForiOS
a library for composing asynchronous and event-based programs using observable sequences and LINQ-style query...

Reactive Extensions - EventAggregator for .NET 3.5
Event Aggregator using Reactive Extensions 1.0, compatible with .NET 3.5 applications.

Reactive Extensions for Linq2Rest
Generates OData style Rx Observables from LINQ queries against a REST service.

Reactive Extensions - Interfaces Library
Reactive Extensions Interfaces Library containing essential interfaces.

Reactive Extensions for JavaScript - All Libraries
Reactive Extensions for JavaScript libraries.

Created by: Microsoft
Id: Rx-Main
Version: 2.2.4
Last Published: 29/04/2014
Downloads: 235625
License
View License
Project Information
Report Abuse
Description:
Reactive Extensions Main Library combining the interfaces, core, LINQ, and platform services libraries.
Tags: Rx Reactive Extensions Observable LINQ Events
Dependencies:
Rx-Interfaces (≥ 2.2.4)
Rx-Core (≥ 2.2.4)
Rx-Linq (≥ 2.2.4)
Rx-PlatformServices (≥ 2.2.4)

Each item above may have sub-dependencies subject to additional license agreements.

(2)

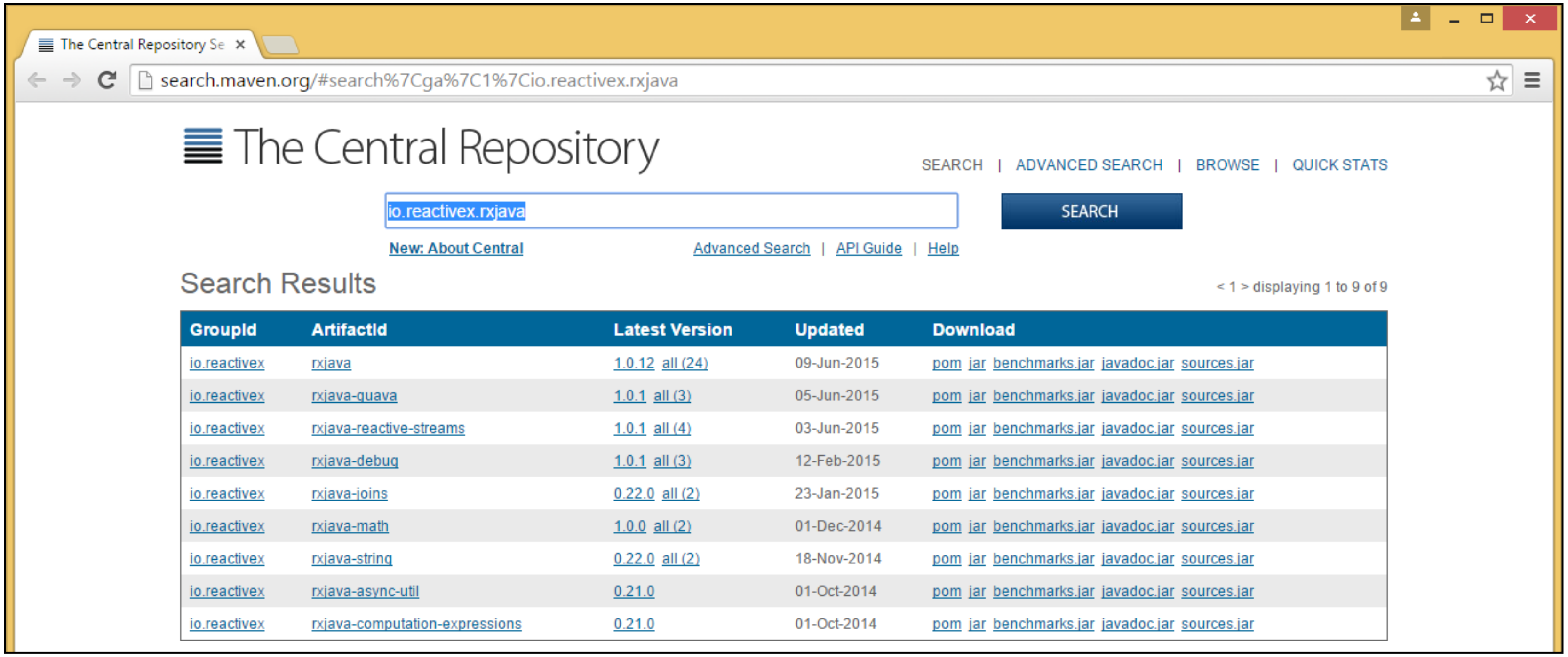
https://www.nuget.org/packages/Rx-Main/

Rx = Observables + LINQ + Schedulers.

https://msdn.microsoft.com/en-us/data/gg577609.aspx

RxJava é para JVM!

Binários e Dependências no http://search.maven.org


The Central Repository Se
search.maven.org/#search%7Cga%7C1%7Cio.reactivex.rxjava
The Central Repository
SEARCH | ADVANCED SEARCH | BROWSE | QUICK STATS
io.reactivex.rxjava
SEARCH
New: About Central
Advanced Search | API Guide | Help

Search Results

< 1 > displaying 1 to 9 of 9

GroupId	ArtifactId	Latest Version	Updated	Download
io.reactivex	rxjava	1.0.12 all (24)	09-Jun-2015	pom jar benchmarks.jar javadoc.jar sources.jar
io.reactivex	rxjava-guava	1.0.1 all (3)	05-Jun-2015	pom jar benchmarks.jar javadoc.jar sources.jar
io.reactivex	rxjava-reactive-streams	1.0.1 all (4)	03-Jun-2015	pom jar benchmarks.jar javadoc.jar sources.jar
io.reactivex	rxjava-debug	1.0.1 all (3)	12-Feb-2015	pom jar benchmarks.jar javadoc.jar sources.jar
io.reactivex	rxjava-joins	0.22.0 all (2)	23-Jan-2015	pom jar benchmarks.jar javadoc.jar sources.jar
io.reactivex	rxjava-math	1.0.0 all (2)	01-Dec-2014	pom jar benchmarks.jar javadoc.jar sources.jar
io.reactivex	rxjava-string	0.22.0 all (2)	18-Nov-2014	pom jar benchmarks.jar javadoc.jar sources.jar
io.reactivex	rxjava-async-util	0.21.0	01-Oct-2014	pom jar benchmarks.jar javadoc.jar sources.jar
io.reactivex	rxjava-computation-expressions	0.21.0	01-Oct-2014	pom jar benchmarks.jar javadoc.jar sources.jar

http://search.maven.org/#search%7Cga%7C1%7Cio.reactivex.rxjava

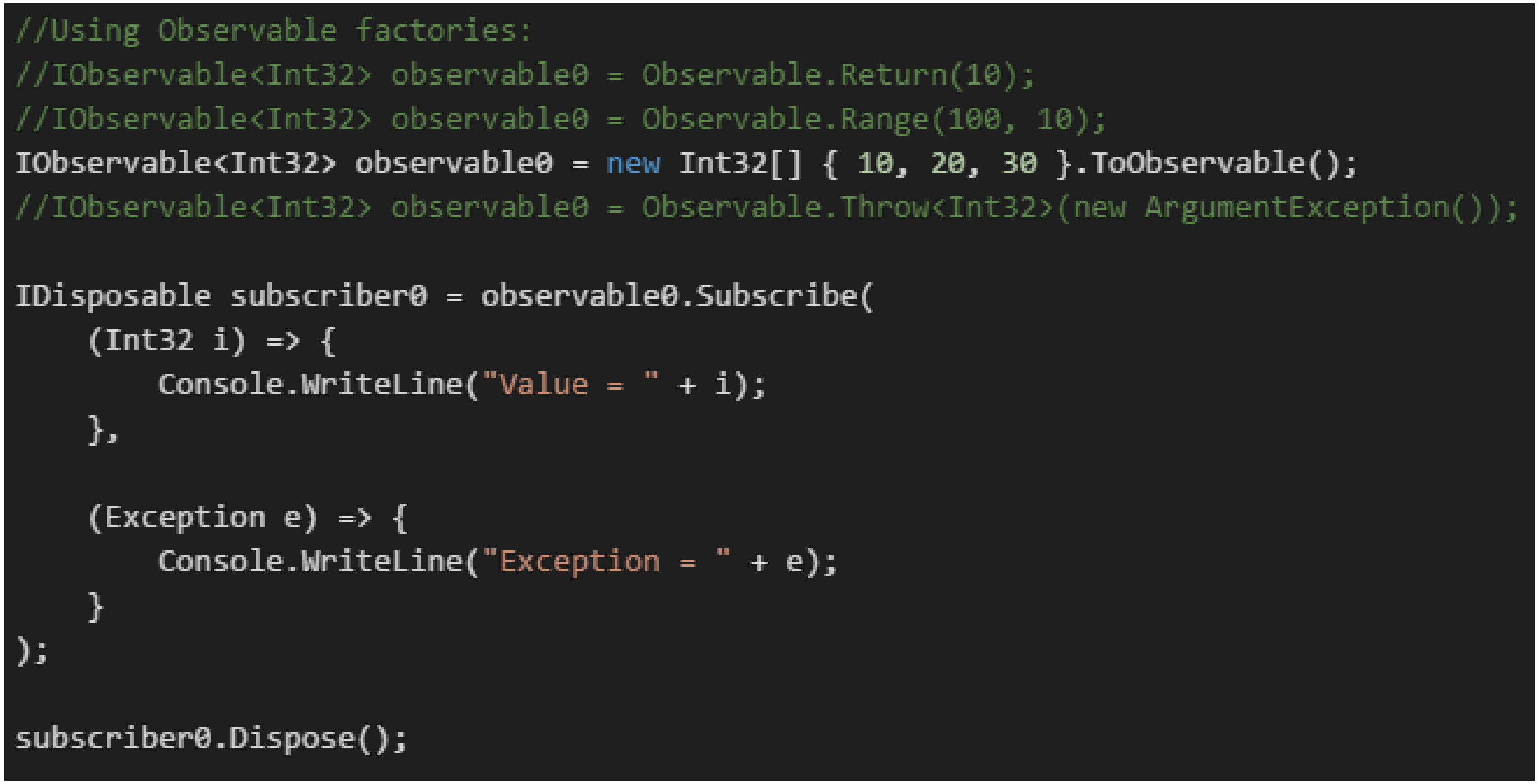
RxJava is a Java VM implementation of Reactive Extensions: a library for composing asynchronous and event-based programs by using observable sequences

