# Data Types in Swift

## Programming Safety Through Type Awareness

Alex Vollmer
@alexvollmer
http://alexvollmer.com



pluralsight
hardcore dev and IT training

# Cornerstones of Swift's Philosophy

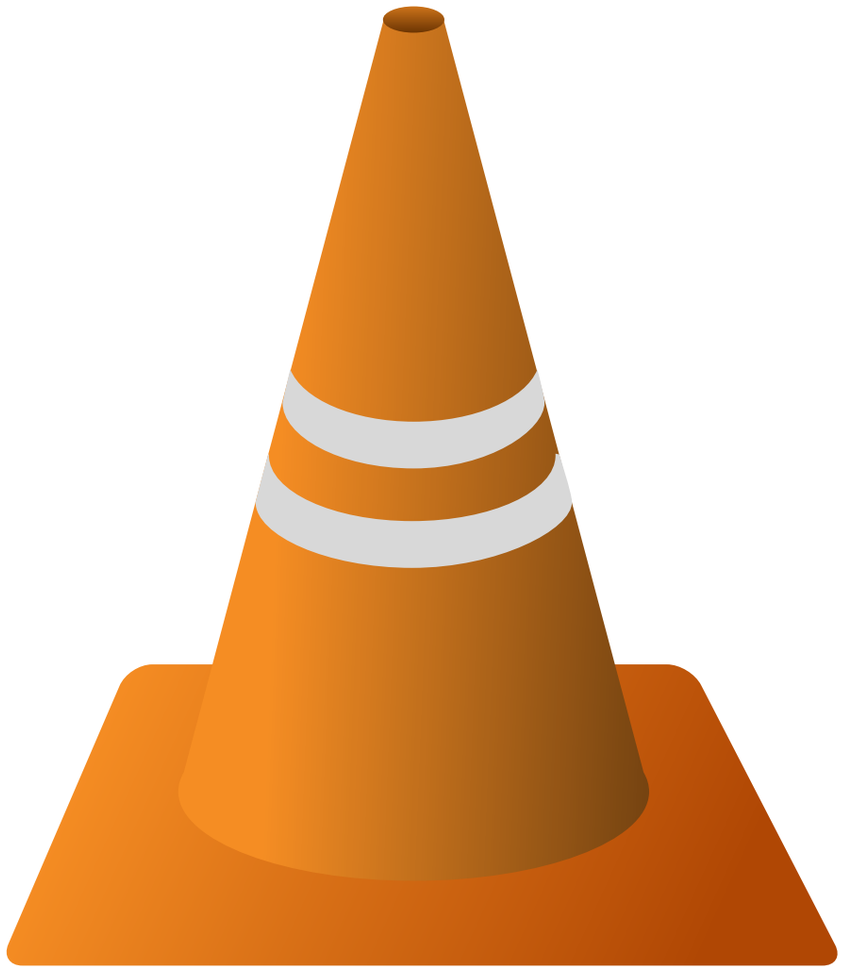
**Easy to learn**

**Modern**

Productive

**Safe**

# How Is a Language Made Safe?



**Prevents bad practices**

**No surprises**

**Can be reasoned**

**Unambiguous**


(206) 555-1111

John Doe **What is this?** $12.99

123 Main Street
Anytown WA 98111

# What is this?

Classes

Protocols

Extensions

Structs

Tuples

Functions

Closures

Enumerations

let

Constant

Immutable

Variable

Mutable

var

## Objective-C

```
NSArray *immutable = @[@"alpha", @"bravo", @"charlie"];

NSMutableArray *mutable =
  [NSMutableArray arrayWithArray:@"alpha", @"bravo", @"charlie"];
```

