ANGULAR
ARCHITECTS
INSIDE KNOWLEDGE

Reactive Extensions for JS
Operators

Alex Thalhammer

Outline

- Motivation
- Example
- Transformation Operators
- Filtering Operators
- Combination Operators
- Error Handling
- Higher Order Observables
- Reference

Motivation

Observables vs Promises – Operators

Observables (Streams)	**Promises (Single Event)**
More features	Less powerful
Can emit zero, **one or multiple** values over time.	Emit a **single** value at a time.
Lazy: they're not executed until we subscribe using the subscribe() method.	**Eager**: execute immediately after creation.
Subscriptions are **cancellable** using the unsubscribe() method, which stops the listener from receiving further values.	Are **not cancellable**.
RxJS provides a **ton of functionality** to operate on observables like the map, forEach, filter, reduce, retry, and retryWhen operators.	Don't provide any operations.
Deliver errors to the subscribers.	Push errors to the child promises.
Used by Angular in HTTP Client & Route Params	Used by Angular in Router.navigate

Example

Example with Pipeable Operators

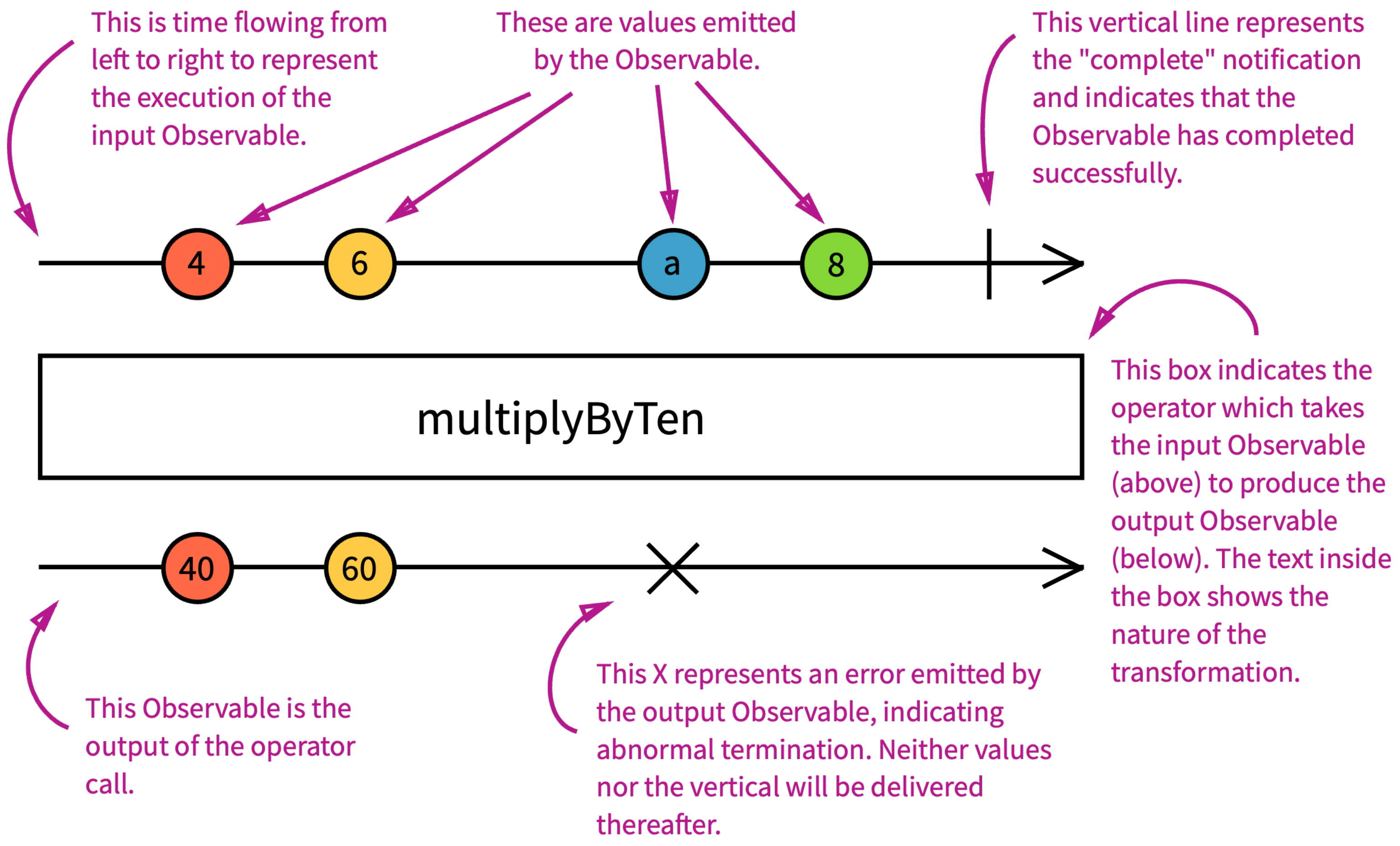
```
import { map } from 'rxjs/operators';

this.httpClient.get<Booking>("http://www.angular.at/api/...")
    .pipe(map((booking) => new Date(booking.date)))
    .subscribe({
        next: (date) => { … },
        error: (err) => { console.error(err); }
        complete: () => { console.log('complete'); }
    });
```

