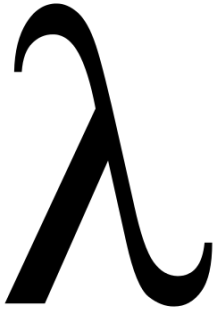
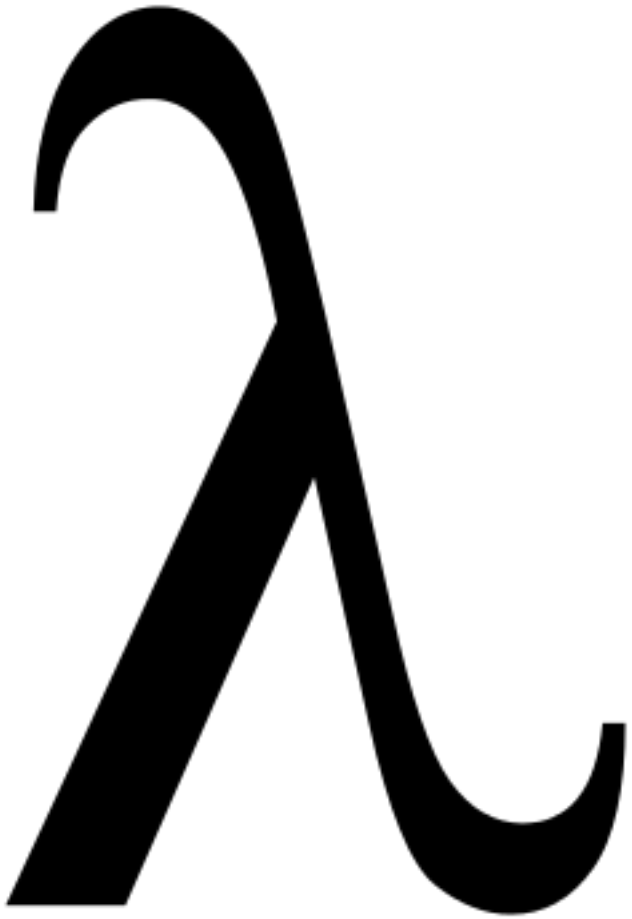
Functional and Reactive Workshop

https://github.com/mattpodwysocki/jsday-workshop-2016



Principal SDE
Open Sourcerer
@mattpodwysocki
github.com/mattpodwysocki

Functional Programming



In functional programming, programs are executed by evaluating *expressions*, in contrast with imperative programming where programs are composed of *statements* which change global *state* when executed. Functional programming typically avoids using mutable state.

Functional programming requires that functions are *first-class*, which means that they are treated like any other values and can be passed as arguments to other functions or be returned as a result of a function. Being first-class also means that it is possible to define and manipulate functions from within other functions

Functional Programming in JS?

```
// Accept a function
function exec (cmd) {
  return cmd();
}

// Return a function
function addPartial(x) {
  return y => x + y;
}

// Create a function
const fn = (x, y) => x + y
```

Going from Imperative...

```
const values = ['1','foo','3,'4','bar'];
let result = 0;
for (let i = 0; i < values.length; i++) {
  let val = parseInt(values[i], 10);
  if (!Number.isNaN(val) {
    result += val;
  }
}
console.log(result);
```

To Functional...

```
const values = ['1','foo','3,'4','bar'];
let result = values
   .map(x => parseInt(x, 10))
   .filter(x => !Number.isNaN(x))
   .reduce((sum, x) => sum + x);

console.log(result);
```

Reactive Programming

What is Reactive Programming?

Merriam-Webster defines reactive as *"readily responsive to a stimulus"*, i.e. its components are "active" and always ready to receive events. This definition captures the essence of reactive applications, focusing on systems that:

react to events

the event-driven nature enables the following qualities

react to load

focus on scalability by avoiding contention on shared resources

react to failure

build resilient systems with the ability to recover at all levels

react to users

honor response time guarantees regardless of load

Reactive Programming

===

Collections + Time

From Functional...

