BUOYANT

Creators of LINKERD

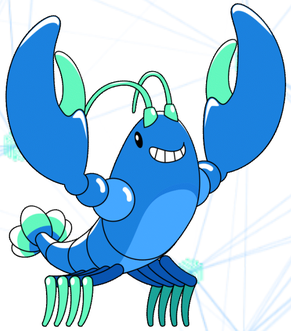
BUOYANT SERVICE MESH ACADEMY



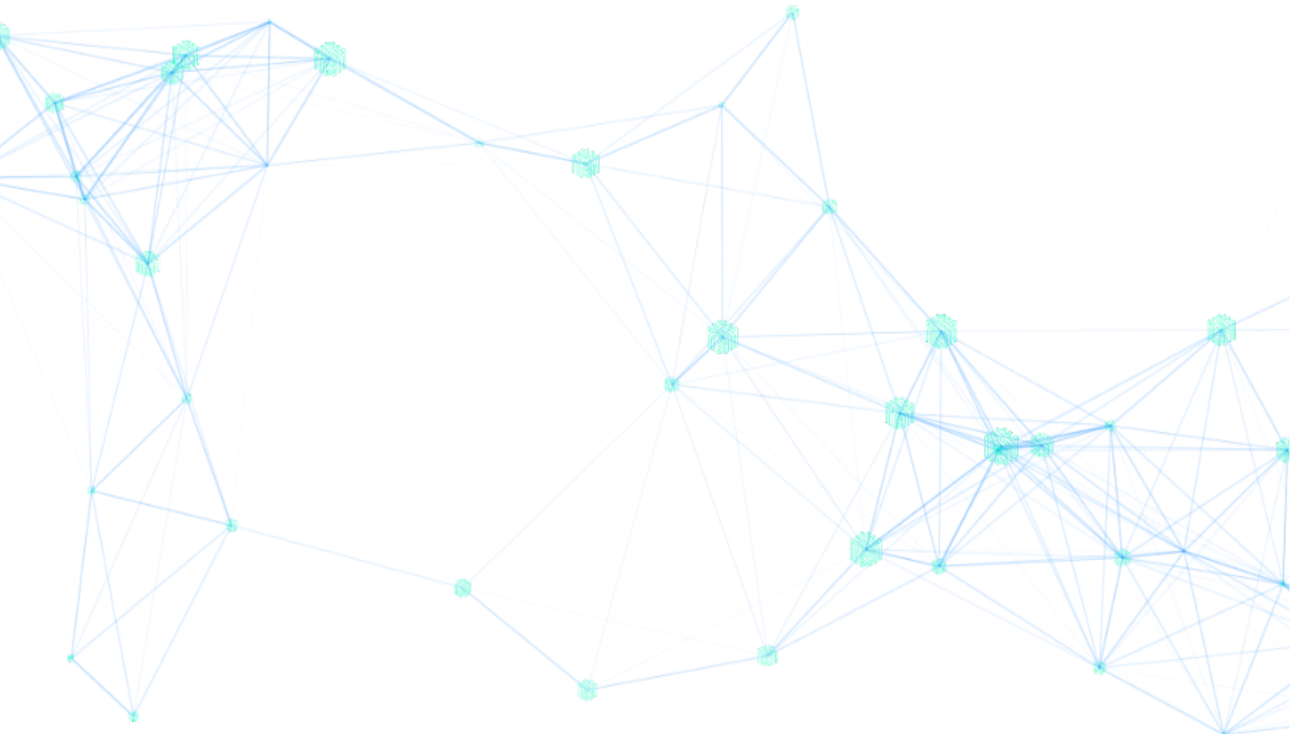
Linkerd 2.17: Federated Services and Egress

@BuoyantIO

buoyant.io



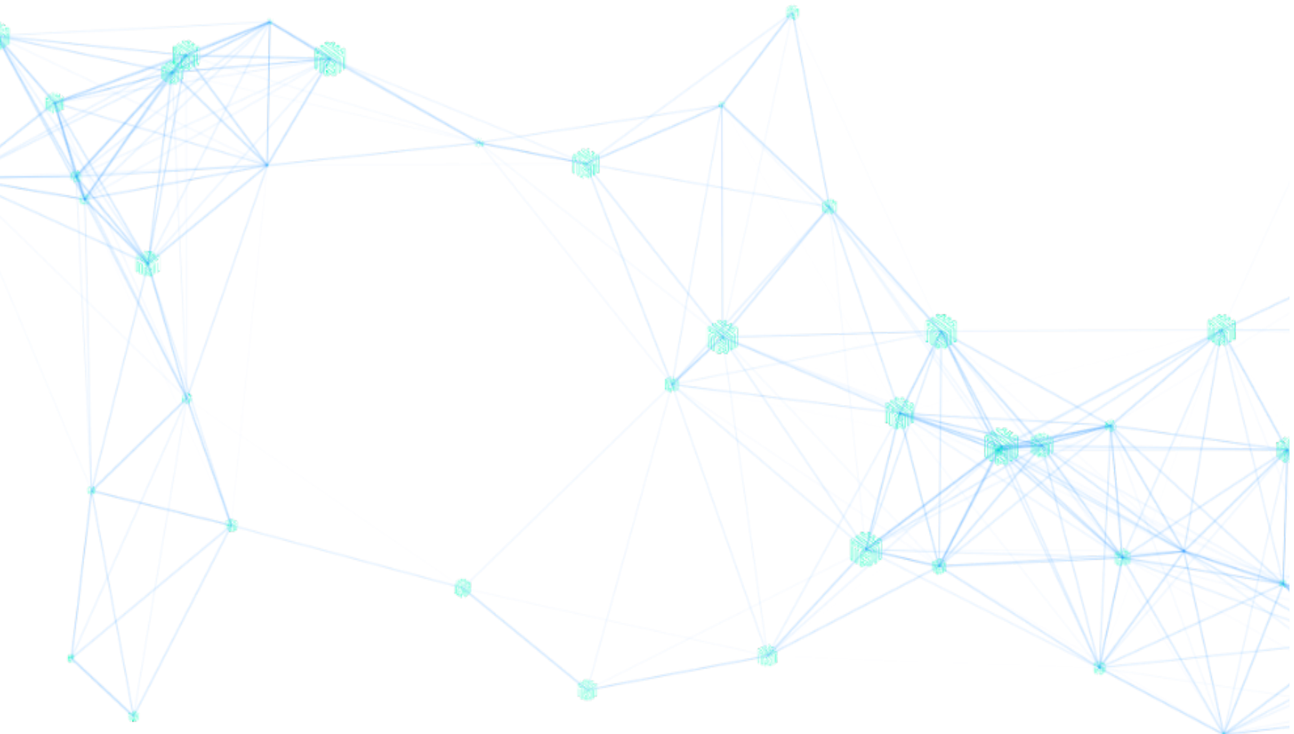
What's on the agenda?



- Intro to **federated Services**
- Federated Services **demo**!
- Intro to **egress**
- Egress **demo**!

BUOYANT

How do you follow along?



- https://github.com/BuoyantIO/service-mesh-academy/tree/main/2-17-features
- I’m using Buoyant Enterprise for Linkerd 2.17, but Linkerd edge-24.11.8 should work too.
- The demo also requires both k3d and kind!
  https://k3d.io/
  https://kind.sigs.k8s.io/

BUOYANT

Buoyant Enterprise for LINKERD

Rust-based network security and reliability for modern applications. Built on open source and designed for the enterprise.

- Zero-trust security and compliance across your entire network
- Global traffic management and control
- Full L7 application observability
- Built for the enterprise

Learn more & get a free trial at buoyant.io/**enterprise-linkerd**



BUOYANT

Creators of LINKERD

BUOYANT
SERVICE MESH
ACADEMY



January 16 2025: 😱

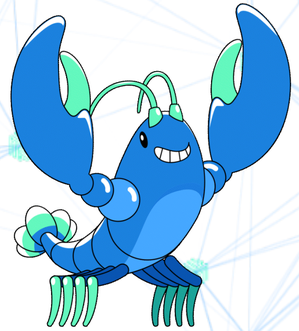
OpenTelemetry and Linkerd

with Michele Mancioppi

of Dash0

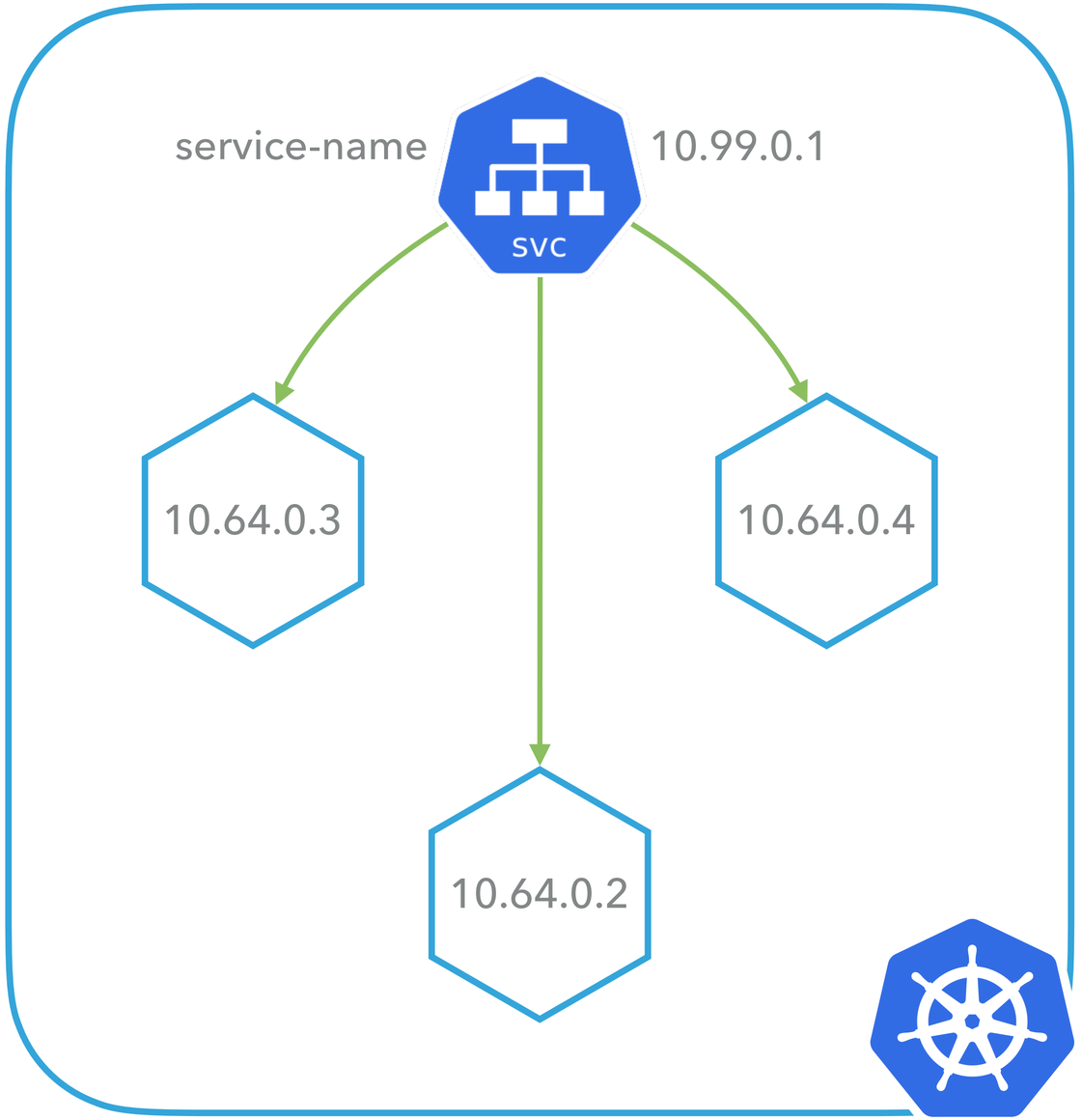
BUOYANT
SERVICE MESH
ACADEMY

Federated Services



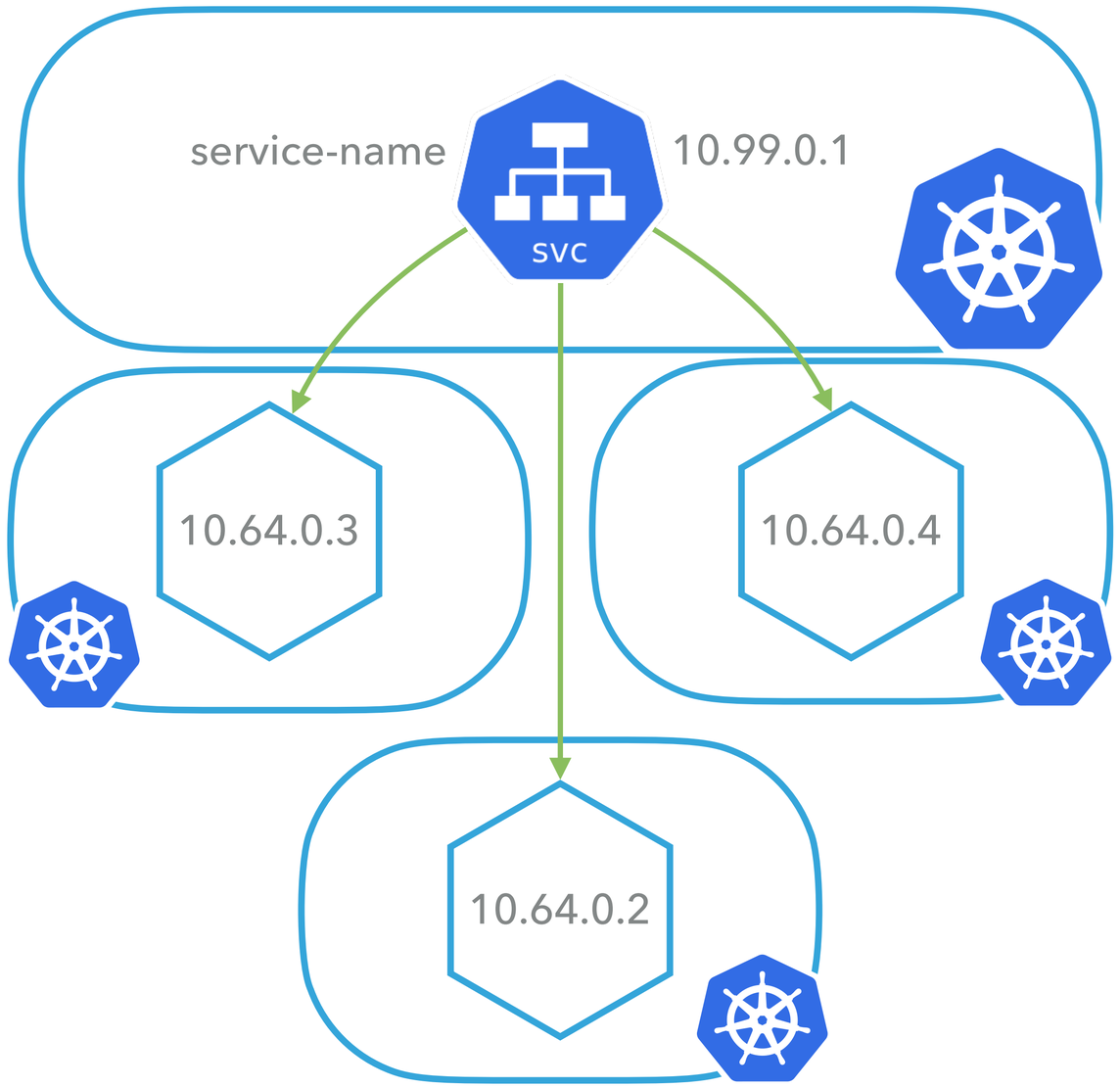
Federated Services

- “Service” is horribly overloaded in Kubernetes,
  - Here we’re talking about the Kubernetes resource that matches a DNS name to a bunch of Pods
  - More specifically, “endpoints”



