RxJava 4th Study

MASH UP 안드로이드팀 김가온

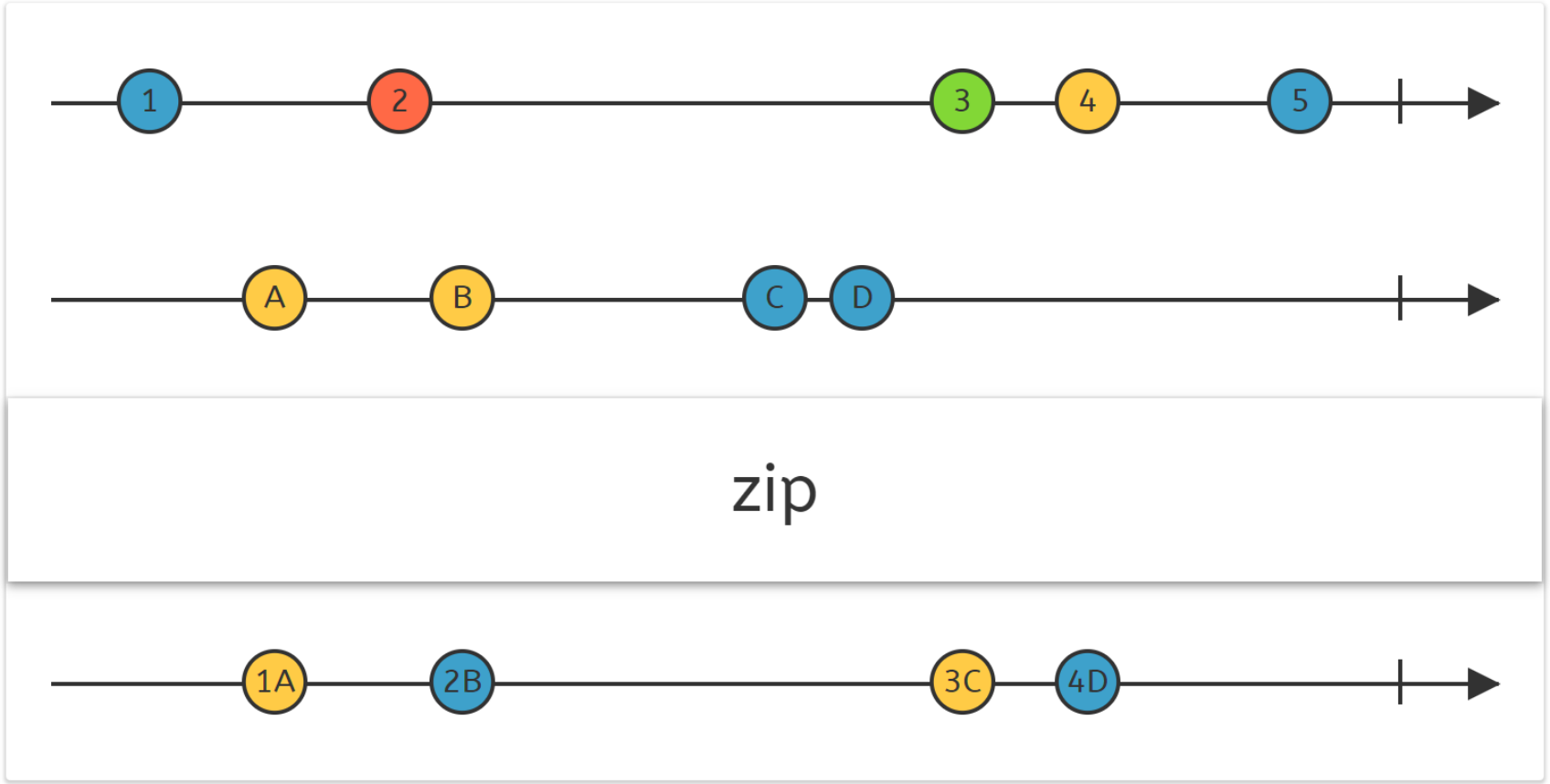
결합 연산자

생성 연산자와 변환 연산자는 1개의 Observable을 다룸
결합 연산자는 여러 개의 Observable을 조합하여 활용

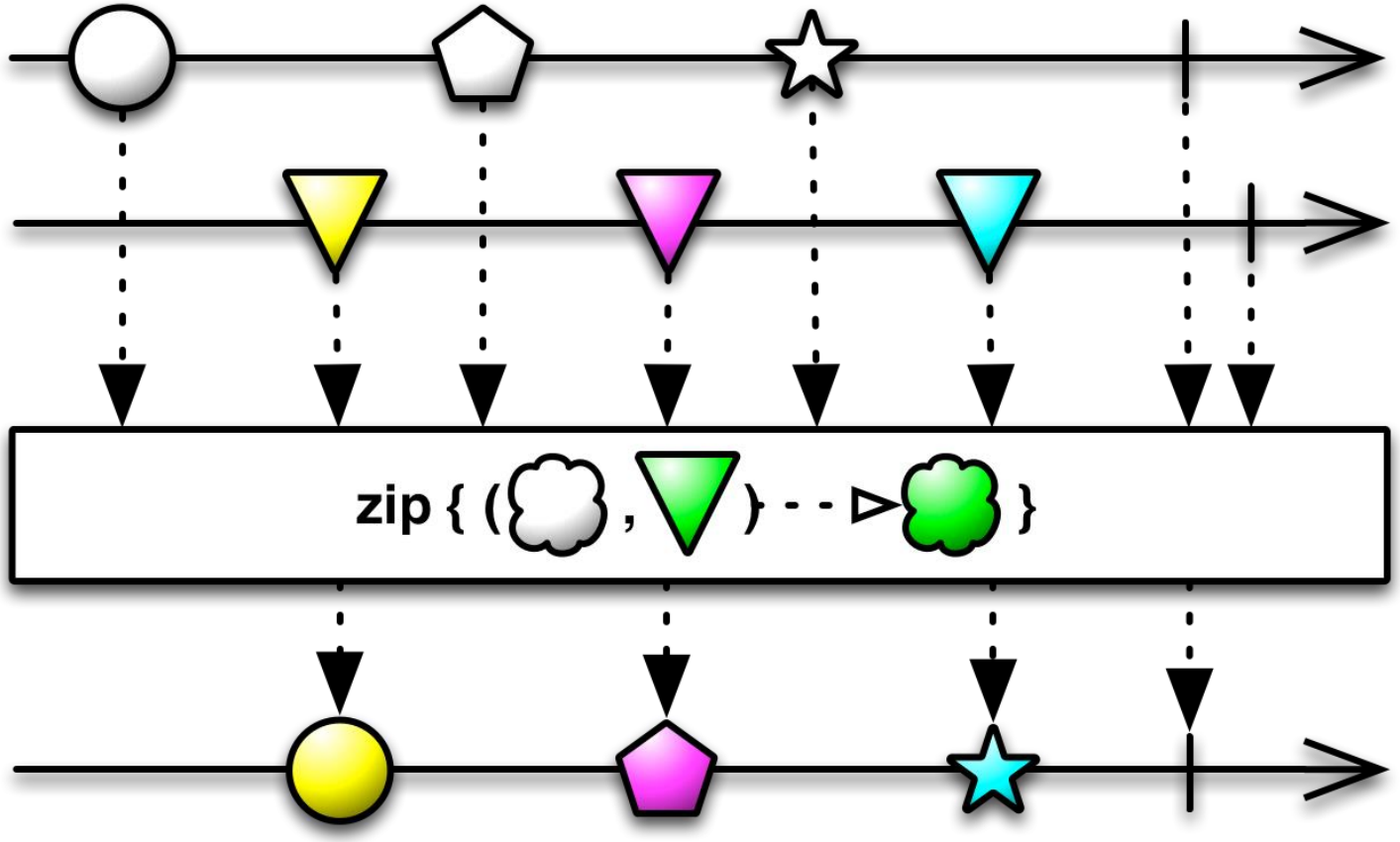
- zip()
- combineLatest()
- concat()

zip()

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



zip()



zip()

최대 9개의 Observable 결합 가능

```
@SchedulerSupport(SchedulerSupport.NONE)
public static <T1, T2, R> Observable<R> zip(
    ObservableSource<? extends T1> source1,
    ObservableSource<? extends T2> source2,
    BiFunction<? super T1, ? super T2, ? extends R> zipper)
```

zip()

```
7           String[] colors = {"RED", "GREEN", "YELLOW"};
8           String[] shapes = {"BALL", "PENTAGON", "STAR"};
9
10          Observable<String> source = Observable.zip(
11              Observable.fromArray(colors),
12              Observable.fromArray(shapes),
13              (color, shape) -> color + "-" + shape);
14
15          source.subscribe(System.out::println);
```

zip() – 숫자 결합

```
7           Observable<Integer> source = Observable.zip(
8                   Observable.just(100, 200, 300),
9                   Observable.just(10, 20, 30),
10                  Observable.just(1, 2, 3),
11                  (a, b, c) -> a + b + c);
12
13          source.subscribe(System.out::println);
```