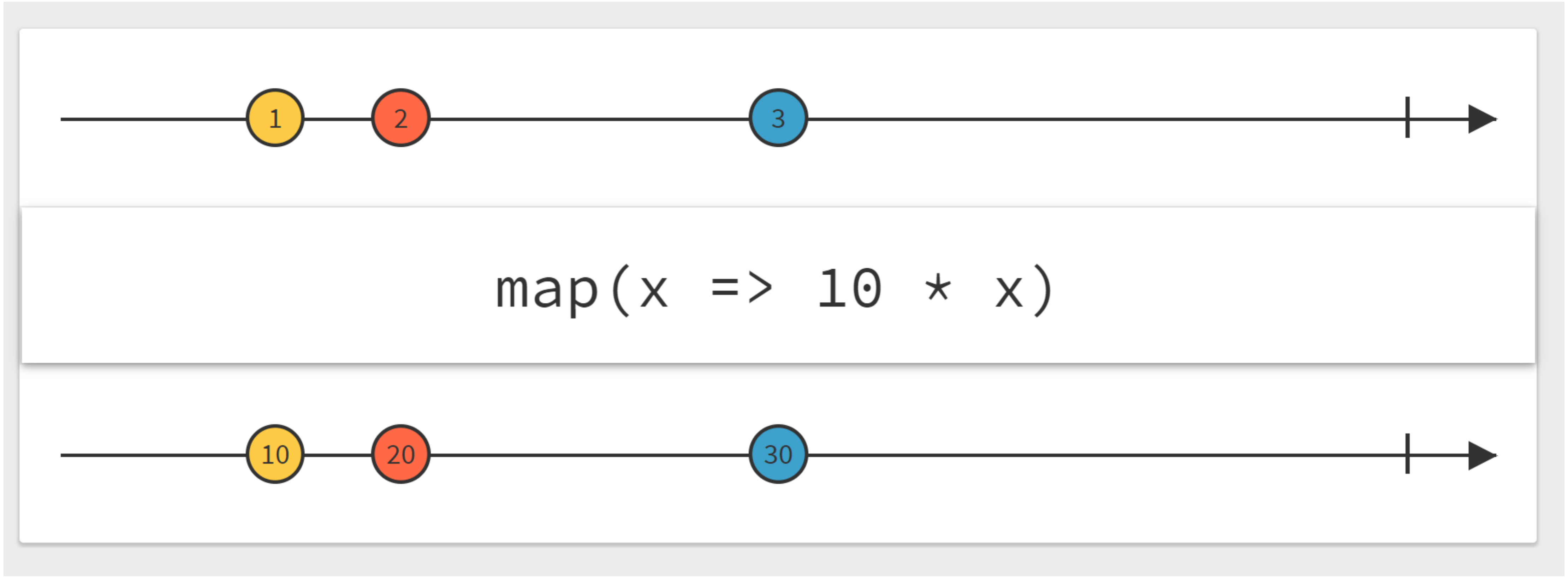
Transformation Operators

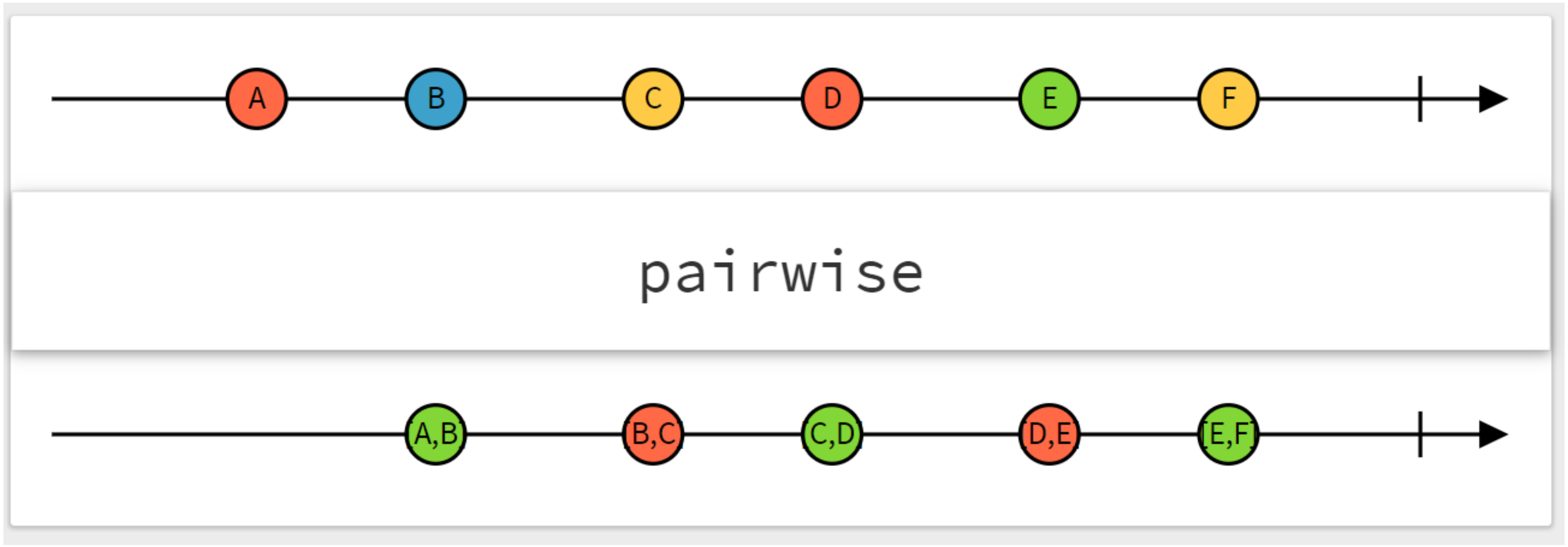
Operators

[https://rxjs.dev/guide/operators]

