databricks

From Load Balancer Sprawl to Multicluster Service Discovery

Direct 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



01

The cross-cluster challenge

Why we looked beyond load balancers and shared proxies

02

Direct routing inside a cluster

How clients discover pods and choose where to send requests

03

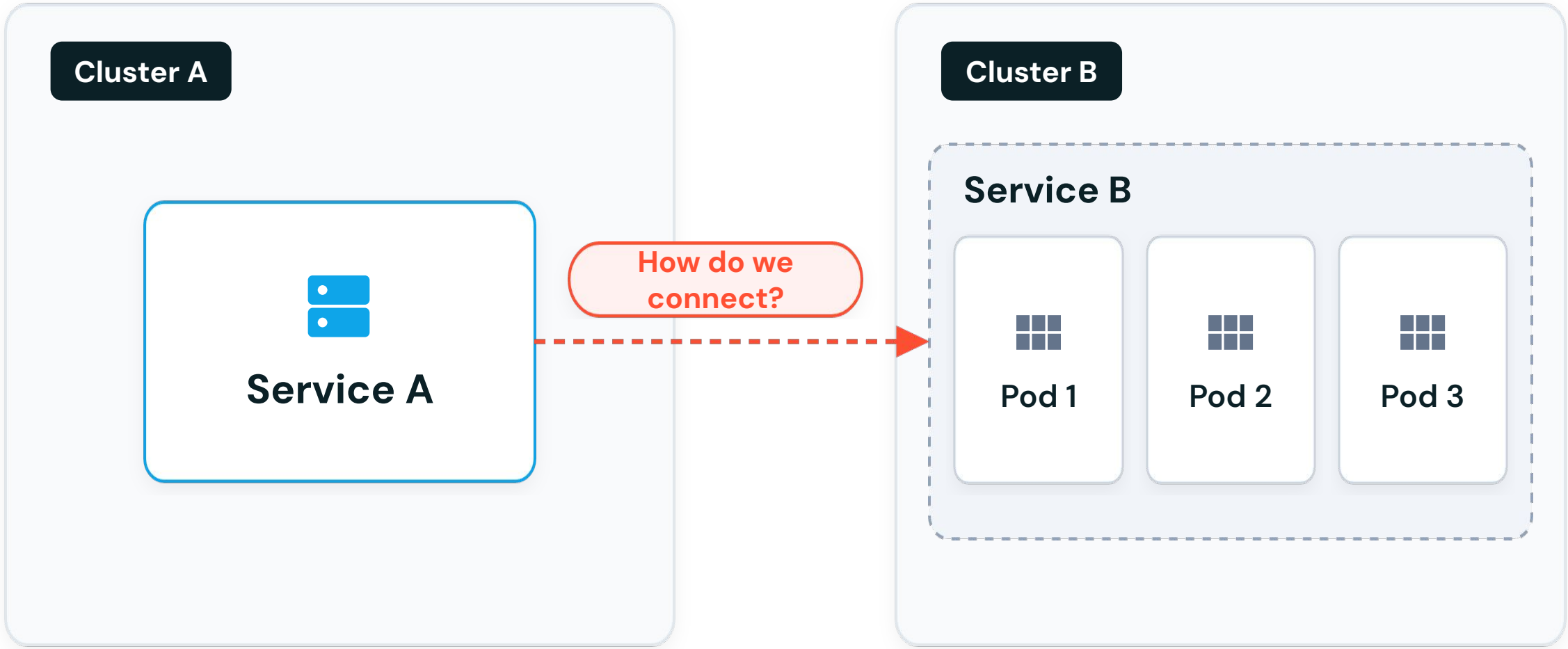
Taking direct routing across clusters

Making remote pods reachable and discoverable



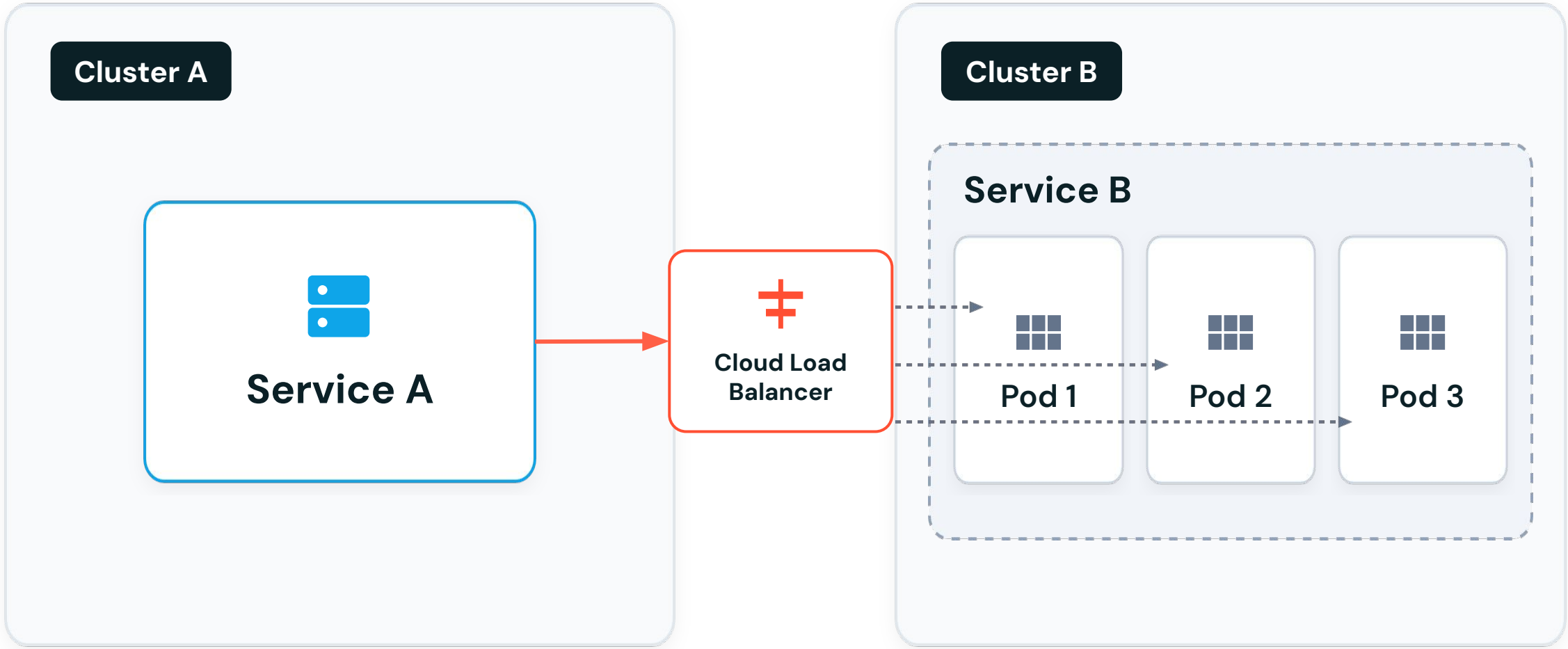
I just need to call another service

But it runs in another cluster



Give the service a reachable address

Option 1: A cloud load balancer



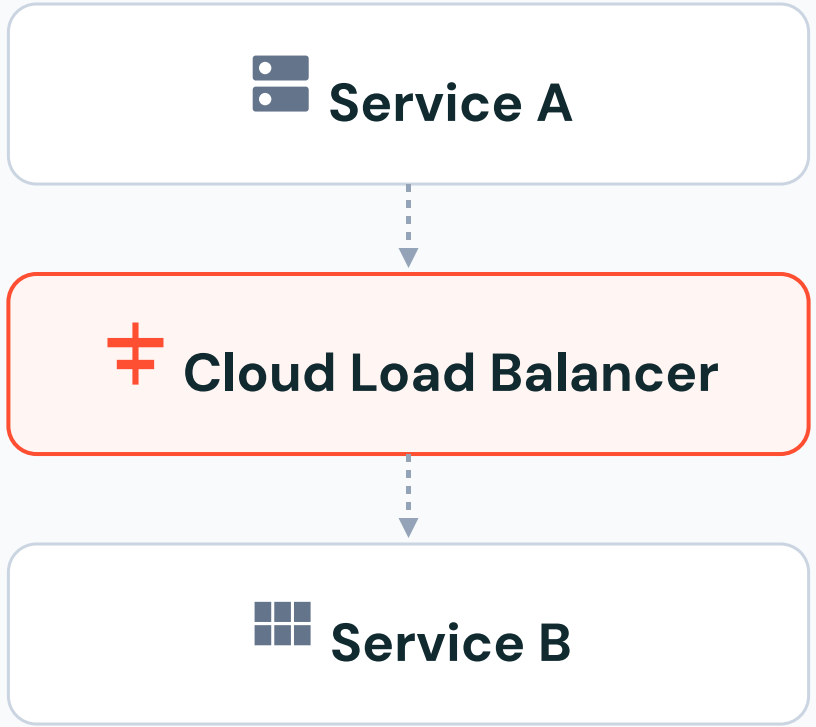
More infrastructure to manage

Option 1: A cloud load balancer

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

02 Network access
Allowlists and security rules

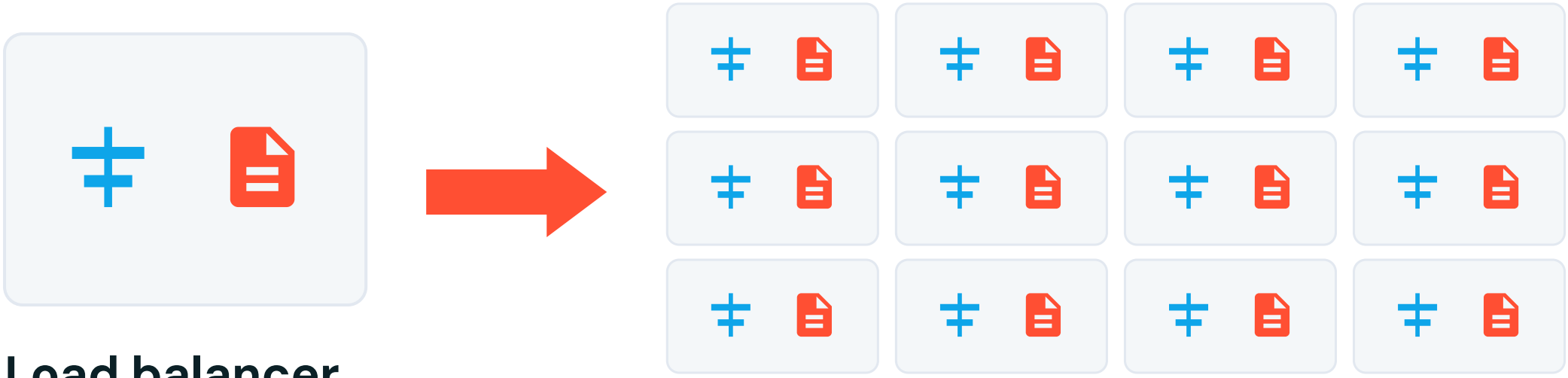
03 Ongoing ownership
Configuration changes and troubleshooting





One setup becomes many

Across services. Across clusters.