https://github.com/ReactiveX/RxJava

Elementos do Rx

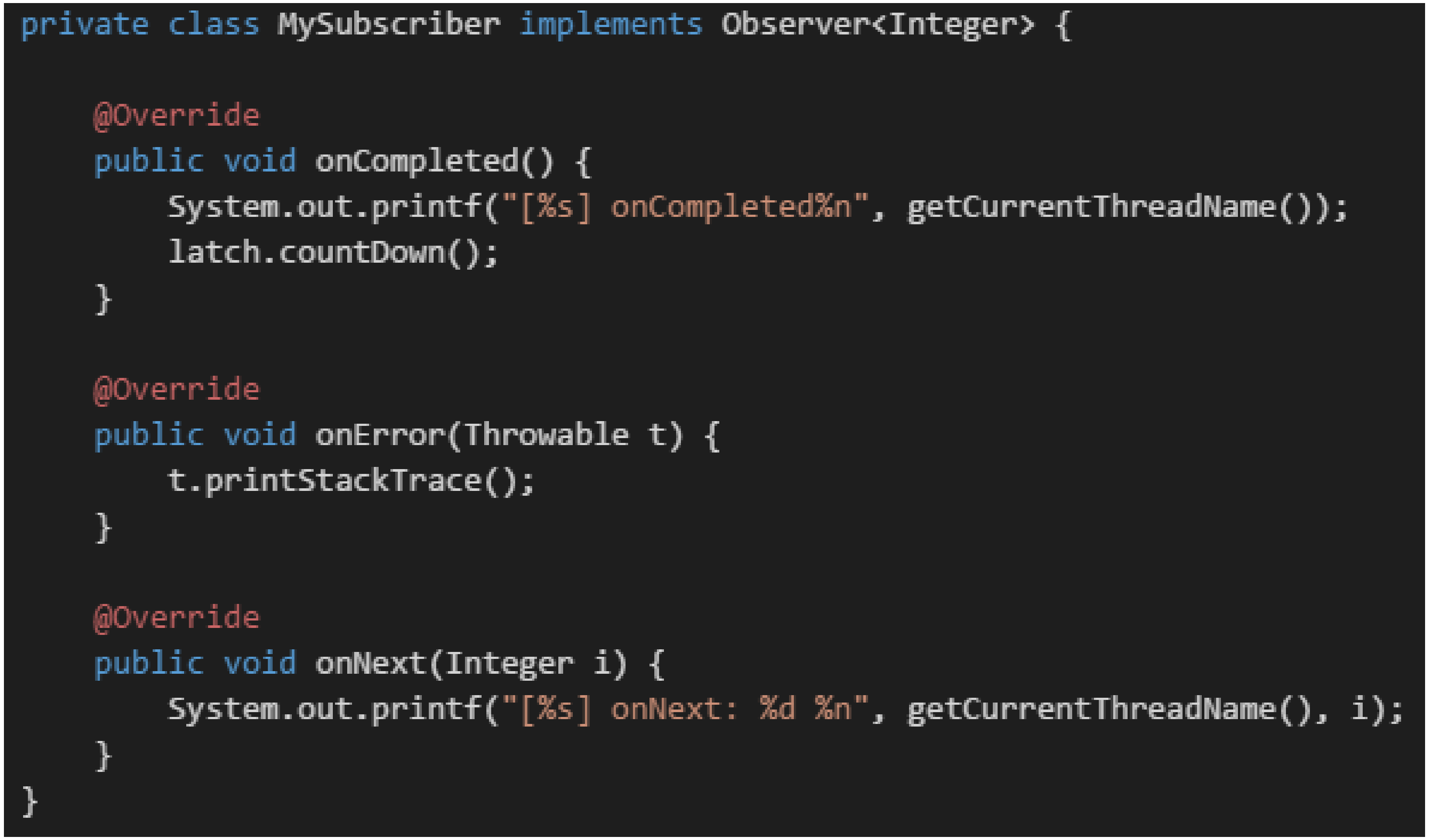
- Observable (http://reactivex.io/documentation/observable.html)
  - *Stream* asíncrona que emite eventos:
    - **`OnNext`**, **`OnCompleted`**, **`OnError`**
- Observer (http://www.introtorx.com/Content/v1.0.10621.0/02_KeyTypes.html#IObserver)
  - Assina e reage a eventos de um *Observable*
- Operadores (http://reactivex.io/documentation/operators.html)
  - Criação, Transformação, Seleção, Combinação, …
- Subject (http://reactivex.io/documentation/subject.html)
  - *Proxy* que atua como Observable e Observer
- Scheduler (http://reactivex.io/documentation/scheduler.html)
  - Controle do comportamento *multithreading*

Observable Factories



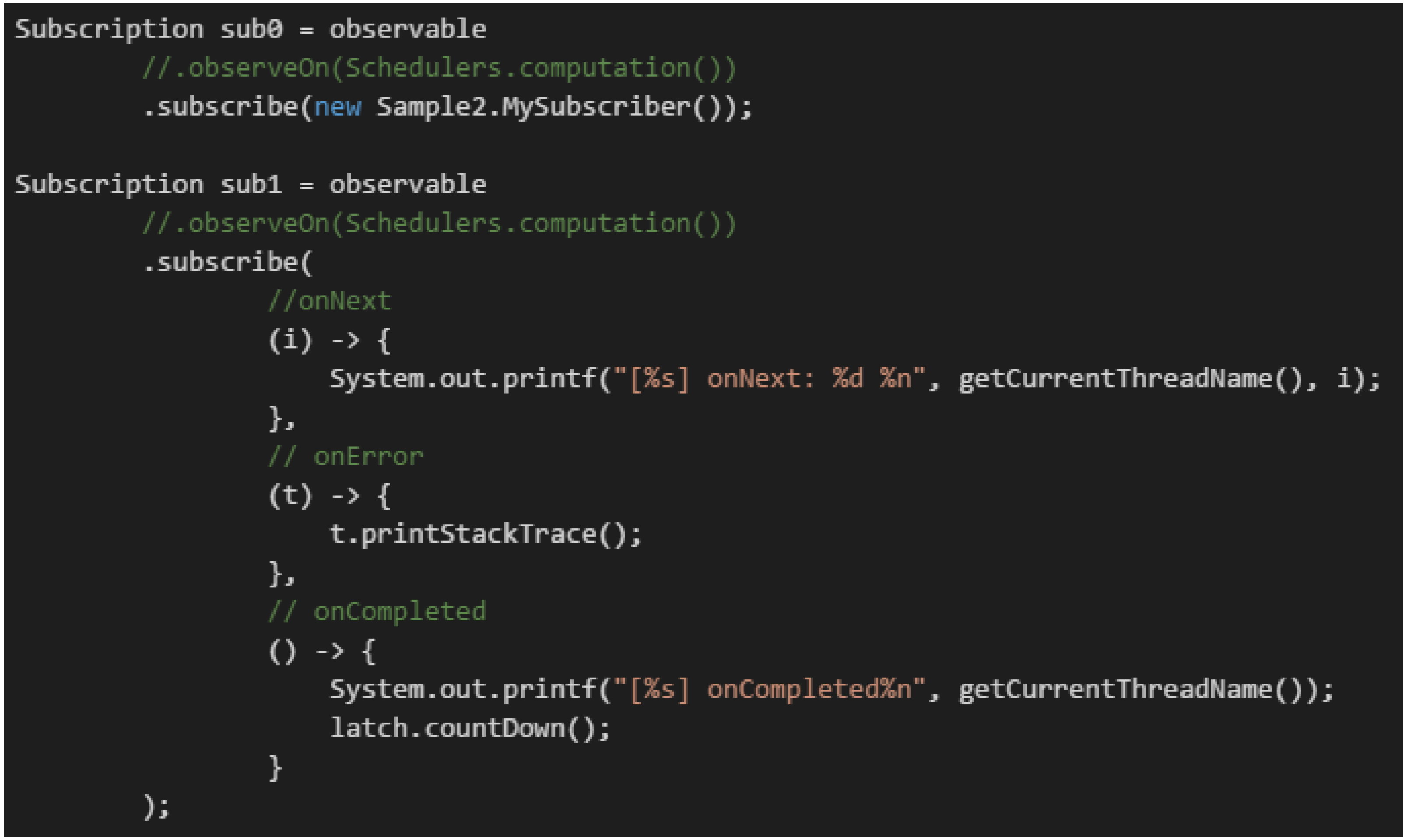
```
//Using Observable factories:
//IObservable<Int32> observable0 = Observable.Return(10);
//IObservable<Int32> observable0 = Observable.Range(100, 10);
IObservable<Int32> observable0 = new Int32[] { 10, 20, 30 }.ToObservable();
//IObservable<Int32> observable0 = Observable.Throw<Int32>(new ArgumentException());

IDisposable subscriber0 = observable0.Subscribe(
    (Int32 i) => {
        Console.WriteLine("Value = " + i);
    },

    (Exception e) => {
        Console.WriteLine("Exception = " + e);
    }
);

subscriber0.Dispose();
```

