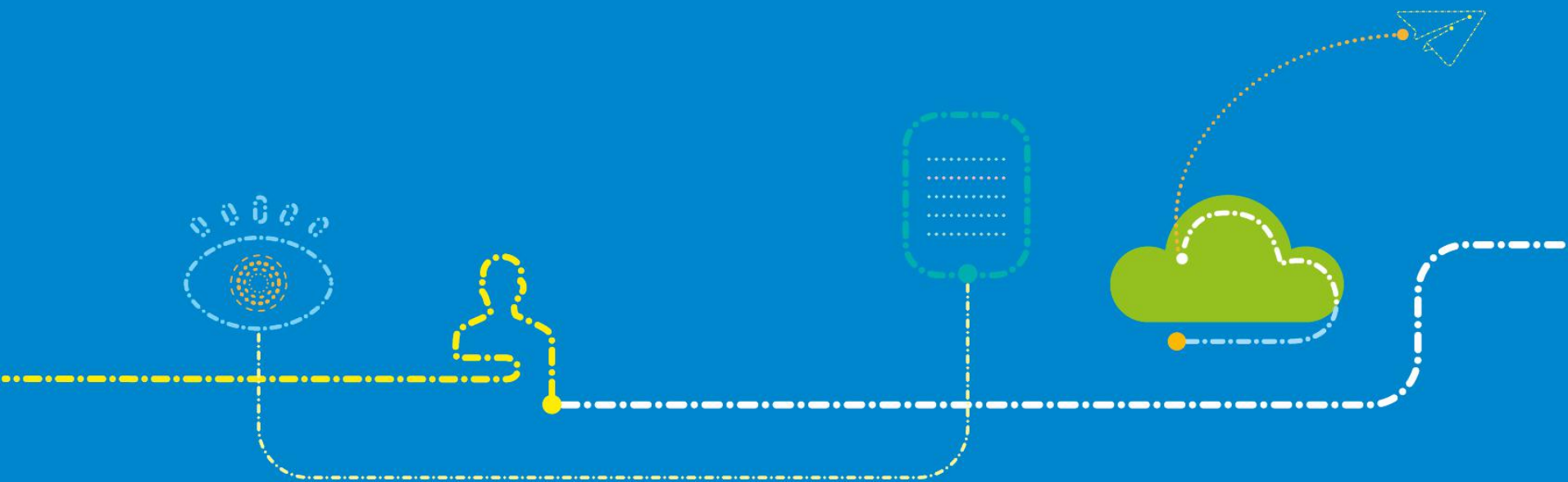


基于5G网络管理系统的服务网格实践

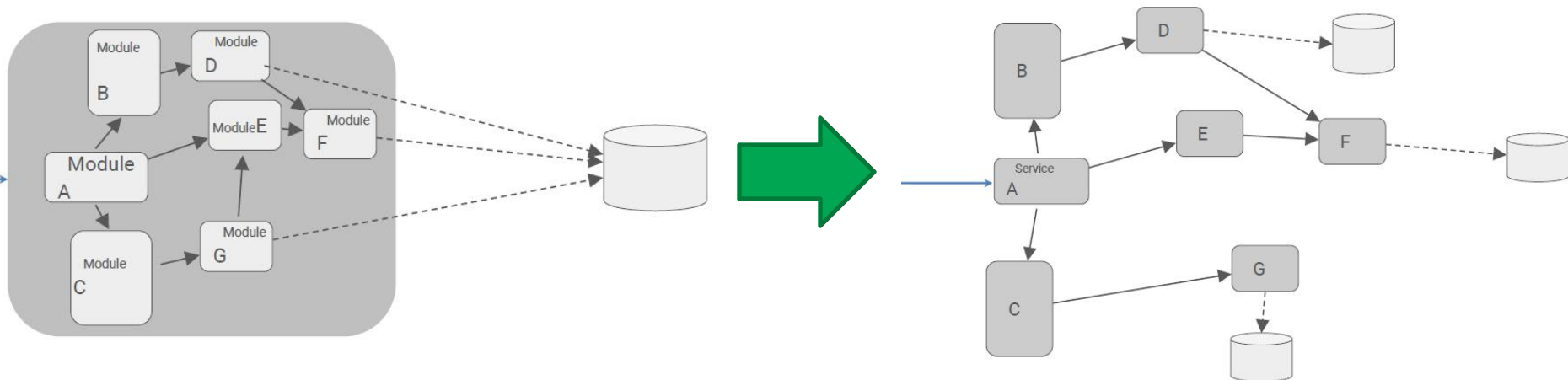
ZTE中兴

服务网格技术介绍、实践与发展趋势

ZTE 中心研究院 赵化冰



Monolith向Microservice 的演变



作为一种架构模式，微服务将复杂系统切分为数十乃至上百个小服务，每个服务负责实现一个独立的业务逻辑。

运行态：模块间方法调用变成了进程间的远程调用。

Microservice 架构带来的挑战

微服务架构引入了**分布式系统**带来的一系列复杂问题，包括：

问题：

- 客户端如何找到服务提供者
- 如何保证远程调用的可靠性
- 如何保证服务调用的安全性
- 如何保证系统的可见性
- 如何进行端到端流程调试
- 如何保证系统的健壮性
- 如何对系统进行容错性测试

