
0x41con presenters
Plugging random cables into their computer
FBI Denver
@FBIDenver
Avoid using free charging stations in
hotels or shopping centers. Bad acto
figured out ways to use public USB p
introduce malware and monitoring s
onto devices. Carry your own charge
USB cord and use an electrical outle

All work and no play makes Jack a dull boy

0x41con 2023

Saagar Jha


Saagar Jha

aka “saagarjha”

- Software engineer at Google
  - No I am not hiring
  - I don’t get paid enough to speak for my employer
  - Want to learn more about ppl
  - Please send 0-days to:
    - saagar@saagarjha.com
    - @saagar@saagarjha.com
    - twitter.com/_saagarjha


Swift

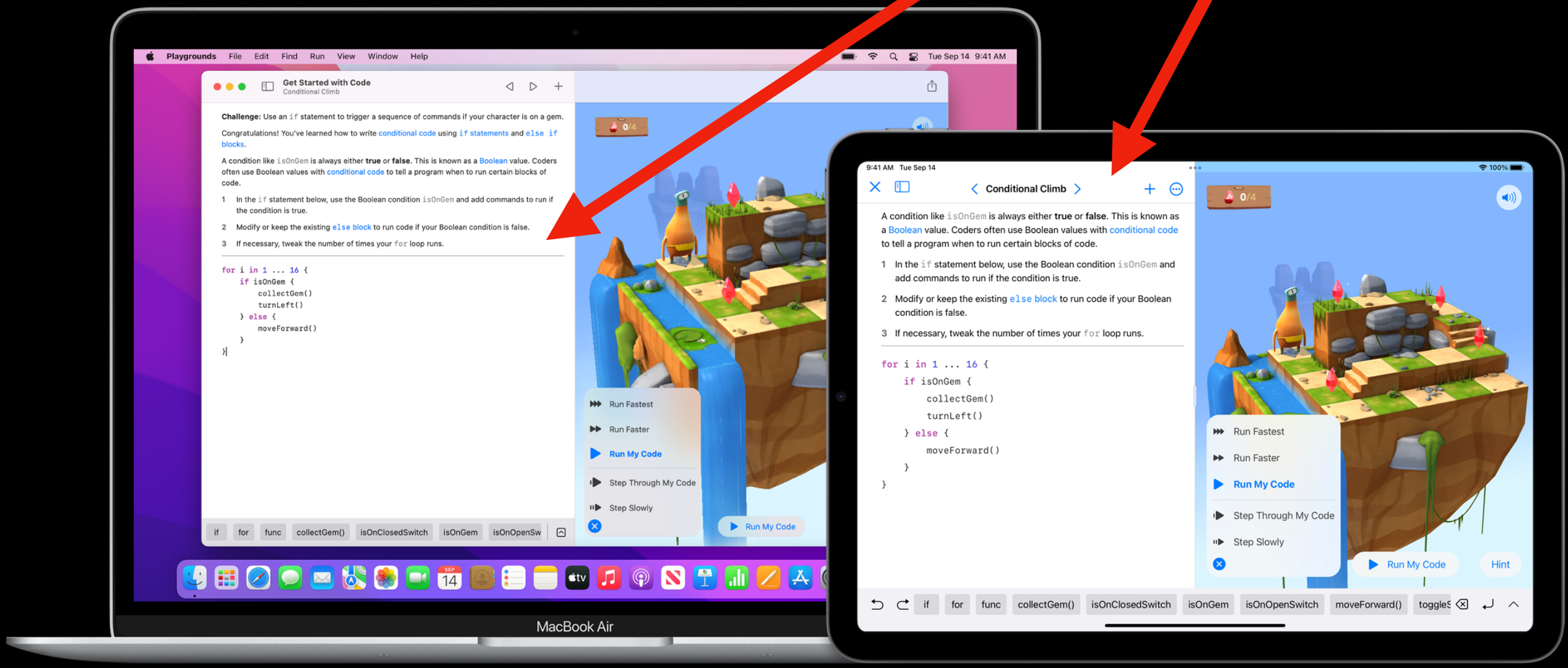
Swift

Swift Playgrounds

Swift Playgrounds

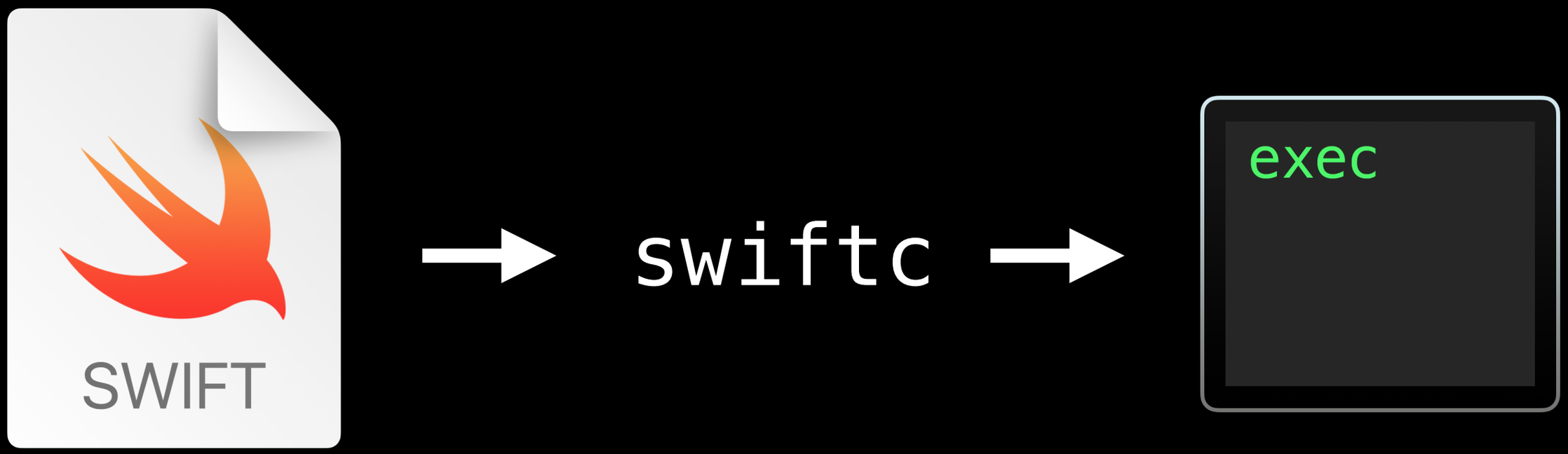
Swift Playgrounds

Real Swift code!