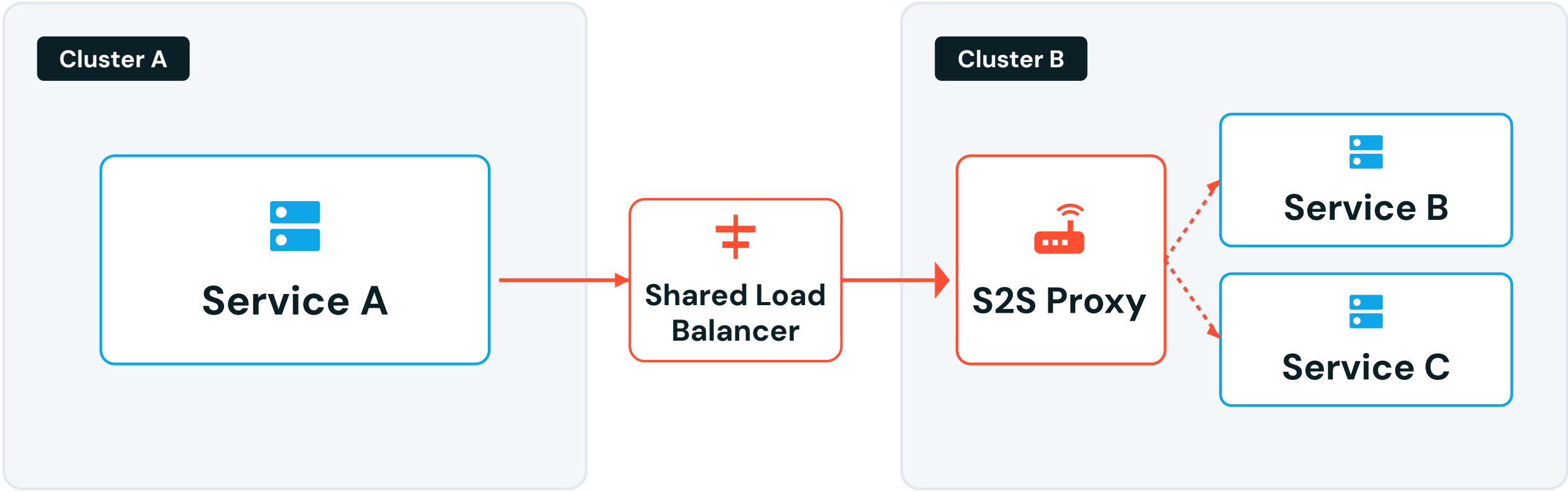
Load balancer
+ configuration

Across services and clusters.

More Load Balancers • 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. Harder to troubleshoot.



Extra hop

Every request passes through the proxy.



Shared fleet

Capacity management and Shared failure risk



More to troubleshoot

When a call fails, check both sides of the proxy.



A shared proxy came with tradeoffs

Another hop. A shared fleet. Harder to troubleshoot..



Extra hop

Every request passes through the proxy.



Shared fleet

Capacity management and Shared failure risk



More to troubleshoot

When a call fails, check both sides of the proxy.

Could we remove the intermediate proxy?



01

The cross-cluster challenge

Why we looked beyond load balancers and shared proxies

02

Direct routing inside a cluster

How clients discover pods and choose where to send requests

03

Taking direct routing across clusters

Making remote pods reachable and discoverable



INSIDE A SINGLE CLUSTER

Clients discover Individual backend pods

Pod IP addresses + endpoint metadata

EXAMPLE ENDPOINT

Service B- Pod 1

Address: 10.20.1.7:8443

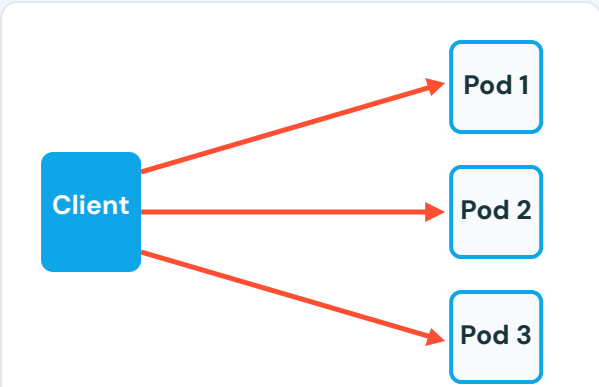
Zone: zone-a

Ready: true



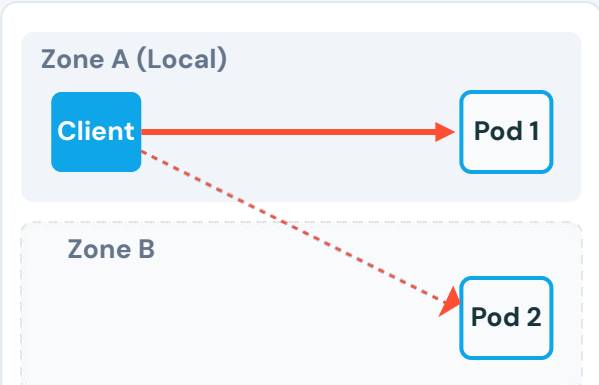
Client-side routing inside a cluster

Capabilities enabled by direct routing



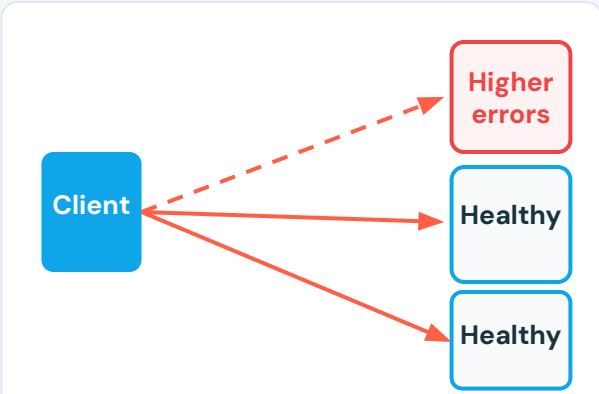
Per-request balancing

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

01

The cross-cluster challenge

Why we looked beyond load balancers and shared proxies

02

Direct routing inside a cluster

How clients discover pods and choose where to send requests

03

Taking direct routing across clusters

Making remote pods reachable and discoverable



How do we name a service across clusters?

Two clusters can have a service with the same name.

Inside one cluster

`service-b.local`

The caller's cluster is implied.

Across clusters

`service-b.cluster-a`

`service-b.cluster-b`

The caller can specify the cluster.



Routing across clusters.

Clients need to discover remote pods and reach them directly.

Connectivity

Can I reach their IPs?

Direct pod-to-pod network routability

Discovery

Which pods can I call?

Endpoints + Metadata discovery across clusters



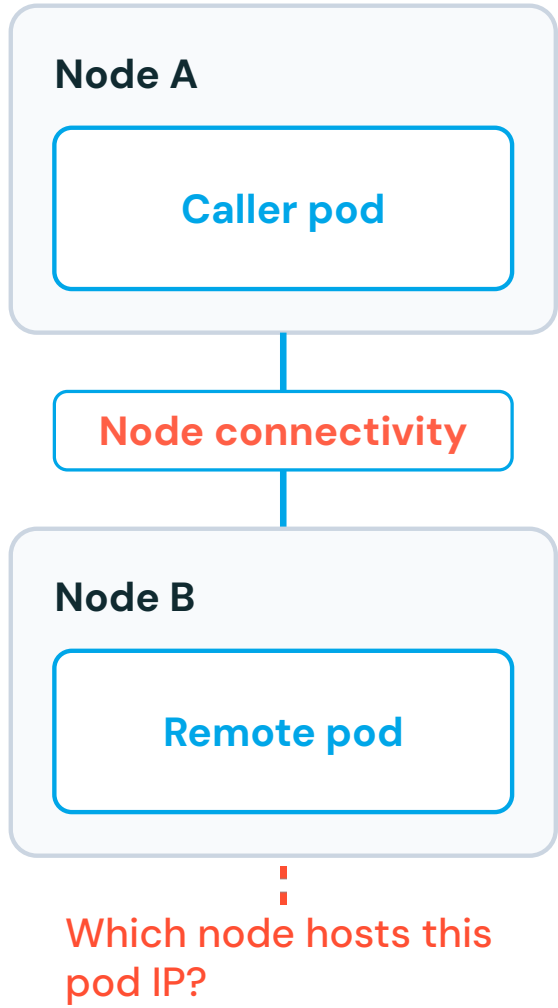
What cross-cluster pod connectivity requires