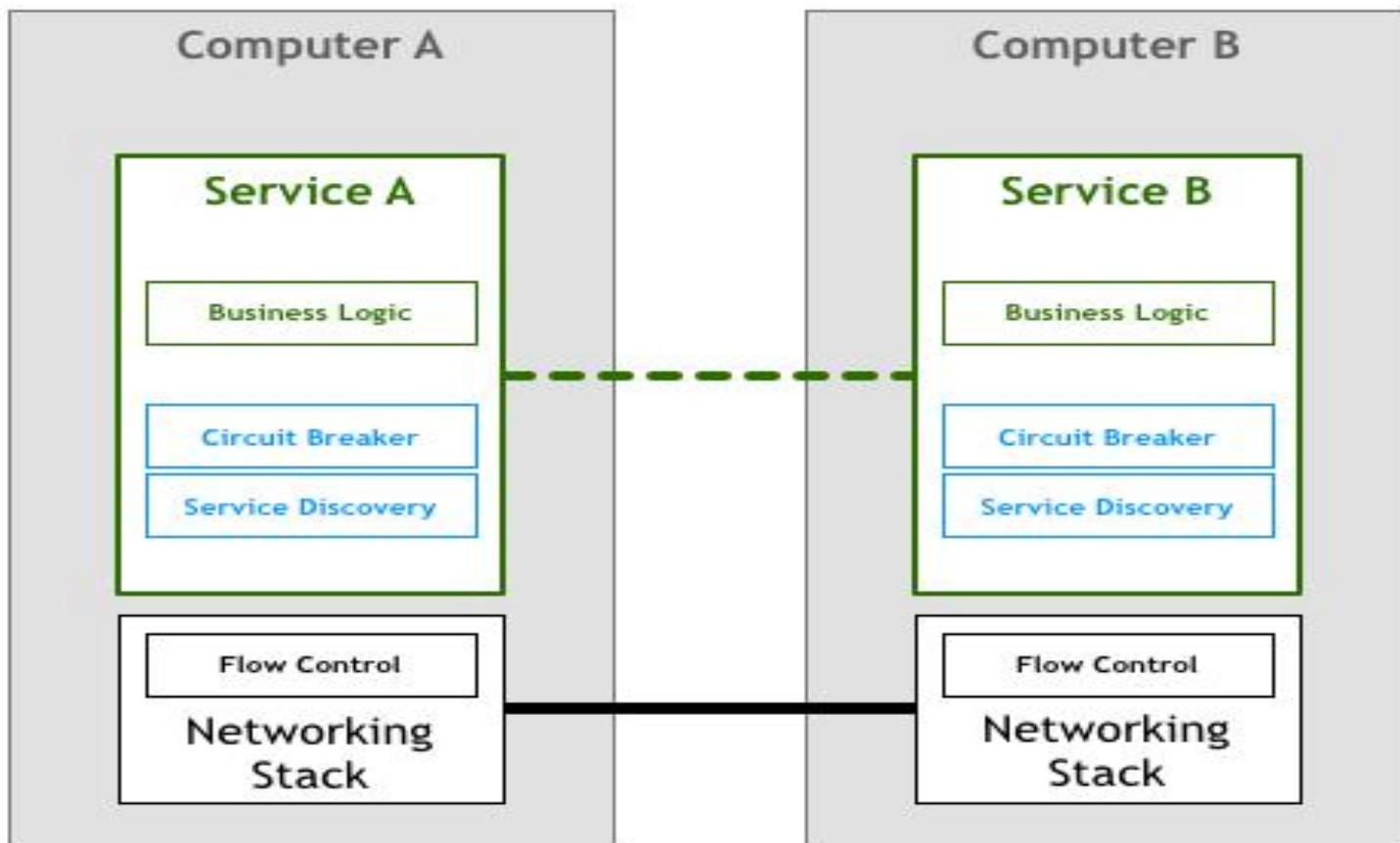
方案：

- 服务注册和发现
- 超时，重试，负载均衡
- 服务认证和鉴权
- 性能指标收集及分析/日志收集
- 分布式调用追踪
- 熔断，限流，故障恢复
- 故障模拟/Chaos测试

微服务架构在带来收益的同时提高了系统的复杂度，并加大了应用运维的难度。

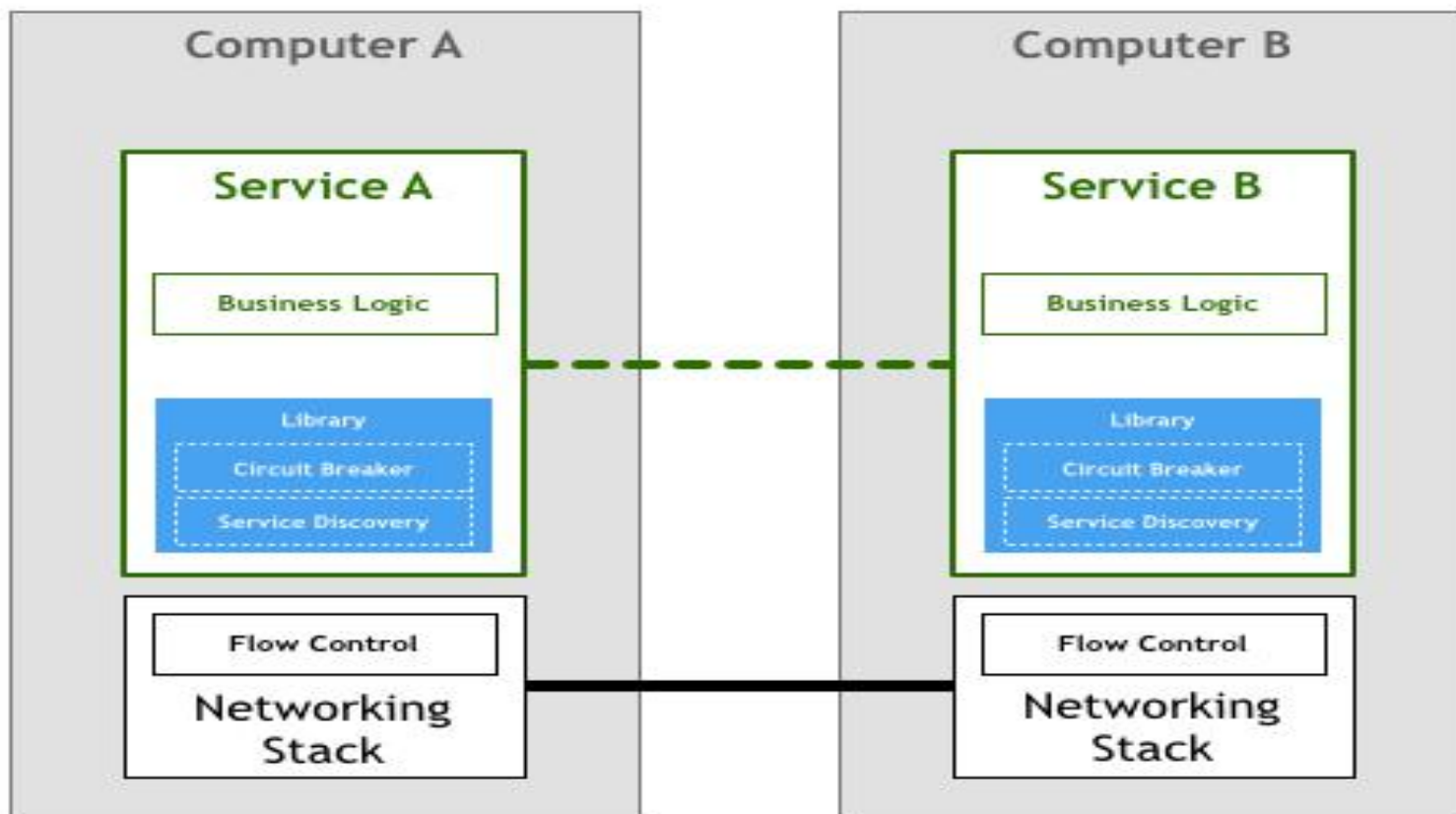
解决方案：原始时代

由不同微服务的开发人员各自处理服务之间的通讯，包括服务发现，重试，超时，容错，安全等逻辑。



解决方案：类库模式

针对不同的语言提取出类库，来解决微服务的通讯的一些共性问题。



类库带来的问题

微服务带来的一个巨大优势，就是开发团队可以根据业务特点和人员技能**灵活采用不同的技术栈**来实现一个微服务以及可以**独立部署、升级微服务**。但是，当我们将服务通讯和治理相关代码封装到类库和框架时，会面临下述问题：

