Hello and Welcome to “Inside Kubernetes Ingress”, a KubeCon and CloudNativeCon North America 2020 presentation

I am Dominik Tornow, Principal Engineer at Cisco and I focus on systems modeling, specifically conceptual and formal modeling to support the development and documentation of complex software systems.

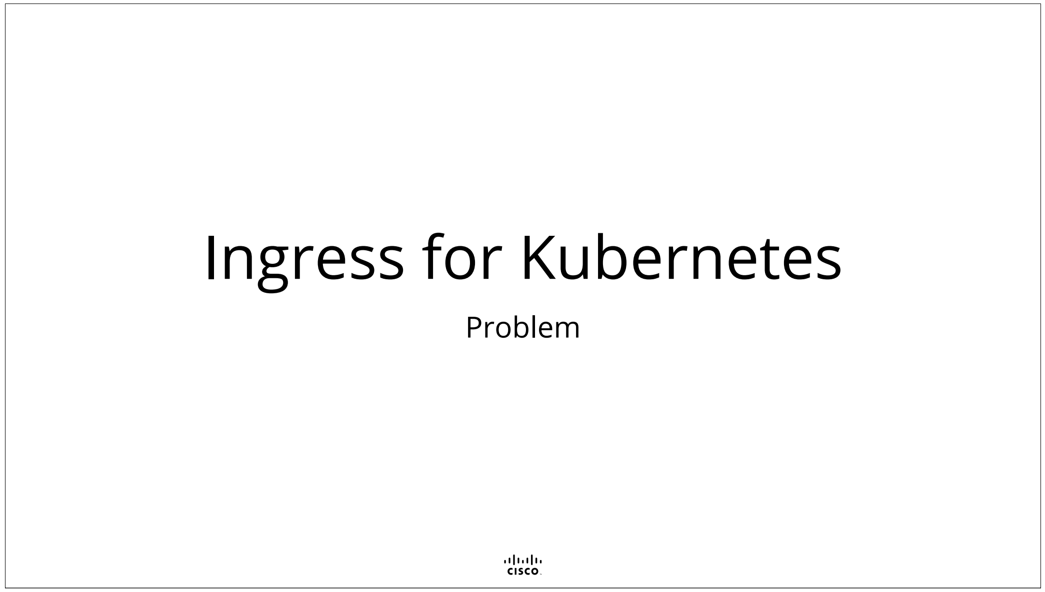
This presentation focuses on the concepts behind Ingress for Kubernetes, it does not focus on its possible implementations or on its possible features

Inside Kubernetes Services

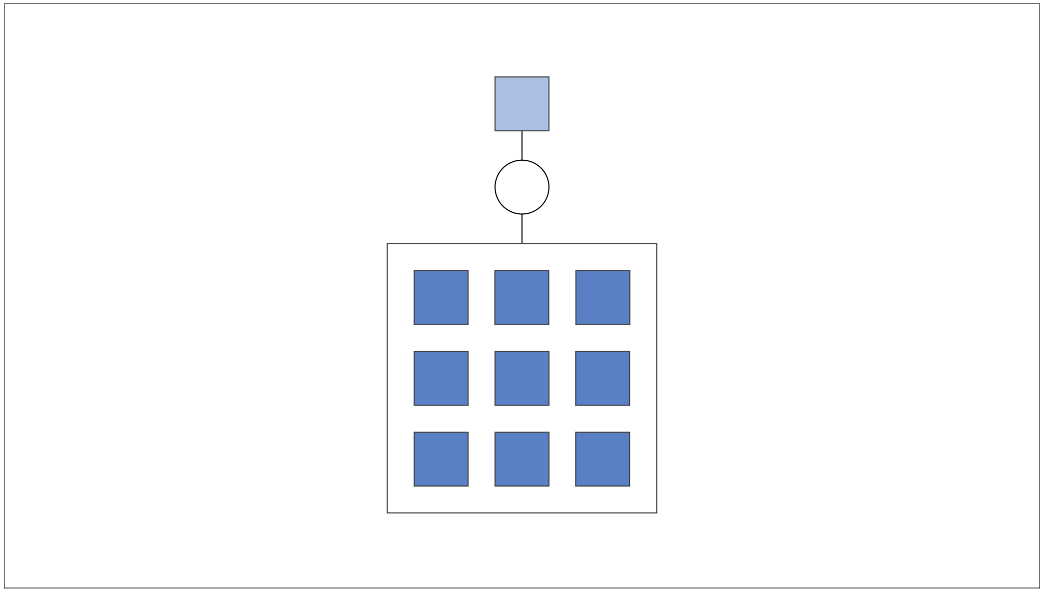
KubeCon + CloudNativeCon North America 2019

https://youtu.be/Hk77mToouEI

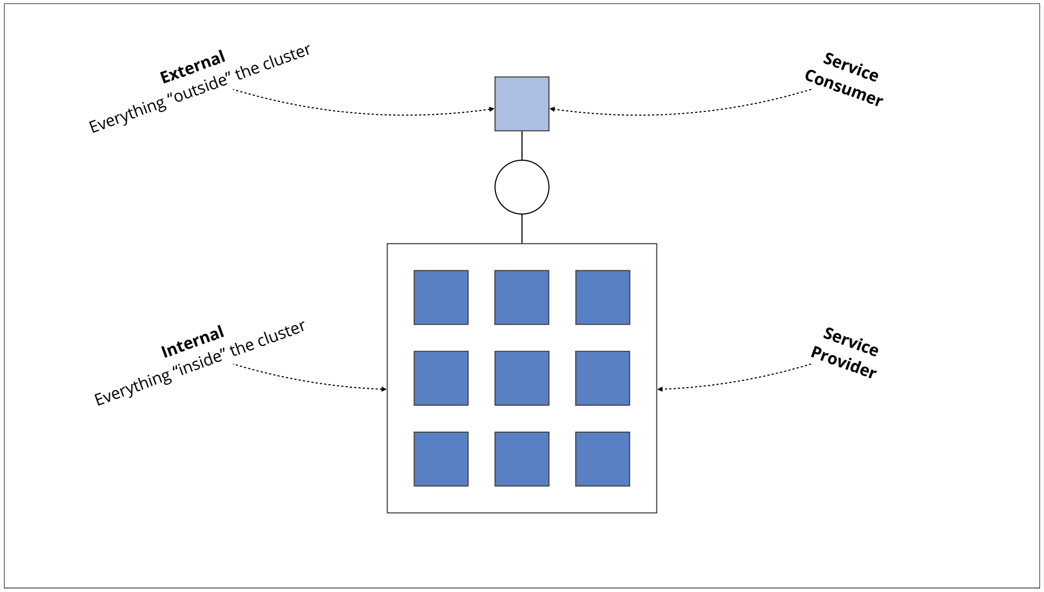
Kubernetes Ingress is related to Kubernetes Services. To deep dive into Kubernetes Services visit Inside Kubernetes Services, A KubeCon and Cloud Native Con North America 2019 presentation



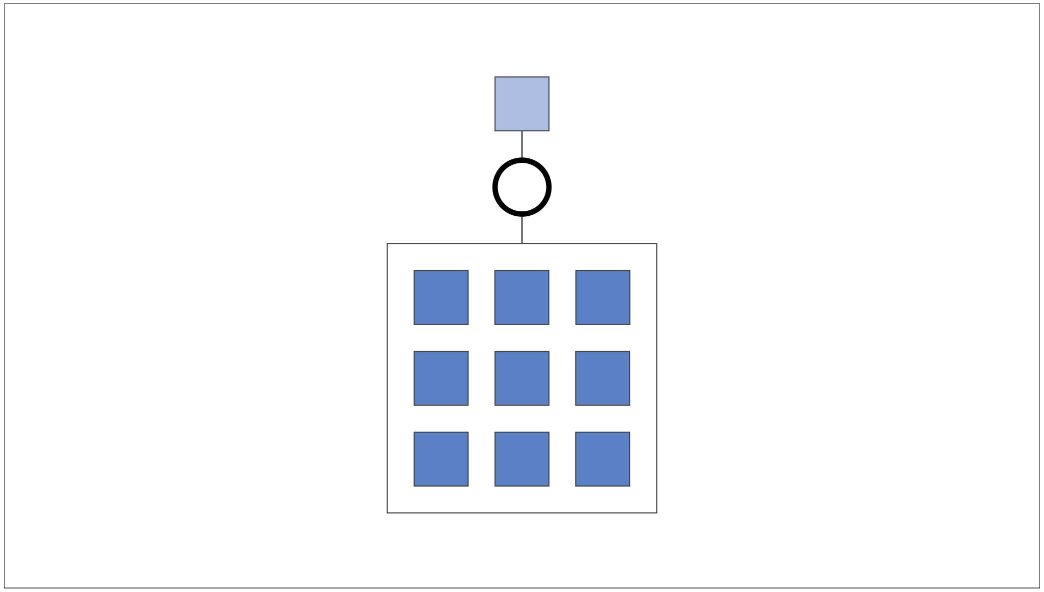
What problem does Ingress for Kubernetes address



Ingress for Kubernetes enables …



the external consumption of a set of Kubernetes HTTP Services hosted on one Cluster …

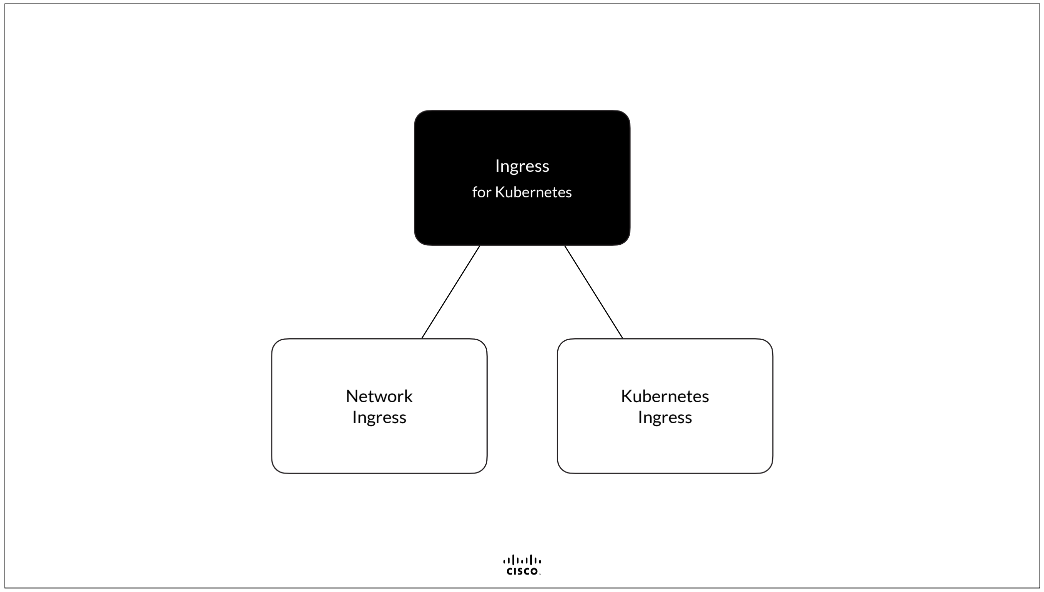


via one HTTP Endpoint

Ingress for Kubernetes

Solution

How does Ingress for Kubernetes address this problem



To enable the external consumption of a set of Kubernetes HTTP Services hosted on one Cluster via one HTTP Endpoint, Ingress for Kubernetes addresses two different concerns, Network Ingress as well as Kubernetes Ingress

- Network Ingress addresses the question of how to **admit** traffic into the cluster
- Kubernetes Ingress addresses the question of how to **route** traffic within cluster

A Kubernetes cluster is typically defined as a set of Kubernetes nodes, a set of physical or virtual machines. However, this presentation is not concerned with nodes, so we will reason about a cluster as the set of pods that ran, run, or will run on the cluster's nodes.

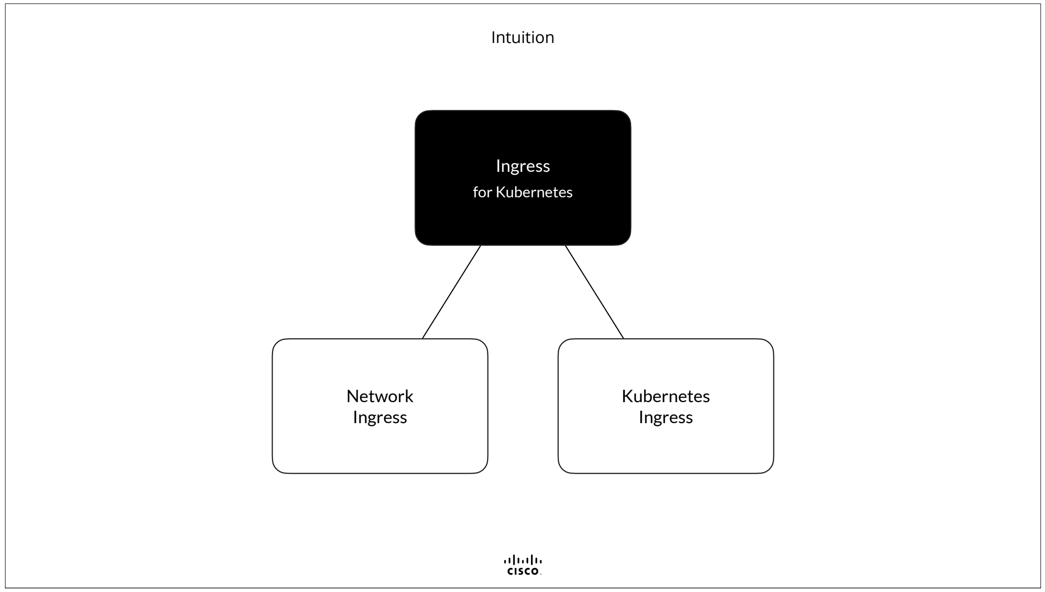
The first topic of this presentation will discuss Network Ingress, the admission of traffic. However, as Kubernetes does not specify how to implement Network Ingress, leaving the implementation up to the operator of a Kubernetes cluster, we will discuss only the what not the how.

The second topic of this presentation will discuss Kubernetes Ingress, the routing of traffic

Ingress for Kubernetes

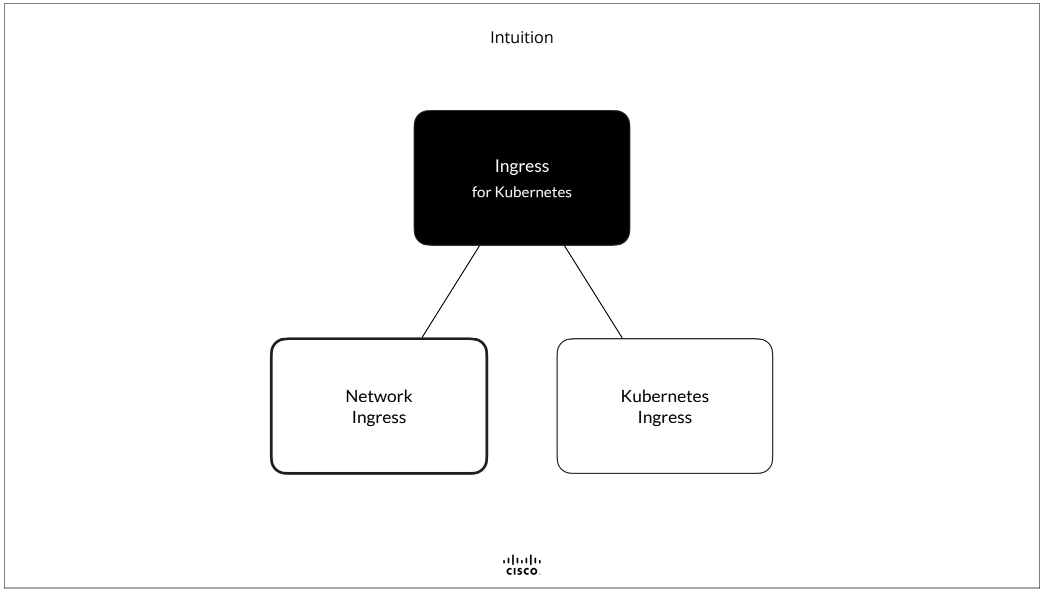
Intuition

Before we develop a **definition** of Ingress for Kubernetes we will spend the next few minutes to develop a **intuition** of Ingress for Kubernetes

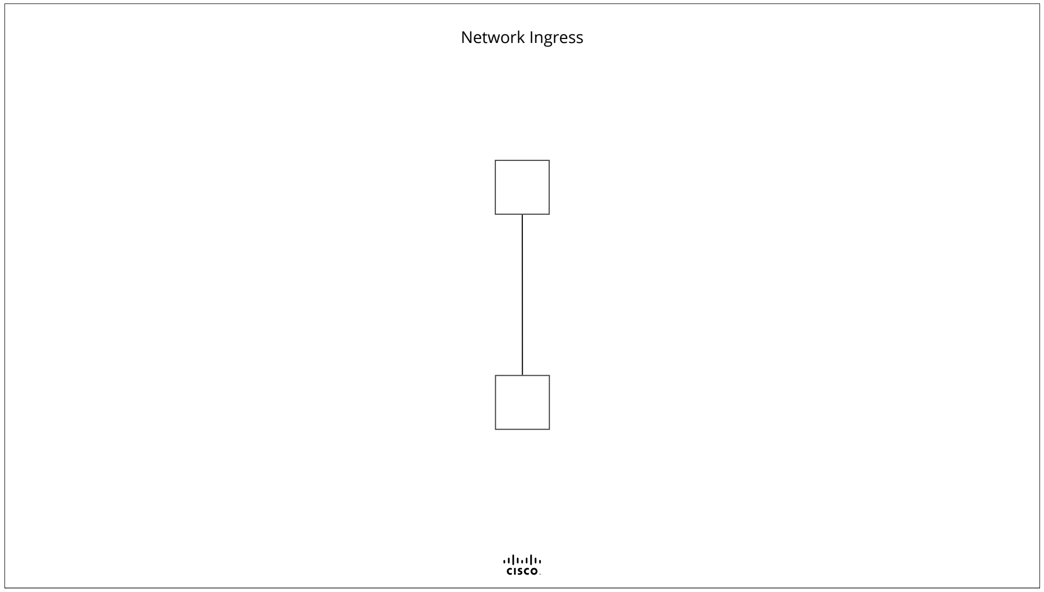


In order to develop an intuition of Ingress for Kubernetes, we will develop an intuition of both Network Ingress and Kubernetes Ingress

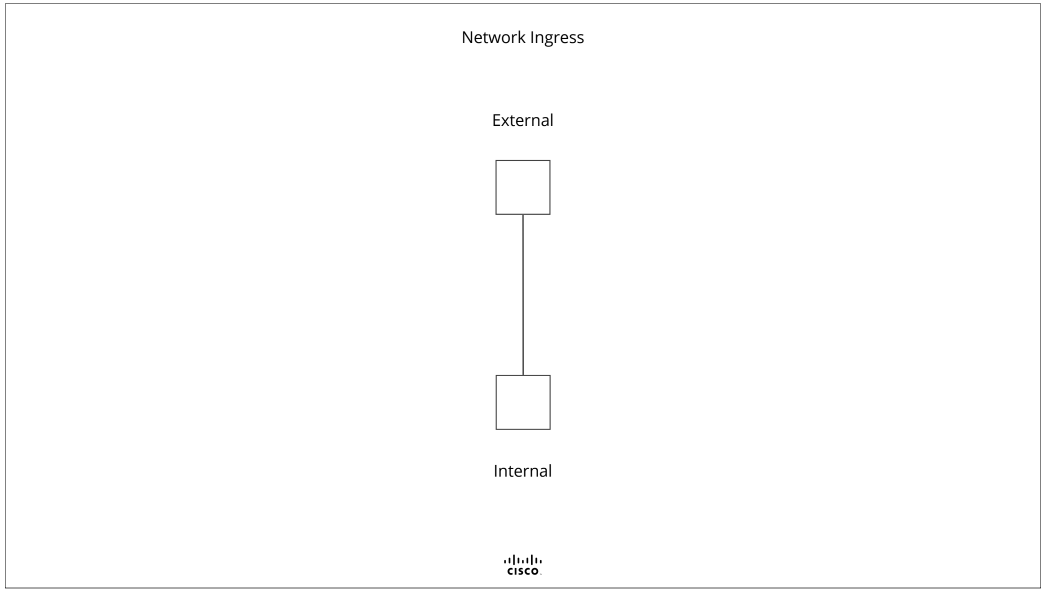
First up …



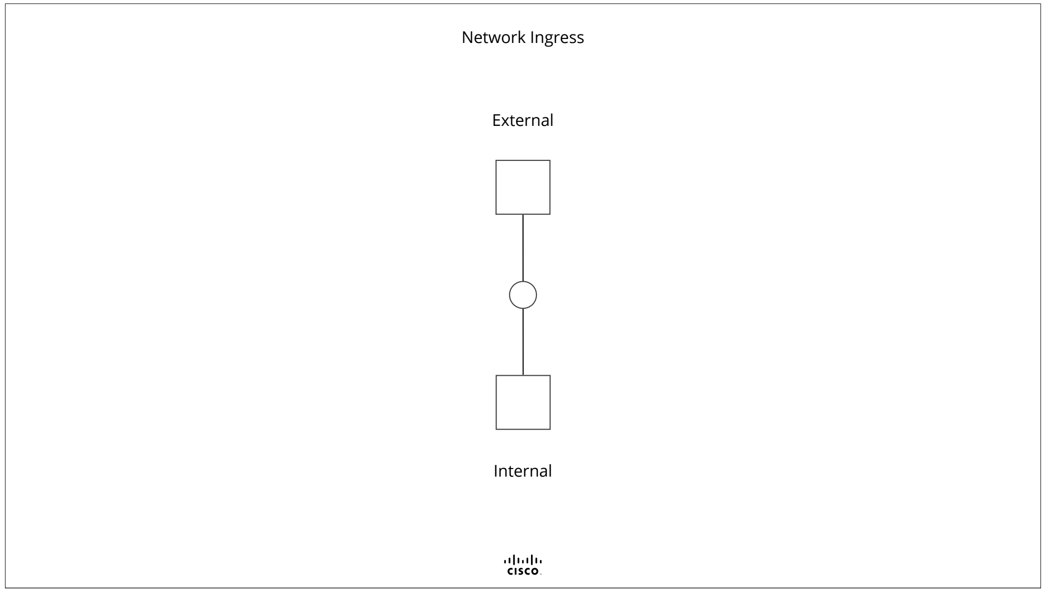
Network Ingress, the admission of traffic



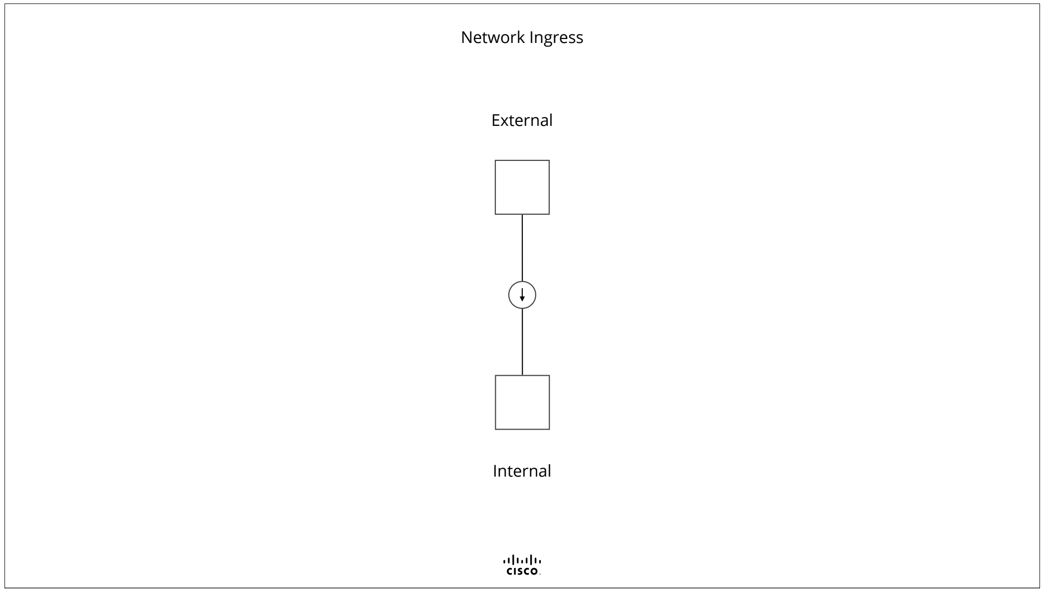
Let there be two communicating endpoints, a service consumer and a service provider.



