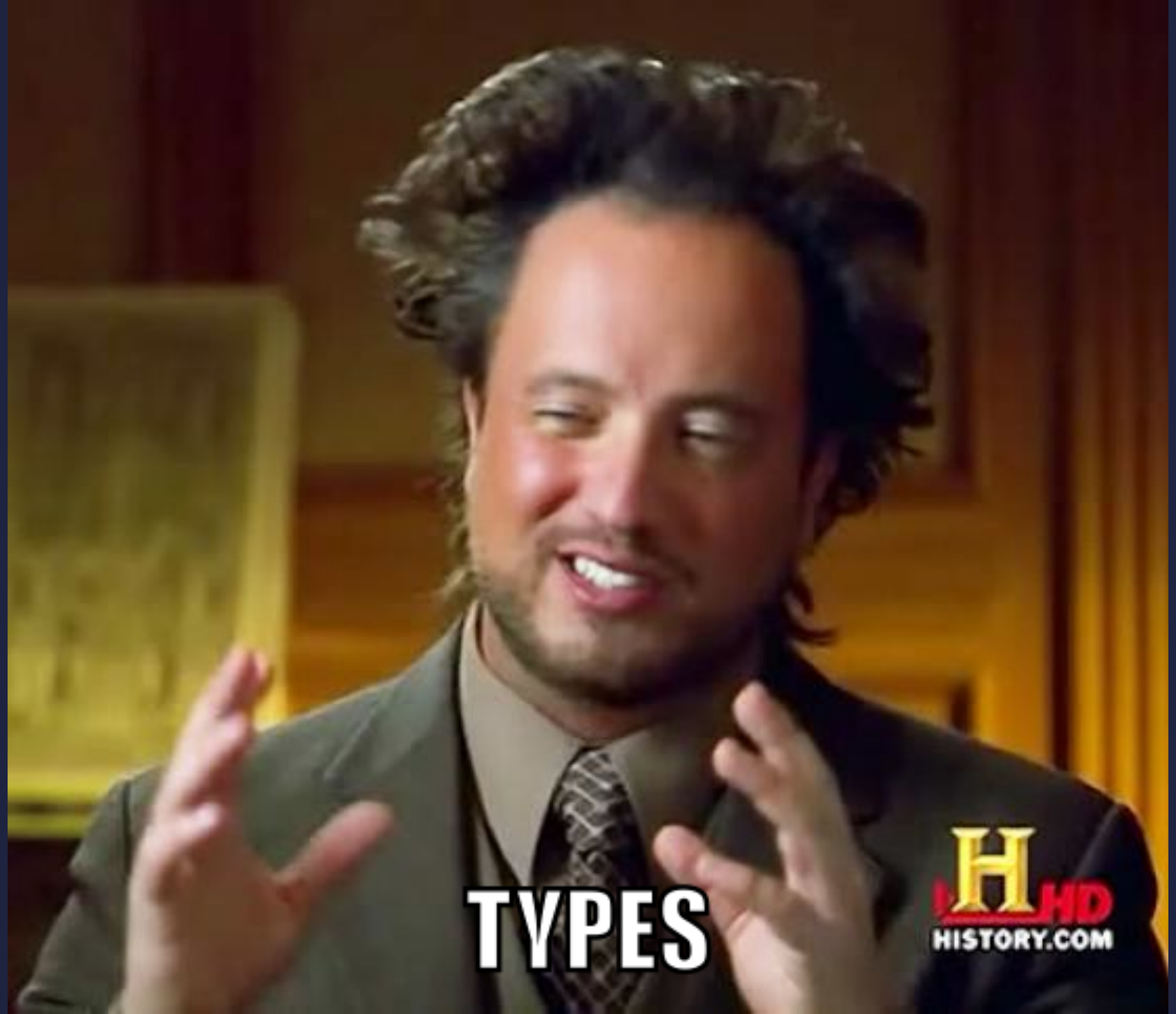
CORRECT BEHAVIOR THROUGH TYPE SAFETY

@JSPAHRSUMMERS

REACTIVECOCOA

MANTLE

CARTHAGE


TYPES
HD
HISTORY.COM

TYPES PREVENT ERRORS

(by restricting what you can do)

