Introdução

- Biblioteca para programação reativa
- Utilização de Observables
- Facilita a composição de operações baseadas em *callbacks* ou assincronas
- Trata eventos como coleções de dados
- Manipula conjuntos de dados com operadores
- Composição de streams de dados

Observables

- *Streams* - Coleções de dados.
- De qualquer número de dados.
- Sobre qualquer período de tempo (imediato/curto/infinito).
- Lazy - Preguiçosos, só geram valores após terem alguém inscrito(*subscribed*) neles.
- Podem ter a inscrição cancelada (*unsubscribed*).
- Tem lógica de *teardown* para liberar os recursos após o uso.
- Emitem três tipos de notificações: próximo/erro/completado (*next/error/complete)*)
- Geralmente são representados por variáveis com $ no final
- Podem ser cold ou hot

Producers

- Fonte de dados dos valores emitidos pelo *Observable*
- Geralmente os *producers* são criados quando o *Observable* recebe uma nova *subscription* (*cold*)
- Se o *producer* for criado fora do processo de *subscription*, então quer dizer que esse producer é compartilhado por todos os *consumers (Observable hot)* daquele observable.
- Dizemos que um Observable é *unicast* quando ele mantêm a relação de 1 *producer* para 1 *consumer.*
- Dizemos que um Observable é *multicast* quando ele mantêm a relação de 1 *producer* para N *consumers.*
- Devido a essa relação, podemos dizer que os Observables *hot* são sempre multicast.

Producers comuns em aplicações web

- Eventos do *DOM* (clicks, keydown, etc) - 0 a N valores
- Animações - Canceláveis
- *AJAX* (requisições http) - 1 valor
- *WebSockets* - 0 a N valores
- Inputs alternativos (voz, gamepad, etc) - 0 a N valores

Observer

- Consumidores de valores emitidos pelos Observables
- Possuem callbacks para cada um dos tipos de eventos emitidos pelos Observables (next / error / complete)
- Podem ser implementados de forma parcial, listando somente um ou dois callbacks

Subjects

- É um tipo especial de Observable que permite emitir valores para multiplos consumidores (multicast)
- São Observables que também são Observers (next/error/complete), que servem para emitir os eventos respectivos.
- Subjects são sempre *multicast*
- RxJS fornece 4 tipos de subjects
  - Subject - Geralmente representam eventos ao longo do tempo
  - BehaviorSubject - Usado para armazenar valores ao longo do tempo, possui o conceito de "valor atual" e sempre emite o valor atual quando recebe uma nova subscription.
  - ReplaySubject - Armazena N valores que são emitidos quando recebem uma nova subscription.
  - AsyncSubject - Emite o último valor somente após ser completado.

Operadores

Operadores

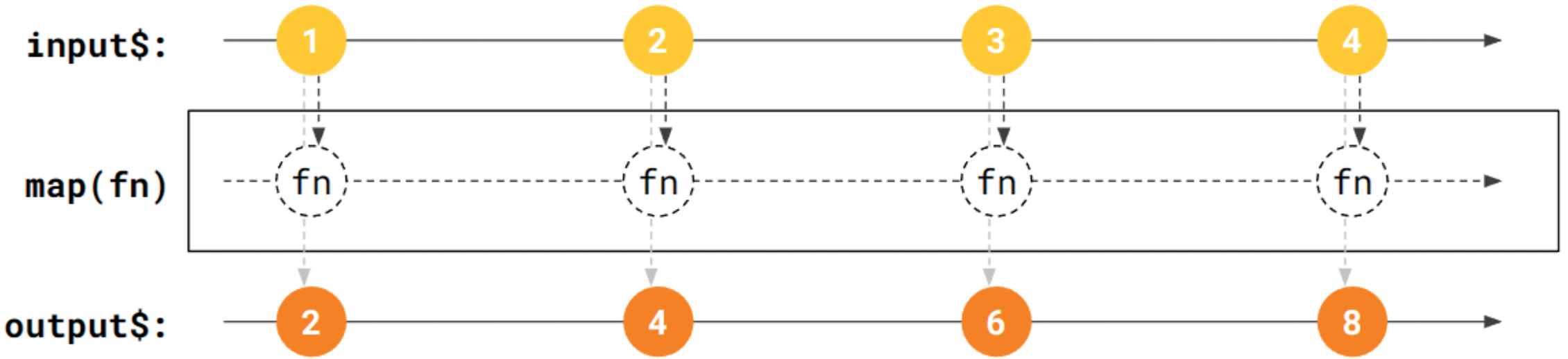
- São funções que geram Observables ou transformam um ou mais observables em outro observable.
- Podem ser estáticos ou *pipeable*
- Operadores *pipeados* são usados dentro da função .pipe() para encadear os operadores em sequência.
- RxJS possui muitos operadores (50+) e é possível criar novos operadores customizados.

