QCon 全球软件开发大会 【北京站】2016

今日头条
服务化探索及实践历程

今日头条 | 夏绪宏

自我介绍

夏绪宏

今日头条

Baidu

PLT

Performance

HHVM

PHP Committer

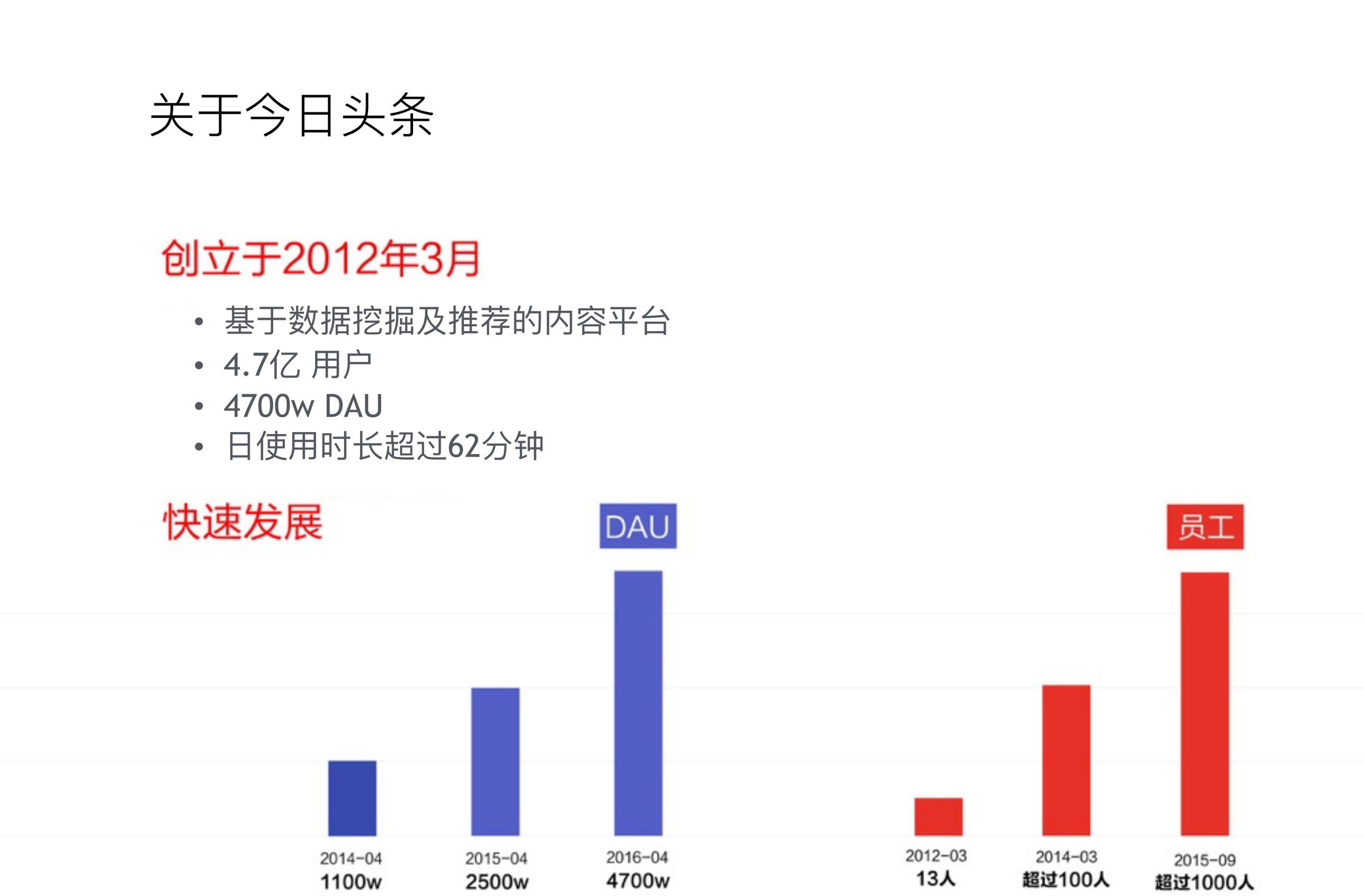
Cloud Computing

关于今日头条

创立于2012年3月

- 基于数据挖掘及推荐的内容平台
- 4.7亿 用户
- 4700w DAU
- 日使用时长超过62分钟

快速发展



01 复杂度、耦合及抽象

微服务架构：
SOA，原则，得失

今日头条的架构
演进及服务化历程

服务化不简单：
基础设施的构建

软件架构发展史

直接操作硬件

抽象硬件

民用计算

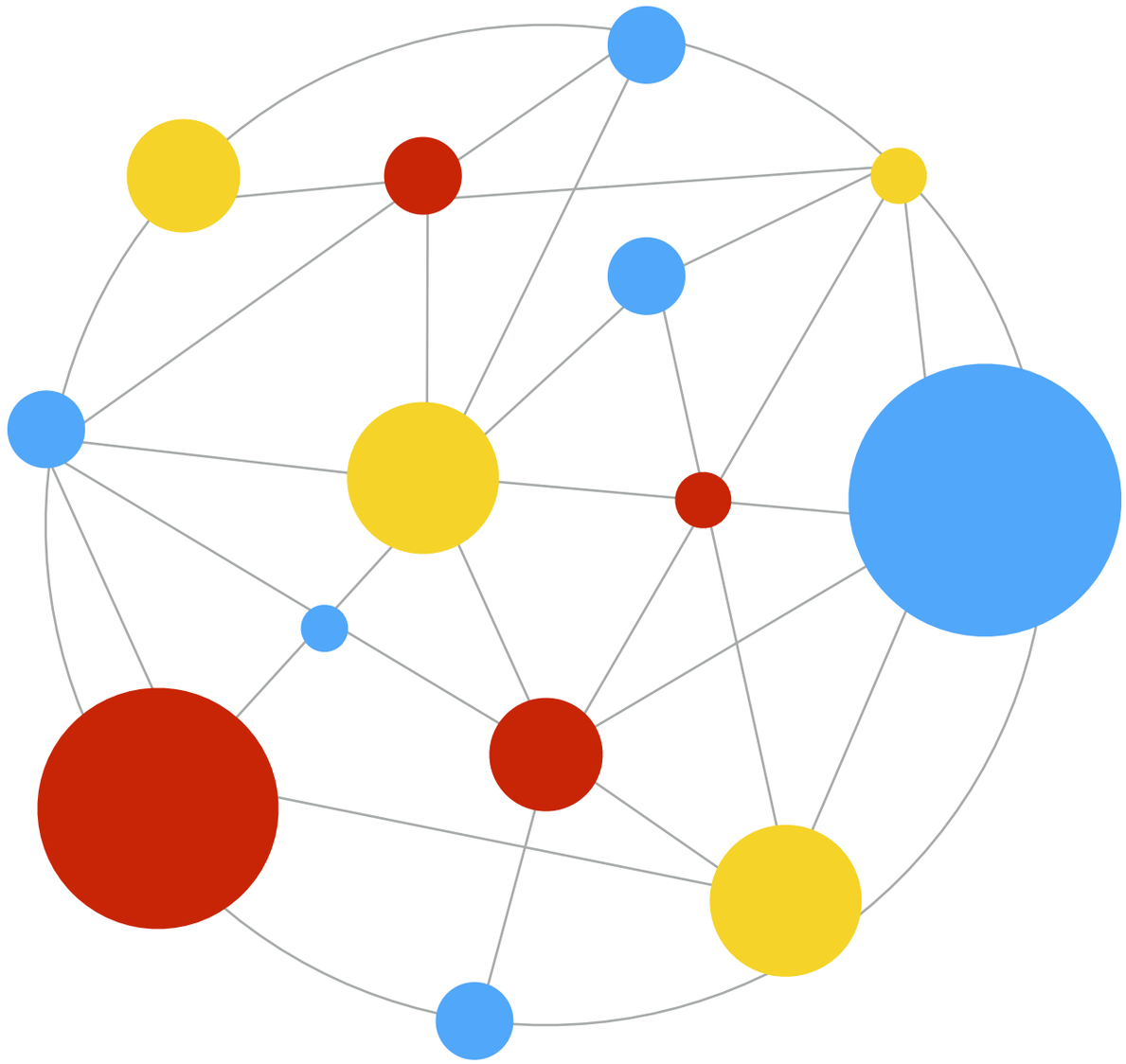
web网络

规模和终端量的爆发，改变生活

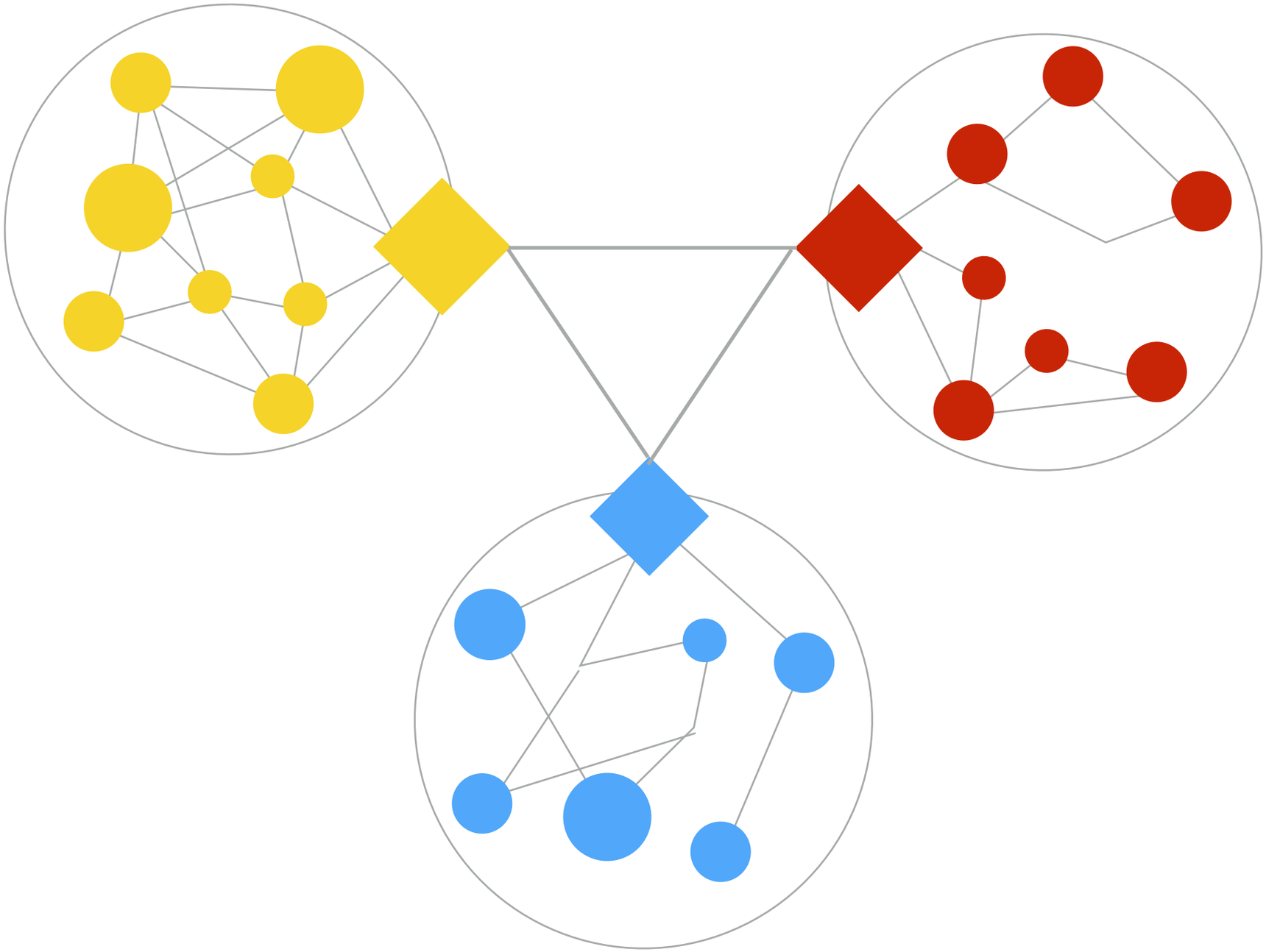
进一步的和生活结合，变成水电一样的基础设施

垂直规模到水平规模的变化

关于复杂度



关于抽象



耦合无处不在

02

微服务架构：SOA，原则，得失

复杂度、
耦合及抽象

今日头条的架构
演进及服务化历程

服务化不简单：
基础设施的构建

微服务架构

一种架构方式

API及周边设施

微服务架构

In short, the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API.

