2019 MAY 22, HONG KONG

OBSERVABLE

OBSERVABLE

RxSwift

RxJava

RxJS

RxScala

Async Push Model

ReactiveX

PROGRAMMING MODEL

- **Sync**
  - f(**input**) -> return **something**
    If something error, throw Exception inside

- **Async**
  - f(**input, onSuccessCallback, onErrorCallback**)
    If something error, call onErrorCallback()

PROGRAMMING MODEL (ASYNC)

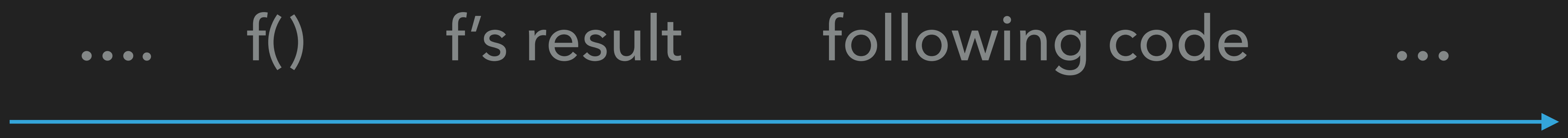
- JavaScript
    1. Pass **callback functions** as parameter
    2. Return a **Promise**
    3. Use syntax sugar for Promise: **await/async**

PROGRAMMING MODEL (ASYNC)

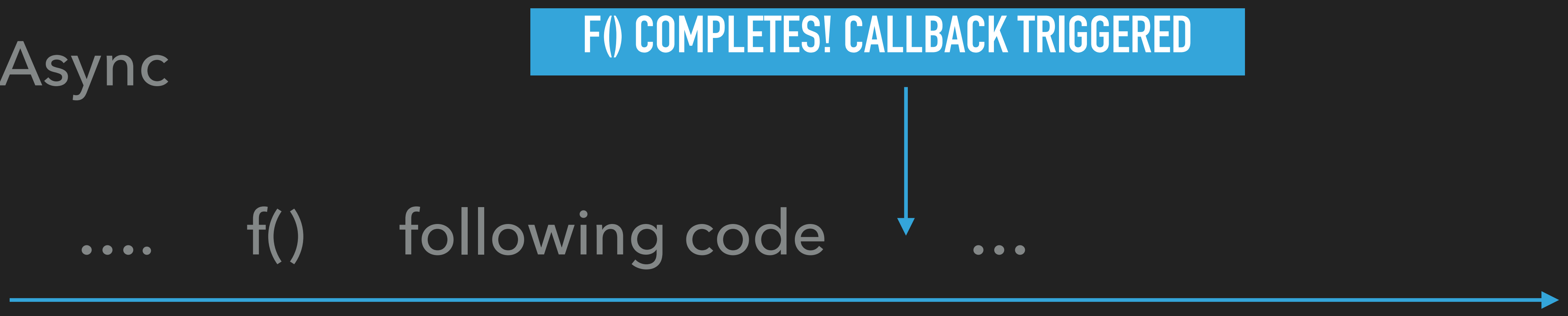
- Java
  1. Pass **anonymous inner class** object as parameter
  2. Pass **functional interface** (Java 8)
  3. Return a **CompletableFuture**

IMAGE A TIMELINE

- Sync



- Async



START OVER, AS A LIST (MULTIPLE VALUES)

- **Sync**
  - f(**input**) -> return **an iterator**
- **Async**
  - f(**input, onResultSuccessCallback (n-th value), onResultErrorCallback**)

ASYNC CASES

- Event: **onKeyPress, onMouseClick, onMouseMove**
- Timer: **Tick every 2 second**
- Remote API call

PUSH MODEL (ASYNC)

- Pull Model: I call function, then wait for response
- Push Modal: I **subscribe** something, then you call me (I am the observer)
- So, a API library called **ReactiveX** was born
  Asynchronous programming with observable streams
- **Observable**: To async generate results (1 or more)
  Observer: To subscribe
- Design Pattern: **Iterator, Observer**

IMAGE A TIMELINE (EVENT)

- Click events as Observable
  rxjs.fromEvent(document, "click")

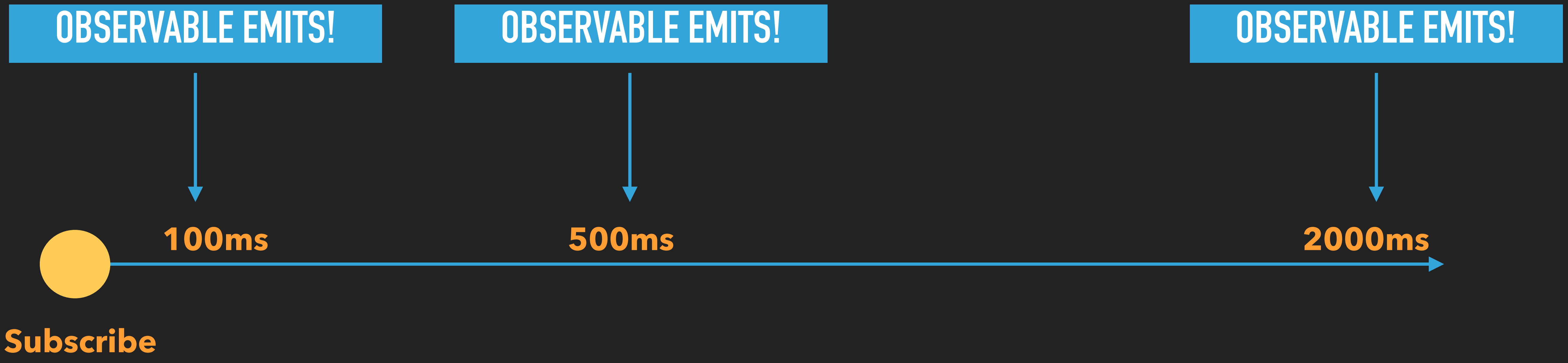
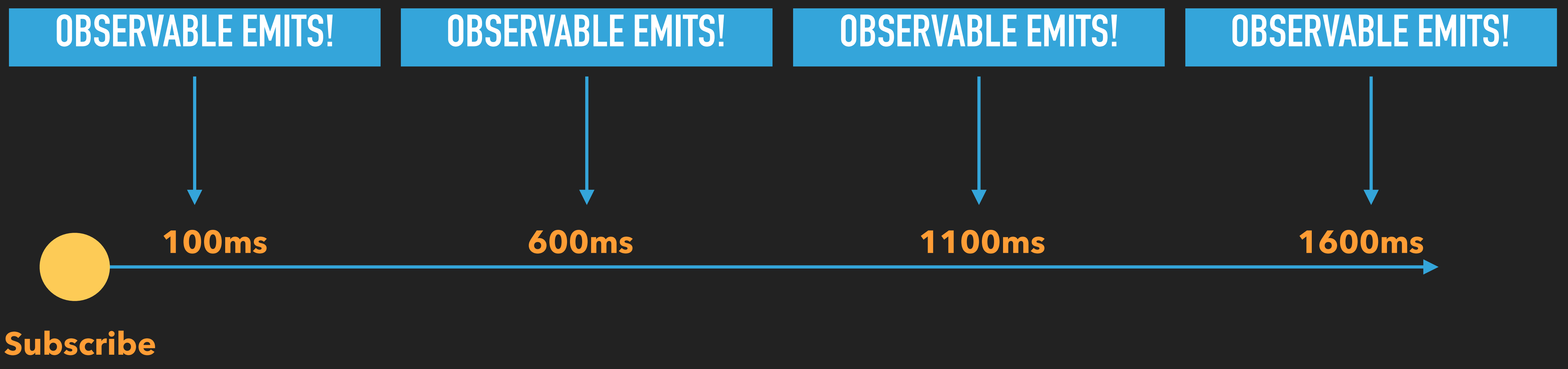


IMAGE A TIMELINE (TIMER)

- An interval timer as Observable

  rxjs.timer(100, 500)

  rxjs.interval(n) = rxjs.timer(n, n)



