React Native
思考和实践

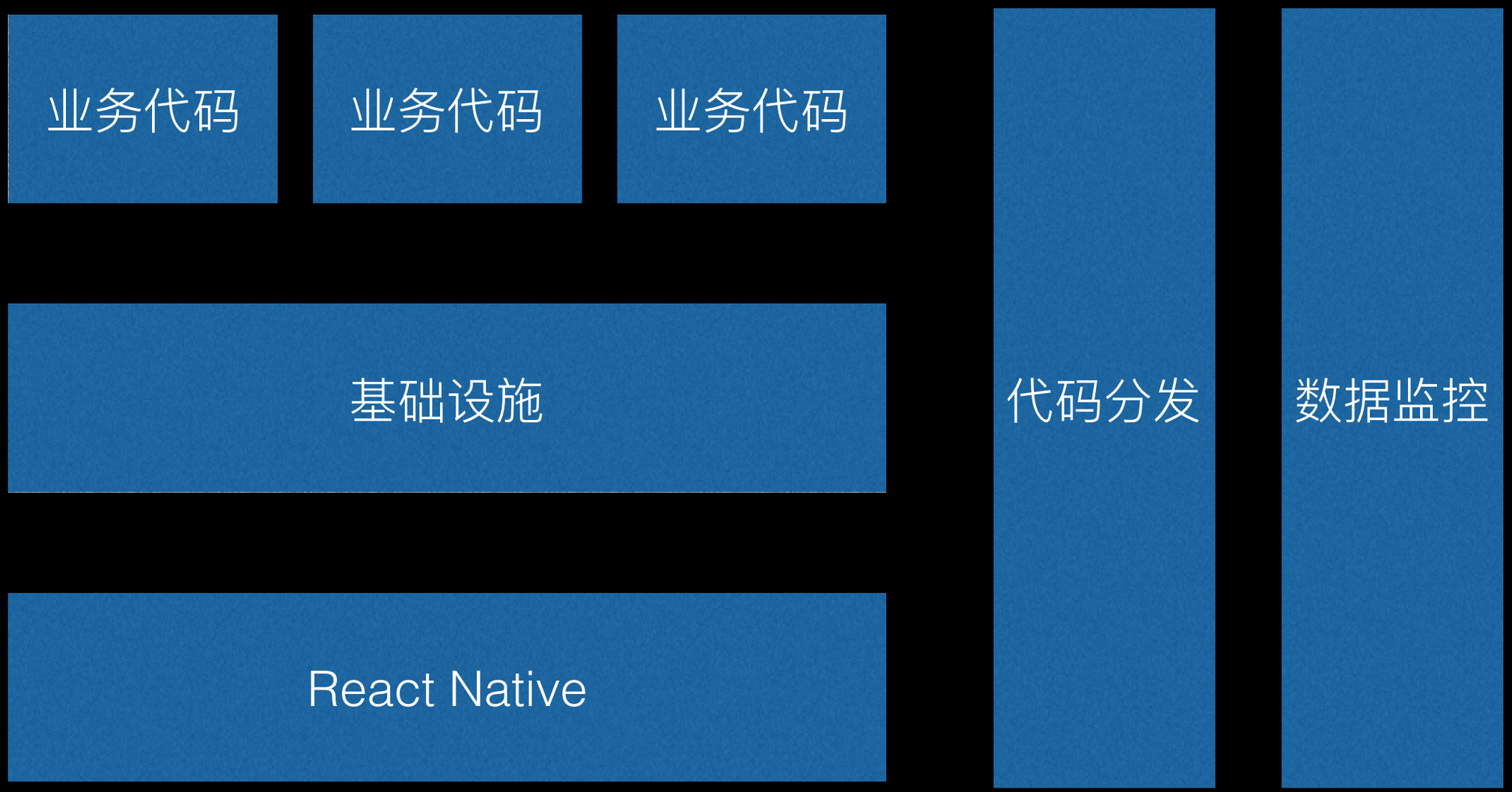
美团大众点评 寇祖阳

- 为什么选择 React Native
- 整体结构
- 代码的组织和分发
- 数据监控

为什么选择 React Native?

