

# A Type is Worth a Thousand Tests

Manuel M T Chakravarty  
*Applicative &  
Tweag I/O*

-  mchakravarty
-  TacticalGrace
-  justtesting.org
-  haskellformac.com

Let's talk about...

Let's talk about...



Swift

Language  
Design

Let's talk about...



Swift

Let's talk about...

Language  
Design



Ecosystem

Swift

Let's talk about...

# Why is it extraordinary



Language      Design      Ecosystem

Swift?

**Unique  
Event**

# **Java**

# **C#**

# **Objective-C**

# **Java**

Scala  
Kotlin

# **C#**

# **Objective-C**

# **Java**

Scala  
Kotlin

# **C#**

F#

# **Objective-C**

# **Java**

Scala  
Kotlin

# **C#**

F#

Objective-C

# **Swift**

# Why do languages become popular?

Why do languages become popular?

**Platforms**

# Why do languages become popular?

JavaScript

**Web**

C/C++

**Unix**

Objective-C

**Cocoa**

Java

**JVM**

**Platforms**

Python/Ruby

**Cloud**

C#

**.NET**

# Why do languages become popular?

JavaScript

**Web**

C/C++

**Unix**

Swift

**Cocoa**

Java

**JVM**

**Platforms**

Python/Ruby

**Cloud**

C#

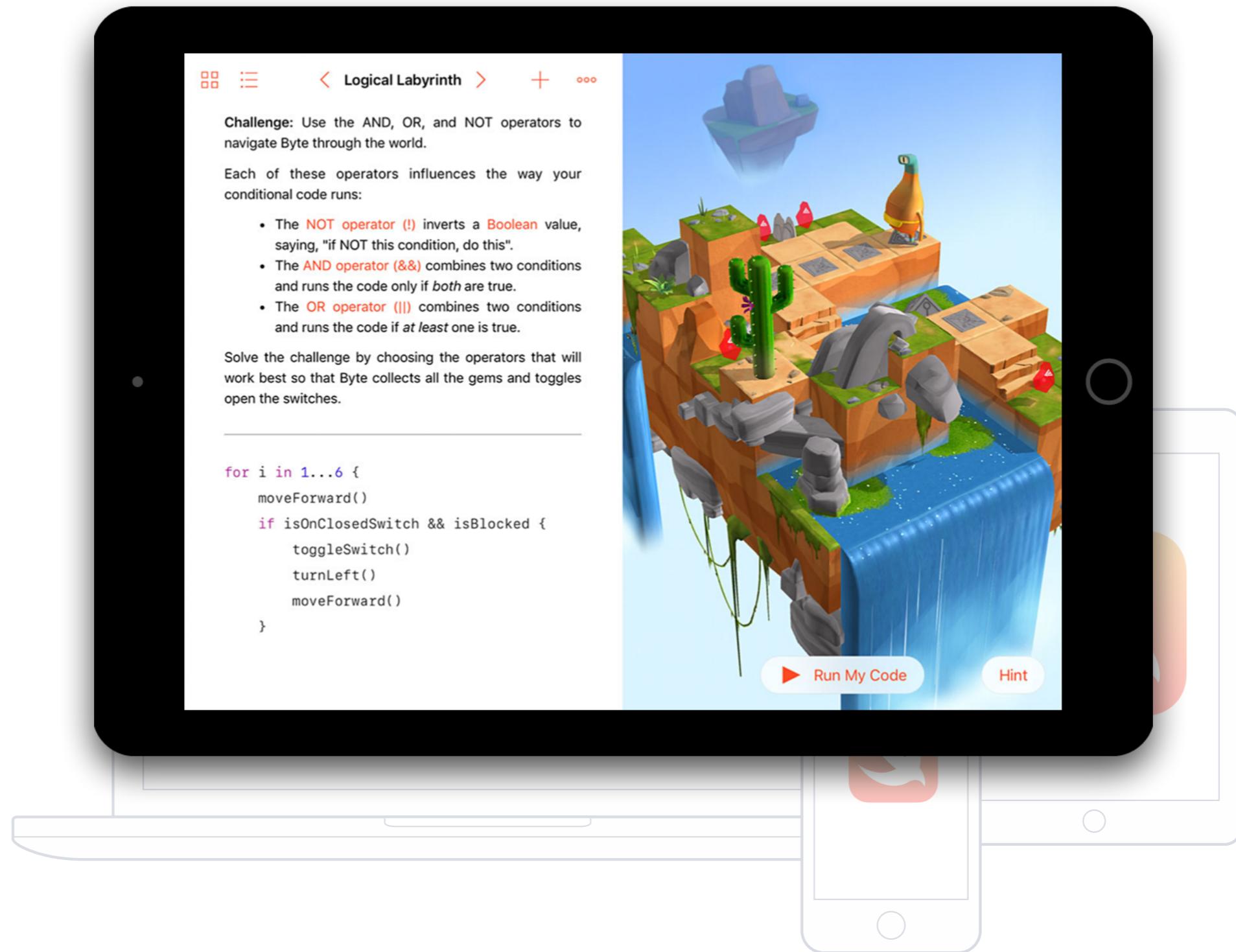
**.NET**

# Why will Swift be popular?

# Why will Swift be popular?

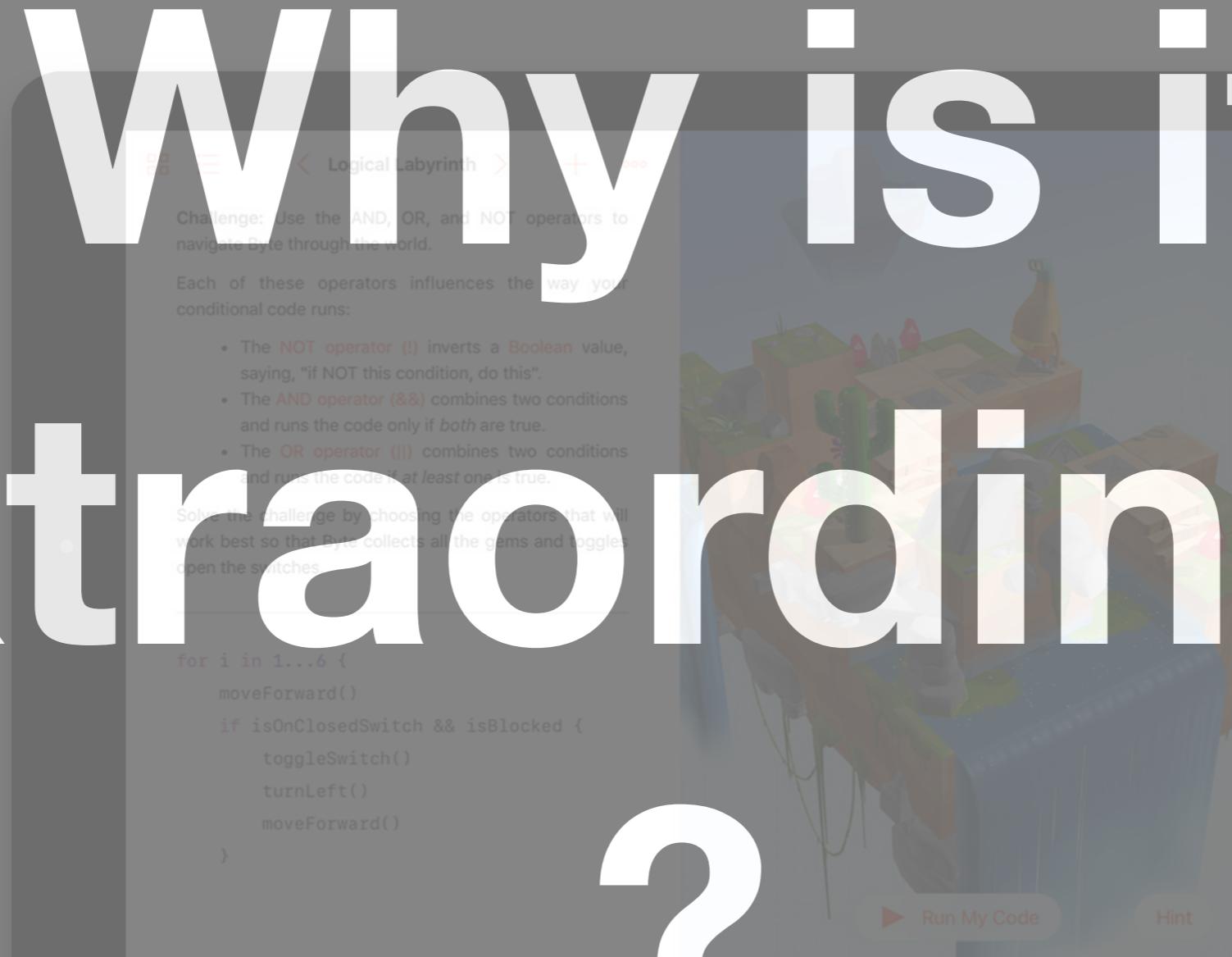


# Why will Swift be popular?



# Why will Swift be popular?

# Why is it extraordinary



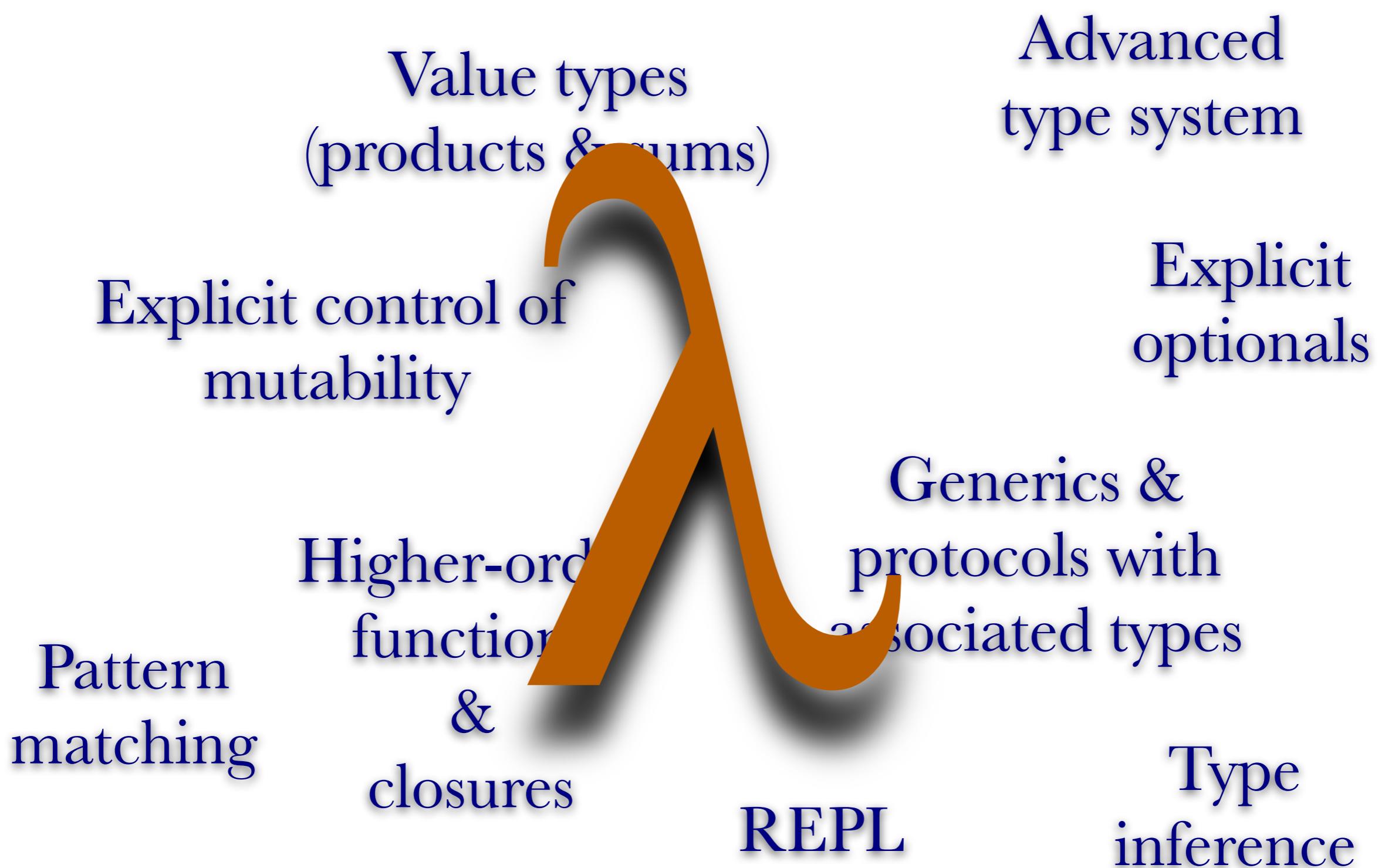
# Functional Programming

# From Objective-C to Swift

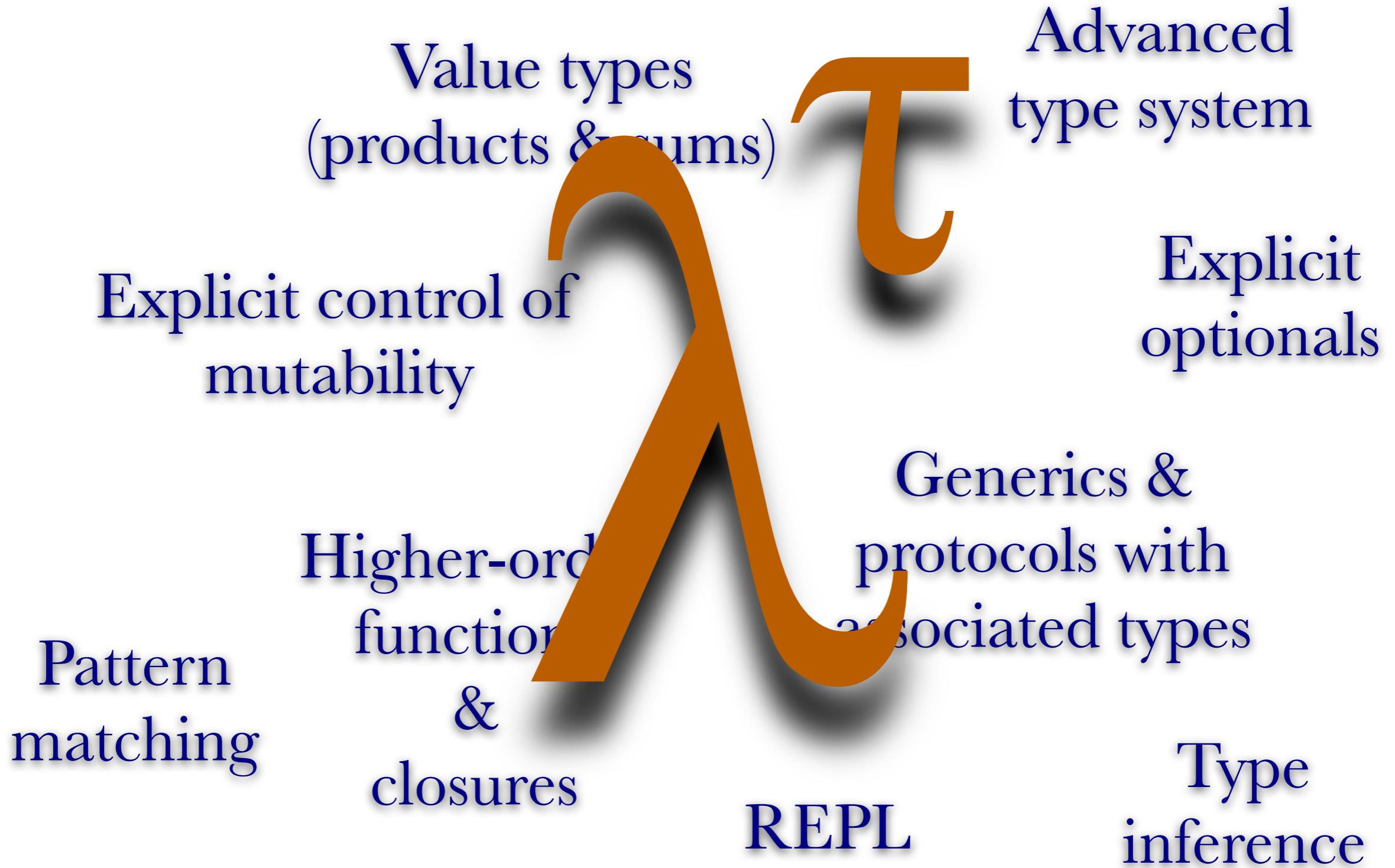
# From Objective-C to Swift

Value types (products & sums)	Advanced type system
Explicit control of mutability	Explicit optionals
Pattern matching	Generics & protocols with associated types
Higher-order functions & closures	REPL
Type inference	

# From Objective-C to Swift



# From Objective-C to Swift



“Swift encourages typed  
functional programming in a  
mainstream language.”

