

# App Startup Time: Past, Present, and Future

Session 413

Louis Gerbarg, Senior Linker Engineer

Review of app launch startup advice

New tools to help find slow initializers

Brief history of dyld

The all new dyld that is coming in this  
years Apple OS platforms

Best practices

# Preamble

We want your feedback

- Please file bug reports with “DYLD USAGE:” in the title

## Terminology

- Startup time
  - For the purposes of this talk startup time is everything that happens before `main()` is called
- Launch Closure
  - All of the information necessary to launch an application

# Improving App Startup Time

Do less!

- Embed fewer dylibs
- Declare fewer classes/methods
- Use fewer initializers

Use more Swift

- No initializers
- Swift size improvements

# Improving App Startup Time

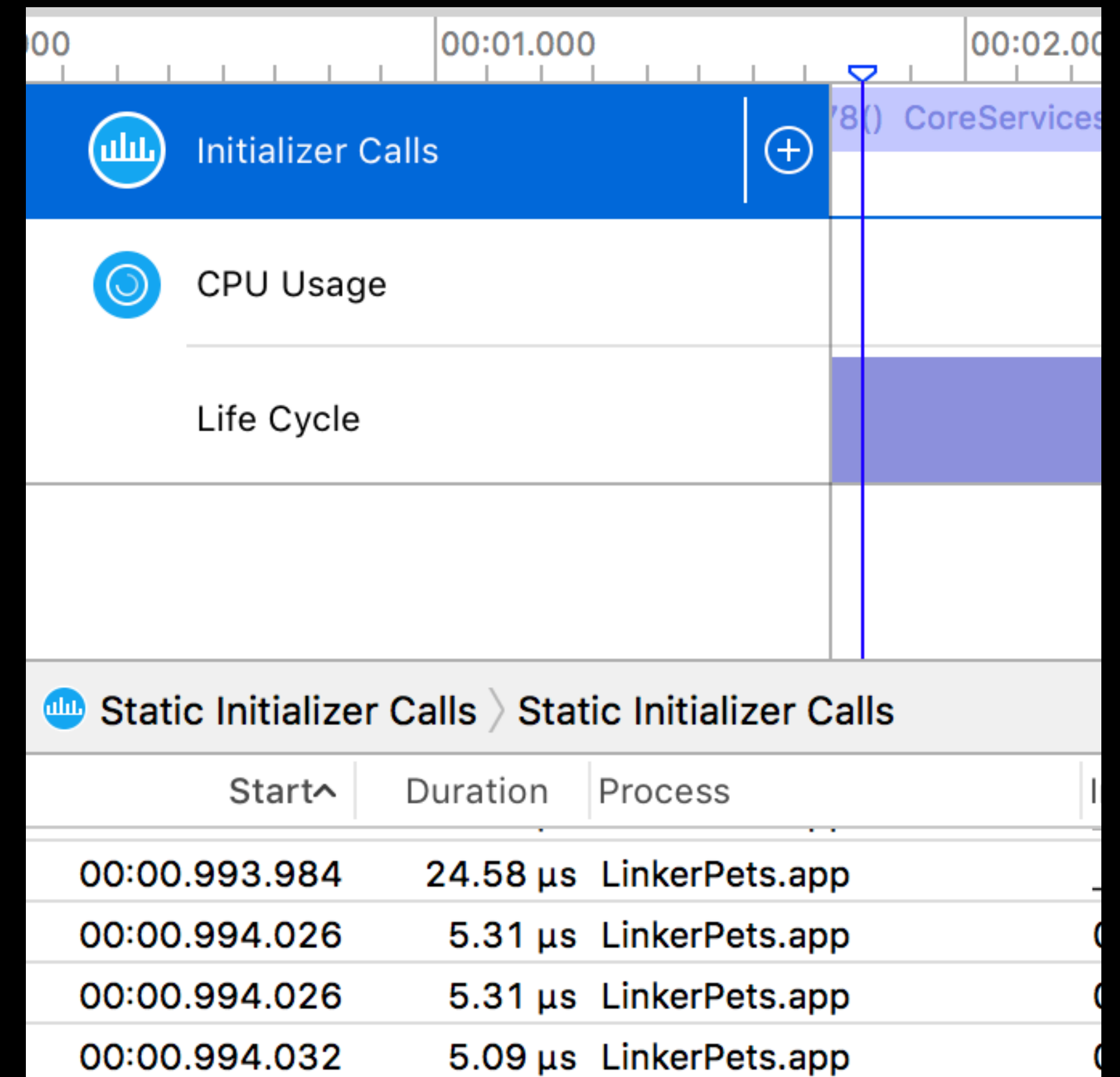
Static initializer tracing

NEW

New in iOS 11 and macOS High Sierra

Provides precise timing for each static initializer

Available through Instruments



# Dynamic Linking Through the Ages

dyld 1.0 (1996–2004)

Shipped in NeXTStep 3.3

Predated POSIX dlopen() standardized

- Third-party wrapper functions

Before most systems used large C++ dynamic libraries

Prebinding added in macOS Cheetah (10.0)

# Dynamic Linking Through the Ages

dyld 2.0 (2004–2007)

Shipped in macOS Tiger

Complete rewrite

- Correct C++ initializer semantics
- Full native dlopen()/dlsym() semantics

Designed for speed

- Limited sanity checking
- Security “Issues”

Reduced prebinding

# Dynamic Linking Through the Ages

dyld 2.x (2007–2017)

