#WWDC18

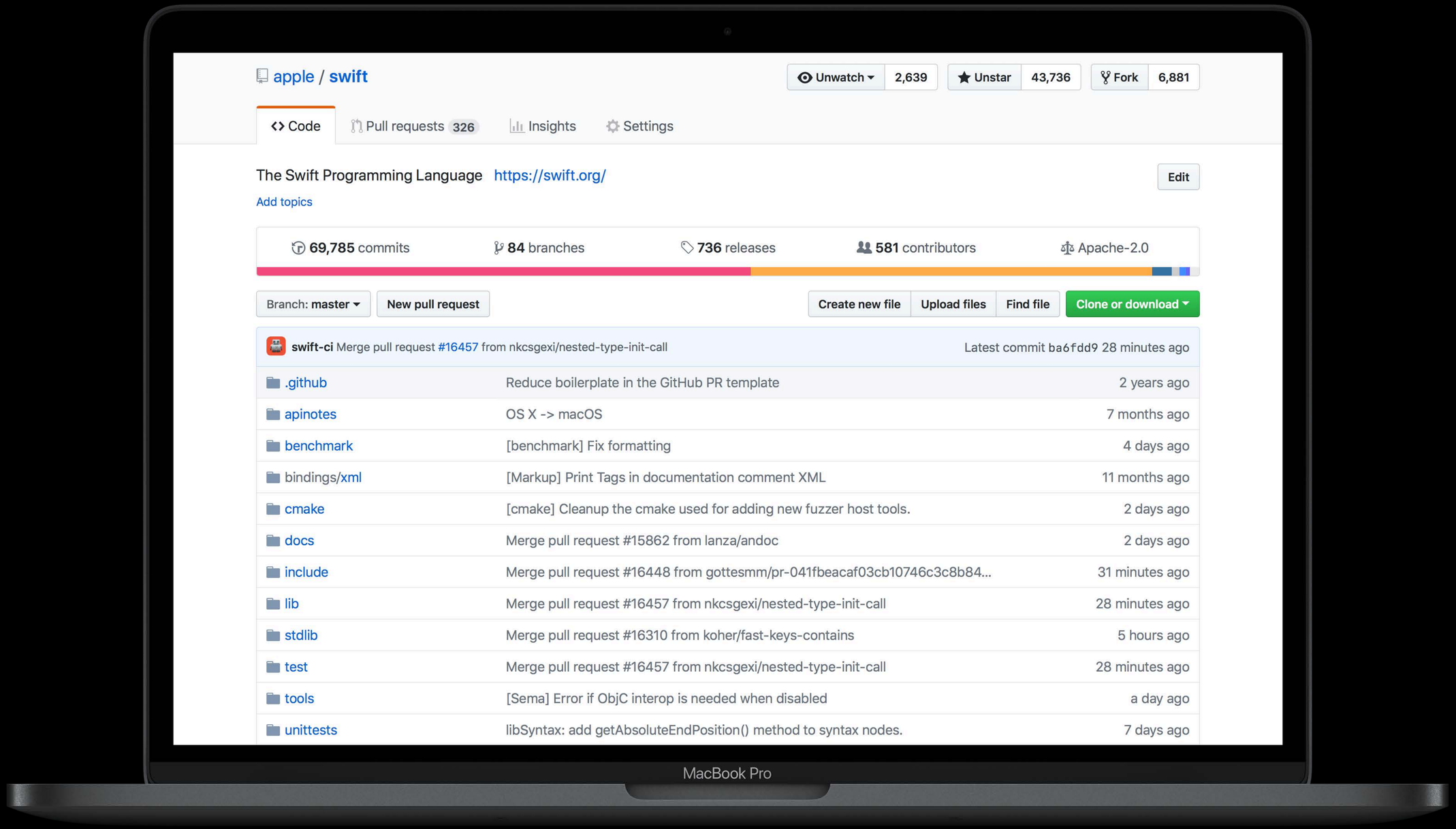
What's New in Swift

Session 401

Ted Kremenek, Languages & Runtimes Manager
Slava Pestov, Swift Compiler Engineer

Swift open source

Swift 4.2

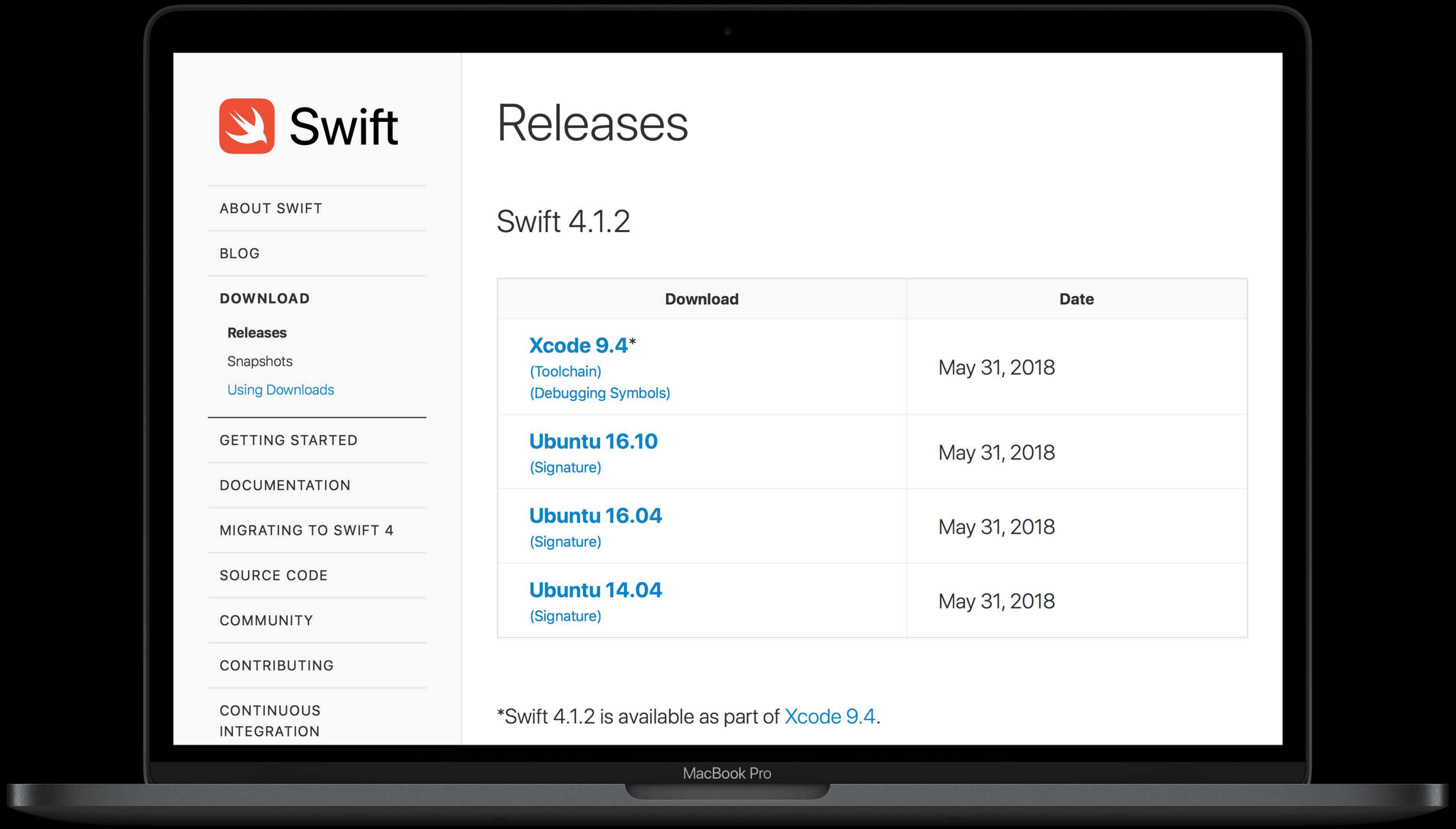

apple / swift
Unwatch 2,639
Unstar 43,736
Fork 6,881
Code
Pull requests 326
Insights
Settings
The Swift Programming Language https://swift.org/
Edit
Add topics
69,785 commits
84 branches
736 releases
581 contributors
Apache-2.0
Branch: master
New pull request
Create new file
Upload files
Find file
Clone or download
swift-ci Merge pull request #16457 from nkcsgexi/nested-type-init-call
Latest commit ba6fdd9 28 minutes ago
.github Reduce boilerplate in the GitHub PR template 2 years ago
apinotes OS X -> macOS 7 months ago
benchmark [benchmark] Fix formatting 4 days ago
bindings/xml [Markup] Print Tags in documentation comment XML 11 months ago
cmake [cmake] Cleanup the cmake used for adding new fuzzer host tools. 2 days ago
docs Merge pull request #15862 from lanza/andoc 2 days ago
include Merge pull request #16448 from gottesmm/pr-041fbeacaf03cb10746c3c8b84... 31 minutes ago
lib Merge pull request #16457 from nkcsgexi/nested-type-init-call 28 minutes ago
stdlib Merge pull request #16310 from koher/fast-keys-contains 5 hours ago
test Merge pull request #16457 from nkcsgexi/nested-type-init-call 28 minutes ago
tools [Sema] Error if ObjC interop is needed when disabled a day ago
unittests libSyntax: add getAbsoluteEndPosition() method to syntax nodes. 7 days ago
MacBook Pro

600

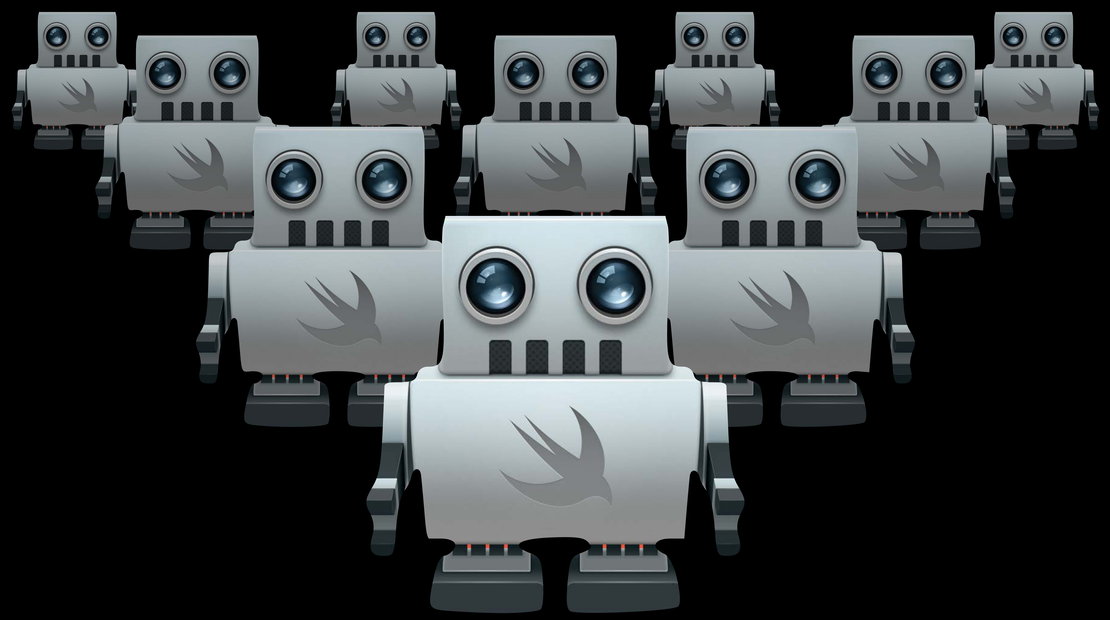
Code contributors to Swift

18k

Merged pull requests on GitHub


Swift
ABOUT SWIFT
BLOG
DOWNLOAD
Releases
Snapshots
Using Downloads
GETTING STARTED
DOCUMENTATION
MIGRATING TO SWIFT 4
SOURCE CODE
COMMUNITY
CONTRIBUTING
CONTINUOUS INTEGRATION
Releases
Swift 4.1.2
Download
Date
Xcode 9.4*
(Toolchain)
(Debugging Symbols)
May 31, 2018
Ubuntu 16.10
(Signature)
May 31, 2018
Ubuntu 16.04
(Signature)
May 31, 2018
Ubuntu 14.04
(Signature)
May 31, 2018
*Swift 4.1.2 is available as part of Xcode 9.4.
MacBook Pro