# Haskell for Mac

## Adopting Swift right out of the gate

# Haskell for Mac

## Adopting Swift right out of the gate

The screenshot shows the Haskell for Mac IDE interface. On the left, the project structure is displayed:

- PROJECT INFORMATION: Diagrams
- HASKELL PROGRAM C...: Diagrams (selected), Chart.hs, Diagrams.hs, ExampleStocks.hs, Rasterific.hs
- NON-HASKELL SOURC...: None
- SUPPORTING FILES: SourceS...ro\_R.svg, SourceS...o\_RB.svg

The main area shows the contents of `Chart.hs`:

```
21 -- Some of the Chart example from the package docs
22
23 -- Amplitude Modulation
24
25 signal :: [Double] -> [(Double, Double)]
26 signal xs
27   = [ (x,(sin (x*3.14159/45) + 1) / 2 * (sin
28     (x*3.14159/5)))
29   | x <- xs ]
30
31 amplitudeModulation
32   = do
33     layout_title .= "Amplitude Modulation"
34     plot (line "am" [signal [0,(0.5)..400]])
35     plot (points "am points" (signal [0,7..400]))
36
37 values :: [(String,Double,Bool)]
38 values = [ ("Mexico City",19.2,False)
39           , ("Mumbai",12.9,False)
40           , ("Sydney",4.3,False)
41           , ("London",8.3,False)
42           , ("New York",8.2,True)
43         ]
44
45 -- Relative Population
46
47 pitem (s,v,o) = pitem_value .~ v
48             $ pitem_label .~ s
49             $ pitem_offset .~ (if o then 25 else 0)
50             $ def
51
52 relativePopulation
53   = do
54     pie_title .= "Relative Population"
55     pie_plot . pie_data .= map pitem values
56
57 -- MSFT vs APPL
58
59 lineStyle n colour = line_width .~ n
60             $ line_color .~ opaque colour
61             $ def
62
```

To the right, a generated chart titled "Stock Prices" is shown. The chart displays two sets of data over time (July to September 2009). The legend indicates:

- AAPL spread (green line)
- AAPL closing (green line)
- AAPL candle (blue line with dots)
- MSFT spread (purple line)
- MSFT closing (purple line)
- MSFT candle (red line with dots)

The chart shows significant price volatility for both companies during the period.

“Swift encourages typed functional programming in a mainstream language.”

# Why Type Systems?

# Catch bugs & prevent crashes

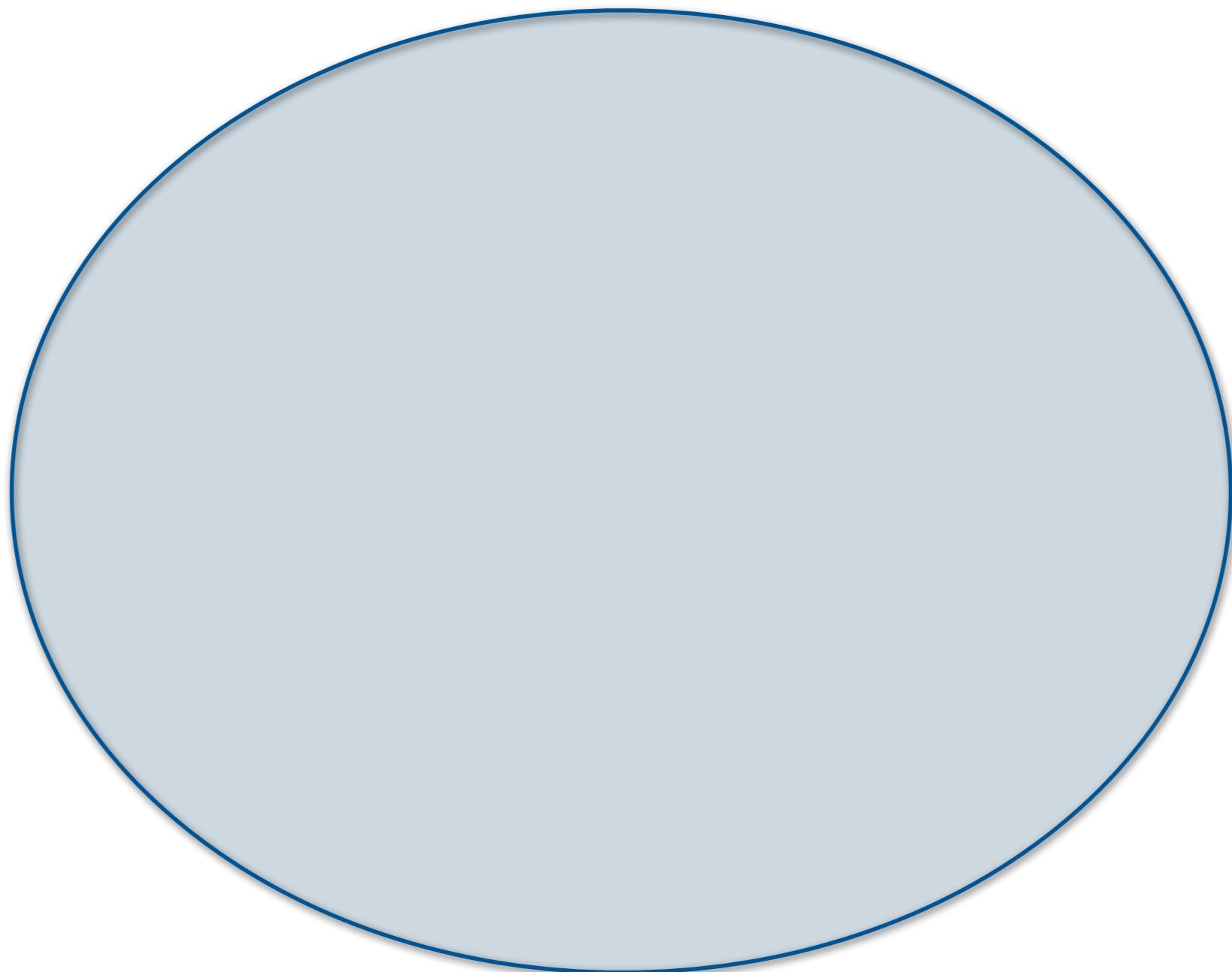


# Less testing!

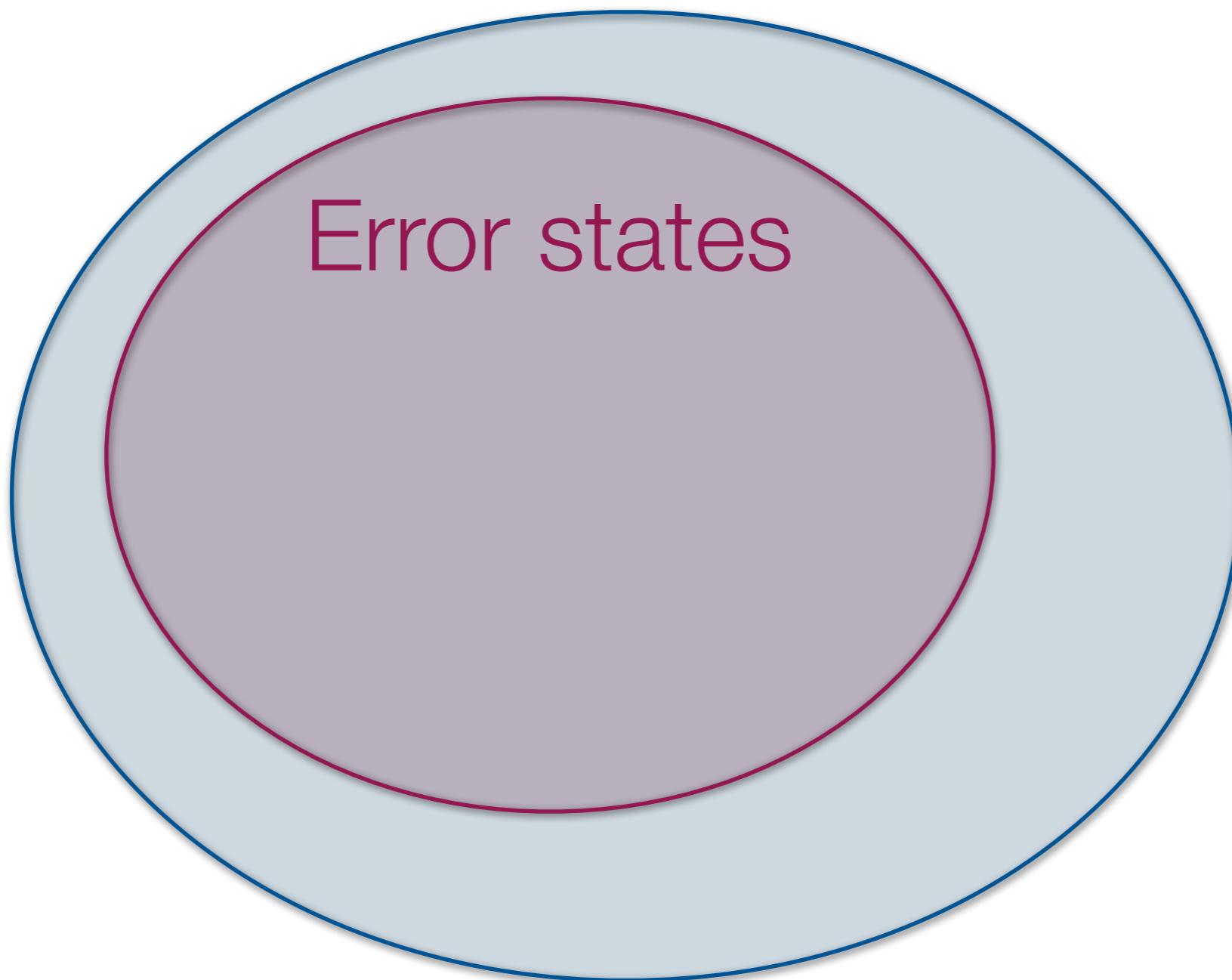


“What if you could gain the  
same confidence in correctness  
with fewer or simpler tests?”