- 提高迭代速度
- 降低工作量
- 适合我们展示为主的应用

整体结构



代码组织

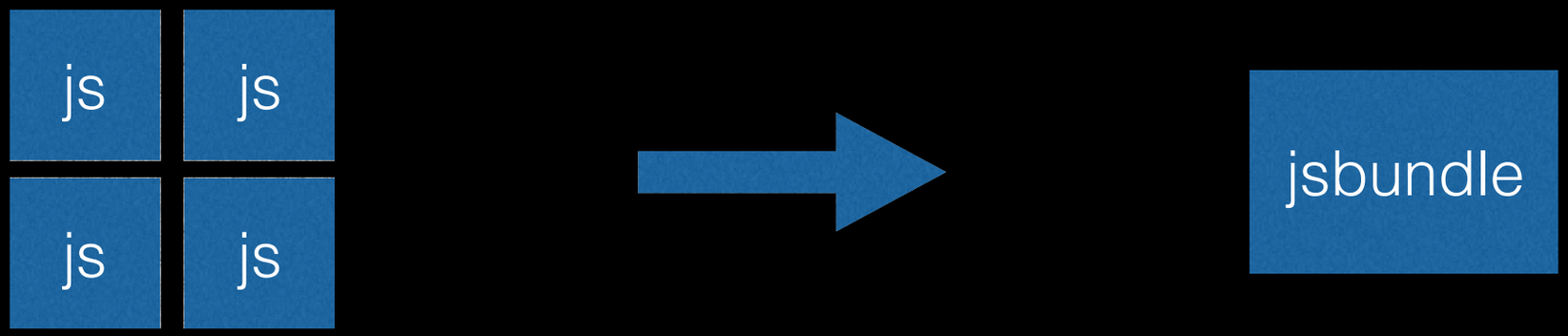
- 学习前端的实践
- 能够支持多个团队共同开发
- 考虑 RN 的特殊情况

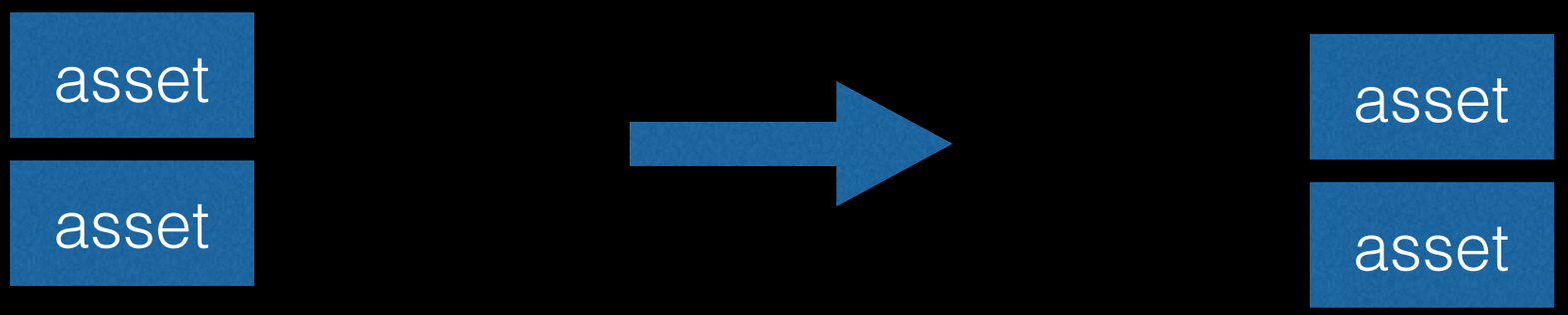
代码组织

- npm
- asset