Conjuntos de dados possuem operações...

- Busca (*filtering*)
- Transformar (*mapping)*
- Acumular (*reducing)*
- Combinar (*joining)*
- Achatar (flattening)

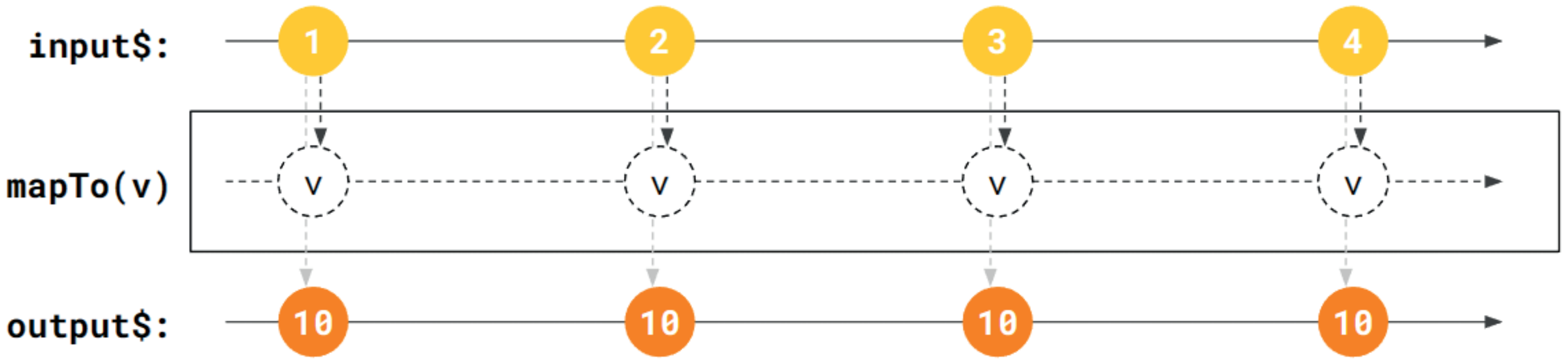
map

```
fn:  x => x * 2
```



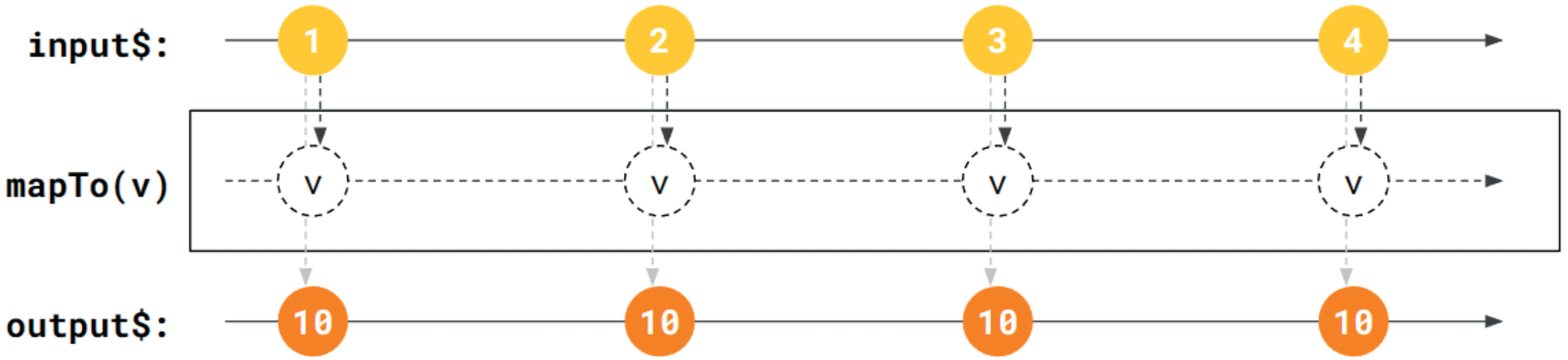
