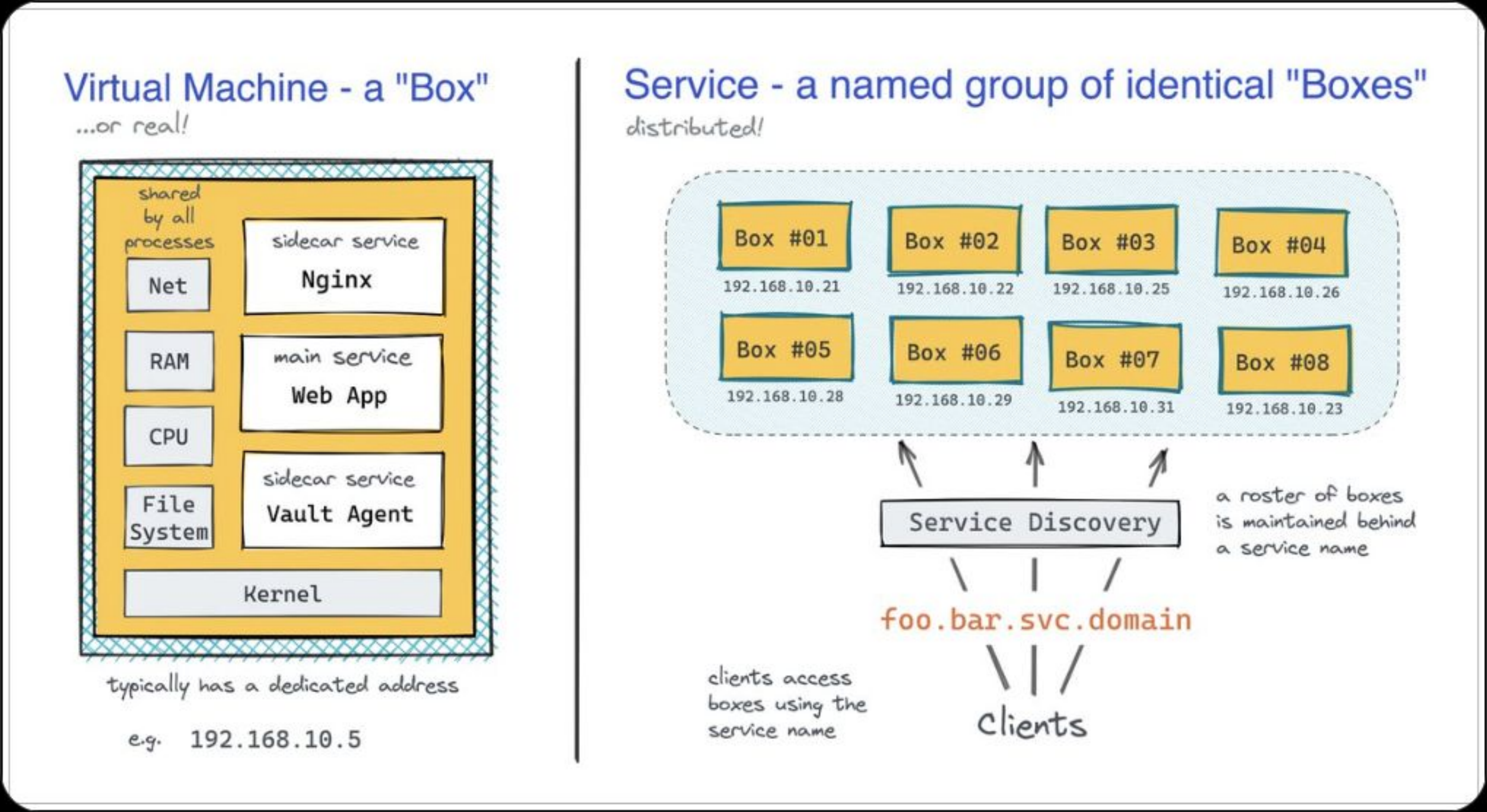
Kubernetes basics explained by analogy 🧵

...or "How Kubernetes Just Repeats Good Old Deployment Patterns"