These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies.

-- James Lewis and Martin Fowler

微服务架构

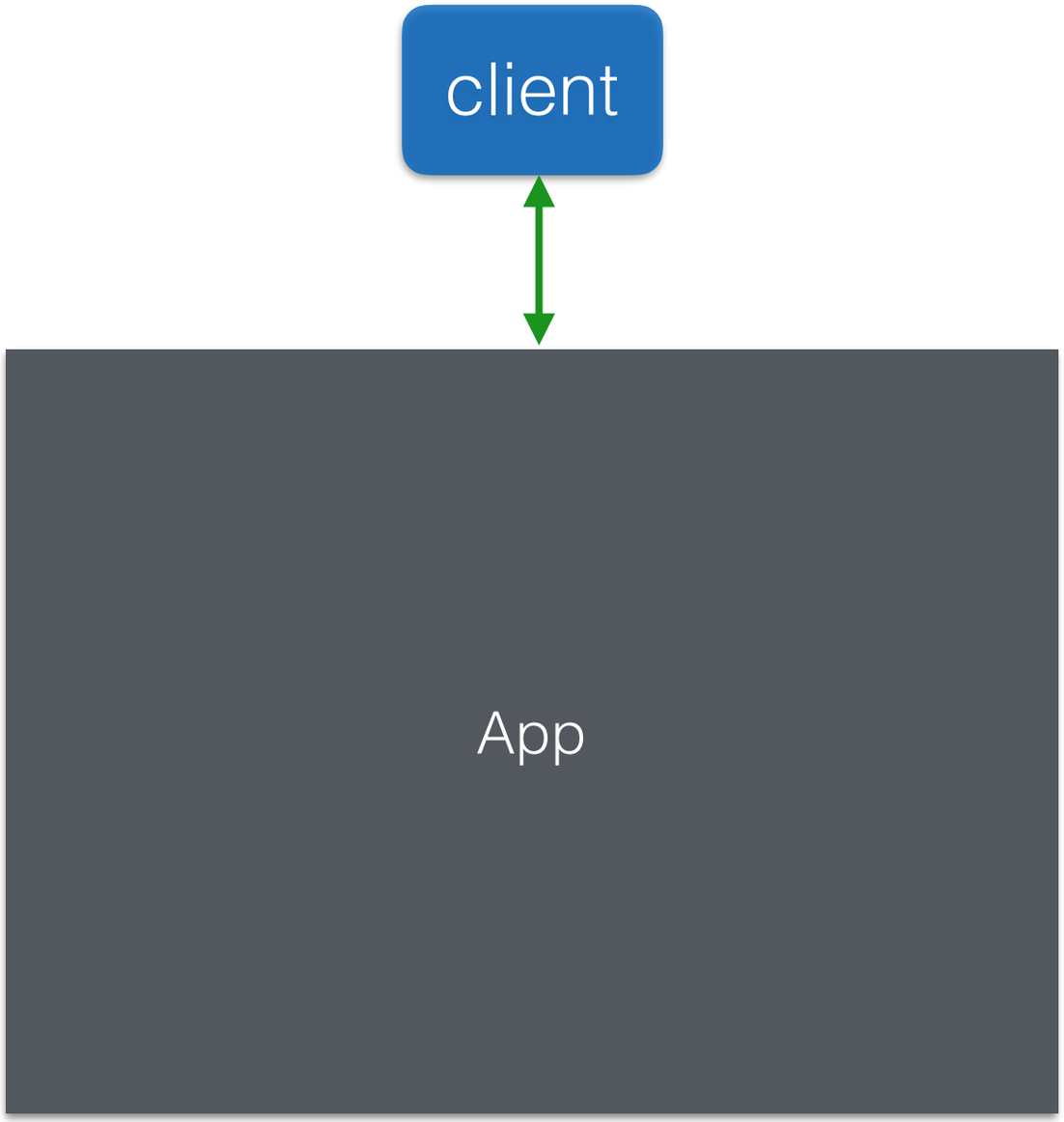
- own process：进程解耦
- lightweight：易于管理和理解
- built around business capabilities：自包含
- independently deployable：部署解耦
- automated deployment machinery：自动化

微服务架构

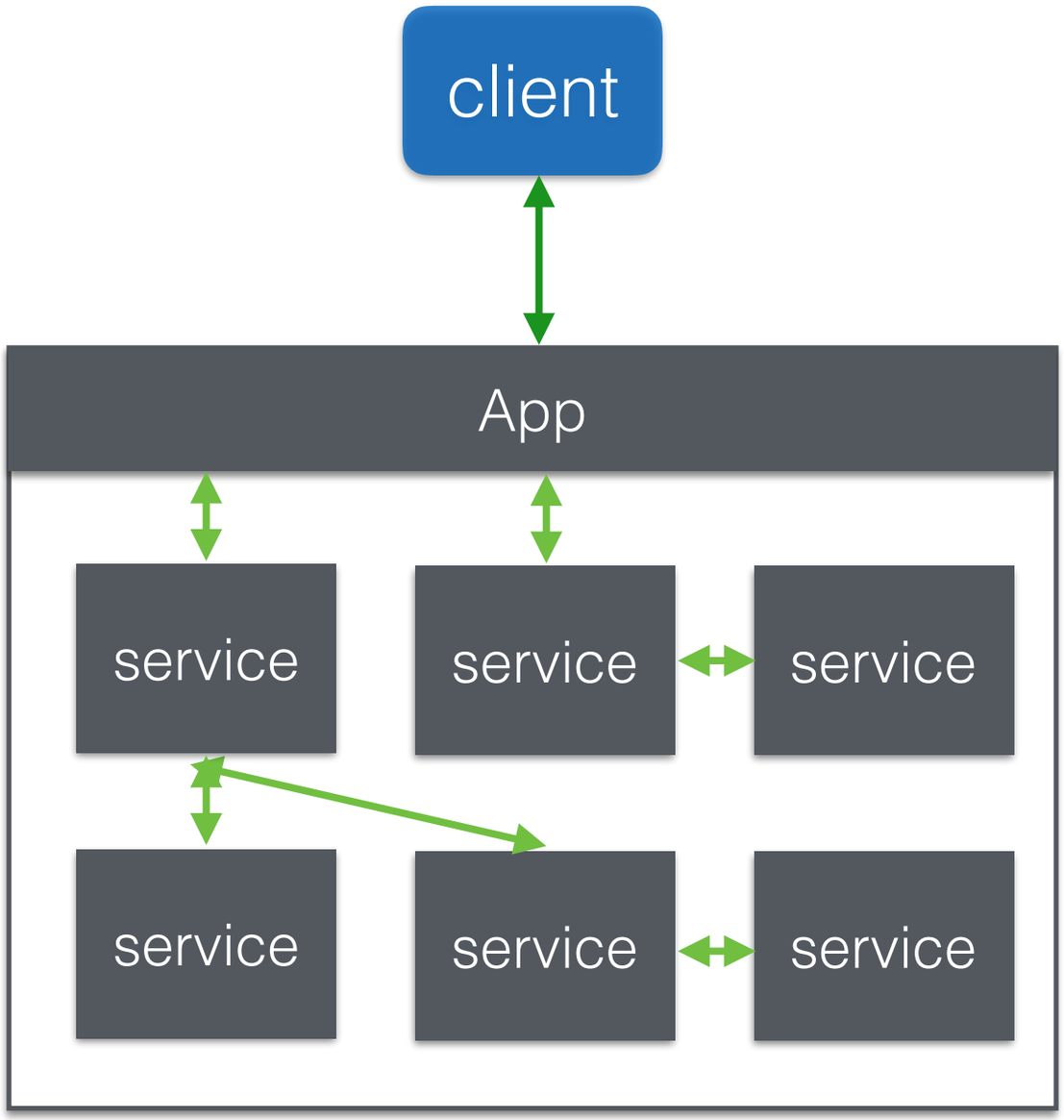
- **Microservices**
  - 通过服务化组件化
  - 围绕业务组织服务
  - 去中心
  - 去中心存储
  - 基础设施自动化
  - 考虑错误
  - 可进化的设计
  - …