Community-Hosted Continuous Integration

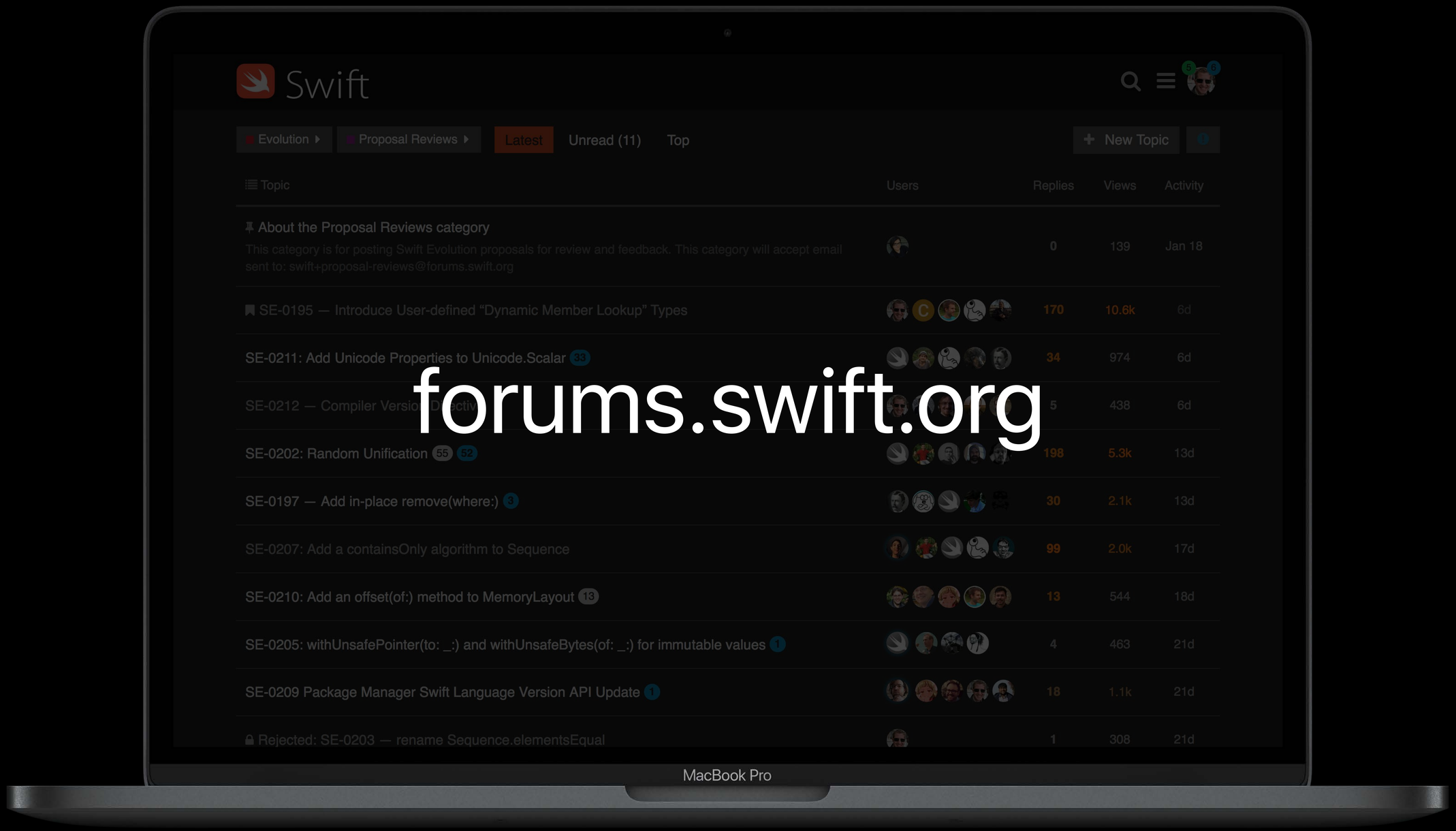


Debian 9.1 on Raspberry Pi

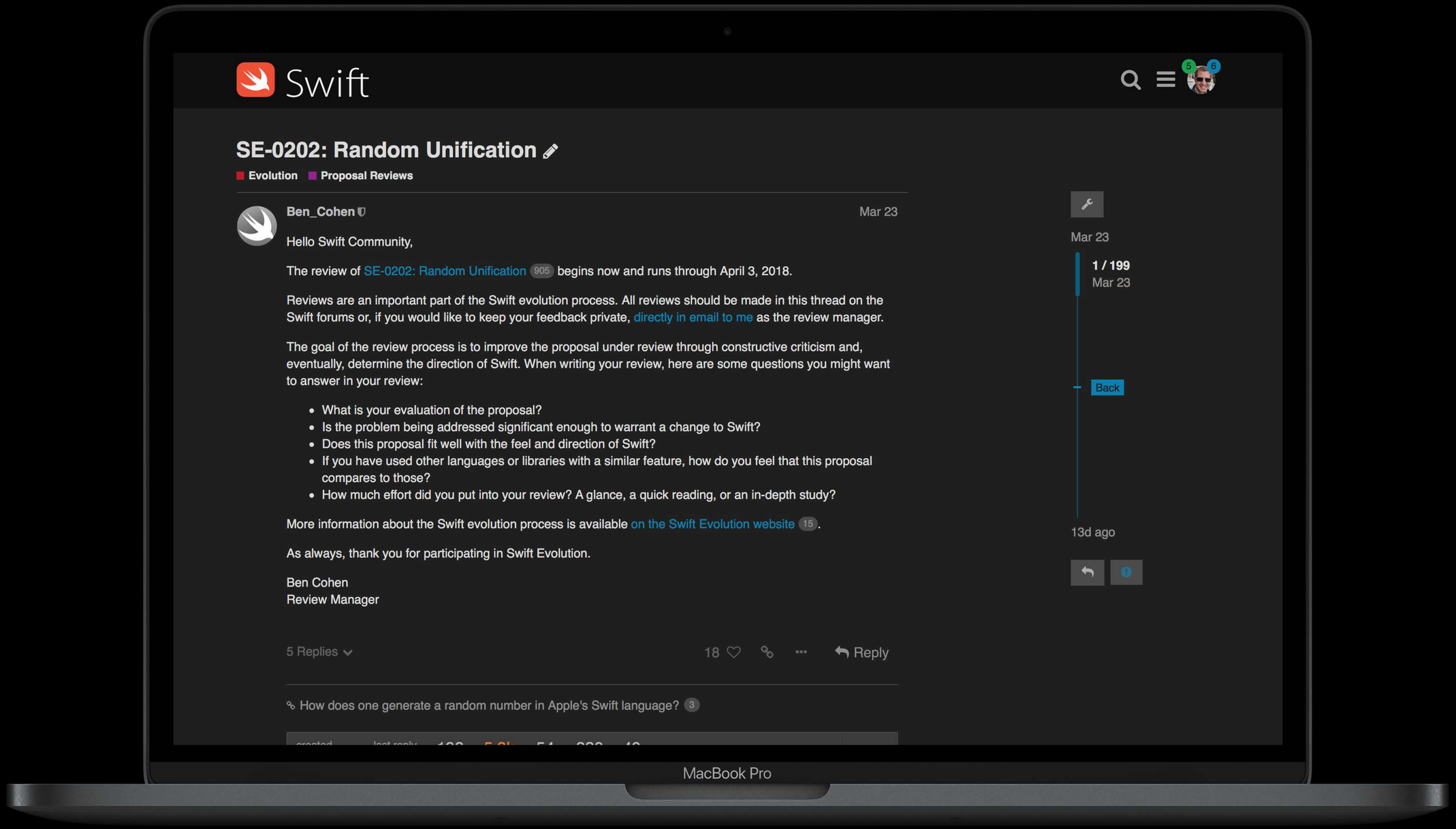
Fedora 27

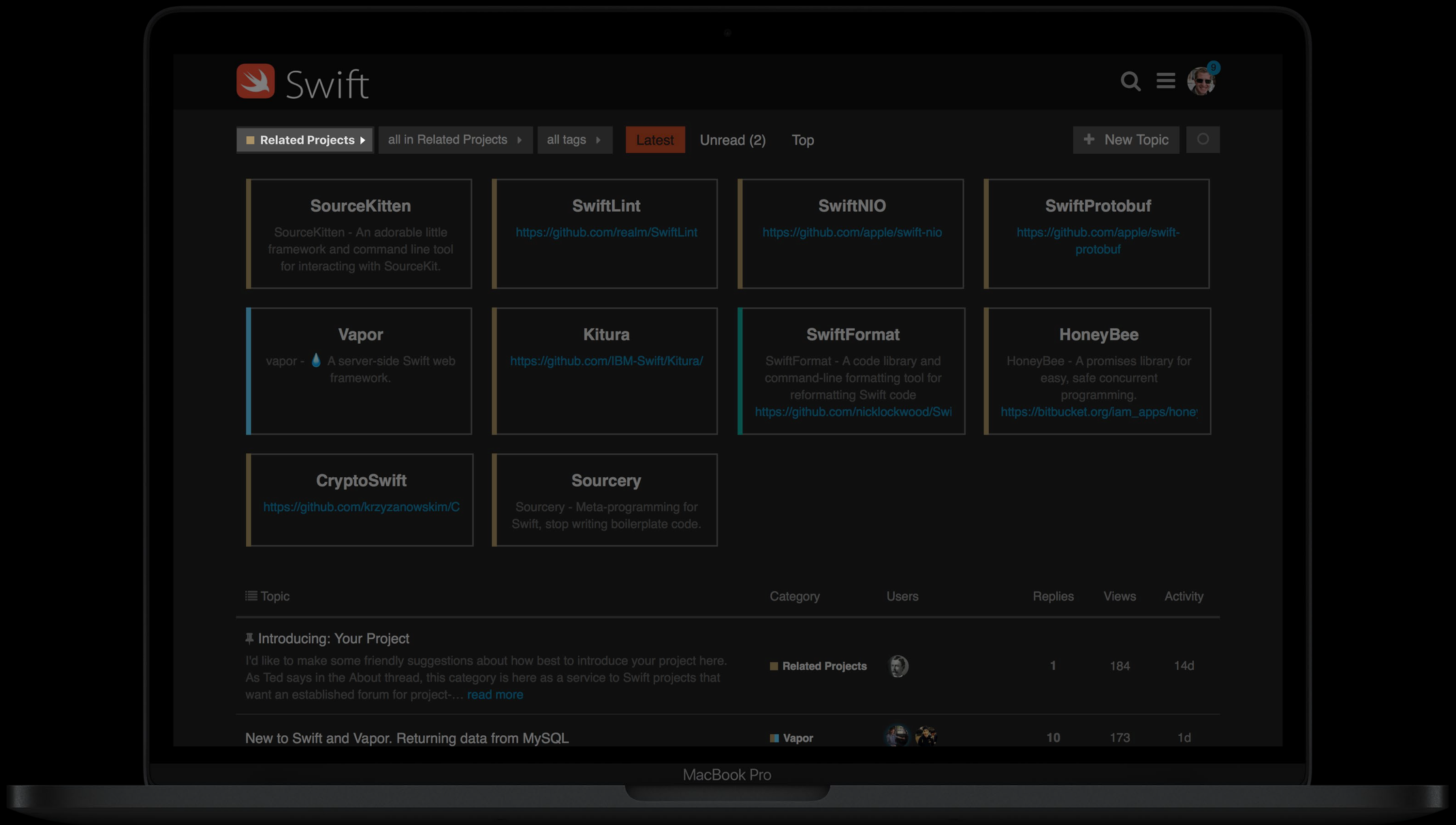
Ubuntu on PowerPC

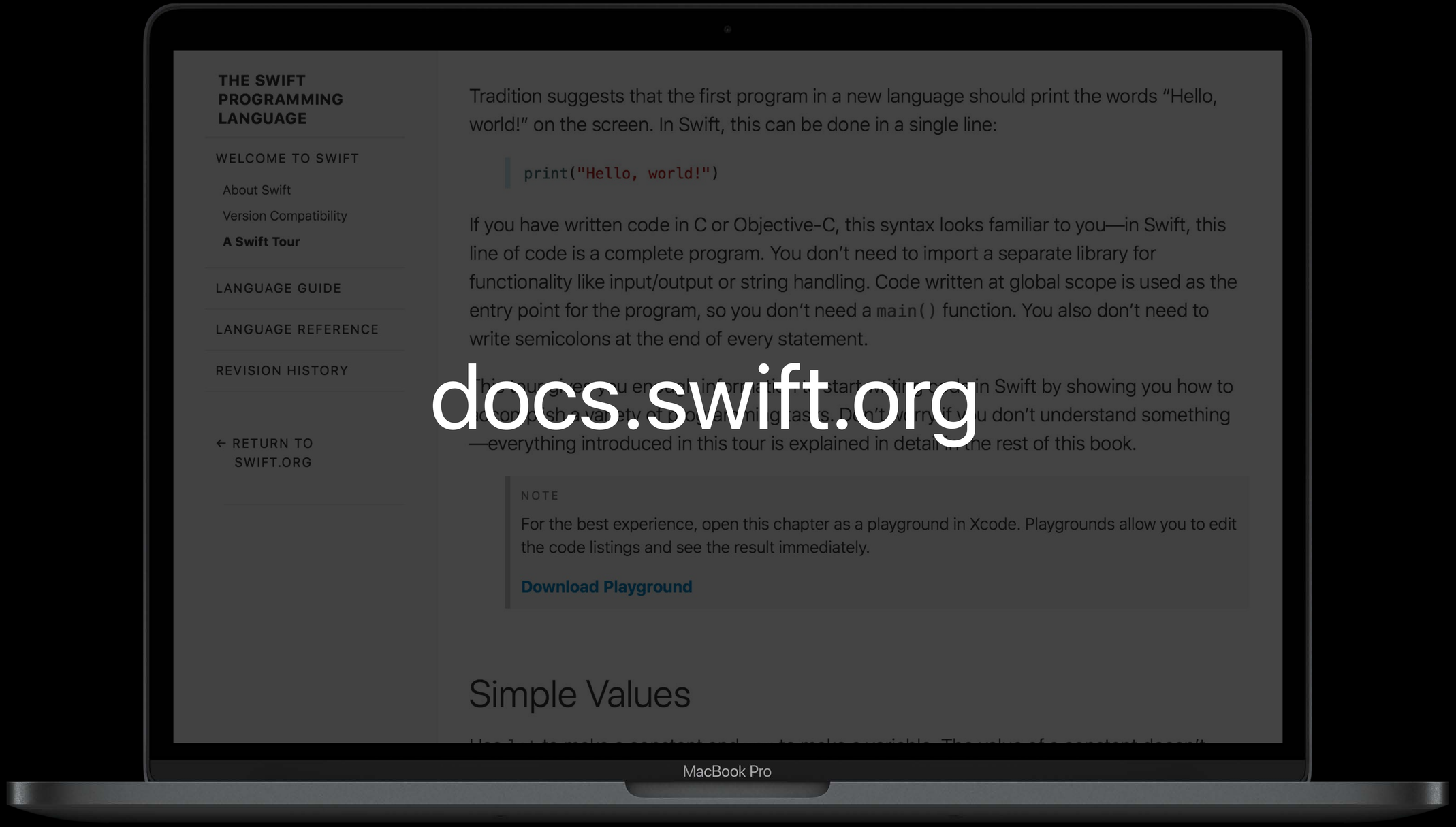
Swift for TensorFlow



forums.swift.org


Swift
SE-0202: Random Unification
Evolution
Proposal Reviews
Ben_Cohen
Mar 23
Hello Swift Community,
The review of SE-0202: Random Unification 905 begins now and runs through April 3, 2018.
Reviews are an important part of the Swift evolution process. All reviews should be made in this thread on the Swift forums or, if you would like to keep your feedback private, directly in email to me as the review manager.
The goal of the review process is to improve the proposal under review through constructive criticism and, eventually, determine the direction of Swift. When writing your review, here are some questions you might want to answer in your review:
What is your evaluation of the proposal?
Is the problem being addressed significant enough to warrant a change to Swift?
Does this proposal fit well with the feel and direction of Swift?
If you have used other languages or libraries with a similar feature, how do you feel that this proposal compares to those?
How much effort did you put into your review? A glance, a quick reading, or an in-depth study?
More information about the Swift evolution process is available on the Swift Evolution website 15.
As always, thank you for participating in Swift Evolution.
Ben Cohen
Review Manager
5 Replies
18
Reply
How does one generate a random number in Apple's Swift language? 3
Mar 23
1 / 199
Mar 23
Back
13d ago
MacBook Pro