Observer



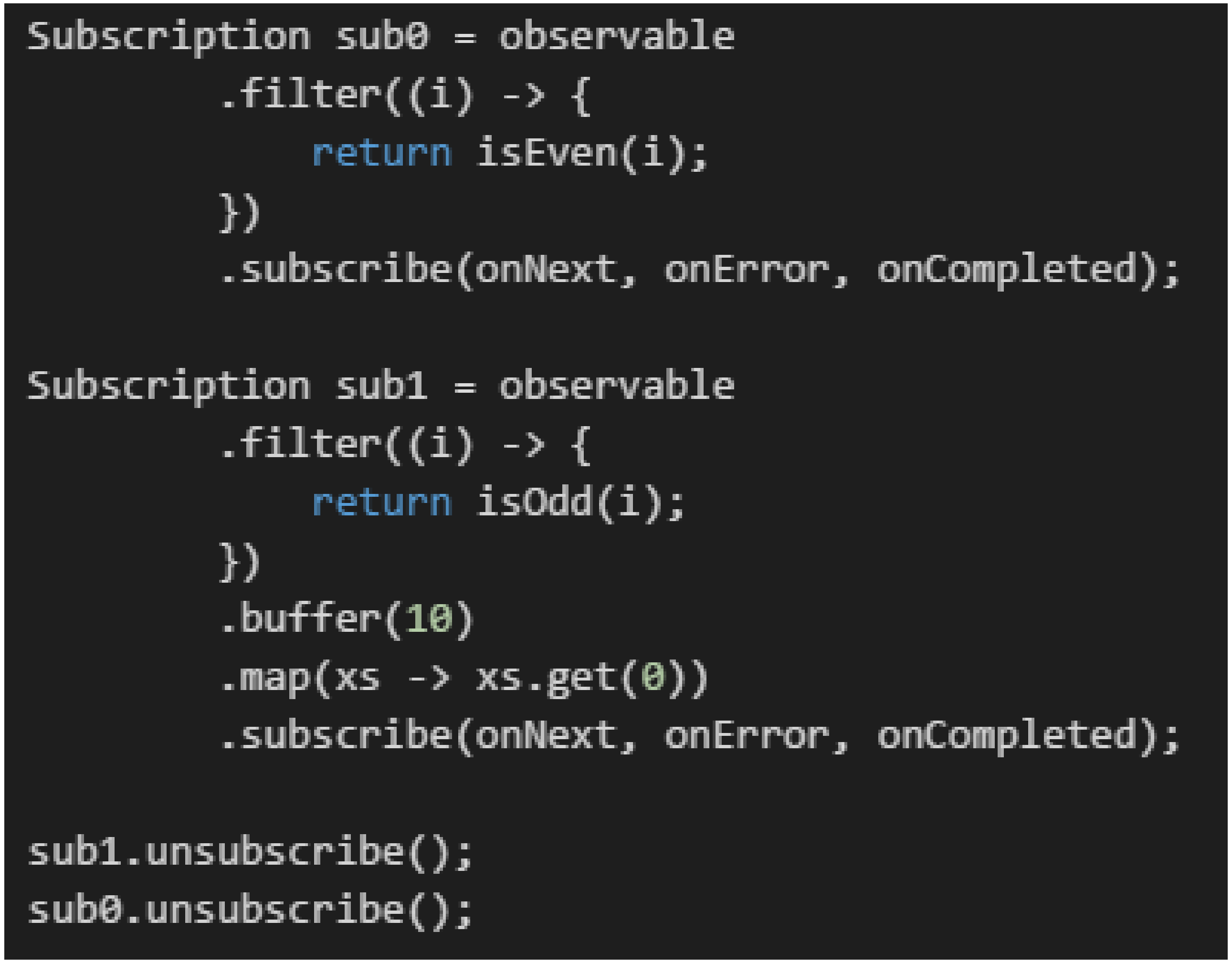
```
private class MySubscriber implements Observer<Integer> {

    @Override
    public void onCompleted() {
        System.out.printf("[%s] onCompleted%n", getCurrentThreadName());
        latch.countDown();
    }

    @Override
    public void onError(Throwable t) {
        t.printStackTrace();
    }

    @Override
    public void onNext(Integer i) {
        System.out.printf("[%s] onNext: %d %n", getCurrentThreadName(), i);
    }
}
```

Schedulers



```
Subscription sub0 = observable
        //.observeOn(Schedulers.computation())
        .subscribe(new Sample2.MySubscriber());

Subscription sub1 = observable
        //.observeOn(Schedulers.computation())
        .subscribe(
                //onNext
                (i) -> {
                    System.out.printf("[%s] onNext: %d %n", getCurrentThreadName(), i);
                },
                // onError
                (t) -> {
                    t.printStackTrace();
                },
                // onCompleted
                () -> {
                    System.out.printf("[%s] onCompleted%n", getCurrentThreadName());
                    latch.countDown();
                }
        );
```

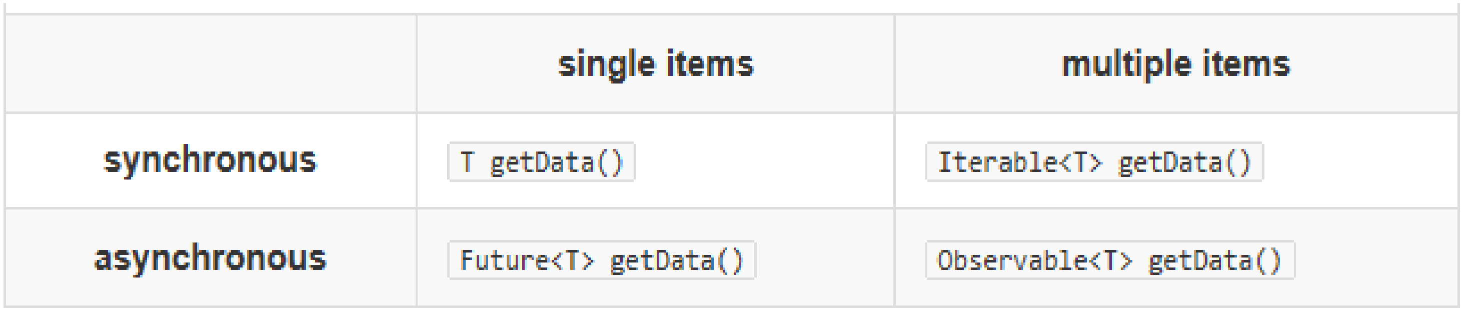
Operators



```
Subscription sub0 = observable
        .filter((i) -> {
            return isEven(i);
        })
        .subscribe(onNext, onError, onCompleted);

Subscription sub1 = observable
        .filter((i) -> {
            return isOdd(i);
        })
        .buffer(10)
        .map(xs -> xs.get(0))
        .subscribe(onNext, onError, onCompleted);

sub1.unsubscribe();
sub0.unsubscribe();
```

