



REACTIVE
PROGRAMMING
IN JAVASCRIPT

@ahomu / 2015.02.21 - Frontrend Final Conference

- 0 +

```
// カウンター値の保持
var current = 0;

// click されたら...
plusEl.addEventListener('click', function(e) {
  // current を加算/減算して...
  current++;
  // innerHTML を更新する
  counterEl.innerHTML = current;
});

minusEl.addEventListener('click', function(e) {
  current--;
  counterEl.innerHTML = current;
});
```

```
// click で 1/-1 が流れるストリーム
var plus = getClickStream(plusEl).map(1);
var minus = getClickStream(minusEl).map(-1);

// 1/-1 のストリームをひとつにマージ
var source = plus.merge(minus);

// ストリームから値が来ると現在値(current)を計算
var current = source.scan(0, function(acc, v) {
    return acc + v;
});


// current の変更を subscribe して innerHTML を更新
current.subscribe(function(v) {
    counterEl.innerHTML = v;
});
```

佐藤 歩 Ayumu Sato

- ~~Web Frontend~~ Android Engineer
- <http://aho.mu>
- HTML5 Experts 幽霊部員

今日のアジェンダ

- What is **Reactive** Programming?
- Reactive in Frontend **JavaScript**
- **FRP** with Reactive Extensions

A person in silhouette is looking at a large screen displaying a presentation slide. The slide features a title and technical diagrams. The person is holding a small device, possibly a tablet or smartphone, in their hands.

What is Reactive Programming?

イベントや値の関係性に 注目したパラダイム

- イベントや値の関係性 = データフロー
- データフローの宣言を元に変更を自動的に伝播させる
- $c = a + b$ なら c が示す値は a , b と共に変わって欲しい


```
current = accumulate(merge(plus, minus))
```

```
var plus    = getClickStream(plusEl).map(1);  
var minus   = getClickStream(minusEl).map(-1);  
var source  = plus.merge(minus);  
  
var current = source.scan(0, function(acc, v) {  
    return acc + v;  
});
```

手続きの記述 → 関係性の宣言

データがどのような関係なのかに集中する

Reactive なコードは 宣言的なデータフローを示す

- 実装はコーディング時の考え方とライブラリに依存する
- OOP な言語やデザインパターンでも実現はできる
- **Reactive っぽい = データの関係性を宣言的に表しているか**

RP 界限はノイズが多い

- いわゆる「広義」と「狭義」すら境界が曖昧
- Reactive っぽさ (特に自動伝播) の手段は色々ある
- 各手段が Reactive を自称したりしなかったりする
- ref. [Reactive Porn - steps to phantasien](#)

① Actor Model

- Erlang Actor
- Akka(Scala) Actor
- 並列・分散な処理システムにおける数学モデル
- 非同期な並列処理をいいカンジにする (乱暴)

② Functional Reactive

- Haskell FRP Libraries
- Microsoft **Reactive Extensions**
- 関数型のパラダイムを取り込んだ RP
- GUI 開発で主流なのはこっち方面

③ The Reactive Manifesto

- Reactive な Application の特性を述べた文書

Organizations are independently discovering patterns for building software that look the same. These systems are more

resilient and better positioned to meet modern demands.

These days, because application requirements have changed dramatically in recent years. Only a few years ago a large application had tens

of millions of users, massive storage and processing requirements, and gigabytes of data. Today applications are deployed on everything from mobile devices to supercomputers.

Users expect immediate responses, and 100% uptime. Data is measured in petabytes. Front-end architectures are

We believe that a coherent approach to systems architecture is needed, and we believe that all necessary aspects are already recognized individually: we want systems that are Responsive, Resilient, Elastic and Message Driven. We call these Reactive Systems.

Reactive Streams

Reactive Streams is an initiative to provide a standard for asynchronous stream processing with non-blocking back pressure on the JVM.

④ Reactive Streams

The Problem

Handling streams of data—especially “live” data whose volume is not

- Actor と Reactive Extensions の交差点が Reactive Streams
- Node の Stream は似てるけど今の所は関係なさそう
- ref. 2014 akka-streams-tokyo-japanese
- ref. 非同期ストリーム処理の標準化を目指す "Reactive Streams"

an asynchronous boundary—think passing elements on to another thread or thread pool—while ensuring that the receiving side is not forced to buffer arbitrary amounts of data. In other words, back pressure is an integral part of this model in order to allow the queues which mediate between threads to be bounded. The benefits of asynchronous processing would be negated if the communication of back pressure were synchronous (see also the [Reactive Manifesto](#)), therefore care has been taken to mandate fully non-blocking and asynchronous behavior of all aspects of a Reactive Streams implementation.



Reactive in Frontend JavaScript

JavaScript で UI を操作する

- API からデータを受け取って HTML に反映
- ユーザー入力を受け取って HTML に反映
- イベントを受け取って HTML に反映

API からデータを受け取って HTML に反映

```
$.getJSON('http://example.com/api', function(res) {  
  var html = '';  
  res.list.forEach(function(item) {  
    html += '<li>' + item.title + '</li>'  
  });  
  listEl.innerHTML = html;  
});
```

ユーザー入力を受け取って HTML に反映

```
buttonEl.addEventListener('click', function() {  
    targetEl.innerHTML = 'clicked!';  
}, false)
```

イベントを受け取って HTML に反映

```
bbView.listenTo(bbModel, 'change', function(model) {  
    view.render(model);  
});
```

非同期で変化する状態を管理して
都度 UI に反映するコードを
書くのはダルいこと

すごくダルい、**本当にダルい**

Reactive な仕組みが 自然と求められる

Reactive = 片方の変化を他方に自動で伝播する仕組み

これまでの JavaScript の
中にみる Reactive

Observer パターンは 初歩的な解決のひとつ

- プログラムとして動いている仕組みは同じ
- Reactive 的には Observer パターンを隠蔽して宣言したい
- かつては Reactive な在り方のひとつだったのかも(?)