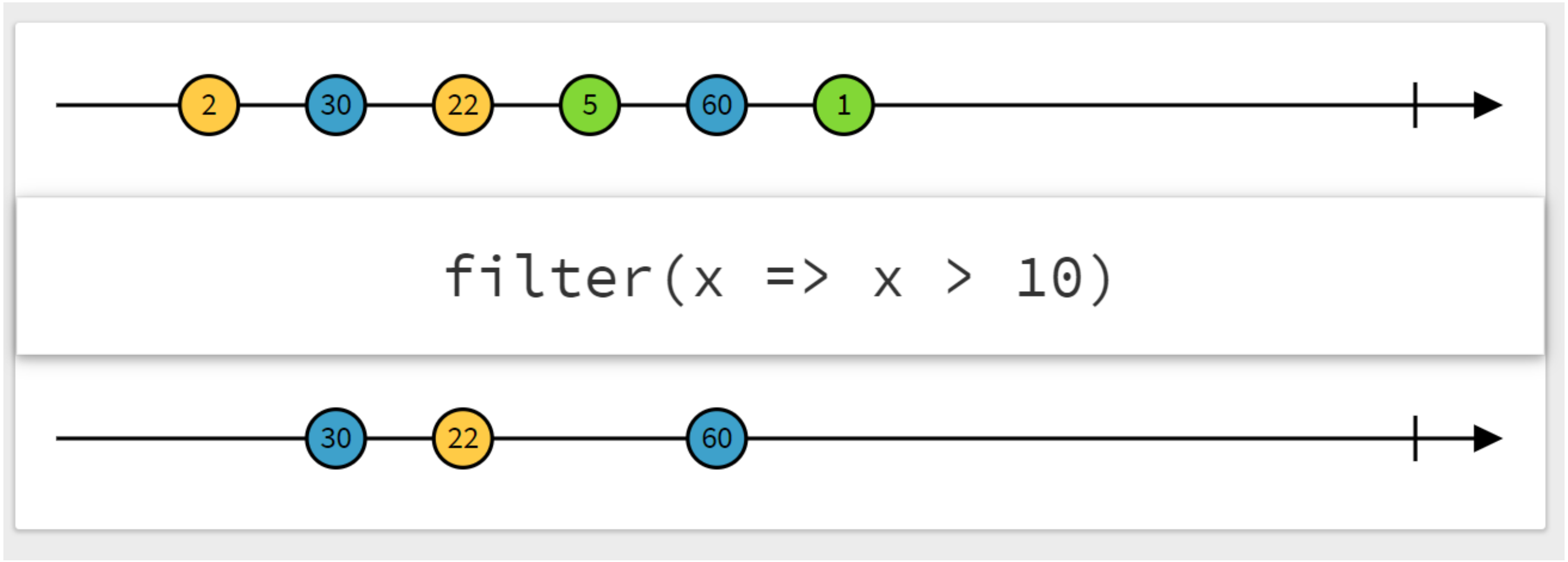

This is time flowing from left to right to represent the execution of the input Observable.
These are values emitted by the Observable.
This vertical line represents the "complete" notification and indicates that the Observable has completed successfully.
4
6
a
8
multiplyByTen
This box indicates the operator which takes the input Observable (above) to produce the output Observable (below). The text inside the box shows the nature of the transformation.
40
60
This Observable is the output of the operator call.
This X represents an error emitted by the output Observable, indicating abnormal termination. Neither values nor the vertical will be delivered thereafter.

