Basics of Kuberentes Networking

Contacts

Péter Megyesi

megyesi@leannet.eu

twitter.com/M3gy0

linkedin.com/in/M3gy0

Dávid Szabó

szabo@leannet.eu

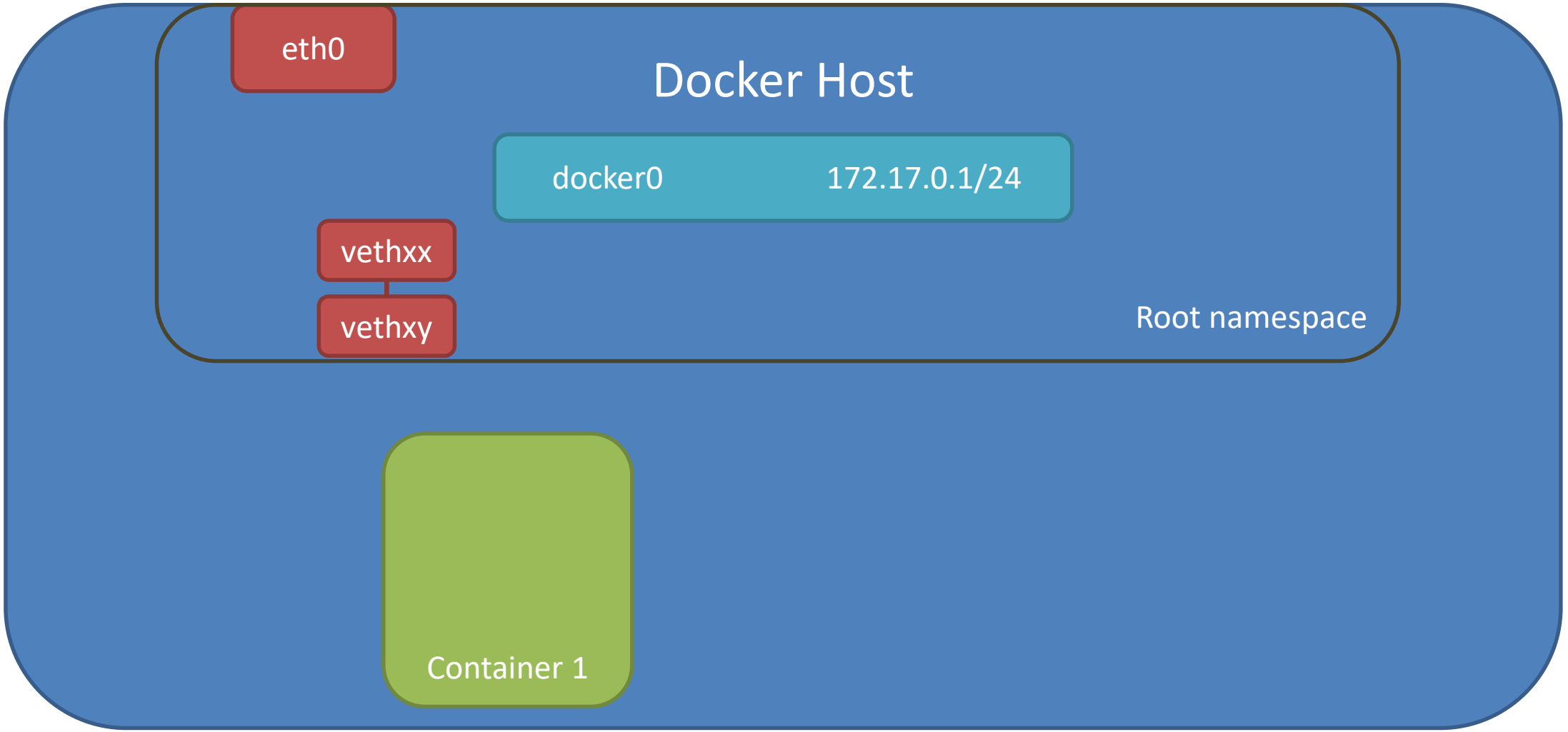
twitter.com/szabvid

linkedin.com/in/szabvid

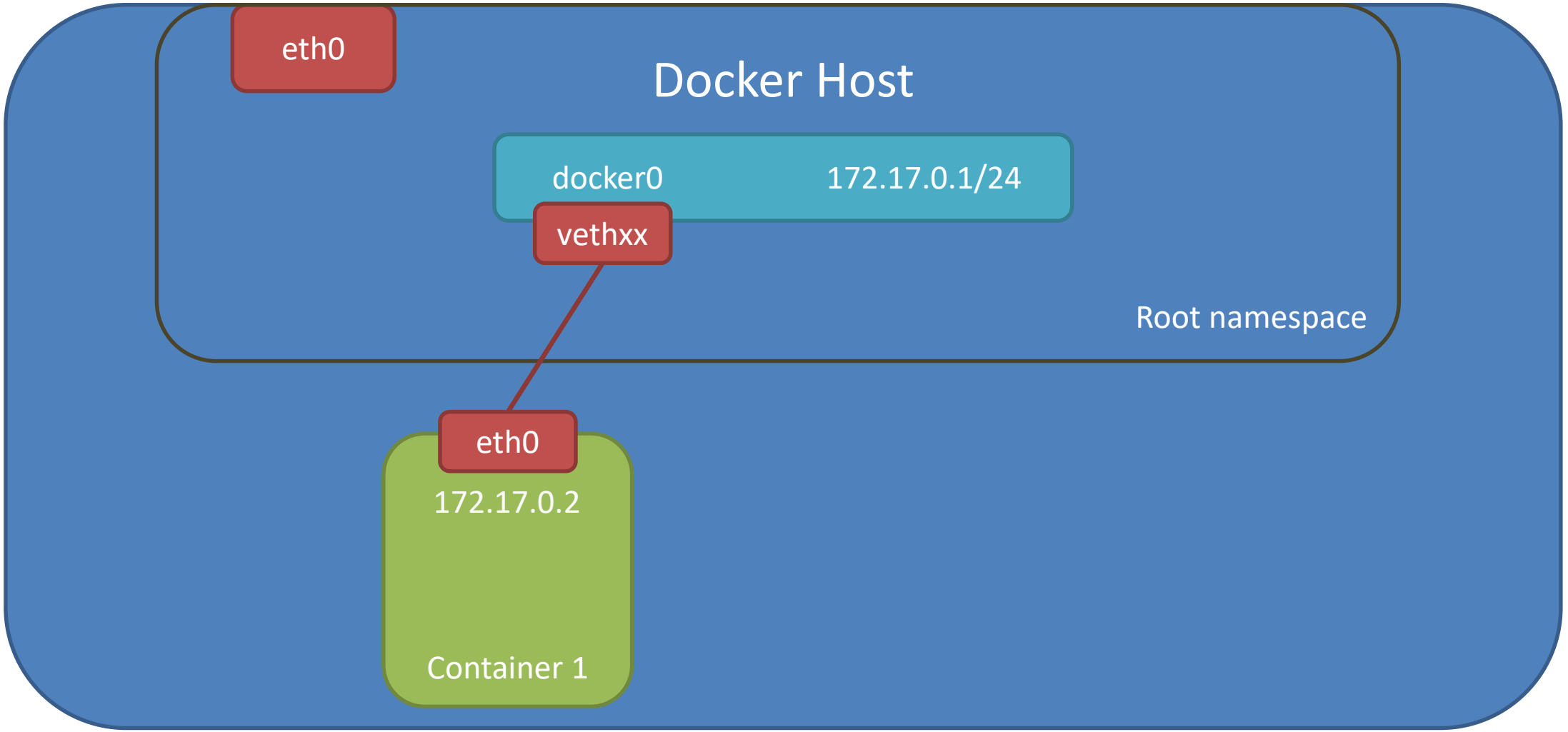
Course Outline

1. What is Kubernetes?
   - Components
   - Installation
2. Basics of Docker
   - Namespaces
   - Building and running Docker images
3. Pods and Deployments
   - Running basic workloads in Kubernetes
   - Scale, Update, Rollback
4. Advanced Pod configuration
   - Args, Envs, ConfigMaps, Secrets
   - Init- and sidecar containers
   - Scheduling and debugging
5. **Networking in Kubernetes**
   - **What are network plugins?**
   - **Service abstraction and ingress**
6. Persistent storage
   - Basics of storage: block vs. object vs. file system
   - StoragesClass, PVC, PV
7. Security
   - RBAC: Roles, ServiceAccounts, RoleBindings
   - Security context and network security policy
8. Advanced topics
   - Helm
   - Custom resources and operators

