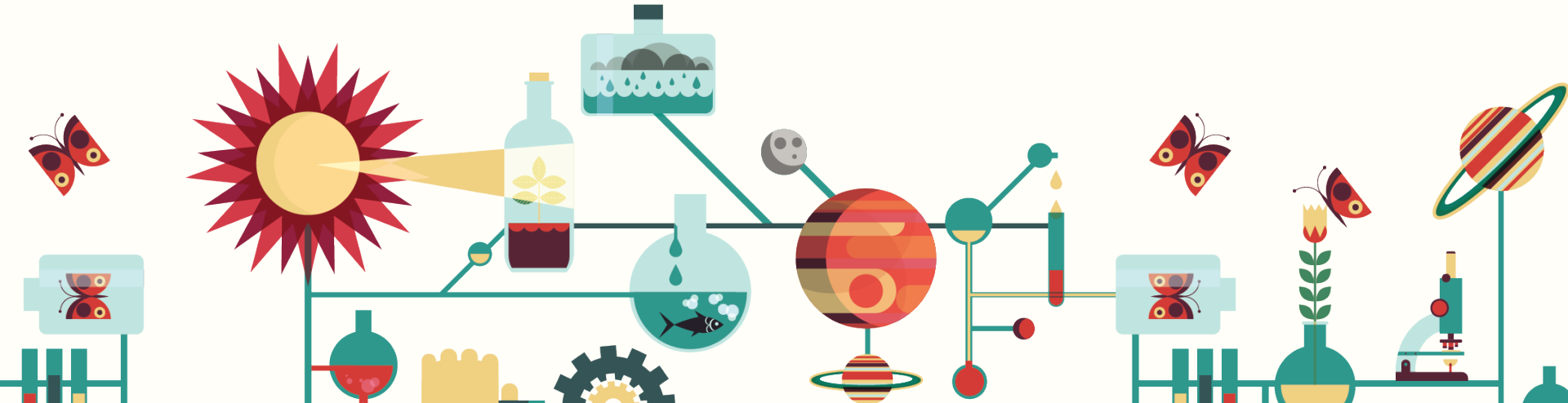
versal

Real-World Functional Programming

James Earl Douglas
@jearldouglas

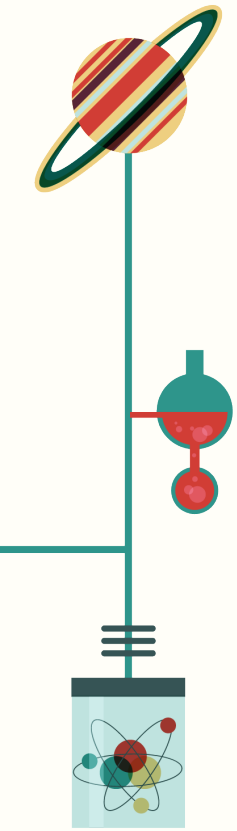
Kelley Robinson
@kelleyrobinson

bit.ly/real-world-fp



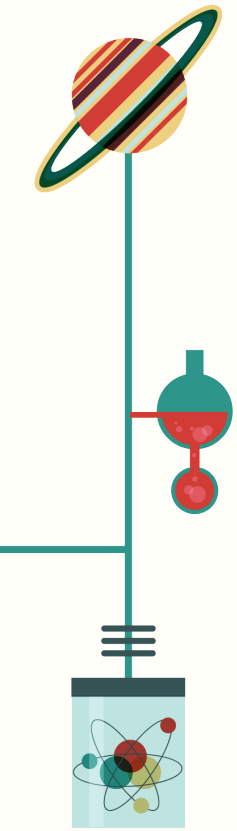
Key Concepts

Characteristics that differentiate functional programming



- Statelessness
- Immutable data
- Referential transparency

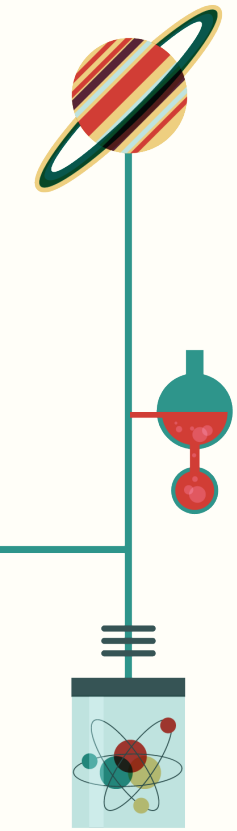
Statelessness



```
function add(a,b) {
  return a + b;
}

var x = add(1,1) // 2
var y = add(1,1) // 2
var z = add(1,1) // 2
```