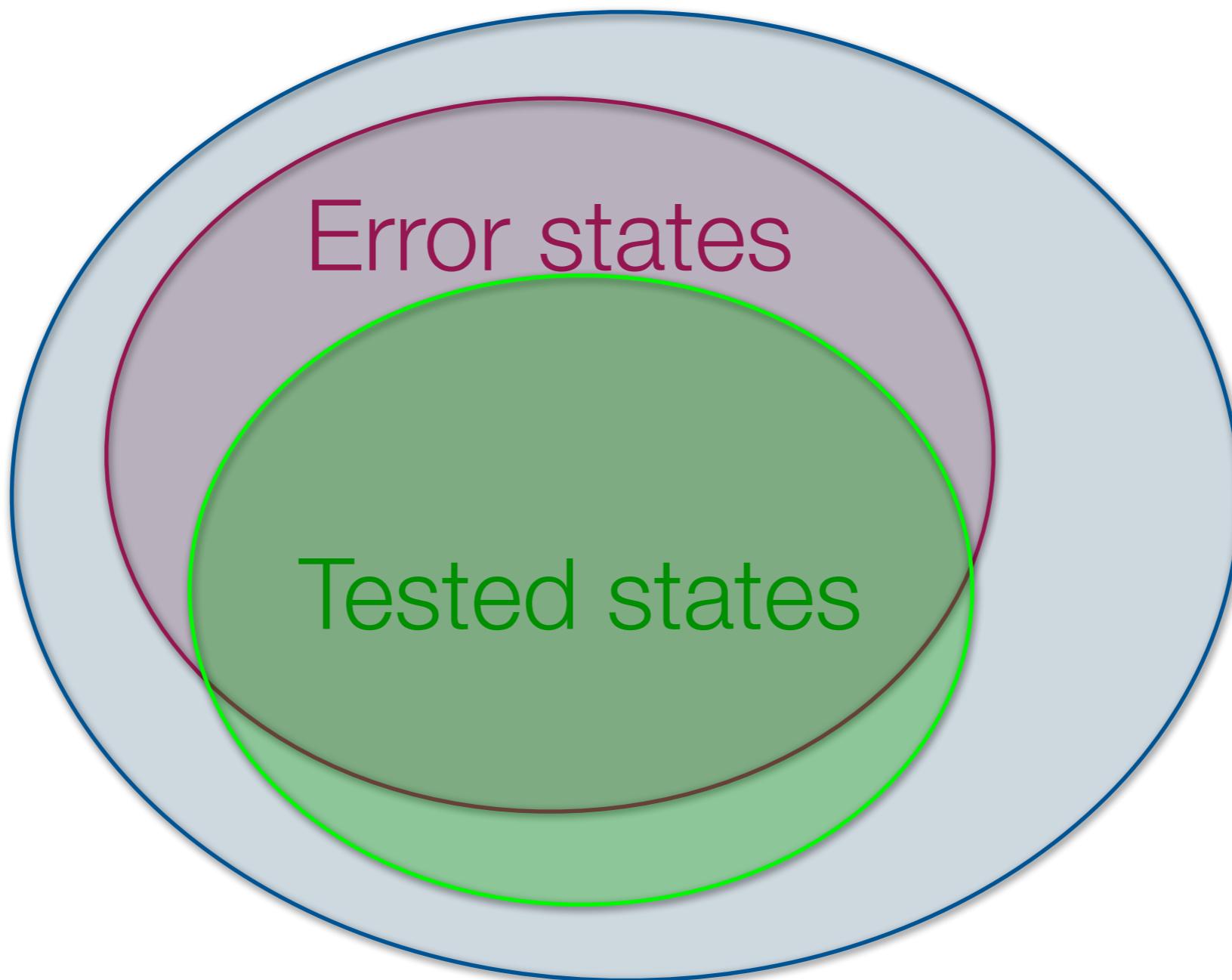
# Possible states of your application



# Possible states of your application

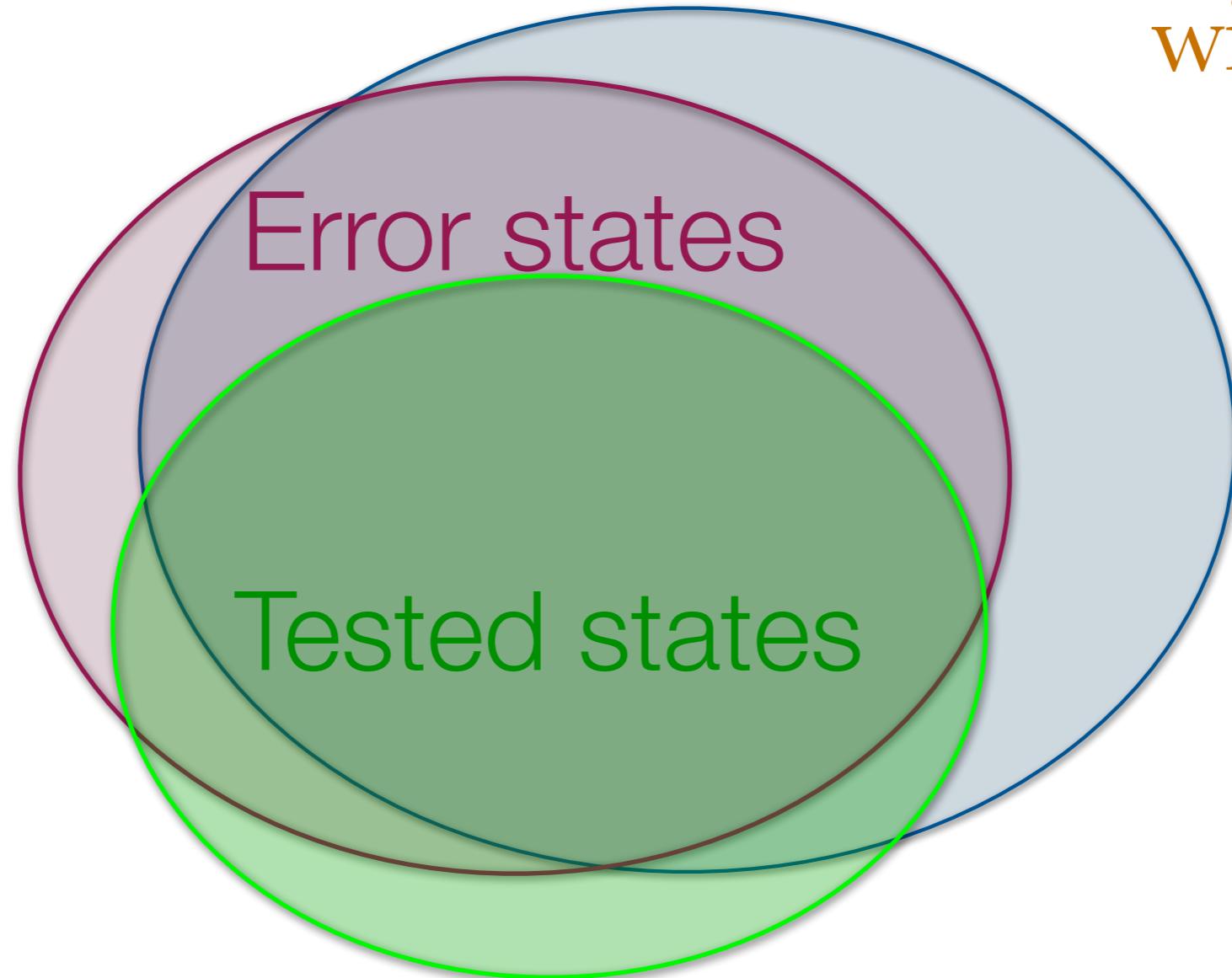


# Possible states of your application



# Possible states of your application

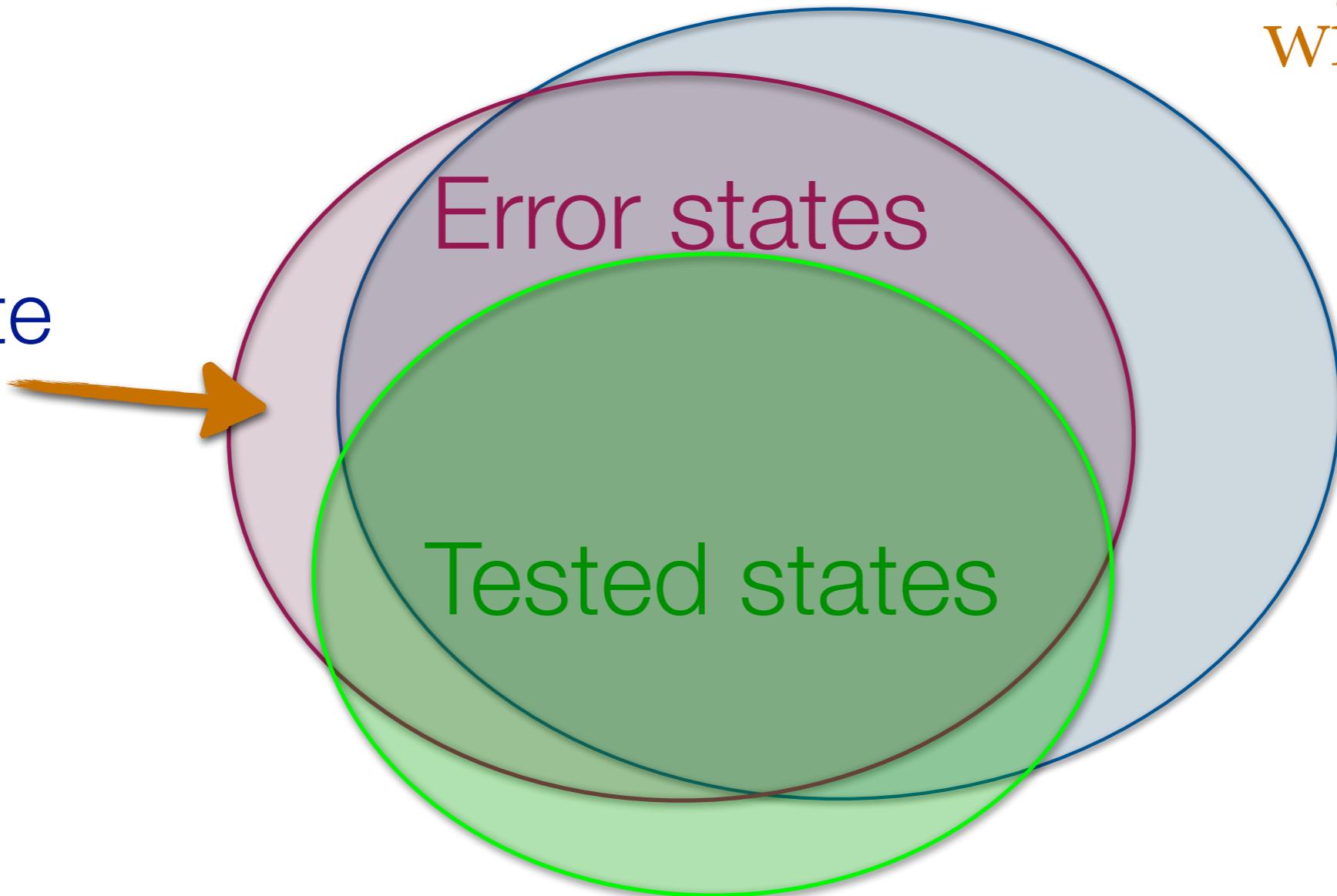
with types



# Possible states of your application

with types

eliminate  
bugs

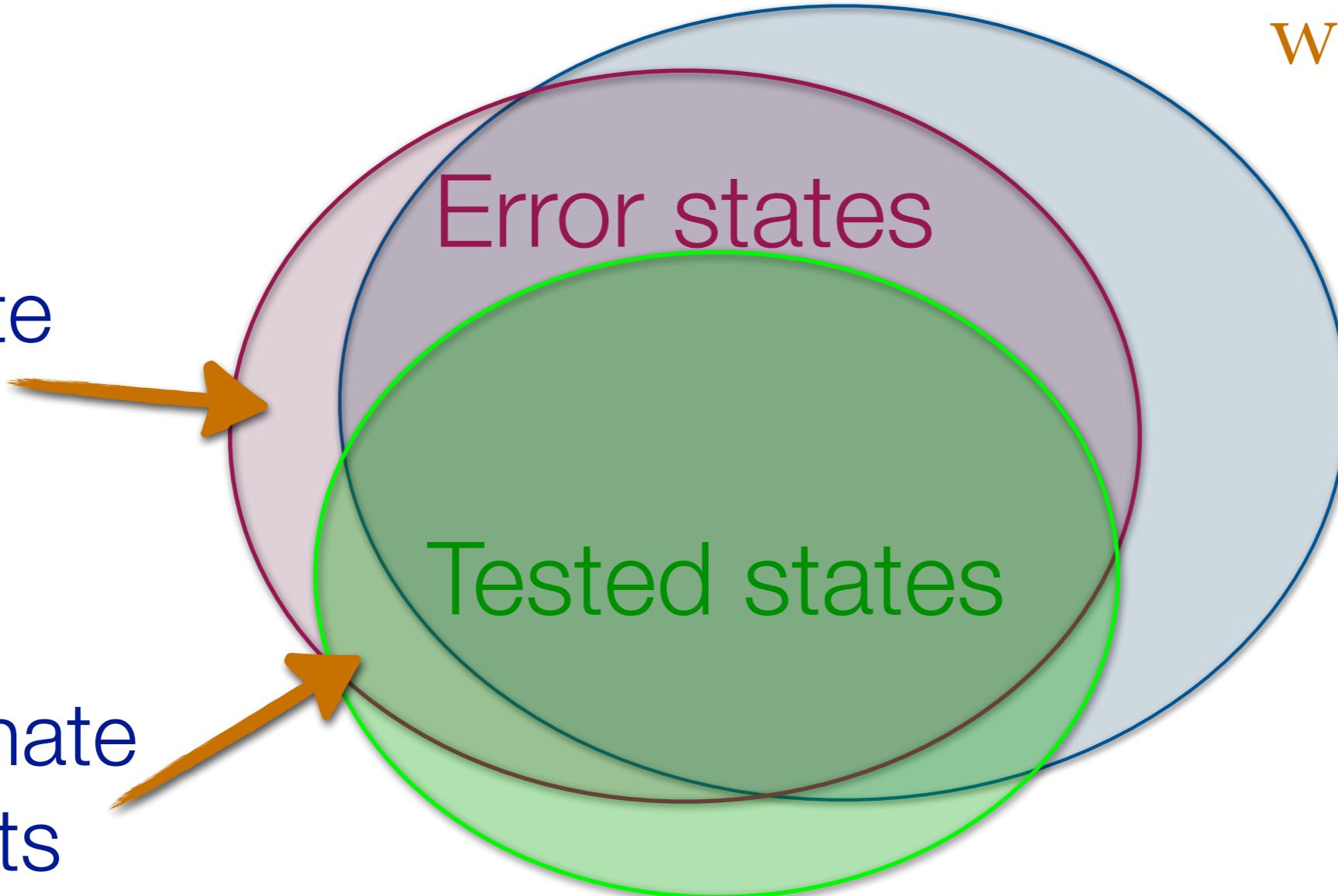


# Possible states of your application

with types

eliminate  
bugs

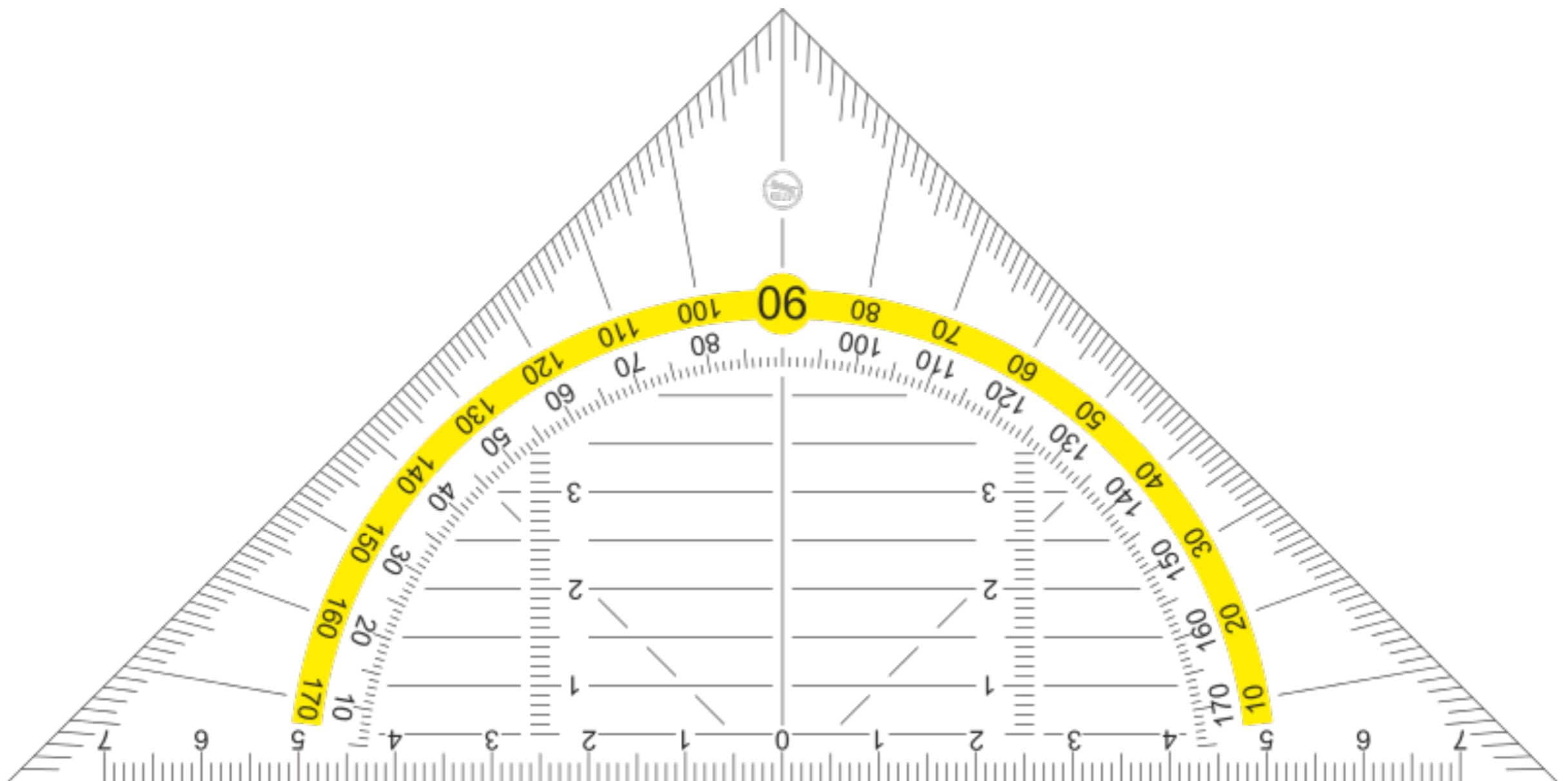
eliminate  
tests

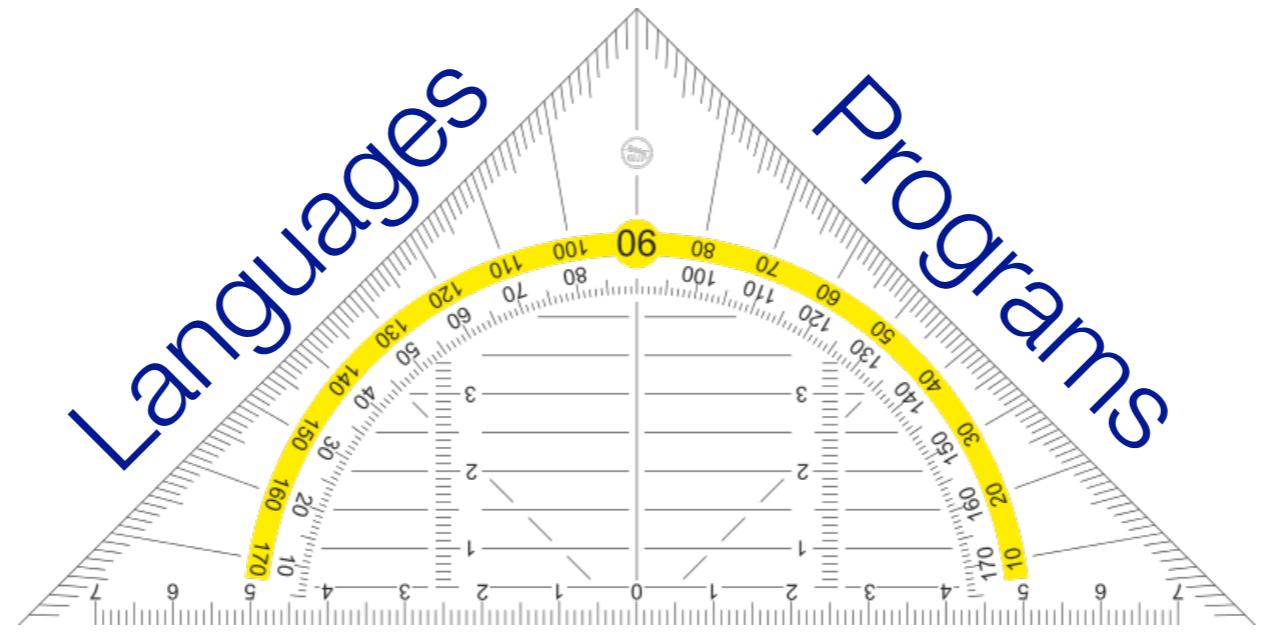


# Are types restraints?

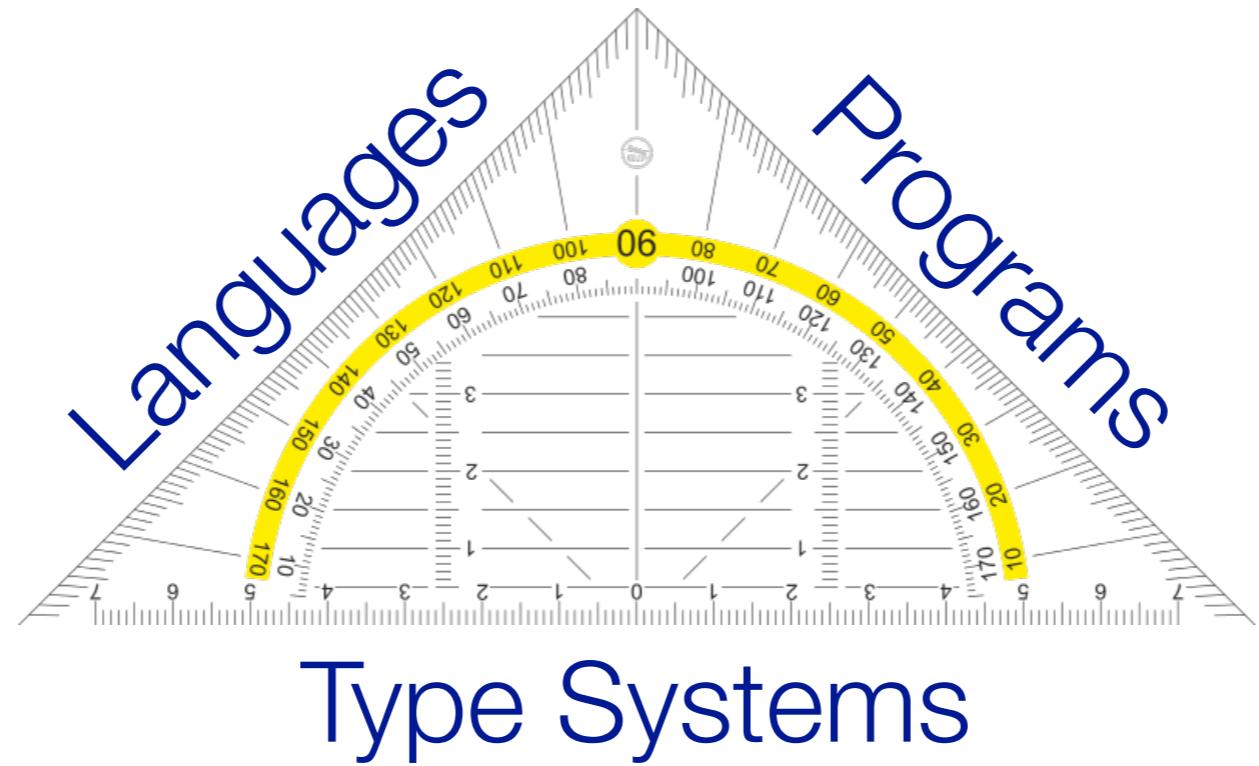


# Types are a design tool!





Type Systems



“Make undesirable states  