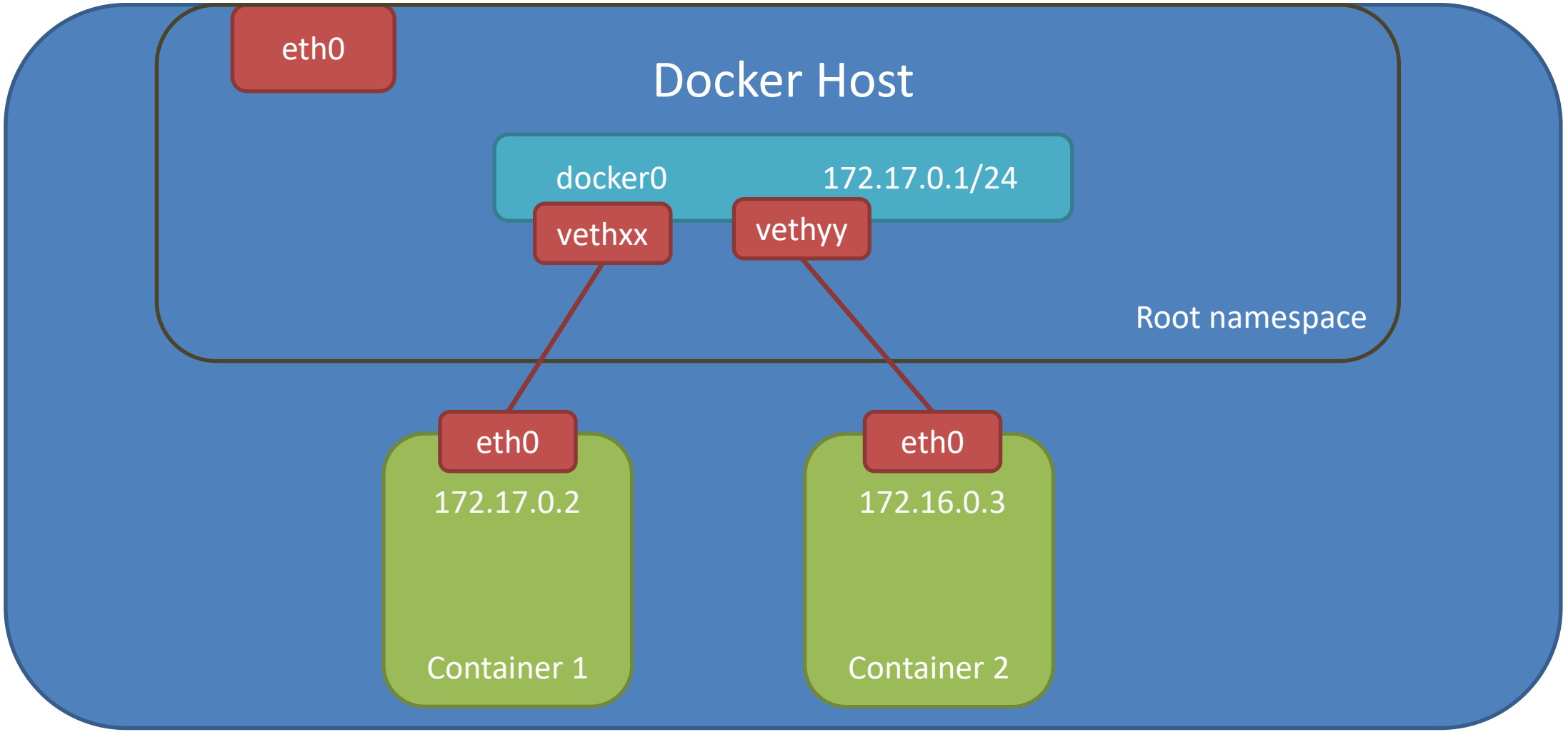
The Docker Model



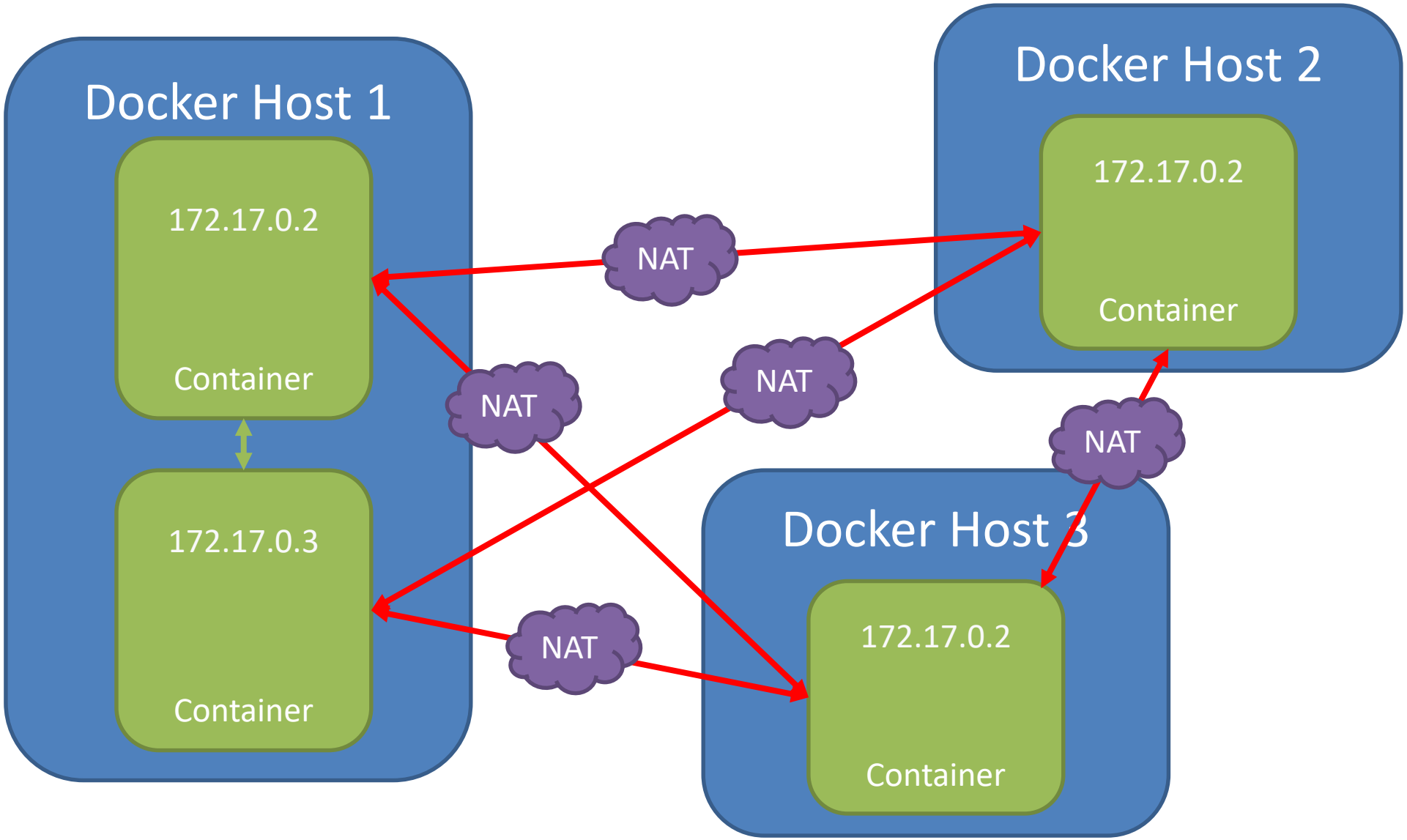
The Docker Model



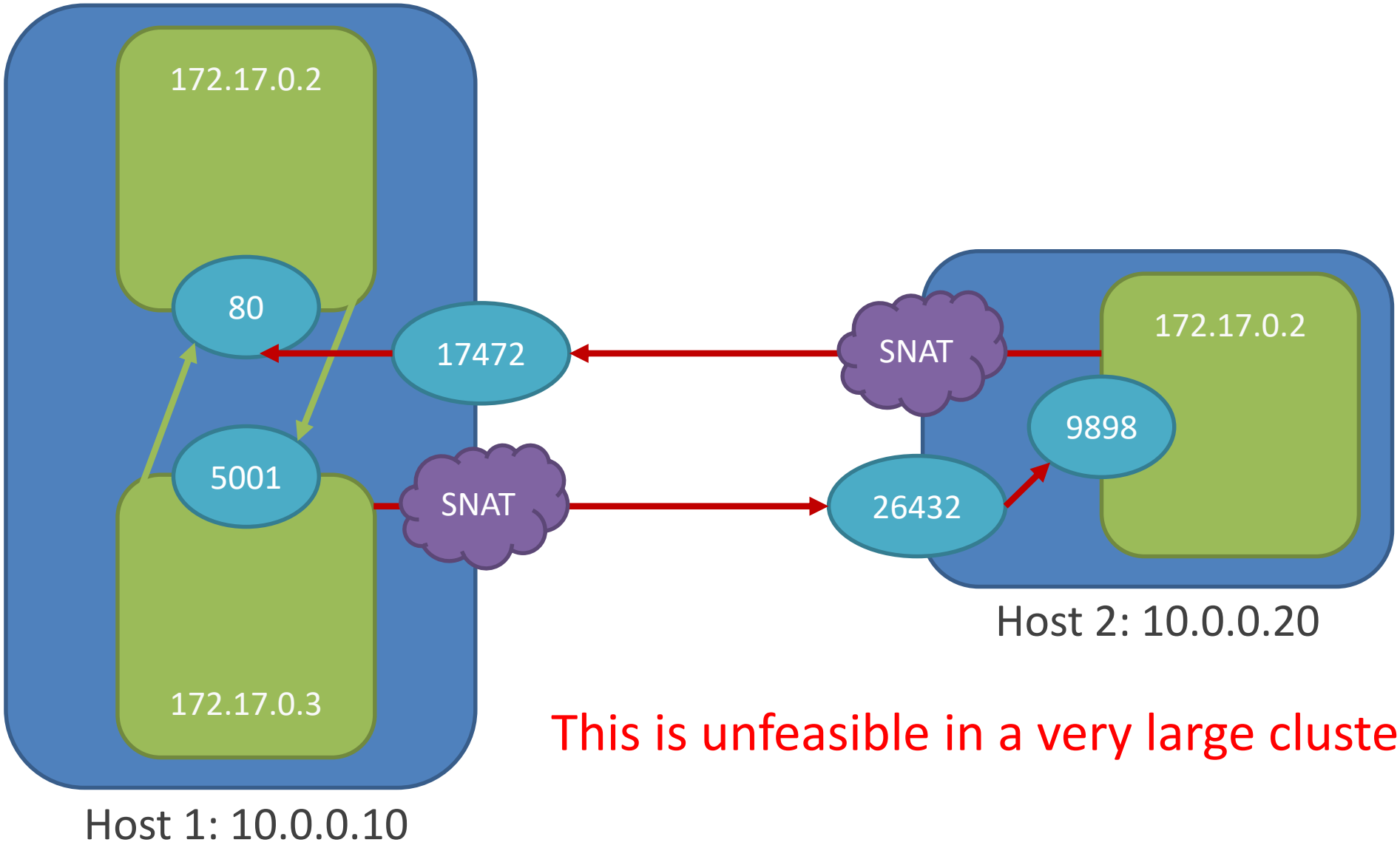
The Docker Model



The Docker Model

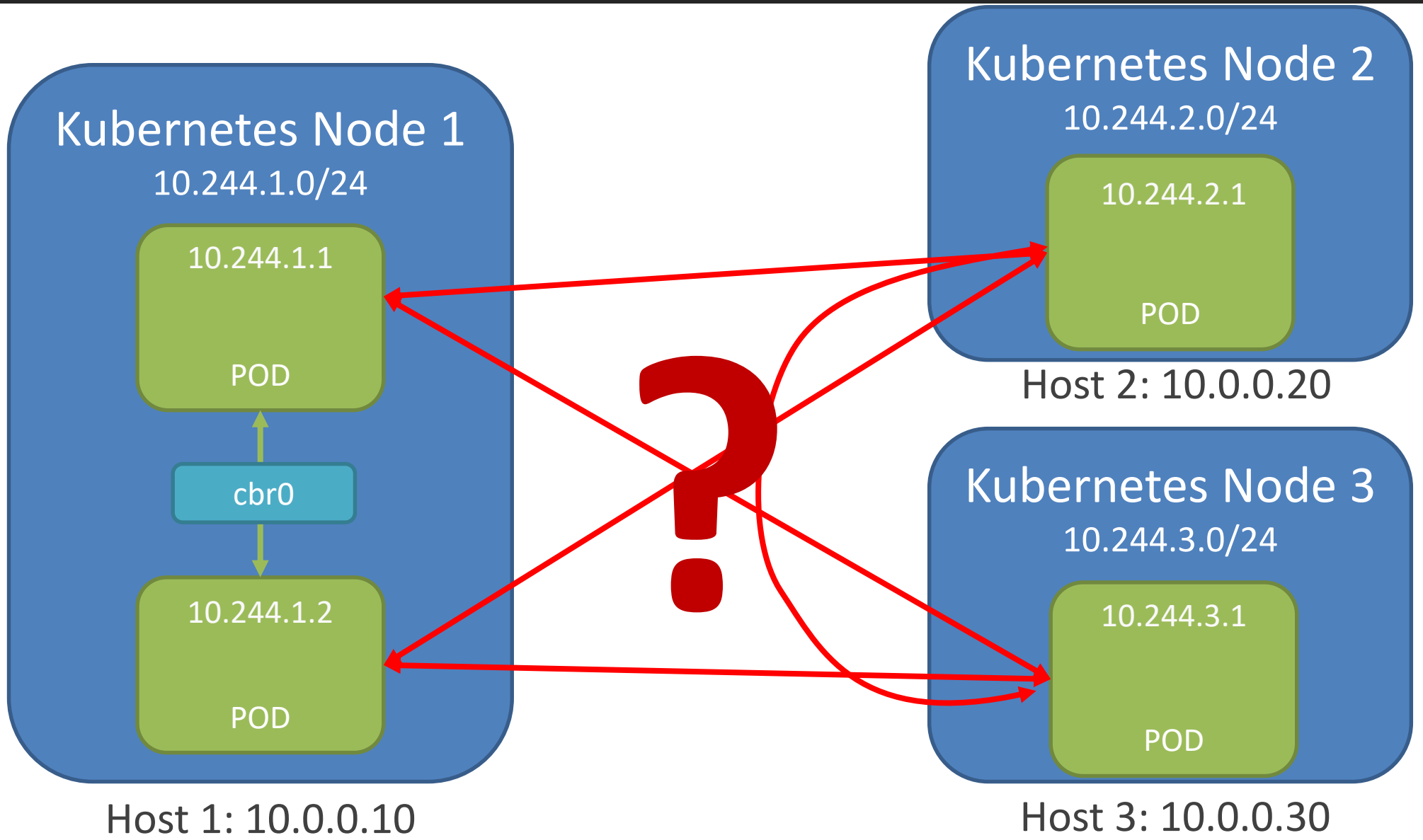


Docker Host Ports



This is unfeasible in a very large cluster!

The Kubernetes Model – The IP per POD Model



Networking in Kubernetes

Pod-to-Pod communication

- Each Pod in a Kubernetes cluster is assigned an IP in a flat shared networking namespace
- Pods on any node can communicate with all pods on all nodes without NAT
- Agents on a node (e.g. kubelet) can communicate with all pods on that node
  - Pods in the host network of a node can communicate with all pods on all nodes without NAT

Pod-to-Service communication

- Requests to the Service IPs are intercepted by a Kube-proxy process running on all hosts
- Kube-proxy is then responsible for routing to the correct Pod

External-to-Internal communication

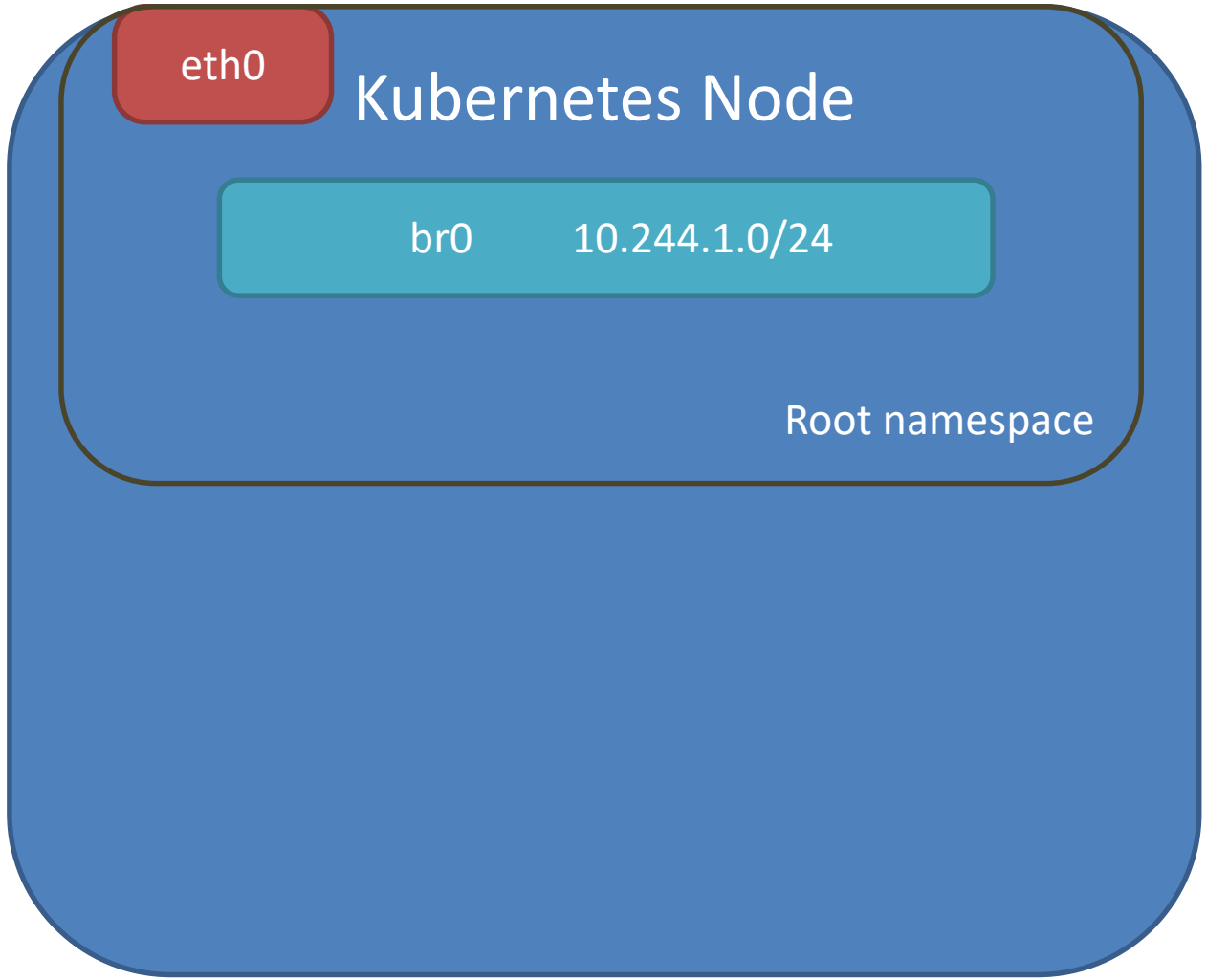
- Node port are can be assigned to a service on every Kuberentes host
- Public IPs can be implemented by configuring external Load Balancers which target all nodes in the cluster
- Once traffic arrives at a node, it is routed to the correct Service backends by Kube-proxy

The Container Network Interface

CNI in Kubernetes

Script / binary placed on every host

- Kubelet calls it with the right environmental variables and STDIN parameters



Example for configuration

```
{
    "cniVersion": "0.2.0",
    "name": "myplugin",
    "type": "leanplugin",
    "bridge": "br0",
    "isGateway": true,
    "ipam": {
        "type": "host-local",
        "subnet": "10.244.1.0/24",
        "gateway": "10.244.1.1",
        "routes": [
            { "dst": "1.1.1.1/32", "gw":"10.244.1.254"}
        ]
    }
}
```

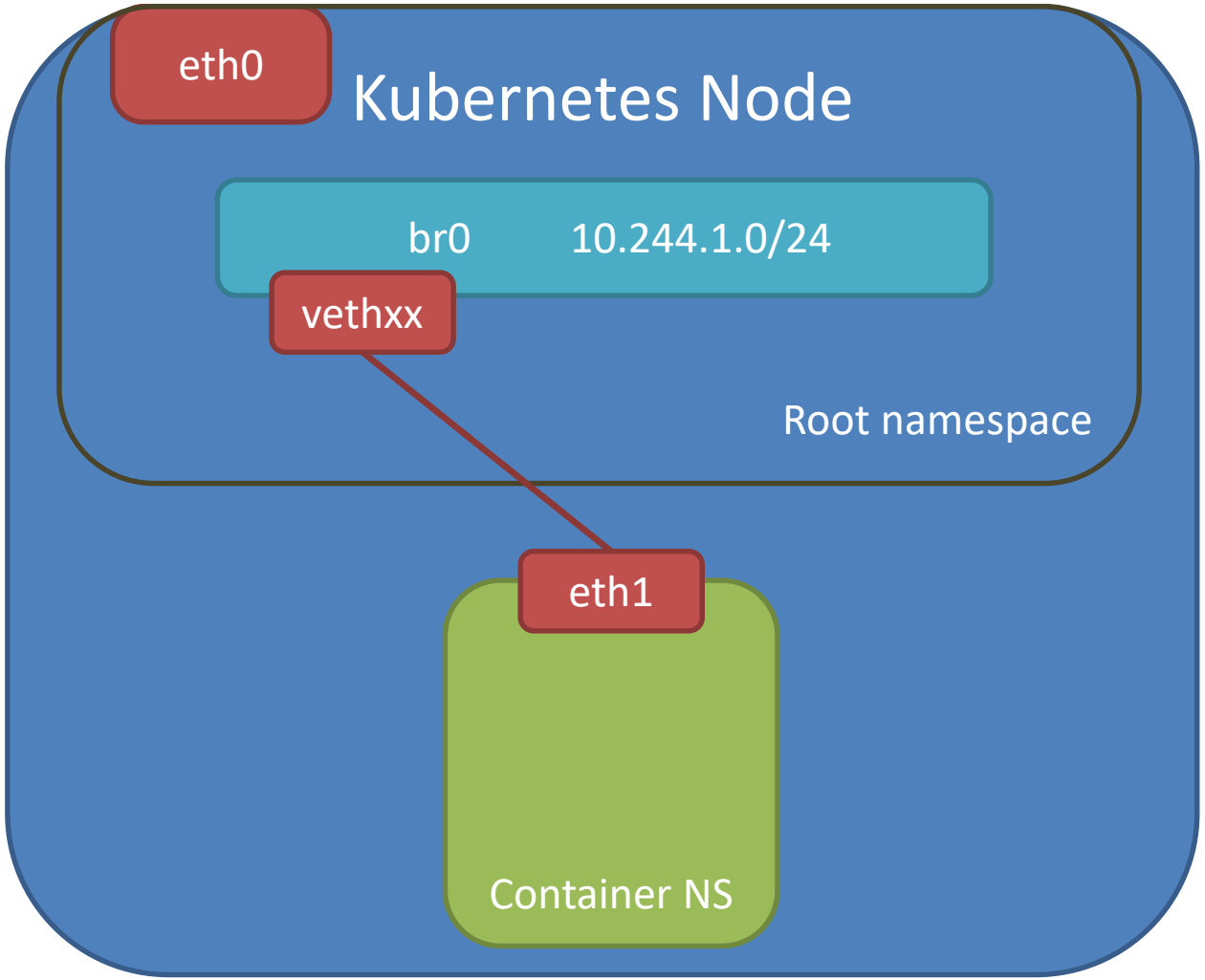
CNI in Kubernetes

Script / binary placed on every host

- Kubelet calls it with the right environmental variables and STDIN parameters

Example environment variables

- CNI_command: add or delete
- CNI_netns: /proc/<PID>/ns/net
- CNI_ifname: eth1
- CNI_path: /opt/bin/cni
- CNI_containerid
- K8S_pod_name
- K8S_pod_namespace



