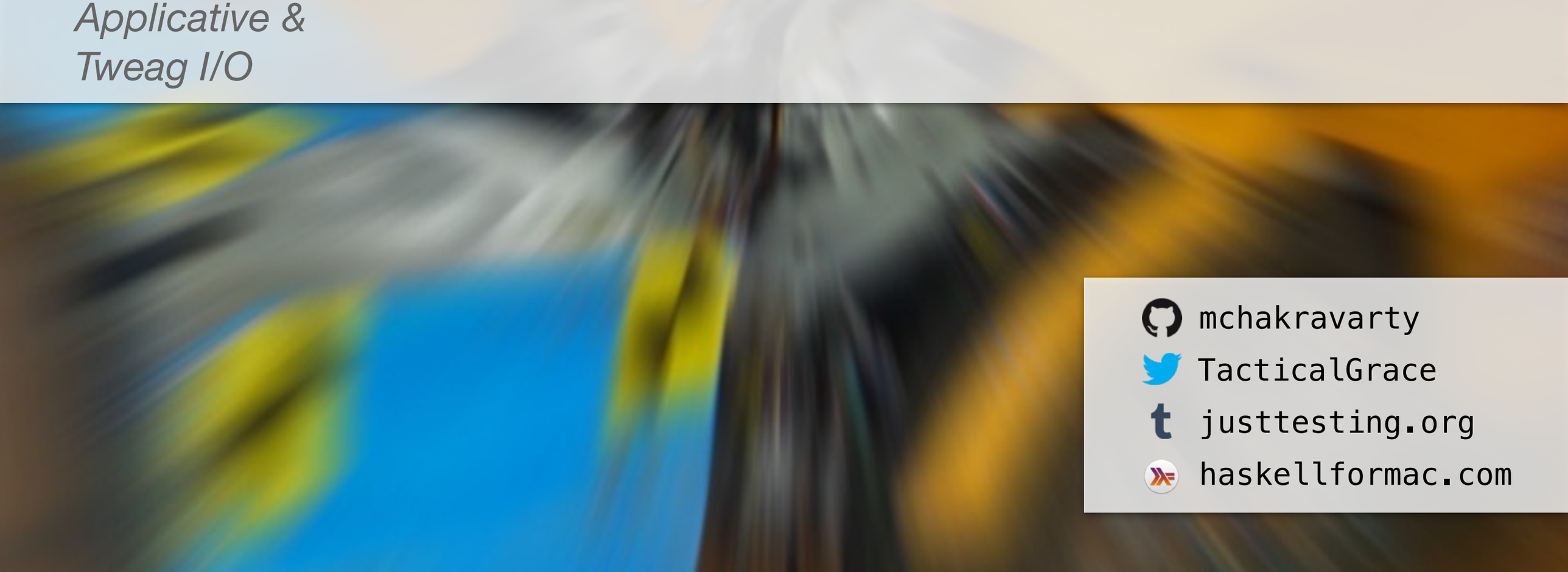
A Type is Worth a Thousand Tests

Manuel M T Chakravarty
Applicative &
Tweag I/O



Let’s talk about…

Let’s talk about…

Swift

Let's talk about…

Language Design

Swift

Let's talk about…

Language Design

Ecosystem

Swift

Let's talk about…

Why is it extraordinary?

Language

Ecosystem

Design

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

Objective-C

Cocoa

Python/Ruby

Cloud

C/C++

Unix

Java

JVM

C#

.NET

Platforms

Why do languages become popular?

JavaScript

Web

Swift

Cocoa

Python/Ruby

Cloud

C/C++

Unix

Java

JVM

C#

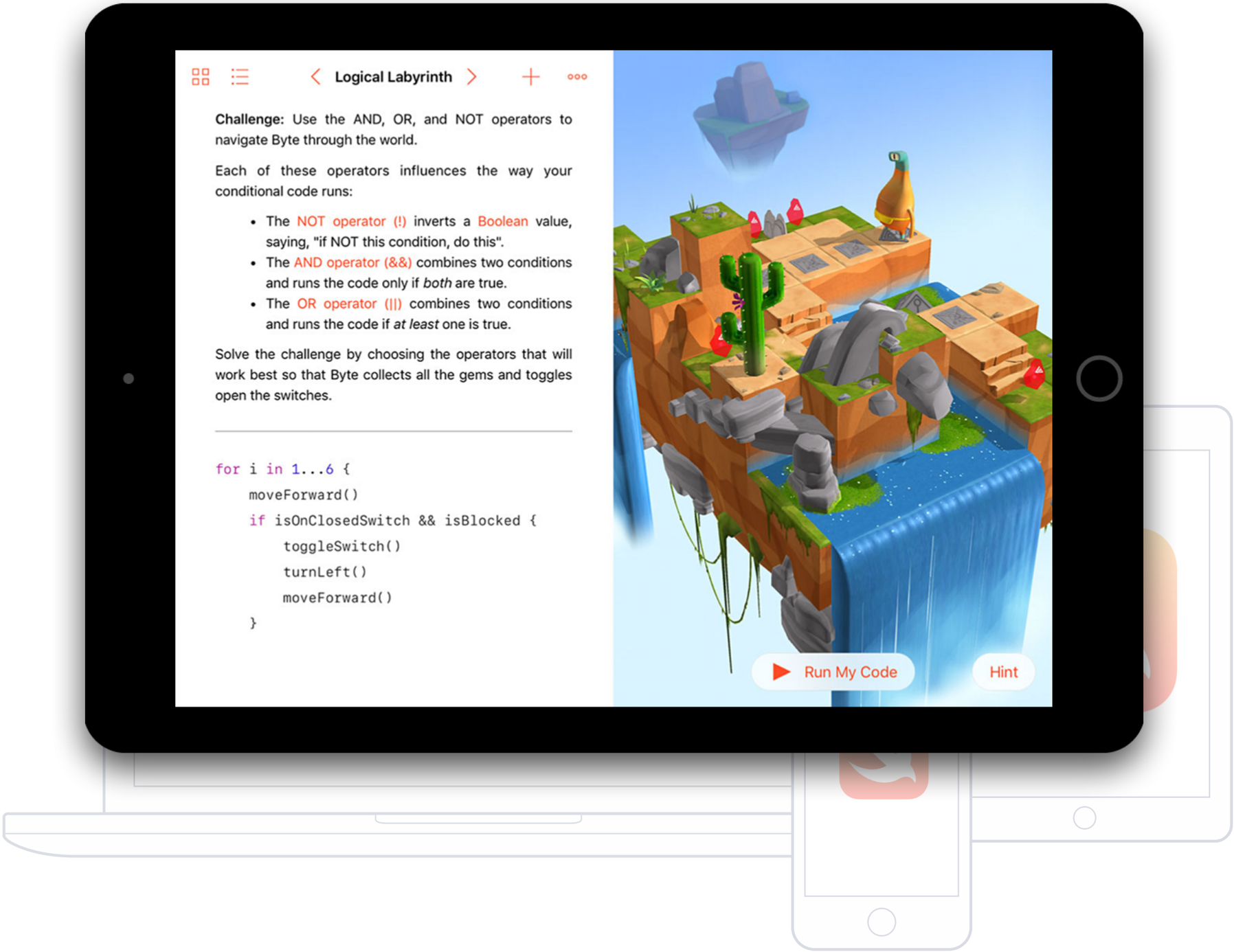
.NET

Platforms

Why will Swift be popular?

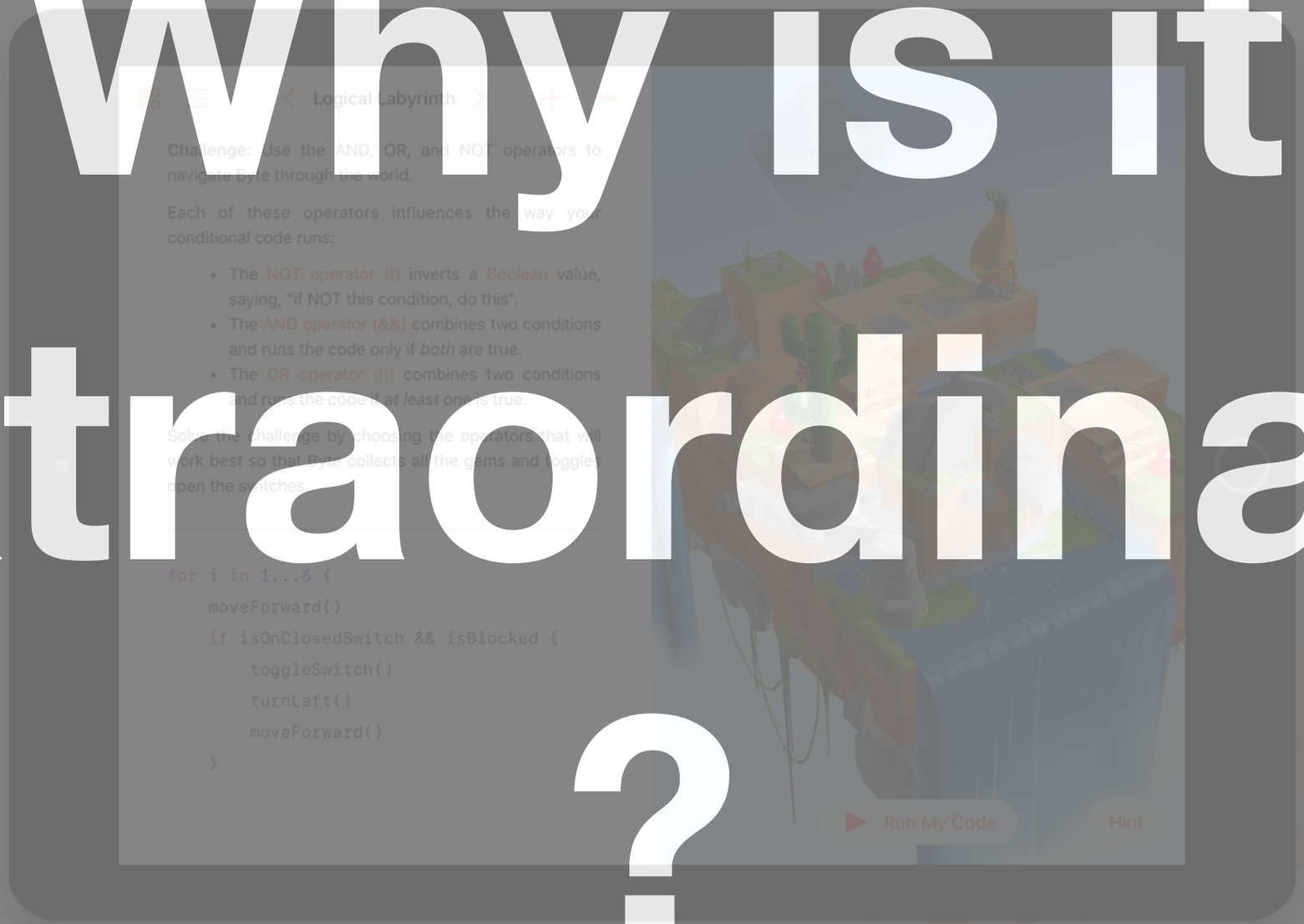
Why will Swift be popular?