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



What cross-cluster pod connectivity requires

A network path from the caller to the remote pod

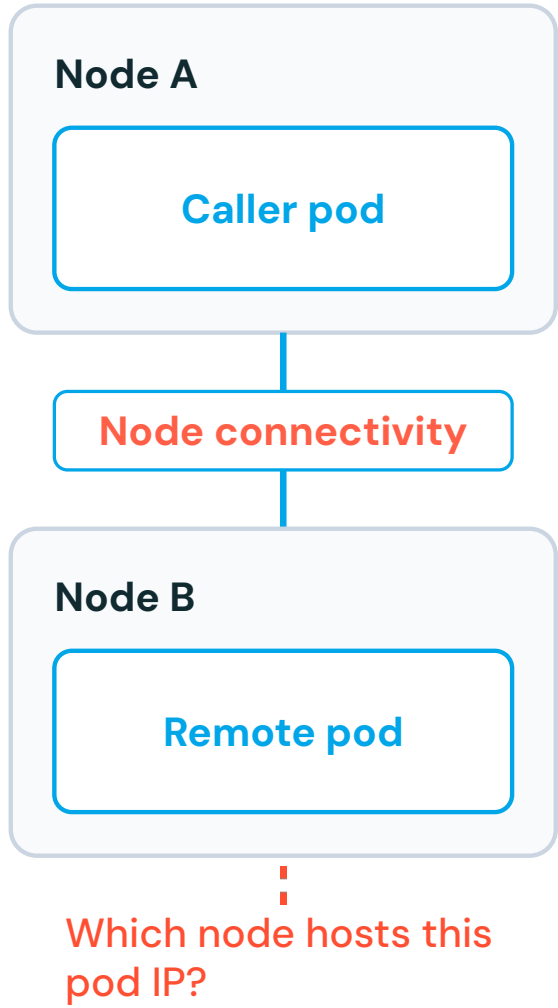
01 Reach the nodes

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

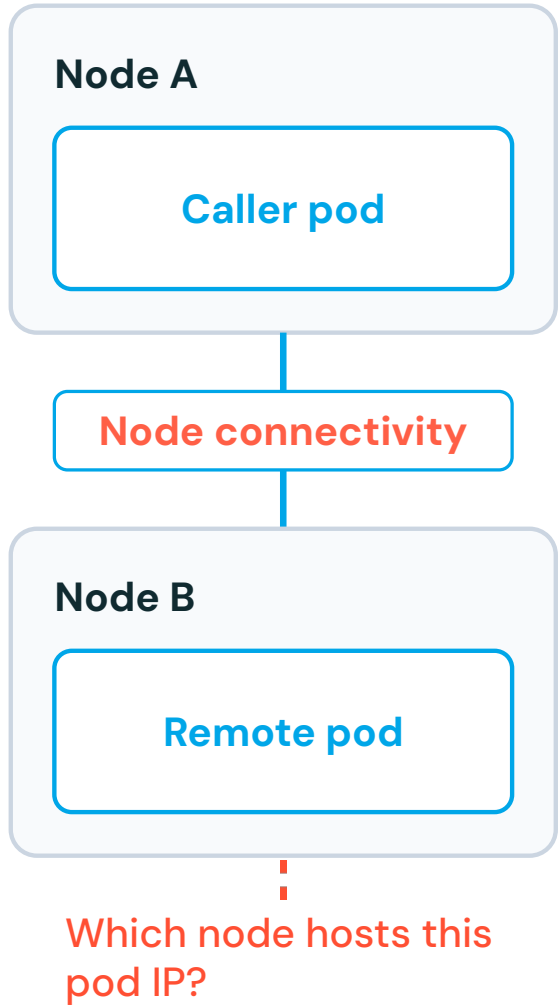
Already available in our setup

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.

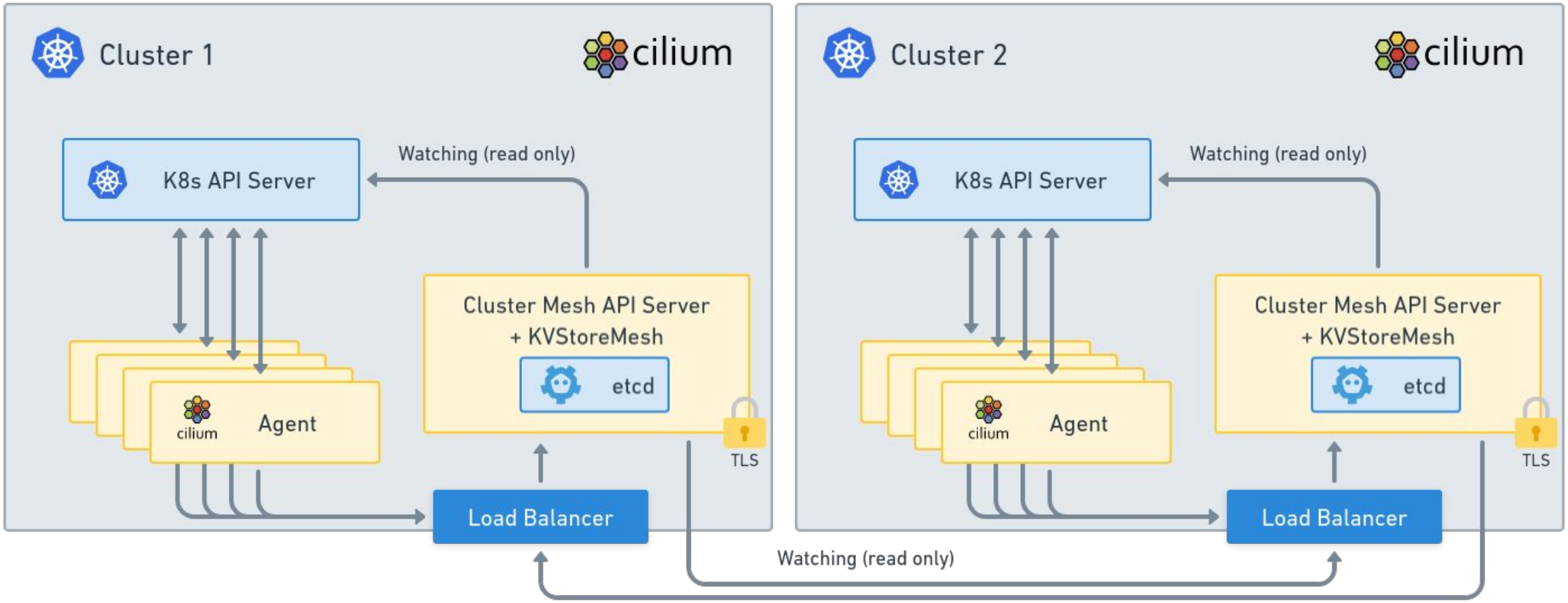
Works with our client libraries

Per-request balancing and zone affinity keep working.



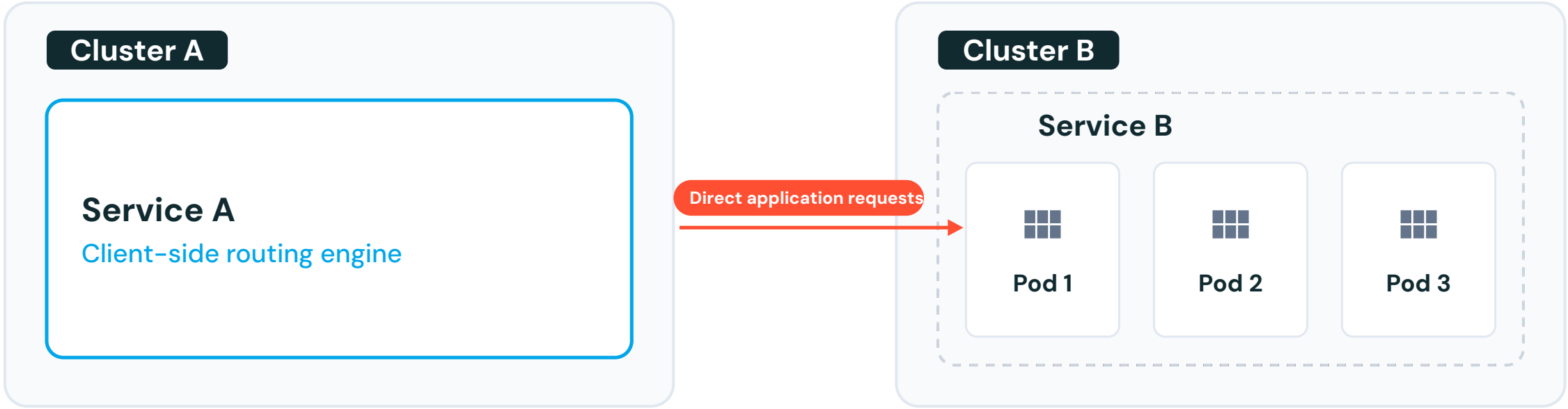
What is Cilium ClusterMesh?

Connects Kubernetes clusters running Cilium.



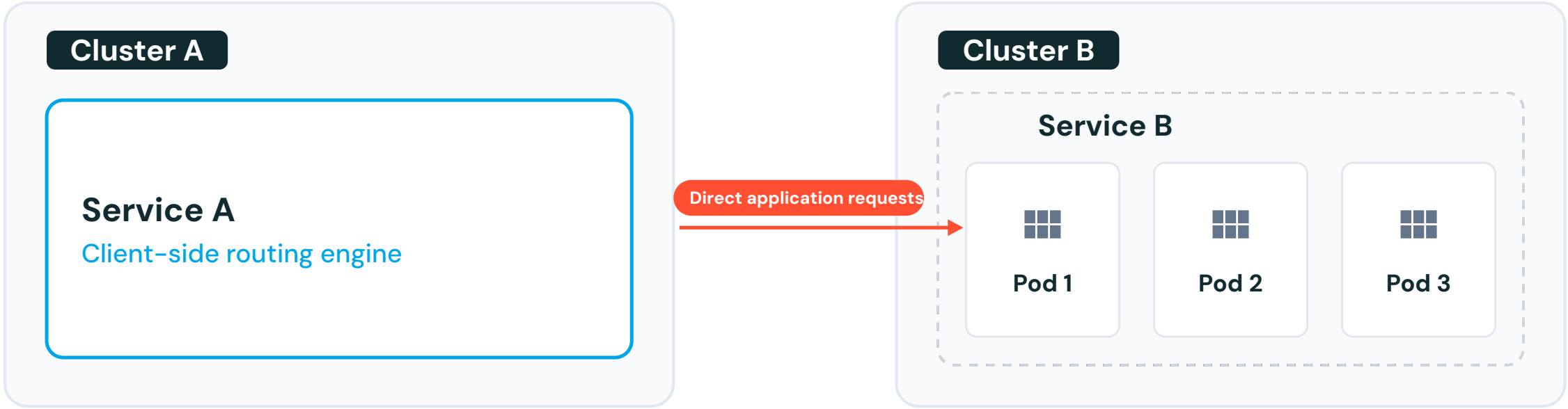
Making pod IPs reachable across clusters.

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?

Routing across clusters.

Clients need to discover remote pods and reach them directly.

Connectivity

Can I reach their IPs?

Direct pod-to-pod network routability

Discovery

Which pods can I call?

Endpoints + Metadata discovery across clusters



Approach One: A Cilium-native option

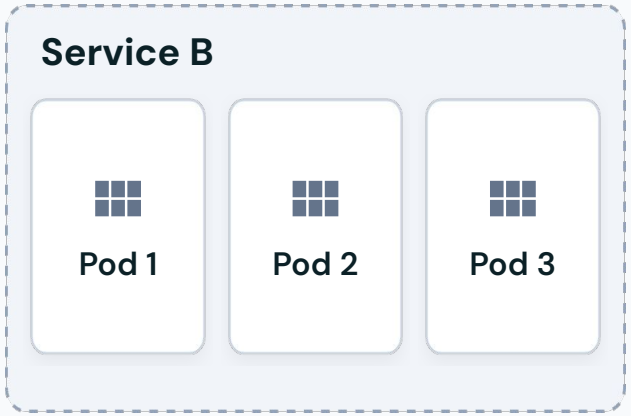
Annotate the Service in every cluster.