微服务架构

- SOA: 服务方式
- Microservices: 服务构建方式



SOA



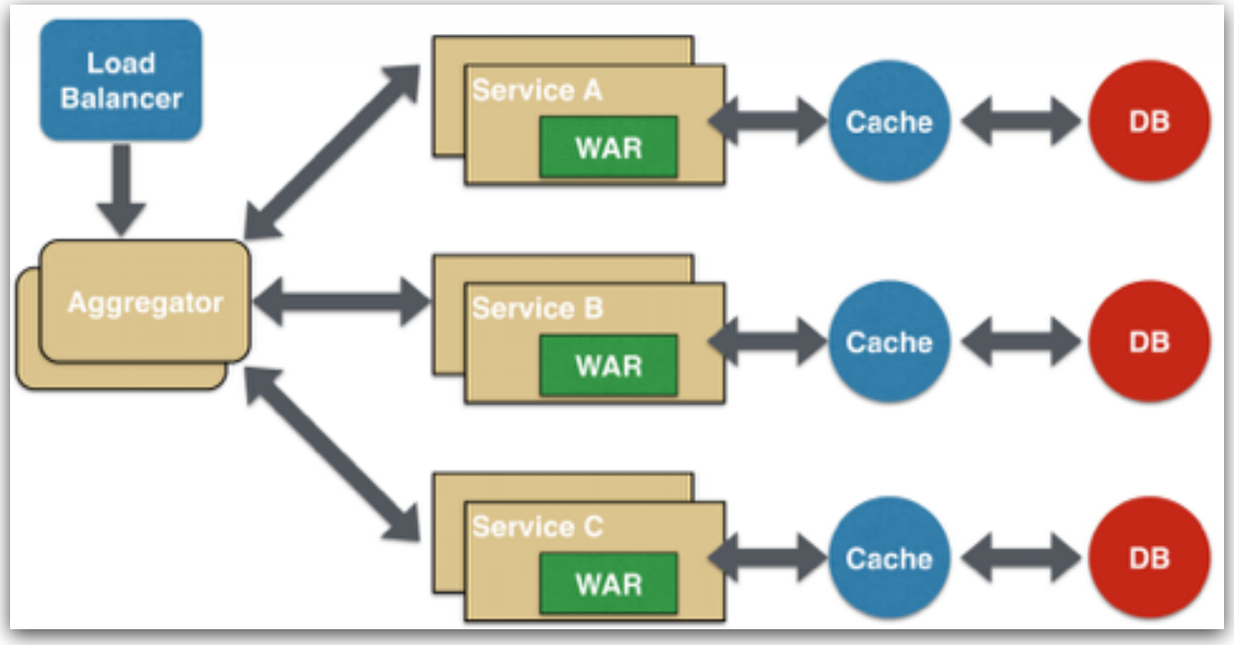
Microservices

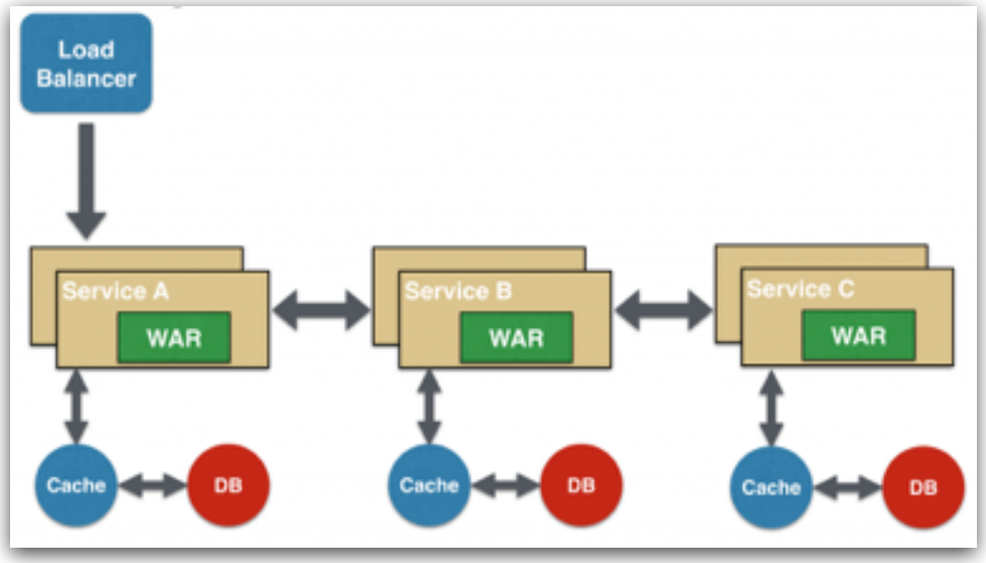
微服务原则

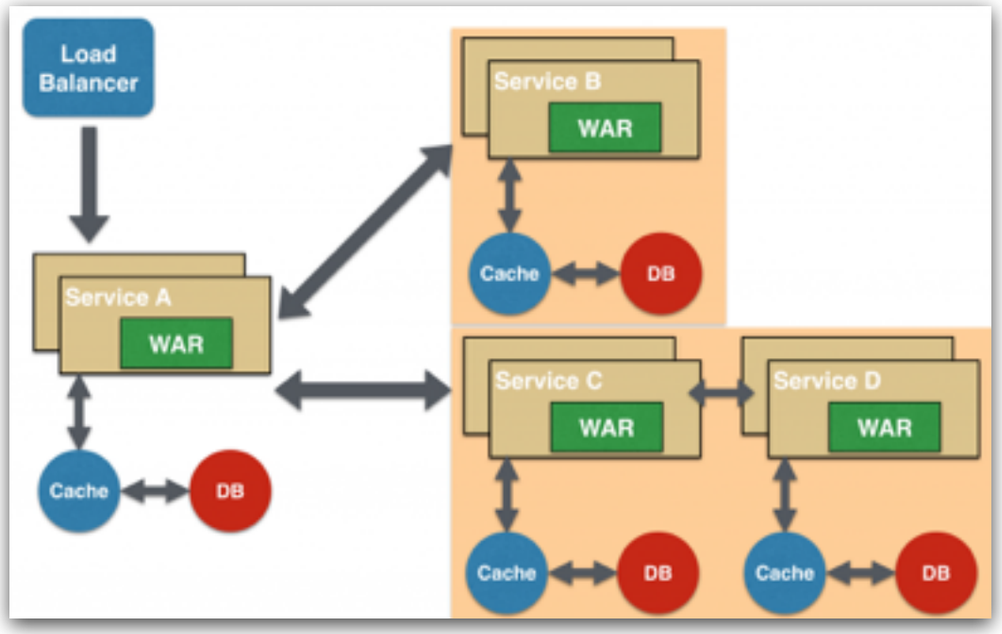
- **设计原则**
  - API语言无关性
  - 强接口约束性
  - 高内聚
  - 服务间的正交性
- **拆分原则**
  - 尽可能的小，又不能太小 -_-|||
  - 能够独立部署

- **有时，原则就是用来打破的**
  - **私有服务**：使用方可控，不兼容升级、优化客户端
  - **其他折衷**：性能、历史原因、etc...

微服务构建方式







http://www.infoq.com/cn/news/2015/04/micro-service-architecture

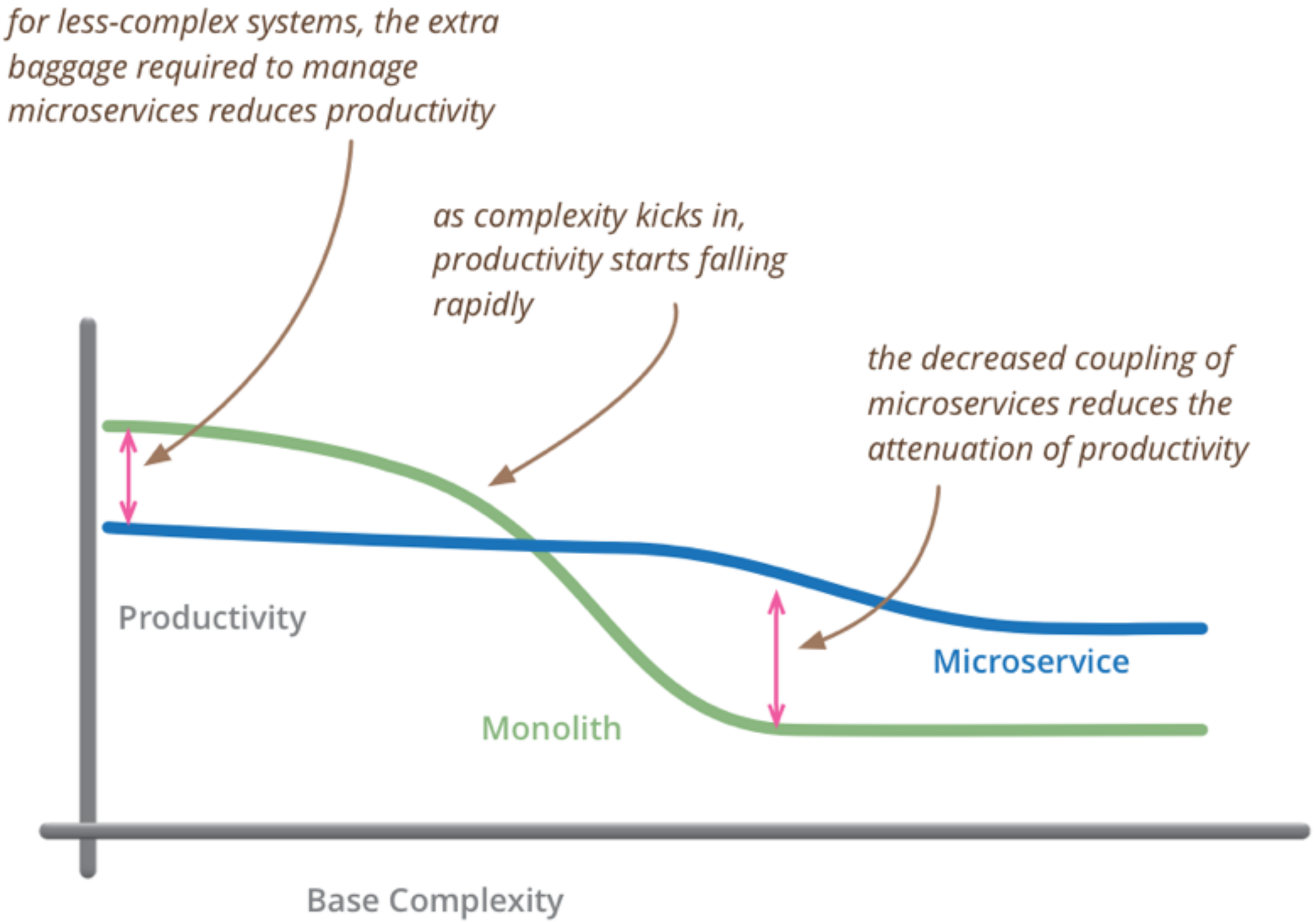
要不要采用微服务架构?

