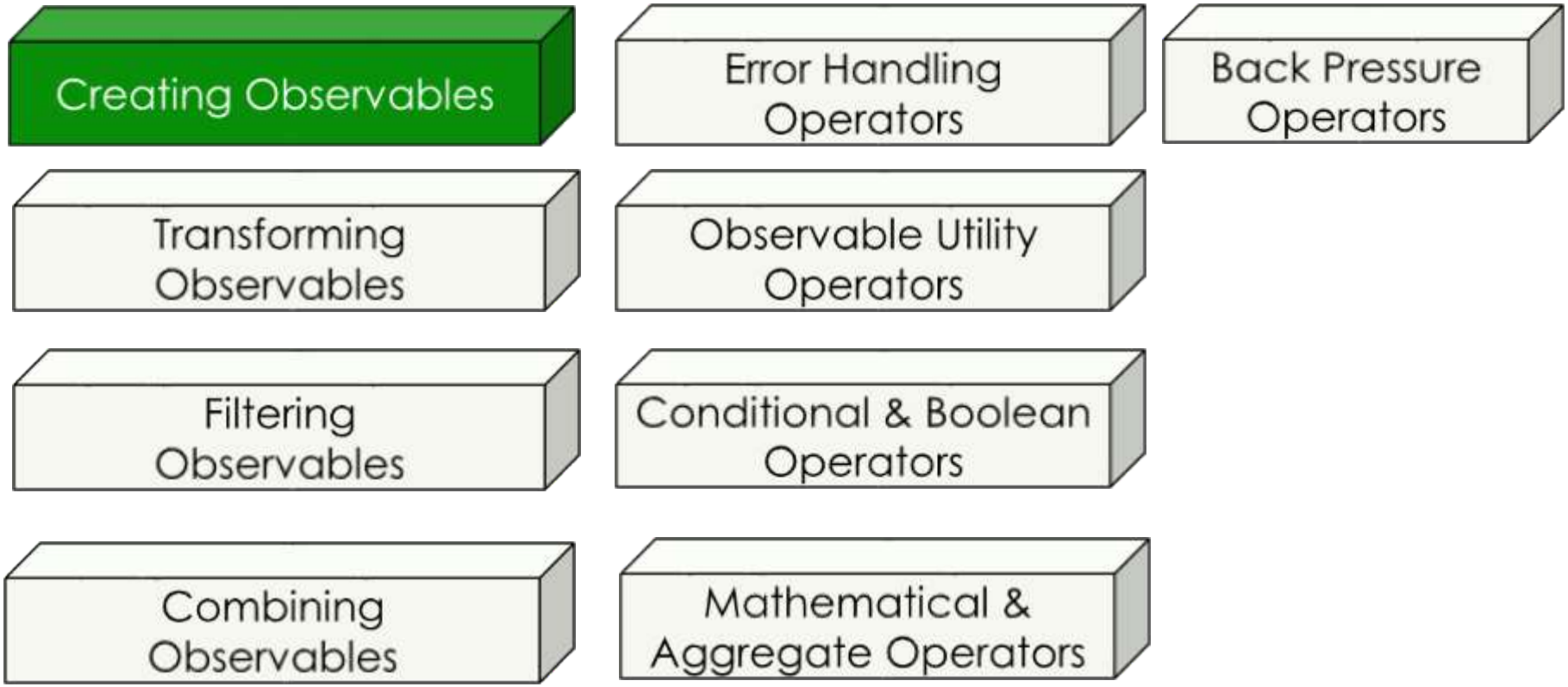
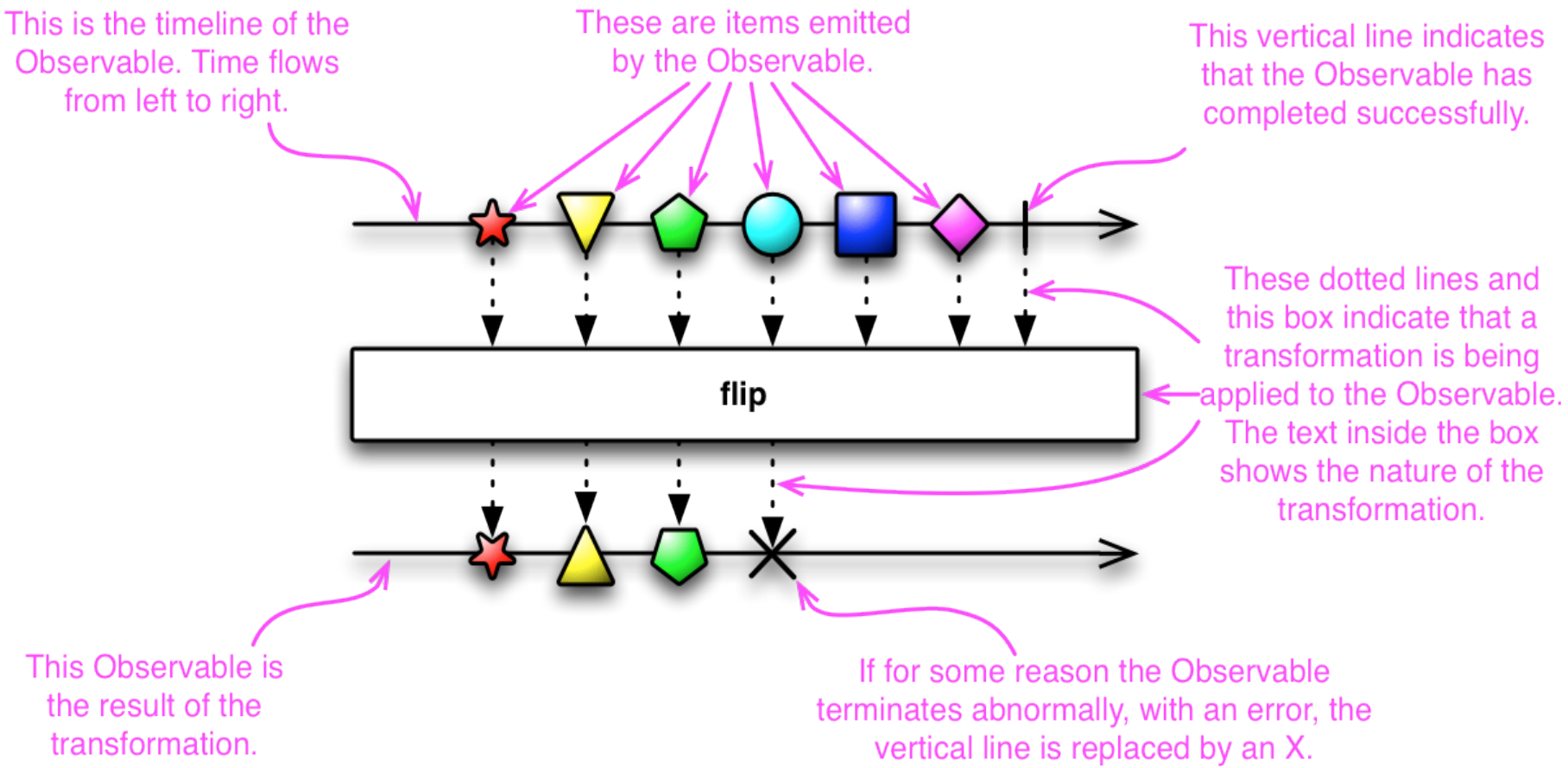
Observable Operator Categories

