得物iOS工程的演进

得物iOS架构组

工程演进规划


工程化
组件化
容器化


工程化
组件化
容器化


工程化
组件化
容器化


工程化
组件化
容器化


工程化
组件化
容器化
工程健康

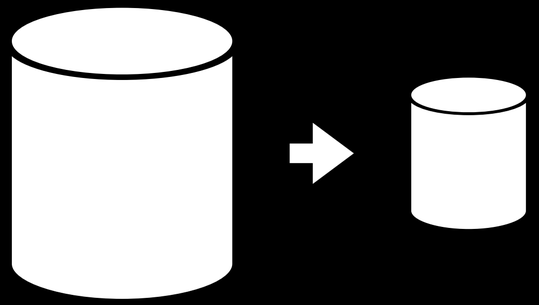

工程化
组件化
容器化
工程健康

包体积大小治理

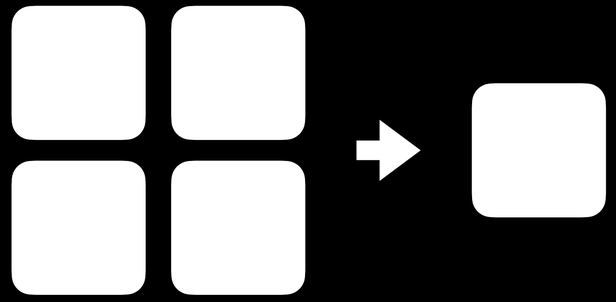
包体积大小 = 资源 + 代码

包体积大小 = 资源 + 代码

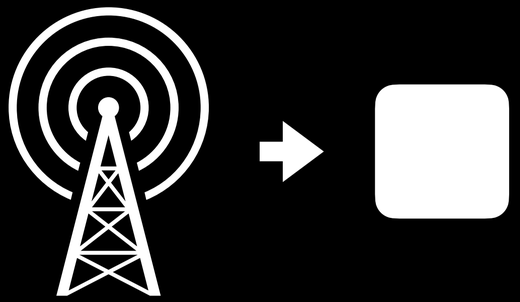
资源的治理



资源压缩

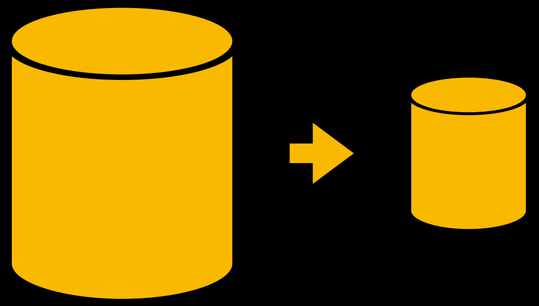


重复资源

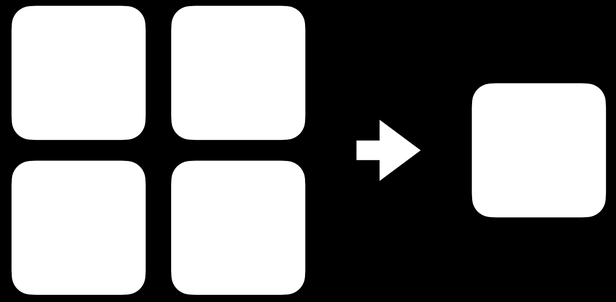


资源下发

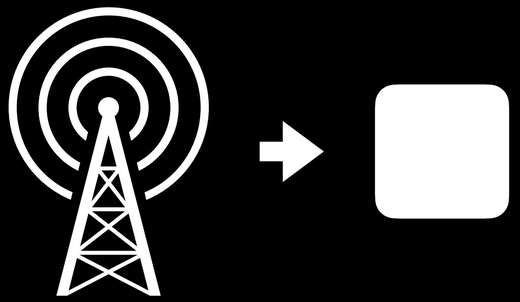
资源的治理



资源压缩

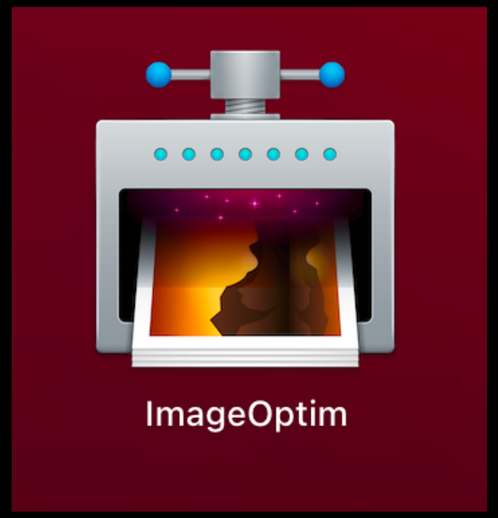


重复资源



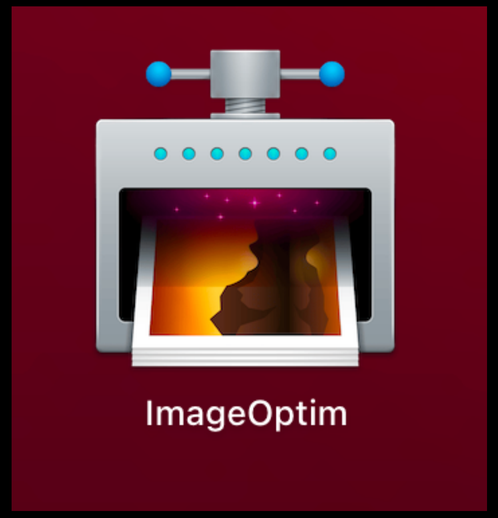
资源下发

无损



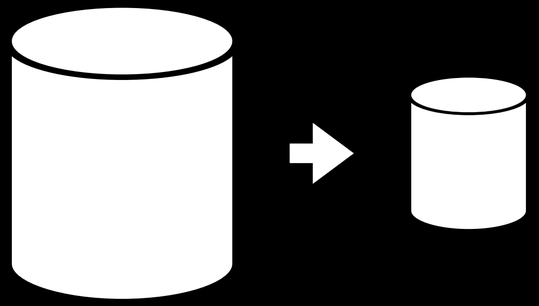
无收益

80%
有损

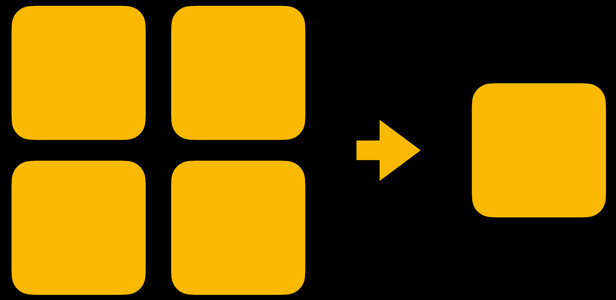


收益10M

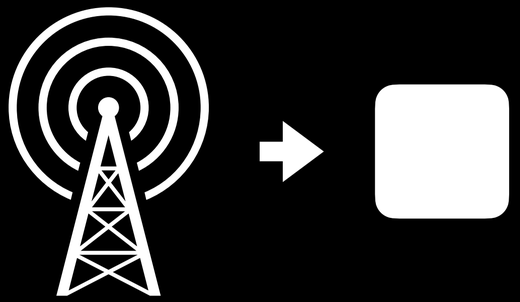
资源的治理



资源压缩



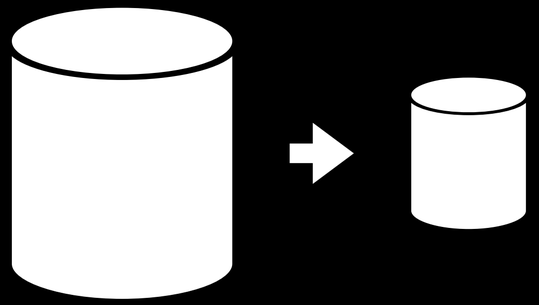
重复资源



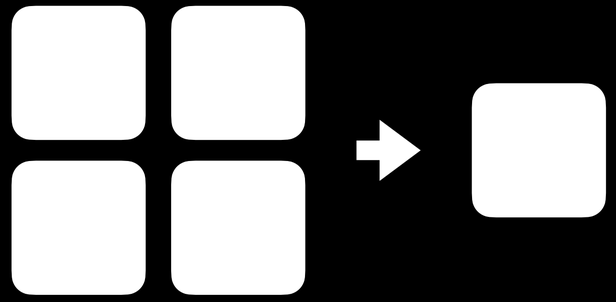
资源下发


脚本扫描
手工合并
脚本合并
机器学习相似资源
手工合并

资源的治理



资源压缩



重复资源

资源下发

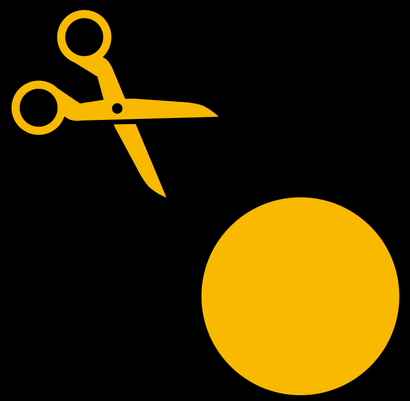

苹果自带资源下发
自建资源下发平台
资源增量下发


苹果自带资源下发
自建资源下发平台
资源增量下发

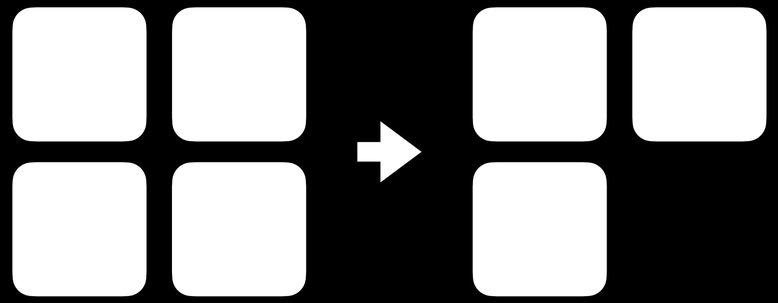

苹果自带资源下发
自建资源下发平台
资源增量下发