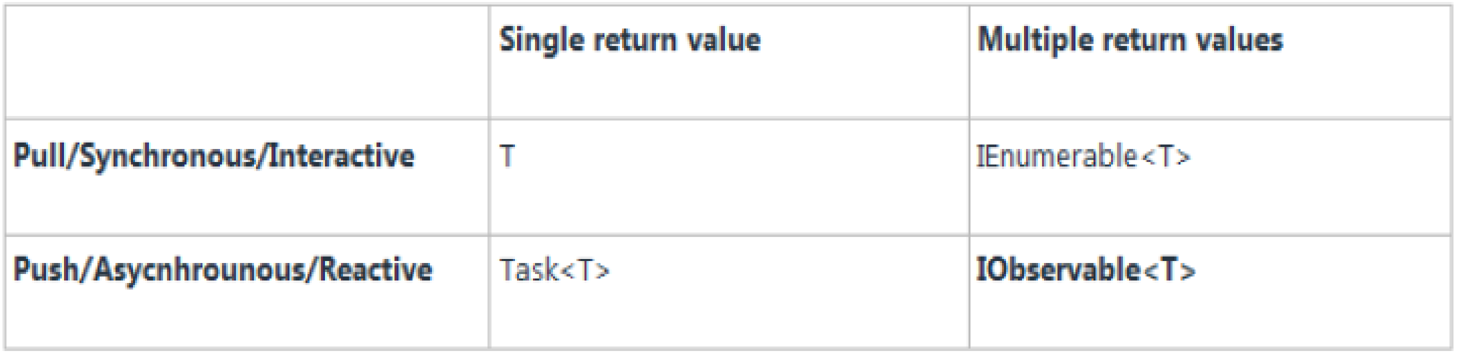
Single vs. Multiple
Synchronous vs. Asynchronous

Java



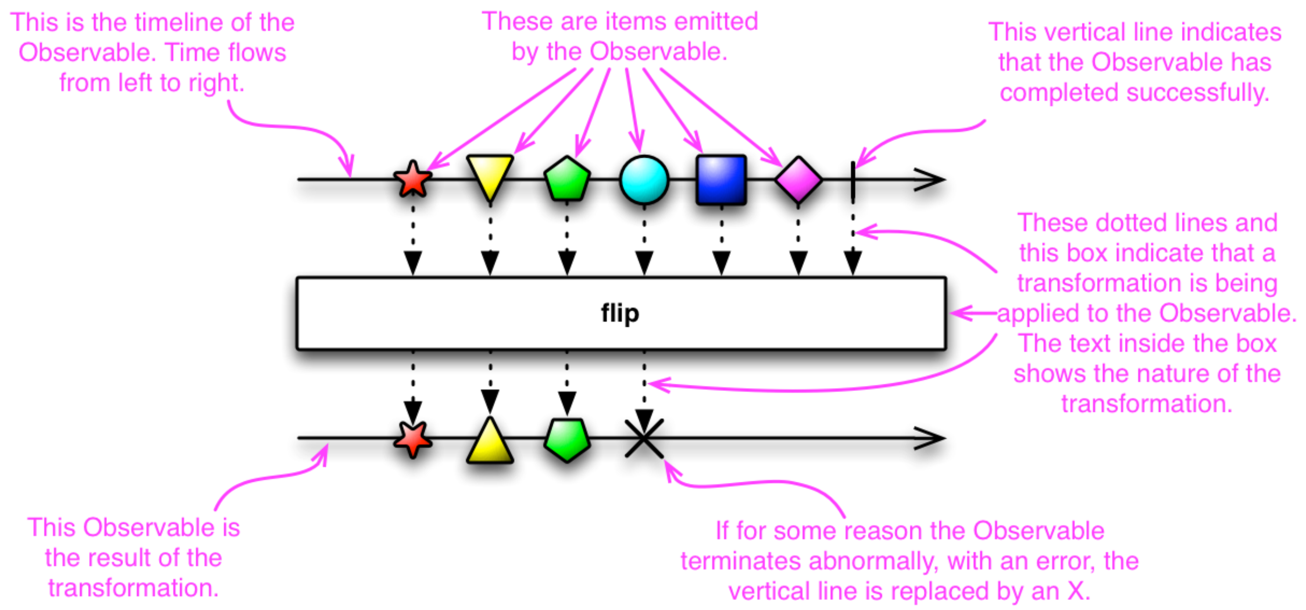
	single items	multiple items
synchronous	`T getData()`	`Iterable<T> getData()`
asynchronous	`Future<T> getData()`	`Observable<T> getData()`

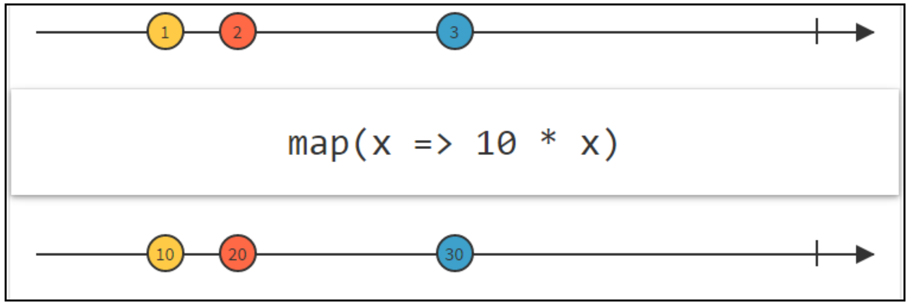
.NET



	Single return value	Multiple return values
Pull/Synchronous/Interactive	T	IEnumerable<T>
Push/Asycnhrounous/Reactive	Task<T>	**IObservable<T>**

Marble Diagrams


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.
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.
flip
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.



```
--1-----2---------------3----------------------|->
vvvvvvvvvvvvvvv map(x => 10 * x) vvvvvvvvvvvvvvv
--10----20--------------30---------------------|->
```

http://rxmarbles.com/

http://rxwiki.wikidot.com/marble-diagrams

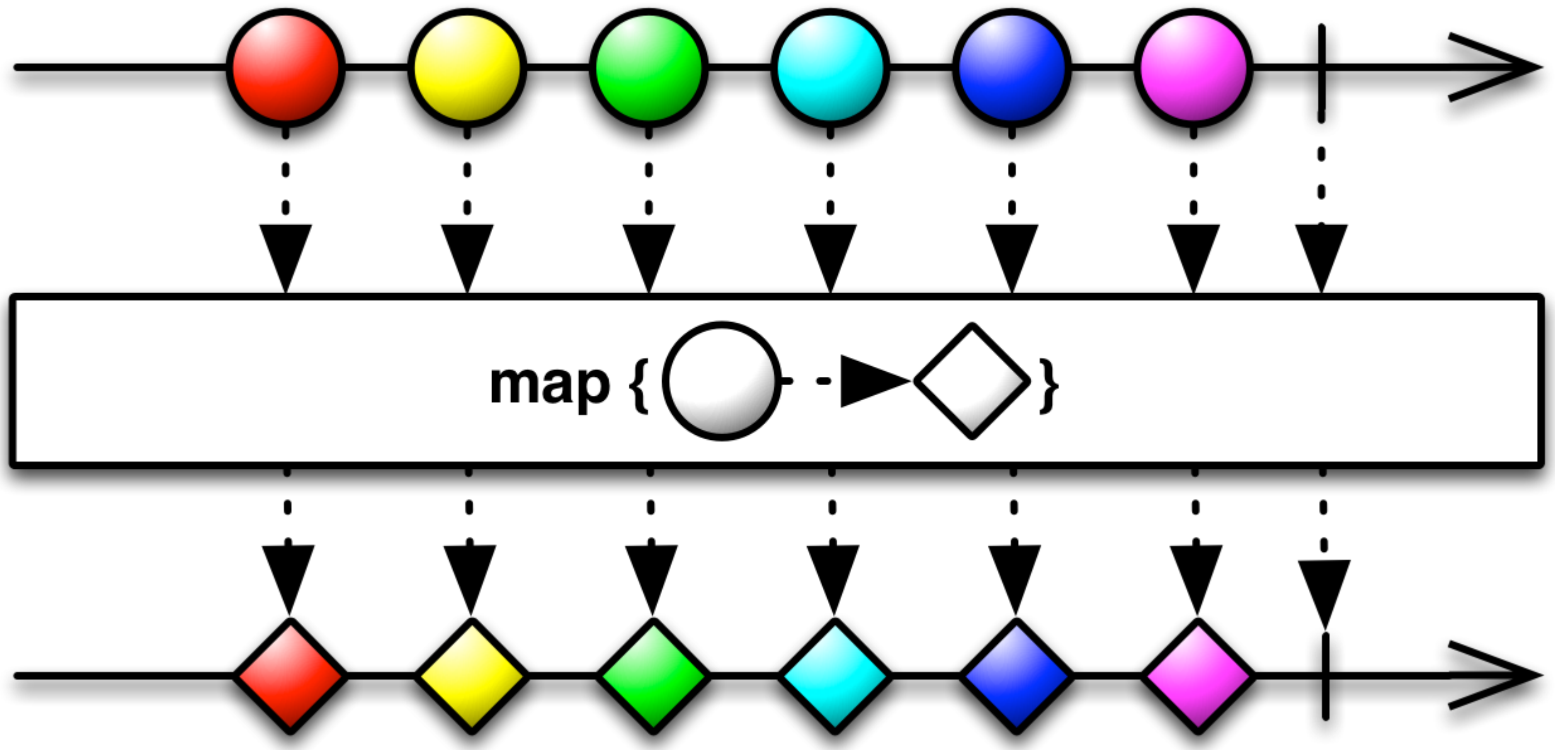
Rx Operators (Marble Diagrams)

