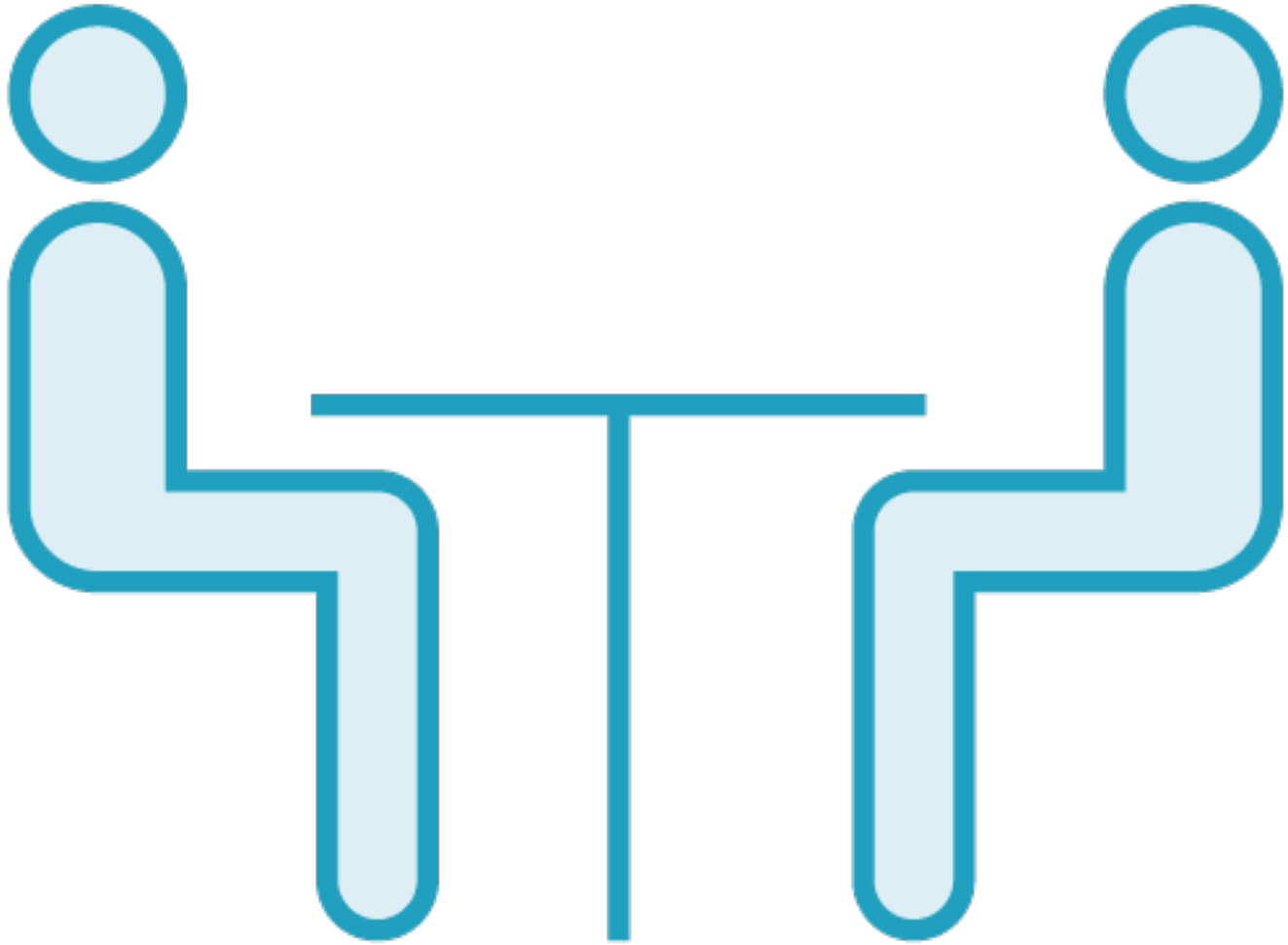
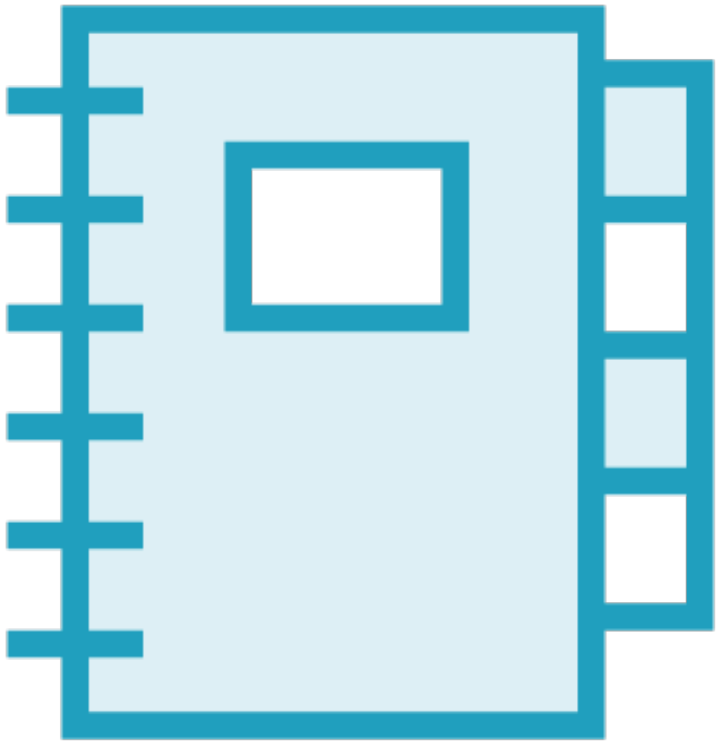
Side Effects May Be Harmful

