

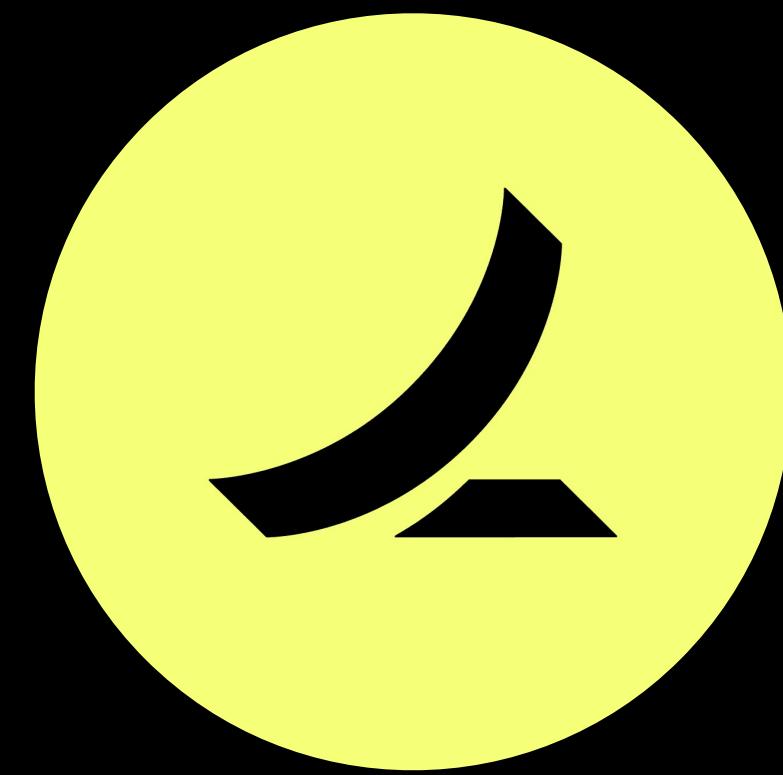
Beyond Xcode

Batteries Not Included

hats



Computer Science
University of Waterloo

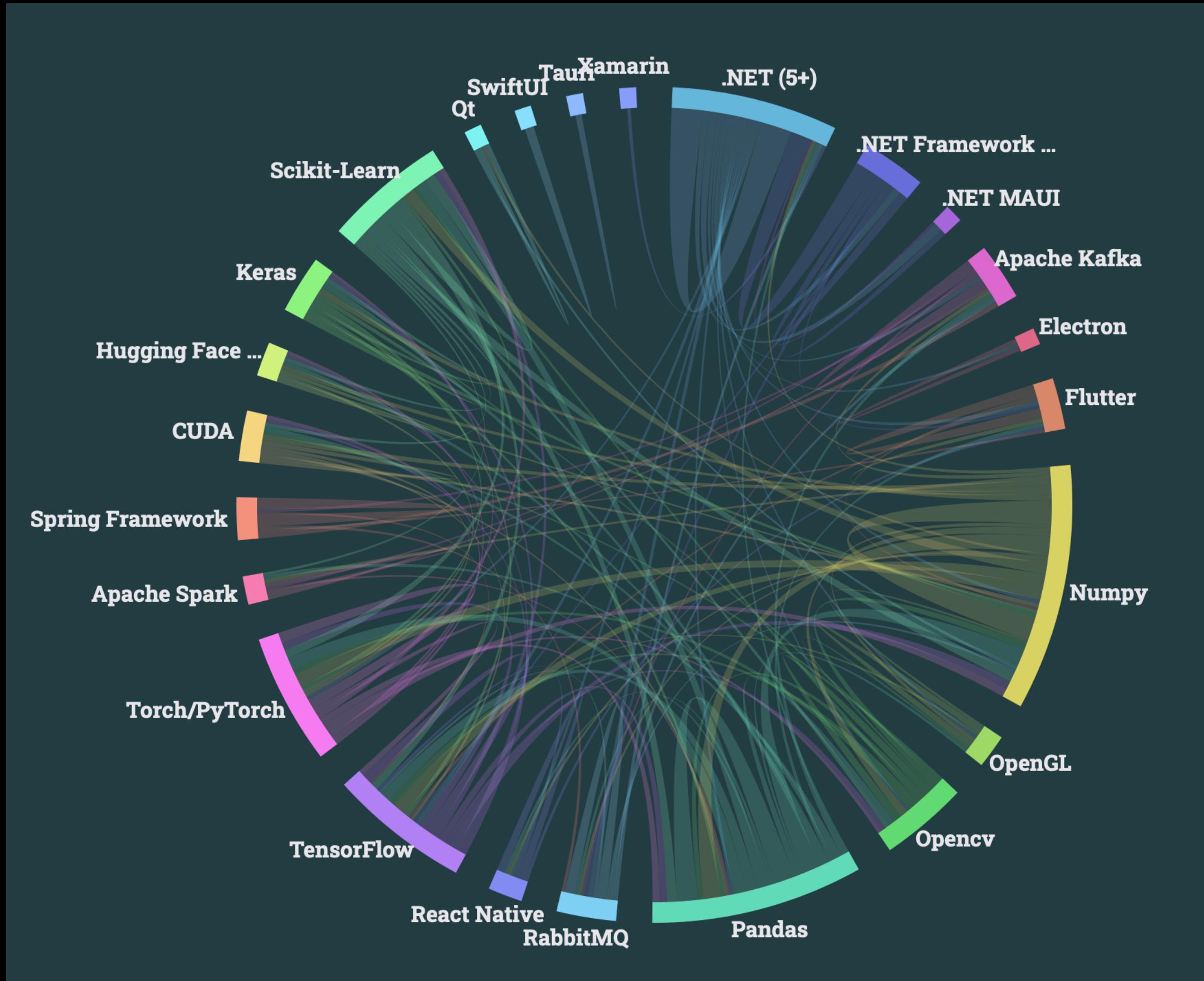


iOS Developer
Ramp



Maintainer
Theos

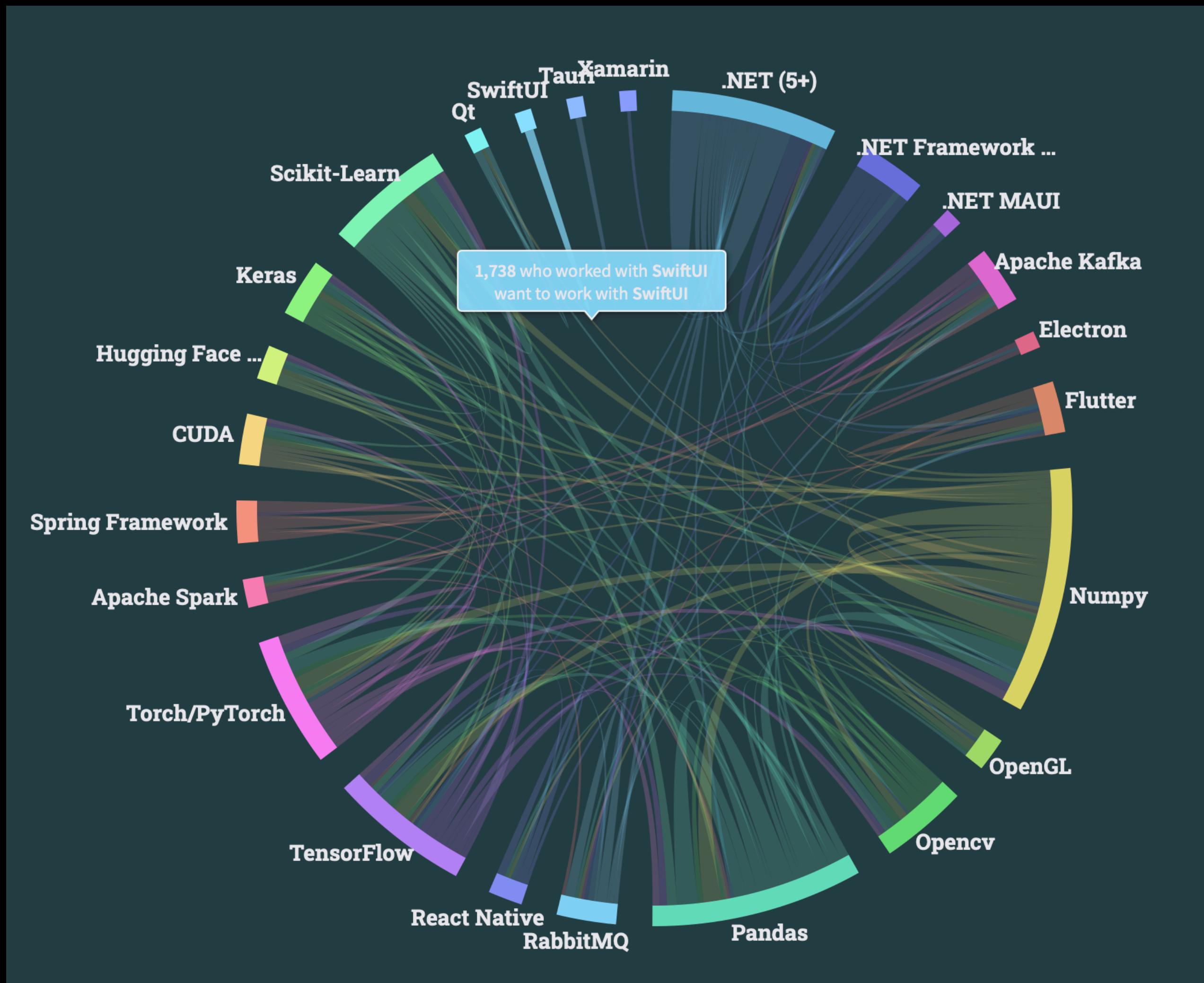
we ❤️ swift(ui)



Which [non-web] **frameworks and libraries** have you done extensive development work in over the past year, and which do you want to work in over the next year?

(StackOverflow 2023 Developer Survey)

we ❤️ swift(ui)

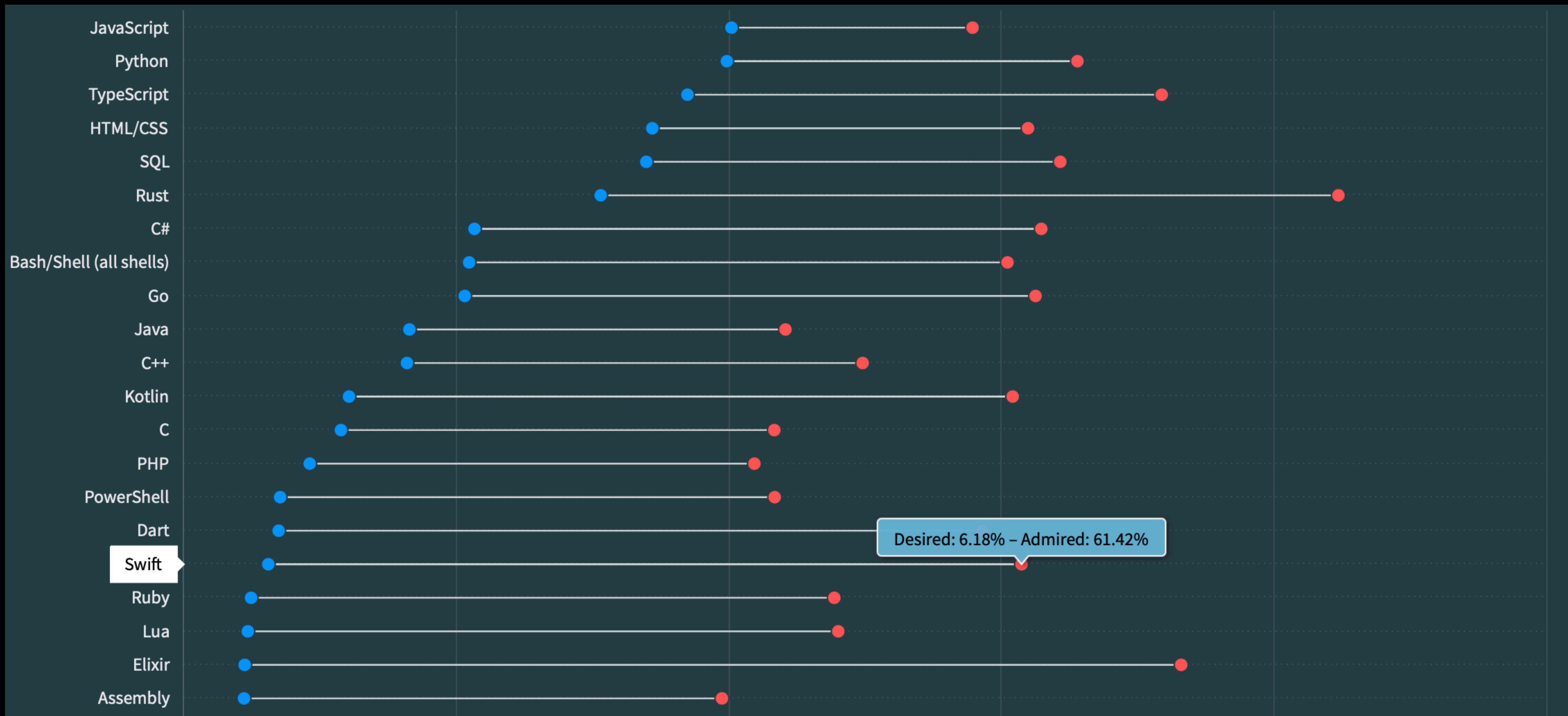


Which [non-web] frameworks and libraries have you done extensive development work in over the past year, and which do you want to work in over the next year?

(StackOverflow 2023 Developer Survey)



...but we're siloed



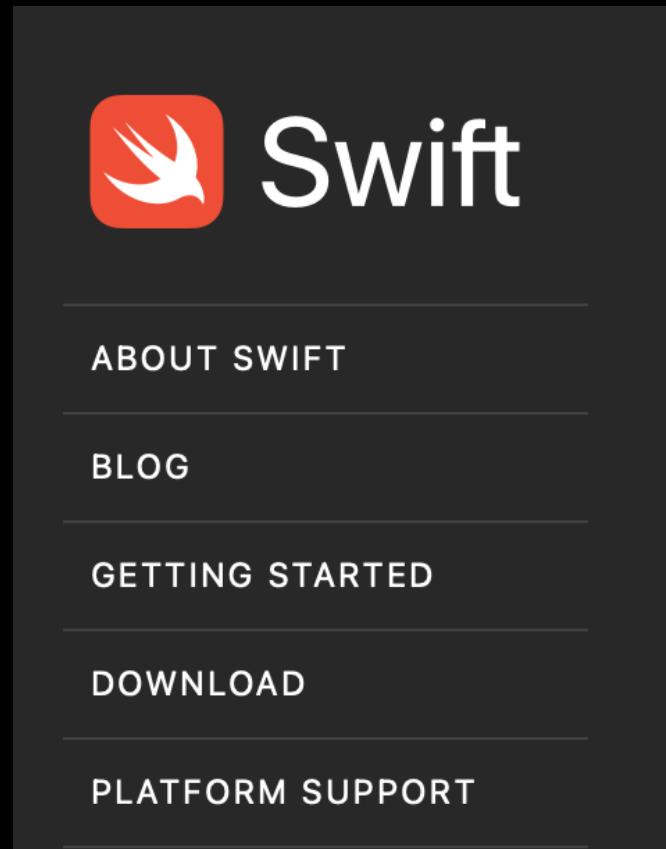
Which **programming, scripting, and markup languages** have you done extensive development work in over the past year, and which do you want to work in over the next year? (StackOverflow 2023 Developer Survey)

why?

“Swift is for Apple devices.”

why?

“Swift is for Apple devices.”



Welcome to Swift.org

Welcome to the Swift community. Together we are working to build a programming language to empower everyone to turn their ideas into apps on any platform.

Announced in 2014, the Swift programming language has quickly become one of the fastest growing languages in history. Swift makes it easy to write software that is incredibly fast and safe by design. Our goals for Swift are ambitious: we want to make programming simple things easy, and difficult things possible.

A screenshot of a GitHub repository page for "swift / README.md". The page title is "Welcome to Swift". The text describes Swift as a high-performance system programming language with a clean syntax, seamless access to existing C and Objective-C code and frameworks, and memory-safety by default. There are buttons for "Preview", "Code", and "Blame", and a "Raw" download link.

In which fields would you like to see Swift in the future?

(1:15:32)

Lattner: My goal for Swift has always been and still is total world domination. It's a modest goal. [Applause]

the plan

myths



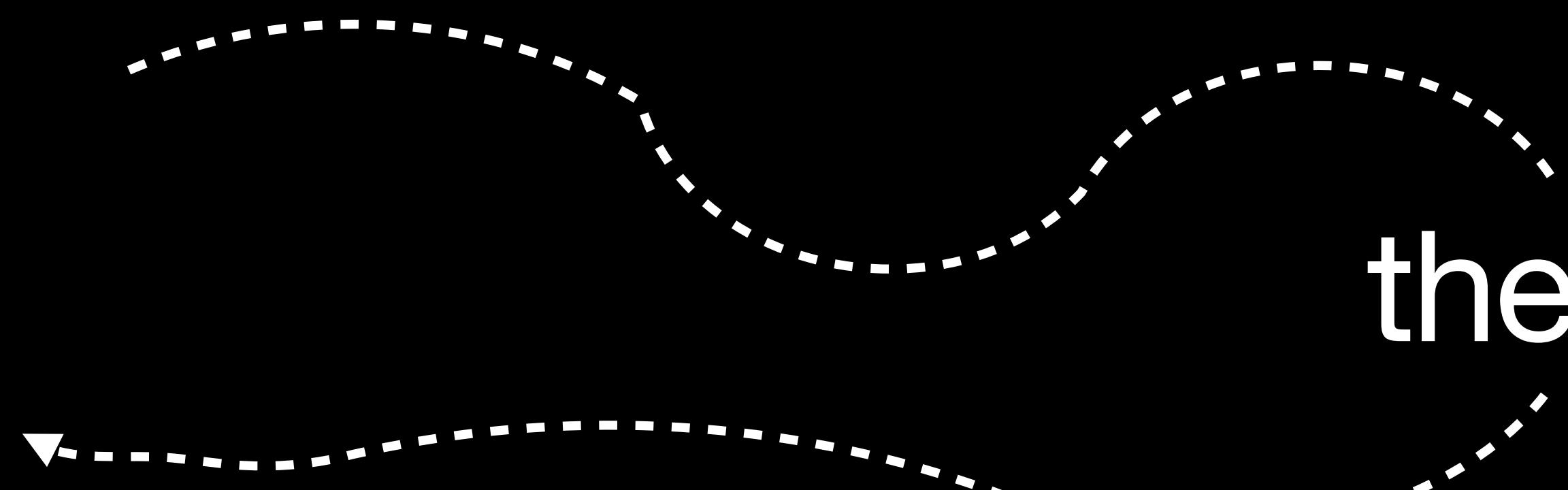
reality

the plan

myths

reality

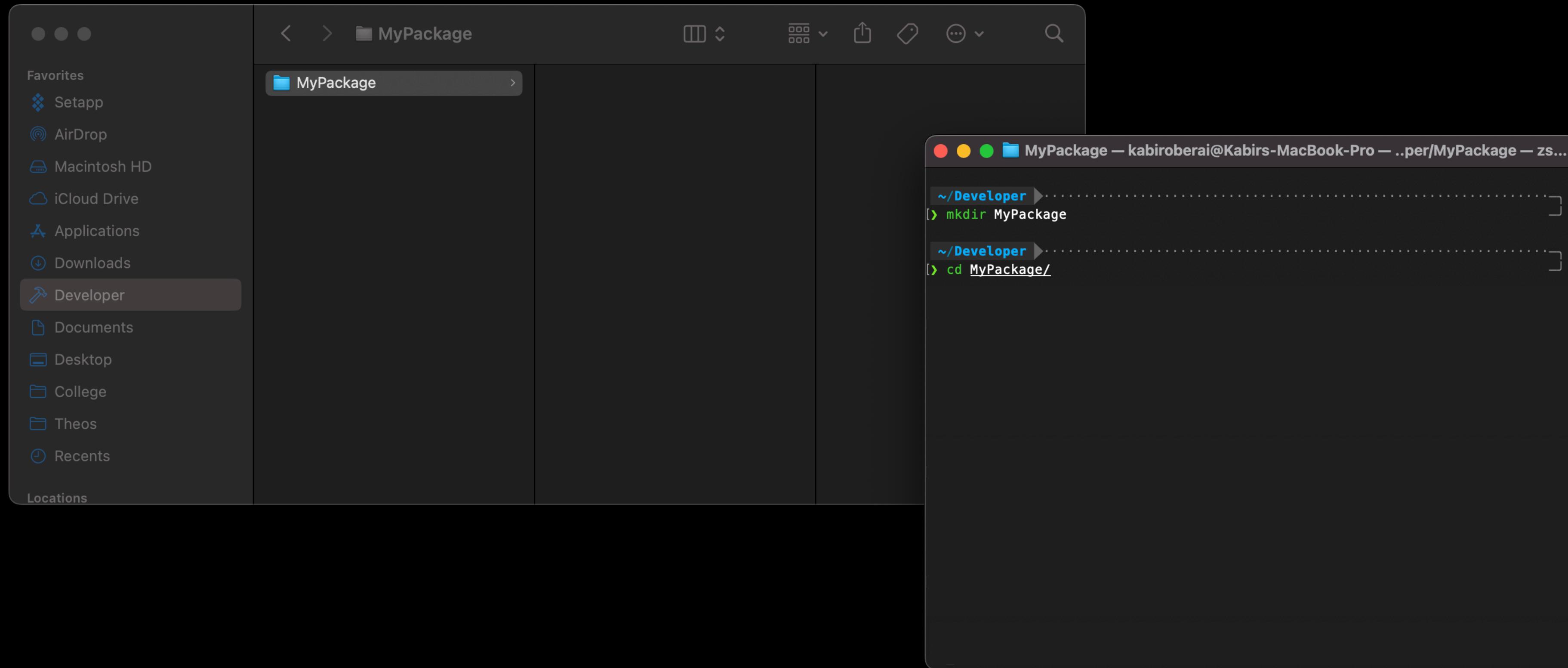
theory



myth 1: Xcode is the only decent way to build Swift code.

myth 1: Xcode is the only decent way to build Swift code.

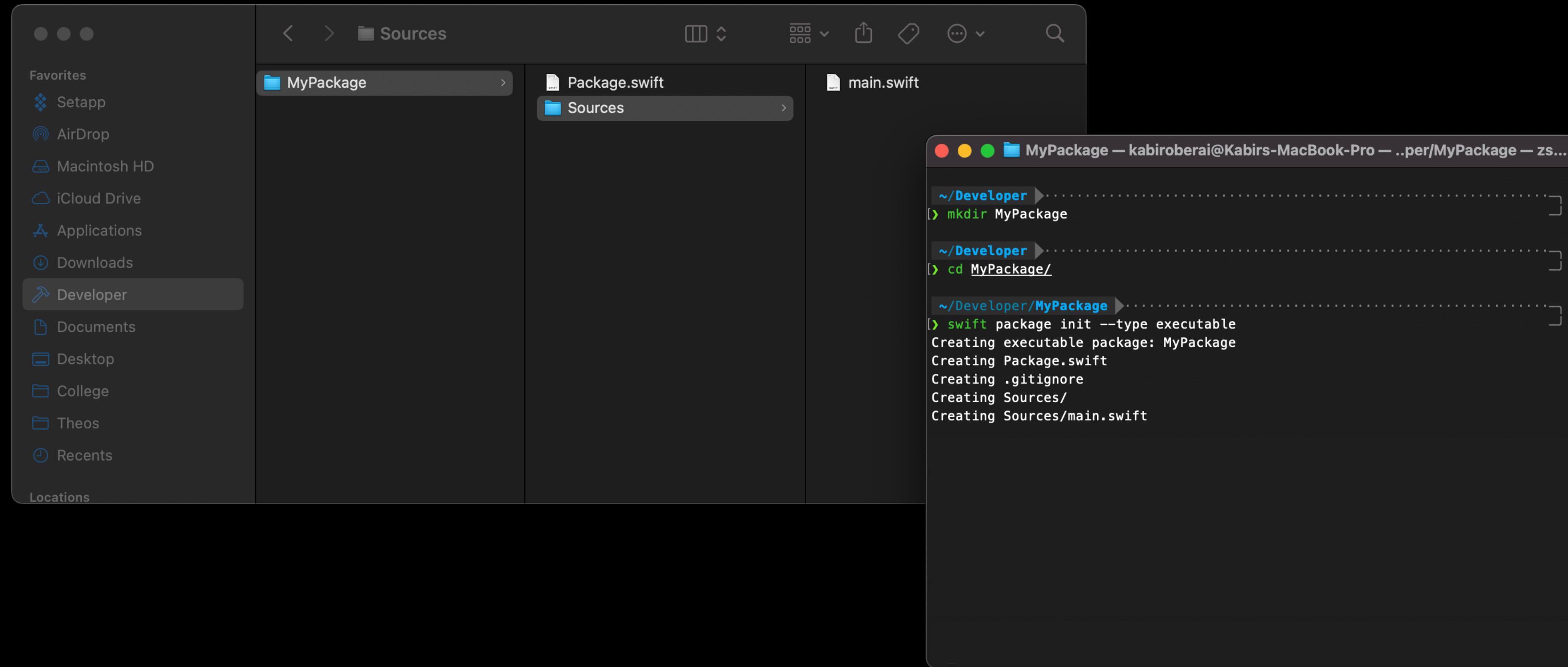
reality: SwiftPM is your friend on the command line 



<https://swift.org/getting-started/#using-the-package-manager>

myth 1: Xcode is the only decent way to build Swift code.

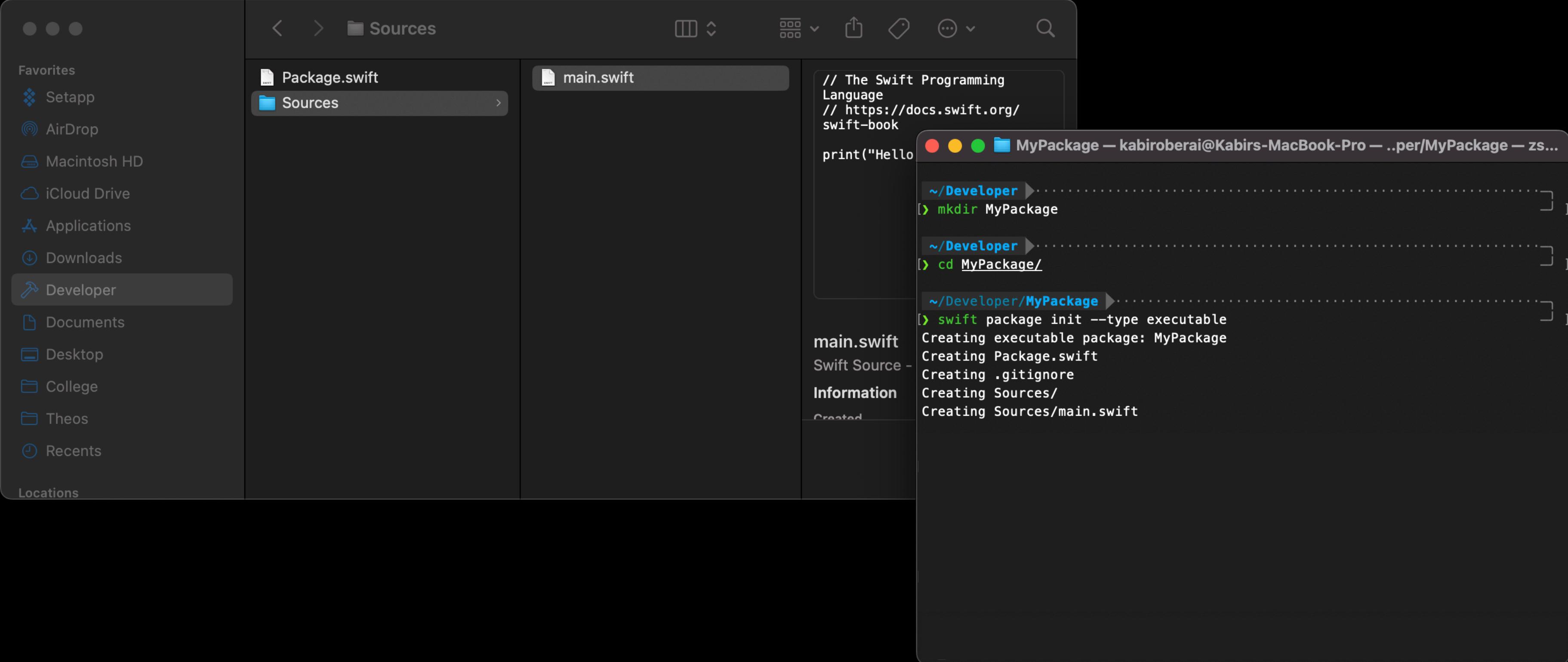
reality: SwiftPM is your friend on the command line 



<https://swift.org/getting-started/#using-the-package-manager>

myth 1: Xcode is the only decent way to build Swift code.

reality: SwiftPM is your friend on the command line 



The screenshot shows a macOS terminal window with a dark theme. The path ~/Developer/MyPackage is displayed at the top. The user has run three commands: 'mkdir MyPackage', 'cd MyPackage/', and 'swift package init --type executable'. The terminal output shows the creation of a package named 'MyPackage' with files like 'Package.swift', 'Sources', '.gitignore', 'Sources/main.swift', and 'Sources/Information'. The 'Sources' folder contains a 'main.swift' file with the following code:

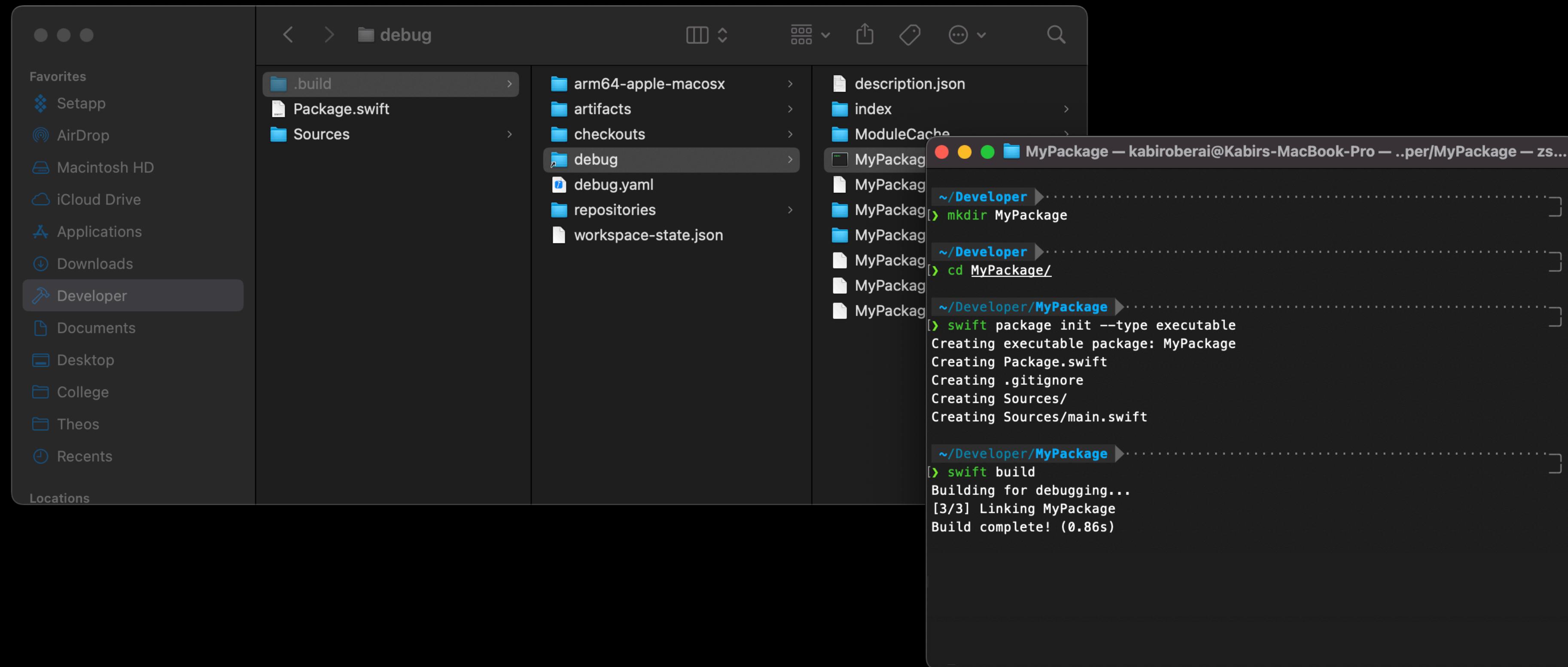
```
// The Swift Programming Language
// https://docs.swift.org/swift-book

print("Hello")
```

<https://swift.org/getting-started/#using-the-package-manager>

myth 1: Xcode is the only decent way to build Swift code.