zip() – 시간 결합

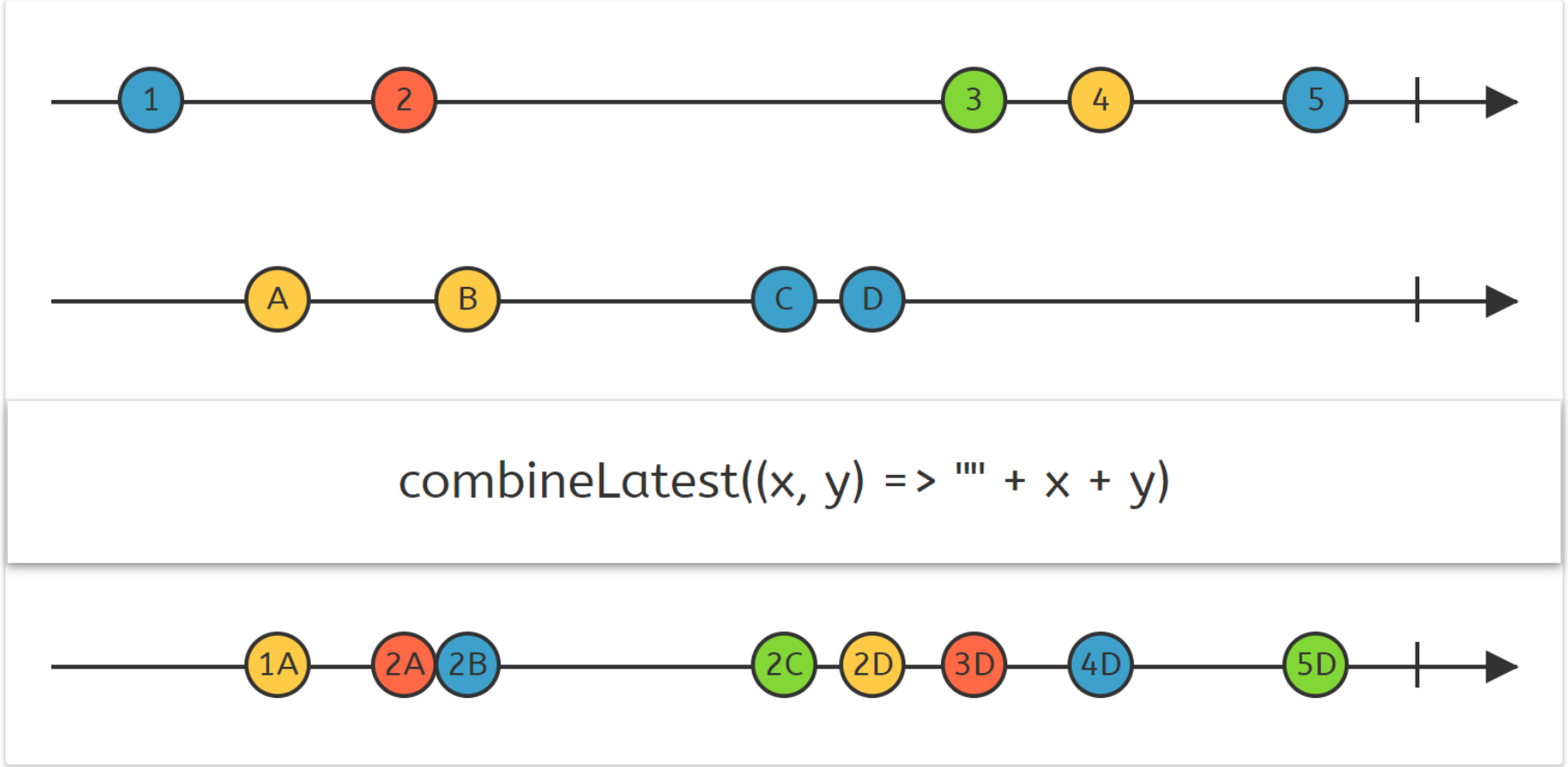
```
7   public static void main(String[] args) {
8
9       Observable<String> source = Observable.zip(
10          Observable.just("RED", "GREEN", "BLUE"),
11          Observable.interval( period: 200L, TimeUnit.MILLISECONDS),
12          (value, i) -> value);
13
14      long startTime = System.currentTimeMillis();
15
16      source.subscribe(value -> {
17          long time = System.currentTimeMillis() - startTime;
18          System.out.println(time + " | " + "value = " + value);
19      });
20
21      CommonUtils.sleep( millis: 1000);
```

zipWith()

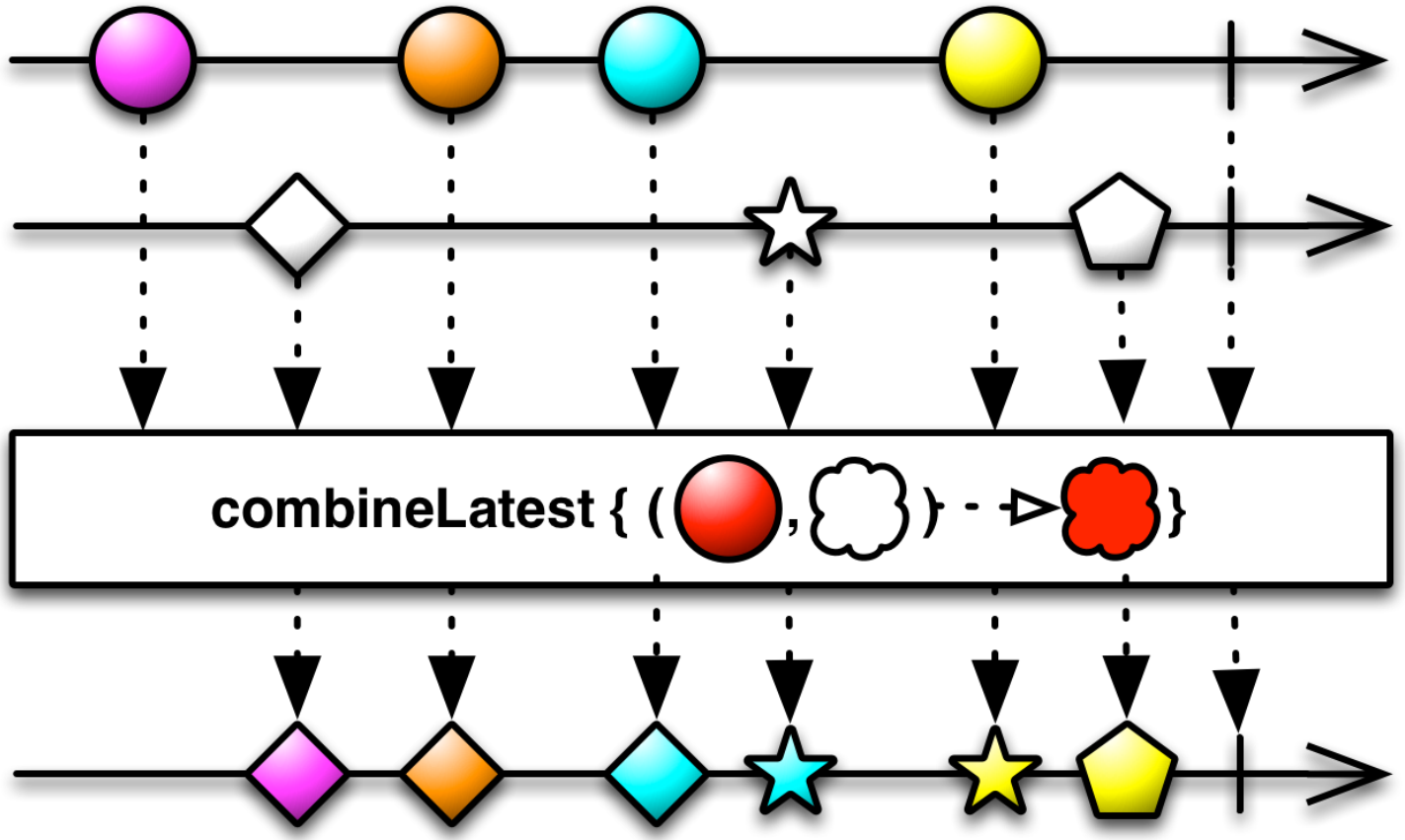
```
7           Observable<Integer> source = Observable.zip(
8               Observable.just(100, 200, 300),
9               Observable.just(10, 20, 30),
10              Observable.just(1, 2, 3),
11              (a, b, c) -> a + b + c)
12              .zipWith(Observable.just(1000, 2000, 3000),
13                  (abc, d) -> abc + d);
14
15          source.subscribe(System.out::println);
```

combineLatest()

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



combineLatest()



combineLatest()

최대 9개의 Observable 결합 가능

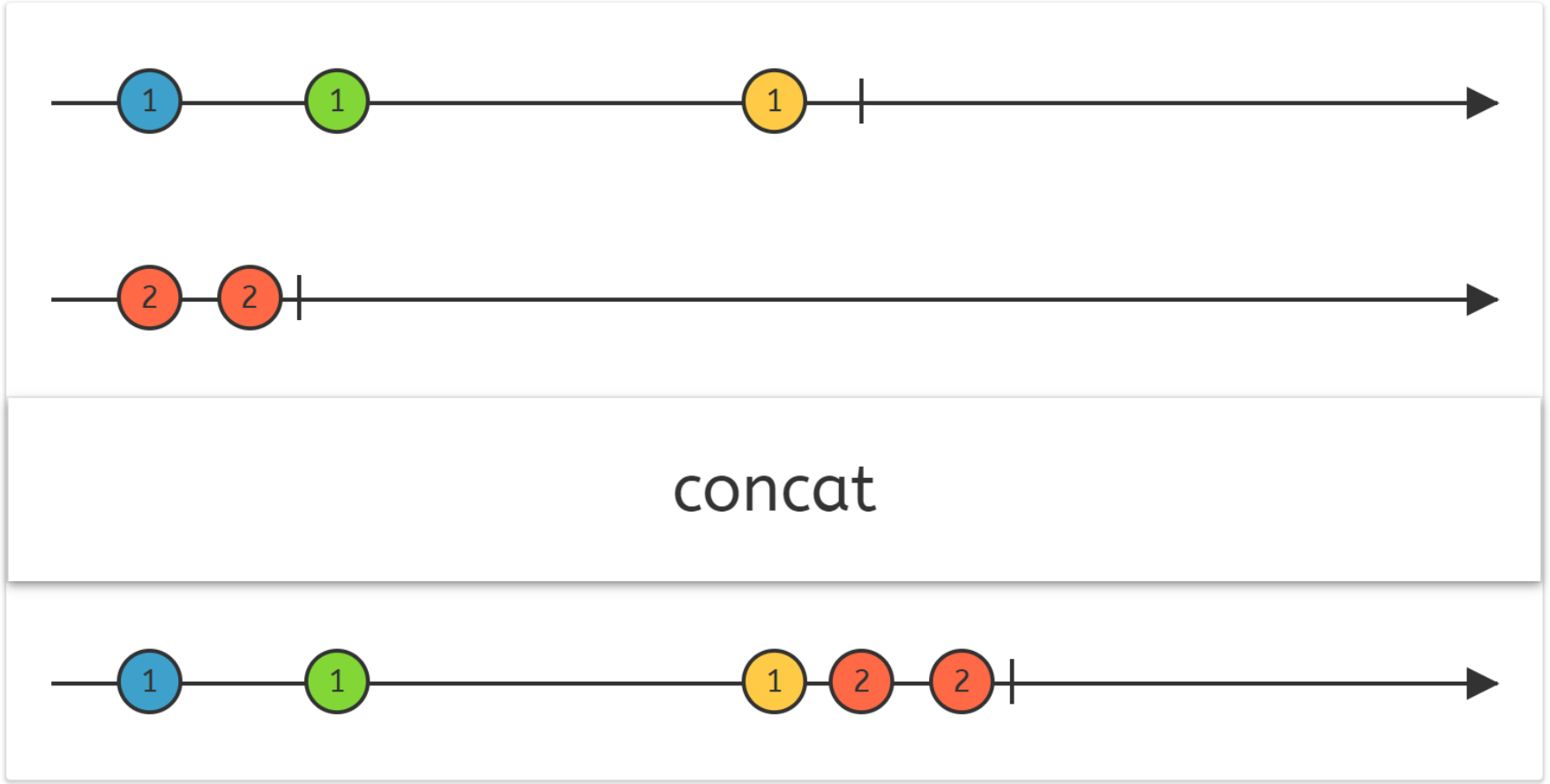
```
@SchedulerSupport(SchedulerSupport.NONE)
public static <T1, T2, R> Observable<R> combineLatest(
    ObservableSource<? extends T1> source1,
    ObservableSource<? extends T2> source2,
    BiFunction<? super T1, ? super T2, ? extends R> combiner)
```

combineLatest()

```
8         String[] data1 = {"1", "2", "3", "4"};
9         String[] data2 = {"<>", "**", "()"};
10
11        Observable<String> source = Observable.combineLatest(
12            Observable.fromArray(data1)
13                .zipWith(Observable.interval( period: 100L, TimeUnit.MILLISECONDS),
14                    (shape, i) -> shape),
15            Observable.fromArray(data2)
16                .zipWith(Observable.interval( initialDelay: 150L,  period: 200L, TimeUnit.MILLISECONDS),
17                    (shape, i) -> shape),
18            (v1, v2) -> v1 + v2);
19
20        source.subscribe(value -> System.out.println(Thread.currentThread().getName() + " | " + "value = " + value));
21        CommonUtils.sleep( millis: 1000);
```

concat()