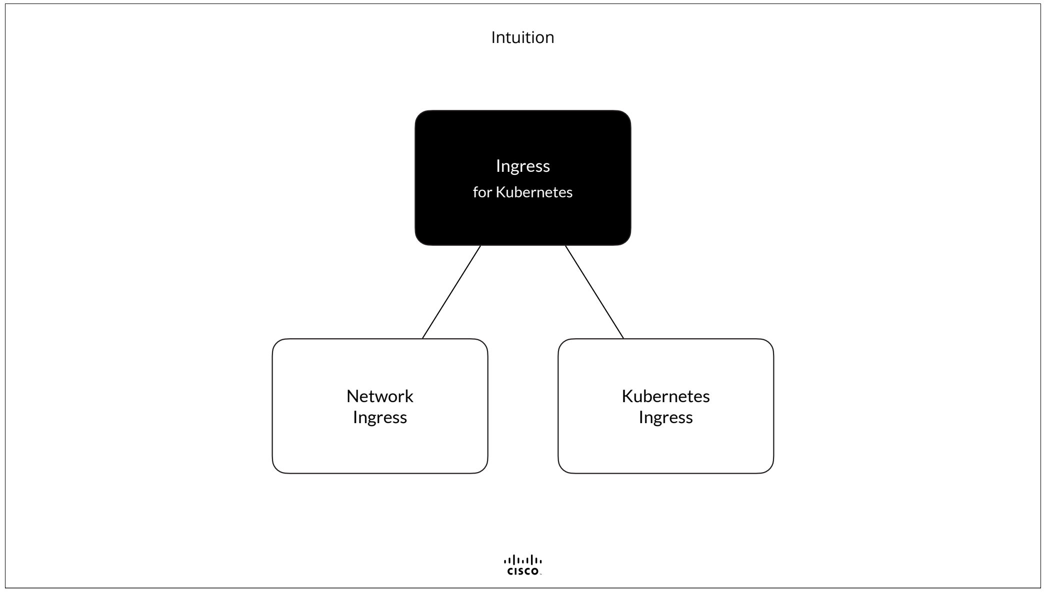
The service consumer is not hosted on the the Kubernetes cluster, it is external. The service provider is hosted on the Kubernetes cluster, it is internal



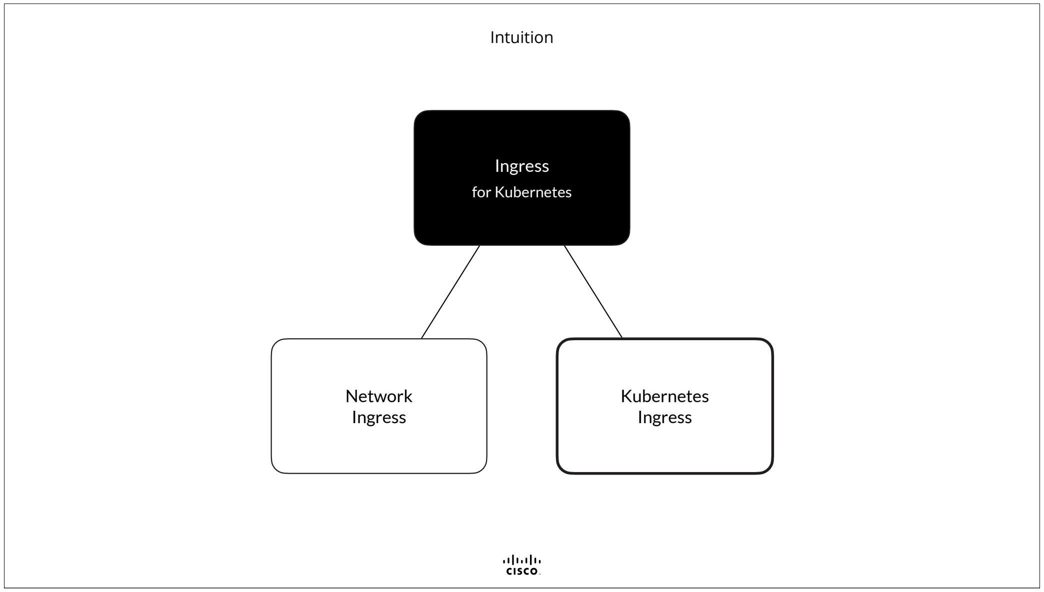
Network Ingress denotes the point or means of admission. Furthermore …



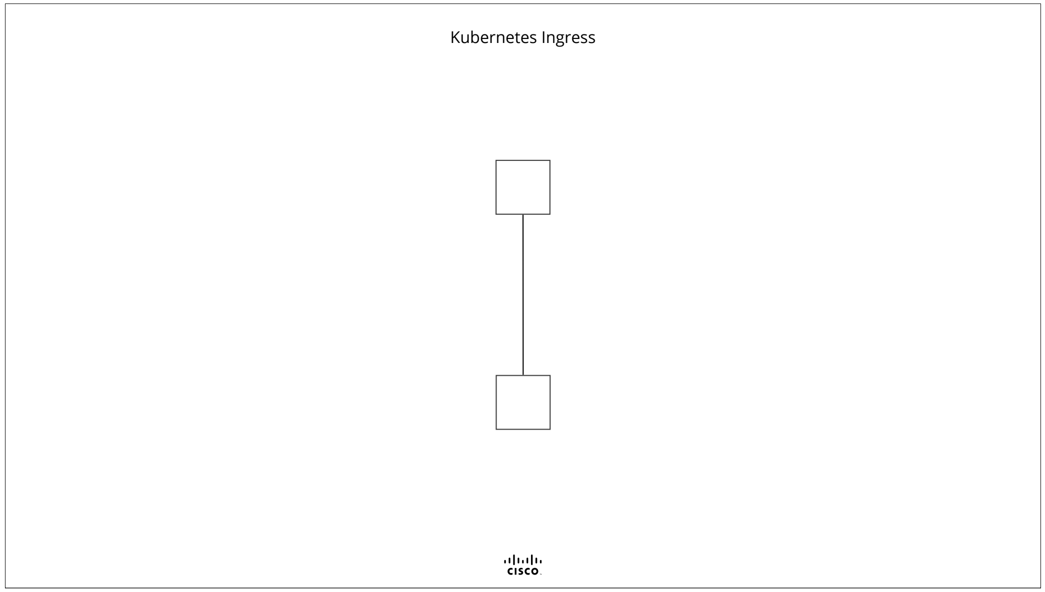
Network Ingress implies directionality, crossing from external to internal.


Intuition
Ingress
for Kubernetes
Network
Ingress
Kubernetes
Ingress

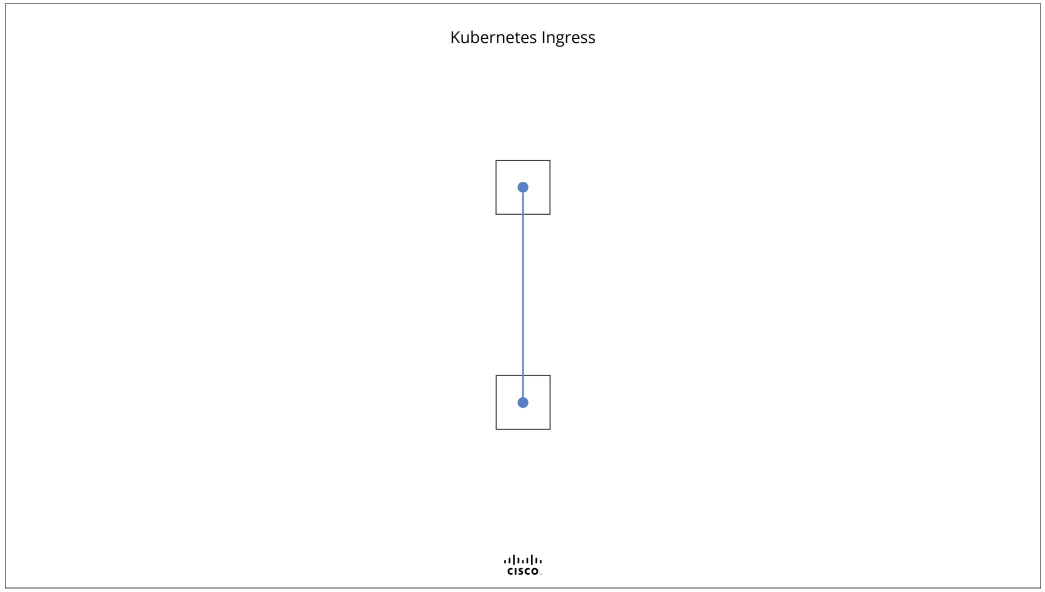
Next up …



Kubernetes Ingress, the routing of traffic

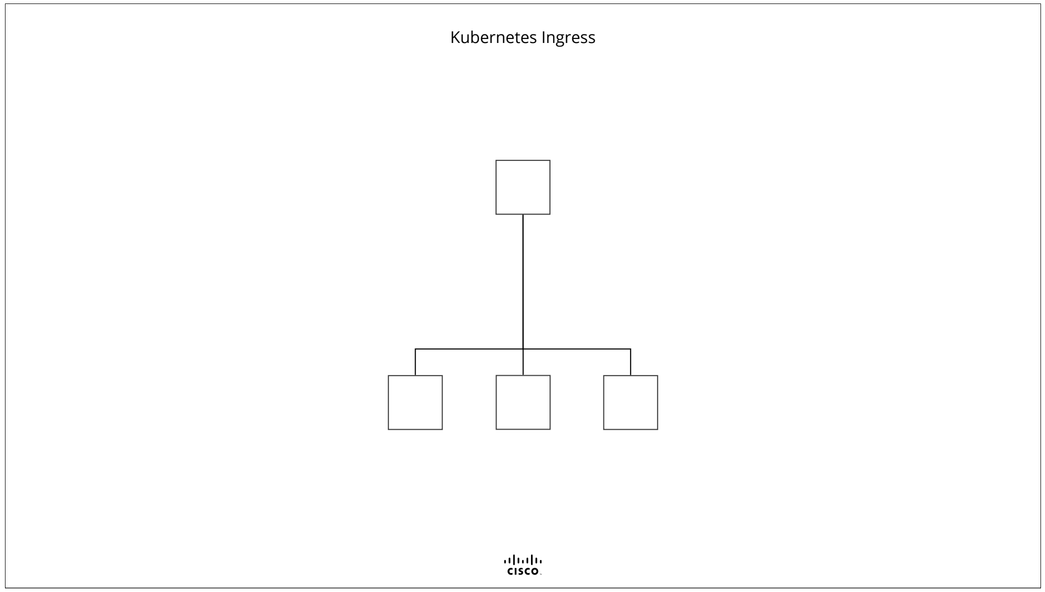


Previously, there were two communicating endpoints, a service consumer and a service provider. The service consumer has to learn the address of the service provider …



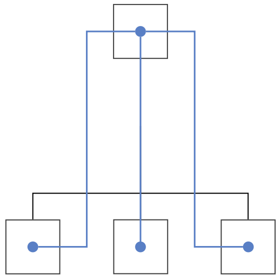
to actually consume the provided service.

However, a persistent trend complicates this picture. One Monolithic service provider …

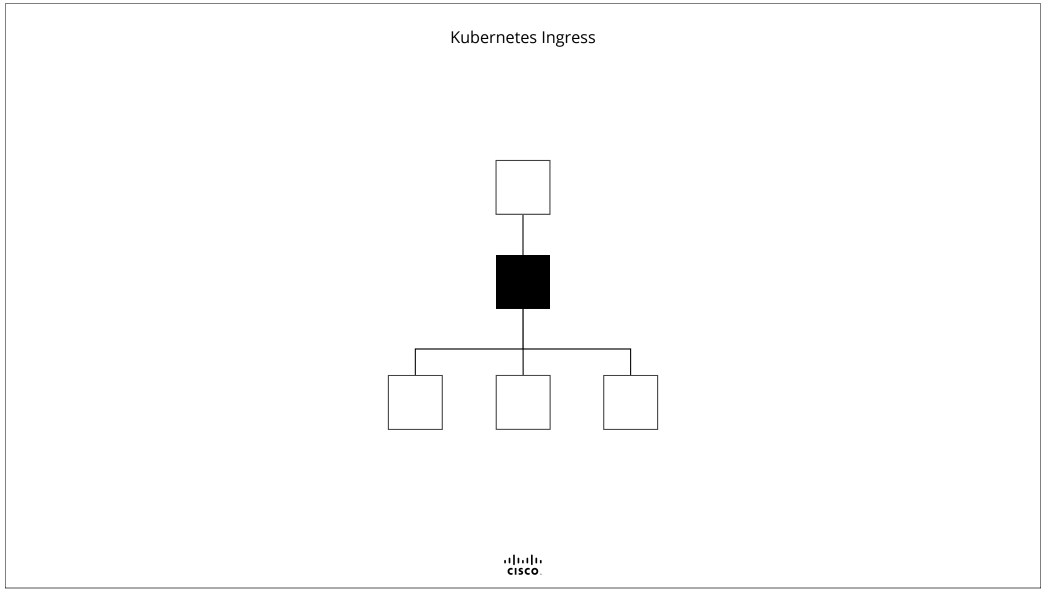


is broken up into many service providers, micro services. Now, the service consumer has to learn the address of each service provider …

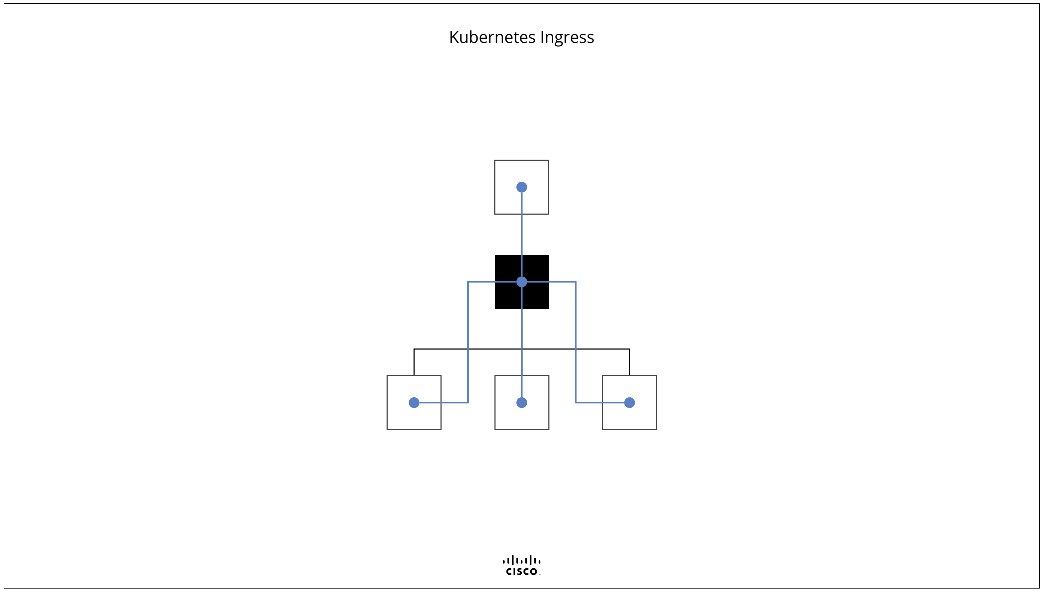
Kubernetes Ingress



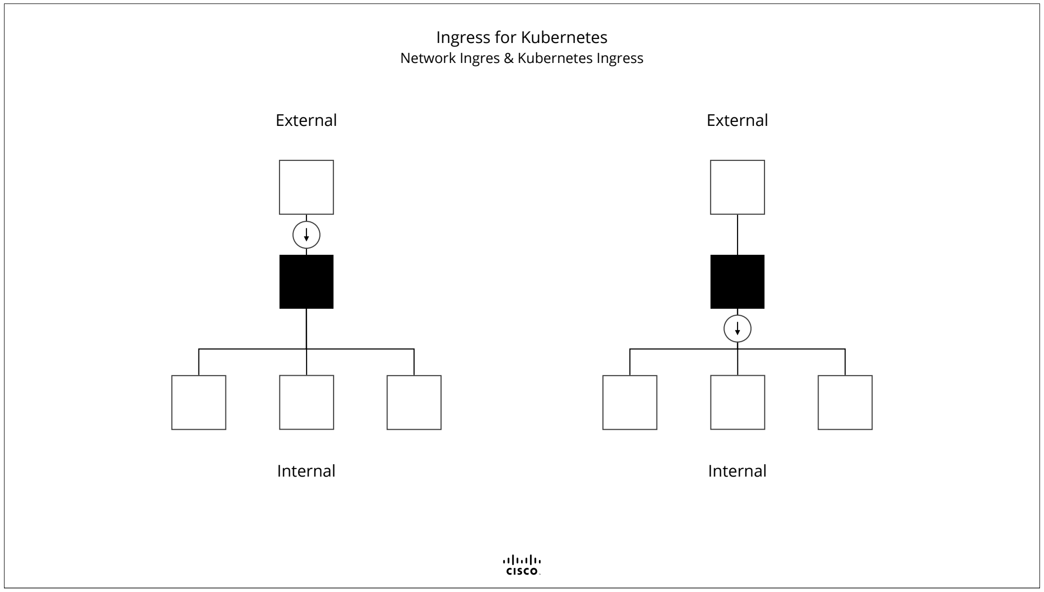
to consume the services



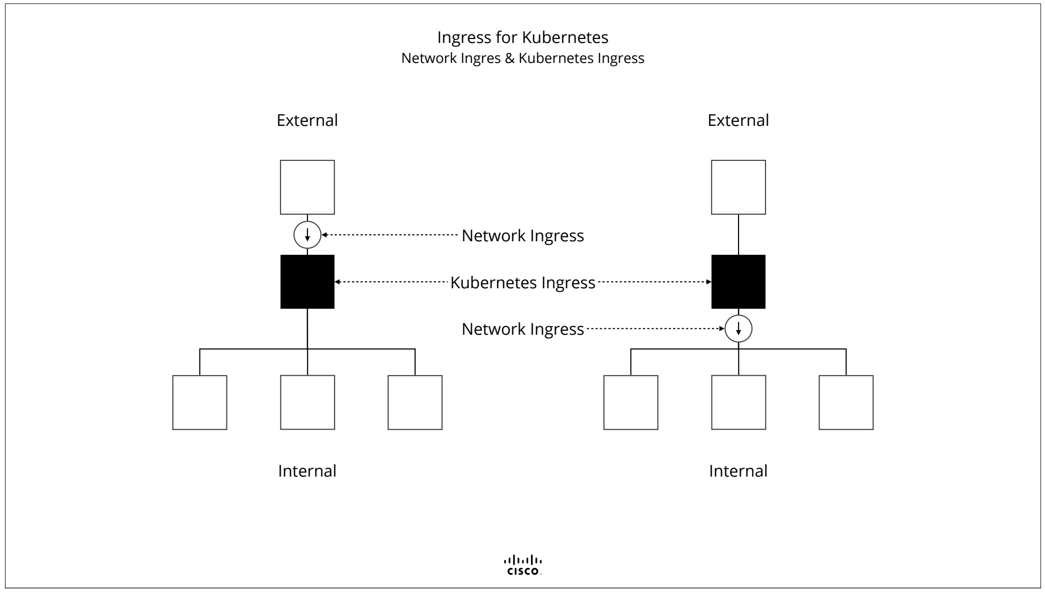
Kubernetes Ingress is a proxy, an API gateway, that exposes multiple service providers as a single endpoint therefore greatly simplifying …



consuming the services

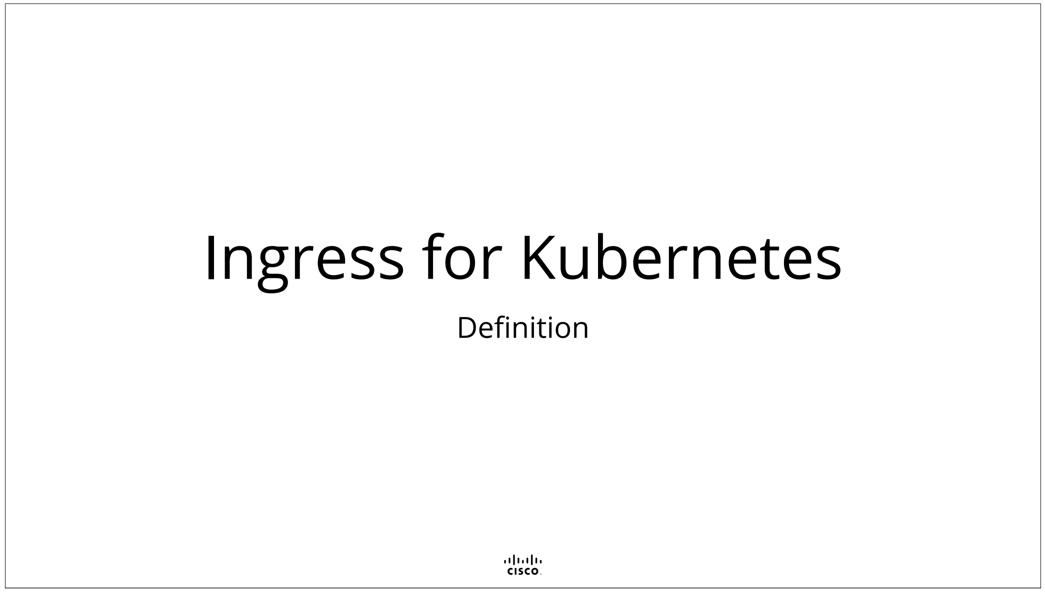


Putting both together …

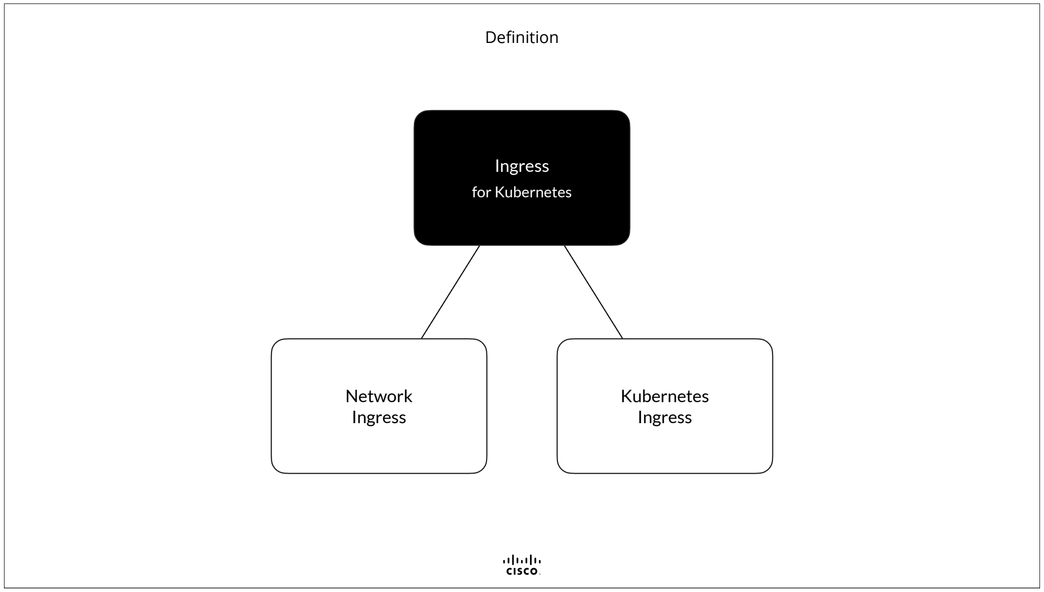


Ingress for Kubernetes is the composition of Network Ingress and Kubernetes Ingress, where Network Ingress is the admission of traffic into the Kubernetes Cluster and Kubernetes Ingress is the routing of traffic within the Kubernetes Cluster.