More architectures and platforms

- x86, x86\_64, arm, arm64
- iOS, tvOS, watchOS

Improved security

- Codesigning, ASLR, bounds checking

Improved performance

- Prebinding completely replaced by shared cache



# Dynamic Linking Through the Ages

## Shared Cache

Introduced in iOS 3.1 and macOS Snow Leopard

- Replaced prebinding

Single file that contains most system dylibs

- Rearranges binaries to improve load speed
- Pre-links dylibs
- Pre-builds data structures used dyld and ObjC

Built locally on macOS, shipped as part of all other Apple OS platforms

# Dynamic Linking Through the Ages

dyld 3 (2017)

Announcing it today

Complete rethink of dynamic linking

On by default for most macOS system apps in this weeks seed

Will be on be the default for system apps for 2017 Apple OS platforms

Will completely replace dyld 2.x in future Apple OS platforms

# dyld 3

Why?

Performance

- What is the minimum amount of work we can do to start an app?

Security

- Can we have more aggressive security checks?

Reliability

- Can we design something that is easier to test?

# dyld 3

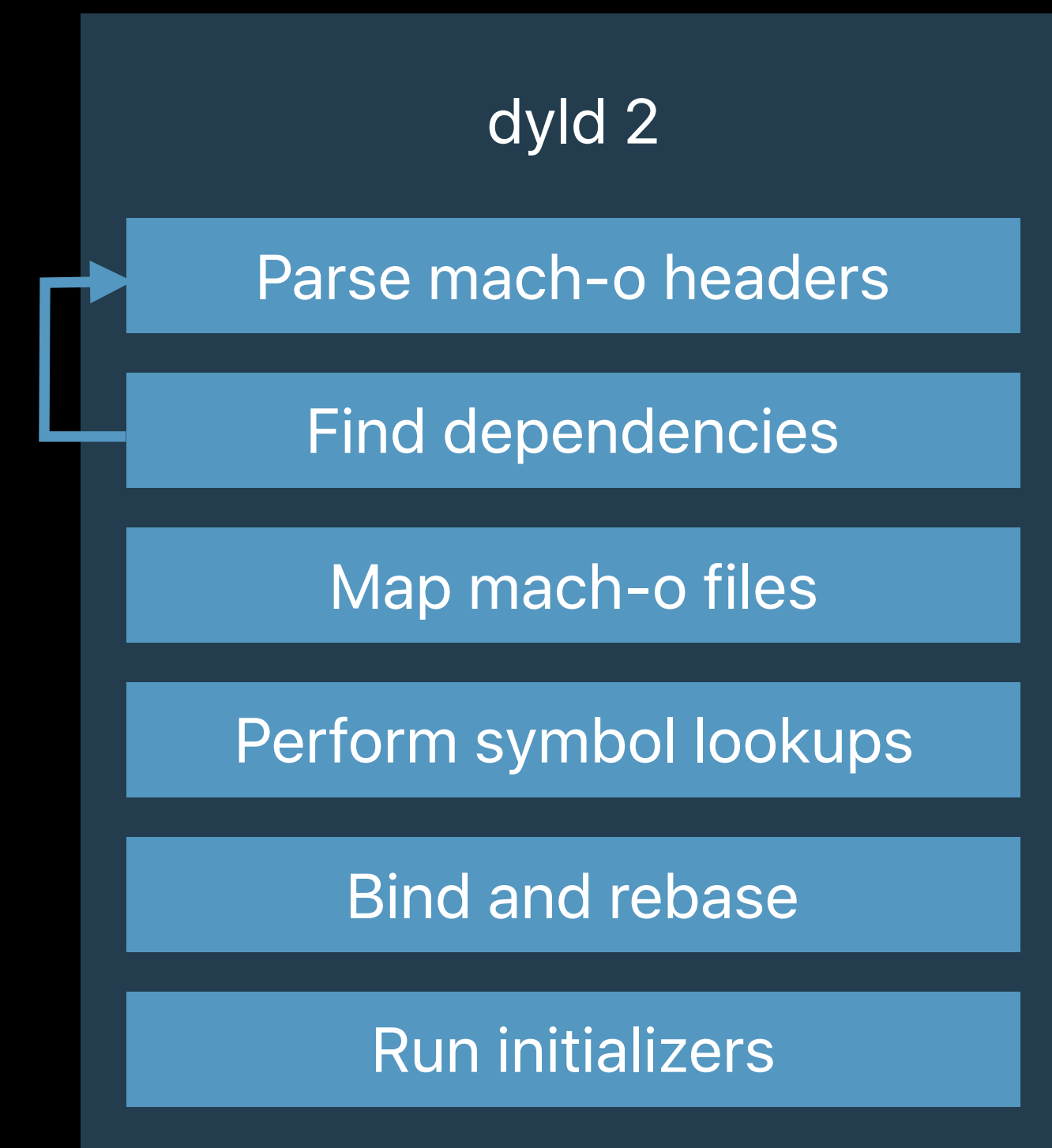
How?