或许已经在用微服务架构了

或许可以更接近微服务架构

或许你还不需要微服务

微服务的考虑



http://martinfowler.com/microservices/

复杂度、
耦合及抽象

微服务架构：
SOA，原则，得失

03 今日头条的架构演进及服务化历程

服务化不简单：
基础设施的构建

主要技术栈

Go

C/C++

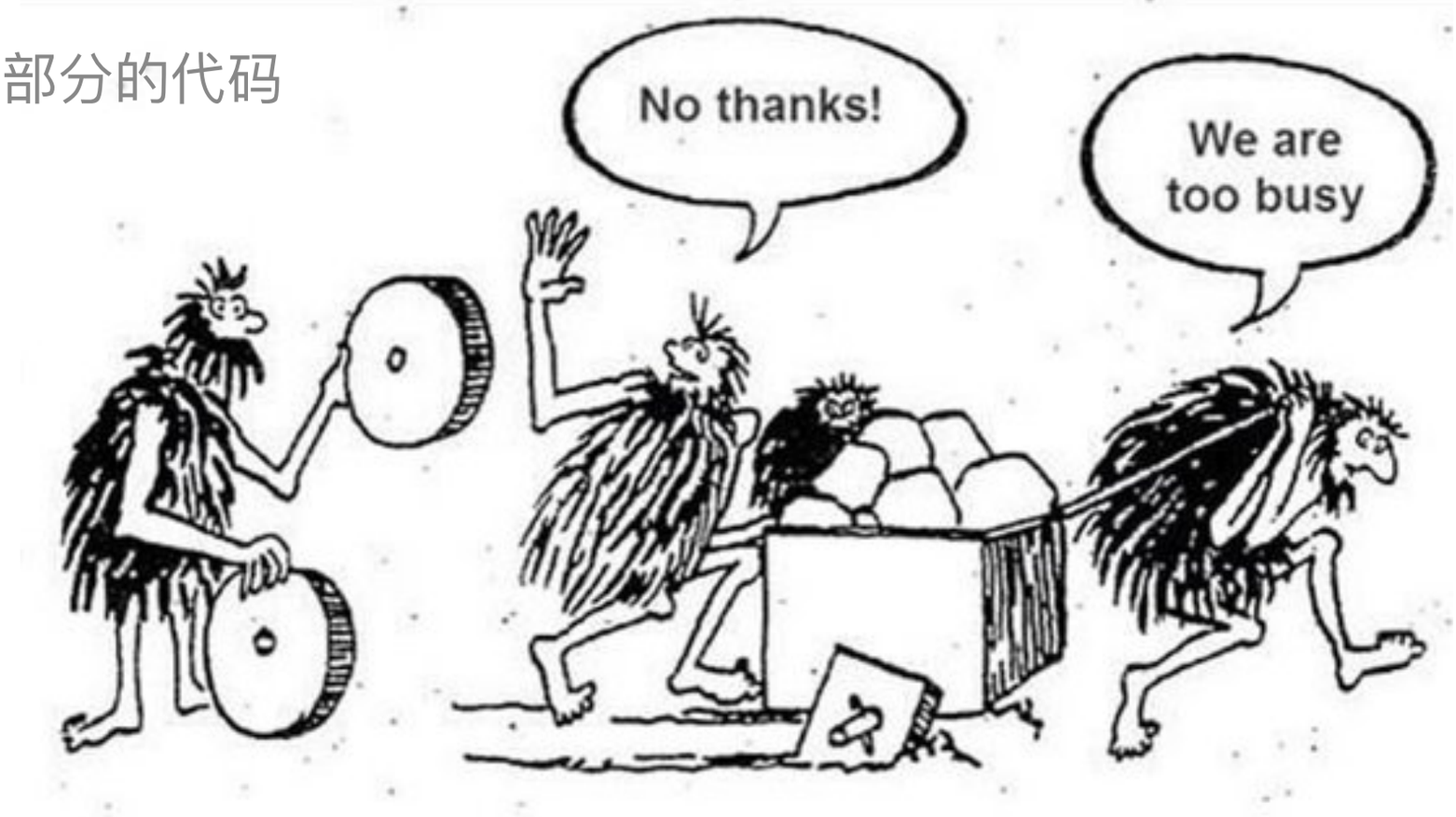
kafka

nsq

背景

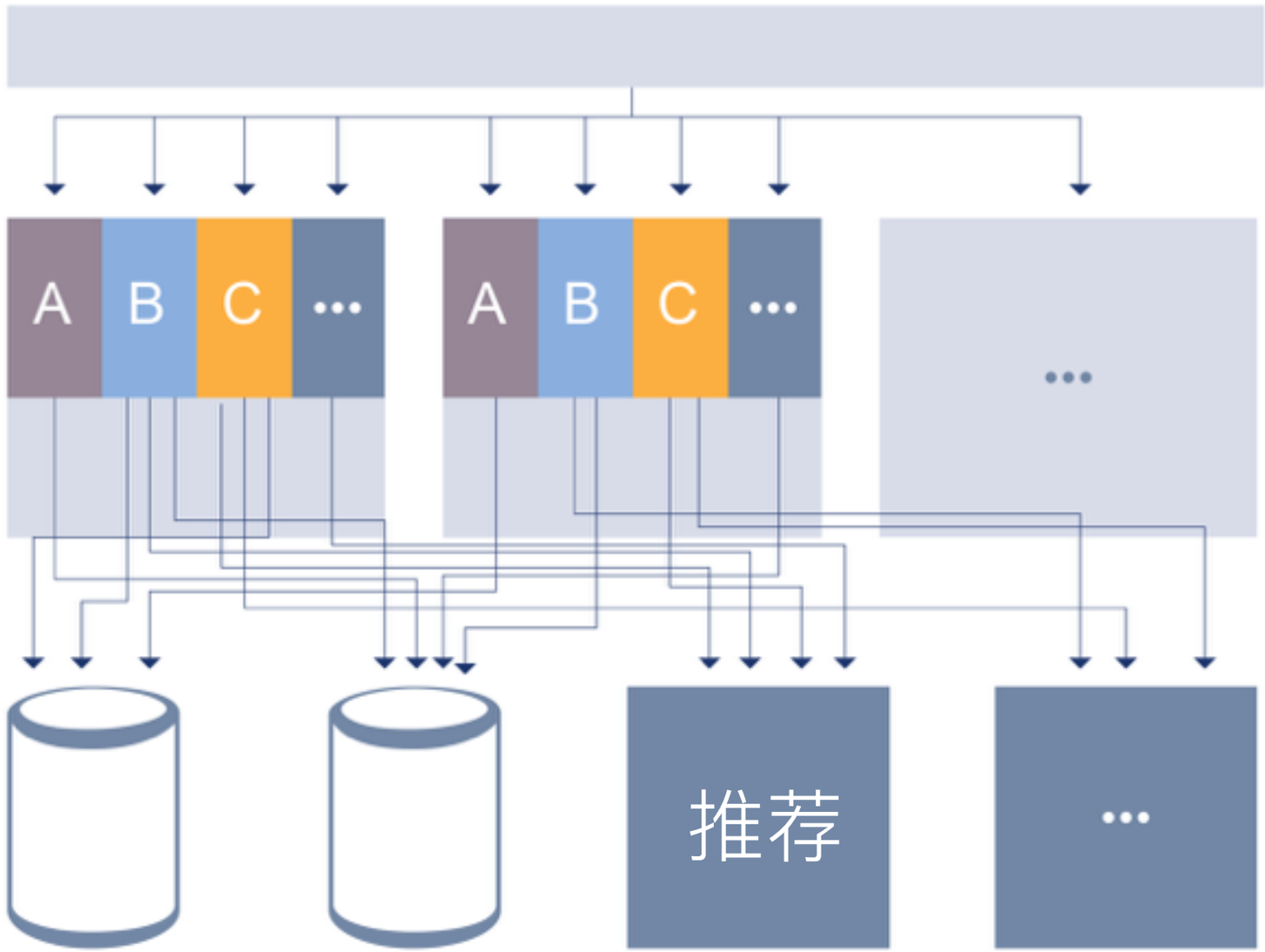
- **初创期业务迭代需求**
  - 代码量：服务端最大的3个git库共 ~**200w+**行代码
  - 大库主干开发模式
  - 早期核心开发者编写大部分的代码
  - 人员快速增长

- **技术债务积累**



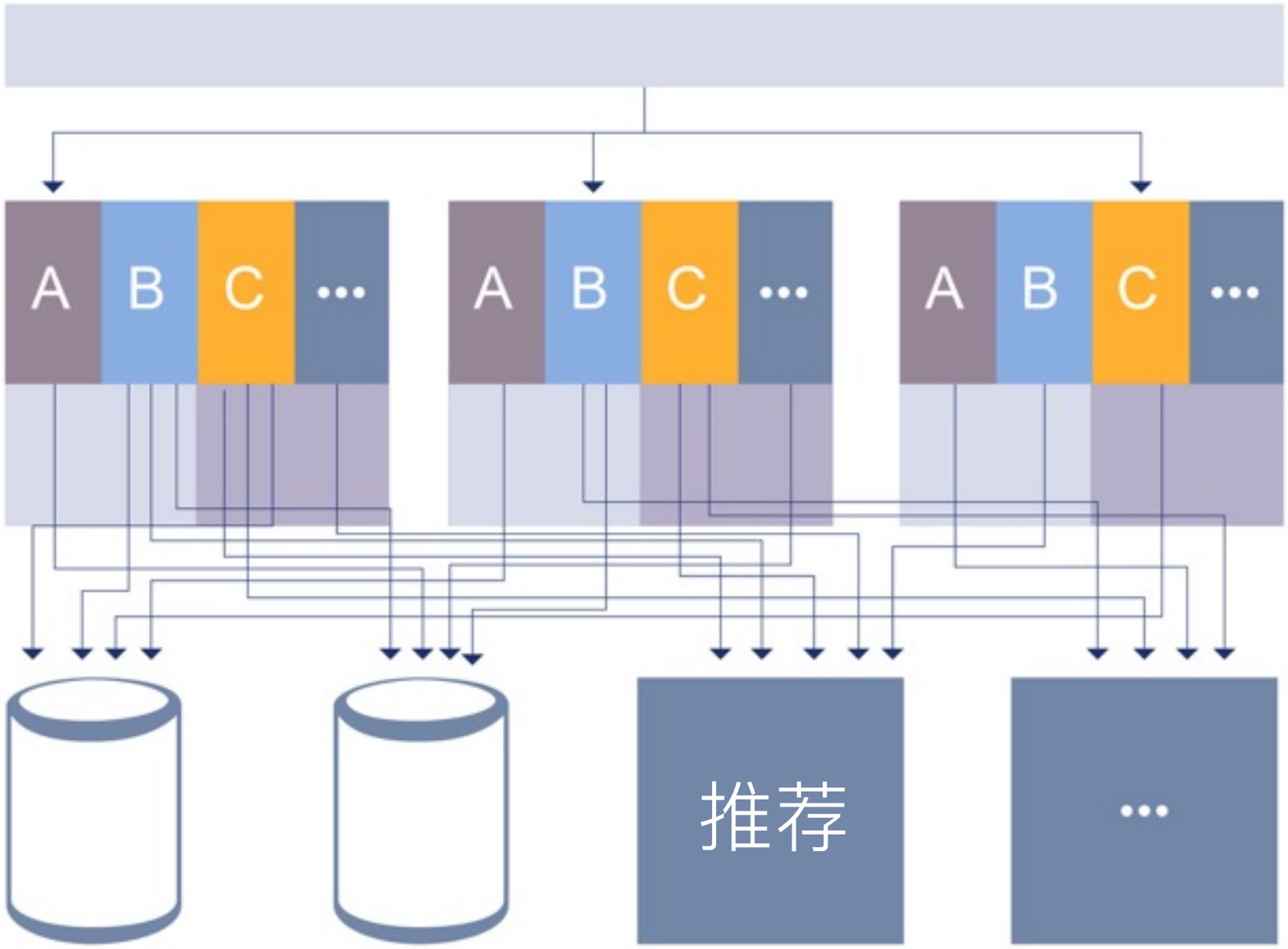
1.0 时代

- **典型三层架构**
- **优势：开发部署快**
- **问题**
  - 部署无隔离
  - 代码没有拆分



1.1 时代

- **服务部署隔离**
- **拆库**：
  - 业务逻辑拆分
- **问题**
  - 人员变多：协同冲突
  - 上线相互影响
  - 异常蔓延
  - 对数据库直接访问
  - 变更困难



演进思路

- **目标：**
  - 松耦合
  - 高可用
  - 易复用、可组合
  - 可靠的快速迭代
  - 开发者友好
  - 运维友好

目标不是服务化？

演进思路

- **思路：**
  - 立规范：部署，交互的收敛统一
  - 打基础：基础库，框架
  - 渐进迭代：
    - 逐步拆分，逐步替换
- 服务化
- 平台化