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

Generics &
protocols with
associated types

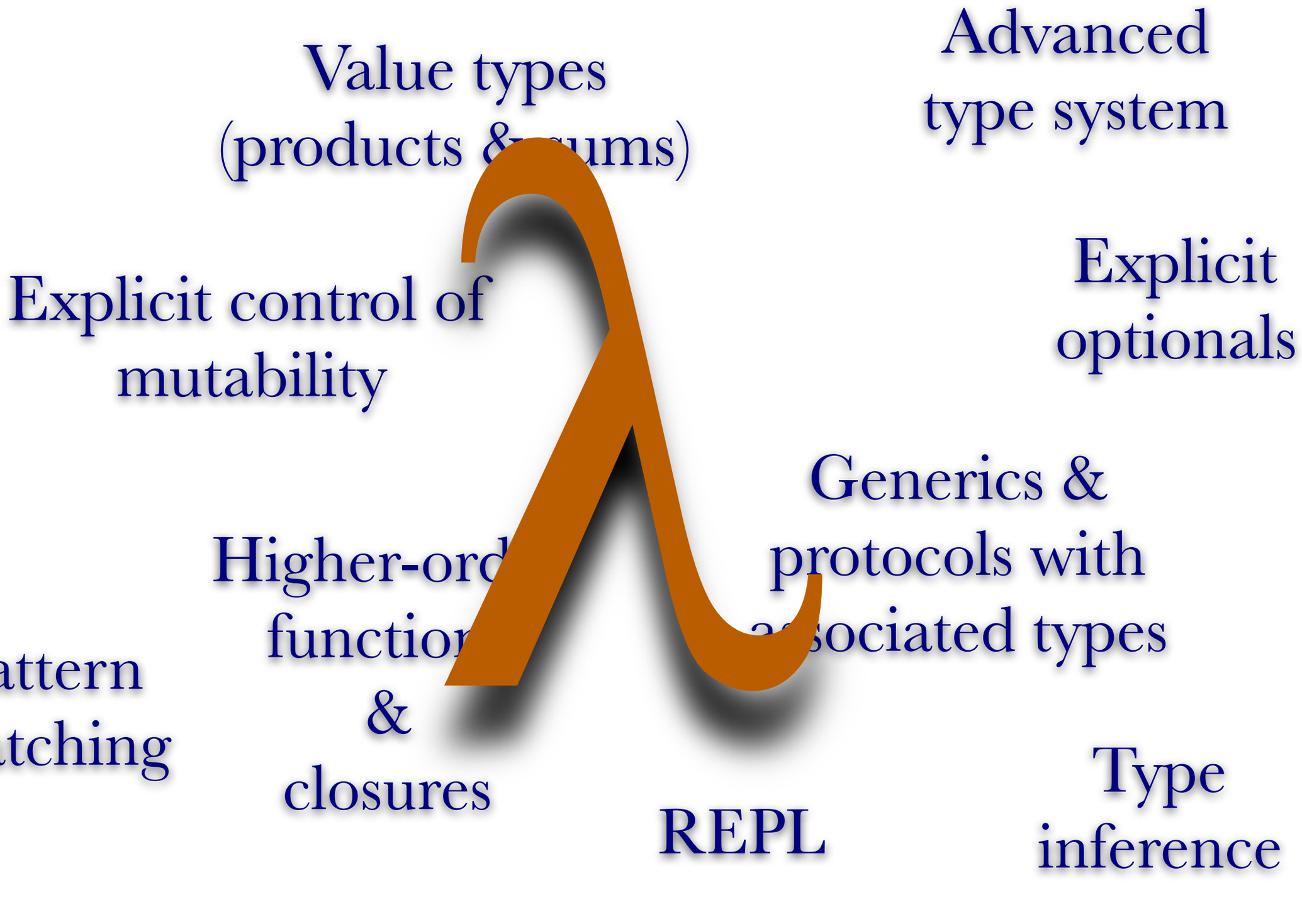
Higher-order
functions
&
closures

Pattern
matching

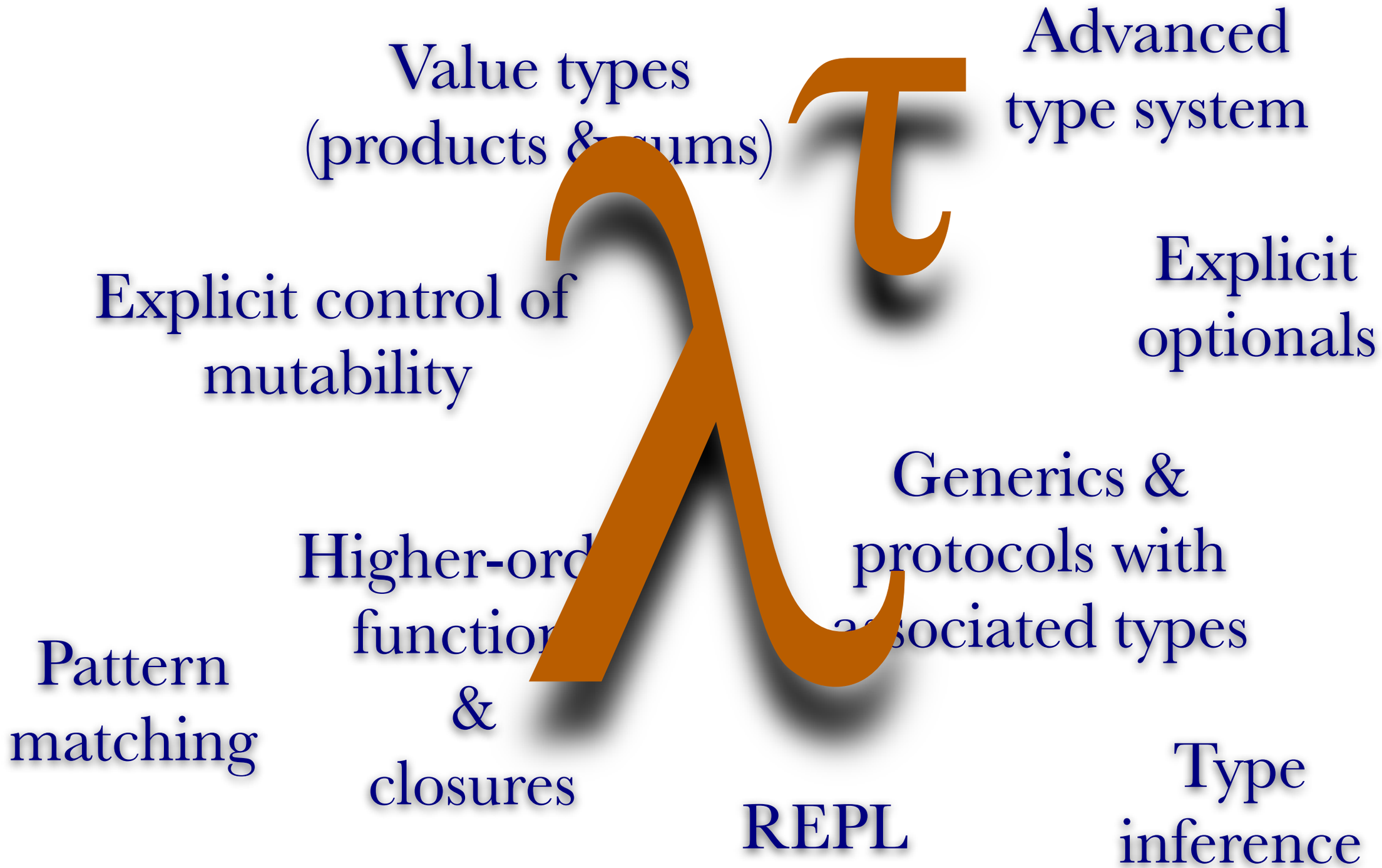
REPL

Type
inference

From Objective-C to Swift



From Objective-C to Swift



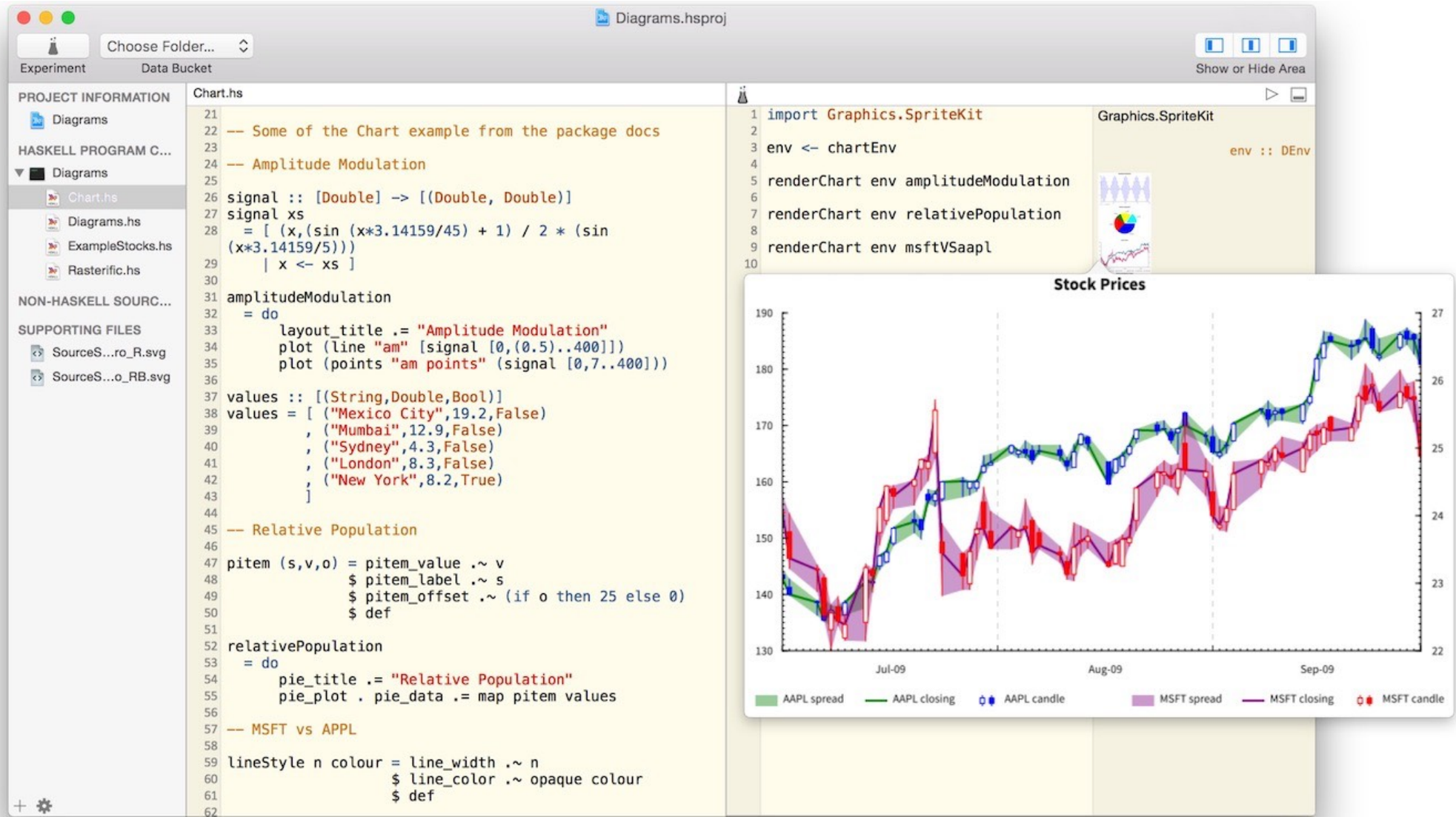
“Swift encourages typed functional programming in a mainstream language.”

Haskell for Mac

Adopting Swift right out of the gate

Haskell for Mac

Adopting Swift right out of the gate

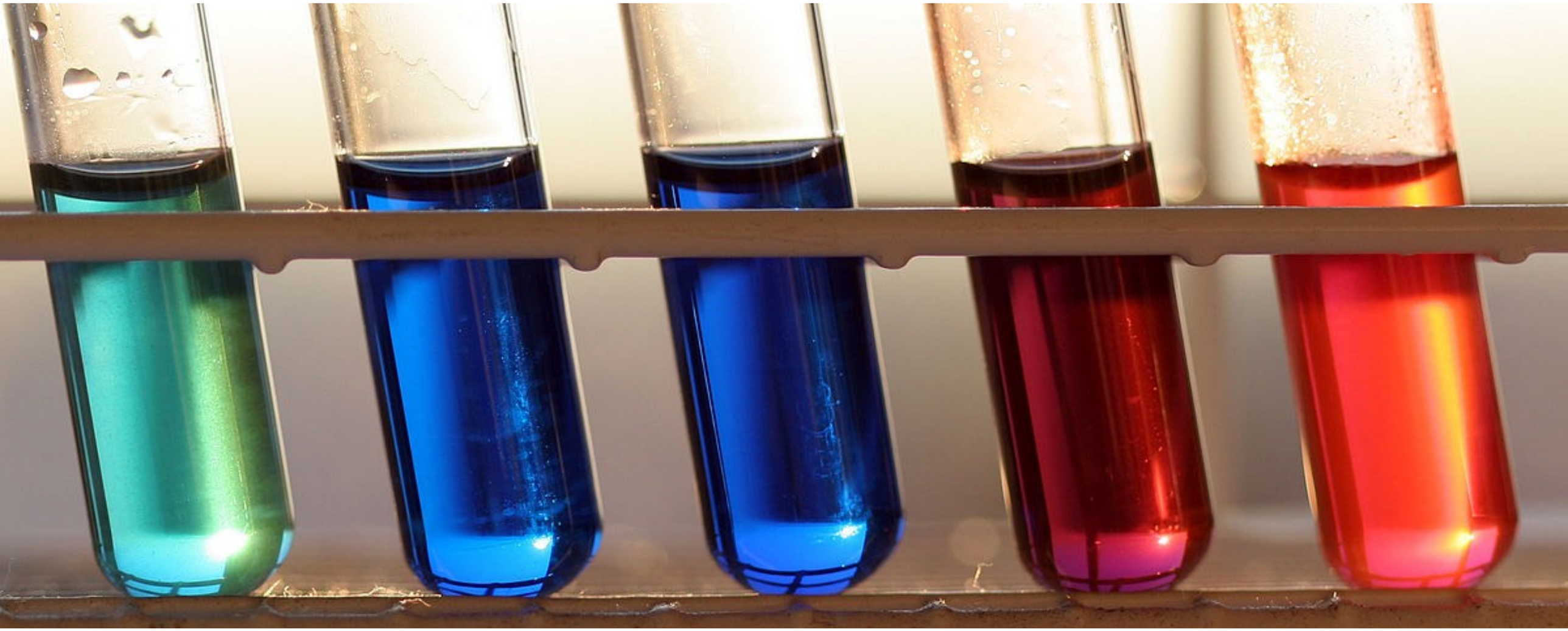


“Swift encourages typed functional programming in a mainstream language.”

Why Type Systems?

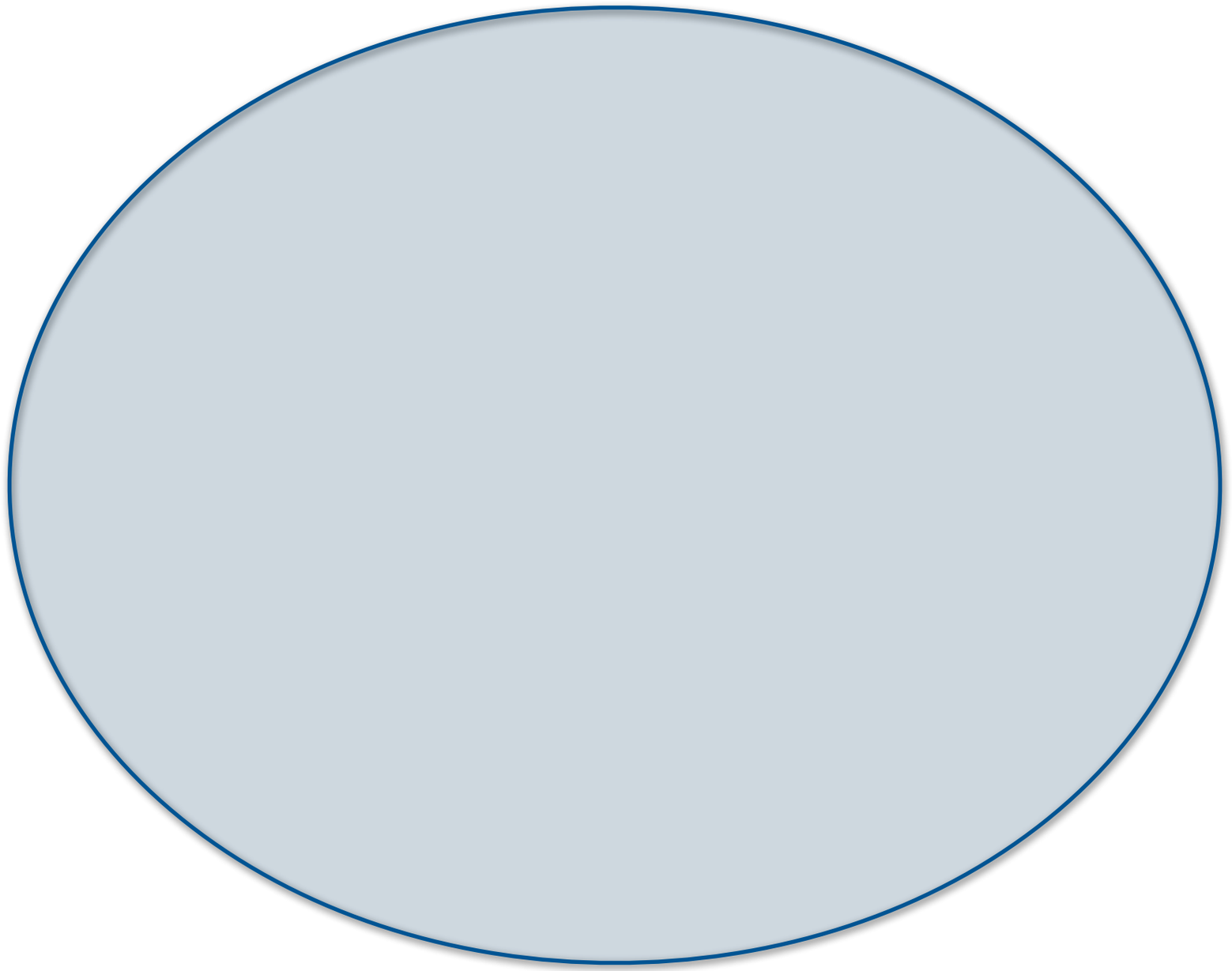
Catch bugs & prevent crashes

Less testing!

