

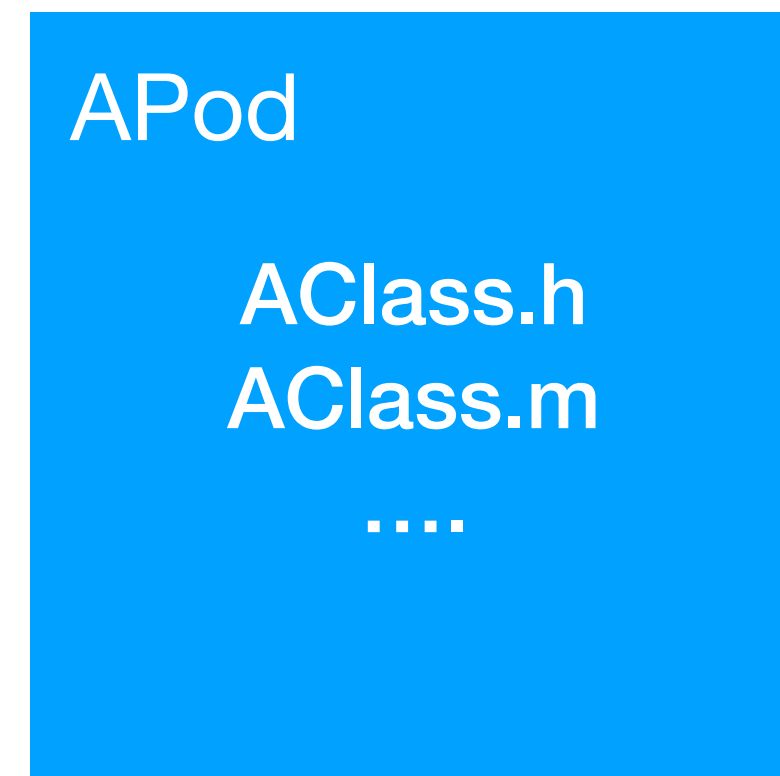
使用 Xcode Cache 为构建打包提速



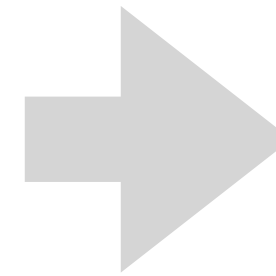
为什么要用 XCache, Pod 二进制不香吗? 🤔

1. 为什么要用 XCache

什么是 Pod 二进制?



源码



Pod 二进制

1. 为什么要用 XCache

问题

```
// PodA/AClass.m
#import <PodB/BClass.h>

@implementation AClass

- (void)func {
    [BClass method1];
}

@end
```

PodA/AClass.m

```
// PodB/BClass.h
@interface BClass : NSObject

+ (void)method1;

@end

// PodB/BClass.m
#import "BClass.h"
@implementation BClass

+ (void)method1 {
    // do something ...
}

@end
```

PodB/BClass.m

1. 为什么要用 XCache

问题

```
// PodA/AClass.m
#import <PodB/BClass.h>

@implementation AClass

- (void)func {
    [BClass method1];
}

@end
```

PodA 二进制

```
// PodB/BClass.h
@interface BClass : NSObject

//+ (void)method1;

@end

// PodB/BClass.m
#import "BClass.h"
@implementation BClass

//+ (void)method1 {
    // do something...
//}

@end
```


PodB 代码变更，源码编译通过

1. 为什么要用 XCache

Pod 二进制污染

```
7
8 #import "AClass.h"
9
10 @implementation AClass
11
12 + (void)func {
13     [BClass method1:BValueMacro];
14 }
15
16 @end
17
```

Thread 1: "+[BClass method1:]: unrecognized selector sent to class 0x10e3f2160"



PodBinaryPollution > Thread 1 > 13 +[AClass func]

Ex Exception = (NSException *) "+[BClass method1:]: unrecognized selector sent to class 0x10e3f2160"

A self = (const Class) AClass

```
*** Terminating app due to uncaught exception 'NSInvalidArgumentException', reason: '+[BClass method1:]: unrecognized selector sent to class 0x10e3f2160'
terminating with uncaught exception of type NSException
CoreSimulator 757.5 - Device: iPod touch (7th generation) (72717C5D-A678-4F1E-9185-061E43FD0F1E)
- Runtime: iOS 14.5 (18E182) - DeviceType: iPod touch (7th generation)
2021-06-09 14:10:40.977767+0800 PodBinaryPollution[86404:131751714] +[BClass method1:]: unrecognized selector sent to class 0x10e3f2160
2021-06-09 14:10:40.978806+0800 PodBinaryPollution[86404:131751714] *** Terminating app due to uncaught exception 'NSInvalidArgumentException', reason: '+[BClass method1:]: unrecognized selector sent to class 0x10e3f2160'
*** First throw call stack:
(
    0  CoreFoundation          0x00007fff20422fba __exceptionPreprocess + 242
    1  libobjc.A.dylib          0x00007fff20193ff5 objc_exception_throw + 48
    2  CoreFoundation          0x00007fff20431c48 __CFExceptionProem + 0
    3  CoreFoundation          0x00007fff204274cf forwarding + 1455
```


1. 为什么要用 XCache

Pod 二进制污染

```
7
8 #import "AClass.h"
9
10 @implementation AClass
11
12 + (void)func {
13     [BClass method1:BValueMacro];
14 }
15
16 @end
17
```

Thread 1: "+[BClass method1:]: unrecognized selector sent to class 0x10e3f2160"

Exception = (NSException *) "+[BClass method1:]: unrecognized selector sent to class 0x10e3f2160"

self = (const Class) AClass

*** Terminating app due to uncaught exception 'NSInvalidArgumentException', reason: '+[BClass method1:]: unrecognized selector sent to class 0x10e3f2160'

terminating with uncaught exception of type NSException

CoreSimulator 757.5 - Device: iPod touch (7th generation) (72717C5D-A678-4F1E-9185-061E43FD0F1E)

- Runtime: iOS 14.5 (18E182) - DeviceType: iPod touch (7th generation)

2021-06-09 14:10:40.977767+0800 PodBinaryPollution[86404:131751714] +[BClass method1:]: unrecognized selector sent to class 0x10e3f2160

