Property-based testing

hands-on workshop

@jovaneyck@mastodon.social

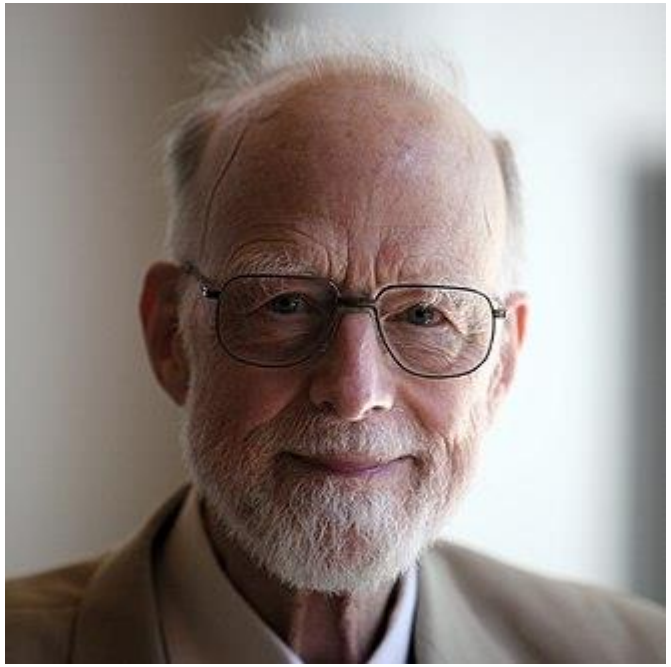
```
const sum = require('./sum');

test('adds 1 + 2 to equal 3', () => {
  expect(sum(1, 2)).toBe(3);
});
```


-38?
1.2114e-38?
Infinity?
NaN?

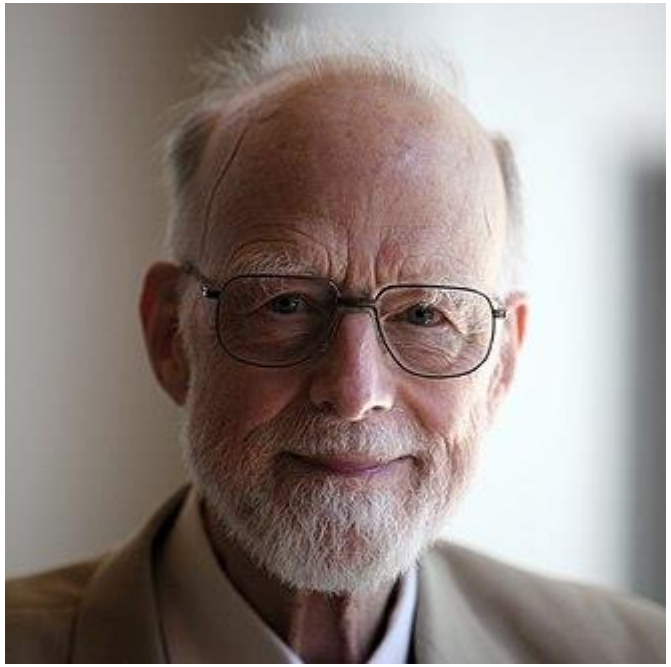
```
{ N ≥ 1 }
{ 1 = 1! && 0 ≤ 1 ≤ N }
k := 1
{ 1 = k! && 0 ≤ k ≤ N }
f := 1
{ f = k! && 0 ≤ k ≤ N }
while (k < N) do
    { f = k! && 0 ≤ k ≤ N && k < N && N-k = V}
    { f*(k+1) = (k+1)! && 0 ≤ k+1 ≤ N && N-(k+1) < V}
    k := k + 1
    { f*k = k! && 0 ≤ k ≤ N && N-k < V}
    f := f * k
    { f = k! && 0 ≤ k ≤ N && N-k < V}
end
{ f = k! && 0 ≤ k ≤ N && k ≥ N}
{ f = N! }
```

```
{ N ≥ 1 }
{ 1 = 1! && 0 ≤ 1 ≤ N }
k := 1
{ 1 = k! && 0 ≤ k ≤ N }
f := 1
{ f = k! && 0 ≤ k ≤ N }
while (k < N) do
    { f = k! && 0 ≤ k ≤ N && k < N && N-k = V}
    { f*(k+1) = (k+1)! && 0 ≤ k+1 ≤ N && N-(k+1) < V}
    k := k + 1
    { f*k = k! && 0 ≤ k ≤ N && N-k < V}
    f := f * k
    { f = k! && 0 ≤ k ≤ N && N-k < V}
end
{ f = k! && 0 ≤ k ≤ N && k ≥ N}
{ f = N! }
```



