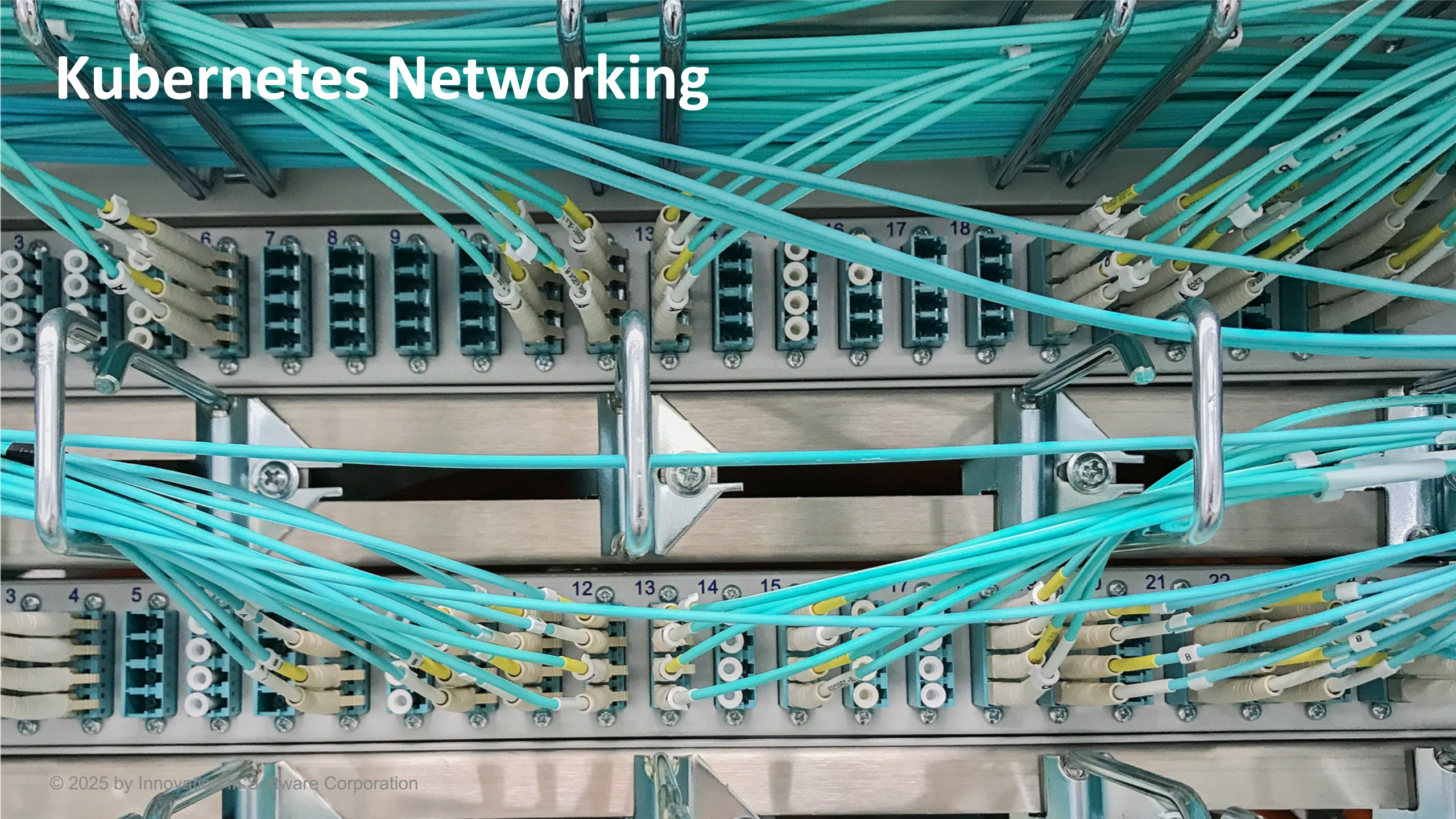
Running and Securing Kubernetes Apps

TEKsystems Global Services

WORKFORCE DEVELOPMENT

Networking

Kubernetes Networking



Networking Problems and Solutions

1. Highly-coupled container-to-container communications => linux
2. Pod-to-Pod communications => various
3. Pod-to-Service communications => service
4. External-to-Service communications => service or ingress

Kubernetes Networking Model Requirements

- All containers can communicate with all other containers without NAT
- All nodes can communicate with all containers (and vice versa) without NAT
- The IP that a container sees itself as is the same IP that others see it as

Container to Container Communication

- Containers live inside of pods
- Group of one or more containers that are always co-located and co-scheduled, and run in a shared context
- Containers within a POD share an IP address and port space, and can find each other via localhost (net namespace).
- Communicate with each other using standard inter-process communications (ipc namespace).

Pod to Pod Requirements

- Pods can communicate with all other pods without NAT
- Regardless of which host they land on
- Every pod gets its own IP address
- No need to explicitly create links between pods

Pod Networking Approaches

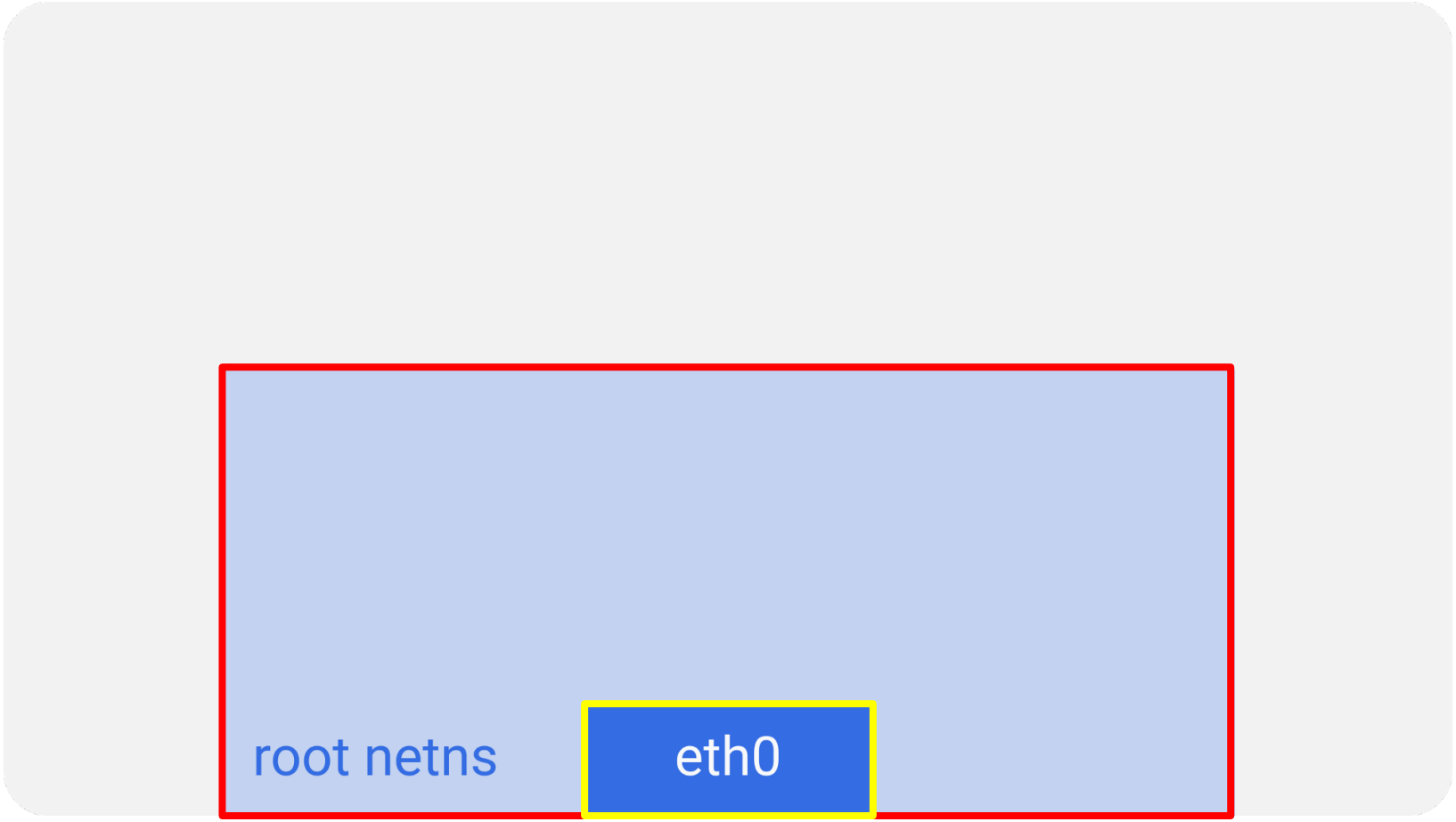
Using an overlay network

- An overlay network obscures the underlying network architecture from the pod network through traffic encapsulation (for example vxlan).
- Encapsulation reduces performance, though exactly how much depends on your solution.

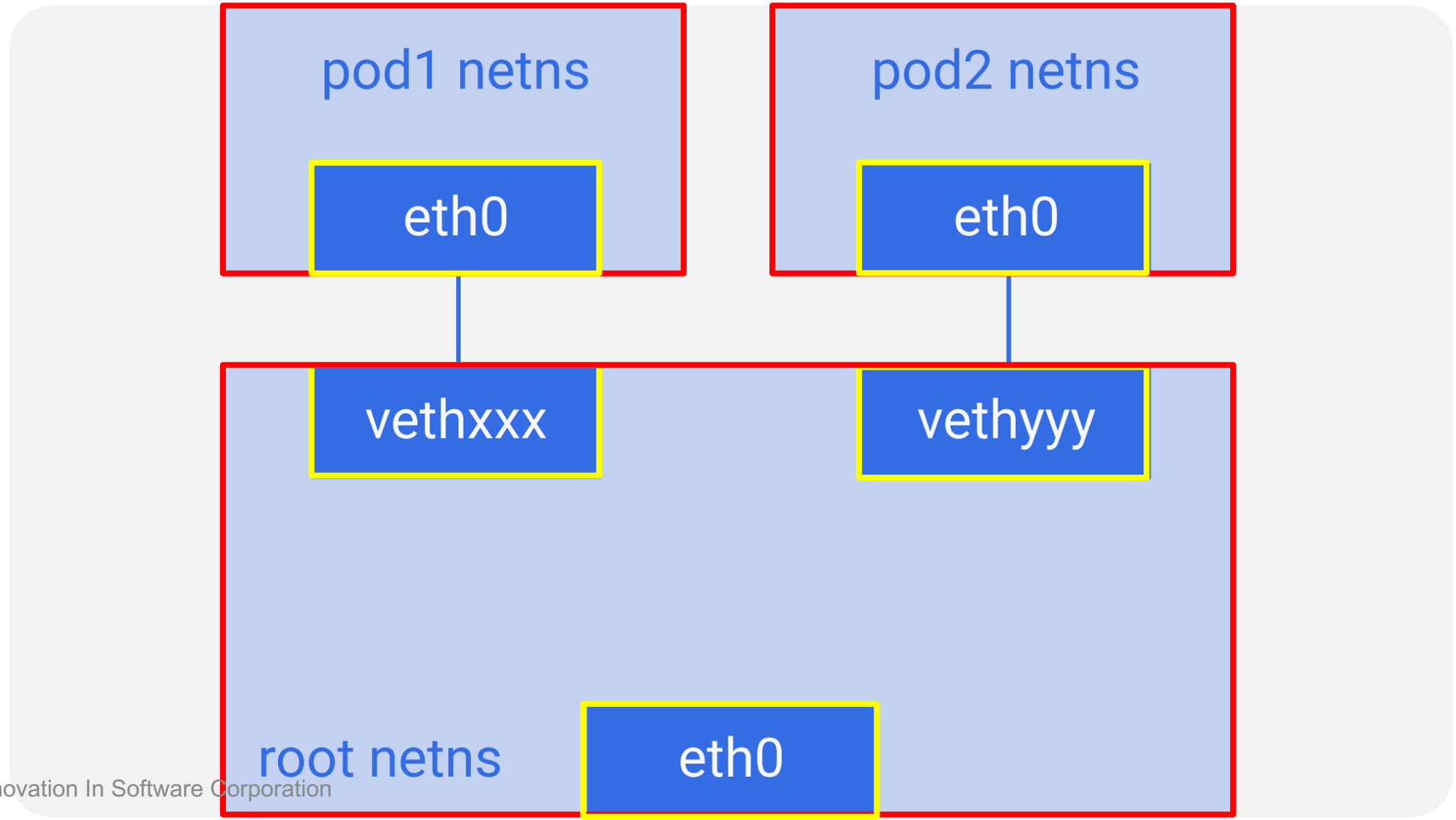
Without an overlay network

- Configure the underlying network fabric (switches, routers, etc.) to be aware of pod IP addresses.
- This does not require the encapsulation provided by an overlay and so can achieve better performance.