unrepresentable.”

“Swift encourages typed functional programming in a mainstream language.”

“Swift encourages typed  
functional programming in a  
mainstream language.”

“Swift encourages typed functional programming in a mainstream language.”

## Value types

Immutable model & UI state machines



“Swift encourages typed functional programming in a mainstream language.”

## Value types

Immutable model & UI state machines

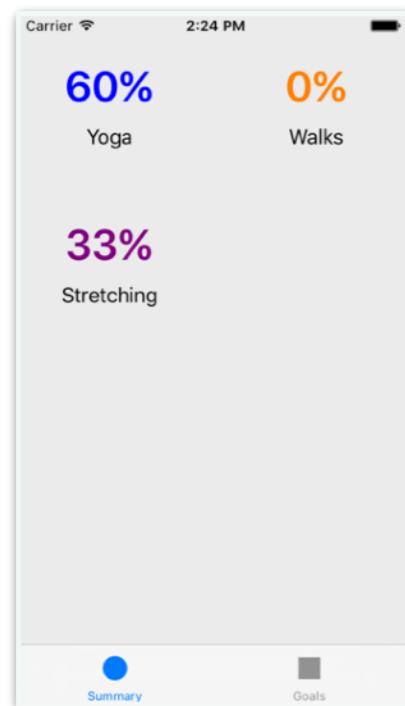


## Protocols with associated types

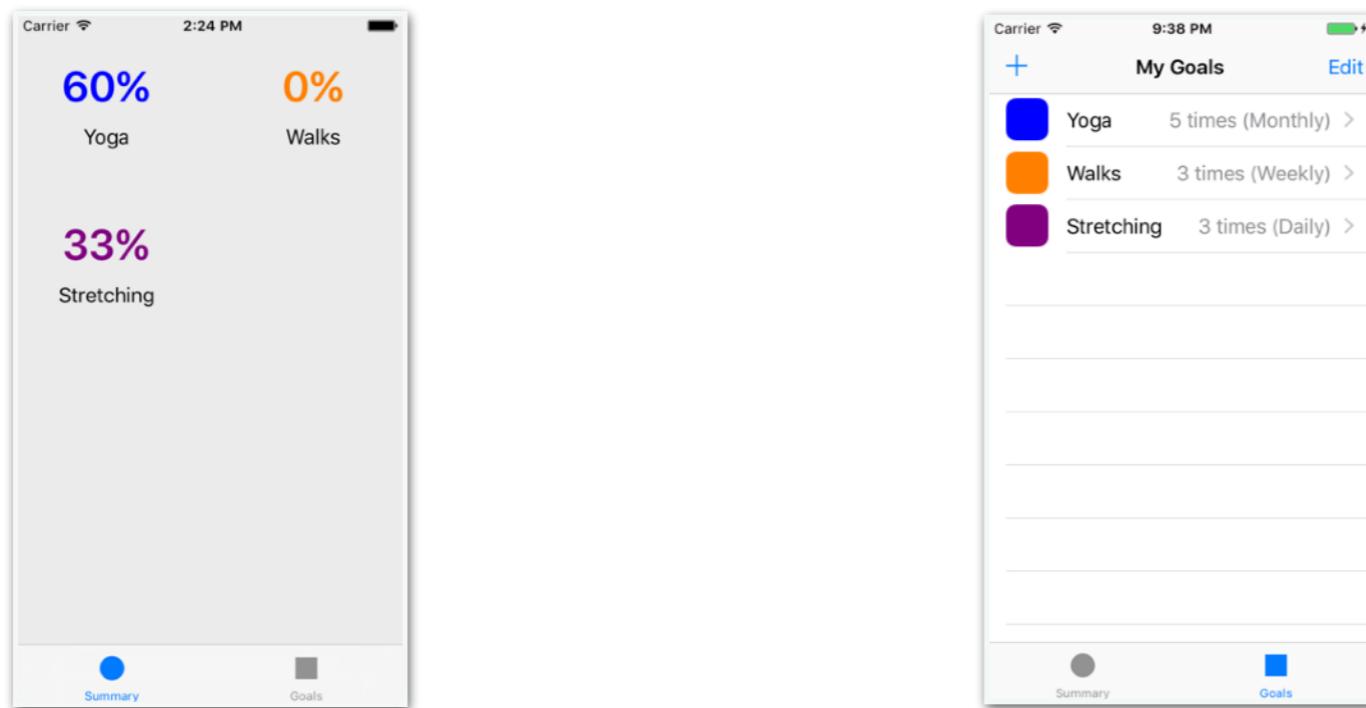
Generic change propagation



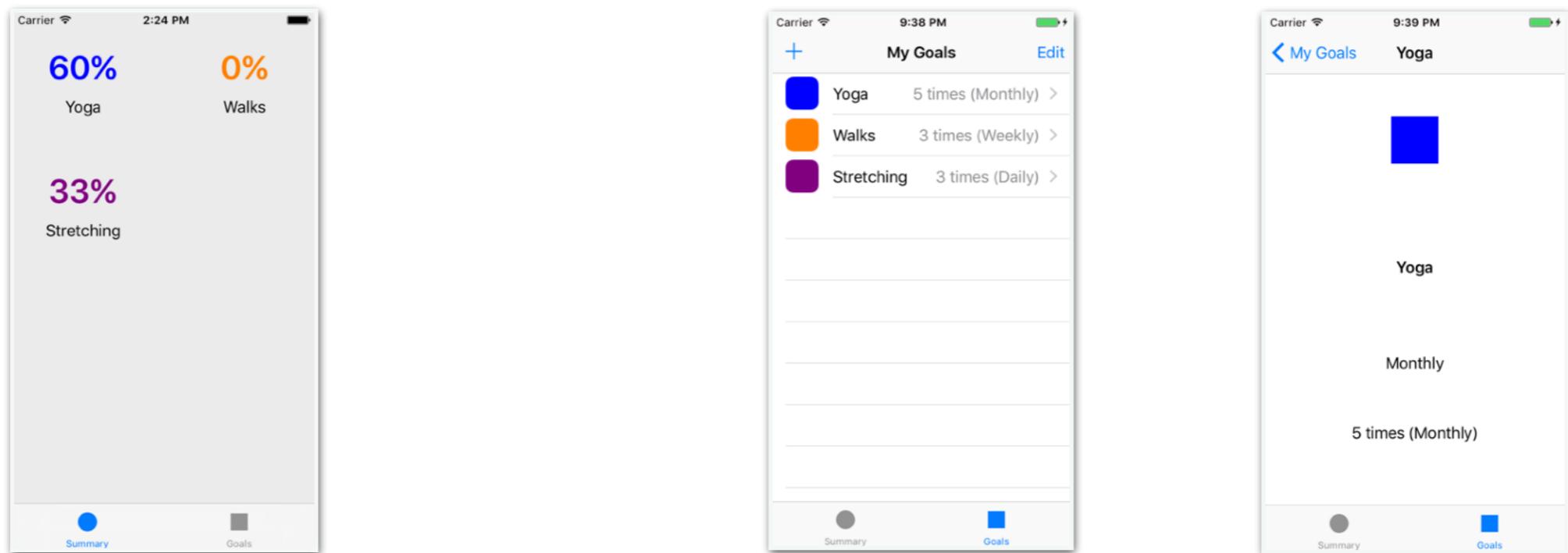
# Goals.app



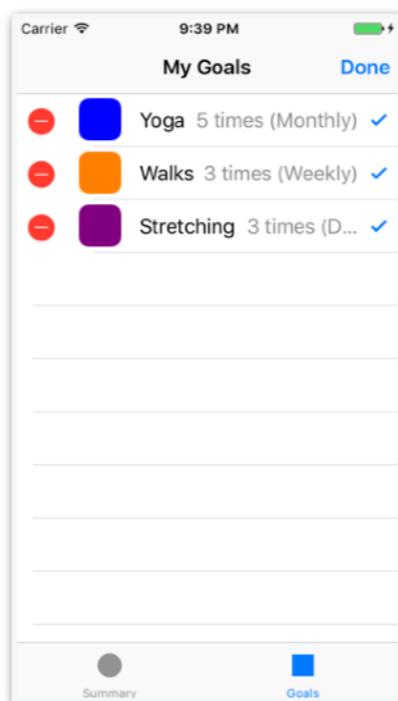
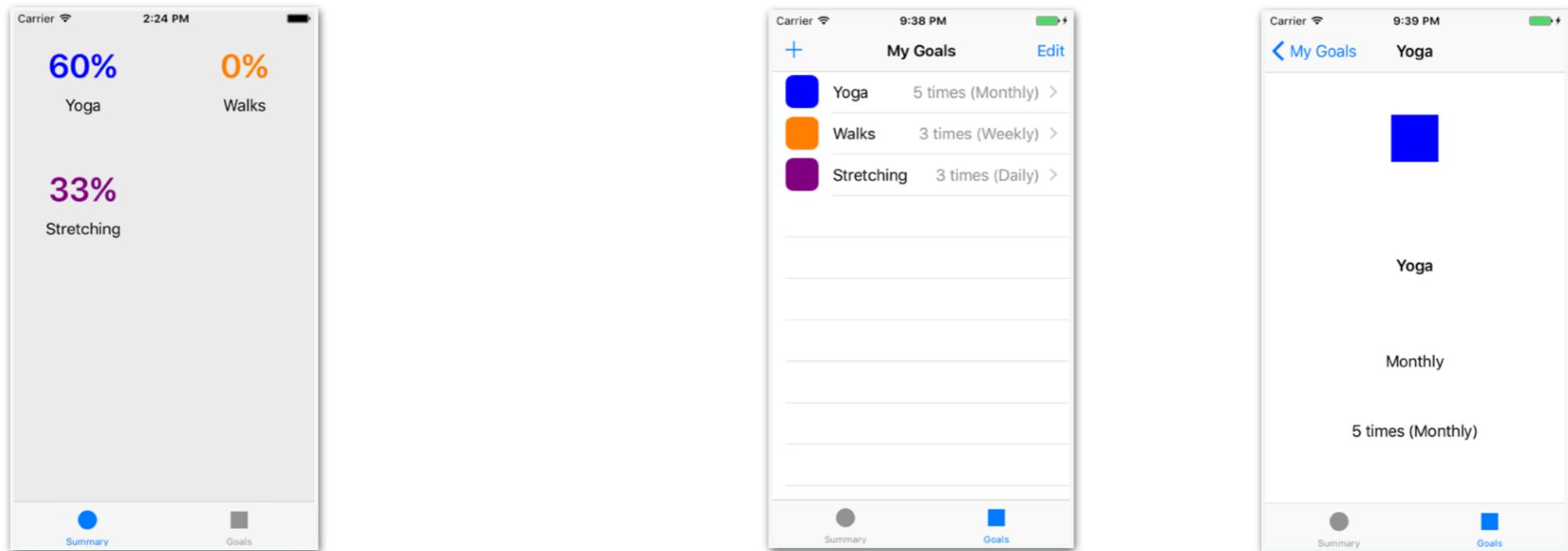
# Goals.app



