const =

“👋 Bye bye foreach - map() filter() reduce()”

leonardo elias

{ front-end developer, ...rest }

leonardoelias commented 8 days ago • edited

Leonardo Elias dos Santos
*@leonardoelias_*

Bye bye foreach - map() filter() reduce()

Você NÃO pode continuar programando sem aprender o quão poderoso são os métodos map, filter e reduce. Além de ser uma introdução ao paradigma funcional.

ideia && propósito

(e)

evinorecruiter commented on 11 Dec 2017 • edited

Descrição da vaga

Como Front End Dev & Tech Lead, seus maiores desafios serão:

- Planejar a arquitetura das nossas aplicações para que sigam as melhores práticas do mercado;
- Implementar técnicas de desenvolvimento visando a evolução constante das nossas aplicações e do time;
- Acompanhar o dia a dia do time de front-end garantindo entregas de qualidade;
- Codificar, integrar e manter nosso site Front-end (NodeJS, React);
- Entregar projetos bem testados, sustentáveis e de alta complexidade;
- Trabalhar com a equipe de Produto para entender os requisitos, antecipando as mudanças para proporcionar uma experiência incrível ao usuário;
- Desenvolver software usando programação funcional (immutability, side effects, high order functions, referential transparency, map/reduce).

Local

Alocado no escritório na Bela Cintra, próximo à Paulista

Benefícios

- VT (com desconto de 6%) ou estacionamento no prédio (com desconto de R$43,00/mês)
- VR ou VA (R$28,00/dia)
- Vale Alimand – para consumo interno (R$65,00/mês)

problema

paradigma

paradigma funcional

o paradigma funcional não e nenhuma novidade

~1950

porque agora ?

memória

x

processamento

funcional gasta mais memória

memória

a mémoria não era tão barata ou acessível quanto e hoje

memoria

1957 - $400MI/1megabyte

1980 - $50mil/1megabyte

mémoria

1957 - $400MI/1megabyte

1957 - $50mil/1megabyte

....


SanDisk

2017 - $0,0046

processamento

antes, quanto mas, GHz mais rápido seria o software seria.

processamento

antes, quanto mas, GHz mais rápido seria o software seria.

Ano	Modelo	
1978	VAX-11/780	
1991	HP PA-RISC	0.05 GHz
1993	PowerPC 604	0.1 GHz
1996	Alpha 21164	0.5 GHz
1999	AMD Althon	1.0 GHz
2000	Intel Pentium 4	2.0 GHz
2006	Ontem Pentium D	3.2 GHz
2017	AMD Ryzen	4.1 GHz

processamento

GHz parou de crescer, mas agora temos múltiplos cores

Ano	Modelo	Cores
2001	IBM Power 4	2
2006	Core 2	2
2008	Cores I7	6
2011	Alpha 21164	10
2017	Ryzen	32

mudou, mas e aí ?

“The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software”

