BUOYANT

The creators of LINKERD

Life Without Sidecars

Is eBPF's Promise Too Good to Be True?

@BuoyantIO

buoyant.io

Zahari Dichev

Software engineer, Buoyant

@zaharidichev

@zaharidichev

@Zahari Dichev

BUOYANT

LINKERD

Agenda

- What is Linkerd and why you need it
- The three pillars of a modern service mesh
- The basics of eBPF and how it relates to networking
- Can eBPF replace the proxy
- Sidecar proxies - should we get rid of them
- Q & A

What is Linkerd ?

Linkerd

- Simple and easy to use service mesh
- Uses a purpose-built micro proxy
- CNCF graduated project
- Thriving open source community
- Uses the sidecar model

Observability

- Golden metrics for HTTP and gRPC traffic
- TCP level metrics
- Topology graphing for insights into the runtime communication characteristics of your services
- Request sampling - on demand inspection of requests

Security

- Automatic mTLS for all meshed traffic without code changes
- Authorisation policies for fine grained traffic control at L7
- HTTP access logging audit
- Per pod isolation for security and operability reasons

Reliability

- Advanced retries and timeouts
- Traffic splitting and automatic canary deployments
- Multi-cluster connectivity and automated failover
- Intelligent latency aware load - balancing

High level architecture

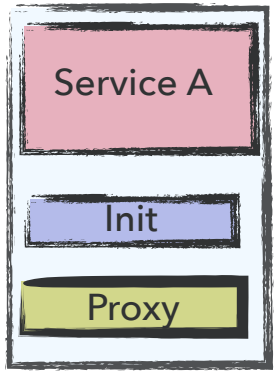
Control plane

- TLS certificates for the proxy
- Service Discovery
- Service profiles
- Dashboard + Metrics
- Identity aware policy
- API interface for CLI commands

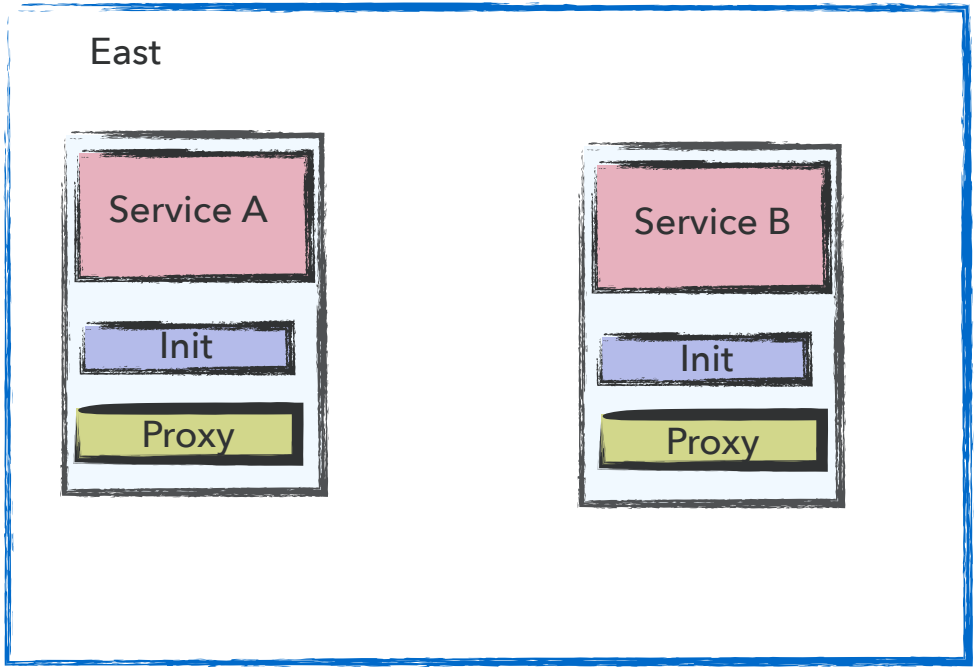
Proxy (Data plane)

- Ultralight transparent proxy written in Rust
- Automatic Prometheus metrics export for HTTP and TCP traffic
- Latency-aware load balancing
- Automatic mTLS
- On-demand diagnostic tap API