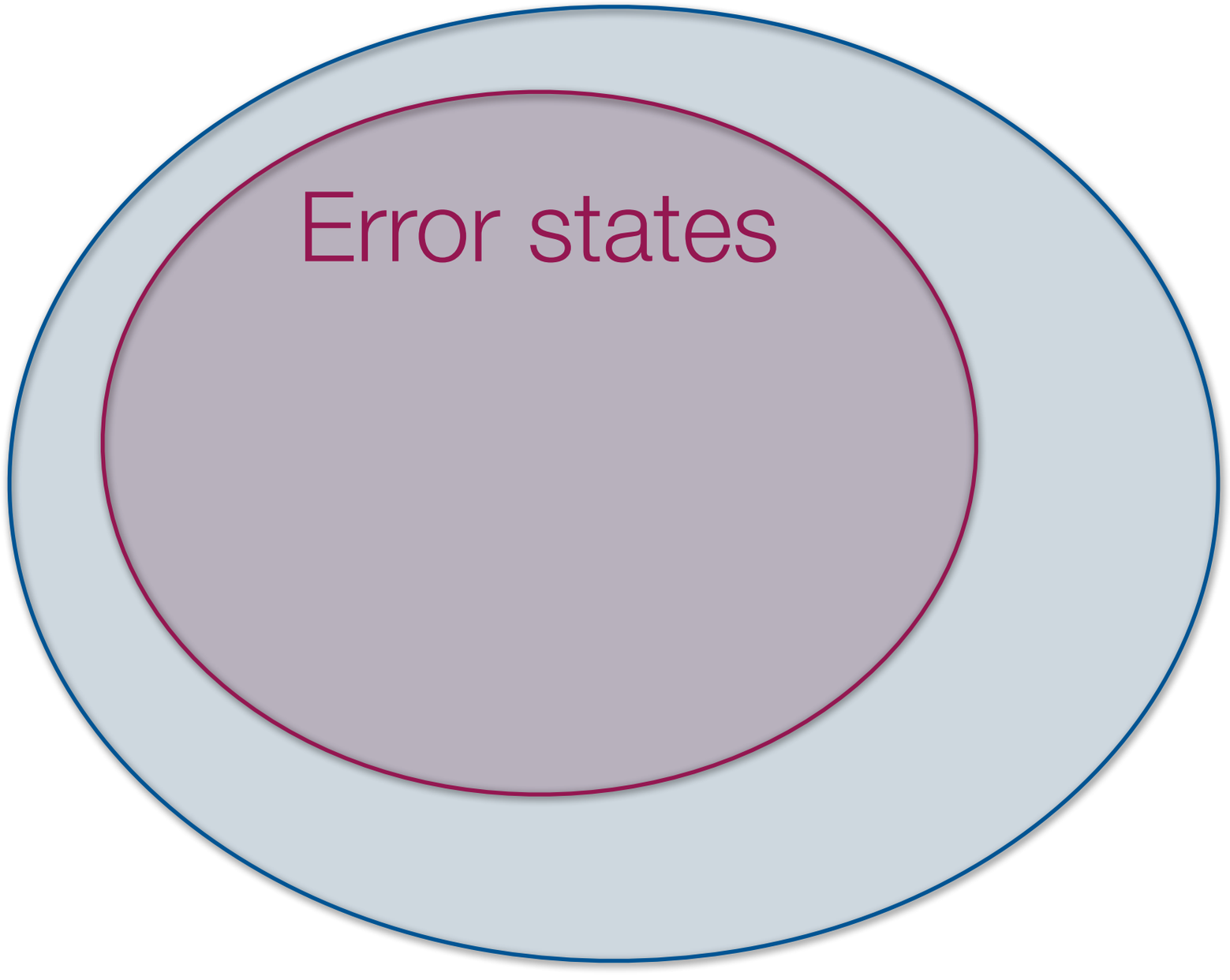


“What if you could gain the same confidence in correctness with fewer or simpler tests?”