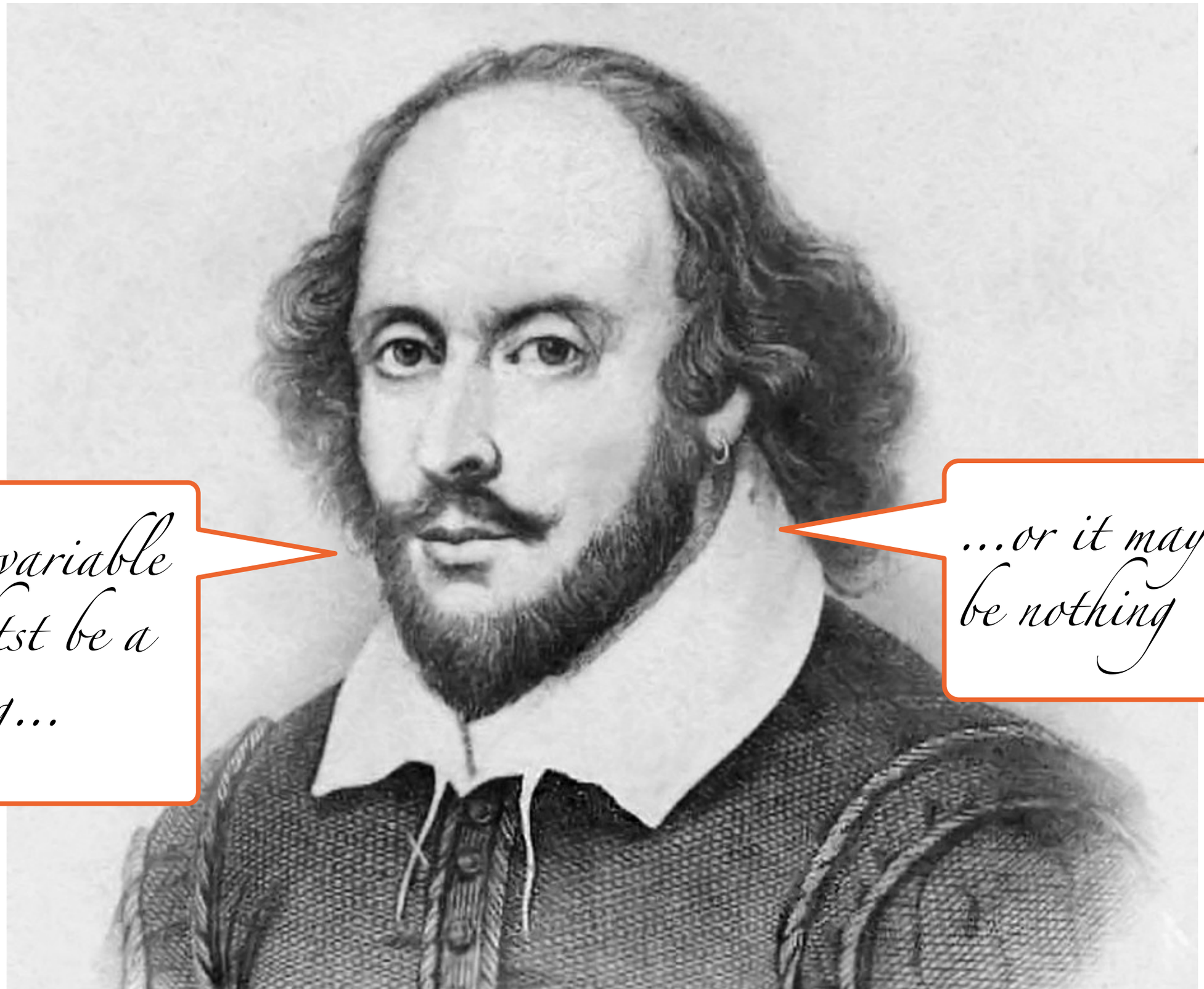
## Swift

```
var mutable = ["alpha", "bravo", "charlie"]

let immutable = ["alpha", "bravo", "charlie"]
```

> Constants are used throughout Swift to make code safer and clearer in intent when you work with values that do not need to change.
>
> — The Swift Programming Language