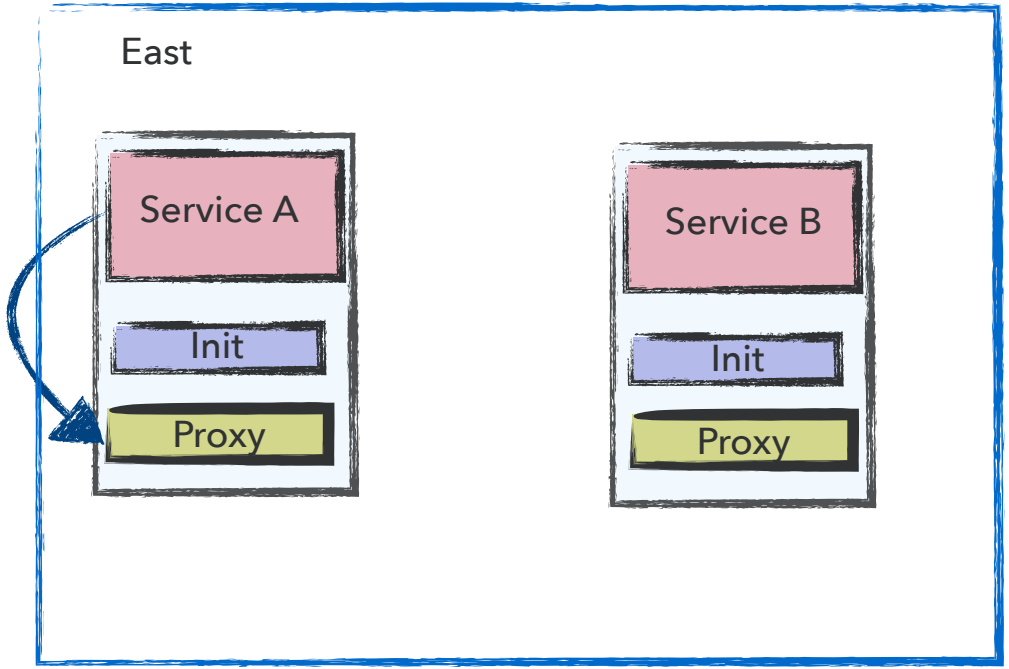
Request path



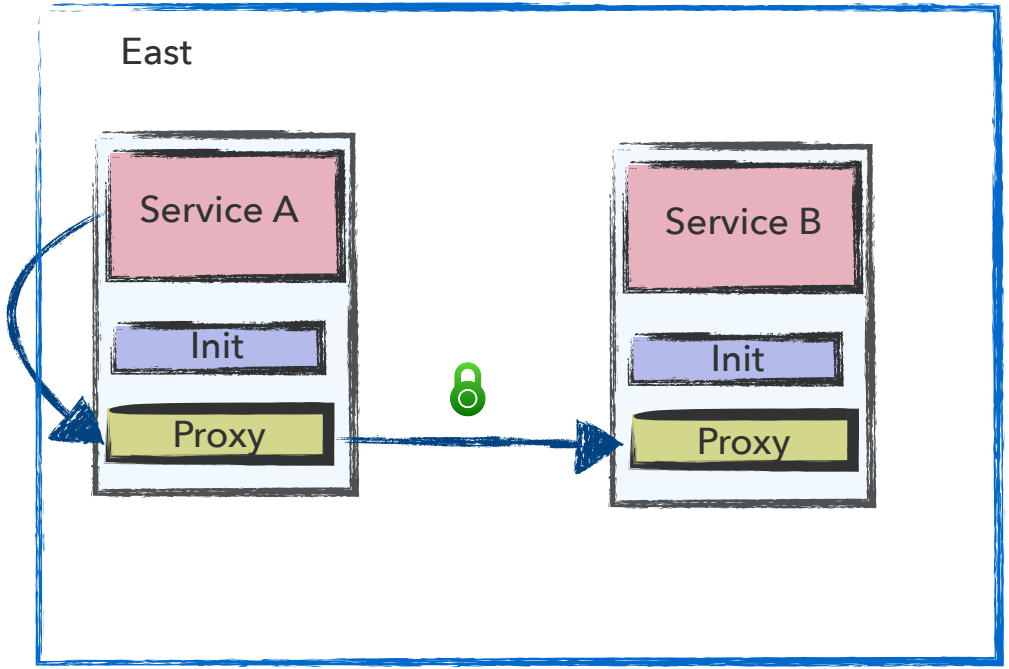
Request path



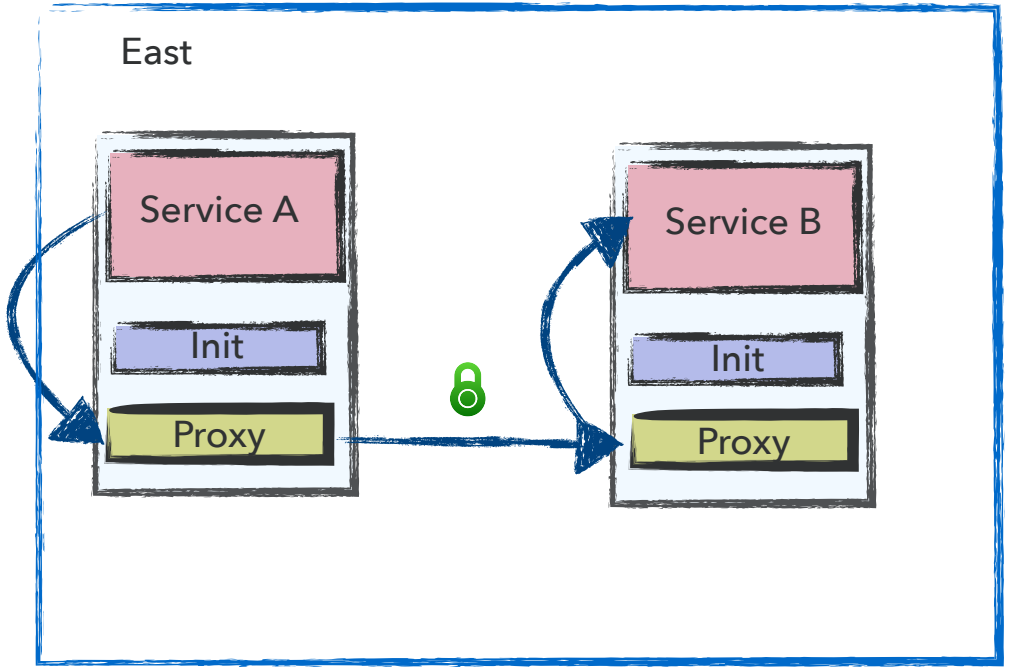
Request path



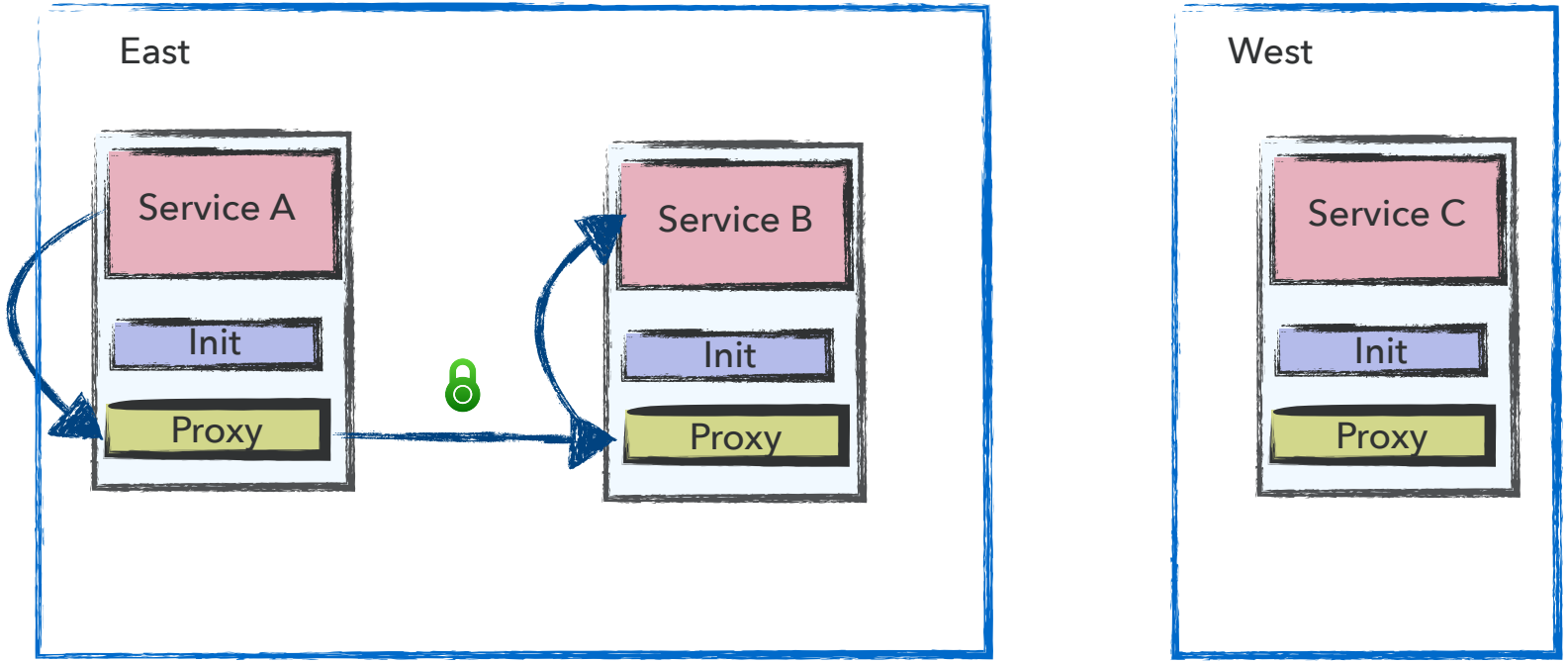
Request path



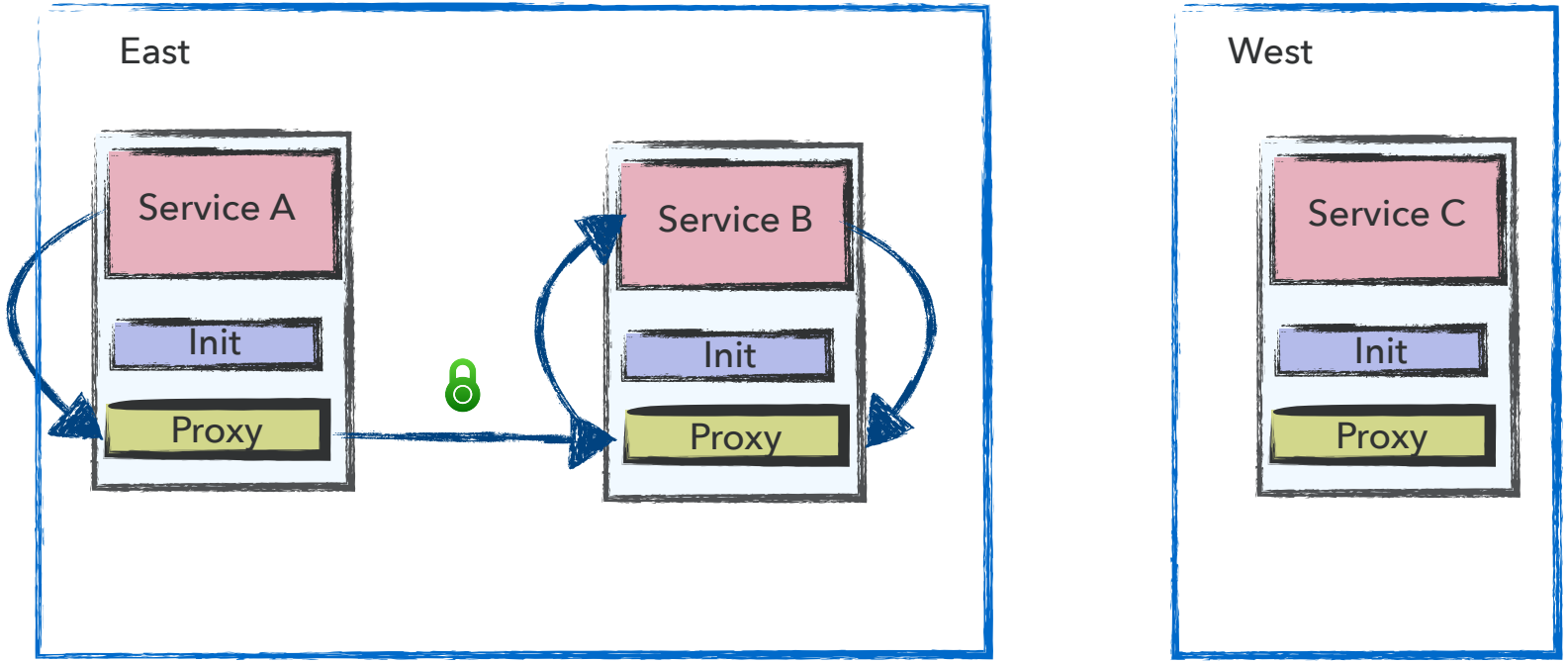
Request path



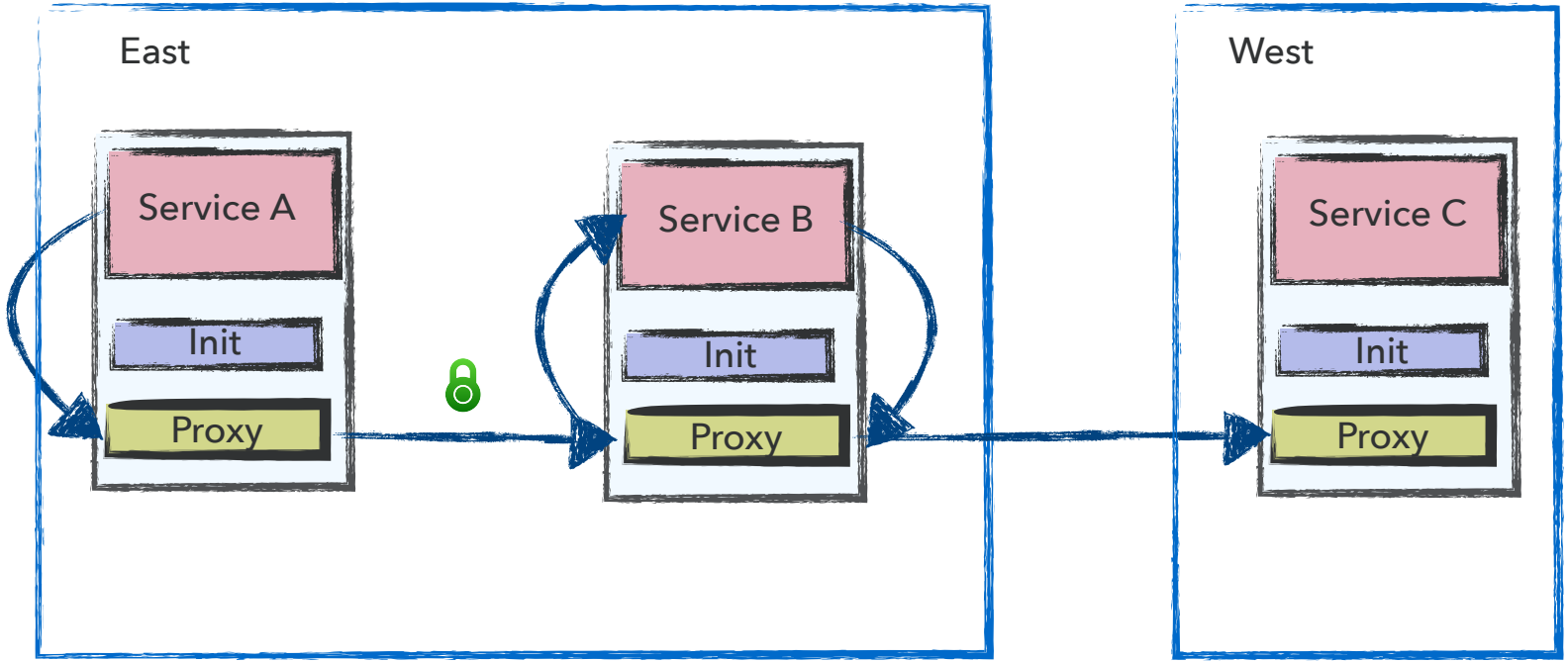
Request path



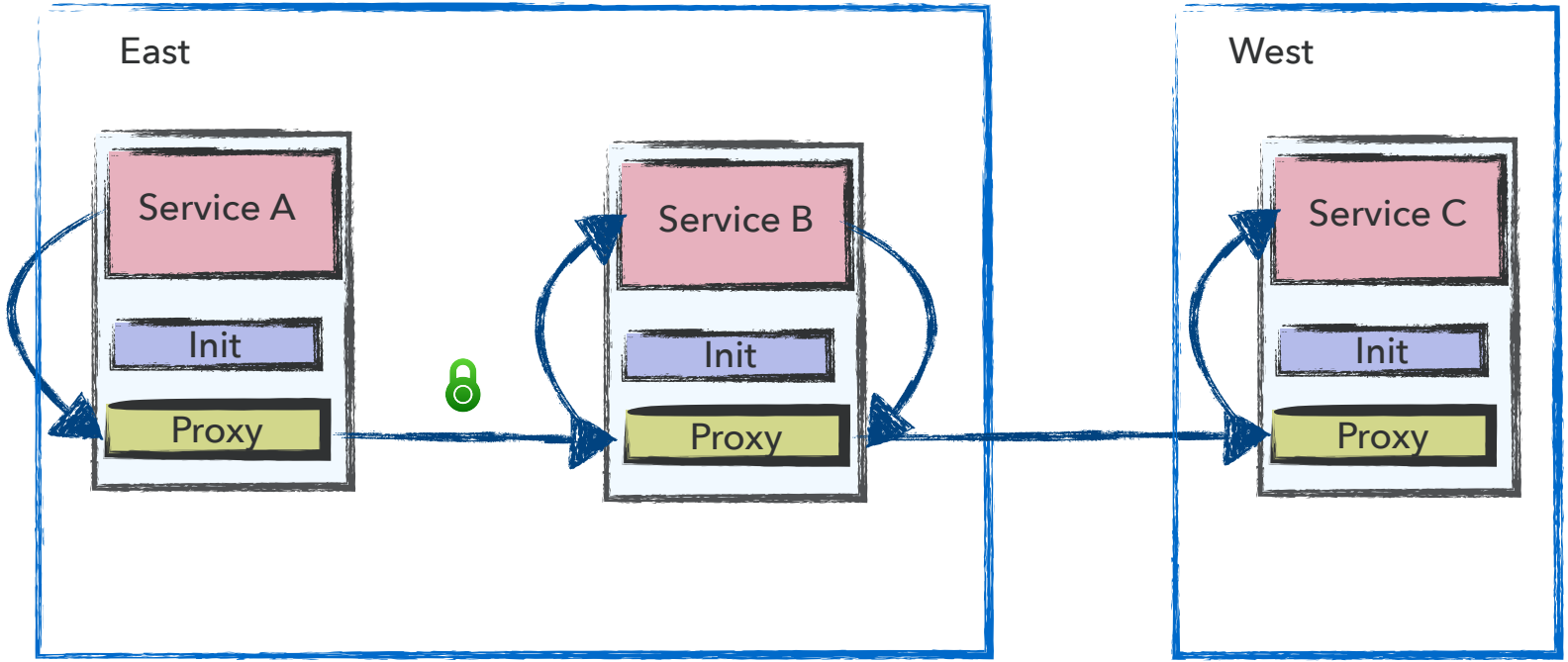
Request path



Request path



Request path



eBPF overview

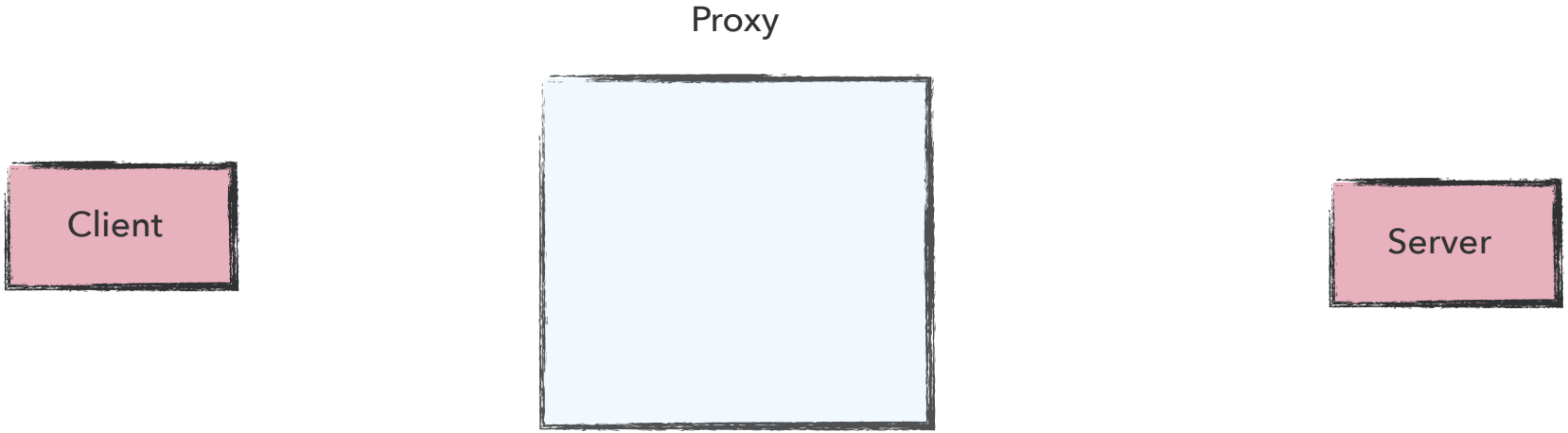
eBPF overview

- Interface to make the kernel programmable
- Event driven, runs in kernel space
- Efficient - allows developers to avoid data transfers to user space and minimises context switches
- Safe - programs are subject to strict requirements so halting the kernel is impossible

