Kmesh: 内核创新为服务网格带来全新体验

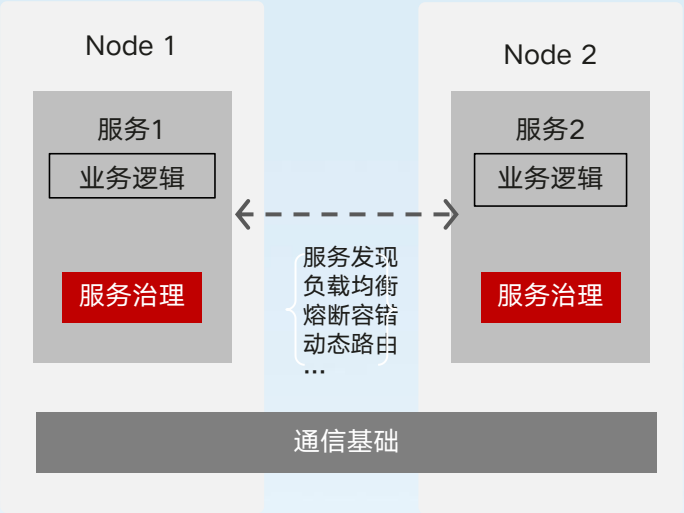
吴长冶 华为

Kmesh: 内核创新为服务网格带来全新体验

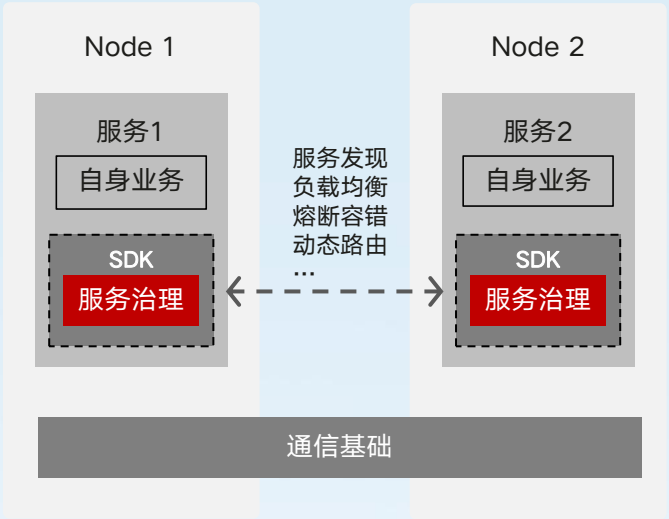
吴长冶 华为

服务治理演进：逐步从业务中解耦，下沉到基础设施

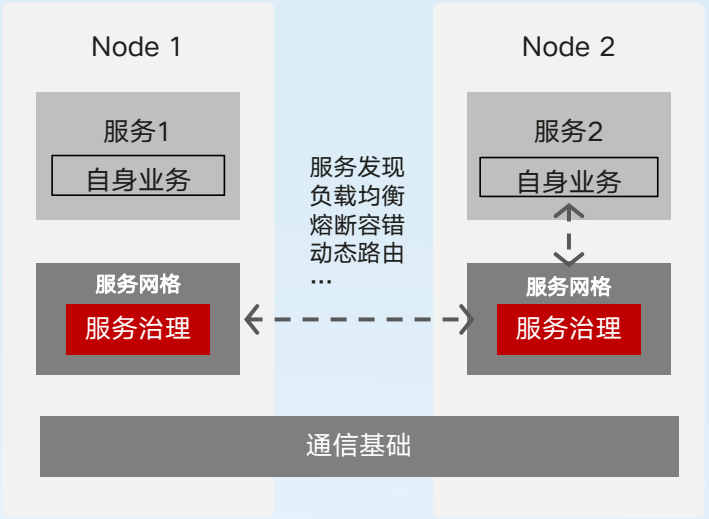
第一代：服务治理能力内嵌在业务代码中
典型技术：SOA、ESB



第二代：服务治理能力抽象到统一SDK实现
典型技术：Spring Cloud、Dubbo



第三代：服务治理能力归一到服务网格



优势

- 简单使用依赖少

- 代码重复少
- 治理逻辑代码和业务代码分开

- 独立进程，用户业务非侵入、语言无关
- 治理逻辑升级业务无感知
- 可以渐进的微服务化

劣势

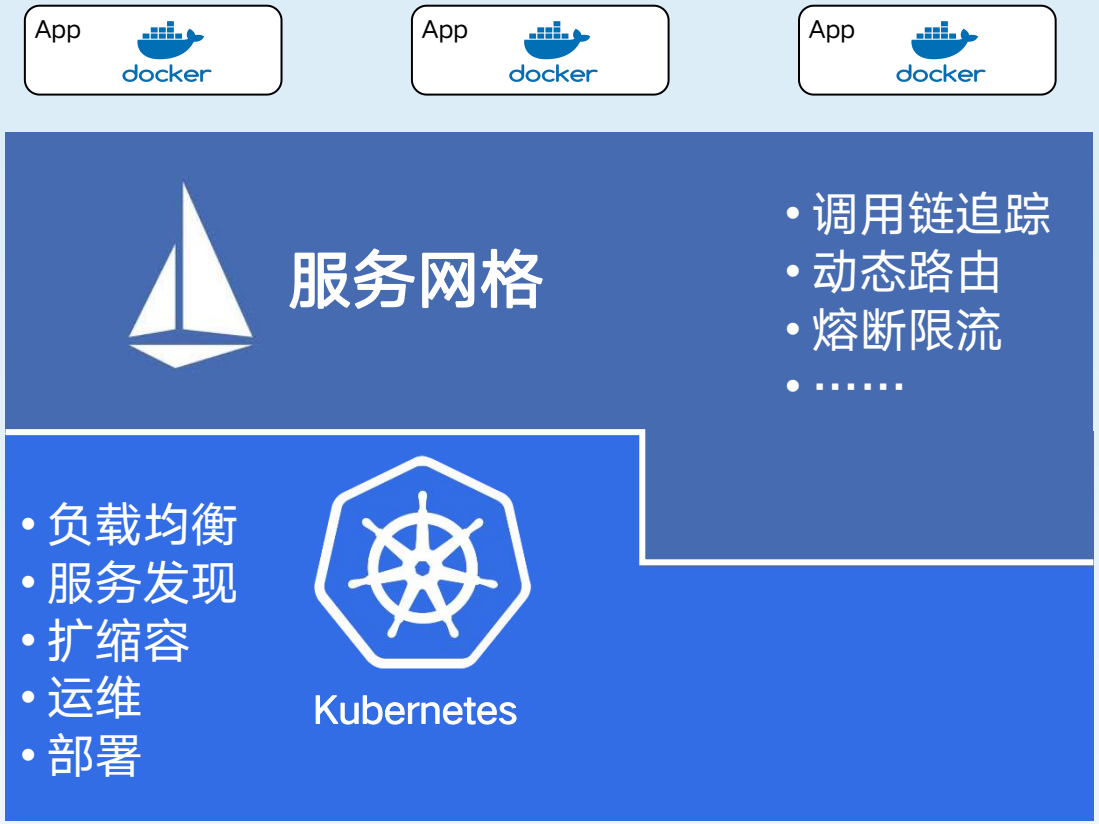
- 代码耦合
- 代码重复高
- 运维复杂
- 解耦差，开发要求高

- SDK语言绑定、代码侵入
- 基于SDK开发学习门槛高
- 在用系统改造代价大
- 治理能力升级影响用户业务

- 代理的性能和资源开销

服务网格作为云原生下一代技术，已成为云上基础设施标配

