Purifying TypeScript

Hola!

I’m Tim Clifford

Engineering Lead at BCGDV
🇦🇺 living in 🐻

@timothyclifford

Mission statement

Understand how to apply functional programming techniques with TypeScript.

Agenda

1. Functional programming overview
2. TypeScript overview
3. Functional programming with TypeScript
4. Summary
5. Where to next

1. Functional programming

OO pattern/principle	**FP pattern/principle**
• Single Responsibility Principle	• Functions
• Open/Closed principle	• Functions
• Dependency Inversion Principle	• Functions, also
• Interface Segregation Principle	• Functions
• Factory pattern	• Yes, functions
• Strategy pattern	• Oh my, functions again!
• Decorator pattern	• Functions
• Visitor pattern	• Functions ▯

Functional what?

Programming

Building software by composing pure functions, avoiding shared state, mutable data, & side-effects.

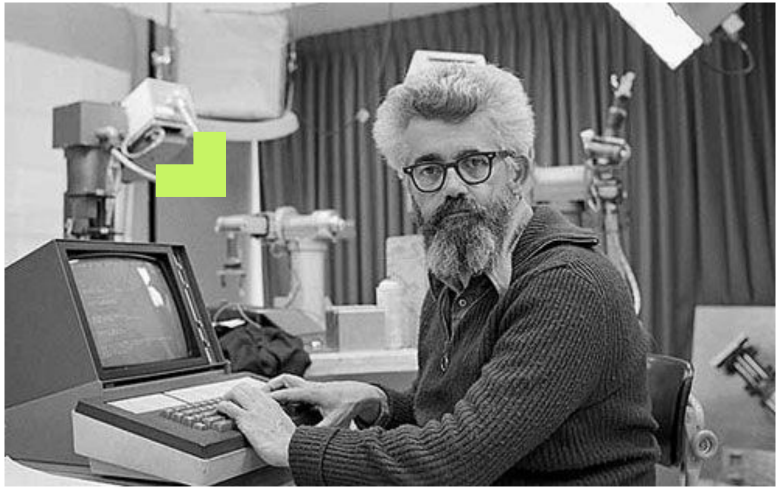
Languages

Enable or enforce the principles of functional programming.

The stricter language, the more functional.

