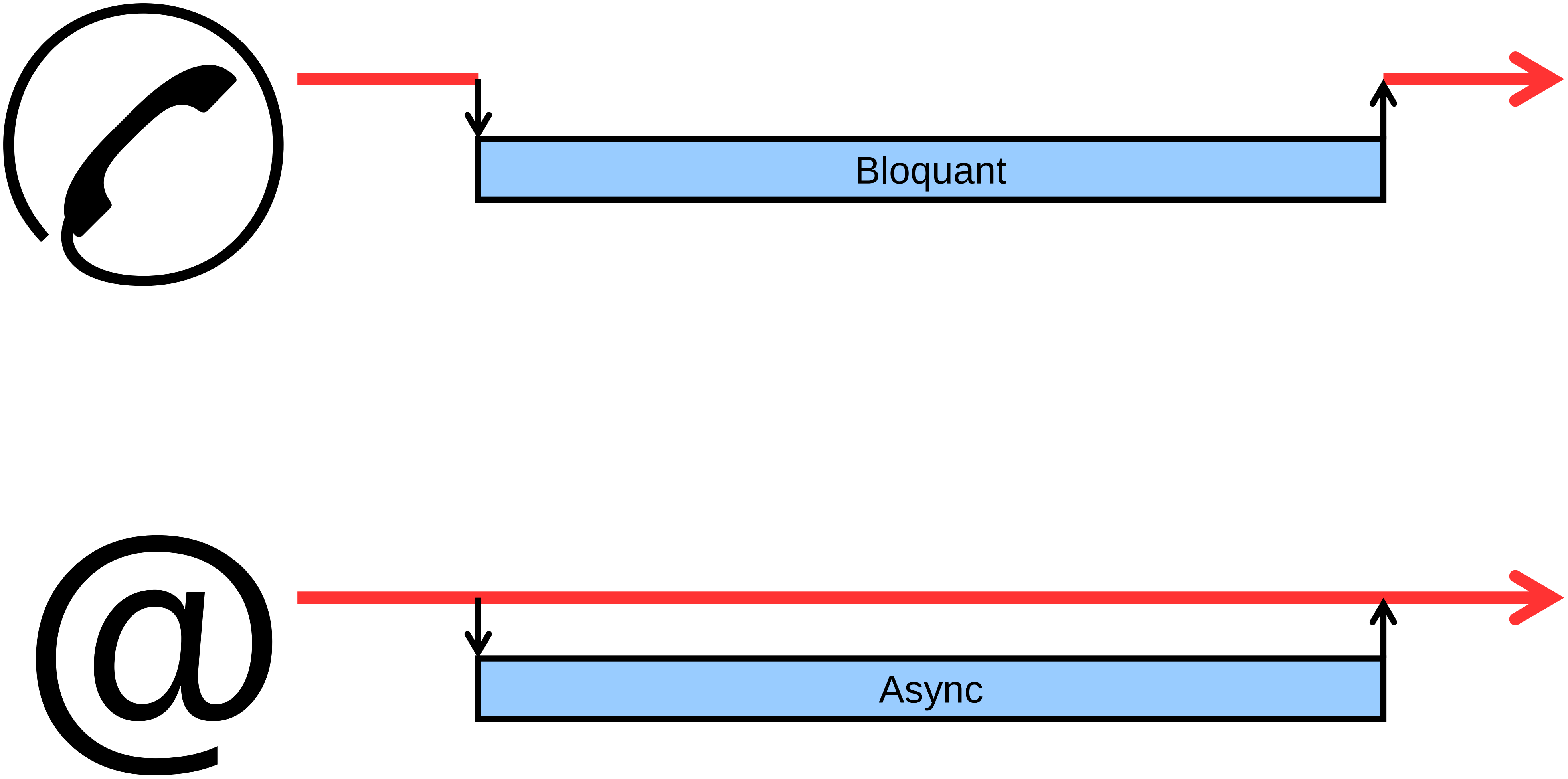
RxJava

Sommaire

- Bloquant vs Asynchrone
- Asynchrone
- Rx
- @FunctionalInterface RxJava
- Pull vs push
- Observable
- Observer
- Marble diagram
- Cold vs Hot
- ConnectableObservable
- Subject
- Scheduler
- Opérateur
- Effet de bord
- Composition
- BlockingObservable
- Backpressure
- Factorisation

Bloquant vs Asynchrone



Asynchronisme

- IHM
- IO
  - Disque
  - Réseau (de plus en plus utilisé)

Faiblesses

- Le réseau est lent.
- Le réseau n'est pas fiable.

Timings

Opération	Durée
Ping LAN	500 µs
Lire 1 Mo (disque)	2 ms
Ping WAN	150 ms

Timings à l'échelle humaine

- 1 op / s

Opération	Durée
Ping LAN	6 jours (500 µs)
Lire 1 Mo (disque)	23 jours (2 ms)
Ping WAN	5 ans (150 ms)
Requête de 1 seconde	32 ans

Threads

- Il suffit de lancer les traitements dans des Threads, et hop !

- Mais :
  - Comment récupérer les résultats ?
  - Comment gérer les erreurs ?
  - Comment faire les enchaînements ?

Callback

- API asynchrone avec deux callbacks :
  - une pour le résultat.
  - une pour l'erreur.

```
uneFonction(param1, param2,... ,
             Consumer<T> enCasDeResultat,
             Consumer<Throwable> enCasDeProblème)
```

Callback Hell

```
func1(param1, v1 -> {
  func2(v1, v2 -> {
    func3(v2, v3 -> {
      func4(v3, v4 -> {
        use(v4)
      }, ex4 -> {
      });
    }, ex3 -> {
    });
  }, ex2 -> {
  });
}, ex1 -> {
});
```

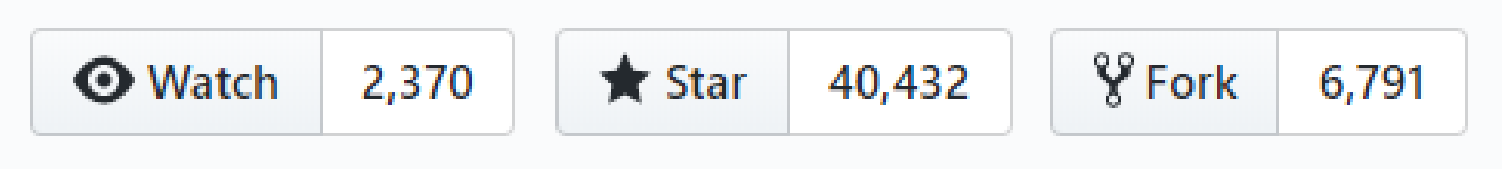
Observable

```
Observable<T> funcXYZ(paramXYZ)
```

```
func1(param1)
  .flatMap(v1 -> func2(v1))
  .flatMap(v2 -> func3(v2))
  .flatMap(v3 -> func4(v3))
  .subscribe(v4 -> use(v4),
          Ex -> ...,
          () -> ...);
```

Historique Rx

- Rx.net 2007 @ Microsoft (Erik Meijer *et al.*)
- RxJava 2011 @ Netflix (Ben Christensen *et al.*)
- Evangélisé par Jafar Husain (ex-Microsoft)
- Sur gitHub depuis 08/01/2013



- Implémentations : Java, JavaScript, C#, Scala, Clojure, C++, Ruby, Python, Groovy, Jruby, Kotlin, Swift.
- Platforms & frameworks : RxNetty, RxAndroid, RxCocoa, etc.

rx.functions

Type d'entrée	Type de sortie	Interface
A	B	Func1<A,B>
A	Boolean	Func1<A, Boolean>
A	void	Action<A>
/	B	Func0<B>
/	void	Action0

java.util.function pour RxJava 2

Type d'entrée	Type de sortie	Interface
A	B	Function<A,B>
A	boolean	Predicate<A>
A	void	Consumer<A>
/	B	Supplier<B>
/	void	Runnable

Reactive Streams

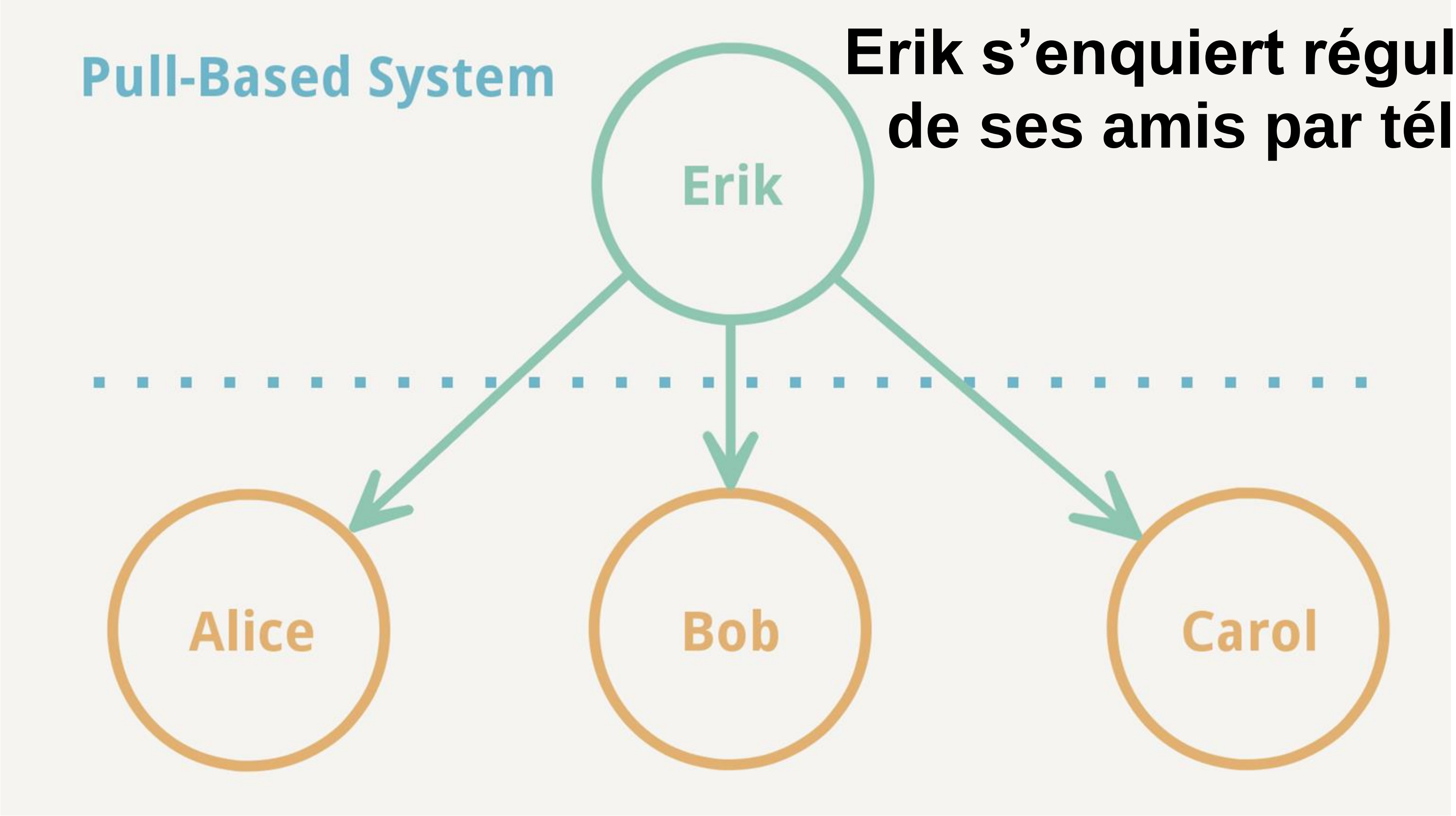
- Java 9+… : java.util.concurrent.Flow

- Akka Streams
- MongoDB
- Ratpack
- Reactive Rabbit (RabbitMQ/AMQP)
- Reactor
- **RxJava**
- Slick 3.0
- Vert.x 3.0

Rx (features)

- Nombre de valeurs :
  - Zero
  - N
  - Infinit
- Évaluation paresseuse.
- Synchrone / Asynchrone.
- Annulable.
- Gestion des erreurs.

Pull vs Push



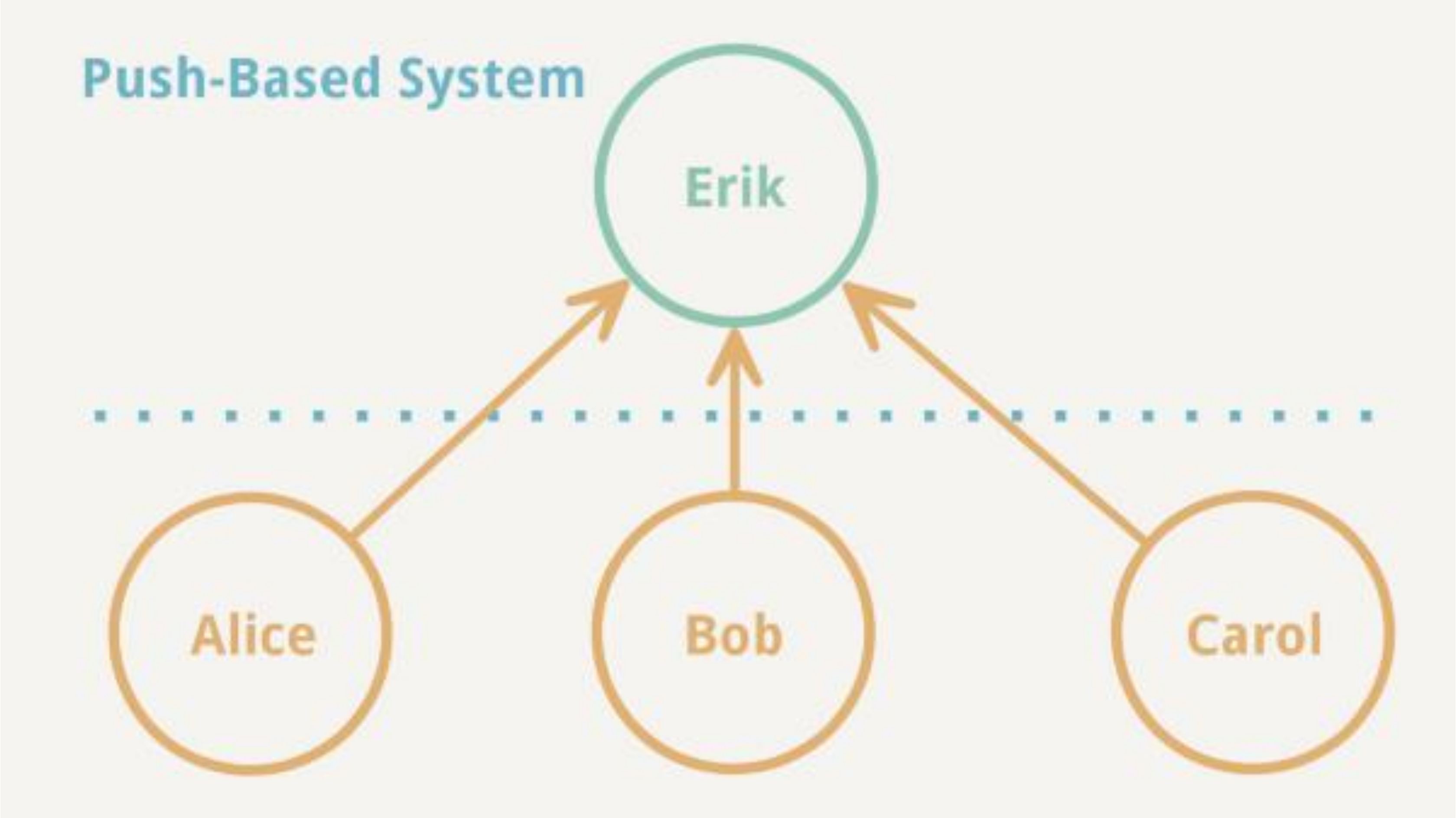
Erik s’enquiert régulièrement de ses amis par téléphone