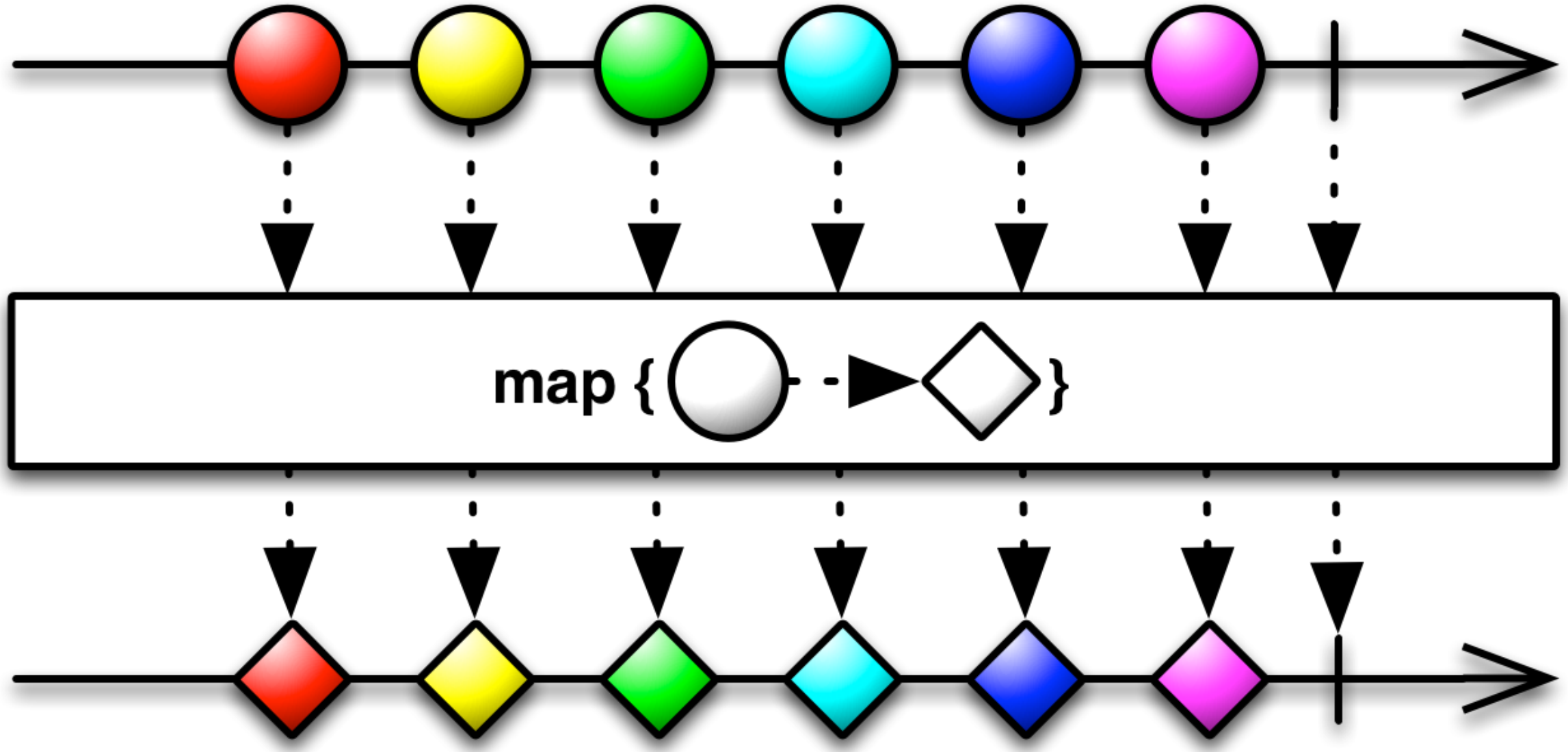
```
const values = ['1','foo','3,'4','bar'];
const result = values
  .map(x => parseInt(x, 10))
  .filter(x => !Number.isNaN(x))
  .reduce((sum, x) => sum + x);

console.log(result);
```

To Reactive...

```
const values = ['1','foo','3,'4','bar'];
let result = Rx.Observable.from(values)
   .delay(1000)
   .map(x => parseInt(x, 10))
   .filter(x => !Number.isNaN(x))
   .reduce((sum, x) => sum + x);
let subscription = result.subscribe(
  x => console.log(x)
);
```