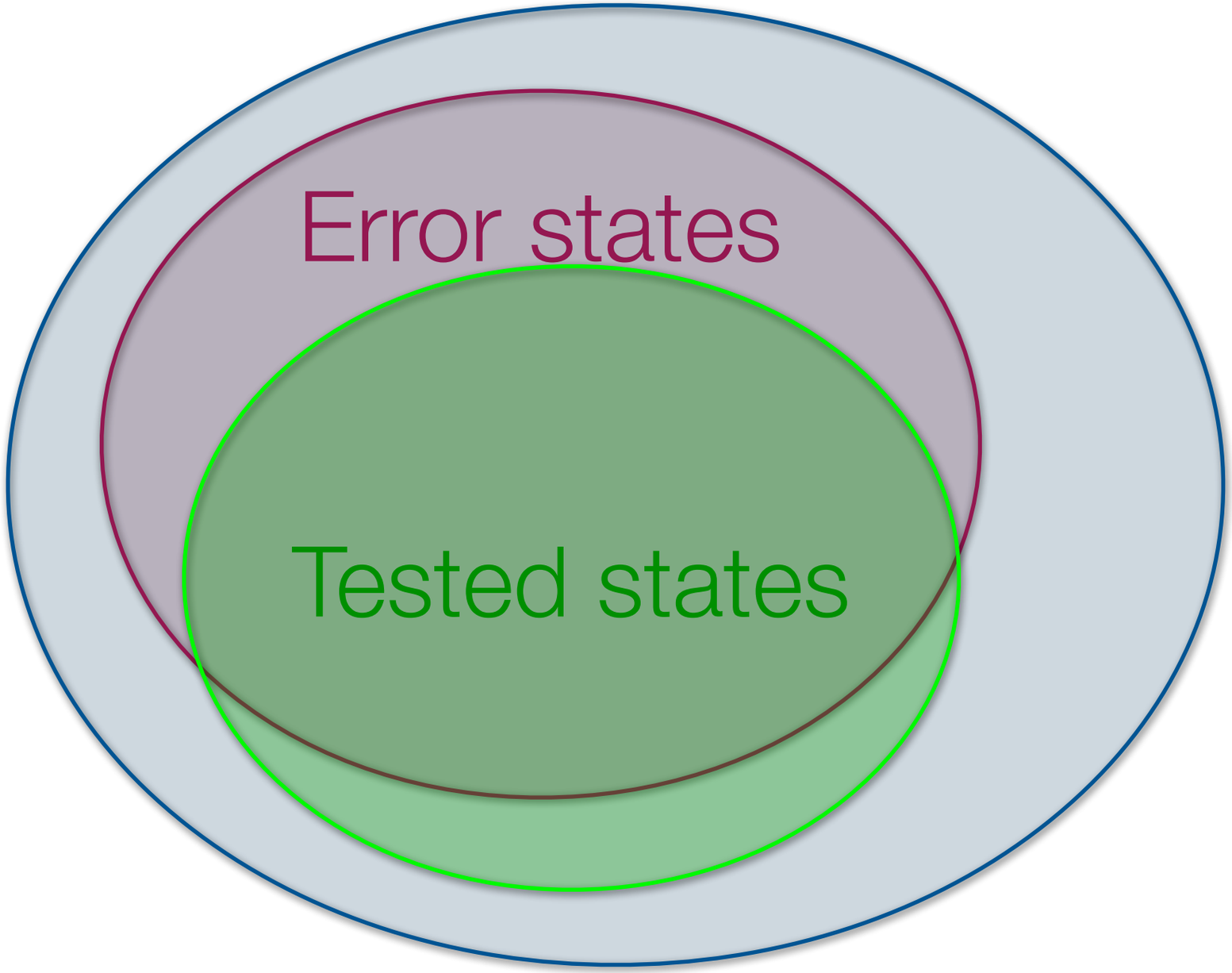
Possible states of your application



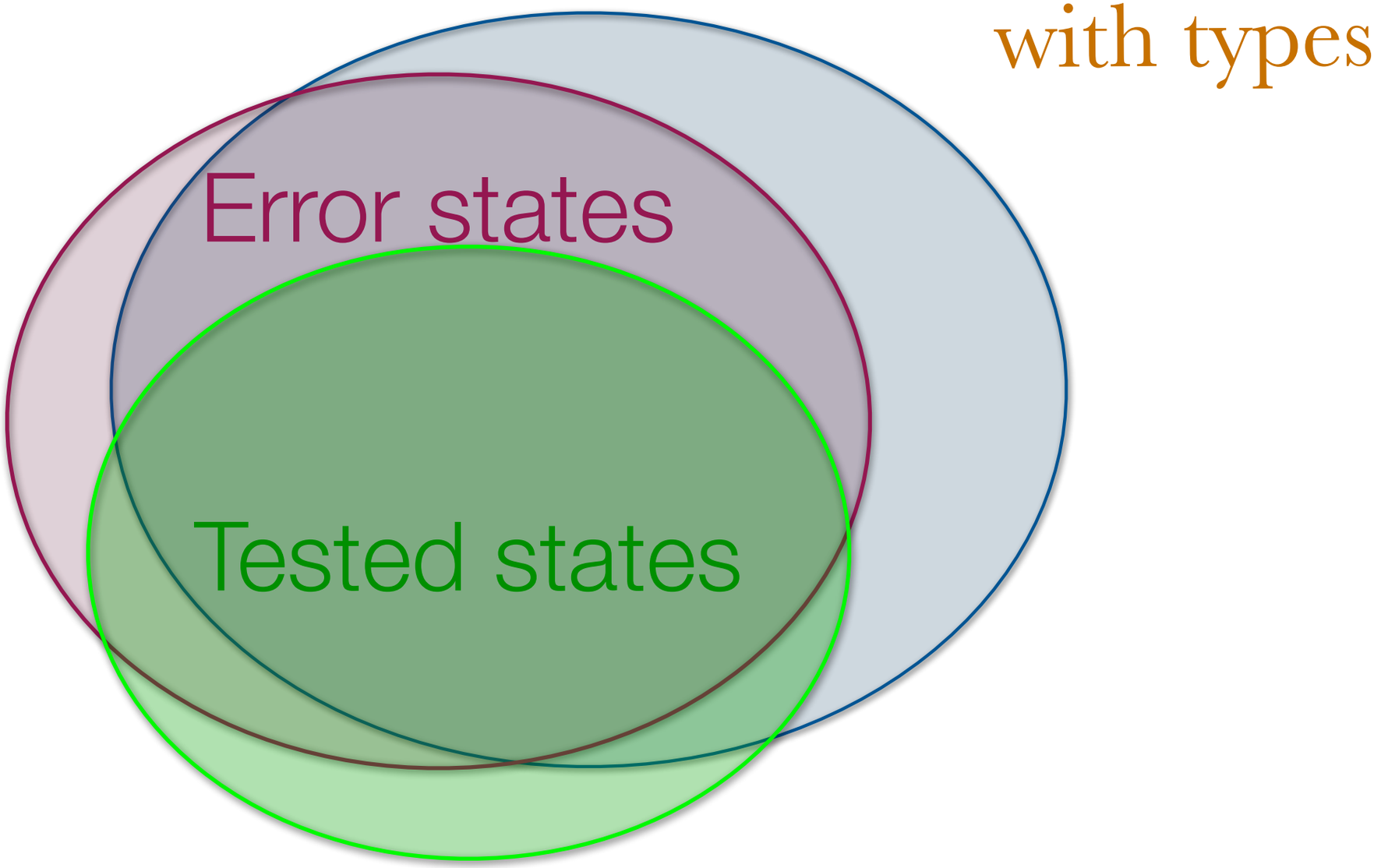
Possible states of your application



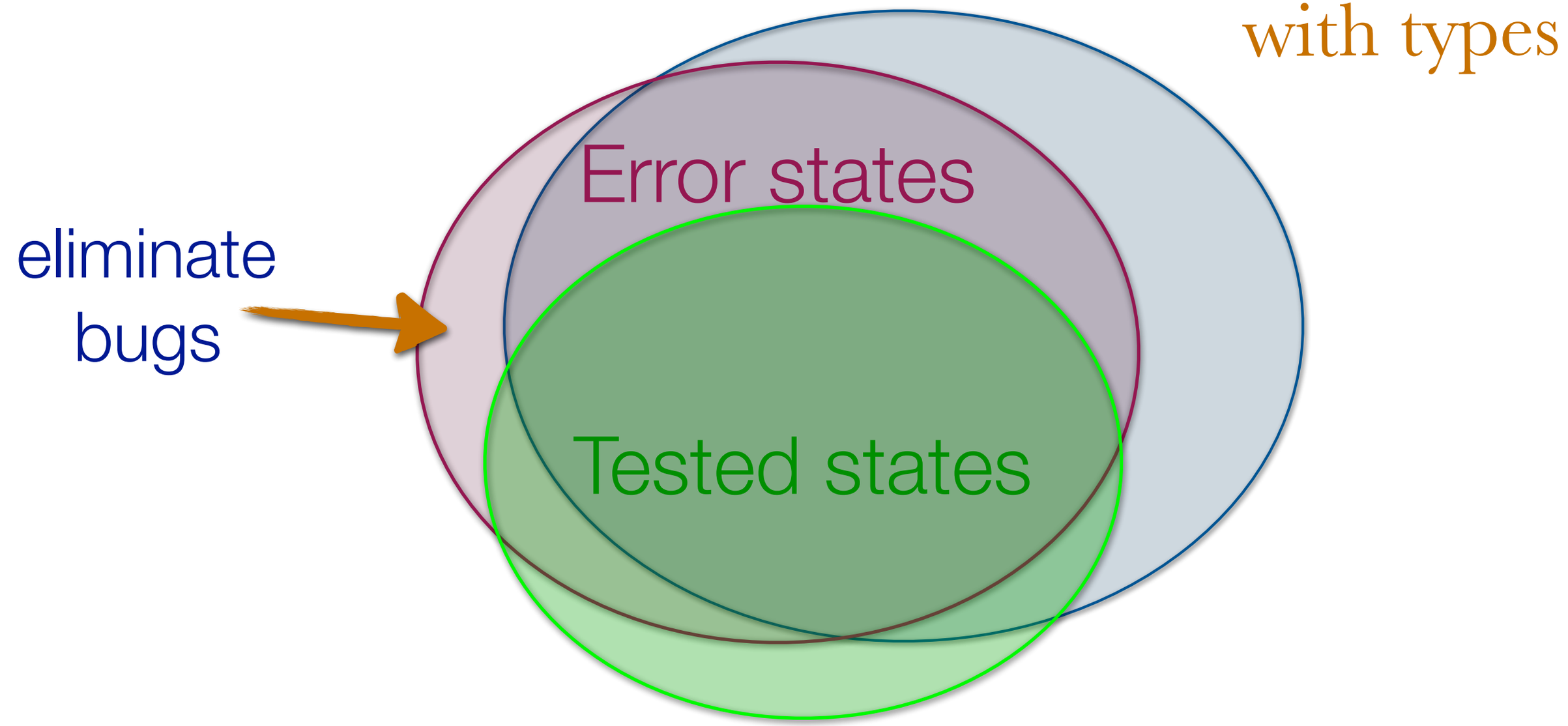
Possible states of your application



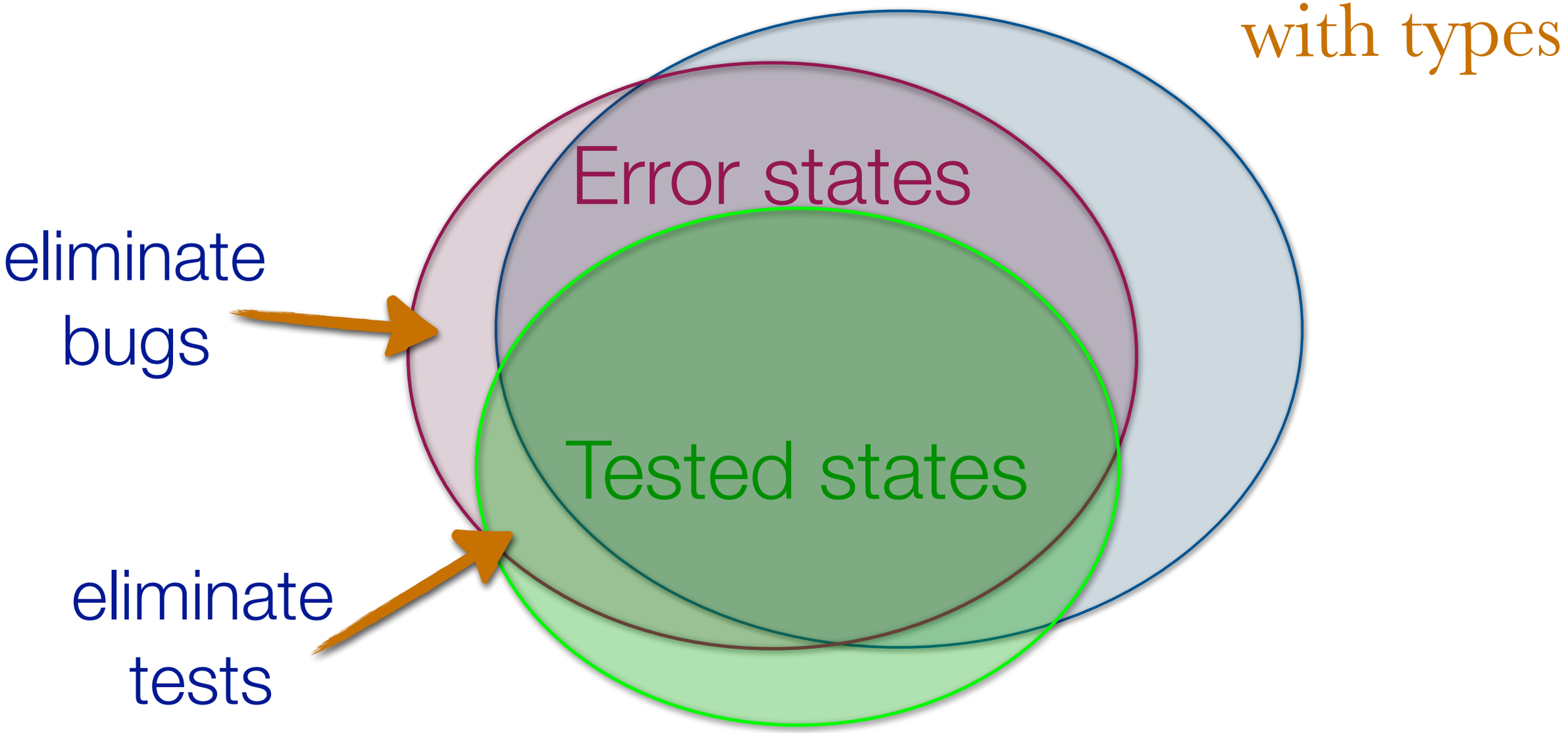
Possible states of your application



Possible states of your application

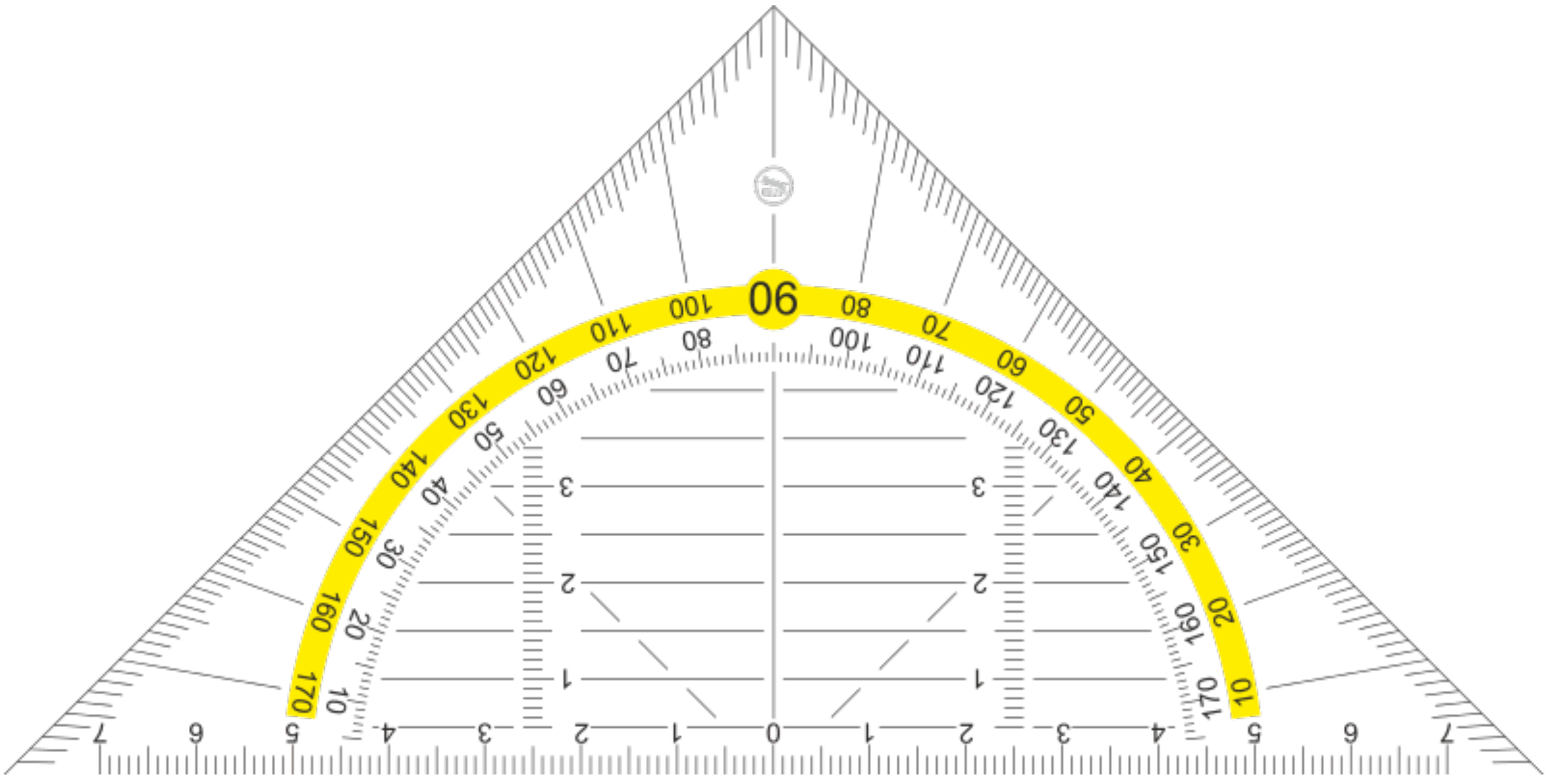


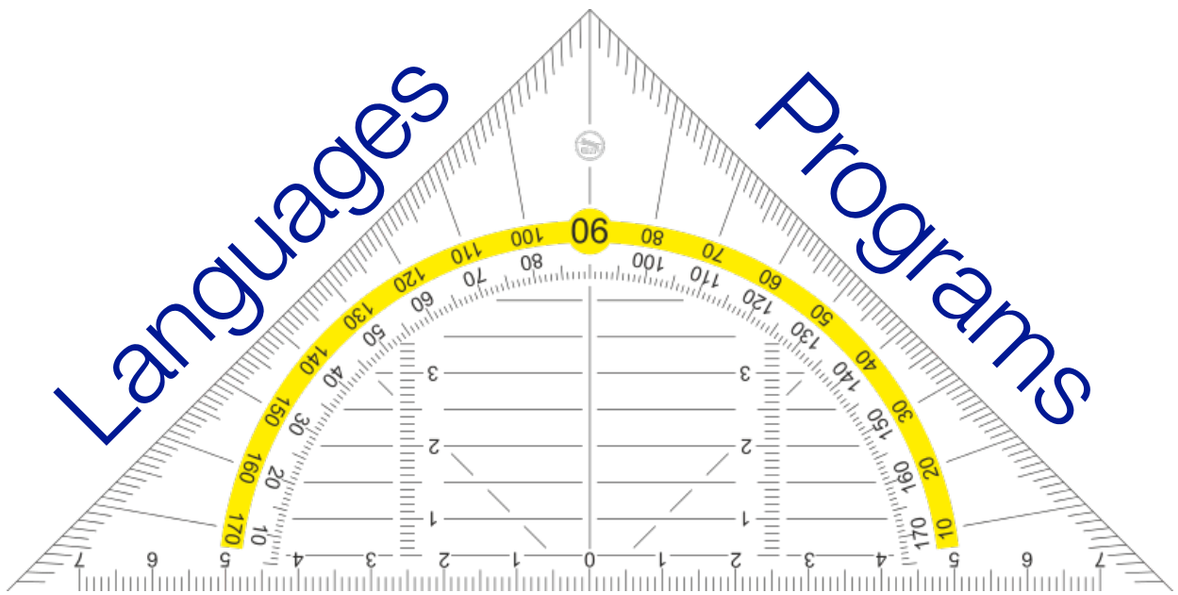
Possible states of your application



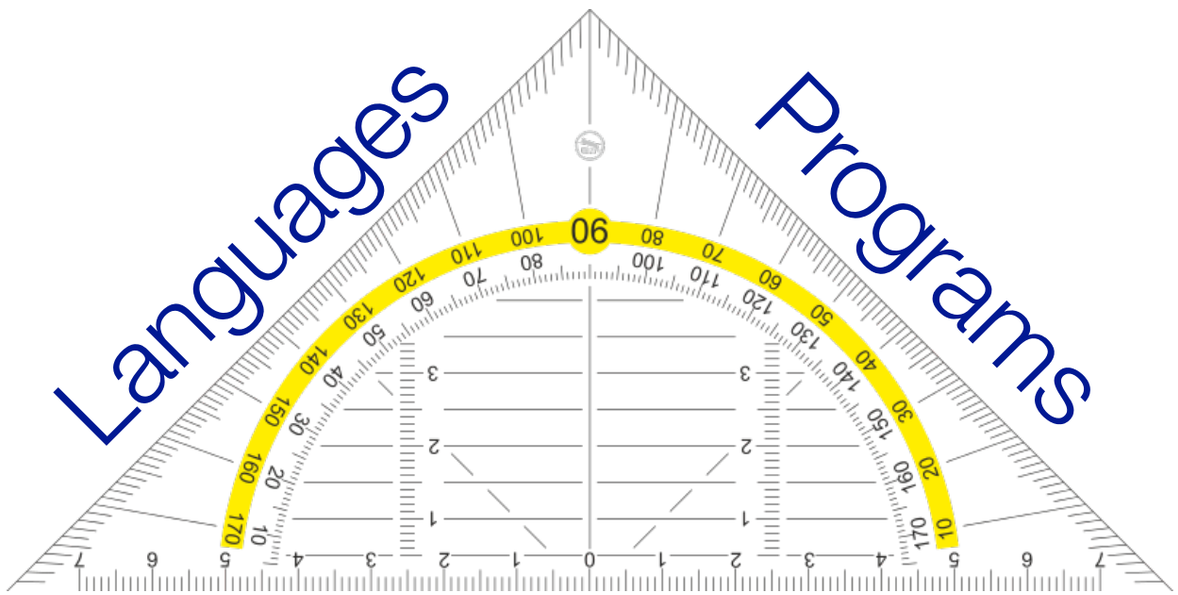
Are types restraints?

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."

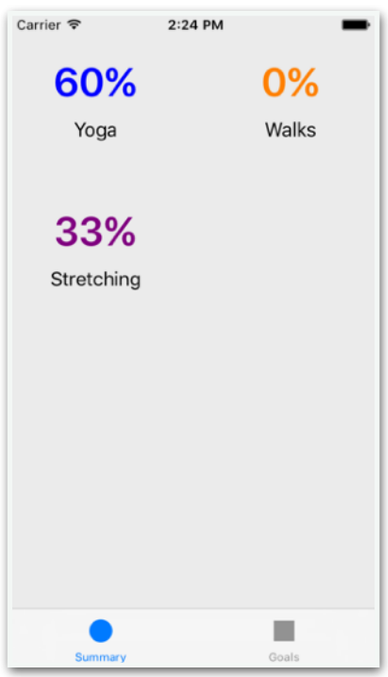
Value types

Immutable model & UI state machines

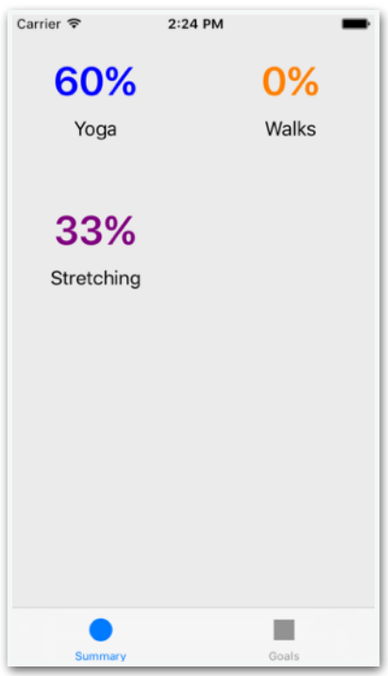
Protocols with associated types

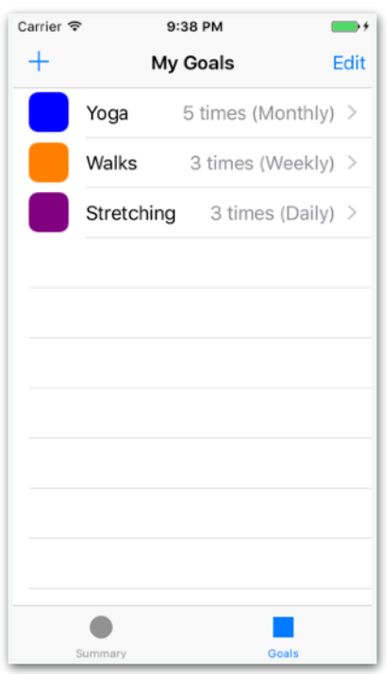
Generic change propagation

Goals.app

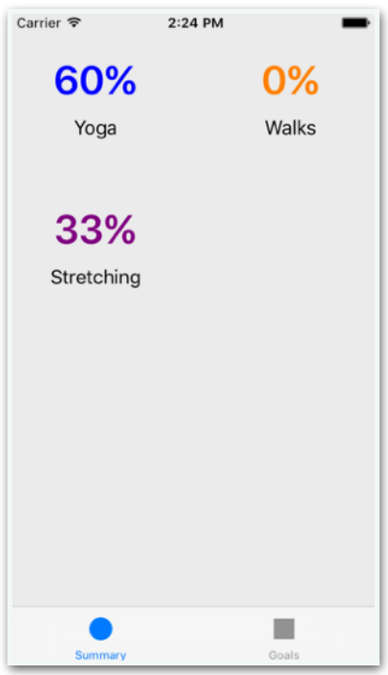


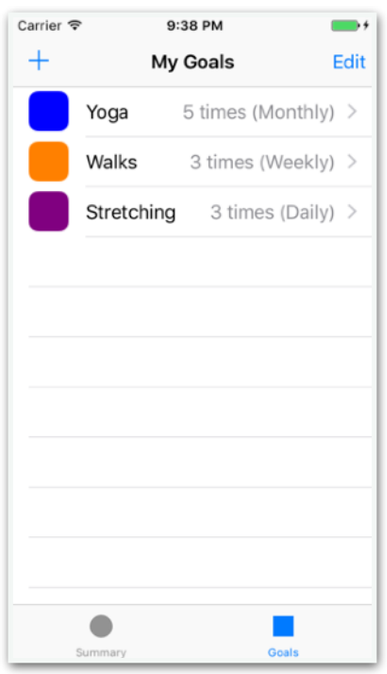
Goals.app

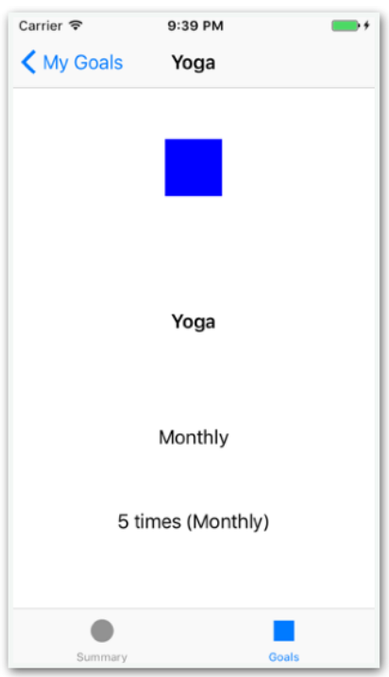

Carrier
2:24 PM
60%
Yoga
0%
Walks
33%
Stretching
Summary
Goals


Carrier
9:38 PM
My Goals
Edit
Yoga
5 times (Monthly)
Walks
3 times (Weekly)
Stretching
3 times (Daily)
Summary
Goals