主流编程语言

- Java
 - Scala
 - Groovy
 - Kotlin
- C
- C++
- C#
- Python
- PHP
- Ruby

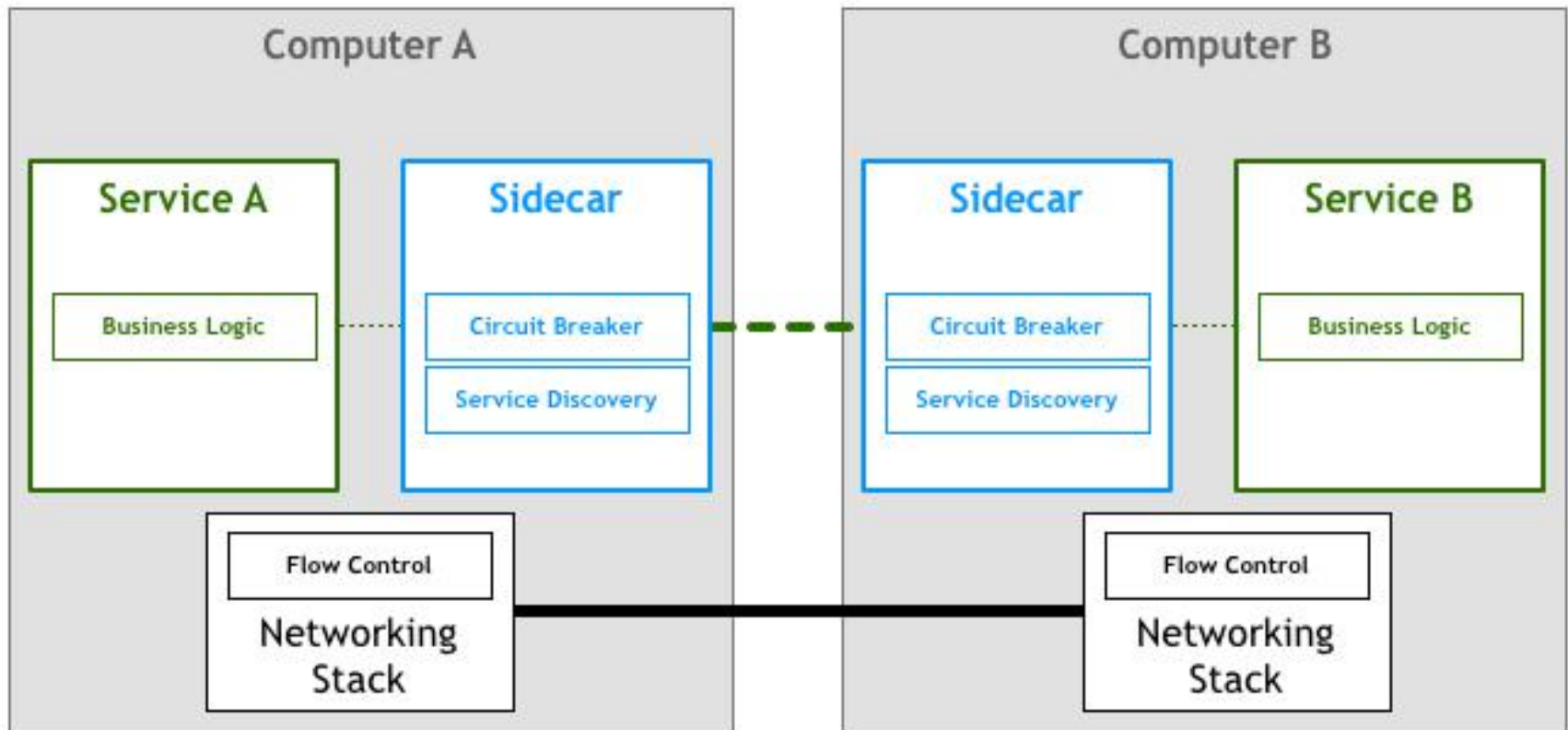
新兴编程语言

- Golang
- Node.js
- Rust
- R
- Lua/OpenResty

- 需要针对不同的程序语言开发不同的代码库，反过来会影响微服务应用开发语言和框架的选择，限制技术选择的灵活性。
- 随着时间的变化，代码库会存在不同的版本，不同版本代码库的兼容性和大量运行环境中微服务的升级将成为一个难题，导致微服务不能独立部署和升级。
- 类库的机制比较复杂，对开发人员而言有较高的学习成本，导致其不能将其全部精力聚焦于业务逻辑。

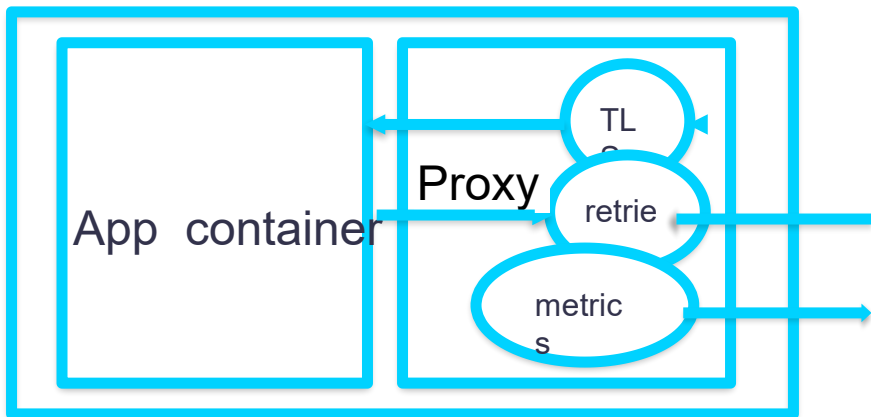
解决方案：边车模式

应用程序在进行网络通信时并不需要关注TCP/IP协议栈的实现，微服务需要关注服务间的通信细节吗？



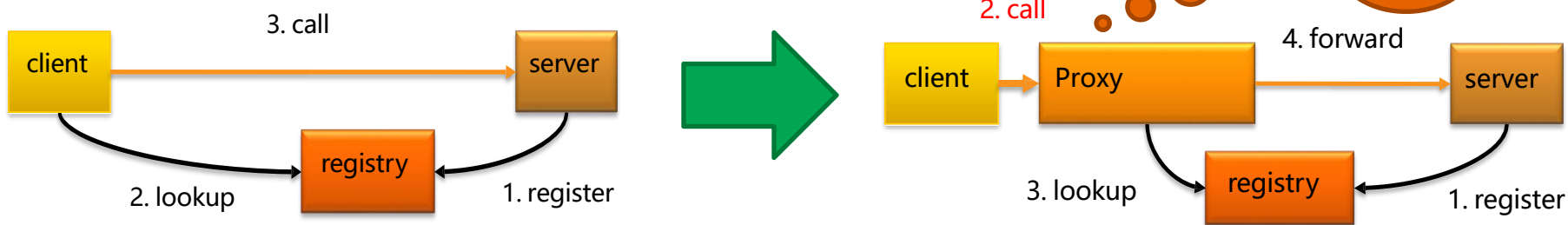
边车模式：将为微服务提供通信服务的这部分逻辑从应用程序进程中抽取出来，作为一个单独的进程进行部署，并将其作为服务间的通信代理。

微服务通讯代理的这种部署方式被形象地称为“**Sidecar**”，即三轮摩托车的“跨斗”