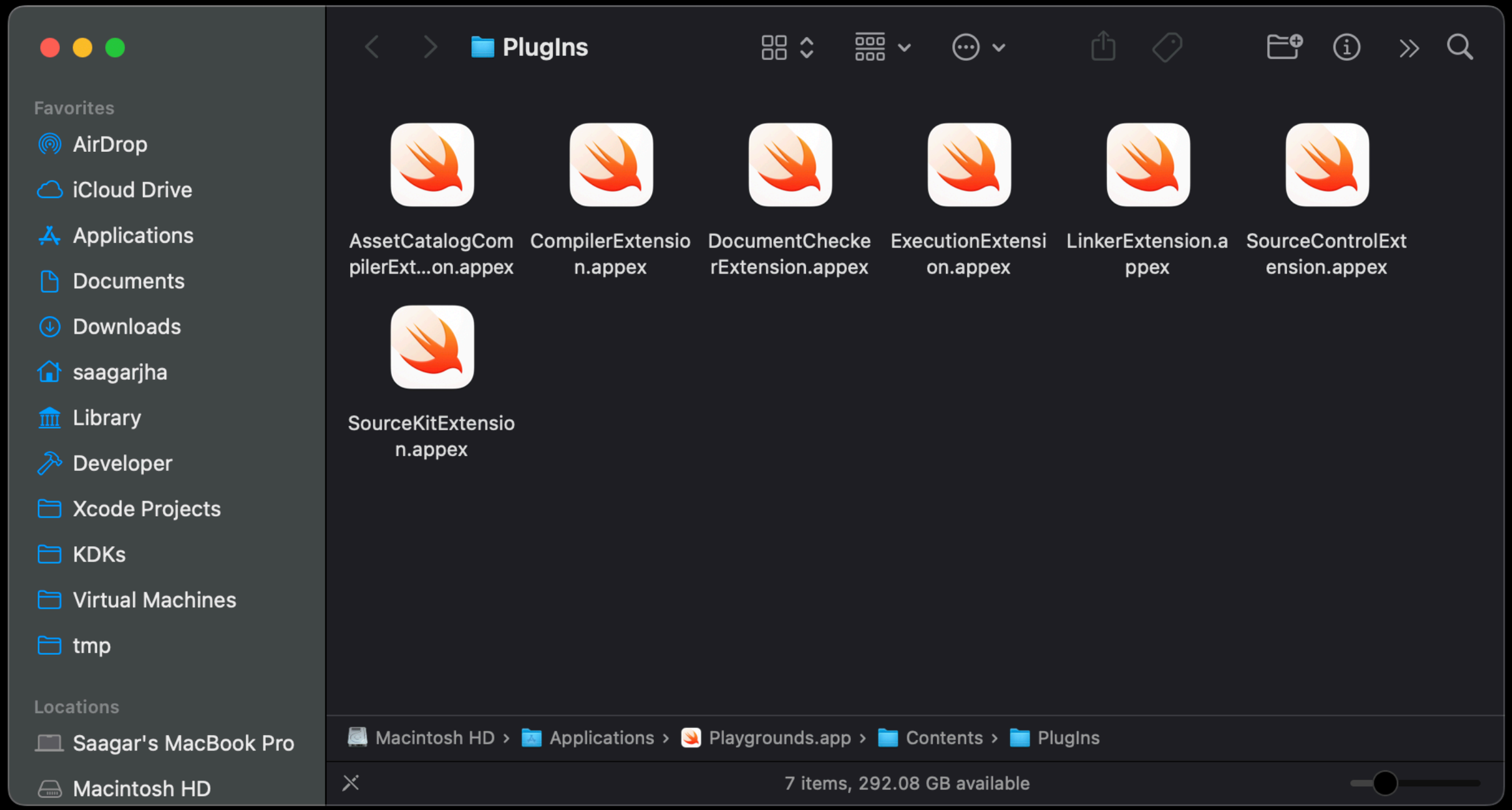
How does this work?

How does this work?

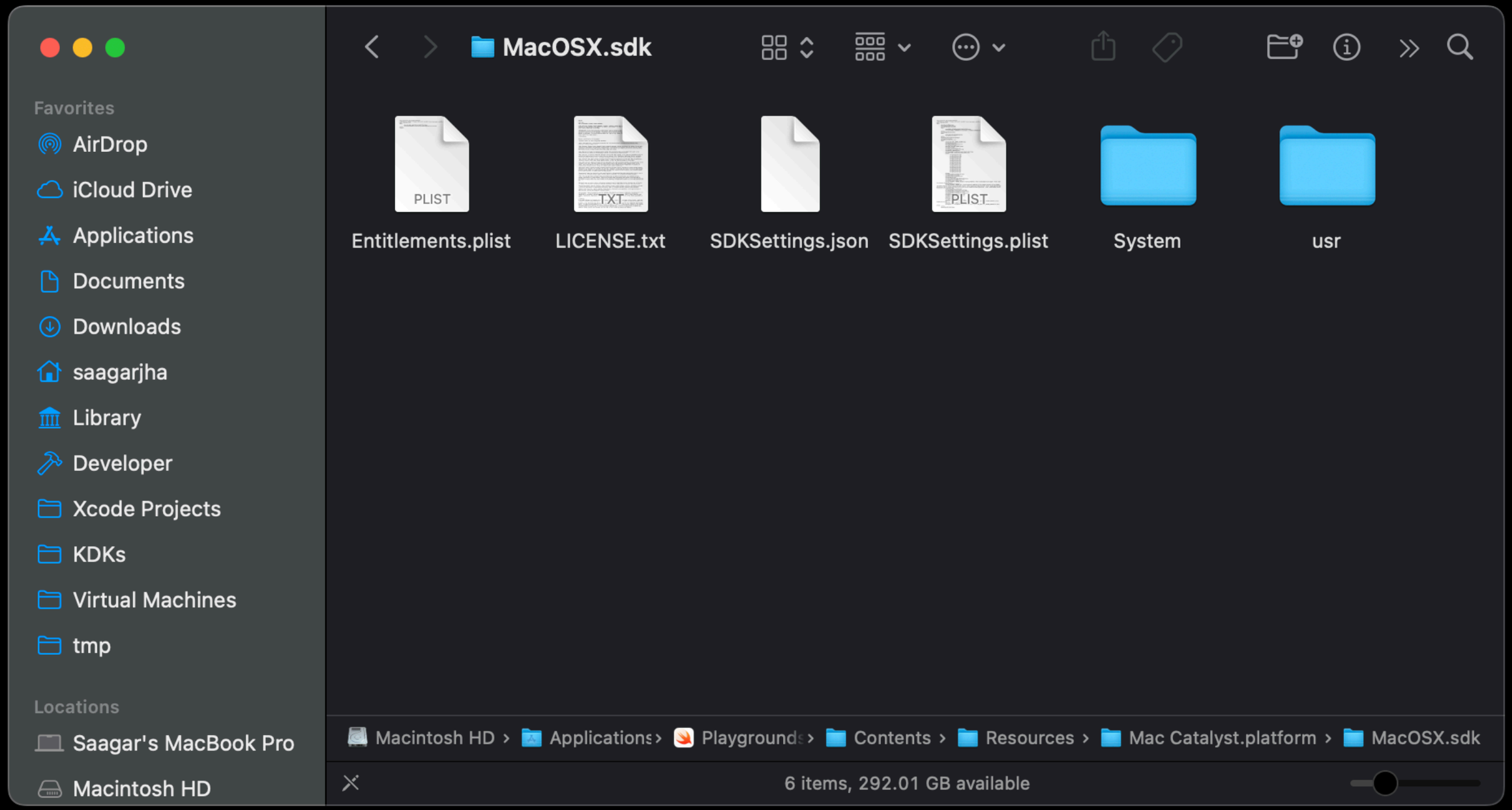


Typical Swift compilation process

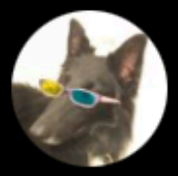
How does this work?



How does this work?



How does this work?



comex
@comex

So has anyone tried messing with the new Swift Playgrounds iPad app? How is the code signing handled? How's the sandbox?

12:36 AM · Jul 6, 2016

How does this work?

