

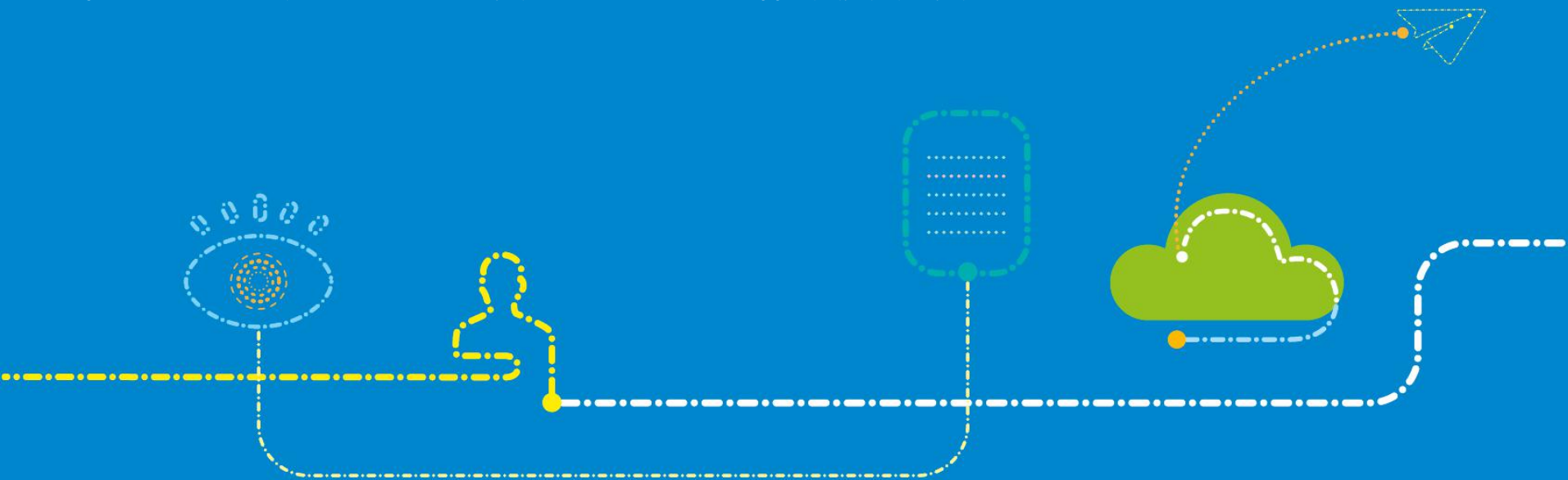
中信银行Service Mesh交流

ZTE中兴

中兴Service Mesh产品化实践之路

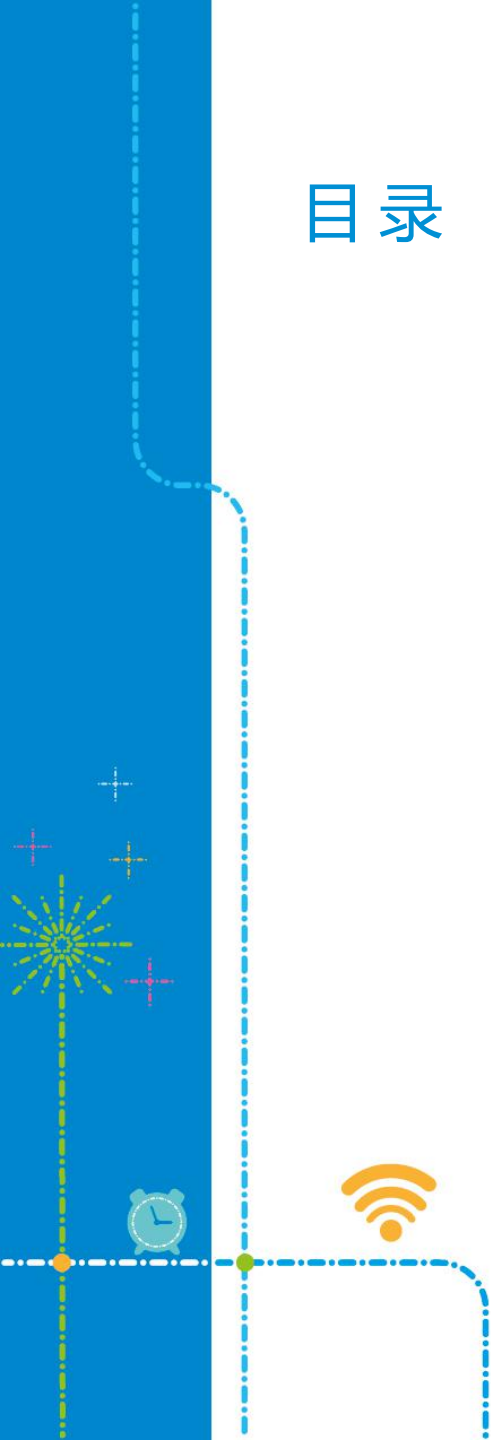
赵化冰

中兴通讯 软件专家/Istio社区成员/ServiceMesher治理委员会成员



目录

- 一. 服务网格的起源
- 二. 从SDN看服务网格
- 三. 中兴ServiceMesh实践
- 四. ServiceMesh发展趋势
- 五. 开放交流

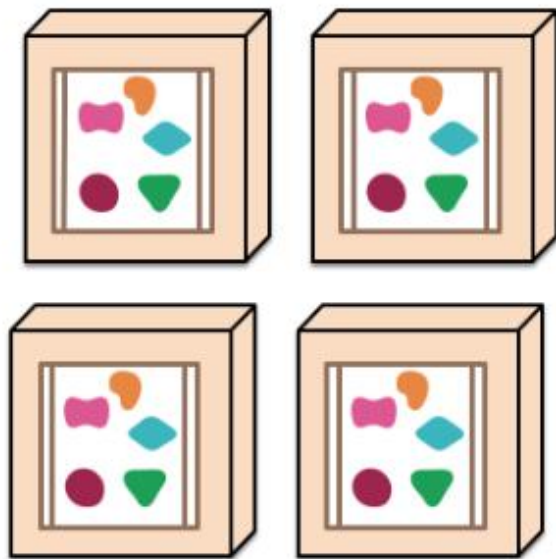


Monolith和Microservice架构模式

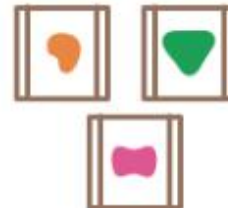
A monolithic application puts all its functionality into a single process...



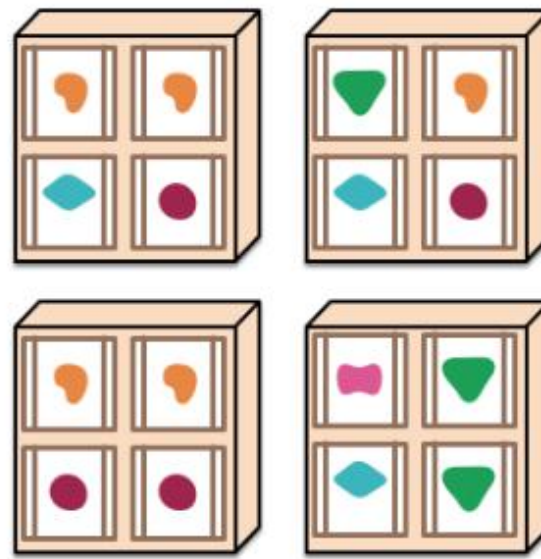
... and scales by replicating the monolith on multiple servers



A microservices architecture puts each element of functionality into a separate service...



... and scales by distributing these services across servers, replicating as needed.



Microservice 架构带来的挑战

微服务架构引入了**分布式系统**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

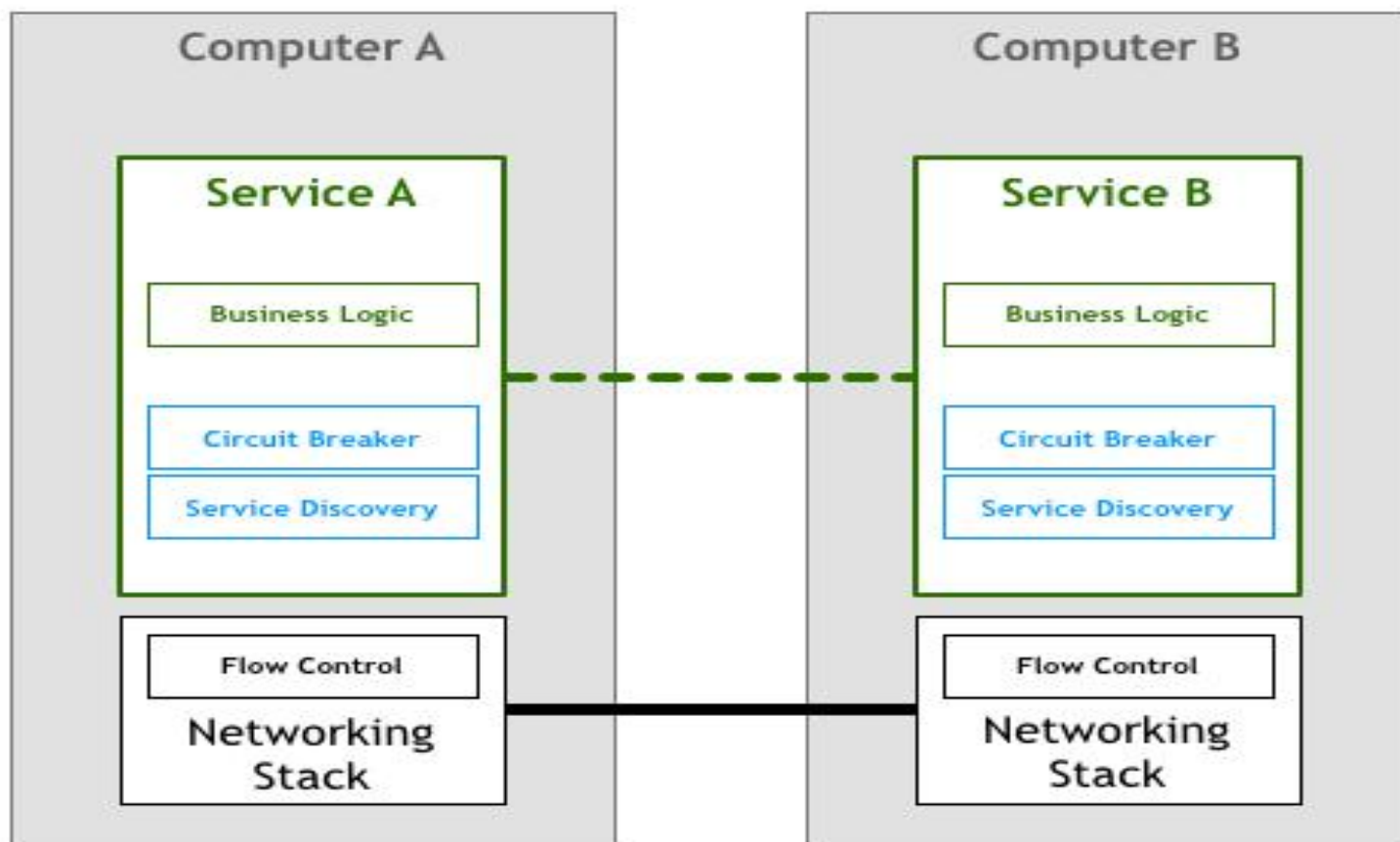
方案：

- 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。

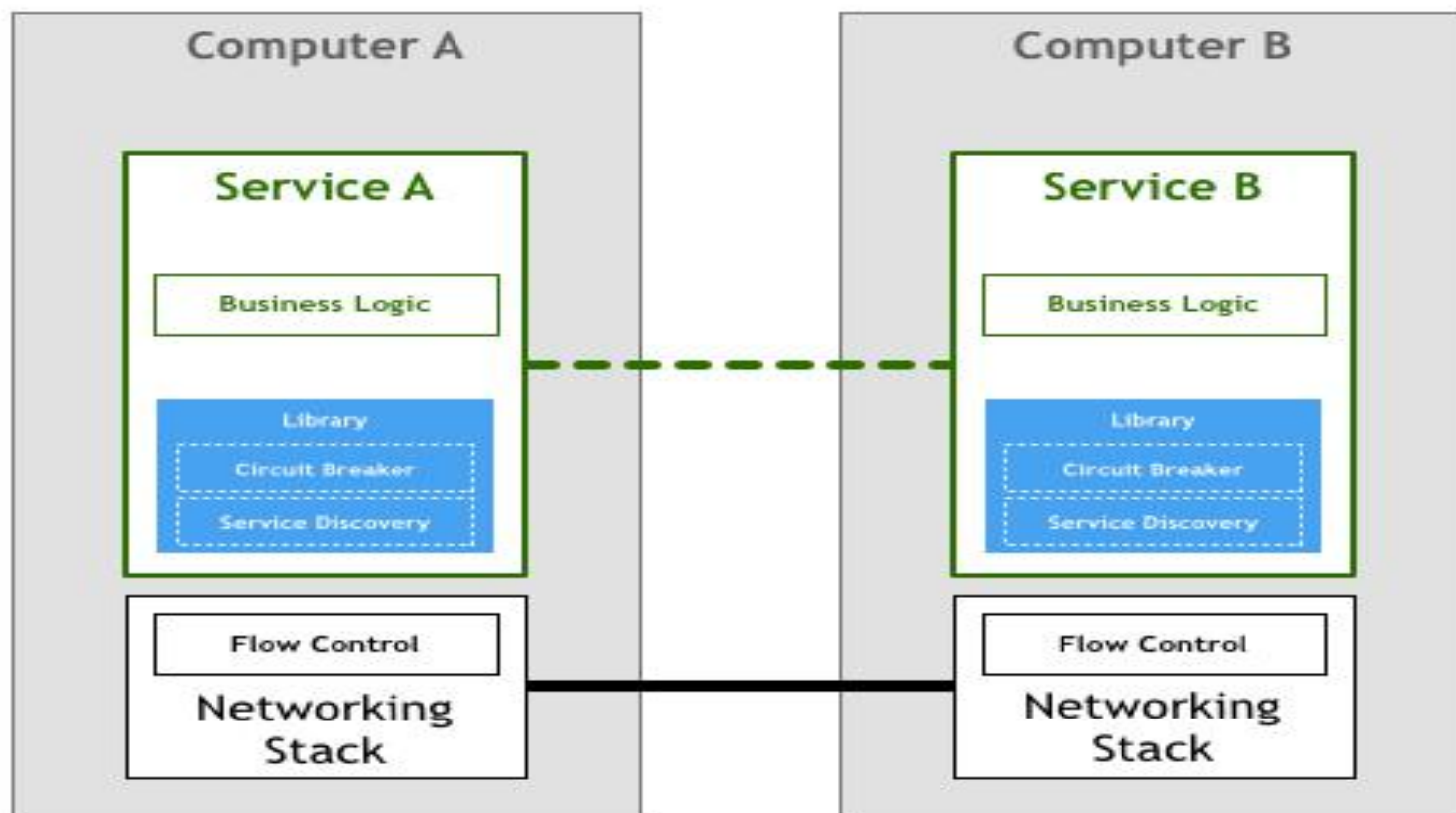
解决方案：原始时代

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。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，有个小问题冒出来了☺