Functional Programming with Collections

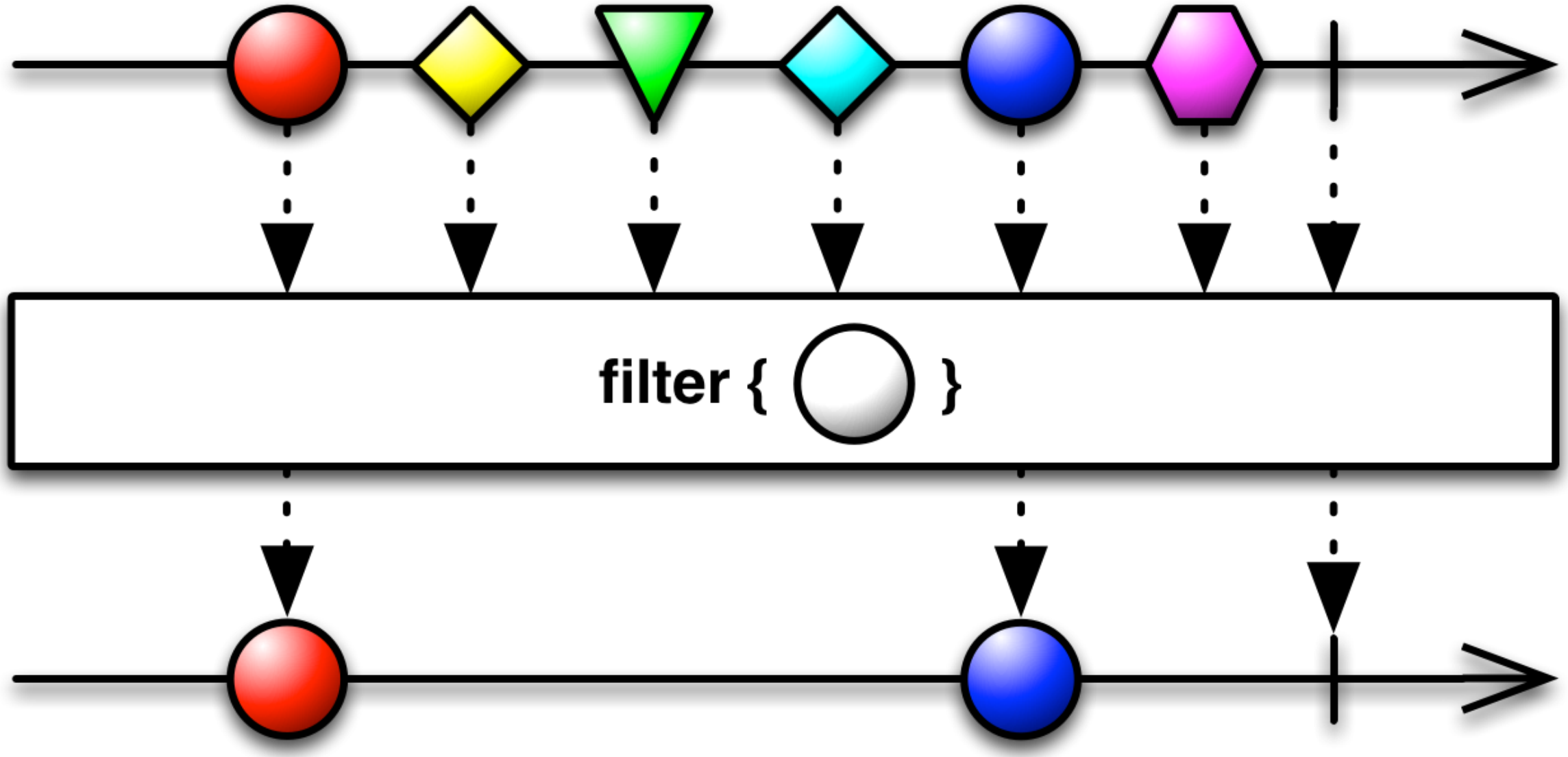

map {

map(selector)

```
const values = [1,2,3];

const result = values.map(x => x * x);

console.log(result)
// => [1,4,9]
```