Swift
Related Projects
all in Related Projects
all tags
Latest
Unread (2)
Top
New Topic
SourceKitten
SourceKitten - An adorable little framework and command line tool for interacting with SourceKit.
SwiftLint
https://github.com/realm/SwiftLint
SwiftNIO
https://github.com/apple/swift-nio
SwiftProtobuf
https://github.com/apple/swift-protobuf
Vapor
vapor - A server-side Swift web framework.
Kitura
https://github.com/IBM-Swift/Kitura/
SwiftFormat
SwiftFormat - A code library and command-line formatting tool for reformatting Swift code
https://github.com/nicklockwood/Swi
HoneyBee
HoneyBee - A promises library for easy, safe concurrent programming.
https://bitbucket.org/iam_apps/hone
CryptoSwift
https://github.com/krzyzanowskim/C
Sourcery
Sourcery - Meta-programming for Swift, stop writing boilerplate code.
Topic
Category
Users
Replies
Views
Activity
Introducing: Your Project
I'd like to make some friendly suggestions about how best to introduce your project here. As Ted says in the About thread, this category is here as a service to Swift projects that want an established forum for project-… read more
Related Projects
1
184
14d
New to Swift and Vapor. Returning data from MySQL
Vapor
10
173
1d
MacBook Pro


THE SWIFT PROGRAMMING LANGUAGE
WELCOME TO SWIFT
About Swift
Version Compatibility
A Swift Tour
LANGUAGE GUIDE
LANGUAGE REFERENCE
REVISION HISTORY
← RETURN TO SWIFT.ORG
Tradition suggests that the first program in a new language should print the words “Hello, world!” on the screen. In Swift, this can be done in a single line:
print("Hello, world!")
If you have written code in C or Objective-C, this syntax looks familiar to you—in Swift, this line of code is a complete program. You don’t need to import a separate library for functionality like input/output or string handling. Code written at global scope is used as the entry point for the program, so you don’t need a main() function. You also don’t need to write semicolons at the end of every statement.
docs.swift.org
—everything introduced in this tour is explained in detail in the rest of this book.
NOTE
For the best experience, open this chapter as a playground in Xcode. Playgrounds allow you to edit the code listings and see the result immediately.
Download Playground
Simple Values
MacBook Pro

Apple Engagement in the Broader Swift Community

Many Swift-related conferences and podcasts

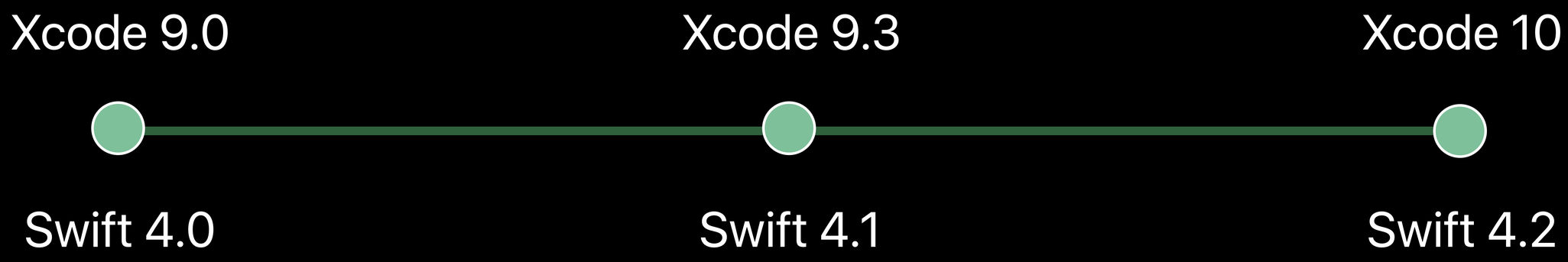
Apple Engagement in the Broader Swift Community

Event driven networking for Swift	try! Swift Tokyo
Implementing Swift Generics	LLVM Developers' Meeting
Creating Refactoring Transformations for Swift	Swift Summit
Swift's Reflective Underpinnings	Swift Summit
Extending the Standard Library	dotSwift
Swift 4.1 highlights	Swift Unwrapped

try! Swift San Jose
Friday, June 8

Swift 4.2

What is Swift 4.2?


Xcode 9.0
Xcode 9.3
Xcode 10
Swift 4.0
Swift 4.1
Swift 4.2

Faster builds

Language features to improve efficiency and remove boilerplate

SDK improvements for Swift

Converging towards binary compatibility


Swift 4.0
Swift 4.1
Swift 4.2

Swift 4.0

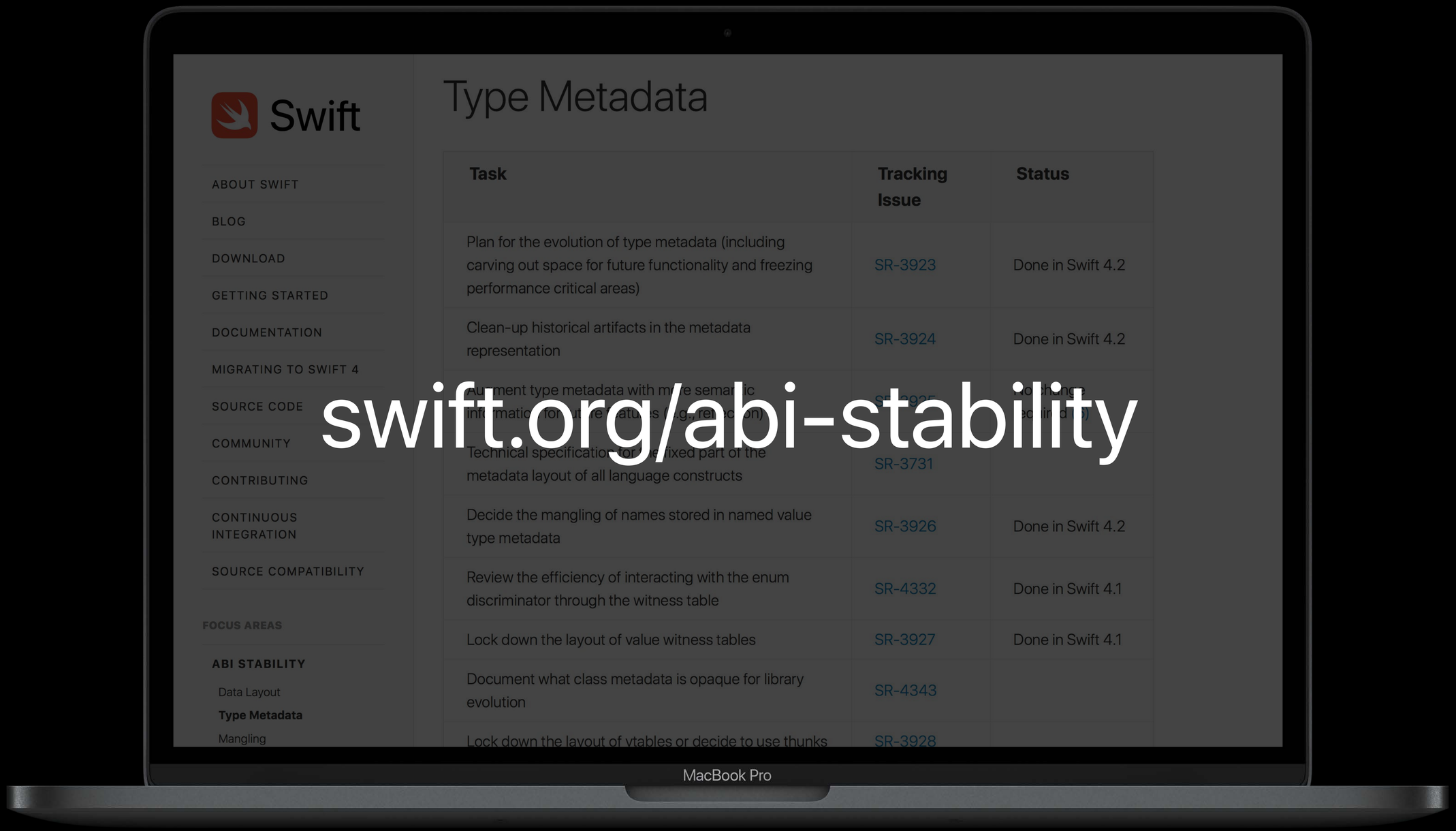
Swift 4.1

Swift 4.2

Swift 5
(early 2019)

Binary compatibility with future Swift compiler releases

Swift runtime ships in the OS

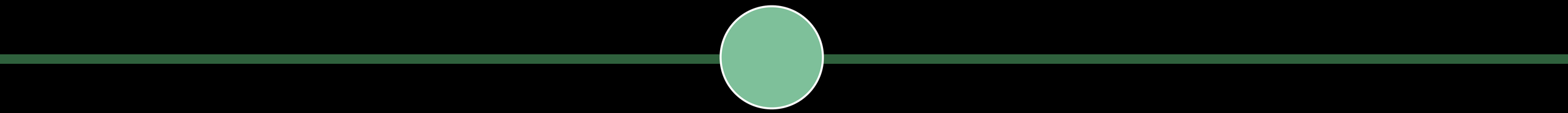

Swift
ABOUT SWIFT
BLOG
DOWNLOAD
GETTING STARTED
DOCUMENTATION
MIGRATING TO SWIFT 4
SOURCE CODE
COMMUNITY
CONTRIBUTING
CONTINUOUS INTEGRATION
SOURCE COMPATIBILITY
FOCUS AREAS
ABI STABILITY
Data Layout
Type Metadata
Mangling
Type Metadata
Task
Tracking Issue
Status
Plan for the evolution of type metadata (including carving out space for future functionality and freezing performance critical areas)
SR-3923
Done in Swift 4.2
Clean-up historical artifacts in the metadata representation
SR-3924
Done in Swift 4.2
Technical specification for the fixed part of the metadata layout of all language constructs
SR-3731
Decide the mangling of names stored in named value type metadata
SR-3926
Done in Swift 4.2
Review the efficiency of interacting with the enum discriminator through the witness table
SR-4332
Done in Swift 4.1
Lock down the layout of value witness tables
SR-3927
Done in Swift 4.1
Document what class metadata is opaque for library evolution
SR-4343
Lock down the layout of vtables or decide to use thunks
SR-3928
swift.org/abi-stability
MacBook Pro

Swift 4.0

Swift 4.1

Swift 4.2

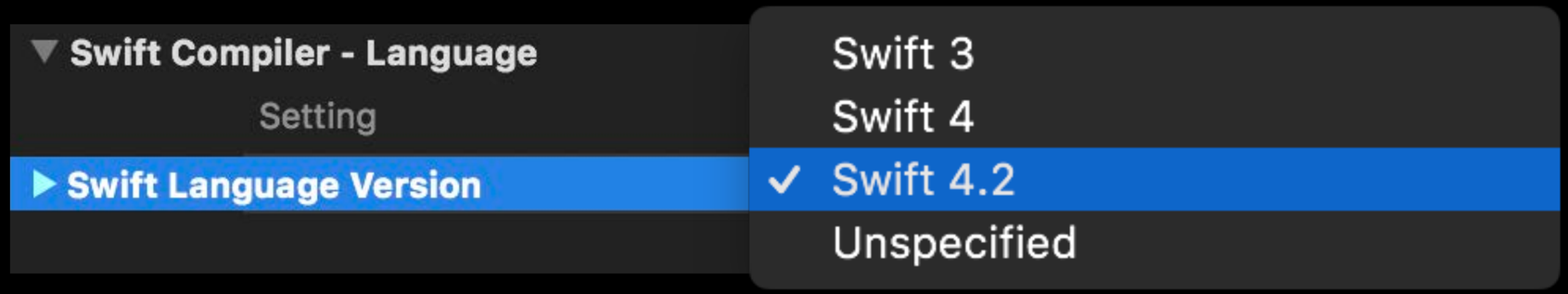
Swift 5
(early 2019)



