databricks

# From Load Balancer Sprawl to Multicluster Service Discovery

## Extending pod-level discovery and client-side routing across Kubernetes clusters

Vincent Cheng, Gaurav Nanda



KubeCon + CloudNativeCon North America 2026

# The Databricks platform

## Make data and AI accessible across the organization

### What we do

- Unified platform for data + AI applications
- Creators of Apache Spark, Delta Lake, MLflow and Unity Catalog













### Growth

- >80% YoY growth
- >$7B revenue run rate
- 20,000+ customers
- 70% of Fortune 500 companies



# Operating at global scale

## Highly distributed, latency-sensitive systems

- 3 Clouds (AWS, Azure and GCP)
- 70+ regions globally
- 2000+ clusters
- Millions of VMs launched daily

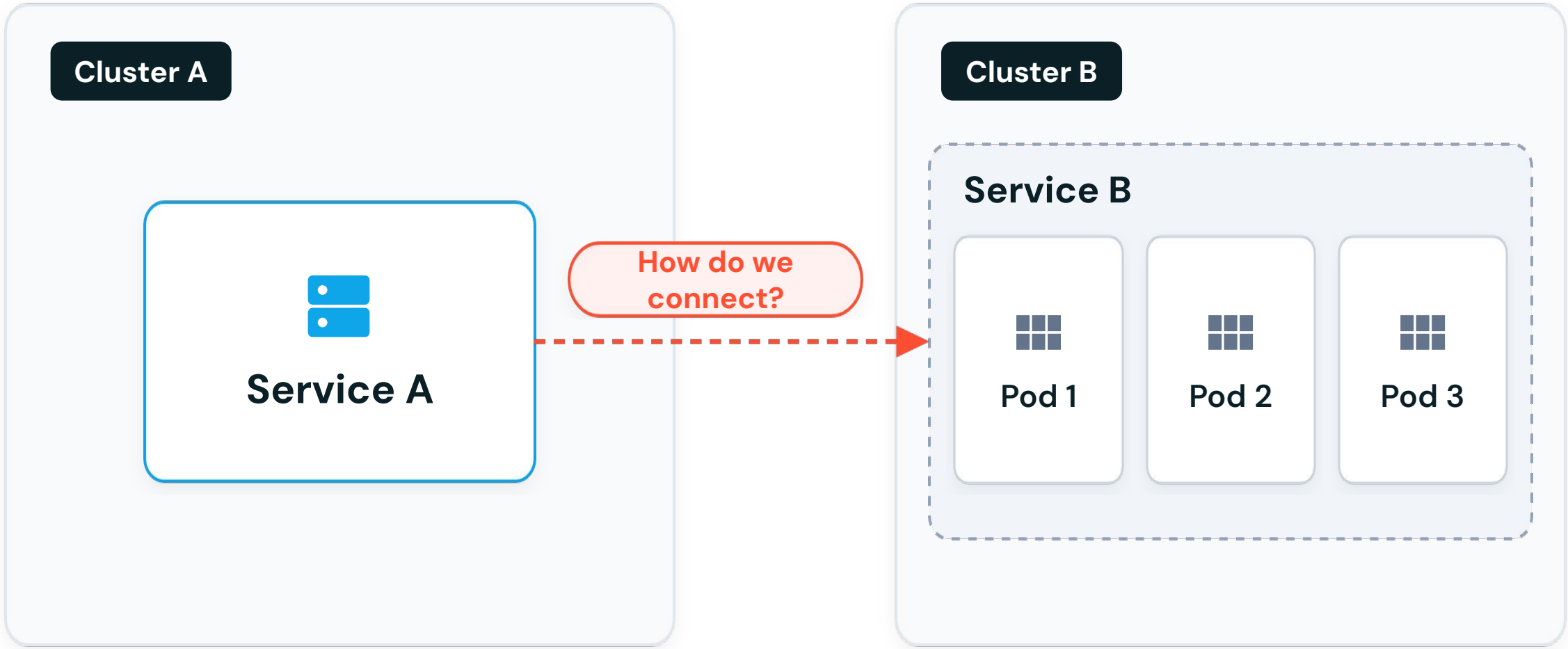


# That’s where our story begins



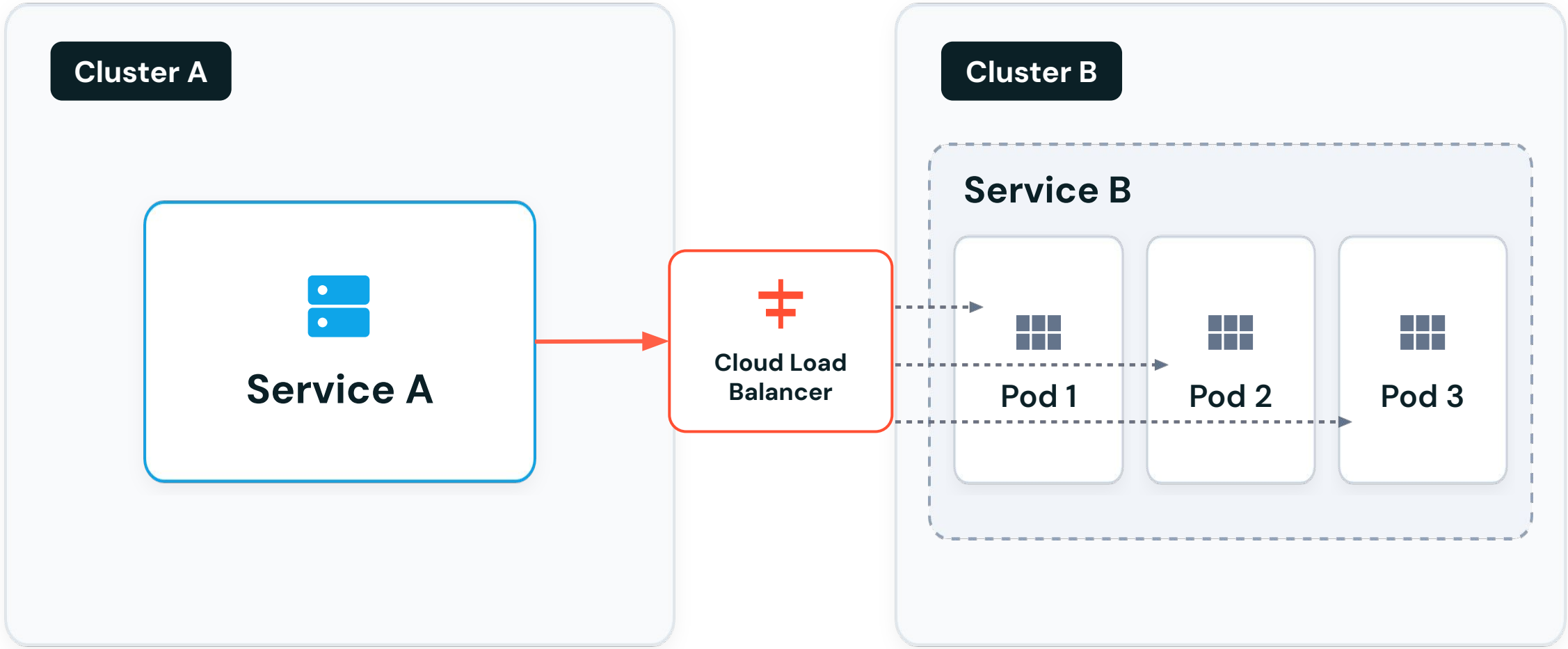
# I just need to call another service

## But it runs in another cluster



# Give the service a reachable address

## Option 1: A cloud load balancer



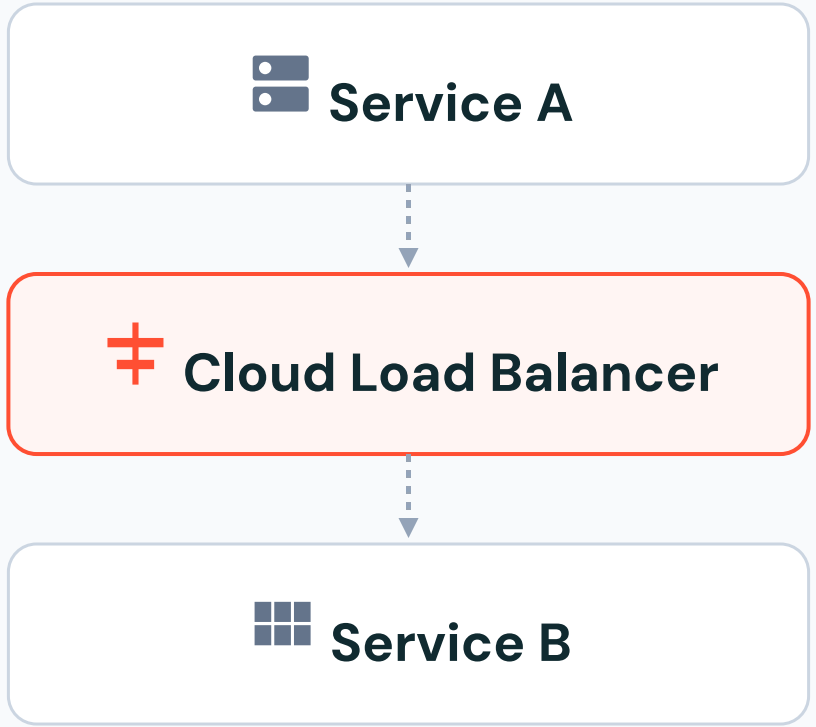
# More infrastructure to manage

## Option 1: A cloud load balancer

**01 Load balancer setup**
Cloud-specific annotations, timeouts, subnets

**02 Network access**
Allowlists and security rules

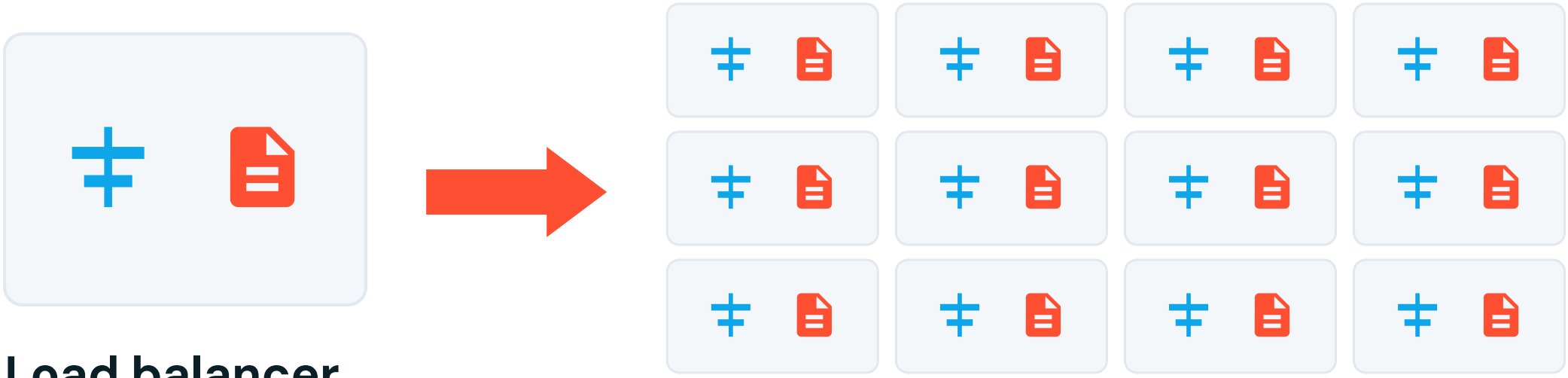
**03 Ongoing ownership**
Configuration changes and troubleshooting






# One setup becomes many

Across services. Across clusters.