Origins of functional programming

λ



* made up numbers

89,526,124

Blog posts advocating functional programming

185,244

StackOverflow questions about monads

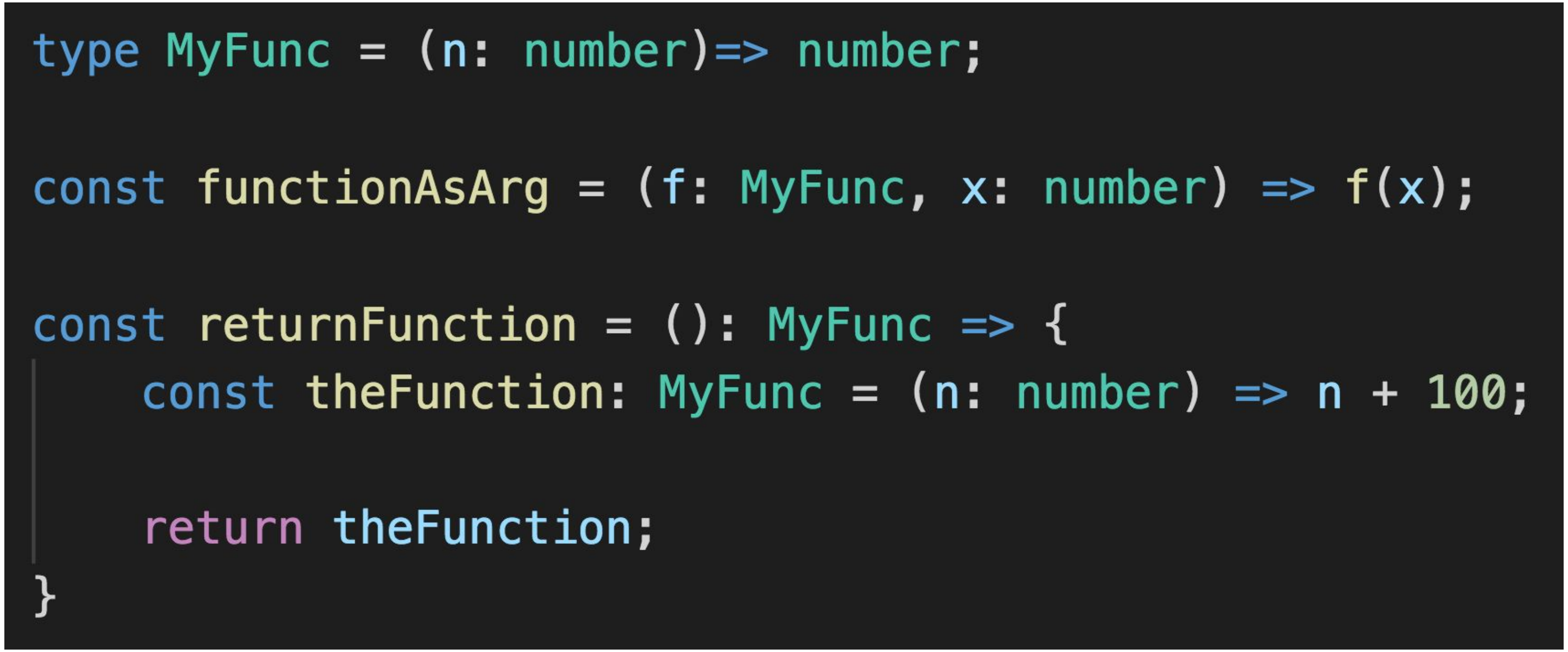
300%

Increase in OO vs FP arguments

Functional programming concepts

(some...)

First class functions



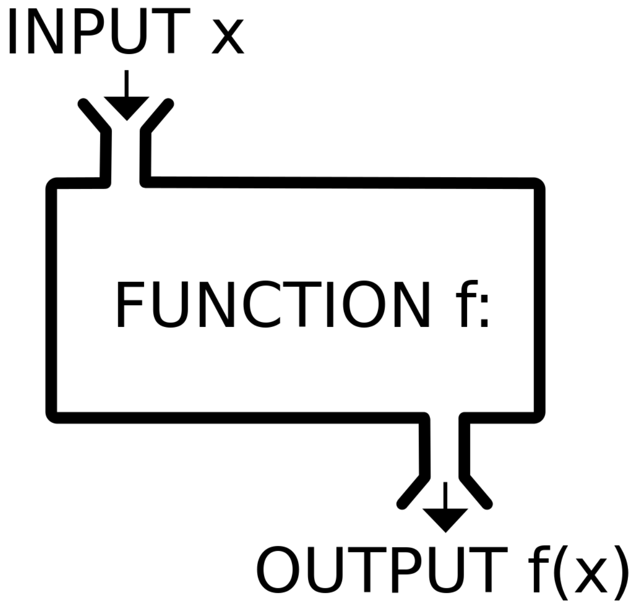
```
type MyFunc = (n: number)=> number;

const functionAsArg = (f: MyFunc, x: number) => f(x);

const returnFunction = (): MyFunc => {
    const theFunction: MyFunc = (n: number) => n + 100;

    return theFunction;
}
```