データバインディングは 局所的な Reactive

- 何らかの仕組みで View と Model を宣言的に結びつける
- 局所的だが GUI におけるリアクティブ要素のひとつ
- ref. [データバインディングとリアクティブプログラミング](#)

React with Flux は Reactive なデータフロー

- データを DOM へ効率よく伝播させるのはRPっぽい
- 1方向データバインディングを実現する手段の一種
- 細かい異論はある: ref. [Don't React - webbisauna 14](#)

人類によるこれまでの抵抗

1. コールバックを避けた非同期処理の抽象化

Promise 一族, Async.js などですたくなるよう努める

2. UI への反映を容易にする抽象化

Data Bindings や Templating (vdom含) で雑にする

Promise による非同期処理の抽象化

```
// single
promise.then(function(res) { andDo(res); });

// parallel and chain
Promise
  .all([
    promiseFn1(), promiseFn2(), promiseFn3()
  ])
  .then(function(results) {
    return promiseFn4(results);
  })
  .then(function(result) { andDo(result); });
```

データバインディングによる抽象化

Search: `<input ng-model="query">`

Sort by:

`<select ng-model="orderProp">`

`<option value="name">Alphabetical</option>`

`<option value="age">Newest</option>`

`</select>`

`<ul class="phones">`

`<li ng-repeat="phone in phones | filter:query | orderBy:orderProp">`

`{{phone.name}}`

`<p>{{phone.snippet}}</p>`

``


``

しかし

- Promise (などの Flow Control)
 - Click のように離散的なイベントは扱えない
- Data Bindings
 - View と Model を結びつけるための局所的 Reactive

非同期とか気にせず
すべて同じように扱って
何でも Reactive に宣言したい

そんな夢を叶えるモデルが、5年以上前からあります



FRP with Reactive Extensions

関数型プログラミング + リアクティブプログラミング

- 略して FRP (Functional Reactive Programming)
- RP が関数型プログラミングを取り入れたもの
- **Event** と **Behavior** という概念があったり (割愛)

FRP の GUI 的な代表格が Reactive Extensions

- 略して Rx ・ Microsoft .NET 方面の生まれ
- Haskell 方面にある元祖? FRP と C# の LINQ の子供
- JavaScript の主要な FRP ライブラリはこれの系統

Reactive Extensions シリーズ

- [ReactiveX/RxAndroid](#)
- [ReactiveCocoa/ReactiveCocoa](#)
- **[Reactive-Extensions/RxJS](#)**
- [neuecc/UniRx](#)
- いずれも Reactive Extensions インスパイア系

どのようなライブラリなのか？

- 全ての値や入力を非同期データストリームとして見なす
- 入力に `map` や `filter` などの高階関数で処理を適用する
- 非同期データストリームを中心に、その繋がりと処理を記述する

クライアントサイドの処理は 時間軸の変化に依存する

- ユーザー入力イベント
- ディスク I/O
- ネットワーク I/O
- タイマー処理

ref. [Functional Reactive Programming with Bacon.js](#)

非同期データストリーム

(すべてを時間軸に沿って要素が連続するリストにする)

| | Synchronous | Asynchronous |
|-------------|---|--|
| Multiple(n) | Array <code>[1, 2, 3].filter().map()</code> | Stream / Observable <code>Stream.filter().onValue()</code> |
| Single(1) | Object (without Array) <code>{}, 1, true, 'foo'</code> | Promise <code>Promise.then()</code> |

ref. [Functional Reactive Programming with RxJS](#)

Events as List



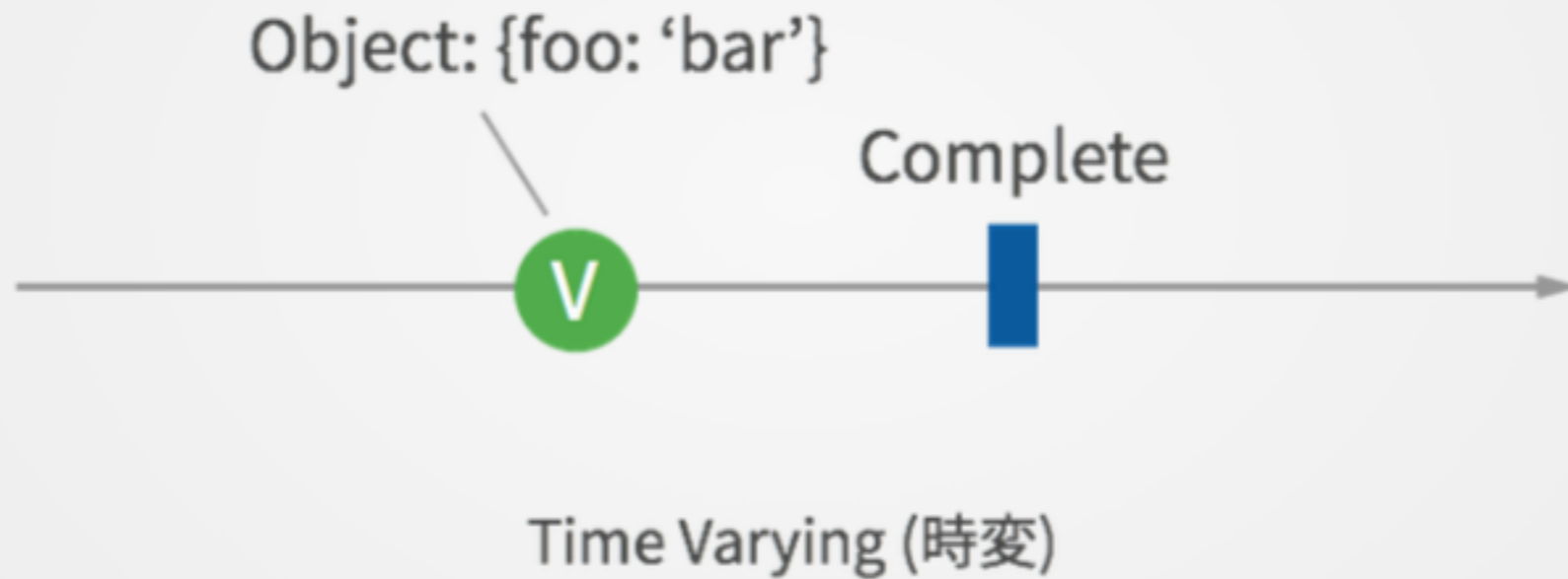
Time Varying (時変)

Error, Complete



Time Varying (時変)

