Catching Up with Swift

Ash Furrow, Artsy

“What’s the worst that could happen?”

Agenda

1. *Swift was needed*
2. *Swift met those needs, mostly*
3. *Writing Swift is great, mostly*
4. *Using Swift in production*
5. *The future of Swift*

Swift Was Needed.

Objective-C

- *From the early 1980's*
- *Originally, a C preprocessor*
  - *Then an advantage*
  - *Now a burden*

Objective-C

- *Moderate use until the mid 2000's*
- *Used to make OS X apps*
  - *Niche market*
- *Then the iPhone happened*

TIOBE Programming Community Index Objective-C

Source: http://bit.ly/1E2BOCs

Objective-C

- *Sudden interest, despite:*
- *Esoteric syntax*
- *Unusual memory management*
- *Arcane knowledge*

Objective-C

- *Three distinct groups of developers emerged:*
  - *First wave, developers from the 80's/90's.*
  - *Second wave, OS X developers from the 2000's.*
  - *Third wave, developers attracted by the iPhone.*

Friction.

Objective-C

- *Tension.*
- *Disenfranchisement.*
- *Resentment.*

“While hardware performance increases over time, the human capacity to deal with complexity does not.”

—John Siracusa

Objective-C

- *New features were developed by Apple:*
  - *Dot property syntax.*
  - *Closures.*
  - *Decreased boilerplate, headers.*
  - *Automatic reference counting.*
  - *Collection literals.*
  - *Primitive boxing syntax.*

“See? Objective-C is getting better! We don’t need to replace it!”

—First/Second Wave Developers

False.

Language Evolution

- *Machine code.*
- *Assembly.*
- *Procedural languages (C).*
- *Object-oriented languages (C++, Objective-C).*
- *Virtual machines (Java, C#, Ruby, etc).*

Eventually, writing Objective-C will seem

archaic

and relying on it would be a

competitive disadvantage.

“Eventually.”

Replacing your home-grown programming language takes

decades.

Objective-C

- *Could not escape its C roots.*

- *Apple began work on Swift in 2010.*

Objective-C improved

because

of

Swift

Objective-C Replacement

- *Needs to...*
  - *abandon all C roots.*
  - *be memory managed.*
  - *have native unicode strings, native collections.*
  - *be concise.*
  - *have named parameters.*

Swift Met Those Needs.

Mostly.

Swift

- *Announced June 2014.*
- *Betas released until a 1.0 in the Autumn.*
- *“Objective-C without the C.”*
  - *Mischaracterization.*

Swift

Needed to...