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"
```



Approach One: A Cilium-native option

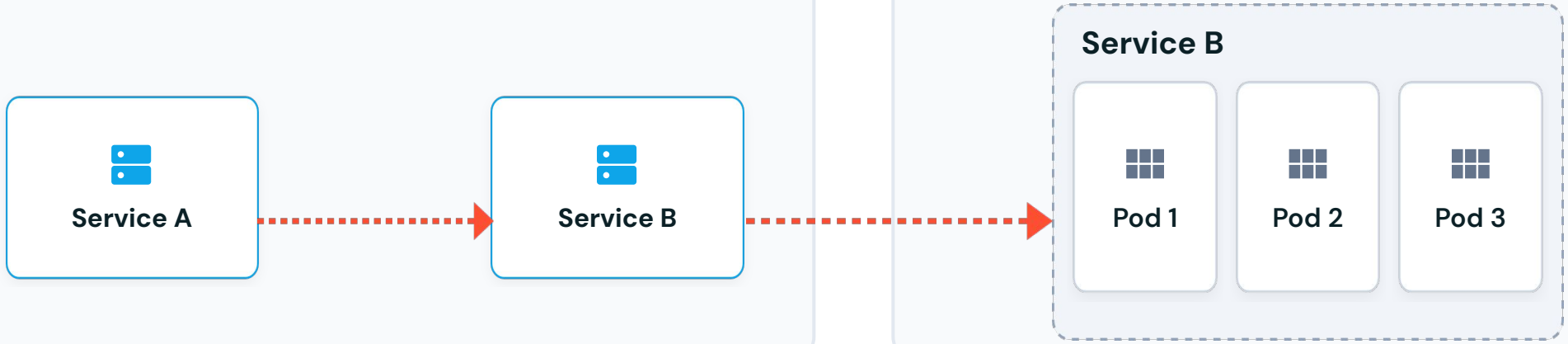
Annotate the Service in every cluster.

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 cross-cluster traffic what it does for in-cluster traffic?



Approach Two: EDS as Cilium subscriber

Just read what ClusterMesh is already propagating

KEY

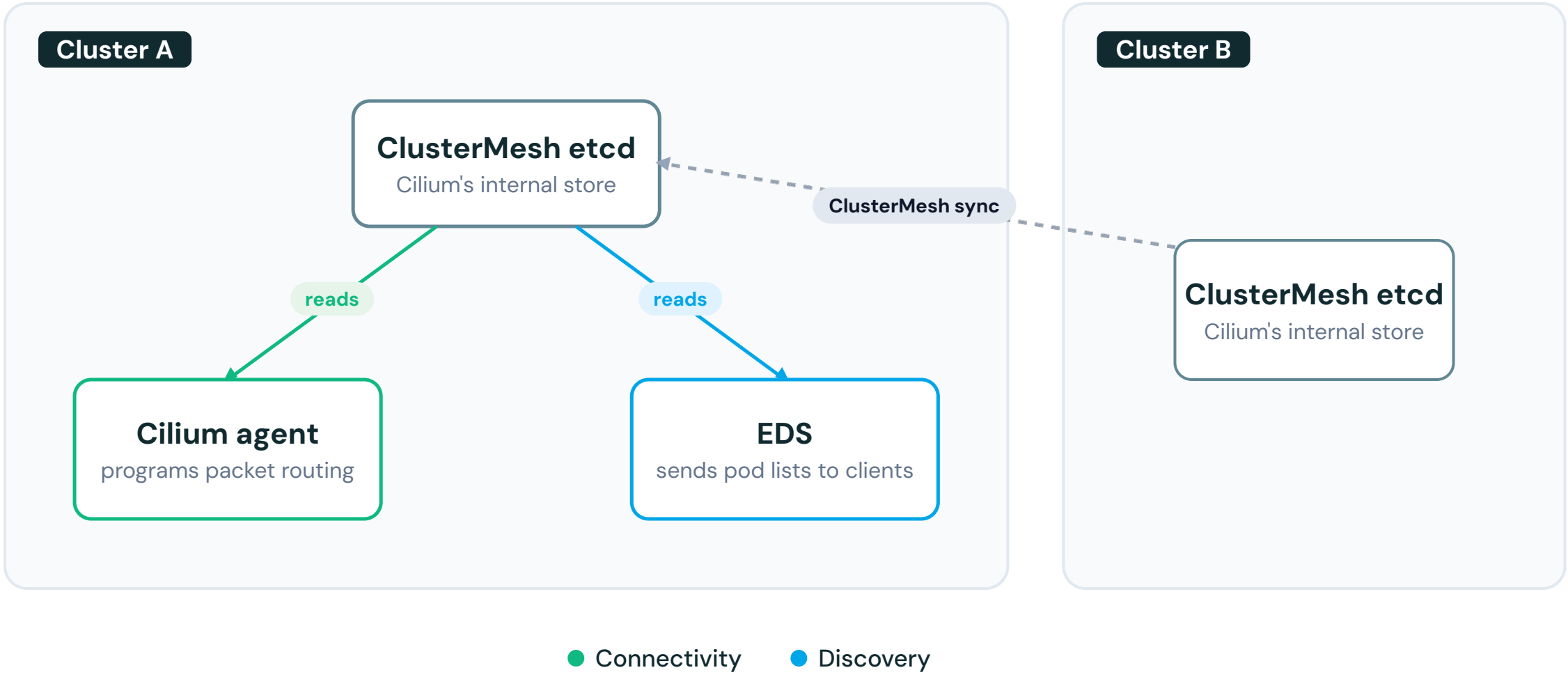
`cilium/cache/services/v1/<cluster>/<namespace>/<service>`

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 } }
},
...
```

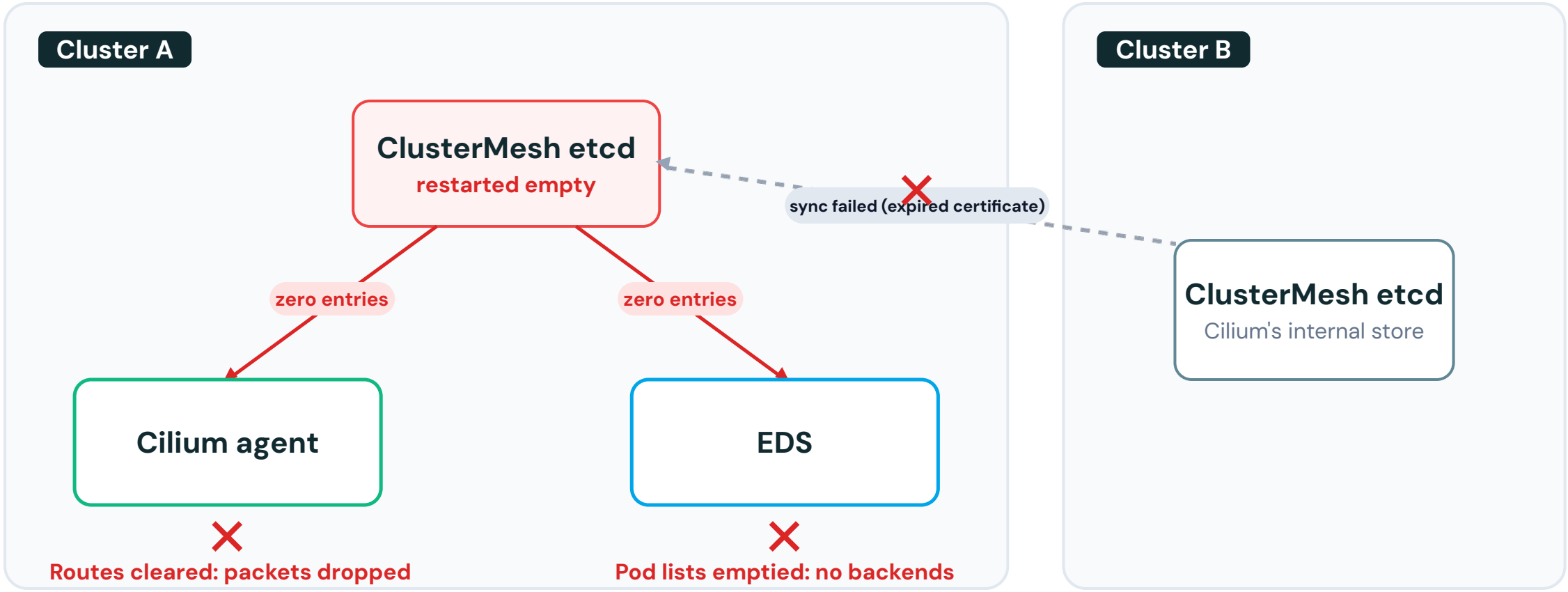
Approach Two: EDS as Cilium subscriber

EDS as just another subscriber to the ClusterMesh etcd



One fault. Two outages.