# To Be or Not To Be



# Optionals

```
var s1: String
var s2: String?
var s3: String!
```

# Optional Binding

```objc
NSFileManager *fm = [NSFileManager defaultManager];
NSString *path = @"/Users/alex/.gitignore";
NSData *gitignore = [fm contentsAtPath:path];
if (gitignore) {
    // TODO: parse contents
}
```

```swift
let fm = NSFileManager.defaultManager()
let path = "/Users/alex/.gitignore"
if let gitignore = fm.contentsAtPath(path) {
    // TODO: parse contents
}
```

# Optionals Summarized

Part of type-safety

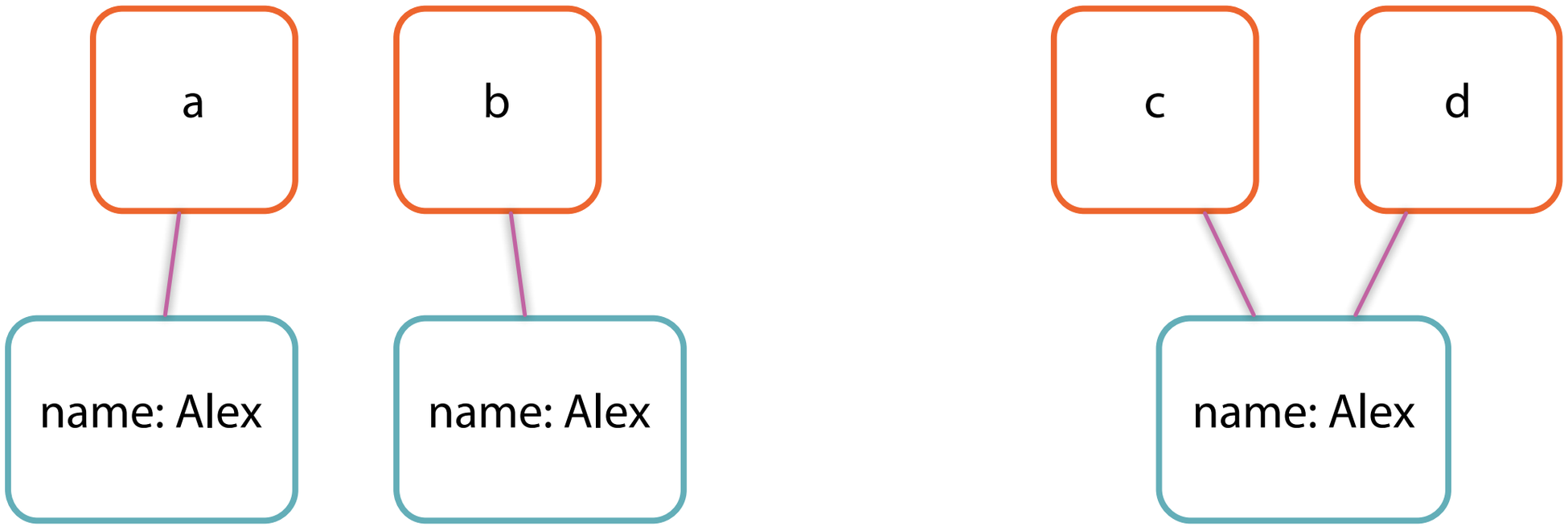
Compiler will catch these

Unwrap to access

Use Optional Binding

Any type can be optional

# Value & Reference Types



# Reference Types

```
class Person {
    var firstName: String
    var lastName: String

    init(firstName: String, lastName: String) {
        self.firstName = firstName
        self.lastName = lastName
    }
}


func printPerson(p: Person) -> Void {
    println("Person is \(p.firstName) \(p.lastName)")
}
```

# Value Types

## Structures

## Enumerations

## Tuples

# Classes & Objects

```
class Person {
    var firstName: String
    var lastName: String

    init(firstName: String, lastName: String) {
        self.firstName = firstName
        self.lastName = lastName
    }
}
```