Move complex operations out of process

- Most of dyld is now a regular daemon

Make the rest of dyld as small as possible

- Reduces attack surface
- Speeds up launch
  - The fastest code is code you never write
  - Followed closely by code you almost never execute



# dyld 3

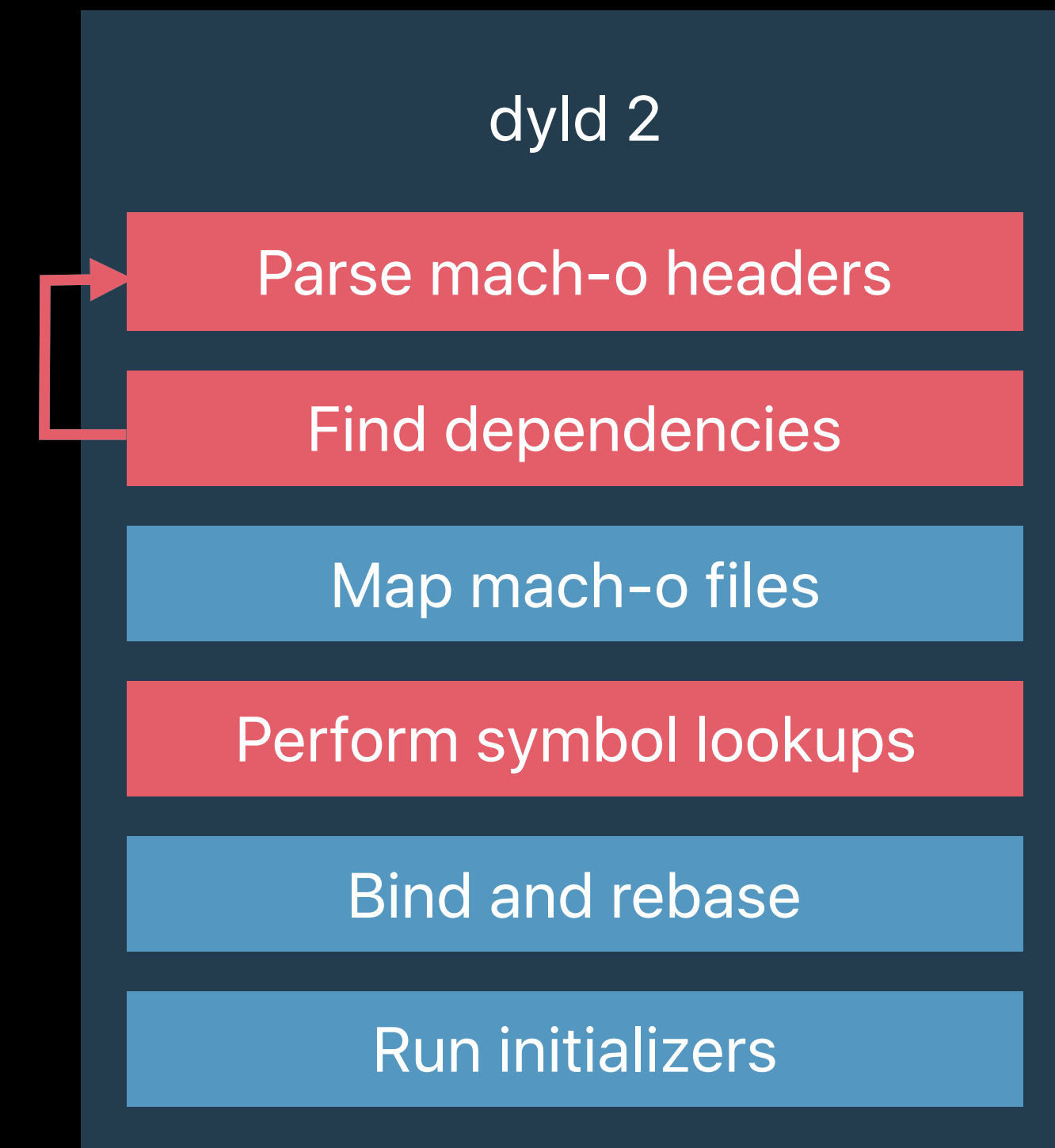
How?