2021-06-09 14:10:40.978806+0800 PodBinaryPollution[86404:131751714] *** Terminating app due to uncaught exception 'NSInvalidArgumentException', reason: '+[BClass method1:]: unrecognized selector sent to class 0x10e3f2160'



确保稳定性的前提下，迭代构建加速

XCACHE 工作机制 & 实践





2. XCache 工作机制 & 实践

XCache 的优劣势

- Xcode 增量编译
 - 代码文件粒度的缓存
 - 代码间的依赖分析

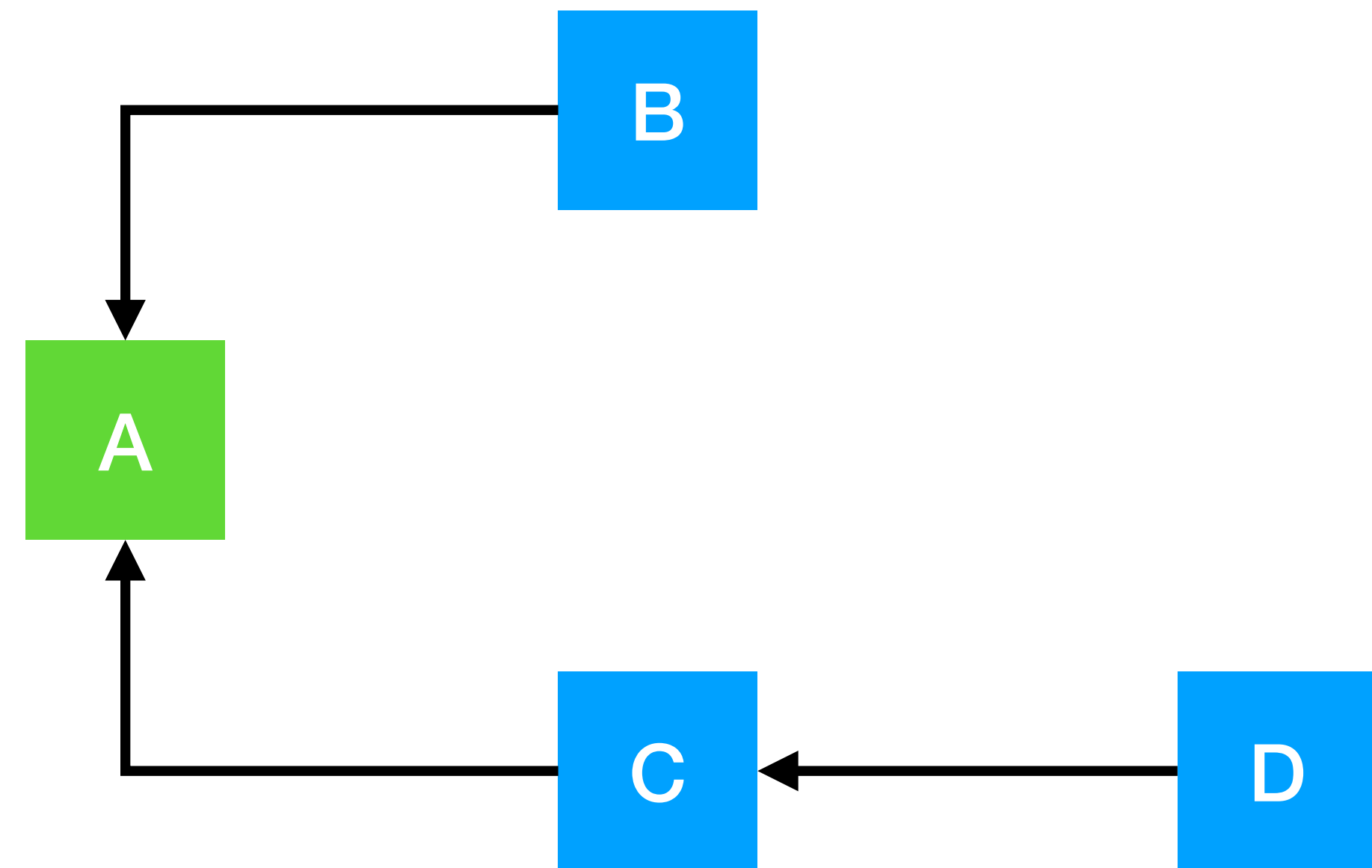


- ✓  **Prepare build**
 - ✓ Workspace PodBinaryPollution | Scheme PodBinaryPollution | Destination iPod touch (7th generation)
 - Using new build system
 - Building targets in parallel
 - Planning build
 - Analyzing workspace
 - Using build description from memory
 - Build preparation complete
- ✓  **Build target PodBinaryPollution**
 - ✓ Project PodBinaryPollution | Configuration Debug | Destination iPod touch (7th generation) | SDK Simulator - iOS 14.5
 - > ✓ Run custom shell script '[CP] Embed Pods Frameworks' 0.2 seconds
 - ✓ Sign PodBinaryPollution.app 0.1 seconds
 - ✓ **Build succeeded 2021/6/18, 2:16 PM 0.3 seconds**
 - ✓ No issues