包体积大小 = 资源 + 代码

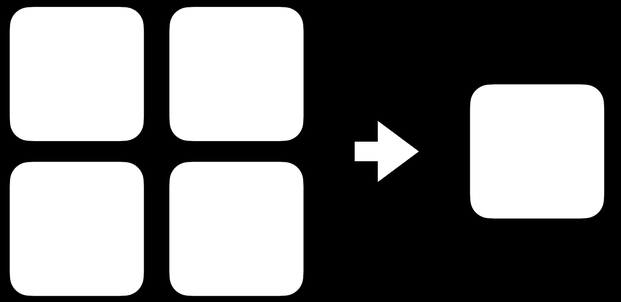
代码的治理



三方库裁剪



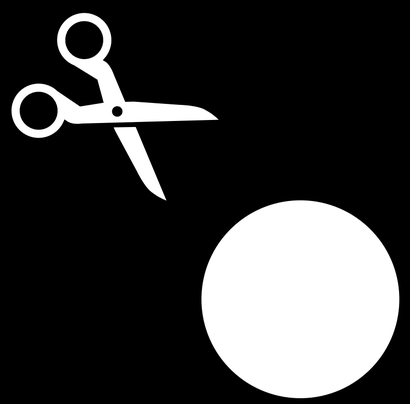
无用代码



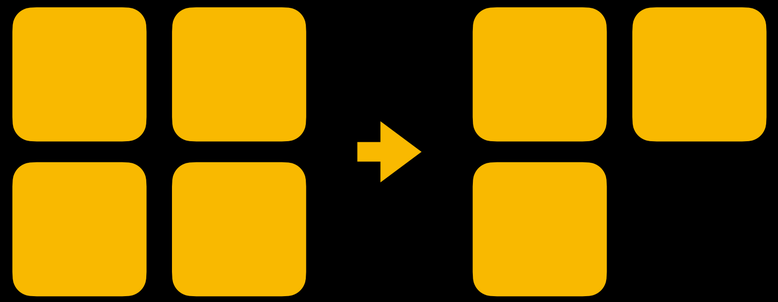
重复代码


解析
LINK-MAP
排序
识别库作用
功能重复下线

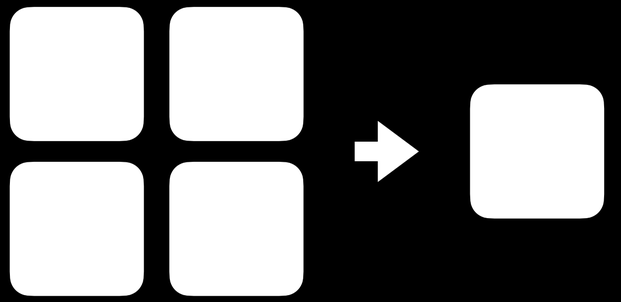
代码的治理



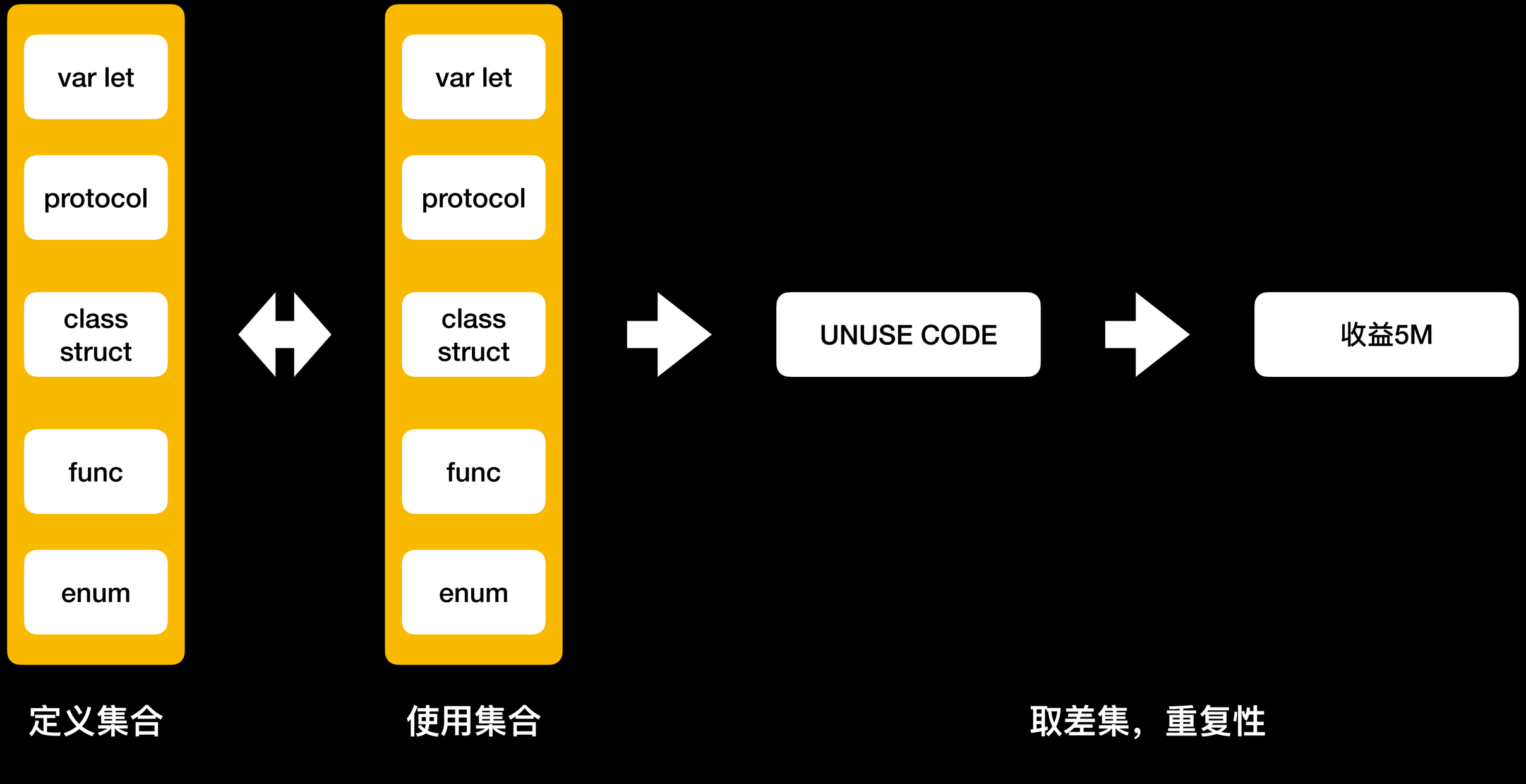
三方库裁剪



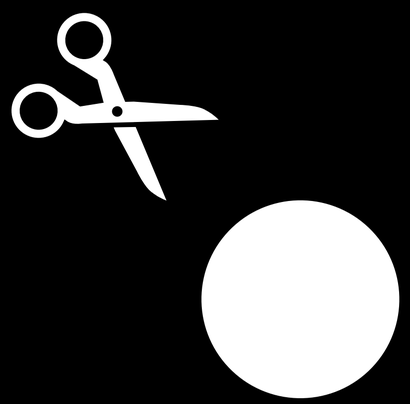
无用代码



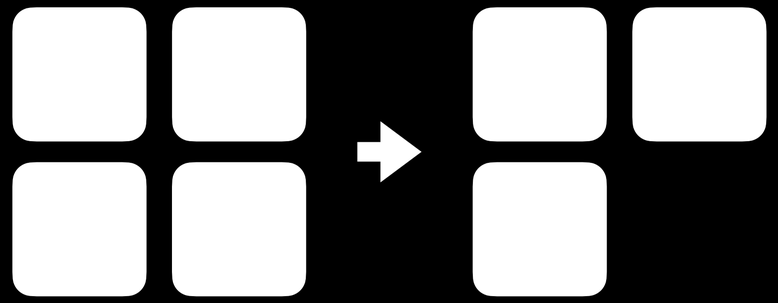
重复代码


var let
protocol
class
struct
func
enum
var let
protocol
class
struct
func
enum
UNUSE CODE
收益5M
定义集合
使用集合
取差集，重复性

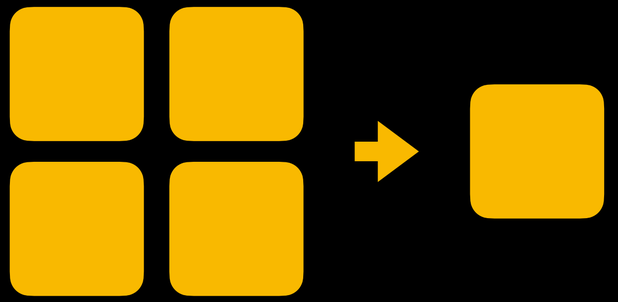
代码的治理



三方库裁剪



无用代码



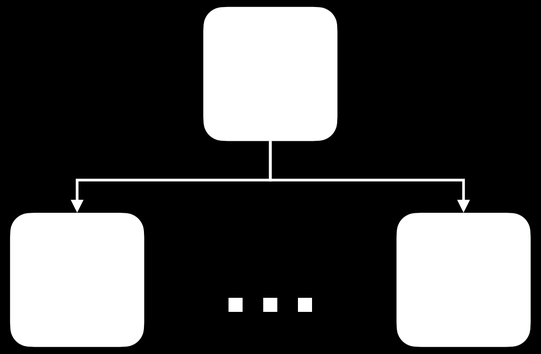
重复代码

最终成果：350M -> 250M

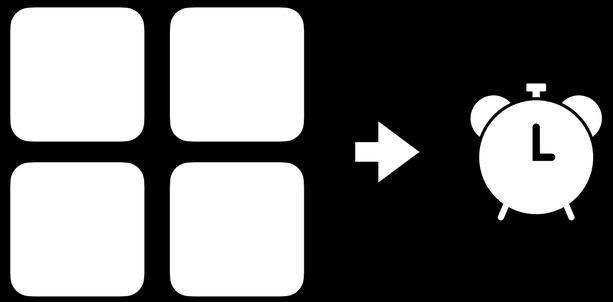
减少100M

工程代码治理

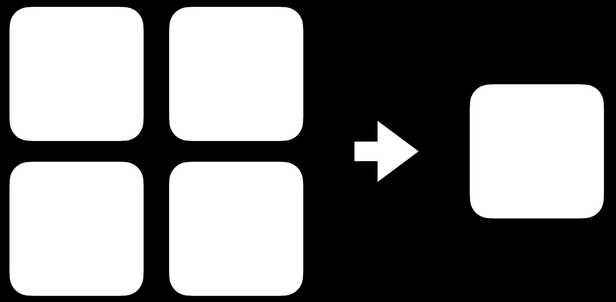
代码的治理



长依赖链组件



长编译耗时组件



无用、重复代码

Crash治理


纯人工
半自动
全自动


纯人工
半自动
全自动
手工跟进Bugly
手工区分业务线
新增crash落表跟进


纯人工
半自动
全自动
手工跟进Bugly
脚本爬取Bugly
手工区分业务线
自动识别业务线
新增crash落表跟进
自动分派落表

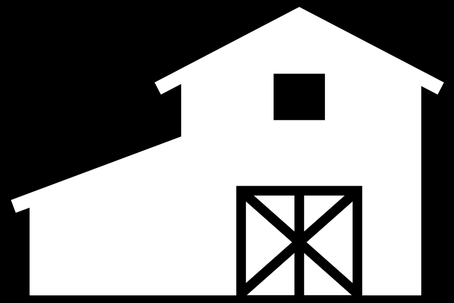

纯人工
半自动
全自动
手工跟进Bugly
脚本爬取Bugly
自动收集、解析、告警
手工区分业务线
自动识别业务线
自动实时识别并分派
新增crash落表跟进
自动分派落表
自动跟进修复情况