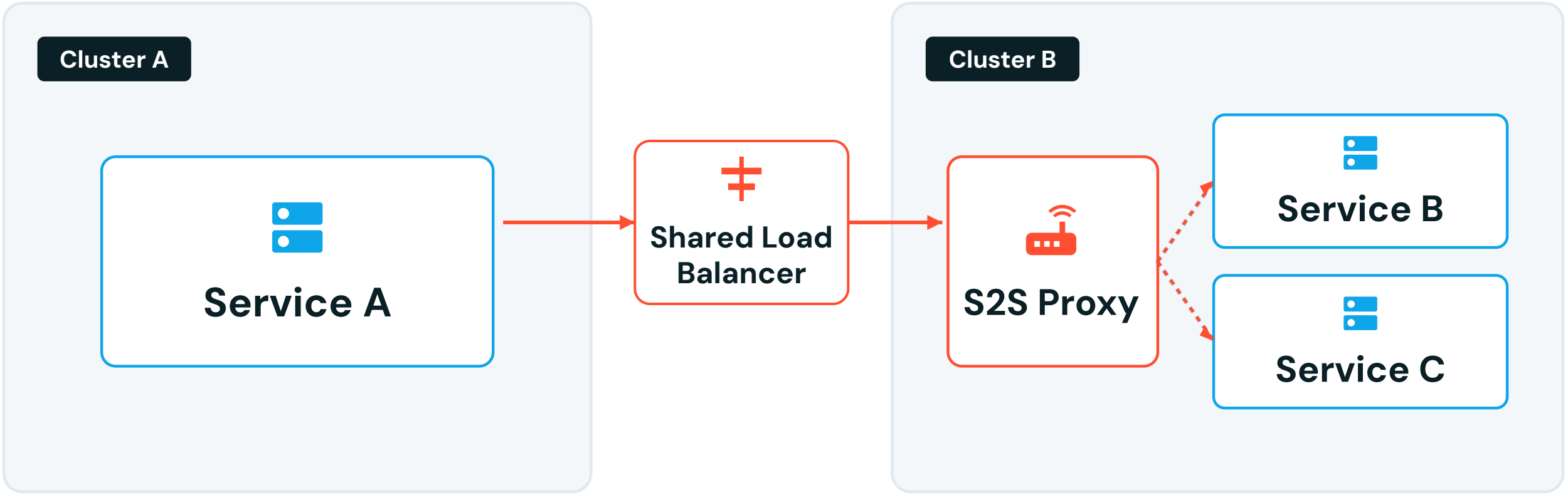
Load balancer
+ configuration

Across services and clusters.

More infrastructure • More configuration • Capacity pressure

# Another approach: share the entry point

## Option 2: A shared service-to-service proxy




Fewer per-service load balancers.

# A shared proxy came with tradeoffs

Another hop. A shared fleet. Coordinated releases.



### Extra hop

Every request passes through the proxy.



### Shared fleet

Capacity management and Shared failure risk



### Release dependency

Routing changes follow proxy rollouts.



# A shared proxy came with tradeoffs

Another hop. A shared fleet. Coordinated releases.



### Extra hop

Every request passes through the proxy.



### Shared fleet

Capacity management and Shared failure risk



### Release dependency

Routing changes follow proxy rollouts.

**Could we remove the intermediate proxy?**



INSIDE A SINGLE CLUSTER

# Clients discover Individual backend pods

Pod IP addresses + endpoint metadata

EXAMPLE ENDPOINT

**Pod 1**

**Address:** 10.20.1.7:8443

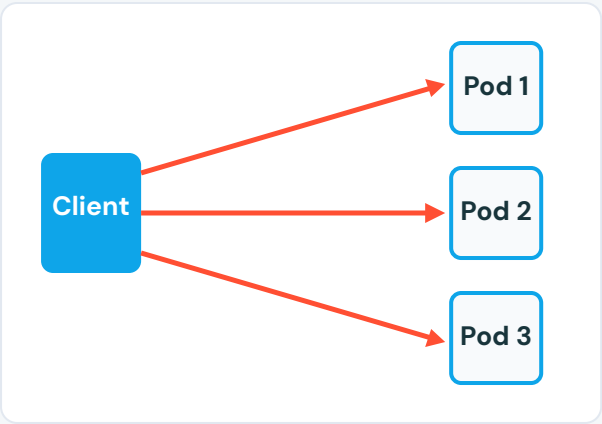
**Zone:** zone-a

**Ready:** true



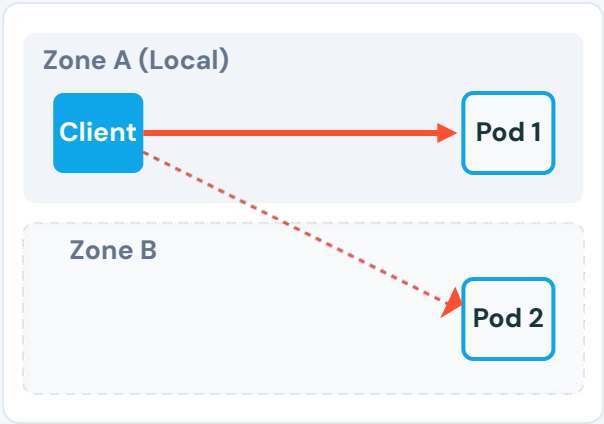
# Client-side routing inside a cluster

## Capabilities we wanted to preserve across clusters



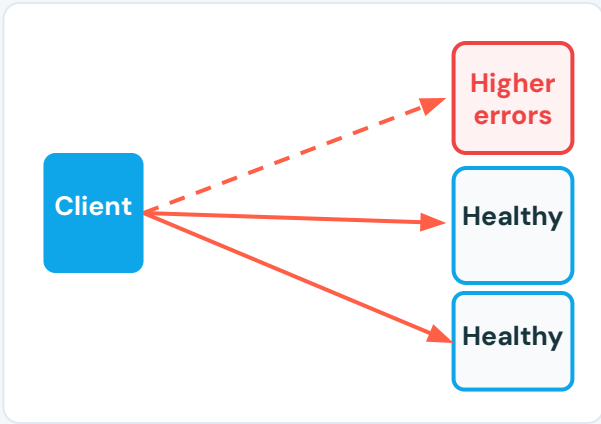

### Per-request balancing

Choose a backend for each request.




### Soft zone affinity

Prefer the local zone. Spill over when needed.



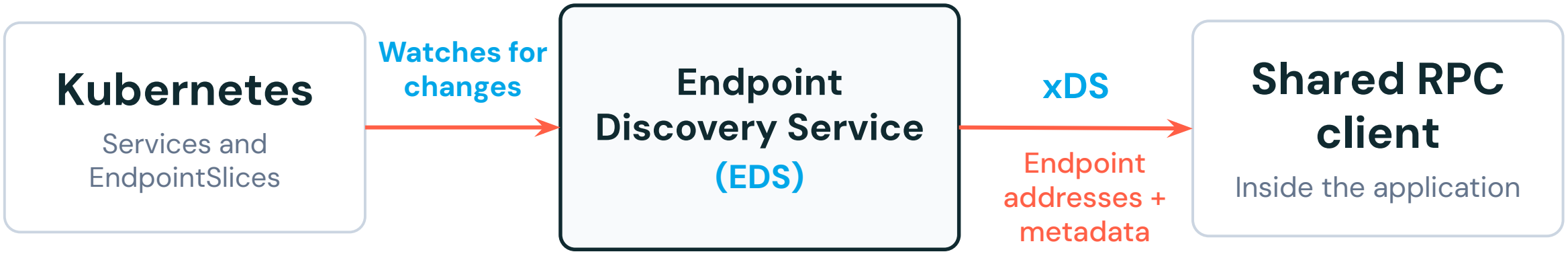

### Adaptive routing

Respond to load, errors, and latency.

# How clients receive endpoint information

## Our discovery control plane distributes updates over xDS

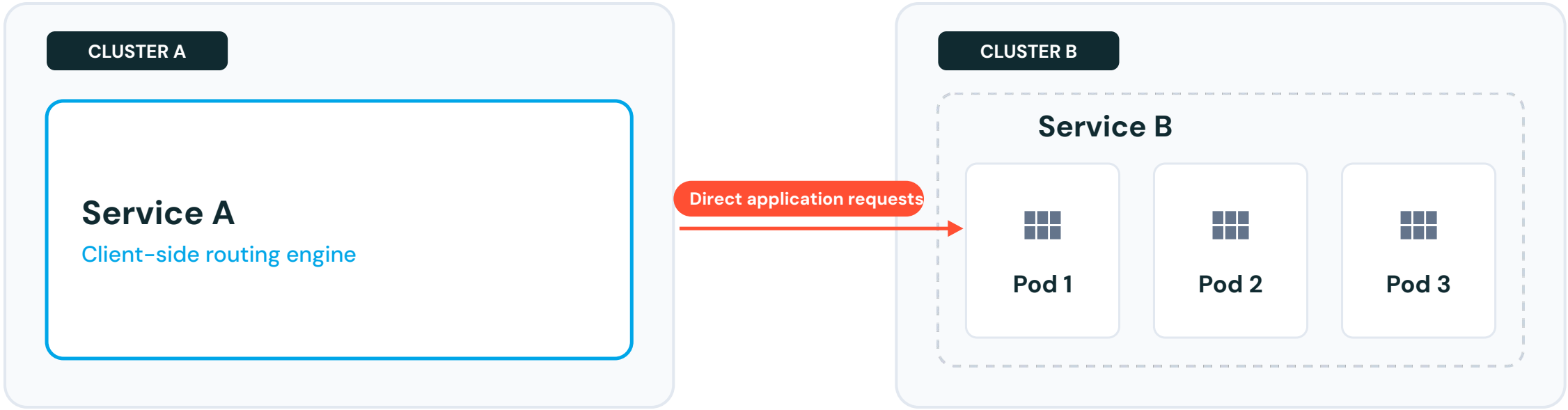
Architecture blog




**Clients subscribe and receive updates as endpoints change**

# Same routing capabilities. Across clusters.

Clients need to discover remote pods and reach them directly.




## Discovery

Which pods can I call?