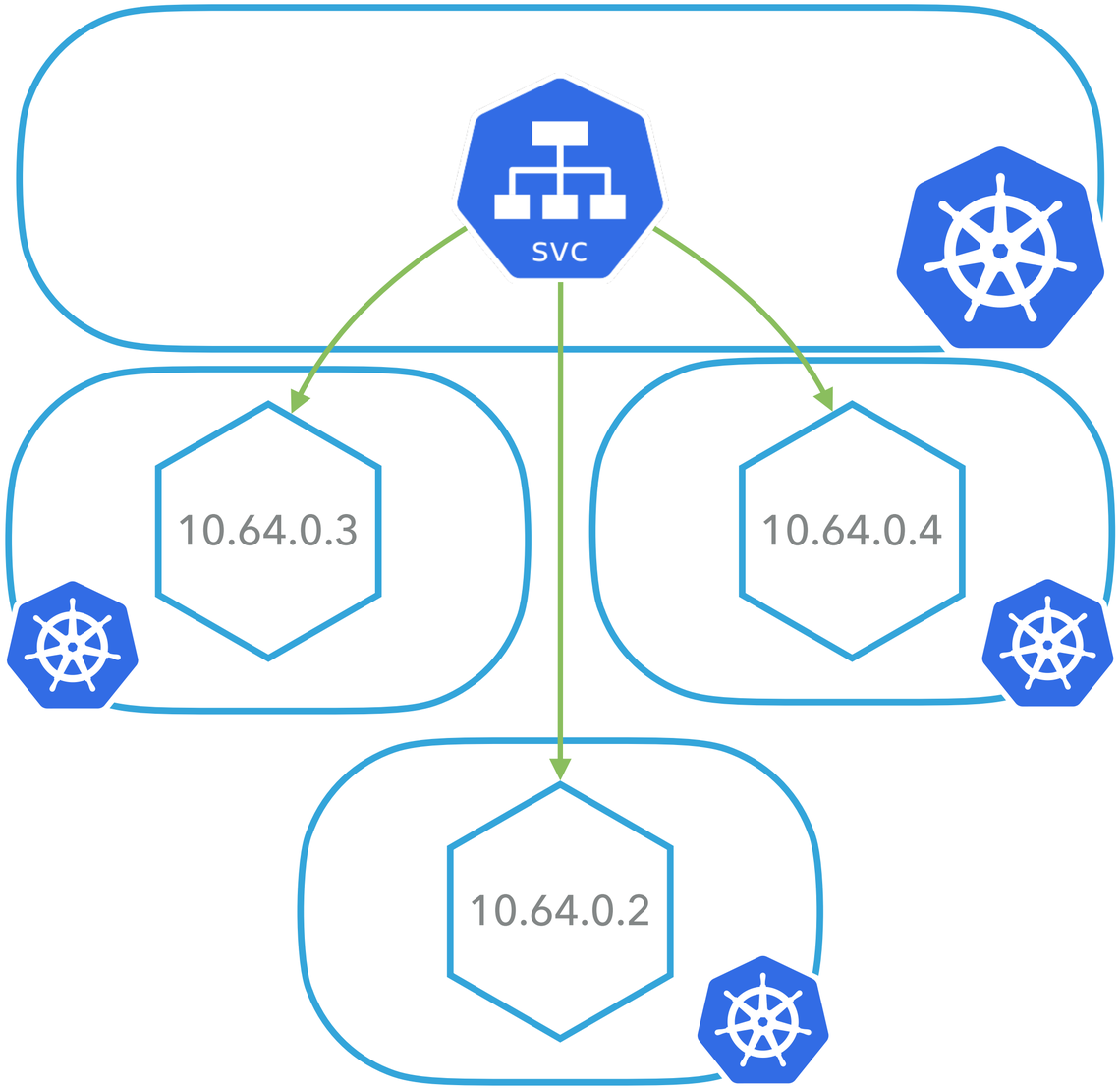
Federated Services

- “Service” is horribly overloaded in Kubernetes,
  - Here we’re talking about the Kubernetes resource that matches a DNS name to a bunch of Pods
  - More specifically, “endpoints”
- Suppose all the endpoints don’t have to be in the same cluster?



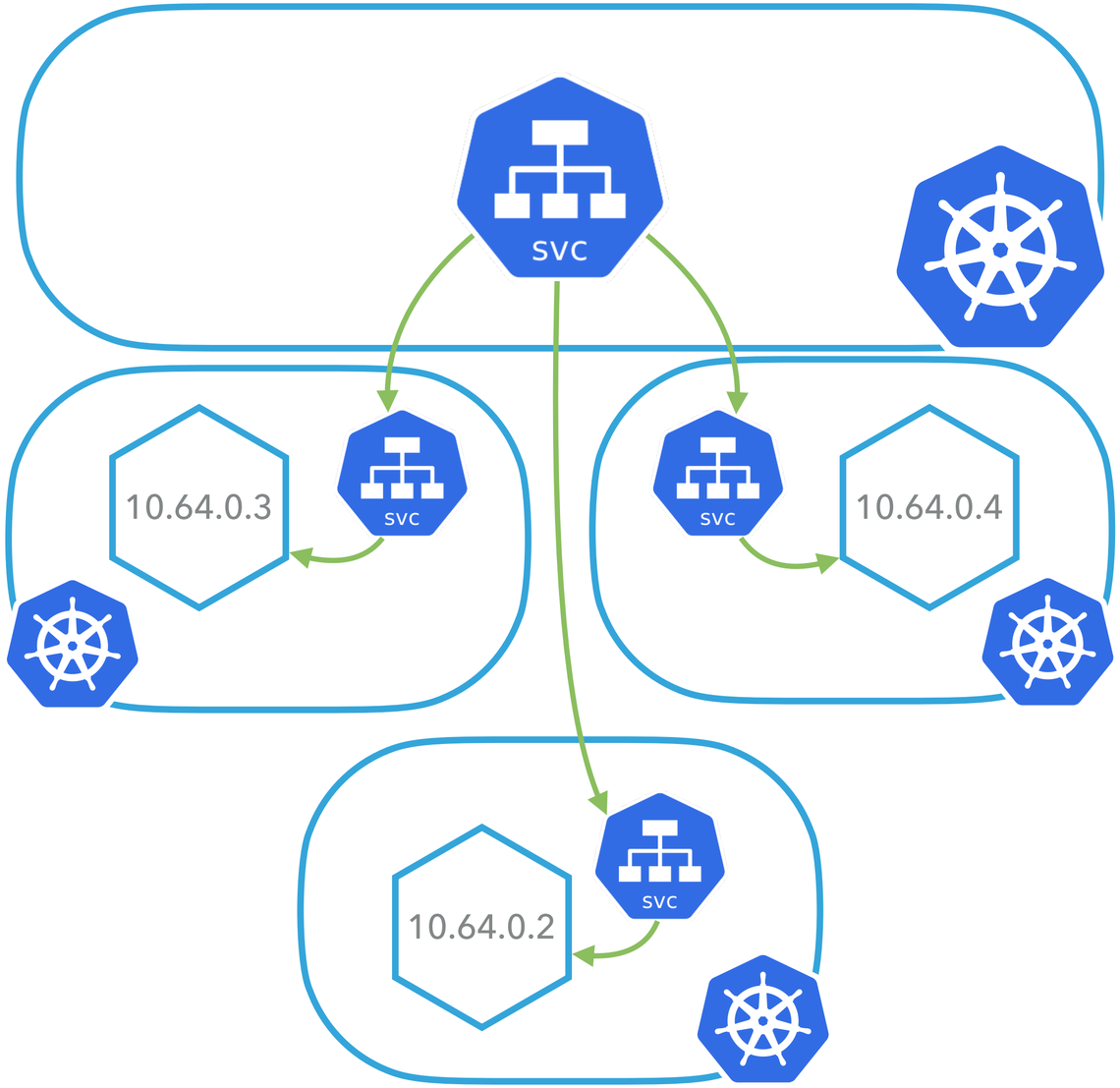
Federated Services

- This is a federated Service!



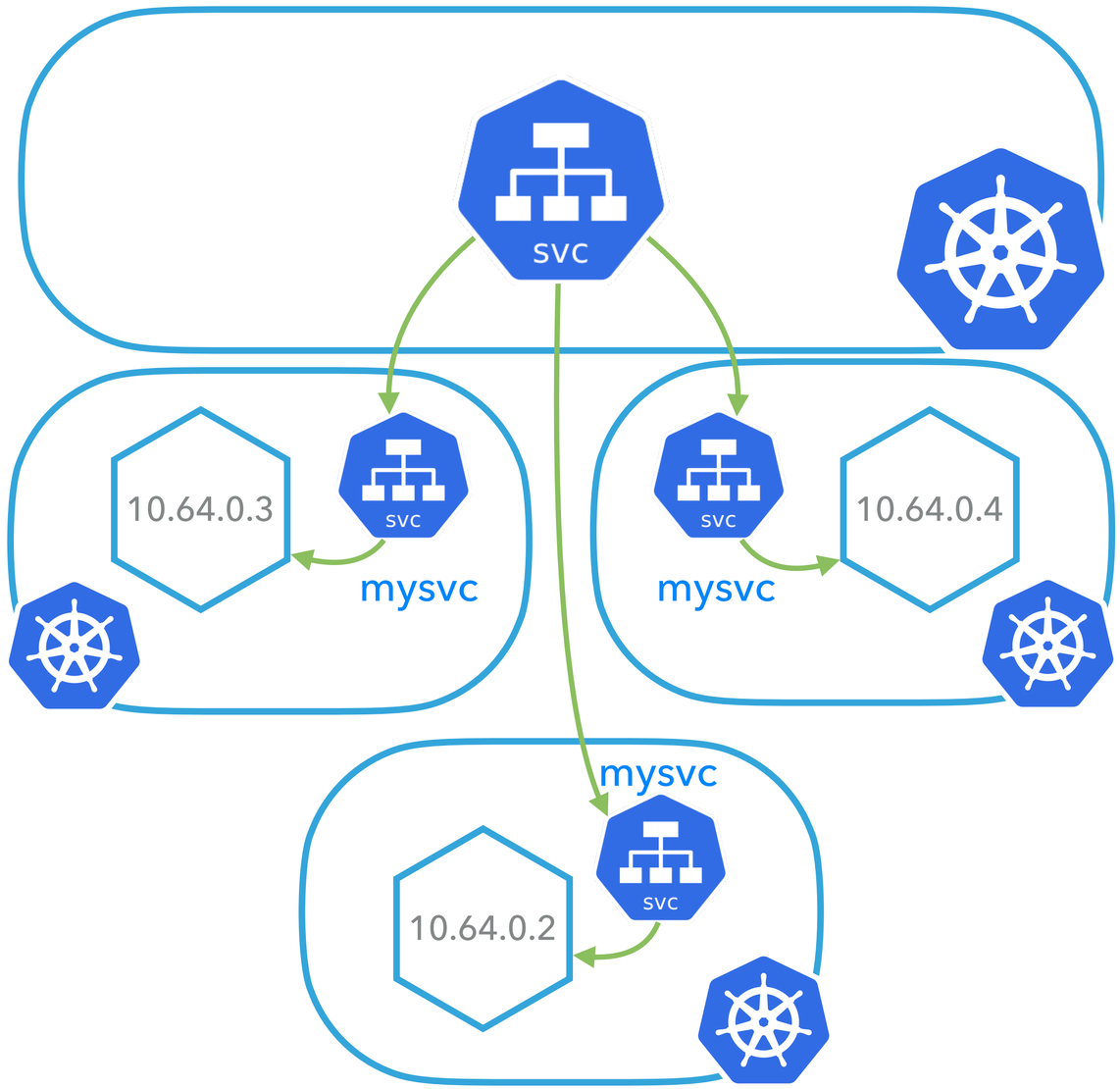
Federated Services

- This is a federated Service!
- Well, almost. It really relies on a Service in each remote cluster.