1. For a long time, people had been deploying services as groups of virtual (or physical) machines.

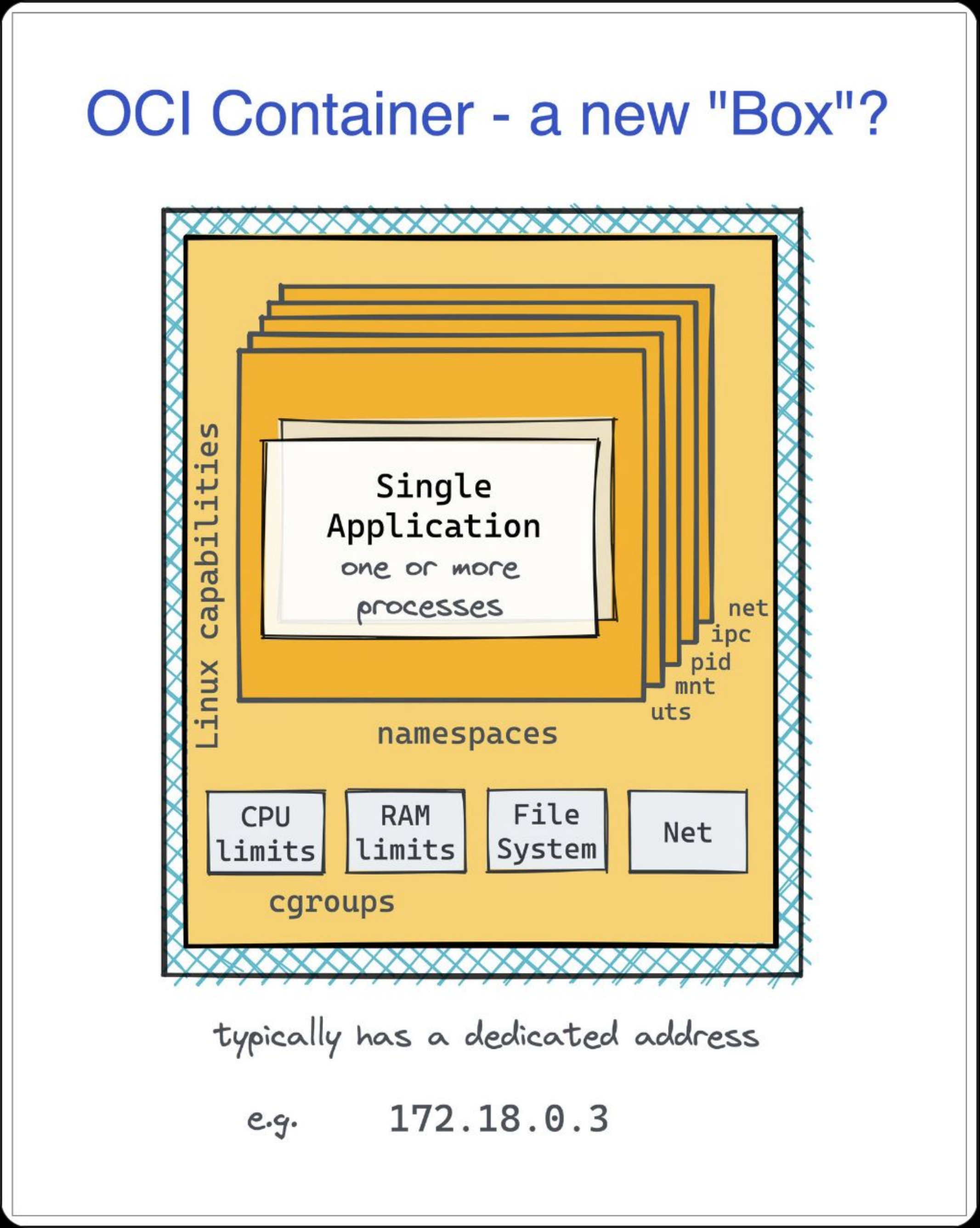
But VMs were often slow and bulky. Hence, not very efficient.



2. Then containers gained quite some popularity.

With containers, it became easier to distribute services. Reproducibility also improved. But containers haven't become a replacement for VMs.