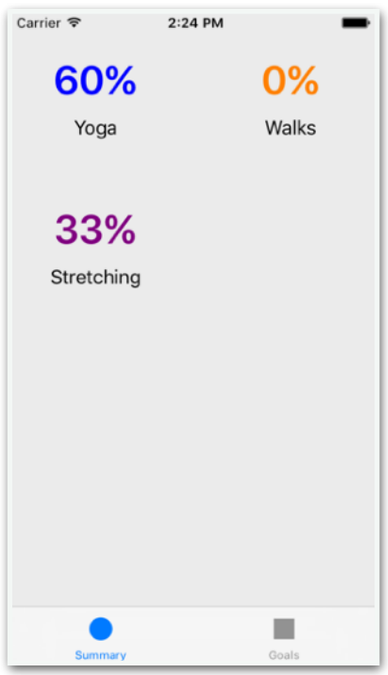
Goals.app

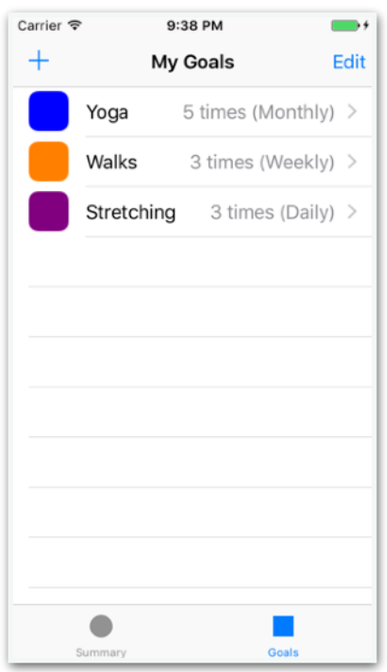

Carrier
2:24 PM
60%
Yoga
0%
Walks
33%
Stretching
Summary
Goals

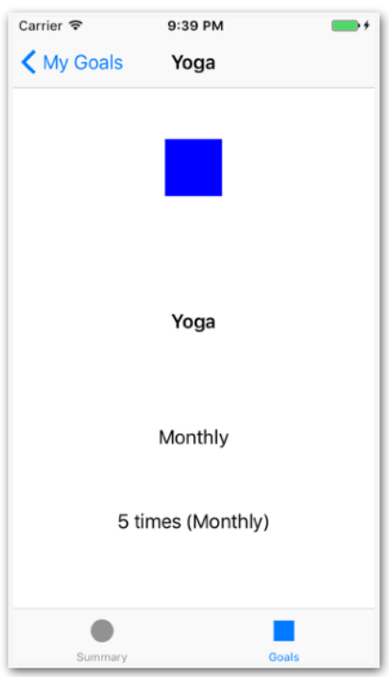

Carrier
9:38 PM
My Goals
Edit
Yoga
5 times (Monthly)
Walks
3 times (Weekly)
Stretching
3 times (Daily)
Summary
Goals

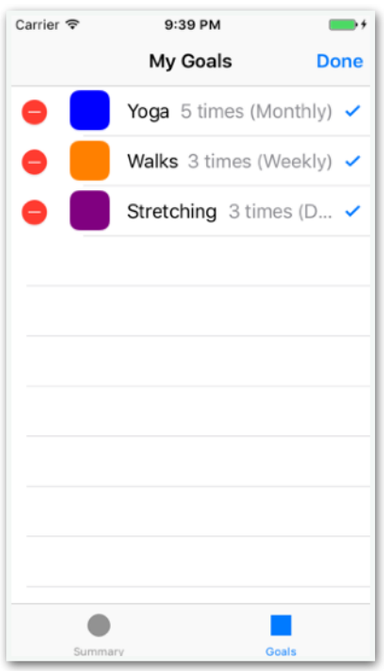

Carrier
9:39 PM
My Goals
Yoga
Yoga
Monthly
5 times (Monthly)
Summary
Goals

Goals.app

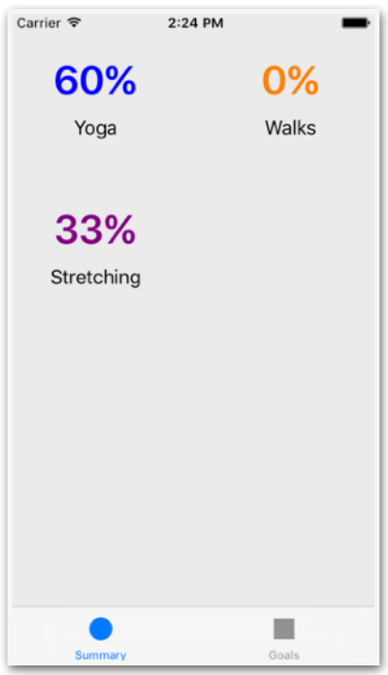

Carrier
2:24 PM
60%
Yoga
0%
Walks
33%
Stretching
Summary
Goals

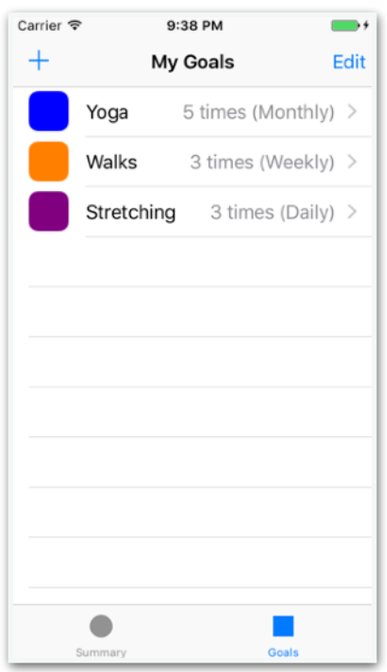

Carrier
9:38 PM
My Goals
Edit
Yoga
5 times (Monthly)
Walks
3 times (Weekly)
Stretching
3 times (Daily)
Summary
Goals

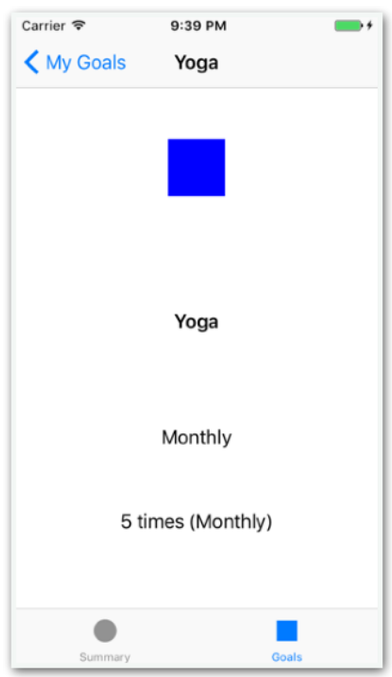

Carrier
9:39 PM
My Goals
Yoga
Yoga
Monthly
5 times (Monthly)
Summary
Goals

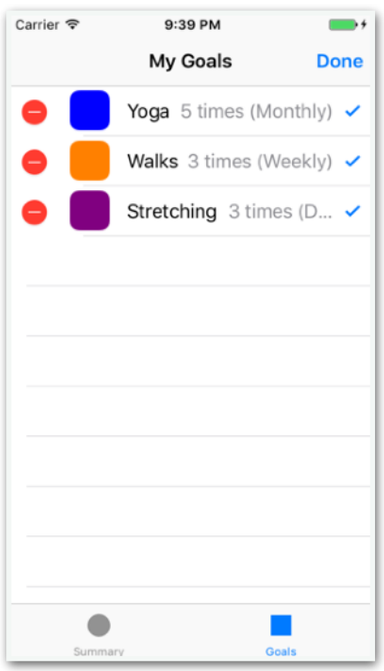

Carrier
9:39 PM
My Goals
Done
Yoga 5 times (Monthly)
Walks 3 times (Weekly)
Stretching 3 times (D...
Summary
Goals

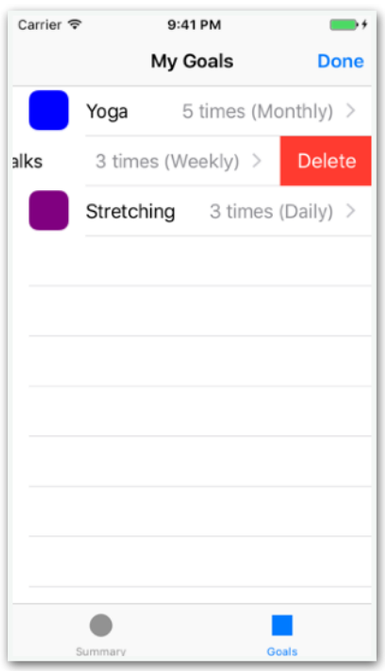
Goals.app


Carrier
2:24 PM
60%
Yoga
0%
Walks
33%
Stretching
Summary
Goals


Carrier
9:38 PM
My Goals
Edit
Yoga
5 times (Monthly)
Walks
3 times (Weekly)
Stretching
3 times (Daily)
Summary
Goals


Carrier
9:39 PM
My Goals
Yoga
Yoga
Monthly
5 times (Monthly)
Summary
Goals


Carrier
9:39 PM
My Goals
Done
Yoga 5 times (Monthly)
Walks 3 times (Weekly)
Stretching 3 times (D...
Summary
Goals


Carrier
9:41 PM
My Goals
Done
Yoga
5 times (Monthly)
alks
3 times (Weekly)
Delete
Stretching
3 times (Daily)
Summary
Goals

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

struct

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

generic type parameter

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
```

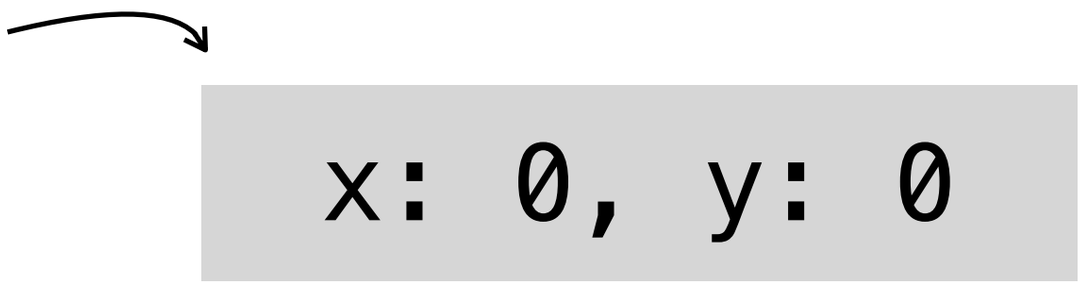
```
var w: RefPnt
```

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

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

Reference types versus value types

```
var v: RefPnt
var w: RefPnt
```




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

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

Reference types versus value types

```
var v: RefPnt
```

```
var w: RefPnt
```

```
x: 10, y: 0
```

```
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
```

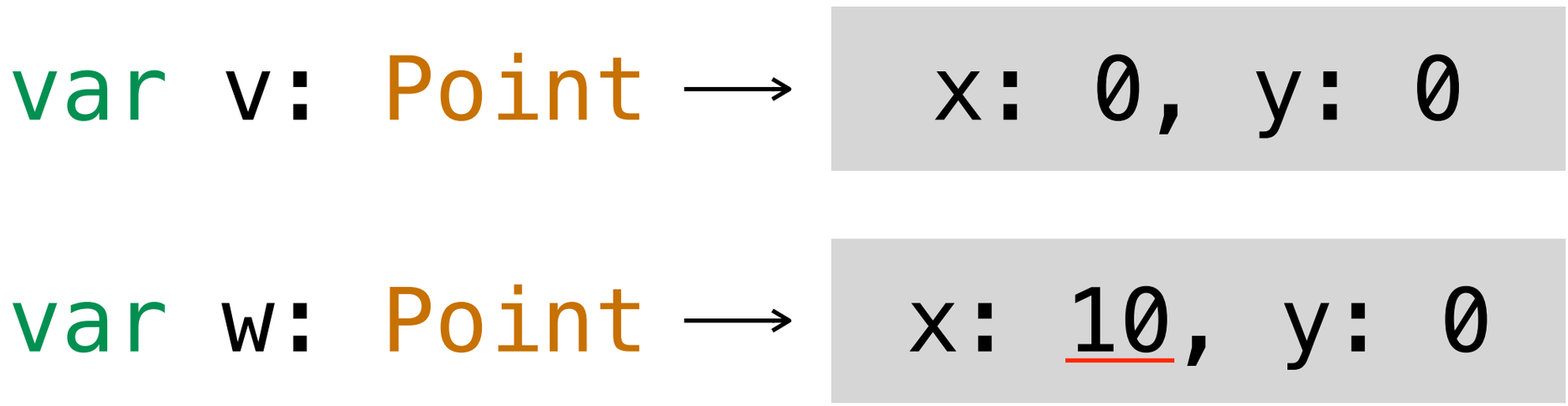
Reference types versus value types