Rien de nouveau

Pas de réponse

Blah blah blah blah blah blah blah blah blah blah blah blah

Pull vs Push



Les amis d'Erik l'informent des nouvelles

Pull vs Push

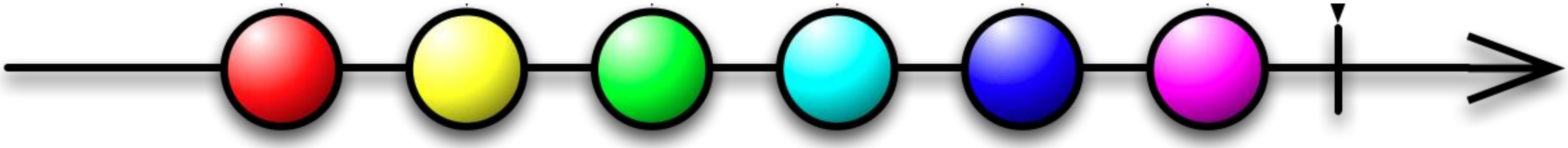
	Synchrone / Pull	Asynchrone / Push
1 valeur	function(p1, p2, …)	**Single<T>**
N valeurs	Iterable<T>	**Observable<T>**

Pull vs Push

	Synchrone / Pull	Asynchrone / Push
Résultat	T next()	**onNext(T)**
Exception	T next() throws Exception	**onError(Exception)**
Fin	If (!hasNext()) {...}	**onCompleted()**

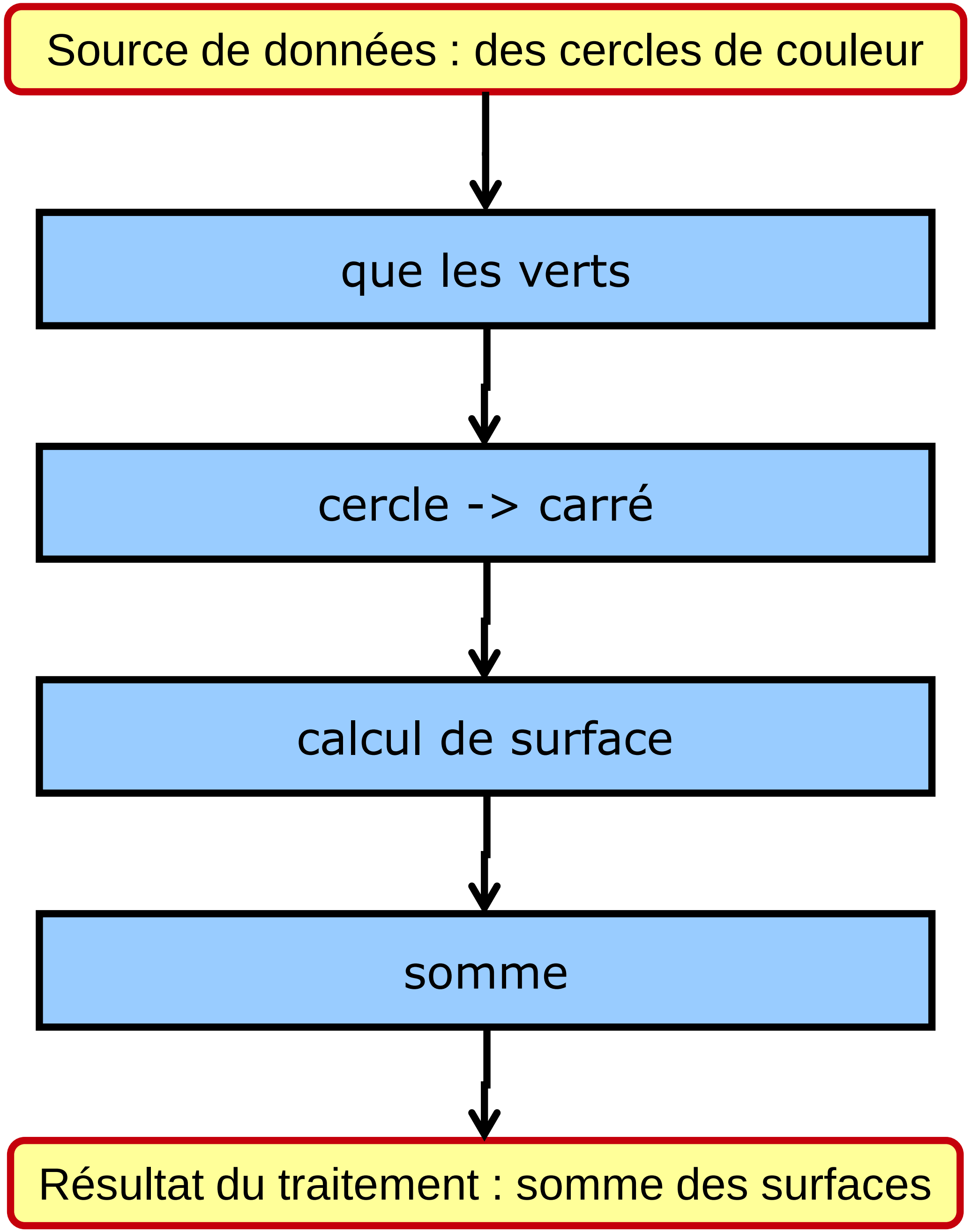
Observable

onNext* (onError | onCompleted)?



Observer<T>

```
interface Observer<T> {
  void onNext(T value);
  void onError(Throwable error);
  void onCompleted();
}
```



Source de données : des cercles de couleur

que les verts

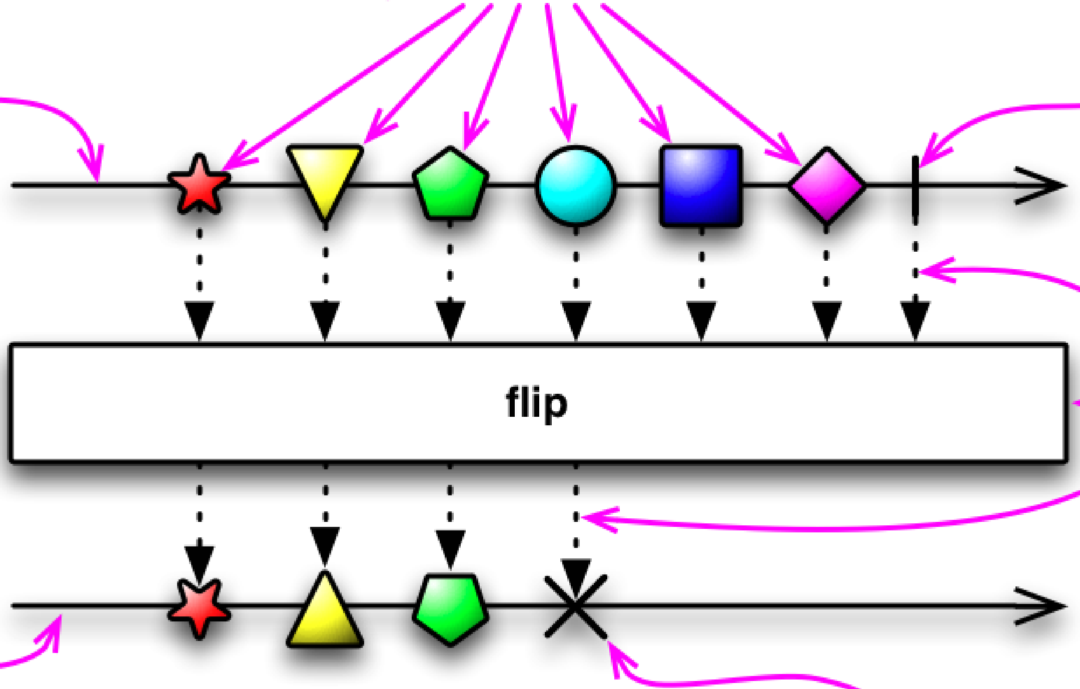
cercle -> carré

calcul de surface

somme

Résultat du traitement : somme des surfaces

Marble diagram



This is the timeline of the Observable. Time flows from left to right.

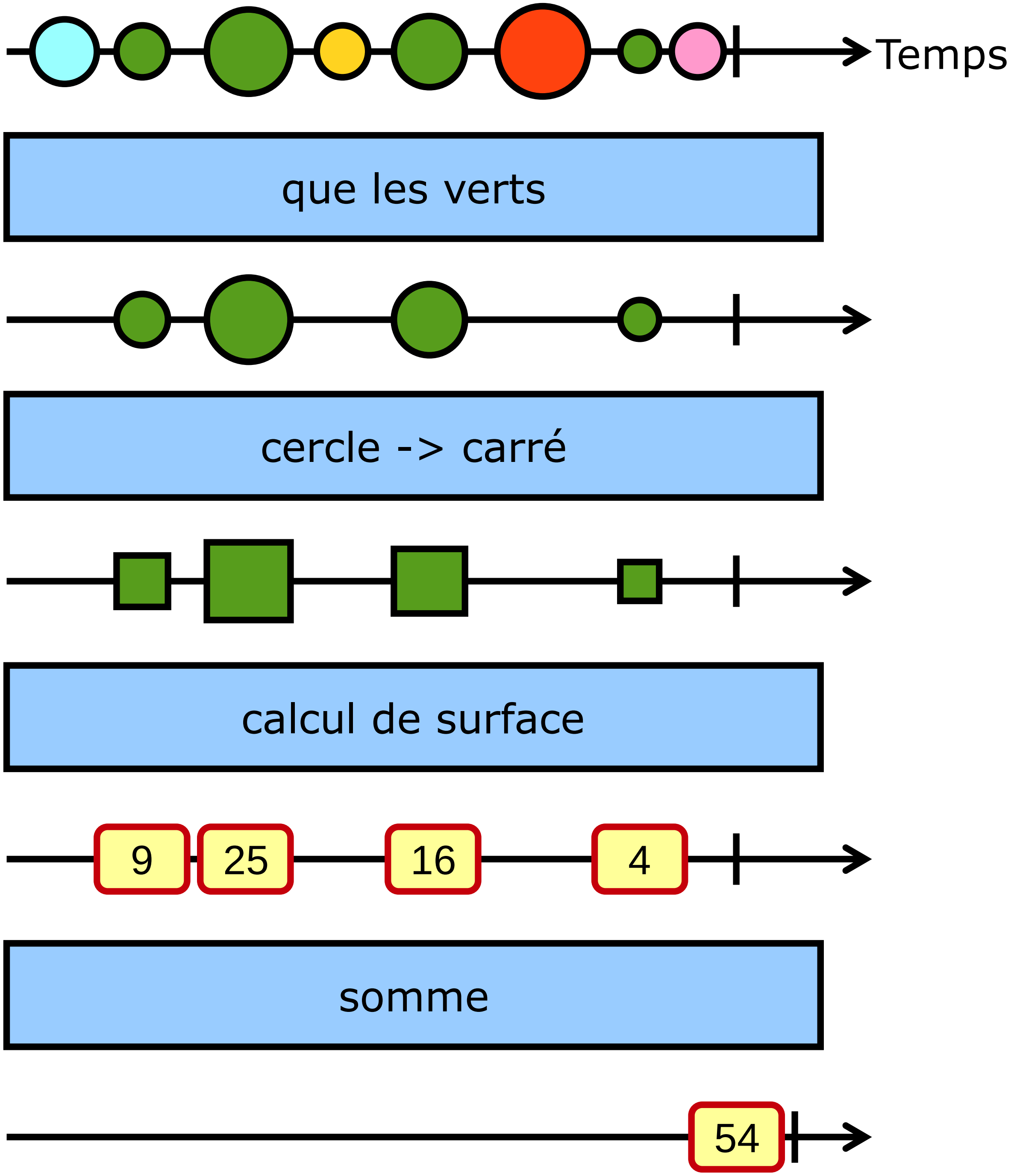
These are items emitted by the Observable.

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.

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.


Temps
que les verts
cercle -> carré
calcul de surface
9
25
16
4
somme
54

Create

```
Observable<T> create(Action1<Subscriber<T>> f);
```

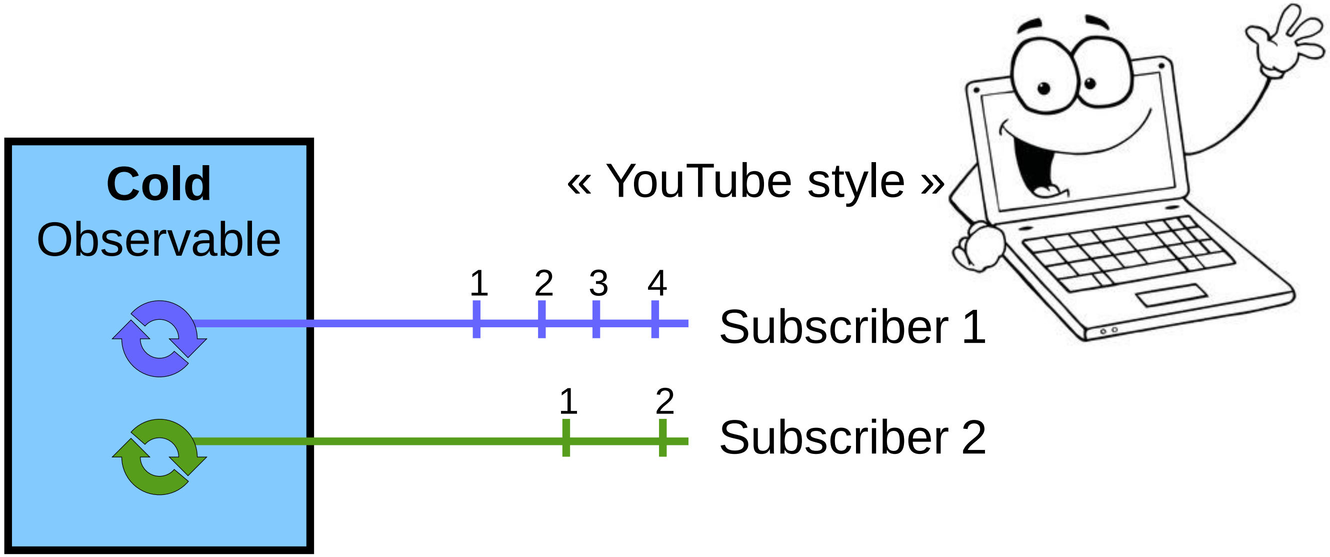
```
class Subscriber<T> implements Observer<T>, Subscription {...}
```

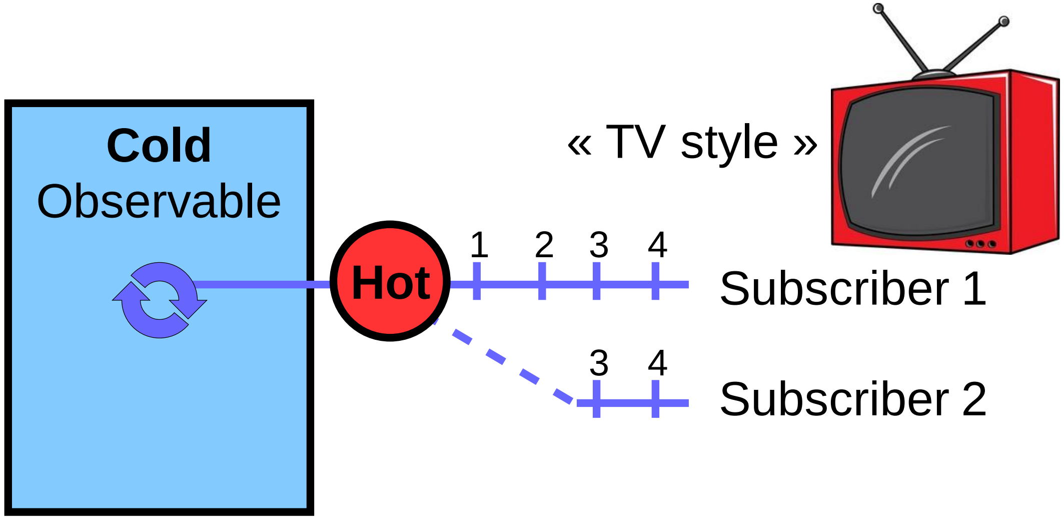
```
Observable.create((Subscriber pusher) -> {
  for (int i = 0; i < 5 && !pusher.isUnsubscribed(); i++) {
    pusher.onNext(i);
  }
  pusher.onCompleted();
});
```

Subscription (annulable)