In effect, Kubernetes Ingress is an API Gateway

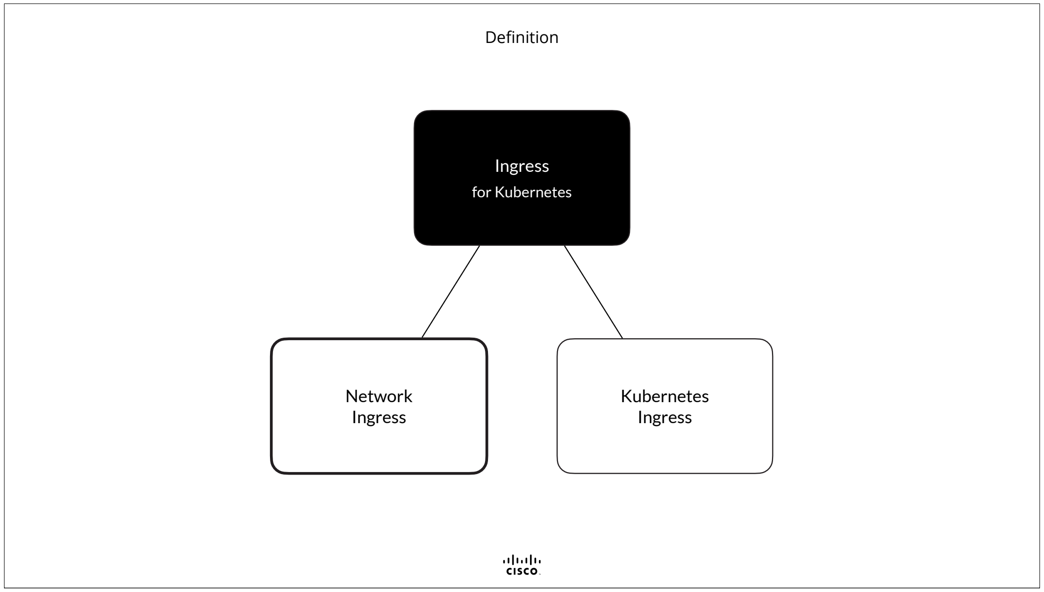


With an intuition of Ingress for Kubernetes, we will spend the rest of the presentation to develop a set of related **definitions** of Ingress for Kubernetes

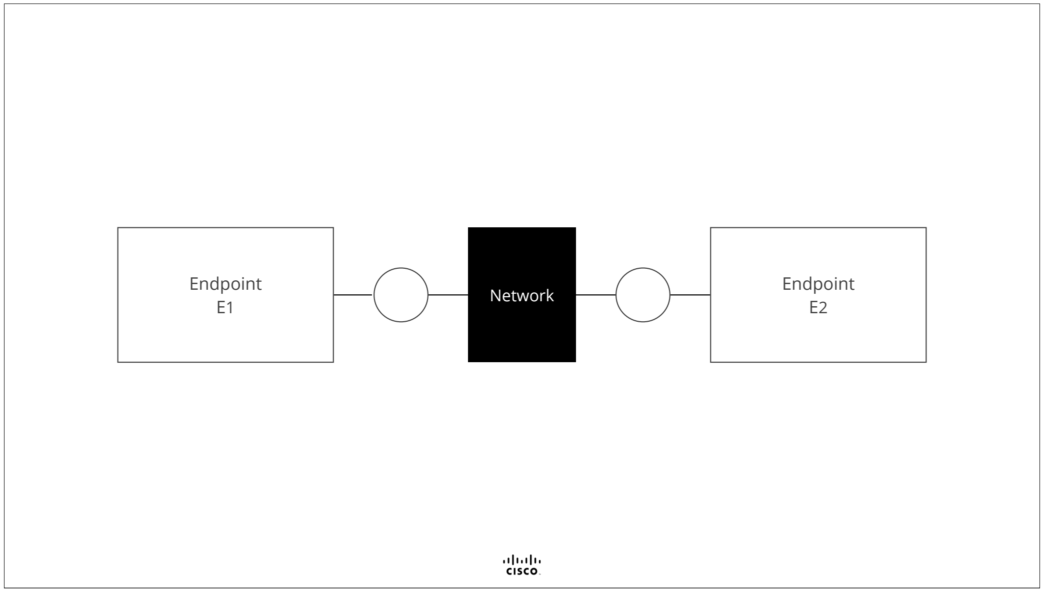


In order to develop definitions for Ingress for Kubernetes, we will once again develop definitions for both Network Ingress and Kubernetes Ingress

First up …



Network Ingress, the admission of traffic



In software engineering, a distributed system is an unbounded set of components, from heron out called Endpoints. Endpoints communicate by exchanging messages via a **network**.

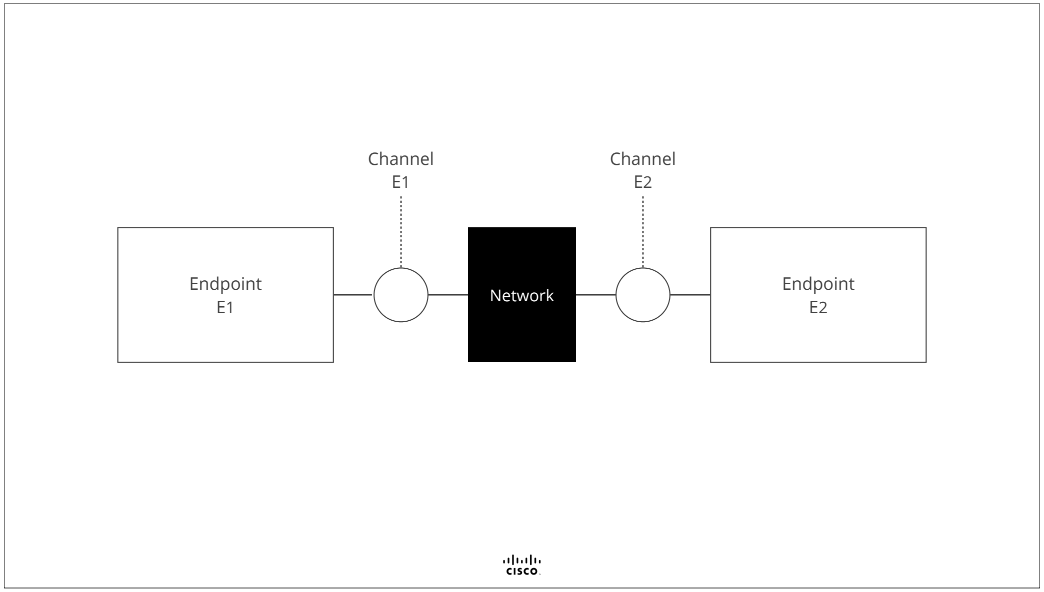
The behavior of a distributed system is attributed to

- the behavior of its Endpoints
- and the communication between them.

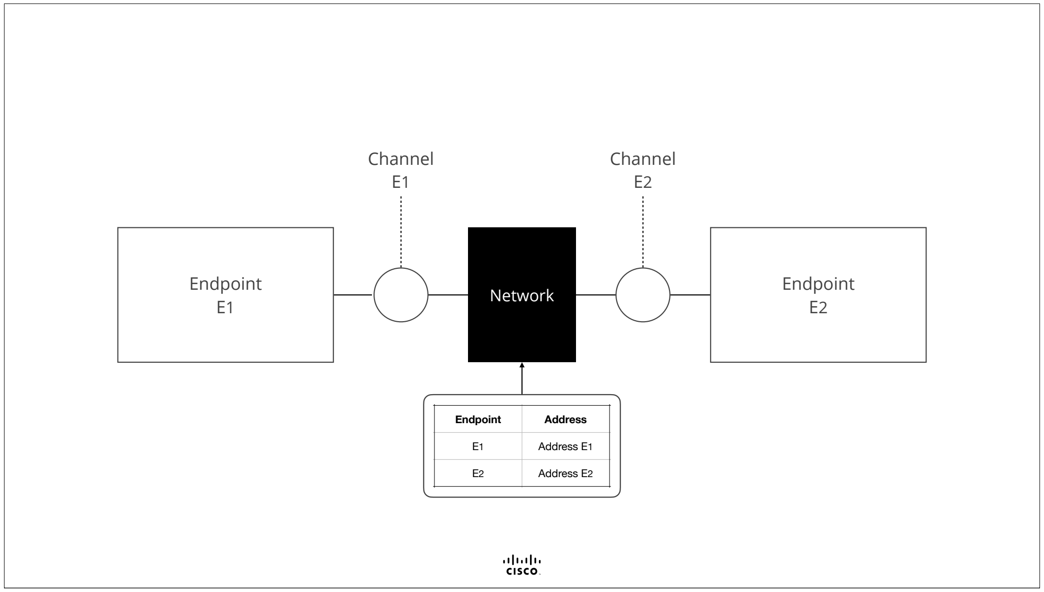
The complexity of a distributed system is attributed to

- the autonomy of its Endpoints
- and the intricacy of the communication between them.

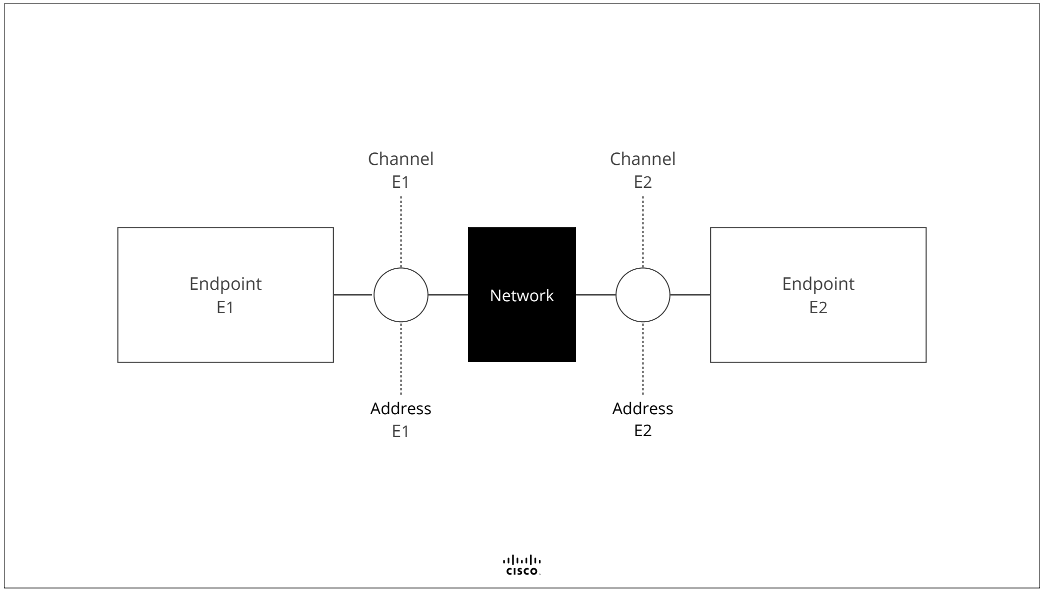
Without loss of generality let's focus this discussion on two Endpoints, E1 and E2



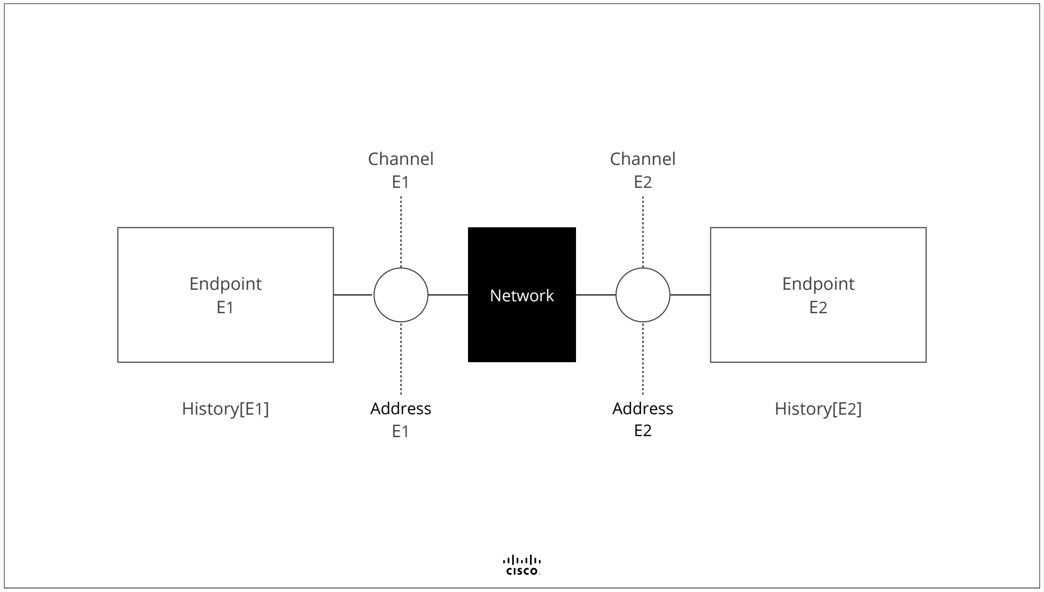
An Endpoint is connected to the network via a channel



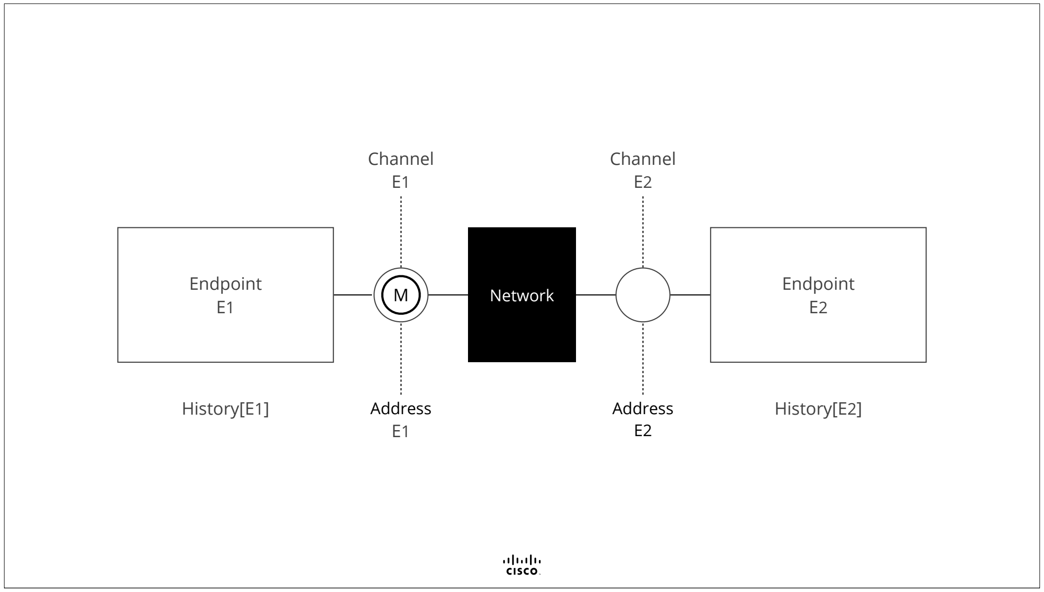
The network maintains an association between Endpoints and Addresses



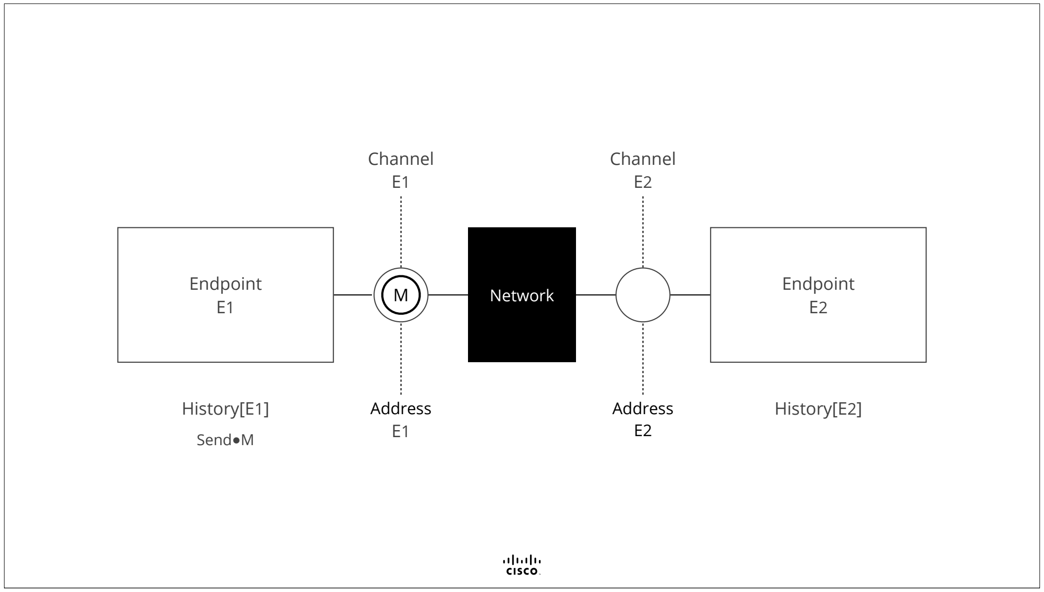
From heron out we will graphically represent this association as if the Address is a property of the Channel



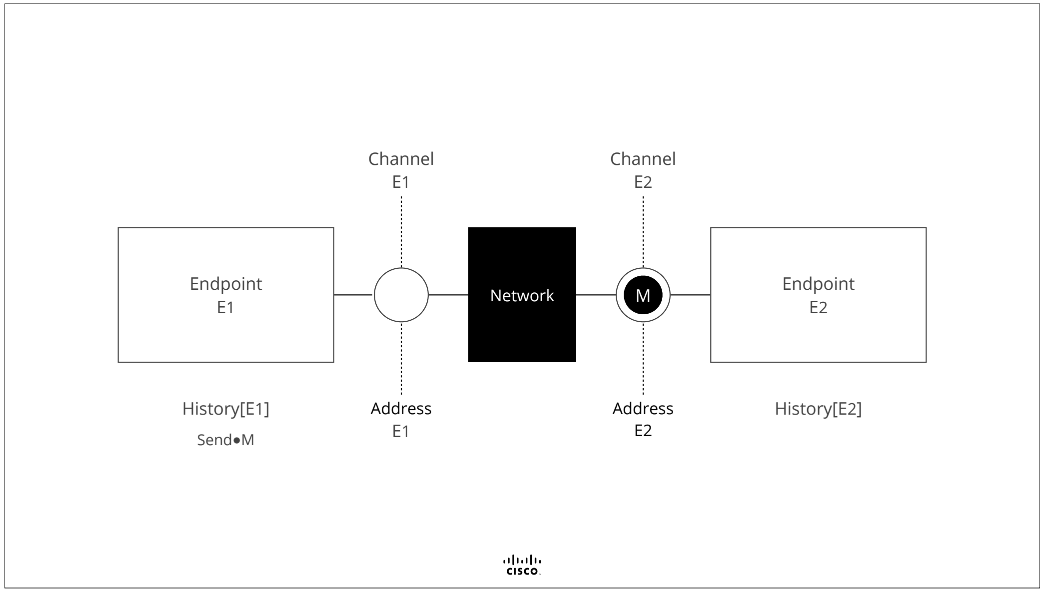
We keep track of the sequence of send events and receive events in an Endpoint’s History



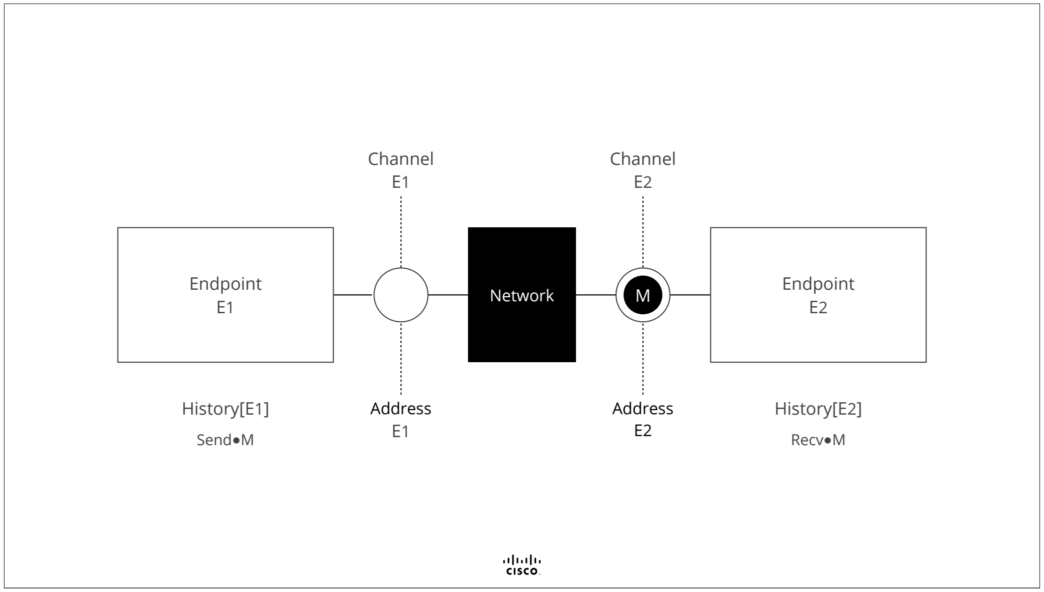
If an Endpoint wants to send a message it will place that message in its channel



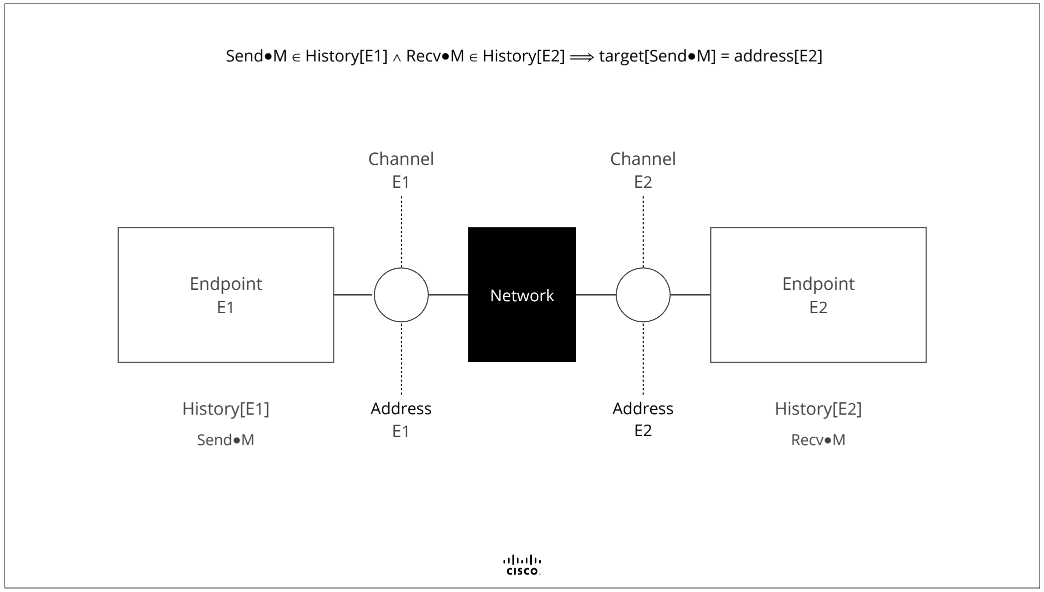
An Endpoint placing a message in its channel is represented by a Sent Event



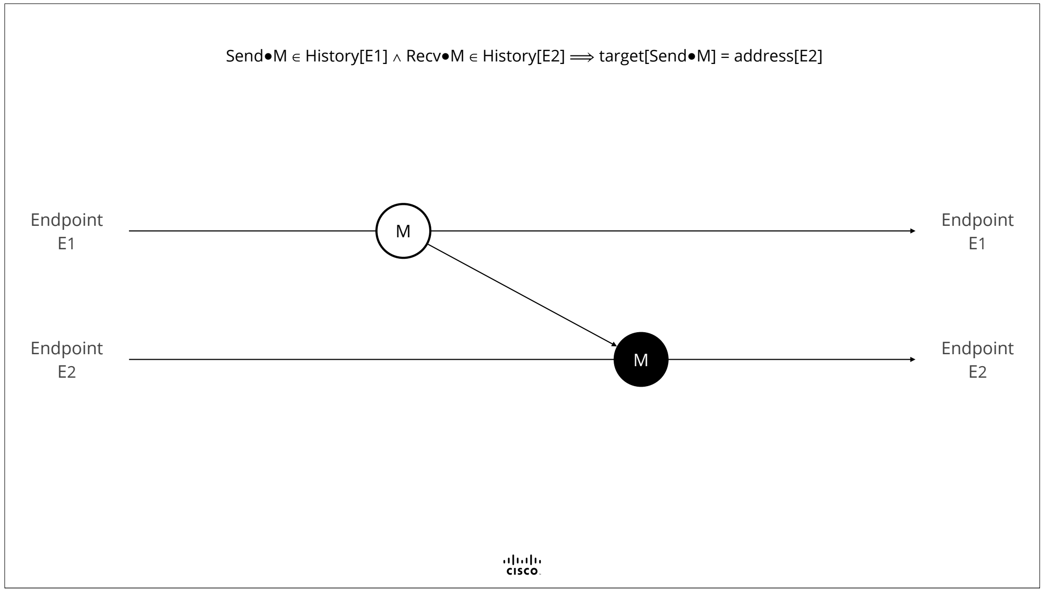
The network picks up the message from the sending Endpoint's channel and determines the receiving Endpoint's channel and places the message in that channel



The Network placing a message in and endpoint’s channel is represented by a Recv Event

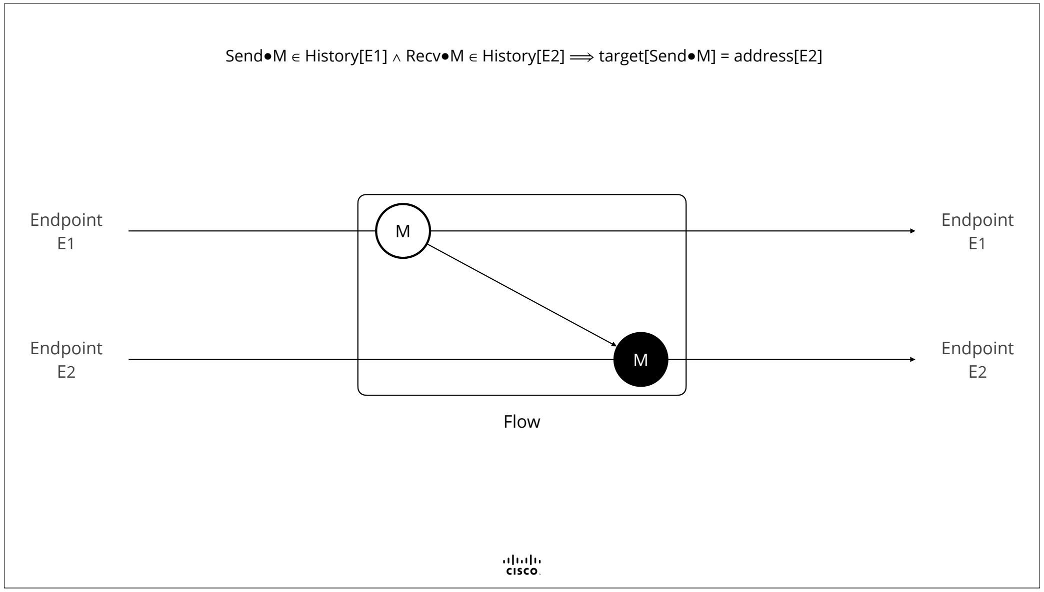


In this network model, Send Events are tagged with a target address. the network places the message in the channel of the endpoint who's address matches the message's target address

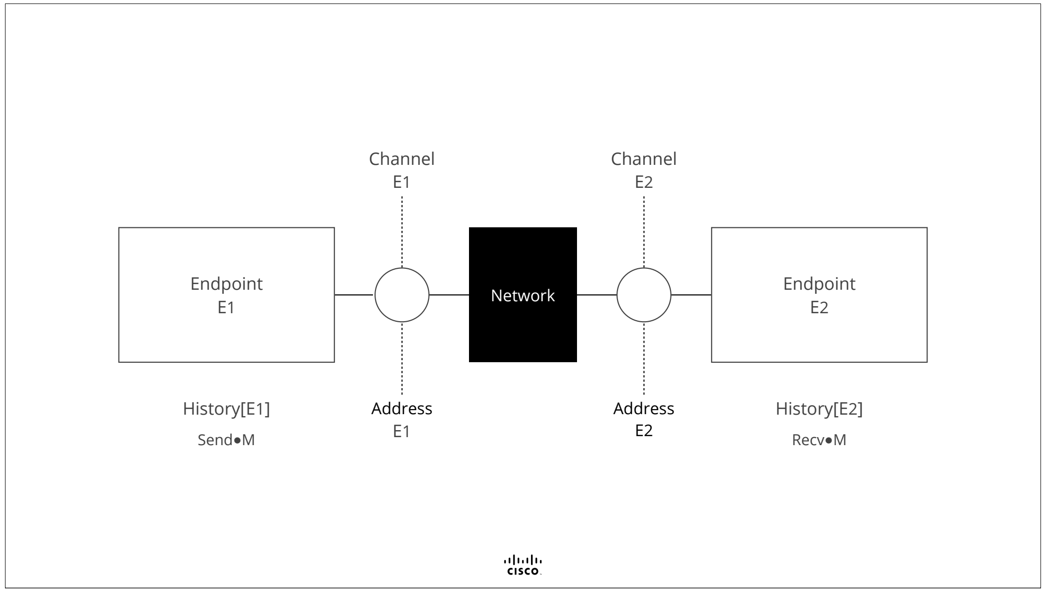


This can also be represented graphically as a Time Space Diagram.