Sidecar模式可用于插入其他横切面逻辑：
安全、策略、Telemetry、分布式调用跟踪、缓存等等

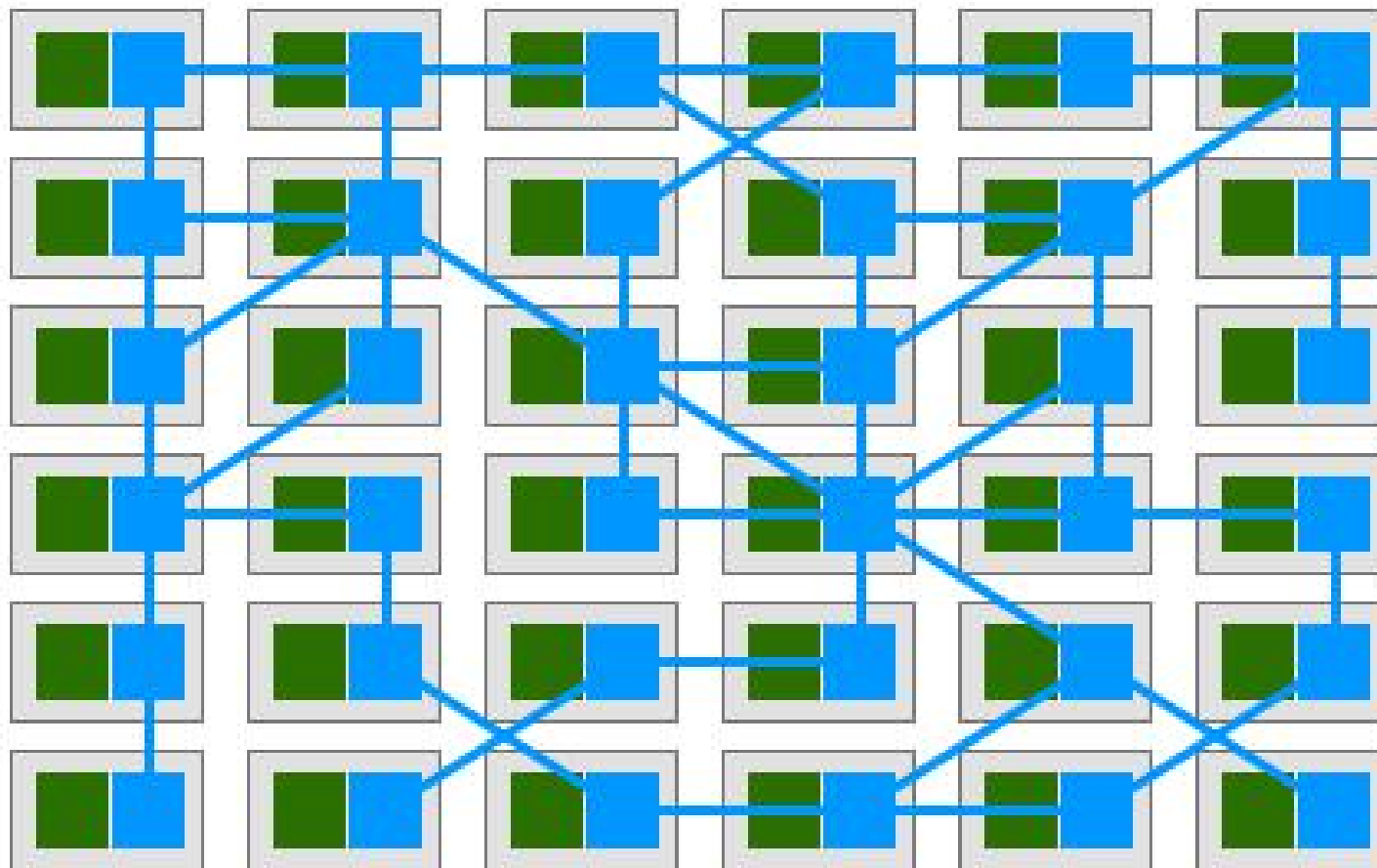
服务间的通信层处理被迁移到以“跨斗”方式部署



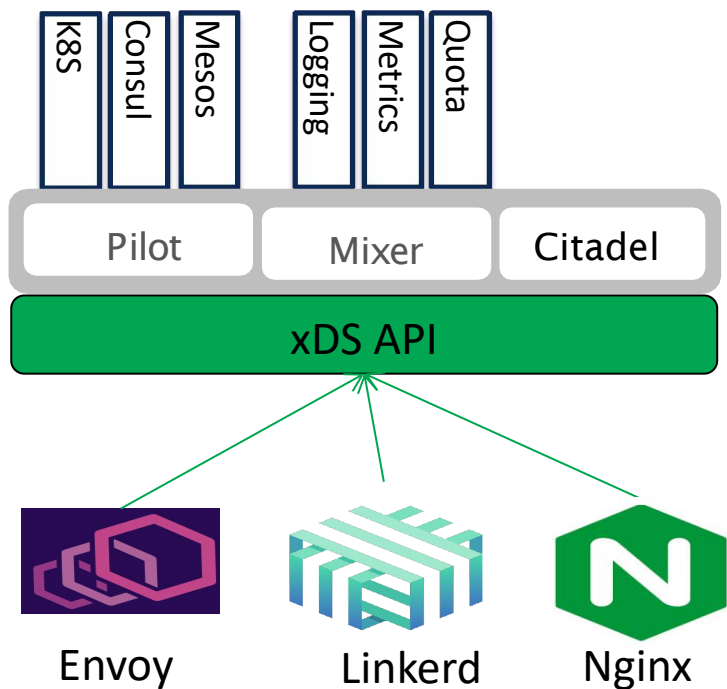
Proxy对应用**无侵入**，应用对Proxy**不感知**

服务开发者只需要关注产品的商用价值重点：业务逻辑

在集群中部署的多个Proxy形成支撑服务间通信的一个“网格”

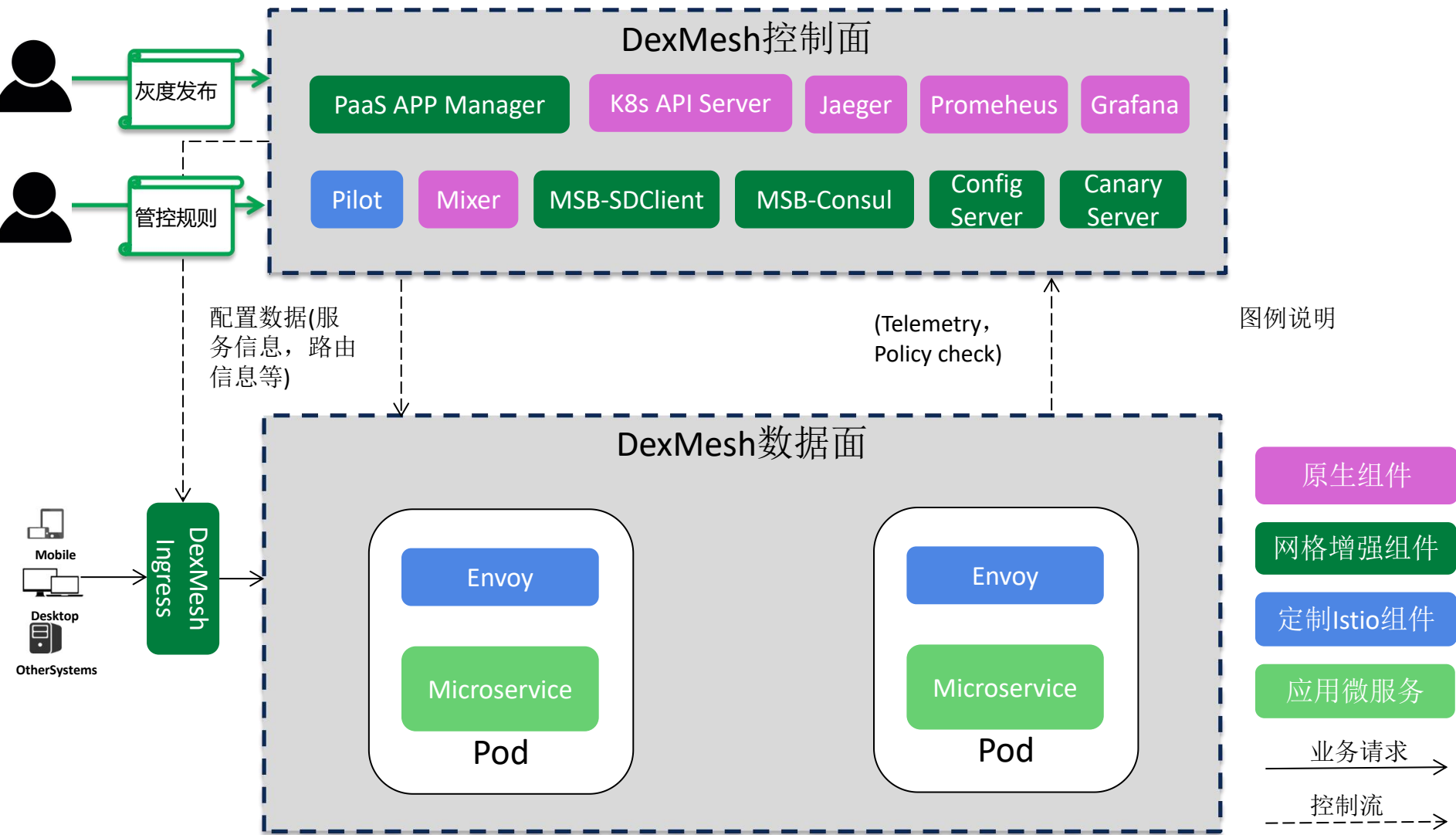


Istio简介



- Istio提出了**控制面**的概念，以对网格中的代理进行统一管理 and 配置
 - **Pilot**: 服务发现和流量管理
 - **Mixer**: 遥测报告和网格策略（如黑白名单和限流）
 - **Citadel**: 双向SSI认证和基于角色的访问控制
- Istio架构具有高度的可扩展性和灵活性
 - 可以接入不同的Service Registry: Kubernetes, Consul, Mesos...
 - 可以接入不同的微服务后端基础设施: Prometheus, Heapster, AWS CloudWatch...
 - 通过标准的数据面接口解耦控制面和数据面: Envoy, Linkerd, Nginx都可以作为数据面接入门格中