拆分示例

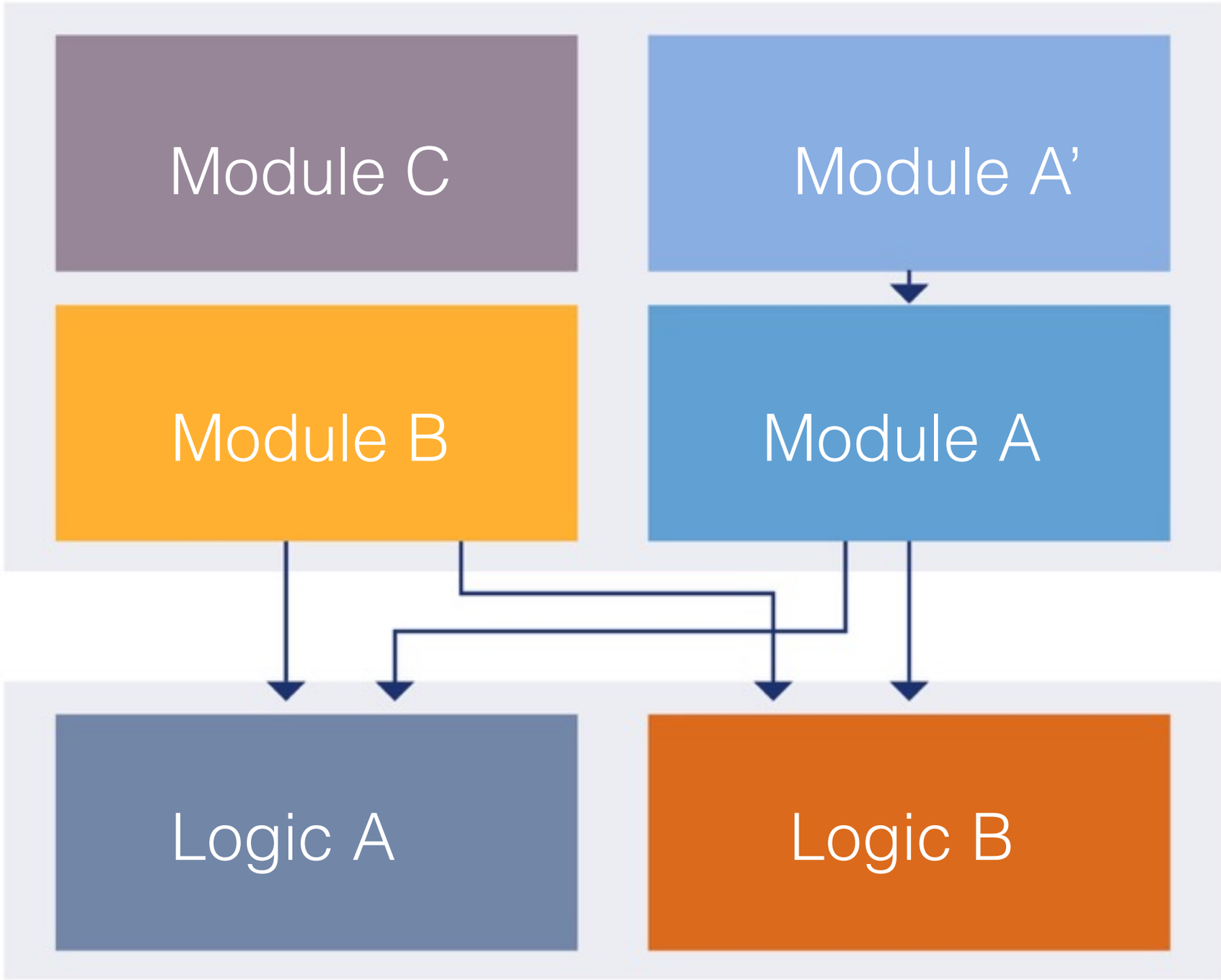
1. **库拆分：**
   - 把相关的业务部分拆到新库
2. **服务化：**
   - 按业务拆分，提供API
3. **保证迭代兼容性**
   - 仍然允许直接依赖老的公共库，但是必须收敛和封装（hook Python import，运行时强制检查）
   - 依赖外部服务完成服务化后从封装的地方改为使用服务

性能问题，拆分后重构成Go实现

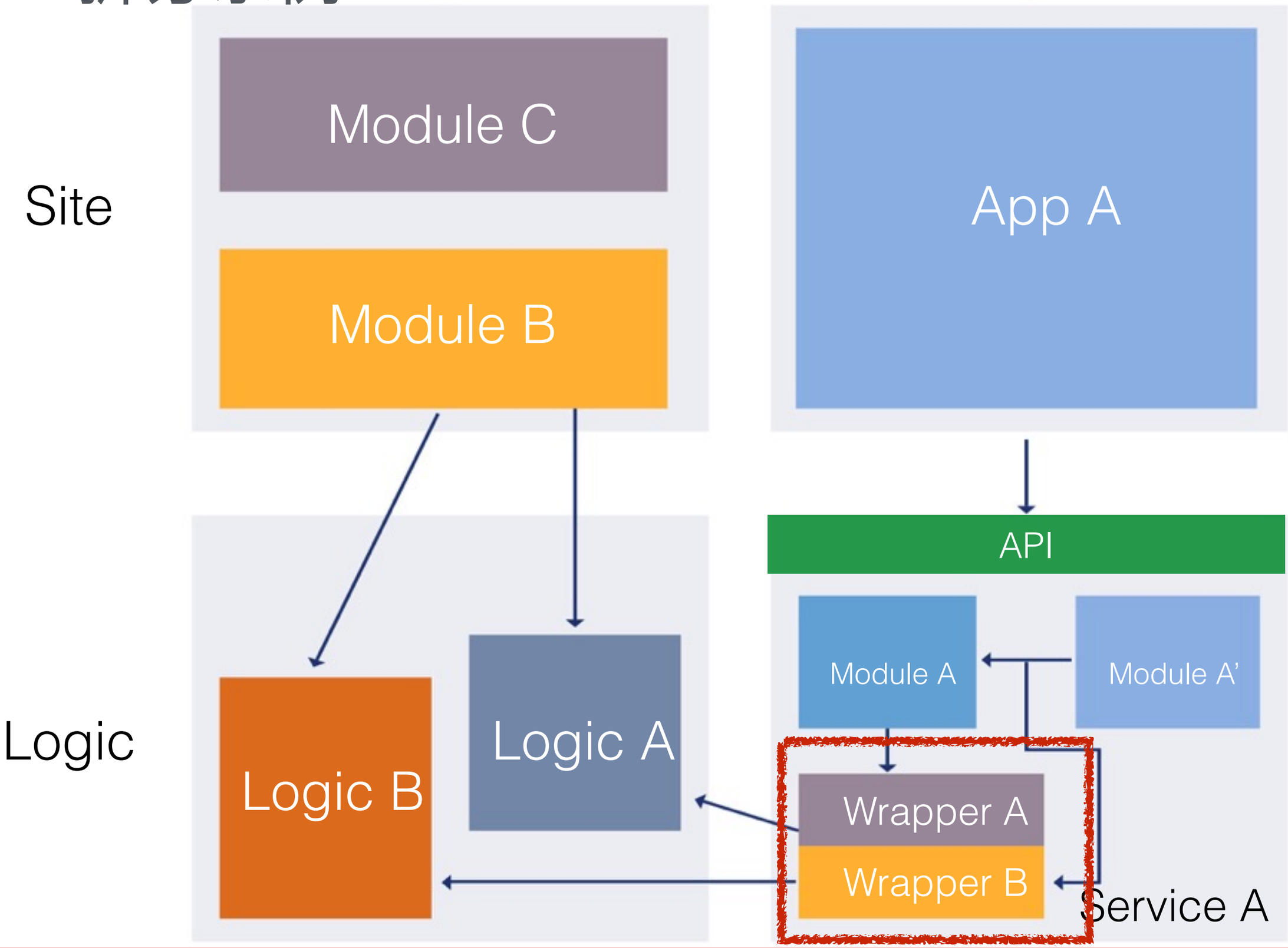
拆分示例

Site

Logic



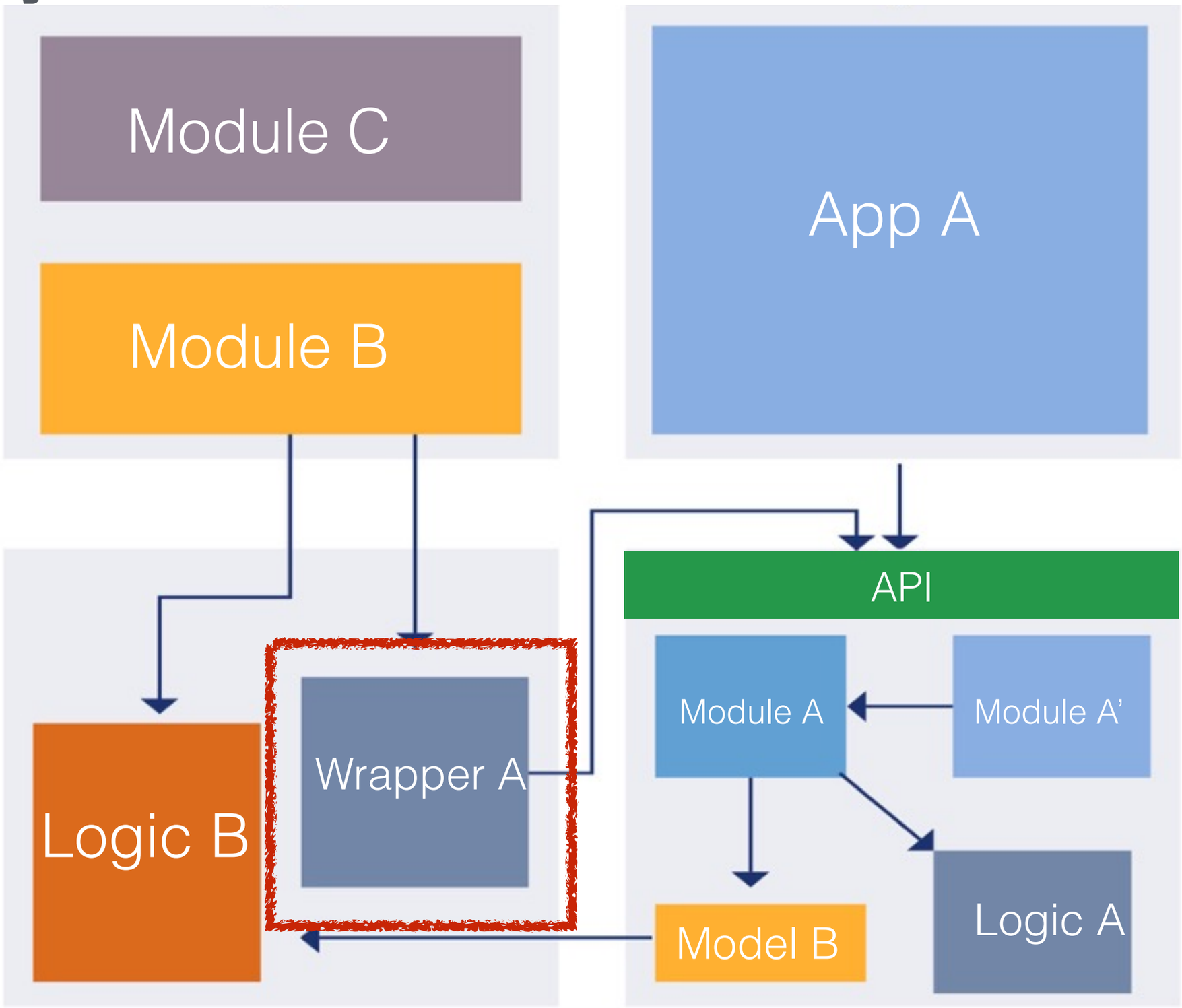
拆分示例


Site
Module C
Module B
App A
API
Logic
Logic B
Logic A
Module A
Module A'
Wrapper A
Wrapper B
Service A