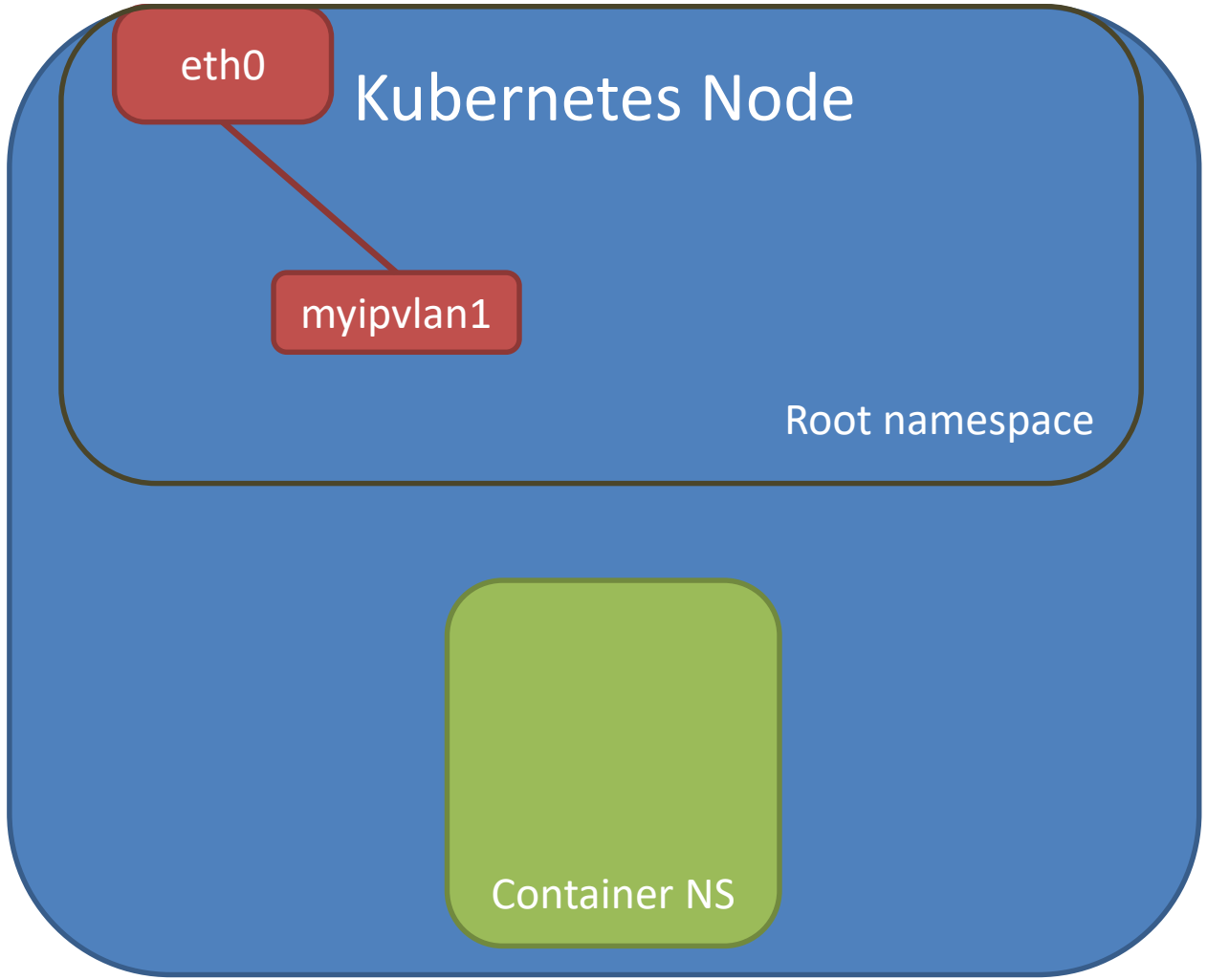
CNI Without LinuxBridge

Create IPVLAN subinterface

- ip link add myipvlan1 link eth0 type ipvlan mode l2

Put it into container namespace

- ip link set myipvlan1 netns $CNI_NETNS



CNI Without LinuxBridge

Create IPVLAN subinterface

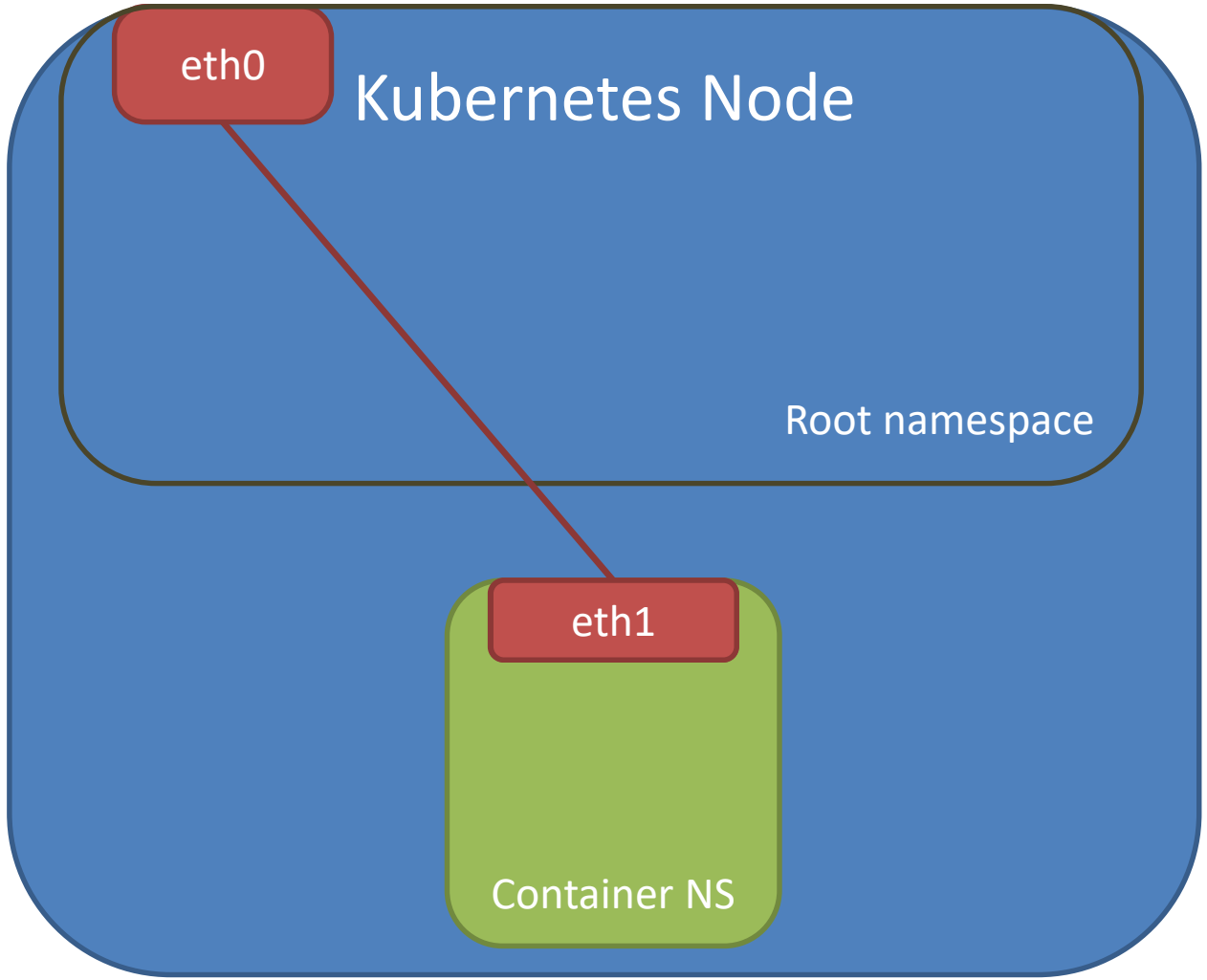
- ip link add myipvlan1 link eth0 type ipvlan mode l2

Put it into container namespace

- ip link set myipvlan1 netns $CNI_NETNS

Rename the interface

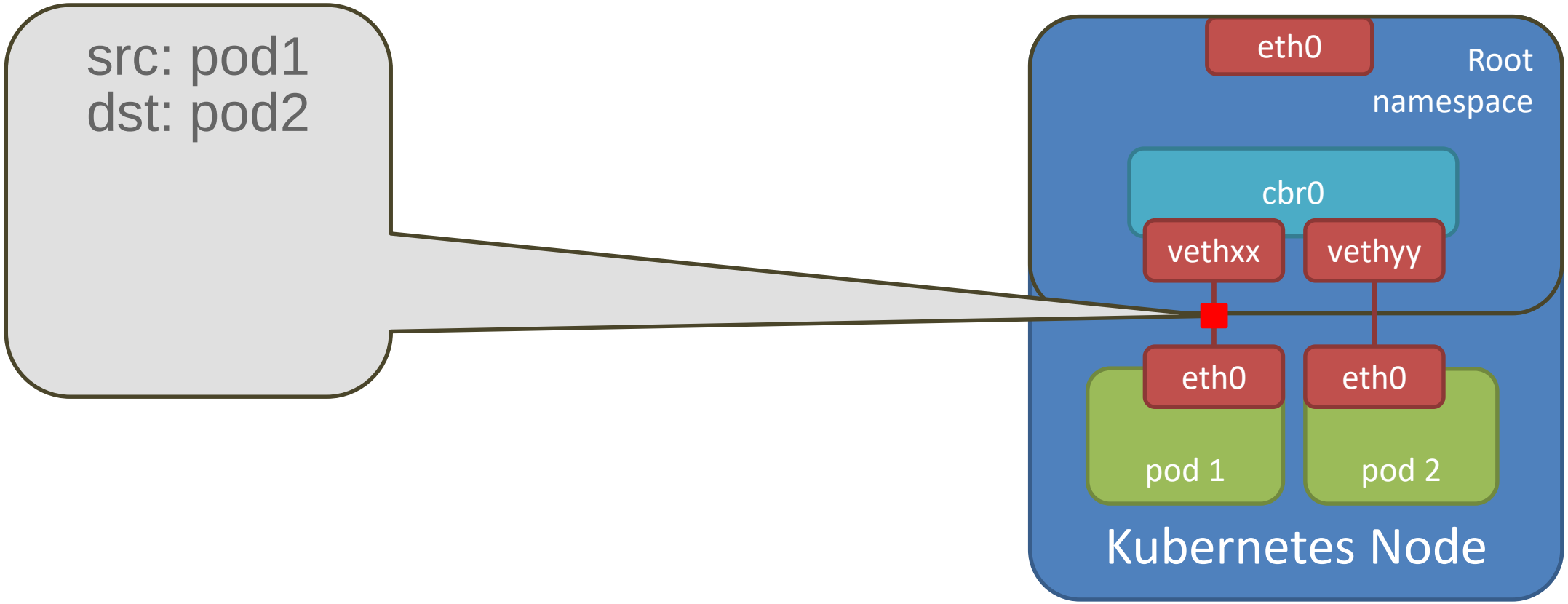
- ip netns $CNI_NETNS exec ip link set dev myipvlan1 name $CNI_IFNAME



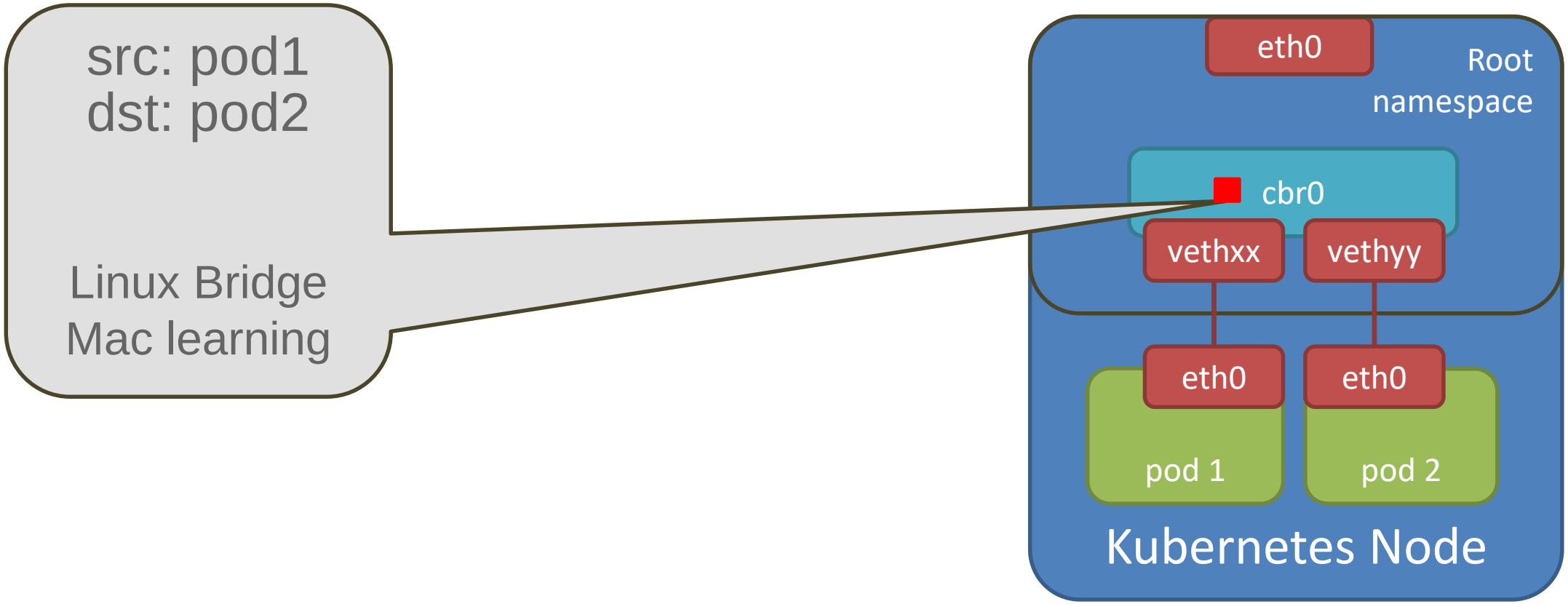
Cluster Networking in Kubernetes

Credit for the great visualization of packet lives:
Tim Hockin and Michael Rubin
„The ins and outs of networking in Google Container Engine and Kubernetes”
@Google Cloud Next '17