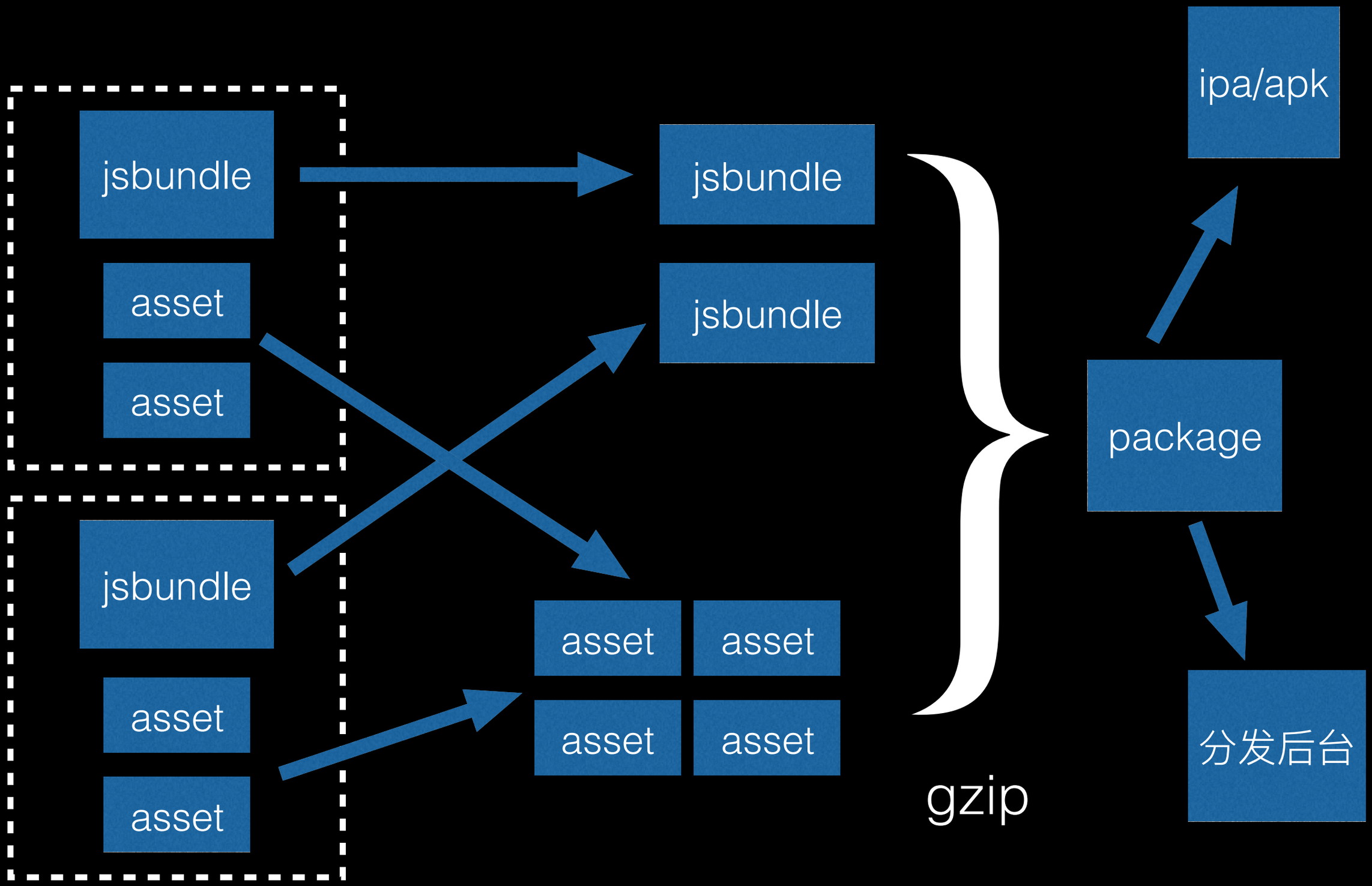
```
sample_business
├── asset
│   └── sample_image.png
├── package.json
└── script
    ├── index.android.js
    ├── index.ios.js
    └── sample_module.js
```