Herb Sutter

gráfico titular

Os computadores não são

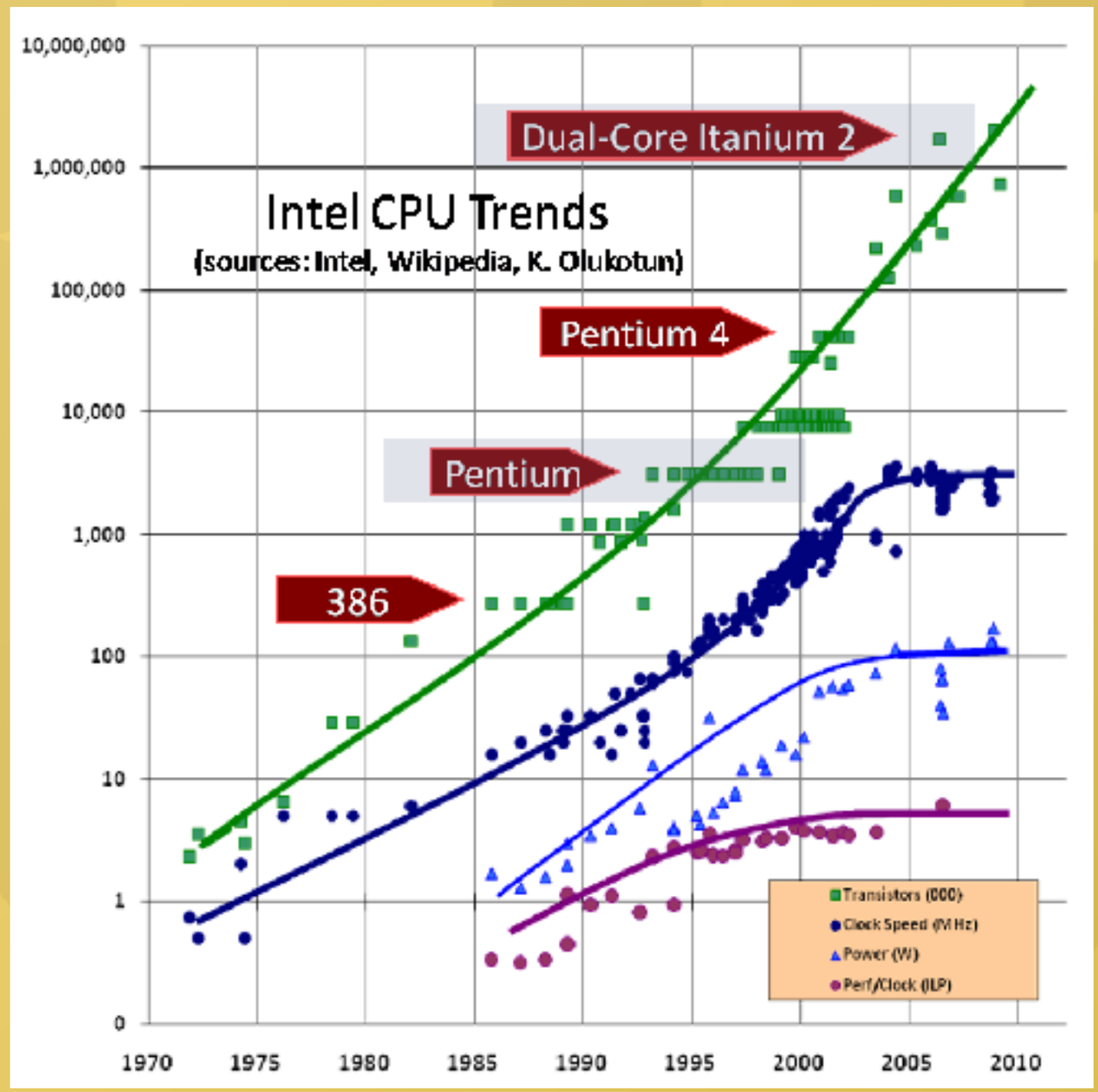
mais os mesmos, os softwares

não são mais os mesmos.

É preciso tirar proveito dos

múltiplos cores dos processadores

para que o software seja eficiente.



This article appeared in Dr. Dobb's Journal, 30(3), March 2005.

JS

Java

Swift

elixir

Scala

Erlang

elixir

OCaml

Clojure

elm

Perl 6

onde está javascript ?

fisrt-class e higher order function

- funções são tratadas como valores
- é possível passar funções como **argumentos**
- é possível receber funções como **resultados**
- reaproveitamento total de lógicas

fisrt-class e higher order function

- maior poder de abstração
- menor complexidade
- mais expressividade
- funções podem representar qualquer coisa, inclusive valores

```
const add = (a,b) => a + b

const makeAdd = (a) => b => add(a,b)
```

```
axios.get(...).then(response => {...})

setTimeout(() => {...}, 250)

[1,2,3].map(x => x * x)
```

pure functions

- não possui efeitos colaterais (side effect)
- não altera os contextos que têm acesso
- não altera os argumentos que recebe
- dada a sua entrada, tem que retornar a mesma saída

pure functions

- previsibilidade
- atomicidade
- testes simples e baratos
- sem surpresas

pure functions

- é mais intuitivo do que aparenta

- inúmeros bugs deixam de existir só em adotar esta filosofia

```
let counter = 0
const increments = () => conter++

increments() // counter = 1
increments() // counter = 2
```

```
let counter = 0
const increments = (value = 0) => value + 1

increments() // counter = 1
increments() // counter = 1
```

```
let counter = 0
const increments = (value = 0) => value + 1

counter = increments(counter) // 1
counter = increments(counter) // 2
```

```
const daysThisMonth = function () {
    var date = new Date();
    var year = date.getFullYear();
    var month = date.getMonth();
    var start = new Date(year, month, 1);
    var start = new Date(year, month + 1,1);

    return Math.round((end- start) / (1000 * 60 * 60 * 24))
}
```

```
const daysThisMonth = function (year, month) {
    var start = new Date(year, month, 1);
    var start = new Date(year, month + 1,1);

    return Math.round((end- start) / (1000 * 60 * 60 * 24))
}
```

```
// impura
saveUser(user) {
    http("POST", "/usuario". usuario)
}

// pura
saveUser(user) = {
    method: "POST",
    path: "/usuario",
    data: usuario
}
```

immutability

- anda lado a lado com funções puras
- variáveis não tem seus valores modificados
- sempre gera novos dados
- passa-se a pensar em estados e não em variáveis