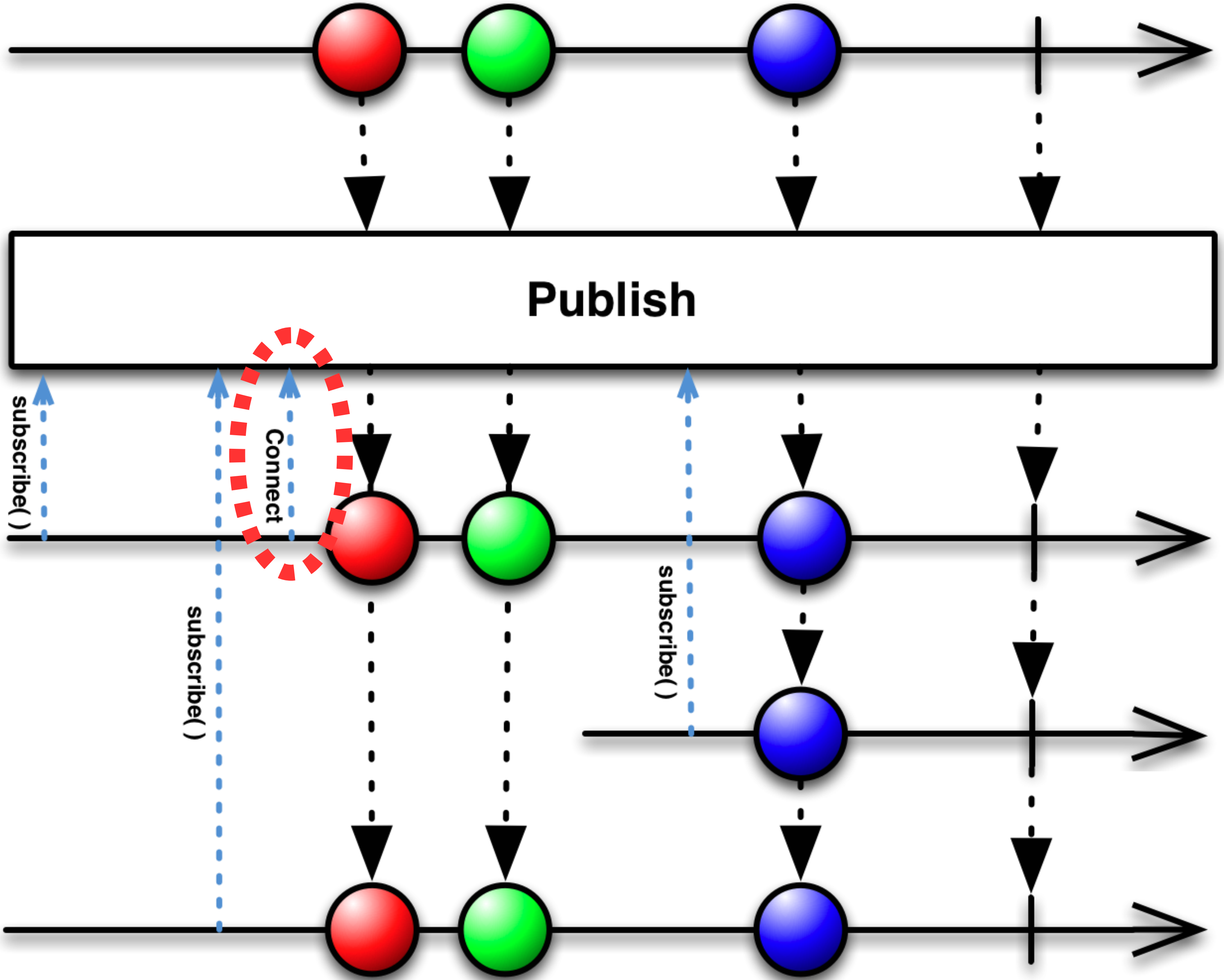
```
interface Subscription {
  void unsubscribe();
  boolean isUnsubscribed();
}
```

Cold vs Hot

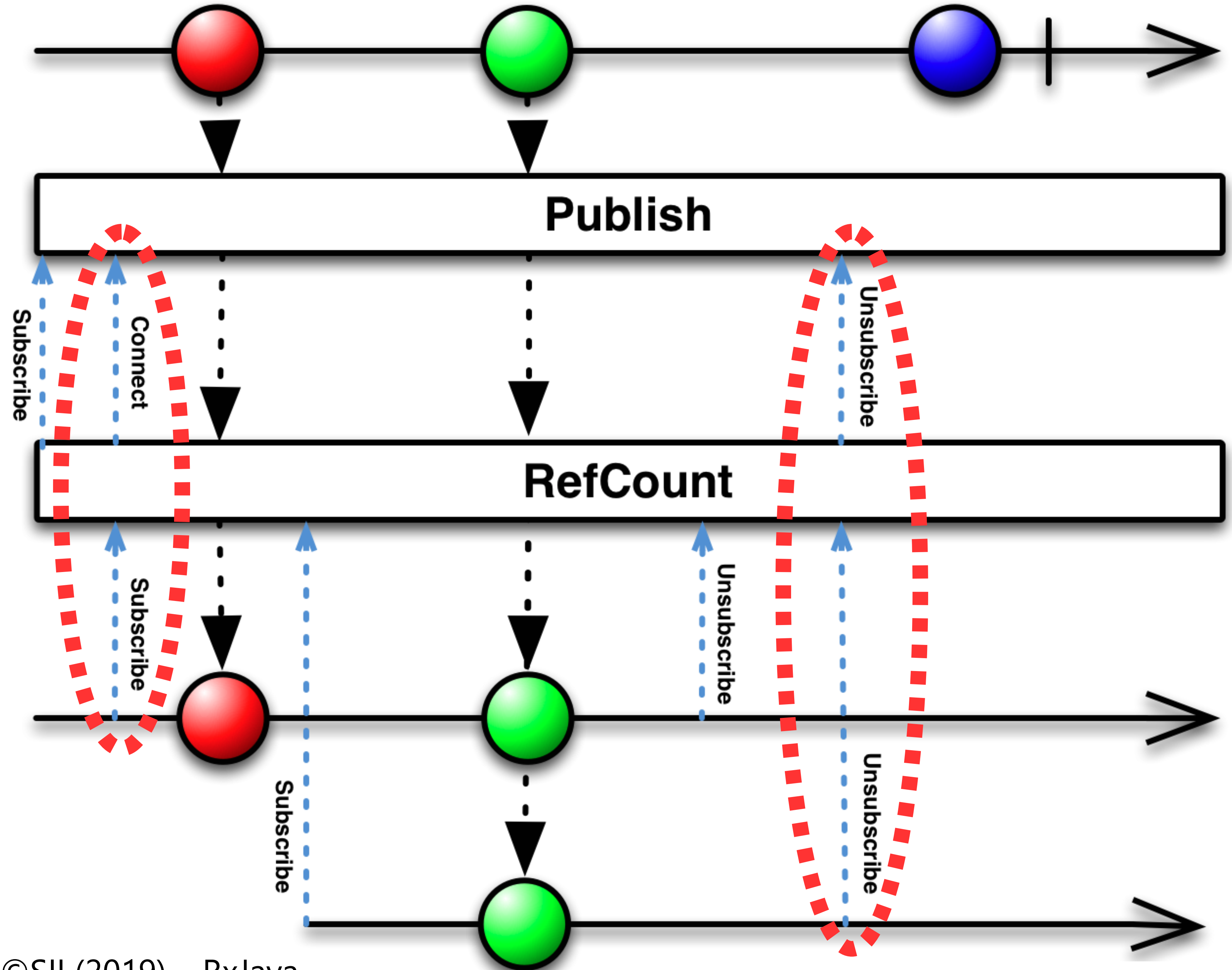

Cold
Observable
« YouTube style »
1 2 3 4
Subscriber 1
1 2
Subscriber 2


Cold
Observable
« TV style »
Hot
1 2 3 4
Subscriber 1
3 4
Subscriber 2

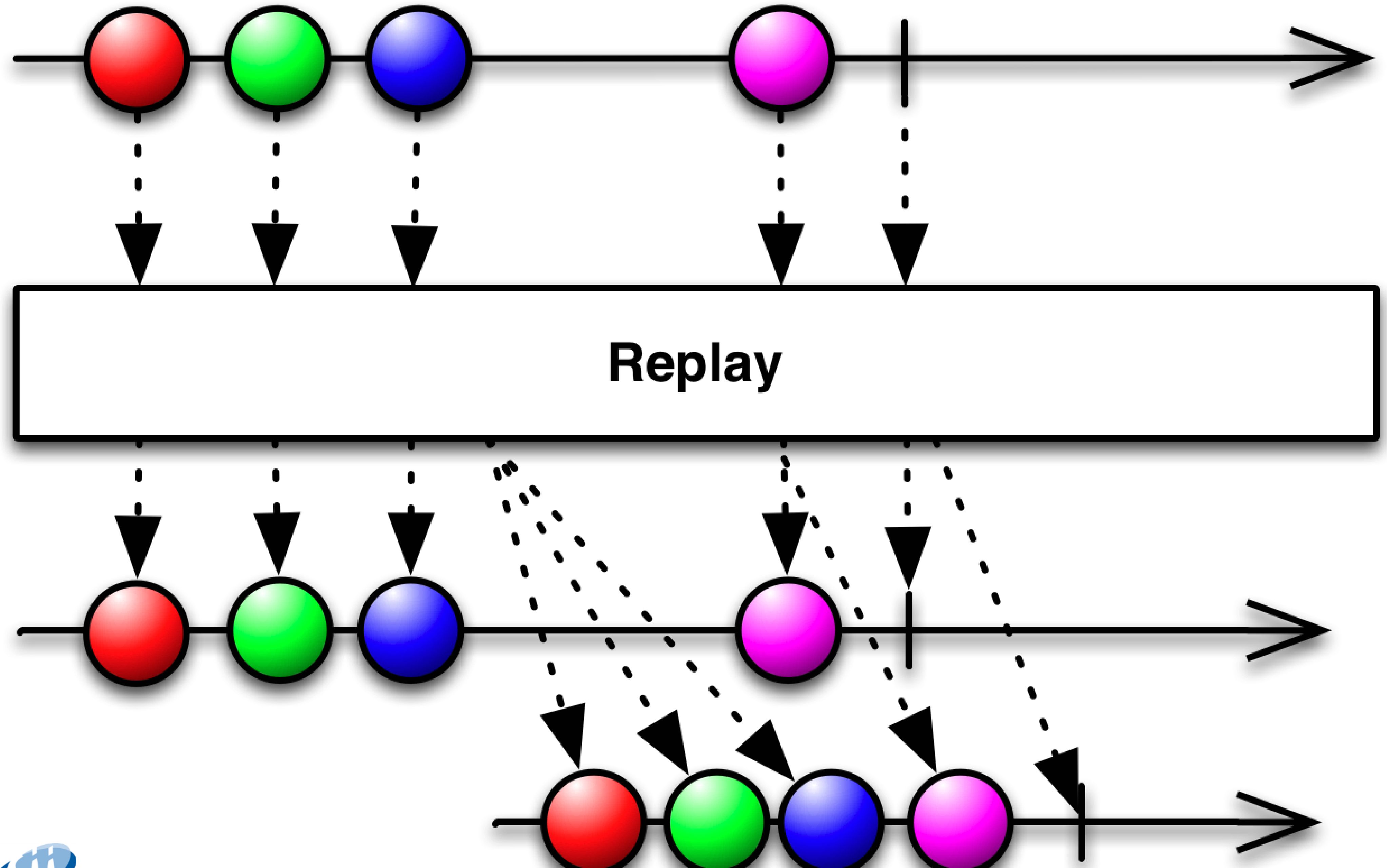
ConnectableObservable



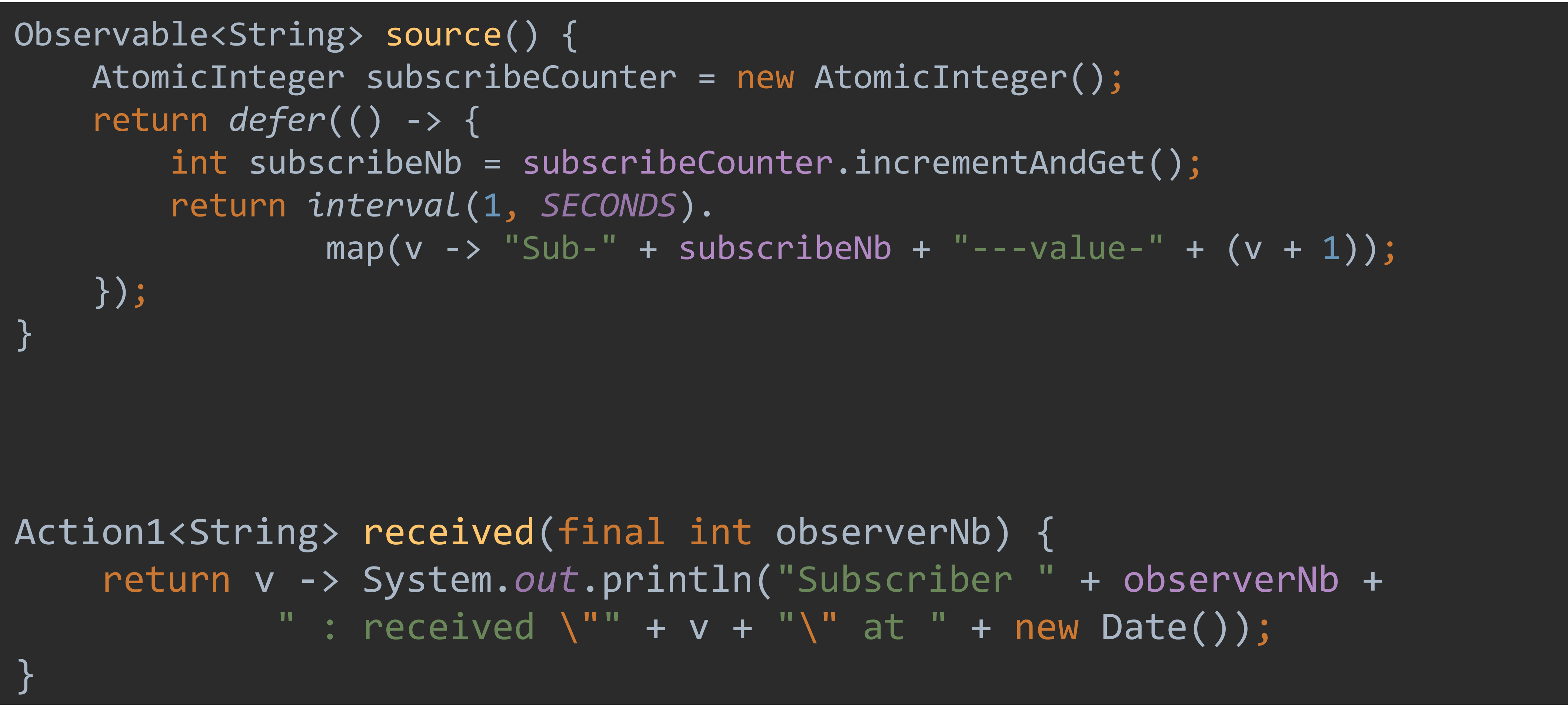
refCount


Publish
Subscribe
Connect
Unsubscribe
RefCount
Subscribe
Subscribe
Unsubscribe
Unsubscribe

