CHINA DEVOPS DAYS

阿里巴巴在DevOps实践中的创新和思考

2018 中国·上海

CHINA DEVOPS DAYS

林帆

花名金戟，阿里巴巴研发效能事业部技术专家
前ThoughtWorks高级DevOps技术咨询师

国内早期的DevOps和容器技术实践者和传播者
CNUT全球运维大会、CSDN架构峰会一线讲师
丰富的一线开发和运维工程经验

著有《CoreOS实践之路》和
《容器即服务：从零构建企业级容器集群》两本书籍

今天的话题

1. 更简，从Gitflow到Aoneflow
2. 更快，从部署10分钟到10秒
3. 更轻，从虚拟硬件到虚拟服务

CHINA DEVOPS DAYS

代码管理，
从Gitflow到Aoneflow

Aoneflow文章被问及最多的两个问题

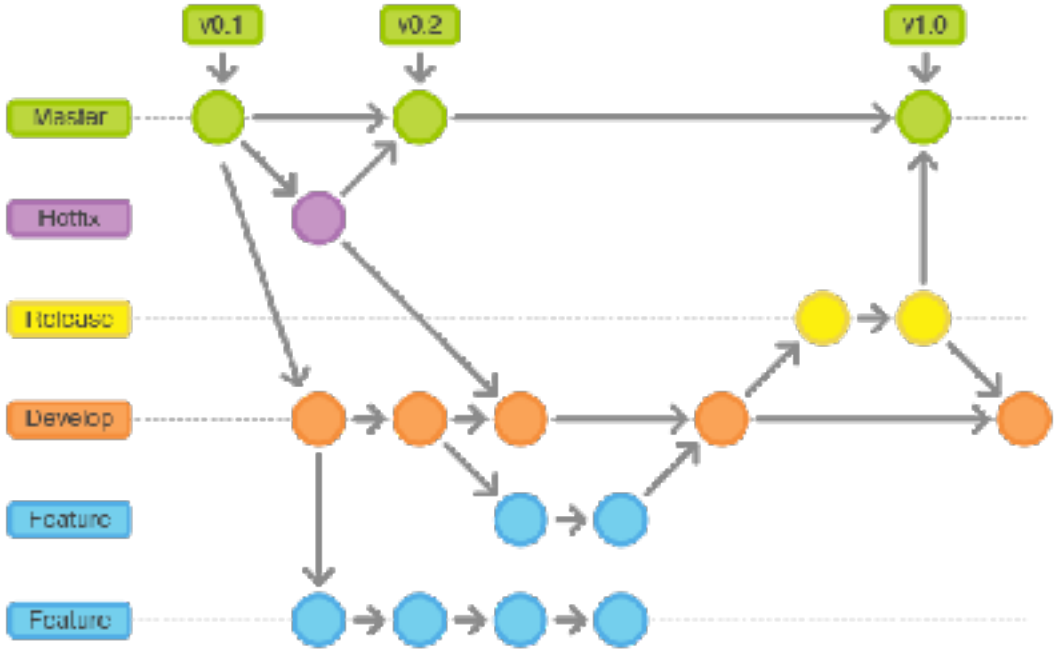
阿里巴巴的研发模式为什么没有走向“单主干(trunk based)”？

- 项目规模大，Aoneflow在持续集成和特性隔离之间做了平衡
- 开发者习惯，平台的支持情况
- 许多新的小型项目也在尝试单主干模式

Aoneflow就是简化的Gitflow？

- 两者除了都有“特性分支”和“发布分支”以外，流程和机制完全不同

Gitflow有什么问题？



集成时间滞后

- 特性分支在功能完成前，“不敢”随意合并回Develop分支，造成代码集成时间严重滞后

代码集中冲突

- 每次功能完成后进行“大集成”，十分容易出现大范围代码冲突

特性易合难分

- 特性一旦集成到Develop分支便难以再次分离，单个特性问题可能导致整体发布延期

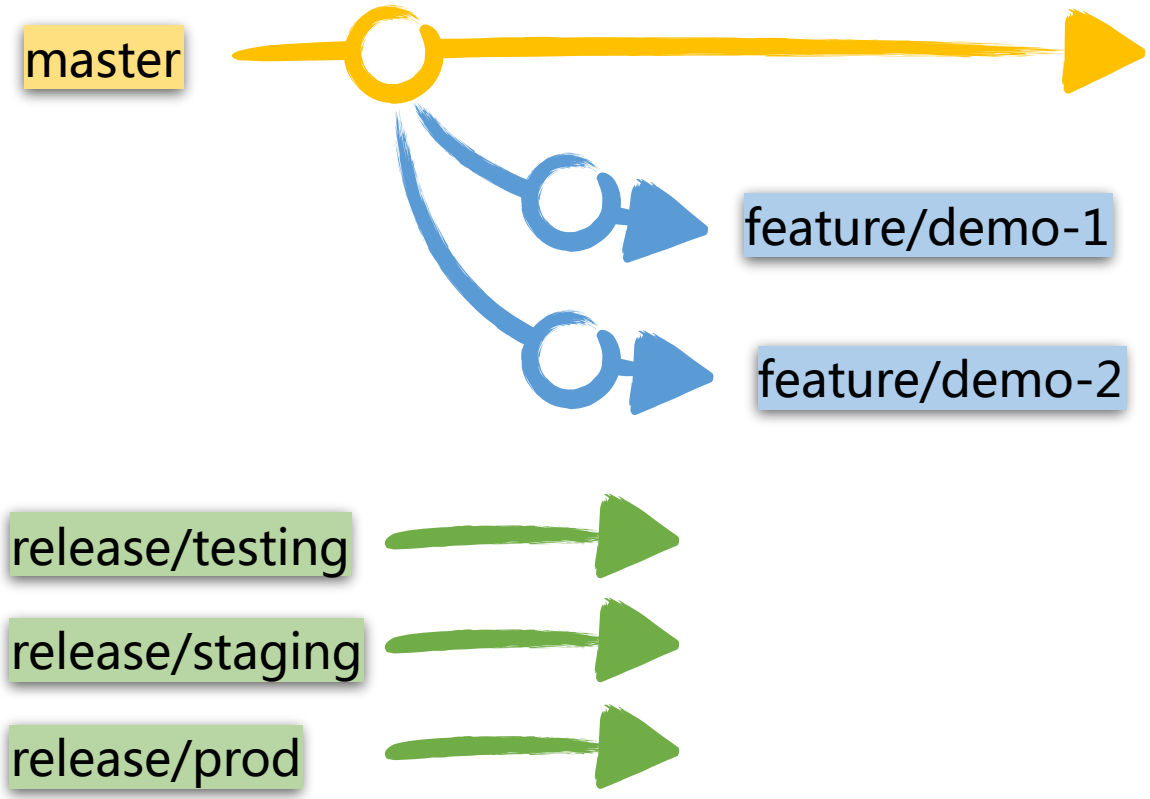
分支关系复杂

- 发布分支拉出后，直到合回主干，若有特性修改或Hotfix需要维护多处CherryPick合并

阿里人为何偏爱Aoneflow？

- 保留“特性分支”的工作方式，便于大型团队协作
- 简化Gitflow的复杂分支策略，上手容易
- 灵活的特性分支组合集成，集成后亦可快速剥离
- 实现“准持续集成”
  - 略低于单主干，远高于Gitflow的集成频率
  - 选择性的特性持续集成（方便灵活，但其实并非优点）