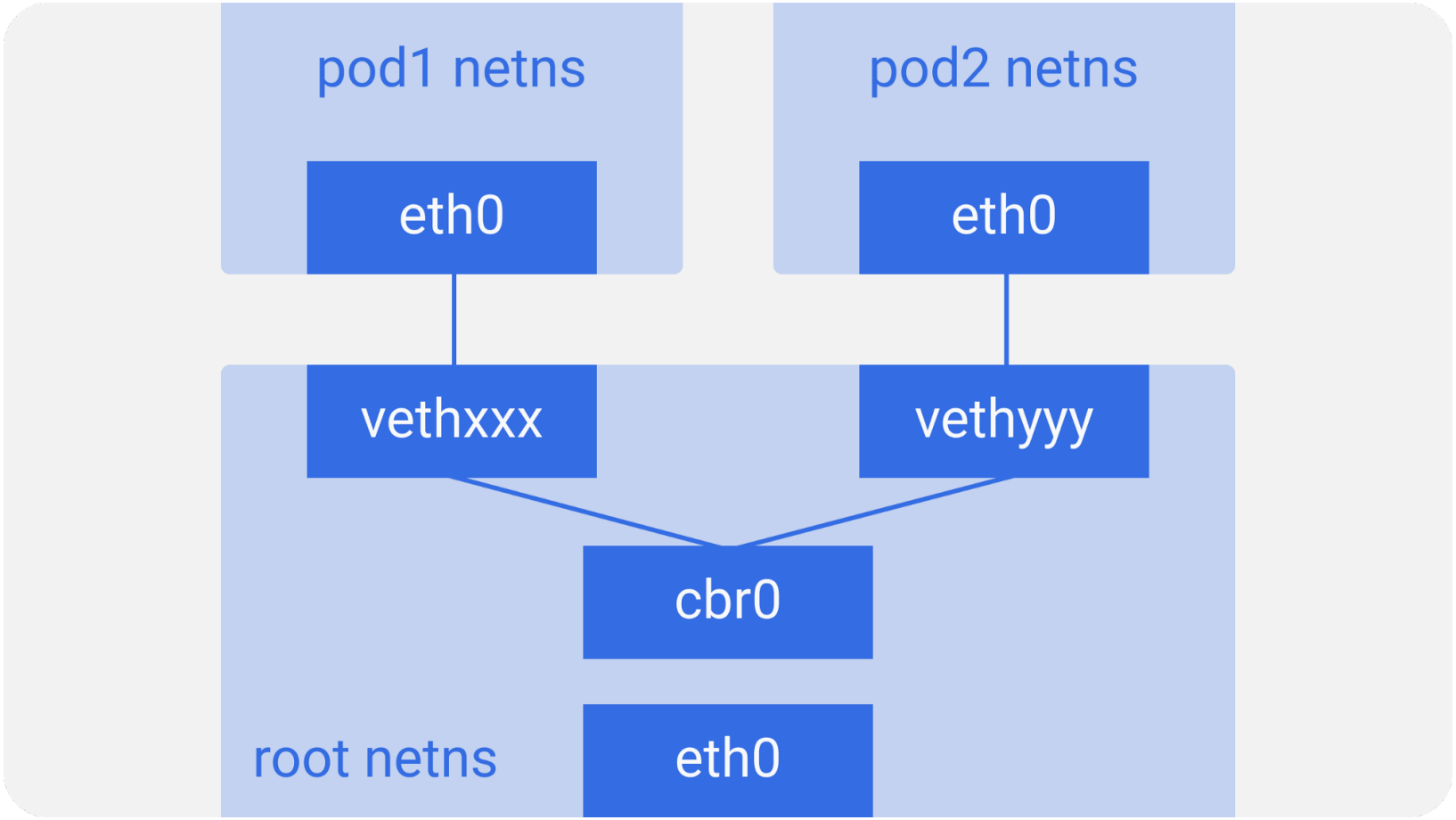
Node Namespace



Node Namespaces

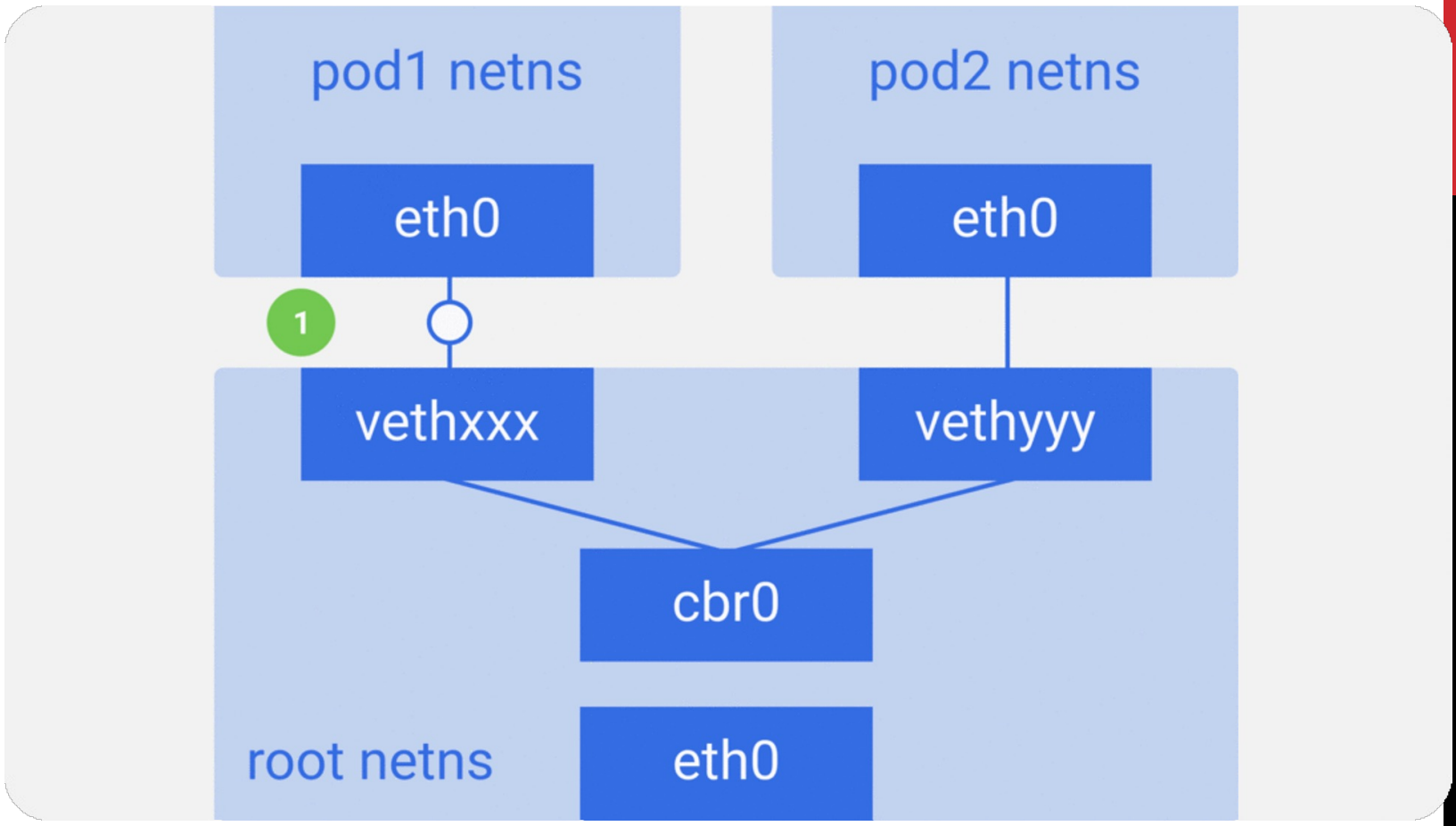


Pod to Host



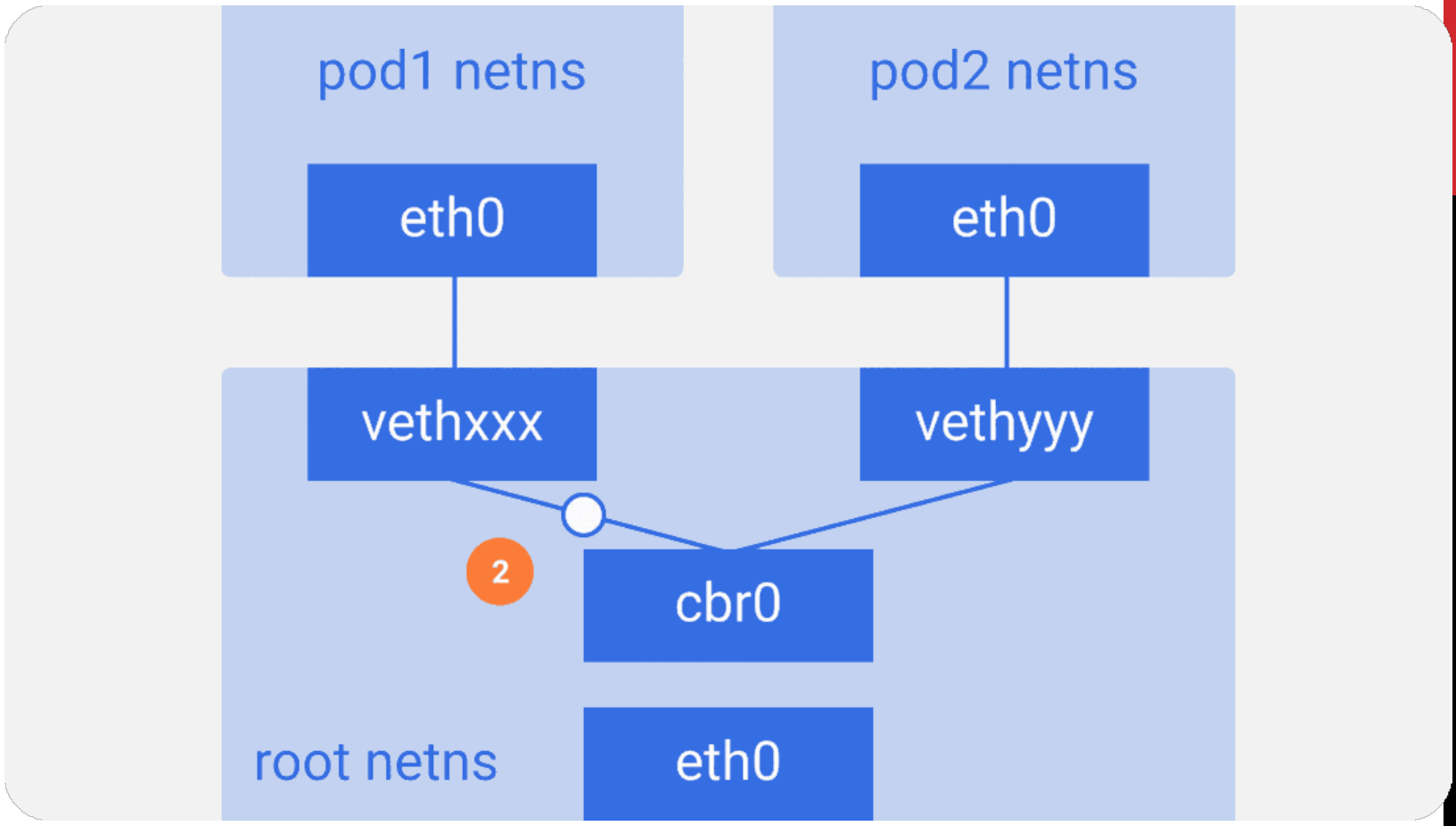
Single Host POD to POD Network

1. Packet leaves pod1's netns at eth0 and enters the root netns at vethxxx.



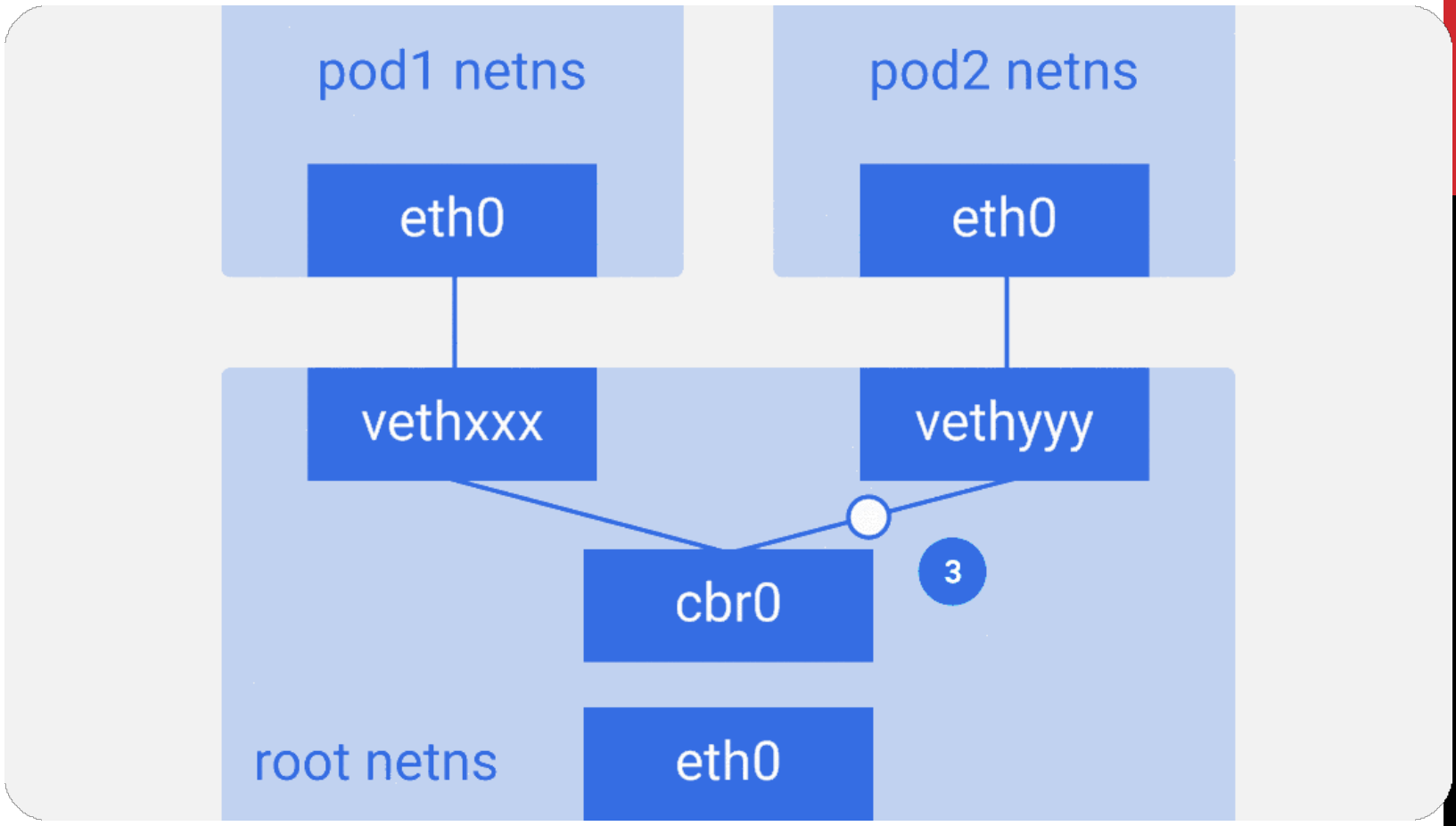
Single Host POD to POD Networking

2. It's passed on to cbr0, which discovers the destination using an ARP request, saying "who has this IP?"



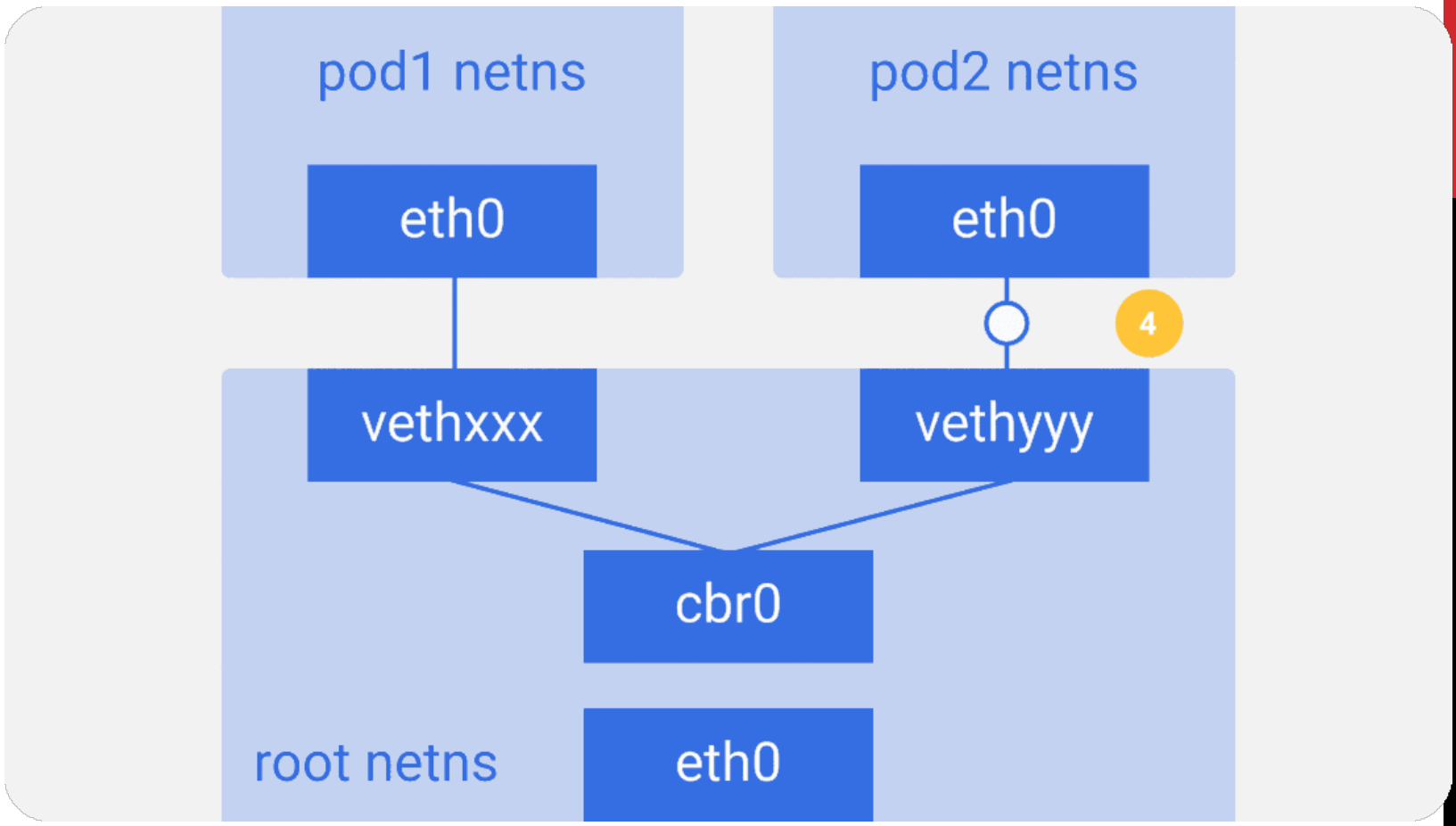
Single Host POD to POD Networking

3. vethyyy says it has that IP, so the bridge knows where to forward the packet.



Single Host POD to POD Networking

4. The packet reaches vethyyy, crosses the pipe-pair and reaches pod2's netns.

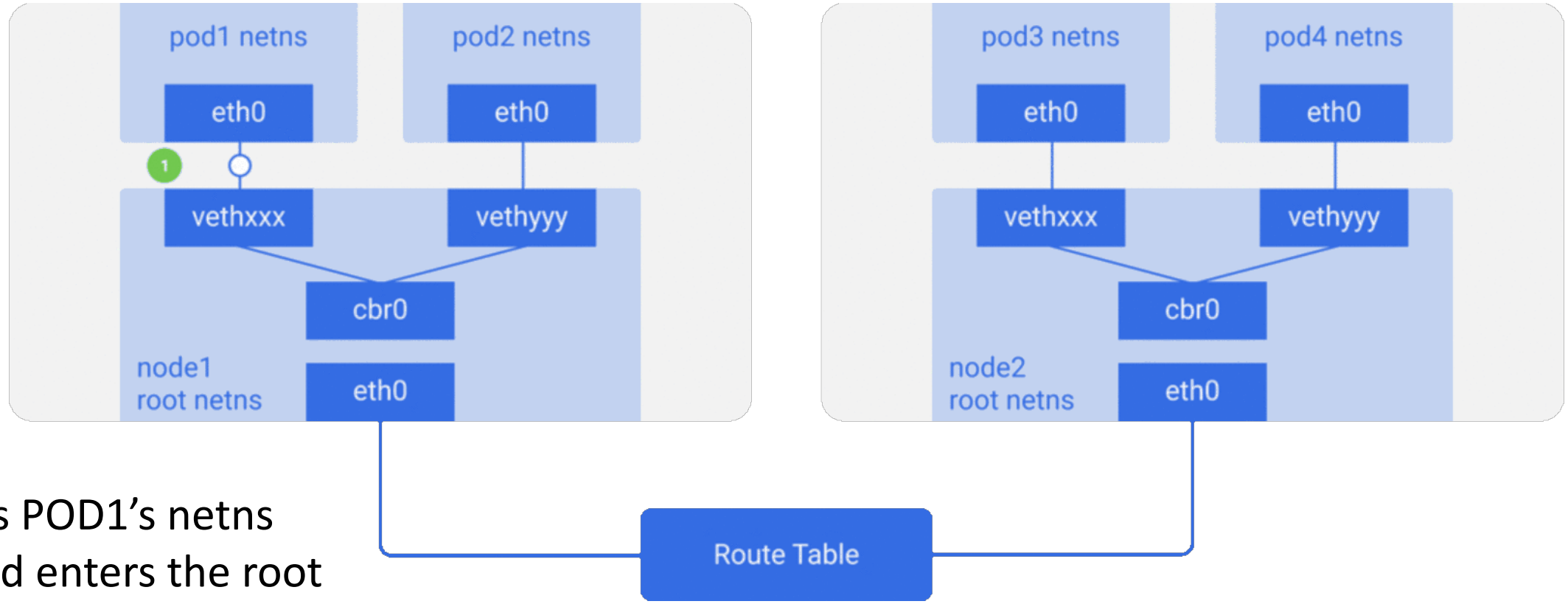


Kubernetes Inter-node Communication

Inter-node Communication

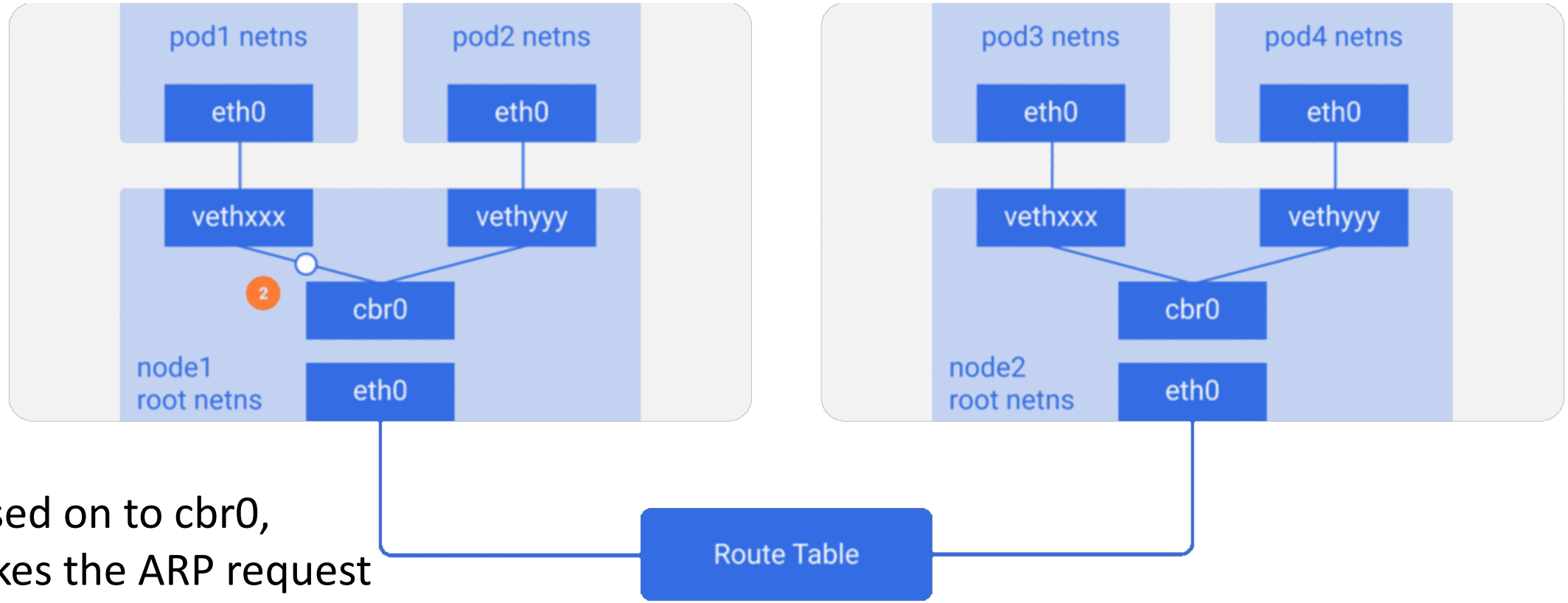
- PODs must be reachable across nodes
- Kubernetes doesn’t care how:
  - L2 (ARP across nodes)
  - L3 (IP routing across nodes)
  - Overlay networks

Inter-node Communication



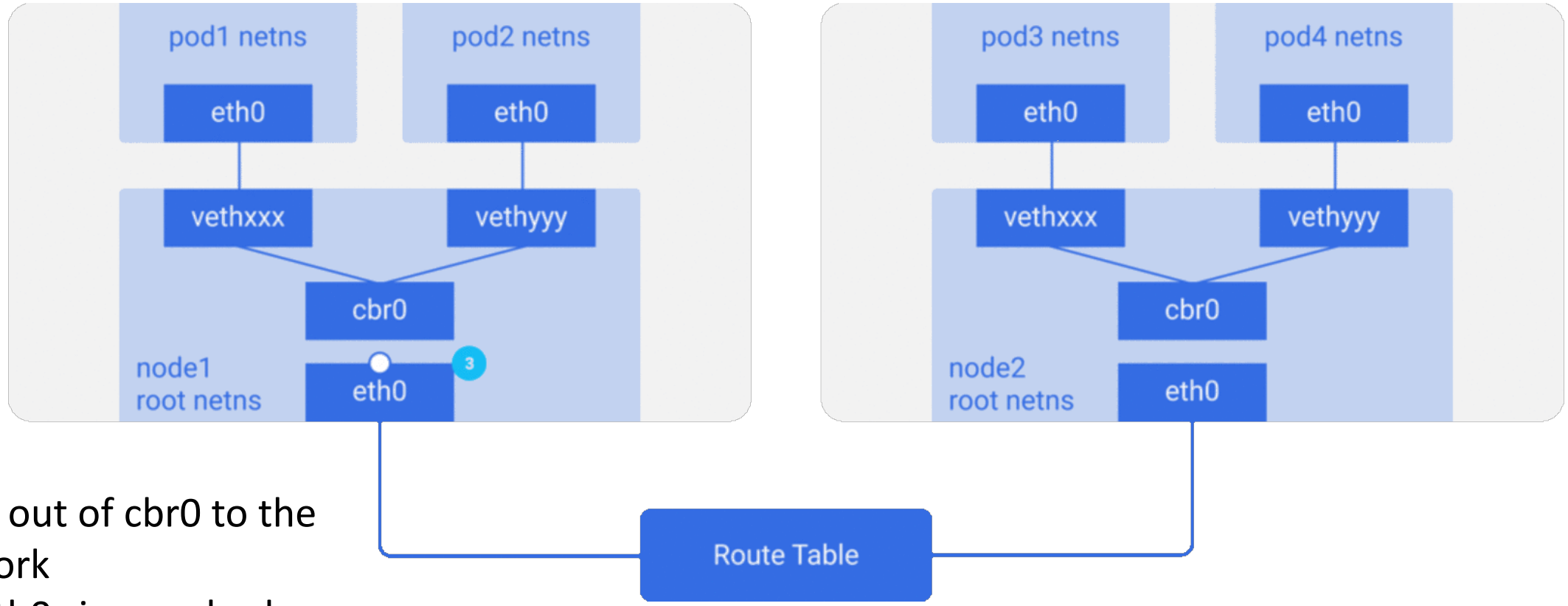
1. It leaves POD1's netns at eth0 and enters the root netns at vethxxx.

Inter-node Communication



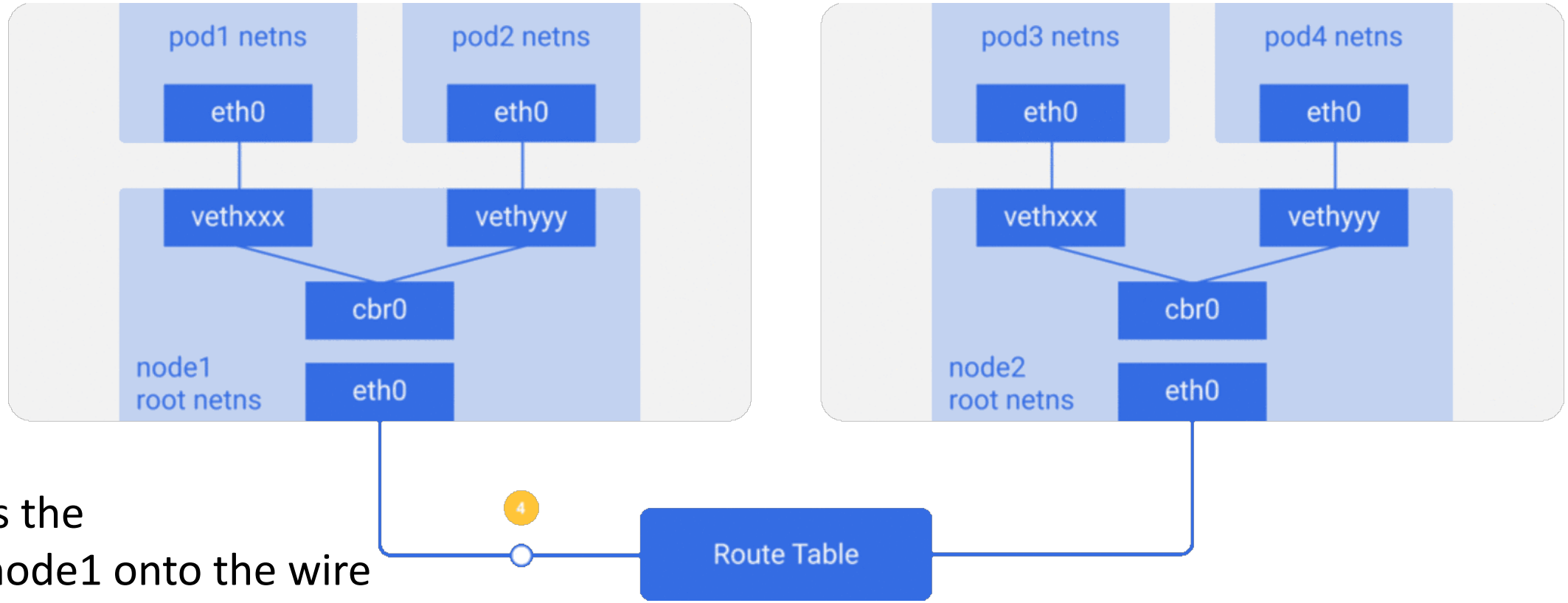
2. It's passed on to cbr0, which makes the ARP request to find the destination.