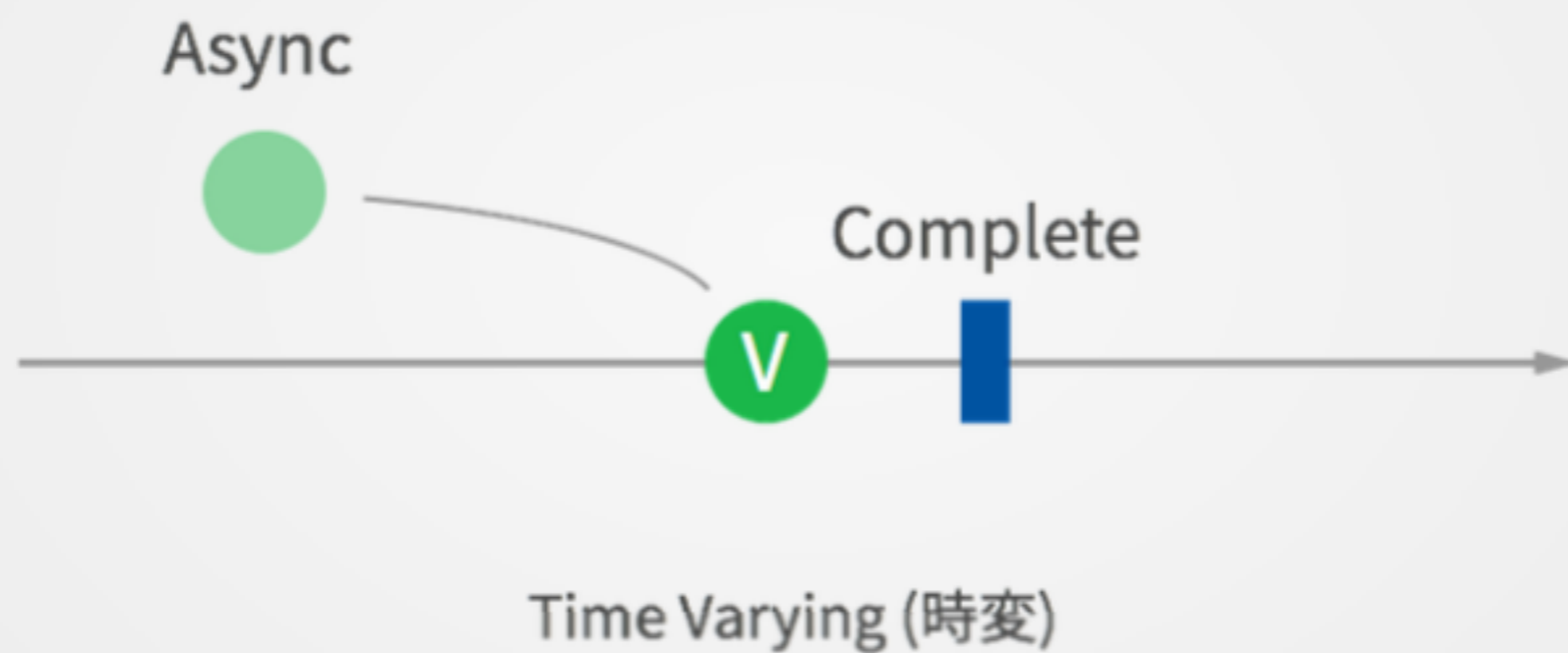
Single Value



From Array



Async Value



ストリームをリストと見なすことで
関数型のイディオムを
活かせるようになった

JavaScript は first-class function なので尚のこと

RxJS



Clean Composable Code

The Need to go Reactive

Reactive Programming is a hot topic as of late, especially with such things as the [ReactJS](#) applications, especially as the web has changed over the years from being a simple static page to complex, with animations, to the data visualization. Each time, we're adding more complexity, more data, and more interactivity to the page. How do we manage it all? How do we make it all moving smoothly? That's where [RxJS](#) comes in, which are event-driven, mutable and asynchronous. With the [Reactive Extensions](#), you have all the tools you need to help build these systems.

RxJS

Reactive Extensions 謹製 JavaScript 実装

The [Reactive Extensions for JavaScript \(RxJS\)](#) is a set of libraries for composing asynchronous and event-based programs using observable sequences and fluent query operators modeled after [Language Integrated Query \(LINQ\)](#). Using RxJS, developers represent asynchronous data streams with [IObservable](#), query asynchronous data streams using [IObservable](#) operators, and parameterize the components of the asynchronous data streams using [IScheduler](#). Simply put, RxJS = [IObservable](#) + [IObservable](#) + [IScheduler](#). Whether you are authoring a web-based application or [desktop](#) or a server-side application in Node.js, you have to deal with asynchronous and event-based programming as a matter of course. Although some patterns are emerging such as the [Promise](#) pattern, handling exceptions, cancellation, and synchronization is difficult and error-prone.

セッション冒頭の RxJS サンプルをおさらい

データストリーム (Observable) をどう扱っているか

```
/*
```

```
+-----+  
| click.map(1) |---->  
+-----+
```

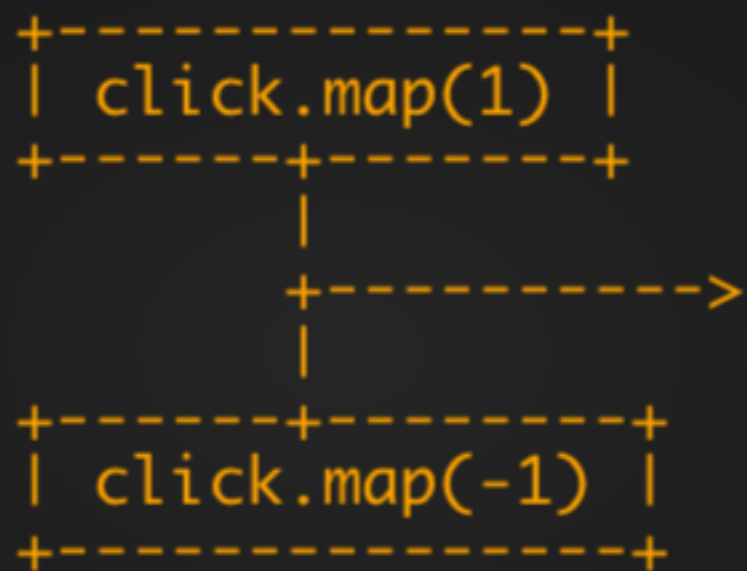
```
+-----+-----+  
| click.map(-1) |---->  
+-----+
```

```
*/
```

```
var plus = Rx.Observable  
    .fromEvent(e1, 'click').map(1);  
var minus = Rx.Observable  
    .fromEvent(e1, 'click').map(-1);
```



```
/*
```