Aoneflow的开发者(用户)视角

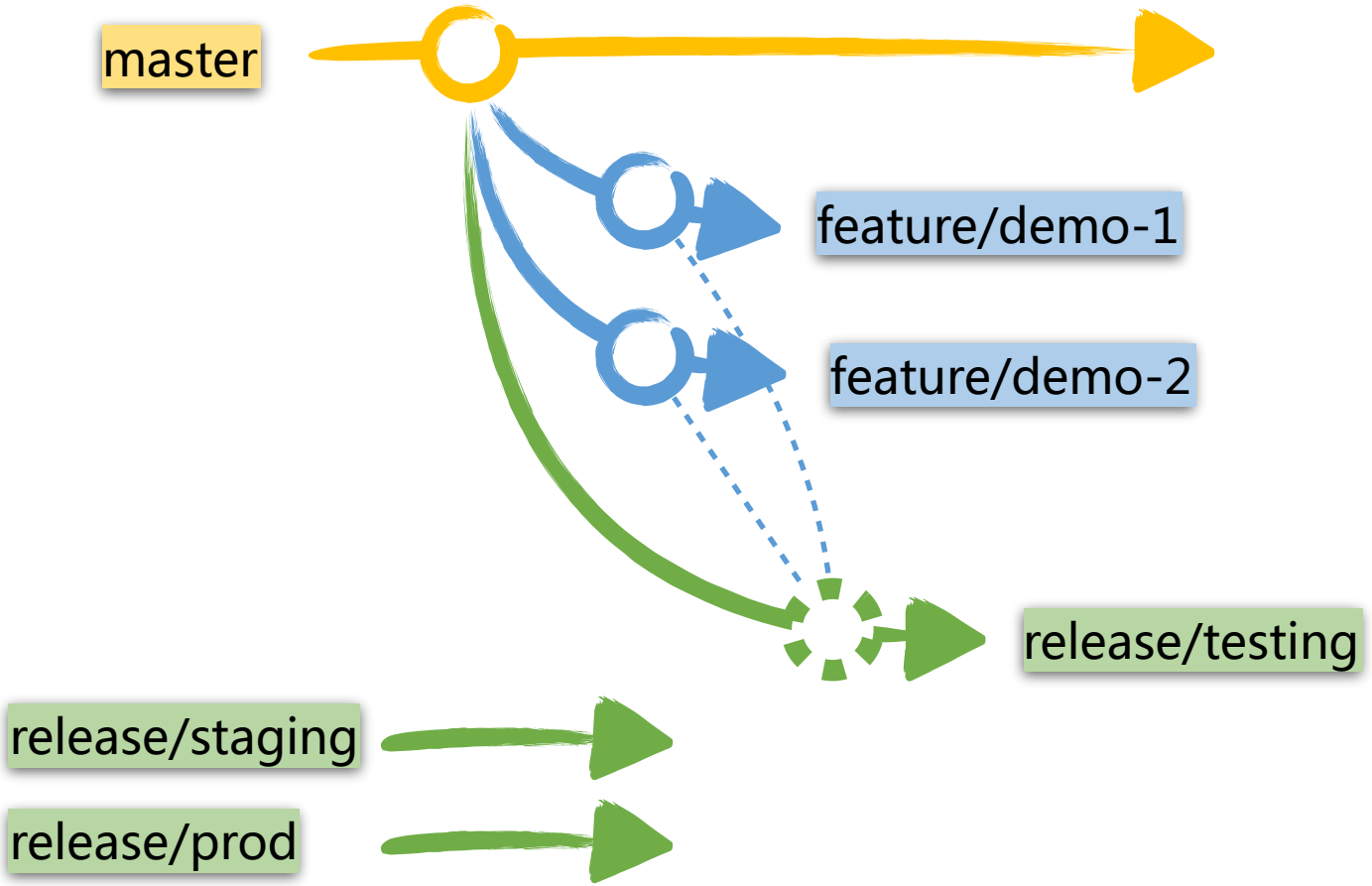


从开发角度来看
主干和发布分支始终存在

主干对应线上最新的稳定版本
发布分支对应各个环境正在测试的版本

开发者通过平台从主干创建特性分支

Aoneflow的开发者(用户)视角

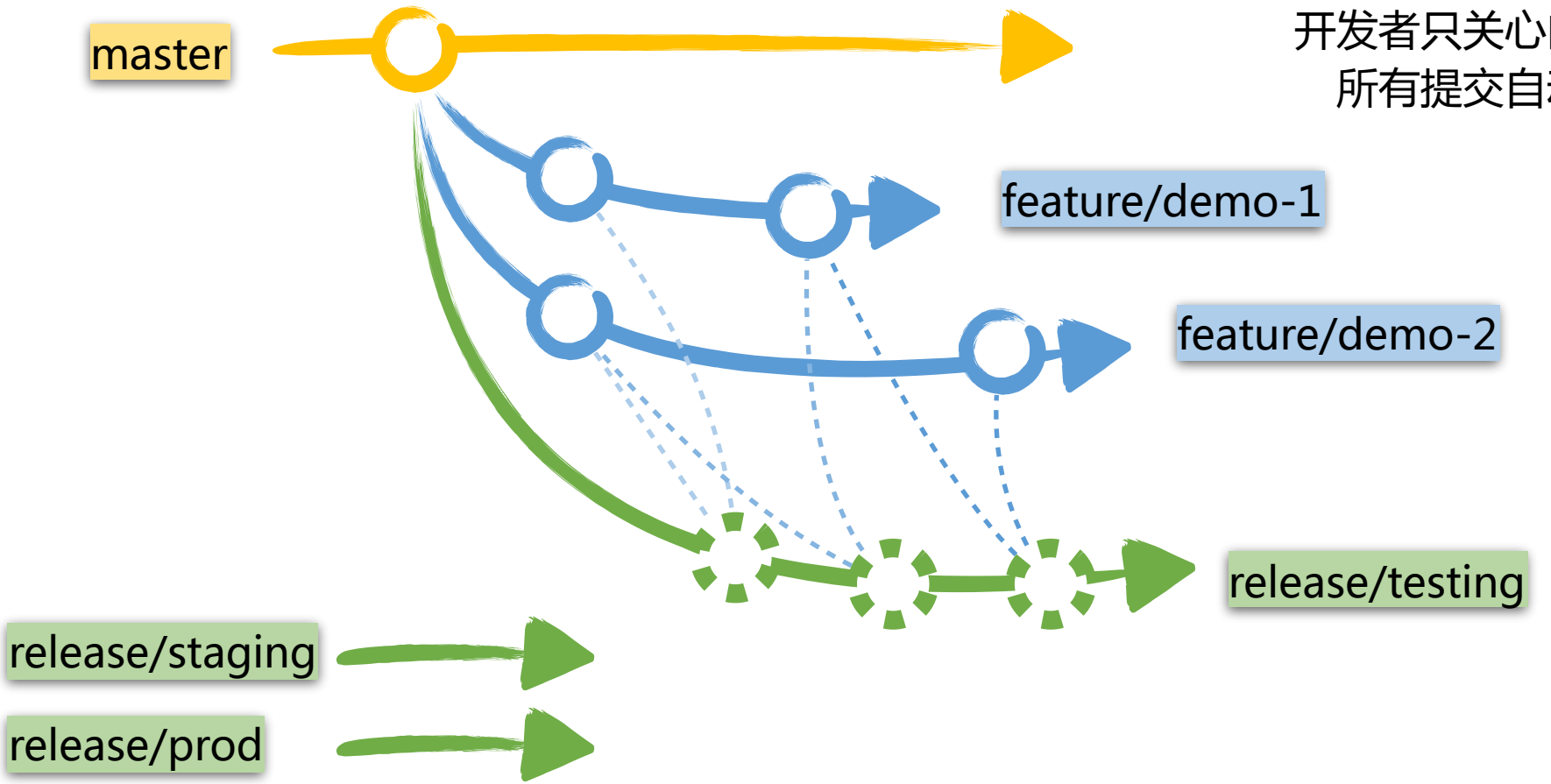


发布分支的生命周期由平台自动管理

开发者选择了几个特性分支
在平台的测试环境页轻轻一点

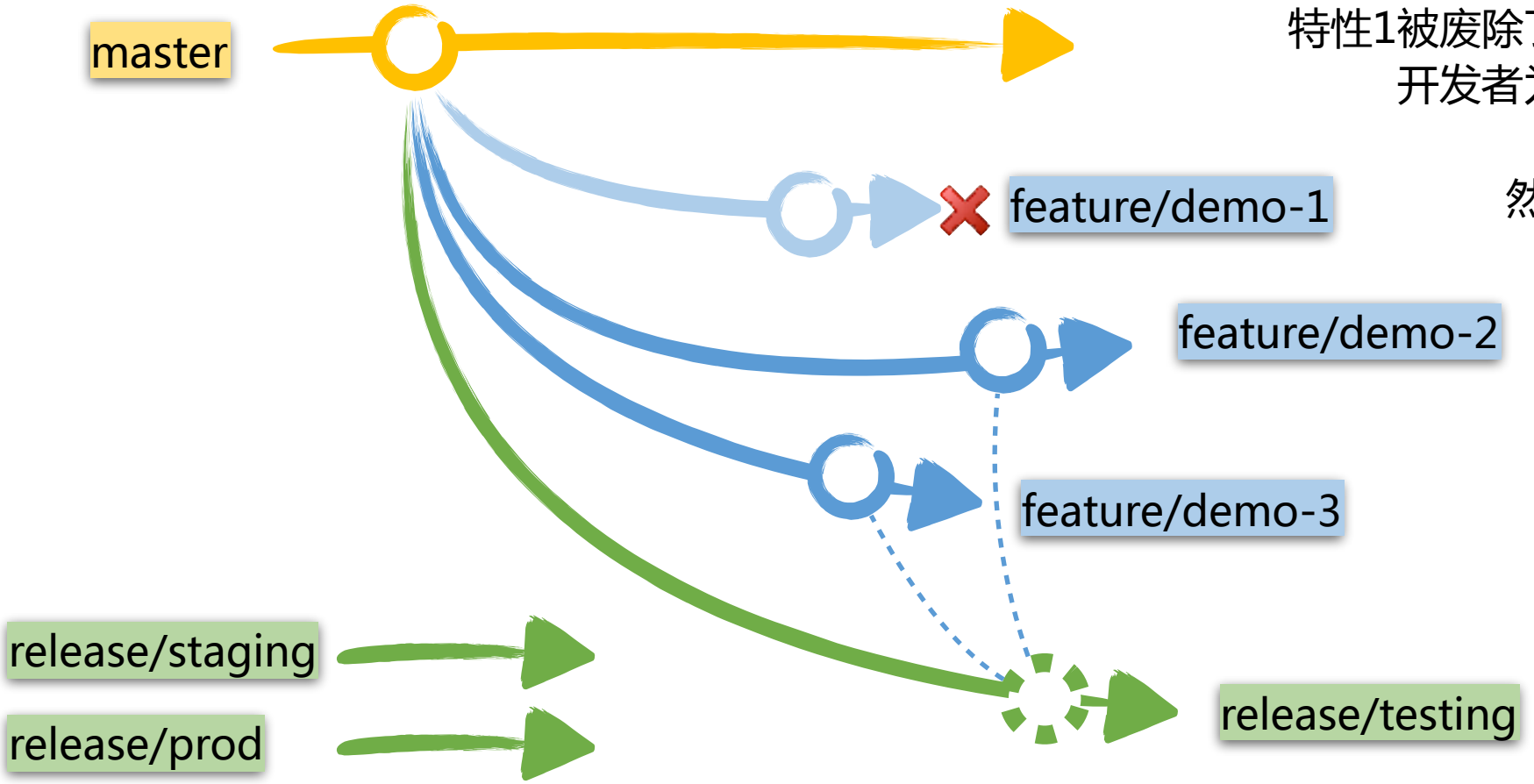
于是组成了测试环境的发布分支

Aoneflow的开发者(用户)视角



开发者只关心向特性分支提交代码
所有提交自动合并到发布分支上

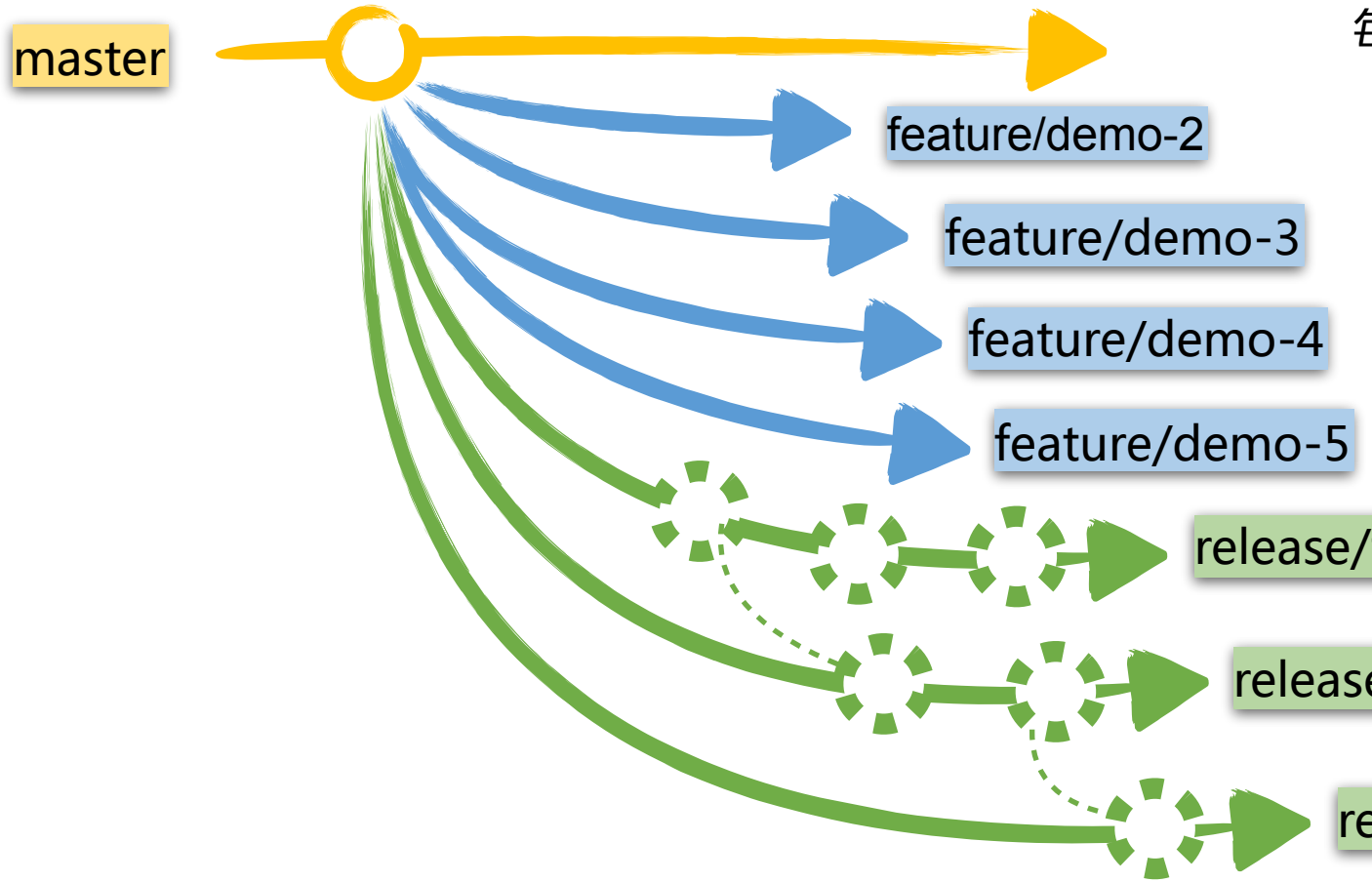
Aoneflow的开发者(用户)视角



特性1被废除了，临时增加了特性3
开发者为特性3创建特性分支

然后在平台上轻轻一点
发布分支重建了
剔除了特性1
增加了特性3