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。
- 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。

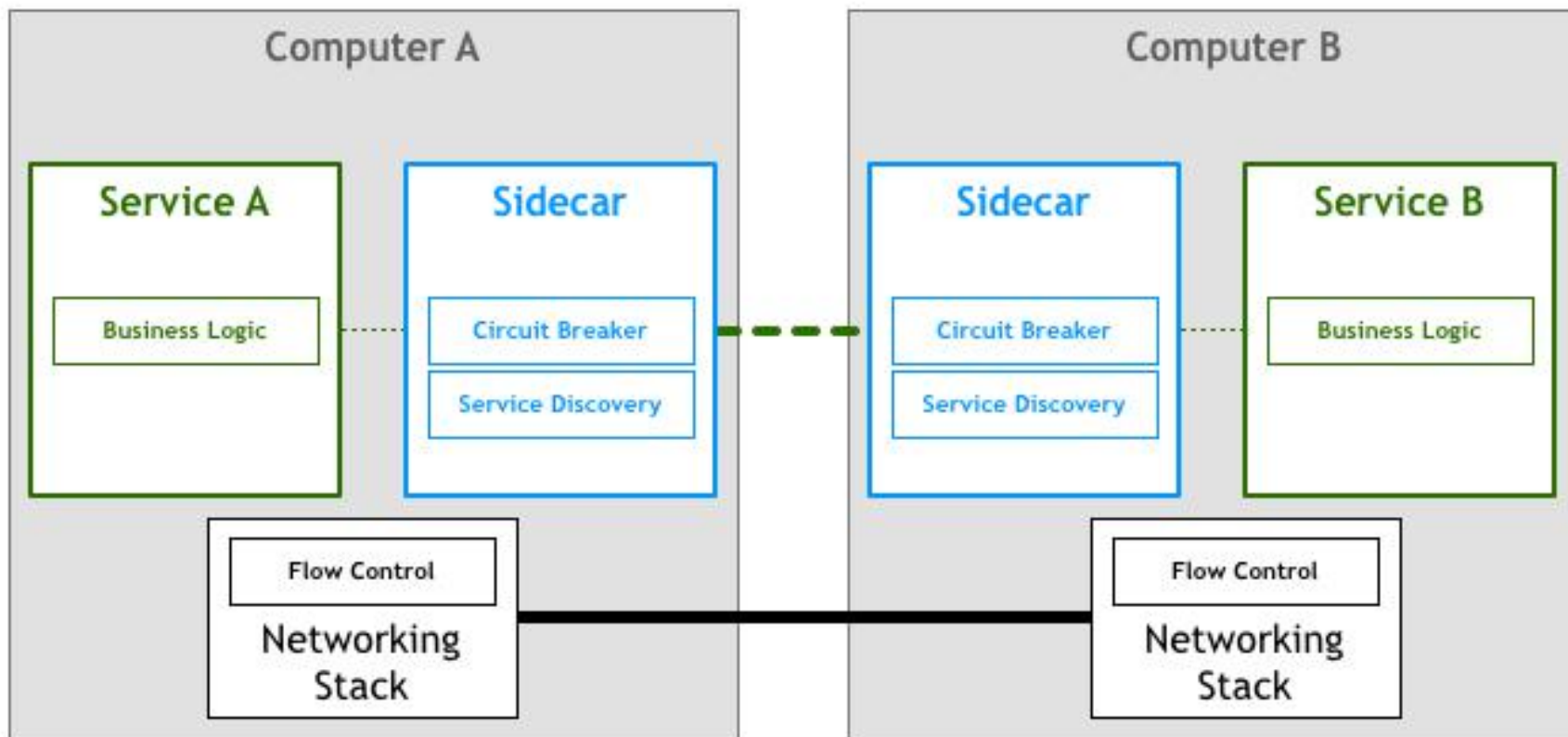
我们需要开发/
维护多少种类
库？



说好的多语言呢？

解决方案：边车模式

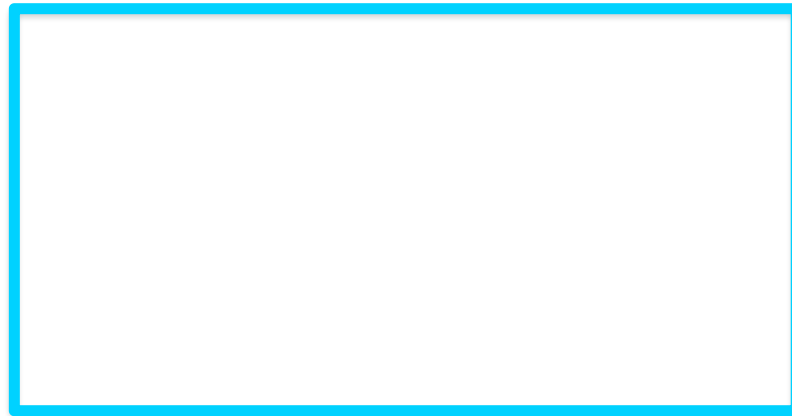
应用程序在进行网络通信时并不需要关注TCP/IP协议栈的实现，微服务需要关注服务间的通信细节吗？



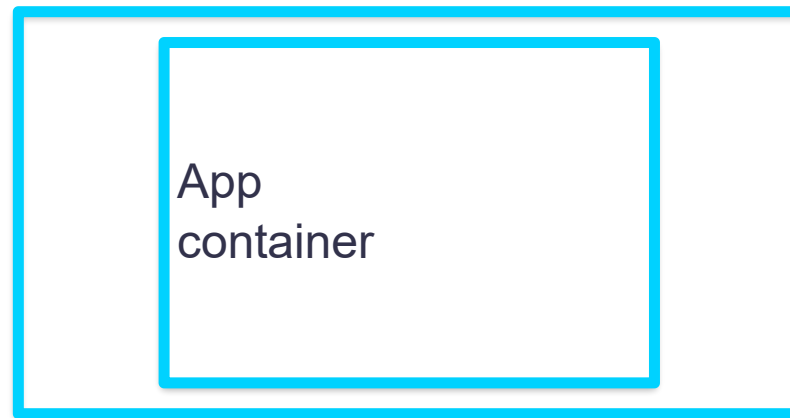
边车模式：将为微服务提供通信服务的这部分逻辑从应用程序进程中抽取出来，作为一个单独的进程进行部署，并将其作为服务间的通信代理。

让我们来回顾一下微服务架构的演进过程

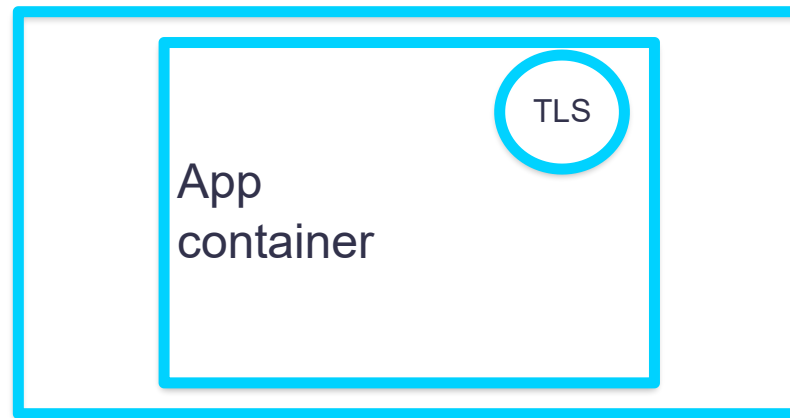
Before



Before



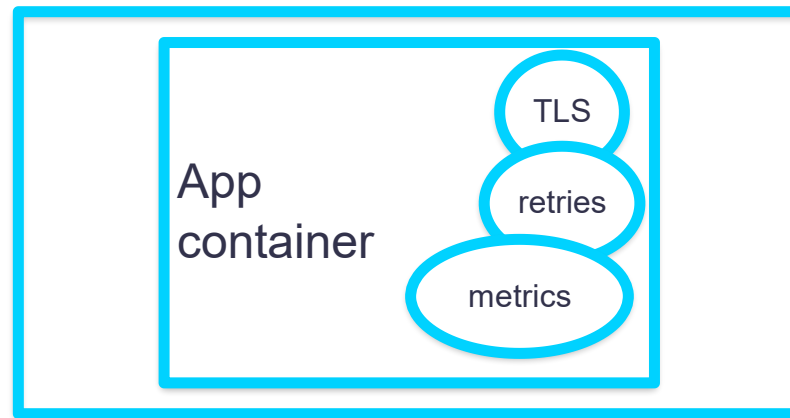
Before



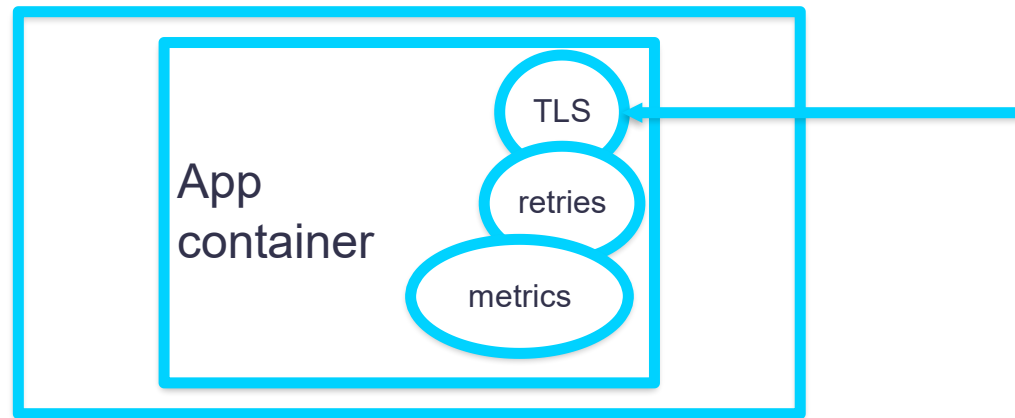
Before



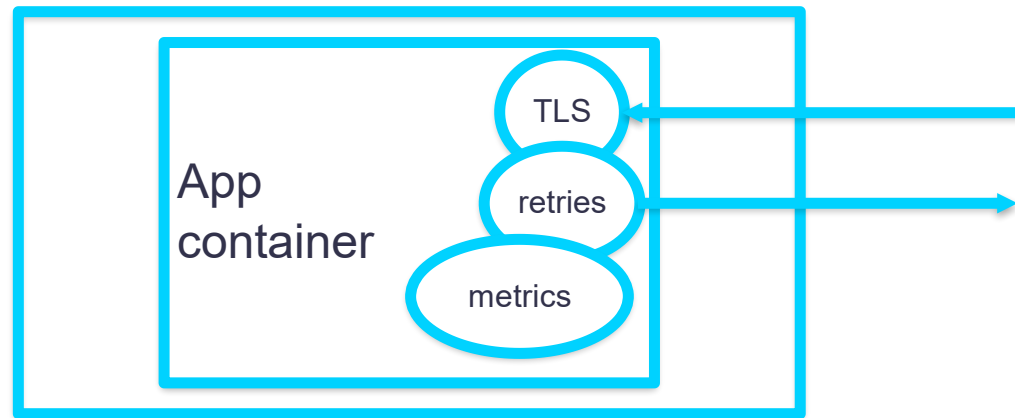
Before



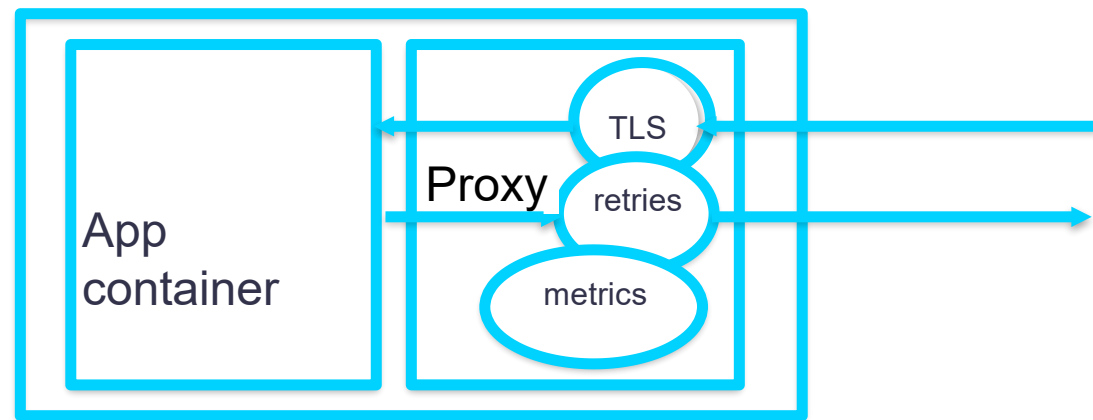
Before



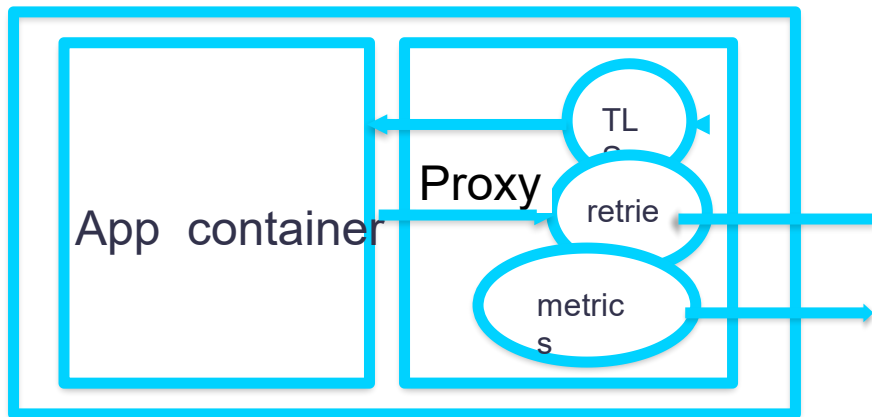
Before



After

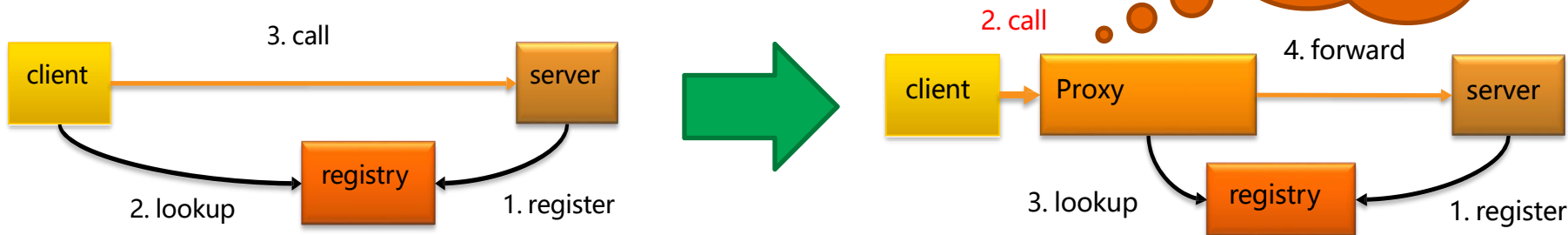


微服务通讯代理的这种部署方式被形象地称为“**Sidecar**”，即三轮摩托车的“挎斗”