拆分示例

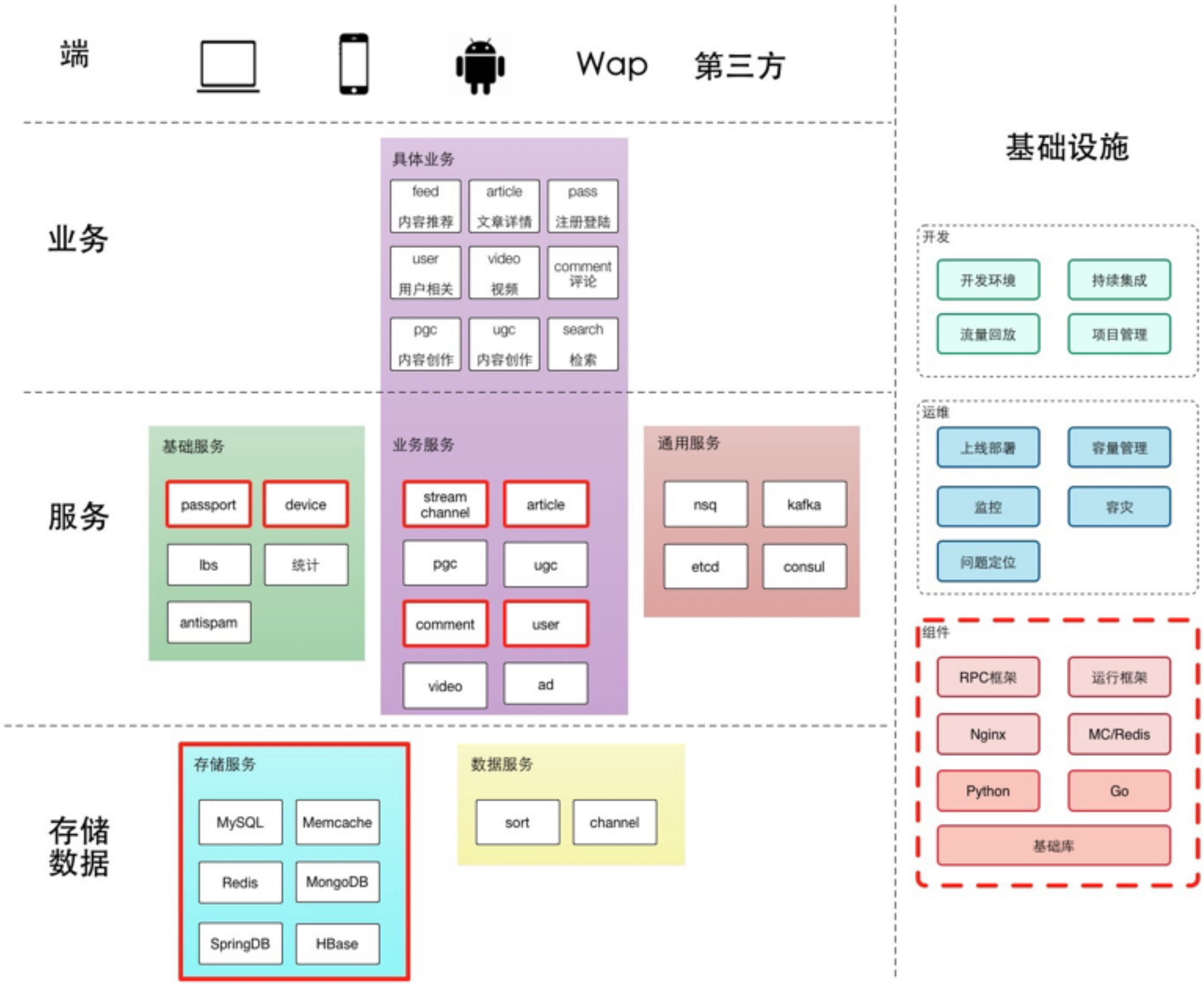
Site

Logic



Service A

拆分后架构


端
Wap
第三方
业务
具体业务
feed
内容推荐
article
文章详情
pass
注册登陆
user
用户相关
video
视频
comment
评论
pgc
内容创作
ugc
内容创作
search
检索
基础服务
passport
device
lbs
统计
antispam
业务服务
stream
channel
article
pgc
ugc
comment
user
video
ad
通用服务
nsq
kafka
etcd
consul
服务
存储
数据
存储服务
MySQL
Memcache
Redis
MongoDB
SpringDB
HBase
数据服务
sort
channel
基础设施
开发
开发环境
持续集成
流量回放
项目管理
运维
上线部署
容量管理
监控
容灾
问题定位
组件
RPC框架
运行框架
Nginx
MC/Redis
Python
Go
基础库

复杂度、
耦合及抽象

微服务架构：
SOA，原则，得失

今日头条的架构
演进及服务化历程

04 服务化不简单：基础设施的构建

服务化历程

打基础

服务平台

服务迁移

2016-04

私有云平台构建

更多服务重构

问题定位服务构建

容灾支持

部分微服务化

部分业务拆库

服务发现

部署平台构建

服务规范化

基础库规范化

构建RPC框架

定规范

2015

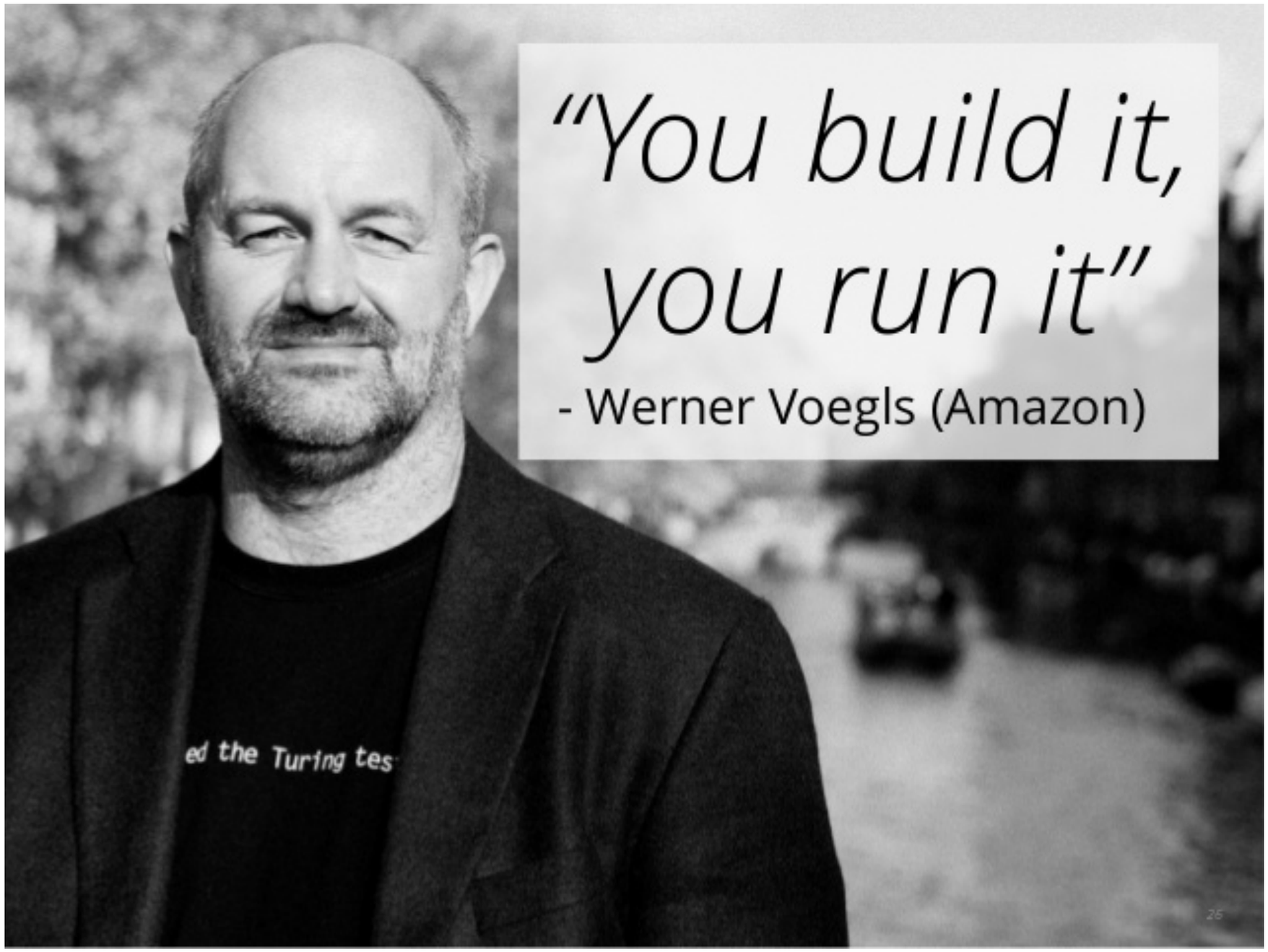
2016

关于服务及服务化

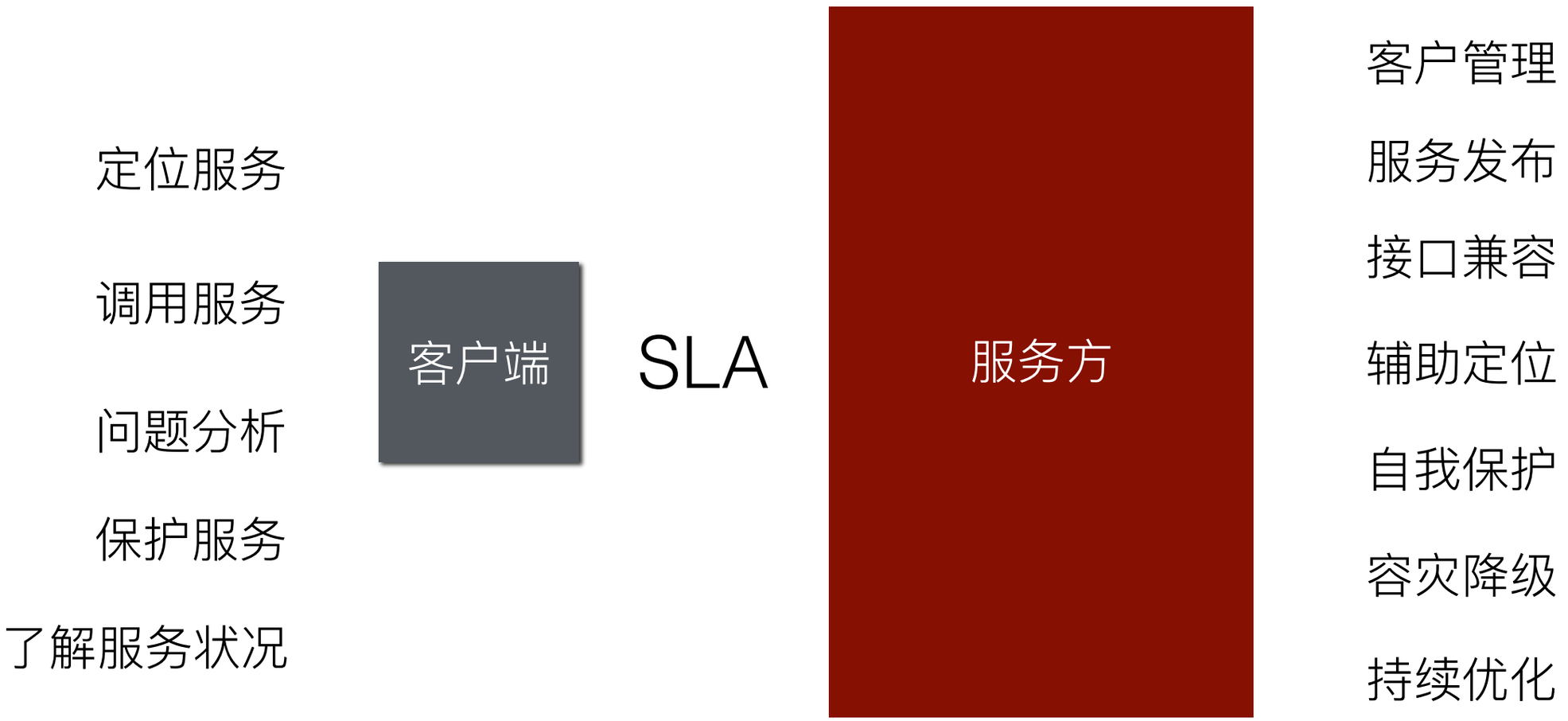
- **什么是服务?**
  - 提供价值的载体
- **什么是服务化?**
  - 明确分工，提供并保证价值，不暴漏细节
- **什么是价值?**
  - 满足基本功能需求
  - 延迟
  - 稳定性
  - 可靠性
  - 开发友好
  - ...