mapTo

```
v:  10
```



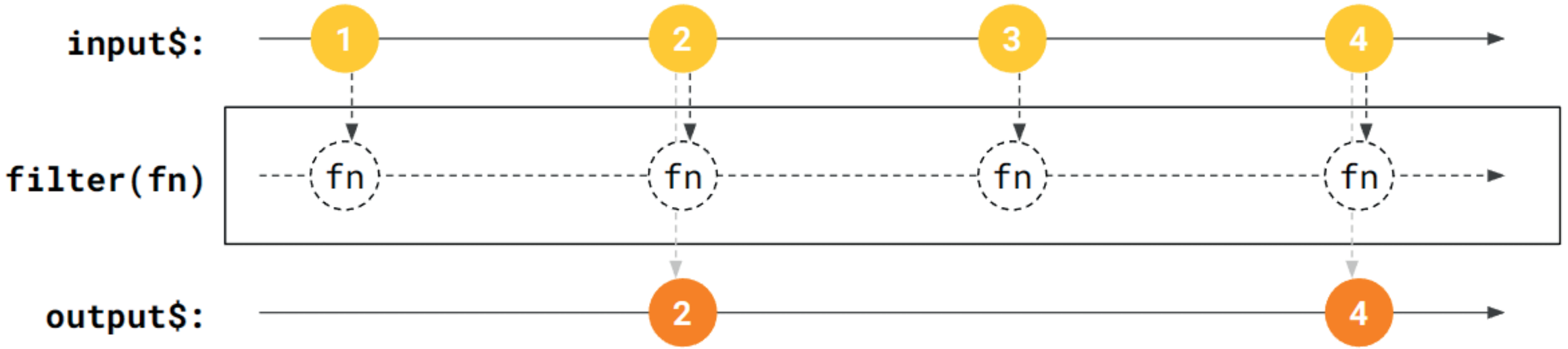
mapTo

```
v:  10
```



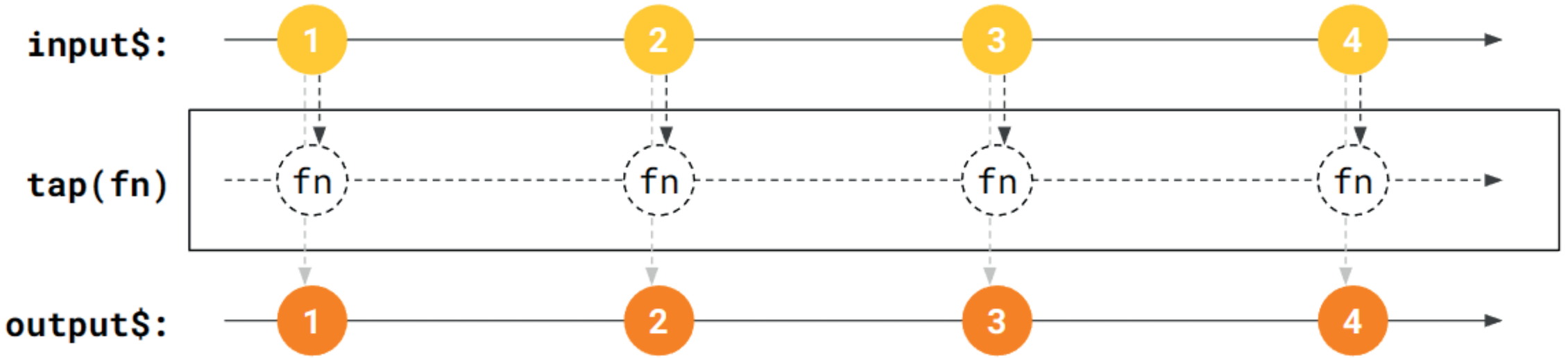
filter

```
fn:  x => x % 2 === 0
```



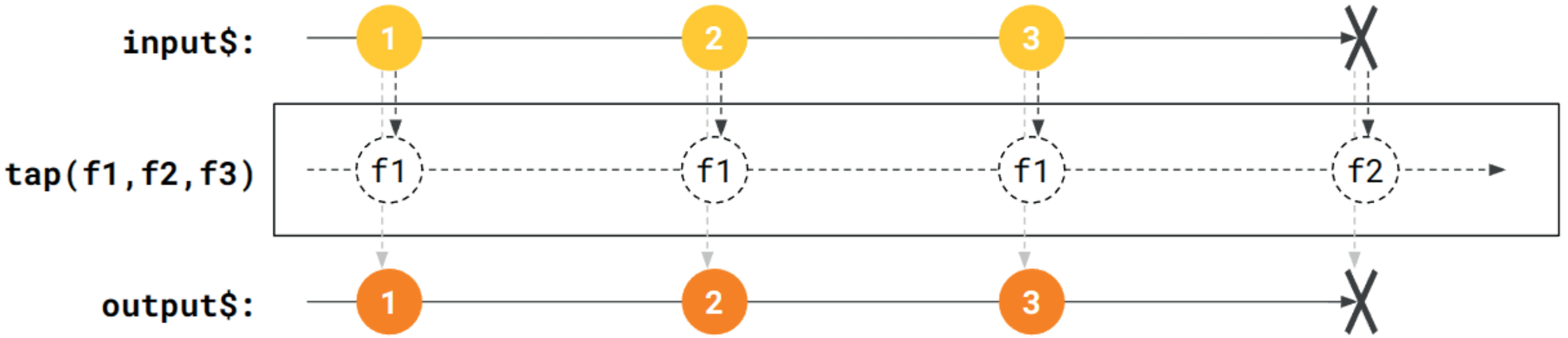
tap

```
fn:  n => console.log(`next: ${n}`)
```