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

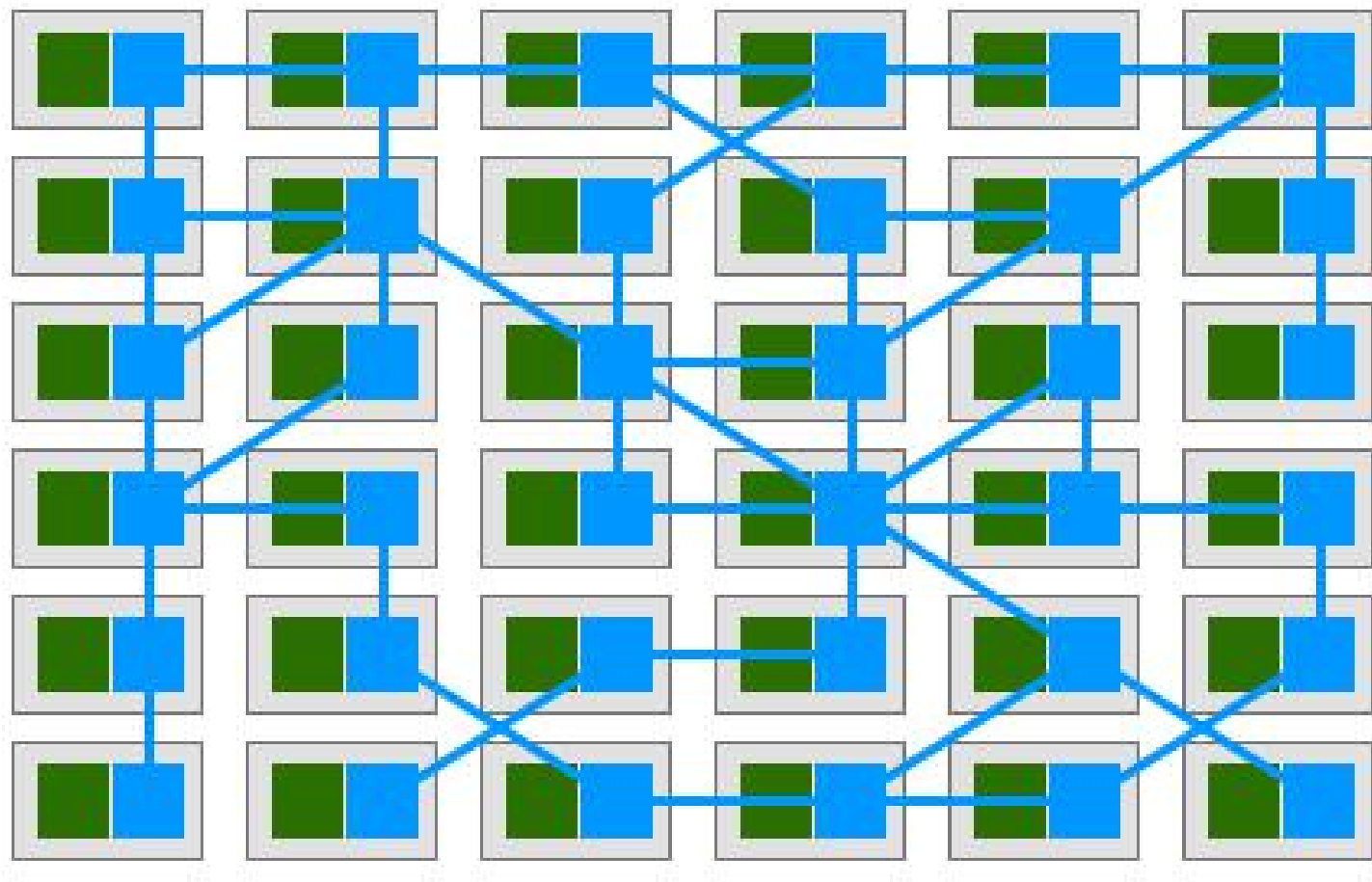
服务间的通信层处理被迁移到以“挎斗”方式部



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”

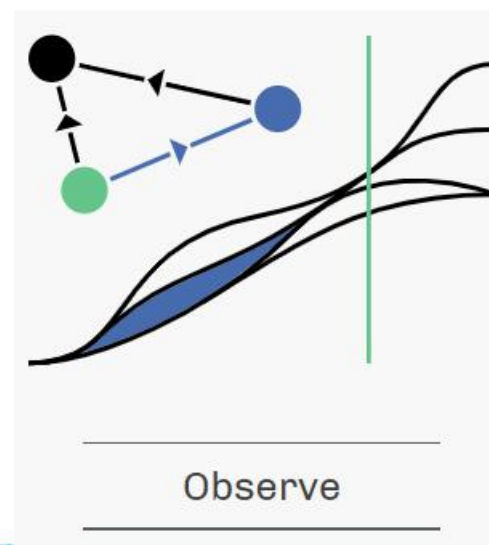
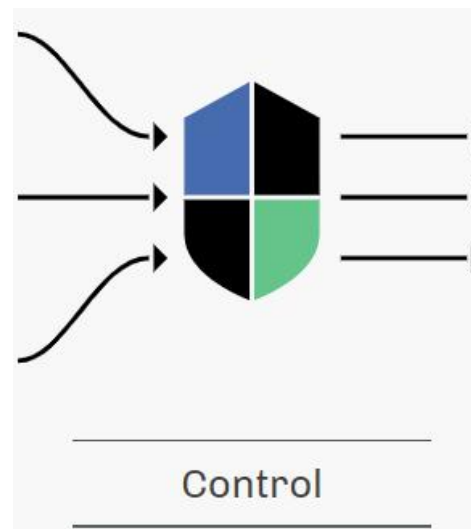
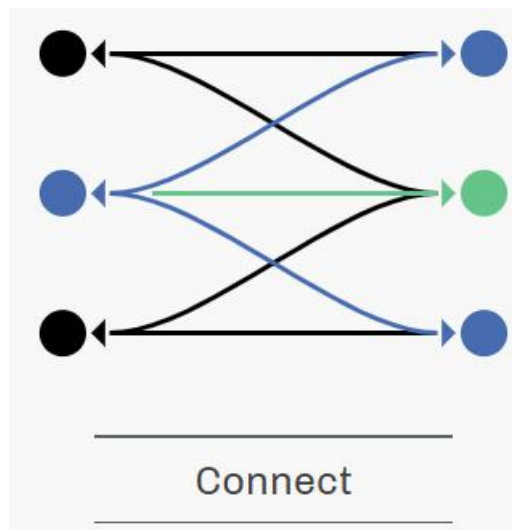


什么是Service Mesh? - by Willian Morgan(Buoyant)

A service mesh is a dedicated infrastructure layer for handling service-to-service communication. It's responsible for the reliable delivery of requests through the complex topology of services that comprise a modern, cloud native application. In practice, the service mesh is typically implemented as an array of lightweight network proxies that are deployed alongside application code, without the application needing to be aware.

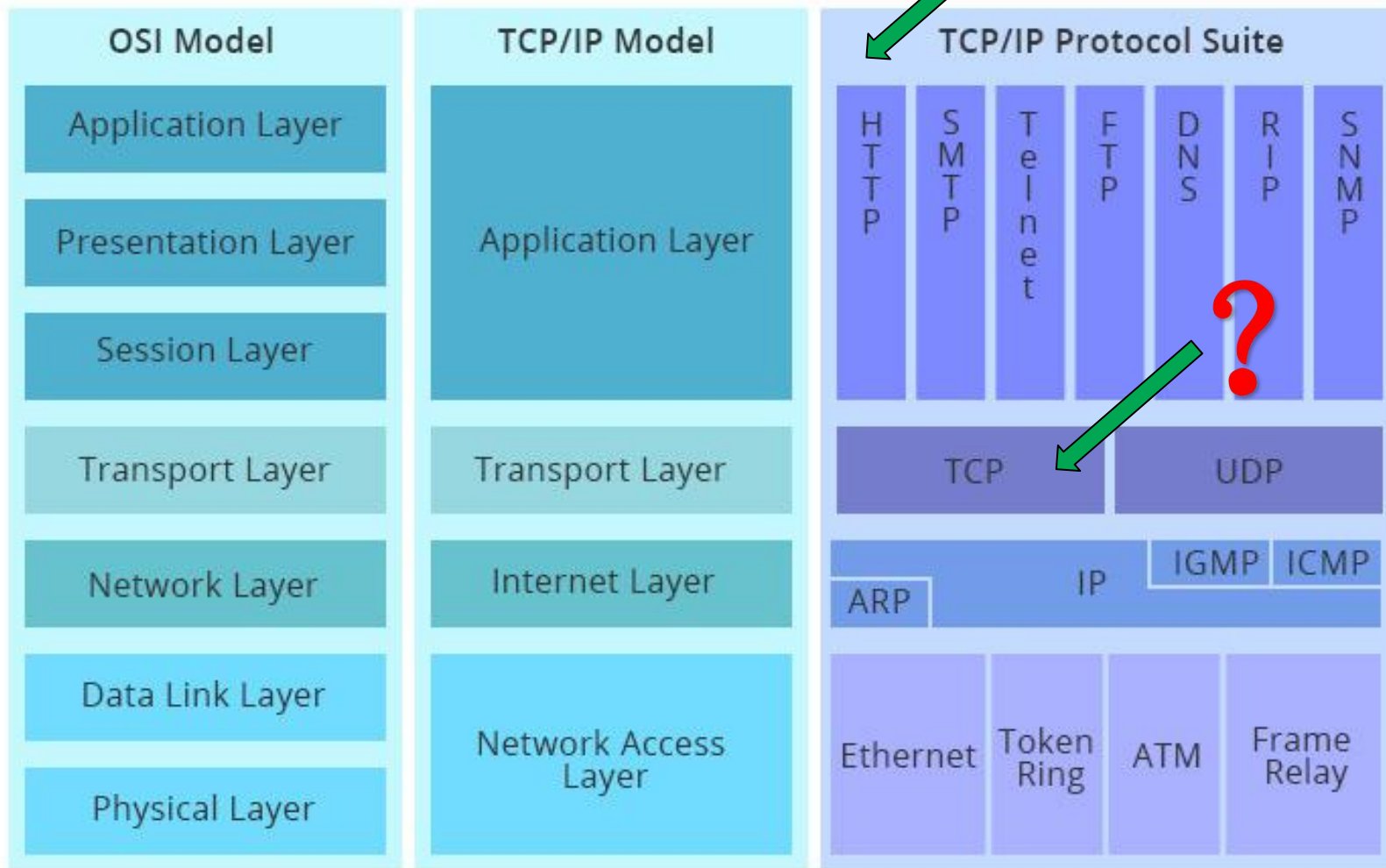
服务网格是一个**基础设施层**，用于处理服务间通信。云原生应用有着复杂的服务拓扑，服务网格负责在这些拓扑中**实现请求的可靠传递**。在实践中，服务网格通常实现为一组**轻量级网络代理**，它们与应用程序一起部署，但**对应用程序透明**。

什么是Service Mesh? - by Istio



什么是Service Mesh? - 从网络的视角

Service Mesh关注点



网络视角:

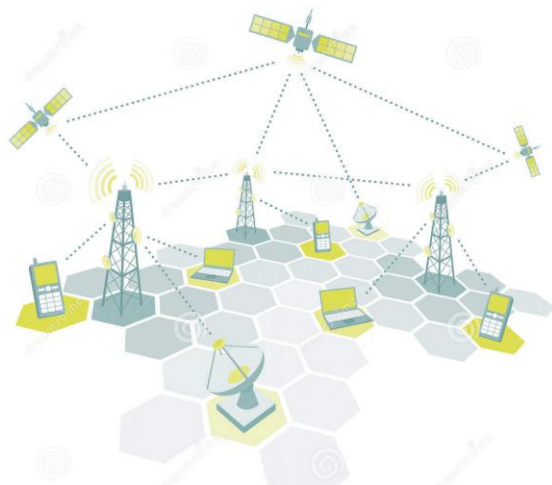
Service Mesh是一个主要针对七层的网络解决方案，解决的是服务间的连通问题

Service Mesh是下一代的SDN吗？

通信网络和微服务系统面临类似的问题：

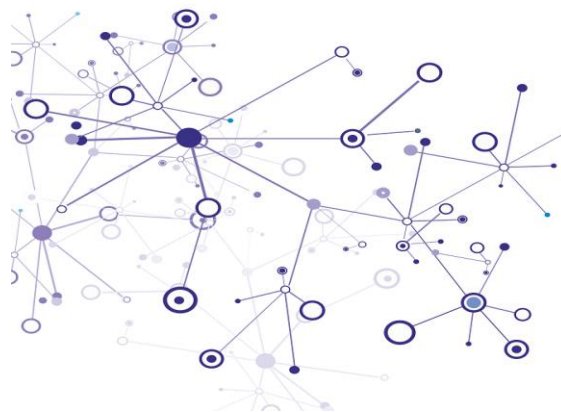
通信网络

- 互不兼容的专用设备
- 基于IP的通信缺乏质量保证
- 低效的业务部署和配置
- ...



微服务系统

- 互不兼容的代码库
- 不可靠的远程方法调用
- 低效的服务运维
- ...



