Opérateurs Rx

Sommaire

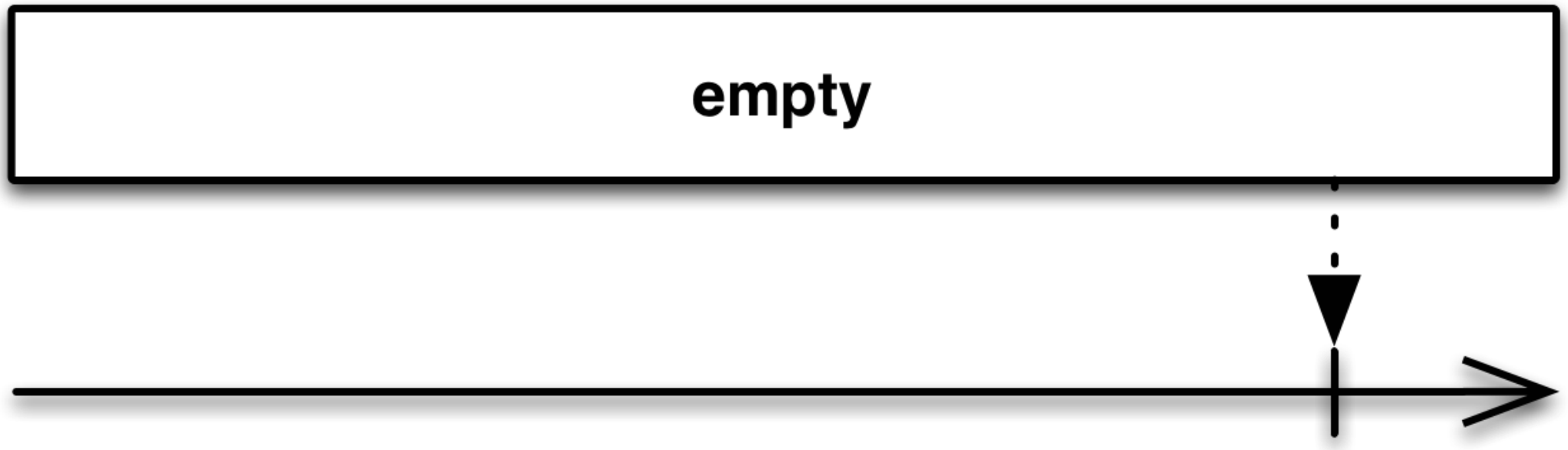
- Création
- Prédicat
- Filtrage
- Transformation
- Réduction
- Combinaison
- Effet de bord
- Erreur
- Backpressure
- Utilitaire
- Scheduling
- Timeout
- Regroupement
- Connectable

Opérateurs

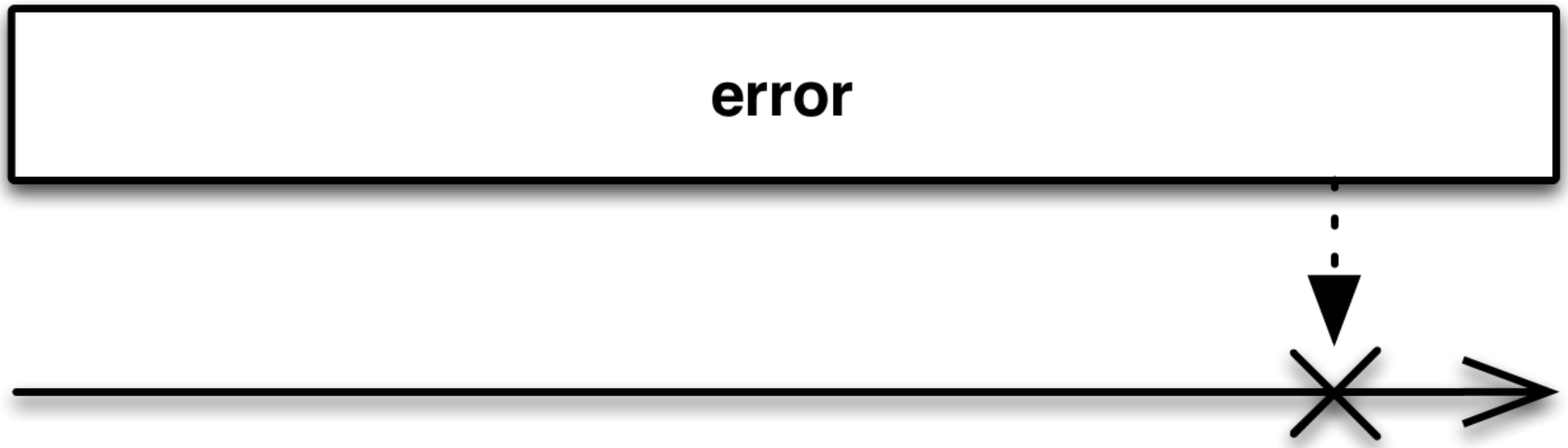

combineLatest
subscribe
just
concat
timer
debounce
zip
startWith
flatMap
buffer
merge
amb
window
timeout
takeLastBuffer
toMultimap
takeLast
replay
repeat
interval
toSortedList
delay
skip
retry
scan
from
mergeDelayError
onBackpressureDrop
onBackpressureBuffer
delaySubscription
onErrorResumeNext
lastOrDefault
throttleWithTimeout
flatMapIterable
toMap
distinct
takeUntil
throttleFirst
throttleLast
concatEager
timeInterval
sample
skipLast
singleOrDefault
concatMapEager
repeatWhen
publish
distinctUntilChanged
timestamp
sequenceEqual
onBackpressureBlock
retryWhen
firstOrDefault
zipWith
switchIfEmpty
doOnTerminate
switchMap
toSingle
observeOn
unsafeSubscribe
ignoreElements
doOnRequest
doOnCompleted
groupBy
forEach
single
map
create
cache
range
count
take
serialize
filter
error
unsubscribeOn
doOnNext
doOnError
elementAt
flatMapIterable
lastOrDefault
throttleWithTimeout
onBackpressureLatest
onBackpressureDrop
mergeDelayError
toMultimap
mergeWith
takeLastBuffer
onExceptionResumeNext
isEmpty
defaultIfEmpty
materialize
empty
delaySubscription
onErrorResumeNext
withLatestFrom
countLong
groupJoin
switchOnNext
dematerialize
compose
doOnEach
reduce
first
cast
extend
using
defer
never
contains
exists
collect
nest
toList
fromCallable
skipUntil
last
ofType
finallyDo
osObservable
elementAtOrDefault
doOnSubscribe
toBlocking
takeWhile
onErrorReturn
limit
concatMap
skipWhile
takeFirst
ambWith
subscribeOn
doOnUnsubscribe
concatWith
share
join
all
lift

Creation

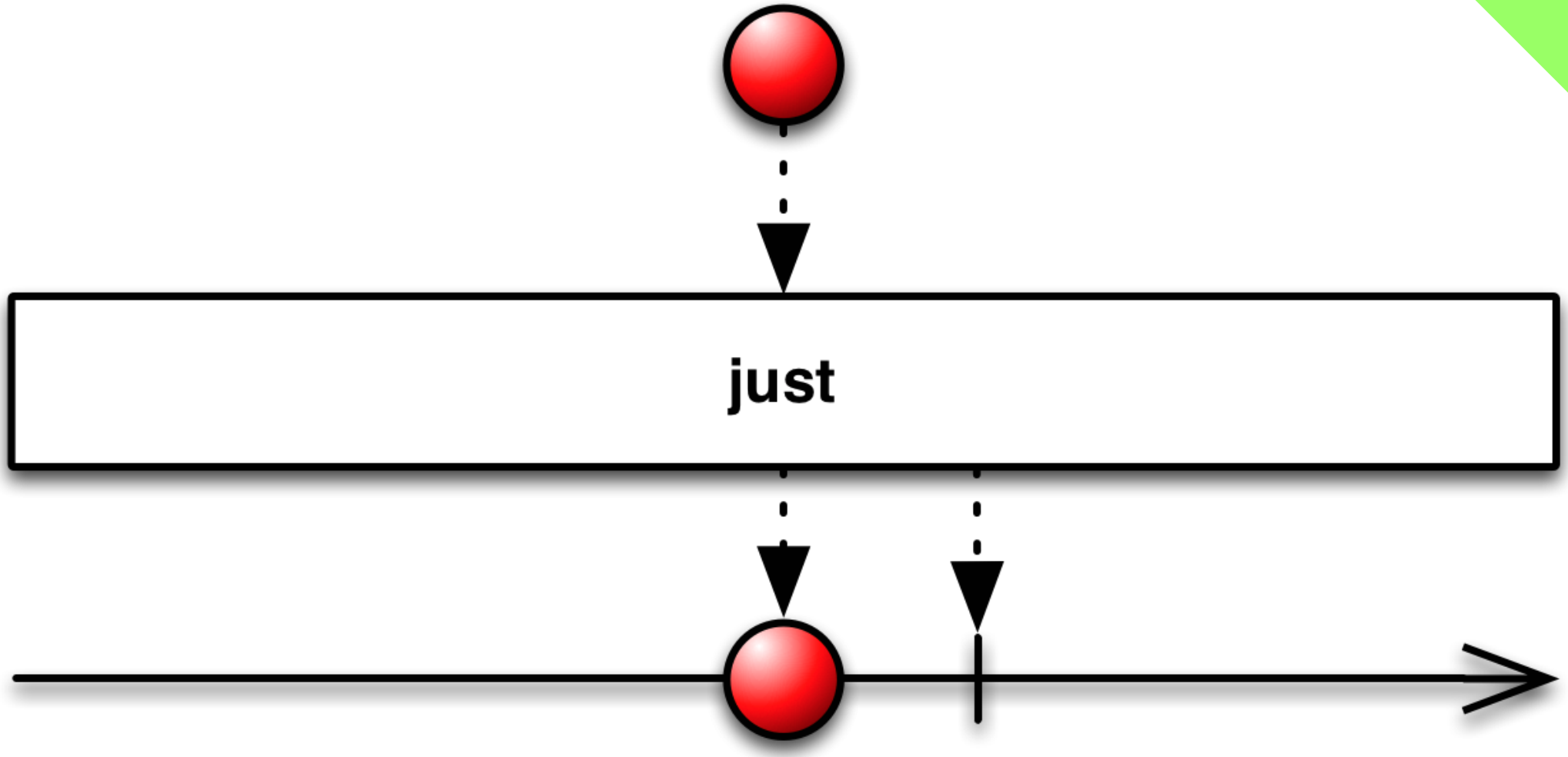
- Zéro onNext
- Zéro onError
- Zéro onCompleted



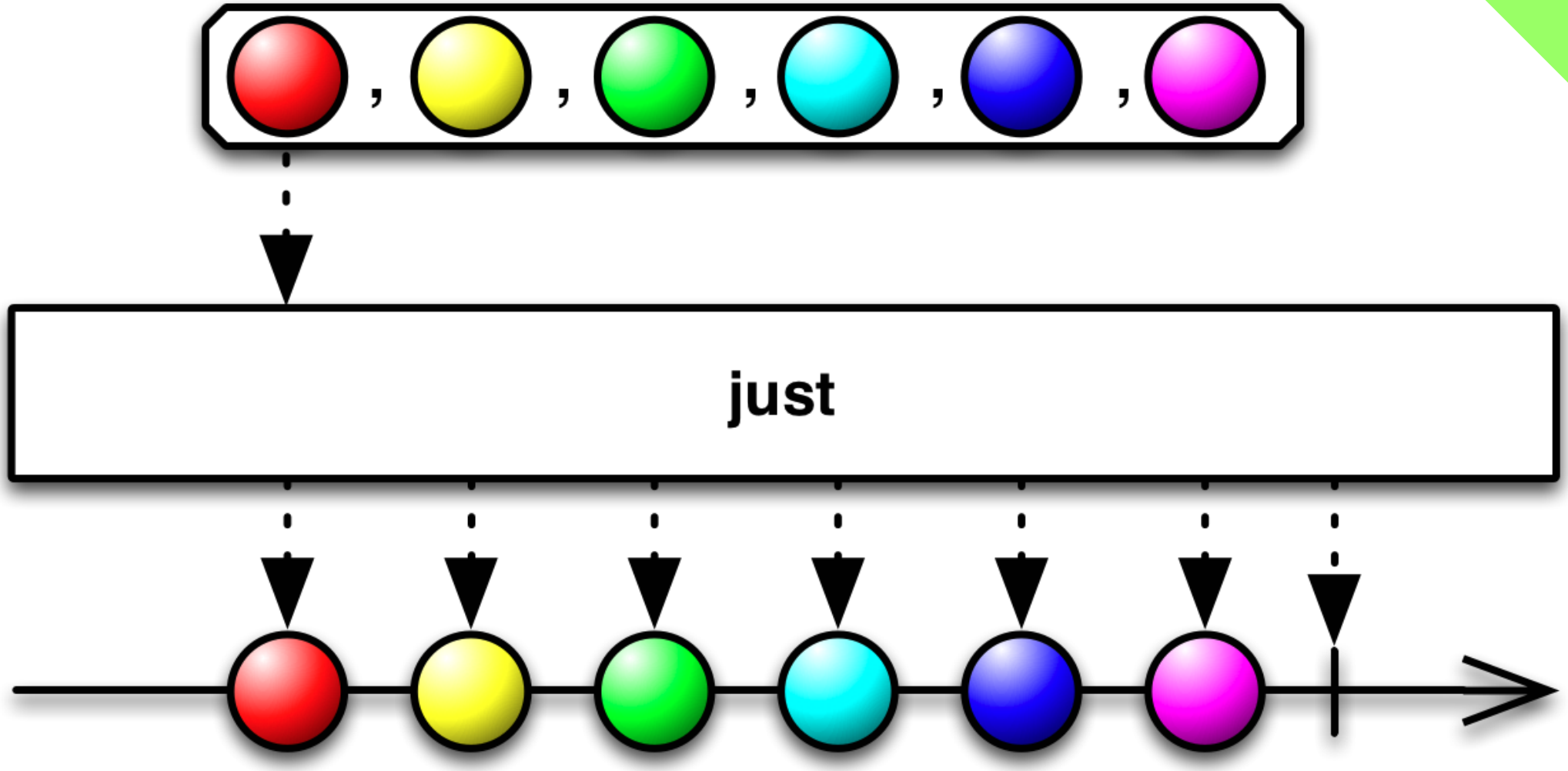
- Zéro onNext
- Un onCompleted



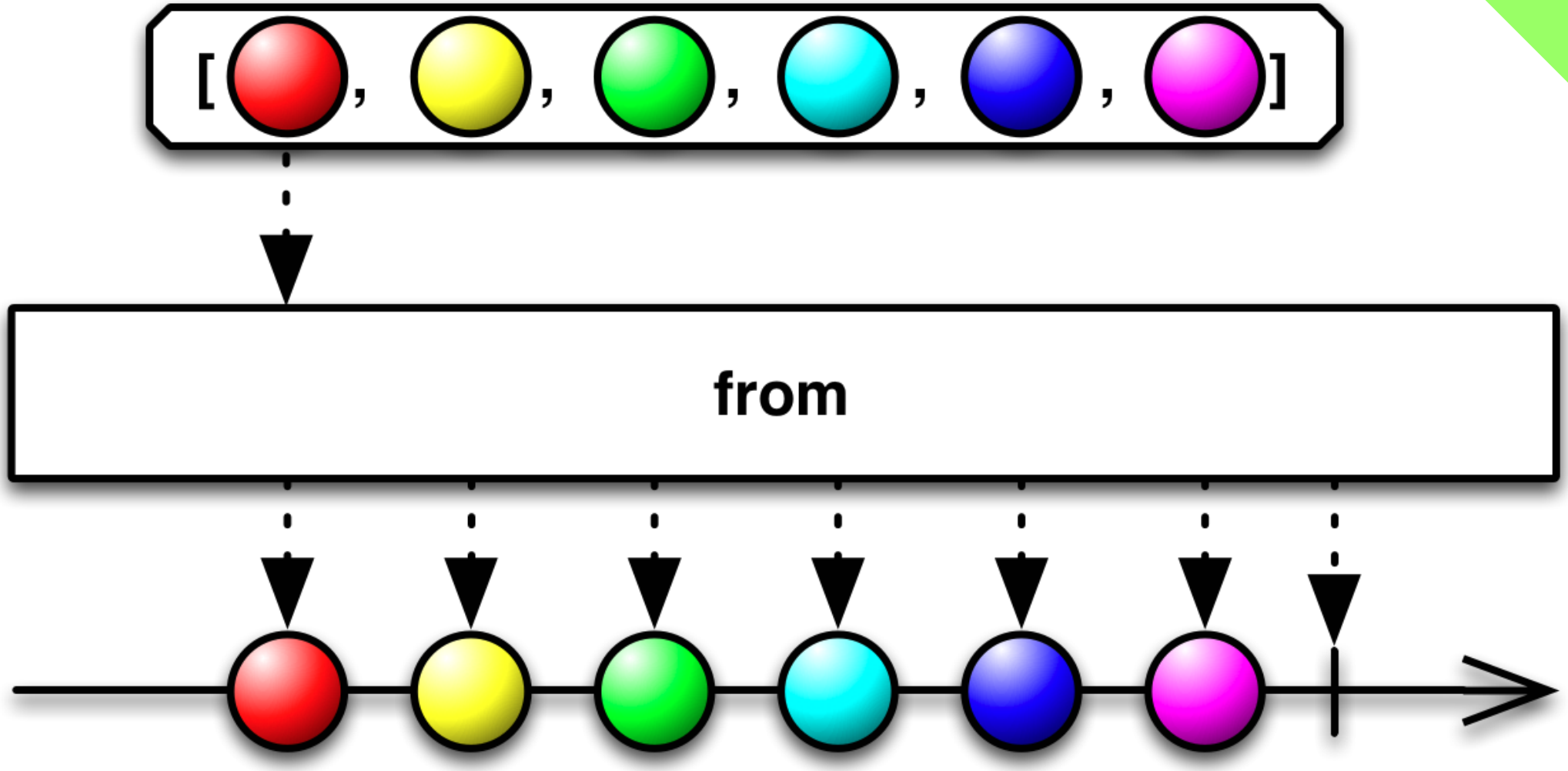
- Zéro onNext
- Un onError

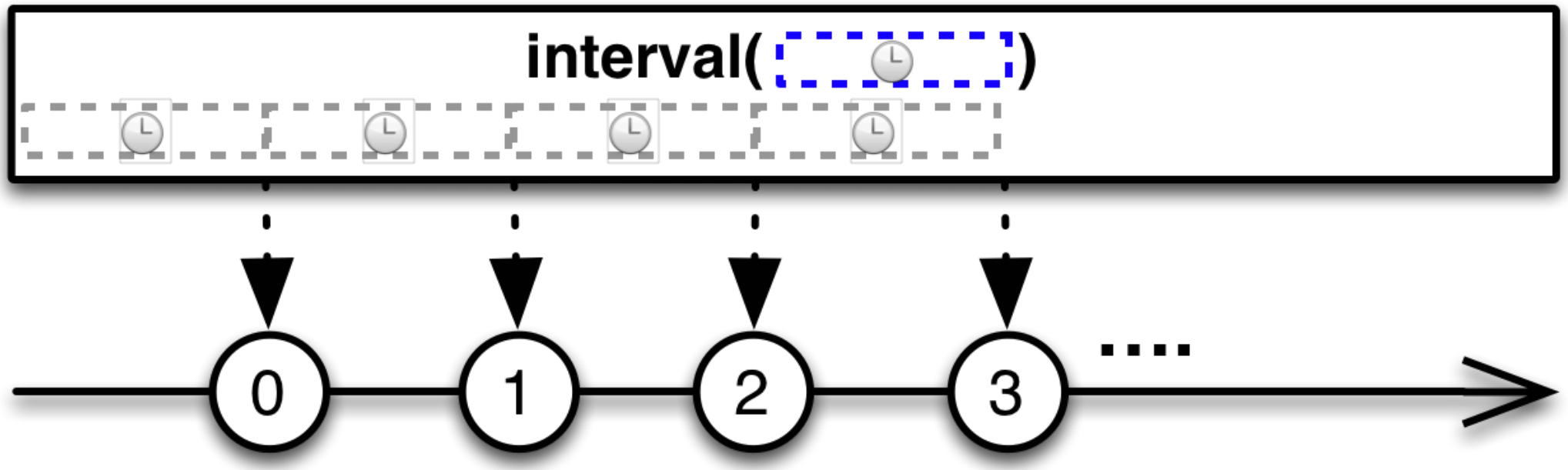


- Un onNext
- Un onCompleted

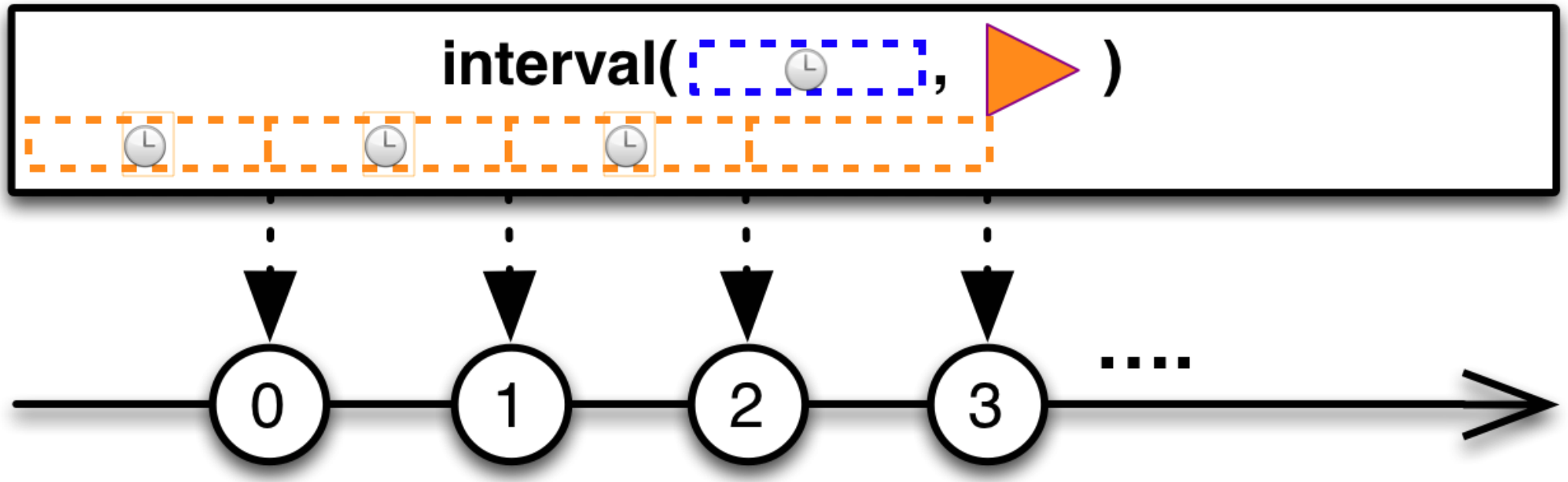


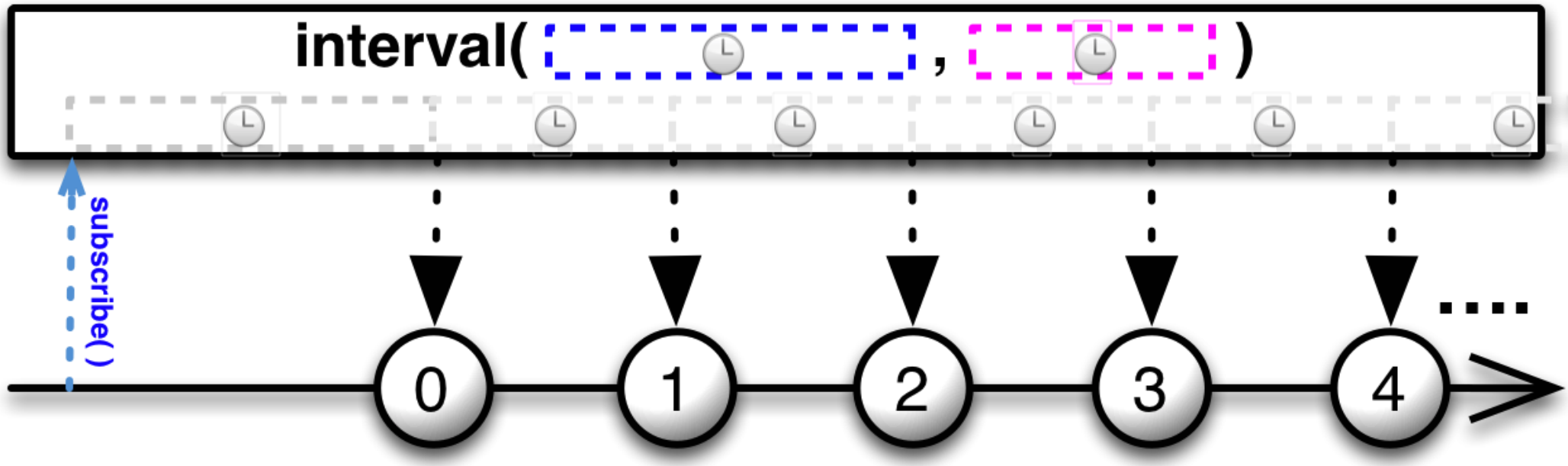
- N onNext
- Un onCompleted

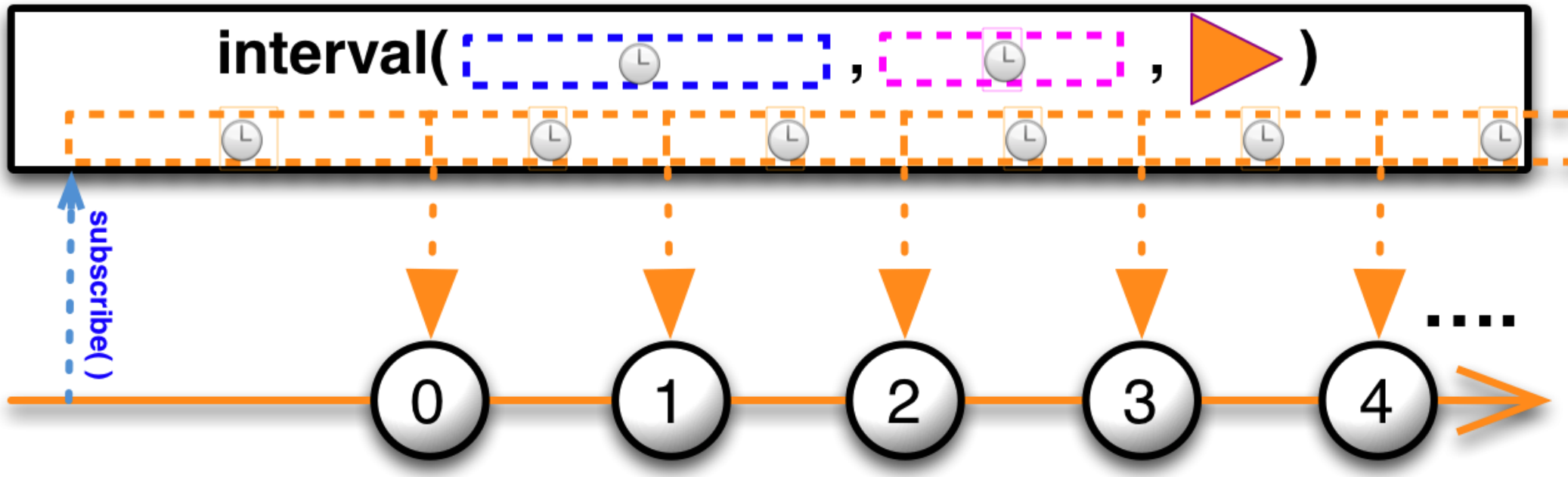

[, , , , ,]
from

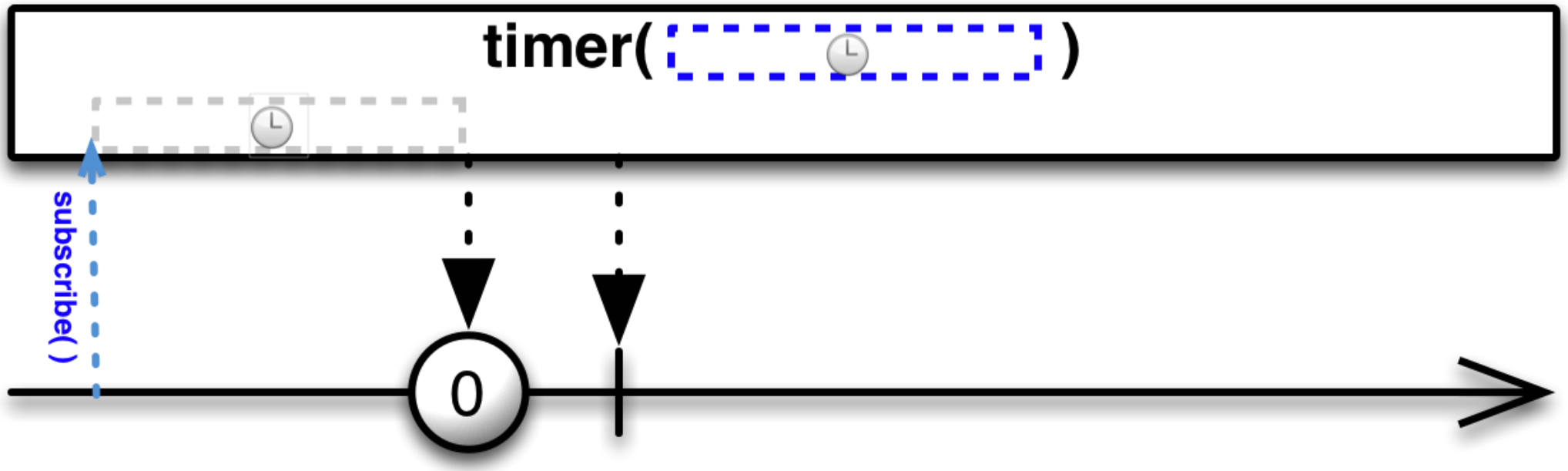


- Cas d'usage : Cadencement d'un traitement.

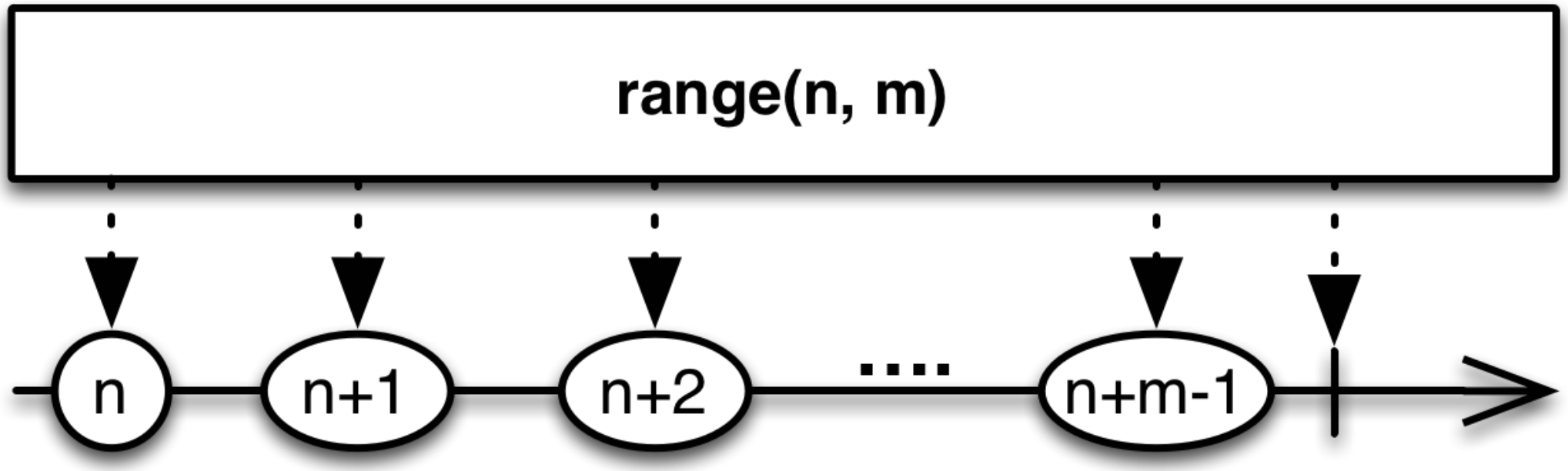

interval(,)
0
1
2
3
....


interval(,)
subscribe( )
0
1
2
3
4

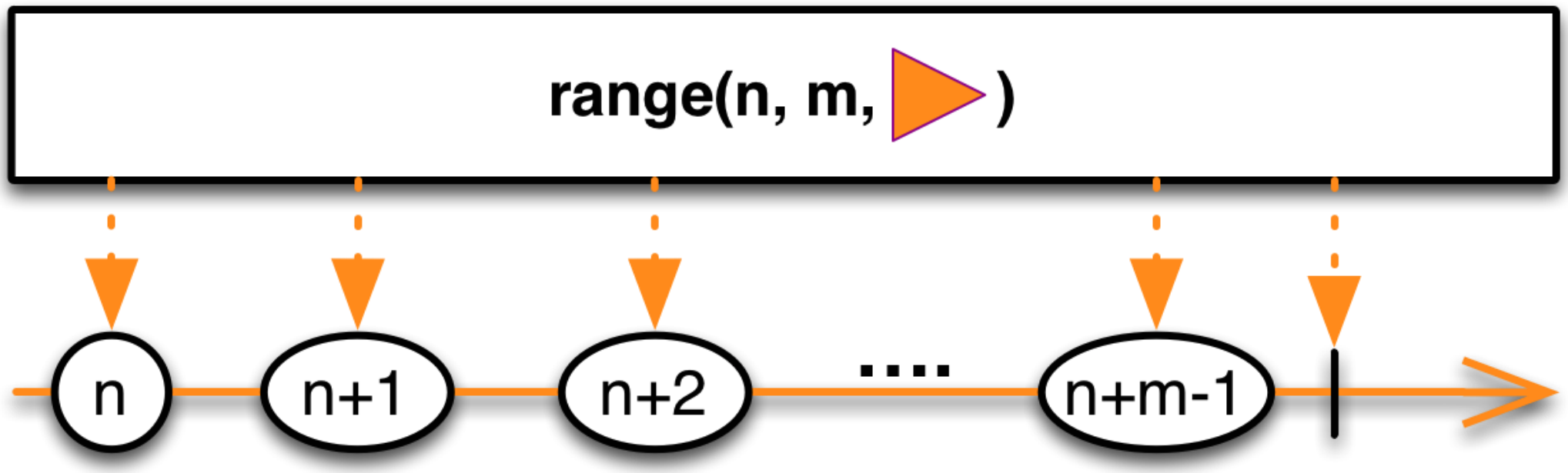

interval(, ,)
subscribe( )
0
1
2
3
4
....

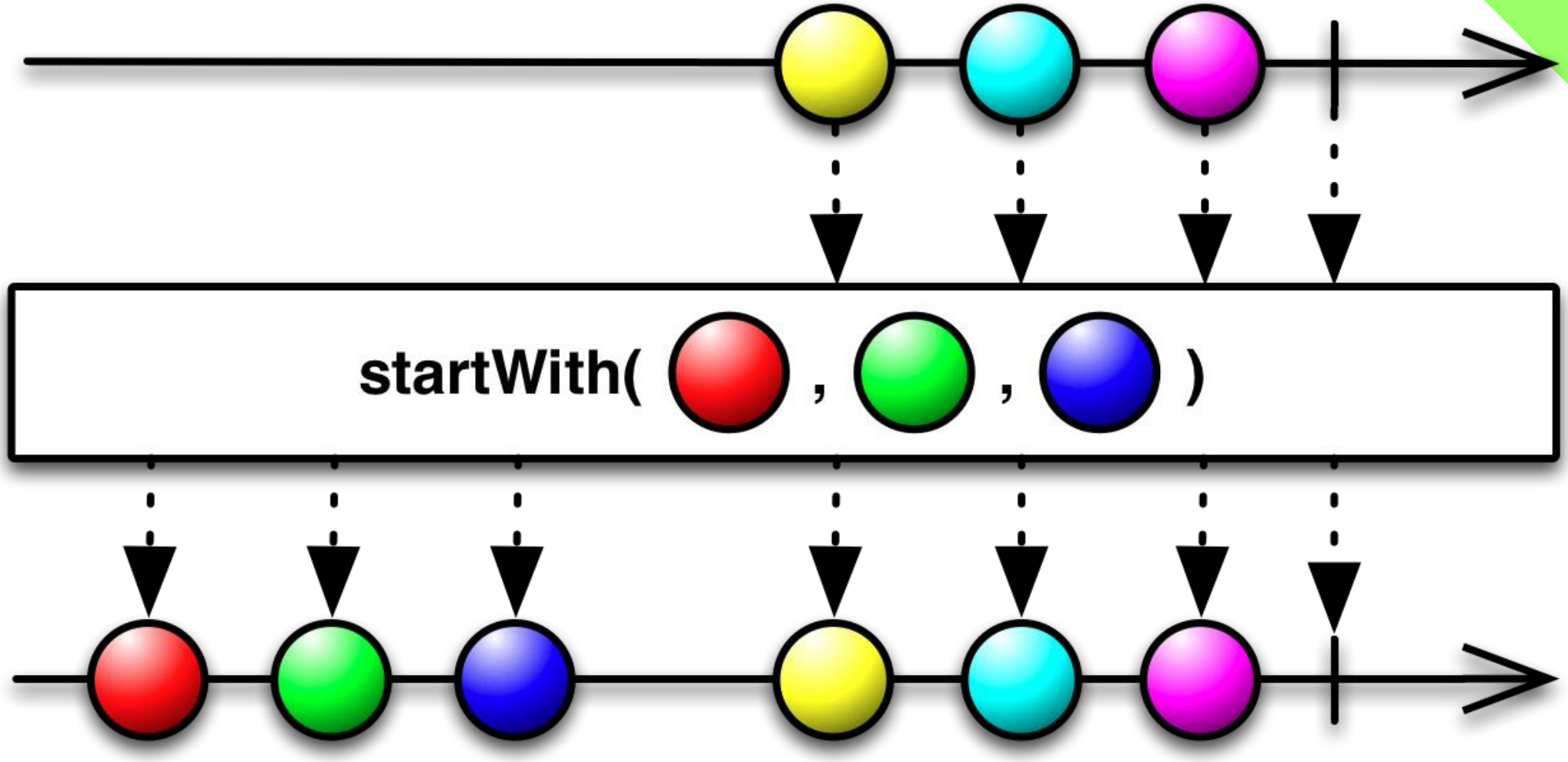


- Cas d'usage : différer un traitement.

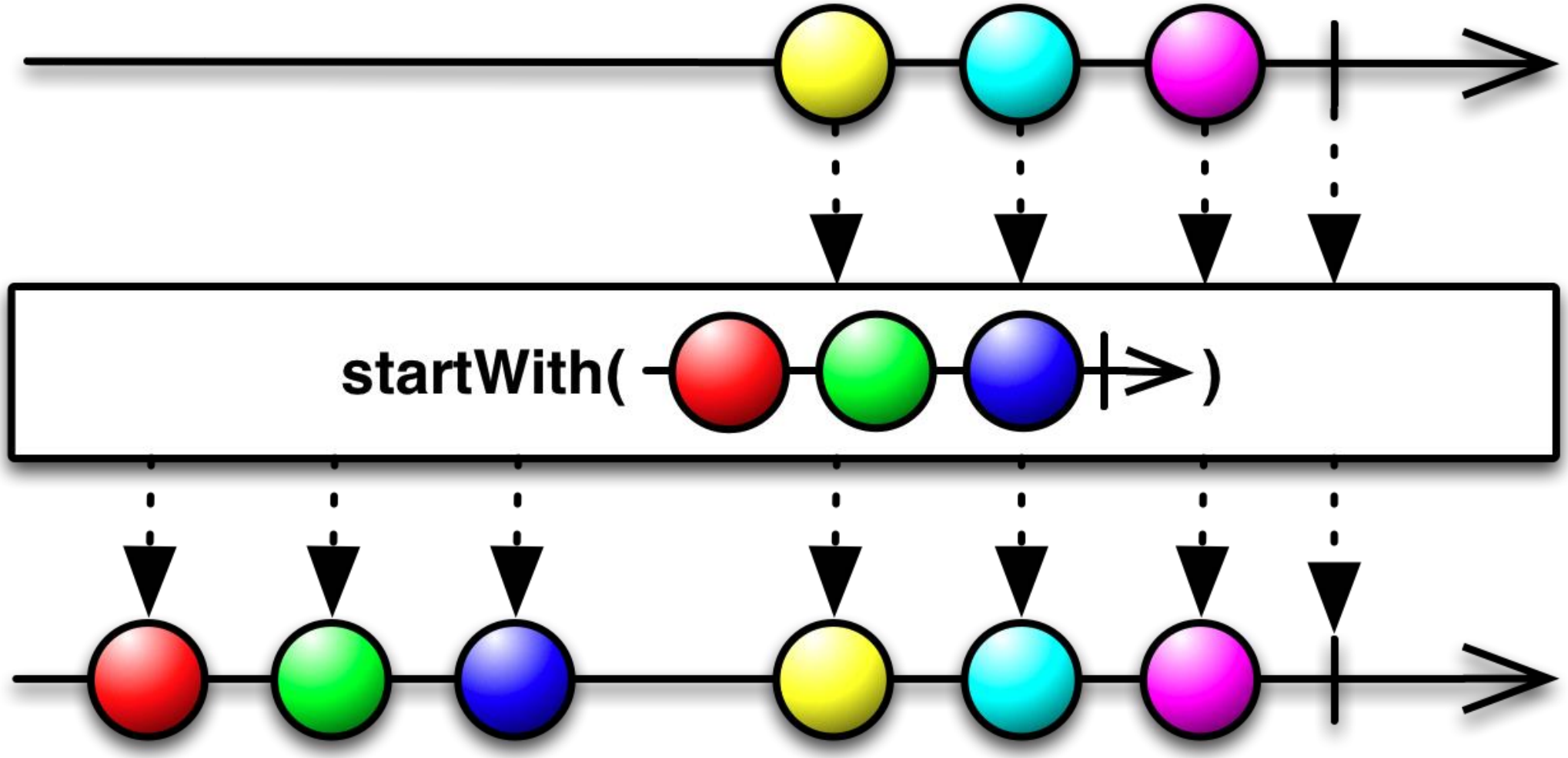


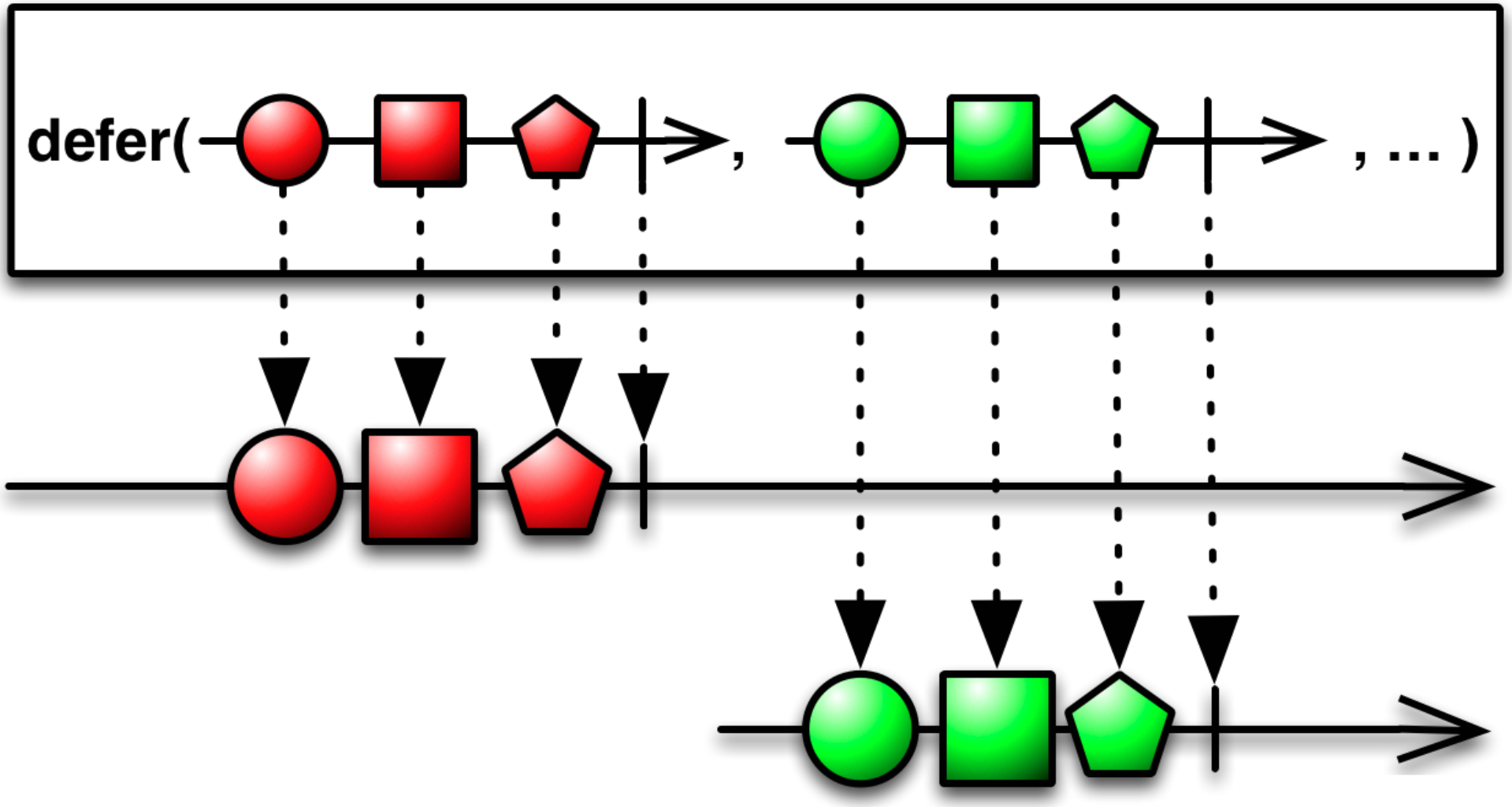
- range(int start, int count)
- Cas d'usage : lorsque l'on pense à une boucle d'entiers.


range(n, m,)
n
n+1
n+2
....
n+m-1

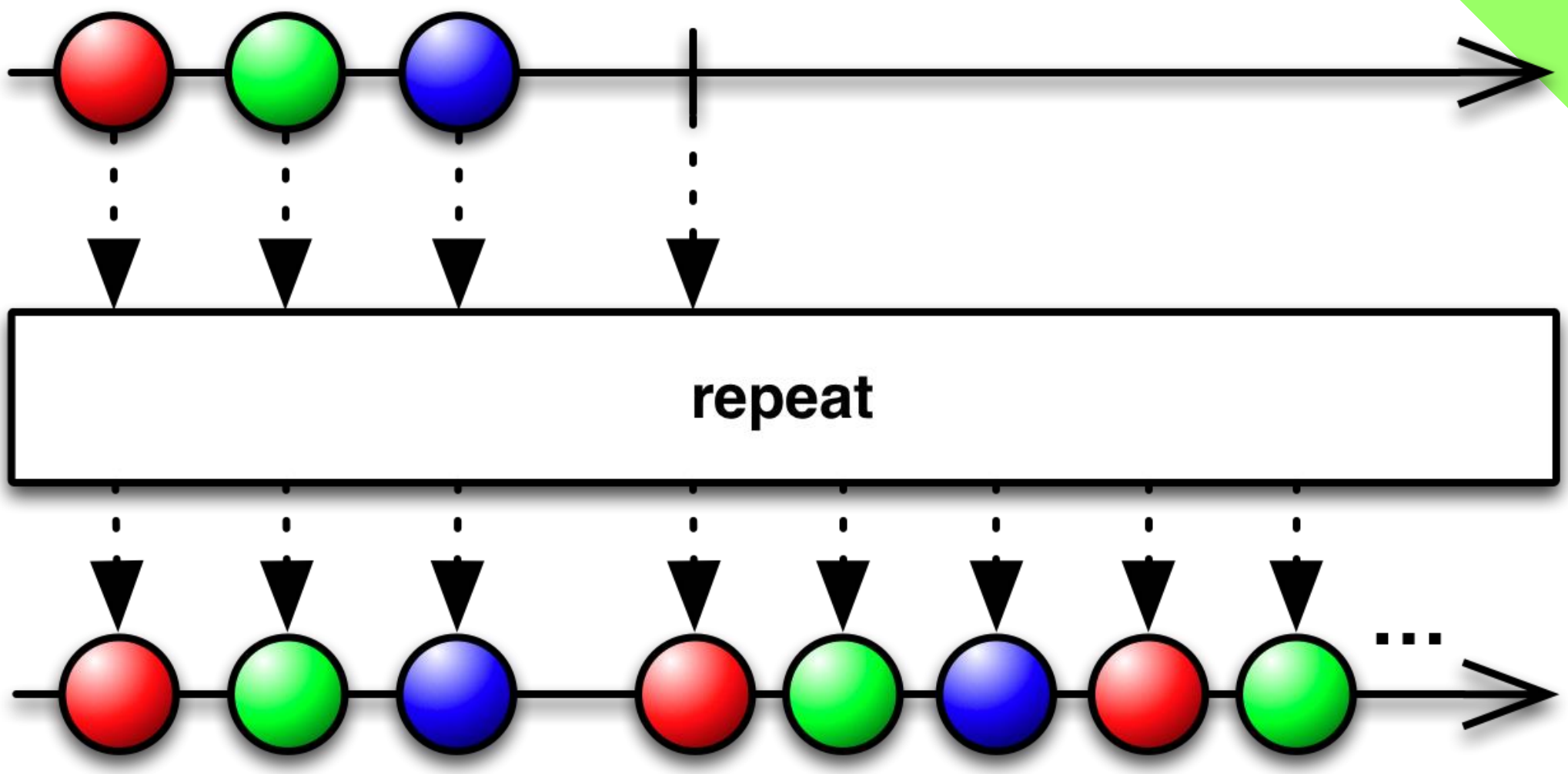


- Cas d'usage : ajout d'un préfixe.

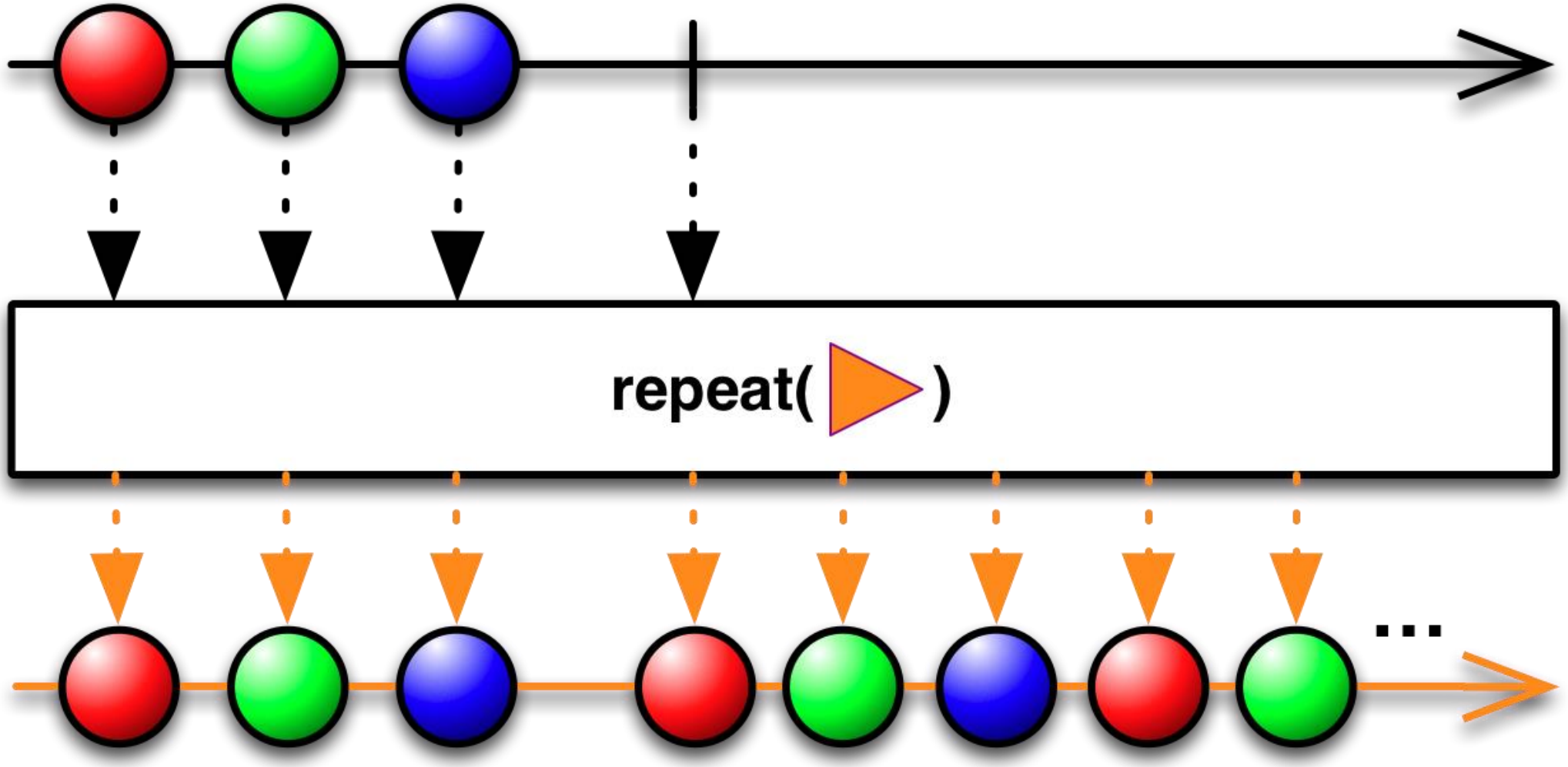

startWith( )

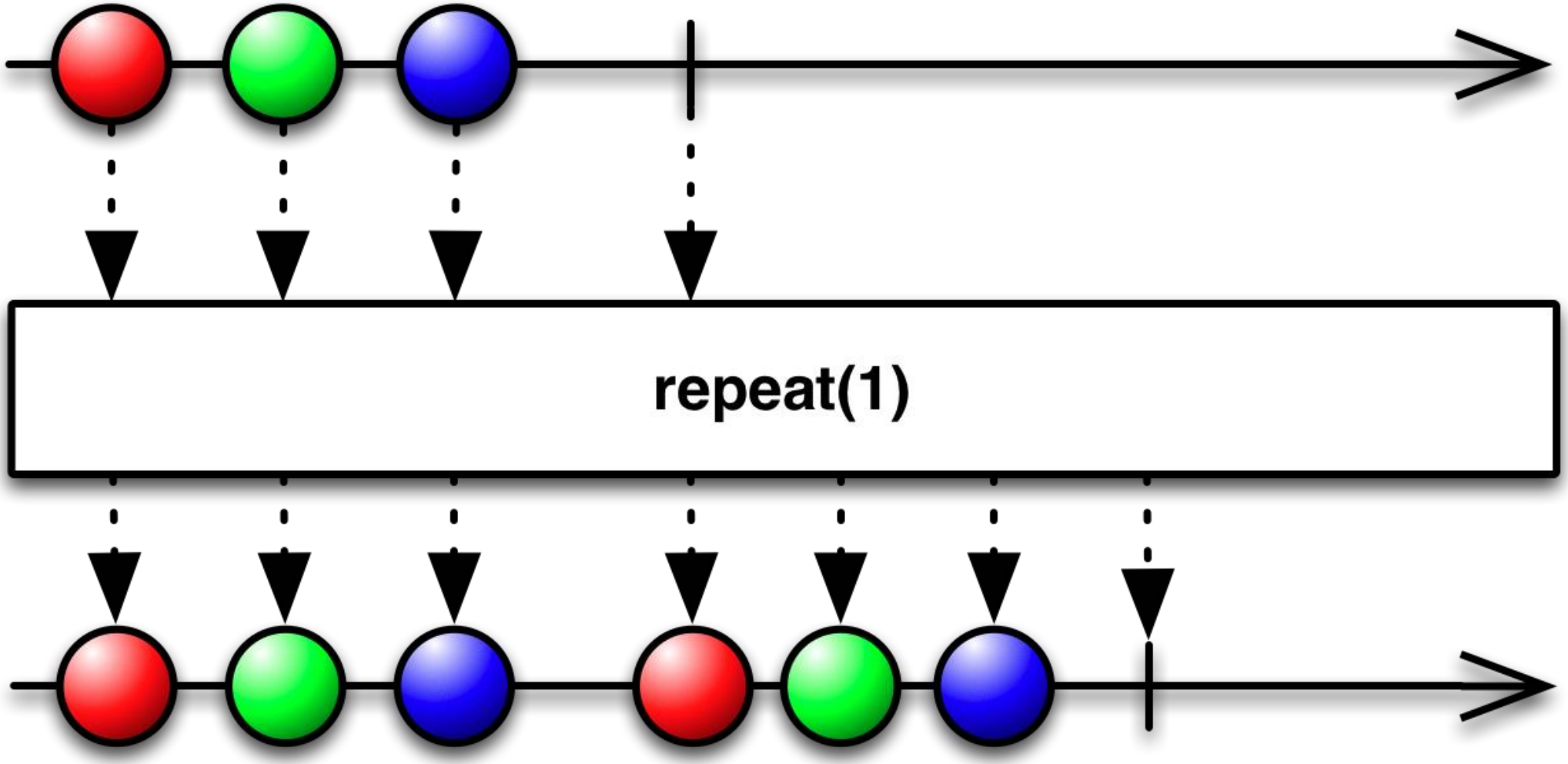


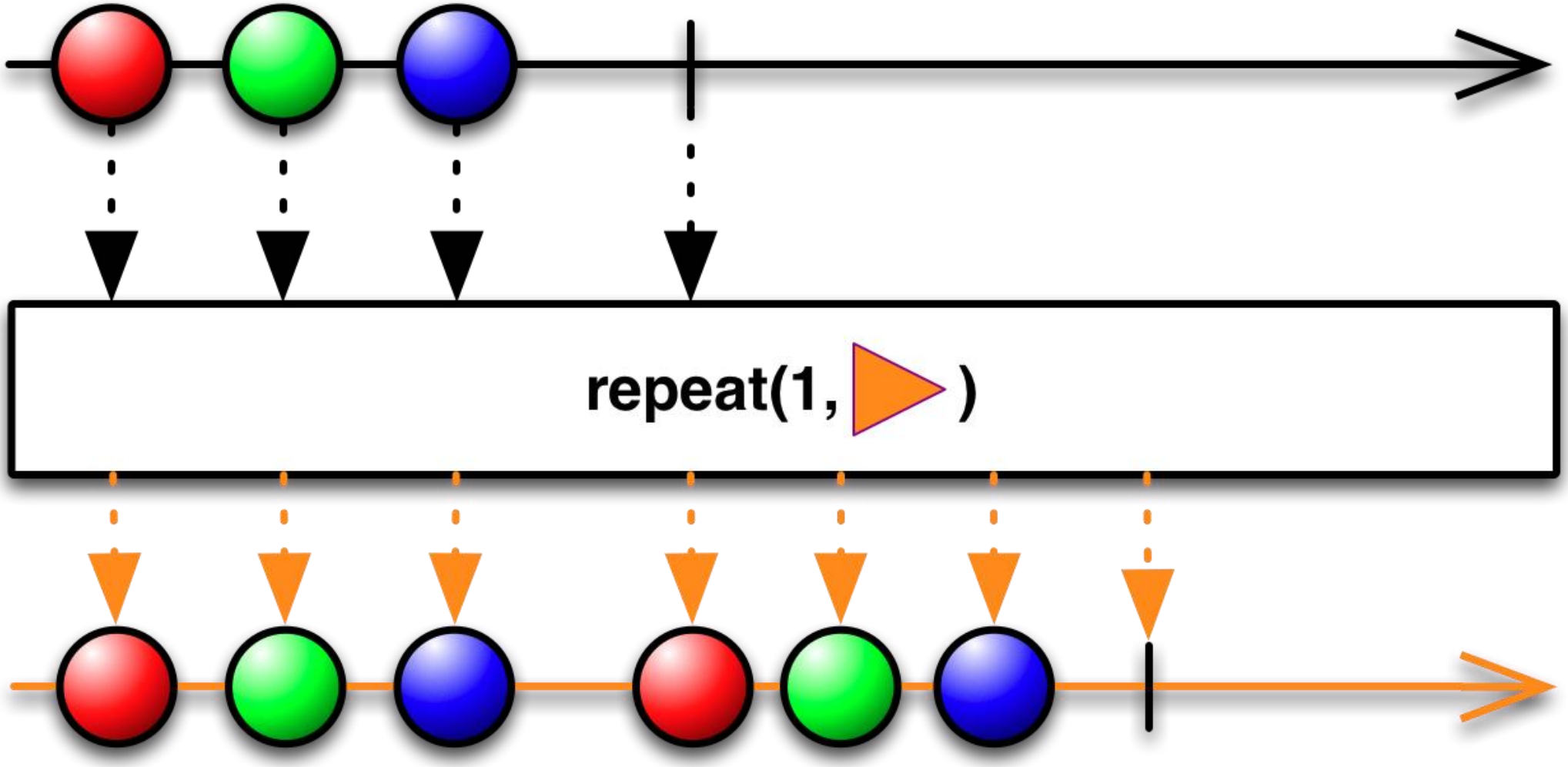
- Cas d'usage : Transformer un Observable chaud en Observable froid.

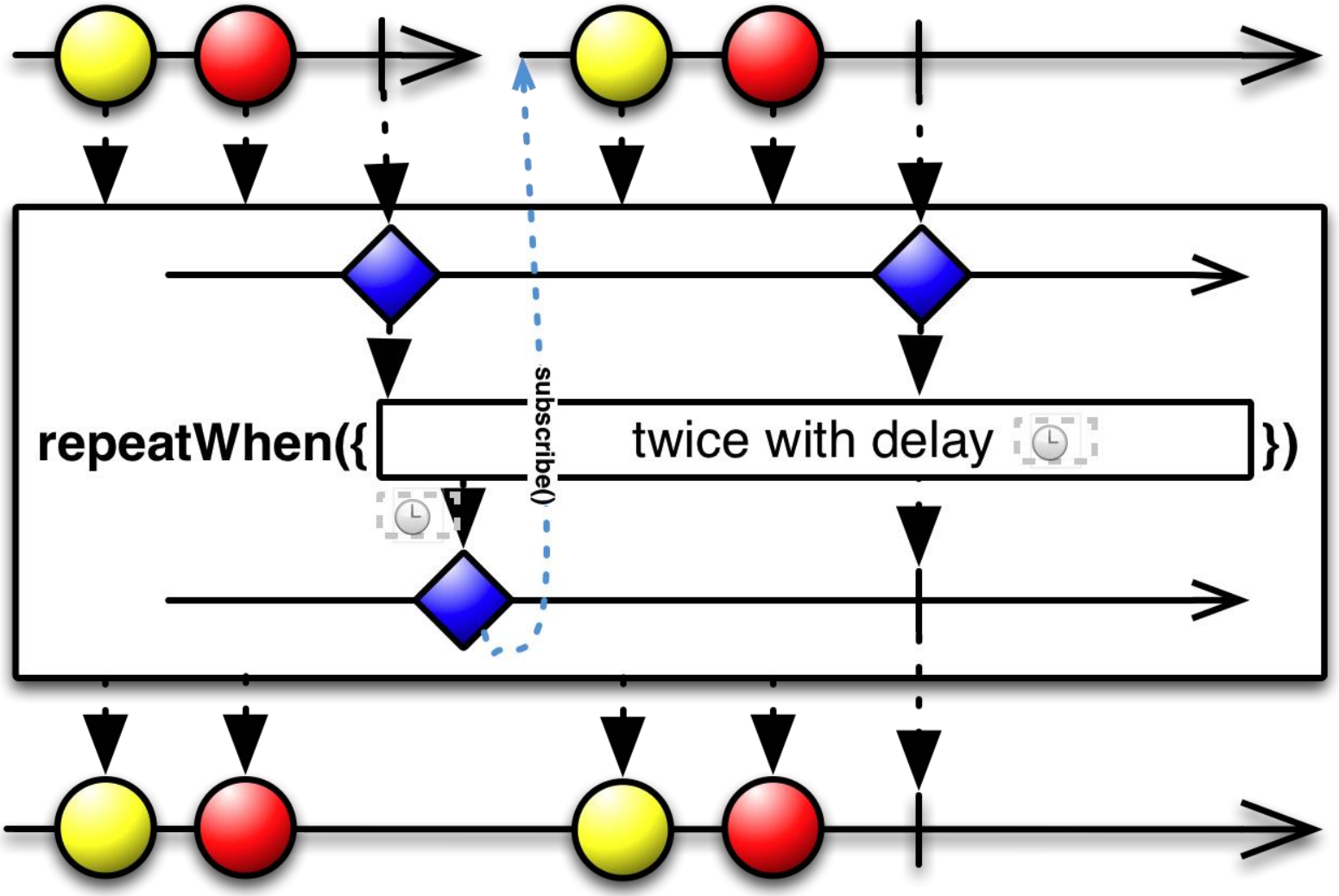


- Cas d'usage : Boucle sur un ensemble de valeurs.


repeat( )

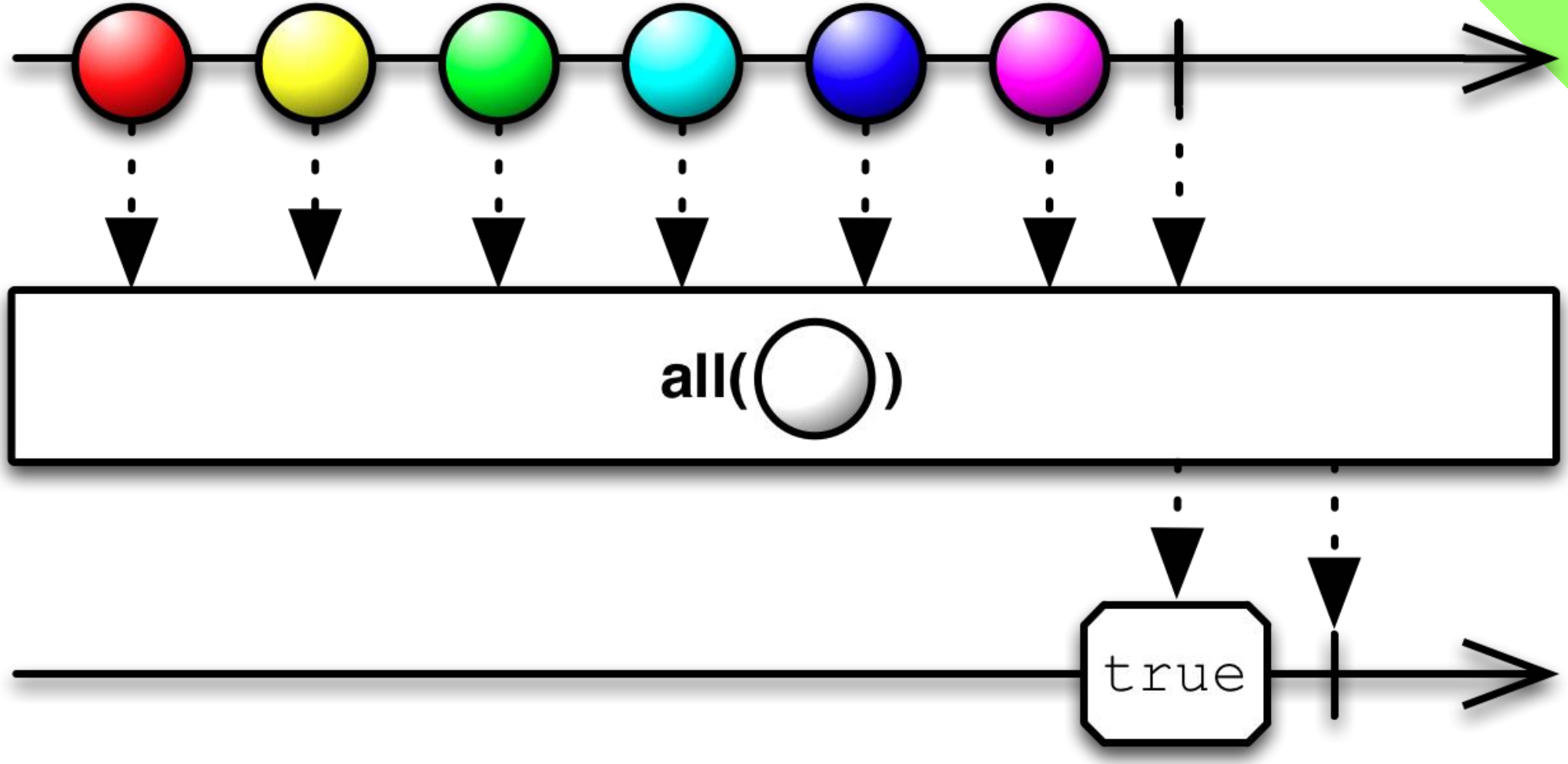

repeat(1)

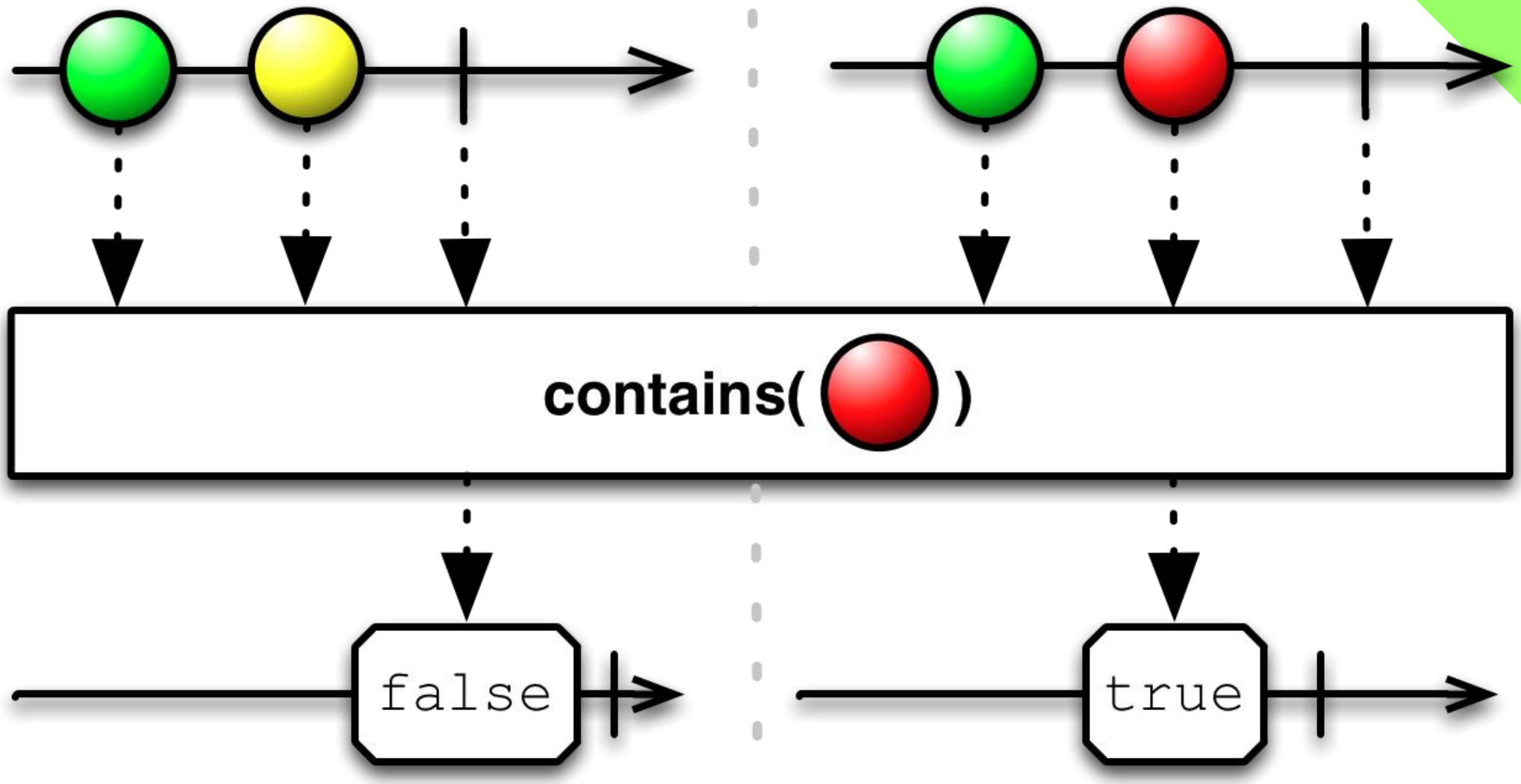

repeat(1,)

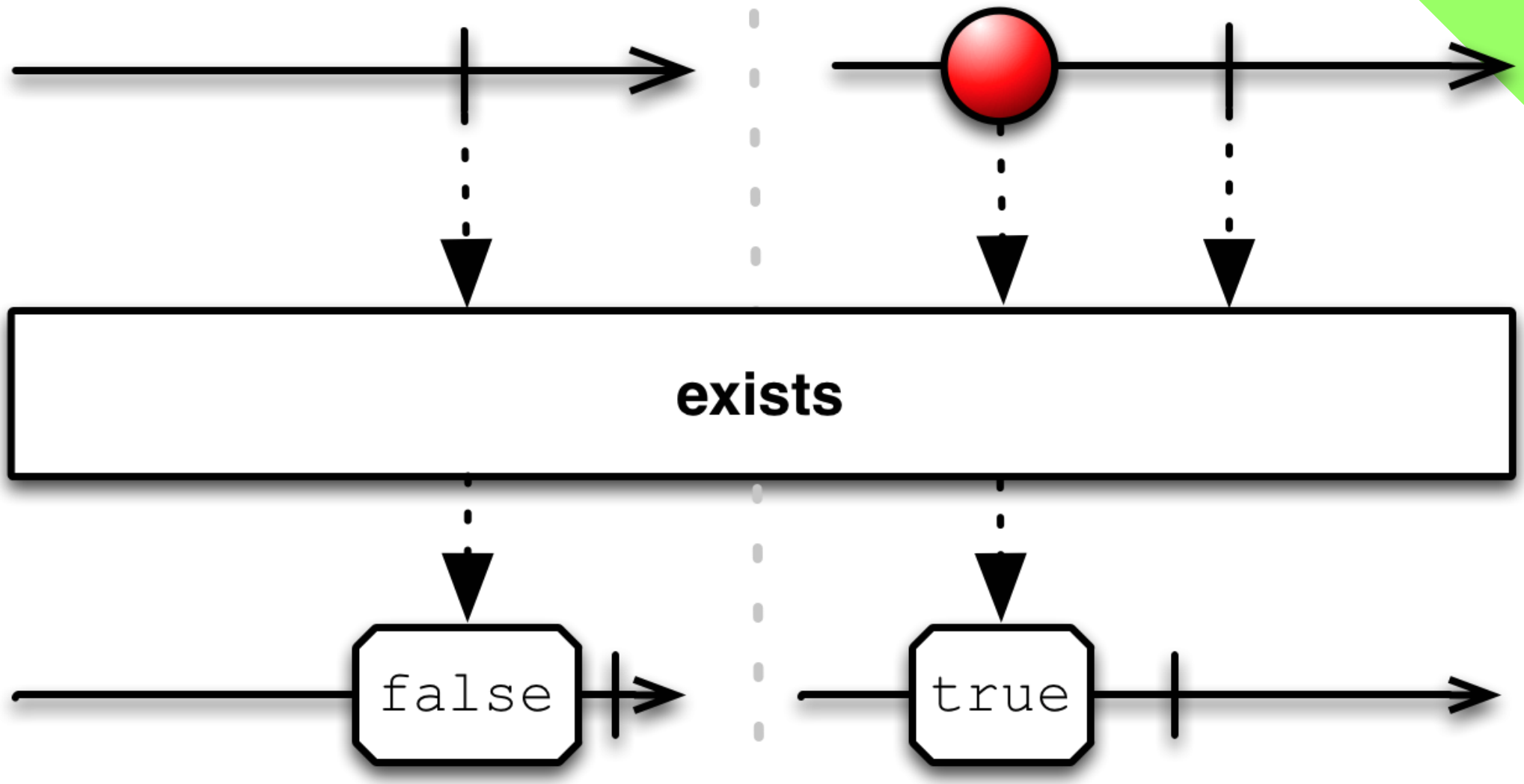


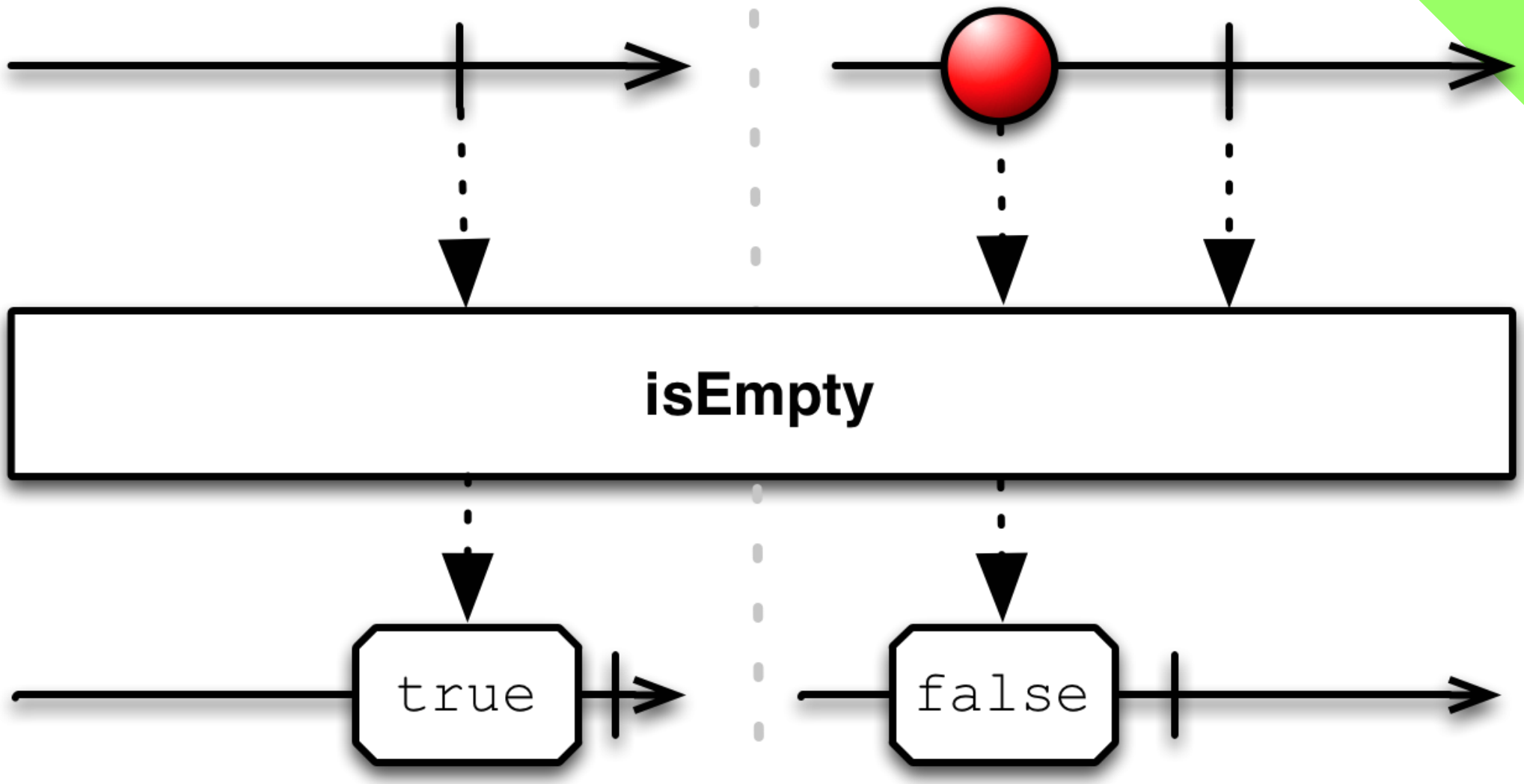
- Cas d'usage : Boucle conditionnelle sur un ensemble de valeurs.