😕 *abandon all C roots.*

😕 *be memory managed.*

😄 *have native unicode strings, native collections.*

😄 *be concise.*

😄 *have named parameters.*

Abandon C Roots

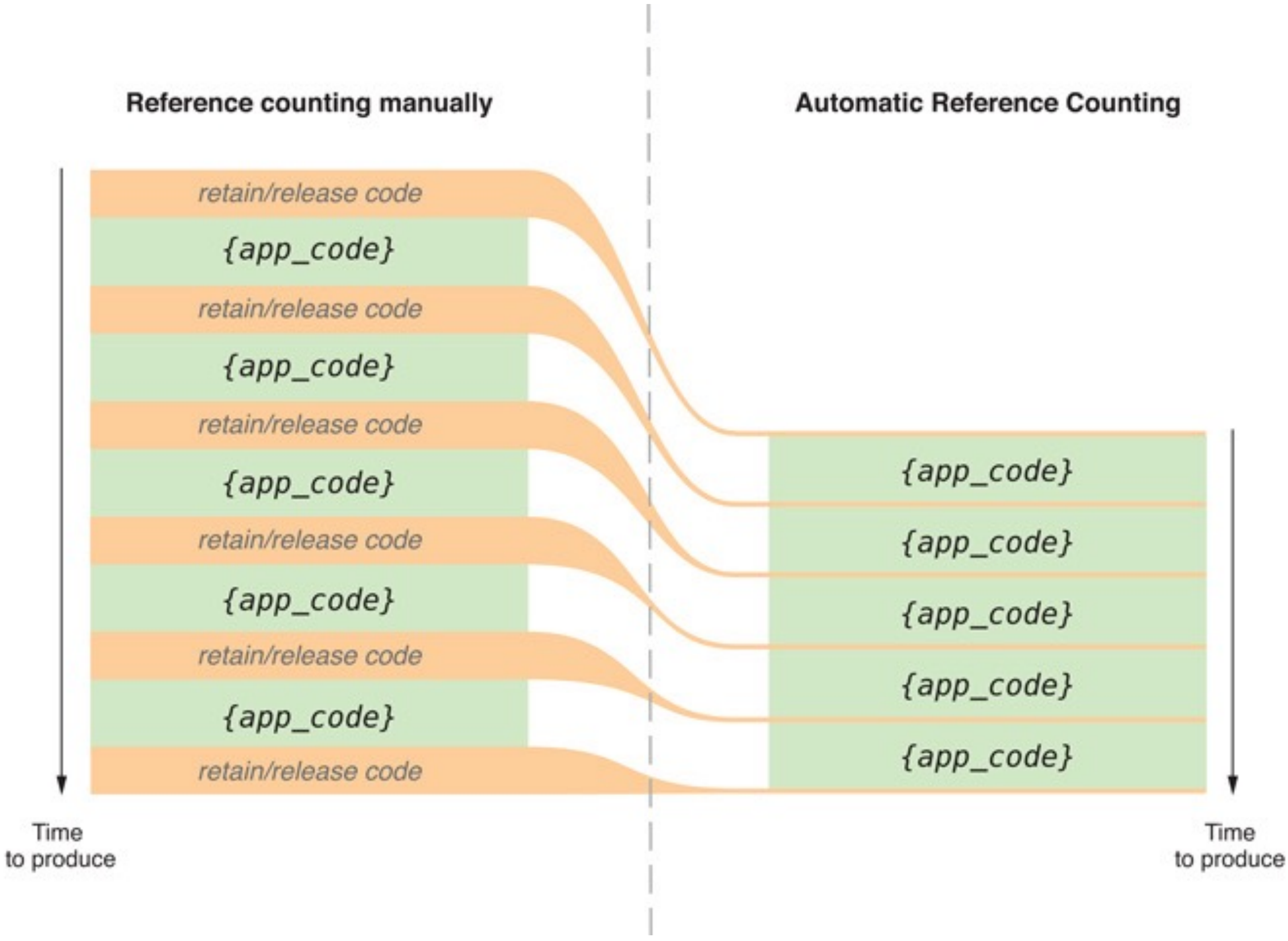
- *Swift needed to have full Objective-C interop.*
  - *Which means full C interop.*
- *It's possible to write Swift to interact with C APIs.*
  - *It's ugly and discouraged.*
  - *Well, I discourage it anyway.*

Be Memory Managed

- *Objective-C introduced ARC in 2011.*
- *Replaced garbage collection on OS X.*
- *Replaced manual memory management on iOS and OS X.*

Be Memory Managed

- *Automatic Reference Counting.*
  - *Same as manual memory management.*
  - *Inserted for the developer at compile-time.*
  - *Reasoned about and optimized by compiler.*


Reference counting manually
Automatic Reference Counting
retain/release code
{app_code}
retain/release code
{app_code}
retain/release code
{app_code}
retain/release code
{app_code}
retain/release code
{app_code}
retain/release code
{app_code}
{app_code}
{app_code}
{app_code}
{app_code}
{app_code}
Time
to produce
Time
to produce

Be Memory Managed

Pros

- *Familiar, stable technology.*
- *No garbage collector overhead.*

Cons

- *Can’t detect reference cycles.*

Native Unicode Strings

- *Got ‘em.*
- *Strings are a Swift struct.*
  - *Bridgeable to Objective-C NSString instances.*
  - *Handle double-byte Unicode characters.*

Native Unicode Strings

```
func ツ() -> RACSignal {
    return hideAllTheThingsSignal()
}
```

```
func 📷(snapshottable: Snapshotable) {
    expect(snapshottable).to( recordSnapshot() )
}
```

Native Collections

- *Collections are also Swift structs, on generics.*
- *Array<T>, Dictionary<K, V>, and Set<T>.*
- *Bridgeable to Objective-C equivalents.*
- *Concise syntax.*

Be Concise

- *Subjective, but I'm happy.*
- *Simple things are easy.*
- *Difficult things are possible.*

Named Parameters

- *Optional(ish)*
- *Compiler does weird things for Objective-C interop.*

¯_(ツ)_/¯

Named Parameters

```
func compare(lhs: String, to rhs: String) -> Bool {
    return lhs == rhs
}

compare("Hi", to: "Hello")
```

Writing Swift is Great.

Mostly.

Problem-solving in Swift needs to be

different from

problem-solving with

Objective-C syntax.


IT'S A SHAME!