Service Mesh是下一代的SDN吗？

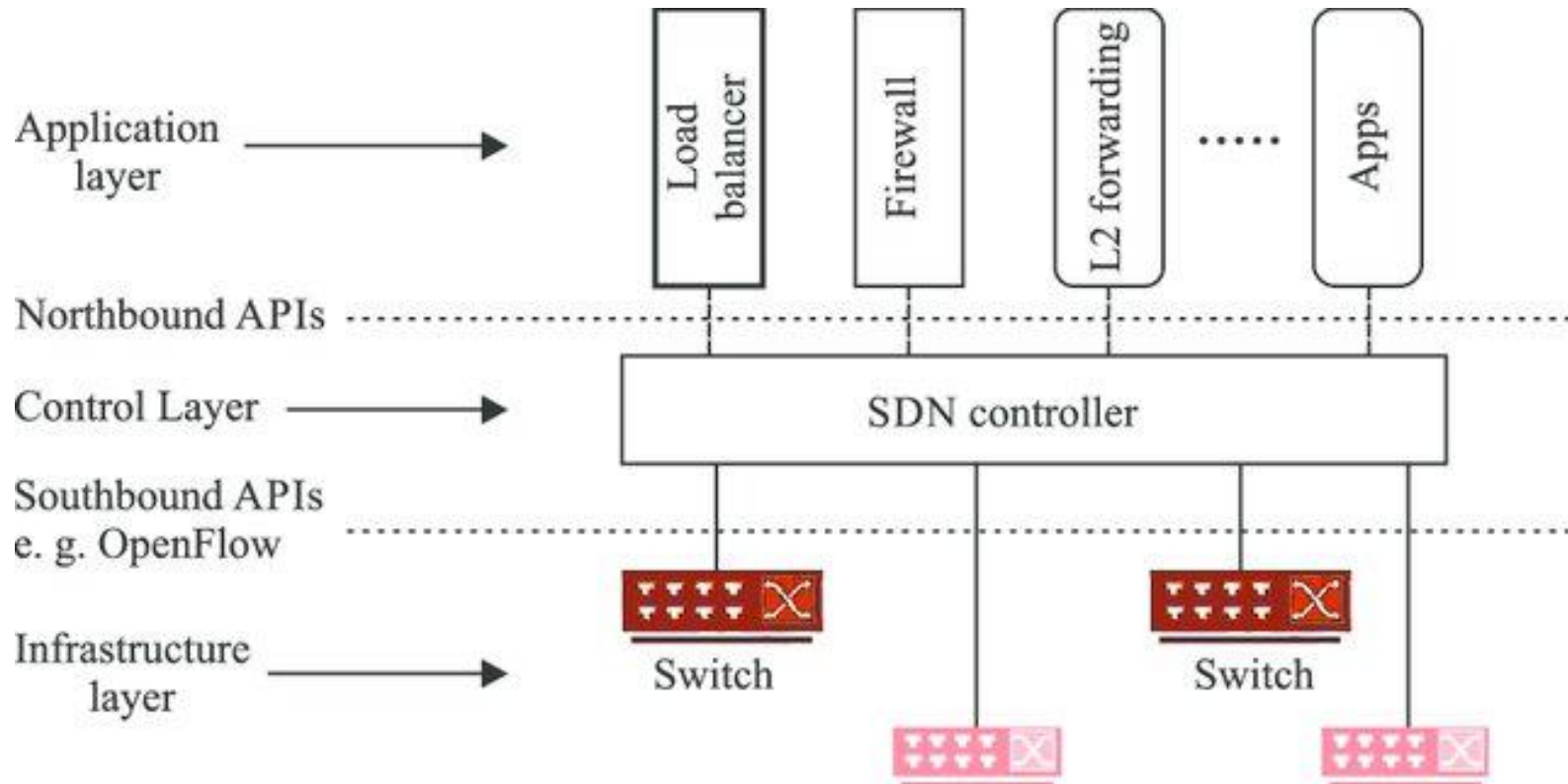
SDN : 主要关注1到4层

Service Mesh: 主要关注4到7层

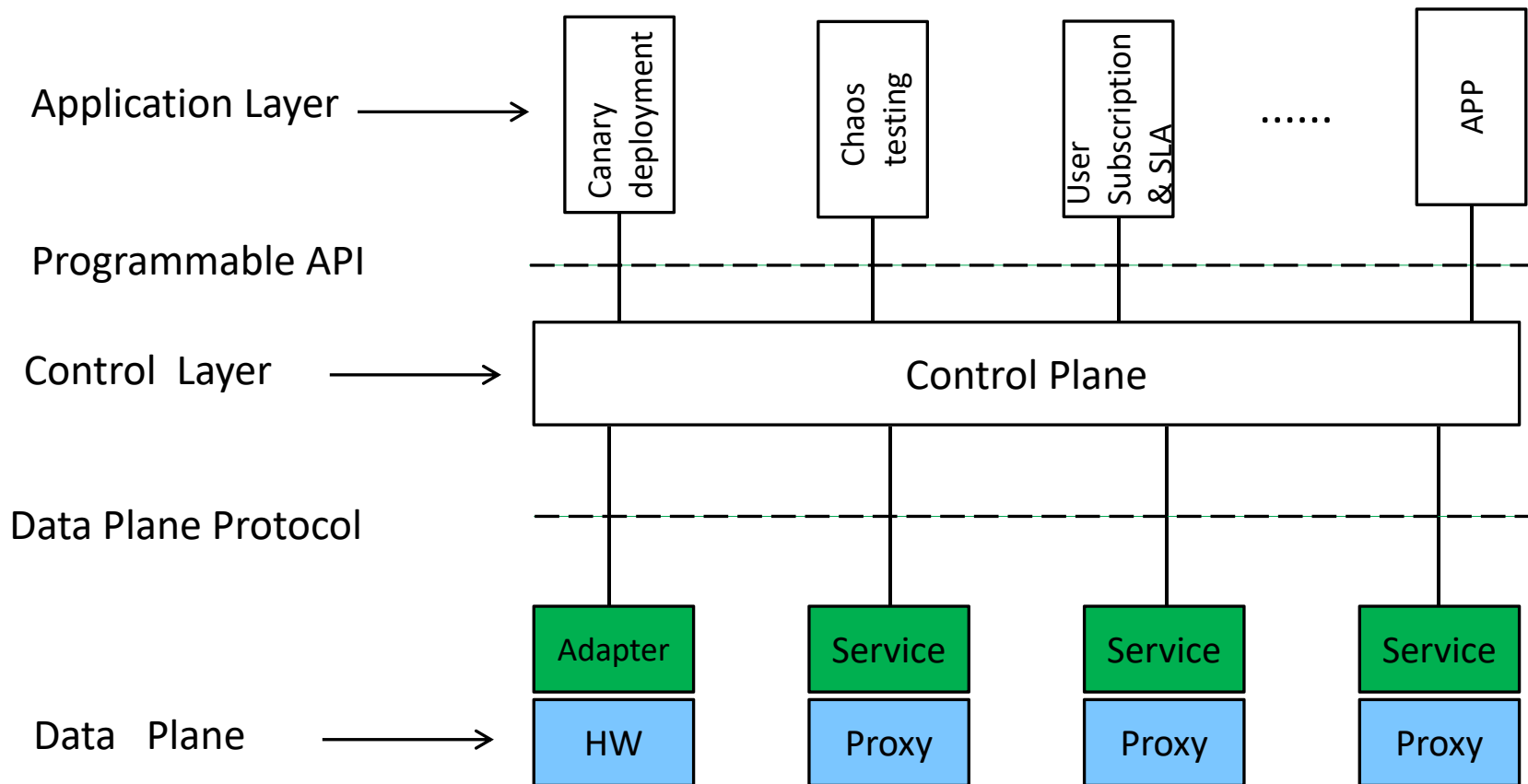
Network Layer	Look at	Solution
Layer 7+	Message Body	Service Mesh
Layer 7	HTTP Header	Service Mesh
Layer 4	TCP Port	SDN/Service Mesh?
Layer 3	IP Address/Protocol	SDN
Layer 2	MAC/VLAN	SDN
Layer 1	Input Port	SDN

类似的问题，不同的网络层次，通信网络的解决方案能为Service Mesh提供哪些借鉴？

通信网络的解决方案：SDN



微服务应用层通信的解决方案-Service Mesh

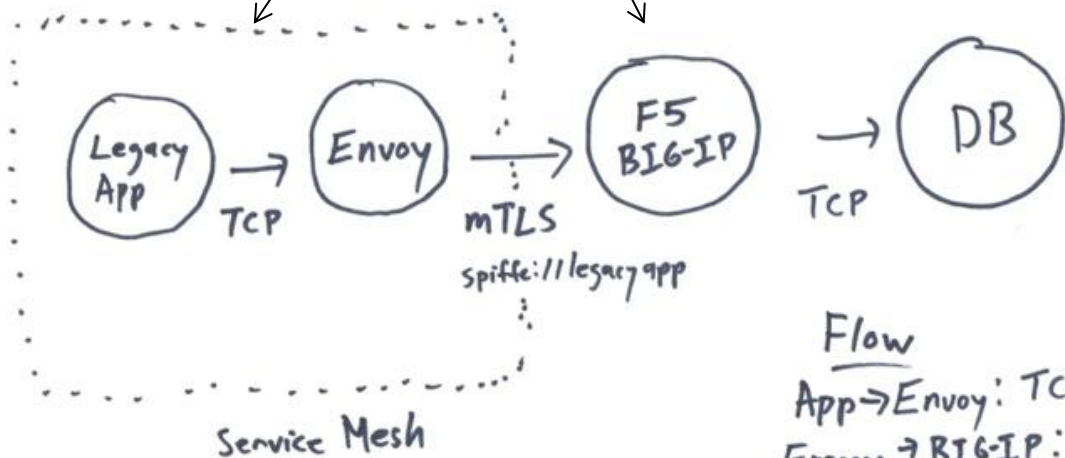


通过Service Mesh控制面统一管理F5和Envoy



Security Policy:

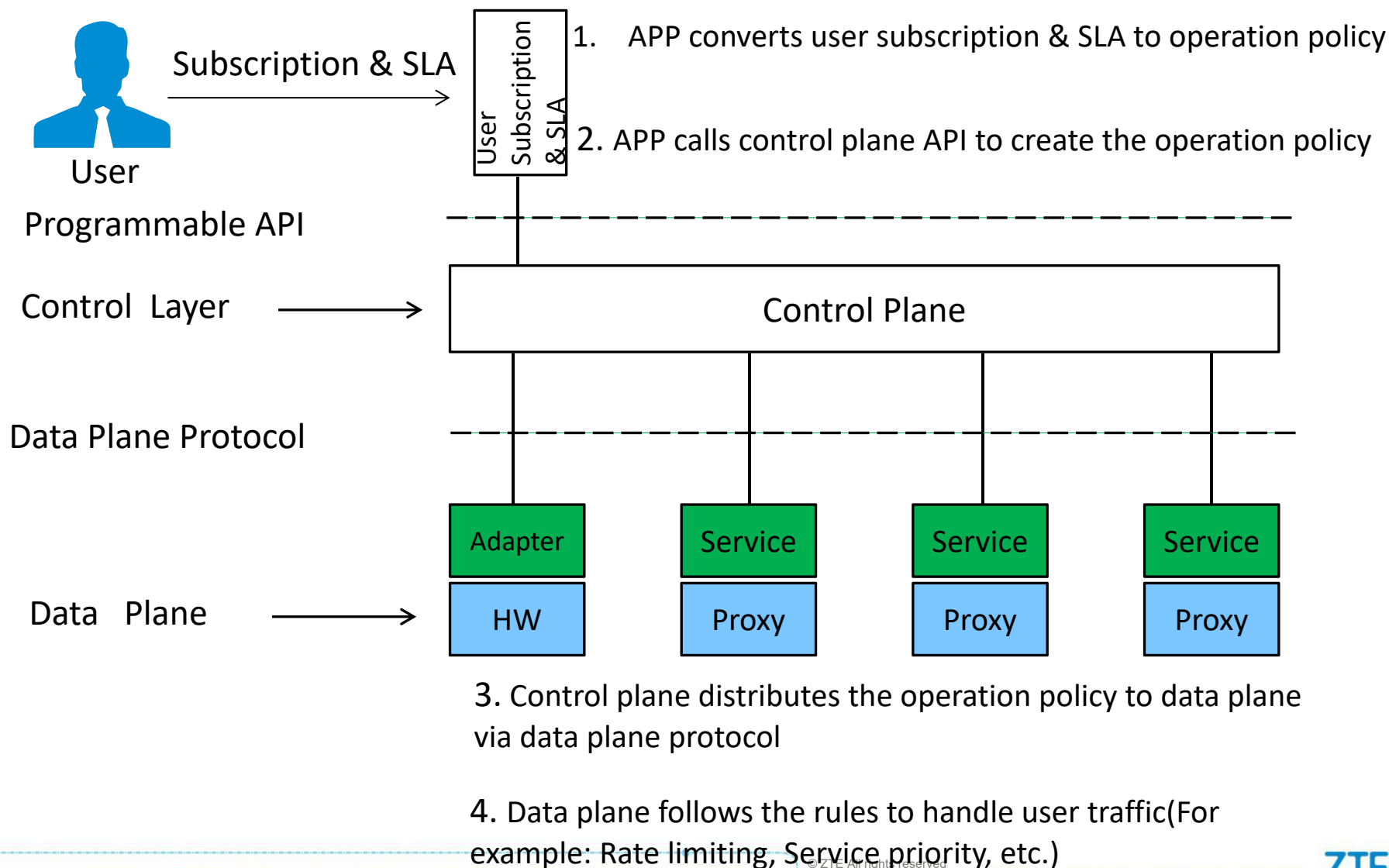
- Enable mTLS between services and the DB
- Only authorized services can access the DB