Inter-node Communication



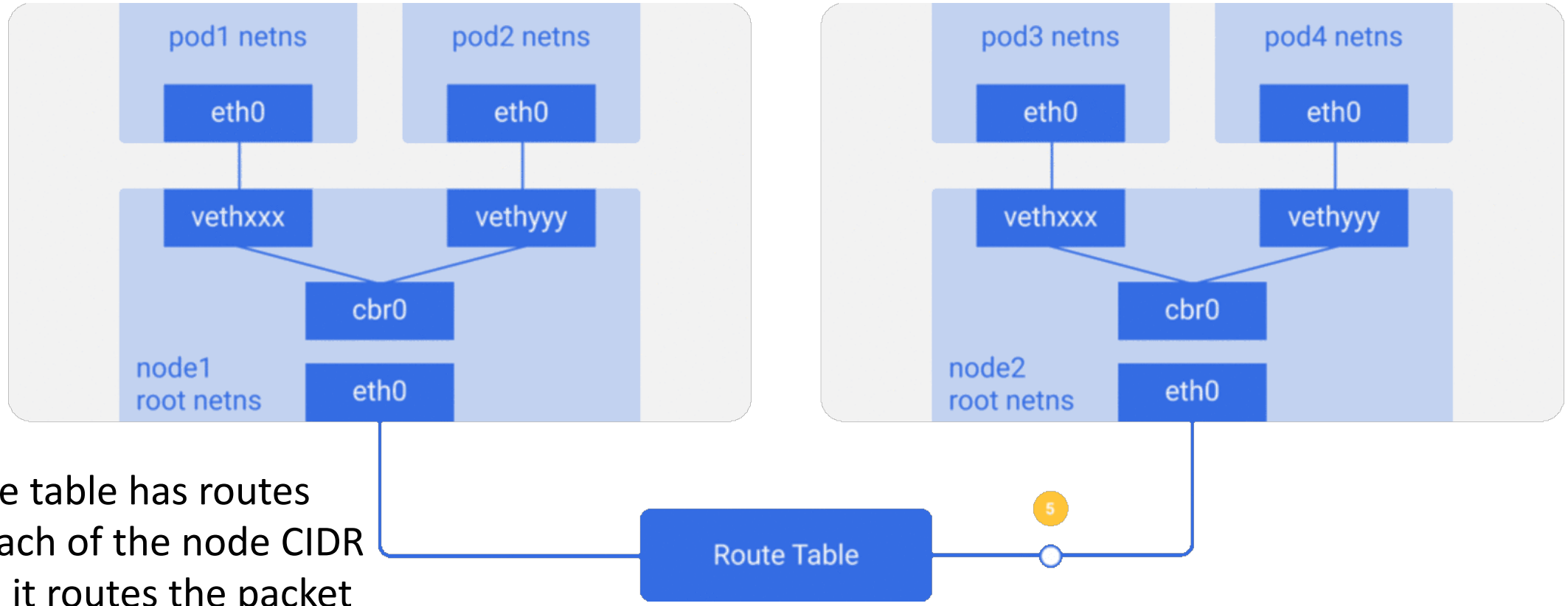
3. It comes out of cbr0 to the main network interface eth0 since nobody on this node has the IP address for POD4.

Inter-node Communication



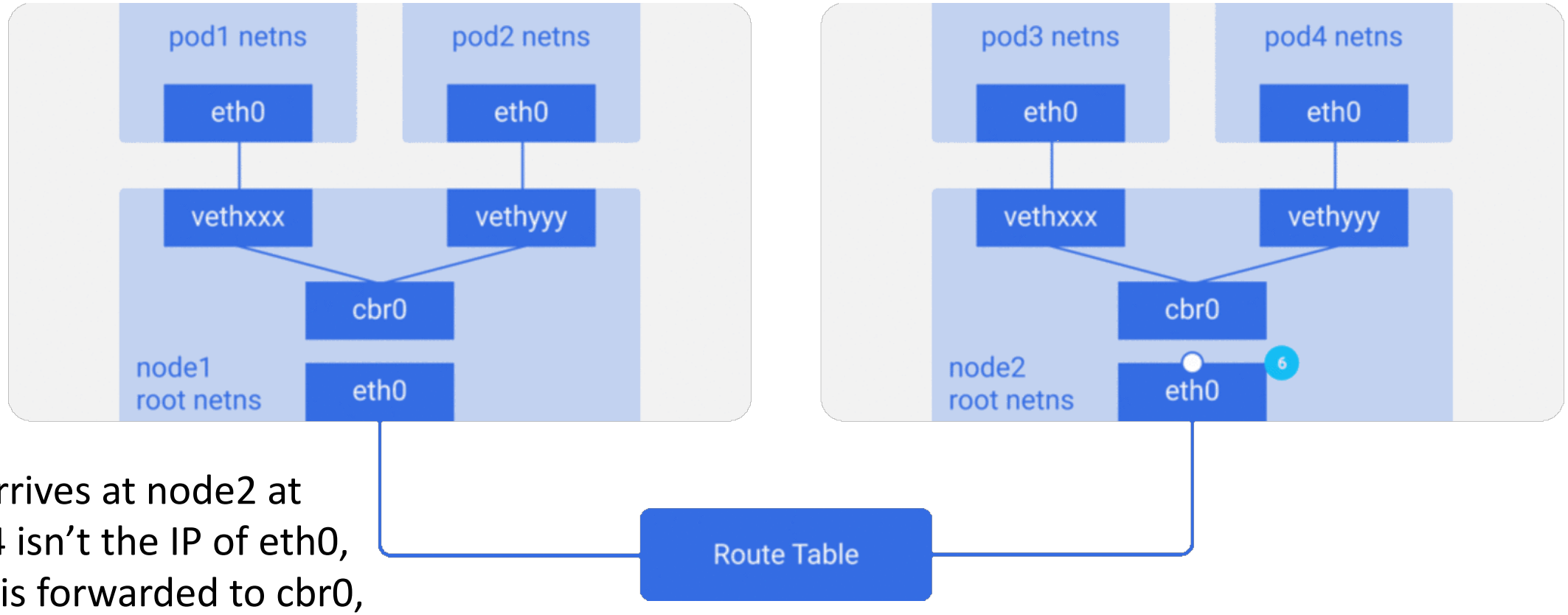
4. It leaves the machine node1 onto the wire with src=POD1 and dst=POD4.

Inter-node Communication



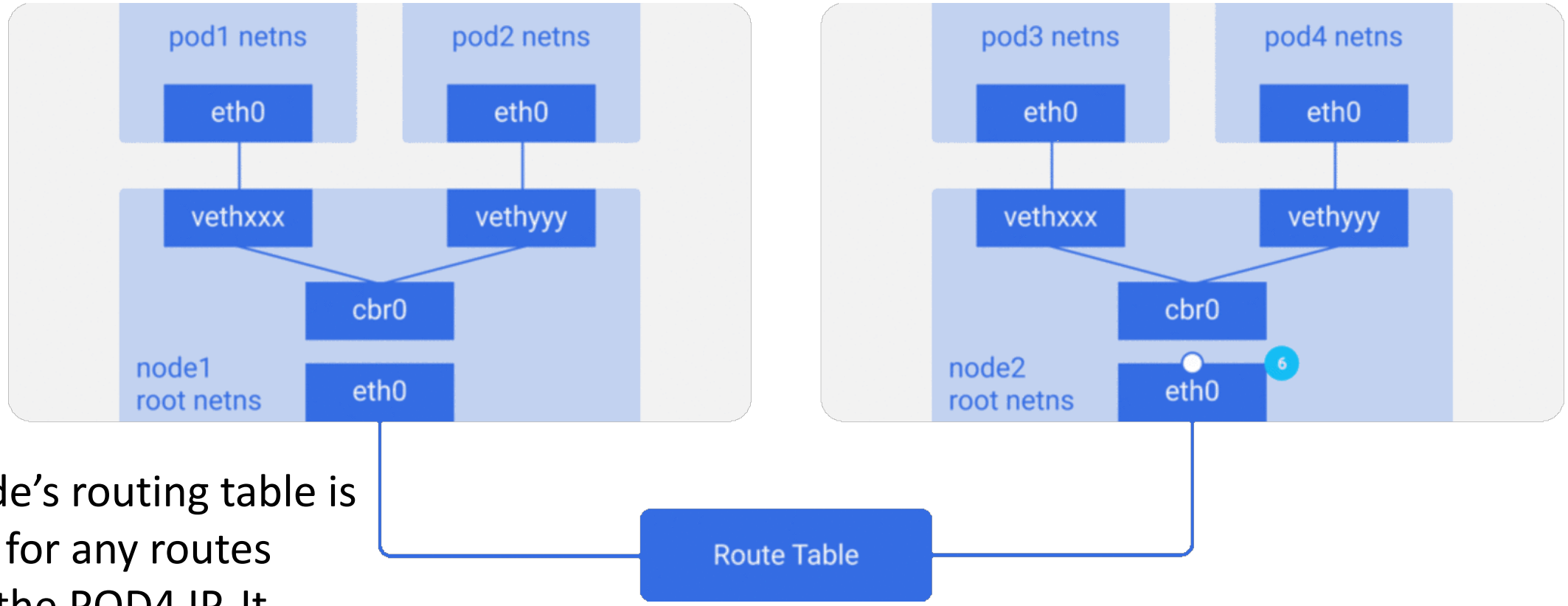
5. The route table has routes setup for each of the node CIDR blocks, and it routes the packet to the node whose CIDR block contains the POD4 IP.

Inter-node Communication



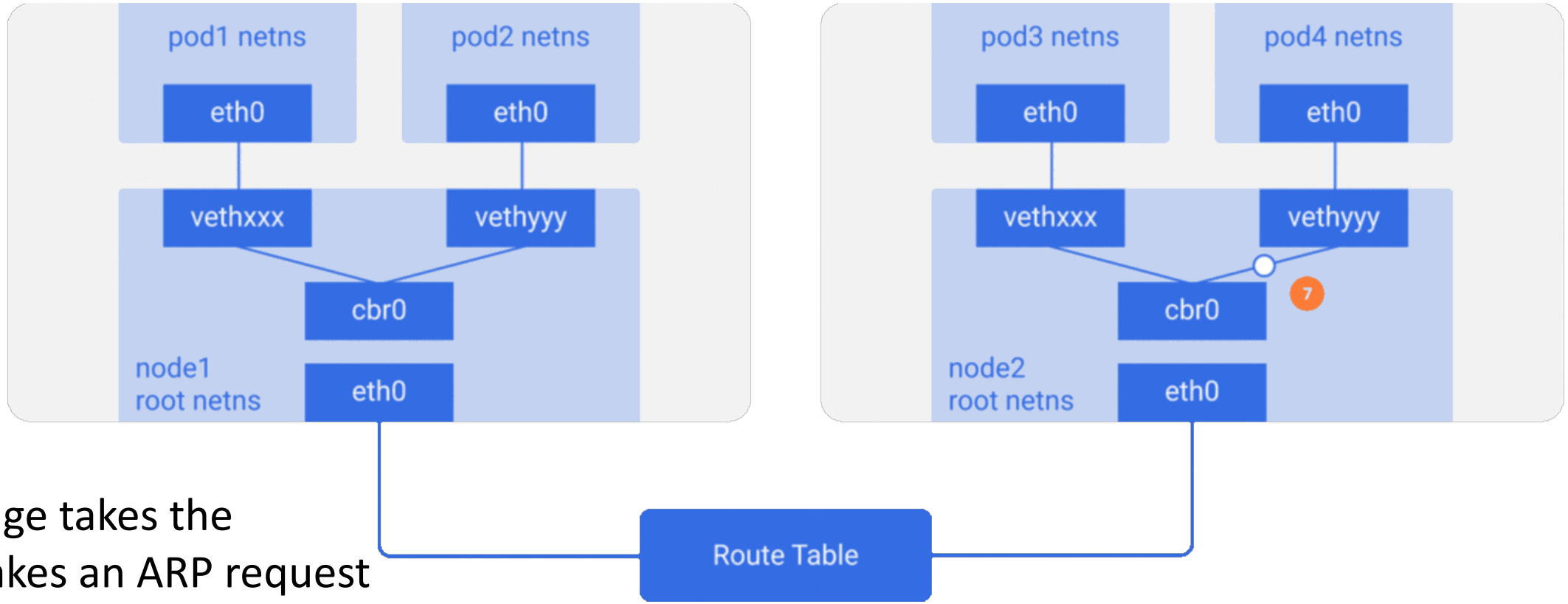
6. Packet arrives at node2 at eth0. POD4 isn't the IP of eth0, the packet is forwarded to cbr0, nodes are configured with IP forwarding enabled.

Inter-node Communication



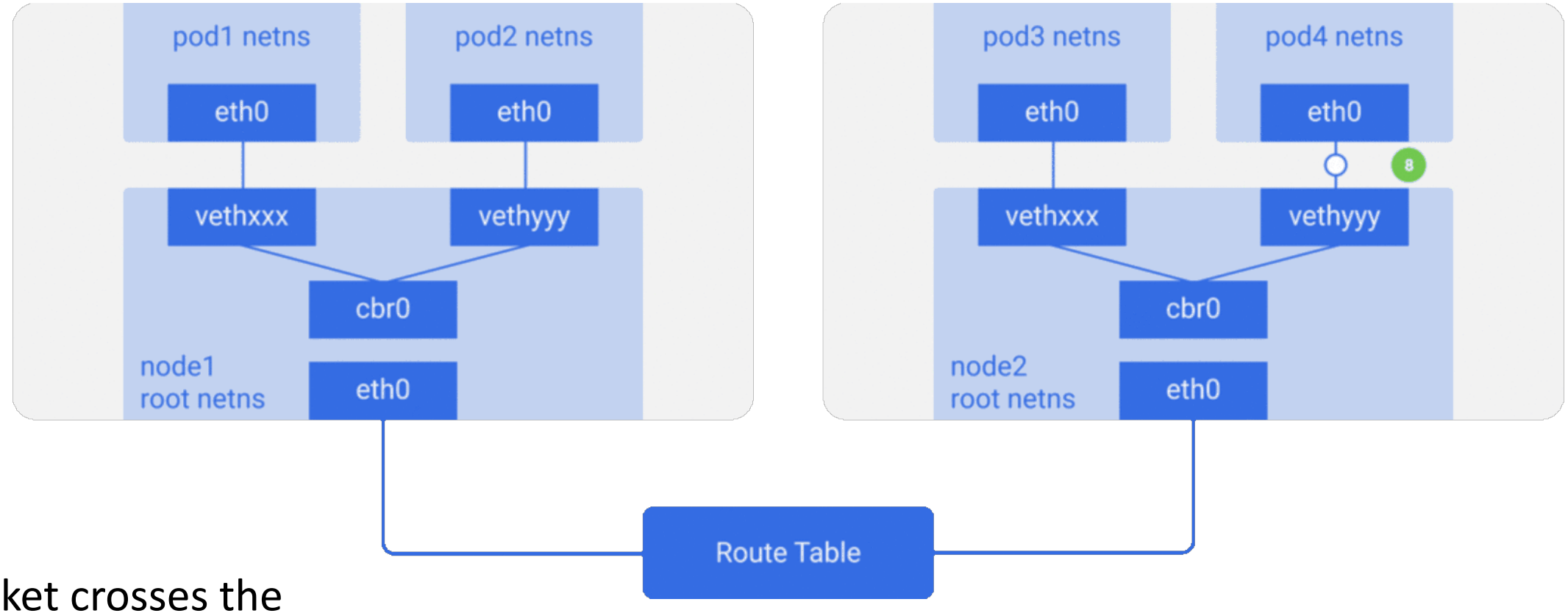
6. The node's routing table is looked up for any routes matching the POD4 IP. It finds cbr0 as the destination for this node's CIDR block.

Inter-node Communication



7. The bridge takes the packet, makes an ARP request and finds out that the IP belongs to vethyyy.

Inter-node Communication



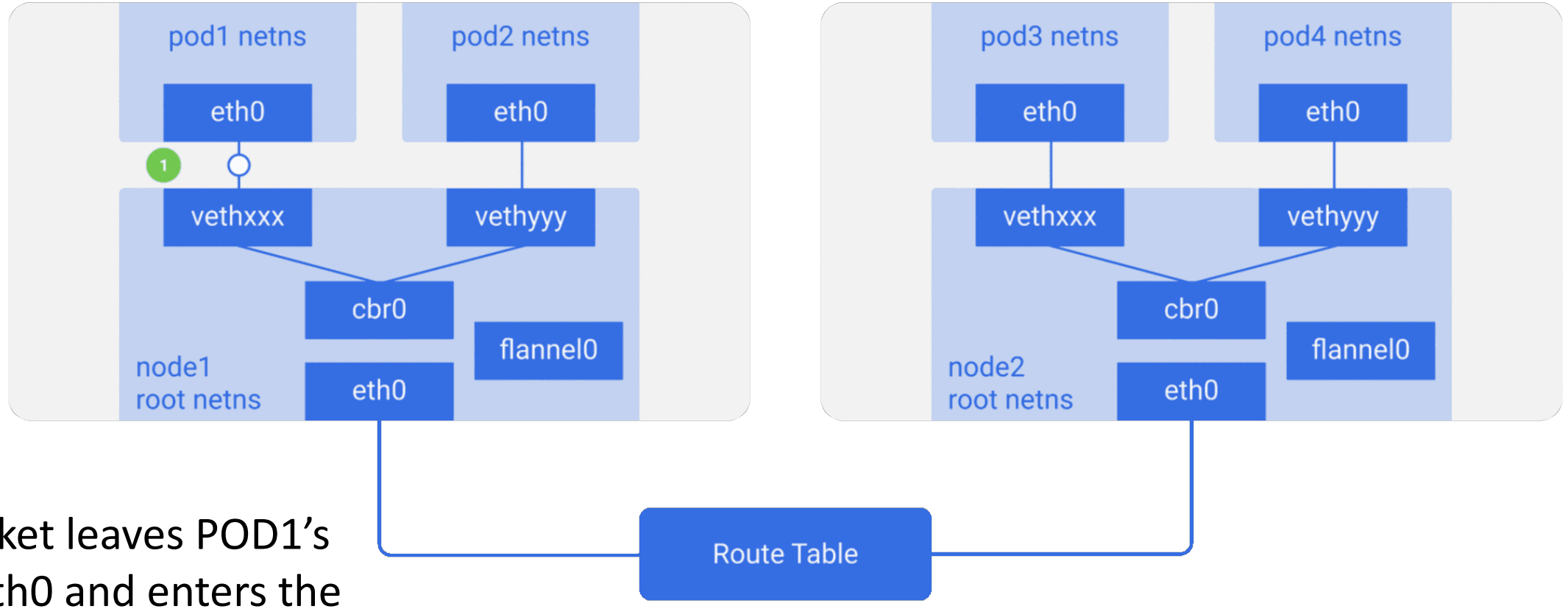
8. The packet crosses the pipe-pair and reaches POD4

Kubernetes Overlay Networking

Overlay Networking

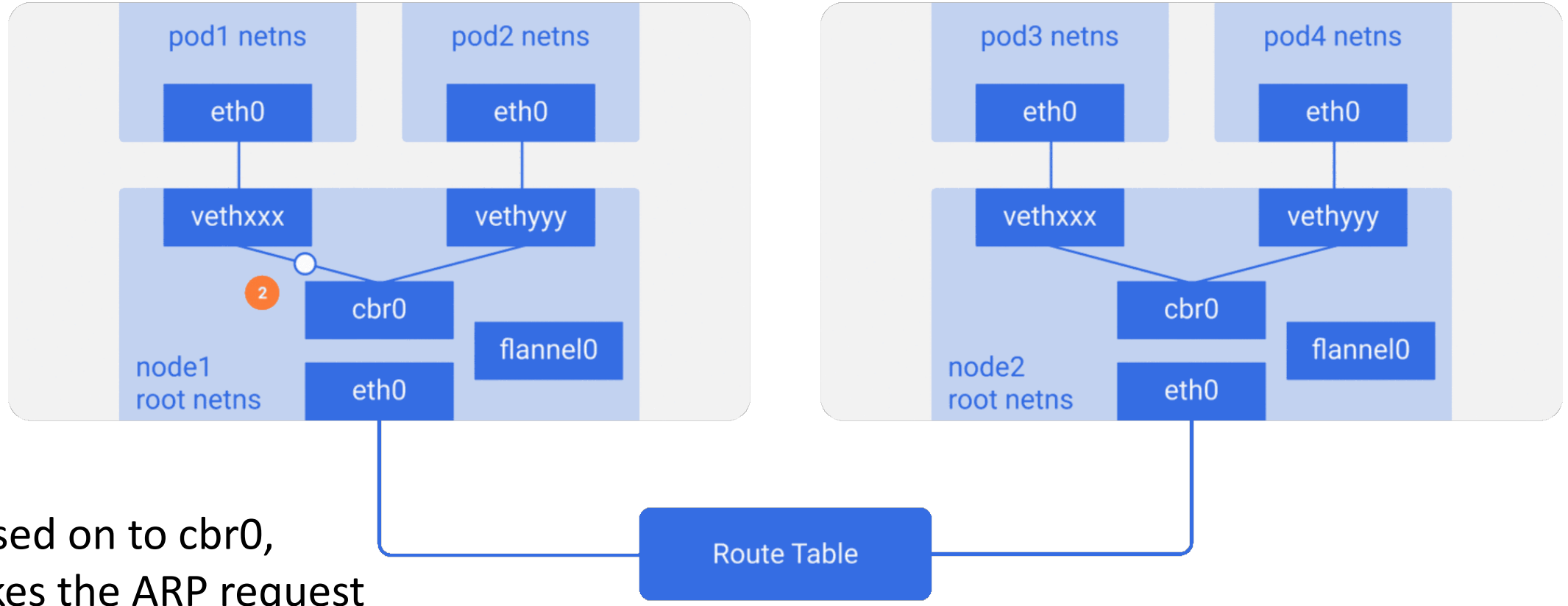
- Not required but very common
- Provide virtual IP space
- Avoid route limits (Public Cloud)
- Encapsulating a packet-in-packet, which traverses the native network across nodes.
- Can reduce throughput

Flannel Overlay Networking



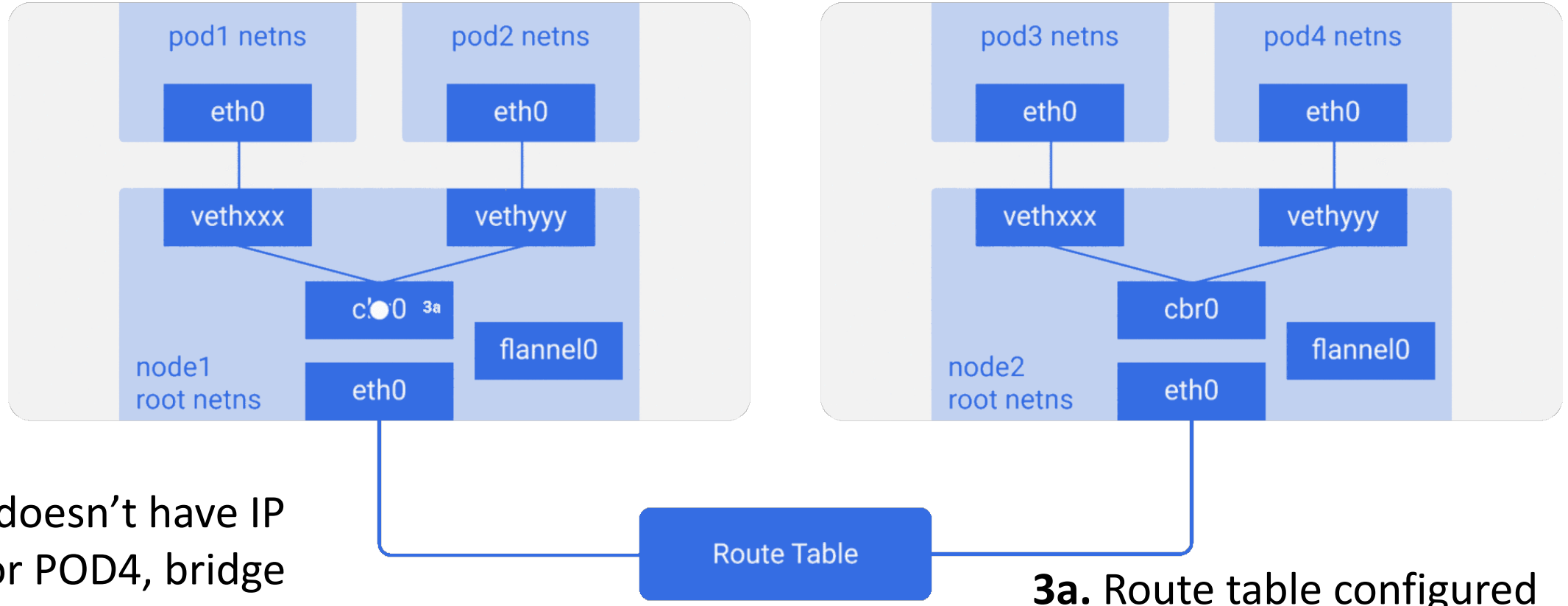
1. The packet leaves POD1's netns at eth0 and enters the root netns at vethxxx.