Nathan Taylor
SOFTWARE ENGINEER

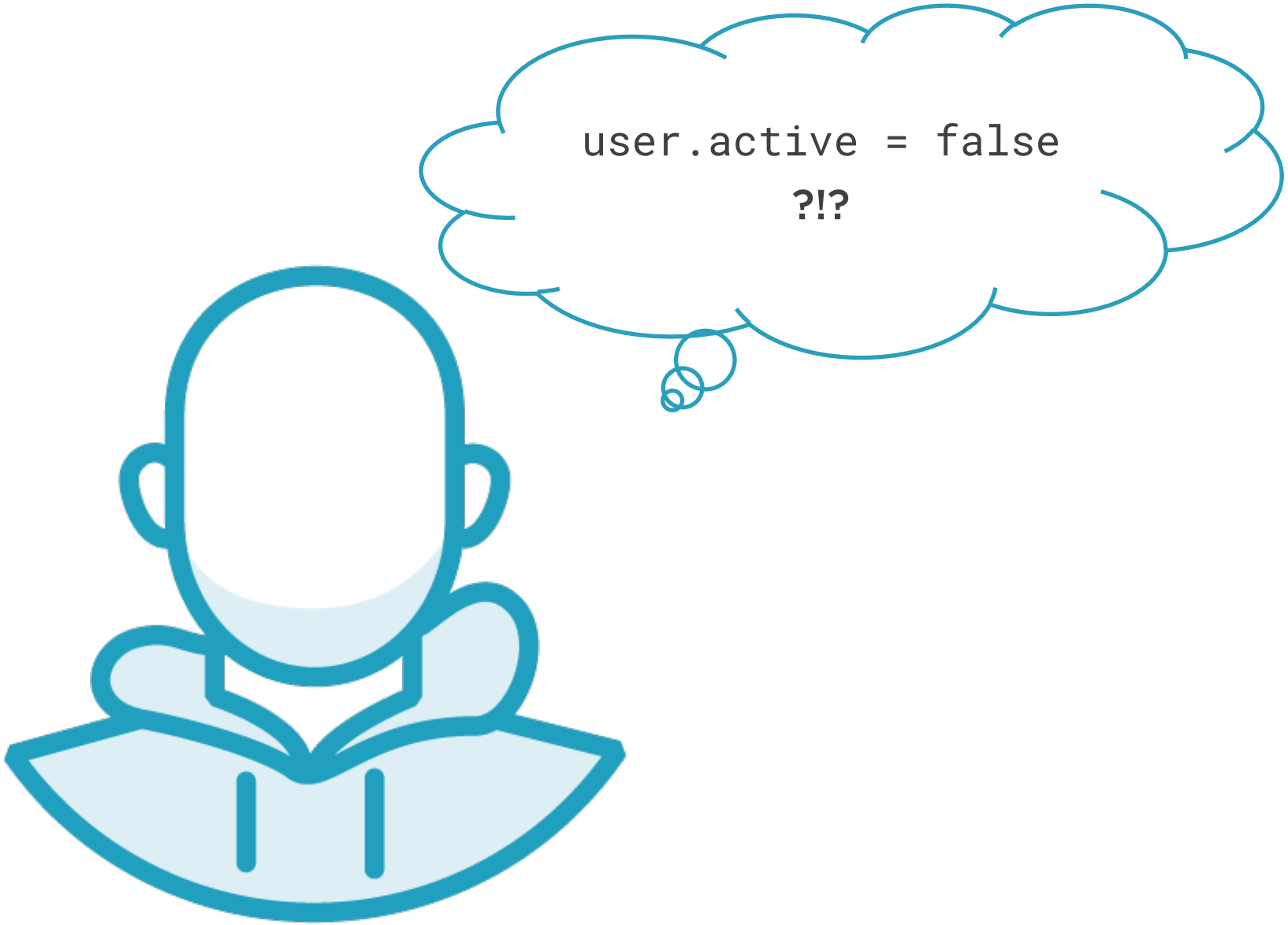