Flow
App → Envoy: TCP
Envoy → BIG-IP: mTLS
BIG-IP → DB: TCP

<https://aspenmesh.io/2019/03/expanding-service-mesh-without-envoy/>

App Example: User Subscription and SLA Management



总结：他山之石，可以攻玉

- 解决类似的问题：运维和通信的问题
- 相似的解决方案：数据面+控制面+应用
- 不同的协议层次：SDN 2-4层，Service Mesh 主要为7层

SDN对Service Mesh发展的启发：

➤ 北向接口

- 面向业务和运维
- 具有较高的抽象层次，不难提取统一的控制面标准？
- 主要面向layer 7及以上？
- SMI能否统一控制面标准？

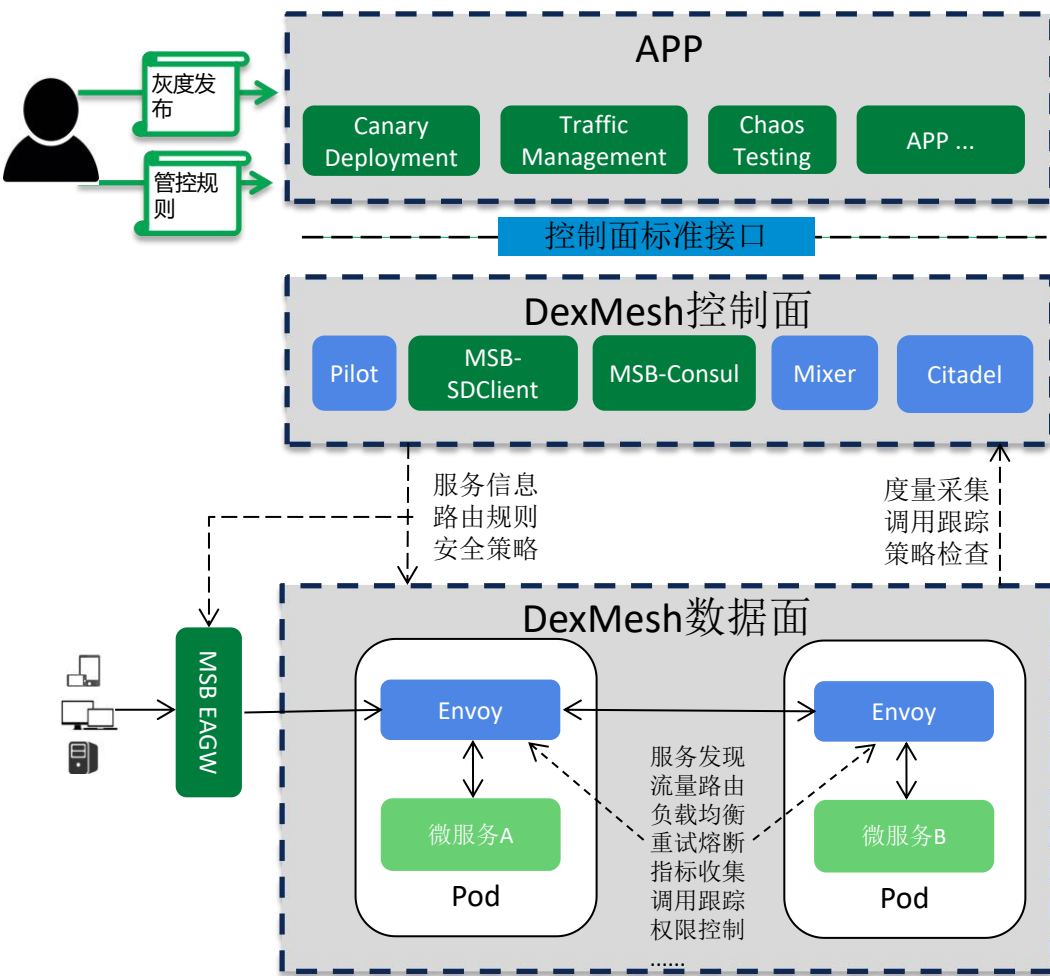
➤ 南向接口

- 面向流量和路由配置
- xDS v2将统一数据面标准？
- xDS接口包含有较多实现相关内容：Listener, Filter, 能否可以成为一个通用的接口协议？是否会出现Envoy之外的大量数据面实现？
- 建议：对xDS接口进行改进，采用通用+自定义扩展的方式，去掉实现细节

➤ Service Mesh的扩展

- 控制面对数据面软硬件的统一控制能力？
- 通过控制面API接入各种丰富的应用场景 - 下一个热点？

DexMesh架构-高层视图



基于开源Istio的服务网格实现方案:

- 使用Kubernetes的应用部署和管理能力, 但不使用其服务发现能力
- 使用自研的服务注册发现方案, 兼容现有系统
- 在网关处叠加ingress sidecar以通过网格控制面统一控制内外部流量
- 通过控制面标准接口开发灰度发布、流量控制、混沌测试等各种应用APP

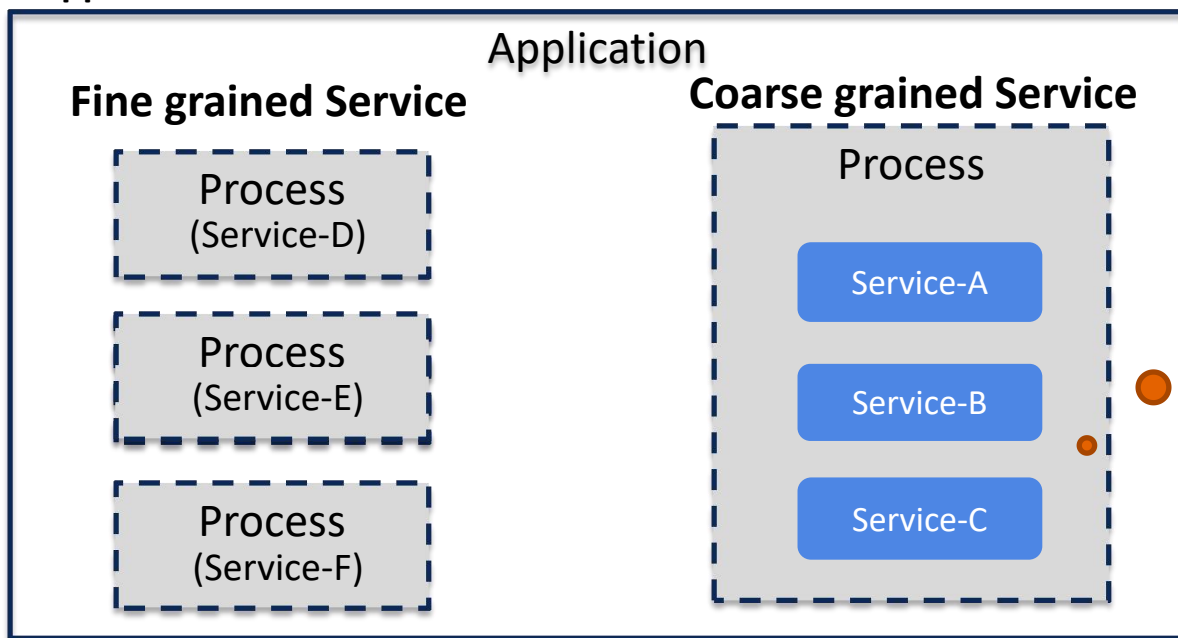
关键技术	DexMesh功能
运行环境	支持 PaaS运行环境
服务通信	服务发现、负载均衡、失败重试、熔断限流
调用跟踪	支持HTTP/Kafka消息的调用跟踪, 支持和Jaeger、Zipkin集成
流量控制	灰度发布、混沌测试
指标收集	调用成功率, 调用延迟, 错误码...
服务安全	服务认证、通信加密、操作鉴权

如何兼容存量应用/SOA服务？

需要同时兼容细粒度的微服务应用和粗粒度的SOA应用

- 服务的调用粒度和部署粒度存在不一致的情况
- 存量SOA类应用没有动力进行微服务拆分改造
- 资源有限的场景下，要求对部分微服务进行合并部署

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

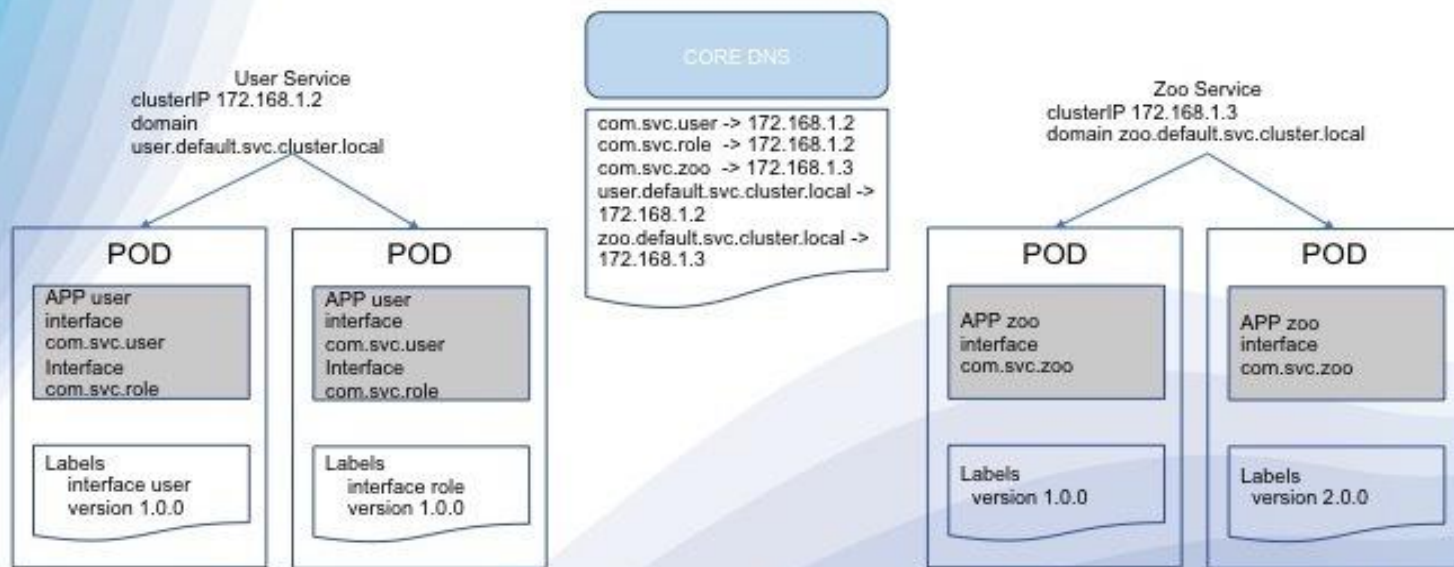


How to map multiple logic services inside one process to Istio service?

如何兼容存量应用/SOA服务-DNS映射方案

改造Kubernetes DNS服务，通过DNS将多个服务名映射到一个Kubernetes的Cluster IP上

落地形态 Services, PODS & DNS



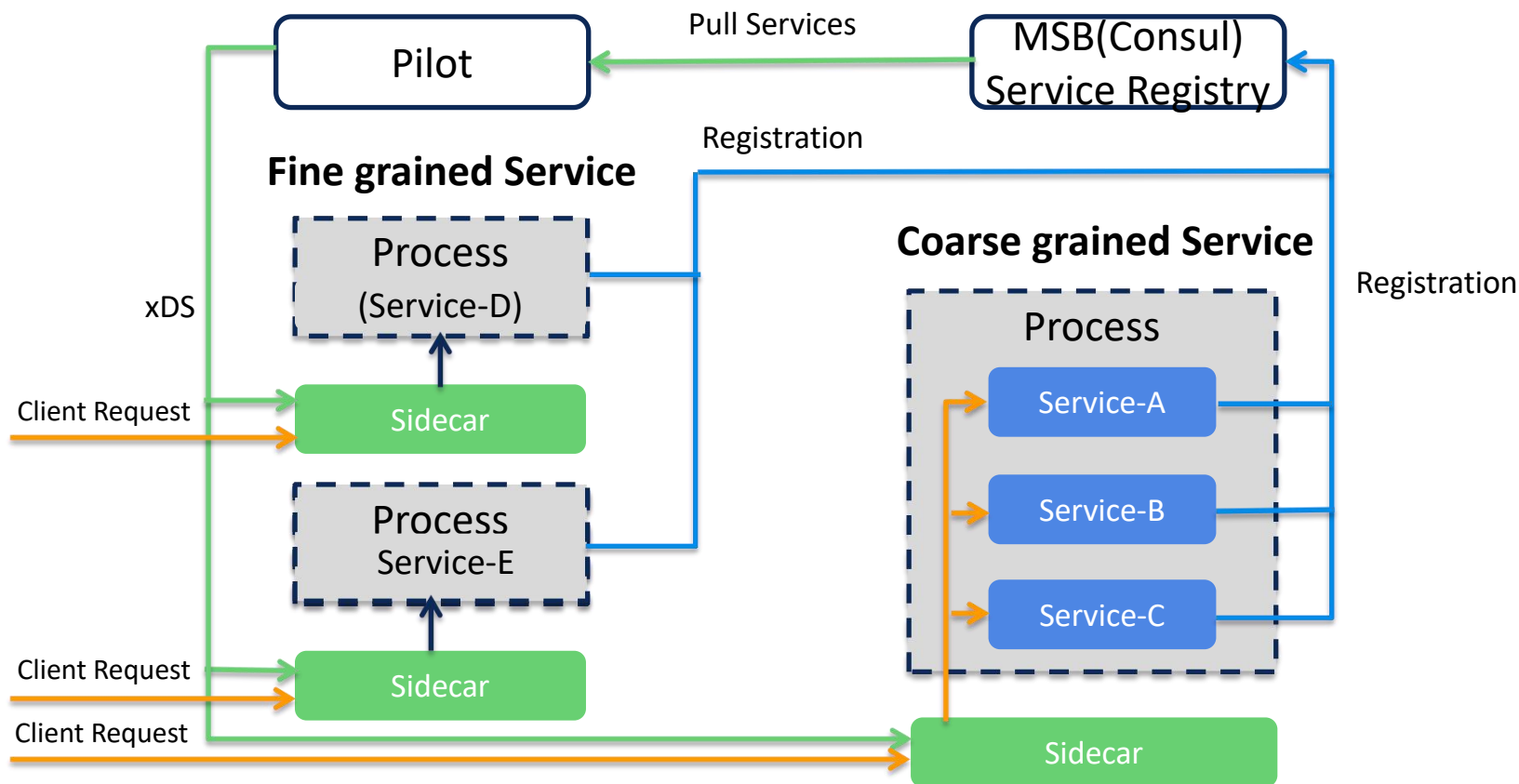
来自于蚂蚁金服在ServiceMesher社区的分享

该方案只解决了服务访问，未解决服务治理，有较大局限性，只建议作为服务改造期间的过渡方案：

- 服务注册依然是以应用名为基础，对应的 k8s service 和 service 上的 label也是应用级别
- 因此提供的服务治理功能，也是以 k8s 的 Service 为基本单位，包括灰度，蓝绿，版本拆分等所有的 Version Based Routing 功能
- 这意味着，只能进行应用级别的服务治理，而不能继续细分到接口级别

如何兼容存量应用/SOA服务-自建服务注册中心方案

采用自建服务注册中心，将一个应用进程按照业务逻辑注册为多个服务



- 在解决服务访问问题的同时，可以充分利用服务网格提供的所有治理能力
- 我们已有自建服务注册中心，只需要和Pilot进行集成
- 可以扩展Service Mesh，支持混合云架构

Consul Registry遇到的坑

1.0版本的Consul Registry只能算PoC（原型验证），远未达到产品要求
CPU占用率超高不下 (Pilot+Consul 占用冲高到 400%)

- TIME_WAIT Sockets 太多导致FD耗光

Consul Registry优化

- 增加数据缓存，减少无谓的Consul Catalog API调用
- 将Polling改为Watch，大幅降低Consul服务数据变化后的同步时延

优化效果：200个服务的规模下

- 控制面Pilot和Consul资源占用，CPU占用率降低了一个数量级
- 服务数据变化同步时延从分钟级降低到秒级
- Consul调用导致的TIME_WAIT Sockets数量减少到个位级