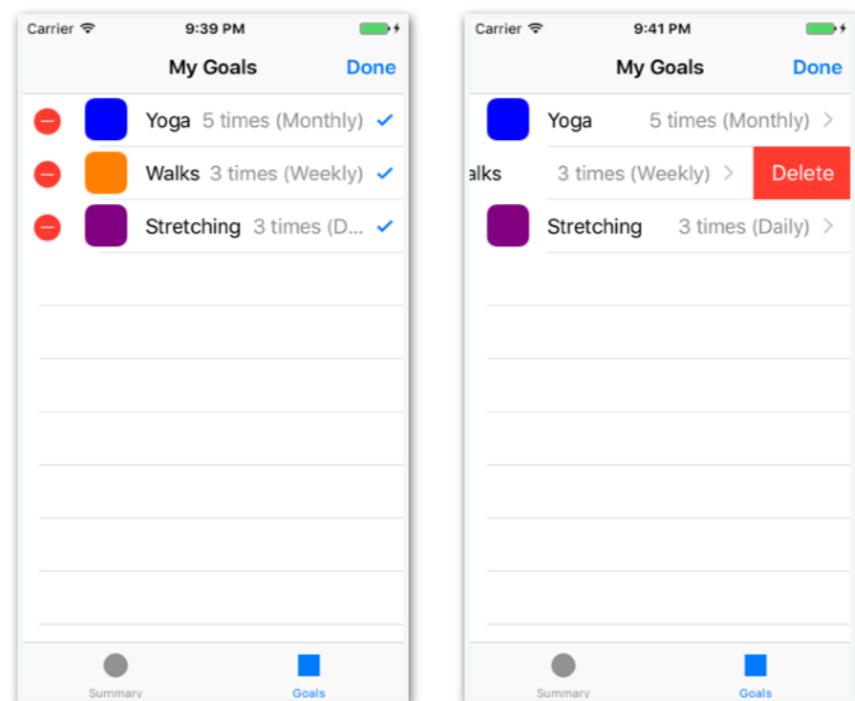
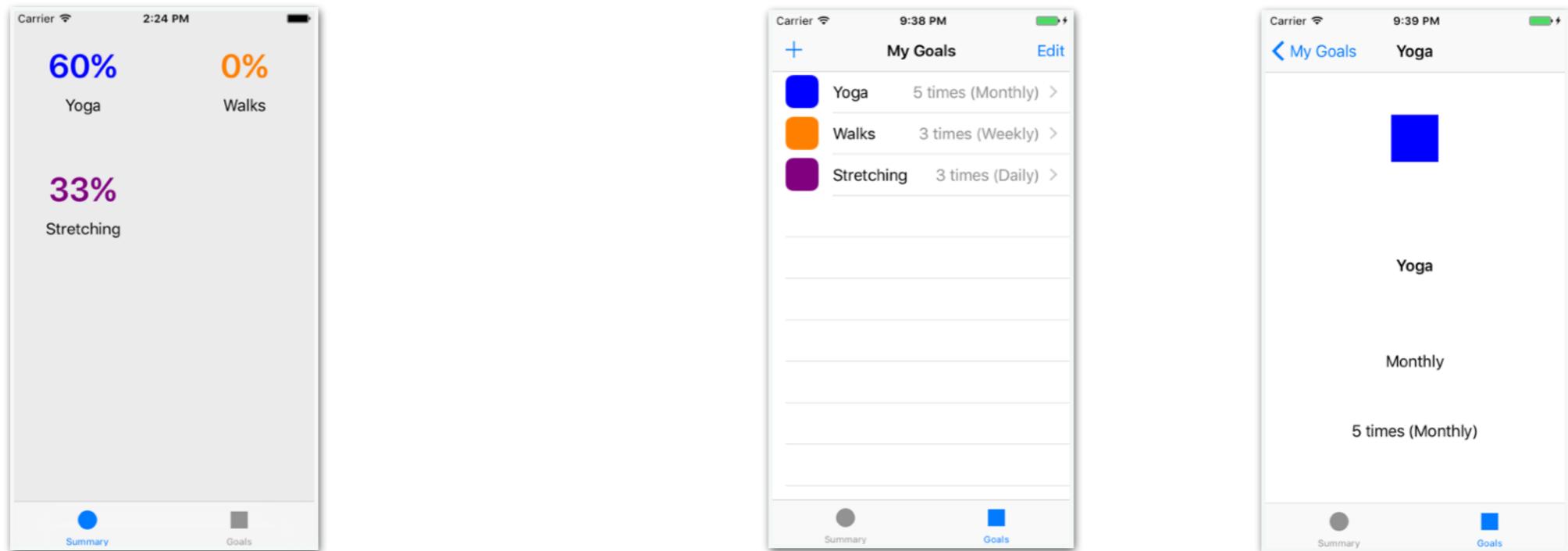
# Goals.app



# Goals.app



# Goals.app



A close-up photograph of a pile of gold bullion. In the foreground, several gold bars are visible, each featuring a crown emblem and the text "CROWNE MINT". Behind them, a stack of gold coins is visible, with one coin clearly showing the text "CROWN GOLD TEN GRAMS".

Value Types  
Localise & structure change

# Value types in Swift

# Value types in Swift

struct

```
struct Point {  
    var x: Double  
    var y: Double  
}
```

# Value types in Swift

struct

```
struct Point {  
    var x: Double  
    var y: Double  
}
```

enum with associated values

```
enum Result<ResultType> {  
    case result(value: ResultType)  
    case error(err: NSError)  
}
```

# Value types in Swift

generic type  
parameter

struct

```
struct Point {  
    var x: Double  
    var y: Double  
}
```

enum with associated values

```
enum Result<ResultType> {  
    case result(value: ResultType)  
    case error(err: NSError)  
}
```

# Reference types versus value types

```
var v: RefPnt
```

```
var w: RefPnt
```

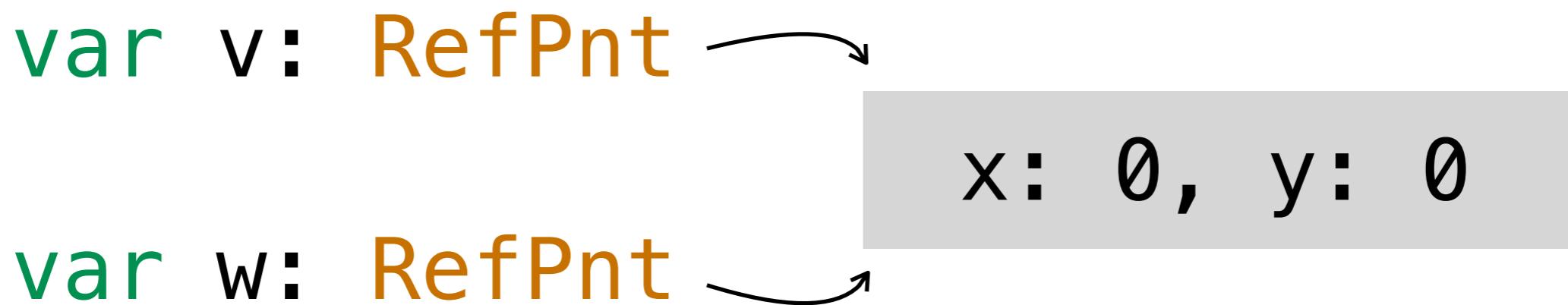
```
class RefPnt { var x, y: Double ... }
```

# Reference types versus value types

```
var v: RefPnt →  
      x: 0, y: 0  
var w: RefPnt
```

```
class RefPnt { var x, y: Double ... }  
v = RefPnt(x: 0, y: 0)
```

# Reference types versus value types



```
class RefPnt { var x, y: Double ... }

v = RefPnt(x: 0, y: 0)
w = v
```

# Reference types versus value types

```
var v: RefPnt → x: 10, y: 0  
var w: RefPnt ↗
```

```
class RefPnt { var x, y: Double ... }  
  
v = RefPnt(x: 0, y: 0)  
w = v  
w.x = 10
```

# Reference types versus value types

```
var v: Point
```

```
var w: Point
```

```
struct Point { var x, y: Double }
```

# Reference types versus value types

```
var v: Point → x: 0, y: 0
```

```
var w: Point
```

```
struct Point { var x, y: Double }  
v = Point(x: 0, y: 0)
```

# Reference types versus value types

```
var v: Point → x: 0, y: 0
```

```
var w: Point → x: 0, y: 0
```

```
struct Point { var x, y: Double }

v = Point(x: 0, y: 0)
w = v
```

# Reference types versus value types

```
var v: Point → x: 0, y: 0
```

```
var w: Point → x: 10, y: 0
```

```
struct Point { var x, y: Double }

v = Point(x: 0, y: 0)
w = v
w.x = 10
```