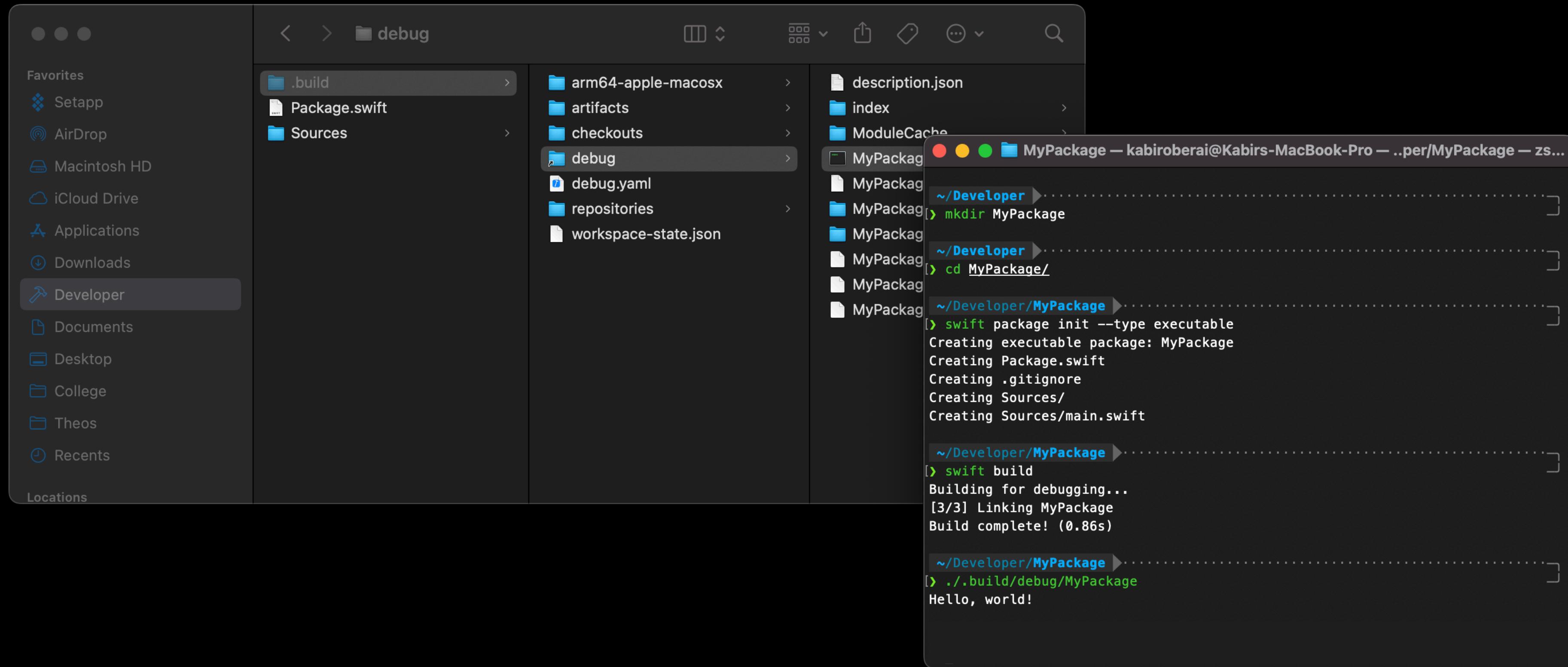
reality: SwiftPM is your friend on the command line 



<https://swift.org/getting-started/#using-the-package-manager>

myth 1: Xcode is the only decent way to build Swift code.

reality: SwiftPM is your friend on the command line 



The screenshot shows a macOS terminal window with a dark theme. The path in the title bar is `debug`. The left sidebar shows 'Favorites' including Setapp, AirDrop, Macintosh HD, iCloud Drive, Applications, Downloads, and Developer (which is selected). The main pane displays a file tree for a Swift package:

- `.build`
- `Package.swift`
- `Sources`
- `debug` (selected)
- `arm64-apple-macosx`
- `artifacts`
- `checkouts`
- `debug.yaml`
- `repositories`
- `workspace-state.json`

The right pane shows the terminal output:

```
MyPackage — kabiroberai@Kabirs-MacBook-Pro — ..per/MyPackage — zs...
~/Developer/MyPackage mkdir MyPackage
~/Developer/MyPackage cd MyPackage/
~/Developer/MyPackage swift package init --type executable
Creating executable package: MyPackage
Creating Package.swift
Creating .gitignore
Creating Sources/
Creating Sources/main.swift

~/Developer/MyPackage swift build
Building for debugging...
[3/3] Linking MyPackage
Build complete! (0.86s)

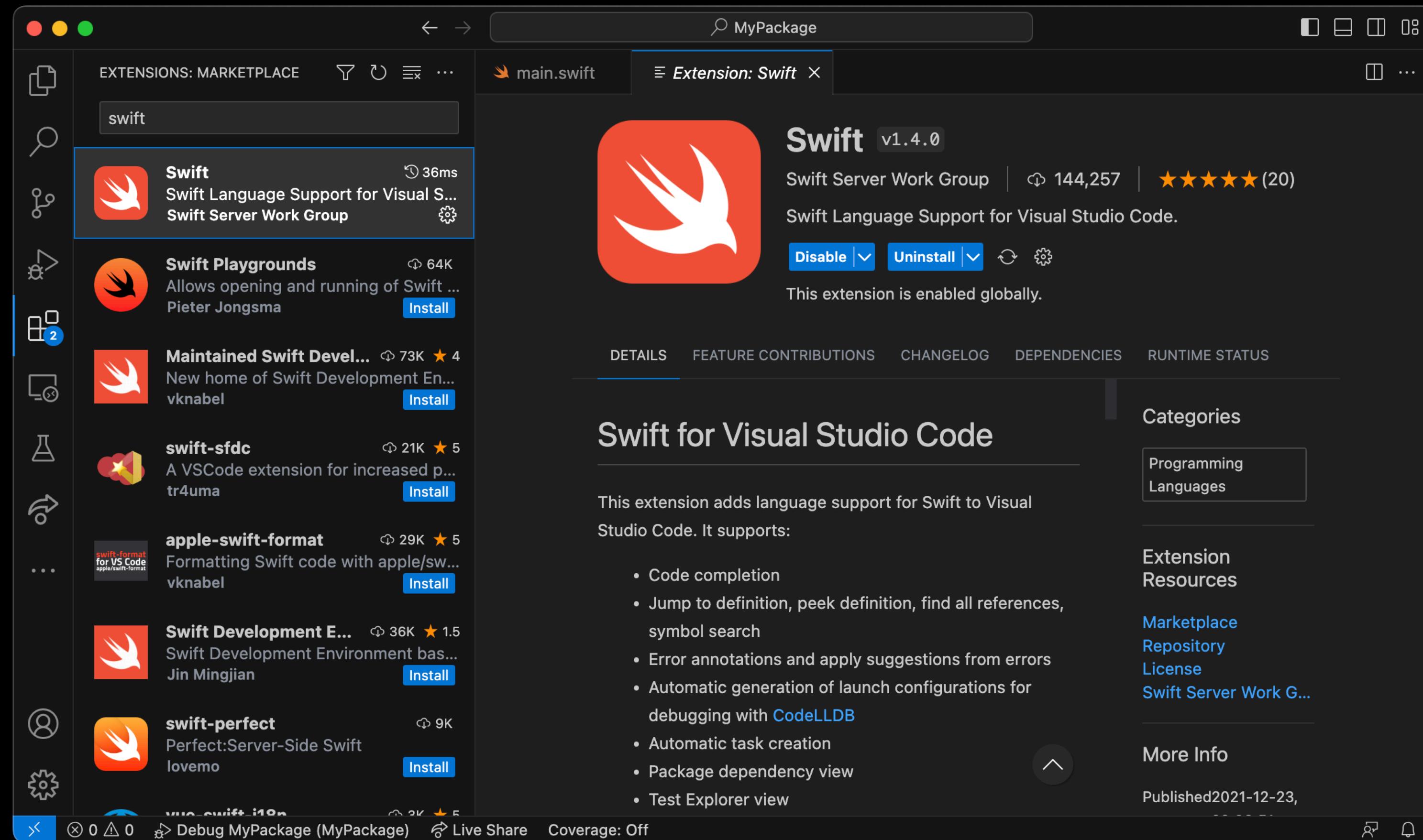
~/Developer/MyPackage ./..build/debug/MyPackage
Hello, world!
```

<https://swift.org/getting-started/#using-the-package-manager>

myth 2: Xcode is the only decent way to edit Swift code.

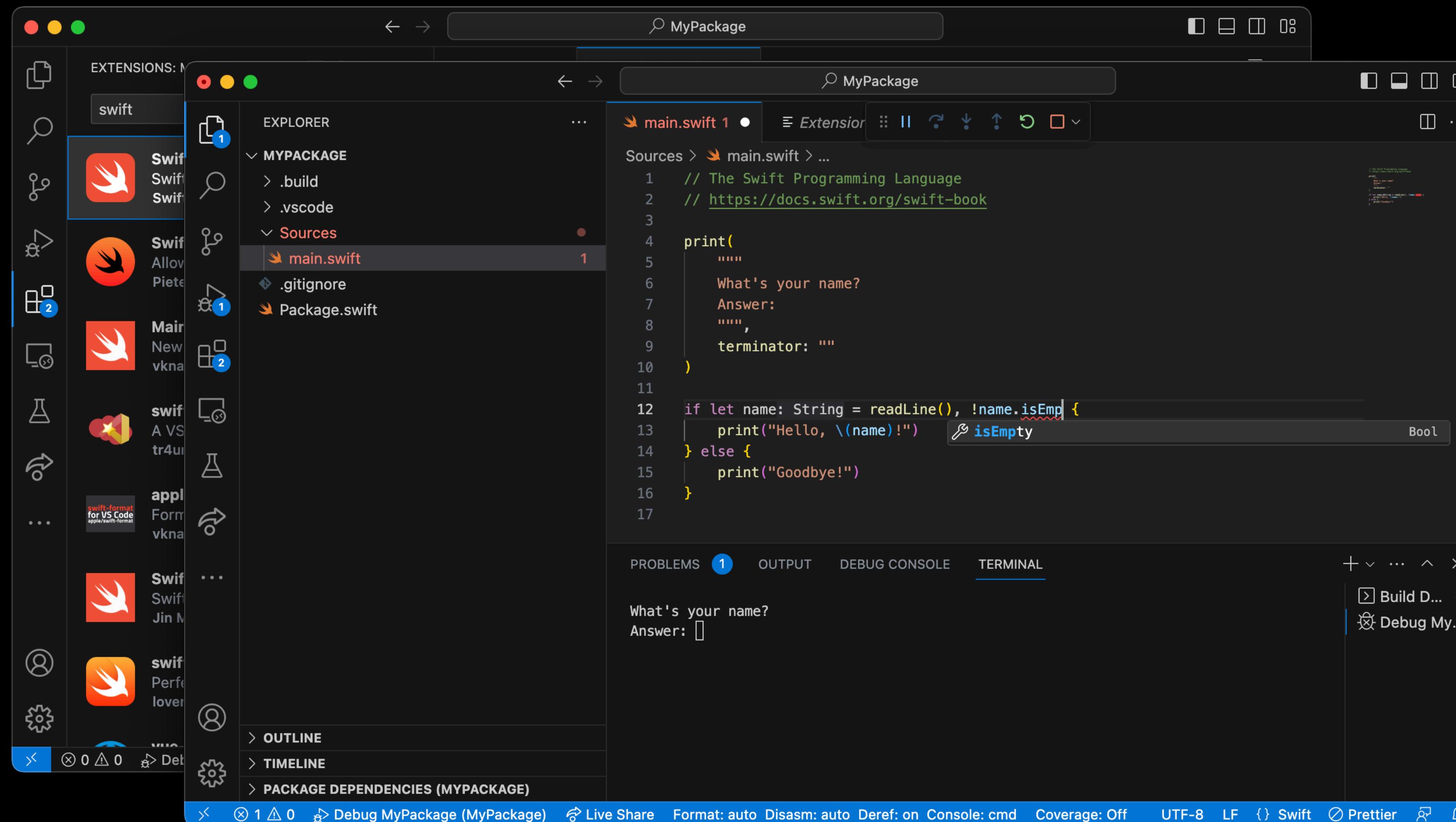
myth 2: Xcode is the only decent way to edit Swift code.

reality: SourceKit-LSP supports your favorite IDEs 🔎



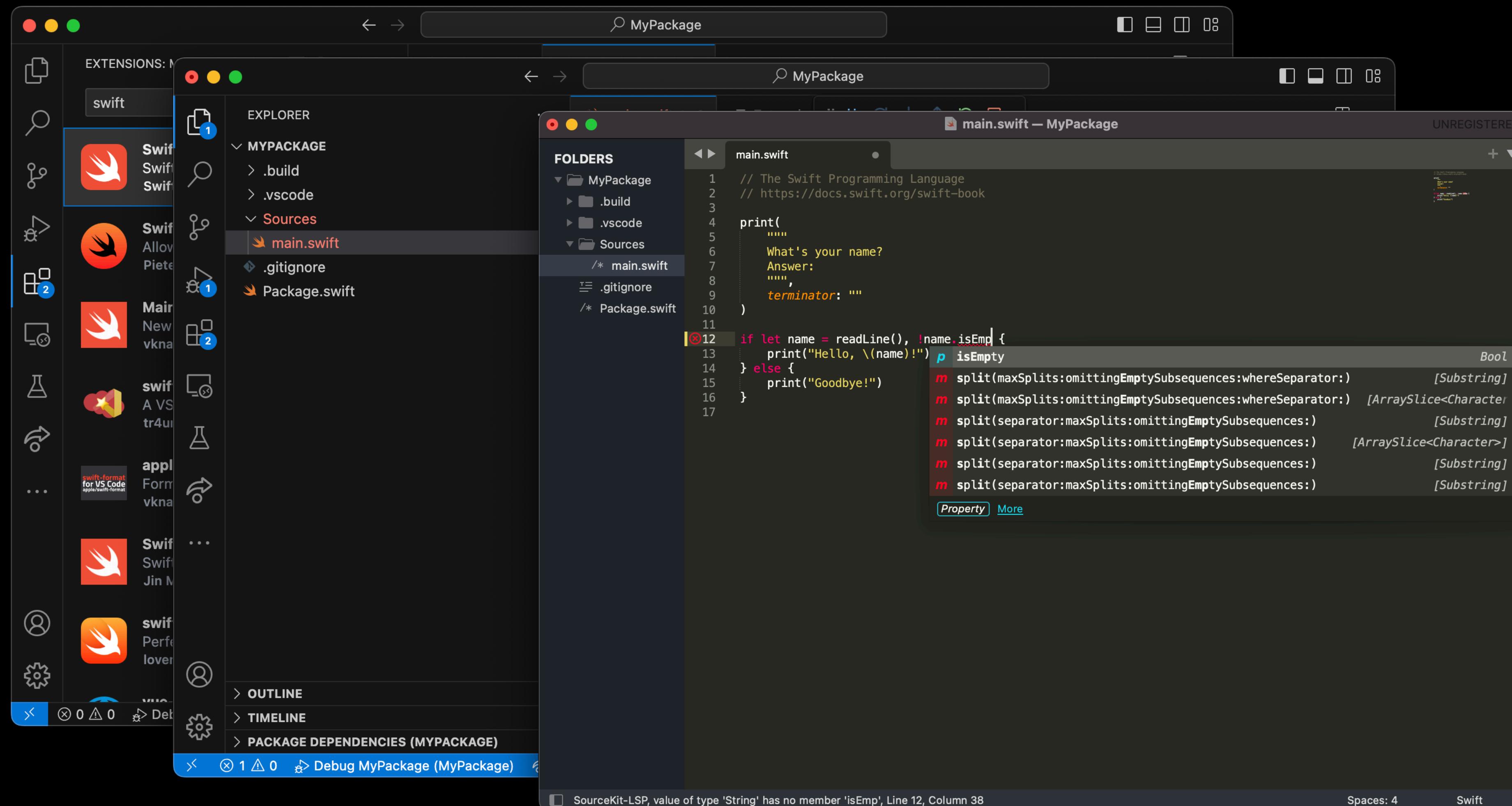
myth 2: Xcode is the only decent way to edit Swift code.

reality: SourceKit-LSP supports your favorite IDEs



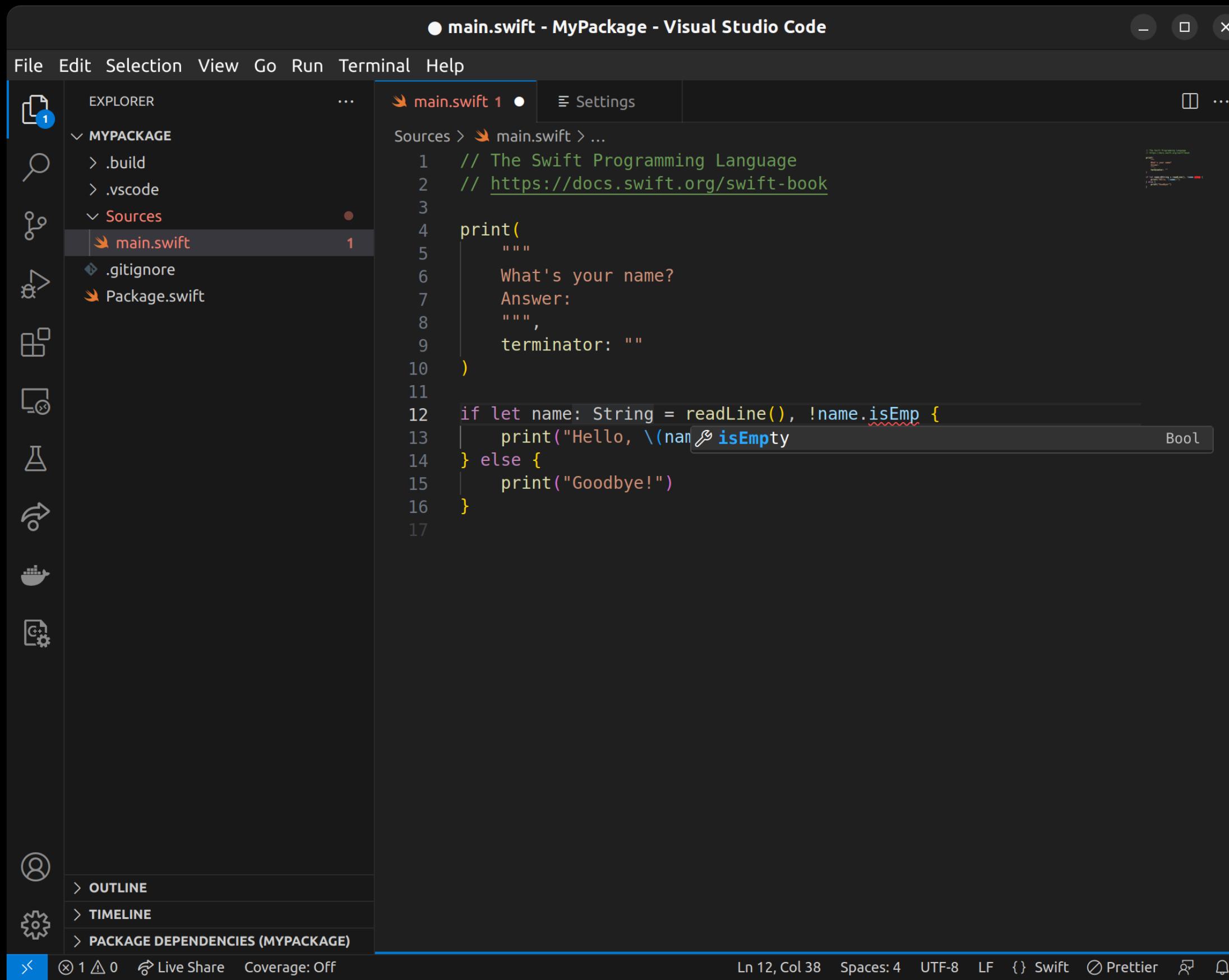
myth 2: Xcode is the only decent way to edit Swift code.

reality: SourceKit-LSP supports your favorite IDEs 🔎



myth 3: You can only write (edit+build) Swift code on macOS

myth 3: You can only write (edit+build) Swift code on macOS
reality: You can write Swift code on Linux,

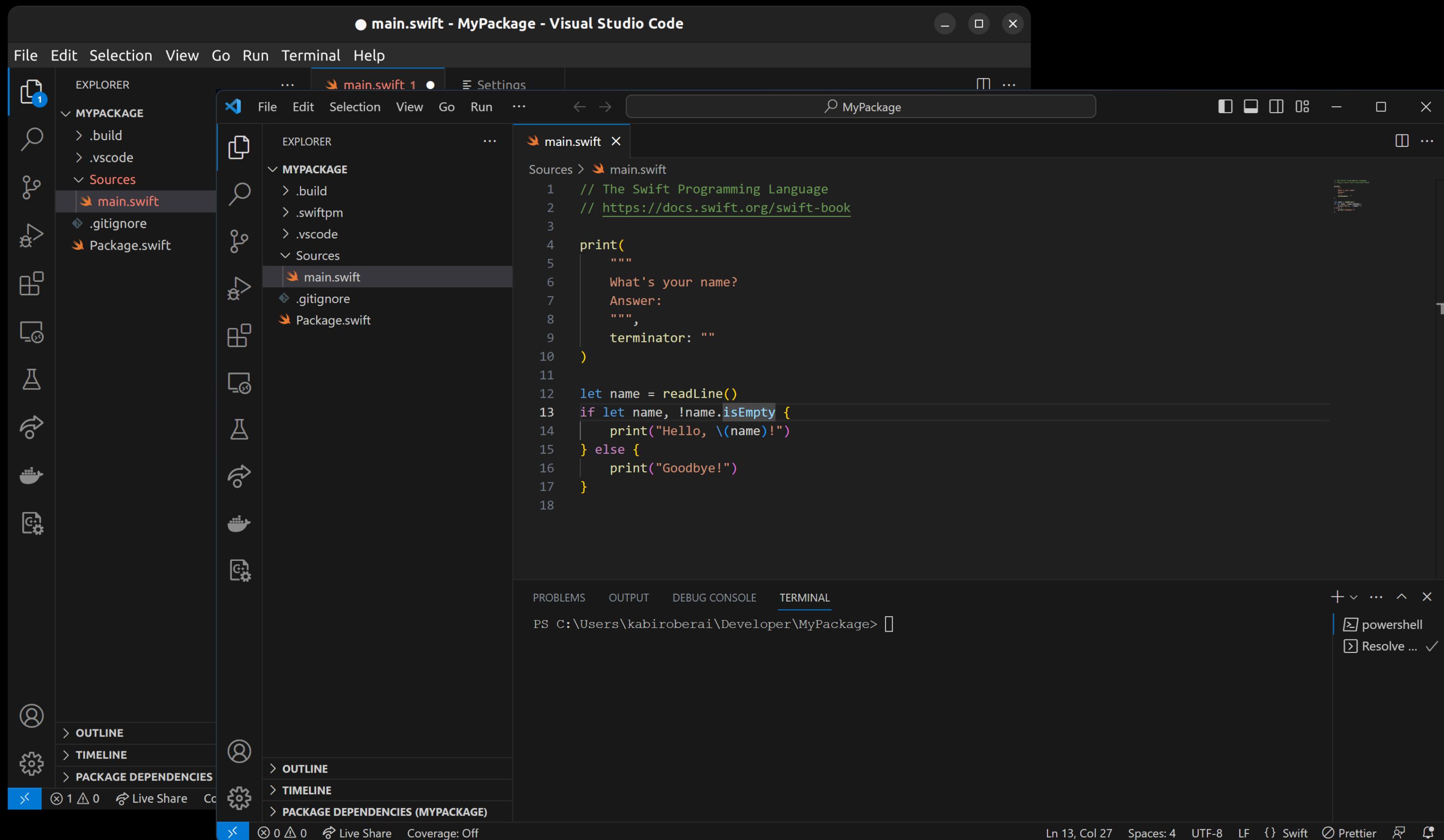


The screenshot shows the Visual Studio Code interface with a dark theme. The title bar reads "main.swift - MyPackage - Visual Studio Code". The left sidebar has icons for File Explorer, Search, Find, Open, Recent, and Settings. The "EXPLORER" tab is selected, showing a tree view of the project structure under "MYPACKAGE": ".build", ".vscode", "Sources", and "main.swift". The "Sources" node has a red dot indicating changes. The "main.swift" file is selected and shown in the main editor area. The code is as follows:

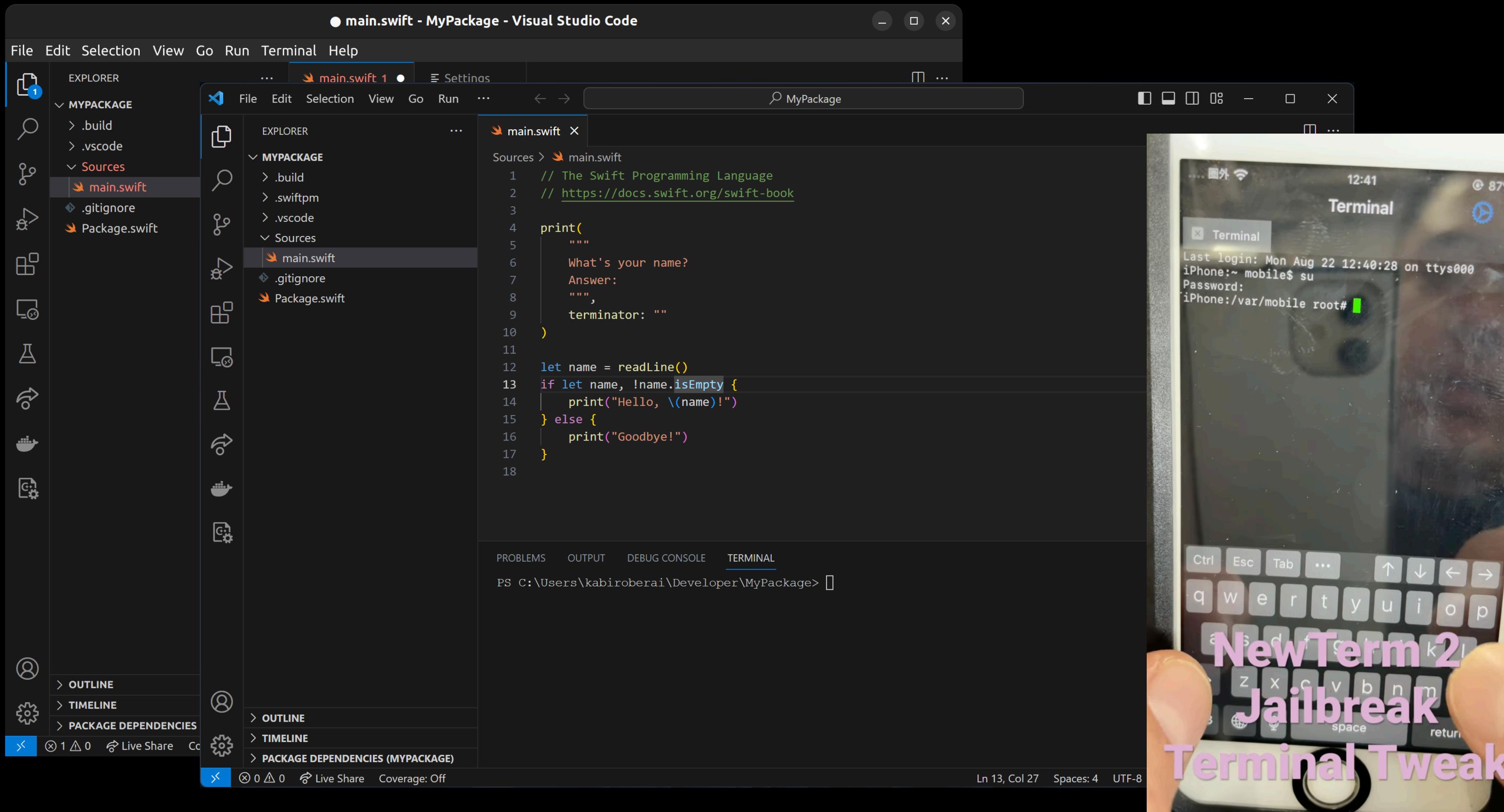
```
1 // The Swift Programming Language
2 // https://docs.swift.org/swift-book
3
4 print(
5     """
6     What's your name?
7     Answer:
8     """,
9     terminator: ""
10)
11
12 if let name: String = readLine(), !name.isEmpty {
13     print("Hello, \(name)!")
14 } else {
15     print("Goodbye!")
16 }
17
```

The cursor is at line 12, column 38, where the word "isEmpty" is highlighted in red, indicating a spelling error or a reference to an undeclared variable. The status bar at the bottom shows "Ln 12, Col 38" and other settings like "Spaces: 4", "UTF-8", "LF", "Swift", and "Prettier".

myth 3: You can only write (edit+build) Swift code on macOS
reality: You can write Swift code on Linux, Windows,



myth 3: You can only write (edit+build) Swift code on macOS
reality: You can write Swift code on Linux, Windows,
and your (jail)broken iPhone 5 from 2012 

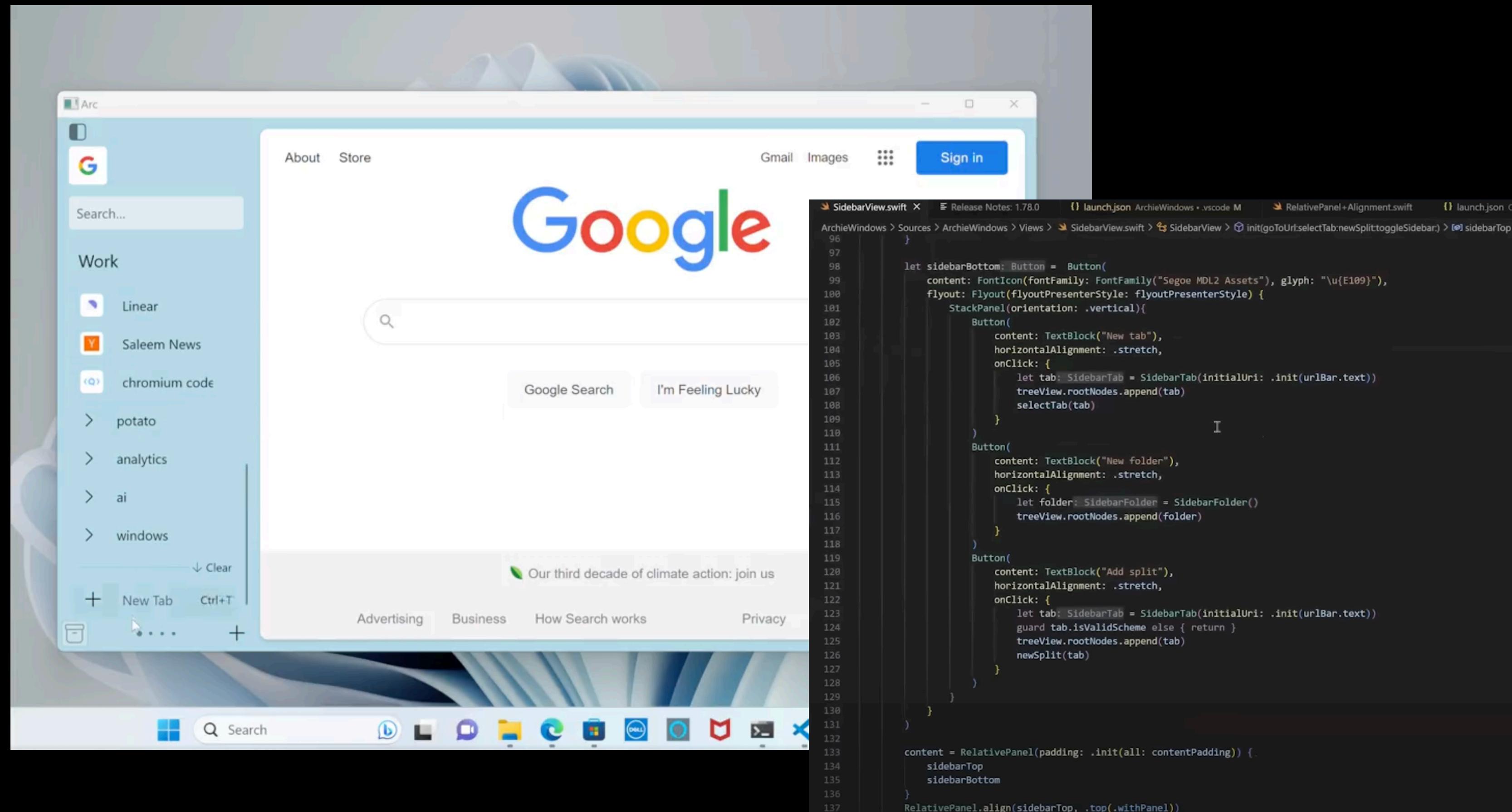


iPhone image: [applemacjp](#)

myth 4: *Swift on Linux and Windows is only for servers.*

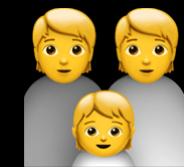
myth 4: Swift on Linux and Windows is only for servers.