Operator Categories



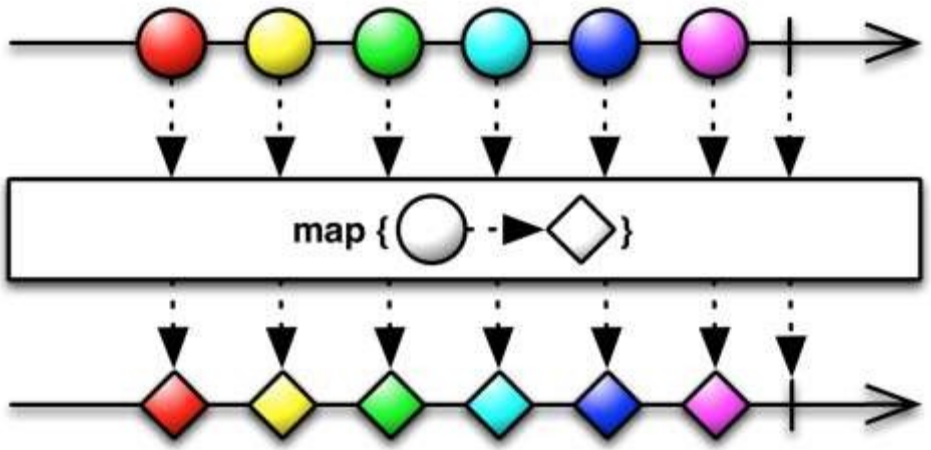
Marble Diagram



Transformational

- **Operations that transform items emitted by an Observable**
- **Commonly used..**
  - **map**
  - **flatmap**