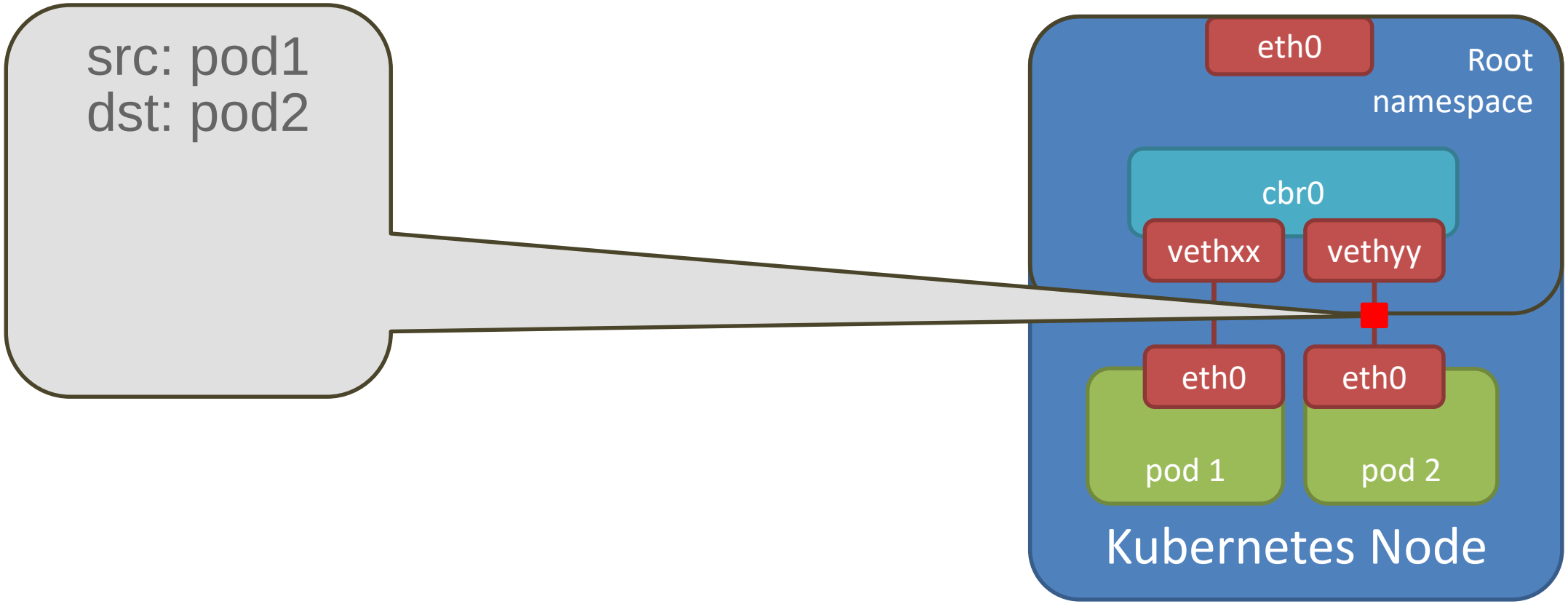
Life of a packet: pod-to-pod, same node



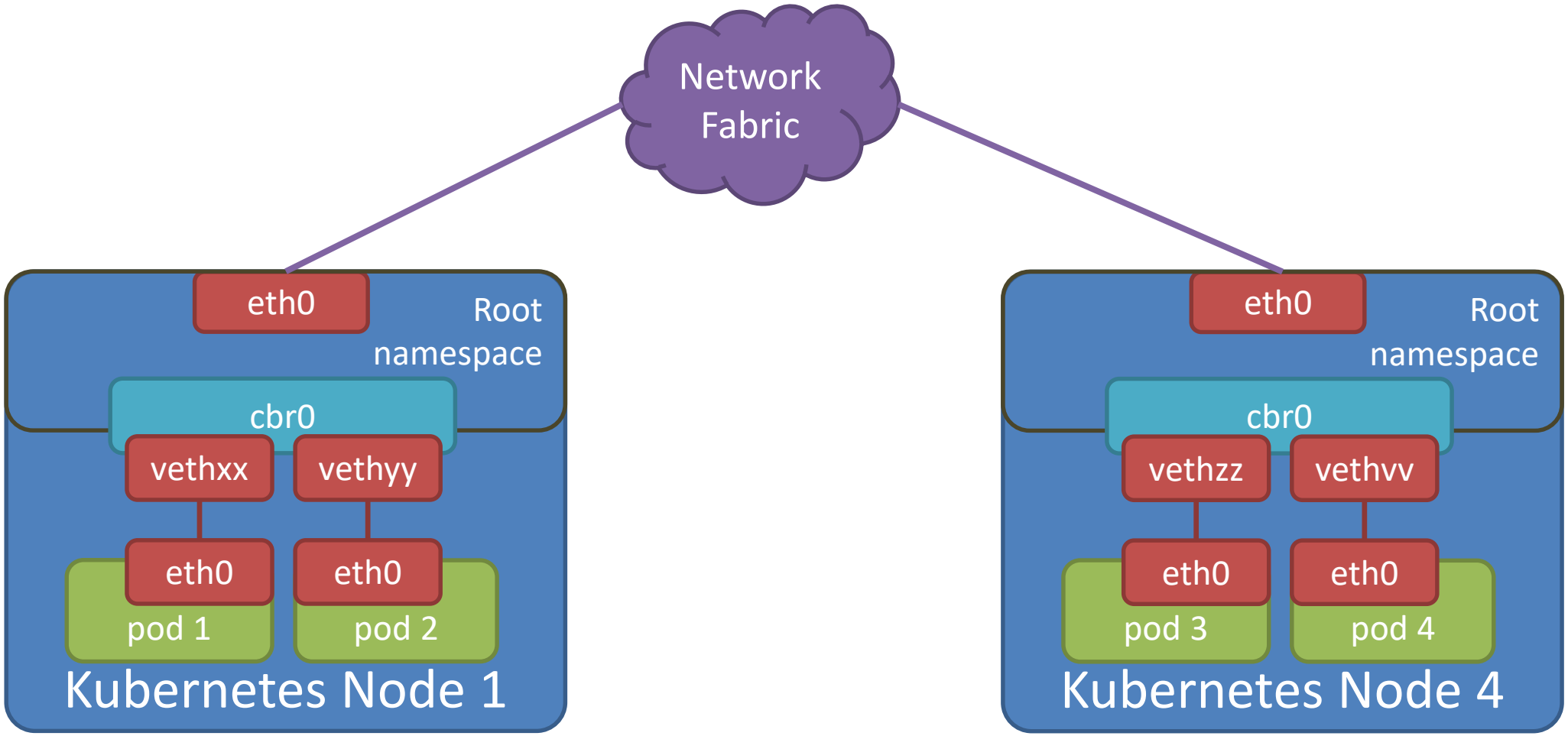
Life of a packet: pod-to-pod, same node



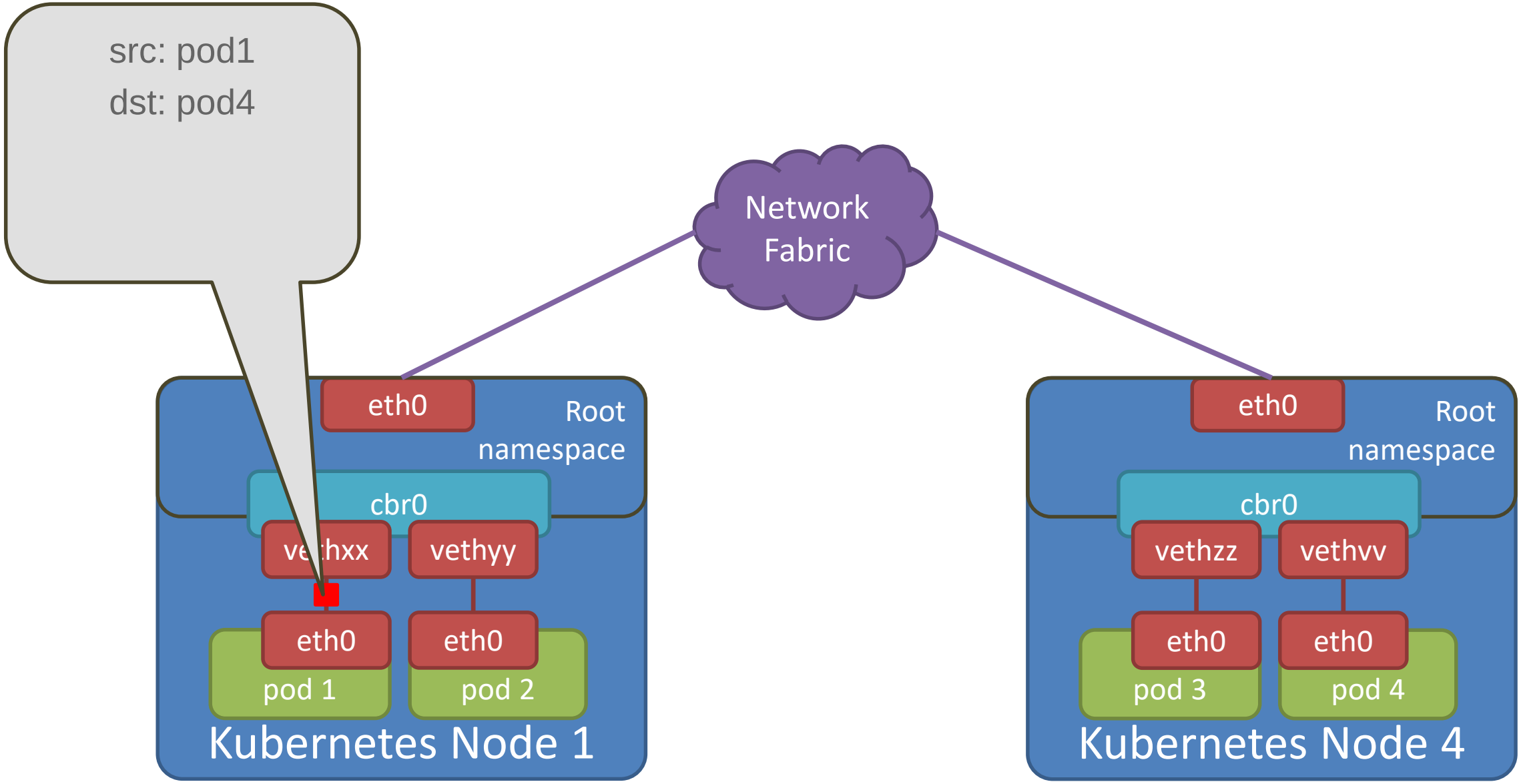
Life of a packet: pod-to-pod, same node



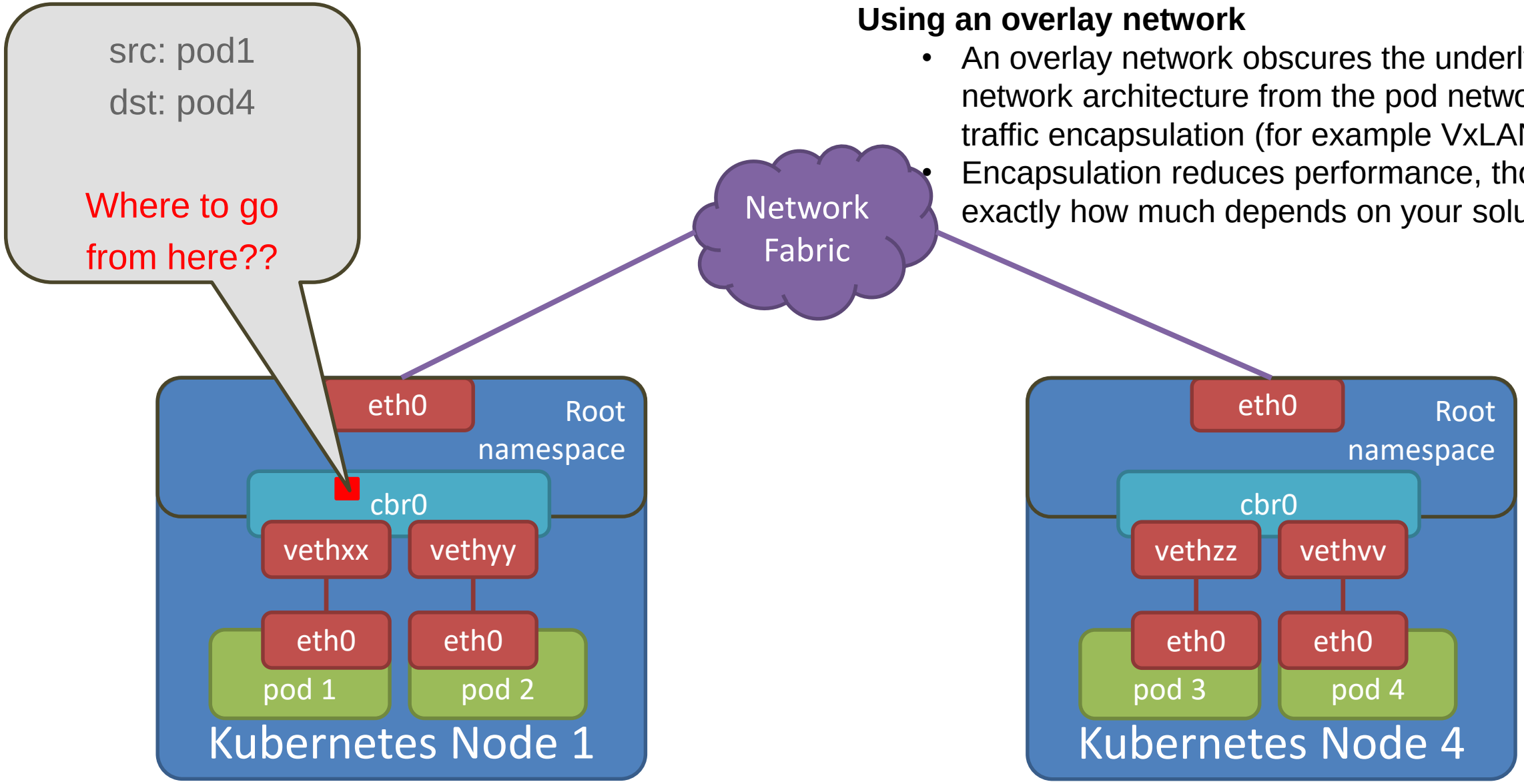
Life of a packet: pod-to-pod, between nodes



Life of a packet: pod-to-pod, between nodes



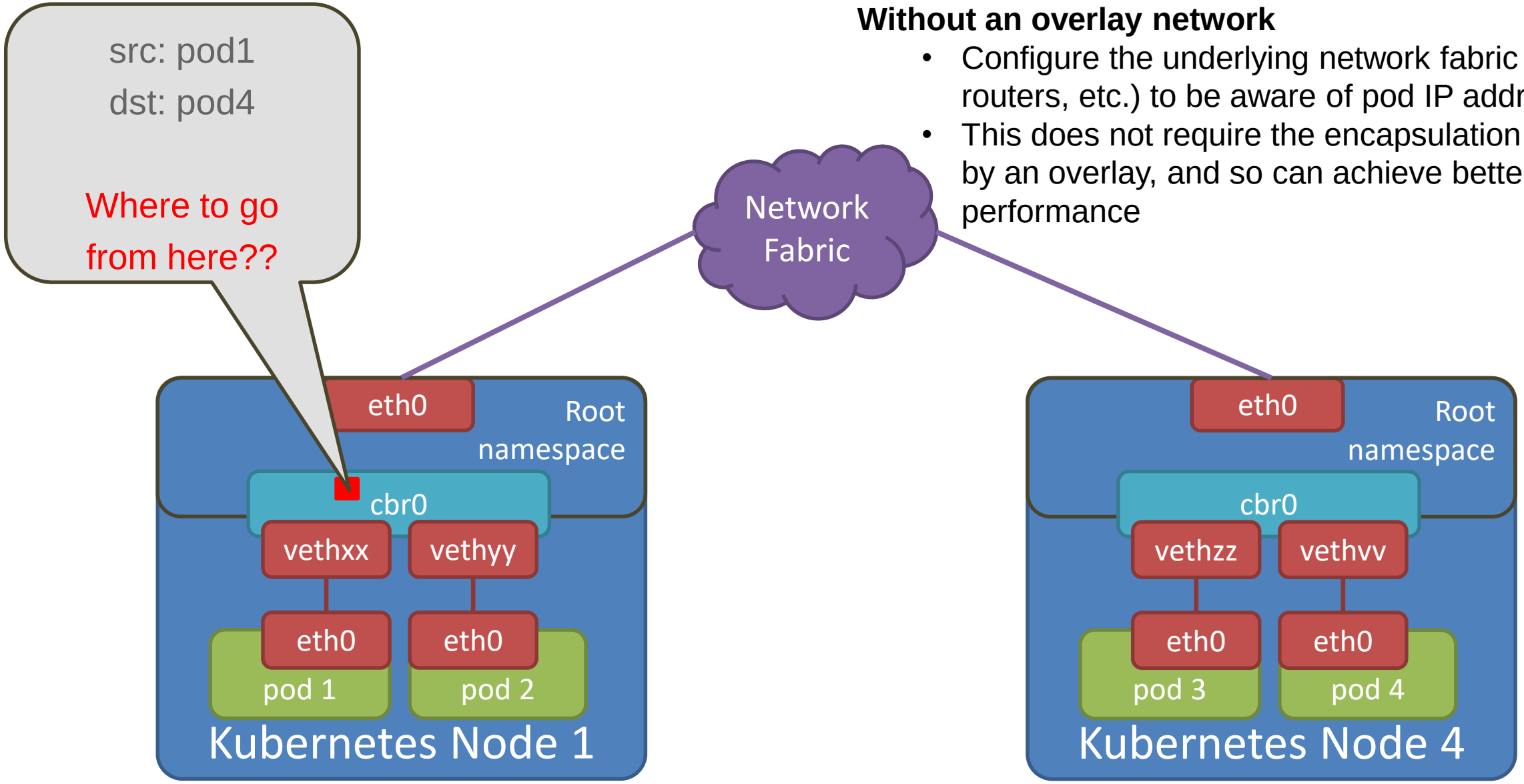
Life of a packet: pod-to-pod, between nodes



Using an overlay network

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

Life of a packet: pod-to-pod, between nodes



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

Kubernetes Cluster Networking Plugins

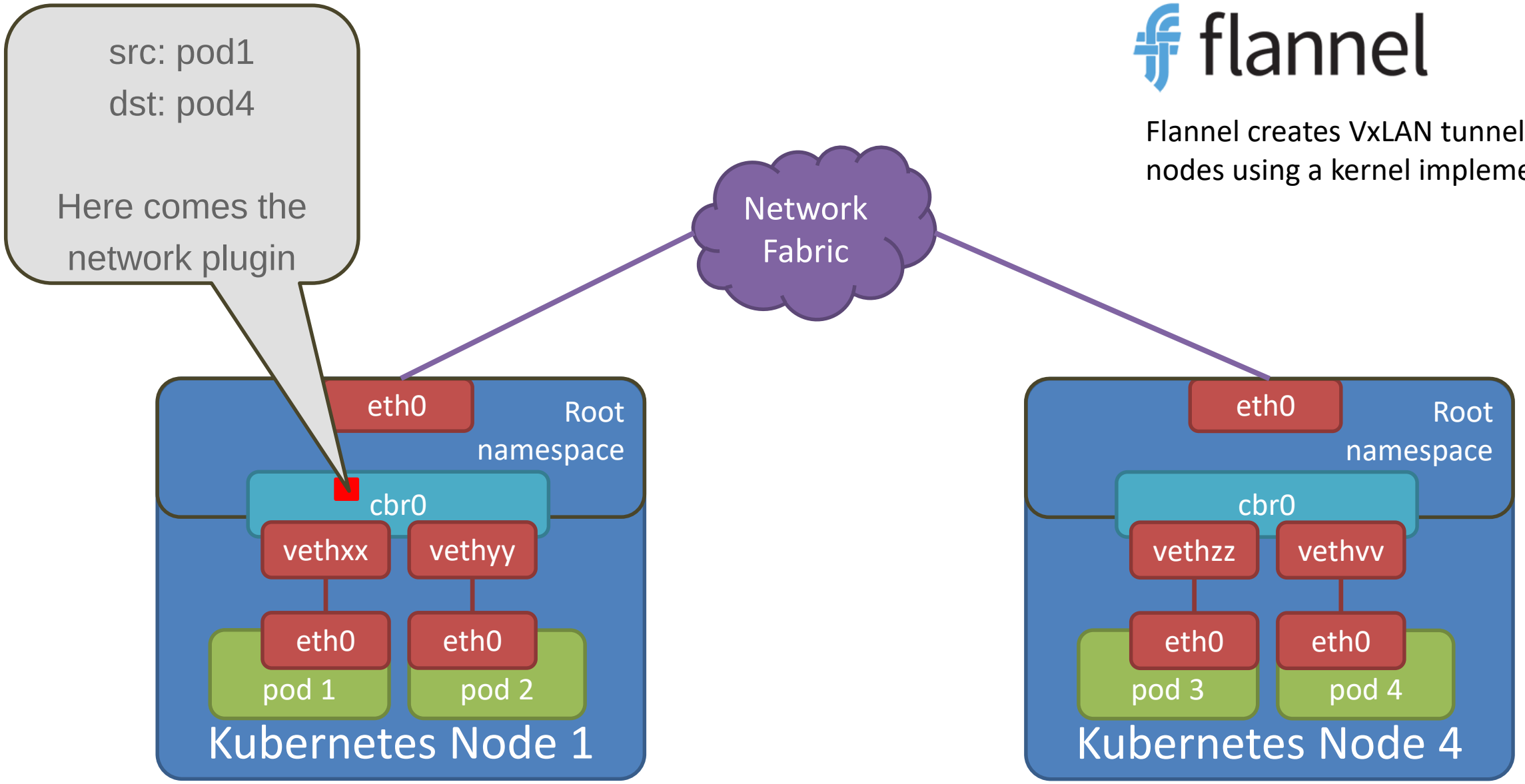
Public clouds which supports Kuberentes program this into the fabric

- E.g. in Google Container Engine: “everything to 10.1.1.0/24, send to this VM”

In other cases we need to use an external plugin

- Flannel
- Calico
- Canal (Flannel + Calico)
- Weave
- Cilium (uses eBPF)
- Contiv (by Cisco, uses VPP switch)
- CNI-Genie (by Huawei, CNCF sandbox project)
- Antrea (by VMware, uses Open vSwitch)
- https://kubernetes.io/docs/concepts/cluster-administration/networking/

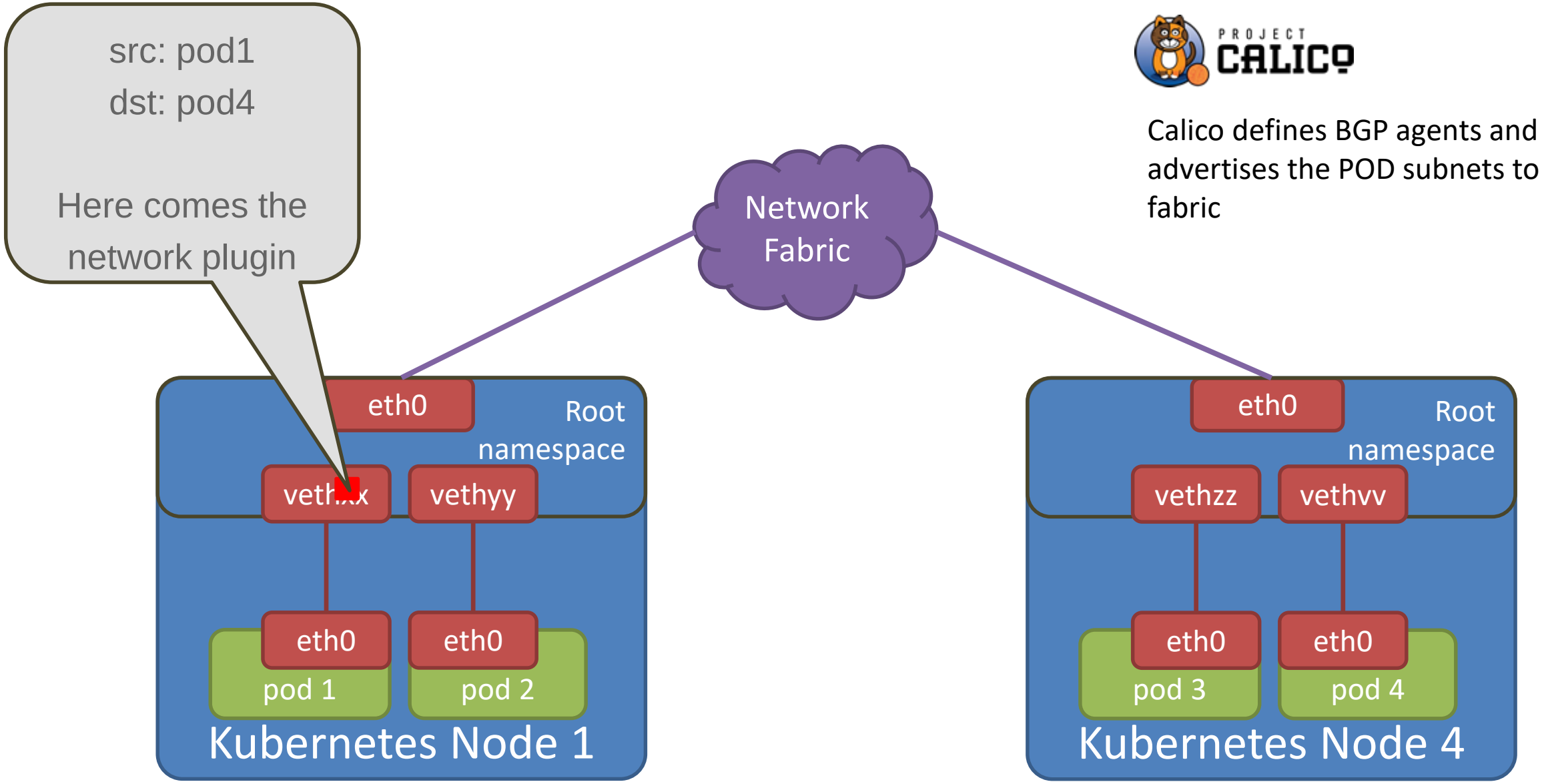
Life of a packet: pod-to-pod, between nodes