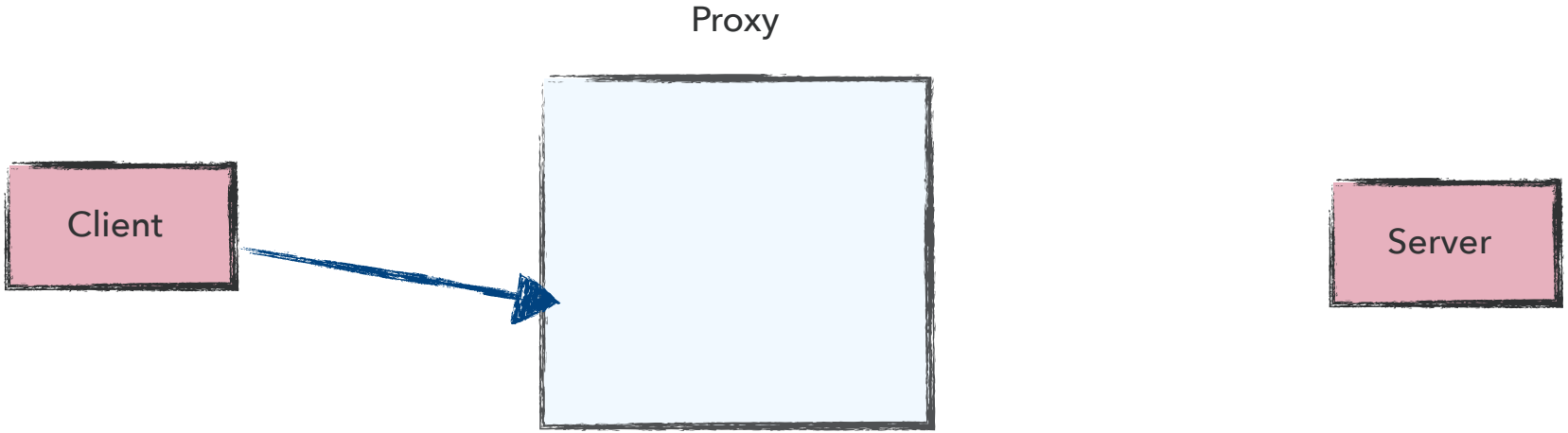
Traditional reverse proxy

Traditional reverse proxy

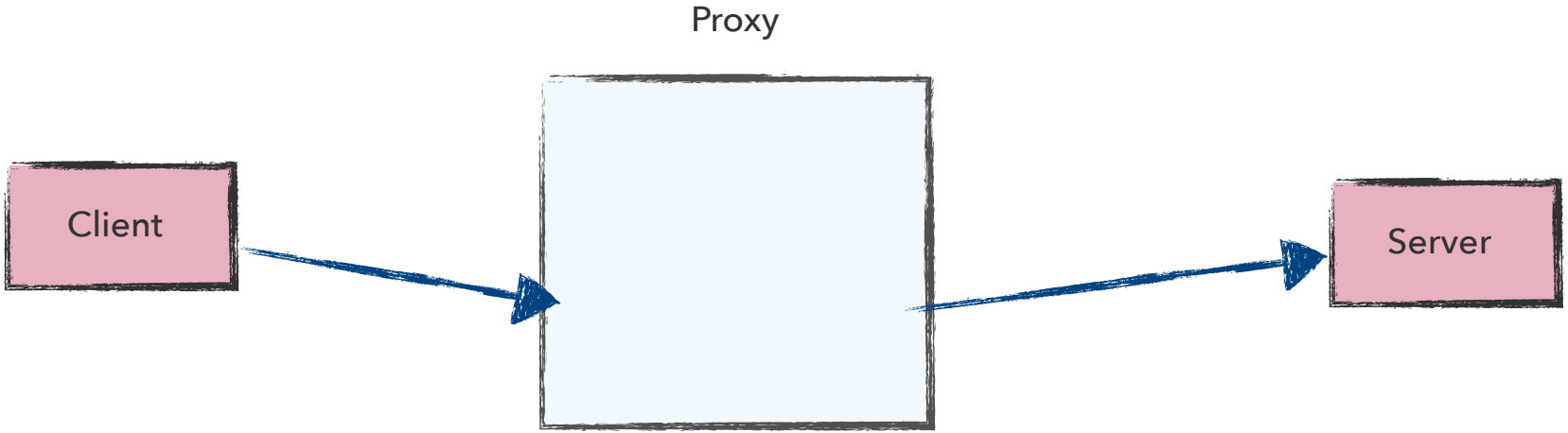
Traditional reverse proxy



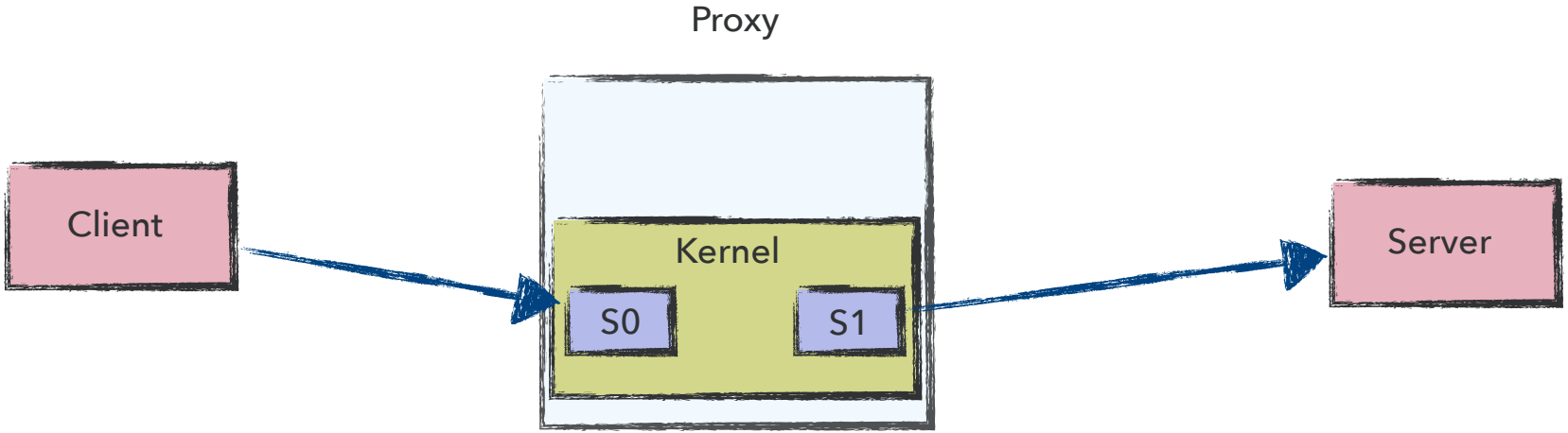
Traditional reverse proxy



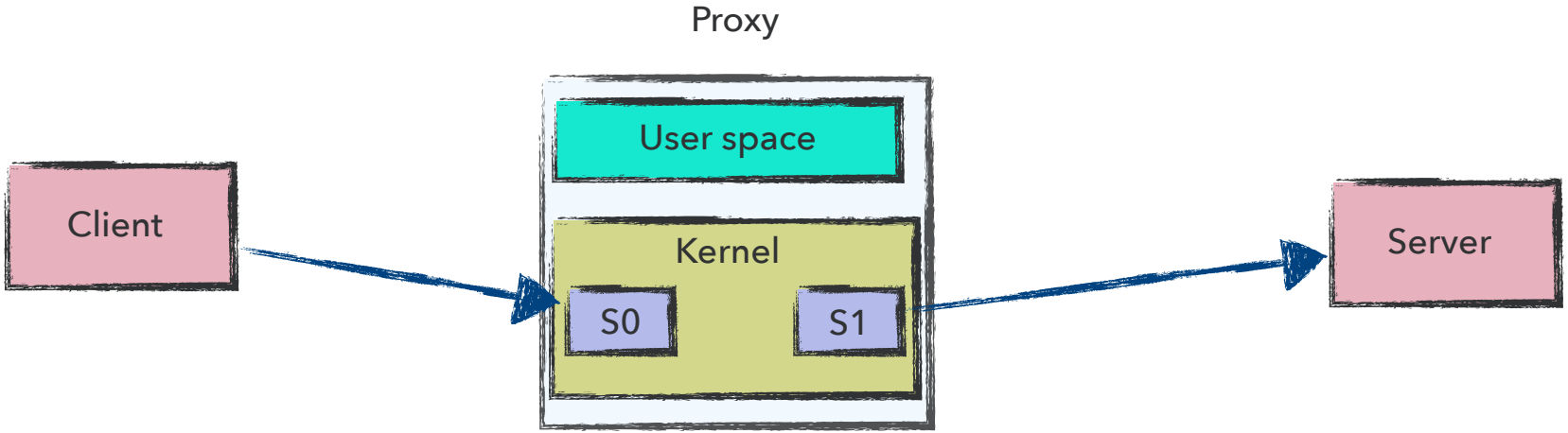
Traditional reverse proxy



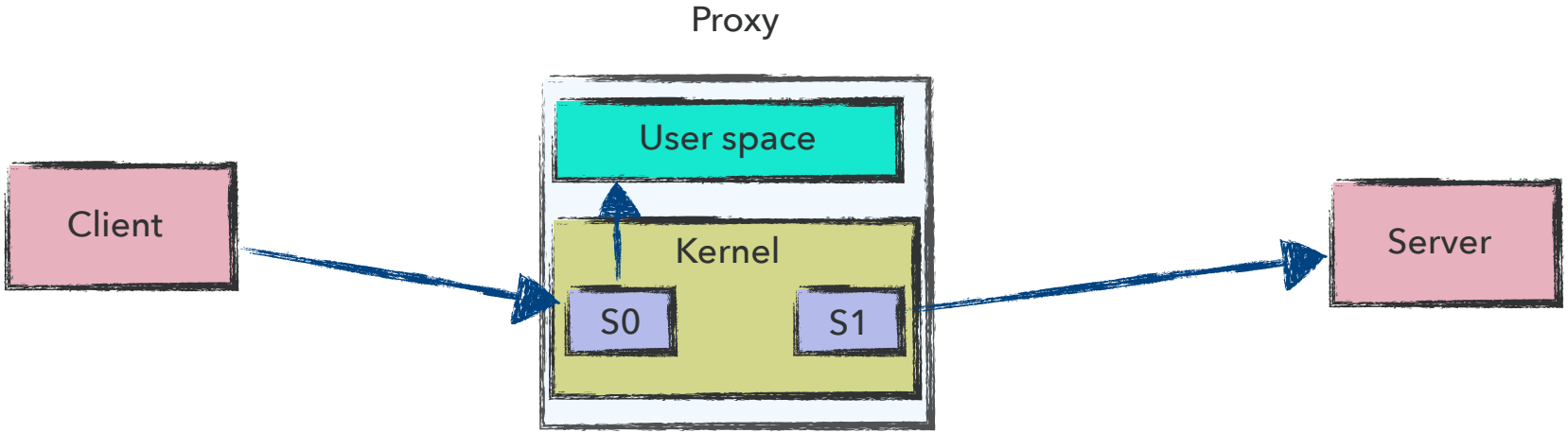
Traditional reverse proxy



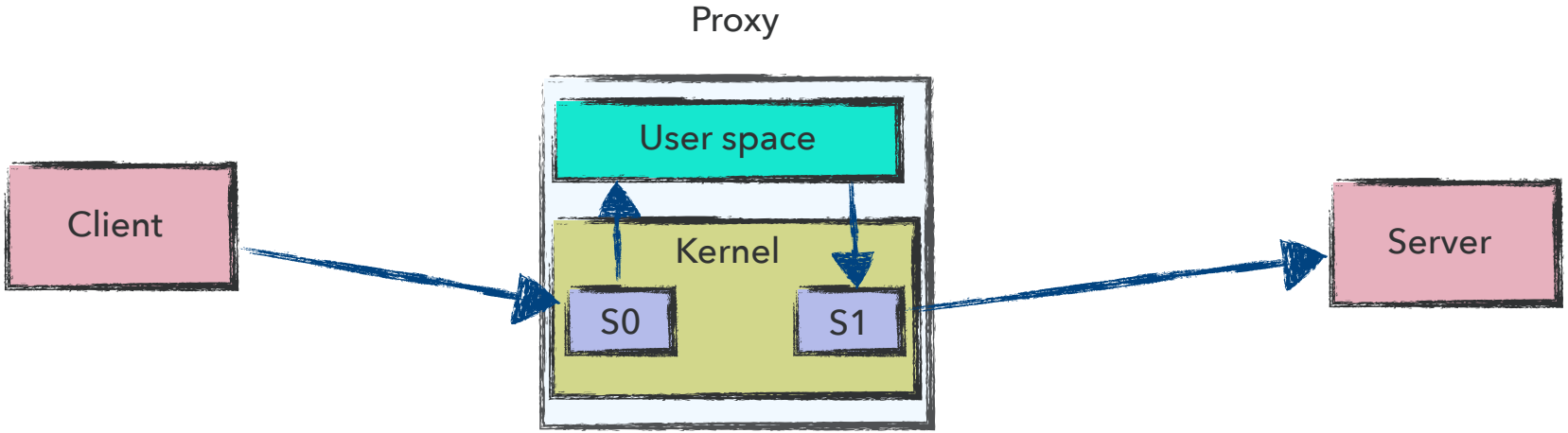
Traditional reverse proxy



Traditional reverse proxy



Traditional reverse proxy



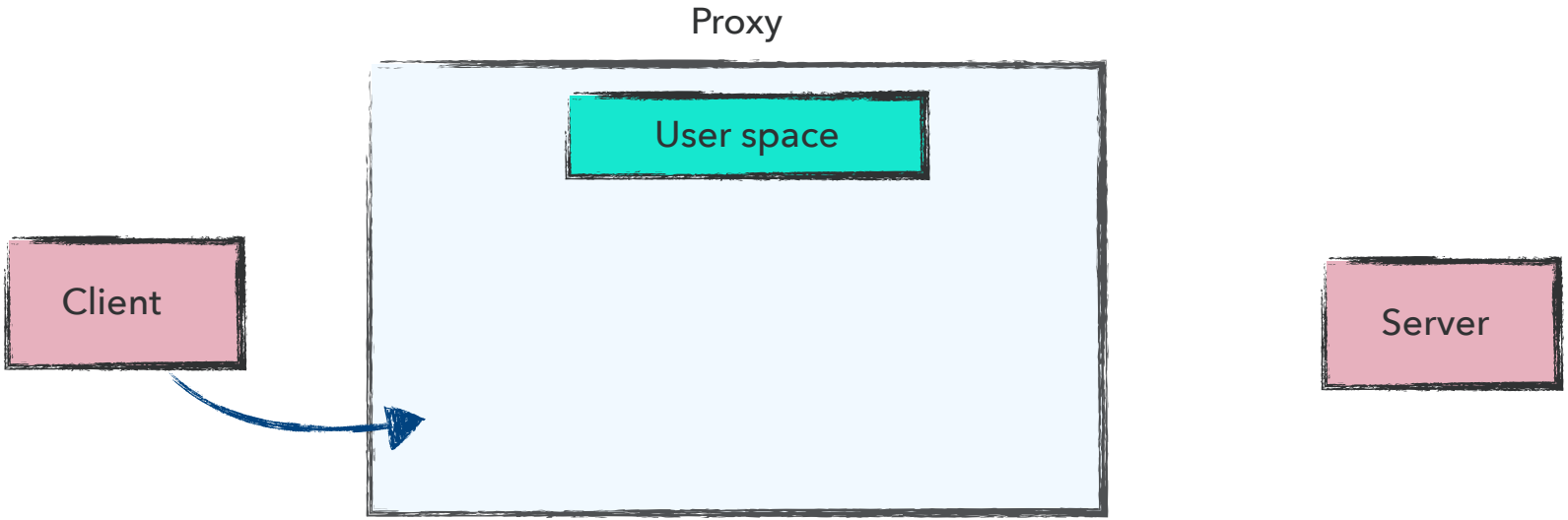
Drawbacks

- Syscall costs
- Wakeup latency
- Data copying costs

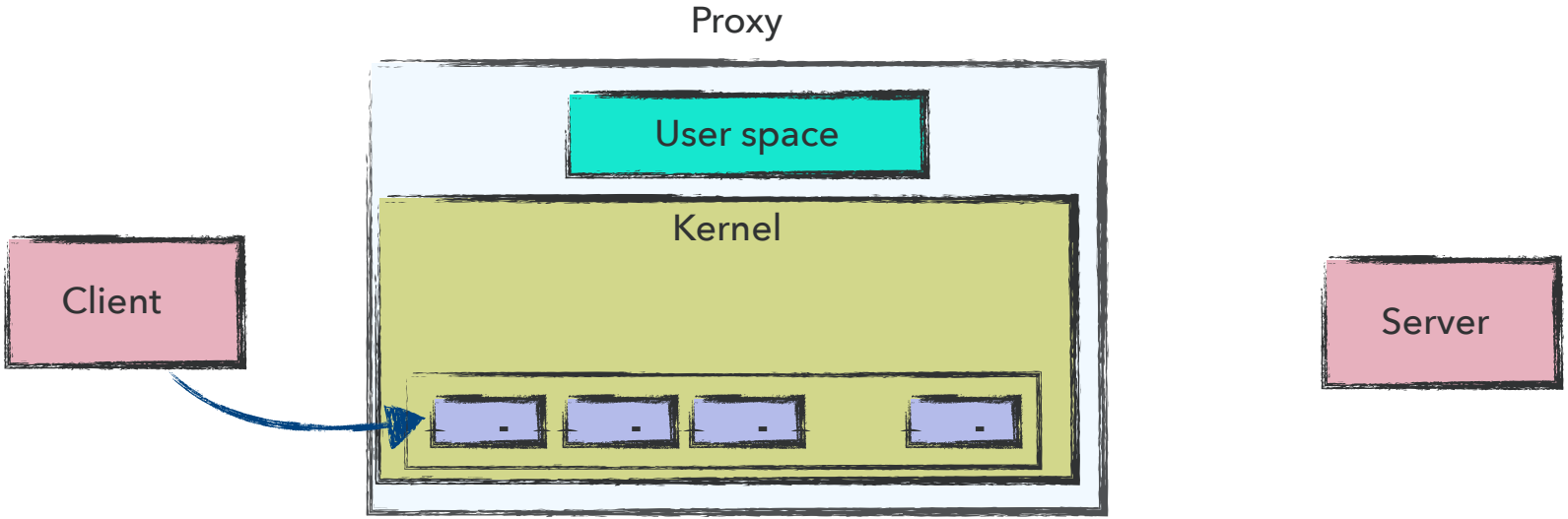
Sockmap

- eBPF's sock map is a data structure that can help in this scenario
- It stores references to sockets and allows eBPF programs to redirect TCP packets entirely in kernel space
- It can be used to accelerate reverse proxies and load balancers