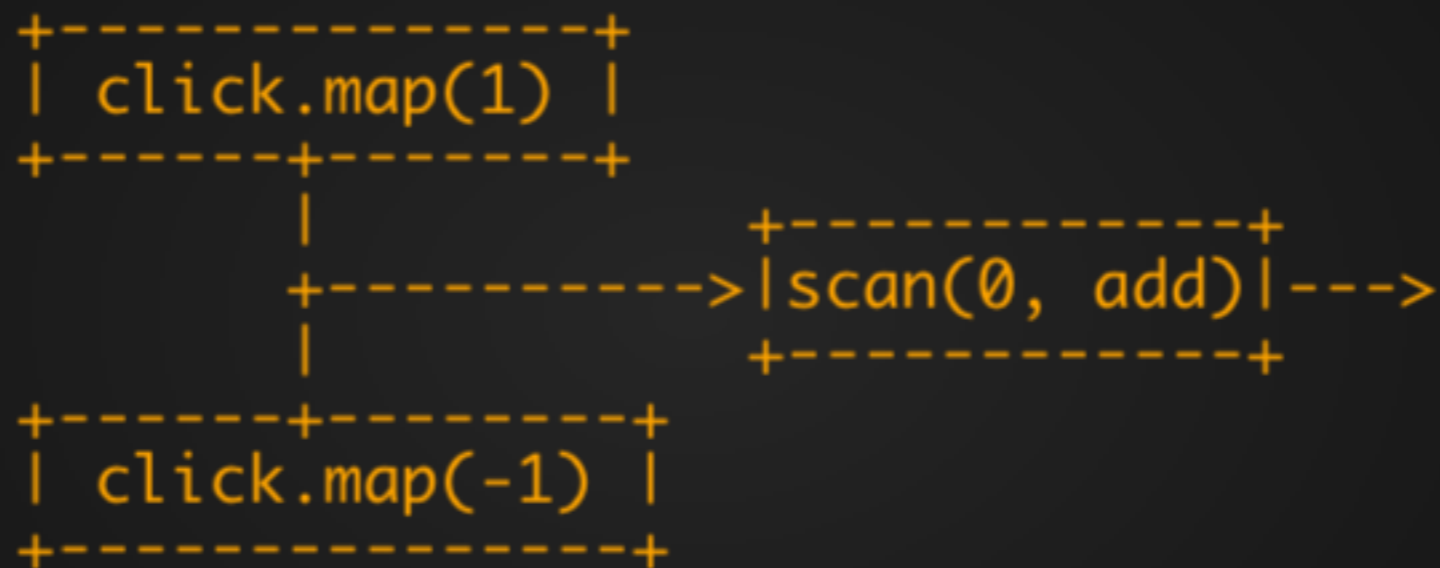


```
*/
```

```
var both = plus.merge(minus);
```

```
@type {Rx.Observable} curValue
```

```
/*
```



```
*/
```

```
var curValue = both.scan(0, function(acc, v) {
    return acc + v;
});
```

```
@type {Rx.Observer} htmlSet
```

```
/*
```

```
-----+  
---->|innerHTML = v|  
-----+
```

```
*/
```

```
var htmlSet = Rx.Observer.create(function(v) {  
    element.innerHTML = v;  
});
```

```

/*
+-----+
| click.map(1) |
+-----+
      |
      +----->|scan(0, add)|----->|innerHTML = v|
      |
+-----+-----+
| click.map(-1) |
+-----+
*/
var subscription = currentValue.subscribe(htmlSet);

```

Rx.Observable の インターフェースが中心になる

Observable = 非同期データストリーム

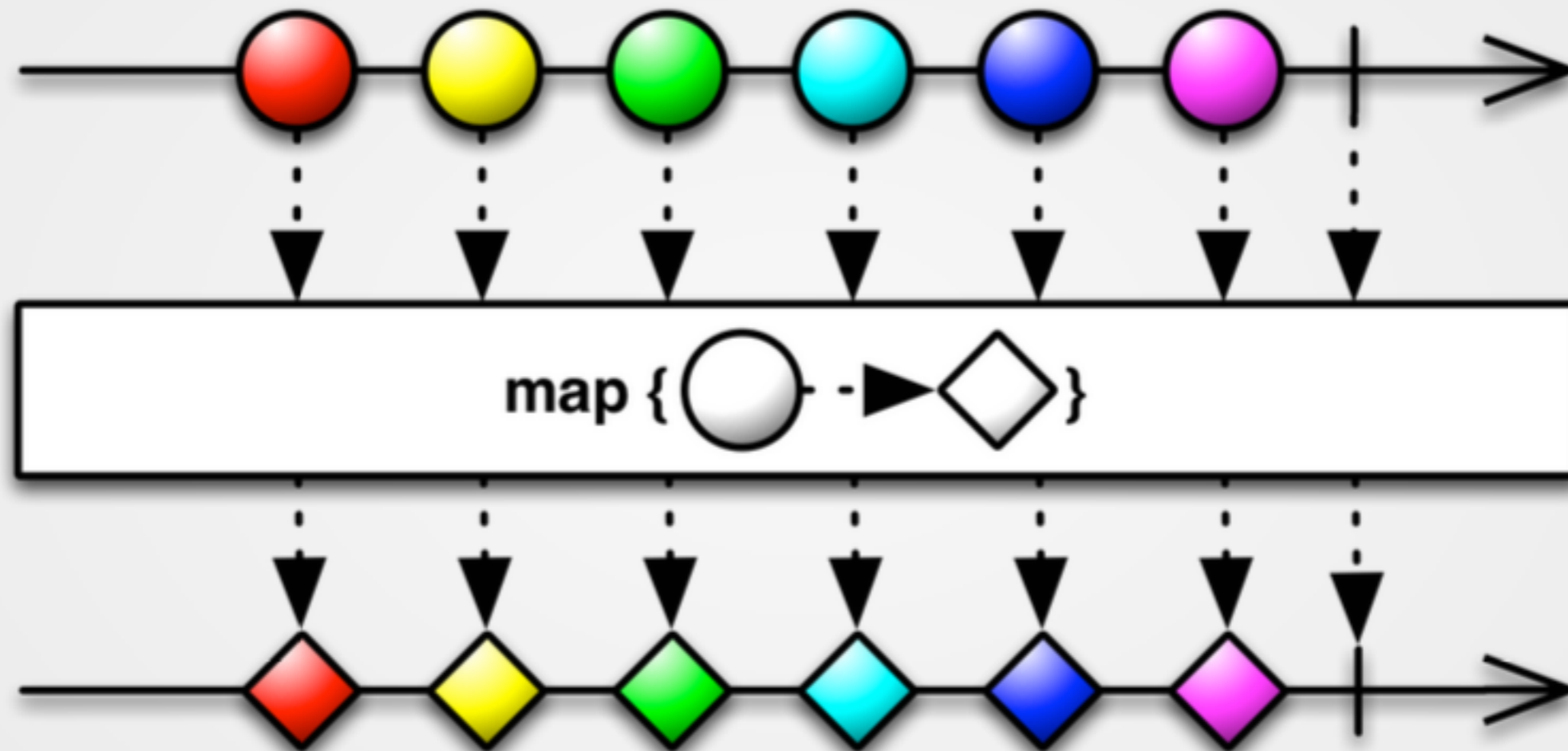
絶望の数

(Observable Instance Methods)

