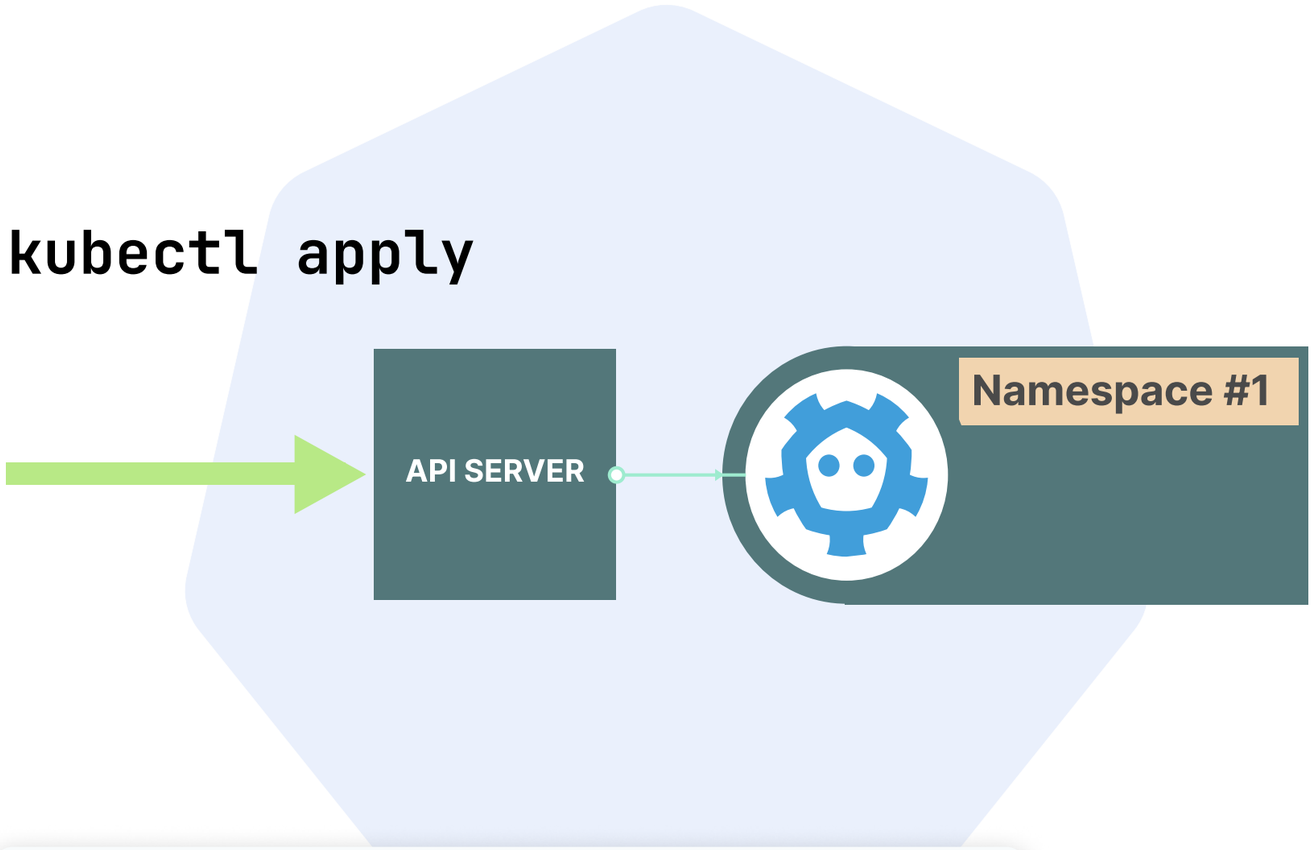
KUBERNETES NAMESPACES

Explore how a simple resource (that does not exist) has vast implications in your Kubernetes cluster.

Daniele Polencic



Namespaces are one of the fundamental resources in Kubernetes.

But they don't provide network isolation, are ignored by the scheduler and can't limit resource usage.

How do they actually work, and what are they useful for?