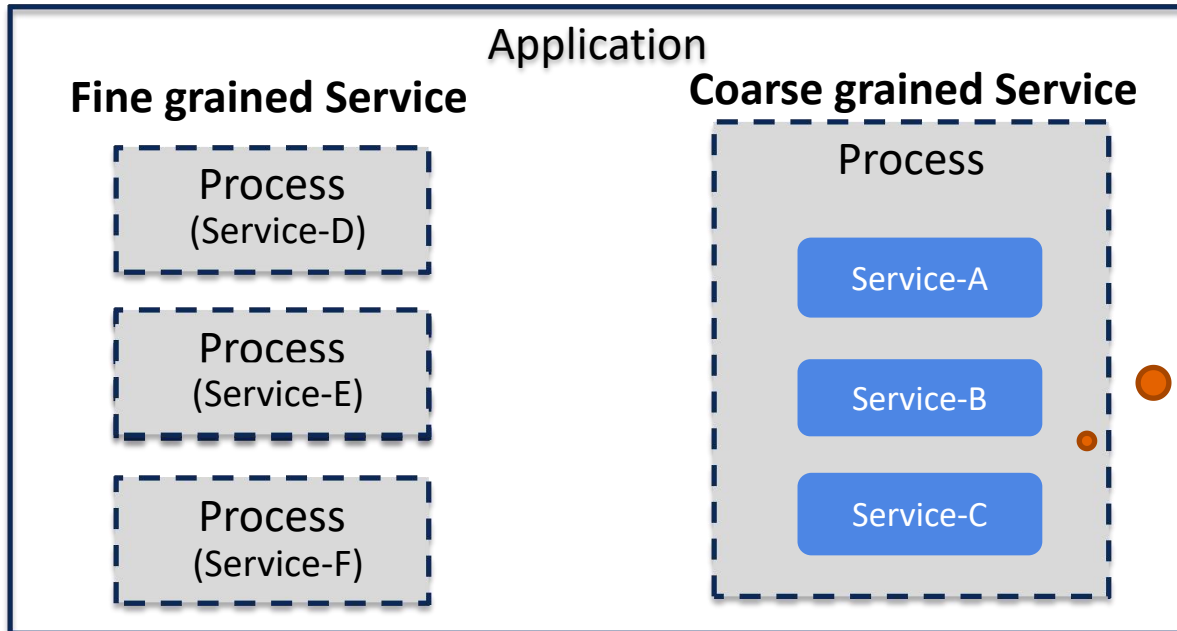
总体架构-高层视图



产品化增强-支持粗粒度的SOA类应用

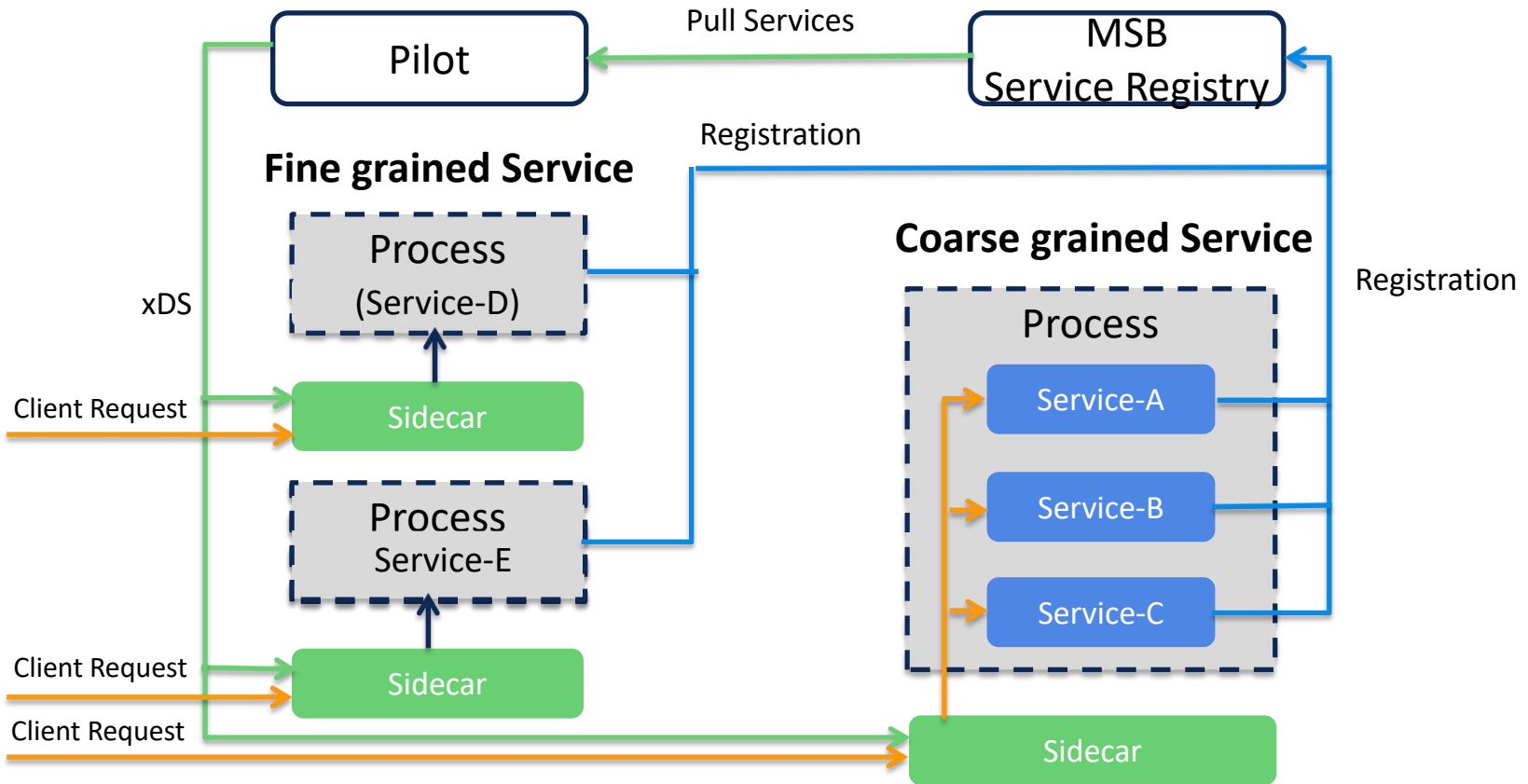
Kubernetes and Istio assumes that one process/container is a service. While this assumption is fine with “pure” Microservices architecture, it does have problems supporting “Coarse Grained” Service.

An example application: Combination of “fine Grained” and “Coarse Grained” Services



How to map multiple logic services inside one process to Istio service?