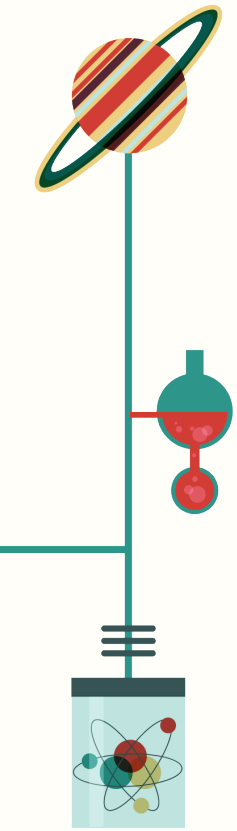
`add` always returns the same output for a given input

Immutable data



```
var x = "hello"              // "hello"

var y = x + ", world"        // "hello, world"

var z = y.substring(0,5)     // "hello"
```

Referential transparency



```
var x = 1
```

- *x* is a synonym for *1*

```
var y = x + 1
```

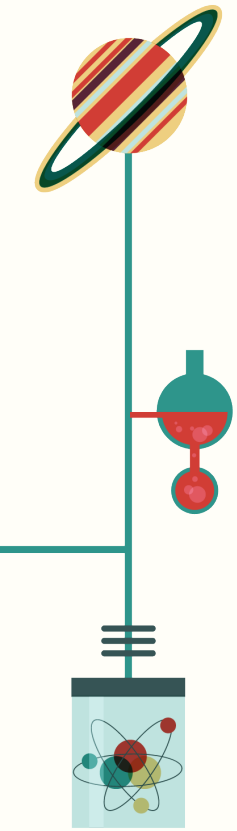
- *y* is a synonym for *x + 1*
- *y* is also a synonym for *2*

```
x = 2
```

- This is a lie, equivalent to *1 = 2*

Benefits

Why you should use functional programming

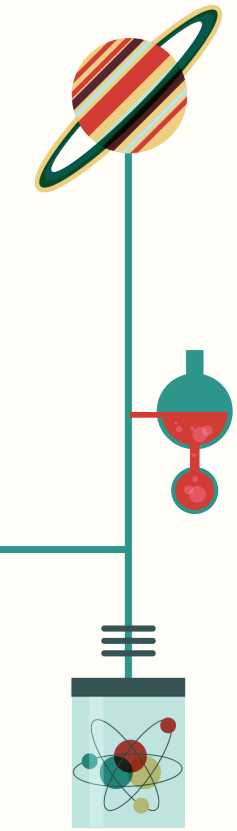


- Easily abstractable
- Applications become modular and composable
- Encourages code reuse

We had a problem . . .

Architectural issues

Existing API design had limitations

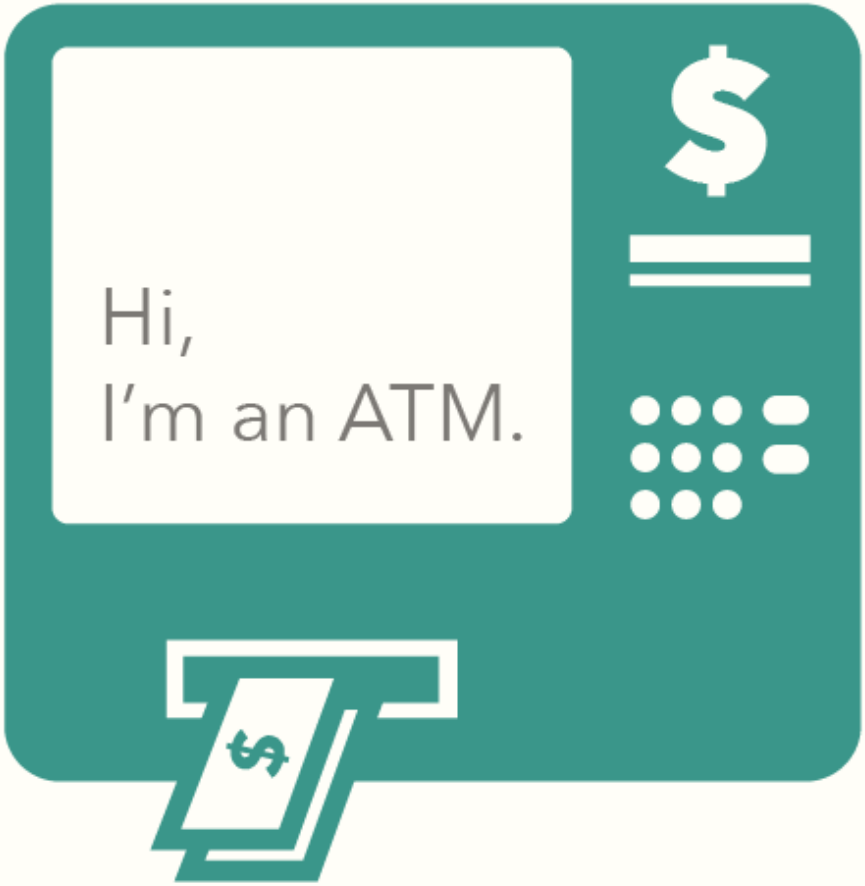


- Locked into using one data structure
- Clumsy persistent data model
- Bottlenecked deployment