immutability

- previsibilidade
- código claro e objetivo
- menos bug e maior precisão

immutability

- chave da computação paralela e concorrência
- é mais intuitivo do que se imagina

```
const addClientInSales = sales => {
    sales.forEach(sale => {
        sale.client = await getClient(sale.client_id)
    });
}
```

```
const addClientInSales = sales => {
    return sales.map(sale => {
        const client = await getClient(sale.client_id)
        return {...sale, client}
    })
}
```

map()

filter()

reduce()

```
const filmes = ["Alice", "Bruno", "Camila"];

for (let i = 0; i < filmes.length; i++) {
    console.log(i, filmes[i])
}
```

```
const filmes = ["Alice", "Bruno", "Camila"];

filmes.forEach(filme => console.log(filme))
```

map()

```
let videoWithIdAndTitle = [];

videos.forEach(video => videoWithIdAndTitle.push({id: video.id, title:
 video.title}))

console.log(videoWithIdAndTitle)
```

```
const videoWithIdAndTitle = videos.map(video => ({id: video.id, title:
 video.title}))

console.log(videoWithIdAndTitle)
```

```
const toPairsOfIdAndtitle = video => ({id: video.id, title: video.title})
const videoWithIdAndTitle = videos.map(video => (toPairsOfIdAndtitle))

console.log(videoWithIdAndTitle)
```

filter()

```
let topRatingVideos = [];

videos.forEach(video => {
    if (video.rating === 5) {
        topRatingVideos.push(video)
    }
})

console.log(topRatingVideos)
```

```
const topRatingVideos = video.filter(video => video.rating === 5)

console.log(topRatingVideos)
```

```
const topRatingVideos = videos
        .filter(m => m.rating === 5)
        .map(m => ({id: m.id, title: m.title, rating: m.rating}))

console.log(topRatingVideos)
```

```
const byRatingEqualsFive = obj => m.rating === 5
const toIdAndTitle = obj => ({id: m.id, title: m.title, rating: m.rating})

const topRatingVideos = videos
        .filter(byRatingEqualsFive)
        .map(toIdAndTitle)

console.log(topRatingVideos)
```

```
const propEquals = propName => value => obj =>
    obj[propName] === 5

const byRatingEqualsFive = obj => propEquals('rating')(5)
const toIdAndTitle = obj => ({id: m.id, title: m.title, rating: m.rating})

const topRatingVideos = videos
        .filter(byRatingEqualsFive)
        .map(toIdAndTitle)

console.log(topRatingVideos)
```

```
const propEquals = propName => value => obj =>
    obj[propName] === 5
const pick = props => obj => props.reduce((accu, curr) => {accu[curr] =
 obj[curr]; return accu}, [])

const byRatingEqualsFive = obj => propEquals('rating')(5)
const toIdAndTitle = obj => pick(['id', 'title'])(obj)

const topRatingVideos = videos
        .filter(byRatingEqualsFive)
        .map(toIdAndTitle)

console.log(topRatingVideos)
```

reduce()

```
cont arr = [0, 1, 2, 3, 4];

arr.reduce(function(valorAnterior, valorAtual) {
  return valorAnterior + valorAtual;
});

// 10
```

passo-a-passo

CODE

OUTPUT

```
1  const arr = [1, 2, 3, 4];
2
3  arr.reduce((total, amount) =>
4    total + amount
5  , 1);
```

VARIABLES

outros parâmetros

```
const dinheiros = [29.76, 41.85, 46.5];

const avg = dinheiros.reduce((total, amount, index, array) =>
{
  total += amount
  if ( index === array.length - 1 ) {
    return total / array.length
  } else {
    return total
  }
})

avg // 39.37
```

```
const dinheiros = [29.76, 41.85, 46.5];

const avg = ( list ) => list.reduce( ( acc, cur ) => acc + cur, 0 ) /
 list.length

avg(dinheiros) // 39.37
```

usando como um filter

```
const dinheiros = [29.76, 41.85, 46.5];
const above30 = dinheiros.reduce((total, amount) => {
  if (amount > 30) {
    total.push(amount);
  }
  return total;
}, []);

above30 // 41.85, 46.5
```

usando como um map

```
const dinheiros = [29.76, 41.85, 46.5];
const doubled = euros.reduce((total, amount) => {
  total.push(amount * 2);
  return total;
}, []);

doubled // [59.52, 83.7, 93]
```

reduzir

```
const fruitBasket = [
    'banana',
    'cherry',
    'orange',
    'apple',
    'cherry',
    'orange',
    'apple',
    'banana',
    'cherry',
    'orange',
    'fig'
];

fruitBasket.reduce((tally, fruit) => {
    if (!tally[fruit]) {
      tally[fruit] = 1;
    } else {
      tally[fruit] = tally[fruit] + 1;
    }
    return tally;
  }, {});
```

```
const count = fruitBasket.reduce( (tally, fruit) => {
    tally[fruit] = (tally[fruit] || 0) + 1 ;
    return tally;
} , {})

count // { banana: 2, cherry: 3, orange: 3, apple: 2, fig: 1 }
```

flat array

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

const flat = data.reduce((total, amount) => {
  return total.concat(amount);
}, []);

flat // [ 1, 2, 3, 4, 5, 6, 7, 8, 9 ]
```

```
const data = [
  { a: "happy", b: "robin", c: ["blue", "green"] },
  { a: "tired", b: "panther", c: ["green", "black", "orange", "blue"] },
  { a: "sad", b: "goldfish", c: ["green", "red"] }
];
```