# Value types

```
var v: Point → x: 0, y: 0
```

```
var w: Point → x: 10, y: 0
```

# Value types

```
var v: Point → x: 0, y: 0
```

```
var w: Point → x: 10, y: 0
```

Localise change

# Value types

```
var v: Point → x: 0, y: 0
```

```
var w: Point → x: 10, y: 0
```

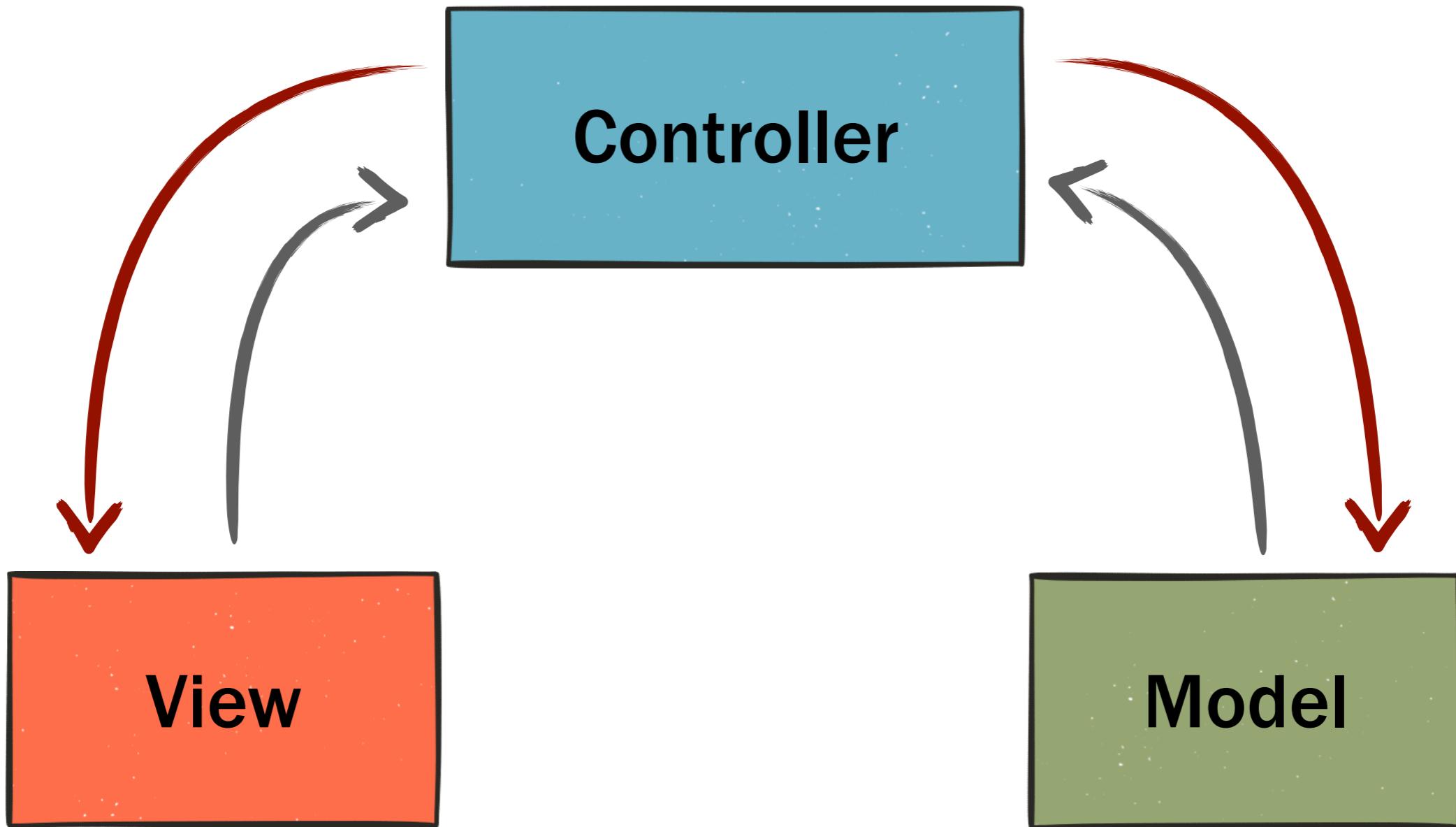
Localise change

Facilitate local reasoning

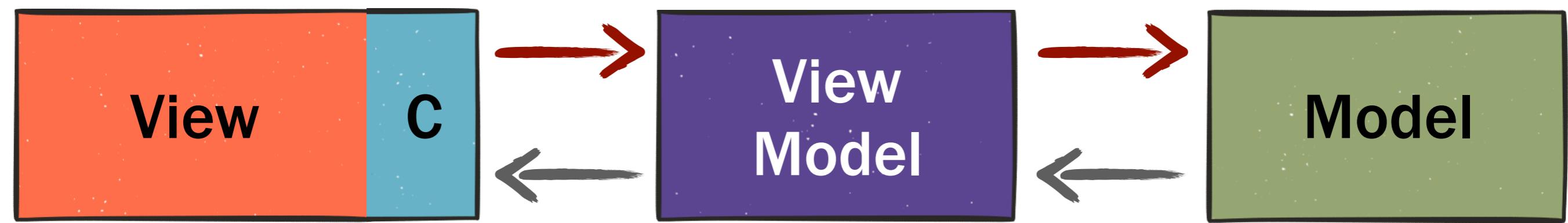
# Value Types

## Immutable Model

# MVC & MVVM Architectures

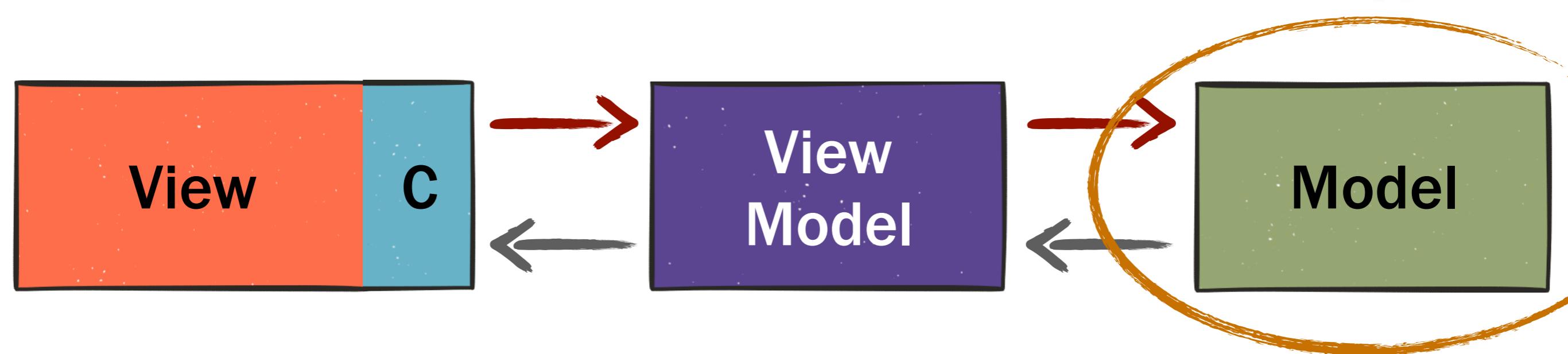


# MVC & MVVM Architectures



# MVC & MVVM Architectures

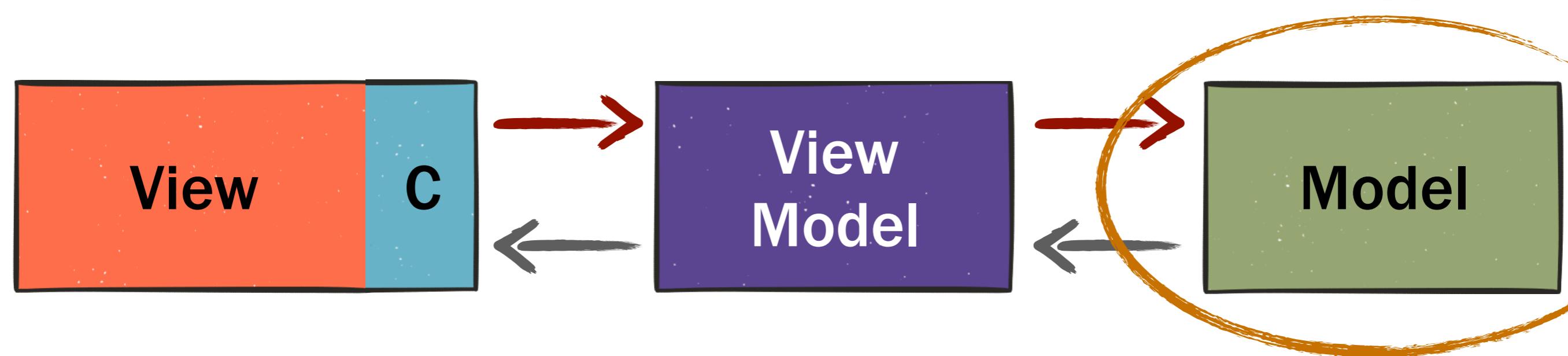
passed around as  
reference type



# MVC & MVVM Architectures

✗ Accidental changes

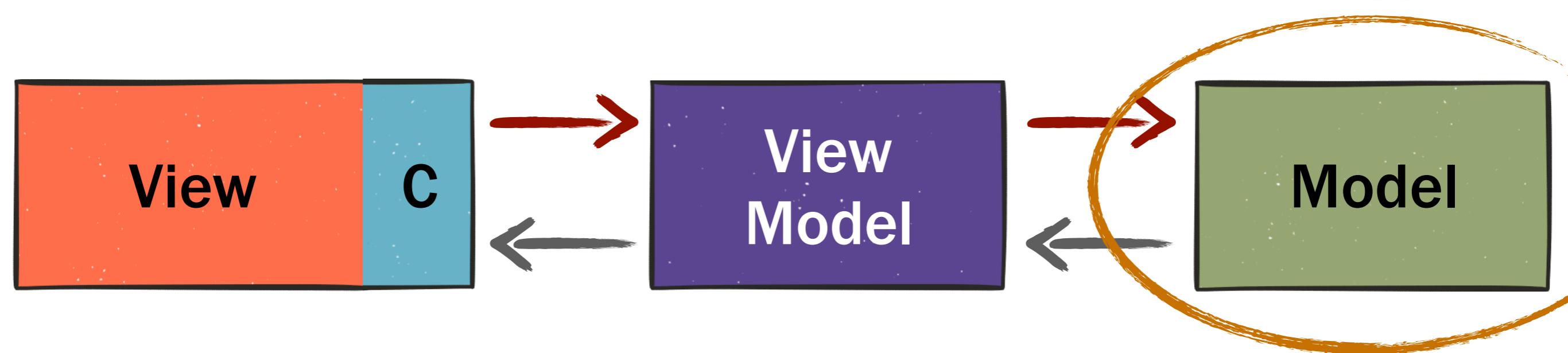
passed around as  
reference type



# MVC & MVVM Architectures

- ✗ Accidental changes
- ✗ Changes in wrong order

passed around as  
reference type



# Immutable Goals

```
struct Goal {  
    let uuid: UUID // fast equality  
    var colour: UIColor  
    var title: String  
    var interval: GoalInterval  
    var frequency: Int  
}
```

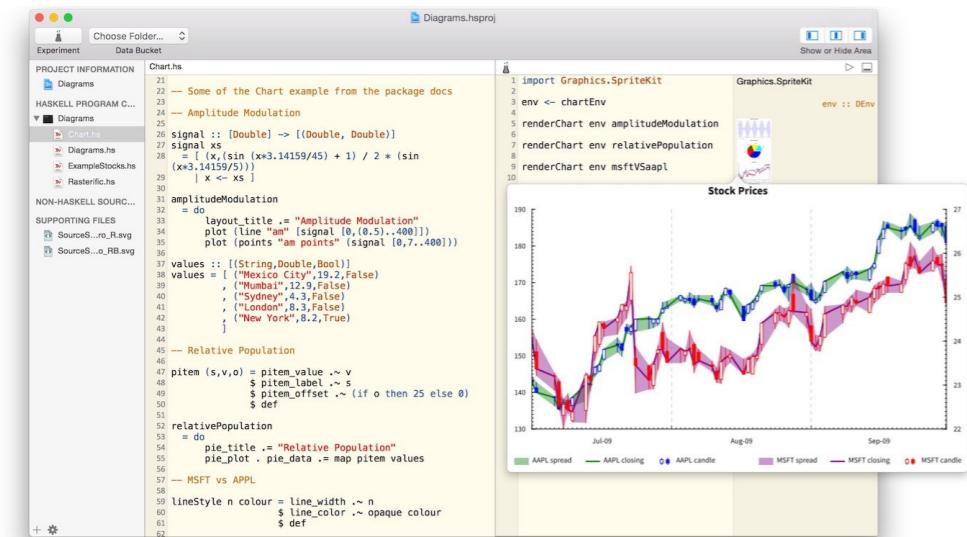
# Immutable Goals

```
struct Goal {  
    let uuid: UUID // fast equality  
    var colour: UIColor  
    var title: String  
    var interval: GoalInterval  
    var frequency: Int  
}  
  
typealias GoalProgress = (goal: Goal, count: Int?)
```

# Immutable Goals

```
struct Goal {  
    let uuid: UUID // fast equality  
    var colour: UIColor  
    var title: String  
    var interval: GoalInterval  
    var frequency: Int  
}  
  
typealias GoalProgress = (goal: Goal, count: Int?)  
  
typealias Goals = [GoalProgress] // array
```

# Immutable Models Do Scale!



The screenshot shows a Haskell IDE interface with the following details:

- Project Information:** A sidebar listing files like `Chart.hs`, `Diagrams.hs`, `ExampleStocks.hs`, and `Rasterific.hs`.
- Code Editor:** The main window displays `Chart.hs` with Haskell code. The code includes imports for `Graphics.SpriteKit` and `Graphics.Spriter`. It defines functions for amplitude modulation, relative population, and MSFT vs APPL. A chart titled "Stock Prices" is shown with multiple lines representing different stock data.
- Diagram View:** A preview pane shows a pie chart titled "Relative Population".

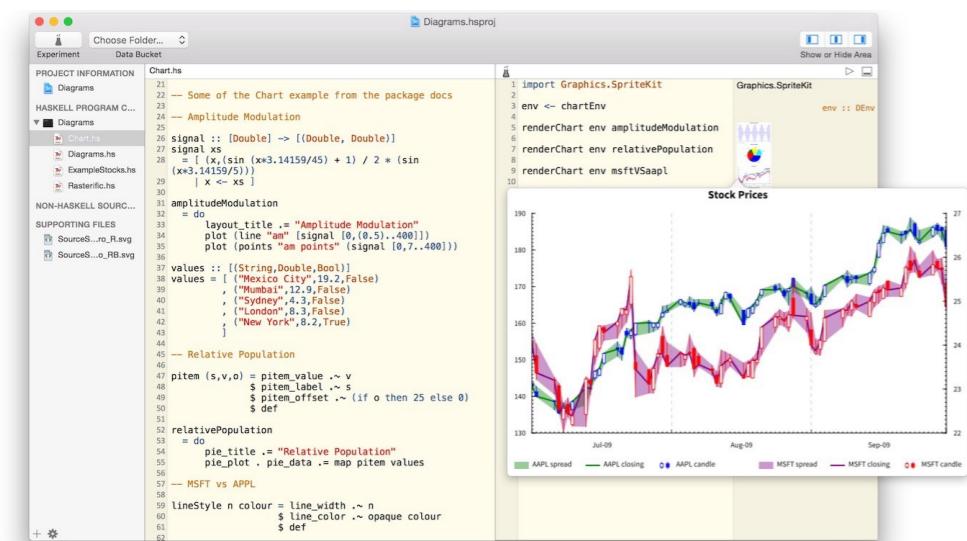


# Immutable Models Do Scale!

## Haskell for Mac

Uses an immutable model

Adopts Cocoa Document Architecture



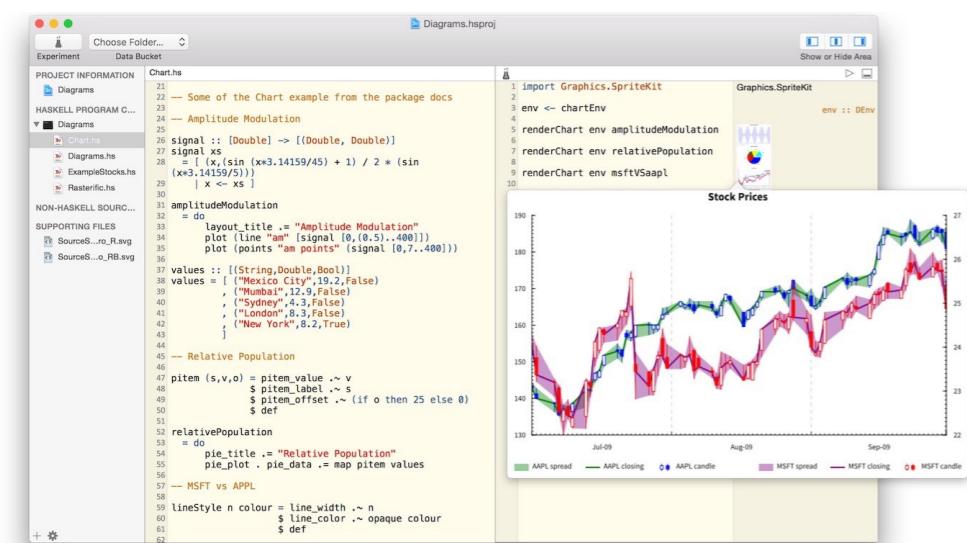
# Immutable Models Do Scale!

## Haskell for Mac

Uses an immutable model

Adopts Cocoa Document Architecture

“Functional Programming in a Stateful World”  
YOW! Lambda Jam 2015  
(on [speakerdeck.com](https://speakerdeck.com))



But what if...

# But what if...

“Use an **immutable** wrapper API  
to wrap mutable structures.”

# But what if...

“Use an **immutable** wrapper API  
to wrap mutable structures.”

✓ React (Native)

**Virtual DOM wraps DOM**

# But what if...

“Use an **immutable** wrapper API  
to wrap mutable structures.”

- ✓ React (Native)

**Virtual DOM wraps DOM**

- ✓ Haskell SpriteKit — Haskell library binding

**Algebraic data type wraps SpriteKit’s mutable object graph**

# Replacing Tests

# Replacing Tests

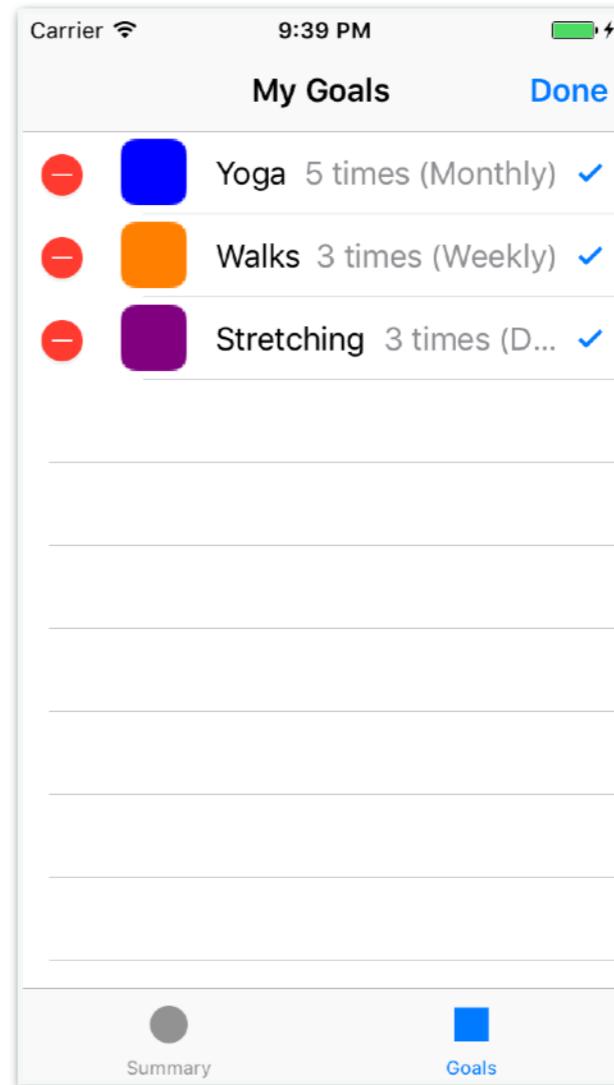
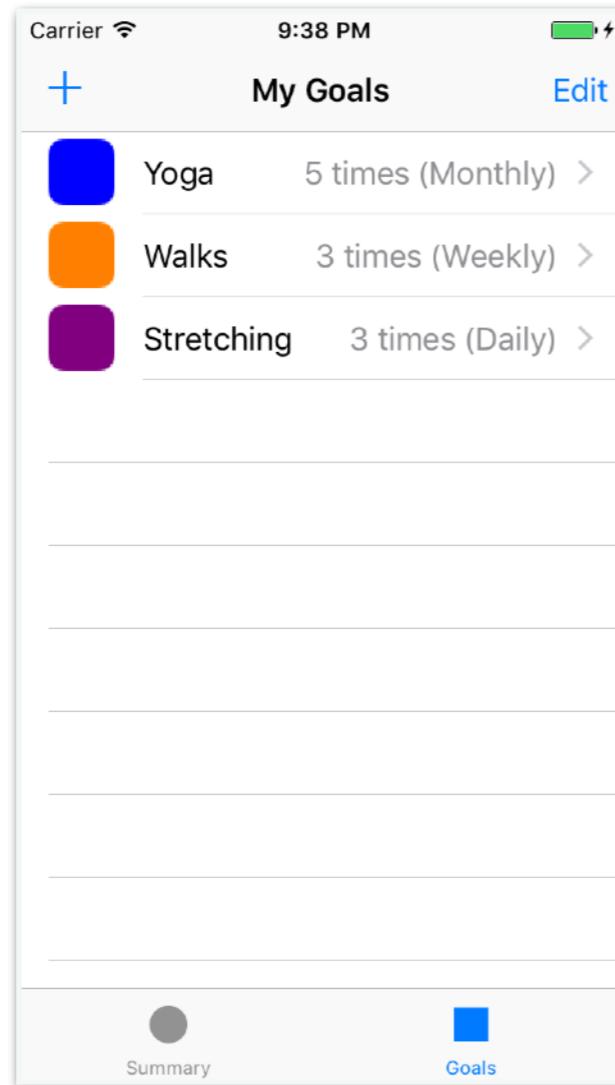
- ✓ Avoid accidental changes of model state
- Replaces integration tests