https://en.wikipedia.org/wiki/Hoare_logic

```
{ N ≥ 1 }
{ 1 = 1! && 0 ≤ 1 ≤ N }
k := 1
{ 1 = k! && 0 ≤ k ≤ N }
f := 1
{ f = k! && 0 ≤ k ≤ N }
while (k < N) do
    { f = k! && 0 ≤ k ≤ N && k < N && N-k = V}
    { f*(k+1) = (k+1)! && 0 ≤ k+1 ≤ N && N-(k+1) < V}
    k := k + 1
    { f*k = k! && 0 ≤ k ≤ N && N-k < V}
    f := f * k
    { f = k! && 0 ≤ k ≤ N && N-k < V}
end
{ f = k! && 0 ≤ k ≤ N && k ≥ N}
{ f = N! }
```



https://en.wikipedia.org/wiki/Hoare_logic

```
const sum = require('./sum');

test('adds 1 + 2 to equal 3', () => {
  expect(sum(1, 2)).toBe(3);
});
```

```
{ N ≥ 1 }
{ 1 = 1! && 0 ≤ 1 ≤ N }
k := 1
{ 1 = k! && 0 ≤ k ≤ N }
f := 1
{ f = k! && 0 ≤ k ≤ N }
while (k < N) do
    { f = k! && 0 ≤ k ≤ N && k < N && N-k = V}
    { f*(k+1) = (k+1)! && 0 ≤ k+1 ≤ N && N-(k+1) < V}
    k := k + 1
    { f*k = k! && 0 ≤ k ≤ N && N-k < V}
    f := f * k
    { f = k! && 0 ≤ k ≤ N && N-k < V}
end
{ f = k! && 0 ≤ k ≤ N && k ≥ N}
{ f = N! }
```

Property-based testing

```
const sum = require('./sum');

test('adds 1 + 2 to equal 3', () => {
  expect(sum(1, 2)).toBe(3);
});
```

```
{ N ≥ 1 }
{ 1 = 1! && 0 ≤ 1 ≤ N }
k := 1
{ 1 = k! && 0 ≤ k ≤ N }
f := 1
{ f = k! && 0 ≤ k ≤ N }
while (k < N) do
    { f = k! && 0 ≤ k ≤ N && k < N && N-k = V}
    { f*(k+1) = (k+1)! && 0 ≤ k+1 ≤ N && N-(k+1) < V}
    k := k + 1
    { f*k = k! && 0 ≤ k ≤ N && N-k < V}
    f := f * k
    { f = k! && 0 ≤ k ≤ N && N-k < V}
end
{ f = k! && 0 ≤ k ≤ N && k ≥ N}
{ f = N! }
```

```javascript
const fc = require('fast-check');

// Code under test
const contains = (text, pattern) => text.indexOf(pattern) >= 0;

// Properties
describe('properties', () => {
  // string text always contains itself
  it('should always contain itself', () => {
    fc.assert(fc.property(fc.string(), (text) => contains(text, text)));
  });
  // string a + b + c always contains b, whatever the values of a, b and c
  it('should always contain its substrings', () => {
    fc.assert(
      fc.property(fc.string(), fc.string(), fc.string(), (a, b, c) => {
        // Alternatively: no return statement and direct usage of expect or assert
        return contains(a + b + c, b);
      })
    );
  });
});
```

```javascript
const fc = require('fast-check');

// Code under test
const contains = (text, pattern) => text.indexOf(pattern) >= 0;

// Properties
describe('properties', () => {
  // string text always contains itself
  it('should always contain itself', () => {
    fc.assert(fc.property(fc.string(), (text) => contains(text, text)));
  });
  // string a + b + c always contains b, whatever the values of a, b and c
  it('should always contain its substrings', () => {
    fc.assert(
      fc.property(fc.string(), fc.string(), fc.string(), (a, b, c) => {
        // Alternatively: no return statement and direct usage of expect or assert
        return contains(a + b + c, b);
      })
    );
  });
});
```

```
λ Ok, 100 tests passed!
```