Alguns exemplos de operadores:

- **map** (transformação)
- **filter** (seleção)
- **flatMap** (transformação)
- **merge** (combinação)
- **concat** (agregação)
- **sample** (seleção)
- **buffer** (transformação)
- **zip** (combinação)
- **groupBy** (transformação)
- **sum** (agregação)

http://reactivex.io/documentation/operators.html

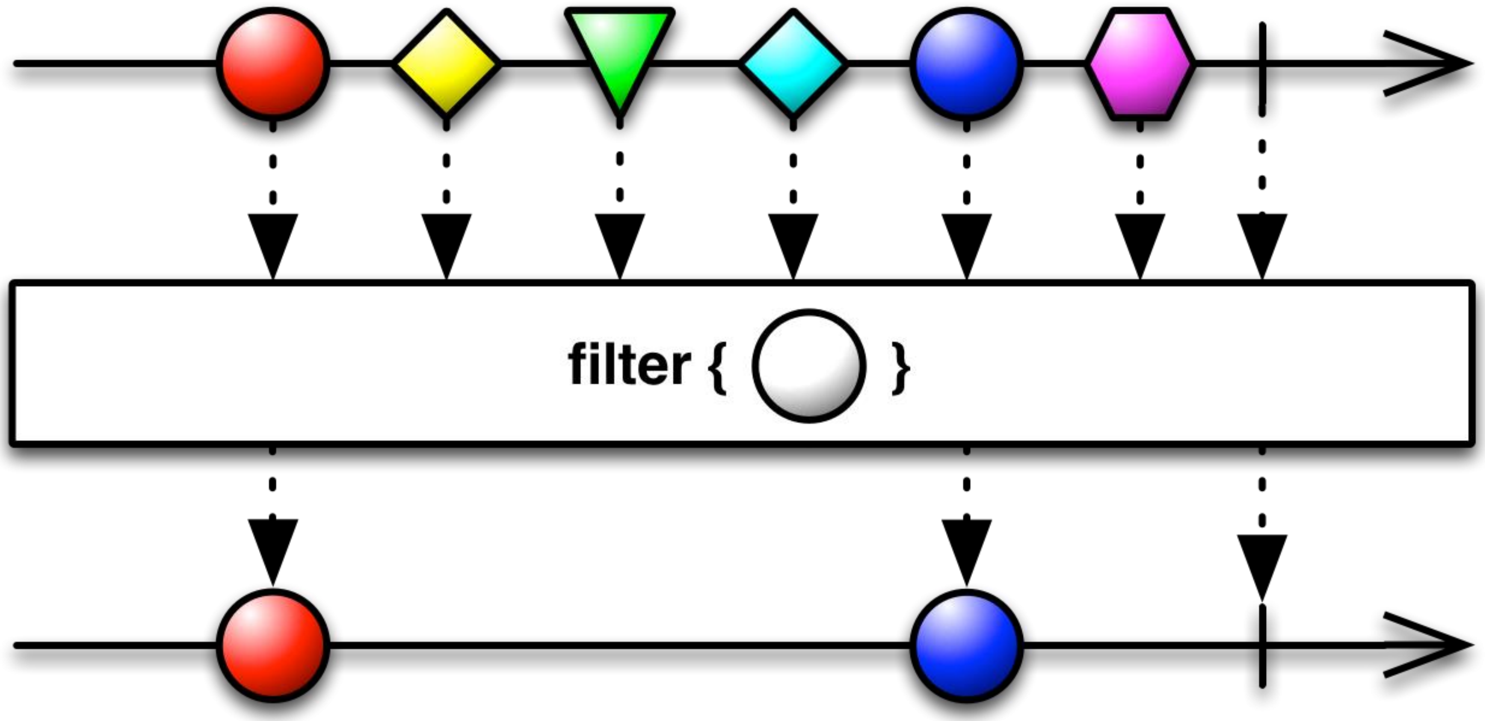
map ≡ Select



http://reactivex.io/documentation/operators/map.html

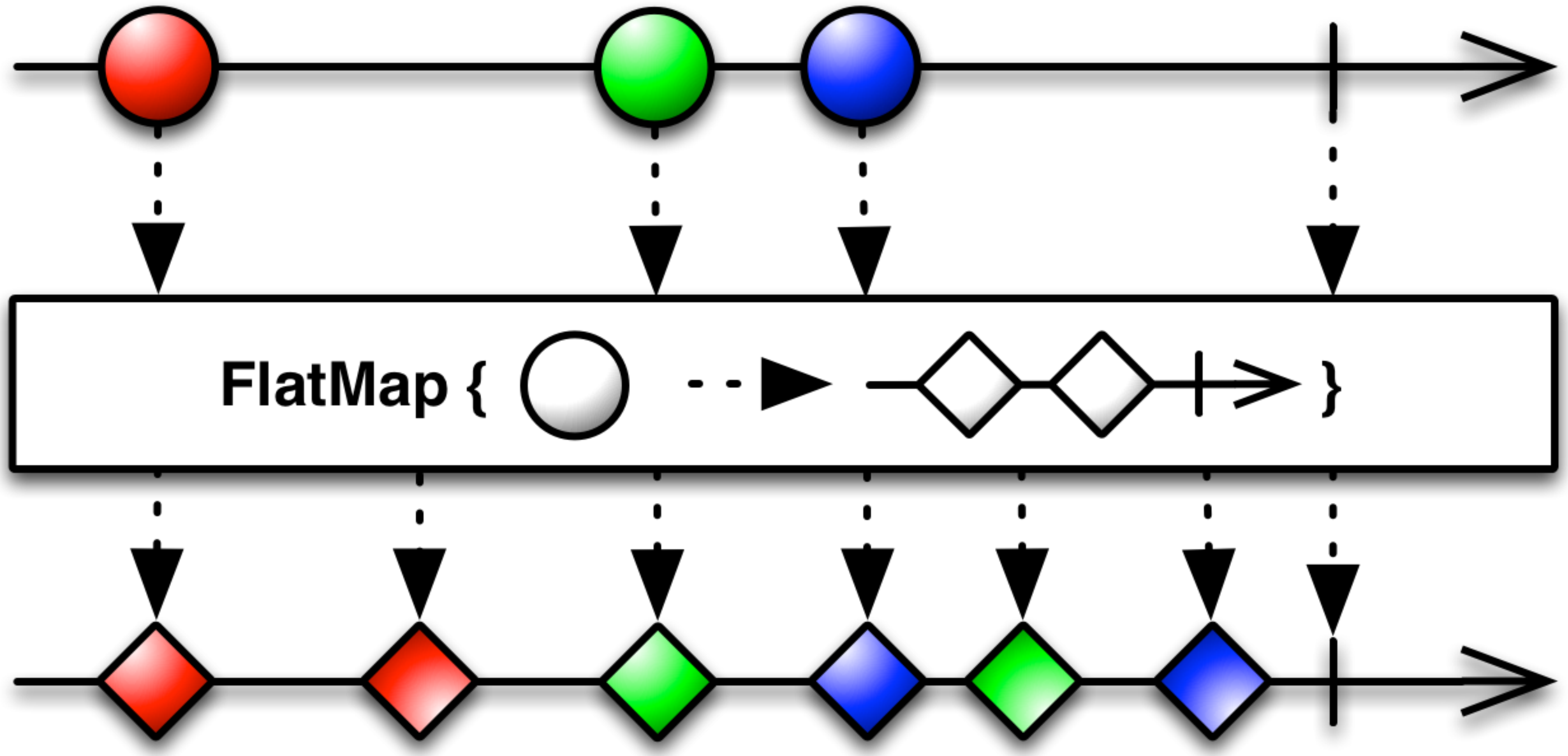
Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