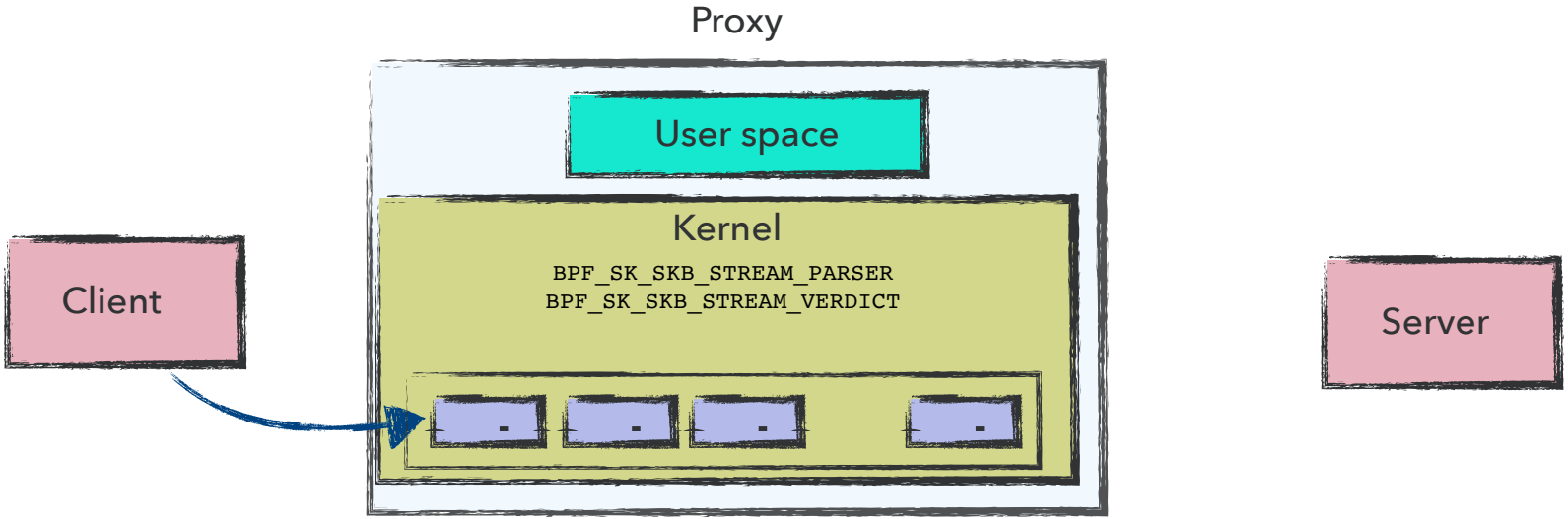
Sockmap based proxy



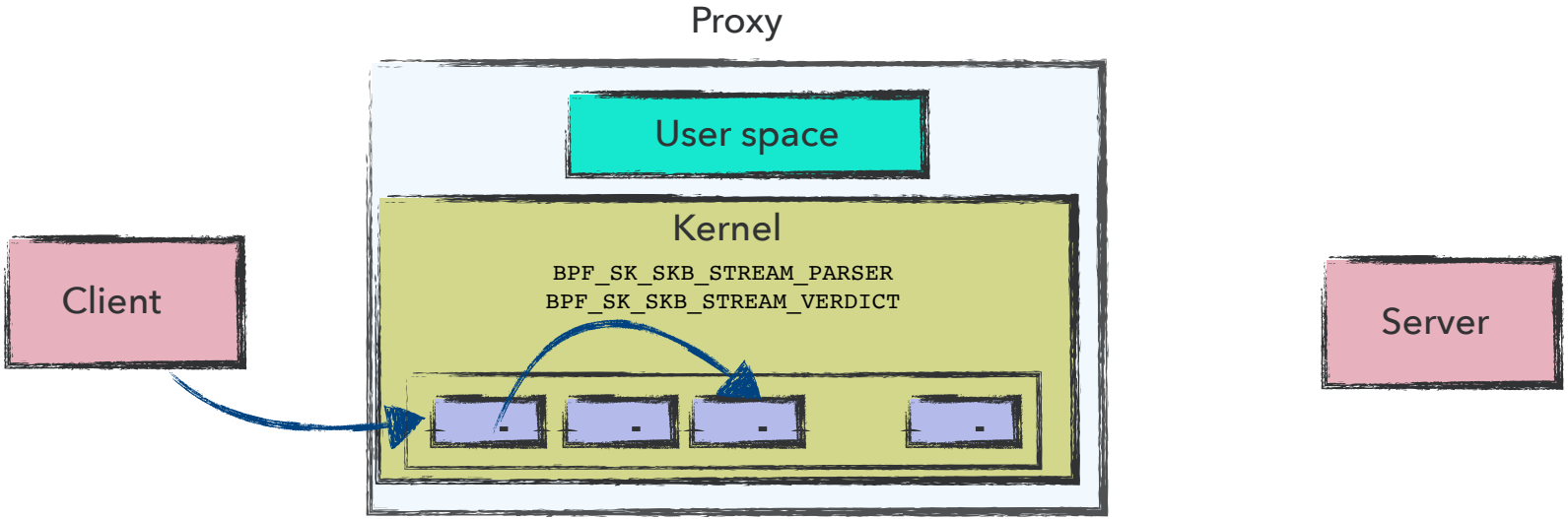
Sockmap based proxy



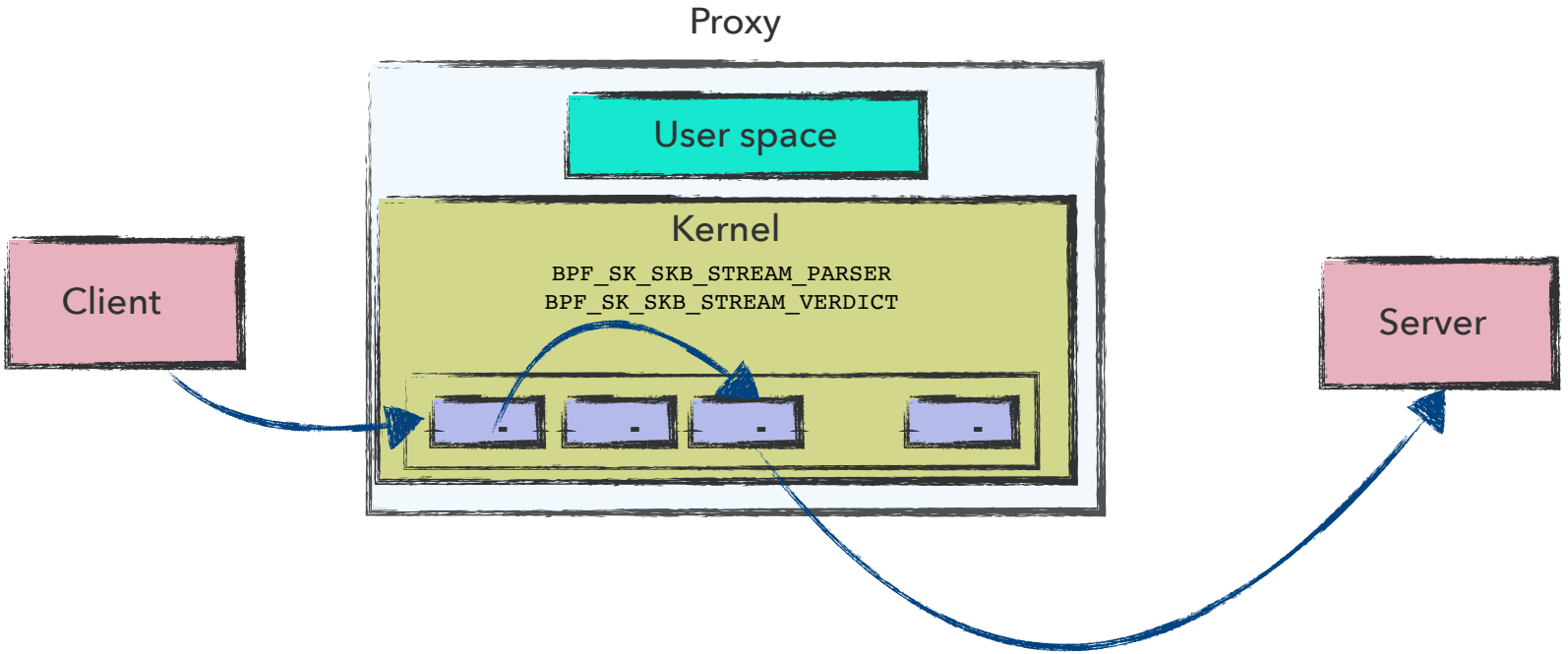
Sockmap based proxy



Sockmap based proxy



Sockmap based proxy



Advantages

- No copying of data from kernel to user space
- No context switching
- Flexibility

eBPF strengths

- Pod and container statistics
- Traffic metrics
- Syscall interception and audit
- Layer 4 routing

The sidecar is going away !!!

The sidecar is going away !!!

BUOYANT

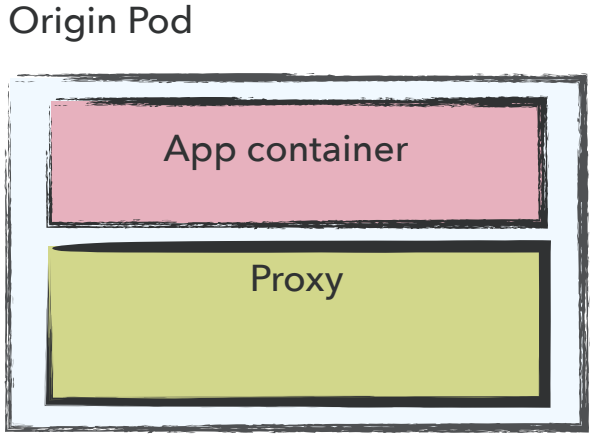
LINKERD

Could it all be true ?

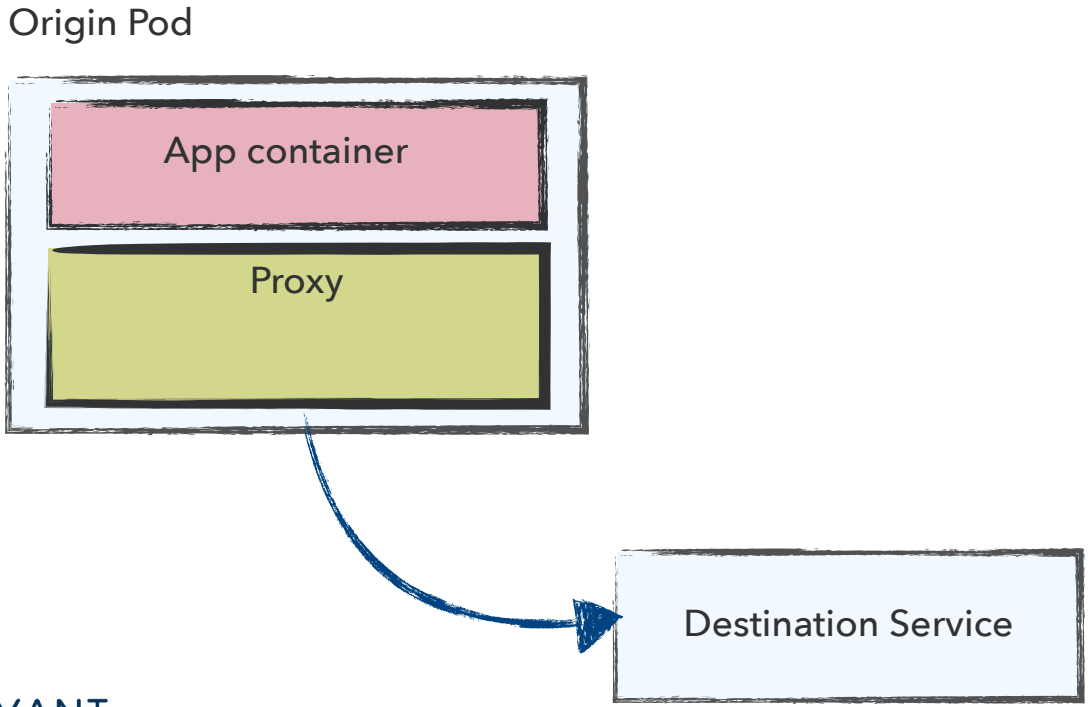
eBPF use cases

- Observability - container level statistics, continue profiling of applications
- Security - syscall interception and audit
- Reliability - load balancing, reverse proxies