BASIC OBSERVABLE API

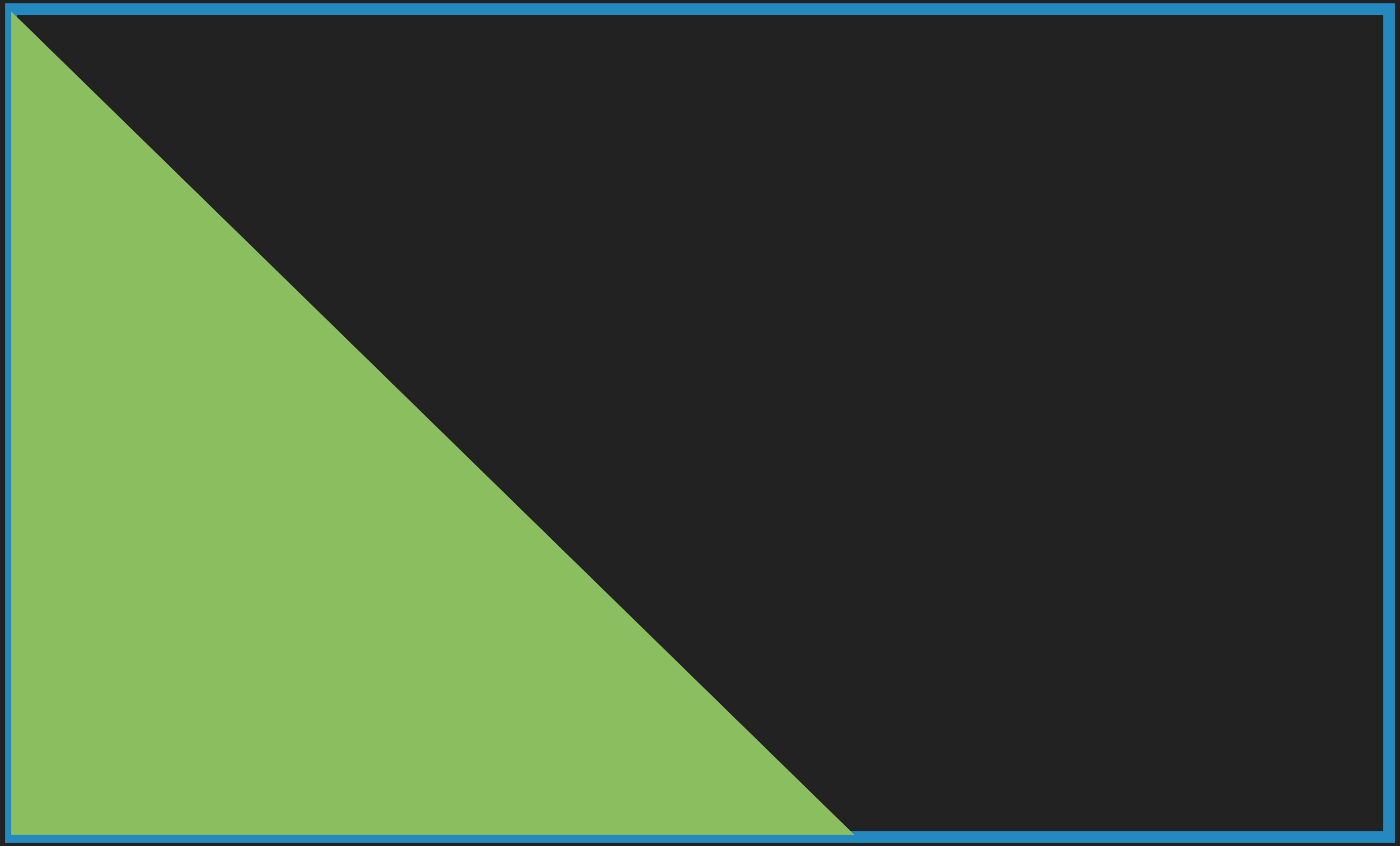
- Generator (Static methods, return an **Observable** object)
    - of, fromEvent, timer, interval, or create your own

- Observable functions
    - Most important one: **subscribe**
    - rxjs.fromEvent(document, `click`).subscribe(console.log)
    - rxjs.interval(500).subscribe(console.log)

BASIC OBSERVABLE API

- Observable functions - Operator
  - map, filter, reduce, skip, tap (Java **forEach**)
  - toArray (Java **collect**)
  - throttleTime, debounceTime, delay
  - …
- In latest RxJS API, use **observableObject.pipe(…..)** us for operator
  For simplicity, we eliminate **pipe** in following examples

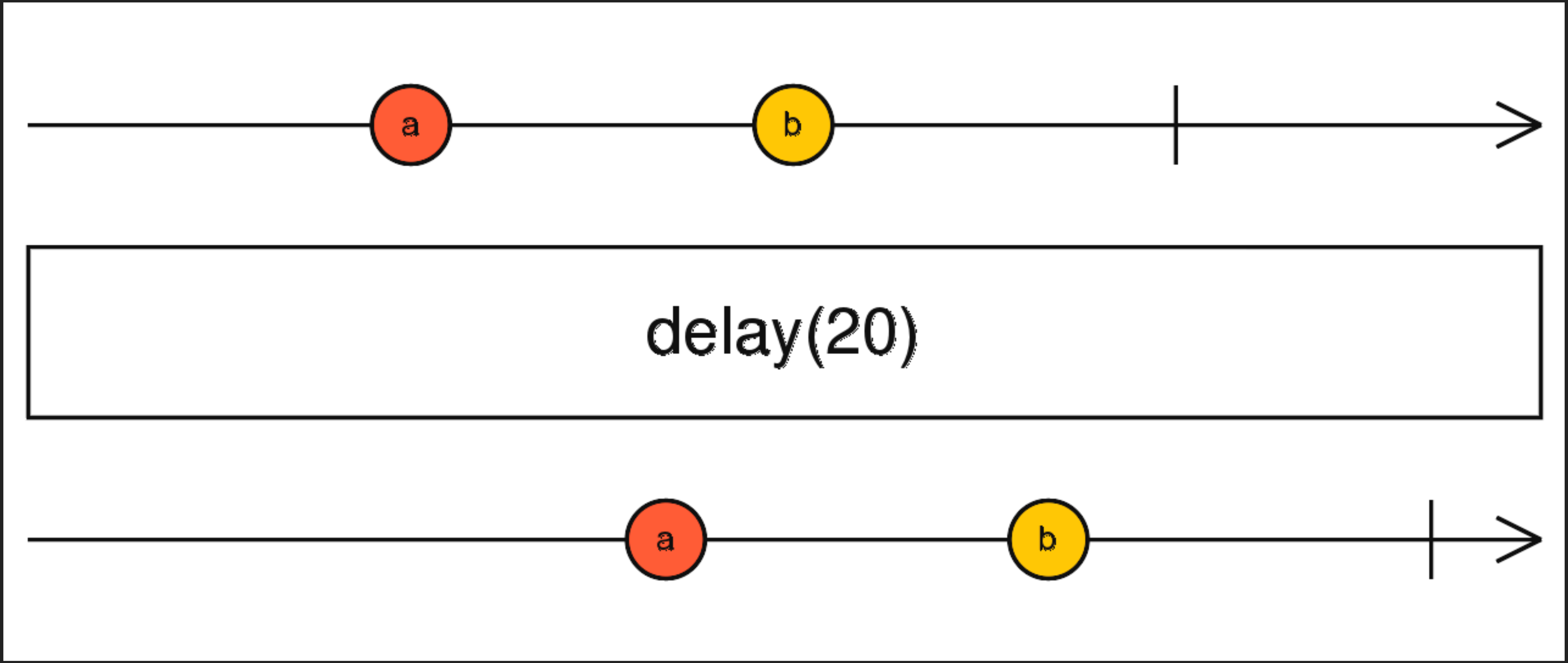
EXAMPLE: ONLY EMIT POSITION WHEN CLICKING IN LOWER AREA



-
  ```
  rxjs.fromEvent(document, `click`):
    filter(event => event.x <= event.y),
    map(event => "(" + event.x + ", " + event.y + ")")
  ```