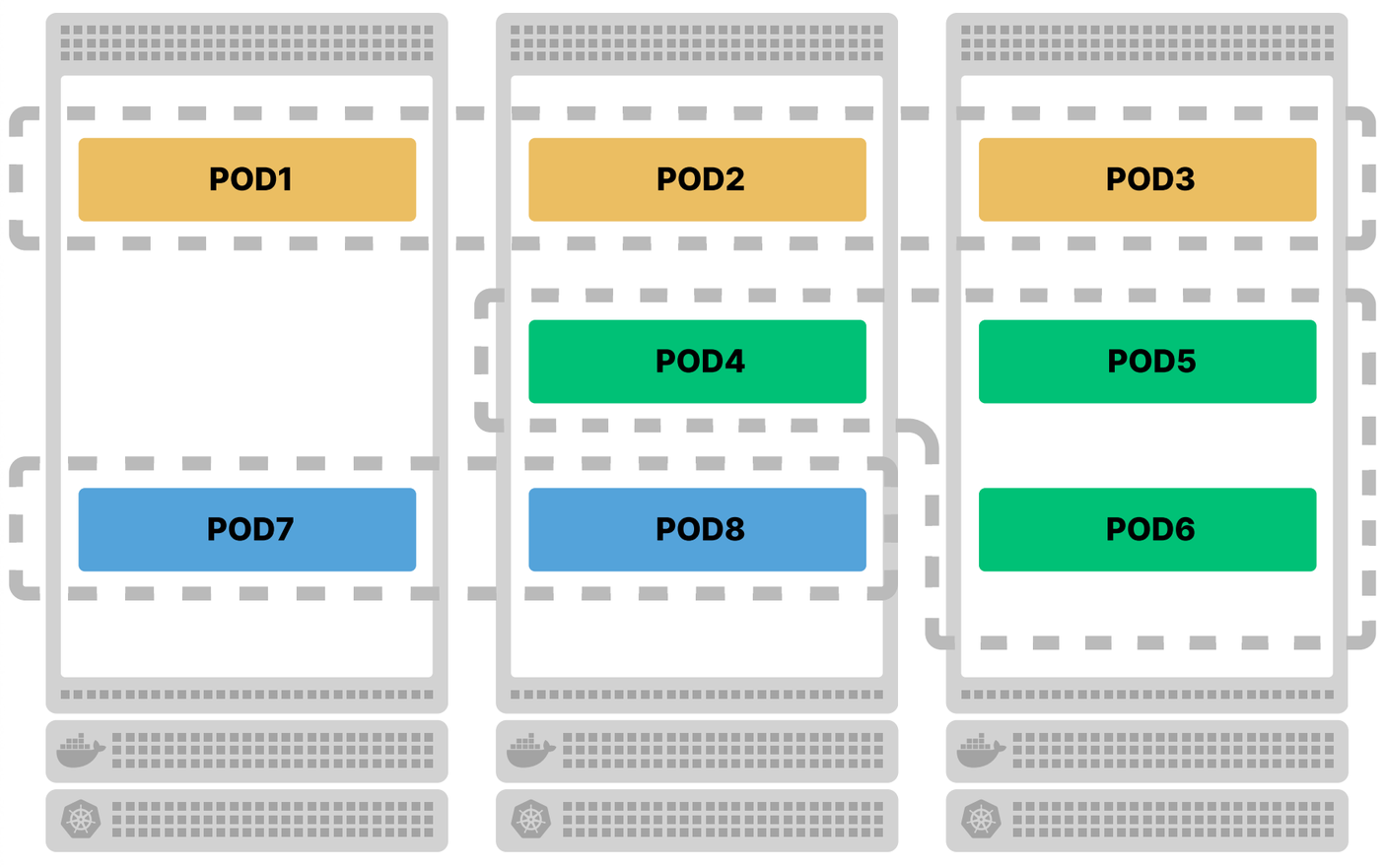
learnk8s

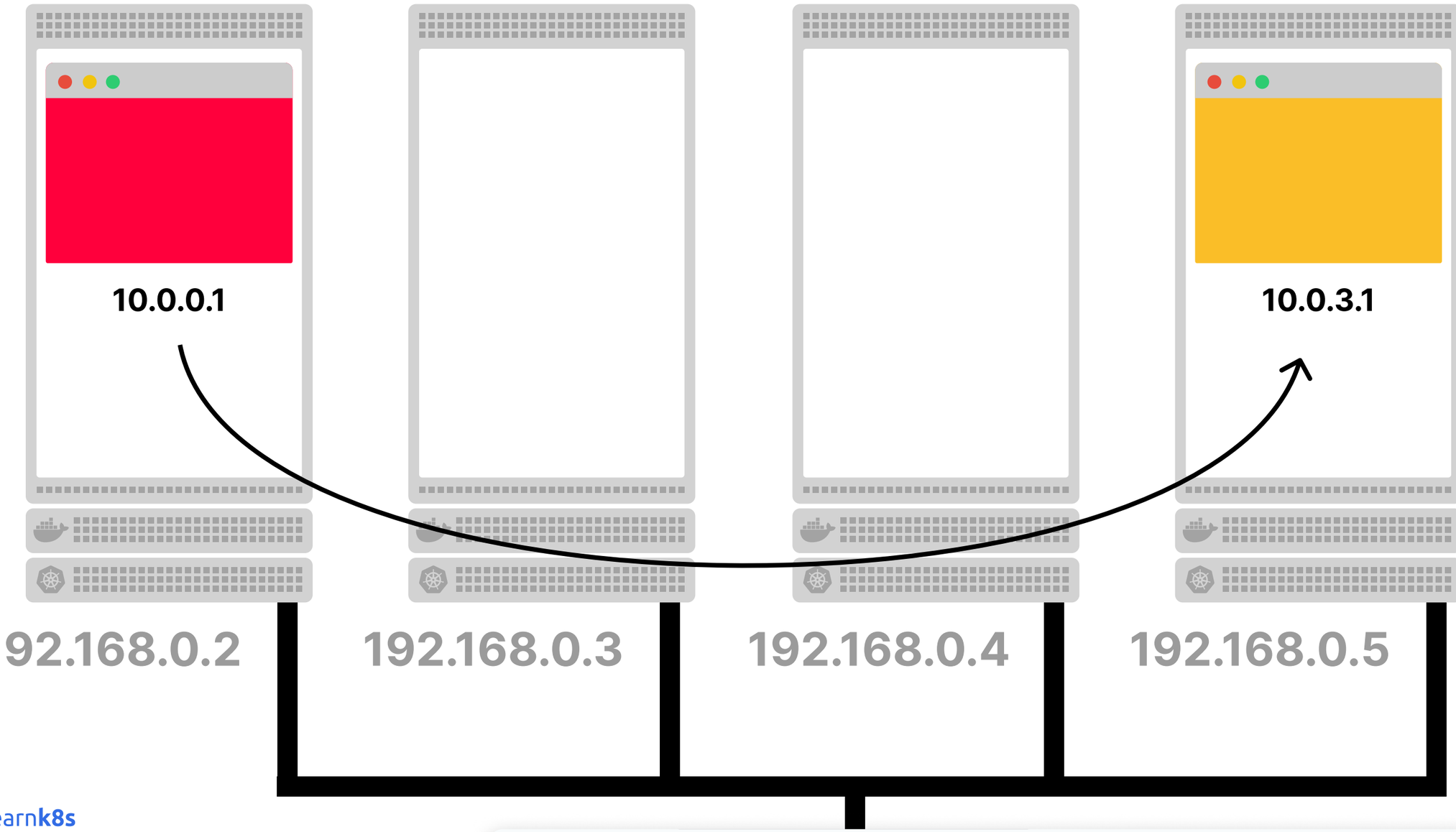
Namespaces are not real and don't exist in the infrastructure.

You could think of namespaces as labels that you can use to group resources when you list them.

Example: show me all pods in the default (label) namespace.