@msolnik studentd has get-task-allow

5:15 AM · Jul 6, 2016

Jailed Just-in-Time Compilation on iOS

Sunday, February 23, 2020

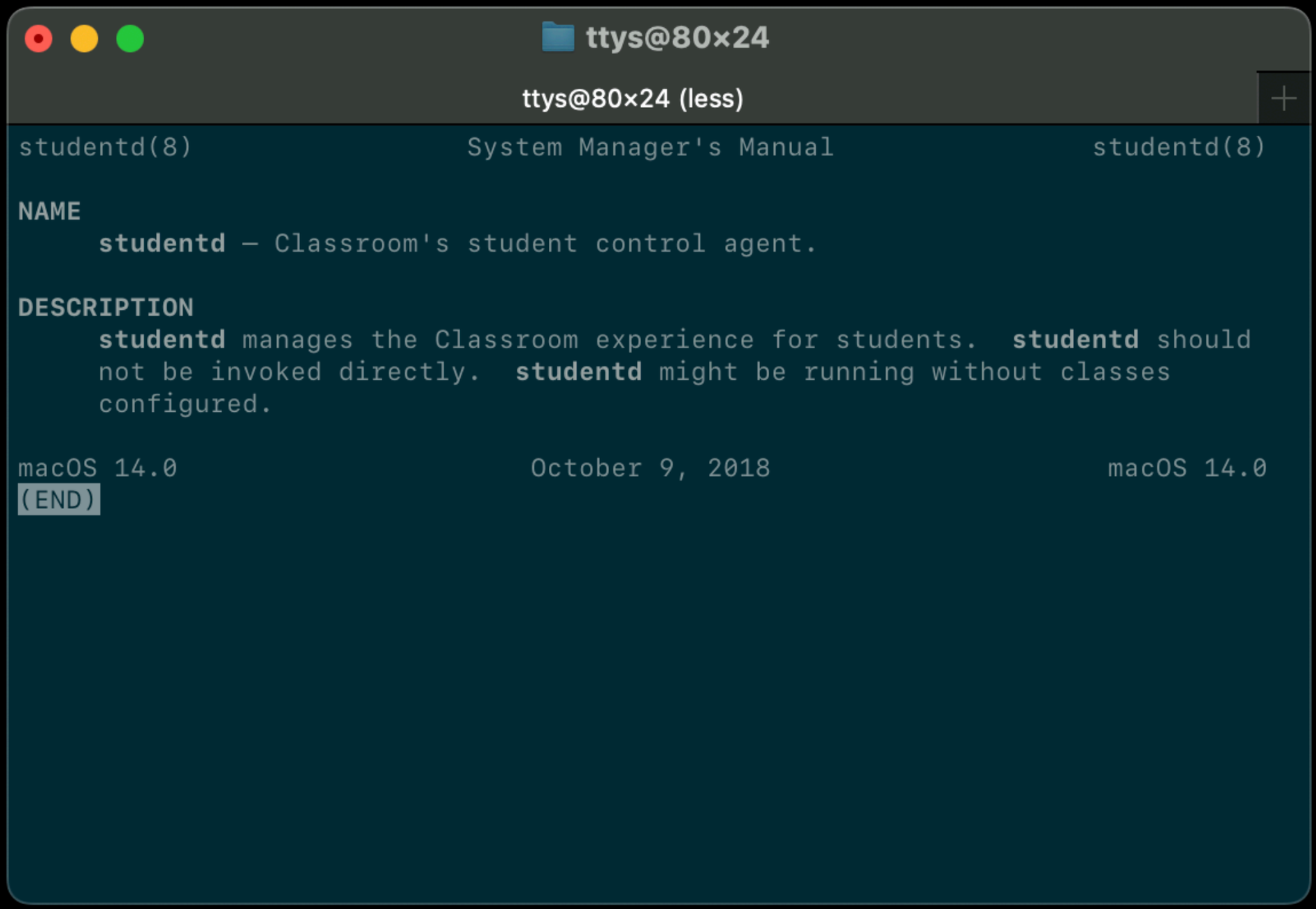
Just-in-time compilation on iOS normally requires applications to possess the dynamic-codesigning entitle
privilege that Apple uniquely awards to system processes that require the high-performance tiers of JavaS
“True” just-in-time compilers require the ability to generate executable pages with an invalid code signatur
practice that is usually prohibited on iOS for third-party apps because it sidesteps code validation guarant
Apple would like to enforce. While these applications cannot use `mmap`’s `MAP_JIT` without this entitleme
usual way to create a RWX region for JIT purposes), there is a method that does work on devices without a
though its combination of being unfit for the App Store and really only being useful for speeding up virtual
makes it seemingly unknown outside of the emulation community. The technique relies on a somewhat arc
effect of how debugging works on iOS to enable a slightly more limited JIT.

> **Update, 6/24/20**
>
> Preliminary testing on iOS 14 seems to indicate that Apple has changed the kernel so that this trick no l
> works.

Introducing the W^X JIT

The simplest way to implement a JIT is to create pages that have both `PROT_WRITE` and `PROT_EXEC` (a
`PROT_READ` –this isn’t the bulletproof JIT) enabled simultaneously, writing code into this region, and exec
this code is generated on the fly, it lack a code signature to back it, and `mmap` (or its Mach VM equivalents

How does this work?



How does this work?

Old (pre 4.0) workflow

- Swift Playgrounds can only run playgrounds
- Code is compiled to dylib by CompilerExtension/LinkerExtension
- Signature is registered with AMFI
  - (via com.apple.private.amfi.can-load-cdhash entitlement)
- ExecutionExtension loads library and runs it

How does this work?

New (4.0+) workflow

- Swift Playgrounds can run playgrounds and SPM-based apps
- New entitlements!
  - Notably, com.apple.private.playgrounds-local-signing-allowed
- Code is ad-hoc signed on-device

Arbitrary Code Execution

Arbitrary Code Execution

Goals

- Code execution of a language other than Swift
- No interpreters (native, compiled code)
- Access to full iOS SDK
- Non-goals
  - Escaping the app sandbox
  - Memory corruption in Swift Playgrounds app

Arbitrary Code Execution

Constraints, pre-4.0

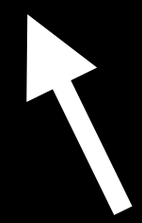
- No support for anything but playgrounds
- No control over compilation process
  - (Happens to be ~default flags, no optimizations, instrumented)
  - Support for limited range of Swift versions
- Code runs as dylib loaded in ExecutionExtension

Arbitrary Code Execution

Constraints, post-4.0

- Similar environment available to playgrounds
- Swift Package Manager-based app targets also supported
- Configuration is done via Package.swift

```
// WARNING:
// This file is automatically generated.
// Do not edit it by hand because the contents will be replaced.
```



lol

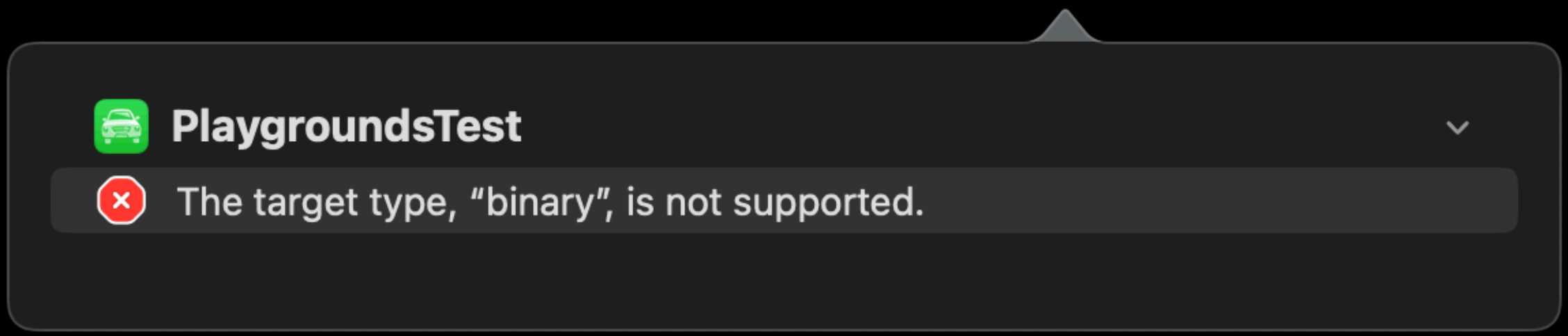
Arbitrary Code Execution

Constraints, post-4.0

- Similar environment available to playgrounds
- Swift Package Manager-based app targets also supported
  - Package.swift can be edited and is fully controlled
    - Can define targets
    - Can set compiler, linker flags
  - Apps launch standalone, in their own process
    - Some instrumentation is still present