如何为服务网格选择入口网关？



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

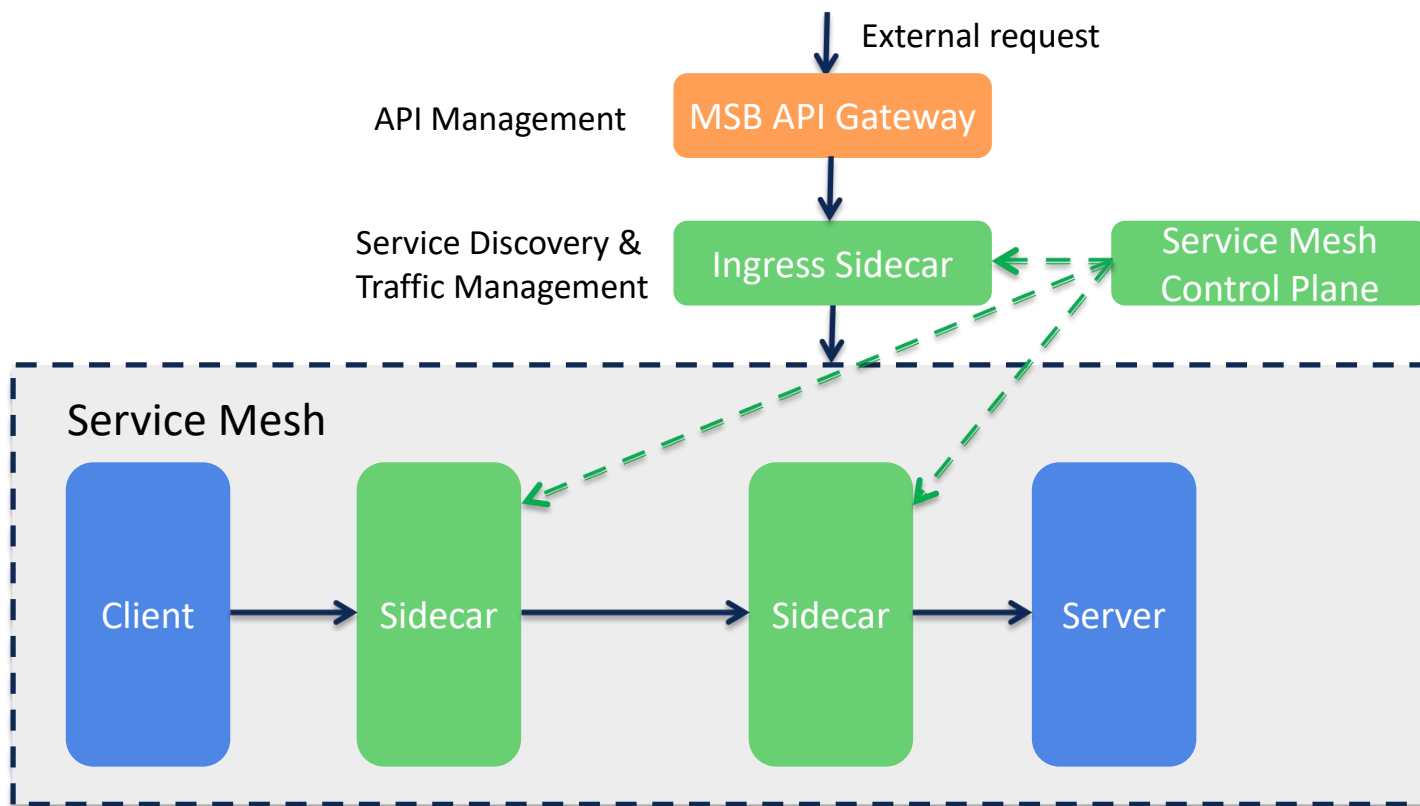


API Gateway

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

Service Mesh和API Gateway的分工与协同



API Gateway: 应用网关逻辑

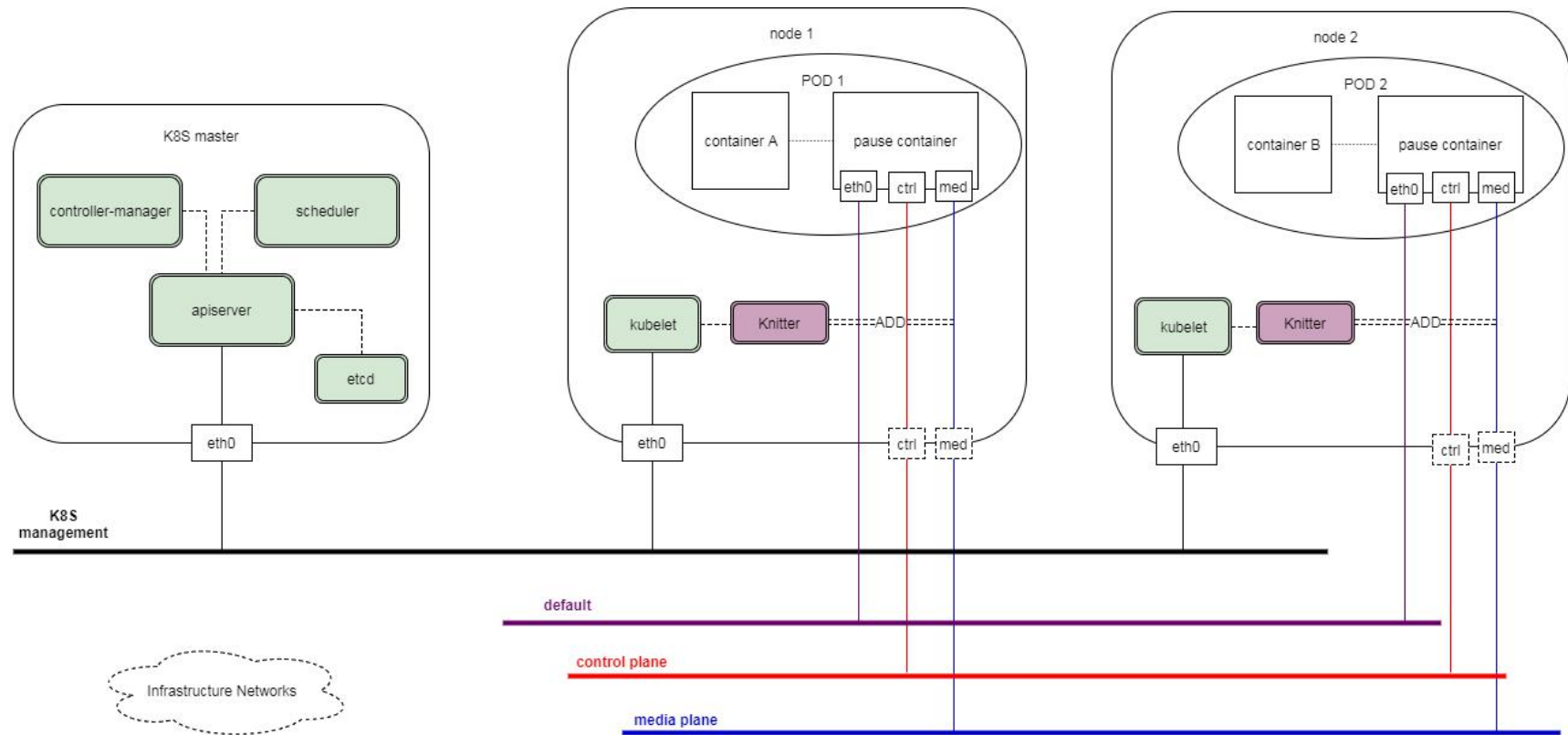
- 使用不同端口为不同租户提供访问入口
- 租户间的隔离和访问控制
- 用户层面的访问控制
- 按用户的API访问限流
- API访问日志和计费

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

拥抱NFV：支持多网络平面

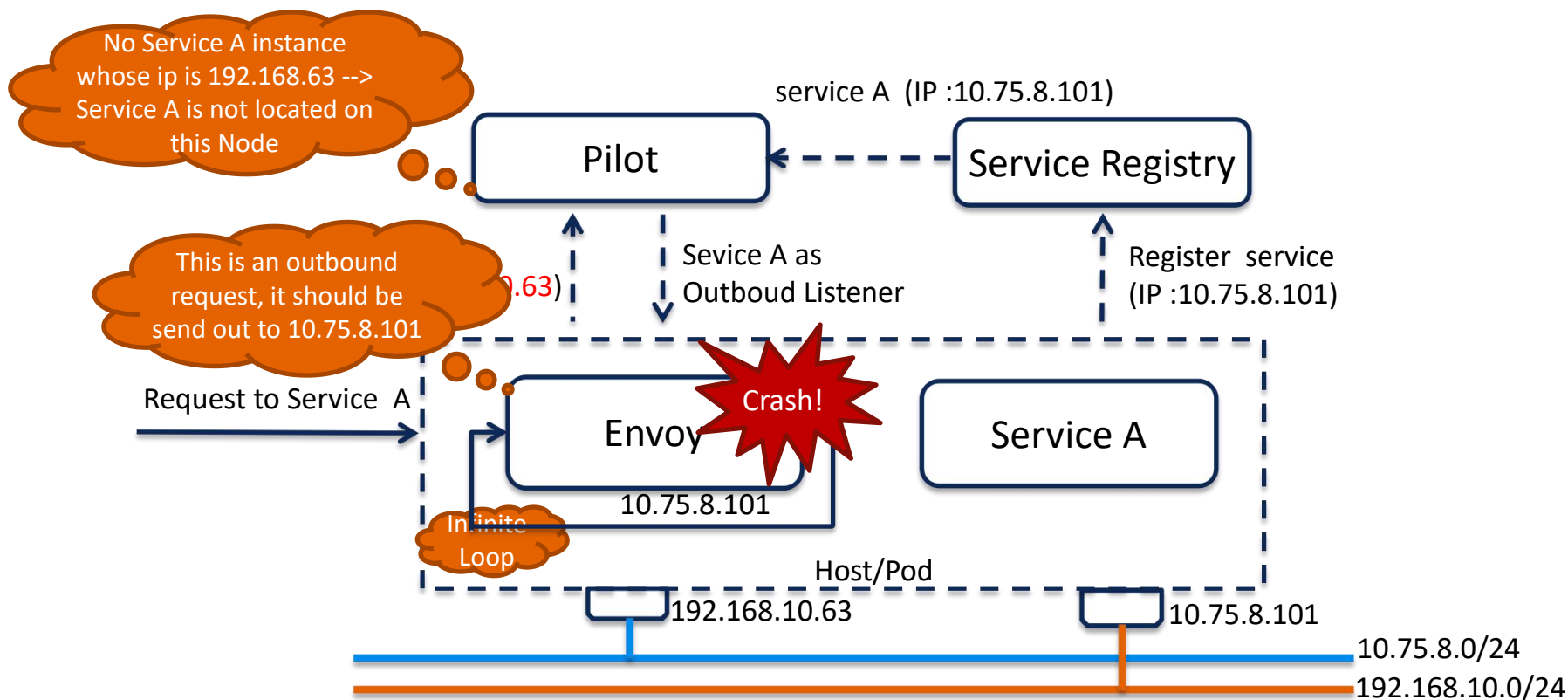
电信系统一般会有多个网络平面的，主要原因包括：避免不同功能的网络之间的**相互影响**；网络设计冗余，增强系统网络的**健壮性**；为不同的网络提供**不同的SLA**；通过网络隔离提高**安全性**；通过叠加多个网络**增加系统带宽**



上图中的Kubernetes集群使用了Knitter网络插件，部署了四个网络平面

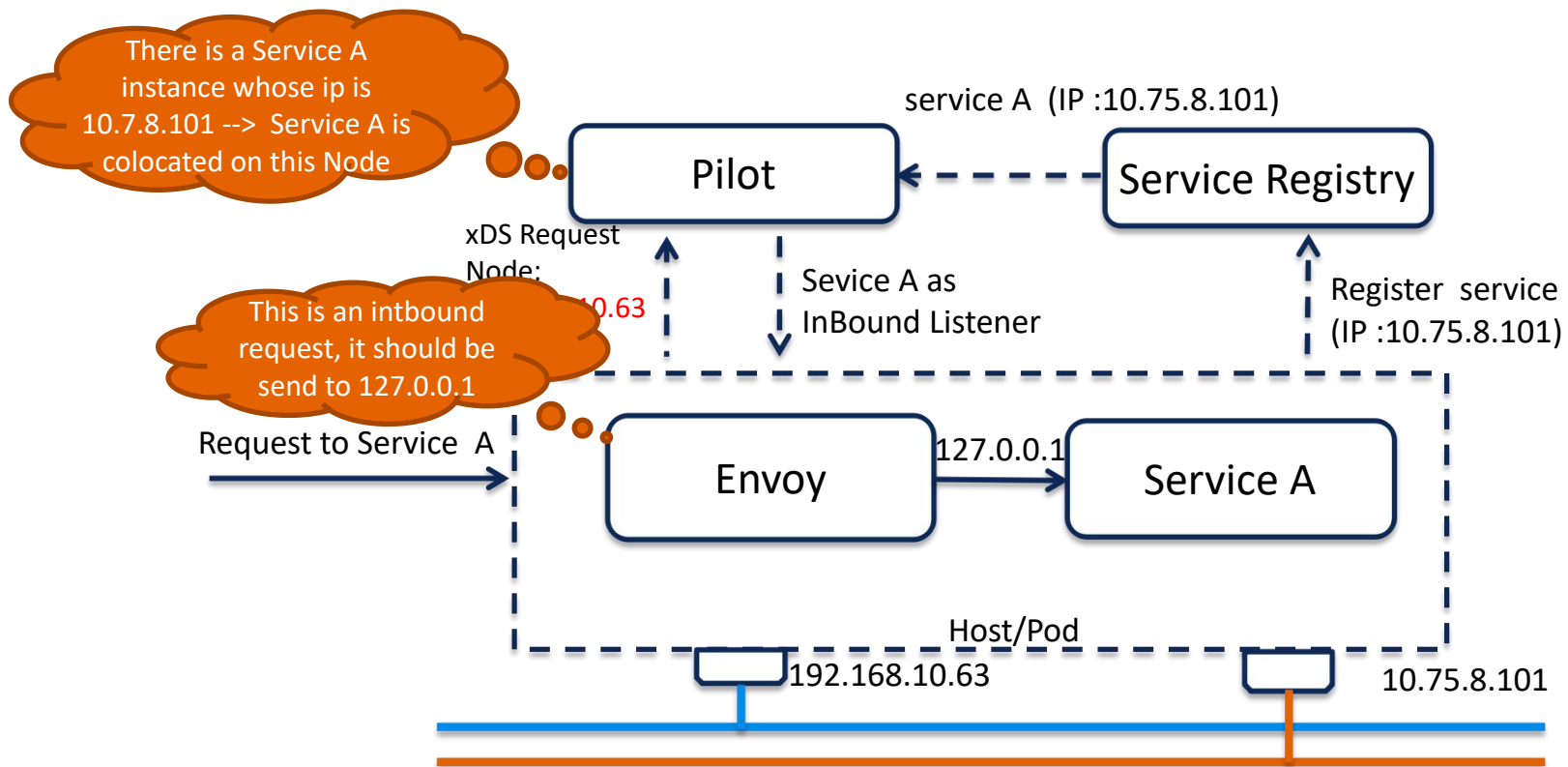
在电信级应用场景下的问题

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。



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。



<https://zhaohuabing.com/post/2018-12-19-multi-network-interfaces-for-istio/>

Service Mesh是否应该处理四层协议?