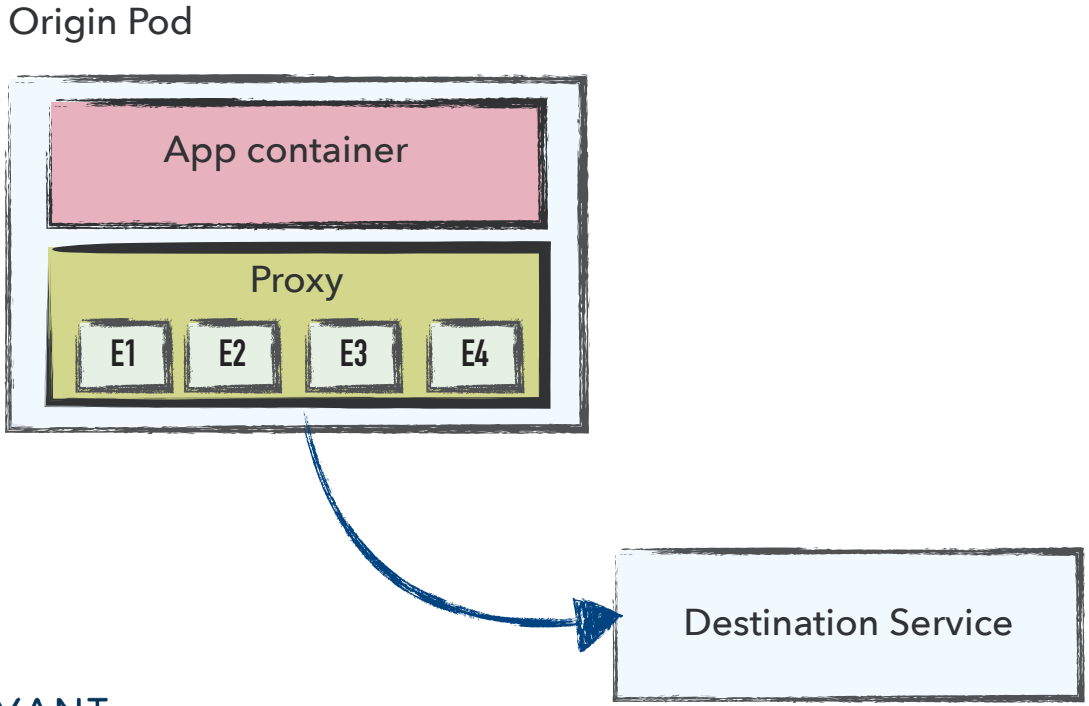
Latency aware load balancing



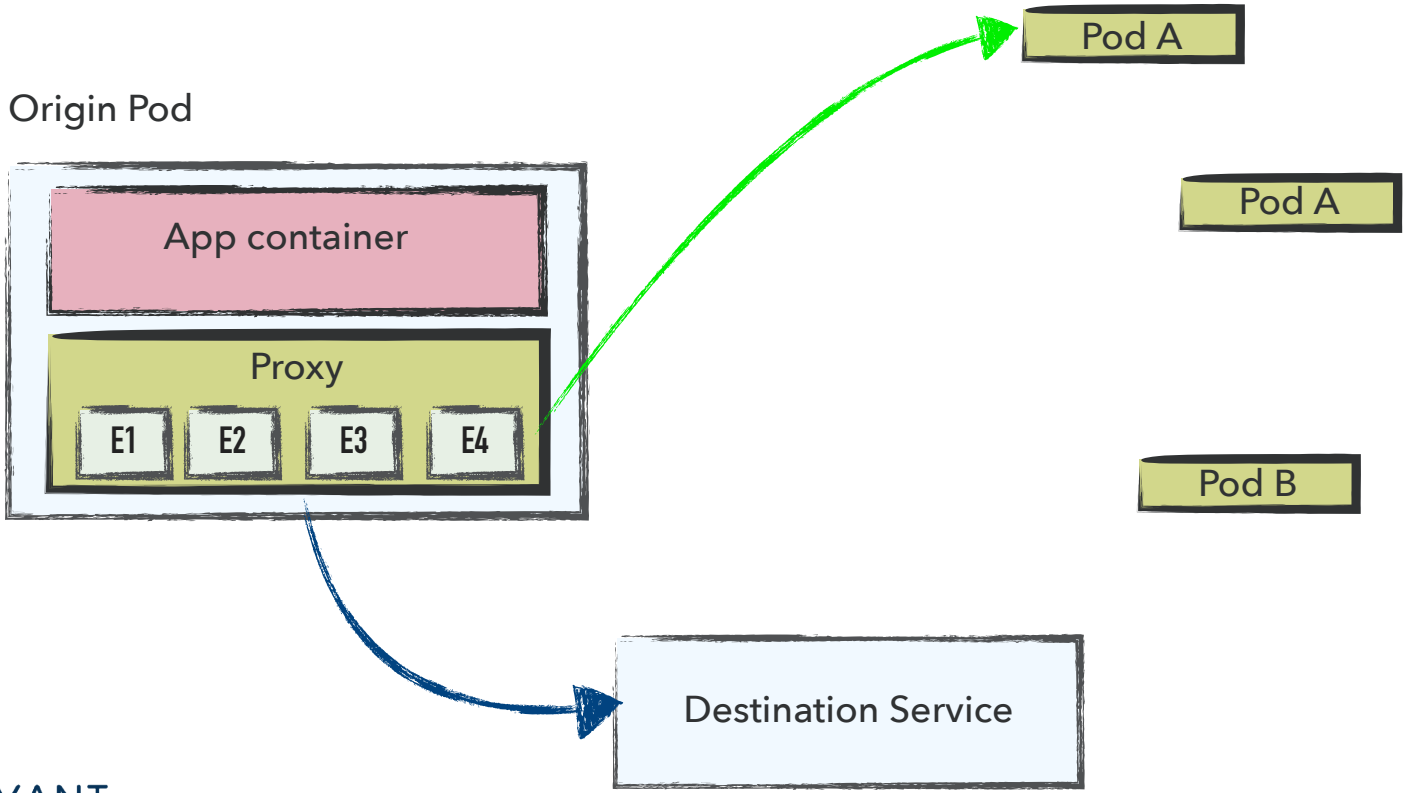
Latency aware load balancing



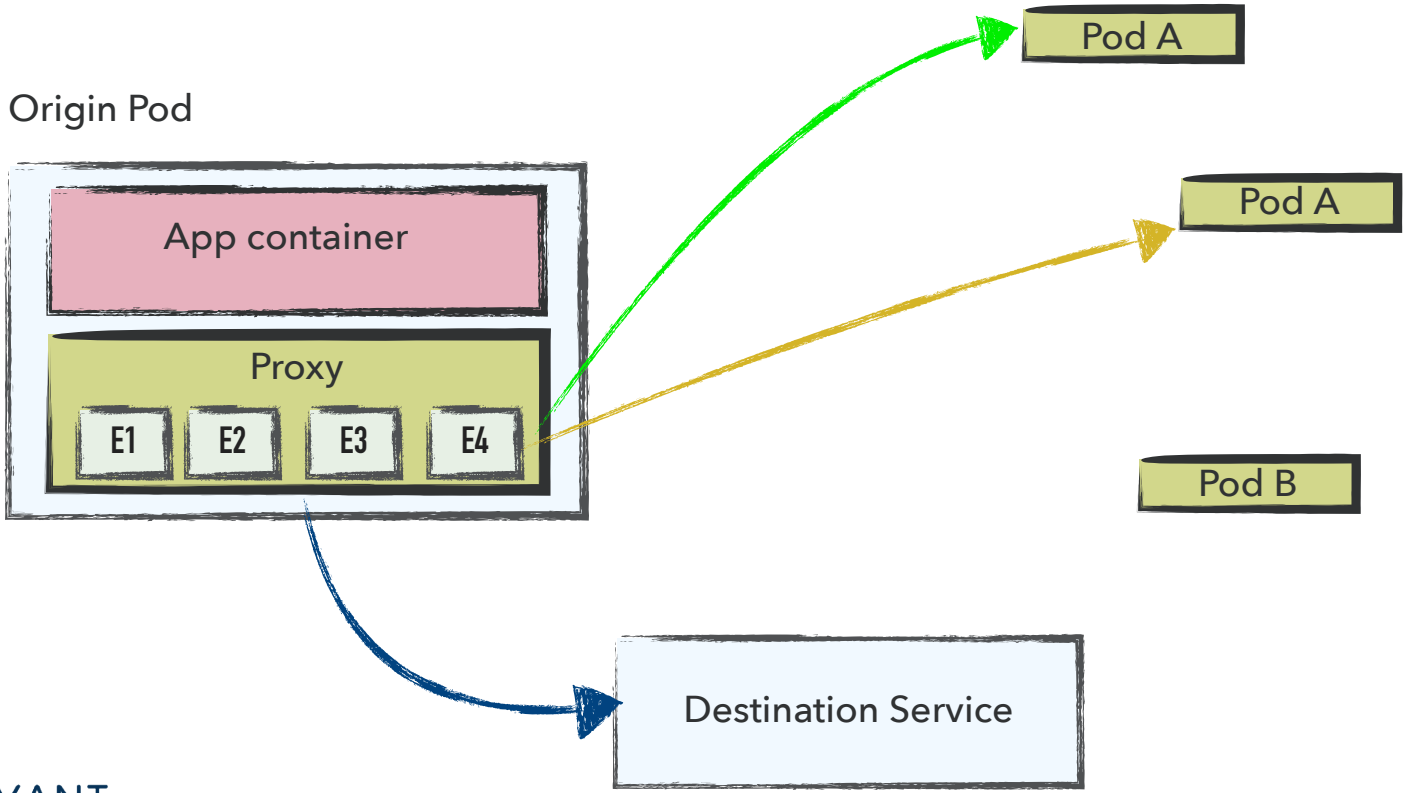
Latency aware load balancing



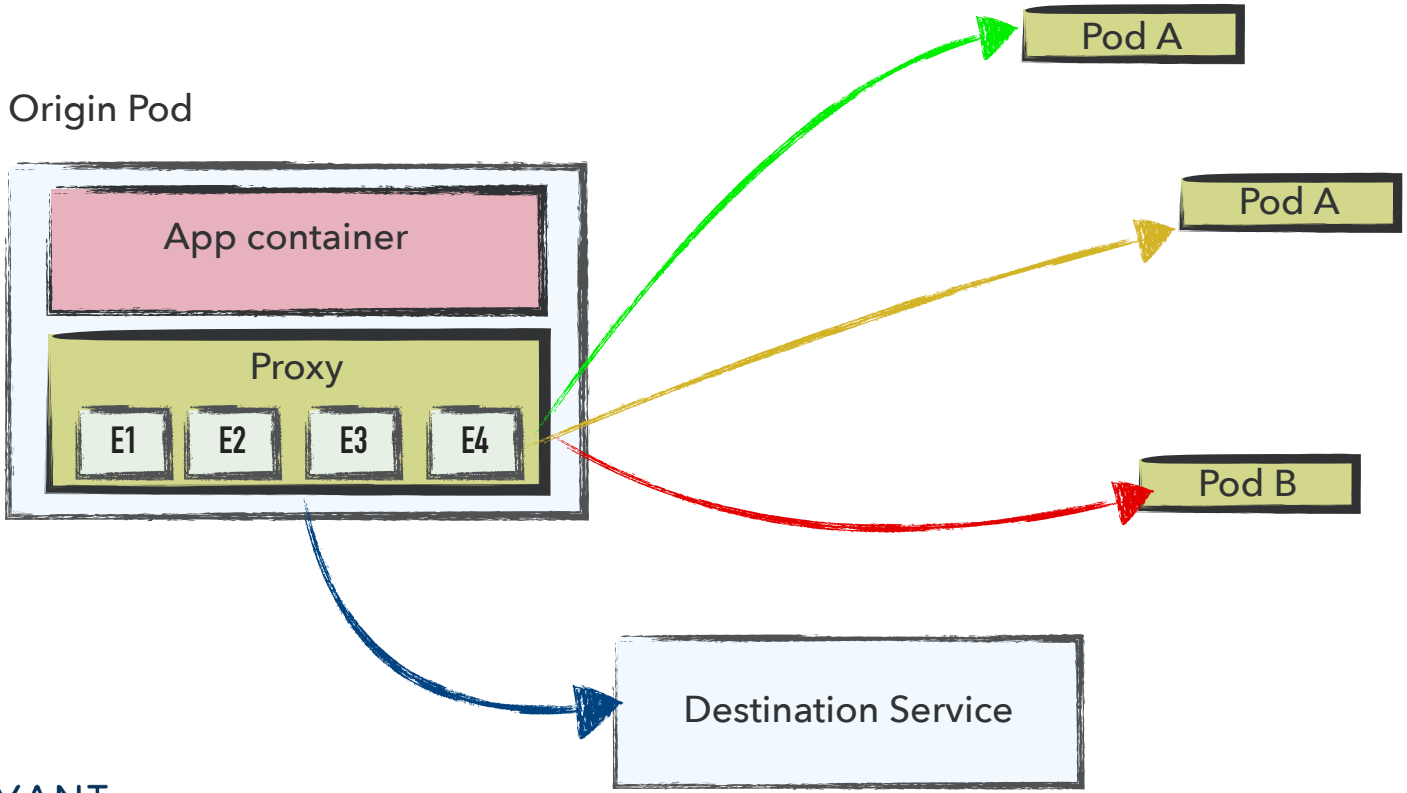
Latency aware load balancing



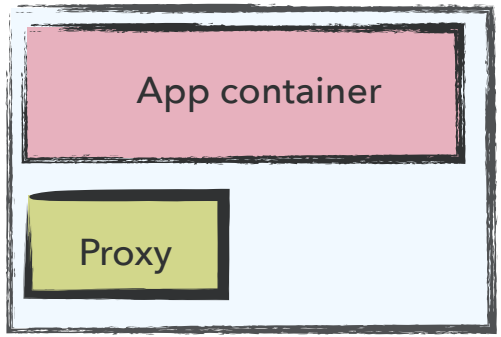
Latency aware load balancing



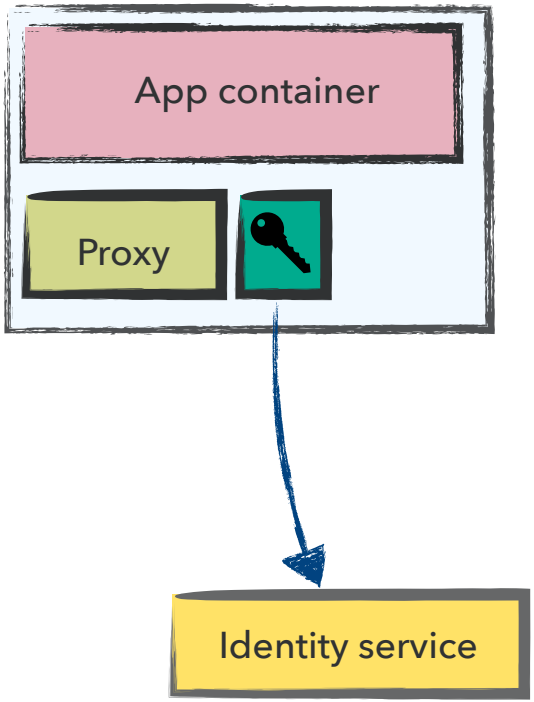
Latency aware load balancing



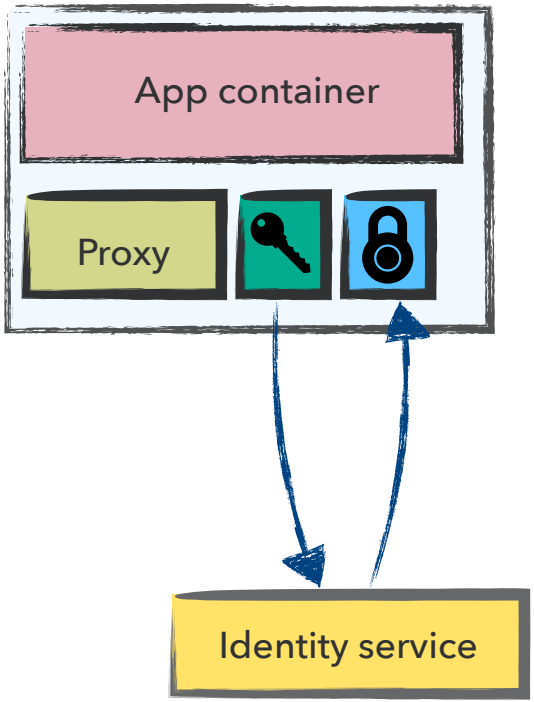
Identity based policy



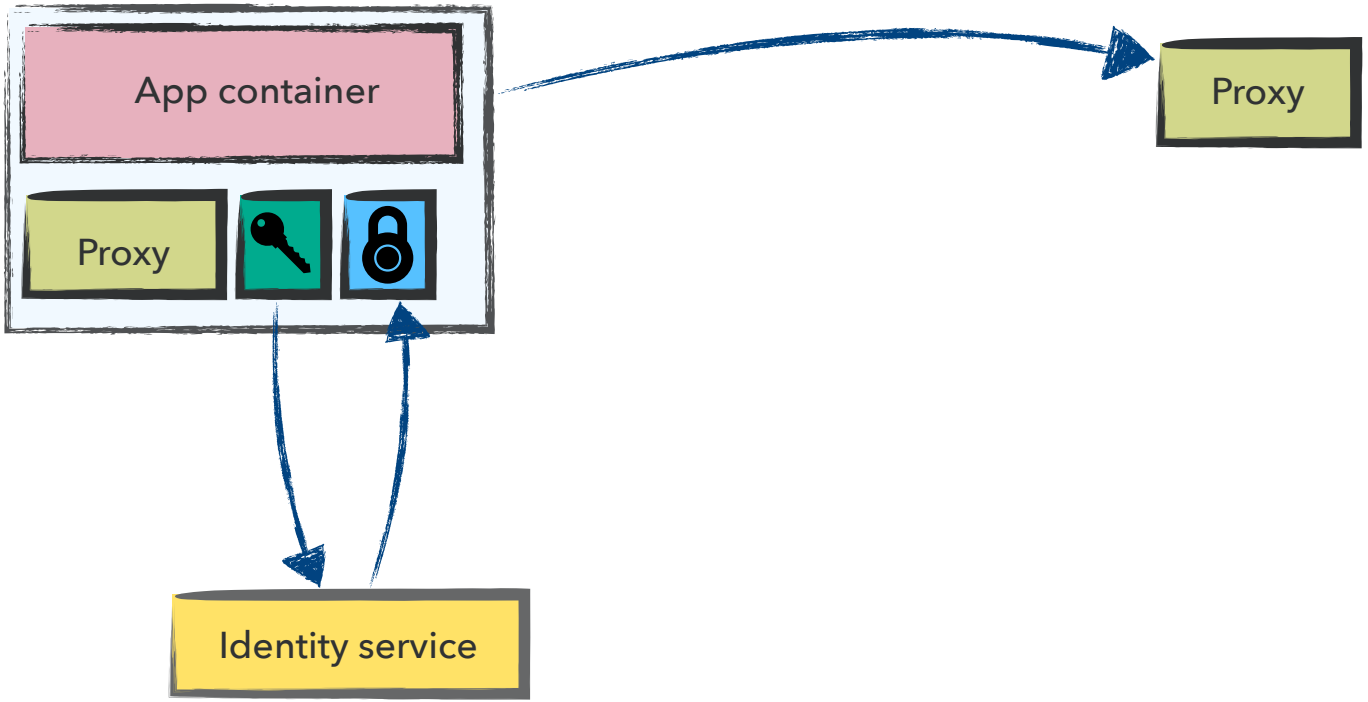
Identity based policy



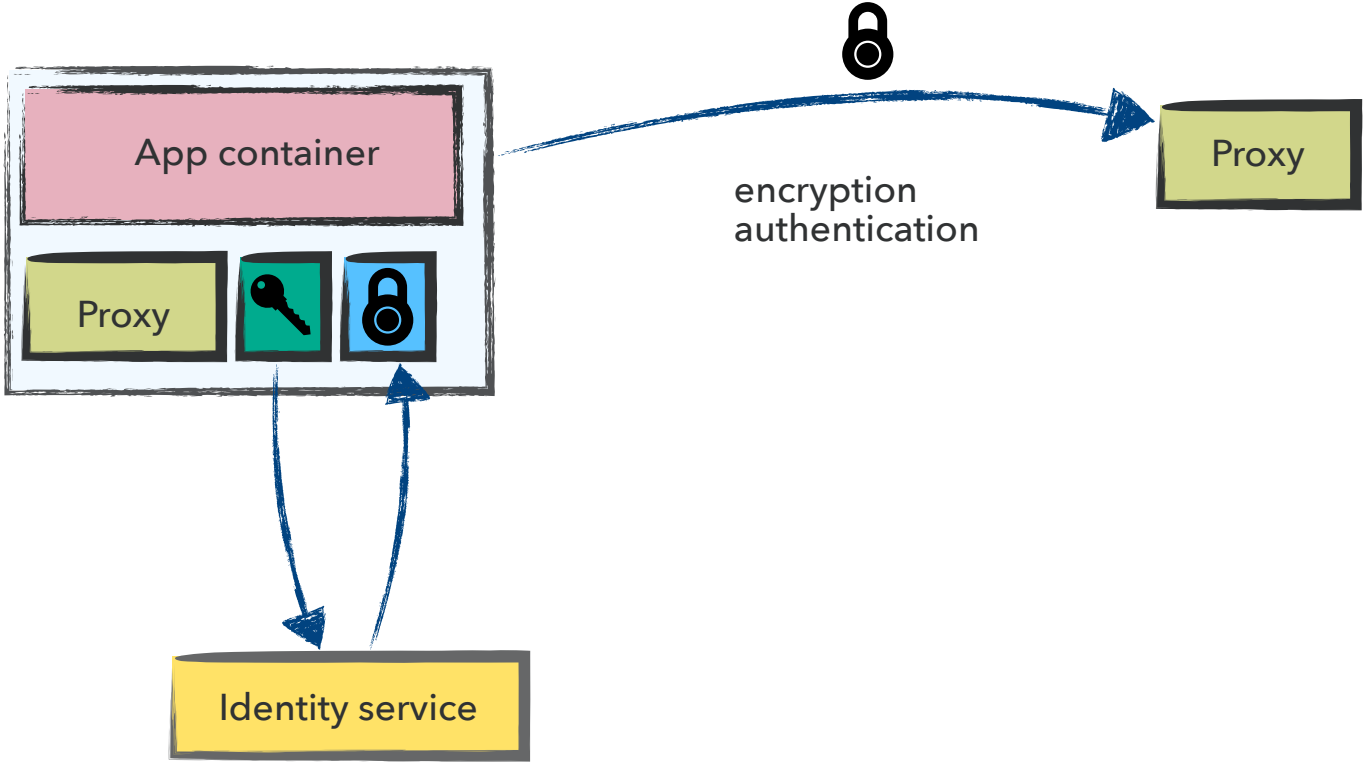