启动流程治理


启动流程分组
代码插桩diff
安全模式

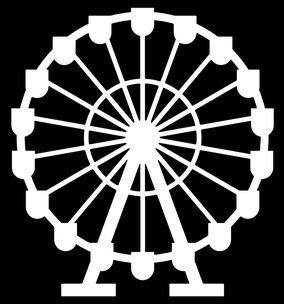

工程化
组件化
容器化
工程健康

工程化



围绕组件化的基础设施

包分发平台



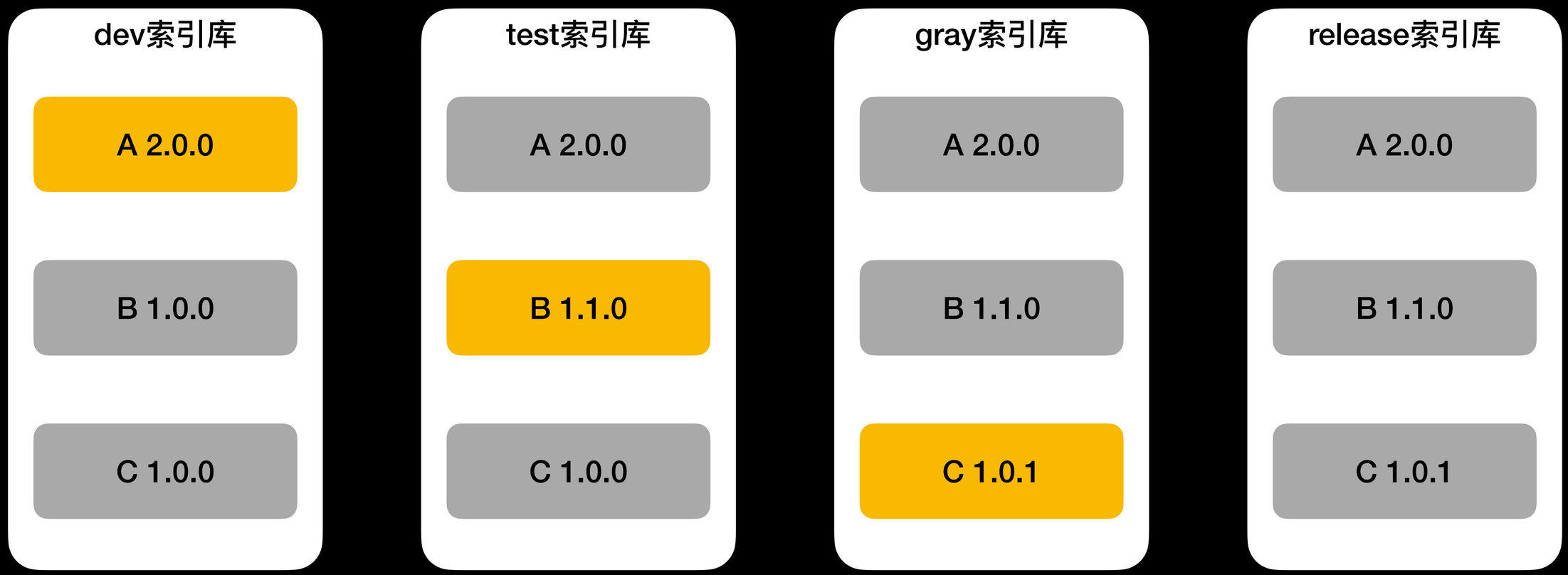
持续交付

围绕组件化的基础设施

提供脚本工具

- 组件创建脚本、组件工具
- 组件发版工具
- 二进制调试工具
- 组件上游依赖查询工具
- 编译成功节点切换工具
- 裙带源码组件切换工具

组件管理


dev索引库
A 2.0.0
B 1.0.0
C 1.0.0
test索引库
A 2.0.0
B 1.1.0
C 1.0.0
gray索引库
A 2.0.0
B 1.1.0
C 1.0.1
release索引库
A 2.0.0
B 1.1.0
C 1.0.1

为什么不在Podfile中写死组件版本?

每一次组件版本发布
就意味着修改一次版本号

总共1431个组件

平均每天组件发版100次左右

按照8h算，4.8分钟发版一次

多组件联合发版十分常见

写死组件版本是不现实的

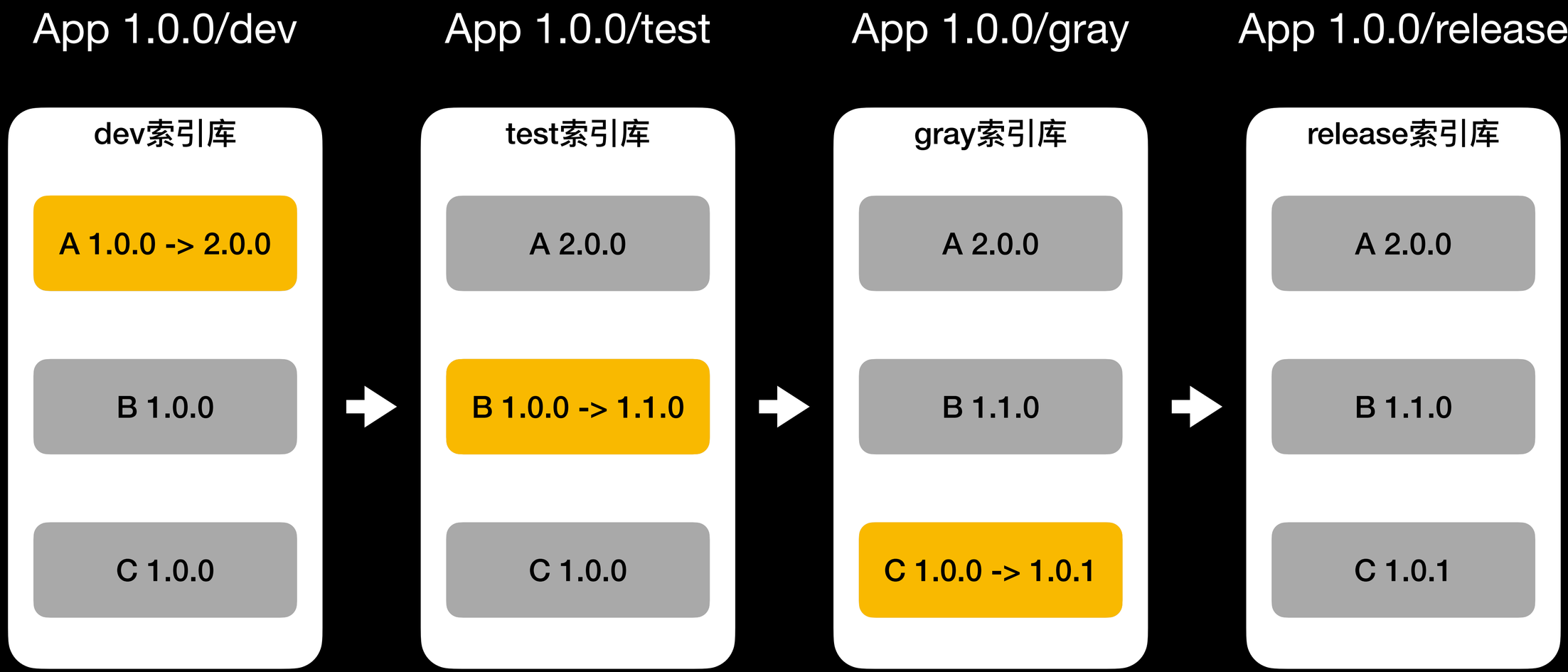
直接按照索引库管理

索引库的创建成本低、管理成本低

工程状态迁移时，只要做索引库迁移即可

不影响多版本同时开发

同一版本从开发到发布


App 1.0.0/dev
App 1.0.0/test
App 1.0.0/gray
App 1.0.0/release
dev索引库
A 1.0.0 -> 2.0.0
B 1.0.0
C 1.0.0
test索引库
A 2.0.0
B 1.0.0 -> 1.1.0
C 1.0.0
gray索引库
A 2.0.0
B 1.1.0
C 1.0.0 -> 1.0.1
release索引库
A 2.0.0
B 1.1.0
C 1.0.1

二进制

单工程独立编译制作

Pros：
1. 每次发版时都完成了二进制制作
2. 可靠性高

Cons：
1. 对于一些大组件制作二进制时间，取决于组件的编译时间
2. 测试包体积变大N倍

全工程编译产物制作

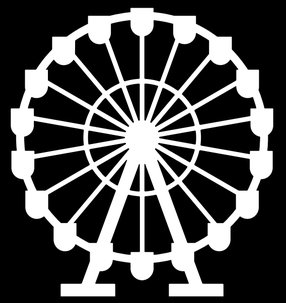
Pros：
1. 编译一次50min，出齐1400个二进制包

Cons：
1. 只能一批一批制作（1h制作一次）
2. 旧版本组件二进制，新版本组件只能源码会偶发ARC不对齐的情况，进而运行时崩溃

基于Oolong的增量制作

Pros：
1. 编译一次20min，出齐1400个二进制包
2. 彻底解决ARC不对齐的情况

Cons：
1. 只能一批一批制作（20min一次）



持续交付

多人同时开发、多组件同时发布
编译稳定性差，出包困难

Xcode Server定时打包

Pros：
1. 能够避免编译错误长时间不被解决的问题

Cons：
1. 间隔时间长，每轮相隔1h
2. 不能稳定保证工程师端编译成功

记录编译成功节点
提供boom脚本进行切换

Pros：
1. 能够避免编译错误长时间不被解决的问题
2. 能够最大限度保证工程师端编译成功

Cons：
1. 编译成功的节点时间上会落后近1h
2. 二进制的各种偶发问题

多机车轮打包
编译错误跟进、熔断机制

Pros：
1. 能够更及时发现编译错误，每轮10min
2. 能够保证工程师端编译成功

Cons：
1. 费机器


1
dev分支
2
test分支
gray分支
3
dev分支
4
dev二进制
test二进制
5
业务大组件1
业务大组件2
6
test分支
gray分支
7
dev分支
8
test分支
gray分支


iOS小助手 机器人 iOS相关日常维护，例如：编译告警、指定处理人、iOS内部公告、新人引导等 14:22
【构建失败】【二进制】origin/4.77.0/test
构建信息
版本：4.77.0
构建：319
分支: origin/4.77.0/test
构建: 点击查看
开始时间: 08-31 14:03:32
组件列表: 点击查看
PodLock: 点击查看
查看所有编译
错误信息
编译开始于: 2021-08-31 14:11:05.352
编译错误共1个
- Swift Compiler Error: Value of type 'CTMediator' has no member
[redacted]
修复建议
▲@[redacted]在
49minutesago提交的9f850b6749节点
我来处理

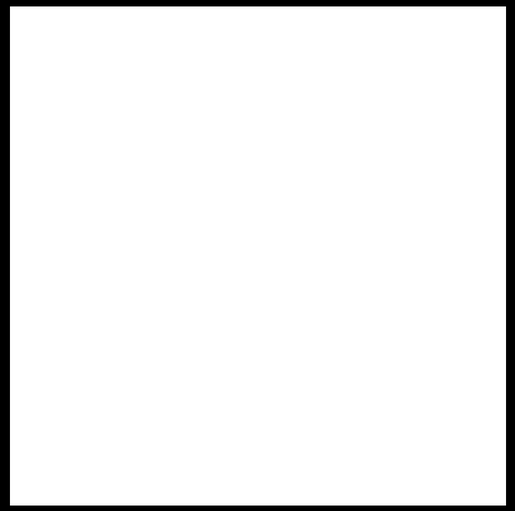

iOS小助手 机器人 iOS相关日常维护，例如：编译告警、指定处理人、iOS内部公告、新人引导等
【修复中】【主工程】origin/4.77.0/test
构建信息
版本：4.77.0
构建：288
分支: origin/4.77.0/test
构建: 点击查看
开始时间: 08-31 15:33:59
组件列表: 点击查看
PodLock: 点击查看
查看所有编译
错误信息
编译开始于: 2021-08-31 15:42:46.605
编译错误共3个
[redacted]
修复建议
▲@[redacted]在
85minutesago提交的0c18afe节点
处理信息
处理人:
处理状态:
处理开始
我来处理
处理完成后，必须要选择错误原因，以结束流程
代码错误
编译系统错误


iOS小助手 机器人 iOS相关日常维护，例如：编译告警、指定处理人、iOS内部公告、新人引导等
【已修复】【二进制】origin/4.77.0/test
构建信息
版本：4.77.0
构建：360
分支: origin/4.77.0/test
构建: 点击查看
开始时间: 09-01 12:42:55
组件列表: 点击查看
PodLock: 点击查看
查看所有编译
错误信息
编译开始于: 2021-09-01 12:51:10.074
编译错误共5个
[redacted]
修复建议
▲@唐明 -[redacted]
inc.com)在16hoursago提交的9ea6a06节点
处理信息
处理人:
处理状态:
处理开始:
处理结束:
处理时间:

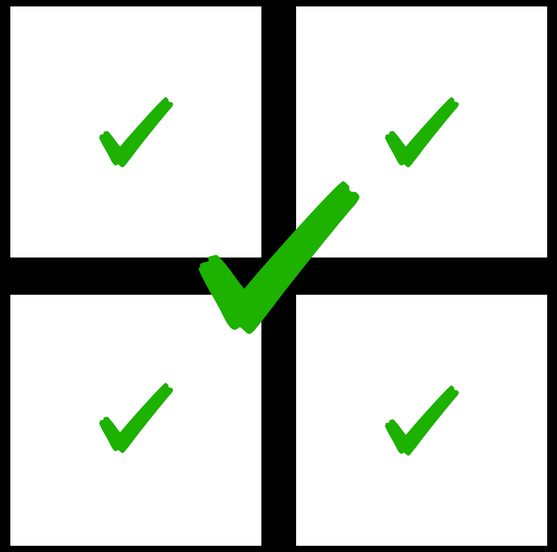

工程化
组件化
容器化
工程健康

组件化解决的是什么问题?