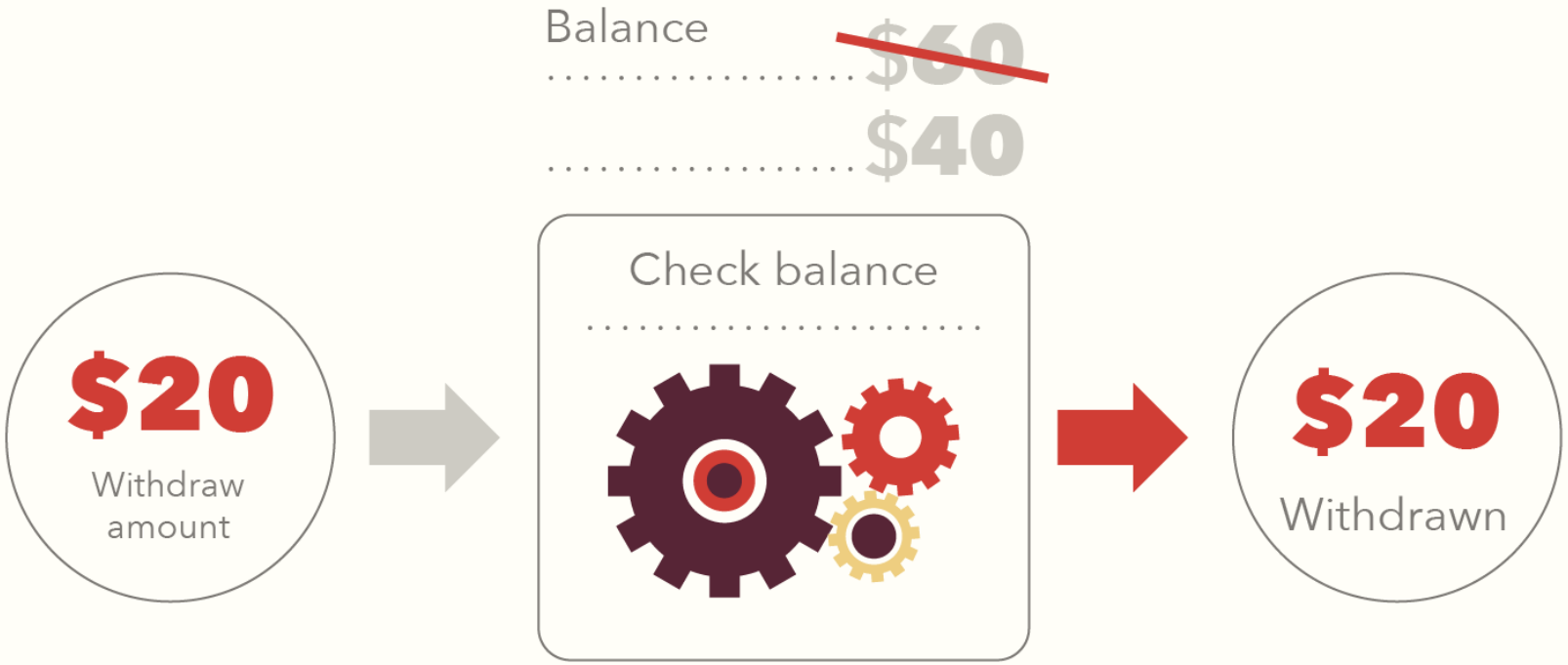
Let’s refactor

Using functional concepts to solve our problems.

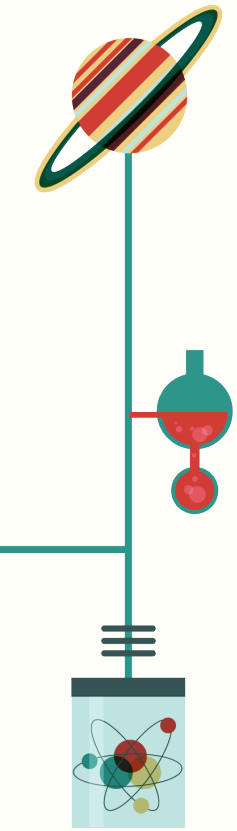
Example - ATM



Withdraw - bad!