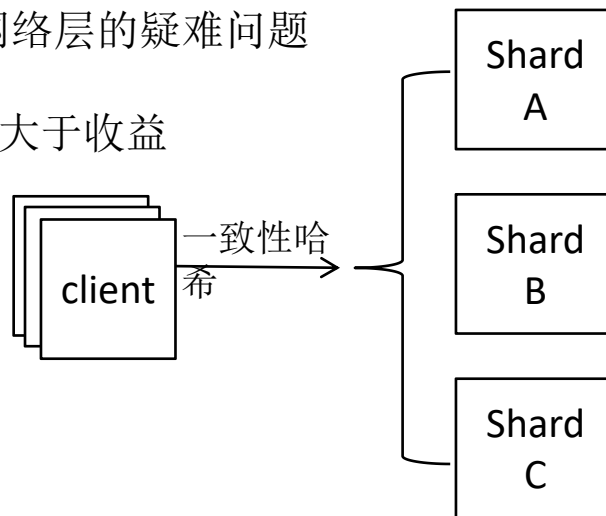
- 收益

- 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
 - 例子：Envoy的LB算法不能处理应用后端集群的Sharding
- 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
- 对四层流量的无差别拦截可能导致系统中出现较多网络层的疑难问题

- 结论：将TCP直接纳入Service Mesh管控还不成熟，成本远大于收益



建议

- Service Mesh应主要关注L7，而不是L4
- 首先考虑使用REST/HTTP作为RPC
- 如果研发资源充足，可以考虑扩展自定义的RPC协议（需对控制面和数据面进行定制和二次开发）