serviceMesh是处理服务与服务之间通信的基础设施层，弥补了Kubernetes在微服务的连接、管理和监控方面的短板，为Kubernetes提供更好的应用和服务管理。



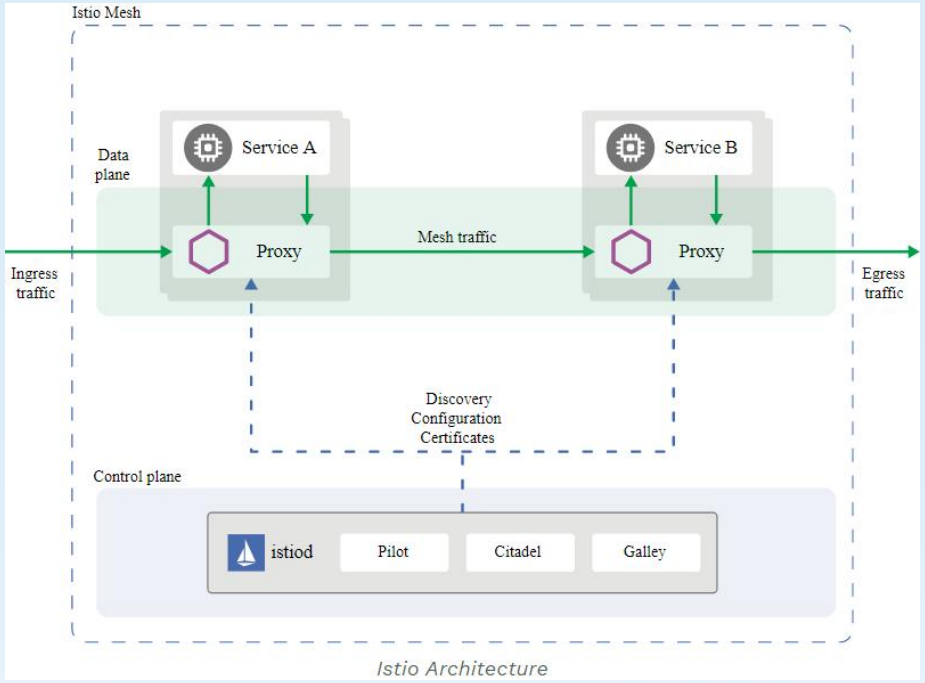
核心理念：

1. **非侵入式Sidecar注入技术**，将数据面组件注入到应用所在的容器，通过劫持应用流量来进行功能实现，应用无感知。
2. **北向API基于K8s CRD实现**，完全声明式，标准化。
3. **数据面与控制面通过xDS gRPC标准化协议**通信，支持订阅模式。

核心特性：

1. **服务&流量治理：**熔断，故障注入，丰富的负载均衡算法，限流，健康检查，灰度发布，蓝绿部署等
2. **流量与访问可视化：**提供应用级别的监控，分布式调用链，访问日志等
3. **安全连接：**通过mTLS、认证、鉴权等安全措施帮助企业在零信任的网络中运行应用

Istio为代表的服务网格已逐步流行，但仍面临一定的挑战



Istio Architecture

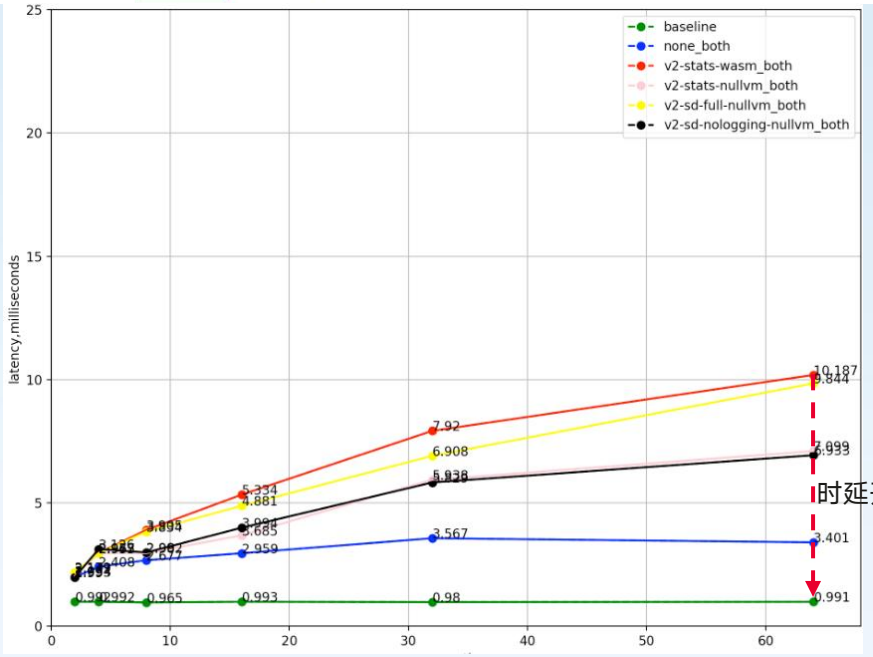
网格数据面存在的挑战：

- **代理层引入额外时延开销**：服务访问多条链路，无法满足时延敏感应用诉求
- **资源占用大**：代理节点占用额外CPU/MEM开销，业务容器部署密度低[1]