filter ≡ Where



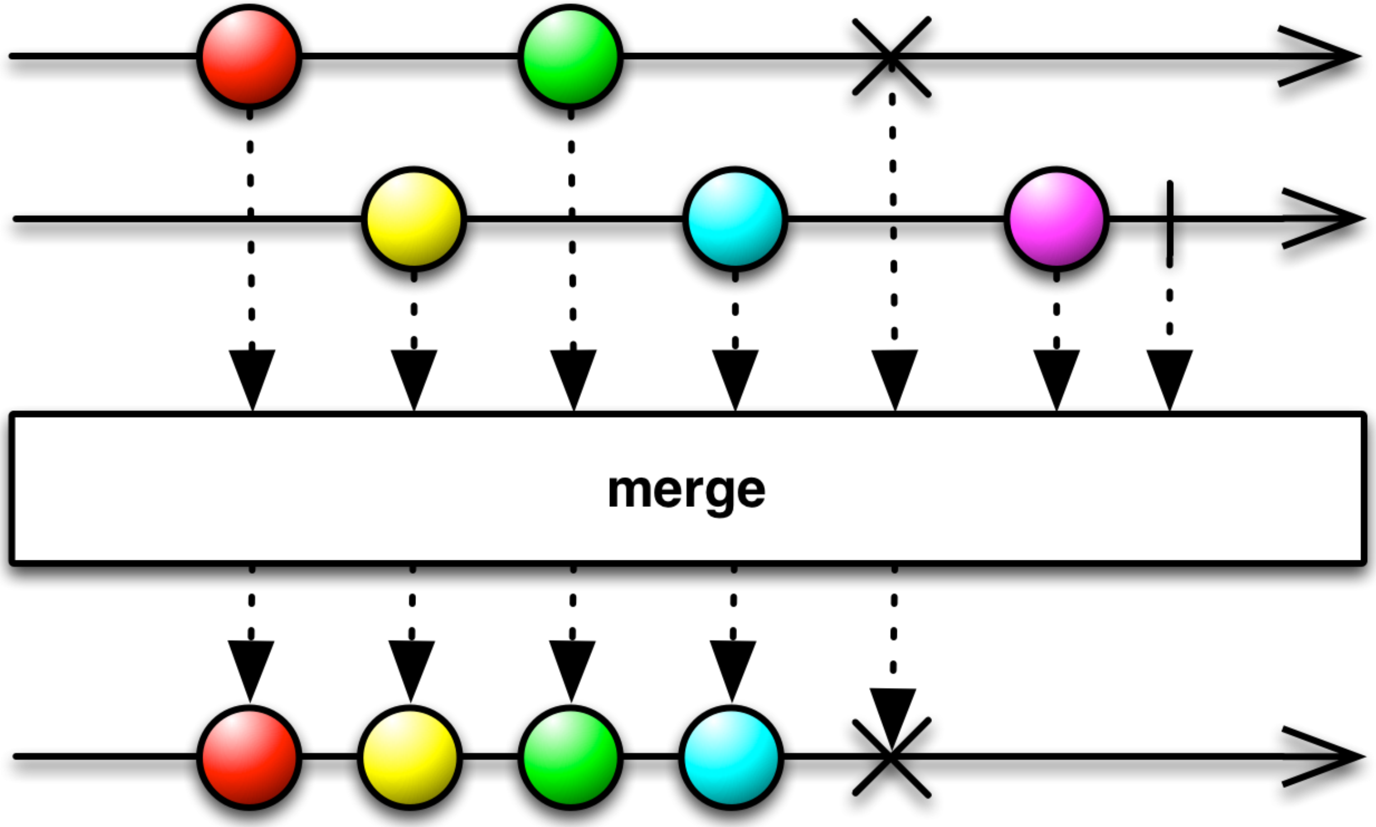
http://reactivex.io/documentation/operators/filter.html

flatMap ≡ SelectMany



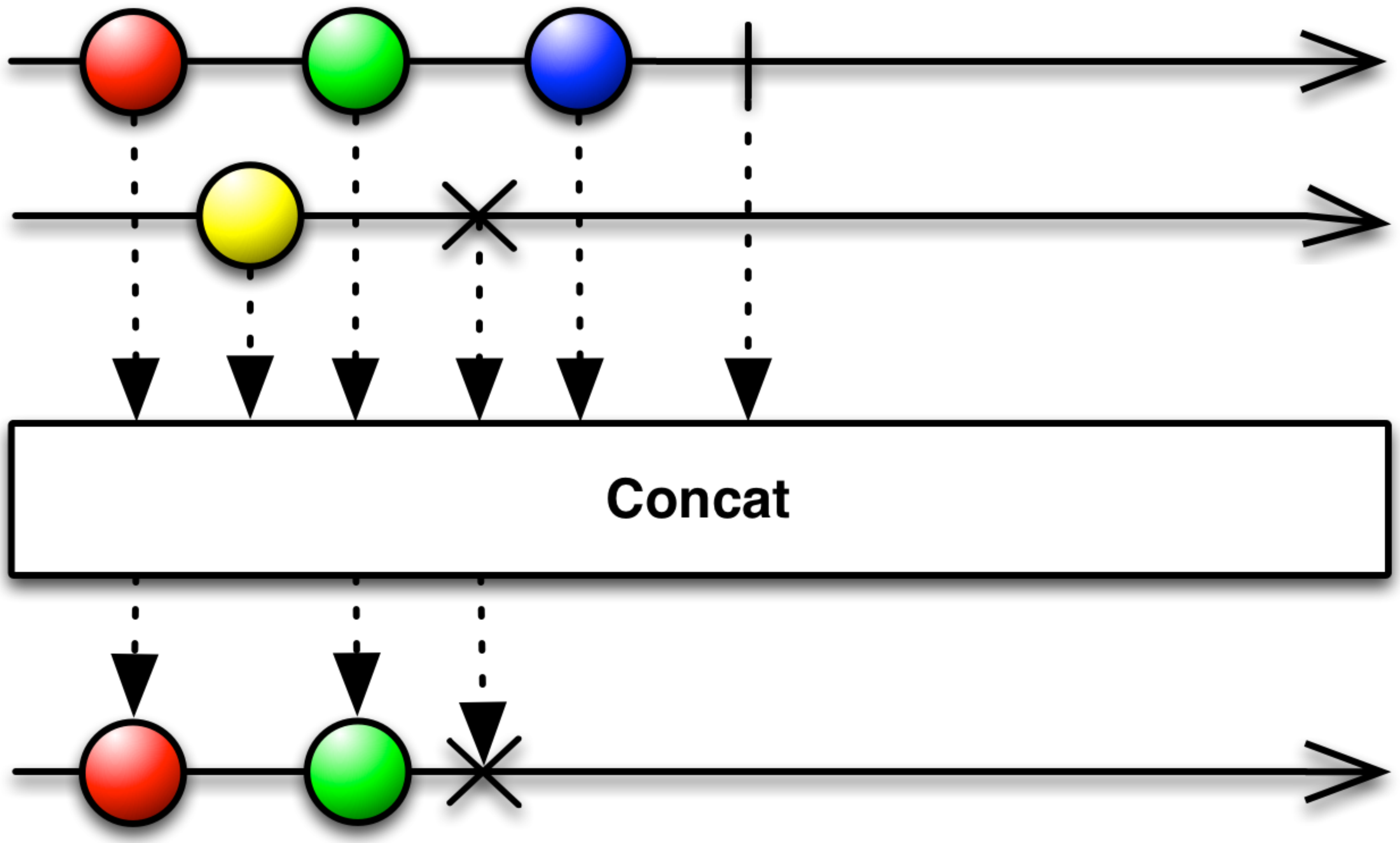
http://reactivex.io/documentation/operators/flatmap.html