@taylonr taylonr.com

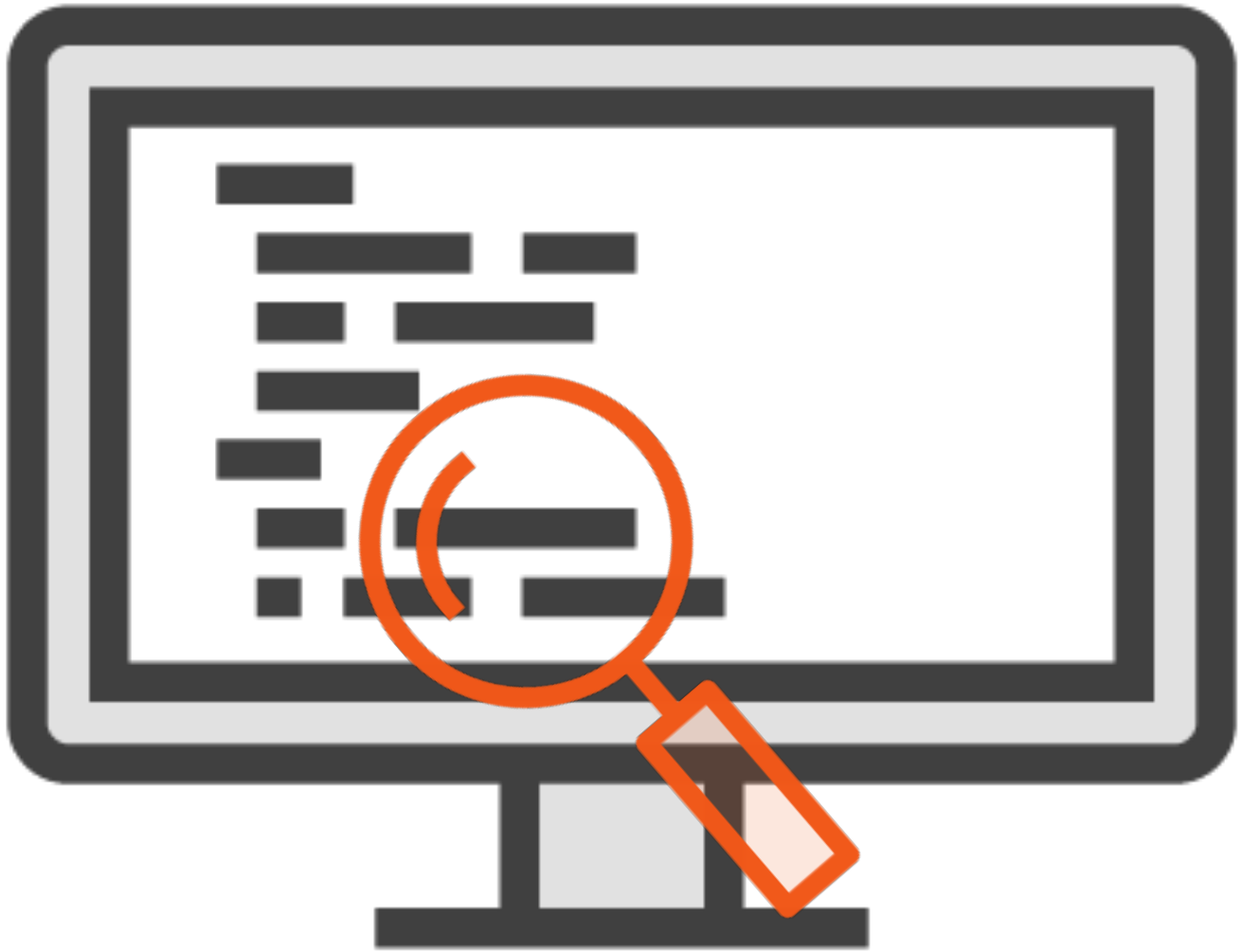


I wish I could clone you!