reality: Consumer apps are coming



myth 4: Swift on Linux and Windows is only for servers.

reality: Consumer apps are coming



The screenshot shows a Visual Studio Code interface with a dark theme. On the left is the Explorer sidebar showing a project structure for 'MYLINUXAPP' with files like '.build', '.vscode', 'Sources/MyLinuxApp/MyLinuxApp.swift', 'Tests', '.gitignore', 'Package.resolved', and 'Package.swift'. The main editor area displays Swift code for a 'GreetingGeneratorApp' using SwiftCrossUI. The code includes imports for SwiftCrossUI, defines a state class 'GreetingGeneratorState' with properties for name and greetings, and a struct 'GreetingGeneratorApp' with a main function. The terminal below shows the build process:

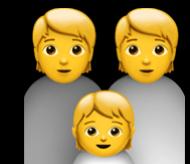
```
kabiroberai@ubuntu-linux-22-04-desktop:~/Developer/MyLinuxApp$ swift build --static-swift-stdlib -c release
Building for production...
Build complete! (0.07s)
kabiroberai@ubuntu-linux-22-04-desktop:~/Developer/MyLinuxApp$ ./.build/release/MyLinuxApp
libEGL warning: pci id for fd 9: lab8:0010, driver (null)

pci id for fd 7: lab8:0010, driver (null)
libGL error: pci id for fd 7: lab8:0010, driver (null)
pci id for fd 9: lab8:0010, driver (null)
^C
kabiroberai@ubuntu-linux-22-04-desktop:~/Developer/MyLinuxApp$ du -hs .build/release/MyLinuxApp
52M .build/release/MyLinuxApp
kabiroberai@ubuntu-linux-22-04-desktop:~/Developer/MyLinuxApp$
```

The status bar at the bottom indicates the file is a Swift file (Swift) and shows line 49, column 1. To the right of the code editor is a window titled 'Greeting Generator' showing a simple UI with a text input field containing 'Kabir' and a button labeled 'Generate'. The output of the application shows the greeting 'Hello, Kabir!' and a history entry 'Hello, Tim Cook!'. The footer of the slide states 'Gtk with stackotter/swift-cross-ui (experimental)'.

myth 4: Swift on Linux and Windows is only for servers.

reality: Consumer apps are coming



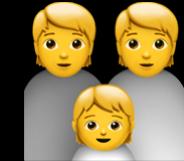
The screenshot shows a Visual Studio Code interface with the following details:

- Title Bar:** MyLinuxApp.swift - MyLinuxApp - Visual Studio Code
- File Explorer:** Shows the project structure for "MYLINUXAPP" including ".build", ".vscode", "Sources/MyLinuxApp" (containing "MyLinuxApp.swift"), "Tests", ".gitignore", "Package.resolved", and "Package.swift".
- Editor:** Displays Swift code for a "GreetingGeneratorApp". The code imports SwiftCrossUI and defines a struct "GreetingGeneratorApp" with an @main attribute. It initializes a "GreetingGeneratorState" observable object and creates a "WindowProperties" object with a title of "Greeting Generator". The "body" of the window is defined as a VStack containing a TextField for the name and a HStack with a Button labeled "Generate". The button's action adds a greeting to the state's greetings array.
- Terminal:** Shows command-line output:
 - Building for production...
 - Build complete! (0.07s)
 - Running the application: ./build/release/MyLinuxApp
 - Output: libEGL warning: pci id for fd 9: lab8:0010, driver (null)
 - Output: pci id for fd 7: lab8:0010, driver (null)
 - Output: libGL error: pci id for fd 7: lab8:0010, driver (null)
 - Output: pci id for fd 9: lab8:0010, driver (null)
 - Output: ^C
- Status Bar:** Shows file path (a970a4dc), line count (0), column count (0), coverage (Off), and language (Swift).
- Bottom Text:** Gtk with [stackotter/swift-cross-ui](#) (experimental)
- Right Window:** A "Greeting Generator" window with a title bar. Inside, there is a text input field with "Kabir", a "Generate" button, and a "Reset" button. Below the input field, the text "Hello, Kabir!" is displayed. A "History:" section shows previous greetings: "Hello, Tim Cook!"

easy distribution

myth 4: Swift on Linux and Windows is only for servers.

reality: Consumer apps are coming



The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** Shows a project structure for "MYLINUXAPP" containing files like ".build", ".vscode", "Sources/MyLinuxApp/MyLinuxApp.swift", "Tests", ".gitignore", "Package.resolved", and "Package.swift".
- Editor:** Displays the file "MyLinuxApp.swift" with the following code:

```
import SwiftCrossUI

class GreetingGeneratorState: Observable {
    @Observed var name: String = ""
    @Observed var greetings: [String] = []
}

@main
struct GreetingGeneratorApp: App {
    let identifier: String = "dev.stackotter.GreetingGenerator"
    let state: GreetingGeneratorState = GreetingGeneratorState()
    let windowProperties: WindowProperties = WindowProperties(title: "Greeting Generator")

    var body: some ViewContent {
        VStack {
            TextField("Name", state.$name)
            HStack {
                Button("Generate") {
                    state.greetings.append("Hello, \(state.name)!")
                }
            }
        }
    }
}
```
- Terminal:** Shows command-line output:

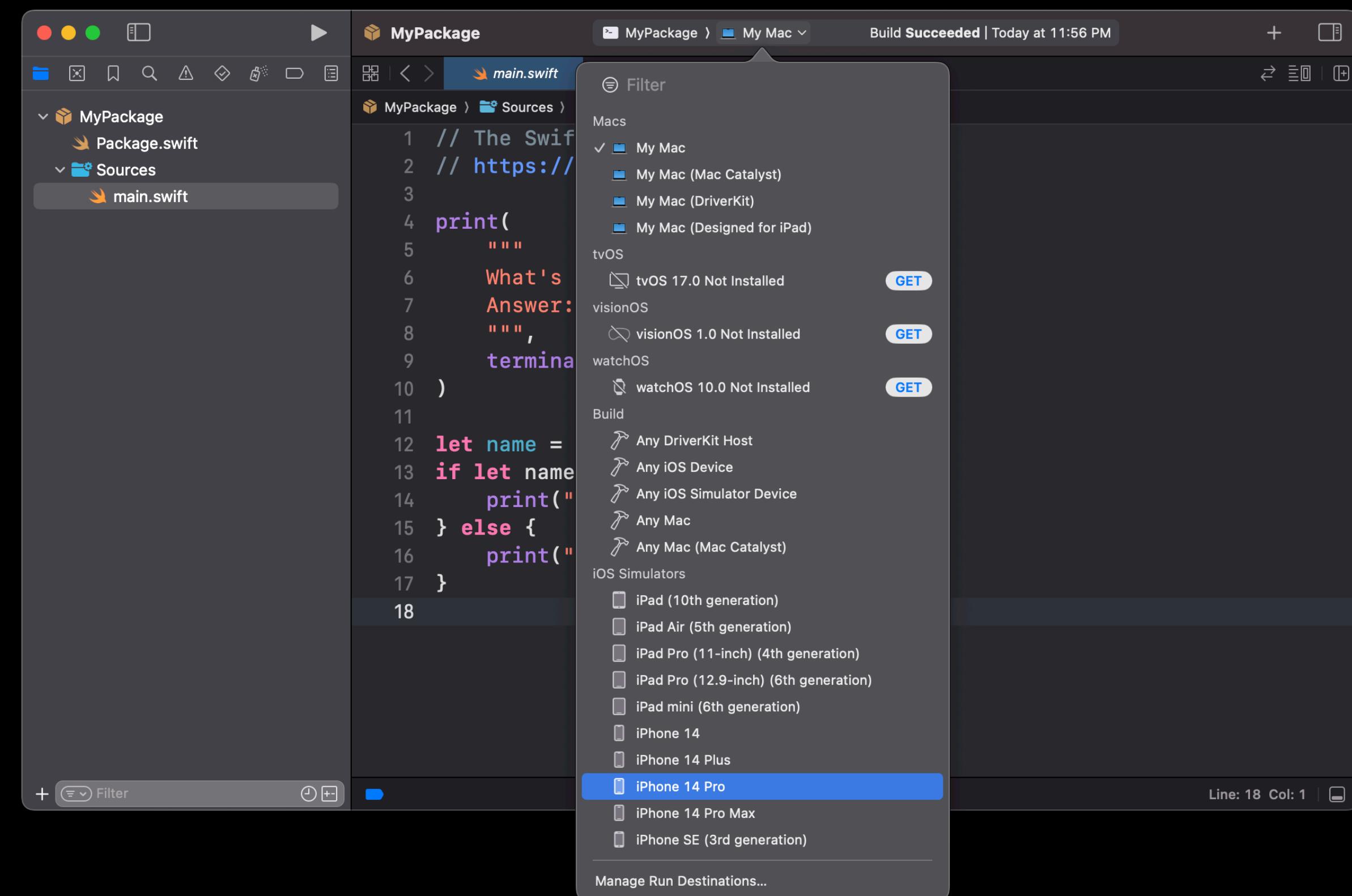
```
kabiroberai@ubuntu-linux-22-04-desktop:~/Developer/MyLinuxApp$ swift build --static-swift-stdlib -c release
Building for production...
Build complete! (0.07s)
kabiroberai@ubuntu-linux-22-04-desktop:~/Developer/MyLinuxApp$ ./.build/release/MyLinuxApp
libEGL warning: pci id for fd 9: lab8:0010, driver (null)

pci id for fd 7: lab8:0010, driver (null)
libGL error: pci id for fd 7: lab8:0010, driver (null)
pci id for fd 9: lab8:0010, driver (null)
^C
kabiroberai@ubuntu-linux-22-04-desktop:~/Developer/MyLinuxApp$ du -hs .build/release/MyLinuxApp
52M .build/release/MyLinuxApp
kabiroberai@ubuntu-linux-22-04-desktop:~/Developer/MyLinuxApp$
```
- Annotations:**
 - A white arrow points from the text "...but large files" in the bottom left of the VS Code interface to the terminal output showing a large file size (52M).
 - A white arrow points from the text "easy distribution" in the center of the VS Code interface to the "Greeting Generator" application window.
- Application Window:** A "Greeting Generator" window is shown with a text input field containing "Kabir", a "Generate" button, and a "Reset" button. The window displays the message "Hello, Kabir!" and a "History:" section with the entry "Hello, Kabir!".
- Page Footer:** The footer of the slide states "Gtk with [stackotter/swift-cross-ui](#) (experimental)".

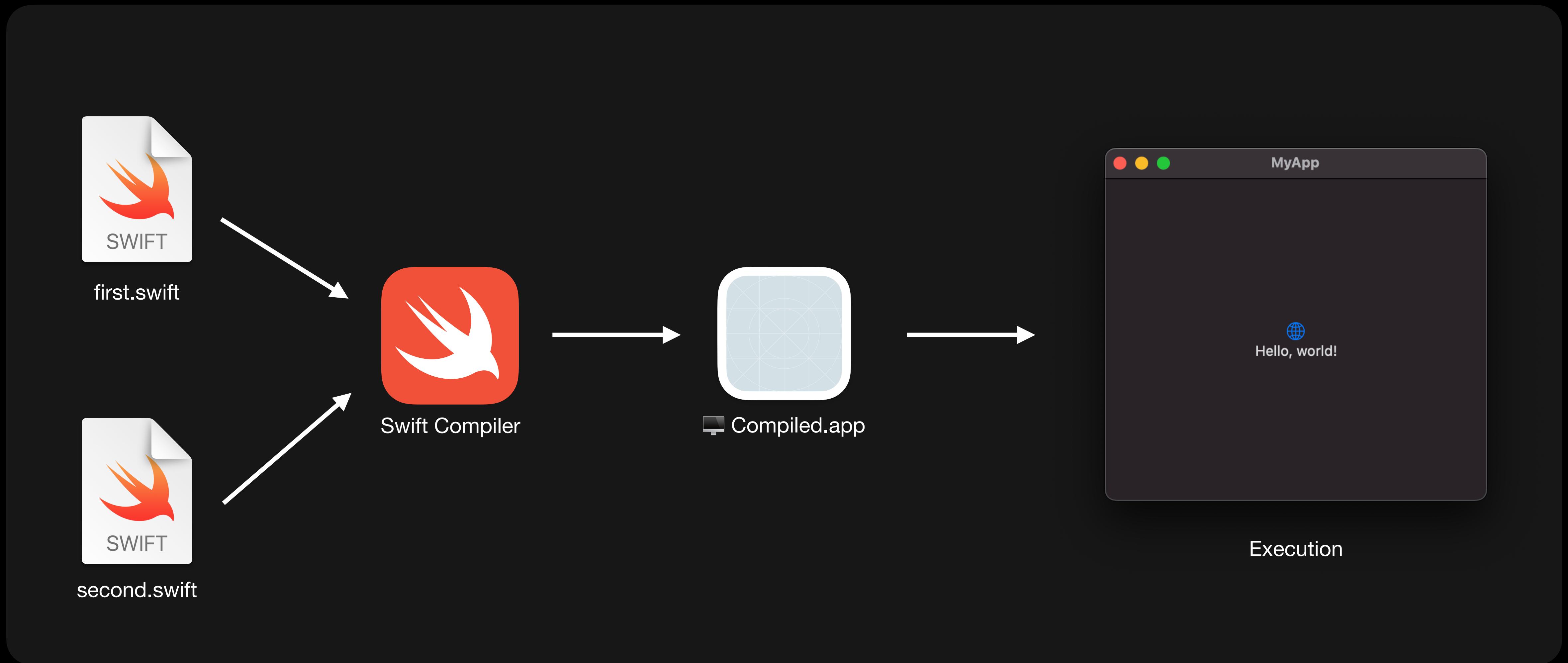
myth 5: You can only build Swift code for your own platform.

myth 5: You can only build Swift code for your own platform.

reality: Swift is a cross-compiler 🎭

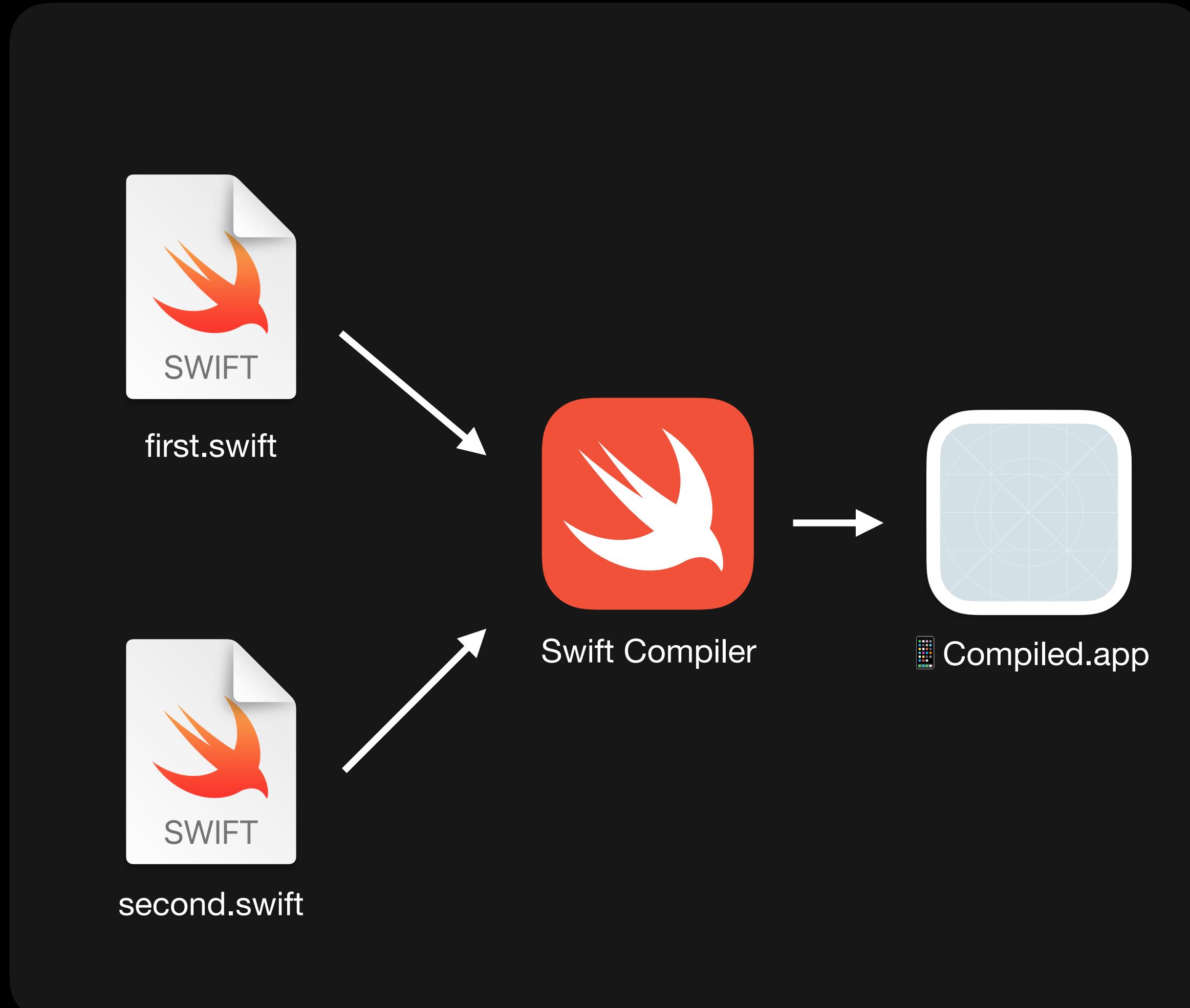


interlude: theory time!



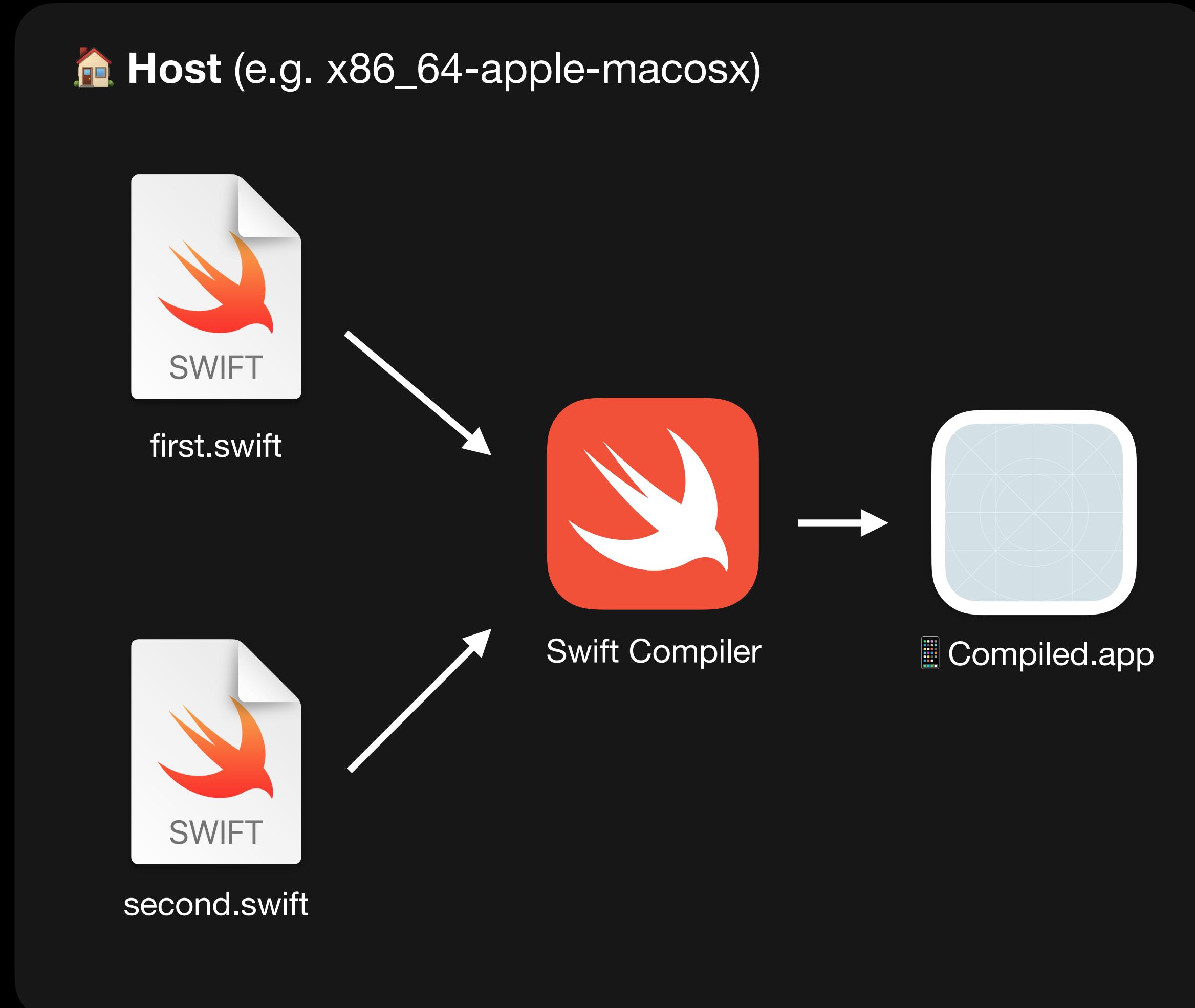
typical build process

interlude: theory time!

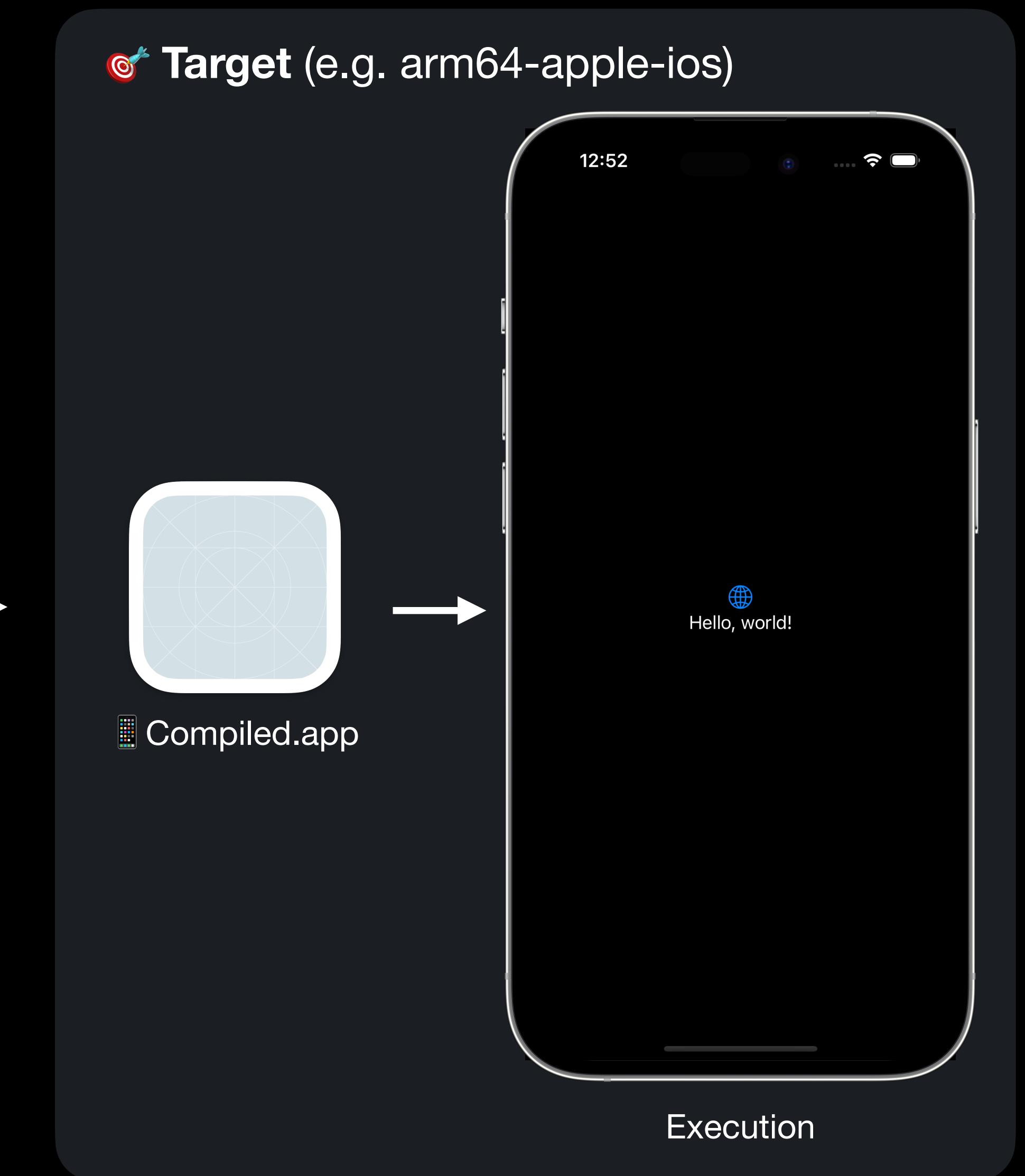


cross-compilation

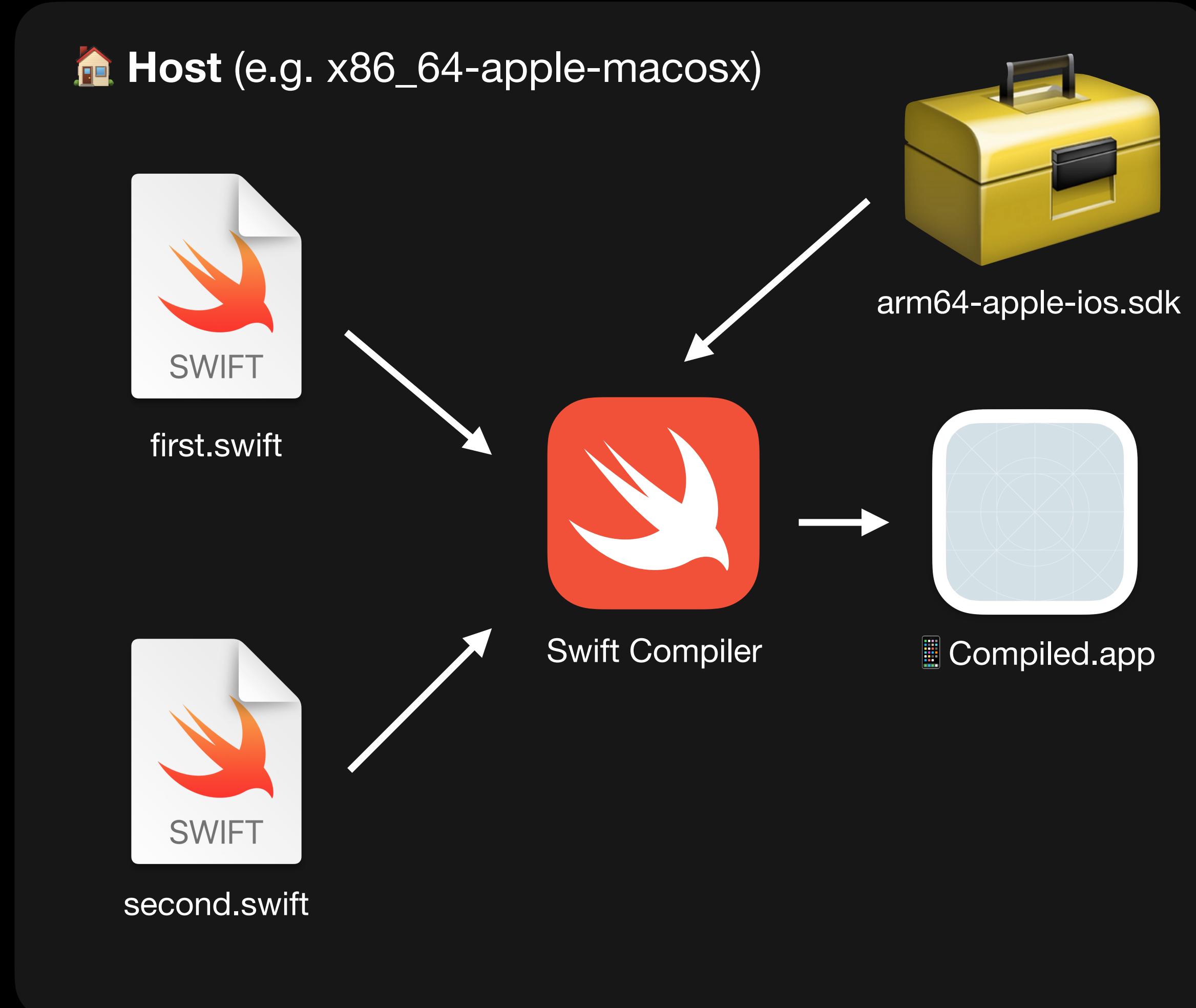
interlude: theory time!