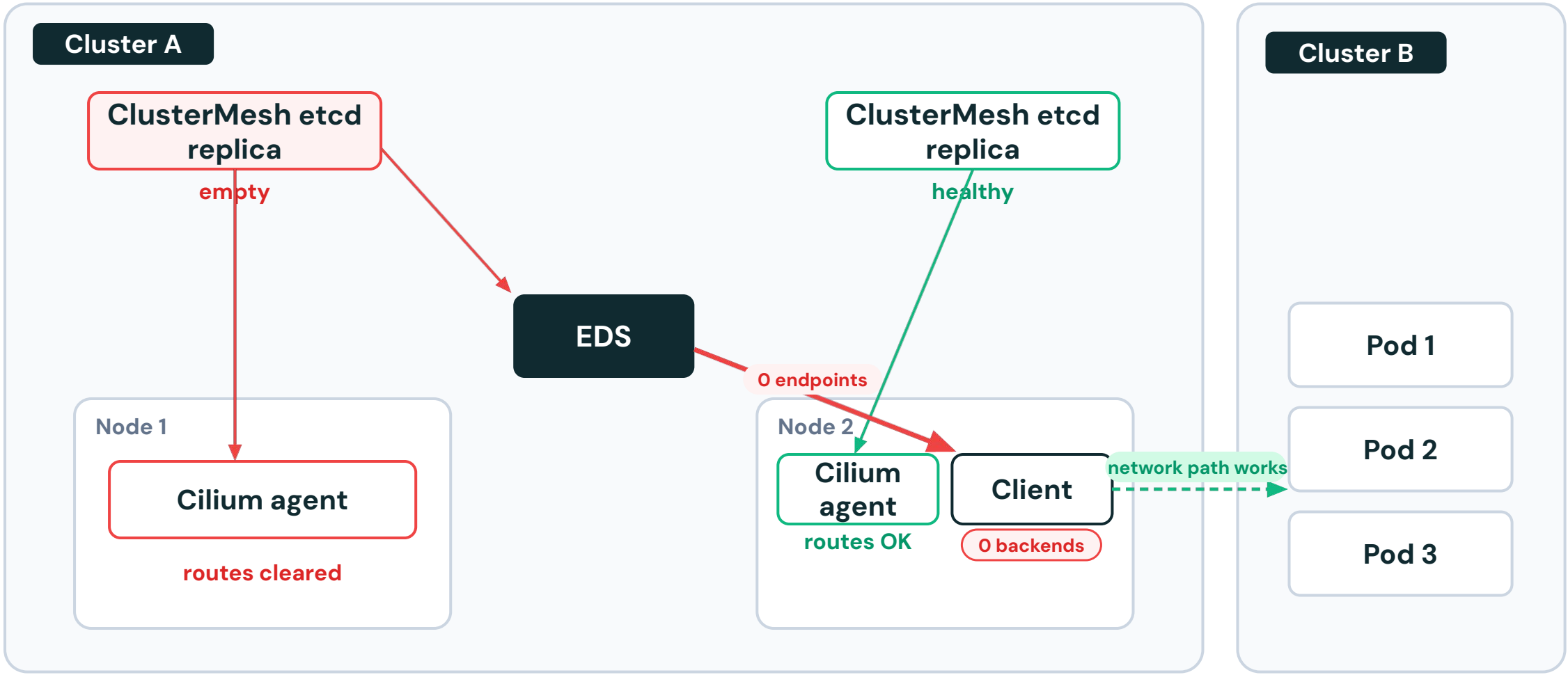
When the store came back stale or empty, both layers broke



Both layers trusted an empty answer.

A partial failure, amplified

EDS spread an etcd's empty answer to every client it served.



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

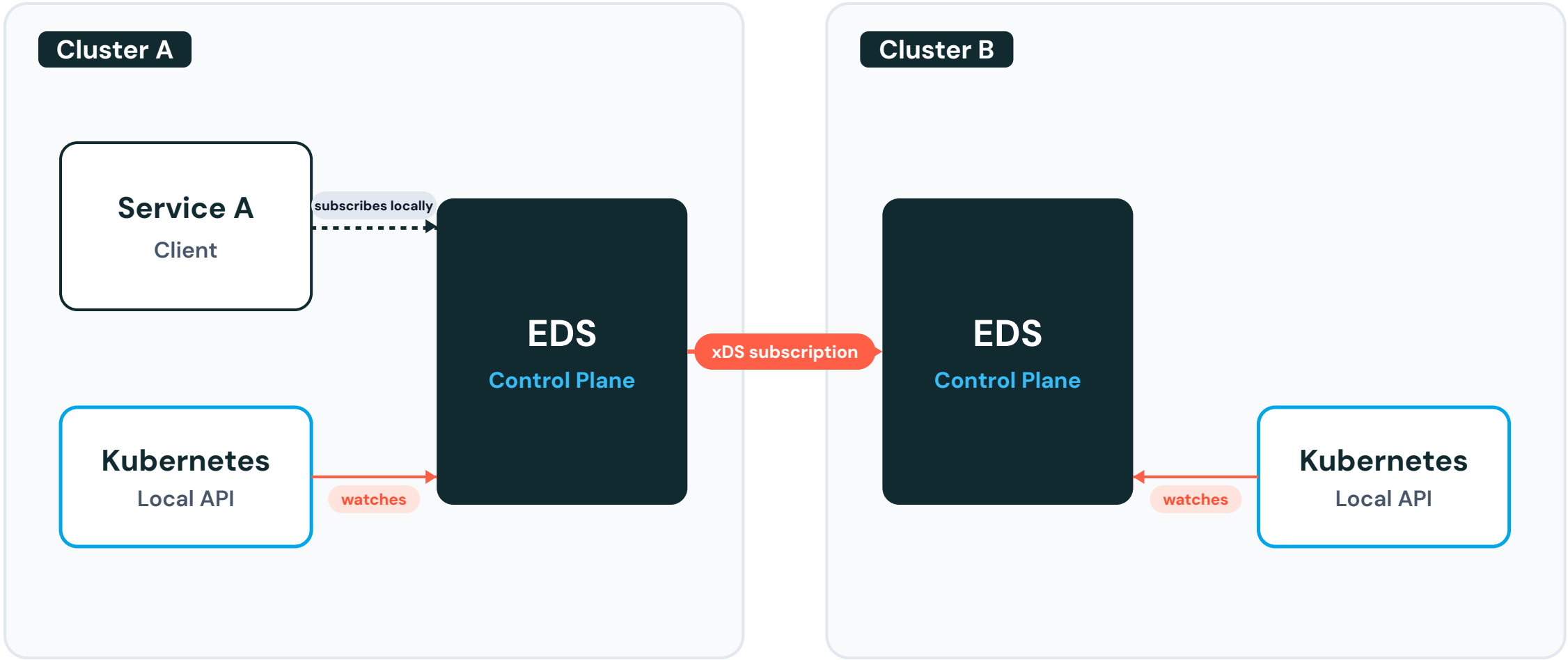
Let Cilium handle connectivity.

Let EDS own discovery itself.



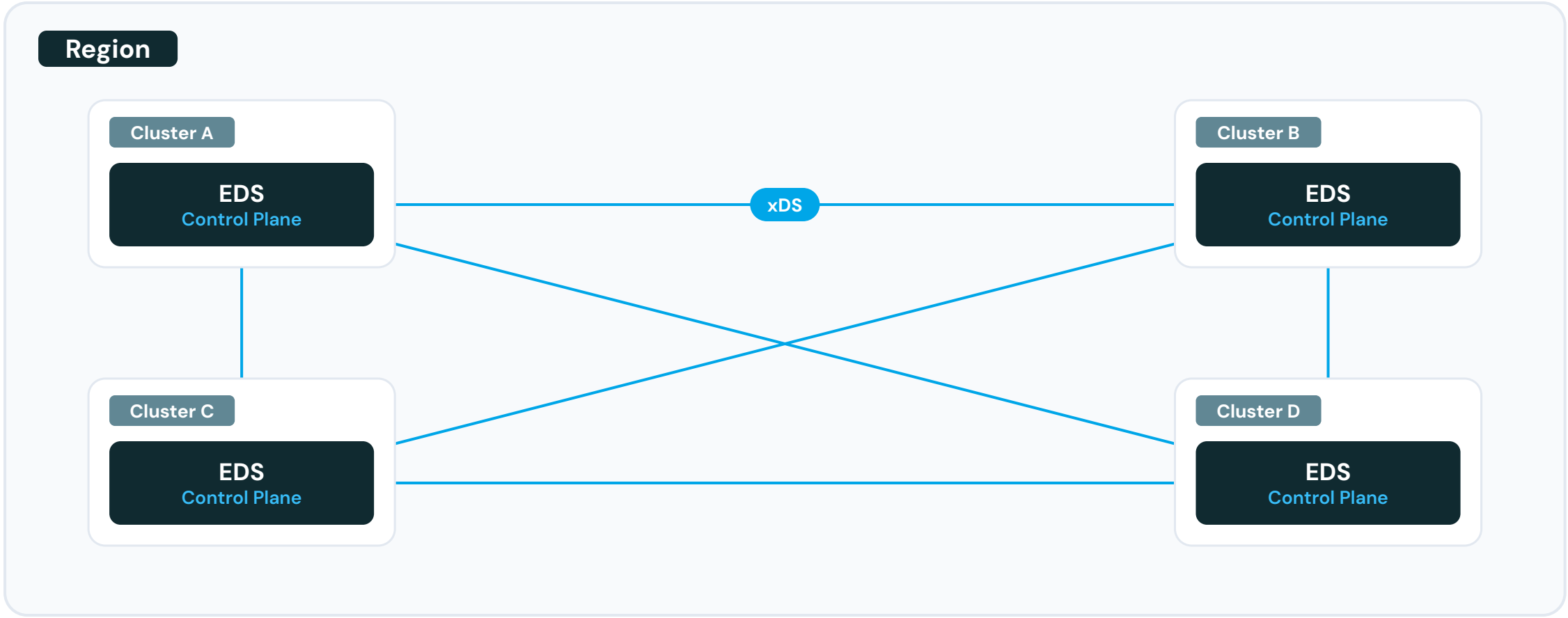
Approach Three: Cross-cluster EDS

An EDS in another cluster is just another xDS subscriber.