Flannel Overlay Networking



2. It's passed on to cbr0, which makes the ARP request to find the destination.

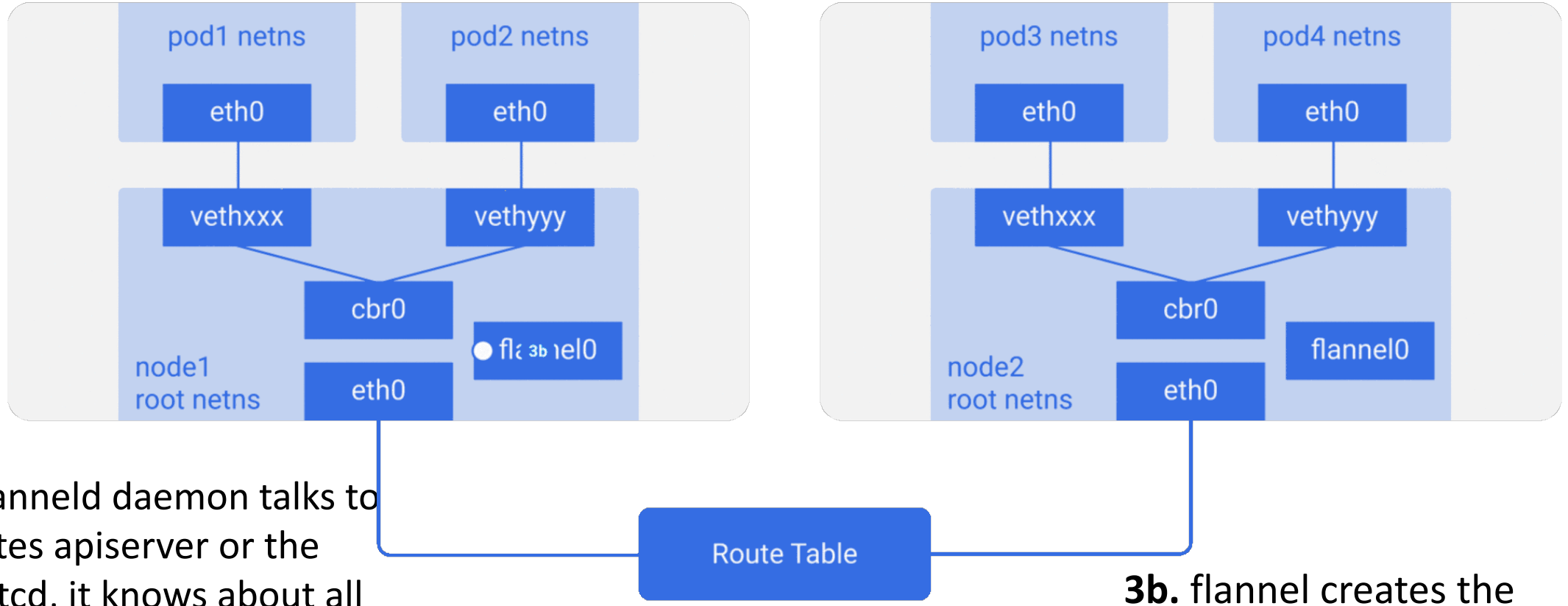
Flannel Overlay Networking



3a. Node doesn't have IP address for POD4, bridge sends to flannel0.

3a. Route table configured with flannel0 as the target for the pod network range.

Flannel Overlay Networking

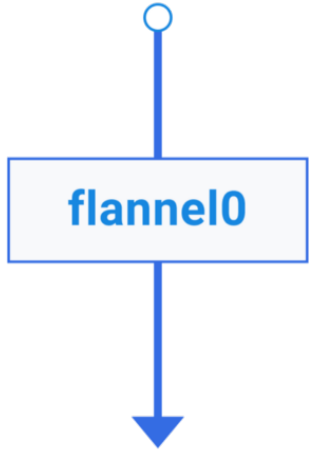


3b. As the flanneld daemon talks to the Kubernetes apiserver or the underlying etcd, it knows about all the pod IPs, what nodes they're on.

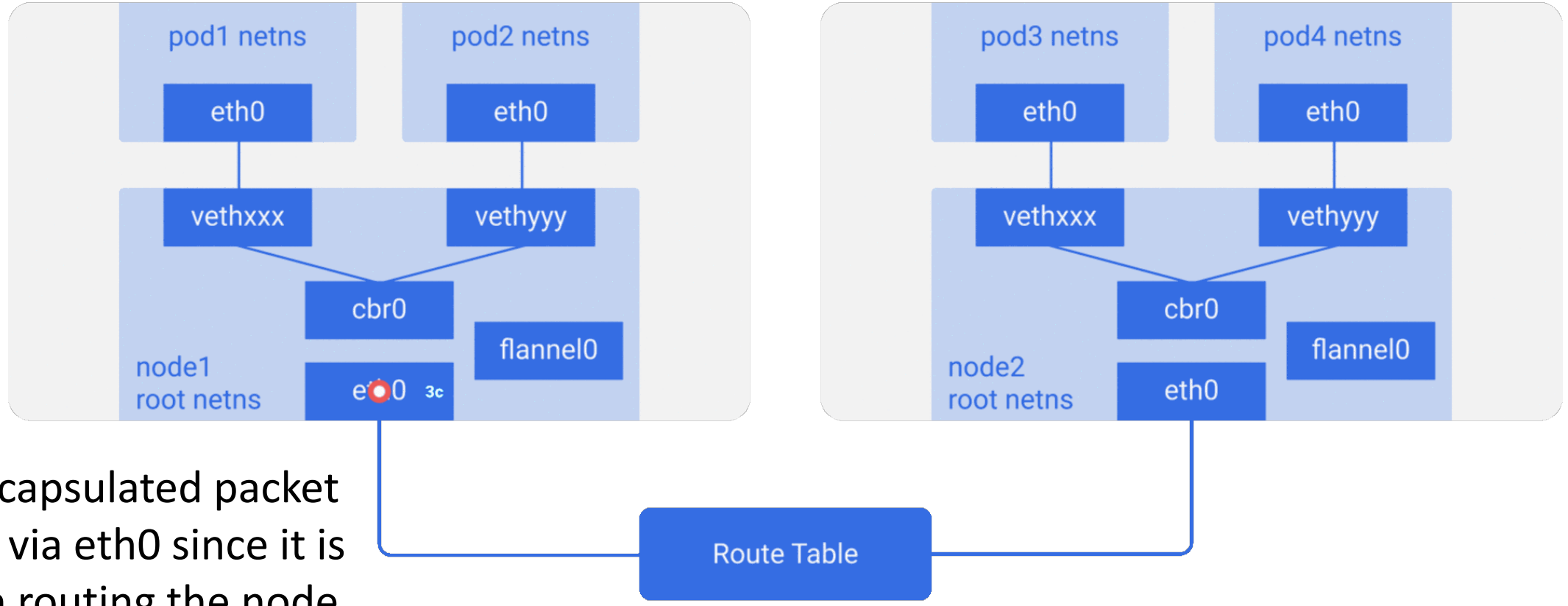
3b. flannel creates the mappings (in userspace) for pods IPs to node IPs.

Flannel Overlay Networking

flannel0 takes this packet and wraps it in a UDP packet with extra headers, changing the source and destination IPs to the respective nodes, and sends it to a special VXLAN port (generally 8472).

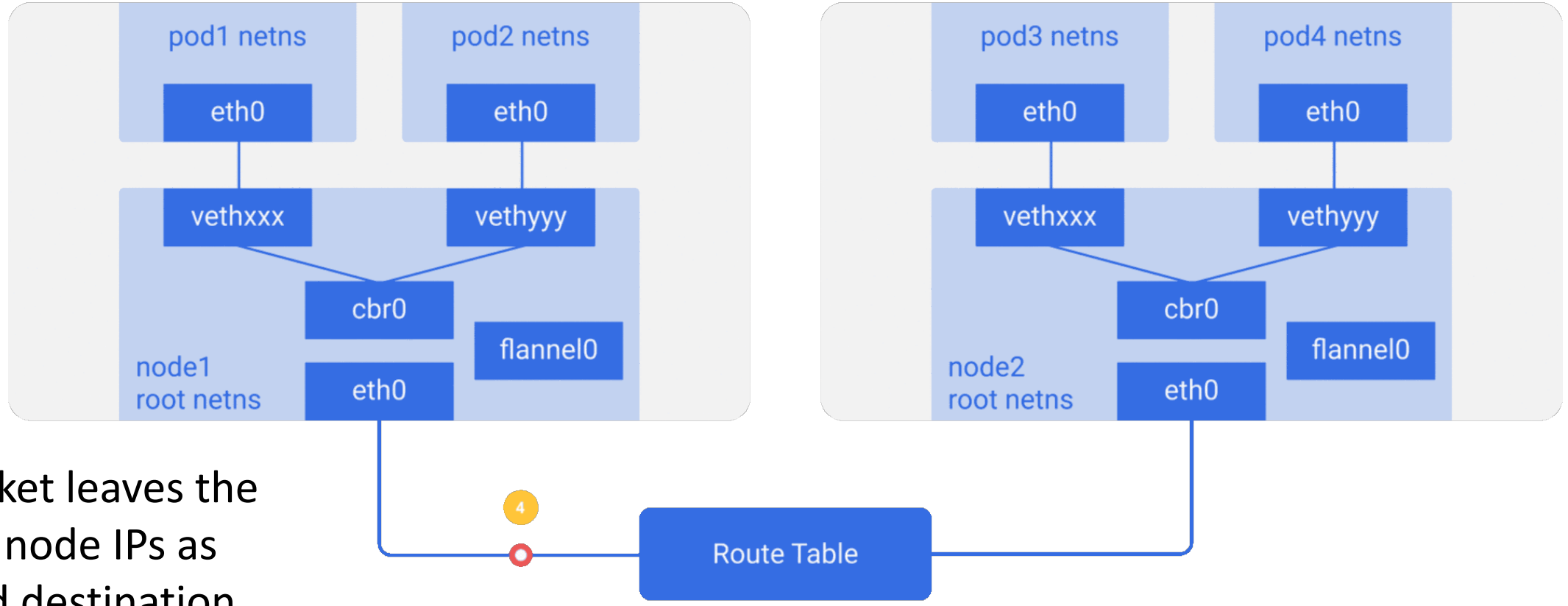


Flannel Overlay Networking



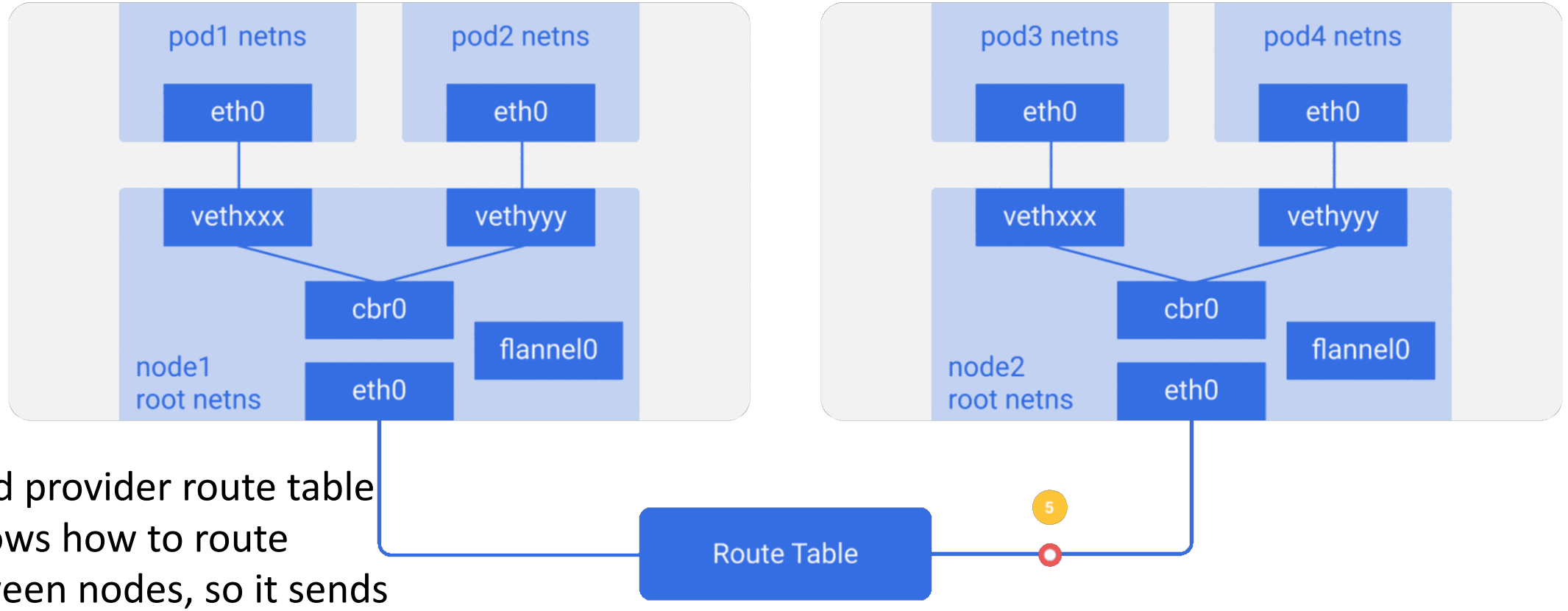
3c. The encapsulated packet is sent out via eth0 since it is involved in routing the node traffic.

Flannel Overlay Networking



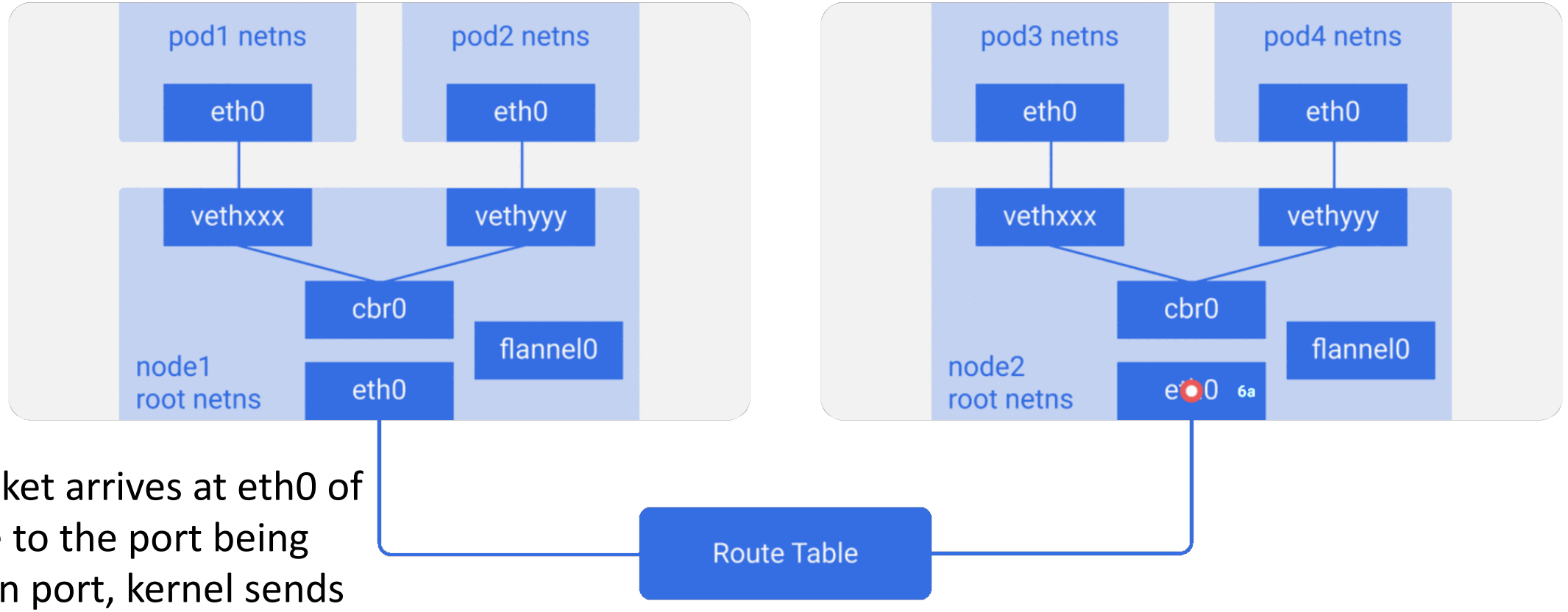
4. The packet leaves the node with node IPs as source and destination.

Flannel Overlay Networking



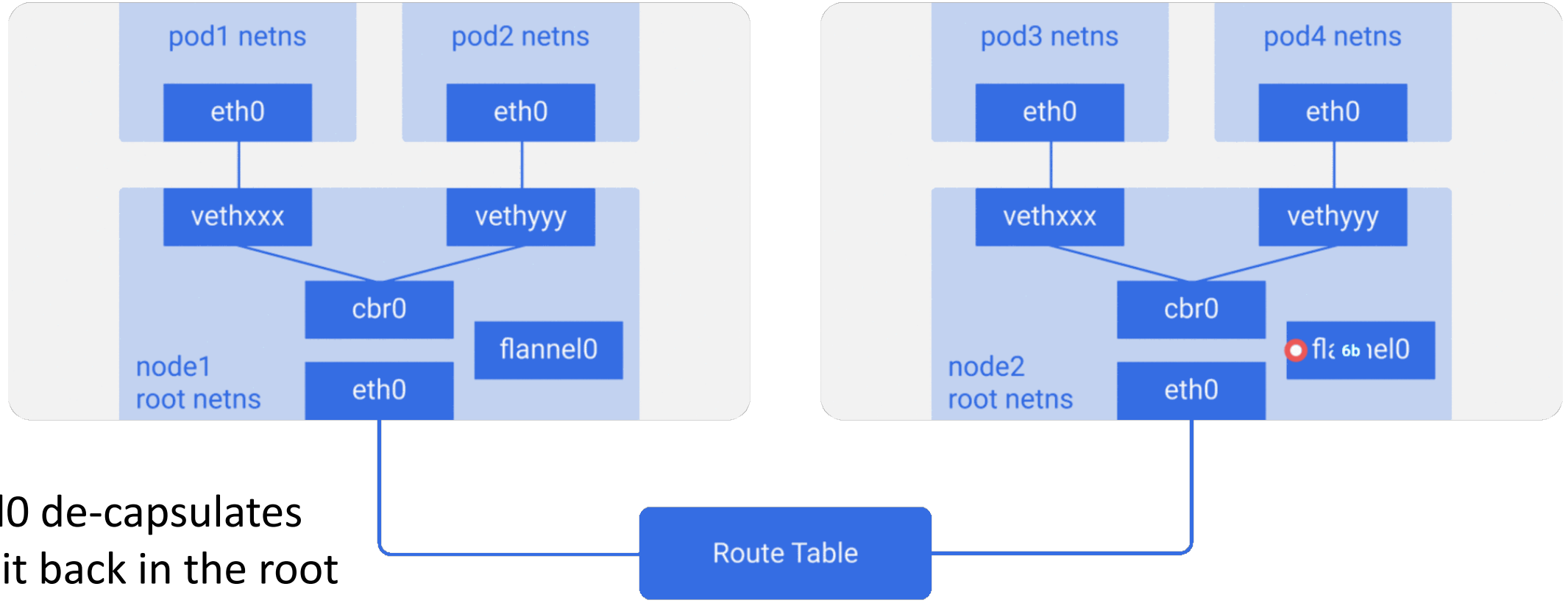
5. The cloud provider route table already knows how to route traffic between nodes, so it sends the packet to destination node2.

Flannel Overlay Networking



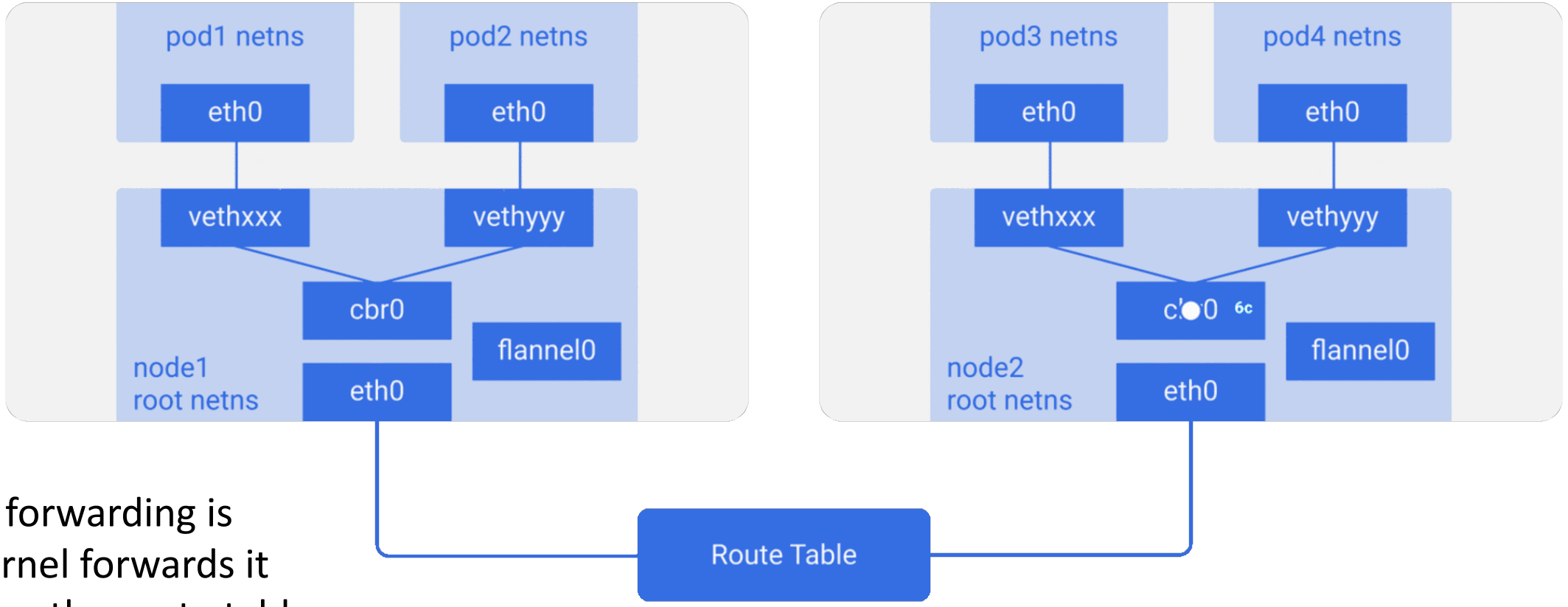
6a. The packet arrives at eth0 of node2. Due to the port being special vxlan port, kernel sends the packet to flannel0.