NSArray

IS MORE RESTRICTED THAN

NSMutableArray

NSArray

PREVENTS MORE ERRORS THAN

NSMutableArray

`Optional<T>`

RESTRICTS ACCESS TO ITS VALUE

PREVENTS MISTAKES WITH `nil`

Array<T>

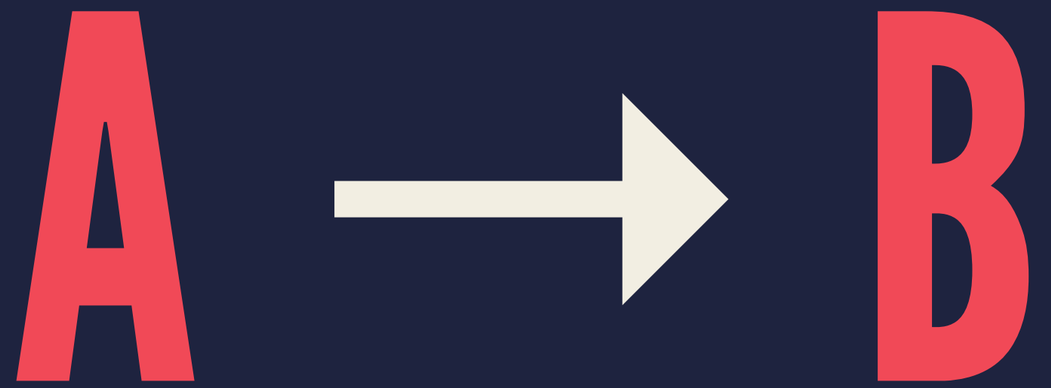
RESTRICTS INSERTIONS

PREVENTS CRASHES AFTER RETRIEVAL

More generally...

TYPES = PROOFS

(the Curry–Howard correspondence)


A
B

A → B

```
func toInteger(num: Double) -> Int {
    let result = round(num)
    return Int(result)
}
```

STRING → STRING

STRING → STRING

```
func identity(s: String) -> String {
    return s
}
```

WE CAN USE

TYPES

TO PROVE

CORRECTNESS

WHAT DOES NSDATA PROVE?

(Not much.)

WHAT DOES NSSTRING PROVE?

Characters, not just bytes!

WHAT DOES NSURL PROVE?

Valid URL, not just a string!

EXTENDS TO
ANY KIND
OF VALIDATION

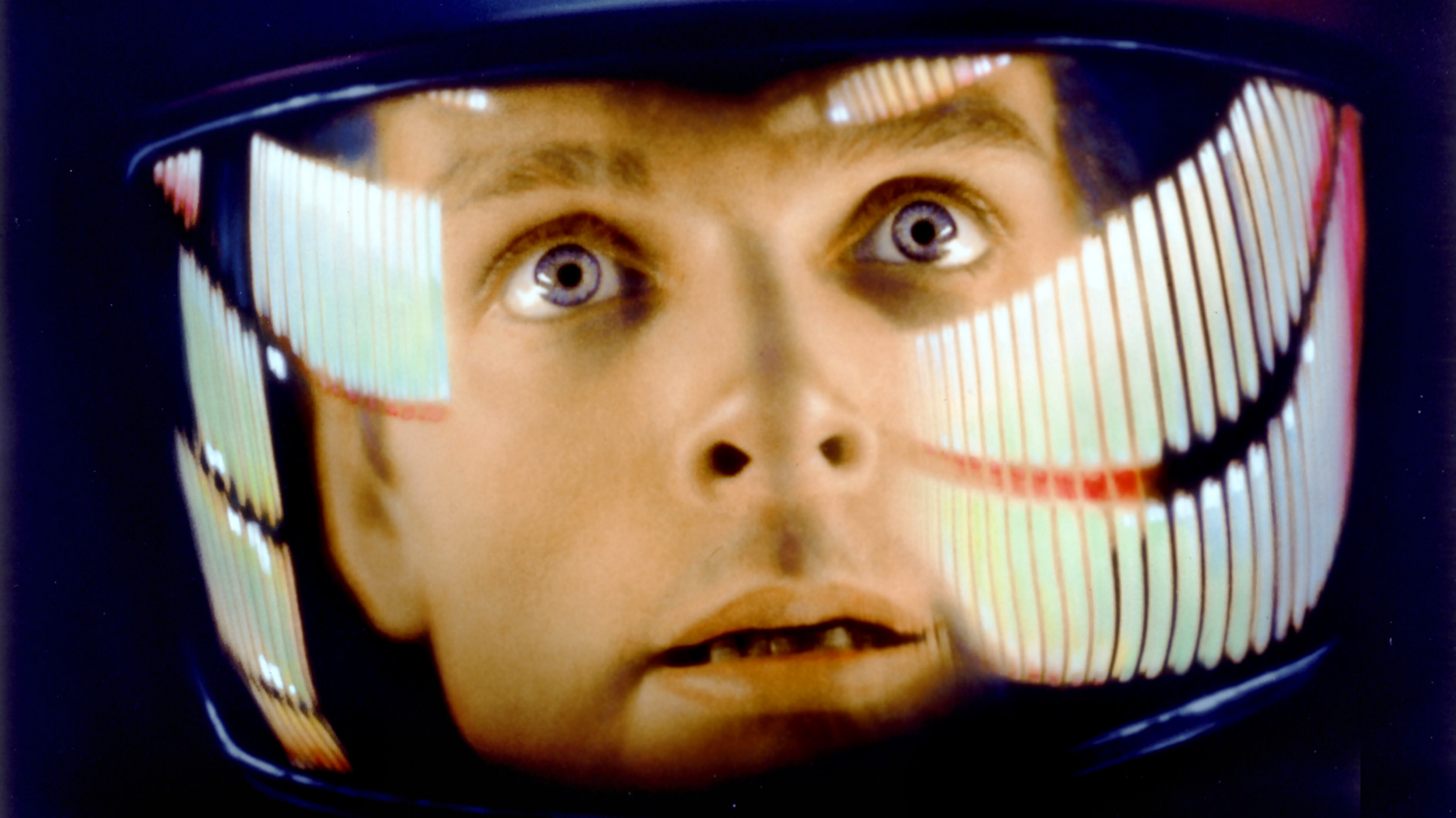
What about tests?