打包

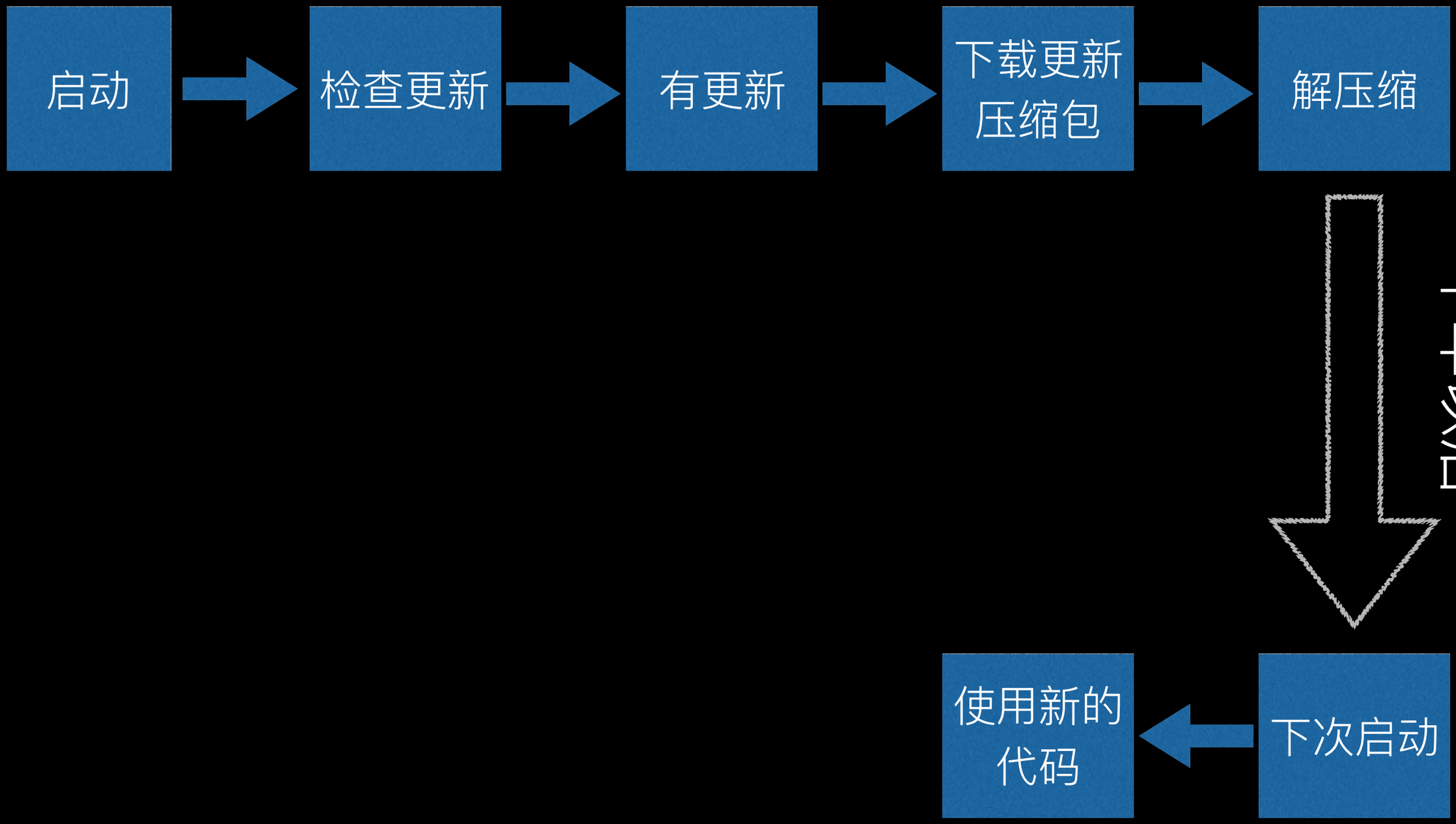




打包


jsbundle
asset
asset
jsbundle
jsbundle
asset
asset
asset
asset
asset
jsbundle
jsbundle
ipa/apk
package
分发后台
gzip