Flannel Overlay Networking



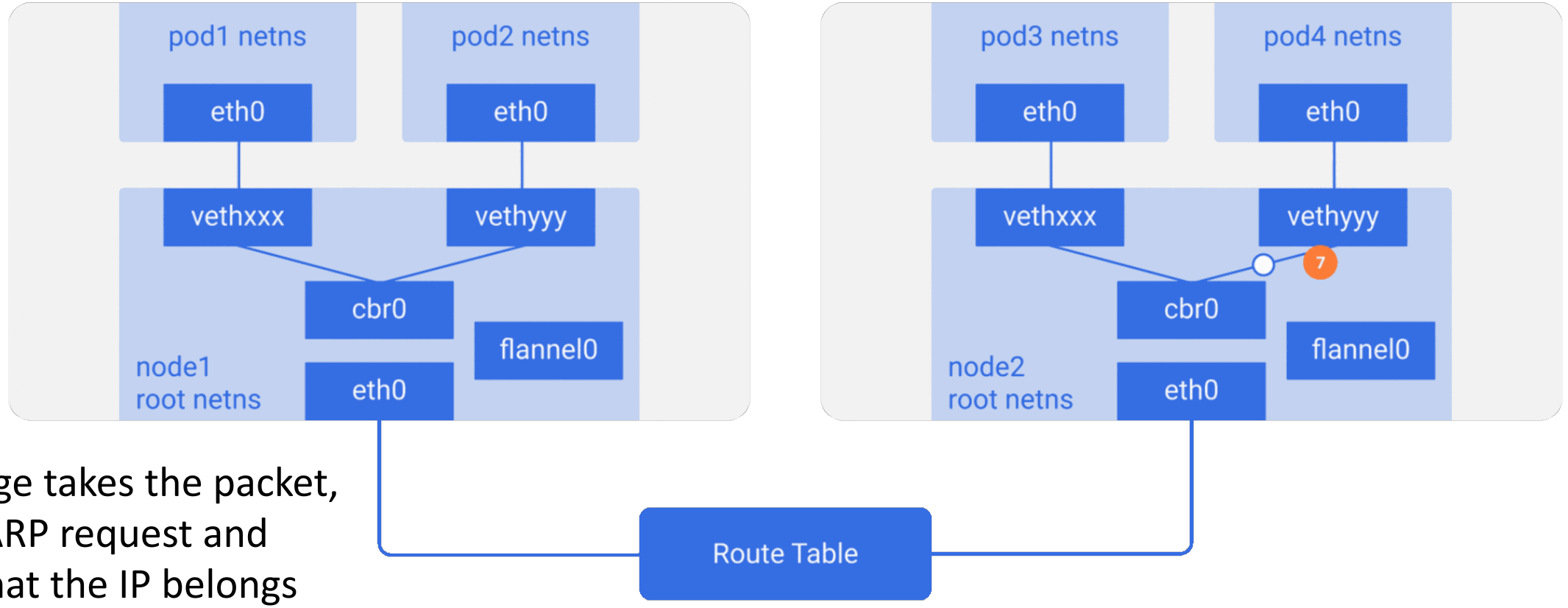
6b. flannel0 de-capsulates and emits it back in the root network namespace.

Flannel Overlay Networking



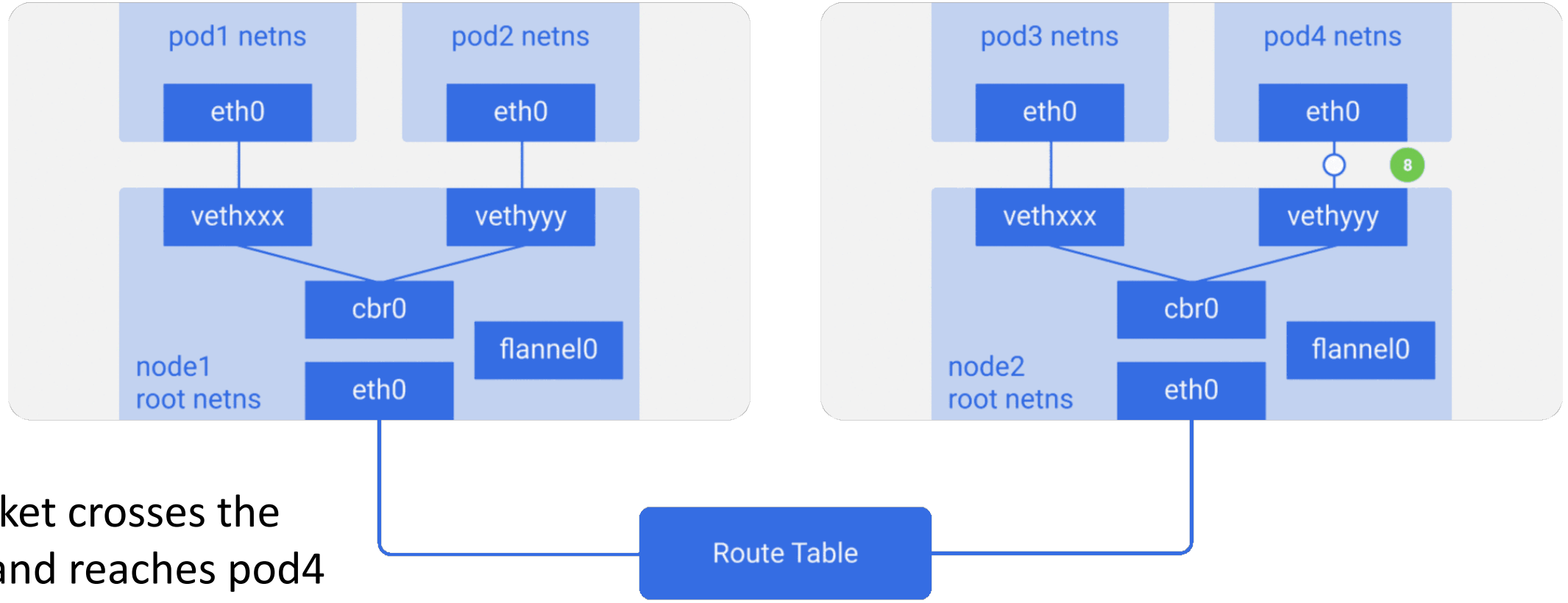
6c. Since IP forwarding is enabled, kernel forwards it to cbr0 as per the route tables.

Flannel Overlay Networking



7. The bridge takes the packet, makes an ARP request and finds out that the IP belongs to vethyyy.

Flannel Overlay Networking



8. The packet crosses the pipe-pair and reaches pod4

Kubernetes Services Overview

Kubernetes Service Objects:Microservices

Services provide abstraction between layers of an application

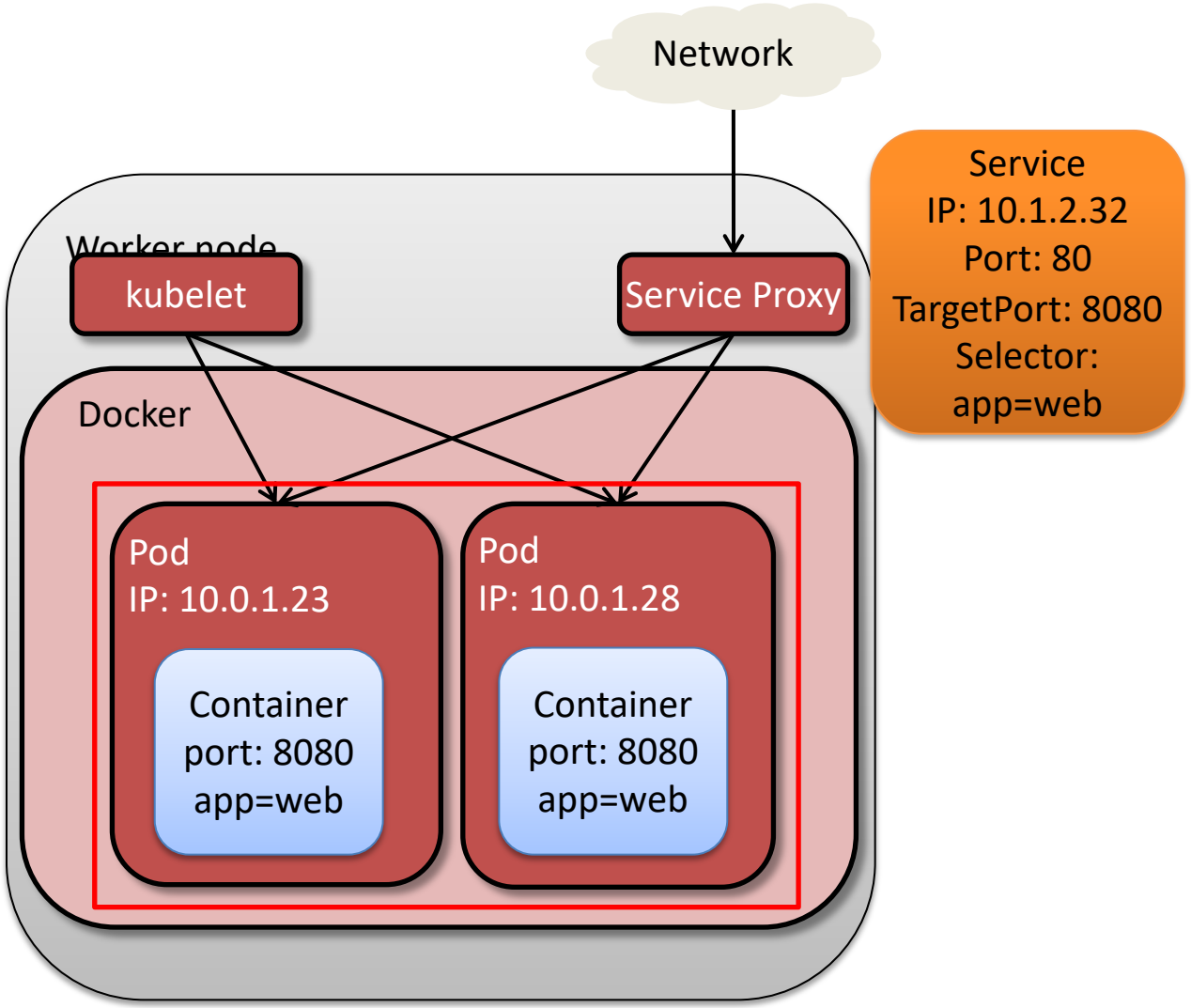
- **Service** object provides a stable IP for a collection of Pods
- Services use a label **selector** to target a specific set of pods as endpoints to receive proxied traffic
- Clients can reliably connect via the service IP and port(s), even as individual endpoint pods are dynamically created & destroyed
- Can model other types of backends using services without selectors

```yaml
apiVersion: v1
kind: Service
metadata:
  name: wordpress
  labels:
    app: wordpress
spec:
  ports:
    - port: 80
      targetPort: 8080
  selector:
    app: wordpress
    tier: frontend
  type: LoadBalancer
```

sampleservice.yaml

Services Provide Abstraction Layer for Applications

- Services can be used for communications between application tiers
- Services can also be used to expose applications outside the K8s cluster
- Services distribute requests over the set of Pods matching the service's selector
  - Service functions as TCP and UDP proxy for traffic to Pods
  - Service maps its defined ports to listening ports on Pod endpoints



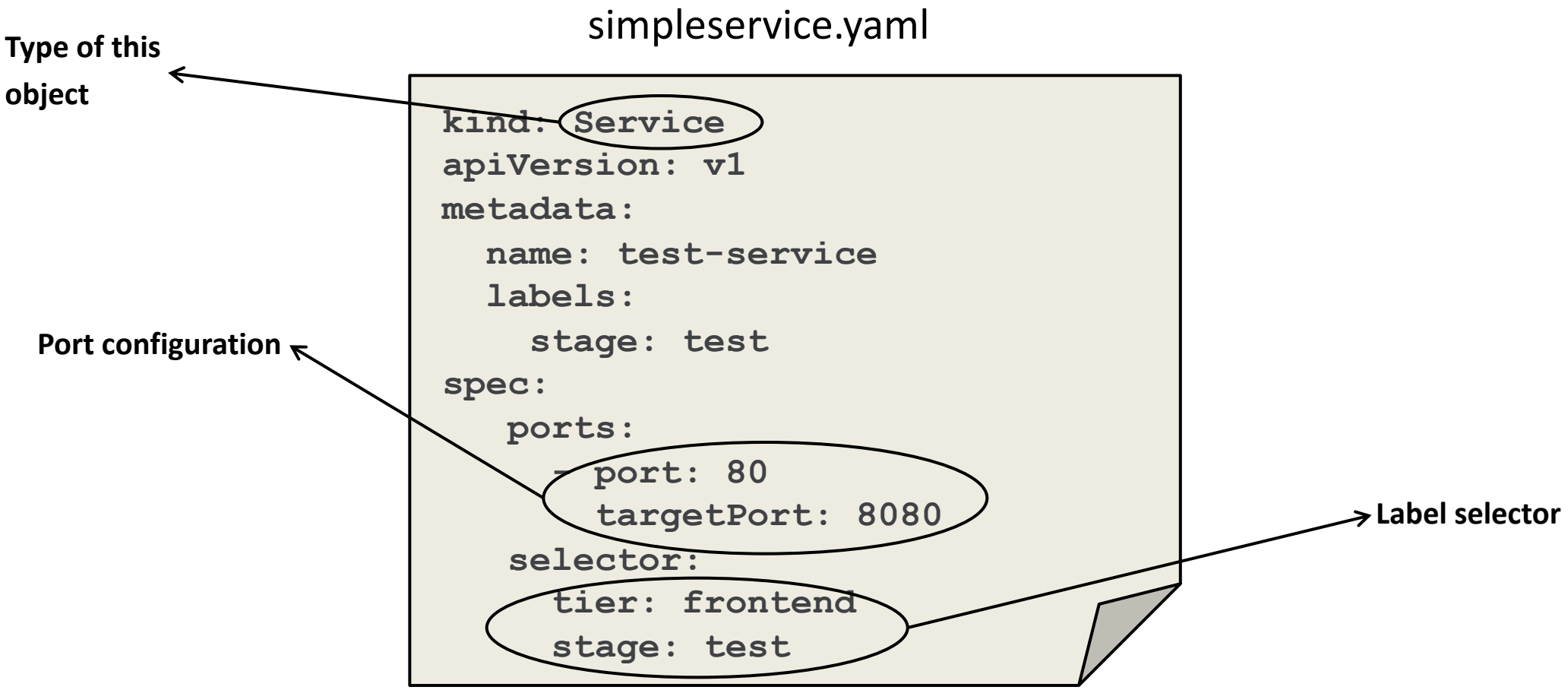
Defining a Service Using a Manifest File

Services can defined in YAML or JSON, like other K8s resources

- **kind** field value is 'Service'
- **metadata** includes
  - **name** to assign to Service
- **spec** includes the ports associated with the Service
  - **port** is the Service's port value
  - t**argetPort** is connection port on selected pods (default: **port** value)
- **selector** specifies a set of label KV pairs to identify the endpoint pods for the Service

```
kind: Service
apiVersion: v1
metadata:
  name: test-service
  labels:
    stage: test
spec:
   ports:
     - port: 80
       targetPort: 8080
   selector:
     tier: frontend
     stage: test
```

Reviewing a Service Manifest File



ServiceTypes and Exposing Applications

- By default, a Service is assigned a cluster-internal IP – good for back-ends
  - **ServiceType** ClusterIP
- To make a front-end Service accessible outside the cluster, there are other ServiceTypes available
  - **ServiceType** NodePort exposes Service on each Node's IP on a static port
  - **ServiceType** LoadBalancer exposes the Service externally using cloud provider's load balancer
- Can also use a Service to expose an external resource via **ServiceType** ExternalName

```
kind: Service
apiVersion: v1
metadata:
  name: test-service
  labels:
    stage: test
spec:
   ports:
     - port: 80
       targetPort: 8080
       nodePort: 30080
   selector:
     tier: frontend
   type: NodePort
```

Exposing Services at L7 through Ingresses

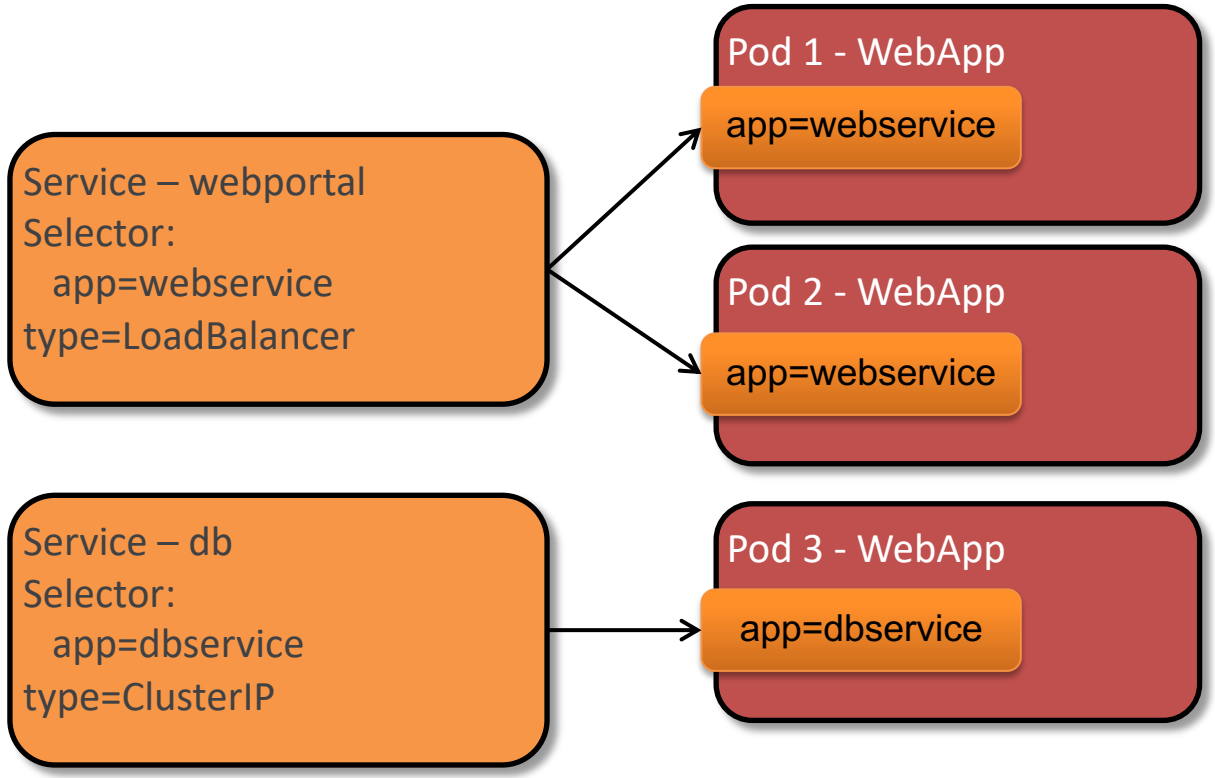
- Kubernetes also provides the facility to define **Ingress** resource to configure an external loadbalancer at L7
- Spec of Ingress resource is a set of rules matching HTTP host/url paths to specific Service backends
- Ingresses require the cluster to be running an appropriately configured Ingress controller to function (e.g. nginx)
- Useful for implementing fanout, Service backends for virtual hosts, etc.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: test-ingress
spec:
  rules:
  -host: bar.foo.com
   http:
      paths:
      - path: /first
        backend:
          serviceName:
firstservice
          servicePort: 80
      - path: /second
        backend:
          serviceName:
secondservice
          servicePort: 80
```

Selecting Pods as Service Endpoints

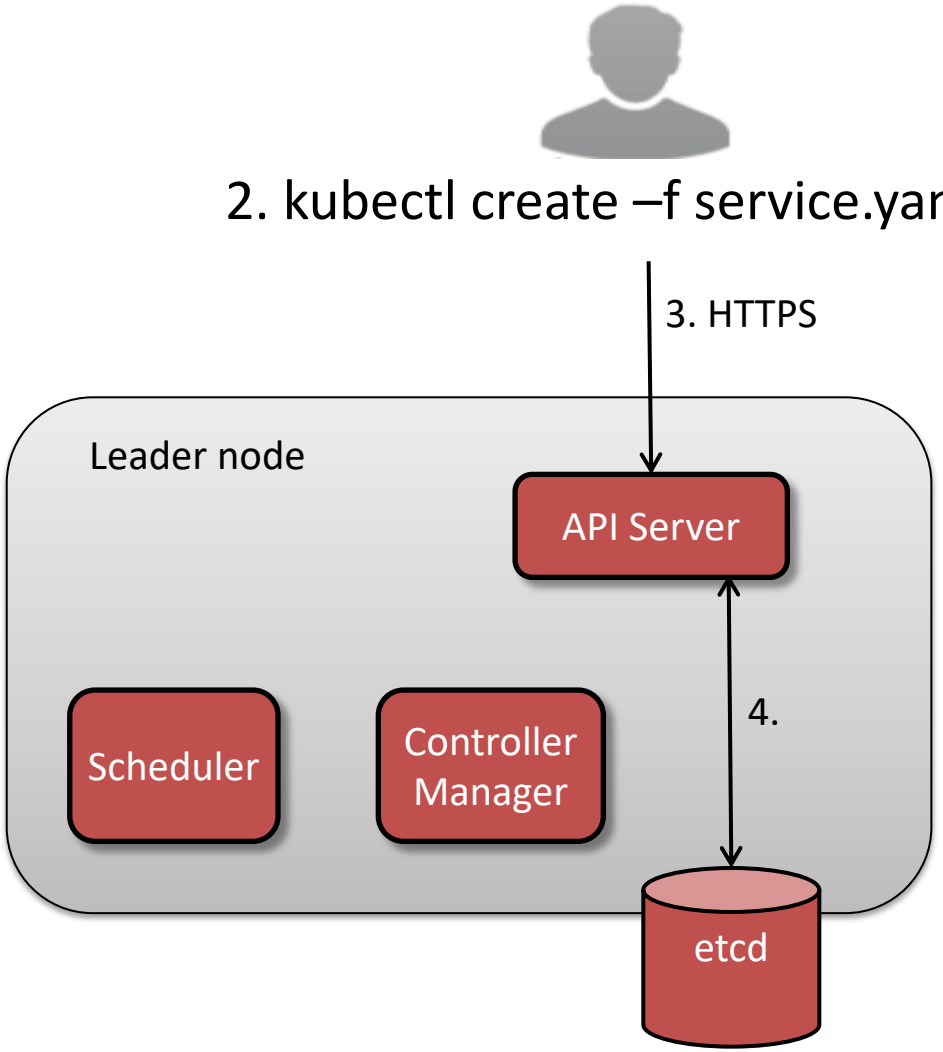
Service's pod selector based on labels

- Multiple pods can have the same label, unlike pod names which are unique in the namespace
- K8s system re-evaluates Service's selector continuously
- K8s maintains **endpoints** object of same name with list of pod IP:port's matching Service's selector



Services Management

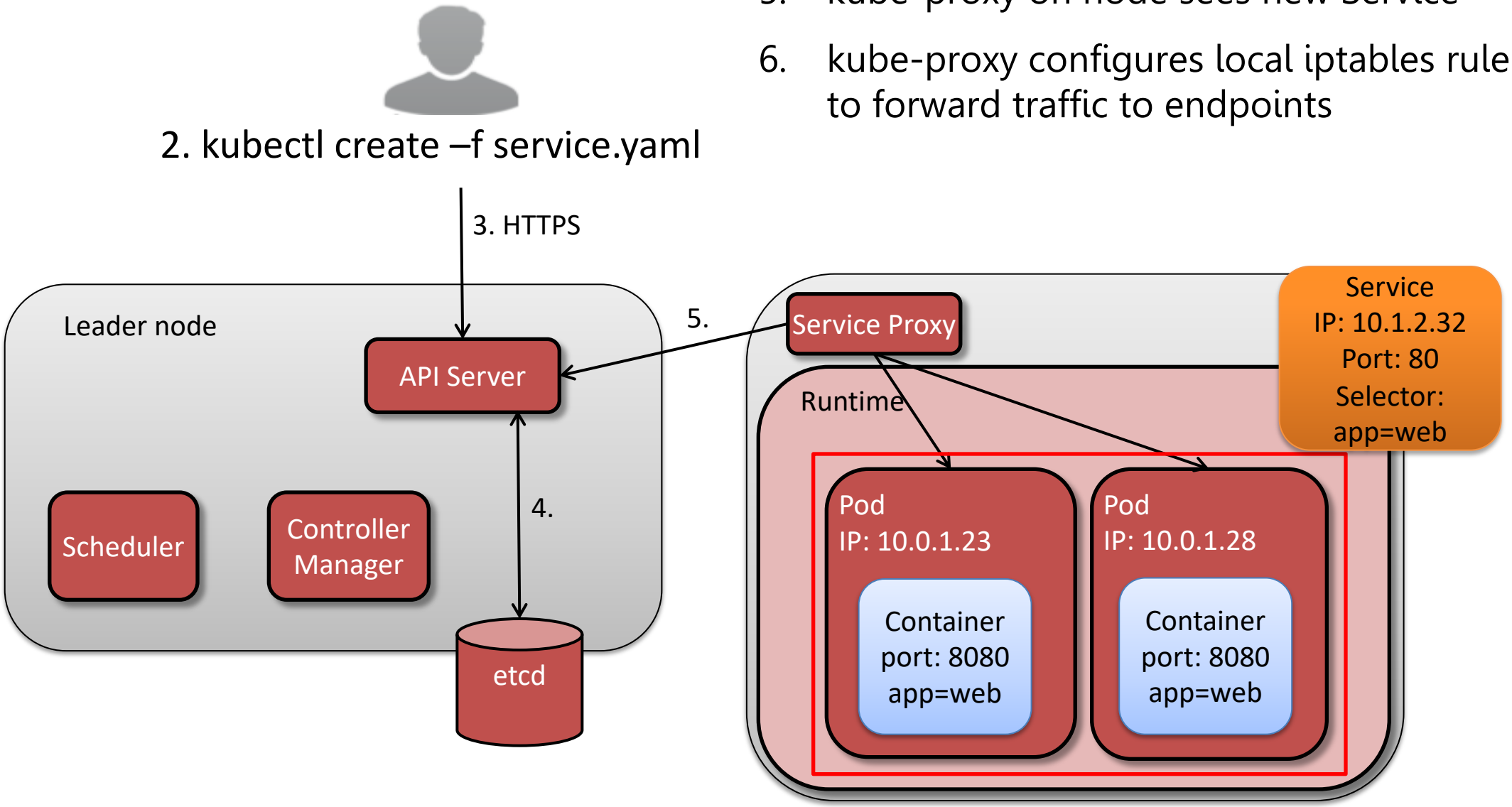
Service Creation Process



1. User writes a Service manifest file
2. User requests creation of Service from manifest via CLI
3. CLI tool marshals parameters into K8s RESTful API request (HTTP POST)
4. kube-apiserver creates new Service object record in etcd

Service Creation Process

5. kube-proxy on node sees new Service
6. kube-proxy configures local iptables rules to forward traffic to endpoints



Accessing Services Externally

Checking the service external IP

- Configured as type LoadBalancer, a configured cluster will provide an externally accessible IP for your Service