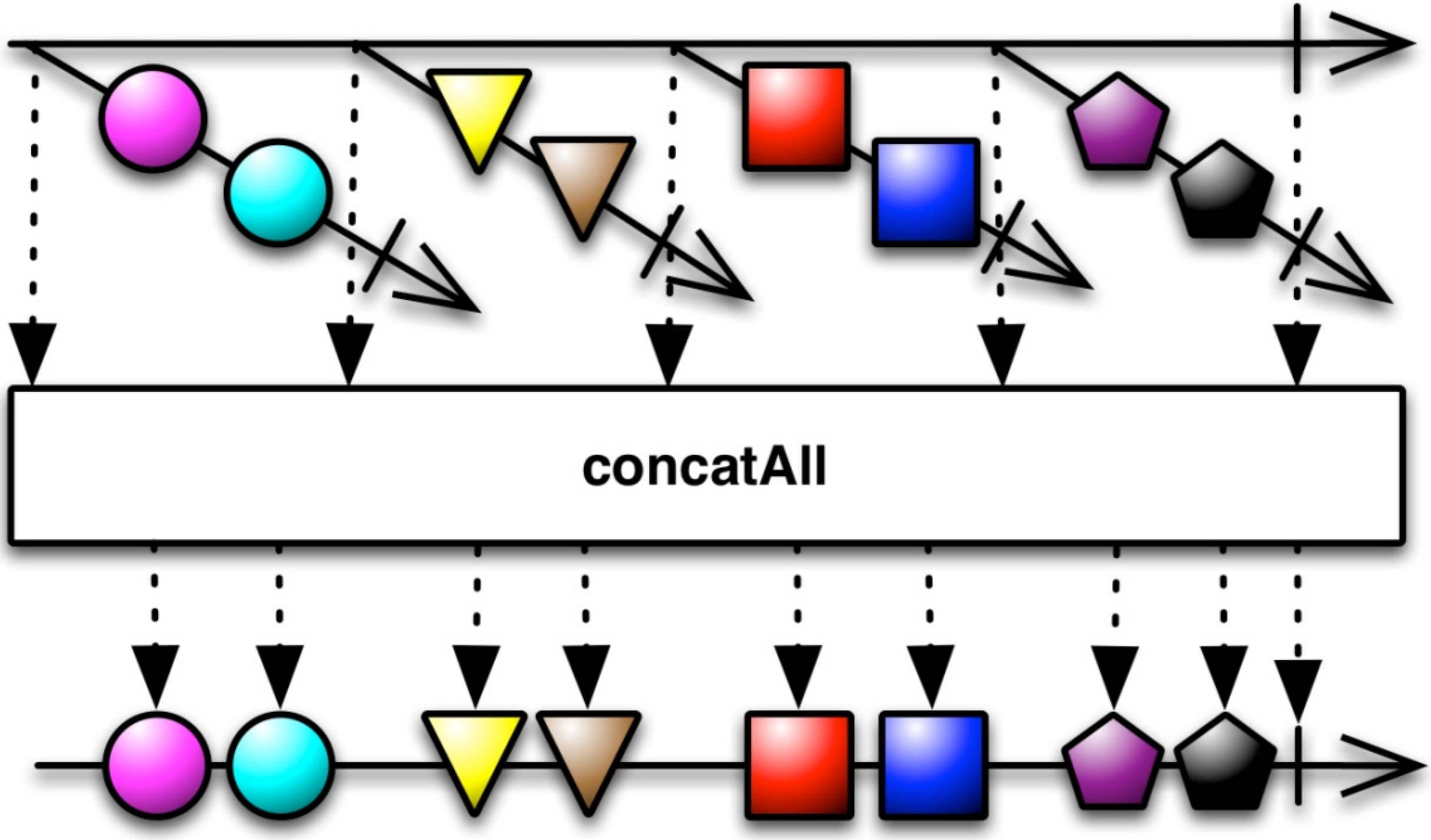

filter { }

filter(predicate)

```
const values = [1,2,3];

const result = values.filter(
  x => x % 2 !== 0
);

console.log(result)
// => [1,3]
```