在线更新



案例分析

- 某天上线了一个崩溃
- 崩溃率飙升
- 迅速下线
- 影响一星期后消除

问题

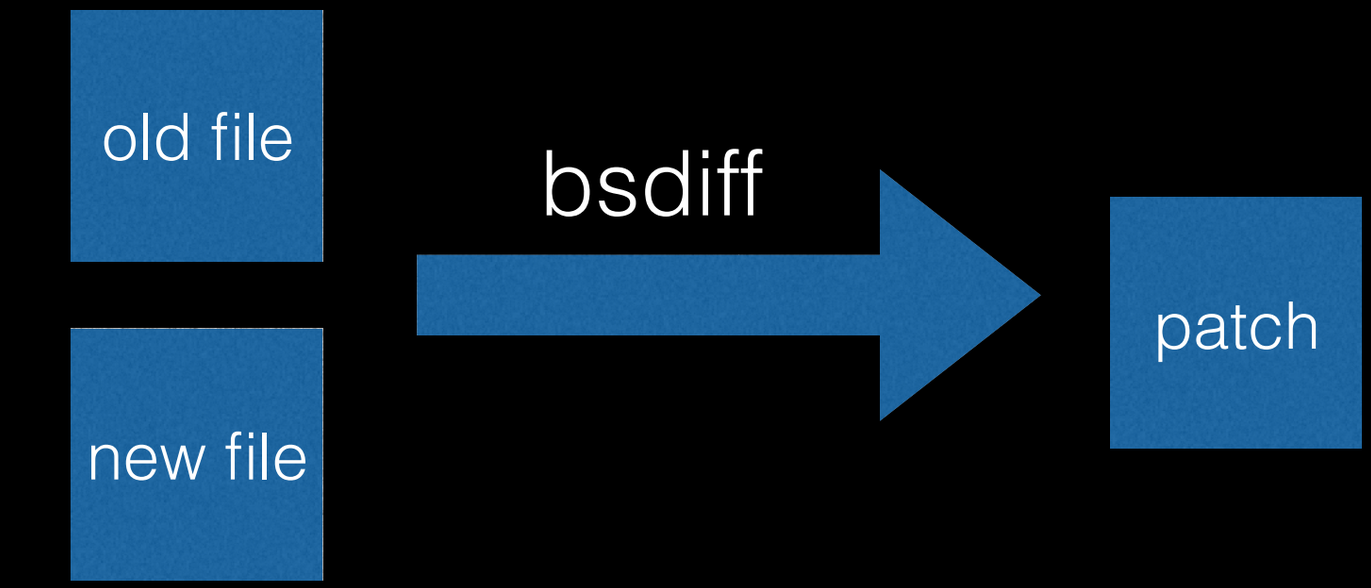
- 没有灰度功能，上线影响面大
- 流量消耗大，包下载成功率低
- 需要第二次启动，生效速率慢