We often picture them as flexible boundaries around objects, but there's no real grouping of resources in the node.





learnk8s

FROM:

Pod ■ **(10.0.0.1)**

TO:

Pod ■ **(10.0.3.1)**

The Kubernetes network should always satisfy two requirements:

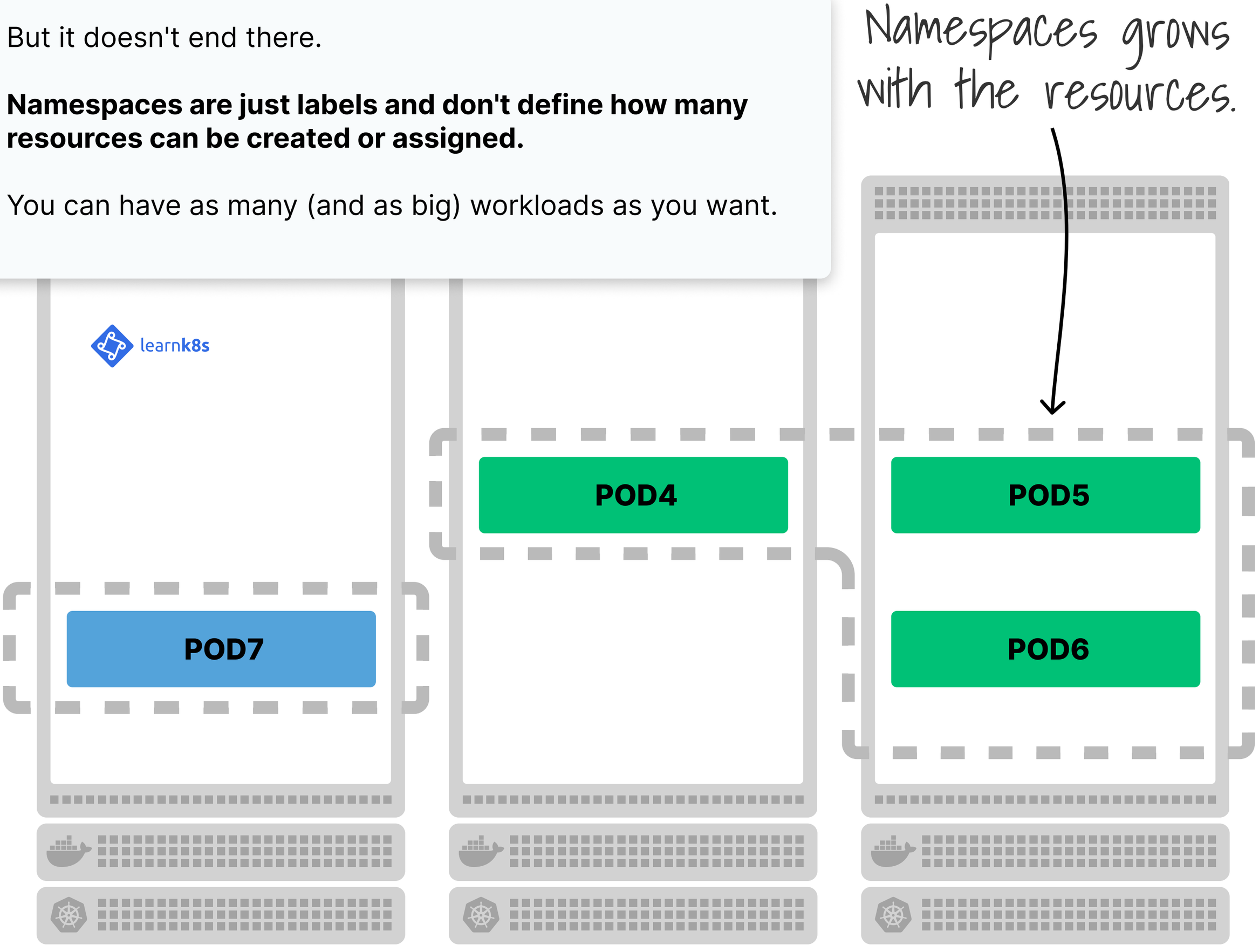
1. Any pod can talk to any pod.
2. Pods have "stable" IP addresses.

Notice how the first requirement already invalidates the idea of namespaces as security boundaries.

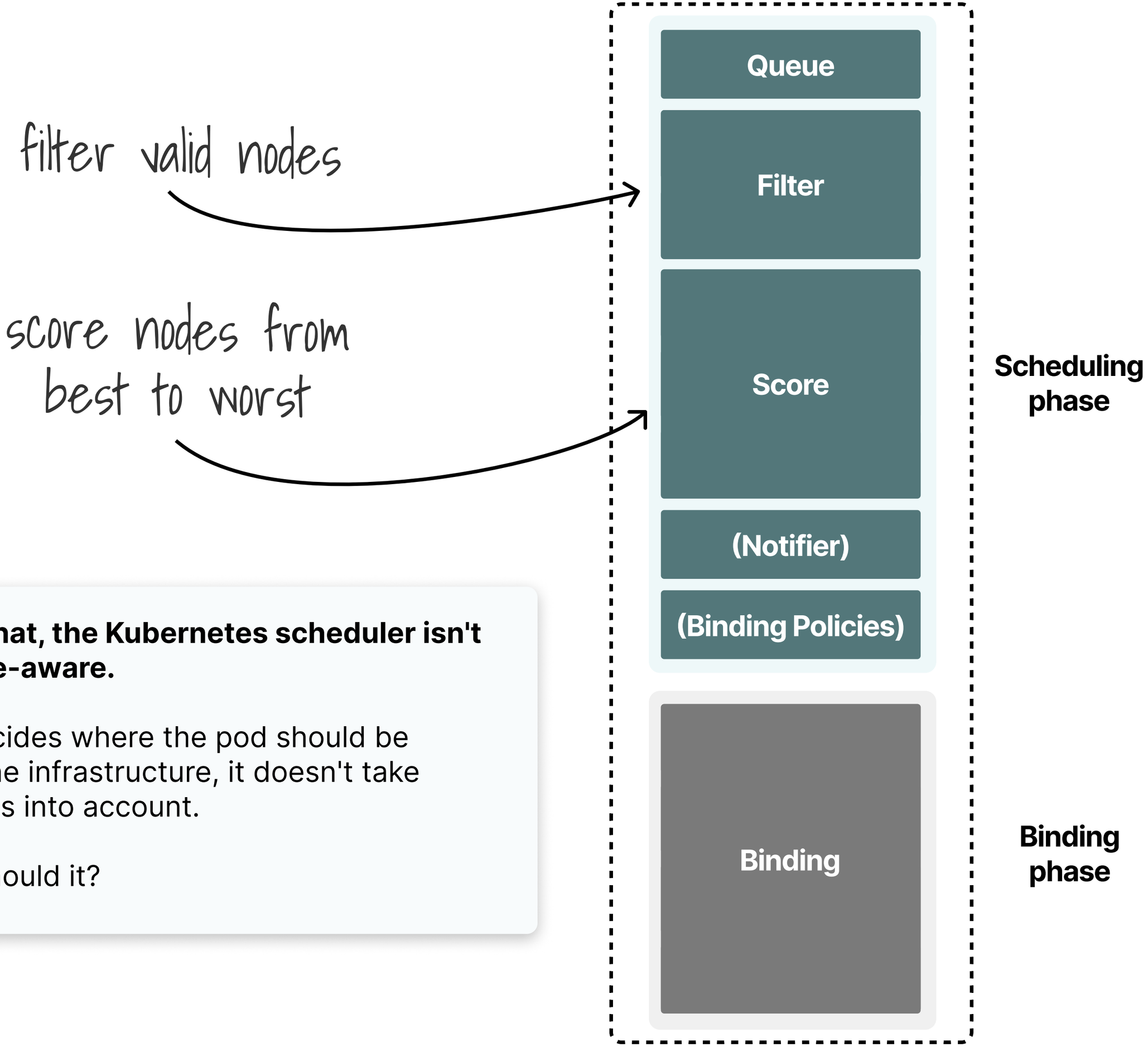
But it doesn't end there.