```
const colors = data.reduce((total, amount) => {
  amount.c.forEach(color => {
    total.push(color);
  });
  return total;
}, []);

colors; //['blue','green','green','black','orange','blue','green','red']
```

```
const uniqueColors = data.reduce((total, amount) => {
  amount.c.forEach( color => {
    if (total.indexOf(color) === -1){
     total.push(color);
    }
  });
  return total;
}, []);

uniqueColors // [ 'blue', 'red', 'green', 'black', 'orange']
```

```
const fruitBasket = [
        'banana',
        'cherry',
        'orange',
        'apple',
        'cherry',
        'orange',
        'apple',
        'banana',
        'cherry',
        'orange',
        'fig'
    ];

const count = fruitBasket.reduce( (tally, fruit) => {
  tally[fruit] = (tally[fruit] || 0) + 1 ;
  return tally;
} , {})

count // { banana: 2, cherry: 3, orange: 3, apple: 2, fig: 1 }
```

pipeline

```
const numberArray = [1,2,3,4,5,6,7,8,9,10];

const evenNumberFilterFn = (number => number%2===0);
const squareMapFn = (number => number*number);
const sumFn = ((sum, number) => sum + number);

const sumOfSquareOfEvenNumbers = numberArray
                                 .filter(evenNumberFilterFn)
                                 .map(squareMapFn)
                                 .reduce(sumFn,0);

console.log(sumOfSquareOfEvenNumbers);
```

```
function increment(input) { return input + 1;}
function decrement(input) { return input - 1; }
function double(input) { return input * 2; }
function halve(input) { return input / 2; }
```

```
let pipeline = [increment, double, decrement];
```

```
function increment(input) { return input + 1;}
function decrement(input) { return input - 1; }
function double(input) { return input * 2; }
function halve(input) { return input / 2; }
```

```
const result = pipeline.reduce(function(total, function) {
  return function(total);
}, 1);