Generics

- *Objective-C is dynamically typed.*
- *Swift is statically typed.*
  - *Awesome.*
  - *(ish).*
- *Compile-time type safety.*

Generics

- *Objective-C distinguishes primitives and classes.*
- *Swift is all like (ノ °□°) ノ ⌒ ┻━┻*
- *Arrays, dictionaries, and sets all use generics.*

Generics

```
struct Stack<T> {
    private var contents = Array<T>()

    mutating func push(value: T) {
        contents.append(value)
    }

    mutating func pop() -> T {
        return contents.removeAtIndex(0)
    }

    var isEmpty: Bool {
        return countElements(contents) == 0
    }
}
```

Generics

```
var intStack = Stack<Int>()
var stringStack = Stack<String>()
var stackStack = Stack<Stack<AnyObject>>()

intStack.push(1)
intStack.pop() // Returns 1
```

Lazy Swift

- *Language-level concept of lazy evaluation.*
- *Applied automatically to global variables.*
- *Can be applied to any property.*

Lazy Swift

- *Assigned on first access.*
- *Can be overridden by setting before first access.*
- *Really cool trick with closures.*

Lazy Swift

```
class MyClass {
    lazy var name = "Ash Furrow"
}

MyClass().name // Returns "Ash Furrow"

let instance = MyClass()
instance.name = "Orta Therox"
instance.name // Returns “Orta Therox"
```

Lazy Swift

```
class MyClass {
    lazy var name = "Ash Furrow”
    lazy var greeting: String = {
        return "Hello, \(self.name)"
    }()
}

MyClass().greeting // Returns "Hello, Ash Furrow"

let instance = MyClass()
instance.name = "Orta Therox"
instance.greeting // Returns "Hello, Orta Therox"
instance.name = "Eloy Durán"
instance.greeting // Returns "Hello, Orta Therox"
```

Extending Types

- *Objective-C has "categories" to extending existing classes.*
- *Swift has "extensions", instead.*
  - *The work on all types.*

Extending Types

```
extension Int {
    var hours: NSTimeInterval {
        return NSTimeInterval(3600 * self)
    }
}
```

```
extension NSTimeInterval {
    var fromNow: NSDate {
        return NSDate(timeIntervalSinceNow: self)
    }
    var ago: NSDate {
        return NSDate(timeIntervalSinceNow: -self)
    }
}

4.hours.fromNow
4.hours.ago
```

Index Paths

- *Used to identify cells in a table view.*
  - *Section, row.*
- *Lots of horrendous code.*
  - *It's so bad.*
  - *Seriously bad.*

Index Paths

```
if (indexPath.section == 0) {
    if (indexPath.row == 0) {

    } else if (indexPath.row == 1) {

    } else if ...
} else if (indexPath.section == 1) {
    if (indexPath.row == 0) {

    } else if (indexPath.row == 1) {

    } else if ...
} else if ...
```