```
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



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
```

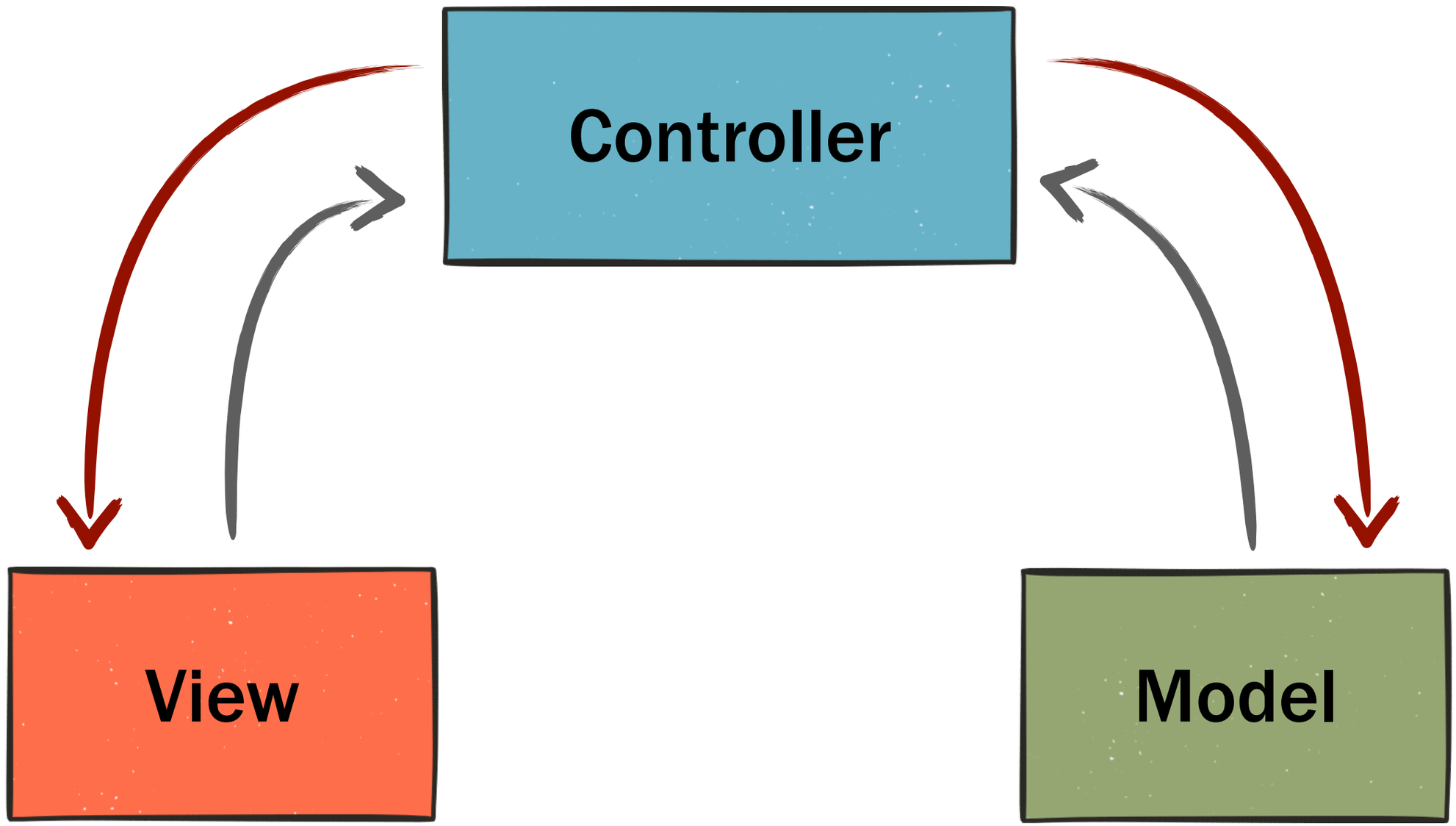
Localise change

Facilitate local reasoning

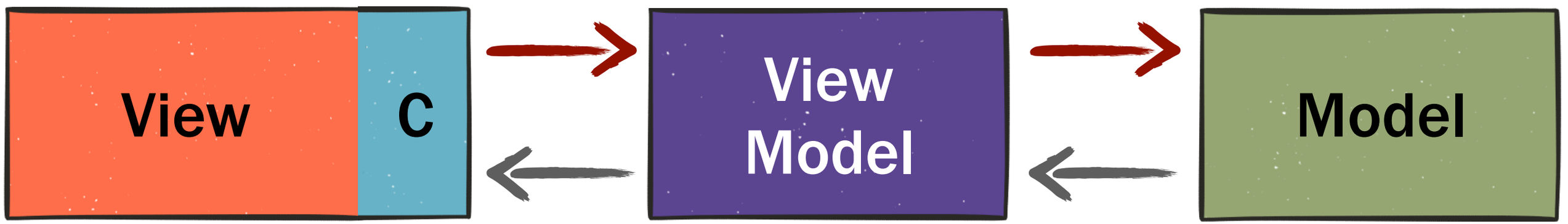
Value Types

Immutable Model

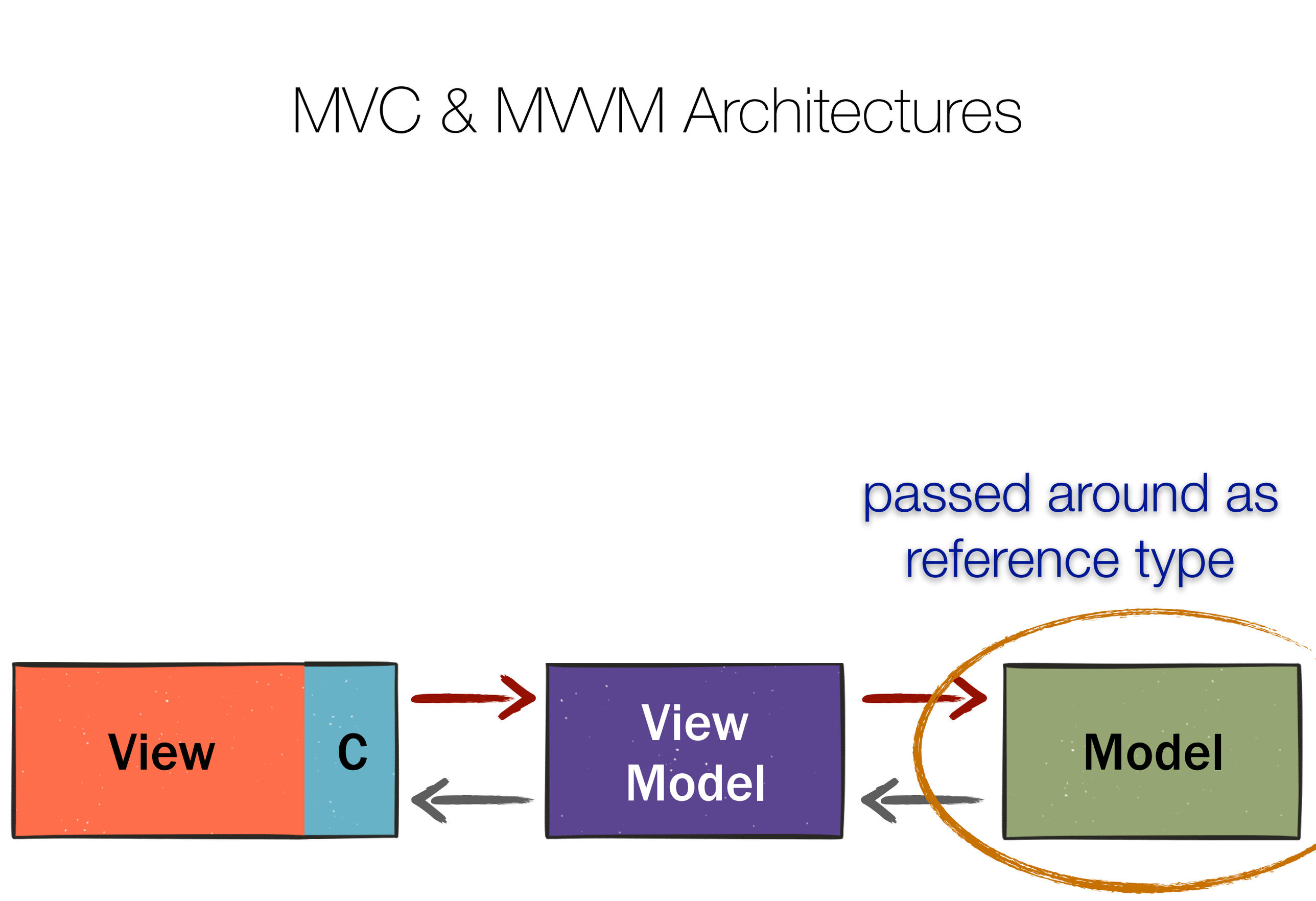
MVC & MVVM Architectures



MVC & MVVM Architectures



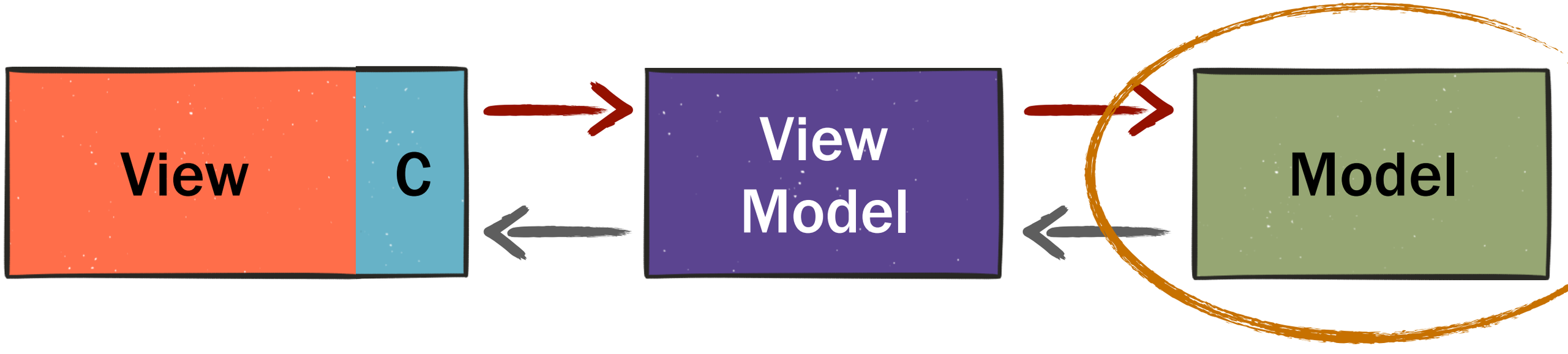
MVC & MVVM Architectures



MVC & MVVM Architectures

× Accidental changes

passed around as reference type

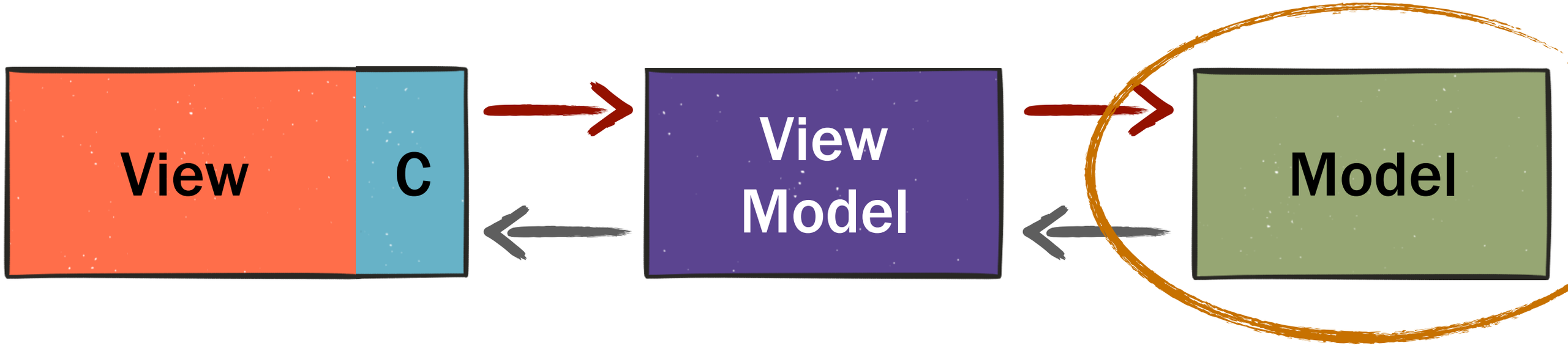


MVC & MVVM Architectures

× 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?)
```