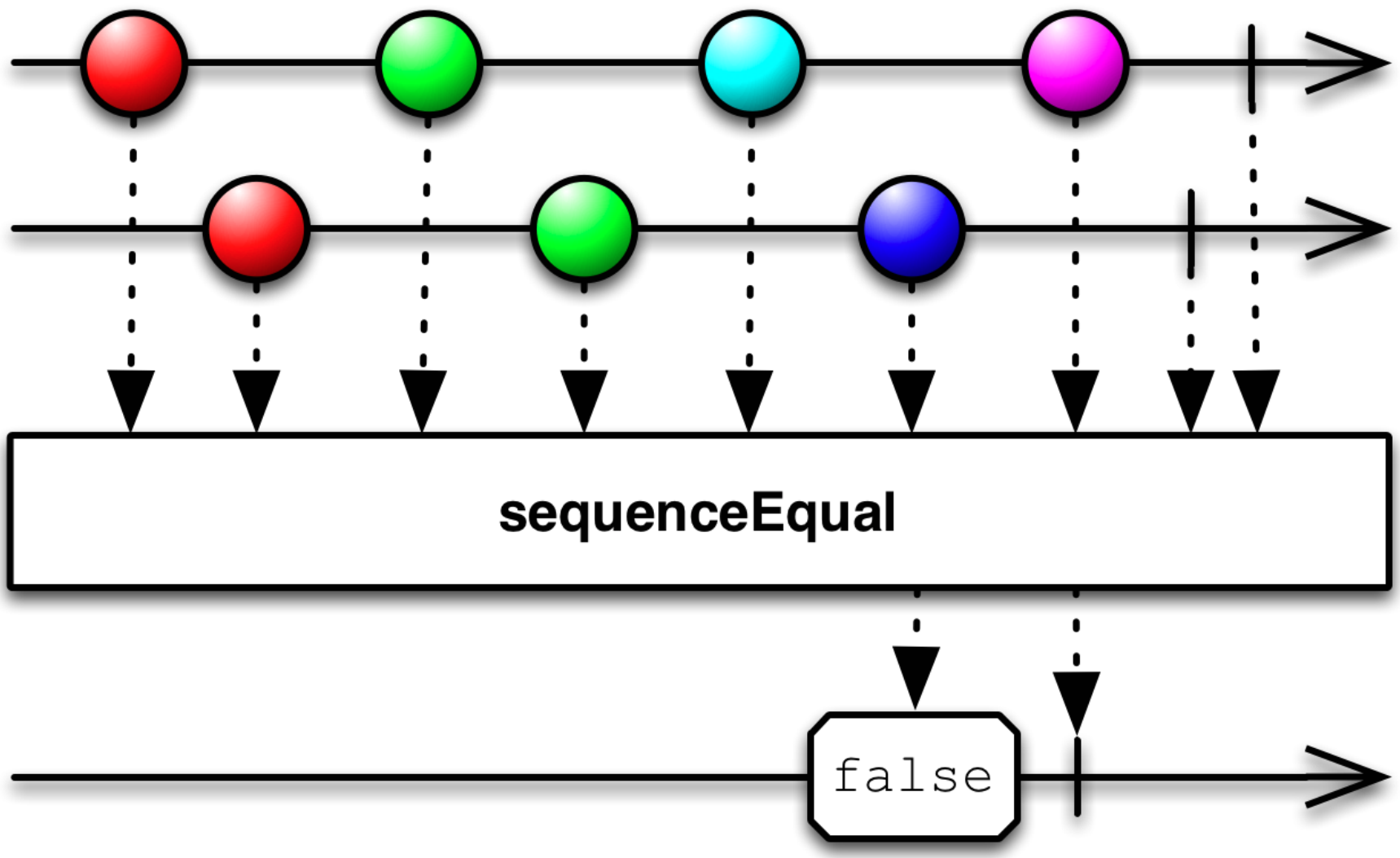
Predicats


all( )
true

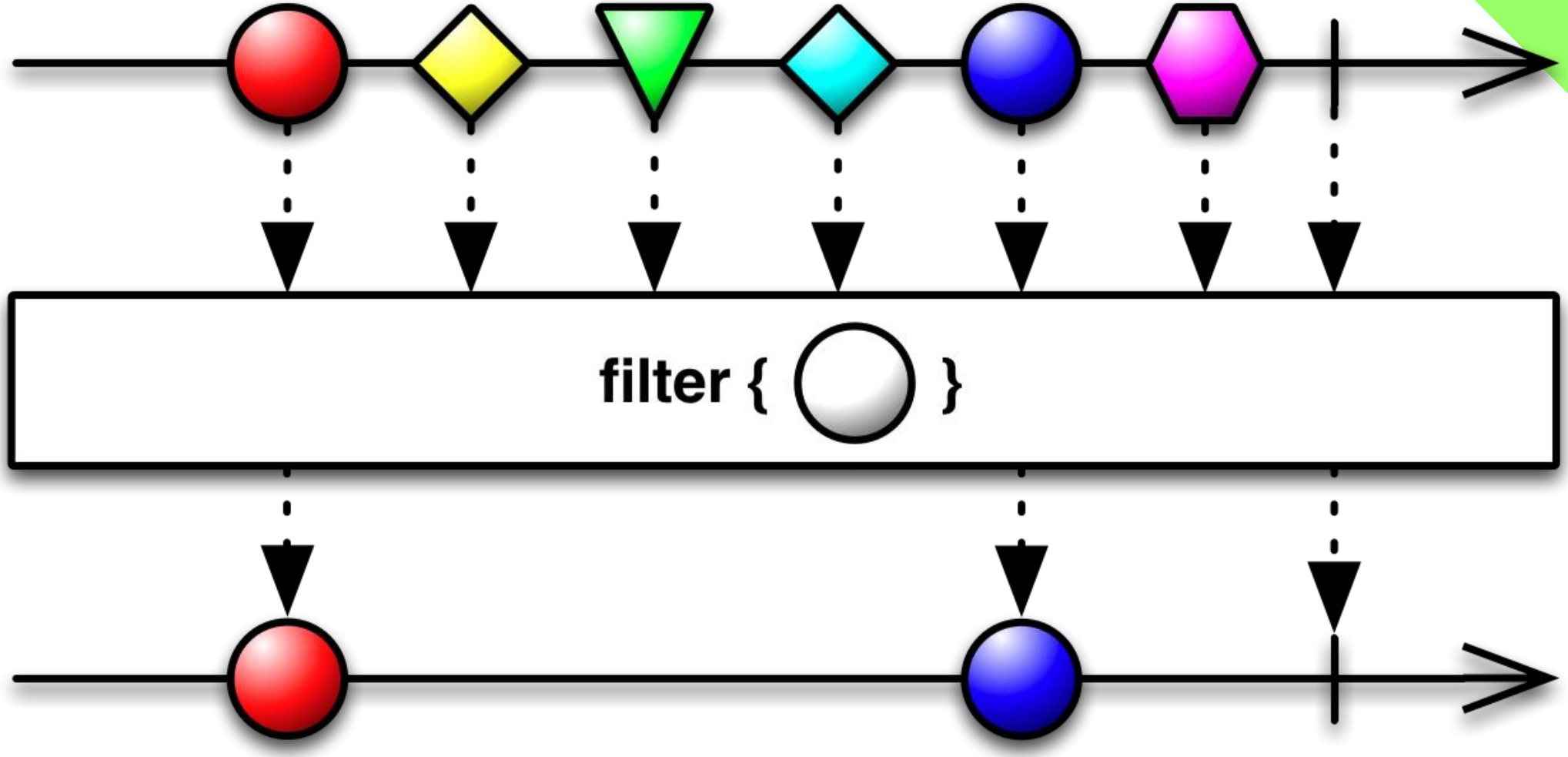

contains( )
false
true

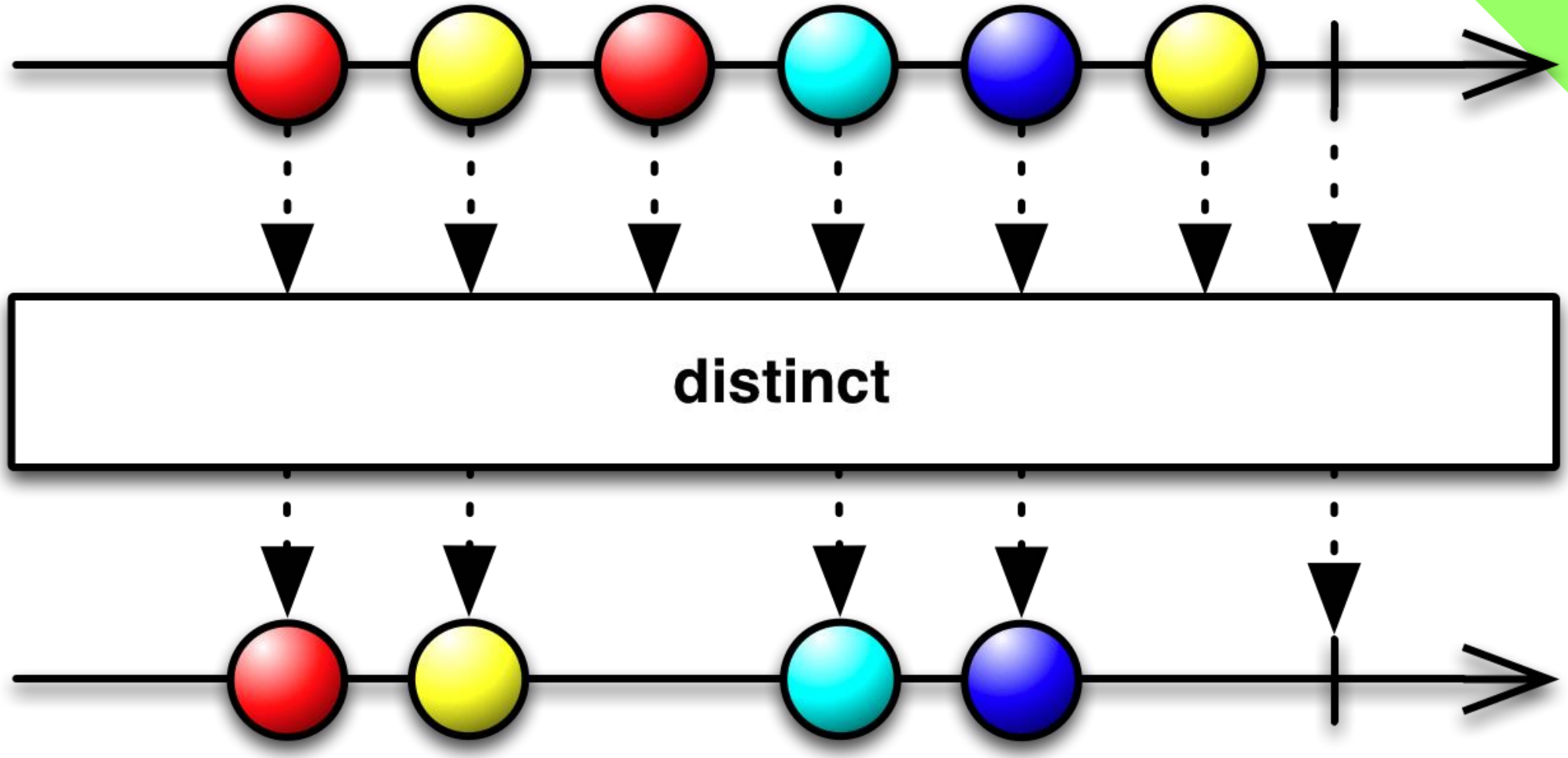

exists
false
true

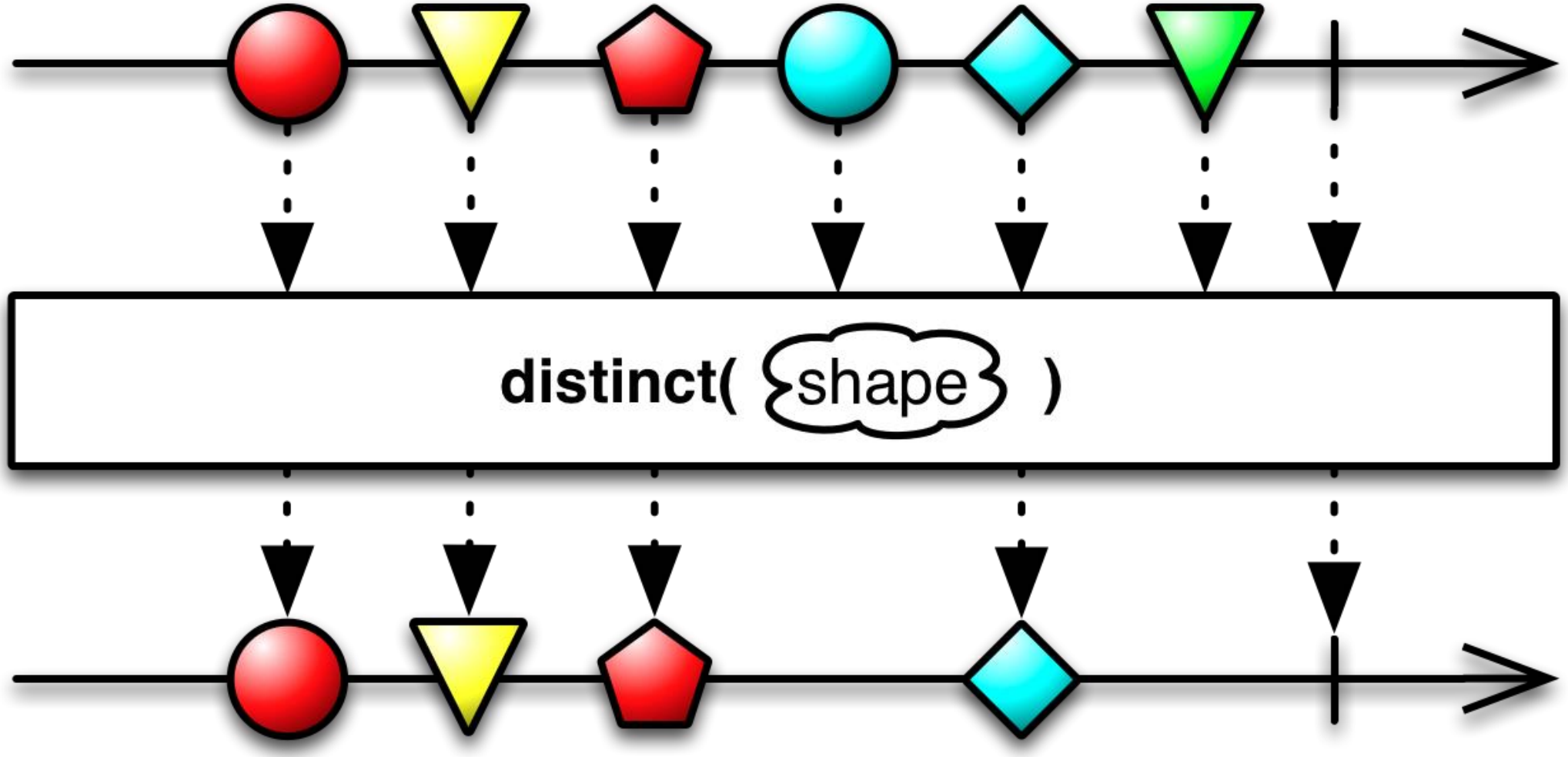

isEmpty
true
false

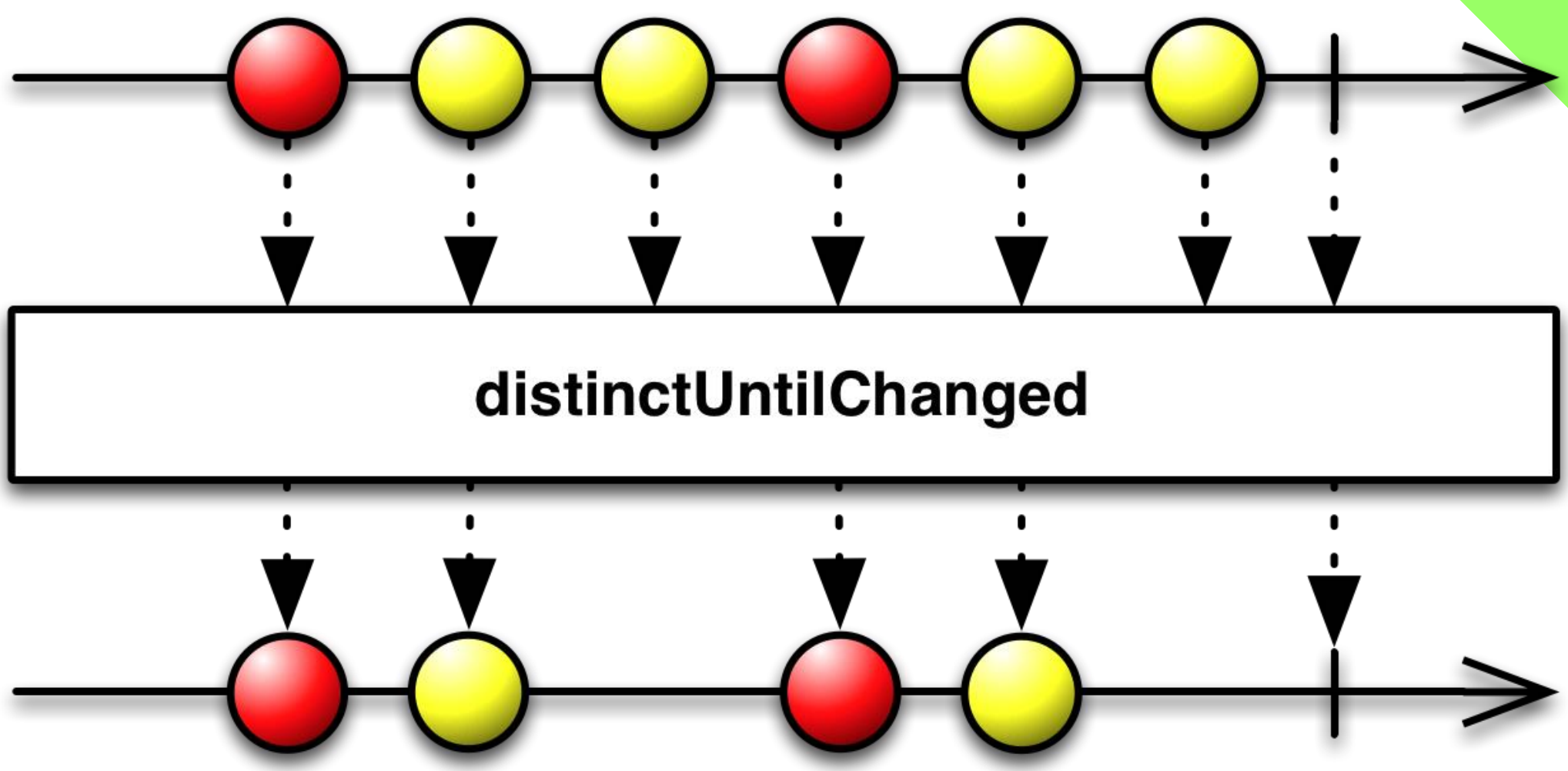

sequenceEqual
false

Filtrage

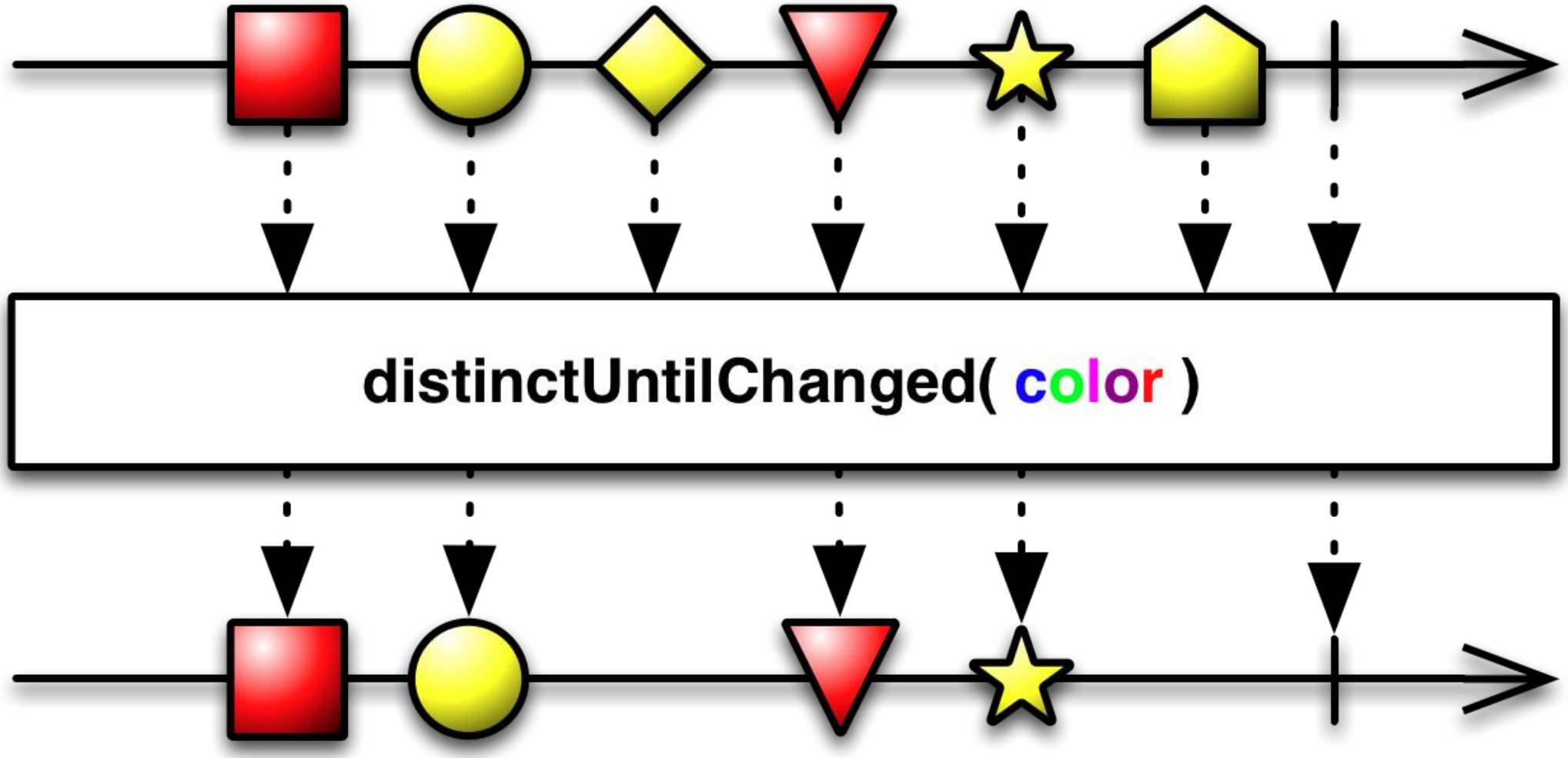

filter { }

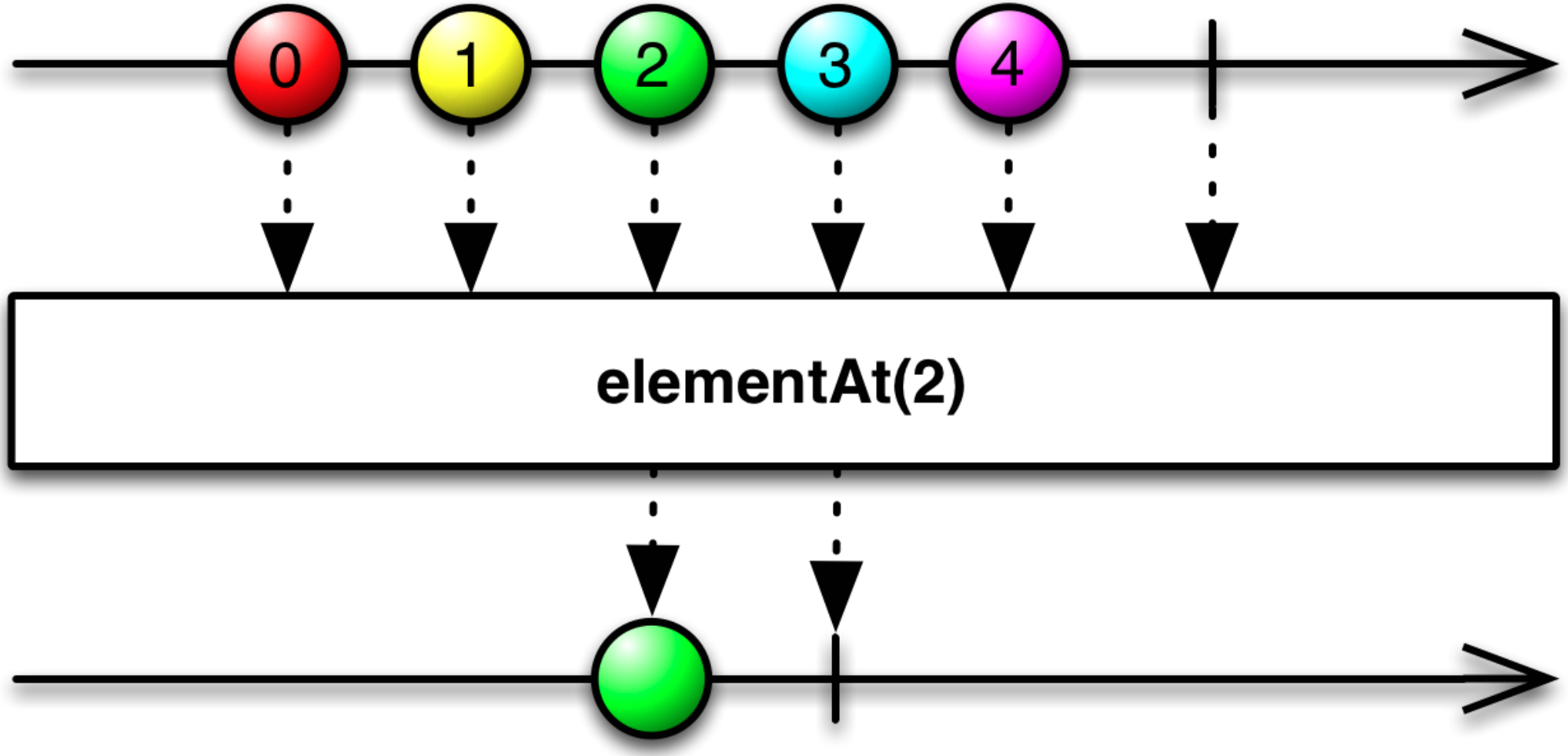

distinct

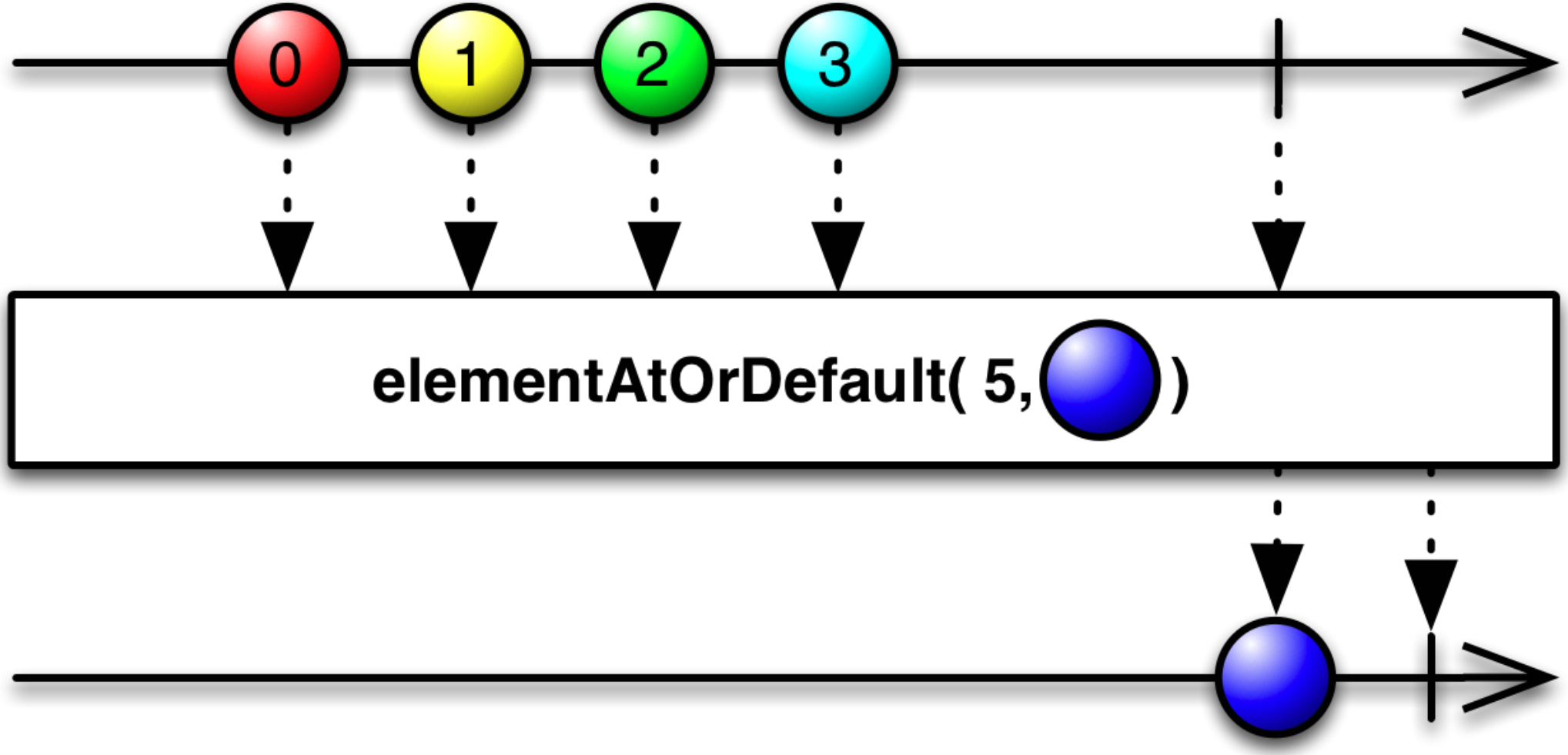

distinct(shape)

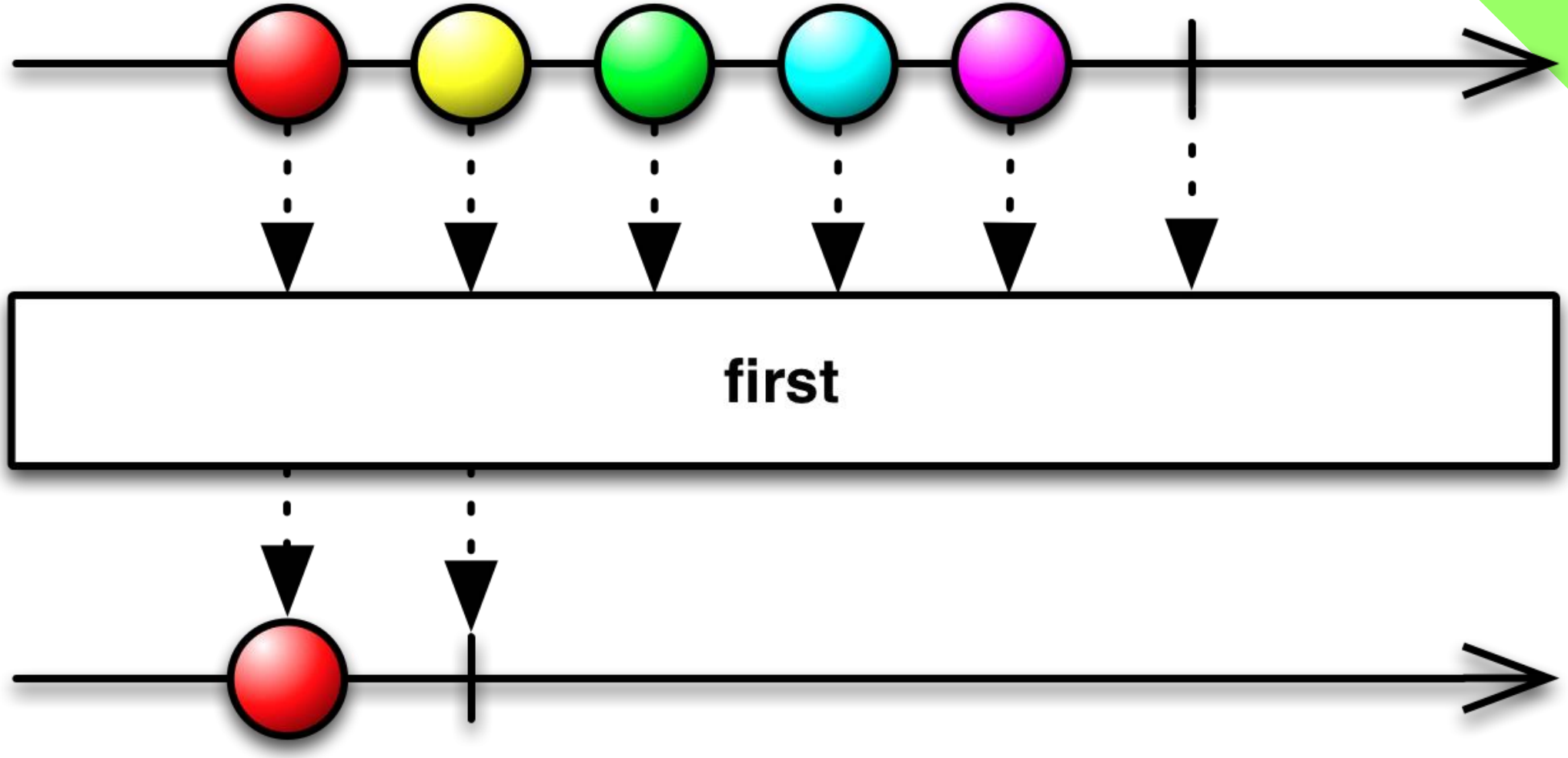


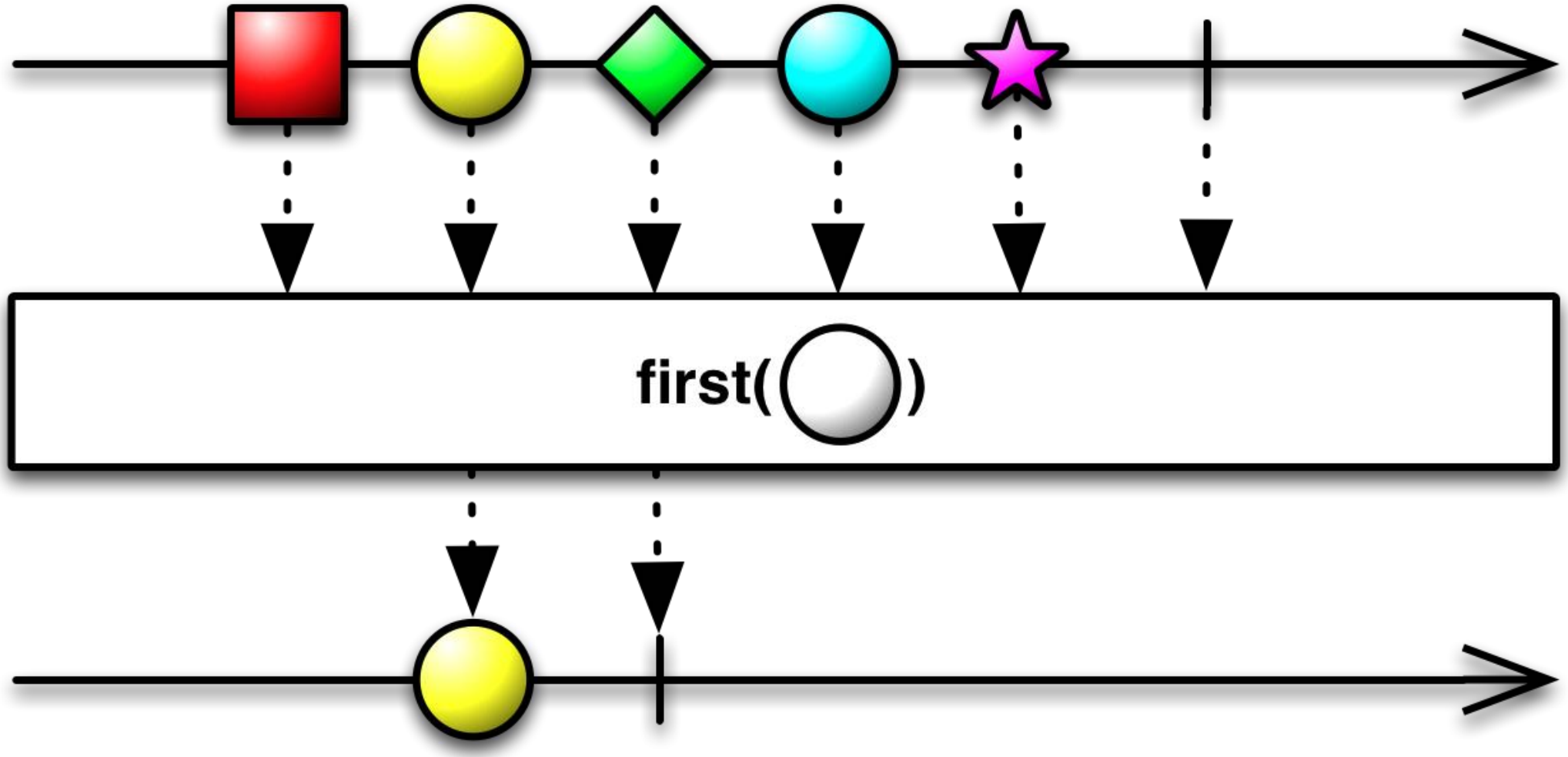
- Cas d'usage : détection de fronts.

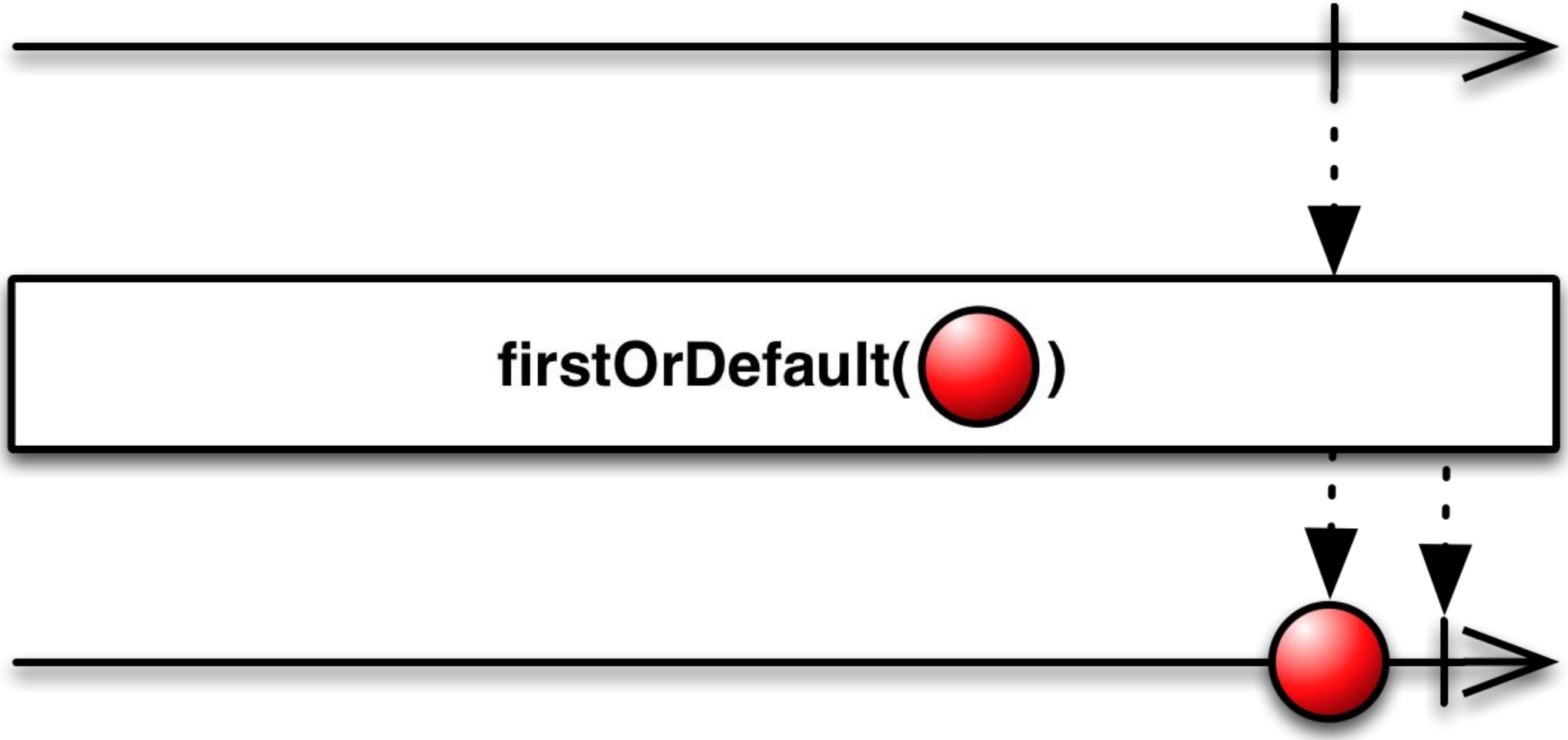

distinctUntilChanged(color)

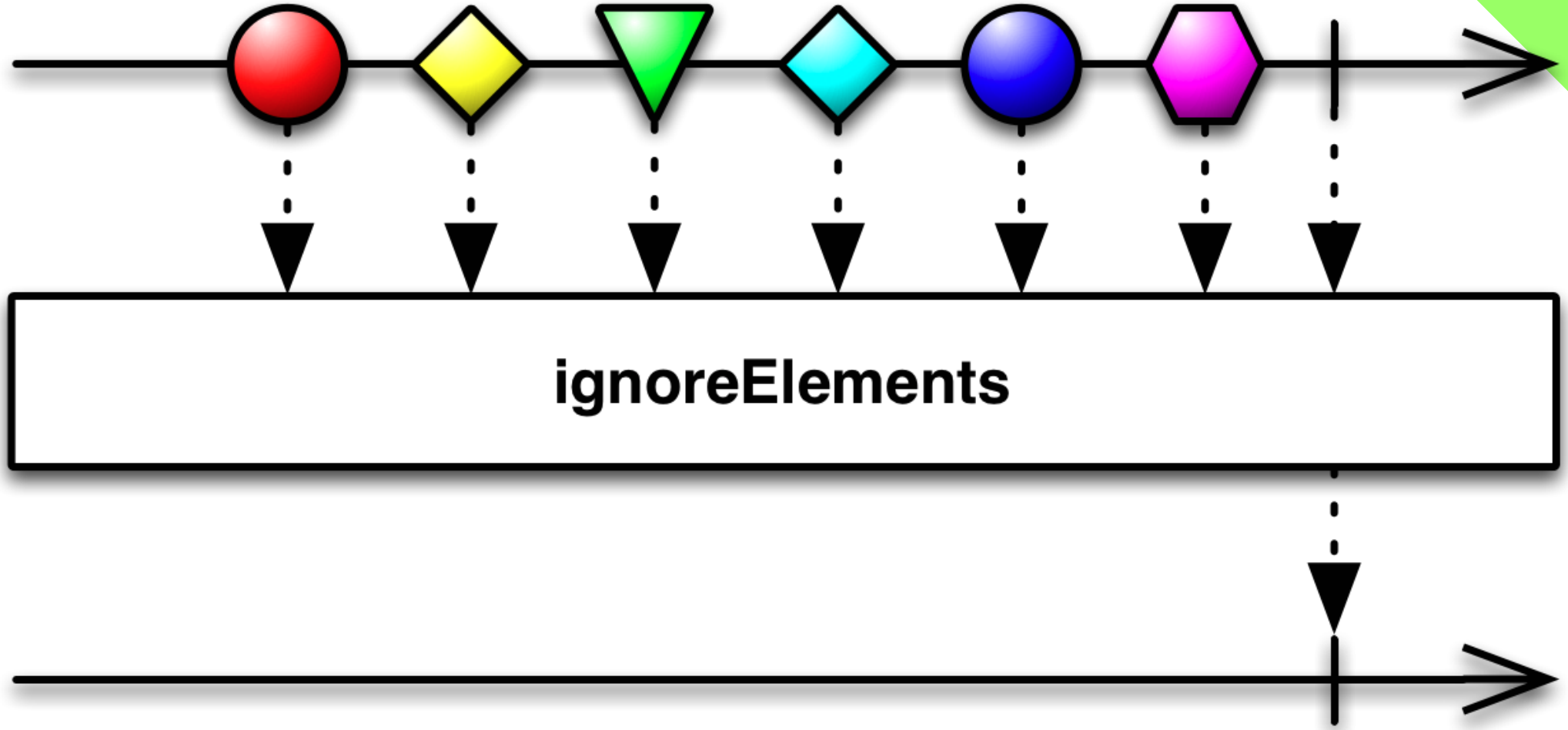

0
1
2
3
4
elementAt(2)

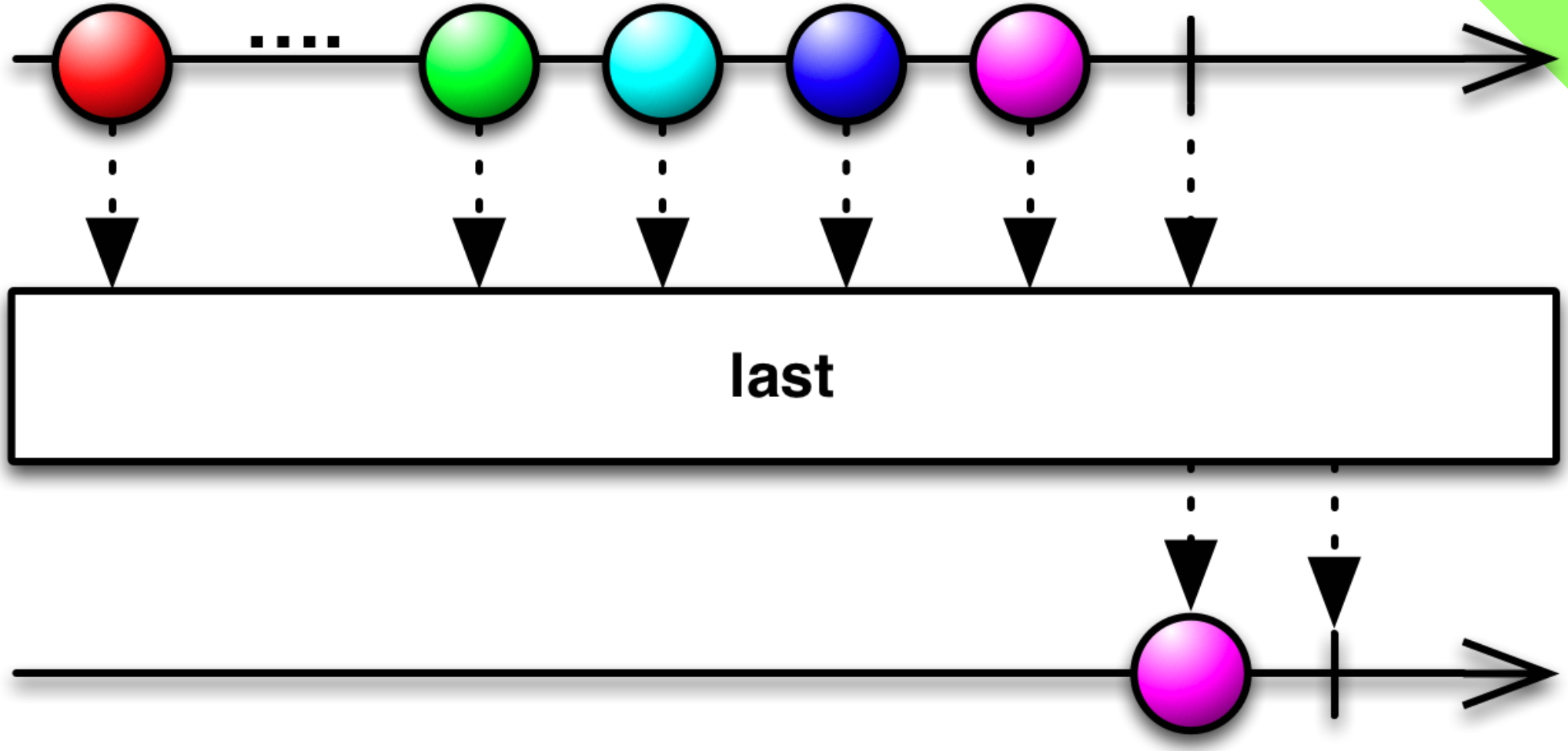

0
1
2
3
elementAtOrDefault(5,)

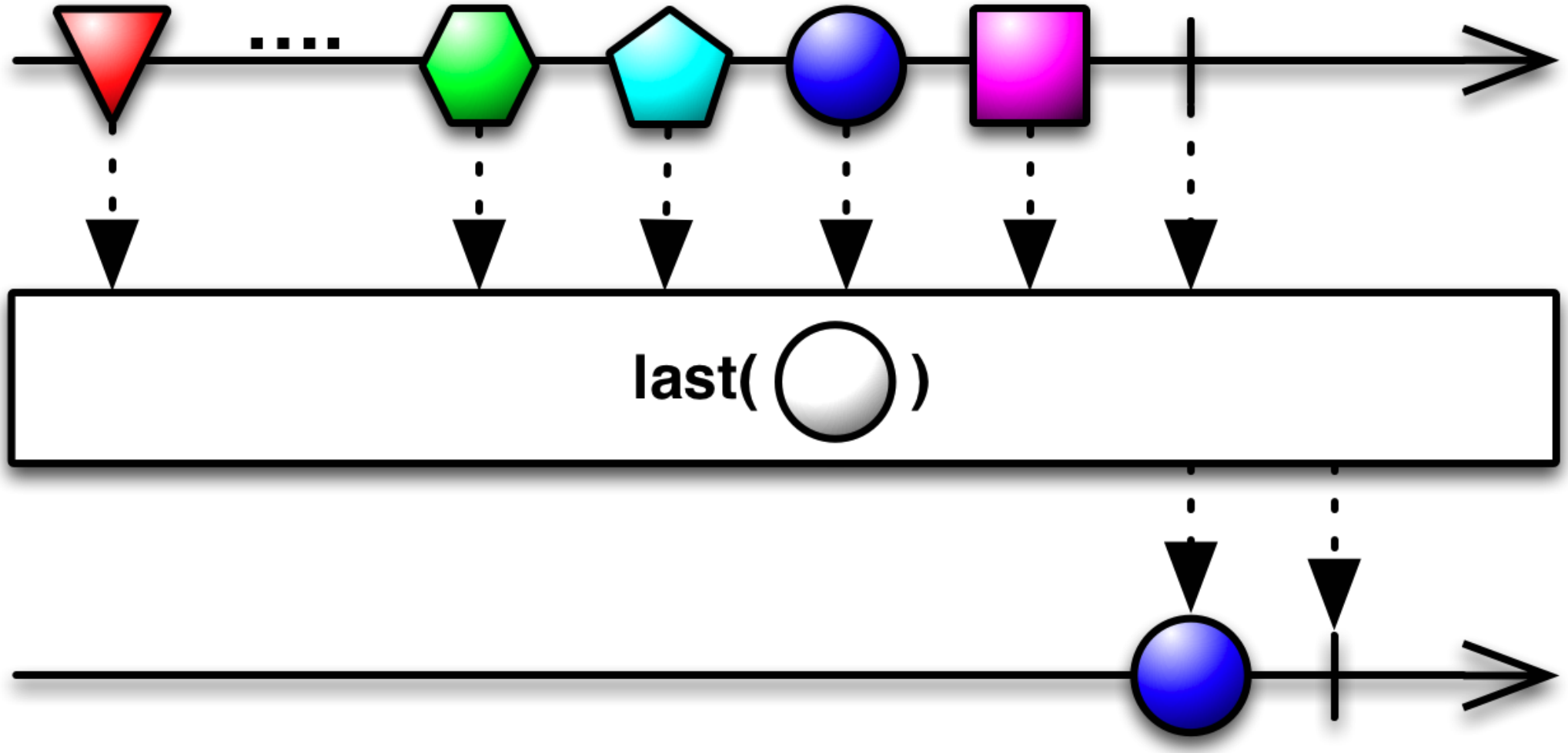

first

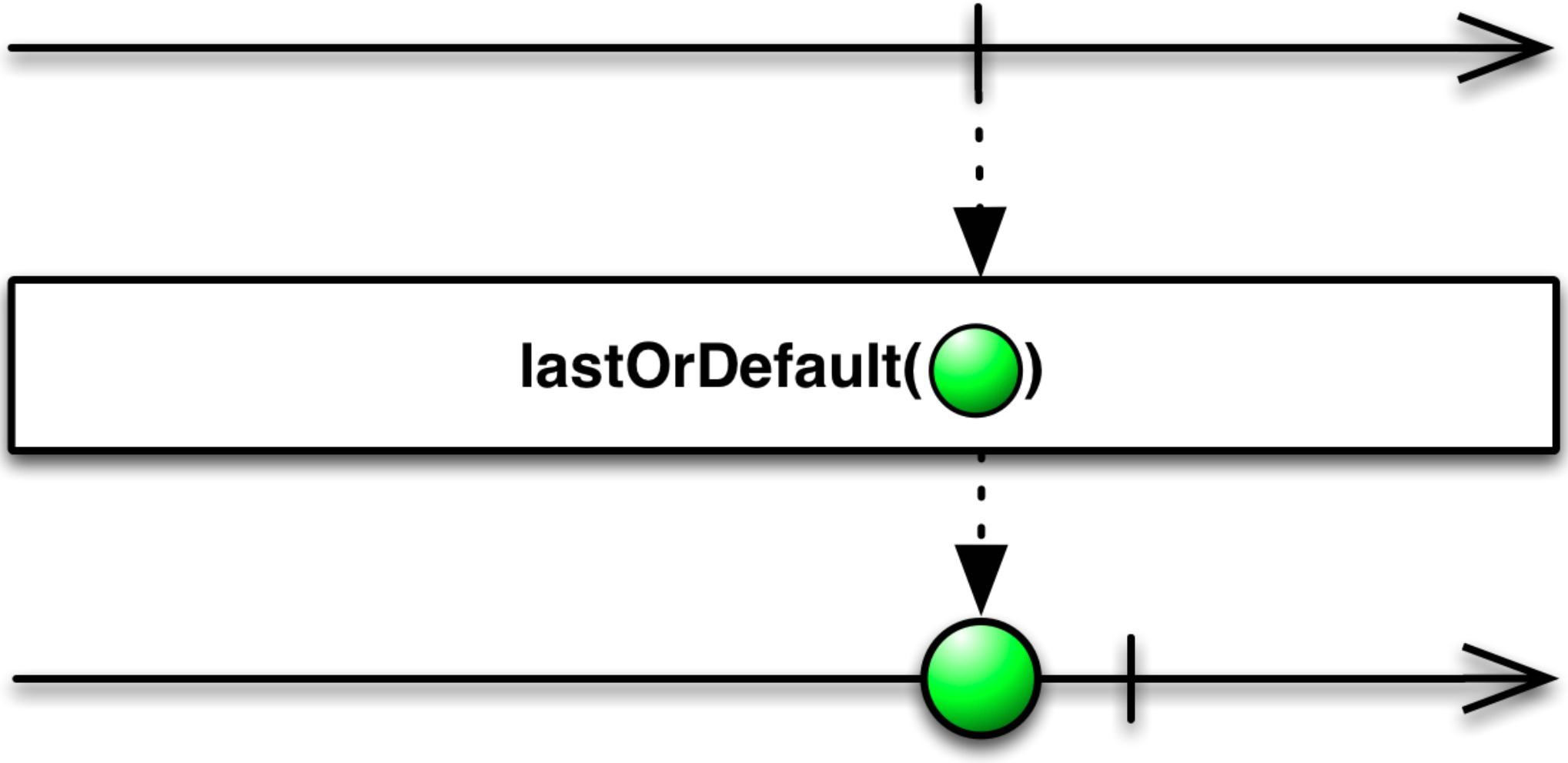

first( )

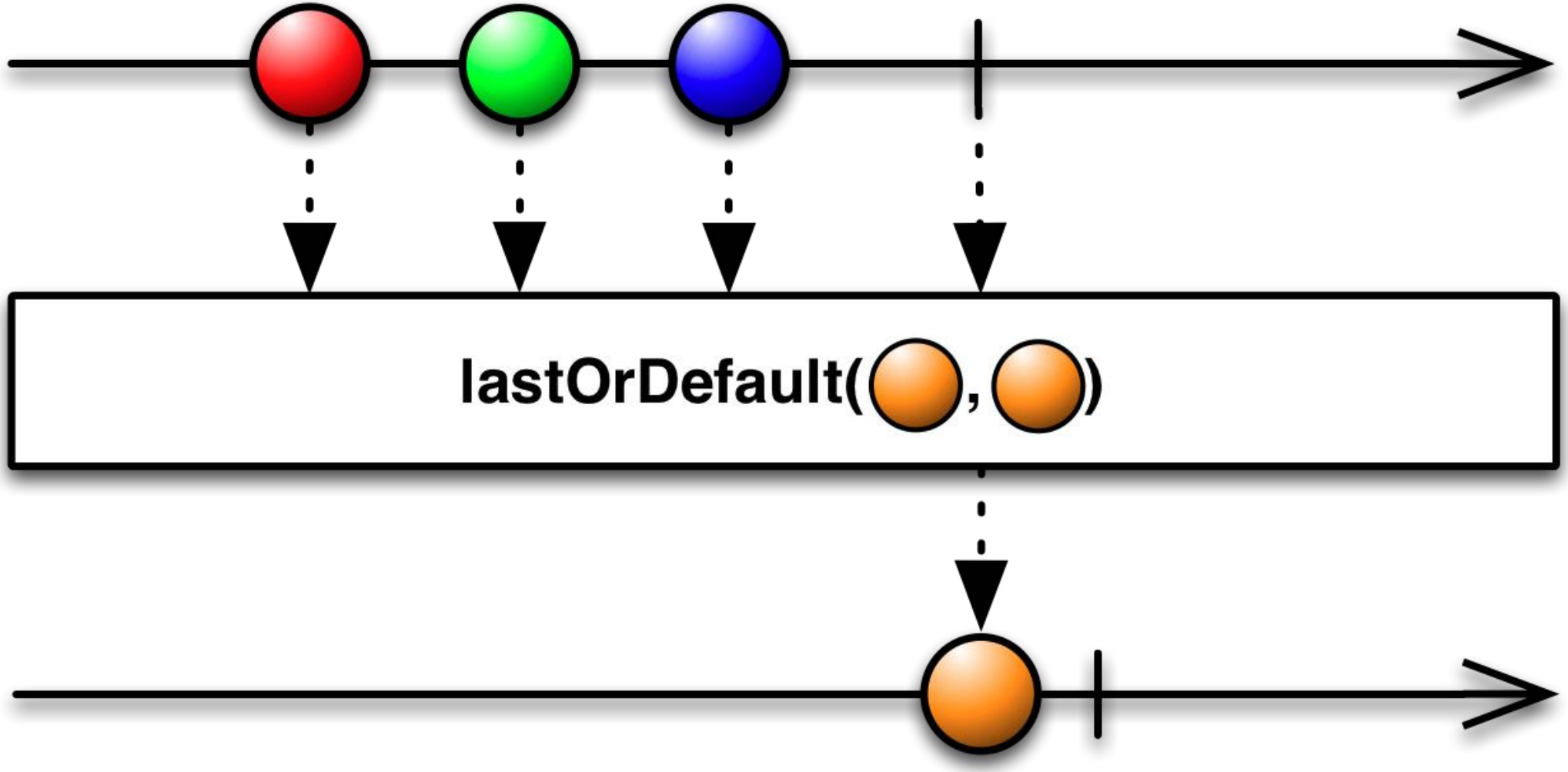

firstOrDefault( )

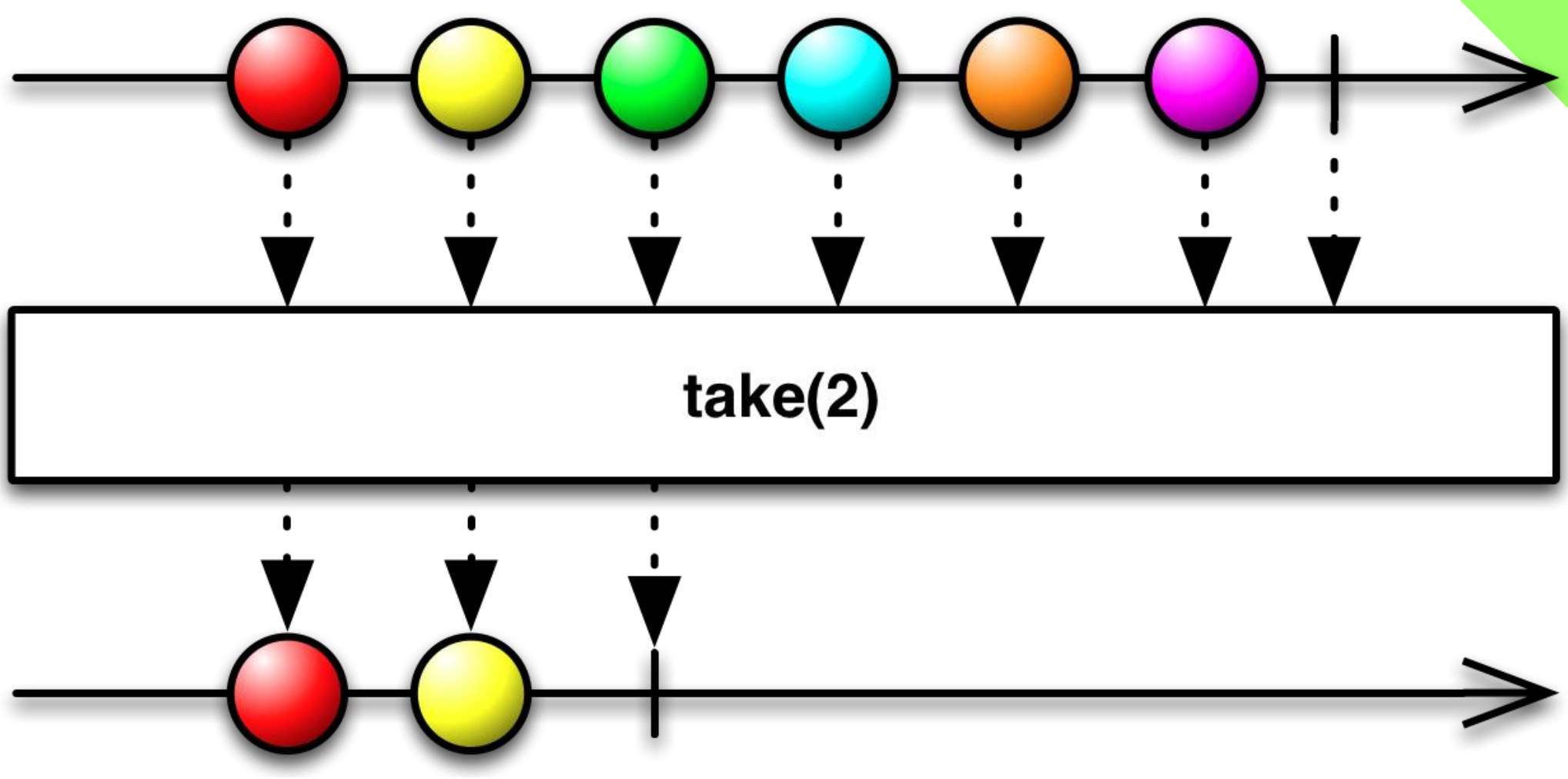

ignoreElements

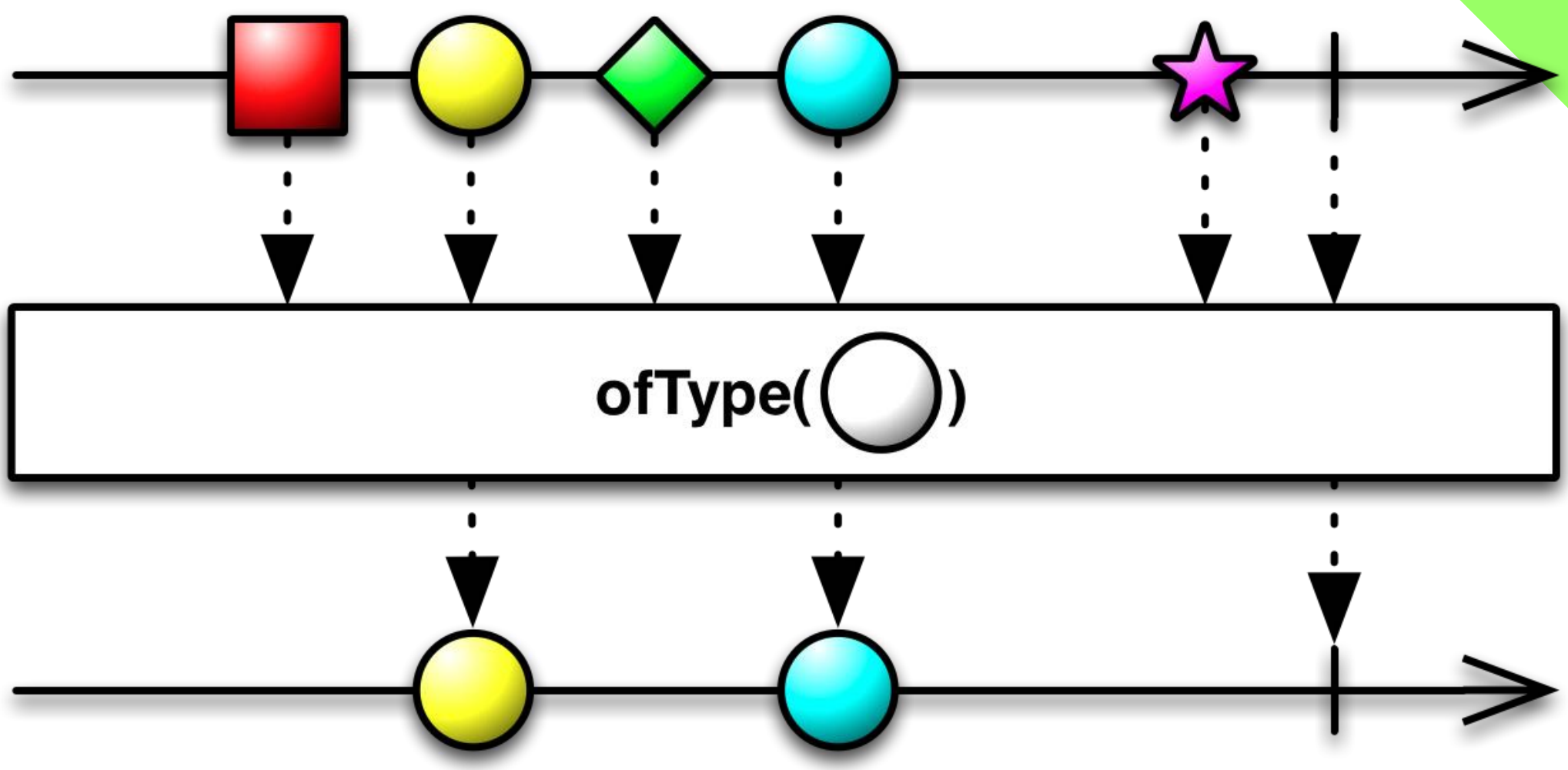

last


last( )

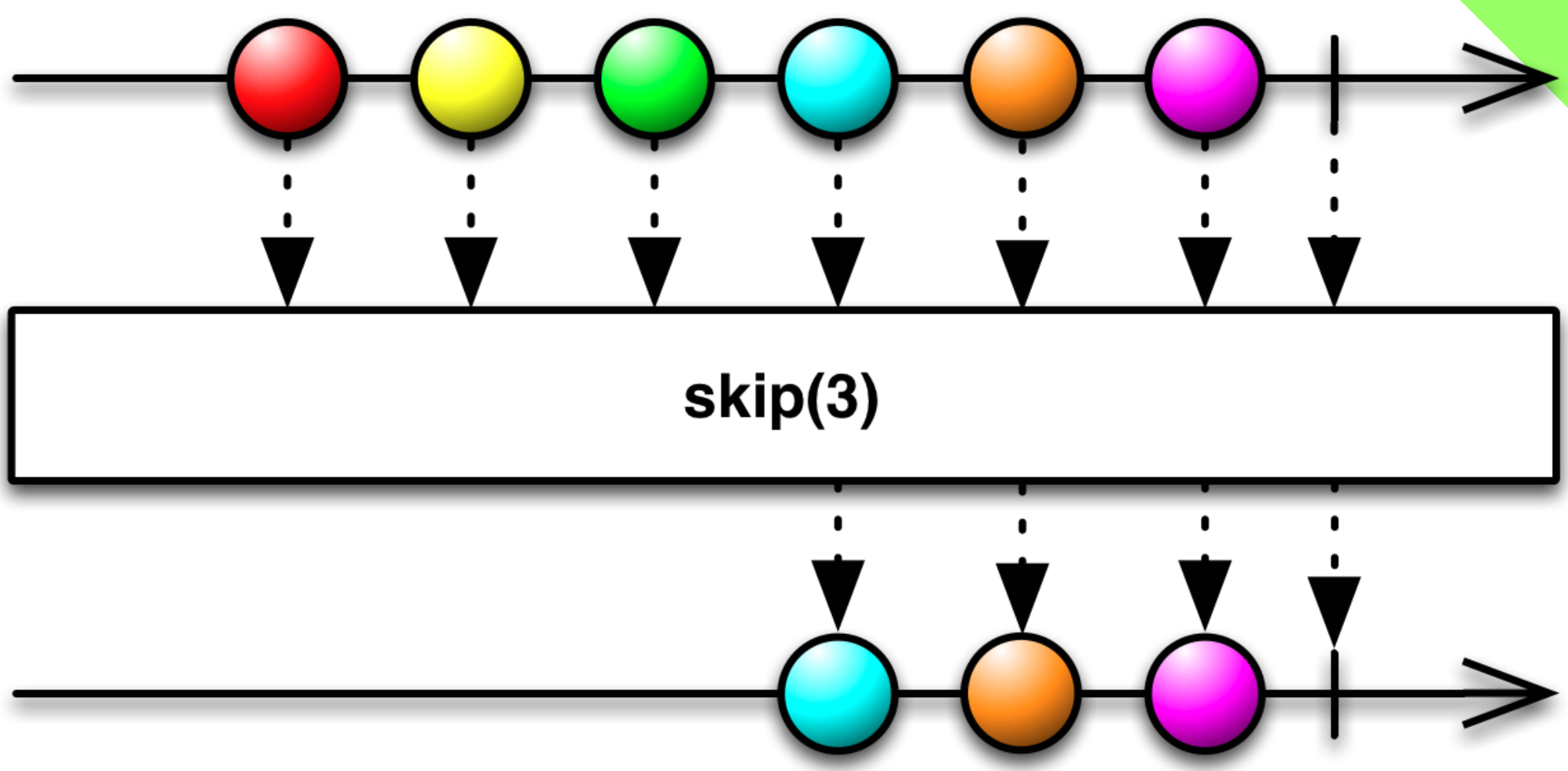

lastOrDefault()

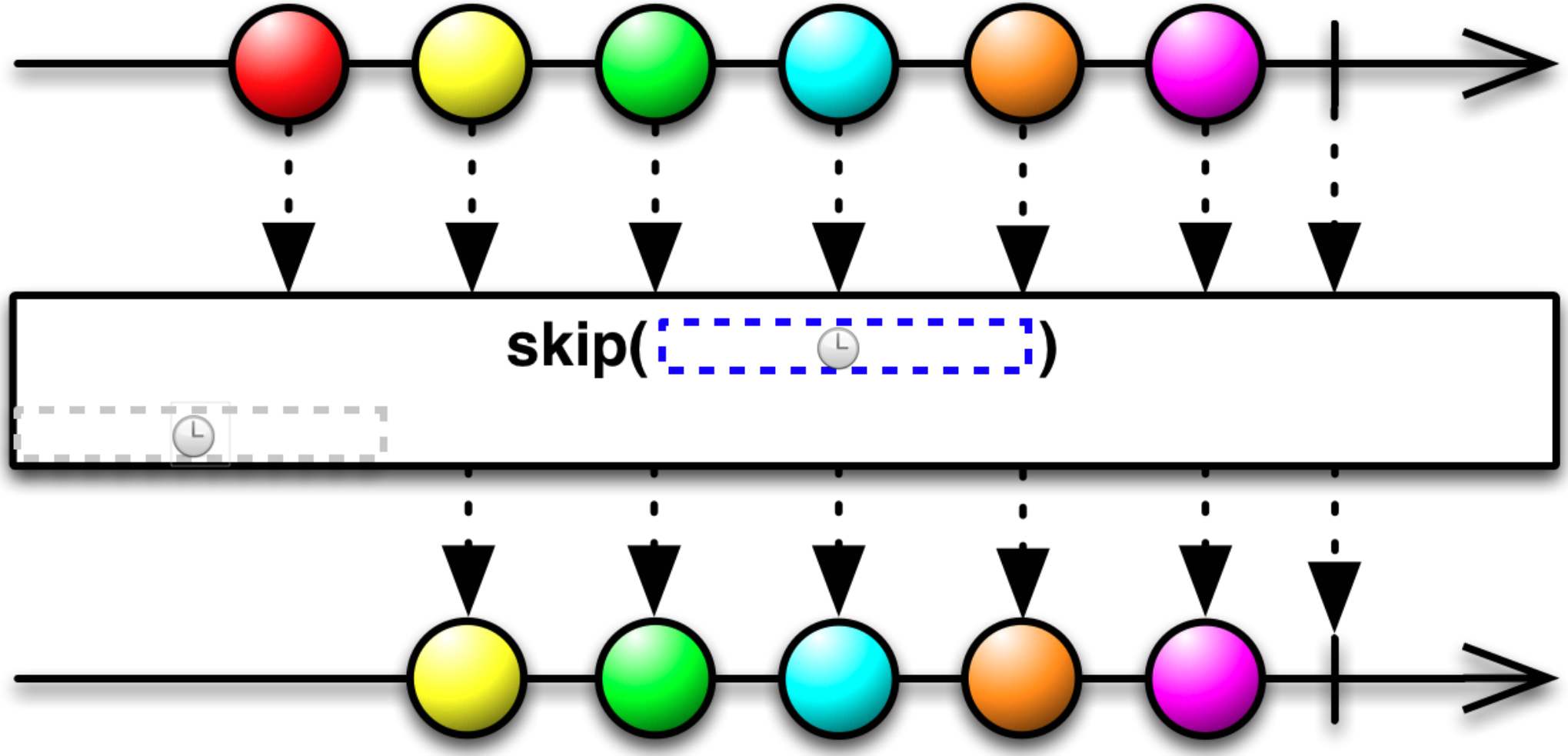

lastOrDefault(,)

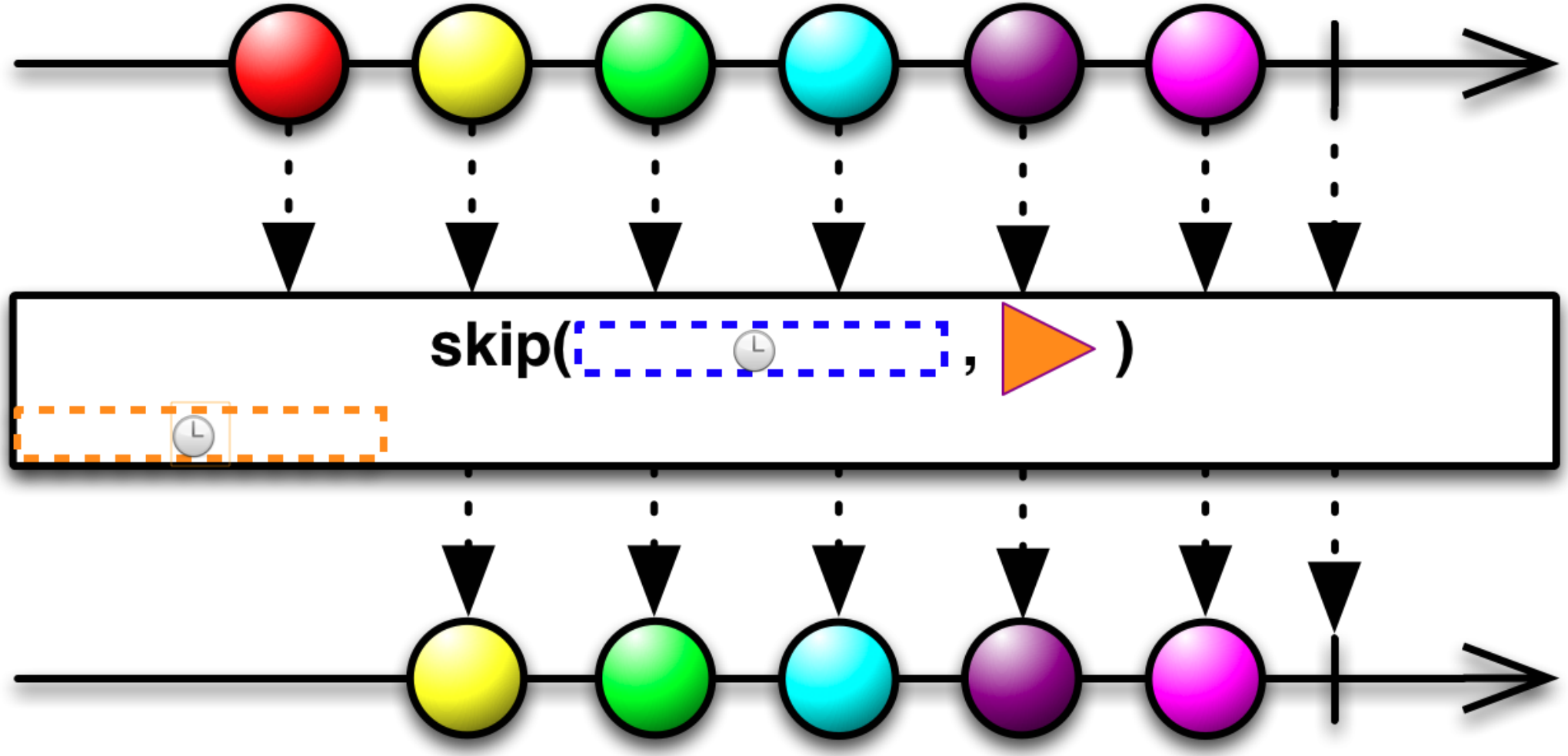

take(2)

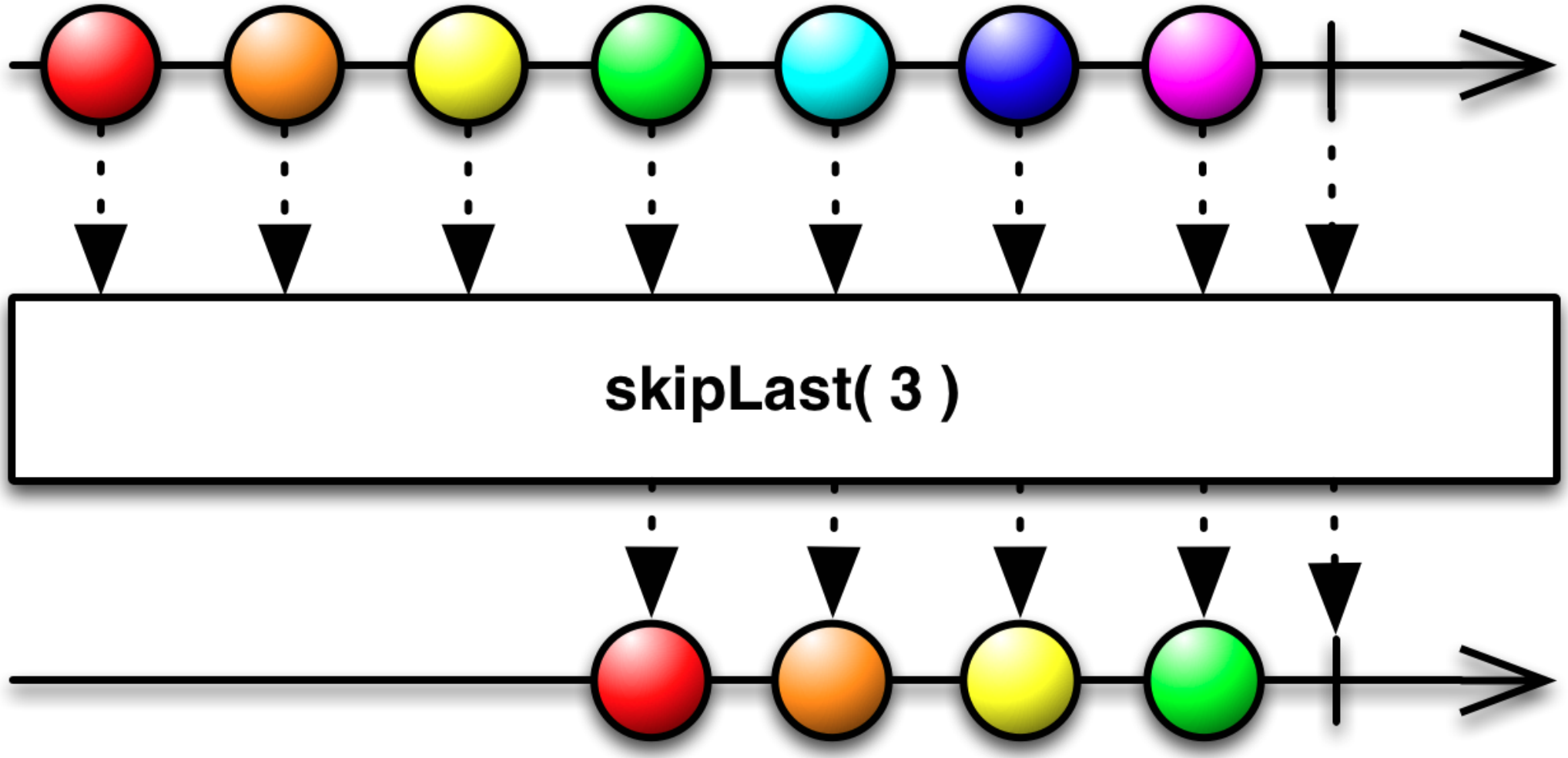


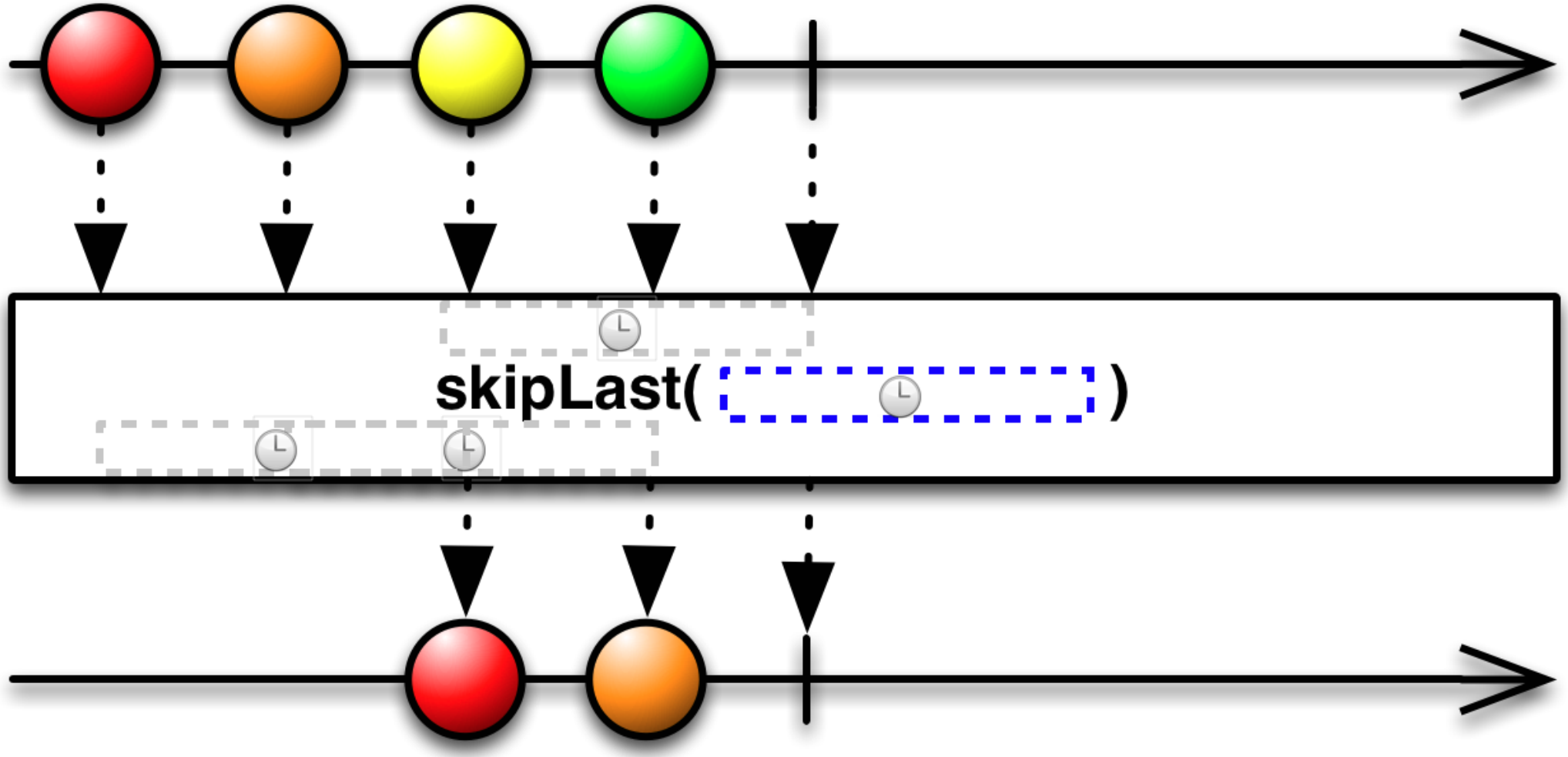
- Observable<**R**> ofType(final Class<**R**> klass)
- Fait aussi un cast !

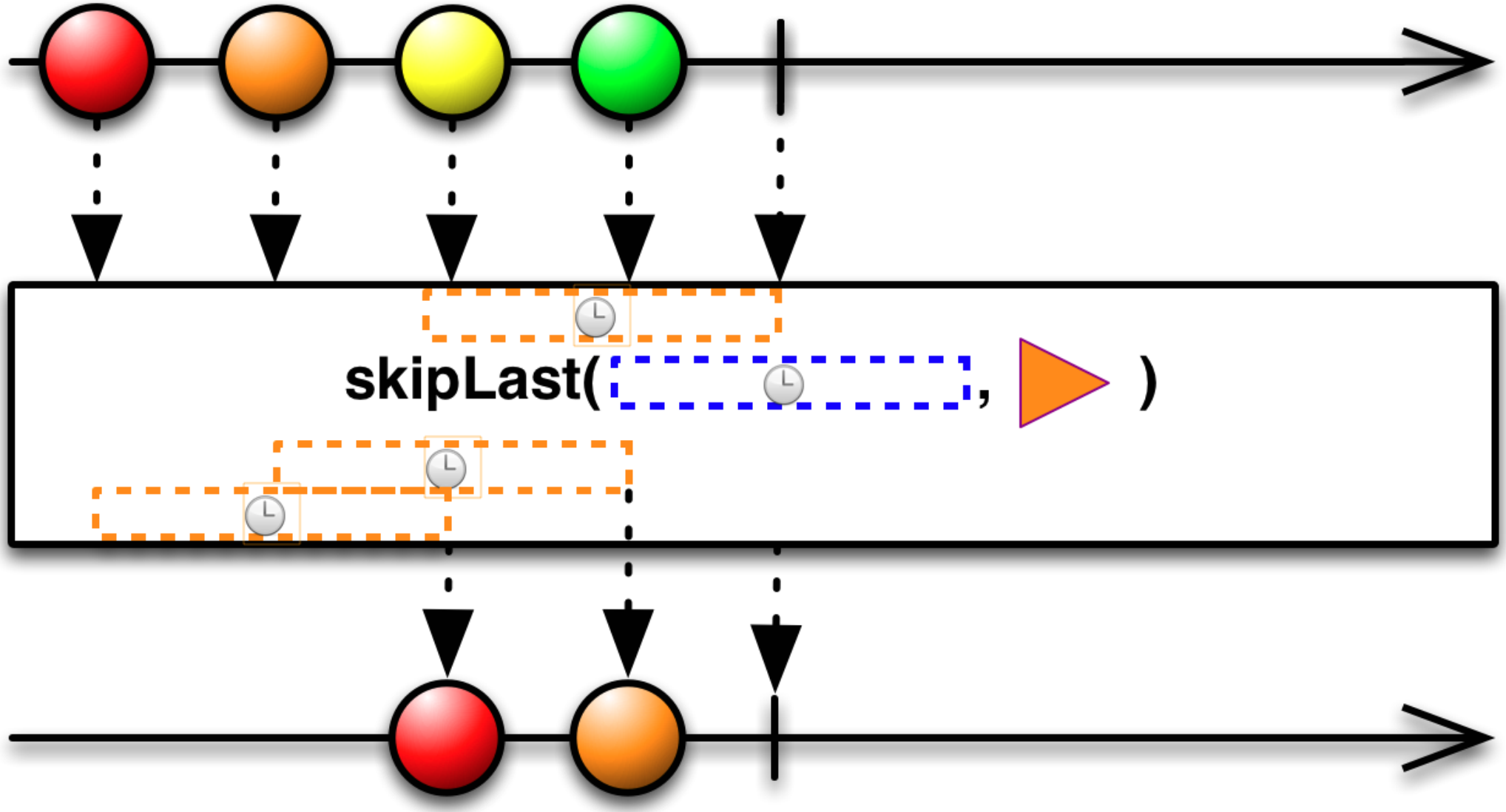

skip(3)

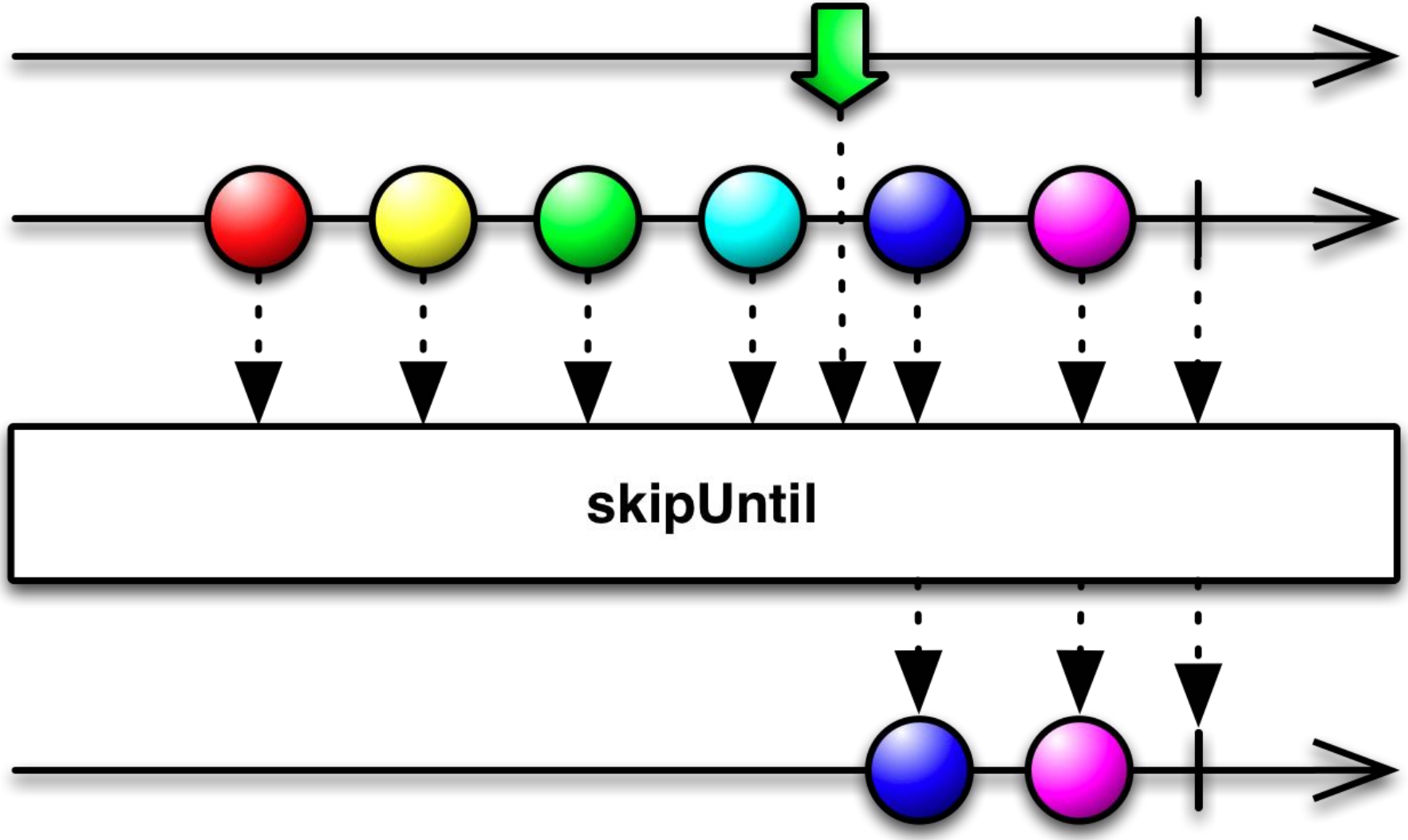

skip( )

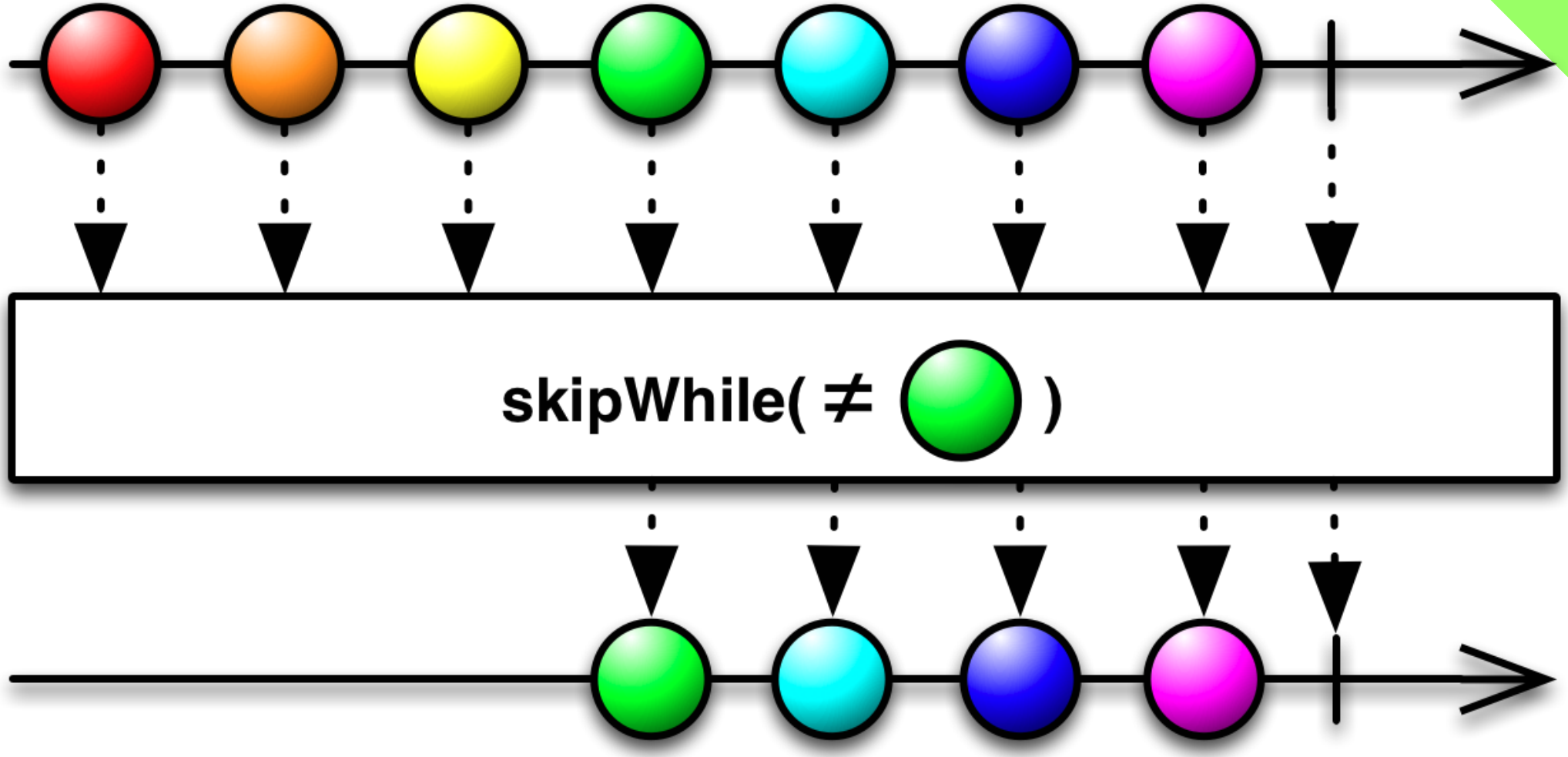

skip(,)

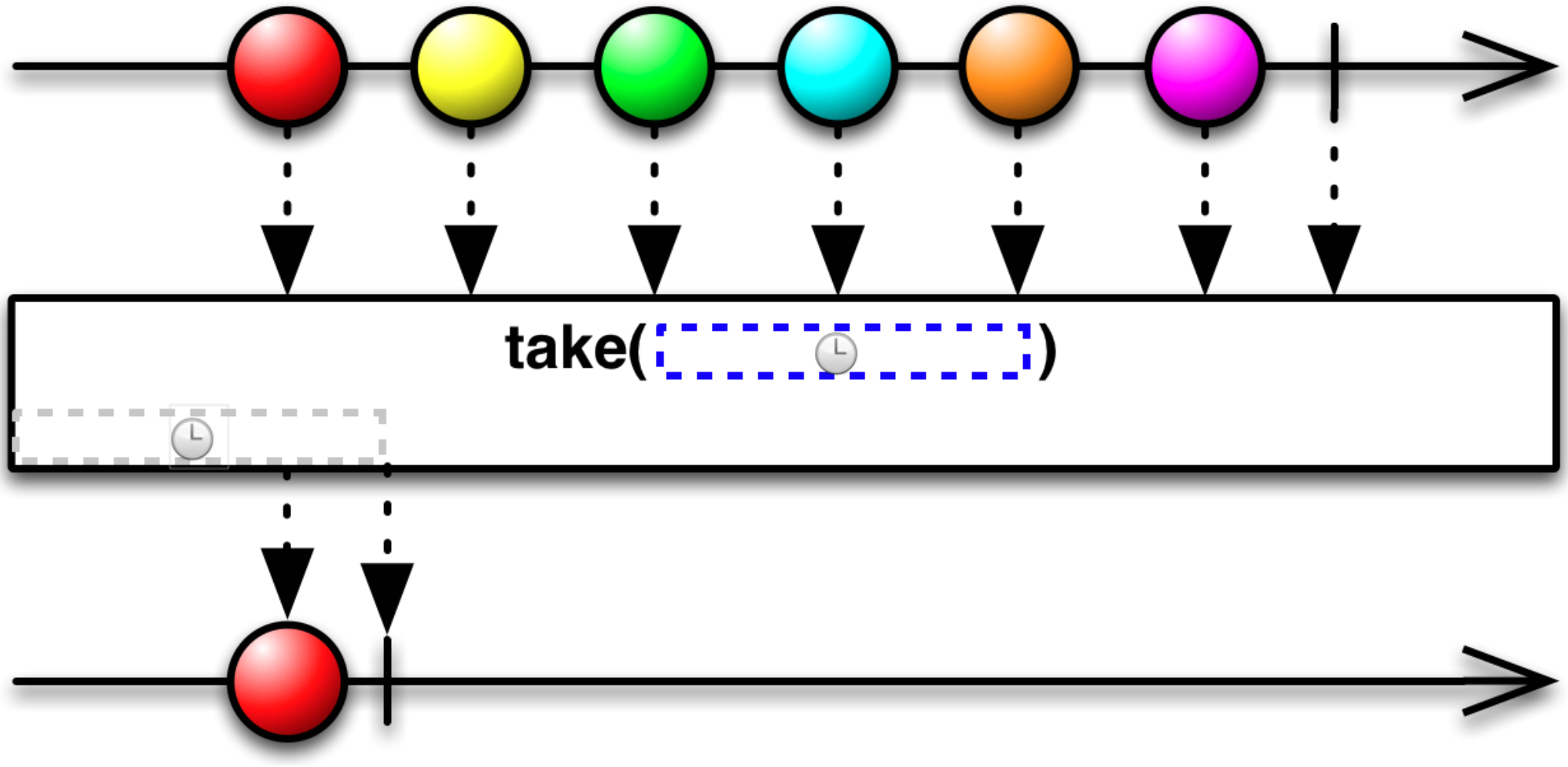

skipLast(3)