Map

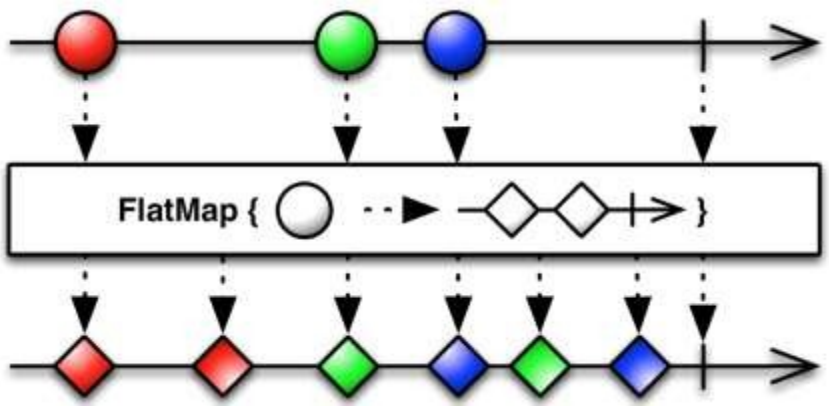


- **Transform the items emitted by an Observable by applying a function to each item**

```
Observable<Integer> o = Observable.range(0,10);

Observable<Integer> timesTen = o.map(t -> t * 10);
```

Flatmap



- **Transforms items emitted from an Observable into Observables and then merges those emissions into a single Observable**

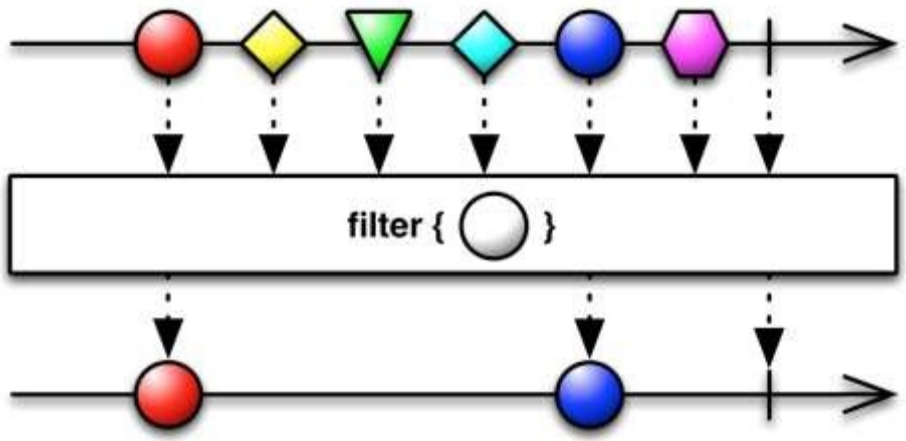
```
Observable<Integer> o = Observable.range(0, 10);

Observable<Integer> numbersAndNegatives
    = o.flatmap(t -> Observable.just(t, -t));
```

Filtering

- **Operators that selectively emit items from a source Observable**
- **Commonly used**
  - **filter**
  - **distinct**
  - **take**
  - **first**