Each timeline represents an Endpoint's history, empty circles represent Send Events, filled circles represent Recv Events.



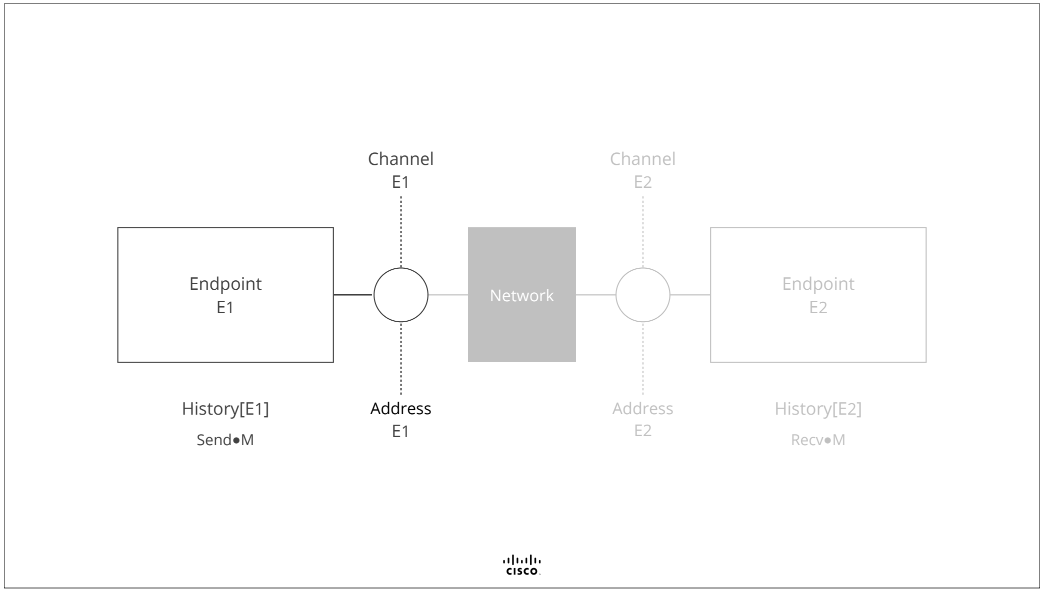
A pair or tuple of corresponding Send and Receive Events is called a Flow



So far, we have applied a Global Point Of View

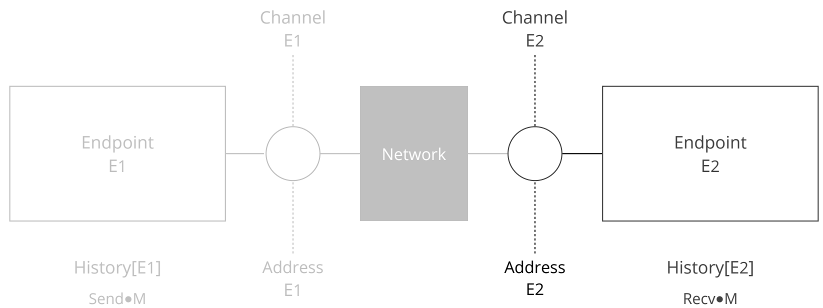
In this model we are able to take the viewpoint of the all knowing observer, we can observe both the inbox and outbox of E1 and E2 at the same time.

Conversely, E1 or E2 cannot



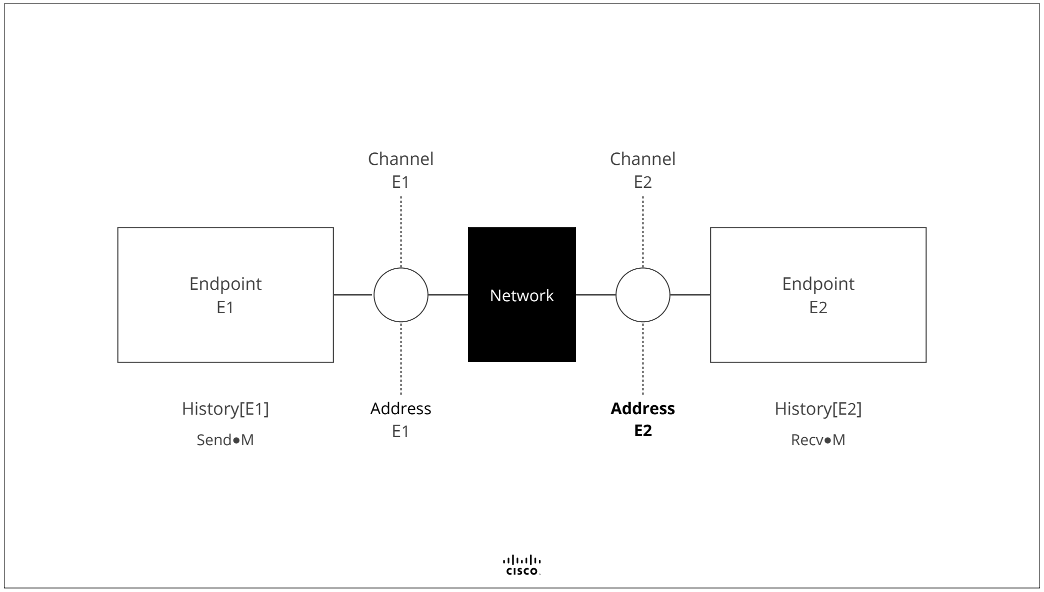
But from a local point of view, we simply cannot

E1 can only observe it’s own channel, and in our model its own address, it simply cannot reach beyond

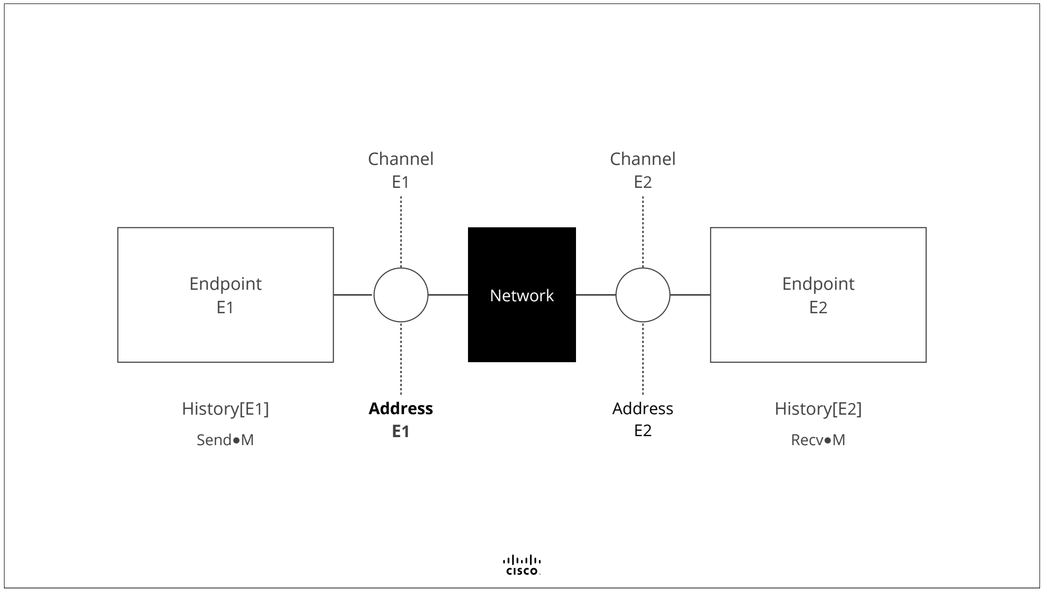


The same is true for E2

E2 can only observe its own channel and its own address

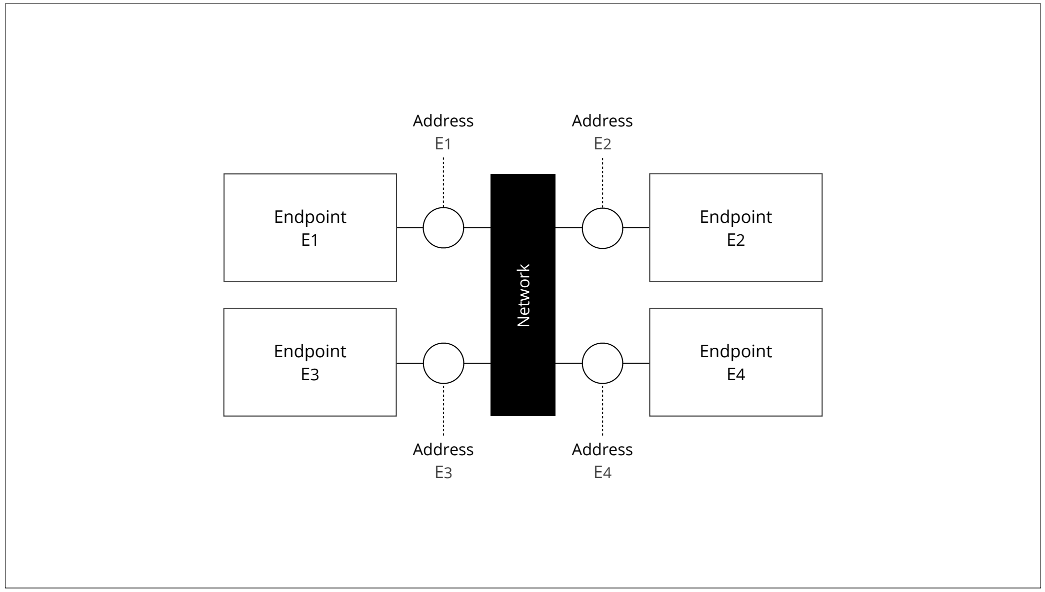


So in order for E1 to send a message to E2, E1 first has to learn the address of E2

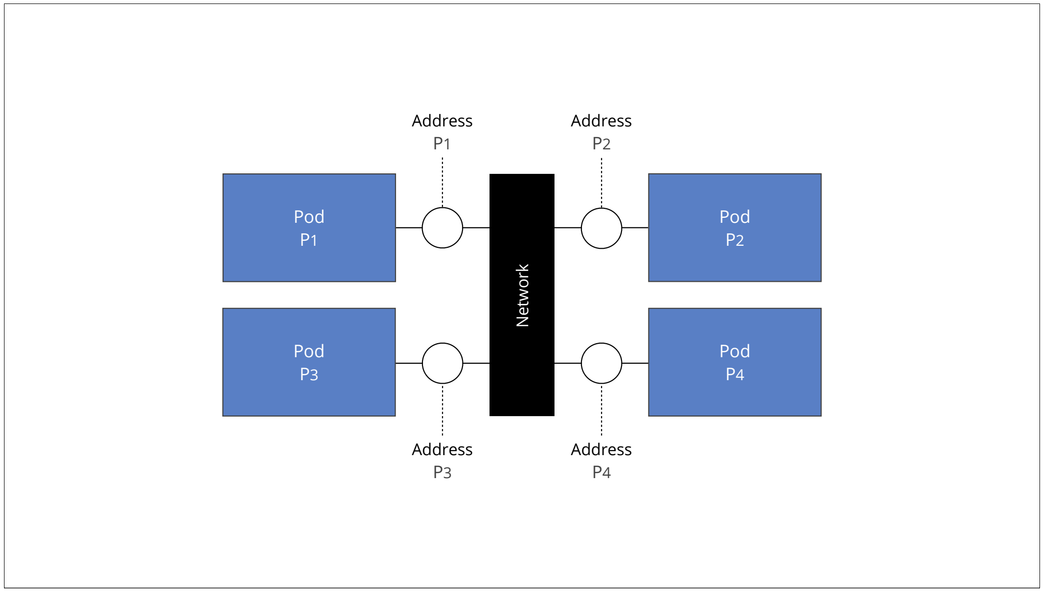


The same is true for E2

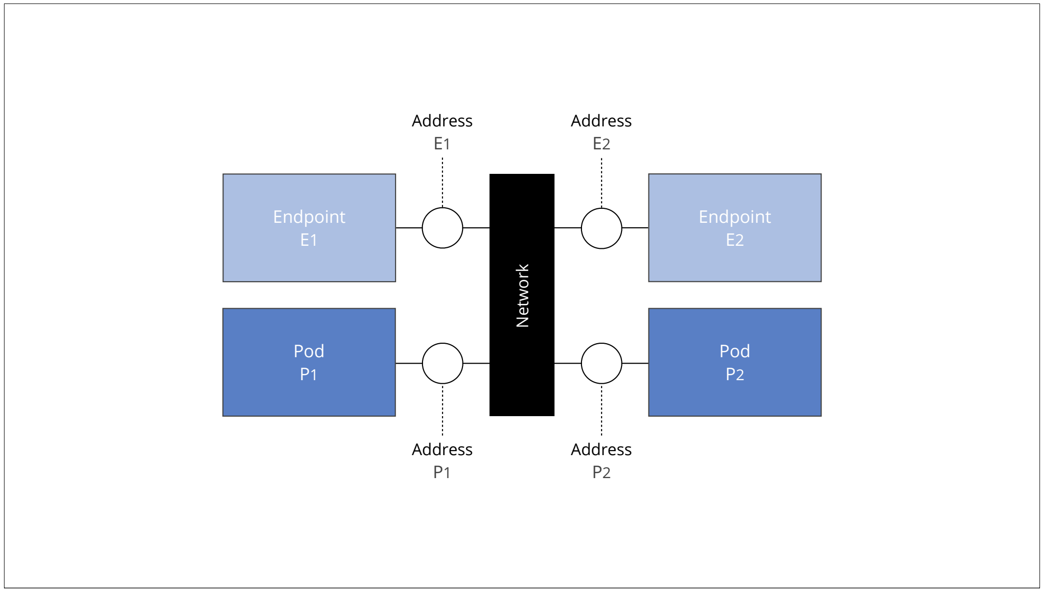
In order for E2 to send a message to E1, E2 first has to learn the address of E1, a process called Endpoint Discovery



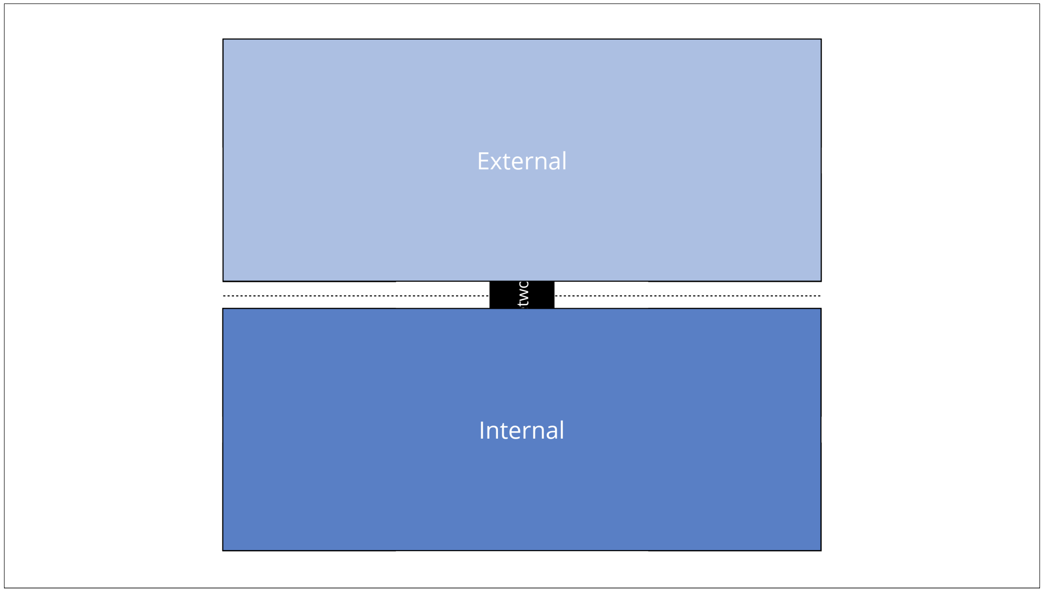
Moving towards the Kubernetes Network Model, in Kubernetes …



Network addressable endpoints are Pods. The Kubernetes Network Model specifies, that any Pod can communicate with all Pods without Network Address Translation



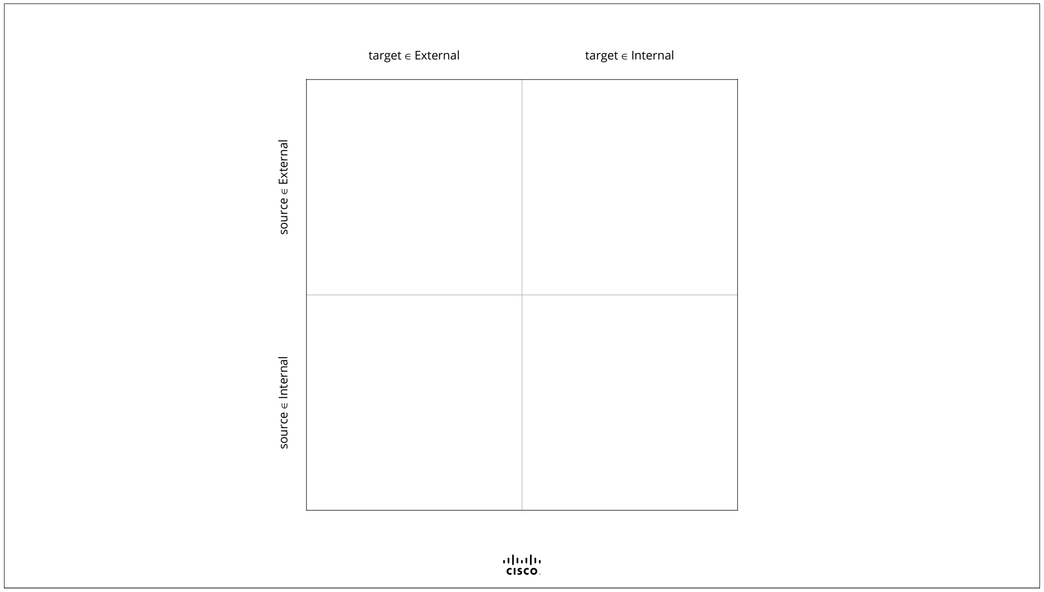
The Kubernetes Network Model does not specify weather external endpoints can or cannot communicate with Pods. As a consequence, depending on your closer, Network Ingress may be trivial or complex to implement



As we discussed earlier, we separate the set of endpoints into external endpoints and internal endpoints who communicate across that line of separation.

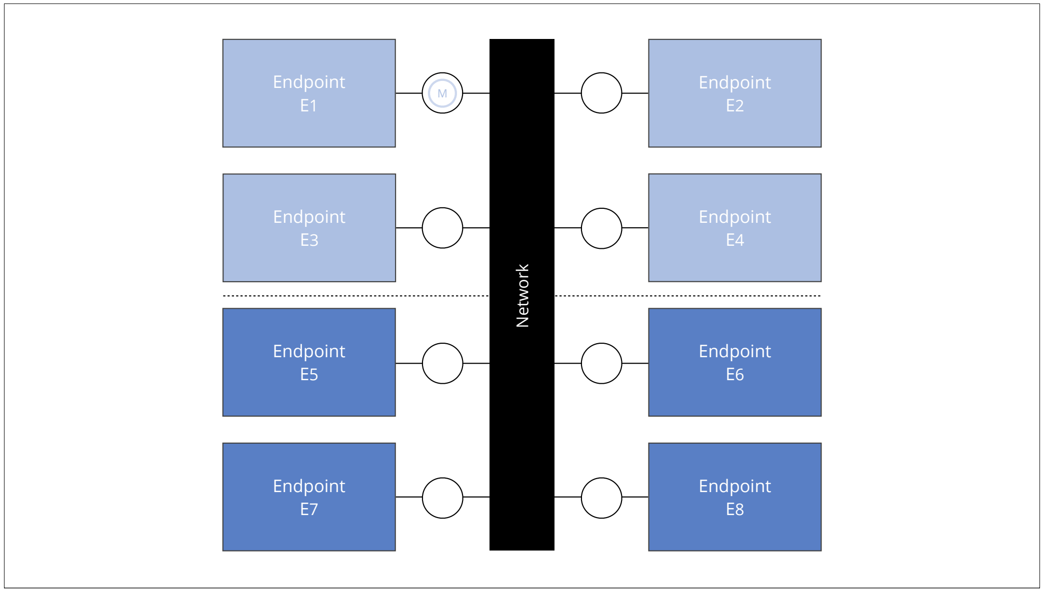
Click

Here, we consider Endpoints 1 through 4 as being external endpoints and 5 through 8 as being internal endpoints

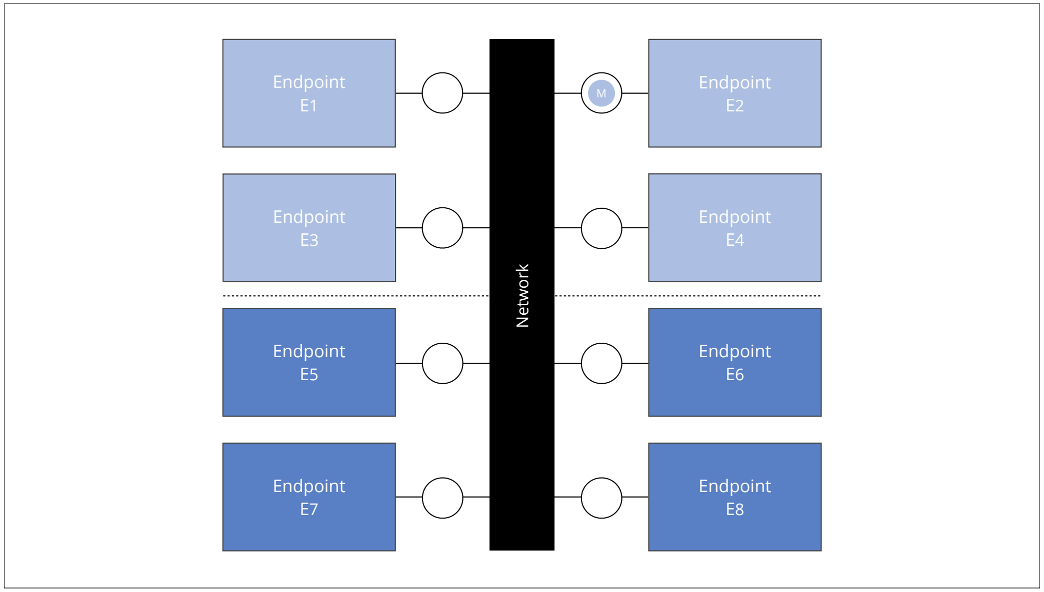


Given the separation of endpoints into external and internal endpoints, we can classify the communication between endpoints according to the membership of the source and target of the communication.