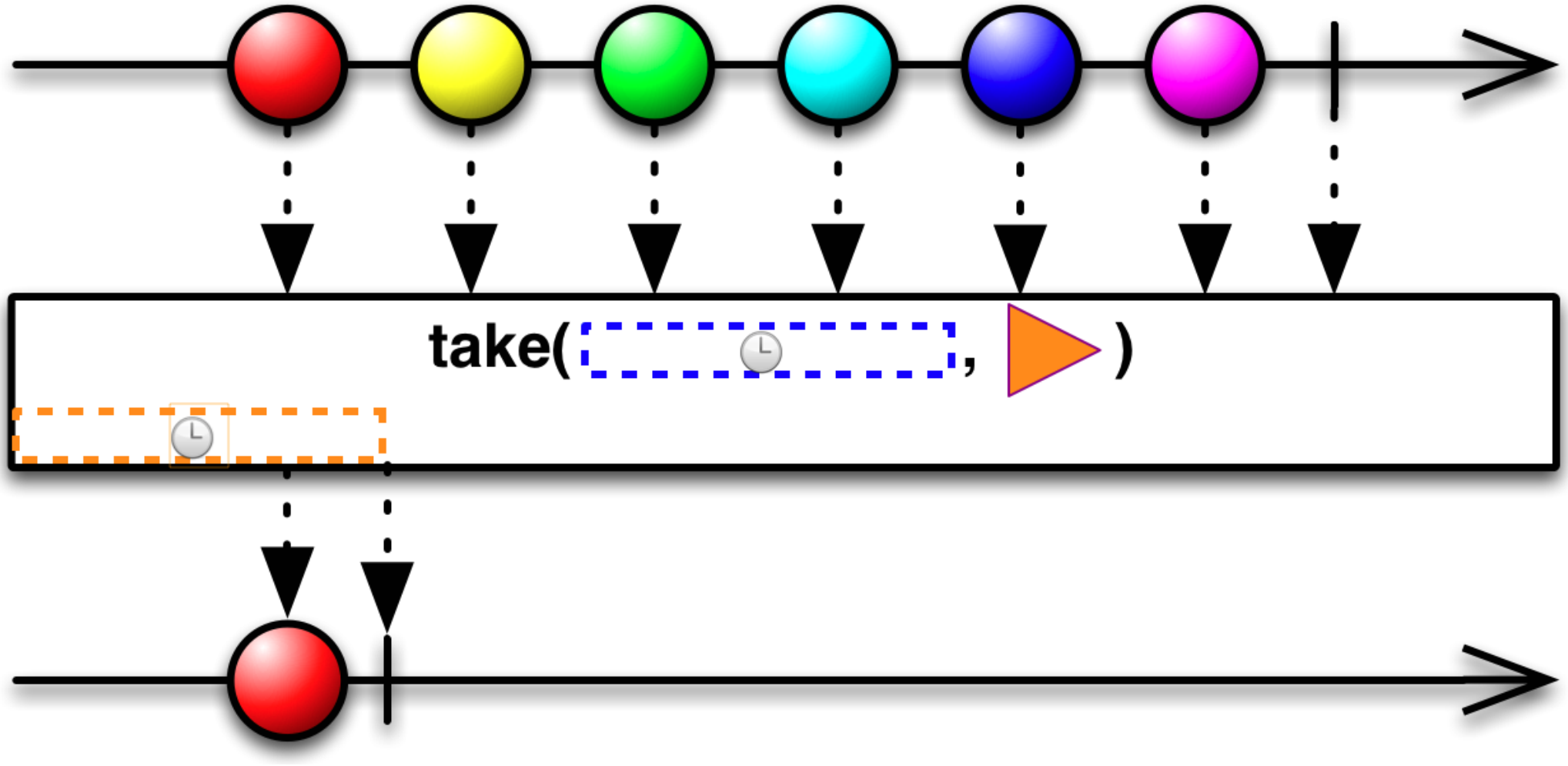

skipLast( )

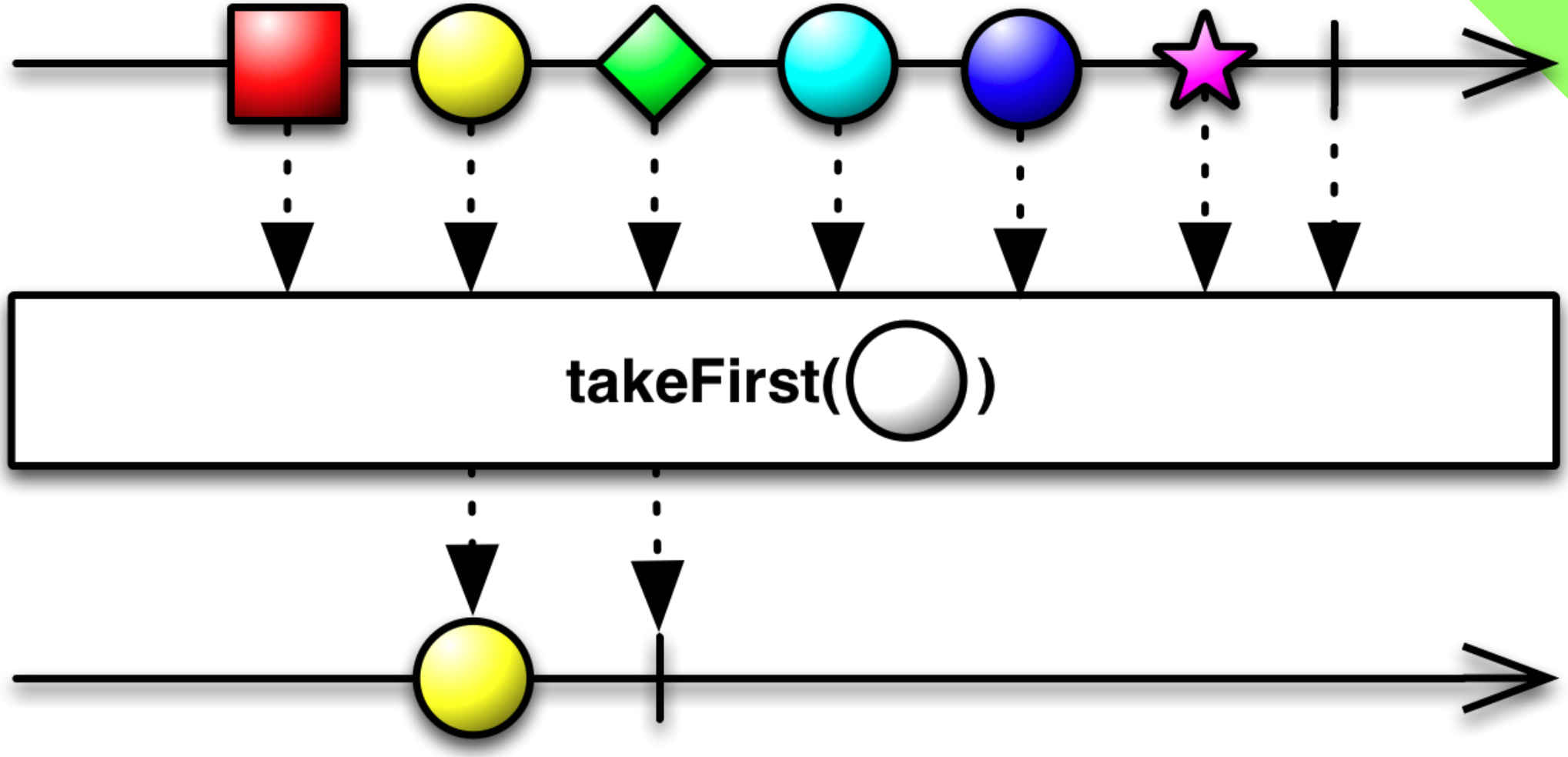

skipLast(,)

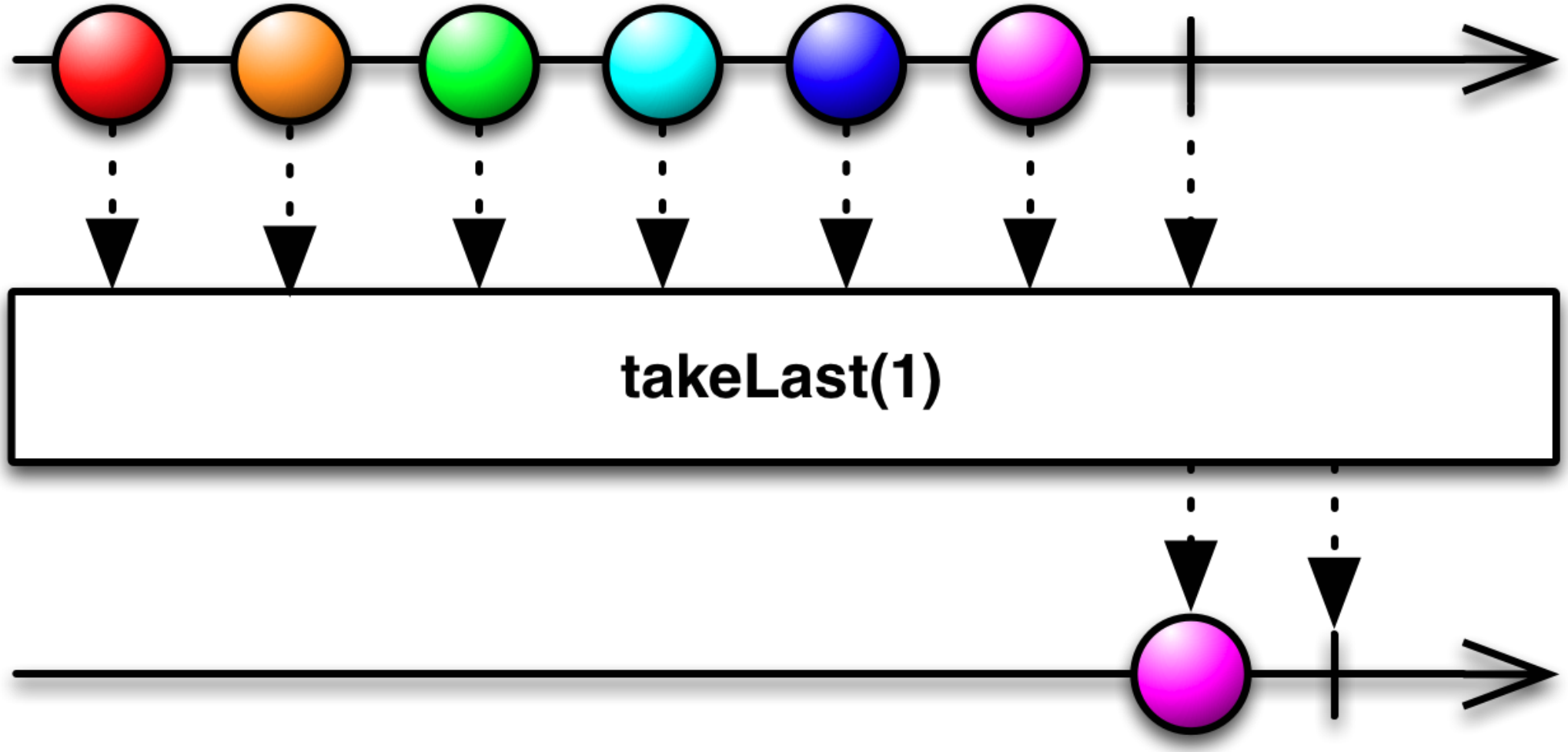

skipUntil

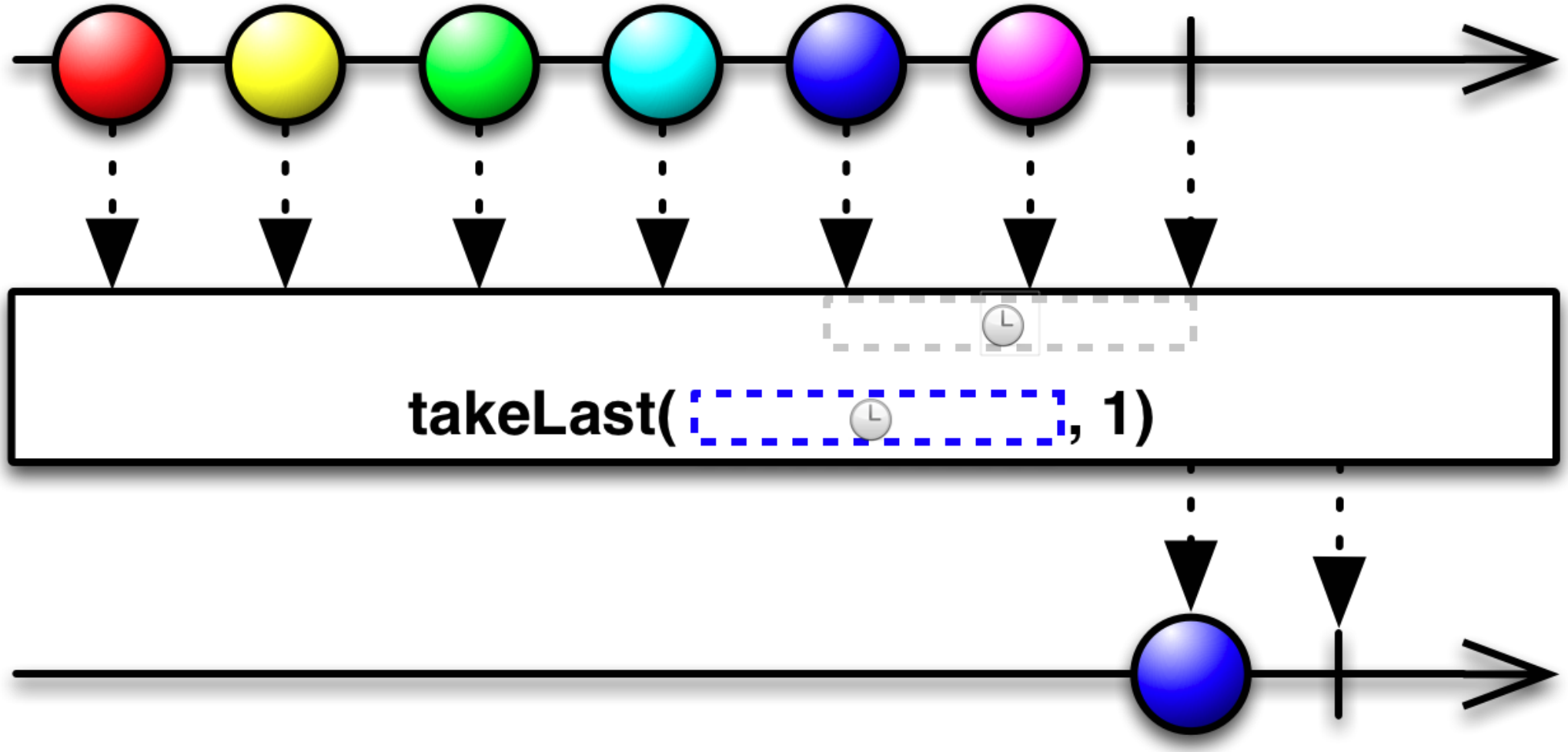

skipWhile(≠)

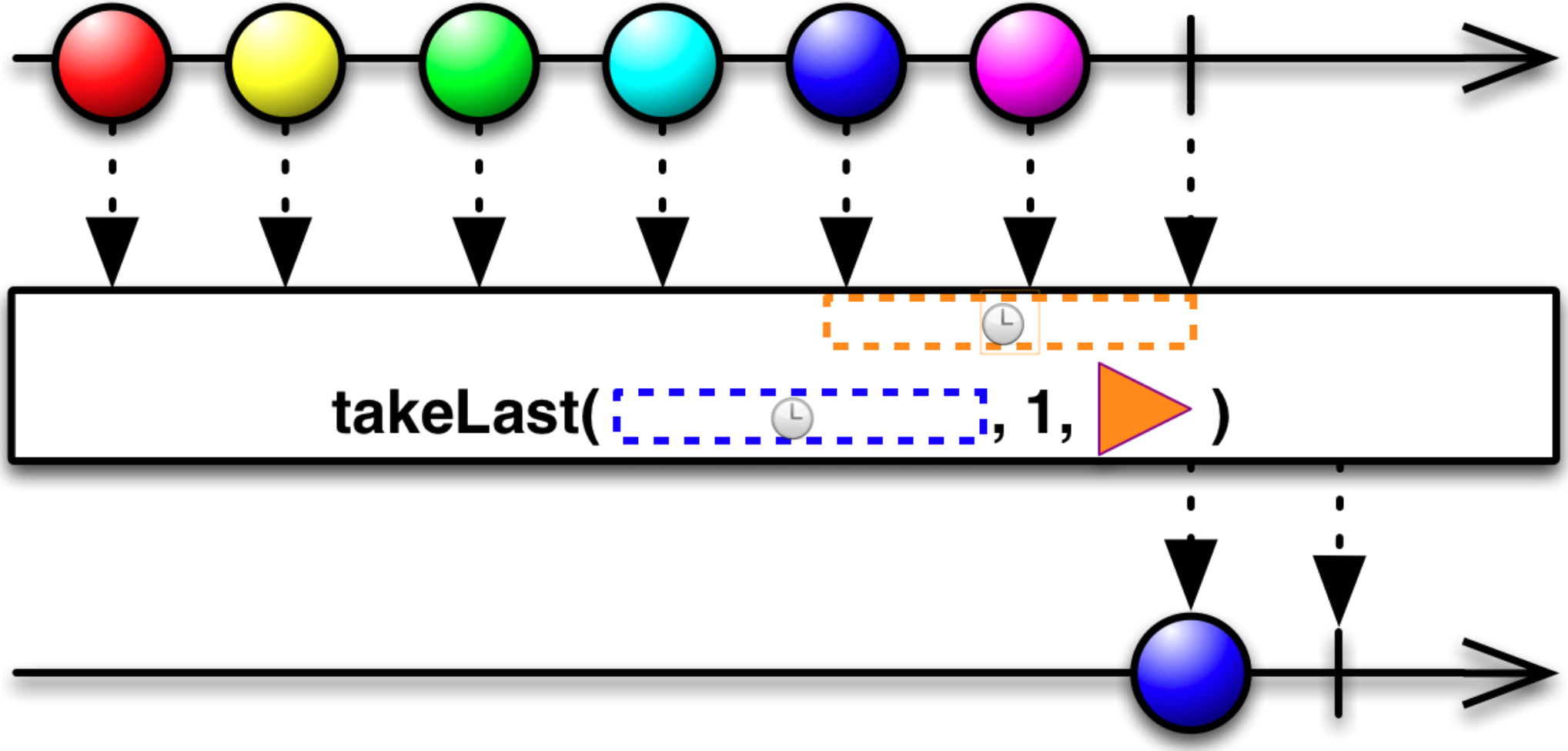

take( )

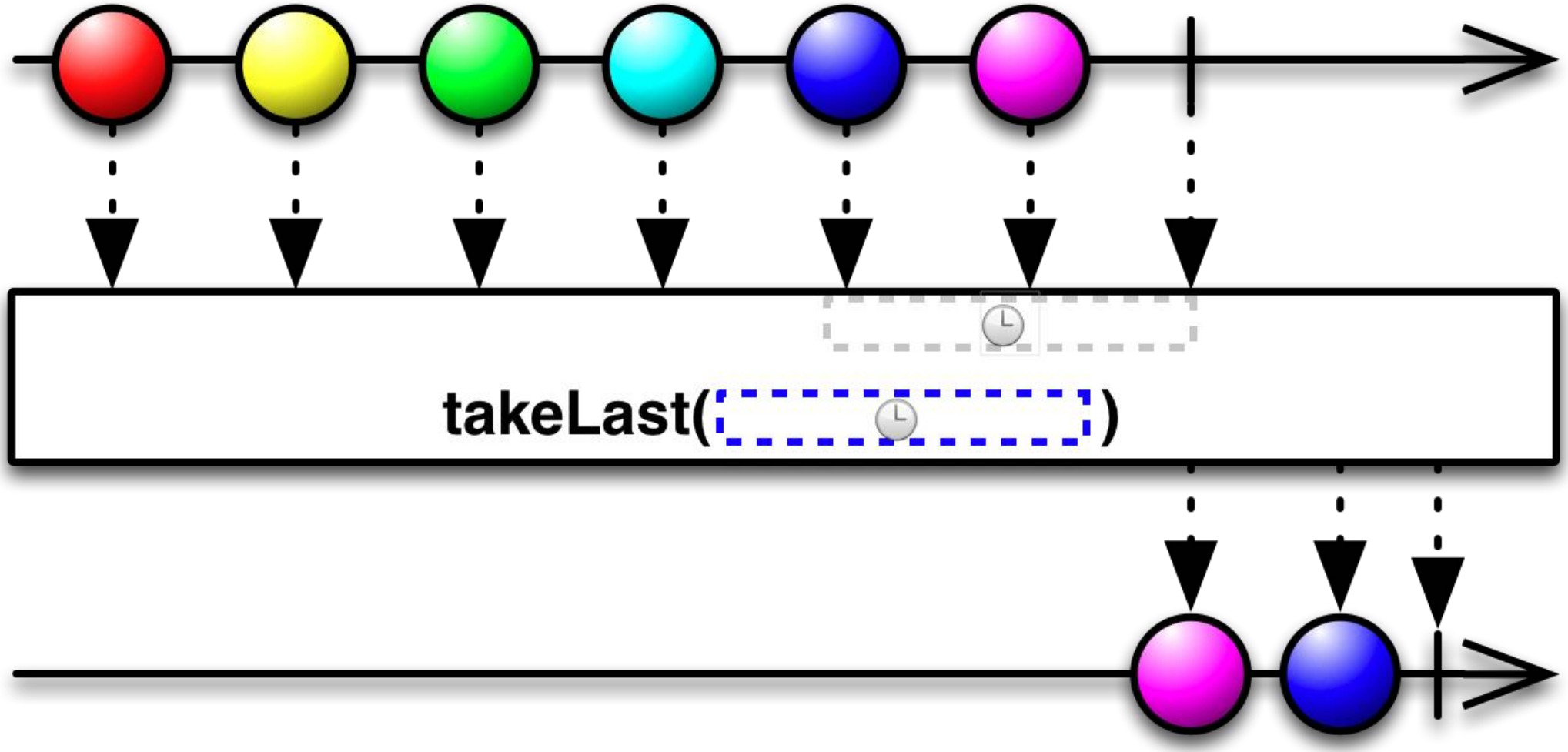

take(,)

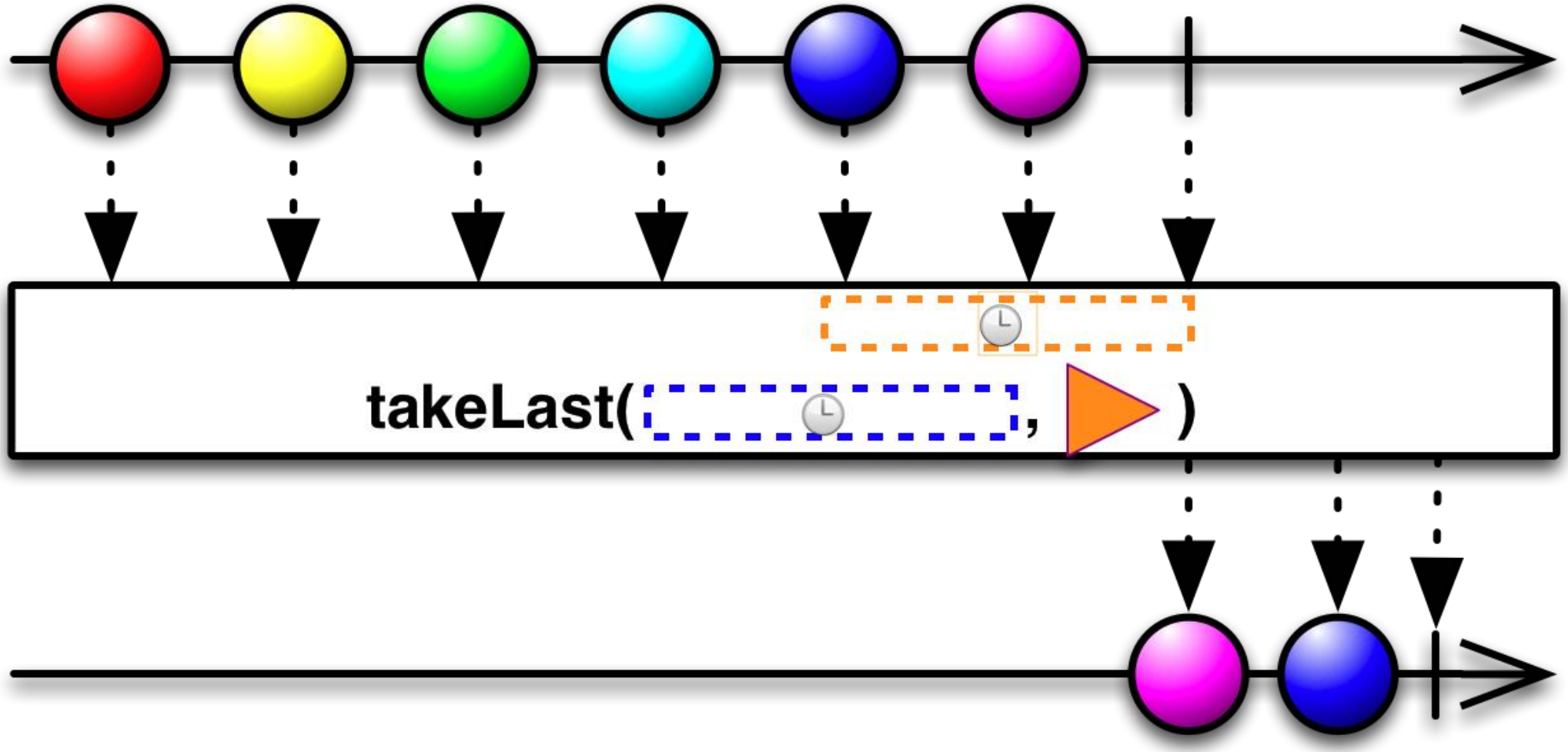

takeFirst( )

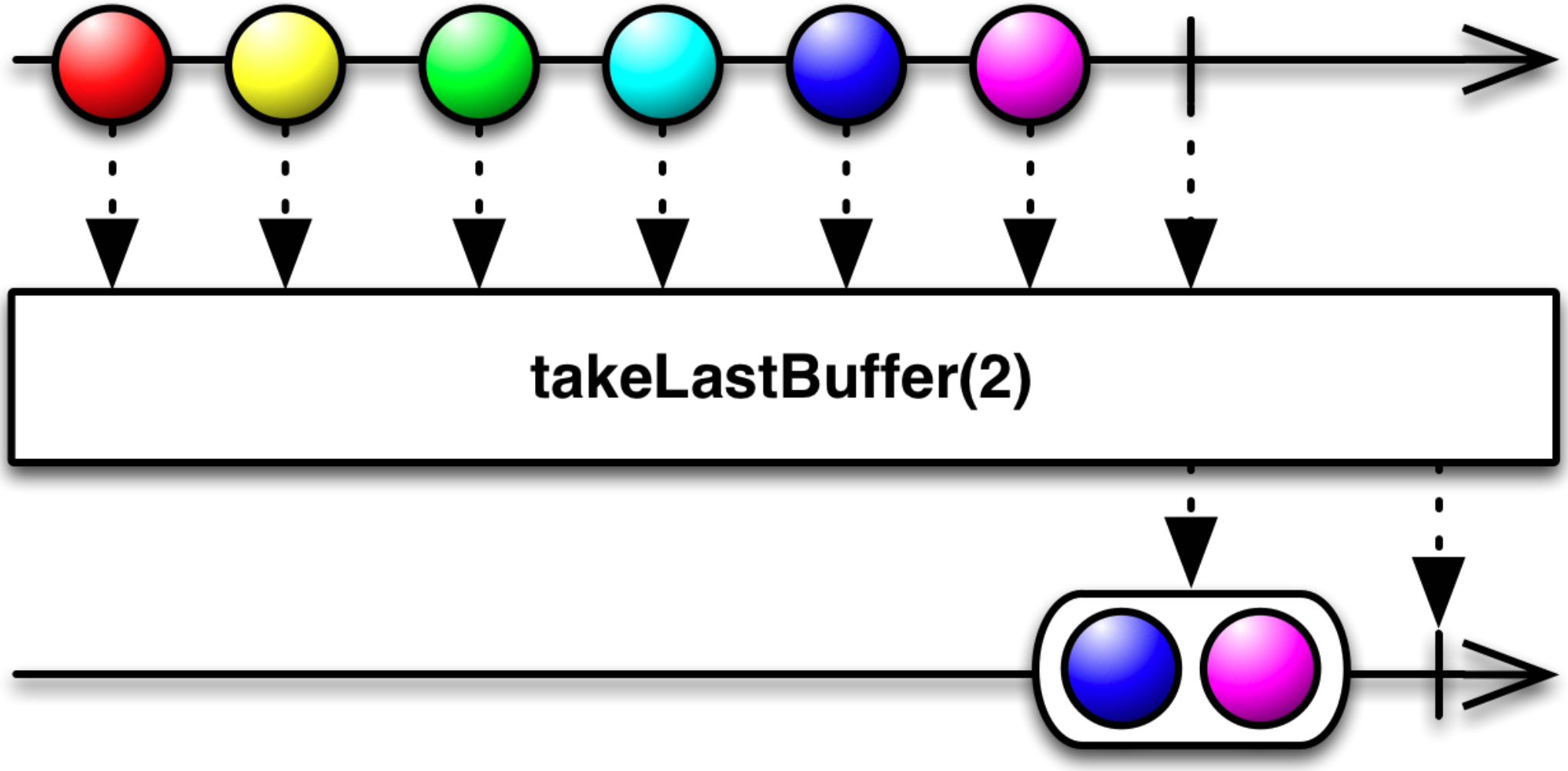

takeLast(1)

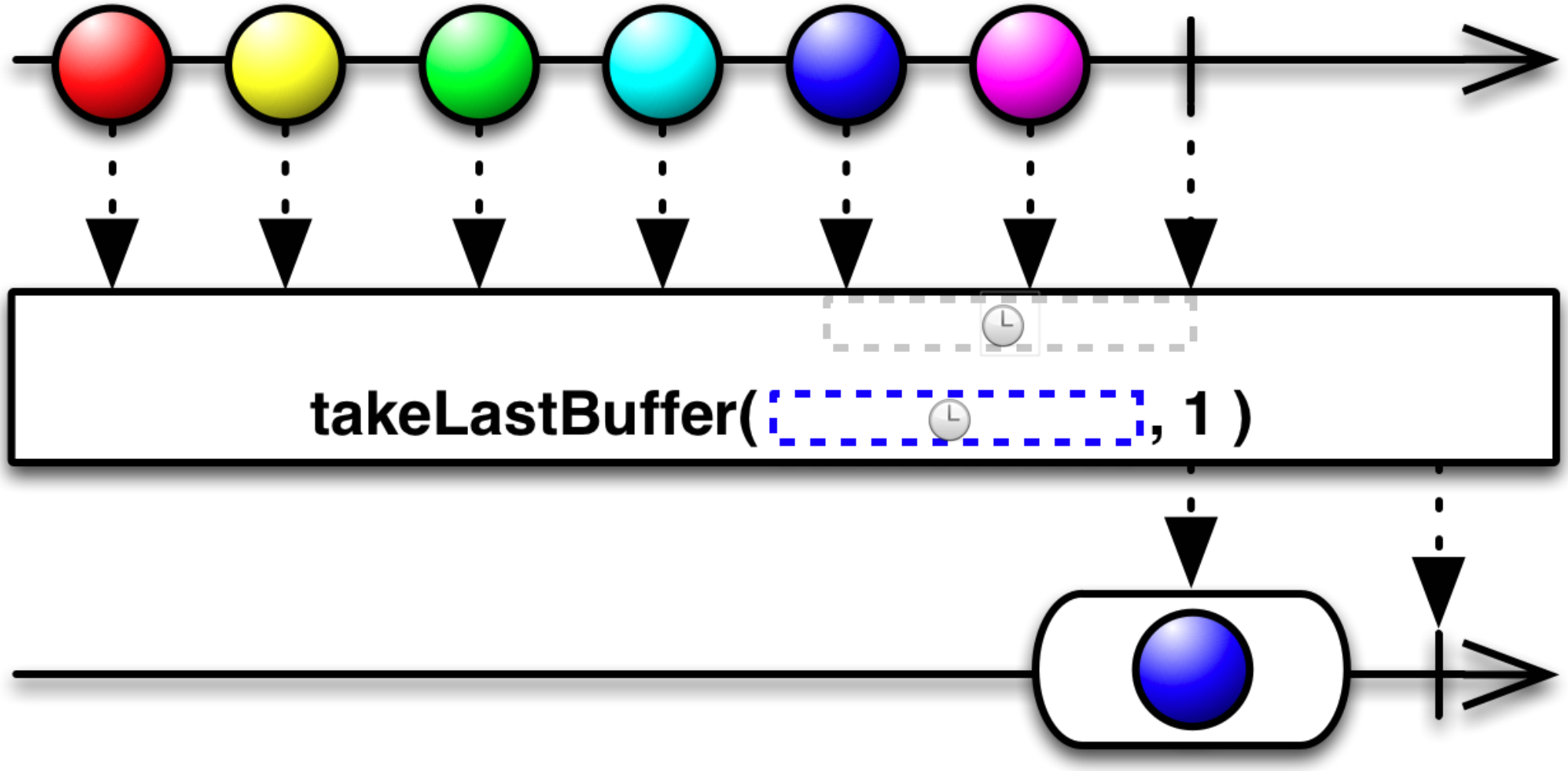

takeLast(, 1)

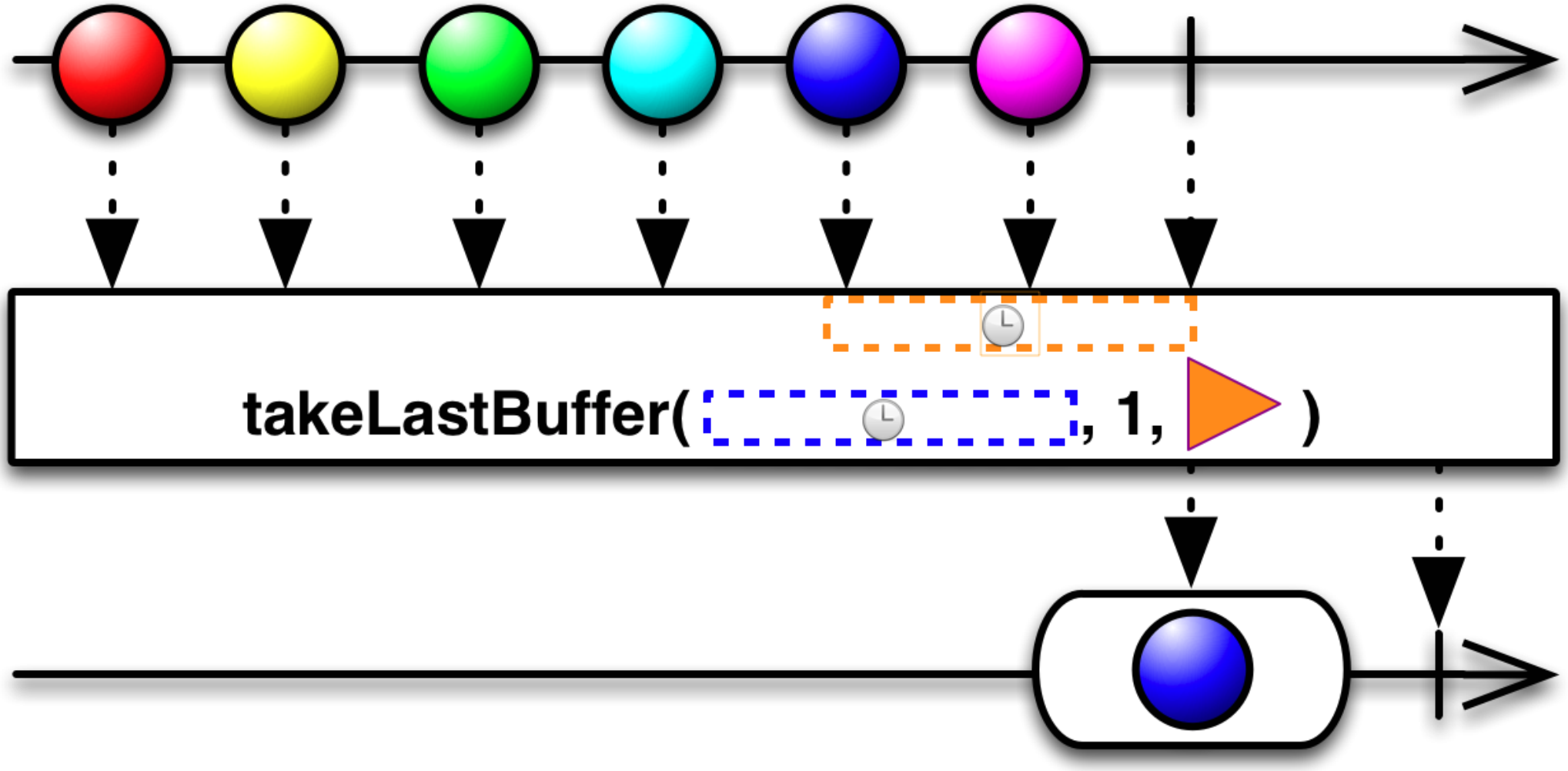

takeLast(, 1,)

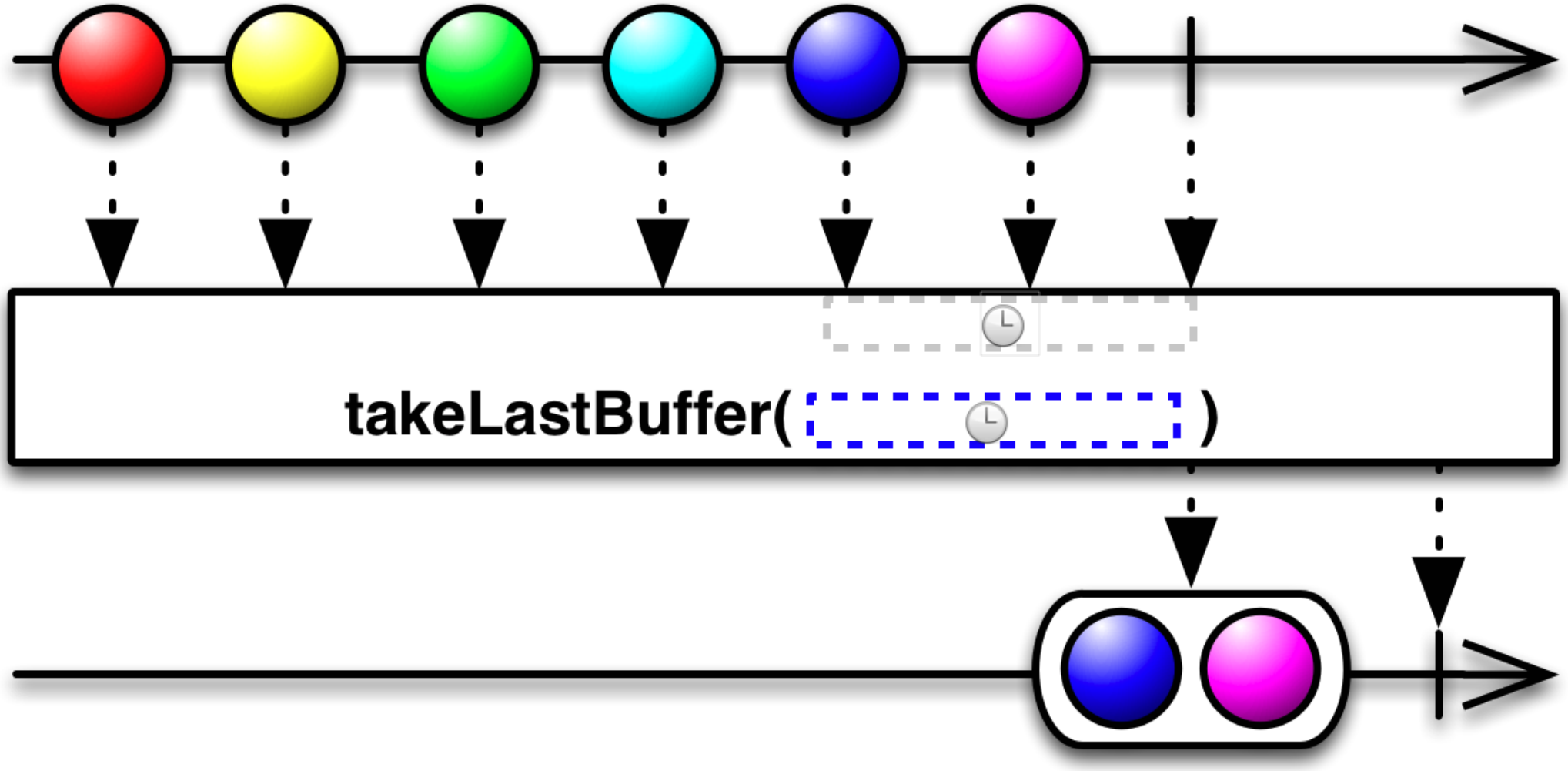

takeLast( )

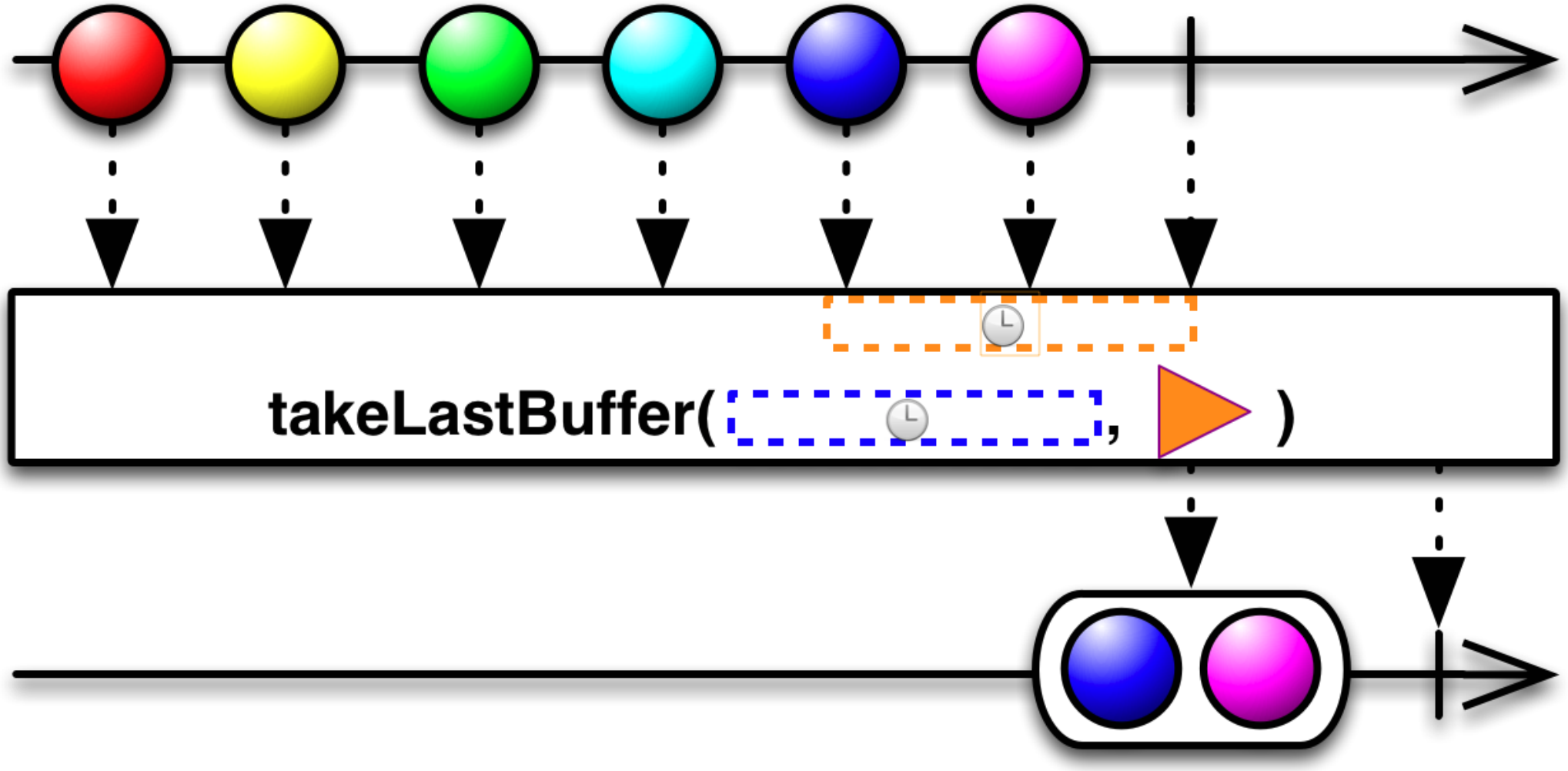

takeLast(,)

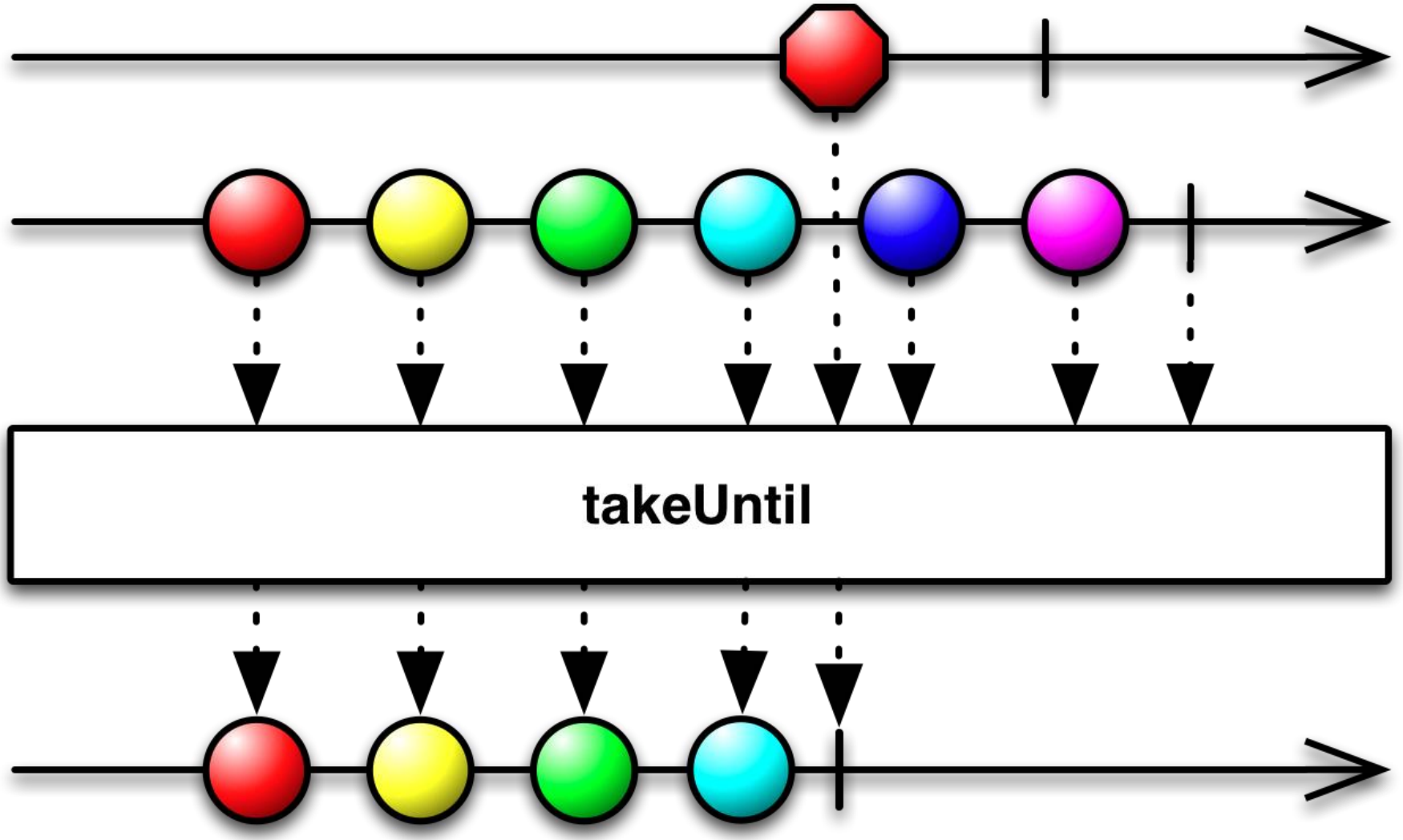

takeLastBuffer(2)


takeLastBuffer(, 1)

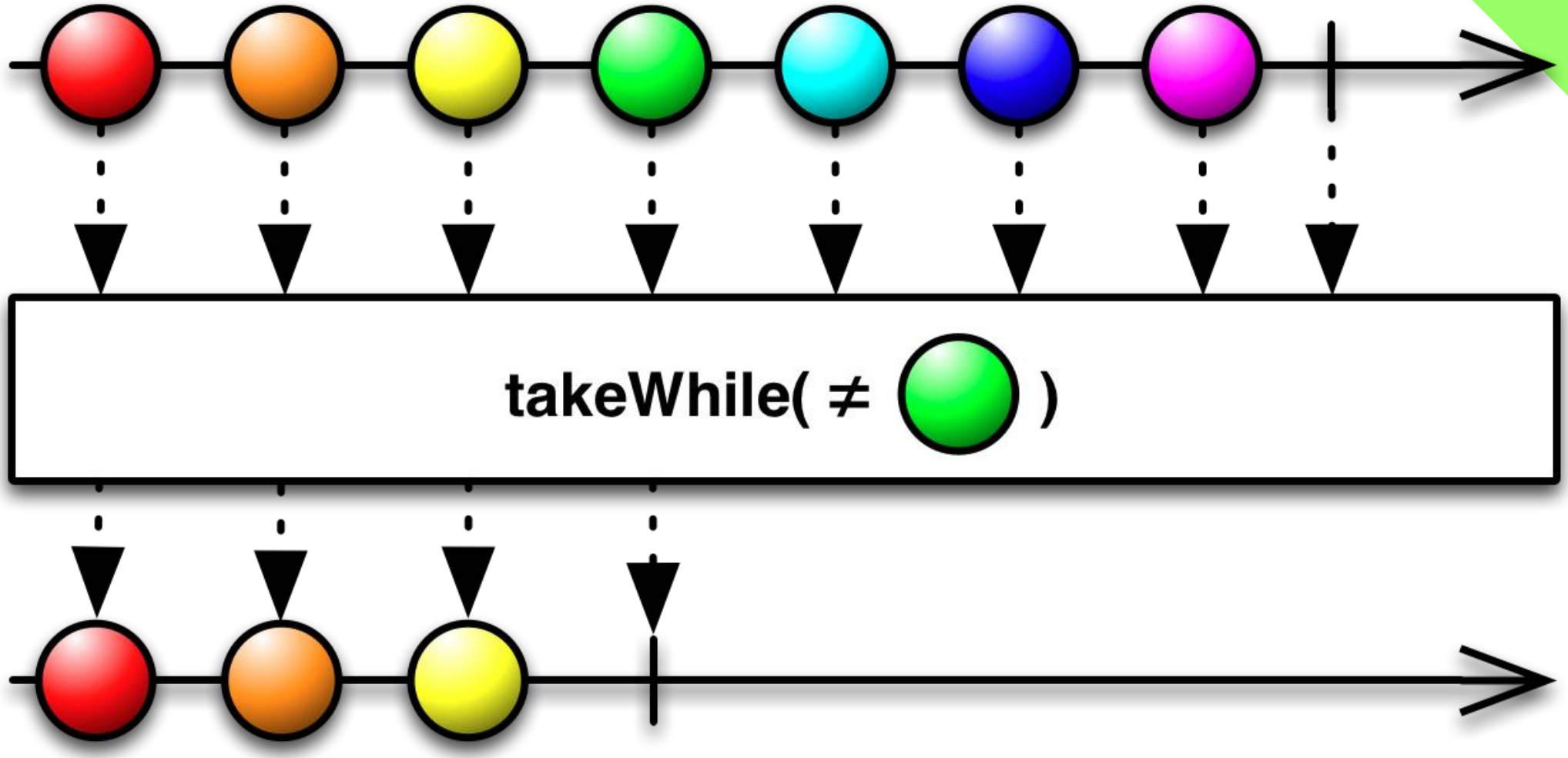

takeLastBuffer(, 1,)


takeLastBuffer( )

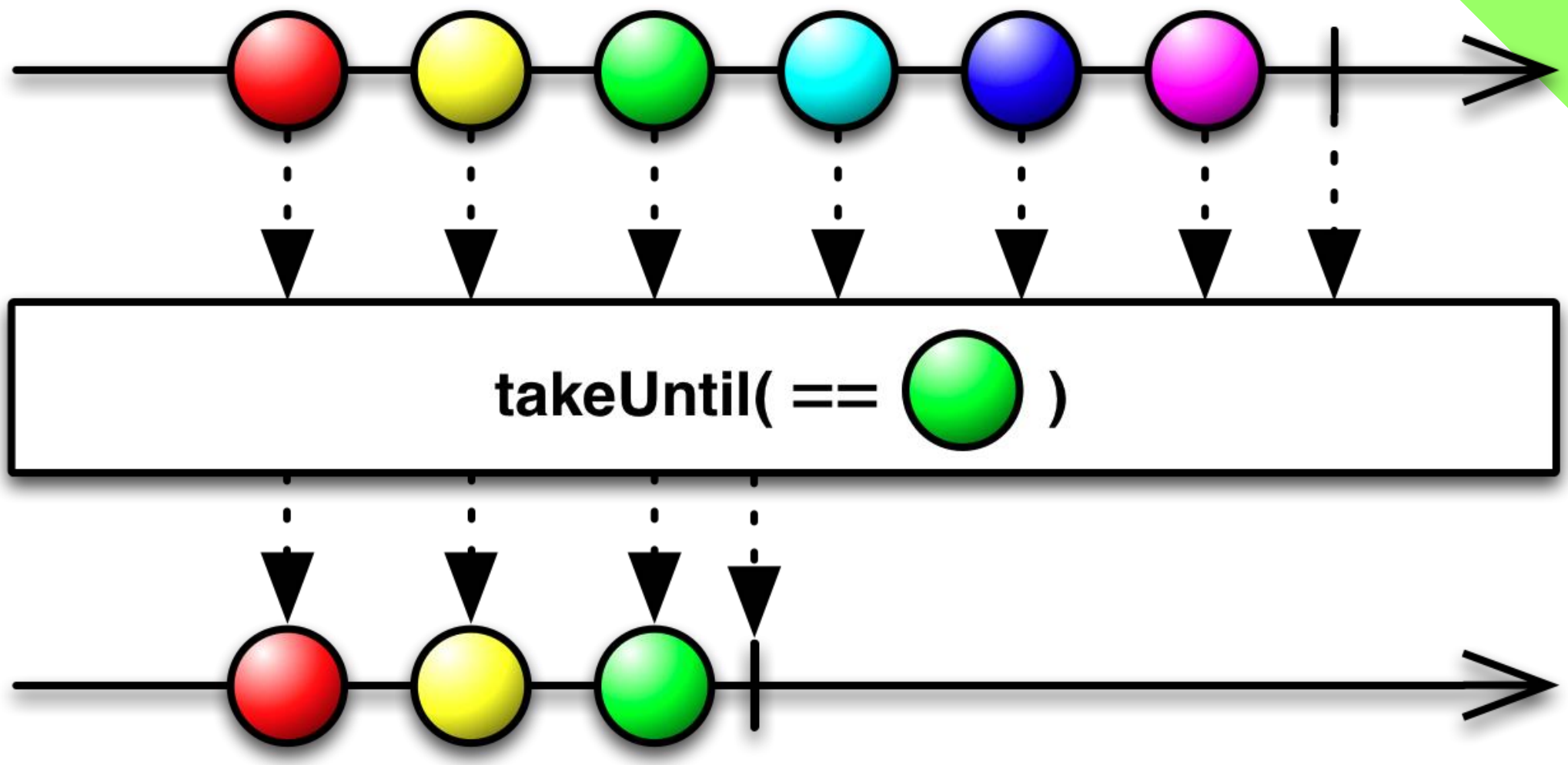

takeLastBuffer(,)

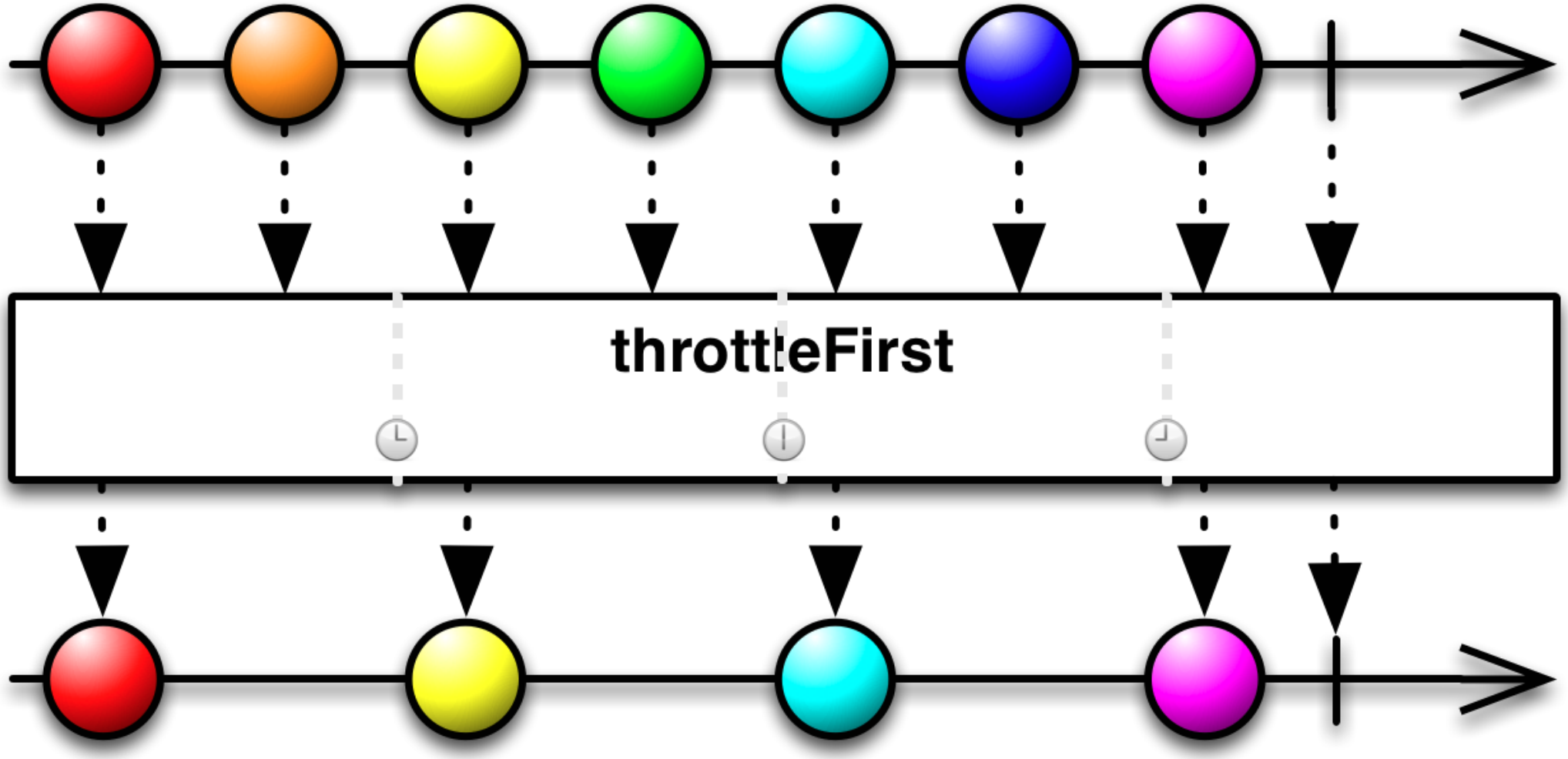


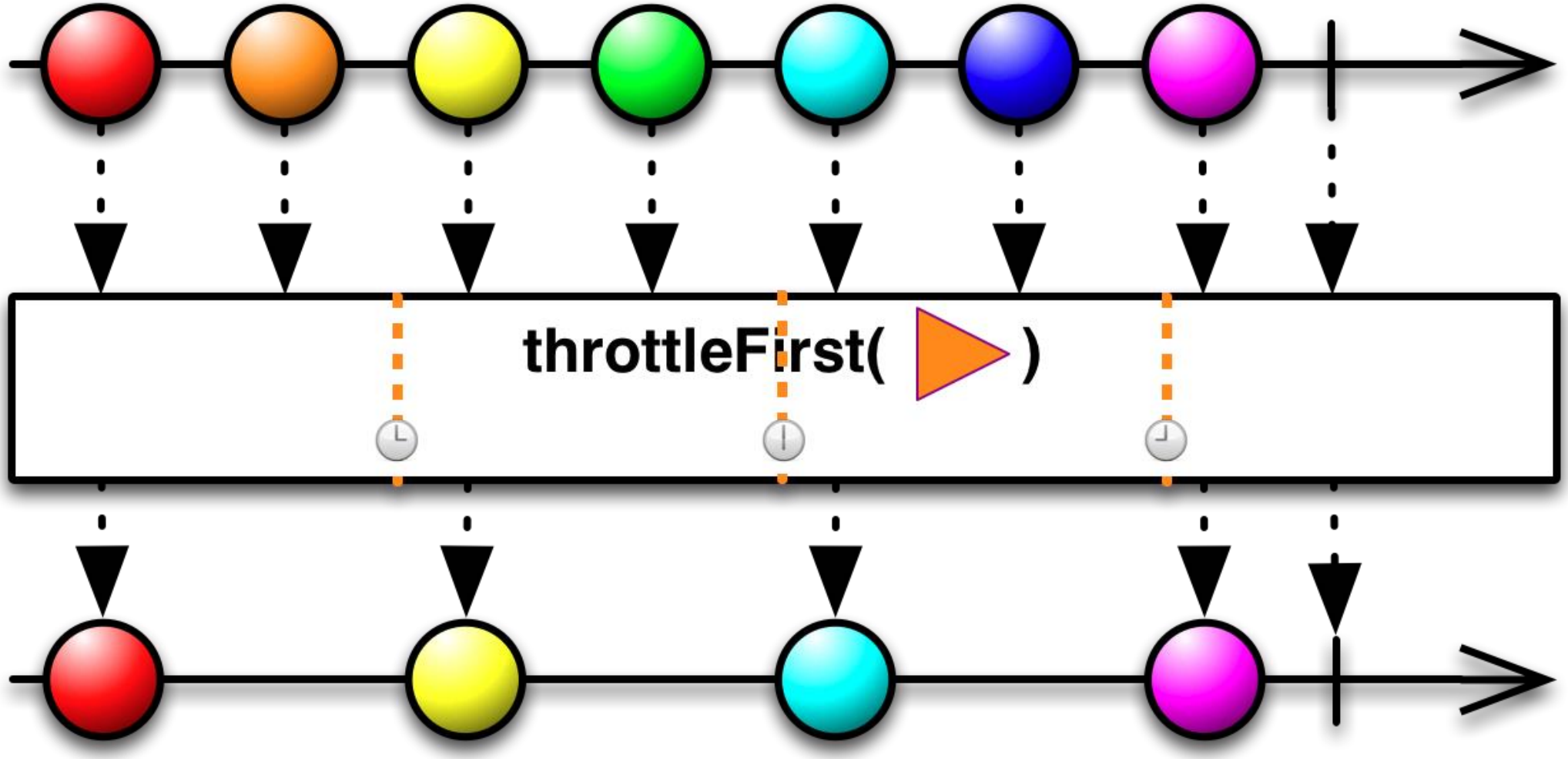
- Cas d'usage : Permet d’interrompre le flux.

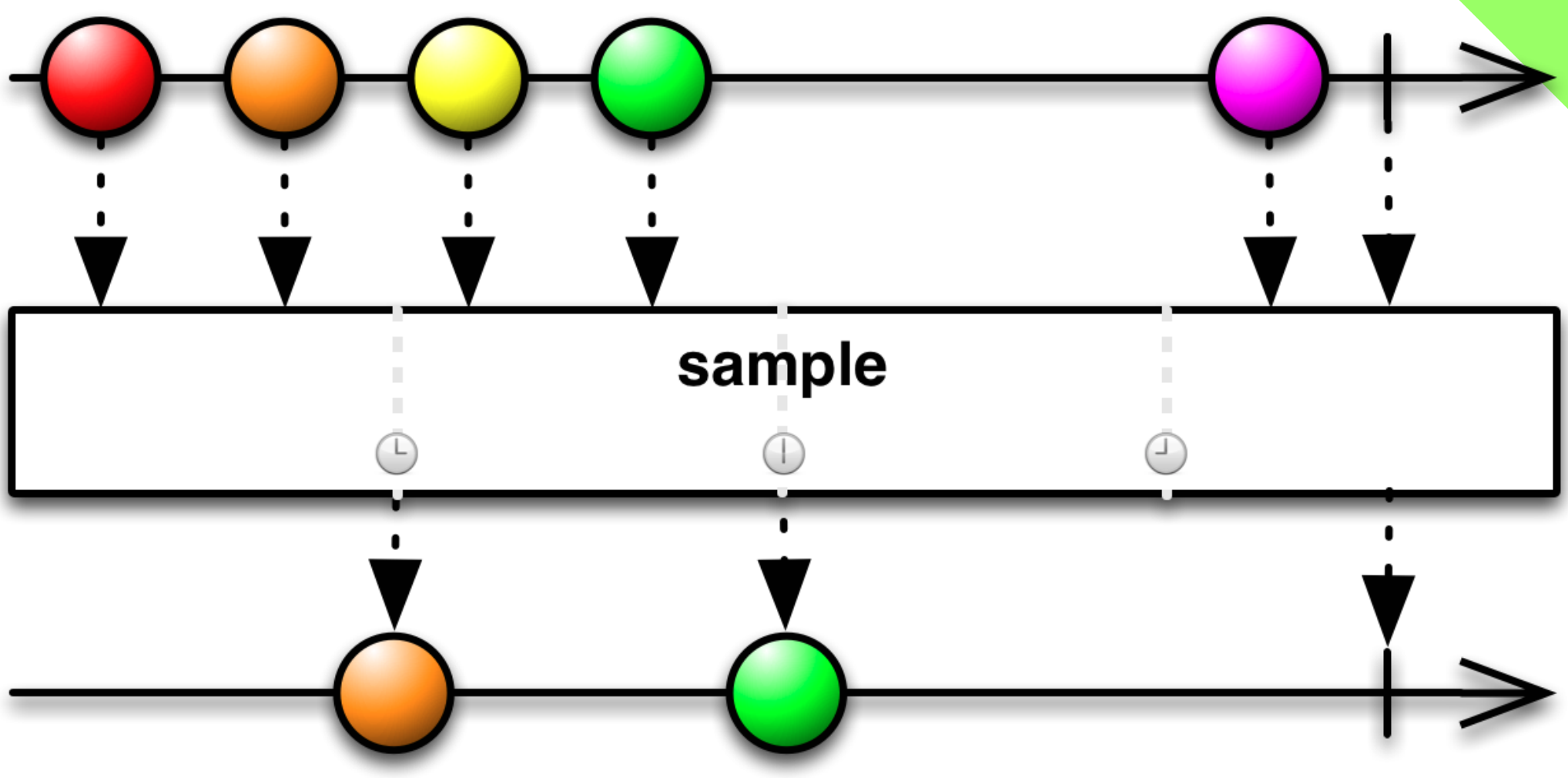


- Cas d'usage : Permet d’interrompre le flux.


takeUntil(==)

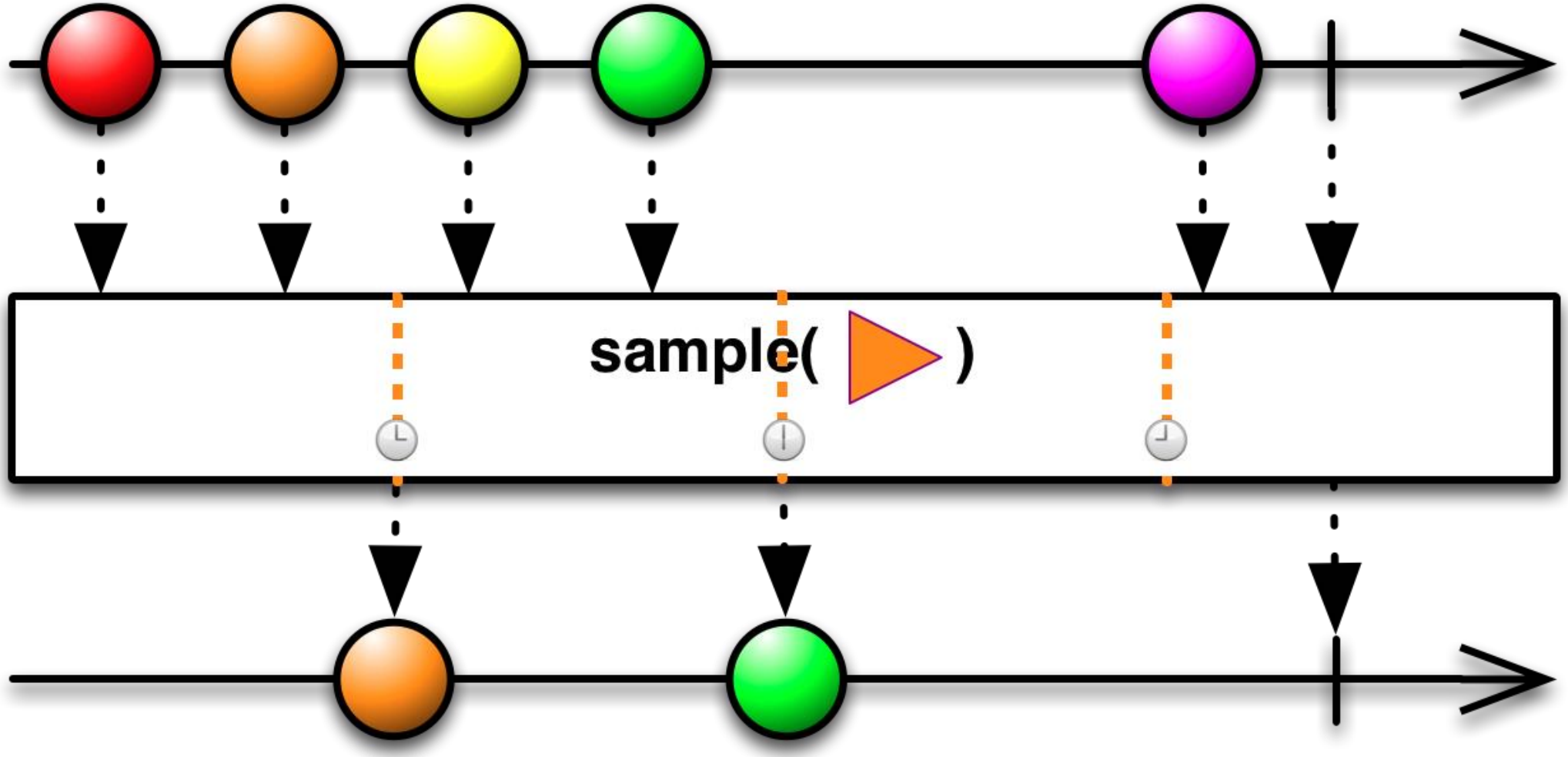

throttleFirst

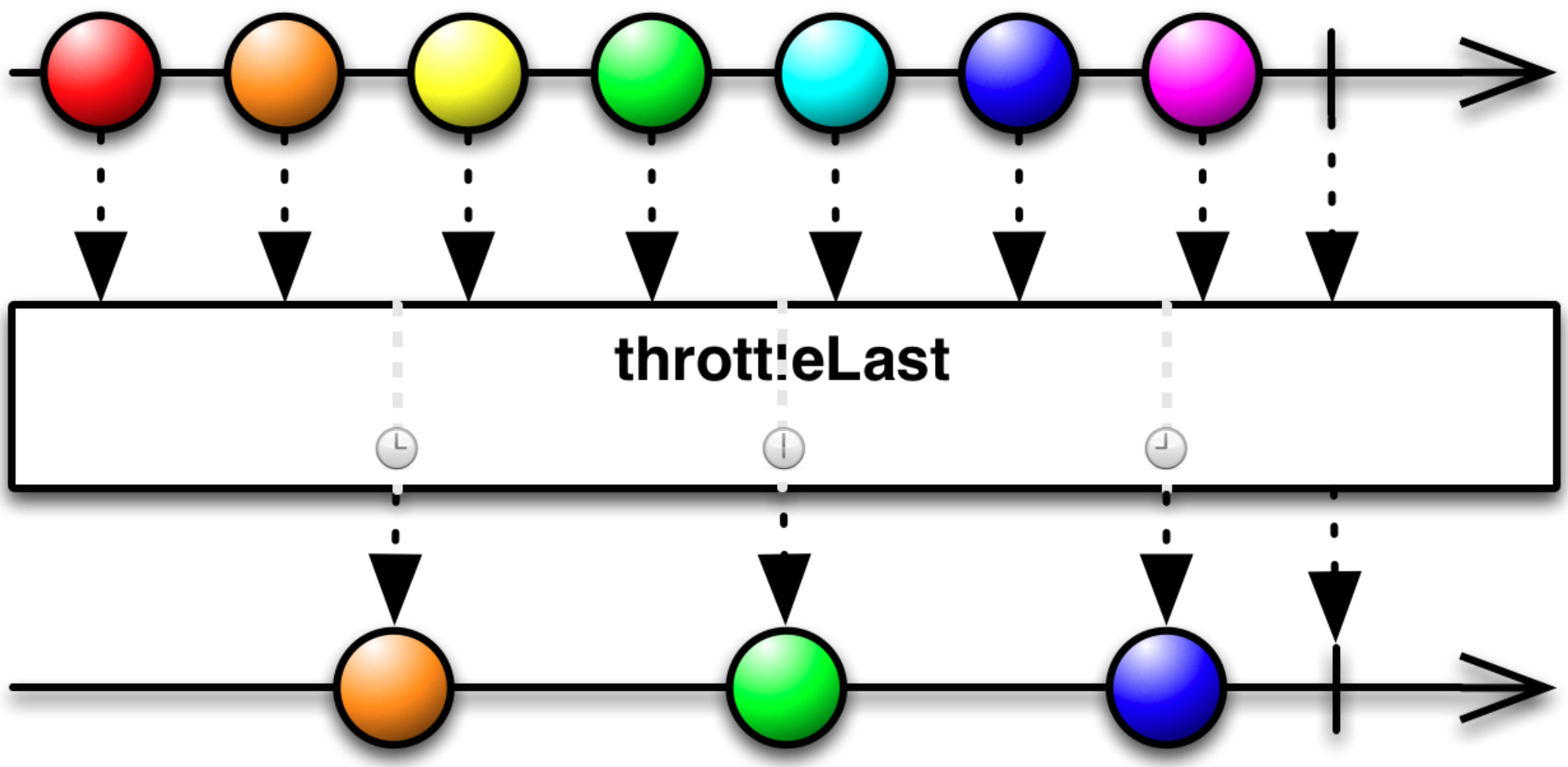

throttleFirst( )



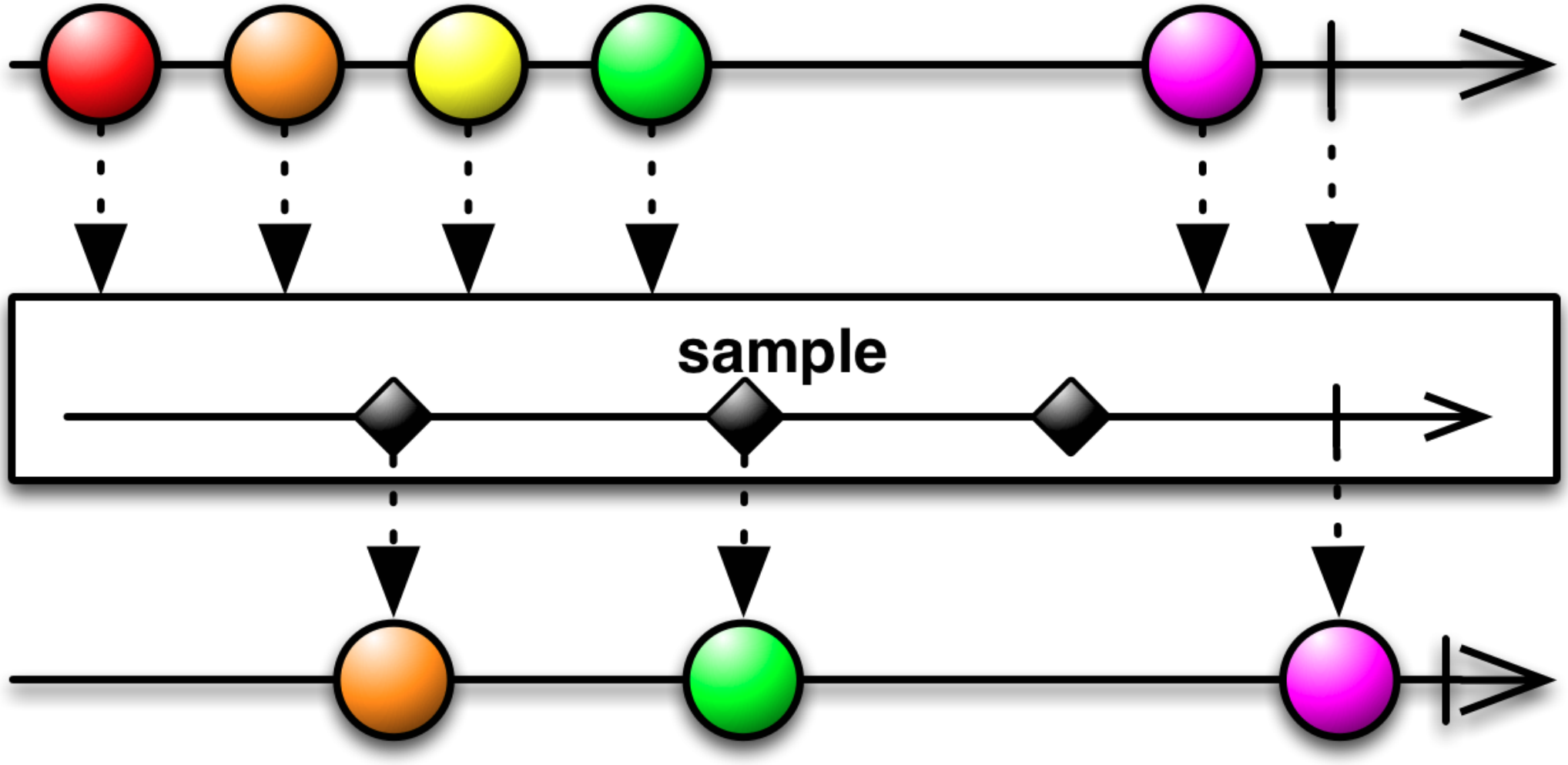
..sample(...) émets la dernière valeur de la période si elle existe.

.Cas d'usage : Peut être une réponse à un problème de Backpressure.

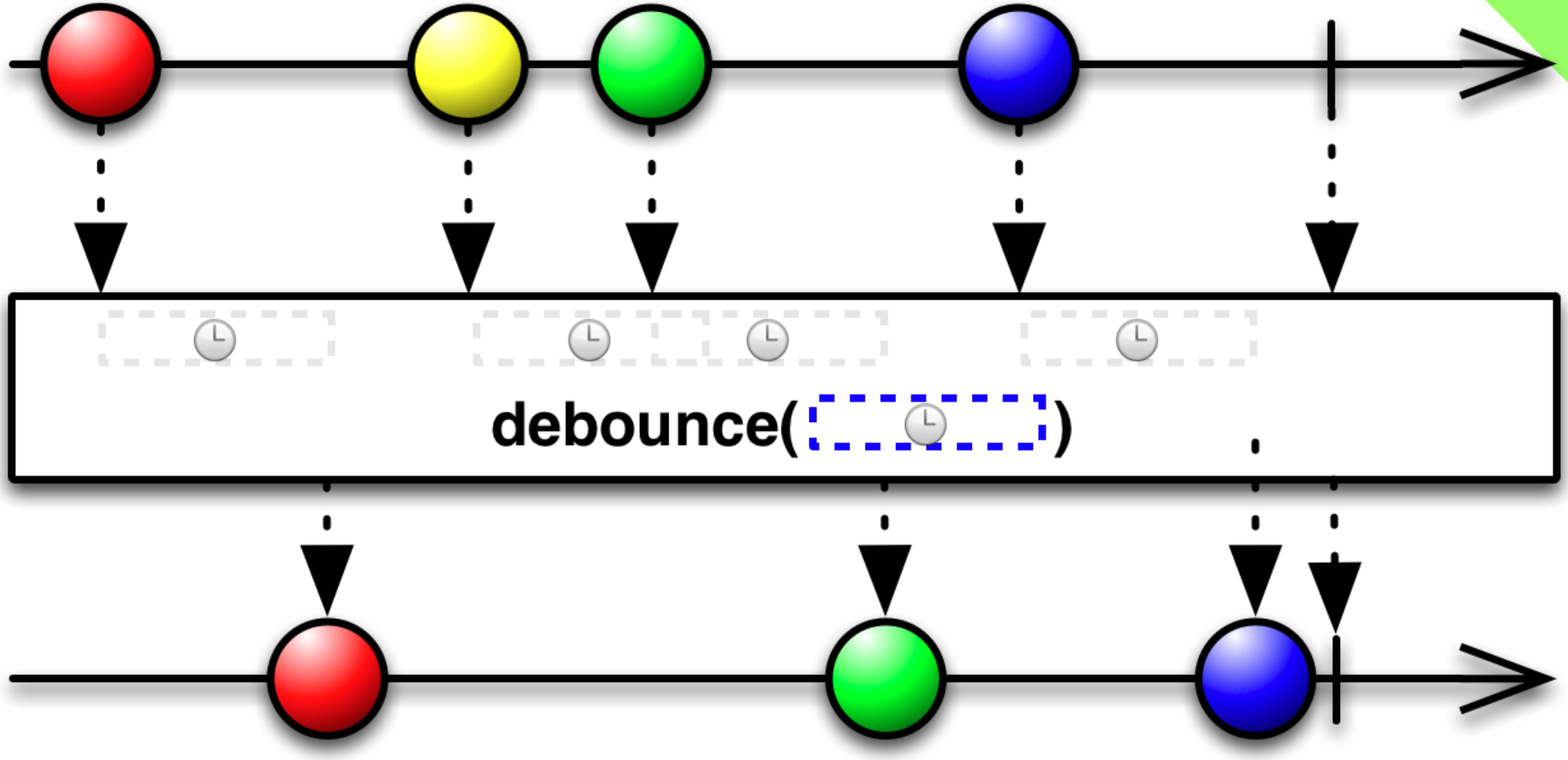

sample( )



..throttleLast(...) == .sample(...)

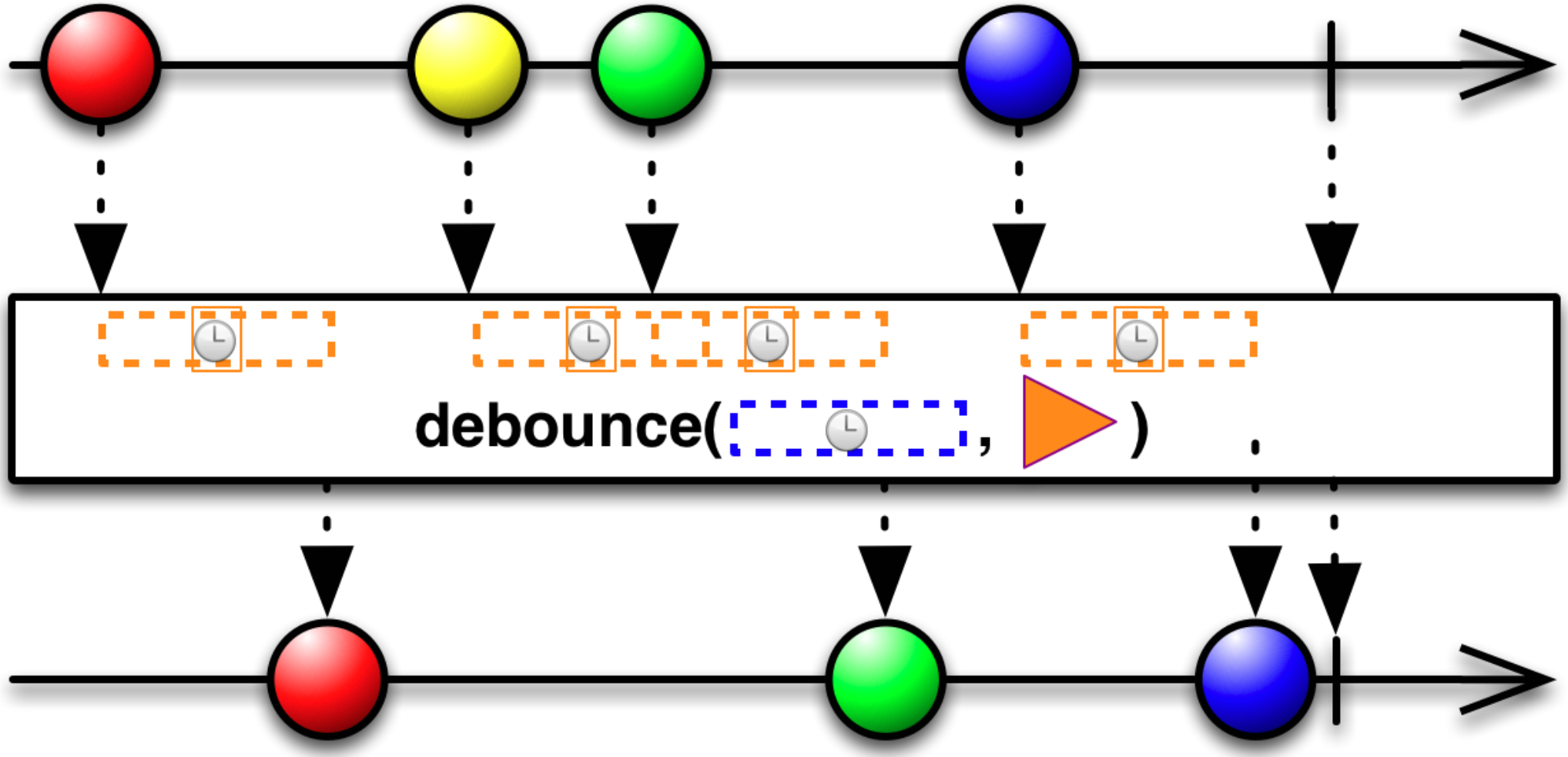


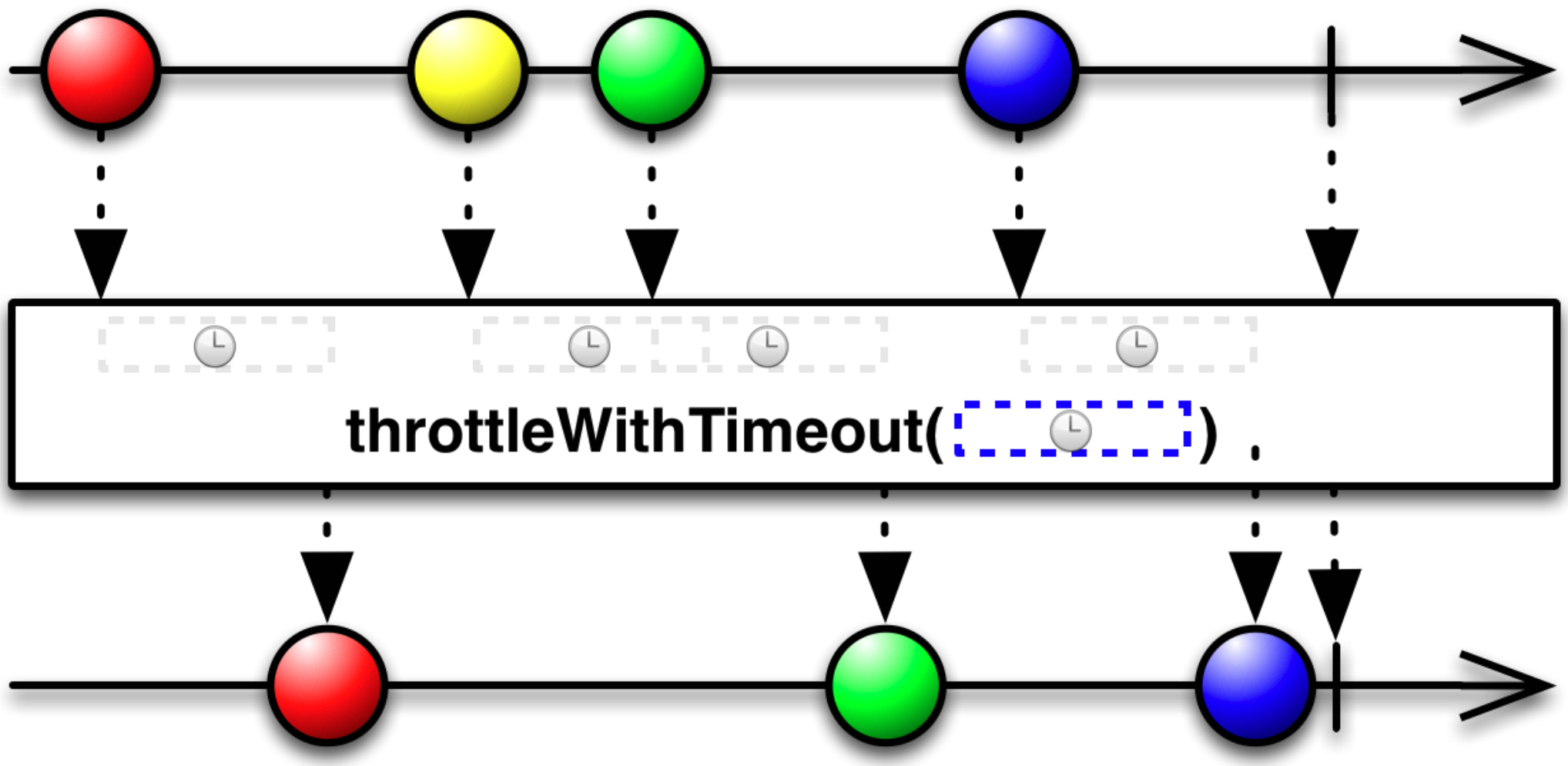
.sample(Observable<U> sampler)



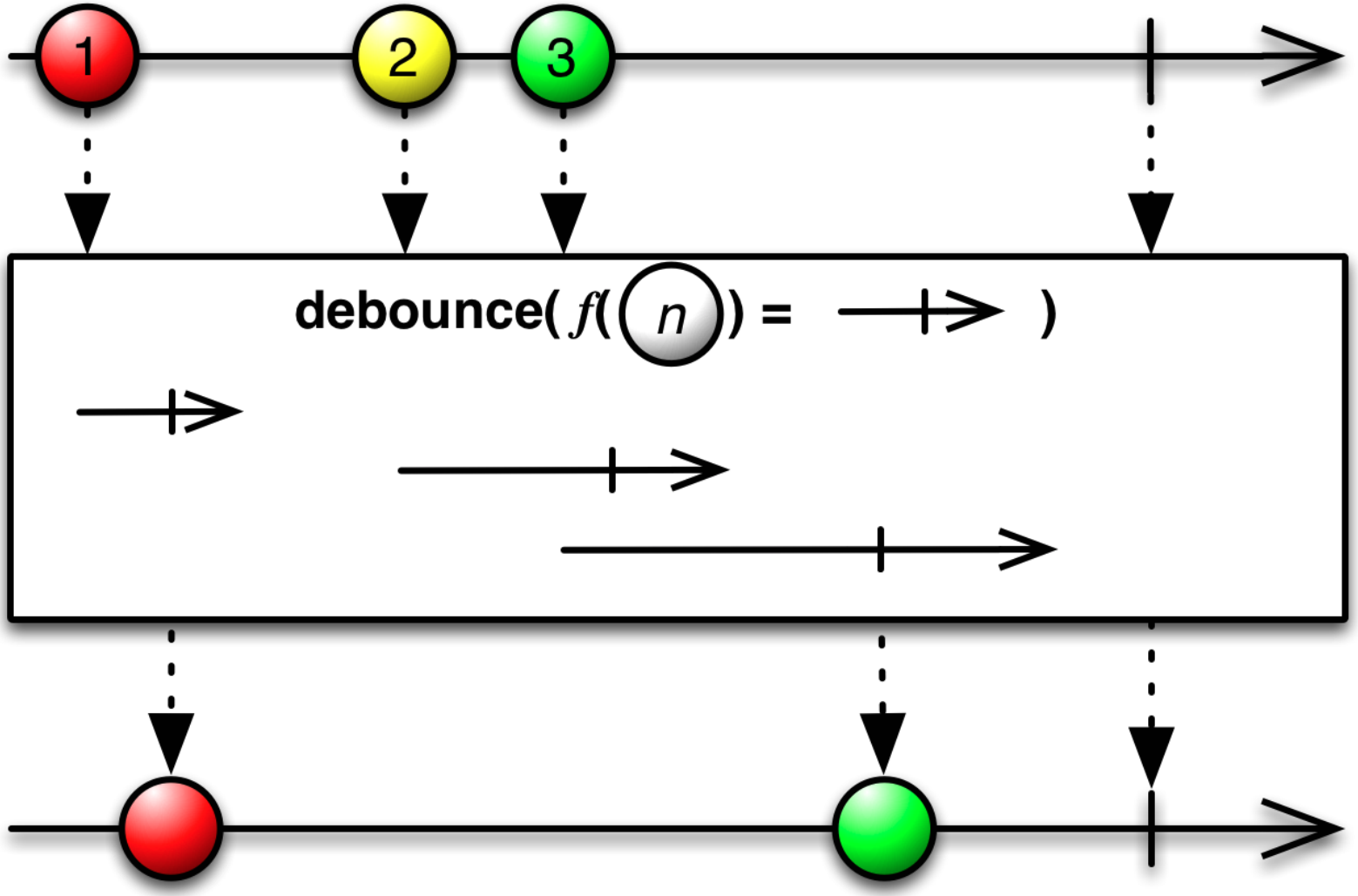
..debounce(...) attend un intervalle d'inactivité pour emmètre la dernière valeur.