ConnectableObservable

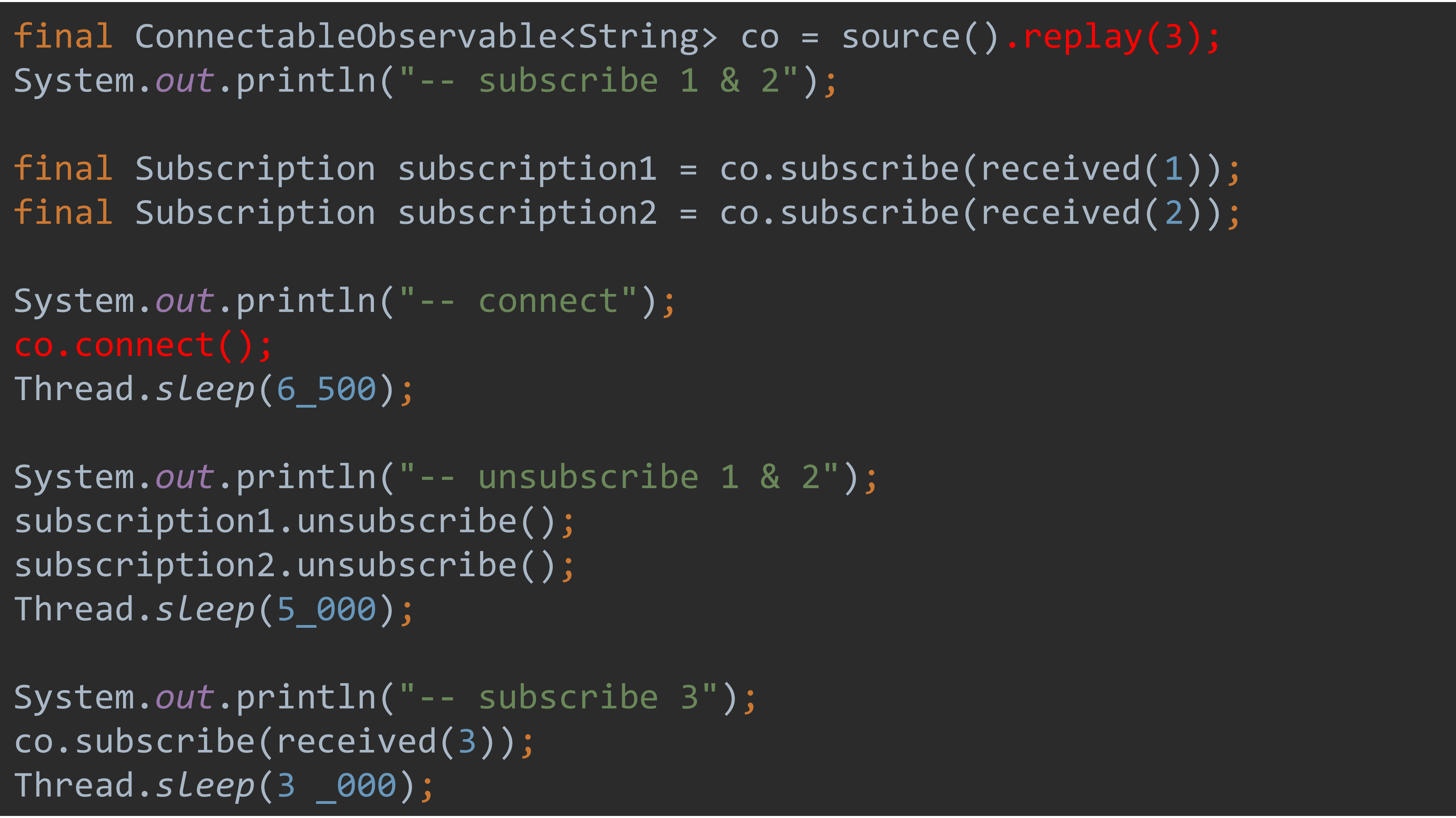


Connect vs refCount



```
Observable<String> source() {
    AtomicInteger subscribeCounter = new AtomicInteger();
    return defer(() -> {
        int subscribeNb = subscribeCounter.incrementAndGet();
        return interval(1, SECONDS).
                map(v -> "Sub-" + subscribeNb + "---value-" + (v + 1));
    });
}




Action1<String> received(final int observerNb) {
    return v -> System.out.println("Subscriber " + observerNb +
            " : received \"" + v + "\" at " + new Date());
}
```

Connect vs refCount



```
final ConnectableObservable<String> co = source().replay(3);
System.out.println("-- subscribe 1 & 2");

final Subscription subscription1 = co.subscribe(received(1));
final Subscription subscription2 = co.subscribe(received(2));

System.out.println("-- connect");
co.connect();
Thread.sleep(6_500);

System.out.println("-- unsubscribe 1 & 2");
subscription1.unsubscribe();
subscription2.unsubscribe();
Thread.sleep(5_000);

System.out.println("-- subscribe 3");
co.subscribe(received(3));
Thread.sleep(3 _000);
```

Connect vs refCount

```
-- subscribe 1 & 2
-- connect
Subscriber 1 : received "Sub-1---value-1" at 2019-09-11T07:15:33.263Z
Subscriber 2 : received "Sub-1---value-1" at 2019-09-11T07:15:33.305Z
Subscriber 1 : received "Sub-1---value-2" at 2019-09-11T07:15:34.256Z
Subscriber 2 : received "Sub-1---value-2" at 2019-09-11T07:15:34.256Z
Subscriber 1 : received "Sub-1---value-3" at 2019-09-11T07:15:35.254Z
Subscriber 2 : received "Sub-1---value-3" at 2019-09-11T07:15:35.255Z
Subscriber 1 : received "Sub-1---value-4" at 2019-09-11T07:15:36.255Z
Subscriber 2 : received "Sub-1---value-4" at 2019-09-11T07:15:36.255Z
Subscriber 1 : received "Sub-1---value-5" at 2019-09-11T07:15:37.254Z
Subscriber 2 : received "Sub-1---value-5" at 2019-09-11T07:15:37.254Z
Subscriber 1 : received "Sub-1---value-6" at 2019-09-11T07:15:38.257Z
Subscriber 2 : received "Sub-1---value-6" at 2019-09-11T07:15:38.257Z
-- unsubscribe 1 & 2
-- subscribe 3
Subscriber 3 : received "Sub-1---value-9" at 2019-09-11T07:15:43.759Z
Subscriber 3 : received "Sub-1---value-10" at 2019-09-11T07:15:43.759Z
Subscriber 3 : received "Sub-1---value-11" at 2019-09-11T07:15:43.760Z
```

Connect vs refCount

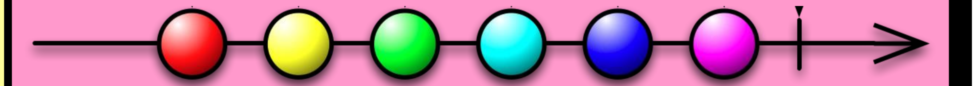
```
final Observable<String> co= source().replay(3).refCount();
System.out.println("-- subscribe 1 & 2");

final Subscription subscription1 = co.subscribe(received(1));
final Subscription subscription2 = co.subscribe(received(2));

System.out.println("-- connect");
Thread.sleep(6_500);

System.out.println("-- unsubscribe 1 & 2");
subscription1.unsubscribe();
subscription2.unsubscribe();
Thread.sleep(5_000);

System.out.println("-- subscribe 3");
co.subscribe(received(3));
Thread.sleep(3 _000);
```

Connect vs refCount

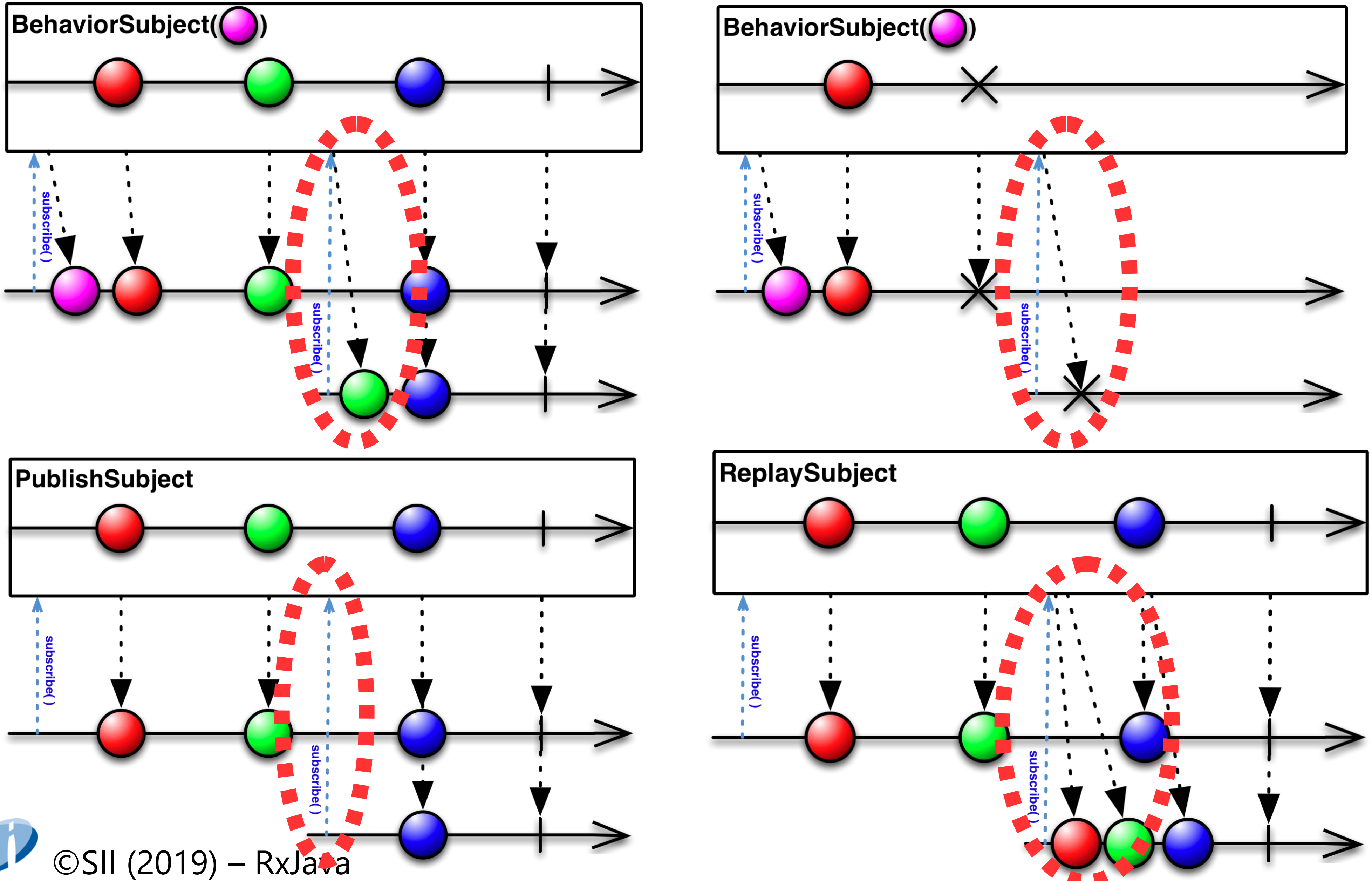
```
-- subscribe 1 & 2
-- connect
Subscriber 1 : received "Sub-1---value-1" at 2019-09-11T07:12:56.677Z
Subscriber 2 : received "Sub-1---value-1" at 2019-09-11T07:12:56.725Z
Subscriber 1 : received "Sub-1---value-2" at 2019-09-11T07:12:57.669Z
Subscriber 2 : received "Sub-1---value-2" at 2019-09-11T07:12:57.669Z
Subscriber 1 : received "Sub-1---value-3" at 2019-09-11T07:12:58.669Z
Subscriber 2 : received "Sub-1---value-3" at 2019-09-11T07:12:58.669Z
Subscriber 1 : received "Sub-1---value-4" at 2019-09-11T07:12:59.668Z
Subscriber 2 : received "Sub-1---value-4" at 2019-09-11T07:12:59.668Z
Subscriber 1 : received "Sub-1---value-5" at 2019-09-11T07:13:00.671Z
Subscriber 2 : received "Sub-1---value-5" at 2019-09-11T07:13:00.671Z
Subscriber 1 : received "Sub-1---value-6" at 2019-09-11T07:13:01.669Z
Subscriber 2 : received "Sub-1---value-6" at 2019-09-11T07:13:01.669Z
-- unsubscribe 1 & 2
-- subscribe 3
Subscriber 3 : received "Sub-2---value-1" at 2019-09-11T07:13:08.175Z
Subscriber 3 : received "Sub-2---value-2" at 2019-09-11T07:13:09.174Z
Subscriber 3 : received "Sub-2---value-3" at 2019-09-11T07:13:10.174Z
```

Subject<T>

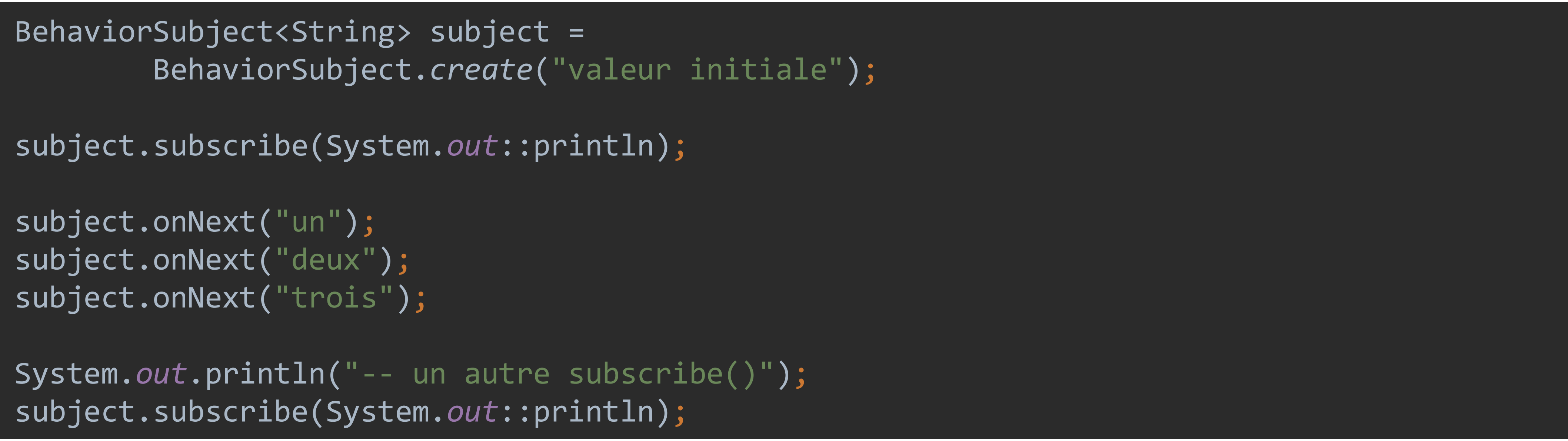
Observer<T>	Observable<T>

.onNext(T)
.onError(Throwable)
.onCompleted()
	Hot
	

Subject<T>


BehaviorSubject( )
subscribe( )
subscribe( )
BehaviorSubject( )
subscribe( )
subscribe( )
PublishSubject
subscribe( )
subscribe( )
ReplaySubject
subscribe( )
subscribe( )

Subject<T>



```java
BehaviorSubject<String> subject =
        BehaviorSubject.create("valeur initiale");

subject.subscribe(System.out::println);

subject.onNext("un");
subject.onNext("deux");
subject.onNext("trois");

System.out.println("-- un autre subscribe()");
subject.subscribe(System.out::println);
```

```
valeur initiale
un
deux
trois
-- un autre subscribe()
trois
```

SerializedSubject<T>

Lorsqu'on utilise un **Subject** on doit s'assurer de ne pas appeler ses méthodes (**onNext**, **onError** et **onCompleted**) depuis plusieurs **Thread**s.

Pour protéger le **Subject** du danger on peut le convertir en **SerializedSubject** :

```
safeSubject = new SerializedSubject(unsafeSubject);
```

```
safeSubject = unsafeSubject.toSerialized();
```

Schedulers

.Computation() : pool de Threads (nb CPU-Cores).

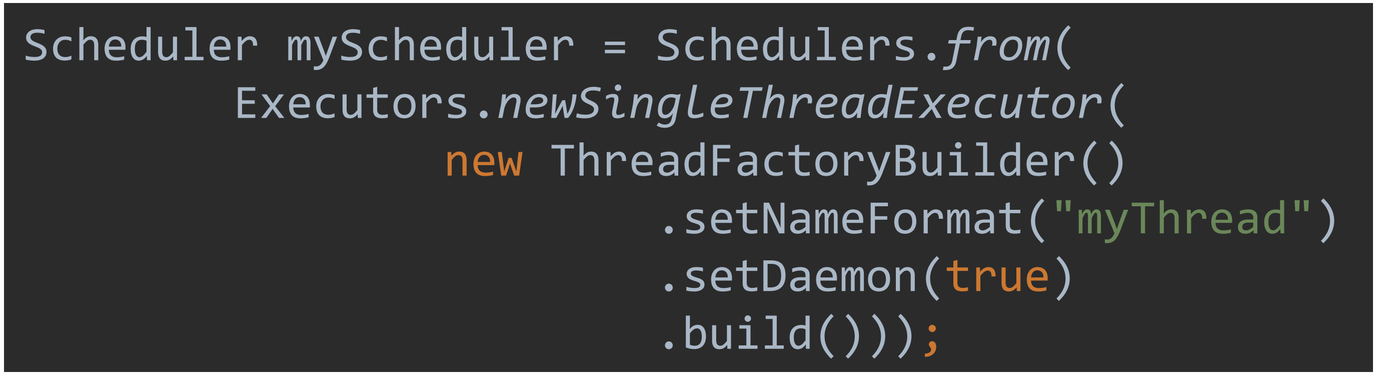
.immediate() : immédiatement sur le Thread courant.

.io() : pool de Threads pouvant grandir au besoin. Utile pour des opérations asynchrones sur des I/O bloquantes.

.newThread() : création d'un nouveau Thread pour chaque traitement programmé.