产品化增强-支持粗粒度的SOA类应用



产品化增强-Ingress API Gateway



K8S Ingress

Load balancing
SSL termination
Virtual hosting

提供七层网关能力，
但和服务网格是割裂的



Istio Gateway

Load balancing
SSL termination
Virtual hosting
Advanced traffic routing
Fault injection
Other benefits brought by Istio

提供七层网关和网格能力，
但缺少API管理能力

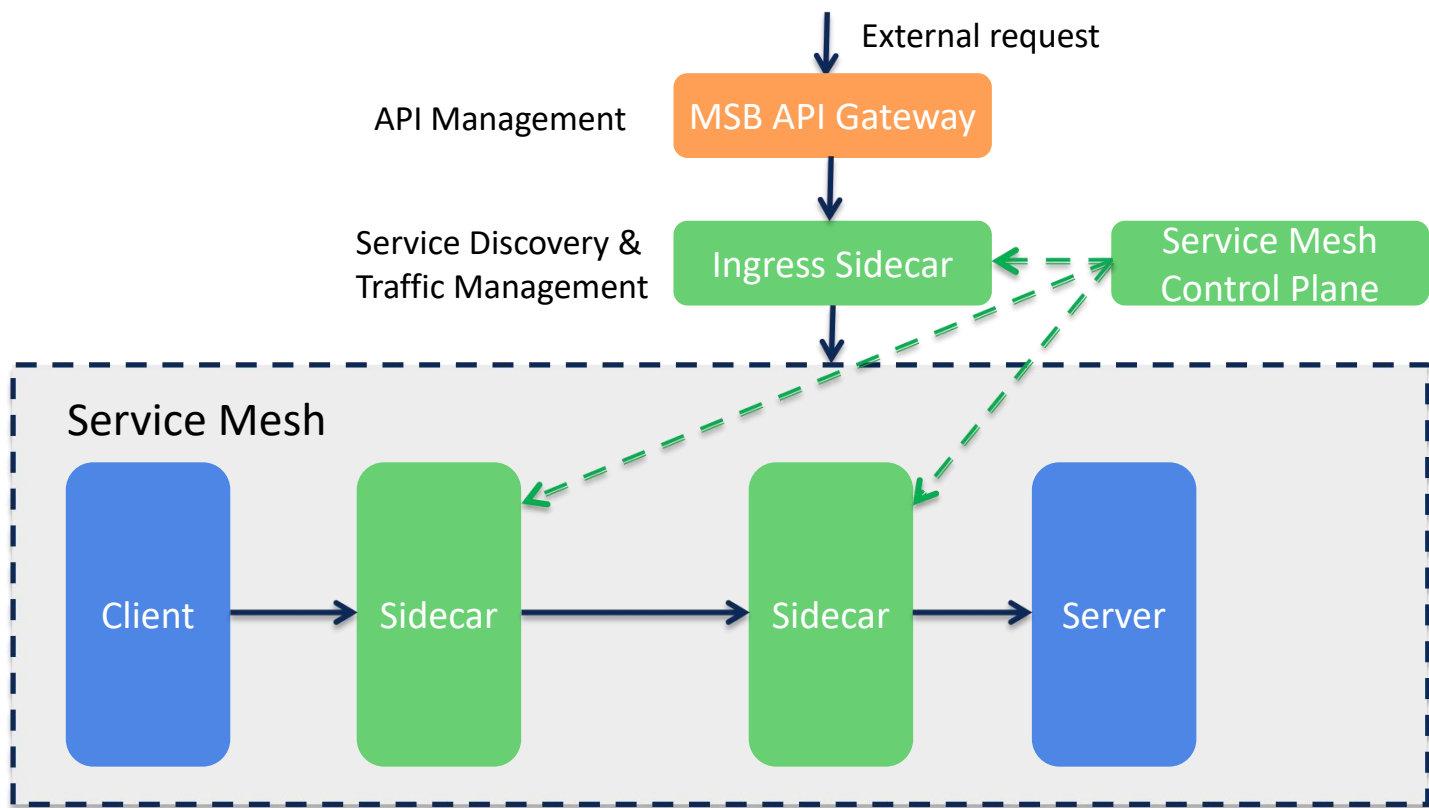


API Gateway

Load balancing
SSL termination
Virtual hosting
Traffic routing
API lifecycle management
API access monitoring
API access authorization
API key management
API Billing and Rate limiting
Other business logic ...

提供API管理能力，
缺少服务网格能力

在DexMesh场景下Mesh和API Gateway的分工与协同



API Gateway: 应用网关逻辑

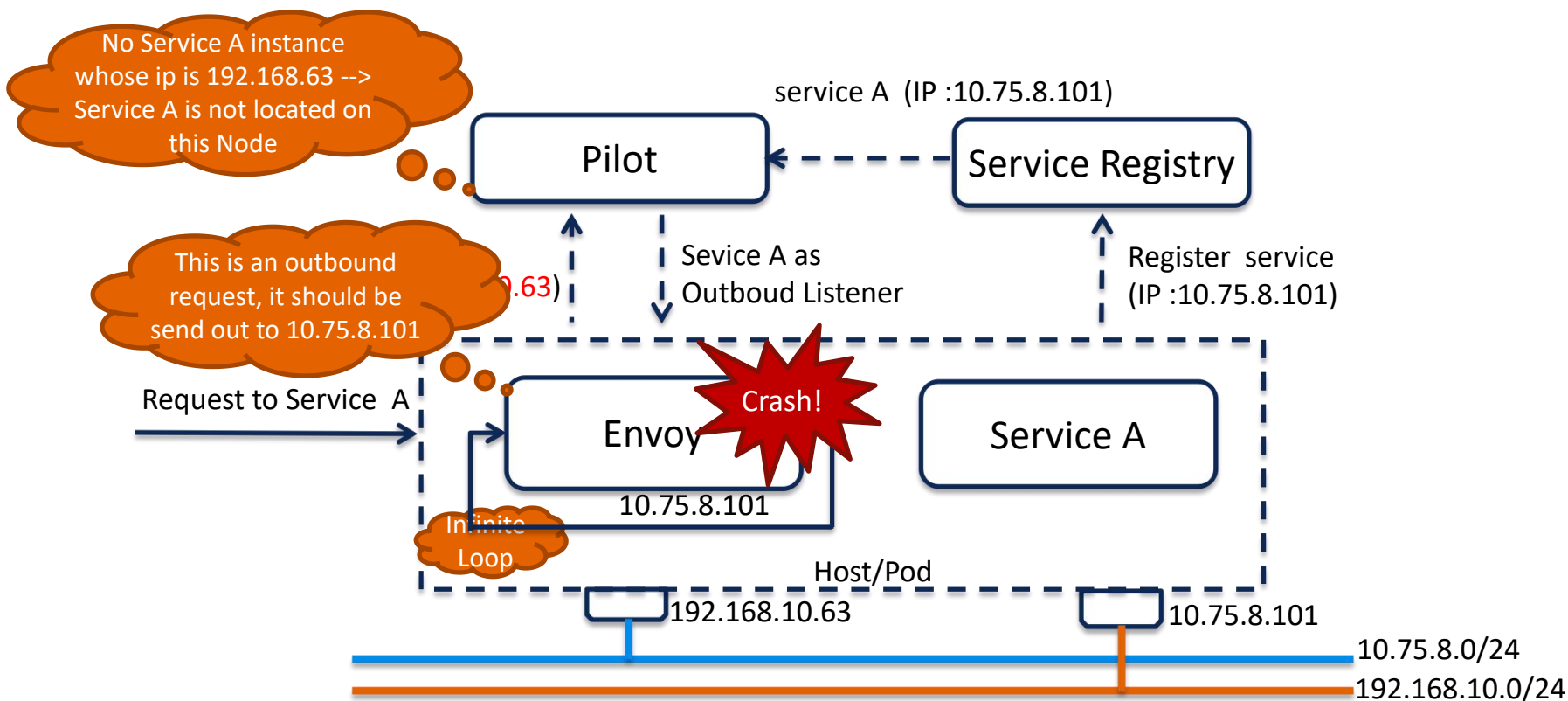
- 使用不同端口为不同租户提供访问入口
- 租户间的隔离和访问控制
- 用户层面的访问控制
- 按用户的API访问限流
- API访问日志和计费