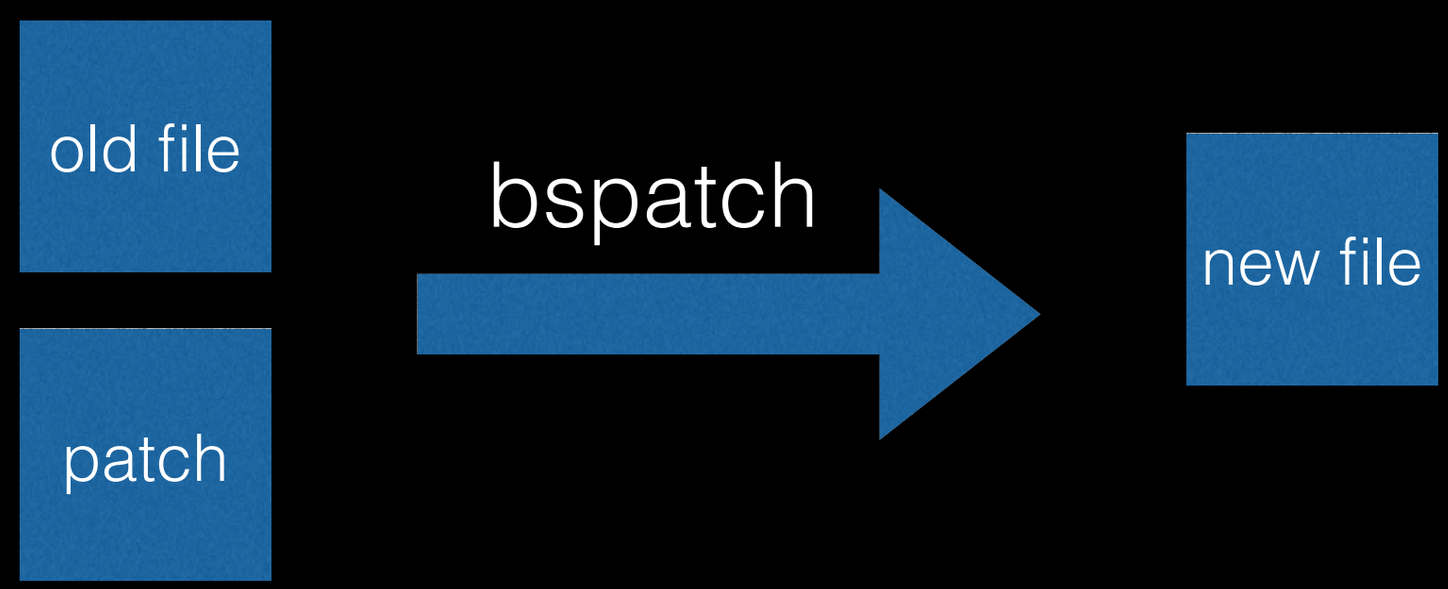
差量更新

- 文件级
- 二进制

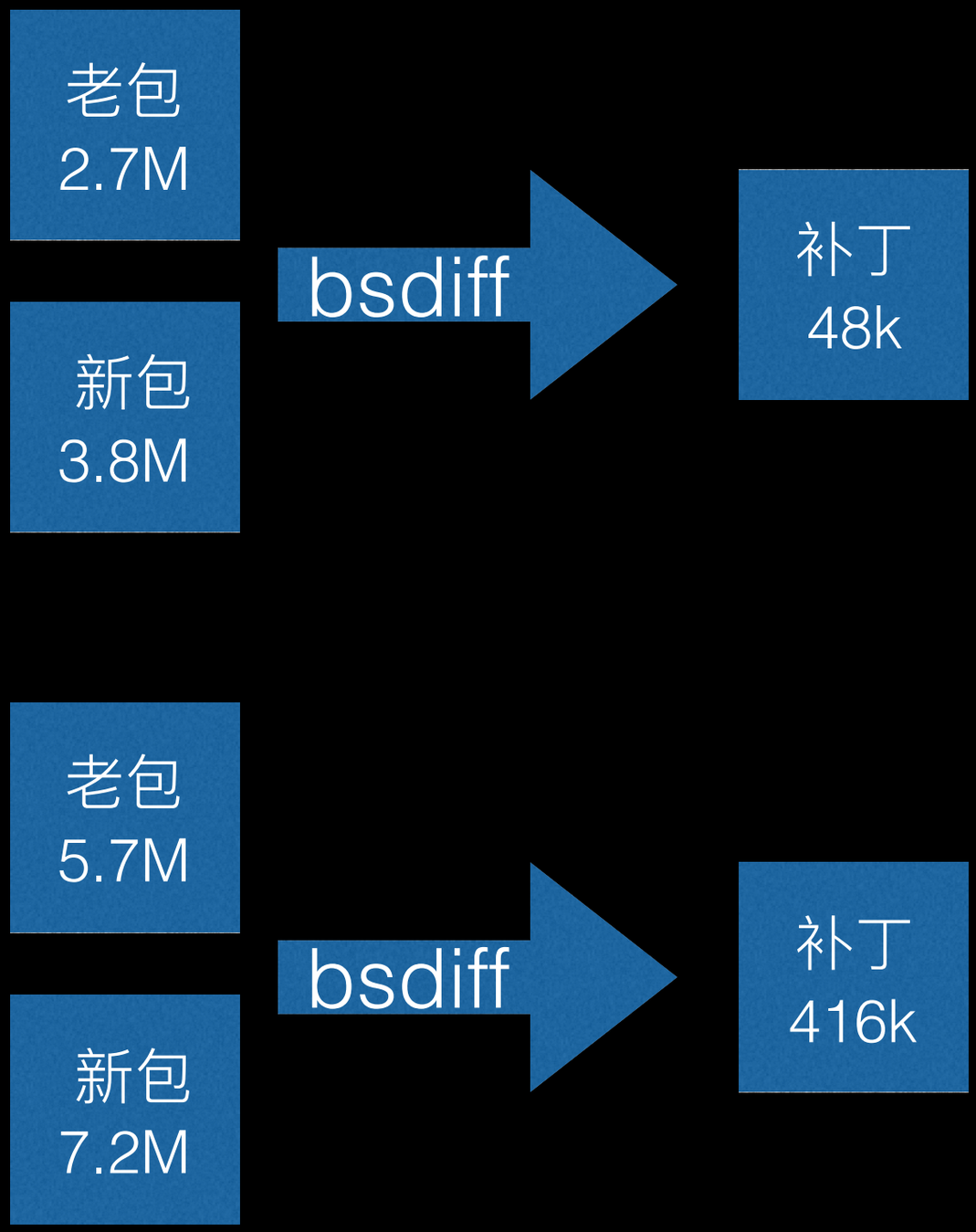
bsdiff

- 用于给可执行文件做差量
- 分为两个程序 bsdiff/bspatch

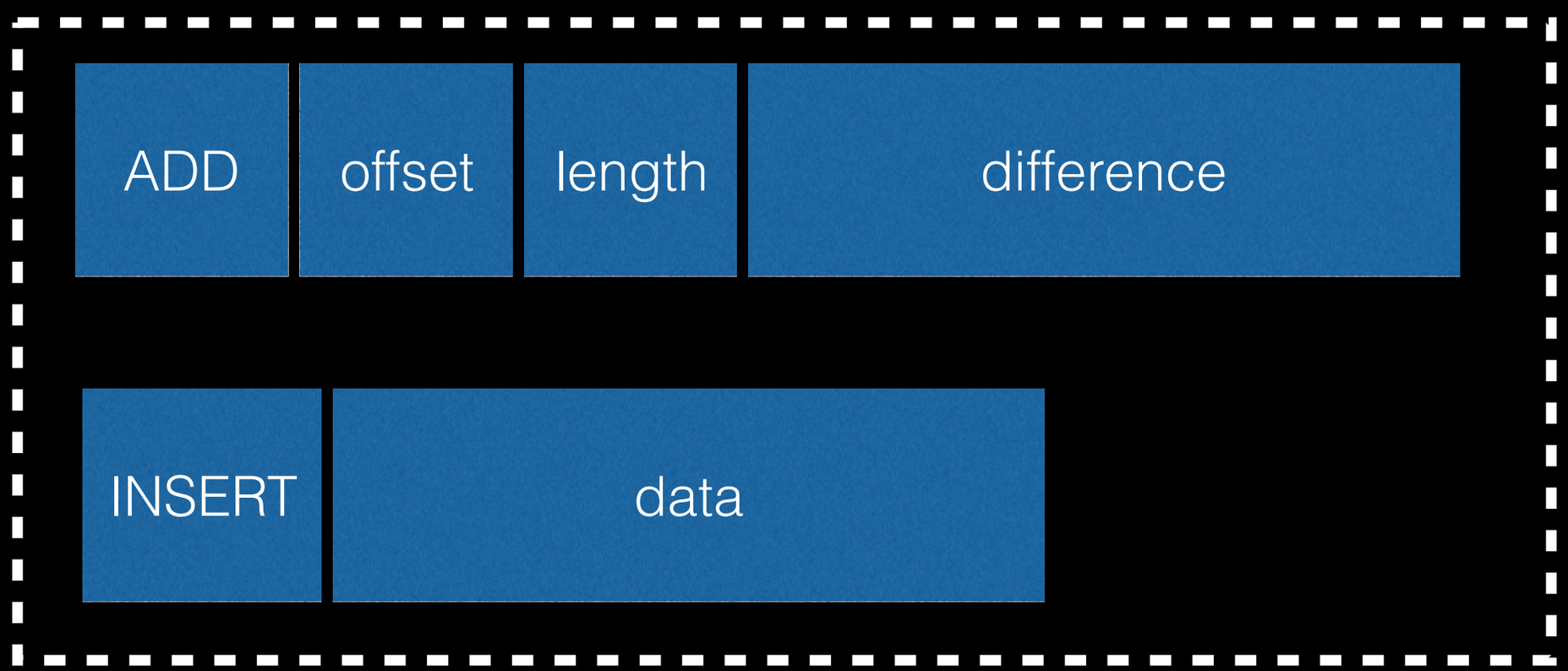

old file
new file
bsdiff
patch


old file
patch
bspatch
new file

效果