Namespaces are just labels and don't define how many resources can be created or assigned.

You can have as many (and as big) workloads as you want.



learnk8s



On top of that, the Kubernetes scheduler isn't namespace-aware.

When it decides where the pod should be placed in the infrastructure, it doesn't take namespaces into account.

And why should it?

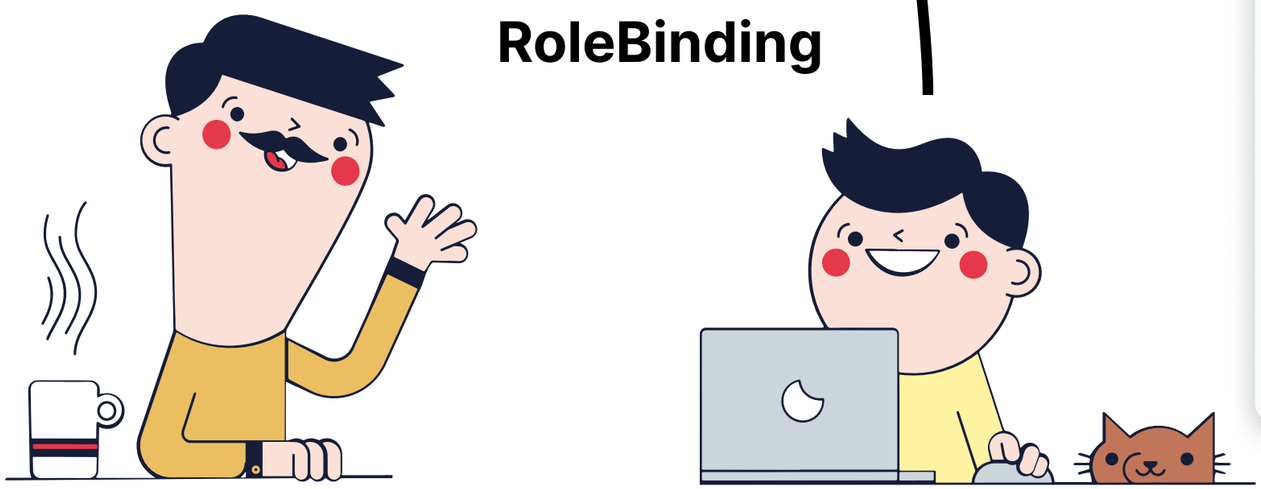
UID	CLUSTER ROLE	PODS read	PODS write	DEPLOYMENT read	DEPLOYMENT write
1	admin1	✓	✓	✓	✓
2	debug	✓	✓	✗	✗
3	reviewer	✓	✗	✓	✗

UID	ROLE	PODS read	PODS write	DEPLOYMENT read	DEPLOYMENT write
1	teamA	✓	✓	✓	✓
2	QA	✓	✗	✓	✗