Operators

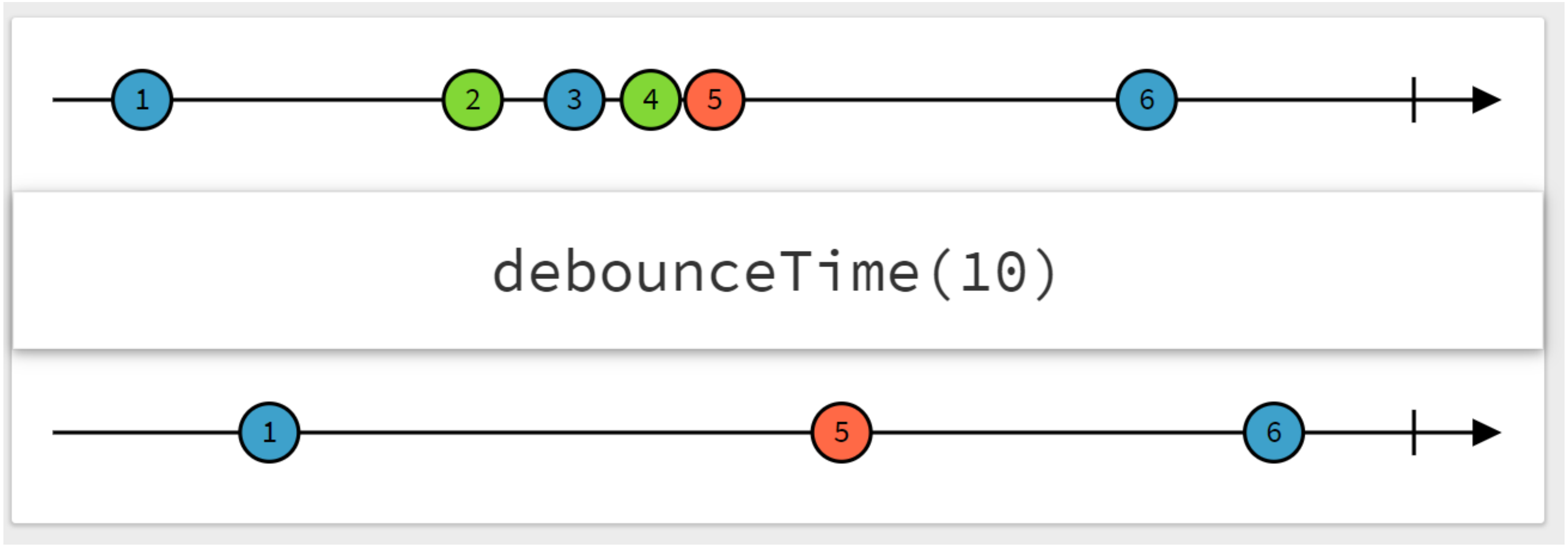


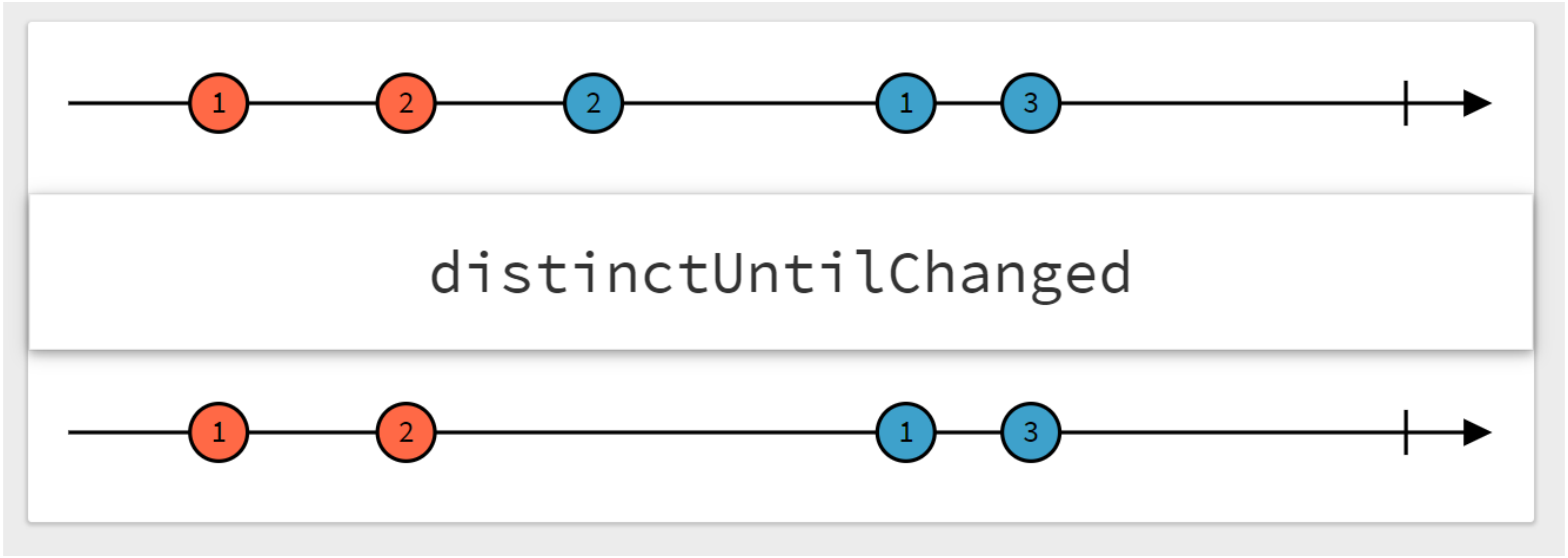

A
B
C
D
E
F
pairwise
A,B
B,C
C,D
D,E
E,F

Filtering Operators



```
filter(x => x > 10)
```


1
2
3
4
5
6
debounceTime(10)
1
5
6

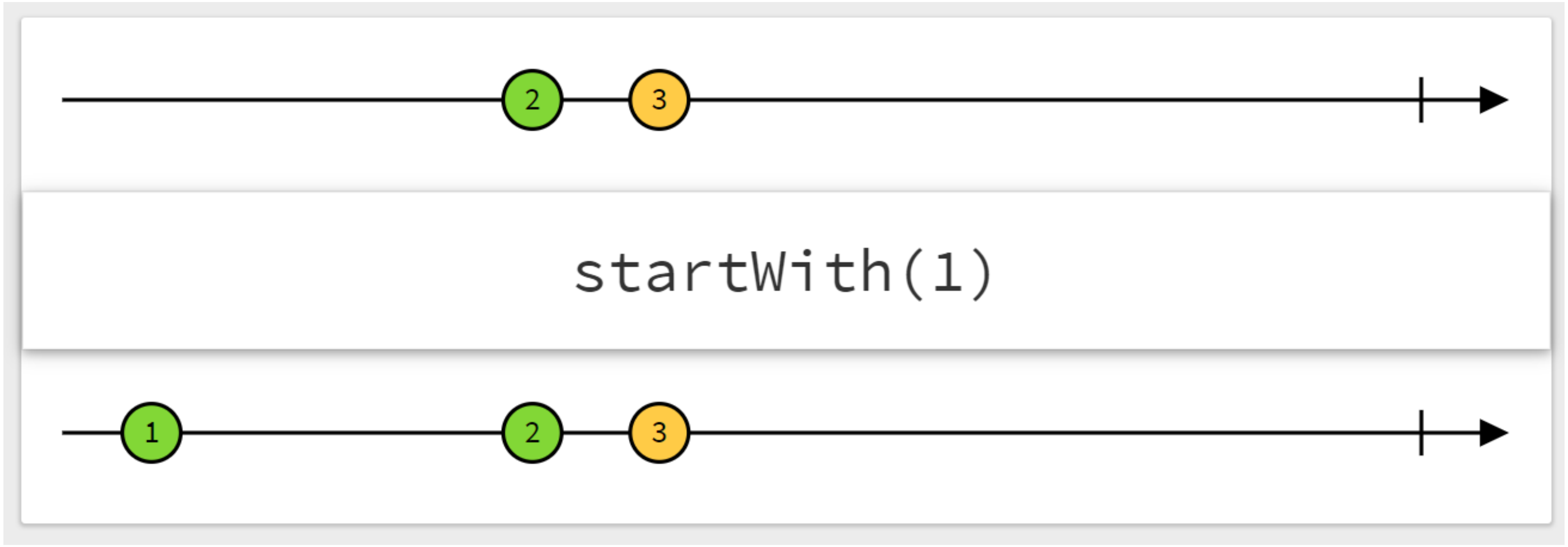

1
2
2
1
3
distinctUntilChanged
1
2
1
3