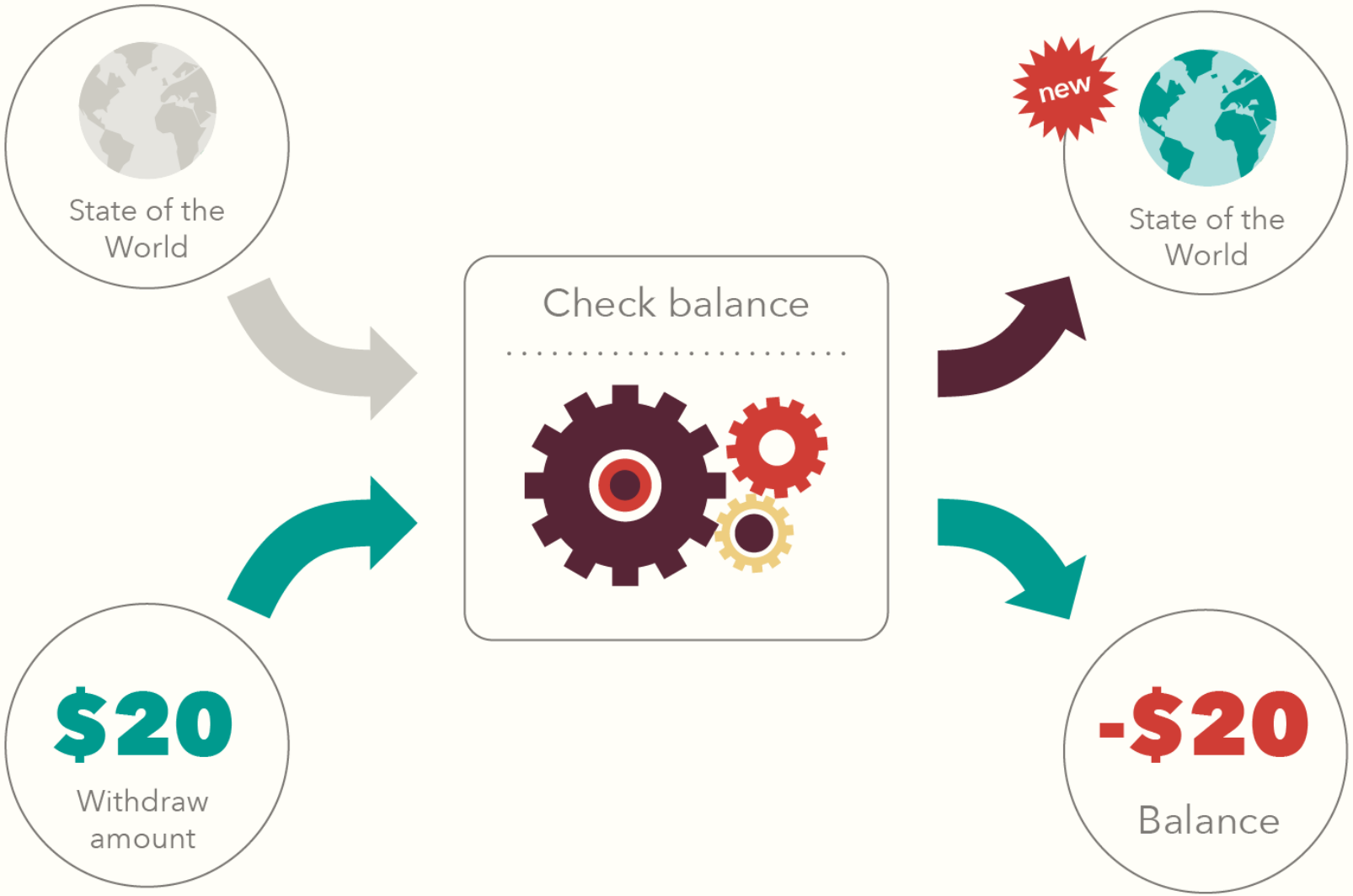


Withdraw - bad!

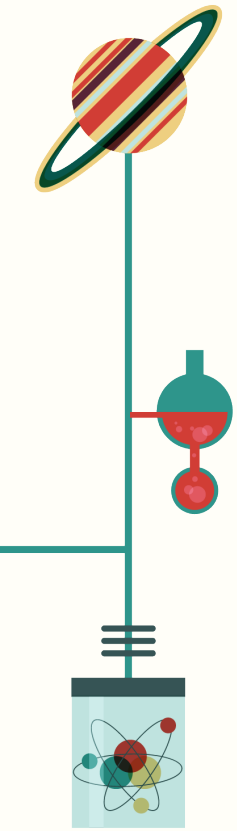


```
function withdraw(amount) {
  if (balance >= amount) {
    balance = balance - amount
    return amount
  } else {
    return 0
  }
}
```

Withdraw - good!