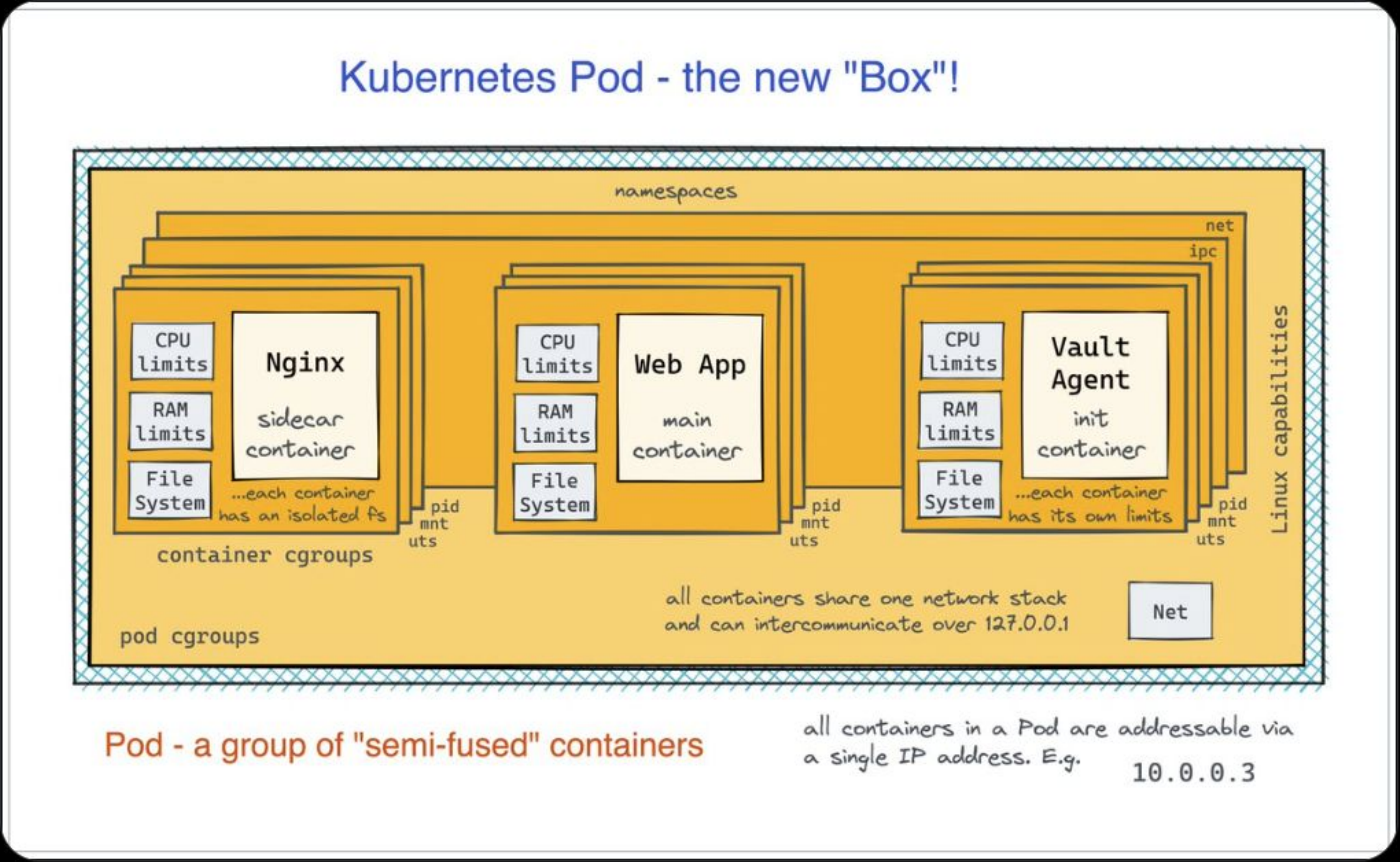
Mainly, because of their deliberate focus on being an environment to run a single app:



3. Instead of containers, another abstraction took off - Kubernetes Pods!

A Pod is a group of semi-fused containers. External borders were preserved, but some of the internal isolation b/w containers substituting a Pod got weakened.