Fewer debugging notes


user.active = false
?!?



We're going to have some openings

Ok.

What's up?

Juggling breakpoints.

What vs. How

Functions

Data Doesn't Change

What Are Side Effects?

Pure Function

1. Doesn’t depend on any data other than what it’s passed
2. Doesn’t modify any data other than what they return

If it modifies some state
outside its scope

```
if(user.password !== saltedPassword(password)){
    this.invalidLogins++;
}
```

Function with a side effect

```
if(user.password !== saltedPassword(password)){
    this.invalidLogins++;
}
```

Function with a side effect

```
$http.get('myApi/users')
```

Another function with a side effect

A program must have side effects

```
customer.billForMonth(MARCH, 127.28);
```

```
customer.billForMonth(MARCH, 127.28);
customer.save();
```

```
const billForMonth(month, monthlyBalance){
    customer.balance += monthlyBalance;
    if(customer.balance > 1000){
        customer.inactive = true;
    }
}
```

Side effects can slow debugging

Why is Immutability Important?

The goal is not to eliminate all side effects

Immutable String

```
while(!file.EOF){
    text += file.readLine();
}
```

Immutable structures reduce complexity

```
users.sort(usernameAscending);
```

How does sort work?

Sorts in place

```
const sortedUsers = R.sortBy(usernameAscending, users);
```

How does sort work?

Returns a new (sorted) array

```
let numbers = [1, 3, 2, 4];

let sortedNumbers = numbers.sort();
```

```
let numbers = [1, 3, 2, 4];

let sortedNumbers =
numbers.sort();

numbers.pop();
```

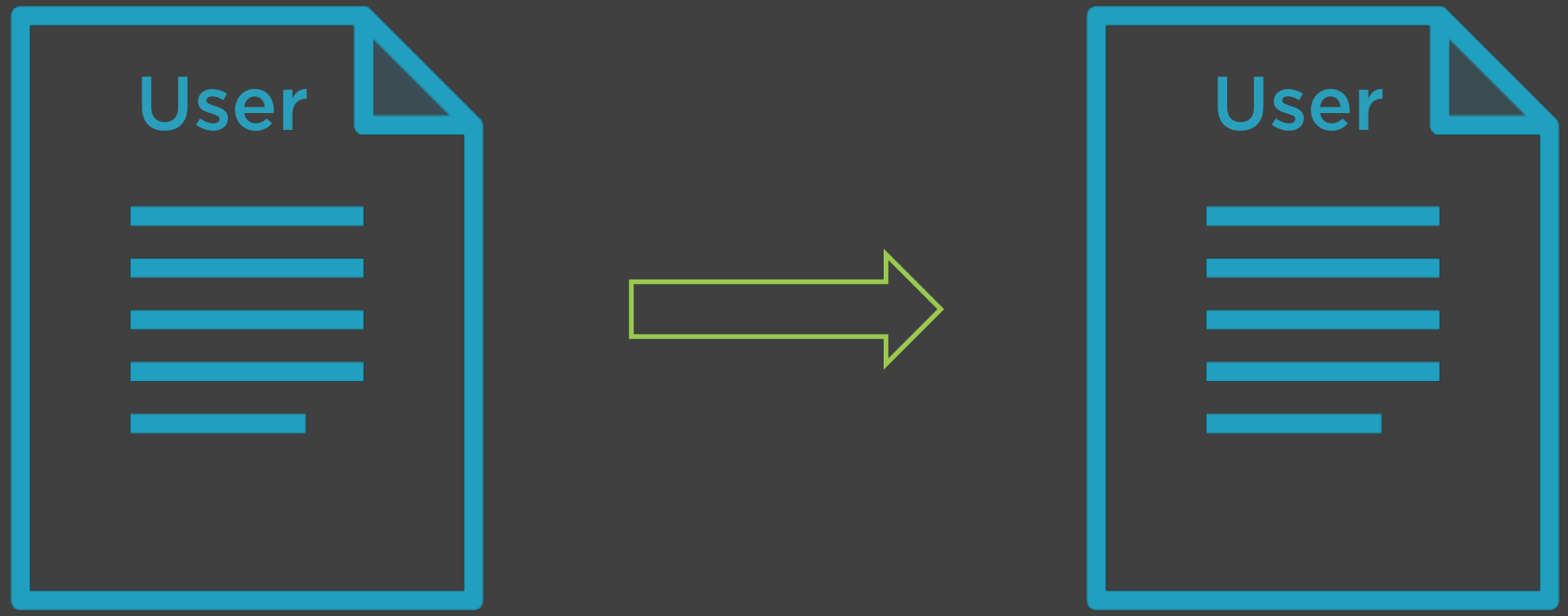
◀ numbers = [1,2,3]
◀ sortedNumbers = [1,2,3]