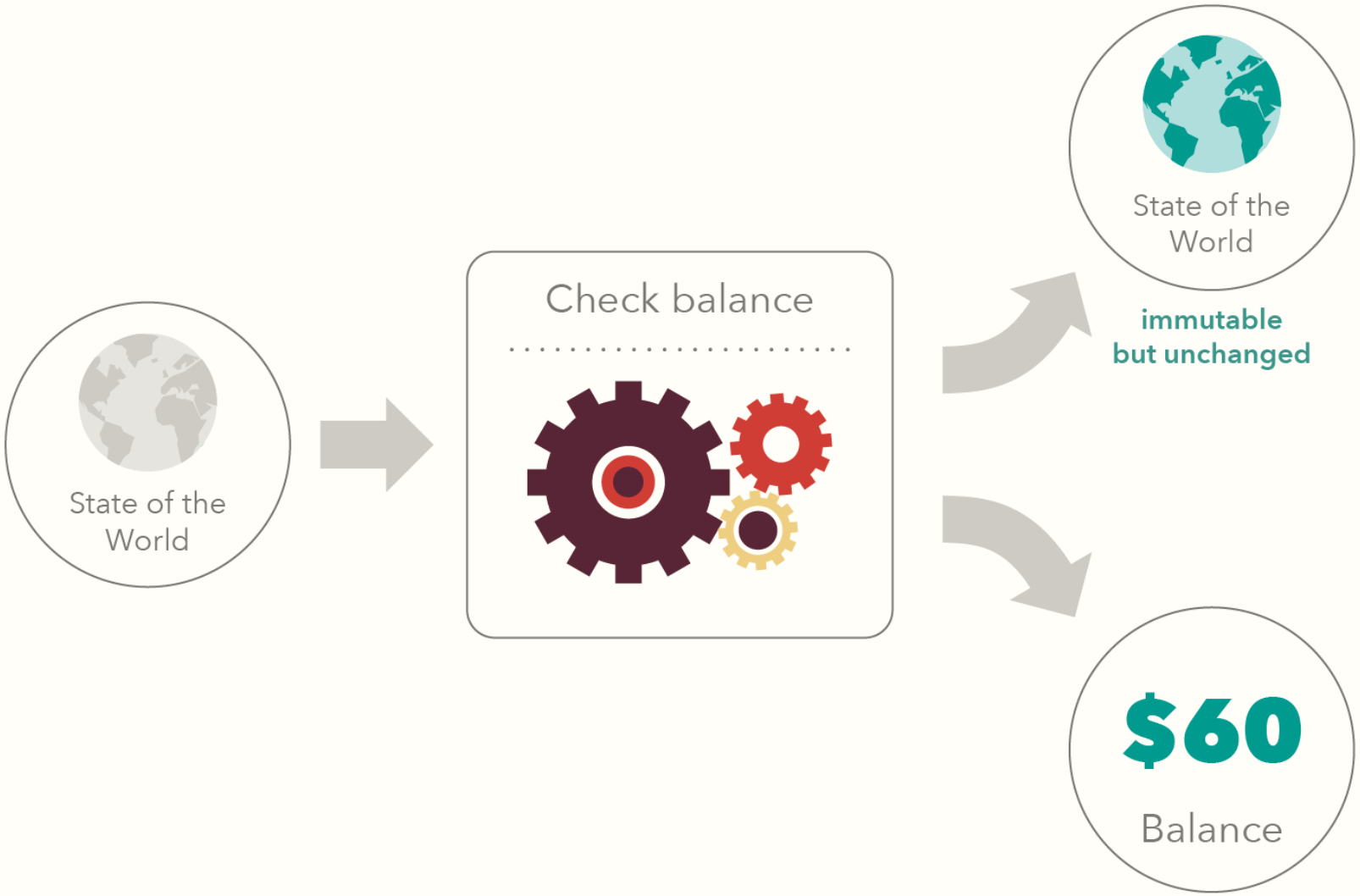


Withdraw - good!