Swift 4.2

Source Compatibility

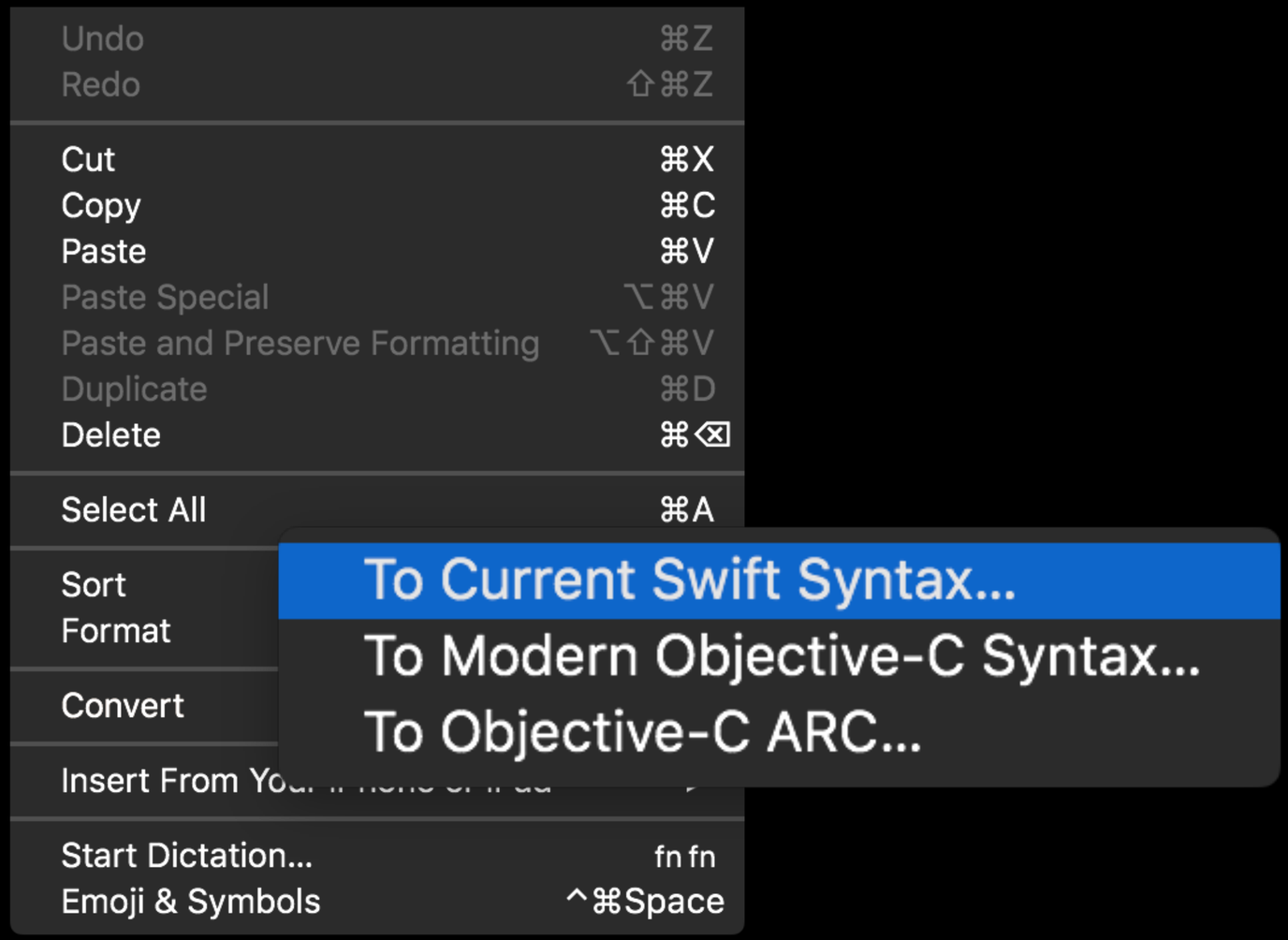
Source Compatibility: One Compiler with Three Modes



Source Compatibility: One Compiler with Three Modes

Swift 3	Accepts Swift 3 code that built with Xcode 8
Swift 4	Accepts Swift 4 code that built with Xcode 9.3
Swift 4.2	Same as with Swift 4 but incorporates new Swift-related SDK improvements