2. XCache 工作机制 & 实践

XCache 的优劣势

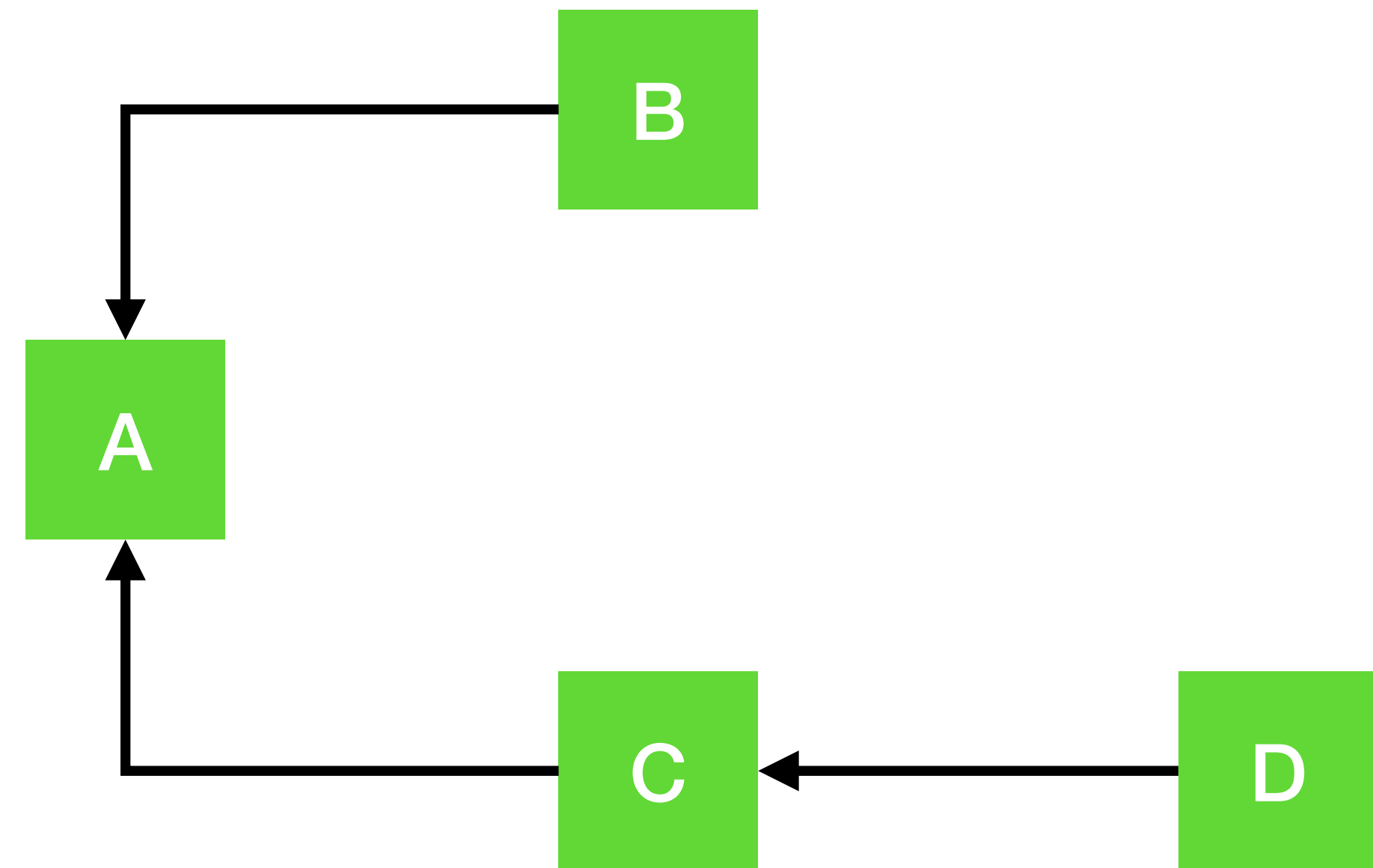
- Xcode 增量编译
 - 代码文件粒度的缓存
 - 代码间的依赖分析
 - 🙅 不分析实际的引用情况



2. XCache 工作机制 & 实践

XCache 的优劣势

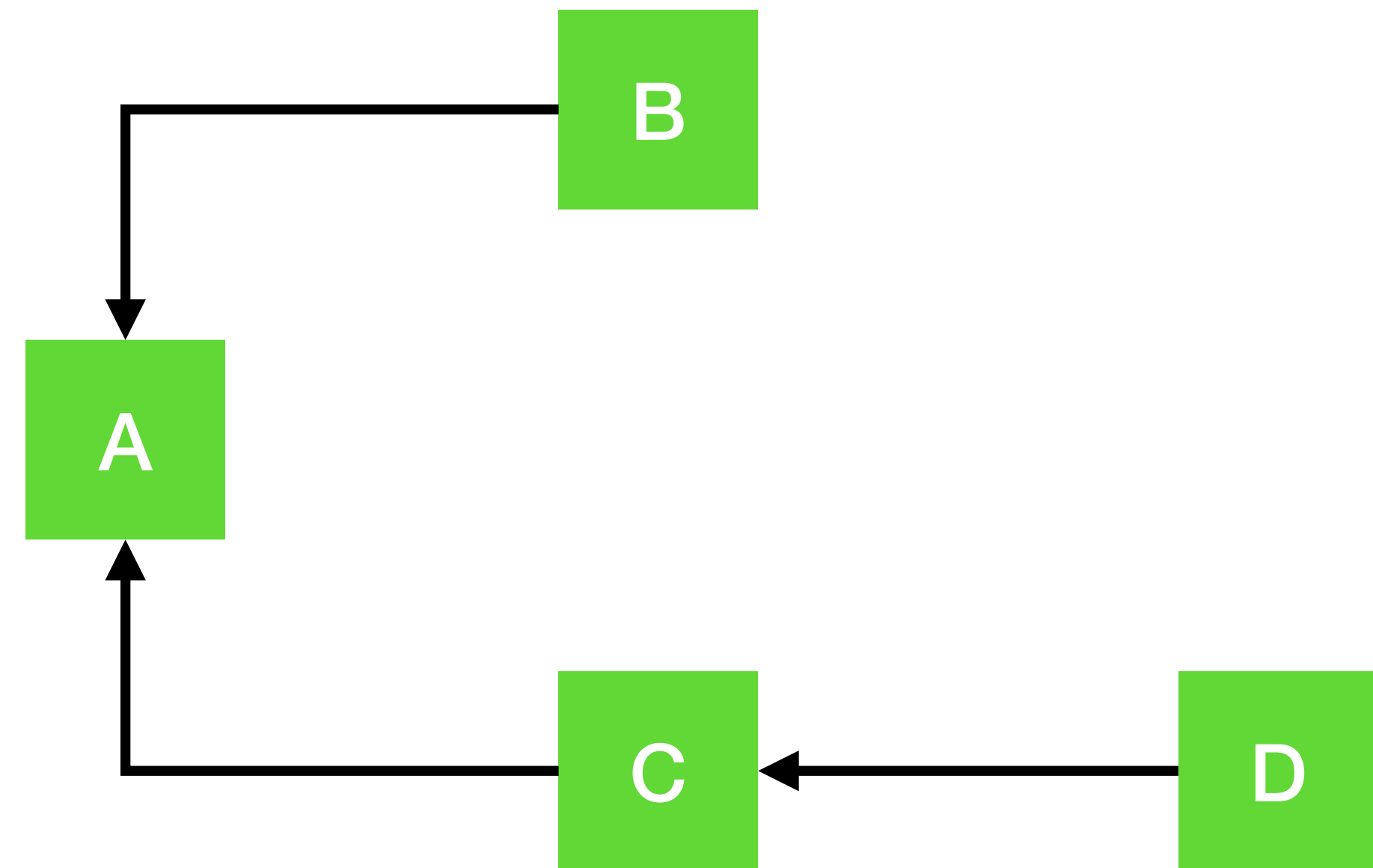
- Xcode 增量编译
 - 代码文件粒度的缓存
 - 代码间的依赖分析
 - 不存在 Pod 二进制污染问题