result // 3
```

Reduce (Composing Software) ×

A Medium Corporation [US] | https://medium.com/javascript-scene/reduce-composing-software-fe22f0c39a1d

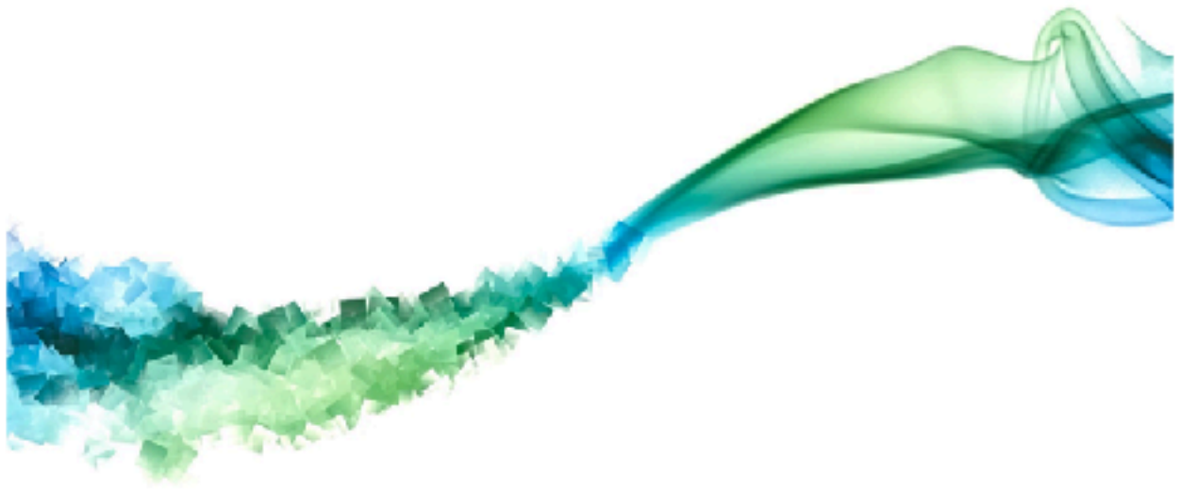
Apps | Javascript : Articles | Jenkins | CC board - Agile Bo... | Ferramentas | tempo | H5O v0.12.2 | Functional Approach... | Defining Grid Areas | CSGO-Skins.com – S... | Outros favoritos