Service Mesh: 统一的微服务通信管理

- 服务发现
- 负载均衡
- 重试, 断路器
- 故障注入
- 分布式调用跟踪
- Metrics 收集

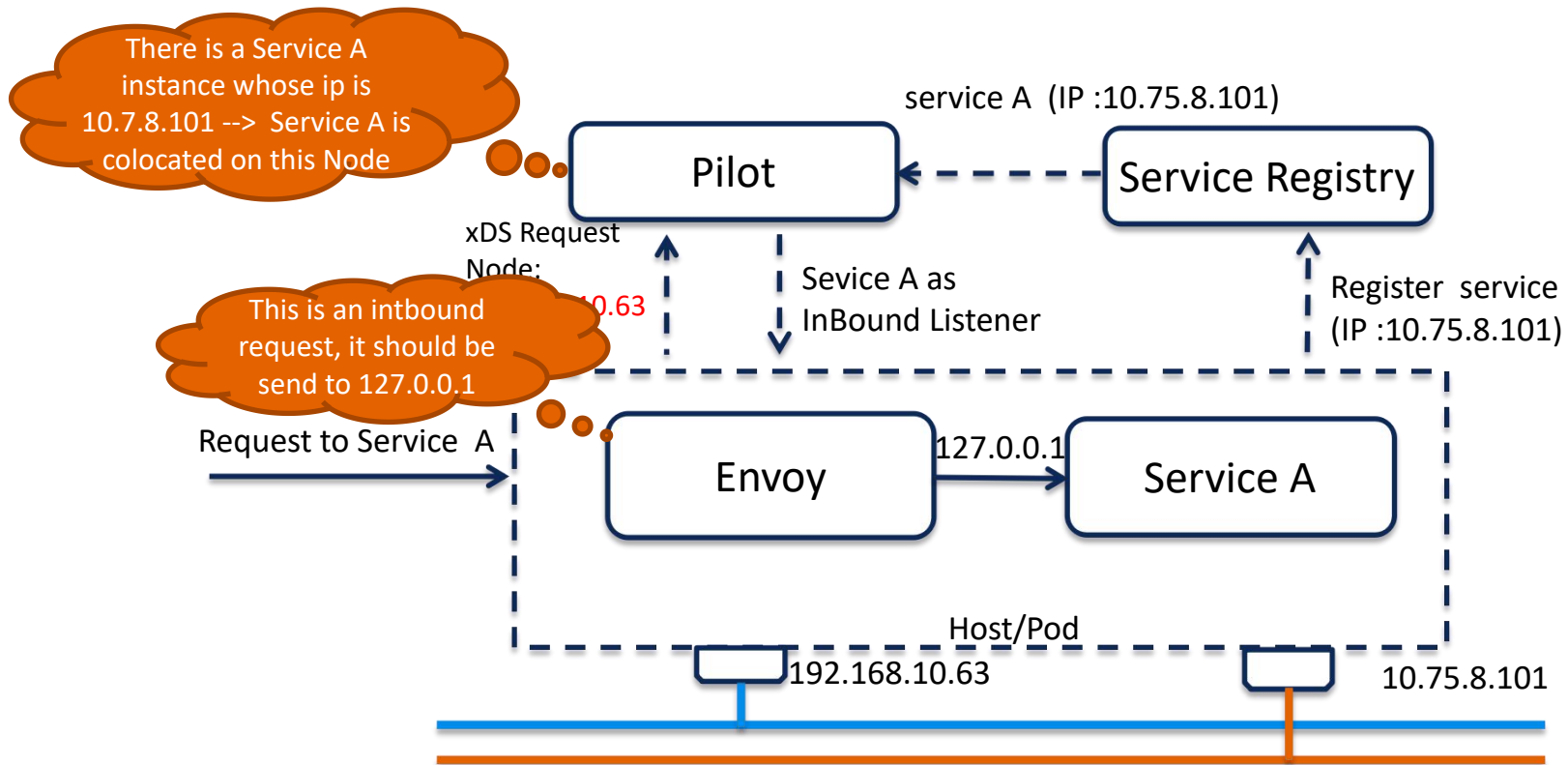
产品化增强-支持多网络平面

Istio1.0中不支持多网络平面，当服务地址和Envoy地址分别位于两个网络上时，会导致转发请求时发生死循环，导致socket耗尽，Envoy不停重启。



产品化增强-支持多网络平面

我们对Istio的代码进行了改造，增加了多网络平面支持。



产品化增强-TCP Service的处理

是否需要将TCP Service纳入Service Mesh管控？

- 收益

- TCP Service可以享受流量管理，可见性，策略控制等Istio承诺的益处

- 成本

- Istio不理解TCP上的应用层协议，其对TCP Service的缺省处理会影响应用层逻辑
 - 要求对应用进行逐一分析，将会导致应用逻辑受到影响的TCP Service排除
- Istio中和HTTP Service 端口冲突会的TCP Service请求会被Envoy直接丢弃
 - 要求对应用进行改造，避免端口冲突

由于系统中绝大部分的微服务间通讯都是基于HTTP的，并且存量应用难以改动，因此将TCP纳入Service Mesh管控的成本远大于收益

产品化增强-TCP Service的处理

在Service Mesh中 Bypass TCP流量，让TCP请求跳过Service Mesh的处理，直接发送到原始请求目的地。

- 方案一：通过IPtables bypass TCP流量
通过IP段或者端口范围区分HTTP和其他TCP流量
- 需要对应用进行改造
- 方案二：在Envoy中 bypass TCP 流量
- 不需要对应用进行改造，但Envoy要具备区分TCP和HTTP流量的能力，需对Envoy进行改造

改造方案：

- Envoy侧：通过一个自定义的envoy listener filter区分HTTP和非HTTP的TCP流量
- Pilot侧：修改Pilot下发的LDS配置，将TCP流量转到一个指定的filter chain 处理，通过tcp_proxy filter将TCP请求发往Passthroughfilter，以达到bypass TCP流量的目的。





















```
listener
├── name: 0.0.0.0_12011
├── address
│   └── socket_address
│       ├── address: 0.0.0.0
│       └── port_value: 12011
├── filter_chains:
│   ├── 0
│   │   ├── filter_chain_match
│   │   │   └── server_names:
│   │   │       └── HTTP.DATA.COM
│   │   └── filters:
│   │       ├── 0
│   │       │   ├── name: envoy.http_connection_manager
│   │       │   └── config
│   │       └── 1
│   │           ├── filter_chain_match
│   │           │   └── server_names:
│   │           │       └── HTTP.DATA.COM
│   │           └── filters:
│   │               ├── 0
│   │               │   ├── name: envoy.tcp_proxy
│   │               │   └── config
│   │               │       ├── stat_prefix: PassthroughCluster
│   │               │       └── cluster: PassthroughCluster
│   └── deprecated_v1
├── listener_filters:
│   ├── 0
│   │   └── name: envoy.listener.http_inspector
```