Aoneflow的开发者(用户)视角



每个发布分支有独立的交付流水线

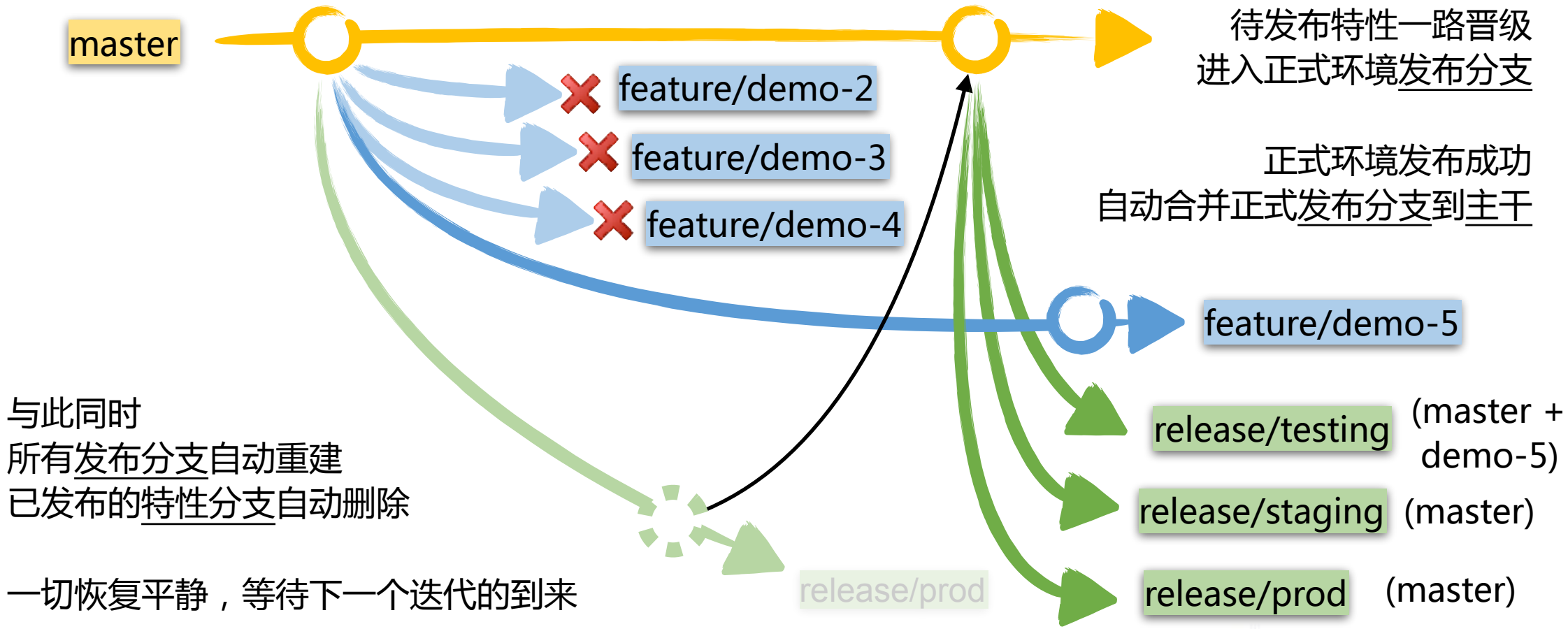
当流水线上的测试验证完成
就可以将特性列表一键拷贝
进入到下一级发布分支流水线

(master + demo-2 + demo-3 + demo-4 + demo-5)

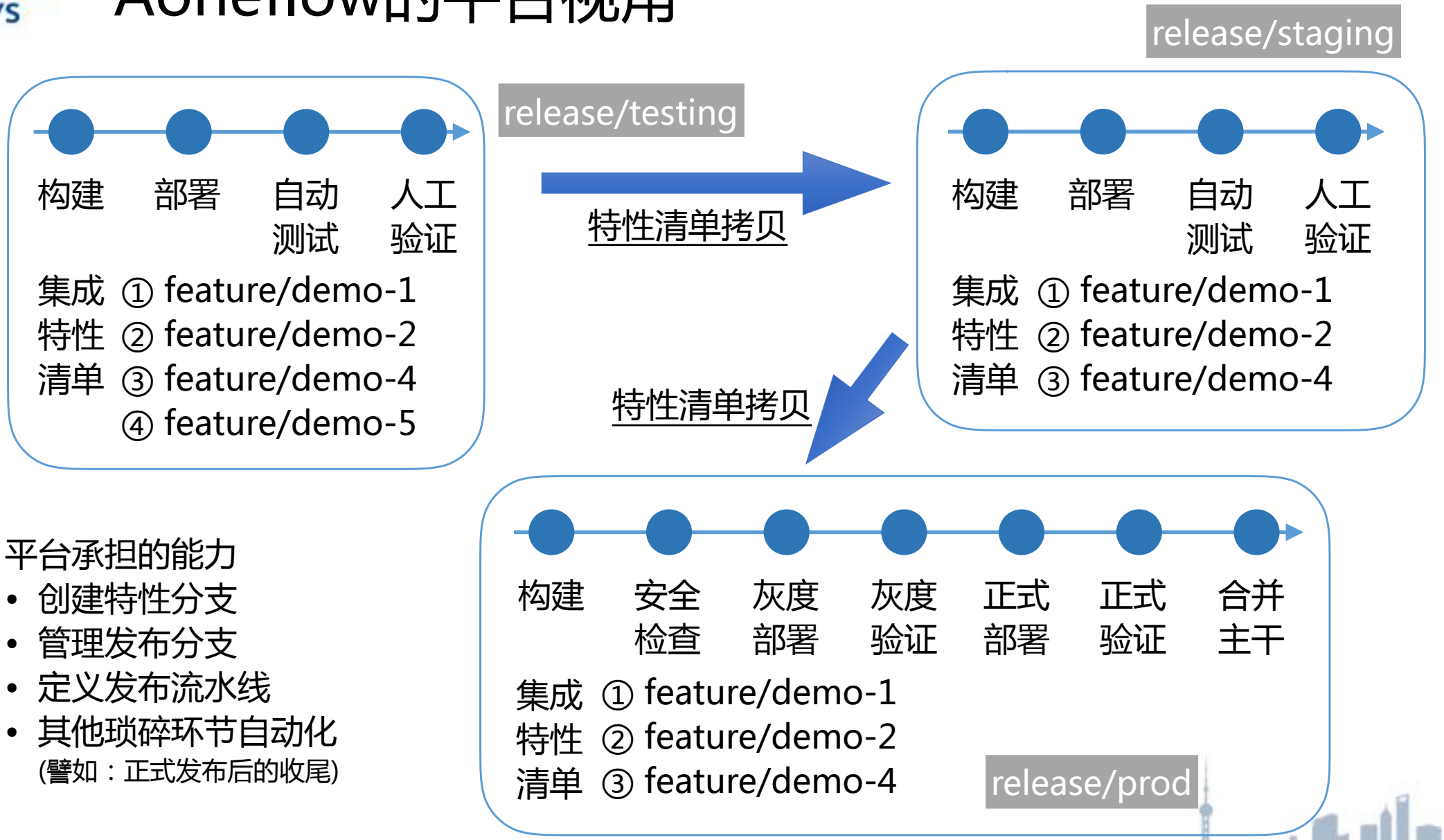
(master + demo-2 + demo-3 + demo-4)

(master + demo-2 + demo-3 + demo-4)

Aoneflow的开发者(用户)视角



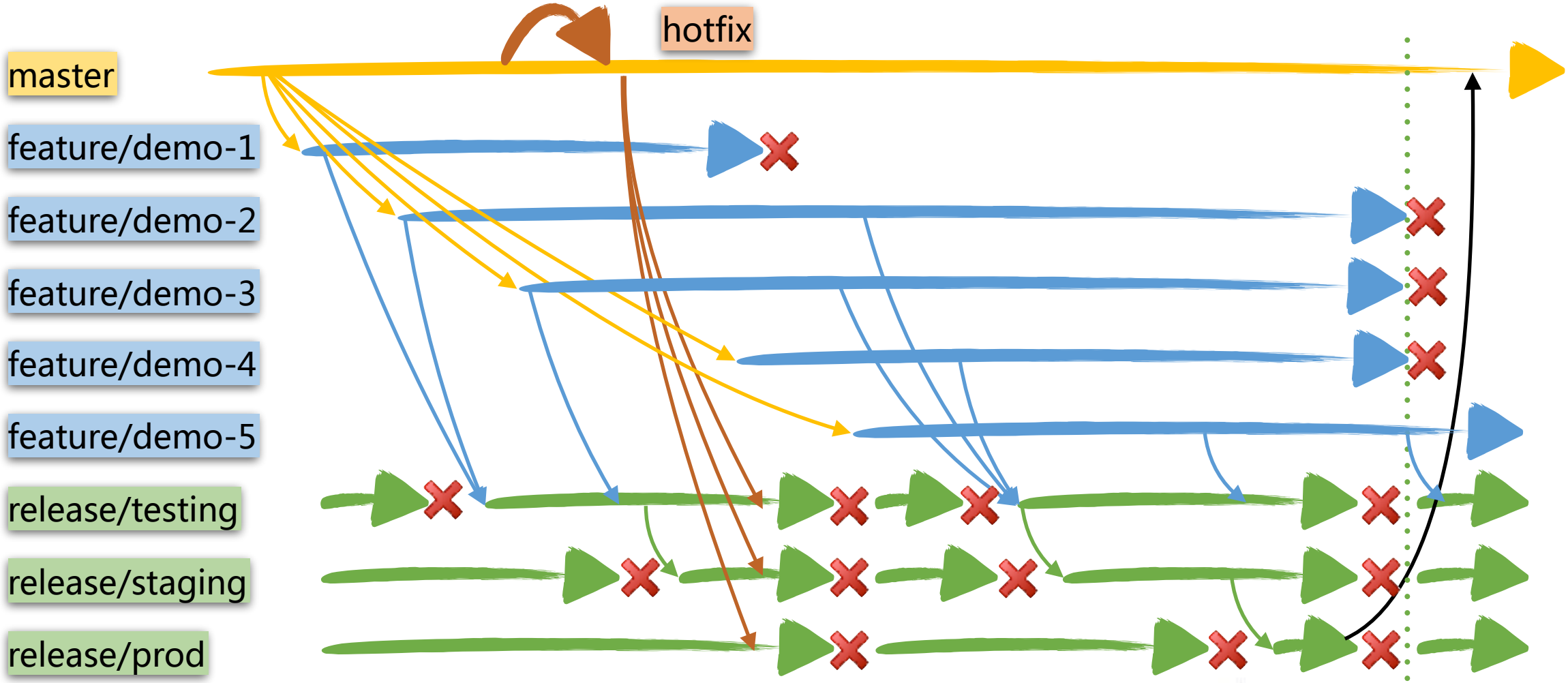
Aoneflow的平台视角



平台承担的能力

- 创建特性分支
- 管理发布分支
- 定义发布流水线
- 其他琐碎环节自动化
  (譬如：正式发布后的收尾)

Aoneflow的代码仓库视角



在用户看来始终存在的发布分支，其实一直在被平台频繁重建...