Eric Elliott Follow

Compassionate entrepreneur on a mission to end homelessness.

Mar 5, 2017 · 5 min read

Reduce (Composing Software)



Smoke Art Cubes to Smoke — MattysFlicks — (CC BY 2.0)

técnicas não são regras

map, filter, and reduce explained with emoji 😂

map([🐮, 🍠, 🐔, 🌽], cook)

=> [🍔, 🍟, 🍗, 🍿]

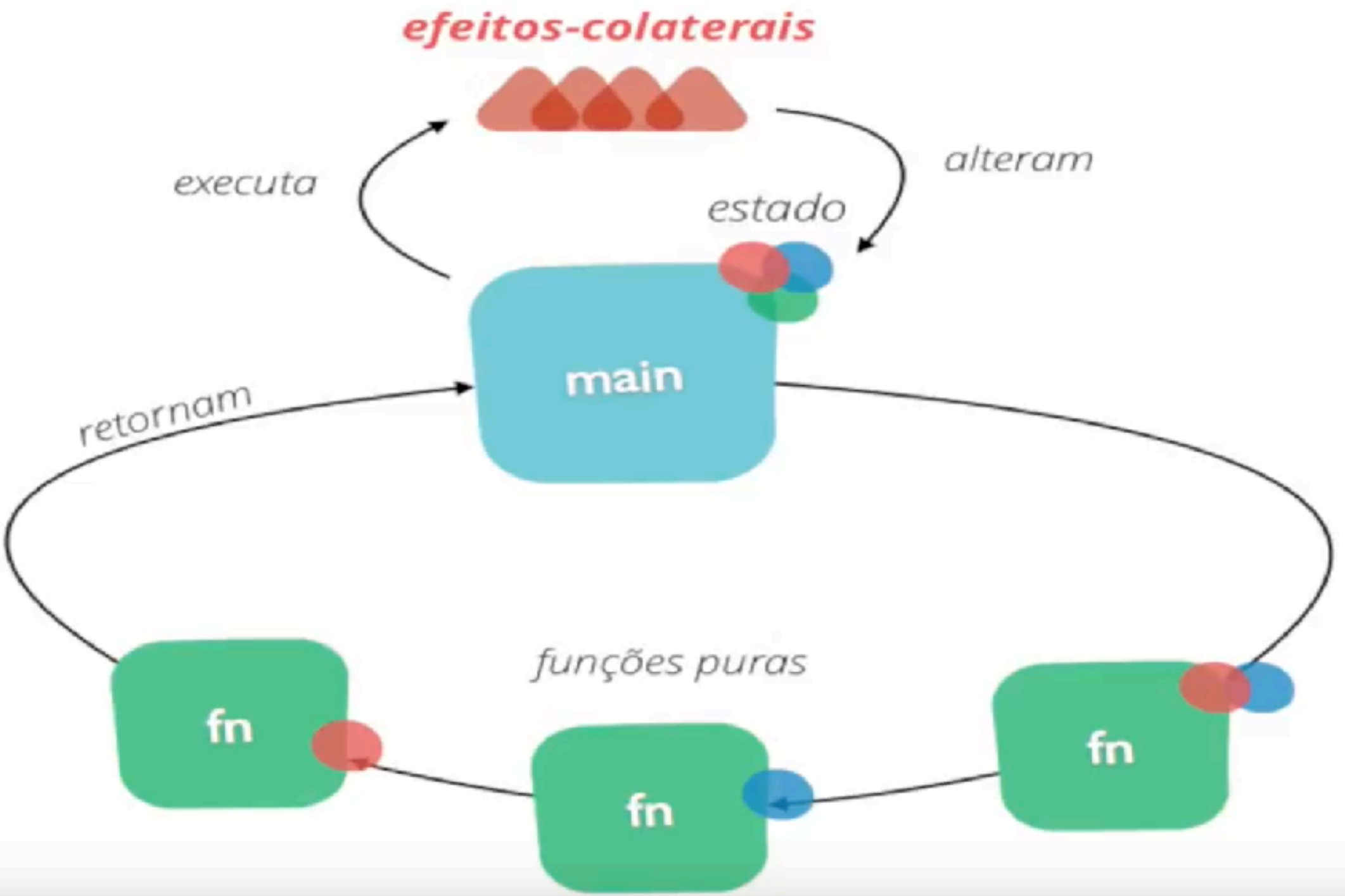