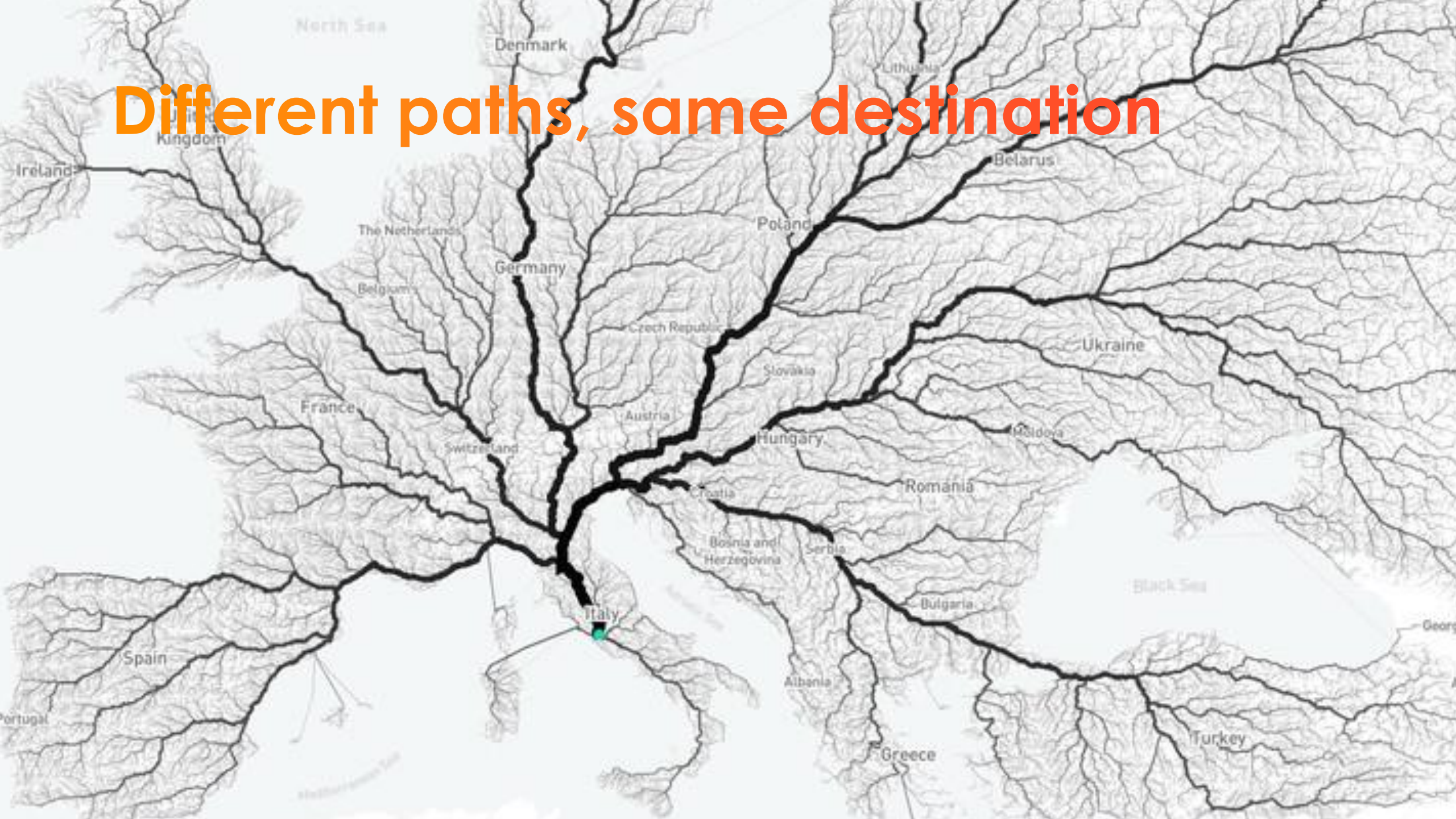
Writing good properties

- You can throw anything at it
- Different paths, same destination
- There and back again
- Some things never change
- The more things change, the more they stay the same