2. XCache 工作机制 & 实践

XCache 的优劣势

- Xcode 增量编译
 - 代码文件粒度的缓存
 - 代码间的依赖分析
 - 不存在 Pod 二进制污染问题
 - ⚠ 单份文件缓存



2. XCache 工作机制 & 实践

如何启用 XCache 快速出包

xcodebuild archive +

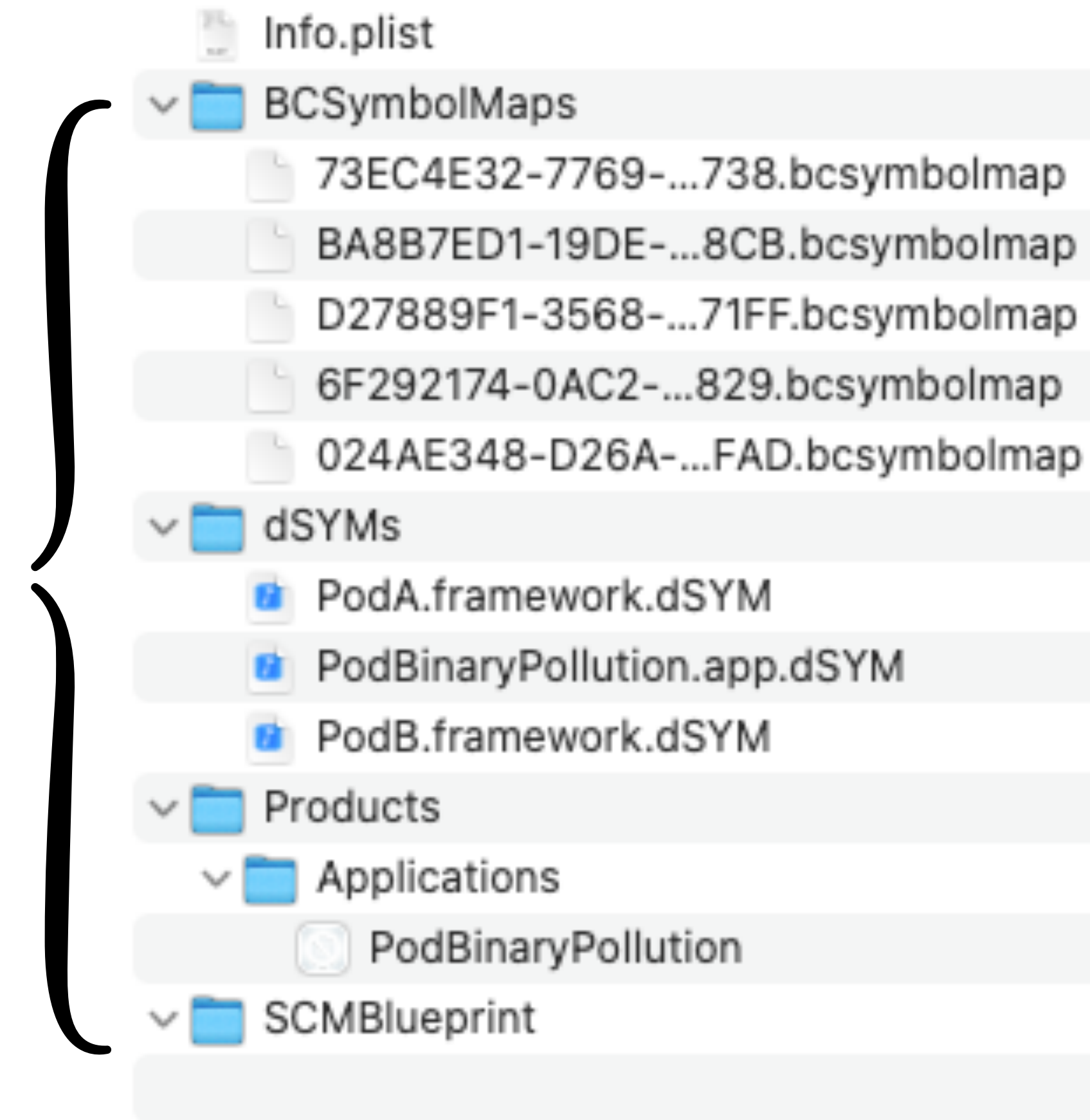


xcworkspace

=



xcarchive



2. XCache 工作机制 & 实践

如何启用 XCache 快速出包

xcodebuild archive +



xcworkspace

=



xcarchive

xcodebuild **export** +



xcarchive

=



ipa

2. XCache 工作机制 & 实践

如何启用 XCache 快速出包

xcodebuild archive +



xcworkspace

=



xcarchive

XCache



xcodebuild build +



xcarchive

=



xxx.app

XCache



2. XCache 工作机制 & 实践

如何启用 XCache 快速出包



2. XCache 工作机制 & 实践

缓存工作机制

1. 当工程路径发生变更后，将会重编译（文件夹 hash）

```
> PodBinaryPollution-aysqvwioppslfaeehgsjapvsijiv  
> PodBinaryPollution-efajryqcykfoblczwojfrcnjhudr  
> PodBinaryPollution-fxflmwjfpxcoykeebmrrgmgmaykc
```