filter([🍔, 🍟, 🍗, 🍿], isVegetarian)

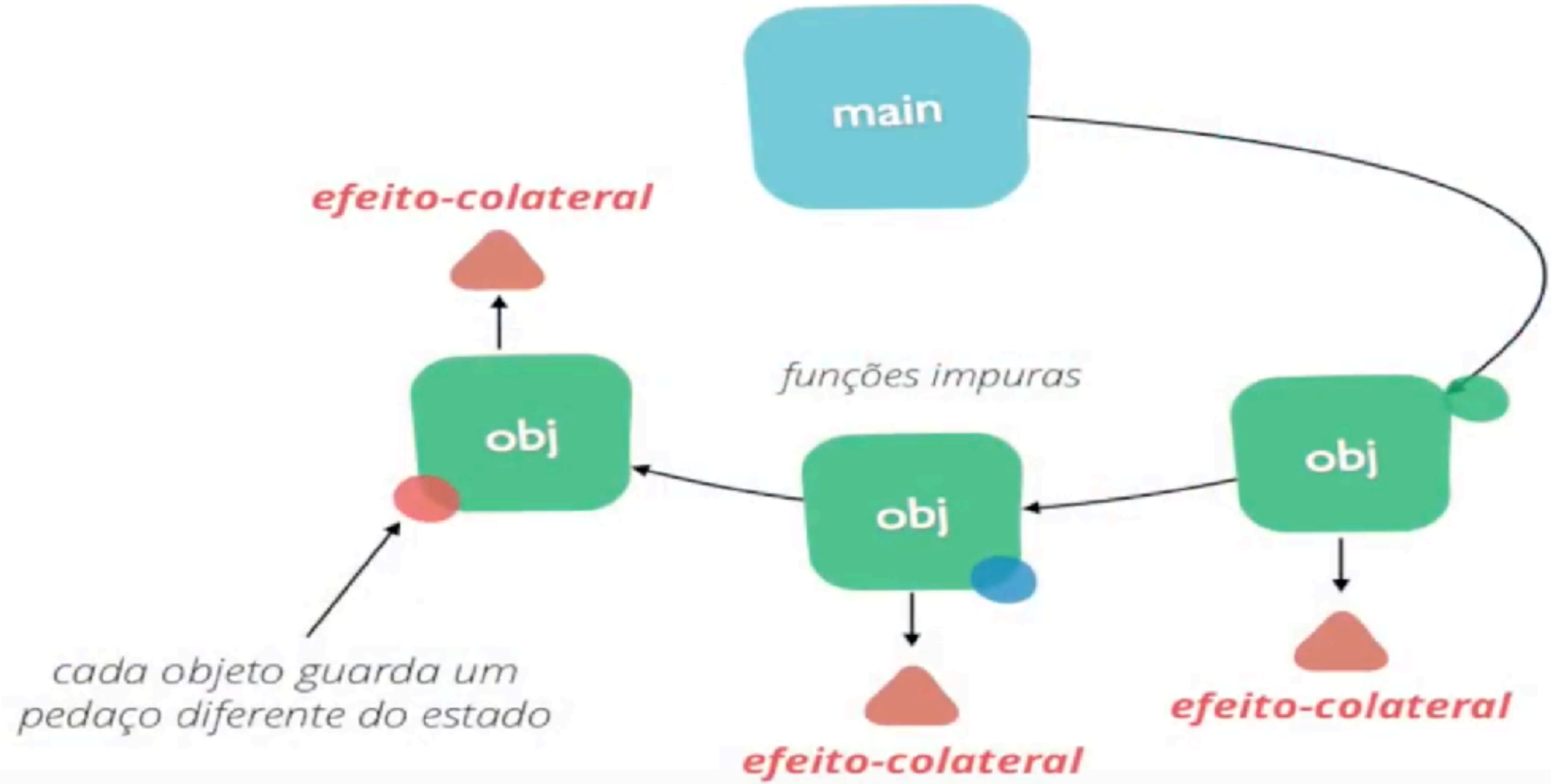
=> [🍟, 🍿]

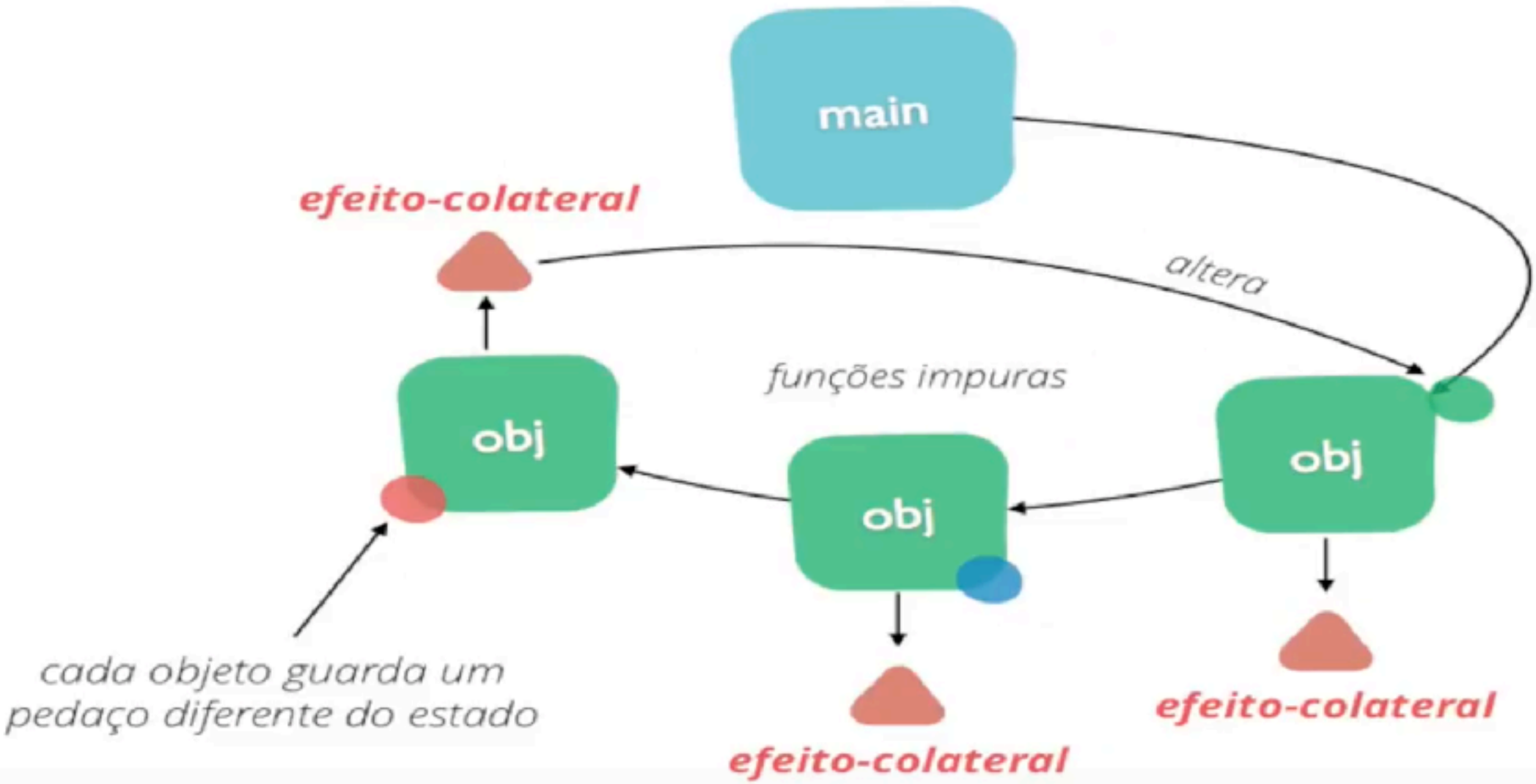
reduce([🍔, 🍟, 🍗, 🍿], eat)

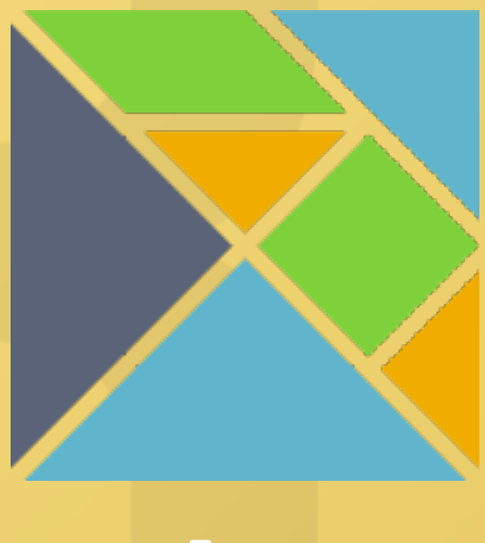
=> 💩

compose && currying

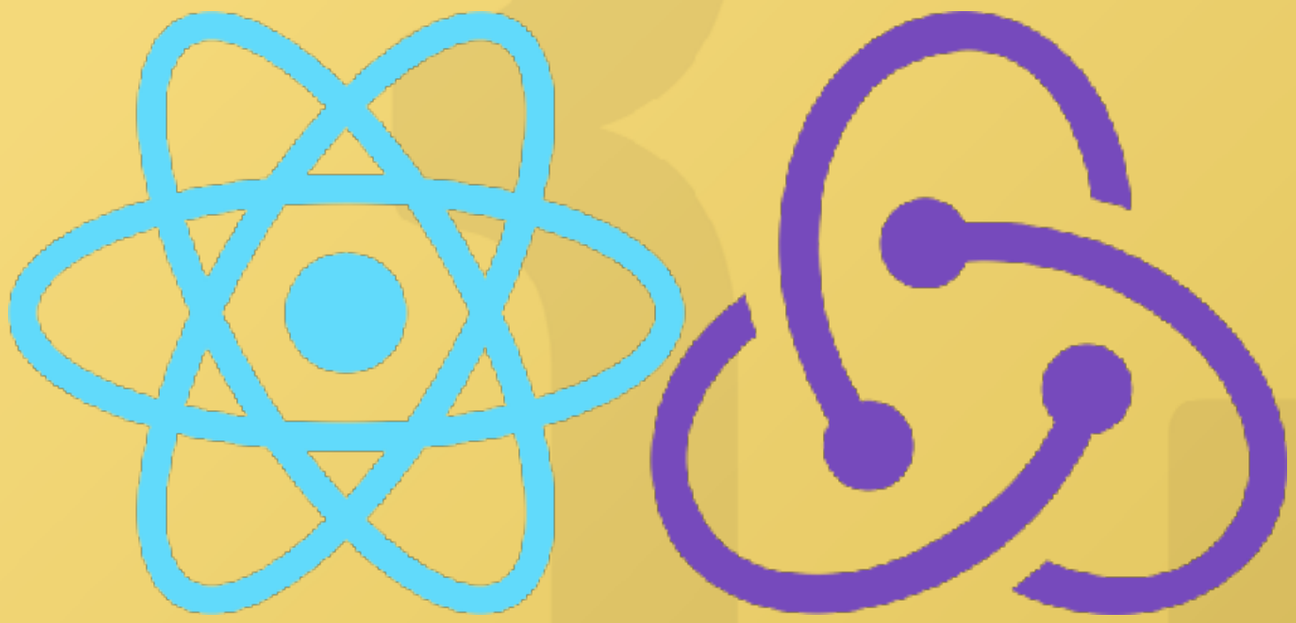

efeitos-colaterais
executa
alteram
estado
main
retornam
funções puras
fn
fn
fn