几种分支模式的比较

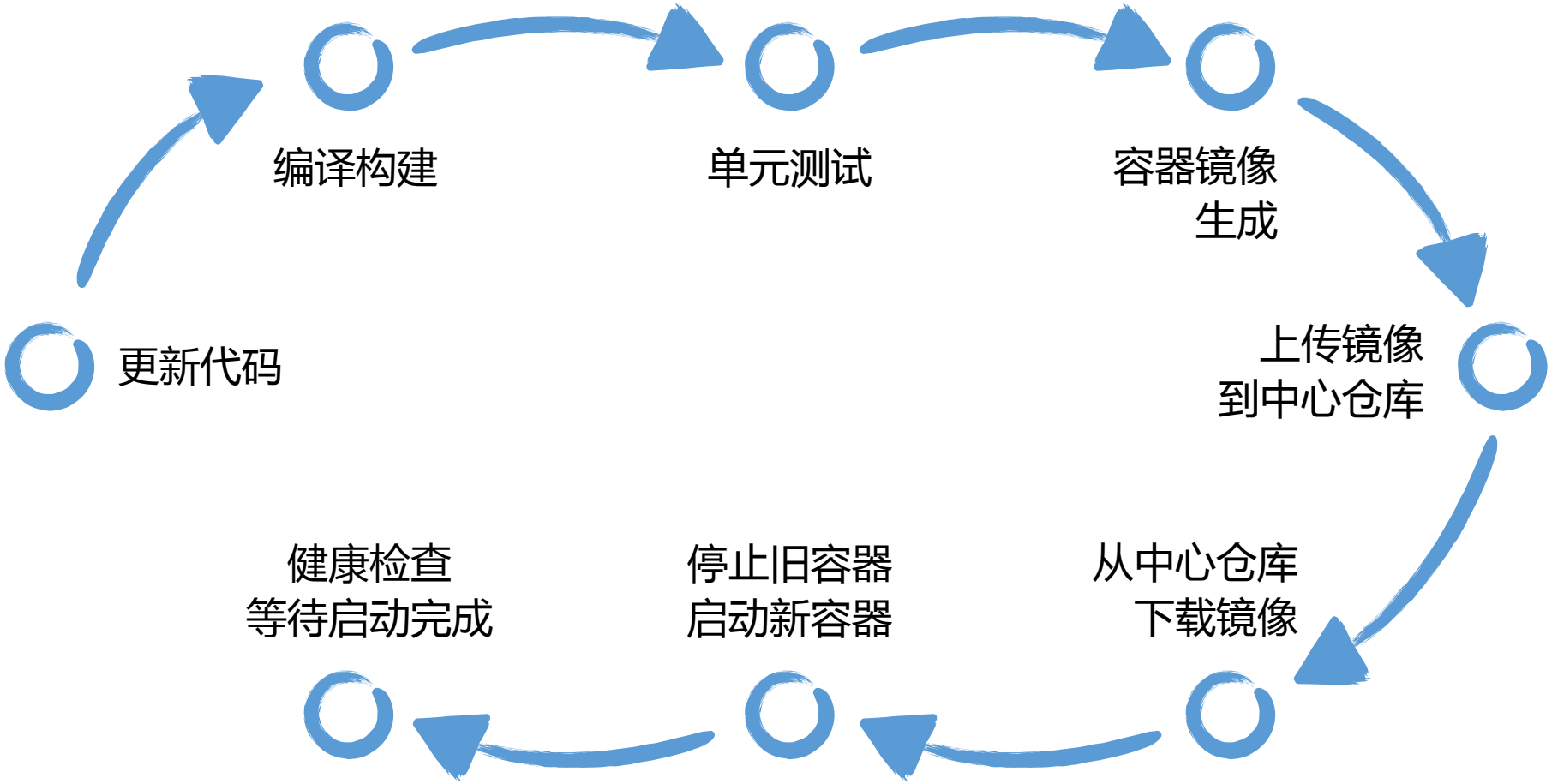
分支模式	集成频率	多版本并行开发	需求中途撤销	打包方式
单主干	所有提交立即集成	特性开关	手工剔除代码	一次打包多次部署
Aoneflow	指定多个特性的提交频繁集成	特性分支	删除特性分支	每个发布分支分别打包
Gitflow	整个特性完成后进行“大集成”	特性分支（且需严格控制特性合并时间）	合并开发分支前：删除特性分支 合并开发分支后：手工剔除代码	开发分支和发布分支分别打包

- 单主干 → 持续集成最佳实践+理想模式
- Aoneflow → 适应阿里现状的改进模式
- Gitflow → 历史遗产

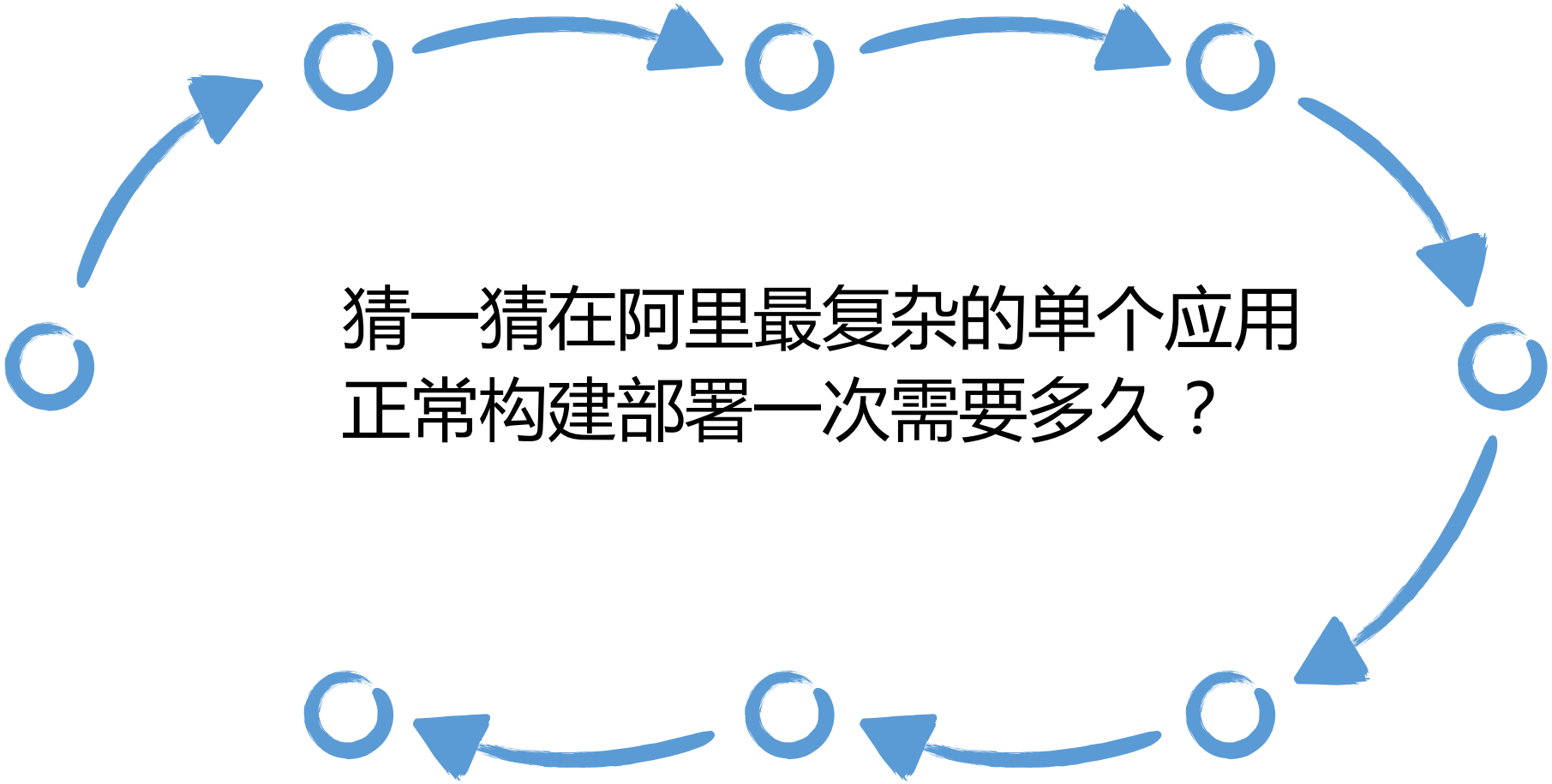
CHINA DEVOPS DAYS

构建部署，
从10分钟到10秒

典型的Java项目构建部署过程



典型的Java项目构建部署过程



一阶段优化：Maven依赖树缓存

构建慢的应用特点

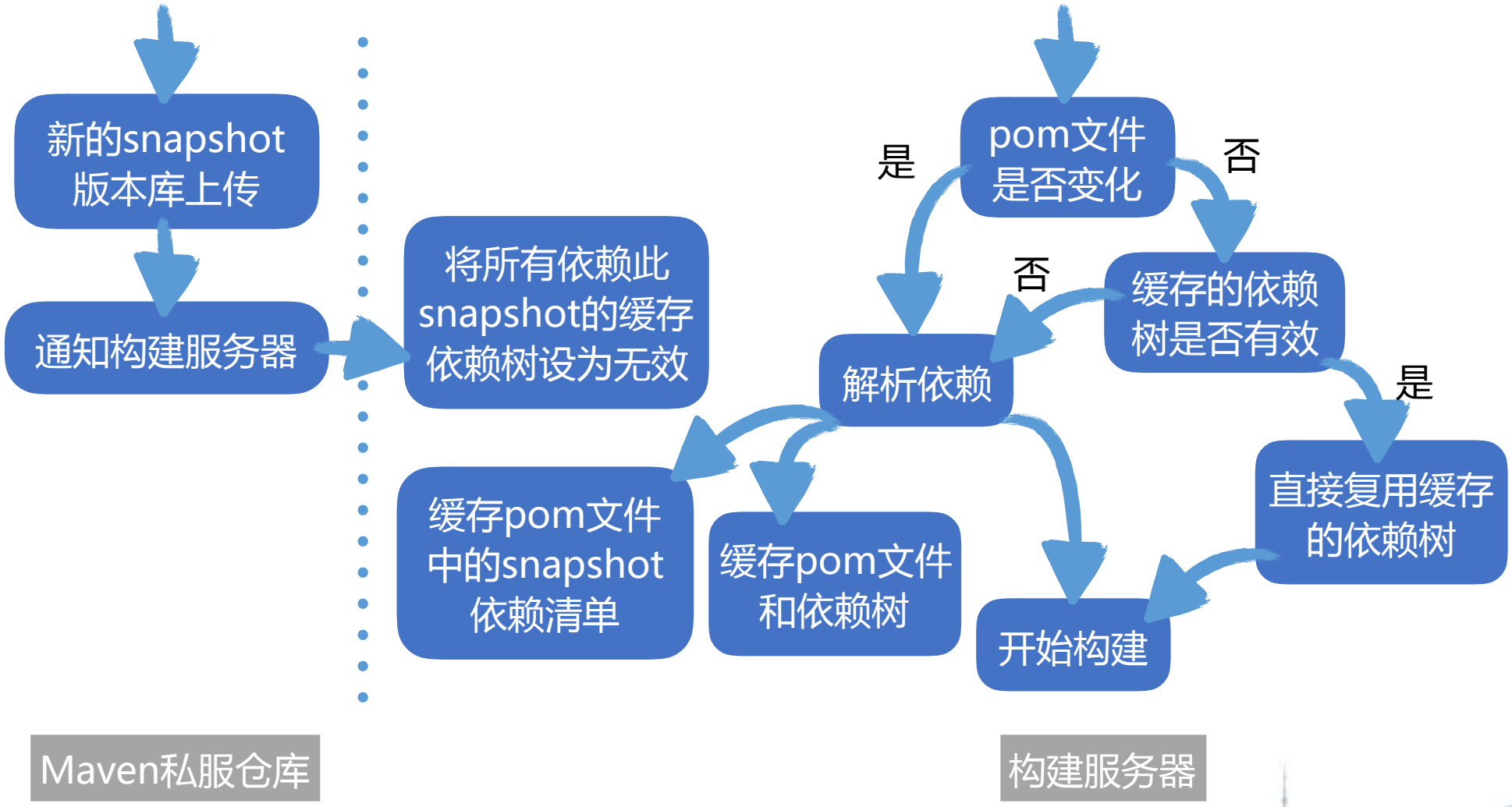
- 项目规模庞大
- 普遍使用Maven构建
- 依赖中包含SNAPSHOT版本的库