Science break!

TESTS ARE SUPPORT
TYPES ARE PROOF

Caveats

- Proofs can contain errors too
- Types can’t represent everything
- Tests may be good enough



PHANTOM TYPES[1]

```
struct FileHandle<A> {
    let handle: NSFileHandle
}

enum Read {}
enum Write {}

func openForReading(path: String) -> FileHandle<Read>
func openForWriting(path: String) -> FileHandle<Write>
func readDataToEndOfFile(handle: FileHandle<Read>) -> NSData
```

[1] See functional snippet #13 from objc.io.

Case study:

ERRORS IN REACTIVECOCOA

SIGNALS CONSIST OF...

- Any number of next events
- Optionally an error or completed event

```
Next* (Error | Completed)?
```

RACSignal *


WHAT'S IN THE BOX?!?!
quickmeme.com

PROPERTY BINDING IN RAC 2

```
RAC(self.imageView, image) = [RACObserve(self, model)
    flattenMap:^ RACSignal * (Model *model) {
        return [model fetchImage];
    }];
```

PROPERTY BINDING IN RAC 2

```
RAC(self.imageView, image) = [RACObserve(self, model)
    flattenMap:^ RACSignal * (Model *model) {
        return [model fetchImage];
    }];
```

```
*** Received error from RACSignal in binding for key
path "image" on UIImageView:
    Error Domain=NSURLErrorDomain Code=-1 "Operation
could not be completed."
```

Solution:

TYPES!

Signal<T, Error>

```
enum NoError {}
```

PROPERTY BINDING IN RAC 3

```
func <~ <T> (property: MutableProperty<T>,
    signal: Signal<T, NoError>)

func <~ <T> (property: MutableProperty<T>,
    producer: SignalProducer<T, NoError>)
```

PROPERTY BINDING IN RAC 3

```
class Model {
    func fetchImage() -> SignalProducer<UIImage, NSError>
}

self.imageProperty <~ self.modelProperty.producer
    // TYPE ERROR: NSError is incompatible with NoError
    |> flatMap(.Latest) { model in
        return model.fetchImage()
    }
```

PROPERTY BINDING IN RAC 3

```
class Model {
    func fetchImage() -> SignalProducer<UIImage, NSError>
}

self.imageProperty <~ self.modelProperty.producer
    |> flatMap(.Latest) { model in
        return model.fetchImage()
            // Ignore any error event
            |> catch { error in .empty }
    }
```

TYPES CAN ALSO DESCRIBE EFFECTS

IO

~~MONAD~~

IN HASKELL

WITH IO, YOU CAN...

- Put a value into it
- Perform side effects using the value
- "Lift" pure functions to apply to the value
- Never[2] get the result back out

[2] Except through the rarely-used `unsafePerformIO`.

WITH A SIGNAL, YOU CAN...

- Put values into it
- Perform time-based operations using the values
- "Lift" pure functions to apply to the values
- Register for delivery[3] to get the results back out

[3] It's possible to synchronously wait for results, but the framework highly discourages this.

Example:

TYPES FOR CONCURRENCY

THE UI PROBLEM™

```
dispatch_async(someBackgroundQueue) {
    // Oops!
    self.textField.text = "foobar"
}
```

CAPTURING UI CODE IN THE TYPE SYSTEM

```
struct UIAction<T> {
    init(_ action: () -> T)

    func enqueue()
    func enqueue(handler: T -> ())

    func map<U>(f: T -> U) -> UIAction<U>
    func flatMap<U>(f: T -> UIAction<U>) -> UIAction<U>
}
```

WHAT IF...?

```
extension UITextField {
    /// Sets the receiver's text to the given string
    /// when the returned action is executed.
    func jss_setText(text: String) -> UIAction<()>
}
```

SLIDES AND NOTES

Available at git.io/vkjl9
(github.com/jspahrsummers/correct-behavior-through-type-safety)

THANKS TO...

Neil Pankey, Rob Rix, James Lawton, Gordon Fontenot, Eli Perkins, Javi Soto, anyone else who I omitted in error, and you!