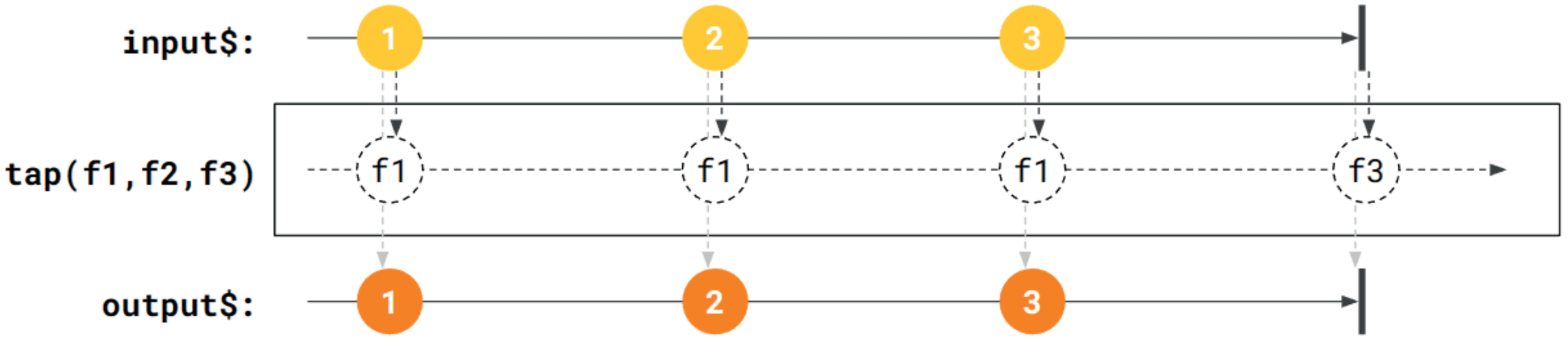
tap

```
f1:  n => console.log(`next: ${n}`)

f2:  e => console.log(`error: ${e}`)    f3:  () => console.log(`complete`)
```



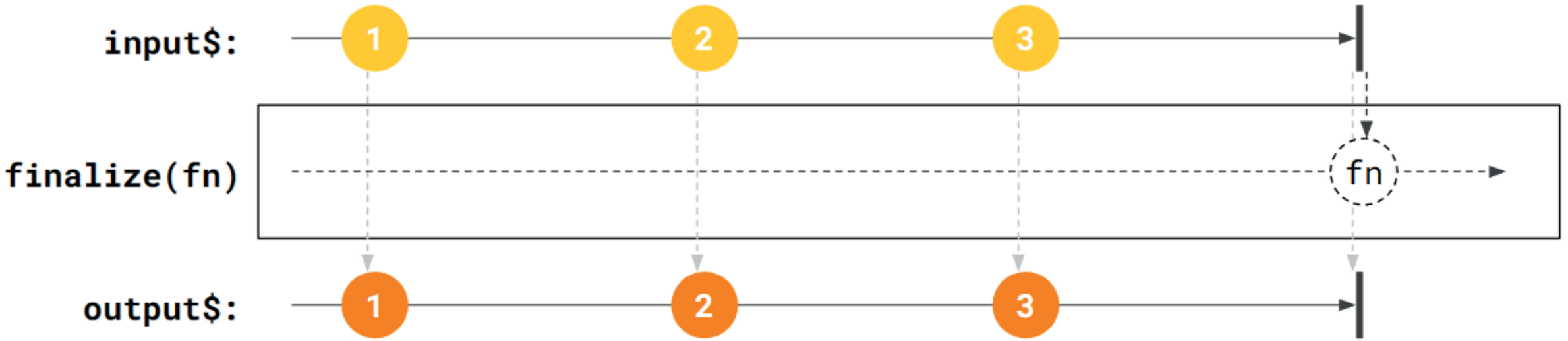
tap

```
f1:  n => console.log(`next: ${n}`)

f2:  e => console.log(`error: ${e}`)    f3:  () => console.log(`complete`)
```