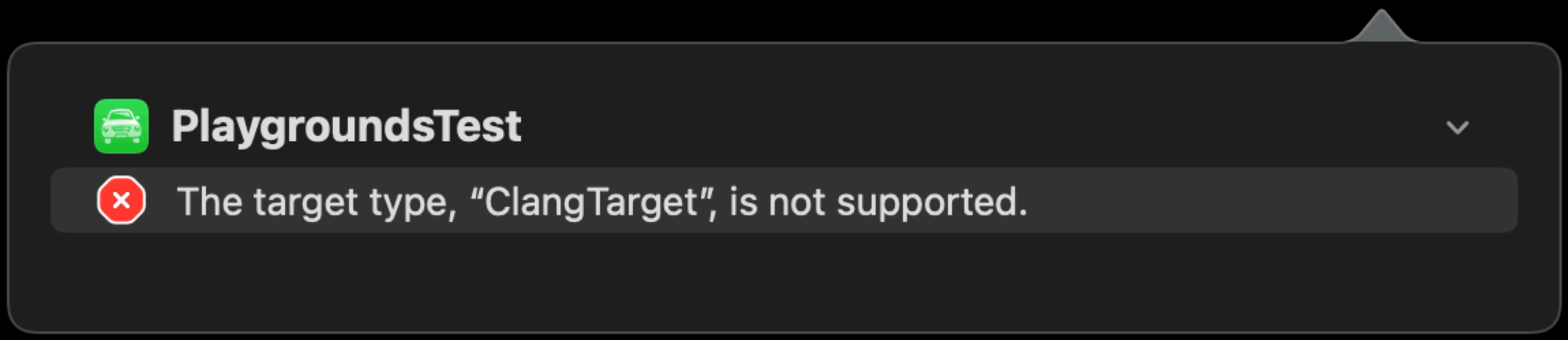
Arbitrary Code Execution

Constraints, post-4.0



Arbitrary Code Execution

Constraints, post-4.0



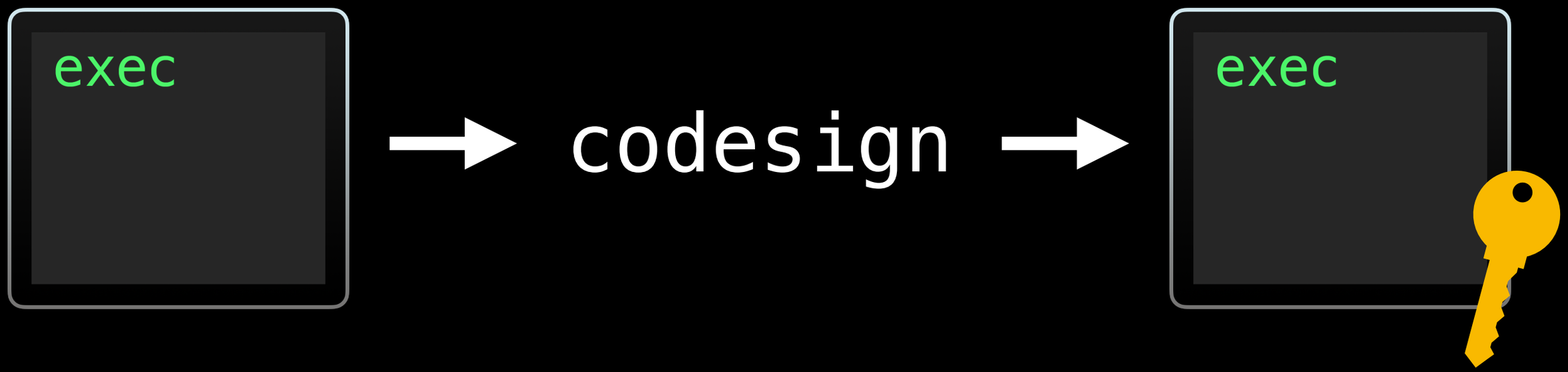
Arbitrary Code Execution

Strategy

- Obvious ways for bringing in non-Swift code will not work
  - (Resources are not signed and easy to copy over)
- Apple will sign and run anything that ends up in the final binary
- Precompile our code and trick Playgrounds into embedding it?
  - Everyone loves a good compiler jail