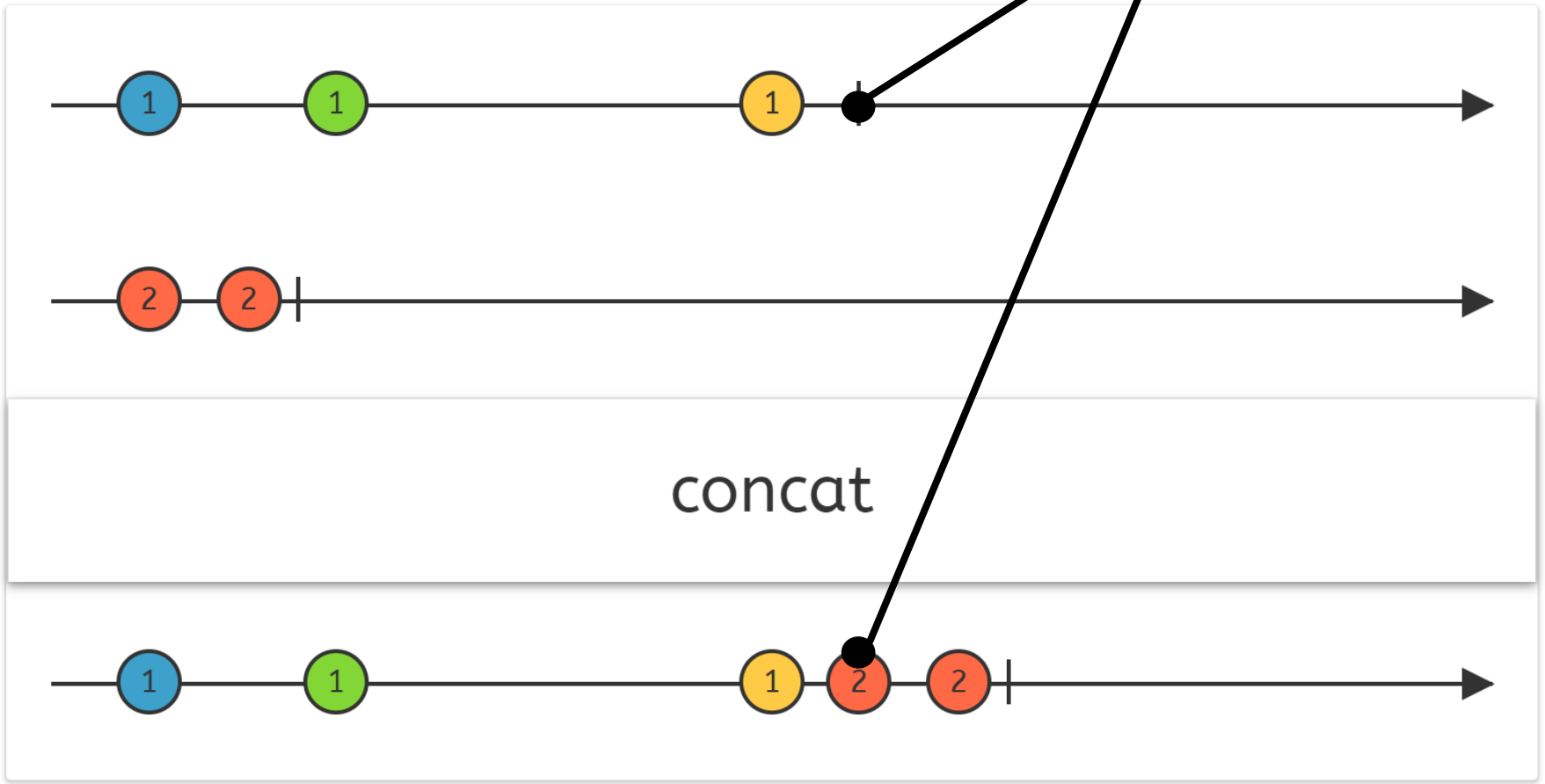
http://reactivex.io/documentation/operators/concat.html



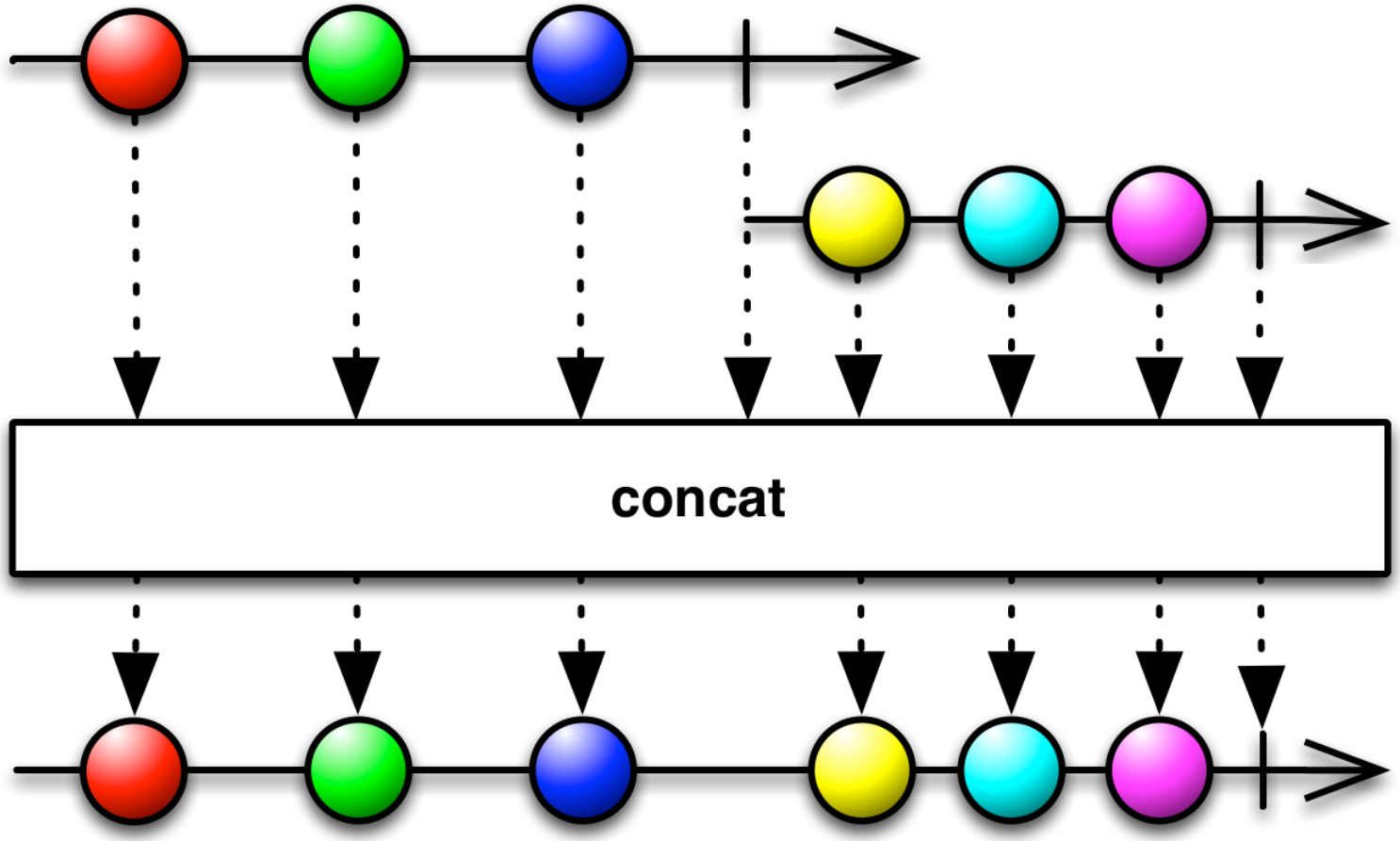
concat()

이전 Observable이 onComplete
이벤트를 반환할 때까지 기다림

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



concat()



concat()

최대 9개의 Observable 결합 가능

```
@SchedulerSupport(SchedulerSupport.NONE)
public static <T> Observable<T> concat(
    ObservableSource<? extends T> source1,
    ObservableSource<? extends T> source2
)
```

concat()

```
8         String[] data1 = {"1", "4", "7"};
9         String[] data2 = {"2", "5", "8"};
10        String[] data3 = {"3", "6", "9"};
11
12        Observable<String> source1 = Observable.fromArray(data1);
13        Observable<String> source2 = Observable.interval( period: 100L, TimeUnit.MILLISECONDS)
14            .map(Long::intValue)
15            .map(idx -> data2[idx])
16            .take(data2.length);
17        Observable<String> source3 = Observable.fromArray(data3);
18
19        Observable<String> source = Observable.concat(source1, source2, source3);
20        source.subscribe(Log::i);
21        CommonUtils.sleep( millis: 1000);
```