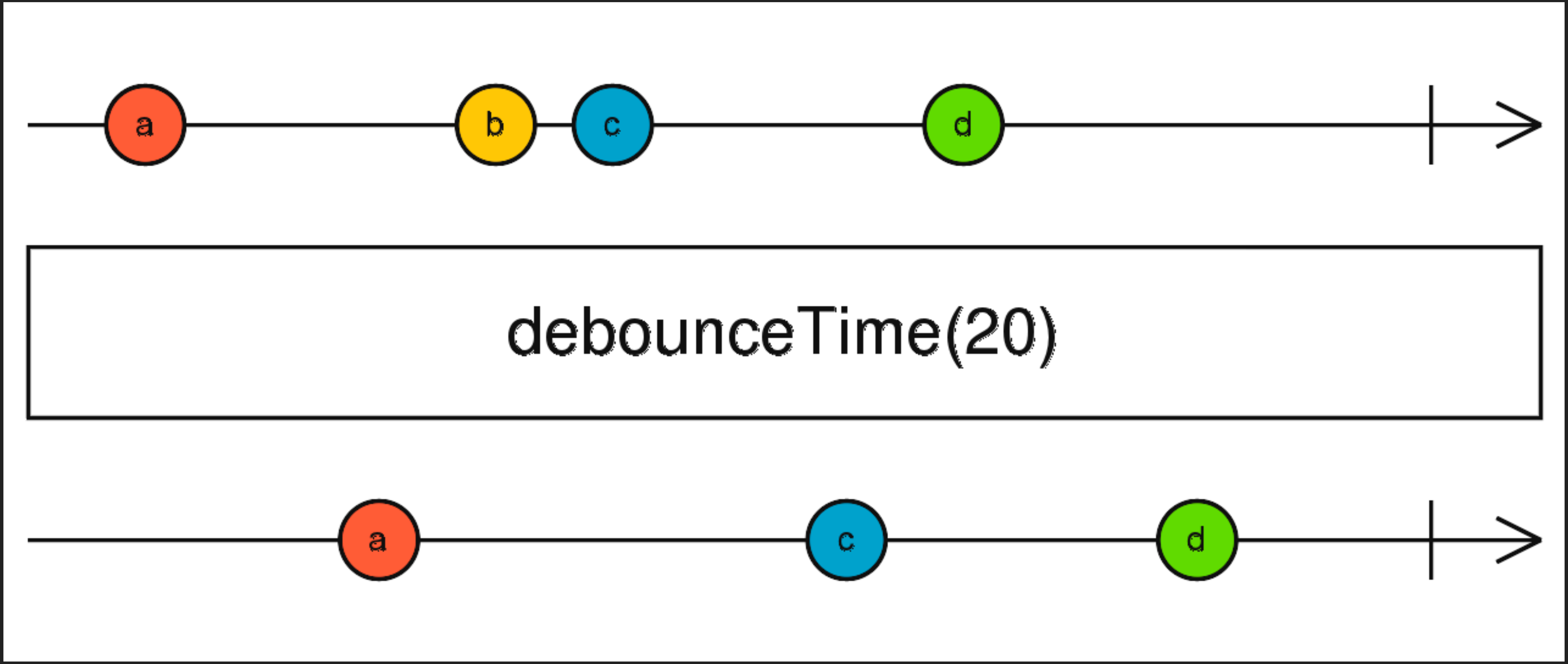
BASIC OBSERVABLE API

- Observable operator: **delay**



BASIC OBSERVABLE API

- Observable operator: **debounceTime**

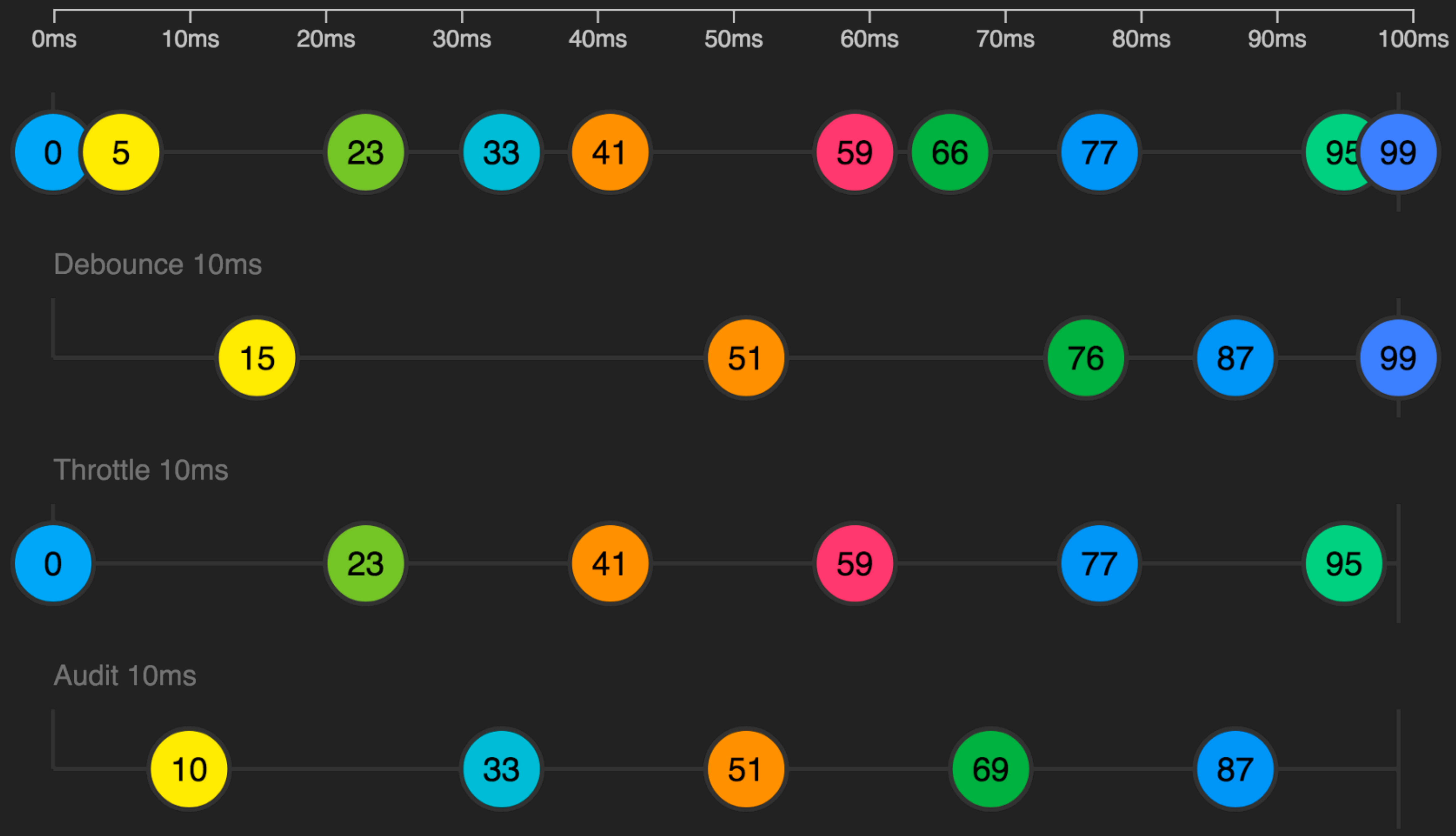


BASIC OBSERVABLE API

- Observable operator: **throttleTime / auditTime**
  - **Throttle:**
    Receive then immediately emit
    Then silence for duration
  - **Audit:**
    Receive then silence for duration
    After duration, emit the last value