Federated Services

- This is a federated Service!
- Well, almost. It really relies on a Service in each remote cluster.
- And all the Services, in every cluster, need the same name *and namespace.*
  - This is the *namespace sameness* principle from SIG Multicluster.



Federated Services

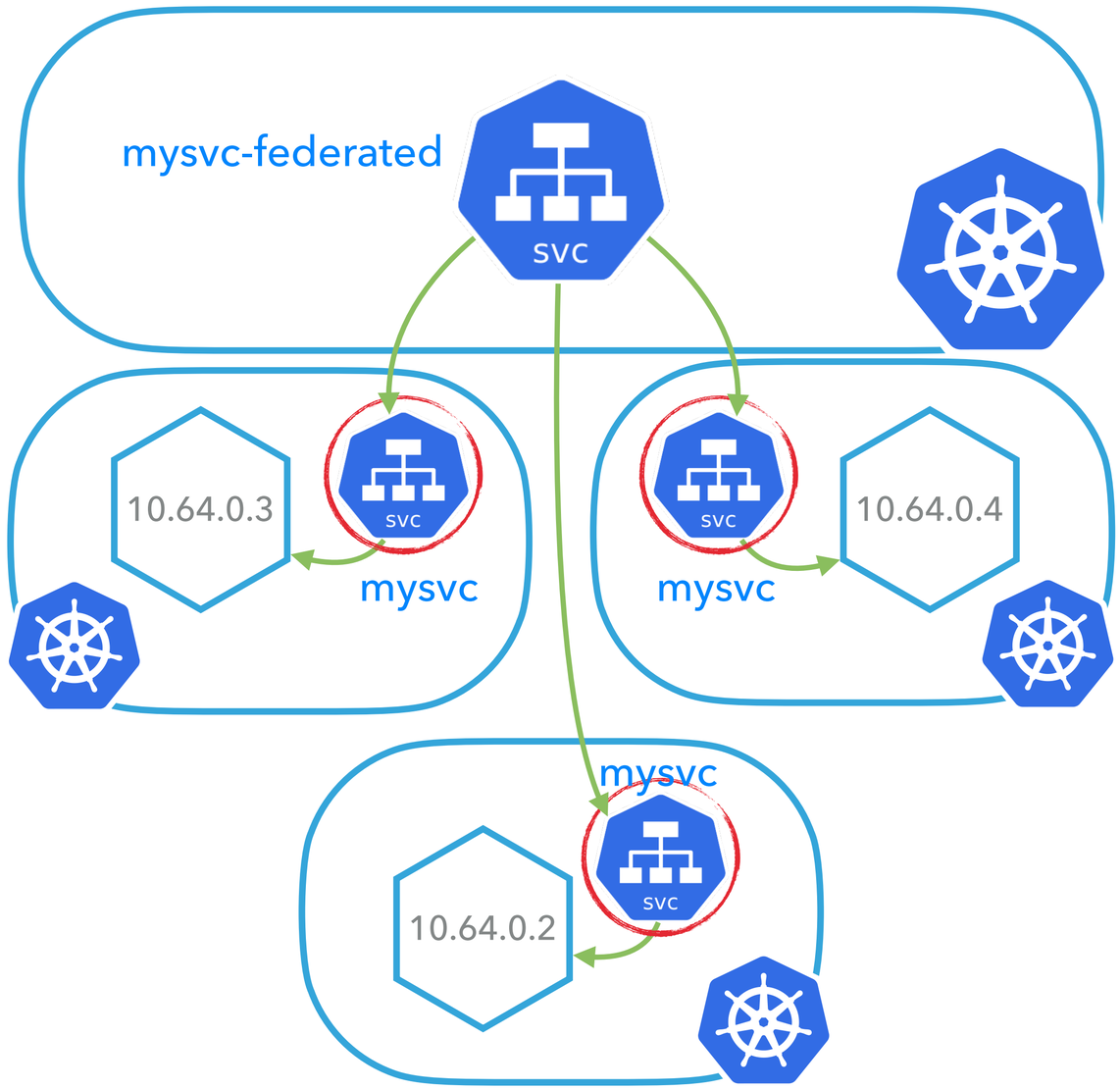
- To use federated Services, set up like pod-to-pod multicluster and have namespace sameness.
- Then label the exported Services with

  `mirror.linkerd.io/federated: member`

  instead of

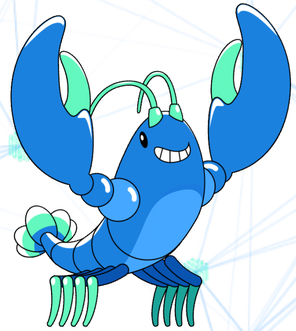
  `mirror.linkerd.io/exported: remote-discovery`

- By default, the federated Service will get a `-federated` suffix.