化整为零，各自独立



每一个部分是正确的，整体就是正确的




Input
Process
Output

IPO模型：

只要P、O正确，I一定是正确的

只要I、O正确，P一定是正确的

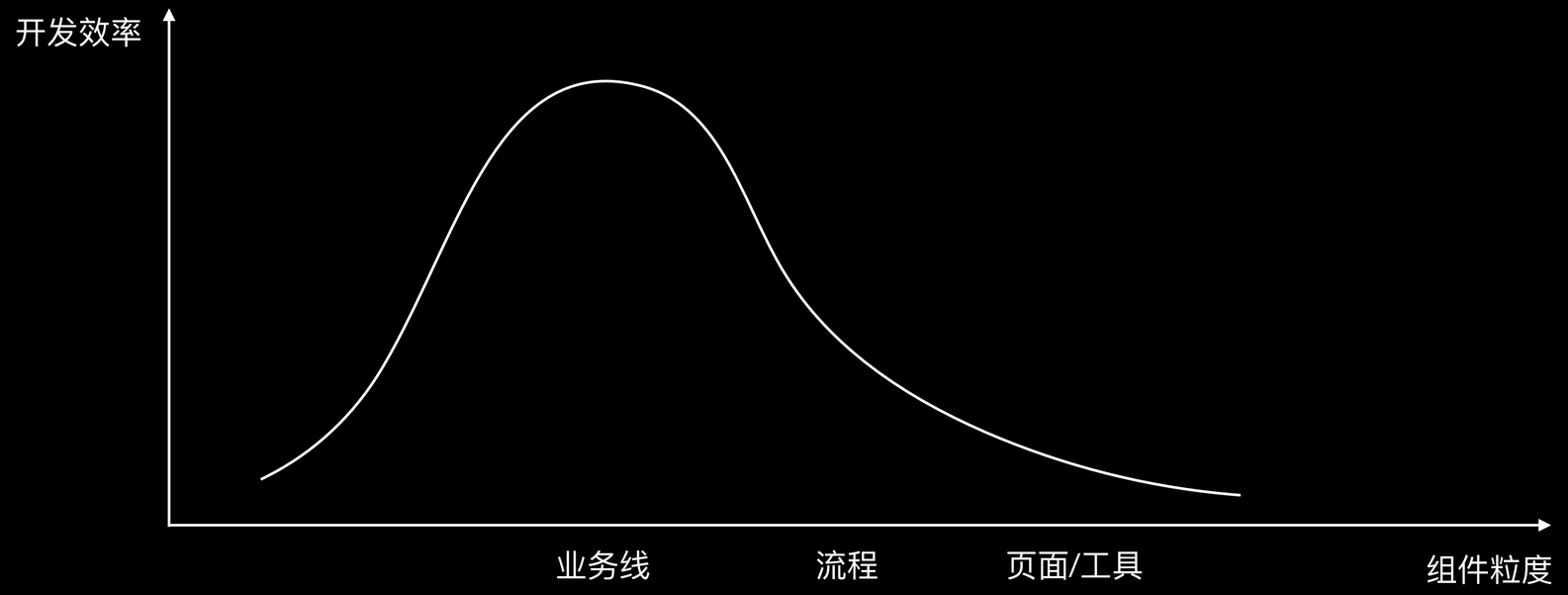
只要I、P正确，O一定是正确的

为什么不用基于注册的组件化方案?