Pure functions

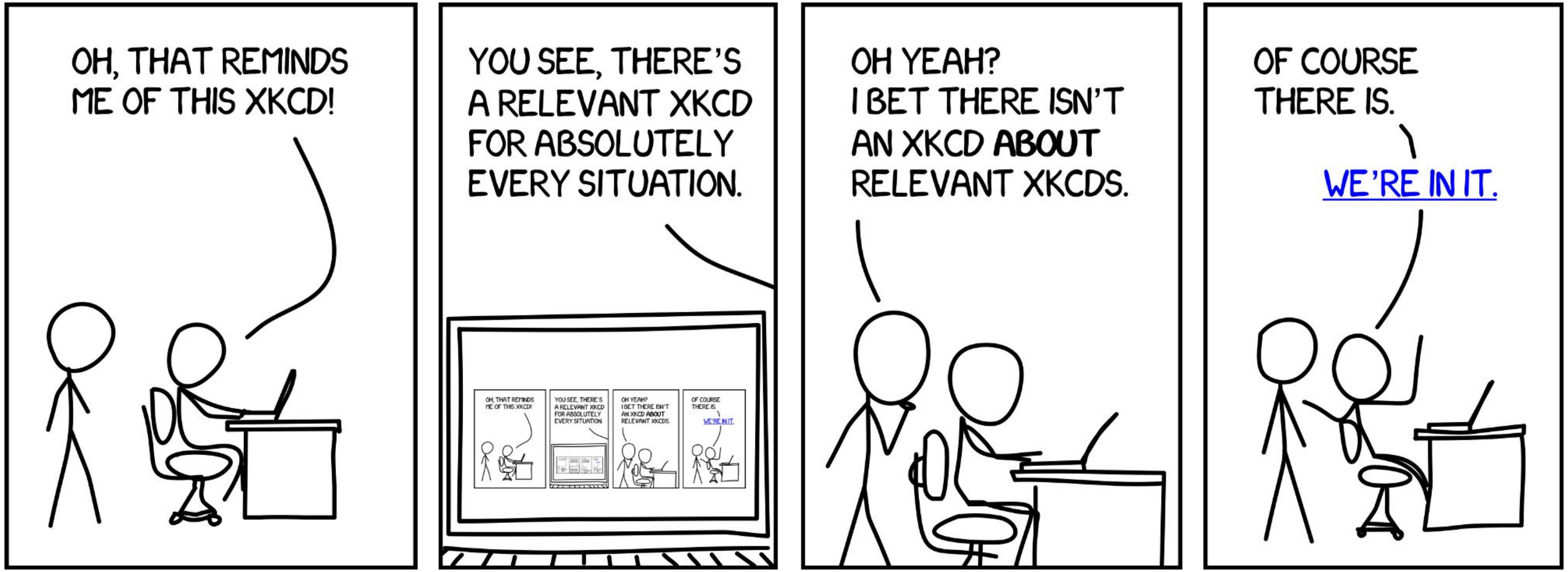


CODE WRITTEN IN HASKELL IS GUARANTEED TO HAVE NO SIDE EFFECTS.
...BECAUSE NO ONE WILL EVER RUN IT?