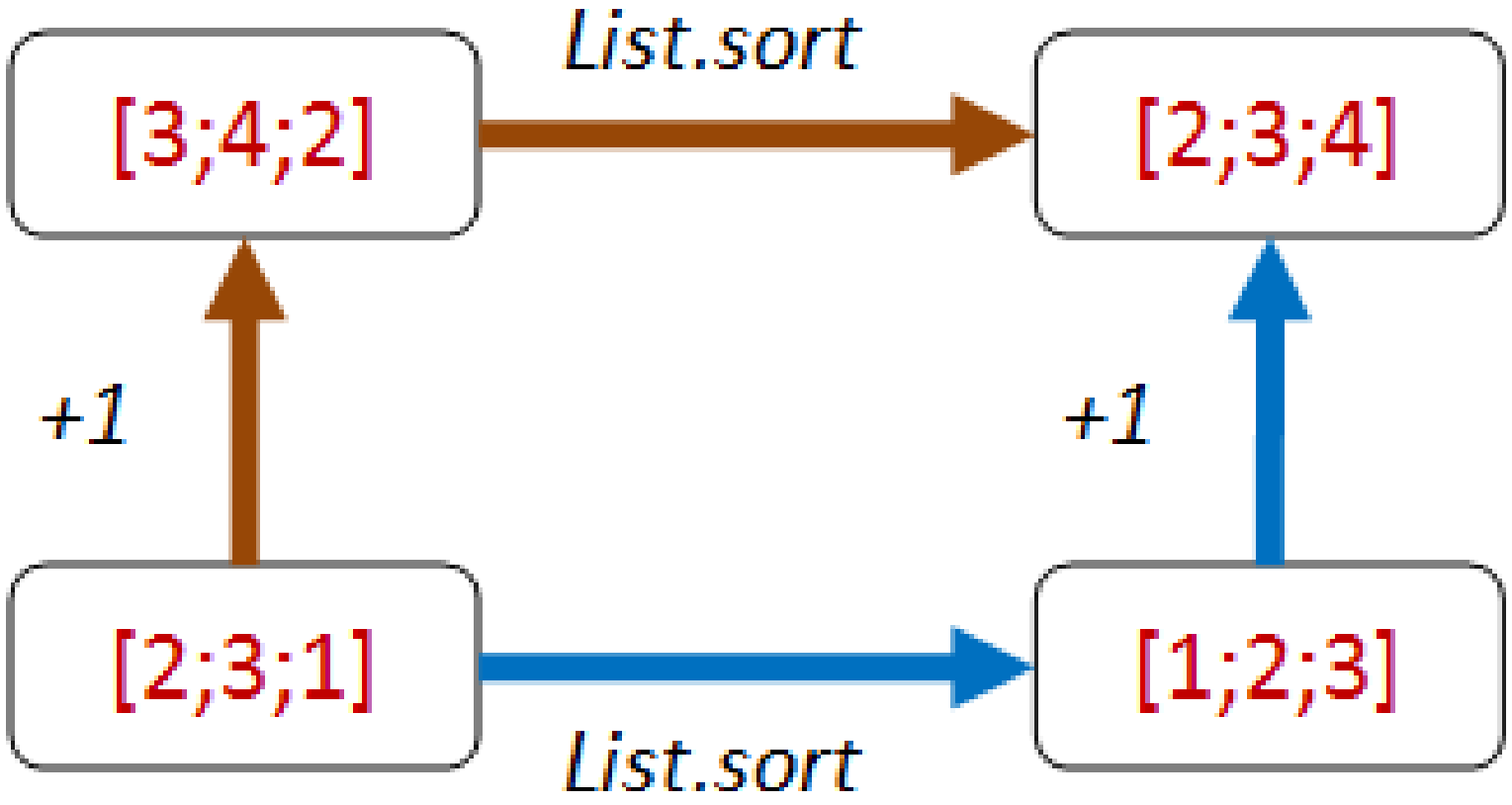
https://github.com/jlink/how-to-specify-it#42-postconditions
https://fsharpforfunandprofit.com/pbt/


