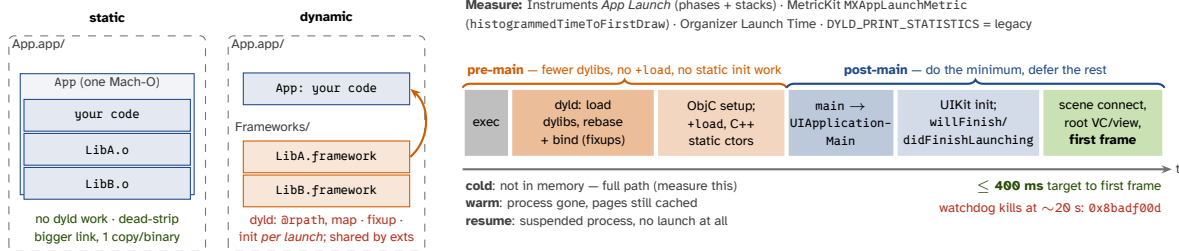


In one line: **Static** code is copied into your binary at link time (no load cost, but one copy per binary); **dynamic** code stays a separate Mach-O that **dyld** maps, fixes up and initialises at every launch. Launch = **pre-main** (dyld + runtime init) + **post-main** (UIKit + your `didFinishLaunching` → first frame); aim for first frame in ≤ 400 ms.



How it works — linking

- .a** = archive of **.o**, static, no resources. **.framework** = bundle (binary + headers + modulemap + resources), static or dynamic. **.xcframework** = per-platform slices (ios-arm64, ios-arm64_t8020, ios-simulator, ...); a fat lipo binary cannot hold device-arm64 and simulator-arm64.
- Dynamic** → **Embed & Sign** (copied to App.app/Frameworks, found via `@rpath` = `@executable_path/Frameworks`). **Static** and system frameworks → **Do Not Embed**.
- System frameworks sit in the **dyld shared cache** (pre-linked); your dylibs pay full load + fixup cost. dyld3/4 cache a launch closure and use chained fixups, but many dylibs still hurt.
- Mergeable libraries** (Xcode 15): framework built with `MERGEABLE_LIBRARY`, app with `MERGED_BINARY_TYPE` — dynamic in Debug (fast iteration), *merged* into the app binary in Release.
- Swift binary frameworks need `BUILD_LIBRARY_FOR_DISTRIBUTION = YES(.swiftinterface, module stability)` or only the exact compiler can import them.

How it works — launch fixes

- Fewer dylibs (static/merge); no +load (use +initialize or explicit init); Swift globals/static let are already lazy — but touching one in `didFinishLaunching` runs its init right there.
- Build only the first screen; lazy DI factories; no sync disk/network on main; defer SDKs to *after* the first frame, heavy work off the main thread.
- Static LaunchScreen is drawn by the system — free; no splash delays.
- Profile **Release, cold**, on a low-end device.

Build settings — who wins

- Target** = one product; **configuration** (Debug/Release) = a named set of setting values; **scheme** = which targets + which configuration per action (Run/Test→Debug, Profile/Archive→Release by default). Optimization level is a *configuration* setting — “edit the scheme” is the wrong answer.
- Precedence, highest first: `xcodebuild SETTING=v` → target (pbxproj) → target .xcconfig → project (pbxproj) → project .xcconfig → SDK default. Assigning *replaces*; write `$(inherited)` to append.
- Swift `#if DEBUG` reads `SWIFT_ACTIVE_COMPILATION_CONDITIONS`; ObjC reads `GCC_PREPROCESSOR_DEFINITIONS` — define it in both.

Example

```
func application(_ a: UIApplication, didFinishLaunchingWithOptions
    o: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
    CrashReporter.start()           // only what must exist before frame 1
    return true                     // root UI built by the scene delegate
}
// after first frame (root view.task / first viewDidAppear):
Task.detached(priority: .utility) {
    Analytics.configure(); await ImageCache.warm() } // off main

// Release.xcconfig
#include "Base.xcconfig"
OTHER_LDFLAGS = $(inherited) -ObjC // keep Pods'/parent flags
SWIFT_ACTIVE_COMPILATION_CONDITIONS = $(inherited) RELEASE
```

Interview traps

- Static + Embed & Sign** = the code ships twice (inside the binary and in Frameworks/); App Store validation may reject it.
- Same static lib linked into a dynamic framework *and* the app → two copies; ObjC logs “Class X is implemented in both ...”, and singletons split.
- “Library not loaded: @rpath/...” = linked but not embedded.
- Categories in a static lib vanish (unrecognized selector) without -ObjC.
- `DispatchQueue.main.async` from `didFinishLaunching` still runs on main — it only moves work past the first frame, not off the thread.
- `DYLD_PRINT_STATISTICS` is ignored by dyld4 (iOS 15+); use the App Launch instrument. iOS 15 *prewarming* may run pre-main early, so don't time launch from inside main.

Remember

“Static = copy, dynamic = dyld; pre-main is dylibs + initialisers; post-main is your code; first frame in 400 ms, watchdog at 20 s.”

Likely questions

- Why are many dynamic frameworks slow? — each one is mapped, fixed up and initialised by dyld per launch.
- Mergeable libraries? — dynamic in Debug, merged statically into the app in Release (Xcode 15).
- Cold vs warm vs resume? — nothing cached / process gone but pages cached / suspended process.
- How do you measure launch? — App Launch instrument locally; MetricKit + Organizer in the field.
- Why `$(inherited)`? — assignment replaces the lower level's value.
- What is rebasing vs binding? — rebase = slide internal pointers by the ASLR offset; bind = resolve symbols in other images.