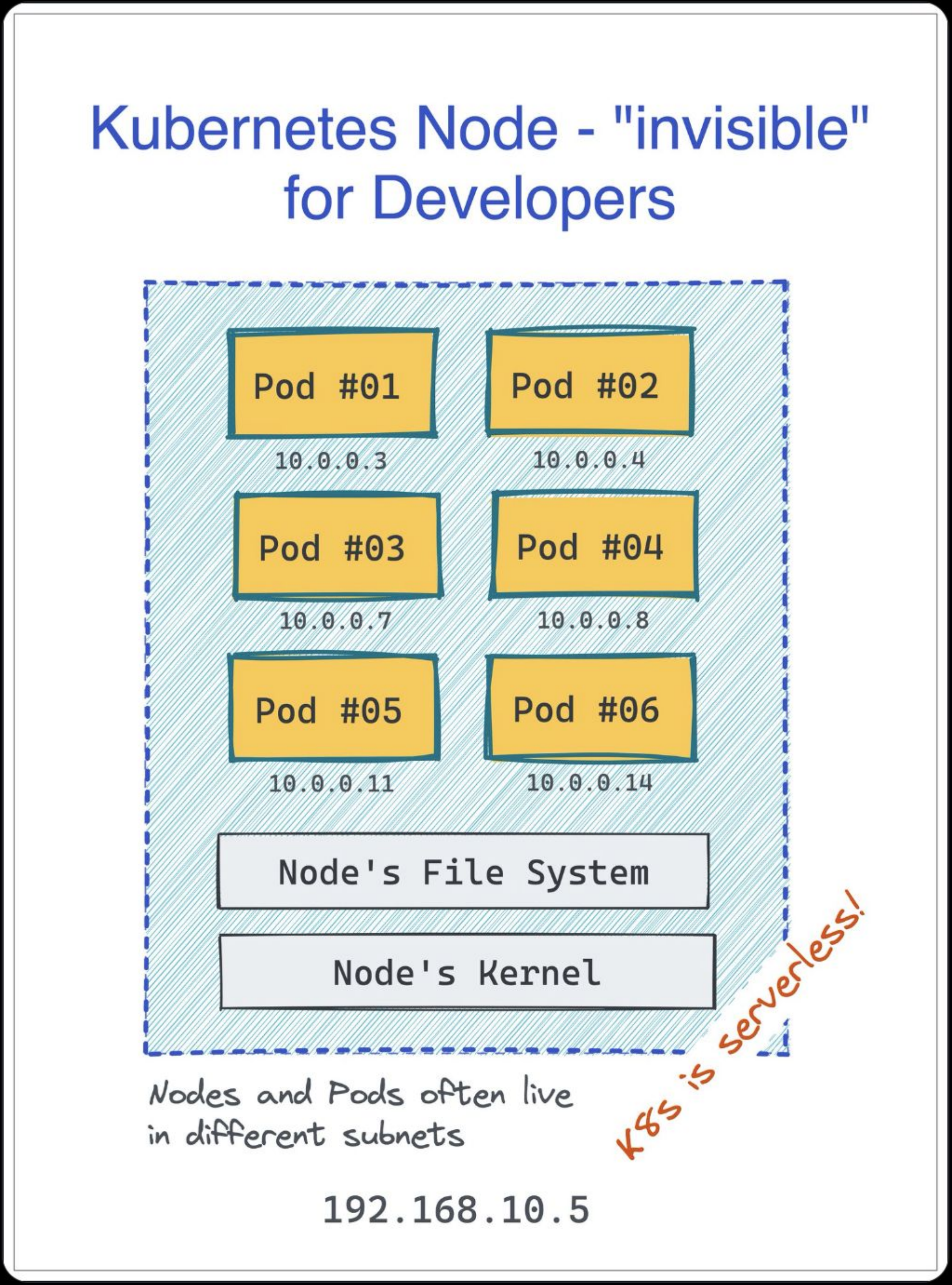
A Pod is a much closer abstraction to a VM.


Kubernetes Pod - the new "Box"!
namespaces
net
ipc
CPU limits
RAM limits
File System
Nginx
sidecar container
...each container has an isolated fs
pid mnt uts
container cgroups
Web App
main container
Vault Agent
init container
...each container has its own limits
Linux capabilities
all containers share one network stack and can intercommunicate over 127.0.0.1
Net
pod cgroups
Pod - a group of "semi-fused" containers
all containers in a Pod are addressable via a single IP address. E.g. 10.0.0.3