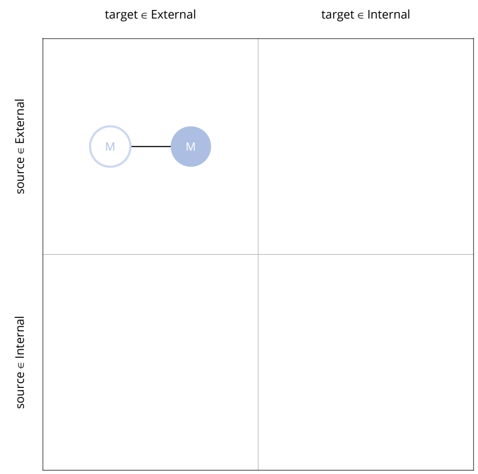
There are four possible combinations



In the first combination, source is a member of the set of external endpoints …

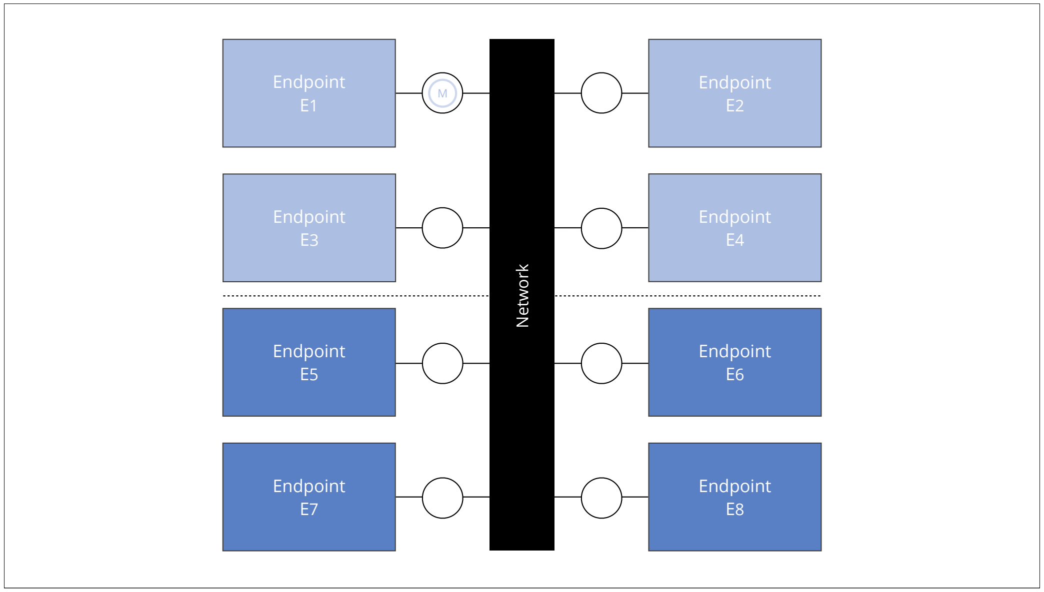


And target is a member of the set of external endpoints

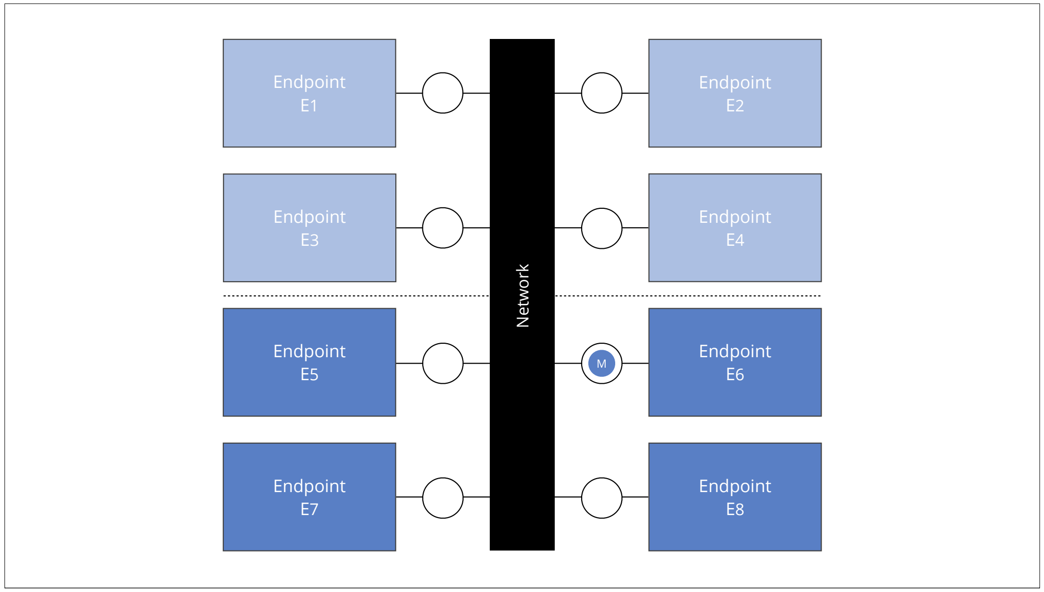


This particular type of flow …

Does not have a name



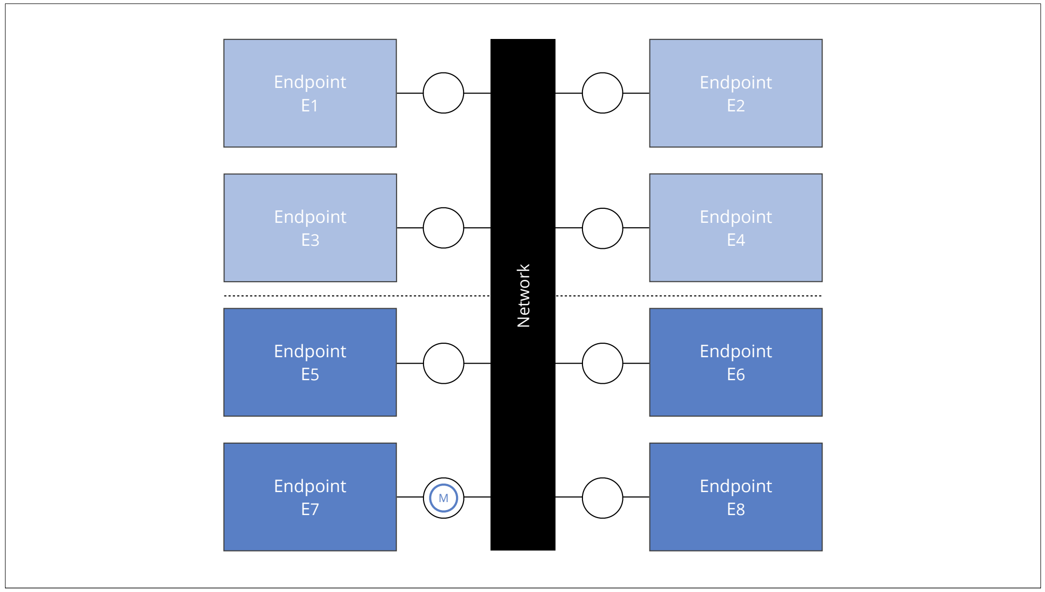
In the second combination, source is a member of the set of external endpoints …



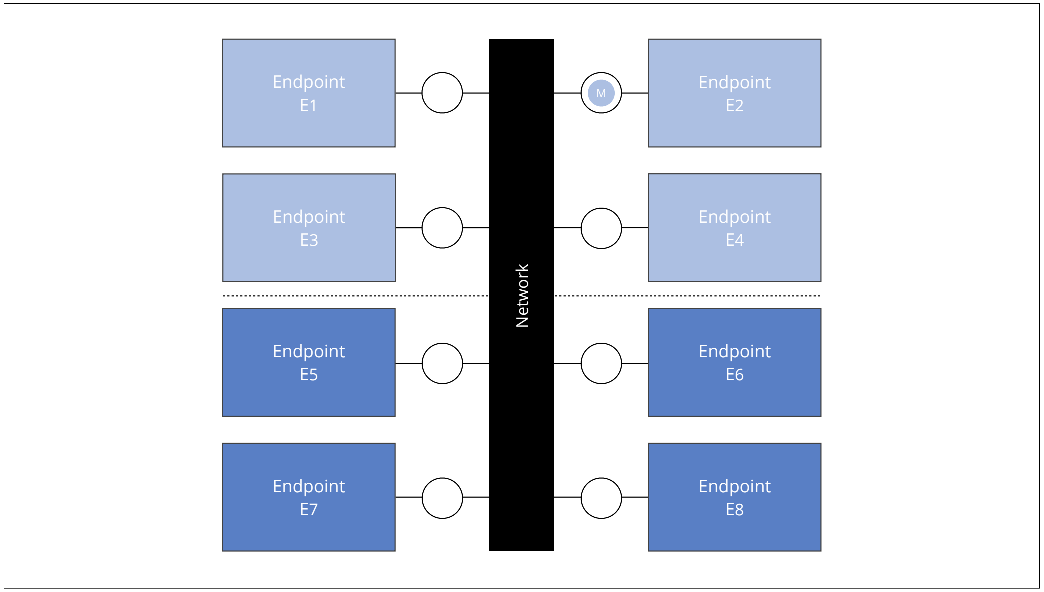
And target is a member of the set of internal endpoints

This particular type of flow …

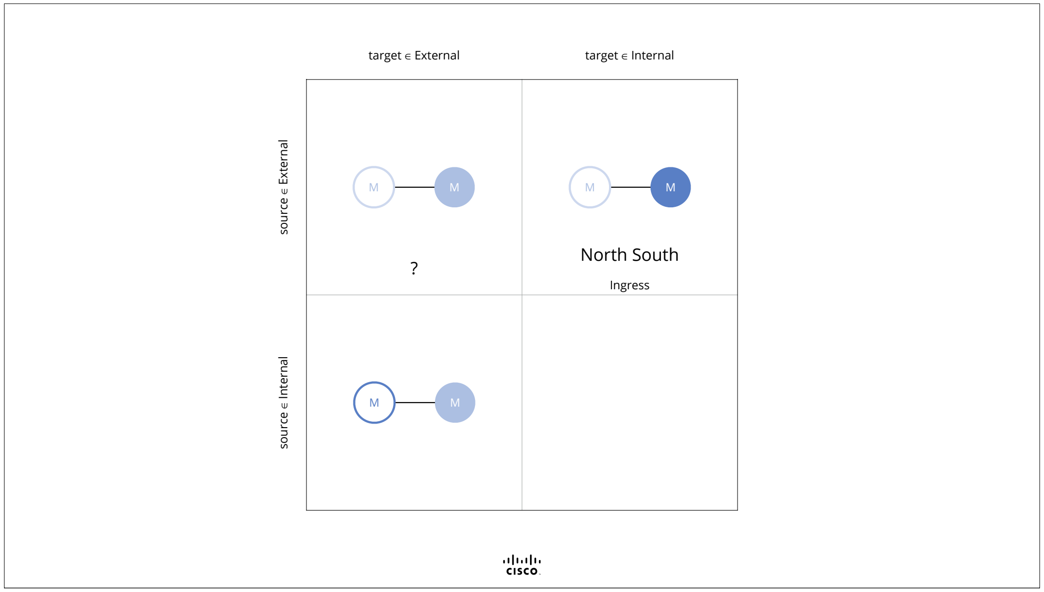
Is called North South Traffic. In addition, given the directionality, this combination constitutes Network Ingress.



In the third combination, source is a member of the set of internal endpoints …

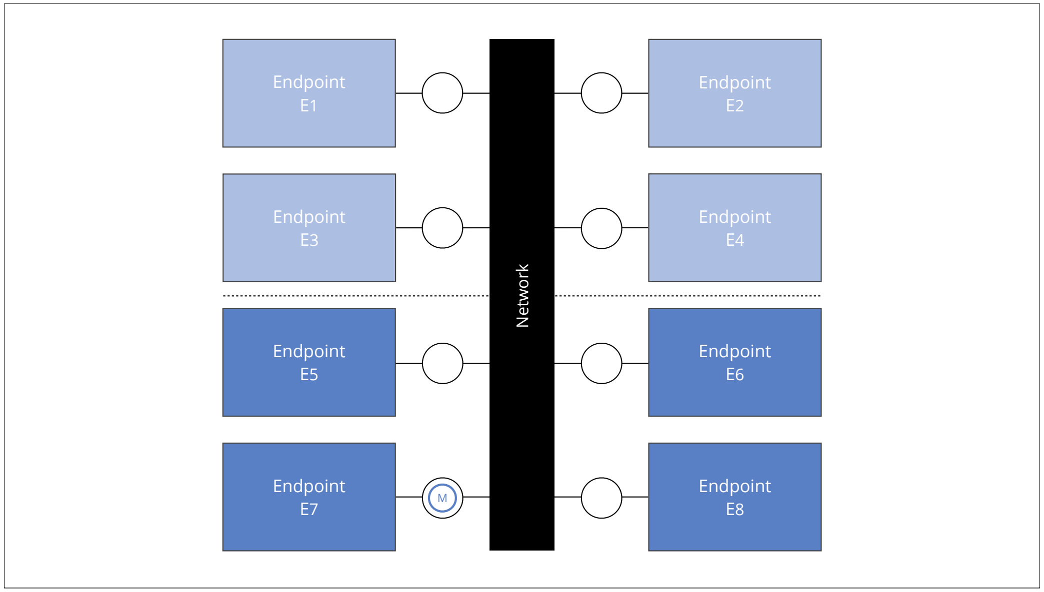


And target is a member of the set of external endpoints

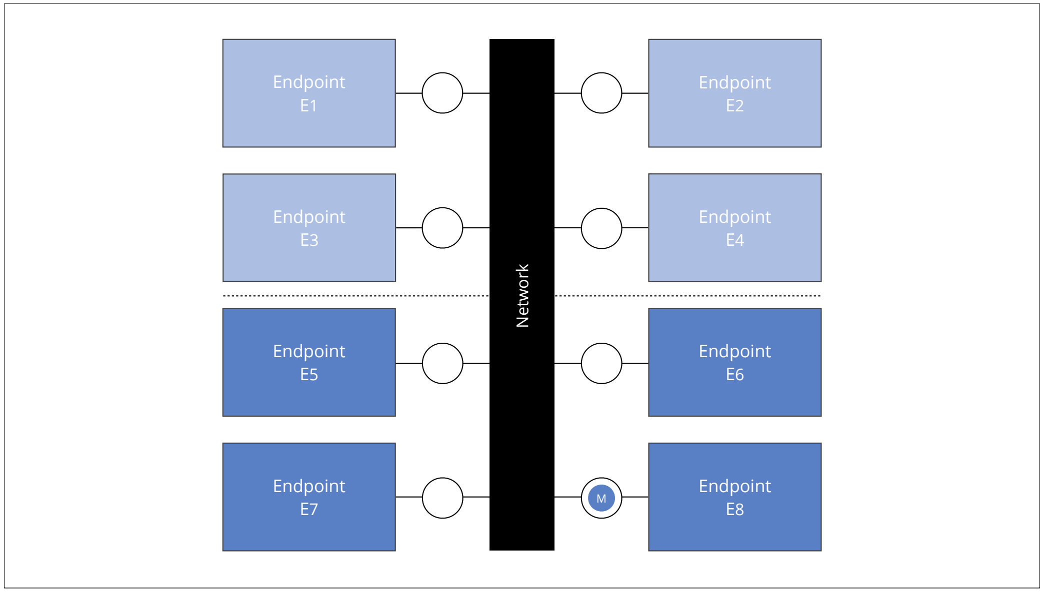


This particular type of flow …

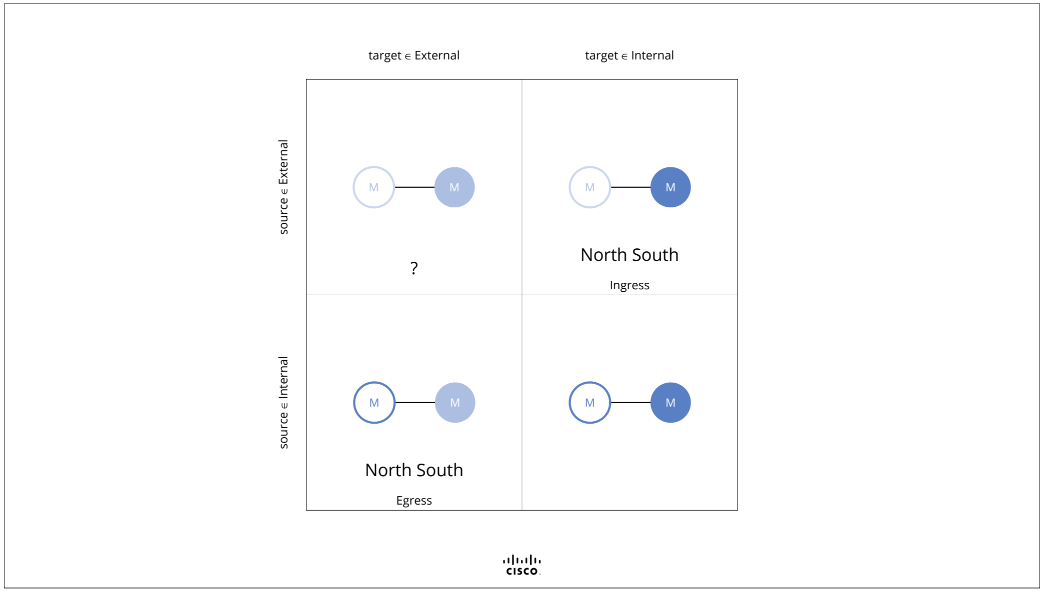
Is again called North South Traffic. In addition, given the directionality, this combination constitutes Network Egress.



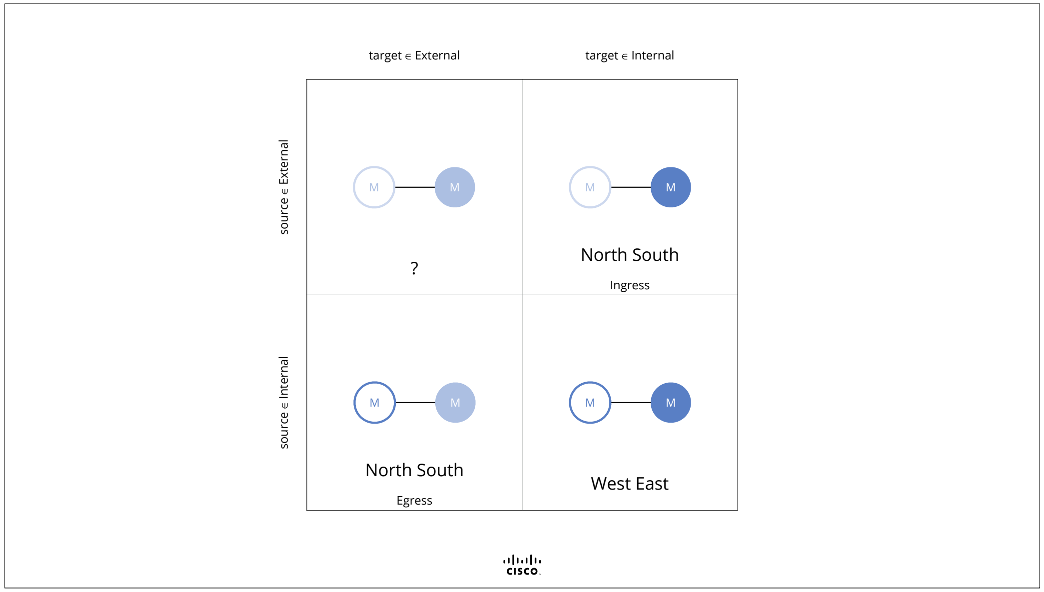
In the fourth and last combination, source is a member of the set of internal endpoints …



And target is a member of the set of internal endpoints



This particular type of flow …



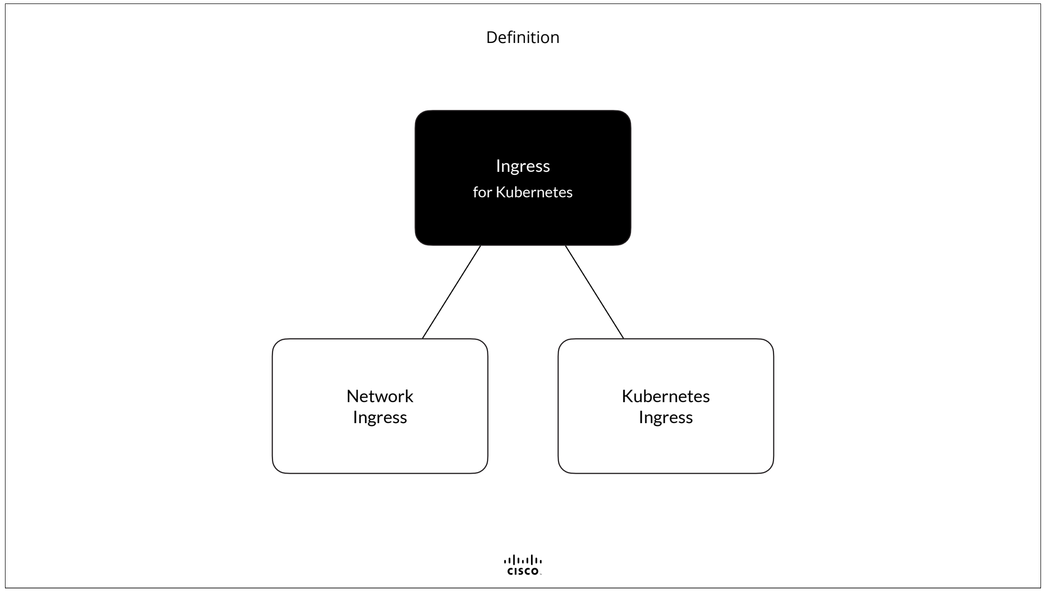
Is called West East Traffic.