More certainty

Locking down increases comprehension

Immutability and Performance

What about performance?


User
User

Too Many Copies?

```
{
    name: ‘Charles Spurgeon’,
    email:
    ‘charles@mailinator.com’,
    account: { … },
    address: { … },
    jobs: [{...}. {...}]
}
```

```
{
    name: ‘Charles Spurgeon’,
    email:
    ‘charles@mailinator.com’,
    account: 0x12345678,
    address: 0x23456789,
    jobs: 0x3456789A
}
```

```
{
    name: ‘Charles Spurgeon’,
    email:
    ‘charles@mailinator.com’,
    account: { … },
    address: { … },
    jobs: [{...}. {...}]
}
```

```
{
    name: ‘Charles Spurgeon’,
    email:
    ‘charles@mailinator.com’,
    account: 0x12345678,
    address: 0x234567BB,
    jobs: 0x3456789A
}
```

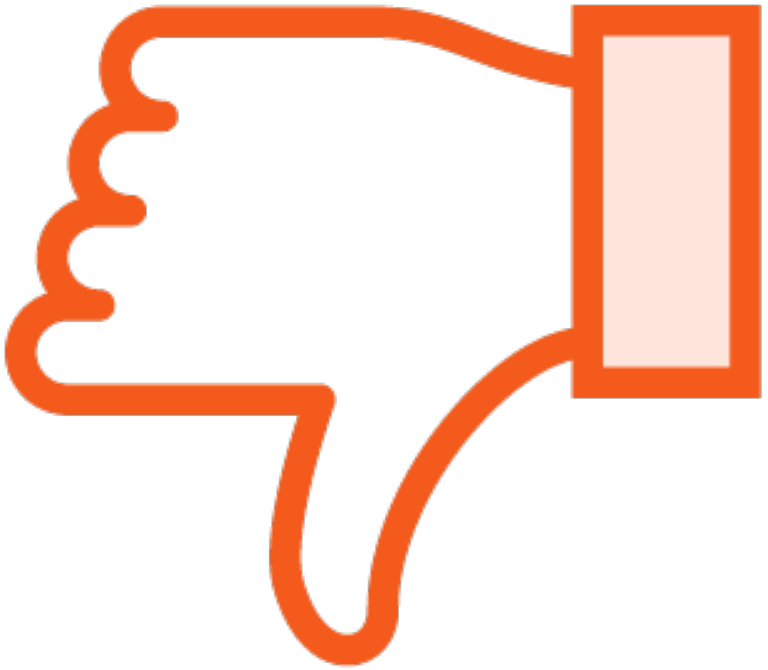
```
{
    name: ‘Charles H.
    Spurgeon’,
    email:
    ‘charles@mailinator.com’,
    account: 0x12345678,
    address: 0x23456789,
    jobs: 0x3456789A
}
```

- Name changed
- References did not

Object Reuse

Garbage Collection

Seamless Immutable



No native JavaScript immutable structures

```
const myObj = {};

myObj.firstName = ‘Nate’;

delete myObj.firstName;
```

```
const myOtherObj = {};
myObj = myOtherObj;
```

Assignment to constant variable

seamless-immutable

Mutable Array

```
let myArray = [3, 5, 1, 2, 4];
myArray.sort(); // [1, 2, 3, 4, 5]
```

Immutable Array

```
let myImmutable = Immutable([3, 5, 1, 2, 4]);

myImmutable.sort(); //sort cannot be called on immutable
                    structure
```

Immutable Array & Ramda

```
const diff = (a, b) => { return a - b; }

R.sort(diff, myImmutable);
```

```
myImmutable.map((x) => {
    console.log(x);
});
```