amb, and, asObservable, average, buffer, bufferWithCount, bufferWithTime, bufferWithTimeOrCount, catch | catchError, combineLatest, concat, concatAll, concatMap, concatMapObserver, connect, includes, controlled, count, debounce, debounceWithSelector, defaultIfEmpty, delay, delayWithSelector, dematerialize, distinct, distinctUntilChanged, do, doOnNext, doOnError, doOnCompleted, doWhile, elementAt, elementAtOrDefault, every, expand, filter, finally | ensure, find, findIndex, first, firstOrDefault, flatMap, flatMapObserver, flatMapLatest, forkJoin, groupBy, groupByUntil, groupJoin, ignoreElements, indexOf, isEmpty, join, jortSort, jortSortUntil, last, lastOrDefault, let | letBind, manySelect, map, max, maxBy, merge, mergeAll, min, minBy, multicast, observeOn, onErrorResumeNext, pairwise, partition, pausable, pausableBuffered, pluck, publish, publishLast, publishValue, share, shareReplay, shareValue, refCount, reduce, repeat, replay, retry, retryWhen, sample, scan, select, selectConcat, selectConcatObserver, selectMany, selectManyObserver, selectSwitch, sequenceEqual, single, singleOrDefault, skip, skipLast, skipLastWithTime, skipUntil, skipUntilWithTime, skipWhile, some, startWith, subscribe, subscribeOnNext, subscribeOnError, subscribeOnCompleted, subscribeOn, sum, switch | switchLatest, switchMap, take, takeLast, takeLastBuffer, takeLastBufferWithTime, takeLastWithTime, takeUntil, takeUntilWithTime, takeWhile, tap, tapOnNext, tapOnError, tapOnCompleted, throttleFirst, throttleWithTimeout, TimeInterval, timeout, timeoutWithSelector, timestamp, toArray, toMap, toSet, transduce, where, window, windowWithCount, windowWithTime, windowWithTimeOrCount, withLatestFrom, zip

via. [RxJS/observable.md at master · Reactive-Extensions/RxJS](#)

Transforming.Map



via. [ReactiveX - Map operator](#)

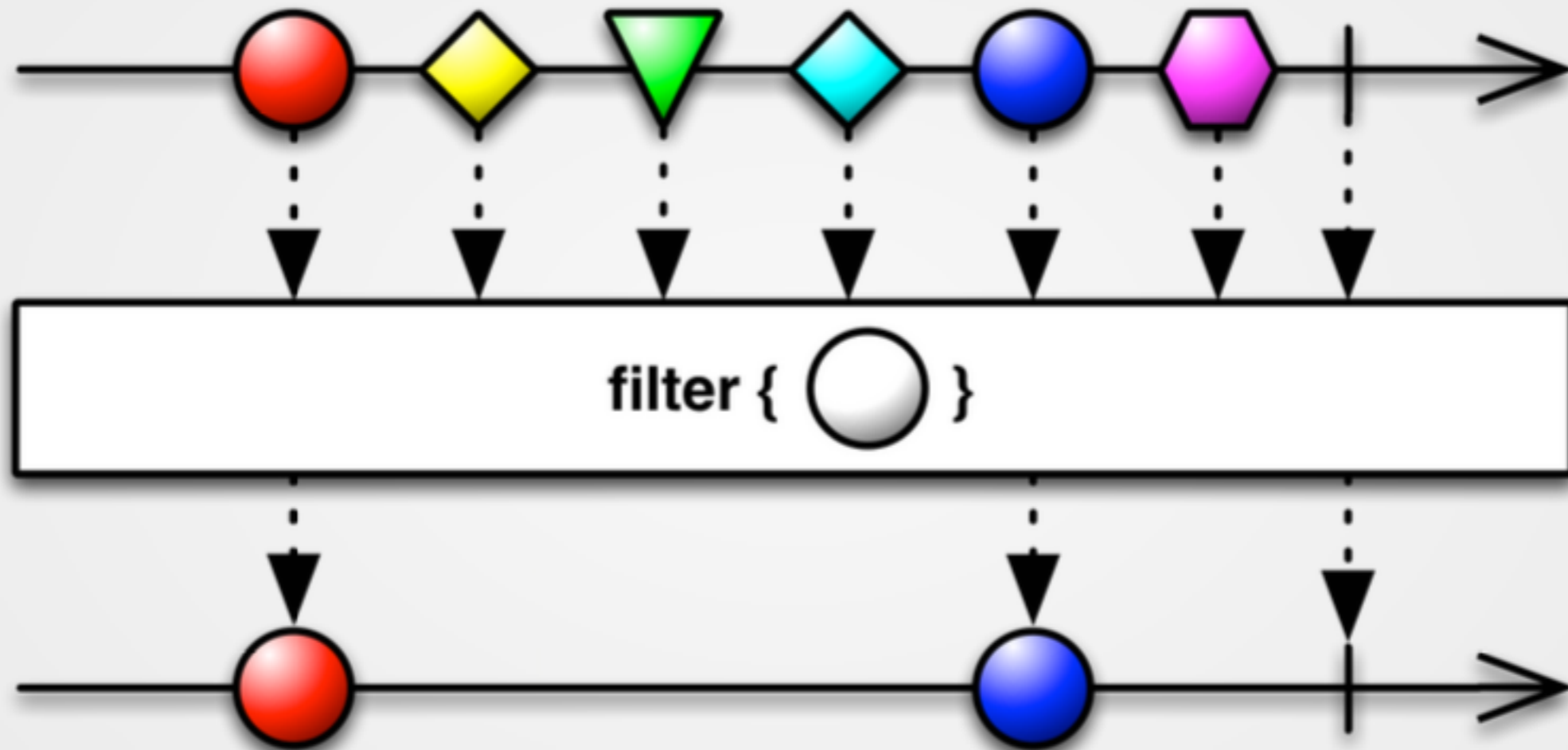
```
Rx.Observable.of(1, 2, 3)
  .map(function(x) {
    return x * x;
  })
  .subscribe(function(value) {
    console.log(value);
  });
```

```
// => 1
```

```
// => 4
```

```
// => 9
```