Definition

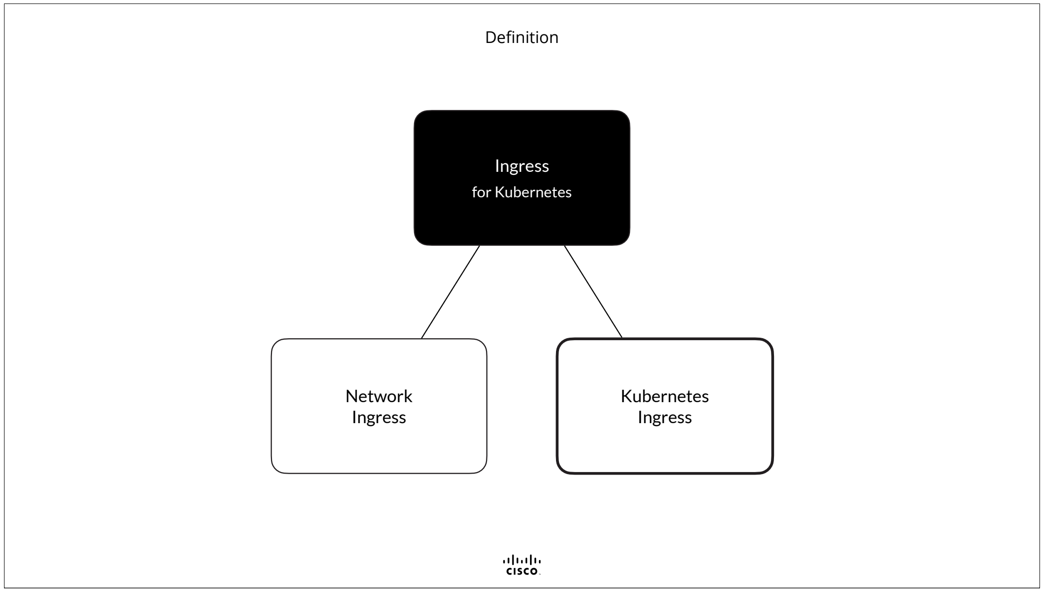
Network Ingress

$$\{ f \in \text{Flow} \mid \text{source}[f] \notin \text{Cluster} \wedge \text{target}[f] \in \text{Cluster} \}$$

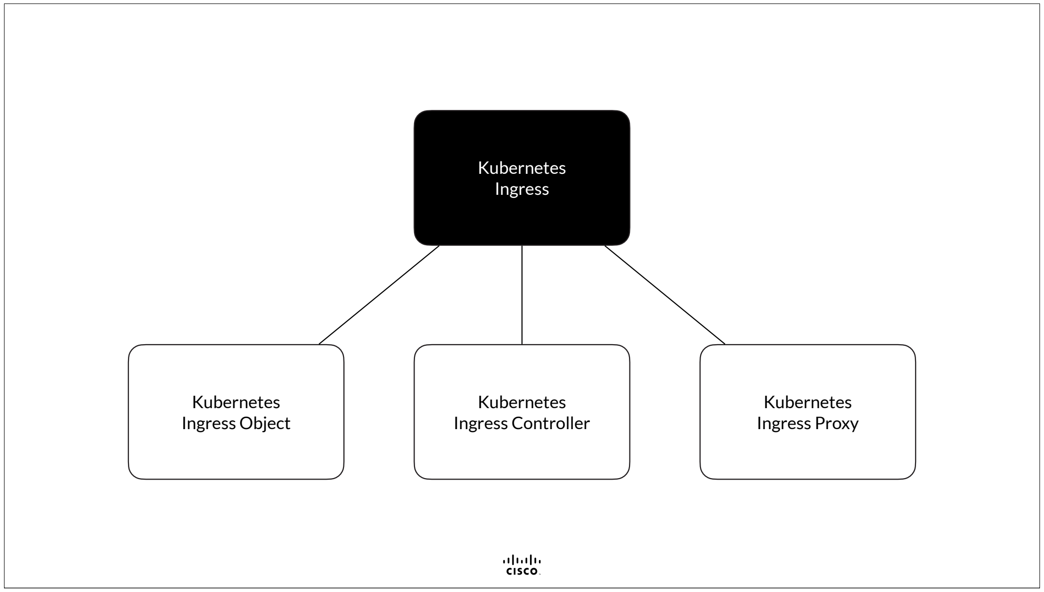
So in conclusion, Network Ingress can be defined as the set of all flows, that originate outside the cluster and terminate inside the cluster


Definition
Ingress
for Kubernetes
Network
Ingress
Kubernetes
Ingress

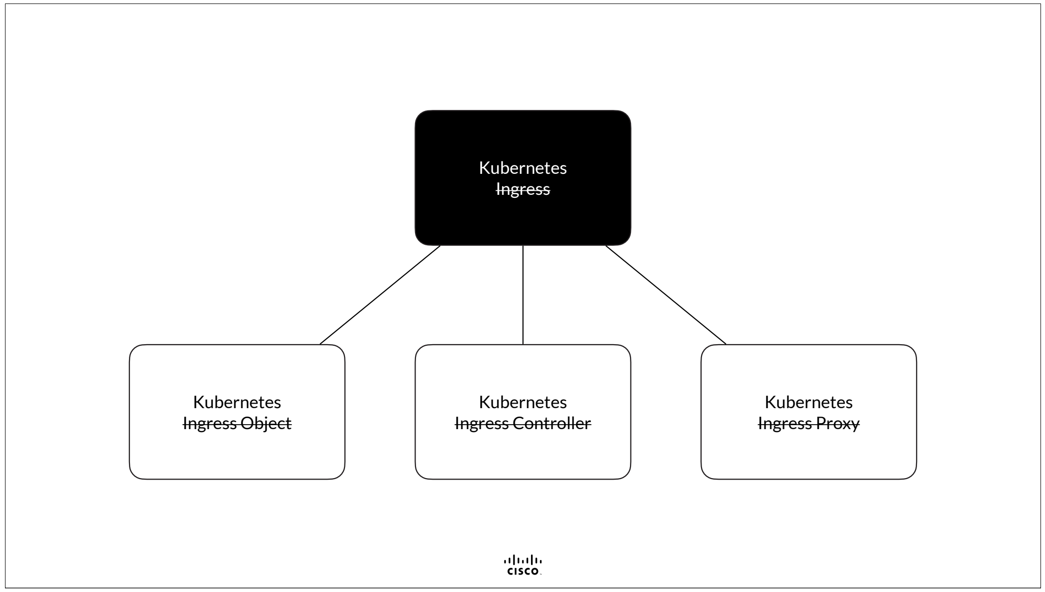
Next up …



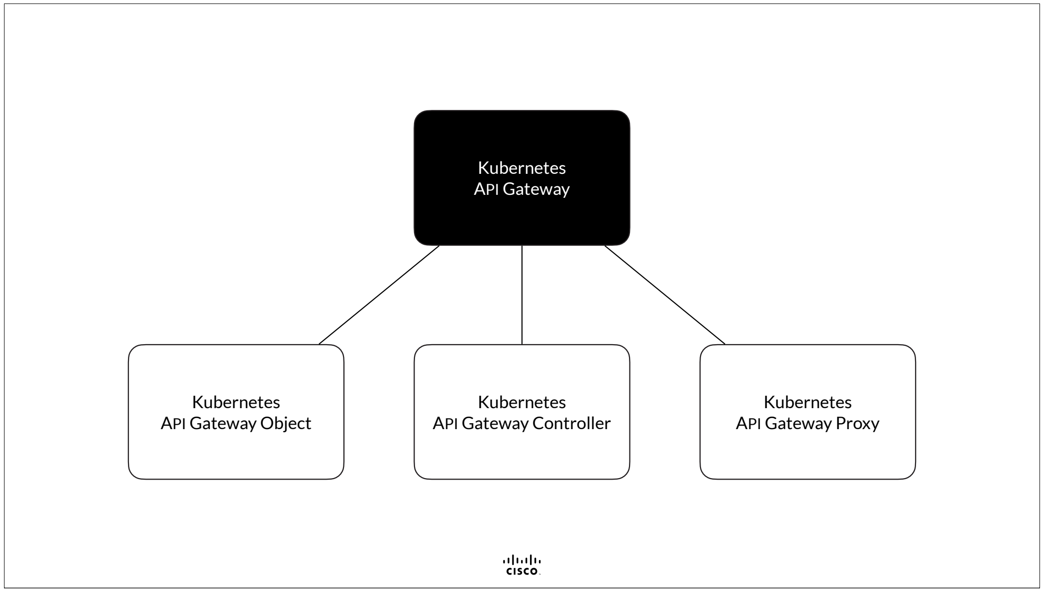
Kubernetes Ingress, the routing of traffic



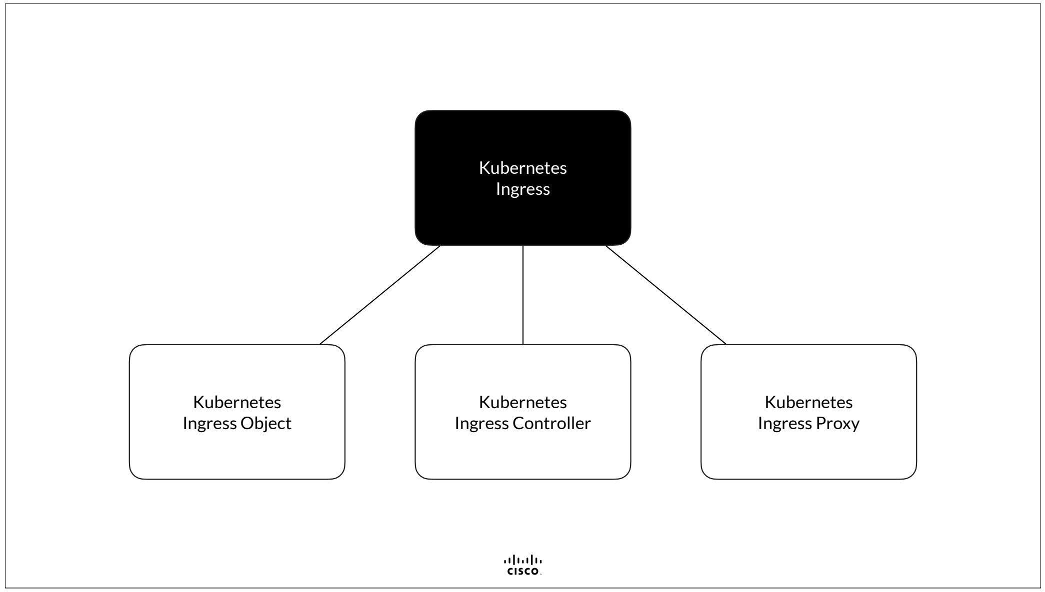
Kubernetes Ingress is composed of three building blocks, the Kubernetes Ingress Resource or Object, Kubernetes Ingress Controller, and the Kubernetes Ingress Proxy



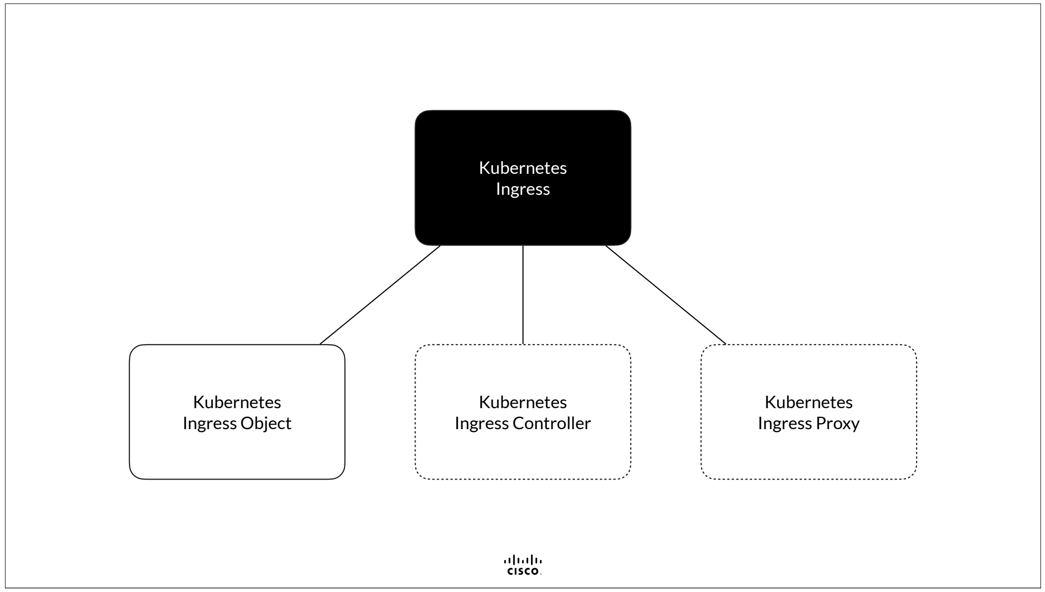
In my personal opinion, given that Kubernetes Ingress does not implement Network Ingress, the name Ingress is not ideal



Personally, I would prefer the name Kubernetes API Gateway

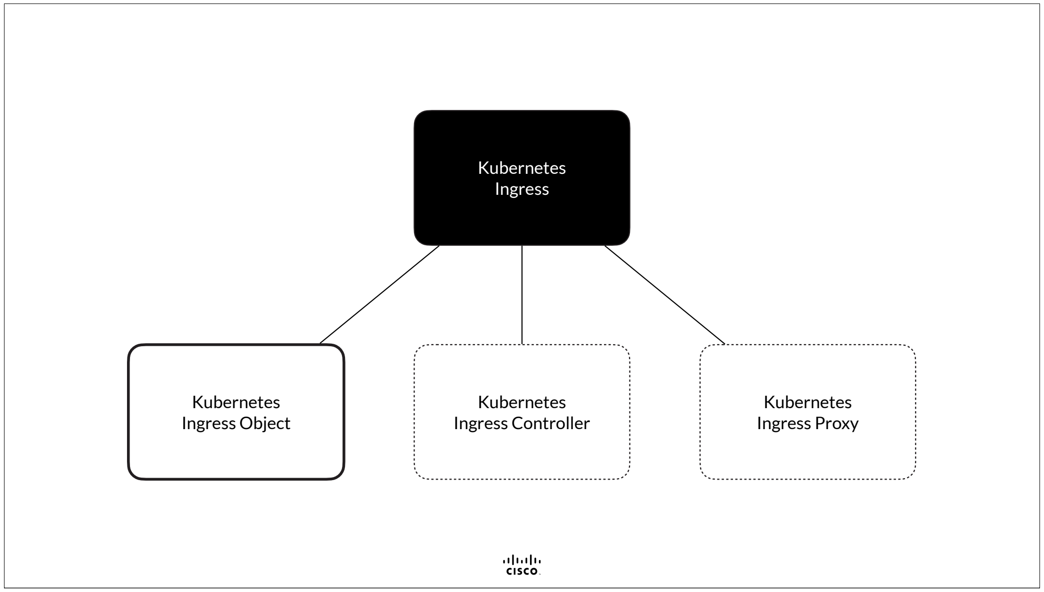


But either way, let’s examine the 3 building blocks of Kubernetes Ingress one by one

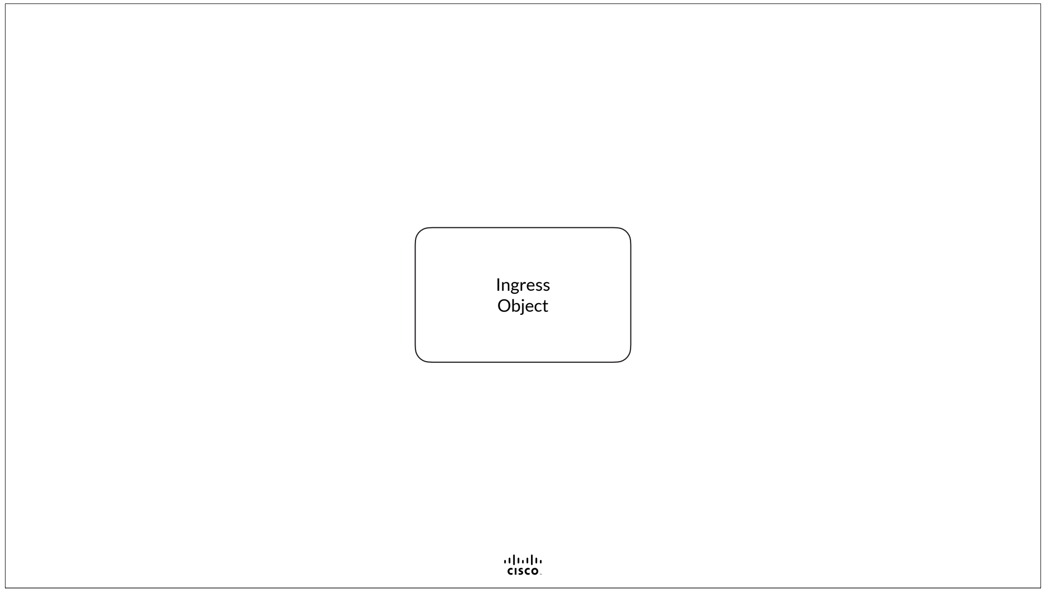


Kubernetes Ingress is indeed exceptional. For core abstractions, Kubernetes provides the resource and the controller out of the box. However, for Kubernetes Ingress, Kubernetes provides only the resource. The cluster operator must choose and instal the ingress controller and ingress proxy of their choice.

So, First up

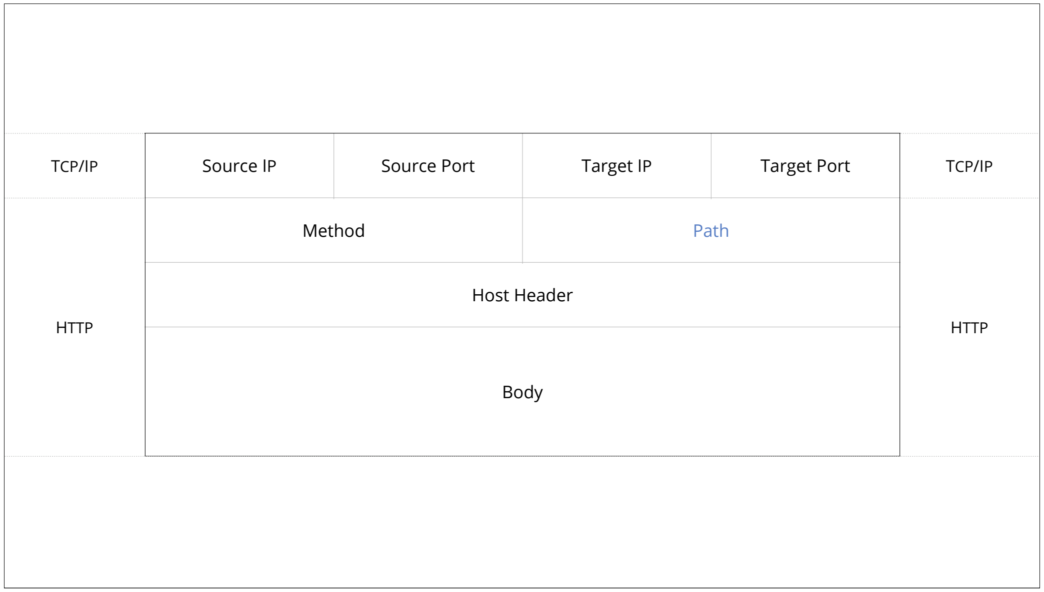


The Kubernetes Ingress Object

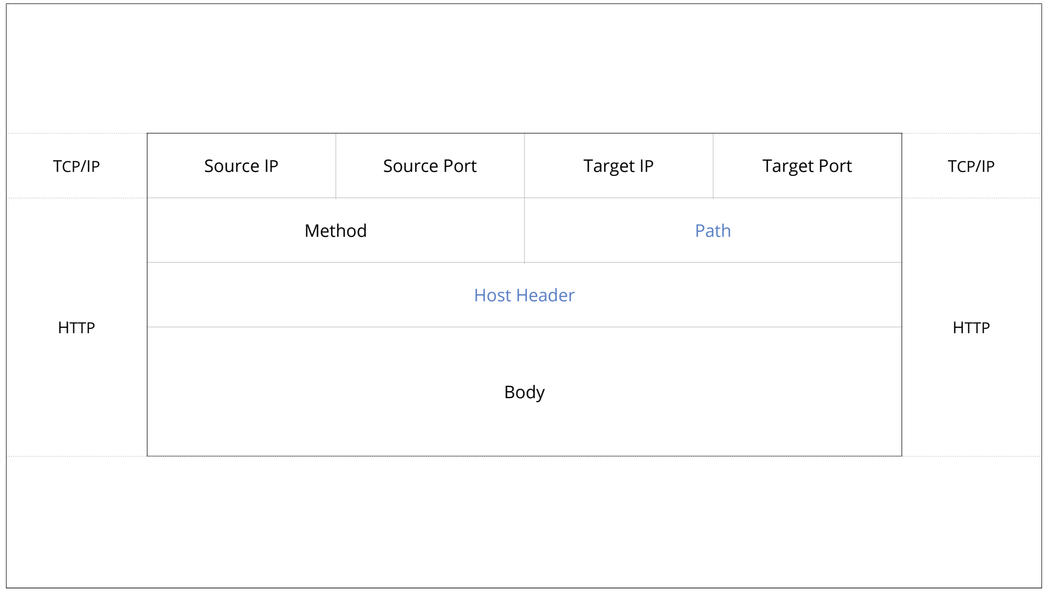


Kubernetes defines a Kubernetes Ingress Object. In effect, the Kubernetes Ingress Object defines a collection of HTTP request-level routing rules that determine the target of that request.

Ingress matches an HTTP request's …


TCP/IP
Source IP
Source Port
Target IP
Target Port
TCP/IP
HTTP
Method
Path
Host Header
Body
HTTP

… Path …



… and Host Header against its routing rules to determine the target Kubernetes Service to proxy the request to

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: foobar
spec:
  rules:
  - host: foo.org
    http:
      paths:
        - path: /a
          backend:
            service:
              name: foo-a
              port:
                number: 8080
        - path: /b
          backend:
            service:
              name: foo-b
              port:
                number: 8181
  - host: bar.org
    http:
      paths:
        - path: /a
          backend:
            service:
              name: bar-a
              port:
                number: 9090
        - path: /b
          backend:
            service:
              name: bar-b
              port:
                number: 9191
```

This example illustrates a Kubernetes Ingress Object. In effect, this Ingress Object defines a collection of 4 request-level routing rules …

Rule	Host	Path	Service
r_1	foo.org	/a	foo-a:8080
r_2	foo.org	/b	foo-b:8181
r_3	bar.org	/a	bar-a:9090
r_4	bar.org	/b	bar-b:9191

that are best represented as a decision table

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: foobar
spec:
  rules:
  - host: foo.org
    http:
      paths:
      - path: /a
        backend:
          service:
            name: foo-a
            port:
              number: 8080
      - path: /b
        backend:
          service:
            name: foo-b
            port:
              number: 8181
  - host: bar.org
    http:
      paths:
      - path: /a
        backend:
          service:
            name: bar-a
            port:
              number: 9090
      - path: /b
        backend:
          service:
            name: bar-b
            port:
              number: 9191
```

For example, the first rule matches an HTTP request with a host header of foo.org and a path of /a to proxy to a Pod that matches a Service named foo-a on port 8080

Rule	Host	Path	Service
r_1	foo.org	/a	foo-a:8080
r_2	foo.org	/b	foo-b:8181
r_3	bar.org	/a	bar-a:9090
r_4	bar.org	/b	bar-b:9191

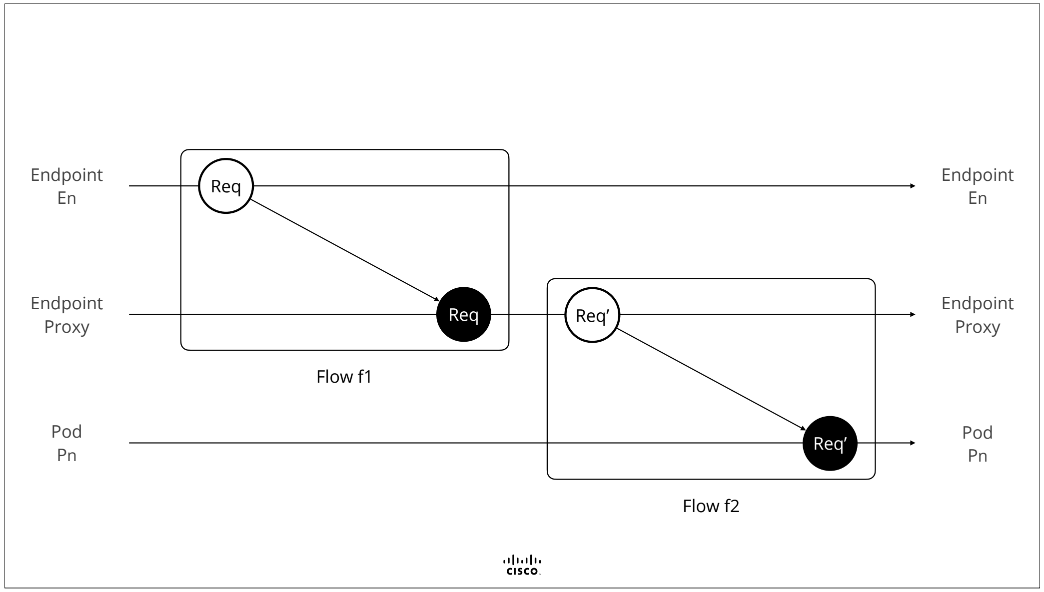
Again, represented as a row in the decision table