2. XCache 工作机制 & 实践

缓存工作机制

1. 当工程路径发生变更后，将会重编译（文件夹 hash）

- CI 构建时保留工作目录，增量更新

```
> PodBinaryPollution-aysqvwioppslfaeehgsjapvsijiv
> PodBinaryPollution-efajryqcykfoblczwojfrcnjhudr
> PodBinaryPollution-fxflmwjfpkcoykeebmrrgmngmaykc
```



2. XCache 工作机制 & 实践

缓存工作机制

1. 当工程路径发生变更后，将会重编译（文件夹 hash）
 - CI 构建时保留工作目录，增量更新
2. 每个文件的修改时间戳发生变更，或者内容变更后，将会重编译
 - ⚠️ git 不会保存文件的 metainfo



缓存工作机制

- [illegible]



2. XCache 工作机制 & 实践

缓存工作机制

1. 当工程路径发生变更后，将会重编译（文件夹 hash）
2. 每个文件的修改时间戳发生变更，或者内容变更后，将会重编译
3. 当构建参数发生变更后，将会重编译



2. XCache 工作机制 & 实践

缓存工作机制

1. 当工程路径发生变更后，将会重编译（文件夹 hash）
2. 每个文件的修改时间戳发生变更，或者内容变更后，将会重编译
3. 当构建参数发生变更后，将会重编译
 - xcproject 目前由 cocoapods 生成，因此不合理的参数变更可通过修改 cocoapods 得到控制



2. XCache 工作机制 & 实践

缓存工作机制

集成状态	集成版本号	集成的分支	集成人	集成时间	用时
● 成功	6.2.2.0  	 更多	master 	2021-06-03 14:50:28	5min46s
● 成功	6.2.2.0 		master 	2021-06-03 14:40:43	33min19s
● 成功	6.2.2.0  	 更多	master 	2021-06-03 14:28:59	6min1s
● 成功	6.2.2.0 	 更多	master 	2021-06-03 14:21:15	42min59s
● 成功	6.2.2.0  	 更多	master 	2021-06-03 14:16:24	5min25s
● 成功	6.2.2.0  	 更多	master 	2021-06-03 14:08:43	5min52s
● 成功	6.2.2.0  	 更多	master 	2021-06-03 13:38:54	5min29s
● 成功	6.2.2.0  	 更多	master 	2021-06-03 13:22:26	4min59s