DEV namespace

oleBinding

RoleBinding



Identity1

Normal user

Identity2

Service Account

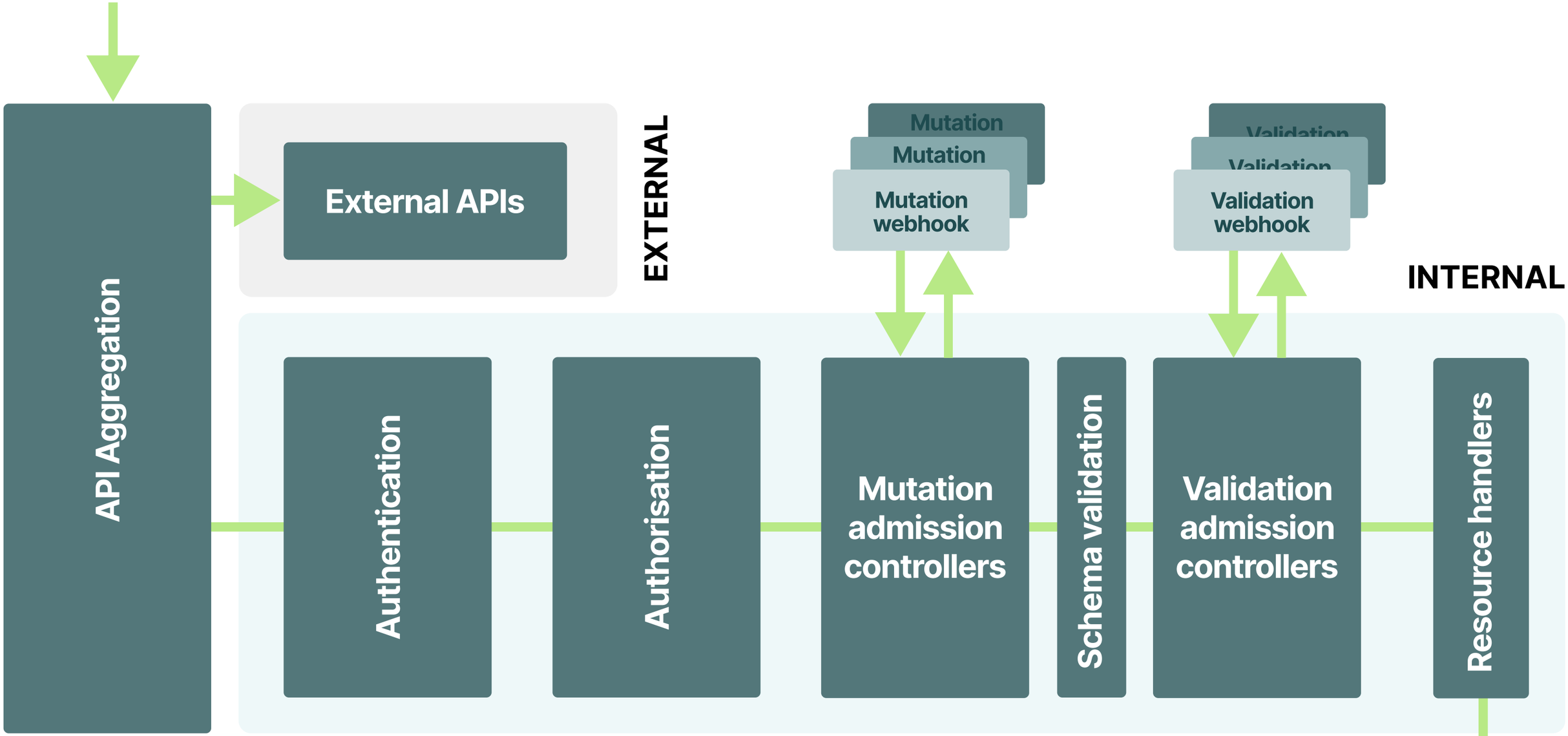
When we look at permissions and RBAC, we see things changing.

You have ClusterRoles that grant you access to resources regardless of the namespace.

But you also have Roles that are restricted by it.

And here's the first clue on how namespaces work: the Kubernetes API.

learnk8s



If you are familiar with the Kubernetes API, you might know there are **Validating and Mutating controllers designed to check and mutate resources before they reach etcd.**

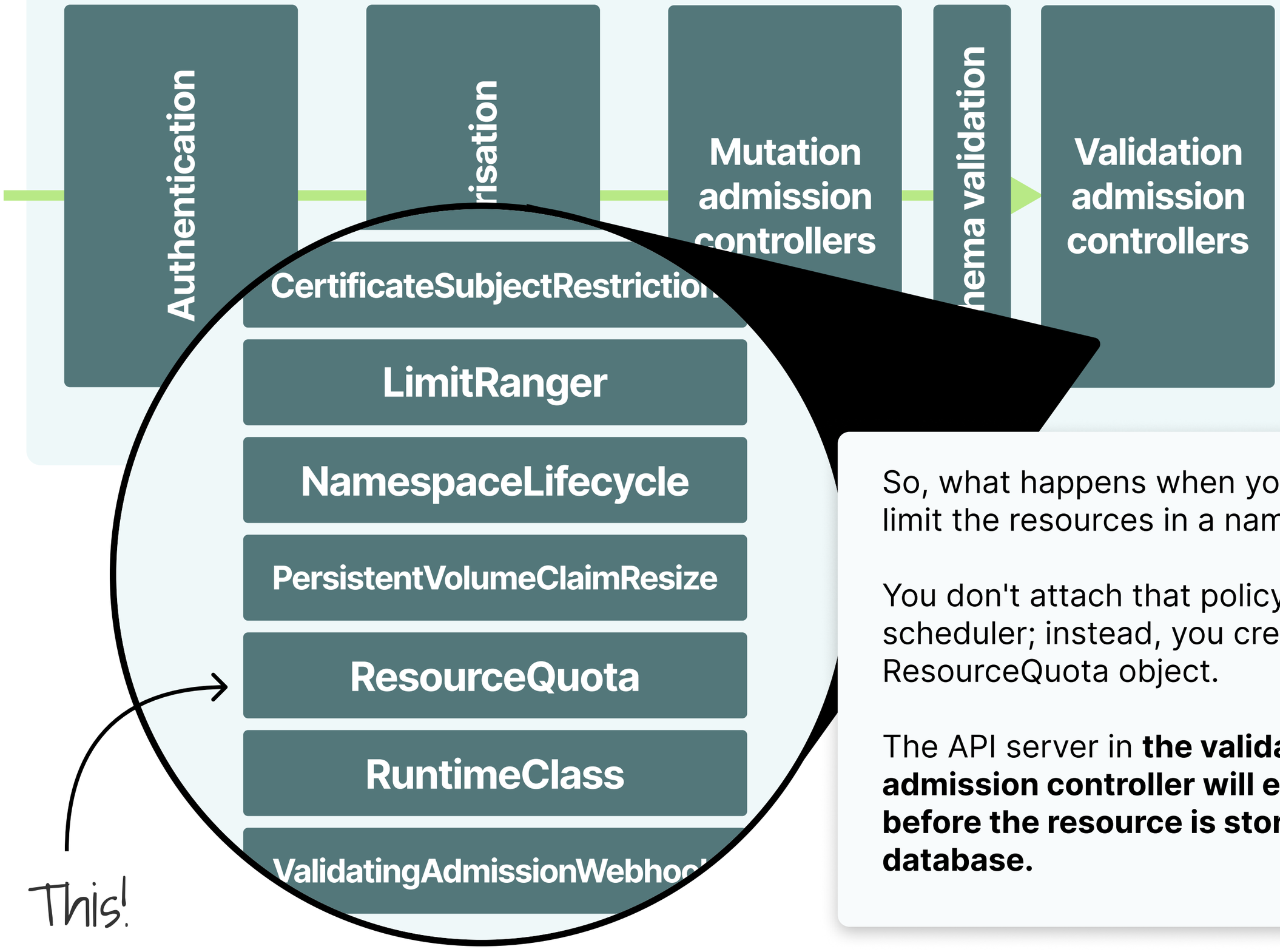
Each controller has several checks or mutating actions designed to modify pods, namespaces, ingress, storage classes and more.