时间线
数据分析

依赖数目众多的三方包，
Maven解析大型依赖树非常耗时，
为了确保构建正确，每个SNAPSHOT版本
的依赖都需要连接中心仓库做时间戳对比，
进一步影响依赖树解析效率。

依赖树解析占据了整个编译过程80%的时间

一阶段优化：Maven依赖树缓存



一阶段优化：Maven依赖树缓存

一阶段优化结果

- 构建时间普遍缩短，部分典型应用构建时长缩短超过80%
- 大型应用部署总耗时减少一半

- 应用镜像的生成和传输成为优化后的部署性能瓶颈

二阶段优化：主包部署

容器给了开发者创造环境的自由，
但也带来了更大的发布包(镜像)和相应更长的包制作和传输时间...

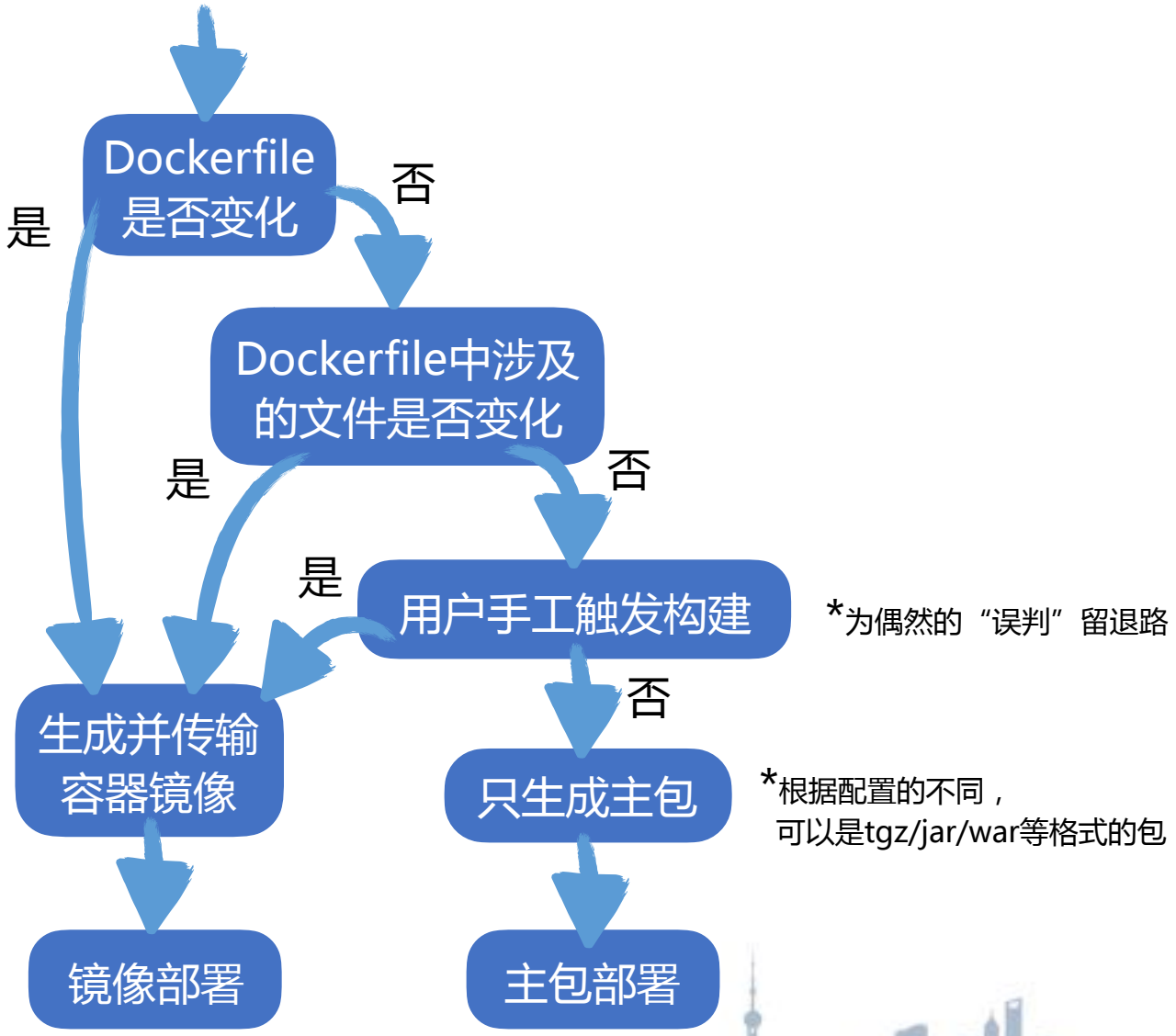
有人想到了阿里Pouch的P2P镜像传输协议，但那只适用于大量分发基础镜像

既要灵活又要速度，
我们最终做了一个违背“业界最佳实践”的决定

二阶段优化：主包部署

在一定条件下
对测试环境的服务
放弃不可变基础设施

在容器上进行主包部署



二阶段优化结果

- 测试环境发包速度从分钟级降到秒级
- 从部分灰度用户推广到集团大部分BU

- 应用自身重启耗时成为最后时间瓶颈

三阶段优化：增量部署

服务启动慢不是平台的错，
但它的确成为了拖慢了部署总时长的最后瓶颈...