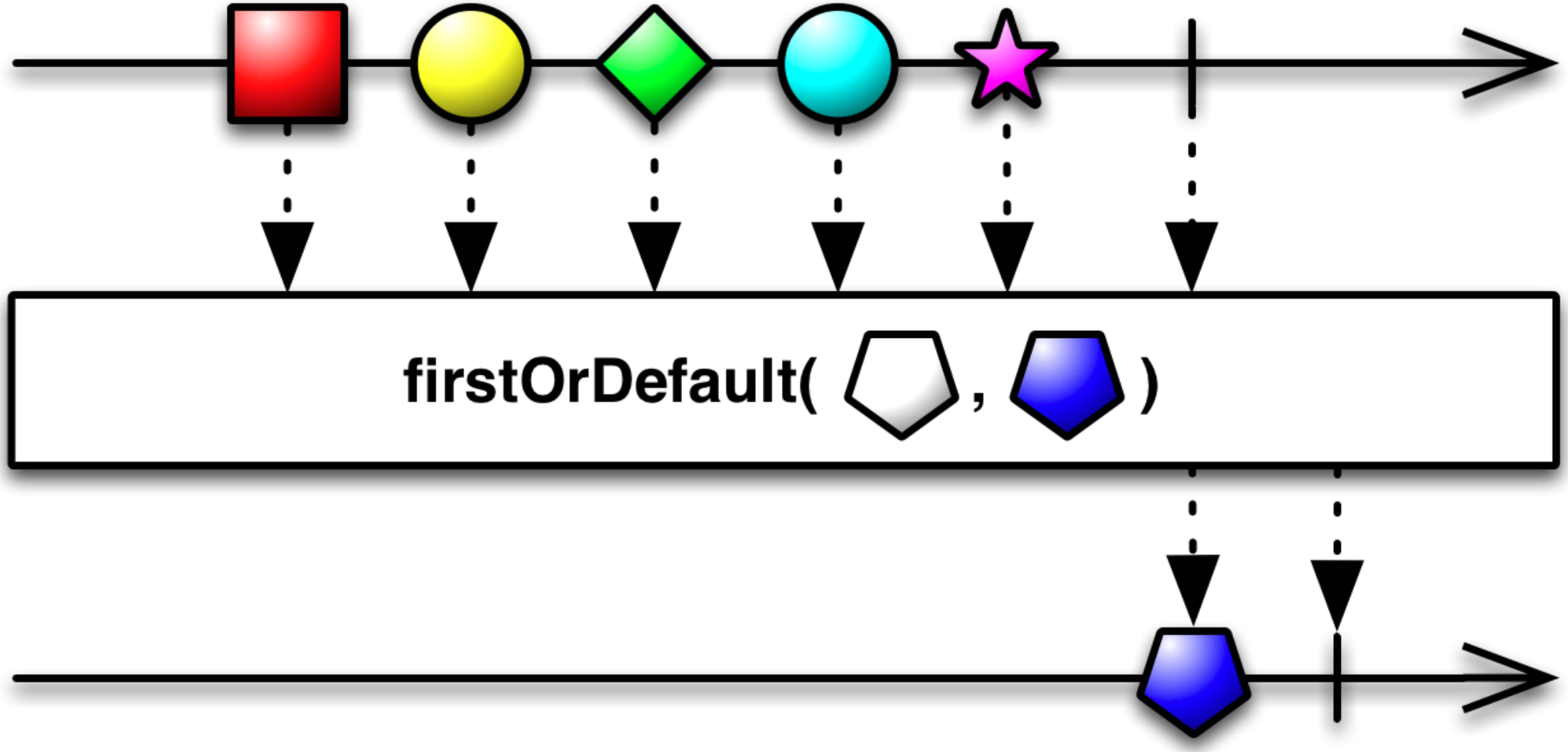
.Cas d'usage : Détecter une inactivité.

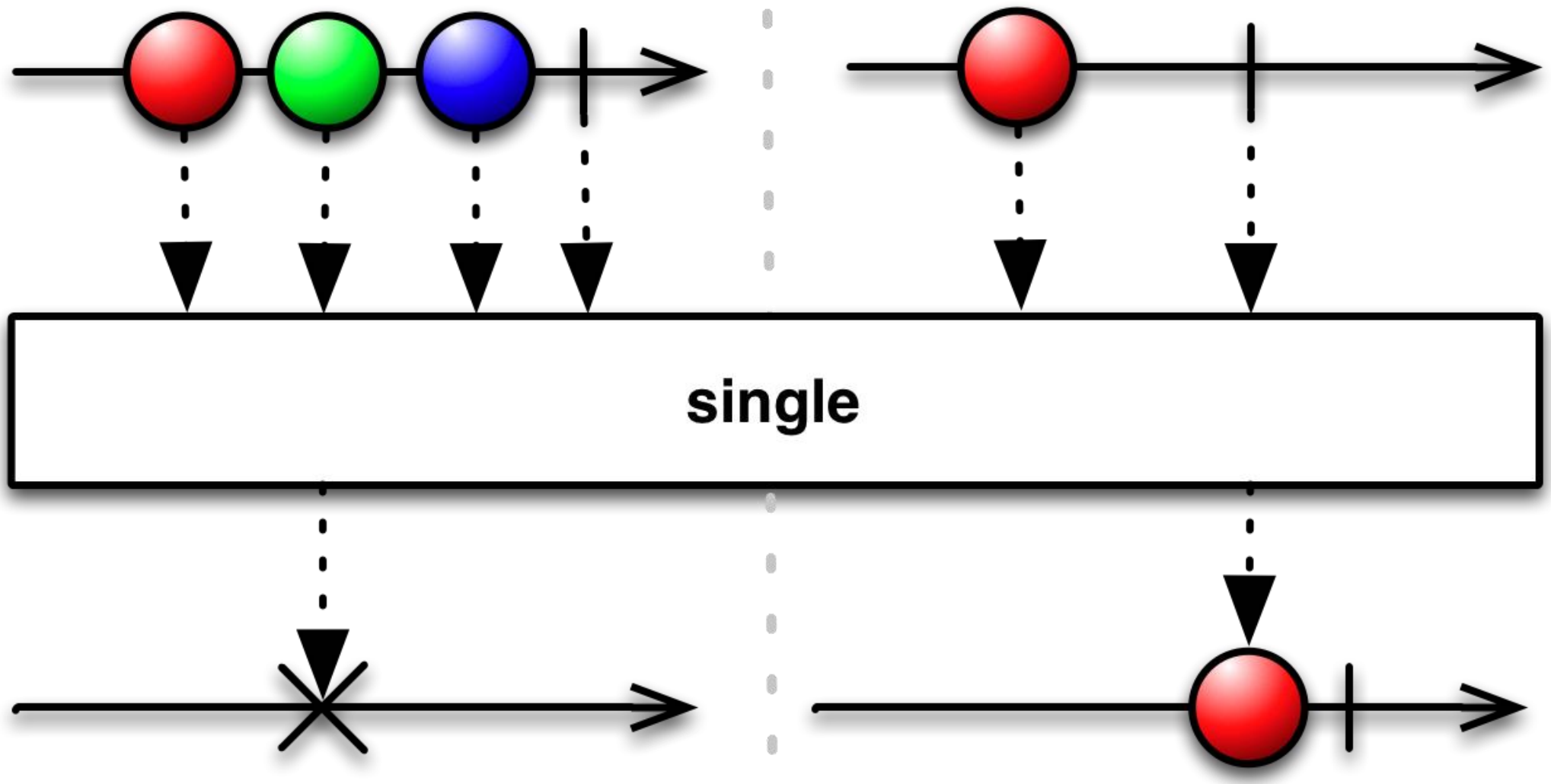

debounce(,)



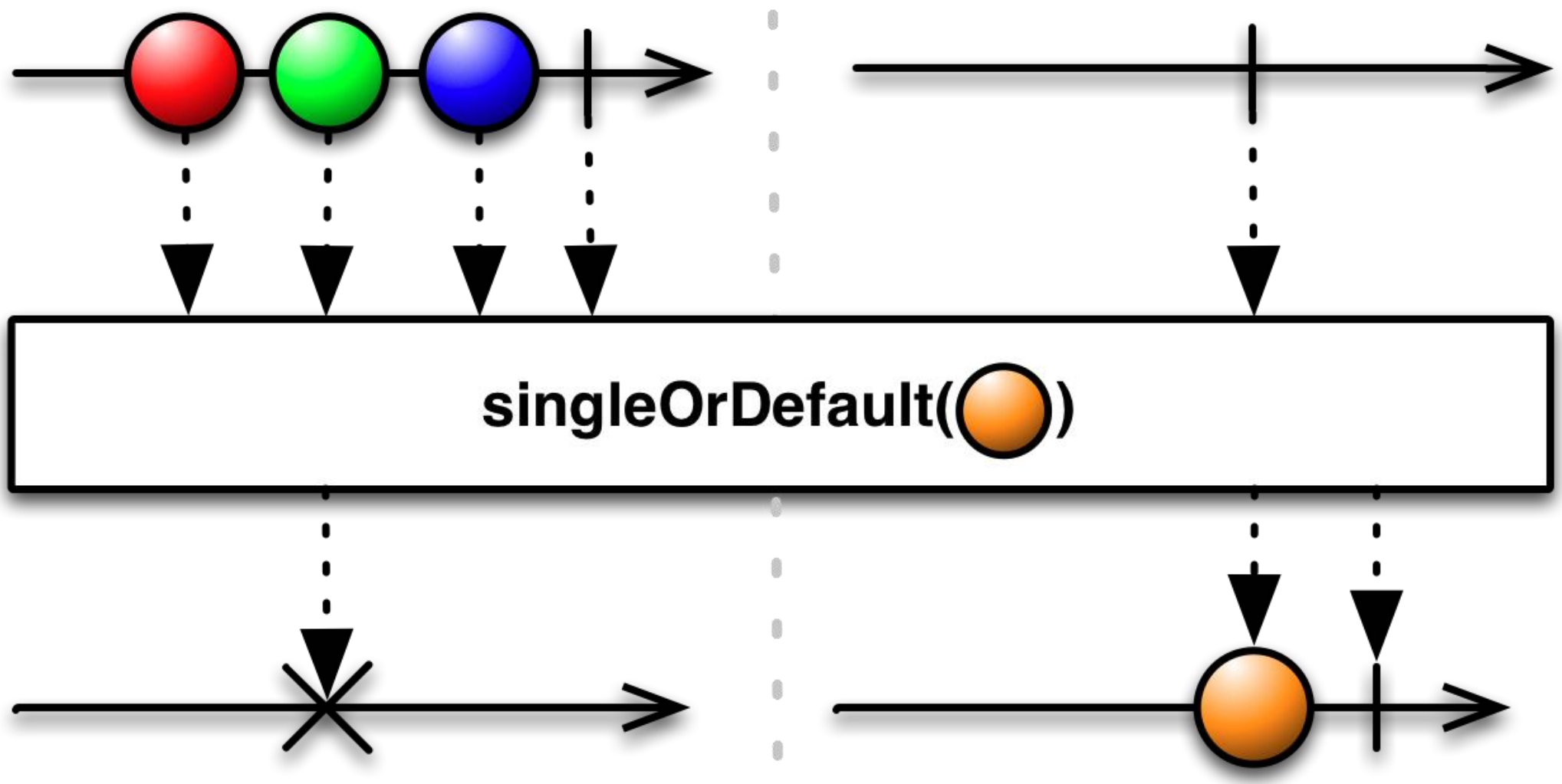
..throttleWithTimeout(...) == .debounce(...)

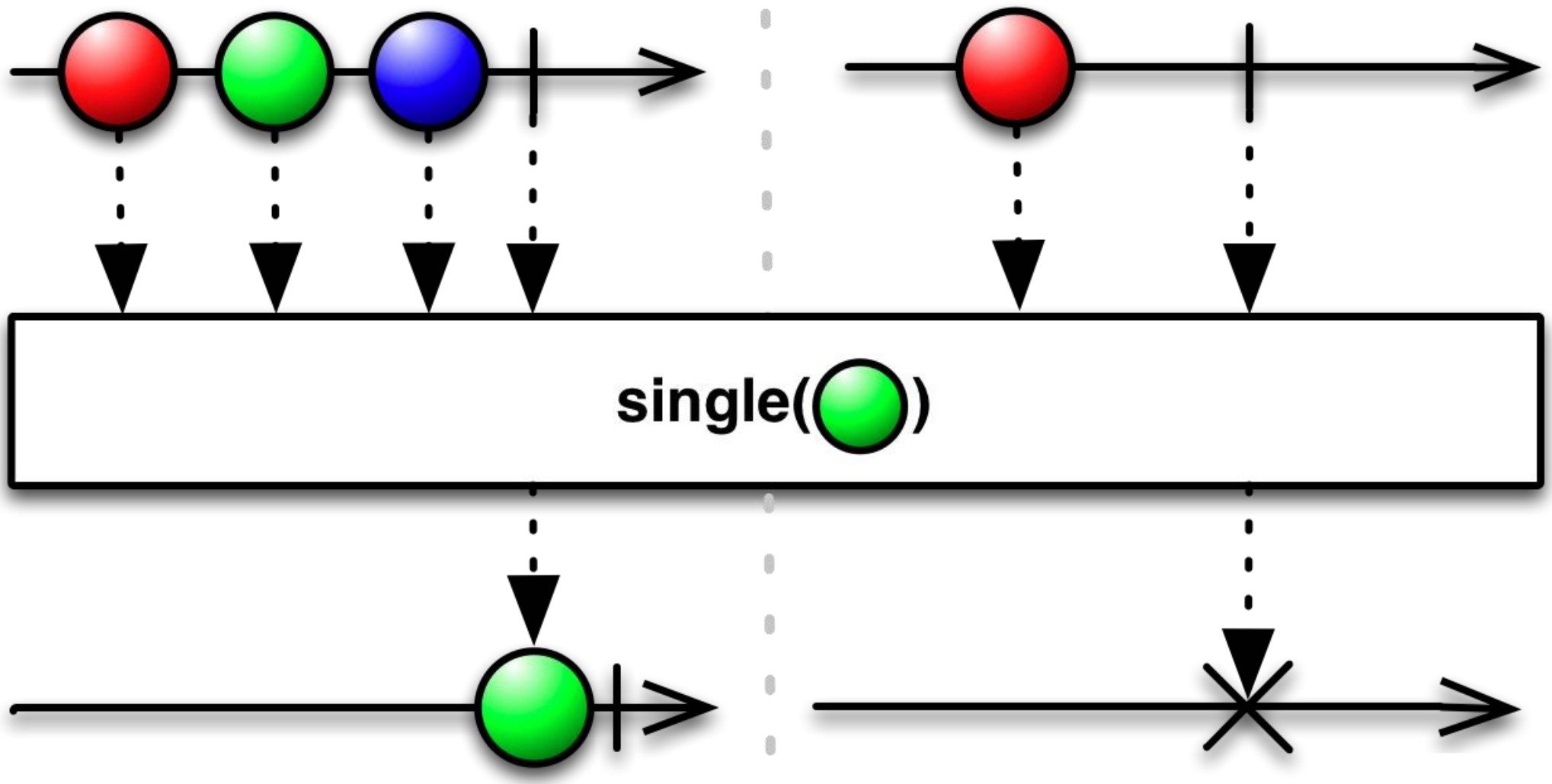

1
2
3
debounce(f(n) =)

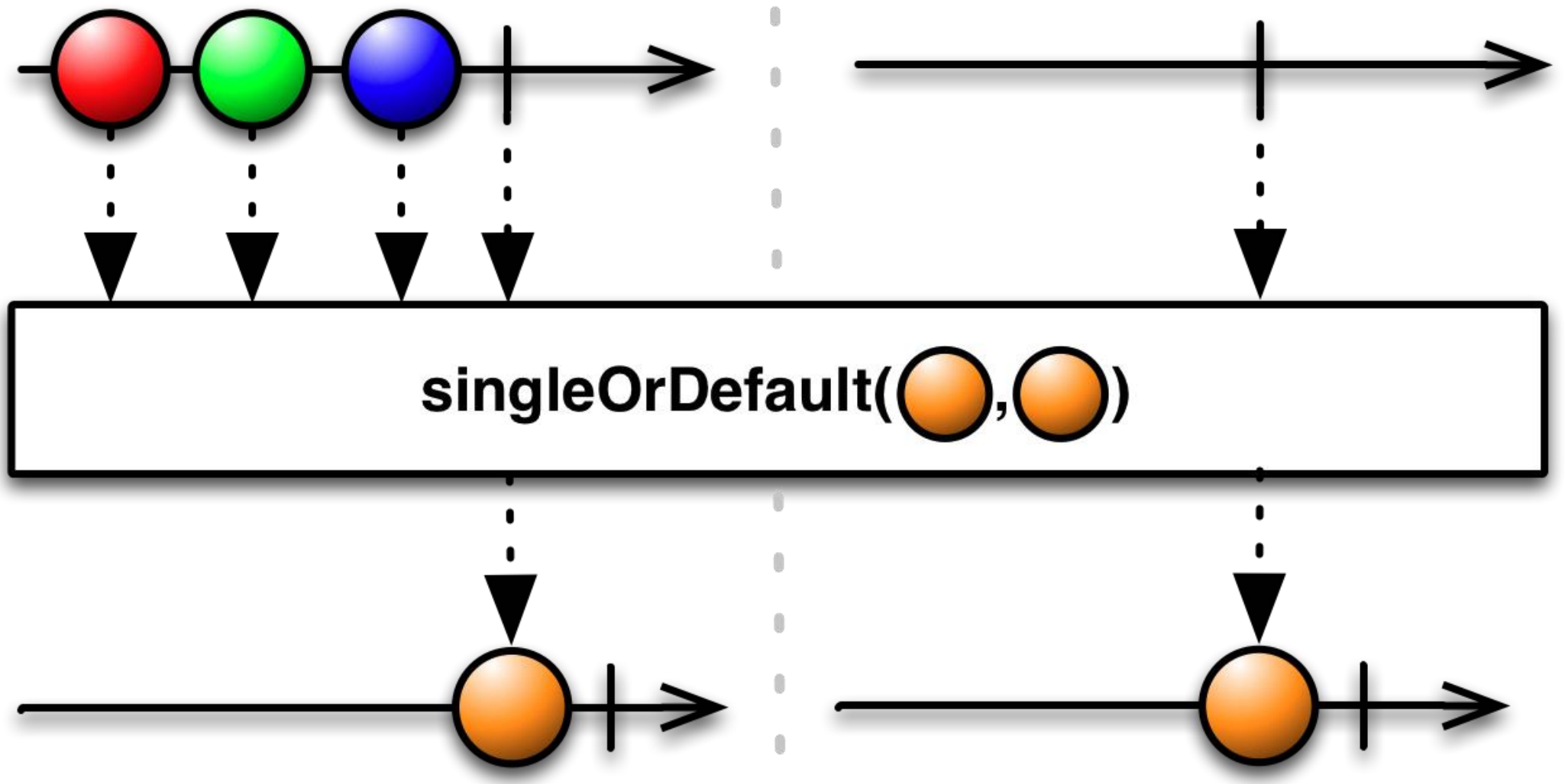

firstOrDefault(,)



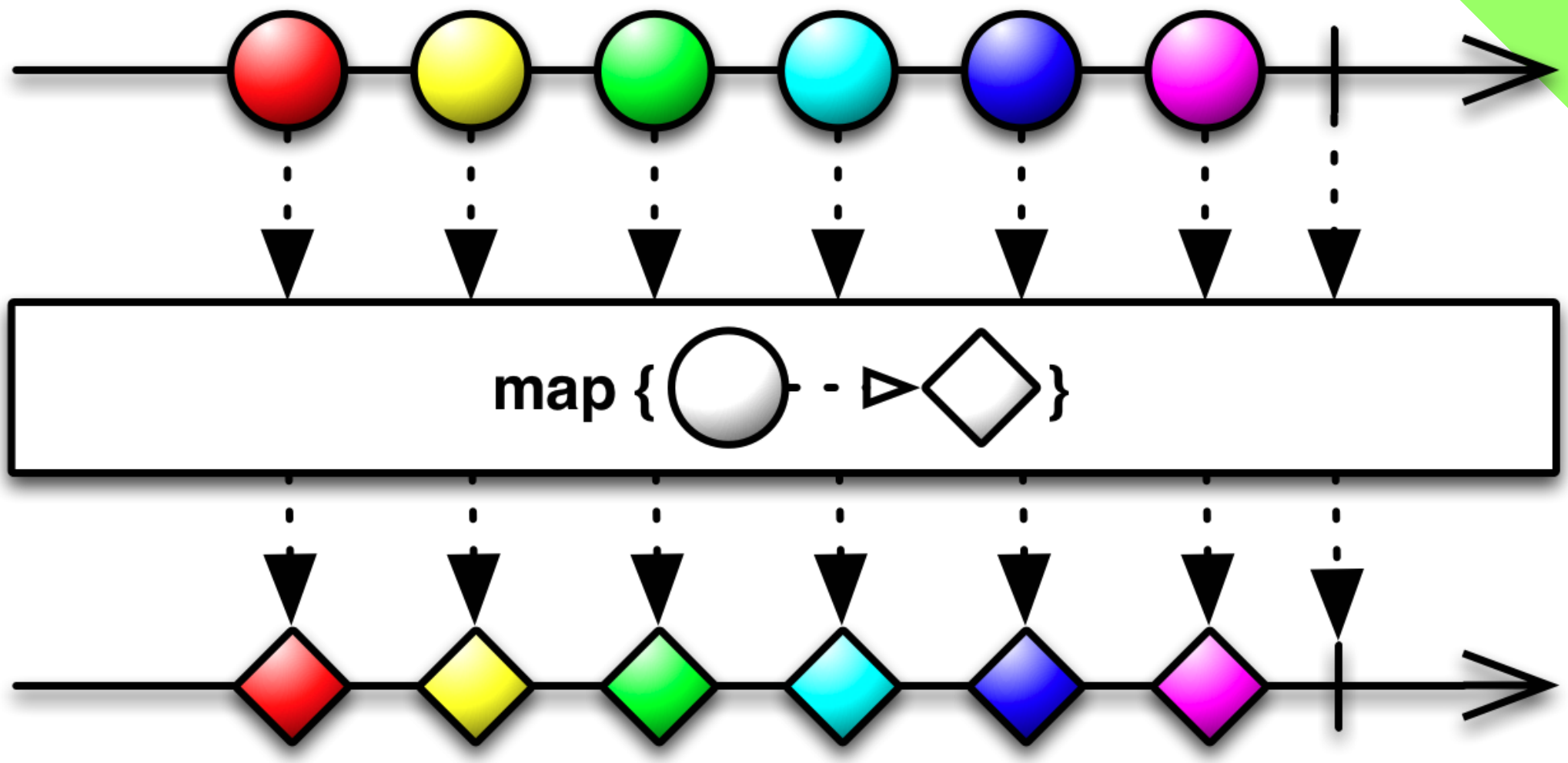
- Cas d'usage : Vérifier qu'il y a bien eu une seule valeur.

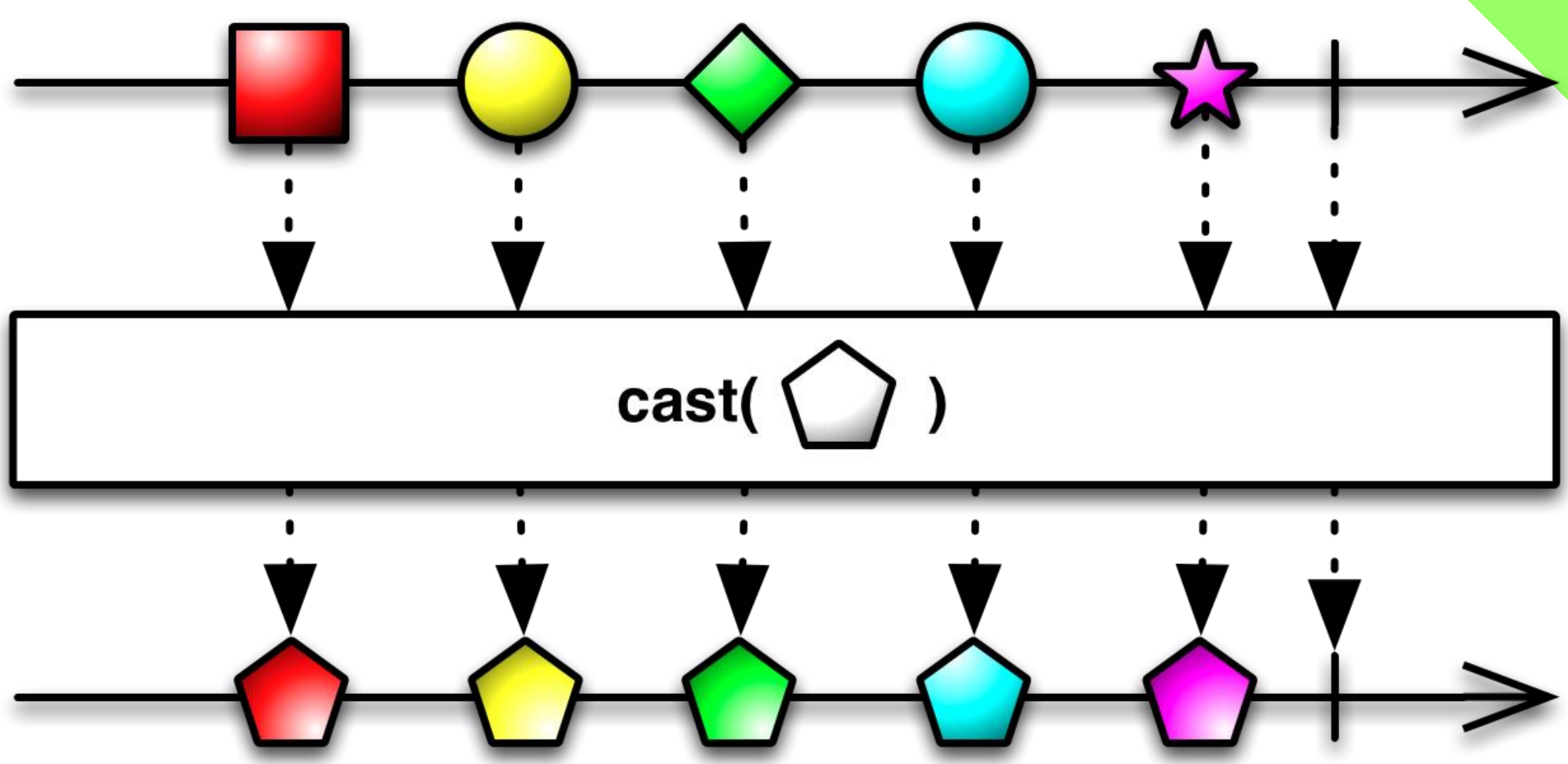

singleOrDefault()

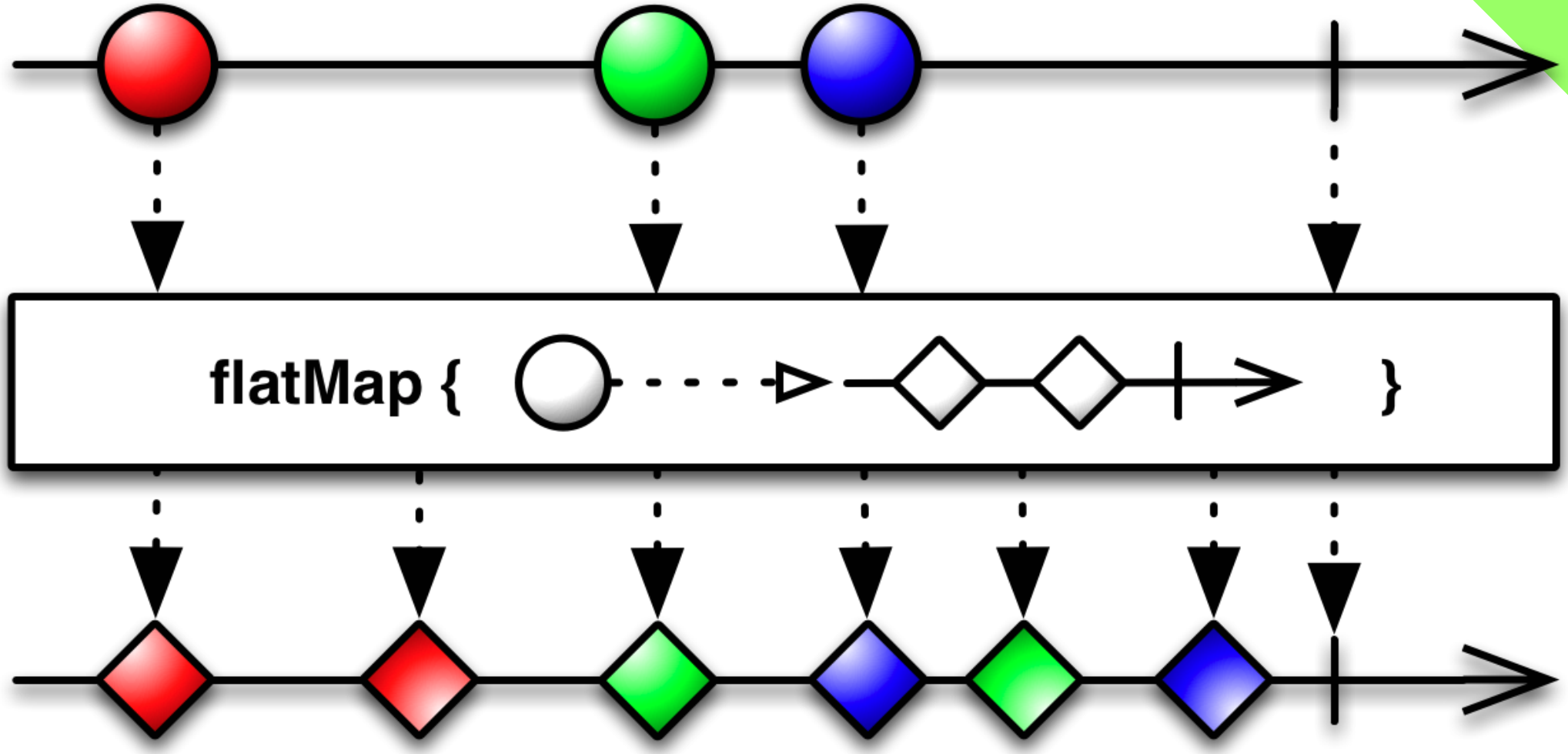

single(

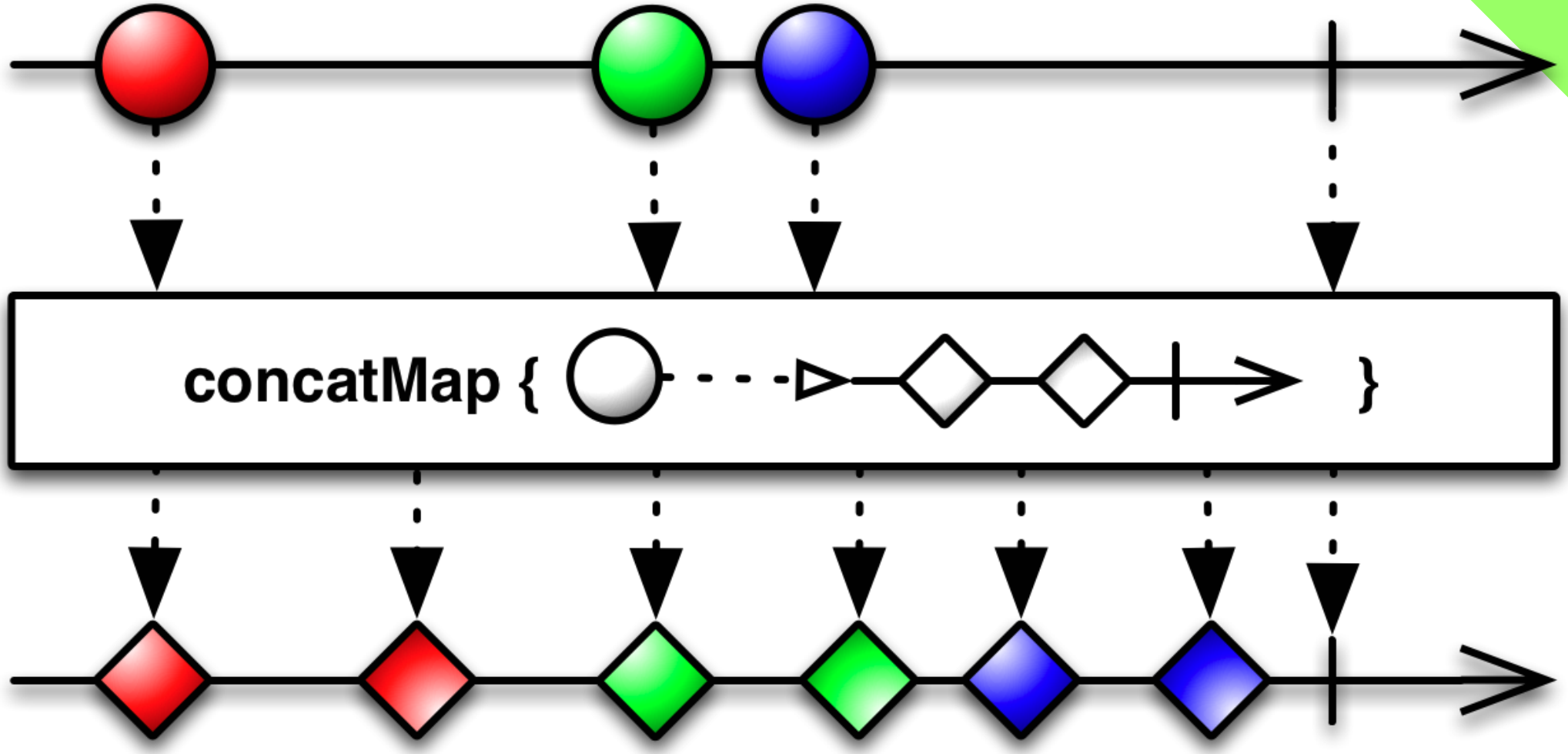

singleOrDefault(,)

Transformation

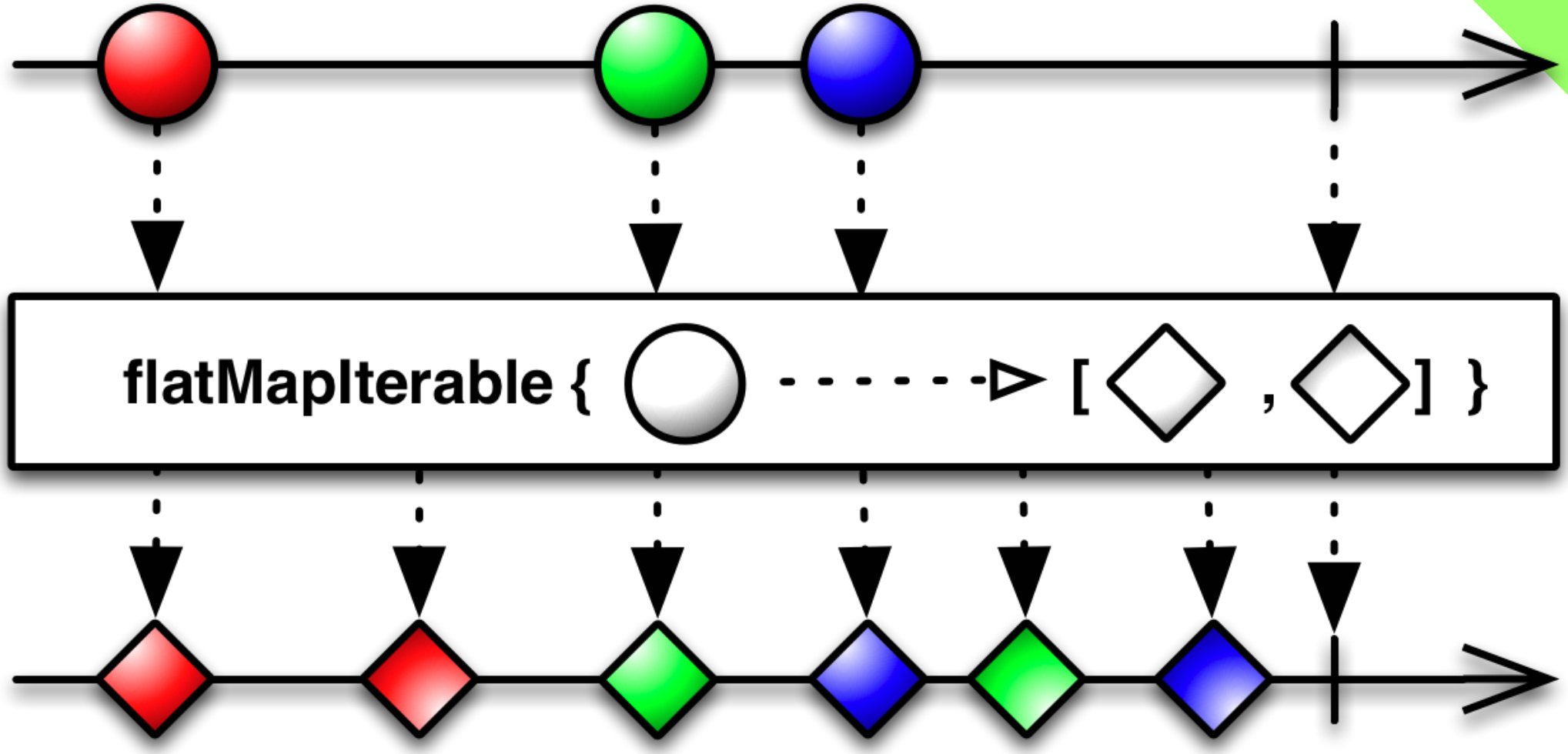

map {


cast( )

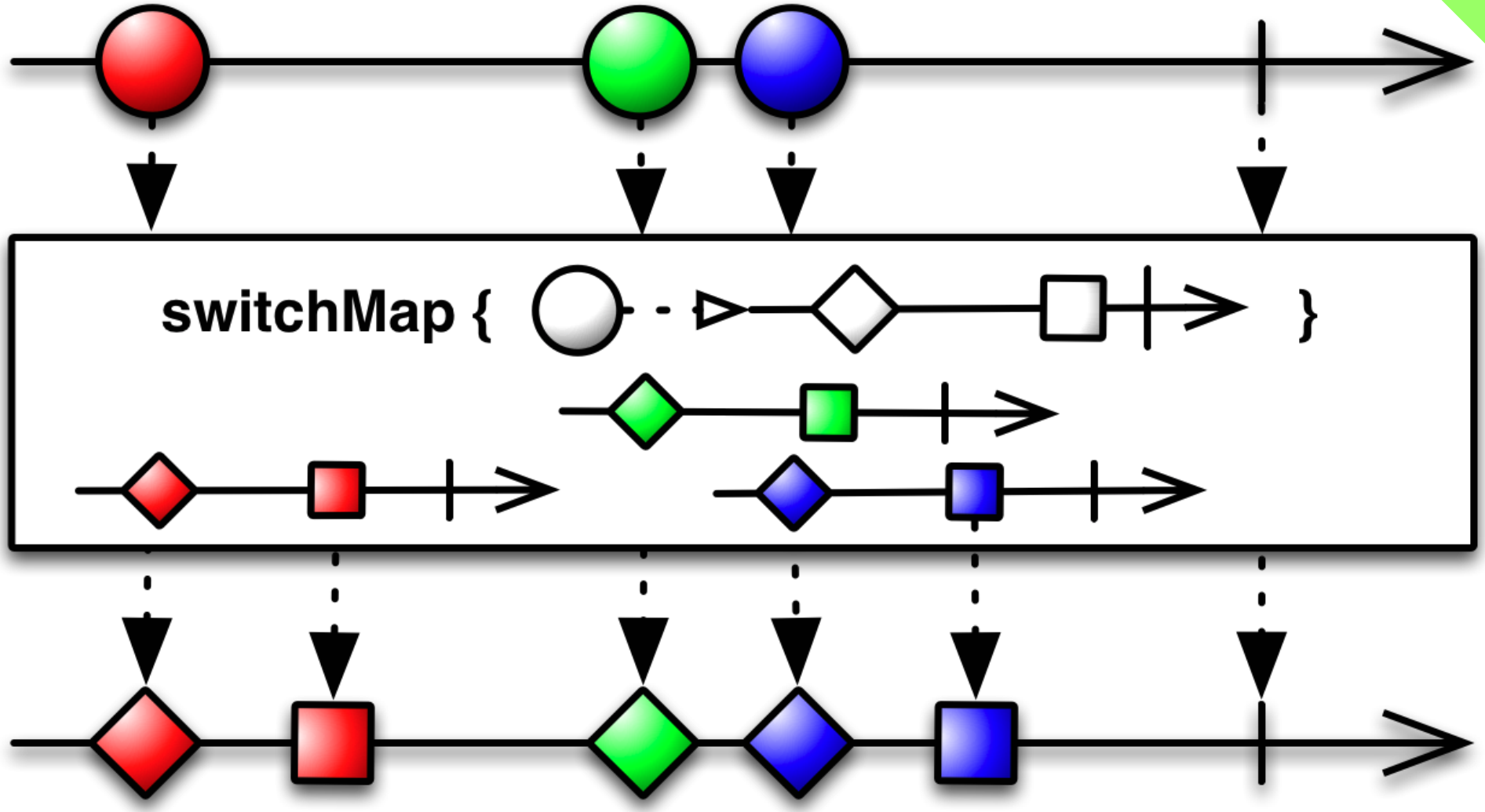

flatMap {
}



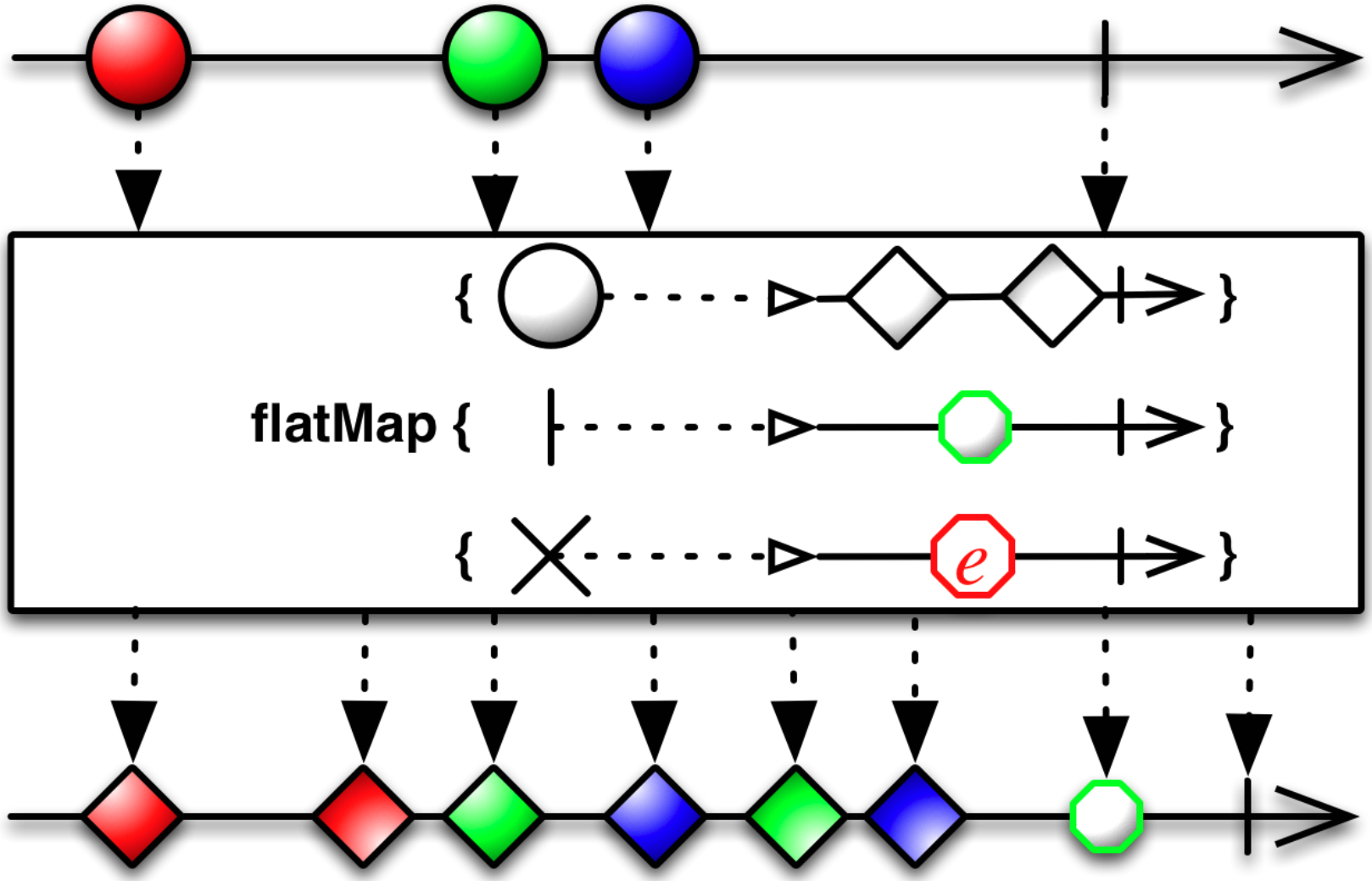
- Cas d'usage : Composition tout en gardant l'ordre.



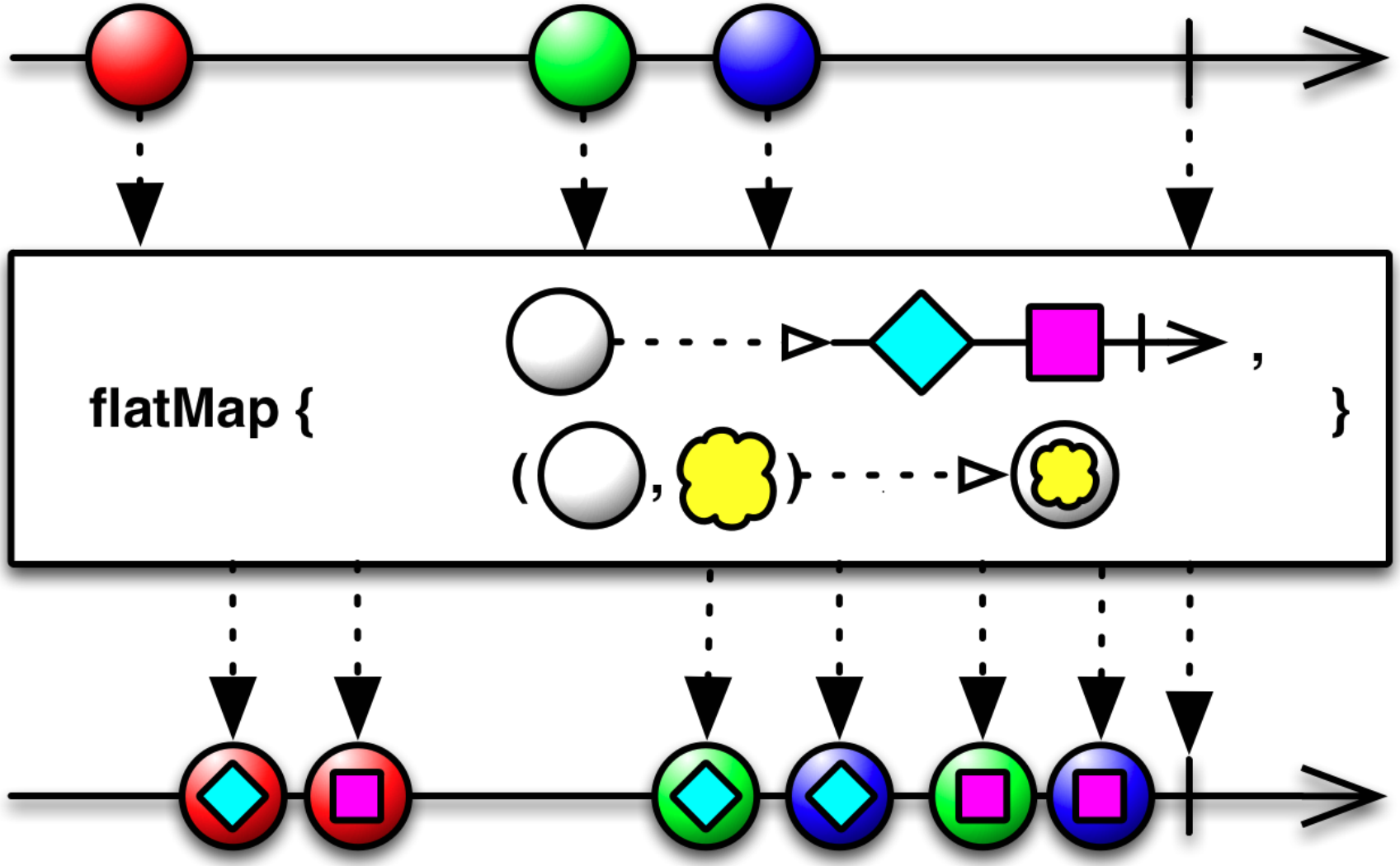
- Cas d'usage : Évite un Observable.from(...).



- Cas d'usage : Composition tout en ayant toujours la valeur la plus « fraîche ».

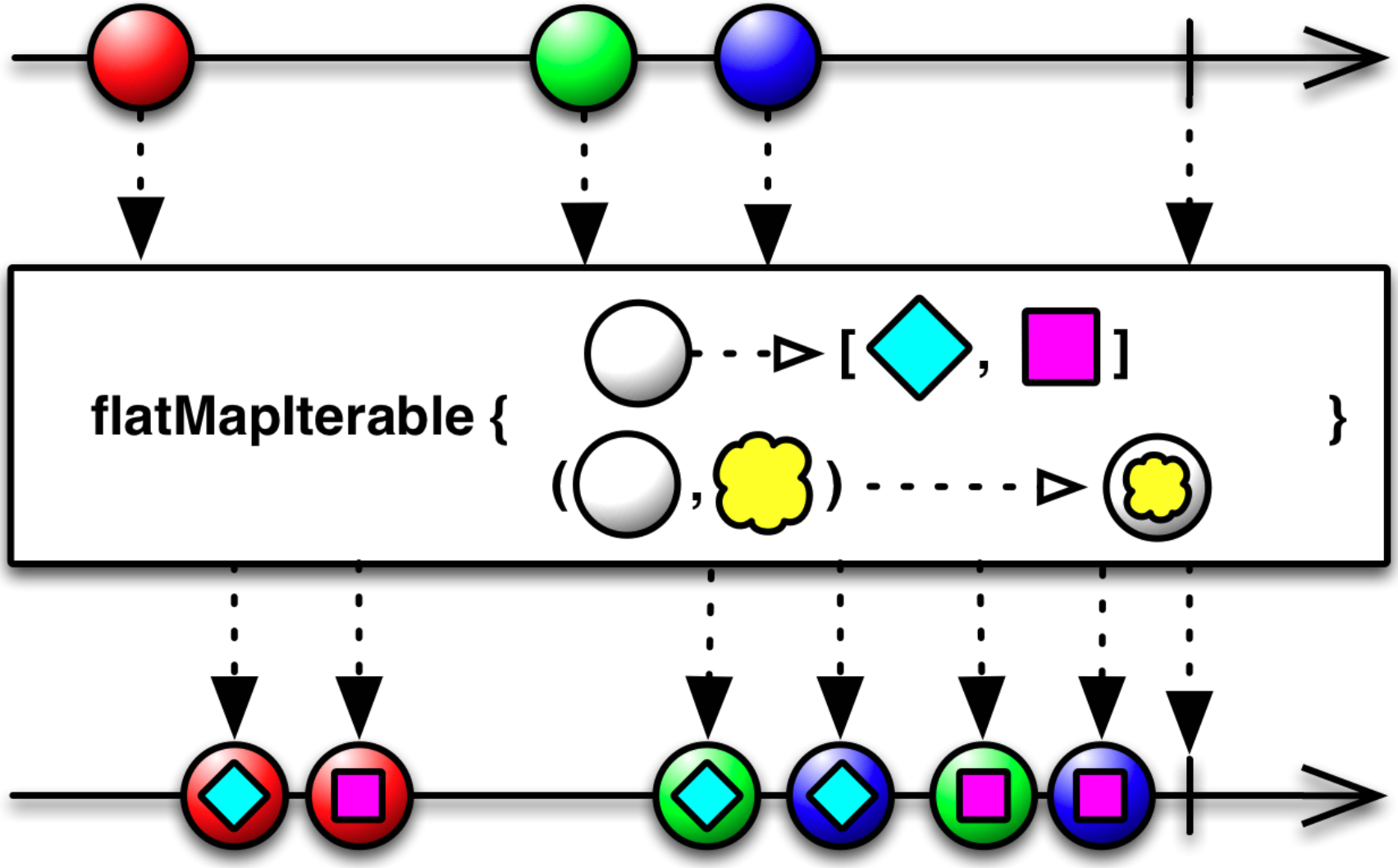


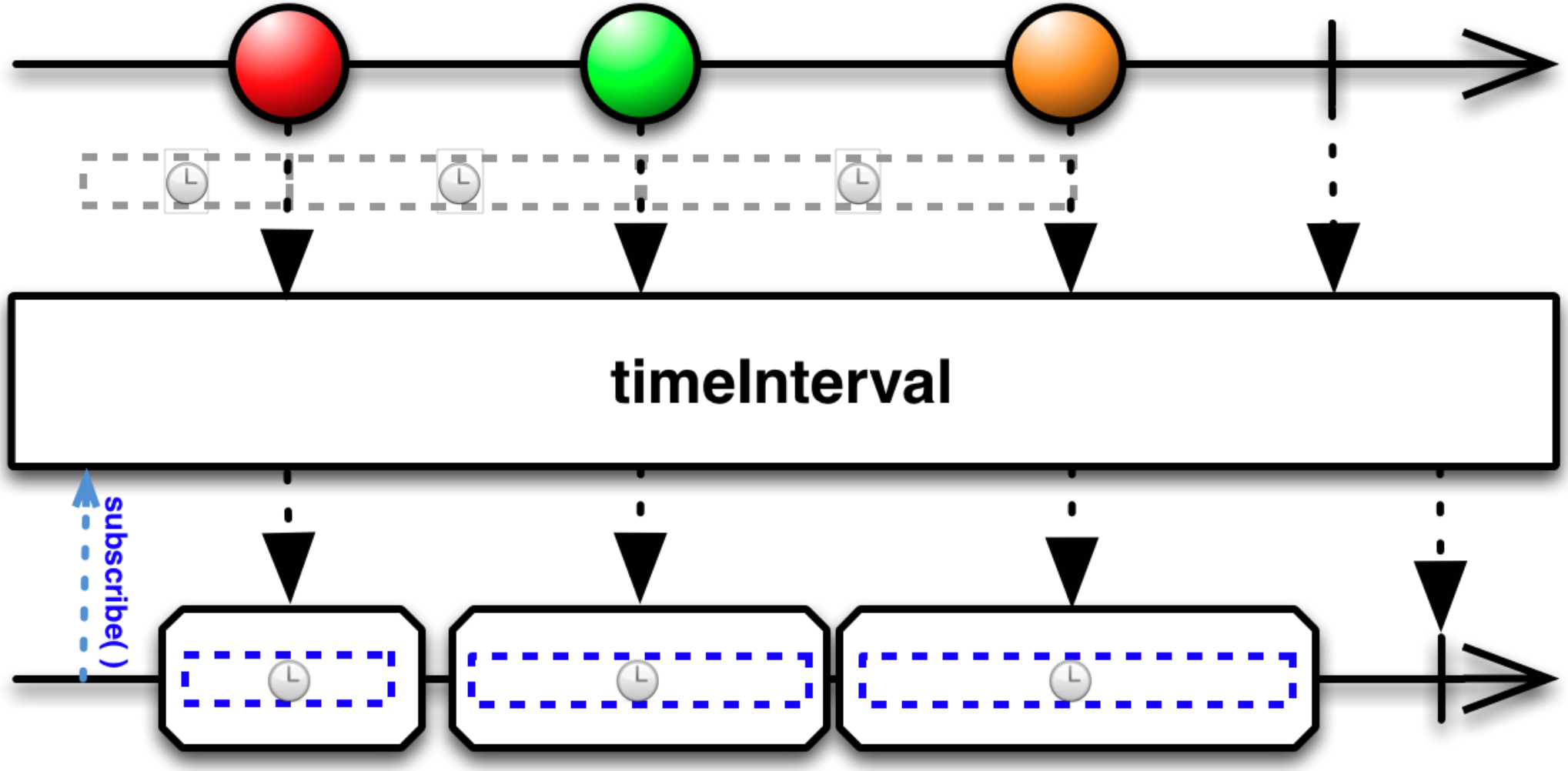
- /!\ Attention avec la terminaison du flux /!\

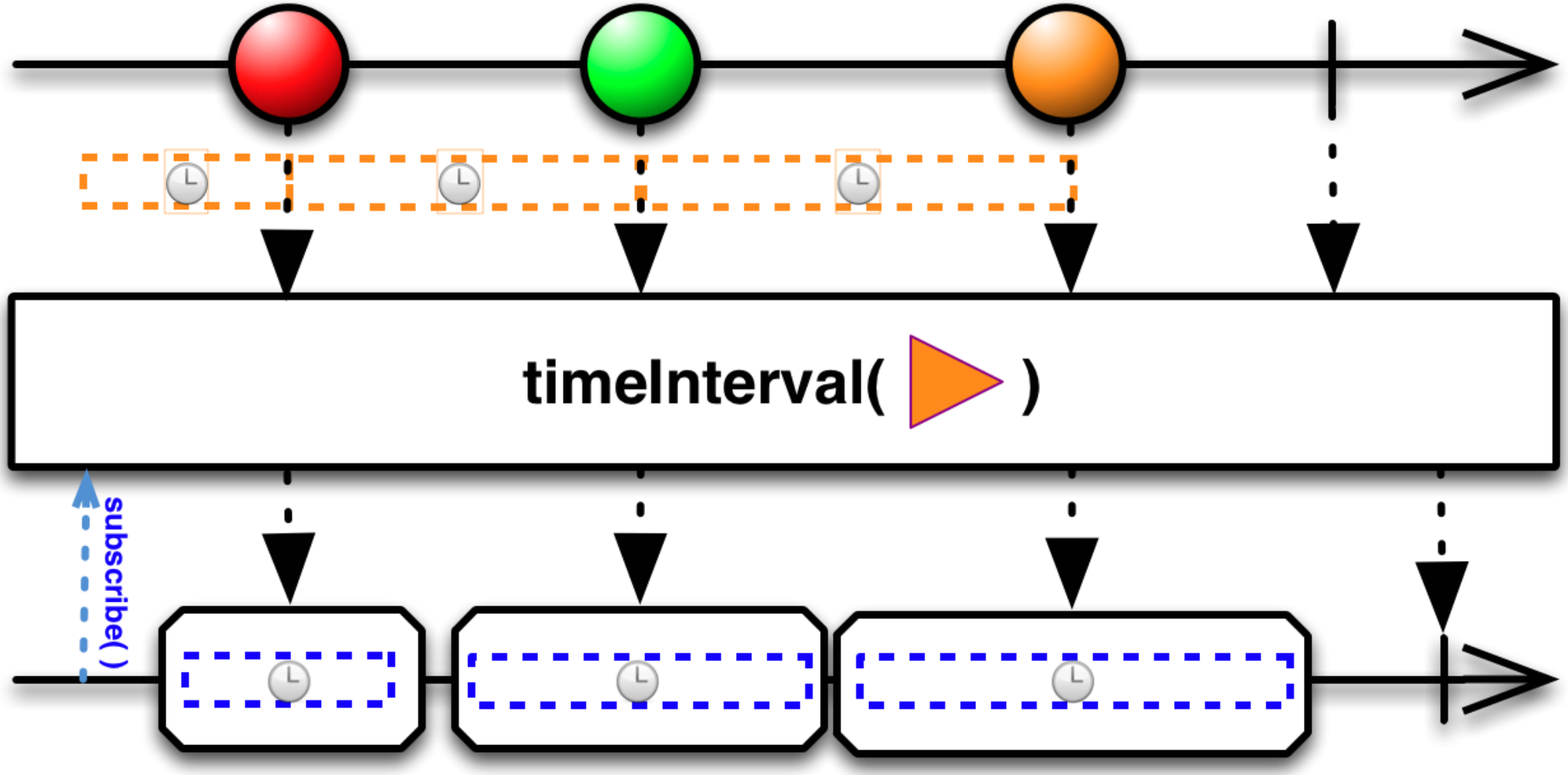


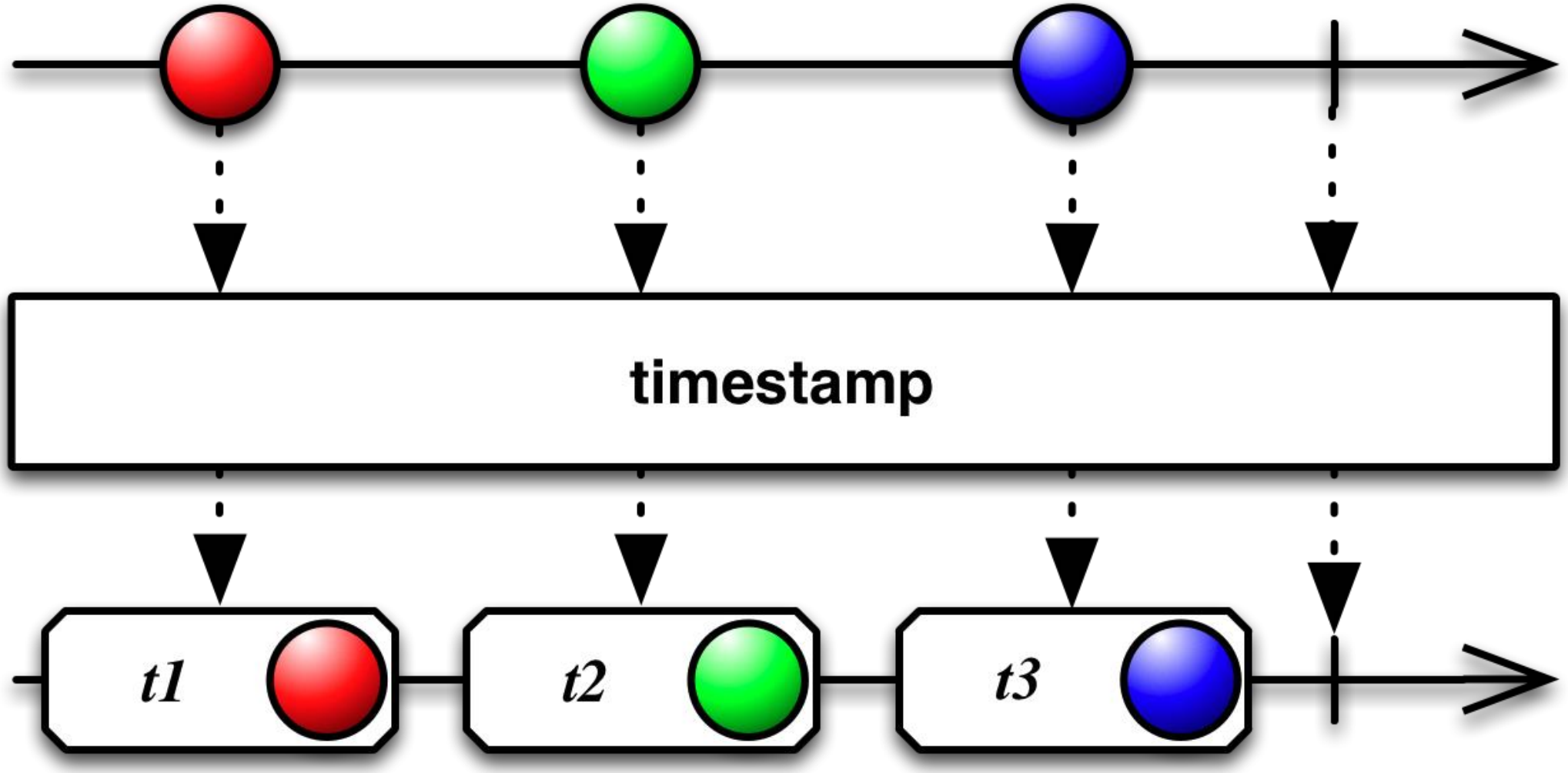
.flatMap(Func1<T, Observable<U>> collectionSelector, Func2<T, U, R> resultSelector)

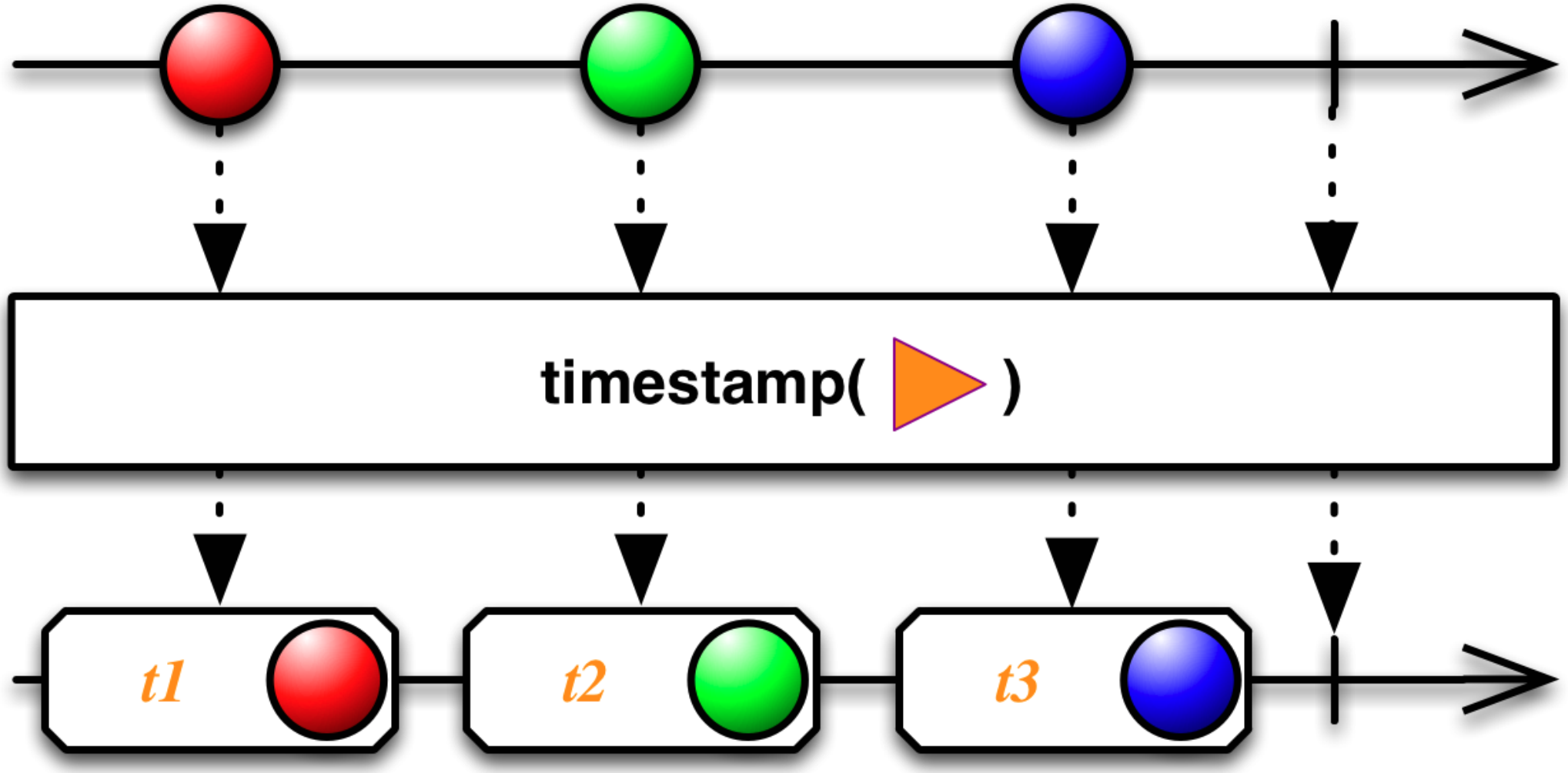
.<==> flatMap(t -> newObs(t).map(u -> f(t, u))


flatMapIterable {
}

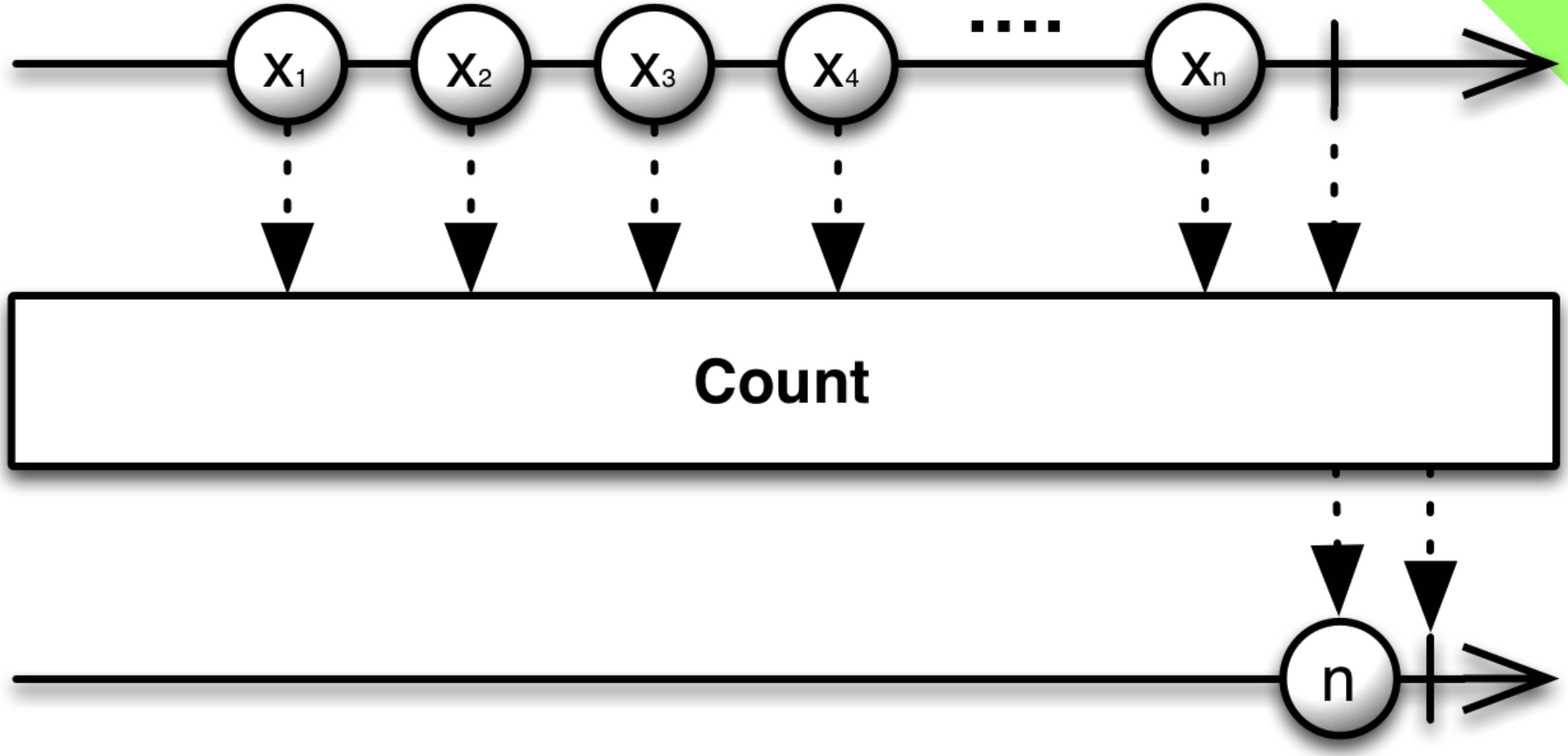

timeInterval
subscribe()

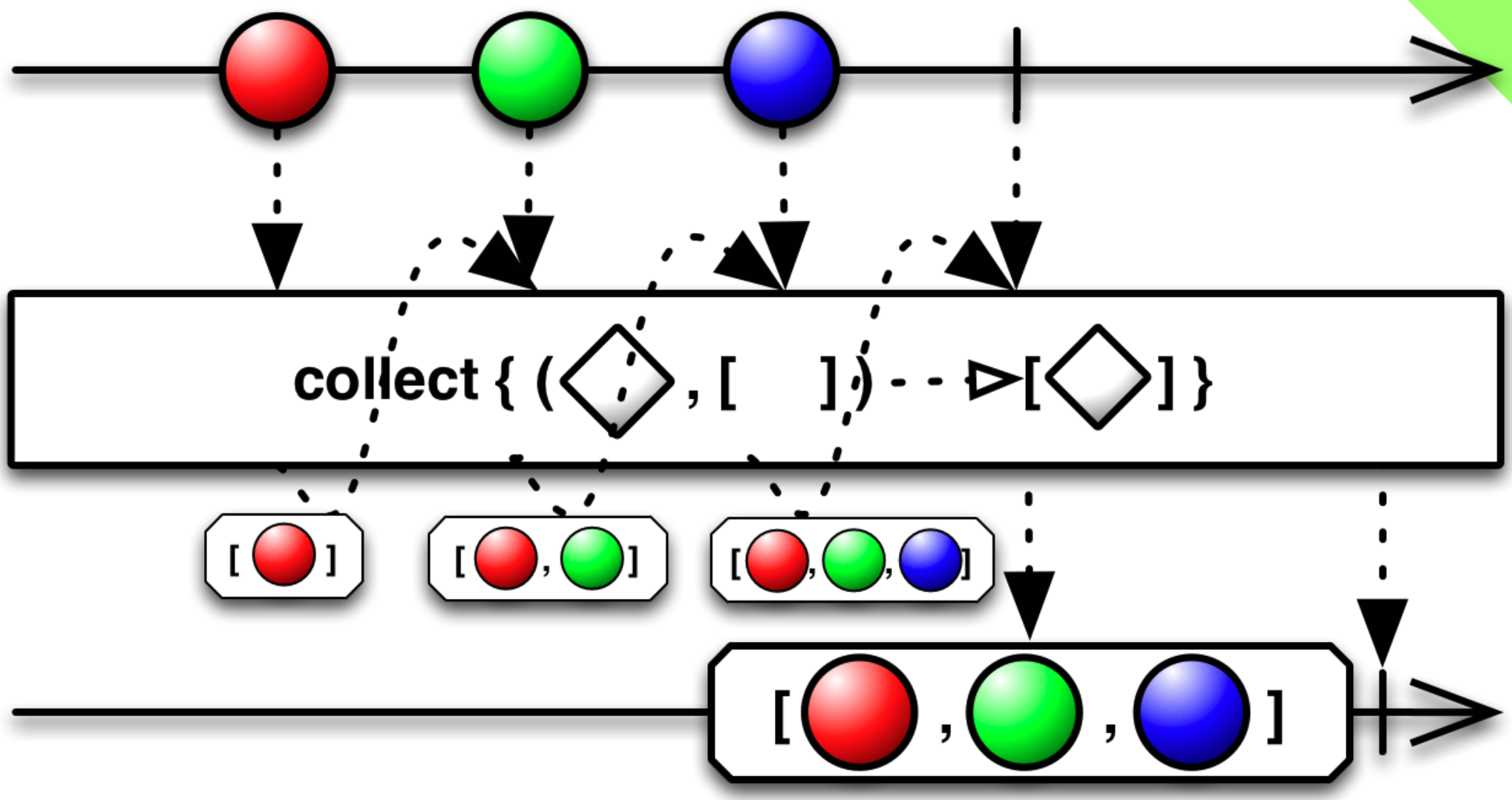

timeInterval( )
subscribe( )


timestamp
t1
t2
t3

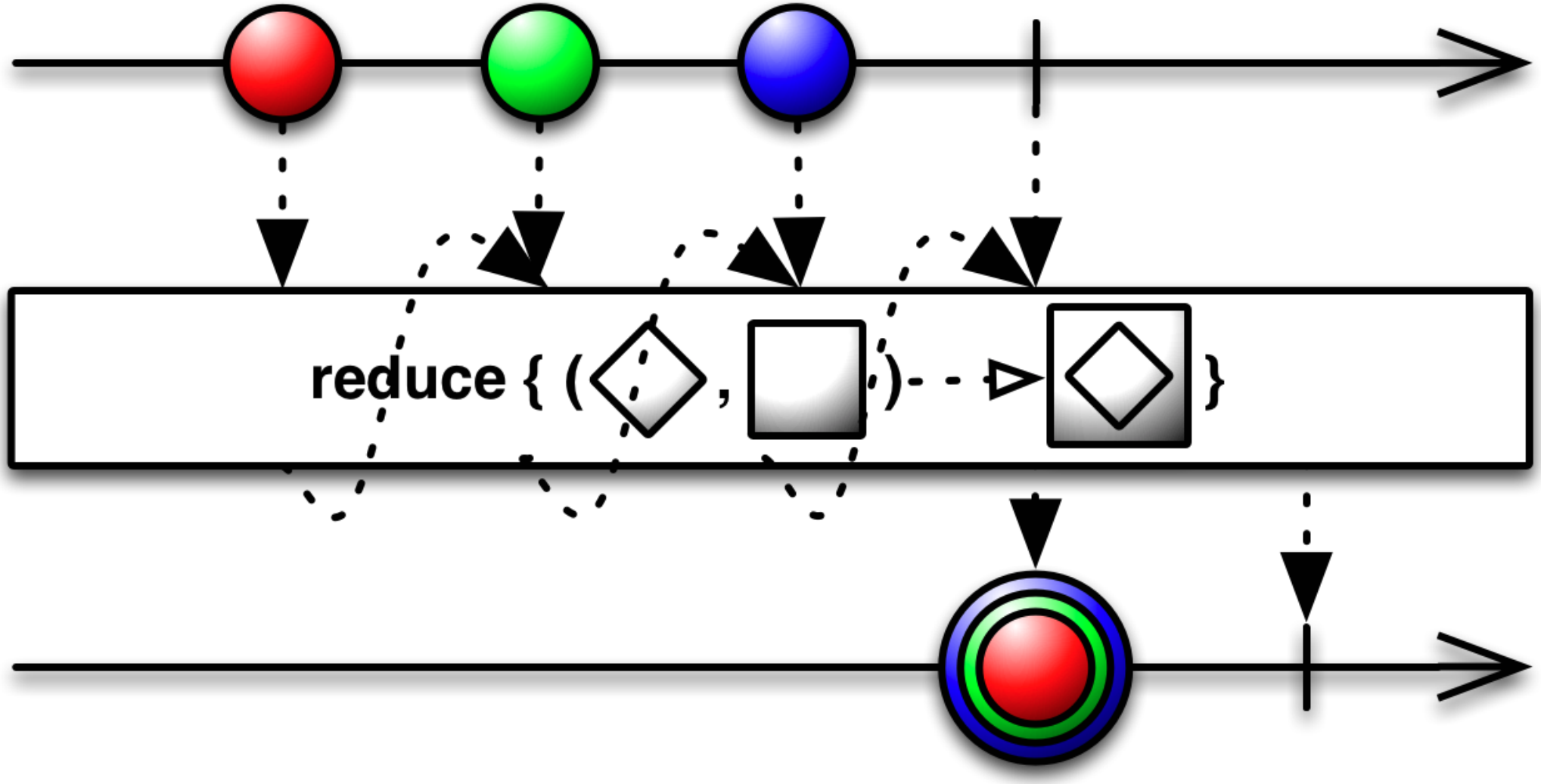

timestamp( )
t1
t2
t3

Reduction

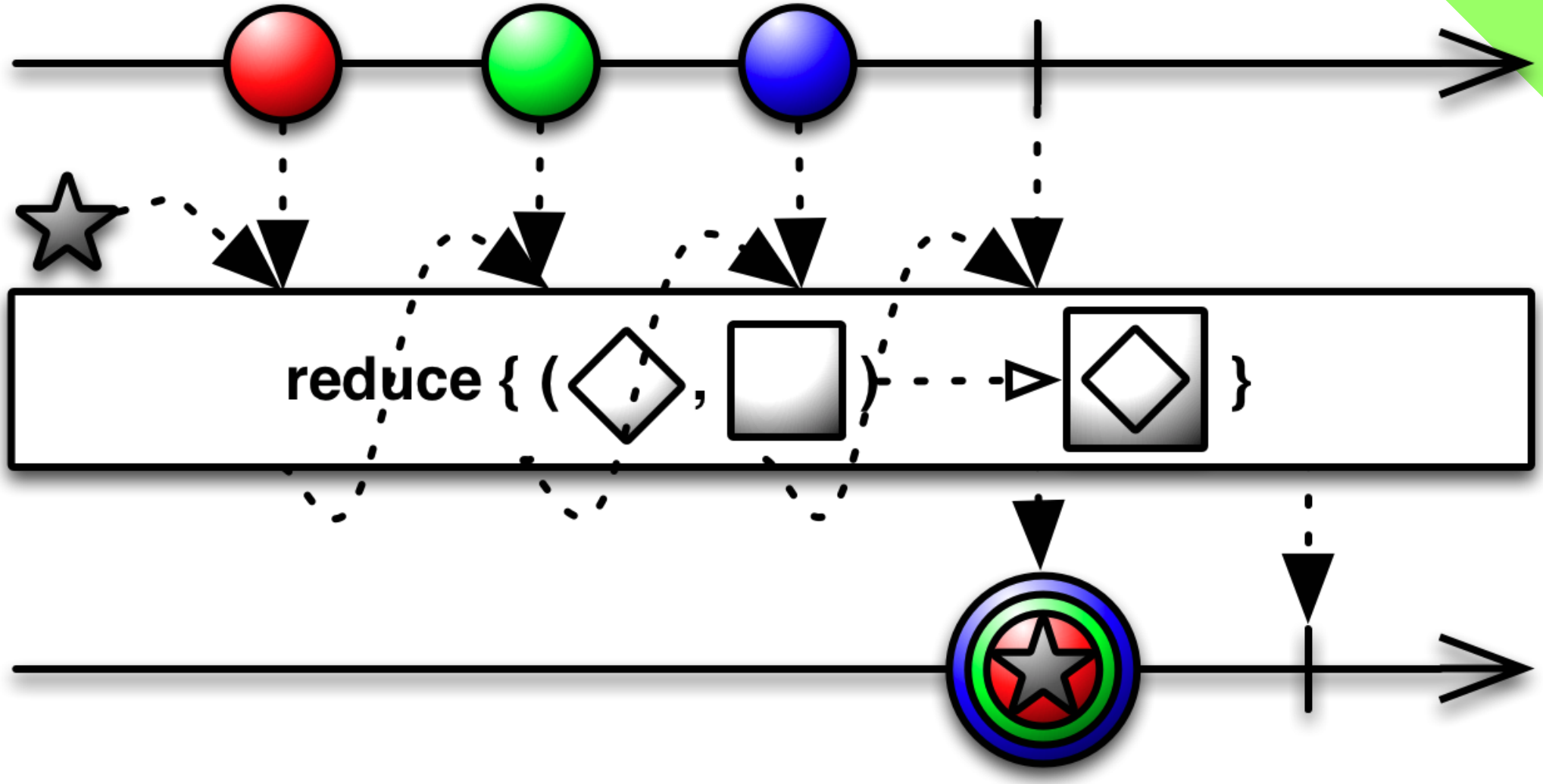

x1
x2
x3
x4
xn
Count
n



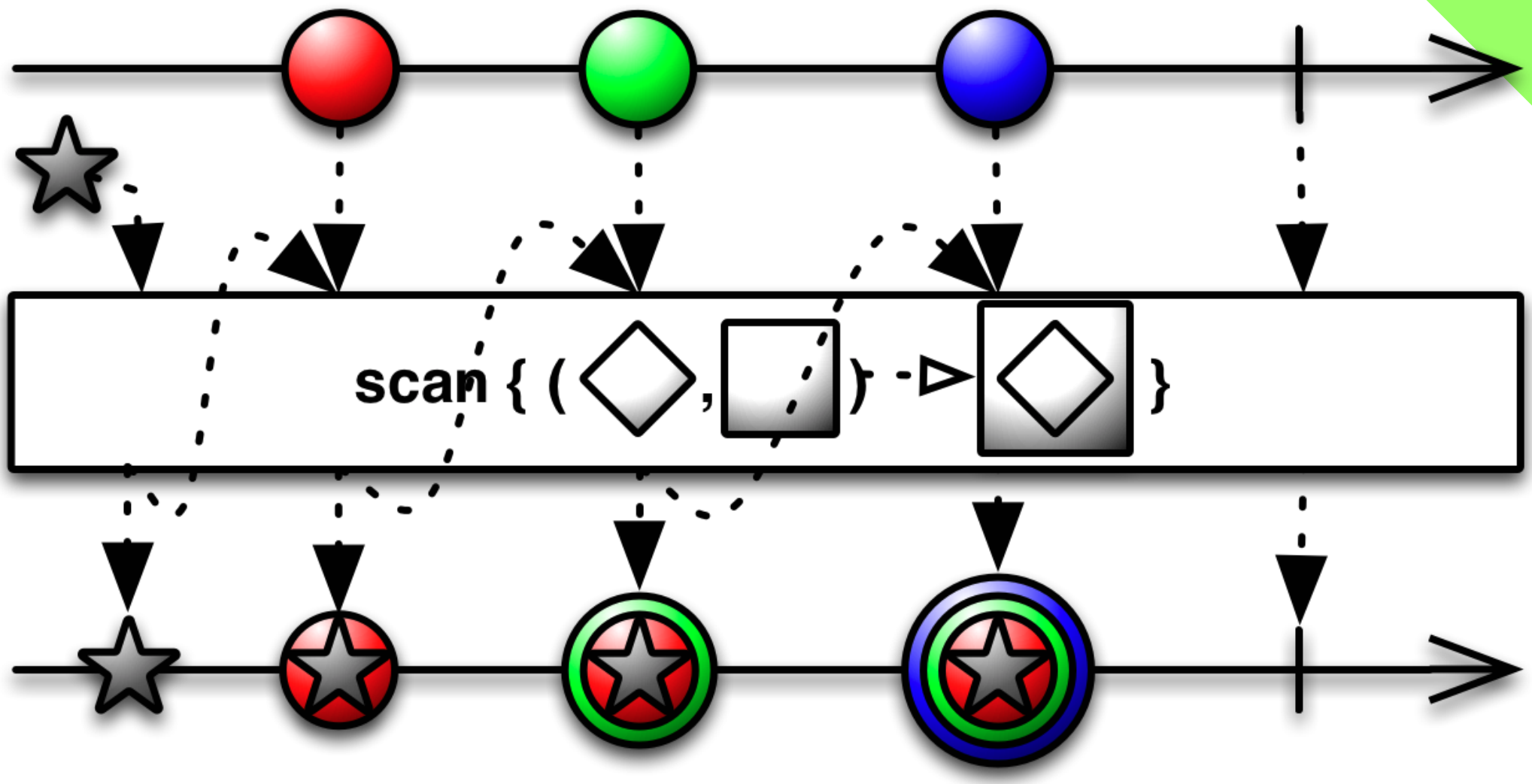
- Cas d'usage : Réduction mutable.