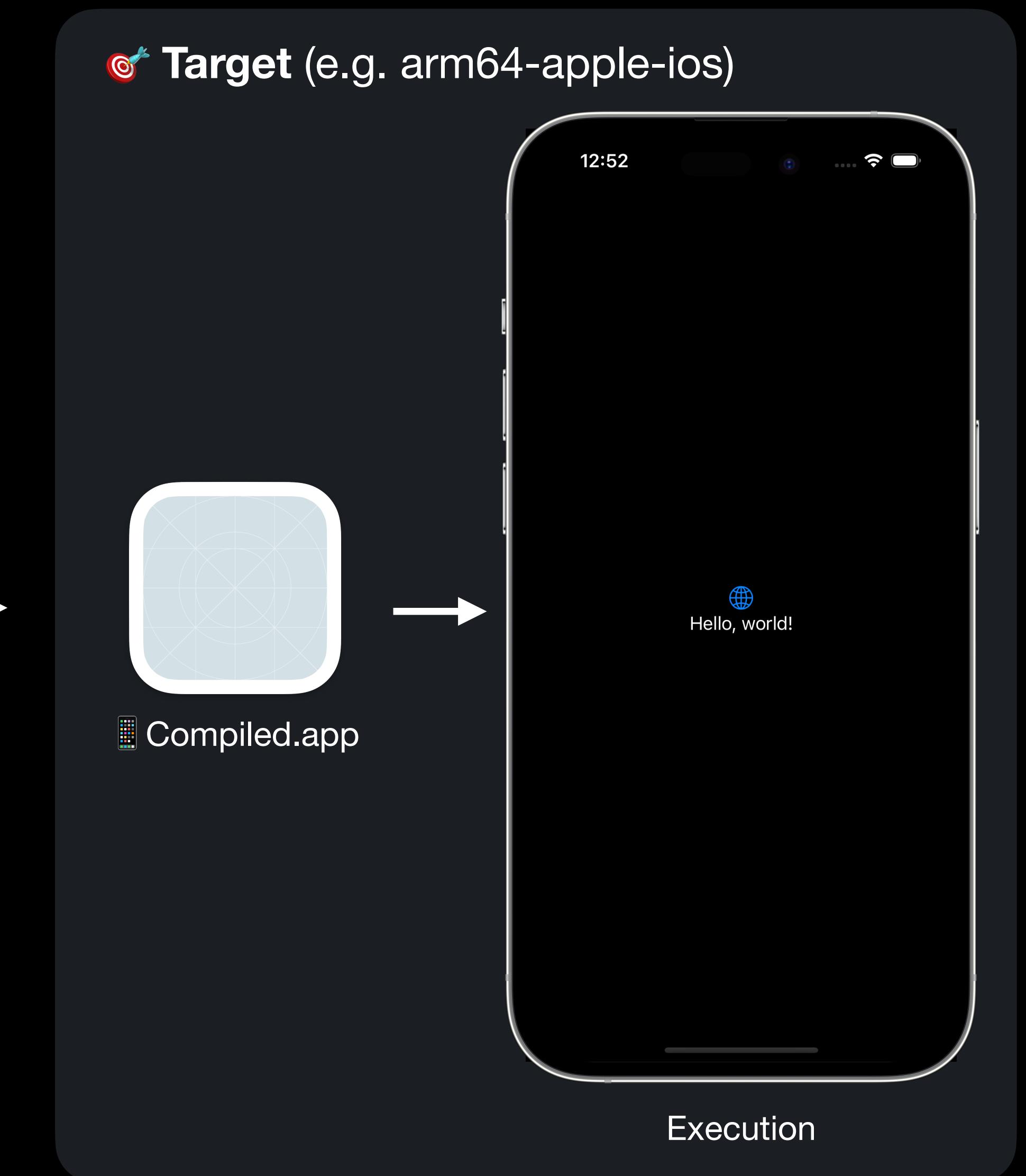
cross-compilation



interlude: theory time!



cross-compilation



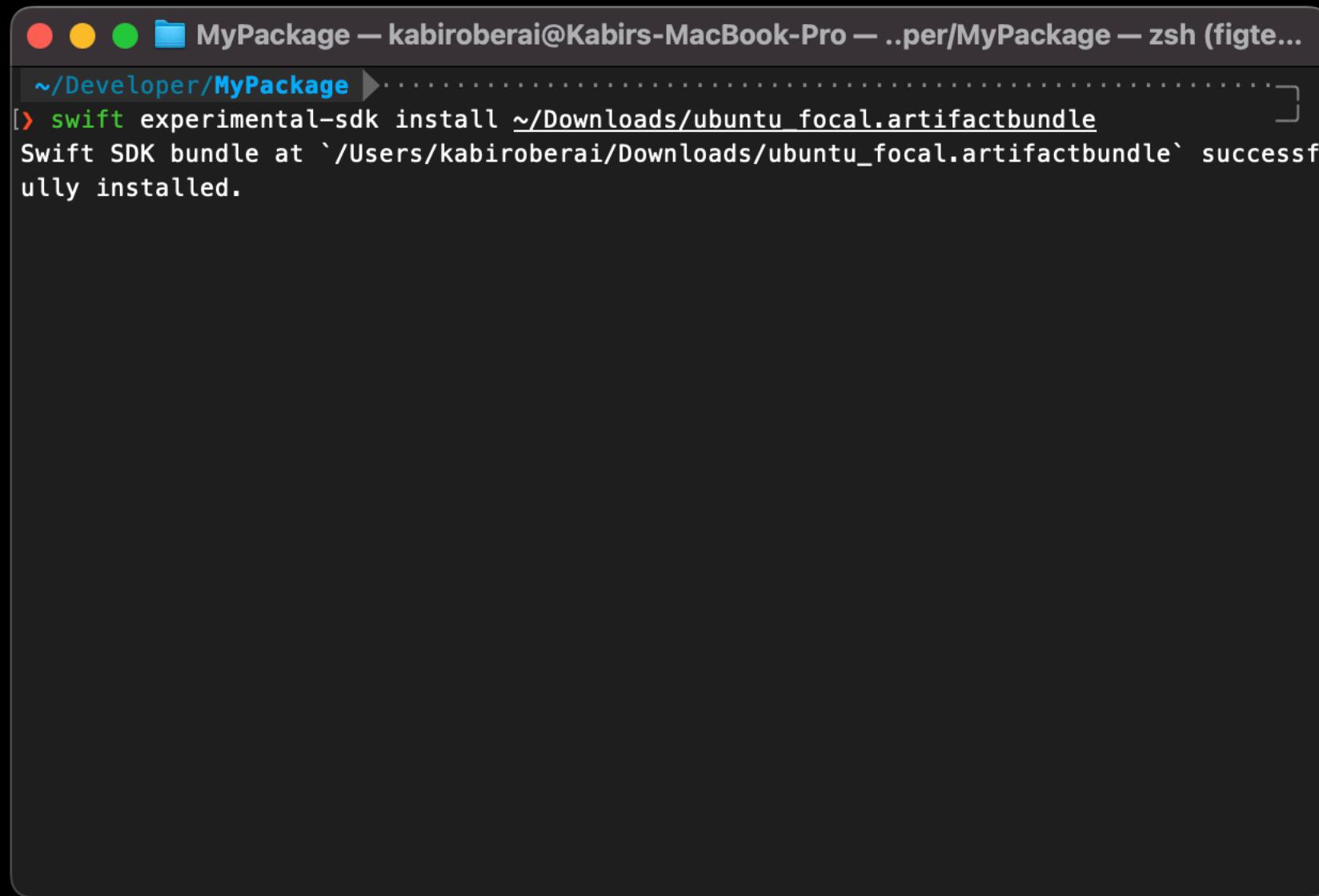
Execution

myth 5: You can only build Swift code for your own platform.

reality: Swift is a cross-compiler 🎭

myth 5: You can only build Swift code for your own platform.

reality: Swift is a cross-compiler 🎭 and SwiftPM is too!

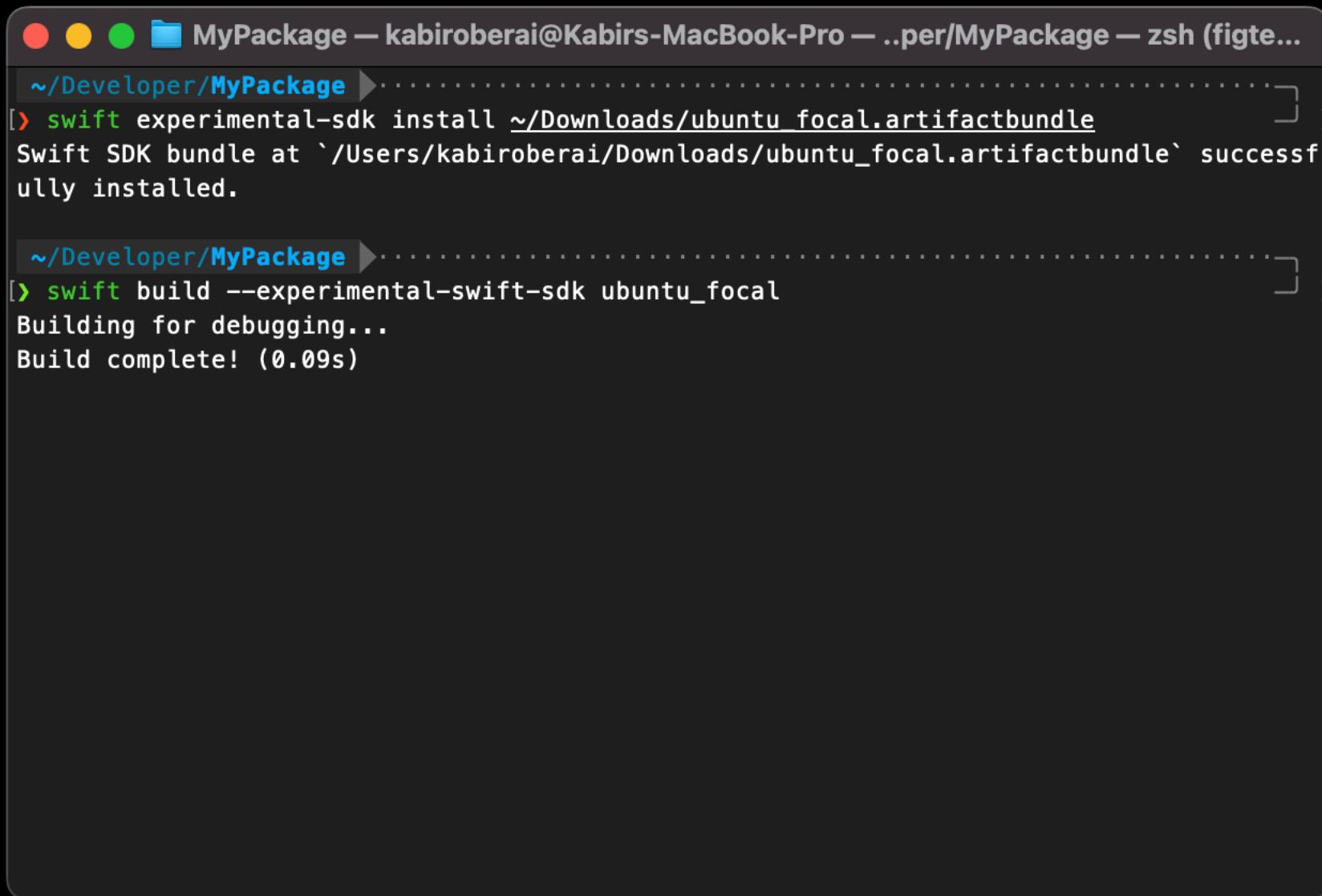


A screenshot of a macOS terminal window titled "MyPackage". The window shows a command being run: "swift experimental-sdk install ~/Downloads/ubuntu_focal.artifactbundle". The output indicates that the Swift SDK bundle was successfully installed at the specified path.

```
~/Developer/MyPackage ➤ swift experimental-sdk install ~/Downloads/ubuntu_focal.artifactbundle
Swift SDK bundle at `/Users/kabiroberai/Downloads/ubuntu_focal.artifactbundle` successfully installed.
```

myth 5: You can only build Swift code for your own platform.

reality: Swift is a cross-compiler 🎭 and SwiftPM is too!



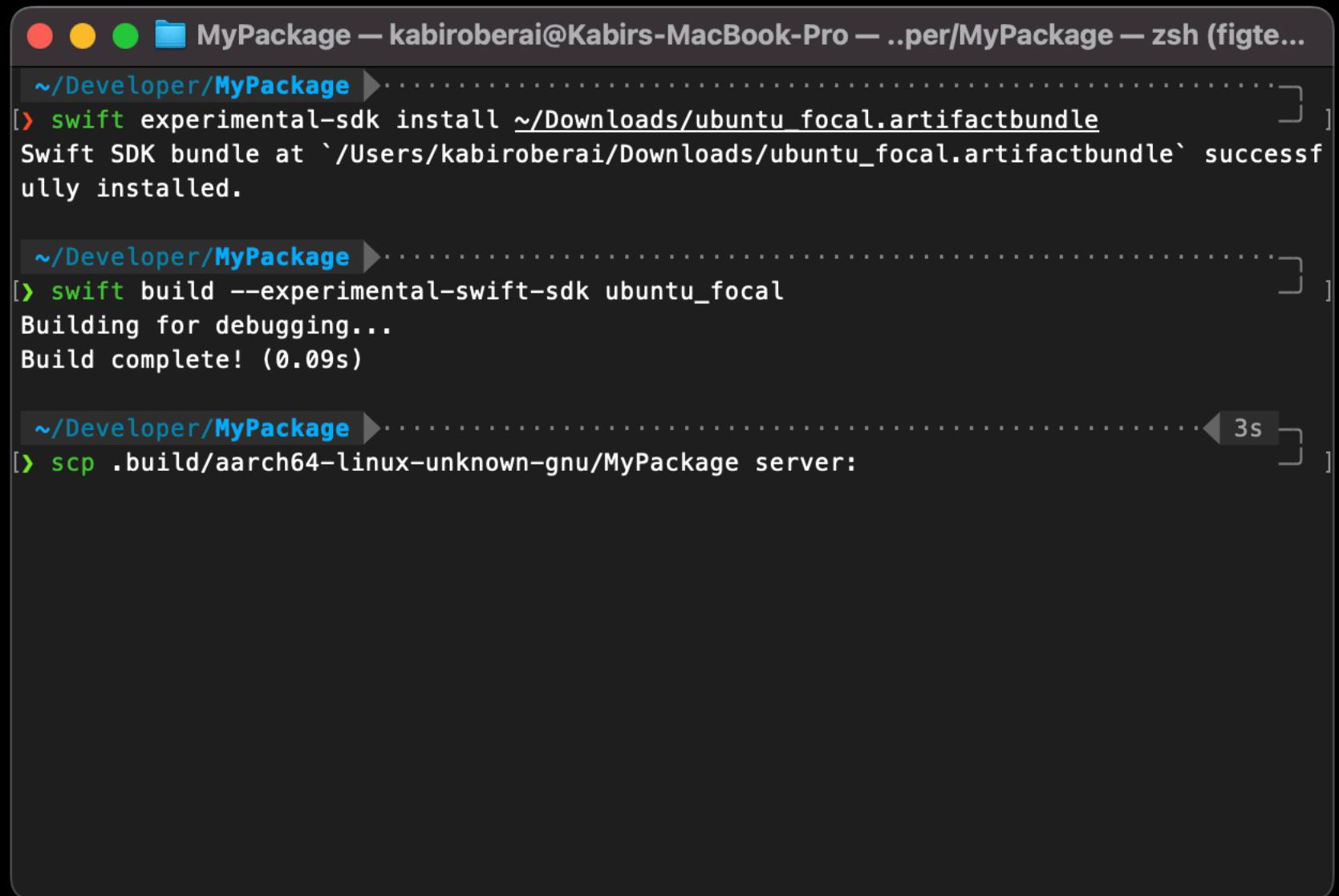
The screenshot shows a dark-themed macOS terminal window titled "MyPackage". The window contains two command-line sessions:

```
~/Developer/MyPackage ➤ swift experimental-sdk install ~/Downloads/ubuntu_focal.artifactbundle
Swift SDK bundle at `/Users/kabiroberai/Downloads/ubuntu_focal.artifactbundle` successfully installed.

~/Developer/MyPackage ➤ swift build --experimental-swift-sdk ubuntu_focal
Building for debugging...
Build complete! (0.09s)
```

myth 5: You can only build Swift code for your own platform.

reality: Swift is a cross-compiler 🎭 and SwiftPM is too!

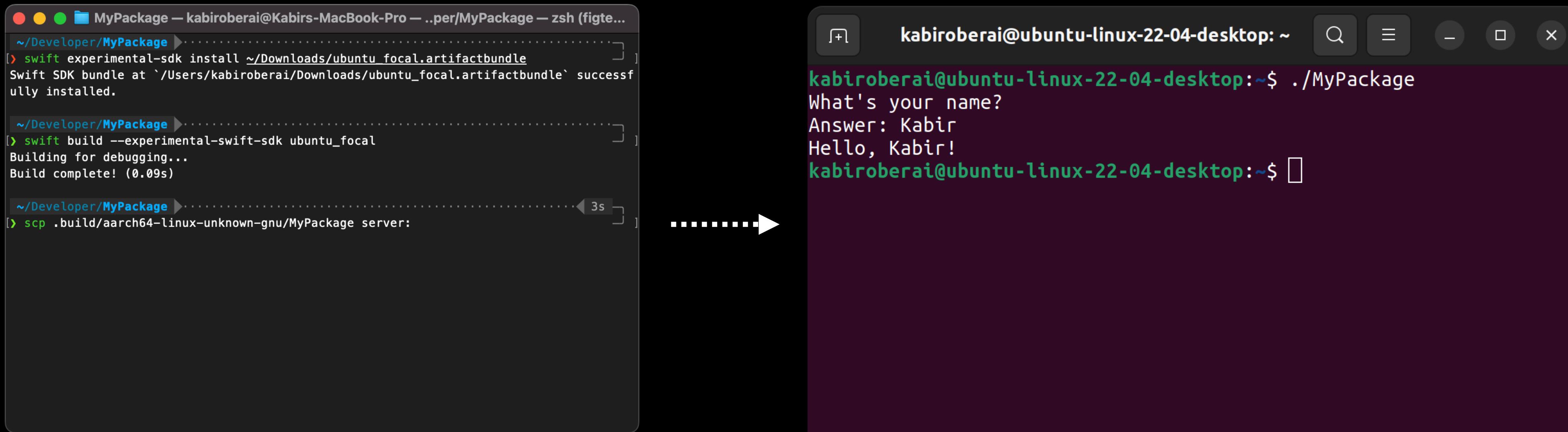


The screenshot shows a terminal window titled "MyPackage" running on a MacBook Pro. The terminal output demonstrates the following steps:

- The user installs an experimental Swift SDK from an Ubuntu focal artifact bundle: `swift experimental-sdk install ~/Downloads/ubuntu_focal.artifactbundle`. The message indicates the Swift SDK bundle was successfully installed at `/Users/kabiroberai/Downloads/ubuntu_focal.artifactbundle`.
- The user builds the package using the experimental Swift SDK: `swift build --experimental-swift-sdk ubuntu_focal`. The message shows the build was successful and completed in 0.09 seconds.
- The user uses SCP to transfer the built artifacts to a server: `scp .build/aarch64-linux-unknown-gnu/MyPackage server:`. A timer in the bottom right corner shows the command took 3 seconds to execute.

myth 5: You can only build Swift code for your own platform.

reality: Swift is a cross-compiler 🎭 and SwiftPM is too!



```
~/Developer/MyPackage ➤ swift experimental-sdk install ~/Downloads/ubuntu_focal.artifactbundle
Swift SDK bundle at `/Users/kabiroberai/Downloads/ubuntu_focal.artifactbundle` successfully installed.

~/Developer/MyPackage ➤ swift build --experimental-swift-sdk ubuntu_focal
Building for debugging...
Build complete! (0.09s)

~/Developer/MyPackage ➤ scp .build/aarch64-linux-unknown-gnu/MyPackage server:
```

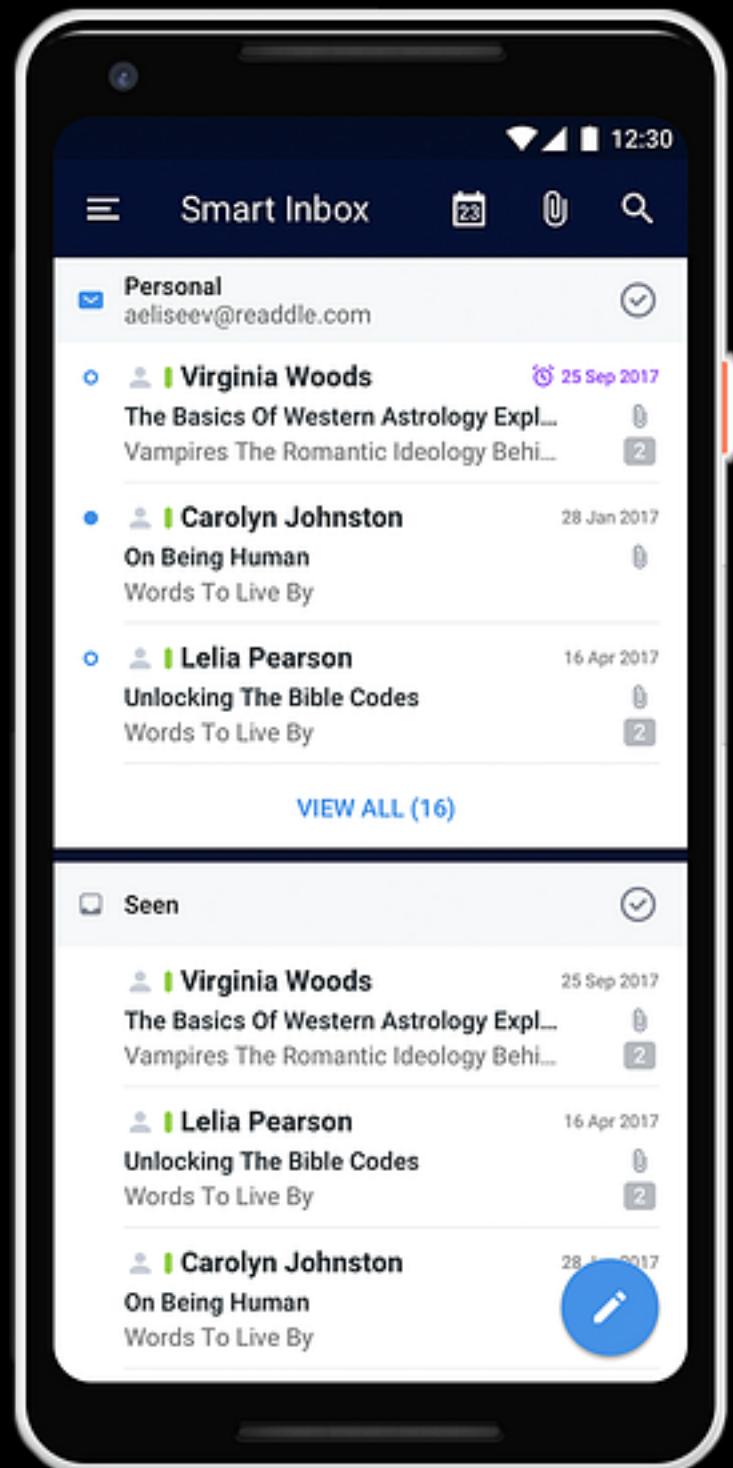
..... ➤

```
kabiroberai@ubuntu-linux-22-04-desktop:~$ ./MyPackage
What's your name?
Answer: Kabir
Hello, Kabir!
kabiroberai@ubuntu-linux-22-04-desktop:~$
```

myth 6: *Swift only targets Darwin, Linux, and Windows.*

myth 6: *Swift only targets Darwin, Linux, and Windows.*

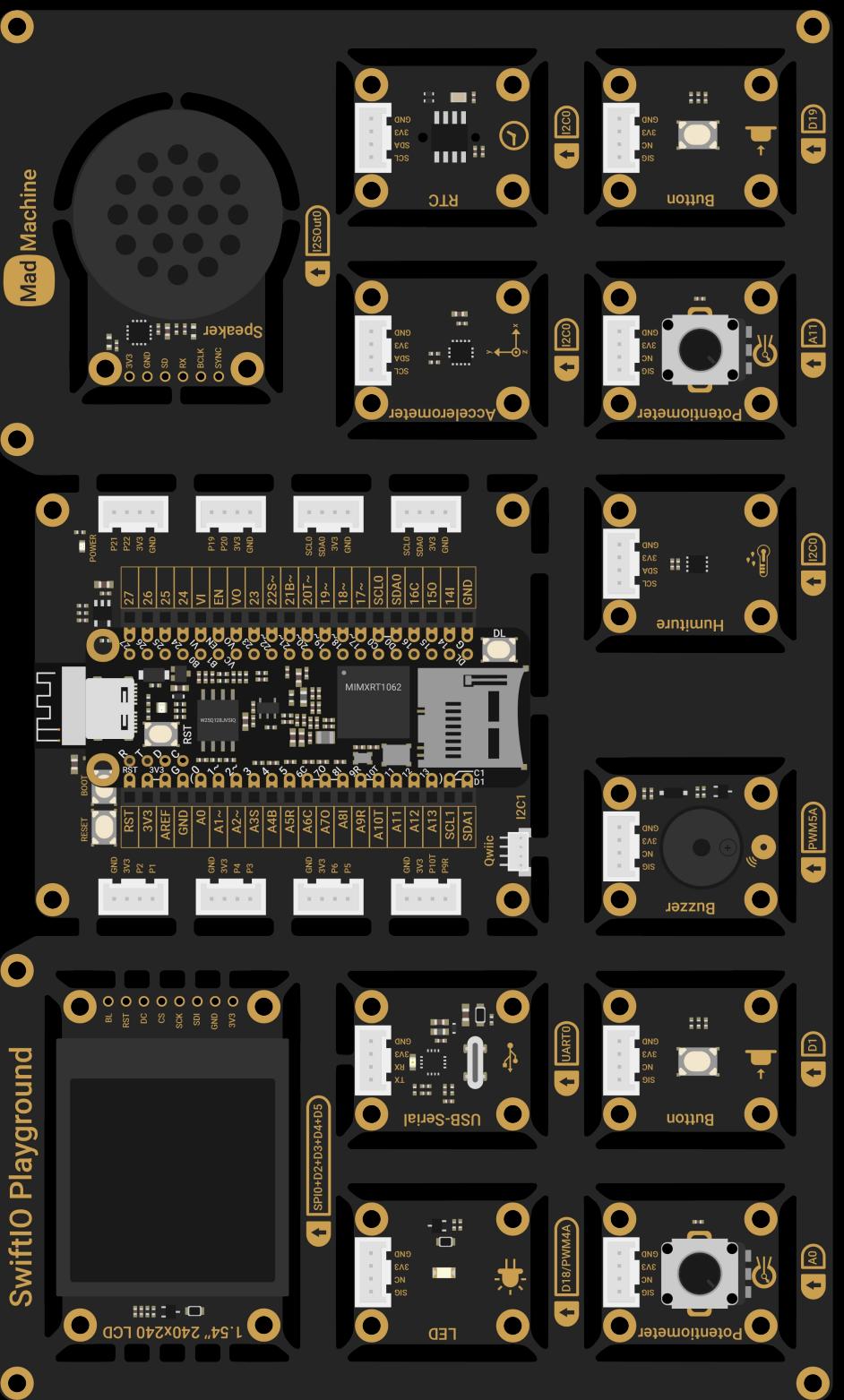
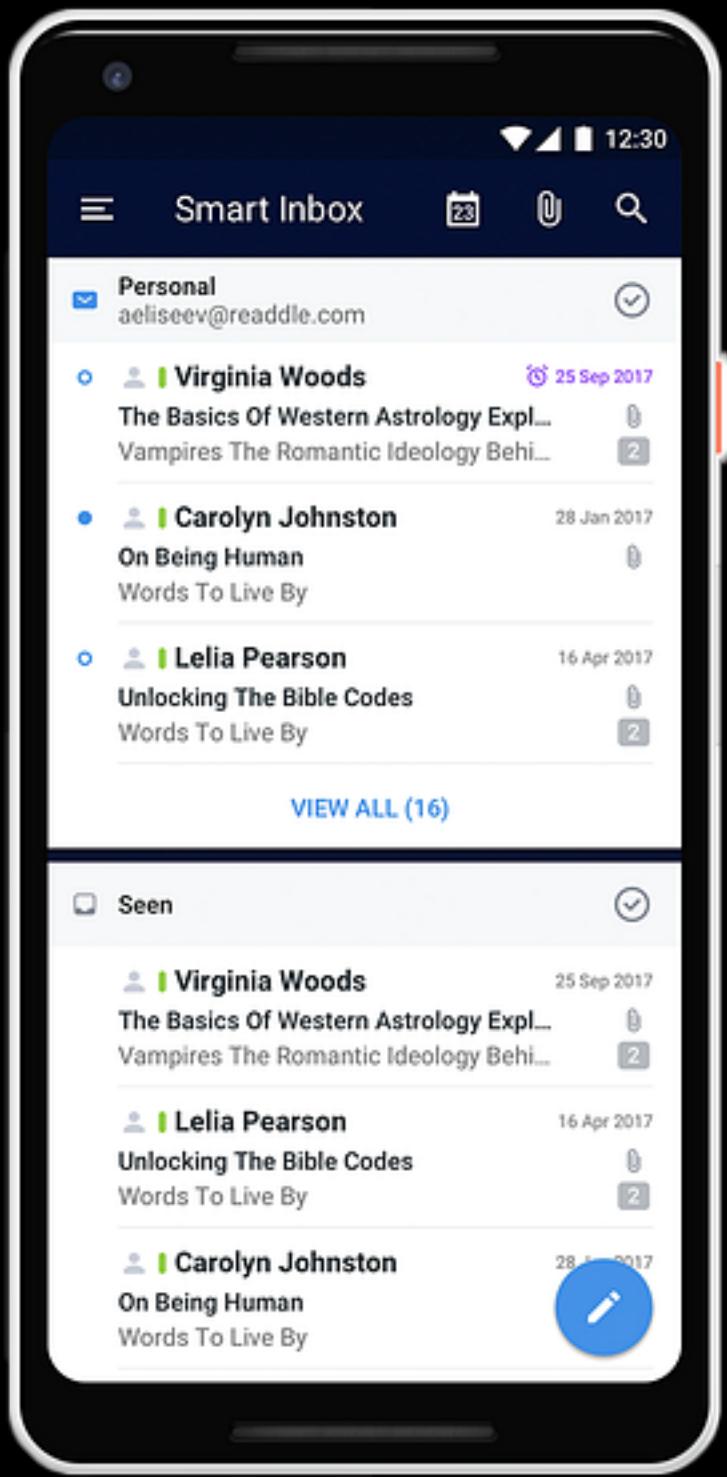
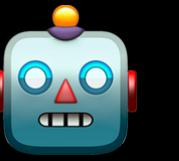
reality: Many other platforms are supported