Endpoints + Metadata discovery across clusters

## Connectivity

Can I reach their IPs?

Direct pod-to-pod network routability

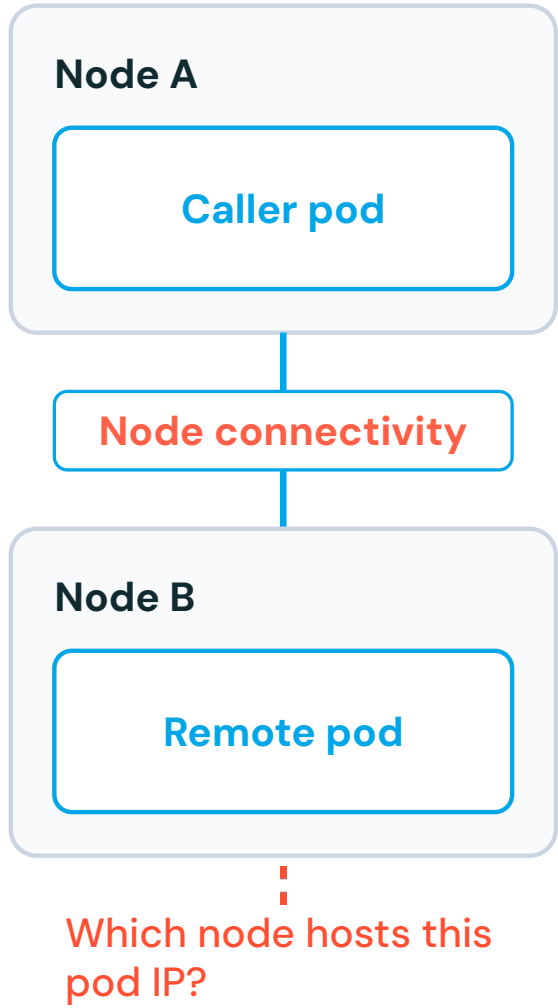
# What cross-cluster pod connectivity requires

A network path from the caller to the remote pod

## 01 Reach the nodes

Routes and access rules between cluster networks

Already available in our setup



# What cross-cluster pod connectivity requires

A network path from the caller to the remote pod

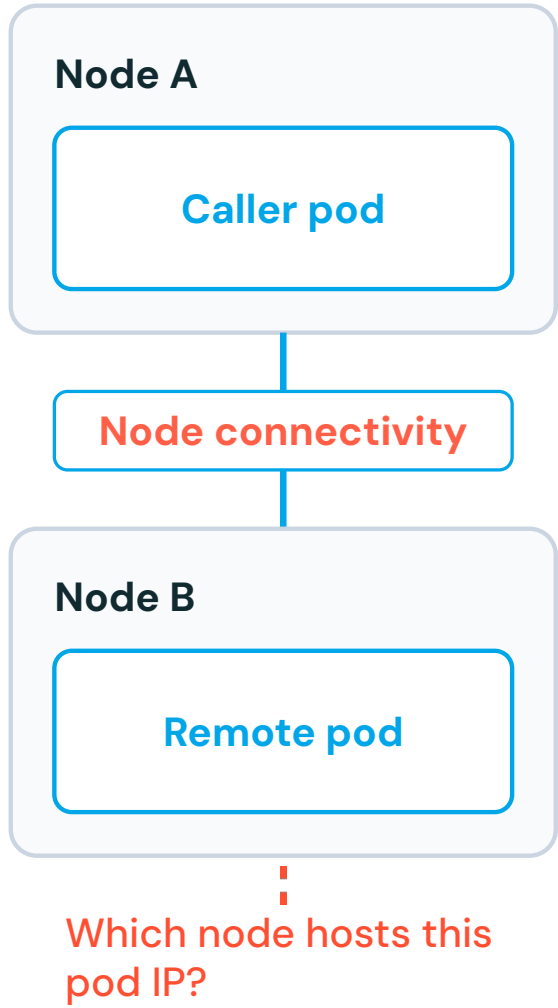
## 01 Reach the nodes

Routes and access rules between cluster networks

Already available in our setup

## 02 Allocate unique pod ranges

Non-overlapping pod addresses across connected clusters



# What cross-cluster pod connectivity requires

A network path from the caller to the remote pod

## 01 Reach the nodes

Routes and access rules between cluster networks

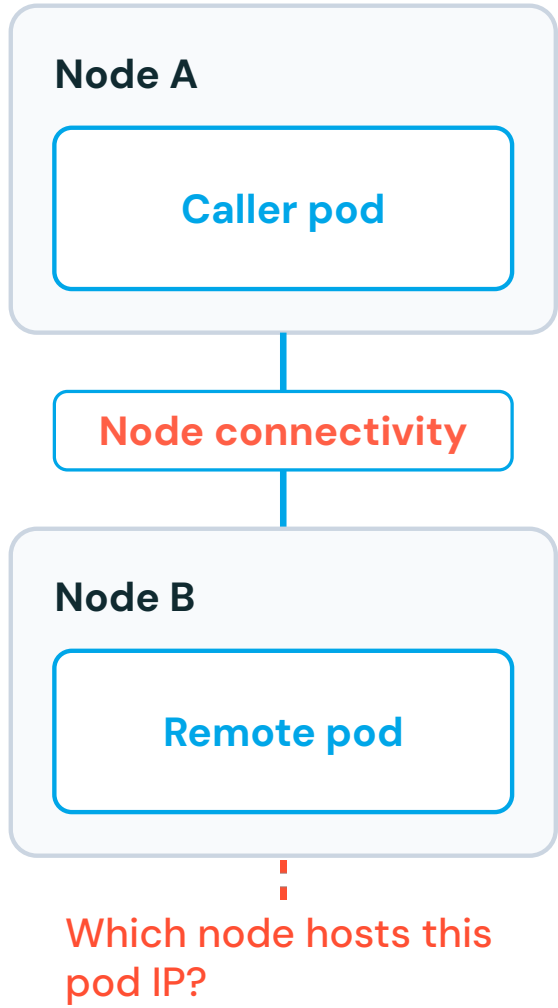
Already available in our setup

## 02 Allocate unique pod ranges

Non-overlapping pod addresses across connected clusters

## 03 Forward traffic to remote pods

Map the destination pod IP to the right node



# What we wanted from a solution

Cross-cluster calls should work like in-cluster calls.



## No large re-architecture

Build on our existing setup.



## No proxies in between

No sidecars, no shared proxy.



## Works with our client libraries

Per-request balancing and zone affinity keep working.



# What we wanted from a solution

Cross-cluster calls should work like in-cluster calls.

### No large re-architecture

Build on our existing setup.

### No proxies in between

No sidecars, no shared proxy.

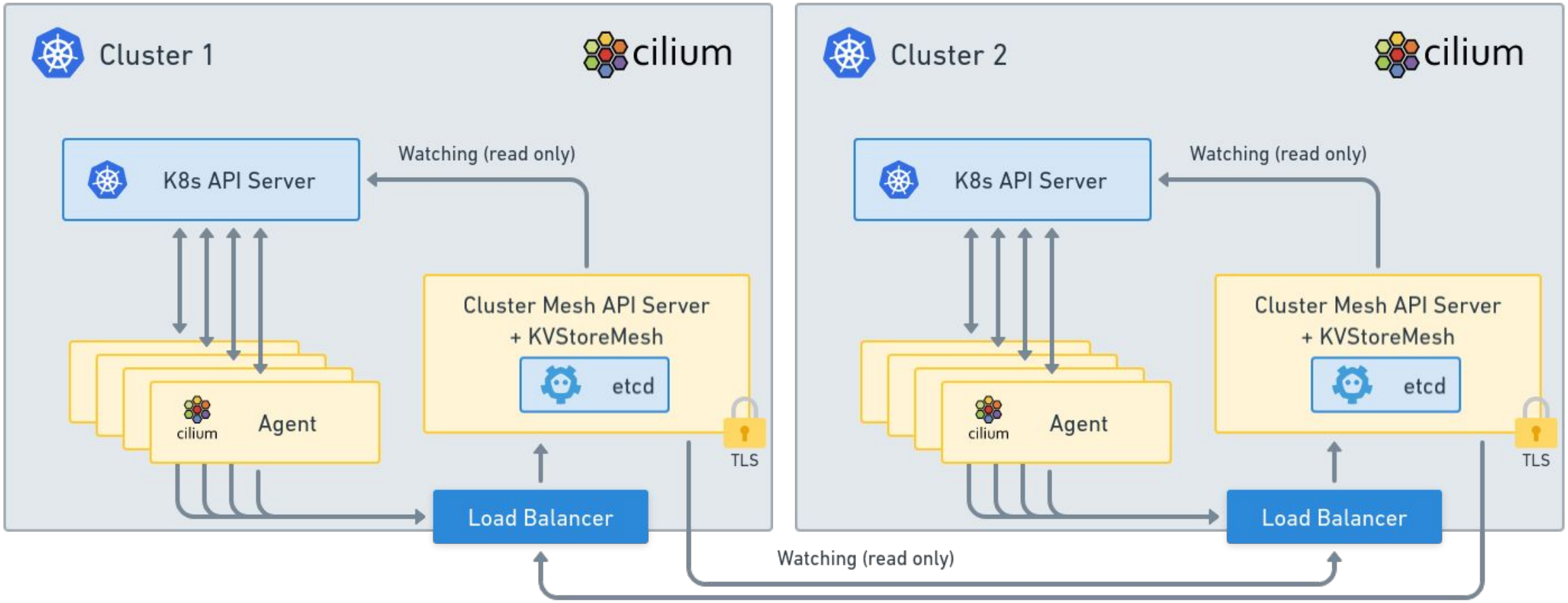
### Works with our client libraries

Per-request balancing and zone affinity keep working.



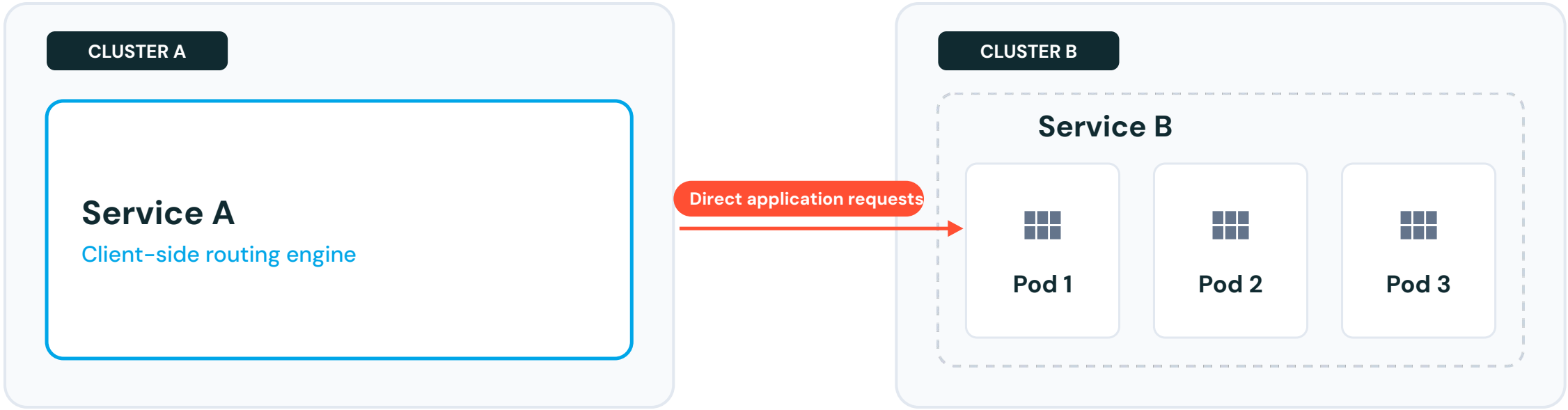
# What is Cilium ClusterMesh?