# Classes & Objects

```
class Person {
    var firstName: String
    var lastName: String

    init(firstName: String, lastName: String) {
        self.firstName = firstName
        self.lastName = lastName
    }
}
```

**Single Inheritance model**

**Supports Polymorphism**

**Classes & Instances**

**Class & instance methods**

**Class & instance properties**

# Functions & Closures

```
func printPerson(p: Person) -> Void {
    println("Person is \(p.firstName) \(p.lastName)")
}
```

```
let stooges = ["Larry", "Moe", "Curly"]
stooges.filter({(name: String) -> Bool in
    return name.hasPrefix("L")
})
```

# Function Syntax

```
func printPerson(p: Person) -> Void {
    ...
}
```

# Function Argument Lists

```
func printPerson(person: Person)

func printPerson(person: Person, times: Int)
```

```
func printPerson(person p: Person, times t: Int)

func printPerson(#person: Person, #times: Int)
```

# Closure Syntax

```
stooges.filter({(name: String)-> Bool in
    return name.hasPrefix("L")
})
```

# Closure Syntax Variations

```
stooges.filter({(name: String) -> Bool in
    $0.hasPrefix("L")
})
```

# Closure Syntax Variations

```
stooges.filter({(name: String)-> Bool in
    return name.hasPrefix("L")
 })
```

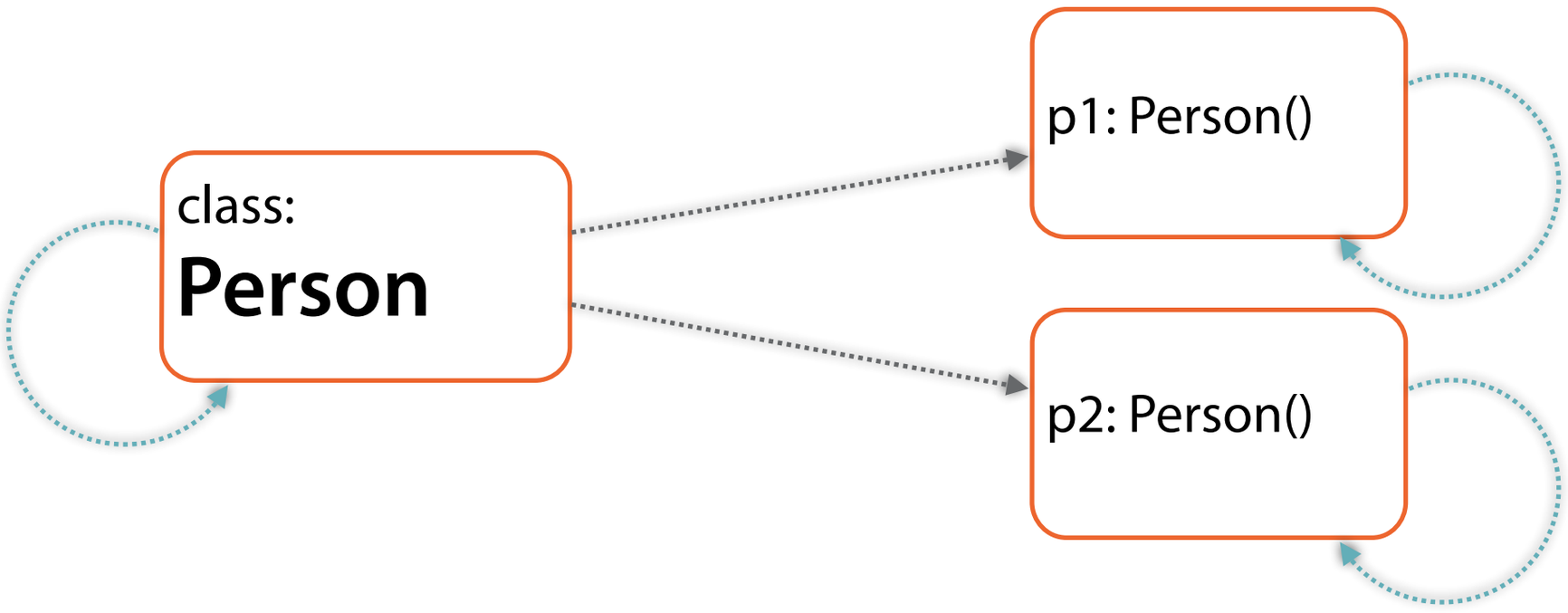
```
stooges.filter() {(name) -> Bool in
    return name.hasPrefix("L")
}
```

```
stooges.filter() {name -> Bool in
    return name.hasPrefix("L")
}
```

```
stooges.filter() {name -> Bool in
    name.hasPrefix("L")
}
```

```
stooges.filter() { $0.hasPrefix("L") }
```