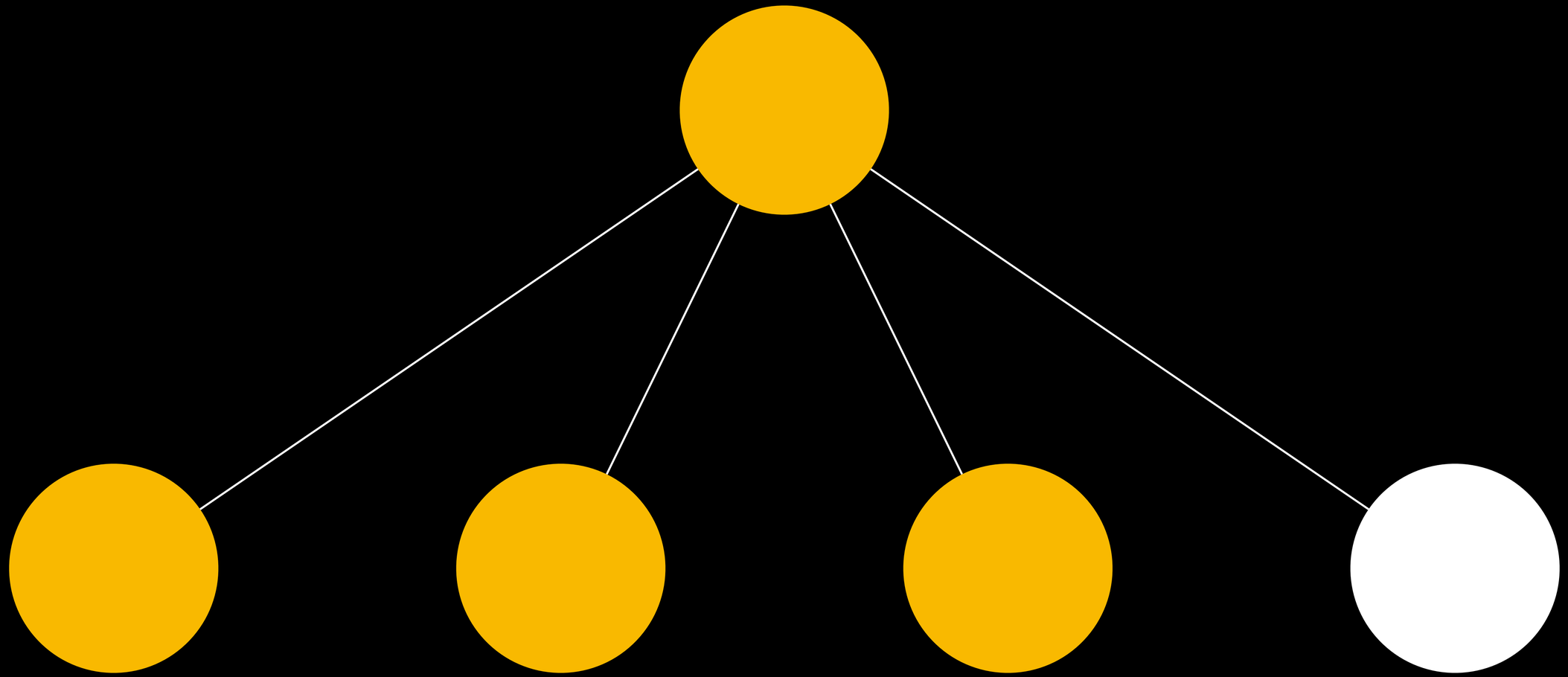
注册时序需要精心维护，否则就会导致调用失效

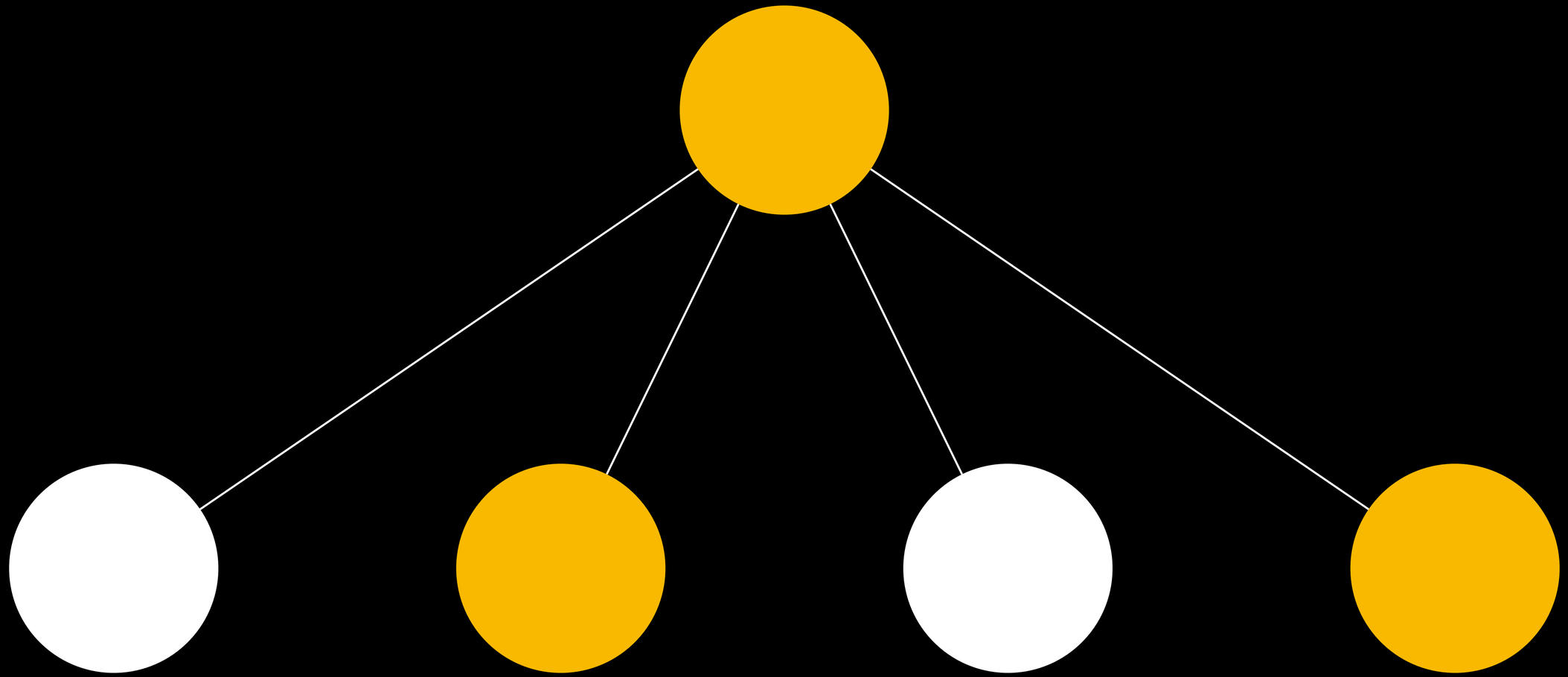
CTMediator保证只要组件存在，就一定可以调用成功

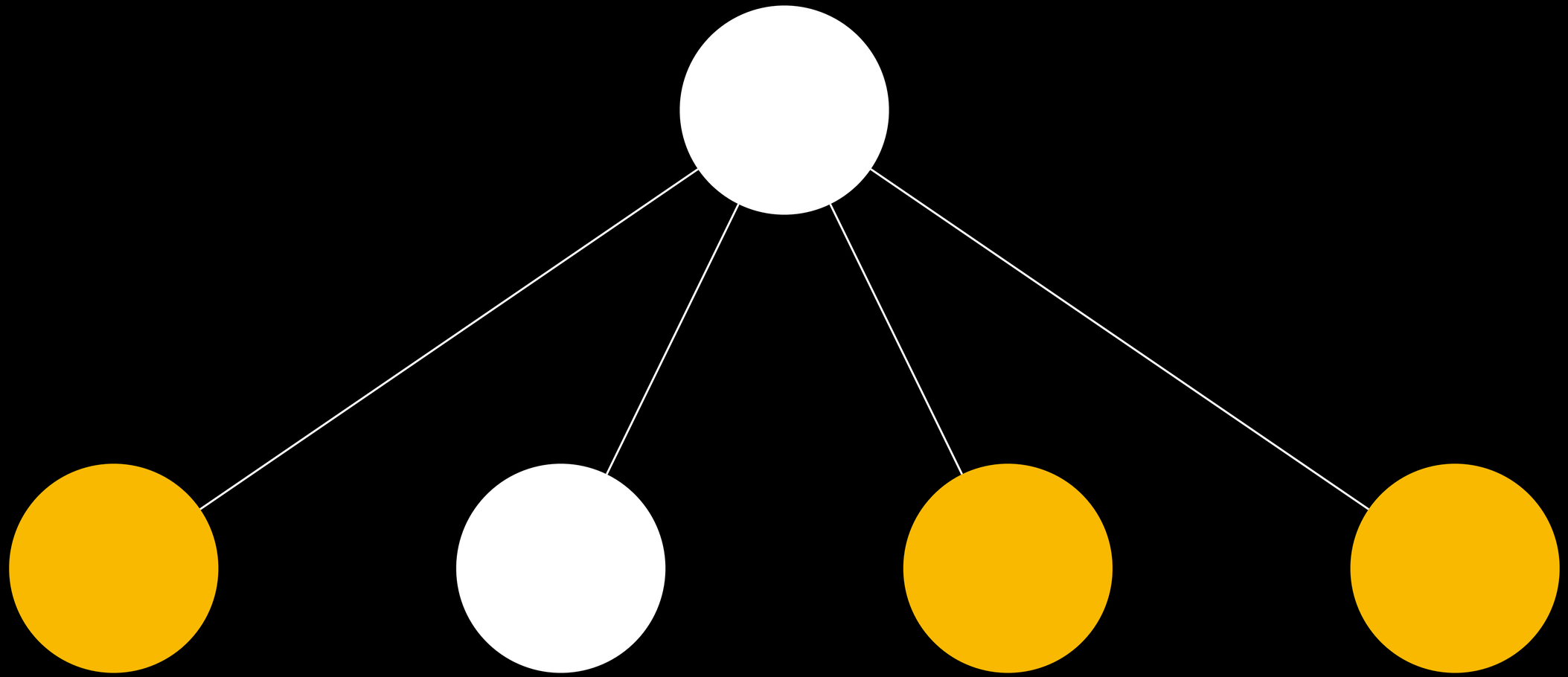
组件拆多少才好?