◀ 3, 5, 1, 2, 4

```
myImmutable.reduce((acc, x)=>
{
    return acc += x;
}, 0);
```

◀ 15

```
myImmutable
    .asMutable()
    .push(10);


console.log(myImmutable);
```

```
◀ [3, 5, 1, 2, 4]
```

```
myImmutable
    .asMutable()
    .push(10);


console.log(myImmutable);


mutableObj =
myImmutable.asMutable();

mutableObj.push(10);

console.log(mutableObj)
```

◀ [3, 5, 1, 2, 4]

◀ [3, 5, 1, 2, 4, 10]

Not just arrays

```
const immutableObj = Immutable({firstName: ‘Nate’});

immutableObj.lastName = 'Taylor';

console.log(immutableObj);
```

Immutable Object

```
{
  firstName: ‘Nate’
}
```

```
const mutableObj = immutableObj.asMutable();

mutableObj.lastName = 'Taylor';

console.log(mutableObj);
```

asMutable()

```
{
   firstName: ‘Nate’,
   lastName: ‘Taylor’
}
```

Set

```
immutableObj.set('age', 39);
```

Set

```
const mutableObj = immutableObj.set('age', 39);
```

Setin

```
const user = Immutable({
    username: ‘taylonr’,
});
```

Setin

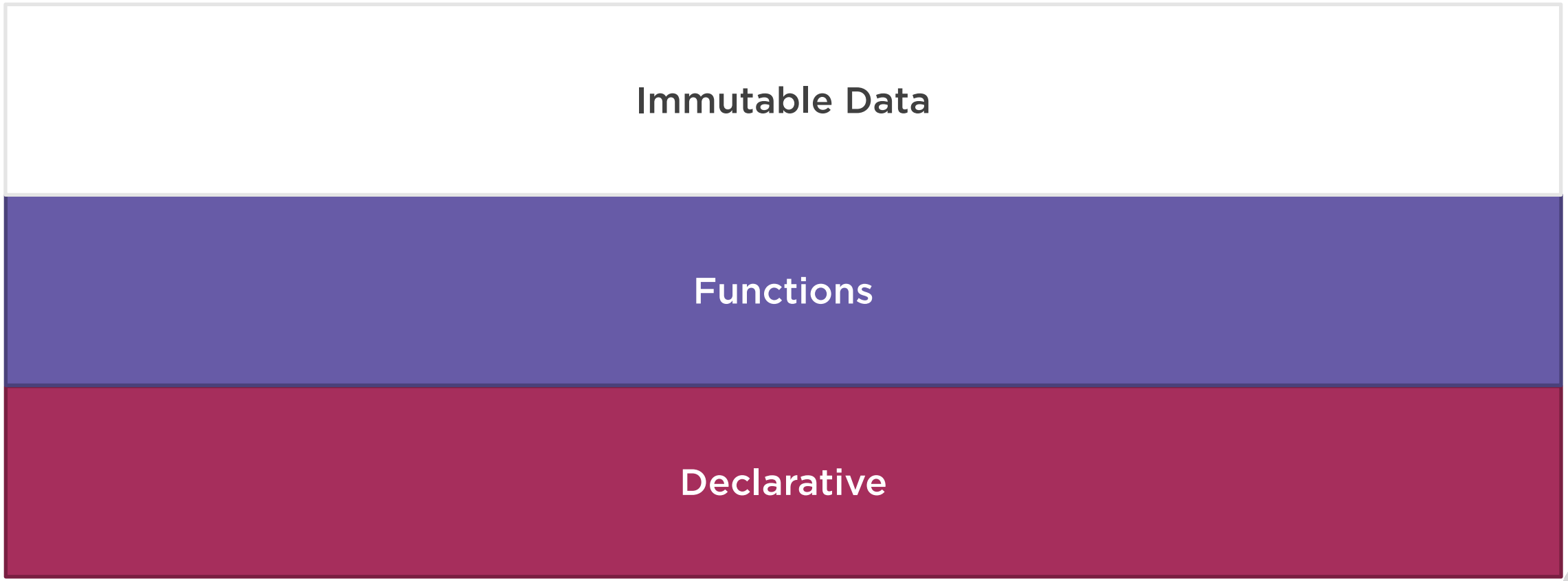
```
const user = Immutable({
    username: ‘taylonr’,
});

const newUse = user.setIn([‘account’, ‘address’, ‘city’],
‘Omaha’)
```

```
{
  username: ‘taylonr’,
  account: {
    address: {
       city: ‘Omaha’
    }
  }
}
```