产品化增强-其他

- 管理界面：服务编排、灰度发布及服务管控
- Consul Registry故障修复及效率优化
- IPV6支持
- 网络参数调优
- 容器镜像轻量化

上游开源社区参与情况

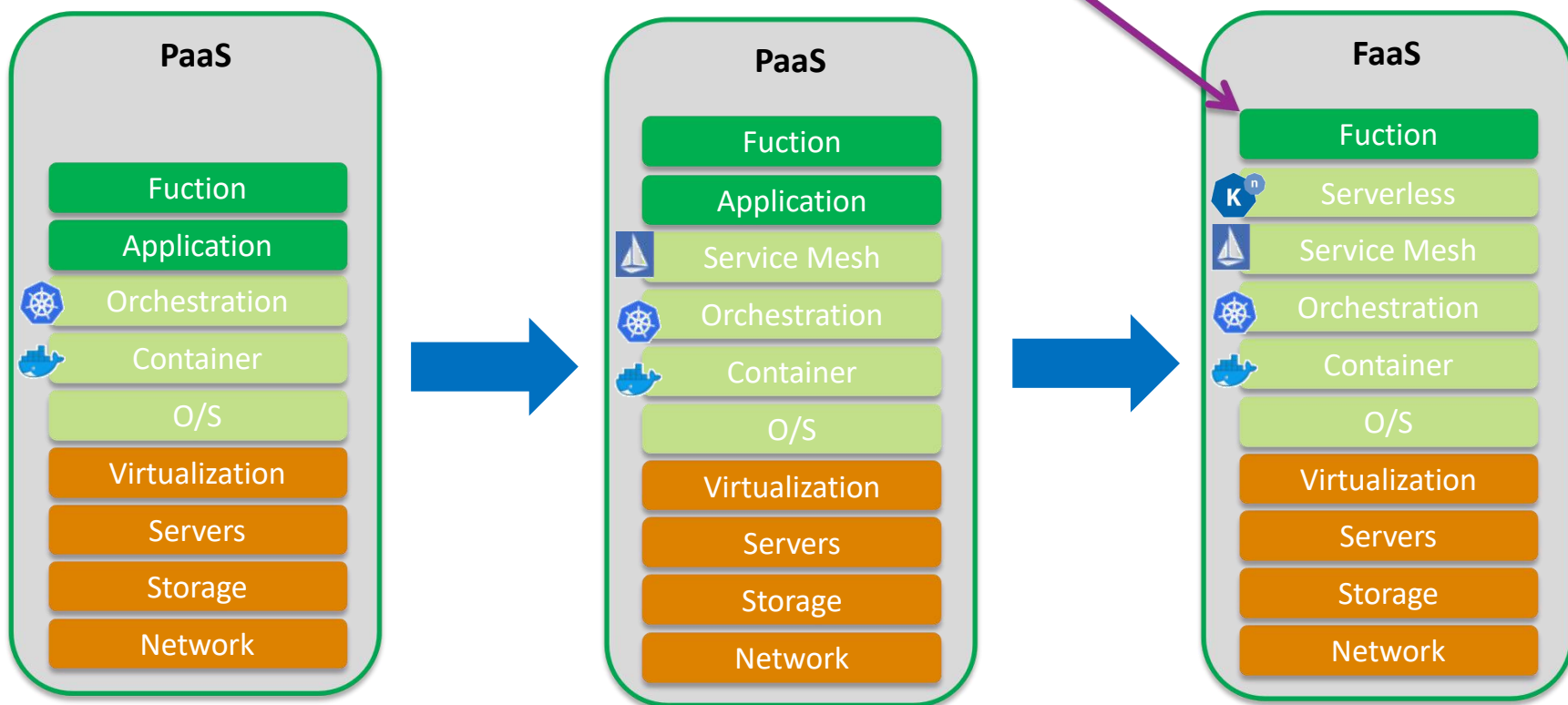
将产品化过程中的优化和改进以PR合入了上游Istio社区：包含一些通用的故障修复、性能优化及功能特性的

<input type="checkbox"/>	 Fix: Consul high CPU usage (#15509)  approved area/perf and scalability cla: yes lgtn   4
<small>#15510 by zhaohuabing was merged 15 hours ago • Approved</small>	
<input type="checkbox"/>	 Remove warn log message of ignored Consul service tag  approved area/user experience cla: yes lgtn   8
<small>#15452 by zhaohuabing was merged 11 days ago • Approved</small>	
<input type="checkbox"/>	 Avoid unnecessary service change events(#11971)  cla: yes ok-to-test   4
<small>#12148 by zhaohuabing was merged on Mar 1 • Approved</small>	
<input type="checkbox"/>	 Use ServiceMeta to convey the protocol and other service properties  approved cla: yes lgtn   13
<small>#9713 by zhaohuabing was merged on Nov 10, 2018 • Approved</small>	
<input type="checkbox"/>	 Support multiple network interfaces(#9441)  approved cla: yes   70
<small>#9688 by zhaohuabing was merged on Dec 15, 2018 • Approved</small>	

Service Mesh 发展趋势-下沉为云基础设施

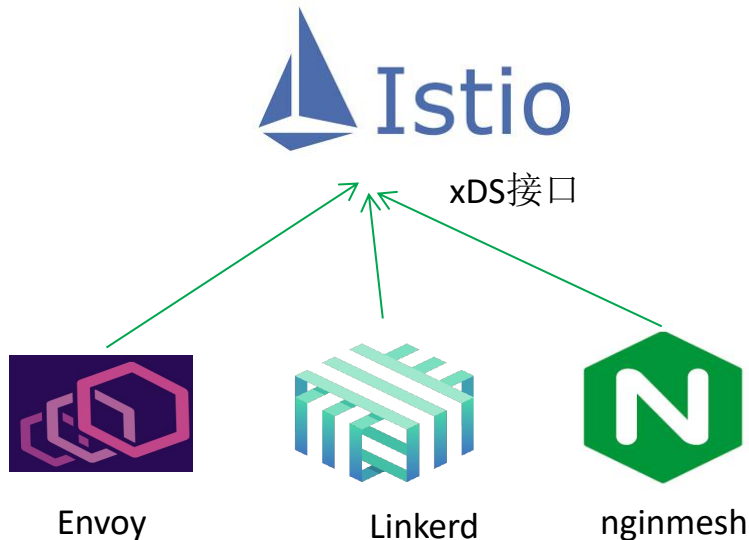
Service Mesh简化了微服务的应用开发，但Service Mesh层自身的技术复杂度较高，实现和运维都比较困难。要享用Service Mesh带来的便利，最方便的方式便是使用云提供上的Service Mesh托管服务，将底层的复杂性交给云提供商。

Leave all the others to the platform except **business logic**



Service Mesh 发展趋势-标准化

Istio 1.0及以前：Istio横空出世，其他项目向Istio的大旗靠拢，要么作为数据面和其集成，要么基于Istio之上进行开发。



- 2017年初 Buoyant CEO William Morgan正式提出Service Mesh的定义
- 2017.1 Linkerd作为第一个Service Mesh项目加入CNCF
- 2017.4 Google, IBM和Lyft公开Istio项目：Istio提出了控制面和标准数据面接口（xDS）的概念，占领Service Mesh架构高度
- 2017年5月发布Istio 0.1版本
- 2017年7月 Linkerd宣布其1.1.1版本支持和Istio集成
- 2017年8月Nginx开源基于Istio的数据面项目Nignmesh

Service Mesh 发展趋势-标准化

2018-2019: Service Mesh生态蓬勃发展，多个Service Mesh开源及闭源项目相继涌现：

- Buoyant暗渡陈仓，发布Linkerd2，包括数据面和控制面组件，不再支持和Istio集成
- Consul 发布connect，支持Service mesh功能，支持采用envoy作为数据面
- 各公有云厂商纷纷发布Service Mesh管理服务
- Solo.io发布Service Mesh管理平台SuperGloo，支持多云，多Mesh的管理

随着Service Mesh的应用，**跨Service Mesh之间的互通和兼容性**开始成为一个亟待解决的问题。

多Mesh管理:



控制面项目:



数据面项目:



Service Mesh 发展趋势-标准化

- **数据面标准化**: Google主导在CNCF成立Universal Data Plane API (通用数据平面API) 工作组, 基于xDS为基础指定L4/L7的数据面标准接口,意在建立Istio的生态系统。
- **控制面标准化**: 微软牵头, 联合 Linkerd, HashiCorp, Solo等发起Service Mesh Interface (SMI), 定义控制面的标准规范, 以期望实现各个Service Mesh的互通性, 挑战目前Istio在控制面上的垄断地位。

基于Mesh的应用:

(跨Mesh的管理、流量调整、全局策略、全局可见性等)



Service Mesh Interface

控制面项目



Linkerd2



Consul



Traffic Director



Azure Service
Fabric Mesh



AWS App Mesh

Universal Data Plane API

数据面项目



Envoy



Linkerd2-proxy



Consul connect



Gloo



AMBASSADOR

谢谢!



5G 先锋