以Istio为例，服务网格基础设施带来的性能开销：

The Istio load tests mesh consists of 1000 services and 2000 sidecars with 70,000 mesh-wide requests per second. After running the tests using Istio 1.15, we get the following results:

- The Envoy proxy uses 0.35 vCPU and 40 MB memory per 1000 requests per second going through the proxy.
- Istiod uses 1 vCPU and 1.5 GB of memory.
- The Envoy proxy adds 2.65 ms to the 90th percentile latency.



Istio 1.15 1000 rps/s时延测试

[1]实测数据，在一个有325个cluster和175个Listener的服务网格中，一个Envoy的实际内存占用量达到了100M左右；网格中一共有466个实例，则所有Envoy占用的内存达到了466*100M=46.6G，每个envoy默认2 core，共计 466 * 2 core = 932 core；

业界探索：网格数据面软件百花齐放，多种技术路线并存

业界对现有网格数据面时延底噪的问题已有共识，为解决该问题，发展出了多种技术路线；

路线1：cilium mesh

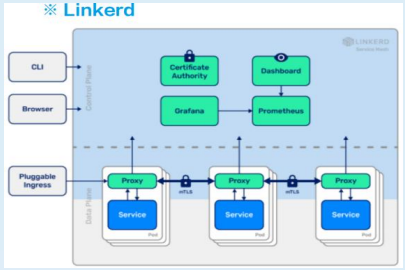
- ebpf + envoy实现高性能sidecarless网格数据面

观点：