merge ≡ Merge



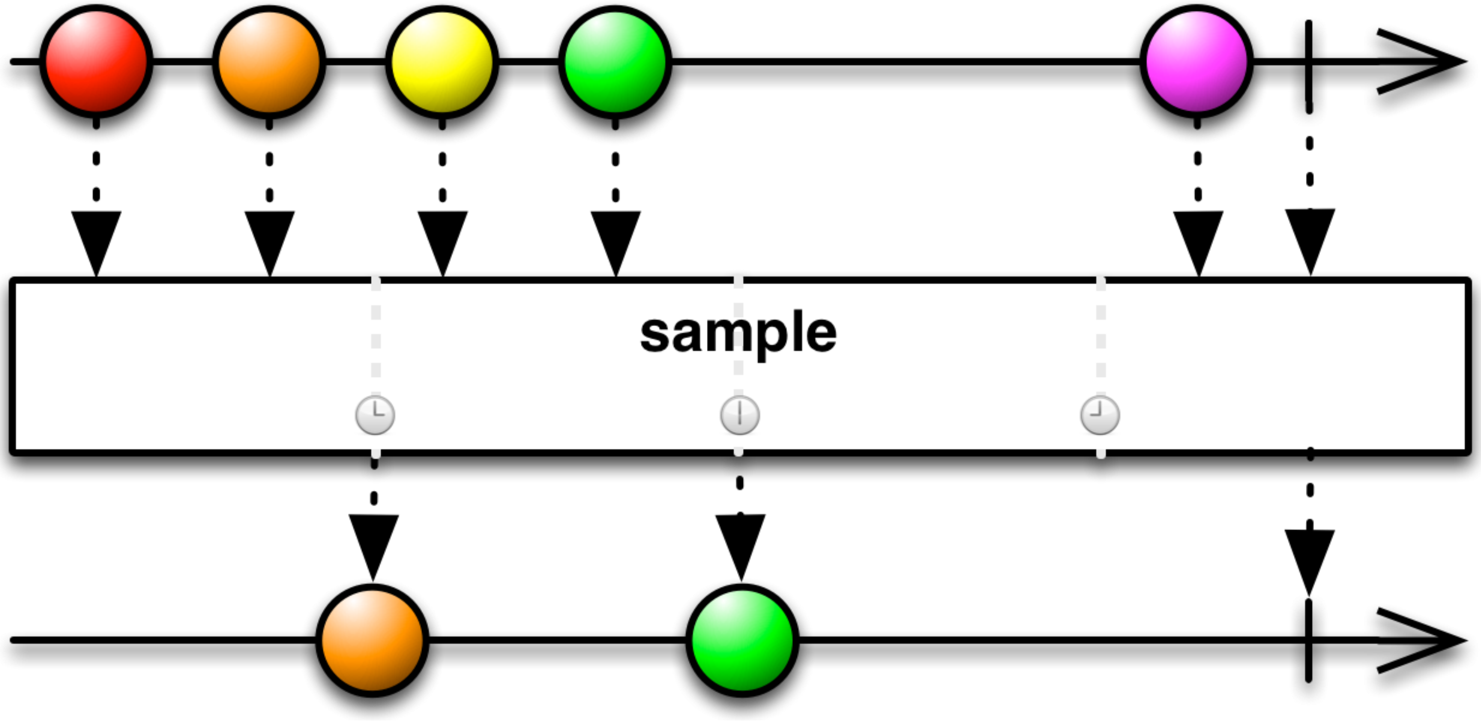
http://reactivex.io/documentation/operators/merge.html

concat ≡ Concat



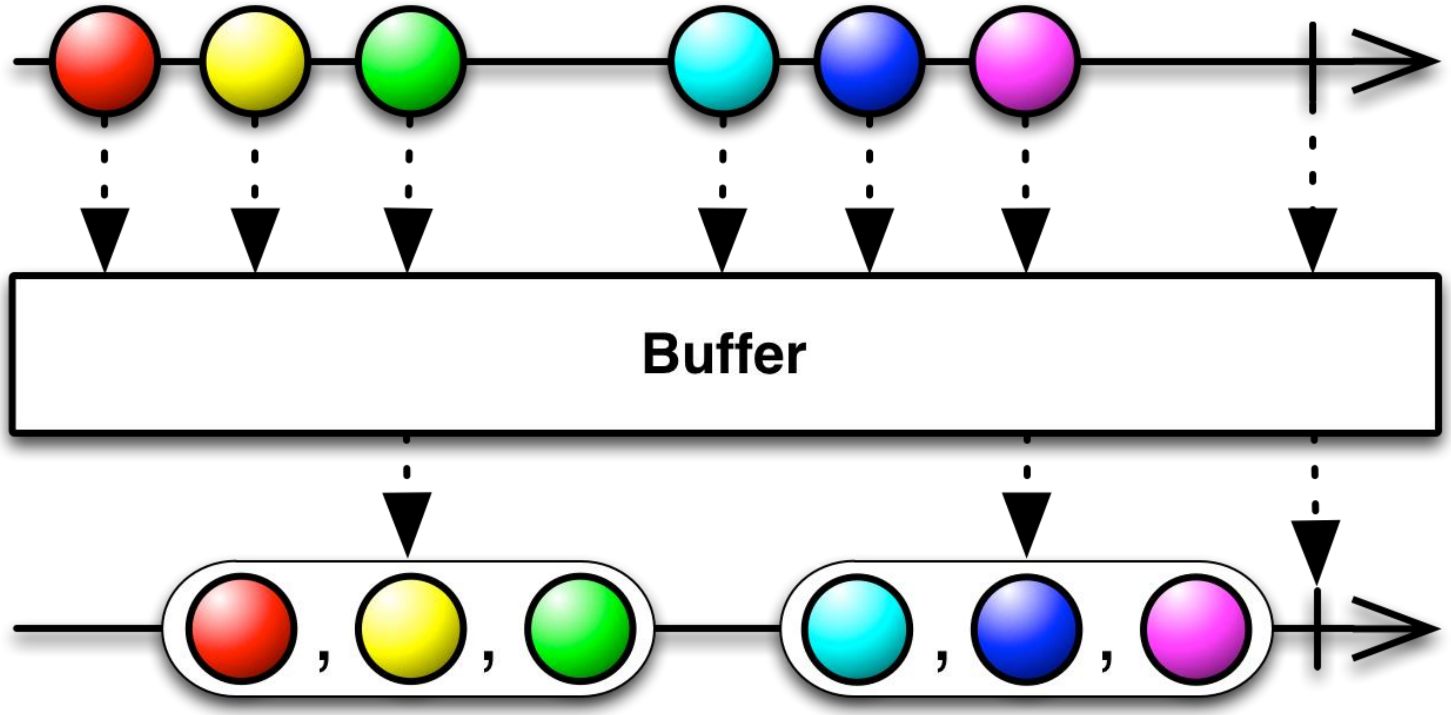
http://reactivex.io/documentation/operators/concat.html

sample ≡ Sample



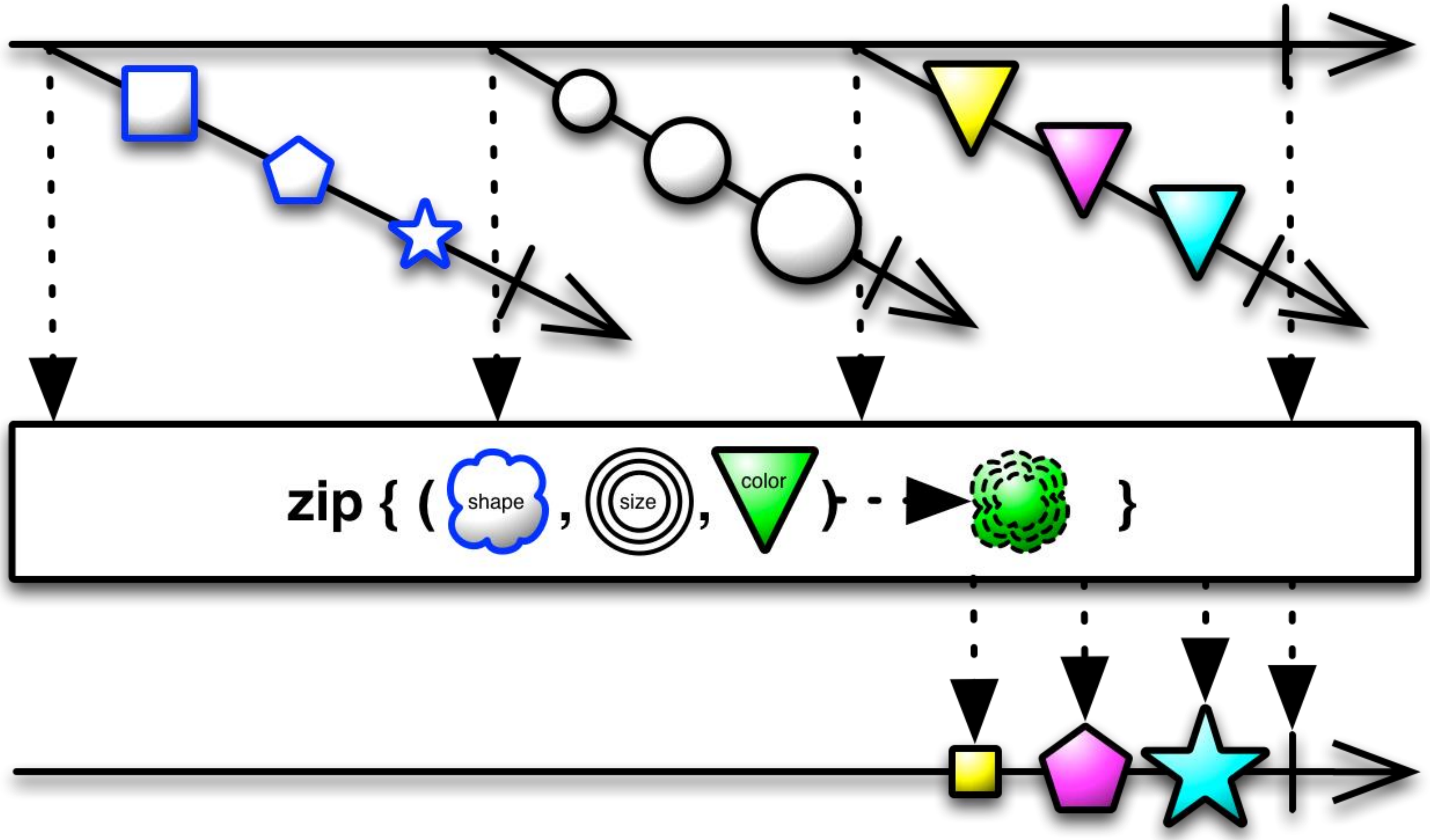
http://reactivex.io/documentation/operators/sample.html