.trampoline() : ajoute un traitement dans la file et sera exécuté sur le Thread courant après tout les traitements déjà dans la file.

.from(executor) : utilise en Executor Java.

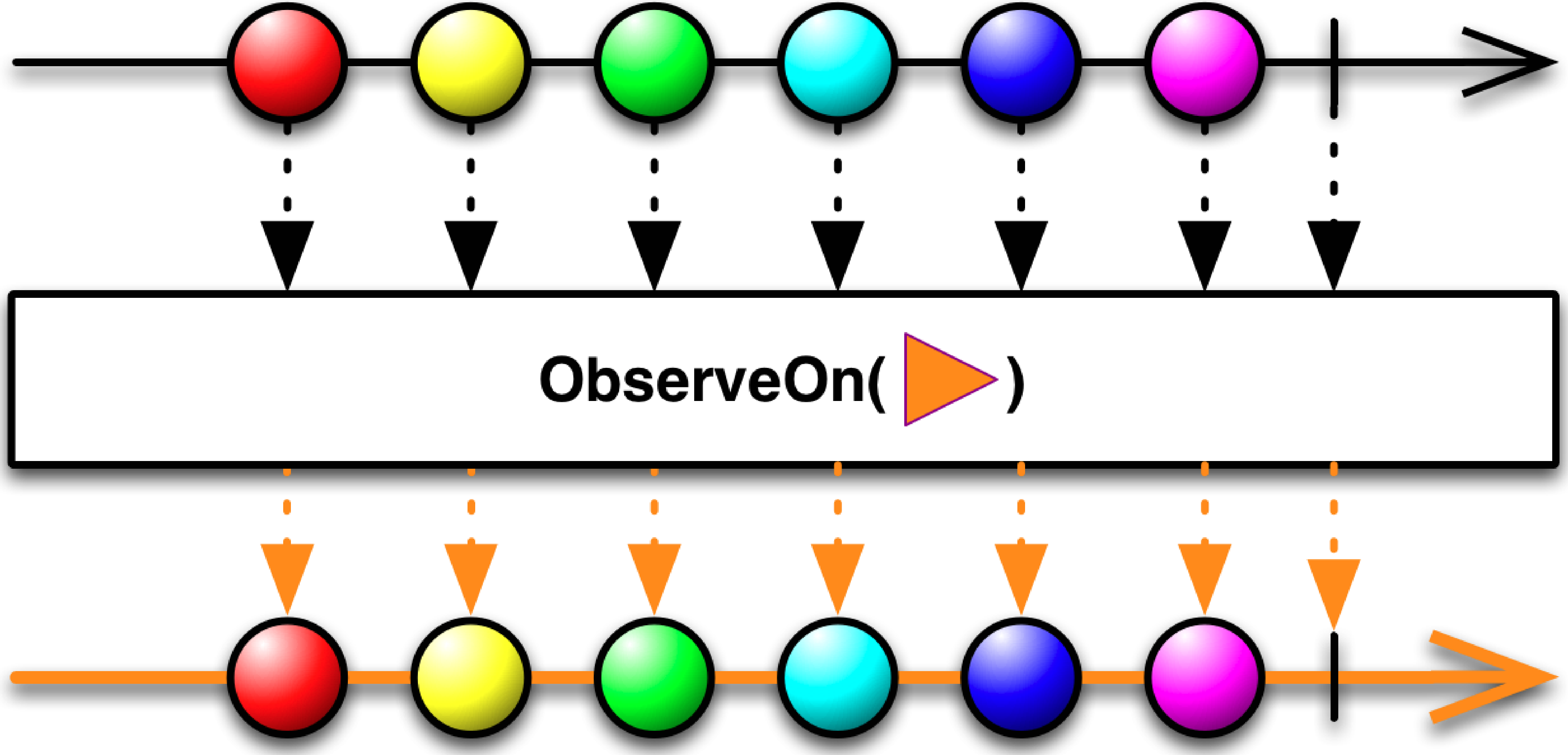


```
Scheduler myScheduler = Schedulers.from(
        Executors.newSingleThreadExecutor(
                new ThreadFactoryBuilder()
                        .setNameFormat("myThread")
                        .setDaemon(true)
                        .build()));
```

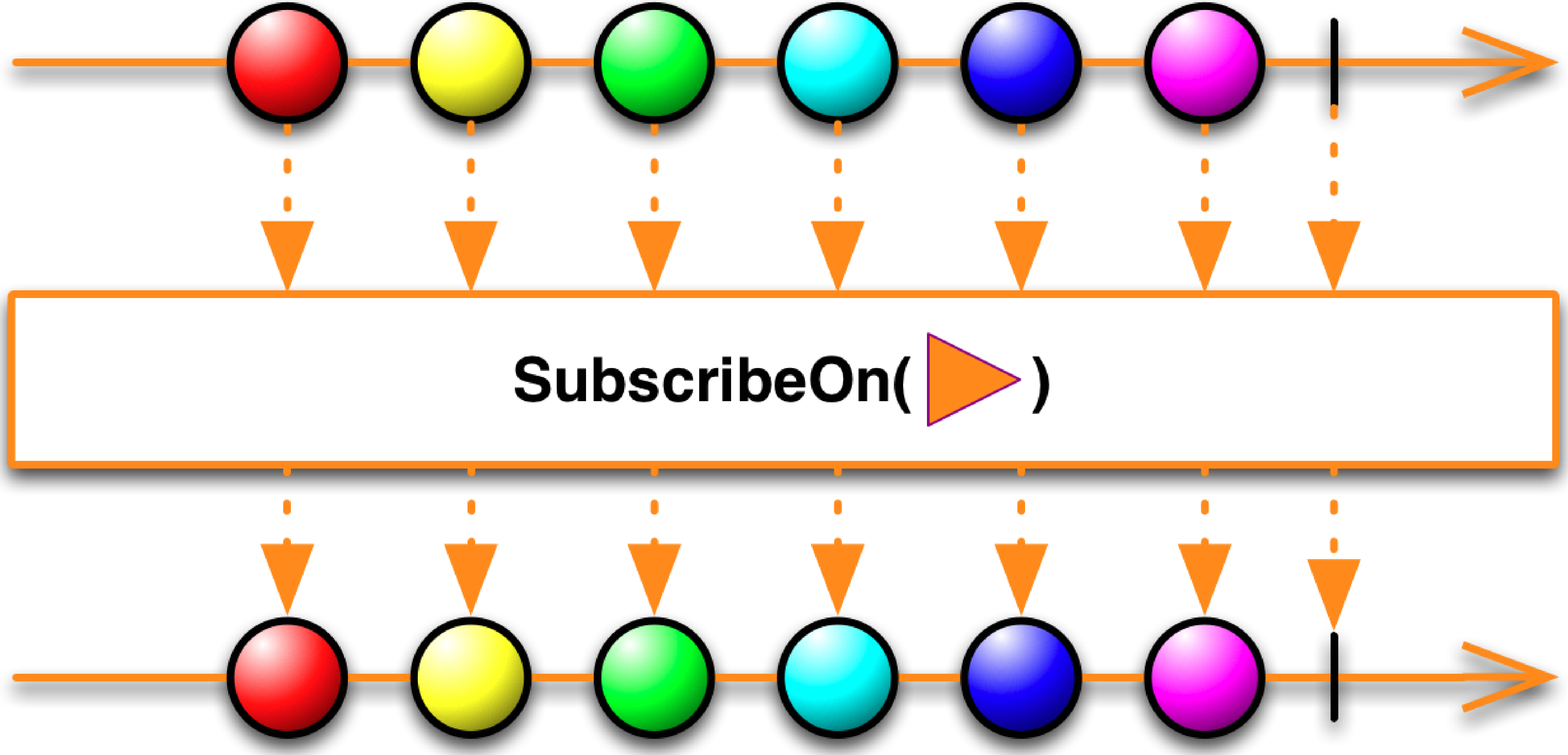
Schedulers

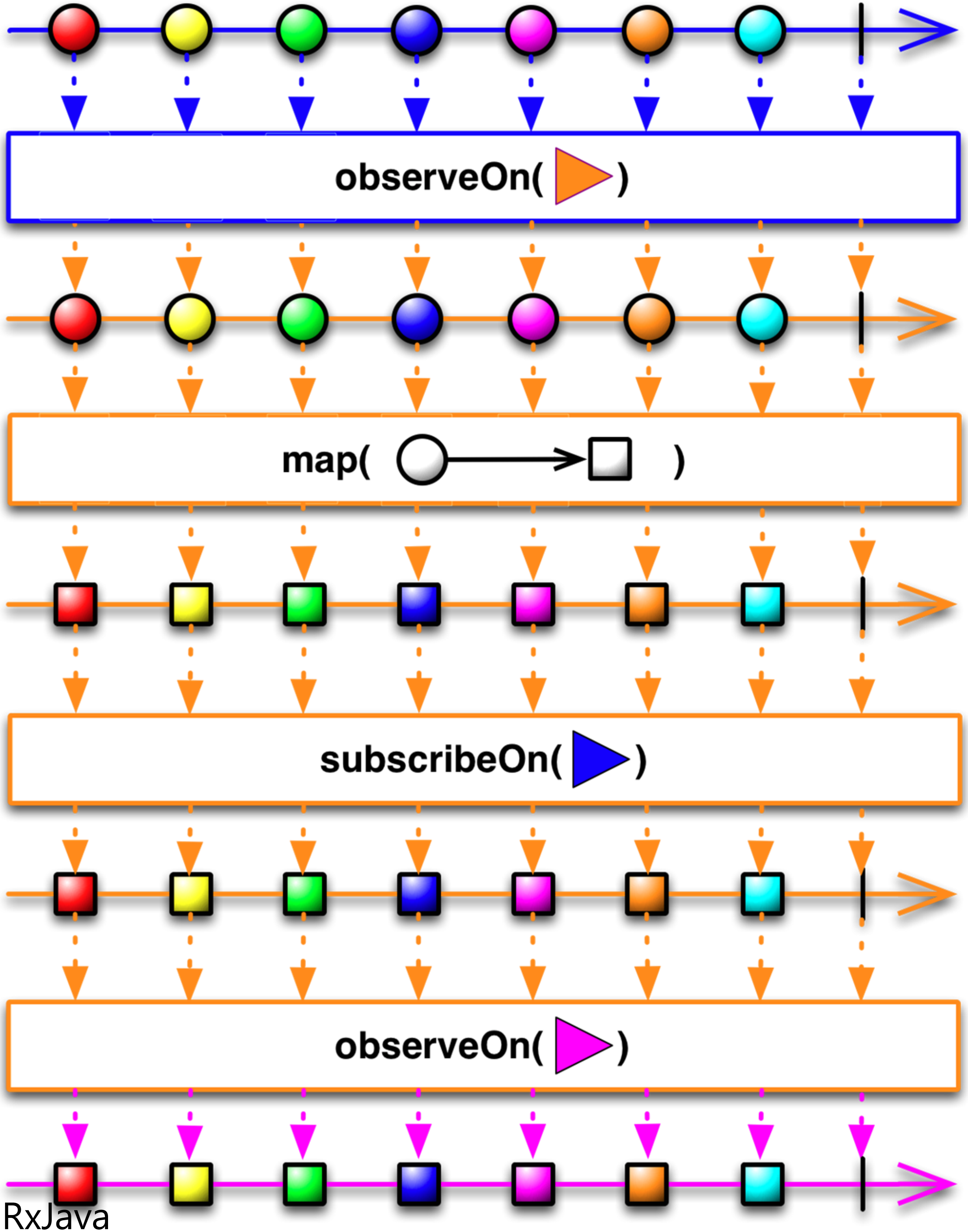
```
myScheduler.createWorker().schedule(…);


Subscription schedule(Action0 action);


Subscription schedule(Action0 action,
                      long delayTime,
                      TimeUnit unit);


Subscription schedulePeriodically(Action0 action,
                                  long initialDelay,
                                  long period,
                                  TimeUnit unit);
```

Scheduler



Scheduler




observeOn(▶)
map(○ → □)
subscribeOn(▶)
observeOn(▶)

.doXYZ(…)

- .**.do**XYZ(…) : pour les effets de bord (**Action0**, **Action1<T>**, etc.) :
  - **doOnCompleted(…)**
  - **doOnEach(…)**
  - **doOnEach(…)**
  - **doOnError(…)**
  - **doOnNext(…)**
  - **doOnRequest(…)**
  - **doOnSubscribe(…)**
  - **doOnTerminate(…)**
  - **doOnUnsubscribe(…)**
  - **finallyDo(…)**

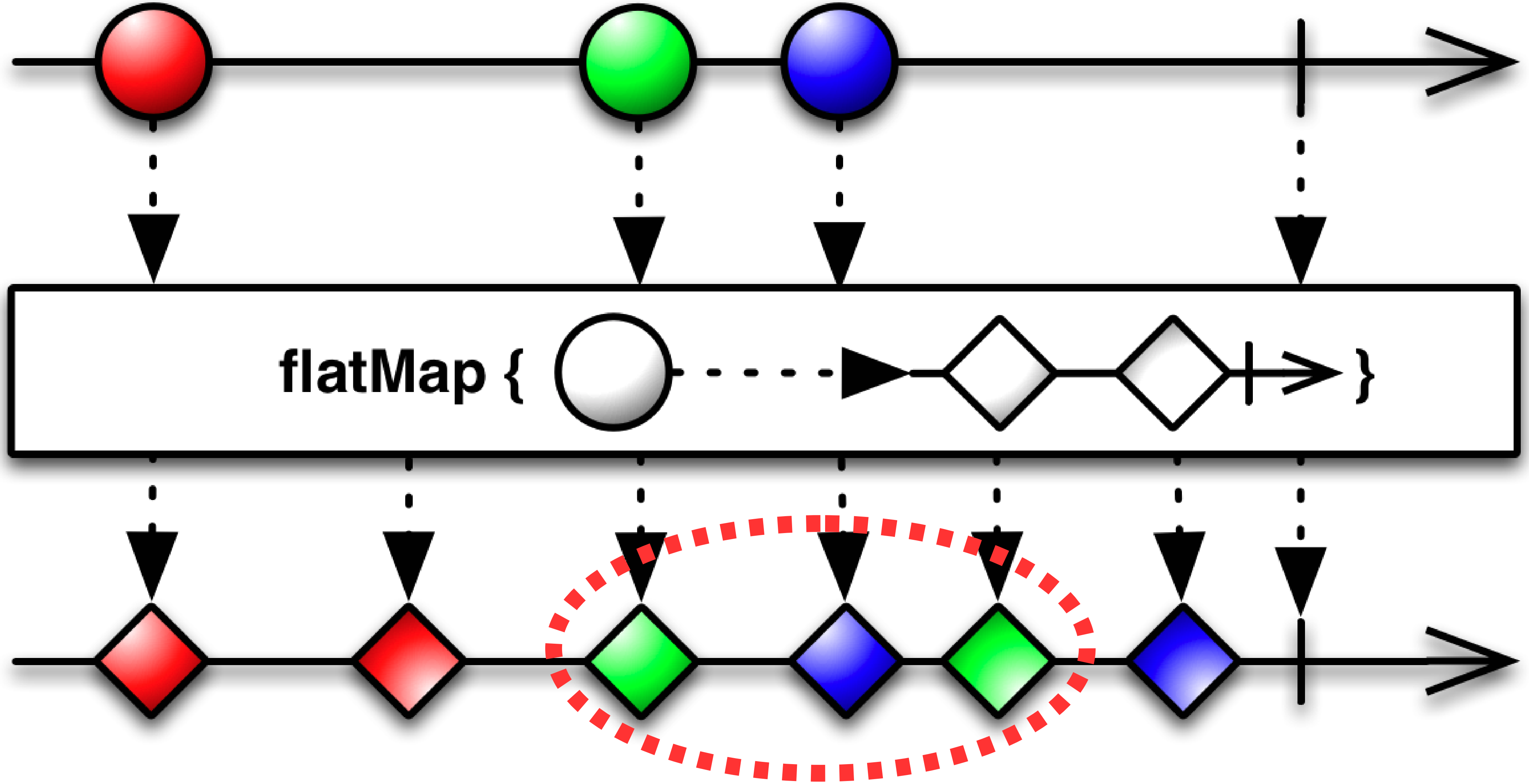
Composition

Optional<U> **flatMap**(Function<T, **Optional**<U>> f)