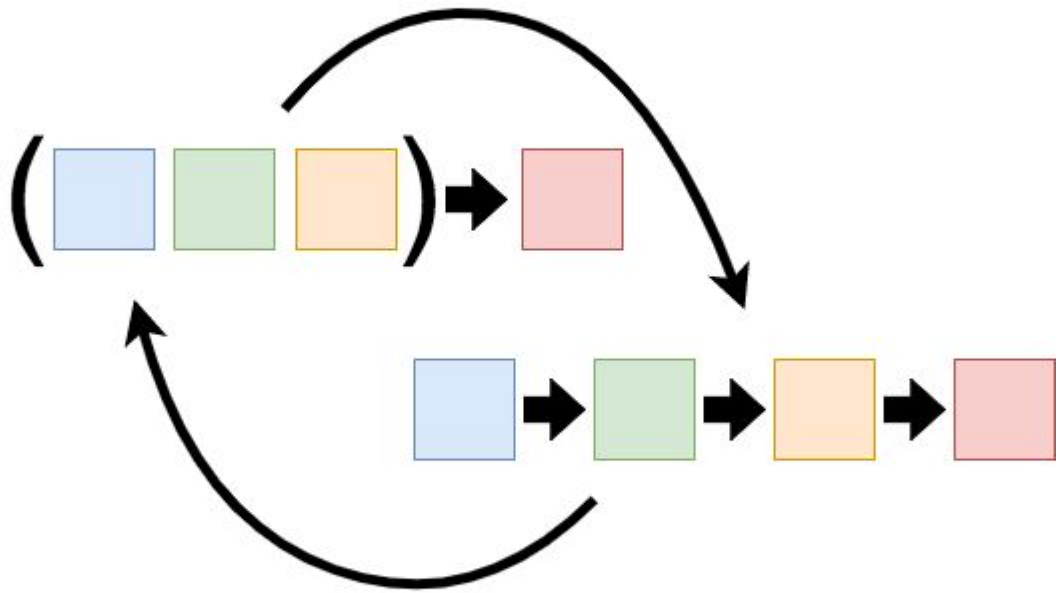
Immutability

Once you go **const**, there’s no going back…

Recursion


OH, THAT REMINDS ME OF THIS XKCD!
YOU SEE, THERE'S A RELEVANT XKCD FOR ABSOLUTELY EVERY SITUATION.
OH YEAH? I BET THERE ISN'T AN XKCD ABOUT RELEVANT XKCDS.
OF COURSE THERE IS.
WE'RE IN IT.