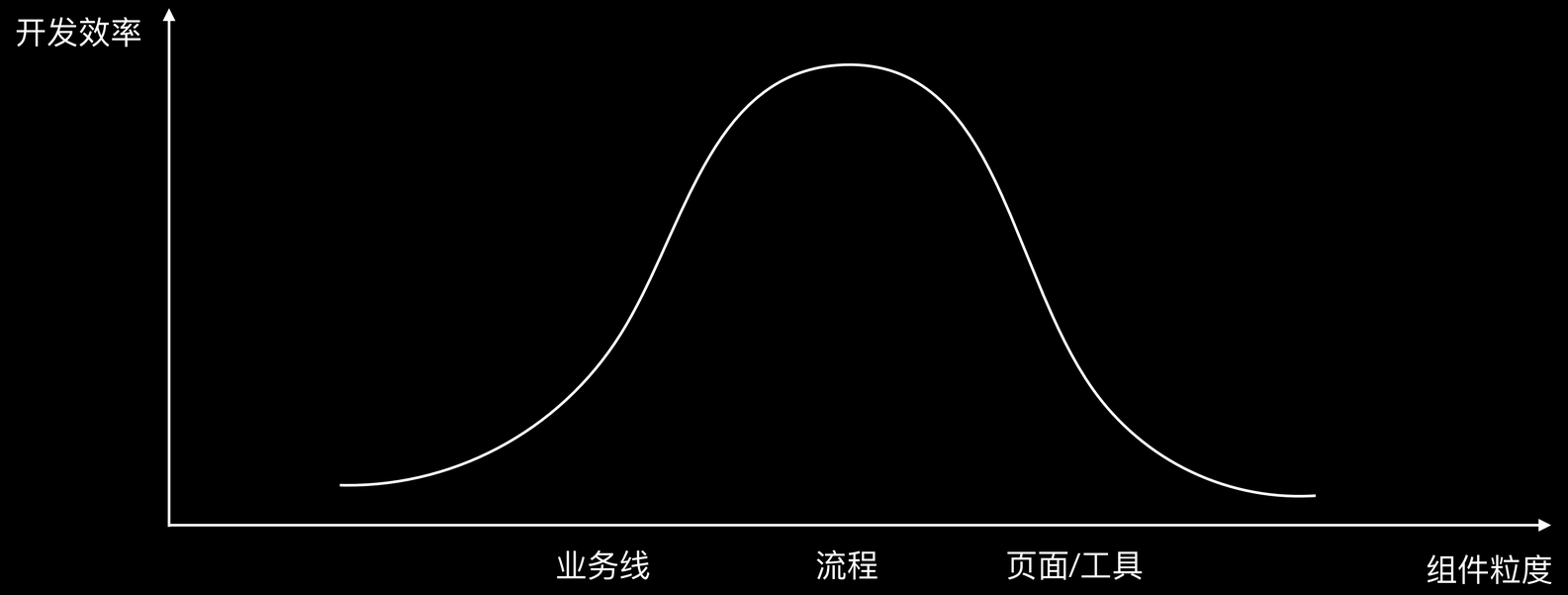


小规模工程 配套 小团队
组件要拆，但是组件数量若是多了，工作效率就反而降低

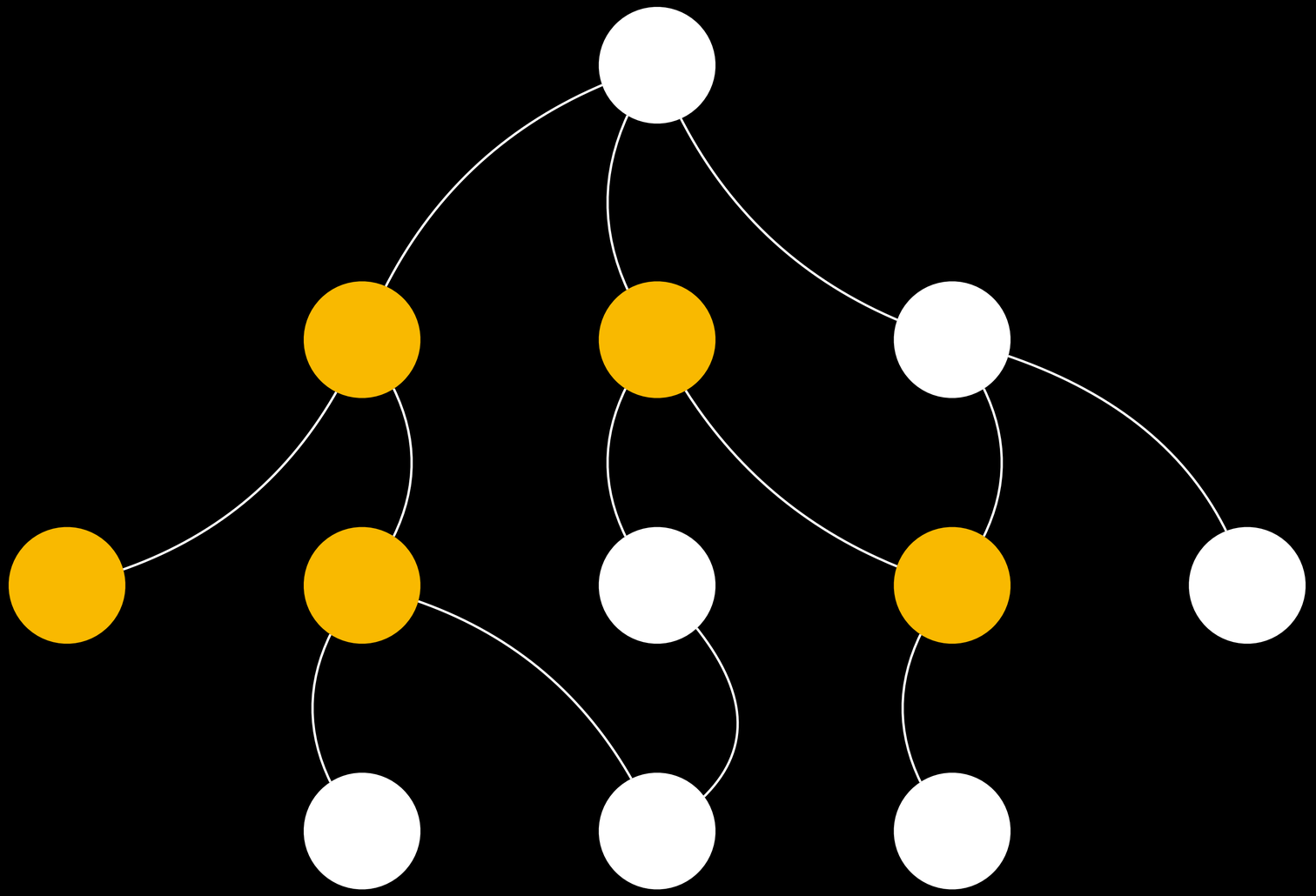


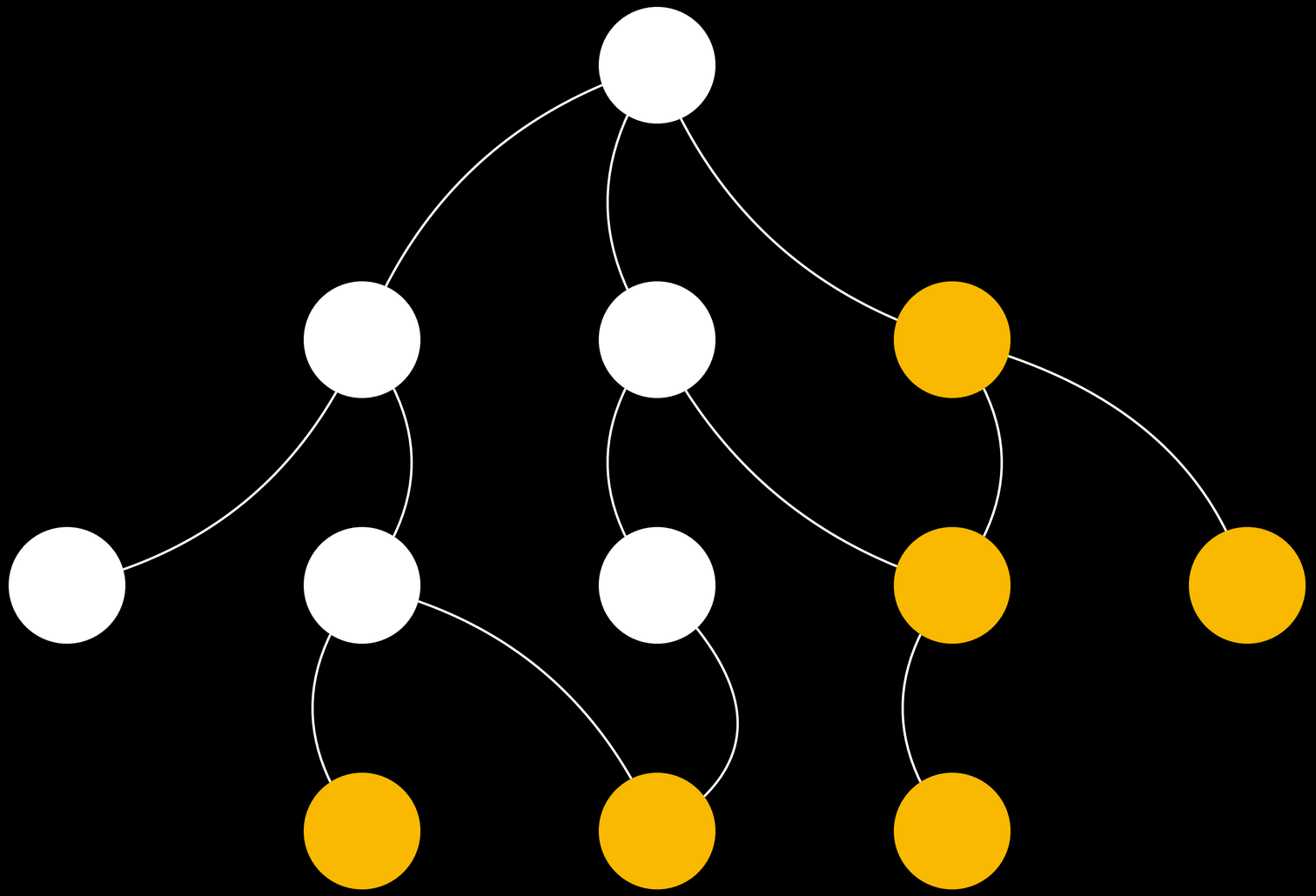


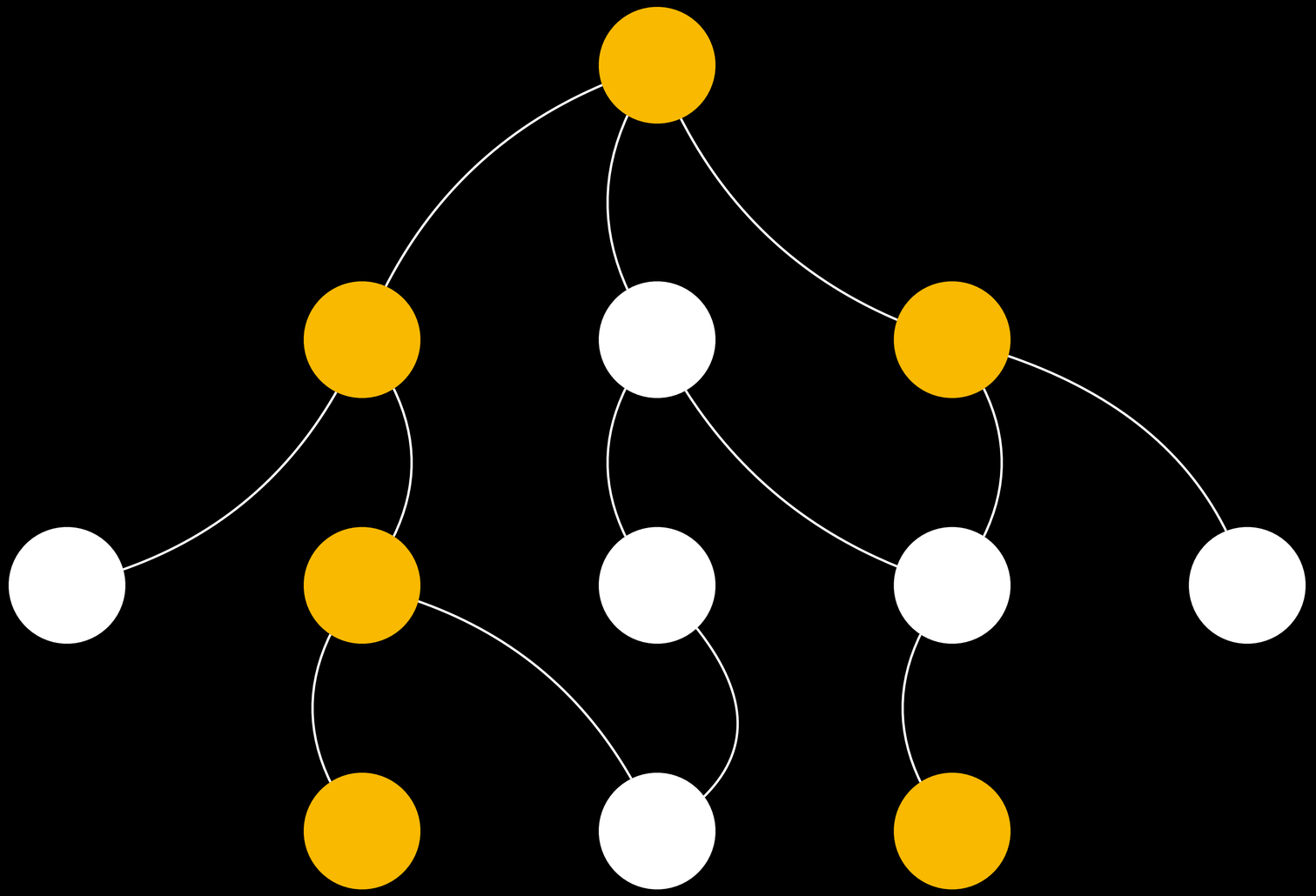


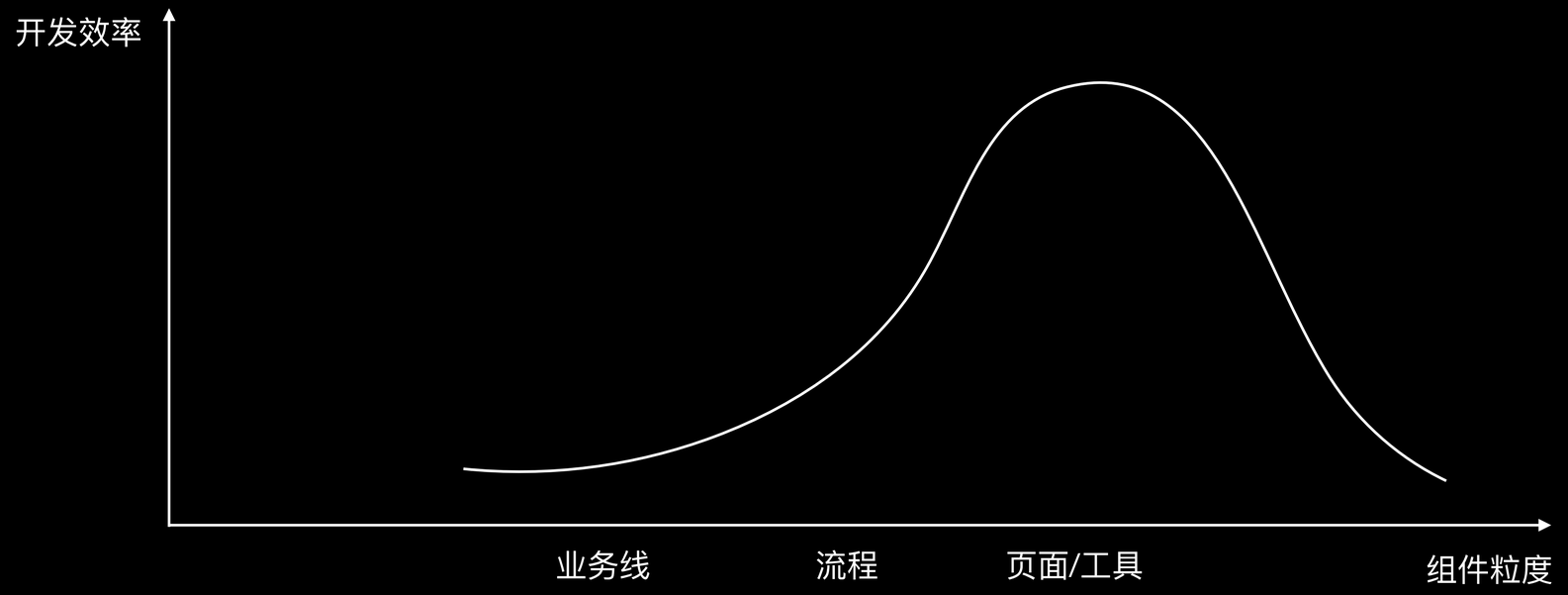


中等规模工程 配套 中等团队
组件要拆，但是拆解粒度要比之前小规模工程时更小
组件数量稍多一点，工作效率就更高

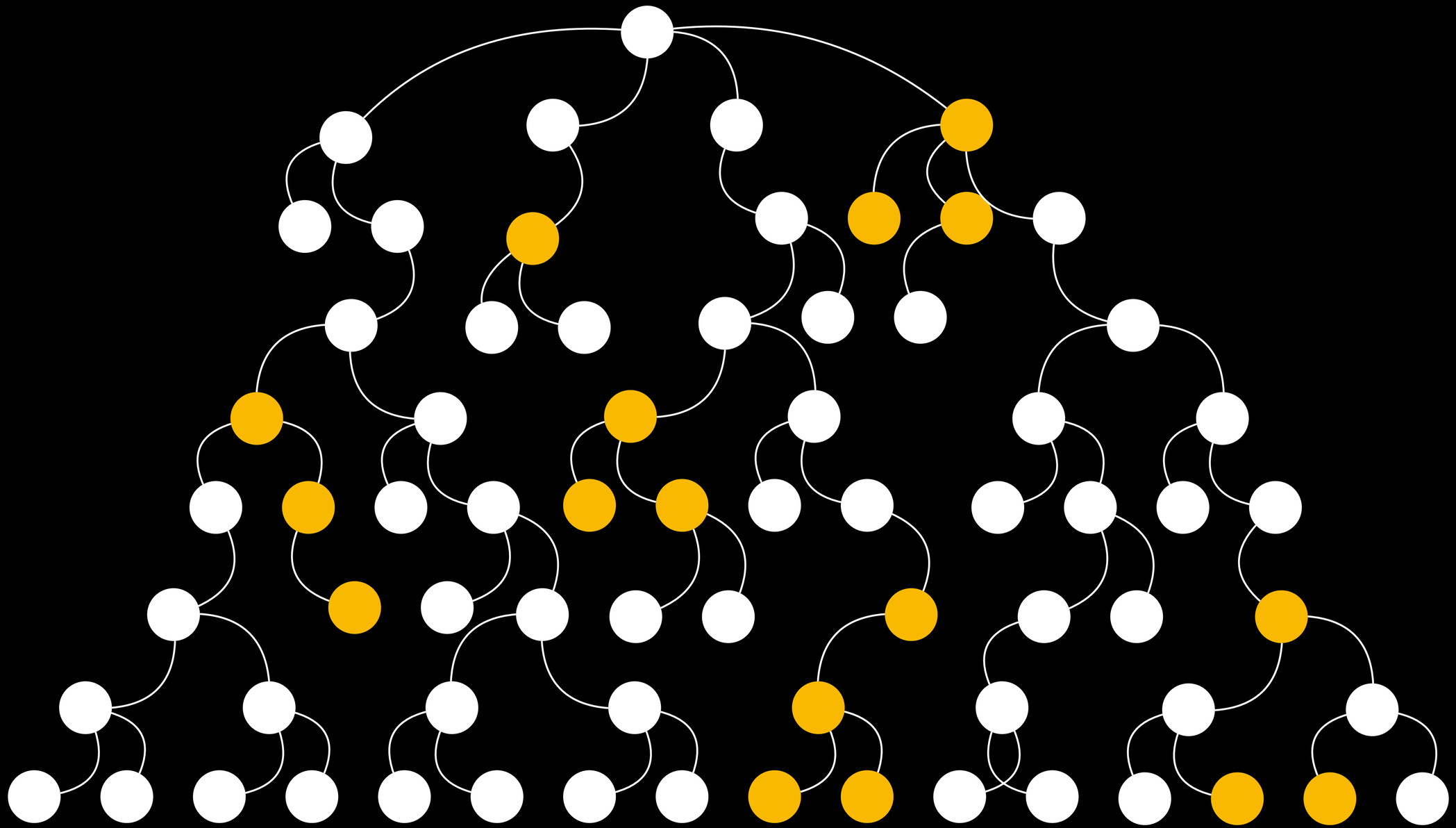


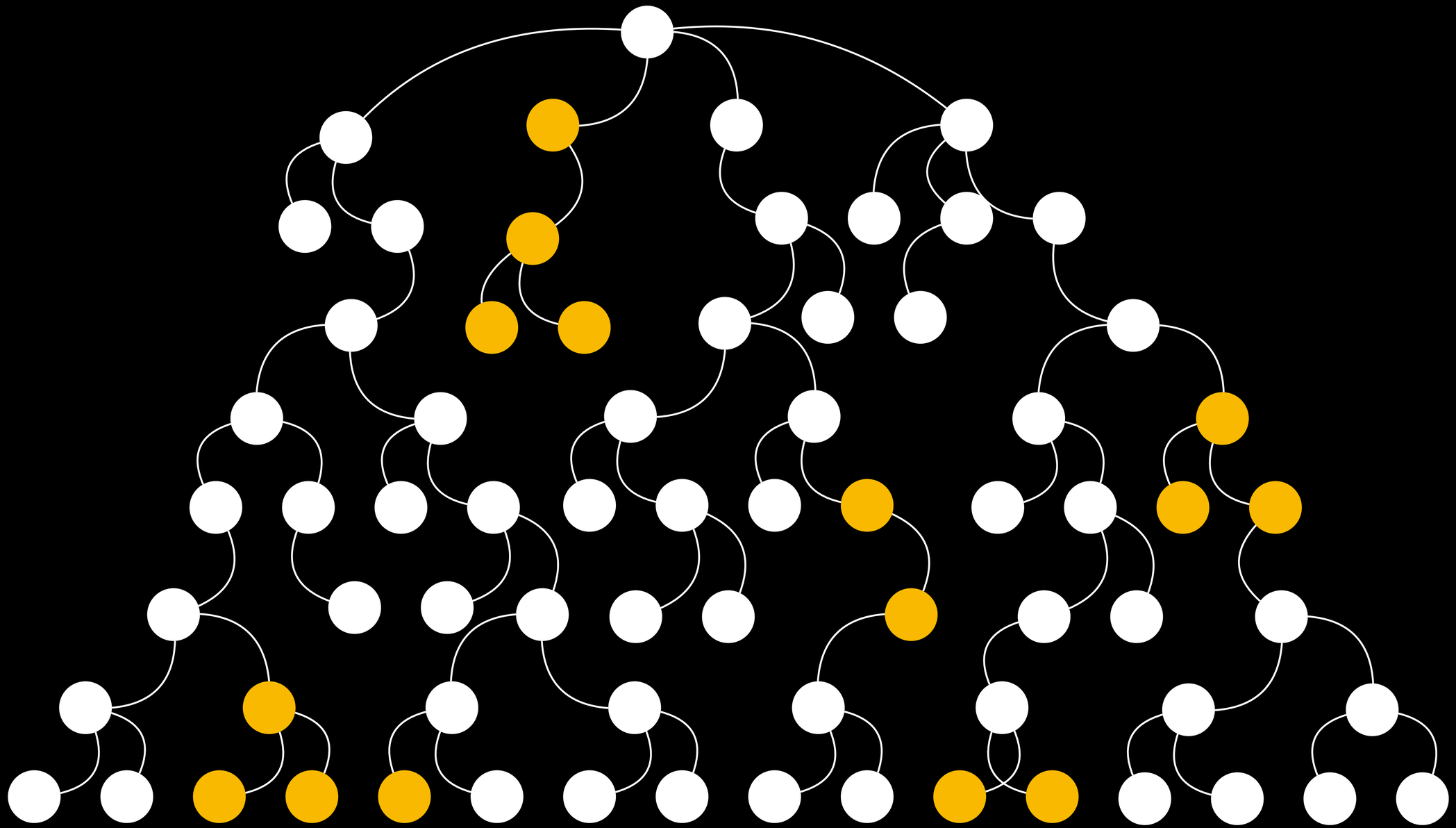


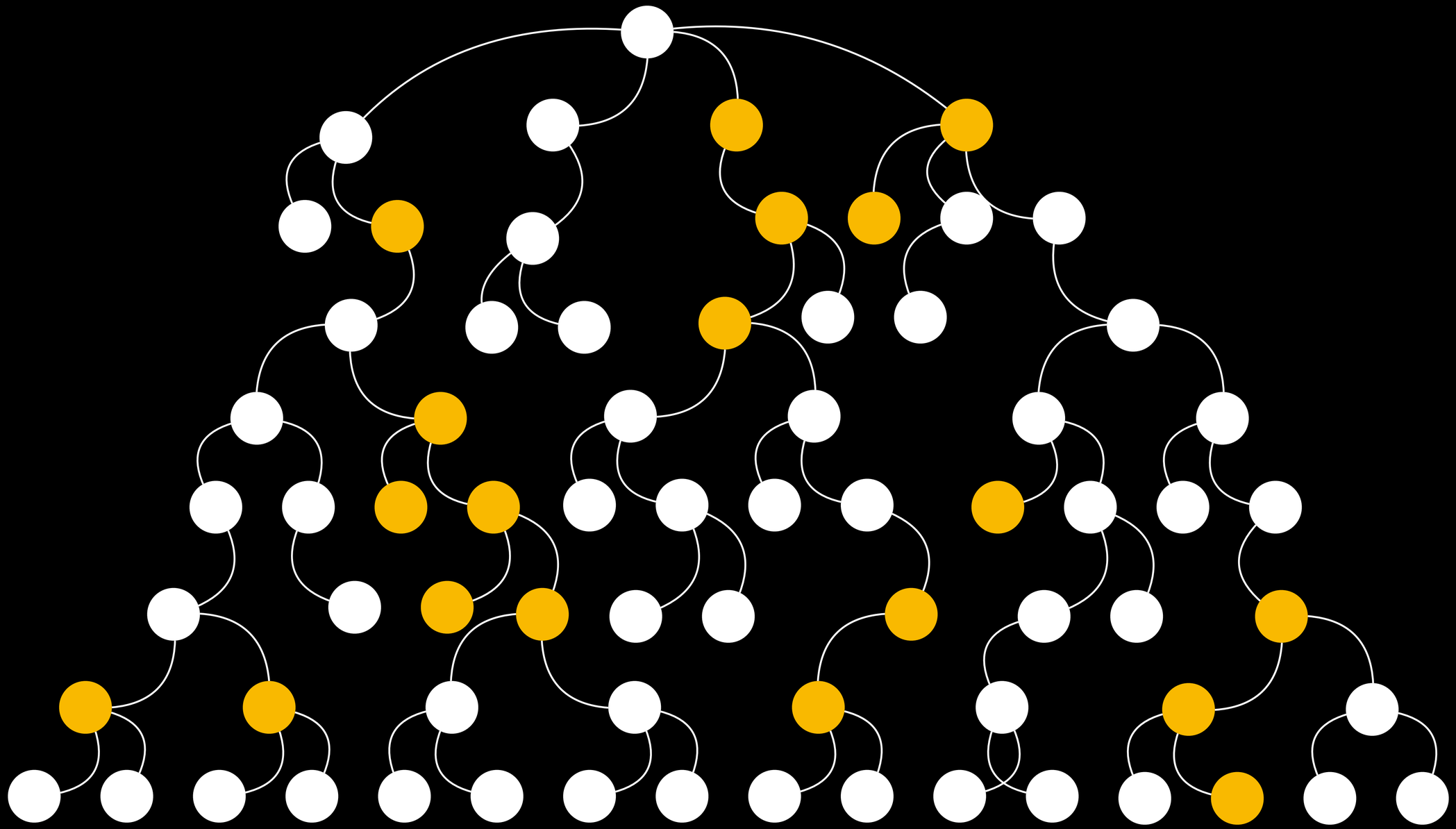