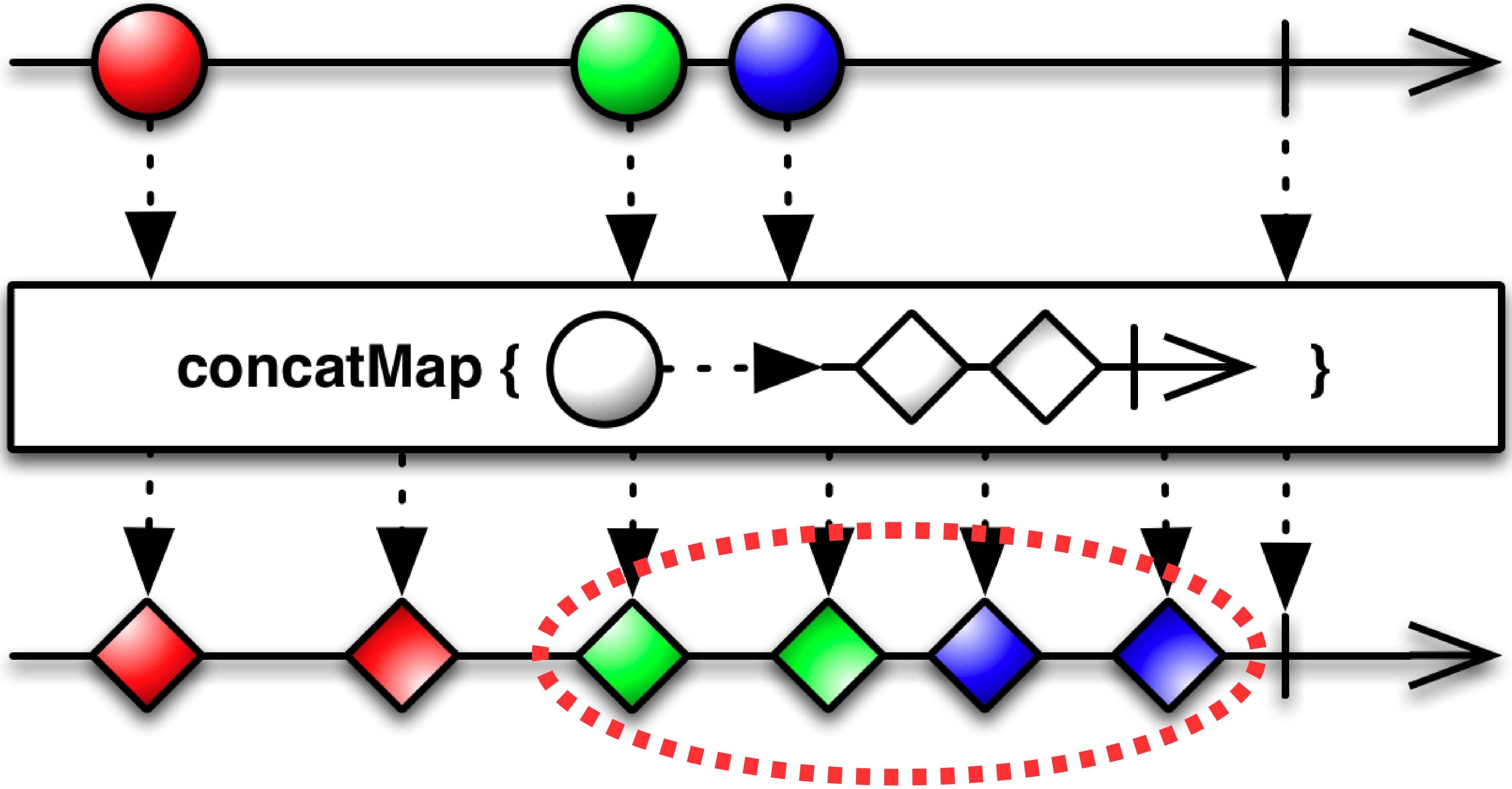
Stream<U> **flatMap**(Function<T, **Stream**<U>> f)

Observable<U> **flatMap**(Func1<T, **Observable**<U>> f)