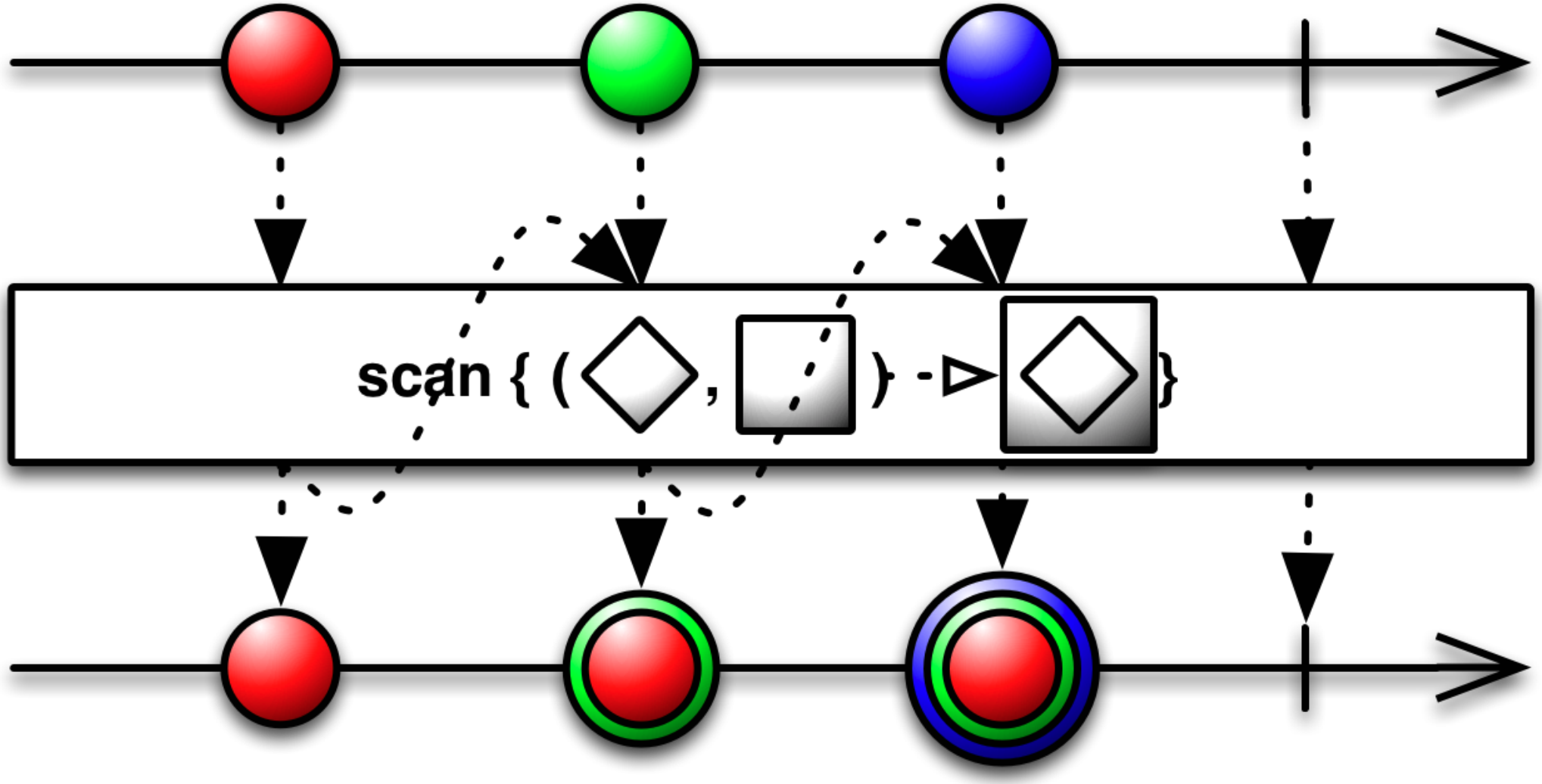
- Cas d'usage : Réduction immutable dans le même type.



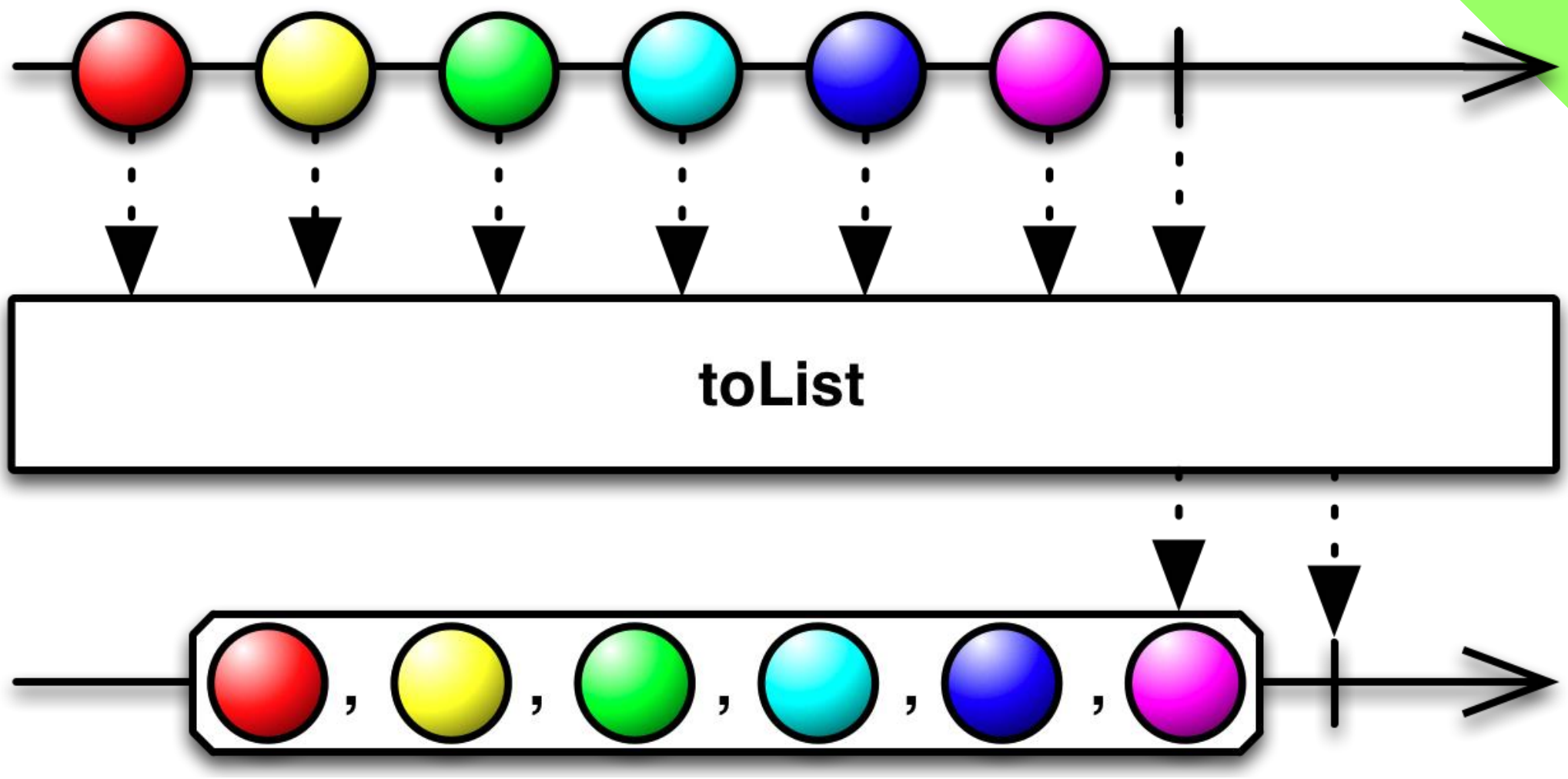
- Cas d'usage : Réduction immutable avec une valeur initiale. Potentiellement pour changer de type.

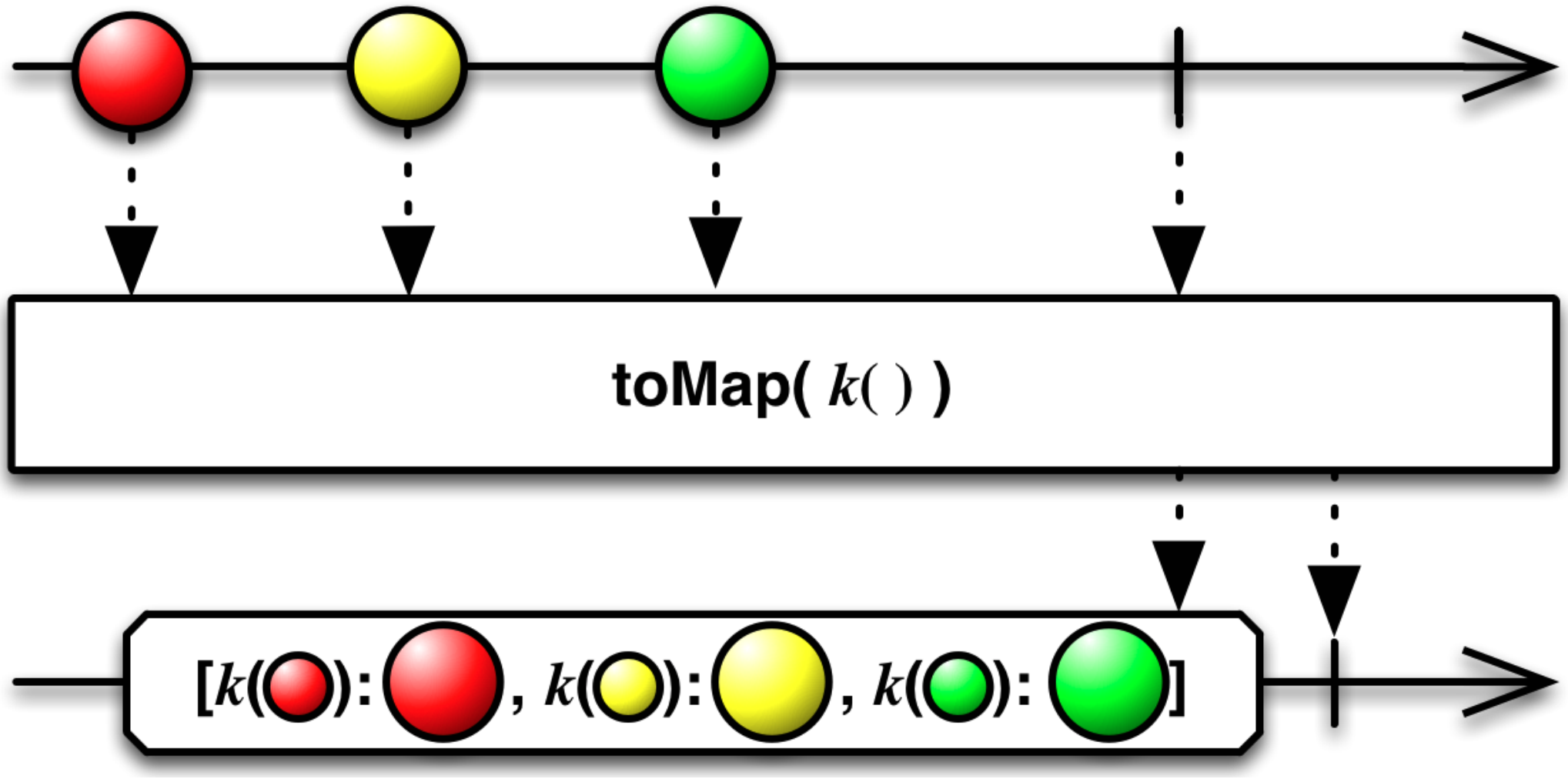


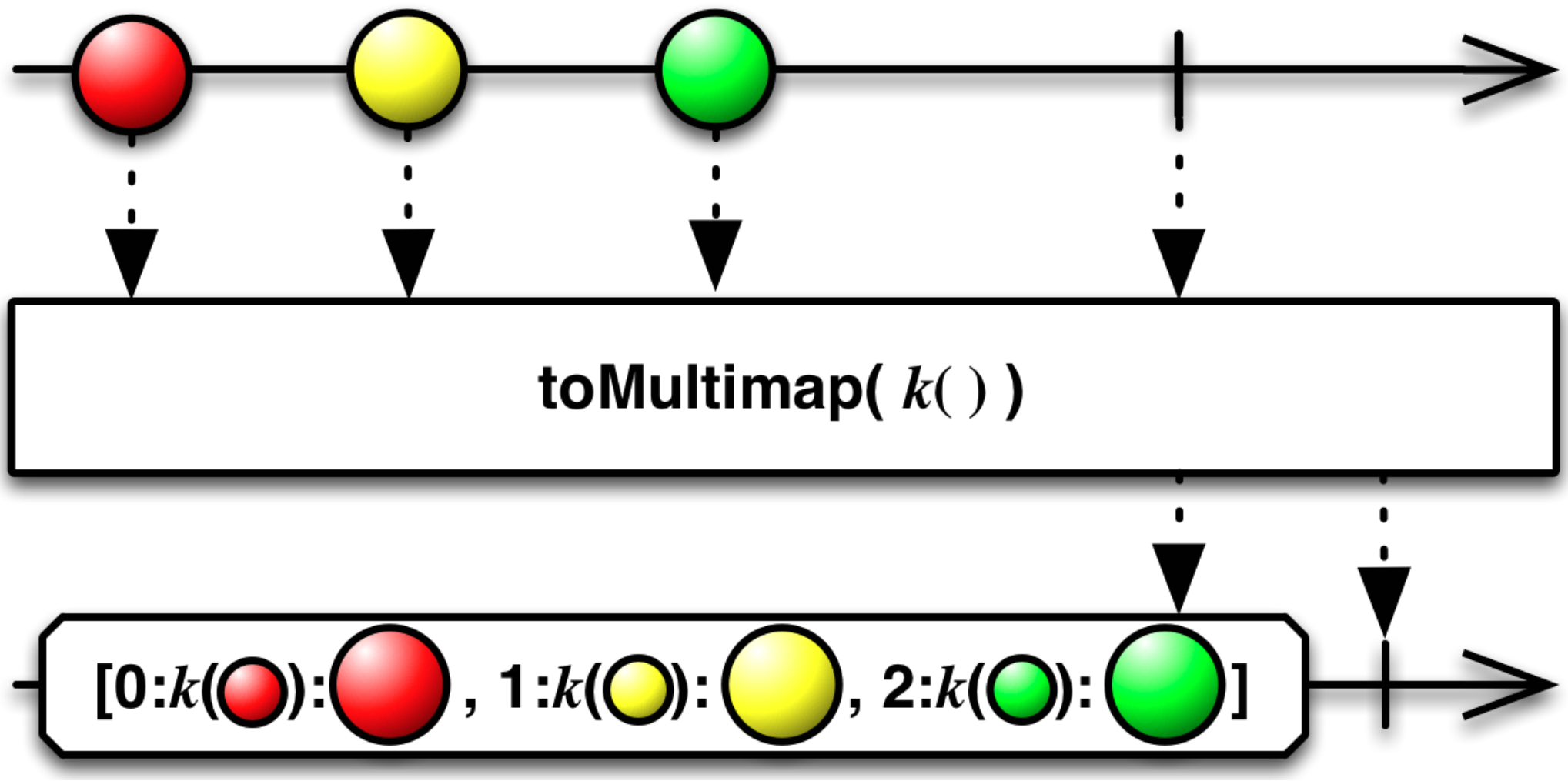
- Cas d'usage : Réduction immutable avec émission des états intermédiaires.
- Attention : la valeur initiale de `.scan(...)` est émise !
- Sans émission de la valeur initiale :

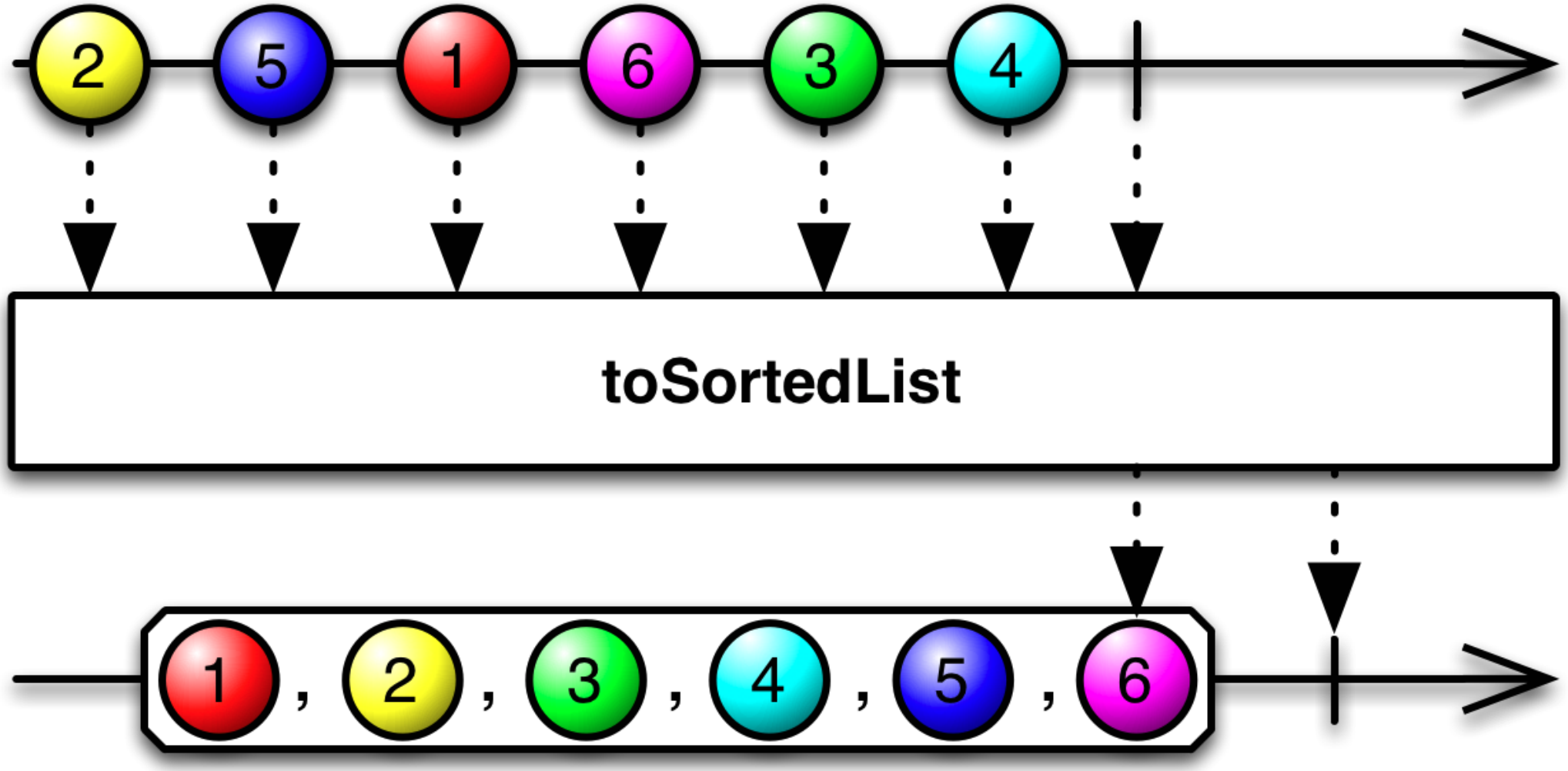


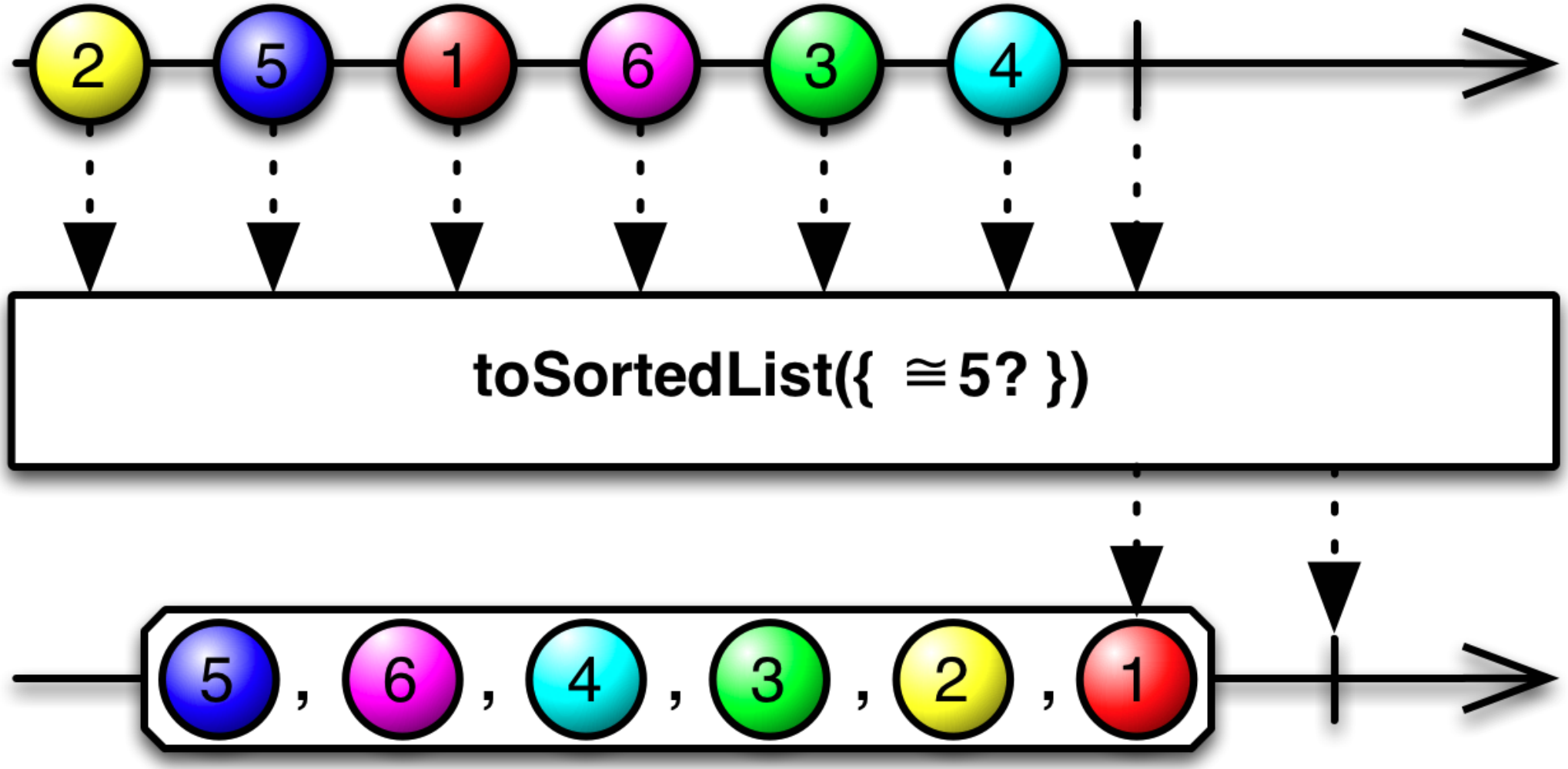
- Cas d'usage : Réduction immutable dans le même type avec émission des états intermédiaires.


toList

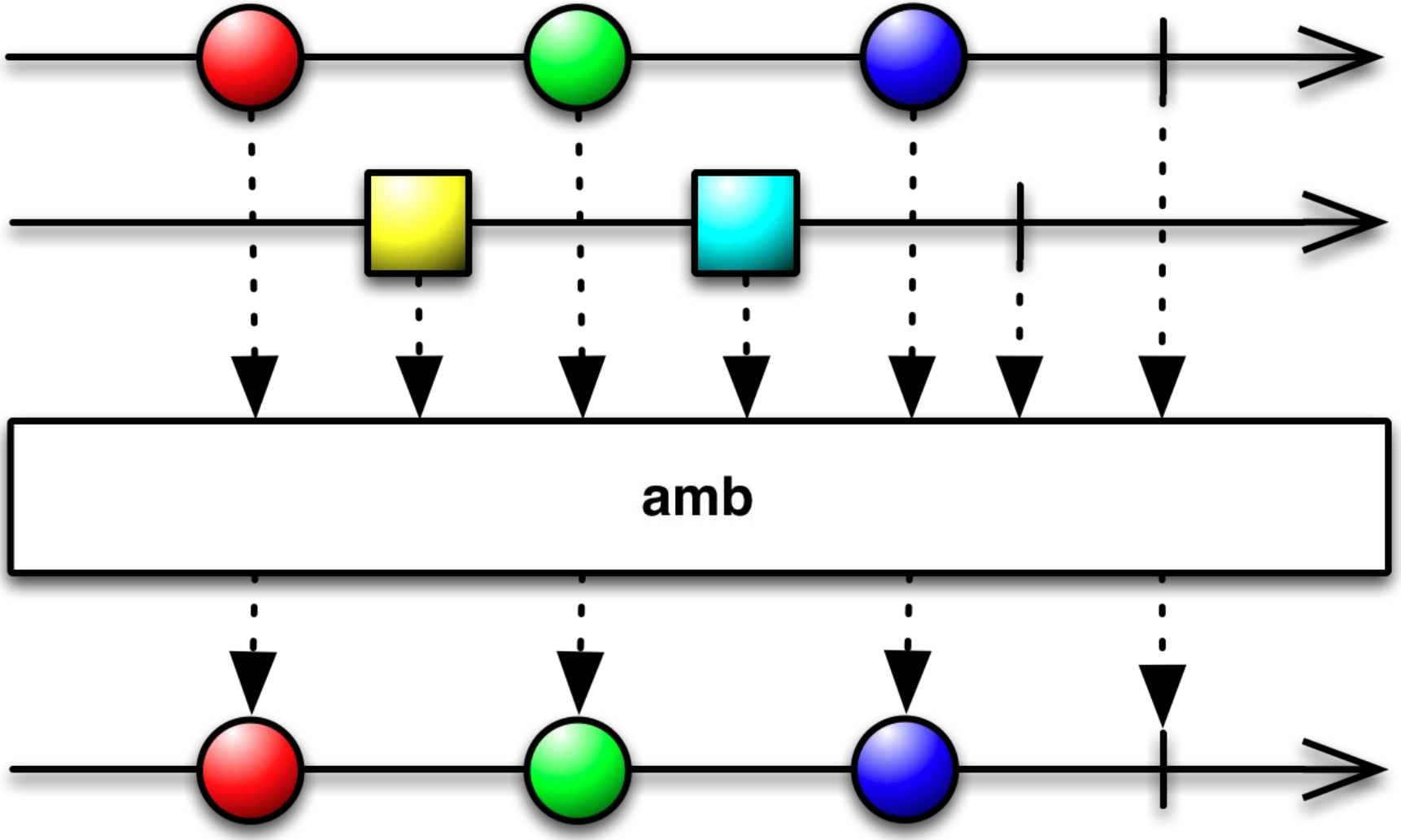

toMap(k( ))
[k( ): , k( ): , k( ):]


toMultimap(k( ))
[0:k( ):
, 1:k( ):
, 2:k( ):
]

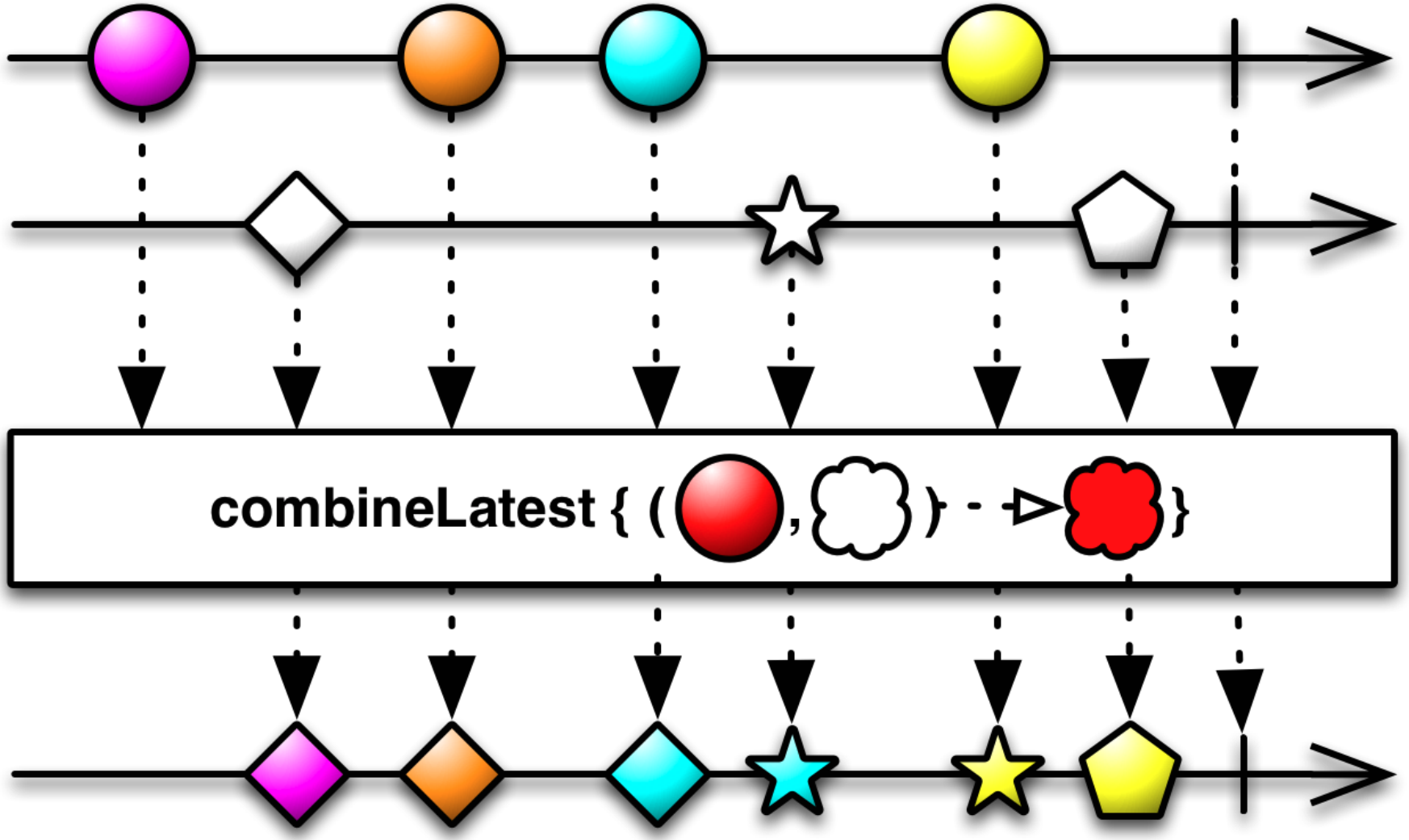

2
5
1
6
3
4
toSortedList
1
2
3
4
5
6


2
5
1
6
3
4
toSortedList({ ≅5? })
5
6
4
3
2
1

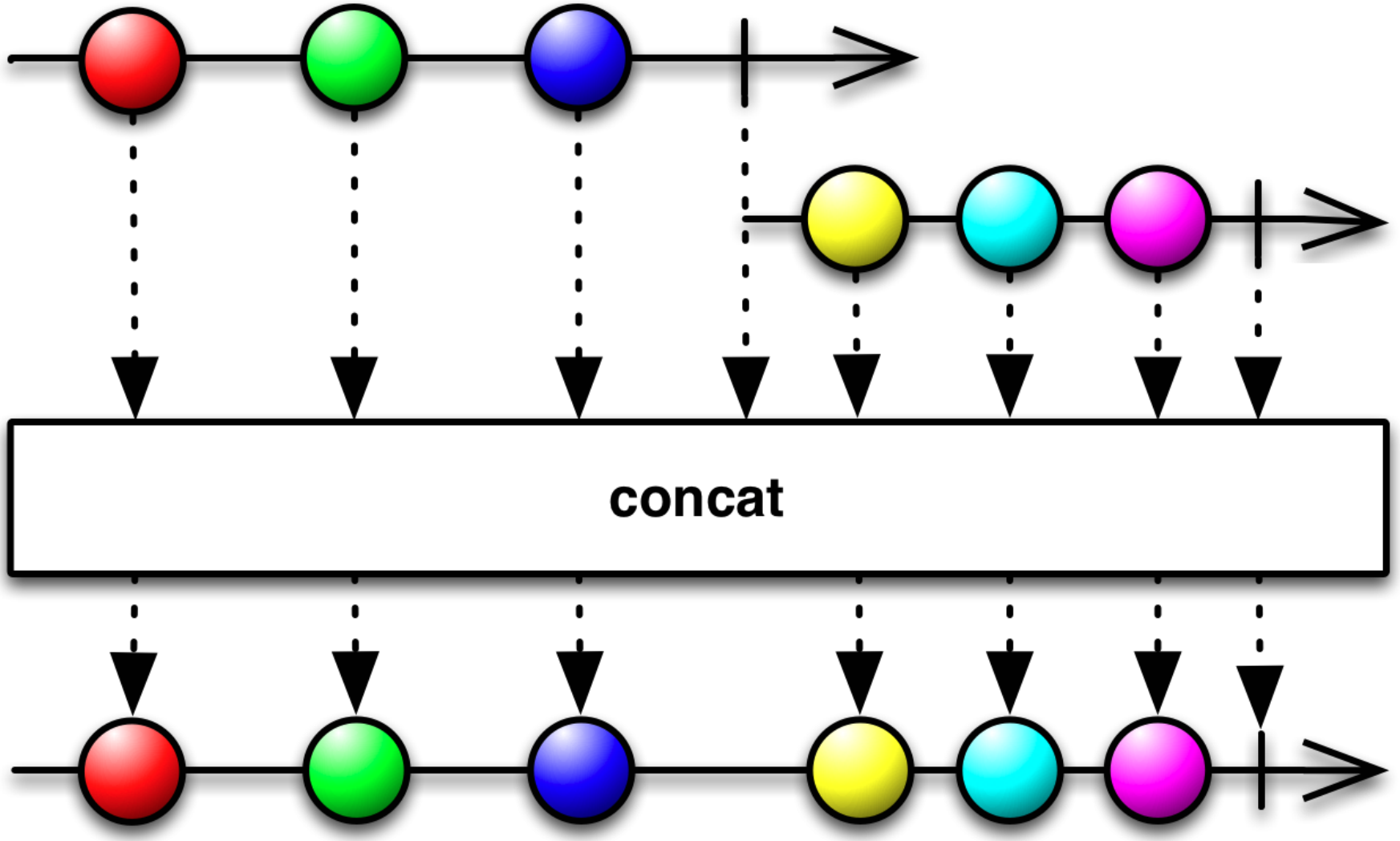
Combinaison

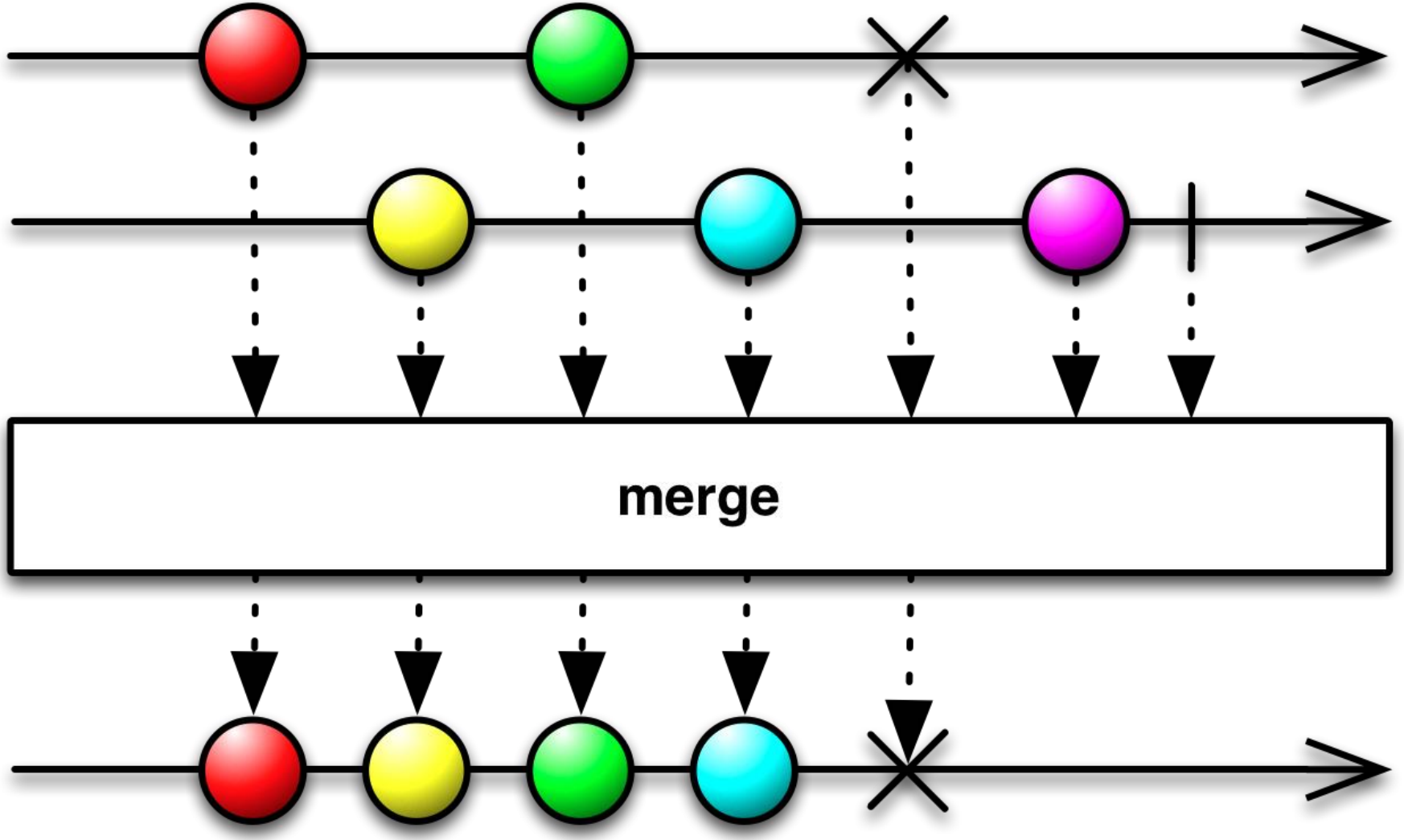


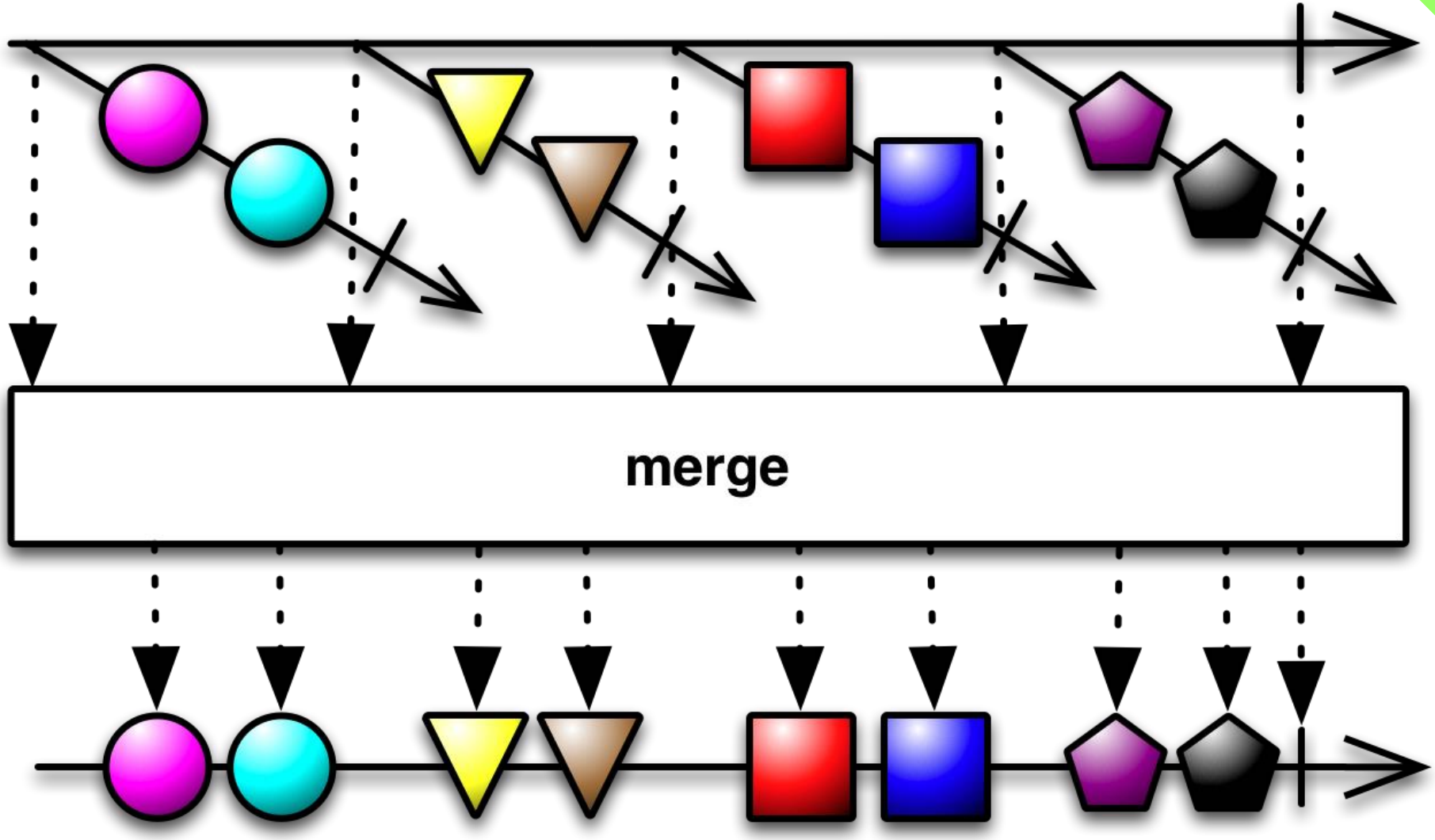
- Cas d'usage : Utiliser le premier qui répond, et annuler les autres.

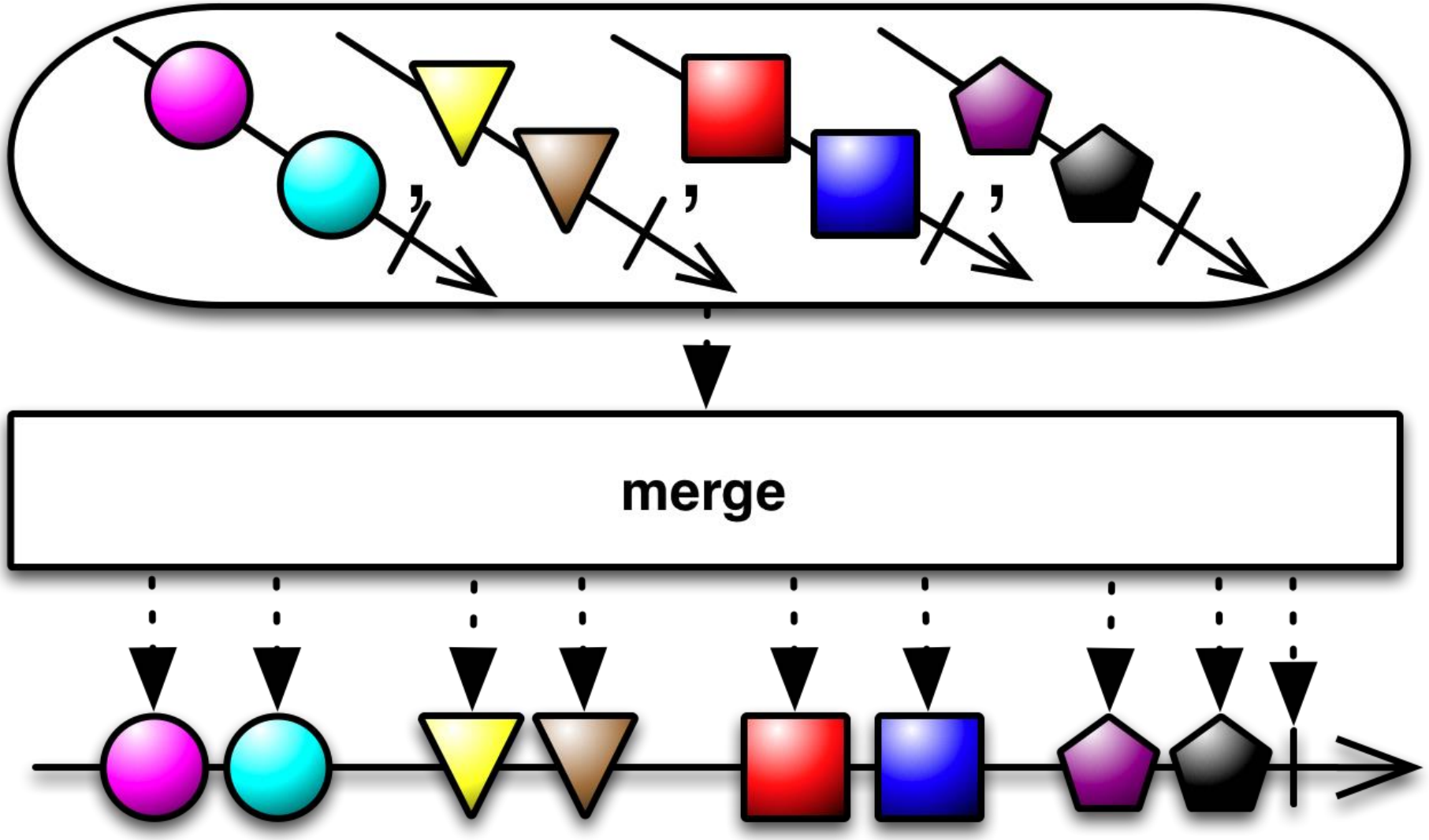


- Cas d'usage : Appliquer une fonction à chaque changement de valeur.

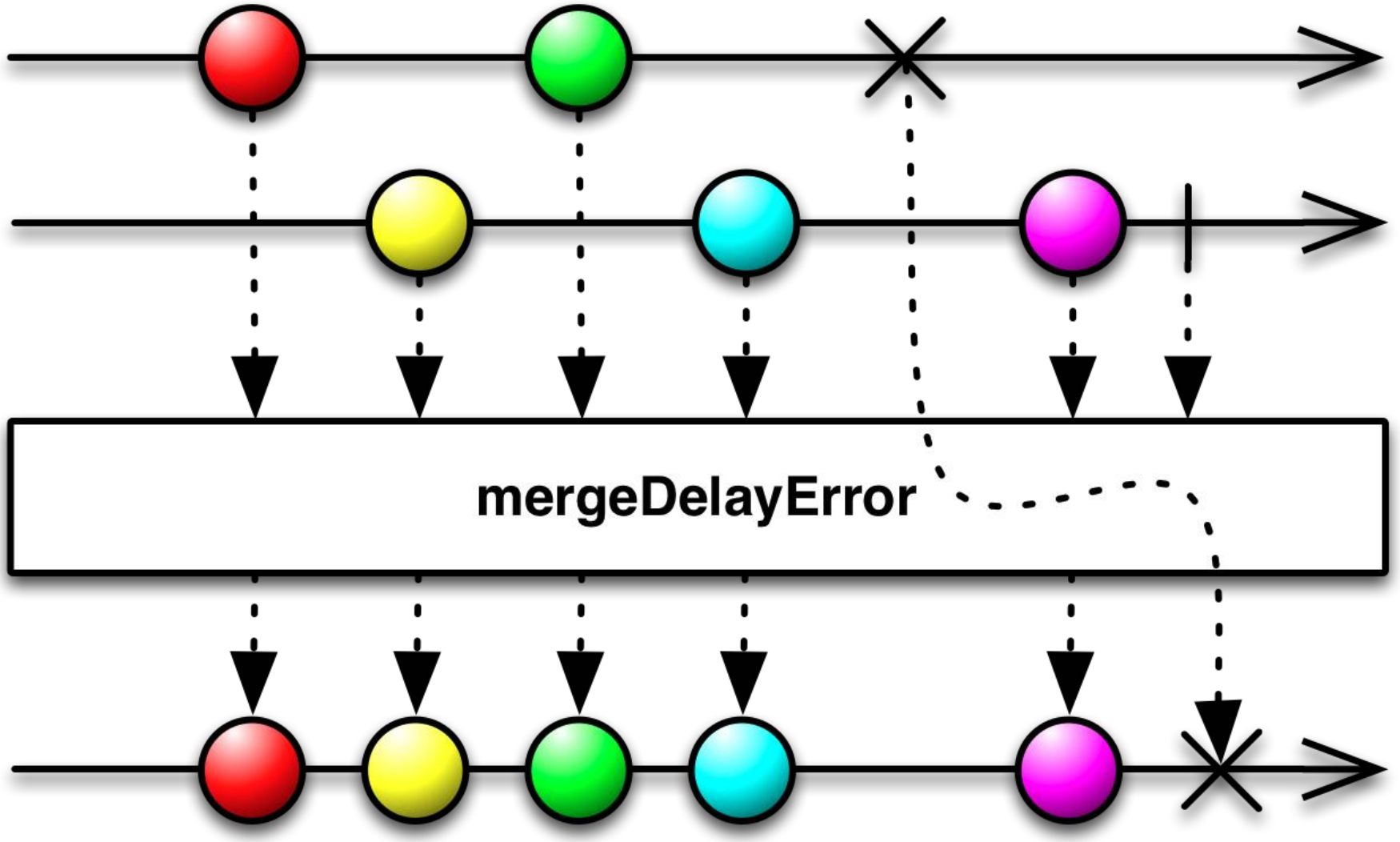

concat

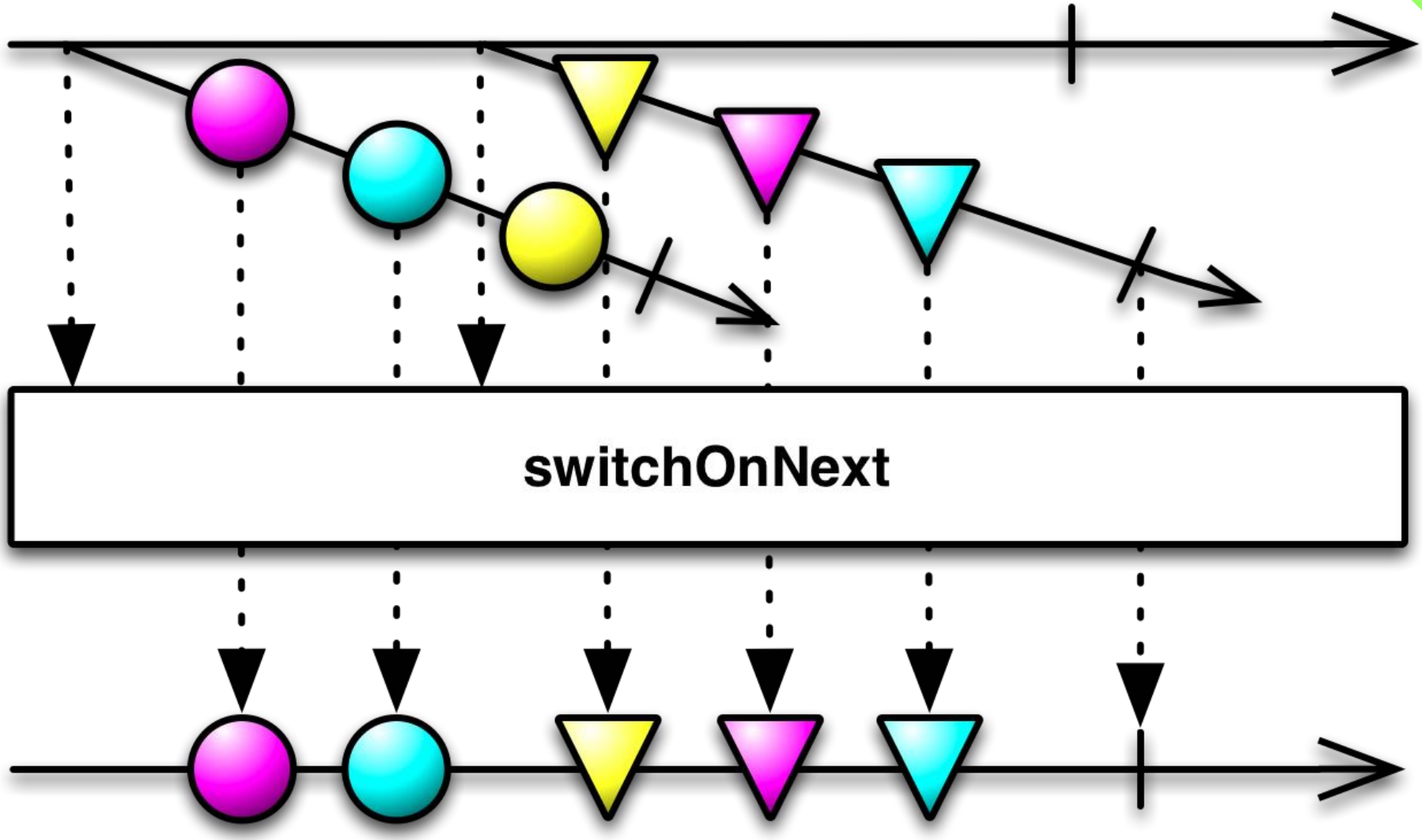

merge

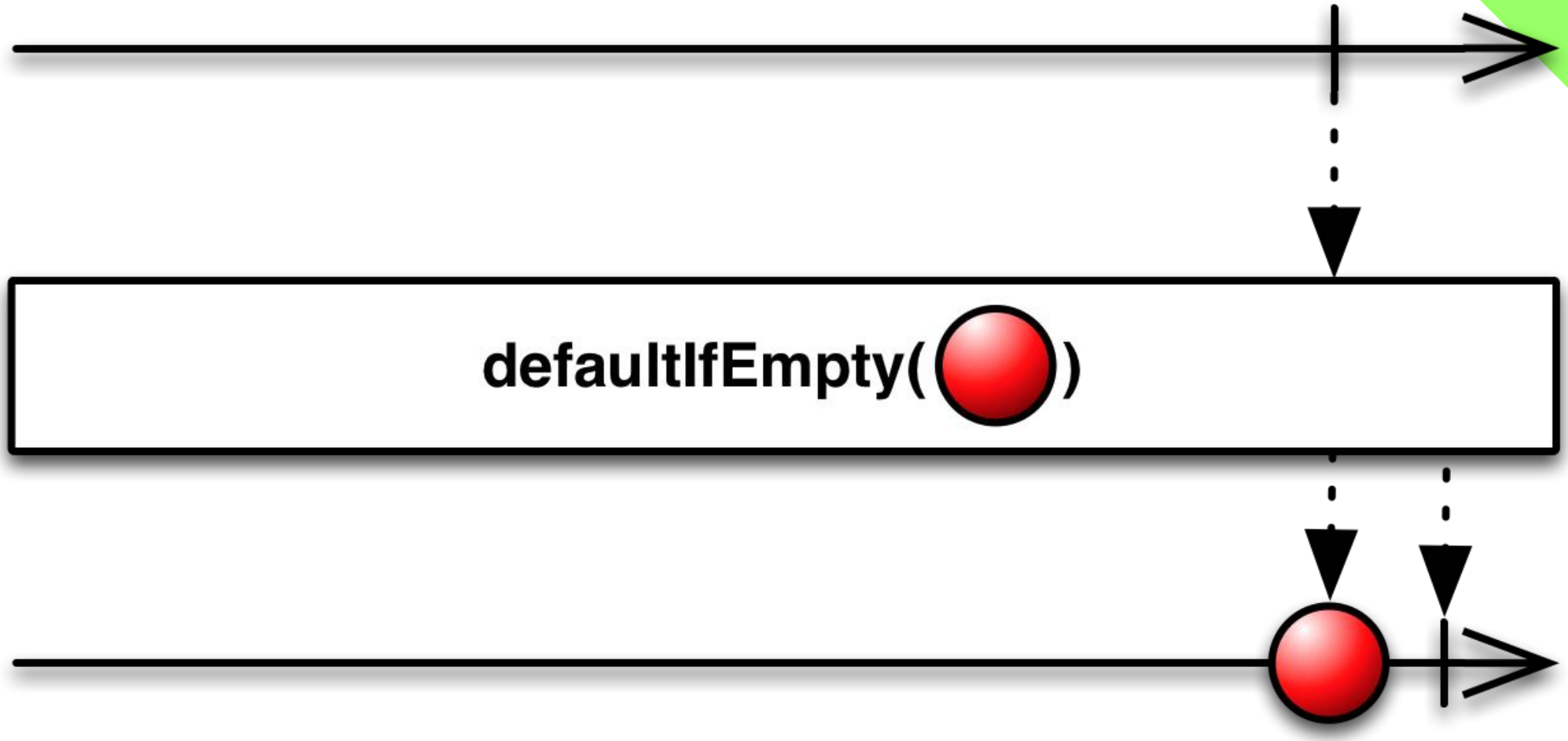

merge

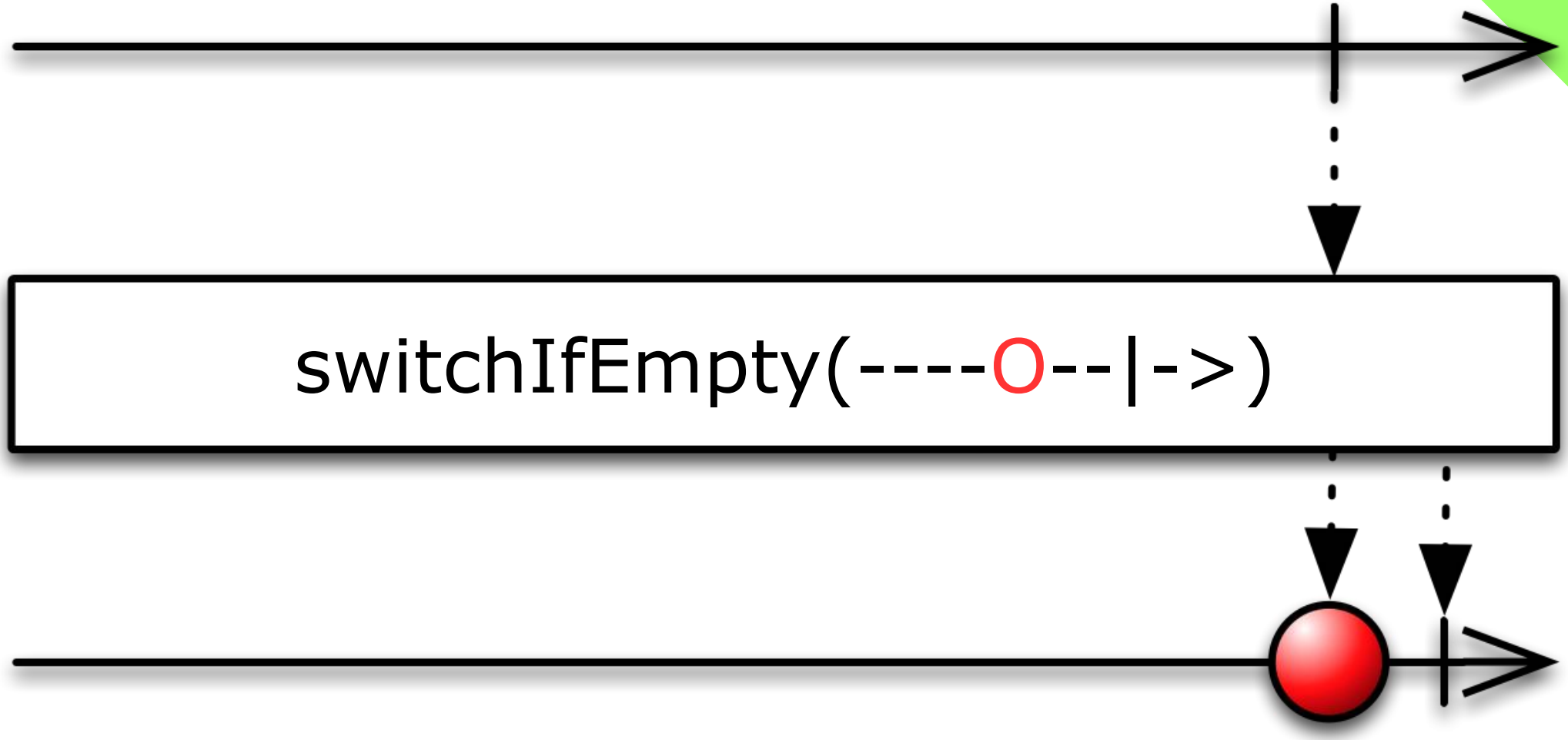


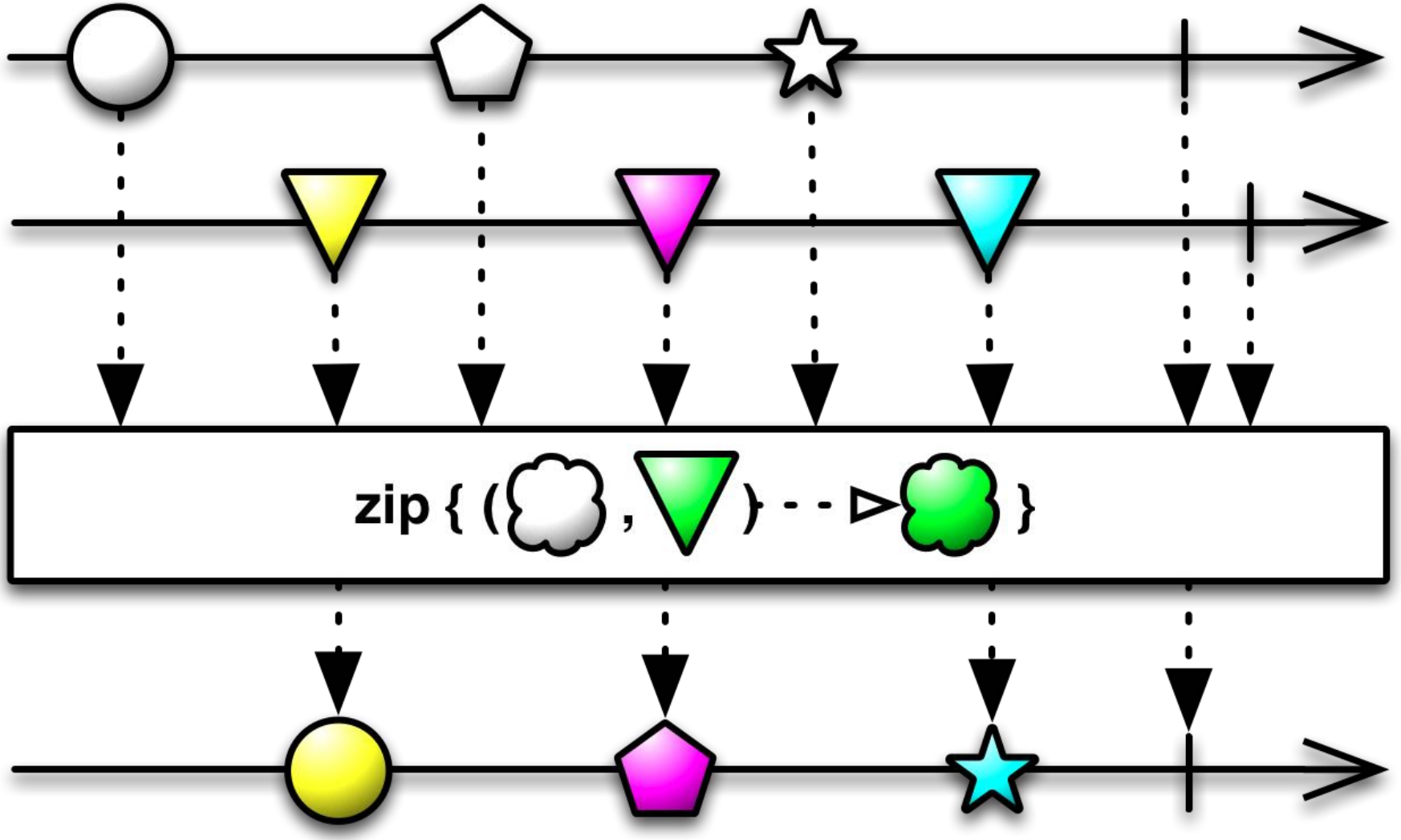
- merge(Observable<T>[ ] sequences)


mergeDelayError

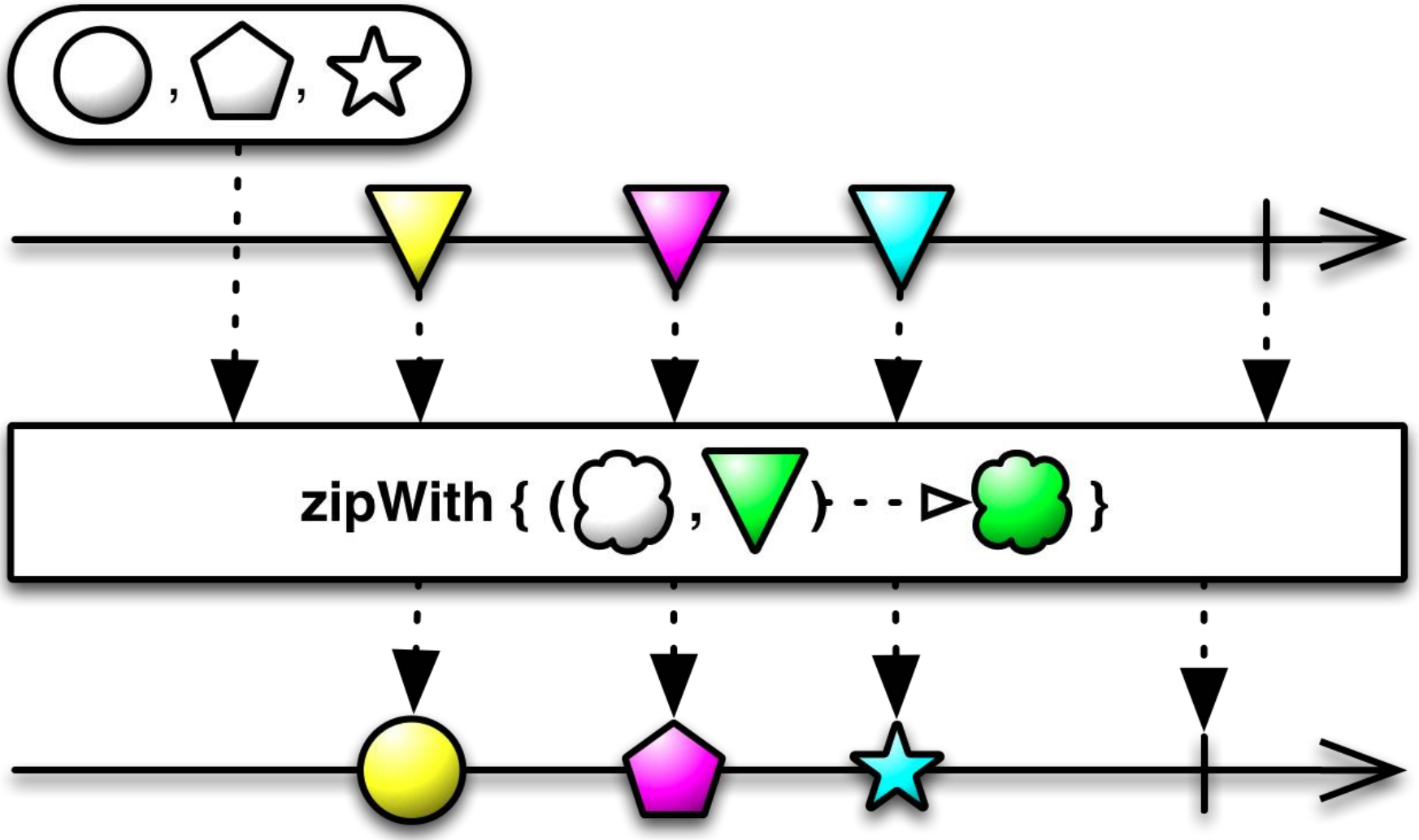

switchOnNext


defaultIfEmpty( )

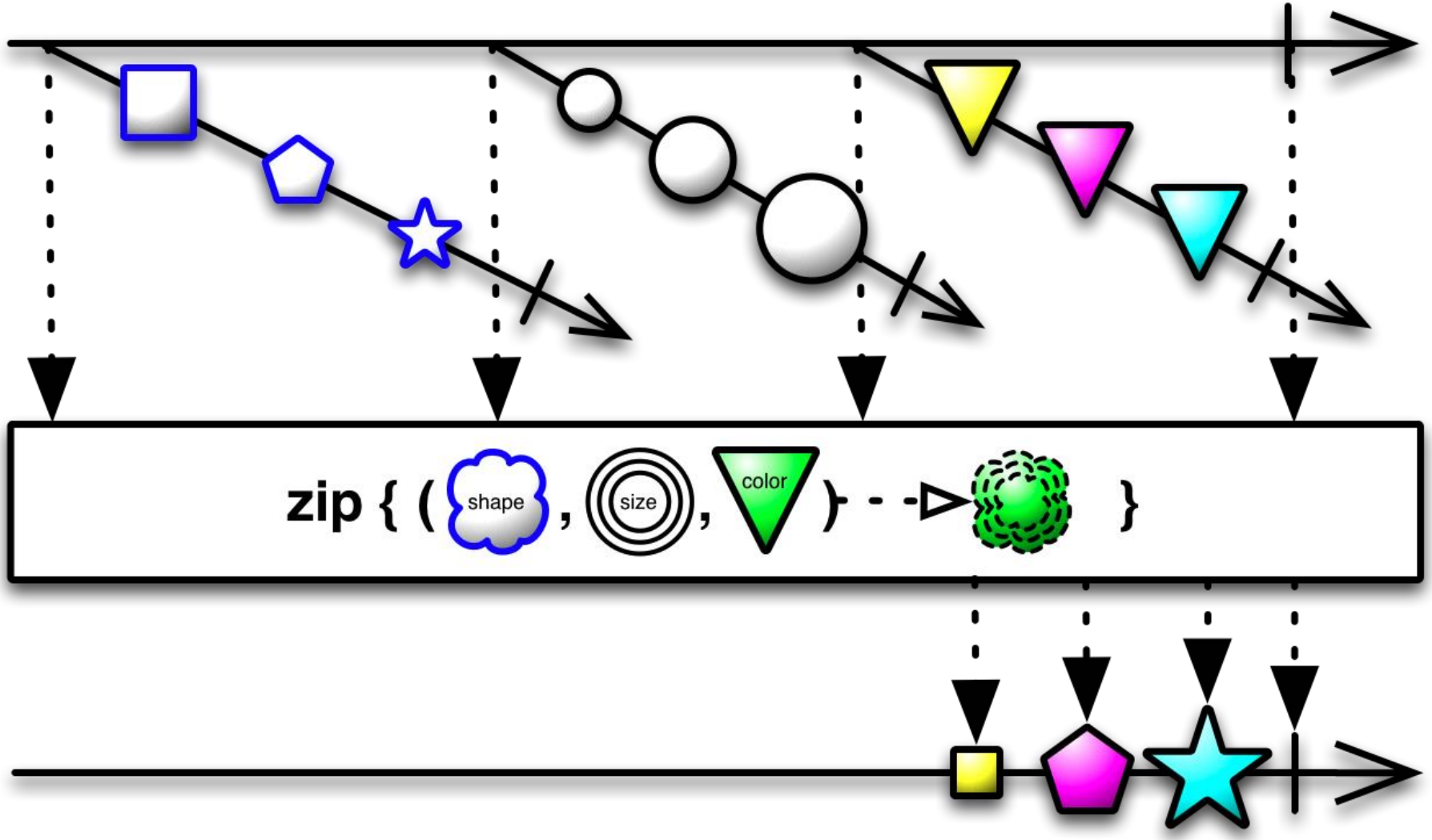

switchIfEmpty(----O--|->)



- Cas d'usage : Appaire les valeurs. Utile aussi pour faire un point de rendez-vous.

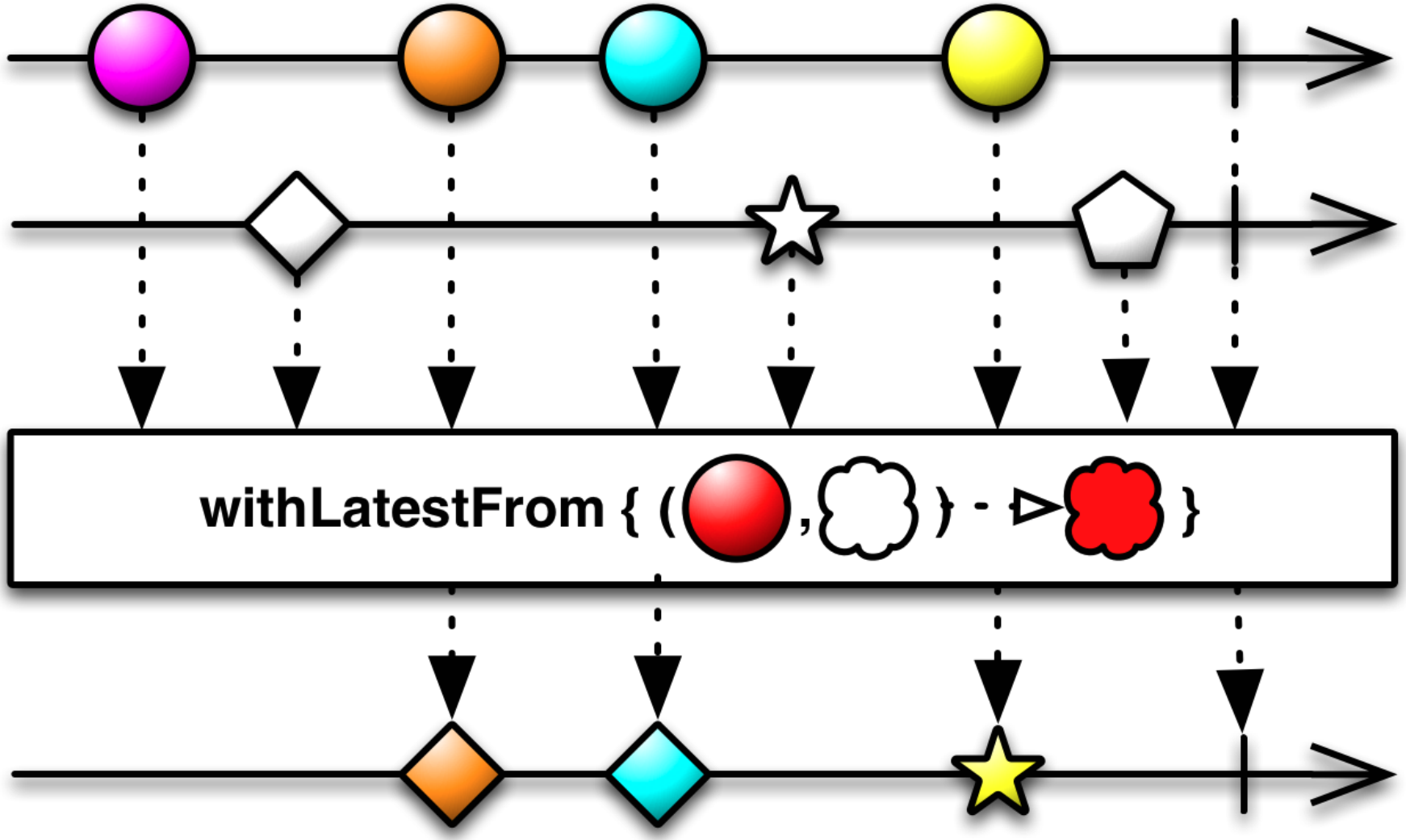


.zipWith(Iterable<T2> other, Func2<T, T2, R> zipFunction)

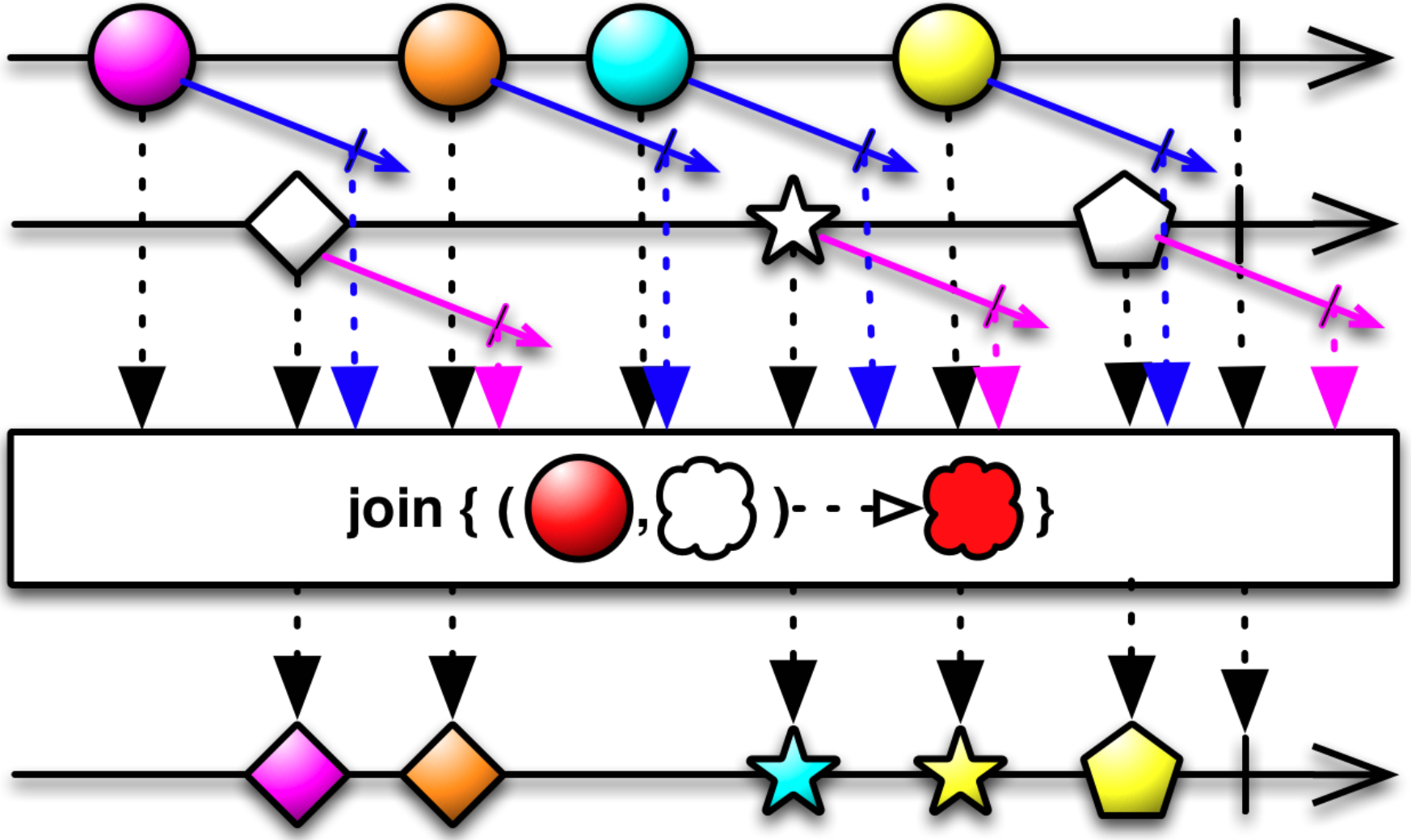


.Observable<R> zip(Observable<Observable<?>> ws, FuncN<R> zipFunction)

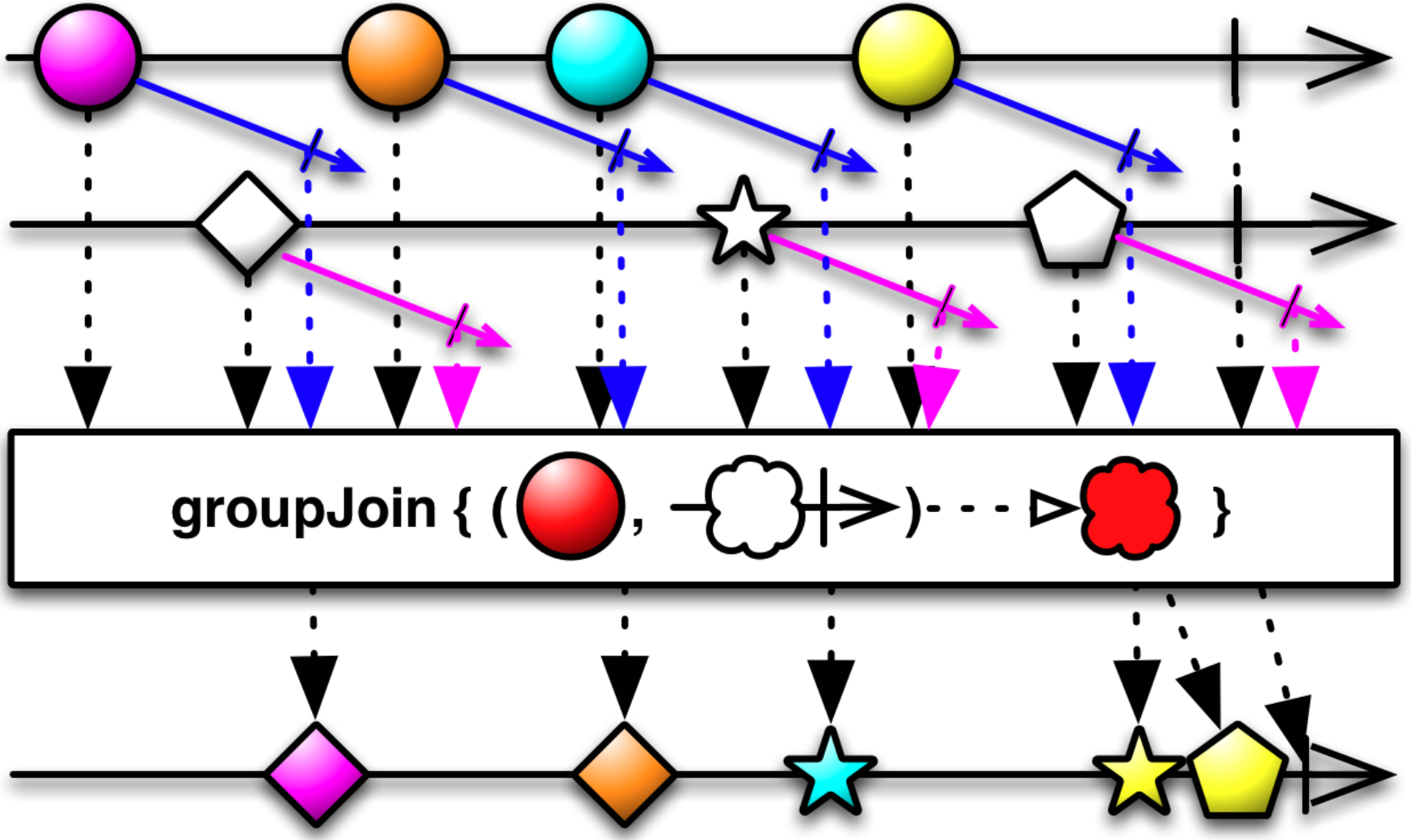
./!\ La zipFunction est appelée au **onCompleted** de l'Observable principal.



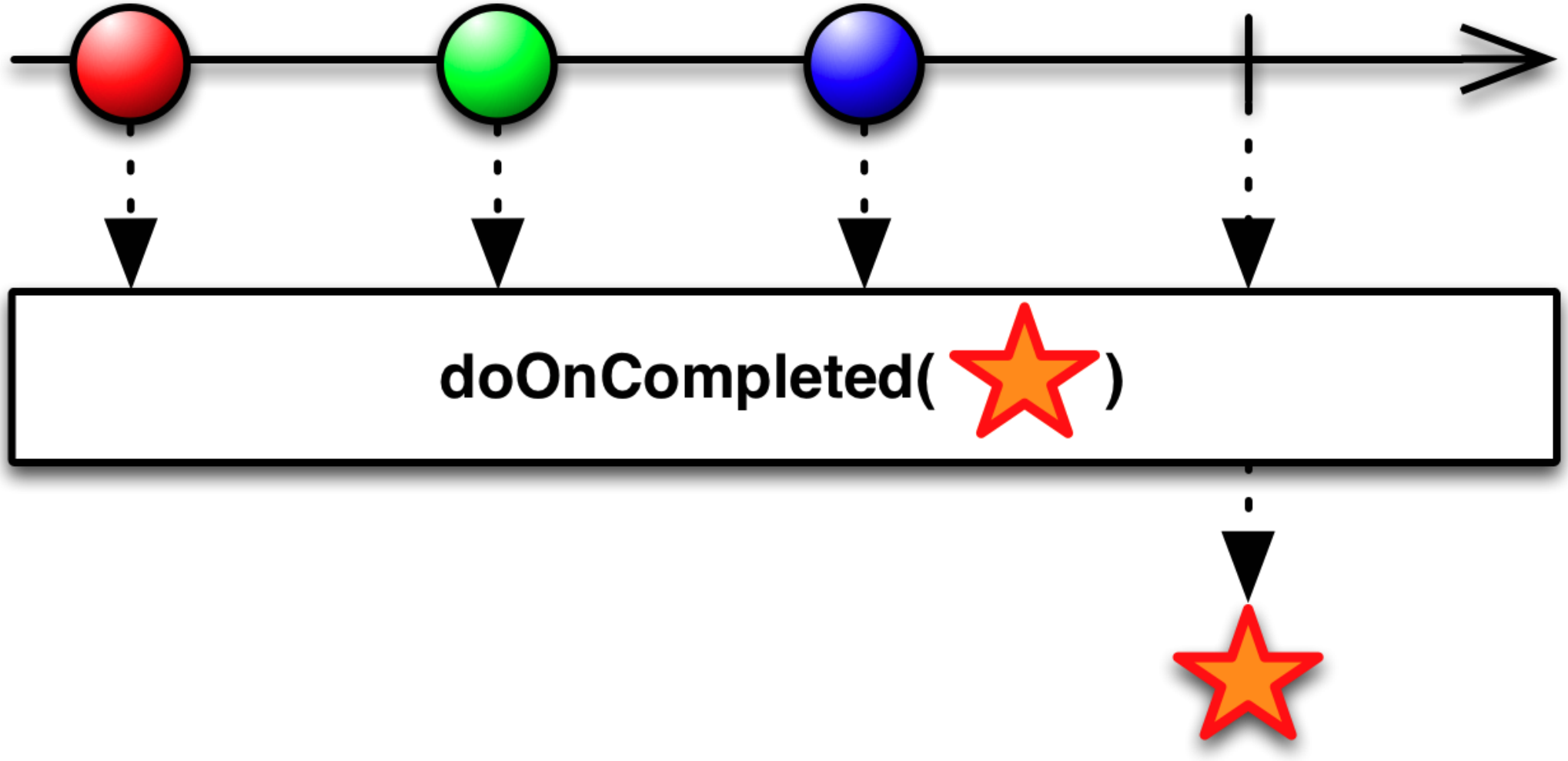
- Cas d'usage : Une sorte de map(...) pouvant utiliser la dernière valeur d'un autre flux.

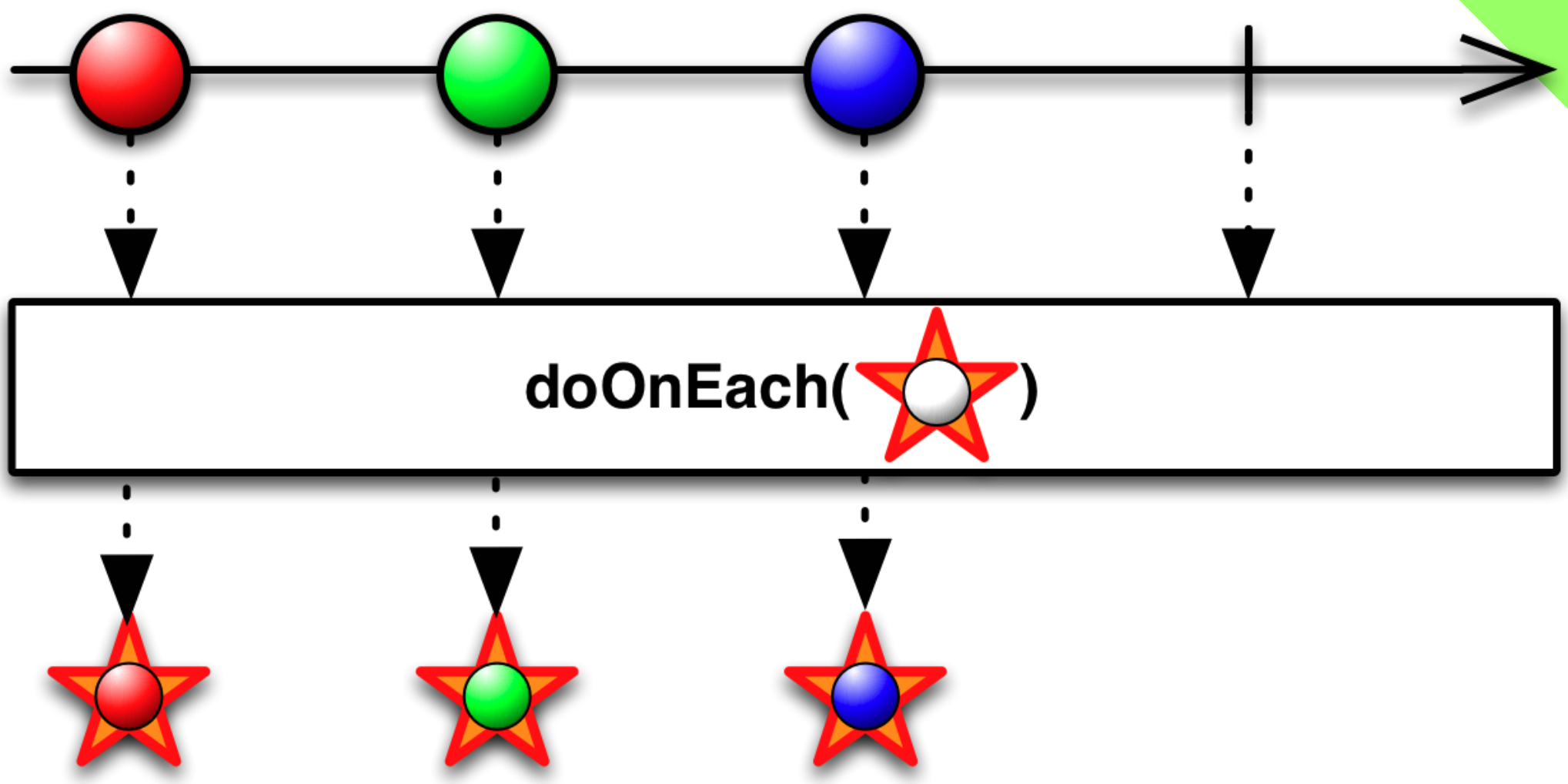


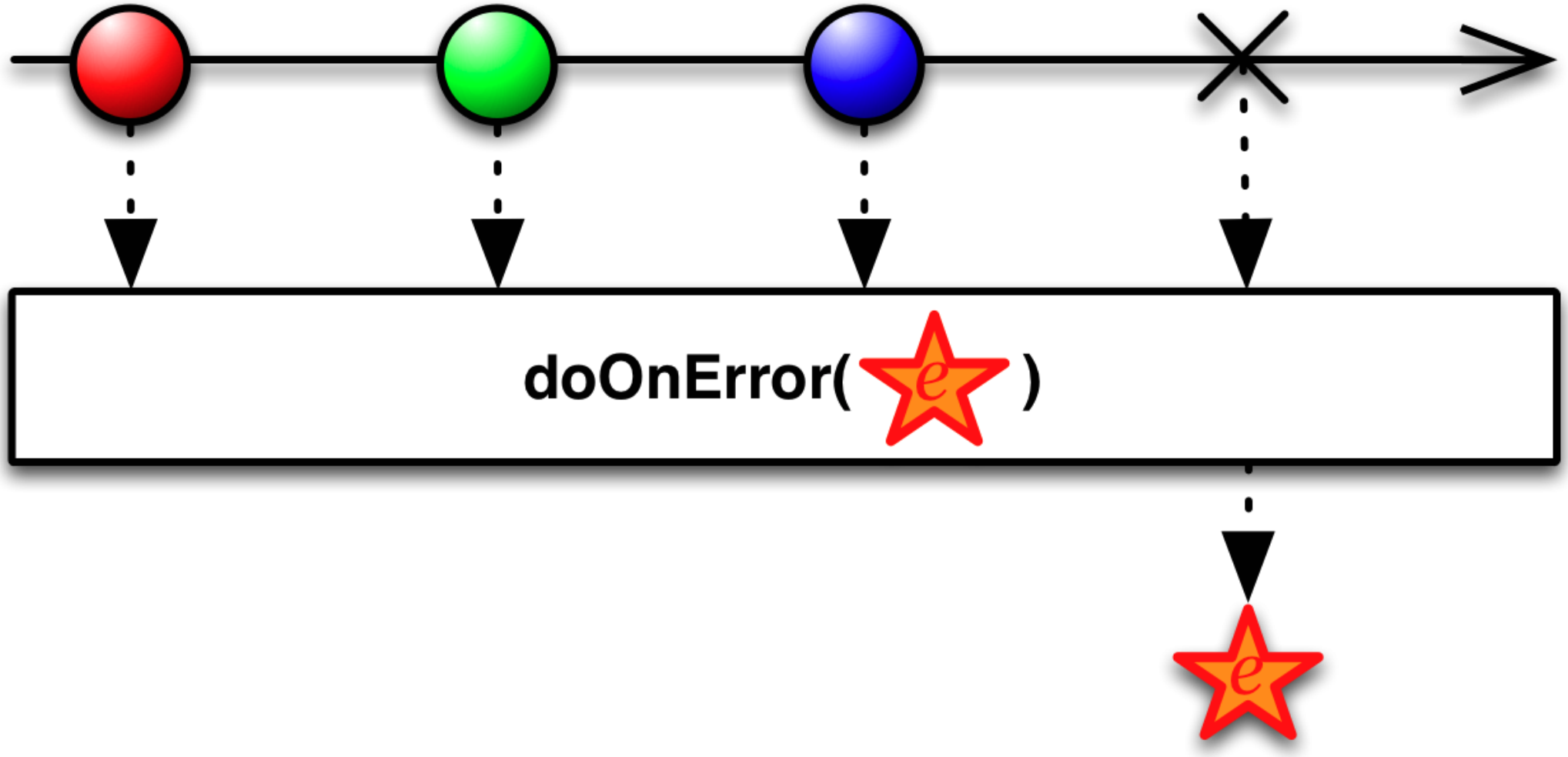
..join(...) combine les entrées lorsque les fenêtres temporelles de chaque entrée se superposent.