Filter

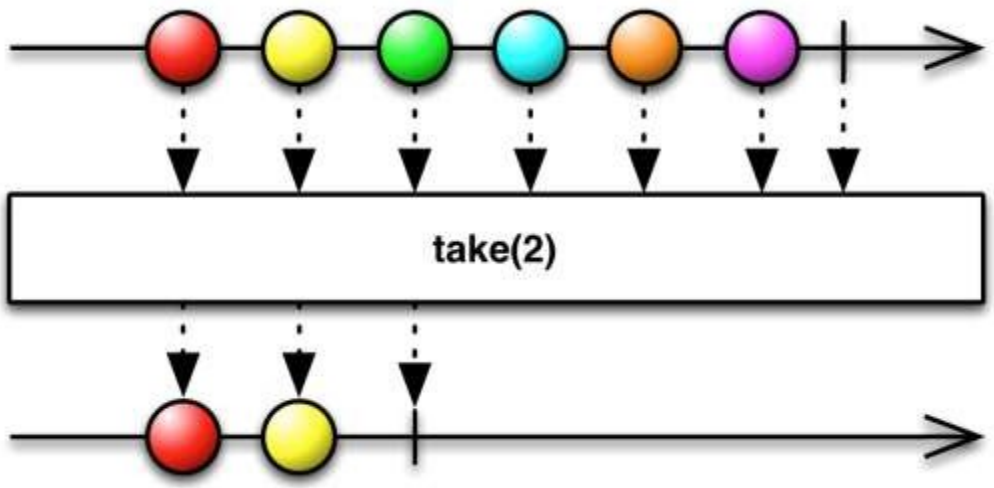


- **Emit only items from an Observable that match a predicate test**

```
Observable<Integer> o = Observable.range(0,10);

Observable<Integer> evenNos = o.filter(t -> (t % 2) == 0);
```

Take



- **Emit only the first *n* items emitted by an Observable.**

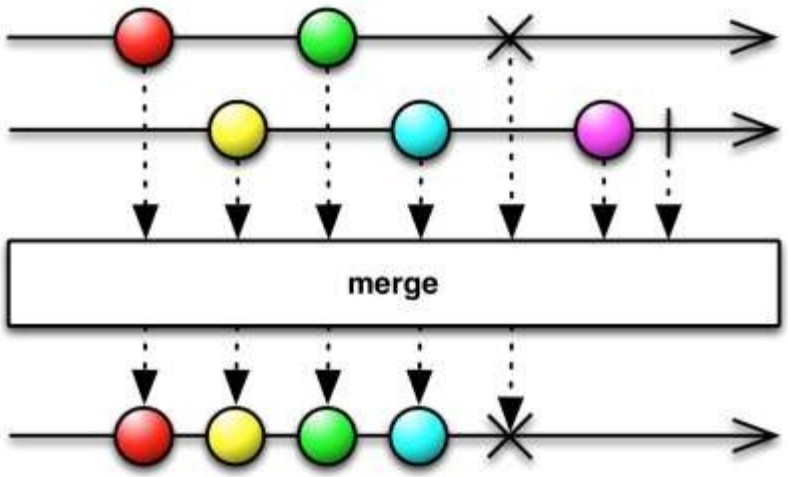
```
Observable<Integer> o = Observable.range(0,10);

Observable<Integer> firstTwo = o.take(2);
```

Combining Observables

- **Operators that work with multiple source Observables to create a single Observable**
- **Commonly used**
  - **Merge**
  - **Concat**
  - **Zip**
  - **zipWith**

Merge

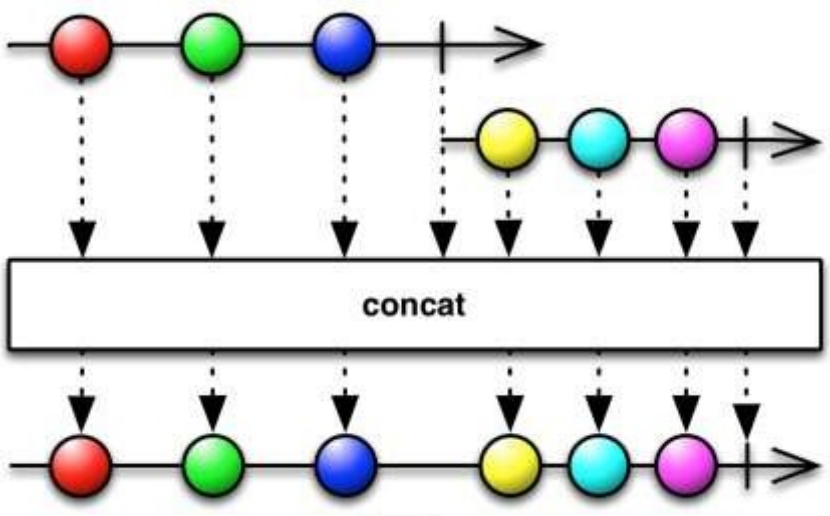


- Combine multiple Observables by merging their emissions
- May interleave items

```
Observable<Integer> first = Observable.range(0,10);
Observable<Integer> second = Observable.range(10, 20);

Observable<Integer> merged
    = Observable.merge(first, second);
```