Connects Kubernetes clusters running Cilium.



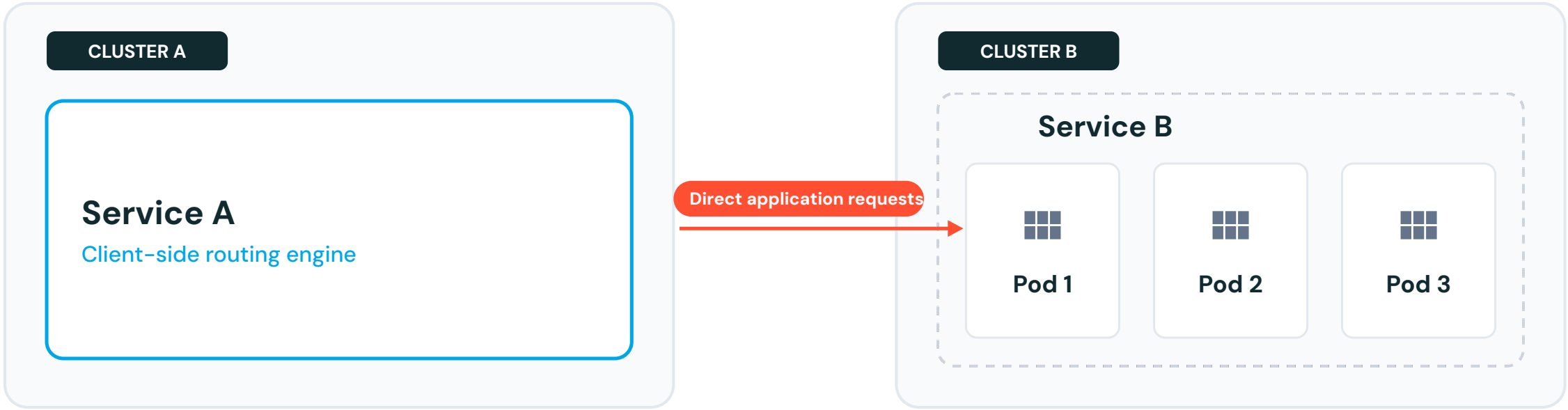
# Making pod IPs reachable across clusters.

Cilium ClusterMesh provides connectivity within a region.



# Making pod IPs reachable across clusters.

Cilium ClusterMesh provides connectivity within a region.




**Reachability established. How does the client discover those pods?**

# A Cilium-native option: global services

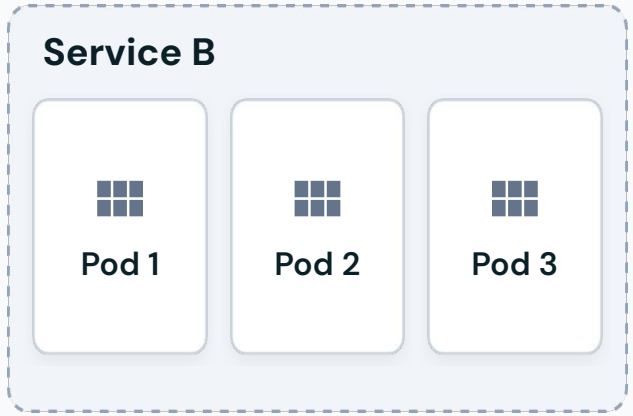
Annotate the Service in every cluster.

CLUSTER A

```
kind: Service
metadata:
  name: service-b
  namespace: bar
  annotations:
    service.cilium.io/global: "true"
```

CLUSTER B

```
kind: Service
metadata:
  name: service-b
  namespace: bar
  annotations:
    service.cilium.io/global: "true"
```



# A Cilium-native option: global services

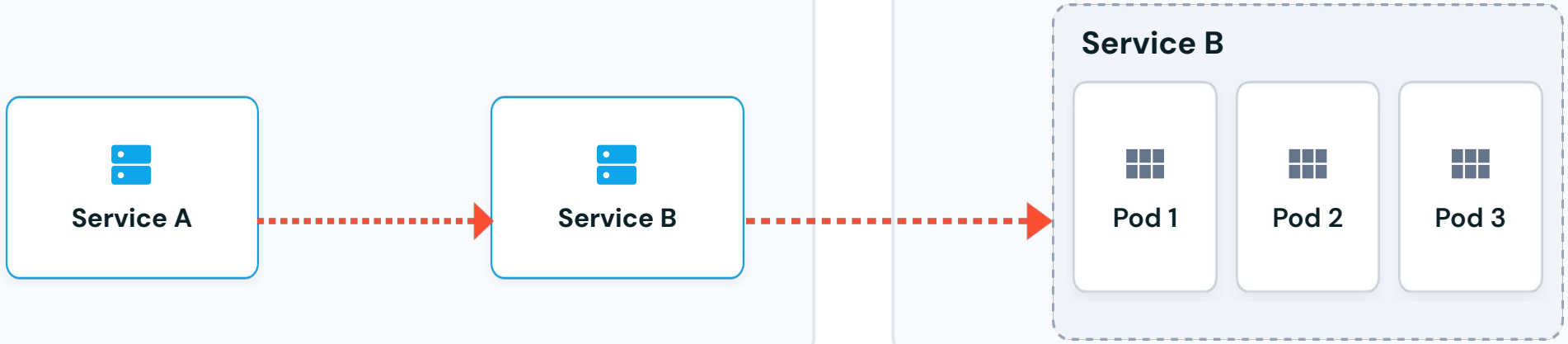
Annotate the Service in every cluster.

CLUSTER A

```
kind: Service
metadata:
  name: service-b
  namespace: bar
  annotations:
    service.cilium.io/global: "true"
```

CLUSTER B

```
kind: Service
metadata:
  name: service-b
  namespace: bar
  annotations:
    service.cilium.io/global: "true"
```



# Let’s revisit our requirements

Naive global services didn’t fulfill all of our goals



## No large re-architecture

Build on our existing setup.



## No proxies in between

No sidecars, no shared proxy.



## Works with our client libraries

Per-request balancing and zone affinity keep working.



# Let’s revisit our requirements

Naive global services didn’t fulfill all of our goals



## No large re-architecture

Build on our existing setup.



## No proxies in between

No sidecars, no shared proxy.



## Works with our client libraries

Per-request balancing and zone affinity keep working.

**Could EDS do for remote traffic what it does for local traffic?**



# Extending EDS to support cross-cluster discovery

First approach: just read what ClusterMesh is already propagating

KEY

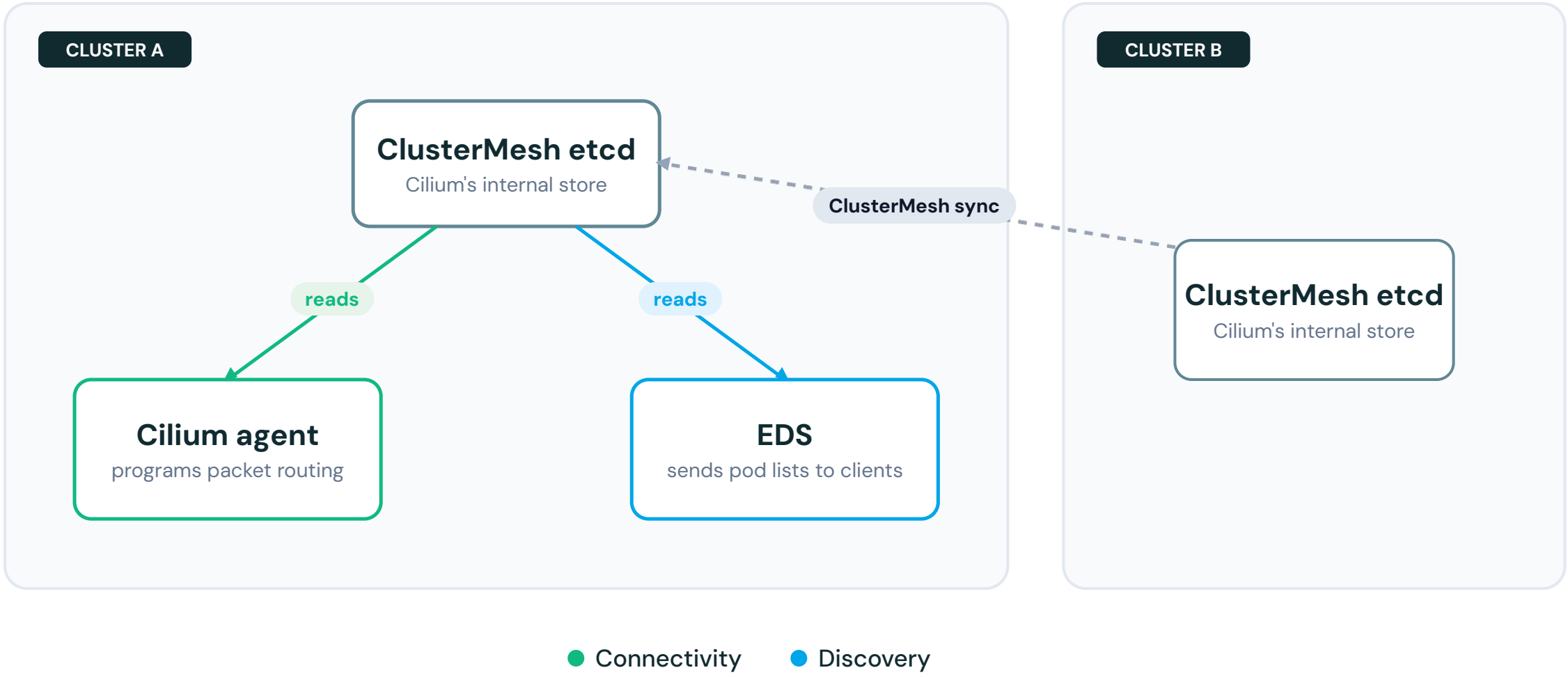
```
cilium/cache/services/v1/<cluster>/<namespace>/<service>
```