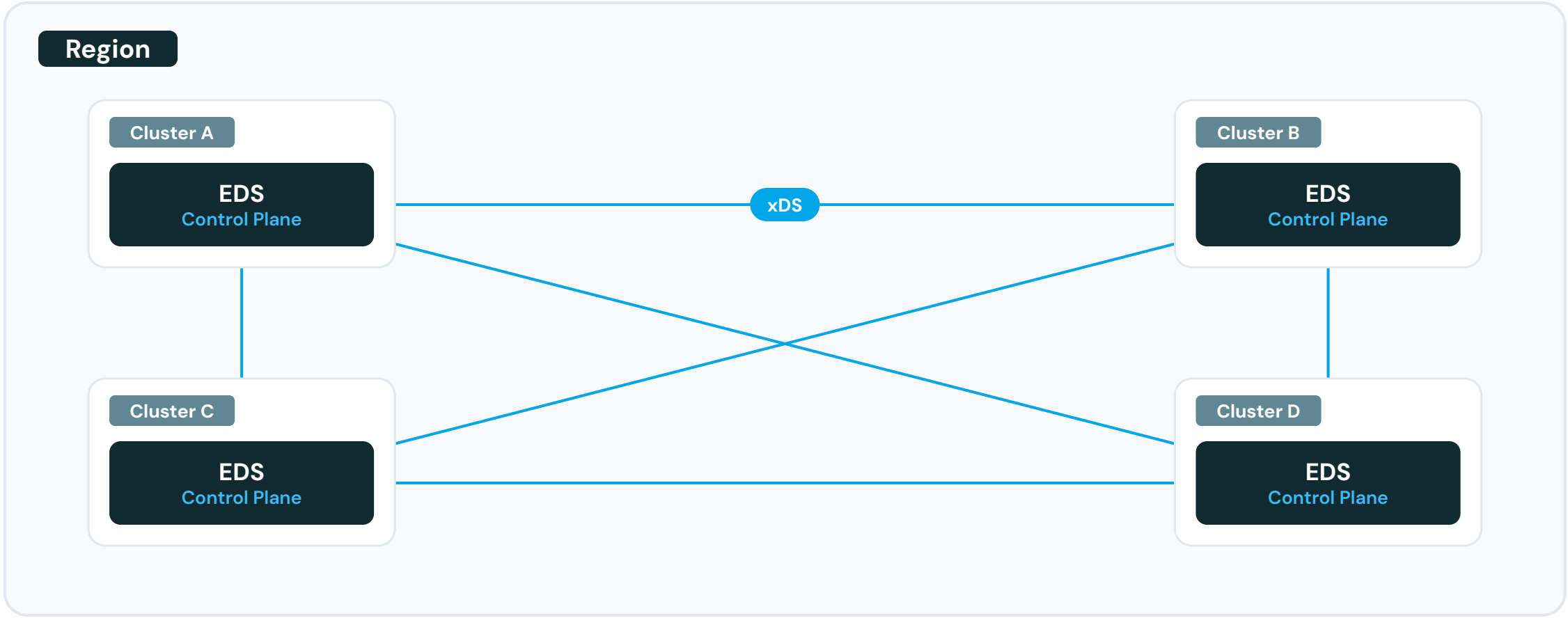
A regional mesh of EDS peers

Every EDS subscribes to every other EDS in the region over xDS.



A regional mesh of EDS peers

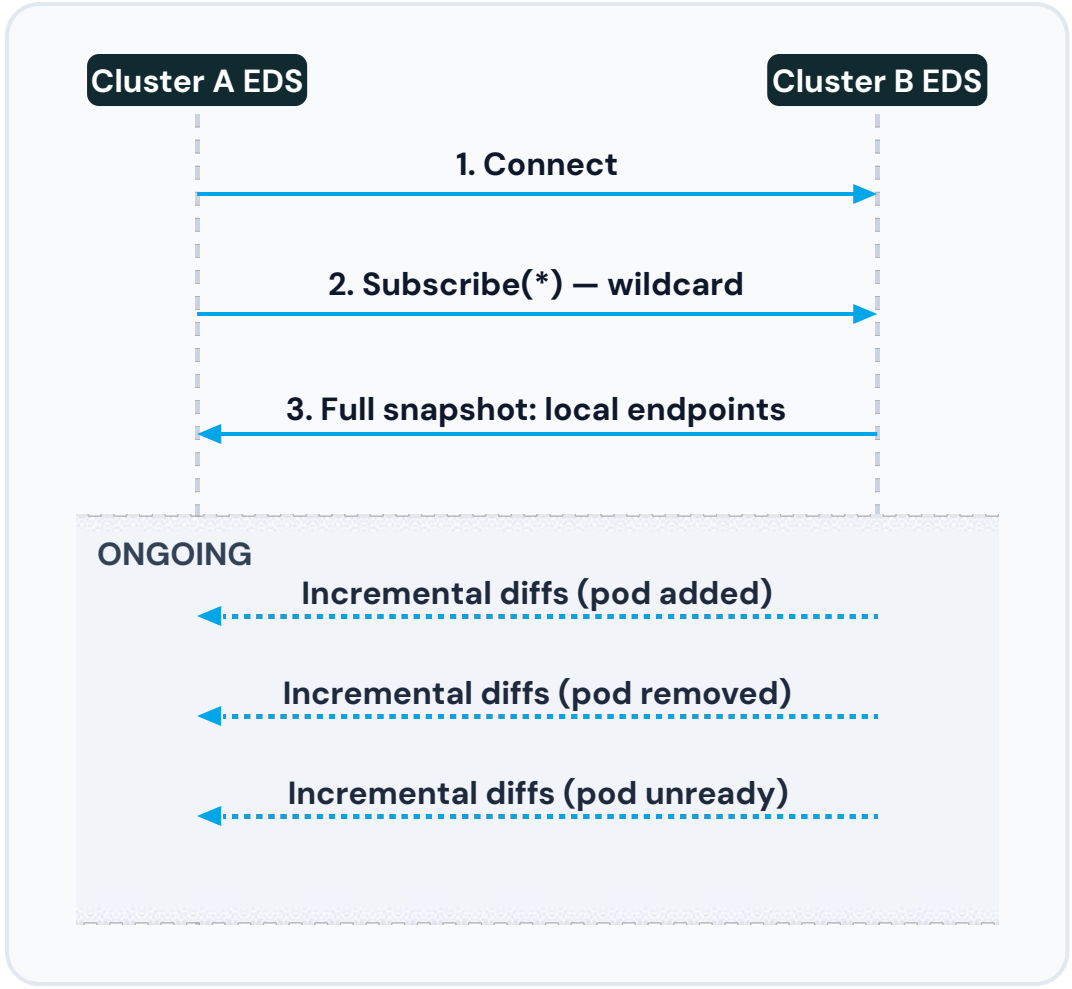
Every EDS subscribes to every other EDS in the region over xDS.

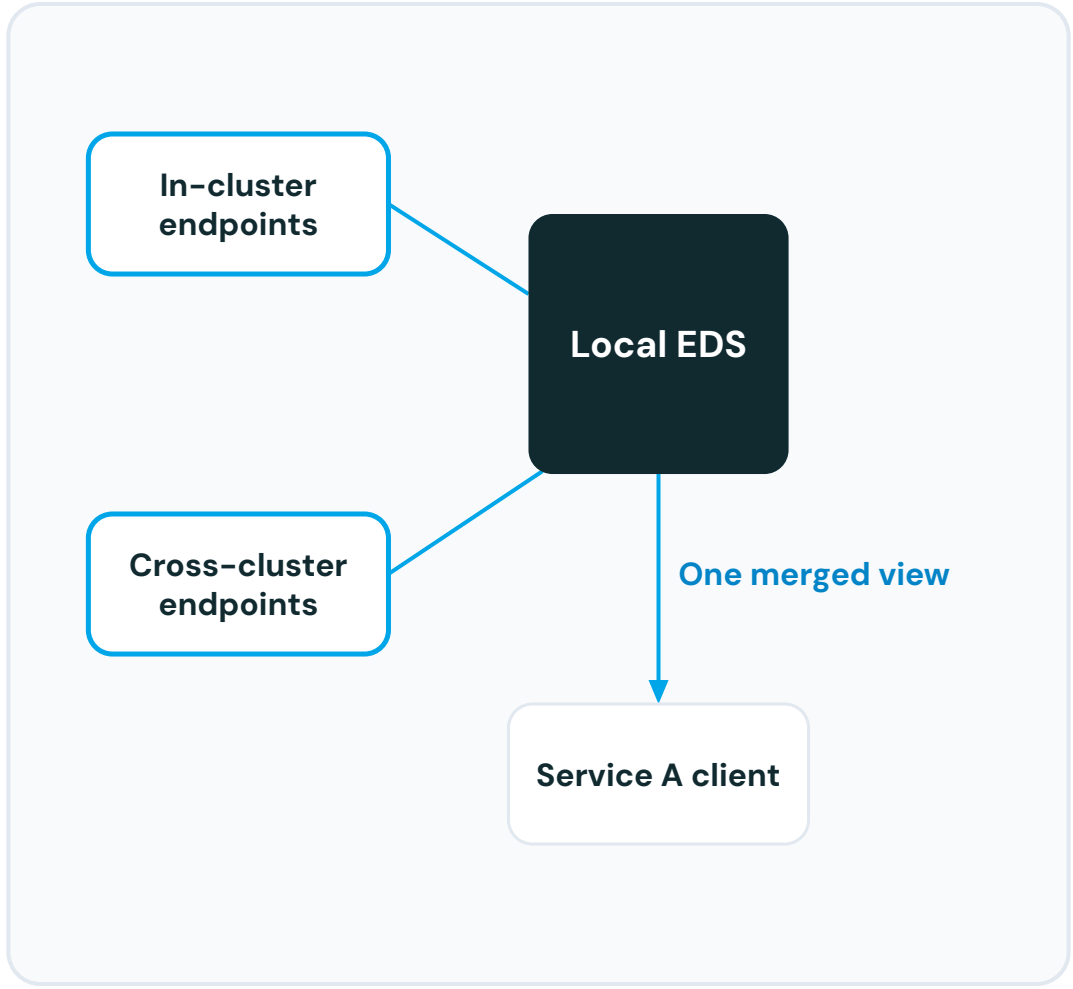


N clusters → each EDS has N-1 peers. Can we make this scale?

How the subscriptions work

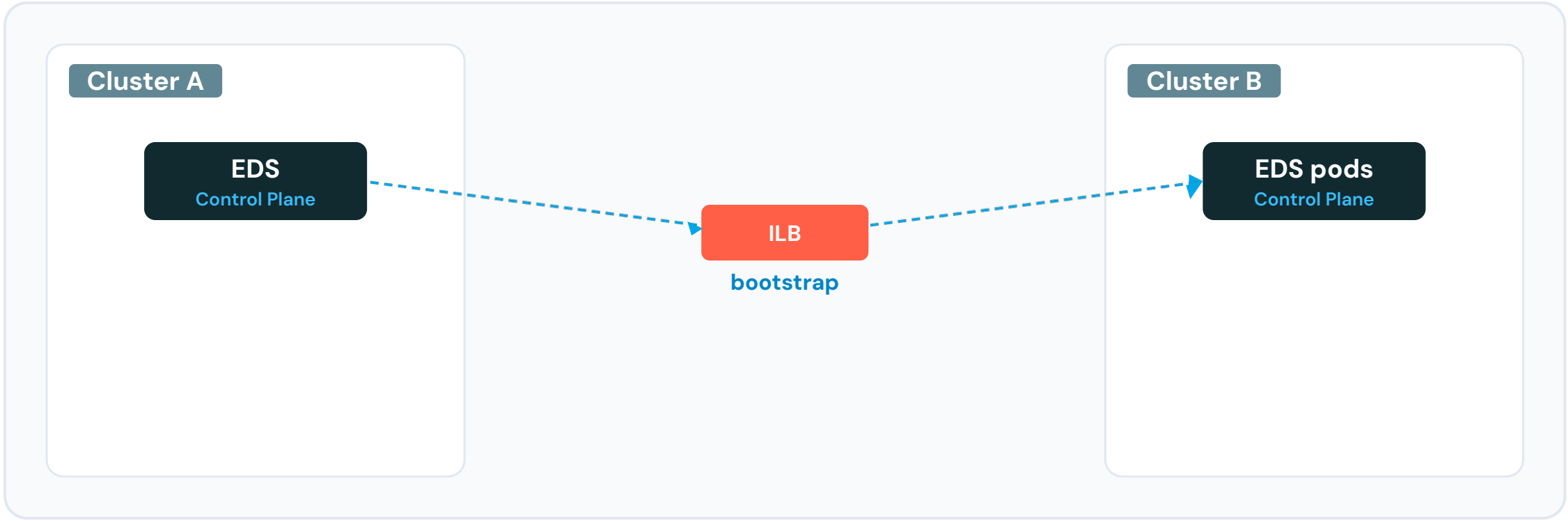
Full state once, then only changes.