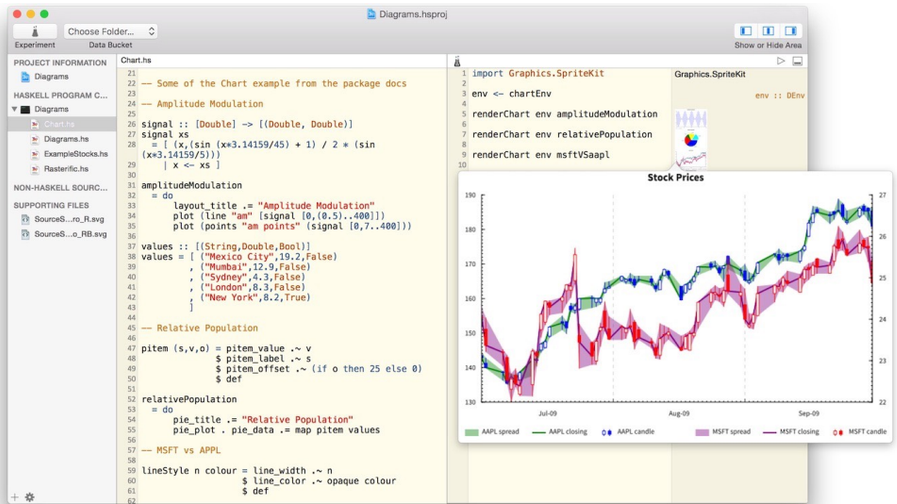
Immutable Goals

```
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!

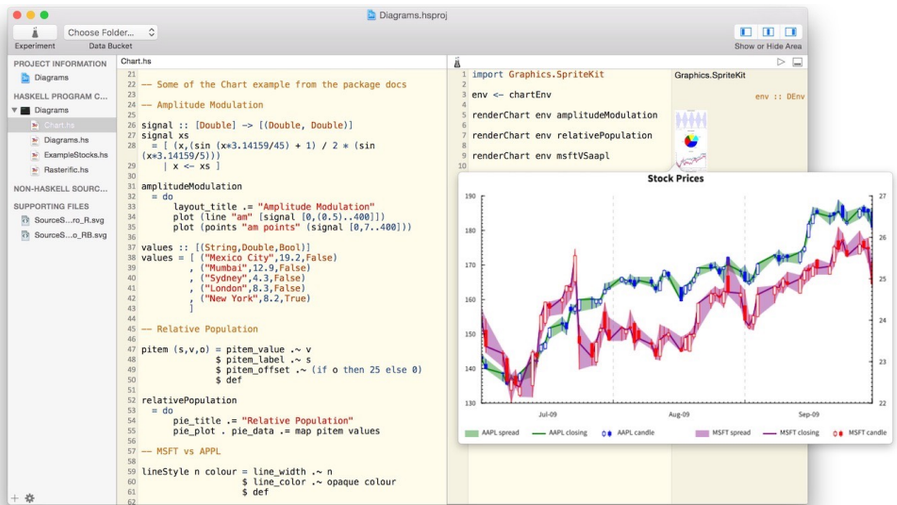


Immutable Models Do Scale!

Haskell for Mac

Uses an immutable model

Adopts Cocoa Document Architecture



Immutable Models Do Scale!

Haskell for Mac

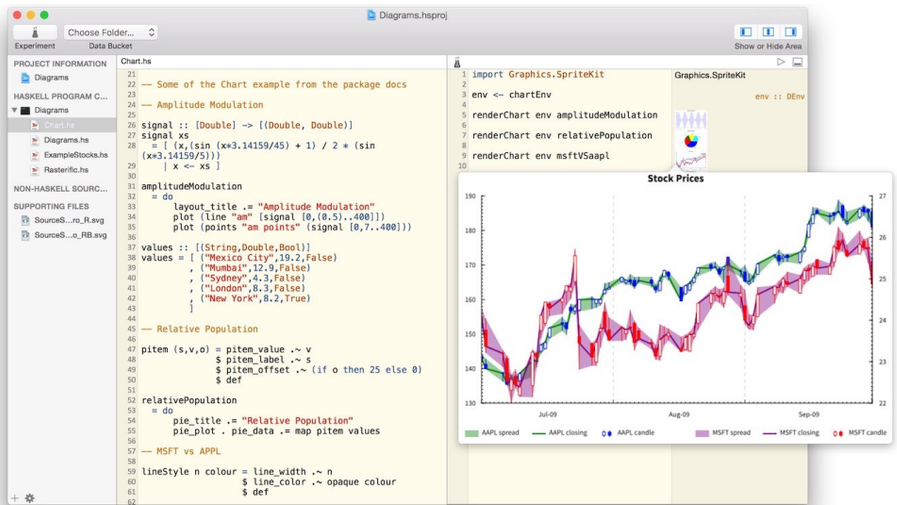
Uses an immutable model

Adopts Cocoa Document Architecture

“Functional Programming in a Stateful World”

YOW! Lambda Jam 2015

(on speakerdeck.com)



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

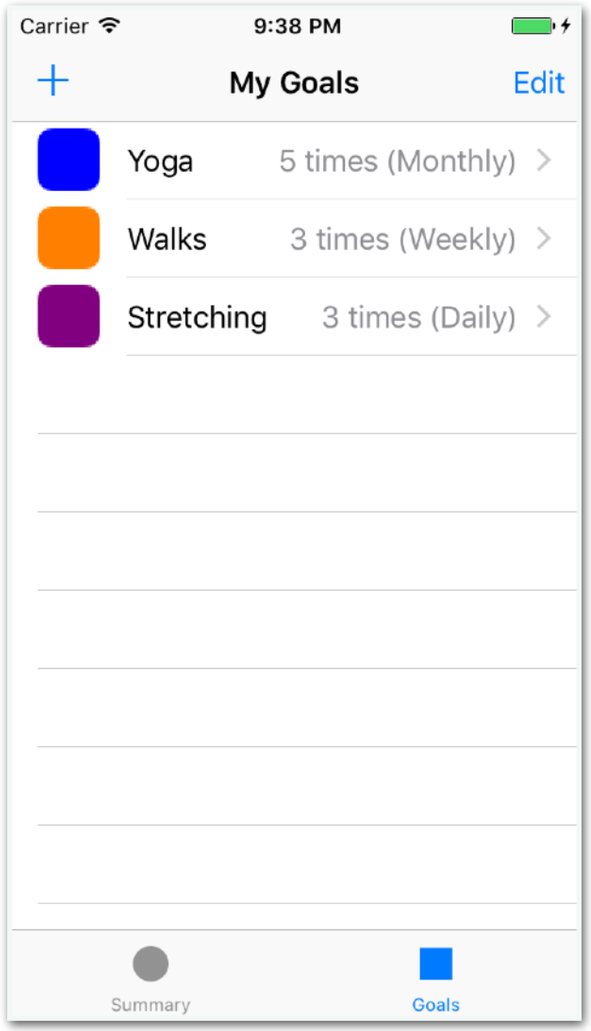
Replaces integration tests

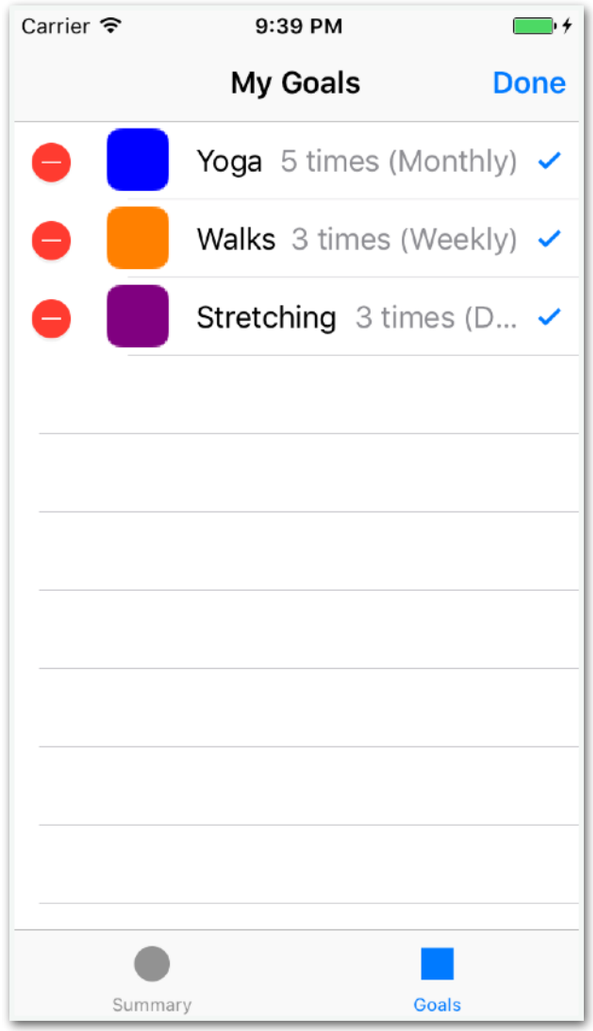
✓ Avoid races on concurrent threads

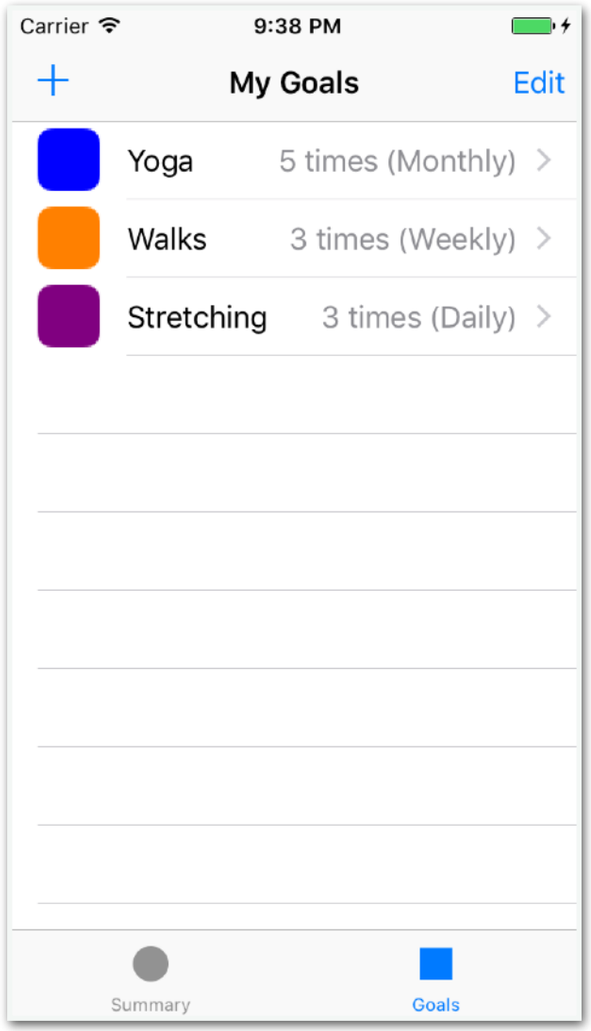
Often not properly tested

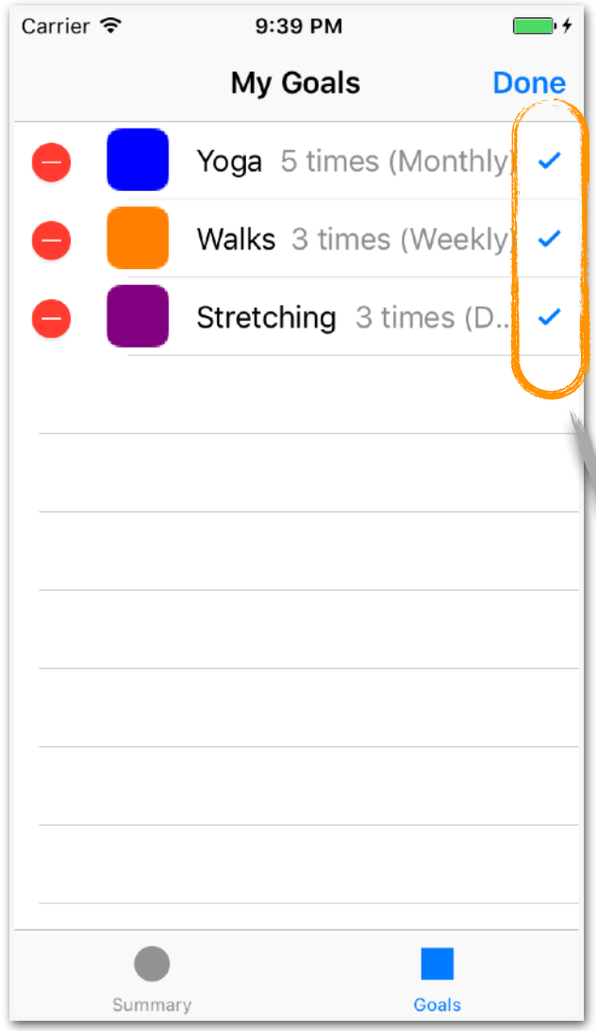
Enums with Associated Values

UI State Machines

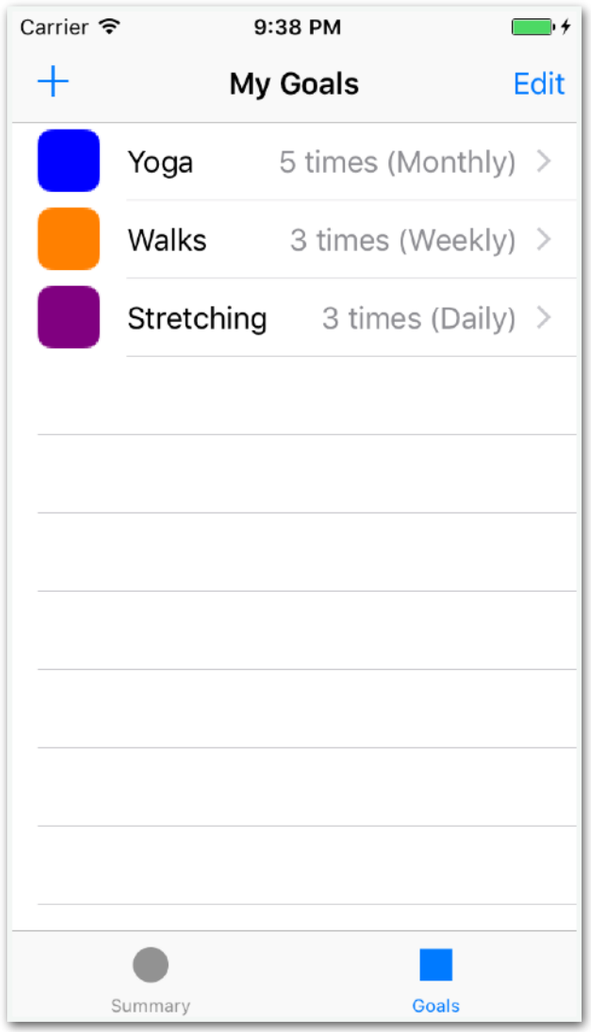

Carrier
9:38 PM
My Goals
Edit
Yoga
5 times (Monthly)
Walks
3 times (Weekly)
Stretching
3 times (Daily)
Summary
Goals

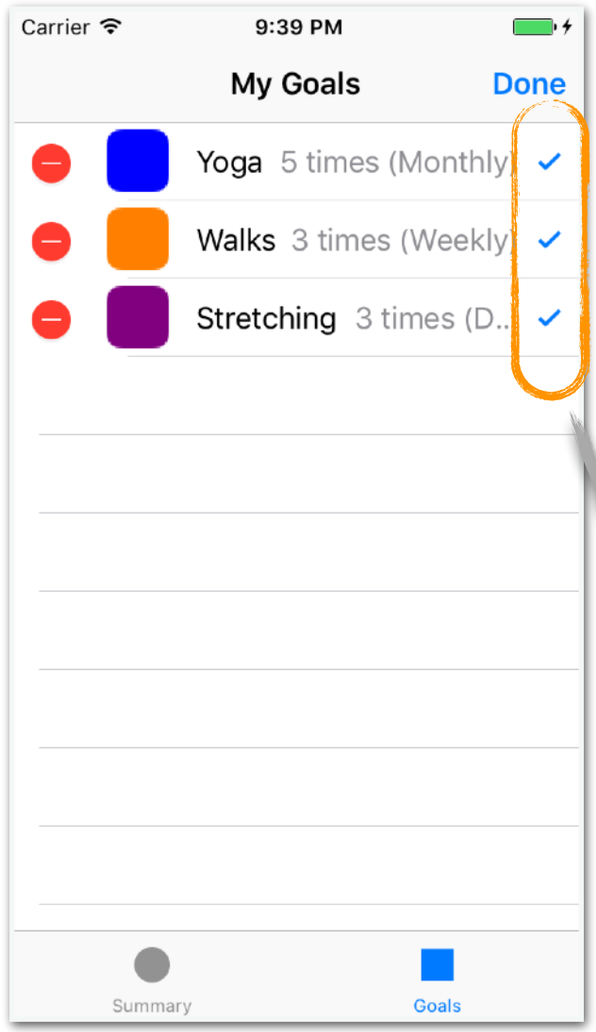

Carrier
9:39 PM
My Goals
Done
Yoga
5 times (Monthly)
Walks
3 times (Weekly)
Stretching
3 times (D...
Summary
Goals





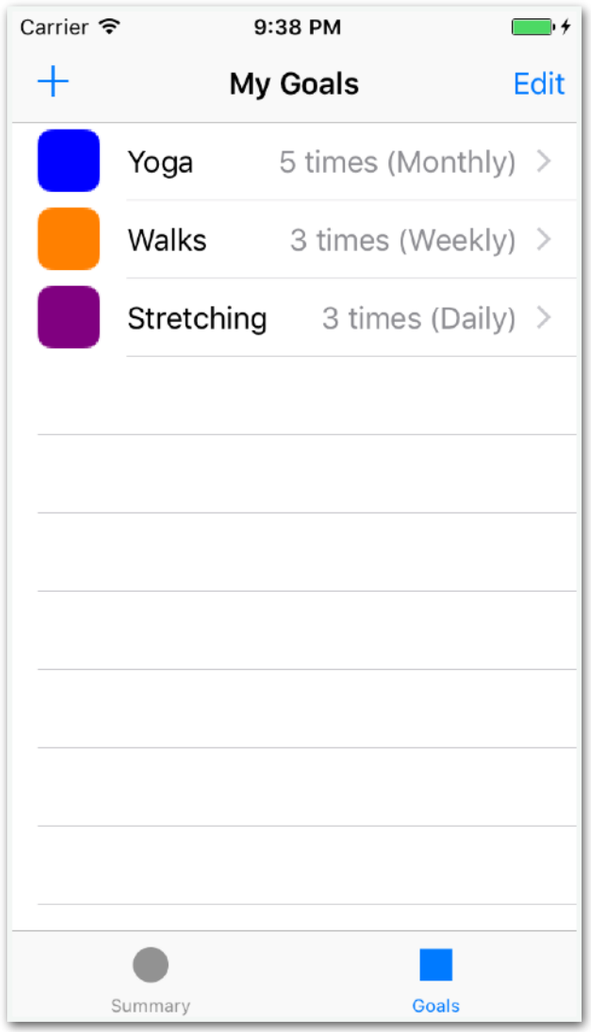
Is goal active?

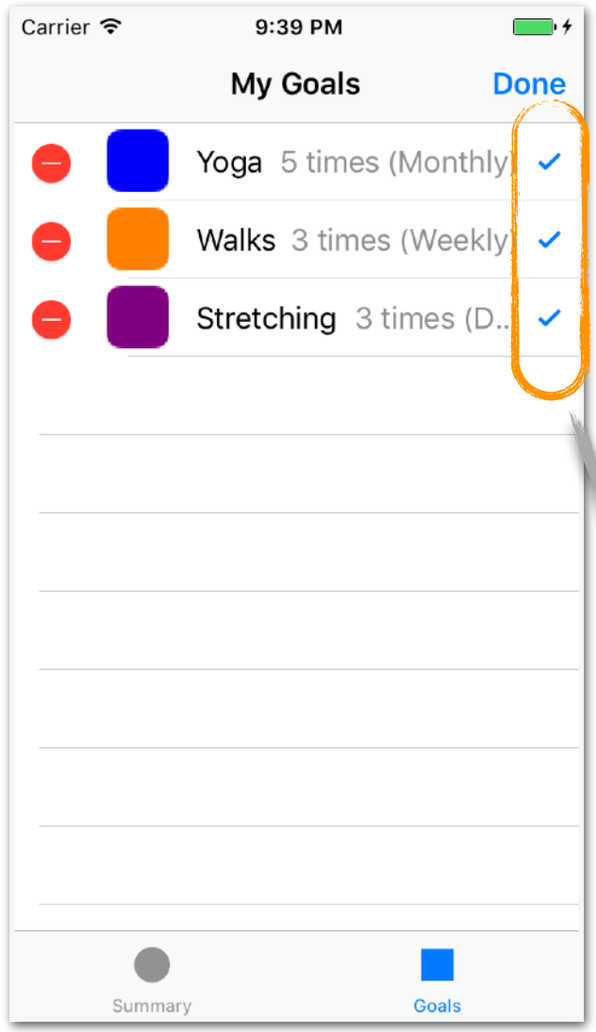




Is goal active?

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

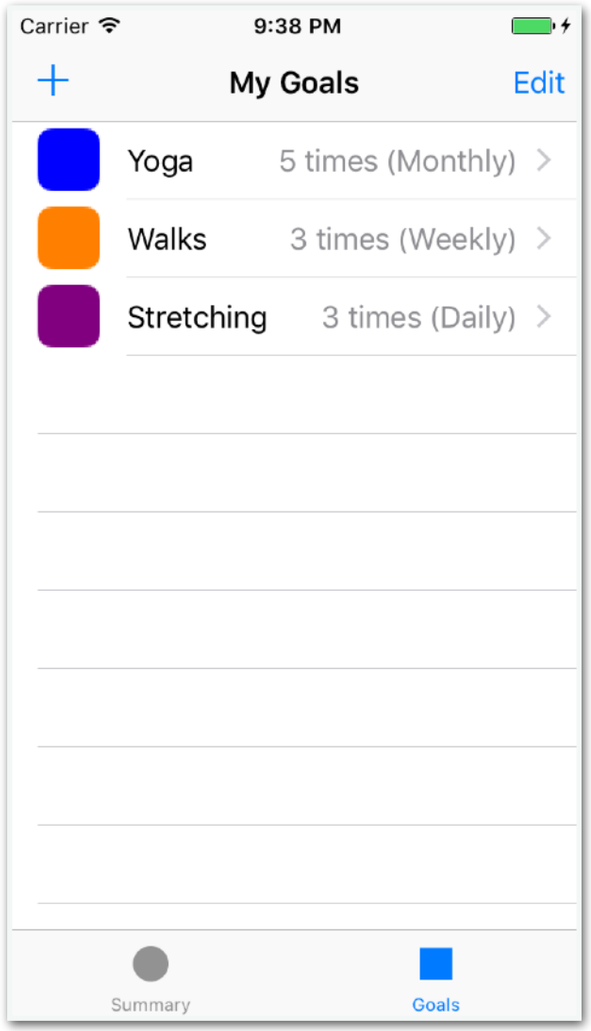


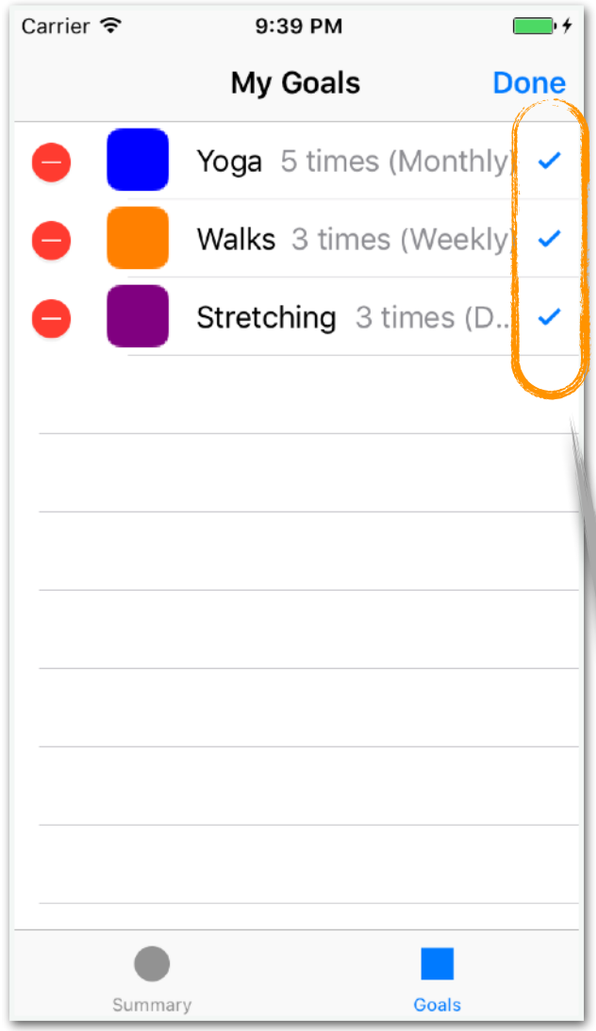


Is goal active?

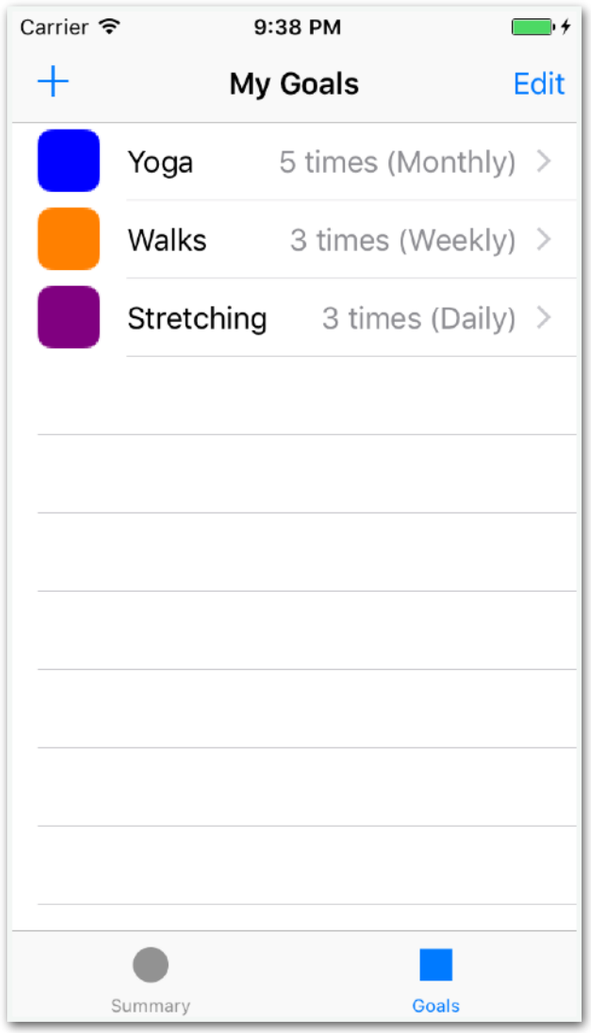
```
typealias GoalProgress = (goal: Goal, count: Int?)
typealias Goals = [GoalProgress]
```

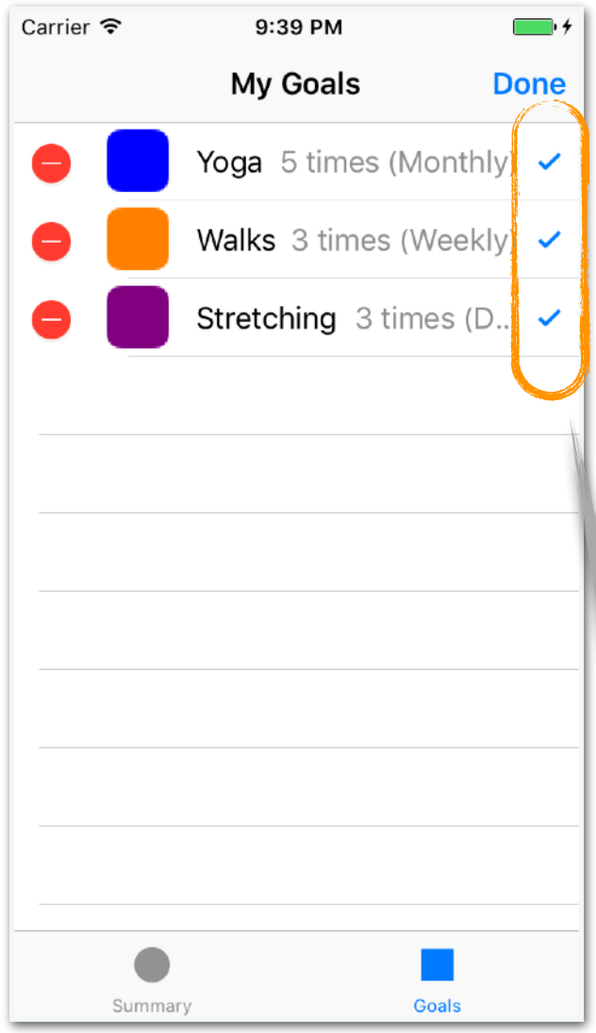
Need to be careful not to lose progress info