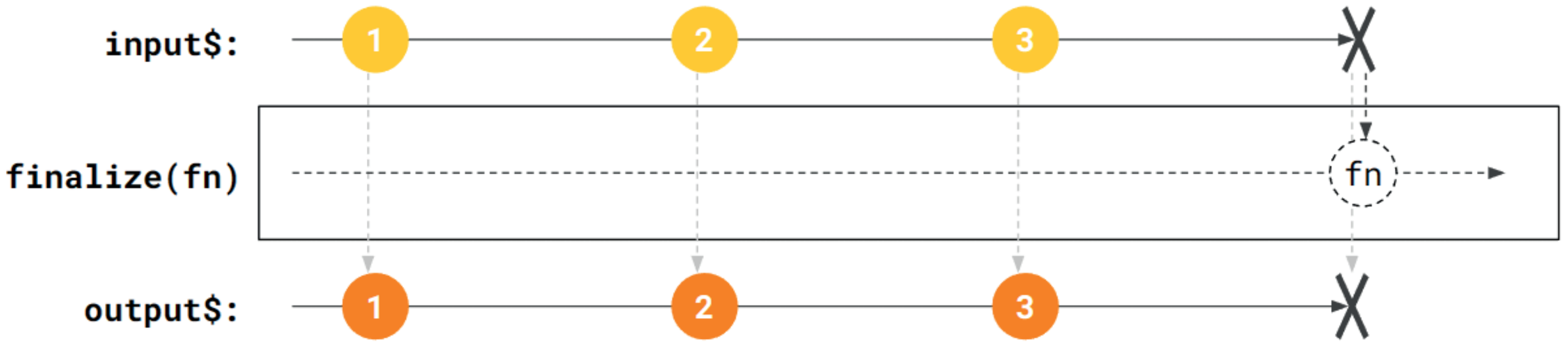
finalize

```
fn:  n => console.log(`erro ou complete: ${n}`)
```