buffer ≡ Buffer



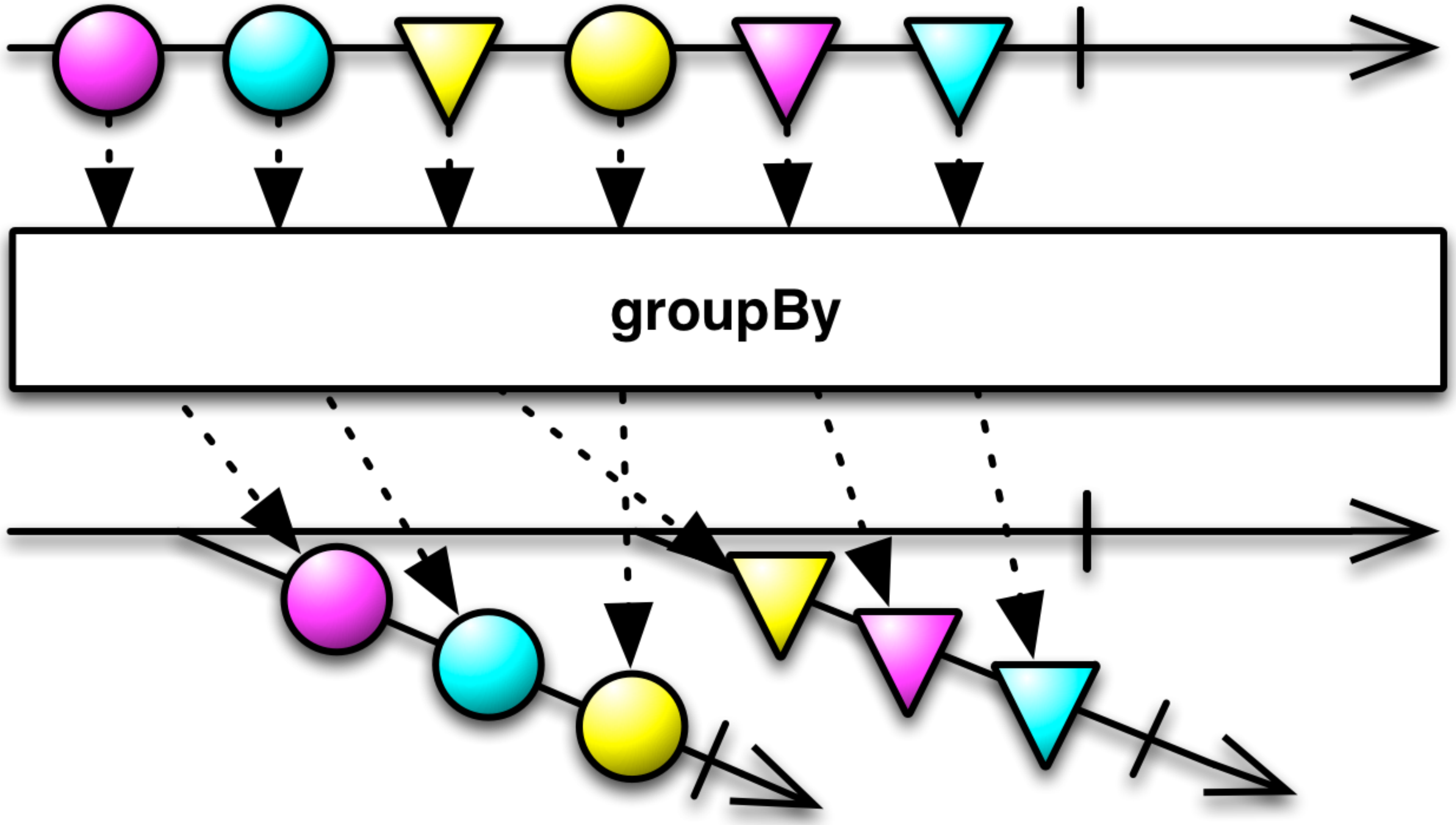
http://reactivex.io/documentation/operators/buffer.html

zip ≡ Zip



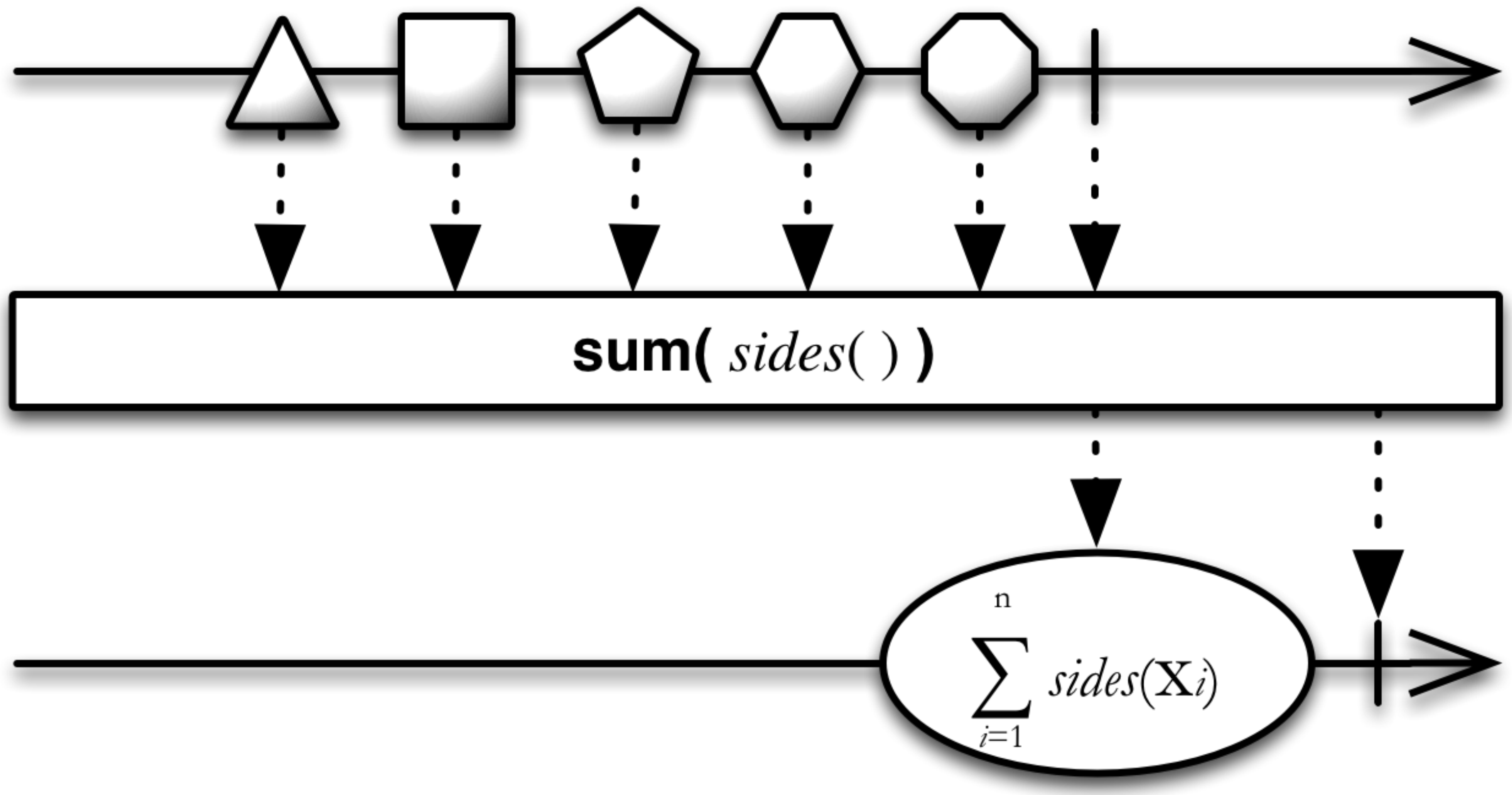
http://reactivex.io/documentation/operators/zip.html

groupBy ≡ GroupBy



http://reactivex.io/documentation/operators/groupby.html

sum ≡ Sum



http://reactivex.io/documentation/operators/sum.html

Workshop de Programação Reativa com ReactiveX

Fabio Galuppo, M.Sc.
http://member.acm.org/~fabiogaluppo
fabiogaluppo@acm.org

Junho 2015

http://bit.ly/rxworkshop