조건 연산자

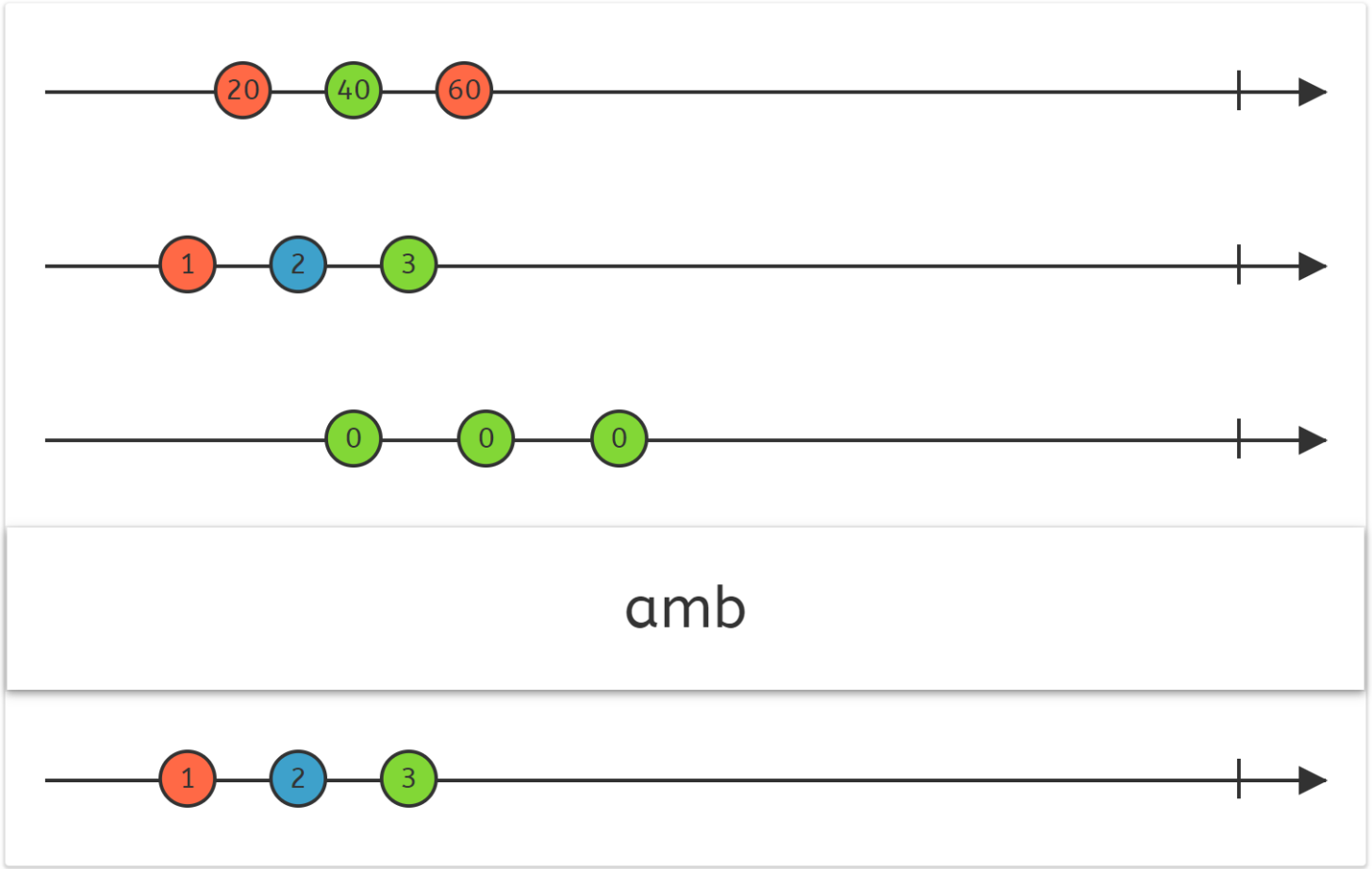
조건 연산자는 Observable의 흐름을 제어하는 역할

필터 연산자는 발행된 값을 채택/기각 결정, 조건 연산자는 흐름 제어에 초점을 맞춤

- amb()
- takeUntil()
- skipUntil()
- all()

amb()

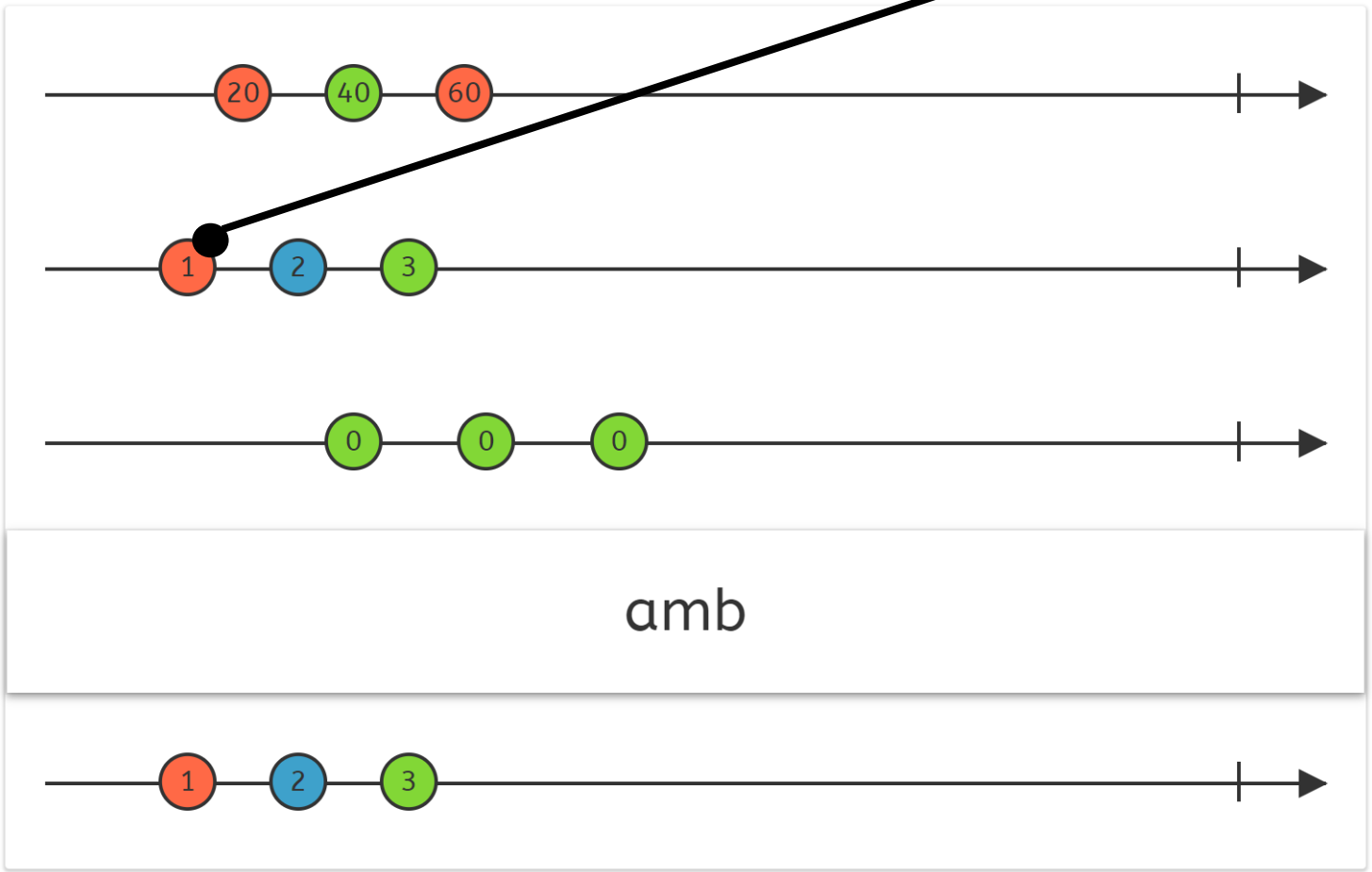
http://reactivex.io/documentation/operators/amb.html



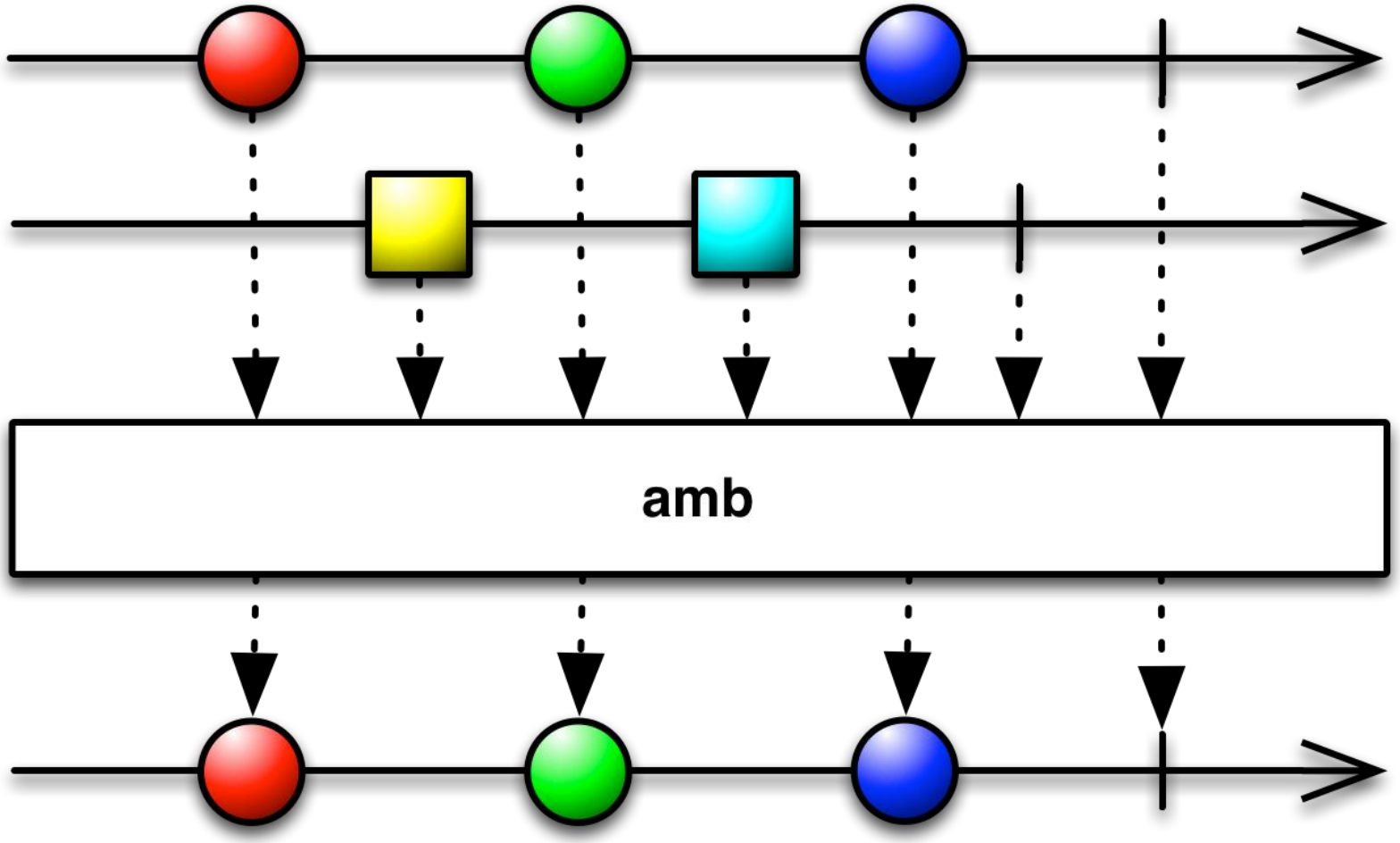
amb()