VALUE (ClusterService)

```
{
"cluster": "cluster-b",
"namespace": "bar",
"name": "service-b",
"backends": {
  "10.2.1.15": { "http": { "Protocol": "TCP", "Port": 8080 } },
  "10.2.3.22": { "http": { "Protocol": "TCP", "Port": 8080 } },
  "10.2.4.9": { "http": { "Protocol": "TCP", "Port": 8080 } }
},
...
```

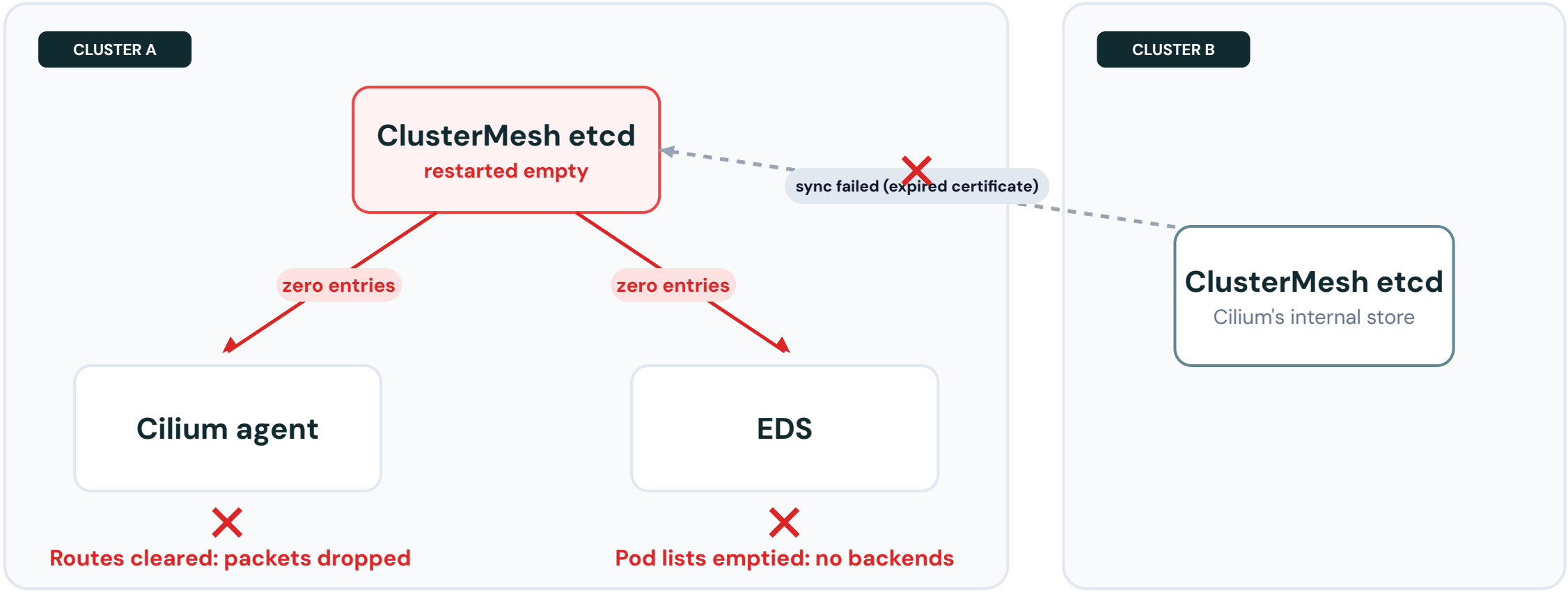
# Extending EDS to support cross-cluster discovery

EDS as just another subscriber to the ClusterMesh etcd



# When the store came back stale or empty, both layers broke

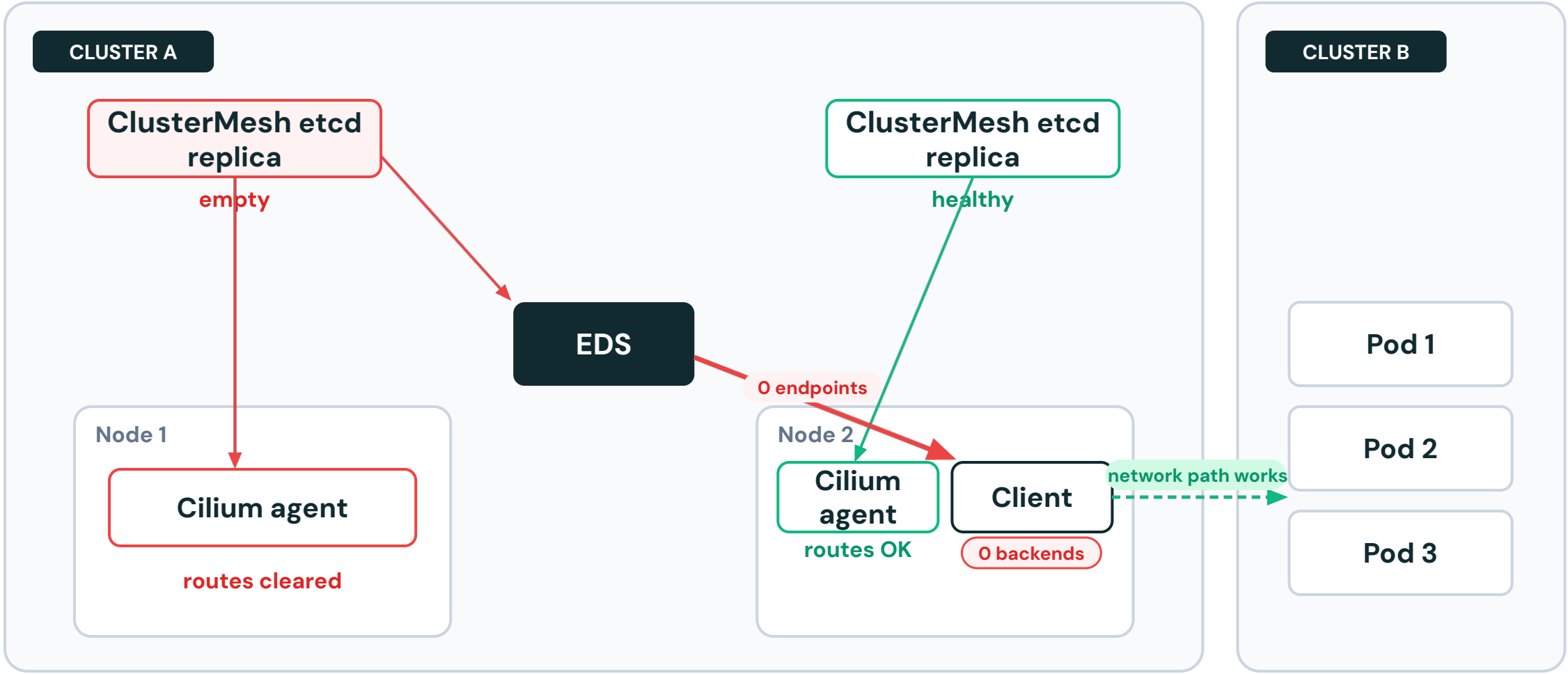
One fault. Two outages.




**Both layers trusted an empty answer.**

# A partial failure, amplified

One bad etcd replica broke some routes. EDS spread its empty answer to every client it served.




**The network could reach Cluster B. EDS told the client there was nothing there.**

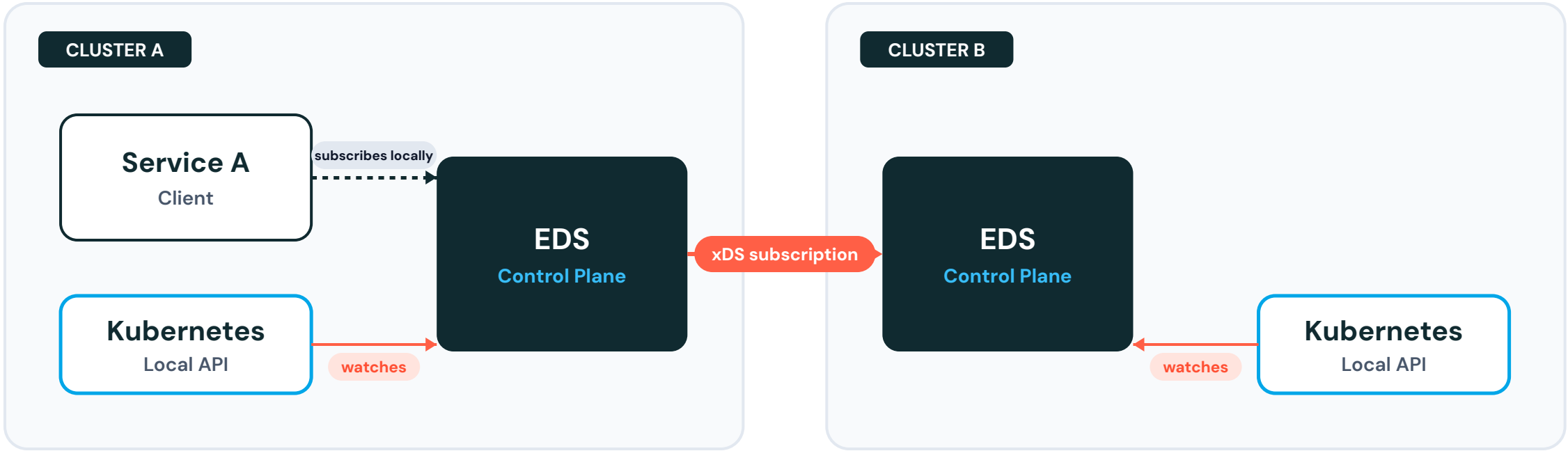
Let Cilium handle connectivity.

Let EDS own discovery itself.