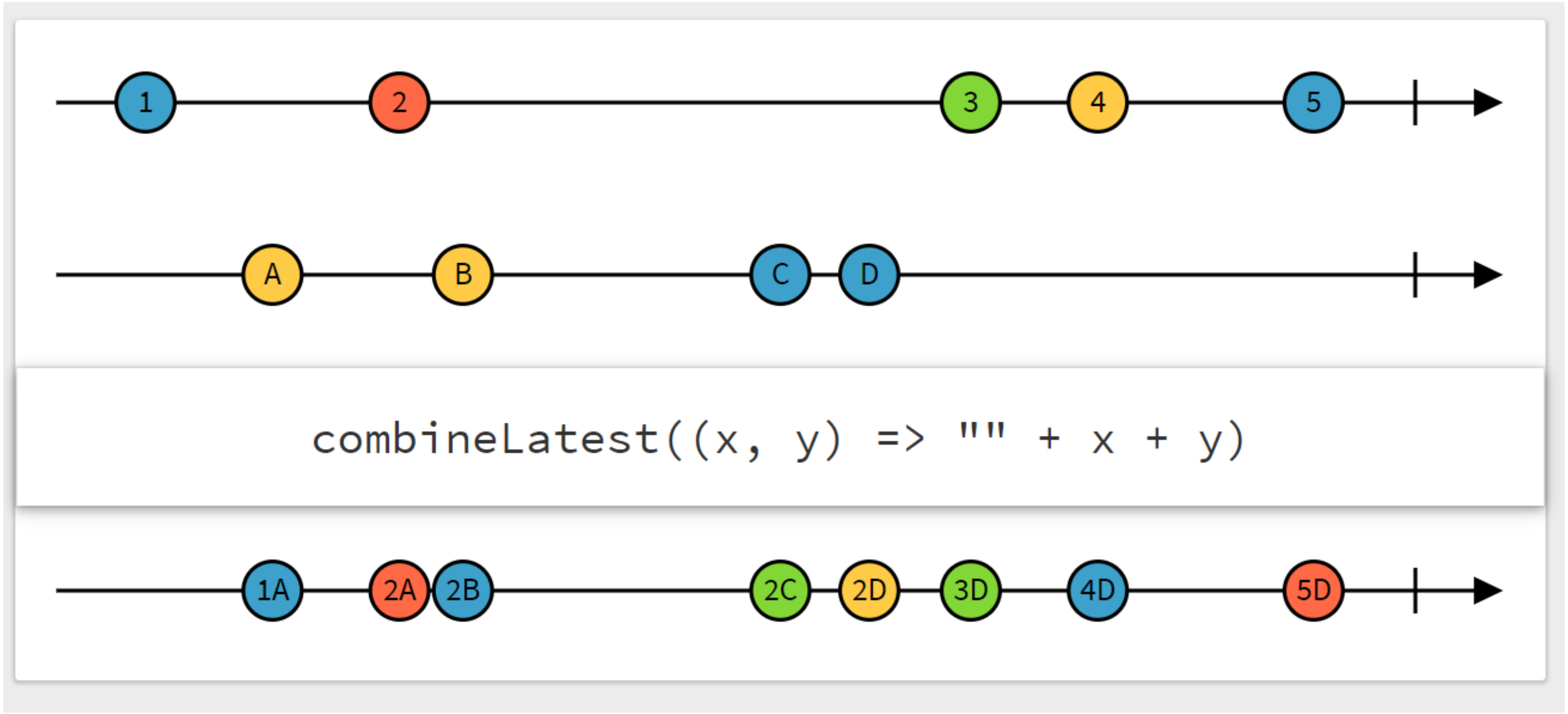
Demo

Simple Lookahead

ANGULAR ARCHITECTS INSIDE KNOWLEDGE SOFTWARE ARCHITECT

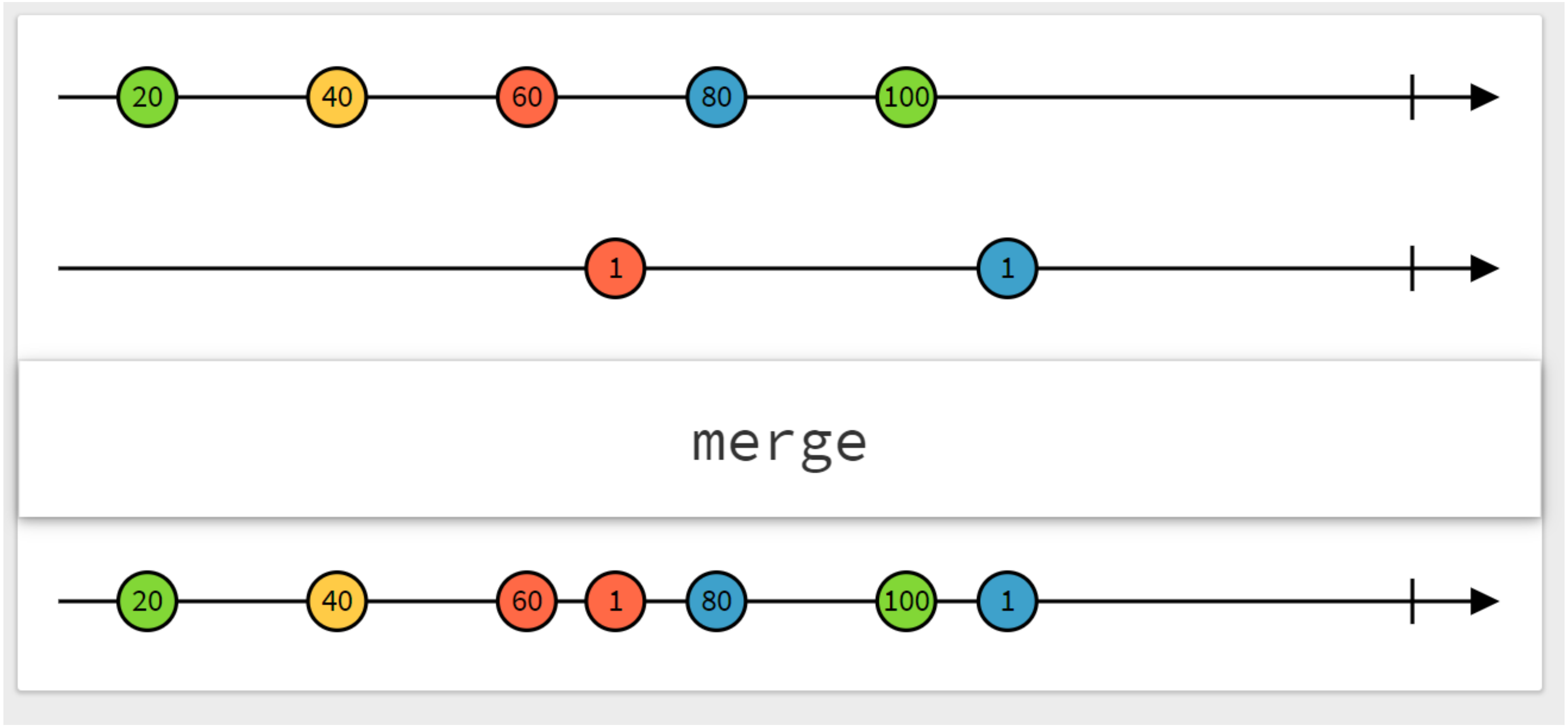
Combination Operators


2
3
startWith(1)
1
2
3


1
2
3
4
5
A
B
C
D
combineLatest((x, y) => "" + x + y)
1A
2A
2B
2C
2D
3D
4D
5D

Digression: combineLatest vs forkJoin

combineLatest	forkJoin
Emits whenever all input $ have emitted at least once.	Emits when all input $ have been completed.
Multiple emits over time.	**Single** emit with last values.
Error will stop the emits.	Error will avoid any emit.
Unsubscribing necessary.	**Unsubscribing** recommended (as almost always).