Arbitrary Code Execution

Embedding Strategy

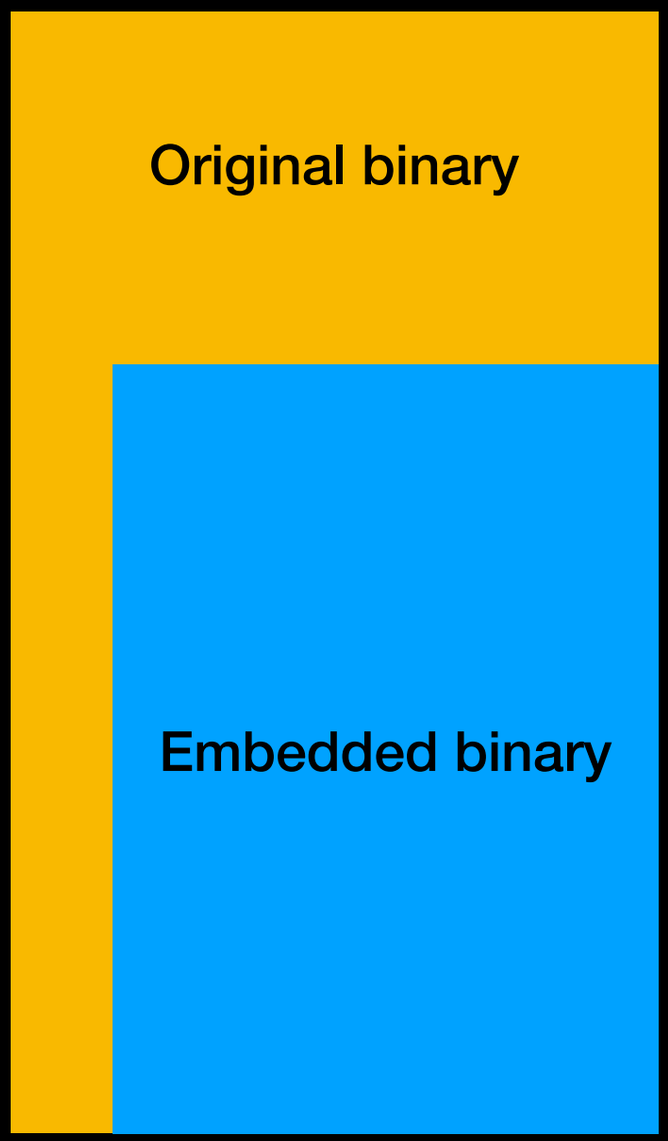


Arbitrary Code Execution

Embedding Strategy

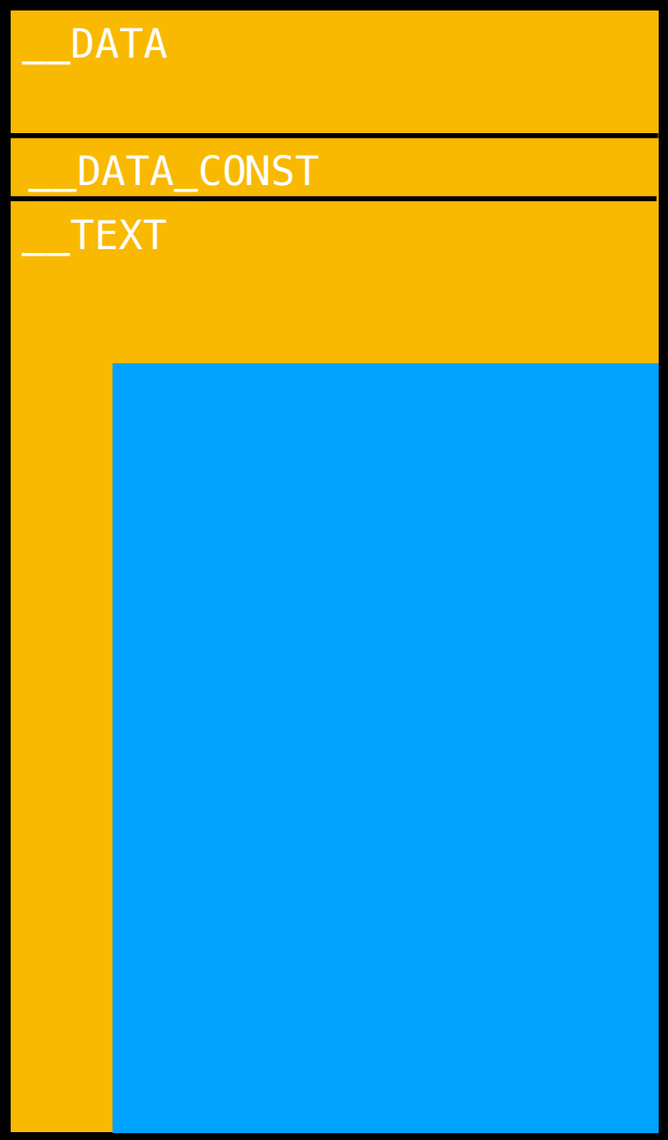
Arbitrary Code Execution

Embedding Strategy



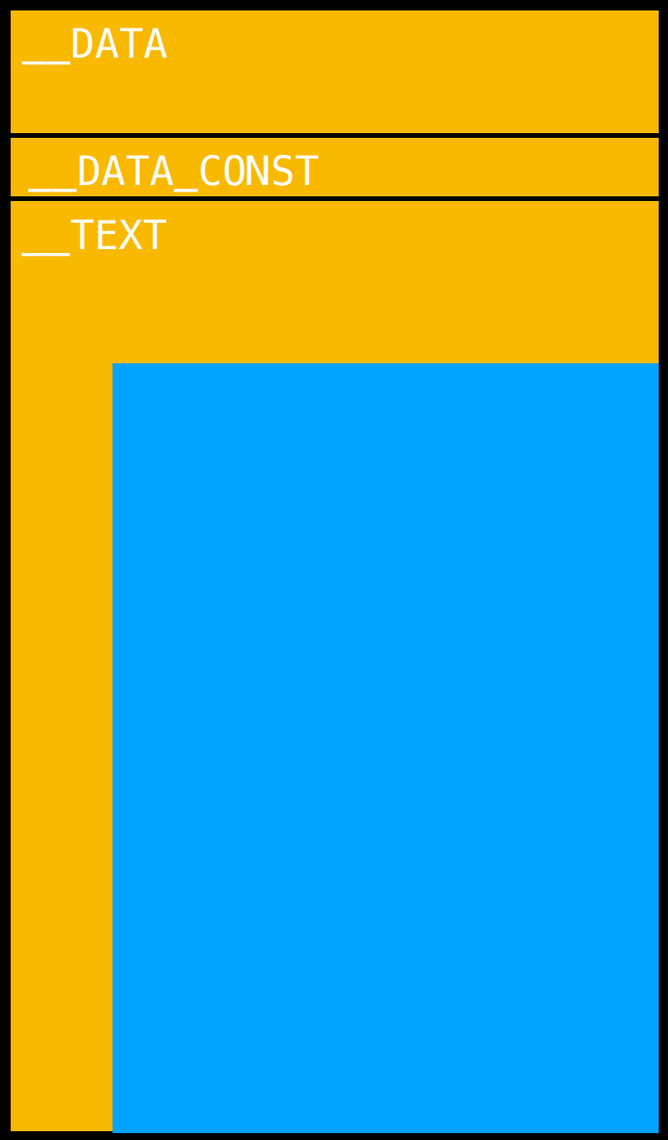
Arbitrary Code Execution

Embedding Strategy



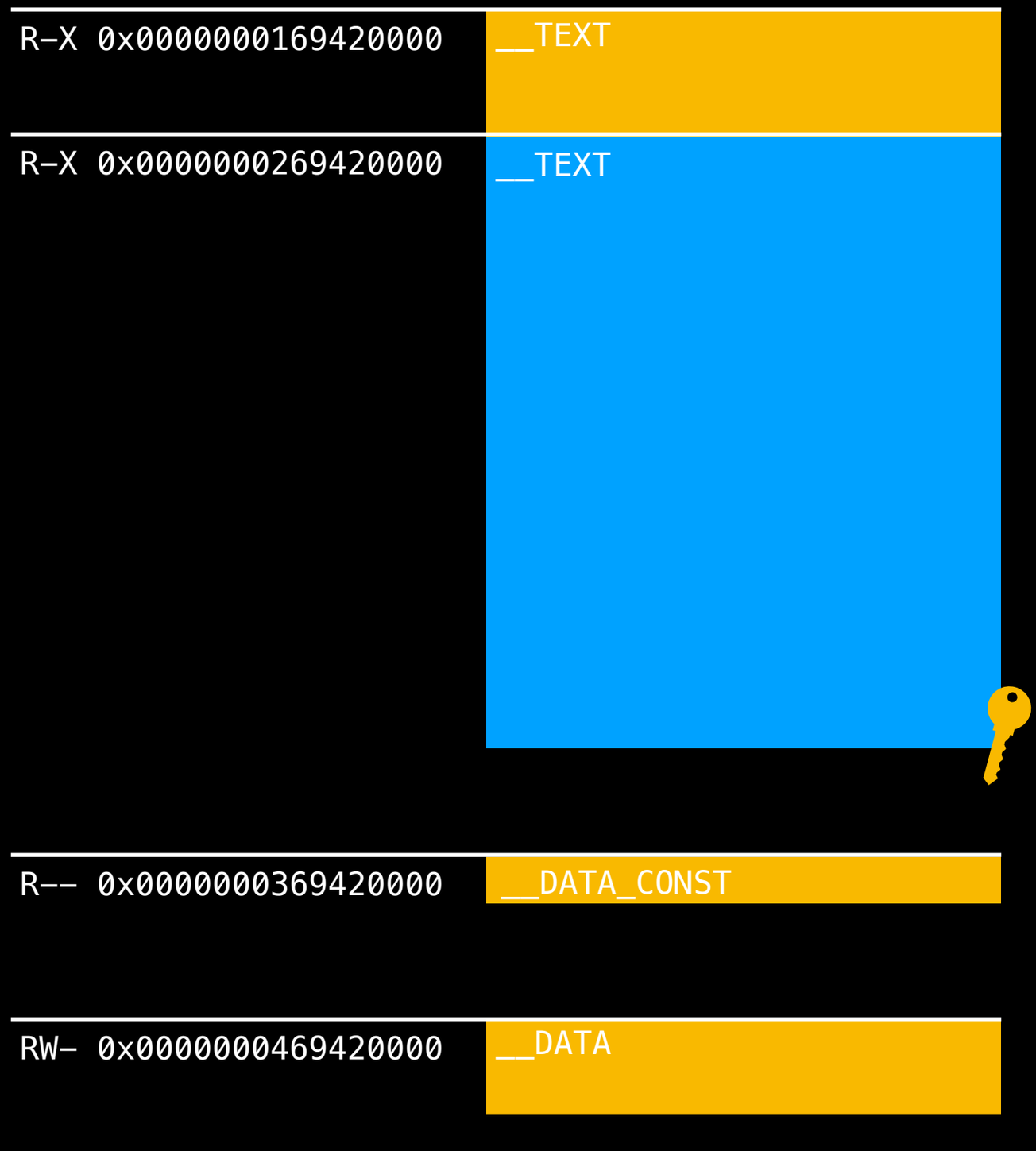