- L7治理能力通过集成envoy实现，本质上是per-node模式，需要解决治理的故障隔离问题；
- cilium融合了cni、mesh等完整功能，略显厚重；

路线2：linker2-proxy

- 基于rust实现超轻量级微代理，主打低时延底噪、安全

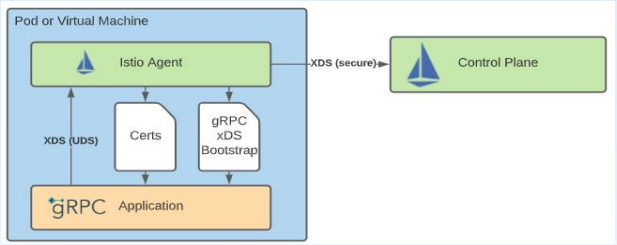


观点：

- 缺乏断路器/故障注入等高级治理能力
- 时延底噪问题仍然存在

路线3：gRPC Proxyless service Mesh

- gRPC原生支持xds协议，实现网格能力

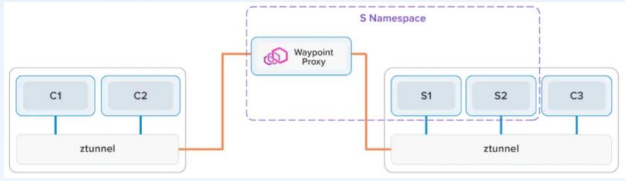


观点：

- 架构上退回到耦合模式，选择该网格方案就需要面对业务耦合、接口绑定、升级耦合、故障半径扩大等问题

istio新模式：ambient mesh

- ztunnel + waypoint proxy，按需部署网格L7治理能力

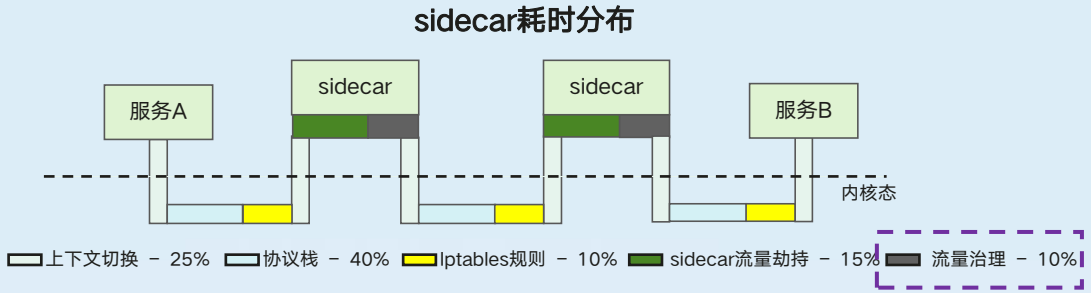


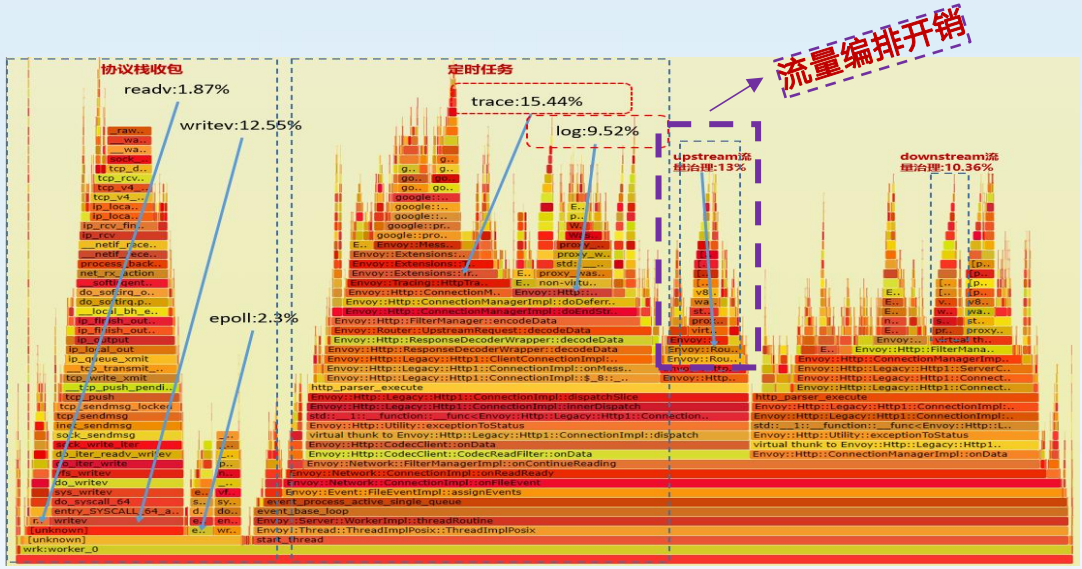
观点：

- 更轻量，L7治理场景下时延底噪反而增加，且同时支持sidecar模式，使得控制面/运维变得更复杂

性能分析 & 我们的思考

性能分析：





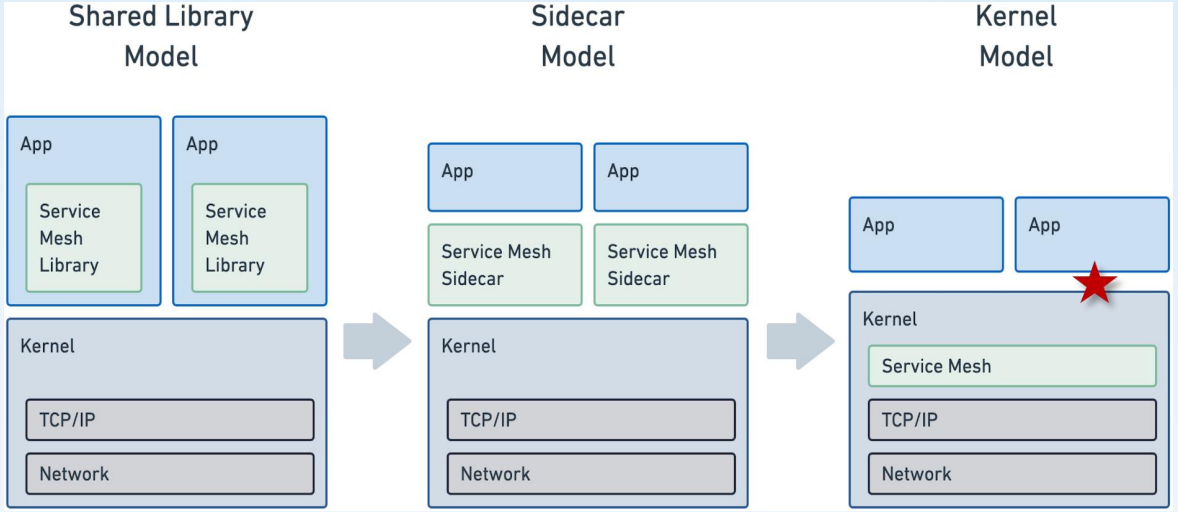
网格耗时分布可以看出：**sidecar架构引入大量时延开销，流量编排只占网格开销的10%**，大部分开销在数据拷贝、多出两次的建链通信、上下文切换调度等。

我们的思考：

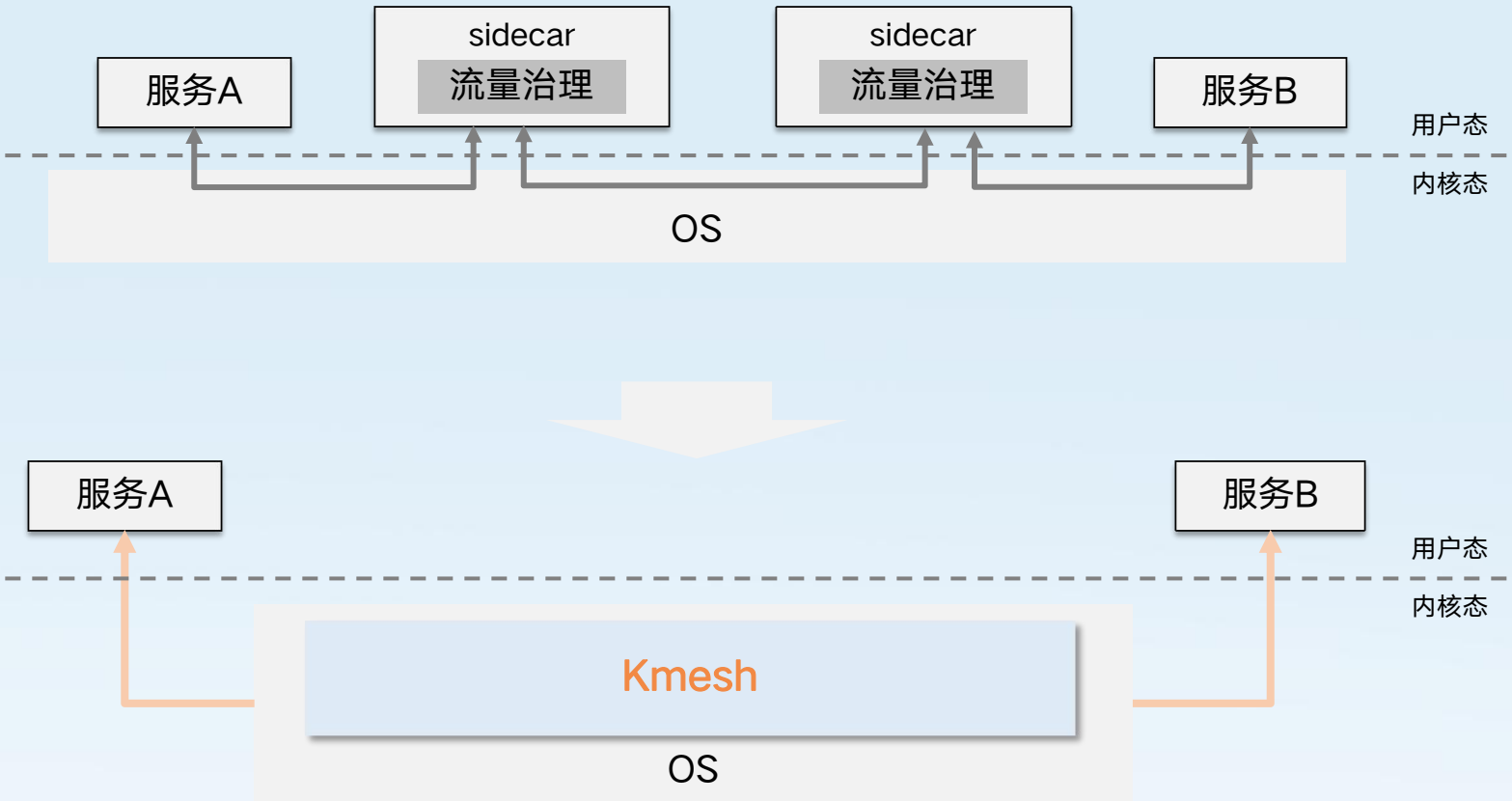
目标：回归云原生需求本源，实现应用透明、高效、低底噪的服务网格基础设施，提供业界性能最优网格数据面。