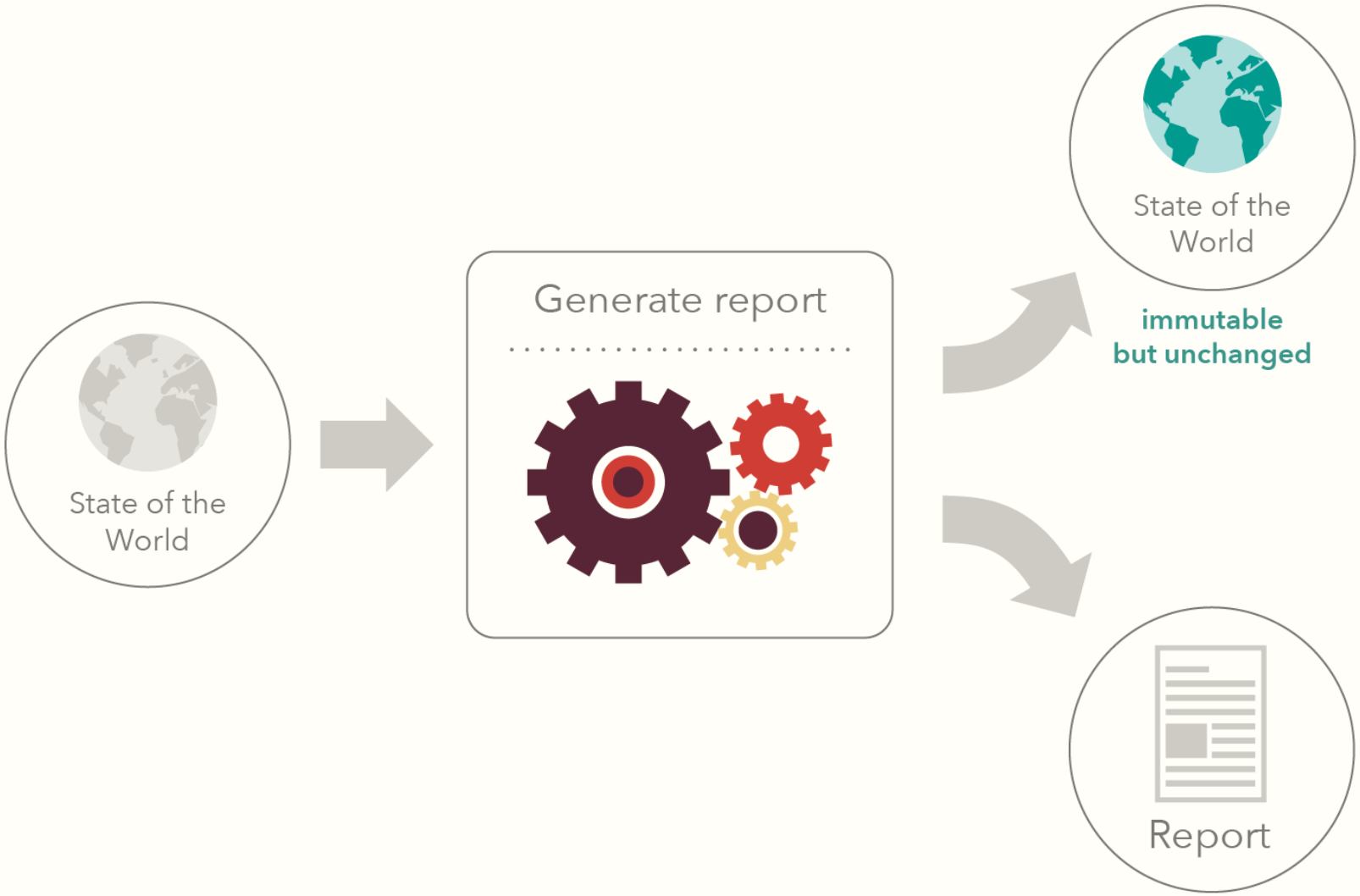


```
function withdraw(amount) {
  return function(balance) {
    if (balance >= amount) {
      return [amount, balance - amount]
    } else {
      return [0, balance]
    }
  }
}
```