# Methods



# **Methods** vs. Functions

Methods can reference the associated type or instance via **self**

**First** argument to method gets a default local name

**Subsequent** arguments get local *and* external names

# Structures vs. Classes

**Classes**

**Structures**

Define properties

Define methods

Define subscript access

Define initializers

Extensible

Protocol conformance

Inherit from a parent

Polymorphism

De-initializers

Reference-counting

# Structures Over Classes

Relatively simple data

No need for the "ceremony" of classes & objects

Data will be copied instead of referenced

Sometimes copying is just what you want

Encapsulated data are all value types

Why not make the container a value type too?

# Guess What…

```
struct Array<T> : MutableCollectionType, Sliceable {
    ...
}
```

```
    struct Dictionary<Key : Hashable, Value> {
        ...
    }
```

```
                struct Bool {
                  ...
                }
```

```
struct String {
    ...
}
```

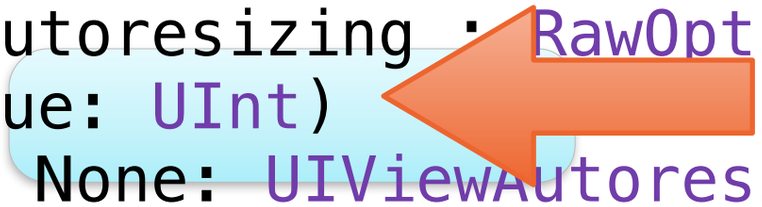
```
            struct Int : SignedIntegerType {
                ...
            }
```

# Structures for Options

```
typedef NS_OPTIONS(NSUInteger, UIViewAutoresizing) {
    UIViewAutoresizingNone                 = 0,
    UIViewAutoresizingFlexibleLeftMargin   = 1 << 0,
    UIViewAutoresizingFlexibleWidth        = 1 << 1,
    UIViewAutoresizingFlexibleRightMargin  = 1 << 2,
    UIViewAutoresizingFlexibleTopMargin    = 1 << 3,
    UIViewAutoresizingFlexibleHeight       = 1 << 4,
    UIViewAutoresizingFlexibleBottomMargin = 1 << 5
};
```

```
struct UIViewAutoresizing : RawOptionSetType {
    init(_ value: UInt)
    static var None: UIViewAutoresizing { get }
    static var FlexibleLeftMargin: UIViewAutoresizing { get }
    static var FlexibleWidth: UIViewAutoresizing { get }
    static var FlexibleRightMargin: UIViewAutoresizing { get }
    static var FlexibleTopMargin: UIViewAutoresizing { get }
    static var FlexibleHeight: UIViewAutoresizing { get }
    static var FlexibleBottomMargin: UIViewAutoresizing { get }
}
```

```
struct UIViewAutoresizing : RawOptionSetType {
    init(_ value: UInt)
    static var None: UIViewAutoresizing { get }
    static var FlexibleLeftMargin: UIViewAutoresizing { get }
    static var FlexibleWidth: UIViewAutoresizing { get }
    static var FlexibleRightMargin: UIViewAutoresizing { get }
    static var FlexibleTopMargin: UIViewAutoresizing { get }
    static var FlexibleHeight: UIViewAutoresizing { get }
    static var FlexibleBottomMargin: UIViewAutoresizing { get }
}
```



```
view.autoresizingMask = .FlexibleHeight | .FlexibleWidth
```