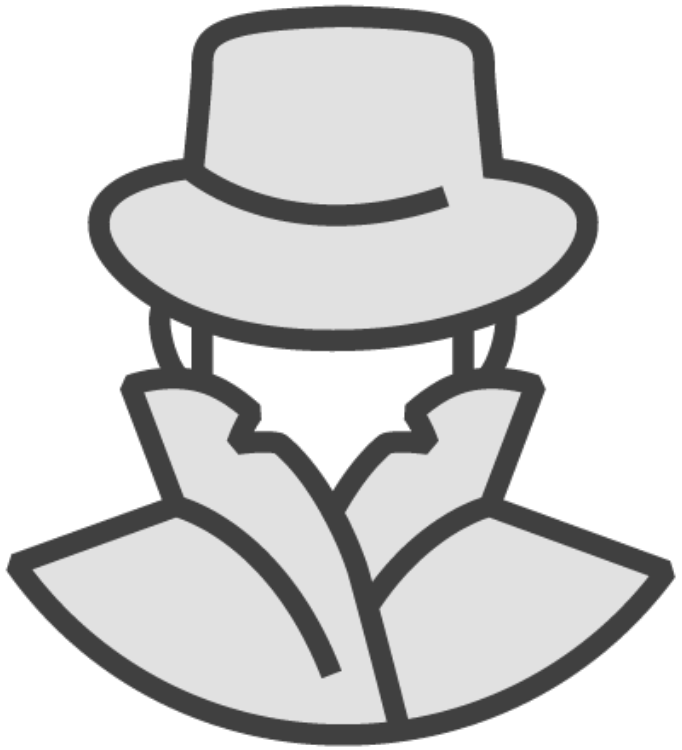
Check out seamless-immutable

github.com/rtfeldman/seamless-immutable

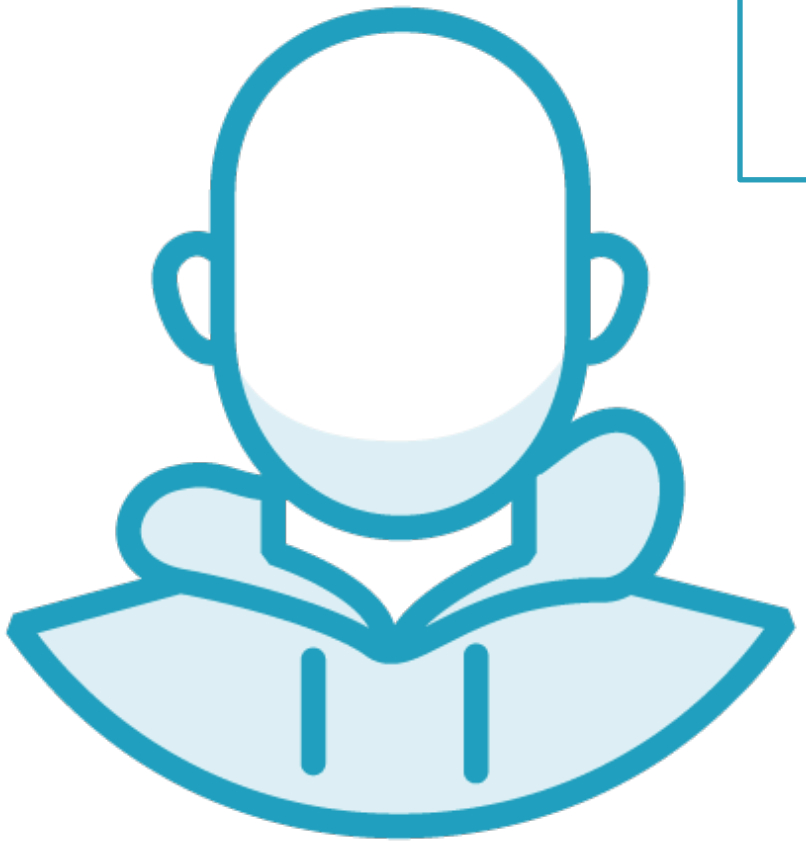
Summary


Immutable Data
Functions
Declarative

Immutable data reduces complexity



Reduce the number of side effects



Help!

Install seamless-immutable

Limit Scope

Expose with asMutable()

Control Adoption