两个问题 & 解决方案

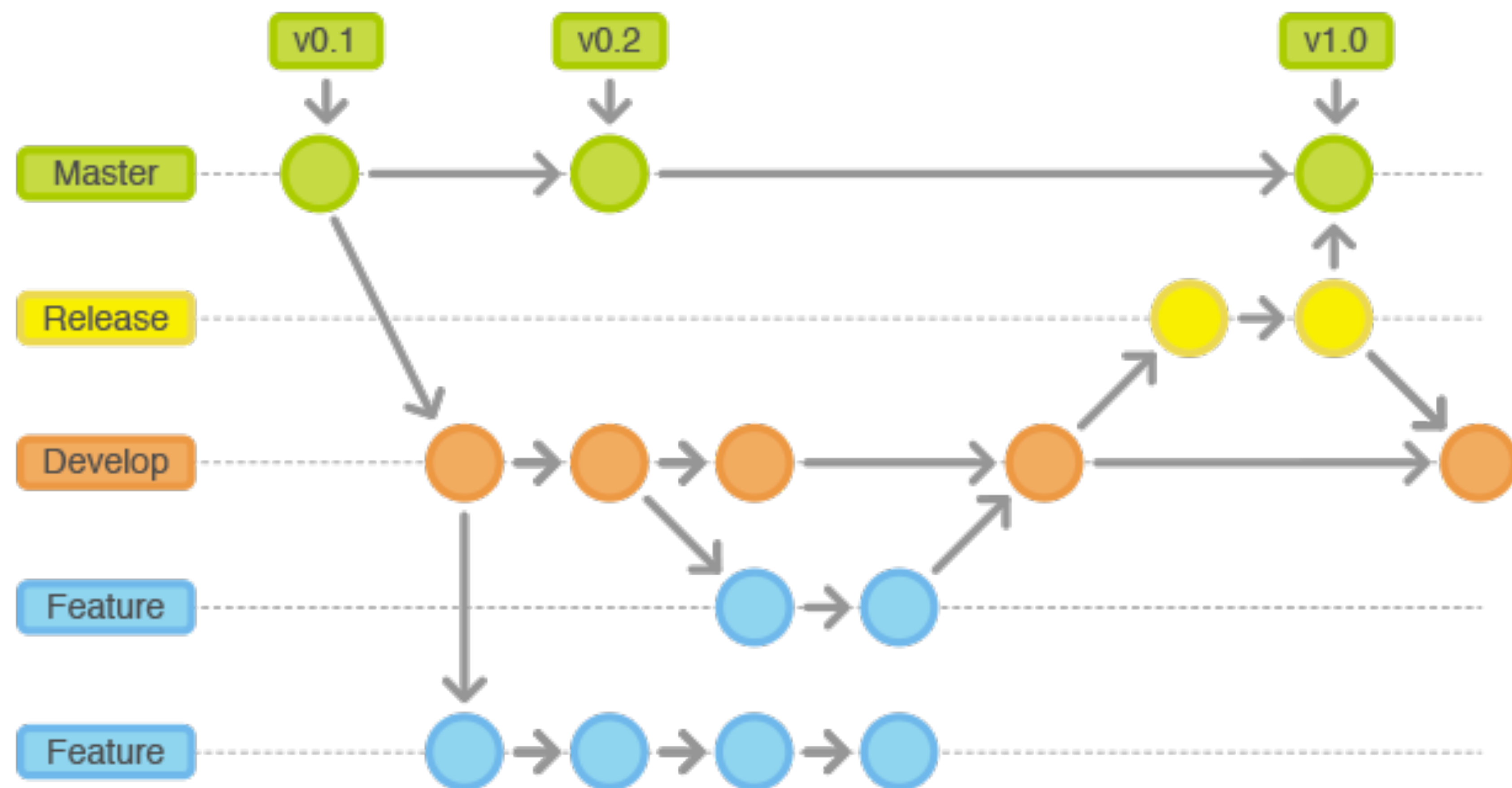


3. 两个问题 & 解决方案

多分支构建加速支持

1. 集成构建

2. 测试打包



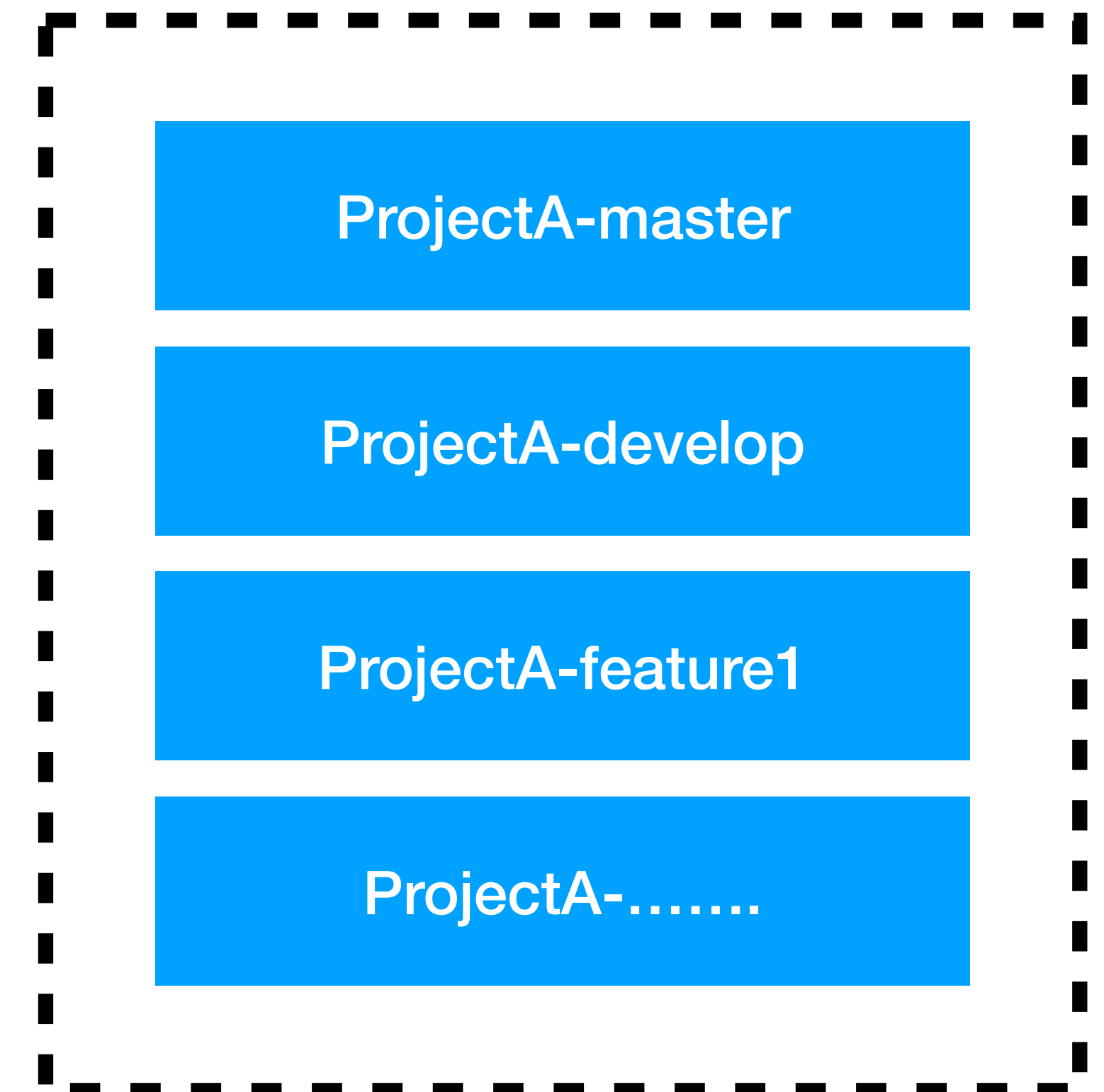
3. 两个问题 & 解决方案

多分支构建加速支持

1. 保留多份工作目录，对应多份项目缓存（LRU）

- 磁盘空间利用率不高
- 回退较老版本编译会缓存 miss

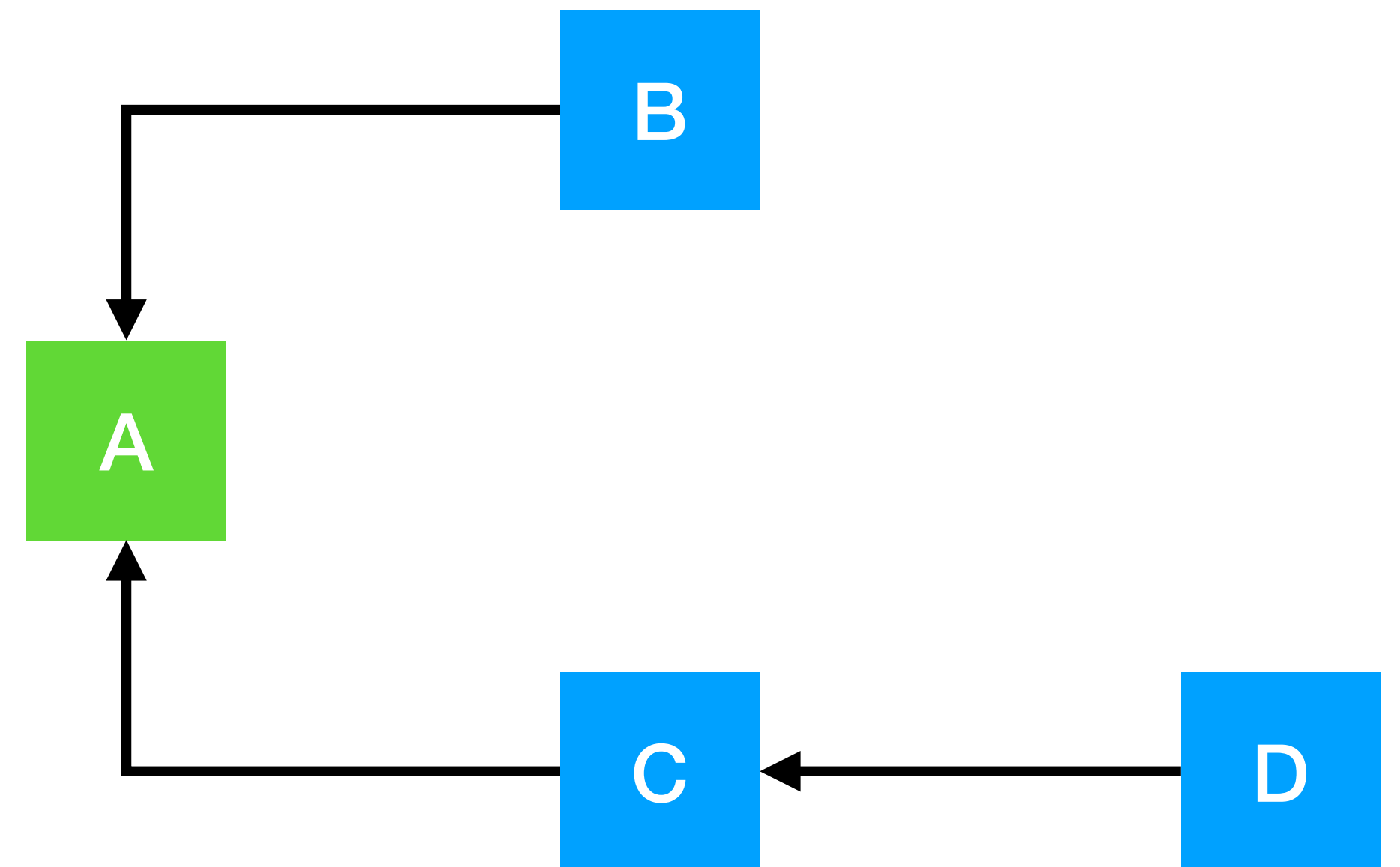
LRU Cache



3. 两个问题 & 解决方案

多分支构建加速支持

1. 保留多份工作目录，对应多份项目缓存（LRU）
2. 模仿 XCache 逻辑，安全启用 Pod 二进制缓存



3. 两个问题 & 解决方案