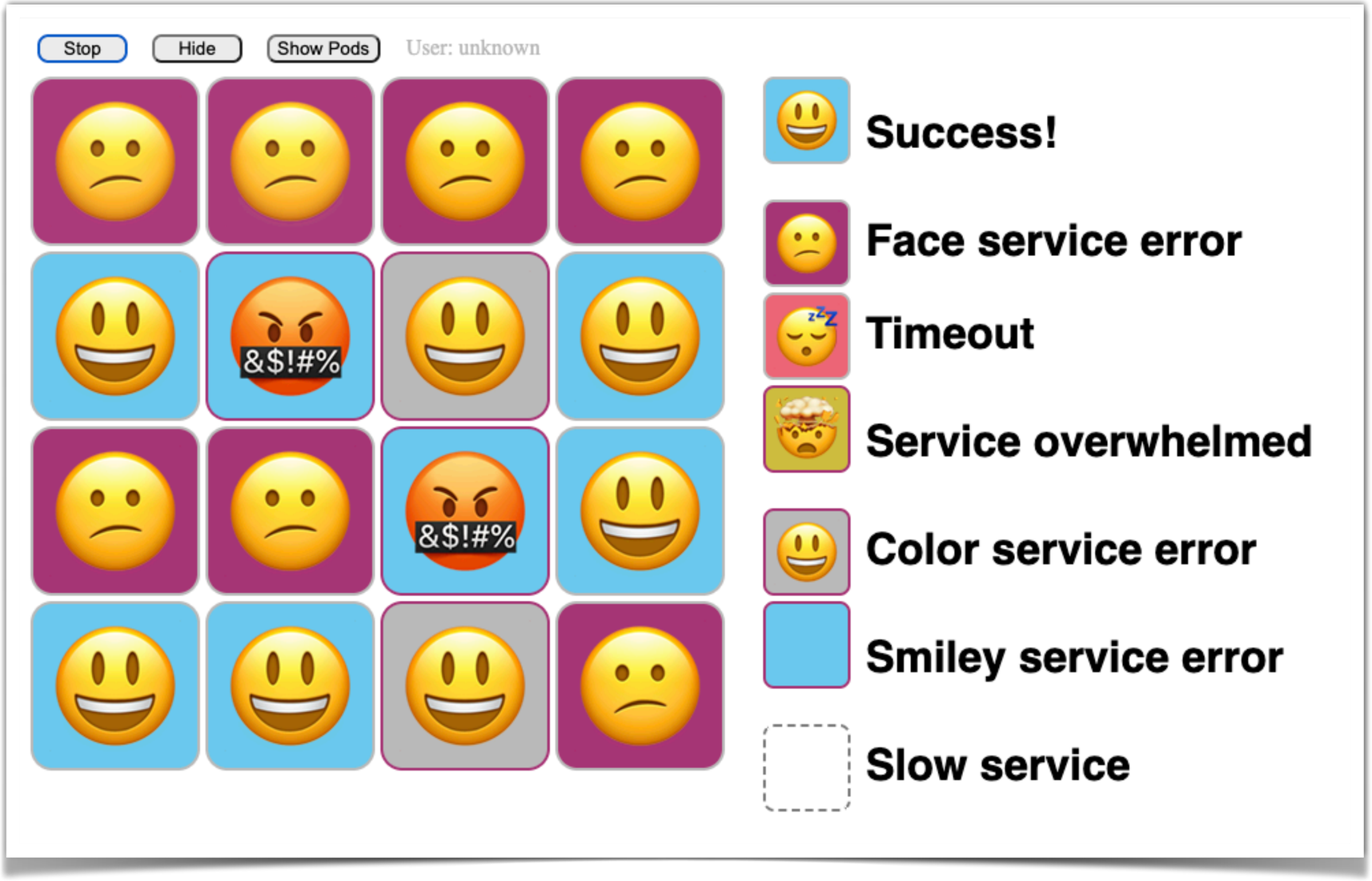


BUOYANT
SERVICE MESH
ACADEMY

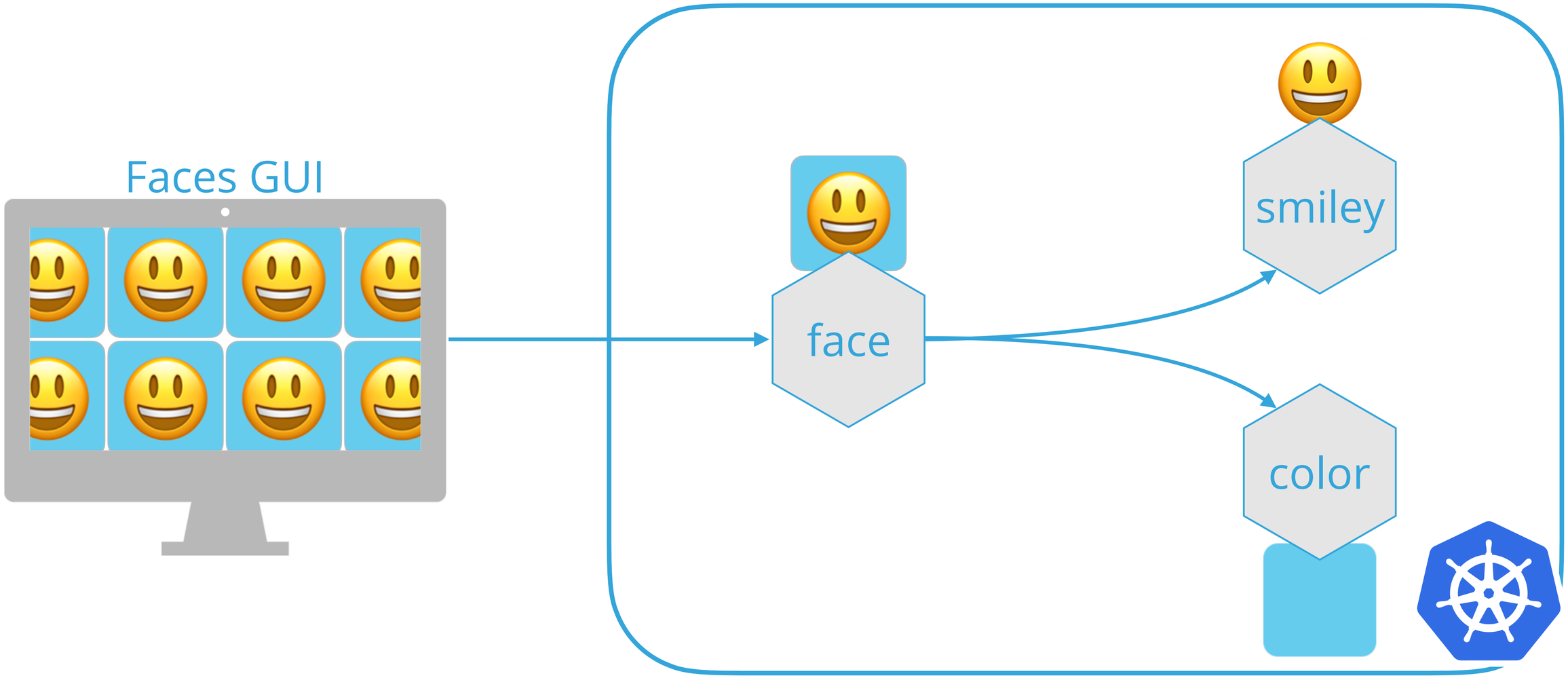
Demo Architecture



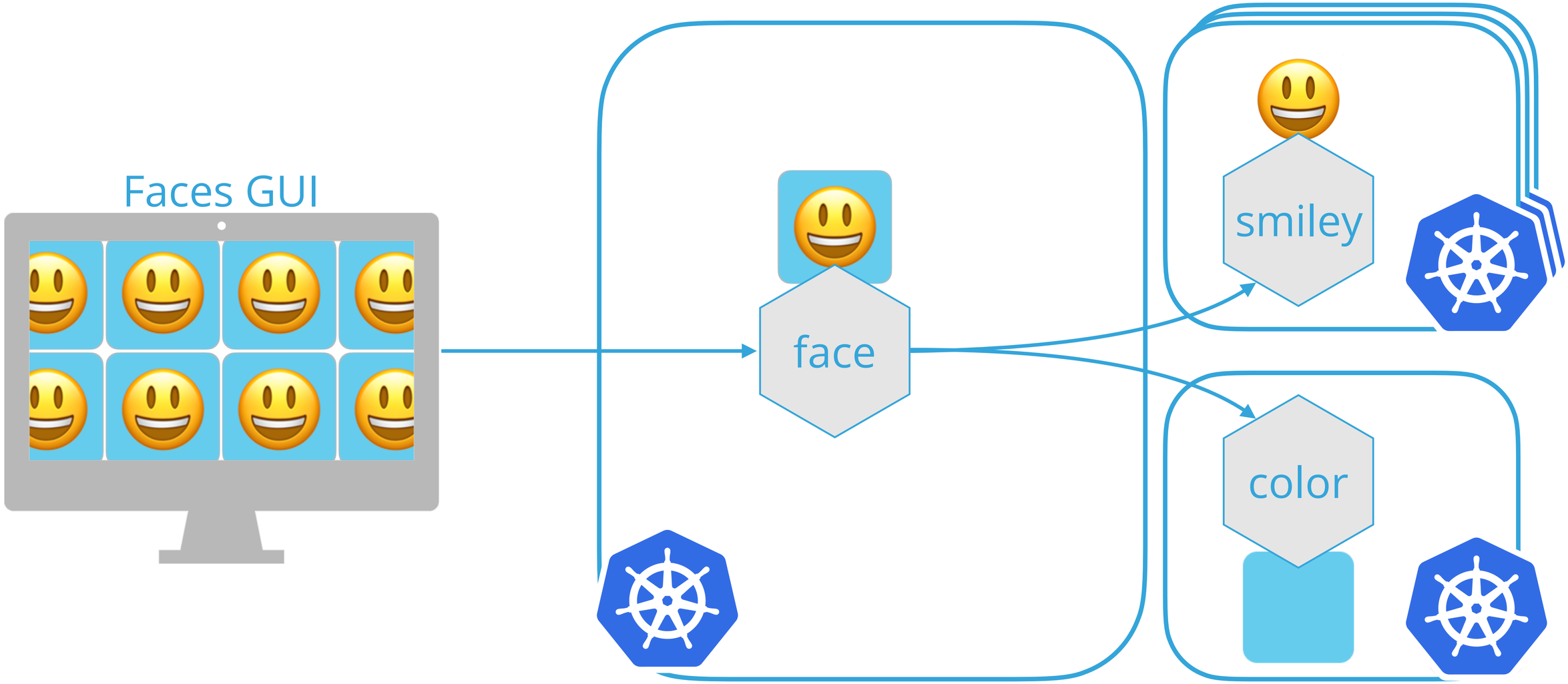
Demo Architecture



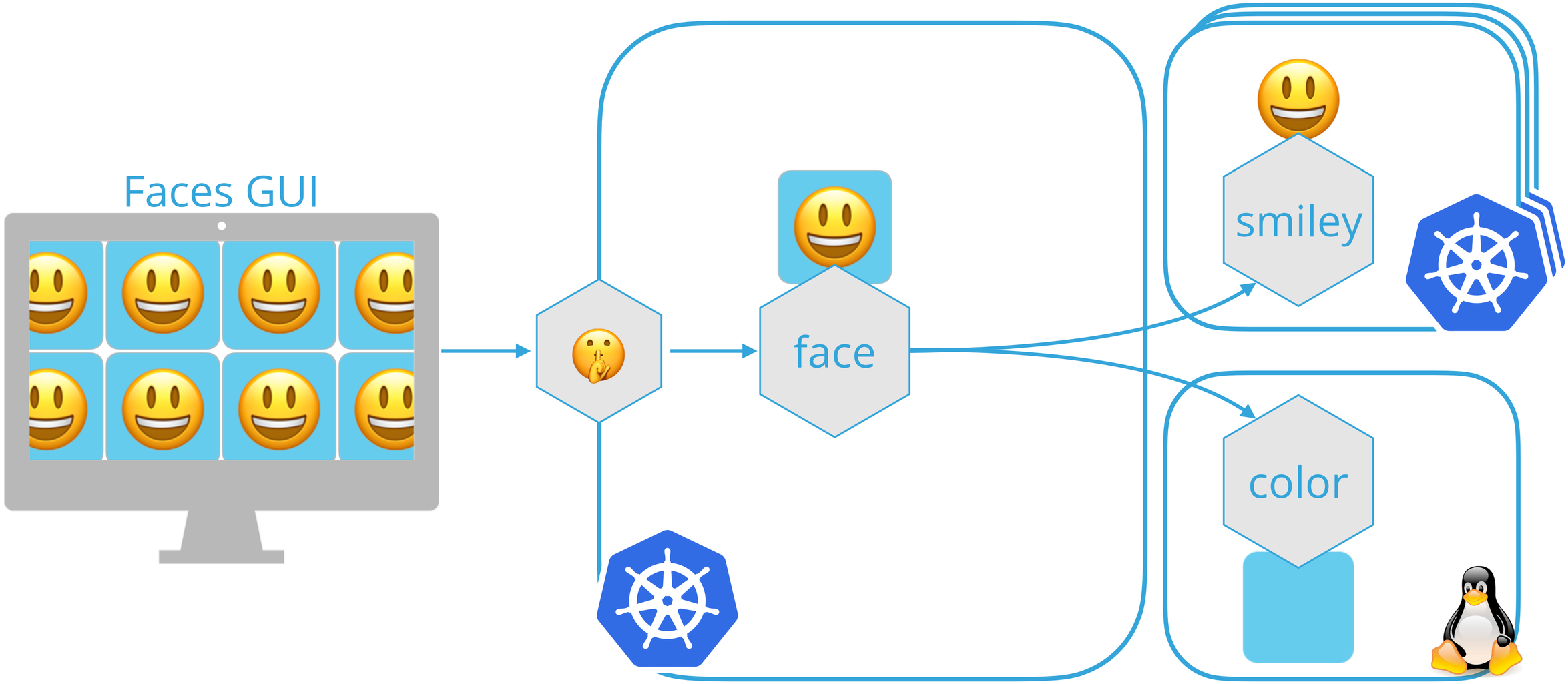
Demo Architecture



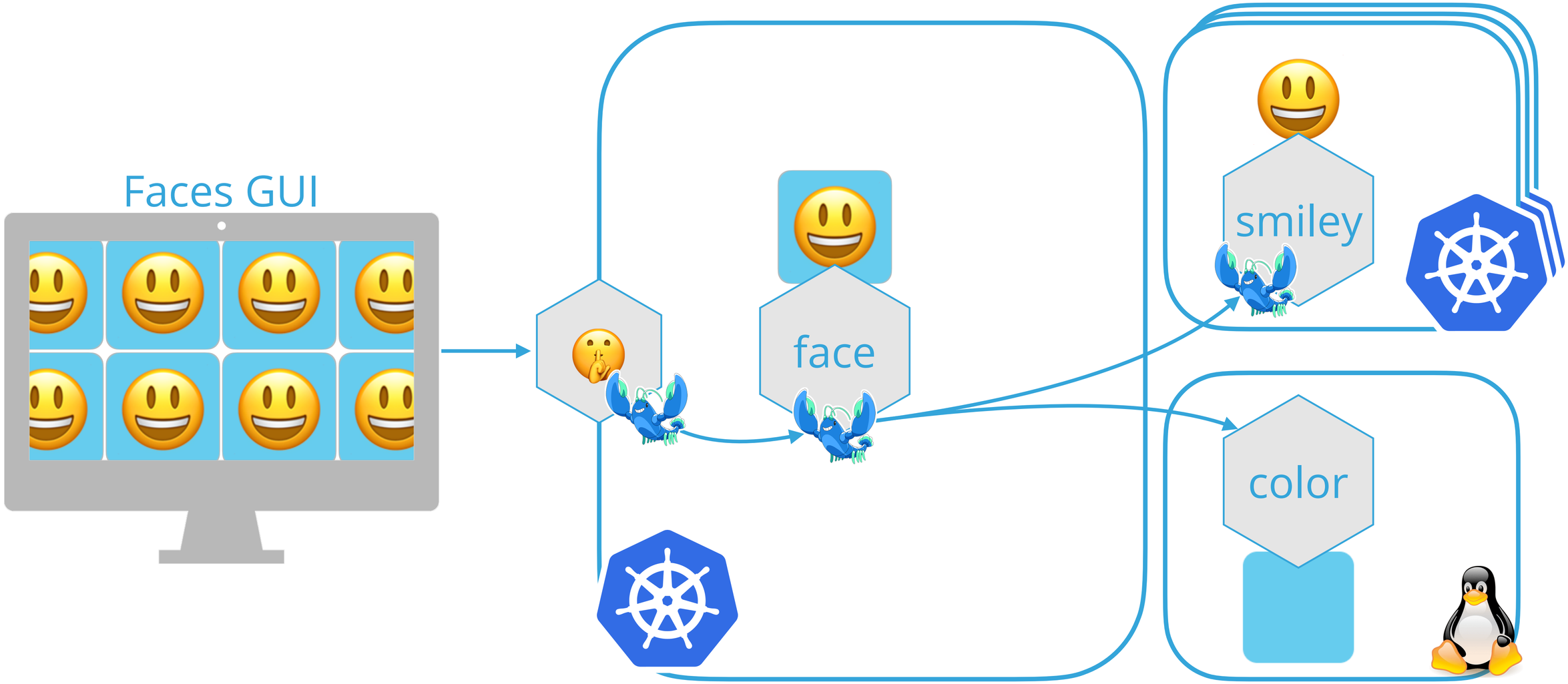
Demo Architecture



Demo Architecture



Demo Architecture

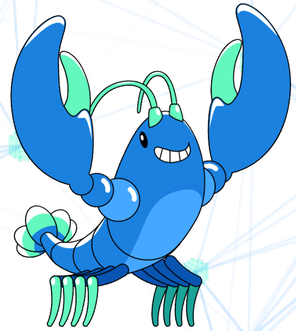


DEMO



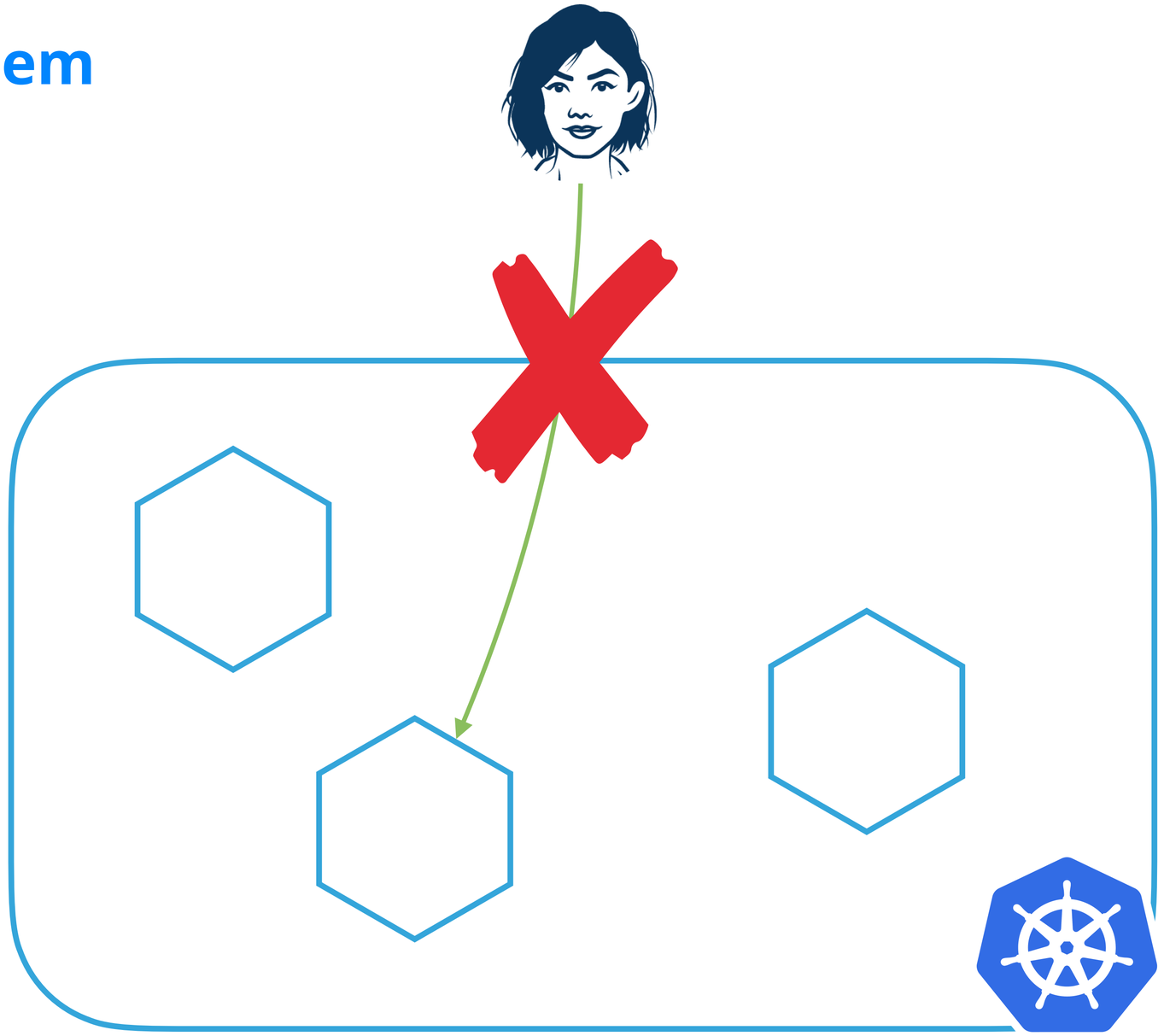
BUOYANT
SERVICE MESH
ACADEMY

Egress



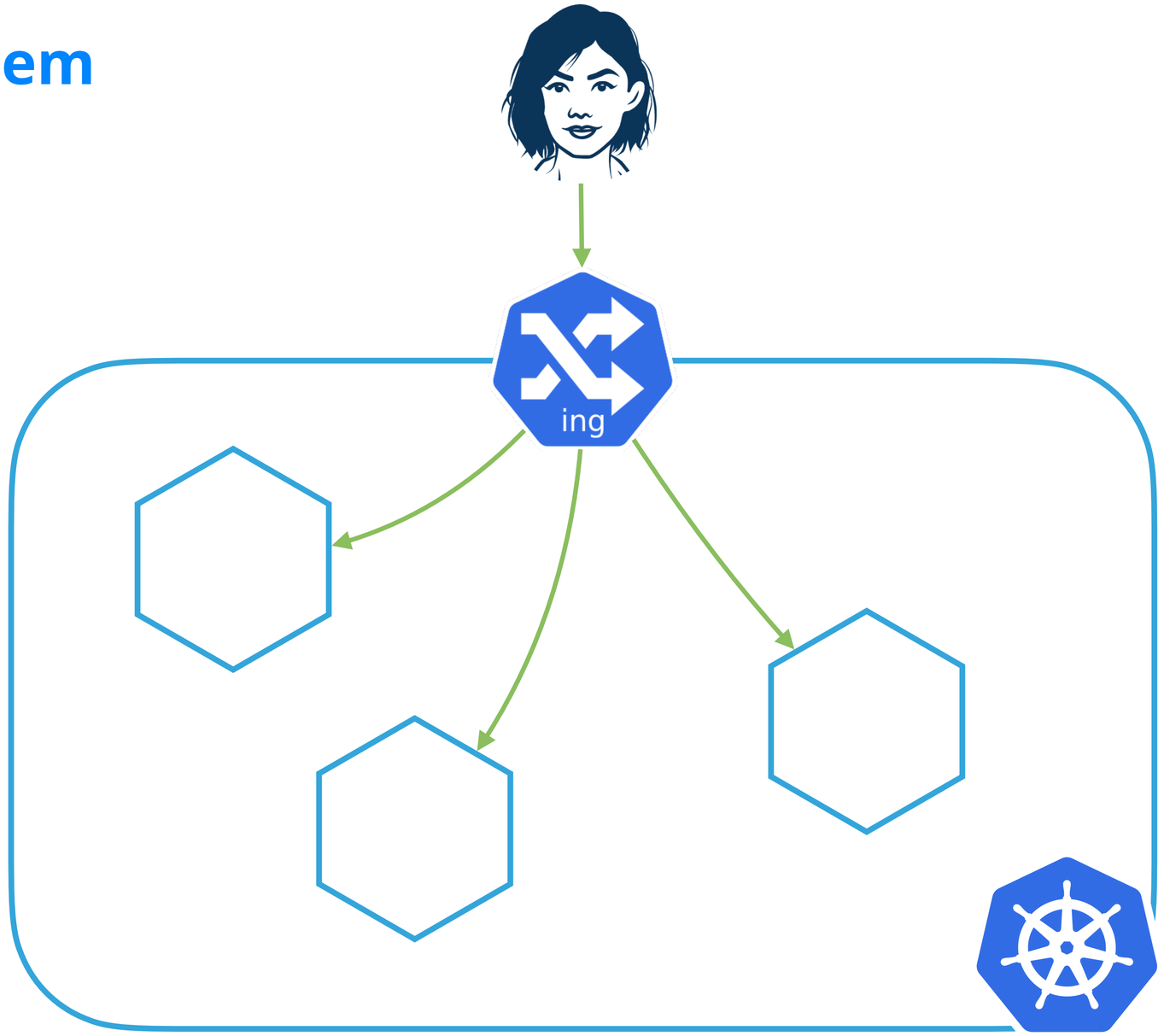
Background: the Ingress Problem

- Users need to use things in the cluster.
- But one does not simply send packets into the cluster.



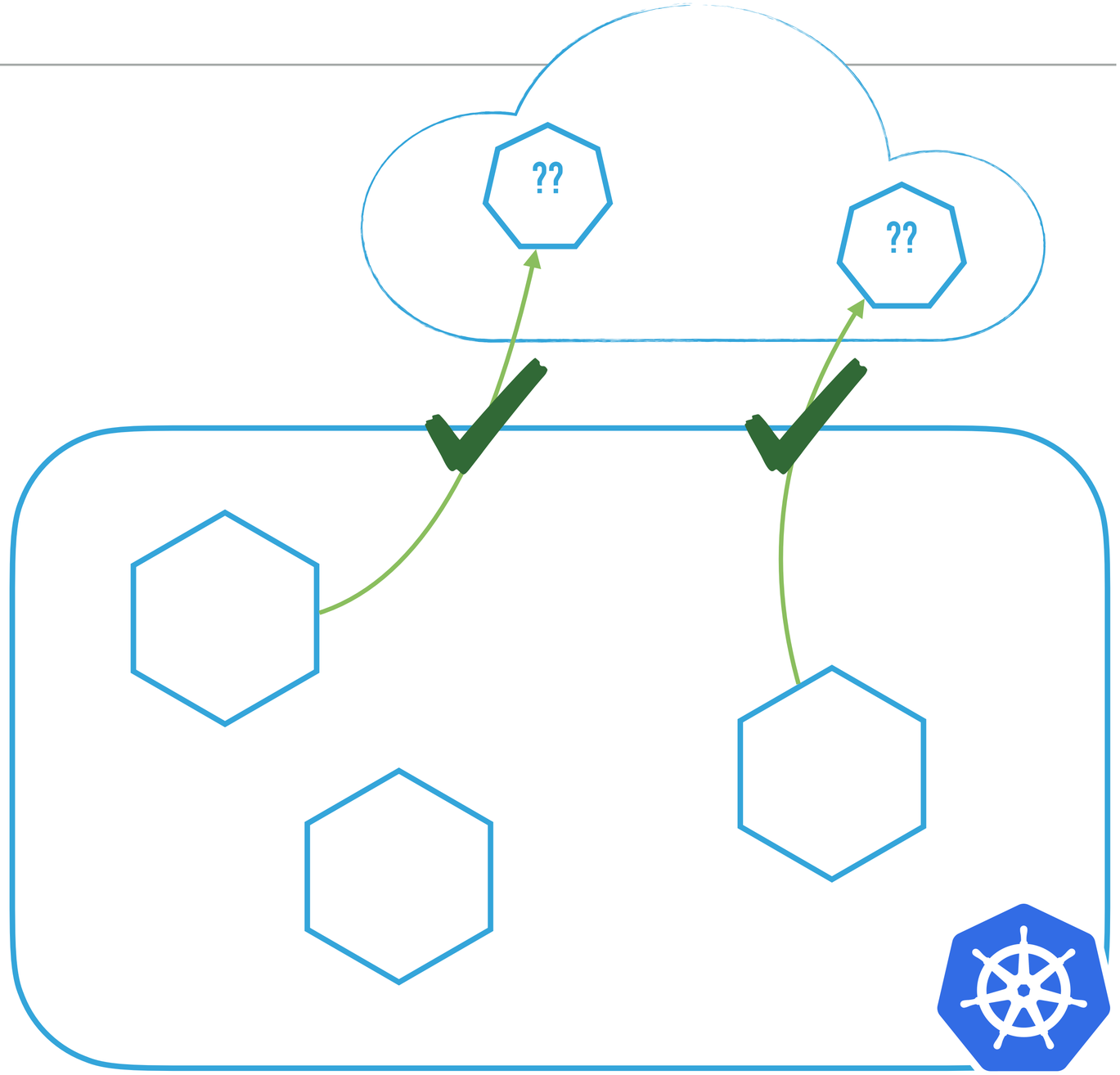
Background: the Ingress Problem

- We need a thing that allows access from outside the cluster to inside…