```
$ kubectl get services test-service
NAME         CLUSTER-IP   EXTERNAL-IP     PORT(S)    AGE
test-service 10.3.247.123 104.154.105.198 8080/TCP   4h
```

Deleting Services

Services can be deleted anytime

- Service deletion does not affect pods targeted by the Service's selector
- Can be done referencing the Service name or a manifest for the resource

```
$ kubectl delete -f simpleservice.yaml
```

```
$ kubectl delete service test-service
```

Service Discovery via DNS

Kubernetes advertises services via cluster DNS

- Kubernetes uses a cluster-internal DNS add-on to create and manage records for all Services in the cluster
- Pod's DNS search list includes its own namespace and cluster default domain by default

```
$ kubectl get svc
NAME           CLUSTER-IP    EXTERNAL-IP   PORT(S)          AGE
kubernetes     10.0.0.1      <none>        443/TCP          11d
test-service   10.0.0.102    <nodes>       8080:30464/TCP   2d

kubectl exec -ti busybox1 -- nslookup test-service.default
Server:    10.0.0.10
Address 1: 10.0.0.10 coredns.kube-system.svc.cluster.local

Name:      test-service.default
Address 1: 10.0.0.102 test-service.default.svc.cluster.local
```

Deployments

What is a Deployment?

Kubernetes controller optimal for stateless applications

- Deployments allow you to declaratively manage pods, including replication
- Deployments support
  - Creating, rolling out, and rolling back changes to homogeneous set of pods
  - Scaling set of pods out and back declaratively
- Deployments include
  - Implicit Replica Set controller to handle pod replicas
  - Template spec of pods to be created and managed – no need to separately create pods
- Deployments used for web applications, mobile back-ends, API's

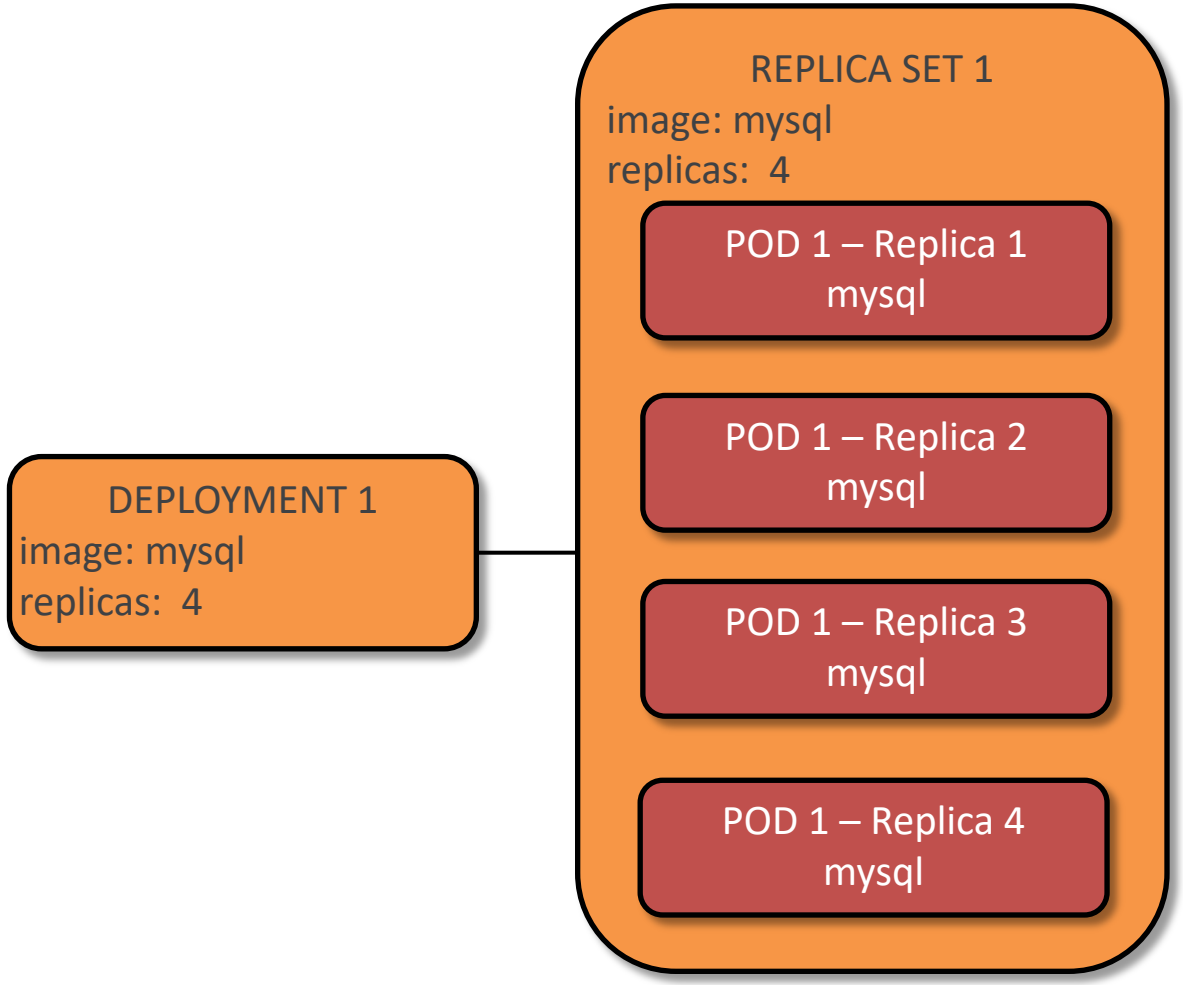
What if my Application isn't Stateless?

- Kubernetes provides other controller objects for applications that need different deployment schemes
- **StatefulSets** (previously PetSets) control deployment of pods for applications that need more stable deployment contexts
  - Pods in StatefulSets have unique ordinal, stable network identity and stable storage using persistent volumes
  - When pods are deployed, they are created in sequence of ordinals 1..N
  - Pod N must be running and ready before Pod N+1 is deployed
  - When pods are destroyed, they are terminated in reverse sequence N..1
- **DaemonSets** ensure that a replica of a specified pod is running on every node (or every selected node) in the cluster
- **Jobs** manage sets of pods where N must run to successful completion

Deployments Control ReplicaSet Controllers

Definition of how many replicated Pods should exist

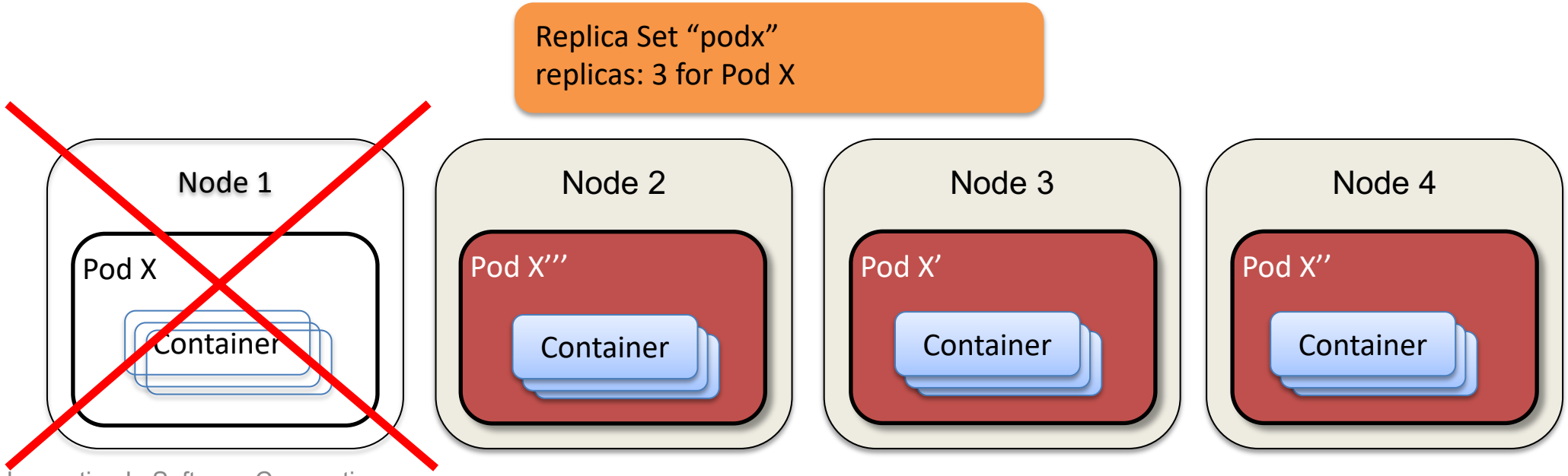
- Deployment creates and manages a Replica Set that manages a set of pods
- Replica count can be adjusted as needed to scale the Replica Set out and back
- Replica Set successor to the ReplicationController object



What is a Replica Set?

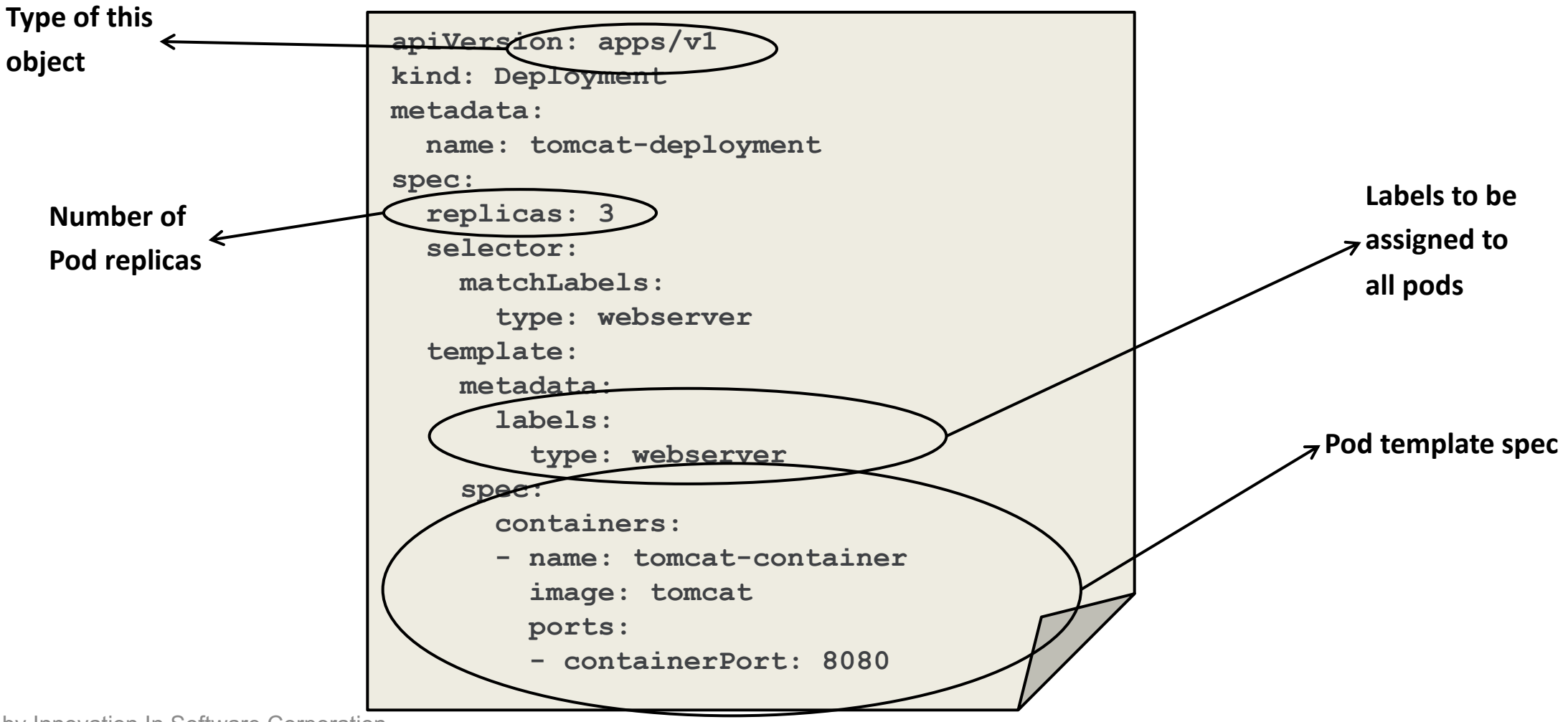
Provides scaling and high availability

- Replica count can be changed to provide scaling on demand as needed
- If the node hosting a pod fails, the Kubernetes cluster will recreate the pod elsewhere to achieve the target number of replicas

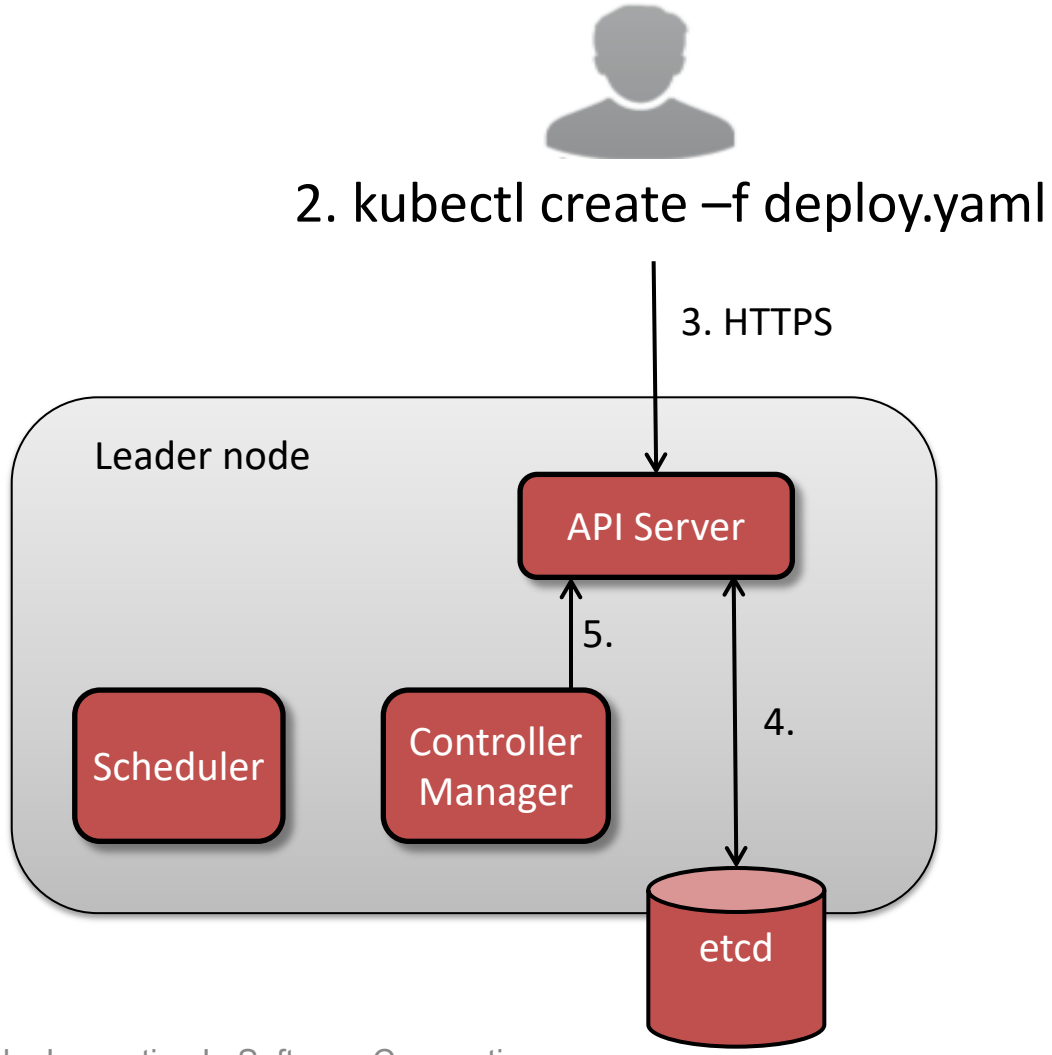


Examining a Deployment Manifest File

simpledeployment.yaml



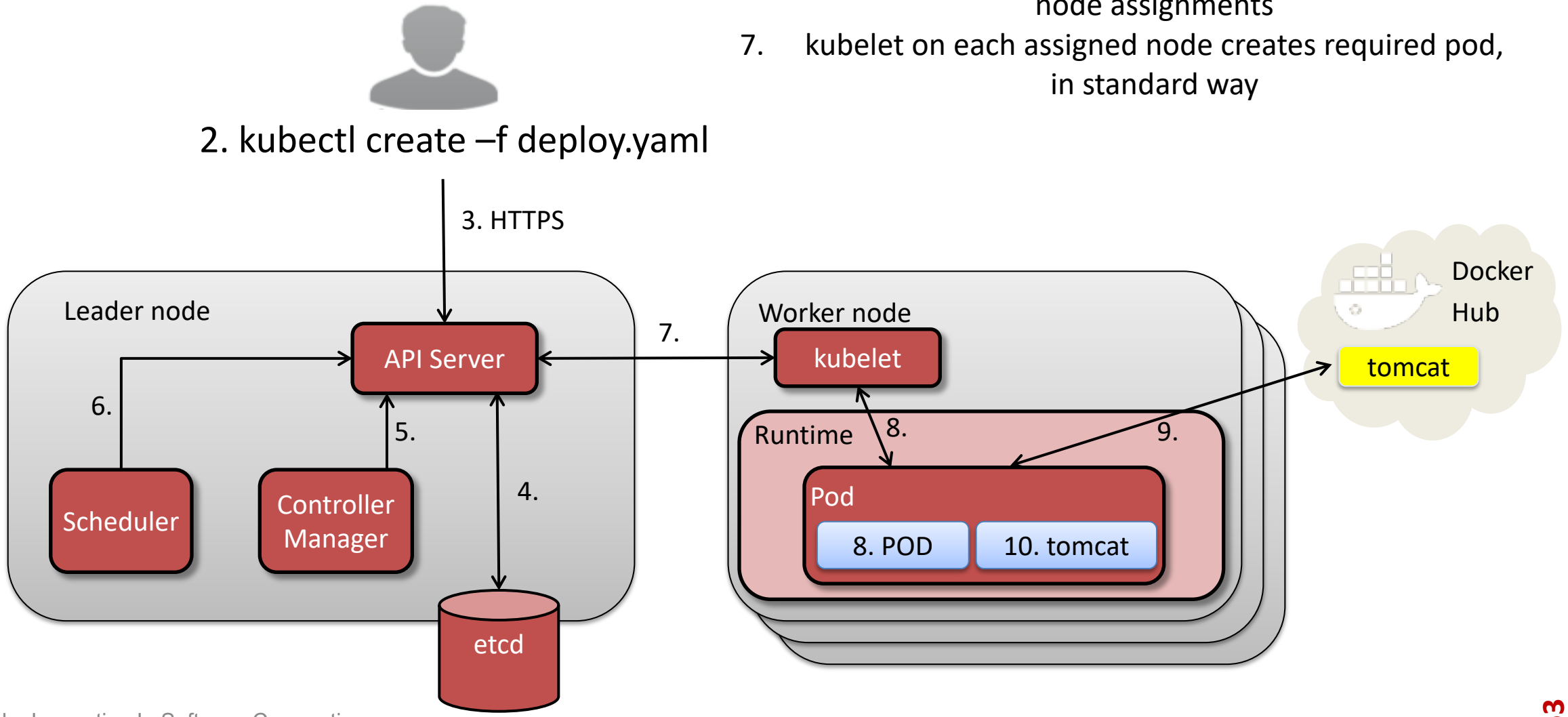
Deployment Creation Process



1. User writes a deployment manifest file
2. User requests creation of deployment from manifest via CLI
3. CLI tool marshals parameters into K8s RESTful API request (HTTP POST)
4. kube-apiserver creates new deployment object record in etcd, and new Replica Set object
5. kube-controller-manager sees new Replica Set and
   a. Evaluates state of existing vs. required replicas
   b. Submits pod creation requests to API to create required number of replicas

Deployment Creation Process

6. Scheduler sees new pod objects created and selects node assignments
7. kubelet on each assigned node creates required pod, in standard way



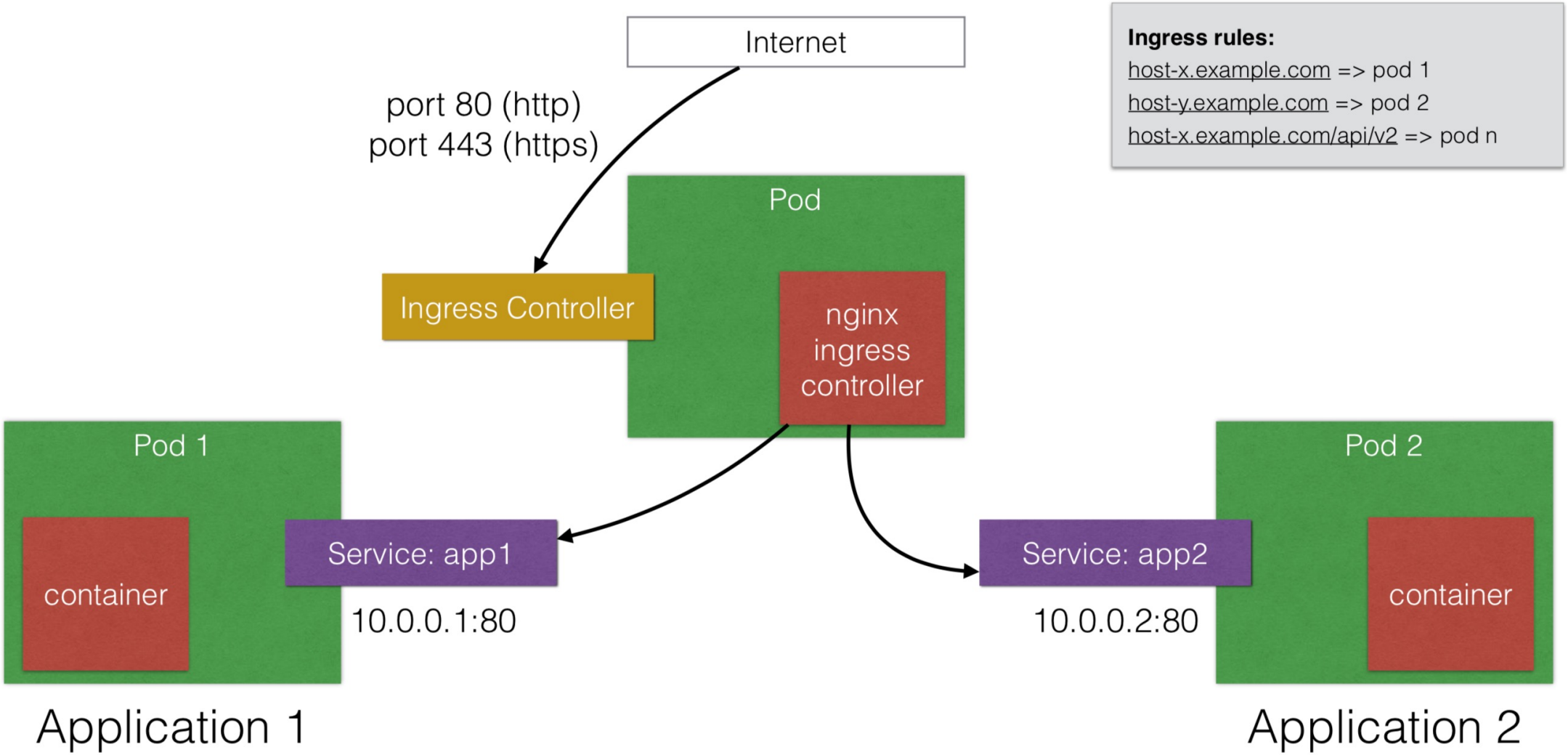
Lab: Deployments

Kubernetes Ingress Overview

Ingress

- L7 (instead of L4)
- HTTP (instead of tcp/udp)
- Ingress can route paths differently:
  - /api/app1 -> app1 pods
  - /api/app2 -> app2 pods
- Requires ingress provider (might be cloud, might be in-cluster)

Ingress



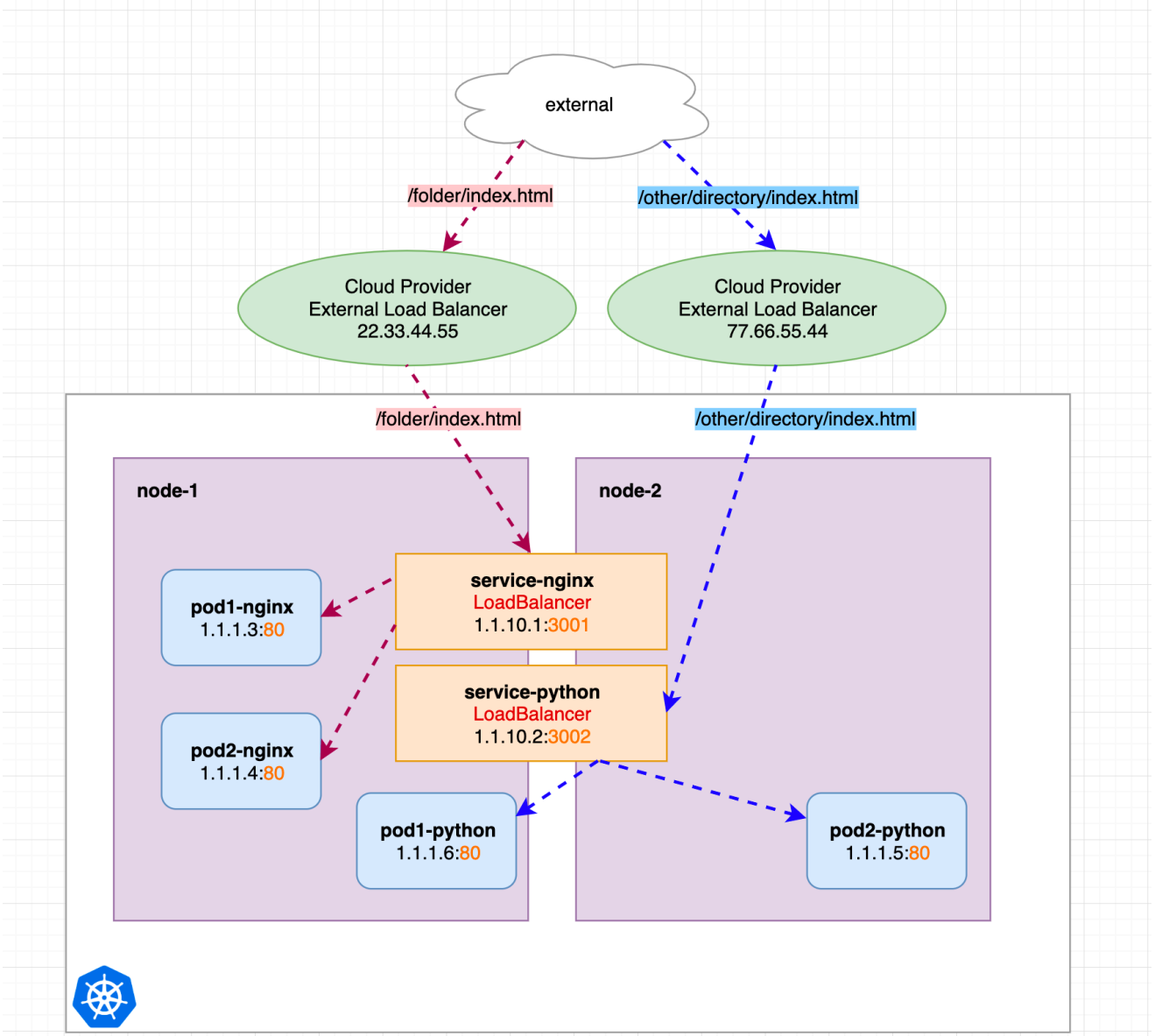
Ingress Definition

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: helloworld-rules
spec:
  rules:
  - host: helloworld-v1.example.com
    http:
      paths:
      - path: /
        backend:
          serviceName: helloworld-v1
          servicePort: 80
  - host: helloworld-v2.example.com
    http:
      paths:
      - path: /
        backend:
          serviceName: helloworld-v2
          servicePort: 80
```