etcd

learnk8s

API SERVER

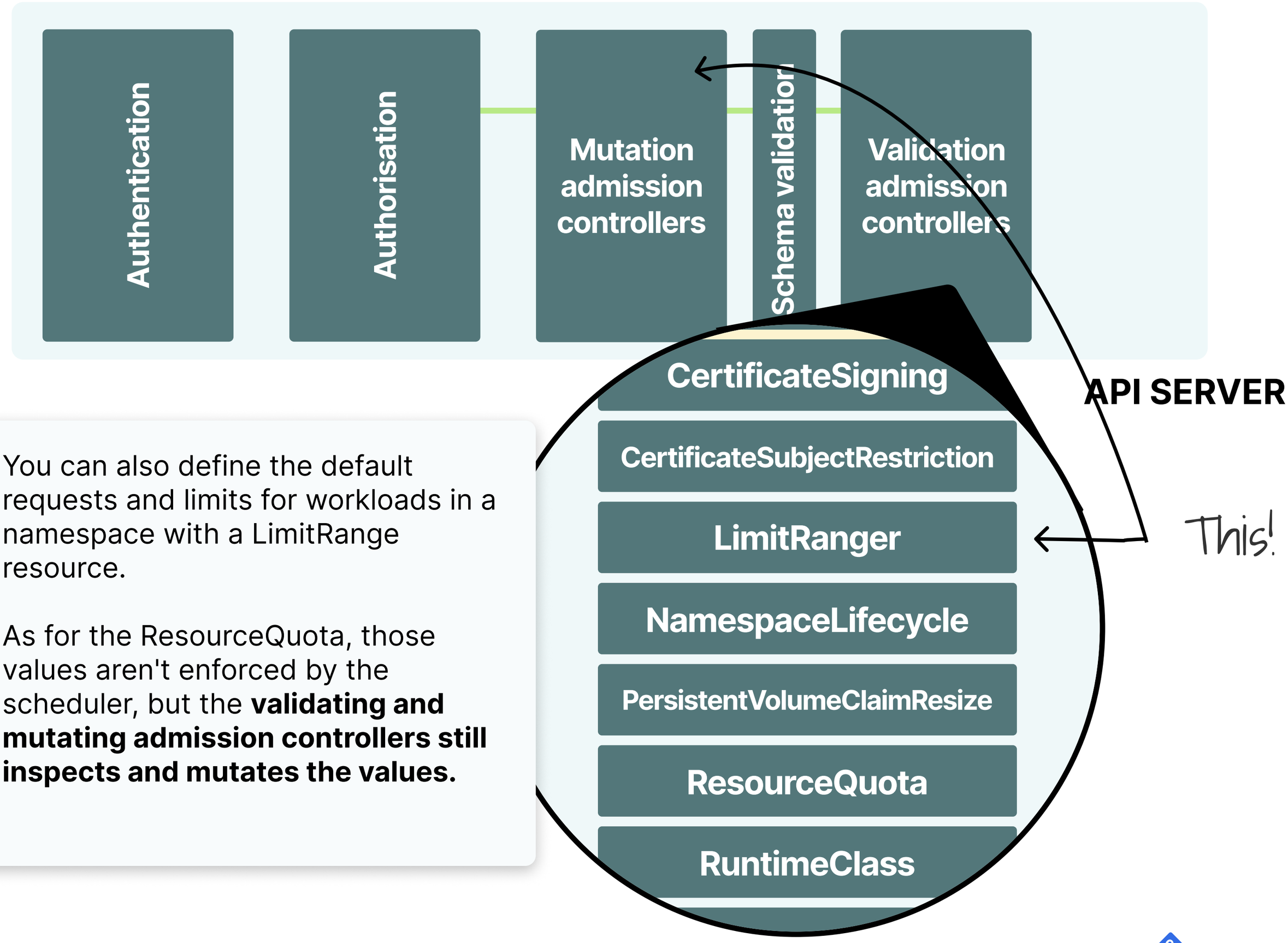
learnk8s



So, what happens when you want to limit the resources in a namespace?

You don't attach that policy to the scheduler; instead, you create a ResourceQuota object.

The API server in **the validating admission controller will enforce it before the resource is stored in the database.**



You can also define the default requests and limits for workloads in a namespace with a LimitRange resource.

As for the ResourceQuota, those values aren't enforced by the scheduler, but the **validating and mutating admission controllers still inspects and mutates the values.**

NETWORK POLICIES

default
NAMESPACE

Pod 1

Pod 3

Pod 2

What about networking?