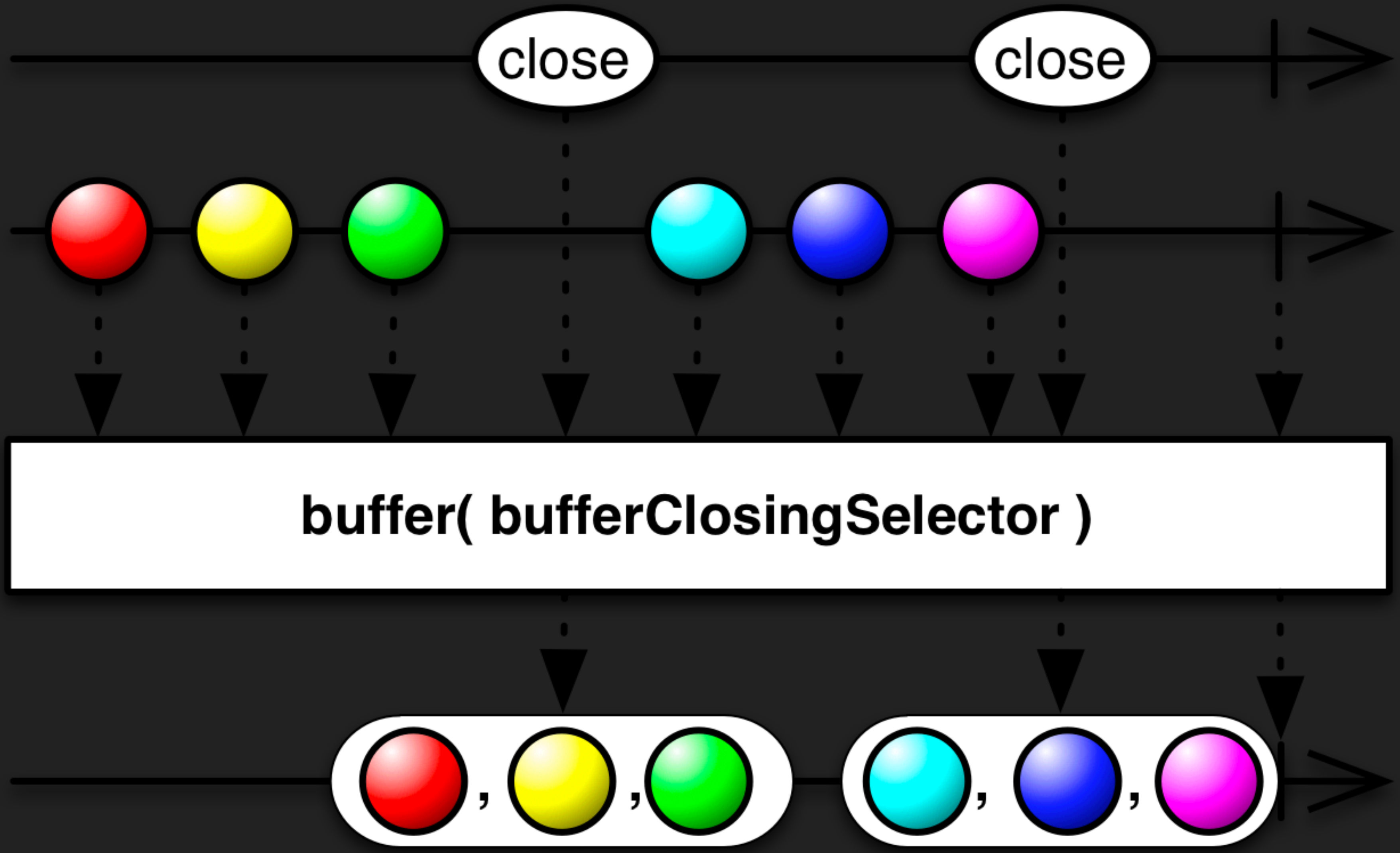
BASIC OBSERVABLE API

- Observable operator: **debounce/throttle/audit**



BASIC OBSERVABLE API

- Observable operator: **buffer**
  It take 1 Observable as parameter. When the parameter emits, it emits the grouped emitted value as an array



EXAMPLE: EMIT WHEN DOUBLE-CLICK

- If two mouse clicks happen between 300ms, then emit one double-click

-
  ```
  var click$ = rxjs.fromEvent(document, `click`):
     click$.buffer( click$.debounce(300) )
               .filter(array => array.length == 2)
  ```

EXAMPLE: SEARCH INPUT

- Requirement: Simply print what you input

-
  ```
  rxjs.fromEvent(inputElement, `input`):
    .map(event => event.targetValue)
    .subscribe(console.log)
  ```

EXAMPLE: SEARCH INPUT

- Requirement: Only print when input changes

-
  ```
  rxjs.fromEvent(inputElement, `input`):
    .map(event => event.targetValue)
    .distinctUntilChanged()
    .subscribe(console.log)
  ```

EXAMPLE: SEARCH INPUT

- Requirement: Only print when input changes, and chars length > 2, and max print every 100 ms

-
  ```
  rxjs.fromEvent(inputElement, `input`):
    .map(event => event.targetValue)
    .filter(value => value.length > 2)
    .distinctUntilChanged()
    .debounceTime(100)
    .subscribe(console.log)
  ```

EXAMPLE: SEARCH INPUT

- Requirement: When input changes, and chars length > 2, then call search API (each API call interval should longer than 100ms), then print API response

-
  ```
  rxjs.fromEvent(inputElement, `input`):
    .map(event => event.targetValue)
    .filter(value => value.length > 2)
    .distinctUntilChanged()
    .debounceTime(100)
    .subscribe(value =>
          API.search(value, result => console.log(result))
    )
  ```

LEGACY API.SEARCH

EXAMPLE: SEARCH INPUT

- Requirement: When input changes, and chars length > 2, then call search API (each API call interval should longer than 100ms), then print API response.

  Previous API should cancel if returned after a new input is triggered

- rxjs.fromEvent(inputElement, `input`):

```
    .map(event => event.targetValue)
    .filter(value => value.length > 2)
    .distinctUntilChanged()
    .debounceTime(100)
    .switchMap(value => API.search(value))
    .subscribe(console.log)
```

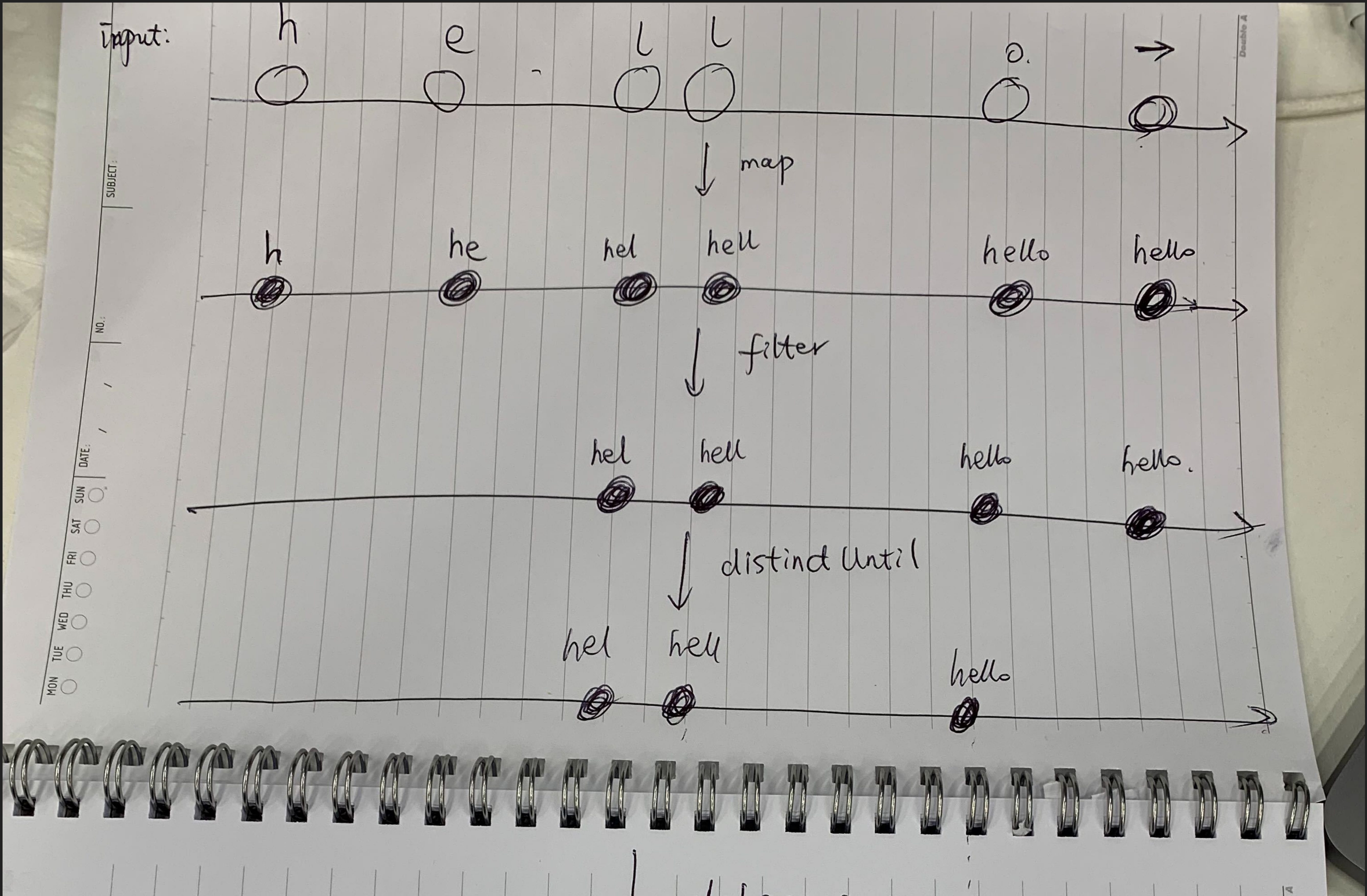
In this case, API.search returns an **Observable**, that only **emit once**