如何Bypass四层流量？

在Service Mesh中 Bypass TCP流量，让TCP请求跳过Service Mesh的处理，缺省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
- 最彻底，流量不经过Envoy，但需要对存量应用的进行大量改造
- 方案二：在Envoy中 bypass TCP 流量
- 流量会经过Envoy，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
```

不谋而合：Istio 1.3中对TCP服务的处理

Istio 1.3改进：提供了HTTP Inspector，并且可以支持按照协议对filterchain进行Match

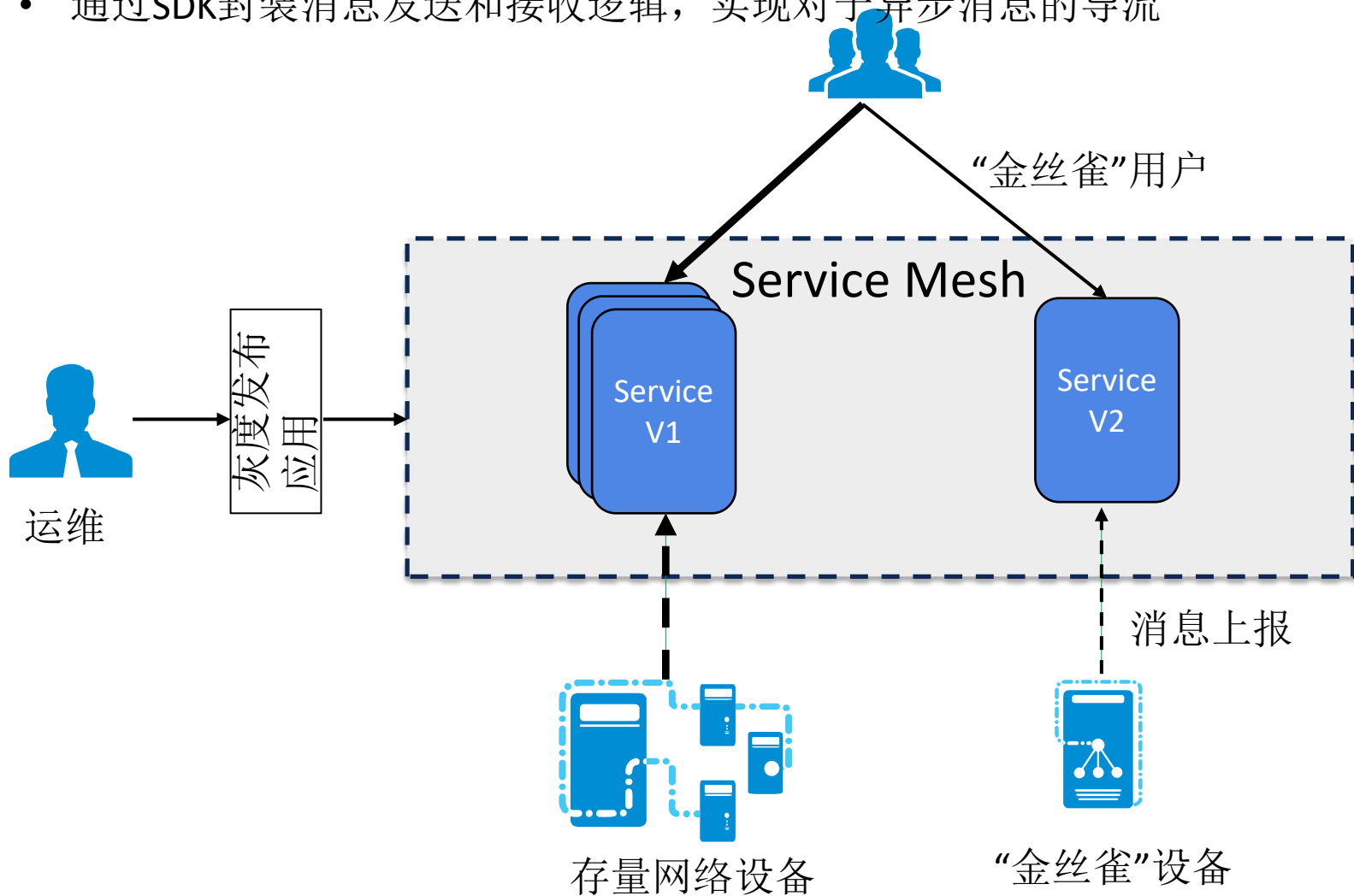
可在一个TCP请求没有对应的TCP服务，并且端口和HTTP服务没有冲突的情况下，TCP请求会被缺省发送到原始目的地。

遗留问题：

在和HTTP服务端口冲突的情况下，TCP请求将会被丢弃，导致客户端请求失败，这一点社区尚未解决。

灰度发布：异步通信的流量管理

- Istio对异步消息这种常见的微服务通信方式没有进行处理
- 通过SDK封装消息发送和接收逻辑，实现对于异步消息的导流



其他一些产品化增强


















- APP：灰度发布、流量控制，混沌测试，更多的APP待业务场景触发
- IPV6支持
- 在Istio中集成方法级的调用跟踪
- 在Istio中集成Kafka调用跟踪
- Sidecar的内存占用优化

上游开源社区参与情况

所有通用的故障修复、性能优化和新特性都提交PR合入了上游社区。包括：

- Consul Registry性能和资源占用优化
- 多网络平面支持

坚持和社区同发展，充分利用社区的力量，在产品化过程中对Istio的改进会持续向社区进行贡献。

<input type="checkbox"/>	 6 Total	Author ▾	Labels ▾	Milestones ▾	Reviews ▾	Assignee ▾	Sort ▾
<input type="checkbox"/>	 Use watching instead of polling to get update from Consul catalog ✓ cla: yes size/XL						 1
	#17881 by zhaohuabing was merged 3 days ago • Approved						
<input type="checkbox"/>	 Fix: Consul high CPU usage (#15509) ✗ area/perf and scalability						 4
	#15510 by zhaohuabing was merged on Jul 23 • Approved						
<input type="checkbox"/>	 Remove warn log message of ignored Consul service tag • area/user experience						 8
	#15452 by zhaohuabing was merged on Jul 13 • Approved						
<input type="checkbox"/>	 Avoid unnecessary service change events(#11971) ✗ ok-to-test						 4
	#12148 by zhaohuabing was merged on Mar 1 • Approved						
<input type="checkbox"/>	 Use ServiceMeta to convey the protocol and other service properties ✗						 13
	#9713 by zhaohuabing was merged on Nov 10, 2018 • Approved						
<input type="checkbox"/>	 Support multiple network interfaces(#9441) •						 70
	#9688 by zhaohuabing was merged on Dec 15, 2018 • Approved						

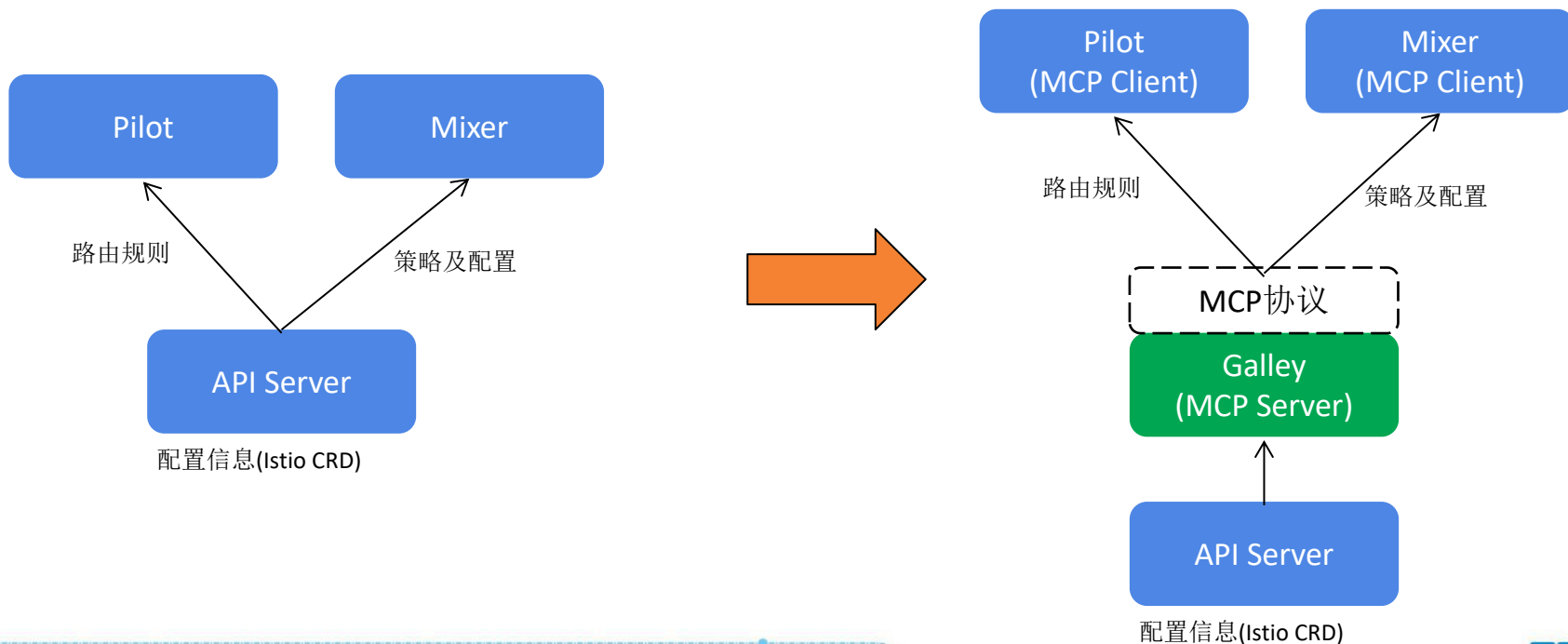
Istio项目演进趋势-网格配置

现状及问题:

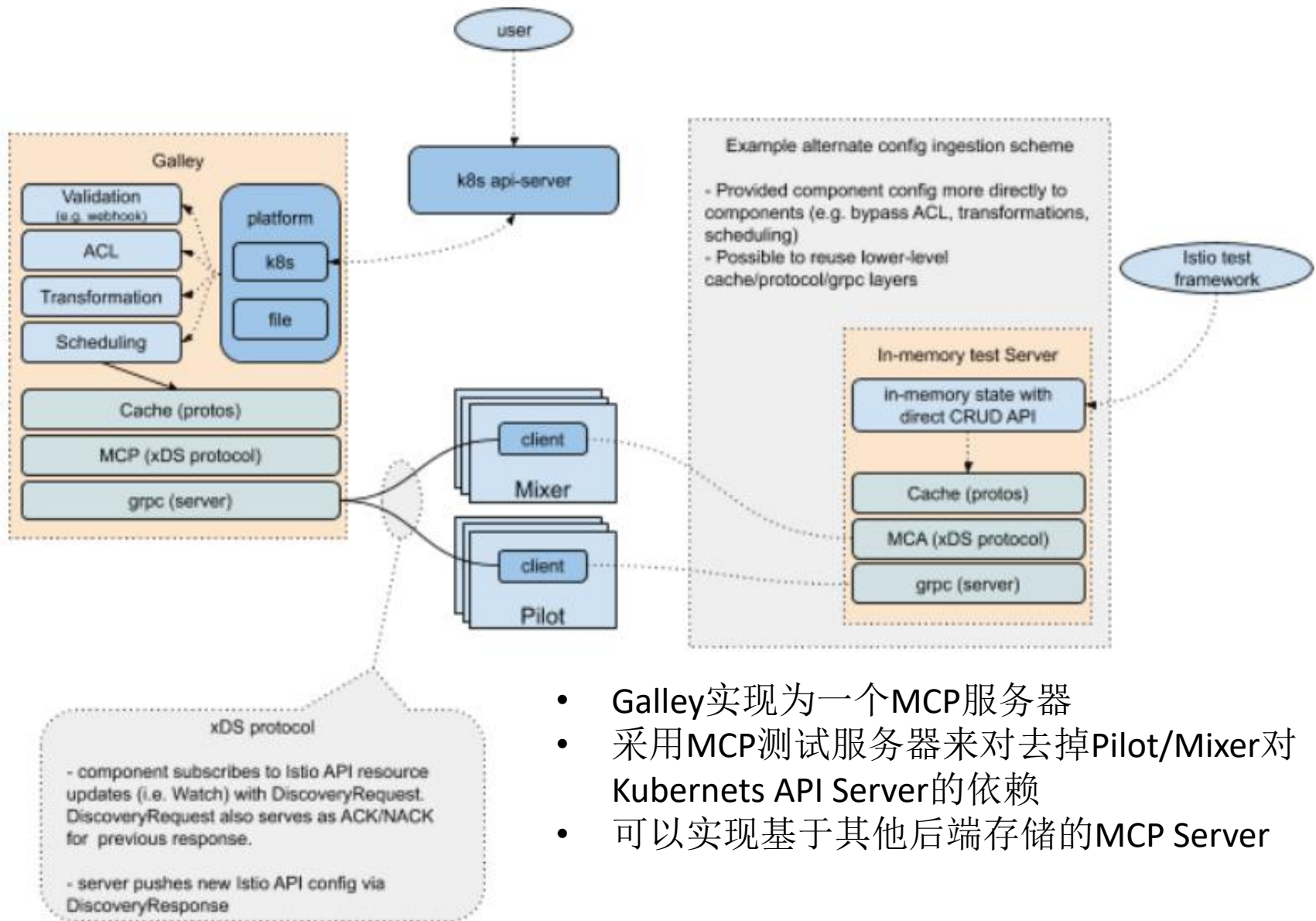
- 控制面组件从API Server直接获取配置数据（Istio CRD）
- 控制面组件和Kubernetes API Server耦合紧密，难以测试
- 缺乏统一的配置管理，各个组件各自订阅配置
- 缺乏配置数据的隔离机制，ACL控制

演进方式:

- 采用MCP（Mesh Configuration Protocol）标准接口来下发配置数据
- 采用galley来管理Kubernetes API Server中的配置数据，提供CRD的 validation，缓存，同步，ACL等配置数据的统一管理



Istio项目演进趋势-网格配置



- Galley实现为一个MCP服务器
- 采用MCP测试服务器来对去掉Pilot/Mixer对Kubernetes API Server的依赖
- 可以实现基于其他后端存储的MCP Server

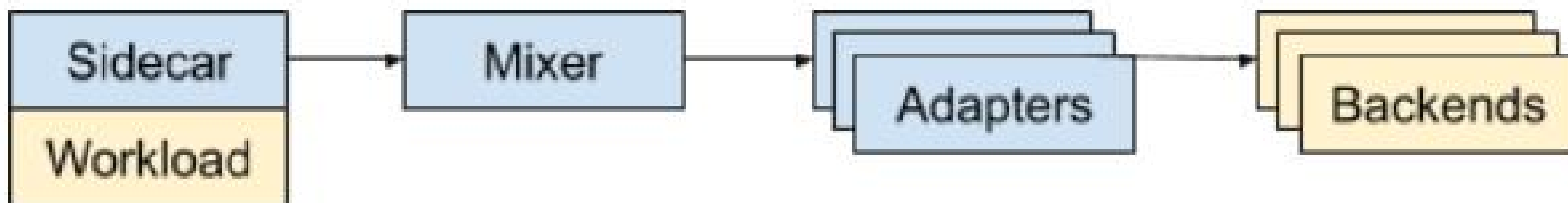
Istio项目演进趋势-Mixer的演进

Mixer V1架构： Mixer作为控制面的一个独立组件

优势：灵活的适配器架构，可以方便地接入多个不同的Backend系统

劣势：

- Envoy在业务调用链中加入了对Mixer的远程调用，带来了较大的性能开销
- 网格中的大量Telemetry和policy check都需要经过Mixer，Mixer成为系统瓶颈
- 采用一个template来抽象不同Backend系统需要的数据，需要编写大量复杂的配置文件

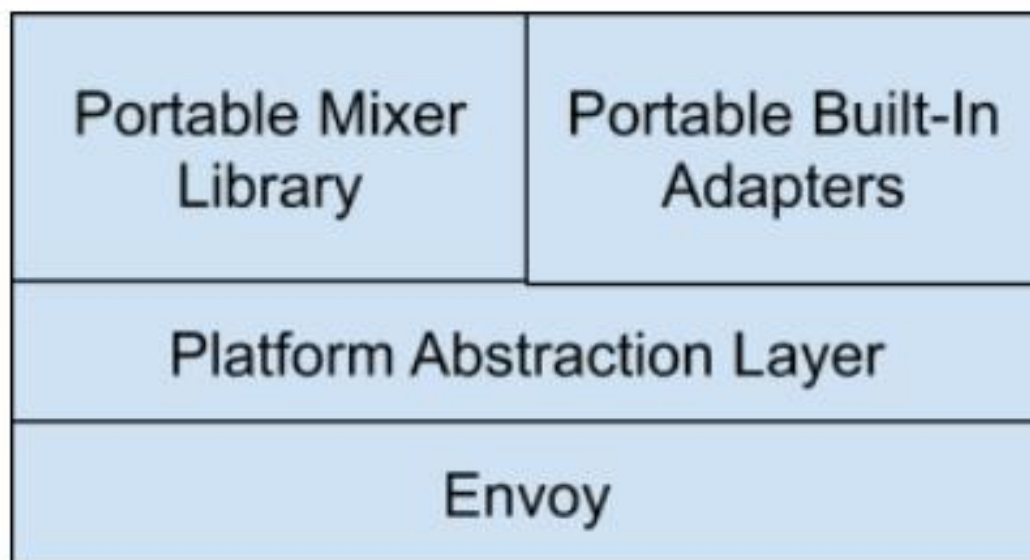


Istio项目演进趋势-Mixer V2 Proposal

Mixer V2: 将Mixer作为插件内置到Envoy中

内置Mixer架构:

- **PAL:** 为各个Adapter实现基础平台，屏蔽不同Proxy之间的差异。将来会提供在web assembly runtime上运行插件的能力。
- **Adapter:** 实现各种业务功能的插件。

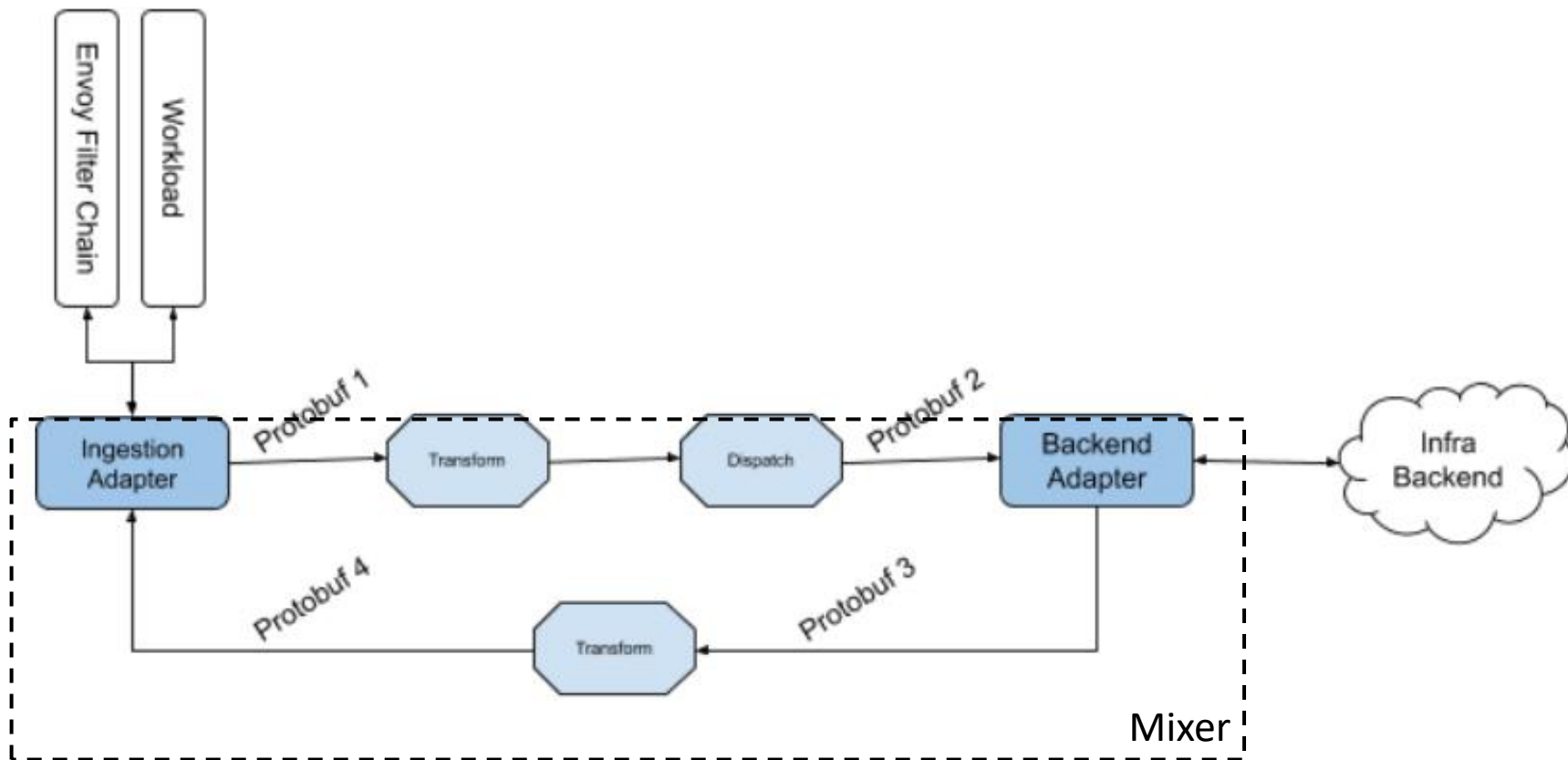


Istio项目演进趋势-Mixer V2 Proposal

Mixer V2 业务处理流程：

Mixer V2由一系列Adapter组成，Adapter分为两类

- Ingestion Adapter: 拦截请求，并将请求解析为一个Backend Adapter可以理解的数据结构
- Backend Adapter: 消费解析的数据结构，并结合后端系统实现特定功能，例如

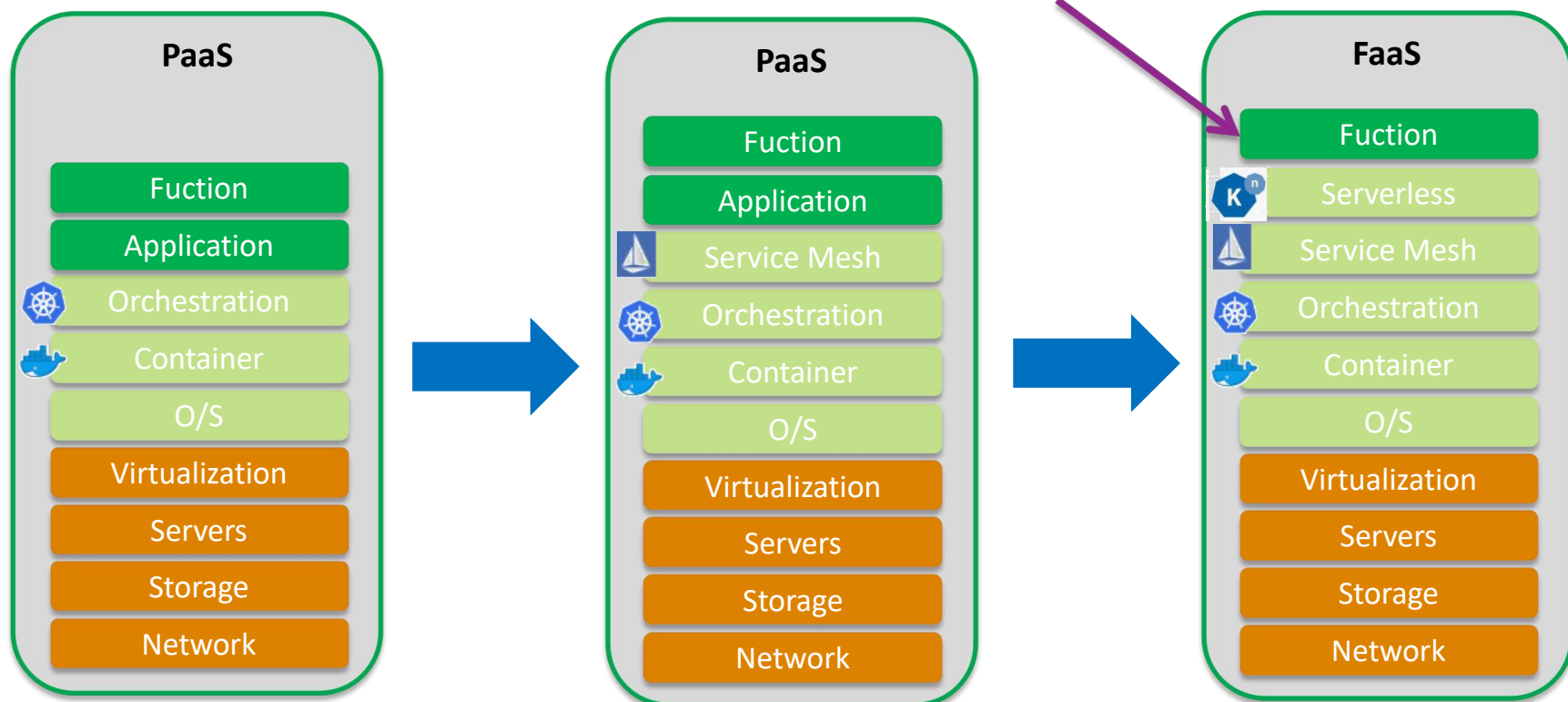


Service Mesh 发展趋势-下沉为公有云基础设施

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。

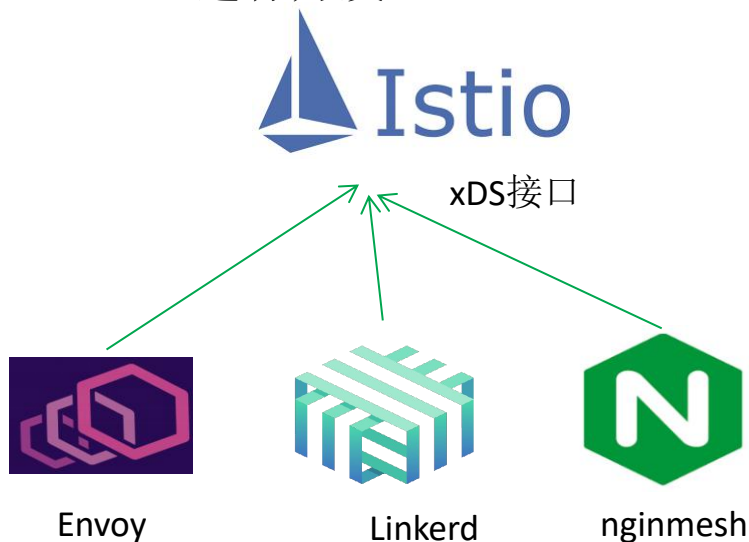
目前Google，AWS，微软，阿里，腾讯等公有云提供商都已经提供了公有云上的Service Mesh托管服务。

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:



Istio



Linkerd2



Consul



Traffic Director



Azure

Azure Service
Fabric Mesh



AWS App Mesh

数据面项目:



Envoy



Linkerd2-proxy



Consul connect



Gloo



AMBASSADOR

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

博客

Github

<https://zhaohuabing.com>

<https://medium.com/@zhaohuabing>

<https://github.com/zhaohuabing>

Any Question?



5G 先锋