挑战：

能否实现sidecarless的服务网格，网格底噪零开销？

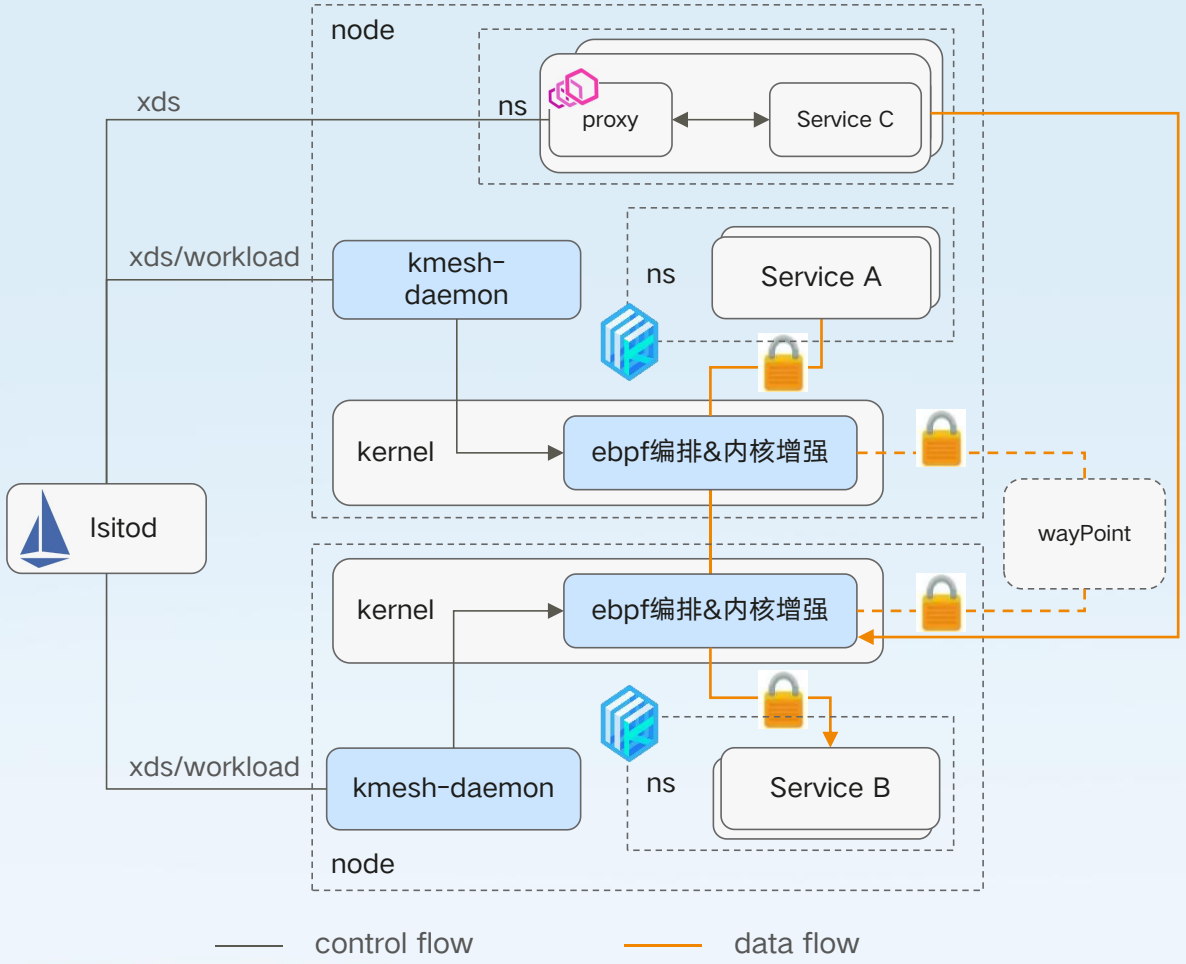


Kmesh：流量治理下沉OS，构建sidecarless服务网格



✓ 无上下文切换　✓ 无数据拷贝　✓ 无代理通信

Kmesh：内核级流量治理引擎



平滑兼容
- 应用无感的流量治理
- 自动对接Istio等软件

高性能
- 网格转发时延60%↓
- 服务启动性能40%↑

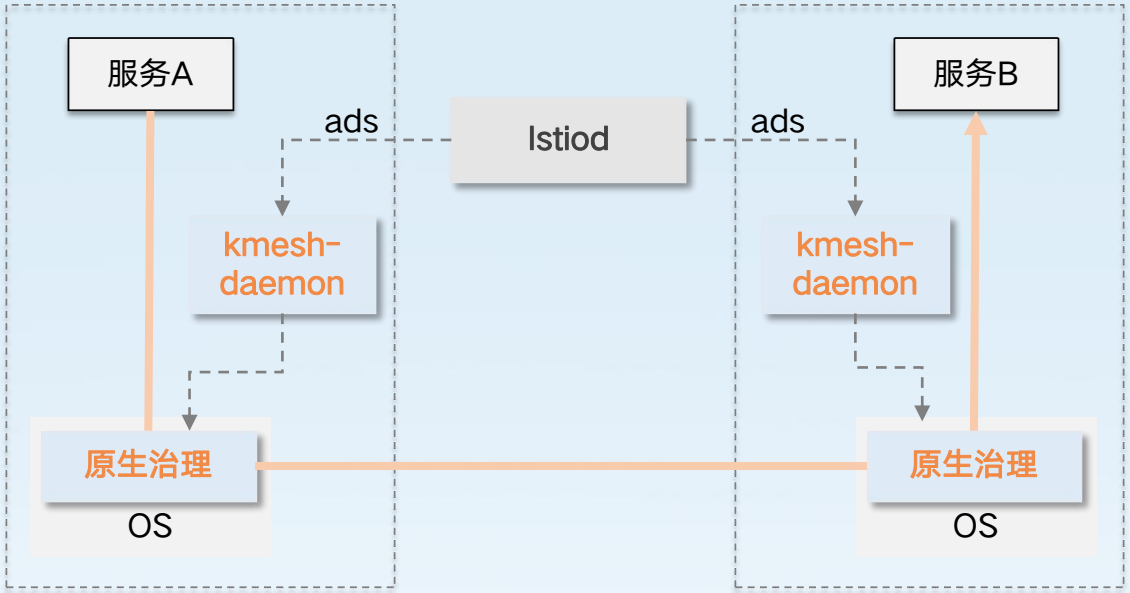
低开销
- 网格底座开销70%↓

安全隔离
- ebpf安全
- 端到端透明加密