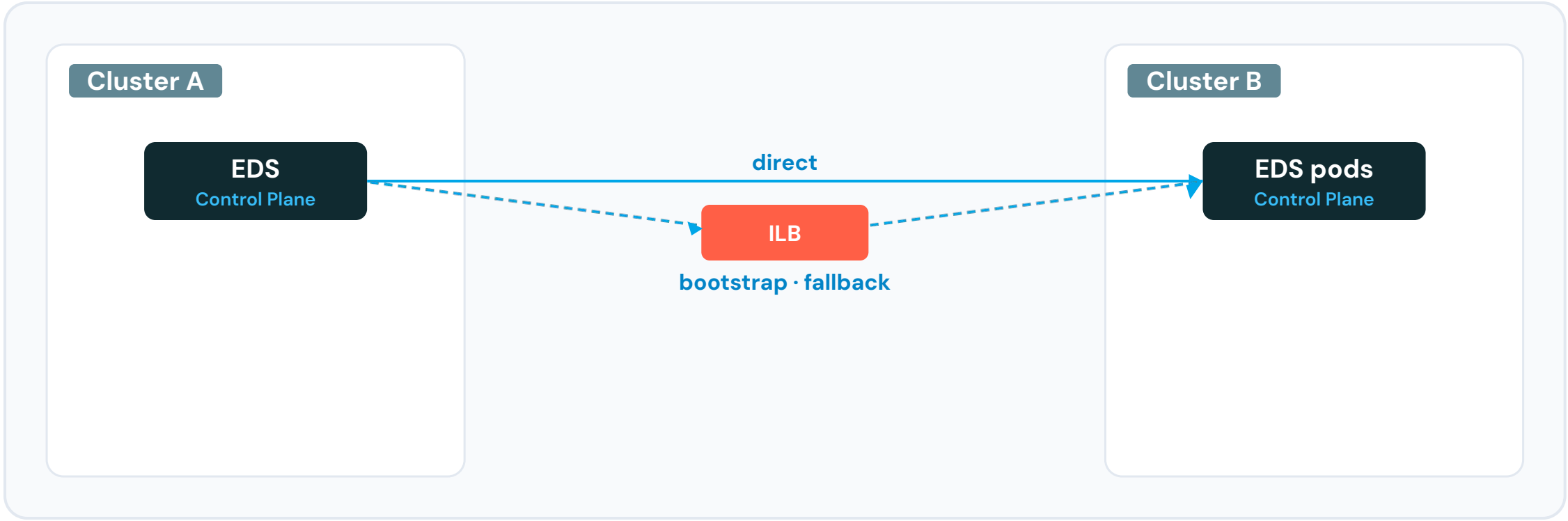
How EDS finds its peers

Start with a load balancer



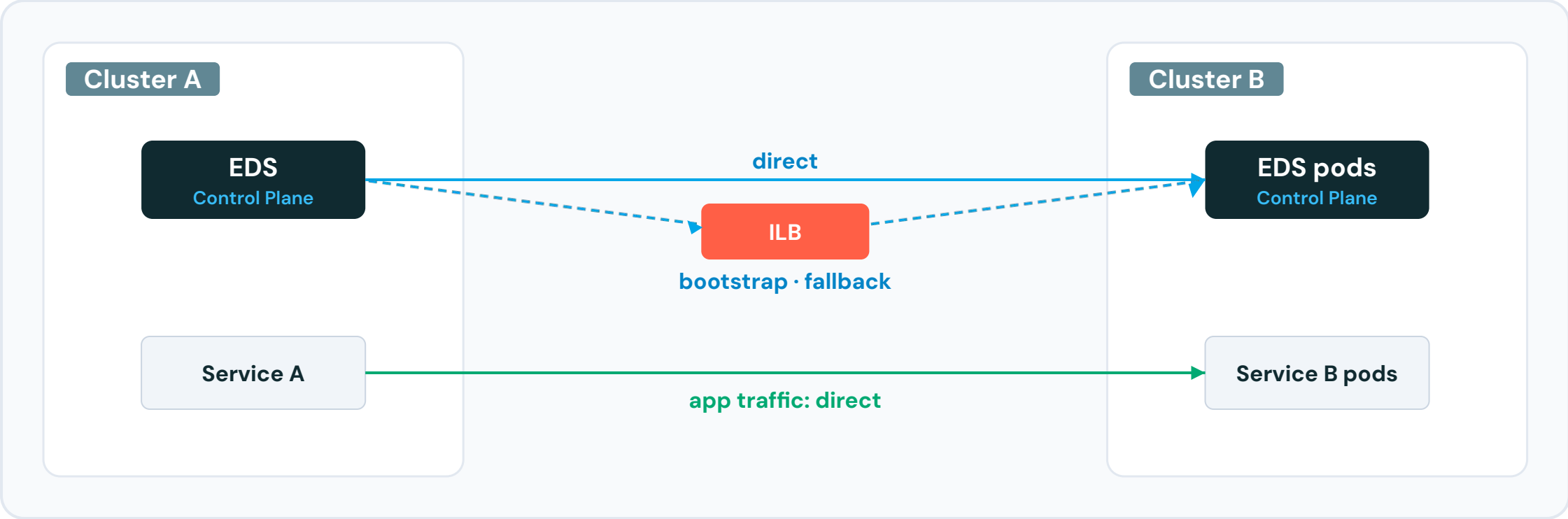
How EDS finds its peers

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

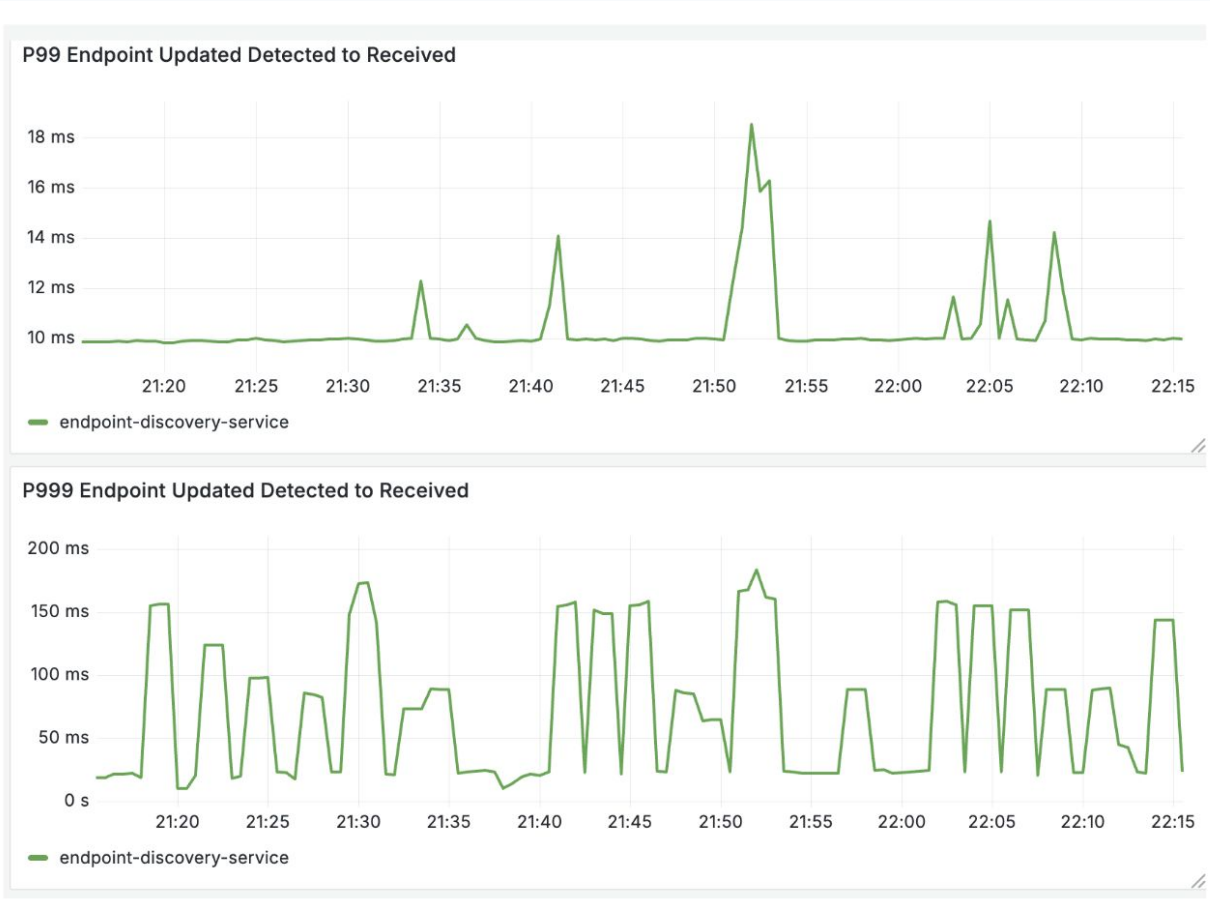
endpoints in one region

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

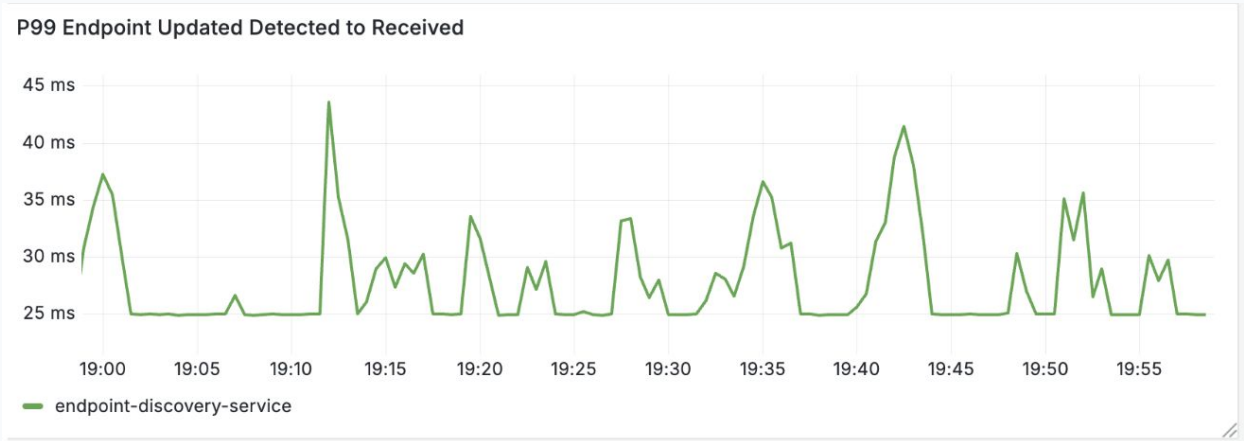
endpoints in one region

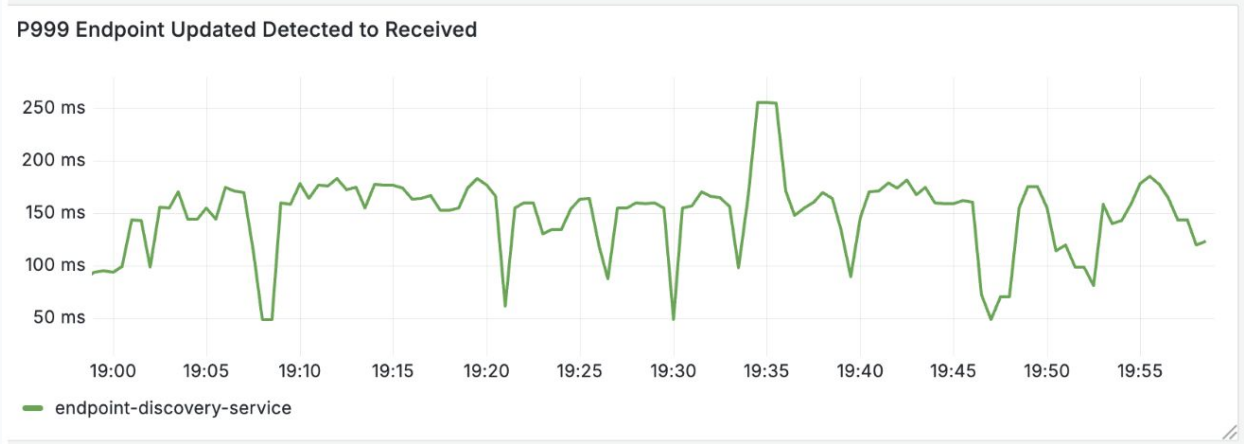
200

clusters meshed
(50K endpoints each)

1,000/s

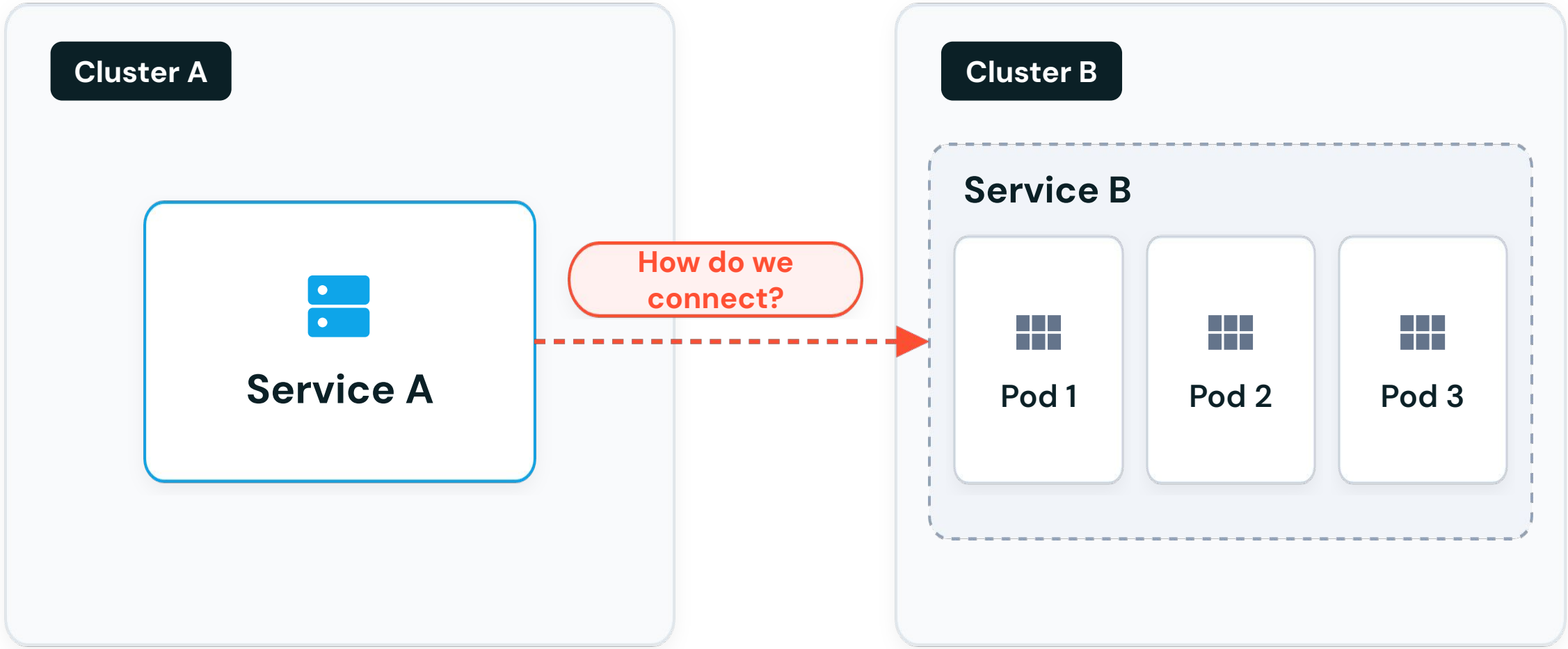
endpoint changes
across the region





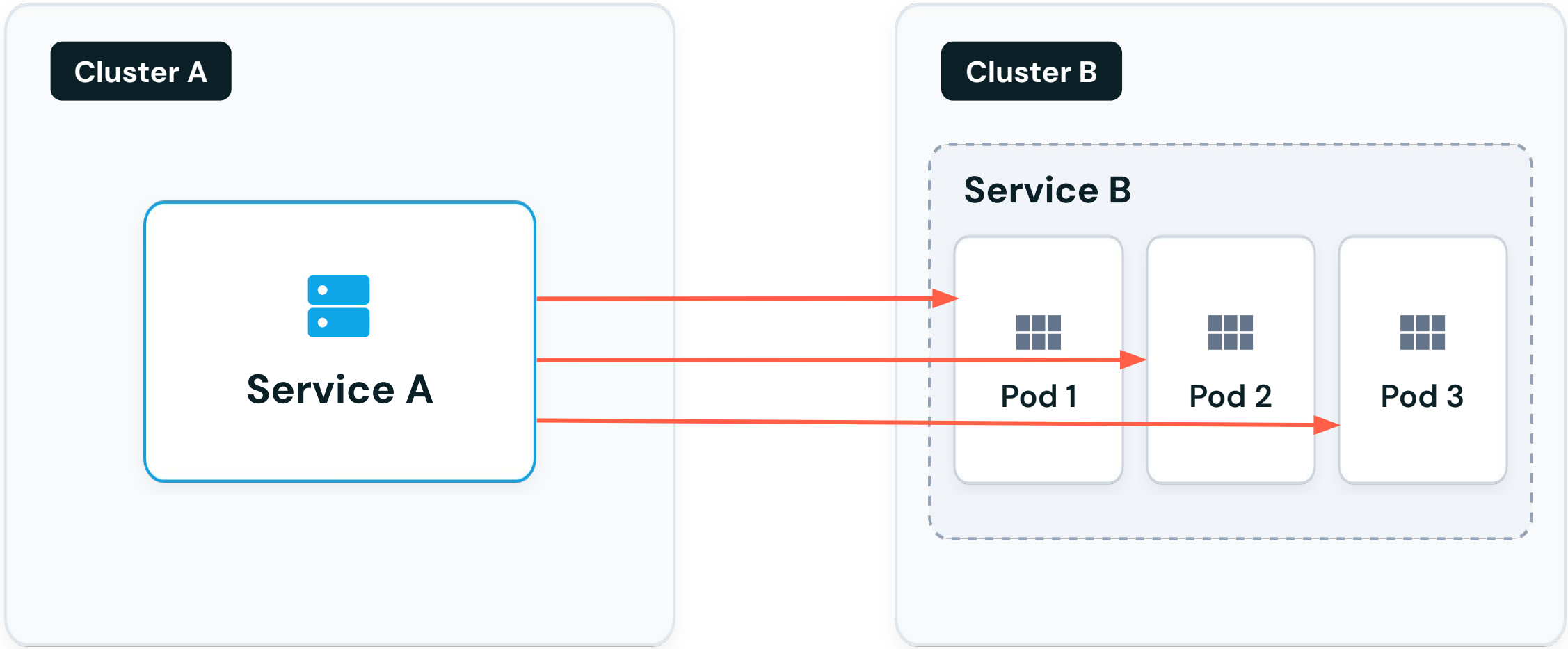
I just need to call another service

But it runs in another cluster



I just need to call another service

Just name the service and send the request.



Ongoing Challenges

- Handling eventual consistency
- Protecting clients against failing EDS pods
- Scaling connectivity and discovery beyond the region



Takeaways



Keep connectivity and discovery separate

They change at different rates and fail differently.



Healthy doesn't mean correct

A working connection can still return a bad answer.



Open source and custom can coexist

Integrate at a clean boundary, not through internals.



Thank You

Questions? / Q&A Session

KubeCon NA 2026