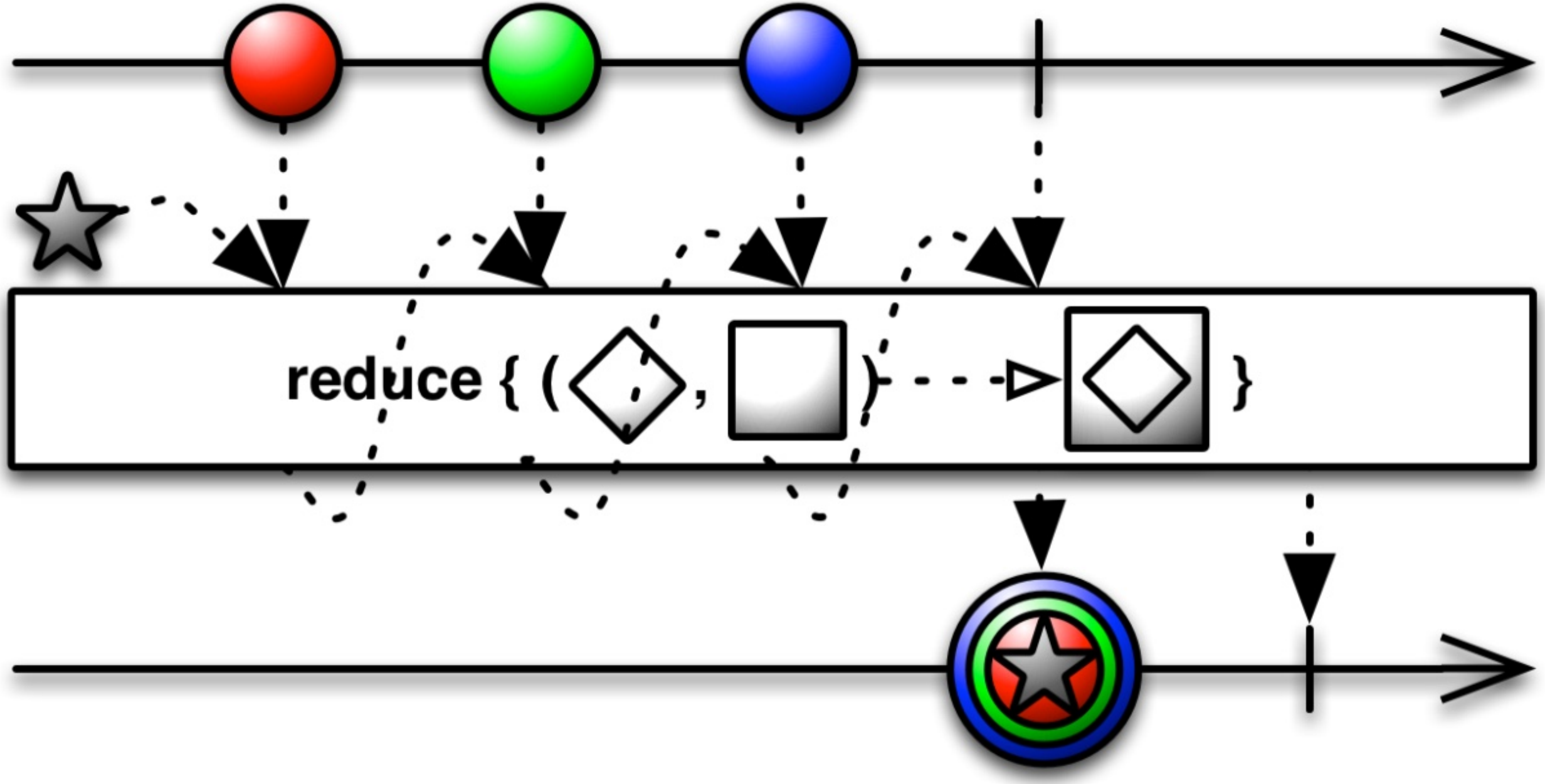

concatAll

concatAll()

```
const values = [
  [1,2,3],
  [4,5,6],
  [7,8,9]
];

const result = values.concatAll();

console.log(result);
// => [1,2,3,4,5,6,7,8,9]
```