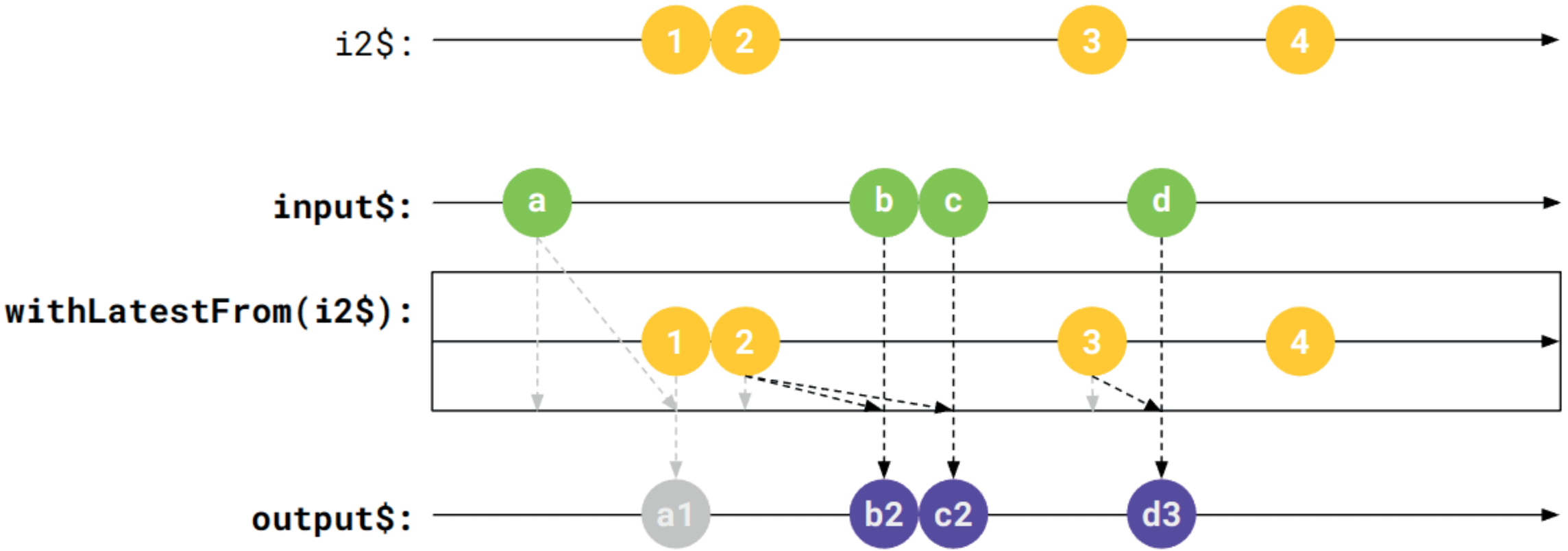


finalize

```
fn:  n => console.log(`erro ou complete: ${n}`)
```

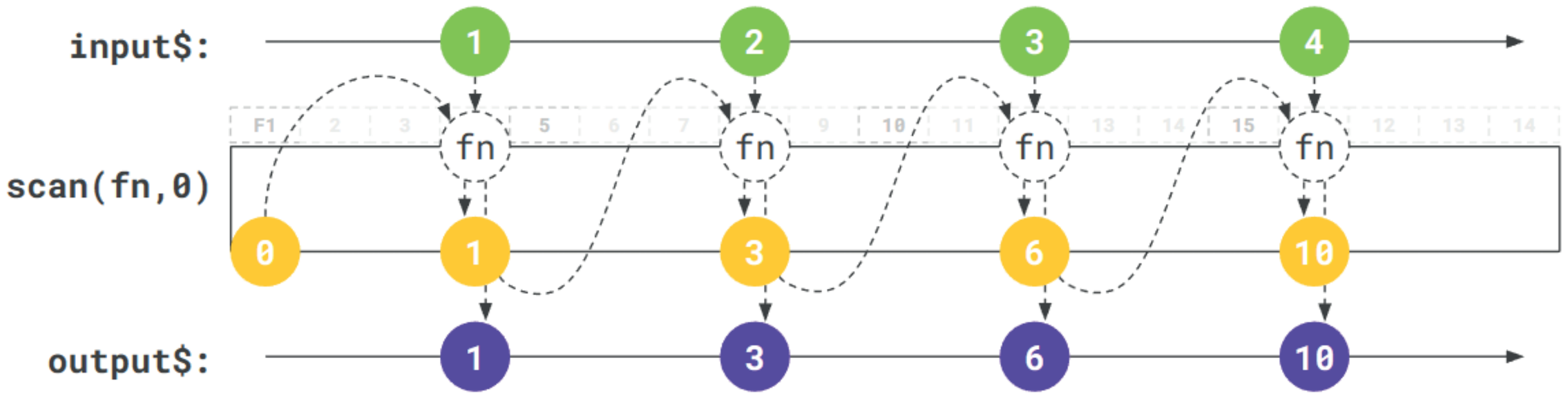


withLatestFrom



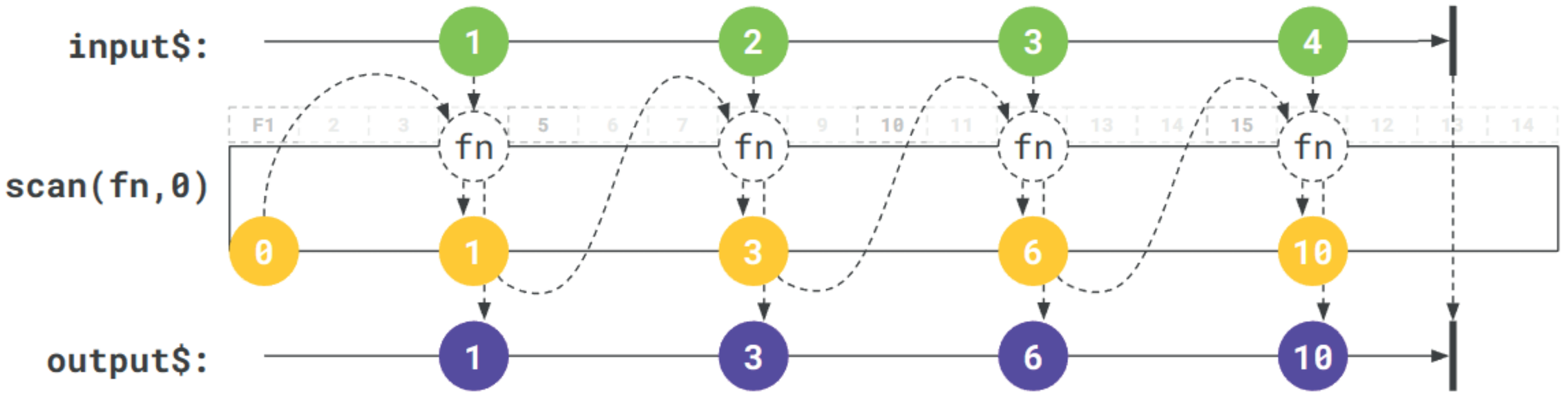
scan

```
fn:  (acc,i) => acc+i
```



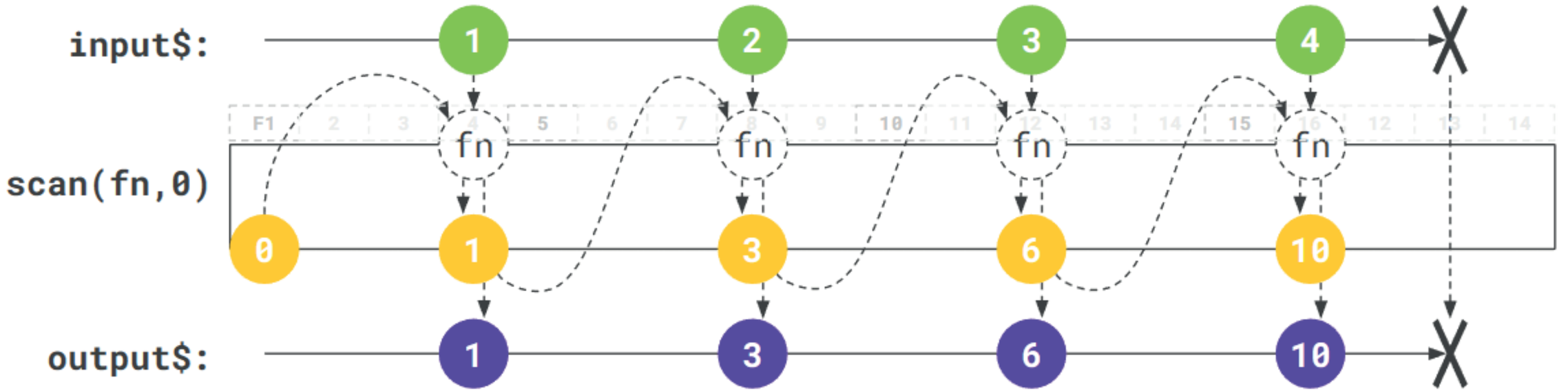