4. A single (virtual or physical) machine can run many independant Pods.

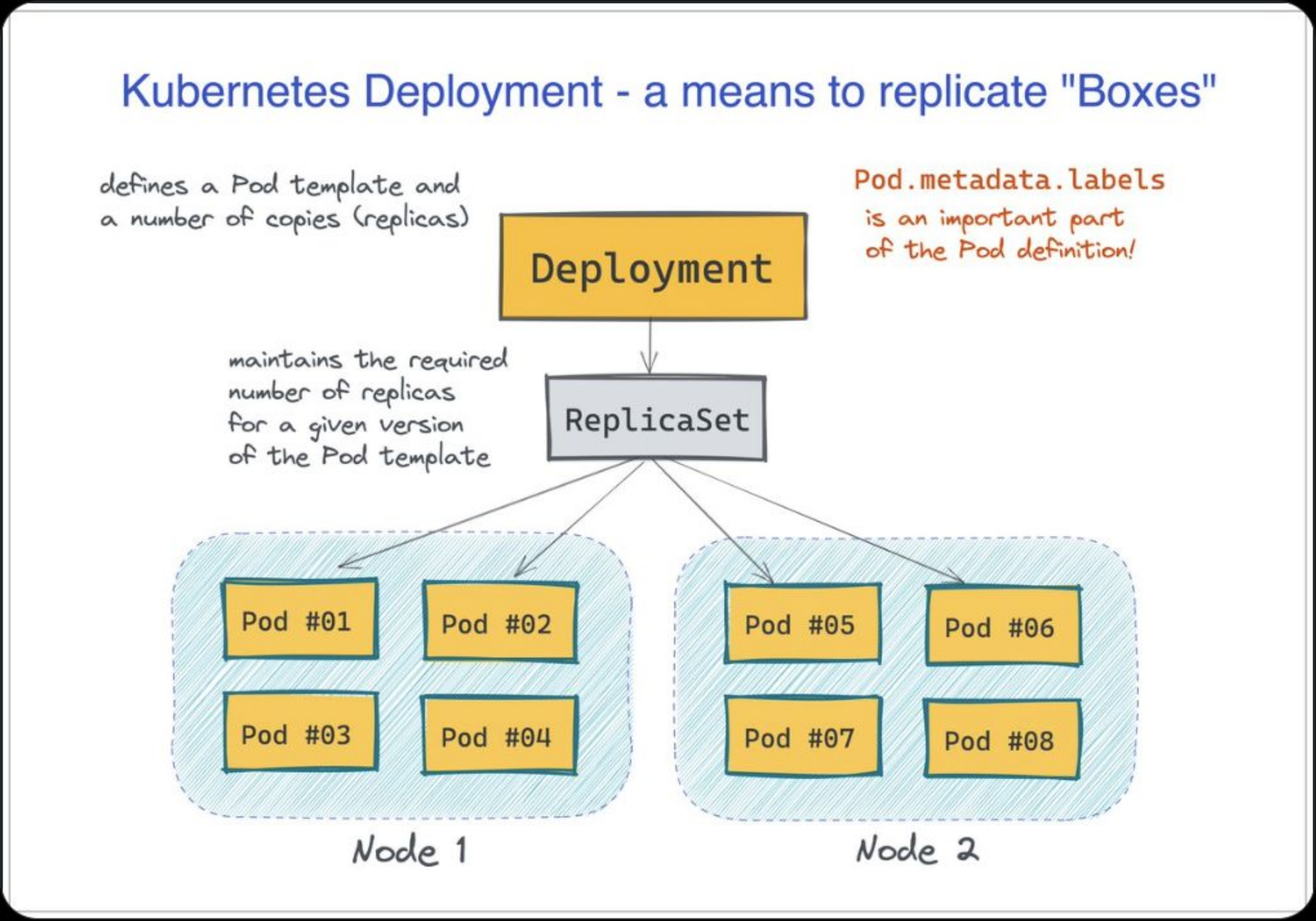
In Kubernetes, machines substituting a cluster are called Nodes, but developers are rarely concerned with this abstraction. For them, Kubernetes is serverless!

More Pods per server means better packing.