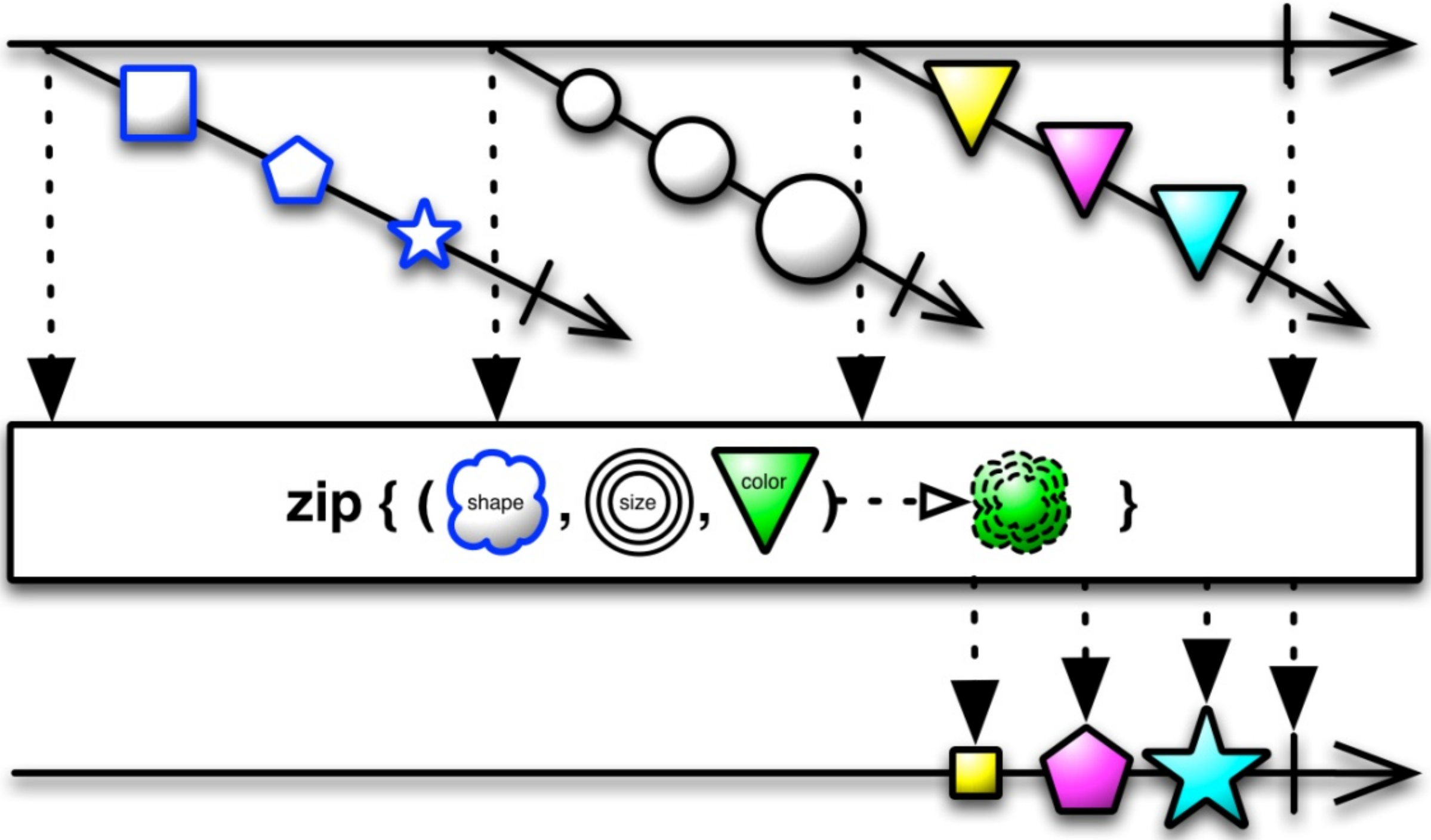

reduce { (,) - - - } }

reduce(reducer [, seed])

```
const values = [1,2,3];

const result = values.reduce(
  (acc, x) => acc + x,
  0
);

console.log(result)
// => 6
```