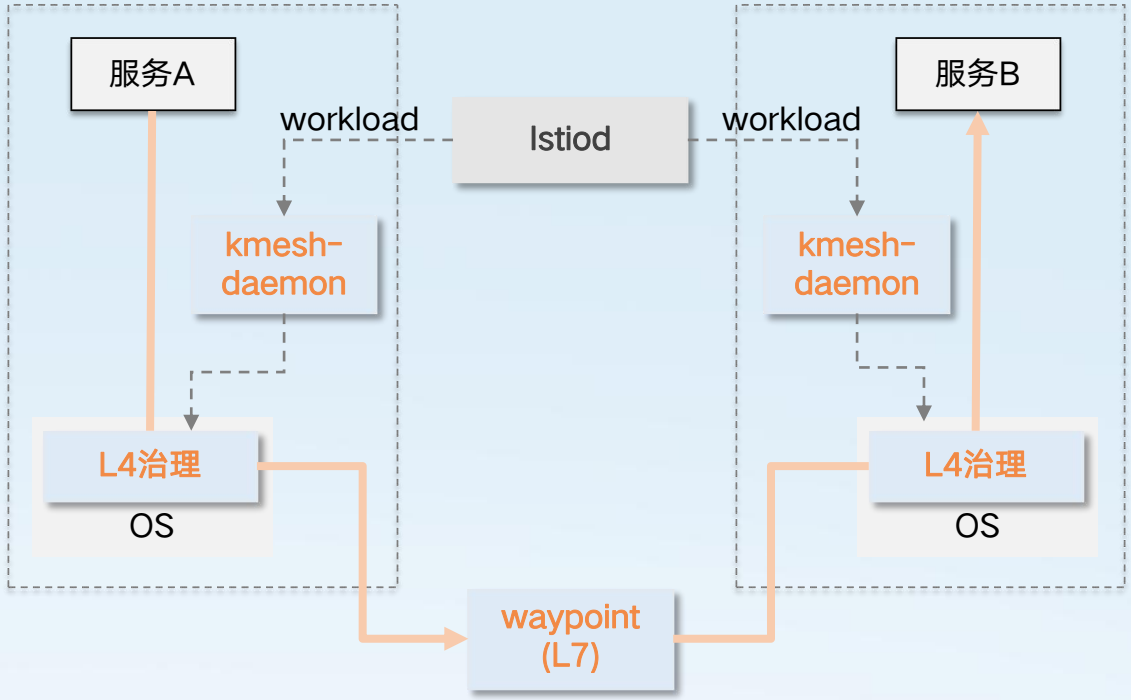
全栈可视化
- 端到端指标采集*
- 主流观测平台对接*

开放生态
- 支持XDS协议标准

Kmesh：两种模式

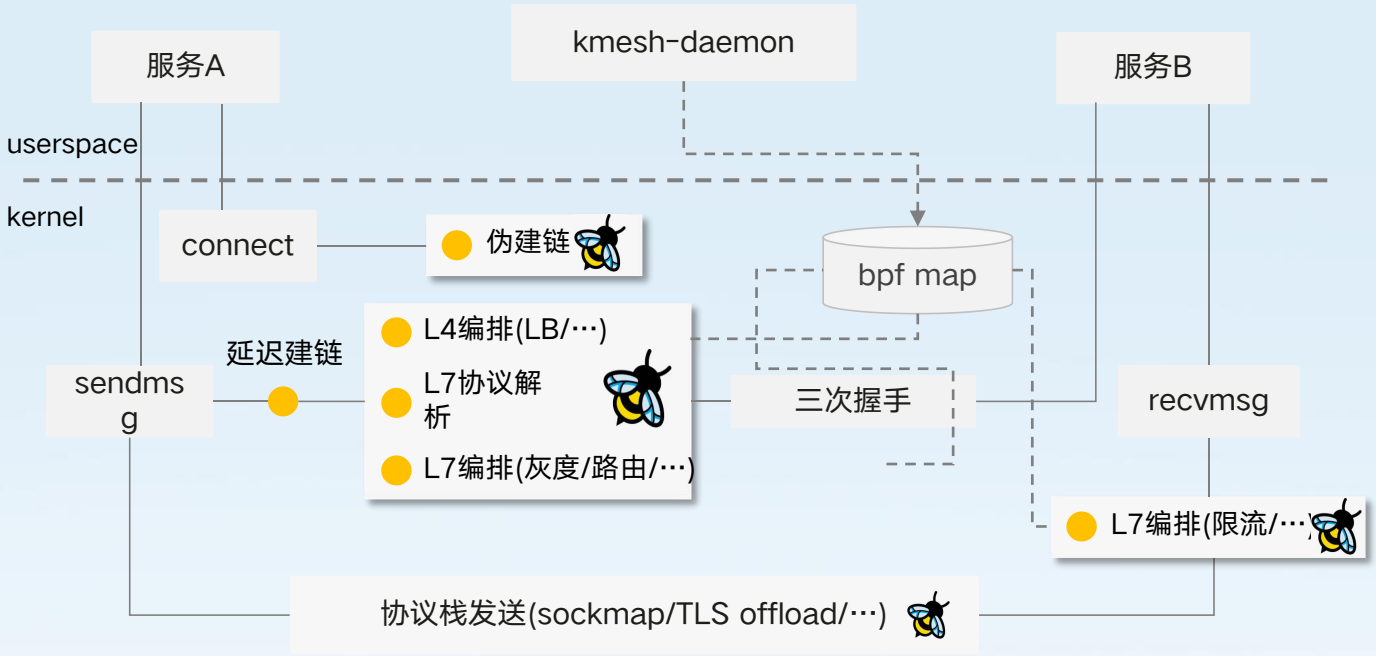


原生治理模式
L4~L7治理下沉 | 极致性能



L7拉远模式
四七层治理分离 | 灵活部署

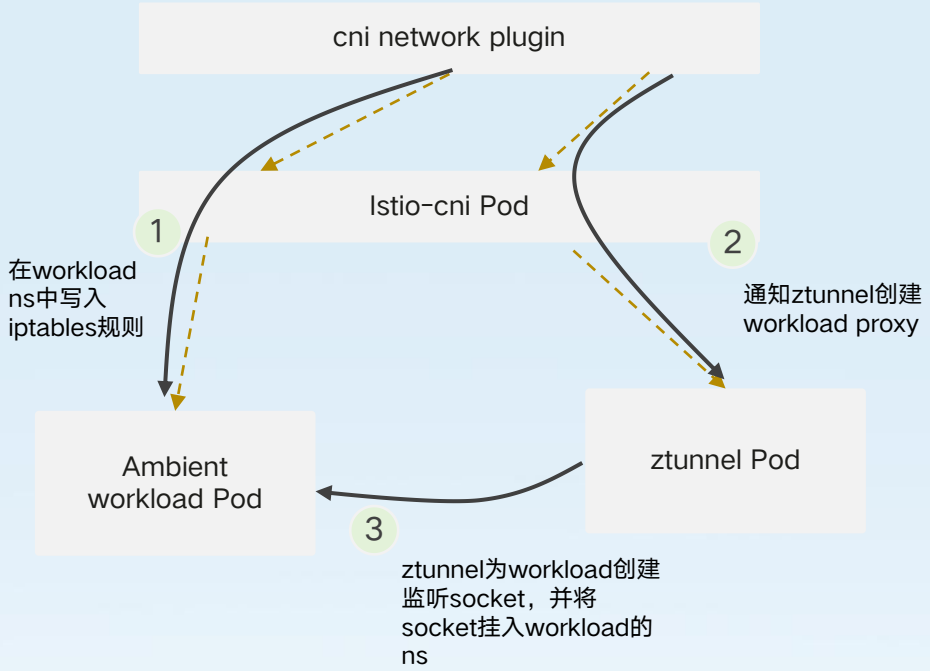
原生治理模式：流量编排运行时



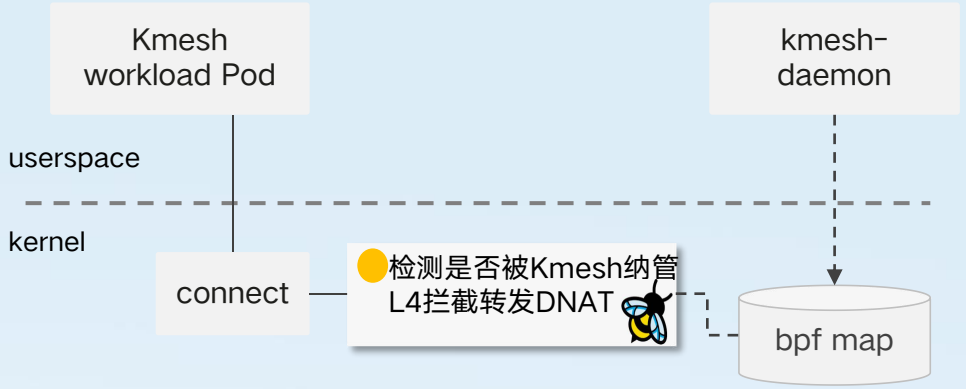
流量编排运行时：

- 基于伪建链、延迟建链等技术，内核中实现L4~L7的编排底座；