所以...Java应用部署能够不重启吗？

JVM有个Hotswap机制
在特定的条件下，似乎值得一试

三阶段优化：增量部署

平均10分钟

最快10秒钟

在符合主包部署条件的基础上
只修改方法体或新增类
的测试环境服务

使用Java Hotswap热部署
无需重启服务

*原始Hotswap只支持修改方法体，
这里使用的是改良版的Hotswap机制

三阶段优化结果

- 整个构建部署过程最快10秒
- 代码提交测试环境“立即”生效

CHINA DEVOPS DAYS

服务集群，
从虚拟硬件到虚拟服务

虚拟化实质是资源的隔离和复用

物理机

无虚拟化

1 → 1

虚拟机

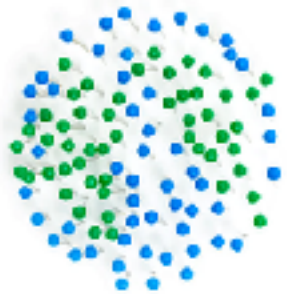
硬件级虚拟化

1 → 10

容器

软件级虚拟化

1 → 100

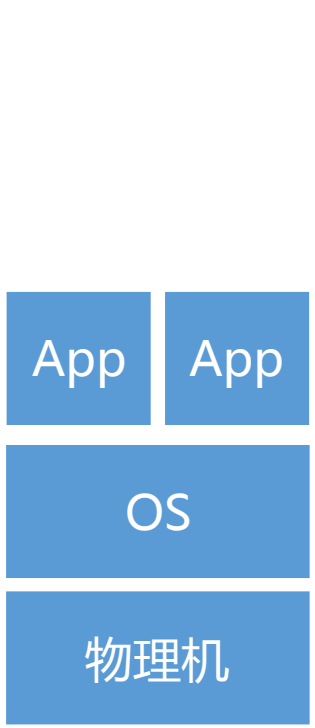


特性环境

服务级虚拟化

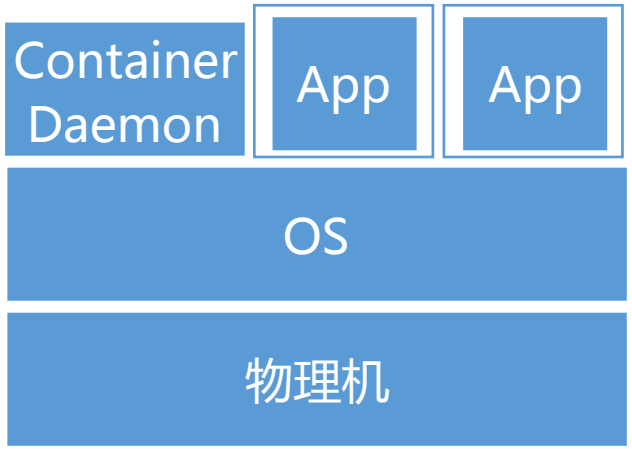
1 → 1000 *搭配容器使用

虚拟机和容器的虚拟化方式



物理机

虚拟机



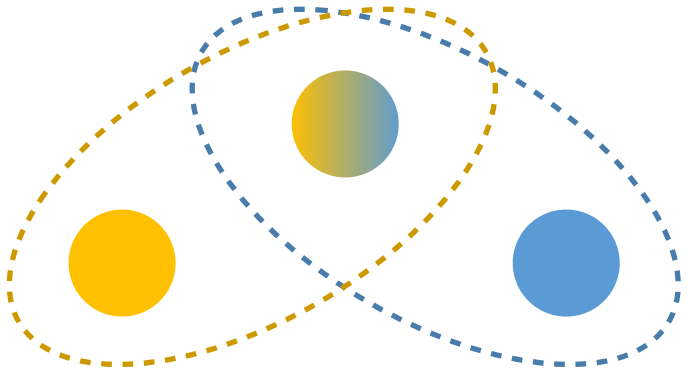
容器

还能更
轻量吗？

什么是服务级虚拟化

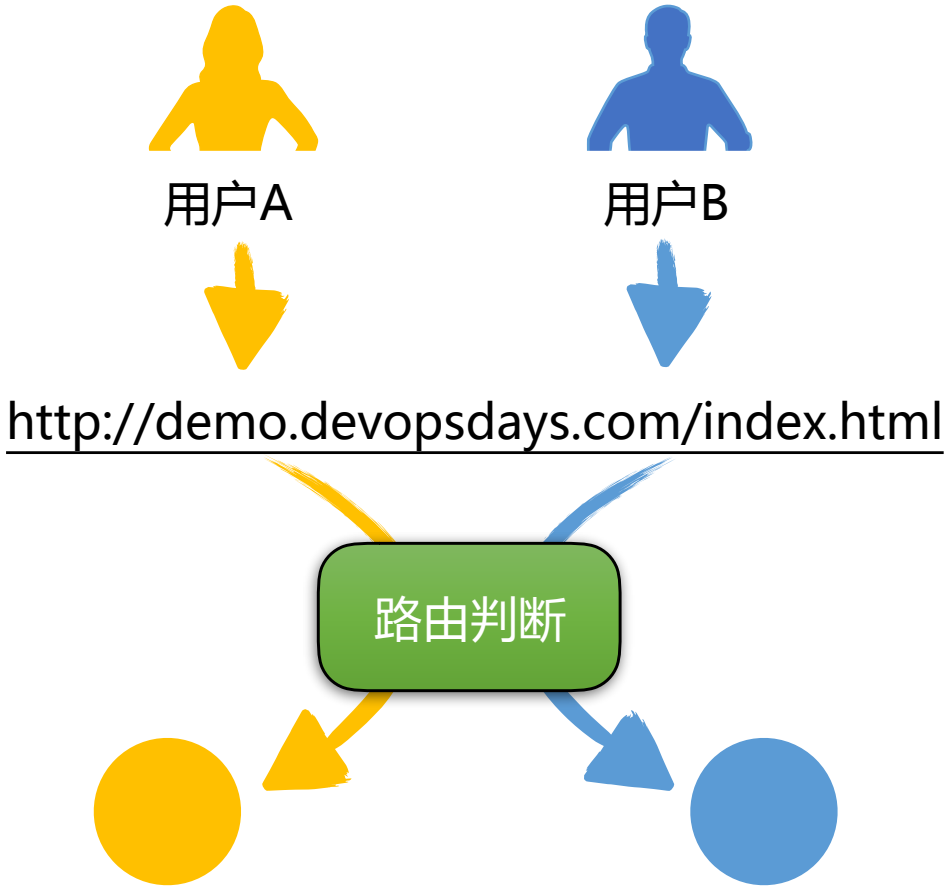
容器已经将计算资源复用的损耗降到极低，
但容器中的服务似乎总还是有许多空闲的时候...