- …but safely.
- This thing is an *ingress controller*.
- There’s an Ingress resource, too, designed to configure ingress controllers, but that’s not really relevant for this talk.



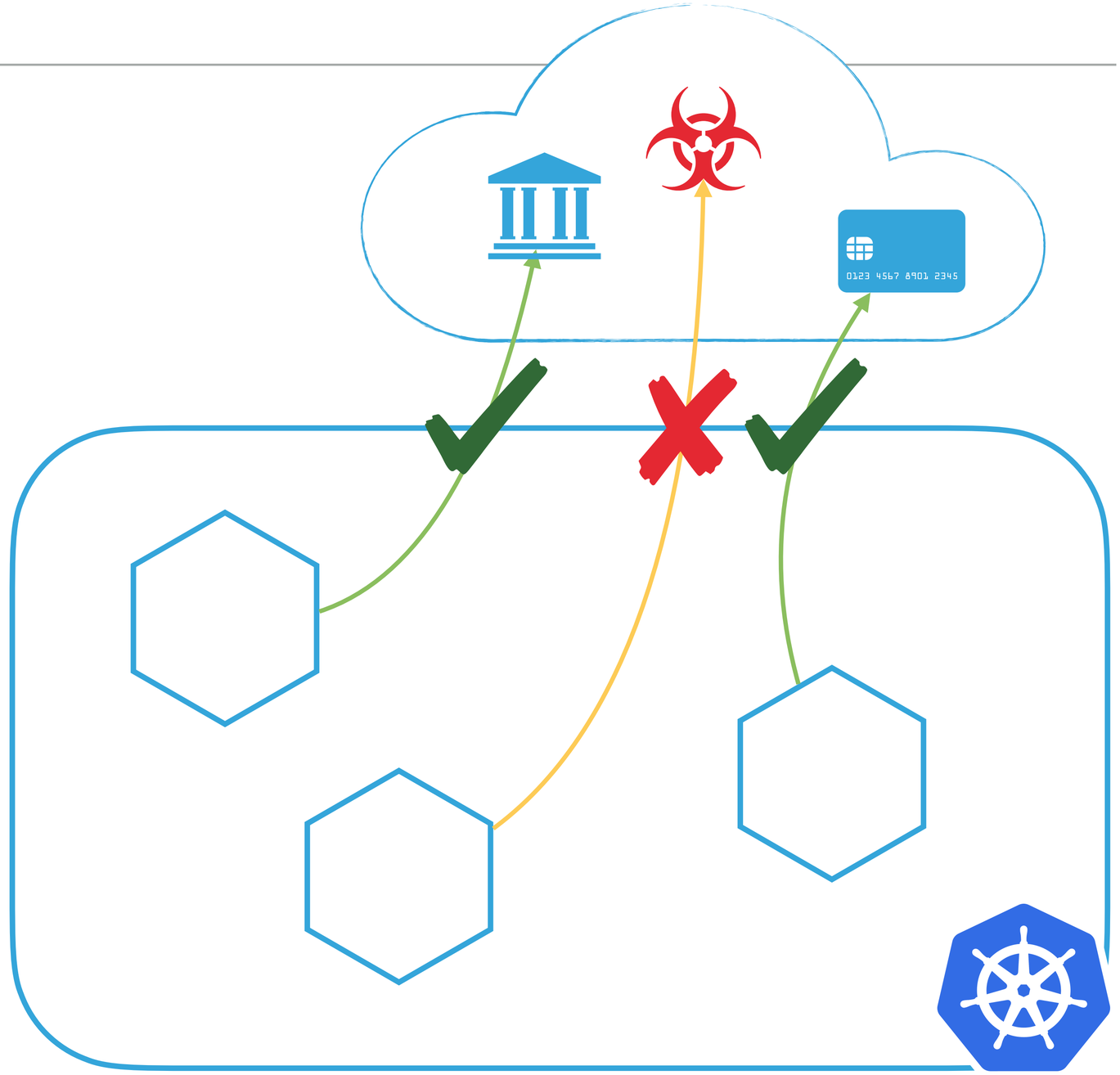
The Egress Problem

- The egress problem is the oppositie: sometimes workloads need to talk to things outside the cluster!
- This is usually fine, which is actually the problem!



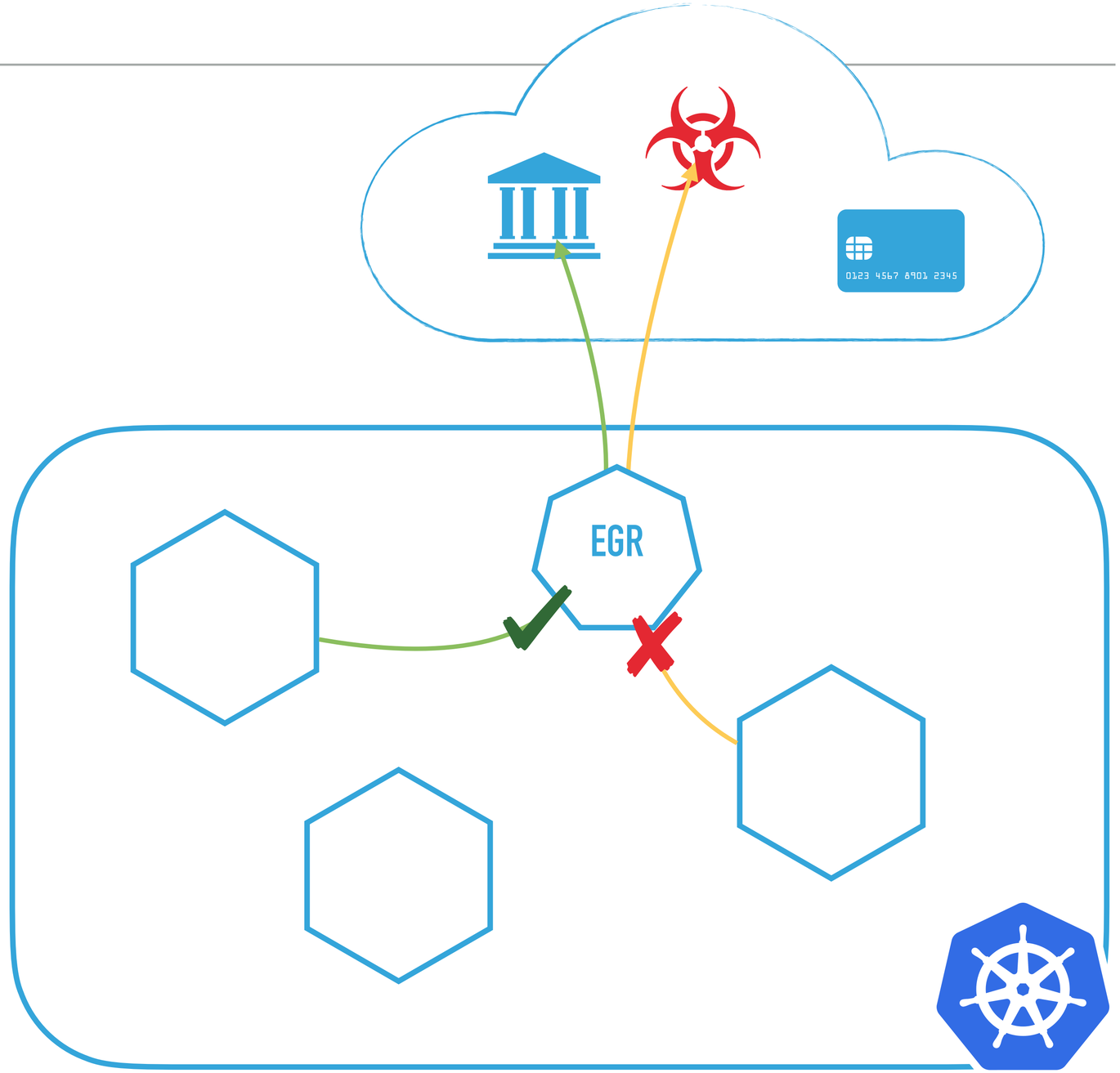
The Egress Problem

- Workloads talking to reasonable things should be allowed, but if a workload is trying to talk to an evildoer, something needs to be done.
- “Something” might just be reporting, or it might be taking action.
- Observability is a must: If you don’t even know what’s happening, there’s not much you can do!