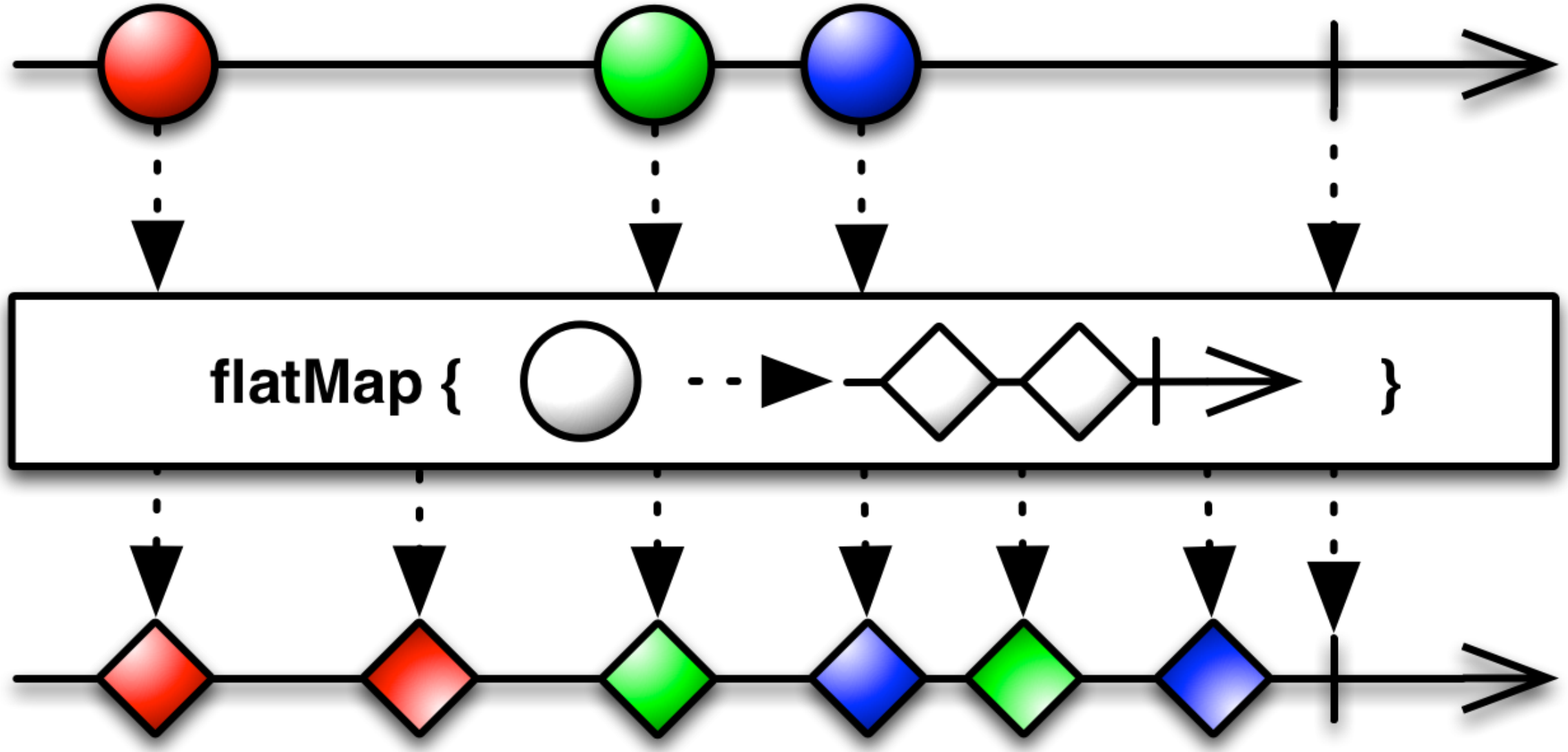

zip { (shape , size , color) - -> }

zip(args [, combiner])

```
const values1 = [1,2,3];
const values2 = ['a','b','c'];

const result = values1.zip(
  values2,
  (x, y) => [x, y]
);

console.log(result);
// => [[1,'a'],[2,'b'],[3,'c']]
```


flatMap {
}

flatMap(selector)

```
const values = [1,2,3];

const result = values.flatMap(
  x => [x + 1]
);

console.log(result);
// => [2,3,4]
```

This is an interactive learning course with exercises you fill out right in the browser. **If you just want to browse the content click the button below**:

Show all the answers so I can just browse.

Functional Programming in Javascript

Functional programming provides developers with the tools to abstract common collection operations into reusable, composable building blocks. You'll be surprised to learn that most of the operations you perform on collections can be accomplished with **five simple functions**:

1. map
2. filter
3. concatAll
4. reduce
5. zip

Here's my promise to you: if you learn these 5 functions your code will become shorter, more self-descriptive, and more durable. Also, for reasons that might not be obvious right now, you'll learn that these five functions hold the key to simplifying asynchronous programming. Once you've finished this tutorial you'll also have all the tools you need to easily avoid race conditions, propagate and handle asynchronous errors, and sequence events and AJAX requests. In short, **these 5 functions will probably be the most powerful, flexible, and useful functions you'll ever learn.**

http://reactivex.io/learnrx/

Reactive Programming

React to Load

Lossy Operators

- **Debounce**
  (debounce, debounceTime)
- **Sample**
  (sample, sampleTime)
- **Throttle**
  (throttle, throttleTime)

Lossless Operators

- **Backpressure**
  (controlled, pausable, pausableBuffered)
- **Buffer**
  (buffer, bufferCount, bufferTime, bufferTimeOrCount)
- **Window**
  (window, windowCount, windowTime, windowTimeOrCount)