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

가장 먼저 데이터를 발행하는 Observable 선택



amb()



amb()

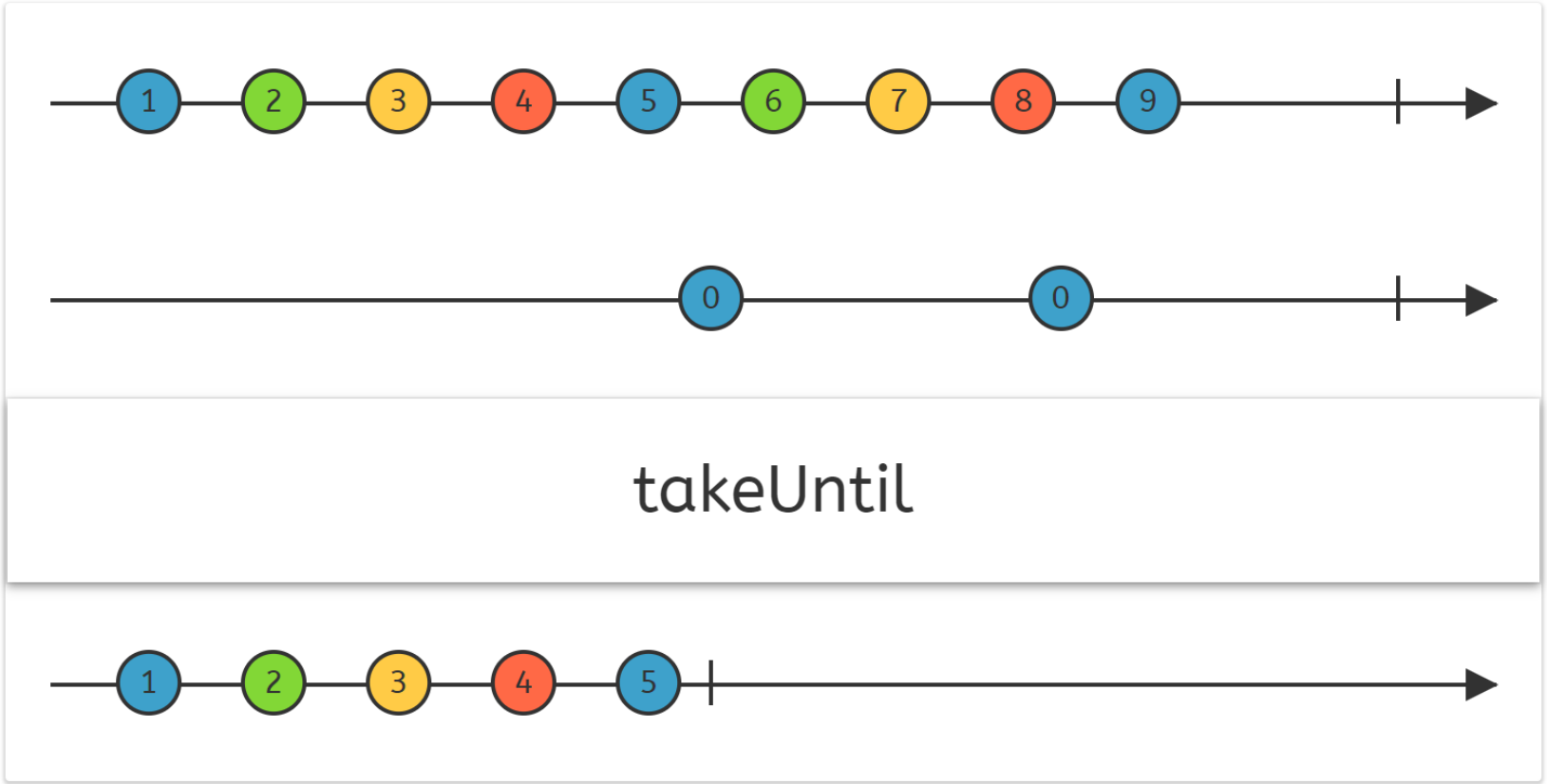
```
@SchedulerSupport(SchedulerSupport.NONE)
public static <T> Observable<T> amb(
    Iterable<? extends ObservableSource<? extends T> sources)
```

amb()

```
11          String[] data1 = {"1", "3", "5"};
12          String[] data2 = {"2", "4", "6"};
13
14          List<Observable<String>> sources = Arrays.asList(
15              Observable.fromArray(data1),
16              Observable.fromArray(data2)
17                  .delay( delay: 100L, TimeUnit.MILLISECONDS)
18          );
19
20          Observable.amb(sources)
21              .subscribe(Log::i);
22
23          CommonUtils.sleep( millis: 1000);
```

takeUntil()

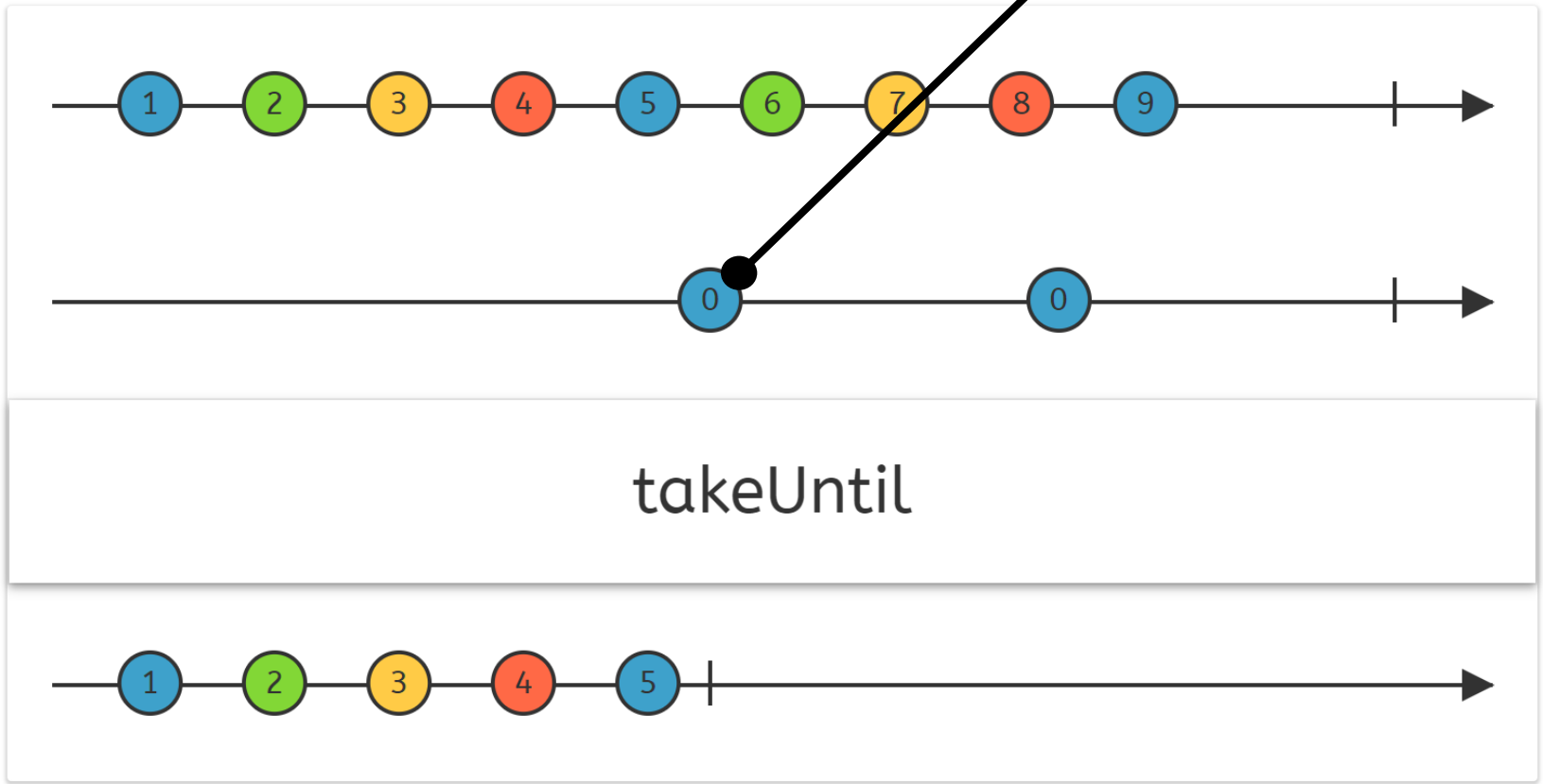
http://reactivex.io/documentation/operators/takeuntil.html