Migrator Support for Swift 4.2



Swift 4.2 SDK Changes

Updates to important framework APIs

Some API changes will land in later Xcode 10 betas

Trajectory with Source Compatibility

Source-impacting SDK changes converging

Last release to support “Swift 3” compatibility mode

Faster Swift Debug Builds

Swift Debug Builds

Build speeds of Xcode 10 compared to Xcode 9

1.6x

Faster build

2.4x

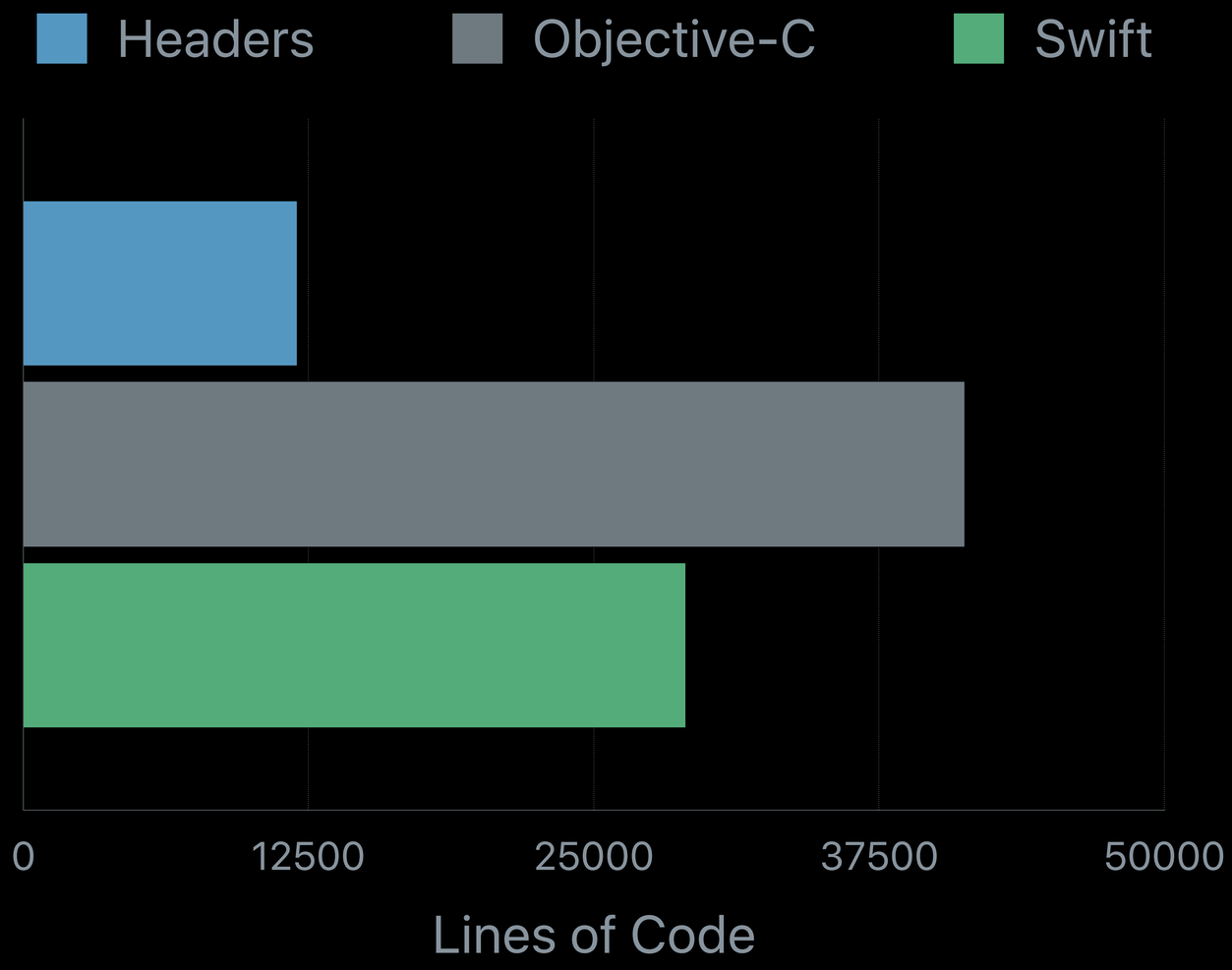
Faster build

2.1x

Faster build

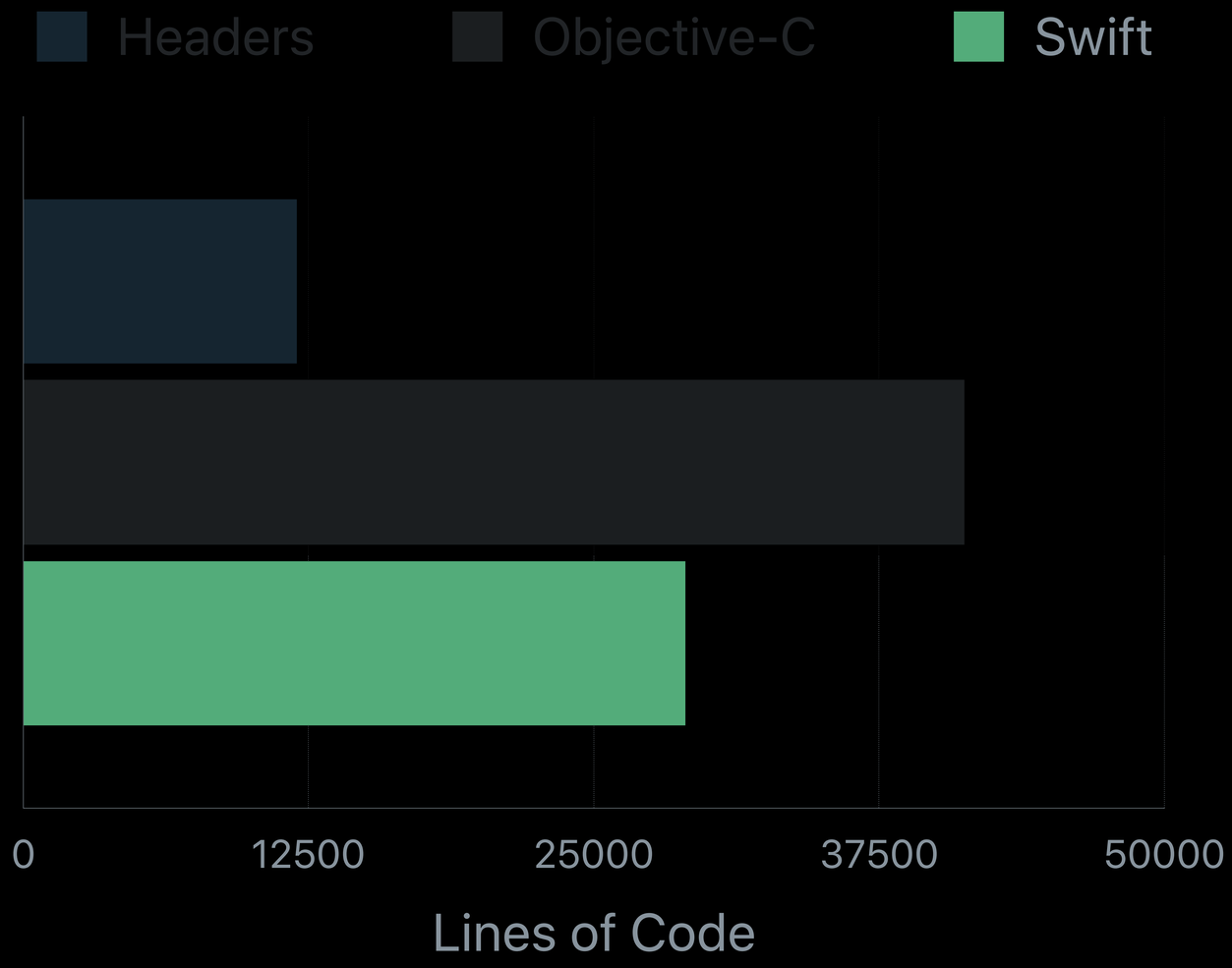
1.6x

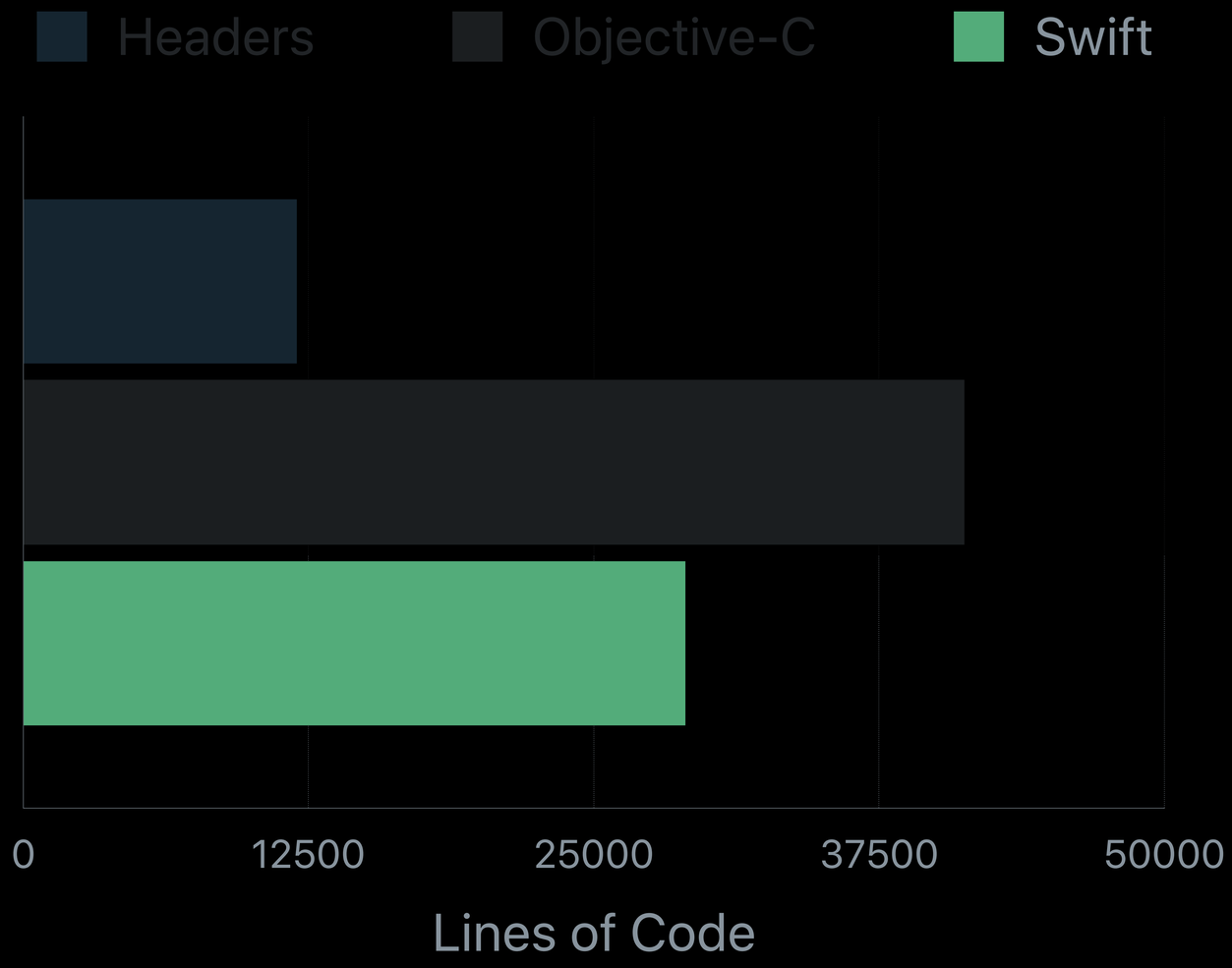
Faster build



1.6x

Faster build




Headers
Objective-C
Swift
0
12500
25000
37500
50000
Lines of Code

Speedup for Debug Builds

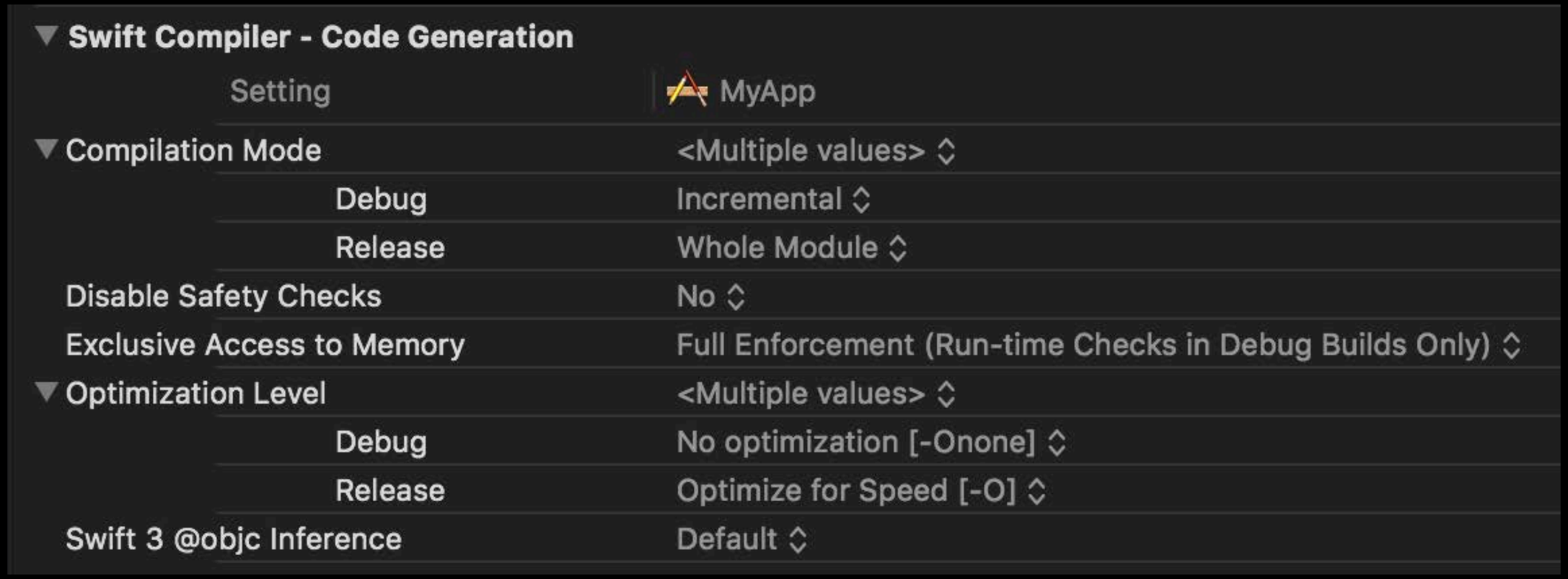
Exact speed increase varies by project size and number of cores

2x speedup of full builds for many projects

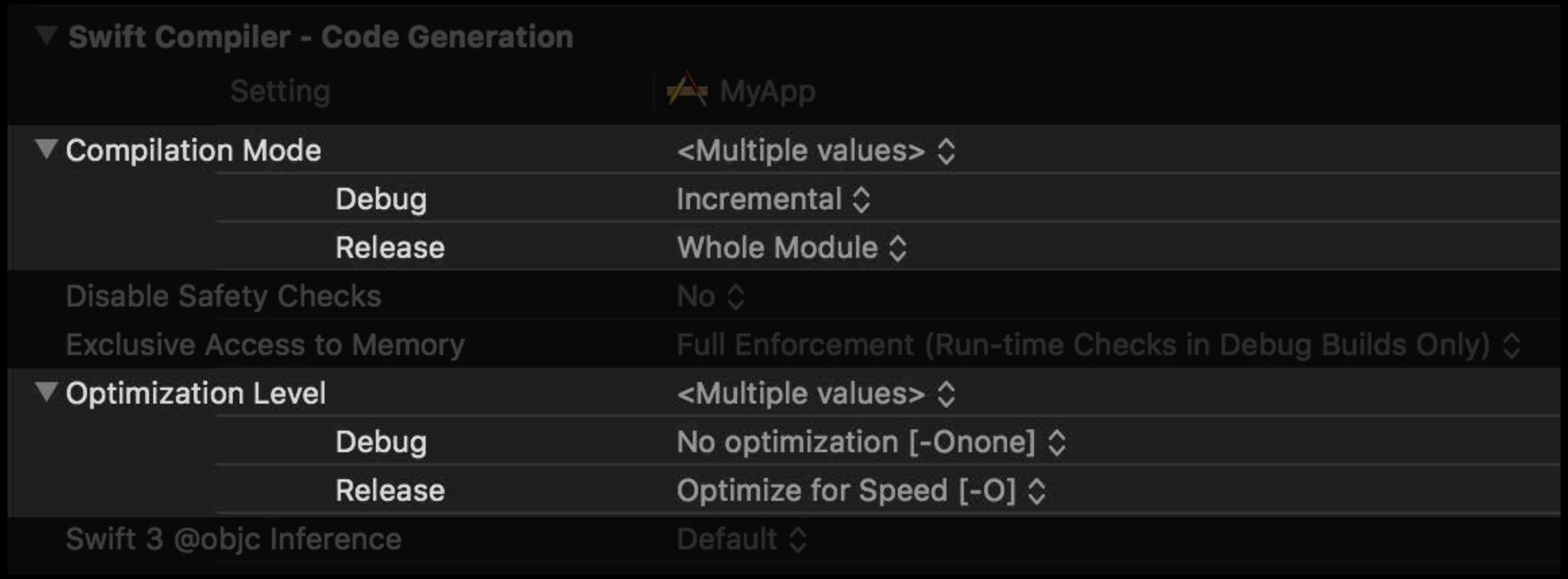
Compilation pipeline optimized to reduce redundant work across files

Building Faster in Xcode	Hall 1	Thursday 9:00AM
Behind the Scenes of the Xcode Build Process	Hall 2	Friday 2:00PM

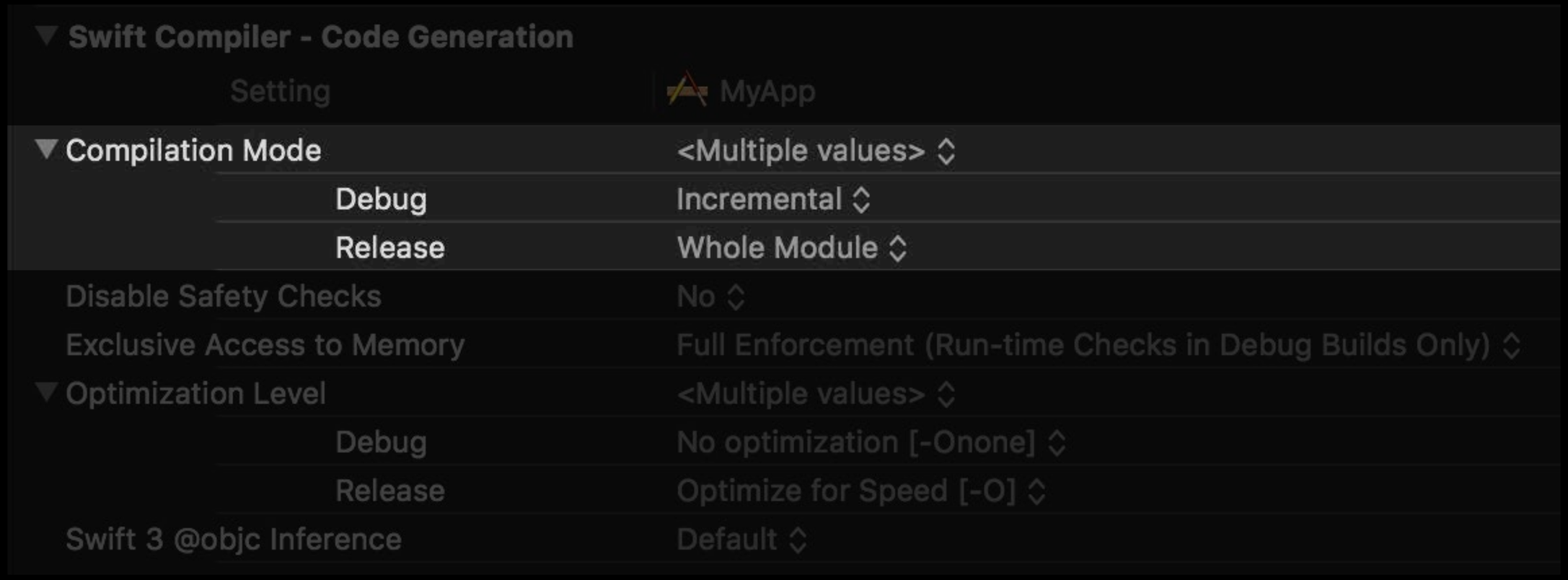
Compilation Mode versus Optimization Level



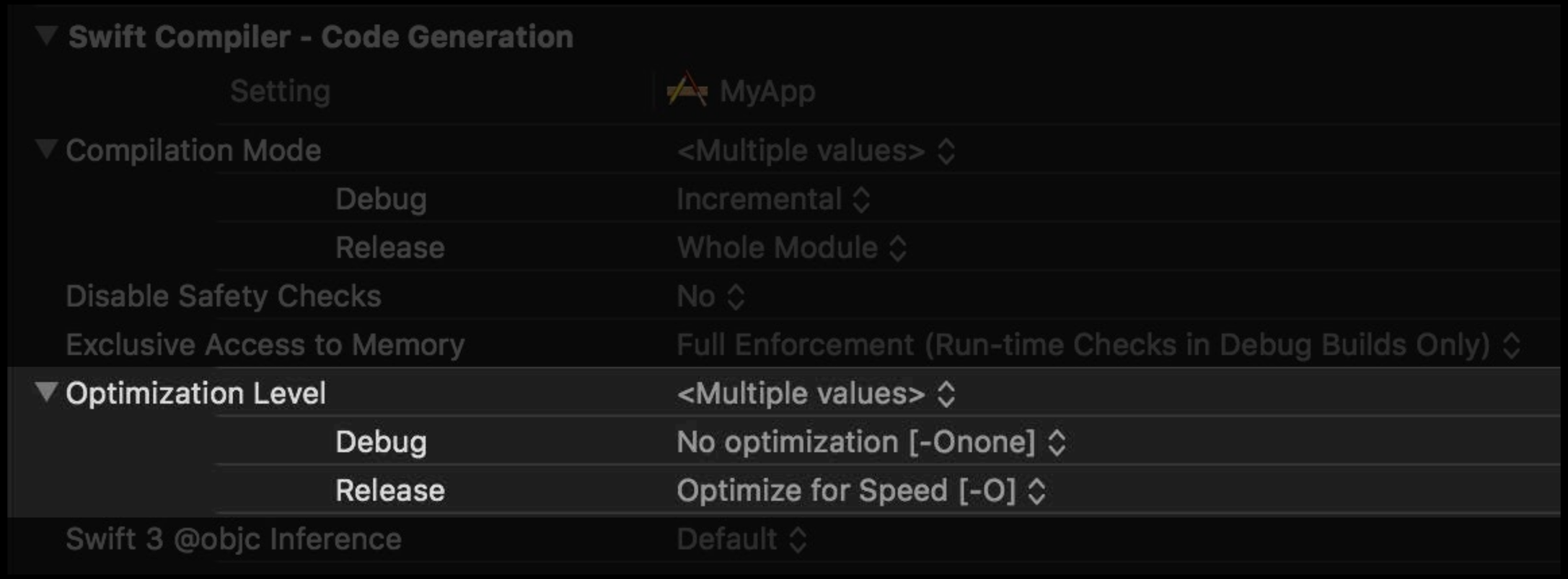
Compilation Mode versus Optimization Level



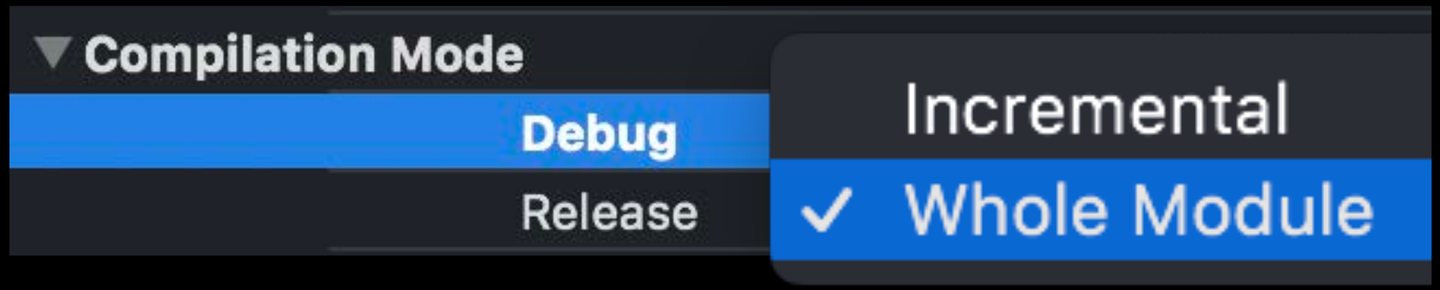
Compilation Mode versus Optimization Level



Compilation Mode versus Optimization Level



Stop Using Debug with Whole Module Compilation



Using **Whole Module** for **Debug** builds was a stopgap to improve builds

Whole Module prevents incremental builds

Use **Incremental** for Debug builds!