多分支构建加速支持

1. 保留多份工作目录，对应多份项目缓存（LRU）
2. 模仿 XCache 逻辑，安全启用 Pod 二进制缓存
 - 源码逐行扫描 `#include` `#import`，记录并分析引用关系
 - ⚠ 坑： `#ifdef`， `has_include`，多份相同头文件时
 - 字符串解析代码的局限性，无法 100% 保证分析结果准确

3. 两个问题 & 解决方案

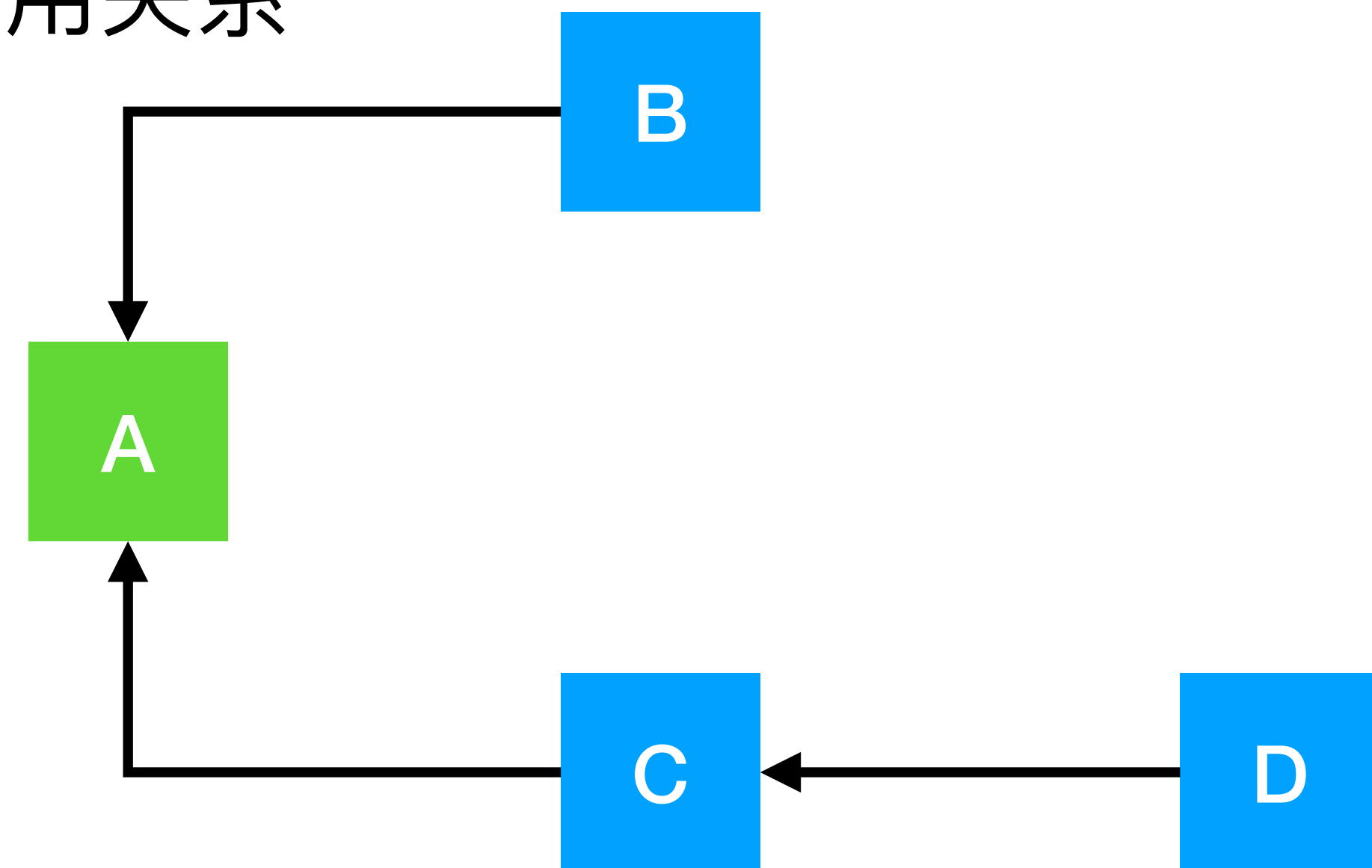
多分支构建加速支持

1. 保留多份工作目录，对应多份项目缓存（LRU）
2. 模仿 XCache 逻辑，安全启用 Pod 二进制缓存
 - 源码逐行扫描 `#include` `#import`，记录并分析引用关系
 - AST 解析头文件依赖关系
 - 实现成本，编译参数获取，执行效率