Arbitrary Code Execution

Embedding Strategy



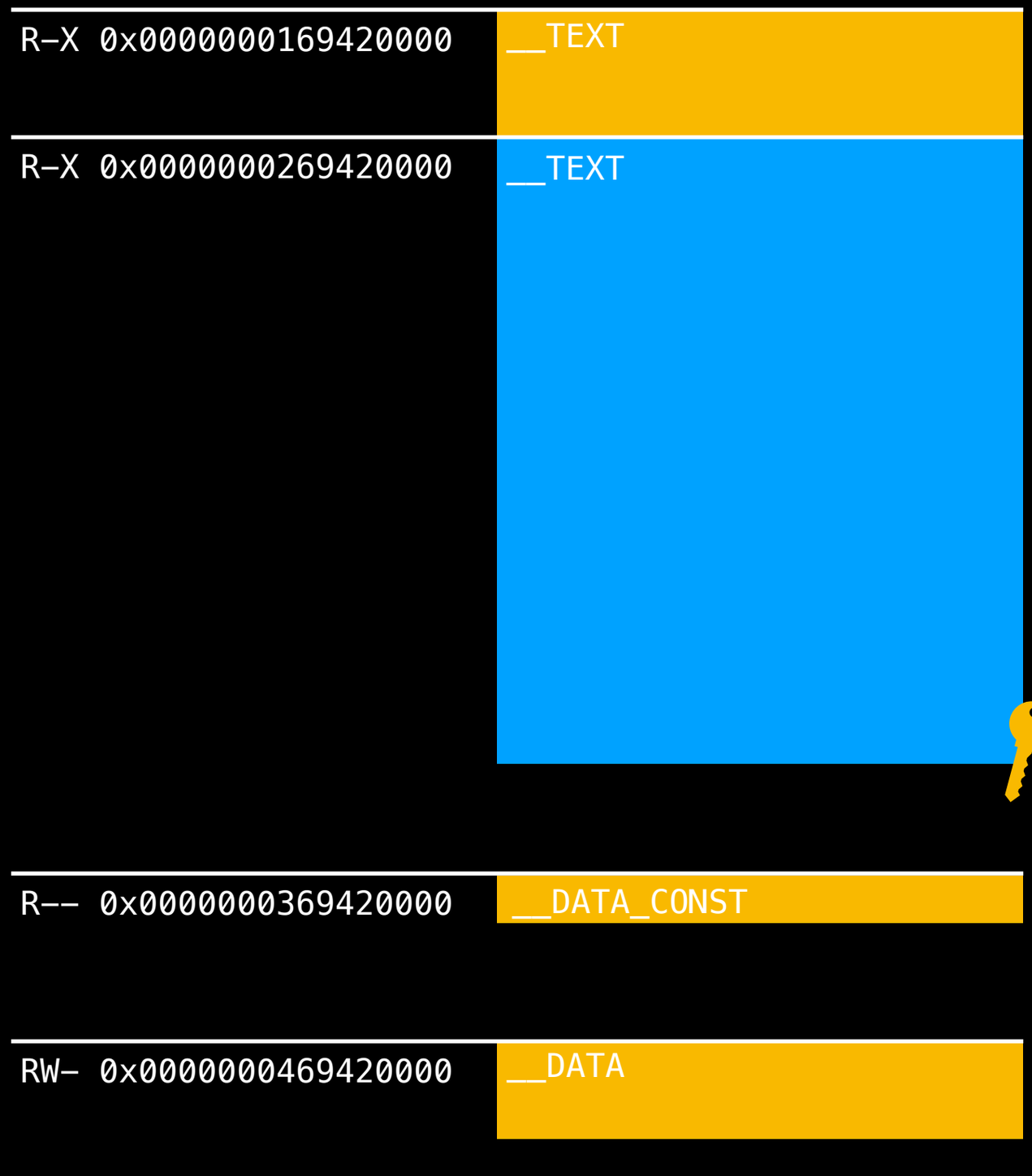
Arbitrary Code Execution

Embedding Strategy



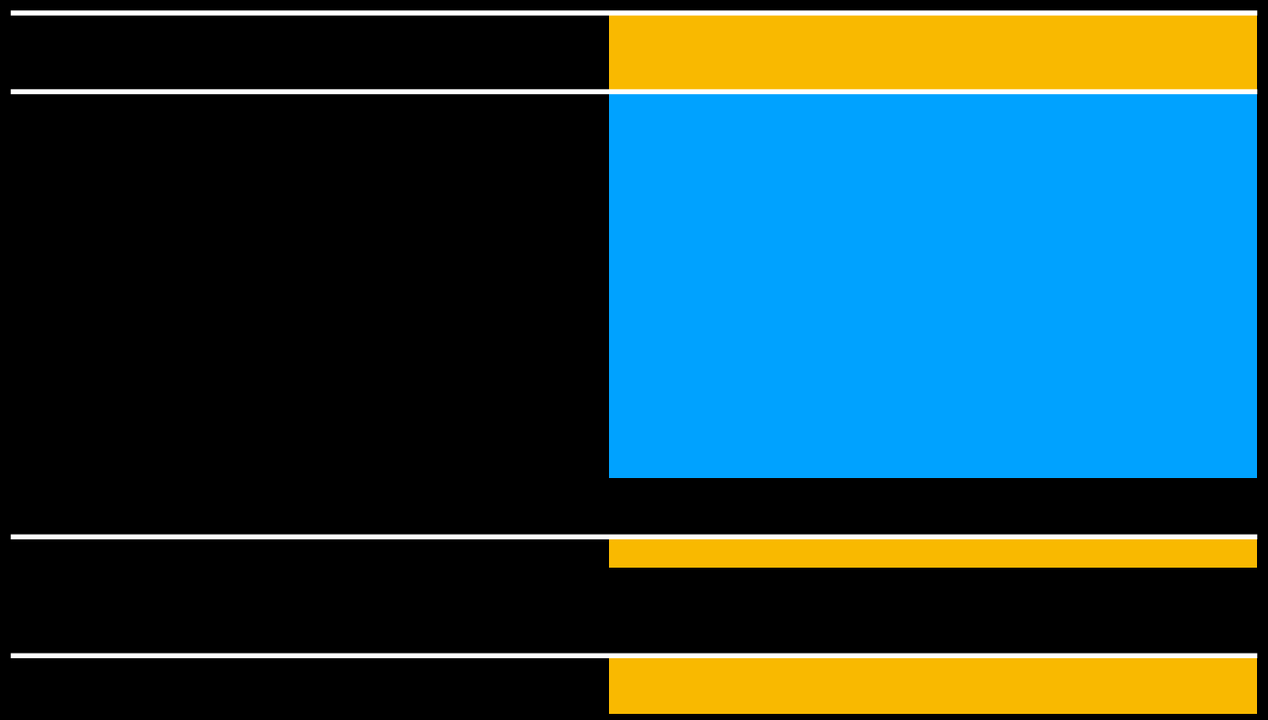
Arbitrary Code Execution

Embedding Strategy



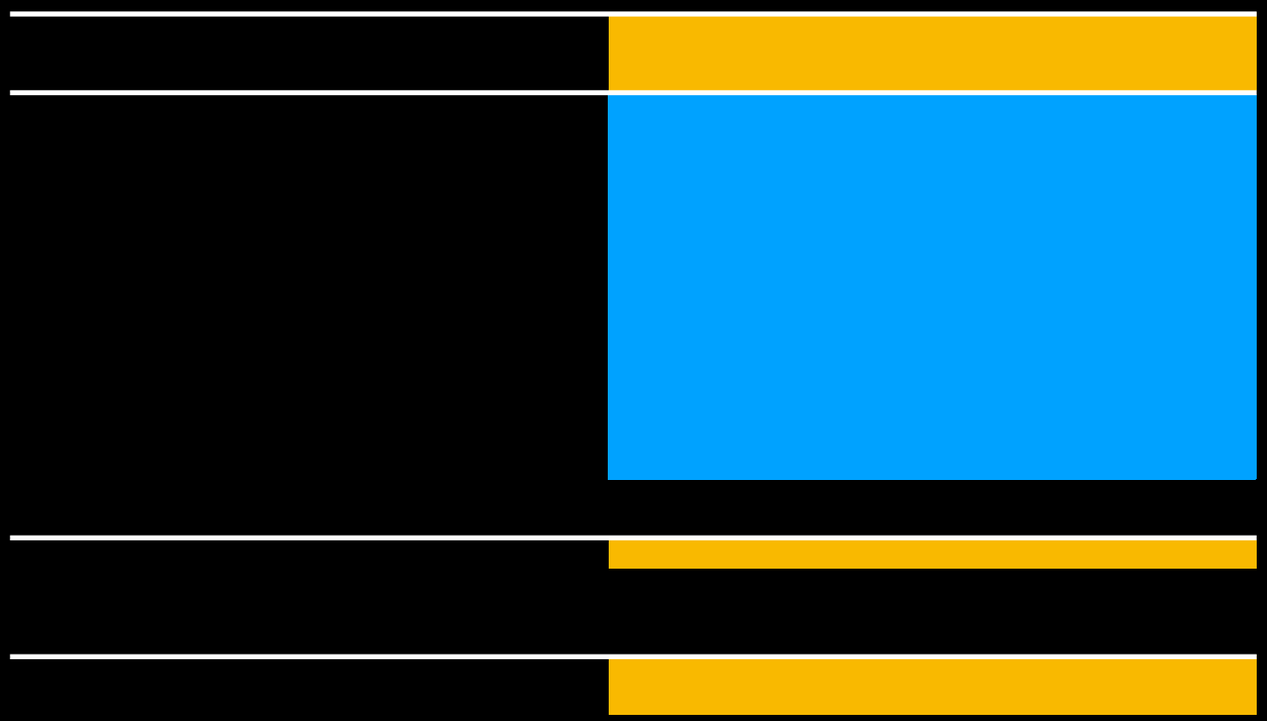
Arbitrary Code Execution

Embedding Strategy



Arbitrary Code Execution