Identity based policy



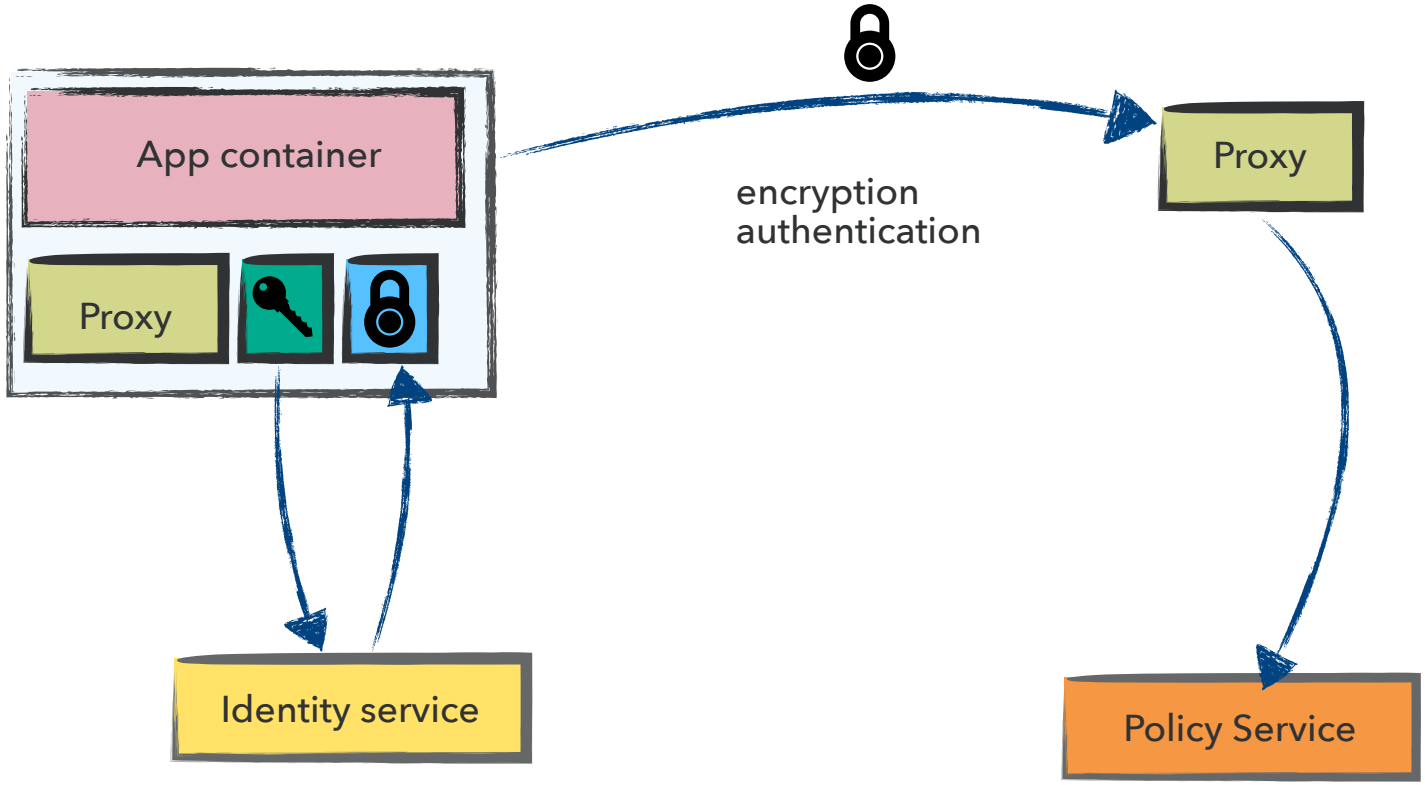
Identity based policy



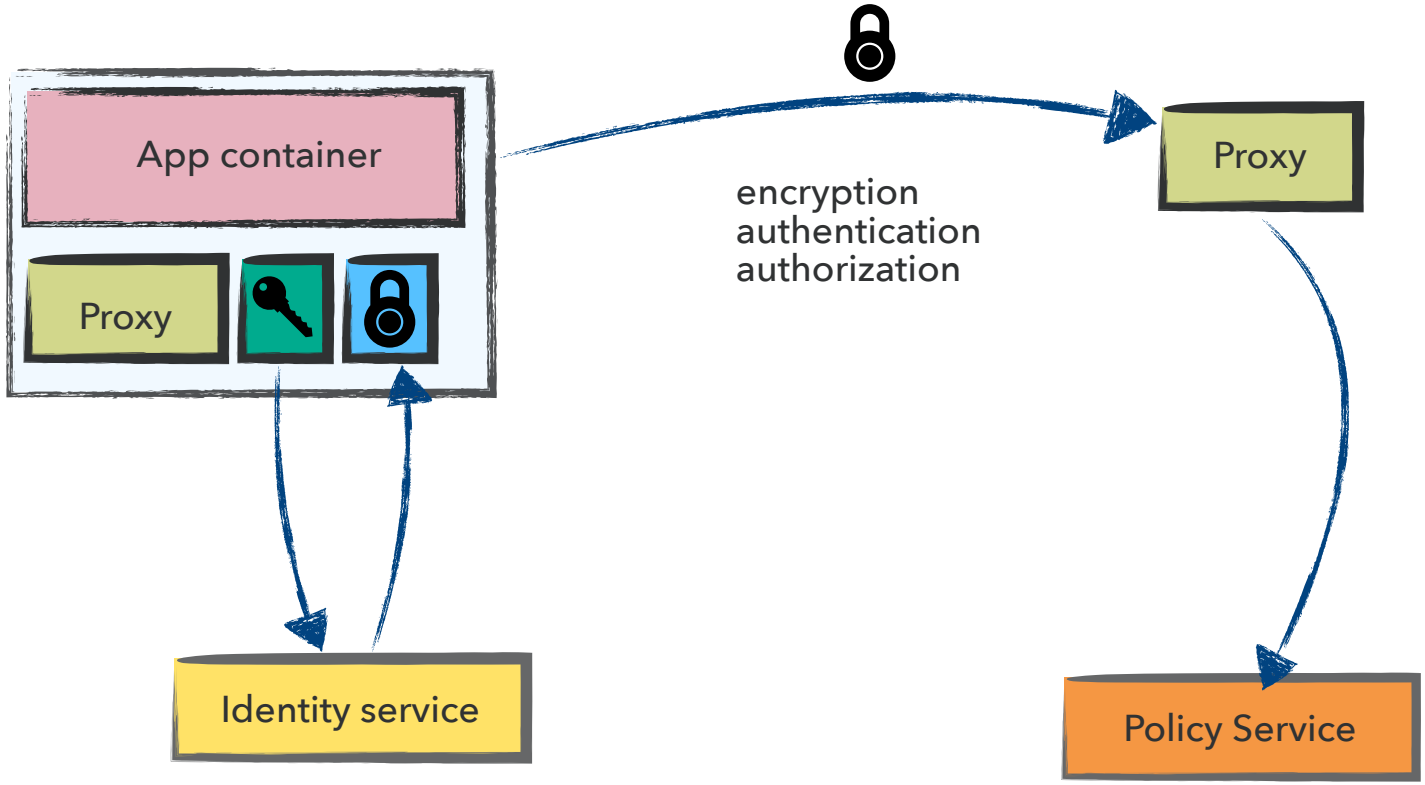
Identity based policy



Identity based policy



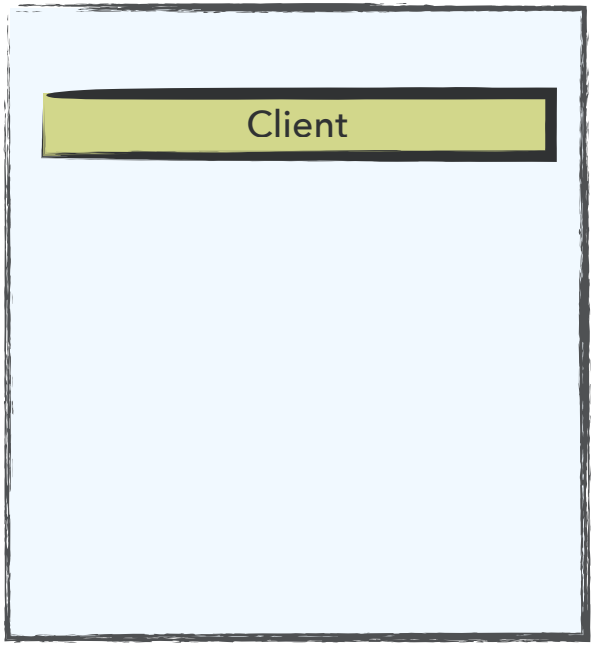
Identity based policy



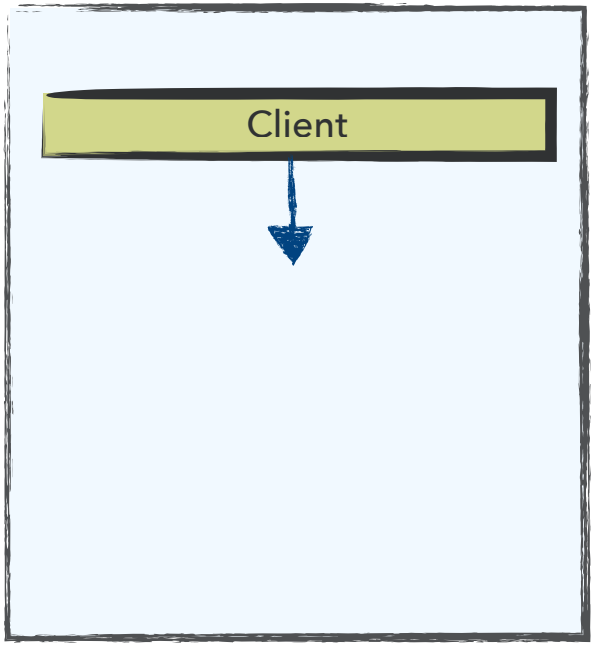
Identity based policy

9090 / admin-http
✓ Any traffic is allowed if: • mTLS'd from the `metrics-api` ServiceAccount
✓ Any traffic is allowed if: • mTLS'd from the `dive-viz` ServiceAccount in the `dive-viz` namespace
✓ Configured Kubernetes probes are allowed ⊘ Otherwise, traffic is denied
All other ports
✓ All traffic is allowed