Identify security sensitive components

- Bounds checking
- @rpath confusion attacks

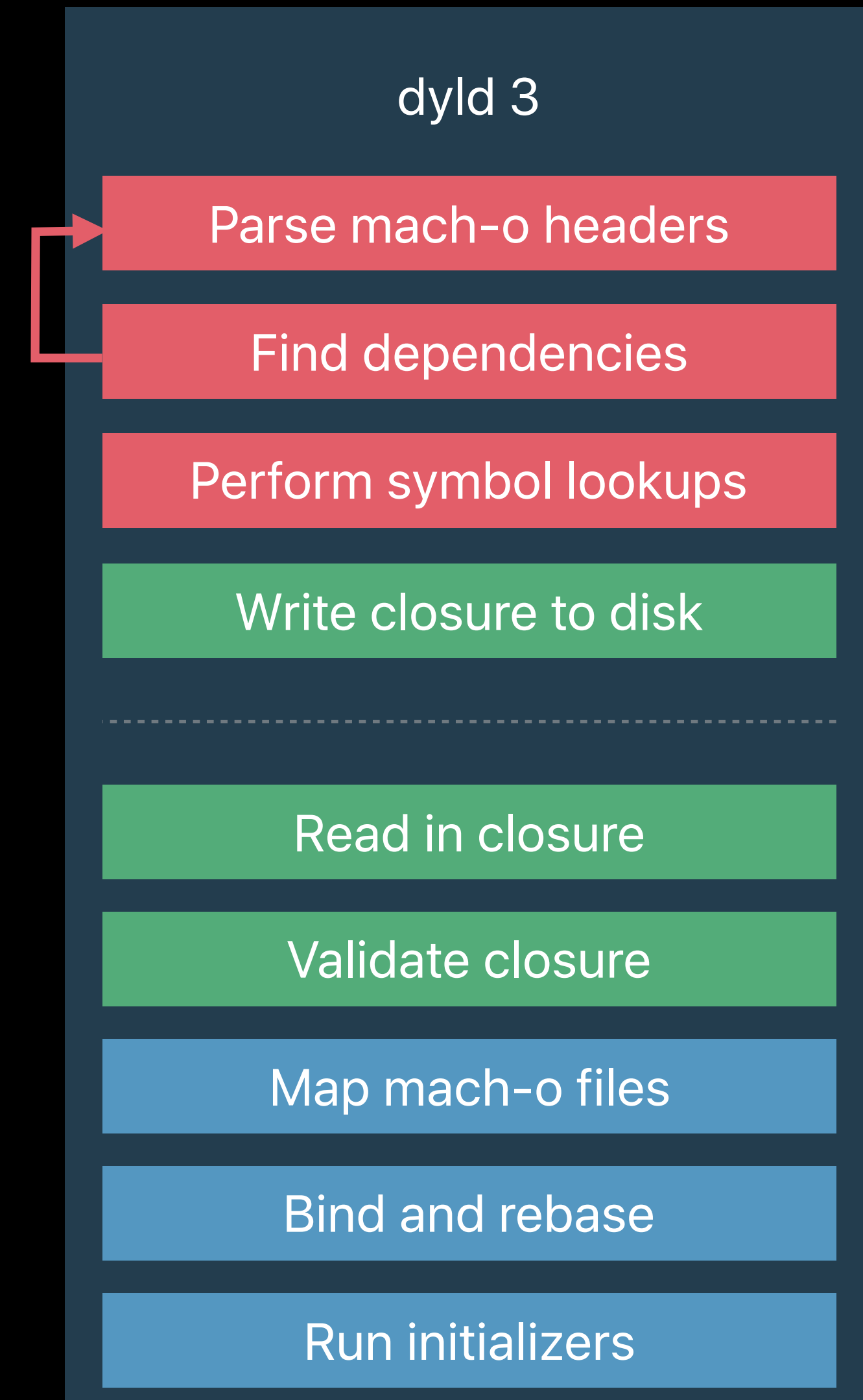
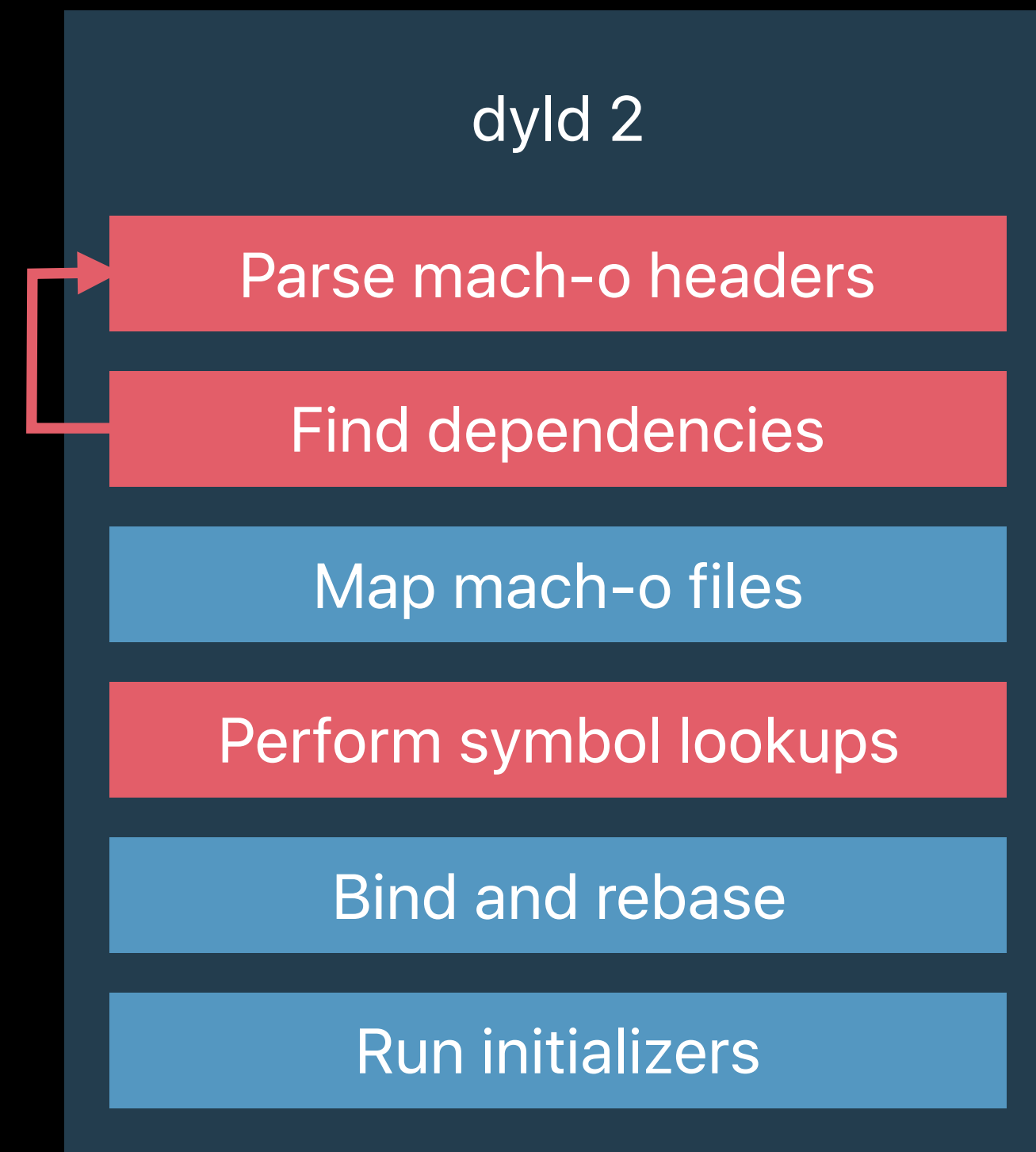
Identify components that are cache-able