5. Deployment of Pods happens through replicating a Pod template.

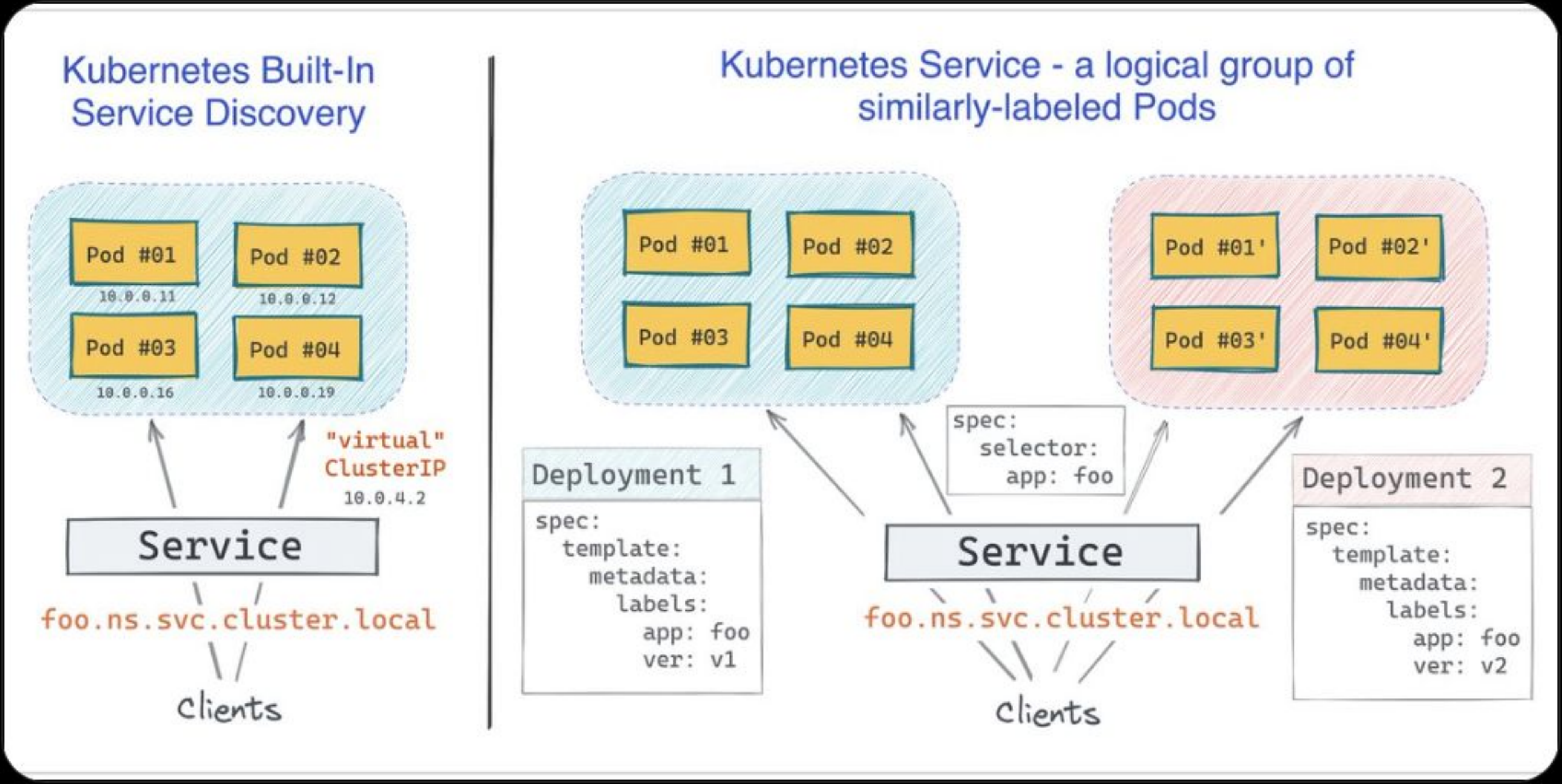
There is a Deployment object in Kubernetes that holds the desired Pod template and the needed number of "copies." But logically, there is not much difference between scaling Pods and VMs.


Kubernetes Deployment - a means to replicate "Boxes"
defines a Pod template and a number of copies (replicas)
Pod.metadata.labels is an important part of the Pod definition!
Deployment
maintains the required number of replicas for a given version of the Pod template
ReplicaSet
Pod #01
Pod #02
Pod #03
Pod #04
Pod #05
Pod #06
Pod #07
Pod #08
Node 1
Node 2

6. Kubernetes Service is a means of grouping Pods behind a logical name.

Kubernetes comes with built-in service discovery.

The implementation is neither client- nor server-side (rather network-side). But from the clients' standpoint, it feels like a good old reverse proxy.



```
spec:
  selector:
    app: foo
```

Deployment 1

```
spec:
  template:
    metadata:
      labels:
        app: foo
        ver: v1
```

Deployment 2

```
spec:
  template:
    metadata:
      labels:
        app: foo
        ver: v2
```