Runtime Optimizations

```
// Calling Convention: “Owned” (+1 retain)

class X { ... }

func caller() {
  // ‘x’ created with +1 reference count
  let x = X()
  foo(x)
}

func foo(x: X) {
  let y = x.value
  ...
  // release x
}
```

```
// Calling Convention: “Owned” (+1 retain)

class X { ... }

func caller() {
  // ‘x’ created with +1 reference count
  let x = X()
  foo(x)
}

func foo(x: X) {
  let y = x.value
  ...
  // release x
}
```

```
// Calling Convention: “Owned” (+1 retain)

class X { ... }

func caller() {
  // ‘x’ created with +1 reference count
  let x = X()
  // retain x
  bar(x) // release x in callee
  // retain x
  baz(x) // release x in callee
  foo(x) // release x in callee
}
```

```
// Calling Convention: “Owned” (+1 retain)

class X { ... }

func caller() {
  // ‘x’ created with +1 reference count
  let x = X()
  // retain x
  bar(x) // release x in callee
  // retain x
  baz(x) // release x in callee
  foo(x) // release x in callee
}
```

```
// Calling Convention: “Owned” (+1 retain)

class X { ... }

func caller() {
  // ‘x’ created with +1 reference count
  let x = X()
  // retain x
  bar(x) // release x in callee
  // retain x
  baz(x) // release x in callee
  foo(x) // release x in callee
}
```

```
// Calling Convention: “Owned” (+1 retain)

class X { ... }

func caller() {
  // ‘x’ created with +1 reference count
  let x = X()
  // retain x
  bar(x) // release x in callee
  // retain x
  baz(x) // release x in callee
  foo(x) // release x in callee
}
```

```
// Calling Convention: “Owned” (+1 retain)

class X { ... }

func caller() {
  // ‘x’ created with +1 reference count
  let x = X()
  // retain x
  bar(x) // release x in callee
  // retain x
  baz(x) // release x in callee
  foo(x) // release x in callee
}
```

```
// Calling Convention: “Guaranteed” (+0 retain)

class X { ... }

func caller() {
  // ‘x’ created with +1 reference count
  let x = X()
  // retain x
  bar(x) // release x in callee
  // retain x
  baz(x) // release x in callee
  foo(x) // release x in callee
}
```

```
// Calling Convention: “Guaranteed” (+0 retain)

class X { ... }

func caller() {
  // 'x' created with +1 reference count
  let x = X()
  // retain x
  bar(x) // release x in callee
  // retain x
  baz(x) // release x in callee
  foo(x) // release x in callee
}
```

```
// release x
```

```
// Calling Convention: “Guaranteed” (+0 retain)

class X { ... }

func caller() {
  // ‘x’ created with +1 reference count
  let x = X()
  bar(x)
  baz(x)
  foo(x)
  // release x
}
```

Small String

Strings now encoded using 16 bytes on 64-bit platforms

Small String

Use 15 bytes to represent the string directly when possible

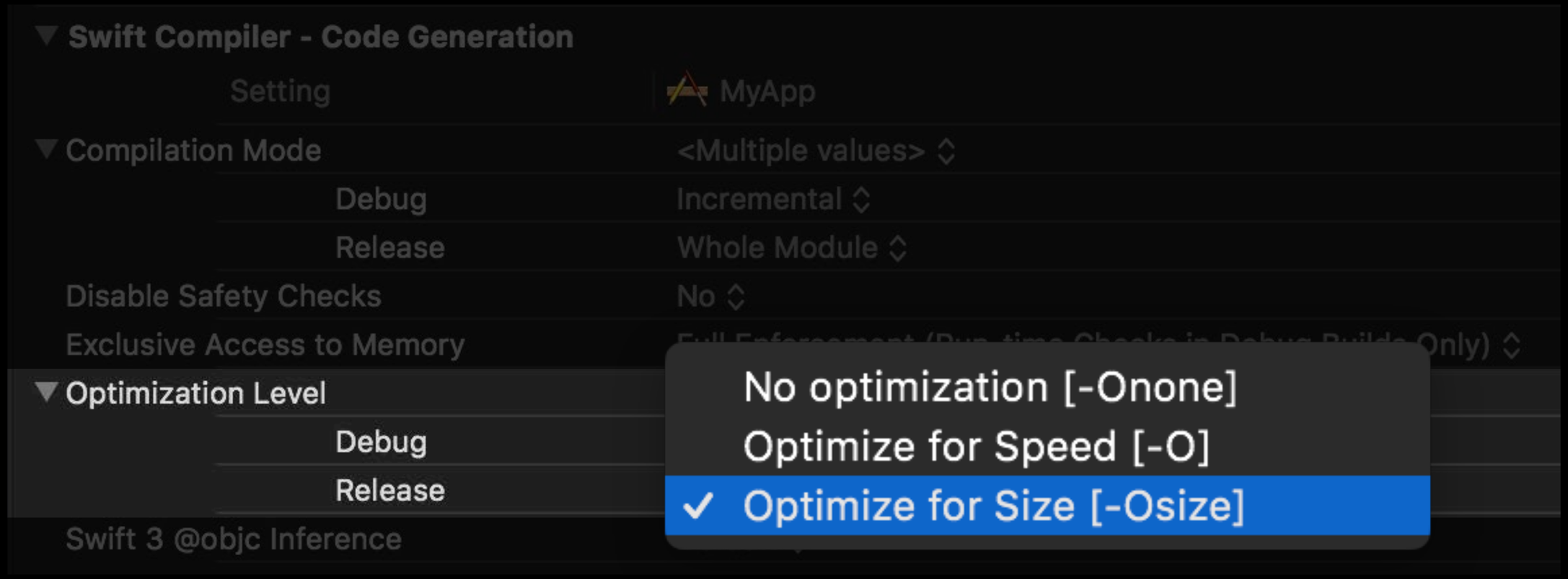
Sentinel indicates the payload is for a small string

No memory allocations for small strings

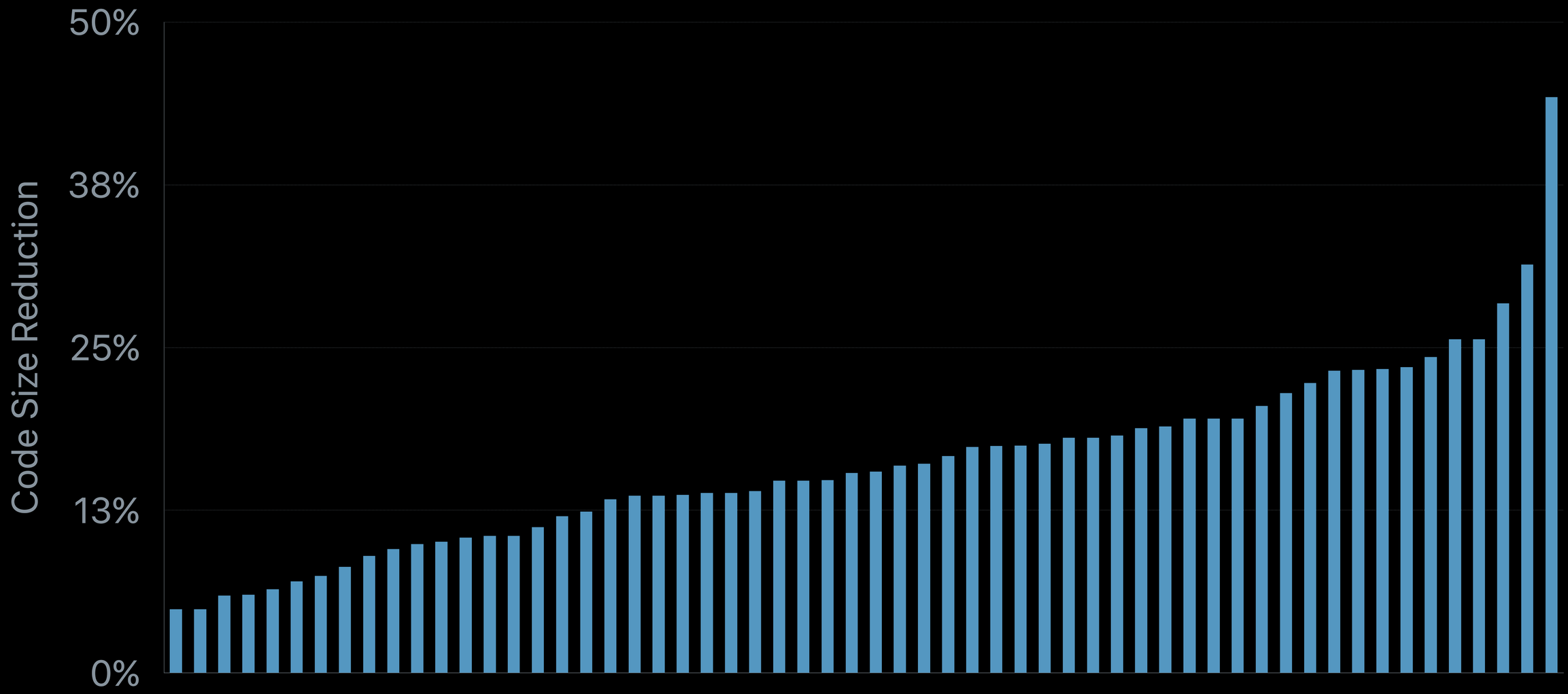
Similar to tagged NSString, but can represent slightly larger strings

Reduced Code Size

Optimize for Size



-Osize Results on Swift Source Compatibility Suite



Performance Tradeoffs of -Osize

Code size reduced by 10% to 30%

Runtime performance usually 5% slower

New Language Features

Slava Pestov, Swift Compiler Engineer

Swift Evolution Process

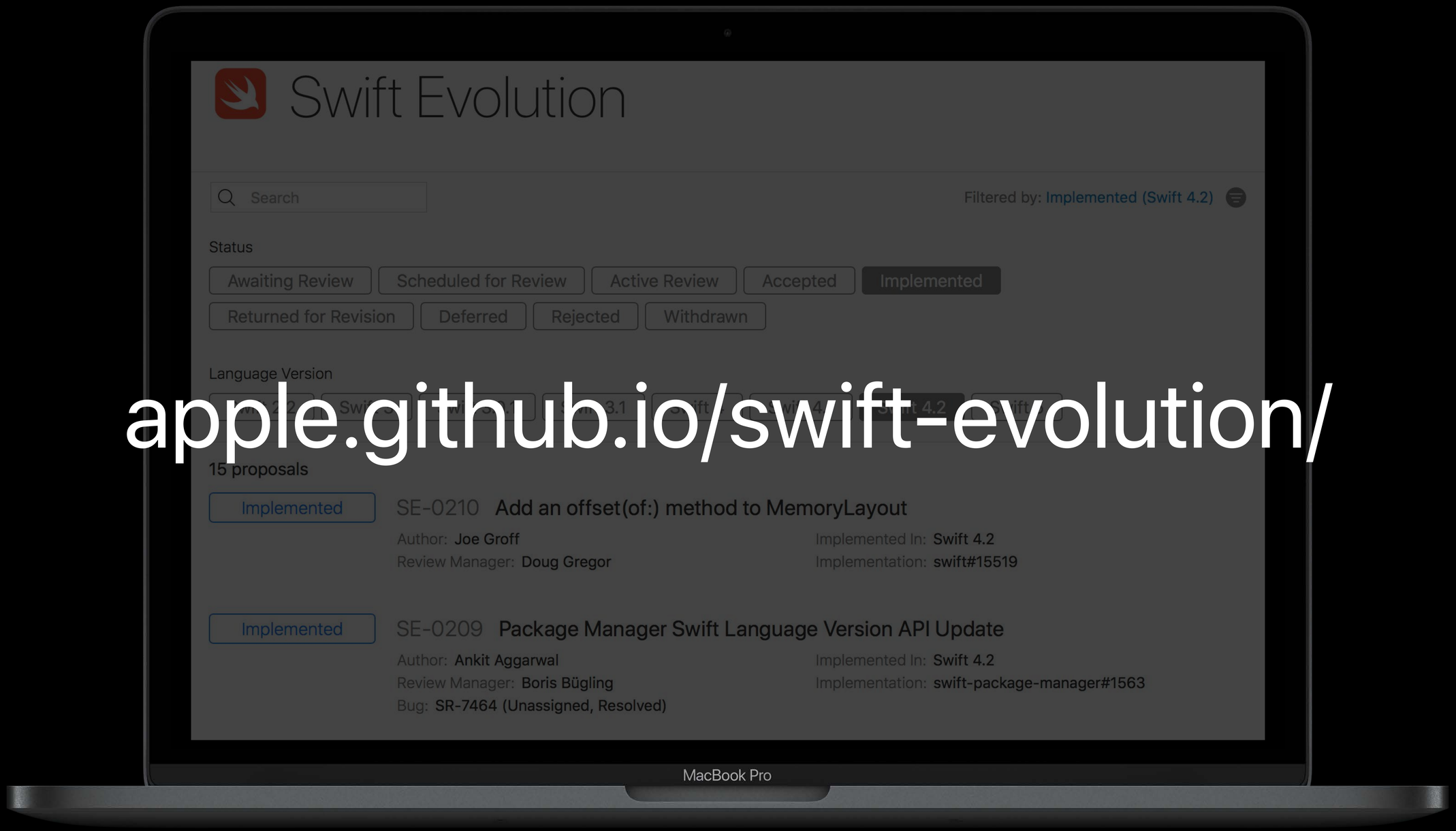
Pitch to forums.swift.org

Write draft proposal

Implementation

Review period

Decision by “core team”



apple.github.io/swift-evolution/

Implemented Proposals

SE-0075	Adding a Build Configuration Import Test
SE-0143	Conditional Conformances
SE-0157	Recursive Constraints on Associated Types
SE-0184	UnsafePointer Improvements
SE-0185	Synthesizing Equatable and Hashable Conformance
SE-0186	Remove Ownership Keyword Support in Protocols
SE-0187	Introduce Sequence.compactMap(_:)
SE-0188	Make Standard Library Index Types Hashable
SE-0189	Restrict Cross-Module Struct Initializers
SE-0190	Target Environment Platform Condition
SE-0191	Eliminate IndexDistance from Collection
SE-0192	Handling Future Enum Cases
SE-0193	Cross-Module Inlining and Specialization
SE-0194	Derived Collection of Enum Cases
SE-0195	Introduce ”Dynamic Member Lookup" Types
SE-0196	Compiler Diagnostic Directives
SE-0197	Adding In-Place removeAll(where:)
SE-0198	Playground QuickLook API Revamp
SE-0199	Adding toggle to Bool
SE-0202	Random Unification
SE-0204	Add last(where:) and lastIndex(where:) Methods
SE-0205	withUnsafePointer(to:_:) for Immutable Values
SE-0206	Hashable Enhancements
SE-0207	Add an allSatisfy Algorithm to Sequence
SE-0210	Add an offset(of:) Method to MemoryLayout