Filtering.Filter



via. [ReactiveX - Filter operator](#)

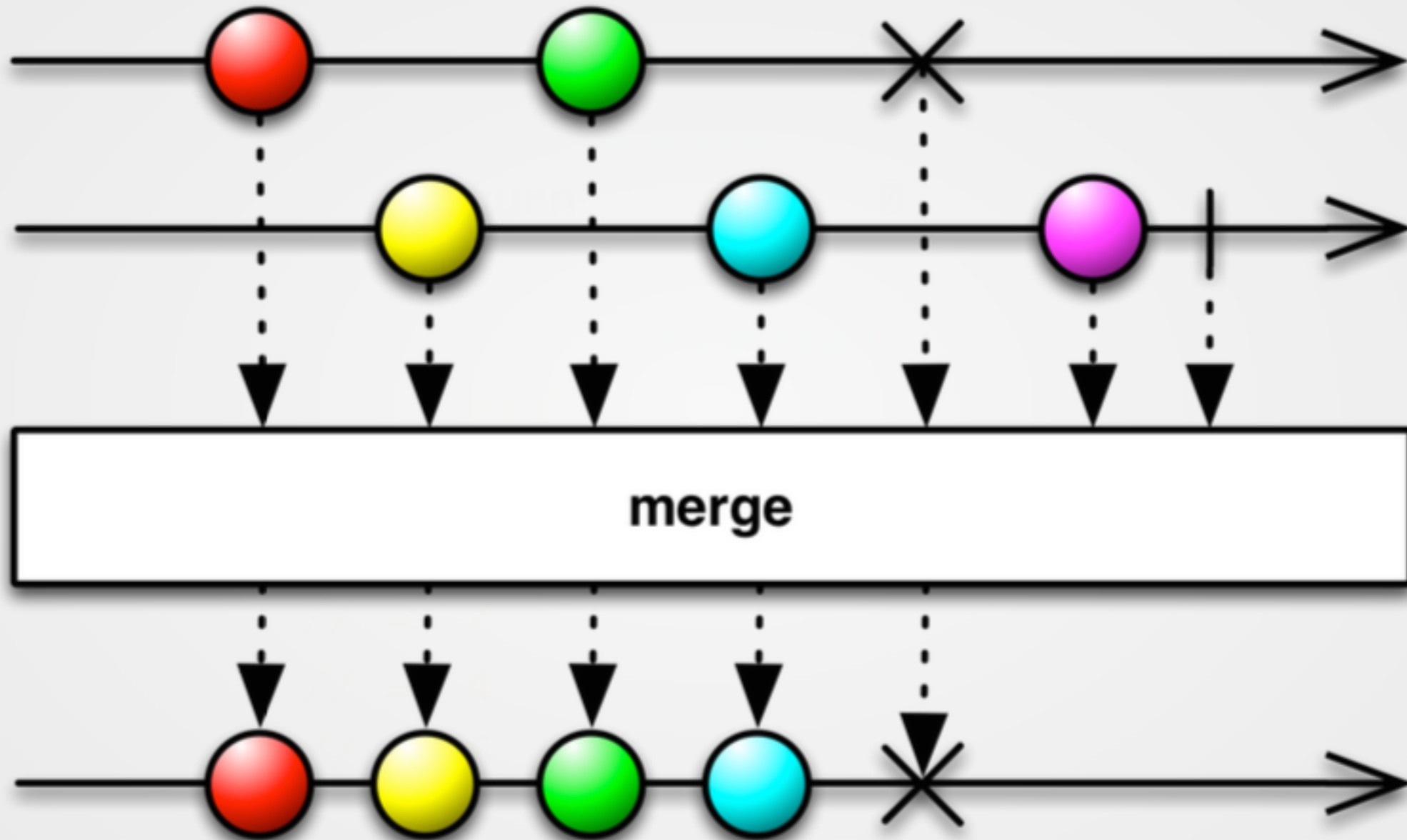
```
Rx.Observable.range(1, 7)
  .filter(function (x) {
    return x % 2 === 0;
  })
  .subscribe(function(value) {
    console.log(value);
  });
```

```
// => 2
```

```
// => 4
```

```
// => 6
```