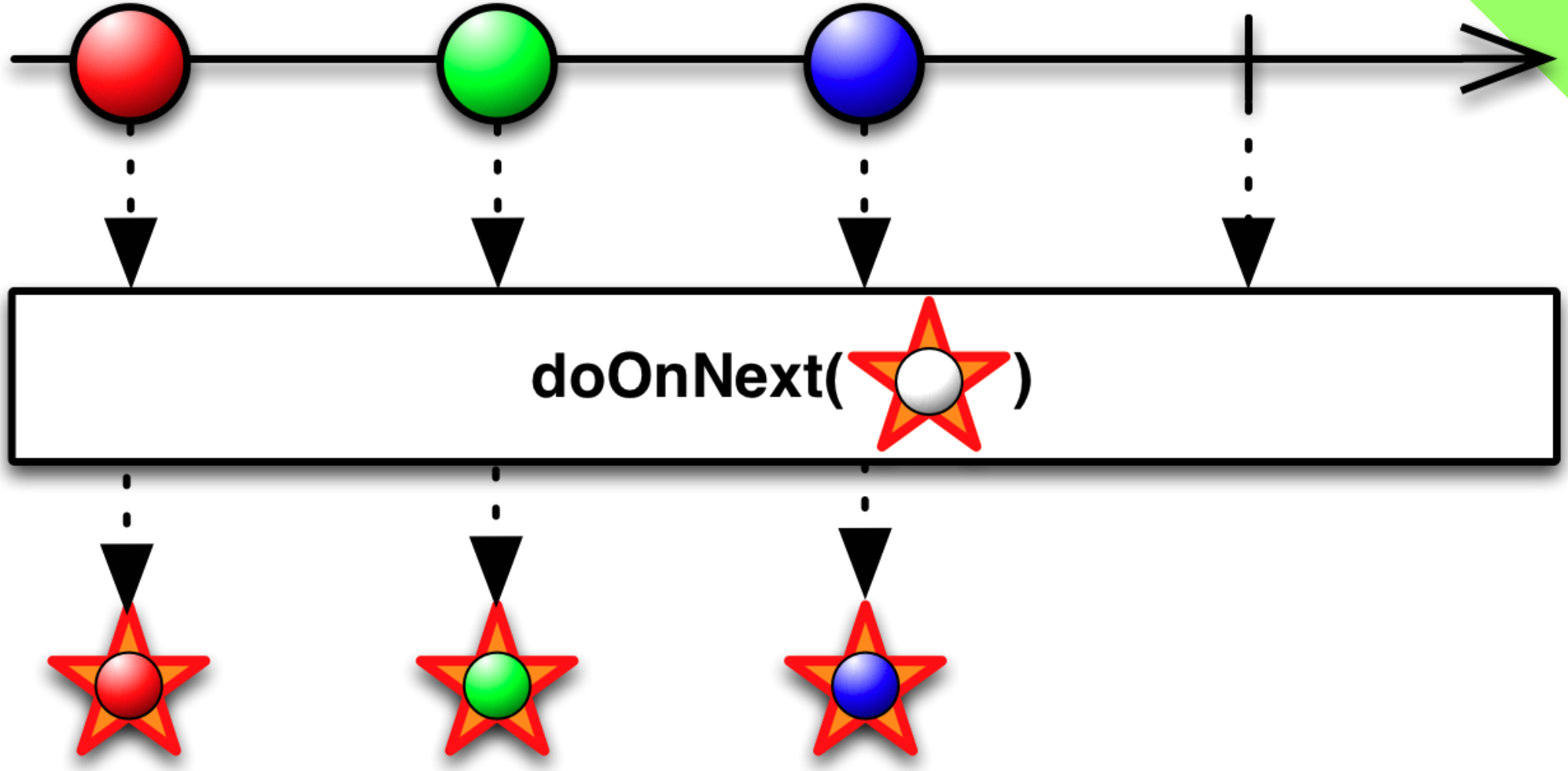

groupJoin { (,)- - - }

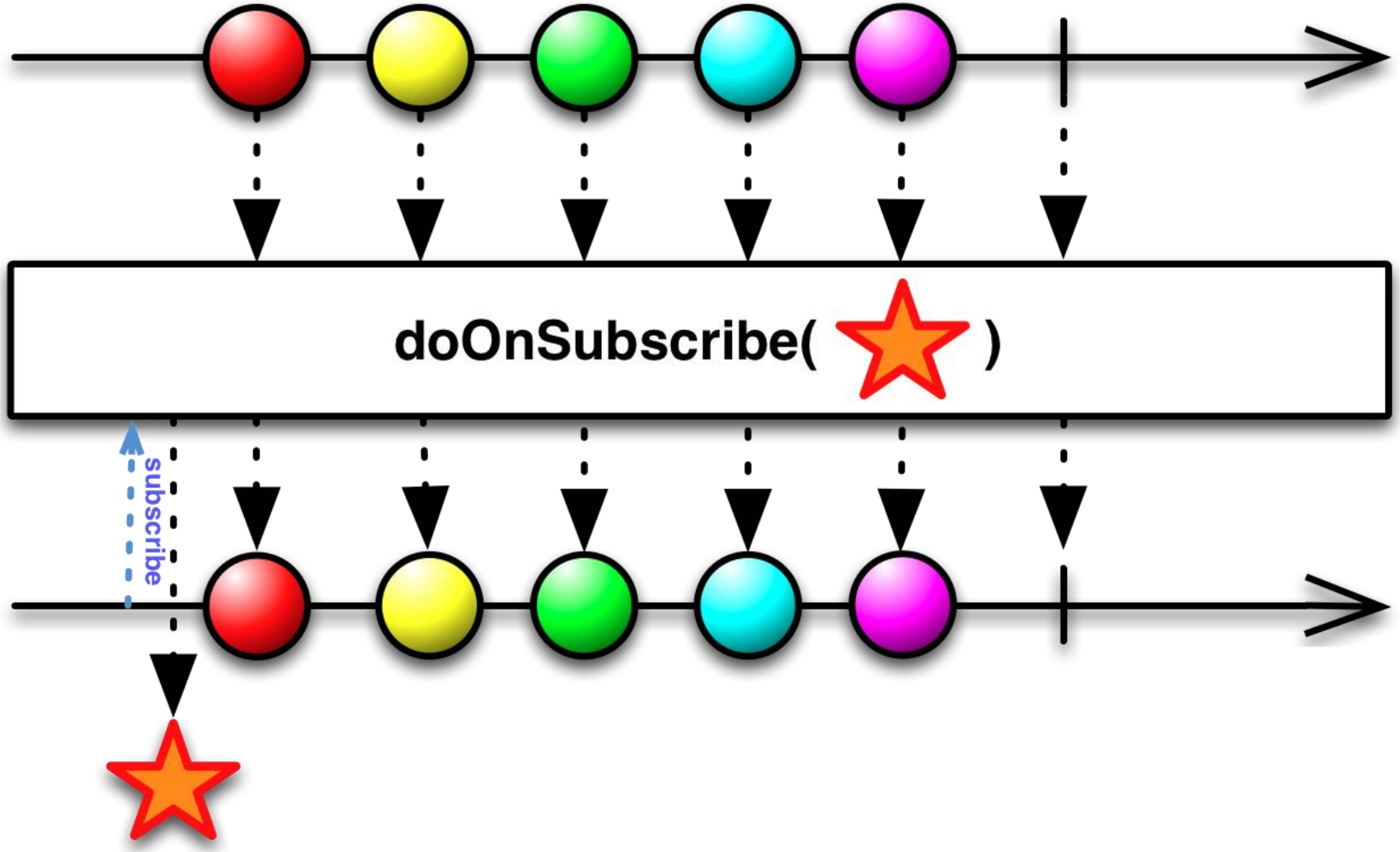
Effet de bord

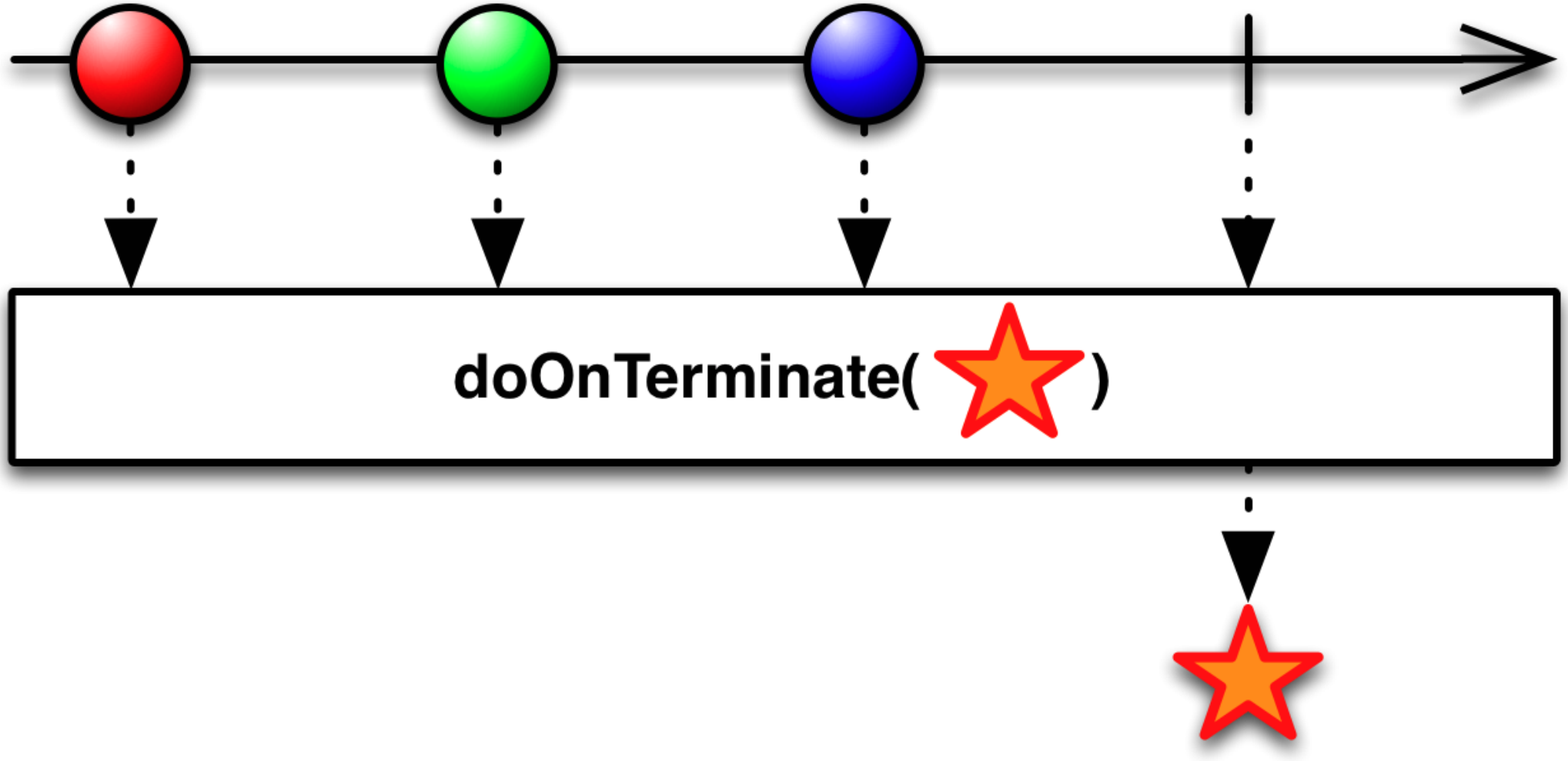

doOnCompleted( )

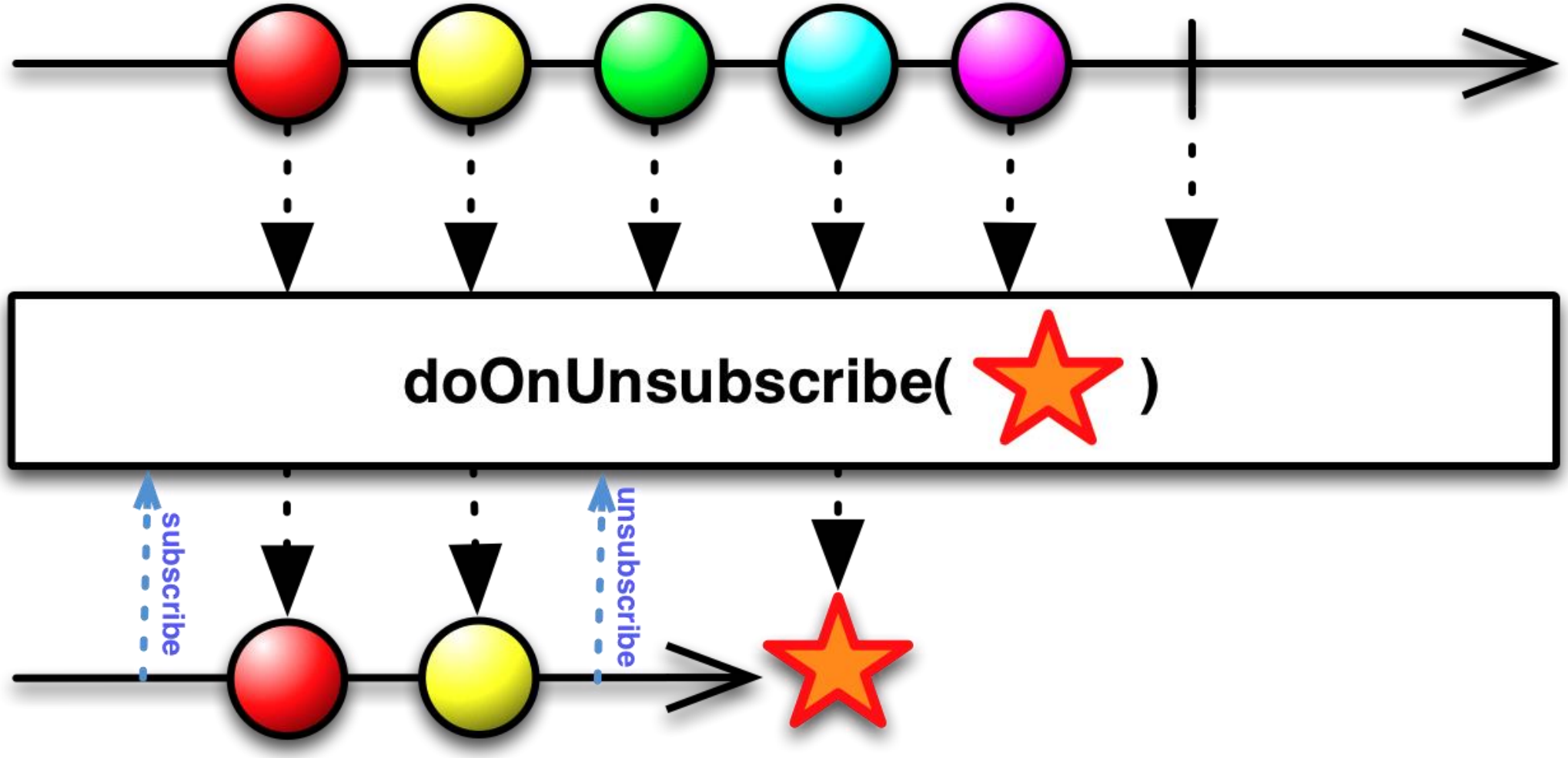

doOnEach(

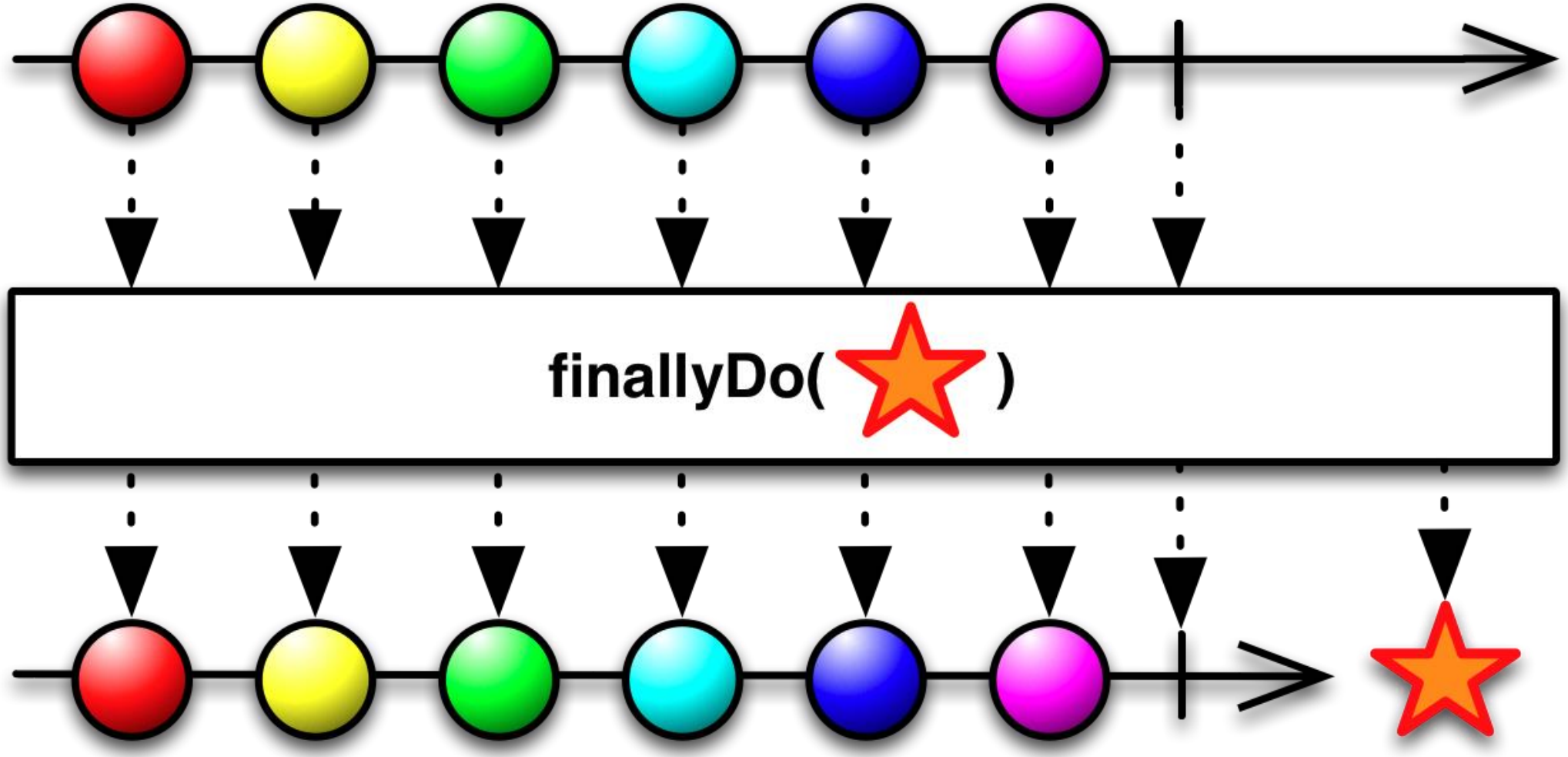

doOnError(
e
)
e


doOnNext( )

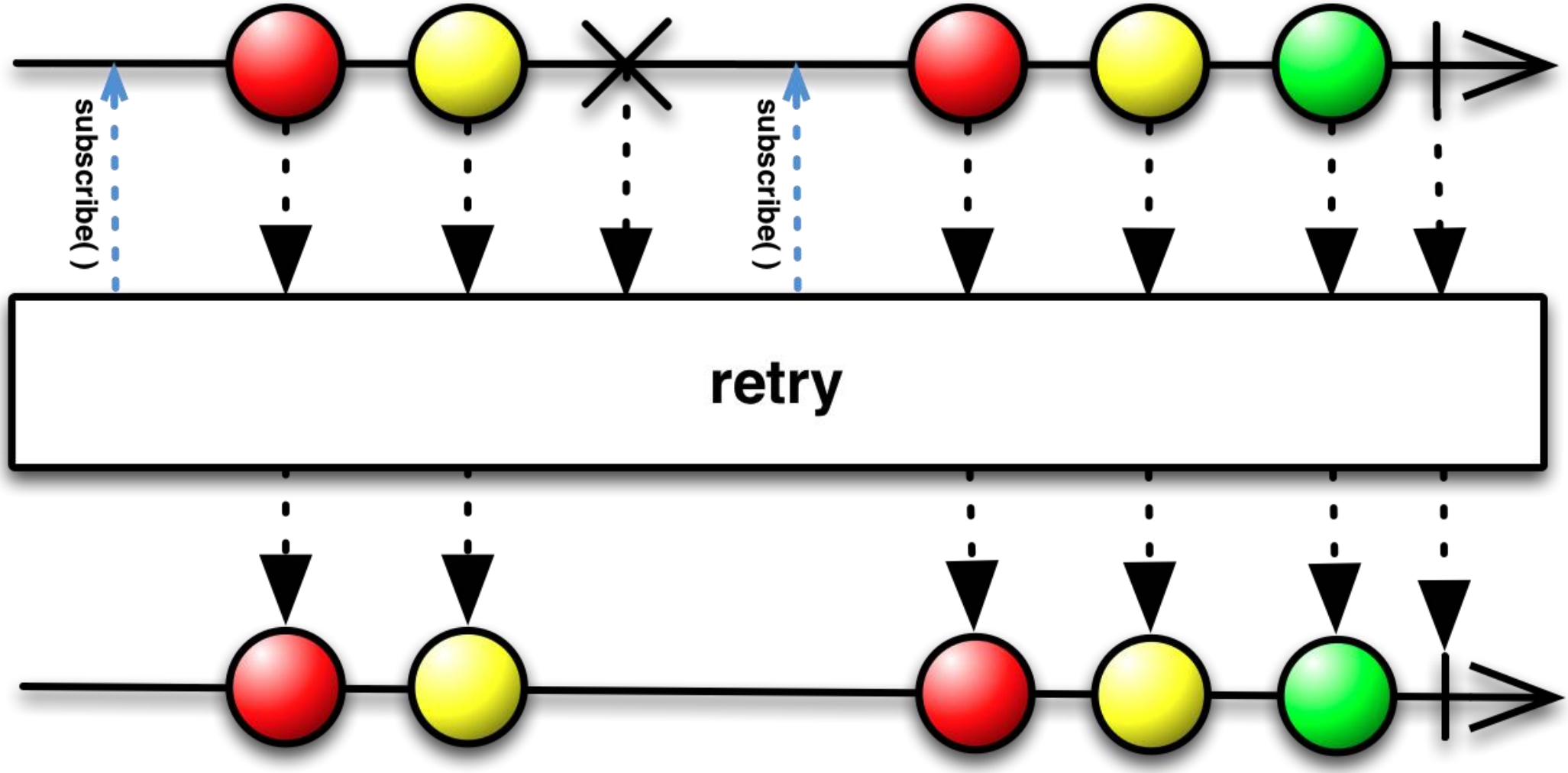

doOnSubscribe( )
subscribe

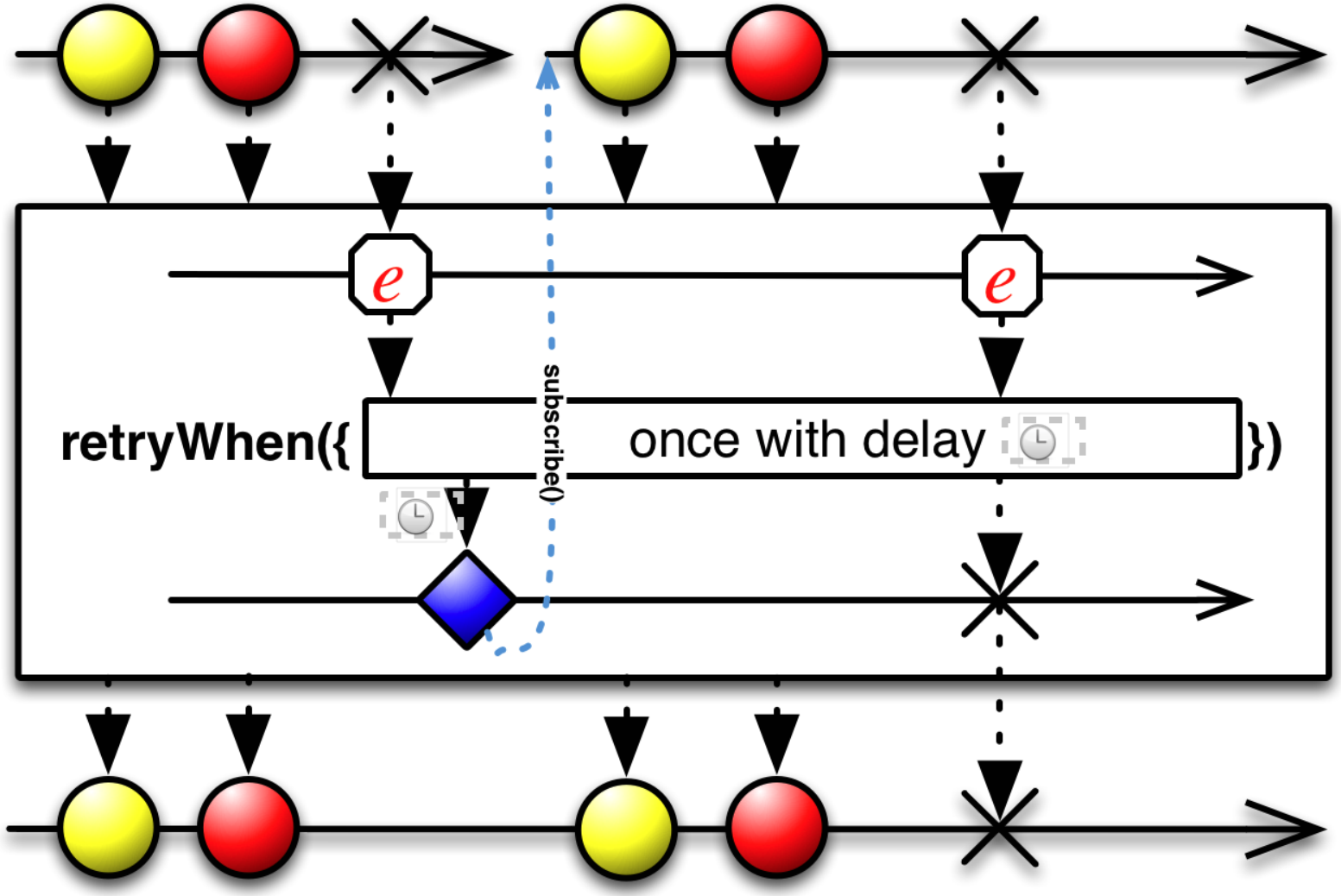

doOnTerminate( )


doOnUnsubscribe( )
subscribe
unsubscribe

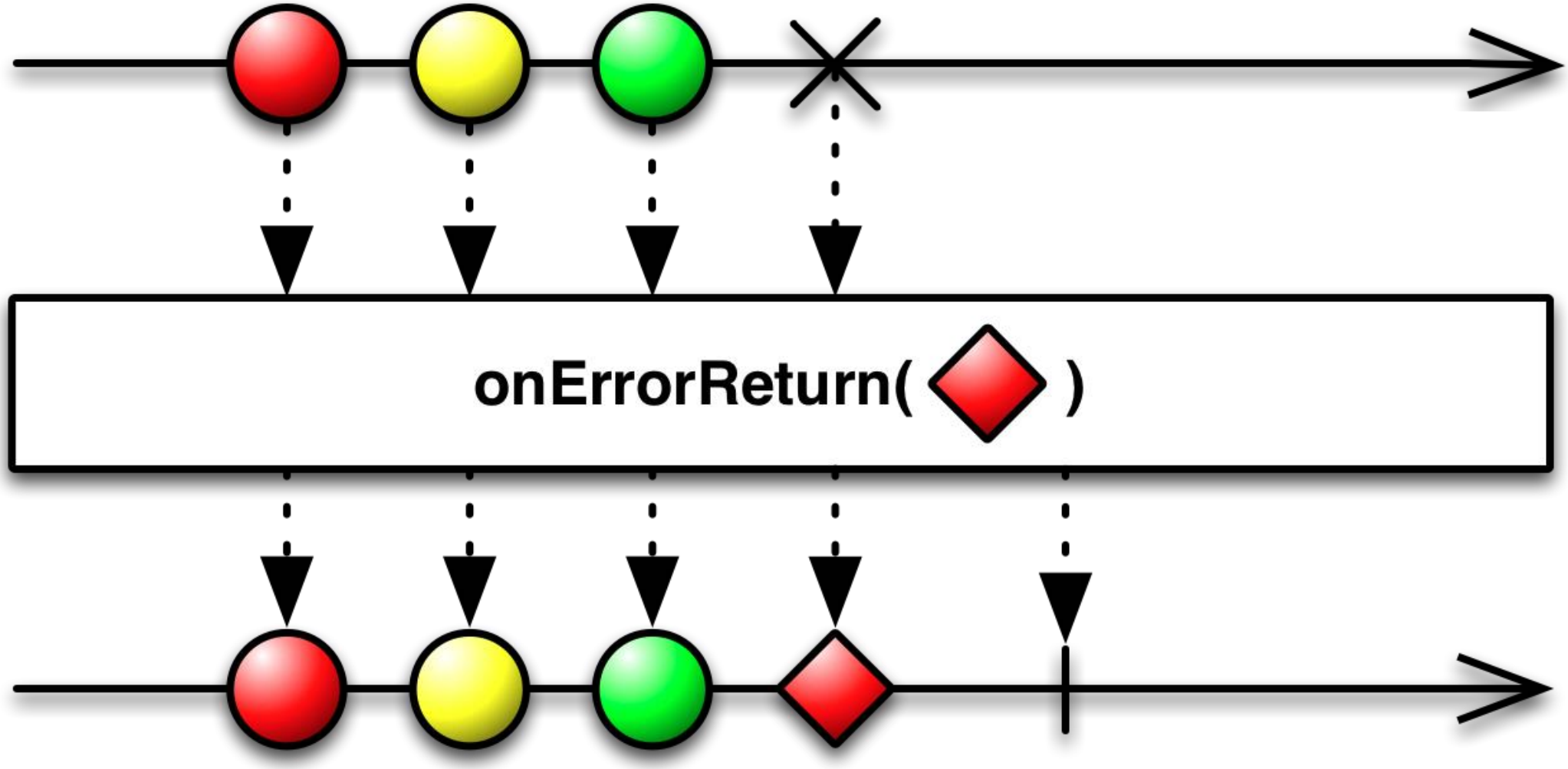

finallyDo( )

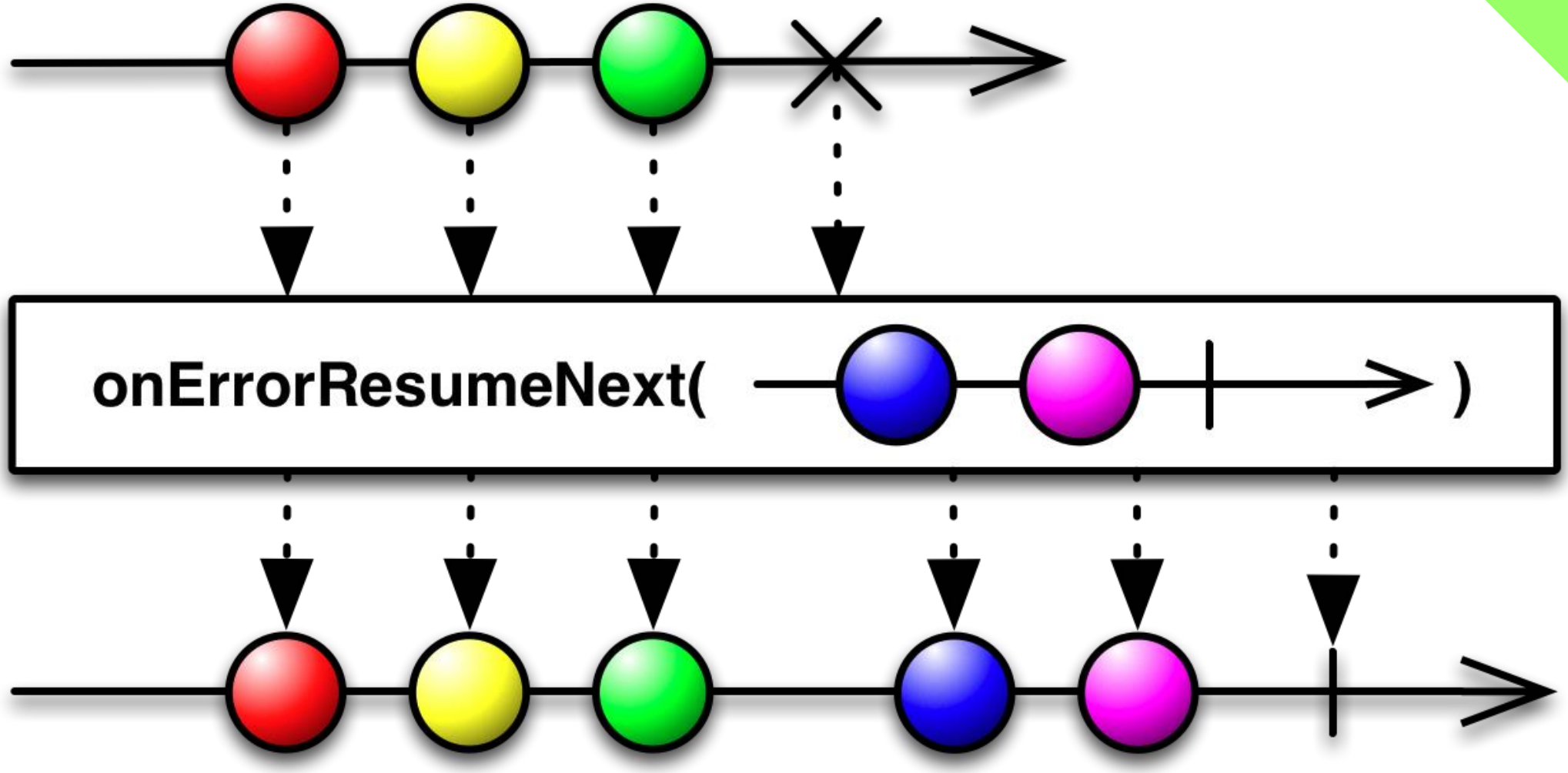
ERROR


subscribe()
subscribe()
retry

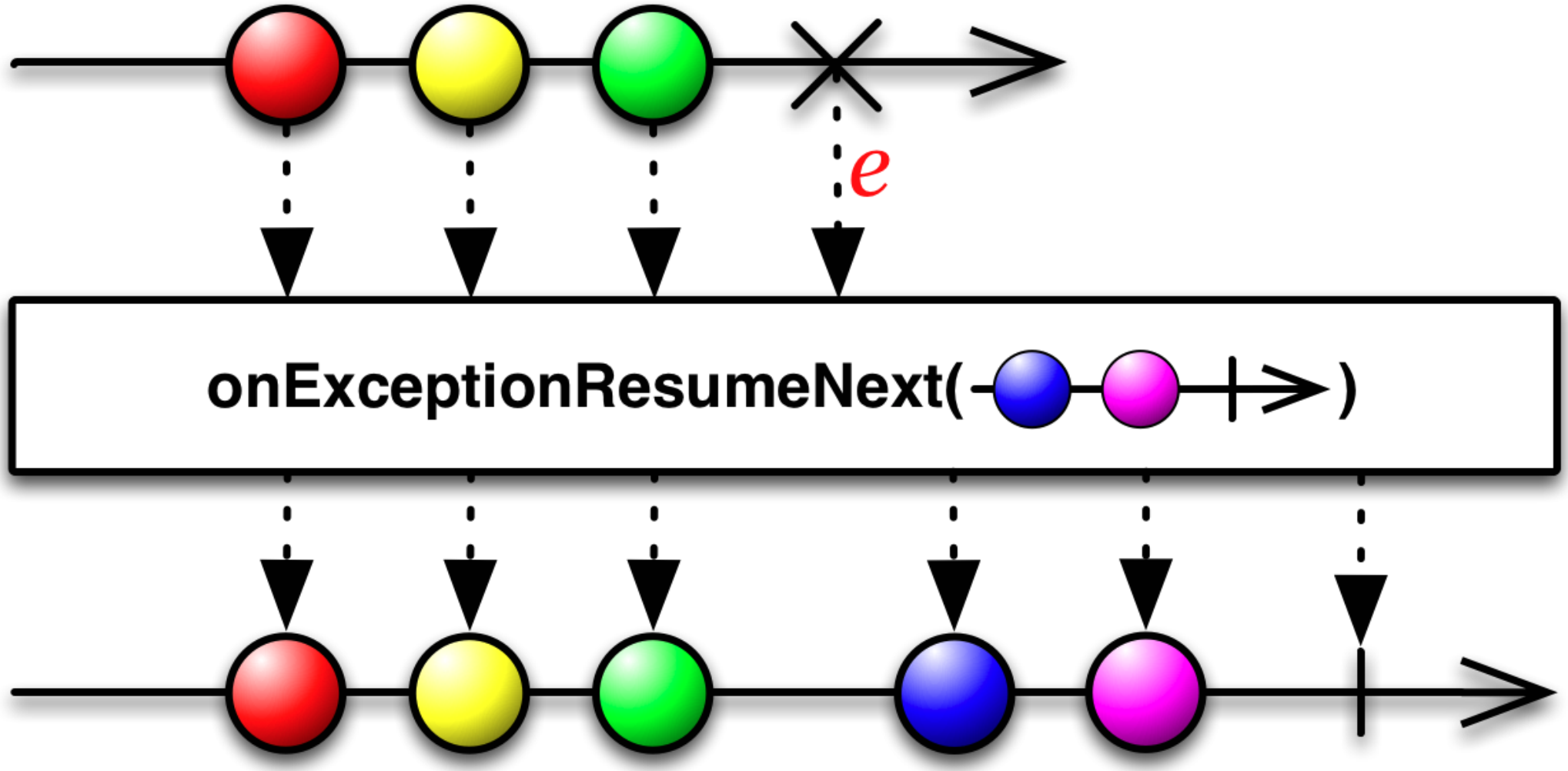


- Cas d'usage : Retentative en cas d'erreur avec une logique pour lancer la reprise.


onErrorReturn( )

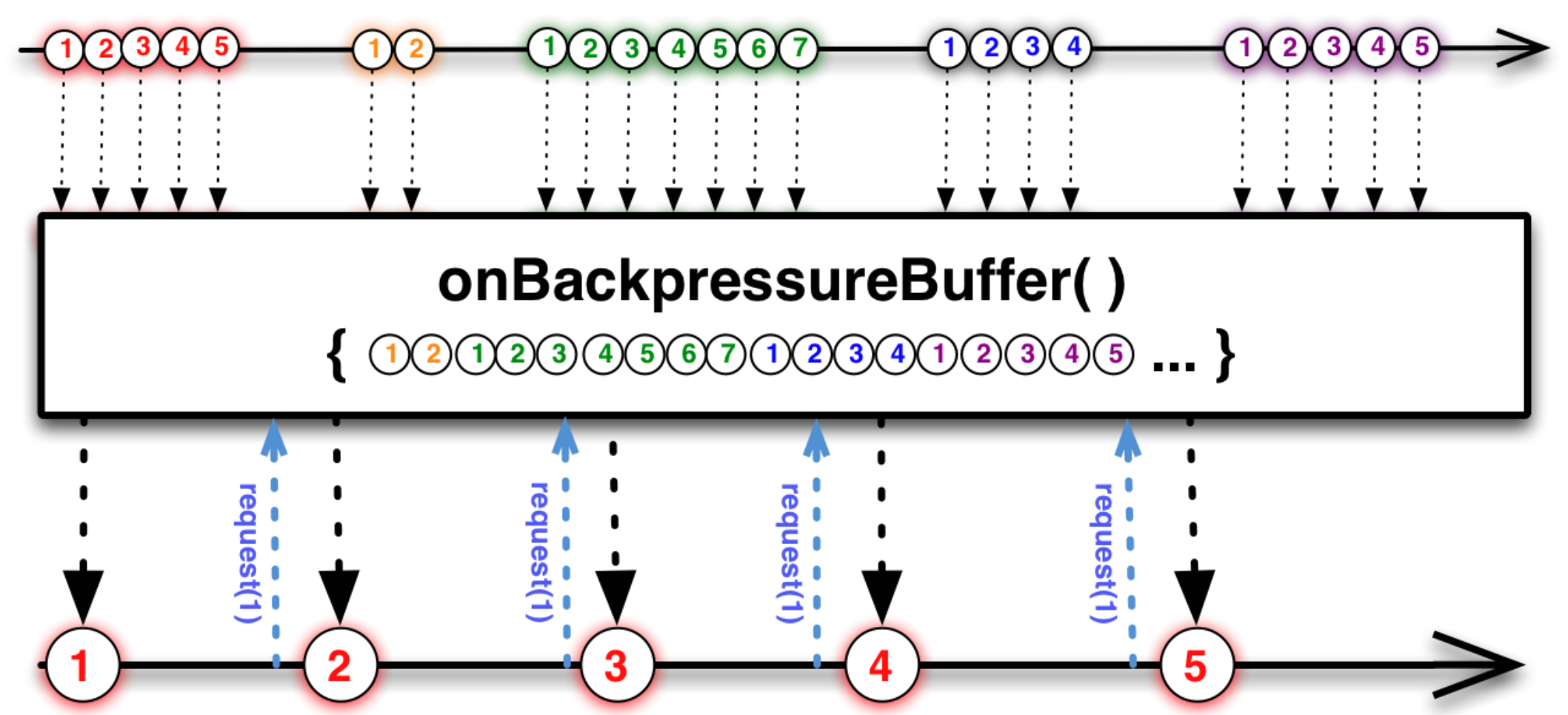


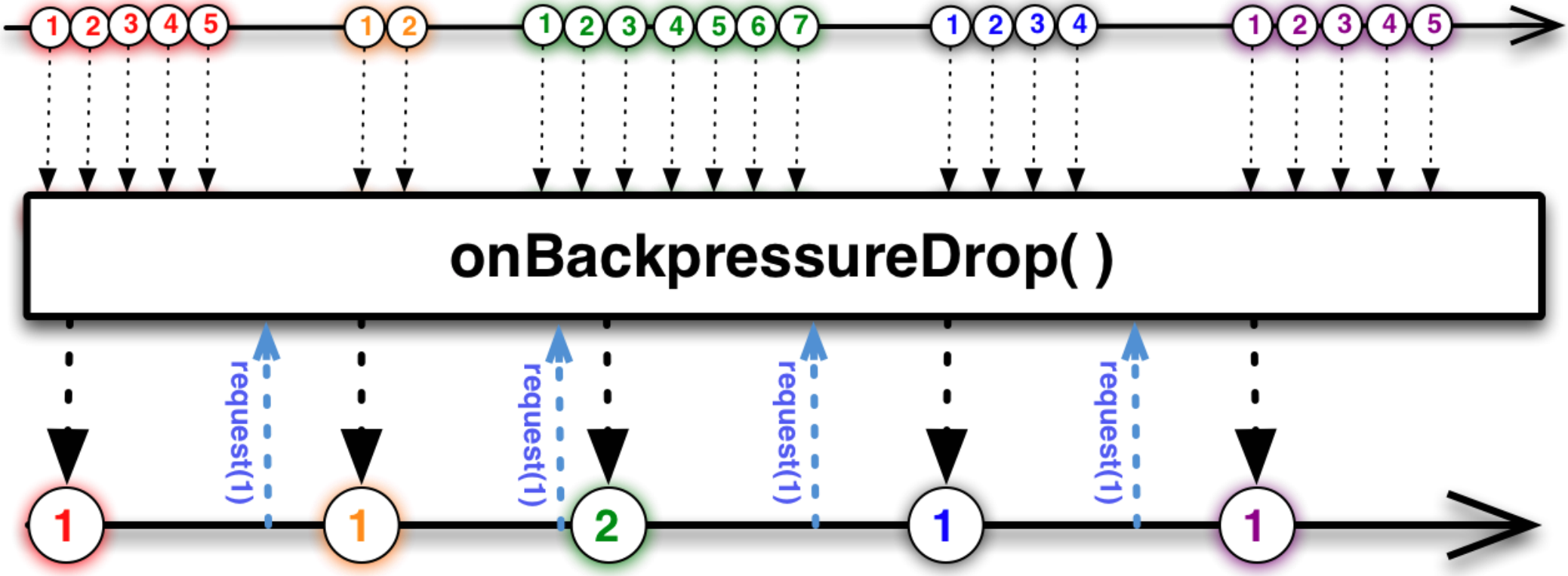
- Cas d'usage : Alternative en cas d'erreur.



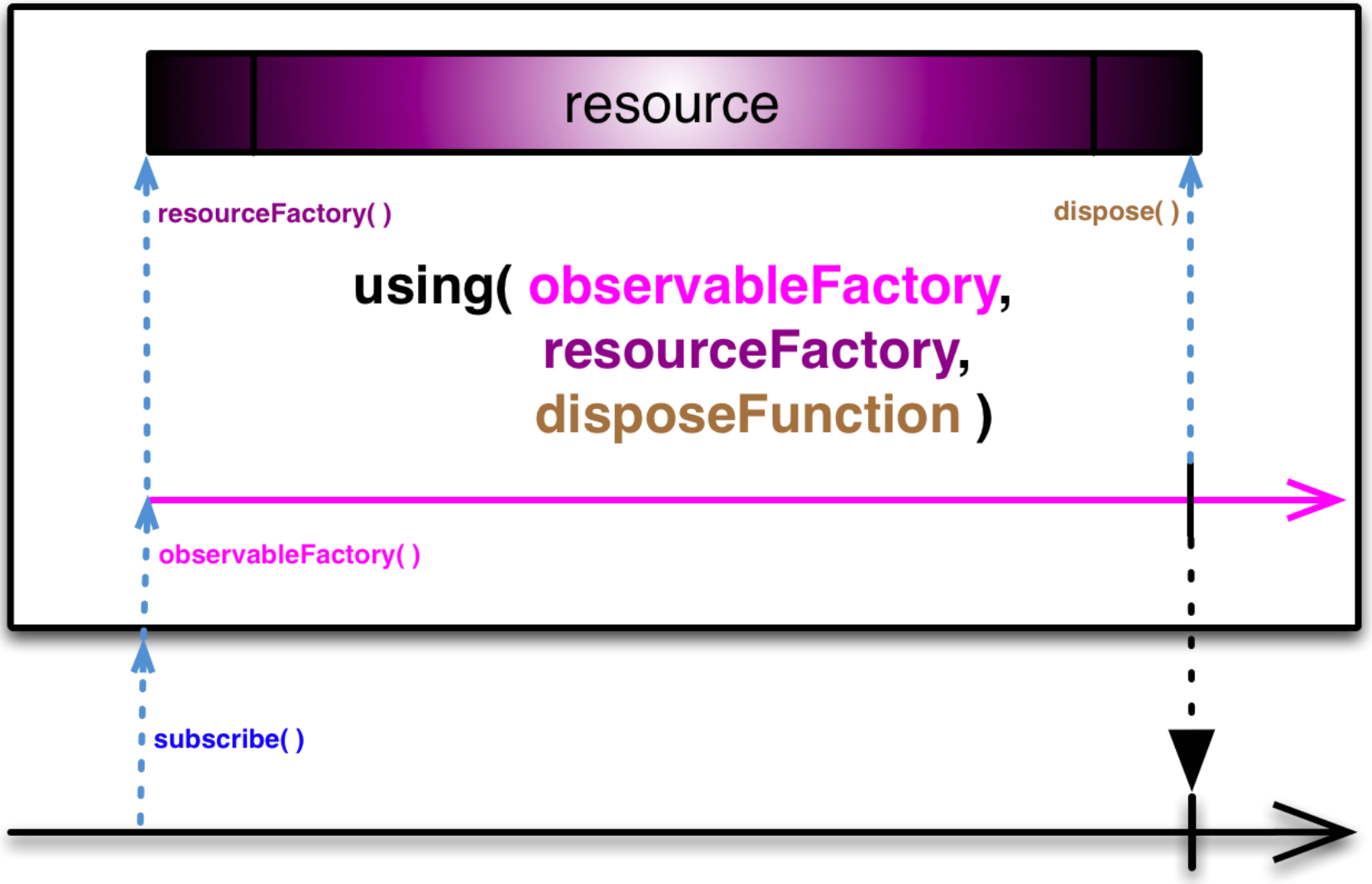
./!\ contrairement à .onErrorResumeNext(...), .onExceptionResumeNext(...) ne catch pas les Throwable.

Back Pressure

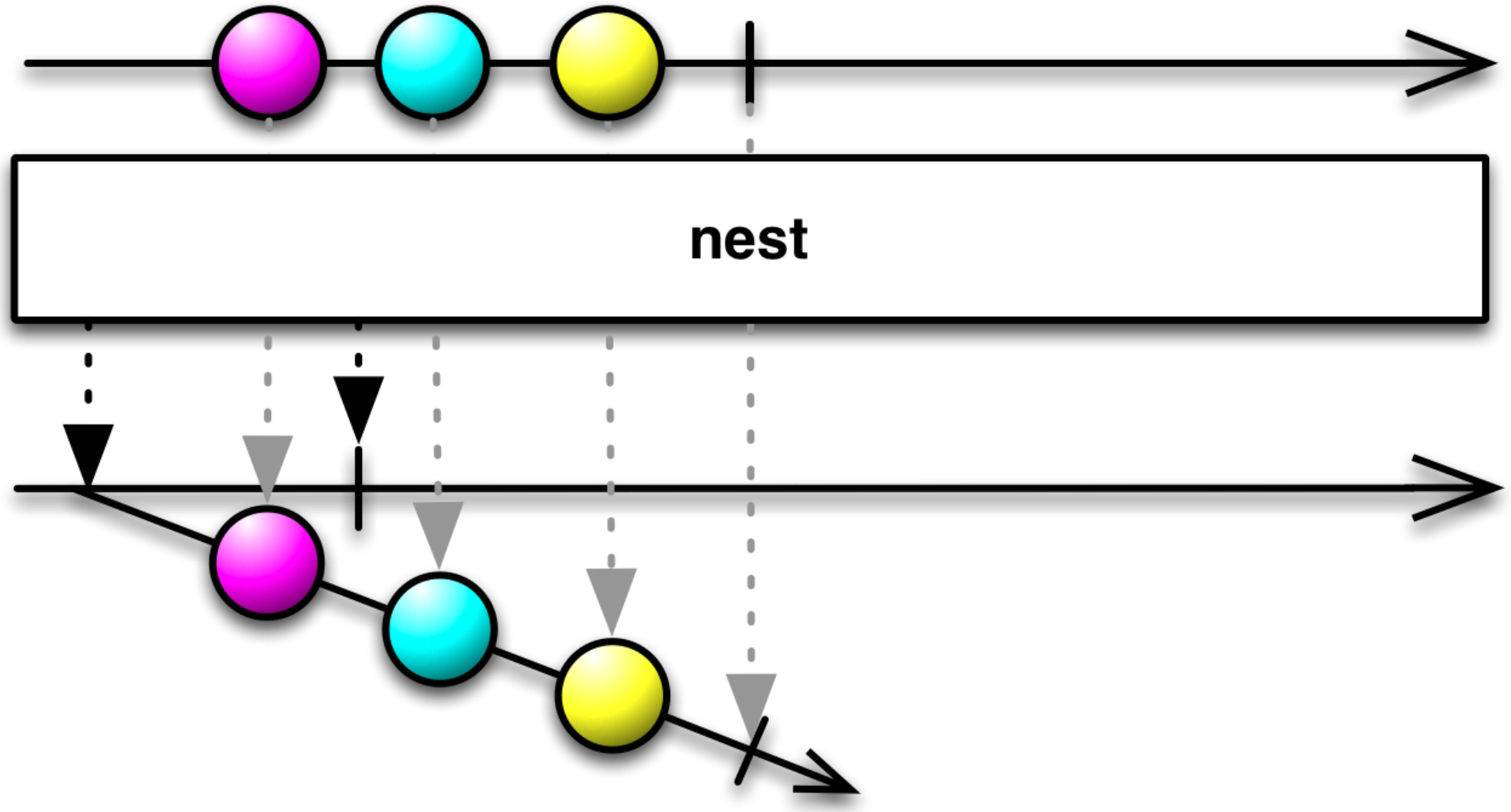

1 2 3 4 5
1 2
1 2 3 4 5 6 7
1 2 3 4
1 2 3 4 5
onBackpressureBuffer( )
{ 1 2 1 2 3 4 5 6 7 1 2 3 4 1 2 3 4 5 ... }
request(1)
request(1)
request(1)
request(1)
1
2
3
4
5

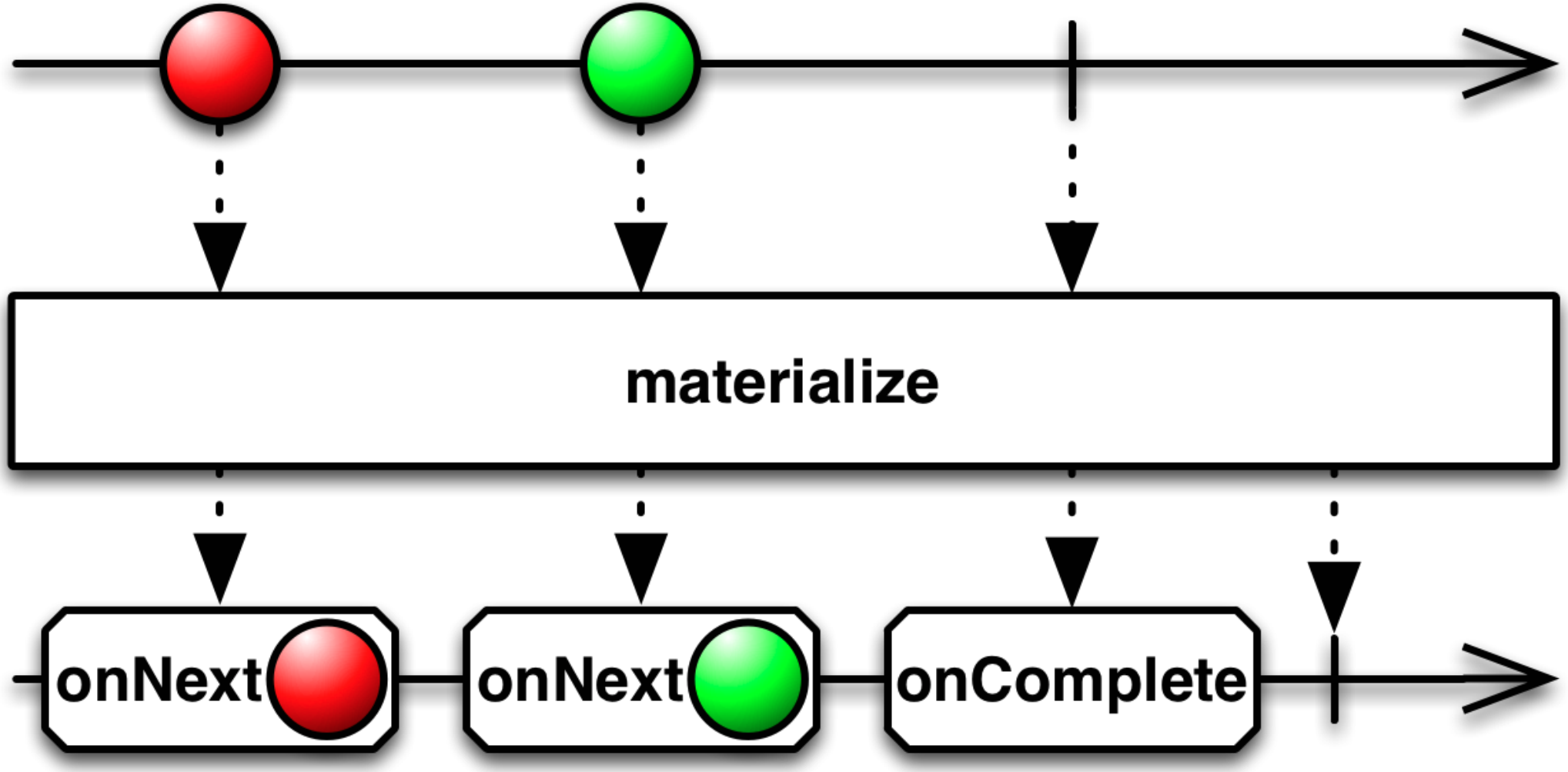

1 2 3 4 5
1 2
1 2 3 4 5 6 7
1 2 3 4
1 2 3 4 5
onBackpressureDrop( )
request(1)
request(1)
request(1)
request(1)
1
1
2
1
1

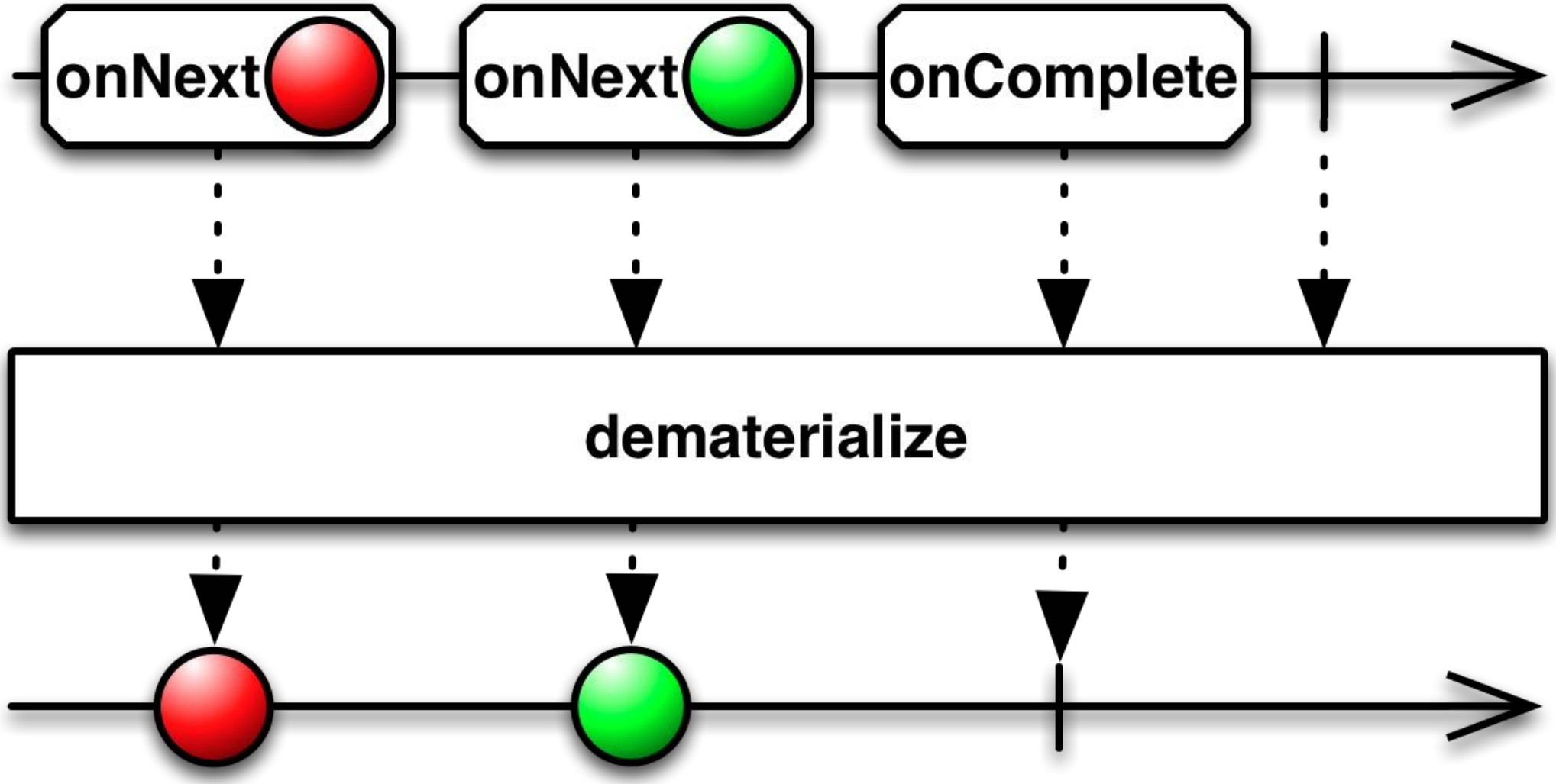
Utilitaire

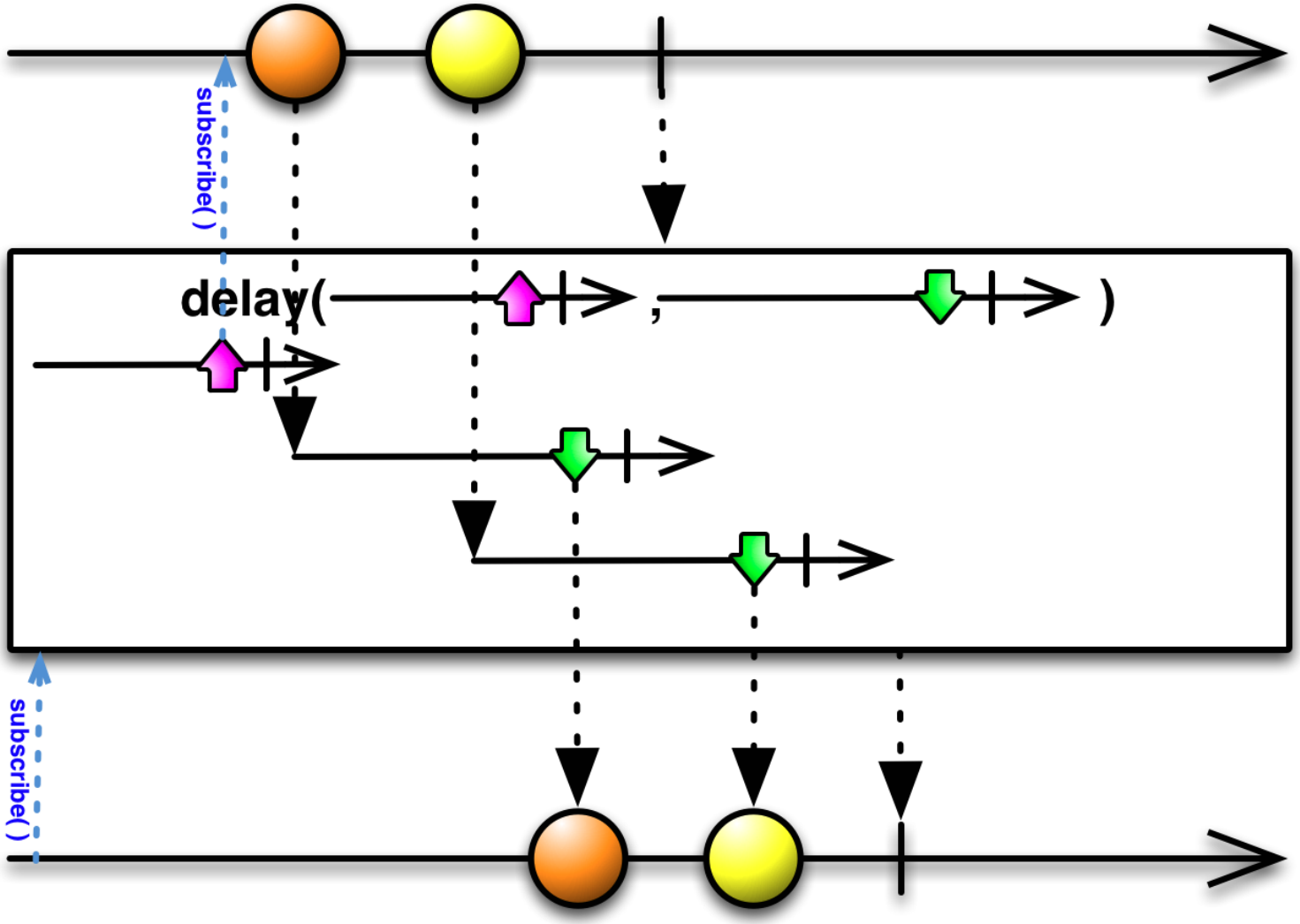


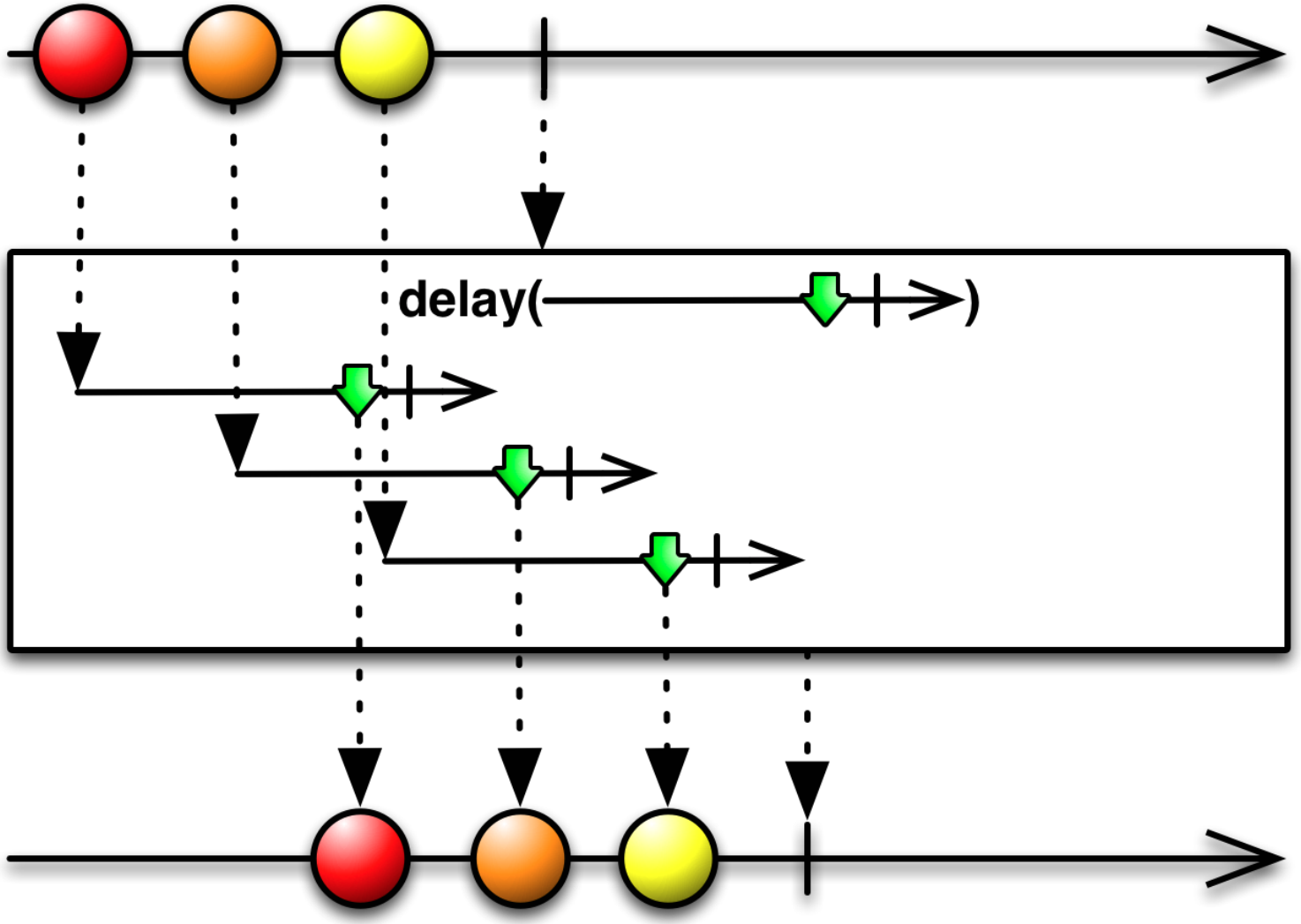
- Cas d'usage : Pour utiliser une ressource (fichier, connexion, etc.) pendant la durée de vie d'un flux.

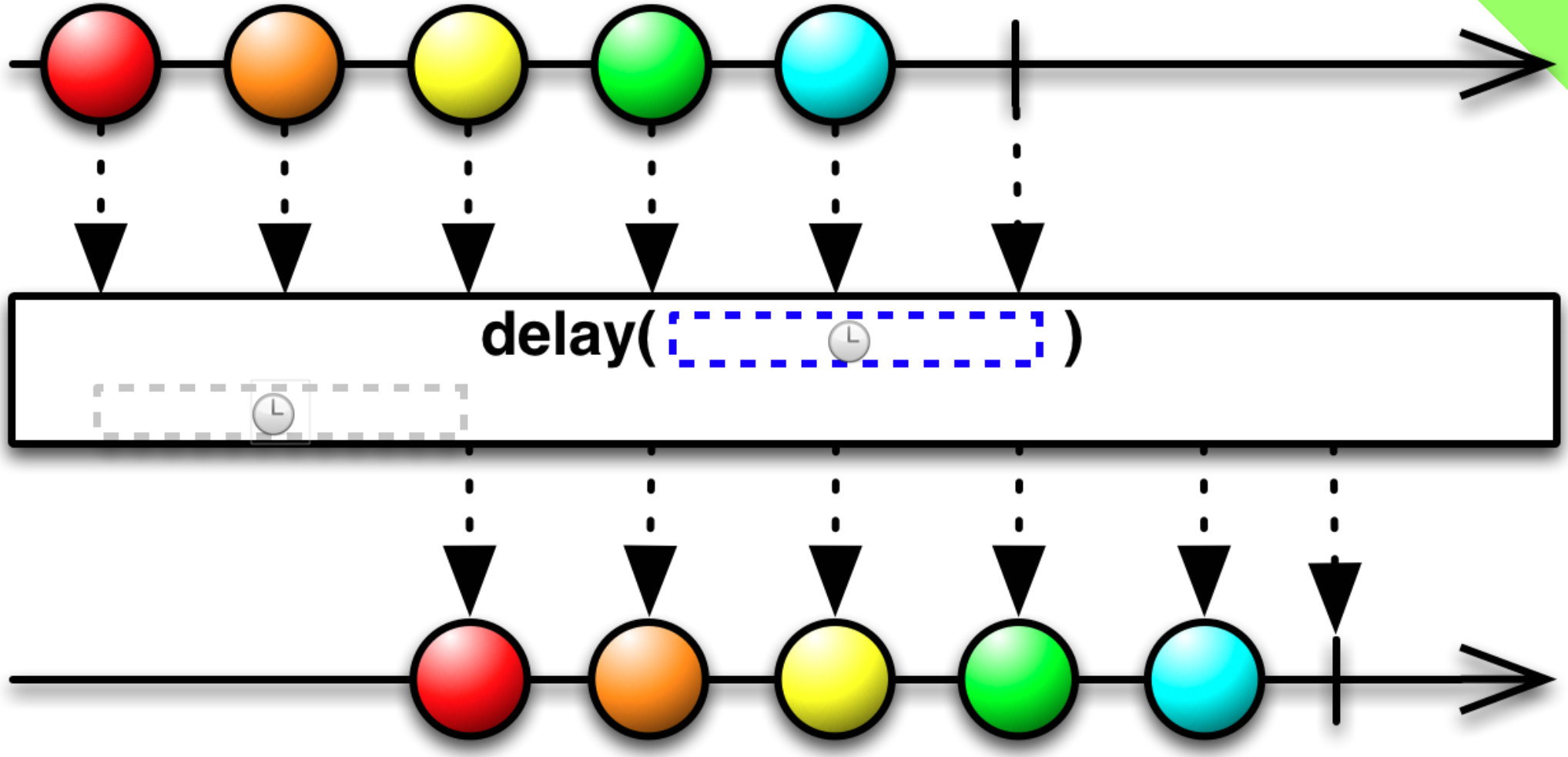

nest


materialize
onNext
onNext
onComplete

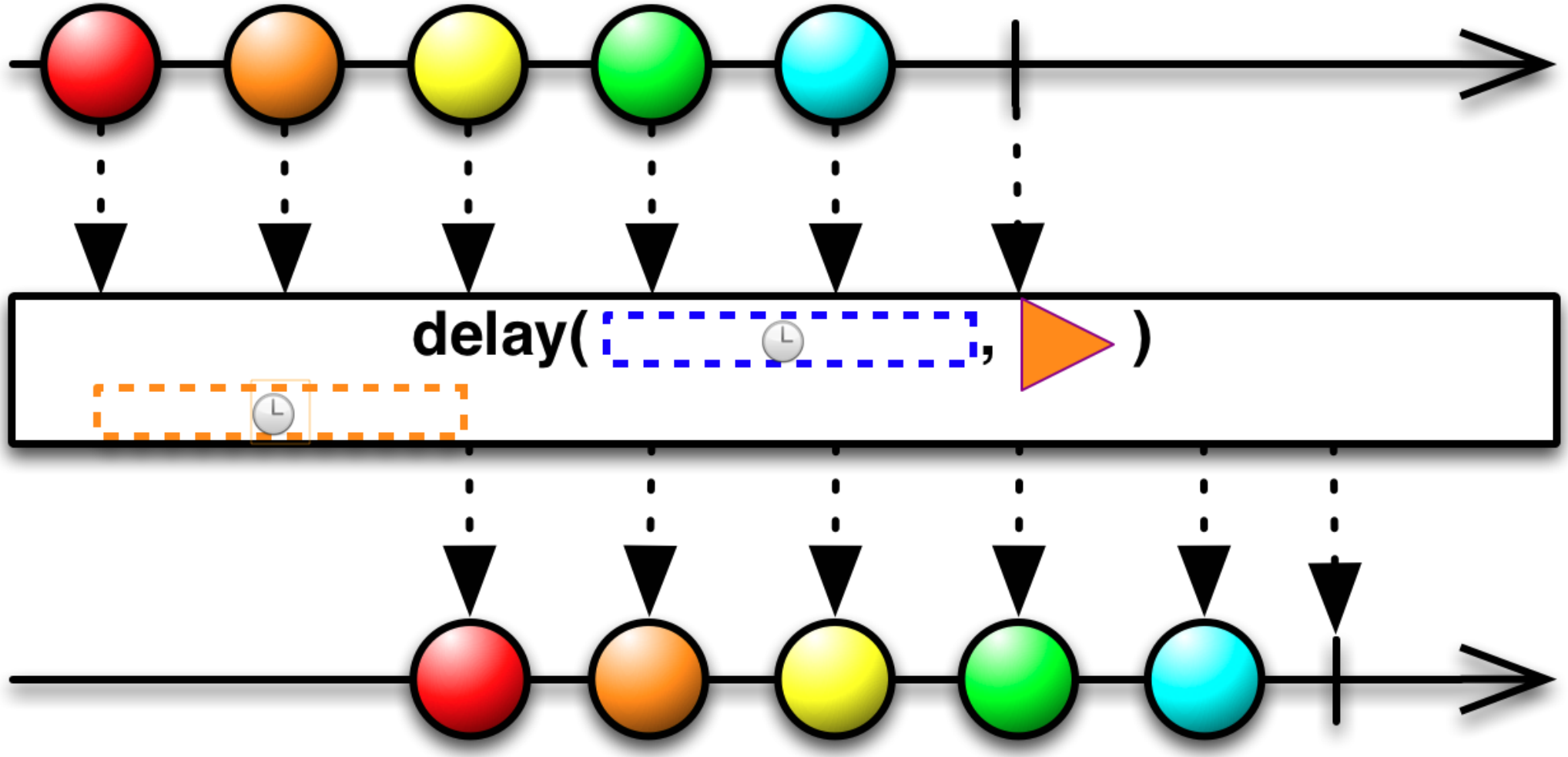

onNext
onNext
onComplete
dematerialize

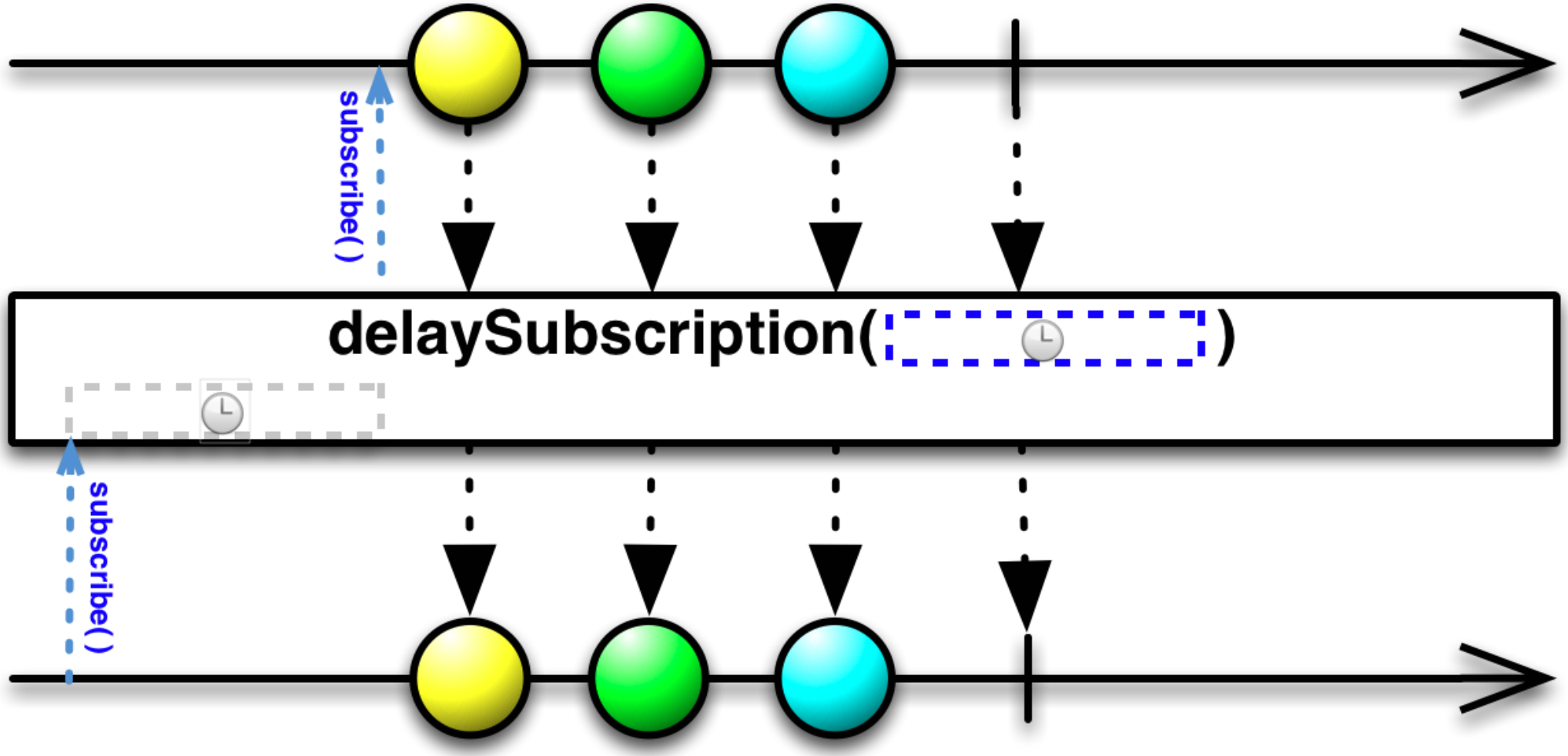

subscribe( )
delay(,)
subscribe( )

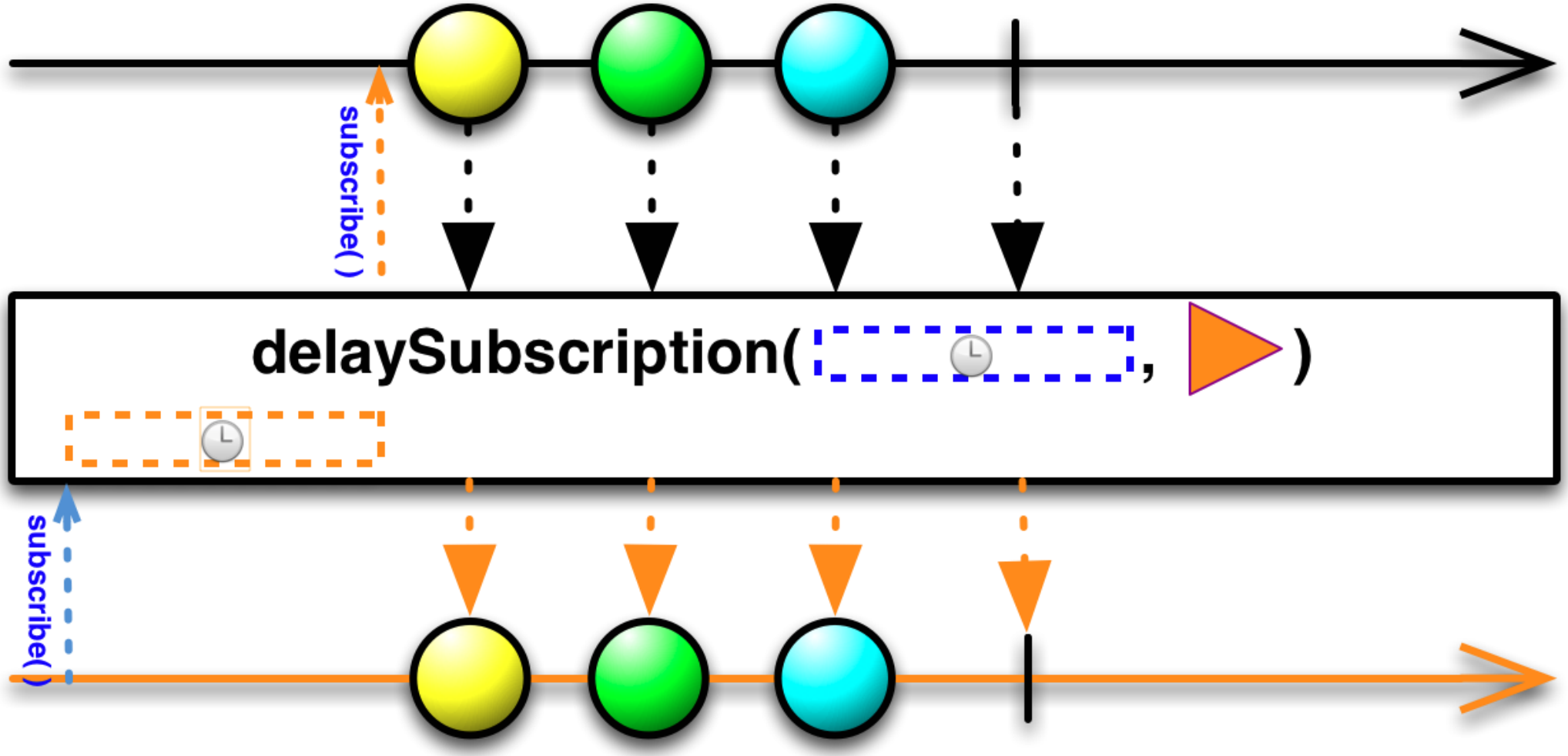

delay(
)

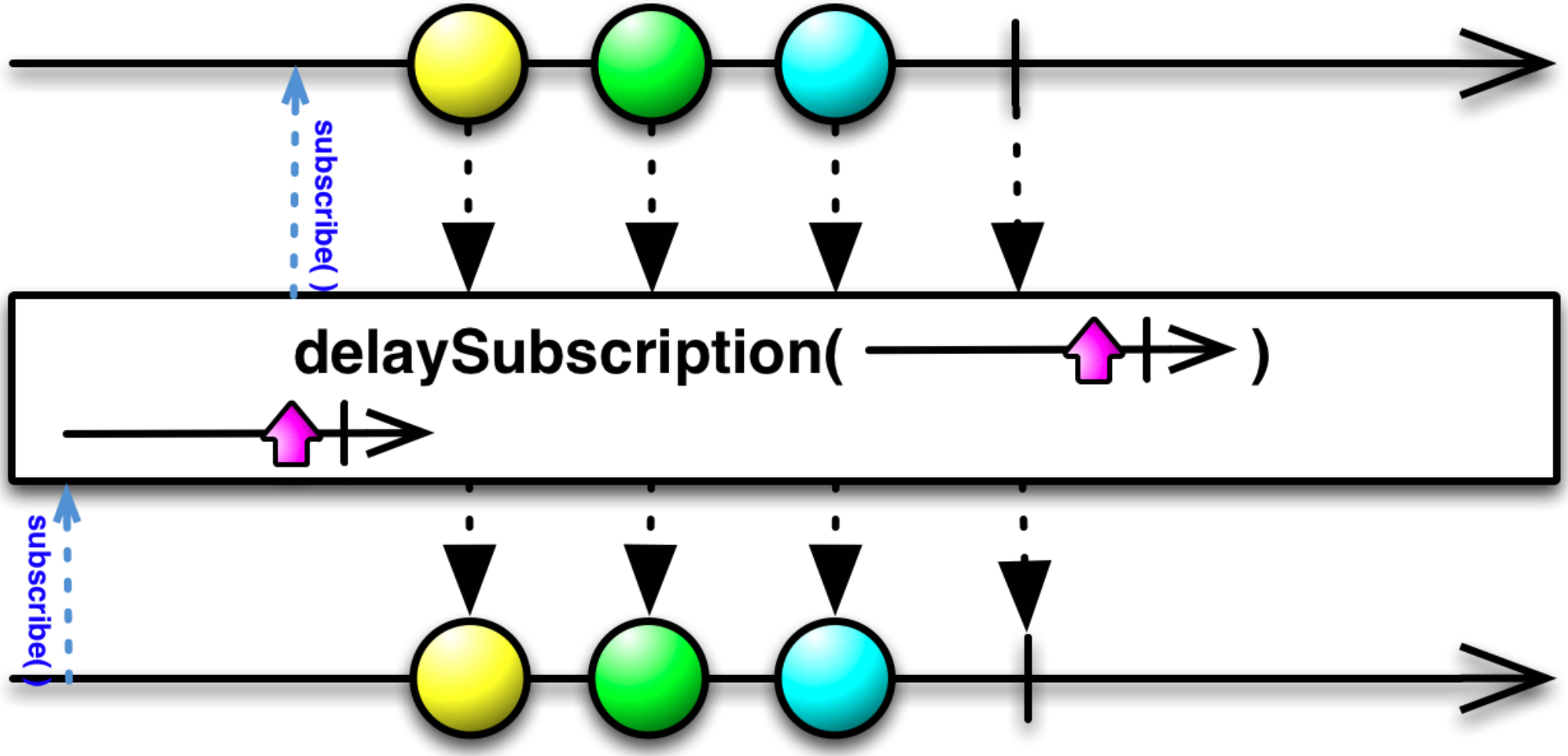


- Cas d'usage : Décalage temporel des valeurs.