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: foobar
spec:
  rules:
  - host: foo.org
    http:
      paths:
        - path: /a
          backend:
            service:
              name: foo-a
              port:
                number: 8080
        - path: /b
          backend:
            service:
              name: foo-b
              port:
                number: 8181
  - host: bar.org
    http:
      paths:
        - path: /a
          backend:
            service:
              name: bar-a
              port:
                number: 9090
        - path: /b
          backend:
            service:
              name: bar-b
              port:
                number: 9191
```

The third rule matches an HTTP request with a host header offer.org and a path of /a to proxy to a Pod that matches a Service named bar-a on port 9090

Rule	Host	Path	Service
r_1	foo.org	/a	foo-a:8080
r_2	foo.org	/b	foo-b:8181
r_3	bar.org	/a	bar-a:9090
r_4	bar.org	/b	bar-b:9191

And again, represented as a row in the decision table

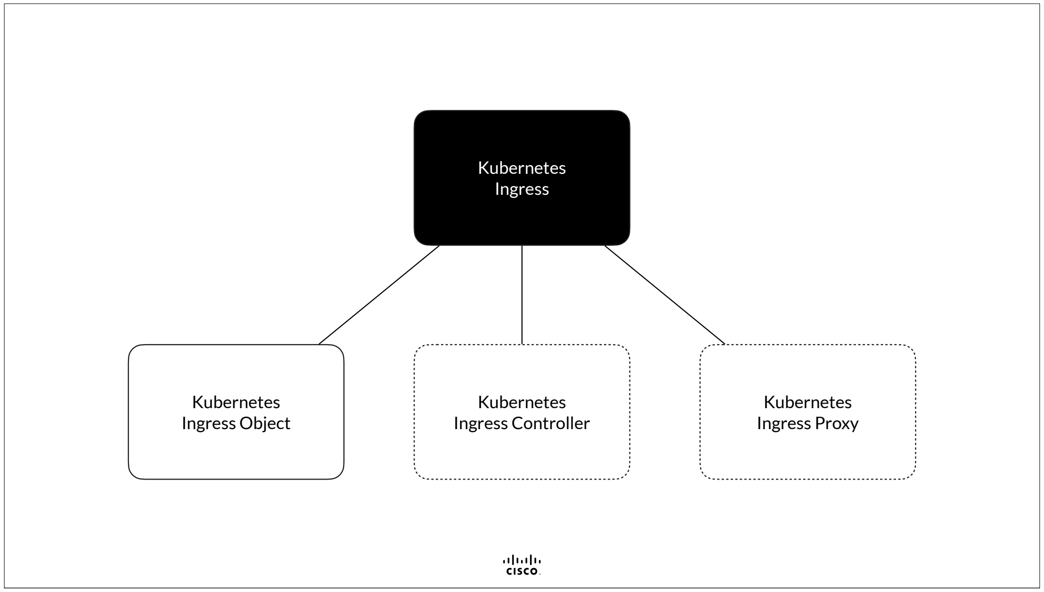


Represented as a Time Space Diagram. When the Kubernetes Ingress Proxy receives a request, it matches the request against the decision table and forwards the request so that a pod that matches the target service receives the request.

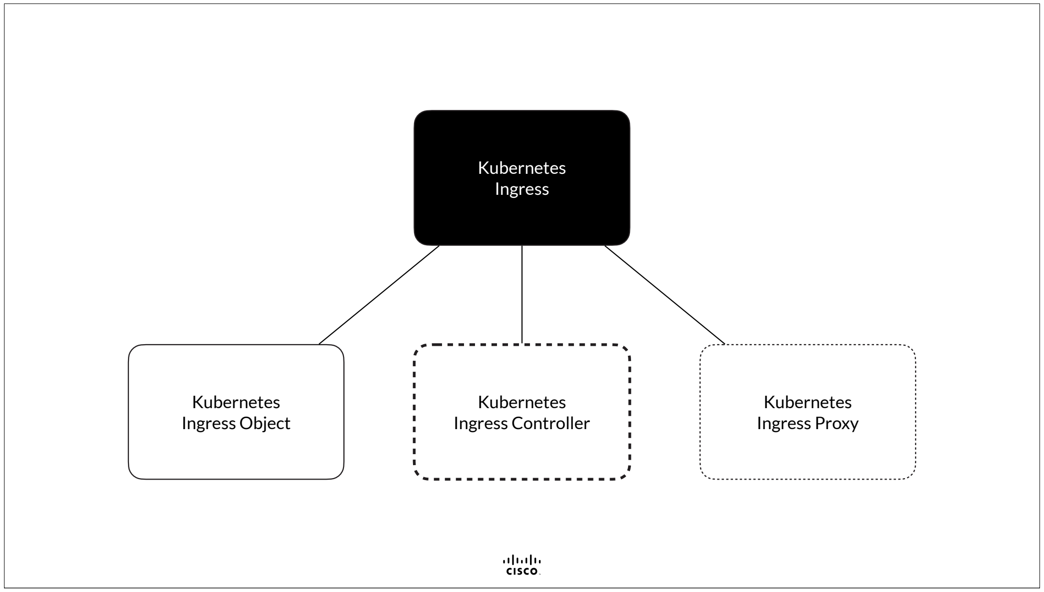
Why do I say "forwards the request so that a pod that matches the target service receives the request" and not simply "forwards the request to the target service"

Because there are implementations that implement their own pod discovery in accordance with Kubernetes services but do not rely on the discovery implemented by Kubernetes and Kubernetes Services.

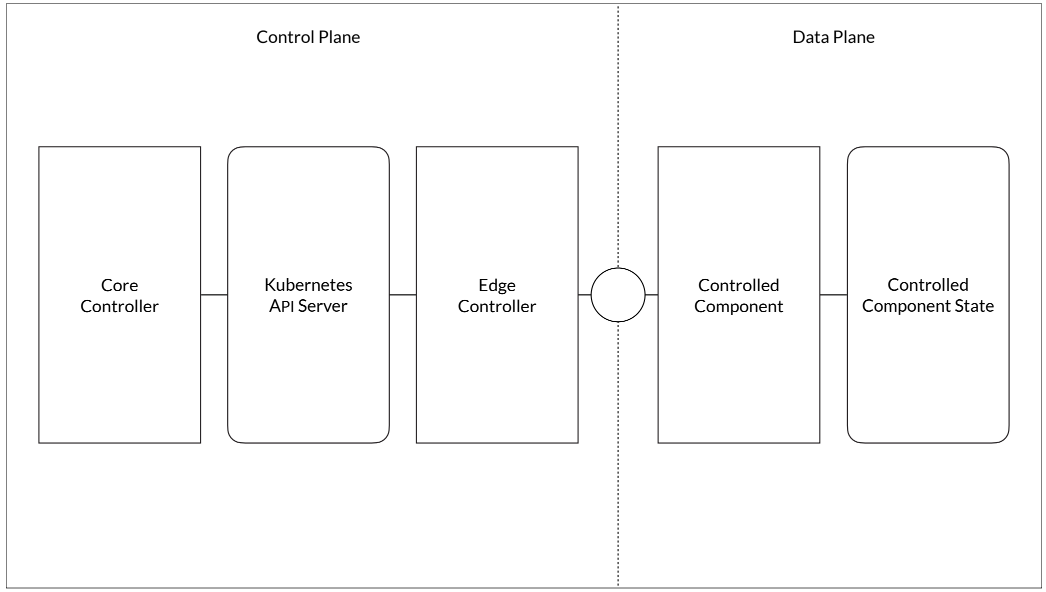
Recv•Req $\in$ History[Proxy] $\wedge$ Recv•Req' $\in$ History[Pn] $\Longrightarrow$ $\exists$ r $\in$ Rules[Proxy] : Req matches condition[rule] $\bigwedge$ Pod matches target-service[r]


Kubernetes
Ingress
Kubernetes
Ingress Object
Kubernetes
Ingress Controller
Kubernetes
Ingress Proxy

Next up …



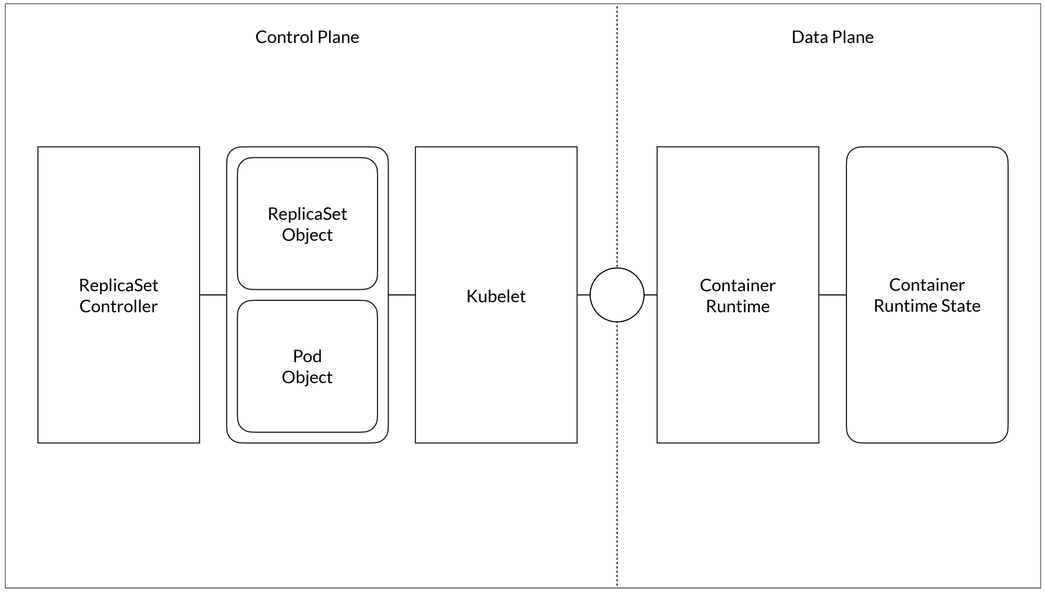
The Kubernetes Ingress Controller, the Control Plane component



Kubernetes centers around the notion of Kubernetes Controllers and Kubernetes Objects. Kubernetes Controllers continuously read and write Kubernetes Objects.

Core Controllers interact exclusively to with the API server to read and write a set of Kubernetes Objects.

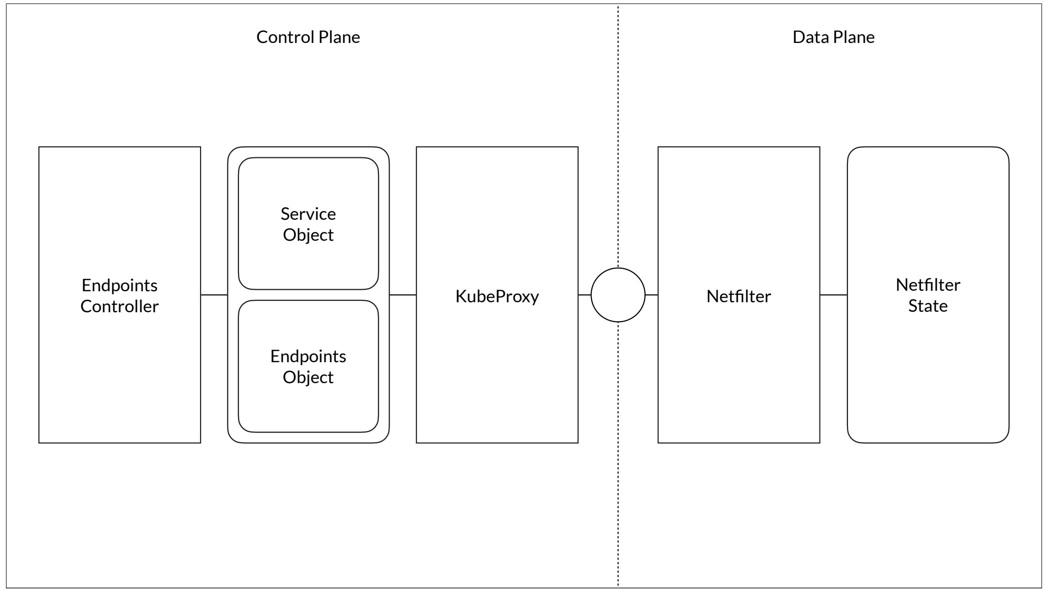
Edge Controller interact with the api server to read and write a set of Kubernetes Objects but additionally communicate with other components



Let’s examine a few familiar examples.

The Kubernetes ReplicaSet Controller is a core controller, it interacts exclusively with the api server. The Replicaset controller read ReplicaSet Objects and writes Pod Objects

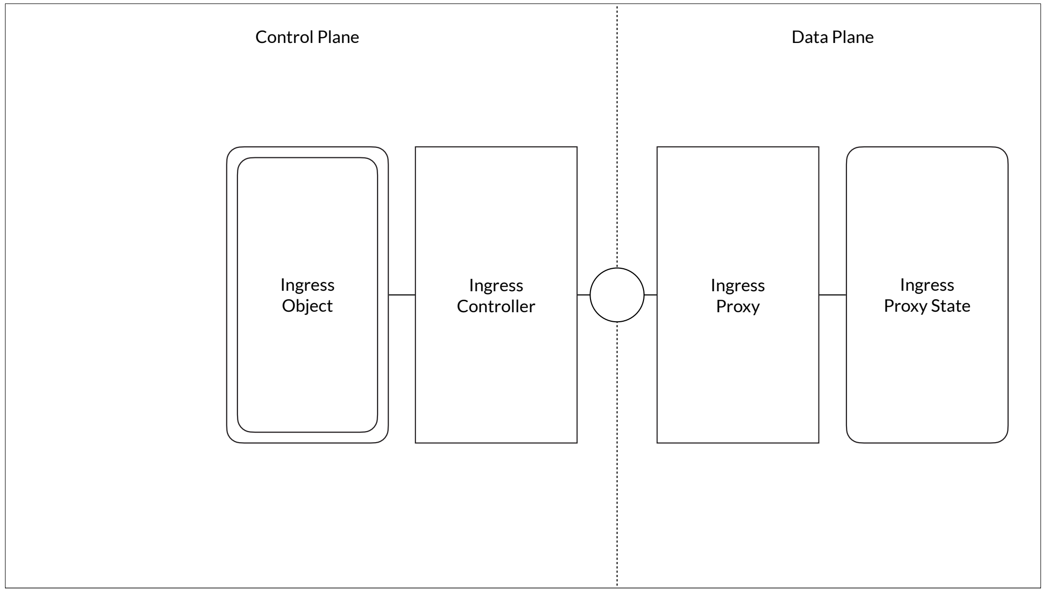
The Kubelet is an edge controller, it interact with the api server and with the container runtime. The Kubelet reads Pod objects and instructs the container runtime to execute containers accordingly



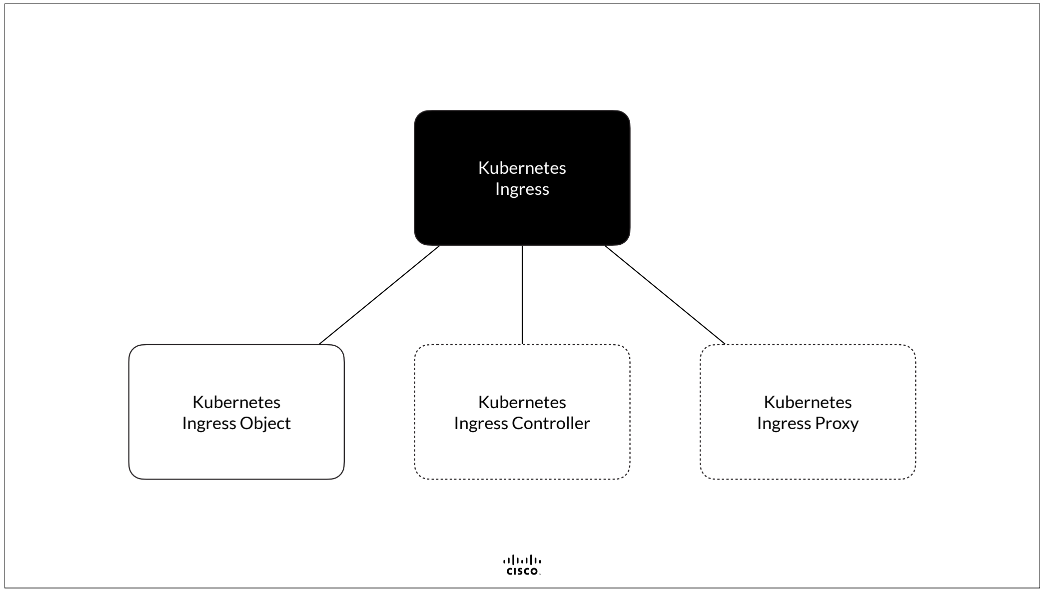
Similarly the Kubernetes Endpoints Controller is a core controller, it interacts exclusively with the api server. The Endpoints controller reads Service Objects and Pod objects and writes Endpoints Objects

The KubeProxy is an edge controller, it interact with the api server and with the Linux Netfilter module. The KubeProxy reads Endpoint objects and instructs the net filter module to create network address translation rules so that a message sent to a service ip address will be forwarded to a pod ip address, with a pod being a member of the endpoints

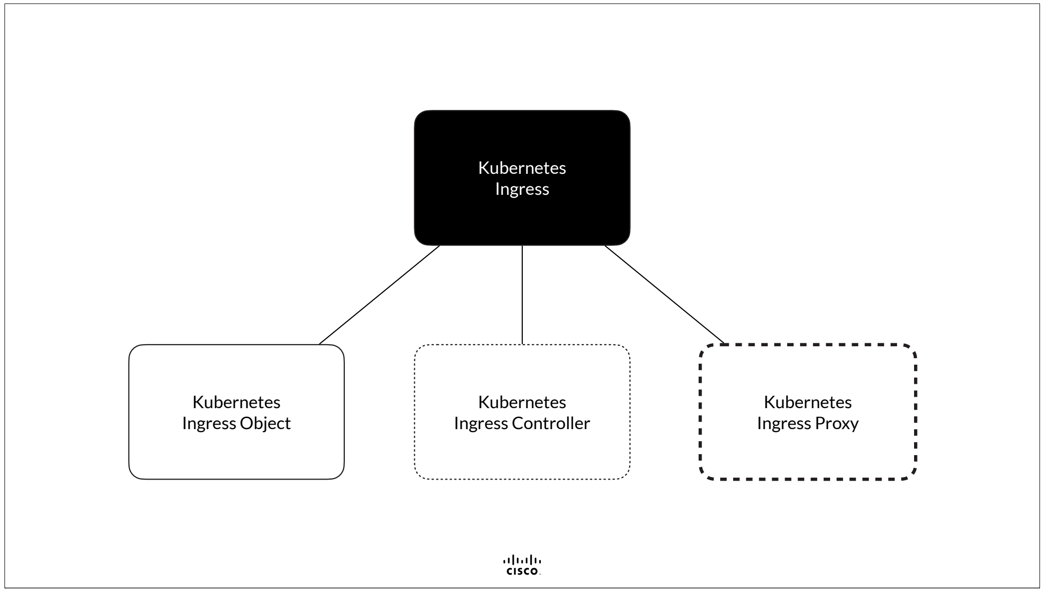
IP Packet
Filter Rules



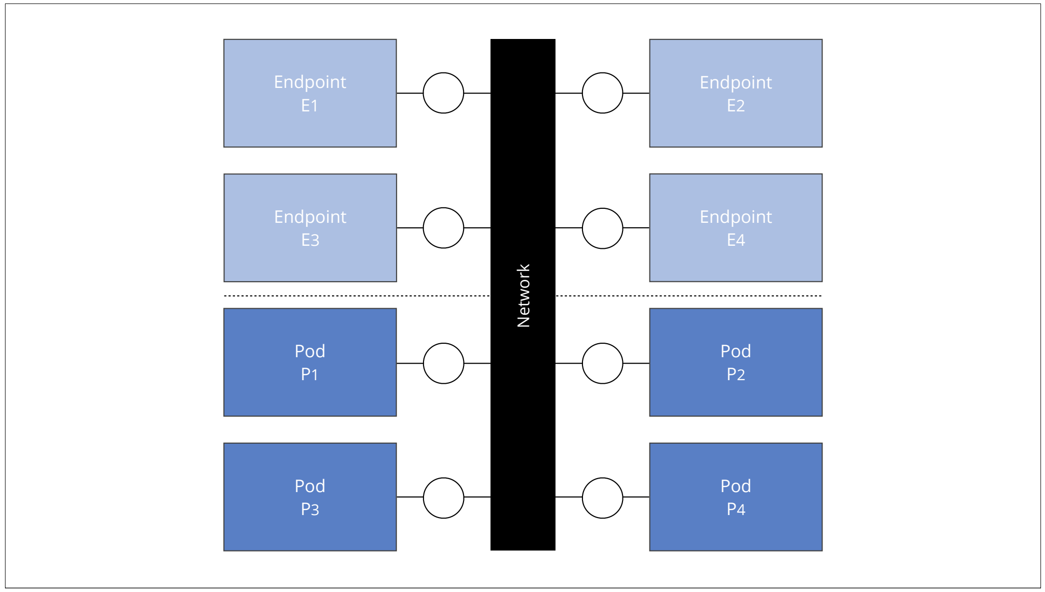
Now onto the ingress controller: an ingress controller is an edge controller, it interacts with the api server and with an ingress proxy. The ingress controller reads ingress objects and instructs the ingress proxy to create routing rules according to the decision table specified in the ingress object



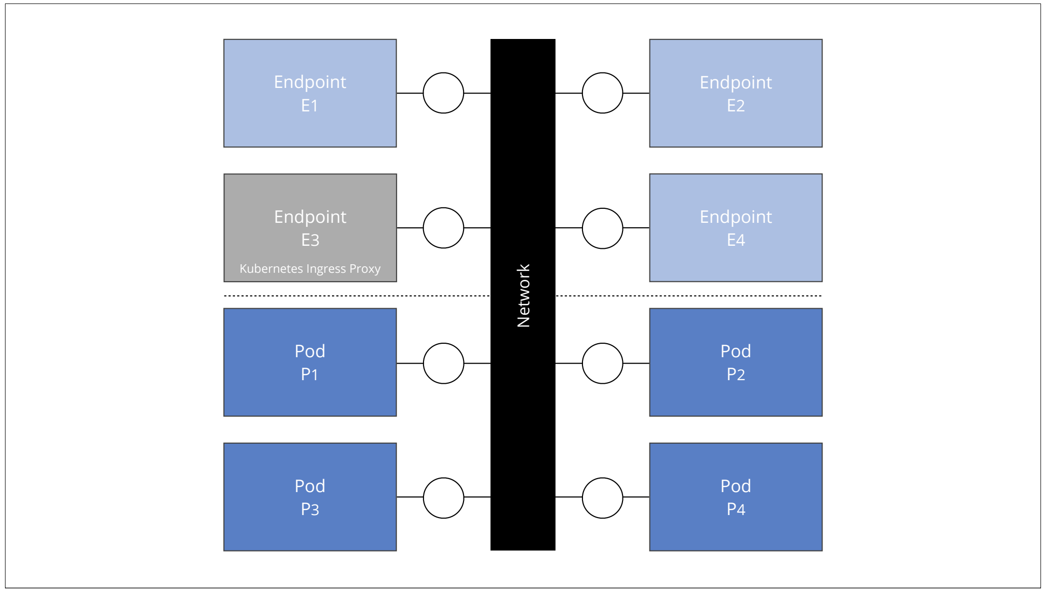