flannel

Flannel creates VxLAN tunnels between nodes using a kernel implementation

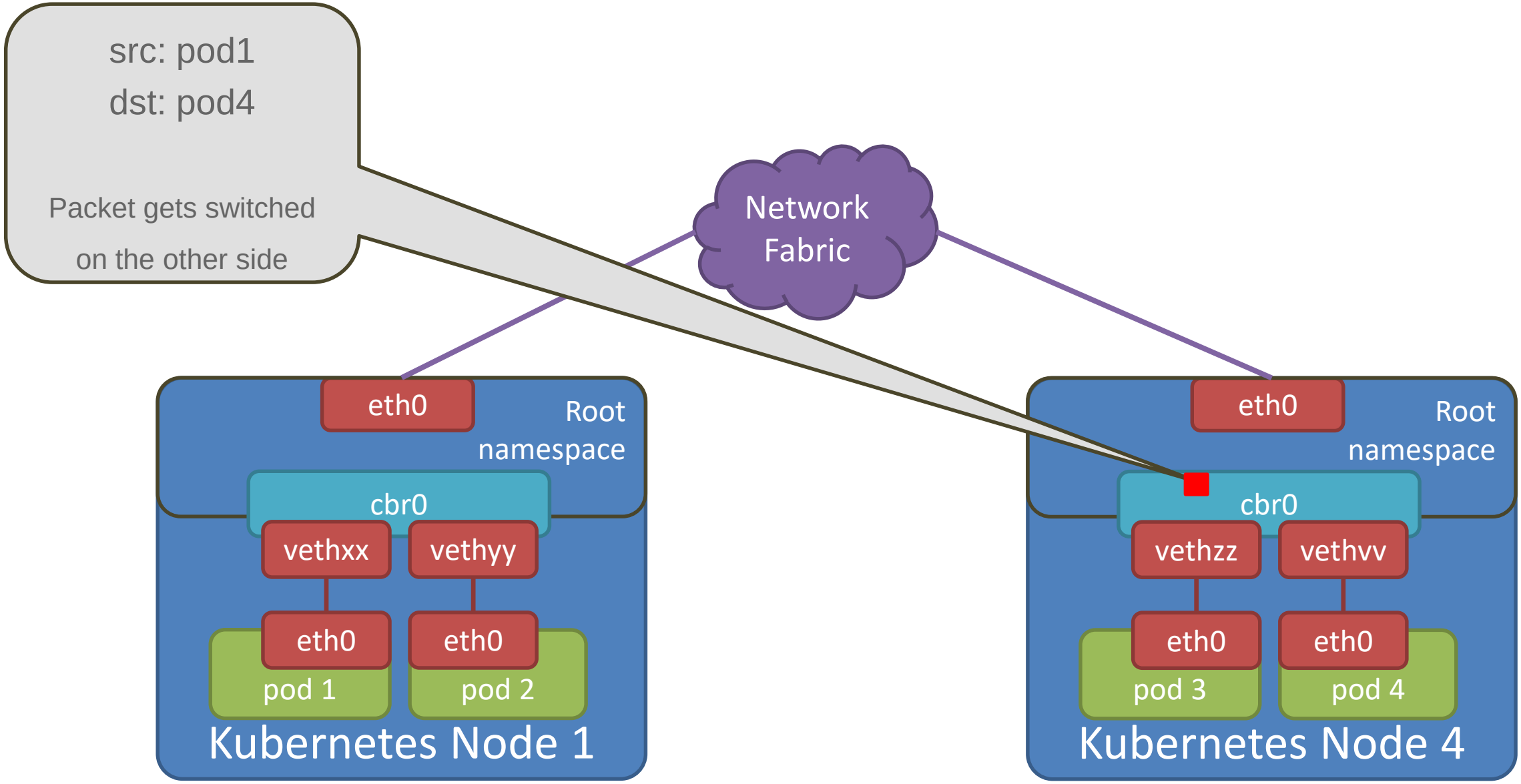
Life of a packet: pod-to-pod, between nodes



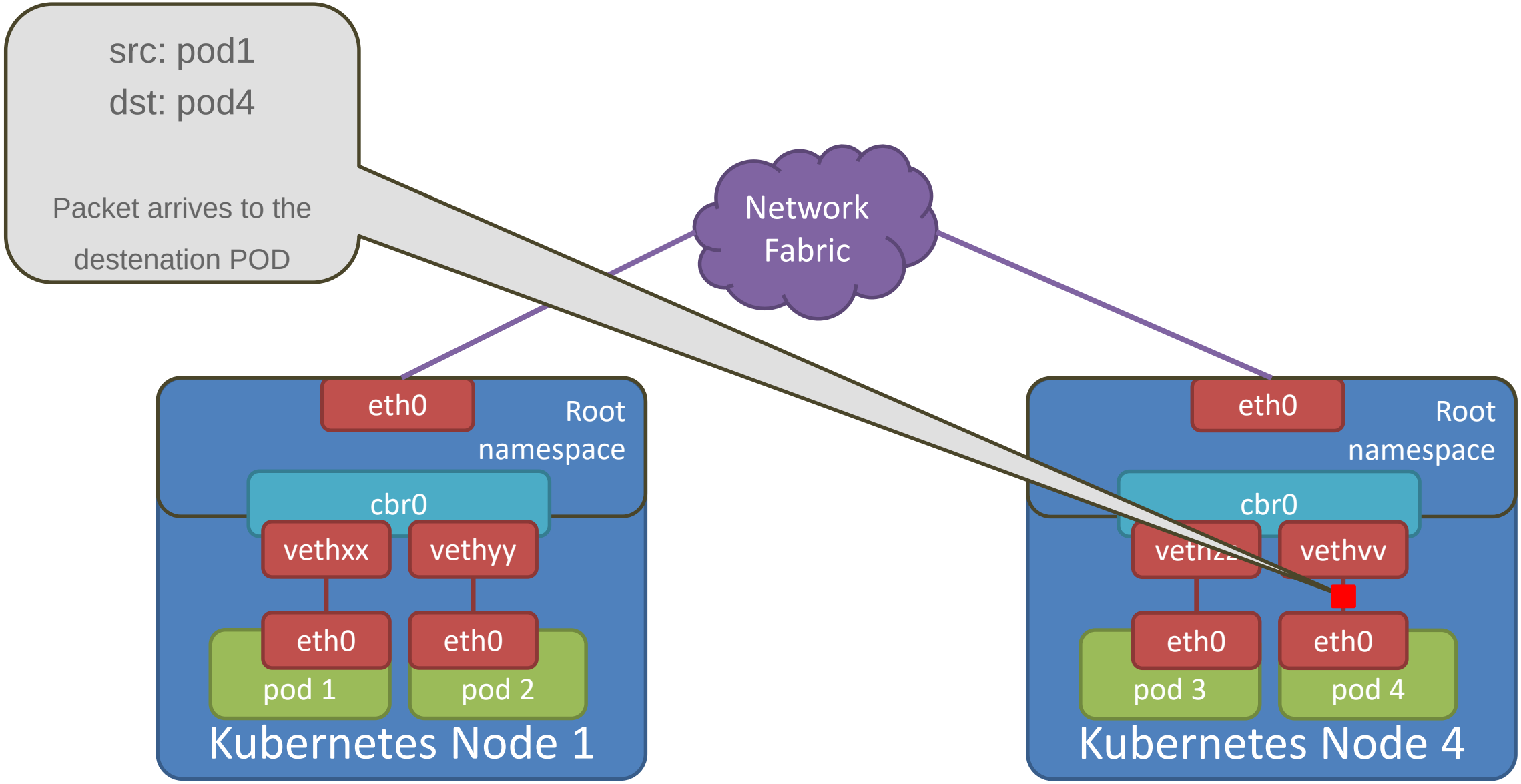
PROJECT CALICO

Calico defines BGP agents and advertises the POD subnets to the fabric

Life of a packet: pod-to-pod, between nodes



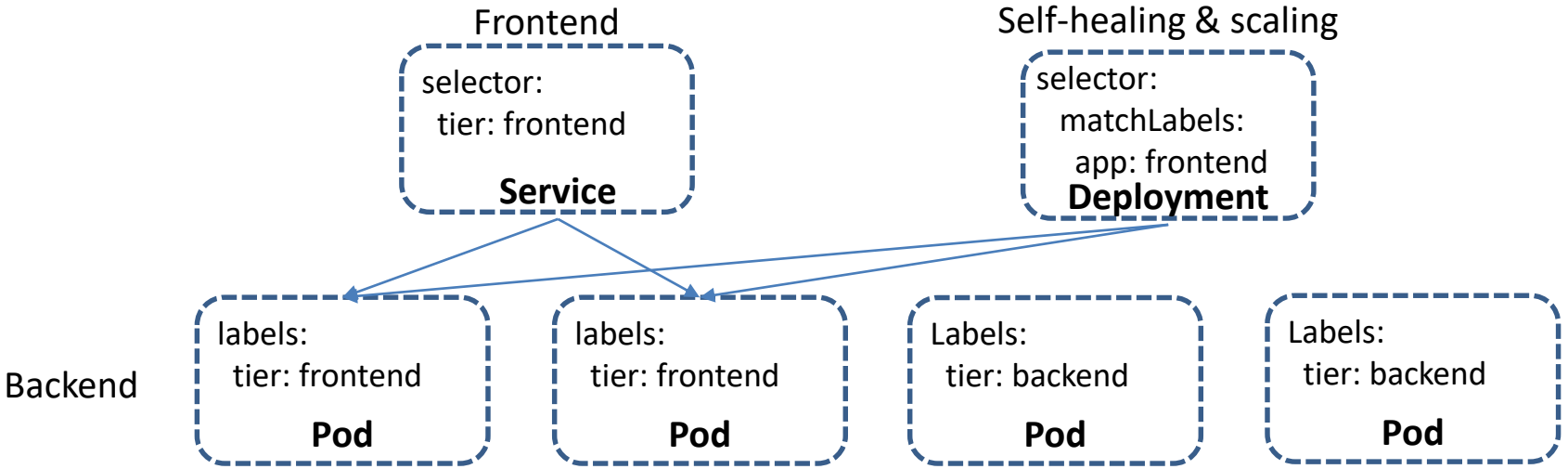
Life of a packet: pod-to-pod, between nodes



Pod to Service Communication in Kubernetes

Service

- Pod IP address keeps changing (**pod IPs ephemeral!!**)
- Need to find a way to hide this from client so it is enough to know one IP
- Service: logical set of backend Pods + stable front-end
  - Frontend: IP address + Port + DNS name --> independent from lifetime of backend pods
  - Backend: Logical set of pods whose label matches with the selector of the service
- Service ~ Load Balancer: route requests to the backend pods
- Endpoint: a separate object that stores all Pods that are selected by the selector



Service: multiport and defined IP address

Demo

Multiport:

- When a Service needs to expose more than one port
- In this case must give all of your ports names so that these are unambiguous

Define you own IP address:

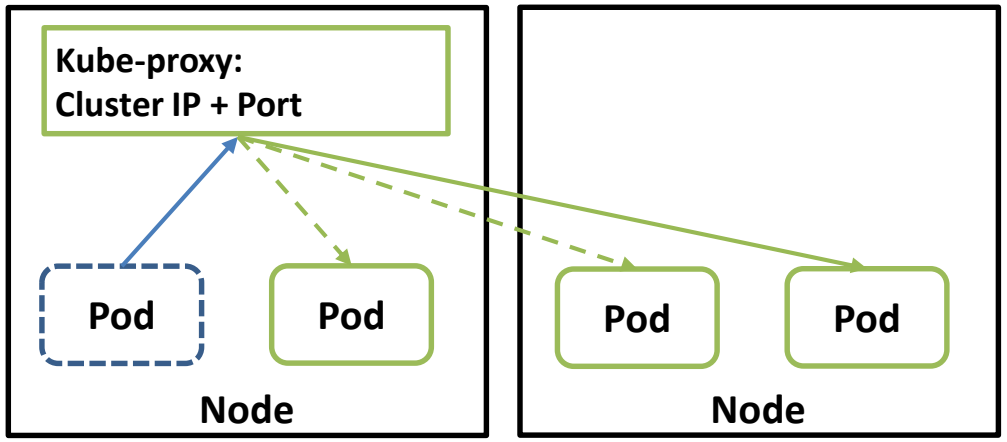
- ClusterIP in `spec.clusterIP` field
  - When:
    - Have an existing DNS entry that you wish to reuse
    - Legacy system that are configured for a specific IP address and difficult to re-configure
    - Must be a valid IPv4 or IPv6 address from within the `service-cluster-ip-range` CIDR range that is configured for the API server.
- ExternalIP in `spec.externalIPs` is also can be set:
  - If there are any that routes to one or more cluster nodes with this IP
  - It will be routed to the service that is defined with (then to its backend Pods)
  - Not managed by Kubernetes, the responsibility of the cluster administrator