Spark for Android ([Readdle](#))

myth 6: *Swift only targets Darwin, Linux, and Windows.*

reality: Many other platforms are supported

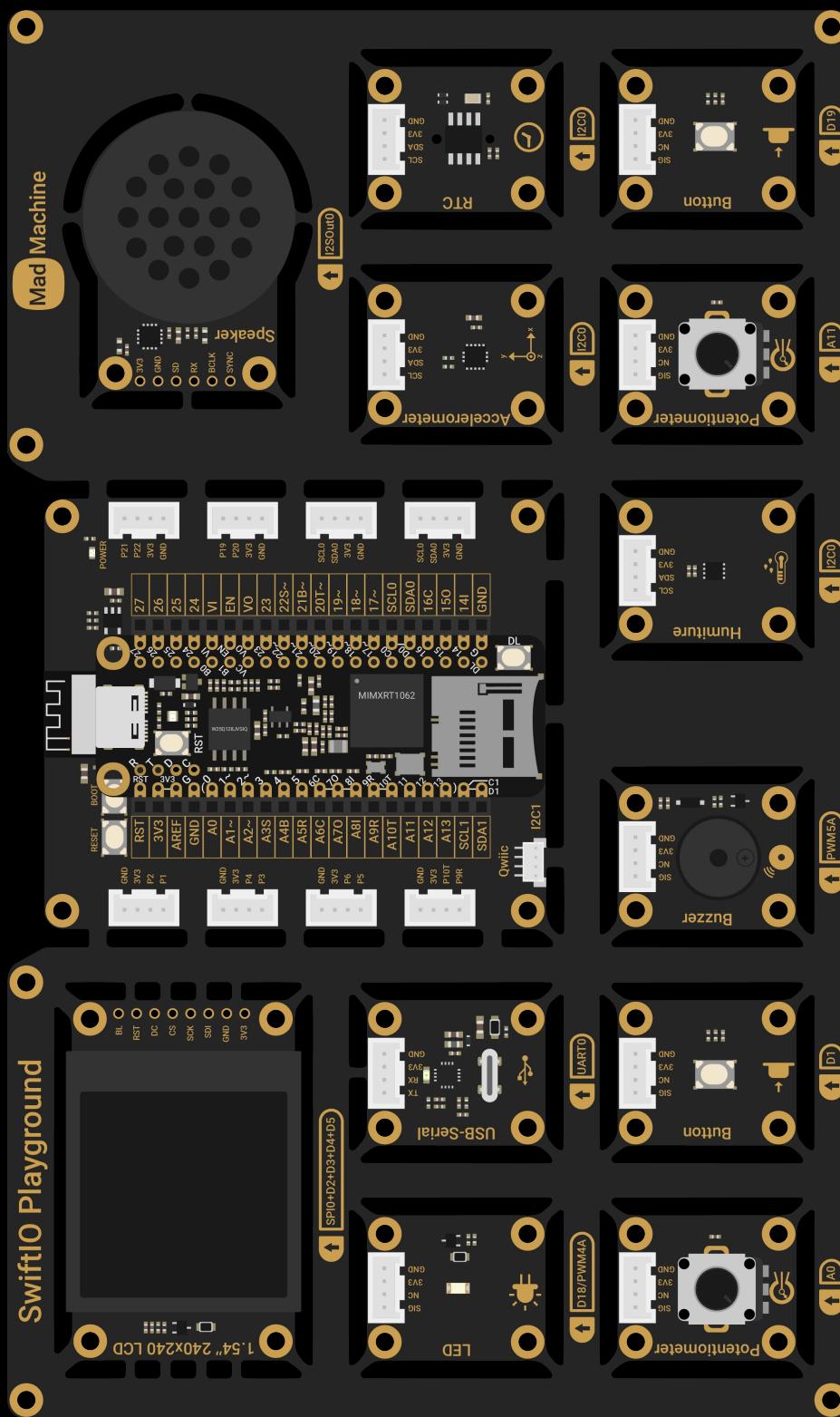
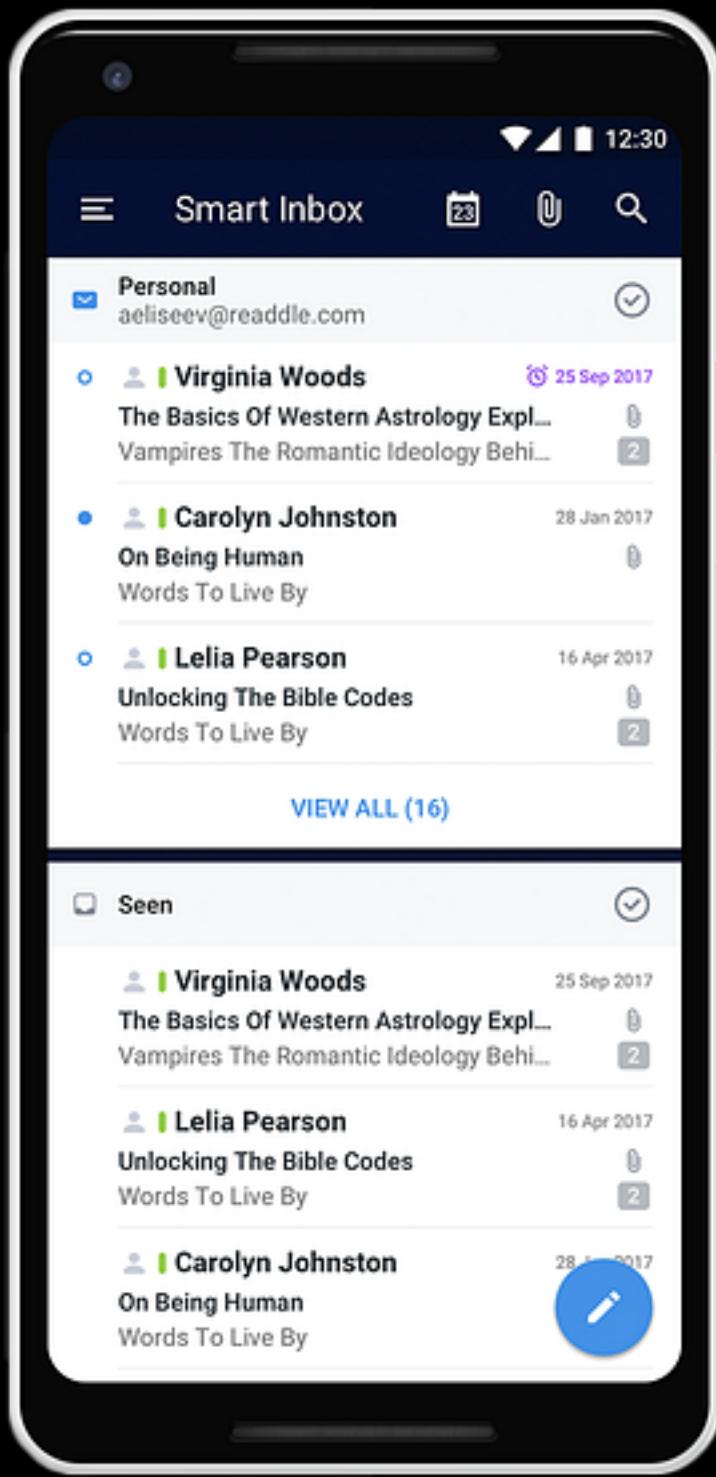


Spark for Android ([Readdle](#))

embedded/bare metal ([MadMachine](#))

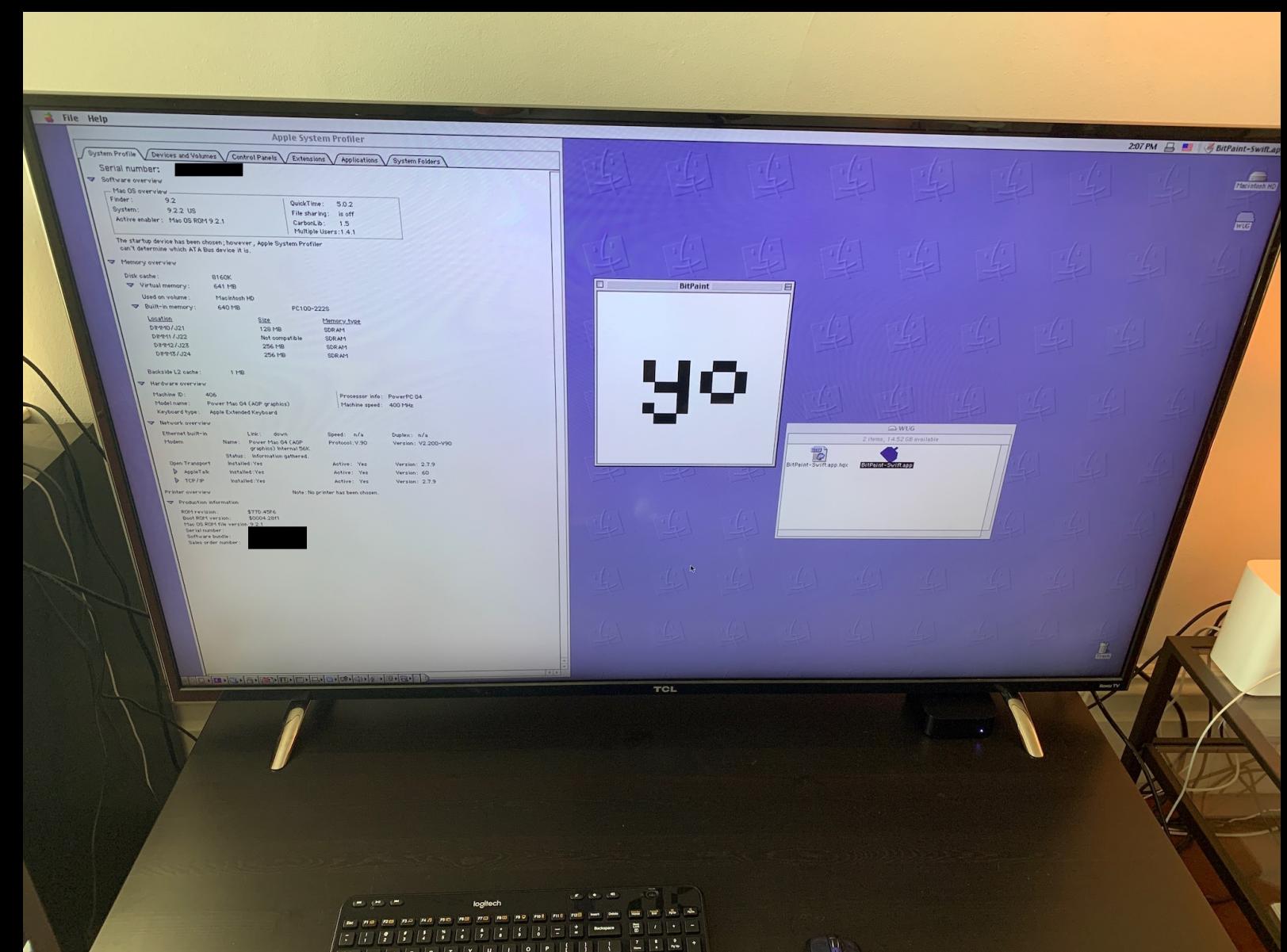
myth 6: Swift only targets Darwin, Linux, and Windows.

reality: Many other platforms are supported



Spark for Android ([Readdle](#))

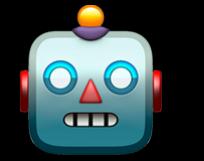
embedded/bare metal ([MadMachine](#))



and... Mac OS 9 ([Jordan Rose](#))

myth 6: *Swift only targets Darwin, Linux, and Windows.*

reality: Many other platforms are supported



The screenshot shows a web-based development environment for Swift. On the left, there's a code editor with dark-themed syntax highlighting. The code is written in Swift and uses the Tokamak framework. It defines a `Counter` view that increments a count and displays it, with a limit of 10. It also handles the `onAppear` and `onDisappear` events. On the right, there's a preview window showing a dark-themed UI with a button labeled "Increment" and a text field showing the value "2". Below the preview is a console window showing the output of the `onAppear` event. The browser address bar shows `pad.swiftwasm.org`.

```
import TokamakShim
struct Counter: View {
    @State var count: Int = 0

    let limit: Int

    @ViewBuilder
    public var body: some View {
        if count < limit {
            VStack {
                Button("Increment") { count += 1 }
                Text("\(count)")
            }
            .onAppear { print("Counter.VStack onAppear") }
            .onDisappear { print("Counter.VStack onDisappear") }
        } else {
            VStack { Text("Limit exceeded") }
        }
    }
}

import TokamakPreview
struct MyApp: PreviewApp {
    var body: some Scene {
        WindowGroup("Tokamak Demo") {
            Counter(limit: 10)
        }
    }
}
MyApp.main()
```

Swift in WebAssembly with [SwiftWasm](#) and [Tokamak](#)

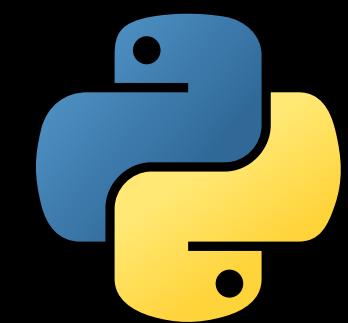
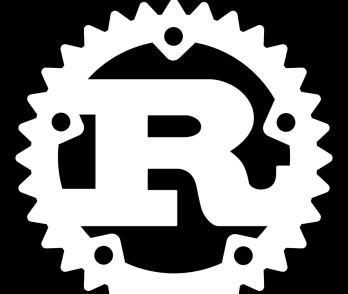
myth 7: *When choosing a cross-platform language, it's either/or.*

myth 7: *When choosing a cross-platform language, it's either/or.*

reality: Swift interoperates with more languages than you think 💋

built-in

rust-bindgen

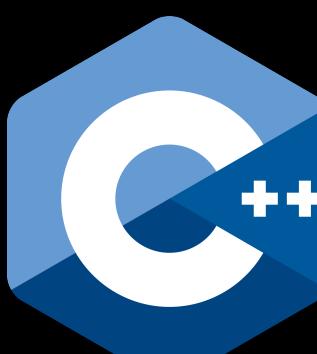


(...and more)

PythonKit



Kotlin Native



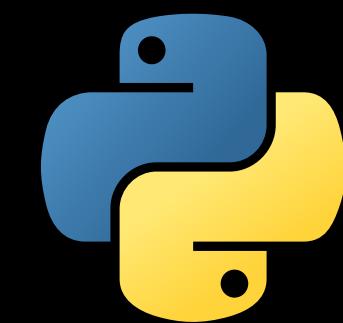
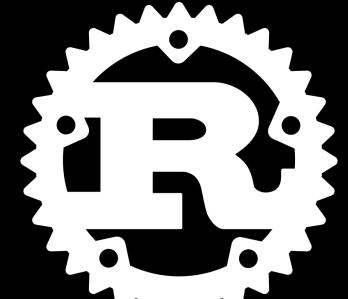
built-in (5.9+)

myth 7: *When choosing a cross-platform language, it's either/or.*

reality: Swift interoperates with more languages than you think 💋

built-in

rust-bindgen



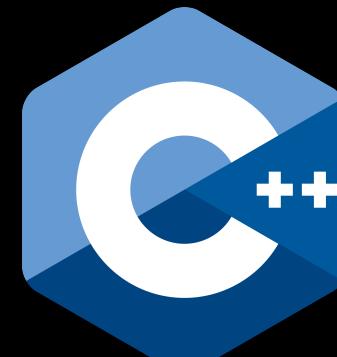
PythonKit



(...and more)



Kotlin Native



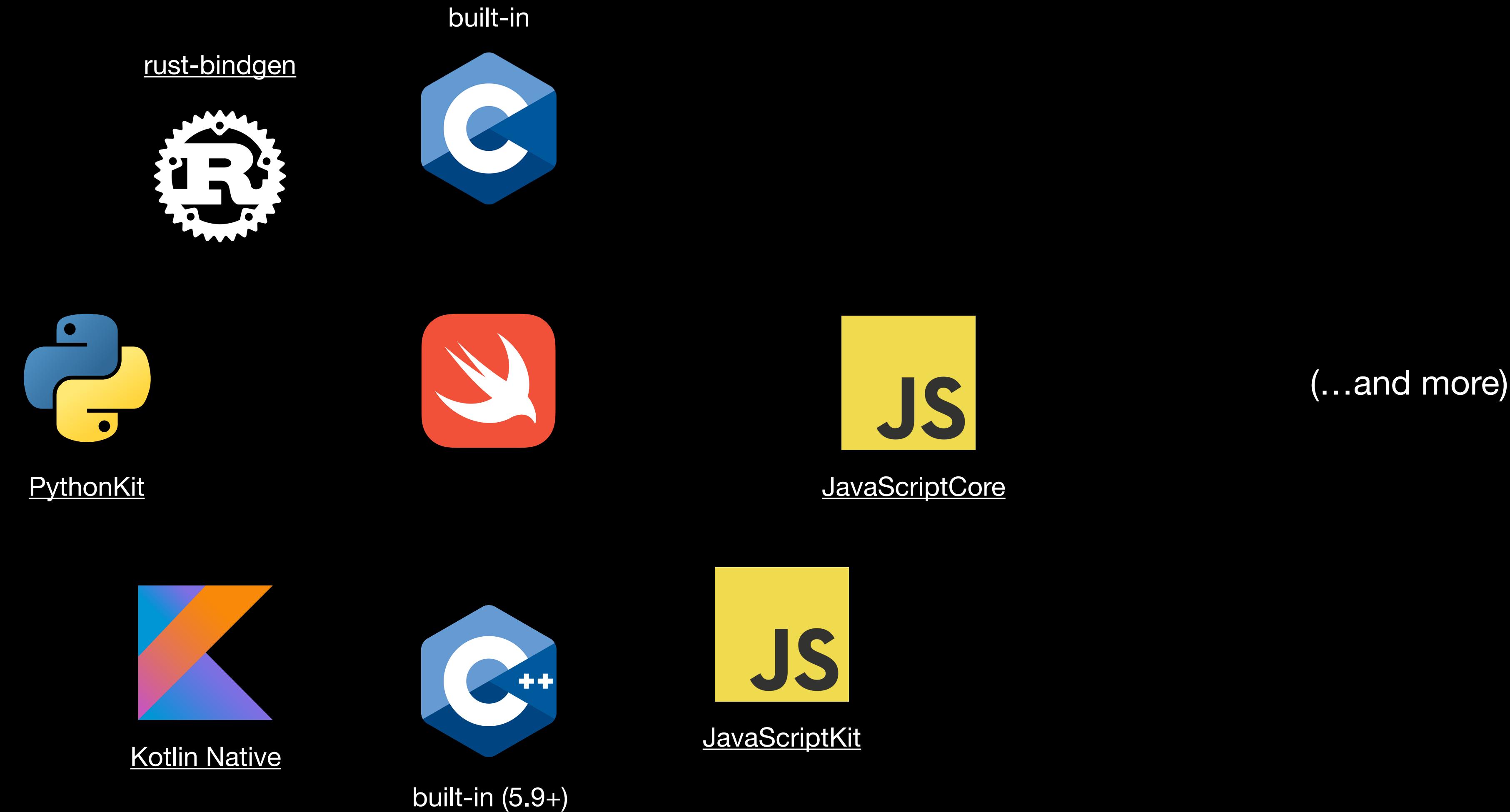
built-in (5.9+)



JavaScriptKit

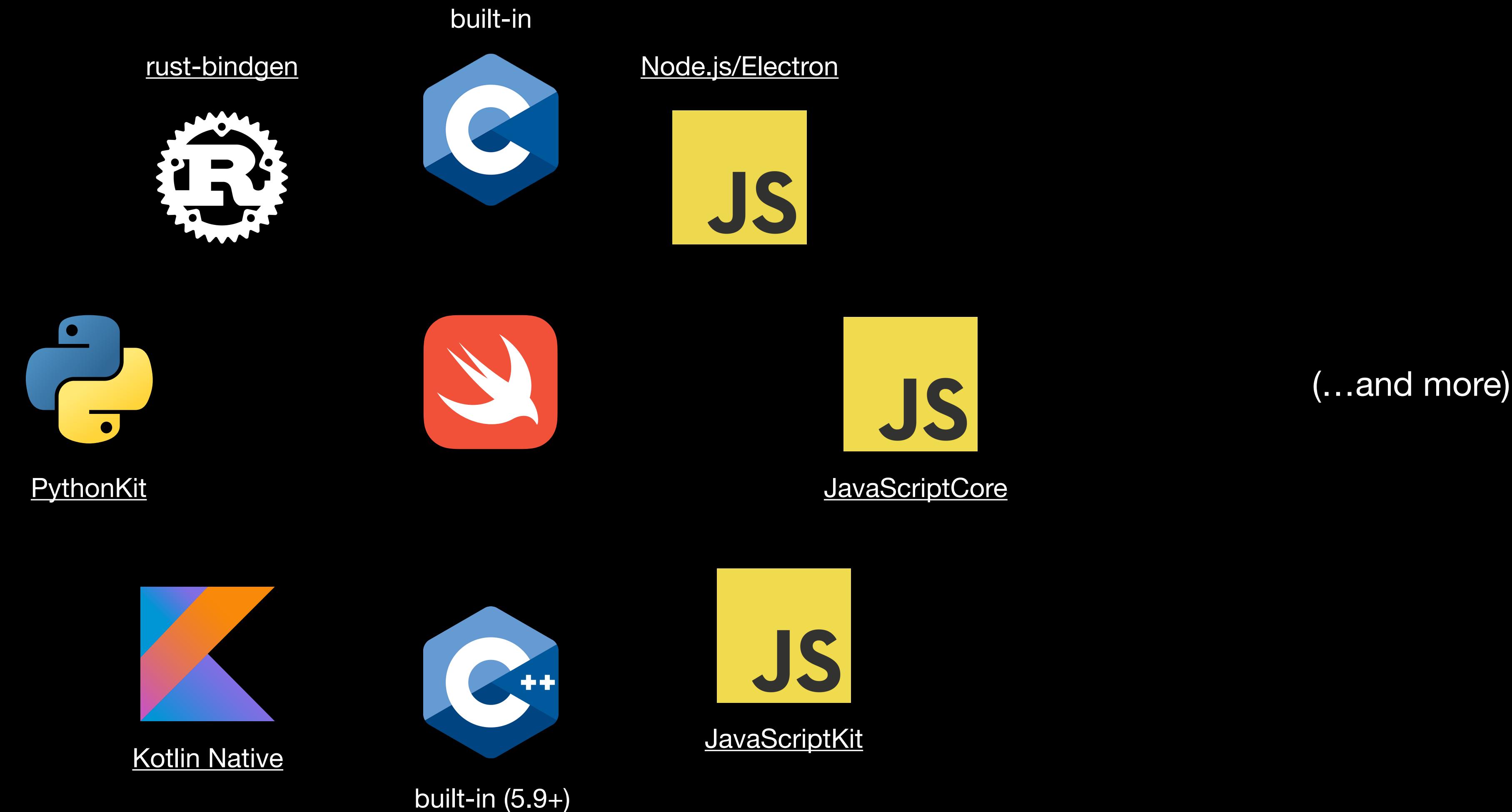
myth 7: *When choosing a cross-platform language, it's either/or.*

reality: Swift interoperates with more languages than you think 💋



myth 7: *When choosing a cross-platform language, it's either/or.*

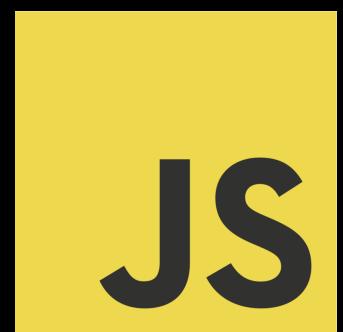
reality: Swift interoperates with more languages than you think 💋



myth 7: *When choosing a cross-platform language, it's either/or.*

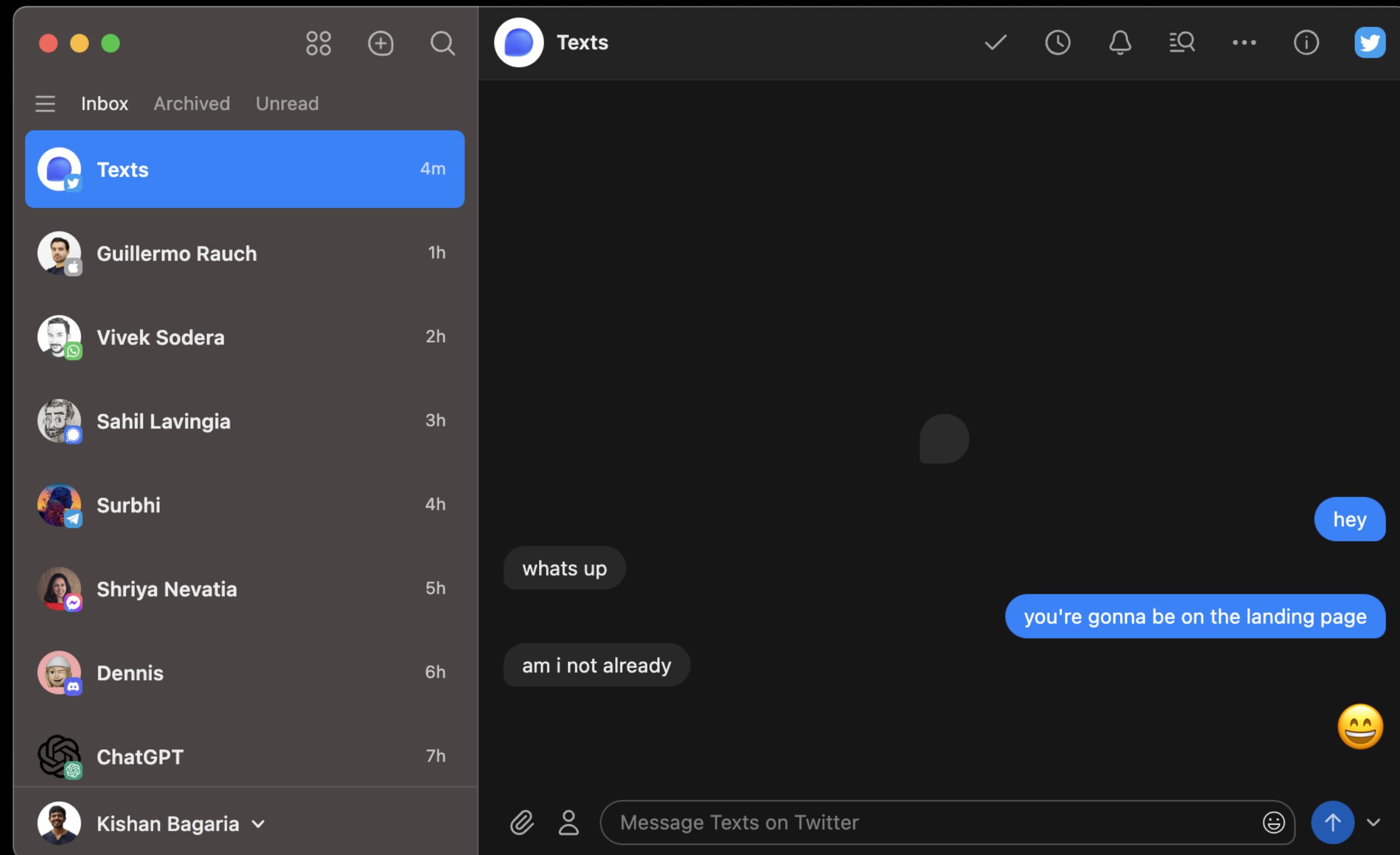
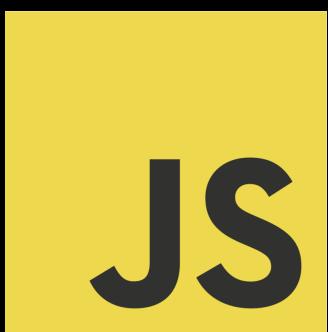
reality: Swift interoperates with more languages than you think 💋

[kabiroberai/NodeSwift](#)



myth 7: When choosing a cross-platform language, it's either/or.
reality: Swift interoperates with more languages than you think 💋

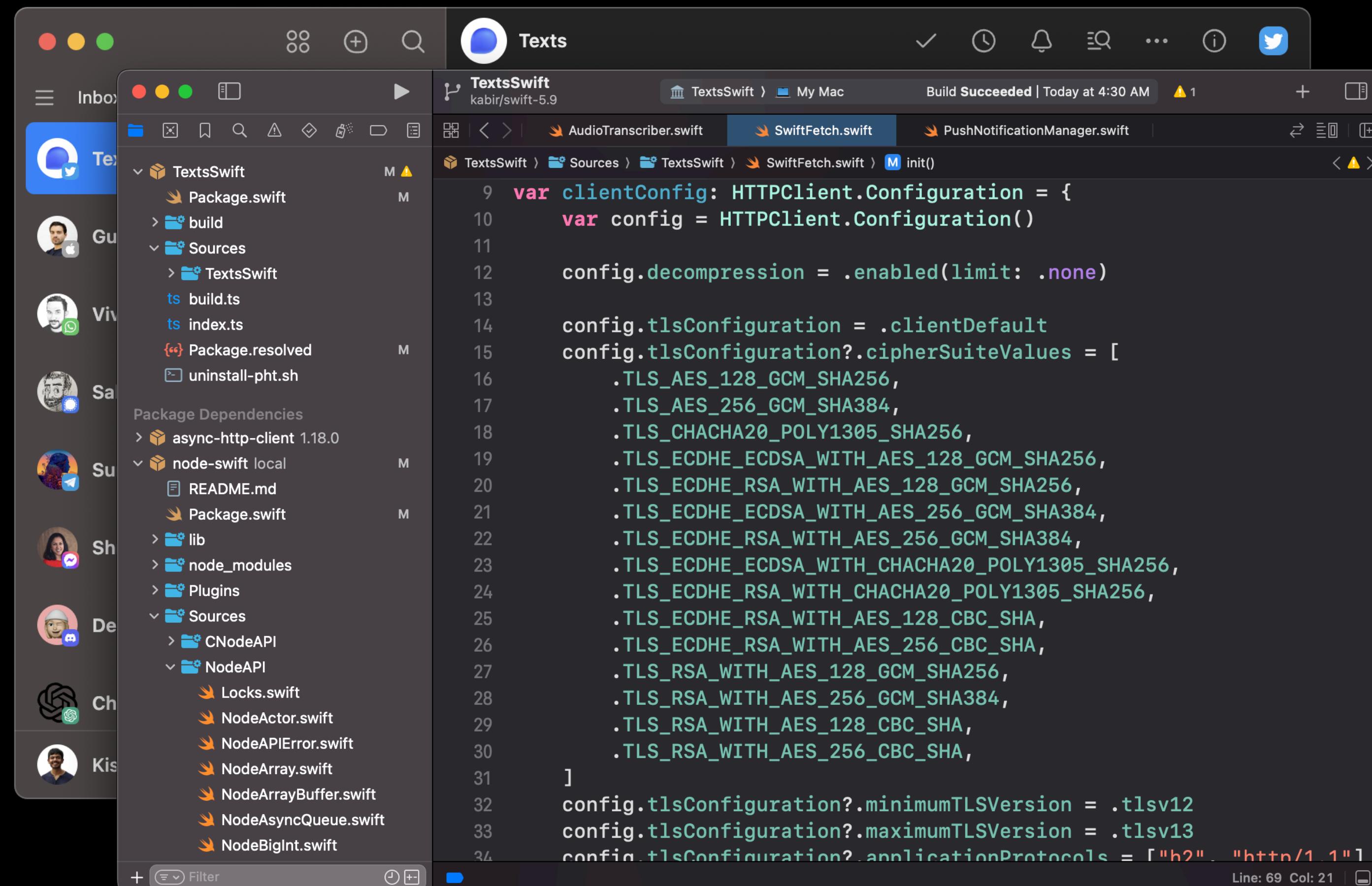
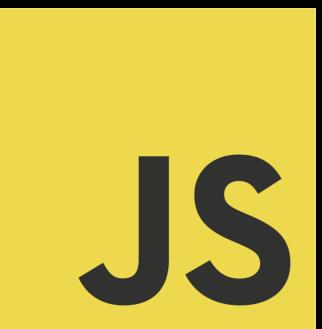
[kabiroberai/NodeSwift](#)



[Texts](#)

myth 7: When choosing a cross-platform language, it's either/or.
reality: Swift interoperates with more languages than you think 💋

[kabiroberai/NodeSwift](#)