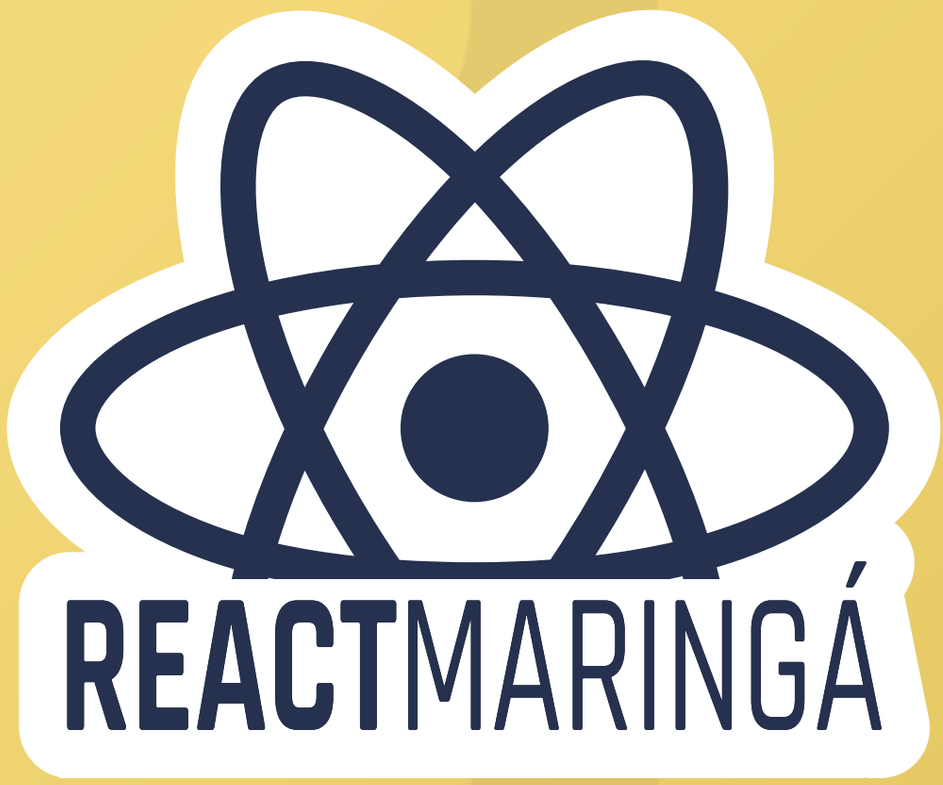

main
efeito-colateral
funções impuras
obj
obj
obj
cada objeto guarda um
pedaço diferente do estado
efeito-colateral
efeito-colateral
efeito-colateral


main
efeito-colateral
altera
funções impuras
obj
obj
obj
cada objeto guarda um
pedaço diferente do estado
efeito-colateral
efeito-colateral



elm




REACTMARINGÁ

muito obrigado