服务化不简单

- **变为多个系统**
  - 微服务的寻址问题
  - 请求跨越多子服务，定位问题麻烦
  - 错误是常态
  - 性能损失
- **调用层次化**
  - 错误的放大，雪崩问题
  - 扇出:并行化
  - 稳定性的保证
- **快速部署迭代的需求**
  - 依赖管理
  - 各种子系统，混部的隔离问题 (PaaS)
  - 容器和微服务


"You build it, you run it"
- Werner Voegls (Amazon)
ed the Turing tes
25

服务化不简单


定位服务
调用服务
问题分析
保护服务
了解服务状况
客户端
SLA
服务方
客户管理
服务发布
接口兼容
辅助定位
自我保护
容灾降级
持续优化

服务化不简单

定位服务

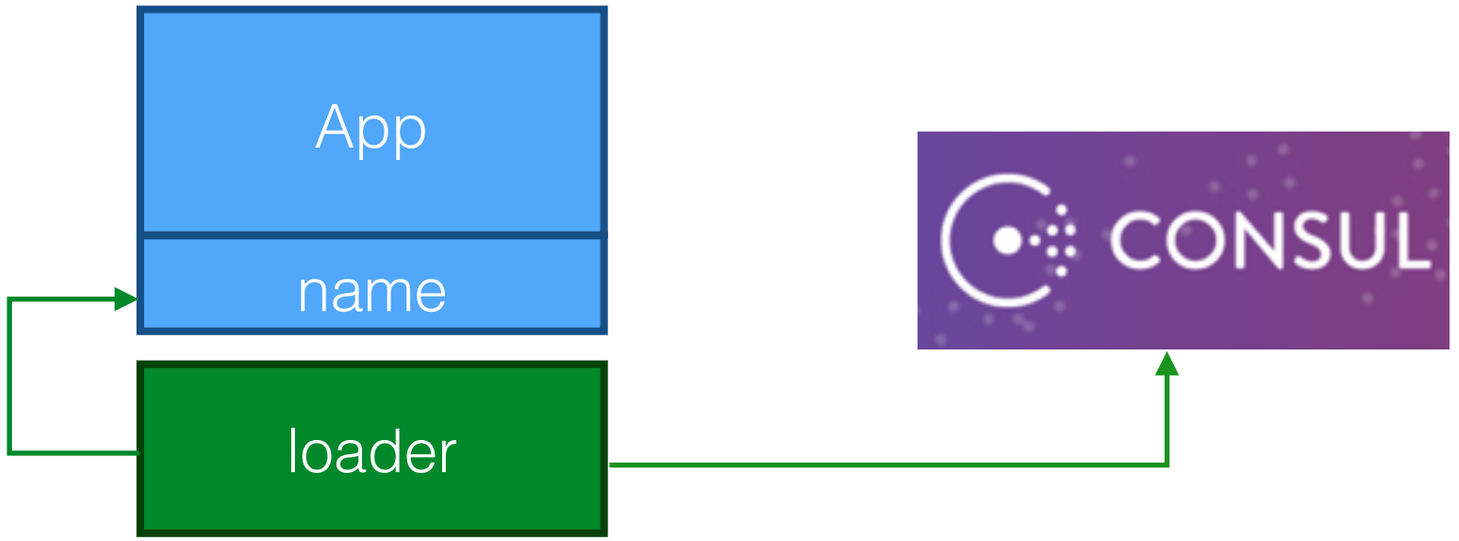
调用服务

问题分析

保护服务

了解服务状况

统一规范：服务必须有唯一名称，统一启动方式，启动时自注册



服务化不简单

定位服务

调用服务

问题分析

保护服务

了解服务状况

服务接口IDL

Thrift based RPC

logid

caller

Code generation

service_rpc(python)

Kite (go)

Kitc(go)