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    tier: frontend
  ports:
    - name: http
      protocol: TCP
      port: 80
      targetPort: 8080
    - name: https
      protocol: TCP
      port: 443
      targetPort: 8081
```

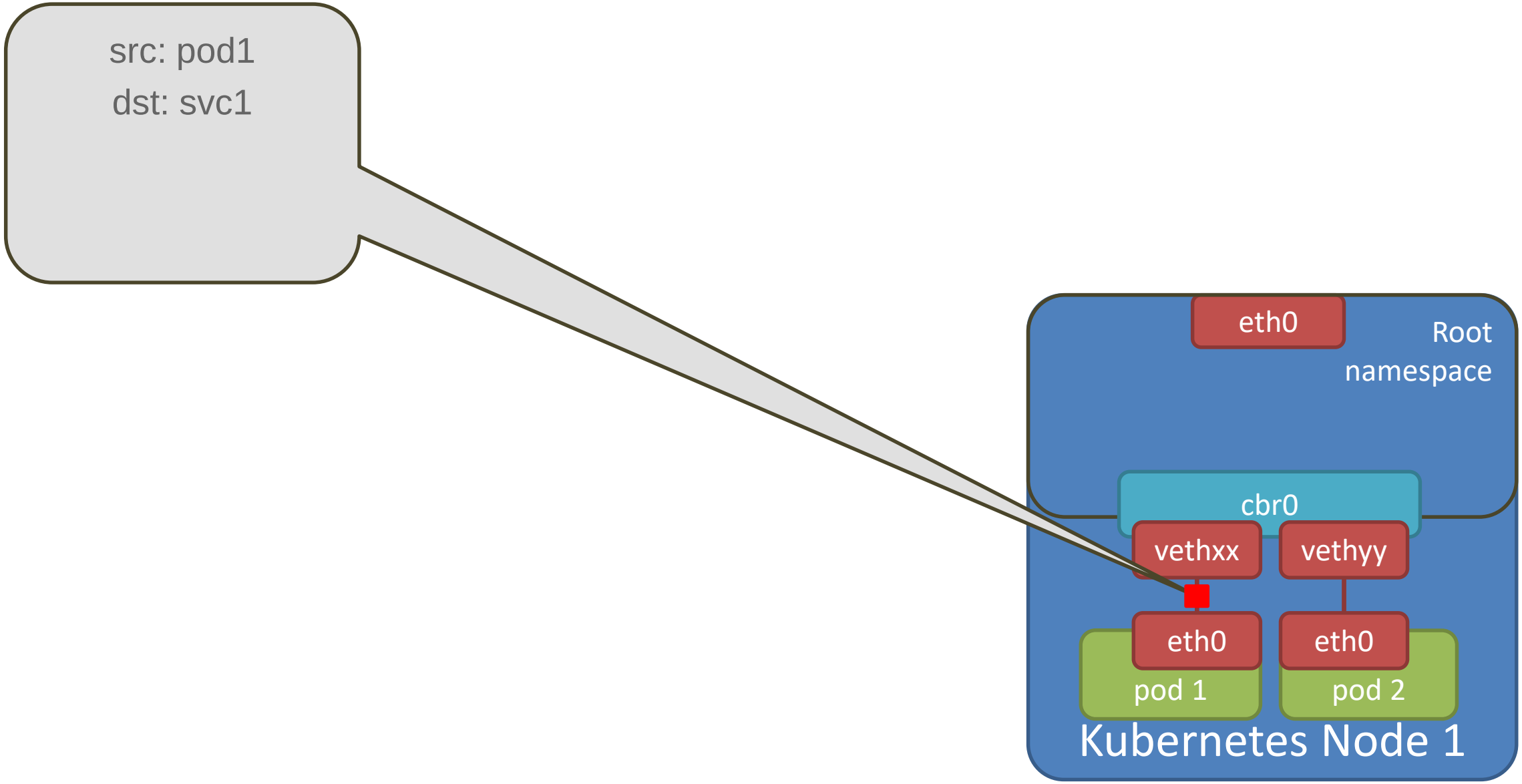
Service: types (ClusterIP)

- ClusterIP (default):
  - Exposes the Service on an internal IP in the cluster.
  - ClusterIP is independent from the backend Pod IP addresses.
  - This type makes the Service only reachable from within the cluster.
  - Route: ClusterIP -> Pod.

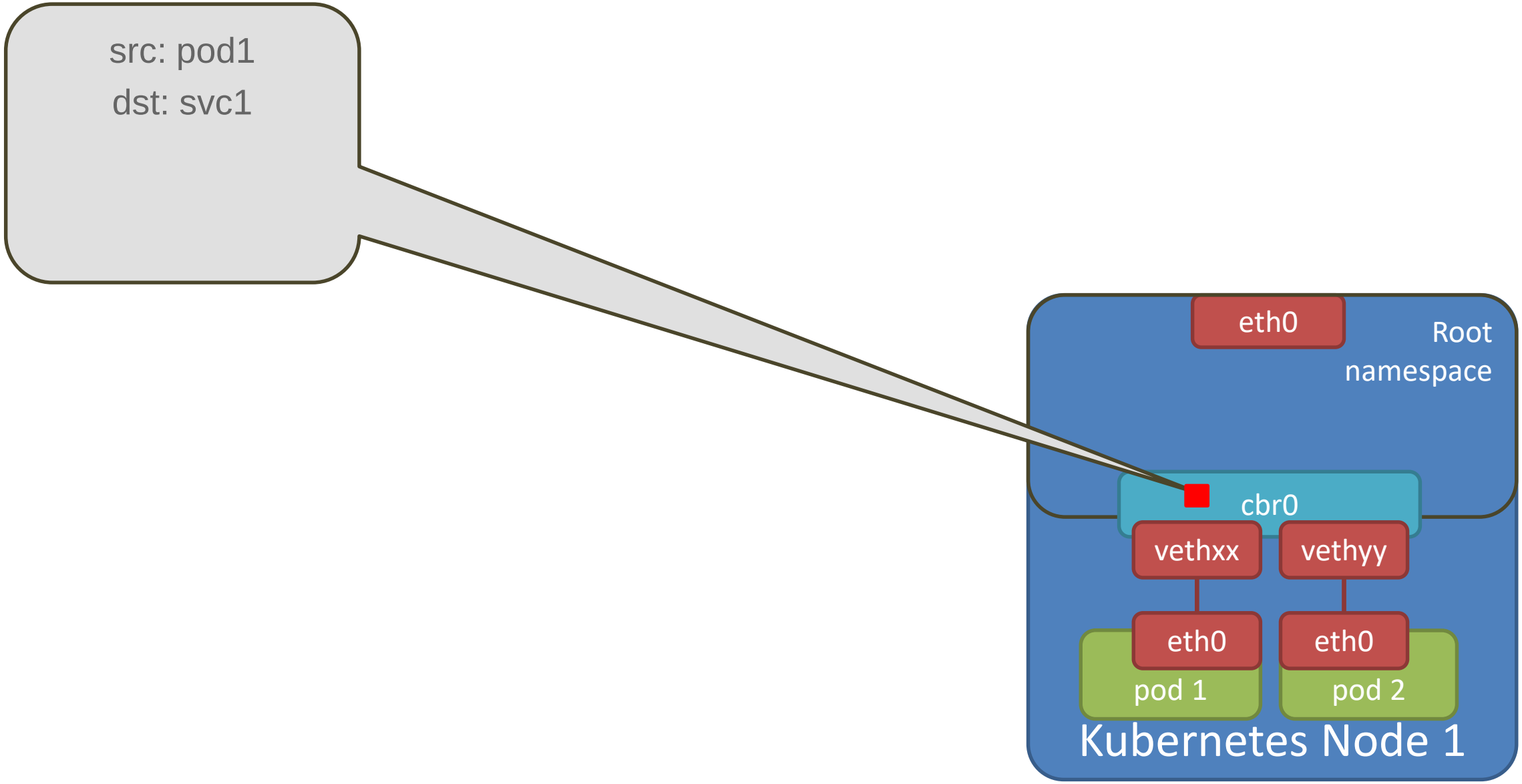
Logical view



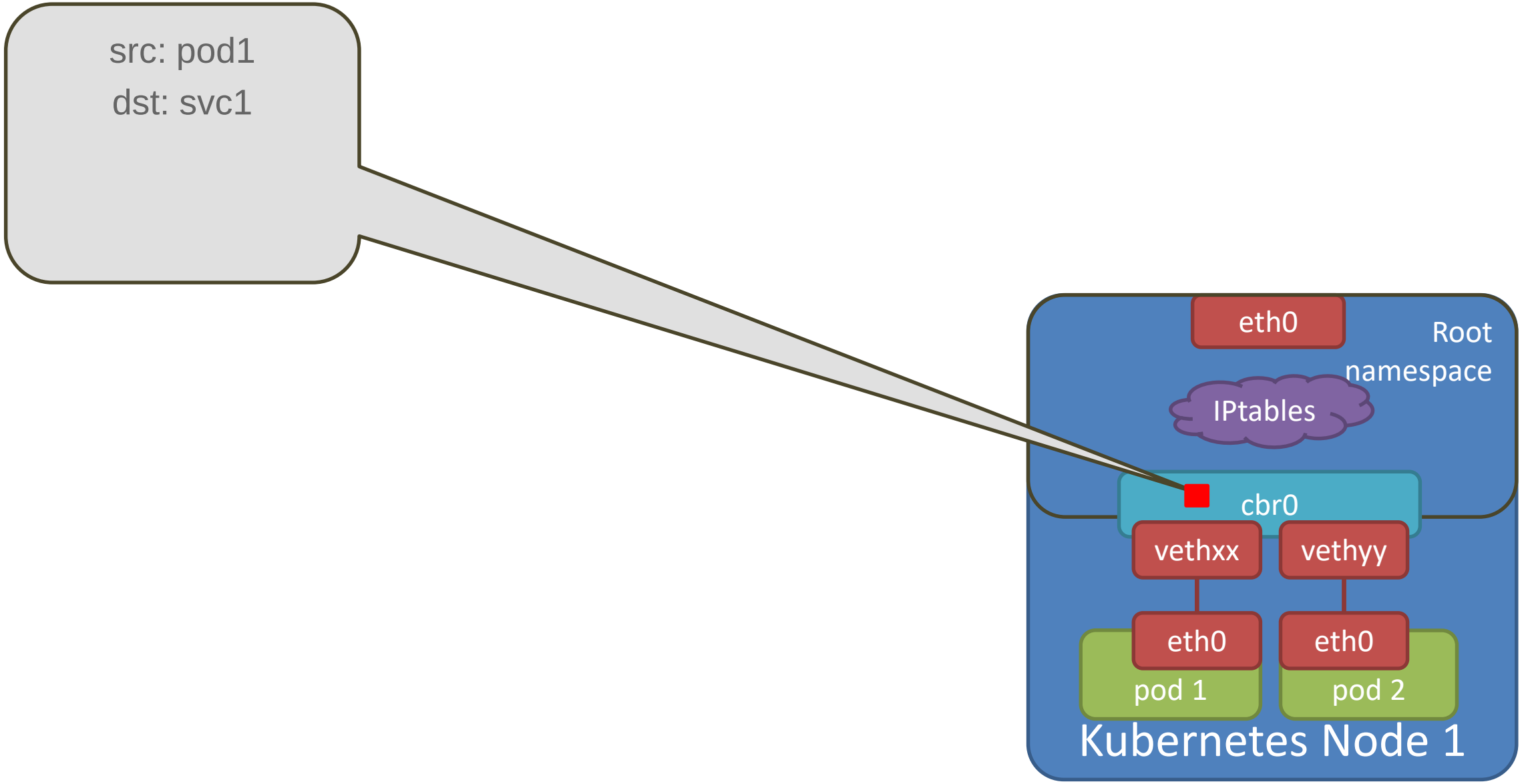
Life of a packet: pod-to-service



Life of a packet: pod-to-service



Life of a packet: pod-to-service



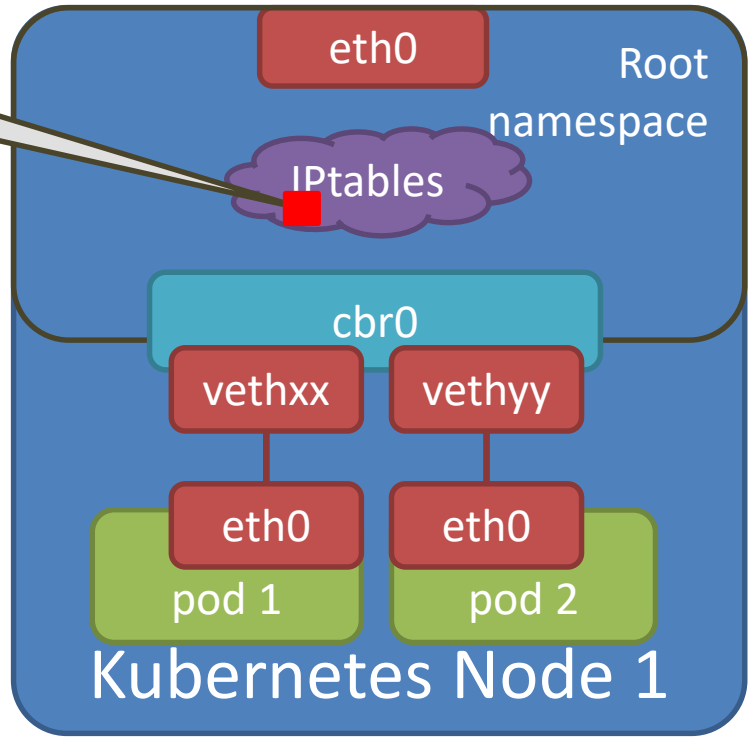
Life of a packet: pod-to-service

src: pod1

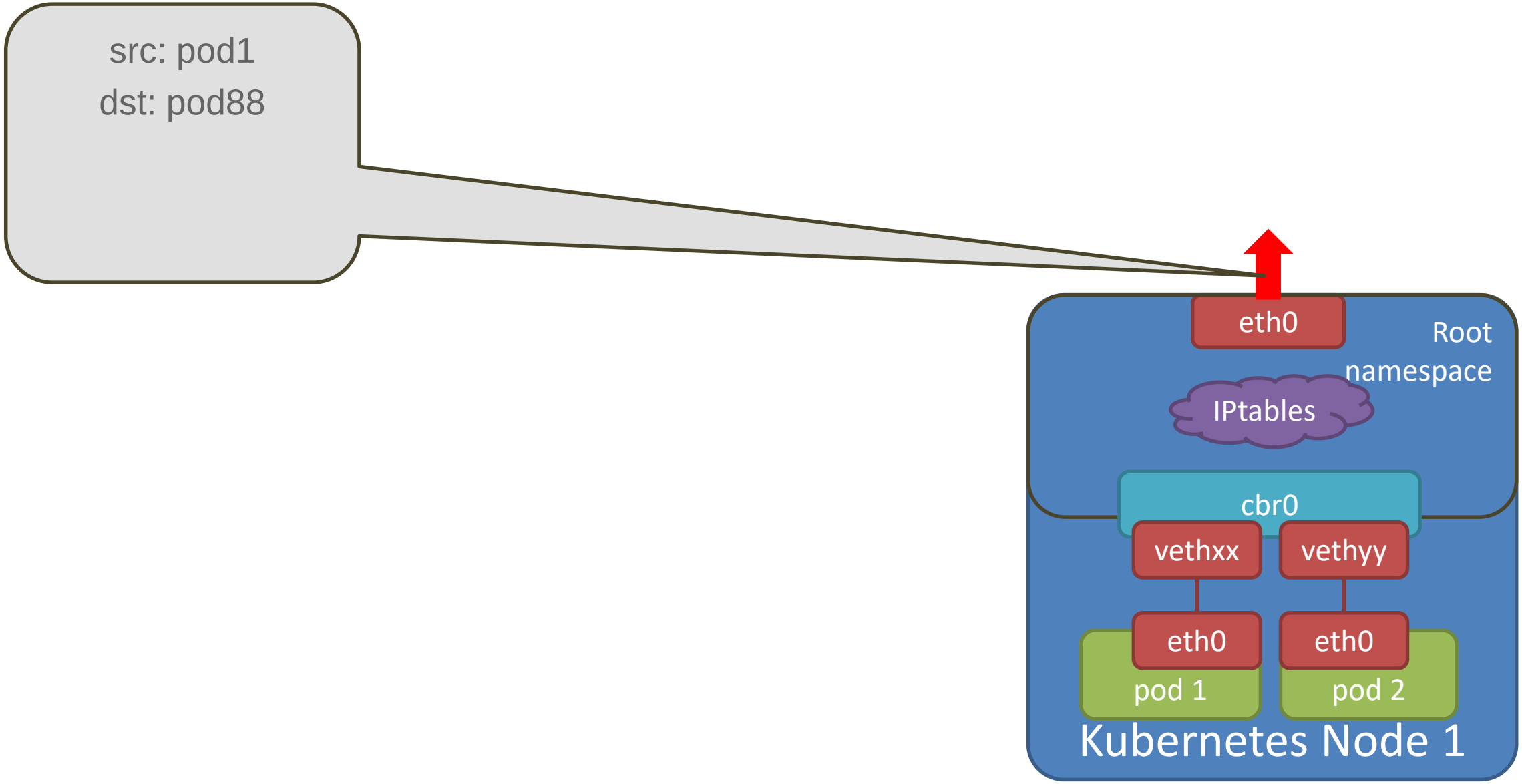
~~dst: svc1~~

dst: pod88

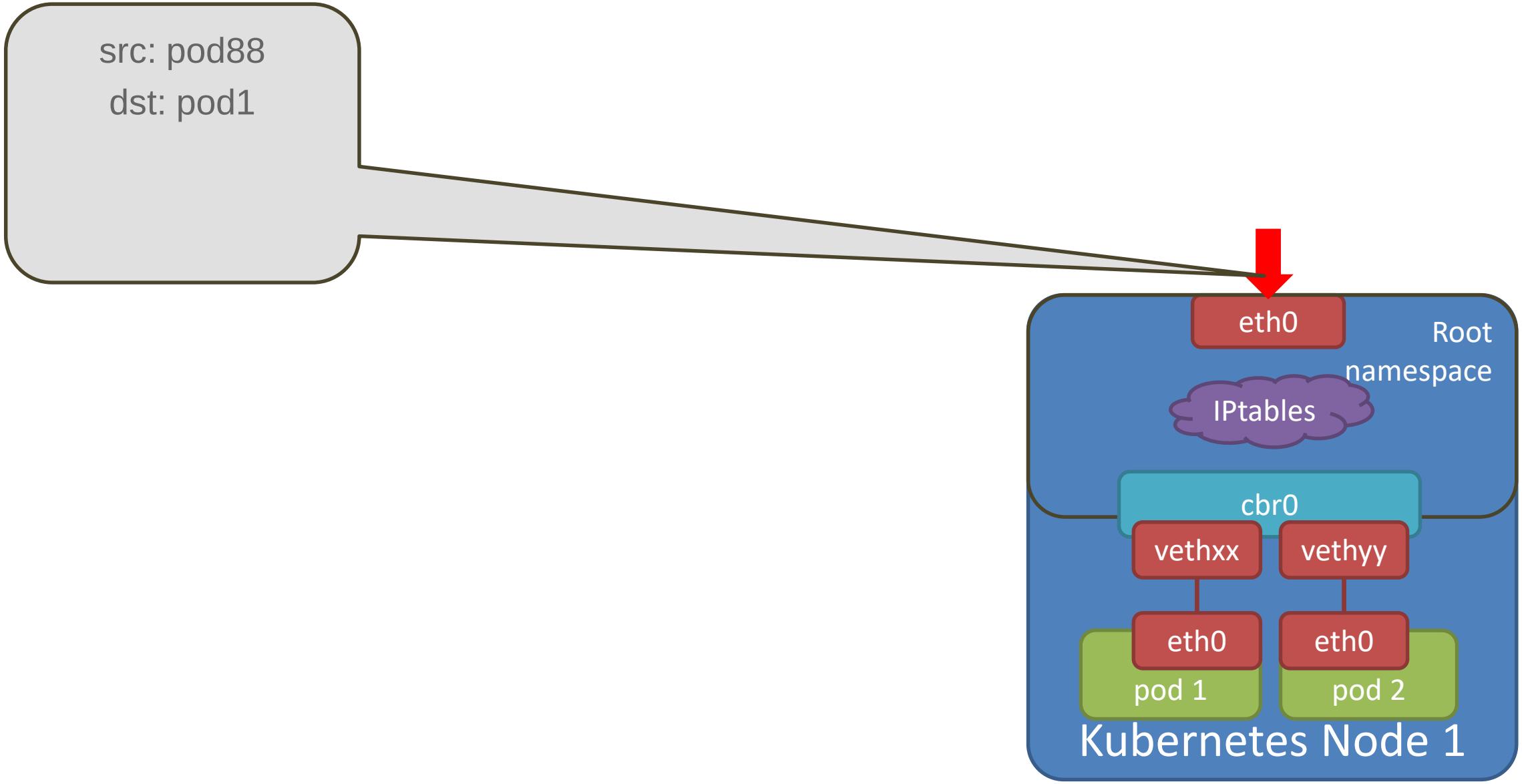
DNAT, conntrack



Life of a packet: pod-to-service



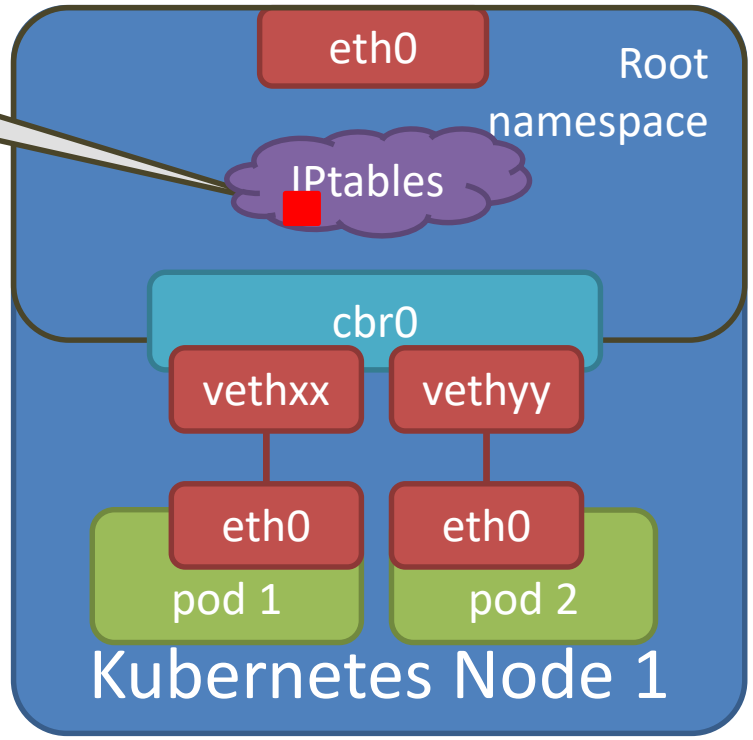
Life of a packet: pod-to-service



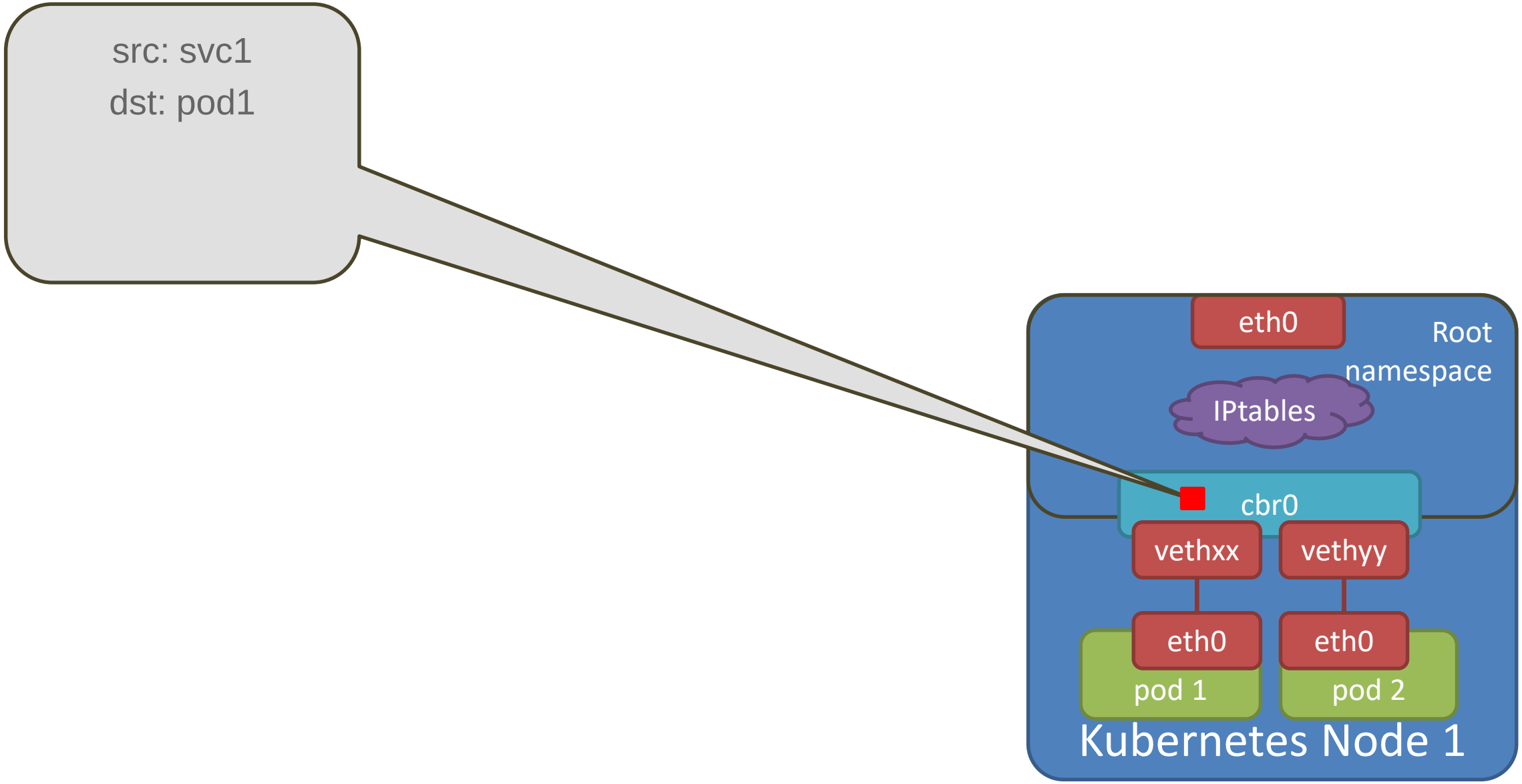
Life of a packet: pod-to-service

~~src: pod88~~
src: svc1
dst: pod1

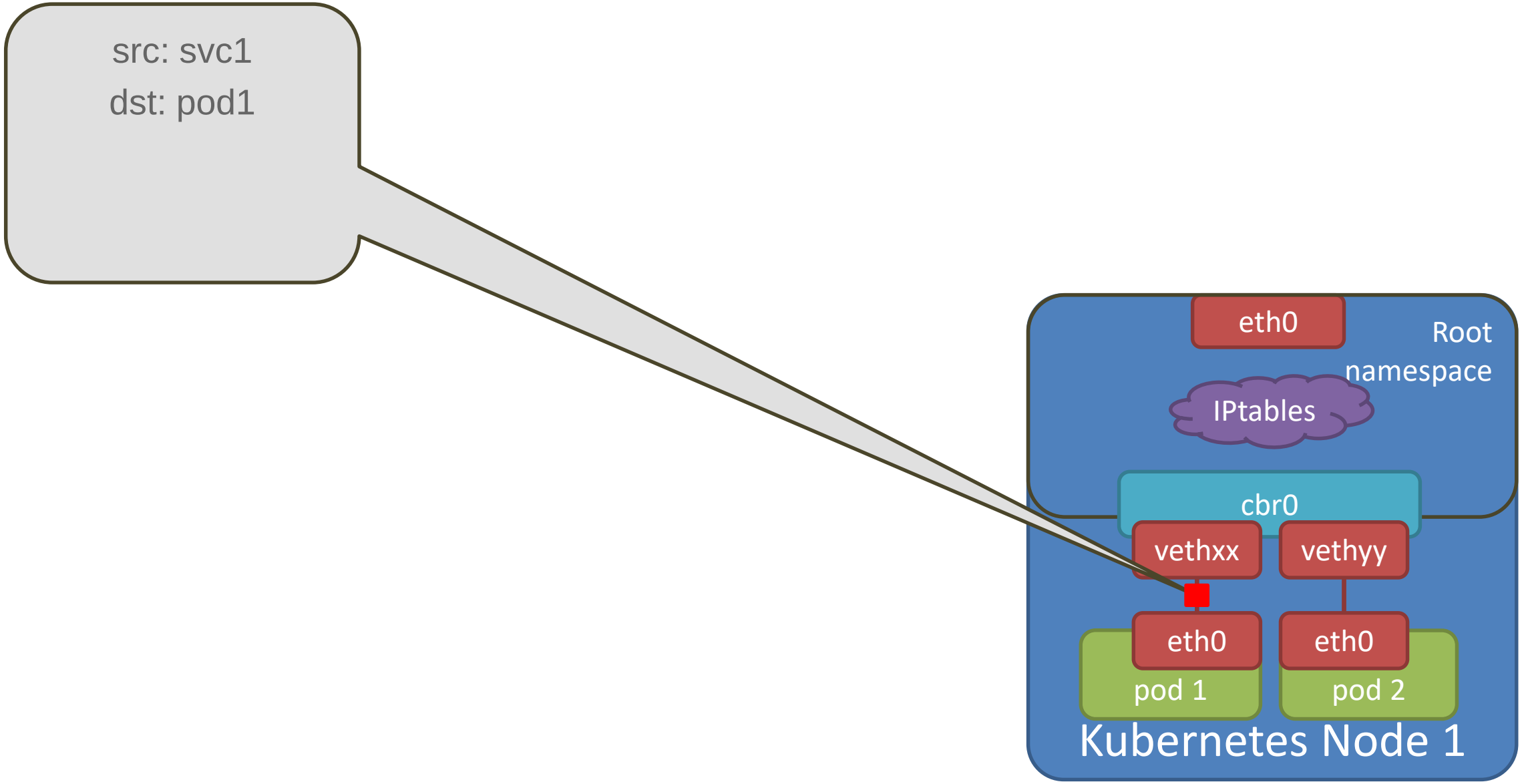
un-DNAT



Life of a packet: pod-to-service



Life of a packet: pod-to-service



Example for IPtables Ruleset

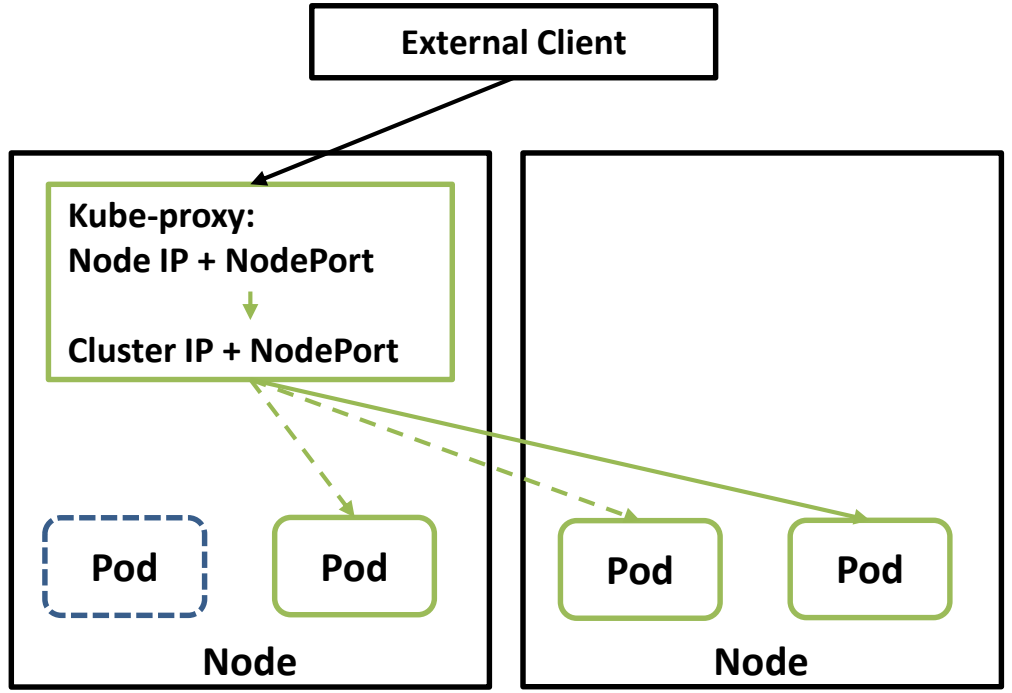
```
Chain KUBE-SERVICES (2 references)
target     prot opt source               destination
KUBE-MARK-MASQ  tcp  -- !10.244.0.0/16        10.110.89.105        /* sock-shop/front-end: cluster IP */ tcp dpt:http
KUBE-SVC-LFMD53S3EZEAOUSJ  tcp  --  anywhere             10.110.89.105        /* sock-shop/front-end: cluster IP */ tcp dpt:http
KUBE-MARK-MASQ  tcp  -- !10.244.0.0/16        10.97.201.132        /* sock-shop/orders-db: cluster IP */ tcp dpt:27017
KUBE-SVC-K7W4GUVR3E4J4SGZ  tcp  --  anywhere             10.97.201.132        /* sock-shop/orders-db: cluster IP */ tcp dpt:27017
KUBE-MARK-MASQ  tcp  -- !10.244.0.0/16        10.97.121.97         /* sock-shop/rabbitmq: cluster IP */ tcp dpt:amqp
KUBE-SVC-HFJ6SIC3BWQ7VZIS  tcp  --  anywhere             10.97.121.97         /* sock-shop/rabbitmq: cluster IP */ tcp dpt:amqp