```
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 {                    // change mode
 case .displaying:
   navigationItem.setLeftBarButton(nil, …)
   editState = .editing(goalsActivity:
                        dataSource.goalsActivity)
```

alter UI state

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

GoalsController: UITableViewController

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

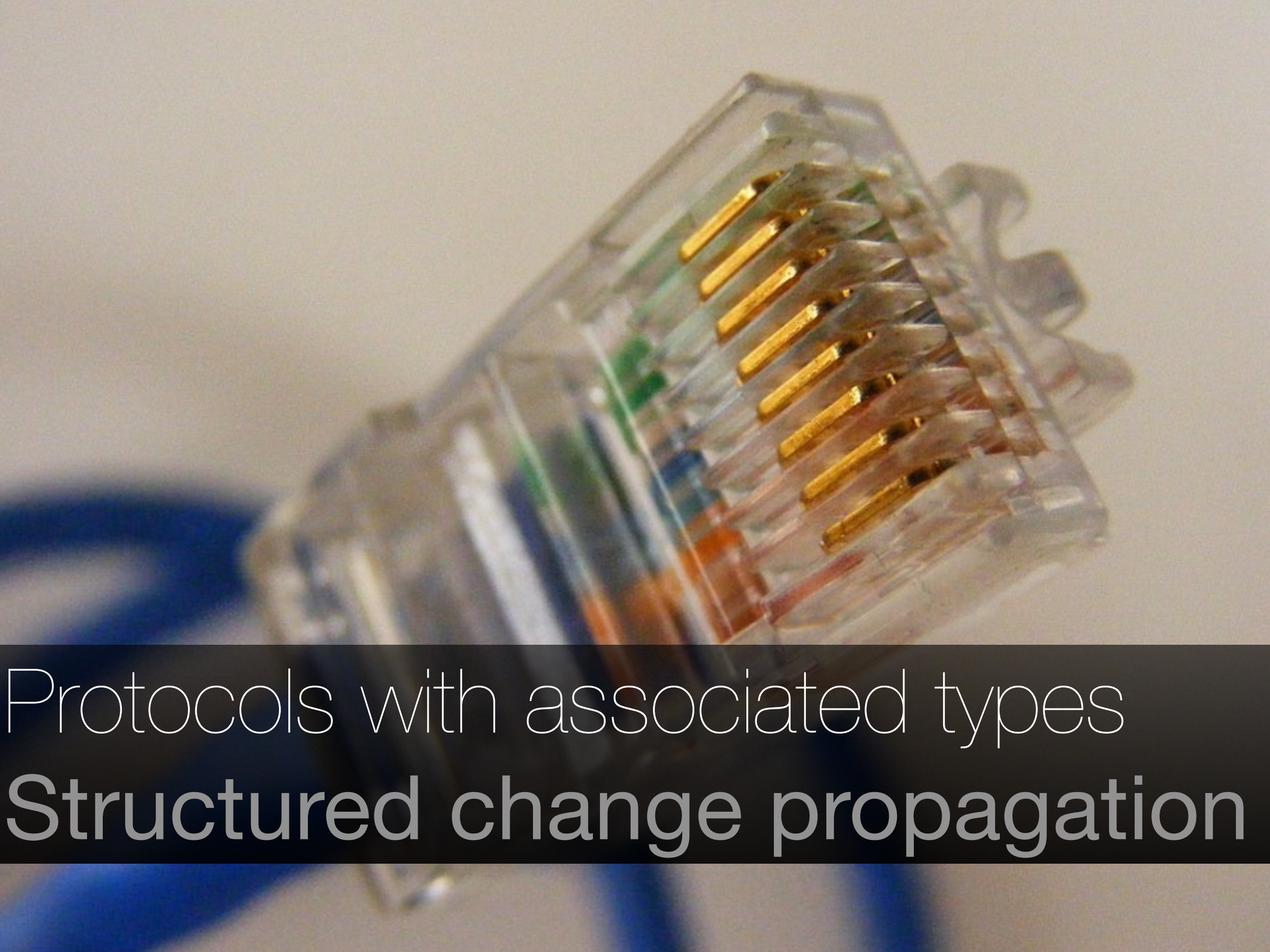
Replacing Tests

✓ 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

mchakravarty

TacticalGrace

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`
`mchakravarty`
`TacticalGrace`
`justtesting.org`

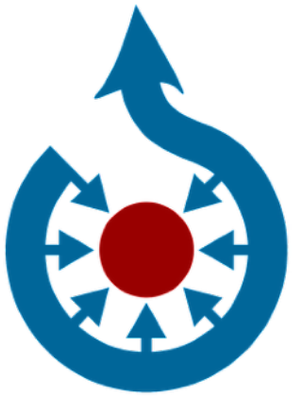
❤️ types >< state

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:Handcuffs01_2008-07-27.jpg
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