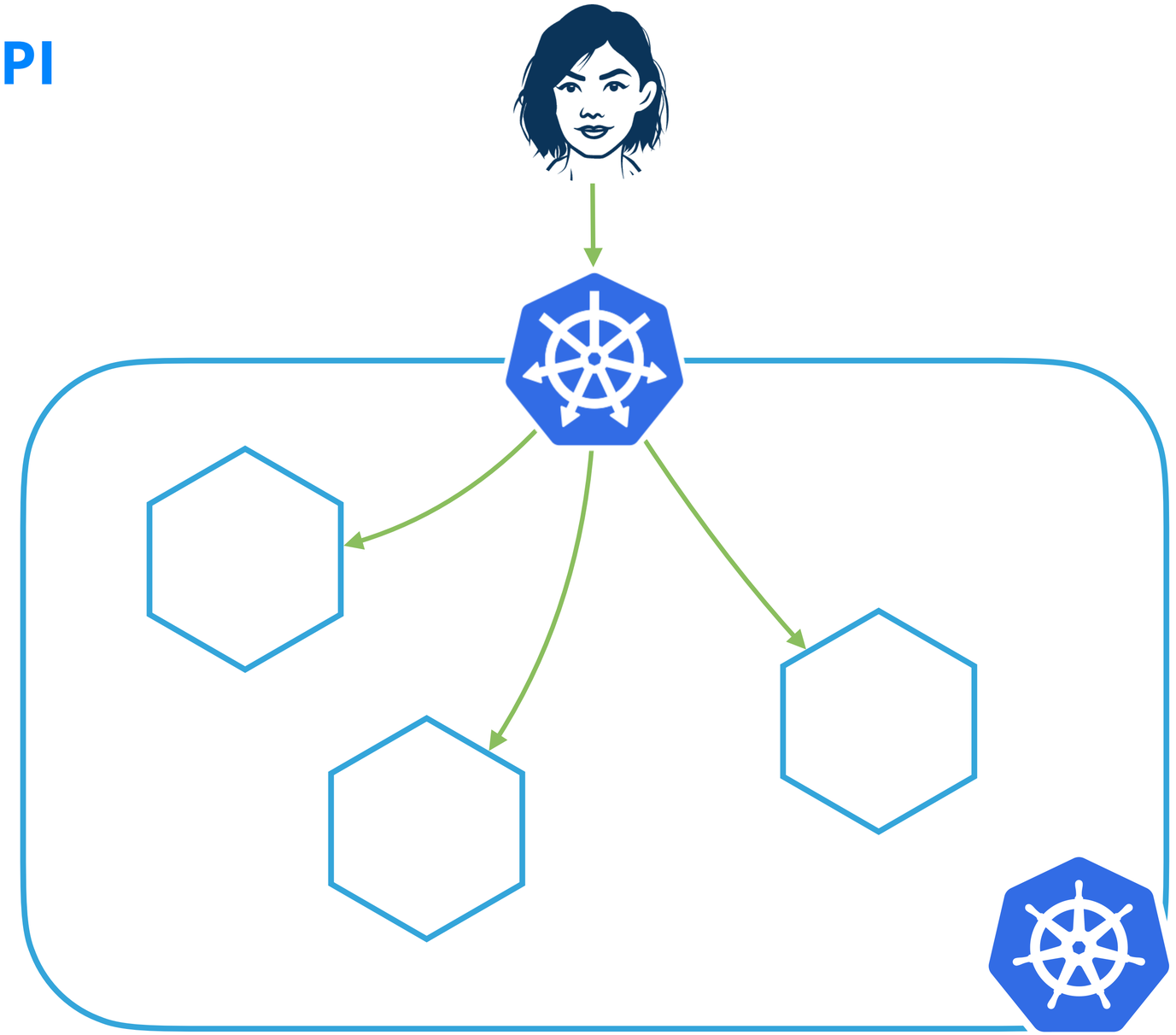
The Egress Problem

- While the obvious action is to block traffic, you might also want to force all egress traffic through an *egress controller*, which is a workload responsible for deciding whether a given request is OK.
- This is the most flexible option, and also the most complex.



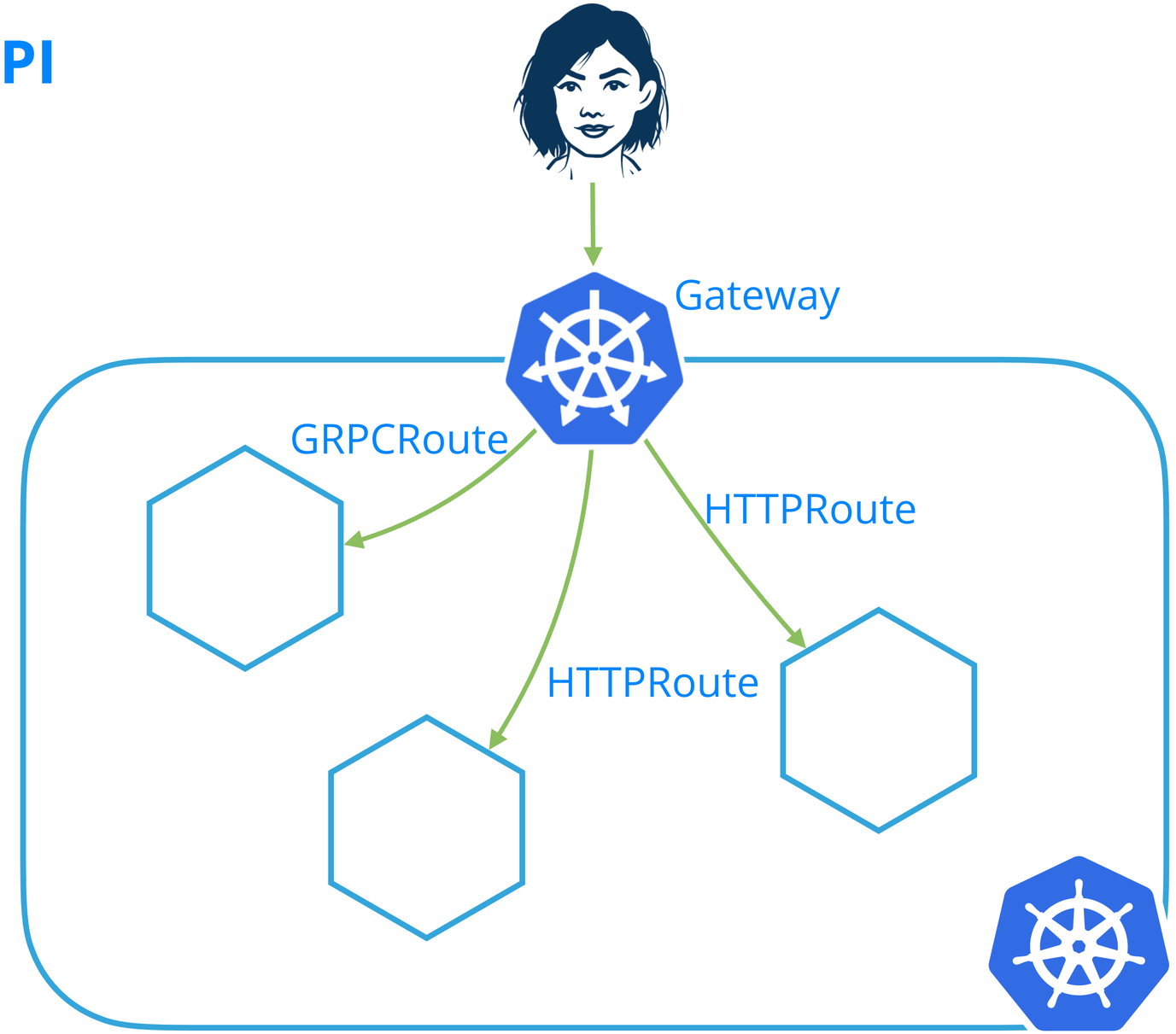
More Background: Gateway API

- Gateway API was designed to be a way of dealing with the ingress problem that was more portable than the old Ingress resource
- The core Gateway API is independent of where you're running it: learn it once, use it anywhere



More Background: Gateway API

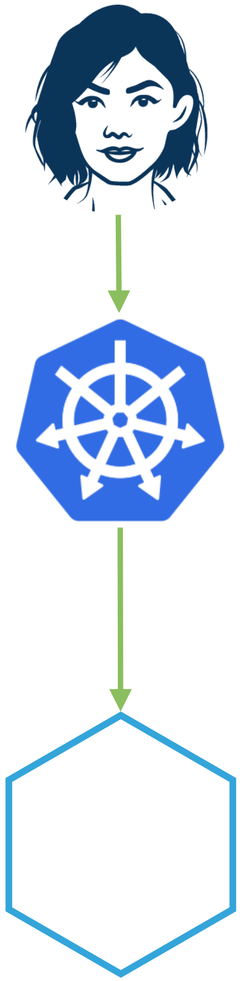
- Gateway API defines various CRDs that configure things
  - Gateway resources configure a gateway controller
  - Route resources (HTTPRoute, GRPCRoute, TLSRoute, TCPRoute) configure how traffic is routed



Gateway API for Ingress

- HTTPRoute defines how to route HTTP traffic. A simple example:
  - HTTP traffic arriving at Gateway faces-gateway…
  - with a `Host` header of `smiley.example.com`…
  - gets sent to Service `smiley`, port 80.

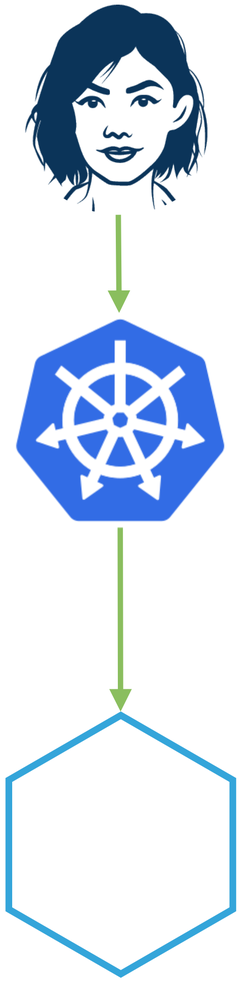
```
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: simplest-smiley-route
  namespace: faces
spec:
  parentRefs:
    - kind: Gateway
      name: faces-gateway
  hostnames:
    - smiley.example.com
  rules:
    - backendRefs:
      - name: smiley
        port: 80
```



Gateway API for Ingress

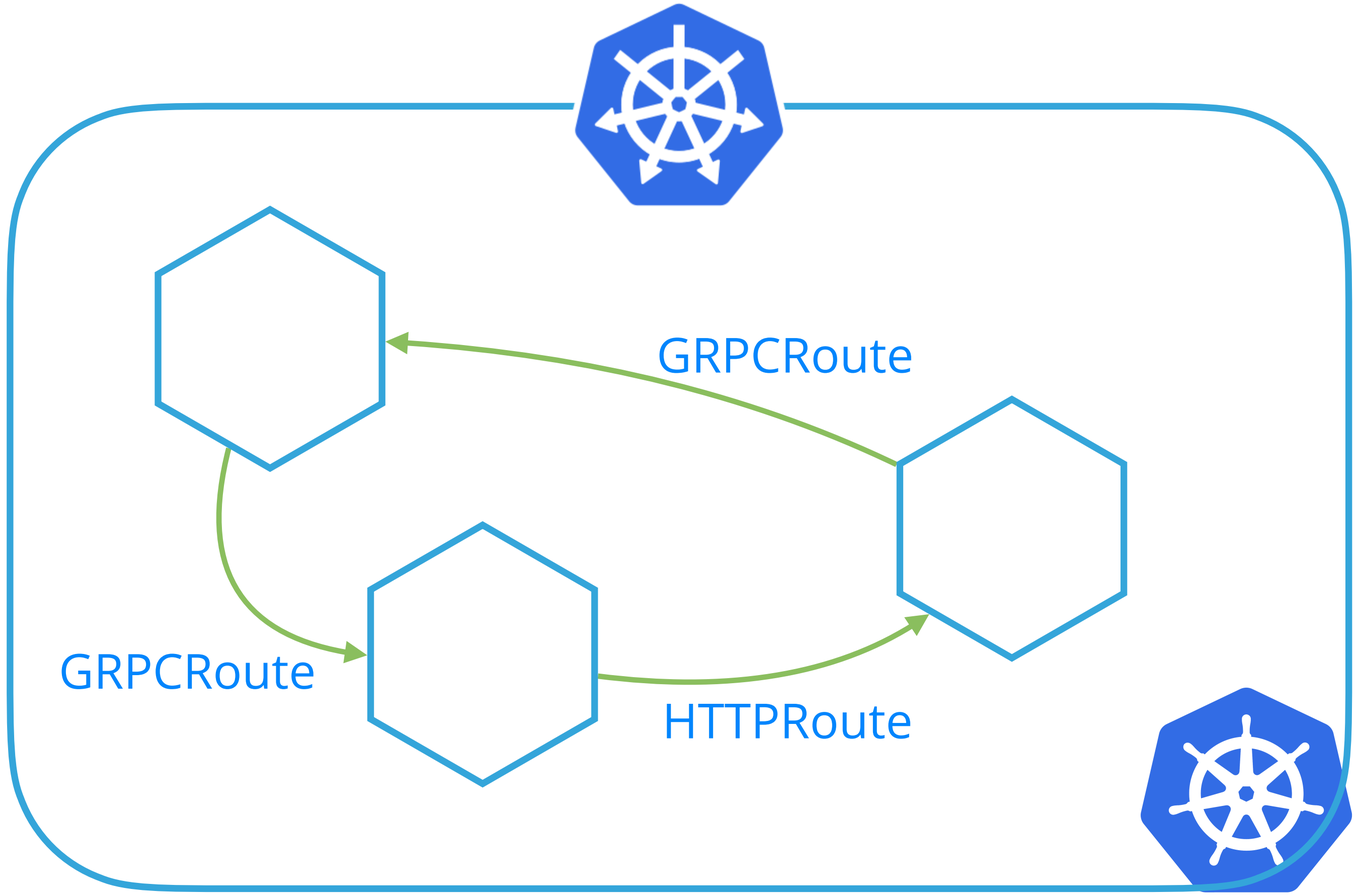
- GRPCRoute provides similar functionality but more tailored for gRPC.

```
apiVersion: gateway.networking.k8s.io/v1
kind: GRPCRoute
metadata:
  name: simplest-color-route
  namespace: faces
spec:
  parentRefs:
    - kind: Gateway
      name: faces-gateway
  hostnames:
    - color.example.com
  rules:
    - matches:
      - method:
          service: ColorService
          method: Edge
    - backendRefs:
      - name: color
        port: 80
```