API CALL FOR FRONT-END

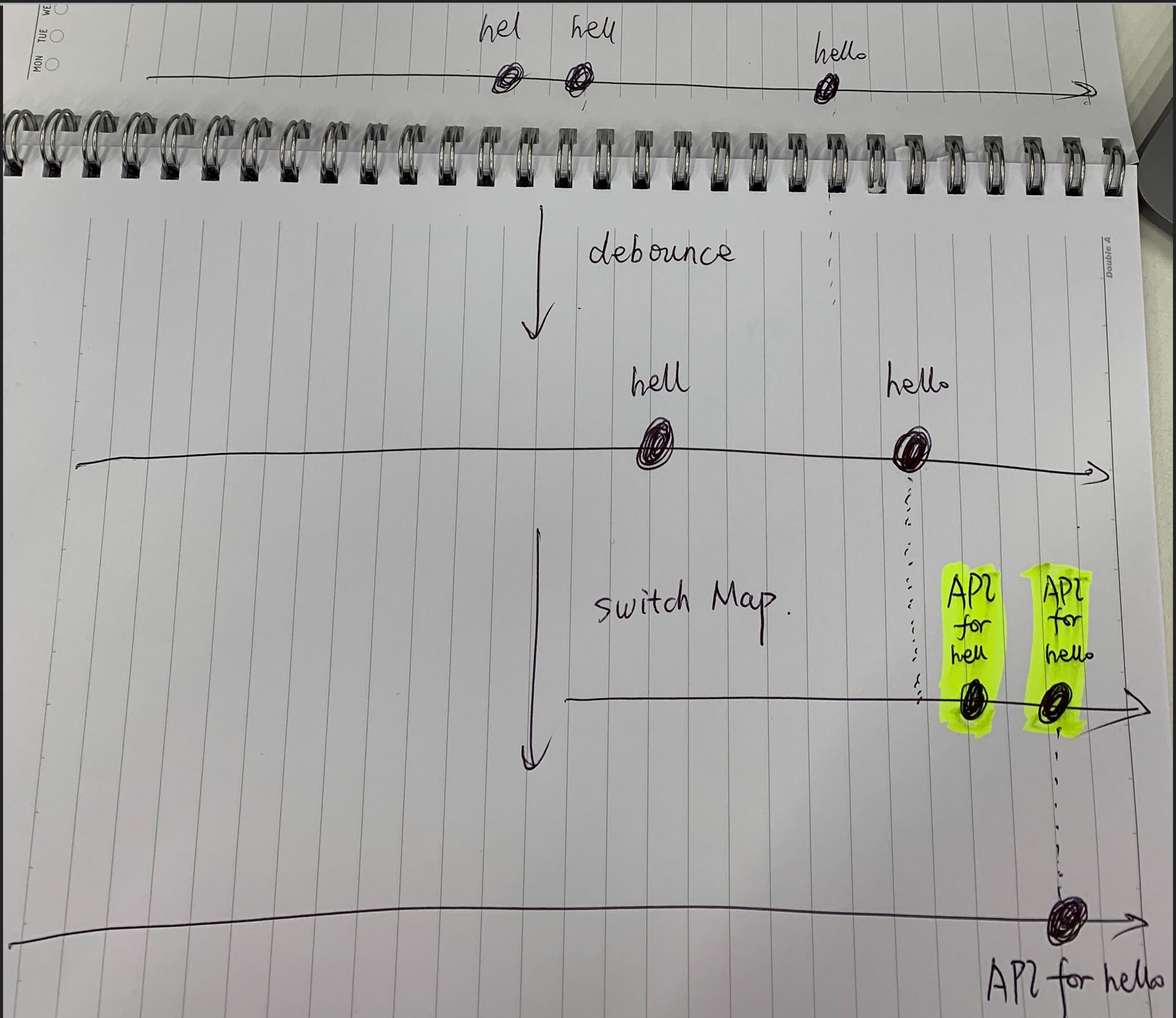
- In our project, we use **Promise**-based API calls:
  XXXService.search(request): Promise<Response>

- Another style (Angular), **Observable**-based API calls:
  XXXService.search(request): Observable<Response>

OBSERVABLE

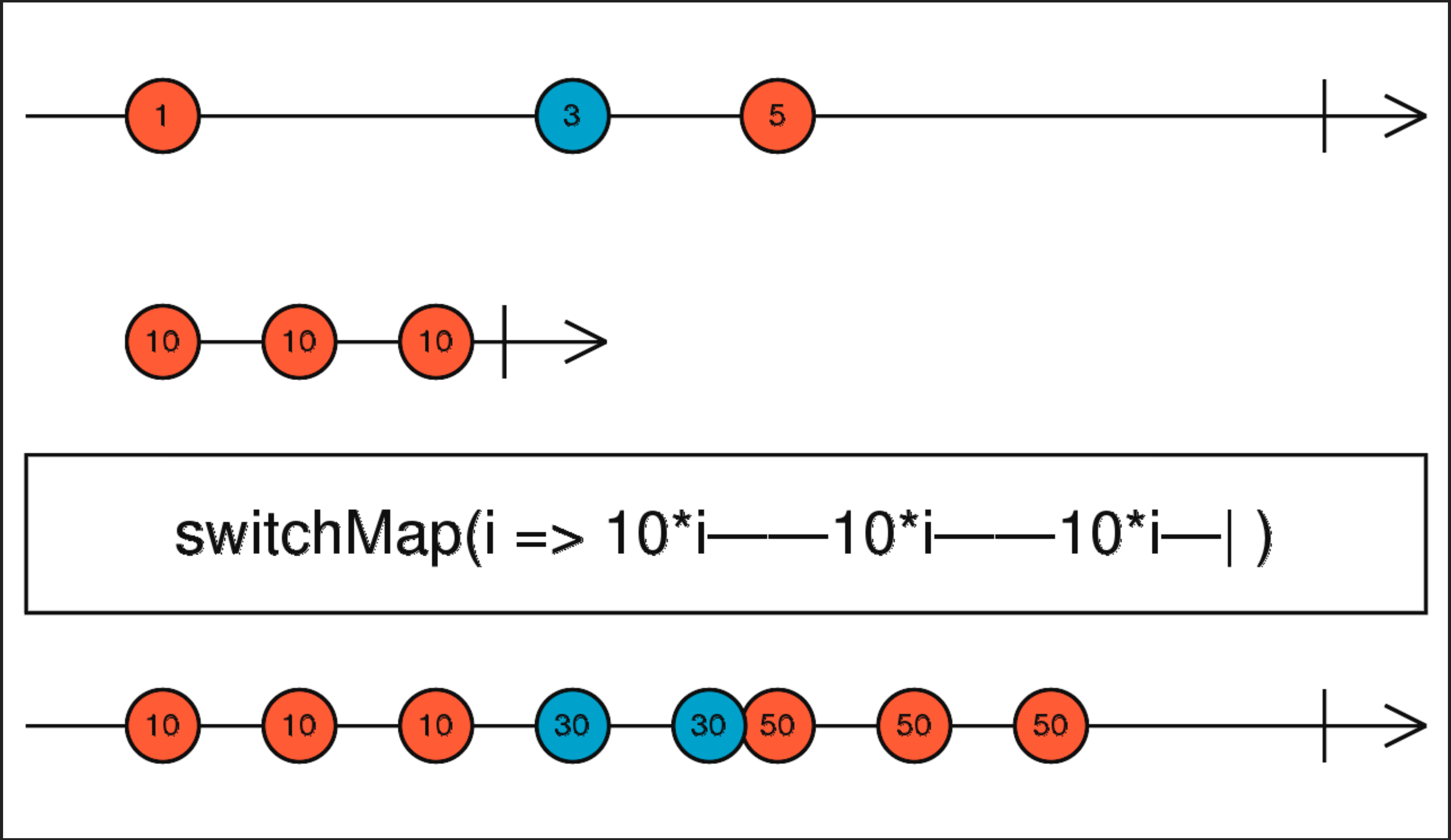

input:
h
e
l
l
o.
→
map
h
he
hel
hell
hello
hello
filter
hel
hell
hello
hello.
distinct Until
hel
hell
hello

OBSERVABLE


hel
hell
hello
debounce
hell
hello
switch Map.
API for hell
API for hello
API for hello