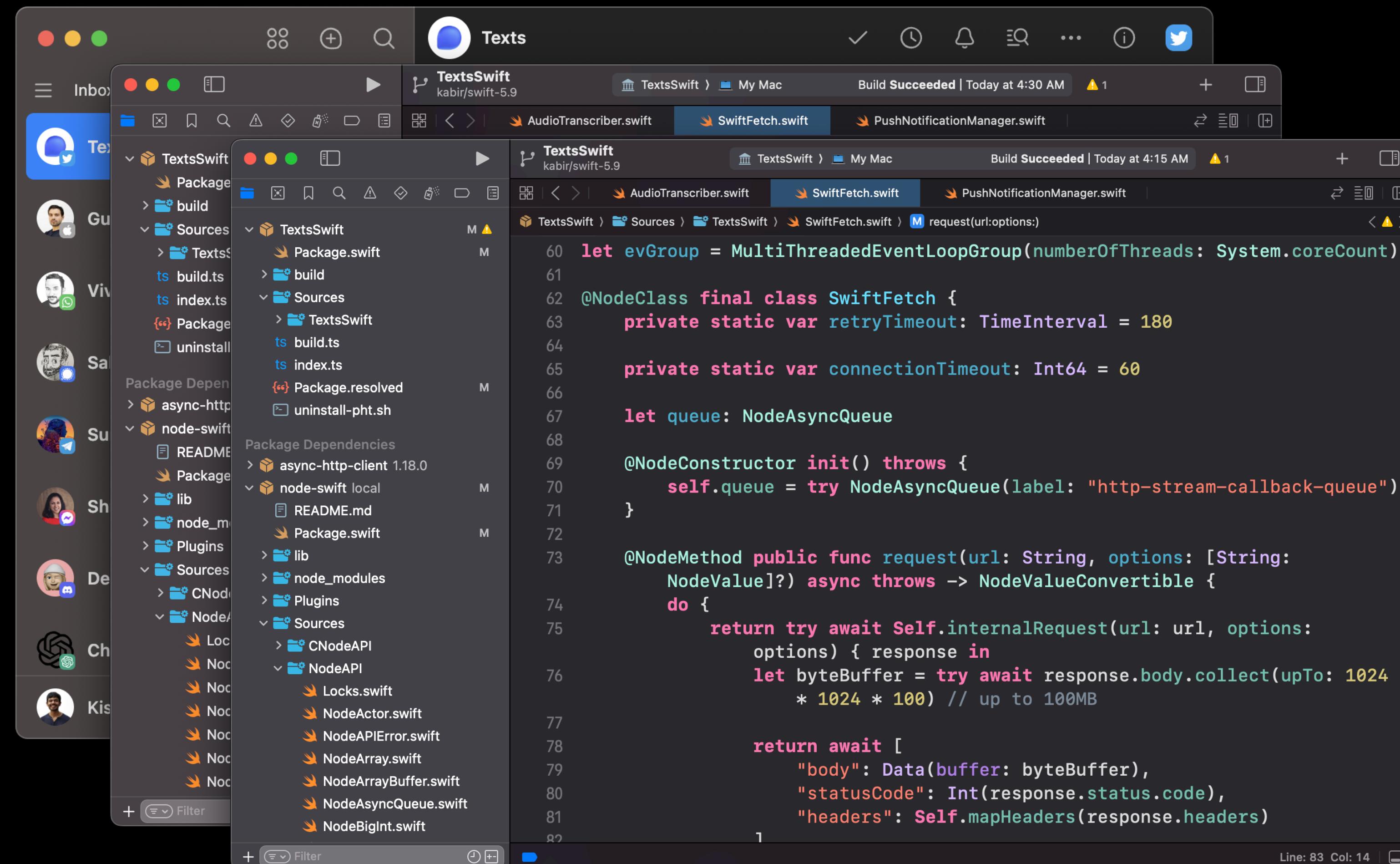
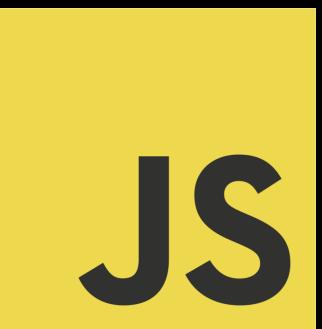
The screenshot shows the Xcode interface with a project named "TextsSwift". The left sidebar lists files and folders, including "TextsSwift", "build", "Sources", "TextsSwift", "build.ts", "index.ts", "Package.resolved", and "uninstall-pht.sh". Below these are "Package Dependencies" for "async-http-client 1.18.0" and "node-swift local". The main editor window displays Swift code for initializing an HTTP client configuration:

```
9 var clientConfig: HTTPClient.Configuration = {
10     var config = HTTPClient.Configuration()
11 
12     config.decompression = .enabled(limit: .none)
13 
14     config.tlsConfiguration = .clientDefault
15     config.tlsConfiguration?.cipherSuiteValues = [
16         .TLS_AES_128_GCM_SHA256,
17         .TLS_AES_256_GCM_SHA384,
18         .TLS_CHACHA20_POLY1305_SHA256,
19         .TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,
20         .TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,
21         .TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,
22         .TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,
23         .TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256,
24         .TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256,
25         .TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,
26         .TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,
27         .TLS_RSA_WITH_AES_128_GCM_SHA256,
28         .TLS_RSA_WITH_AES_256_GCM_SHA384,
29         .TLS_RSA_WITH_AES_128_CBC_SHA,
30         .TLS_RSA_WITH_AES_256_CBC_SHA,
31     ]
32     config.tlsConfiguration?.minimumTLSVersion = .tlsv12
33     config.tlsConfiguration?.maximumTLSVersion = .tlsv13
34     config.tlsConfiguration?.applicationProtocols = ["h2", "http/1.1"]
```

Texts

myth 7: When choosing a cross-platform language, it's either/or. reality: Swift interoperates with more languages than you think 💋

[kabiroberai/NodeSwift](https://github.com/kabiroberai/NodeSwift)



The screenshot shows the Xcode interface with two projects open side-by-side:

- Left Project (TextsSwift):** A Swift package named "TextsSwift". It contains a "Sources" folder with "TextsSwift.swift" and "SwiftFetch.swift". The "SwiftFetch.swift" file contains the following code:

```
let evGroup = MultiThreadedEventLoopGroup(numberOfThreads: System.coreCount)
@NodeClass final class SwiftFetch {
    private static var retryTimeout: TimeInterval = 180
    private static var connectionTimeout: Int64 = 60
    let queue: NodeAsyncQueue
    @NodeConstructor init() throws {
        self.queue = try NodeAsyncQueue(label: "http-stream-callback-queue")
    }
    @NodeMethod public func request(url: String, options: [String: NodeValue]?) async throws -> NodeValueConvertible {
        do {
            return try await Self.internalRequest(url: url, options: options)
        } catch {
            let byteBuffer = try await response.body.collect(upTo: 1024 * 1024 * 100) // up to 100MB
            return await [
                "body": Data(buffer: byteBuffer),
                "statusCode": Int(response.status.code),
                "headers": Self.mapHeaders(response.headers)
            ]
        }
    }
}
```

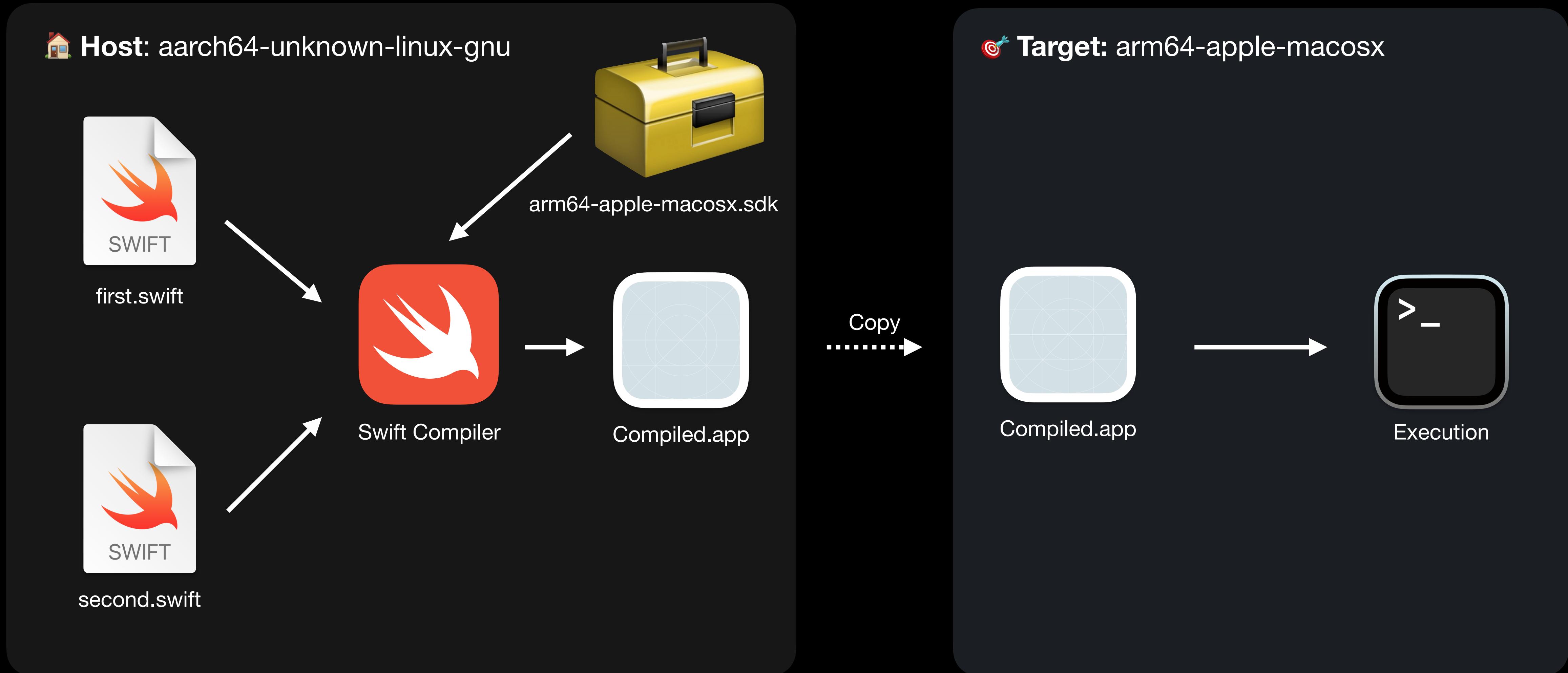
- Right Project (TextsSwift):** Another instance of the "TextsSwift" project. The "Sources" folder contains "TextsSwift.swift" and "SwiftFetch.swift". The "SwiftFetch.swift" file is identical to the one in the left project.

Texts

interlude

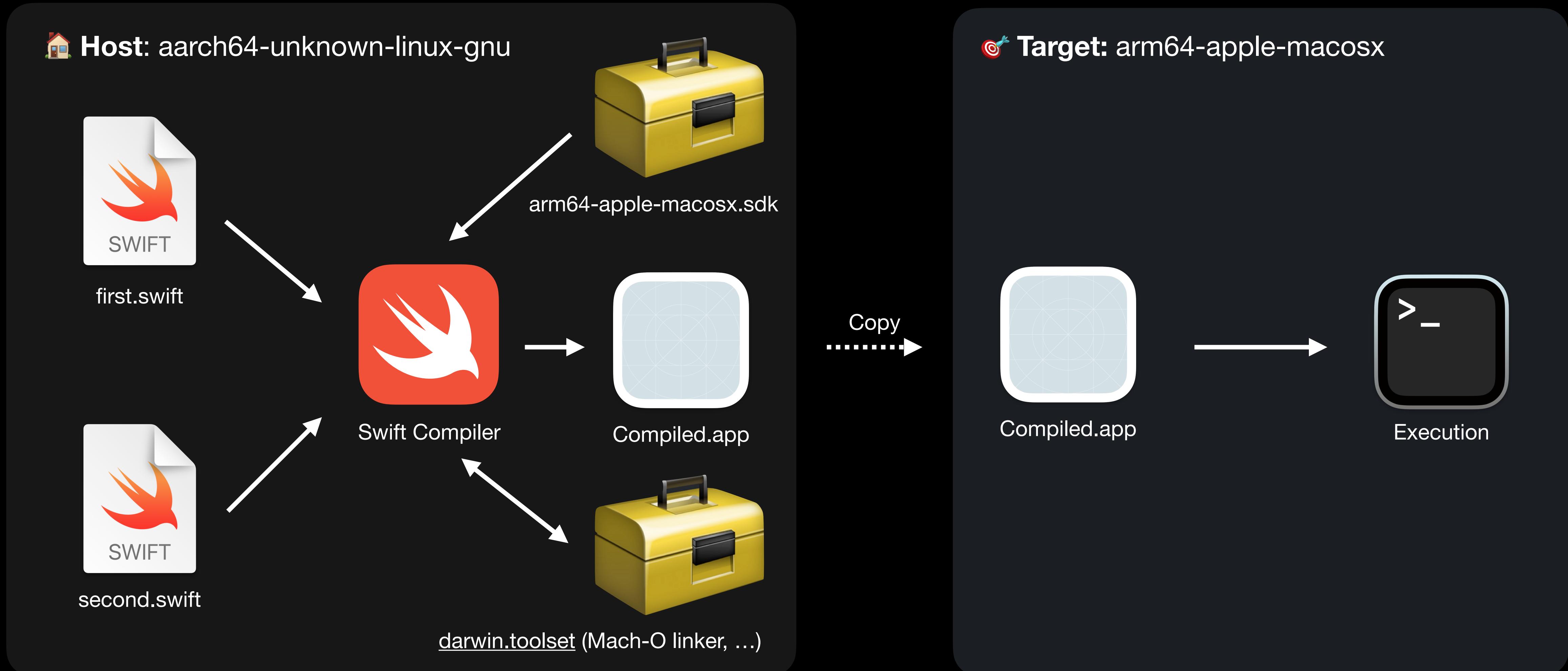
host = Linux
target = macOS?

interlude: more theory



from Linux to macOS?

interlude: more theory

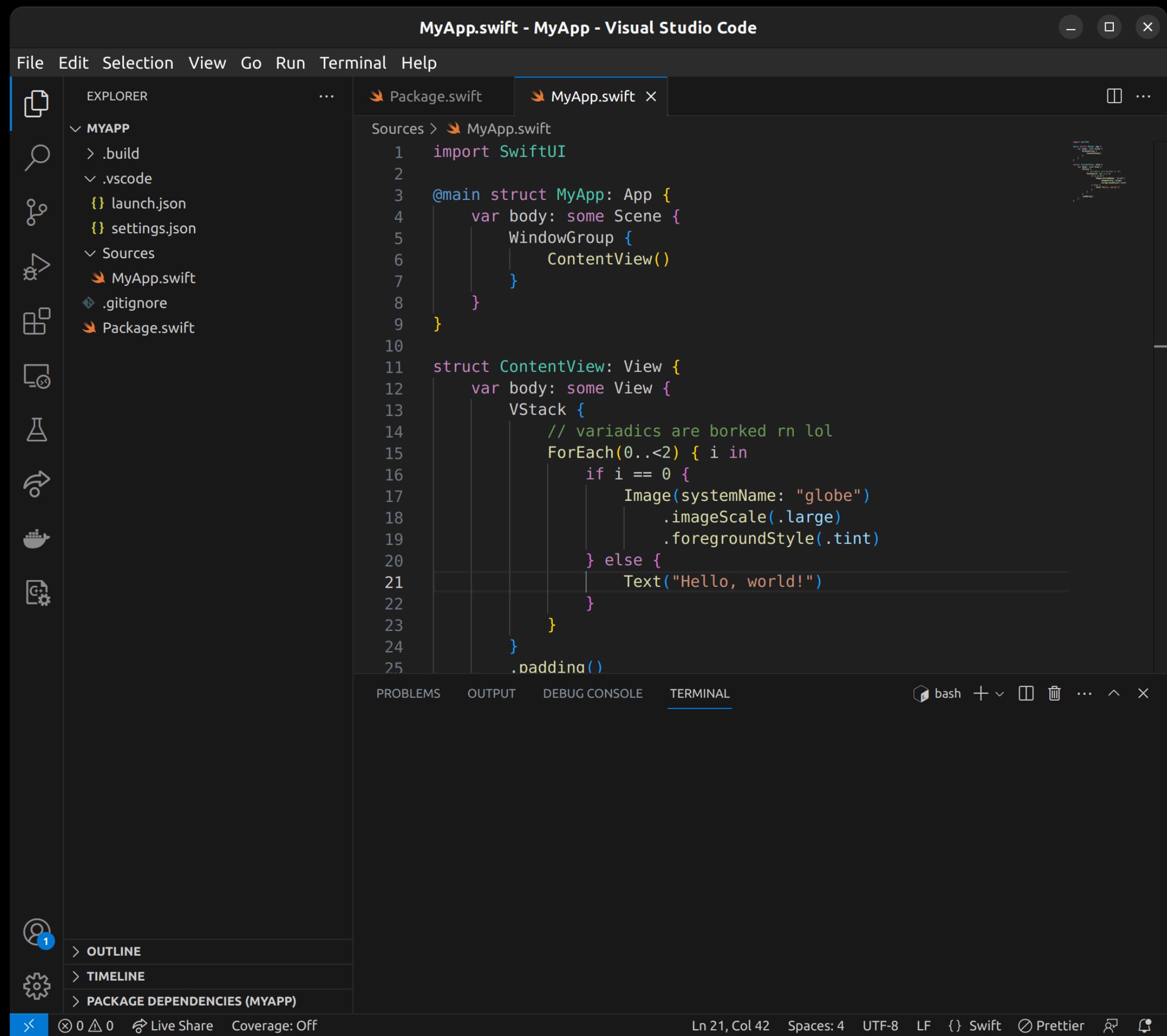


from Linux to macOS?

myth 8: *macOS Swift apps can only be built on macOS*

myth 8: macOS Swift apps can only be built on macOS

reality: Cross-compilation doesn't discriminate



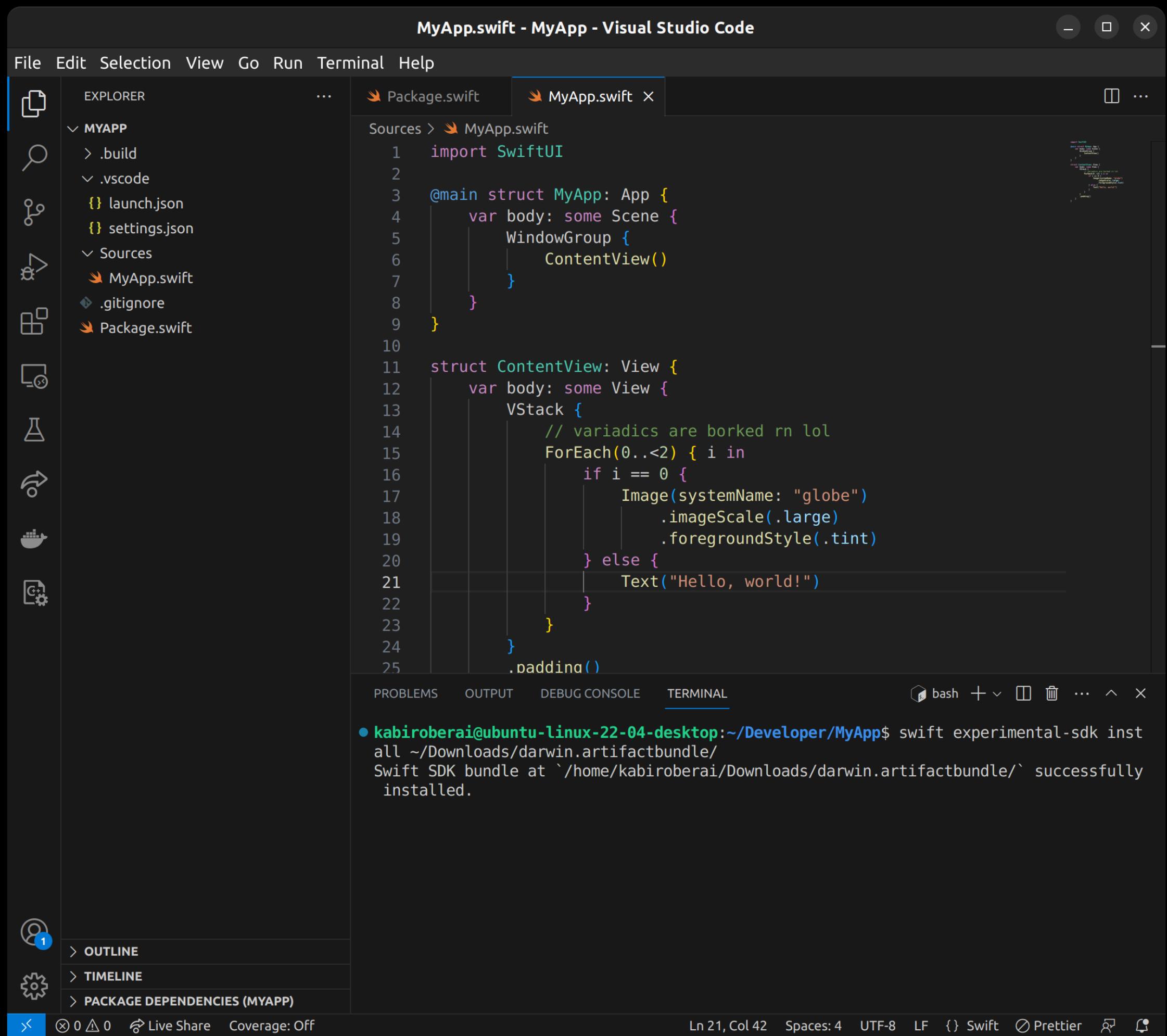
A screenshot of the Visual Studio Code interface. The title bar says "MyApp.swift - MyApp - Visual Studio Code". The left sidebar shows a file tree with a project named "MYAPP" containing ".build", ".vscode", "Sources", and "Package.swift". The main editor tab is "MyApp.swift". The code in the editor is:

```
1 import SwiftUI
2
3 @main struct MyApp: App {
4     var body: some Scene {
5         WindowGroup {
6             ContentView()
7         }
8     }
9 }
10
11 struct ContentView: View {
12     var body: some View {
13         VStack {
14             // variadics are borked rn lol
15             ForEach(0..<2) { i in
16                 if i == 0 {
17                     Image(systemName: "globe")
18                         .imageScale(.large)
19                         .foregroundStyle(.tint)
20                 } else {
21                     Text("Hello, world!")
22                 }
23             }
24         }
25     }
26 }
```

The bottom status bar shows "Ln 21, Col 42 Spaces: 4 UTF-8 LF {} Swift Prettier".

myth 8: macOS Swift apps can only be built on macOS

reality: Cross-compilation doesn't discriminate



The screenshot shows a Visual Studio Code window with a dark theme. The title bar reads "MyApp.swift - MyApp - Visual Studio Code". The left sidebar has icons for File, Edit, Selection, View, Go, Run, Terminal, and Help. The Explorer sidebar shows a project structure for "MYAPP" with files like ".build", ".vscode", "Sources", and "Package.swift". The main editor tab is "MyApp.swift", which contains the following Swift code:

```
1 import SwiftUI
2
3 @main struct MyApp: App {
4     var body: some Scene {
5         WindowGroup {
6             ContentView()
7         }
8     }
9 }
10
11 struct ContentView: View {
12     var body: some View {
13         VStack {
14             // variadics are borked rn lol
15             ForEach(0..<2) { i in
16                 if i == 0 {
17                     Image(systemName: "globe")
18                         .imageScale(.large)
19                         .foregroundStyle(.tint)
20                 } else {
21                     Text("Hello, world!")
22                 }
23             }
24         }
25     }
26     .padding()
27 }
```

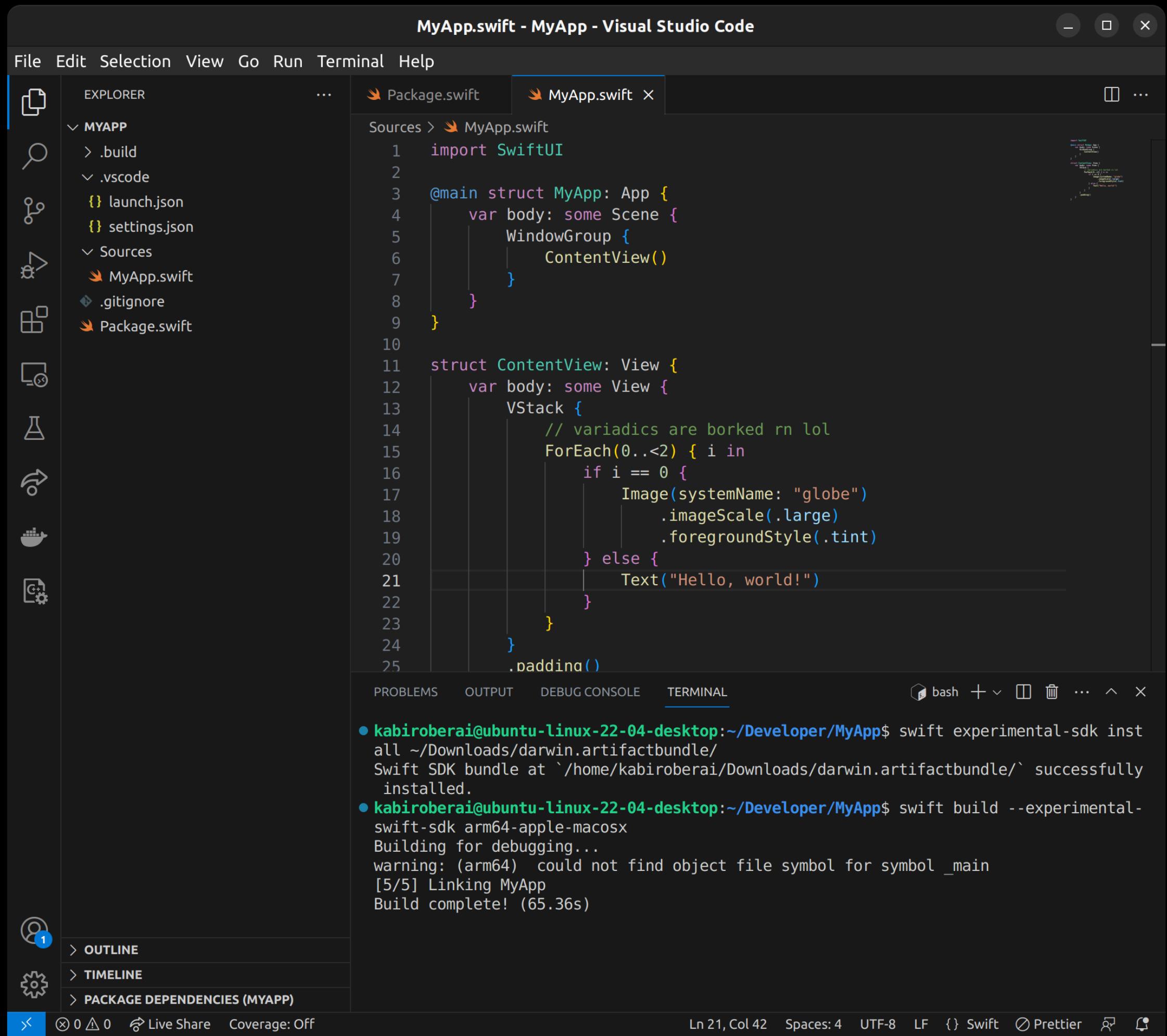
The terminal tab at the bottom shows a command-line session:

```
● kabiroberai@ubuntu-linux-22-04-desktop:~/Developer/MyApp$ swift experimental-sdk install ~/Downloads/darwin.artifactbundle/
Swift SDK bundle at `/home/kabiroberai/Downloads/darwin.artifactbundle/' successfully installed.
```

At the bottom, status bar items include "Ln 21, Col 42", "Spaces: 4", "UTF-8", "LF", "Swift", "Prettier", "Live Share", "Coverage: Off", and file navigation icons.

myth 8: macOS Swift apps can only be built on macOS

reality: Cross-compilation doesn't discriminate



The screenshot shows a Visual Studio Code interface with a dark theme. The left sidebar displays a project structure for 'MYAPP' containing files like .build, .vscode, launch.json, settings.json, Sources, MyApp.swift, .gitignore, and Package.swift. The main editor window shows the contents of 'MyApp.swift'. The code defines an @main struct MyApp that contains a var body: some Scene with a WindowGroup and ContentView(). The ContentView struct contains a VStack with two Image views for a globe and a Text view with the message "Hello, world!". The bottom right of the editor shows a terminal window with the following output:

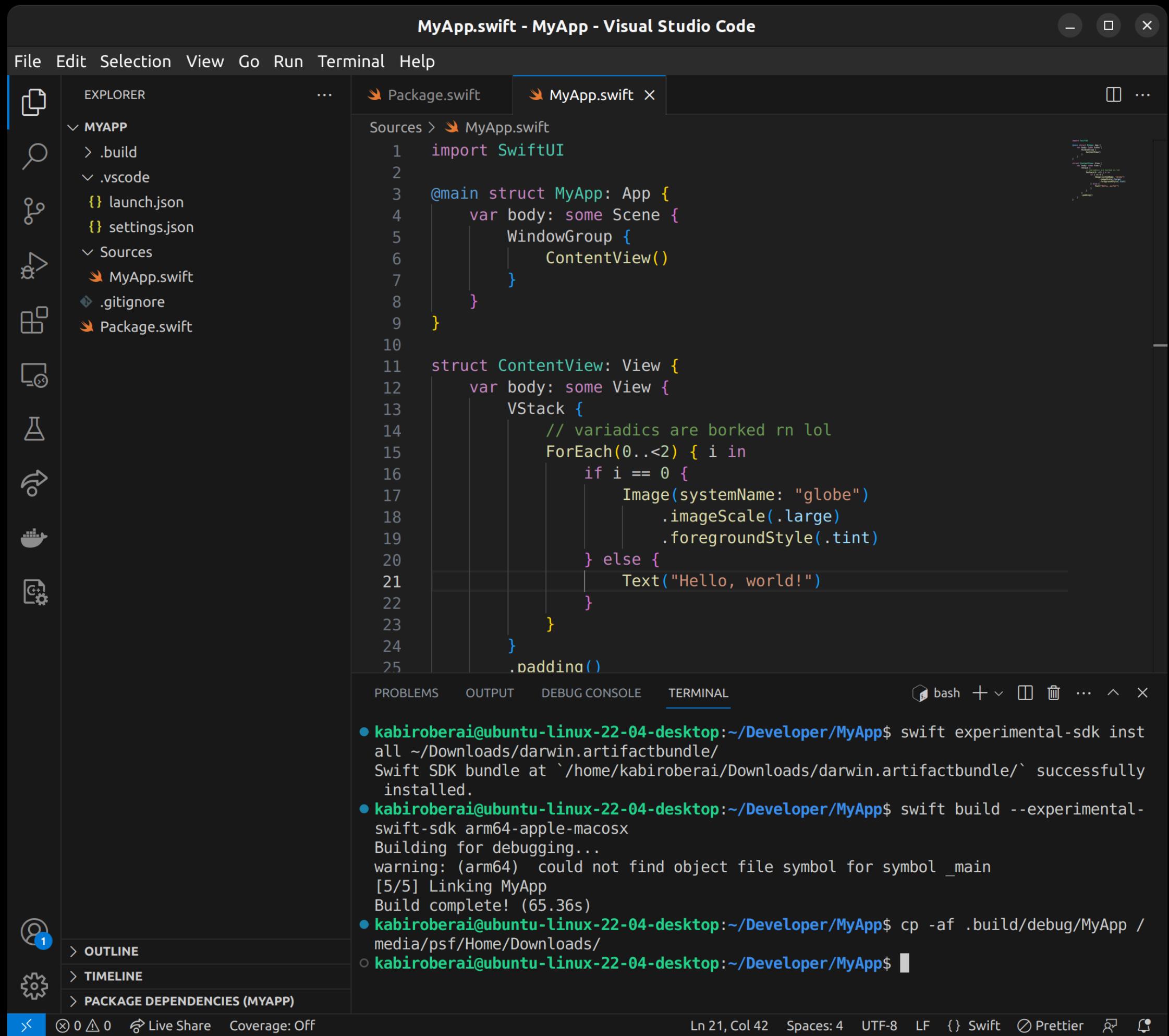
```
kabiroberai@ubuntu-linux-22-04-desktop:~/Developer/MyApp$ swift experimental-sdk install ~/Downloads/darwin.artifactbundle/
Swift SDK bundle at `/home/kabiroberai/Downloads/darwin.artifactbundle/` successfully installed.
kabiroberai@ubuntu-linux-22-04-desktop:~/Developer/MyApp$ swift build --experimental-swift-sdk arm64-apple-macosx
Building for debugging...
warning: (arm64) could not find object file symbol for symbol _main
[5/5] Linking MyApp
Build complete! (65.36s)
```

At the bottom of the interface, there are status indicators for live share and coverage, along with file navigation controls.