Embedding Strategy



Arbitrary Code Execution

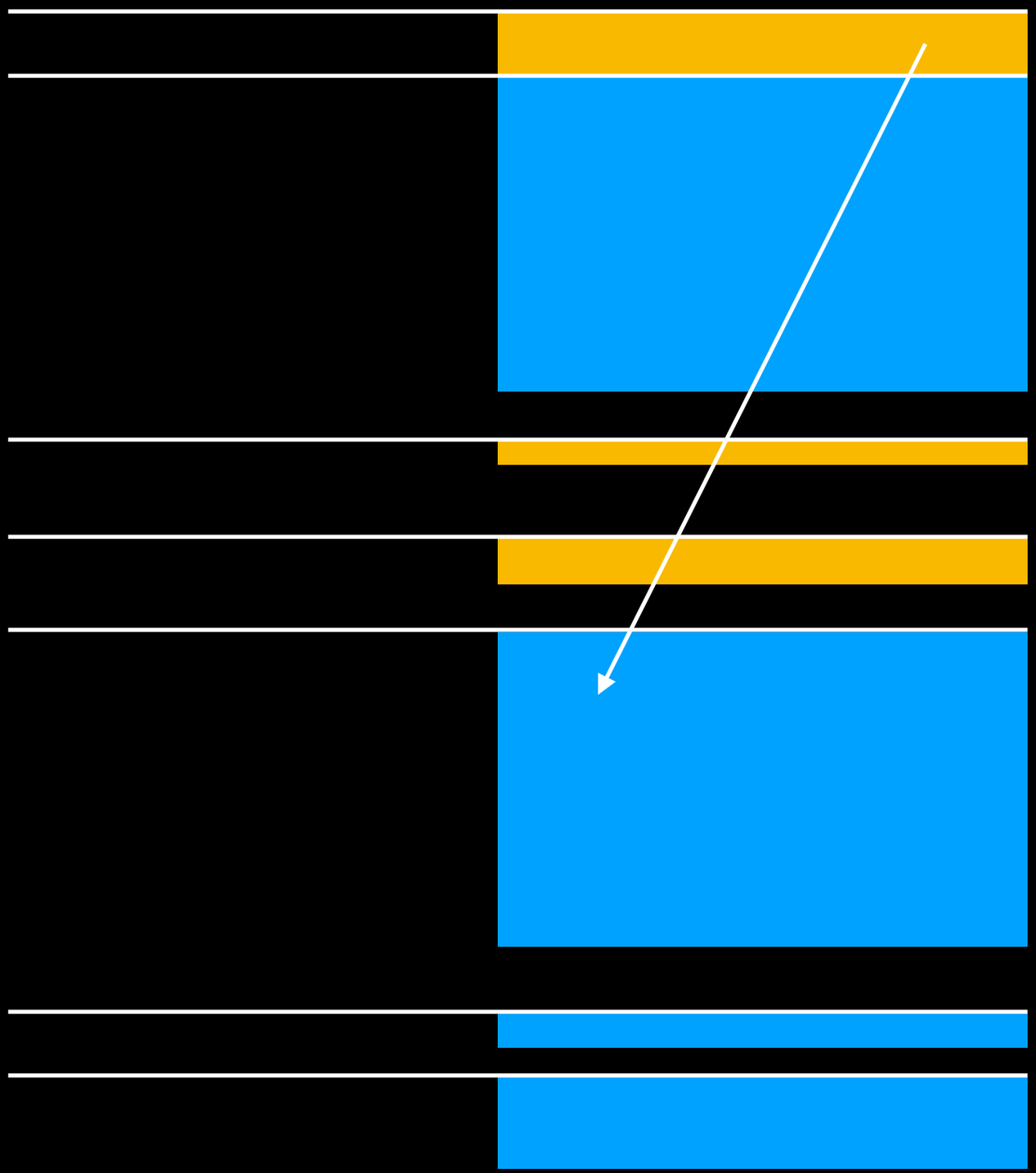
Embedding Strategy

Arbitrary Code Execution

Embedding Strategy

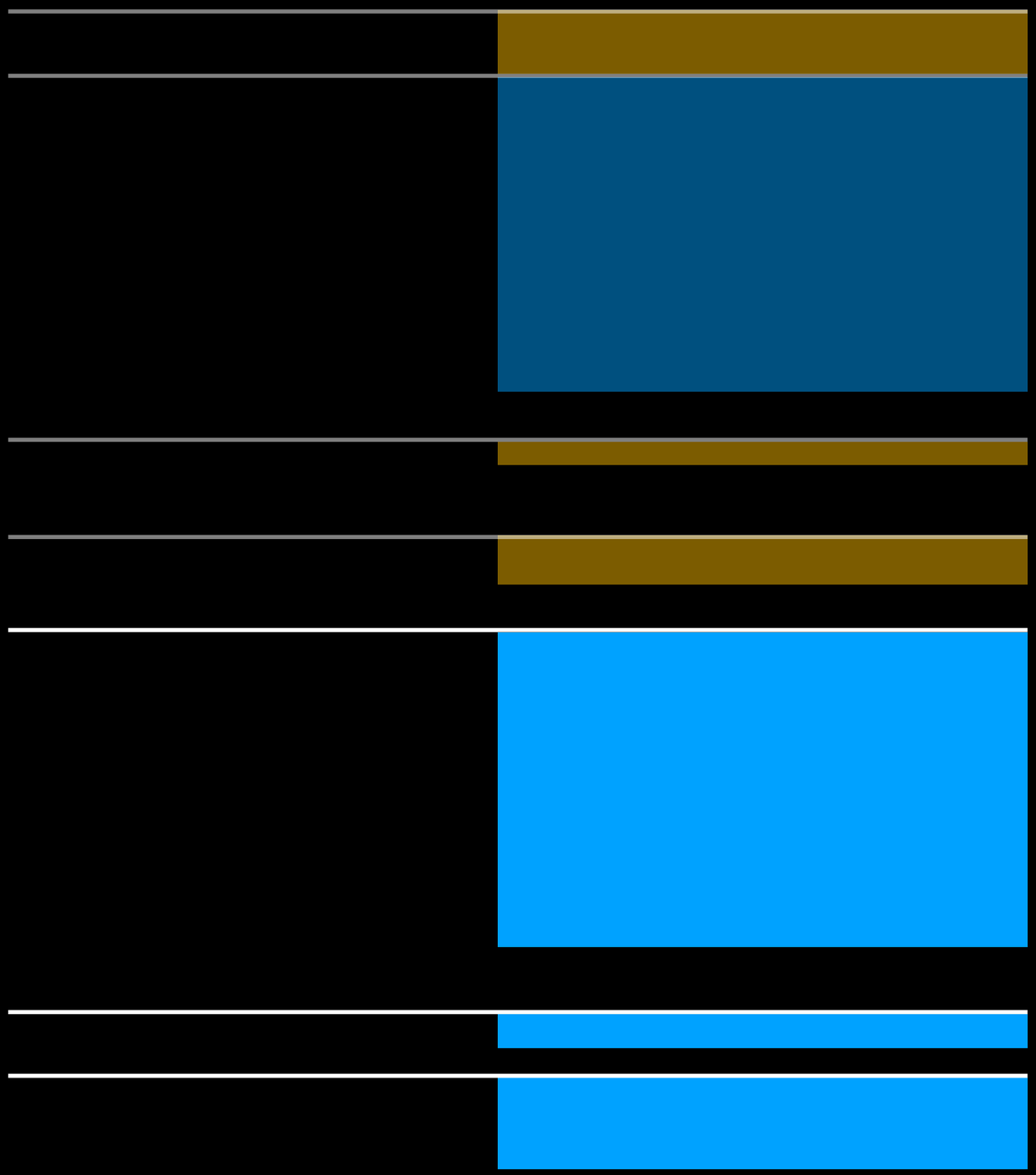
Arbitrary Code Execution

Embedding Strategy



Arbitrary Code Execution

Embedding Strategy



Embedding

Embedding
Using Swift code?