- Dependencies don't change between launches
- Symbol locations within a mach-o do not change between launches



# dyld 3

## Architecture



# dyld 3

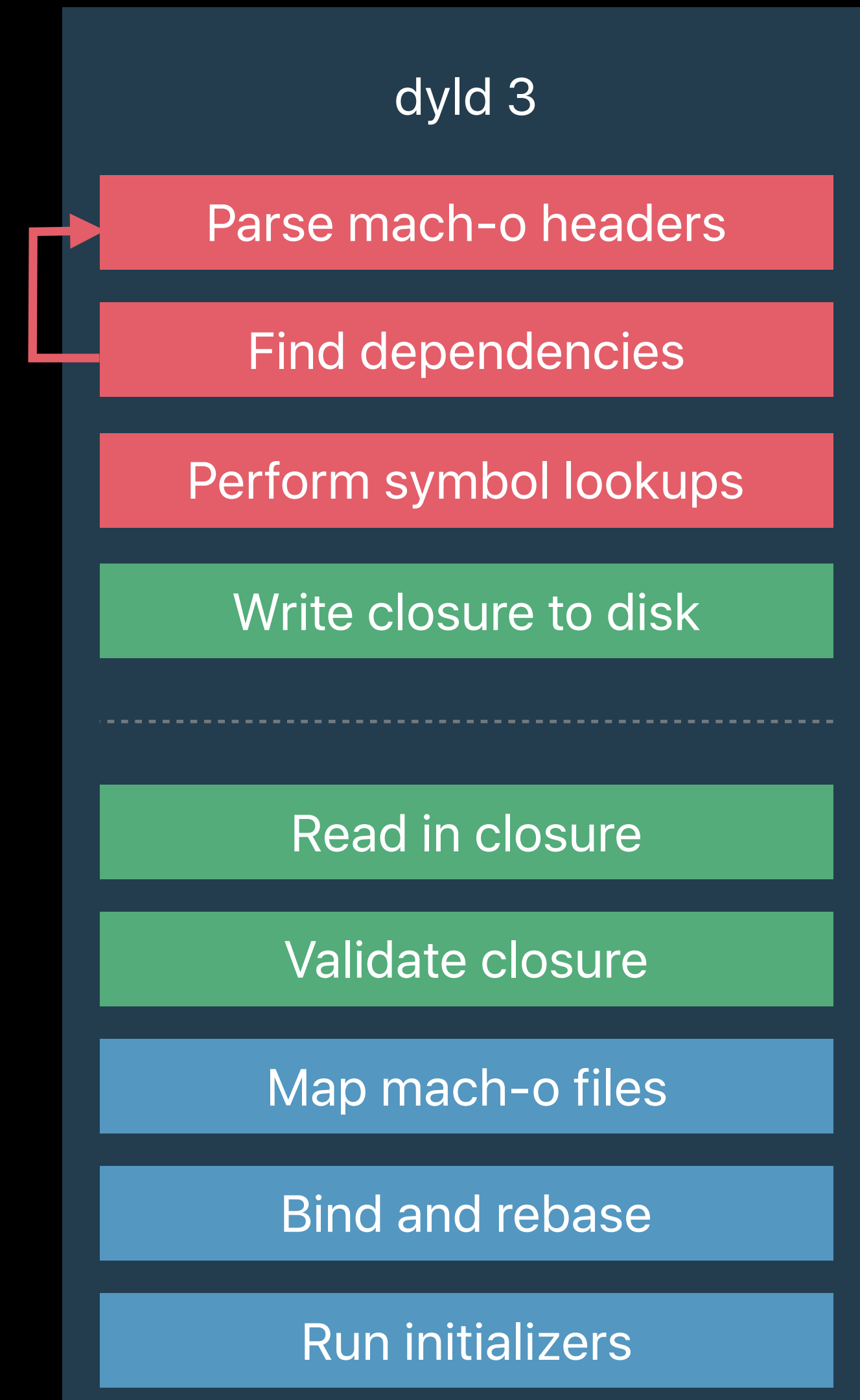
## Architecture

dyld 3 has 3 components

- An out of process MachO parser/compiler
- An in-process engine that runs launch closures
- A launch closure caching service

Most launches use the cache and never invoke the out-of-process mach-o parser/compiler