[kabiroberai/swift-sdk-darwin](https://github.com/kabiroberai/swift-sdk-darwin)

myth 8: macOS Swift apps can only be built on macOS

reality: Cross-compilation doesn't discriminate



The screenshot shows a Visual Studio Code interface with a dark theme. The left sidebar has icons for Explorer, Search, Open, and others. The main area shows an 'EXPLORER' view with a project named 'MYAPP'. Inside 'Sources' are files like 'MyApp.swift' and 'ContentView.swift'. The 'MyApp.swift' file is open in the editor, displaying Swift code for a SwiftUI app. The 'TERMINAL' tab at the bottom shows command-line output from a Linux system (Ubuntu 22.04) demonstrating the cross-compilation of a Swift application:

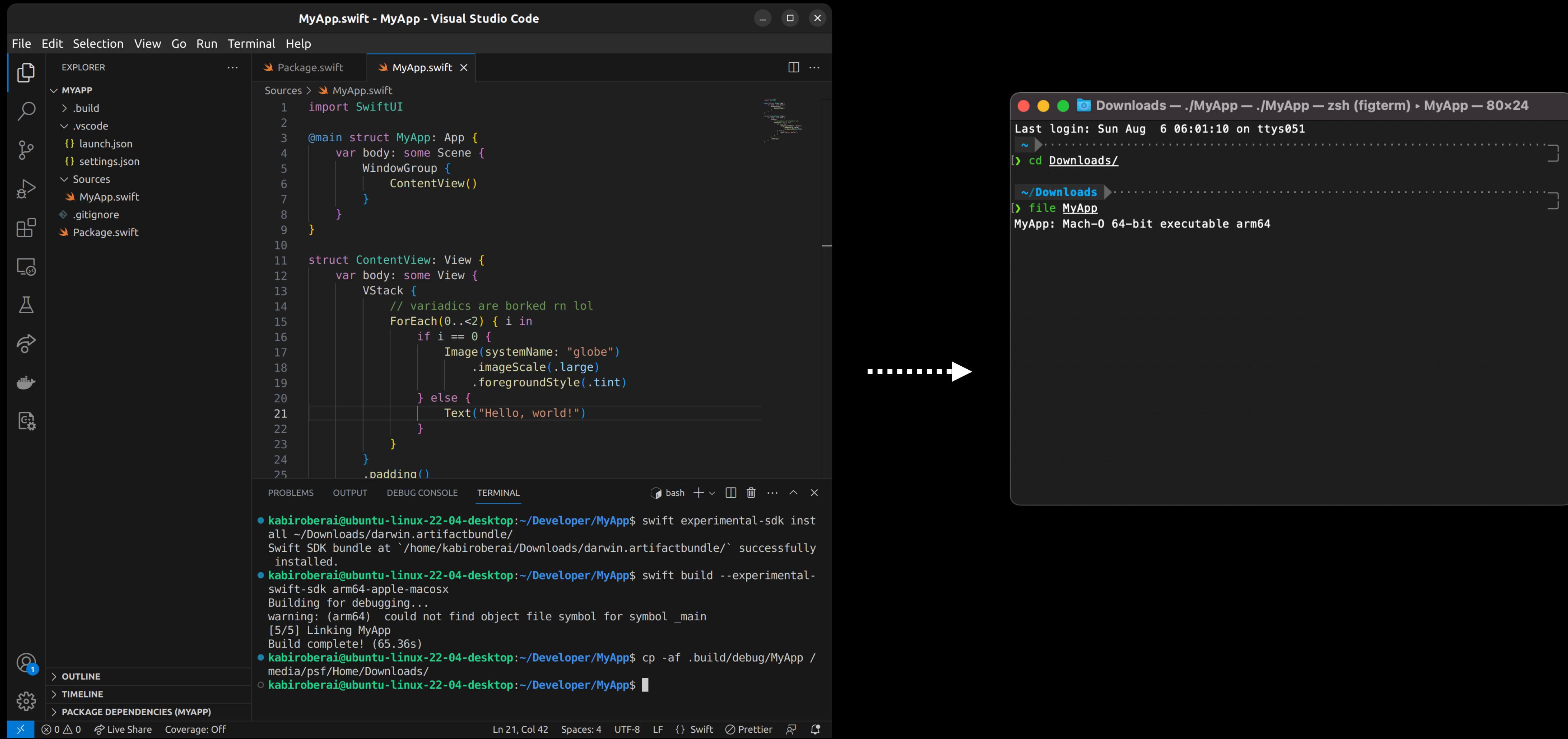
```
kabiroberai@ubuntu-linux-22-04-desktop:~/Developer/MyApp$ swift experimental-sdk install ~/Downloads/darwin.artifactbundle/
Swift SDK bundle at `/home/kabiroberai/Downloads/darwin.artifactbundle/` successfully installed.
kabiroberai@ubuntu-linux-22-04-desktop:~/Developer/MyApp$ swift build --experimental-swift-sdk arm64-apple-macosx
Building for debugging...
warning: (arm64) could not find object file symbol for symbol _main
[5/5] Linking MyApp
Build complete! (65.36s)
kabiroberai@ubuntu-linux-22-04-desktop:~/Developer/MyApp$ cp -af .build/debug/MyApp /media/psf/Home/Downloads/
kabiroberai@ubuntu-linux-22-04-desktop:~/Developer/MyApp$
```

At the bottom, status bar items include 'Ln 21, Col 42', 'Spaces: 4', 'UTF-8', 'LF', 'Swift', 'Prettier', and a 'Live Share' button.

[kabiroberai/swift-sdk-darwin](https://github.com/kabiroberai/swift-sdk-darwin)

myth 8: macOS Swift apps can only be built on macOS

reality: Cross-compilation doesn't discriminate



The image shows a screenshot of Visual Studio Code with two main panes. The left pane displays the code for `MyApp.swift` in a dark-themed editor. The code uses Swift's new UI framework, SwiftUI, to create a simple application. The right pane is a terminal window showing the results of running the command `swift build --experimental-swift-sdk arm64-apple-macosx` on a Linux system. The terminal output shows the build process, which includes linking the application and creating a Mach-O executable for arm64 architecture. A large orange arrow points from the terminal output towards the title bar, indicating the flow from code to execution.

```
MyApp.swift - MyApp - Visual Studio Code

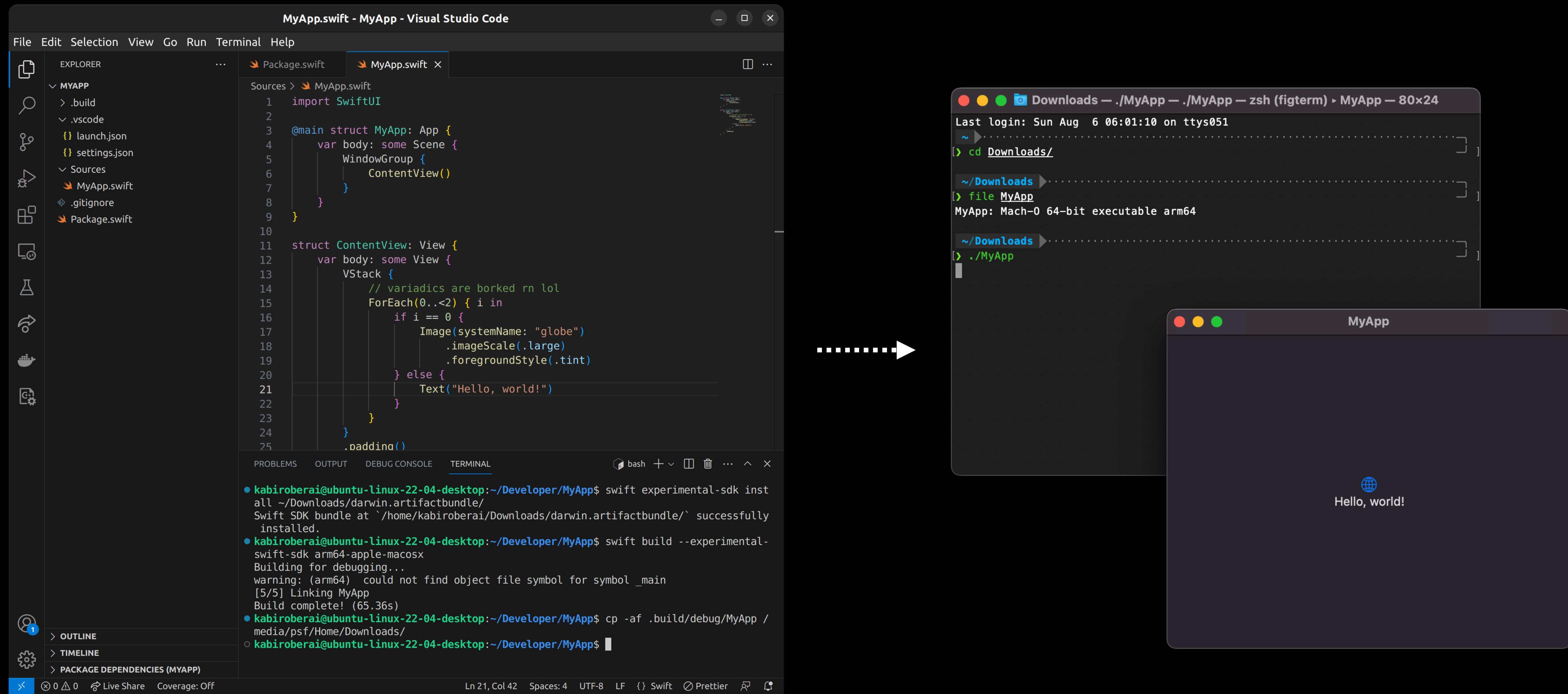
File Edit Selection View Go Run Terminal Help
EXPLORER Sources > Package.swift MyApp.swift ...
MYAPP .build .vscode launch.json settings.json Sources MyApp.swift .gitignore Package.swift
1 import SwiftUI
2
3 @main struct MyApp: App {
4     var body: some Scene {
5         WindowGroup {
6             ContentView()
7         }
8     }
9 }
10
11 struct ContentView: View {
12     var body: some View {
13         VStack {
14             // variadics are borked rn lol
15             ForEach(0..<2) { i in
16                 if i == 0 {
17                     Image(systemName: "globe")
18                         .imageScale(.large)
19                         .foregroundStyle(.tint)
20                 } else {
21                     Text("Hello, world!")
22                 }
23             }
24         }
25     }
26     .padding()
27 }

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
kabiroberai@ubuntu-linux-22-04-desktop:~/Developer/MyApp$ swift experimental-sdk install ~/Downloads/darwin.artifactbundle/
Swift SDK bundle at `/home/kabiroberai/Downloads/darwin.artifactbundle/` successfully installed.
kabiroberai@ubuntu-linux-22-04-desktop:~/Developer/MyApp$ swift build --experimental-swift-sdk arm64-apple-macosx
Building for debugging...
warning: (arm64) could not find object file symbol for symbol _main
[5/5] Linking MyApp
Build complete! (65.36s)
kabiroberai@ubuntu-linux-22-04-desktop:~/Developer/MyApp$ cp -af .build/debug/MyApp /media/psf/Home/Downloads/
kabiroberai@ubuntu-linux-22-04-desktop:~/Developer/MyApp$
```

Ln 21, Col 42 Spaces: 4 UTF-8 LF {} Swift Prettier

myth 8: macOS Swift apps can only be built on macOS

reality: Cross-compilation doesn't discriminate



The diagram illustrates the workflow for cross-compiling a macOS Swift application (MyApp) on a Linux system:

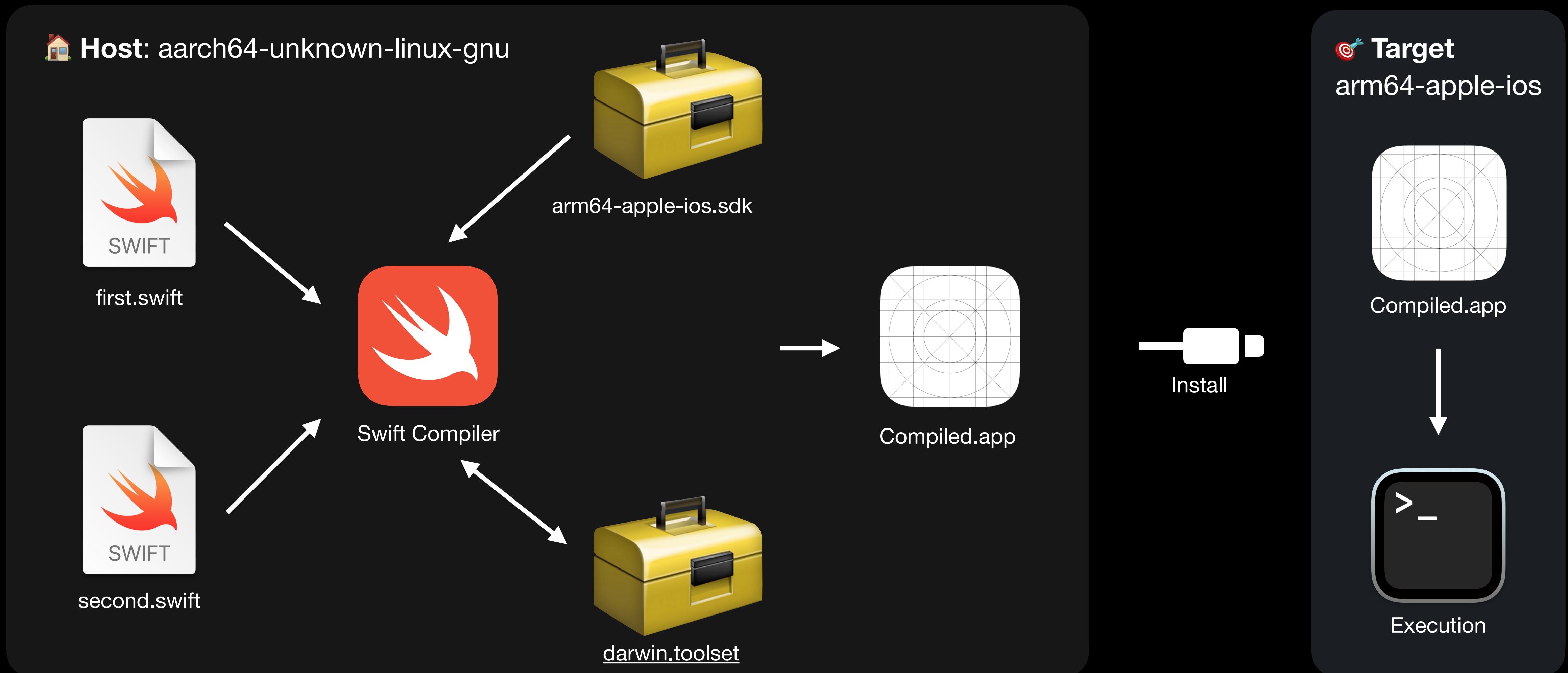
- Visual Studio Code (VS Code):** Shows the project structure and the `MyApp.swift` file content. The code defines a SwiftUI application with a globe icon and the text "Hello, world!".
- Terminal:** Shows the command-line steps to build the app:
 - `swift experimental-sdk install ~/Downloads/darwin.artifactbundle/` installs the Darwin Swift SDK.
 - `swift build --experimental-swift-sdk arm64-apple-macosx` builds the application for debugging on an Apple M1 chip.
 - `cp -af .build/debug/MyApp /media/psf/Home/Downloads/` copies the built executable to the Downloads folder.
- Resulting Application Window:** A macOS-style window titled "MyApp" displays the text "Hello, world!".

[kabiroberai/swift-sdk-darwin](https://github.com/kabiroberai/swift-sdk-darwin)

interlude

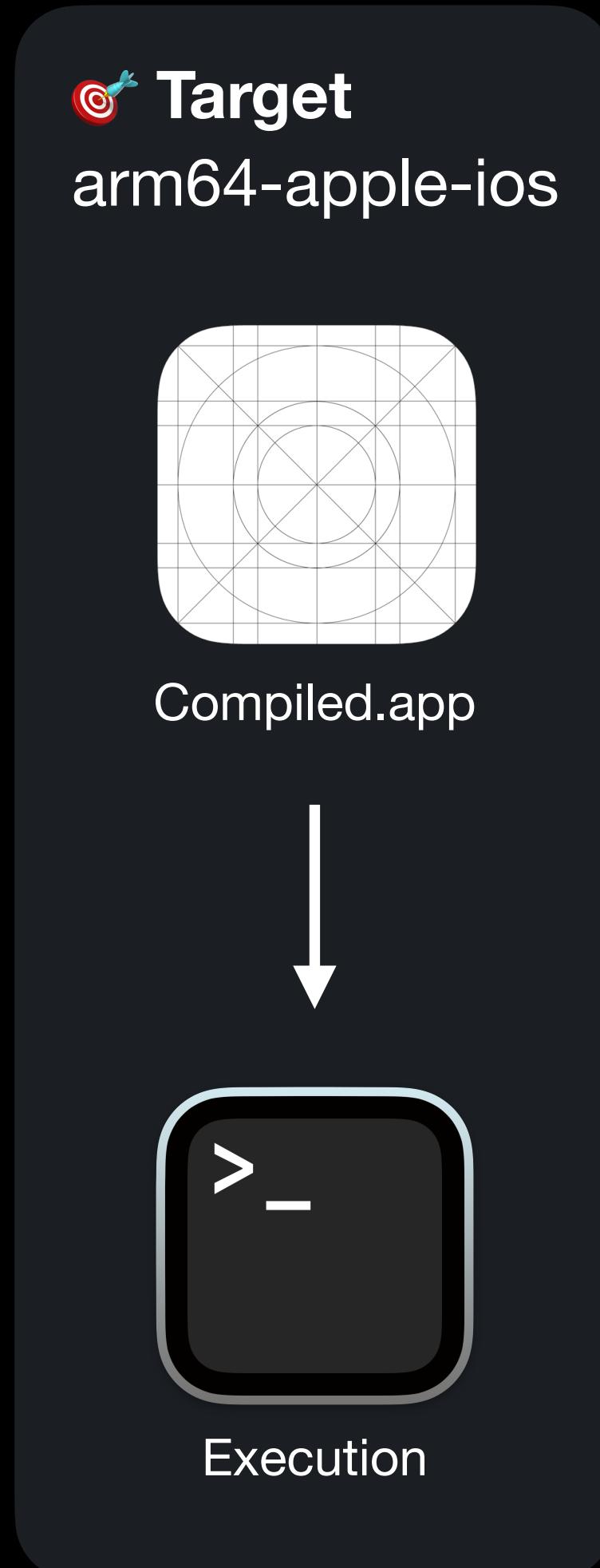
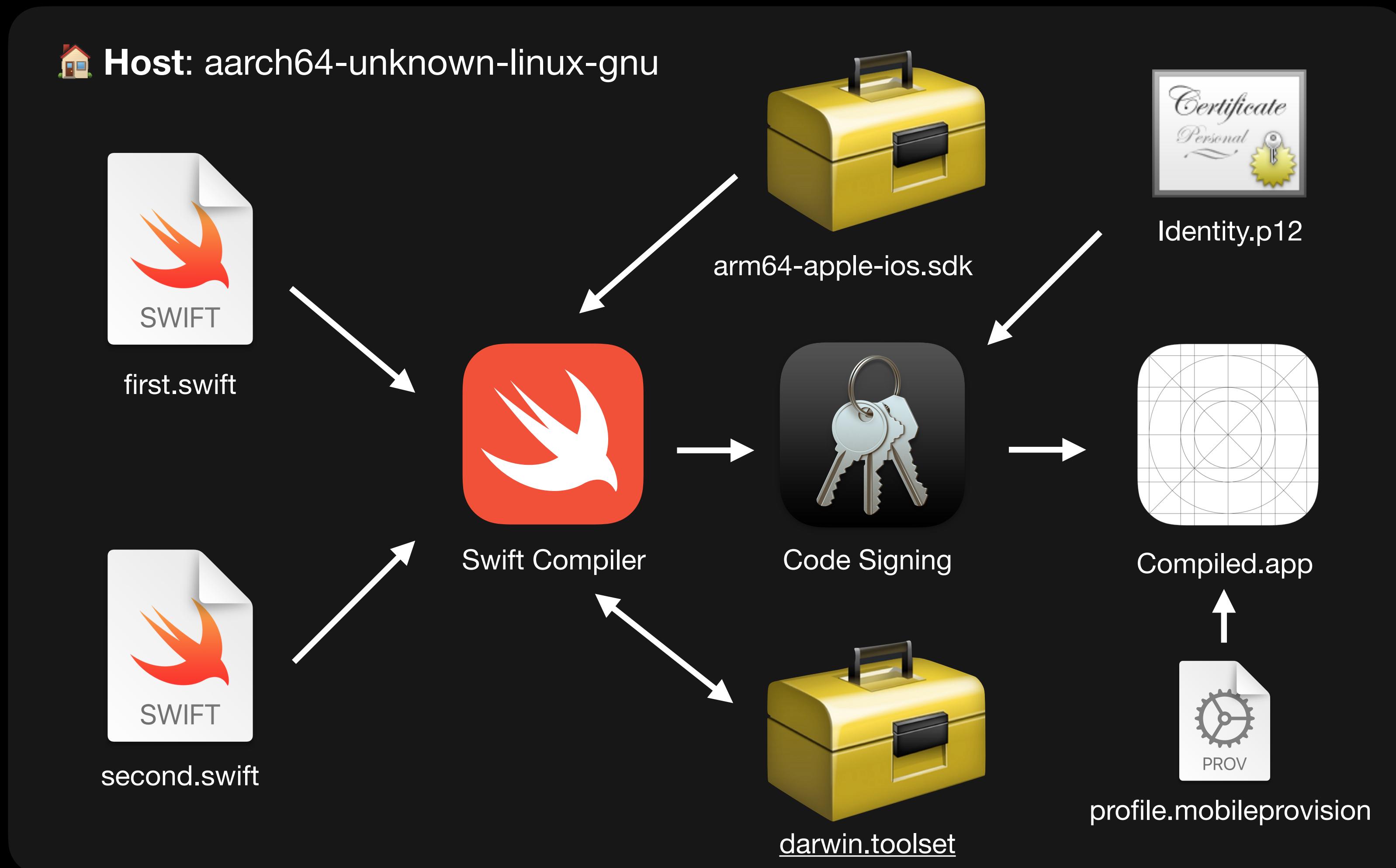
host = Linux
target = iOS?!

interlude: even more theory



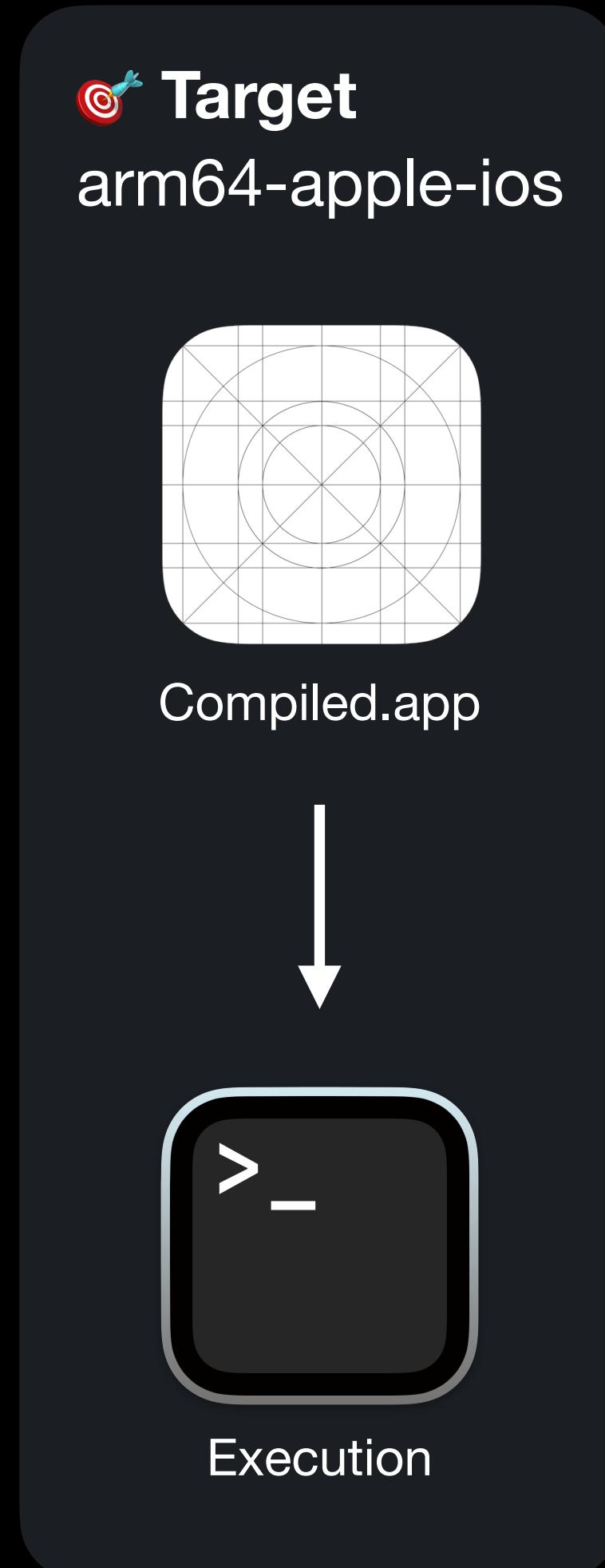
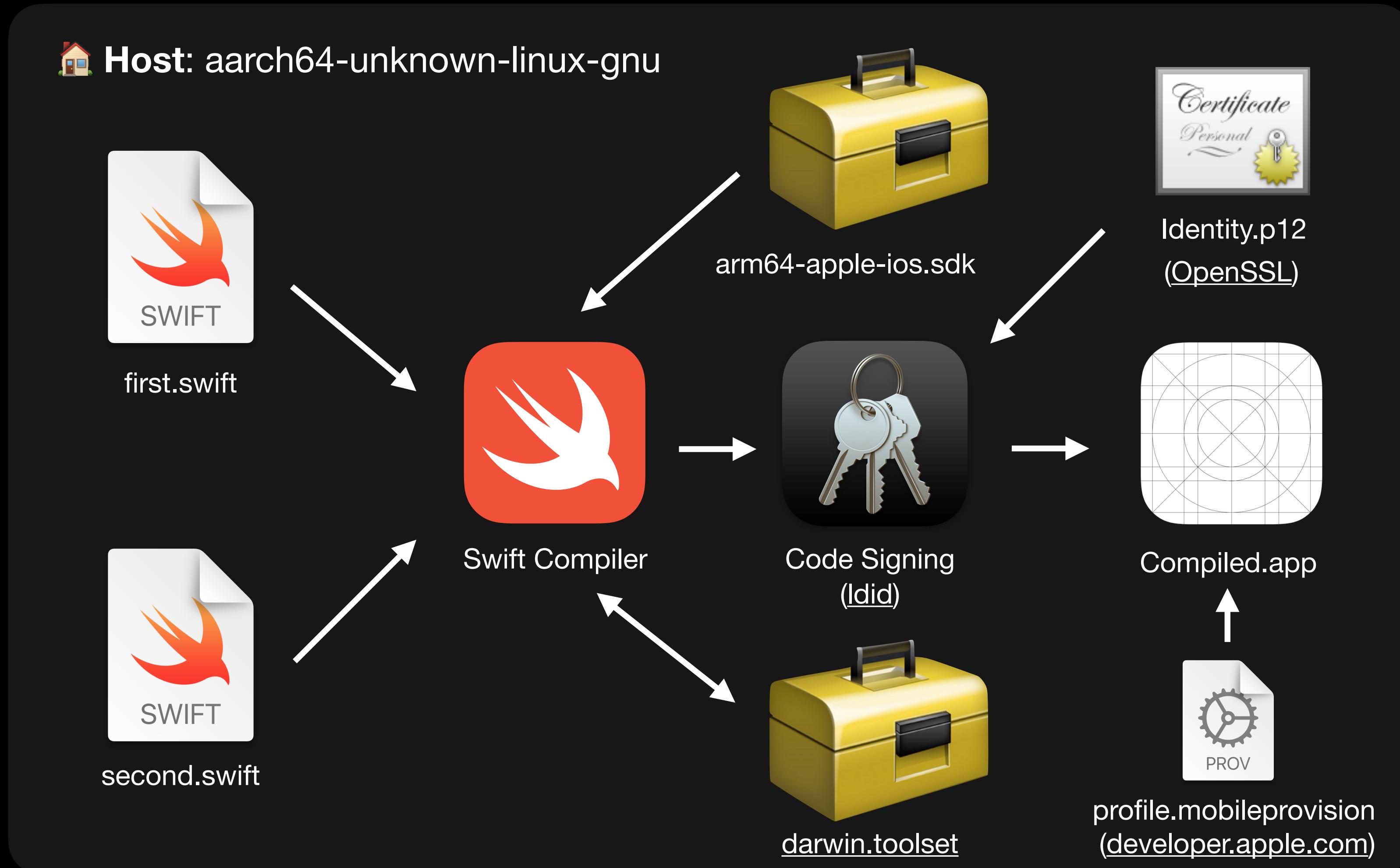
from Linux to iOS

interlude: even more theory



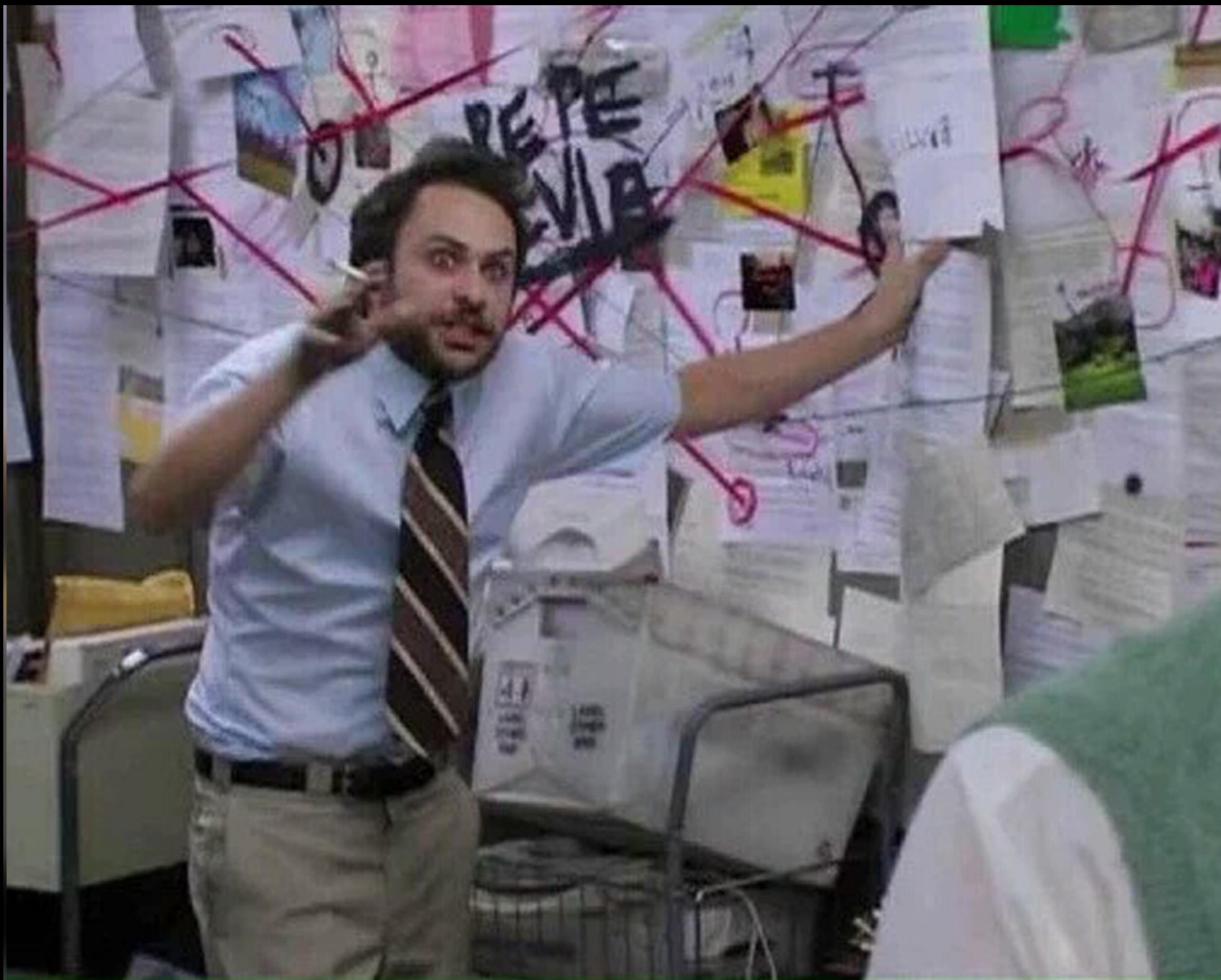
from Linux to iOS

interlude: even more theory



from Linux to iOS

and now,,



myth 9: *iOS Swift apps can only be built on macOS*

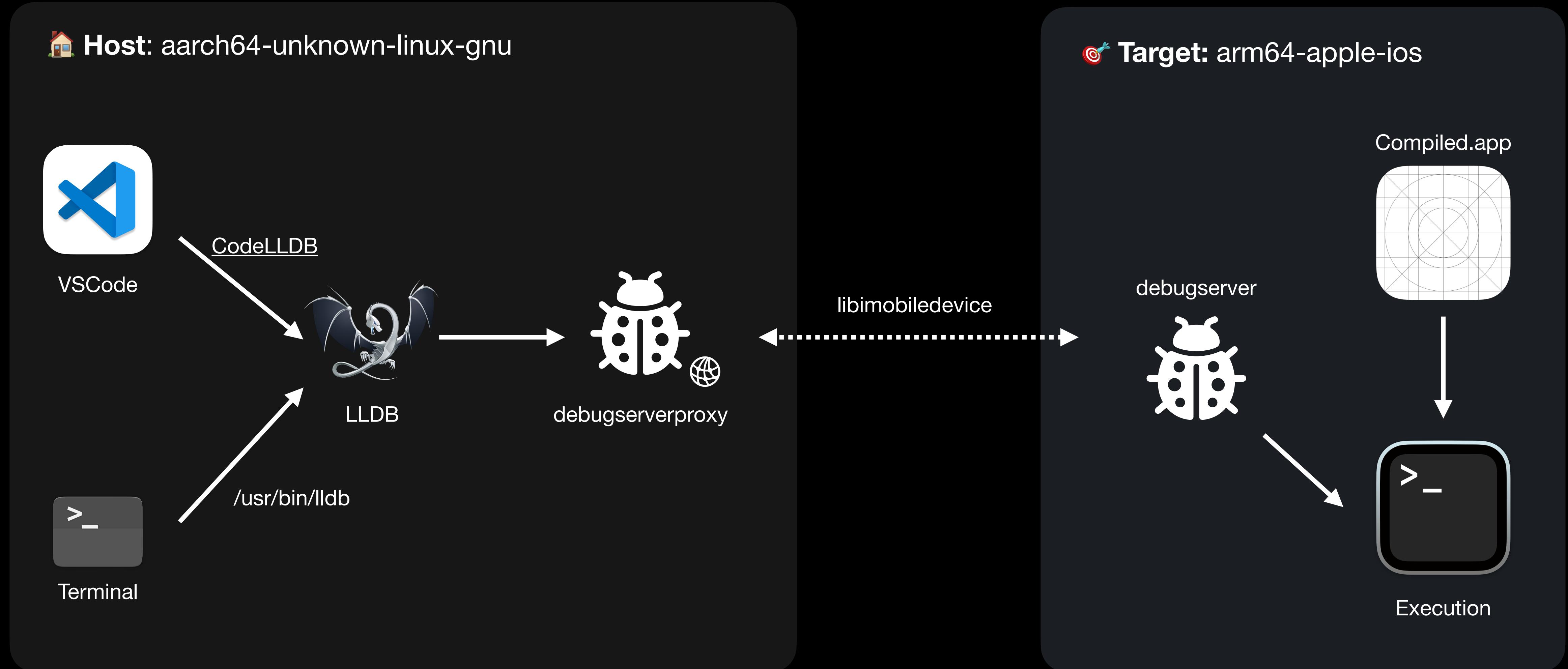
myth 9: *iOS Swift apps can only be built on macOS*