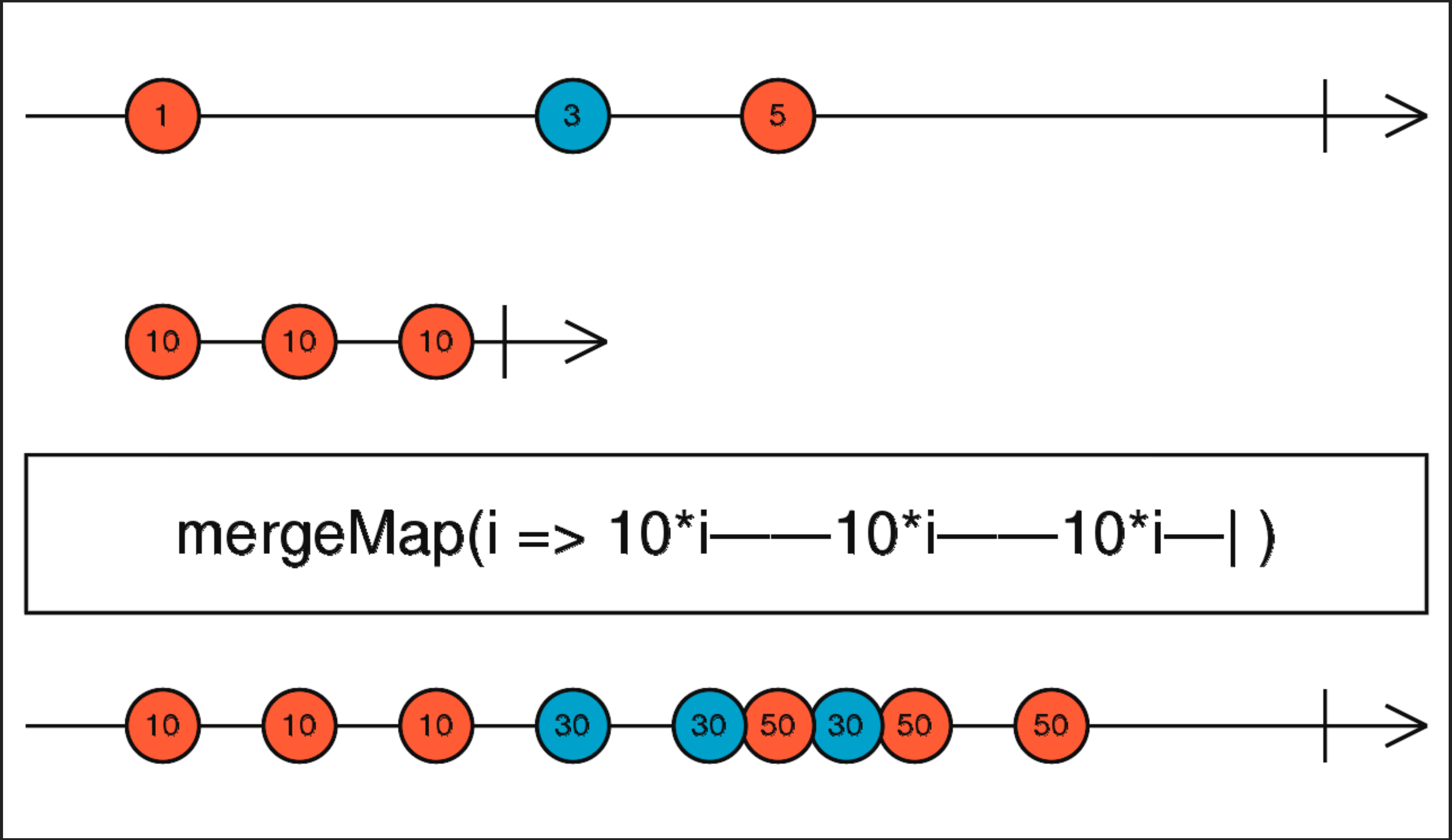
BASIC OBSERVABLE API

- Observable operator: **switchMap**



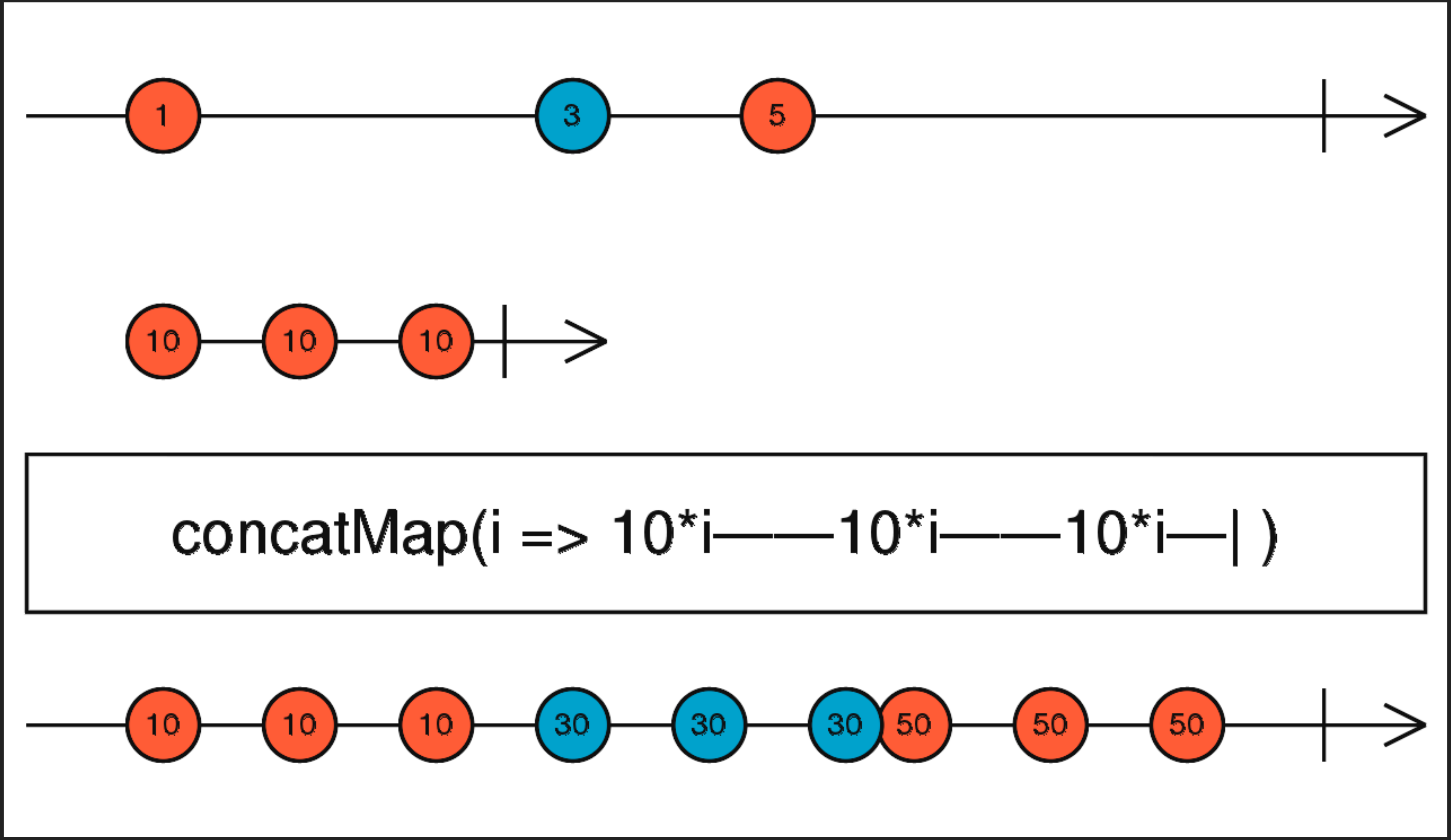
BASIC OBSERVABLE API

- Observable operator: **mergeMap/flatMap**



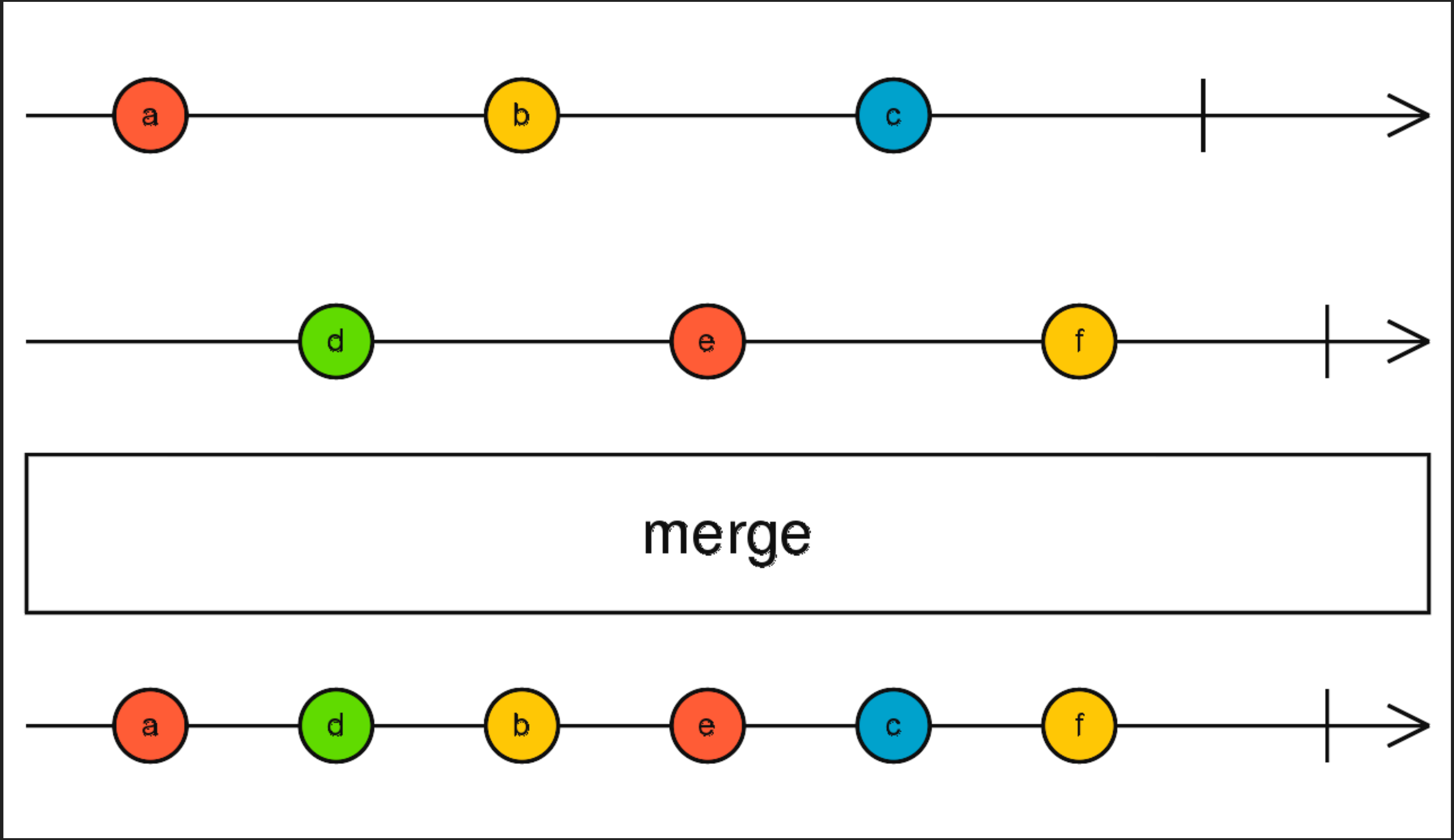
BASIC OBSERVABLE API

- Observable operator: **concatMap**



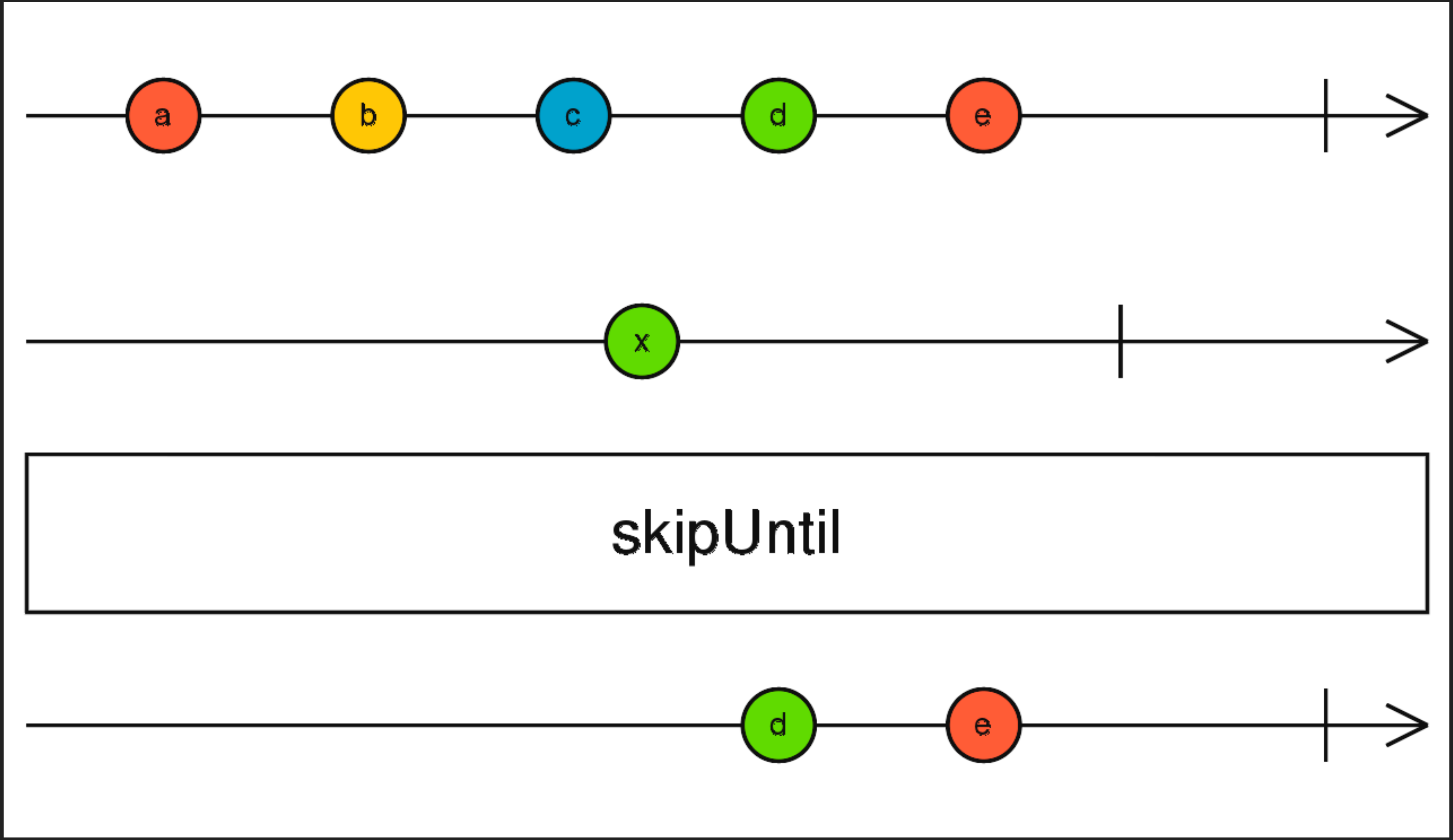
BASIC OBSERVABLE API

- Observable operator: **merge**



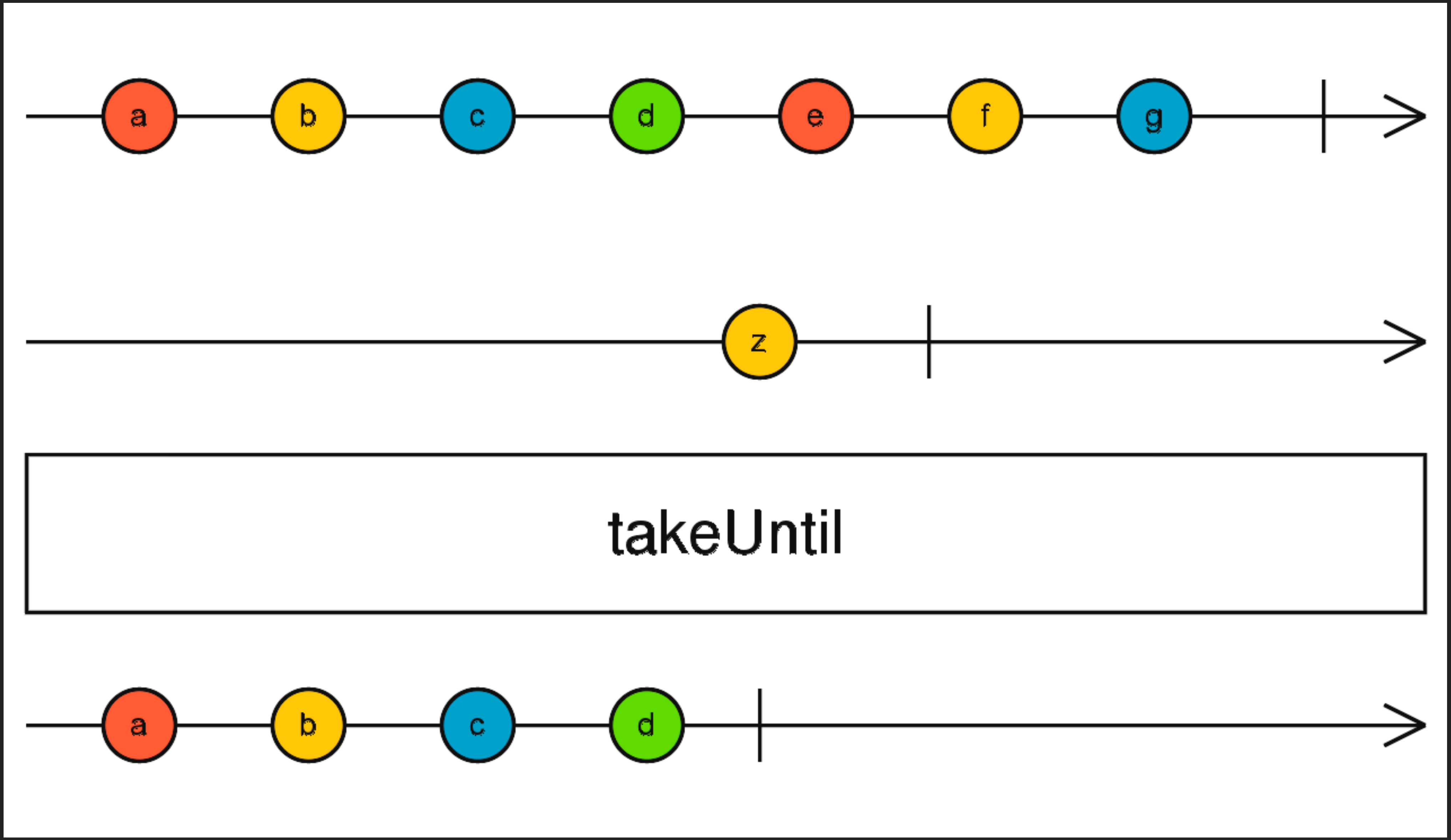
BASIC OBSERVABLE API

- Observable operator: **skipUntil**
  - rxjs.interval(100).skipUntil(**rxjs.fromEvent(document, `click`)**)



BASIC OBSERVABLE API

- Observable operator: **takeUntil**
  - rxjs.interval(100).takeUntil(**rxjs.fromEvent(document, `click`)**)



EXAMPLE: DRAG AND DROP