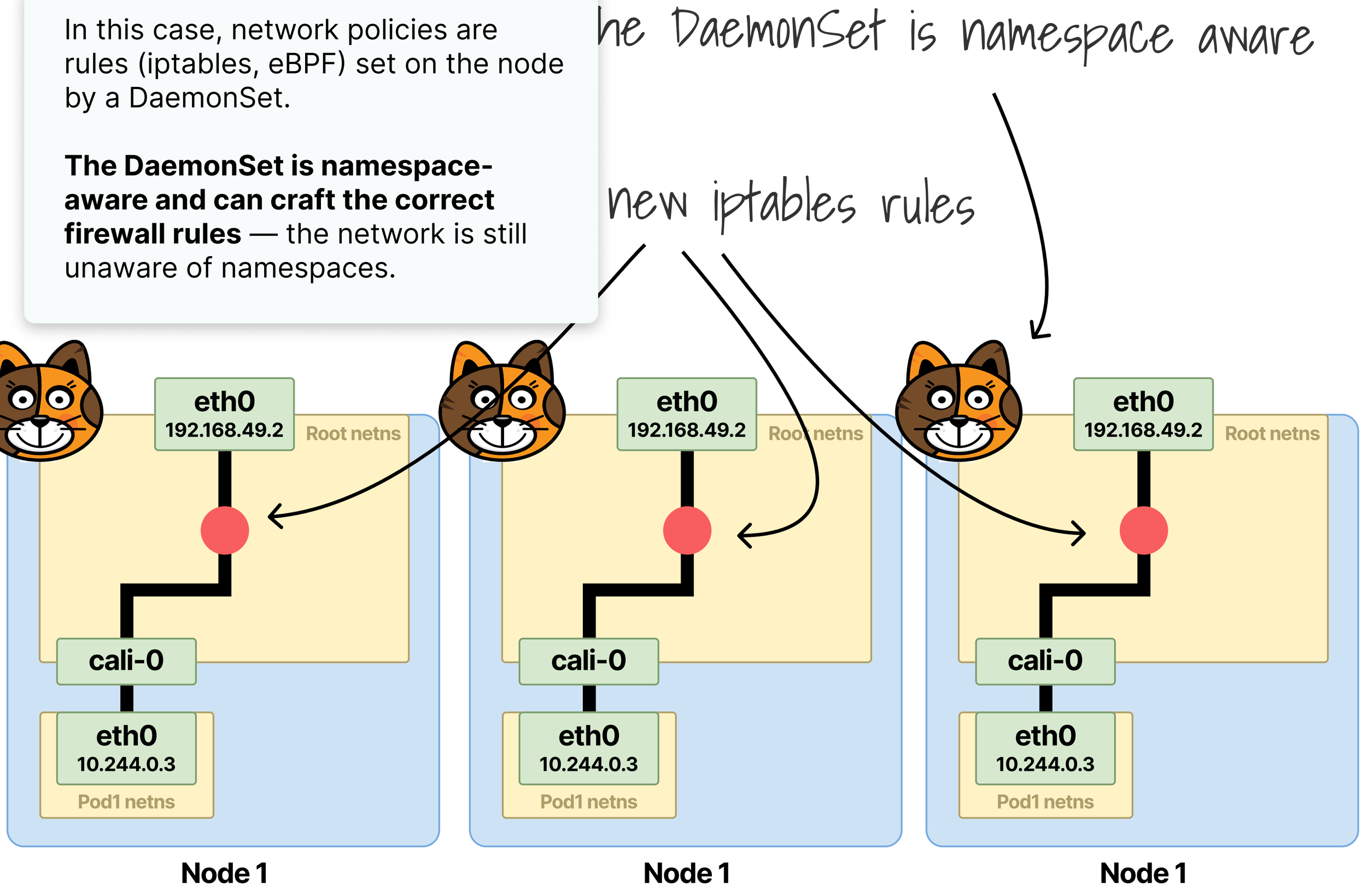
If it's true that any pod can talk to any pod, how do namespaced Network Policies work?