scan

```
fn:  (acc,i) => acc+i
```

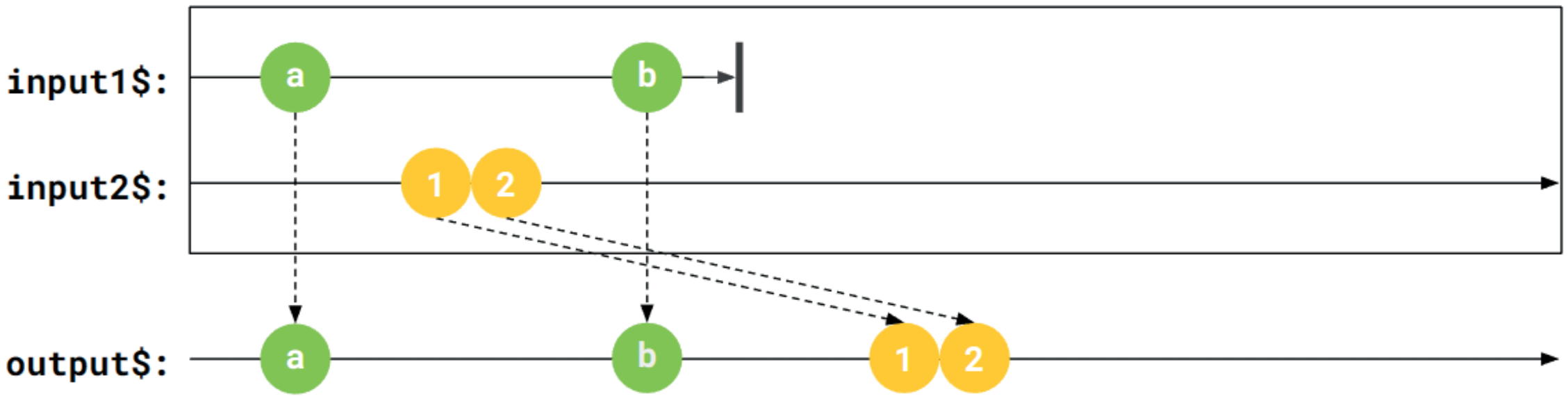


scan

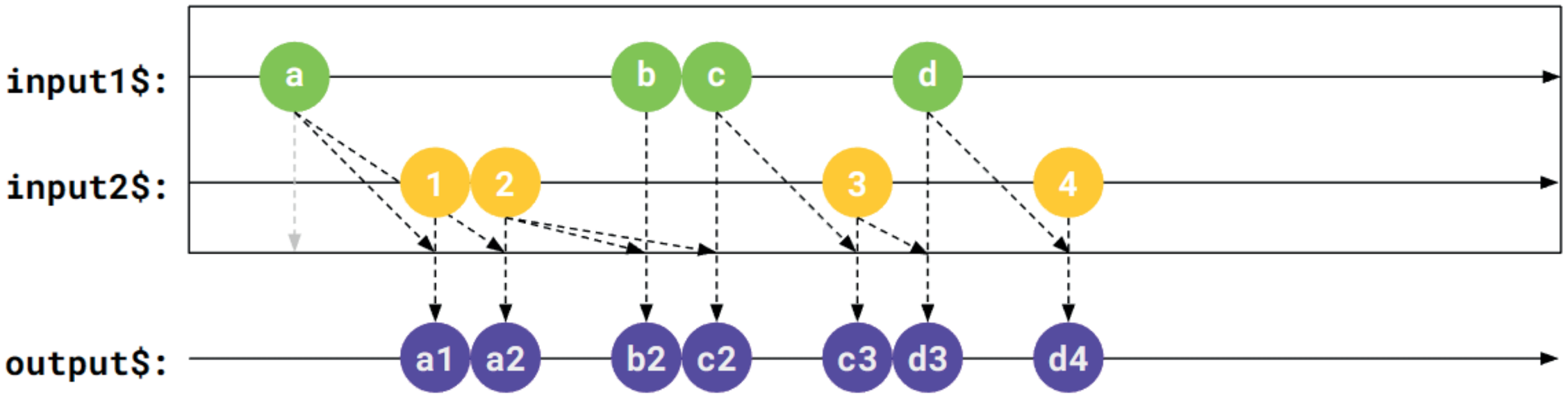
```
fn:  (acc,i) => acc+i
```



concat

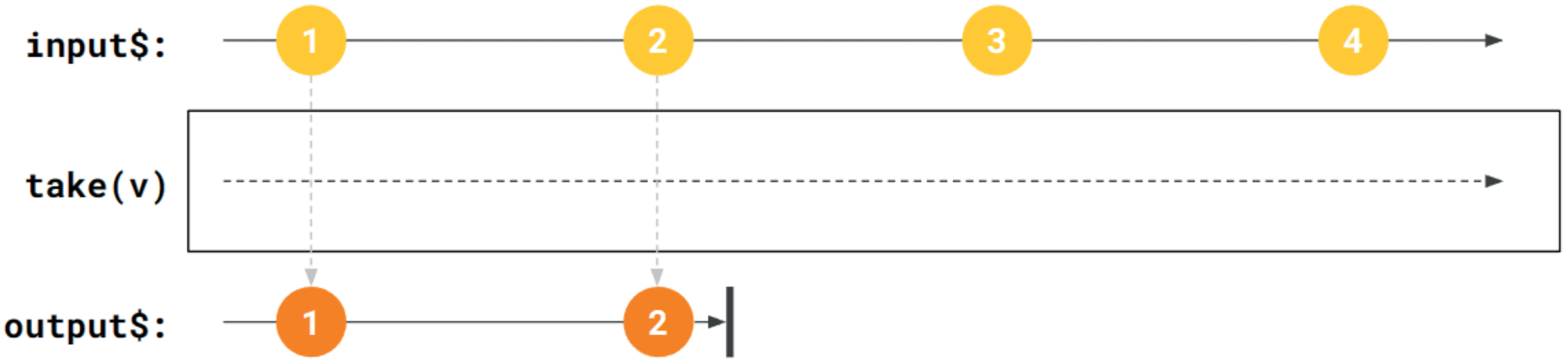


combineLatest



take

```
v:  2
```



Higher-order Observables

- *Observables* que emitem outros Observables
- Muito comuns quando encadeamos operações assíncronas
- Exemplos:
  - Observables de clicks que disparam requisições http
  - Encadeamento de múltiplas requisições http
  - Preenchimento de input disparando busca de dados assincronos