- Launch closures are simpler than mach-o
- Launch closures are built for speed

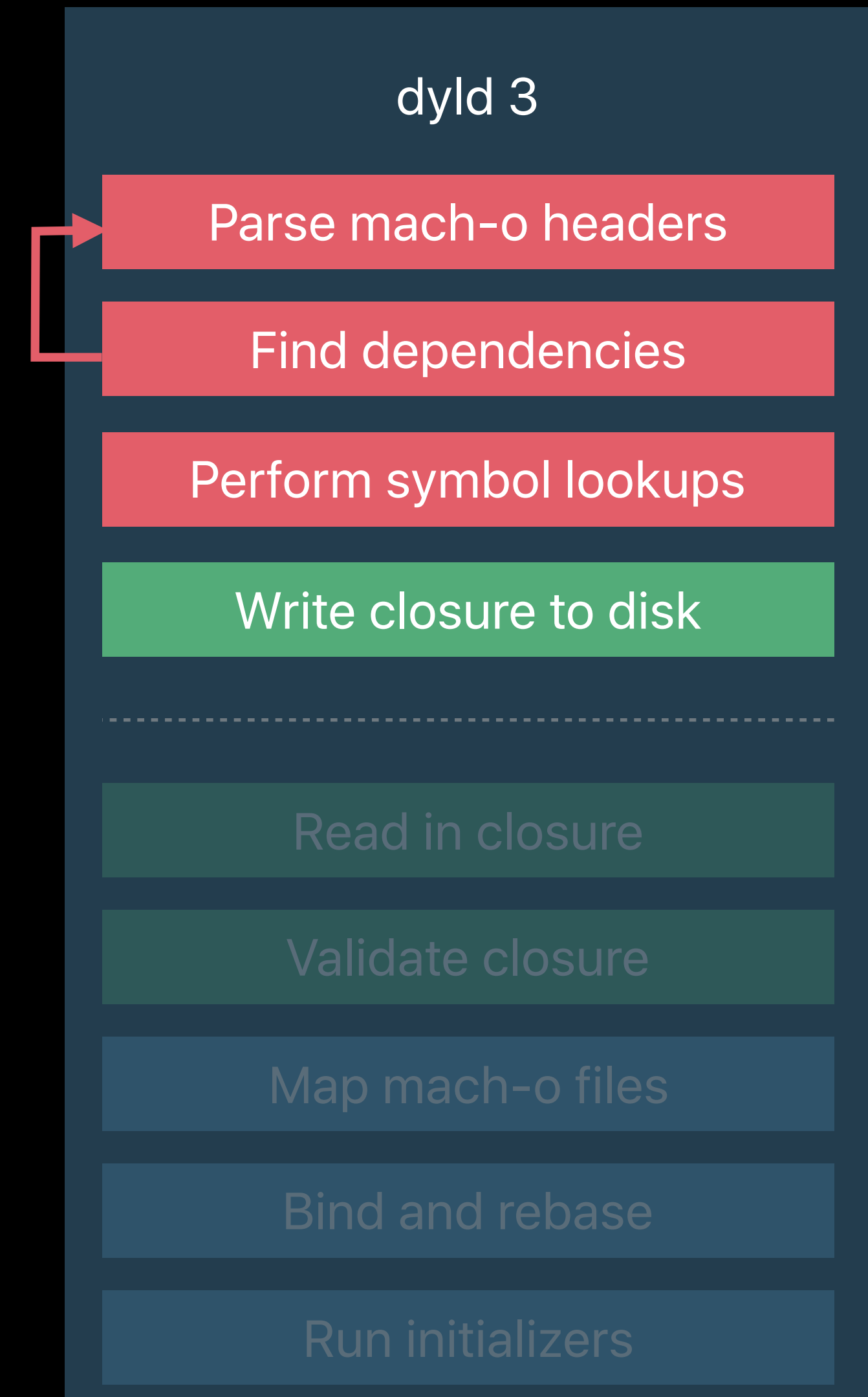


# dyld 3

## Architecture

dyld 3 is an out-of-process mach-o parser

- Resolves all search paths, @rpaths, environment variables
- Parses the mach-o binaries
- Performs all symbol lookups
- Creates a launch closure with results
- Is a normal daemon that can use normal testing infrastructure