Ingress

- Create Ingress in same namespace as service
- Cannot route traffic to a service in a different namespace where you don’t have the ingress object.
- An Ingress controller is required
- Support HTTPS and can terminate at the Ingress controller
- Each cloud has their own implementation of Ingress controller.

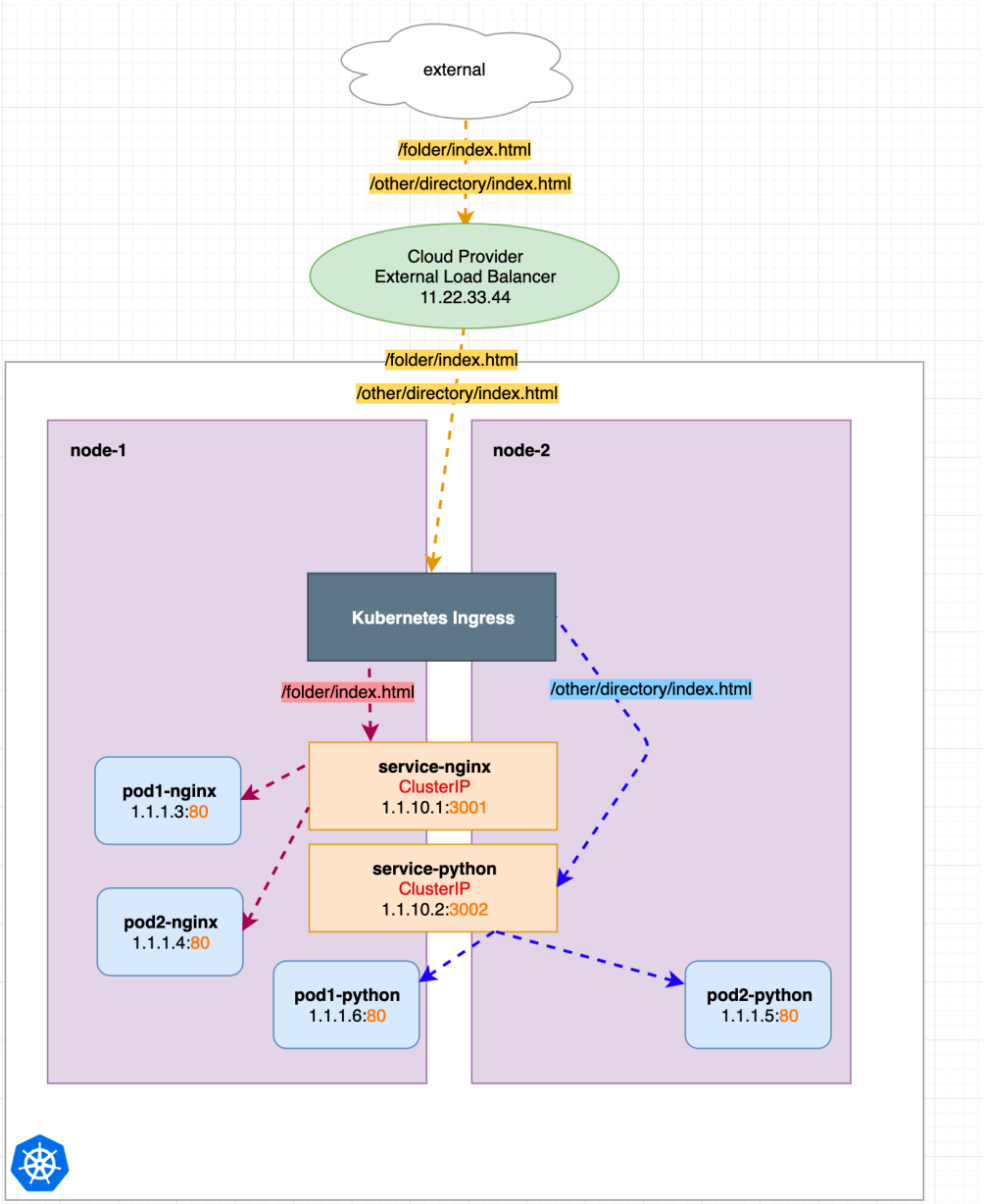
Service Type = Load Balancer



Load Balancer challenges

- Two Load Balancers requires 2 public IPs
- Load Balancers cost money
- The more services we have the more LBs required, which leads to increased bill.

Ingress controller



Ingress controller options

Nginx

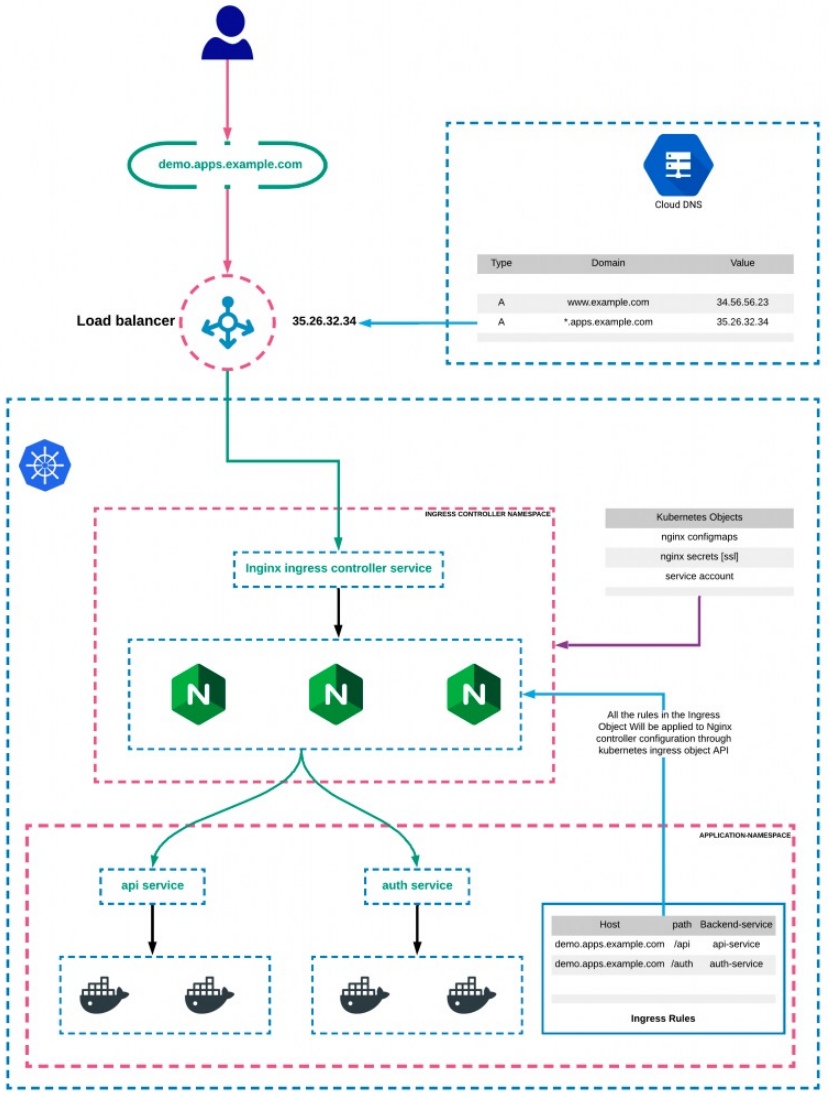
Traefik

Ingress process (nginx)

- nginx.conf is a Go template which can talk to Kubernetes Ingress API and pull latest values for traffic routing in real time.
- Nginx controller talks to Kubernetes Ingress API to check if there is a routing rule created
- If it finds any matching Ingress rules they are applied to Nginx controller configuration nginx.conf inside the pod using the Go template

- PROTIP: nginx.conf available in pod /etc/nginx/nginx.conf with all rules.

Ingress diagram



ConfigMap

ConfigMap

ConfigMap

- Many applications require configuration via:
  - Config Files
  - Command-Line Arguments
  - Environment Variables
- These need to be decoupled from images to keep portable
- ConfigMap API provides mechanisms to inject containers with configuration data
- Store individual properties or entire config files/JSON blobs
- Key-Value Pairs

ConfigMap

- Not meant for sensitive information
- PODs or controllers can use ConfigMaps

1. Populate the value of environment variables
2. Set command-line arguments in a container
3. Populate config files in a volume

```
kind: ConfigMap
apiVersion: v1
metadata:
   creationTimestamp: 2016-02-18T19:14:38Z
   name: example-config
   namespace: default
data:
   example.property.1: hello
   example.property.2: world
   example.property.file: |-
      property.1=value-1
      property.2=value-2
      property.3=value-3
```

ConfigMap from directory

- 2 files in docs/user-guide/configmap/kubectl
  - game.properties
  - ui.properties

game.properties

```
enemies=aliens
lives=3
enemies.cheat=true
enemies.cheat.level=noGoodRotten
secret.code.passphrase=UUDDLRLRBABAS
secret.code.allowed=true
secret.code.lives=30
```

ui.properties

```
color.good=purple
color.bad=yellow
allow.textmode=true
how.nice.to.look=fairlyNice
```

ConfigMap from directory

```
kubectl create configmap game-config --from-file=docs/user-guide/configmap/kubectl
```

```
kubectl describe configmaps game-config
Name: game-config
Namespace: default
Labels: <none>
Annotations: <none>
Data
====
game.properties: 121 bytes
ui.properties: 83 bytes
```

ConfigMap from directory

```
kubectl get configmaps game-config -o yaml
```

```
apiVersion: v1
data:
    game.properties: |-
        enemies=aliens
        lives=3
        …
ui.properties: |
       color.good=purple
       …
kind: ConfigMap
metadata:
    creationTimestamp: 2016-02-18T18:34:05Z
    name: game-config
    namespace: default
    resourceVersion: "407"-
    selfLink: /api/v1/namespaces/default/configmaps/game-config
    uid: 30944725-d66e-11e5-8cd0-68f728db1985
```

ConfigMap from files

```
kubectl create configmap \
game-config-2 \
--from-file=file1 \
--from-file=file2
```

```
kubectl get configmaps \
game-config-2 \
-o yaml
```

```
apiVersion: v1
data:
    game.properties: |-
        enemies=aliens
        lives=3
        …
ui.properties: |
       color.good=purple
       …
kind: ConfigMap
metadata:
    creationTimestamp: 2016-02-18T18:52:05Z
    name: game-config-2
    namespace: default
    resourceVersion: ”516"-
    selfLink: /api/v1/namespaces/default/configmaps/game-config-2
    uid: b4952dc3-d670-11e5-8cd0-68f728db1985
```

ConfigMap options

--from-file=/path/to/directory

--from-file=/path/to/file1 (/path/to/file2)

Literal key=value: --from-literal=special.how=very

ConfigMap in PODs

- Populate Environment Variables

```
apiVersion: v1
kind: ConfigMap
metadata:
    name: special-config
    namespace: default
data:
    special.how: very
    special.type: charm
```

OUTPUT

```
SPECIAL_LEVEL_KEY=very
SPECIAL_TYPE_KEY=charm
```

```
apiVersion: v1
kind: Pod
metadata:
    name: dapi-test-pod
 spec:
   containers:
       - name: test-container
         image: busybox
         command: [“/bin/sh”, “-c, “env” ]
         env:
             - name: SPECIAL_LEVEL_KEY
               valueFrom:
                   configMapKeyRef:
                       name: special-config
                       key: special.how
   restartPolicy: Never
```

ConfigMap Restrictions

- ConfigMaps must be created before they are consumed
- ConfigMaps can only be referenced by objects in the same namespace
- Quota for ConfigMap size not implemented yet
- Can only use ConfigMap for PODs created through API server.

Secrets

Application Secrets

What secrets do applications have?

- Database credentials
- API credentials & endpoints (Twitter, Facebook etc.)
- Infrastructure API credentials (Google, Azure, AWS)
- Private keys (TLS, SSH)
- Many more!

Application Secrets

It is a bad idea to include these secrets in your code.

- Accidentally push up to GitHub with your code
- Push into your file storage and forget about
- Etc.