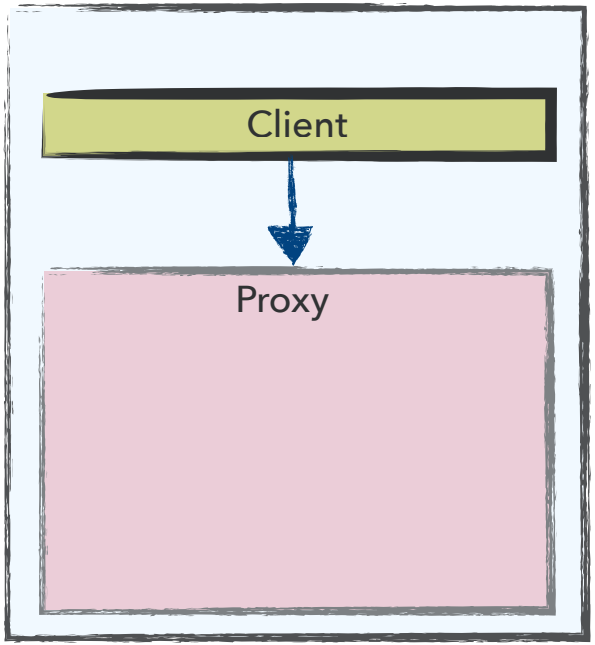
Retries



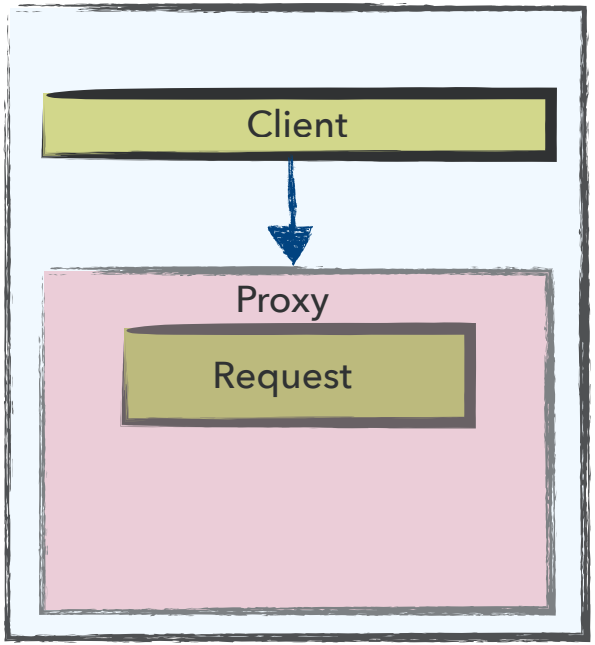
Retries



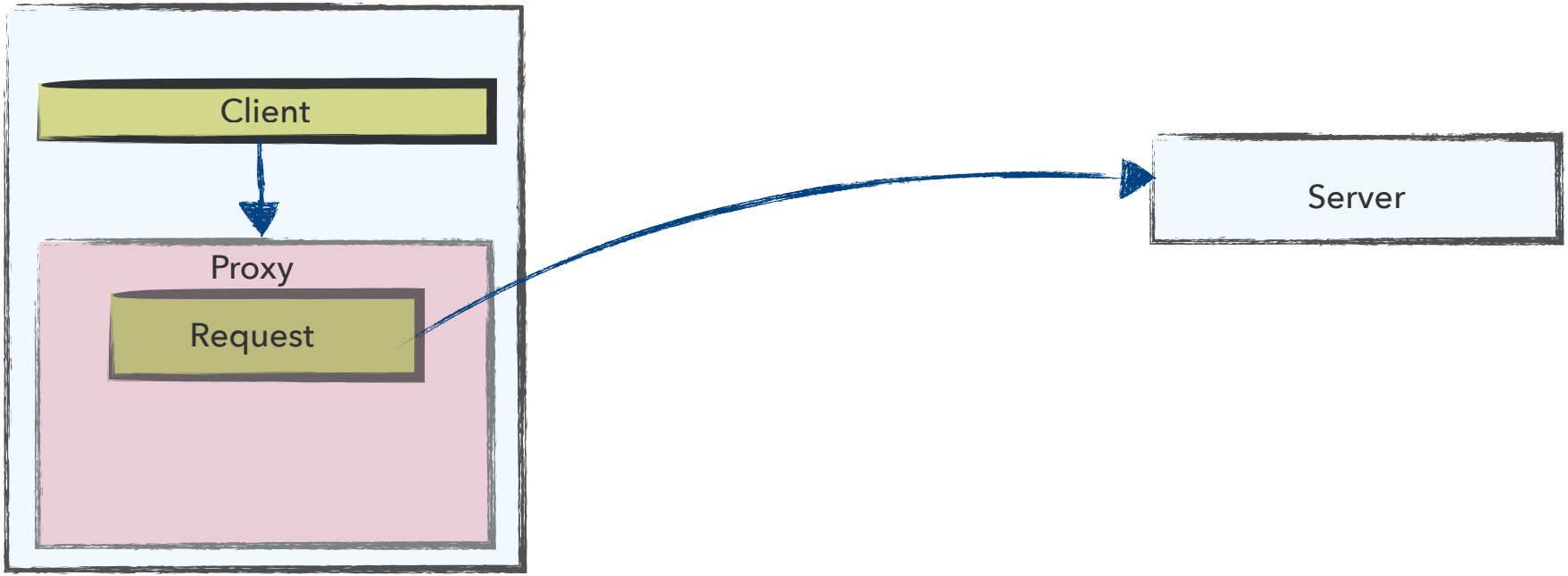
Retries



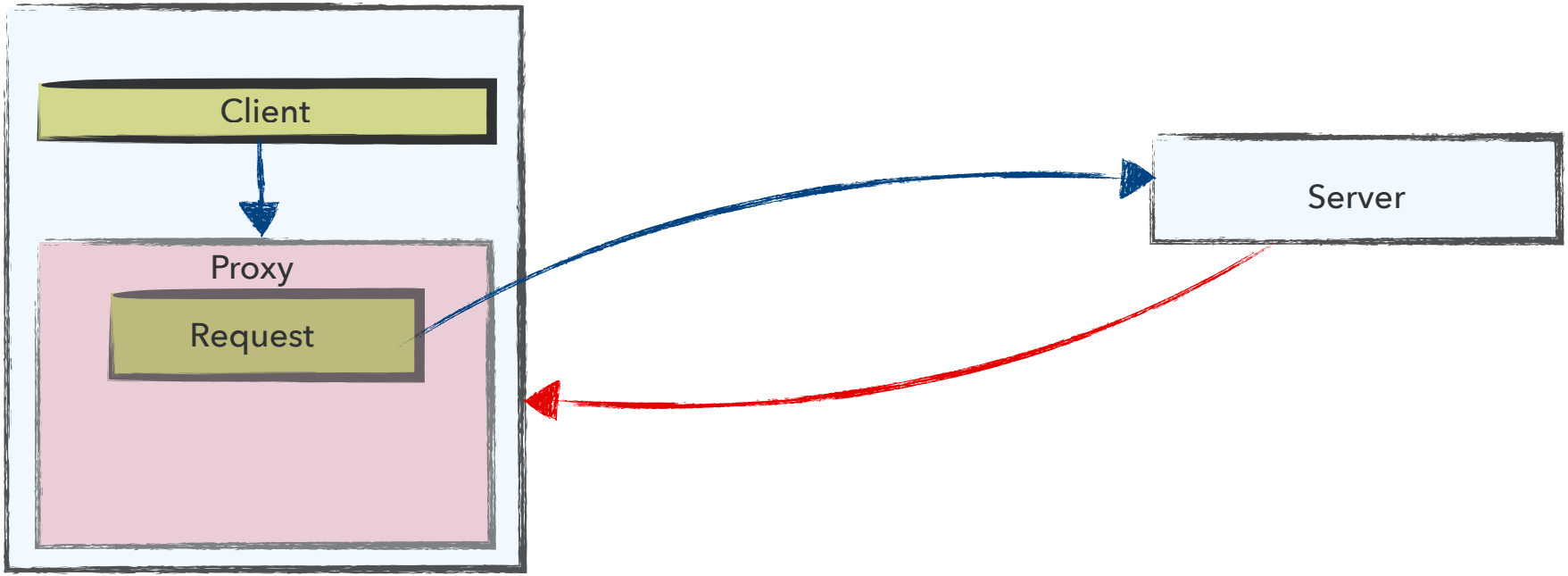
Retries



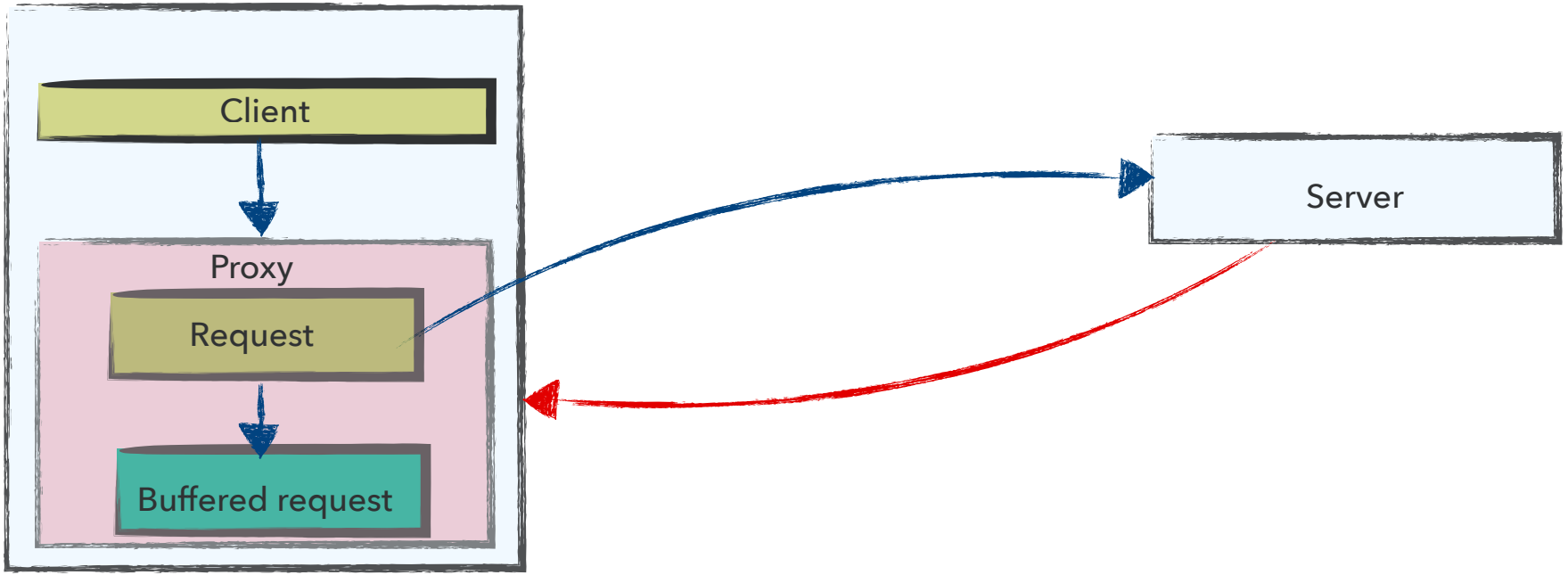
Retries



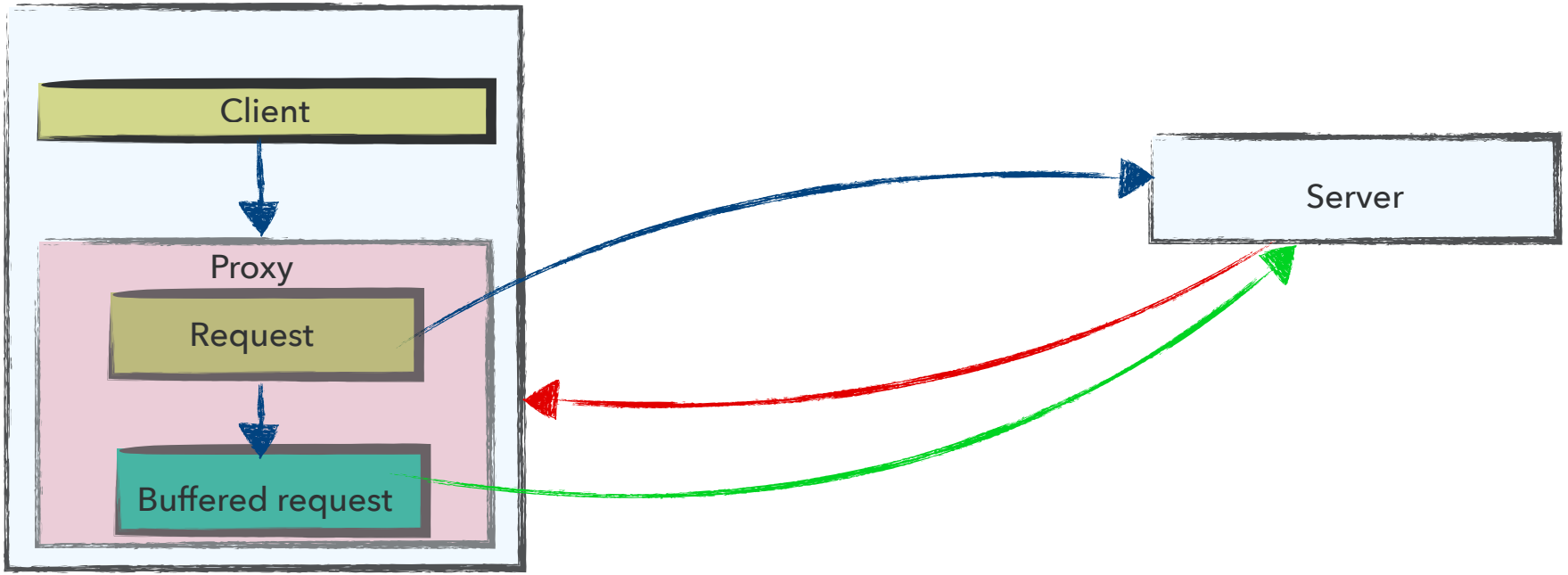
Retries



Retries



Retries



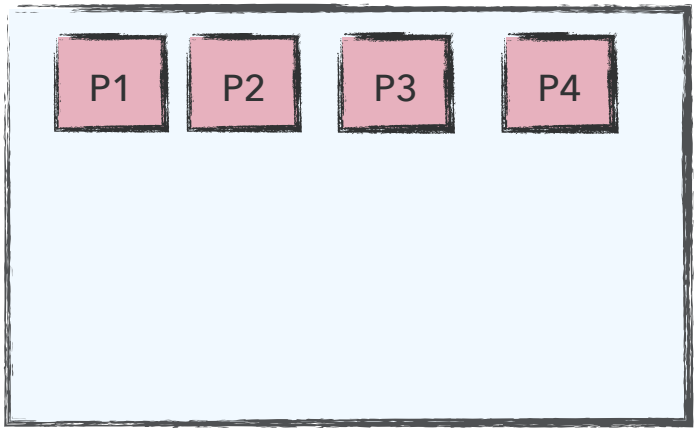
eBPF programs

- Not allowed to block
- No unbounded loops
- Limited in size
- All possible paths of execution need to be known
- Limited state management

What else could we do ?