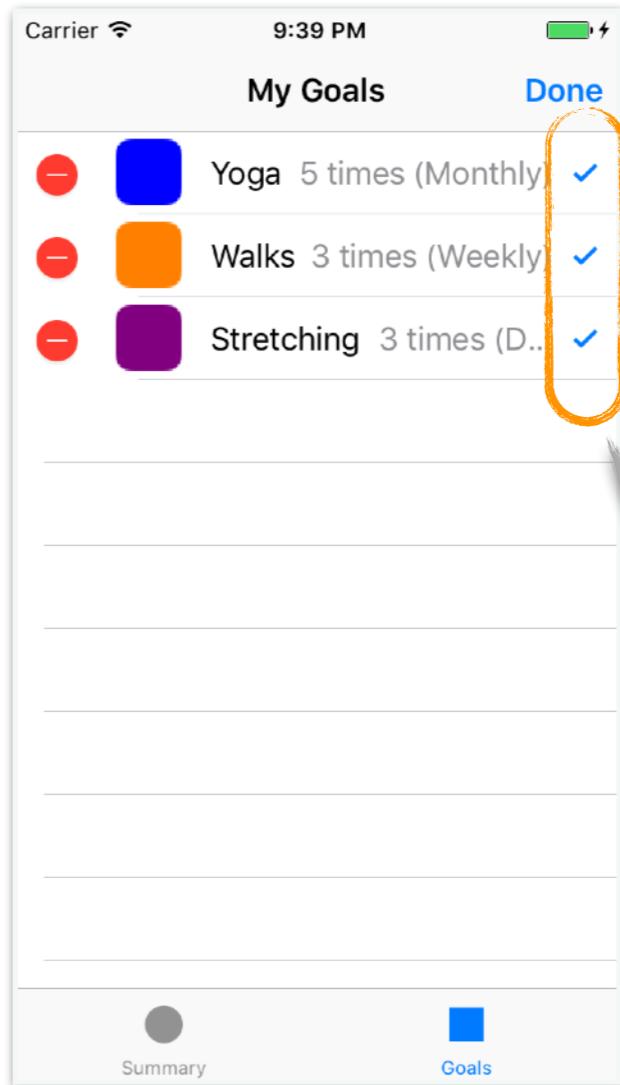
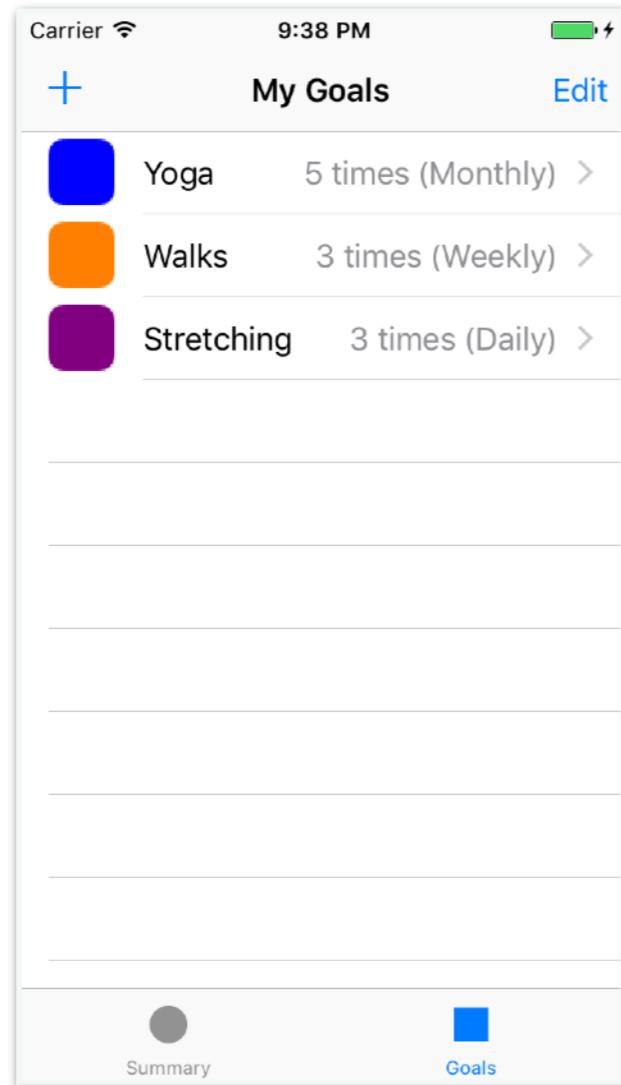
# Replacing Tests

- ✓ Avoid accidental changes of model state  
**Replaces integration tests**
- ✓ Avoid races on concurrent threads  
**Often not properly tested**

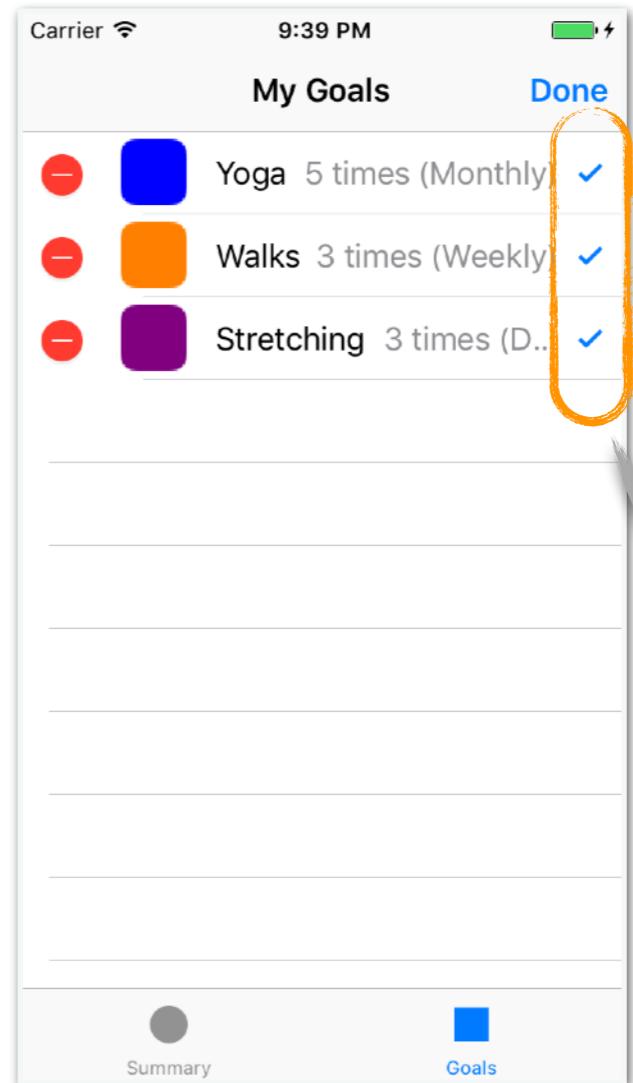
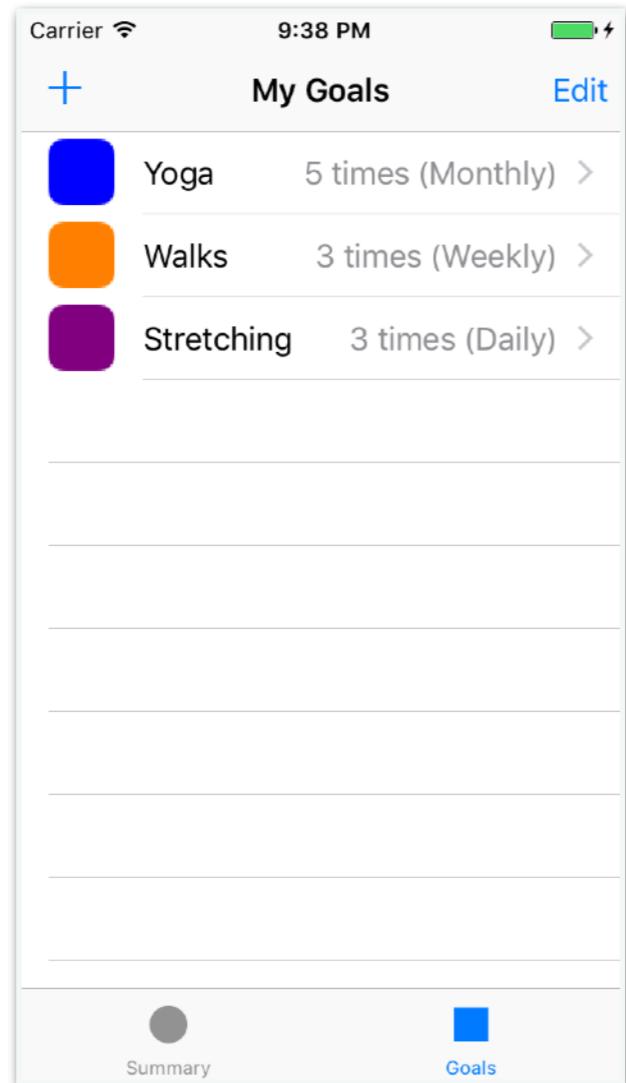
# Enums with Associated Values

## UI State Machines



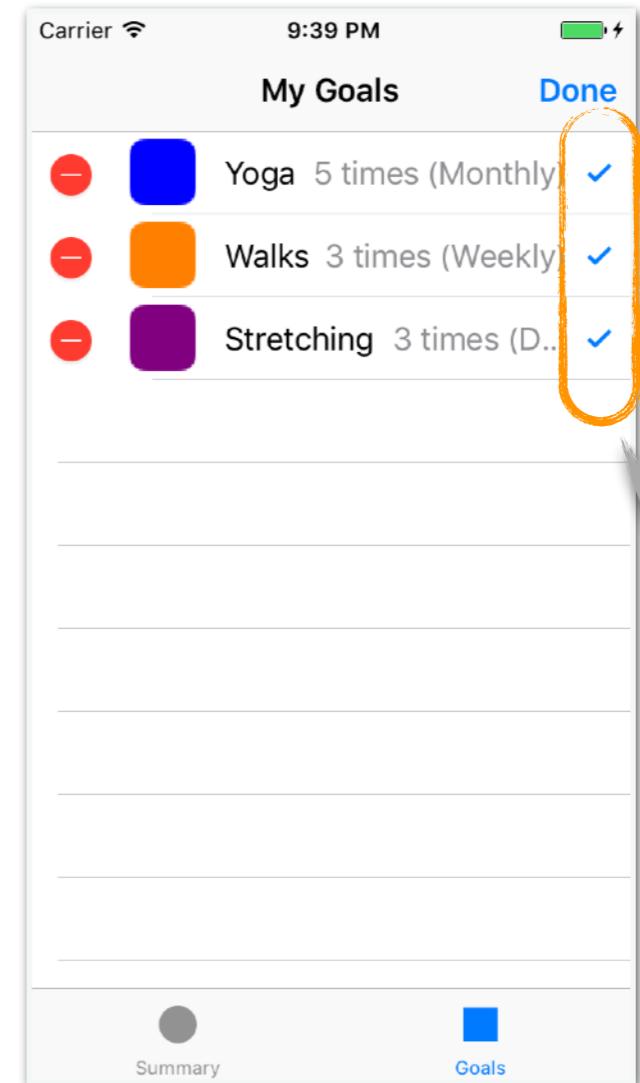
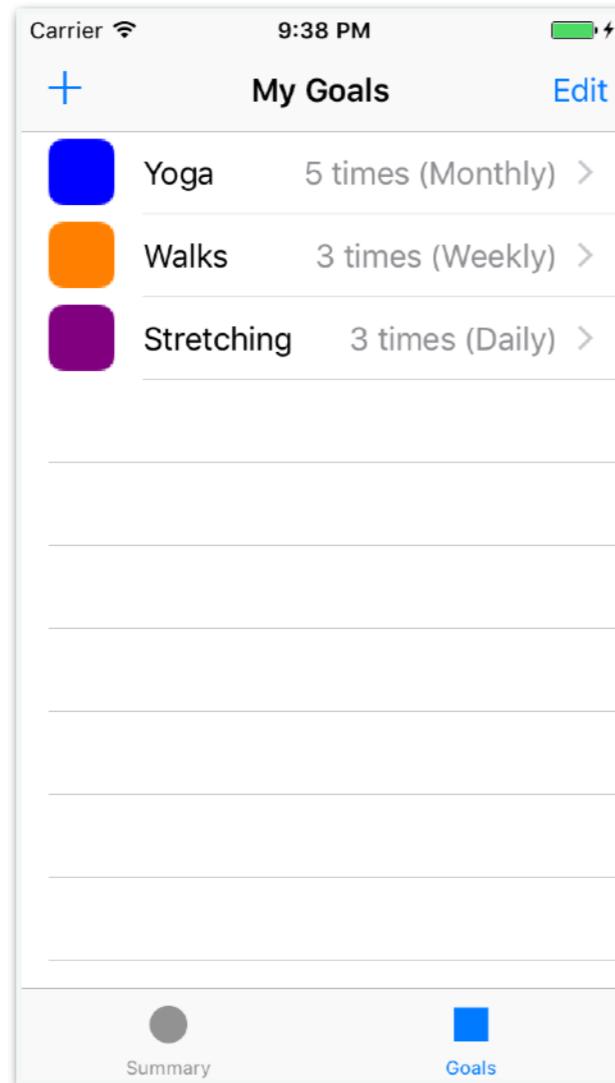


Is goal  
active?



Is goal active?

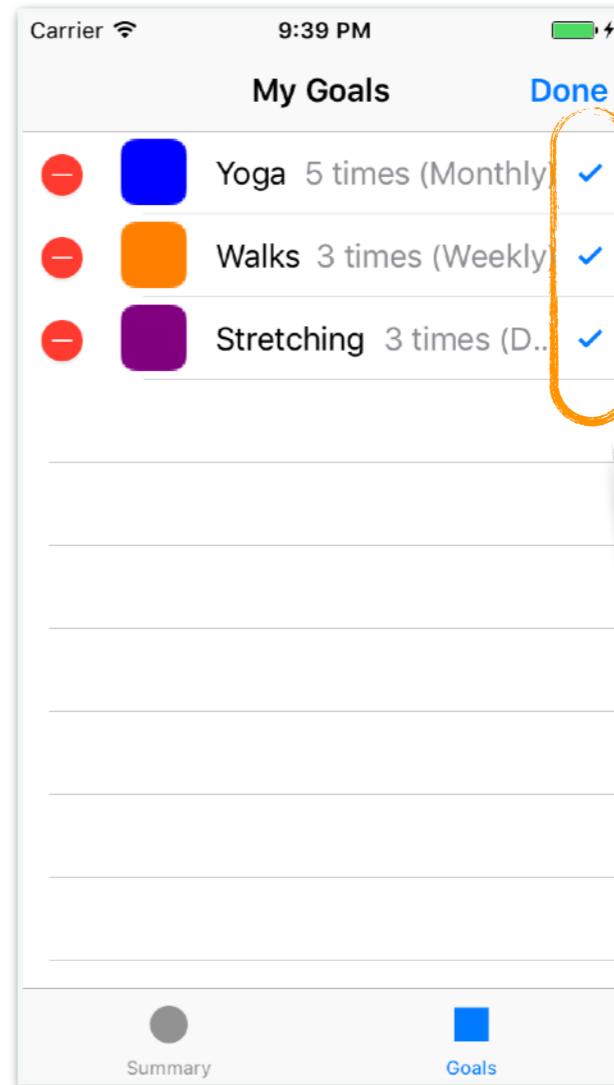
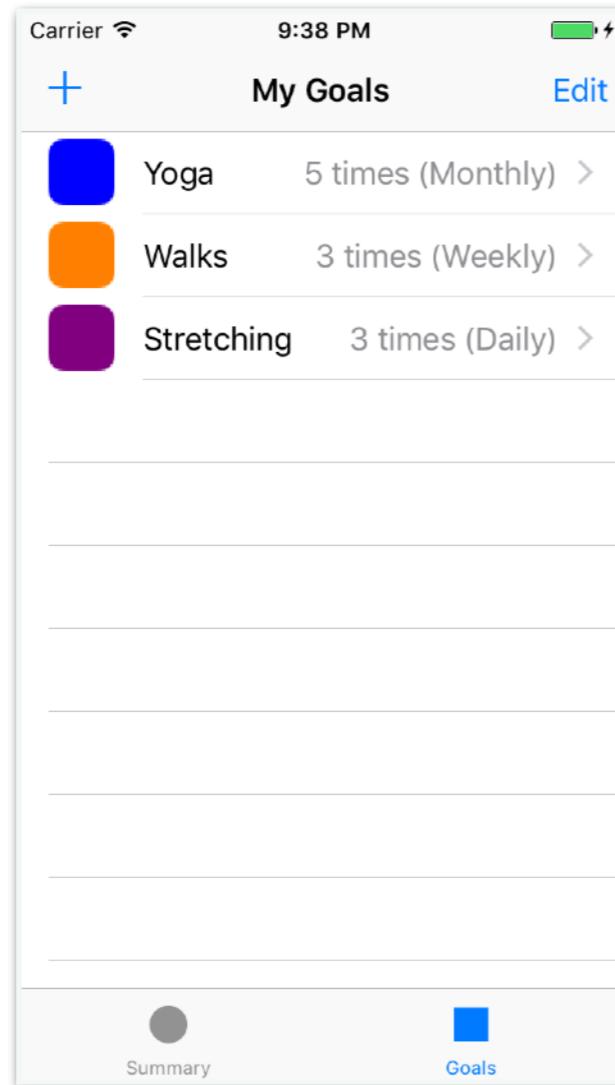
```
typealias GoalProgress = (goal: Goal, count: Int?)  
typealias Goals = [GoalProgress]
```