Index Paths

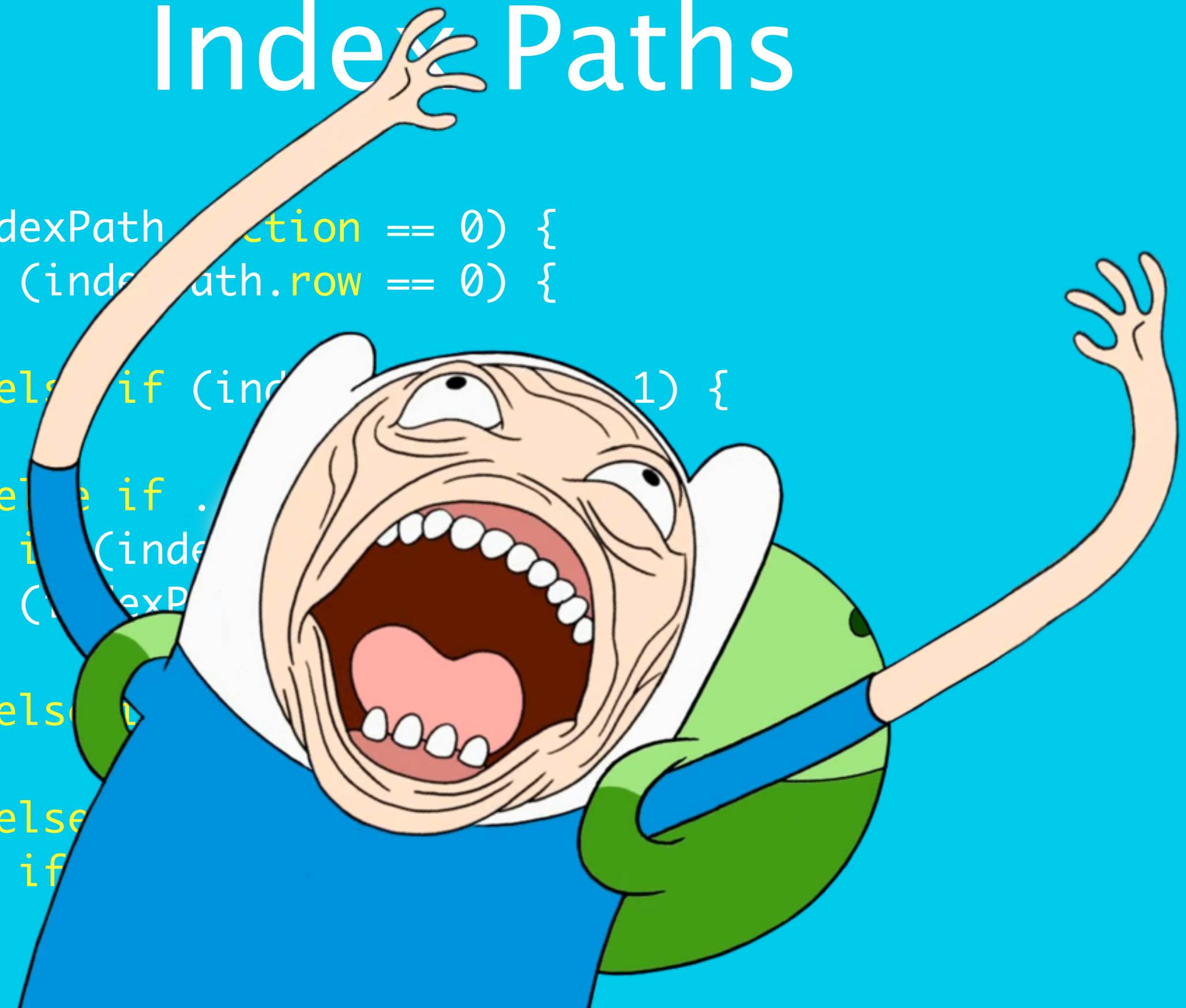
```
if (indexPath      ction == 0) {
    if (inde      ath.row == 0) {

    } els    if (ind                1) {

    } el   e if .
} else i   (inde
    if (i   lexP

    } else

    } else
} else if
```



Index Paths

```
switch (indexPath.section, indexPath.row) {
case (0, 0):
case (0, 1):
case (1, 0):
case (1, 1):
default:
    // nop
}
```

Is that

better?

NO.

Index Paths

```
switch (indexPath.section, indexPath.row) {
case (0, let row):
    // Executed for any section 0, row is row.
case (let section, 0) where section % 2 == 1:
    // Executed for first rows of odd sections.
case (let section, let row) where validate(section):
    // Executed when validate() returns true.
default:
    // Executed on all other cases.
}
```

Is that

better?

Maybe.

Let’s look for new ways to solve familiar problems.

Let’s ask other communities how they solve problems.

Swift in Production

ARTSY


ART.SY
Bid here
ART.SY
Bid here

Open Source by Default

- *Decided to develop the app in the open.*
  - *Because why not?*
    - *No, seriously. Why not?*
- *Helpful for asking for assistance from others.*
  - *“Here’s my code – what’s wrong?”*

github.com/artsy/eidolon

August

- *Swift had been out for two months.*
- *Stability had improved.*
- *Swift seemed ready.*

Nope.

September

- *The language was great.*
- *Lots of frustration with tools.*
- *3rd party tools weren't ready, or didn't exist.*
  - *So we built some.*
  - *And contributed to others.*

October

- *Running behind schedule.*
  - *“Hard deadline.”*
- *Explored options to speed up development.*
- *Brought on an extra developer to help.*

“We don’t expect to meet our deadline.”

—My boss

We made our deadline.

Burnout.

Significant

technical debt.

Problem Solving

- *Compiler optimizations segfault the compiler.*
  - *Disable optimizations.*
- *App is too slow without optimizations.*
  - *Buy faster iPads.*
- *Tools didn’t exist.*
  - *So we built them.*

Swift is still

hands on.

But it's also

awesome.

Future of Swift

Safe Bets

- *Tools will continue to improve.*
  - *Always a year away from being stable.*
- *Language will continue to be awesome.*
  - *And get more awesomer.*

Predictions

- *More functional-esque APIs from Apple.*
- *More functional-esque APIs from the community.*
- *No Swift-only APIs from Apple, for now.*
    - *Apple doesn't want to disenfranchise first/ second wave developers.*

Recap

1. *Swift was needed*
2. *Swift met those needs, mostly*
3. *Writing Swift is great, mostly*
4. *Using Swift in production*
5. *The future of Swift*

Thanks!

@ashfurrow

@ashfurrow

leanpub.com/yourfirstswiftapp