Gateway API for Mesh

- But wait! These routing concepts apply pretty well to traffic between workloads in the cluster as well
- The difference isn't in what a request *is*, but in *where it's presented*



Gateway API for Mesh



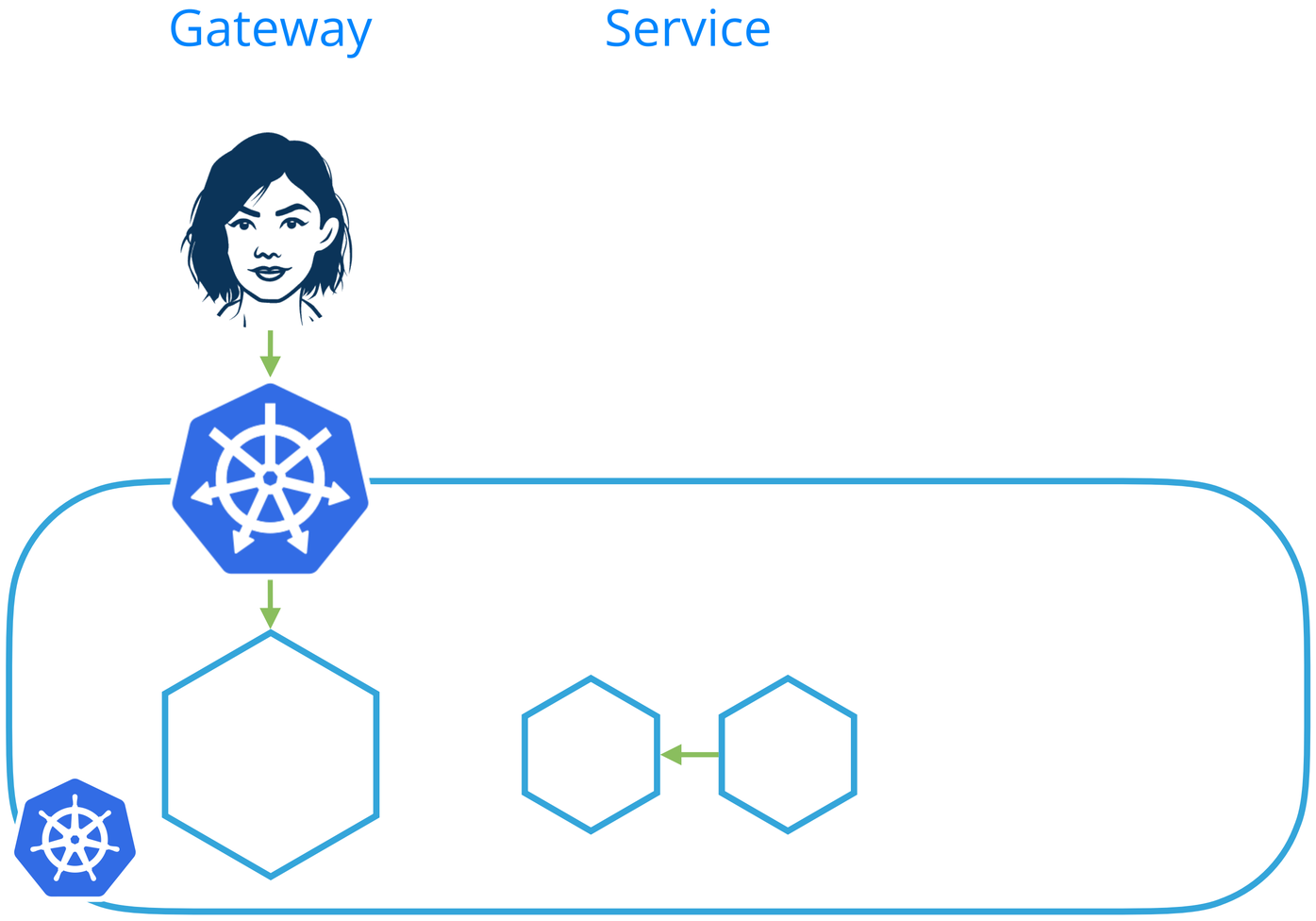
- In the mesh, requests arrive at a Service rather than a Gateway
  - This is reflected in the `parentRefs`
  - Other stuff is the same, although the mesh case might not support all the same things as the ingress case
  - (For example, the mesh is unlikely to deal with CORS.)

```
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: simplest-smiley-route
  namespace: faces
spec:
  parentRefs:
    - kind: Service
      group: ""
      name: smiley
      port: 80
  hostnames:
    - smiley.example.com
  rules:
    - backendRefs:
      - name: smiley
        port: 80
```

ROUTES ➠ CLASSES OF TRAFFIC

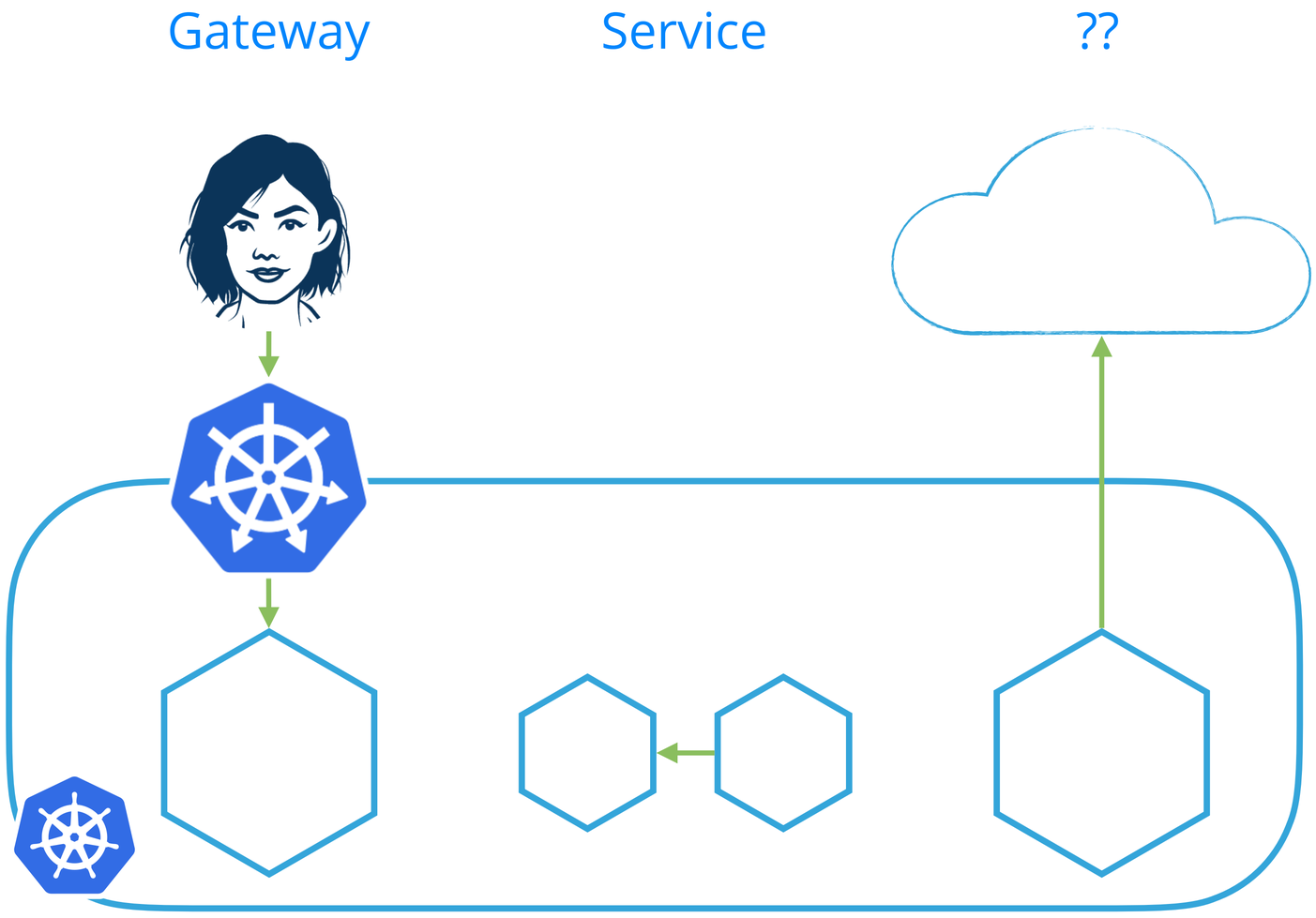
Classes of Traffic

- The `parentRef` doesn't just define a specific bit of routing, it defines *what kind of traffic* we're dealing with.
- For ingress traffic, the `parentRef` must be a Gateway.
- For mesh traffic, the `parentRef` must be a Service.



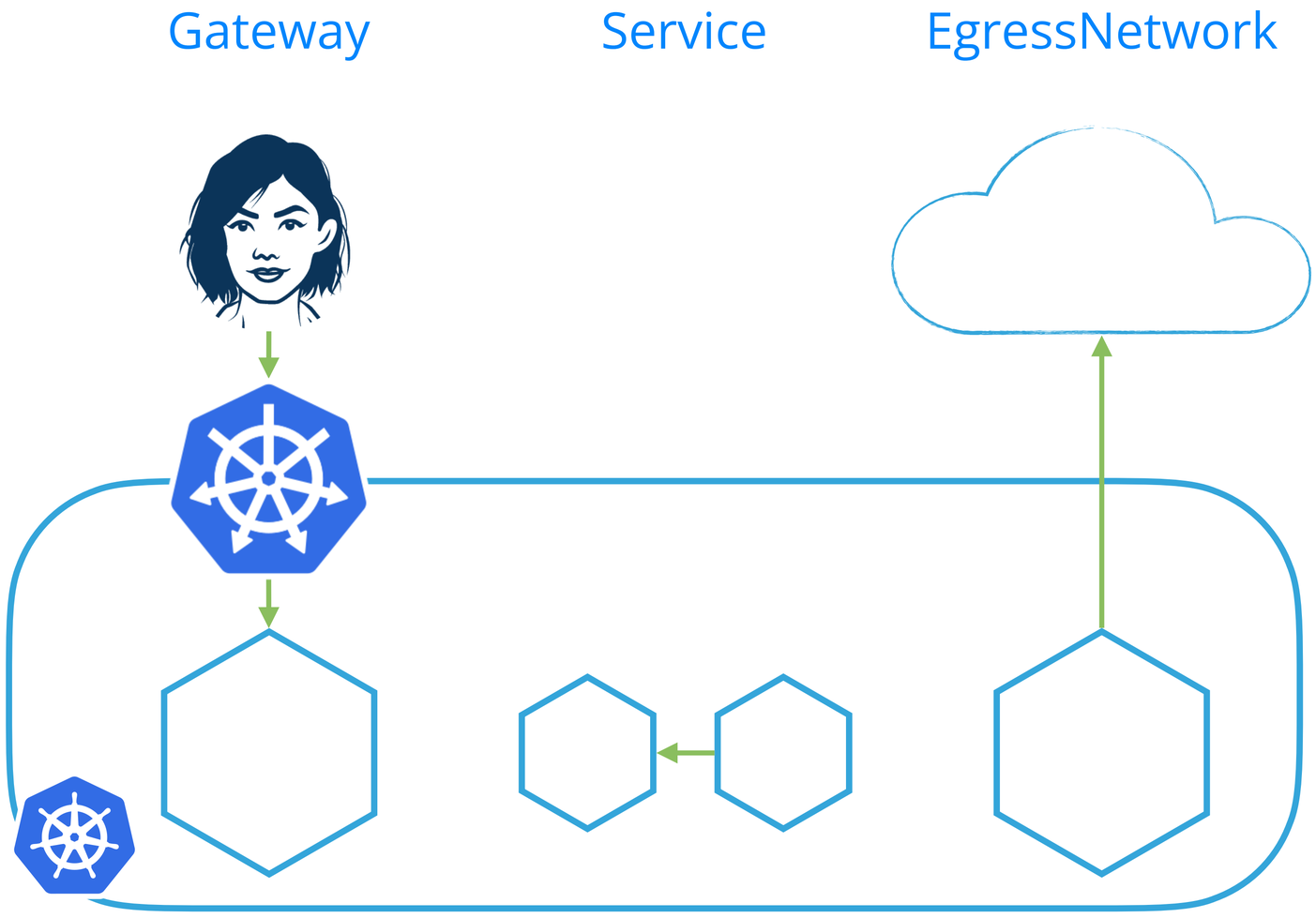
Classes of Traffic

- What about egress traffic?
  - The `parentRef` can't be a Gateway; they define specific rendezvous points associated with a cluster.
  - It can't be a Service; even though ExternalNames are a thing, you can't define the whole Internet with them.