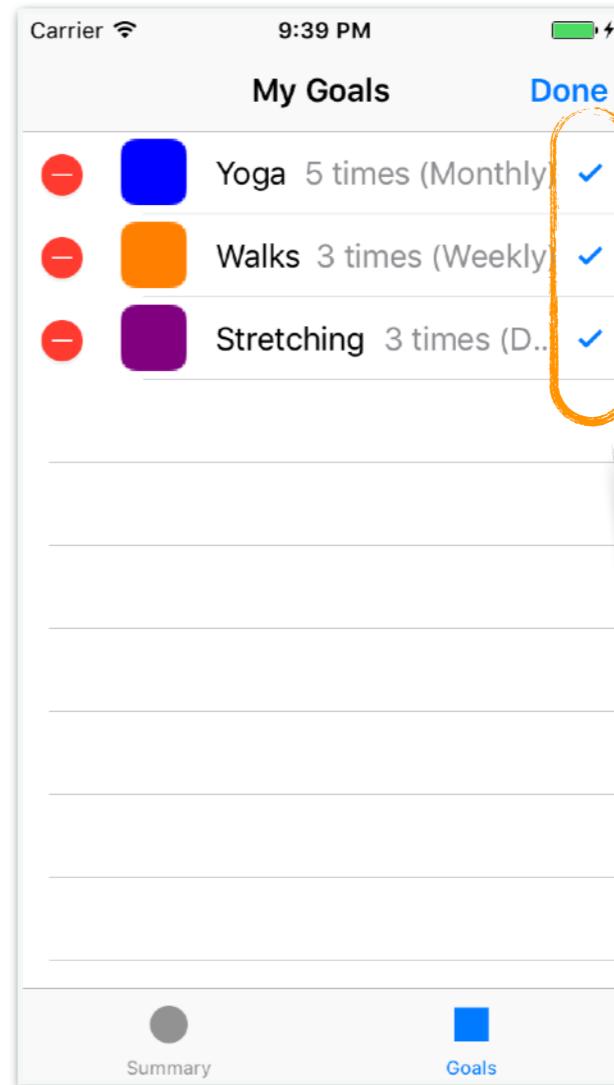
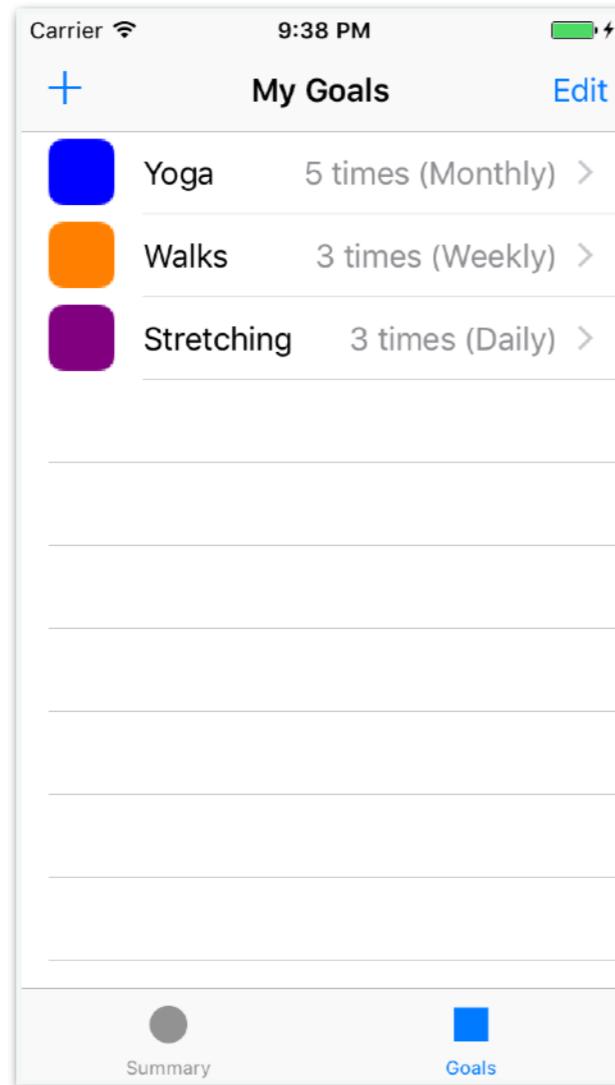
Is goal  
active?

```
typealias GoalProgress = (goal: Goal, count: Int?)  
typealias Goals = [GoalProgress]
```

Need to be careful not to lose progress info



```
var goalsActivity: [Bool]
```



```
var goalsActivity: [Bool]
```

Not valid outside of editing mode

```
enum GoalsEditState {  
    case displaying  
    case editing  
}
```

var goalsActivity: [Bool]

```
enum GoalsEditState {  
    case displaying  
    case editing(goalsActivity: [Bool])  
}
```

```
enum GoalsEditState {  
    case displaying  
    case editing(goalsActivity: [Bool])  
}
```

GoalsController: UITableViewController

```
var editState: GoalsEditState = .displaying
```

...

```
enum GoalsEditState {  
    case displaying  
    case editing(goalsActivity: [Bool])  
}
```

GoalsController: UITableViewController

```
var editState: GoalsEditState = .displaying  
...  
switch editState { // change mode
```

```
enum GoalsEditState {  
    case displaying  
    case editing(goalsActivity: [Bool])  
}
```

## GoalsController: UITableViewController

```
var editState: GoalsEditState = .displaying  
...  
switch editState {  
    case .displaying:  
        navigationItem.setLeftBarButton(nil, ...) // change mode  
        editState = .editing(goalsActivity:  
            dataSource.goalsActivity) alter UI state
```

```
enum GoalsEditState {  
    case displaying  
    case editing(goalsActivity: [Bool])  
}
```

## GoalsController: UITableViewController

```
var editState: GoalsEditState = .displaying  
...  
switch editState {  
    case .displaying:  
        navigationItem.setLeftBarButton(nil, ...)  
        editState = .editing(goalsActivity:  
            dataSource.goalsActivity)  
  
    case .editing(let goalsActivity):  
        navigationItem.setLeftBarButton(addButton, ...)  
        dataSource.commitGoalsActivity(goalsActivity)  
        editState = .displaying }
```

alter  
UI state

```
enum GoalsEditState {  
    case displaying  
    case editing(goalsActivity: [Bool])  
}
```

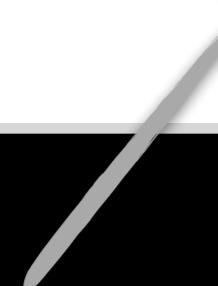
## GoalsController

```
guard case .editing(var goalsActivity) = editState  
else { return }
```

```
enum GoalsEditState {  
    case displaying  
    case editing(goalsActivity: [Bool])  
}
```

access to mode-specific  
state is guarded

GoalsController



```
guard case .editing(var goalsActivity) = editState  
else { return }
```

```
enum GoalsEditState {  
    case displaying  
    case editing(goalsActivity: [Bool])  
}
```

access to mode-specific  
state is guarded

GoalsController

```
guard case .editing(var goalsActivity) = editState  
else { return }  
  
let newIsActive = !goalsActivity[indexPath.item]  
cell.editingAccessoryType = newIsActive ? .checkmark  
: .none
```

# Replacing Tests

# Replacing Tests

- ✓ Avoid inconsistent state changes
- Replaces UI tests

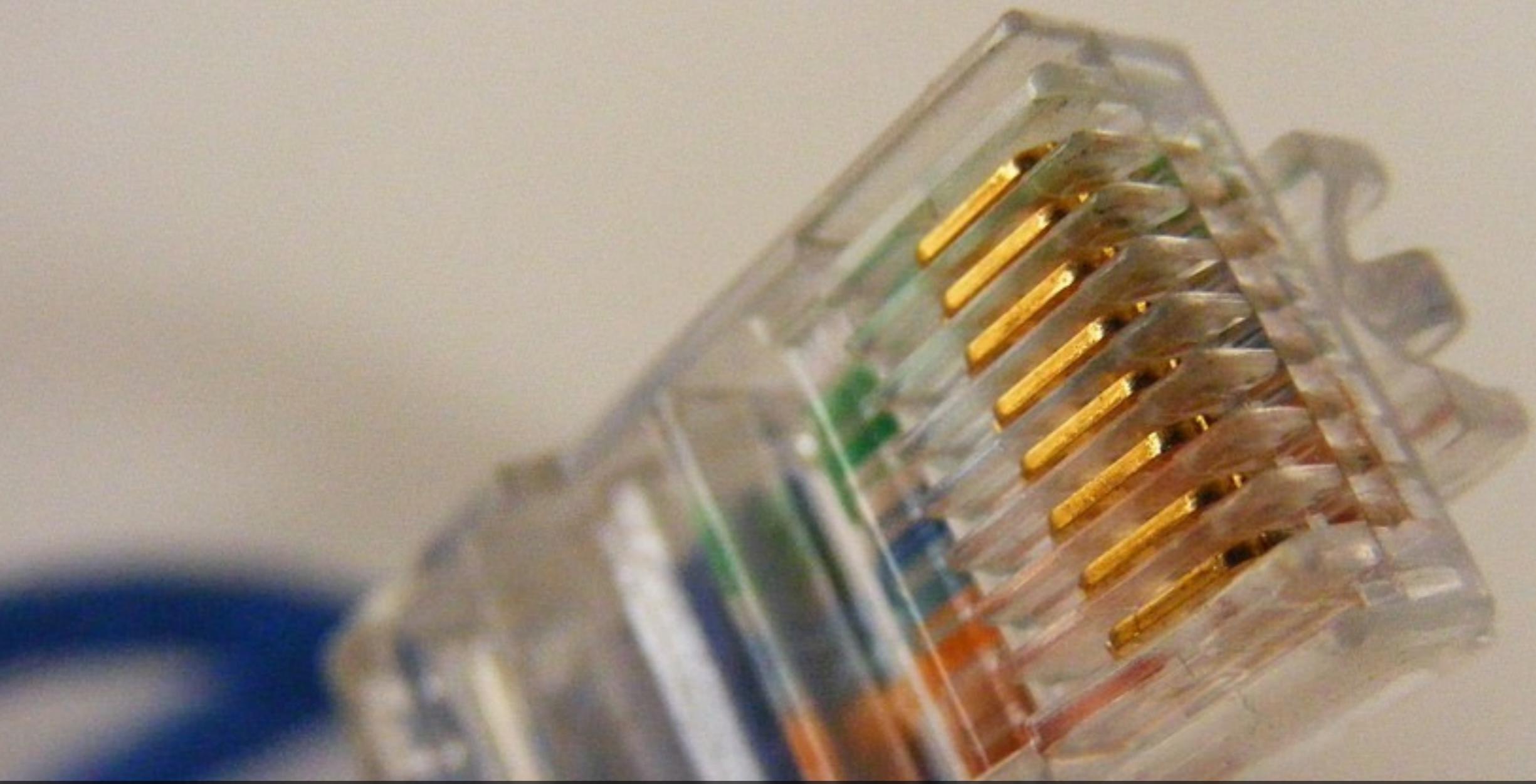
# Replacing Tests

✓ Avoid inconsistent state changes

**Replaces UI tests**

✓ Avoid non-exhaustive transitions

**Replaces exhaustiveness tests**



Protocols with associated types  
Structured change propagation

“Now that the model layer is  
immutable, how do we change  
anything?”

# Replacing Tests

# Replacing Tests

- ✓ Avoid changes from unexpected places  
**Replaces integration tests**

# Replacing Tests

- ✓ Avoid changes from unexpected places

**Replaces integration tests**

- ✓ Avoid conflicting state changes

**Replaces integration tests**

# Check out the source code of Goals.app!

<https://github.com/mchakravarty/goalsapp>



[haskellformac.com](http://haskellformac.com)



[mchakravarty](https://github.com/mchakravarty)



[TacticalGrace](https://twitter.com/TacticalGrace)



[justtesting.org](http://justtesting.org)



types >< state

# Check out the source code of Goals.app!

<https://github.com/mchakravarty/goalsapp>

## Type systems are a design tool

## Swift encourages typed functional programming

## Types replace entire categories of tests



[haskellformac.com](http://haskellformac.com)



[mchakravarty](https://github.com/mchakravarty)



[TacticalGrace](https://twitter.com/TacticalGrace)



[justtesting.org](http://justtesting.org)



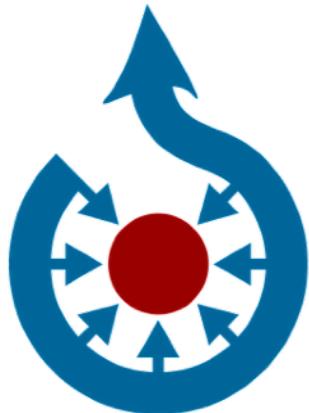
types >< state

Thank you!

# Image Attribution

**flickr**

<https://www.flickr.com/photos/30998987@N03/5408763997>  
<https://www.flickr.com/photos/provisions/7986149891/>  
<https://www.flickr.com/photos/digitalcurrency/2438119267>  
<https://www.flickr.com/photos/pviojoenchile/2760021279/>



Wikimedia

[https://commons.wikimedia.org/wiki/File:Nile\\_blue\\_05.jpg](https://commons.wikimedia.org/wiki/File:Nile_blue_05.jpg)  
[https://commons.wikimedia.org/wiki/File:Handcuffs01\\_2008-07-27.jpg](https://commons.wikimedia.org/wiki/File:Handcuffs01_2008-07-27.jpg)  
[https://commons.wikimedia.org/wiki/File:Set\\_square\\_Geodreieck.svg](https://commons.wikimedia.org/wiki/File:Set_square_Geodreieck.svg)

dribbble

<https://dribbble.com/shots/1667698-Free-vector-Macbook-Ipad-and-Iphone>

openclipart

<https://openclipart.org/detail/463/heart>