- Online Example: https://codepen.io/joshblack/pen/zGZZjX
- Emit the div position {x, y} while dragging

EXAMPLE: DRAG AND DROP

-
  ```
  var mouseDown$ = fromEvent(div, “mousedown”)
  var mouseUp$ = fromEvent(div, “mouseup”)
  var mouseMove$ = fromEvent(document, “mousemove”)
  ```

EXAMPLE: DRAG AND DROP

-
  ```
  mouseDown$.switchMap(pressedEvent => {
      return mouseMove$.takeUtil(mouseUp$)
  })
  ```

- It emits **MouseMove event** during dragging

EXAMPLE: DRAG AND DROP

-
  ```
  mouseDown$.switchMap(pressedEvent => {
      return mouseMove$.map(moveEvent => {
          return {
            x: moveEvent.x,
            y: moveEvent.y,
          }
      }).takeUtil(mouseUp$)
  })
  ```

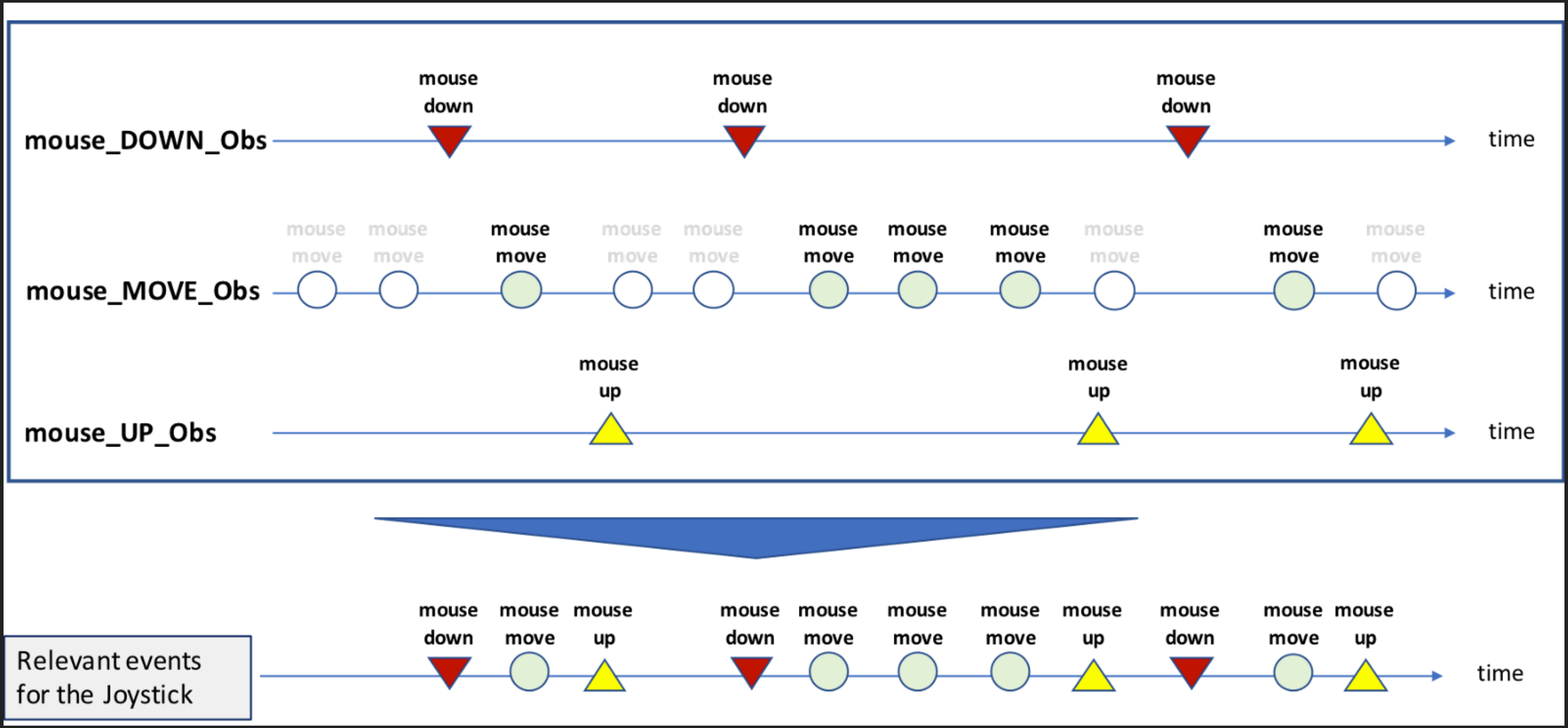
EXAMPLE: DRAG AND DROP

```
mouseDown$.switchMap(pressedEvent => {
    var divStartPosition = {x: div.clientX, y: div.clientY}
    return mouseMove$.map(moveEvent => {
        return {
          x: divStartPosition.x + moveEvent.x - pressedEvent.x,
          y: divStartPosition.y + moveEvent.y - pressedEvent.y,
        }
    }).takeUtil(mouseUp$)
})
```

EXAMPLE: DRAG AND DROP

- Can I change **switchMap** to **concatMap / flatMap** ?
- Answer is **YES**
- Because **mouse-move-then-up** stream must complete before **next mouse-down emits**.

EXAMPLE: DRAG AND DROP



REFERENCES

- RxJS
  https://rxjs-dev.firebaseapp.com/
- ReactiveX
  http://reactivex.io/
- Drag-and-Drop example
  https://varun.ca/drag-with-rxjs/
  https://codepen.io/joshblack/pen/zGZZjX
- Angular RxJS
  https://angular.io/guide/rx-library

Thank You