Chain KUBE-SVC-LFMD53S3EZEAOUSJ (2 references)
target     prot opt source               destination
KUBE-SEP-55SOKLSCEKVOUEYD  all  --  anywhere             anywhere             /* sock-shop/front-end: */ statistic mode random probability 0.25000000000
KUBE-SEP-VW6NJSN2QROYSWMQ  all  --  anywhere             anywhere             /* sock-shop/front-end: */ statistic mode random probability 0.33332999982
KUBE-SEP-KCT3UGP5JLP4PQYI  all  --  anywhere             anywhere             /* sock-shop/front-end: */ statistic mode random probability 0.50000000000
KUBE-SEP-NXIYBBHETPWGYE3W  all  --  anywhere             anywhere             /* sock-shop/front-end: */


Chain KUBE-SEP-55SOKLSCEKVOUEYD (1 references)
target     prot opt source               destination
KUBE-MARK-MASQ  all  --  10.244.1.7           anywhere             /* sock-shop/front-end: */
DNAT       tcp  --  anywhere             anywhere            /* sock-shop/front-end: */ tcp to:10.244.1.7:8079
```

Service: types (NodePort)

Demo

- NodePort:
  - Exposes the Service on the same port of each selected Node in the cluster using NAT.
  - Makes a Service accessible from outside the cluster using <NodeIP>:<NodePort>.
  - Automatically creates ClusterIP.
  - Request is relayed from <NodeIP>:<NodePort> to <ClusterIP>:<NodePort>.
  - Route: NodePort -> ClusterIP -> Pod.

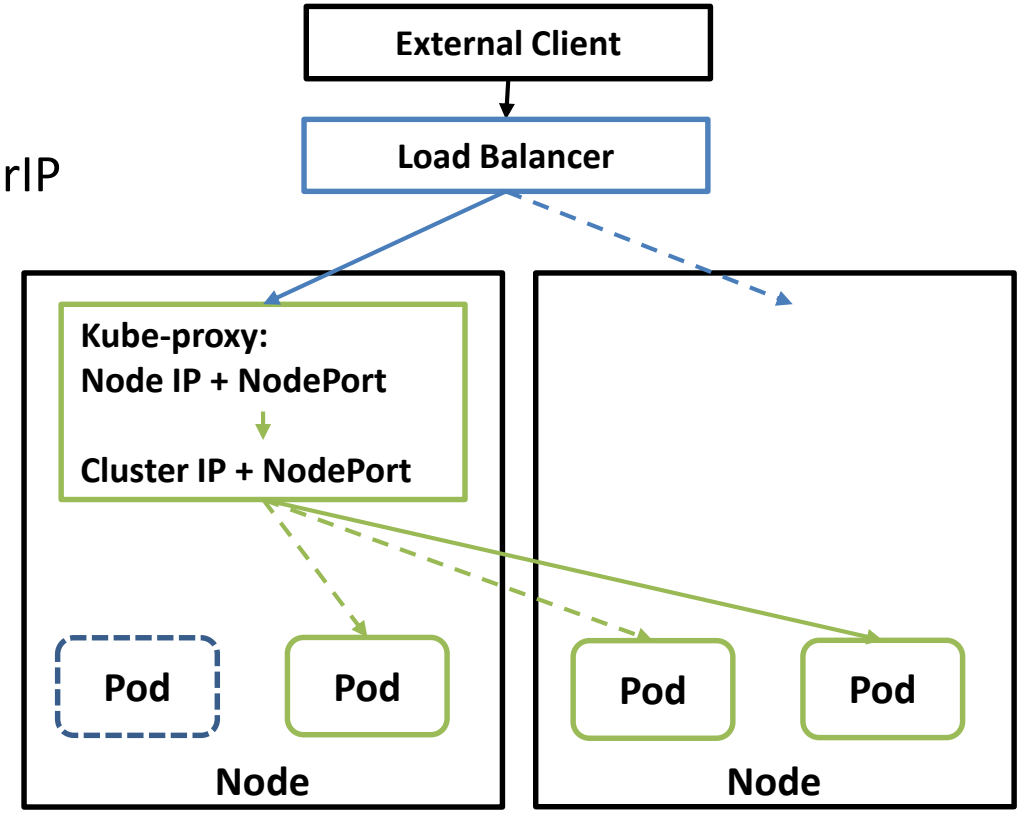
Logical view



Service: types (LoadBalancer)

- LoadBalancer (Ingress is an alternative, see it later):
  - Exposes the Service externally using a cloud provider's load balancer.
  - Require tight cooperation with the cloud provider’s infrastructure (via the cloud-controller-manager)
  - Assigns a fixed, external IP to the Service.
  - Automatically creates NodePort and ClusterIP.
  - Routing: External LB -> NodePort -> ClusterIP -> Pod.
  - Some cloud providers allow you to specify the loadBalancerIP
    - If yes and not specified, an ephemeral IP is created
    - If not then this field is ignored even if it is specified

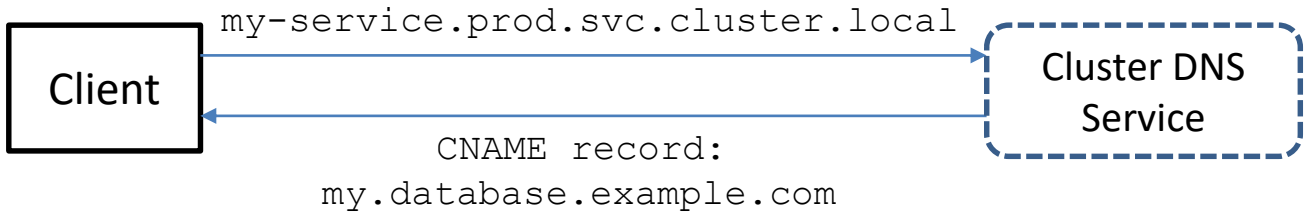
Logical view



Service: types (ExternalName)

Demo

- ExternalName:
  - Exposes the Service using an arbitrary name (specified in `spec.ExternalName`) by returning a CNAME record with the name.
  - No proxy is used.
  - Redirection happens at the DNS level rather than via proxying or forwarding as with other Services
  - E.g.
    1. Lookup to my-service.prod.svc.cluster.local
    2. Cluster DNS Service returns a CNAME record with the value




```
apiVersion: v1
kind: Service
metadata:
  name: my-service
  namespace: prod
spec:
  type: ExternalName
  externalName: my.db.example.com
```

Service: types + 1 (Headless)

- Headless Service:
  - When don't need load-balancing and a single Service IP (e.g. DB replicas with one master + 2 read replicas)
  - In `spec.ClusterIP: None`
  - Cluster IP is not allocated
  - Kube-proxy does not handle these Services
  - There is no load balancing or proxying done by the platform for them
  - With selectors:
    - endpoints controller creates Endpoints records
    - modifies the DNS configuration to return records (addresses) that point directly to the Pods backing the Service
  - Without selectors:
    - endpoints controller does not create Endpoints records
  - Redirection happens at the DNS level rather than via proxying or forwarding as with other Services
  - E.g.
    1. Lookup to my-service.prod.svc.cluster.local
    2. Cluster DNS Service returns a CNAME record with the value

Service: discovery (environment variables and DNS)

Environment variables:

- kubelet adds a set of environment variables for each active Service
- E.g. "redis-master" which exposes TCP port 6379 and allocated cluster IP address 10.0.0.11