服务化不简单

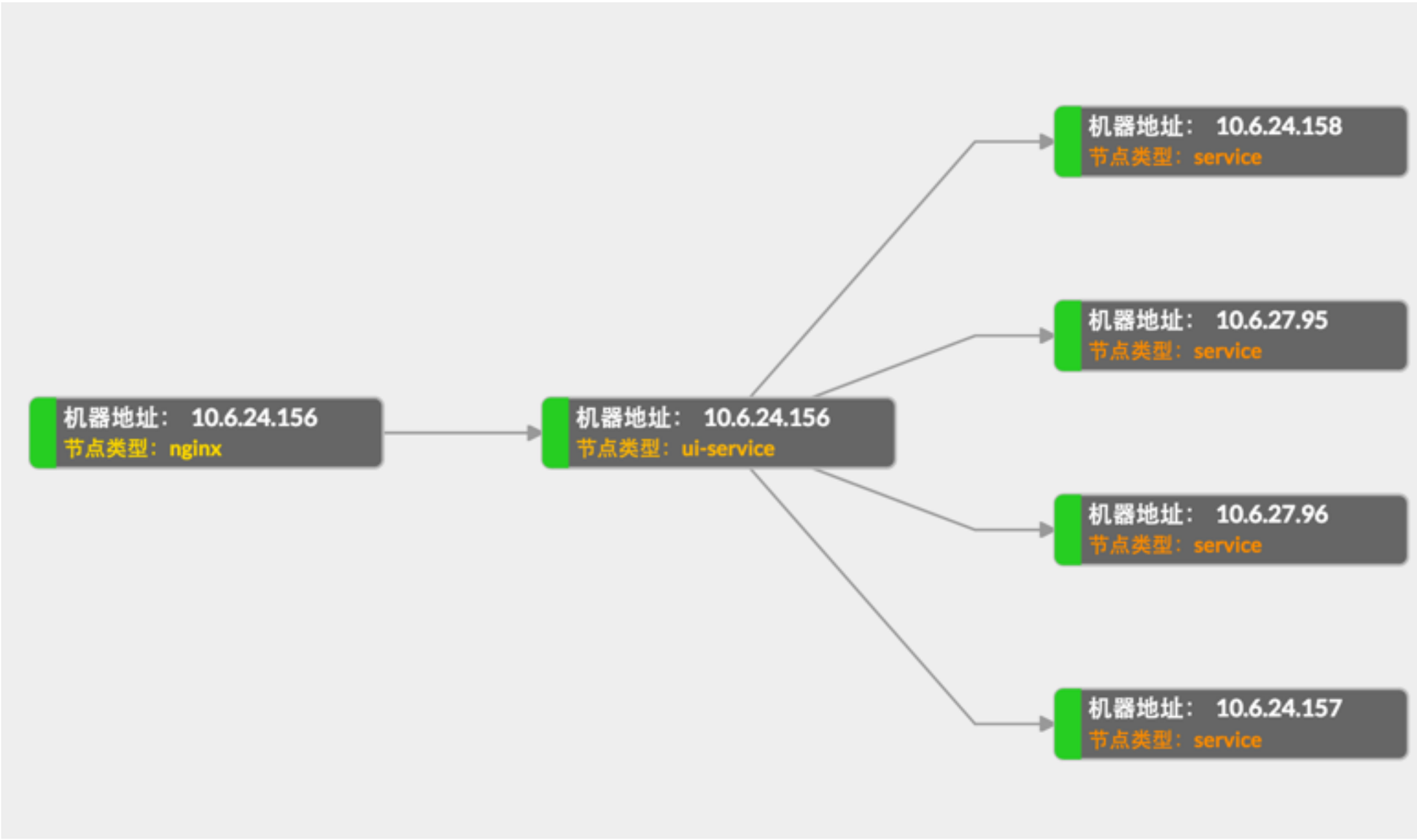
定位服务

调用服务

问题分析

保护服务

了解服务状况



服务化不简单

当一个安分守己的好公民

定位服务

多级调用的放大效应
后端服务能力不足时导致无法恢复

调用服务

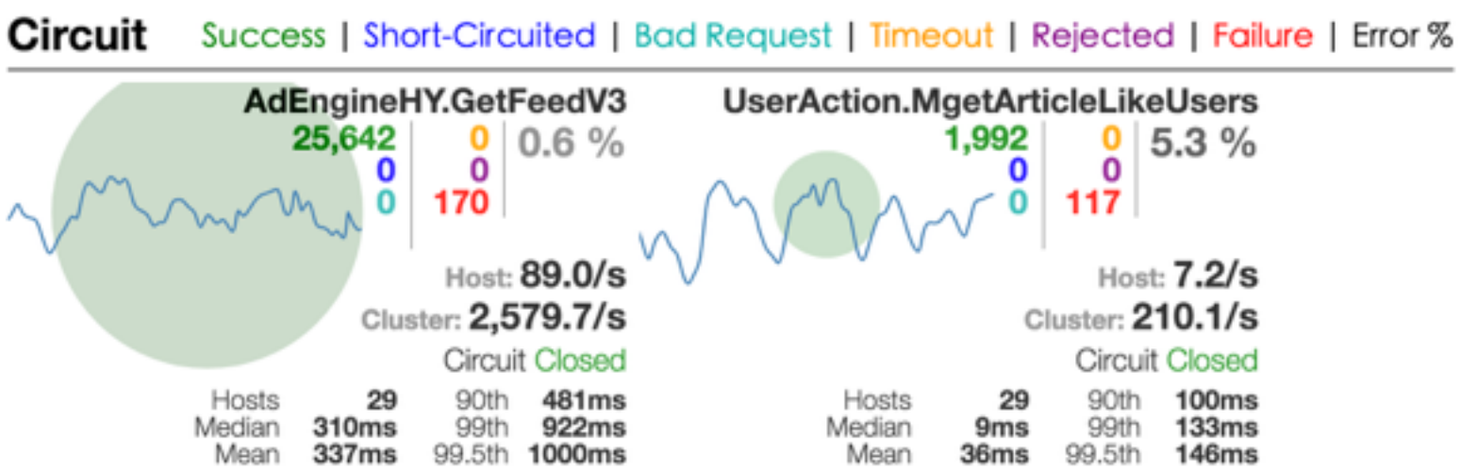
使用Netflix Hystrix提升容错能力。防止雪崩效应

问题分析

断路器、回退机制：应对远程系统的延迟，故障等

保护服务

了解服务状况



服务化不简单

客户管理

服务注册

接口兼容

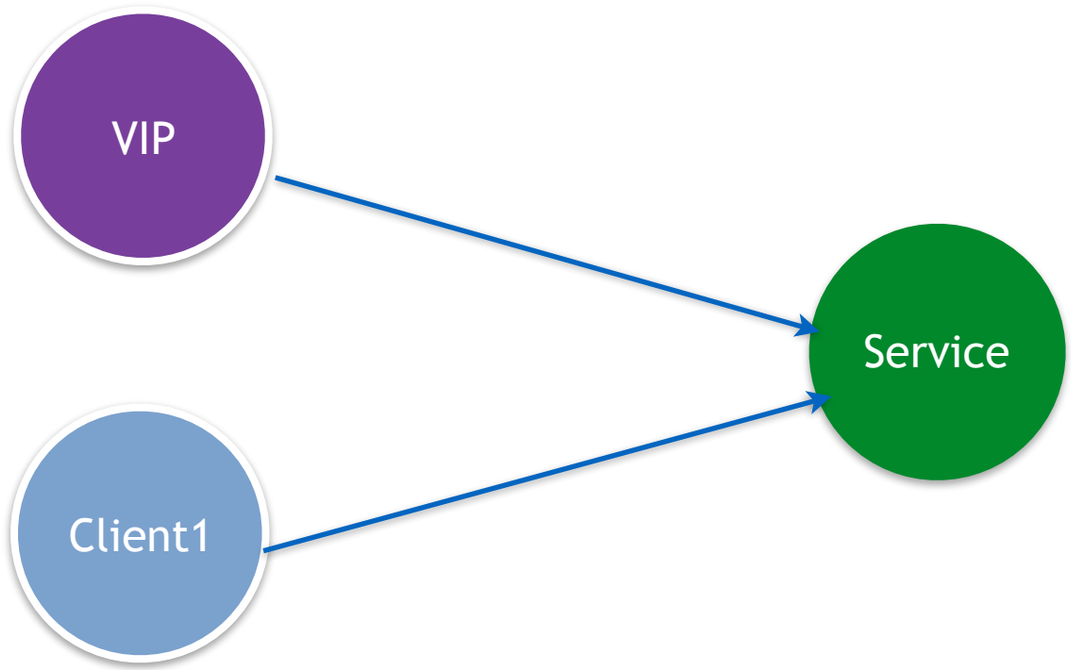
辅助定位

自我保护

容灾降级

持续优化

故障及灾难时区别对待客户(WIP)



服务化不简单

客户管理

服务注册

接口兼容

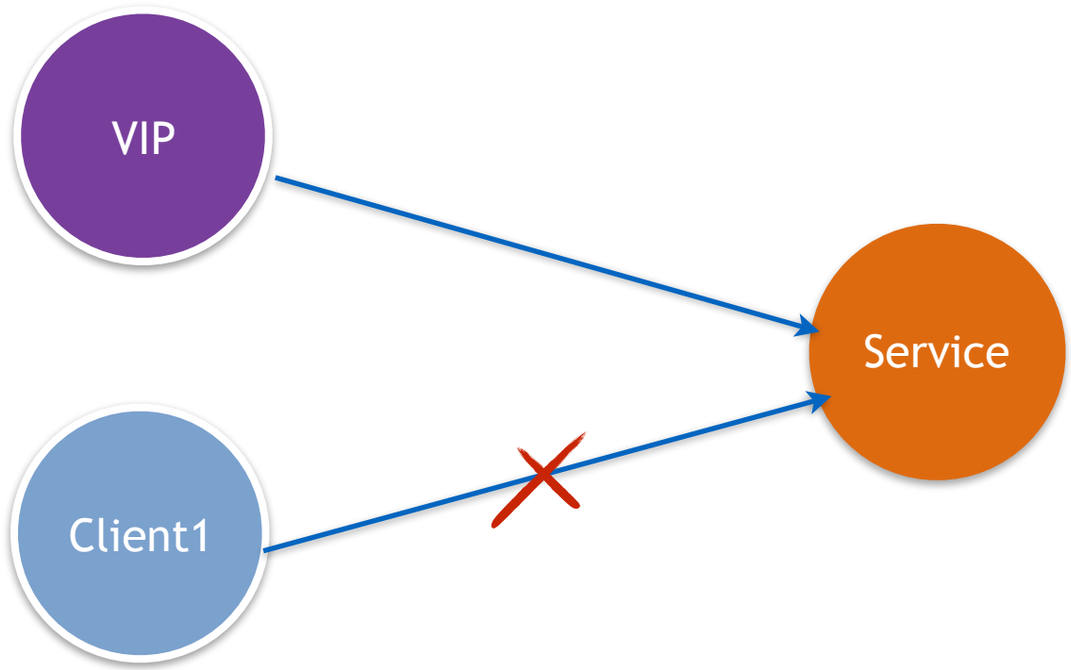
辅助定位

自我保护

容灾降级

持续优化

故障及灾难时区别对待客户(WIP)



服务化不简单

客户管理

服务注册

接口兼容

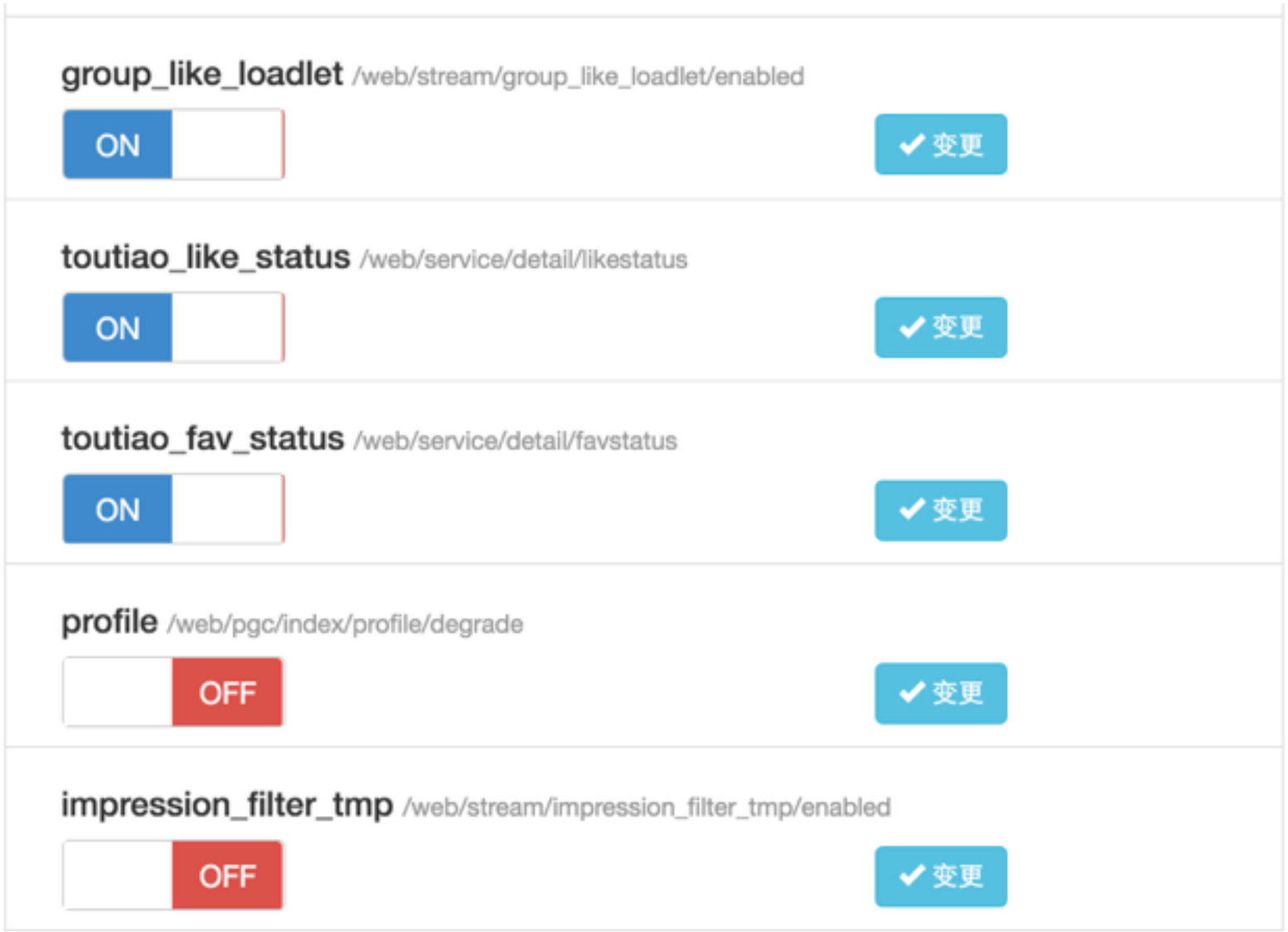
辅助定位

自我保护

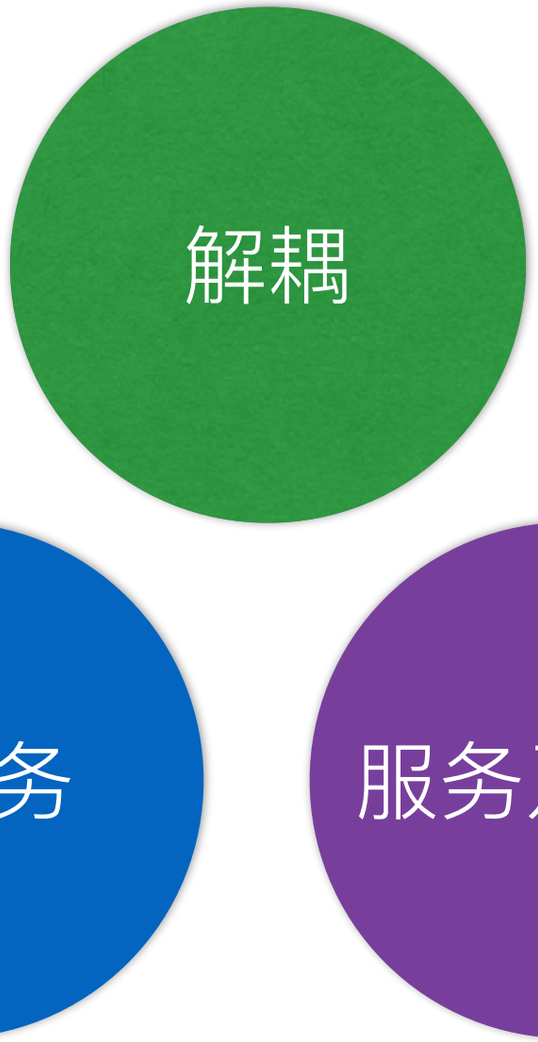
容灾降级

持续优化

自我保护：根据服务压力拒绝过载请求



总结



管理复杂度的利器

一种解耦的模式

构建好服务要做的更多

Thanks