大规模工程 配套 大团队
组件要拆得更细，组件数量更多，工作效率才会高







好的架构没有Common，没有Core

Common、Core意味着指责不明确

将来难以维护，不是好的架构

我们花了很大精力
改造并删除了Common、Core、
AppEngineModule

我们花了很大精力
改造并删除了Common、Core、
AppEngineModule

我们也无法忍受大组件的存在

很少有依赖大组件，会用其全部功能
但只因为小功能就引入大组件
就会影响开发时的编译速度，维护时的排查效率

Argument List Too Long

Argument List Too Long

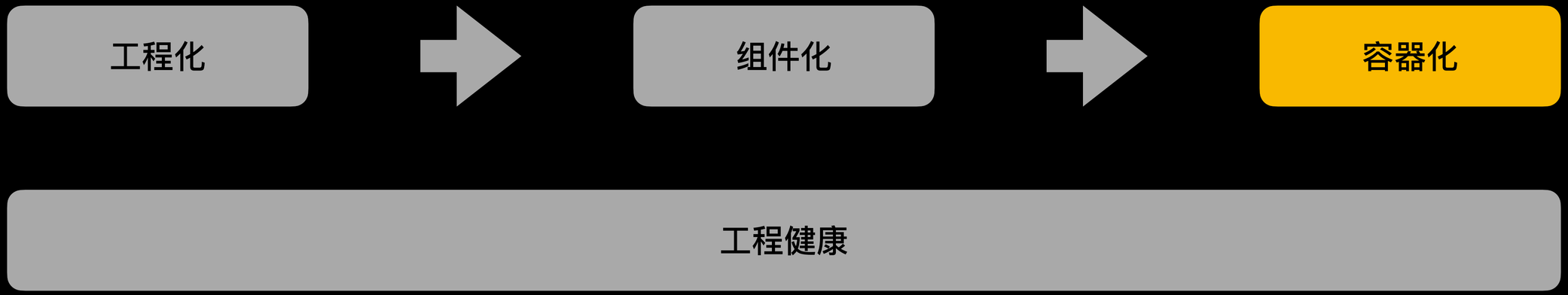
原因

- Cocoapods会为每一个Pod生成编译参数，并写入环境变量
- 命令行的所有环境变量字符串总长不能超过26万个字符

Argument List Too Long

解决方案