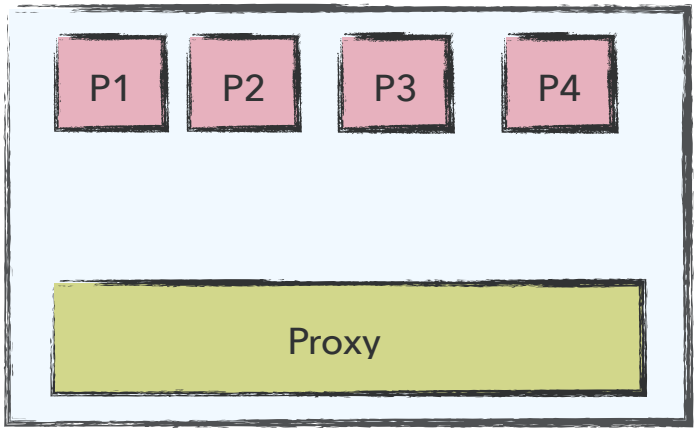
Shared proxy per node

Node 1



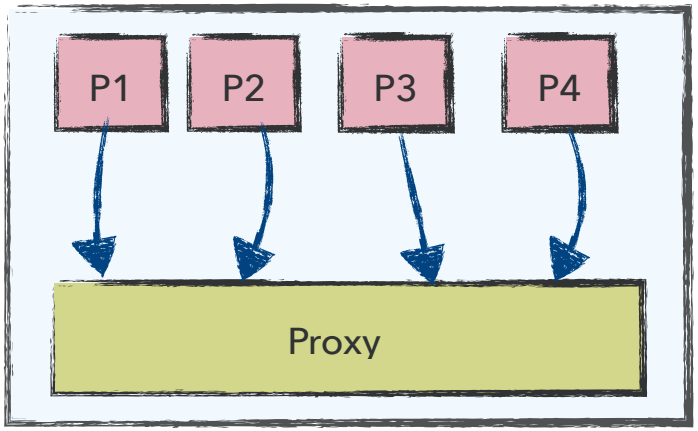
Shared proxy per node

Node 1



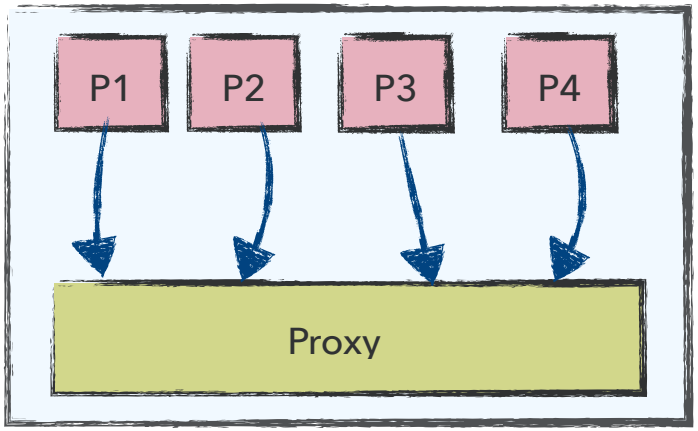
Shared proxy per node

Node 1

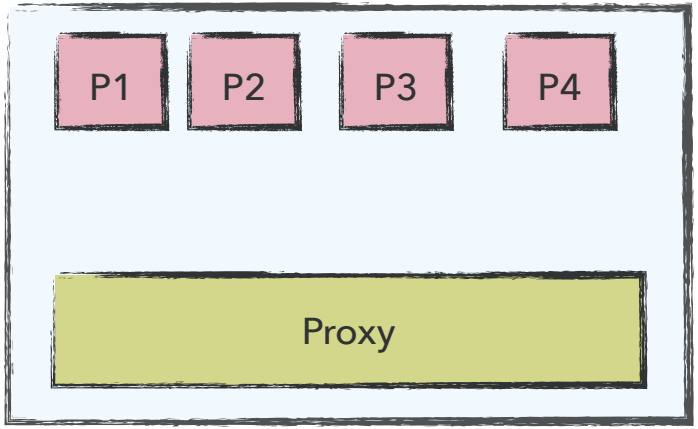


Shared proxy per node

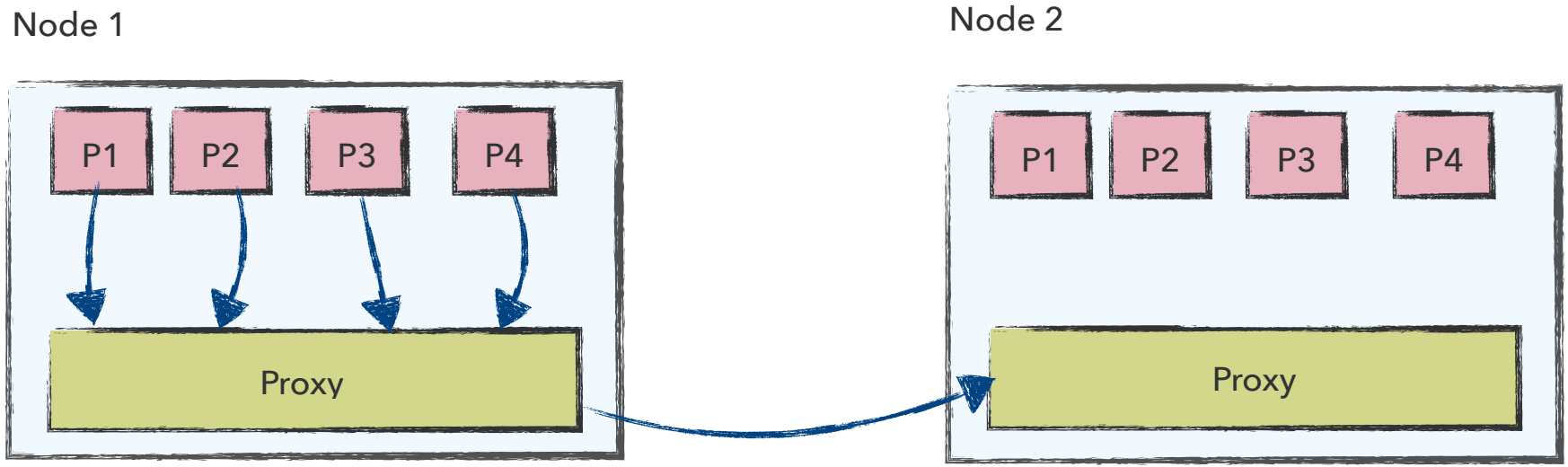
Node 1



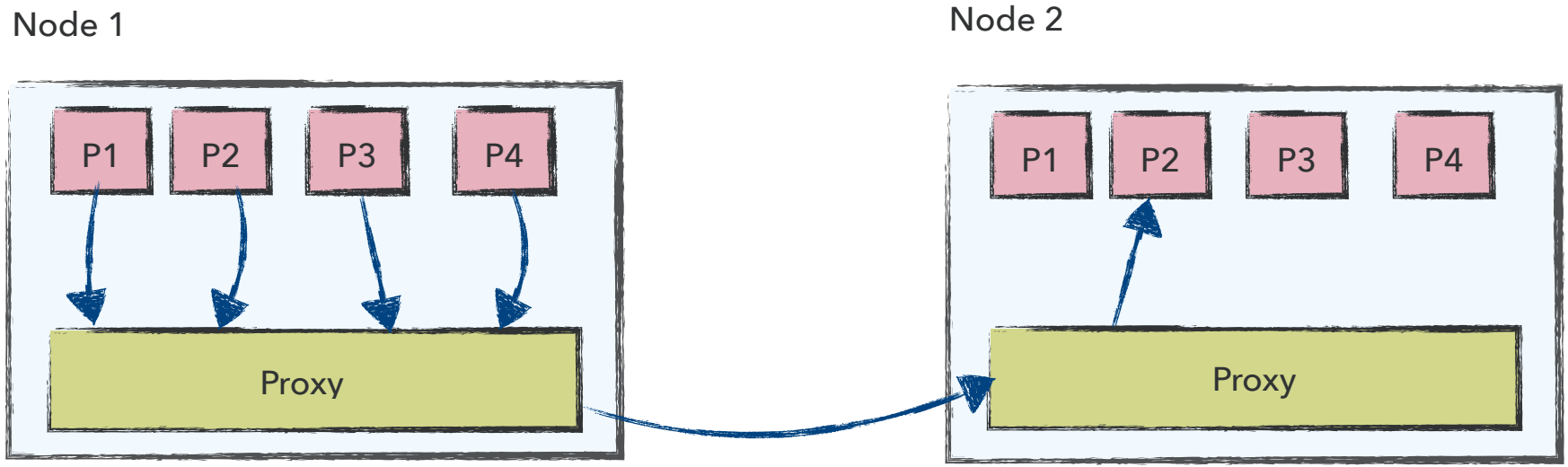
Node 2



Shared proxy per node



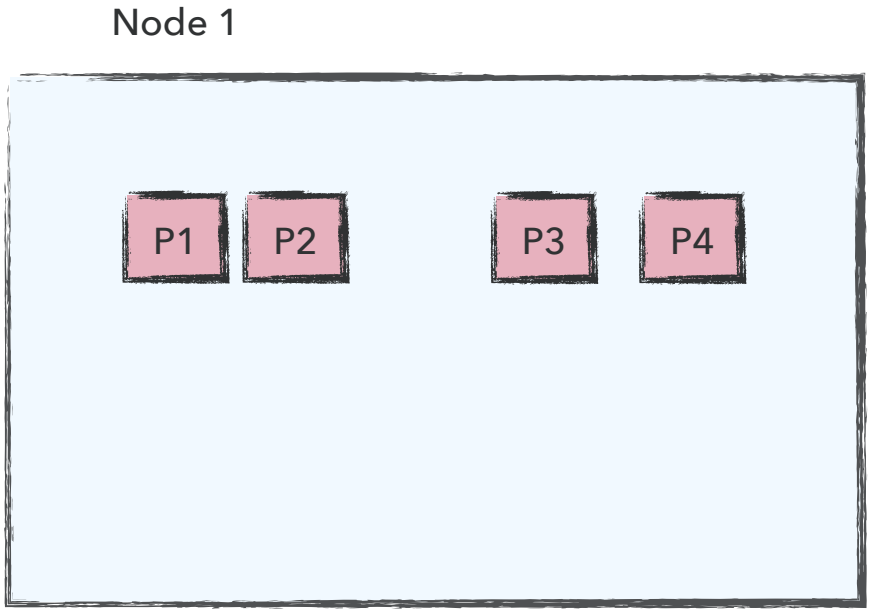
Shared proxy per node



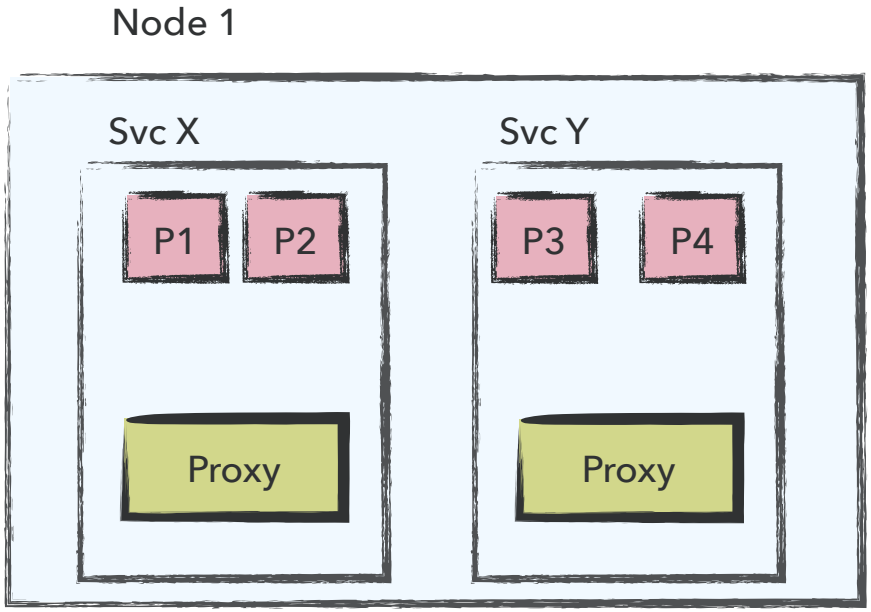
Shared proxy per node

- Resource starvation and fairness concerns
- Lack of resource optimisation granularity
- No isolation of secret material
- Increased blast radius when things go wrong

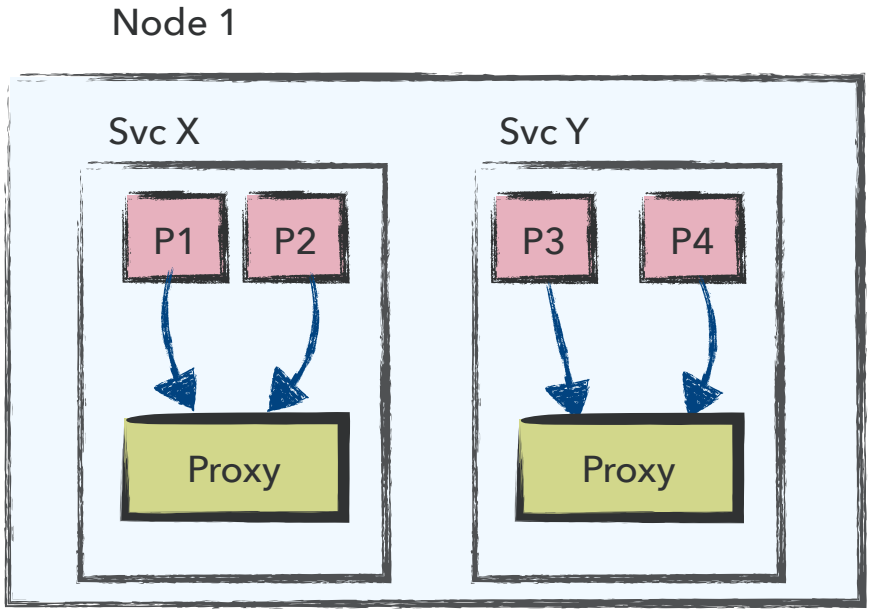
Shared proxy per service account



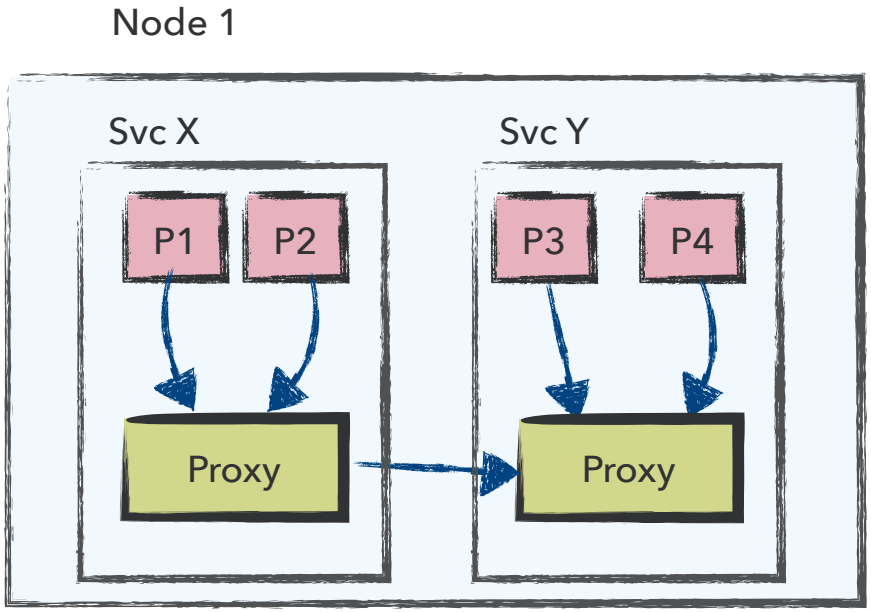
Shared proxy per service account



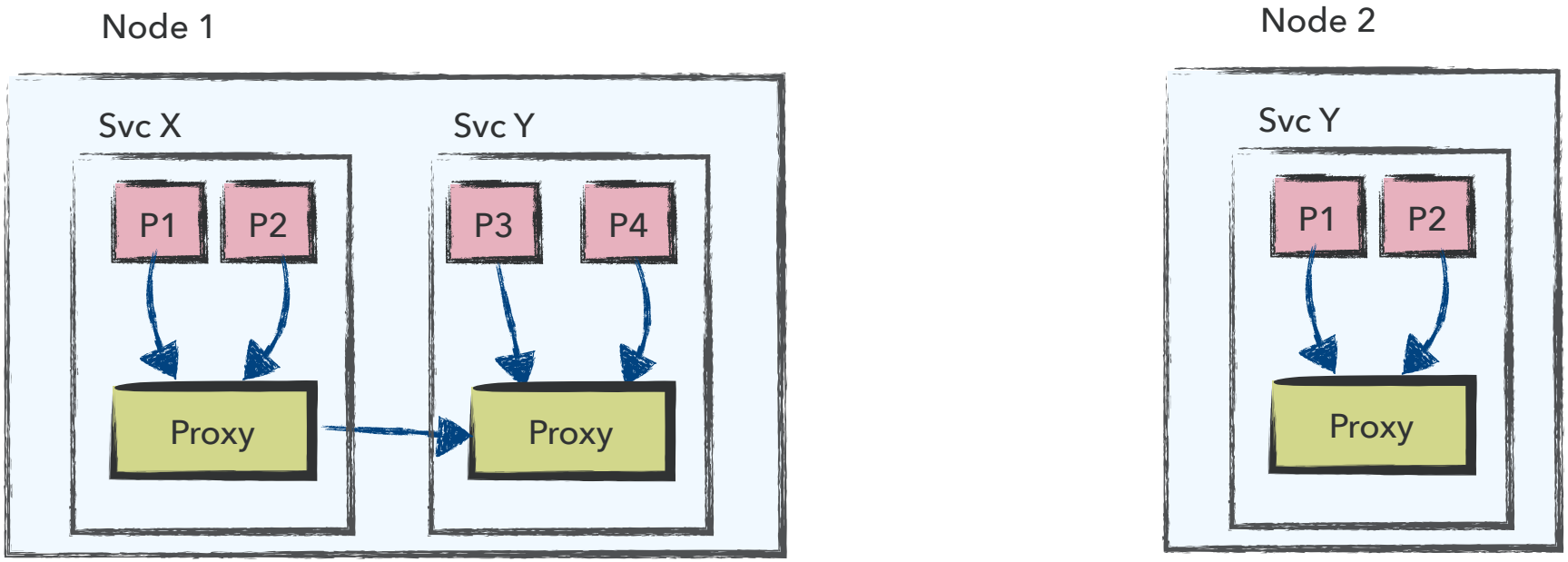
Shared proxy per service account



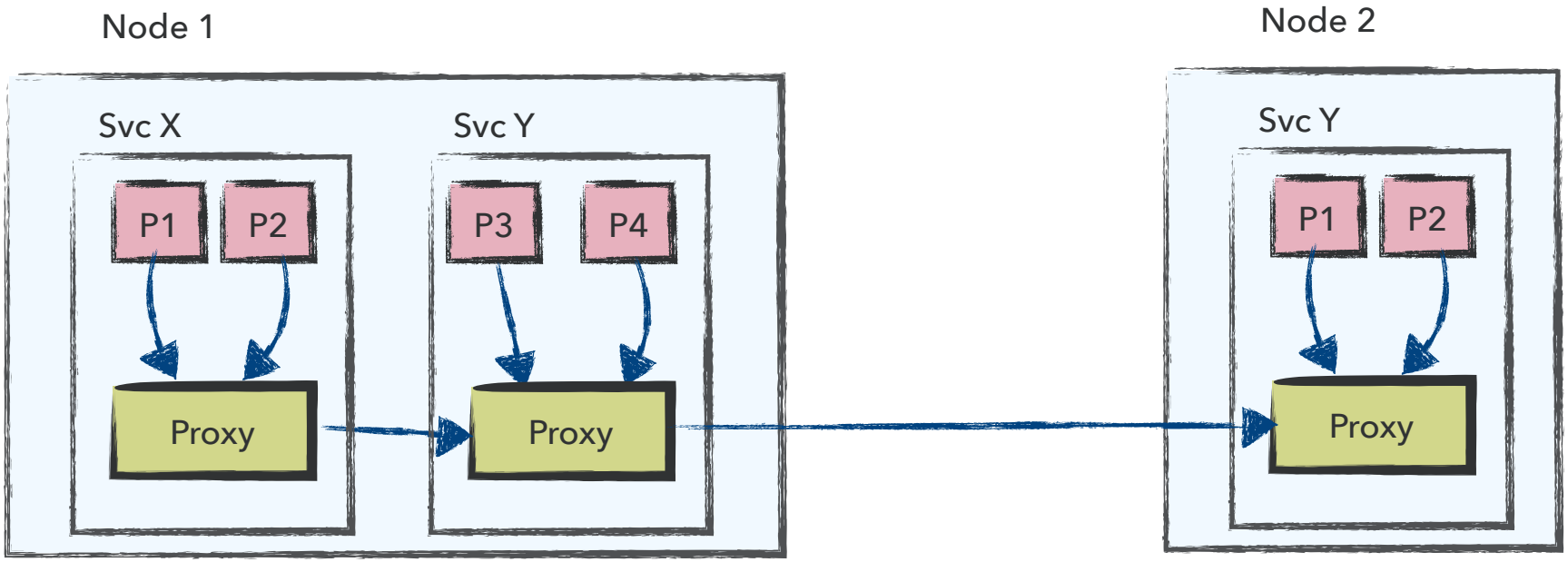
Shared proxy per service account



Shared proxy per service account



Shared proxy per service account



Shared proxy per service account

- Better resource isolation
- Still suffers from security concerns
- Presents increased operational risk
- Does not bring a lot more advantages

Sidecar advantages

- Resource consumption scale with the application
- Proxy failure is limited to a particular instance
- Maintenance is easier and less risky
- The security boundary is very clear and benefits from the very isolation principles that containers are here to bring

Popular folklore

Popular folklore

- Sidecars are slow, bulky and waste resources
- Introduce extra latency
- The service mesh will soon live in the kernel
- Multi-tenant proxy are the way forward
- You are speaking you own book here

Q&A