You can throw anything at it

Different paths, same destination



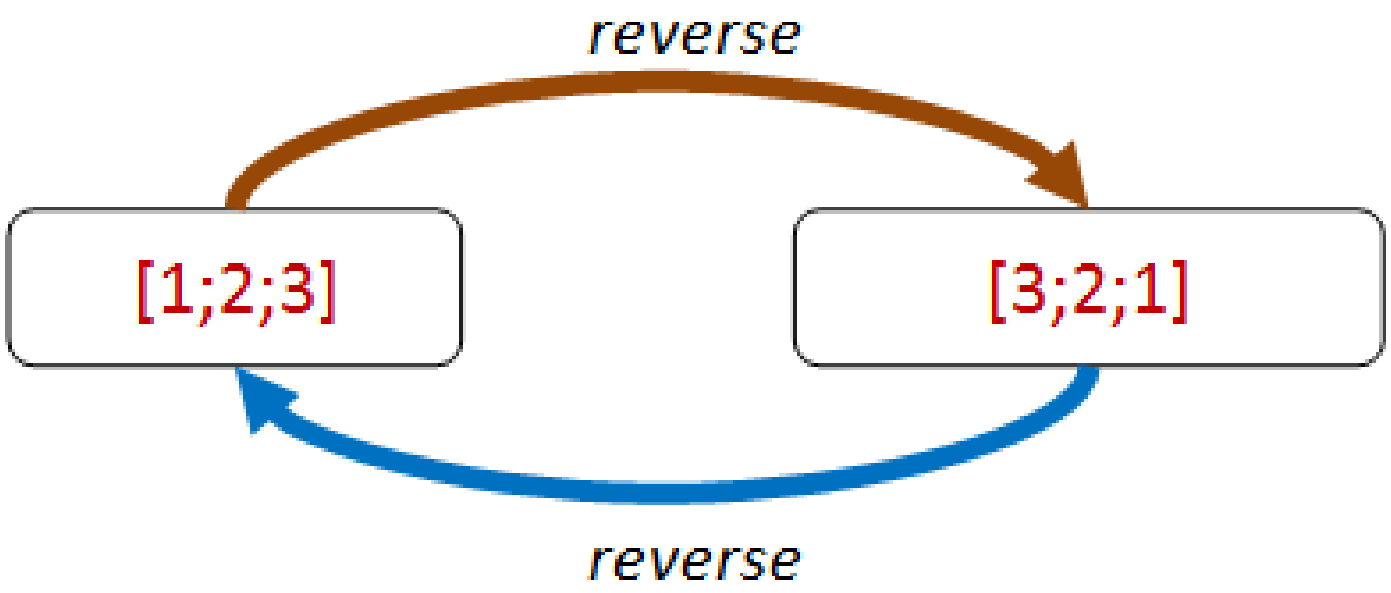
Different paths, same destination



https://fsharpforfunandprofit.com/pbt/

There and back again

There and back again



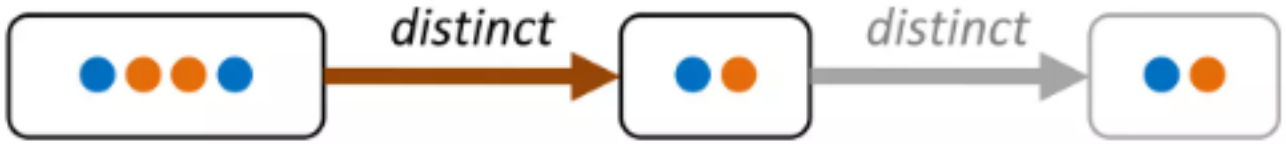
https://fsharpforfunandprofit.com/pbt/

Some things never change

Examples:
- Size of a collection
- Contents of a collection
- Balanced trees

https://fsharpforfunandprofit.com/pbt/

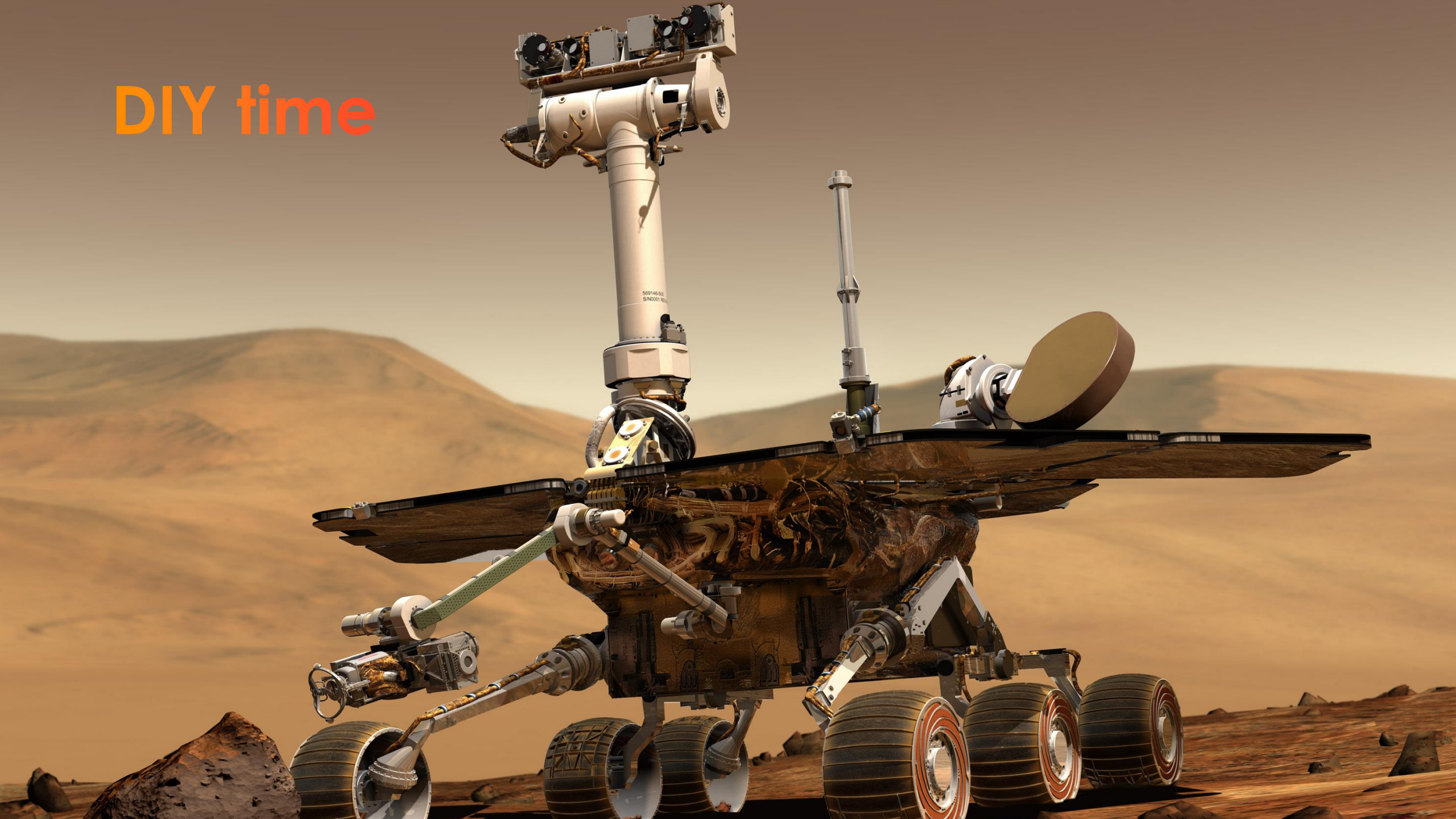