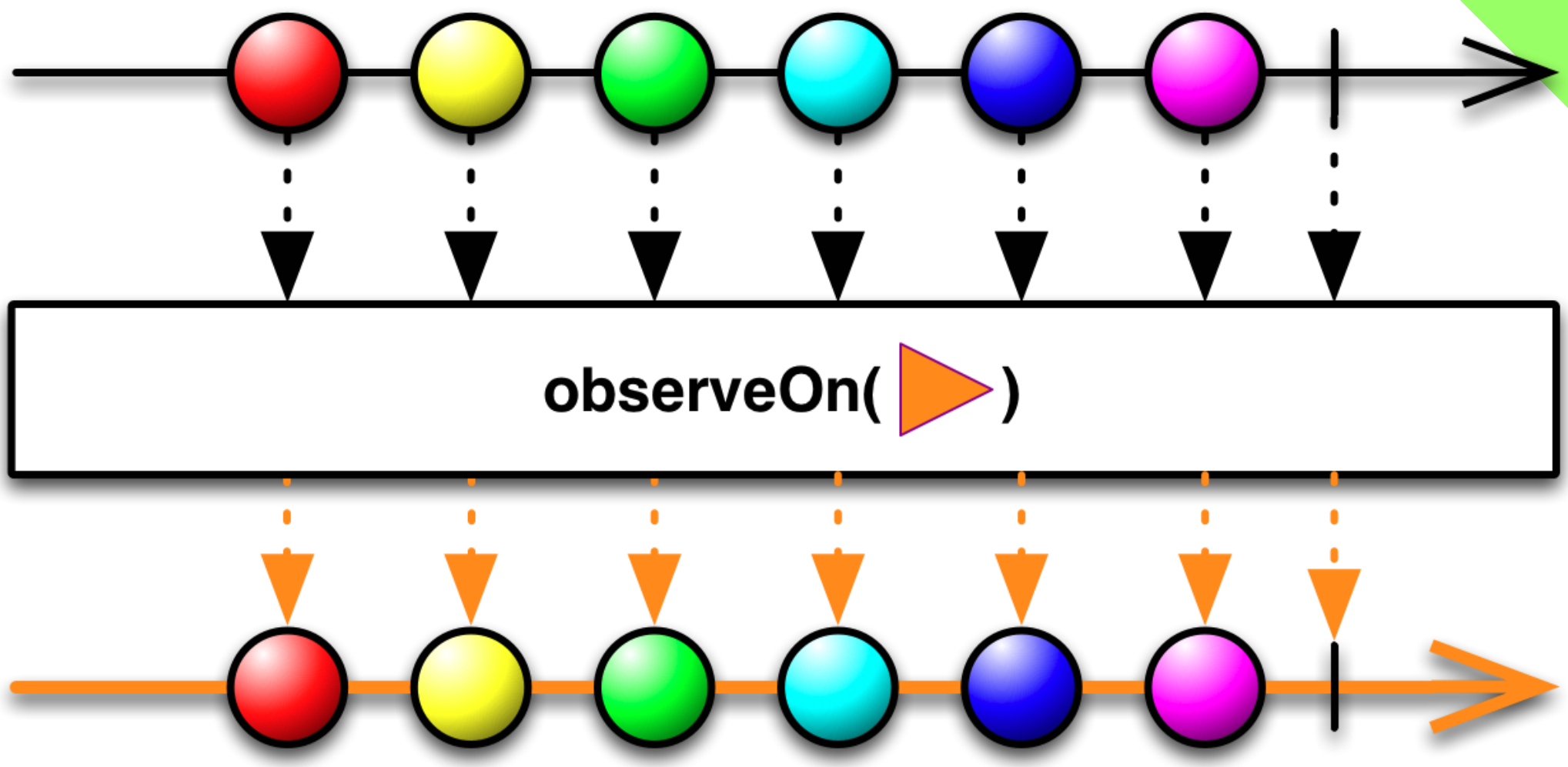

delay(,)


subscribe()
delaySubscription()
subscribe()

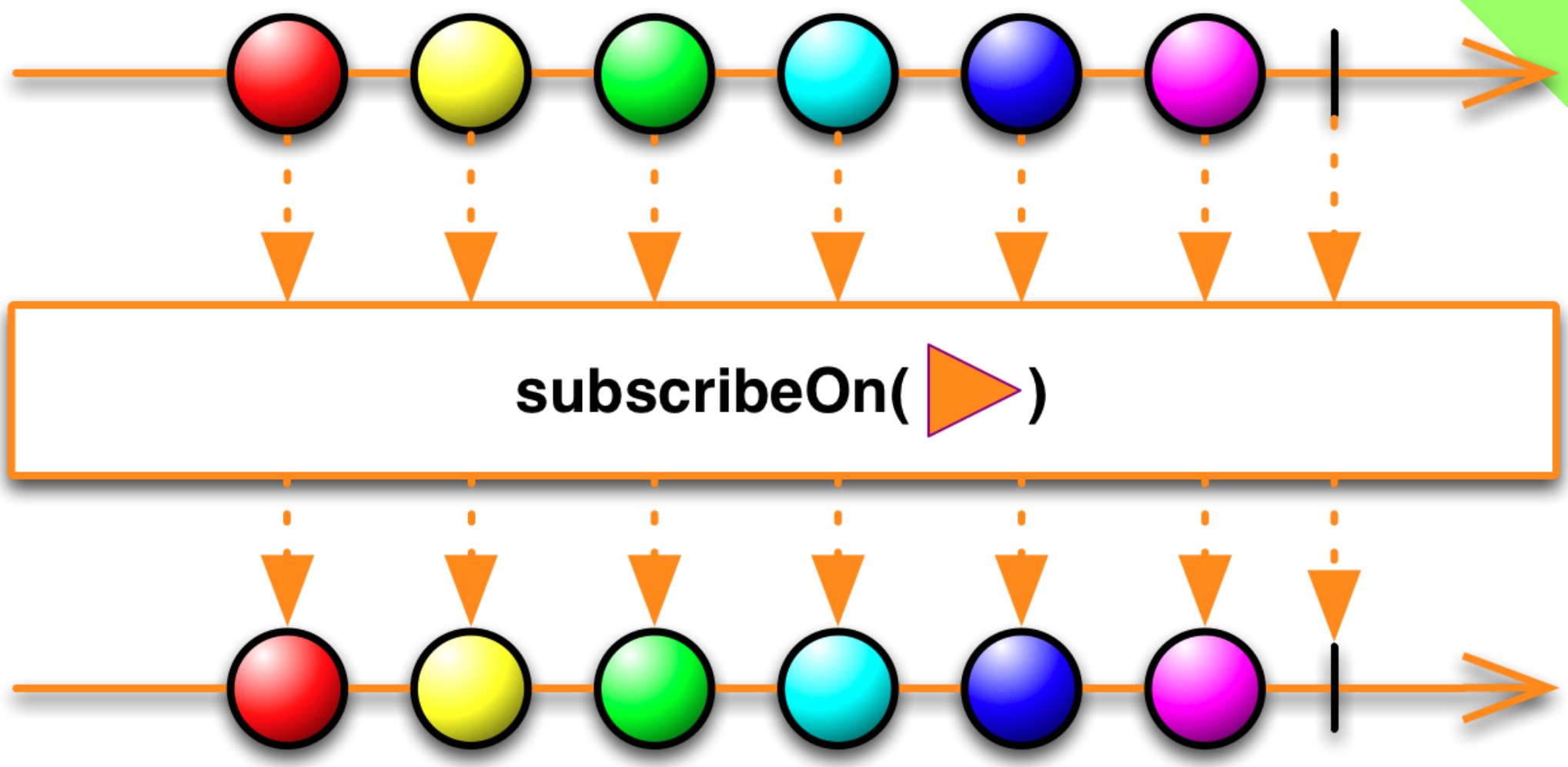

subscribe()
delaySubscription(,)
subscribe()


subscribe( )
delaySubscription( )
subscribe( )

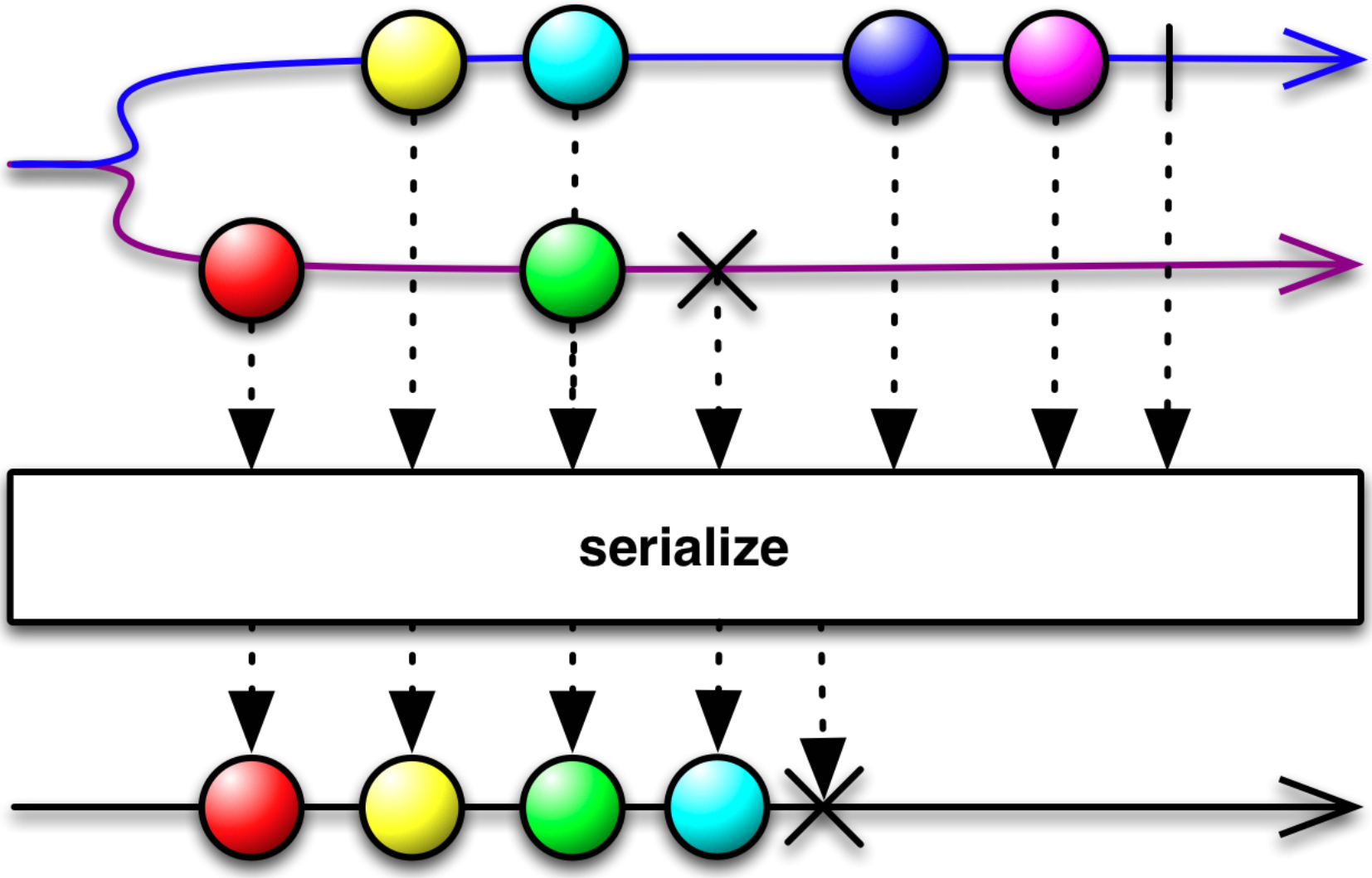
Scheduling



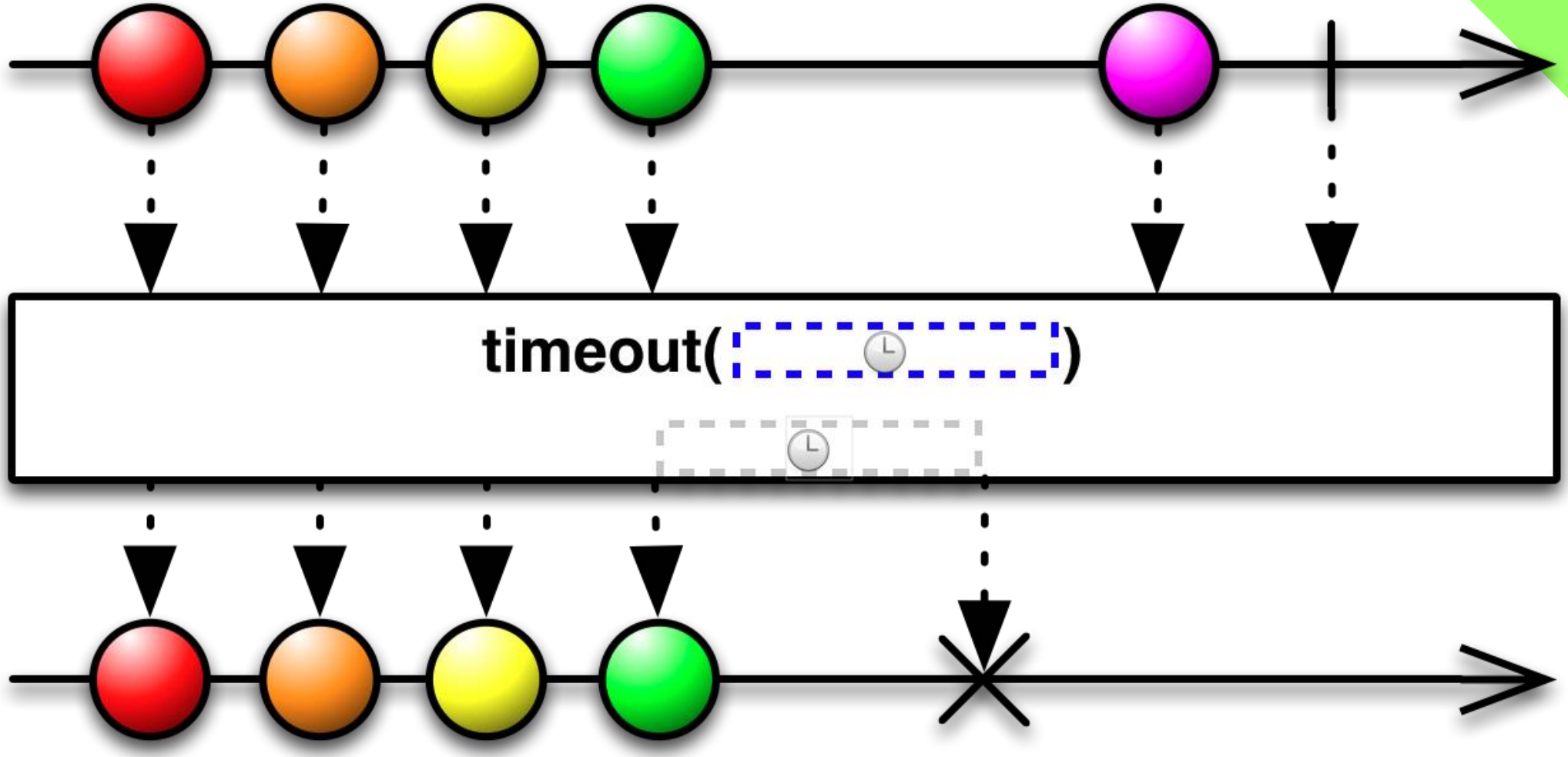
- Cas d'usage : Changement de **Scheduler** pour la suite du traitement.

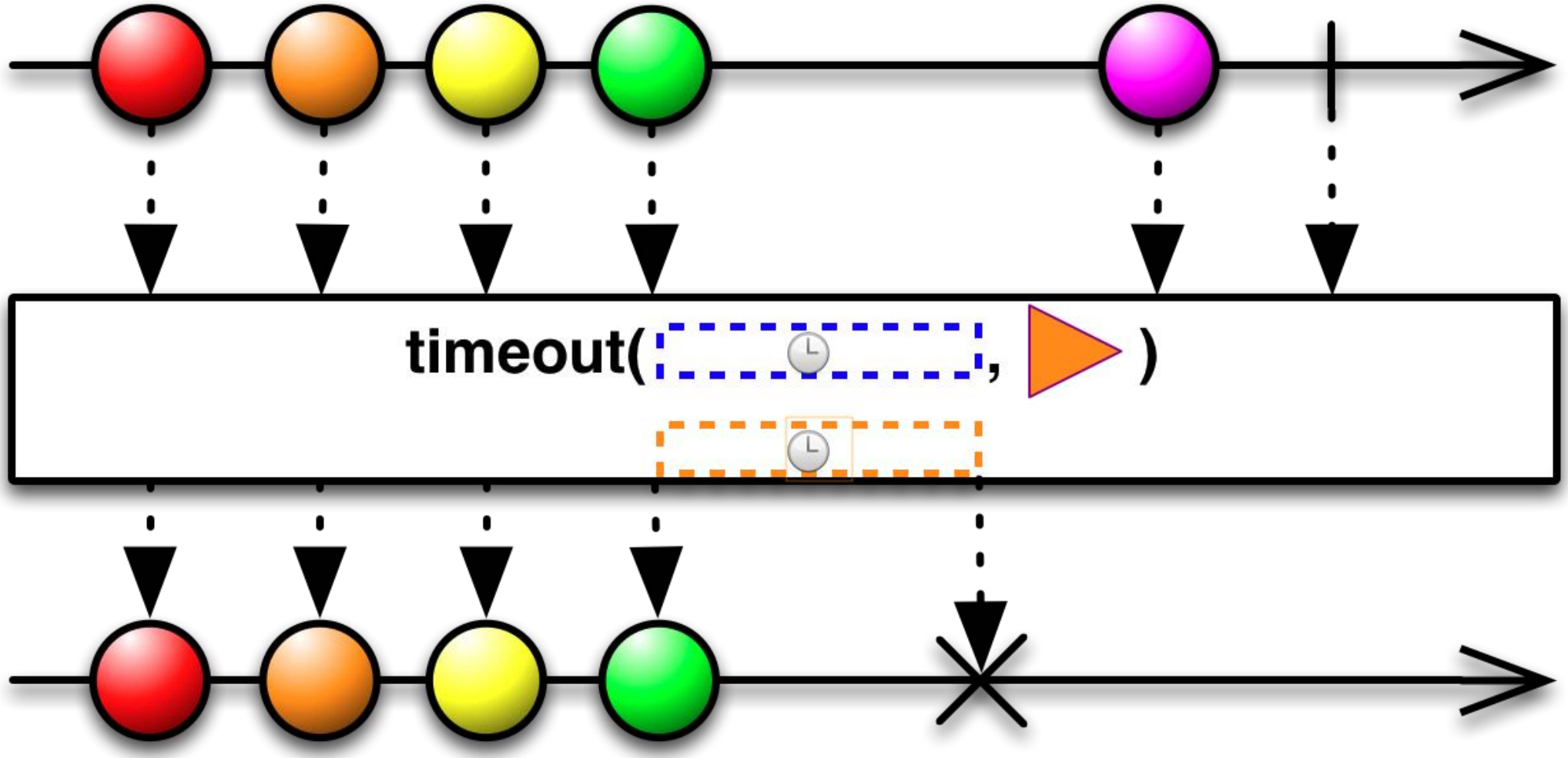


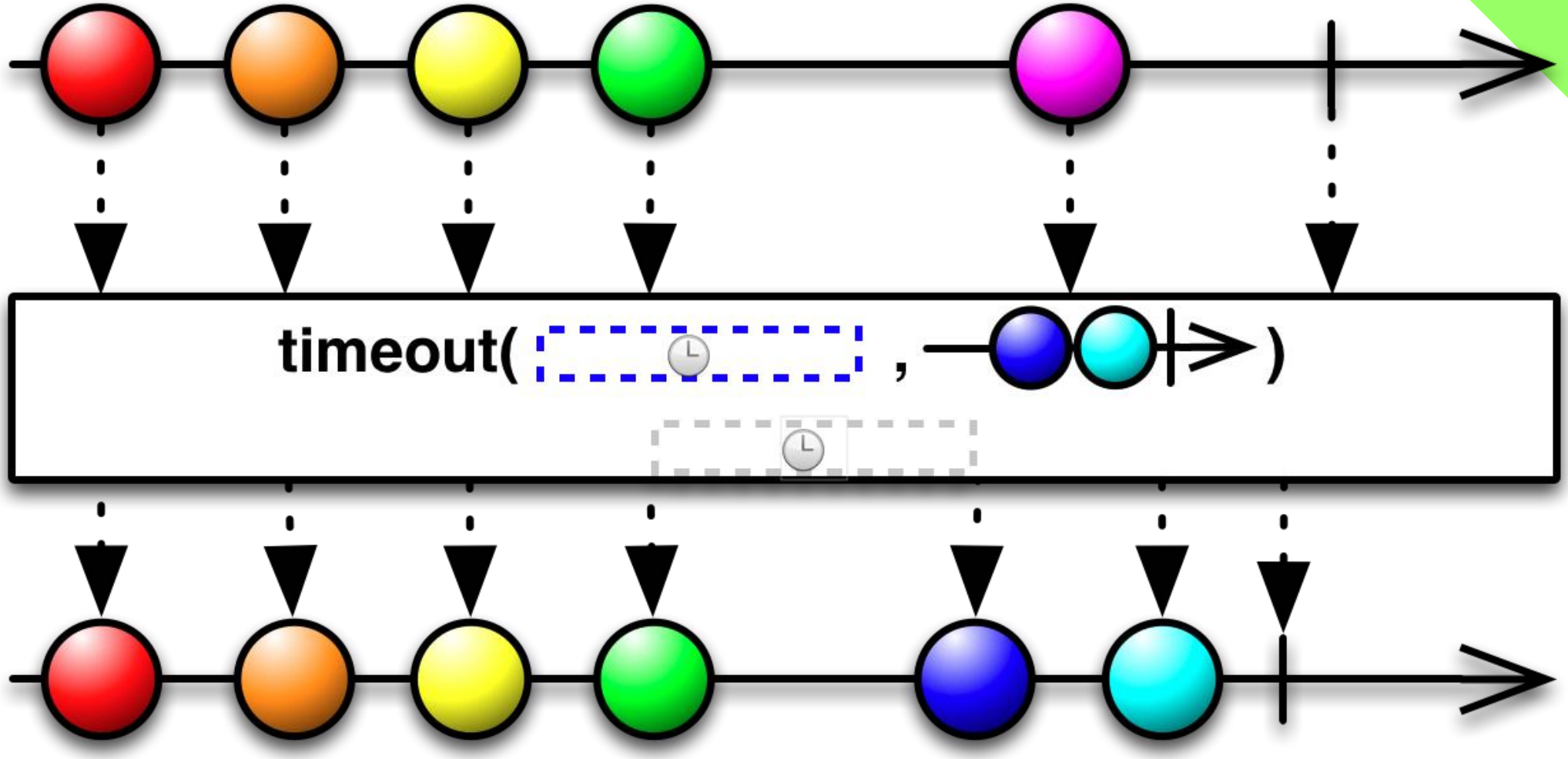
.Cas d'usage : La création du flux de données se fera sur le **Scheduler** passé en paramètre.


serialize

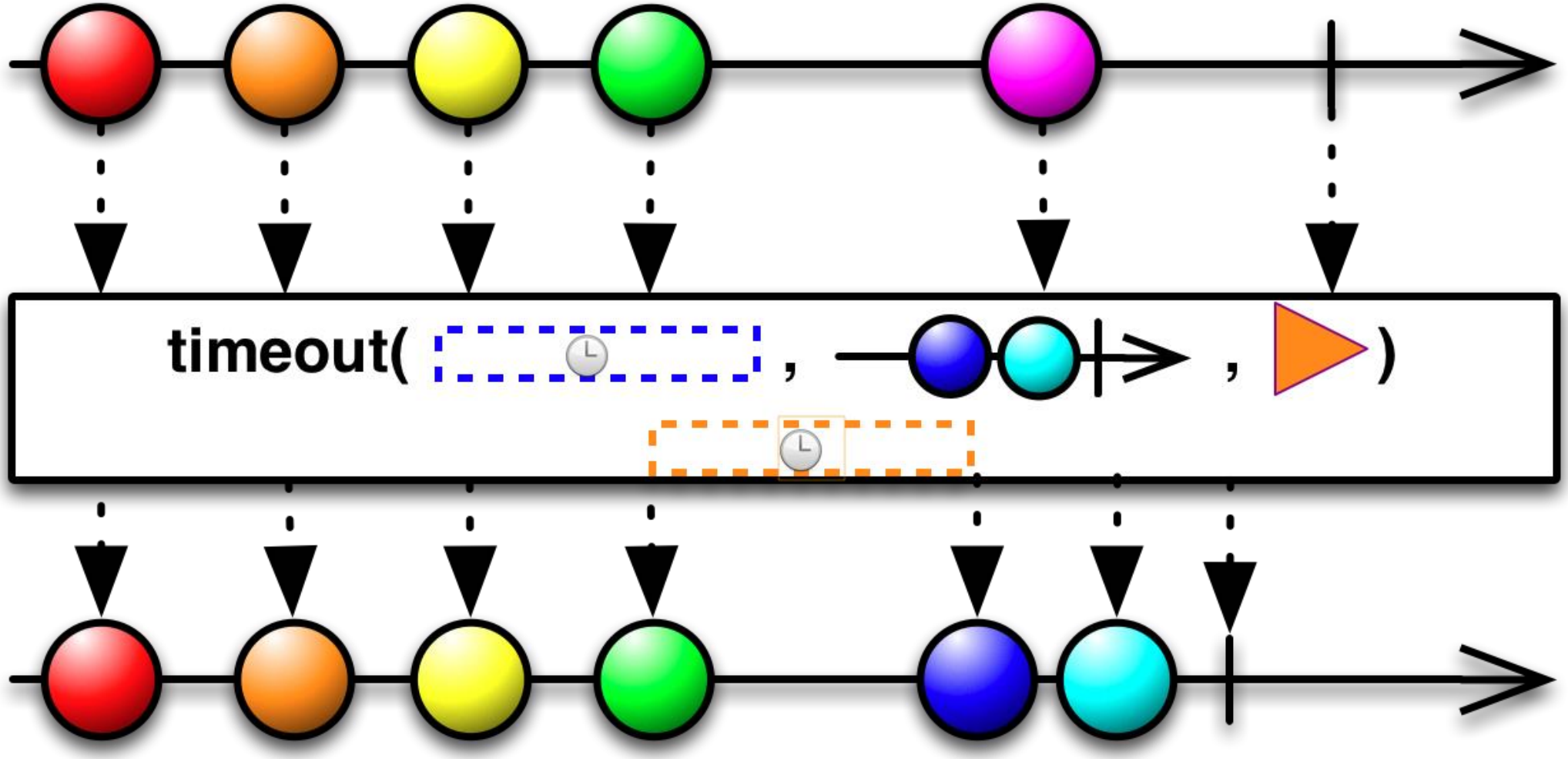
Timeout

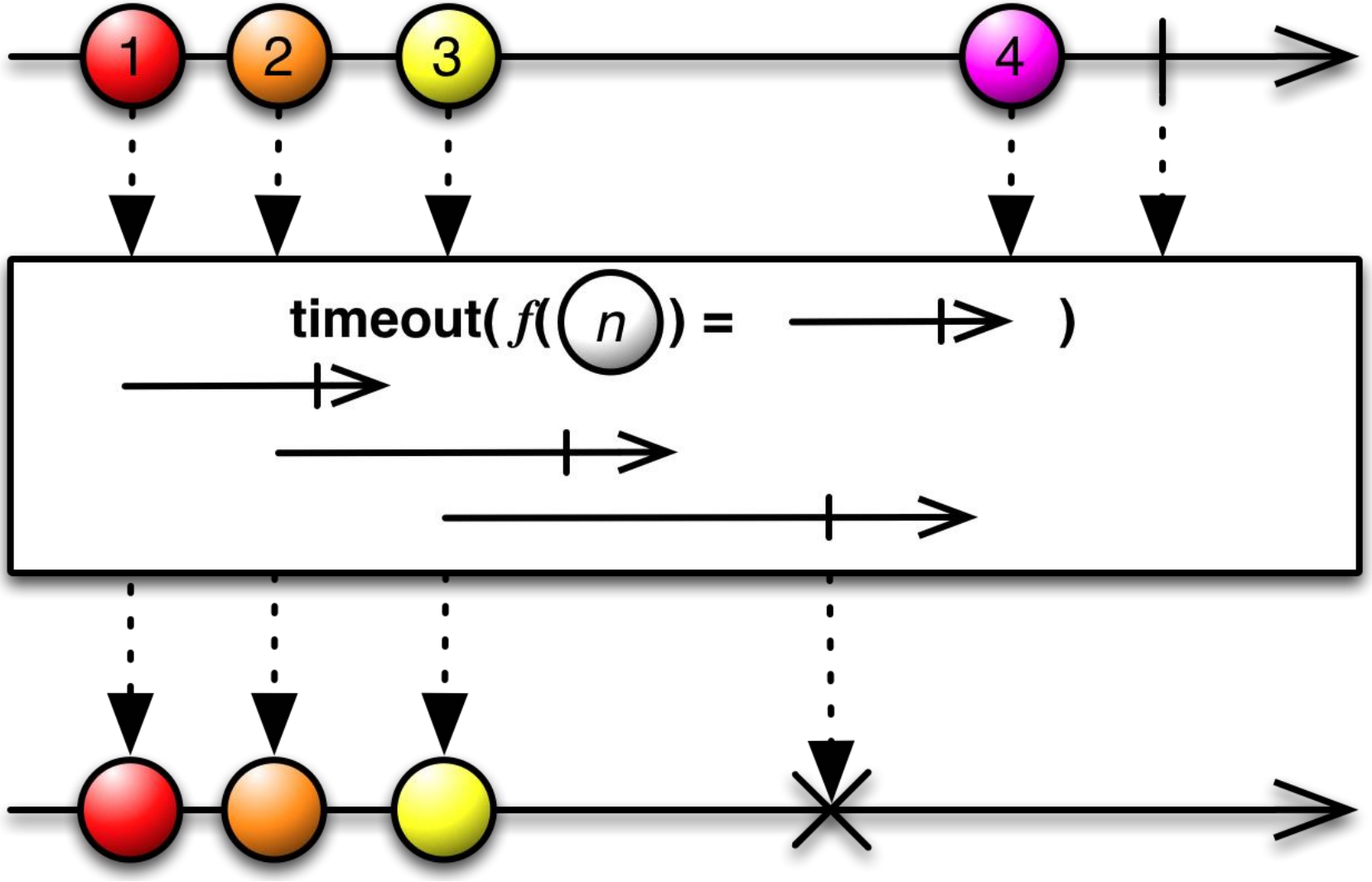

timeout( )

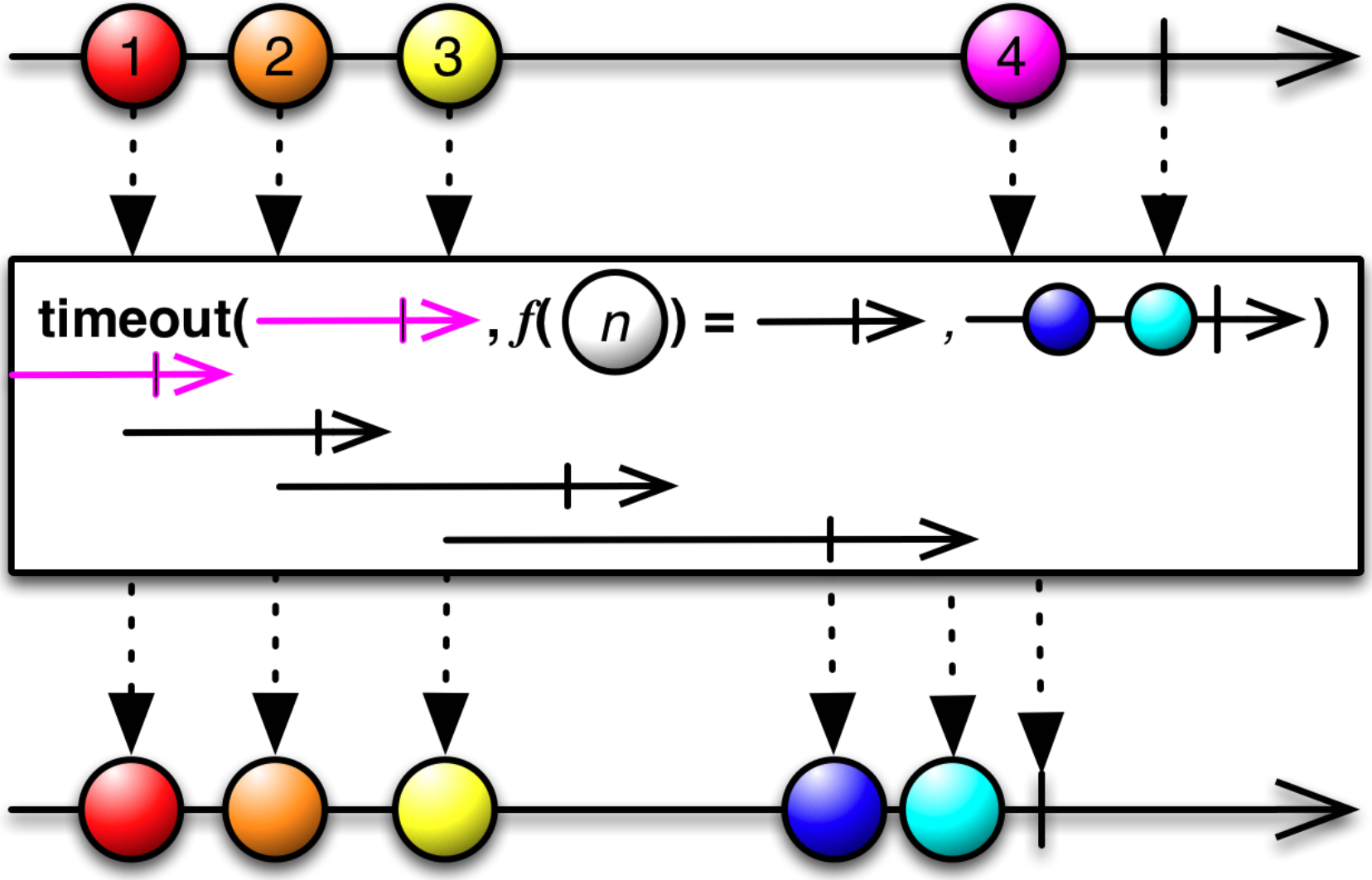

timeout(,)



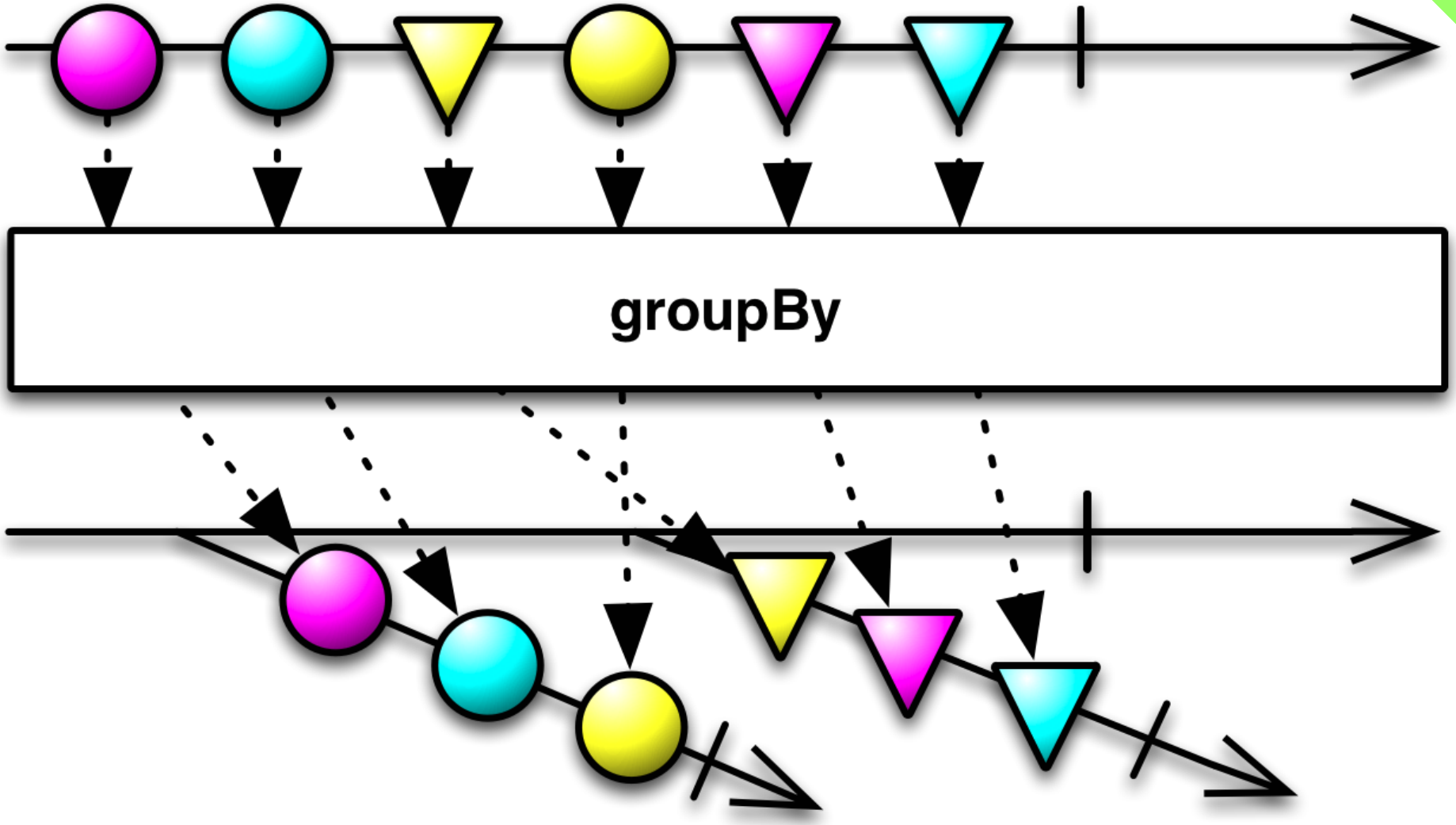
..timeout(...) avec valeurs par défaut.


timeout(, ,)


1
2
3
4
timeout(f(n)) =
)

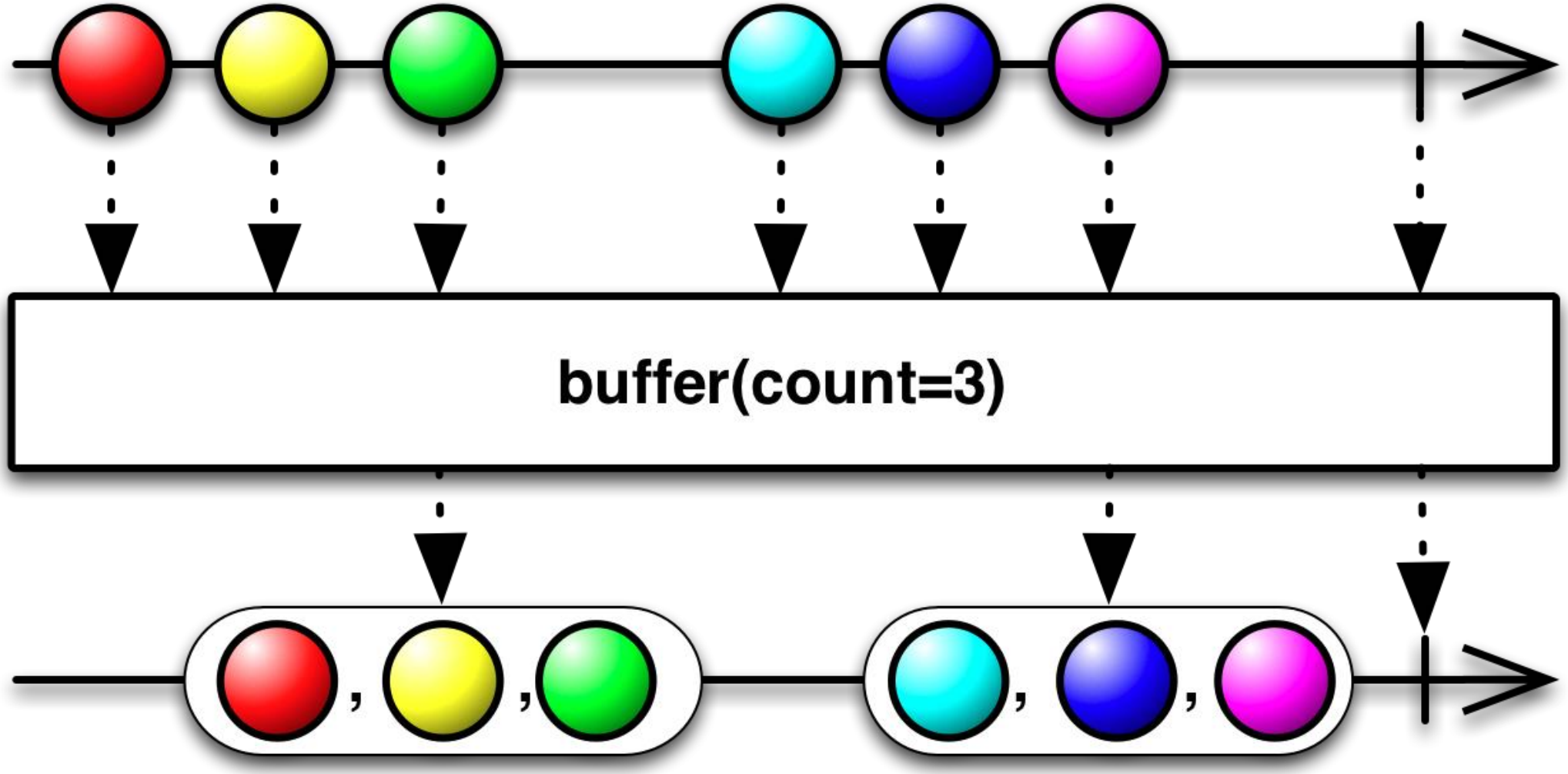

1
2
3
4
timeout(
, f(
n
) =
,
)

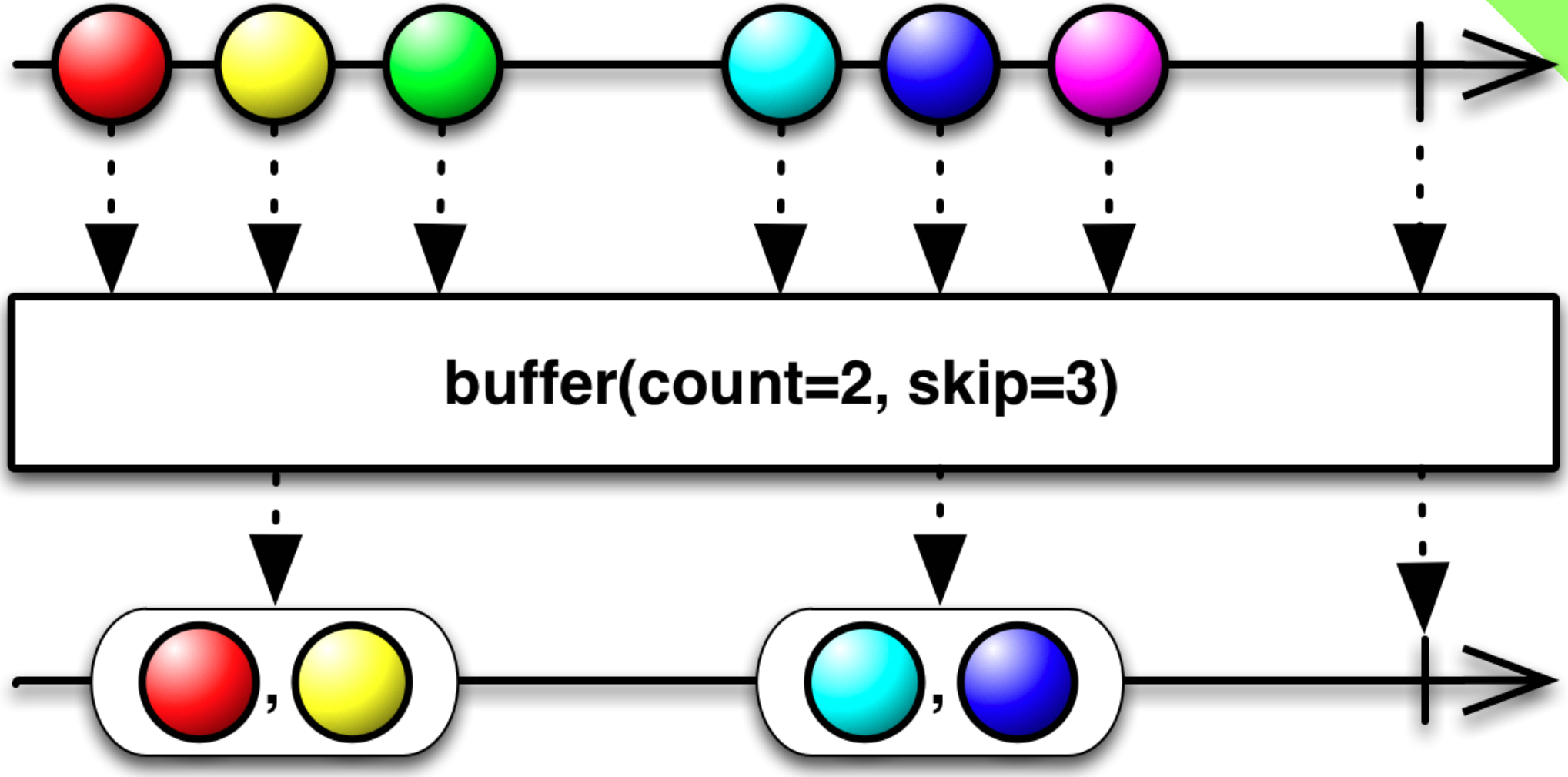
Regroupement

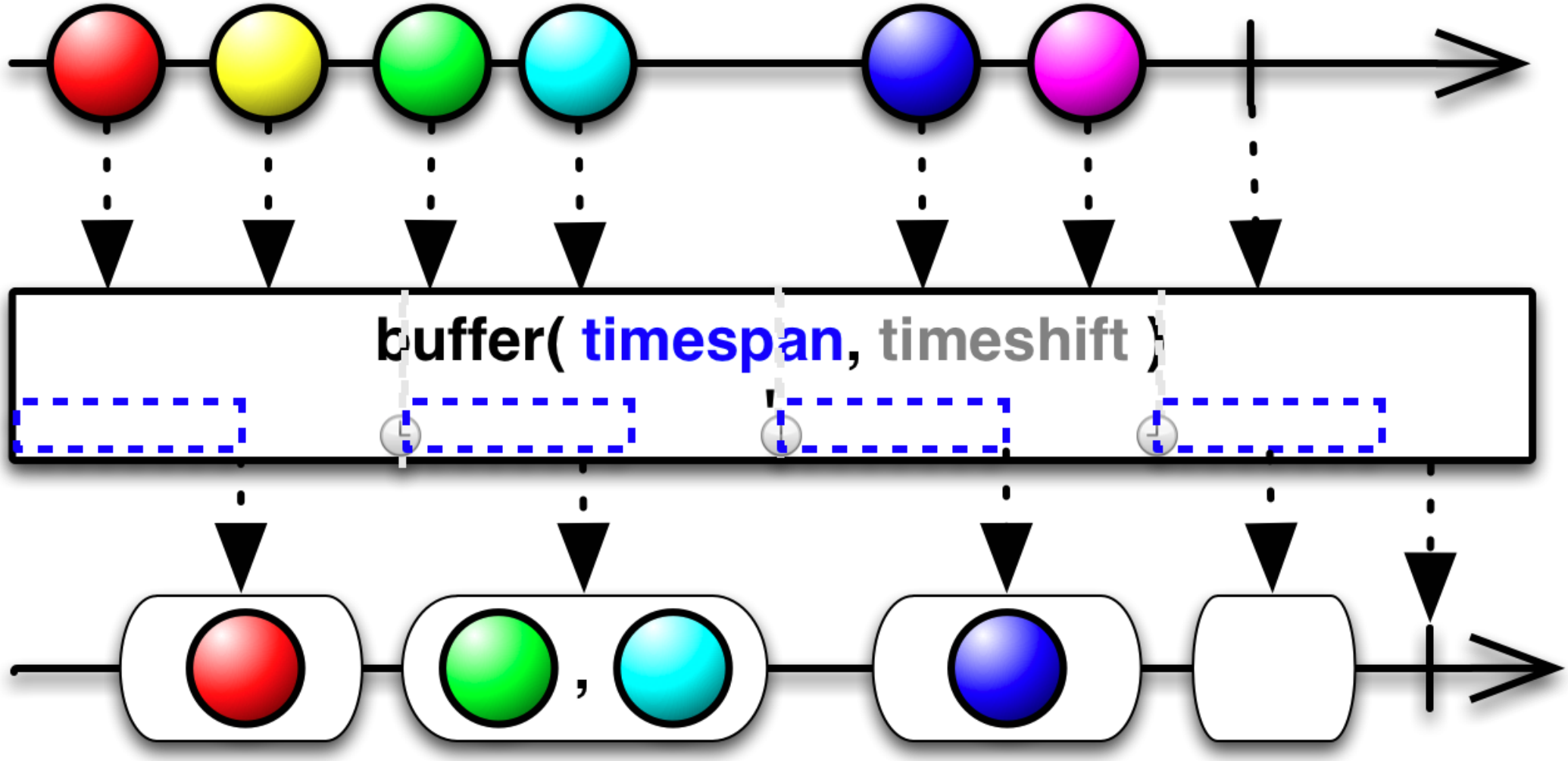


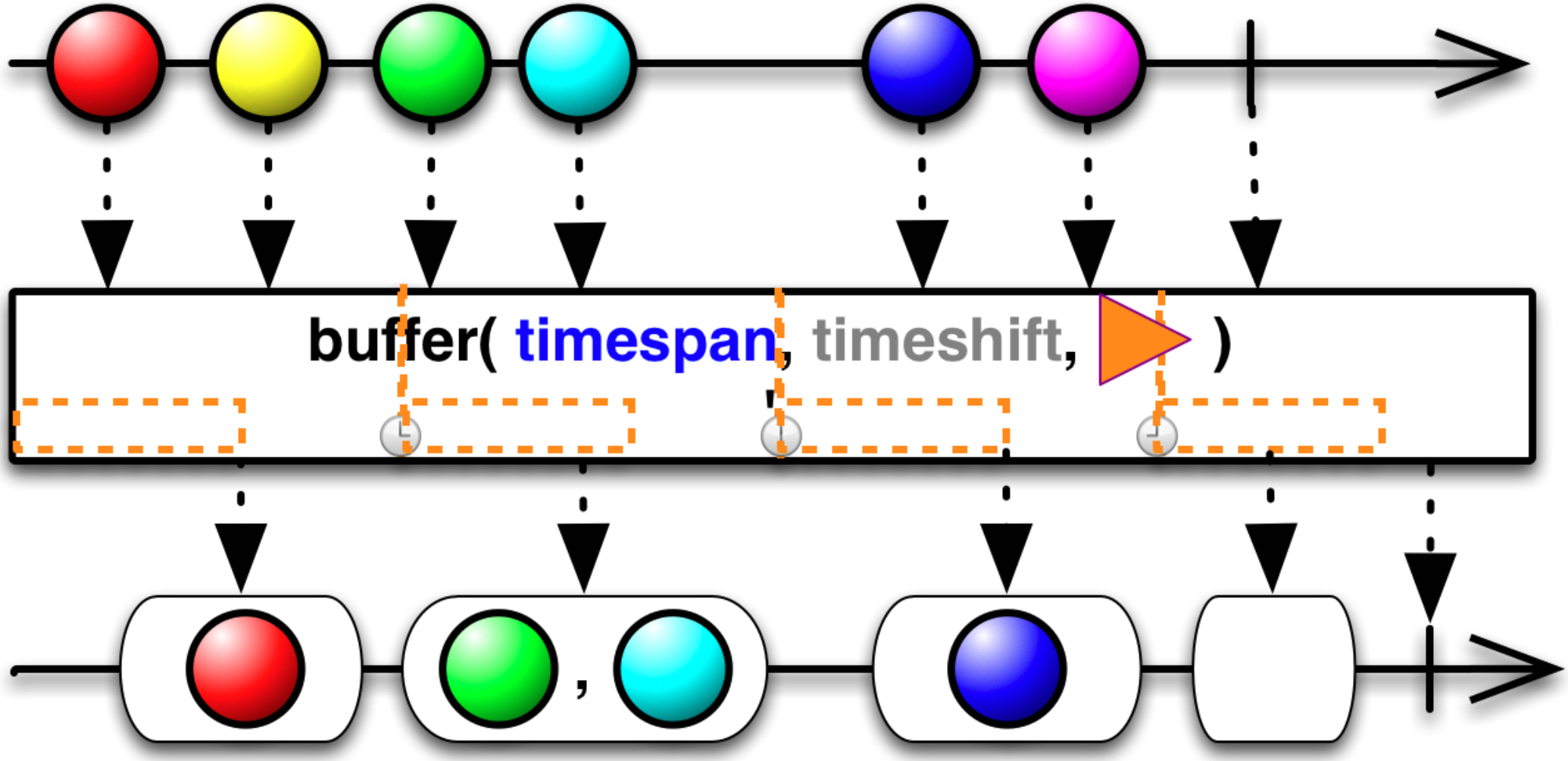
.class **GroupedObservable<K, T>**
  extends Observable<T>

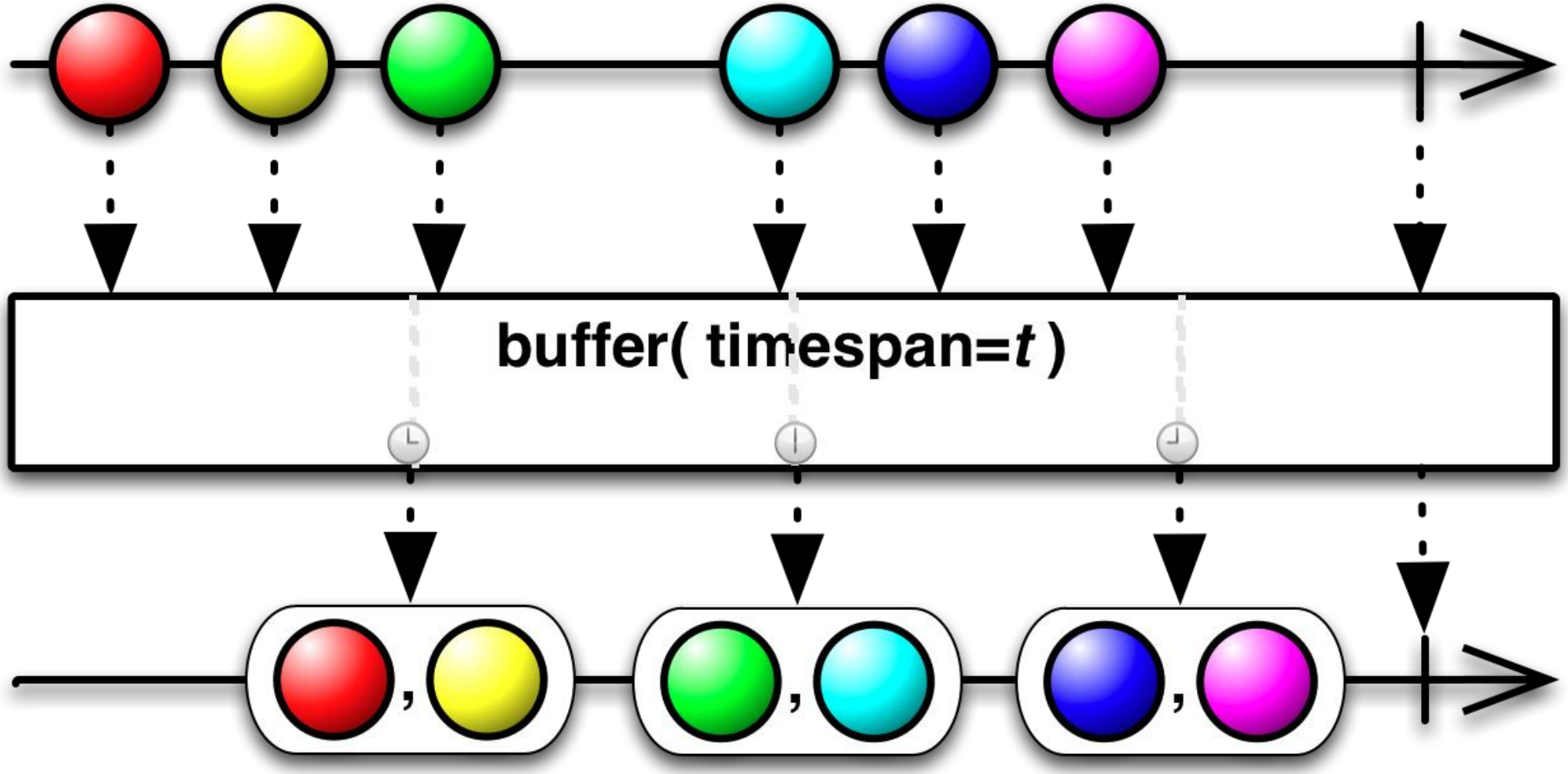
.public K **getKey**()

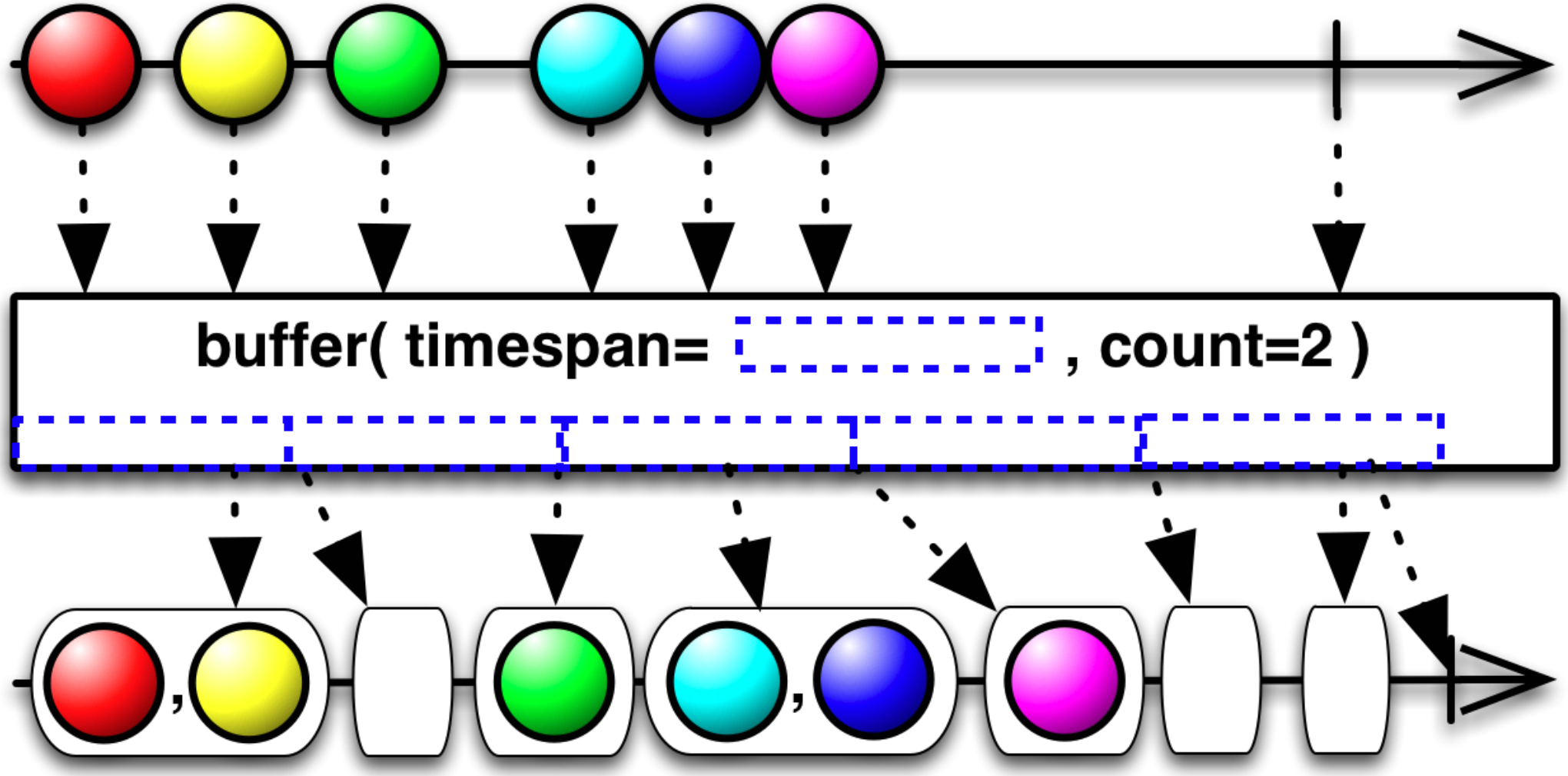

buffer(count=3)

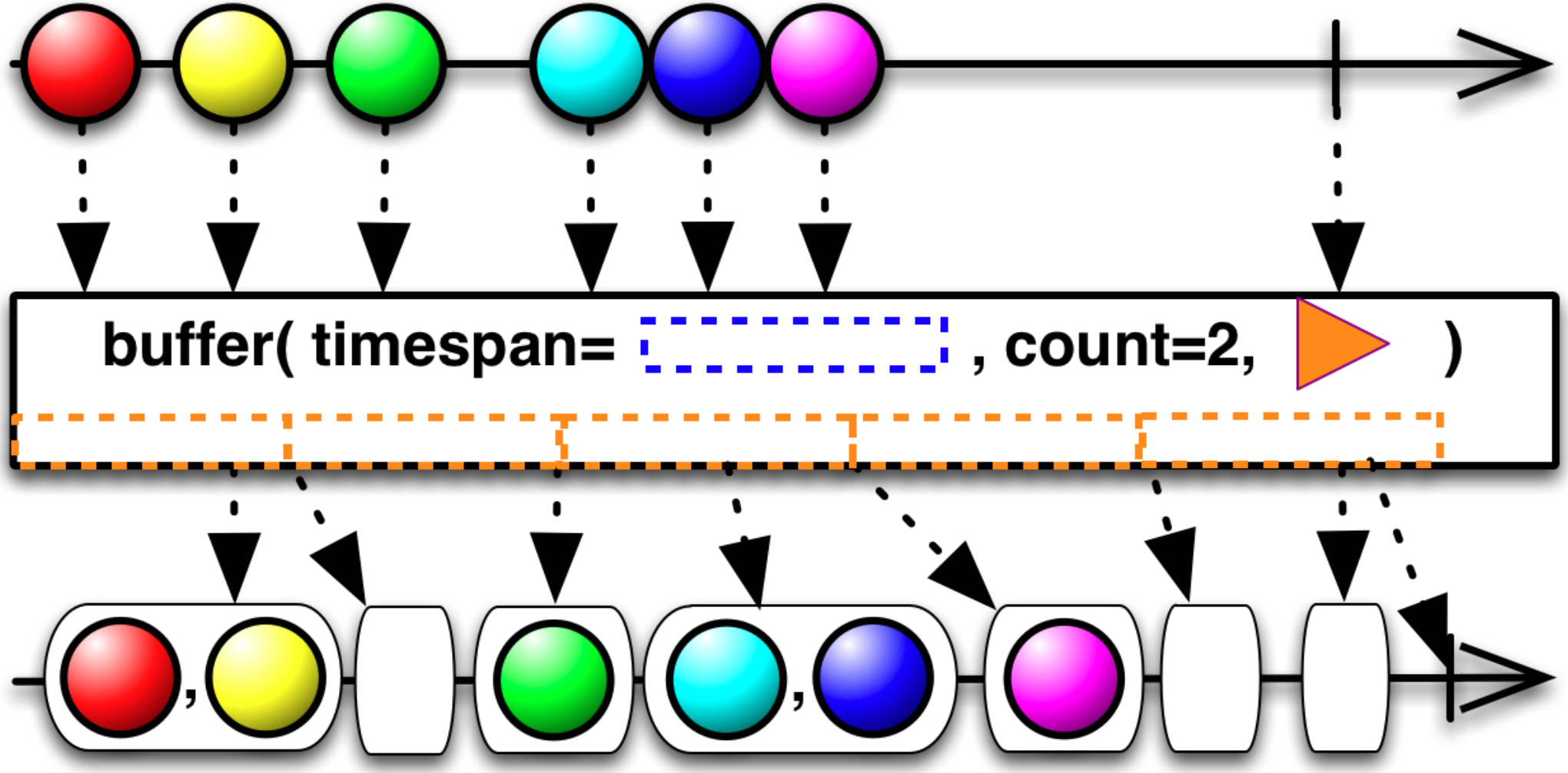

buffer(count=2, skip=3)

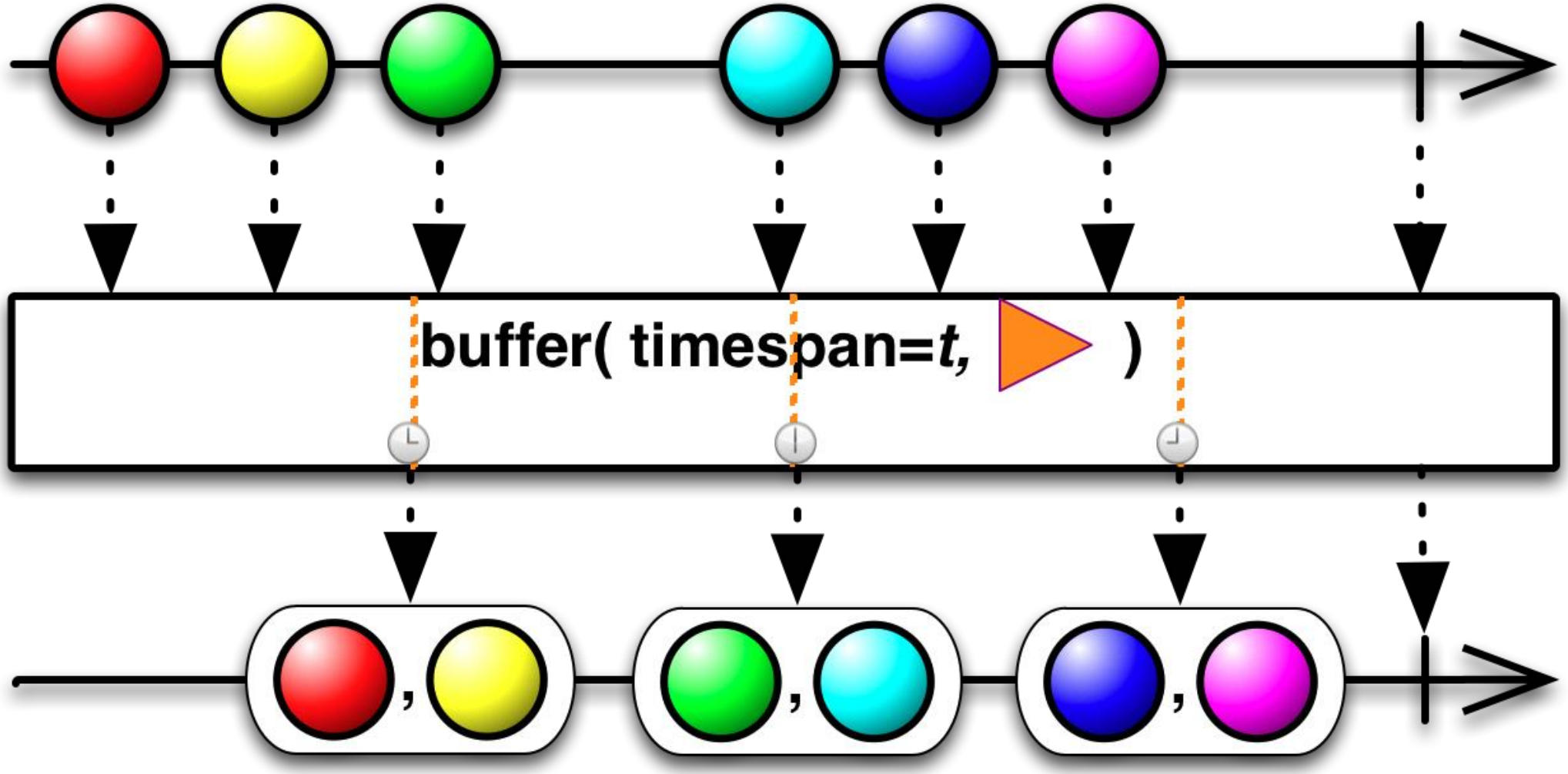

buffer(timespan, timeshift)

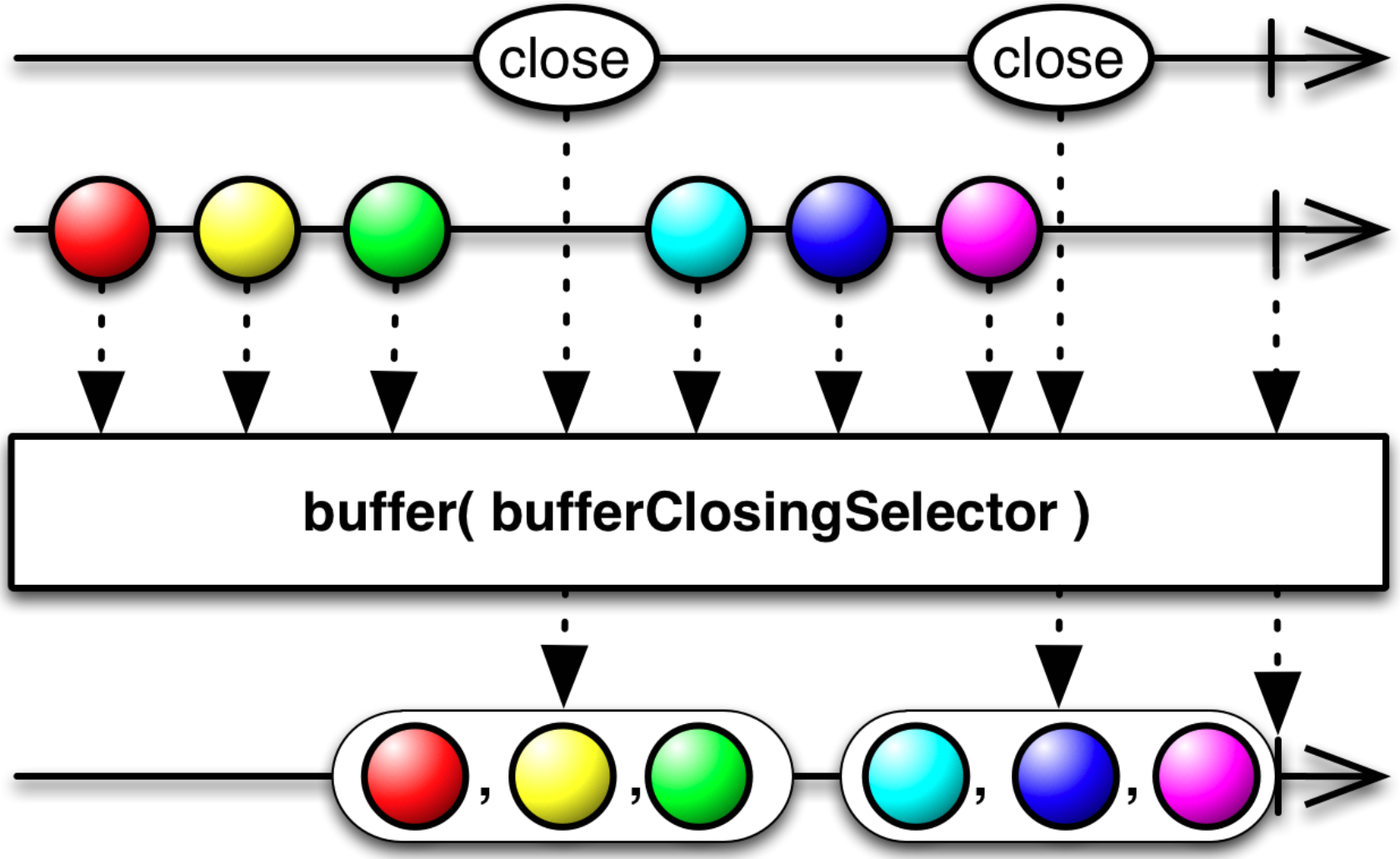

buffer(timespan, timeshift,)


buffer(timespan=t)

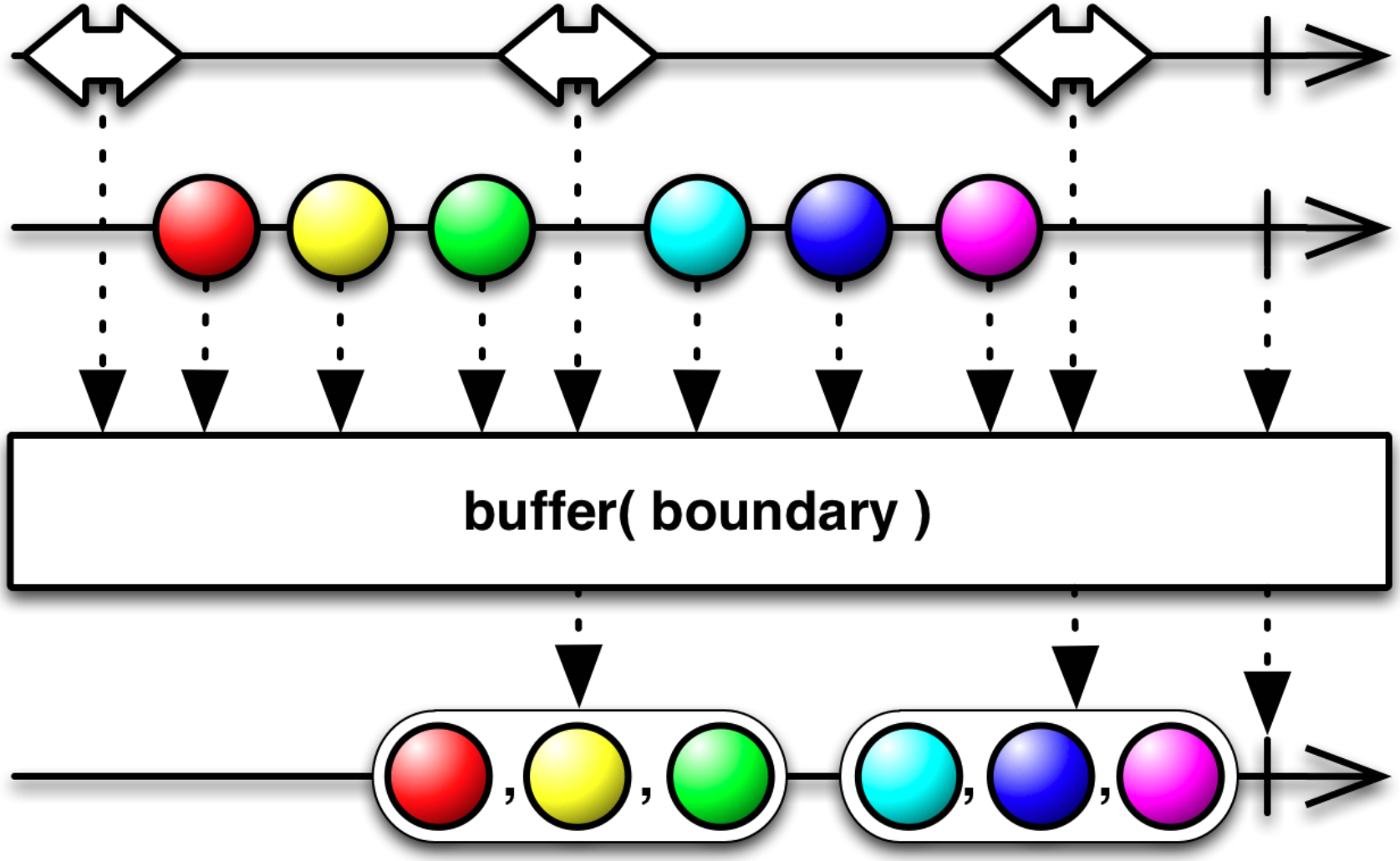

buffer(timespan= , count=2)


buffer(timespan= , count=2,)

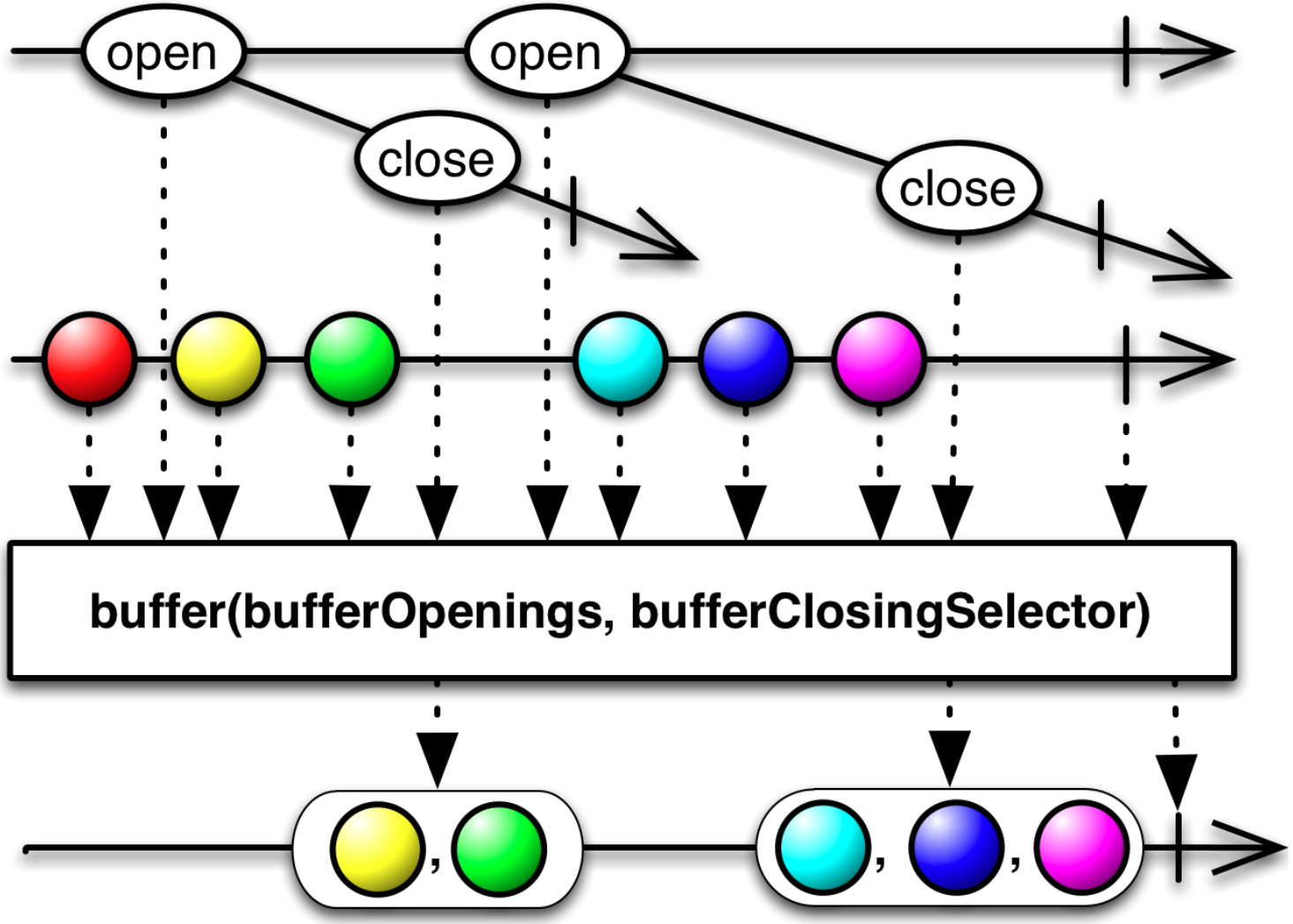

buffer(timespan=t,)

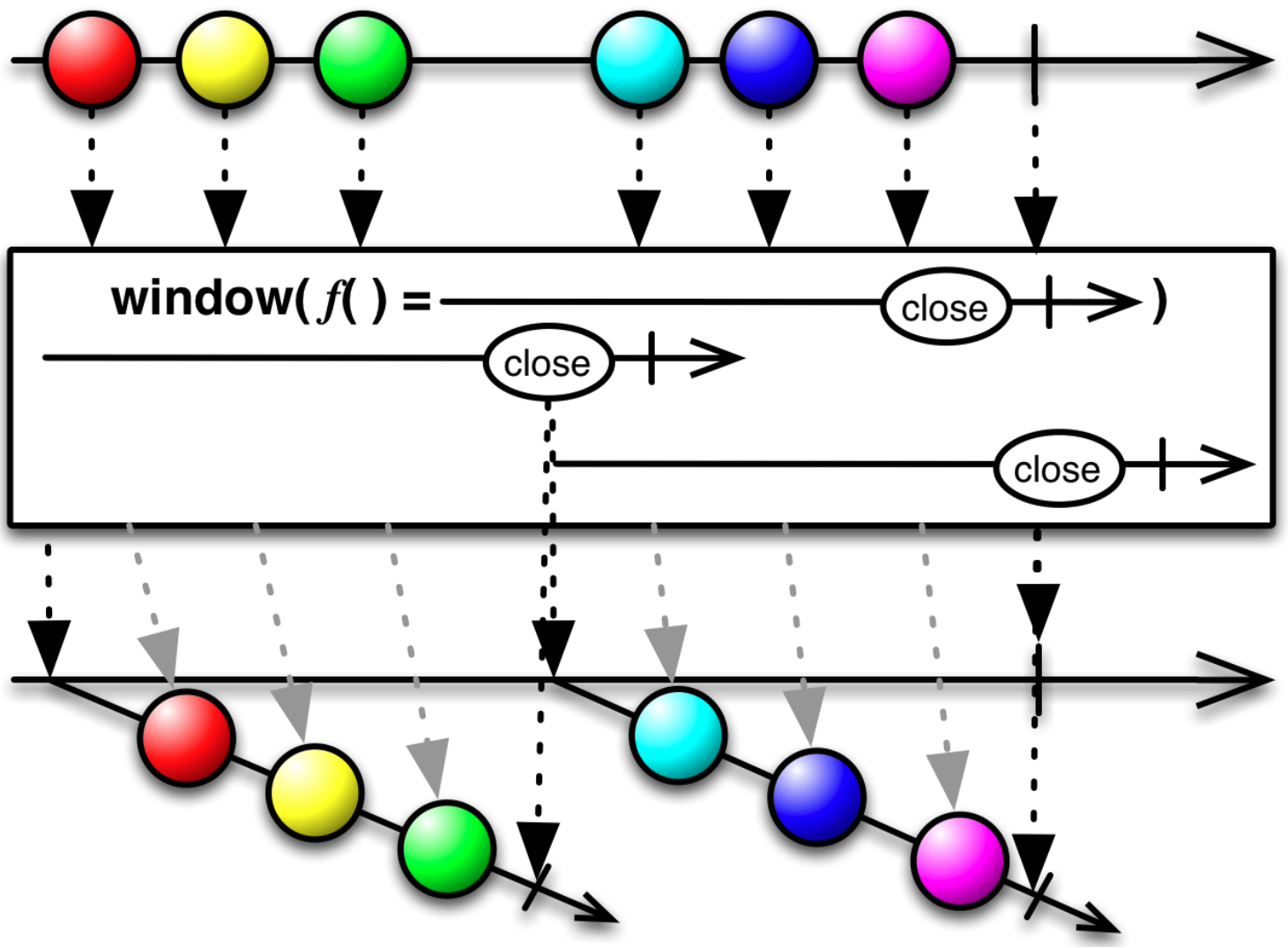


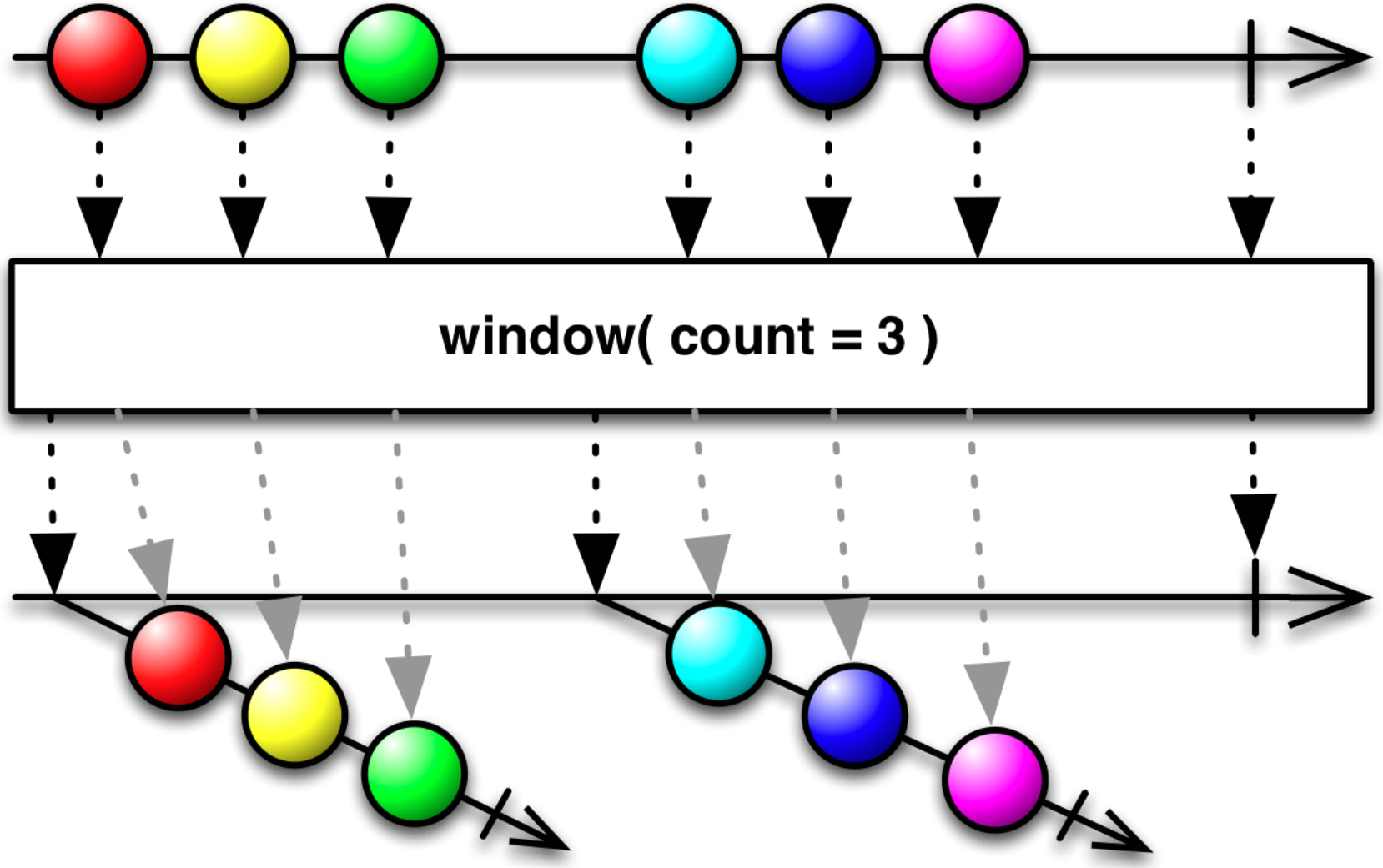
..buffer(Func0<Observable<TClosing>> bufferClosingSelector)