# Extending EDS across clusters

## A remote EDS is just another xDS subscriber.



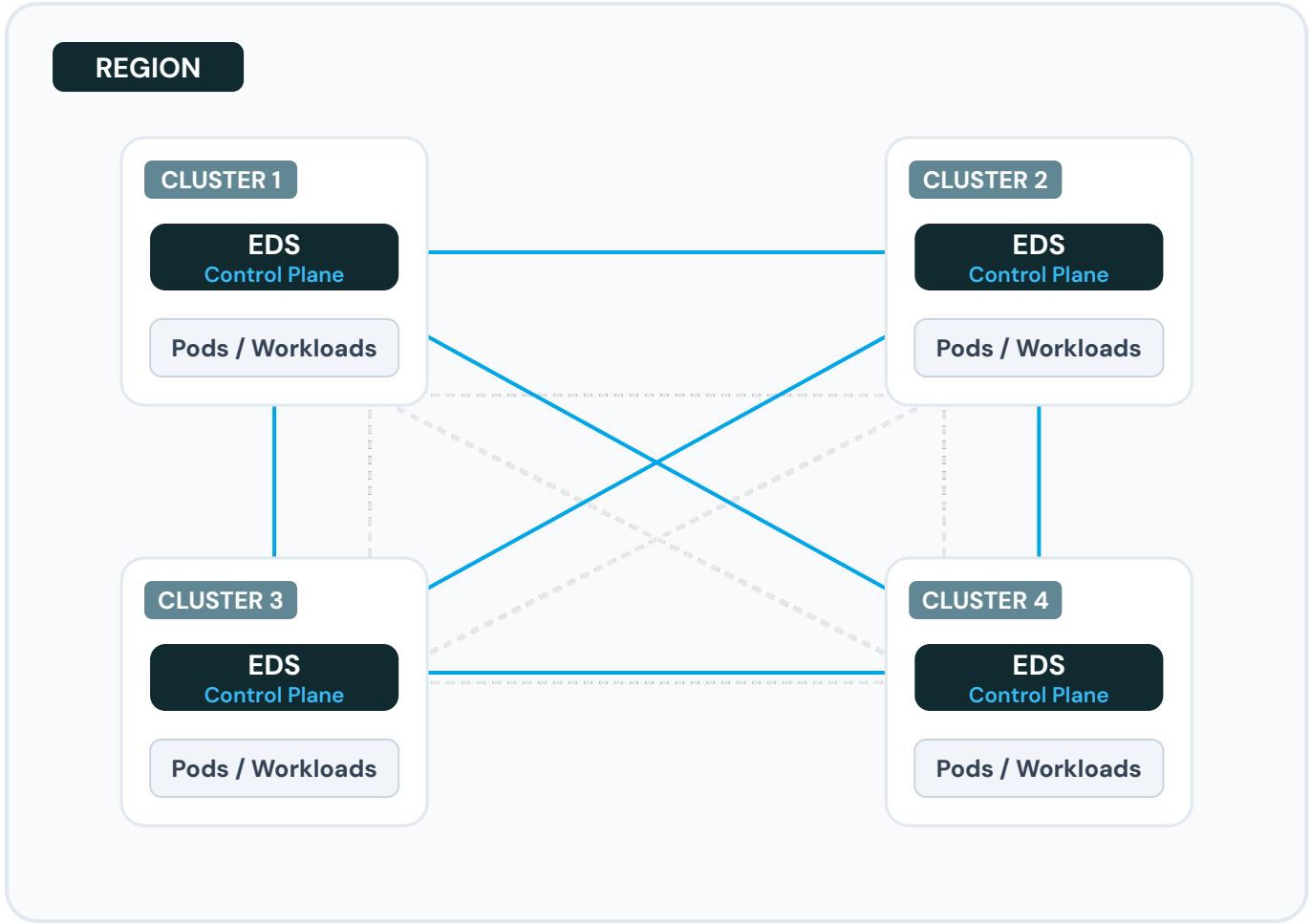

Each EDS subscribes to its peers

Same xDS protocol our clients use

Clients still talk only to local EDS

# A regional mesh of EDS peers

## Discovery on top of connectivity.




**Discovery** | EDS ↔ EDS over xDS

**Connectivity** | Cilium ClusterMesh

### Full Mesh Discovery

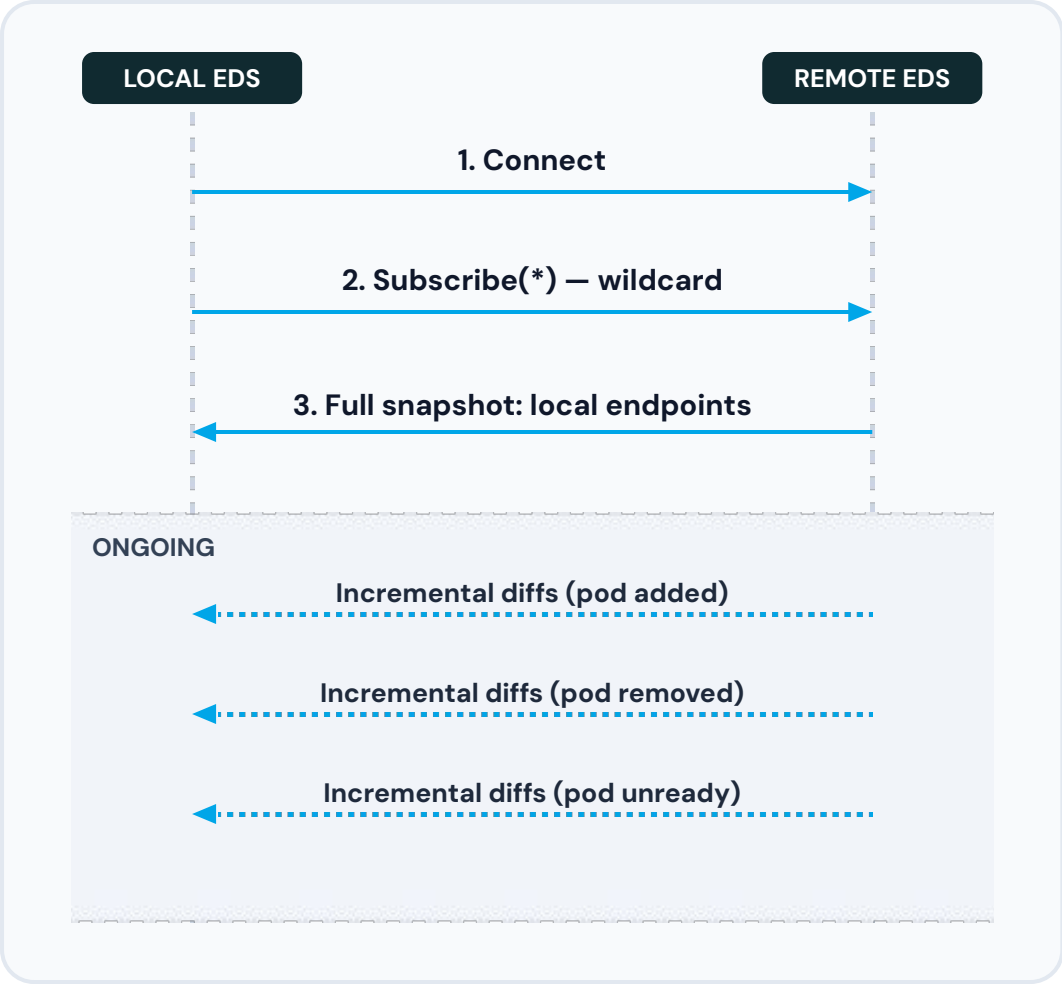
Every EDS peers with every other EDS in the region

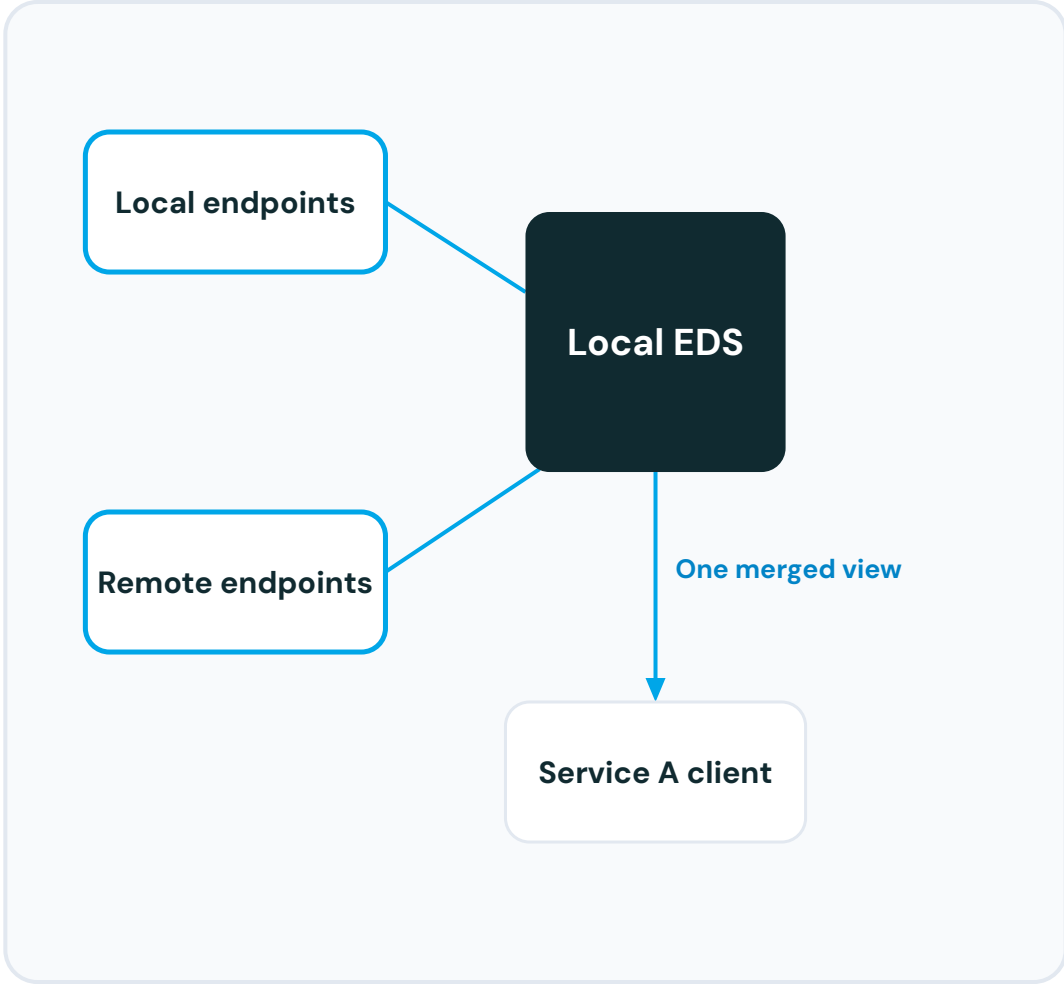
### ClusterMesh Scope

Meshes at the same scope as ClusterMesh

# How the subscriptions work

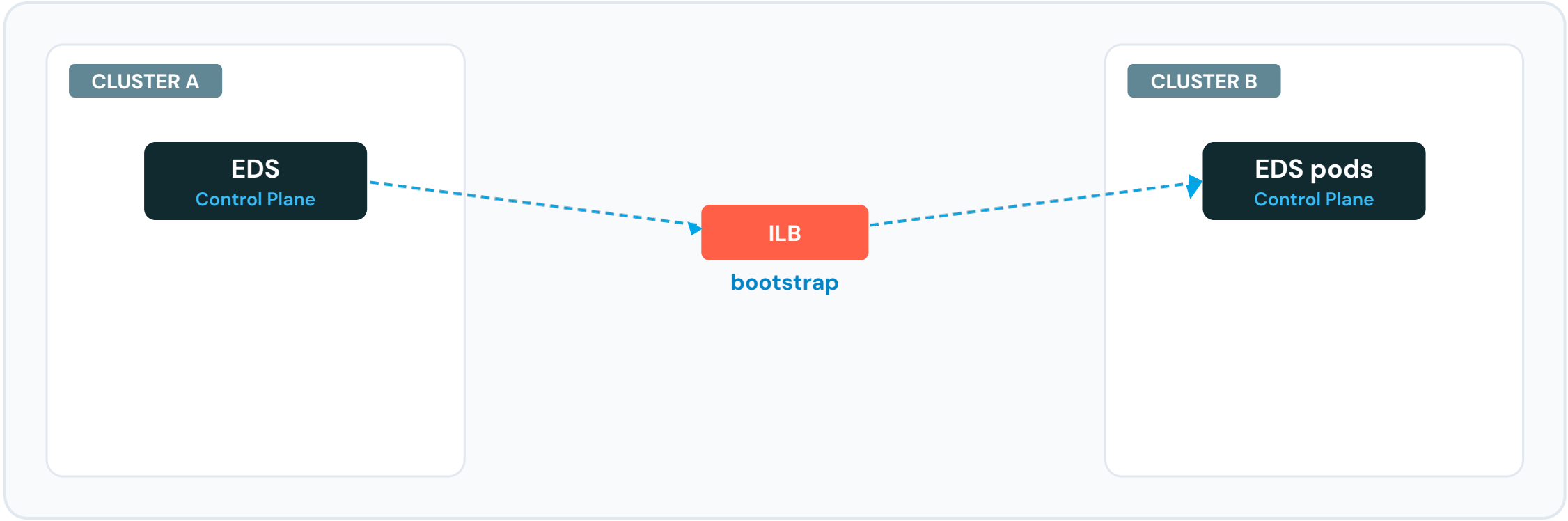
## Full state once, then only changes.