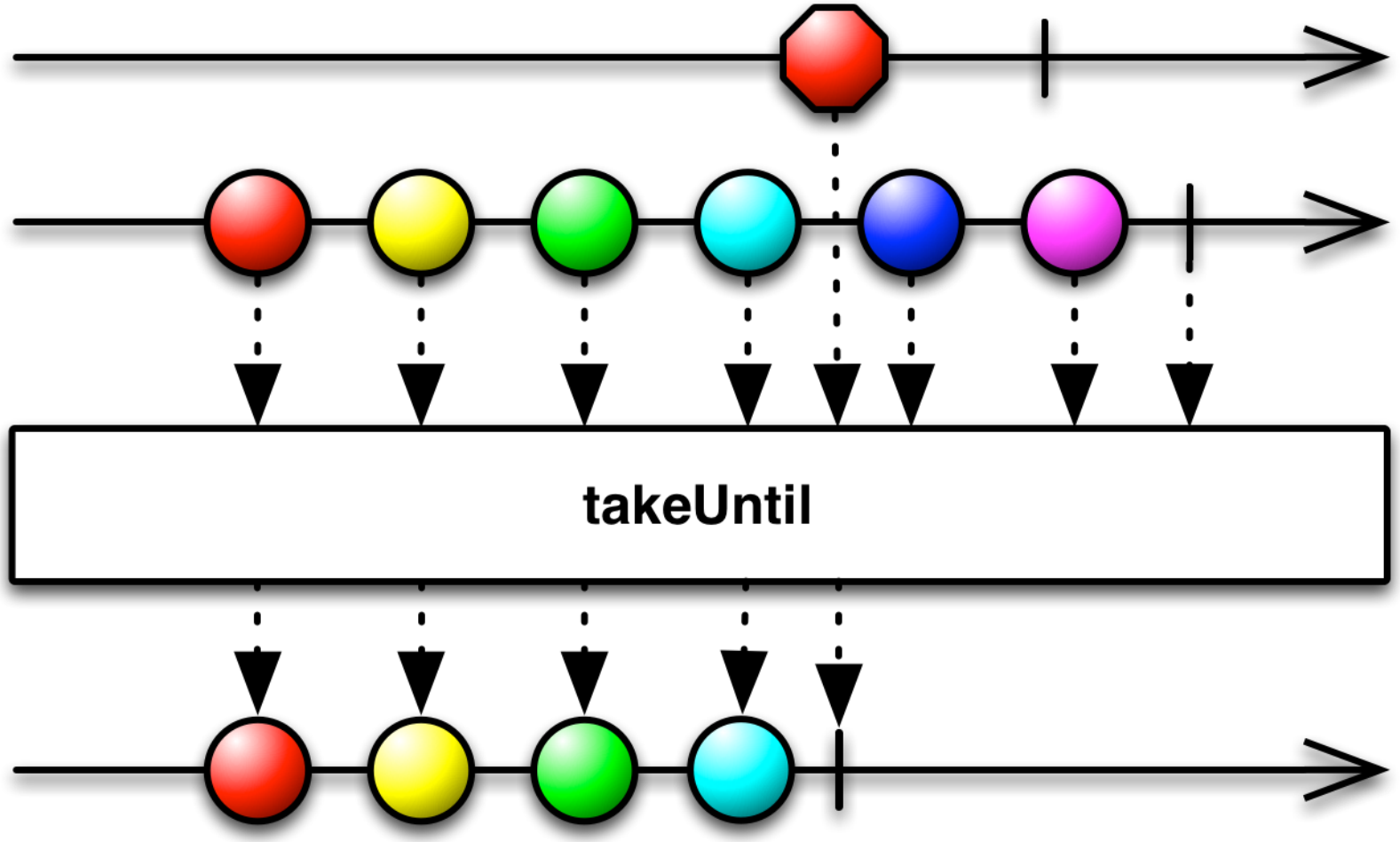
takeUntil()

다른 Observable에서
데이터가 발행되기 전까지만
데이터 발행

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



takeUntil()



takeUntil()

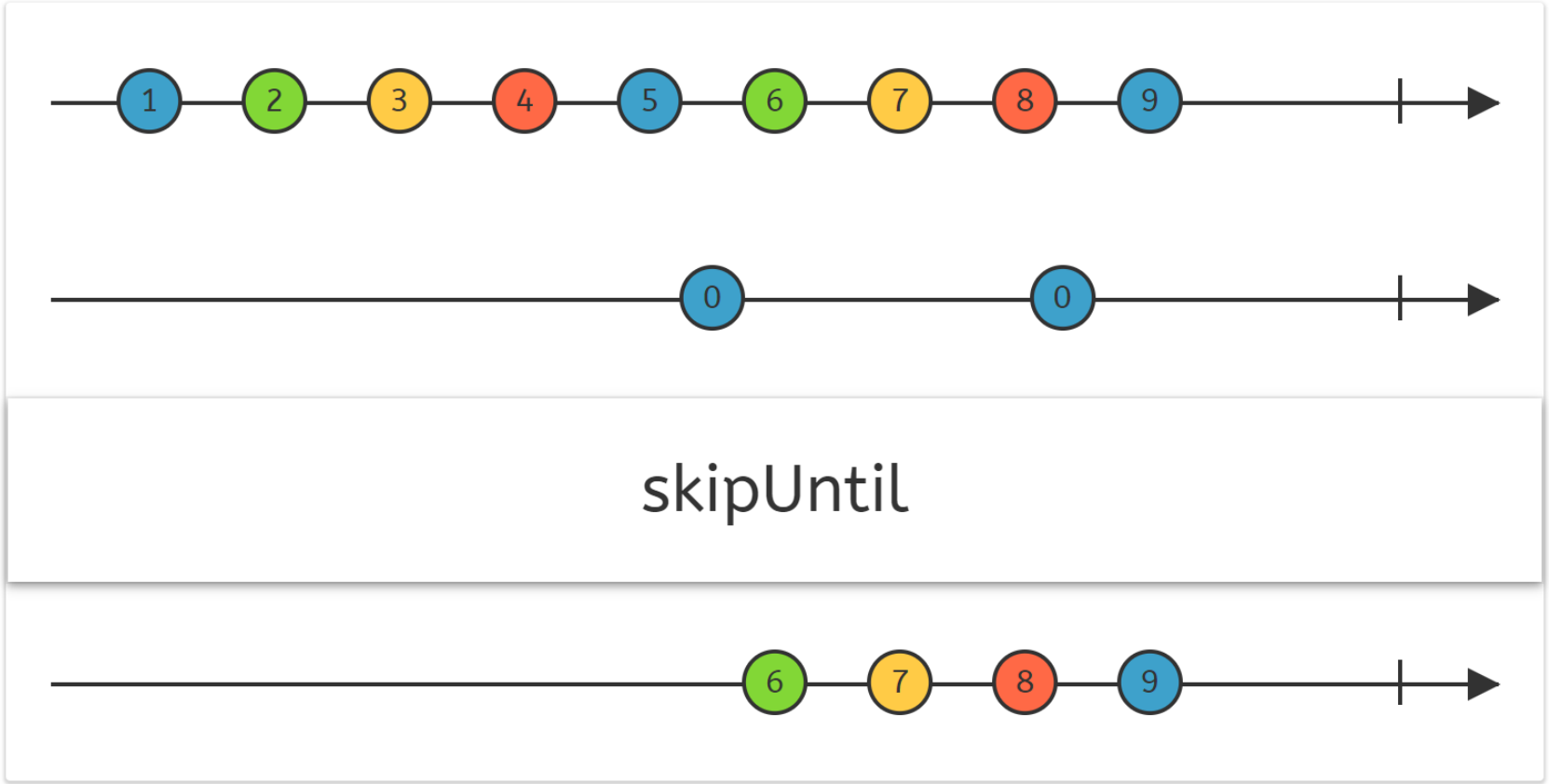
```
@SchedulerSupport(SchedulerSupport.NONE)
public final <U> Observable<T> takeUntil(ObservableSource<U> other)
```

takeUntil()

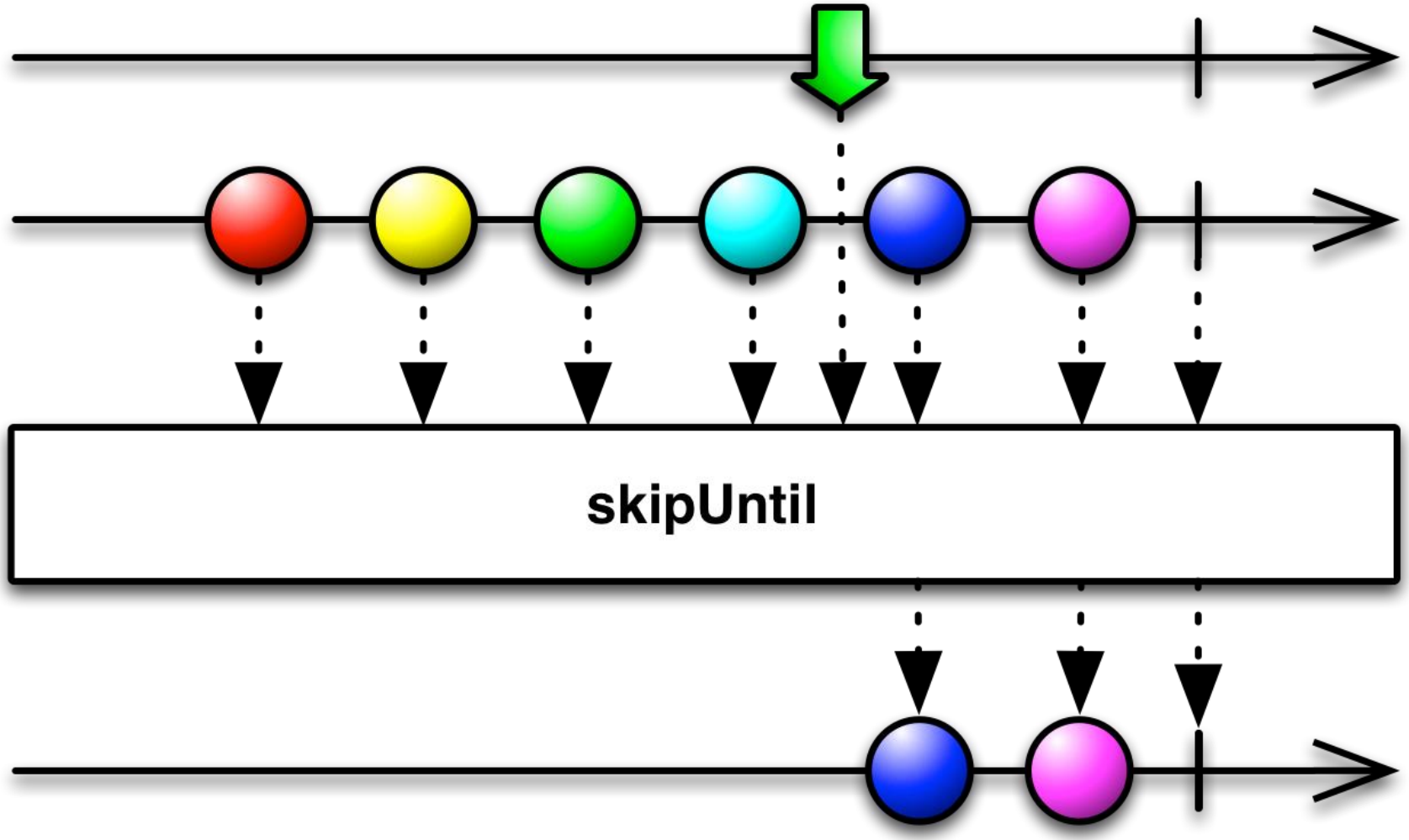
```
11          String[] data = {"1", "2", "3", "4", "5", "6"};
12
13          Observable<String> source = Observable.fromArray(data)
14              .zipWith(Observable.interval( period: 100L, TimeUnit.MILLISECONDS), (val, i) -> val)
15              .takeUntil(Observable.timer( delay: 500L, TimeUnit.MILLISECONDS));
16
17          source.subscribe(Log::i);
18          CommonUtils.sleep( millis: 1000);
```

skipUntil()