- 基于ebpf，在内核协议栈中构筑可编程的全栈流量编排运行时；

L7拉远模式：L4拦截转发

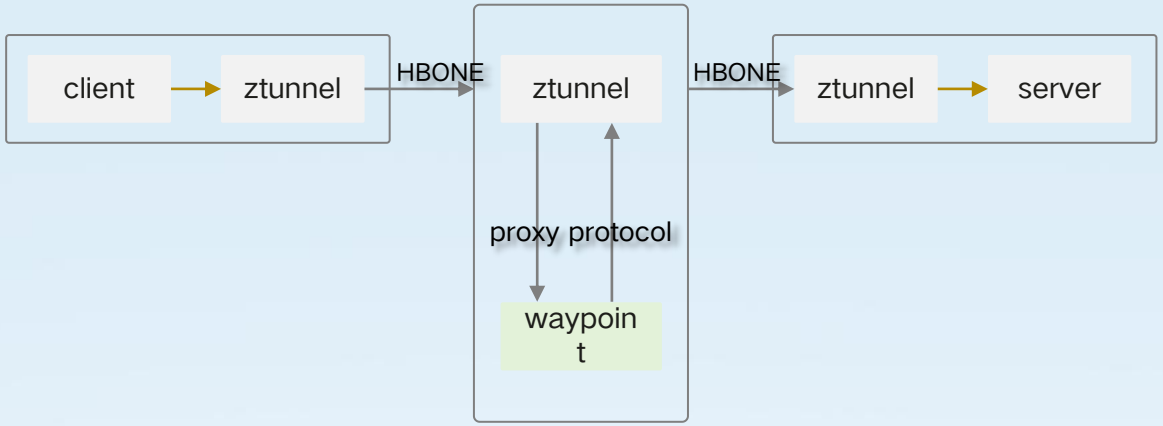


资源隔离 | 兼容性 | 升级？

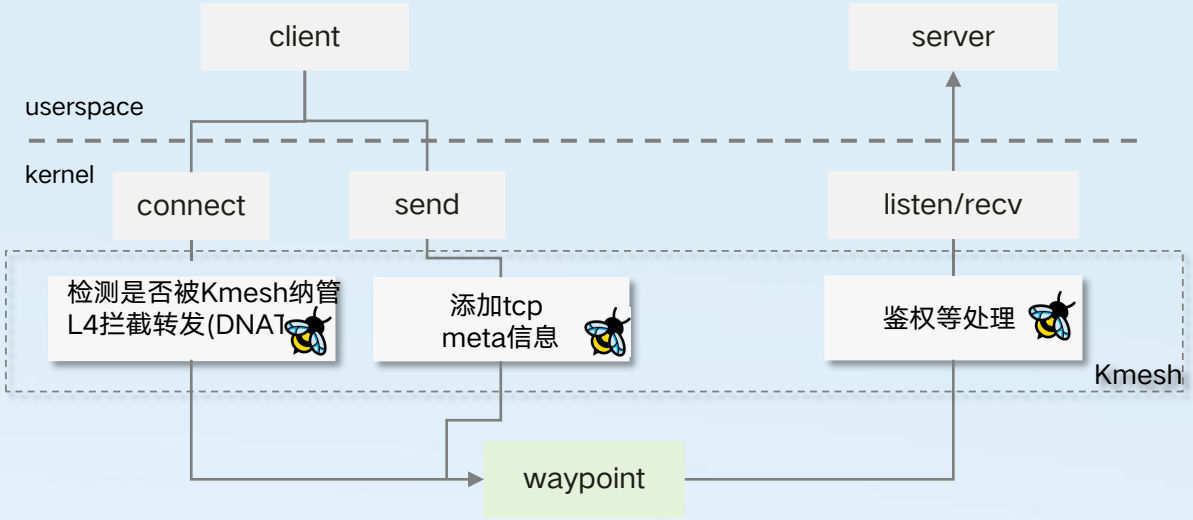


天然隔离 | 更轻量

L7拉远模式：L7 waypoint对接

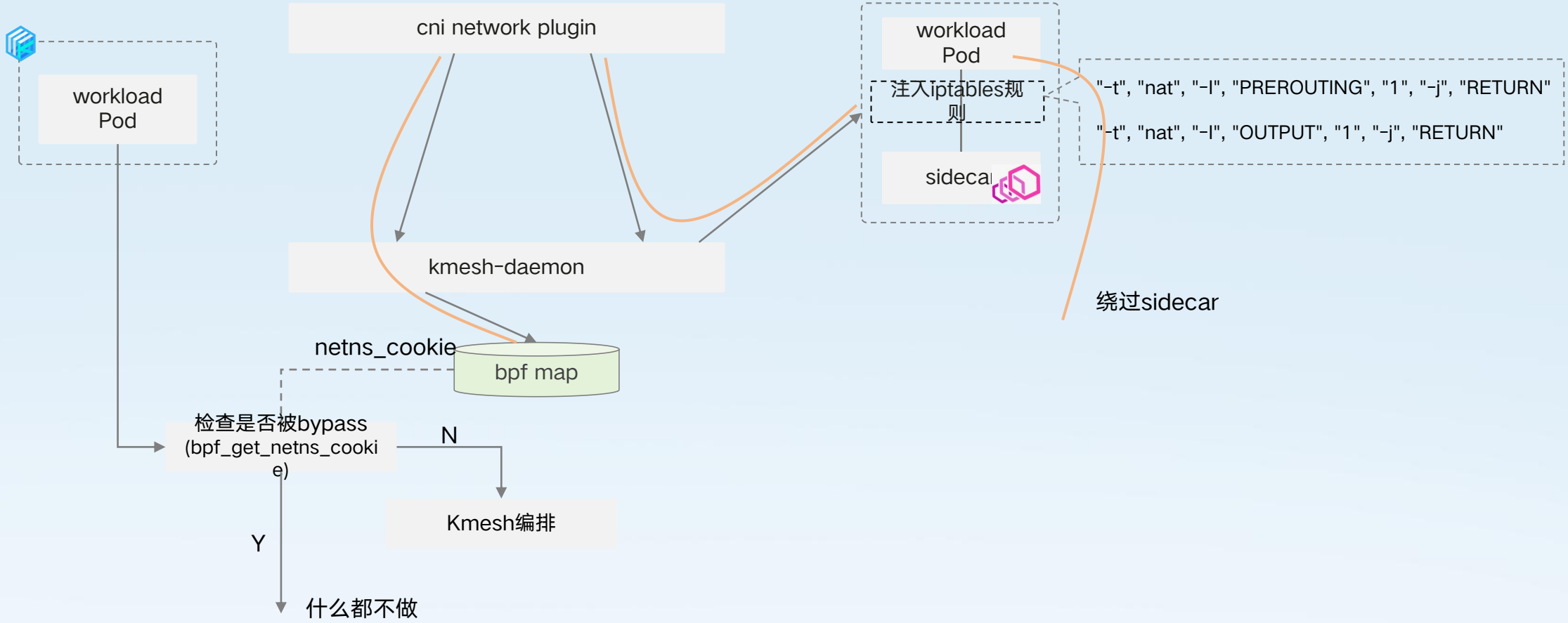


三明治模式
引入proxy protocol简化waypoint复杂度



随流转发 | 路径更短

ByPass网格数据面



demo演示

- helm安装部署
- workload性能测试
- waypoint对接

社区路标

- 24.04：v0.3
  - helm部署
  - waypoint对接
  - Bypass网格数据面
- 24.06
  - mTLS透明传输
  - 可观测（Telemetry）
  - 一致性hash算法
- 24.09
  - 熔断限流
  - 治理可扩展
- 24.12
  - 拓扑感知的LB
  - 可观测(Prometheus等)

欢迎关注Kmesh社区

https://kmesh.net

https://github.com/kmesh-net/kmesh

Wechat

KMESH

Thanks.