20
40
60
80
100
1
1
merge
20
40
60
1
80
100
1

Demo

Combine Streams

Error Handling

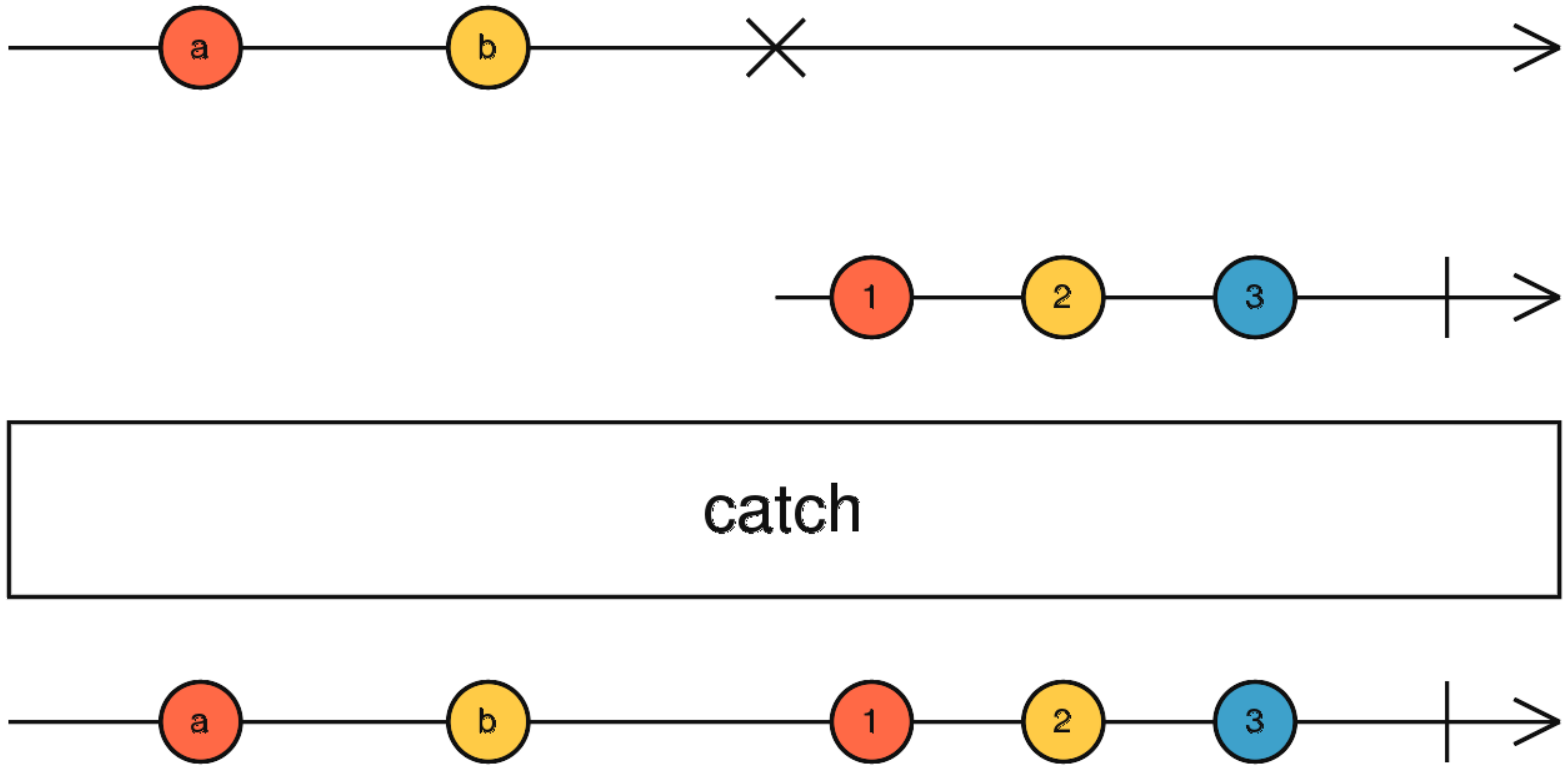
Operators for Error Handling

-
  ```
  catchError((err) => {
    console.log('Error caught:');
    console.log(err);
    return of([]);
  })
  ```

-
  ```
  retry(3)
  ```

-
  ```
  throwError(() => new Error('im an error'))
  ```

-
  ```
  finalize(() => {
    this.isLoading = false;
  })
  ```

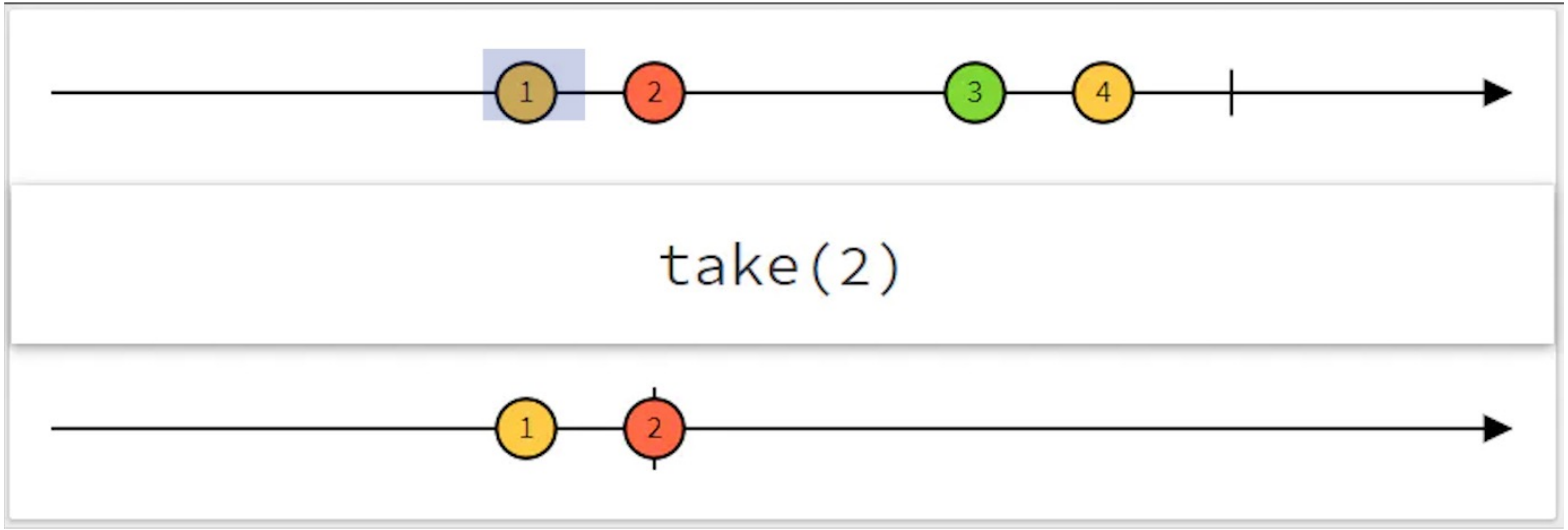

a
b
1
2
3
catch
a
b
1
2
3

Demo

Error Handling

ANGULAR ARCHITECTS INSIDE KNOWLEDGE SOFTWARE ARCHITECT

Taking Operators


1
2
3
4
take(2)
1
2

Taking operators

- take(n)
- first(predicate?: boolean)
- takeUntil(observable) & takeWhile(boolean)