Combining.Merge



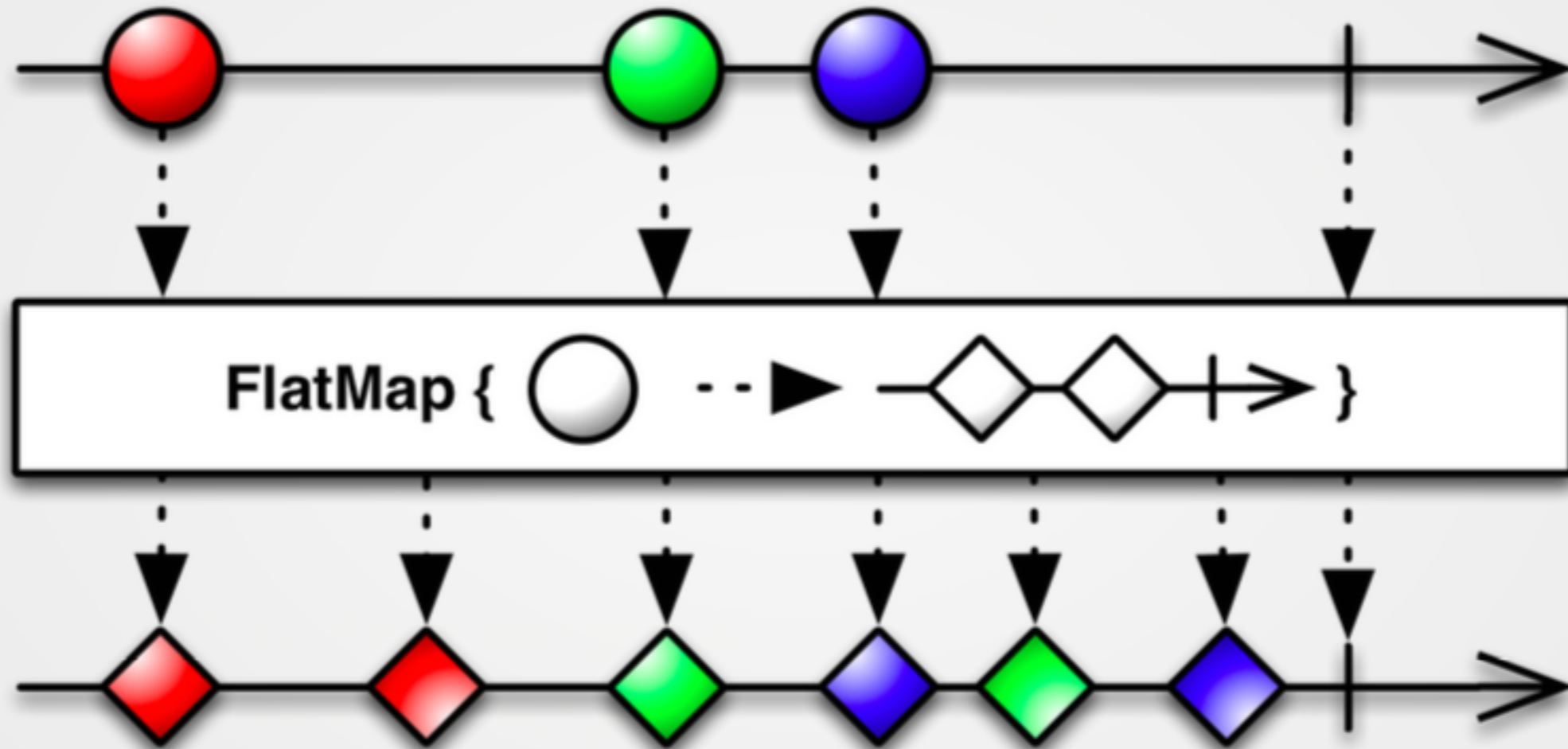
via. [ReactiveX - Merge operator](#)

```
var a = Rx.Observable
    .fromEvent(el, 'click').map('a');
var b = Rx.Observable
    .fromEvent(el, 'click').map('b');

a.merge(b).subscribe(function(value) {
    console.log('From merged: ' + value);
});

// => From merged a
// => From merged b
// => From merged b
```

Transforming.FlatMap



via. [ReactiveX - FlatMap operator](#)

```
Rx.Observable.of(1, 2, 3)
  .flatMap(function (x) {
    return Rx.Observable.of(x, x*x);
  })
  .subscribe(function(value) {
    console.log(value);
  });
```

```
// => 1
```

```
// => 1
```

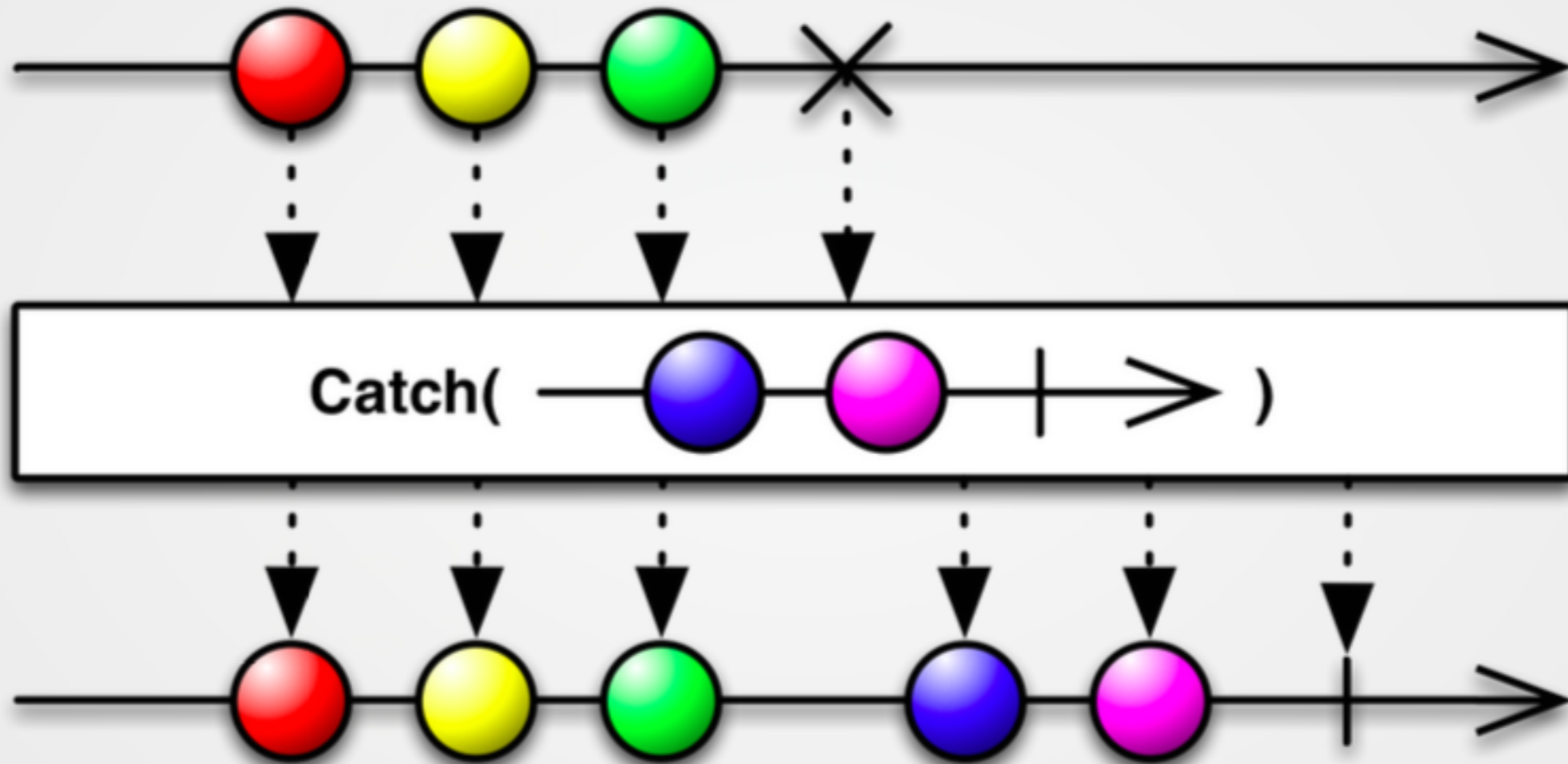
```
// => 2
```

```
// => 4
```

```
// => 3
```

```
// => 9
```