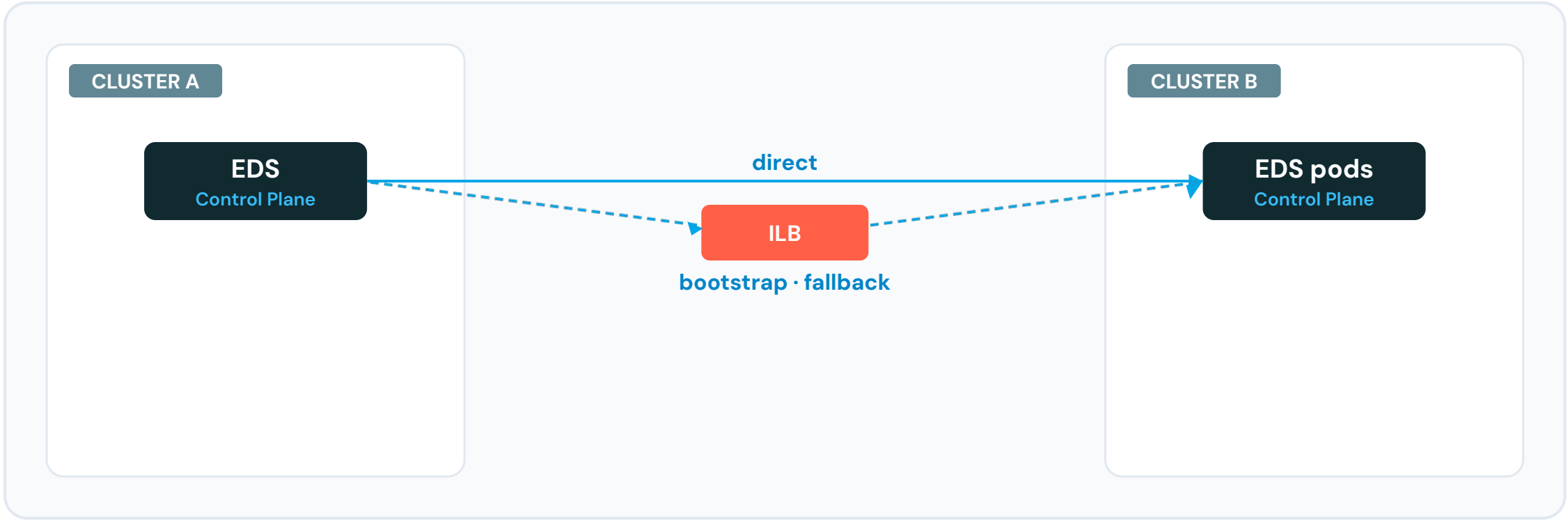
# How EDS finds its peers

## Start with a load balancer



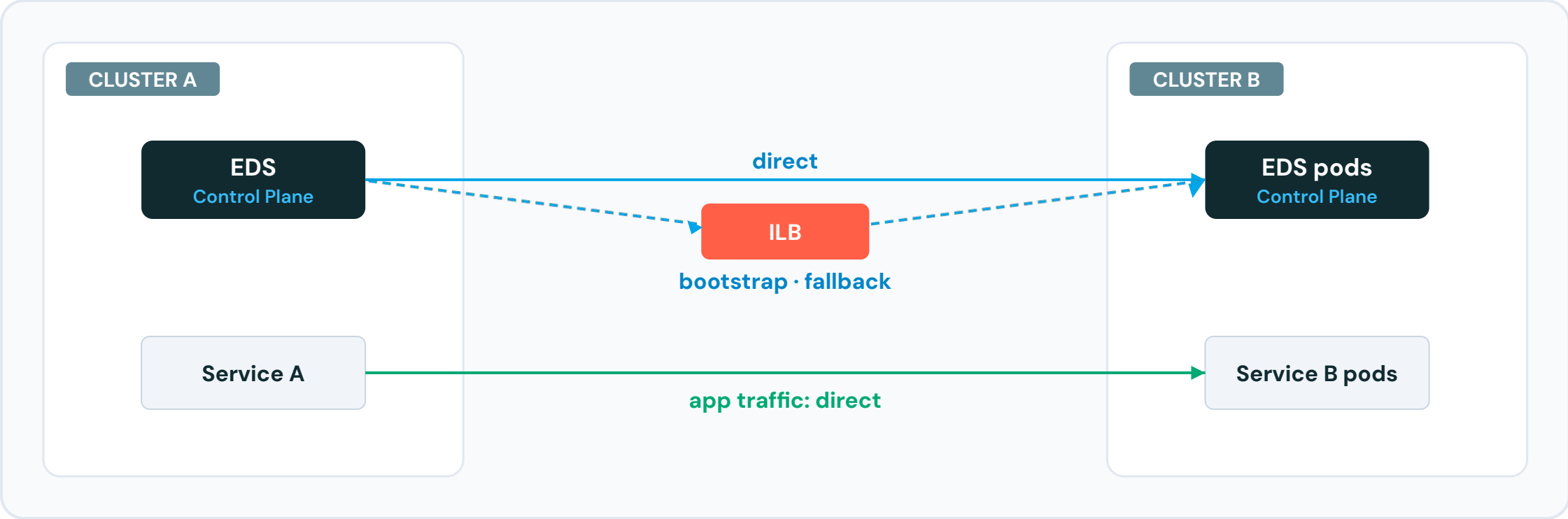
# How EDS finds its peers

## Direct connectivity under steady state



# How EDS finds its peers

A load balancer for discovery, not for traffic.




**One ILB per cluster: just for discovery**

# How far it scales

## Load test of a regional EDS mesh.

**5M**

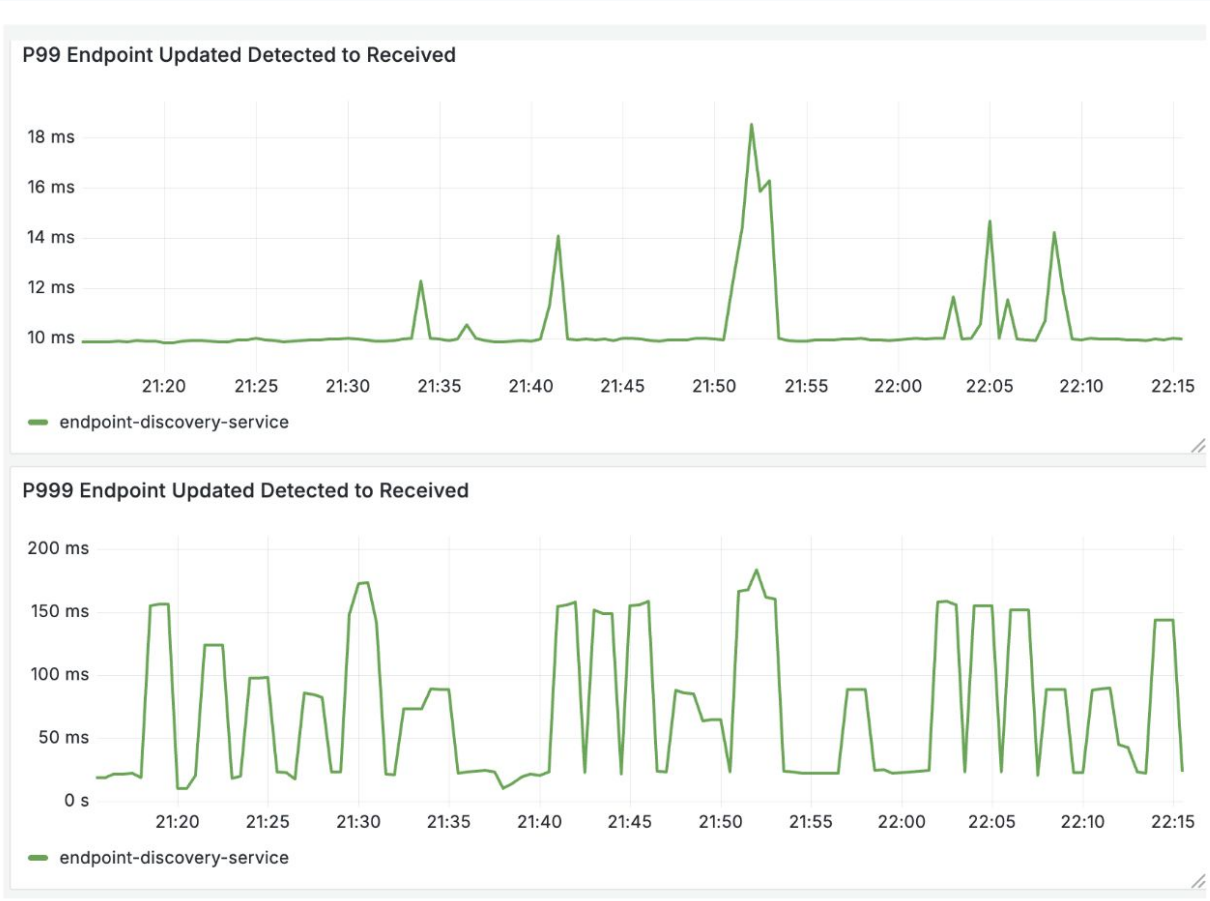
endpoints in one region

**100**

clusters meshed
(50K endpoints each)

**500/s**

endpoint changes
across the region



# How far it scales

## Load test of a regional EDS mesh.

**10M**

endpoints in one region

**200**

clusters meshed
(50K endpoints each)