Classes of Traffic

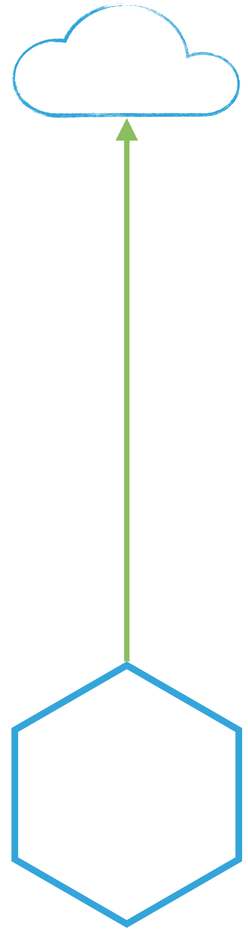
- The solution is a new thing called an **EgressNetwork**, which defines a class of traffic that is *leaving the mesh for a destination matching a particular CIDR.*
  - “Leaving the mesh” is critical: when using Linkerd multicluster, traffic between the clusters is not egress traffic.
  - Nor is traffic to Linkerd external workloads.



EgressNetworks

- EgressNetworks aren't very complex. Here's one that blocks all egress traffic.
- Note the API group here!

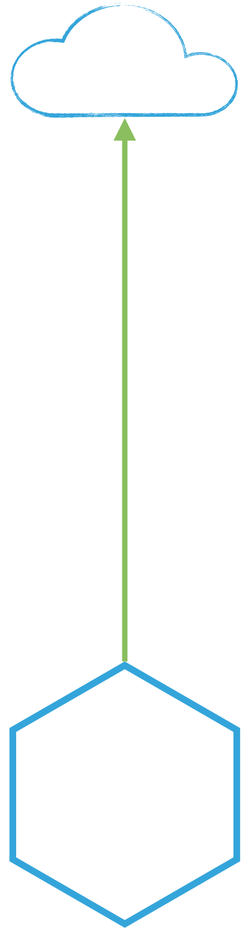
```
apiVersion: policy.linkerd.io/v1alpha1
kind: EgressNetwork
metadata:
  name: all-egress-traffic
  namespace: linkerd-egress
spec:
  trafficPolicy: Deny
```



EgressNetworks

- Here's one that blocks only traffic to a specific network.
- Note that these are in namespace linkerd-egress.
  - That's a special "global egress" namespace (the name is configurable).
  - Egress traffic from a namespace without its own EgressNetwork will use this EgressNetwork.

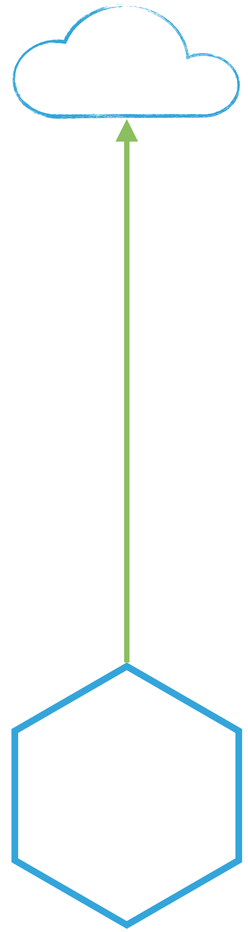
```
apiVersion: policy.linkerd.io/v1alpha1
kind: EgressNetwork
metadata:
  name: all-egress-traffic
  namespace: linkerd-egress
spec:
  trafficPolicy: Deny
  networks:
    cidr: "10.0.0.8/8"
```



EgressNetworks

- In turn, we can use an EgressNetwork as a `parentRef` to describe egress traffic that's allowed.
  - The `parentRef` group is very important.
  - The `parentRef` port defines which port we want to allow this traffic to use.
  - The usual matches are supported.

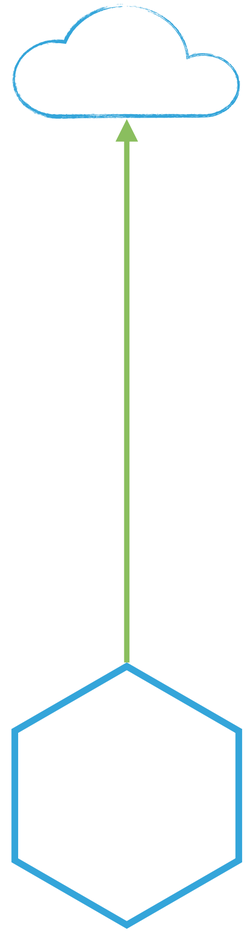
```
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: httpbin-egress
  namespace: default
spec:
  parentRefs:
    - name: all-egress-traffic
      namespace: linkerd-egress
      kind: EgressNetwork
      group: policy.linkerd.io
      port: 80
  hostnames:
    - smiley.example.com
```



EgressNetworks

- We can also use a `backendRef` to force this egress traffic through an egress controller.
- This works for all the route types, though you need to be careful to explicitly allow egress directly from the egress controller!

```yaml
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: httpbin-egress
  namespace: default
spec:
  parentRefs:
    - name: all-egress-traffic
      namespace: linkerd-egress
      kind: EgressNetwork
      group: policy.linkerd.io
      port: 80
  hostnames:
    - smiley.example.com
  backendRefs:
    - name: egress-controller
      namespace: egress-controller-ns
```

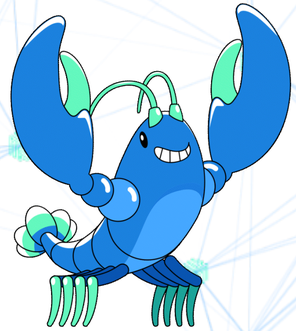


DEMO



BUOYANT
SERVICE MESH
ACADEMY

GOTCHAS



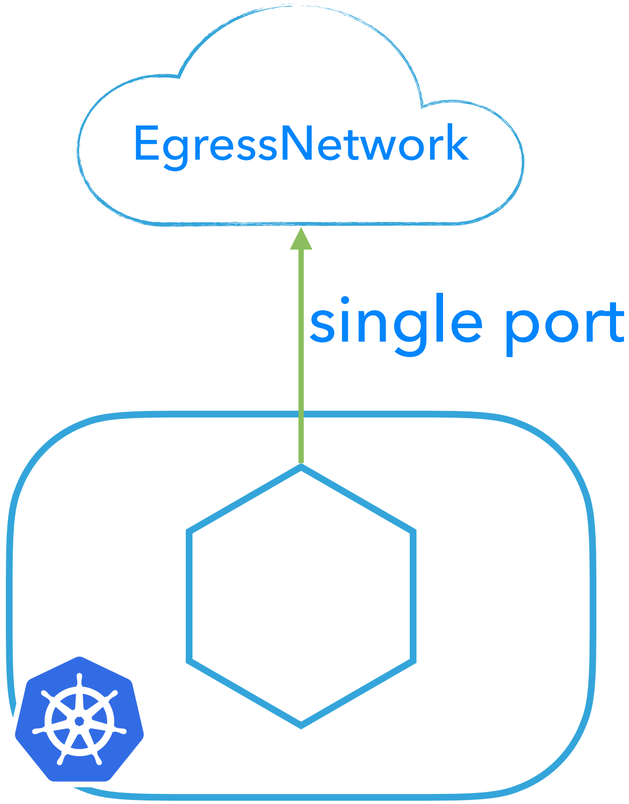
Gotchas

- If you don't give a CIDR for an EgressNetwork, it defaults to "everything but the `ClusterNetworks` you set up when you installed Linkerd"
  - `ClusterNetworks` defaults to "all the RFC1918 networks" so you'll likely need to override that to have usable IPv4 egress
  - We recommend explicitly setting `ClusterNetworks` to the union of your PodCIDR, ServiceCIDR, and NodeCIDR (for all clusters in a multicluster!)



Gotchas

- Any Routes with an EgressNetwork parentRef need to specify a port in the parentRef
- Also, EgressNetworks in namespaces **completely replace** any global default EgressNetworks



Gotchas

- TCPRoute and TLSRoute, per Gateway API, require at least one backendRef

```
apiVersion: gateway.networking.k8s.io/v1
kind: TCPRoute
metadata:
  name: egress-tcproute
  namespace: faces
spec:
  parentRefs:
    - kind: EgressNetwork
      group: policy.linkerd.io
      name: default-egress
      namespace: linkerd-egress
      port: 997
 rules:
    - backendRefs:
      - ???
```

Gotchas

- TCPRoute and TLSRoute, per Gateway API, require at least one `backendRef`
  - If you don't want an egress controller, use the EgressNetwork as the `backendRef`

```
apiVersion: gateway.networking.k8s.io/v1
kind: TCPRoute
metadata:
  name: egress-tcproute
  namespace: faces
spec:
  parentRefs:
    - kind: EgressNetwork
      group: policy.linkerd.io
      name: default-egress
      namespace: linkerd-egress
      port: 997
 rules:
    - backendRefs:
      - kind: EgressNetwork
        group: policy.linkerd.io
        name: default-egress
        namespace: linkerd-egress
```

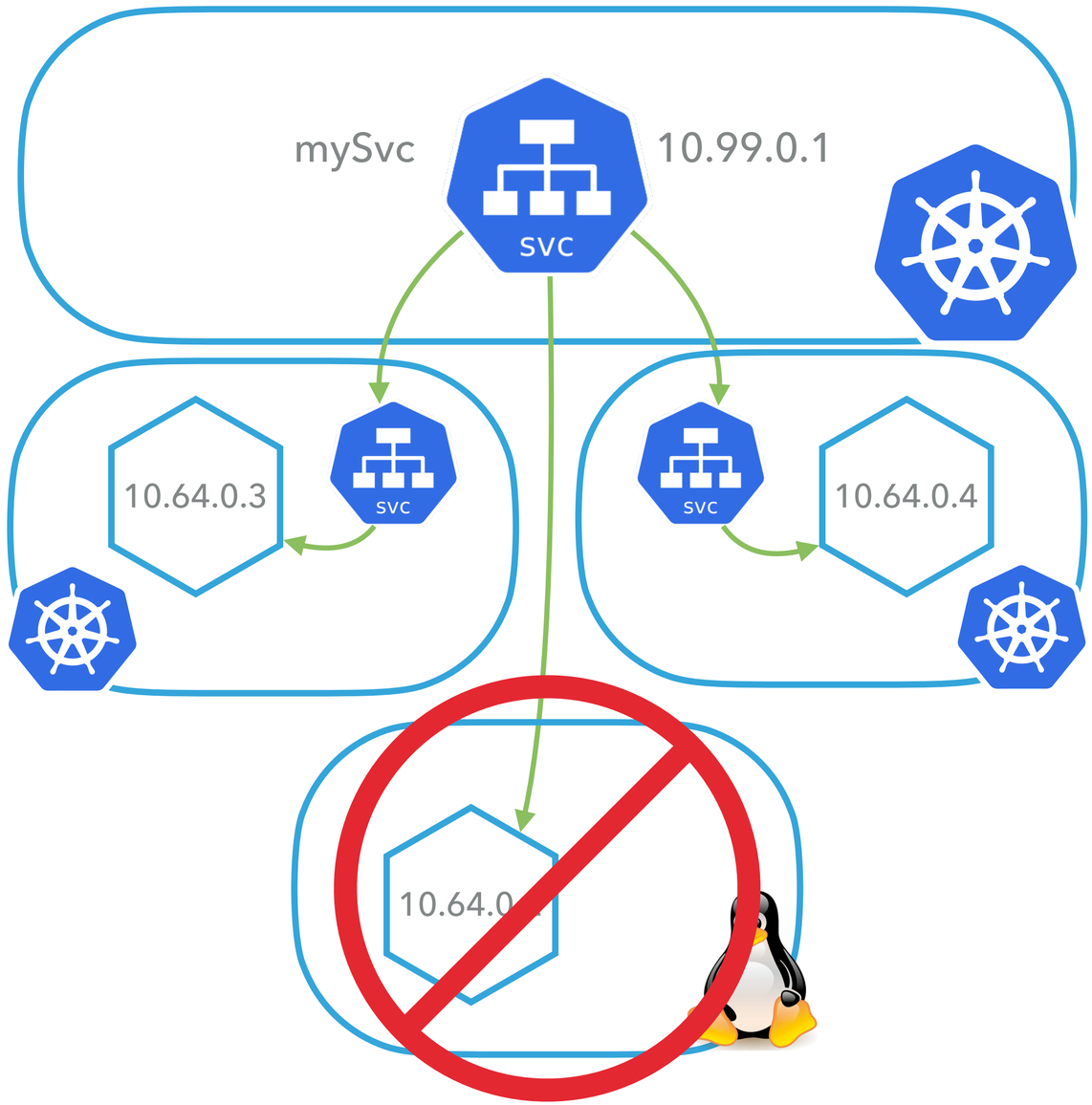
Gotchas

- TCPRoute and TLSRoute, per Gateway API, require at least one backendRef
  - If you don't want an egress controller, use the EgressNetwork as the `backendRef`
  - You must use the same EgressNetwork as the `parentRef`!

```
apiVersion: gateway.networking.k8s.io/v1
kind: TCPRoute
metadata:
  name: egress-tcproute
  namespace: faces
spec:
  parentRefs:
    - kind: EgressNetwork
      group: policy.linkerd.io
      name: default-egress
      namespace: linkerd-egress
      port: 997
 rules:
    - backendRefs:
      - kind: EgressNetwork
        group: policy.linkerd.io
        name: default-egress
        namespace: linkerd-egress
```

Gotchas

- Federated Services and egress **don't compose**.
- This is because federated Services are **Services**, and egress (by definition) can't be a Service.
- Also, remember that Route-based authorization can be tricky with federated Services, and isn't supported for egress traffic.



Federated Services and Egress



Buoyant Enterprise for LINKERD

BUOYANT SERVICE MESH ACADEMY