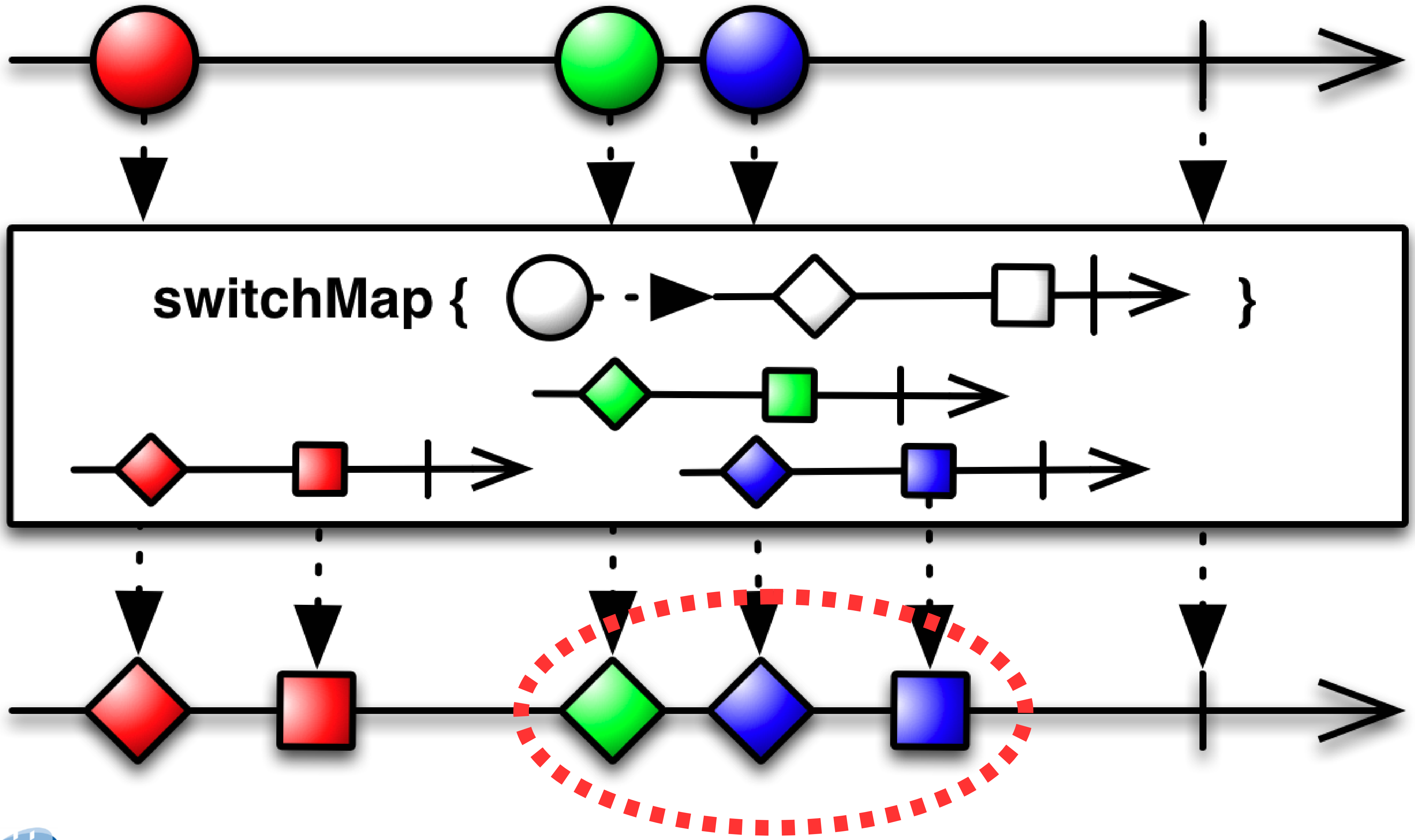
Composition



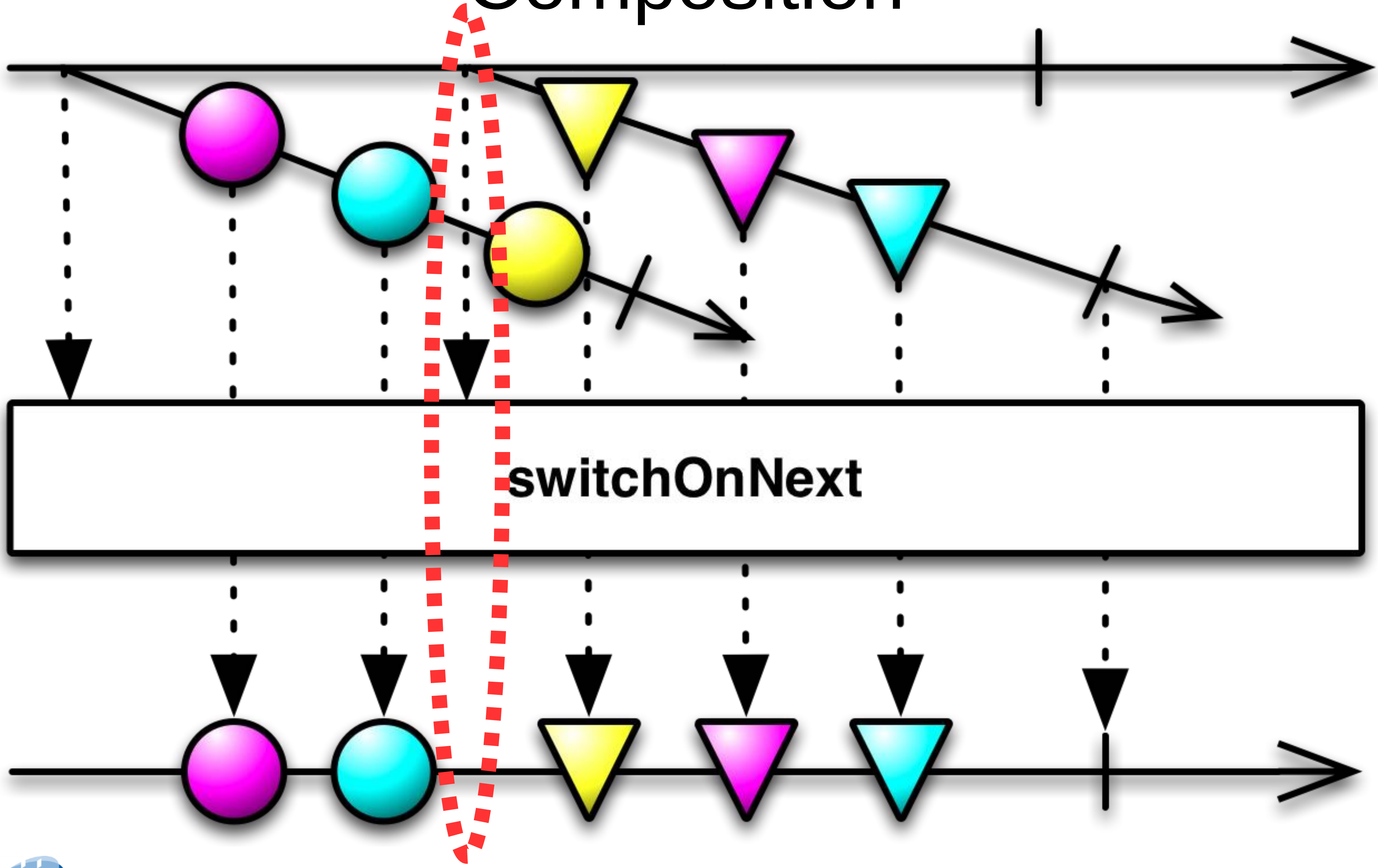
Composition



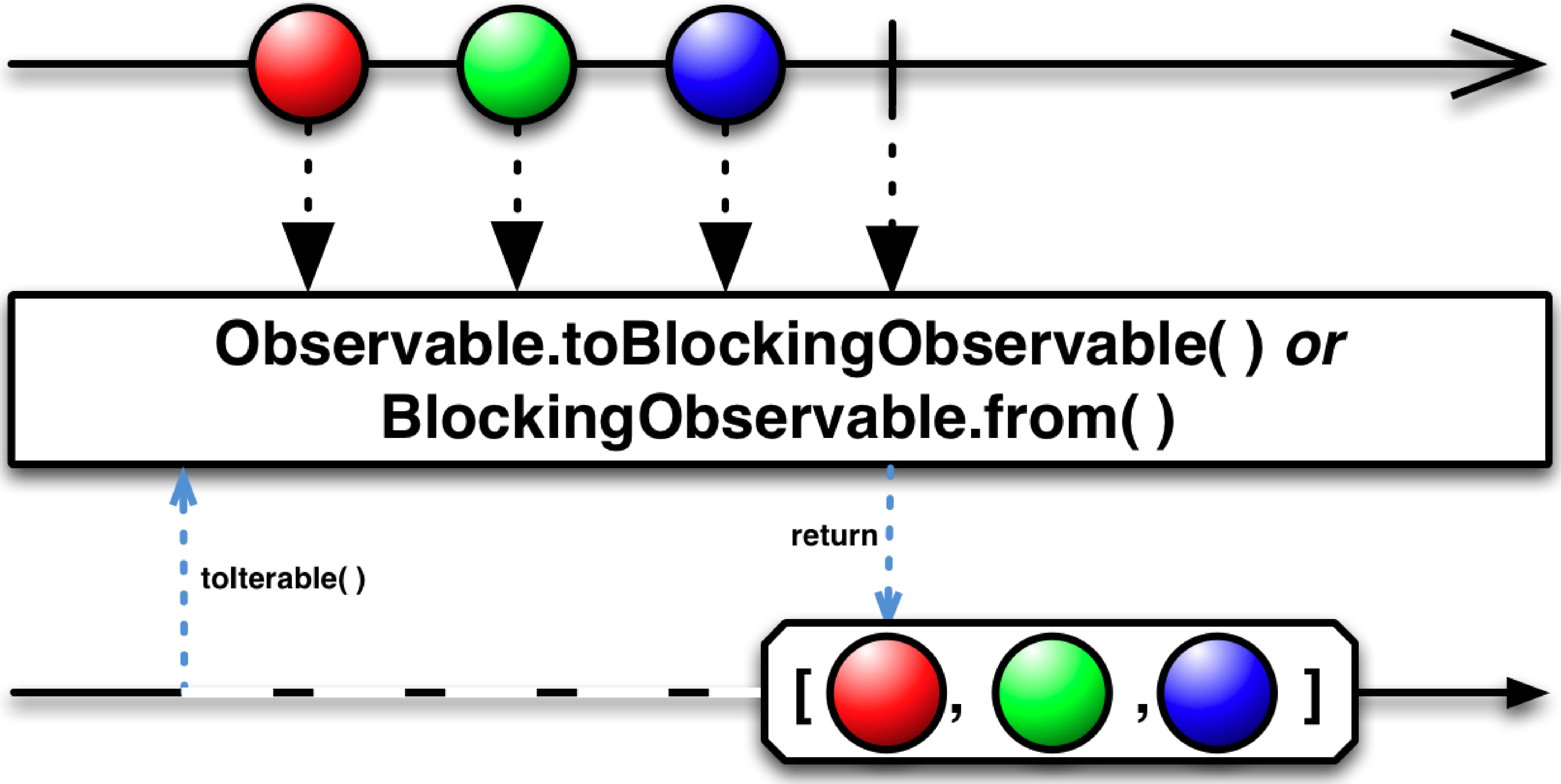
Composition



Composition



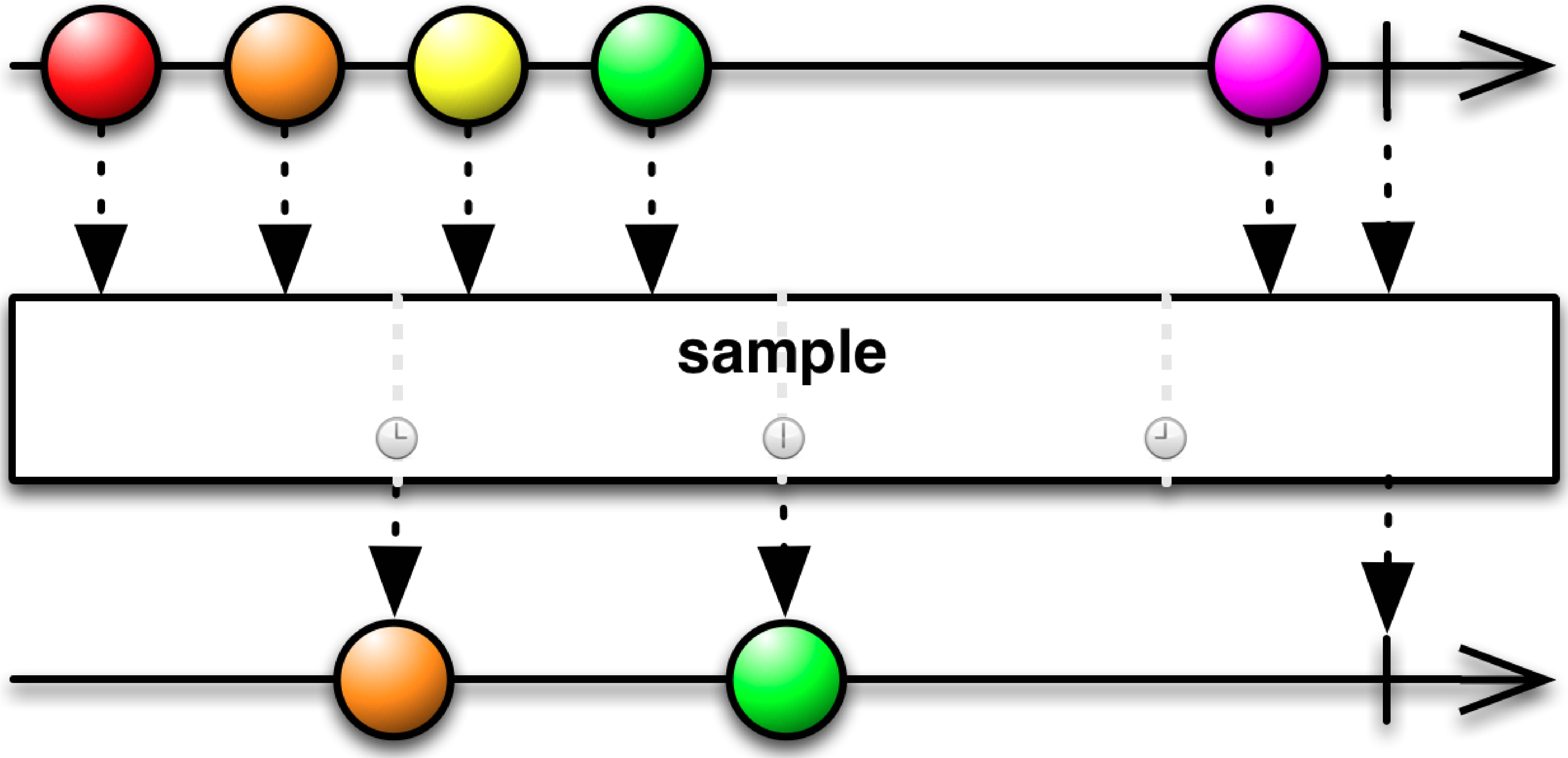
BlockingObservable


Observable.toBlockingObservable( ) *or*
BlockingObservable.from( )
toIterable( )
return
[, ,]

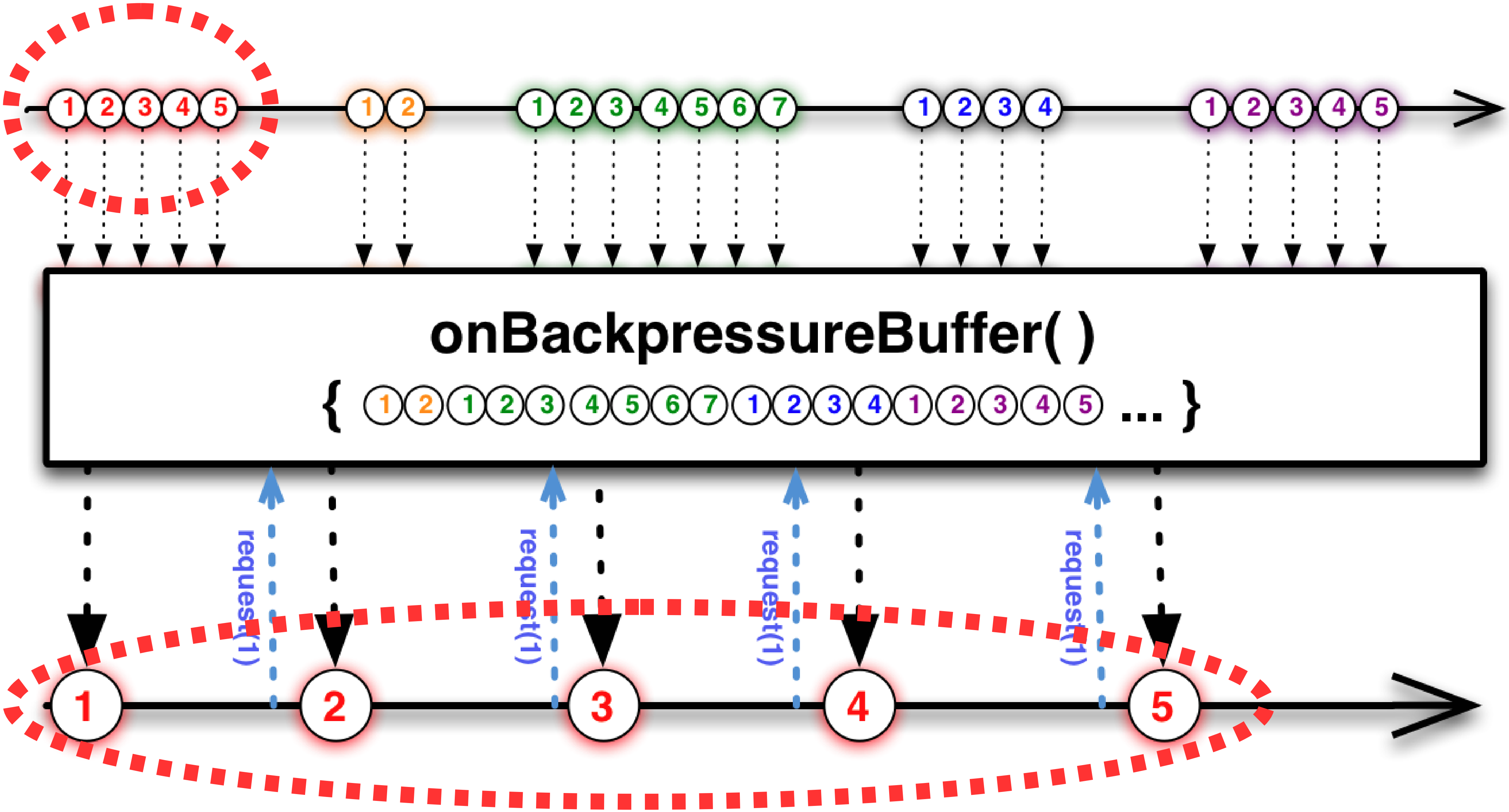
toBlocking() == better Stream

- /!\ N'utiliser **toBlocking()** que pour faire des traitements synchrones ou dans les tests. Rx propose beaucoup plus d'opérateurs que l'API **Stream** de Java8.
- Dans le code de production, **toujours le justifier** avec un petit commentaire.