Currying

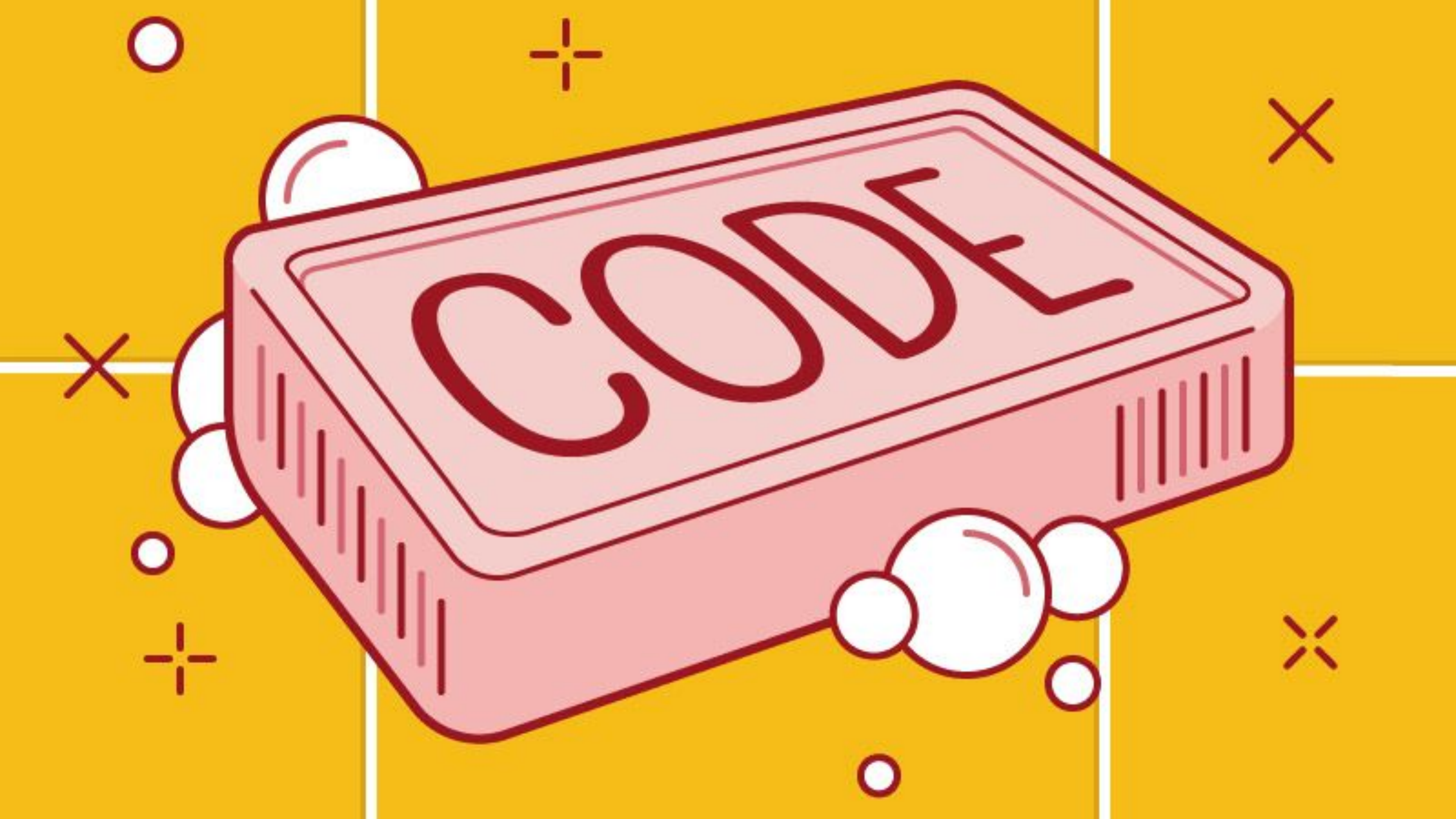


Pattern matching

```
value match {
    case 1 | 2 => doSomething()
    case 3 => doSomethingElse()
    case _ = > doSomethingDefault()
}
```

```
switch (value) {
    case 1:
    case 2:
        doSomething()
        break
    case 3:
        doSomethingElse()
        break
    default:
        doSomethingDefault()
}
```

Benefits of programming functionally


CODE

Test
Shift
?
Er

2.

TypeScript

A typed superset of JavaScript which compiles to JavaScript

TypeScript - the good parts

- Compile time error checking
- Safer refactoring
- Documentation through types
- Types are optional - type inference FTW
- JavaScript superset

https://www.destroyallsoftware.com/talks/wat

The thing you realize about typing is that it's a dial. The higher you place the dial, the more painful the programmer's life becomes, but the safer it becomes too. But you can turn that dial too far in either direction.

- **Anders Hejlsberg**

3.

Functional TypeScript

Show me the code!

4.

Summary

Terminology

Monads, monoids, functors, category theory?

Types

Safety is nice but it’s not free.

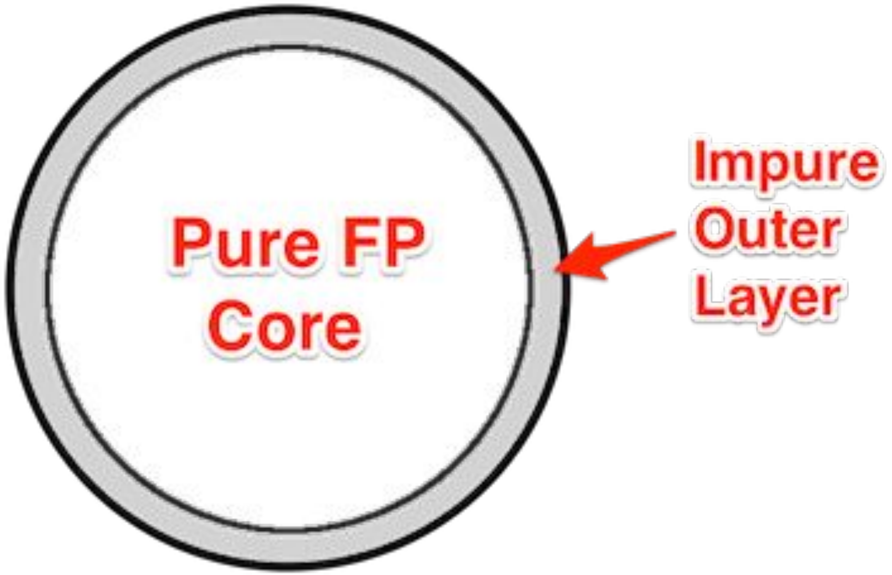
Pure functions

Easy to write, hard to combine.

Side effects

Staying pure is difficult in a mutating world.