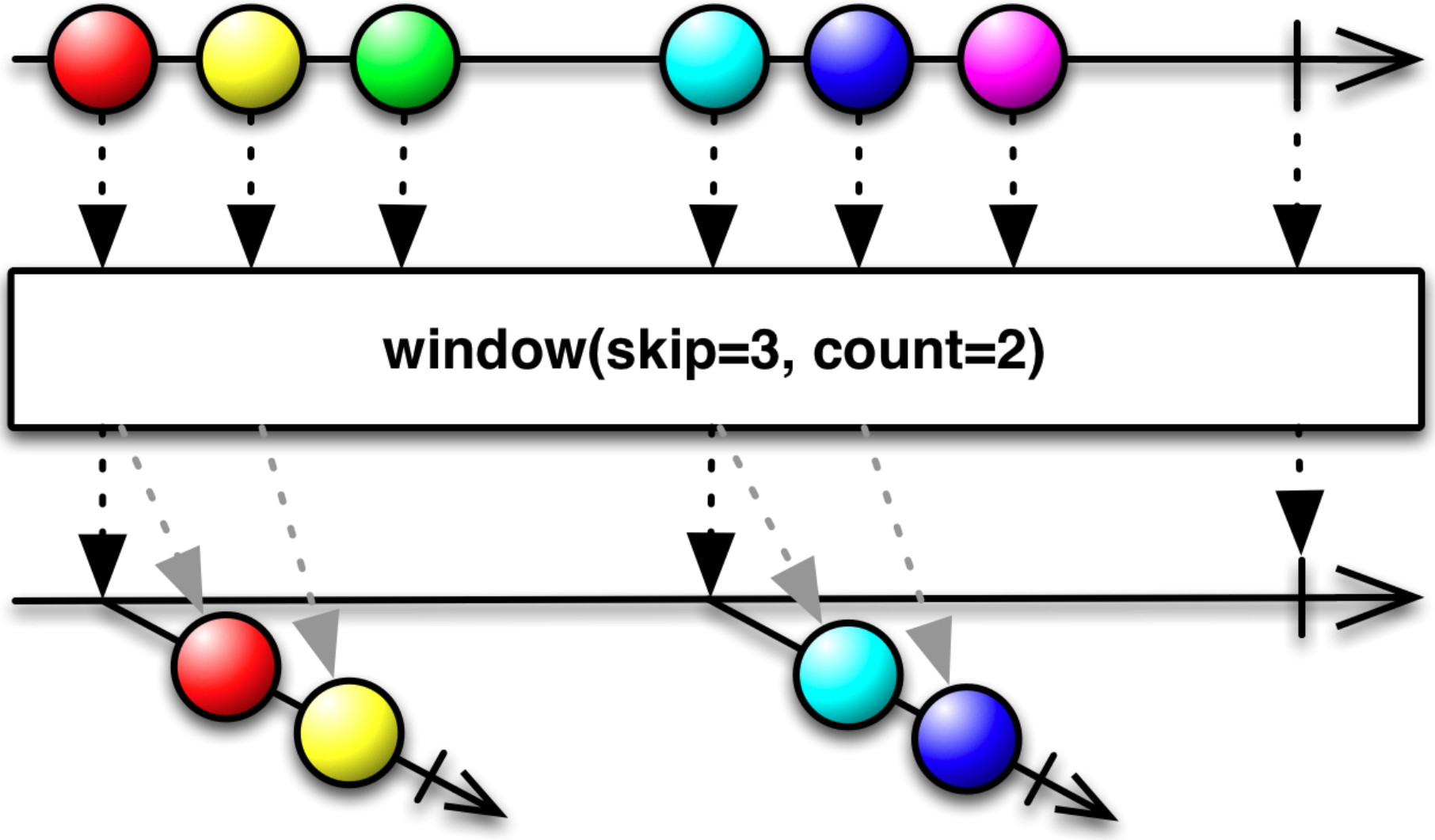


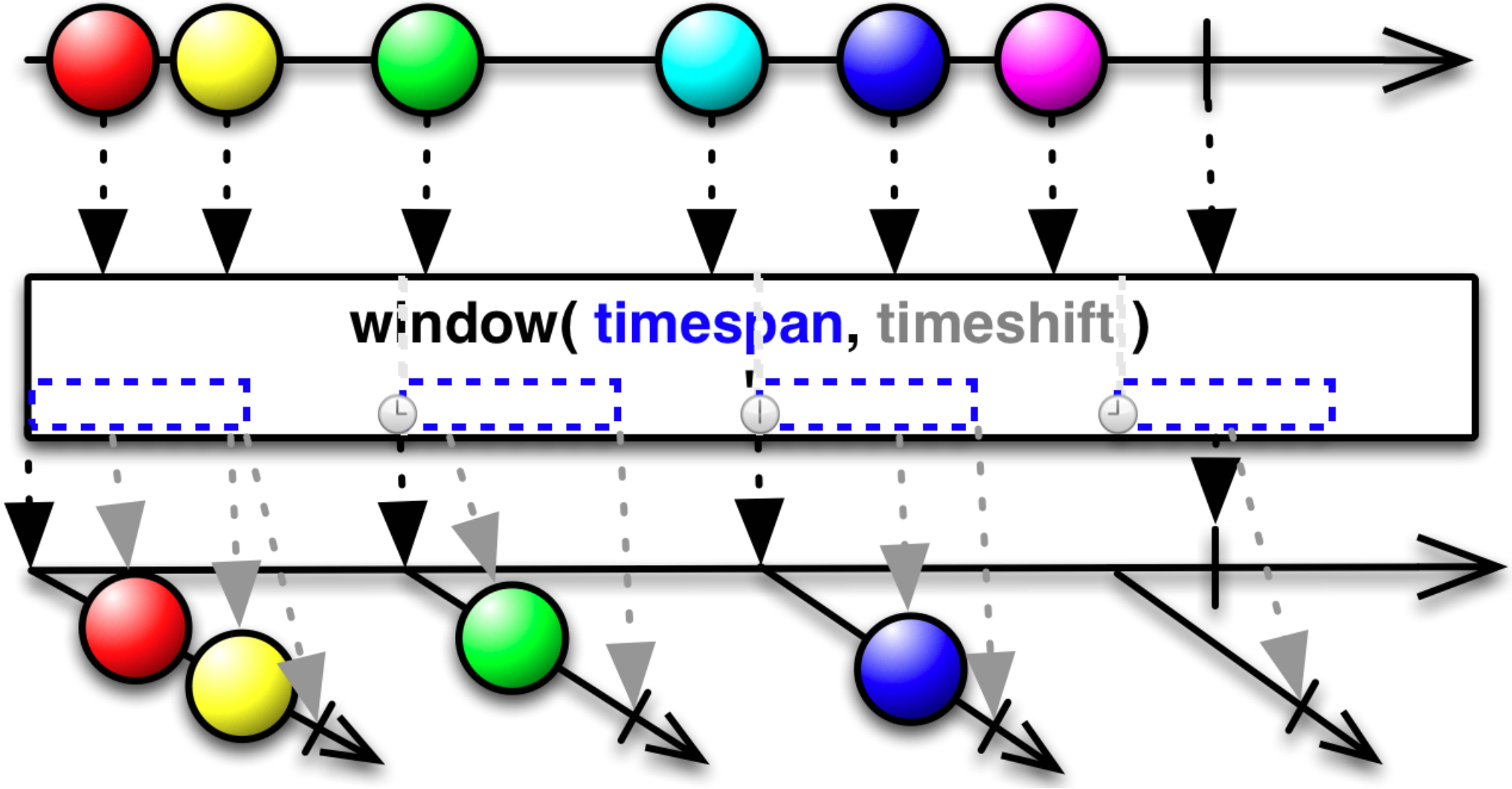
..buffer(Observable<B> boundary)

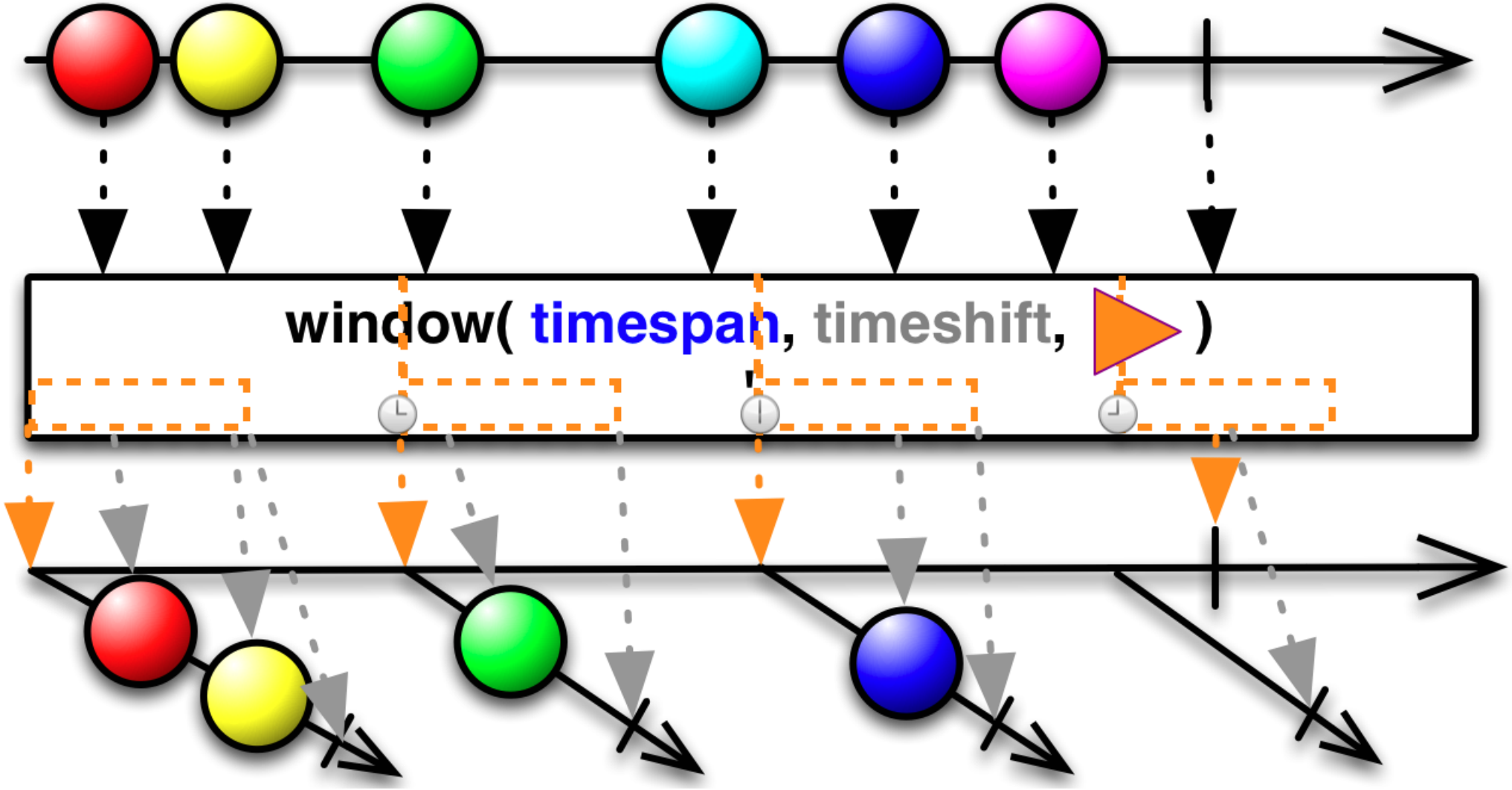

open
open
close
close
buffer(bufferOpenings, bufferClosingSelector)

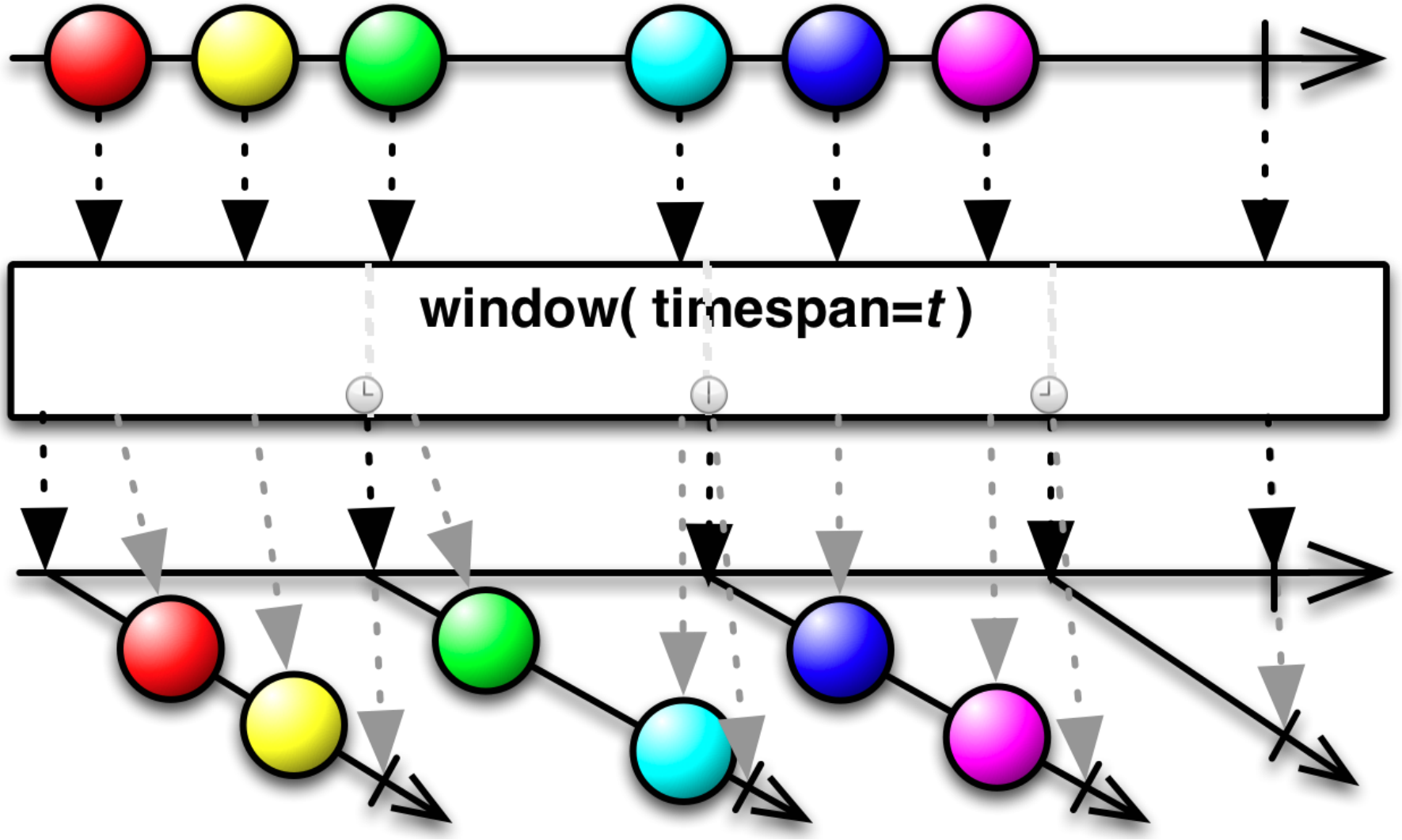

window(f( ) =
close
)
close
close

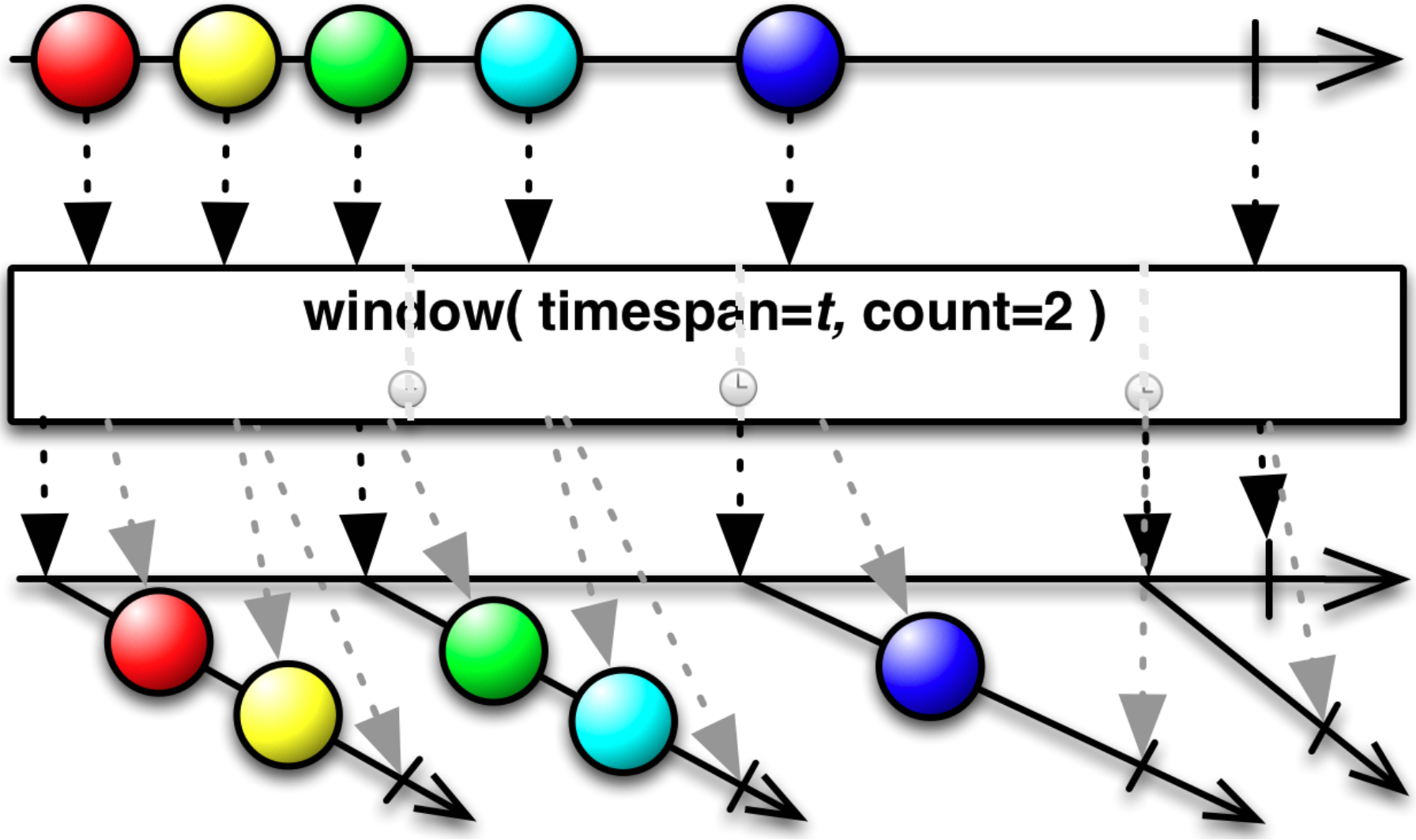

window(count = 3)

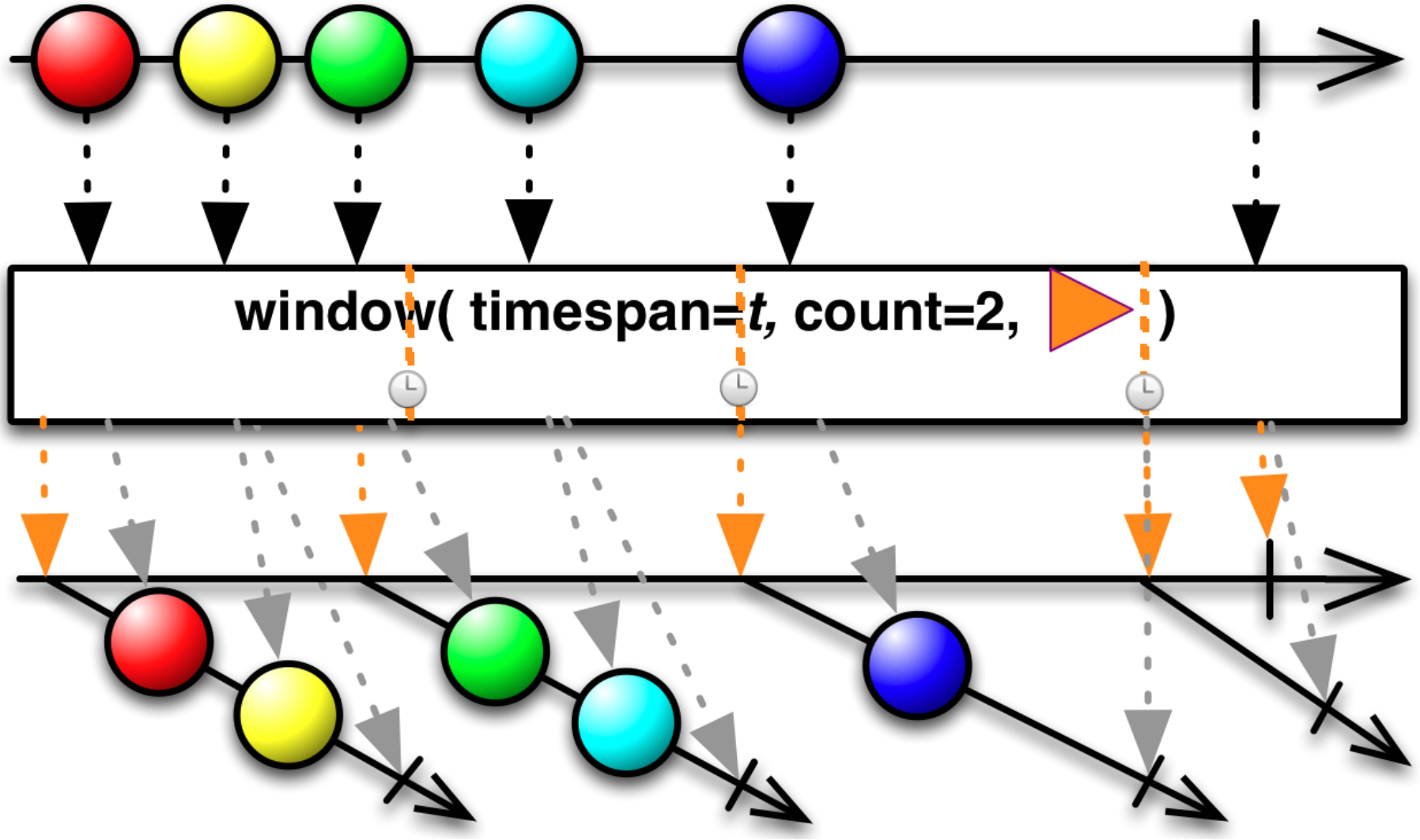

window(skip=3, count=2)

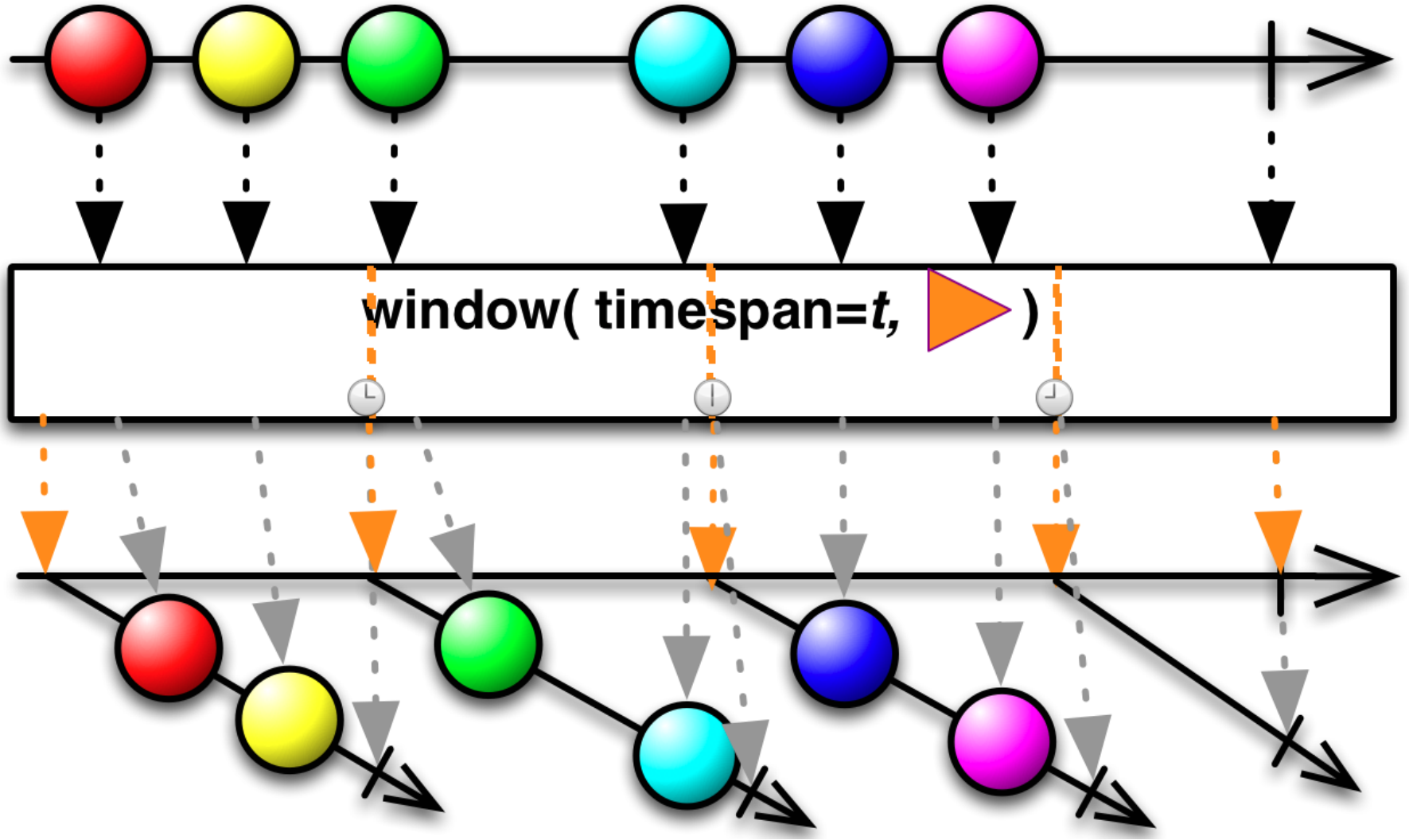

window(timespan, timeshift)

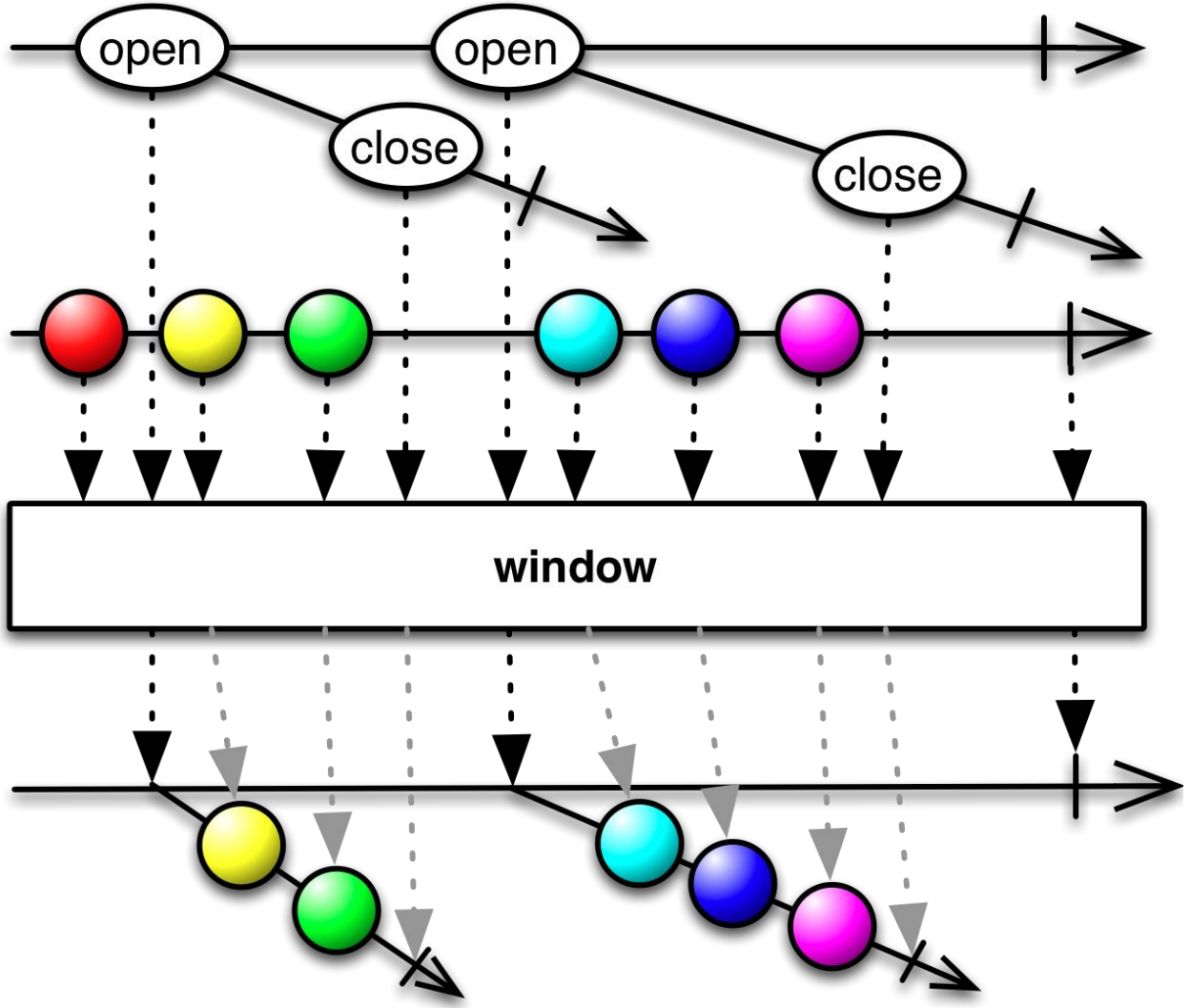

window(timespan, timeshift,)

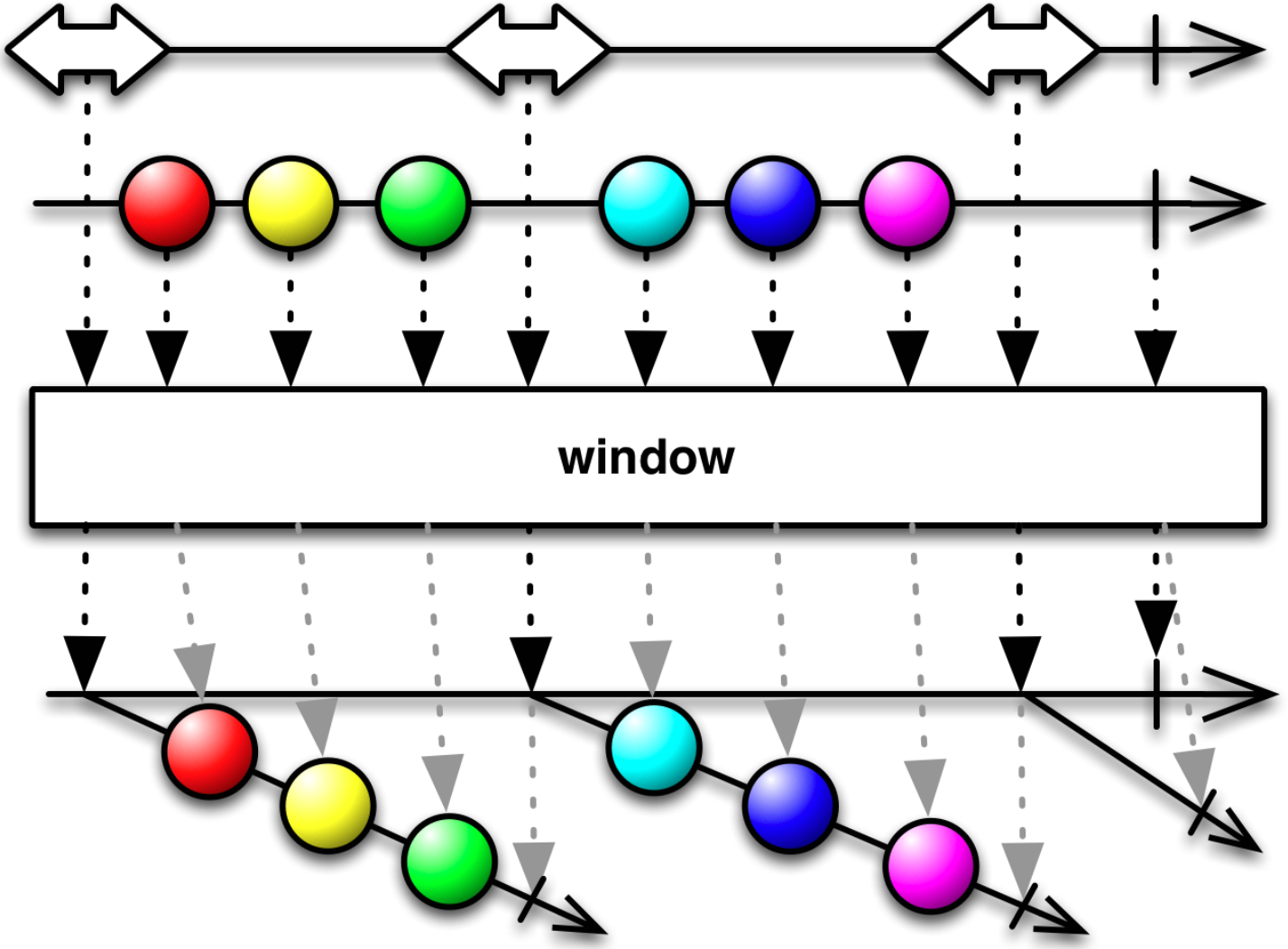

window(timespan=t)


window(timespan=t, count=2)

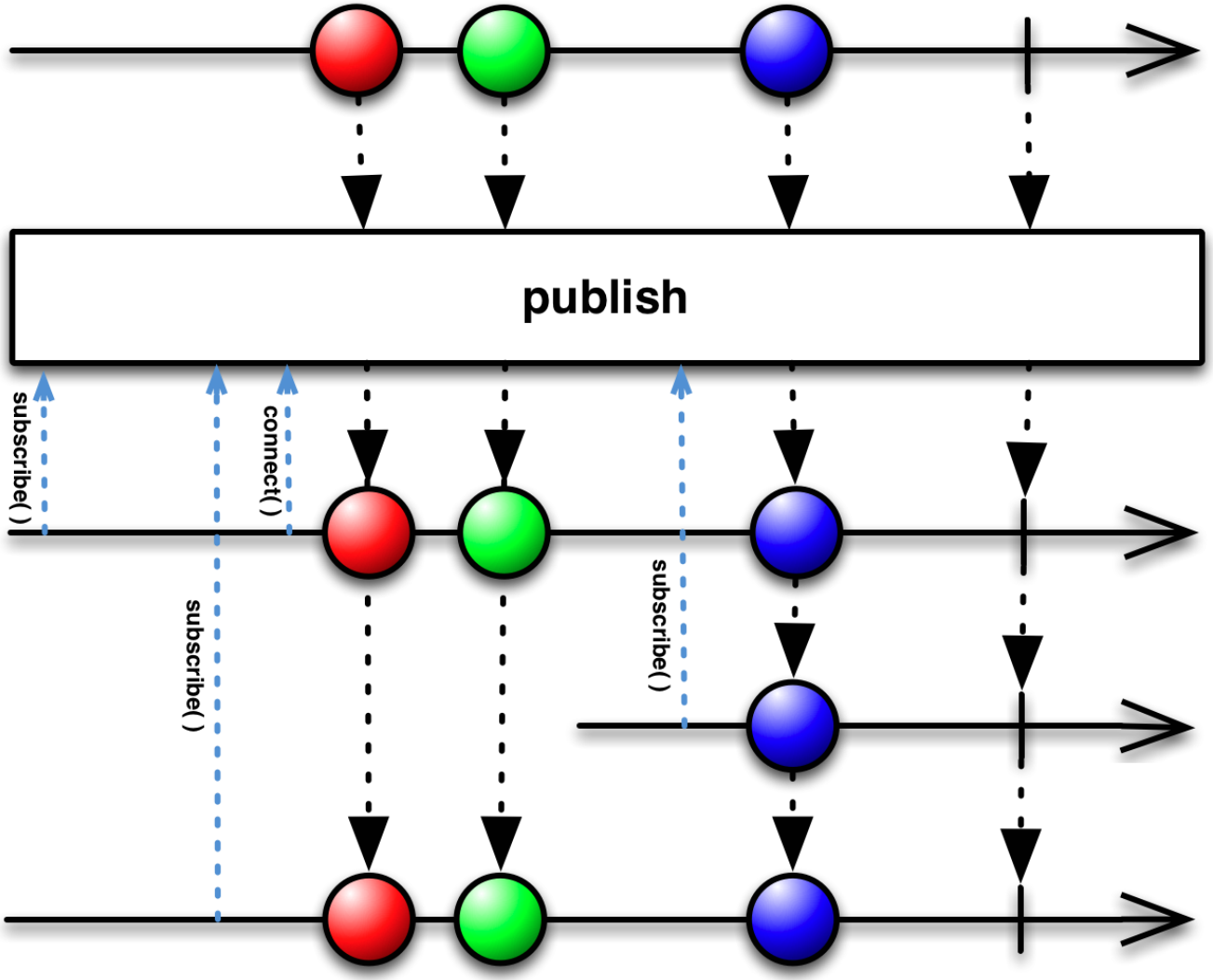

window(timespan=t, count=2,)

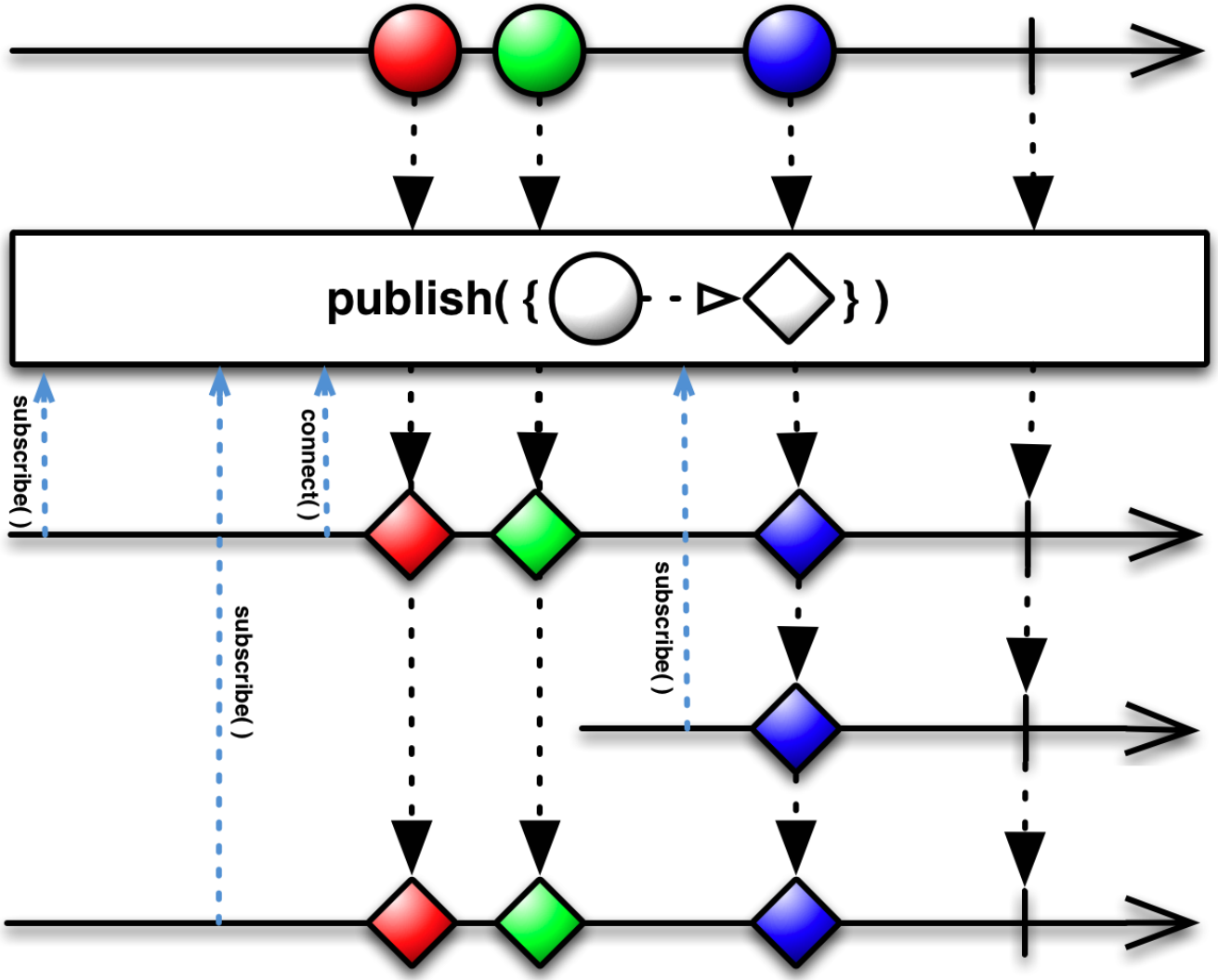

window(timespan=t,)

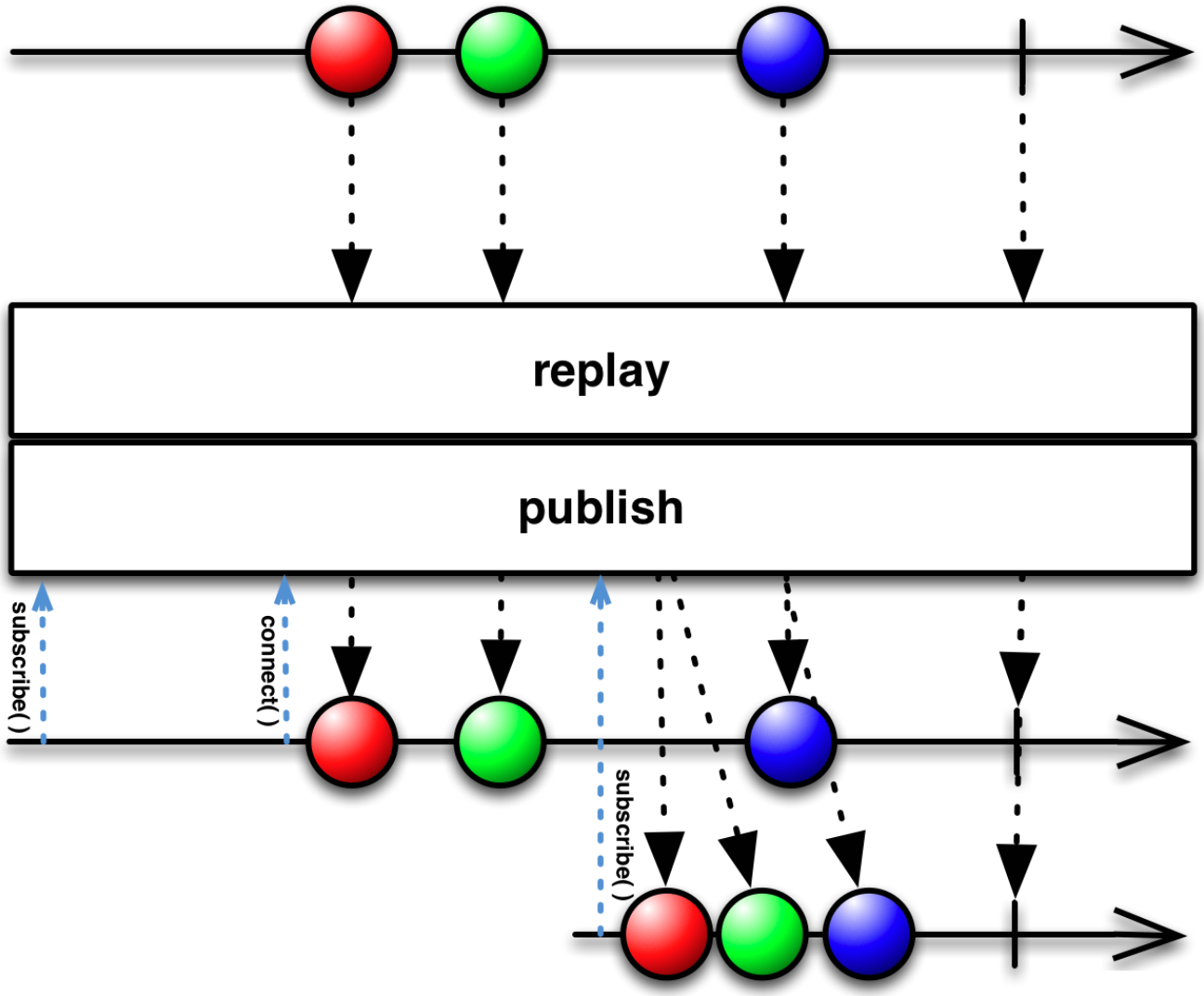

open
open
close
close
window

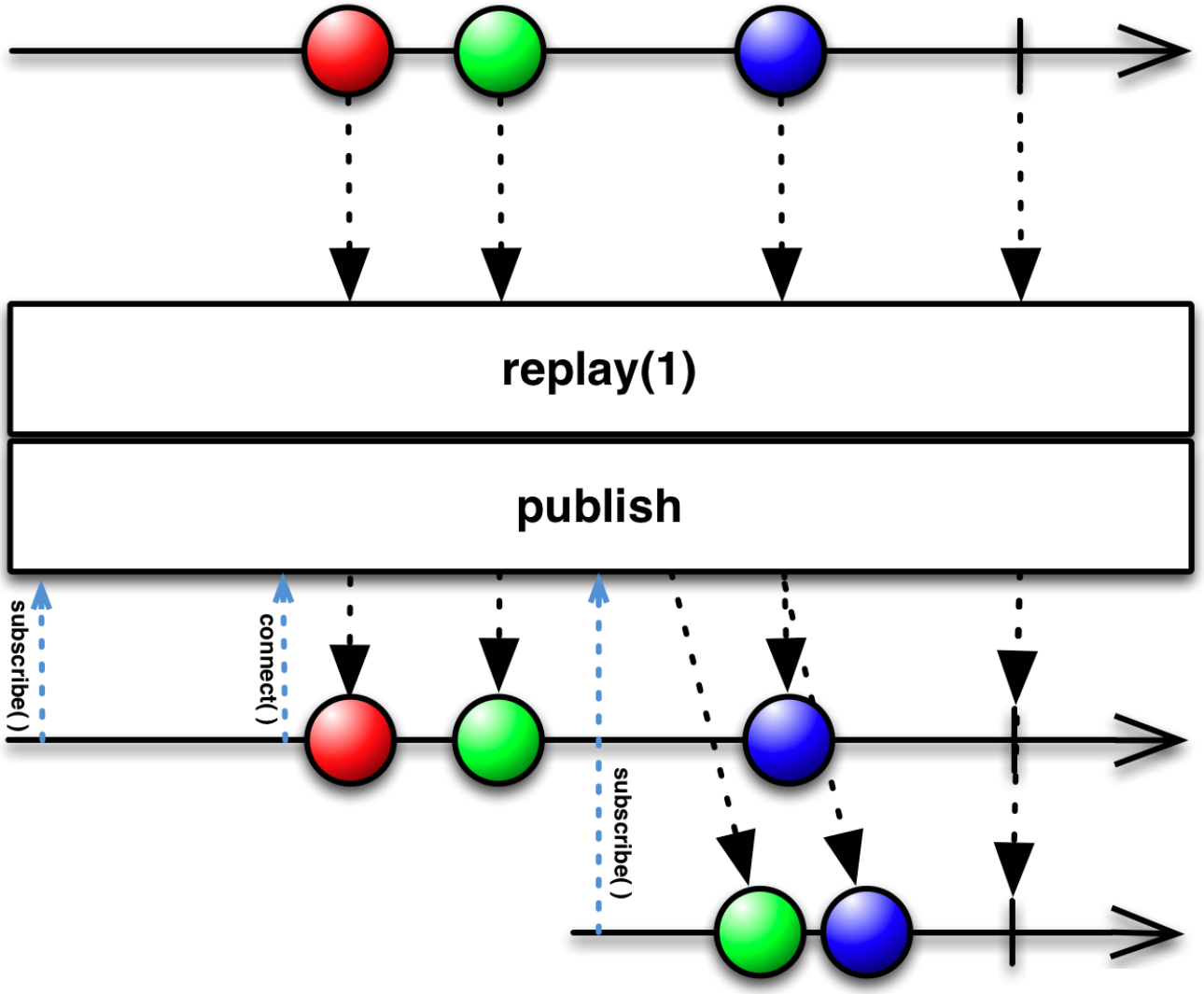

window

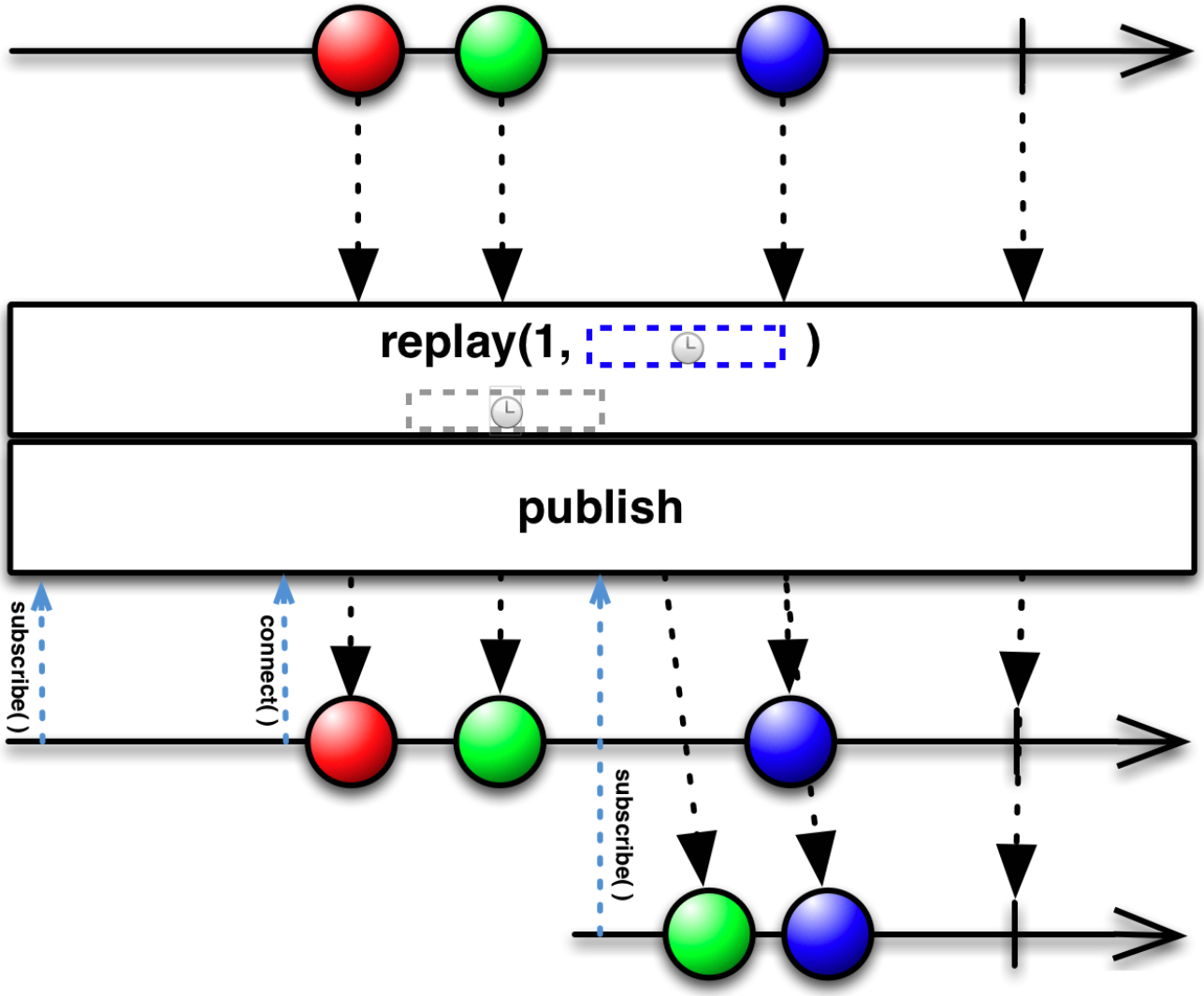
Connectable

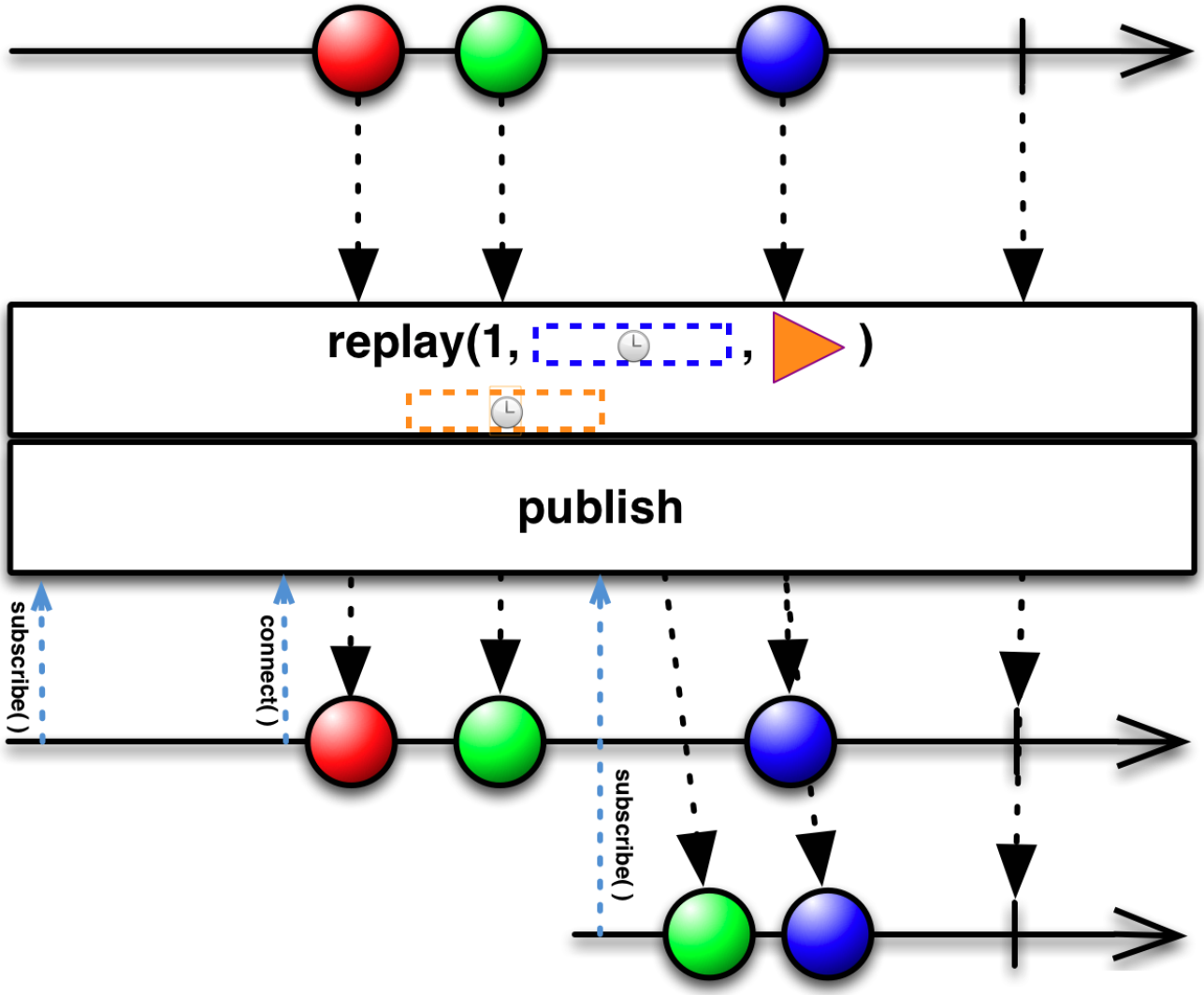

publish
subscribe( )
connect( )
subscribe( )
subscribe( )

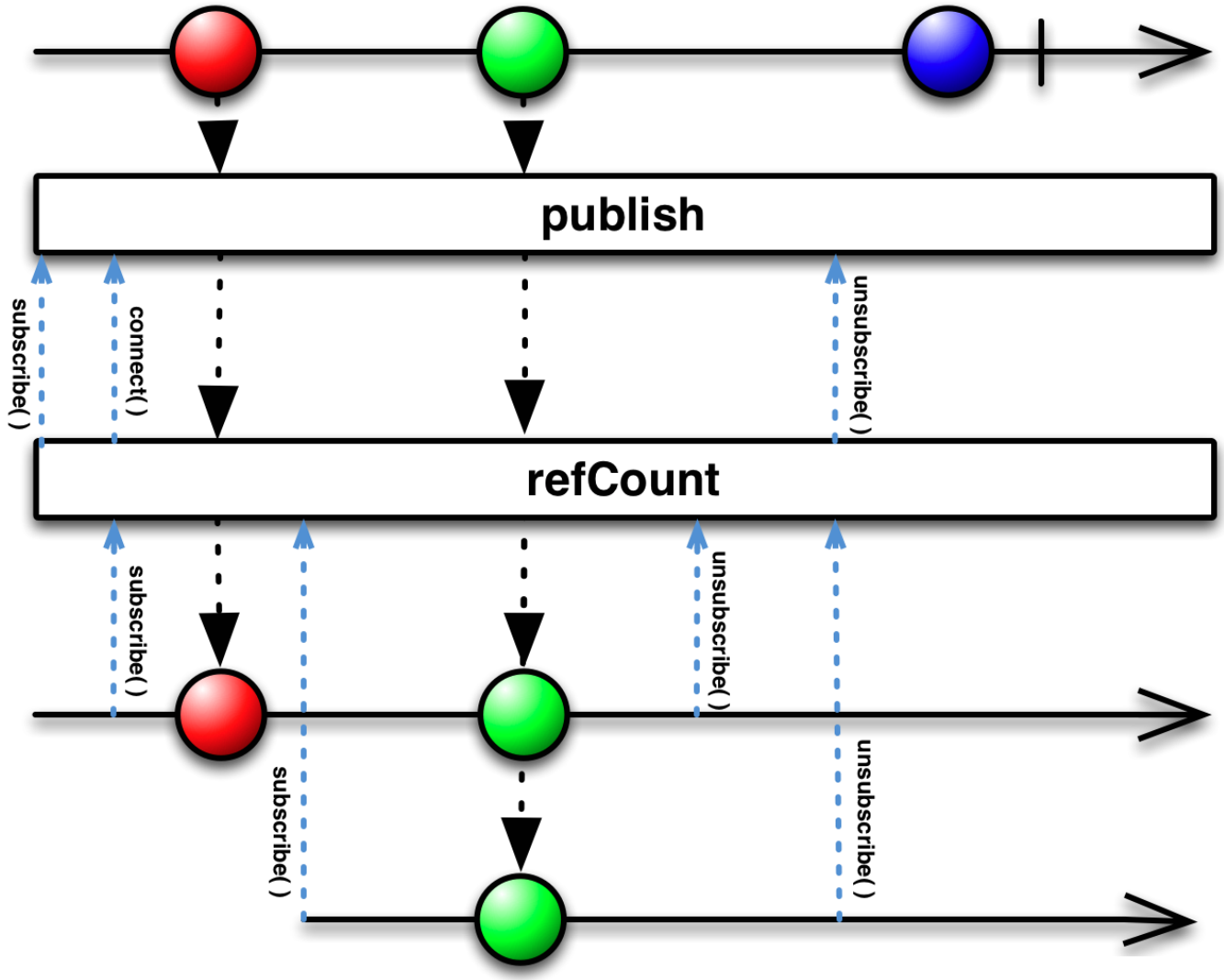

publish({ - })
subscribe( )
connect( )
subscribe( )
subscribe( )

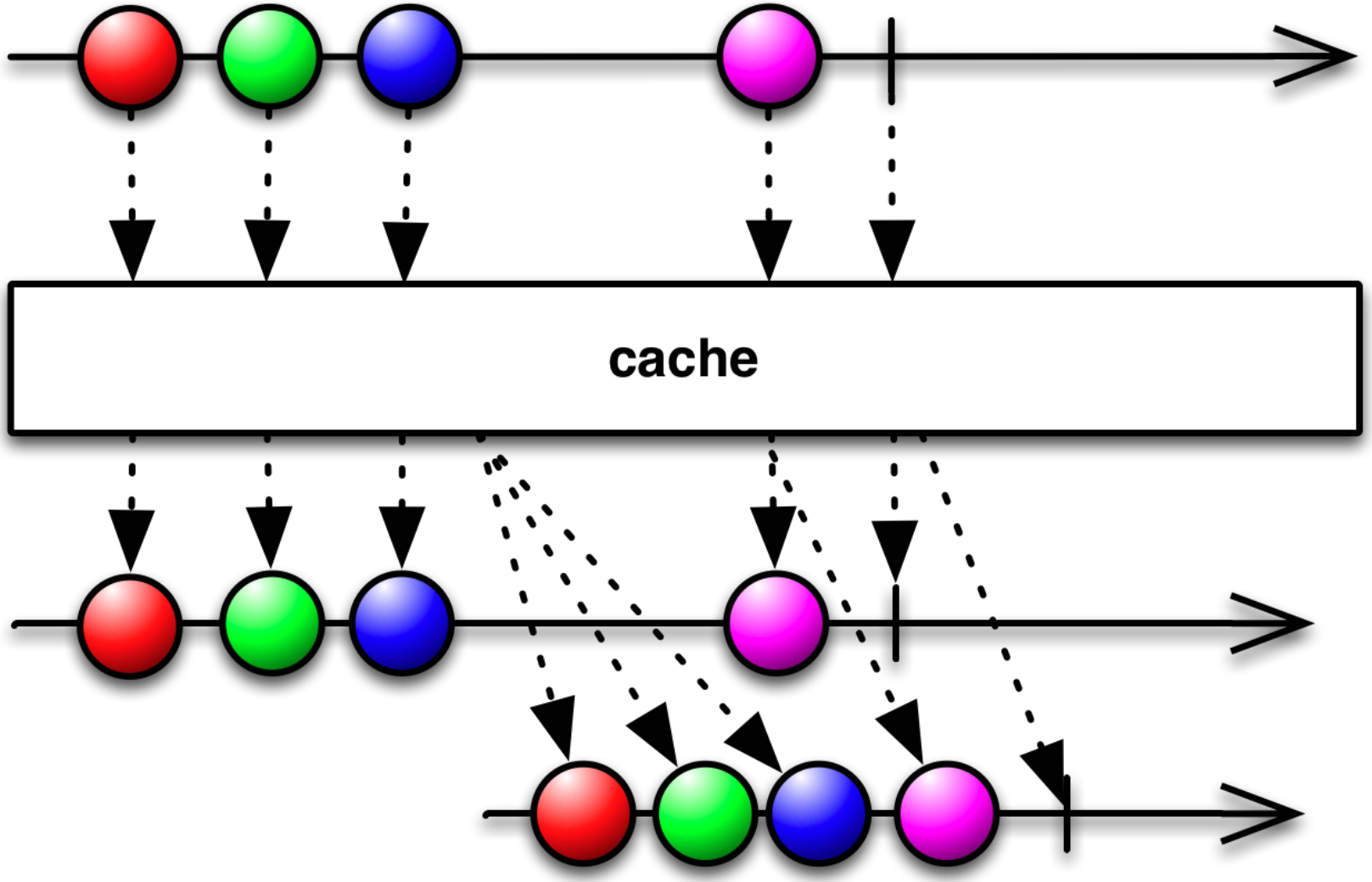

replay
publish
subscribe( )
connect( )
subscribe( )

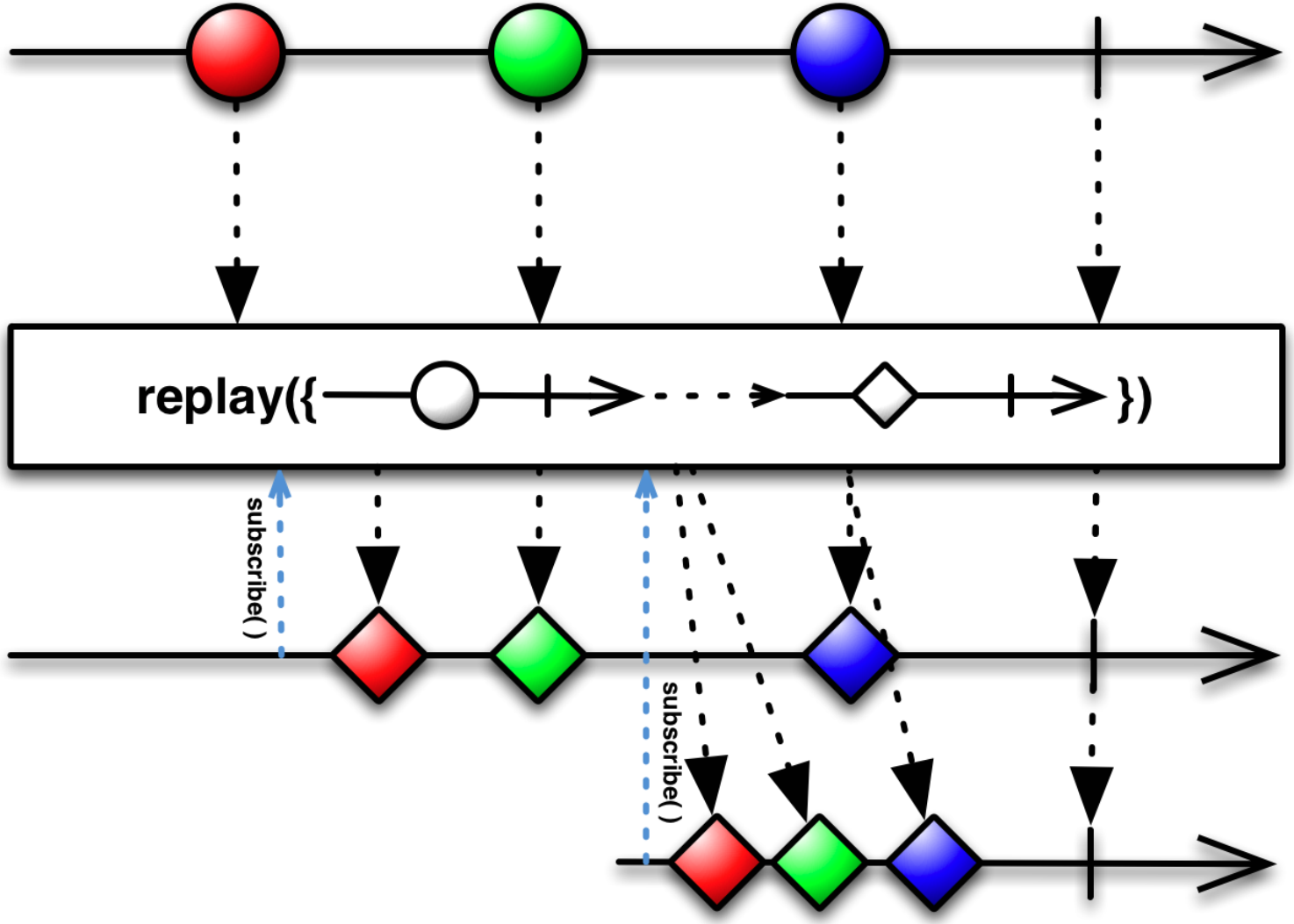

replay(1)
publish
subscribe( )
connect( )
subscribe( )

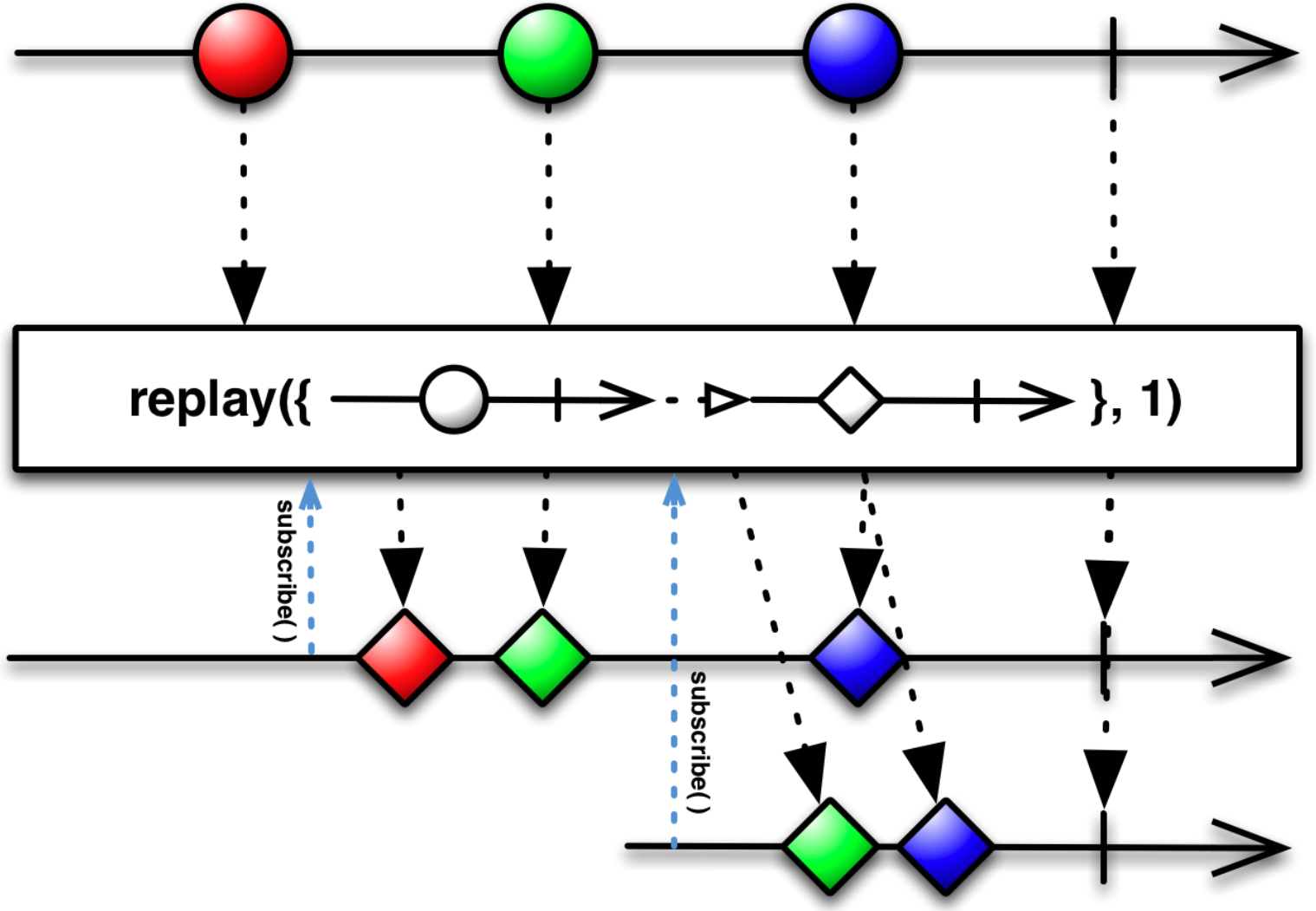

replay(1,)
publish
subscribe( )
connect( )
subscribe( )

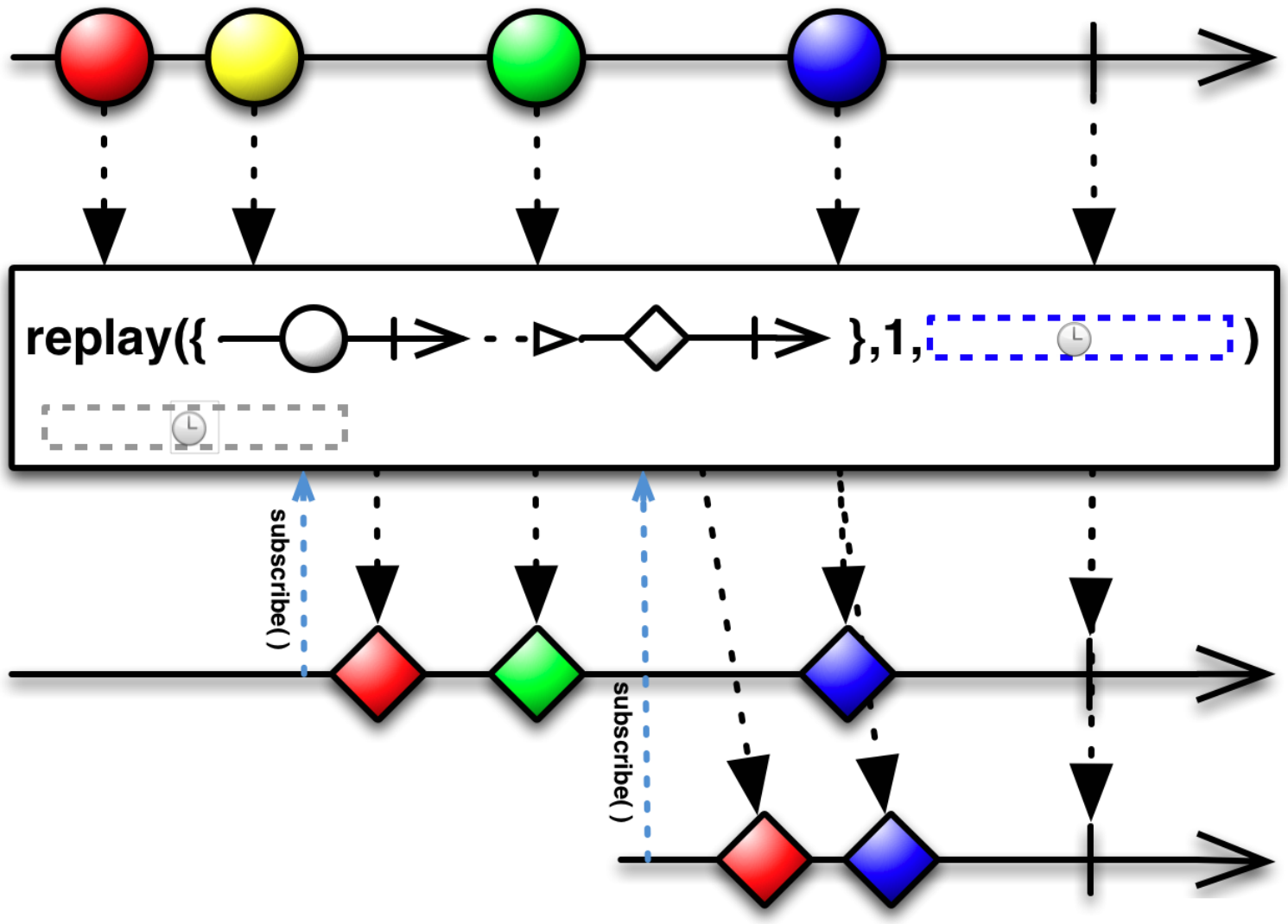

replay(1, ,)
publish
subscribe( )
connect( )
subscribe( )

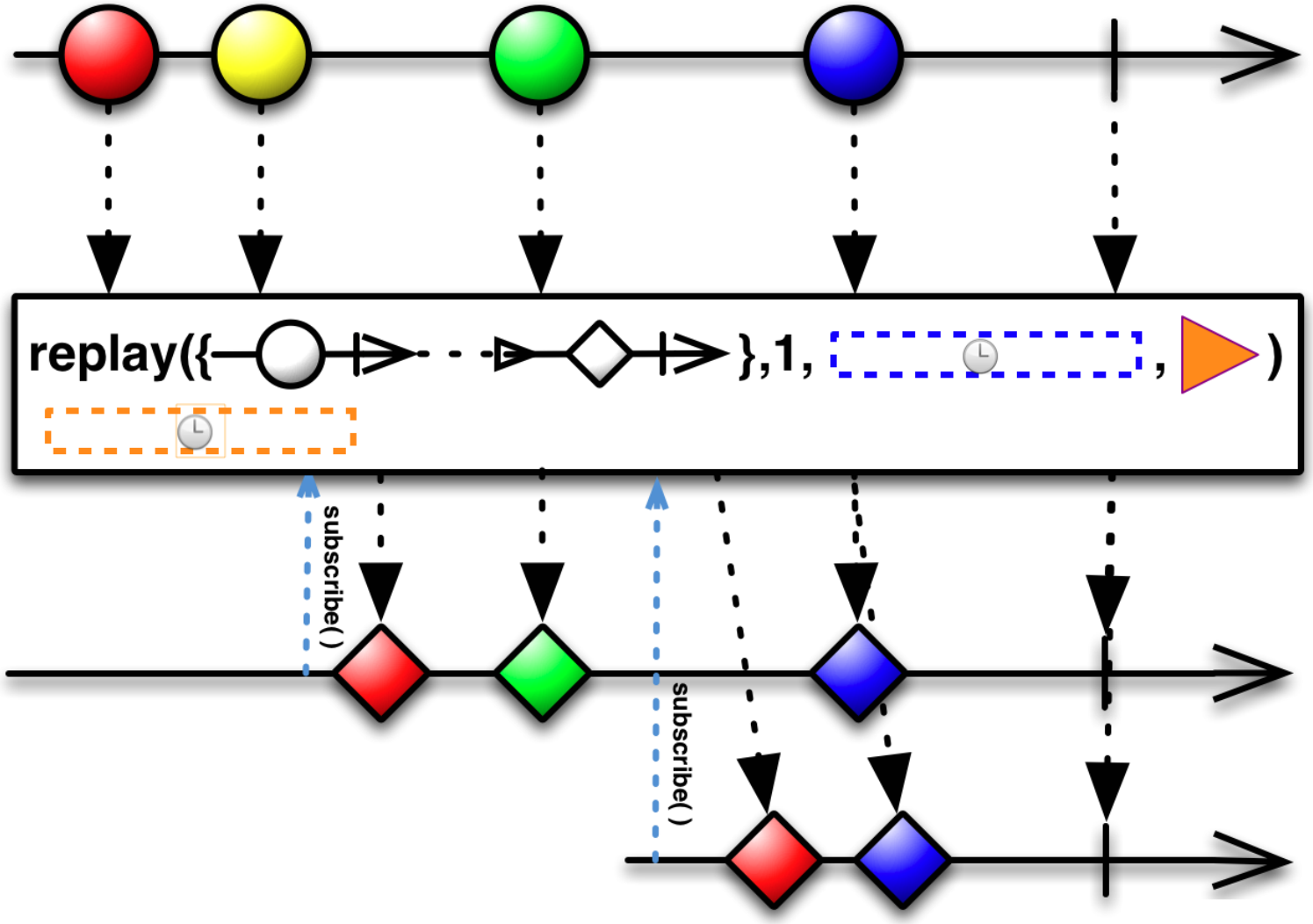

publish
subscribe( )
connect( )
unsubscribe( )
refCount
subscribe( )
unsubscribe( )
subscribe( )
unsubscribe( )

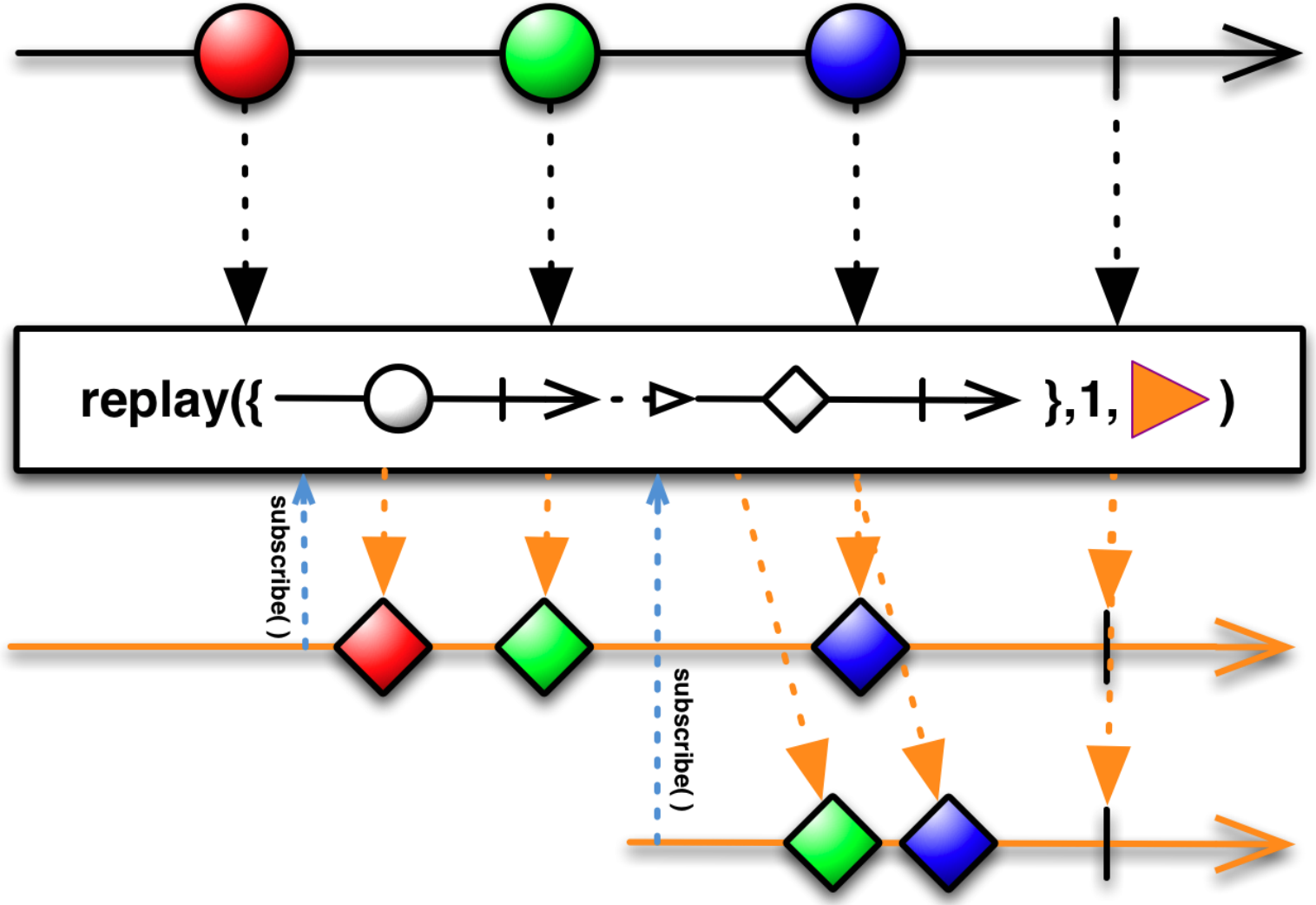

cache

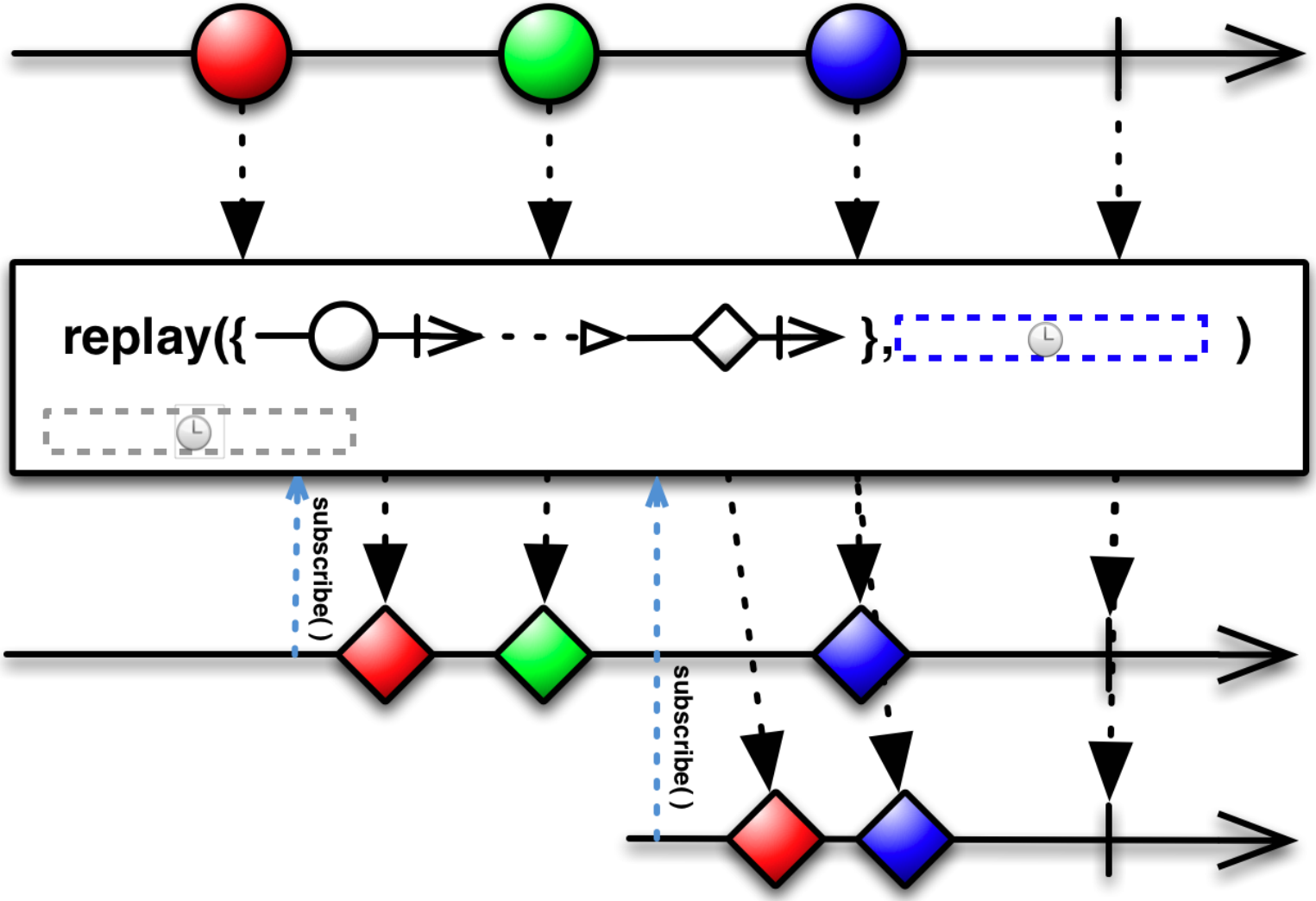

replay({
})
subscribe( )
subscribe( )

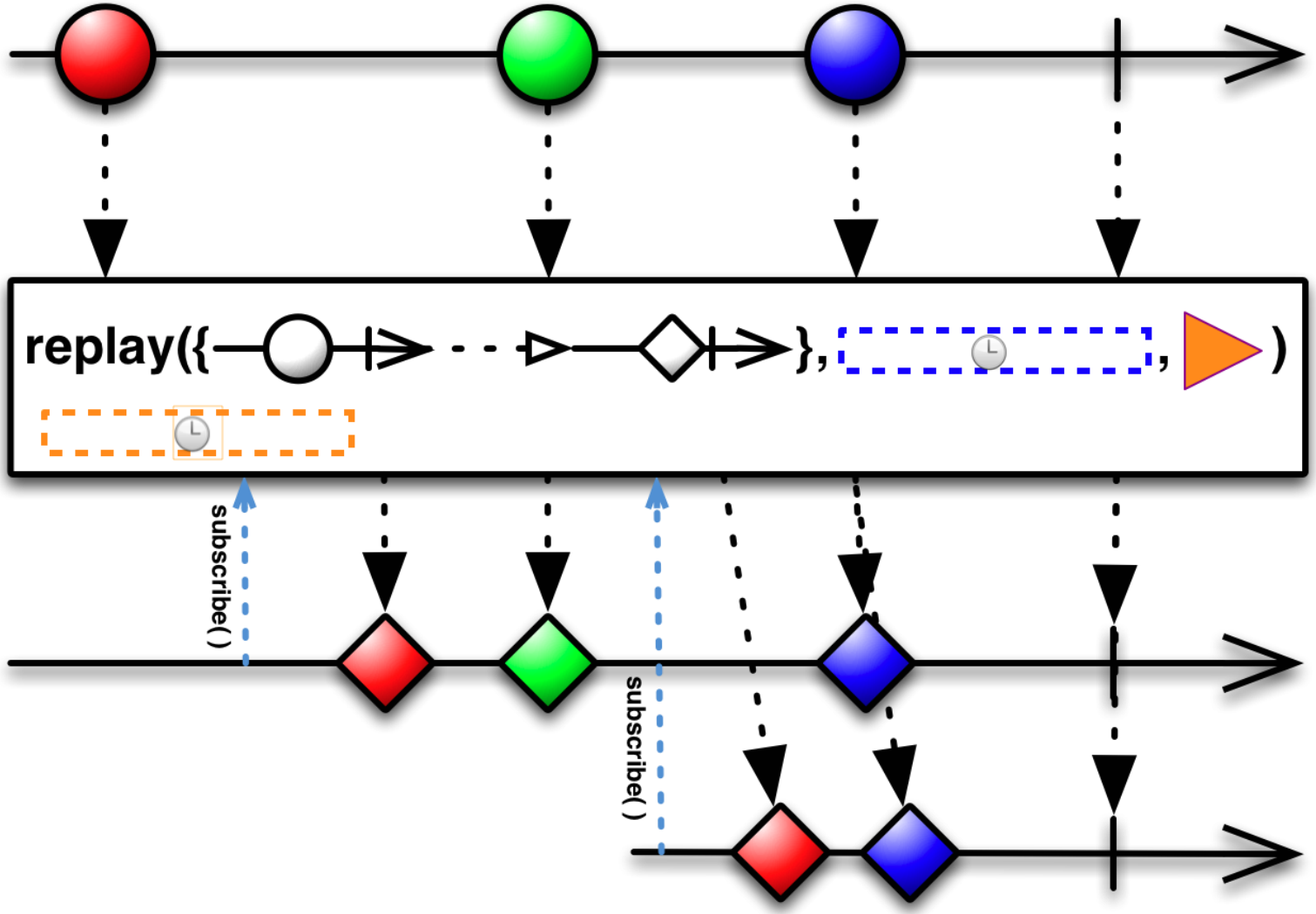

replay({
}, 1)
subscribe( )
subscribe( )

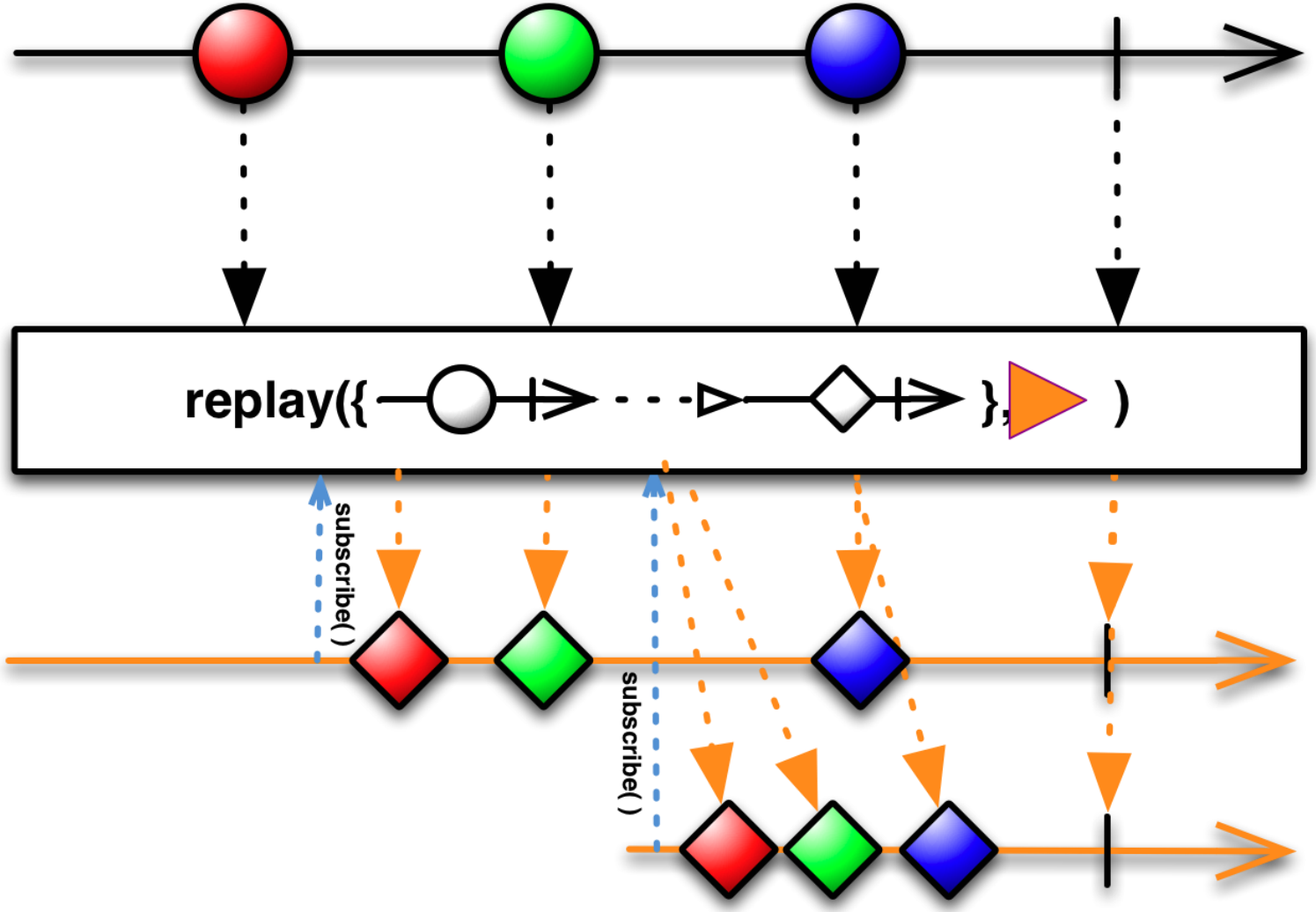

replay({
},1,
)
subscribe()
subscribe()


replay({
},1,
)
subscribe()
subscribe()


replay({
},1,
)
subscribe( )
subscribe( )


replay({
},
)
subscribe( )
subscribe( )


replay({
},
)
subscribe( )
subscribe( )


replay({
},
)
subscribe()
subscribe()