Lastly, Next up



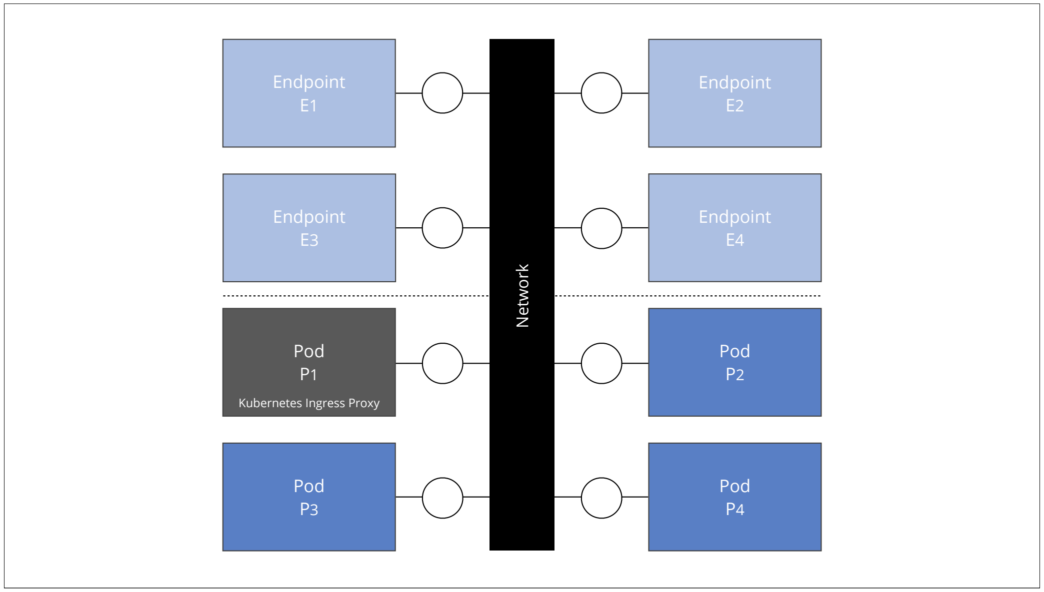
The Kubernetes Ingress Proxy, the Data Plane component



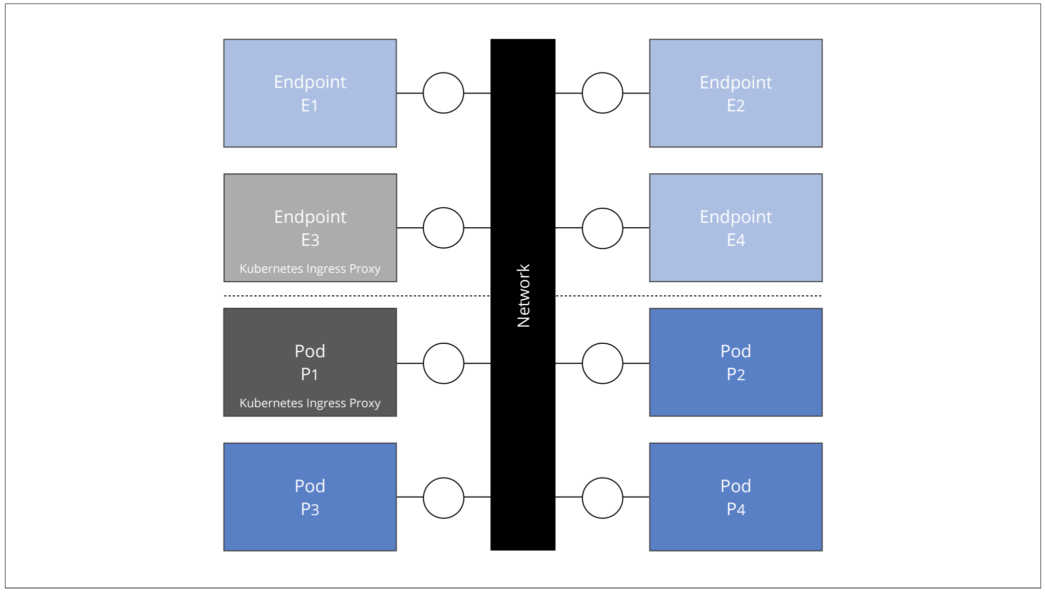
As discussed earlier, network ingress may happen before or after kubernetes ingress, so there are two possibilities …



The ingress proxy may be an external endpoint …



Or the ingress proxy may be an internal endpoint, a pod …



But either way, the task of the ingress proxy is to accept the request, match the request against the decision table specified by the ingress object and installed by the ingress controller and forward the request so that a pod that matches the target service receives the request

Definition

Kubernetes Ingress

$\{ f1 \times f2 \in Flow \times Flow \mid target[f1] = Proxy \wedge target[f2] = Pod \wedge \exists r \in Rules[Proxy]: Req_{f1} \vdash conditions[r] \wedge Pod \vdash target[r] \}$

So in conclusion, Kubernetes Ingress can be defined as the set of all flow pairs, so that the first flow terminates at the proxy, the second flow terminates at a pod and there exists a rule in the decision table, so that the request of the first flow matches the conditions of the rule and the pod matches the target service of the rule

Definition

Ingress for Kubernetes

$\{ f1 \times f2 \in \text{Flow} \times \text{Flow} \mid \text{source}[f1] \notin \text{Cluster} \wedge \text{target}[f2] \in \text{Cluster} \wedge \text{target}[f1] = \text{Proxy} \wedge \text{target}[f2] = \text{Pod} \wedge \exists \ldots \}$

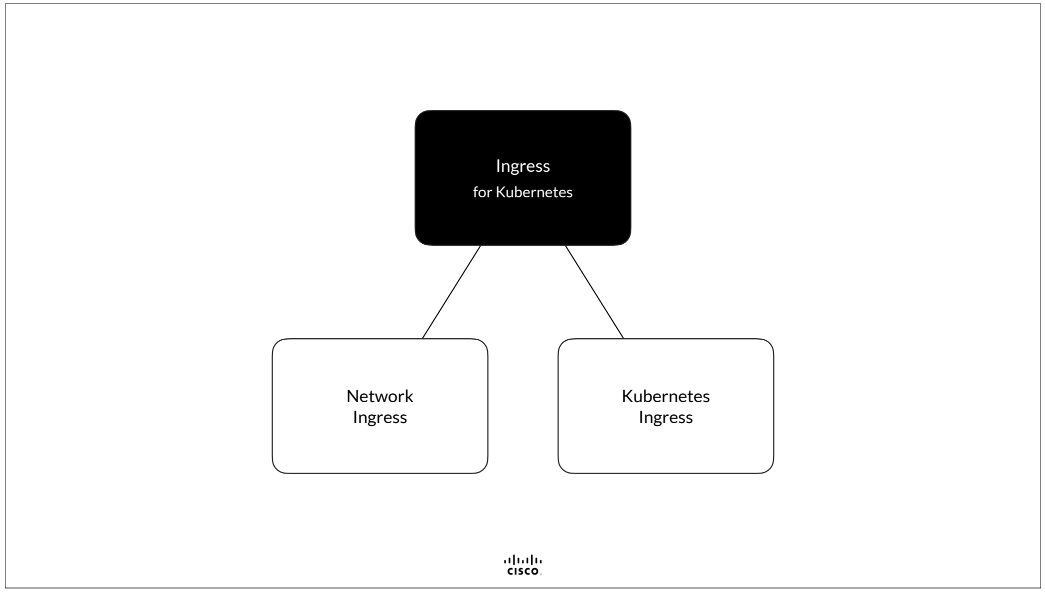
cisco

And now, equipped with this knowledge, we can define Ingress for Kubernetes, simply by composing both formulas

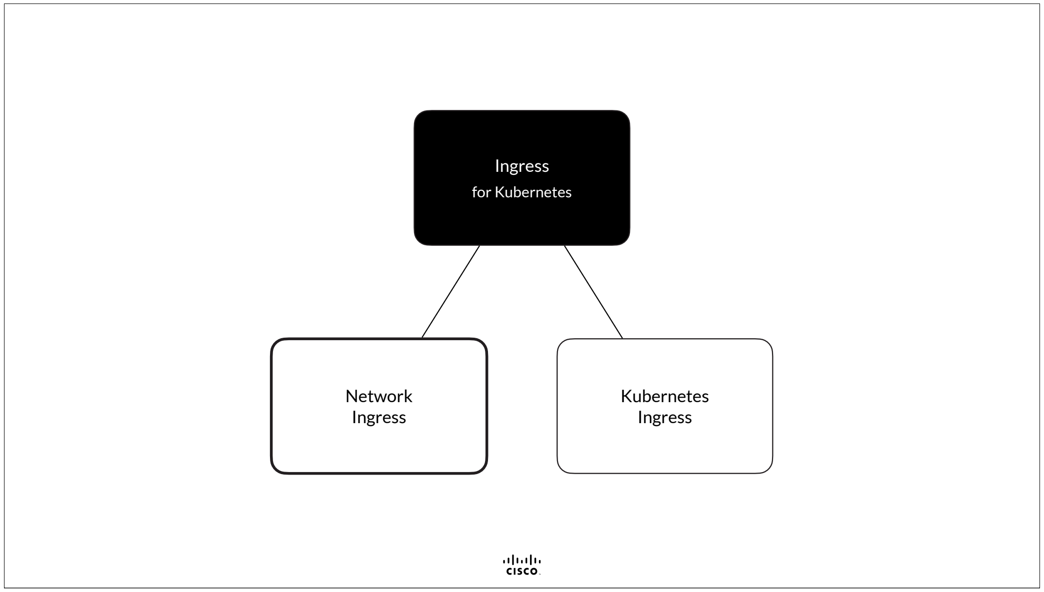
Ingress for Kubernetes is the set of all flow pairs, so that the first flow origins outside the cluster and terminates at the proxy, the second flow terminates inside the cluster at a pod and there exists a rule in the decision table, so that the request of the first flow matches the conditions of the rule and the pod matches the target service of the rule

Conclusion

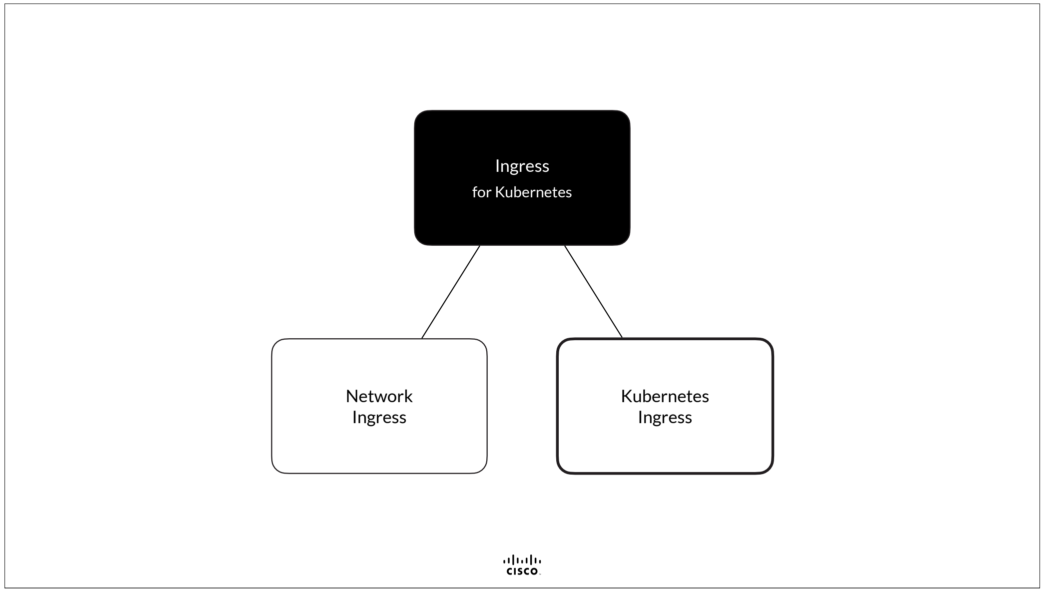
Let’s conclude



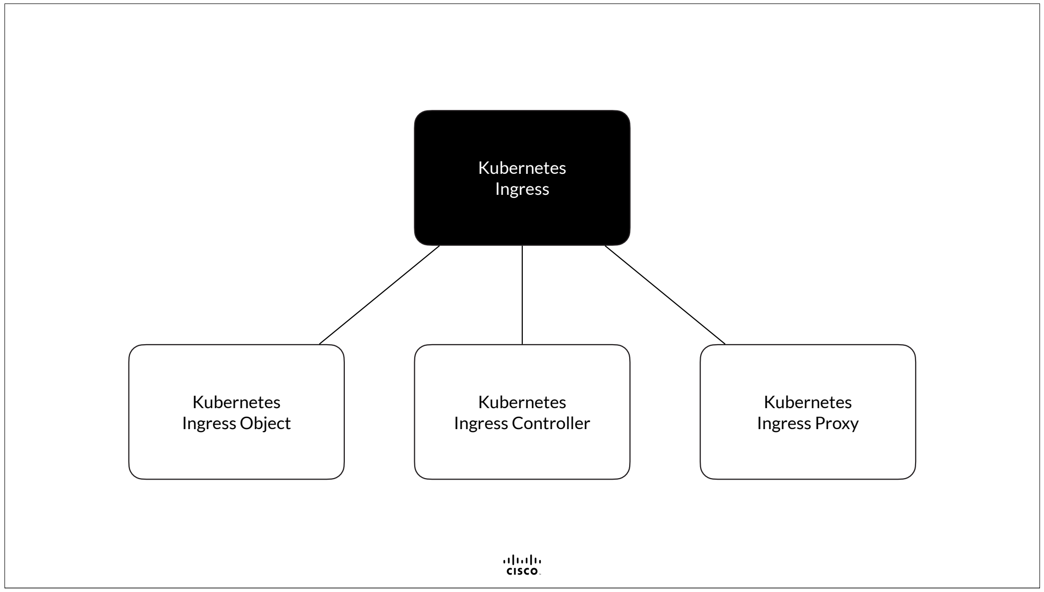
Ingress for Kubernetes encompasses two aspects …



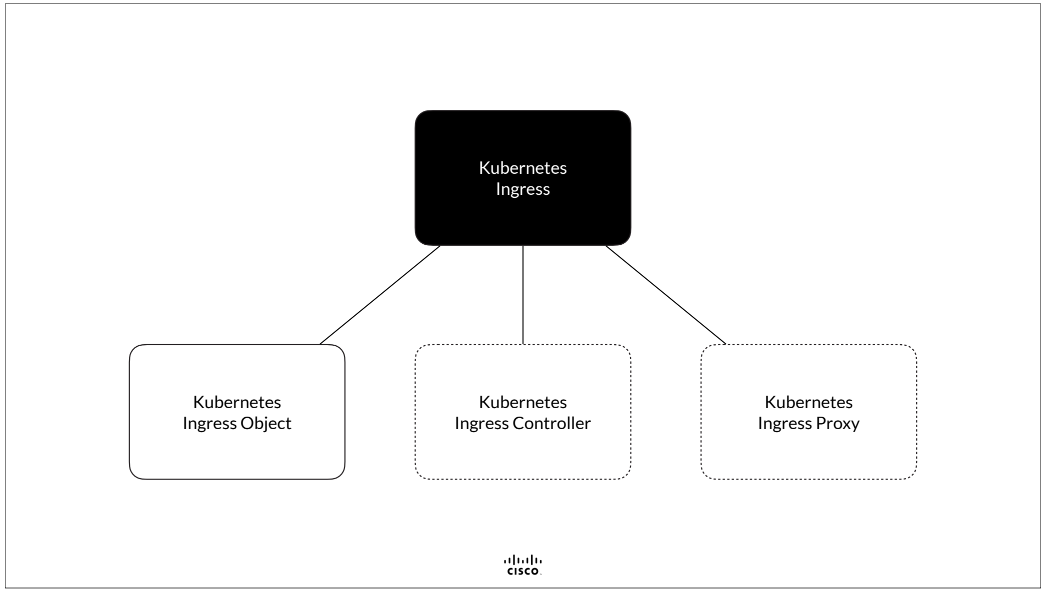
Network ingress, the admission of traffic into the cluster



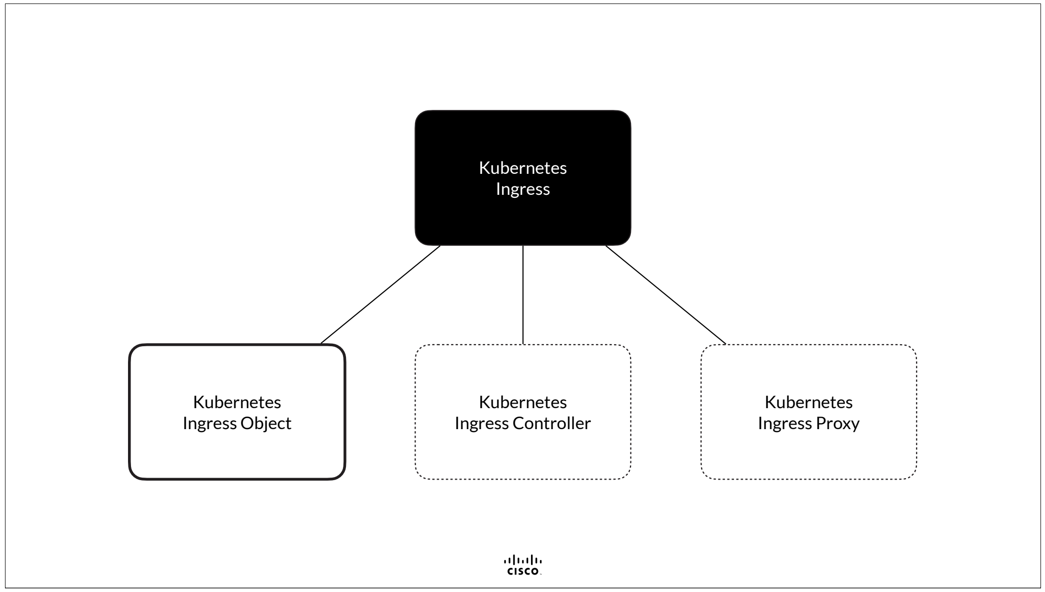
And Kubernetes ingress, the routing of traffic pithing the cluster



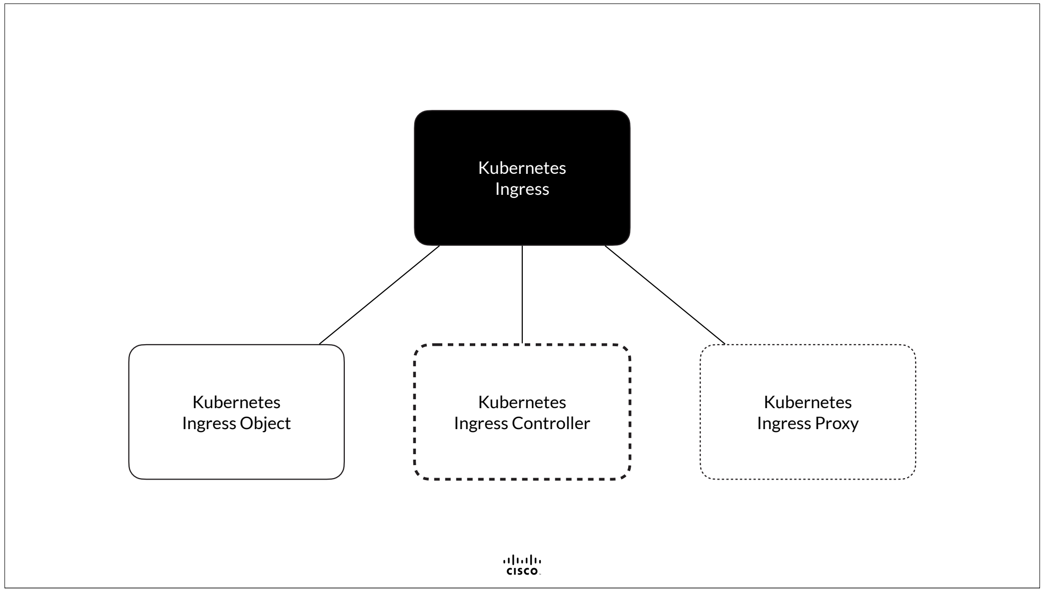
Kubernetes Ingress is composed of three building blocks, the Kubernetes Ingress Resource or Object, Kubernetes Ingress Controller, and the Kubernetes Ingress Proxy



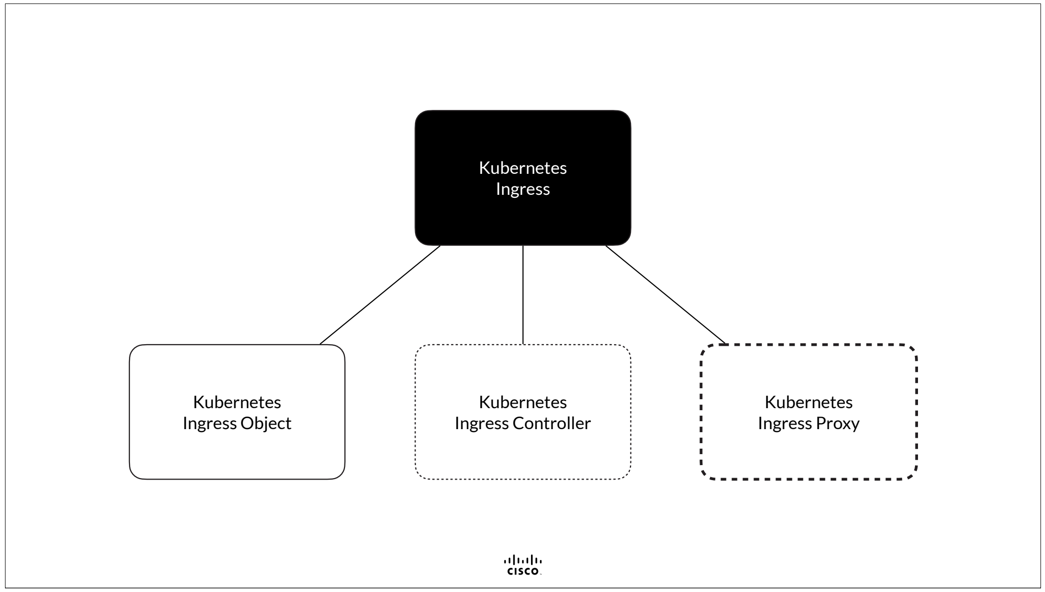
However, Kubernetes provides only the Ingress Object, Ingress Controller and Ingress Proxy are third party components



In effect, the Kubernetes Ingress Object defines a collection of HTTP request-level routing rules that determine the target of that request.

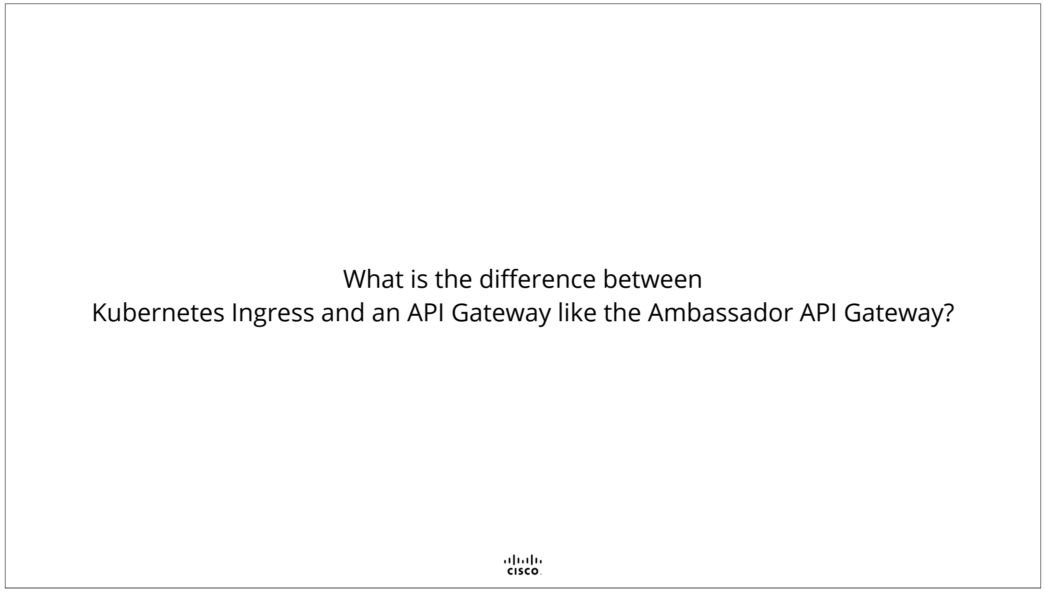


The ingress controller reads ingress objects and instructs the ingress proxy to create routing rules according to the decision table specified in the ingress object

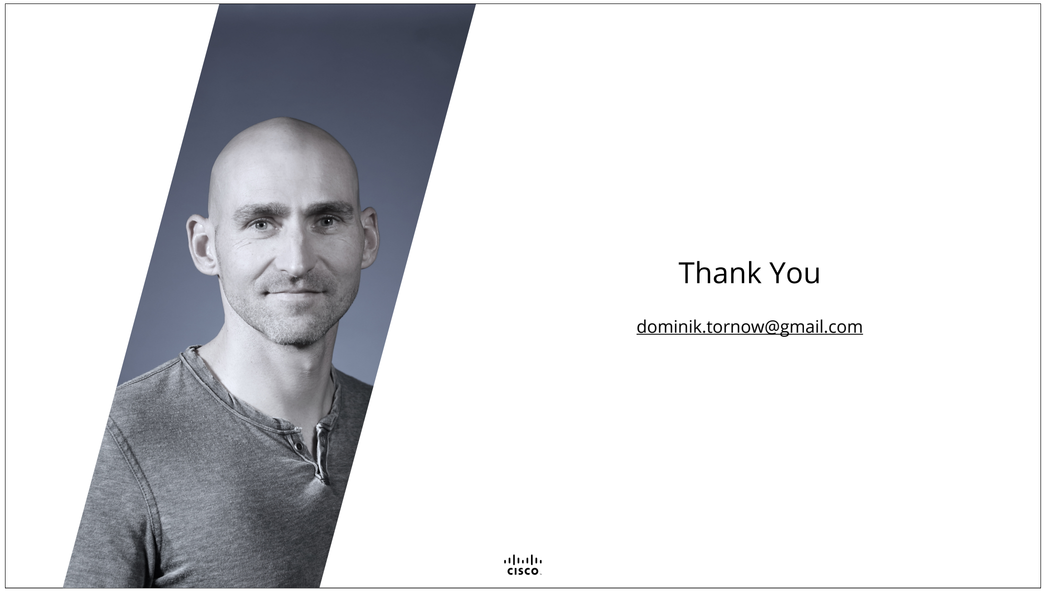


the ingress proxy accepts the request, match the request against the decision table specified by the ingress object and installed by the ingress controller and forward the request so that a pod that matches the target service receives the request

Finally …



That, of course, is a trick question. In effect, the concept of Kubernetes ingress is the concept of an API Gateway, in effect, the kubernetes ingress object is a standardized configuration for API Gateways. Popular API Gateways, like the Ambassador API gateway can be installed to read the Ingress Object and act as the Ingress Controller and Ingress Proxy



If you are watching this presentation during the conference, I will be happy to answer your questions online. If you are watching this presentation after the conference, I will be happy to answer your questions via email.

But either way, Thank you for watching Inside Kubernetes Ingress.