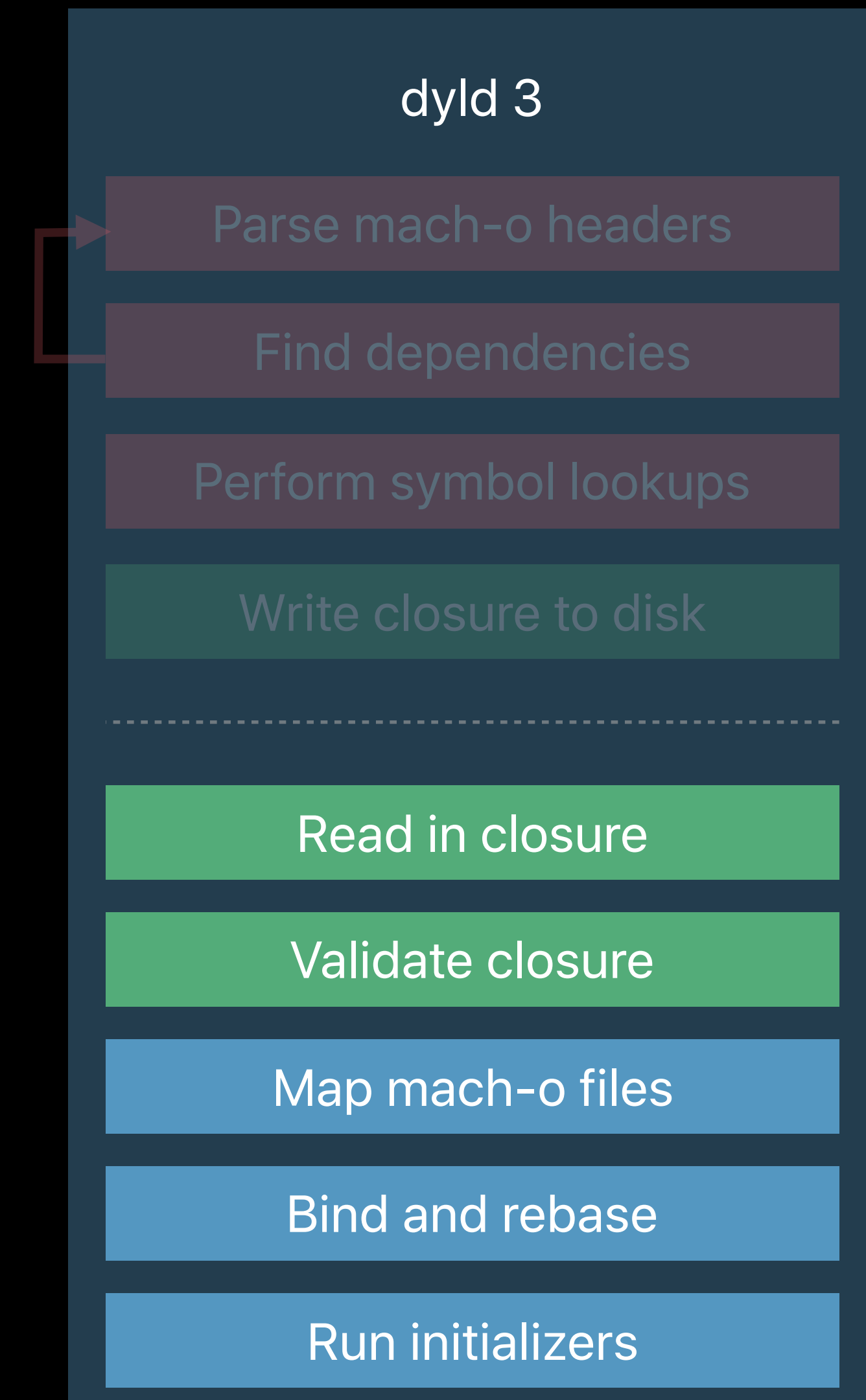
# dyld 3

## Architecture

dyld 3 is a small in-process engine

- Validates launch closure
- Maps in all dylibs
- Applies fixups
- Runs initializers
- Jumps to main()

Never needs to parse mach-o headers or access the symbol tables



# dyld 3

## Architecture

dyld 3 is a launch closure cache

- System app launch closures built into shared cache
- Third-party app launch closures built during install
  - Rebuilt during software update
- On macOS the in process engine can call out to a daemon if necessary
  - Not necessary on other Apple OS platforms

# Preparing for dyld 3

## Potential issues

Fully compatible with dyld 2.x

- Some existing APIs disable dyld 3's optimizations or require slow fallback paths
- Some existing optimizations done for dyld 2.x no longer have any impact

Stricter linking semantics

- Workarounds for old binaries
- New binaries will cause linker errors

# Preparing for dyld 3

## Unaligned pointers in \_\_DATA

When you have a global struct it is placed in the data segment

- Unaligned pointers in the struct will be embedded in the \_\_DATA segment

Fixing up unaligned pointers is more complex

- Can span multiple pages
- Can have atomicity issues

The static linker already emits a warning

ld: warning: pointer not aligned at address 0x10056E59C

```
struct ListHead {  
};  
  
#pragma pack(1) // Changes default alignment globally  
struct List {  
    uint32_t count; // 4 bytes @ 0x0  
    struct ListElement *head; // 8 bytes @ 0x4: MISALIGNED!!  
} __attribute__((__packed__, aligned(1))); // Changes alignment for this struct  
  
static struct ListElement sHead;  
struct List gList = {0, &sHead}; //pointer not aligned at address 0x100001004 (_gList + 4 from  
...)
```

```
struct ListHead {  
};
```

```
#pragma pack(1) // Changes default alignment globally
```

```
struct List {  
    uint32_t count; // 4 bytes @ 0x0  
    struct ListElement *head; // 8 bytes @ 0x4: MISALIGNED!!  
} __attribute__((__packed__, aligned(1))); // Changes alignment for this struct
```

```
static struct ListElement sHead;
```

```
struct List gList = {0, &sHead}; //pointer not aligned at address 0x100001004 (_gList + 4 from  
...)
```

```
struct ListHead {  
};  
  
#pragma pack(1) // Changes default alignment globally  
struct List {  
    uint32_t count; // 4 bytes @ 0x0  
    struct ListElement *head; // 8 bytes @ 0x4: MISALIGNED!!  
} __attribute__((__packed__, aligned(1))); // Changes alignment for this struct  
  
static struct ListElement sHead;  
struct List gList = {0, &sHead}; //pointer not aligned at address 0x100001004 (_gList + 4 from  
...)
```

# Preparing for dyld 3

## Eager symbol resolution

dyld 2 performs lazy symbol resolution

- Symbol lookups are too expensive to do them up front
- Each symbol is looked up the first time you call it
- Missing symbols cause a crash the first time they are called

dyld 3 performs eager symbol resolutions

- Since all symbol lookups are cached it is very fast
- Makes it possible to check if all symbols are present