One more thing ☺

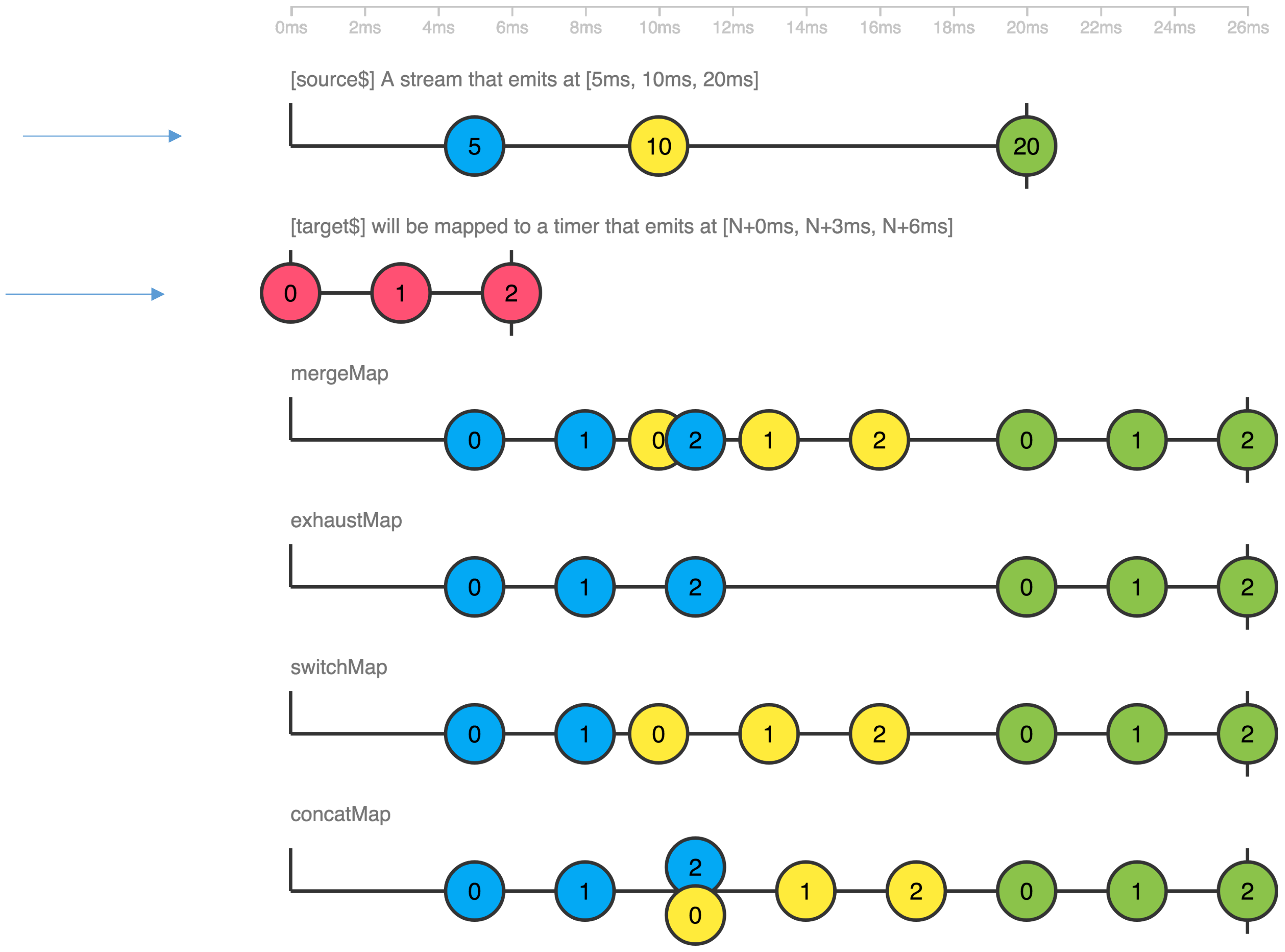
Higher Order Observables

Operators for Higher Order Observables

- mergeMap
  - merges outer (source) and inner observables
- exhaustMap
  - outer is ignored until inner is finished
- switchMap
  - inner will be completed after next outer
- concatMap
  - outer will be sent after inner is finished

from, to

API



0ms 2ms 4ms 6ms 8ms 10ms 12ms 14ms 16ms 18ms 20ms 22ms 24ms 26ms
[source$] A stream that emits at [5ms, 10ms, 20ms]
5
10
20
[target$] will be mapped to a timer that emits at [N+0ms, N+3ms, N+6ms]
0
1
2
mergeMap
exhaustMap
switchMap
concatMap

https://thinkrx.io/rxjs/mergeMap-vs-exhaustMap-vs-switchMap-vs-concatMap/

Lab Time

Custom operators?

- Operators are functions with input and output of type observable
- We can create our own operator function
- To avoid code duplications (DRY as ever)
- There is an example at the end of the lab

Recap

Like RxJS?

- Marble Diagrams
  - http://rxmarbles.com

- Usefull Links
  - https://rxjs.dev/guide/overview
    - https://reactive.how/rxjs/ (Launchpad)
    - https://rxjs-dev.firebaseapp.com/operator-decision-tree (ODT)
  - https://www.learnrxjs.io/
  - https://angular.io/guide/rx-library