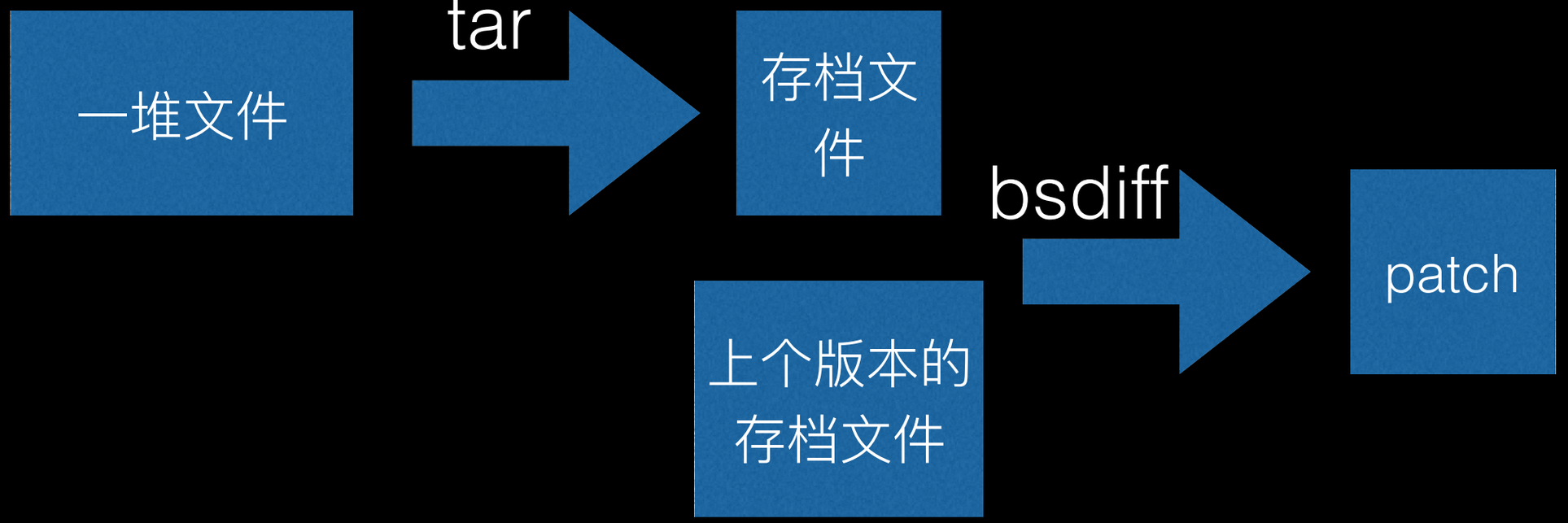


patch 文件结构



bzip2

制作 patch 的过程



改造

- 简化客户端实现
- 使用 gzip 替换 bzip2
- 使用 zip -0 替换 tar

生效速率慢

- 只知道慢，到底有多慢?
- 关键指标是什么?

改进思路

- 增加启动机会
- 无需重启直接生效
- web 化

监控

- 如何保证生产环境中没有问题?
- 崩溃监控
- 可用性监控
- 业务数据监控

可用性监控

- 如何能知道 RN 的界面是可用的?

监控方法

- native 和 RN 相结合
- native 检查整个 root view 有显示，bridge 加载完成
- RN 主动向 native 报告模块加载完成
- native 检查一下带有标记的 view 是否存在

结论

~99%的情况下 RN 页面是能够显示出来的

这就够了吗?

- 复杂业务逻辑难以验证
- 一个页面的最终目的是什么?
- 这个目的如何来衡量?

业务数据监控

- 点击率
- 转化率
- etc

回顾一下

- 整体结构
- 代码的组织和分发
- 数据监控

谢谢大家