- Many trivial options to do this in C-family languages
  - `#embed`, `__attribute__((section))`, `.incbin`, …
- Obviously, none of these translate directly to Swift

Embedding

Using arrays?

- `let code = [/* … */] as [UInt8]`
- In –Onone:
  - Arrays go in `__TEXT,__text`
  - (…they’re compiled into a string of stores from immediates)
- In –O:
  - Arrays (the whole object) are stored in `__DATA,__data`

Embedding

Using strings?

- `let code = "…"`
- String literals go in `__TEXT,__cstring` by default
- All Strings in Swift must be valid Unicode
  - No \x “escape hatch” for arbitrary bytes
  - Storage is UTF-8
- Same parser is used for other “string-like” constructs in source
  - This means Swift cannot represent certain mangled symbols

Embedding

Linker fun

- ld64 takes a lot of interesting arguments
- Can we mess with where things get placed or their permissions?
  - Yes, but your final binary won’t execute if it’s malformed
  - Cannot bypass platform code signing policies
- Let’s read ld(1)

Embedding

Linker fun

- `-sectcreate __TEXT __code [binary]`
  - Also `-sectalign __TEXT __code 0x4000`
- Works!

- …but, LinkerExtension runs in a sandbox
  - Read access to Playgrounds.app/
  - Read access to folder with object files
  - Write access to final binary to produce
- No easy way to get our file into these places

Embedding

Linker fun

- Can we mark `__DATA` as code?
  - `-rename_section __DATA __data __TEXT __data`
- Works!
- …but, Swift Playgrounds requires a recent deployment target
- Chained fixups enabled in iOS 14+
- Kernel does page-in linking of binaries that have chained fixups
- `__DATA_CONST` is read-only
  - Not easy to get arbitrary data into it

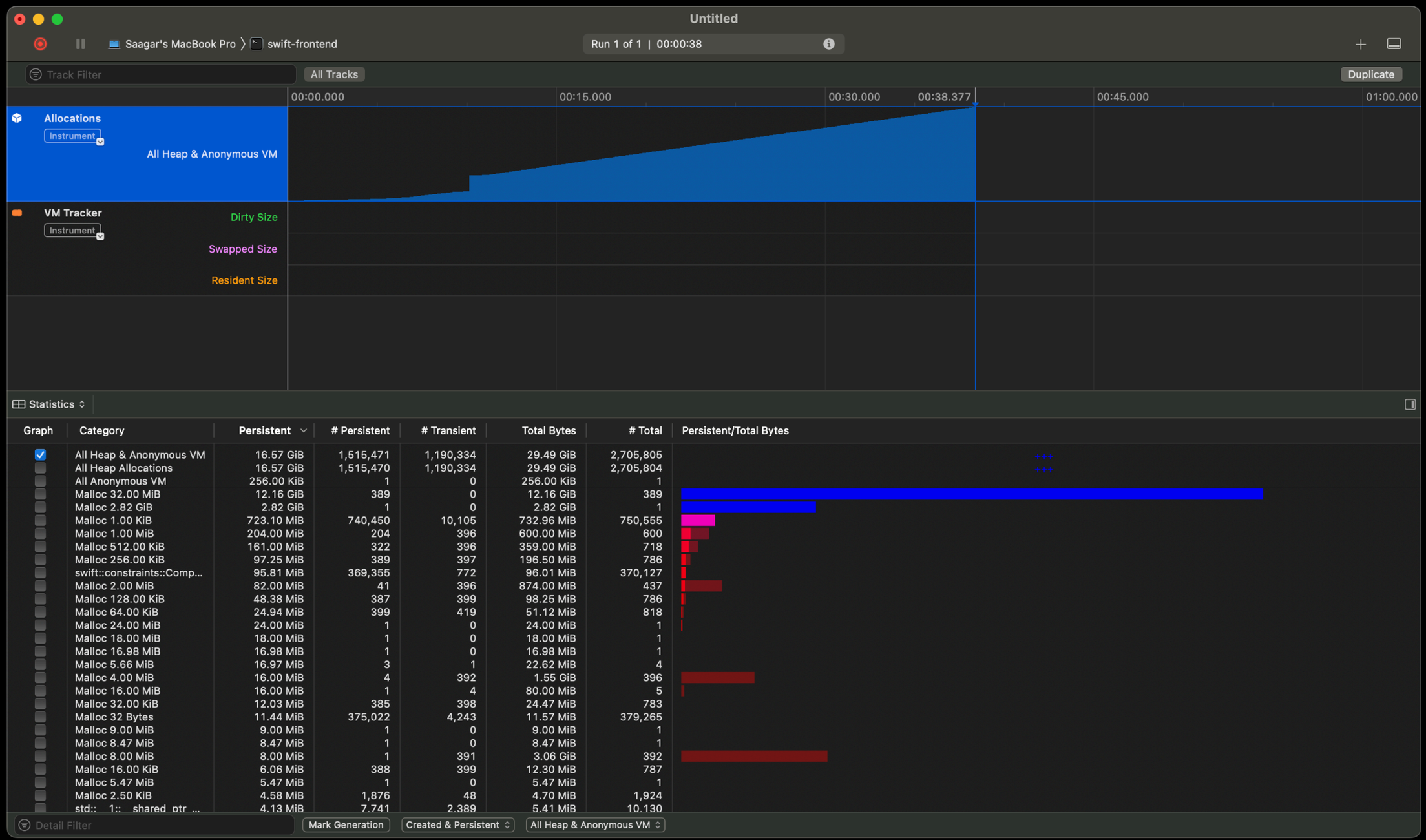
Embedding

Using tuples?

- `let code = (… as UInt8, …)`
- In `-Onone`:
  - Goes into `__DATA,__common`
- In -O:
  - Goes into `__TEXT,__const`
  - No object header
- Works!

Embedding

Using tuples?



VM Tracker

Instrument

Dirty Size

Swapped Size

Resident Size

Statistics

Graph	Category	Persistent	# Persistent	# Transient	Total By
☑	All Heap & Anonymous VM	16.57 GiB	1,515,471	1,190,334	29.49
☐	All Heap Allocations	16.57 GiB	1,515,470	1,190,334	29.49
☐	All Anonymous VM	256.00 KiB	1	0	256.00
☐	Malloc 32.00 MiB	12.16 GiB	389	0	12.16
☐	Malloc 2.82 GiB	2.82 GiB	1	0	2.82
☐	Malloc 1.00 KiB	723.10 MiB	740,450	10,105	732.96
☐	Malloc 1.00 MiB	204.00 MiB	204	396	600.00
☐	Malloc 512.00 KiB	161.00 MiB	322	396	359.00
☐	Malloc 256.00 KiB	97.25 MiB	389	397	196.50
☐	swift::constraints::Comp...	95.81 MiB	369,355	772	96.01
☐	Malloc 2.00 MiB	82.00 MiB	41	396	874.00

Embedding

Using tuples?

Why is Swift so slow (timeout) in compiling this code?

Using Swift `slow`

[SR-657] Out of memory during compilation #43274

Open **swift-ci** opened this issue on Feb 2, 2016 · 4 comments

[SR-7703] Creating an array with 10.000 elements is slow with optimization. #50243

Open **swift-ci** opened this issue on May 16, 2018 · 18 comments

[SR-7070] Compilation never seems to finish for literal array expression #49618

Open **swift-ci** opened this issue on Feb 24, 2018 · 1 comment

Excessive Memory Usage on Linux when Compiling Large Arrays #60549

Open **NeedleInAJayStack** opened this issue on Aug 14, 2022 · 2 comments

What now?

What now?

- I started looking into this in 2018 for a WWDC scholarship
  - Ended up submitting an “Inside Playgrounds” instead
- Swift Playgrounds seems to be resistant for now
  - However, it seems to mostly be by accident
- I believe that this will eventually be possible
  - More analysis
  - New features in the app/Swift

Questions?