3. 两个问题 & 解决方案

多分支构建加速支持

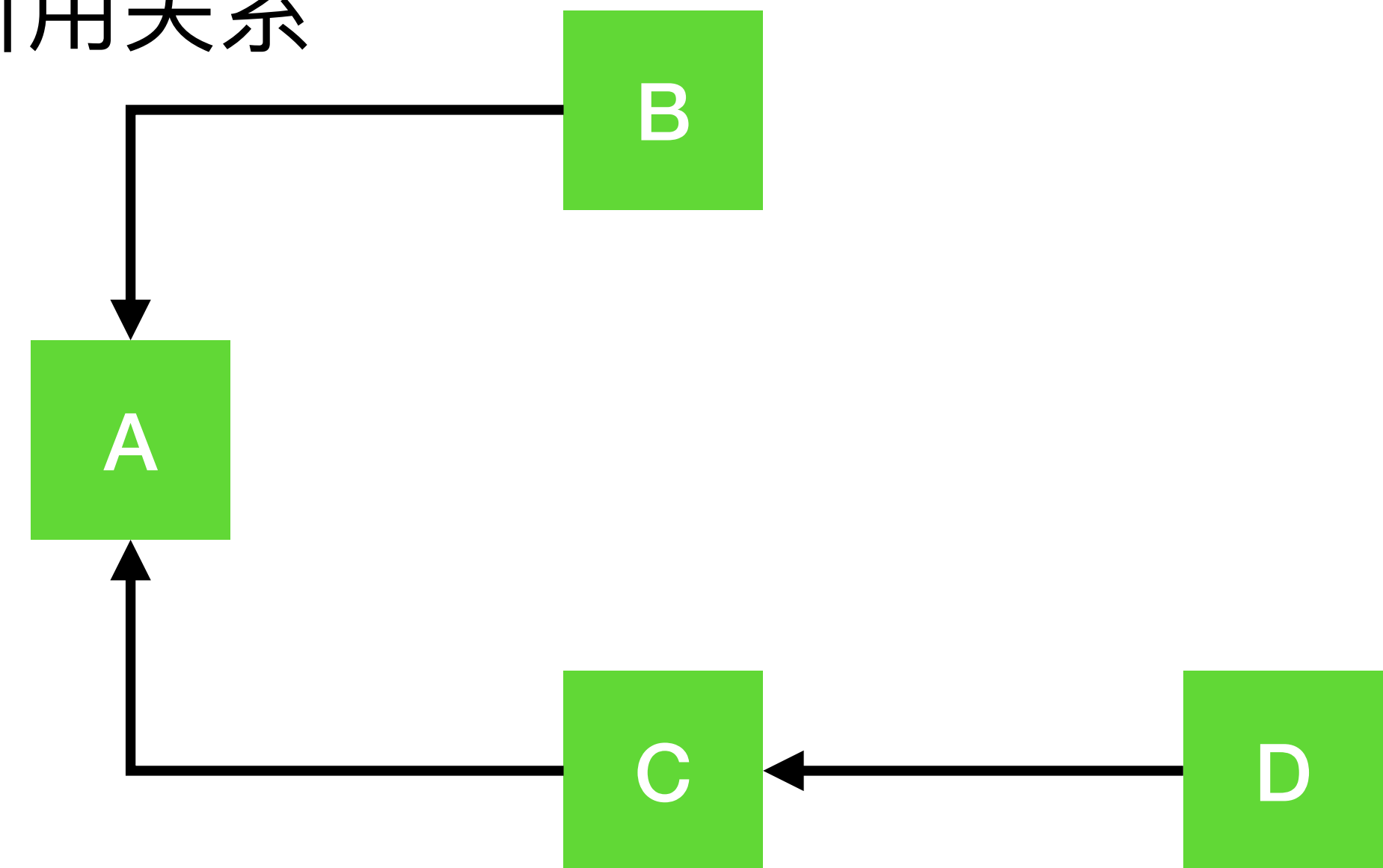
1. 保留多份工作目录，对应多份项目缓存（LRU）
2. 模仿 XCache 逻辑，安全启用 Pod 二进制缓存
 - 源码逐行扫描 `#include` `#import`，记录并分析引用关系
 - AST 解析头文件依赖关系
 - ⚠️ Pod 单个 tag 可能需要多份二进制存储



3. 两个问题 & 解决方案

多分支构建加速支持

1. 保留多份工作目录，对应多份项目缓存（LRU）
2. 模仿 XCache 逻辑，安全启用 Pod 二进制缓存
 - 源码逐行扫描 #include #import，记录并分析引用关系
 - AST 解析头文件依赖关系
 - ⚠️ Pod 单个 tag 可能需要多份二进制存储
 - ✅ 源码编译 Pod 仍然享受 XCache 加速收益



3. 两个问题 & 解决方案

AppStore 出包

xcodebuild archive +



xcworkspace

=



xcarchive

增量编译



xcodebuild build +



xcworkspace

=



xxx.app

增量编译



3. 两个问题 & 解决方案

AppStore 出包

xcodebuild build +



xcworkspace

=



xxx.app

修改 xcproj, 将所有源码
替换为 build 产物静态库

xcodebuild archive +



Pod binaries

=



xcarchive

Summary



4. Summary

第一阶段

1. 保留 CI 构建工作目录
2. ipa 打包脚本

第二阶段

1. 代码文件的时间戳管理
2. 多份项目缓存 LRU
3. AppStore 出包脚本（可选）



第三阶段

1. 安全启用 Pod 二进制缓存
 - 源码逐行扫描
 - AST 解析

Q&A