能否让一个服务分身到多个集群里发挥作用，让这些闲着的时段都利用起来呢？
但是消息会串号吧？

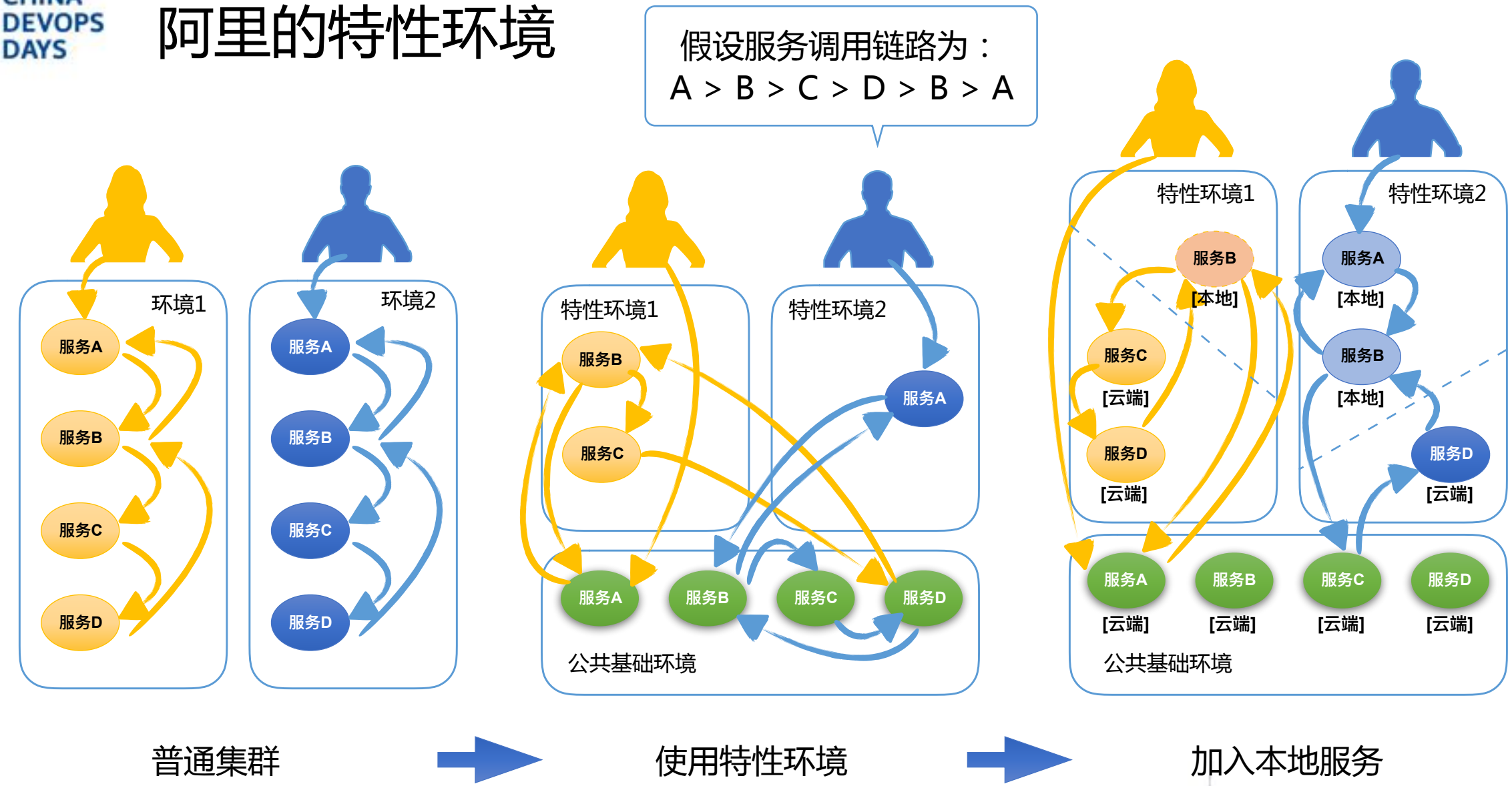


虚拟链路其实我们都不陌生

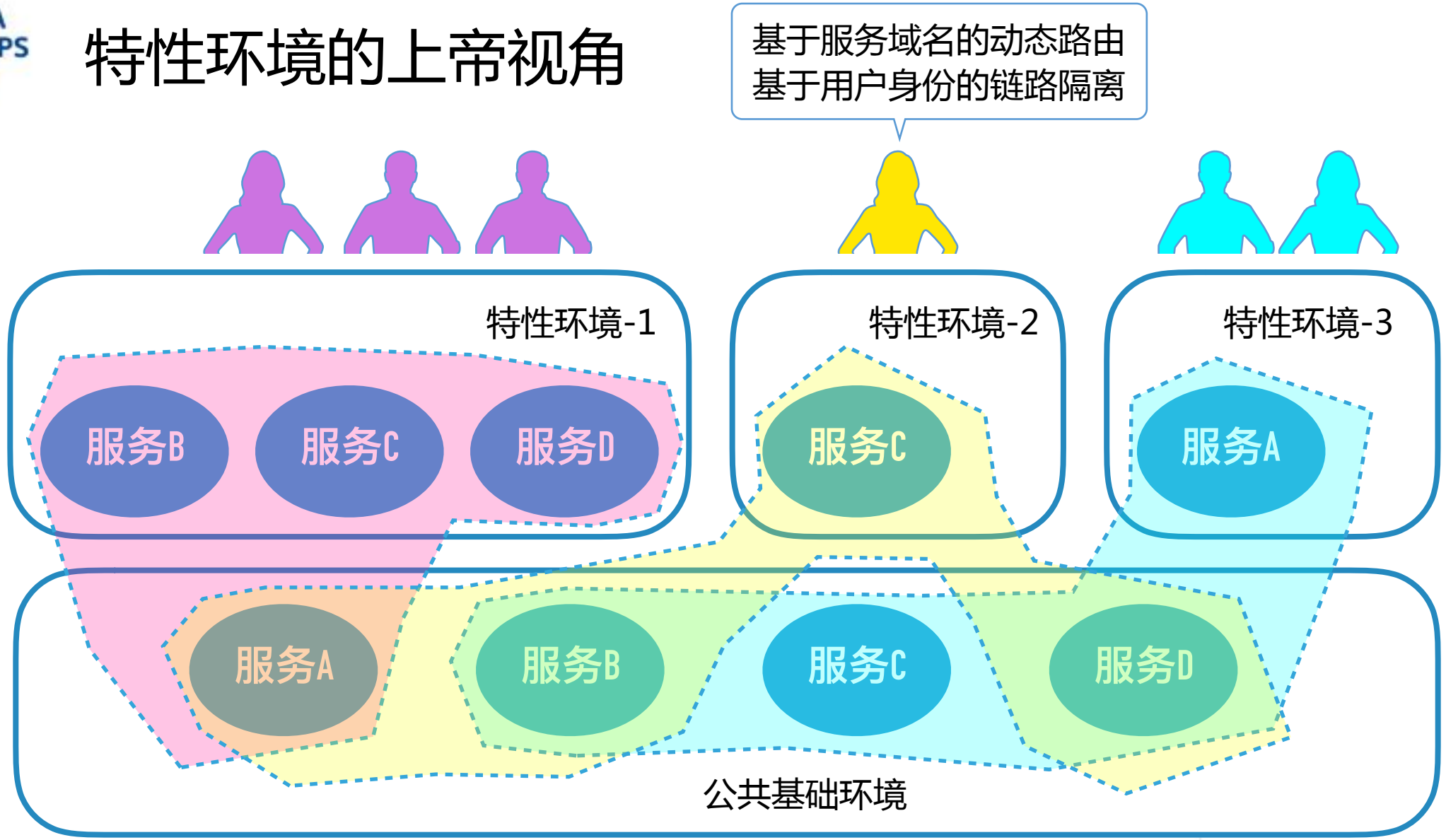
- 金丝雀发布
- 灰度部署
- A/B测试
- …



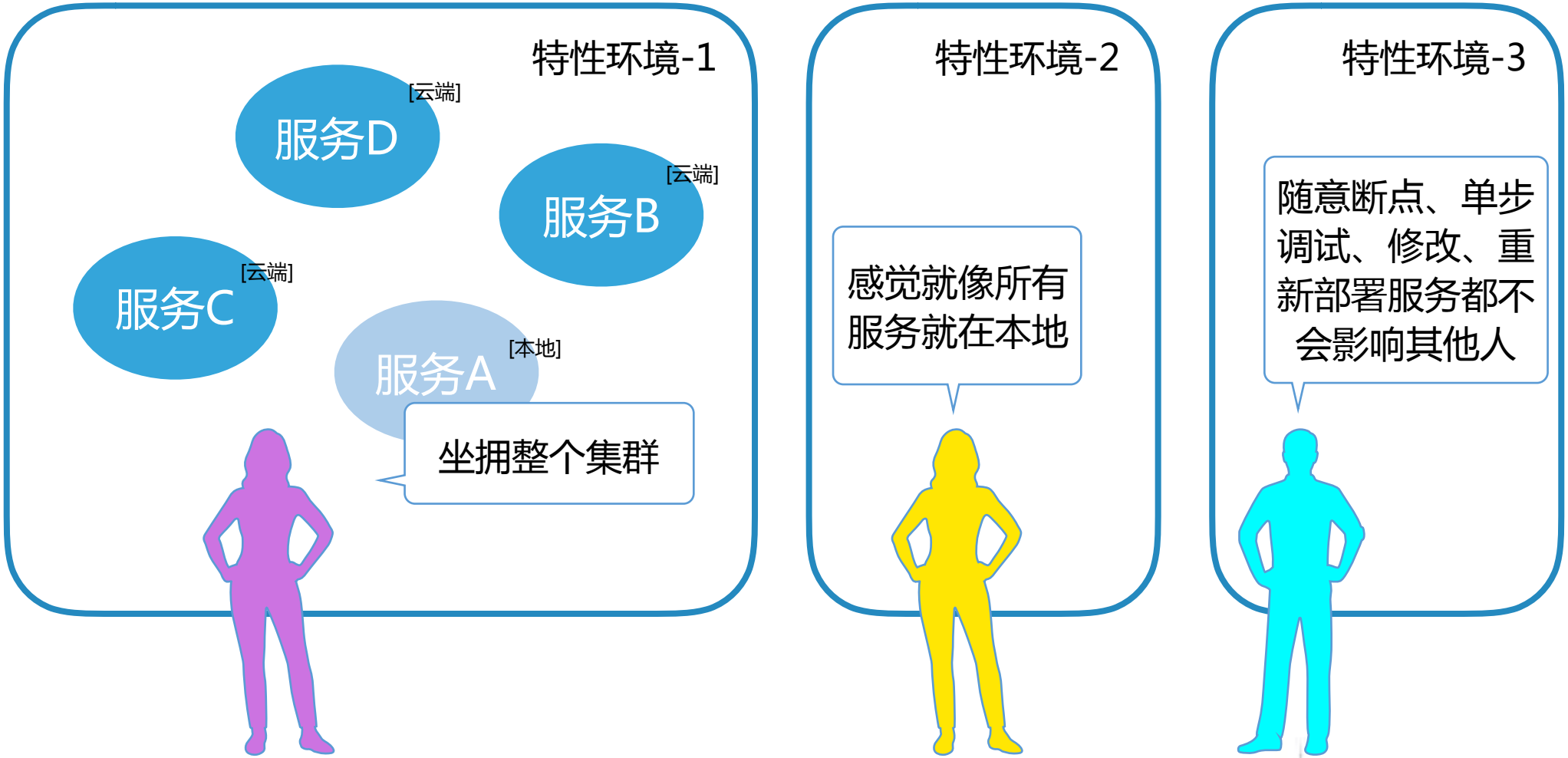
阿里的特性环境


假设服务调用链路为：
A > B > C > D > B > A
环境1
服务A
服务B
服务C
服务D
环境2
服务A
服务B
服务C
服务D
特性环境1
服务B
服务C
特性环境2
服务A
服务A
服务B
服务C
服务D
公共基础环境
特性环境1
服务B
[本地]
服务C
[云端]
服务D
[云端]
特性环境2
服务A
[本地]
服务B
[本地]
服务D
[云端]
服务A
[云端]
服务B
[云端]
服务C
[云端]
服务D
[云端]
公共基础环境
普通集群
使用特性环境
加入本地服务

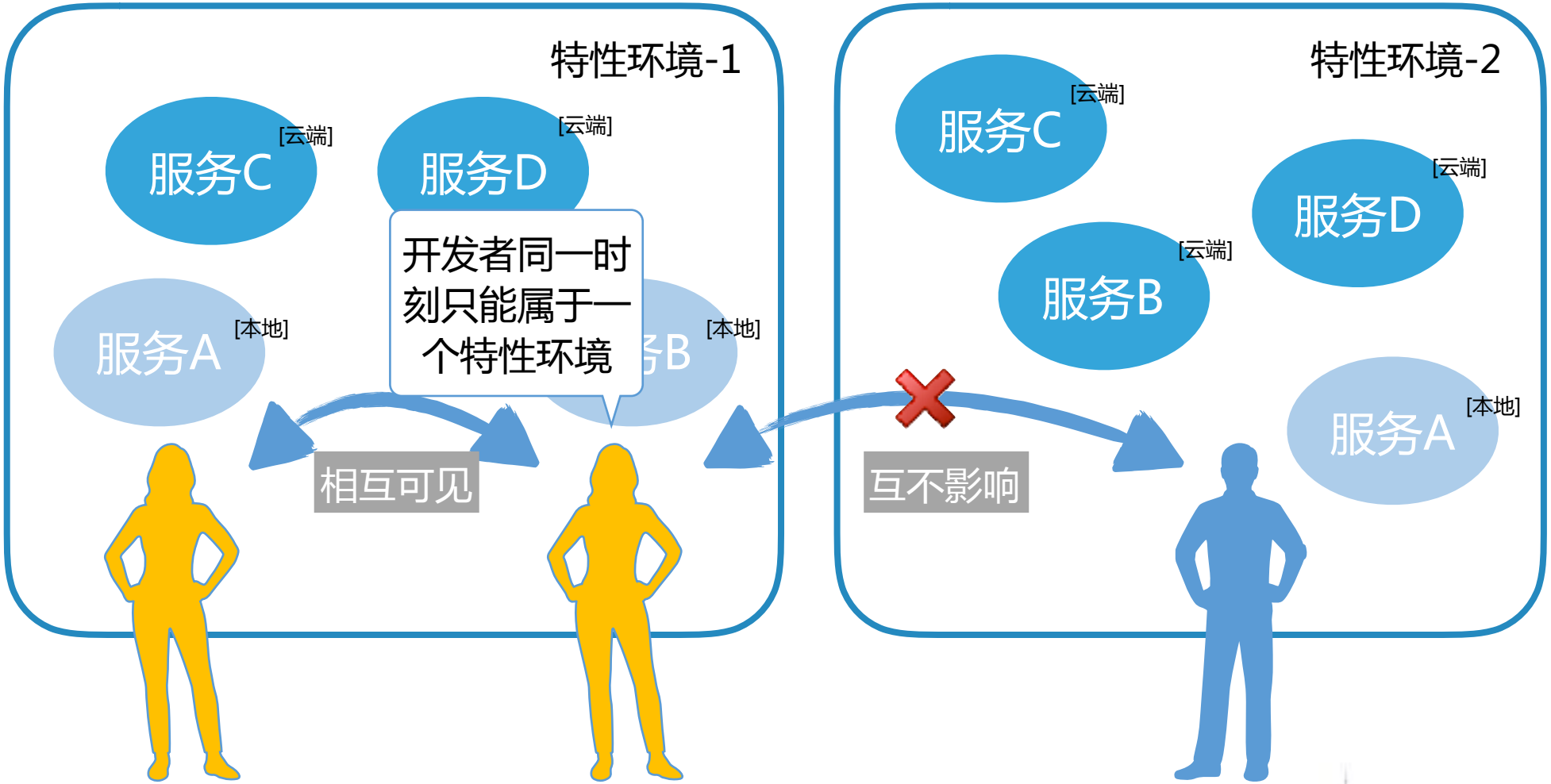
特性环境的上帝视角


基于服务域名的动态路由
基于用户身份的链路隔离
特性环境-1
特性环境-2
特性环境-3
服务B
服务C
服务D
服务C
服务A
服务A
服务B
服务C
服务D
公共基础环境

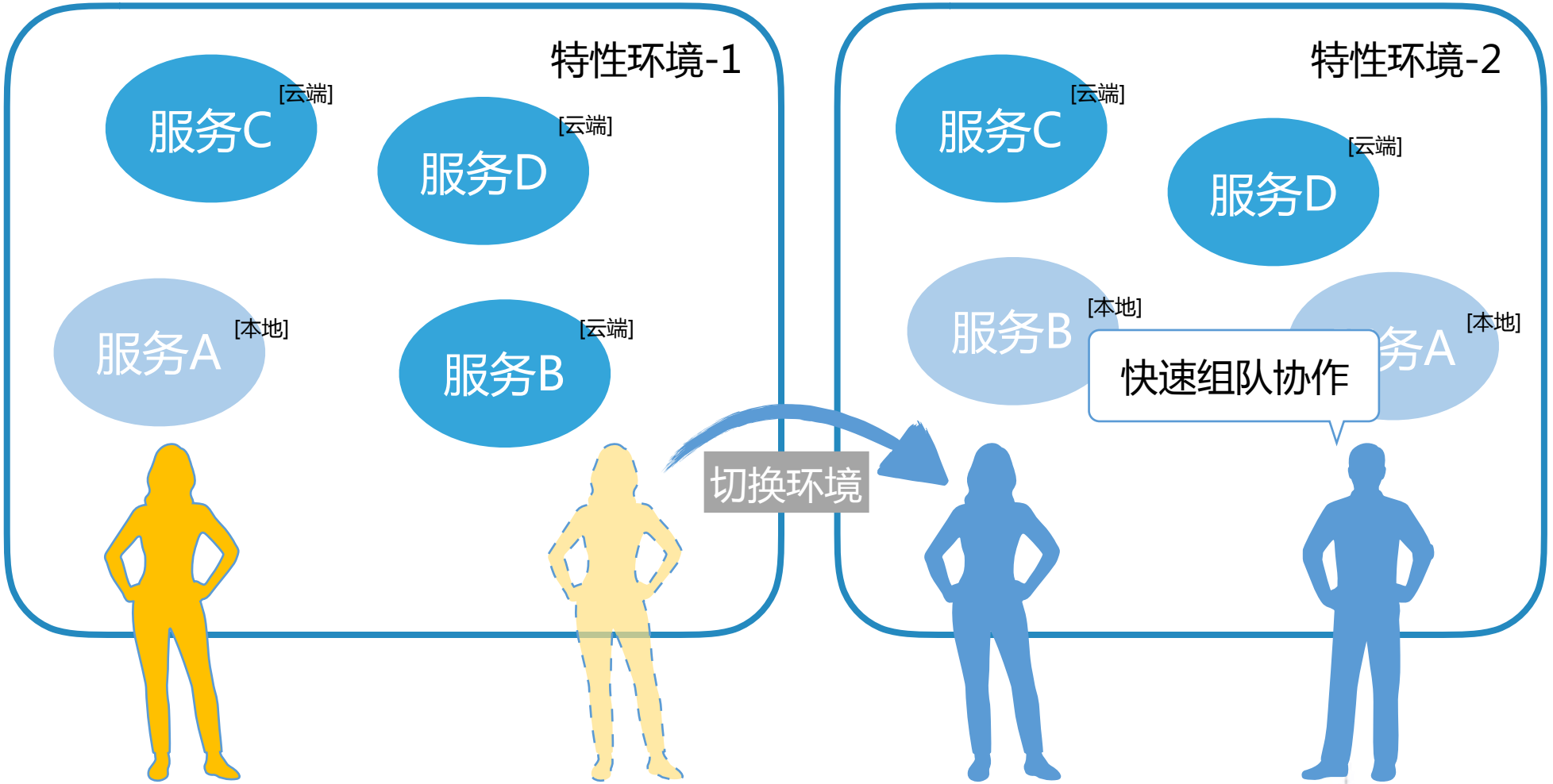
特性环境的开发者视角


特性环境-1
服务D
[云端]
服务B
[云端]
服务C
[云端]
服务A
[本地]
坐拥整个集群
特性环境-2
感觉就像所有服务就在本地
特性环境-3
随意断点、单步调试、修改、重新部署服务都不会影响其他人