Check balance

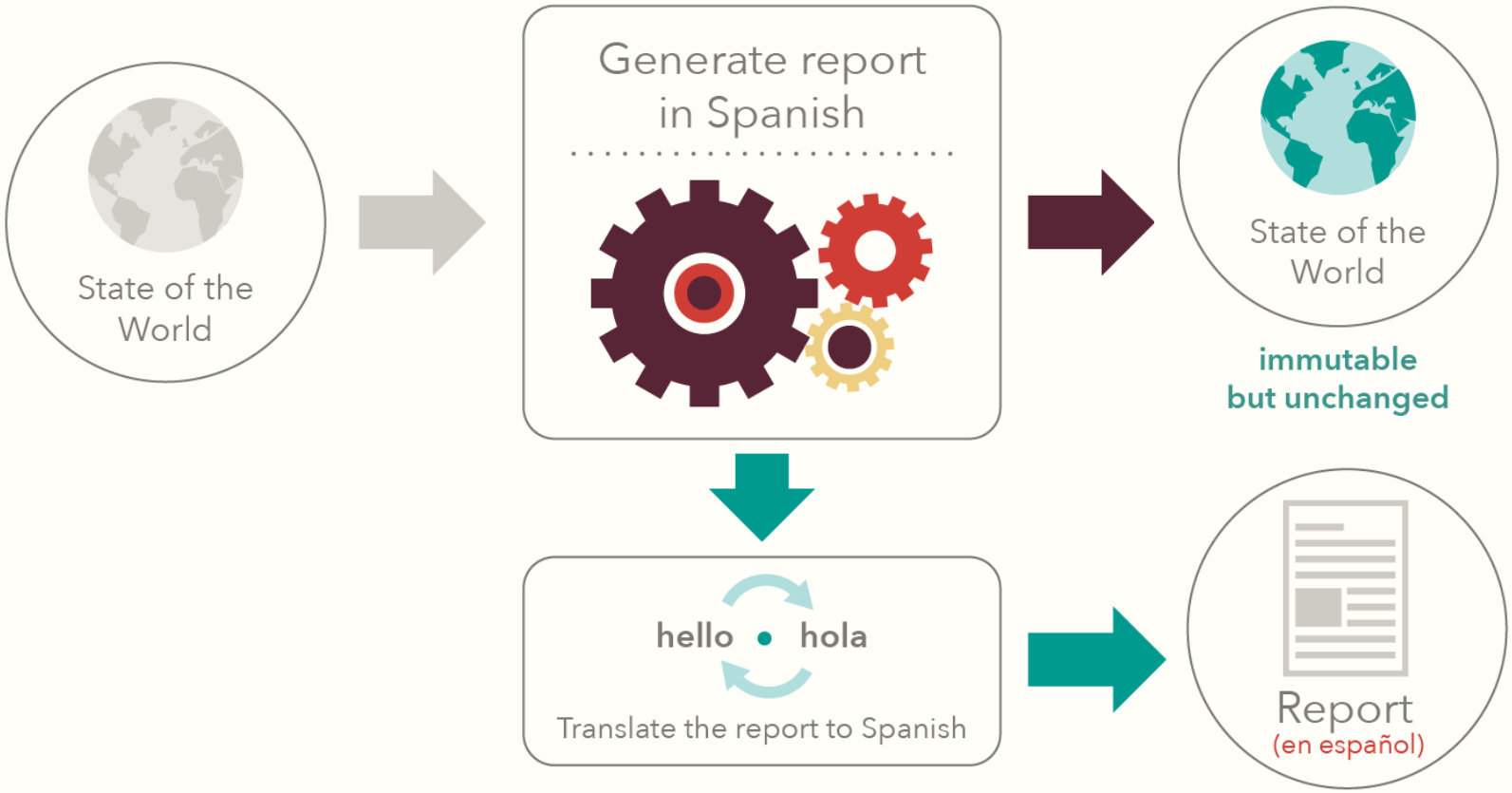


Generate report

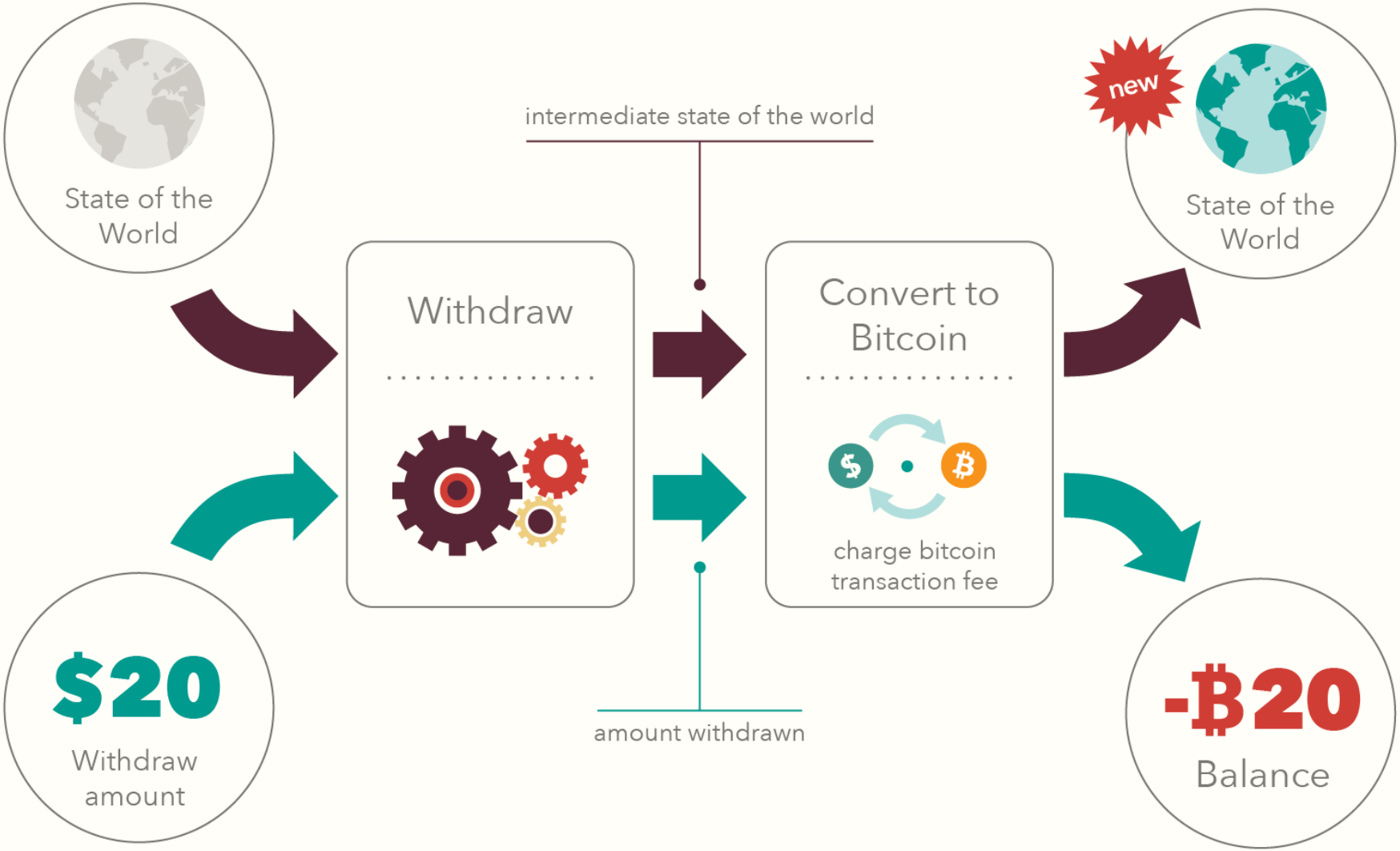


Generate report

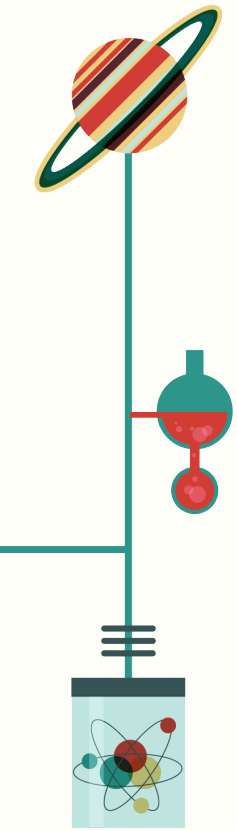
(in Spanish)



Withdraw - in Bitcoin



Withdraw - in Bitcoin



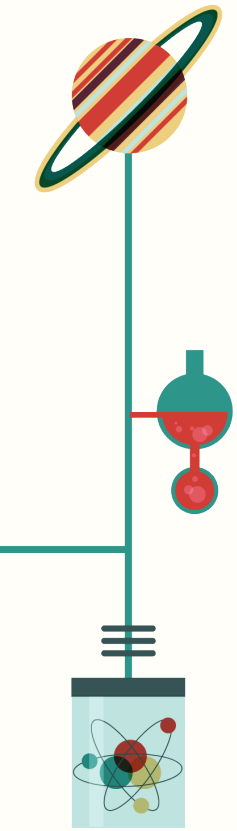
```
function convertToBtc(withdrawal) {
  return function(balance) {
    var result = withdrawal(balance)
      // [amount, new balance]
    var inBtc = result[0] / 575.0
    var fee = result[0] * 0.01
    return [inBtc, result[1] - fee]
  }
}

var get20InBtc = convertToBtc(withdraw(20))
```

Live code time

bit.ly/real-world-fp

Benefits

Why you should use functional programming



- Easily abstractable
- Applications become modular and composable
- Encourages code reuse

Thank you!

james@versal.com
@jearldouglas

kelley@versal.com
@kelleyrobinson

bit.ly/real-world-fp