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

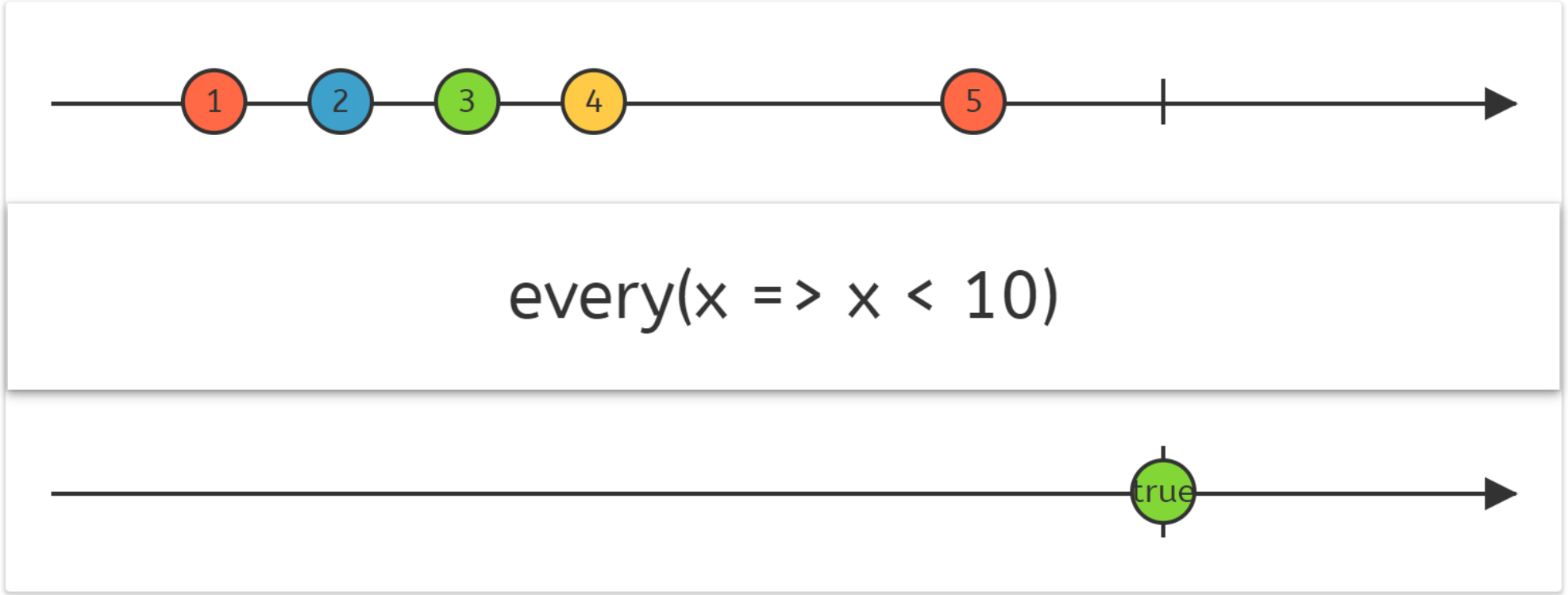


skipUntil()

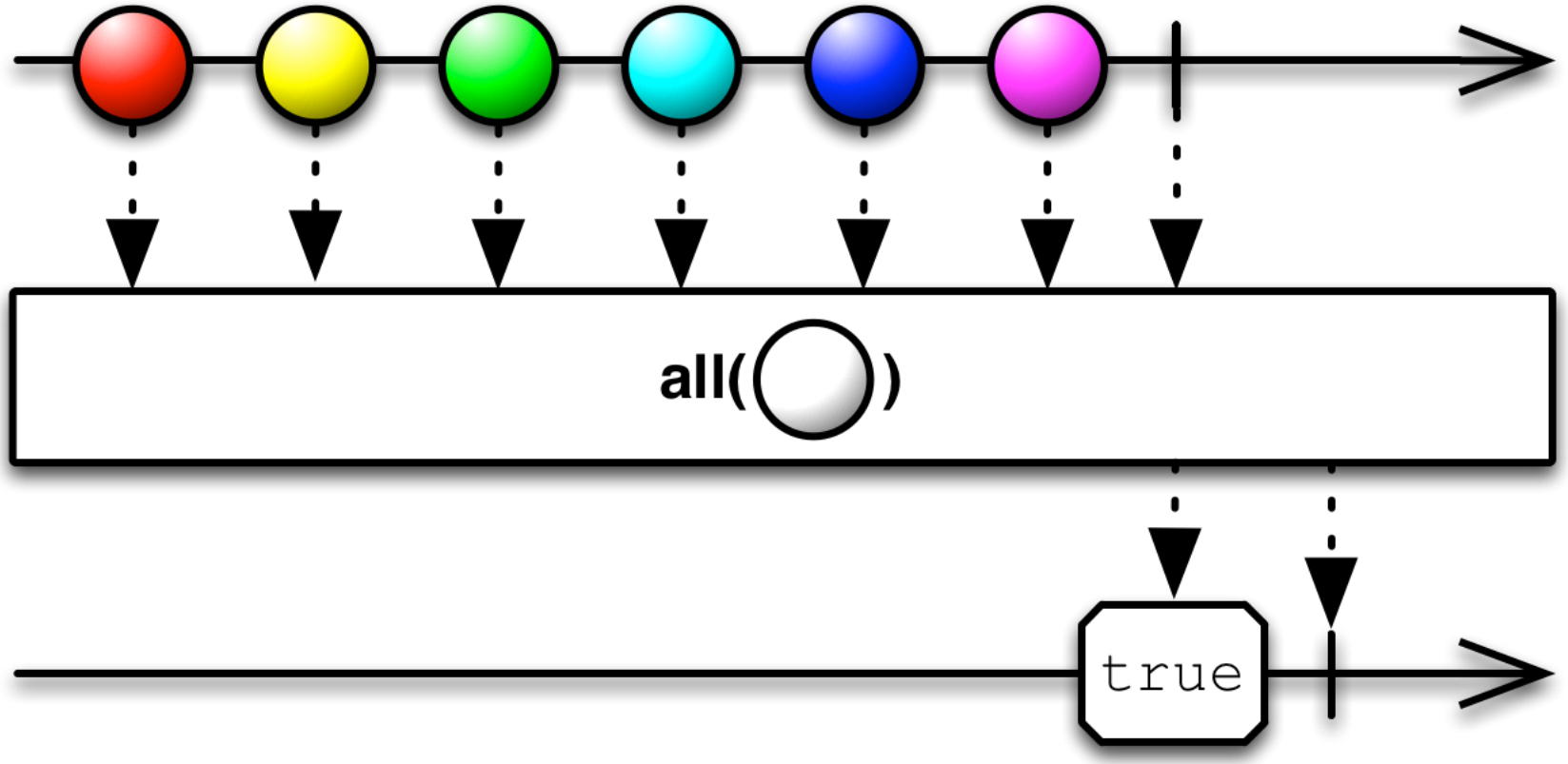


all()

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



all()



all()

```
@SchedulerSupport(SchedulerSupport.NONE)
public final Single<Boolean> all(Predicate<? super T> predicate)
```

all()

```
8           String[] data = {"o", "o", "o", "o", "o"};
9
10          Single<Boolean> source = Observable.fromArray(data)
11              .all("o"::equals);
12          //  .all(val -> "o".equals(val));
13
14          source.subscribe(System.out::println);
```

수학 연산자

RxJava 1.x에서는 RxJavaMath 이용

RxJava 2.x에서는 RxJava2Extensions 이용

- count() - Single
- max() - Flowable<T>
- min() - Flowable<T>
- sum() - Flowable<T>
- average() - Flowable<T>

수학 연산자

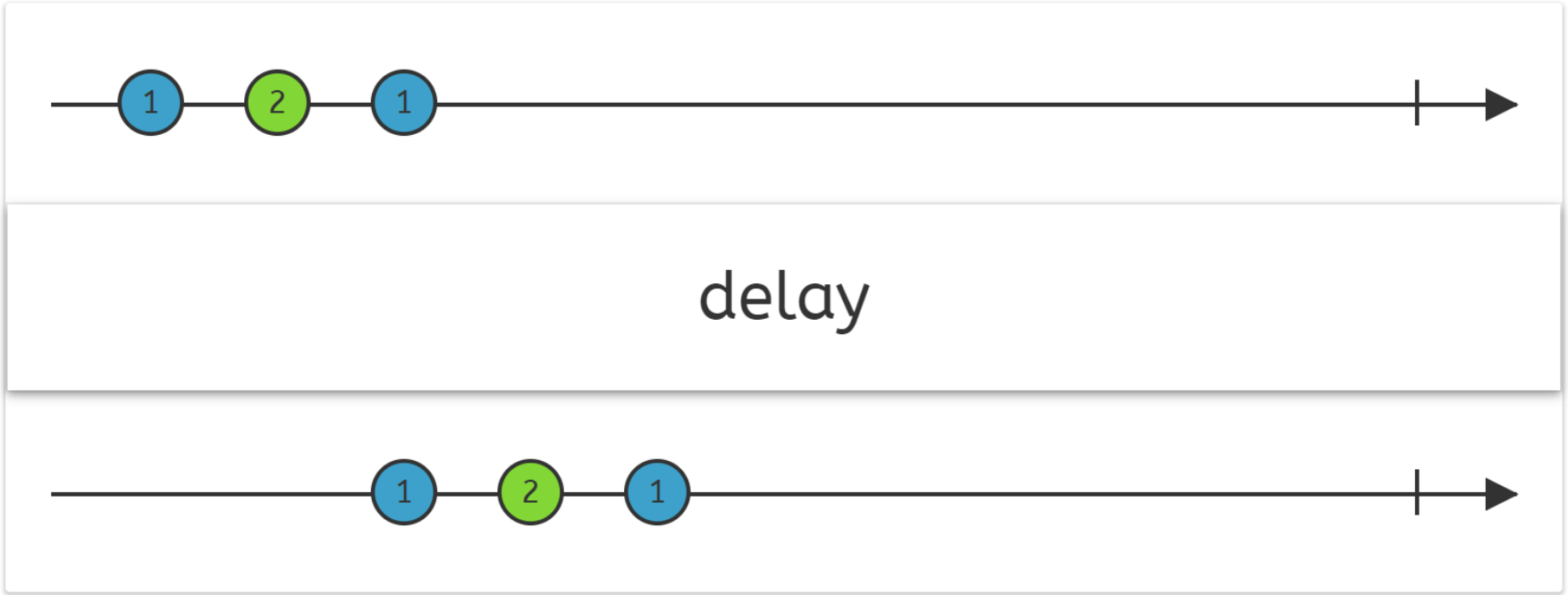
```
10          Integer[] data = {1, 2, 3, 4};
11
12          Single<Long> source = Observable.fromArray(data)
13              .count();
14          source.subscribe(System.out::println);
15
16          Flowable.fromArray(data)
17              .to(MathFlowable::max)
18              .subscribe(System.out::println);
19
20          Flowable<Integer> flowable = Flowable.fromArray(data)
21              .to(MathFlowable::sumInt);
22          flowable.subscribe(System.out::println);
23
24          Flowable<Double> flowable2 = Flowable.fromArray(data)
25              .to(MathFlowable::averageDouble);
26          flowable2.subscribe(System.out::println);
```