特性环境的团队视角


特性环境-1
服务C
[云端]
服务D
[云端]
开发者同一时刻只能属于一个特性环境
服务A
[本地]
[本地]
相互可见
特性环境-2
服务C
[云端]
服务D
[云端]
服务B
[云端]
服务A
[本地]
互不影响

特性环境的团队视角


特性环境-1
服务C
[云端]
服务D
[云端]
服务A
[本地]
服务B
[云端]
切换环境
特性环境-2
服务C
[云端]
服务D
[云端]
服务B
[本地]
务A
[本地]
快速组队协作

几种虚拟化方式的比较

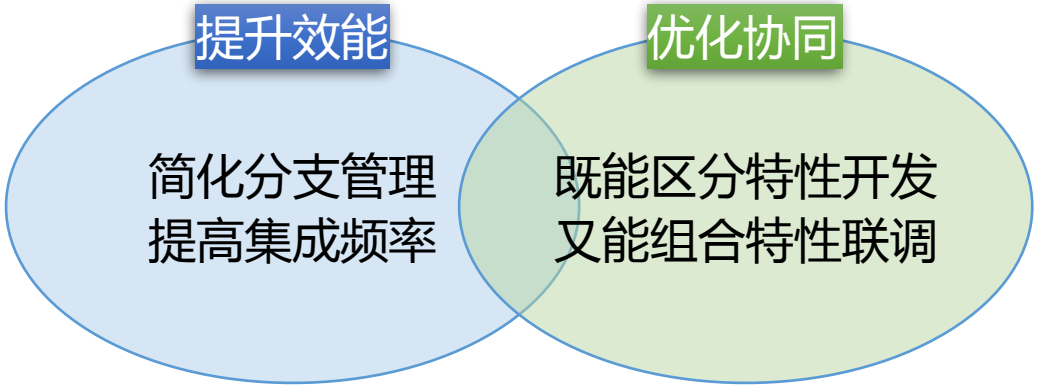
虚拟化方式	复用资源类型	通用性	隔离度	复用度
硬件级虚拟化（虚拟机）	设备资源复用（CPU和内存等）	强（可跨操作系统类型运行任何应用）	高（资源完全隔离）	低（通常低于1:10）
软件级虚拟化（容器）	设备资源复用（CPU和内存等）	较强（可运行相同内核类型的任何应用）	较高（共享内核，命名空间隔离）	较高（通常低于1:100，可与虚拟机联用）
服务级虚拟化（特性环境）	服务资源复用	弱（仅限同类服务集群之间共享应用）	低（仅具有访问路由级别的隔离）	高（可达1:100以上，通常与容器联用）

CHINA DEVOPS DAYS

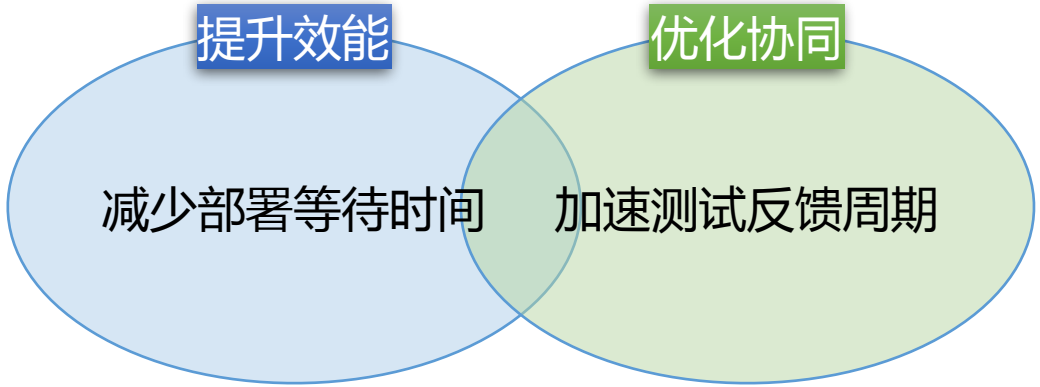
DevOps，
回归效能与协同

DevOps的核心在于效能和协同

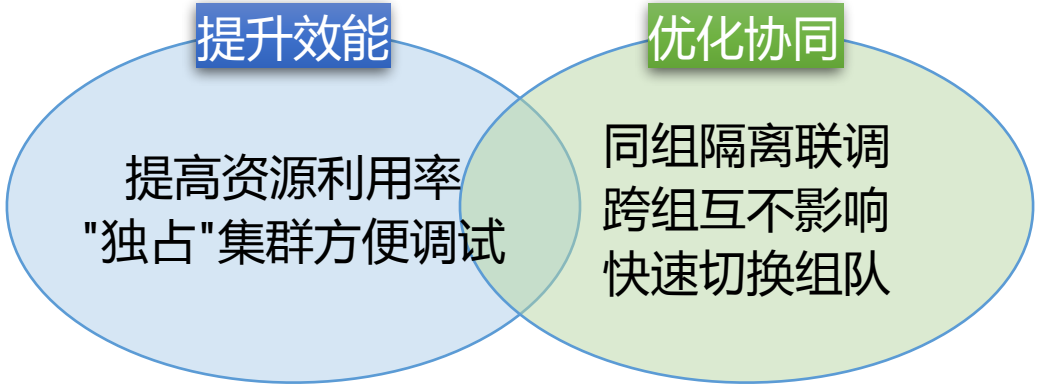
Aoneflow



缓存构建/主包部署/增量部署

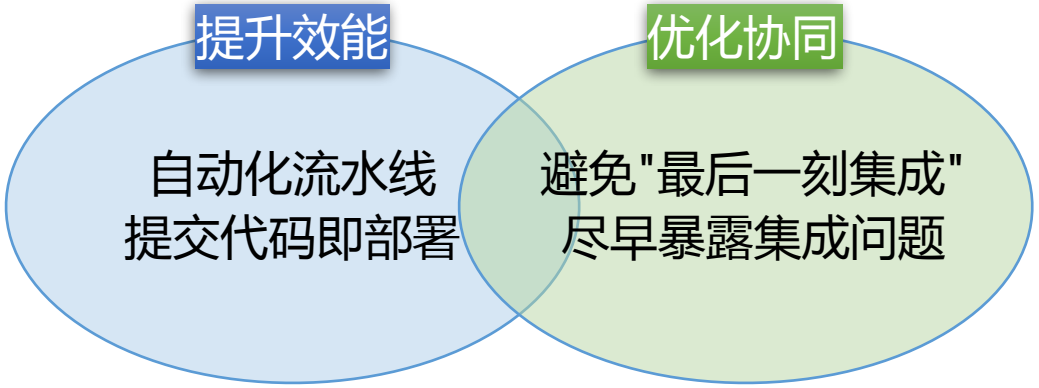


特性环境

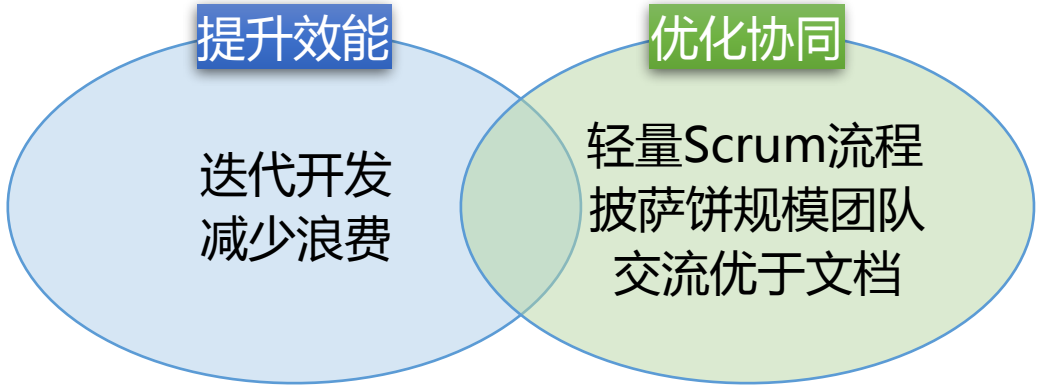


这些与DevOps相关的话题

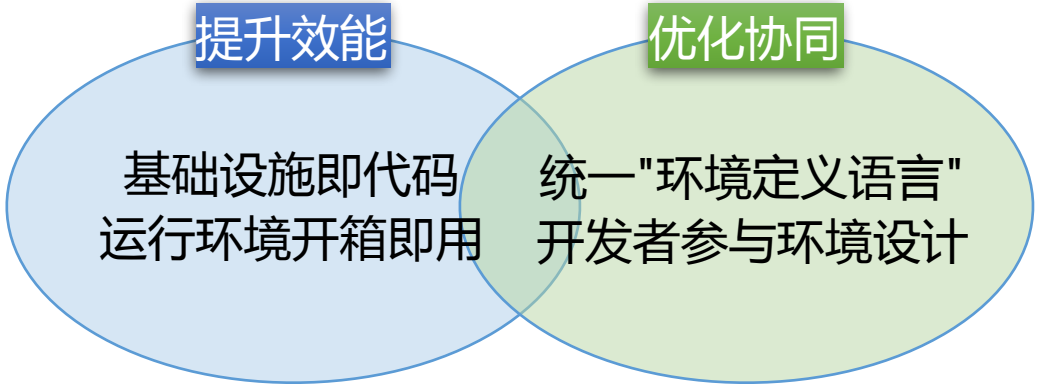
持续集成



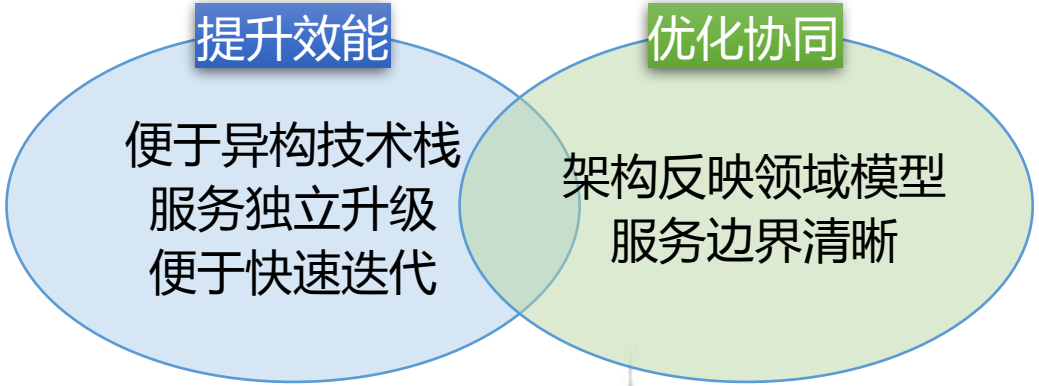
敏捷精益



容器技术



微服务设计



CHINA DEVOPS DAYS

只要目标与方向清晰，
创新不需要规则束缚

CHINA DEVOPS DAYS

"最佳实践"都是别人的，
适合自己才是最好的！

CHINA DEVOPS DAYS

THANKS

We Are Hiring ^^: jinji.lf@alibaba-inc.com

Website：
chinadevopsdays.org/

Global Website:
www.devopsdays.org/events/2018-shanghai/

Official Email:
organizers-shanghai-2018@devopsdays.org

Official Wechat