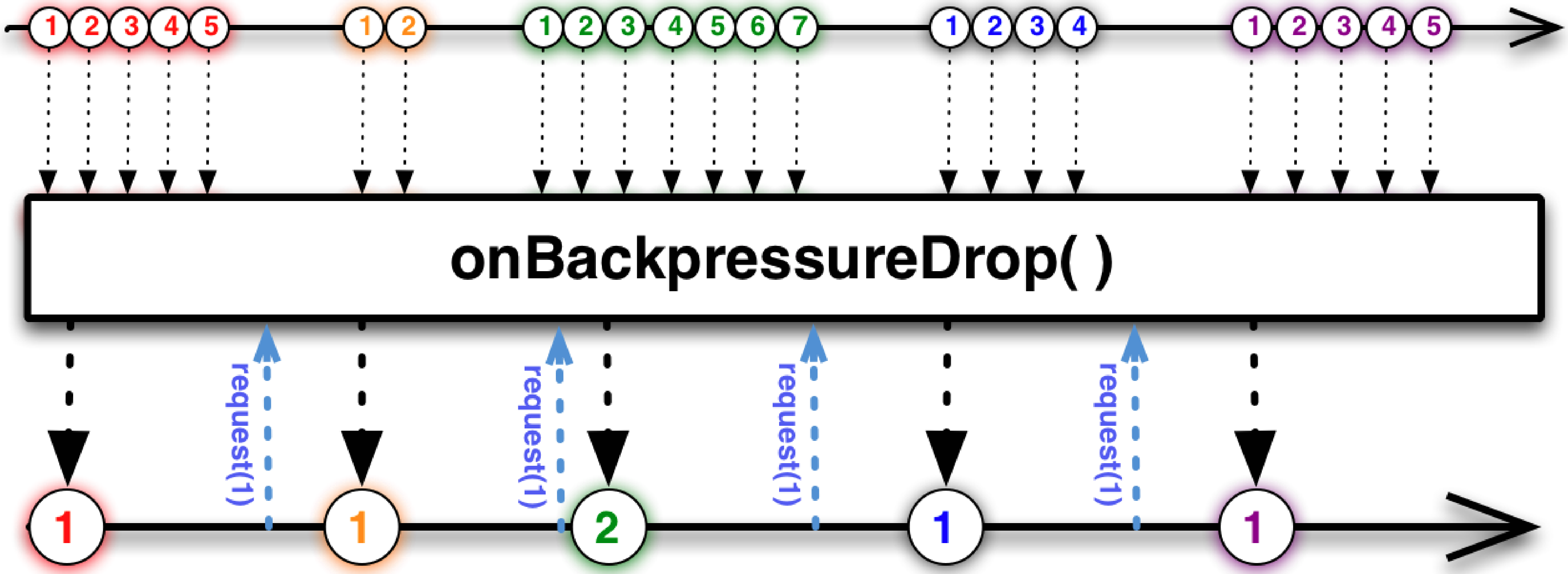
Backpressure



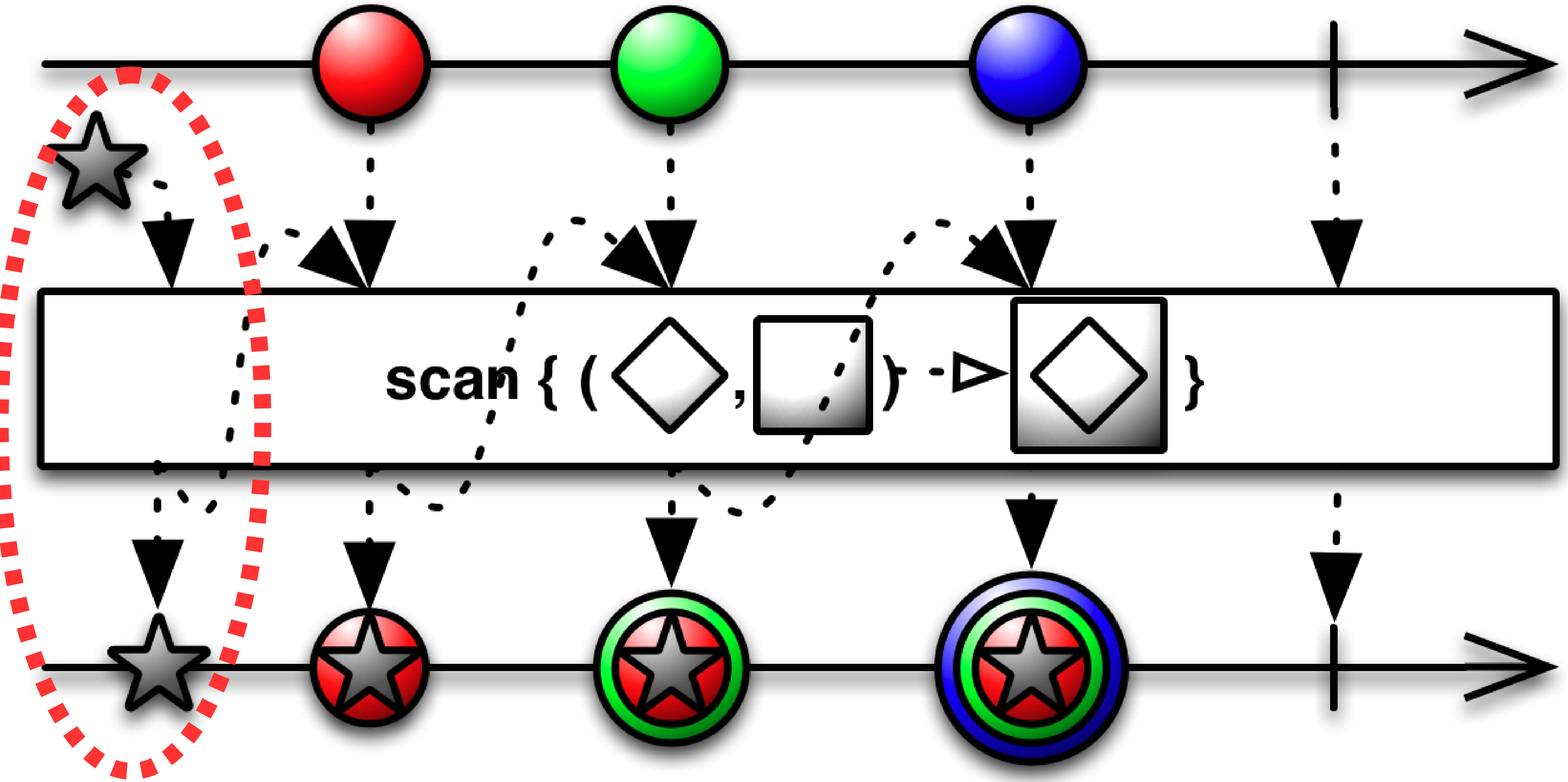
Backpressure



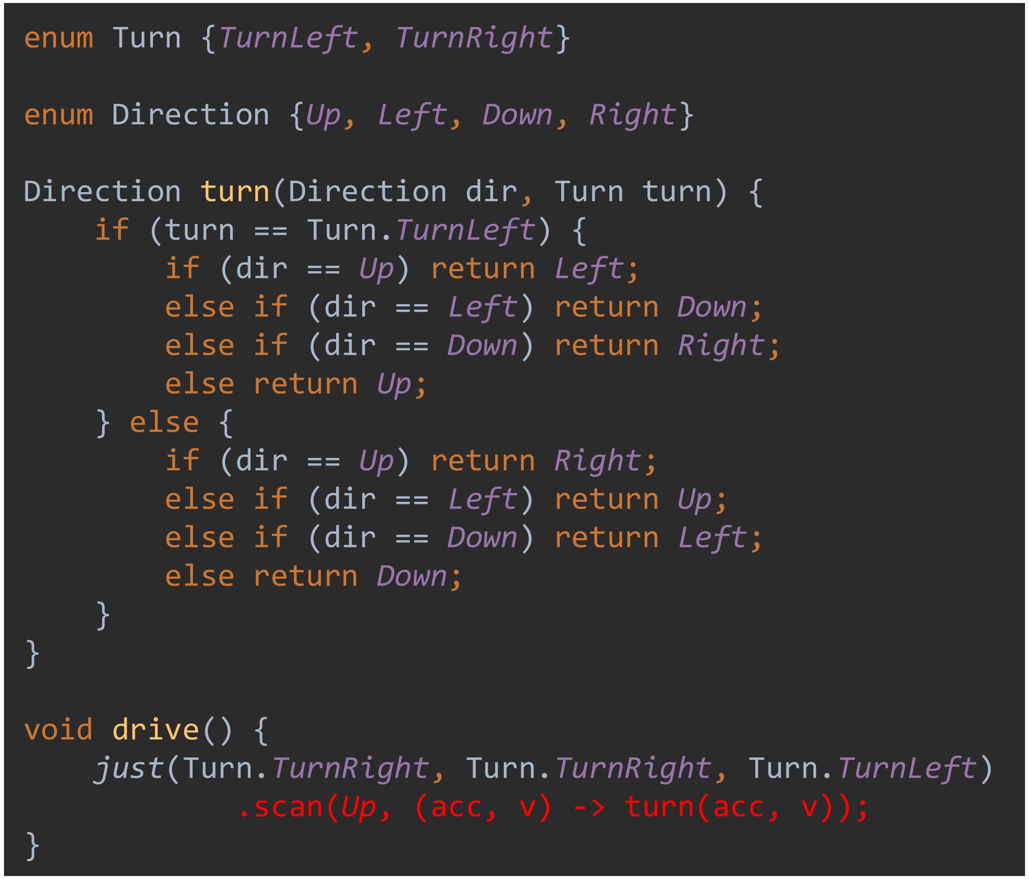
Backpressure



État

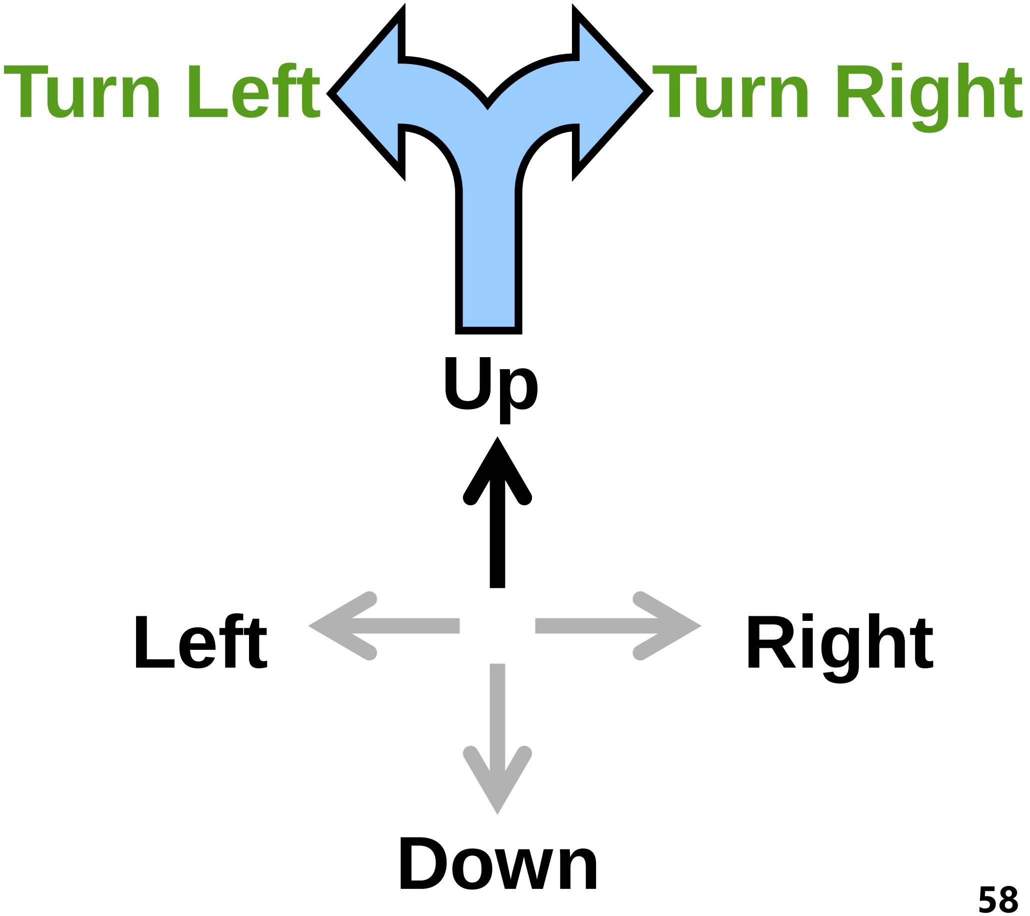


État



```java
enum Turn {TurnLeft, TurnRight}

enum Direction {Up, Left, Down, Right}

Direction turn(Direction dir, Turn turn) {
    if (turn == Turn.TurnLeft) {
        if (dir == Up) return Left;
        else if (dir == Left) return Down;
        else if (dir == Down) return Right;
        else return Up;
    } else {
        if (dir == Up) return Right;
        else if (dir == Left) return Up;
        else if (dir == Down) return Left;
        else return Down;
    }
}

void drive() {
    just(Turn.TurnRight, Turn.TurnRight, Turn.TurnLeft)
            .scan(Up, (acc, v) -> turn(acc, v));
}
```

Up, Right, Down, Right



Test

- TestScheduler
  - **advanceTimeBy(long delayTime, TimeUnit unit)**
  - advanceTimeTo(long delayTime, TimeUnit unit)
- TestSubscriber
  - **AssertValues**(...)
    - getLastSeenThread()
    - assertCompleted()
    - assertNotCompleted()
    - assertError()
    - assertNoTerminalEvent
- TestSubject (utilise TestScheduler)

Compose

- Factorisation d'une portion d'un traitement.

```
Observable<String> str1 = ints1.compose(factorisation());
Observable<String> str2 = ints2.compose(factorisation());
```

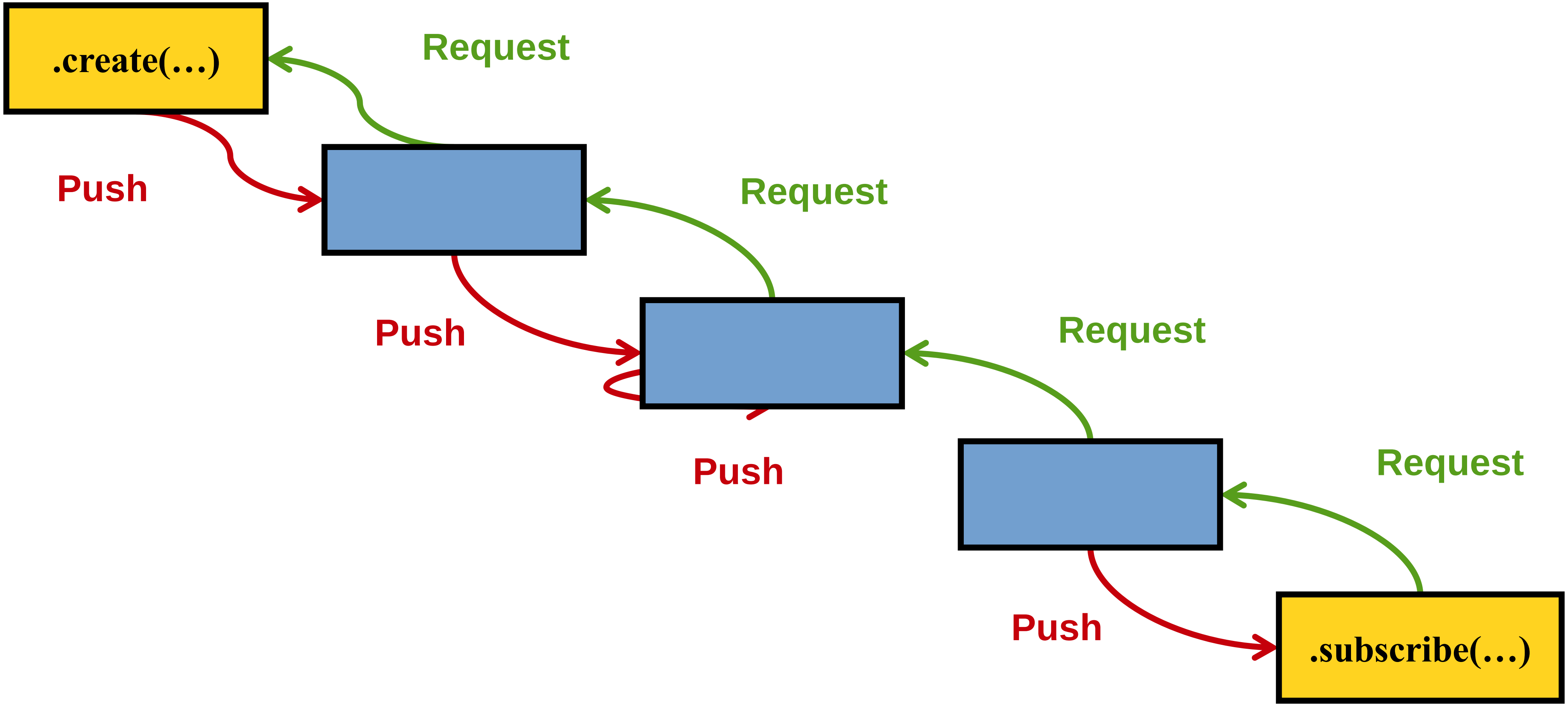
```
Transformer<Integer, String> factorisation() {
    return o -> o.delay(1, SECONDS)
            .map(v -> 1000 + v)
            .map(v -> "V-" + v);
}
```

```
interface Transformer<T, R> extends
        Func1<Observable<T>, Observable<R>>
```

Lift

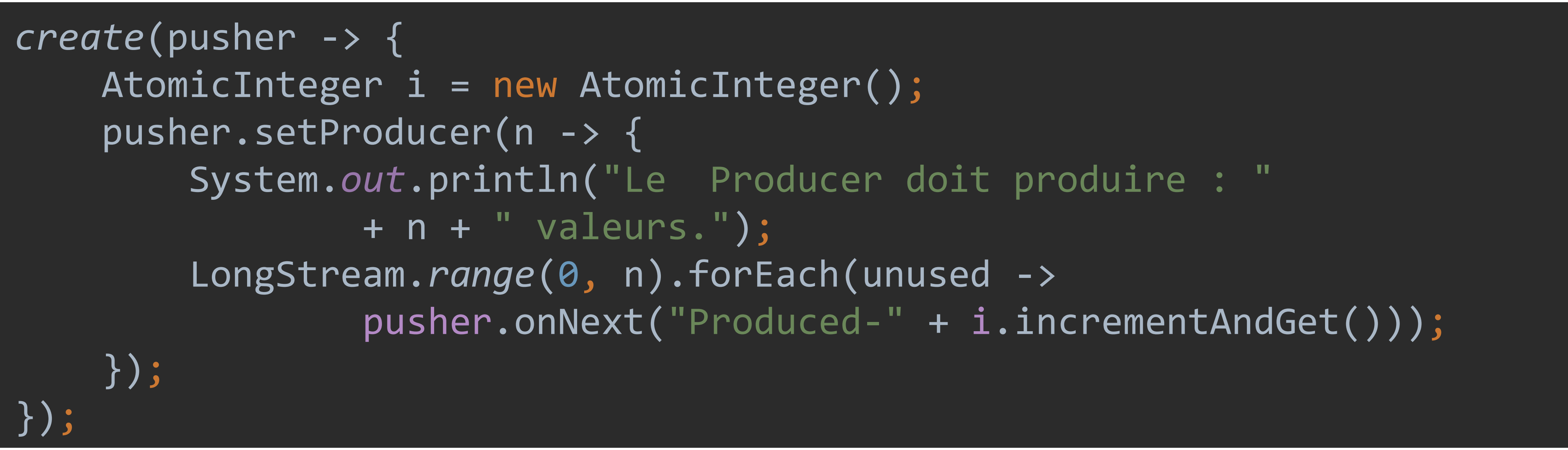
```
Observable<R> lift(final Operator<R, T> operator)

interface Operator<R, T> extends
        Func1<Subscriber<R>, Subscriber<T>>

.lift(child -> new Subscriber<Integer>(child) {
        @Override public void onCompleted() {
            child.onCompleted();
        }

        @Override public void onError(Throwable e) {
            child.onError(e);
        }

        @Override public void onNext(Integer i) {
            child.onNext((i % 2 == 0 ? "Even" : "Odd")
                         + " : " + i);
        }})
```

Producer


.create(…)
Request
Push
Request
Push
Request
Push
Request
Push
.subscribe(…)

Producer

- Permet de contrôler le nombre de valeur « pushées » afin d'éviter les BackpressureException.



```
create(pusher -> {
    AtomicInteger i = new AtomicInteger();
    pusher.setProducer(n -> {
        System.out.println("Le  Producer doit produire : "
                + n + " valeurs.");
        LongStream.range(0, n).forEach(unused ->
                pusher.onNext("Produced-" + i.incrementAndGet()));
    });
});
```

[Rx 1 @Beta] **Single<T>**

- **compose()**
- **concat()** & **concatWith()**
- **create()**
- **delay()**
- **error()**
- **flatMap()**
- **flatMapObservable()** retourne un Observable
- **just()**
- **map()**
- **merge()**
- **merge()** & **mergeWith()**
- **observeOn()**
- **onErrorReturn()**
- **subscribeOn()**
- **timeout()**
- **anObservable.toSingle()**
- **aSingle.toObservable()**
- **zip()** & **zipWith()**

[Rx 1 @Experimental] **Completable**

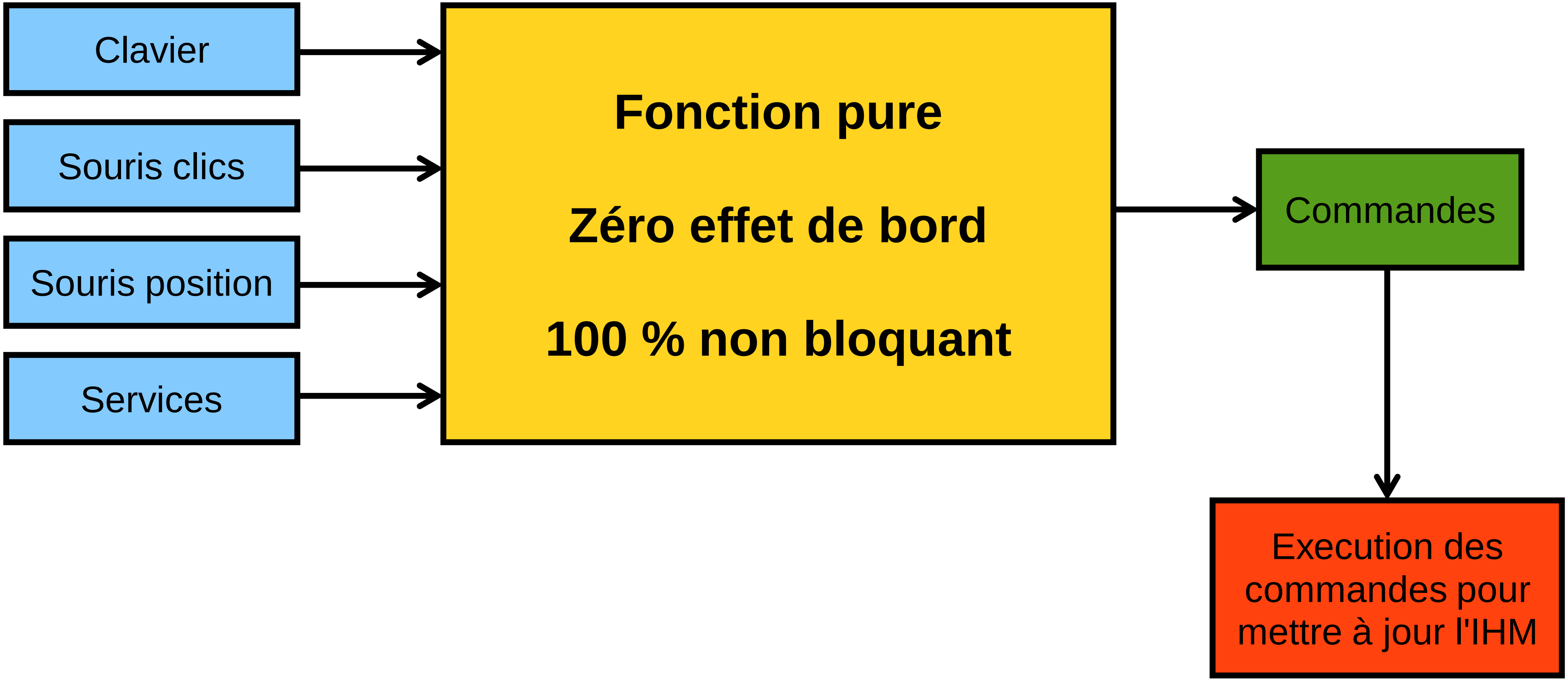
.complete()
.concat()
.create()
.defer()
.error()
.fromAction()
.fromCallable()
.fromFuture()
.fromObservable()
.fromSingle()
.merge()
.mergeDelayError()
.never()
.timer()
.using()
.ambWith()
.await()
.compose()
.andThen()
.concatWith()
.delay()
.doOnComplete()
.doOnUnsubscribe()
.doOnError()

.doOnSubscribe()
.doOnTerminate()
.endWith()
.doAfterTerminate()
.get()
.lift()
.mergeWith()
.observeOn()
.onErrorComplete()
.onErrorResumeNext()
.repeat()
.repeatWhen()
.retry()
.retryWhen()
.startWith()
.subscribe()
.subscribeOn()
.timeout()
.to()
.toObservable()
.toSingle()
.toSingleDefault()
.unsubscribeOn()

FuncN

R call(**Object...** args)

Exercices


Clavier
Souris clics
Souris position
Services
Fonction pure
Zéro effet de bord
100 % non bloquant
Commandes
Execution des commandes pour mettre à jour l'IHM

Commandes

- addPt
- addLine
- addText
- addLog
- uniq(id, commande)
- removeUniq(id)
- group(commandes)
- clear