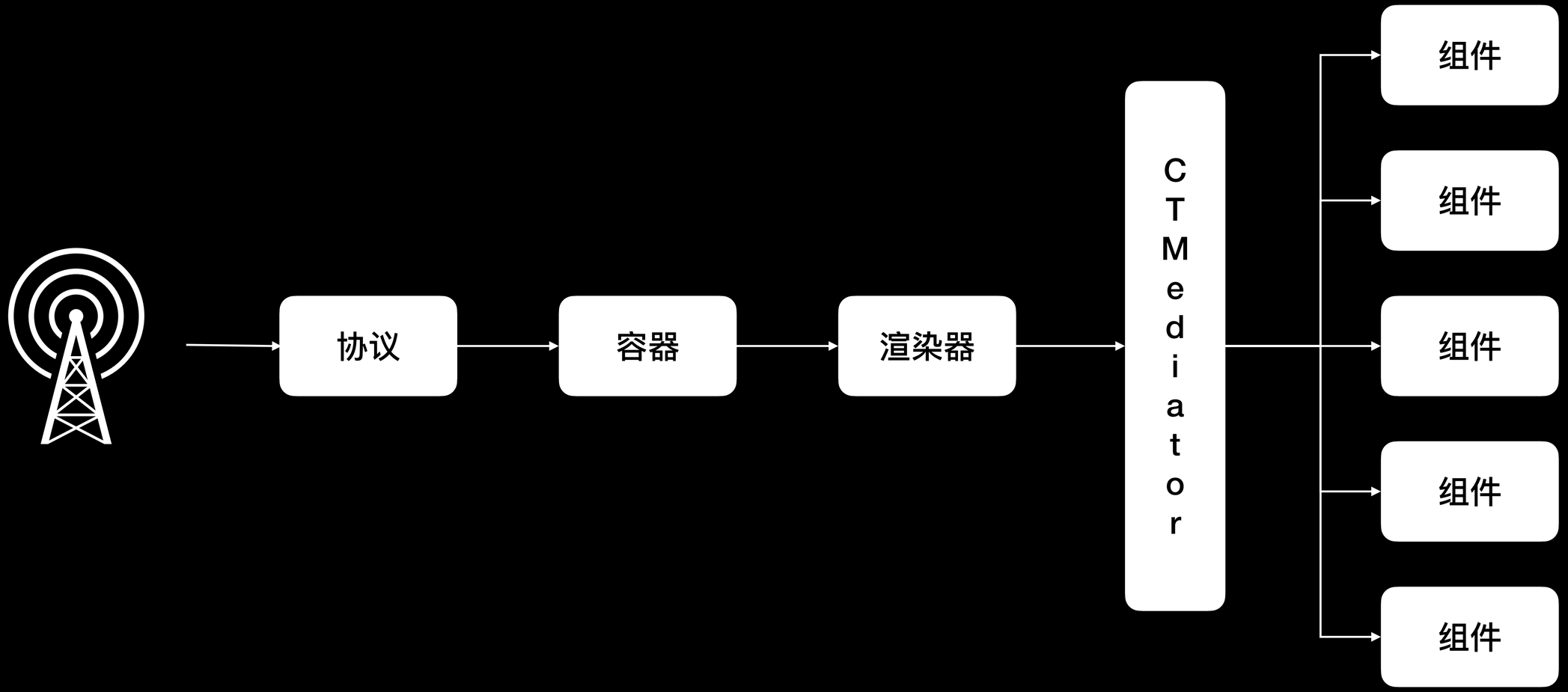
- 合并所有header search path到一个地方
- 合并所有library search path到一个地方
- 合并所有的modulemap文件成一个文件


工程化
组件化
容器化
工程健康

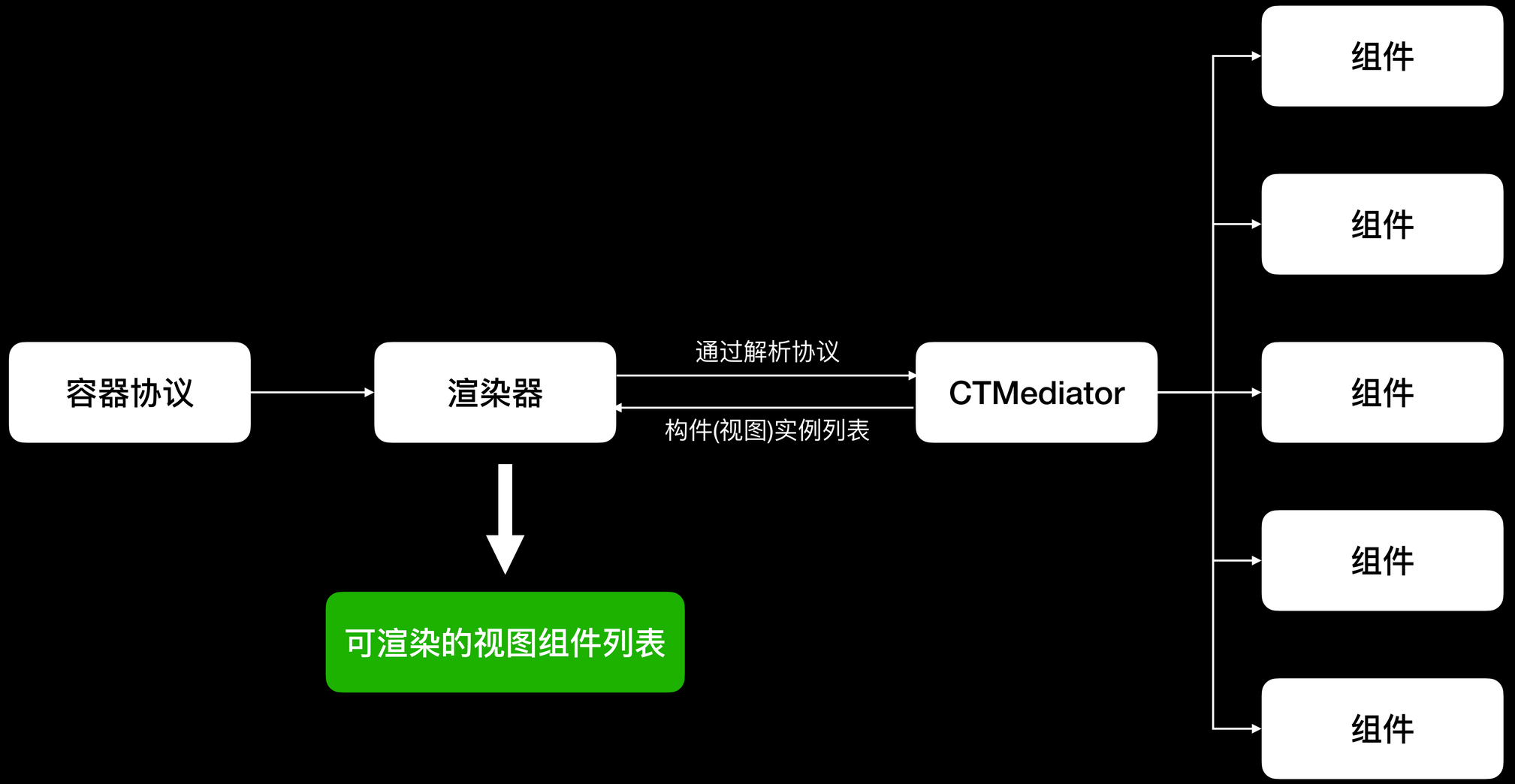
容器化

容器化是如何演化出来的?

容器化

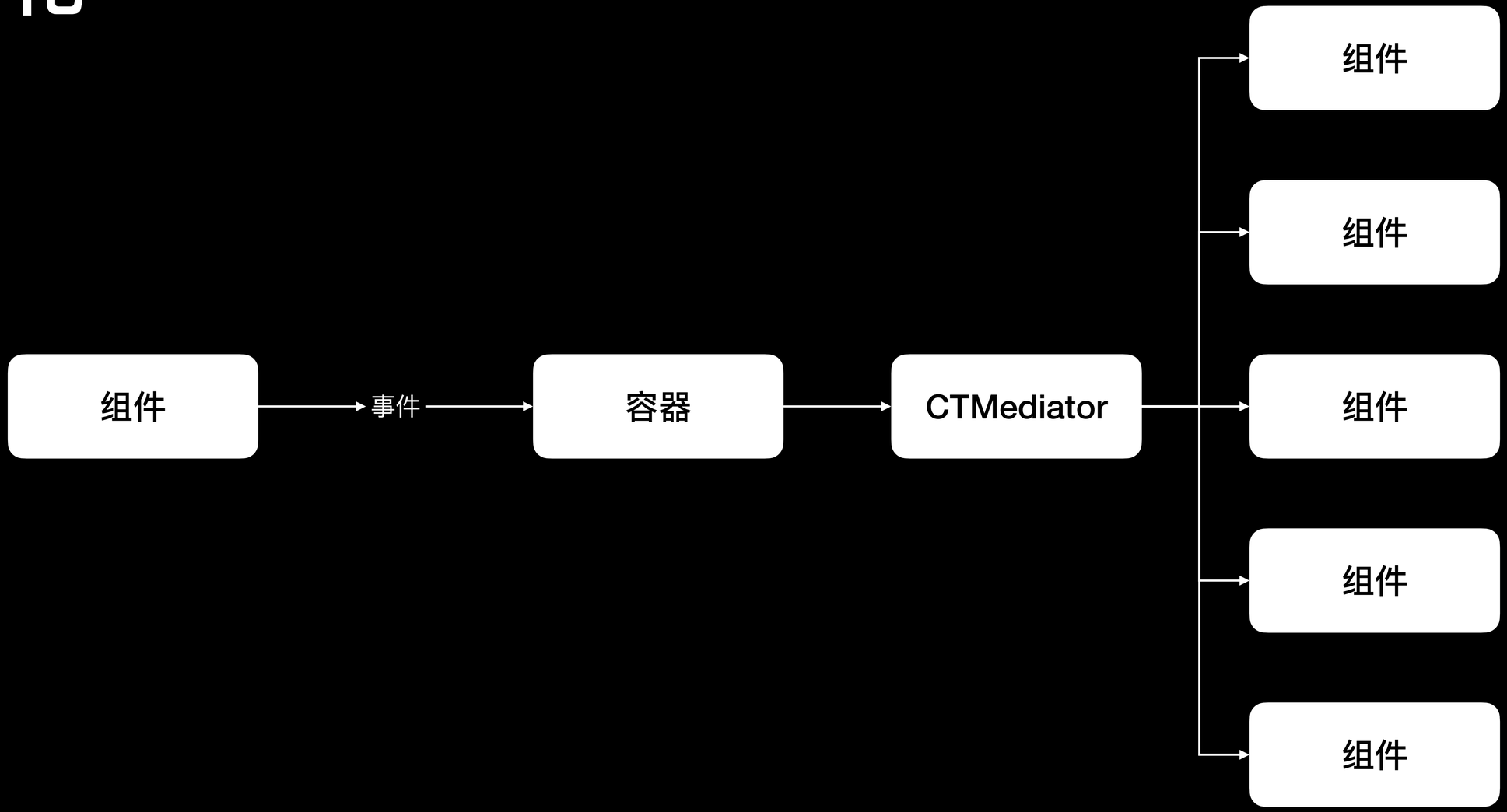


渲染器


组件
组件
容器协议
渲染器
通过解析协议
构件(视图)实例列表
CTMediator
组件
组件
可渲染的视图组件列表
组件

容器化

通信



容器化

Native级性能	即时发布	支持业务高速迭代
动态化	容器化	面向构件开发
可搭建	所见即所得	工程成长

容器化

应用

目前在首页、Tab页、活动页
完成了容器化部署


工程化
组件化
容器化
工程健康

谢谢！