Isolate side-effects



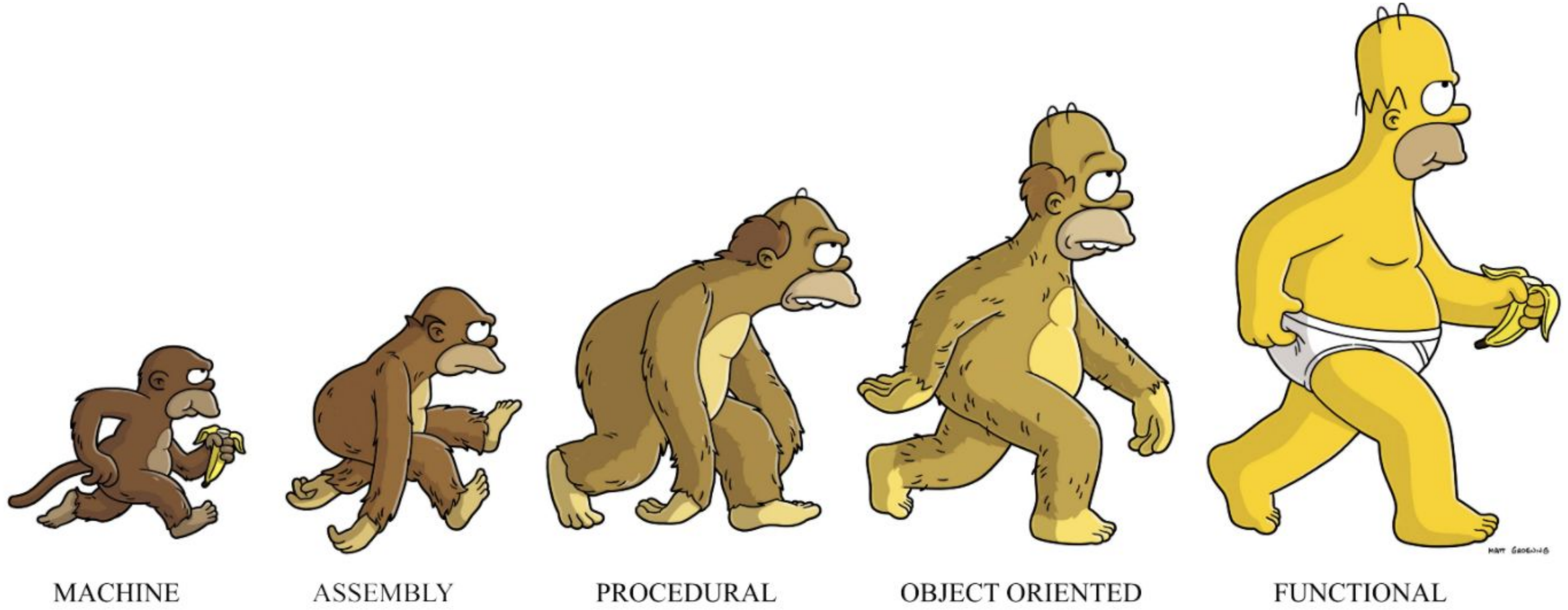
Higher kinded types

Where is my generic generic?

5.

Next steps

Functional evolution



TypeScript libraries

Functional programming libraries

- https://github.com/gcanti/fp-ts
- https://true-myth.js.org/

Immutable data structures

- https://github.com/pelotom/type-zoo
- https://github.com/emmanueltouzery/prelude.ts

Algebraic data types

- https://gigobyte.github.io/pure/

JavaScript libraries with typings

Functional programming libraries

- https://ramdajs.com/
- https://sanctuary.js.org/
- https://folktale.origamitower.com/

Immutable data structures

- https://facebook.github.io/immutable-js/

Futures in JavaScript

- https://github.com/fluture-js/Fluture

Functional > JavaScript

- Elm
- Purescript
- Reason
- Idris
- ScalaJS
- Clojurescript
- Fable
- GHCJS

Gracias!

Slides & code
https://goo.gl/hv54JG