nginx-ingress
NAMESPACE

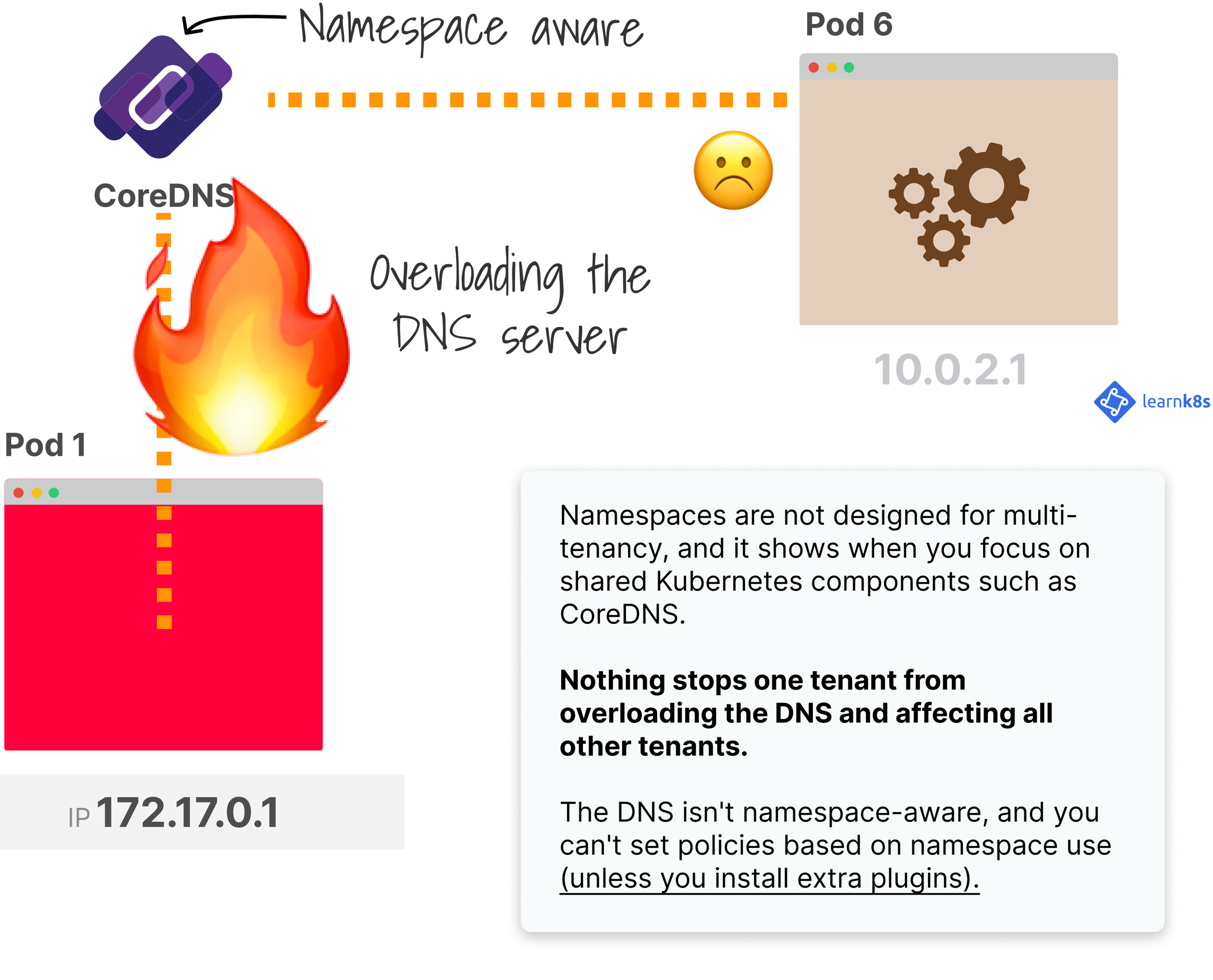
learnk8s

In this case, network policies are rules (iptables, eBPF) set on the node by a DaemonSet.

The DaemonSet is namespace-aware and can craft the correct firewall rules — the network is still unaware of namespaces.



learnk8s



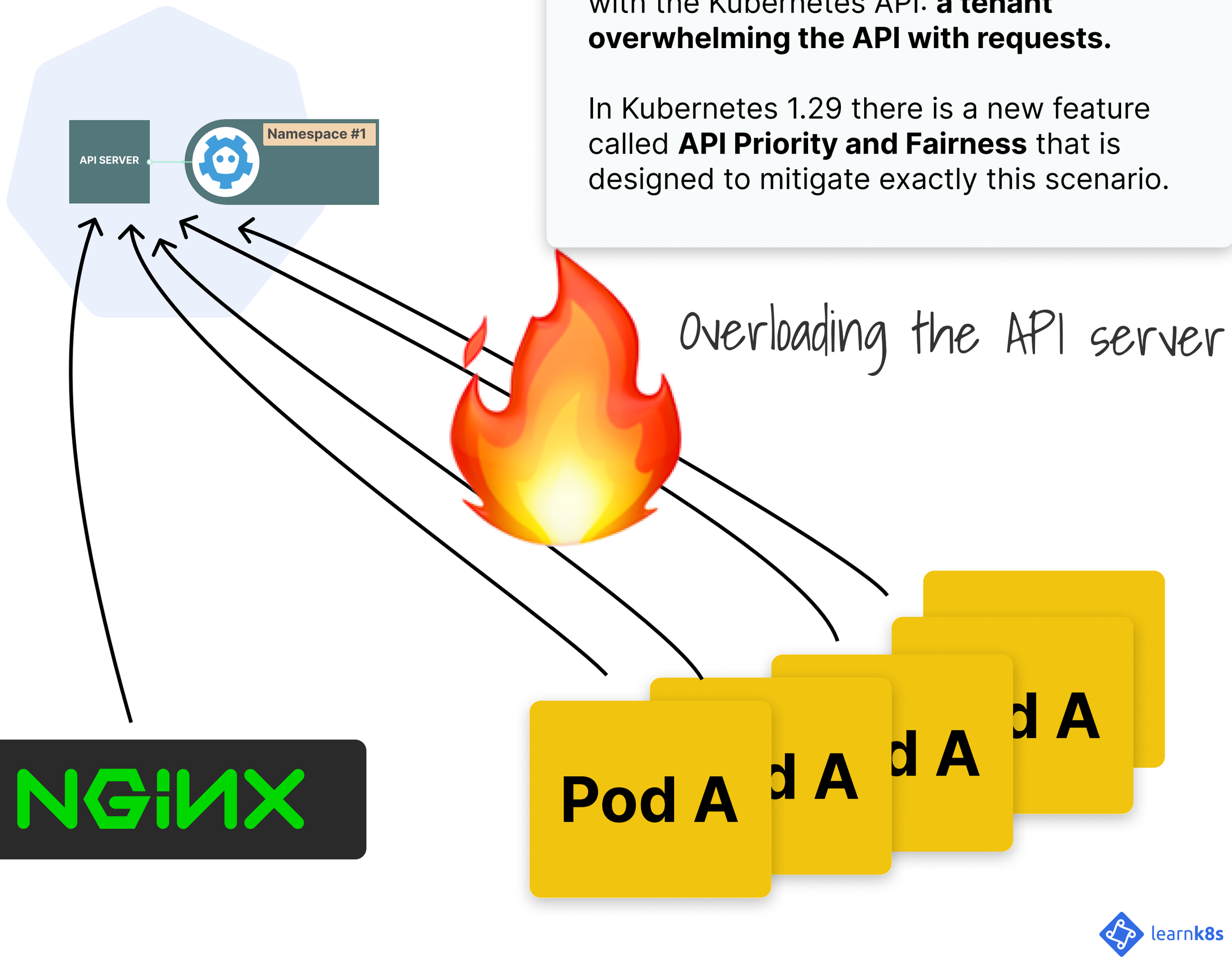
Namespaces are not designed for multi-tenancy, and it shows when you focus on shared Kubernetes components such as CoreDNS.

Nothing stops one tenant from overloading the DNS and affecting all other tenants.

The DNS isn't namespace-aware, and you can't set policies based on namespace use (unless you install extra plugins).

You might experience the same scenario with the Kubernetes API: **a tenant overwhelming the API with requests.**

In Kubernetes 1.29 there is a new feature called **API Priority and Fairness** that is designed to mitigate exactly this scenario.



1 Soft multi-tenancy

Hierarchical Namespace controller, Capsule

2 Control plane isolation

vCluster, Kamaji*, Hypershift*

3 Hard multi-tenancy

Karmada

Namespaces are a great building block for building higher abstractions in Kubernetes.

However, they are **not meant to be used as a multi-tenancy solution by themselves.**

The community has developed several tools to build more robust abstractions for managing tenants to fill this gap.

Join Salman's session on building Kubernetes platforms this Thursday (7th Mar) at 8am PT / 5pm CET!