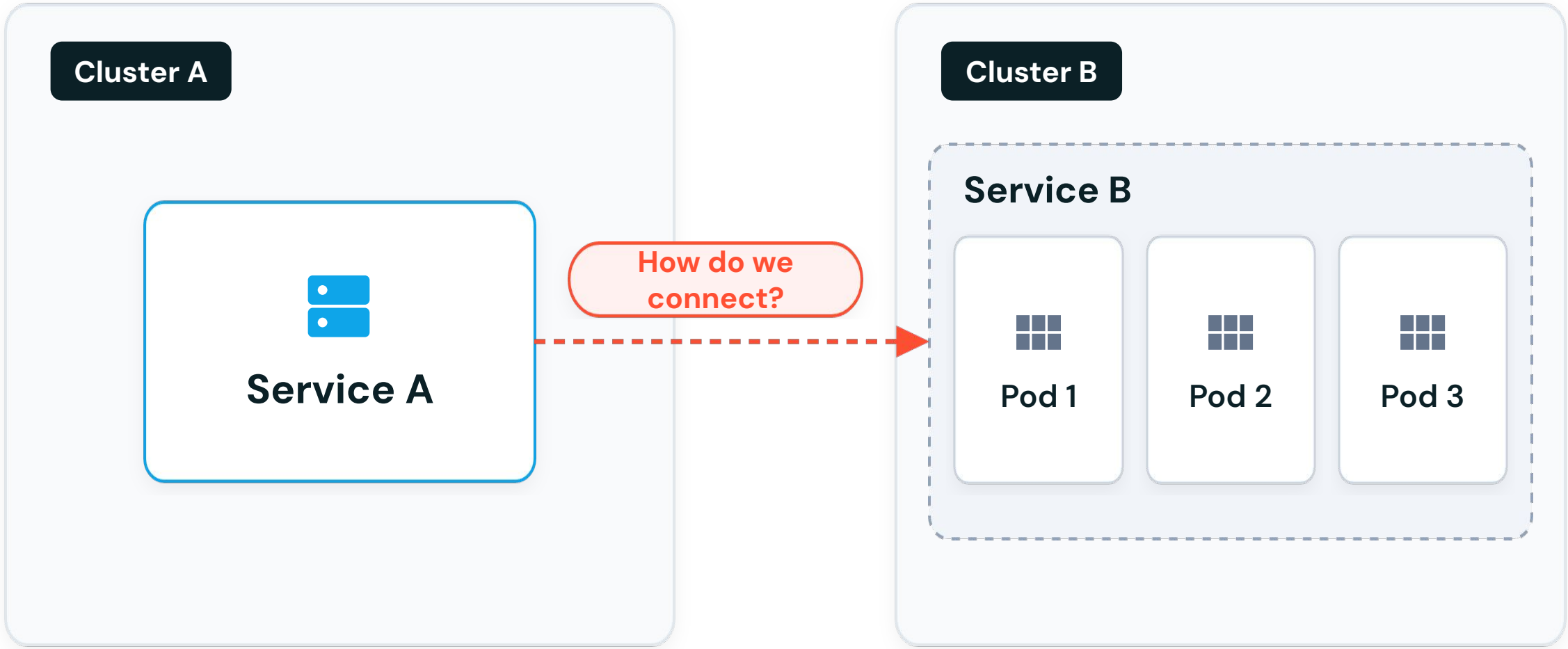
**1,000/s**

endpoint changes
across the region



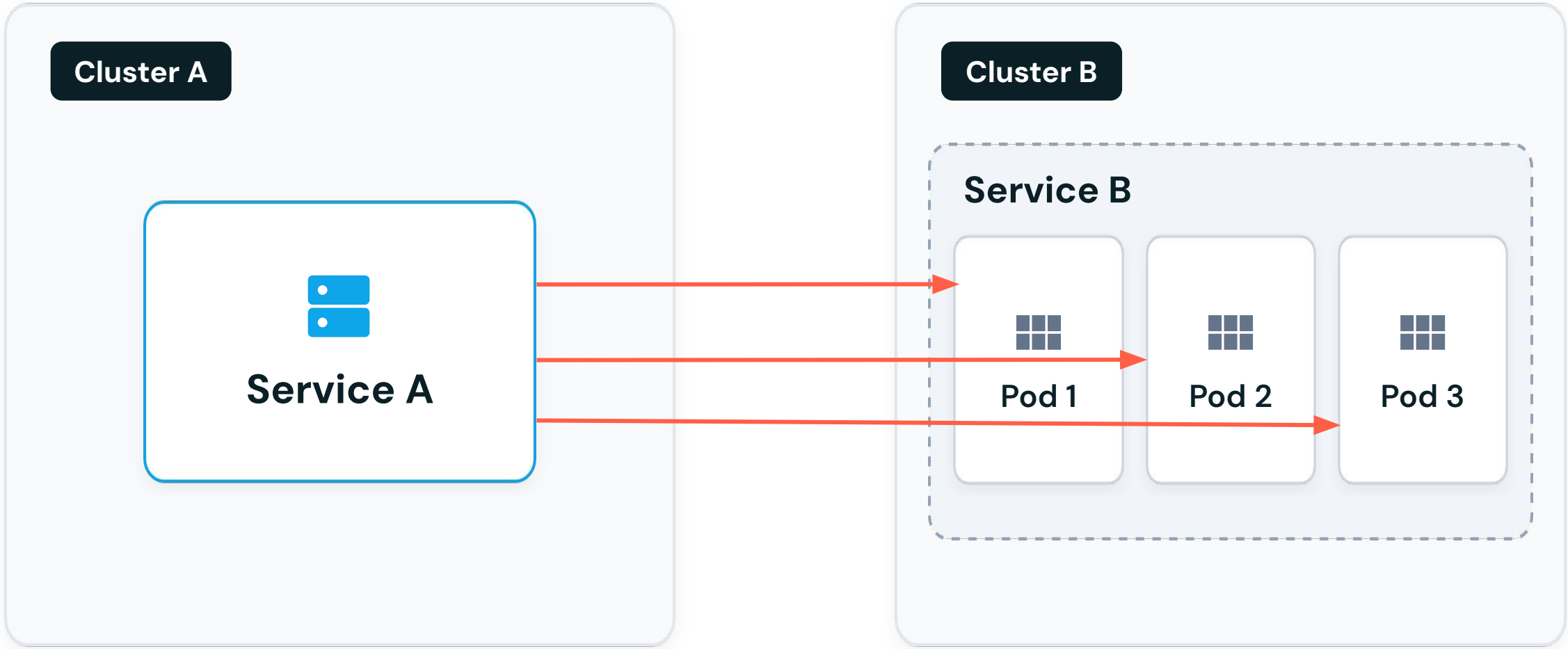
# I just need to call another service

## But it runs in another cluster



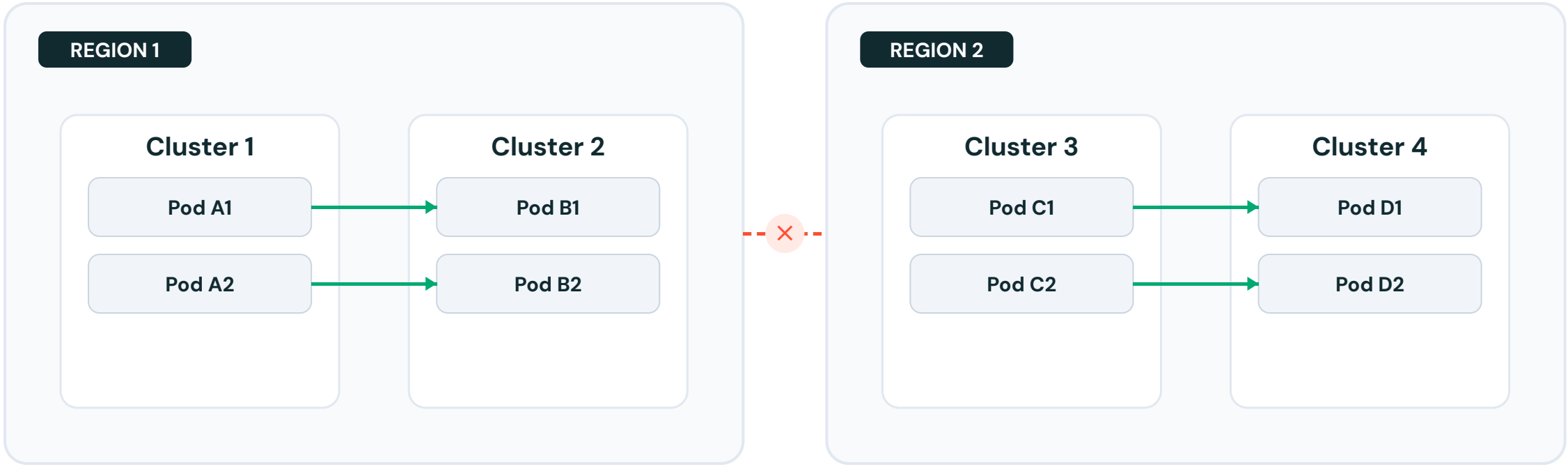
# I just need to call another service

Just name the service and send the request.



# What's next

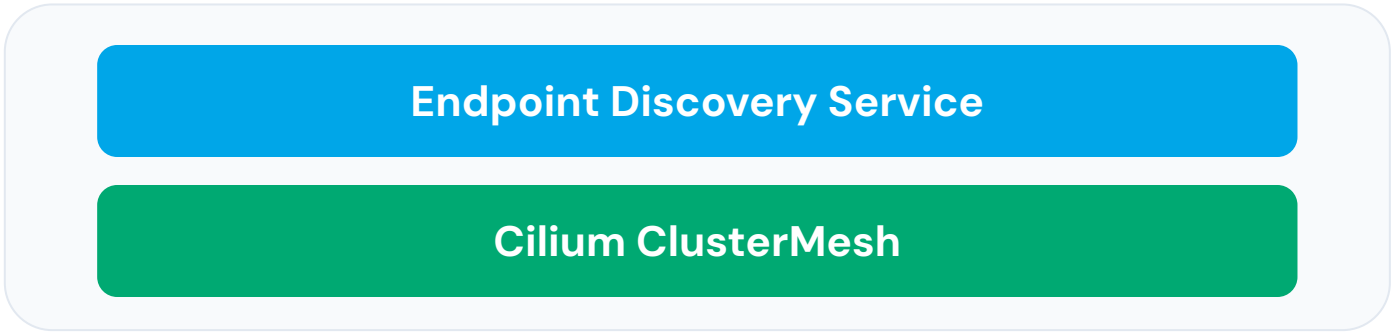
## Beyond the region.




Our meshes stop at the region

Next: pod-level connectivity across regions

# Solve connectivity and discovery separately, and cross-cluster calls can work like local ones.






# Thank You

## Questions? / Q&A Session

KubeCon NA 2026