Concat

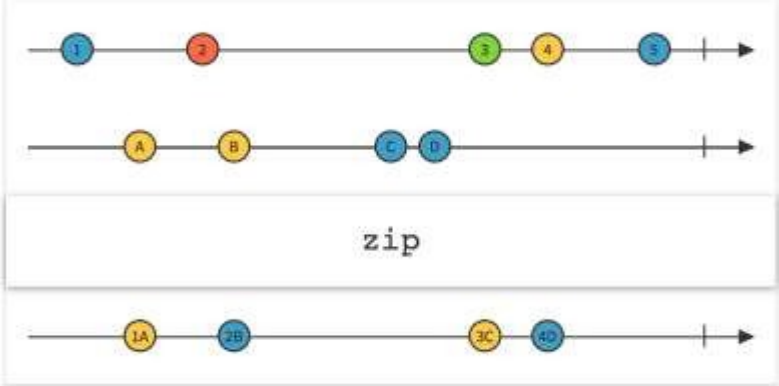


- **Similar to merge but combines observable without interleaving**

```
Observable<Integer> first = Observable.range(0,10);
Observable<Integer> second = Observable.range(10, 20);

Observable<Integer> concat
   = Observable.concat(first, second);
```

Zip



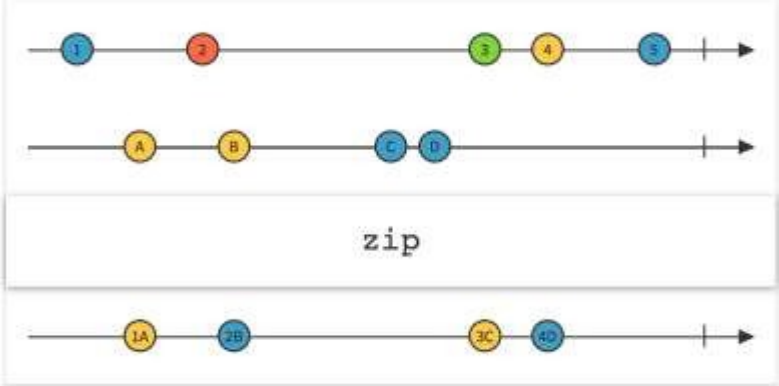
- combine the emissions of multiple Observables together via a specified function and emit single items for each combination based on the results of this function

```
Observable<Integer> first = Observable.just(1, 2, 3);
Observable<String> second = Observable.just("A","B", "C", "D");

Observable<String> zipped =
    Observable.zip(first, second (x,y) -> String.valueOf(x)+y);

// on subscription emits "1A", "2B", "3C"
```

ZipWith



- **Instance-version of zip**

```
Observable<Integer> first = Observable.just(1, 2, 3);
Observable<String> second = Observable.just("A","B", "C", "D");

Observable<String> zipped =
    first.zipWith(second, (x, y) -> String.valueOf(x)+y);

// on subscription emits "1A", "2B", "3C"
```

Decision Tree of Observables

I want to create a new Observable

- ...that emits a particular item: Just
  - ...that was returned from a function called at subscribe-time: Start
  - ...that was returned from an `Action`, `Callable`, `Runnable`, or something of that sort, called at subscribe-time
    - : From
  - ...after a specified delay: Timer
- ...that pulls its emissions from a particular `Array`, `Iterable`, or something like that: From
- ...by retrieving it from a Future: Start
- ...that obtains its sequence from a Future: From
- ...that emits a sequence of items repeatedly: Repeat
- ...from scratch, with custom logic: Create
- ...for each observer that subscribes: Defer
- ...that emits a sequence of integers: Range
  - ...at particular intervals of time: Interval
    - ...after a specified delay: Timer
- ...that completes without emitting items: Empty
- ...that does nothing at all: Never

I want to create an Observable by combining other Observables

- ...and emitting all of the items from all of the Observables in whatever order they are received: Merge
- ...and emitting all of the items from all of the Observables, one Observable at a time: Concat
- ...by combining the items from two or more Observables sequentially to come up with new items to emit

http://reactivex.io/documentation/operators.html#tree

Lab - Combining Observables