## Structures

Define properties

Define methods

Define subscript access

Define initializers

Extensible

Protocol conformance

Inherit from a parent

Polymorphism

De-initializers

Reference-counting

Classes

https://www.flickr.com/photos/mckln/3450139380

# Enumerations

# Objective-C Enumerations

```
typedef NS_ENUM(NSInteger, UIViewAnimationCurve) {
    UIViewAnimationCurveEaseInOut,
    UIViewAnimationCurveEaseIn,
    UIViewAnimationCurveEaseOut,
    UIViewAnimationCurveLinear
};
```

# Swift Enumerations

```
enum UIViewAnimationCurve : Int {
    case EaseInOut
    case EaseIn
    case EaseOut
    case Linear
}
```

```
enum UIViewTintAdjustmentMode : Int {
    case Automatic
    case Normal
    case Dimmed
}
```

# Swift Enumerations

```
enum UIViewAnimationCurve : Int {
    case EaseInOut
    case EaseIn
    case EaseOut
    case Linear
}
```

## Methods

## Properties

## Associated “raw” values

# Using Enumerations

```
enum UIViewAnimationCurve : Int {
    case EaseInOut
    case EaseIn
    case EaseOut
    case Linear
}
```

```
UIView.animationCurve = UIViewAnimationCurve.EaseInOut
```

```
typedef NS_ENUM(NSInteger, UIViewAnimationCurve) {
    UIViewAnimationCurveEaseInOut,
    UIViewAnimationCurveEaseIn,
    UIViewAnimationCurveEaseOut,
    UIViewAnimationCurveLinear
};
```


# Tuples

# Tuples

```
var s1 = (code:(Int, Int), message: String)
```

```
var s1 = (code: 404, message: "Not Found")

    s1.code      // returns 404
    s1.message   // returns "Not Found"

       s1.0  // returns 404
       s1.1  // returns "Not Found"
```

# Tuples & Optionals

```
(Int, String)

(Int?, String?)

(Int, String)?
```

# Why Tuples?

```
let result = CoolService.doSomethingCool()
if result.error != nil {
    println("Uh-oh, error: \(result.error)")
}
else {
    println("The result is \(result.value)")
}
```

(NSError?, String?)

# Rules for Tuples

```
(NSError?, String?)
```

**Return values**

**Unneeded for arguments**

**Short-lived, Temporary**

**More? Use class or struct**

http://www.flickr.com/photos/stevenduong/4071937951

# Protocols

```
protocol Tuneable {
    var pitch: Double { get }
    func tuneSharp()
    func tuneFlat()
}
```

# Extensions

Add computed properties

Define new methods

Define nested types

Protocol conformance

No overriding

Classes

Protocols

Extensions

Structs

Tuples

Functions

Closures

Enumerations

let

Constant

Immutable

Variable

Mutable

var

```
var s1: String
var s2: String?
var s3: String!
```

# Classes & Objects

```
class Person {
    var firstName: String
    var lastName: String

    init(firstName: String, lastName: String) {
        self.firstName = firstName
        self.lastName = lastName
    }
}
```

# Functions & Closures

```
func printPerson(p: Person) -> Void {
    println("Person is \(p.firstName) \(p.lastName)")
}
```

```
let stooges = ["Larry", "Moe", "Curly"]
stooges.filter({(name: String) -> Bool in
    return name.hasPrefix("L")
})
```