reality: Cross-compilation REALLY doesn't discriminate 😊

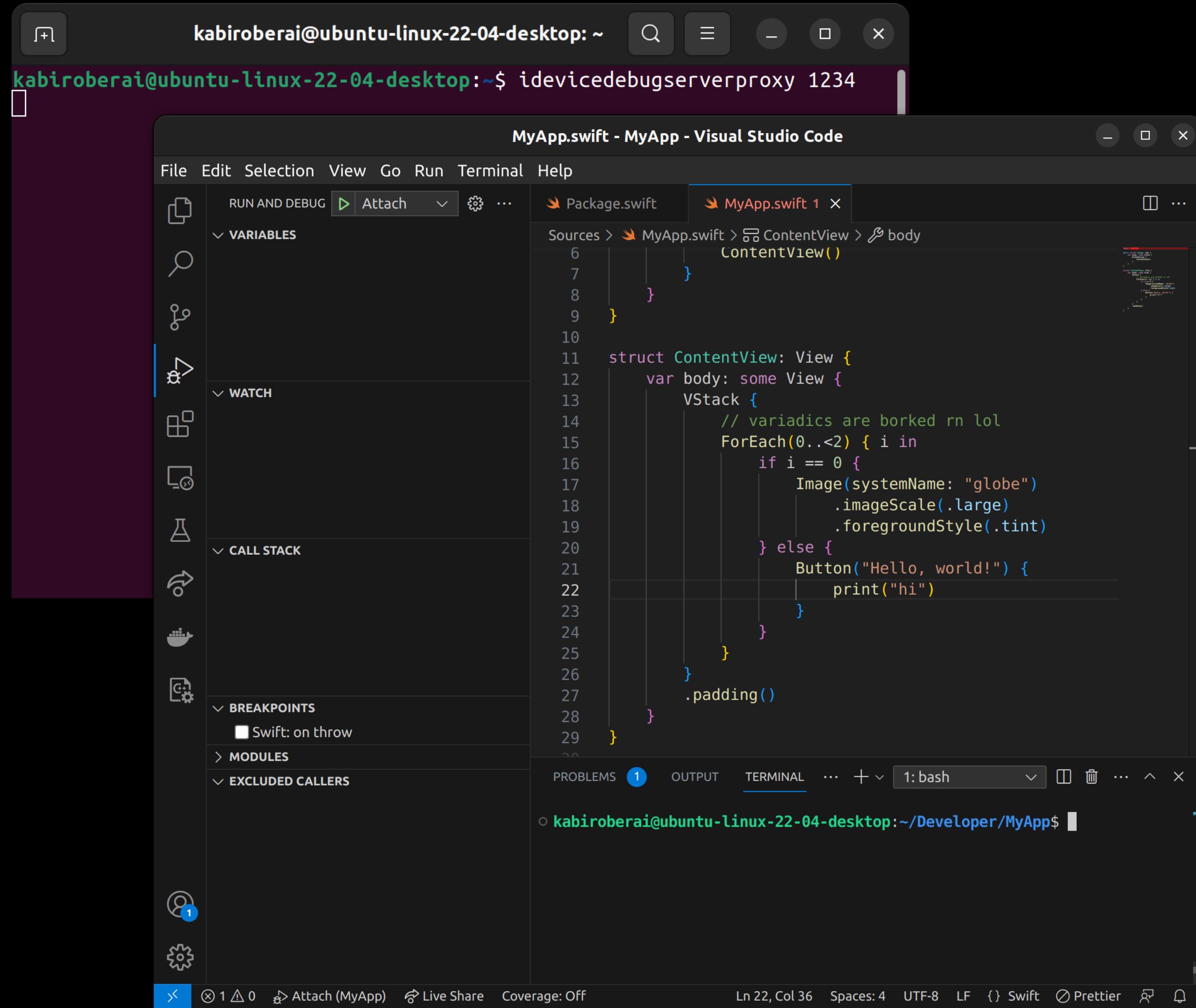
demo



bonus: remote debugging



bonus: remote debugging



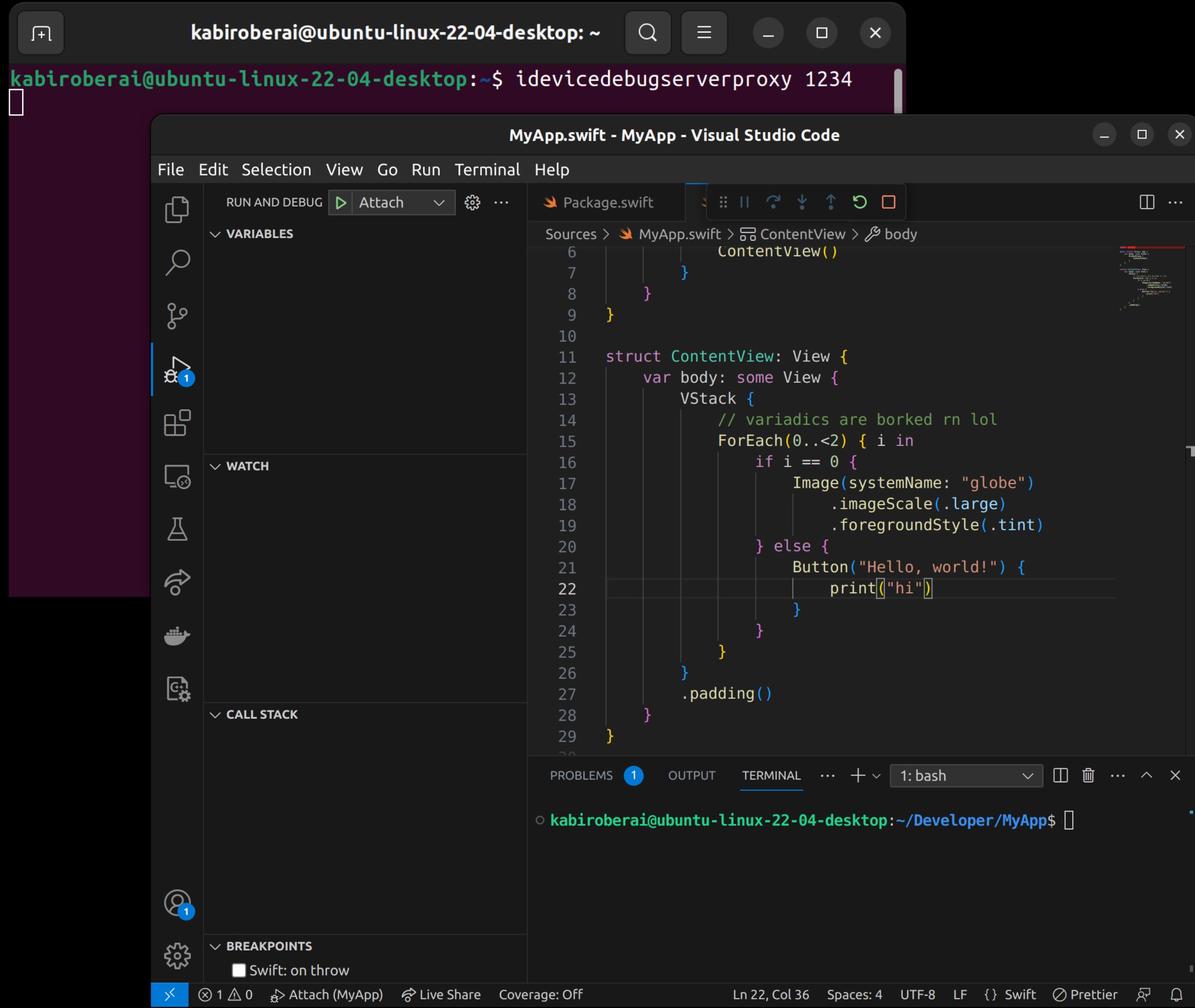
A screenshot of a Linux desktop environment showing a terminal window and a Visual Studio Code (VS Code) interface. The terminal window at the top shows the command:

```
kabiroberai@ubuntu-linux-22-04-desktop:~$ idevicedebugserverproxy 1234
```

The VS Code window below shows the code for `ContentView` in `MyApp.swift`. The code includes a conditional branch where `i == 0` leads to displaying a globe image, while `i > 0` leads to displaying a button that prints "hi". The code also includes a `.padding()` call. The left sidebar of VS Code displays the debugger interface with sections for VARIABLES, WATCH, CALL STACK, BREAKPOINTS, MODULES, and EXCLUDED CALLERS.



bonus: remote debugging



A screenshot of a Linux desktop environment showing a terminal window and a Visual Studio Code (VS Code) interface. The terminal window at the top shows the command:

```
kabiroberai@ubuntu-linux-22-04-desktop:~$ idevicedebugserverproxy 1234
```

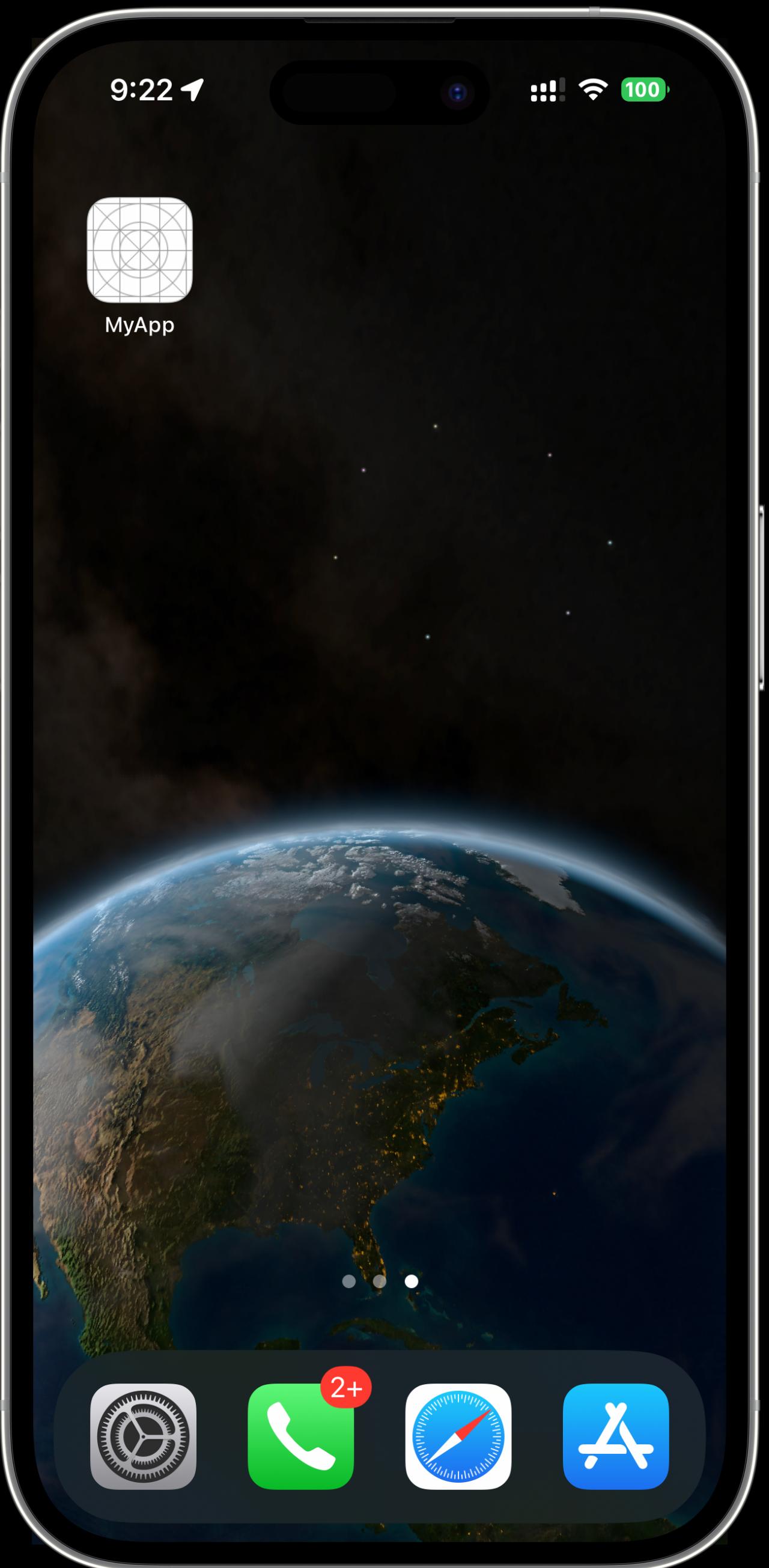
The VS Code window below displays the code for `ContentView` in `MyApp.swift`. The code includes a `ForEach` loop and a `Button` with a `print` statement. The interface shows various debugging tools like RUN AND DEBUG, VARIABLES, WATCH, CALL STACK, and BREAKPOINTS. A sidebar indicates there is 1 problem.

```
MyApp.swift - MyApp - Visual Studio Code
File Edit Selection View Go Run Terminal Help
RUN AND DEBUG Attach ... Package.swift Sources > MyApp.swift > ContentView > body
11 struct ContentView: View {
12     var body: some View {
13         VStack {
14             // variadics are borked rn lol
15             ForEach(0..<2) { i in
16                 if i == 0 {
17                     Image(systemName: "globe")
18                         .imageScale(.large)
19                         .foregroundStyle(.tint)
20                 } else {
21                     Button("Hello, world!") {
22                         print("hi")
23                     }
24                 }
25             }
26         }
27         .padding()
28     }
29 }
```

PROBLEMS 1 OUTPUT TERMINAL ... + 1: bash ... ^ x

kabiroberai@ubuntu-linux-22-04-desktop:~/Developer/MyApp\$

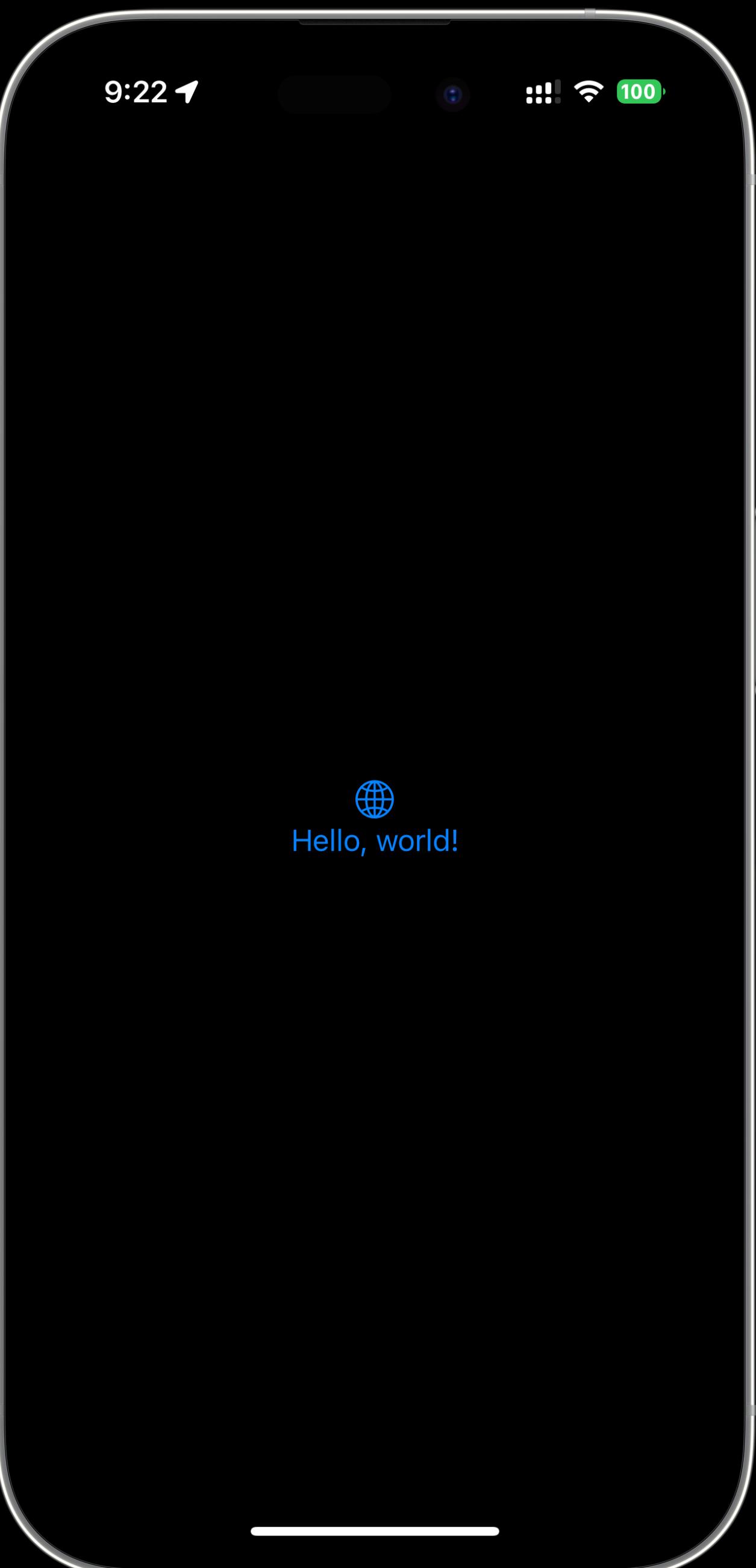
Swift: on throw



bonus: remote debugging

A screenshot of a Linux desktop environment showing a terminal window and a Visual Studio Code (VS Code) interface. The terminal window shows the command `iDeviceDebugServerProxy 1234` being run. The VS Code window displays the code for `ContentView` in `MyApp.swift`. The code includes a `ForEach` loop that prints "Hello, world!" and a `Button` with the text "Hello, world!". The VS Code interface includes a sidebar with tabs for `VARIABLES`, `WATCH`, `CALL STACK`, `BREAKPOINTS`, `MODULES`, and `EXCLUDED CALLERS`. The `DEBUG CONSOLE` tab is active, showing assembly code and a message about resuming the process.

```
dyld`<redacted>:  
-> 0x1f17041b0 <+8>: b.lo 0x1f17041d0 ; <+40>  
0x1f17041b4 <+12>: pacibsp  
0x1f17041b8 <+16>: stp x29, x30, [sp, #-0x10]!  
0x1f17041bc <+20>: mov x29, sp  
  
Process 1996 resuming
```



bonus: remote debugging

A screenshot of a Linux desktop environment showing a terminal window and a Visual Studio Code (VS Code) interface. The terminal window shows the command `iDevicesDebugServerProxy 1234` being run. The VS Code window displays the code for `ContentView` in `MyApp.swift`. A red dot on the left margin indicates a breakpoint at line 22. The code includes a `ForEach` loop and a `Button` with the text "Hello, world!". The VS Code interface also shows the DEBUG CONSOLE tab, which contains assembly-like log output. To the right of the VS Code window is a representation of an iPhone displaying the text "Hello, world!".

```
kabiroberai@ubuntu-linux-22-04-desktop:~$ iDevicesDebugServerProxy 1234
```

```
MyApp.swift - MyApp - Visual Studio Code
```

```
File Edit Selection View Go Run Terminal Help
```

```
RUN AND DEBUG Attach ...
```

```
VARIABLES
```

```
Sources > MyApp.swift > ContentView > body
```

```
ContentView()
```

```
11 struct ContentView: View {  
12     var body: some View {  
13         VStack {  
14             // variadics are borked rn lol  
15             ForEach(0..<2) { i in  
16                 if i == 0 {  
17                     Image(systemName: "globe")  
18                         .imageScale(.large)  
19                         .foregroundStyle(.tint)  
20                 } else {  
21                     Button("Hello, world!") {  
22                         print("hi")  
23                     }  
24                 }  
25             }  
26         }.padding()  
27     }  
28 }
```

```
CALL STACK
```

```
Running
```

```
BREAKPOINTS
```

- Swift: on throw
- MyApp.swift Sources (22)

```
MODULES
```

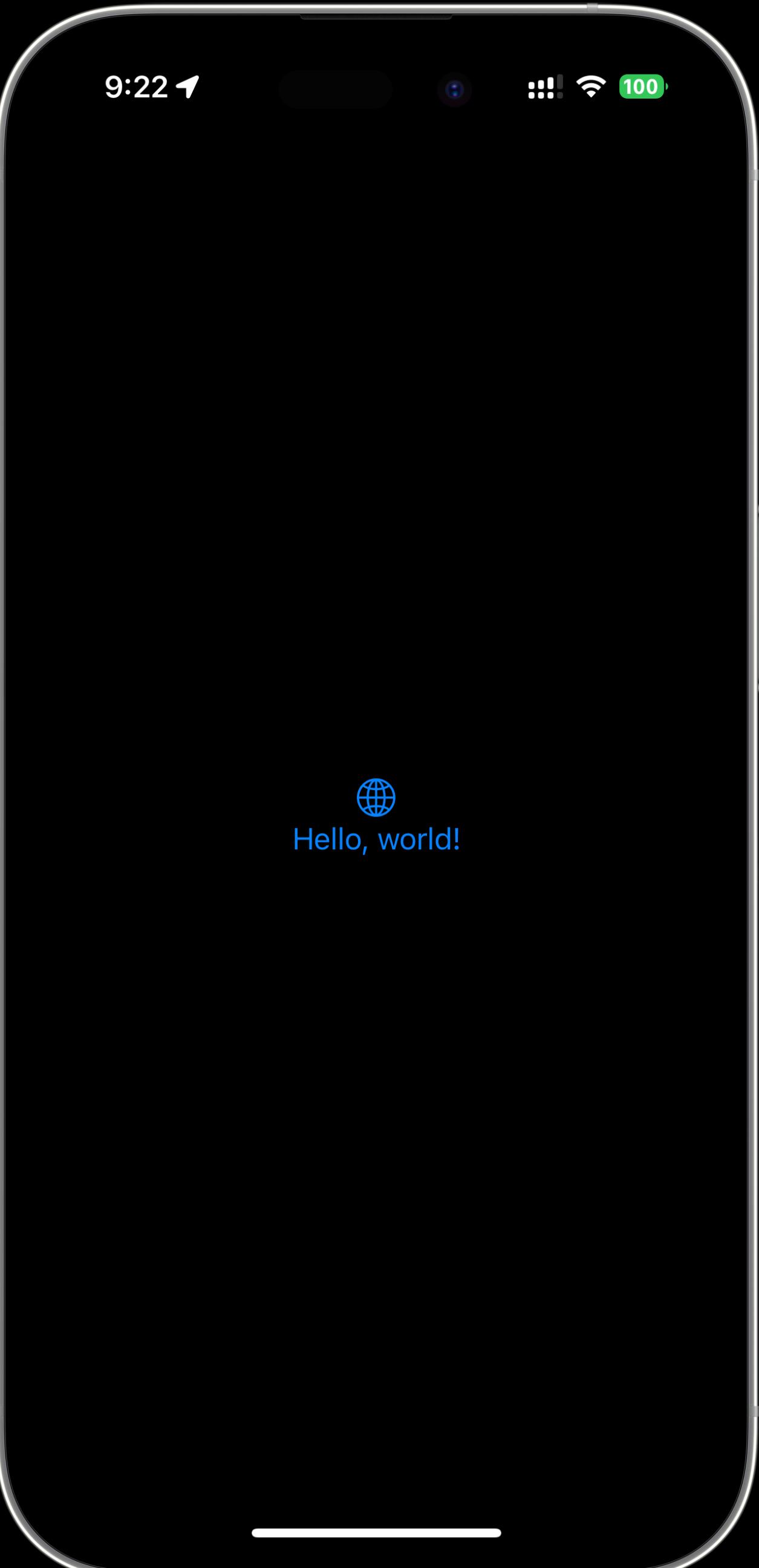
```
EXCLUDED CALLERS
```

```
PROBLEMS 1 DEBUG CONSOLE ... Filter (e.g. text, !exclude)
```

```
dyld`<redacted>:  
-> 0x1f17041b0 <+8>: b.lo 0x1f17041d0 ; <+40>  
0x1f17041b4 <+12>: pacibsp  
0x1f17041b8 <+16>: stp x29, x30, [sp, #-0x10]!  
0x1f17041bc <+20>: mov x29, sp
```

```
Process 1996 resuming
```

```
Attach (MyApp) Live Share Format: auto Disasm: auto Deref: on Console: cmd Coverage: Off LF {} Swift Prettier ⚡
```

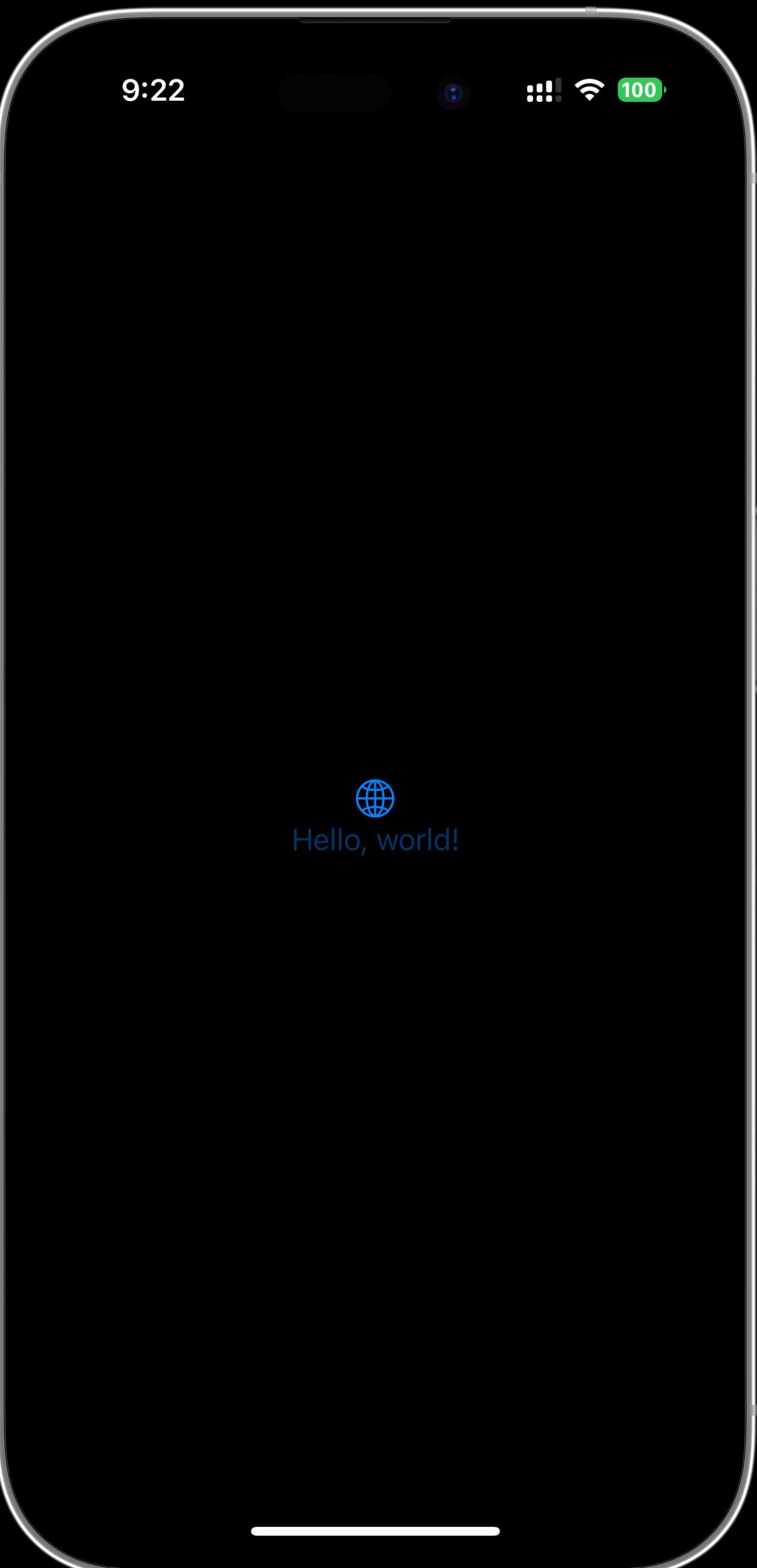


bonus: remote debugging

A screenshot of a Linux desktop environment showing a terminal window and a Visual Studio Code (VS Code) interface. The terminal window shows the command `ideviceDebuggerProxy 1234` being run. The VS Code window displays the code for `ContentView` in `MyApp.swift`. A breakpoint is set on line 22, and the code is paused at this point. The call stack shows the current thread is at `$s5MyApp11ContentViewV4bodyQrvg7SwiftUI`. The left sidebar of VS Code shows the debugger's state, including variables, watch list, and breakpoints. The bottom of the screen shows the macOS dock with various application icons.

```
MyApp.swift - MyApp - Visual Studio Code

File Edit Selection View Go Run Terminal Help
RUN AND DEBUG Attach ... Sources > MyApp.swift > ContentView > body
VARIABLES Local Static Global Registers
WATCH
CALL STACK
PAUSED ON BREAKPOINT
1: tid=91634
$ s5MyApp11ContentViewV4bodyQrvg7SwiftUI
__lldb_unnamed_symbol98318 @ ...
__lldb_unnamed_symbol98326 @ ...
__lldb_unnamed_symbol98325 @ ...
__lldb_unnamed_symbol98322 @ ...
BREAKPOINTS
Swift: on throw
MyApp.swift Sources (22)
MODULES
EXCLUDED CALLERS
PROBLEMS 1 DEBUG CONSOLE ...
dyld`<redacted>:
-> 0x1f041b0 <+8>: b.lo 0x1f17041d0 ; <+40>
0x1f17041b4 <+12>: pacibsp
0x1f17041b8 <+16>: stp x29, x30, [sp, #-0x10]!
0x1f17041bc <+20>: mov x29, sp
Process 1996 resuming
```



conclusion

three simple steps to world domination



solve technical problems



prioritize



build

conclusion

three simple steps to world domination

⚙️ solve technical problems

🥇 prioritize

🎨 build

mostly done:
swift is versatile!

conclusion

three simple steps to world domination

⚙️ solve technical problems

mostly done:
swift is versatile!

🥇 prioritize

🎨 build

your turn 🚀

get in touch!

 [@kabiroberai](https://twitter.com/kabiroberai)

 [/in/kabiroberai](https://www.linkedin.com/in/kabiroberai)

 me@kabiroberai.com



slides & code
ober.ai/swiftto