ErrorHandling.Catch



via. [ReactiveX - Catch operator](#)

```
Rx.Observable.of(1, 2, 3)
  .map(function (x) {
    if (x === 2) throw new Error();
    return x;
  })
  .catch(function (e) {
    return Rx.Observable
      .return(e instanceof Error);
  })
  .subscribe(function(value) {
    console.log(value);
  });

// => 1
// => true
```



```
Rx.Observable.of(1, 2, 3)
  .map(function (x) {
    if (x === 2) throw new Error();
    return x;
  })
  .onErrorResumeNext(Rx.Observable.of(4, 5, 6))
  .subscribe(function(value) {
    console.log(value);
  });
```

```
// => 1
```

```
// => 4
```

```
// => 5
```

```
// => 6
```

フローコントロールの用途で Async.js と読み替える例

- [RxJS/comparing.md at master · Reactive-Extensions/RxJS](#)
- これを参考にすると、フローコントロール的には分かりやすい

各 FRP ライブラリの概観

- [RxJS](#) は ReactiveExtension 由来の正統派
- [Bacon.js](#) は RxJS を参考にした独自コンセプト
- [Kefir.js](#) は Bacon.js に似たAPIの軽量化版
- [Meteor](#) は Tracker.js を中心にサバクラで共有できる Reactive
- [Elm](#) は Haskell っぽいことから HTML/CSS/JS を生成する

Bacon.js

''

*One of the main motivators for reactive-bacon has been the **weirdness of RX with regard to "hot" and "cold" observables**. In reactive-bacon, there are no such things. The EventStream is always consistent with respect to time, so there will be no WTFs from that direction.*

via. [raimohanska/reactive-bacon](https://github.com/raimohanska/reactive-bacon)

よくわからない人もいると思うので
デモを見てみましょう!



Demo 1 (RxJS)

Canvas にドラッグ操作で線を描画する

RxJS でリライトしてみる

Draw Canvas (VanillaJS)



Draw Canvas (RxJS)

Demo 2 (React + Bacon.js)

ReactとBacon.jsを組み合わせて簡単なUIを作ってみる

Sample

0 / 140Submit

投稿したらリストに反映 0秒前

Shift+Enterでも投稿 3時間前

文字数制限 6時間前

文字数カウント 9時間前

スライドを送れなくなったひとは、**ココ**を click してから操作してね

ahomu/demo-react-bacon

<https://github.com/ahomu/demo-react-bacon>

1. **Is San Francisco losing its soul? (2014)** (www.theguardian.com)
15 points by edward 41 minutes ago | 4 comments
2. **Magic** (www.getmagicnow.com)
1001 points by wittyphrasehere 15 hours ago | 110 comments
3. **Money Doesn't Buy Happiness, but Time Just Might Do It** (nautil.us)
28 points by dnetesn 1 hours ago | 5 comments
4. **The Quantum Mechanics of Fate** (nautil.us)
14 points by jeremynixon 1 hours ago | 2 comments
5. **The Dangers of the "Google Analytics-Powered Startup"** (simontorring.com)
29 points by redredredred 2 hours ago | 5 comments
6. **How an American sergeant's journey through Russia inspired historical fiction**
(theamericanscholar.org)
8 points by benbreen 58 minutes ago | 0 comments
7. **5 Lesser Used HTML Elements** (mattsparks.com)
4 points by kazak 14 minutes ago | 1 comments
8. **Internet of Crappy Things** (blog.kaspersky.com)
149 points by kalleboo 10 hours ago | 15 comments
9. **The Common Cure for Heroin Addiction Is Also a Magnet for Police Harassment**
(www.buzzfeed.com)
41 points by Petiver 1 hours ago | 7 comments
10. **Tell HN: Groff needs your help** ()
30 points by cogburnd02 8 hours ago | 6 comments
11. **Segment is hiring engineers to revamp our infrastructure** (segment.com)
1 points by calvinfn 43 minutes ago | 0 comments

スライドを送れなくなったひとは、**ココ**を click してから操作してね

ahomu/hn-react-rxjs

<https://github.com/ahomu/hn-react-rxjs>

実際に書いてみた感想

- Observable と Observer ごとにモジュール化するカンジ？
- React + Flux の Store 周りにはいらぬ気がする
- 割り切って UI コントロール用途で使うほうが幸せかも
- Bacon.js 系のほうが昨今のフロントエンドには混ぜやすそう
- ストリームの入出力をドキュメントで明示しないと大変
- 高階関数の命名規則を考えたほうが可読性上がりそう

A green budgie is perched on a wooden branch on the left side of the image. On the right side, a yellow budgie is sitting on a nest made of crumpled white paper. The background is a light-colored wooden surface. The word "Conclusion" is written in white serif font in the center of the image.

Conclusion

まとめ

- RP はデータフローの関係性を宣言する
- FRP は非同期データストリームを扱うモデル
- RxJS は Observable を制すれば勝てる
(自分は Bacon.js の EventStream/Property が好き)
- クセはあるけど触れてみると発見があって面白い

FRP を知るオススメ文書

- [Reactive Porn](#) (最初に読むのを本当におすすすめしたい)
- [なぜリアクティブプログラミングは重要か。](#)
- [【翻訳】あなたが求めていたリアクティブプログラミング入門](#)
- [Functional Reactive Programming](#) (よくまとまっている)

Bacon.js を知るオススメ文書

- [Functional Reactive Programming and Bacon.js](#)
- [Functional Reactive Programming with Bacon.js](#)
- [Bacon.js examples on CodePen](#)
- [FAQ · baconjs/bacon.js Wiki](#)
- [Bacon.js blog](#)

BACON.JS & TALKIE.JS

@ahomu / Ayumu Sato



share

Talkie.js

[ahomu/Talkie - github.com](https://github.com/ahomu/Talkie)

Bacon.js で書かれたHTMLスライドライブラリ

A ginger cat with green eyes is looking upwards against a cosmic background of stars and a colorful nebula. The text 'THANK YOU' is overlaid in white, bold, sans-serif font across the middle of the image.

THANK YOU

aho.mu

@ahomu