Implemented Proposals

SE-0075	Adding a Build Configuration Import Test	✓
SE-0143	Conditional Conformances	
SE-0157	Recursive Constraints on Associated Types	
SE-0184	UnsafePointer Improvements	✓
SE-0185	Synthesizing Equatable and Hashable Conformance	✓
SE-0186	Remove Ownership Keyword Support in Protocols	✓
SE-0187	Introduce Sequence.compactMap(_:)	
SE-0188	Make Standard Library Index Types Hashable	
SE-0189	Restrict Cross-Module Struct Initializers	
SE-0190	Target Environment Platform Condition	✓
SE-0191	Eliminate IndexDistance from Collection	
SE-0192	Handling Future Enum Cases	
SE-0193	Cross-Module Inlining and Specialization	
SE-0194	Derived Collection of Enum Cases	✓
SE-0195	Introduce ”Dynamic Member Lookup" Types	✓
SE-0196	Compiler Diagnostic Directives	✓
SE-0197	Adding In-Place removeAll(where:)	
SE-0198	Playground QuickLook API Revamp	
SE-0199	Adding toggle to Bool	✓
SE-0202	Random Unification	✓
SE-0204	Add last(where:) and lastIndex(where:) Methods	
SE-0205	withUnsafePointer(to:_:) for Immutable Values	
SE-0206	Hashable Enhancements	
SE-0207	Add an allSatisfy Algorithm to Sequence	
SE-0210	Add an offset(of:) Method to MemoryLayout	

✓ Community proposal and implementation

Collection of Enum Cases

```
// SE-0194 Derived Collection of Enum Cases

enum Gait {
  case walk
  case trot
  case canter
  case gallop
}
```

```swift
// SE-0194 Derived Collection of Enum Cases

enum Gait {
  case walk
  case trot
  case canter
  case gallop


  static var allCases: [Gait] = [.walk, .trot, .canter, .gallop]
}

for gait in Gait.allCases {
  print(gait)
}
```

```
// SE-0194 Derived Collection of Enum Cases

enum Gait {
  case walk
  case trot
  case canter
  case gallop
  case jog

  static var allCases: [Gait] = [.walk, .trot, .canter, .gallop]
}

for gait in Gait.allCases {
  print(gait)
}
```

This code will never print “jog”!

```
// SE-0194 Derived Collection of Enum Cases

enum Gait: CaseIterable {
  case walk
  case trot
  case canter
  case gallop
  case jog
}



for gait in Gait.allCases {
  print(gait)
}
```

Conditional Conformance

Inconsistent Behavior in Swift 4.0

```
extension Sequence where Element: Equatable {
  func contains(_ element: Element) -> Bool
}
```

Inconsistent Behavior in Swift 4.0

```
extension Sequence where Element: Equatable {
  func contains(_ element: Element) -> Bool
}

let animals = ["cat", "dog", "weasel"]
animals.contains("dog") // OK
```

Inconsistent Behavior in Swift 4.0

```
extension Sequence where Element: Equatable {
  func contains(_ element: Element) -> Bool
}

let animals = ["cat", "dog", "weasel"]
animals.contains("dog") // OK

let coins = [[1, 2], [3, 6], [4, 12]]
coins.contains([3, 6])
```

Error because the element type `[Int]` is not equatable

Why Aren't All Arrays Equatable?

```
let a: [(Int, Int) -> Int] = [{ $0 + $0 }, { 0 - $0 }]
let b: [(Int, Int) -> Int] = [{ $0 * 2 }, { -1 * $0 }]

a == b // Does not make sense
```

What If Element Is Equatable?

NEW

```
extension Array: Equatable where Element: Equatable {
  static func ==(lhs: Array<Element>, rhs: Array<Element>) -> Bool {









  }
}
```

What If Element Is Equatable?

NEW

```
extension Array: Equatable where Element: Equatable {
  static func ==(lhs: Array<Element>, rhs: Array<Element>) -> Bool {
    let count = lhs.count
    if count != rhs.count { return false }
    for x in 0..<count {
      if lhs[x] != rhs[x] { return false }
    }
    return true
  }
}
```

Conditional Conformance

NEW

```
let coins = [[1, 2], [3, 6], [4, 12]]
coins.contains([3, 6])  // This now works!
```

Conditional Conformance

NEW

```
extension Optional: Equatable where Wrapped: Equatable { … }
extension Array: Equatable where Element: Equatable { … }
extension Dictionary: Equatable where Value: Equatable { … }
```

Conditional Conformance

NEW

```
extension Optional: Hashable where Wrapped: Hashable { … }
extension Array: Hashable where Element: Hashable { … }
extension Dictionary: Hashable where Value: Hashable { … }
```

Conditional Conformance

NEW

```
extension Optional: Encodable where Wrapped: Encodable { … }
extension Array: Encodable where Element: Encodable { … }
extension Dictionary: Encodable where Key: Encodable, Value: Encodable { … }
```

Conditional Conformance

NEW

```
extension Optional: Decodable where Wrapped: Decodable { … }
extension Array: Decodable where Element: Decodable { … }
extension Dictionary: Decodable where Key: Decodable, Value: Decodable { … }
```

Conditional Conformance Allows Composition

NEW

```
let s: Set<[Int?]> = [[1, nil, 2], [3, 4], [5, nil, nil]]
// Int is Hashable
// => Int? is Hashable
// => [Int?] is Hashable
```

Synthesized Equatable and Hashable

```
// SE-0185 Synthesizing Equatable and Hashable Conformance

struct Restaurant {
  let name: String
  let hasTableService: Bool
  let kidFriendly: Bool
}
```

```
// SE-0185 Synthesizing Equatable and Hashable Conformance

struct Restaurant {
  let name: String
  let hasTableService: Bool
  let kidFriendly: Bool
}

extension Restaurant: Equatable {
  static func ==(a: Restaurant, b: Restaurant) -> Bool {
    return a.name == b.name &&
           a.hasTableService == b.hasTableService &&
           a.kidFriendly == b.kidFriendly
  }
}
```

```
// SE-0185 Synthesizing Equatable and Hashable Conformance

struct Restaurant: Equatable {
  let name: String
  let hasTableService: Bool
  let kidFriendly: Bool
}
```

NEW

```
// SE-0185 Synthesizing Equatable and Hashable Conformance

struct Restaurant: Hashable {
  let name: String
  let hasTableService: Bool
  let kidFriendly: Bool
}
```

NEW

```
// Synthesizing Conditional Equatable and Hashable

enum Either<Left, Right> {
  case left(Left)
  case right(Right)
}
```

```swift
// Synthesizing Conditional Equatable and Hashable

enum Either<Left, Right> {
  case left(Left)
  case right(Right)
}

extension Either: Equatable where Left: Equatable, Right: Equatable {
  static func ==(a: Either<Left, Right>, b: Either<Left, Right>) {
    switch (a, b) {
      case (.left(let x), .left(let y)): return x == y
      case (.right(let x), .right(let y)): return x == y
      default: return false
    }
  }
}
```

```
// Synthesizing Conditional Equatable and Hashable

enum Either<Left, Right> {
  case left(Left)
  case right(Right)
}

extension Either: Equatable where Left: Equatable, Right: Equatable { }
```

NEW

```
// Synthesizing Conditional Equatable and Hashable

enum Either<Left, Right> {
  case left(Left)
  case right(Right)
}

extension Either: Equatable where Left: Equatable, Right: Equatable { }
extension Either: Hashable where Left: Hashable, Right: Hashable { }
```

NEW

NEW

```
// Synthesizing Conditional Equatable and Hashable

enum Either<Left, Right> {
  case left(Left)
  case right(Right)
}

extension Either: Equatable where Left: Equatable, Right: Equatable { }
extension Either: Hashable where Left: Hashable, Right: Hashable { }

// This just works!
var mySet = Set<Either<Int, String>>()
```

Hashable Enhancements

```swift
// Implementing Hashable by Hand in Swift 4.1

struct City {
  let name: String
  let state: String
  let population: Int
}
```

```swift
// Implementing Hashable by Hand in Swift 4.1

struct City {
  let name: String
  let state: String
  let population: Int
}

extension City: Equatable {
  static func ==(a: City, b: City) -> Bool {
    return a.name == b.name && a.state == b.state
  }
}
```

```swift
// Implementing Hashable by Hand in Swift 4.1

struct City {
  let name: String
  let state: String
  let population: Int
}

extension City: Hashable {
  var hashValue: Int {
    return name.hashValue ??? state.hashValue
  }
}
```

```swift
// Implementing Hashable by Hand in Swift 4.1

struct City {
  let name: String
  let state: String
  let population: Int
}

extension City: Hashable {
  var hashValue: Int {
    return name.hashValue ^ state.hashValue
  }
}
```

```swift
// Implementing Hashable by Hand in Swift 4.1

struct City {
  let name: String
  let state: String
  let population: Int
}

extension City: Hashable {
  var hashValue: Int {
    return (name.hashValue &* 58374501) &+ state.hashValue
  }
}
```

Hash Combining Functions Are Hard

Too much magic

Performance problems

Denial of service attacks

Need a better API!

```swift
// Hashable Protocol in Swift 4.1

protocol Hashable {
  var hashValue: Int { get }
}
```

```swift
// Hashable Protocol in Swift 4.2

protocol Hashable {
  func hash(into hasher: inout Hasher)
}
```

```
// Using the Hashable Protocol

extension City: Hashable {
  func hash(into hasher: inout Hasher) {
    name.hash(into: &hasher)
    state.hash(into: &hasher)
  }
}
```

New Hashing Algorithm

Balances hash quality with performance

Random per-process seed

Hash Values Vary From Run to Run

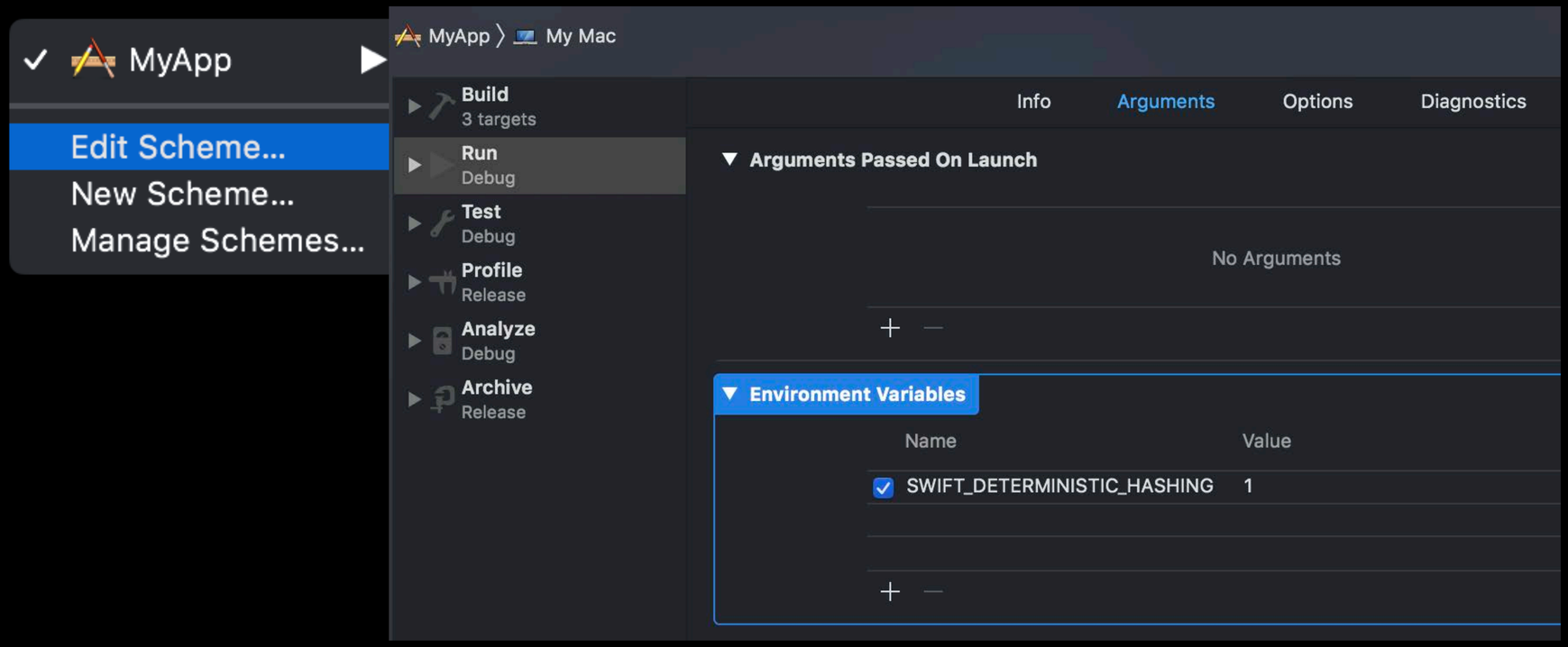
Fix any code that relies on:

- Specific hash values
- `Set` or `Dictionary` iteration order

Hash Values Vary From Run to Run

For debugging problems:

- Set `SWIFT_DETERMINISTIC_HASHING=1` in the scheme editor



Random Number Generation

Random Number Generation in Swift 4.0

Functionality provided through imported C APIs

Requires platform checks

```
#if os(iOS) || os(tvOS) || os(watchOS) || os(macOS)
  return Int(arc4random())
#else
  return random() // or Int(rand())
#endif
```

Random Number Generation in Swift 4.0

Functionality provided through imported C APIs

Common operations are tricky

```
// Return random number in the range 1...6
func diceRoll() -> Int {
  return 1 + (arc4random() % 6)
}
```

Biased: 1, 2, 3, 4 are more likely than 5, 6

```
// SE-0202 Random Unification

let randomIntFrom0To10 = Int.random(in: 0 ..< 10)
let randomFloat = Float.random(in: 0 ..< 1)
```

```
// SE-0202 Random Unification

let randomIntFrom0To10 = Int.random(in: 0 ..< 10)
let randomFloat = Float.random(in: 0 ..< 1)

let greetings = ["hey", "hi", "hello", "hola"]
print(greetings.randomElement()!)
```

```
// SE-0202 Random Unification

let randomIntFrom0To10 = Int.random(in: 0 ..< 10)
let randomFloat = Float.random(in: 0 ..< 1)

let greetings = ["hey", "hi", "hello", "hola"]
print(greetings.randomElement()!)

let randomlyOrderedGreetings = greetings.shuffled()
print(randomlyOrderedGreetings)
```

Default RNG Implementation

Secure RNG on both Apple platforms and Linux

```
// Defining Your Own RNG Algorithm

struct MersenneTwister: RandomNumberGenerator { … }

var mt = MersenneTwister()
```

```
// Defining Your Own RNG Algorithm

struct MersenneTwister: RandomNumberGenerator { … }

var mt = MersenneTwister()

let randomIntFrom0To10 = Int.random(in: 0 ..< 10, using: &mt)
let randomFloat = Float.random(in: 0 ..< 1, using: &mt)

let greetings = ["hey", "hi", "hello", "hola"]
print(greetings.randomElement()!, using: &mt)

let randomlyOrderedGreetings = greetings.shuffled(using: &mt)
print(greetings)
```

Checking Platform Conditions

```
// SE-0075 Adding a Build Configuration Import Test

#if os(iOS) || os(watchOS) || os(tvOS)
  import UIKit
  …
#else
  import AppKit
  …
#endif
```

```
// SE-0075 Adding a Build Configuration Import Test

#if canImport(UIKit)
  import UIKit
  …
#else
  import AppKit
  …
#endif
```

```
// SE-0075 Adding a Build Configuration Import Test

#if canImport(UIKit)
  import UIKit
  …
#elseif canImport(AppKit)
  import AppKit
  …
#else
  #error("Unsupported platform")
#endif
```

```
// SE-0190 Target Environment Platform Condition

#if (os(iOS) || os(watchOS) || os(tvOS)) &&
    (cpu(i386) || cpu(x86_64))
  …
#else
  // FIXME: We need to test this better
  …
#endif
```

```
// SE-0190 Target Environment Platform Condition

#if hasTargetEnvironment(simulator)

  …
#else
  // FIXME: We need to test this better
  …
#endif
```

```
// SE-0190 Target Environment Platform Condition

#if hasTargetEnvironment(simulator)

  …
#else
  #warning("We need to test this better")
  …
#endif
```

Implicitly Unwrapped Optionals

Mental Model

IUO is an *attribute* of a declaration, not a *type* of an expression

First, try type checking value as `T?` — otherwise, force unwrap to get `T`

Value Type Checks as Type T?

Optional `Int` can be stored inside `Any`

Force unwrapping is not performed

```
func computeDangerously(_ b: Bool) -> Int! { return b ? 3 : nil }
func takesAnAny(_ x: Any) { print(x) }

takesAnAny(computeDangerously(.random))
```

Value Type Checks as Type T

Optional `Int` is not allowed

Must force unwrap the result of the call

```
func computeDangerously(_ b: Bool) -> Int! { return b ? 3 : nil }
func takesAnInt(_ x: Int) { print(x) }

takesAnInt(computeDangerously(.random))
```

Value Type Checks as Type T

Optional `Int` is not allowed

Must force unwrap the result of the call

```
func computeDangerously(_ b: Bool) -> Int! { return b ? 3 : nil }
func takesAnInt(_ x: Int) { print(x) }

takesAnInt(computeDangerously(.random)!)
```

Corner Cases

IUO not allowed as part of another type:

```
let array: [Int!] = [3]
print(array[0])
```

Implicitly unwrapped optionals are only allowed at top level and as function results

Corner Cases

Not enforced consistently in Swift 4.0:

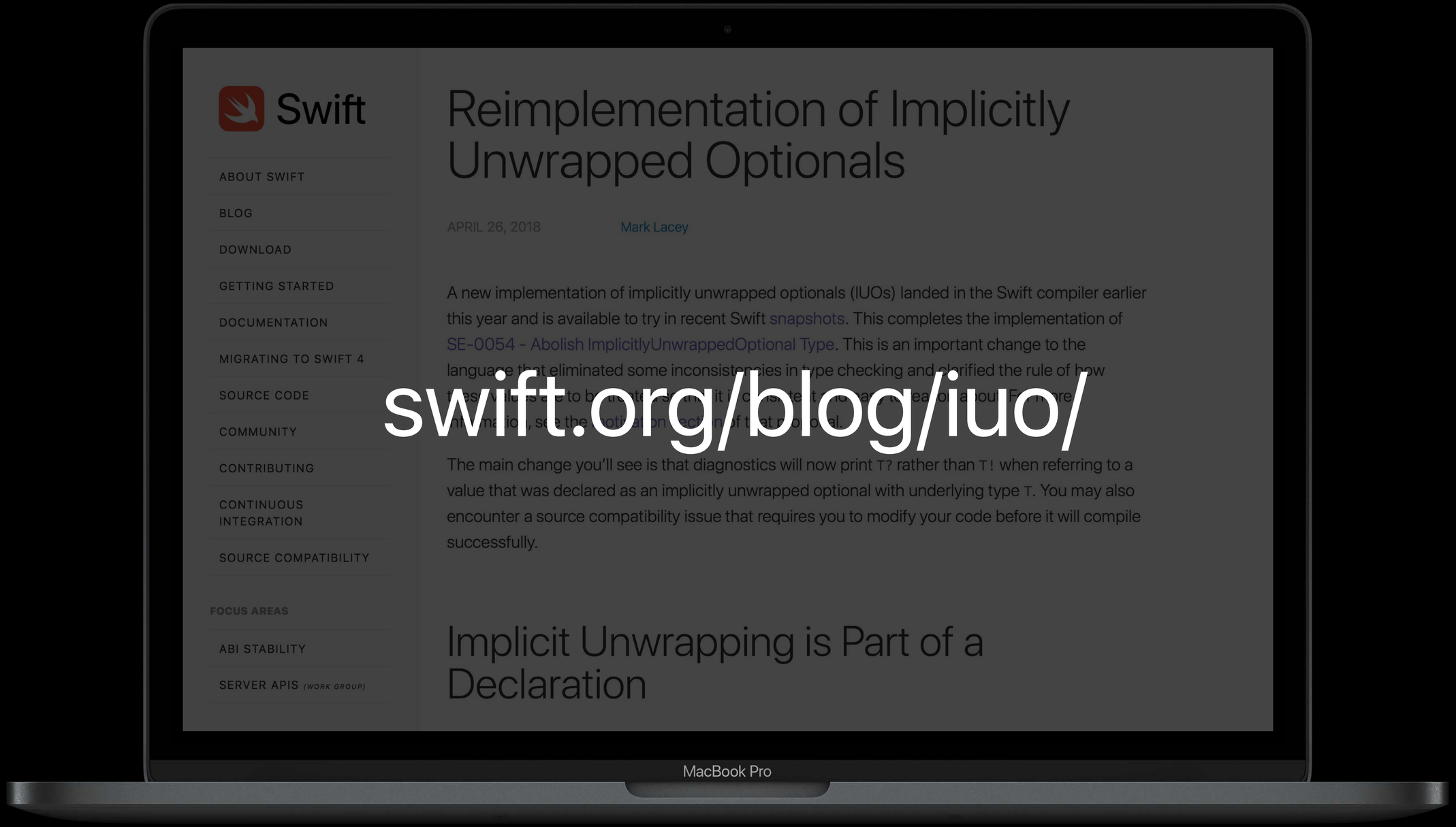
```
typealias T = Int!
let array: [T] = [3]
print(array[0])
```

Corner Cases

IUO in invalid position becomes plain Optional in Swift 4.2:

```
typealias T = Int!
let array: [T] = [3]
print(array[0])
```

Using '!' is not allowed here; treating this as '?' instead


Swift
ABOUT SWIFT
BLOG
DOWNLOAD
GETTING STARTED
DOCUMENTATION
MIGRATING TO SWIFT 4
SOURCE CODE
COMMUNITY
CONTRIBUTING
CONTINUOUS INTEGRATION
SOURCE COMPATIBILITY
FOCUS AREAS
ABI STABILITY
SERVER APIS (WORK GROUP)
Reimplementation of Implicitly Unwrapped Optionals
APRIL 26, 2018
Mark Lacey
A new implementation of implicitly unwrapped optionals (IUOs) landed in the Swift compiler earlier this year and is available to try in recent Swift snapshots. This completes the implementation of SE-0054 - Abolish ImplicitlyUnwrappedOptional Type. This is an important change to the
swift.org/blog/iuo/
The main change you'll see is that diagnostics will now print T? rather than T! when referring to a value that was declared as an implicitly unwrapped optional with underlying type T. You may also encounter a source compatibility issue that requires you to modify your code before it will compile successfully.
Implicit Unwrapping is Part of a Declaration
MacBook Pro

Enforcing Exclusive Access to Memory

SE-0176 Enforce Exclusive Access to Memory

Overlapping access to the same memory location not allowed

Combination of static and dynamic checks

```
// Example of Exclusive Access Violation

struct Path {
  var components: [String] = []

  mutating func withAppended(_ name: String, _ closure: () -> Void) {
    components.append(name)
    closure()
    components.removeLast()
  }
}
```

```
// Example of Exclusive Access Violation

struct Path {
  var components: [String] = []

  mutating func withAppended(_ name: String, _ closure: () -> Void) {
    components.append(name)
    closure()
    components.removeLast()
  }
}

var path = Path(components: ["usr", "local"])
path.withAppended("bin") { print(path) }
```

```
// Example of Exclusive Access Violation

struct Path {
  var components: [String] = []

  mutating func withAppended(_ name: String, _ closure: () -> Void) {
    components.append(name)
    closure()
    components.removeLast()
  }
}

var path = Path(components: ["usr", "local"])
path.withAppended("bin") { print(path) }
```

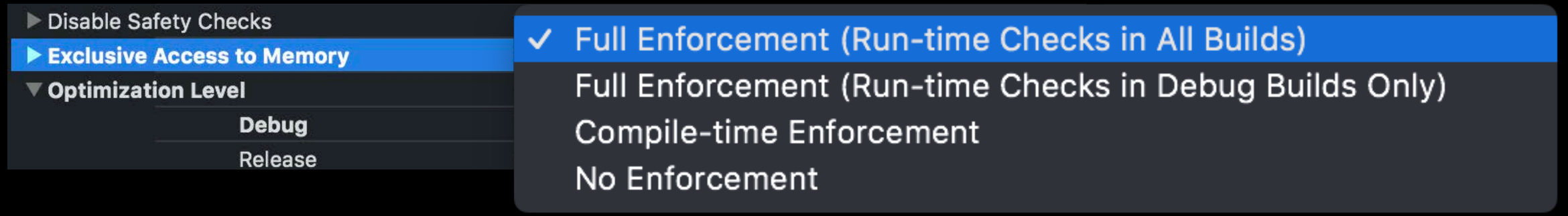
```
// Addressing the Exclusive Access Violation

struct Path {
  var components: [String] = []

  mutating func withAppended(_ name: String, _ closure: (Path) -> Void) {
    components.append(name)
    closure(self)
    components.removeLast()
  }
}

var path = Path(components: ["usr", "local"])
path.withAppended("bin", { print($0) })
```

```swift
// More Complex Violation of Exclusive Access

struct Path {
  var components: [String] = []

  mutating func withAppended<T>(_ name: String, _ closure: () -> T) -> T {
    components.append(name)
    let result = closure()
    components.removeLast()

    return result
  }
}
```

```swift
// More Complex Violation of Exclusive Access

struct Path {
  var components: [String] = []

  mutating func withAppended<T>(_ name: String, _ closure: () -> T) -> T {
    components.append(name)
    let result = closure()
    components.removeLast()

    return result
  }
}

var path = Path(components: ["usr", "local"])
path.withAppended("bin") { print(path) }
```

```
// Addressing More Complex Violation of Exclusive Access

struct Path {
  var components: [String] = []

  mutating func withAppended<T>(_ name: String, _ closure: (Path) -> T) -> T {
    components.append(name)
    let result = closure(self)
    components.removeLast()

    return result
  }
}

var path = Path(components: ["usr", "local"])
path.withAppended("bin", { print($0) })
```

Dynamic Checks Now Available in Release Builds



Adding In-place removeAll(where:)

Optimize for Size

Add an allSatisfy Algorithm to Sequence

Dynamic Member Lookup

`toggle` method on `Bool`

Small String

Add an offset(of:) Method to MemoryLayout

Refinement of Exclusive Access

#if hasTargetEnvironment(simulator)

Conditional Conformances

UnsafePointer Improvements

`CaseIterable`

Eliminate IndexDistance from Collection

Unification of Random

Cross-Module Inlining and Specialization

Faster Incremental Builds

Refinement of IUO

Hashable Index Types

`#error`

`#if canImport(module)`

`#warning`

Add last(where:) and lastIndex(where:) Methods

Synthesized Equatable and Hashable

Hashable Enhancements

“Guaranteed” Calling Convention

Introduce Sequence.compactMap(_:)

Recursive Value Type Metadata

withUnsafePointer(to:_:) for Immutable Values

More Information

https://developer.apple.com/wwdc18/401


WWDC18