시간과 관련된 연산자

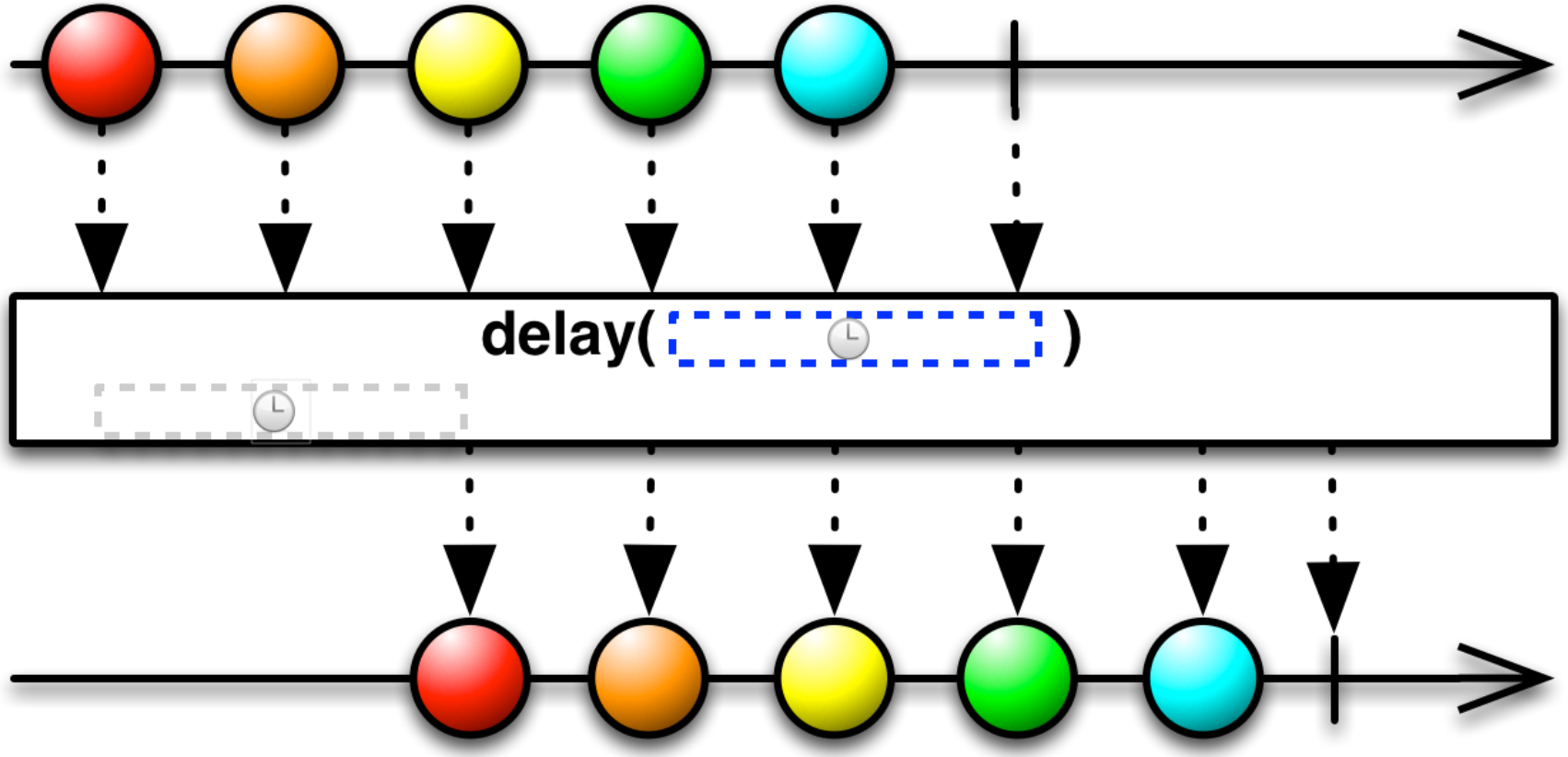
- delay()
- timeInterval()

delay()

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



delay()

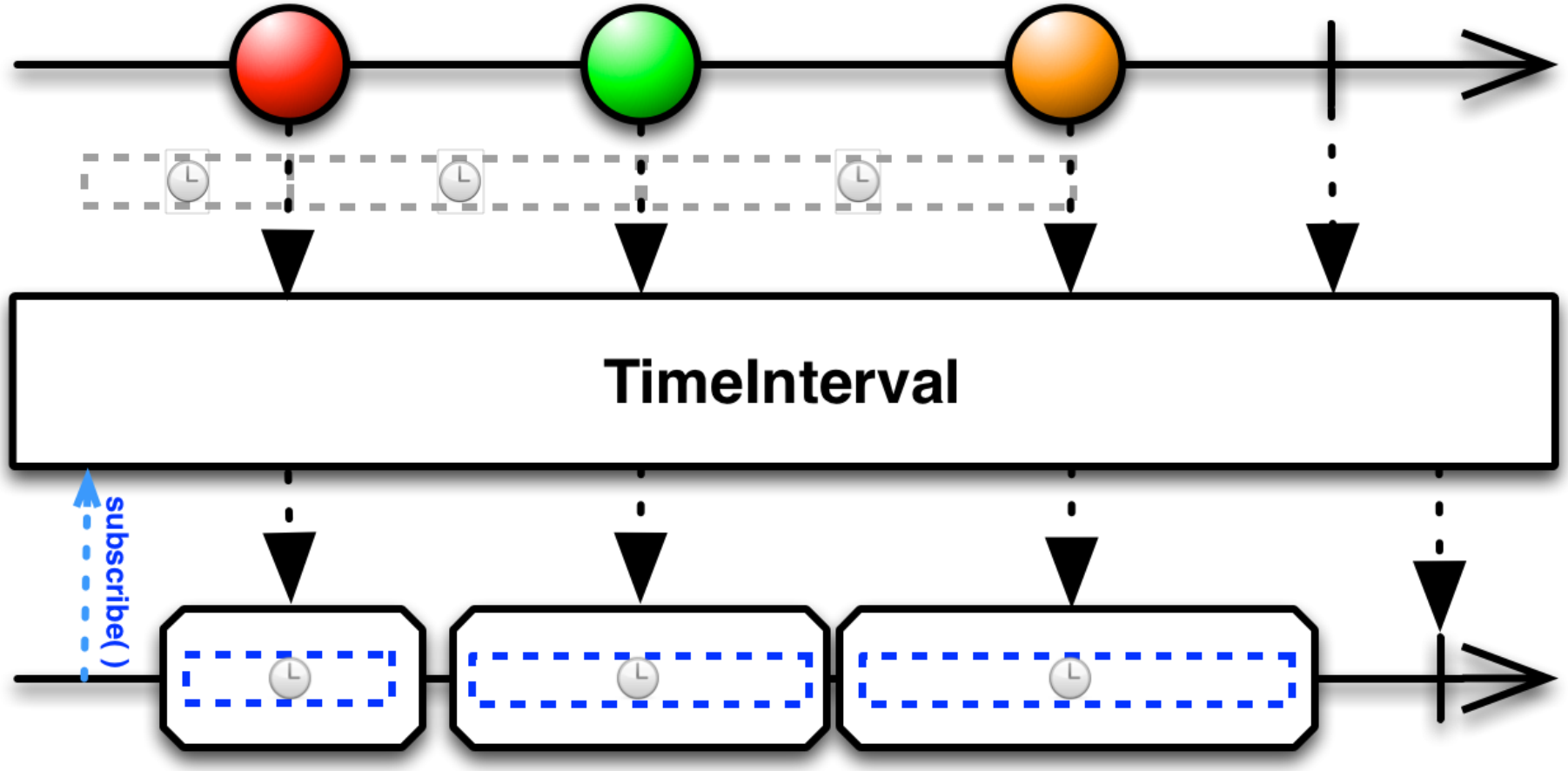


delay()

```
@SchedulerSupport(SchedulerSupport.NONE)
public final Observable<T> delay(long delay, TimeUnit unit)
```

timeInterval()

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



timeInterval()

```
@SchedulerSupport(SchedulerSupport.NONE)
public final Observable<Timed<T>> timeInterval()
```

timeInterval()

```
11          String[] data = {"1", "2", "3"};
12          long startTime = System.currentTimeMillis();
13
14          Observable<Timed<String>> source = Observable.fromArray(data)
15              .delay(item -> {
16                  try {
17                      Thread.sleep(new Random().nextInt( bound: 100));
18                  } catch (InterruptedException e) {
19                      e.printStackTrace();
20                  }
21                  return Observable.just(item);
22              })
23              .timeInterval();
24
25          source.subscribe(value -> {
26              long time = System.currentTimeMillis() - startTime;
27              System.out.println(time + " | " + "value = " + value);
28          });
29          CommonUtils.sleep( millis: 1000);
```

timeInterval()

```
11          String[] data = {"1", "2", "3"};
12          long startTime = System.currentTimeMillis();
13
14          Observable.fromArray(data)
15  λ↑          .doOnNext(s -> {
16              if (s.equals("2")) {
17                  throw new IllegalArgumentException();
18              }
19          })
20          .timeInterval();
21
22          source.subscribe(value -> {
23              long time = System.currentTimeMillis() - startTime;
24              System.out.println(time + " | " + "value = " + value);
25  λ↑      });
26          CommonUtils.sleep( millis: 1000);
27      }
28  }
29
```

```
485 | value = Timed[time=47, unit=MILLISECONDS, value=1]
559 | value = Timed[time=74, unit=MILLISECONDS, value=2]
632 | value = Timed[time=73, unit=MILLISECONDS, value=3]

Process finished with exit code 0
```

감사합니다