React to Failure

Error Handling

- Catch
- OnErrorResumeNext
- Retry
  (retry, retryWhen)

RxMarbles

Interactive diagrams of Rx Observables

TRANSFORMING OPERATORS

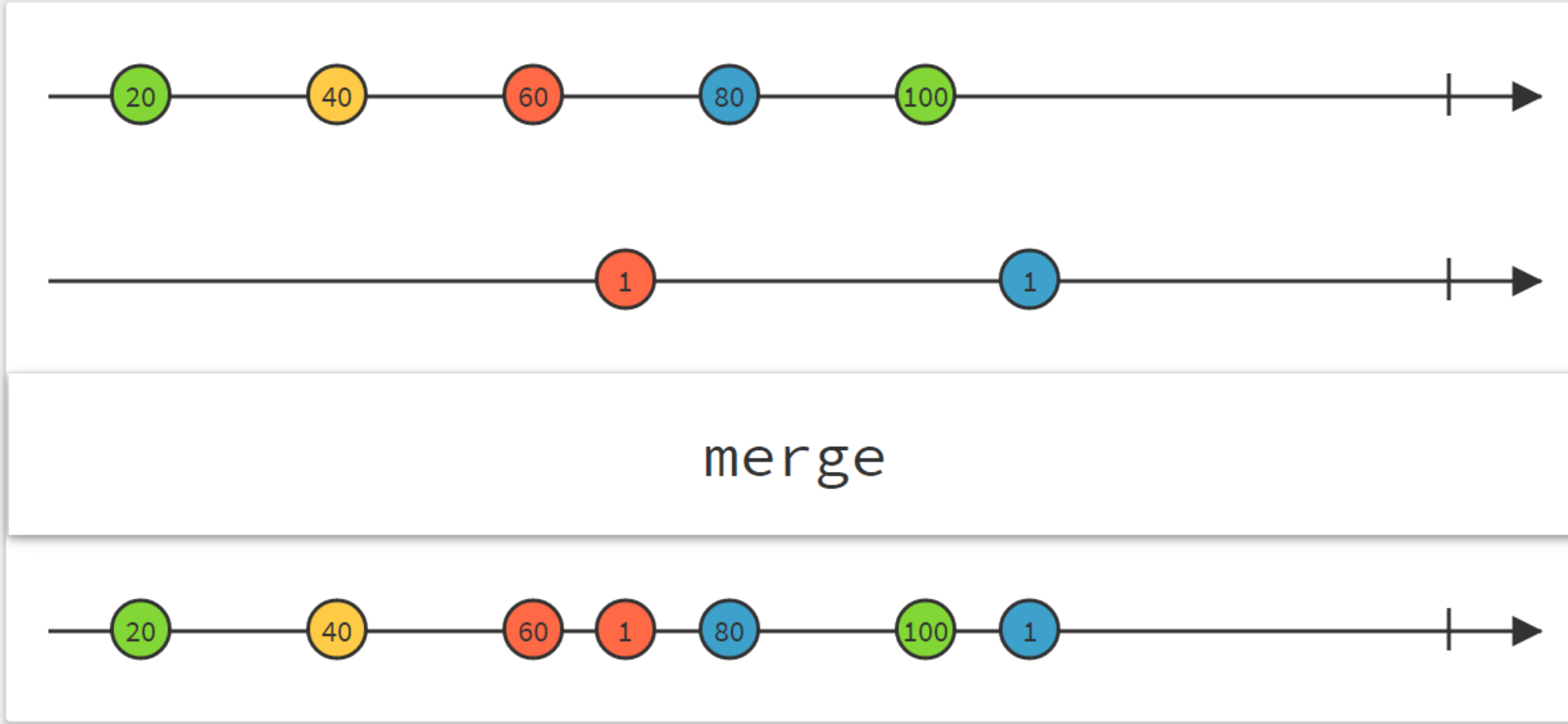
- delay
- delayWithSelector
- findIndex
- map
- scan
- throttle
- throttleWithSelector

COMBINING OPERATORS

- combineLatest
- concat
- merge
- sample
- startWith
- zip

FILTERING OPERATORS

- distinct
- distinctUntilChanged
- elementAt
- filter



http://rxmarbles.com/

Reactive Demos

Reactive applications in practice

Go Build Something Amazing!

https://github.com/mattpodwysocki/jsday-workshop-2016