# Preparing for dyld 3

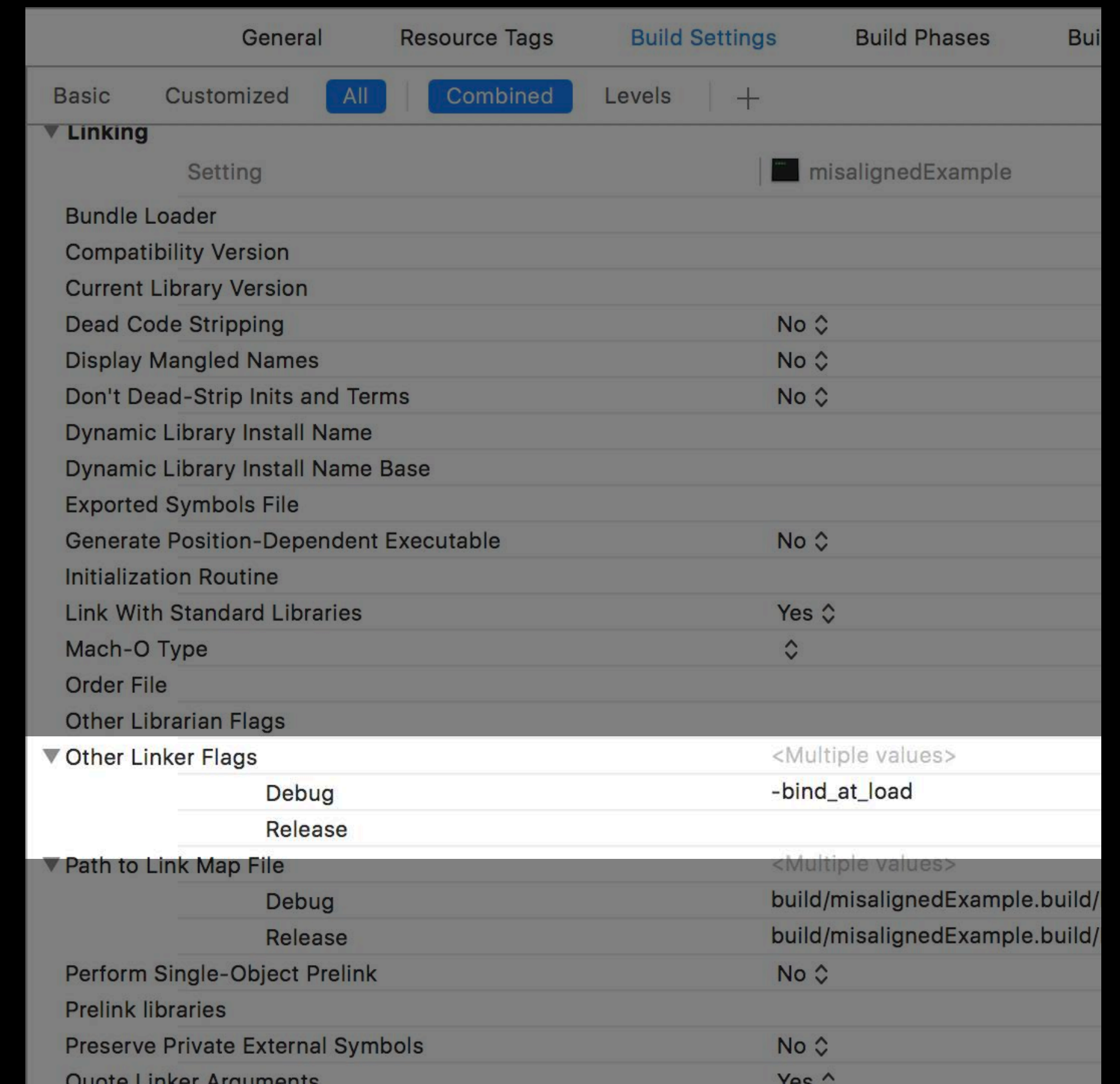
## Eager symbol resolution

Apps built against current SDKs will run with unknown symbols

- Identical behavior to dyld 2, on first call it will crash

Apps built against future SDKs will fail to launch with unknown symbols

- Can simulate behavior today with `-bind_at_load` linker flag
- Only use on test builds, not release



# Preparing for dyld 3

`dlopen()/dlsym()/dladdr()`

Still have problematic semantics

- Still necessary in some cases

Symbols found with `dlsym()` must be found at runtime

- Cannot be pre-linked by dyld 3

We are working on better alternatives

- Want to hear about your use cases
- Not going away, but may be slower in dyld 3

# Preparing for dyld 3

`dlclose()`

Misnomer

- Decrements a refcount, does not necessarily close the dylib
- Not appropriate for resource management

Features that prevent a dylib from unloading

- Objective-C classes
- Swift classes
- C `__thread` and C++ `thread_local` variables

Considering making `dlclose()` a no-op everywhere except macOS

# Preparing for dyld 3

`all_image_infos`

Interface for introspecting dylibs in a process

- Struct in memory
- Wastes a lot of memory
- Going away in future releases
- We will be providing replacement APIs
- Please let us know how you use it

# Preparing for dyld 3

## Best Practices

Make sure your app launches when built with `-bind_at_load` added to `LD_FLAGS`

- Debug builds only

Fix any unaligned pointers in your app's `__DATA` segment

ld: warning: pointer not aligned at address 0x100001004

Make sure you are not depending on terminators running when you call `dlclose()`

Let us know why you are using `dlopen()/dlsym()/dladdr()/all_image_infos`

- File bug reports with "DYLD USAGE:" in their titles

# More Information

<https://developer.apple.com/wwdc17/413>

# Related Sessions

---

Optimizing App Startup Time

---

WWDC 2016

# Labs

Optimizing App Startup Time Lab	Technology Lab E	Fri 11:00AM–12:30PM
---------------------------------	------------------	---------------------