# Value Types

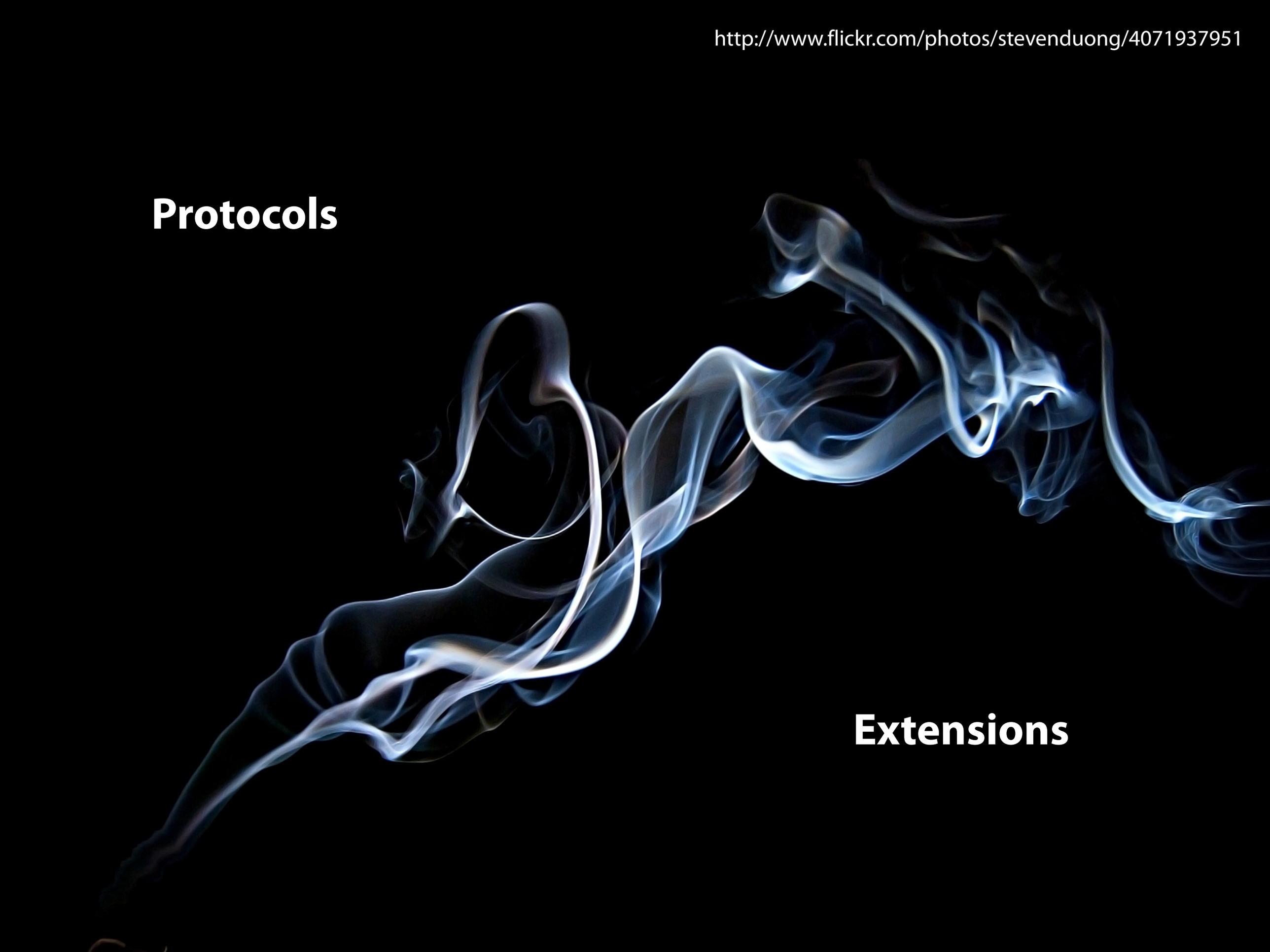
## Structures

## Enumerations

## Tuples

http://www.flickr.com/photos/stevenduong/4071937951

# Protocols



# Extensions