Application Secrets

There are bots crawling GitHub searching for secrets

Real life example:
Dev put keys out on GitHub, woke up next morning with a ton of emails and missed calls from Amazon

- 140 instances running under Dev's account.
- $2,375 worth of Bitcoin mining

Create Secret

- Designed to hold all kinds of sensitive information
- Can be used by Pods (filesystem & environment variables) and the underlying kubelet when pulling images

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
Type: Opaque
data:
   password: mmyWfoidfIuL==
   username: NyhdOKwB
```

Pod Secret

```
kind: Pod
metadata:
  name: secret-env-pod
spec:
 containers:
 - name: mycontainer
   image: redis
   env:
   - name: SECRET_USERNAME
       valueFrom:
         secretKeyRef:
           name: mysecret
           key: username
```

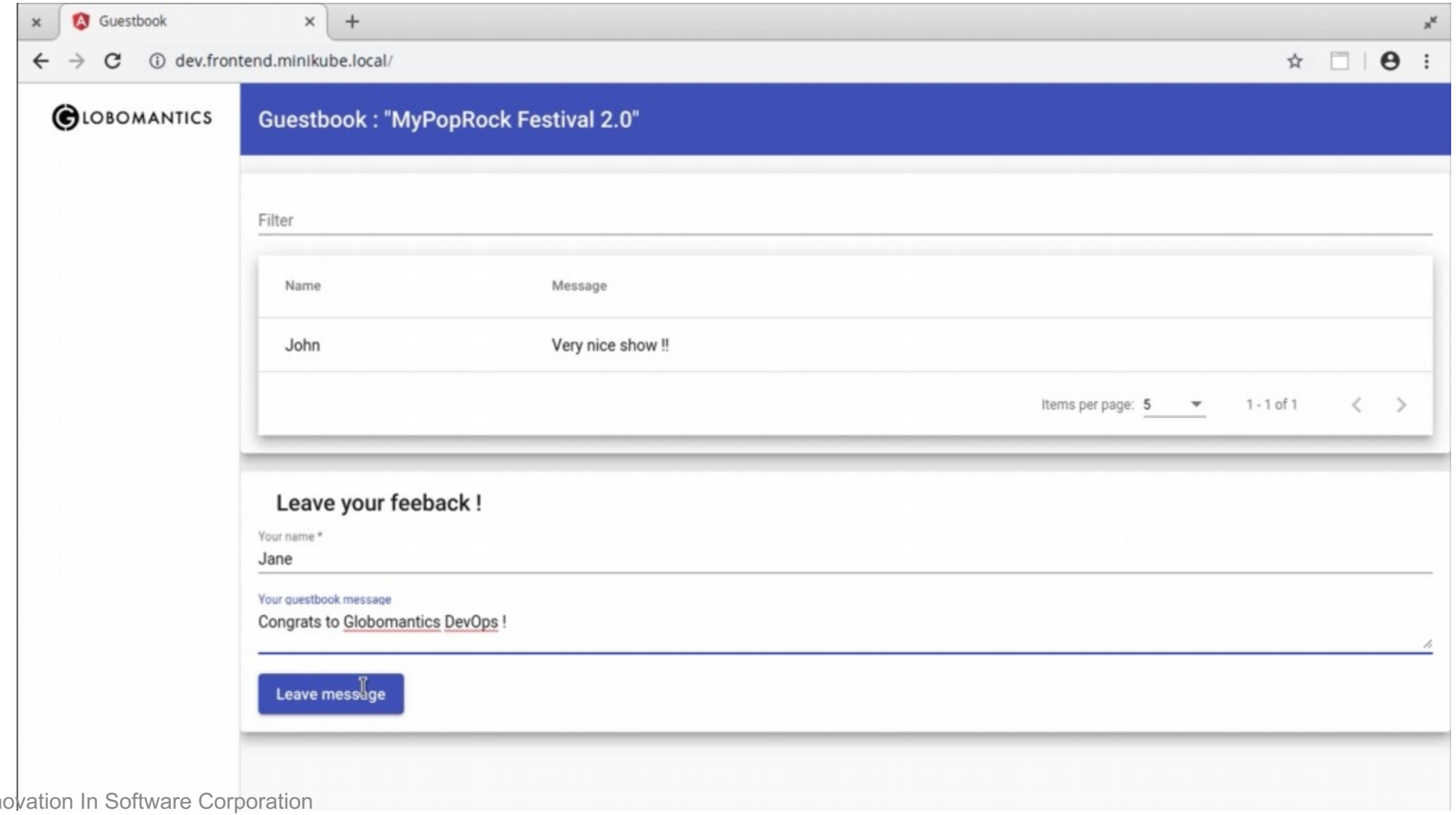
Volume Secret

```
spec:
  containers
  - name: mycontainer
    image: redis
    volumeMounts:
    - name: “secrets”
      mountPath: “/etc/my-secrets”
      readOnly: true
  volumes:
   - name: “secrets”
      secret:
        secretName: “mysecret”
```

Labs: ConfigMap & Secrets

Helm: Package management

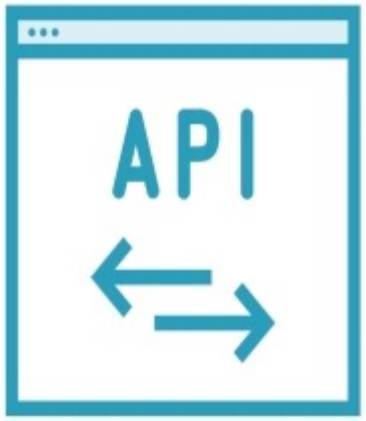
Helm sample application



Sample application components

- ✓ ConfigMap
- ✓ Pod
- ✓ Service
- ✓ Ingress

Frontend



- ✓ Secret
- ✓ Pod
- ✓ Service

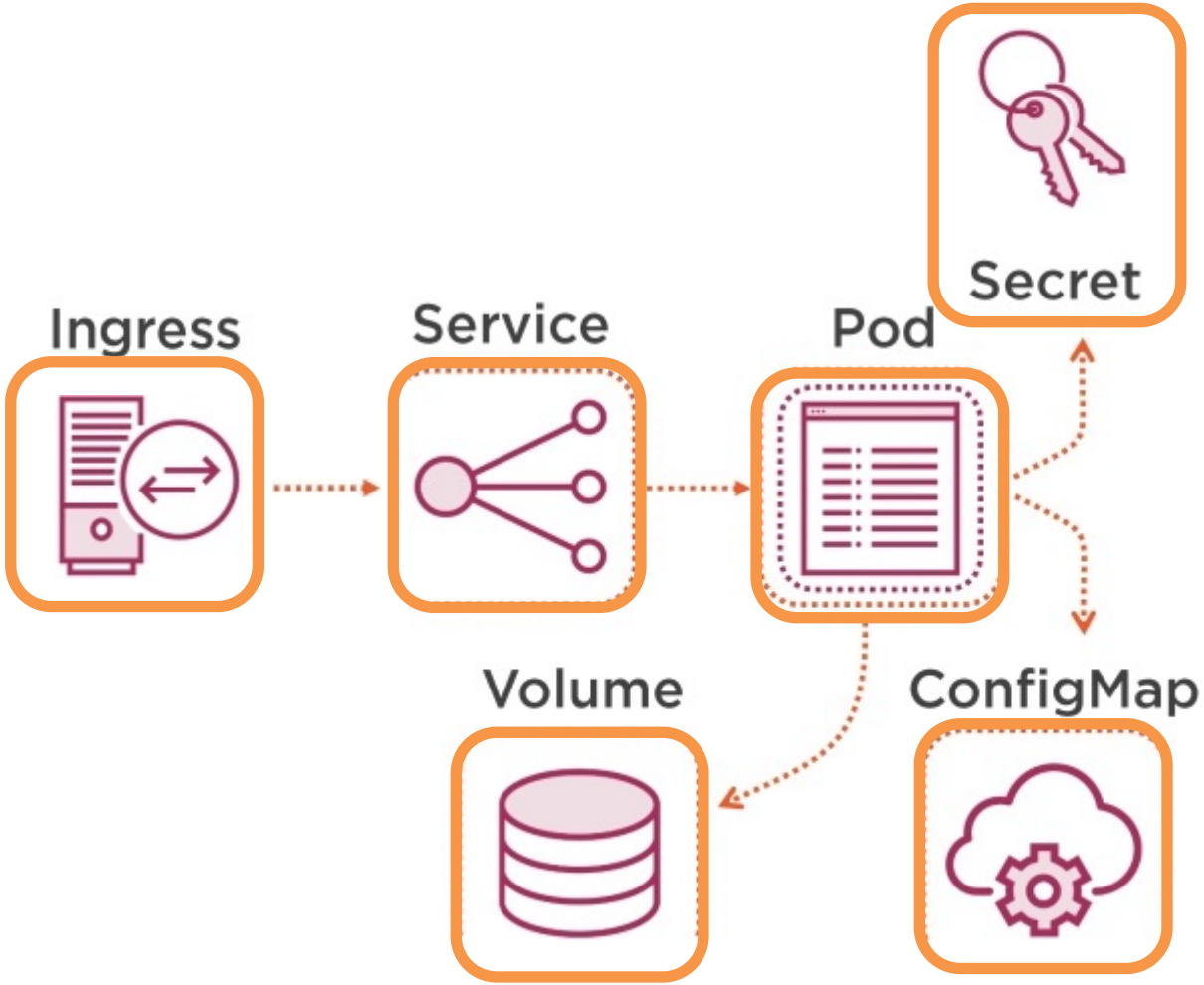
Backend API

- ✓ Secret
- ✓ PV
- ✓ PVC
- ✓ Pod
- ✓ Service

Database

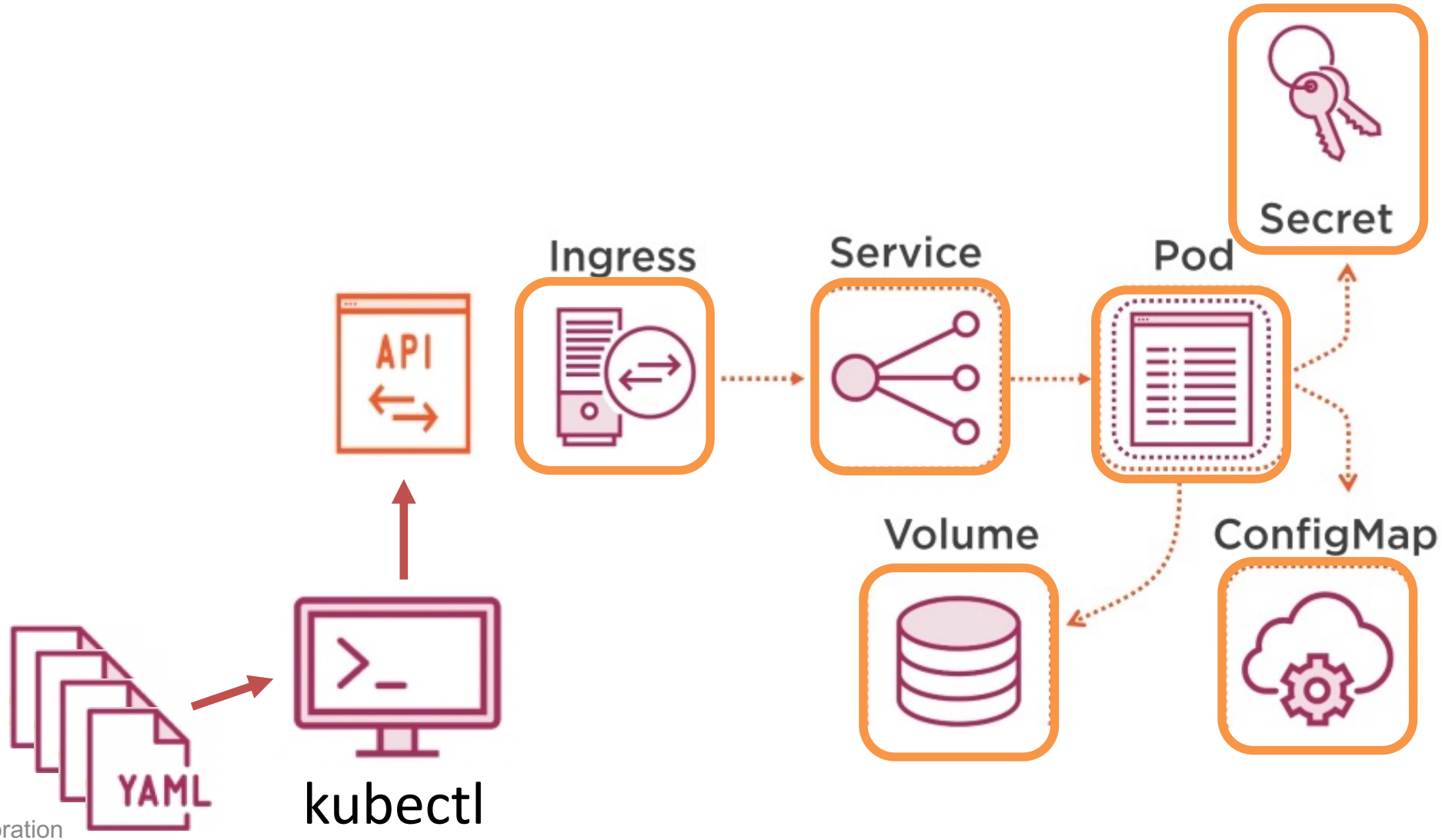
Native Kubernetes way

- Application
- Container
- Pod
- Service
- Ingress
- ConfigMap
- Secrets
- Volumes: PV, PVC, Storage



Native Kubernetes way

- Limitations
  - Packaging
  - Versioning



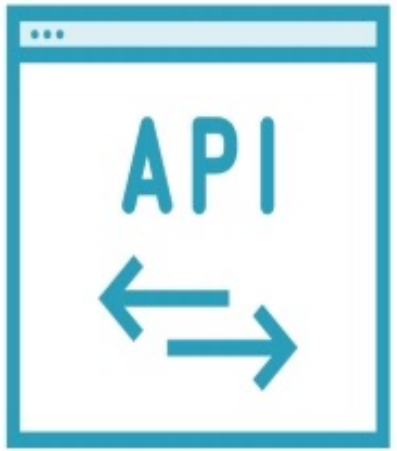
Guestbook: version 1

- ConfigMap
- Pod
- Service
- Ingress

Frontend

Guestbook: version 2

- ConfigMap
- Pod
- Service
- Ingress



- Secret
- Pod
- Service

- Secret
- PV
- PVC
- Pod
- Service

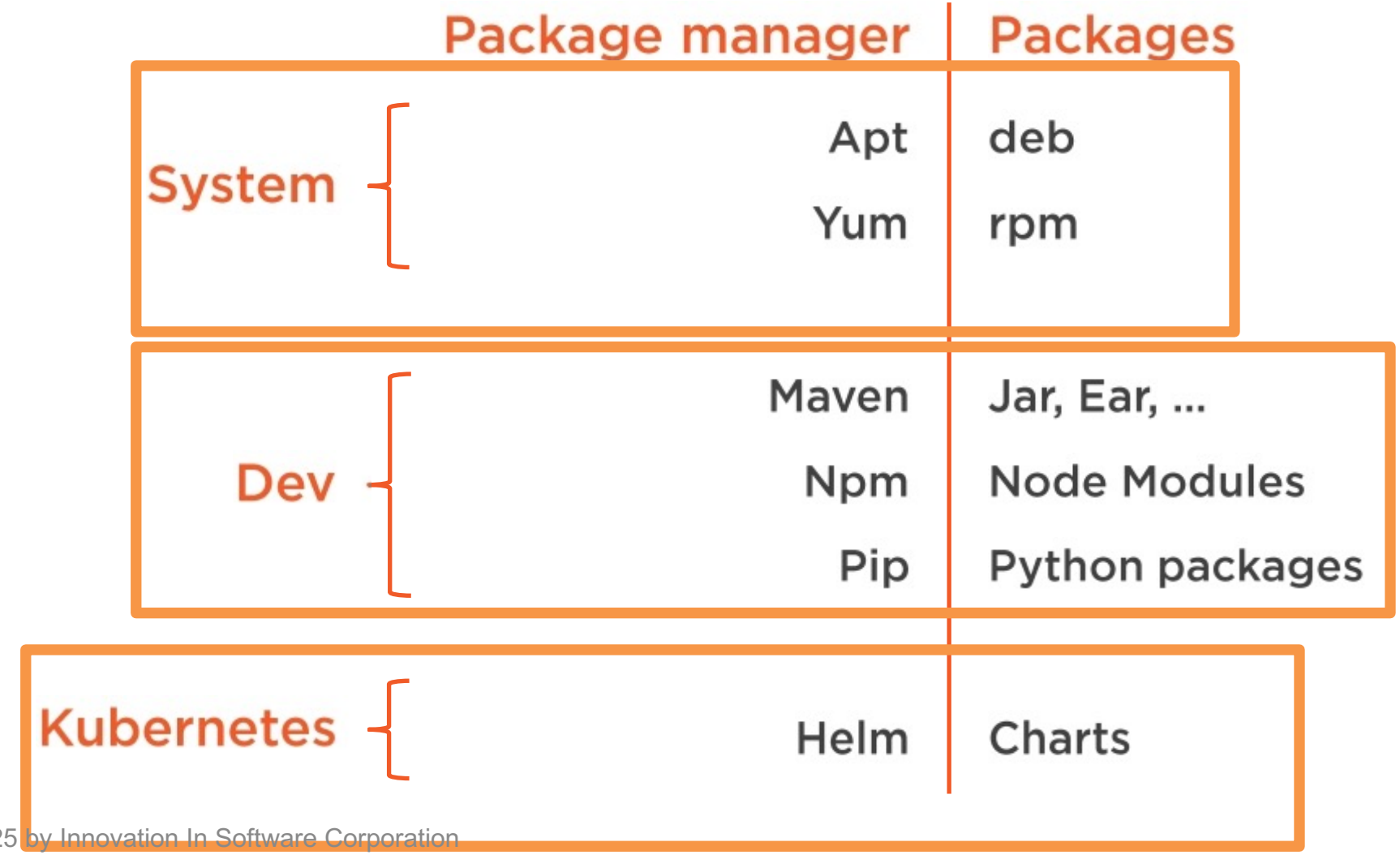
Frontend

Backend

Database

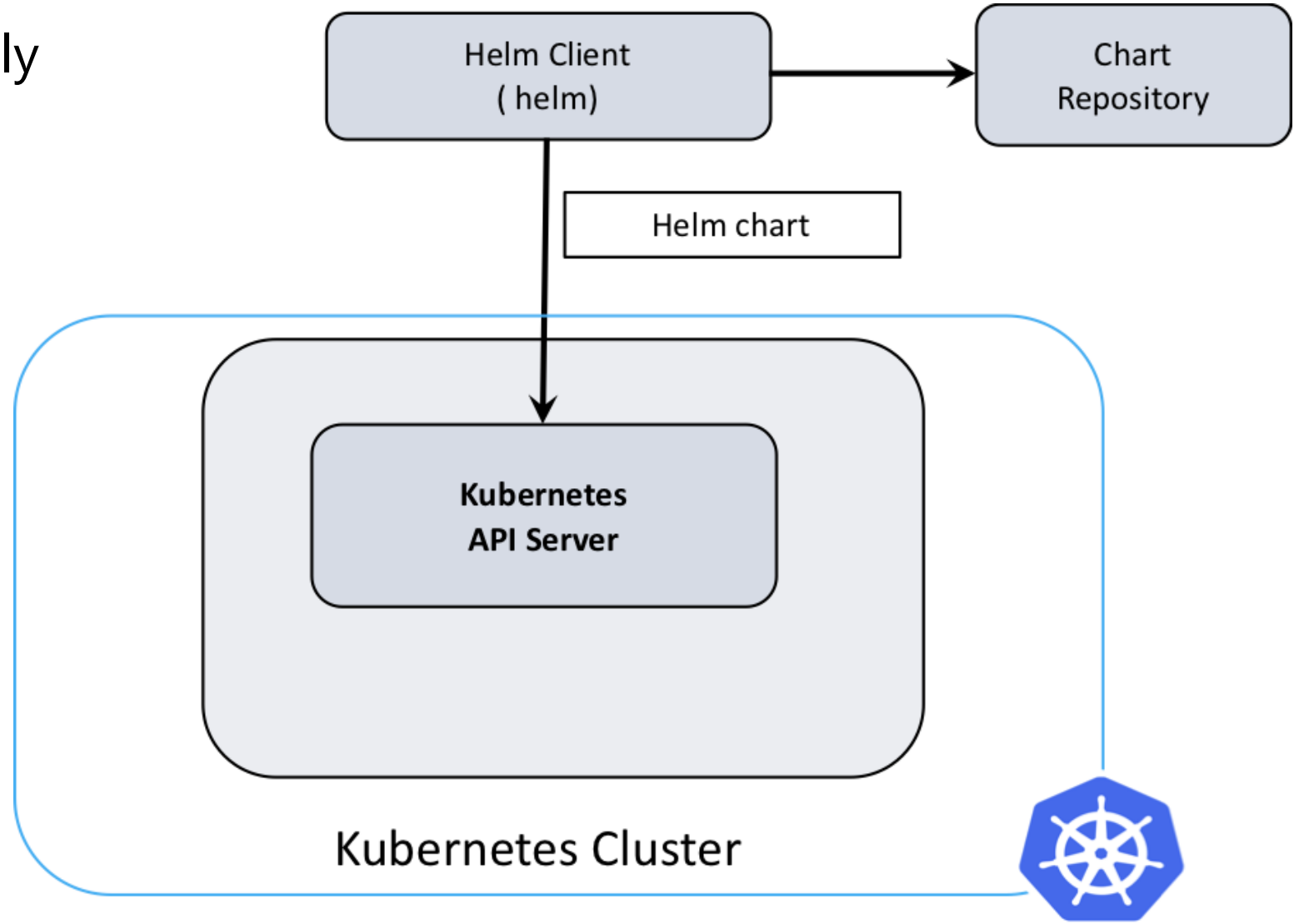
Painful to manage

Helm features



Helm architecture

- Helm:
  - develop charts locally
  - command-line



Helm features

Charts

Templates

Dependencies

Repositories

Helm Chart Structure

Helm Chart structure

- nginx-demo
  - Chart.yaml

- Chart properties
  - name
  - version
  - more..

Helm Chart structure

- nginx-demo
  - Chart.yaml
  - README.md

- Document chart
  - Overrides
  - Maintainer
  - Instructions

Helm Chart structure

- nginx-demo
  - Chart.yaml
  - README.md
- templates
  - deployment.yaml
  - ingress.yaml
  - service.yaml

- Kubernetes object definitions
  - customizable YAML templates

Helm Chart structure

- nginx-demo
  - Chart.yaml
  - README.md
- templates
  - deployment.yaml
  - ingress.yaml
  - service.yaml
- values.yaml

- Kubernetes object definitions
  - customizable YAML templates
    - Provide default values.

Helm Chart structure

- nginx-demo
  - Chart.yaml
  - README.md
- templates
  - deployment.yaml
  - ingress.yaml
  - service.yaml
  - NOTES.txt
- values.yaml

- Provide helpful output after installation
  - How to access application
  - Create user/pass

Helm Chart structure

- nginx-demo
  - Chart.yaml
  - README.md
- templates
  - deployment.yaml
  - ingress.yaml
  - service.yaml
  - NOTES.txt
  - tests
    - test-connection.yaml
- values.yaml

- You can create tests to confirm Chart works as expected.

Helm Chart structure

- nginx-demo
  - Chart.yaml
  - README.md
  - requirements.yaml
- templates
  - deployment.yaml
  - ingress.yaml
  - service.yaml
  - NOTES.txt
  - tests
    - test-connection.yaml
- values.yaml

- Define sub-charts and dependencies

Helm Chart.yaml

Chart.yaml

```yaml
apiVersion: v1
appVersion: "1.0"
description: A Helm chart for Kubernetes
name: nginx-demo
version: 0.1.0
```

Major,Minor,Patch (SemVer 2.0)

- App you are installing

Helm Chart.yaml

Chart.yaml

```
apiVersion: v1
appVersion: "1.0"
description: A Helm chart for Kubernetes
name: nginx-demo
version: 0.1.0
```

Major,Minor,Patch (SemVer 2.0)

- Version of Helm chart

Lab: Helm