The more things change…



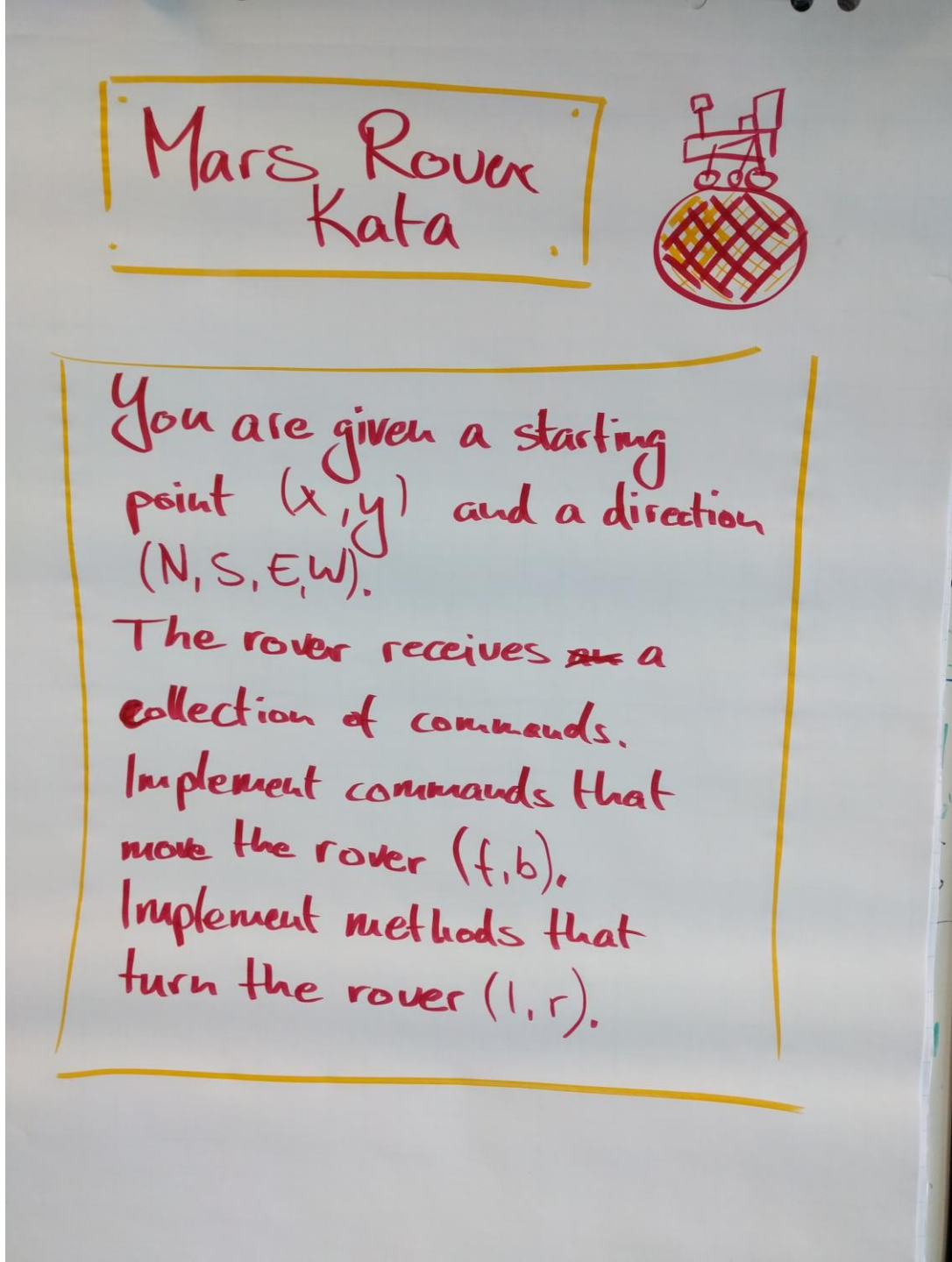
Idempotence:
- Sort
- Filter
- Event processing
- Required for distributed designs

https://fsharpforfunandprofit.com/pbt/

DIY time



Mars rover kata



https://kata-log.rocks/mars-rover-kata


DIY time
https://bit.ly/pbt-workshop

Conclusion

"See how much thought goes into a property-based test? and this is a simple specification! I don't recommend writing these for all your code – **only the really important stuff**."

"Property-based tests are best **combined** with example-based tests. Examples help you start organizing your thoughts, and they're easier for future-you to read and understand when you come back to this code later. **Humans think in examples. Programs don't extrapolate.** Property-based thinking and property-based testing can bridge between us and the computer. Math, it's a tool."

https://jessitron.com/2013/04/25/property-based-testing-what-is-it/