```
REDIS_MASTER_SERVICE_HOST=10.0.0.11
REDIS_MASTER_SERVICE_PORT=6379
REDIS_MASTER_PORT=tcp://10.0.0.11:6379
REDIS_MASTER_PORT_6379_TCP=tcp://10.0.0.11:6379
REDIS_MASTER_PORT_6379_TCP_PROTO=tcp
REDIS_MASTER_PORT_6379_TCP_PORT=6379
REDIS_MASTER_PORT_6379_TCP_ADDR=10.0.0.11
```

- If the pod would rely on this method, the service needs to be created first!

Service: discovery (environment variables and DNS)

DNS:

1. For Services (A/AAAA and SRV records):
   - A DNS Service (e.g. CoreDNS) watches the API for new Services and creates records.
   - Pods should be able to automatically resolve Services by their DNS names.
   - For Service names (A records):
     - Service name: `my-service`, Namespace: `my-namespace`
     - Service is available
       - for pods in the same namespace: `my-service`
       - for pods in other namespaces: `my-service.my-namespace`
         - » More precisely: `my-service.my-namespace.svc.cluster.local`
     - Returns the ClusterIP
   - For named ports (DNS SRV records):
     - `port-name.port-protocol.my-service.my-namespace.svc.cluster.local`
     - Port number is returned

Service: discovery (environment variables and DNS)

Demo

DNS:

2. For Pods (A/AAAA records):
   - `pod-ip-address.my-namespace.pod.cluster-domain.example`
   - E.g.:
     - Pod namespace: `my-ns`, IP: `172.17.0.3`, cluster domain name: `cluster.local`
       - `172-17-0-3.my-ns.pod.cluster.local`
     - If the Pod is created with Deployment or DaemonSet (`name: my-d`) and exposed by a Service:
       - `172-17-0-3.my-d.my-ns.svc.cluster.local`
     - If spec.hostname: foo and spec.subdomain: bar:
       - `foo.bar.my-ns.svc.cluster.local`
     - Hostname is `metadata.name` by default but `spec.hostname` overwrites it.

Service: discovery (environment variables and DNS)

DNS:

2. For Pods (A/AAAA records):
   - DNS policies can be set on a per-pod basis (so how the Pod can access the rest of the world):
   - In `spec.dnsPolicy` field:
     - Default:
       - inherits the name resolution configuration from the node that the pods run on.
     - ClusterFirst:
       - DNS queries that doesn't match the configured cluster domain suffix, such as "www.kubernetes.io", is forwarded to the upstream nameserver inherited from the node.
     - ClusterFirstWithHostNet:
       - for Pods running with hostNetwork, you should explicitly set this DNS policy.
     - None:
       - ignore DNS settings from the Kubernetes environment. All DNS settings are supposed to be provided using the `dnsConfig` field in the Pod Spec.

Service: without selector

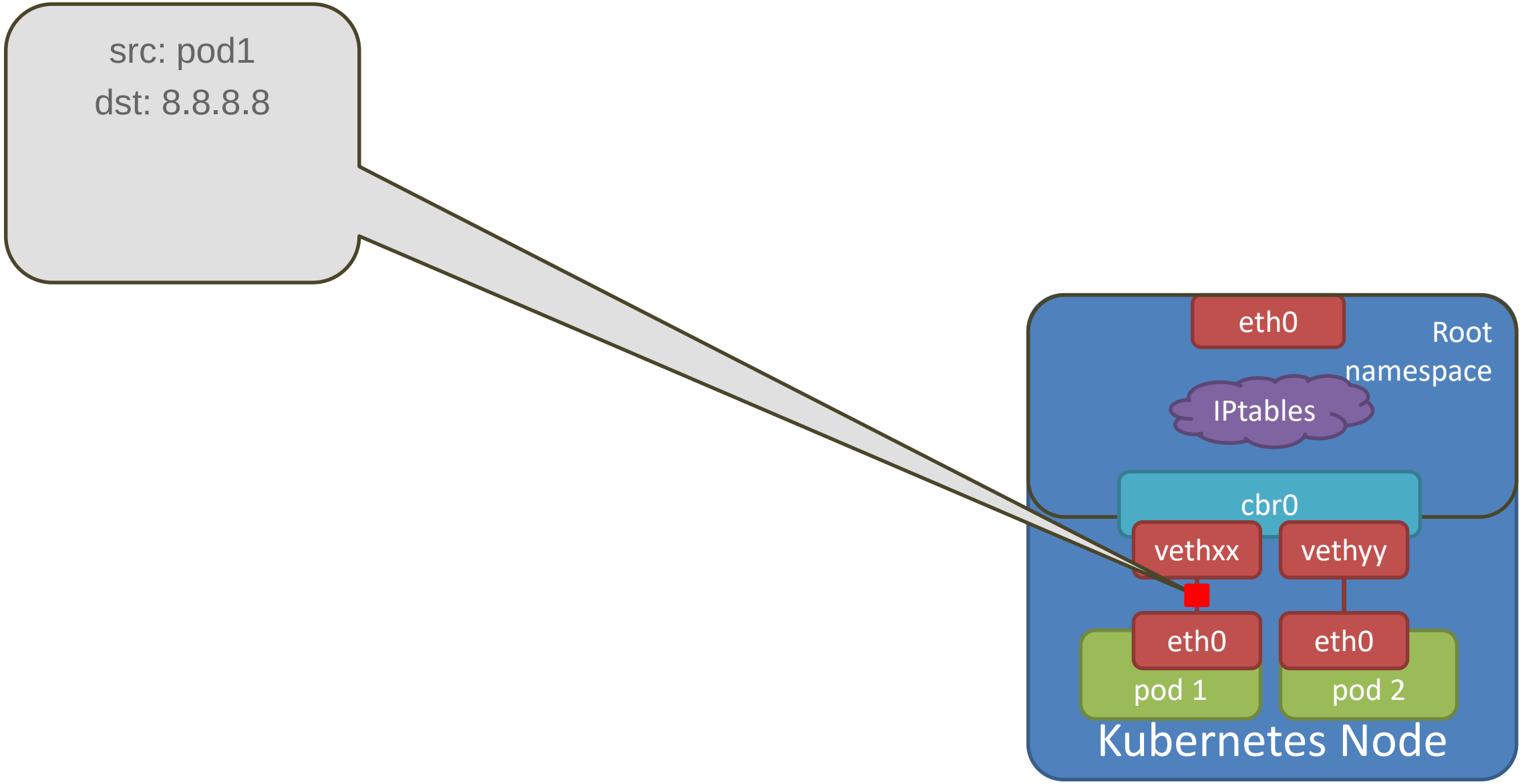
- Services usually abstract access to Pods, but can also abstract other kinds of backends:
  - external database cluster in production, but in your test environment you use your own databases
  - point your Service to a Service in a different Namespace or on another cluster
  - during migration only a proportion of backend run in Kubernetes
- In any of these scenarios you can:
  - define a Service *without* a Pod selector (no Endpoint will be created)
  - create your own Endpoint object
- External Name is another option (one of the 4(5) Service types):
  - Redirection happens at the DNS level rather than via proxying or forwarding

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
  namespace: prod
spec:
  type: ExternalName
  externalName: my.database.example.com
```

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
---
apiVersion: v1
kind: Endpoints
metadata:
  name: my-service
subsets:
  - addresses:
      - ip: 192.0.2.42
    ports:
      - port: 9376
```

Pod to External Communication in Kubernetes

Life of a packet: pod-to-external

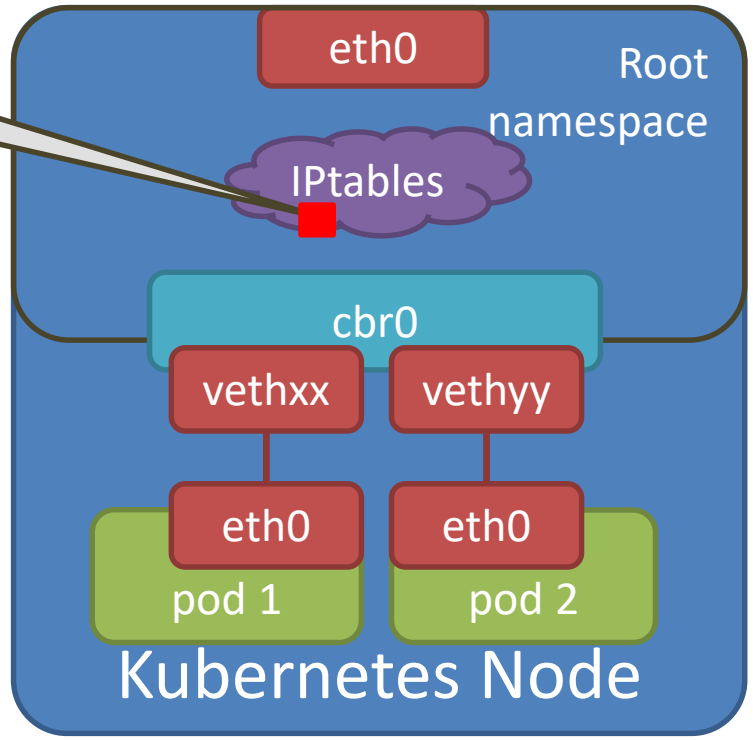


Life of a packet: pod-to-external

src: pod1
dst: 8.8.8.8

POD IP address is private

- Needs NAT to communicate with external



Life of a packet: pod-to-external

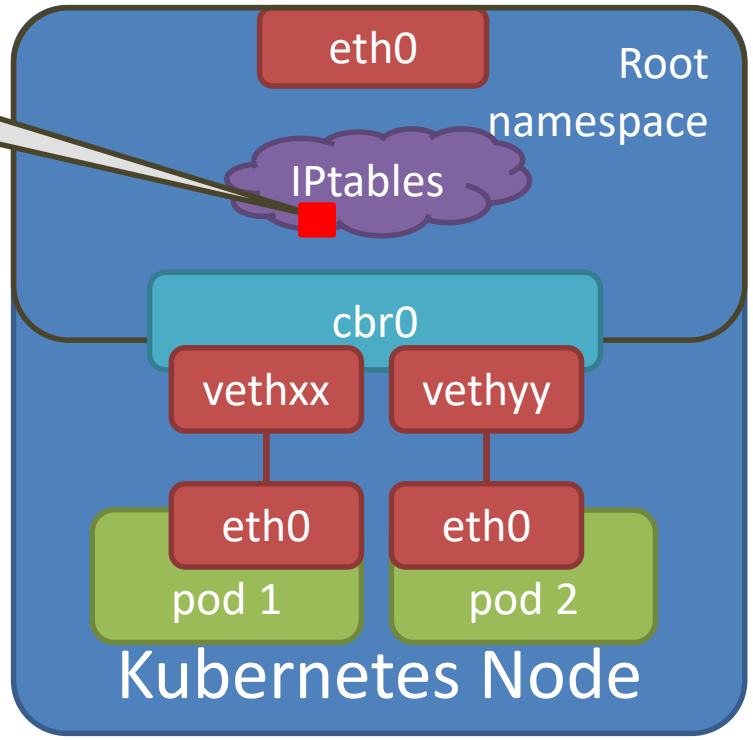
~~src: pod1~~
src: NodeIP
dst: 8.8.8.8

MASQUERADE

POD IP address is private

- Needs NAT to communicate with external

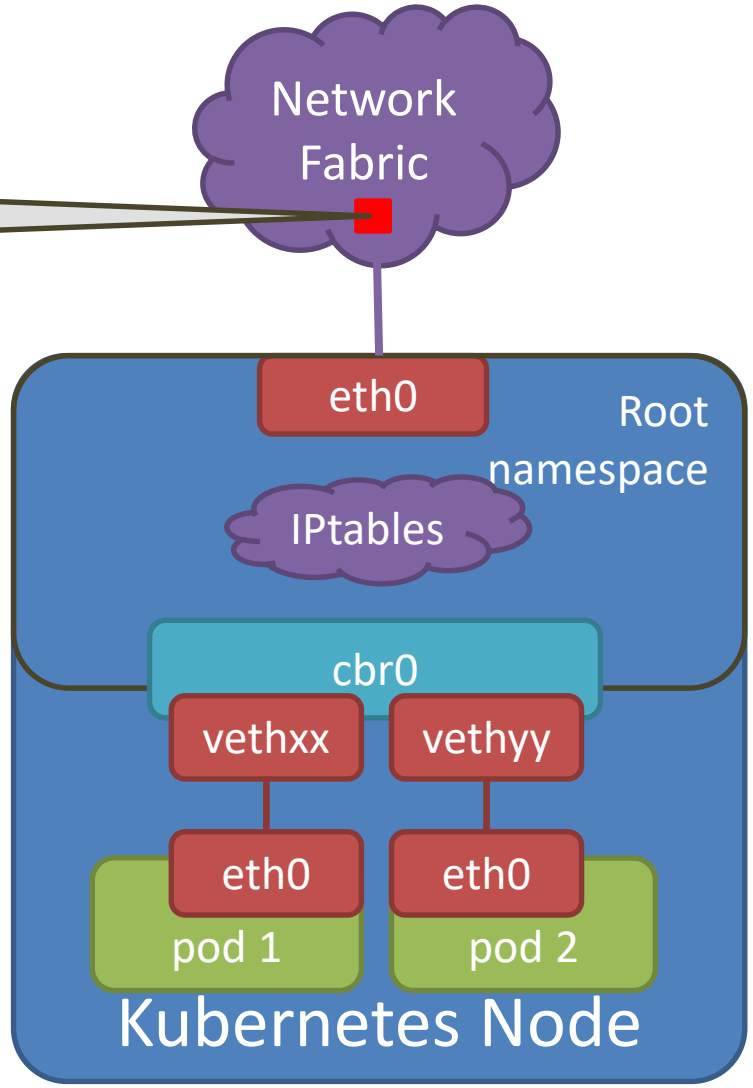
Node IPs are usually also private



Life of a packet: pod-to-external

~~src: NodeIP~~
src: PublicIP
dst: 8.8.8.8

MASQUERADE



POD IP address is private

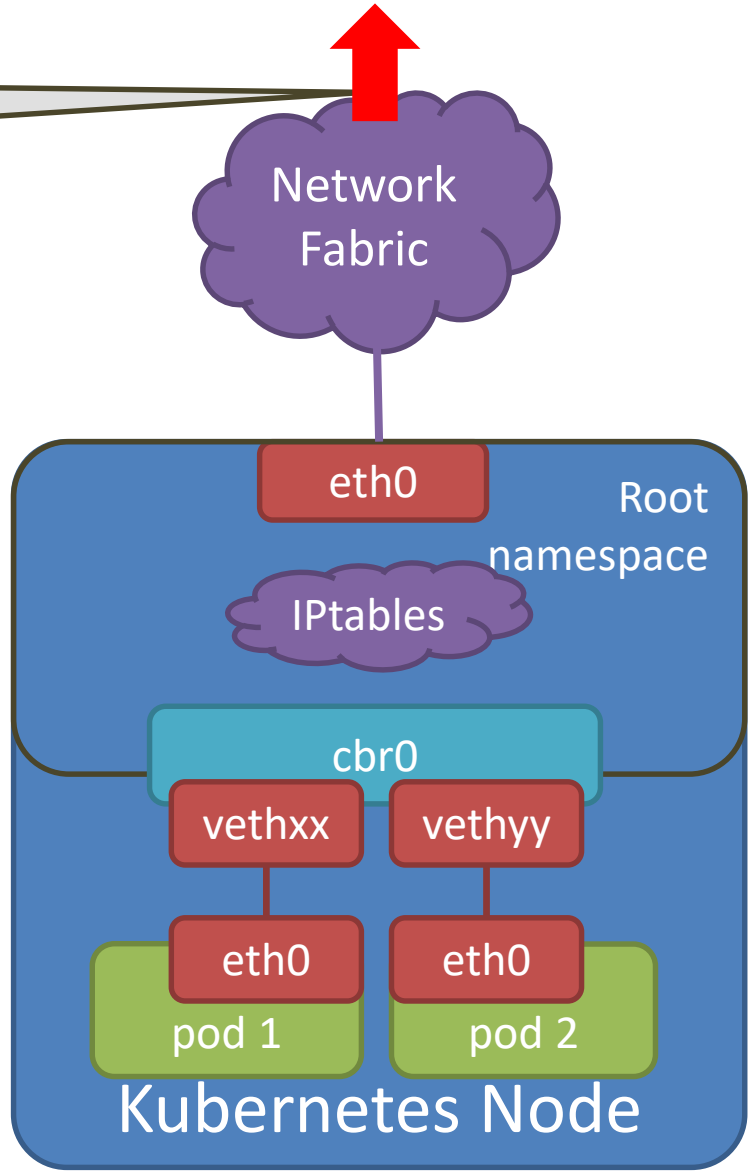
- Needs NAT to communicate with external

Node IPs are usually also private

- Needs second NAT by the fabric

Life of a packet: pod-to-external

src: PublicIP
dst: 8.8.8.8



POD IP address is private

- Needs NAT to communicate with external

Node IPs are usually also private

- Needs second NAT by the fabric

External to Internal Communication in Kubernetes

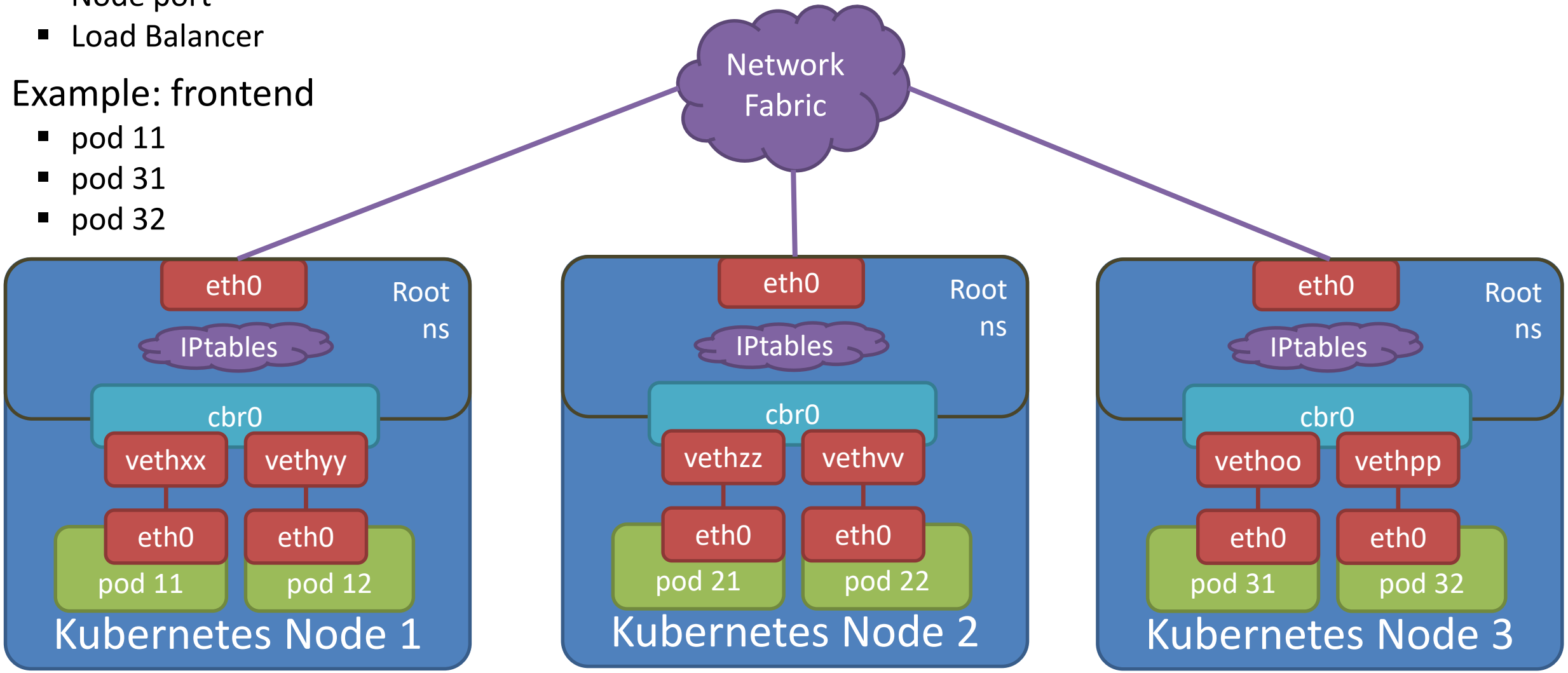
External-to-Internal Traffic

Serveices can be exposed to the outside by

- Node port
- Load Balancer

Example: frontend

- pod 11
- pod 31
- pod 32



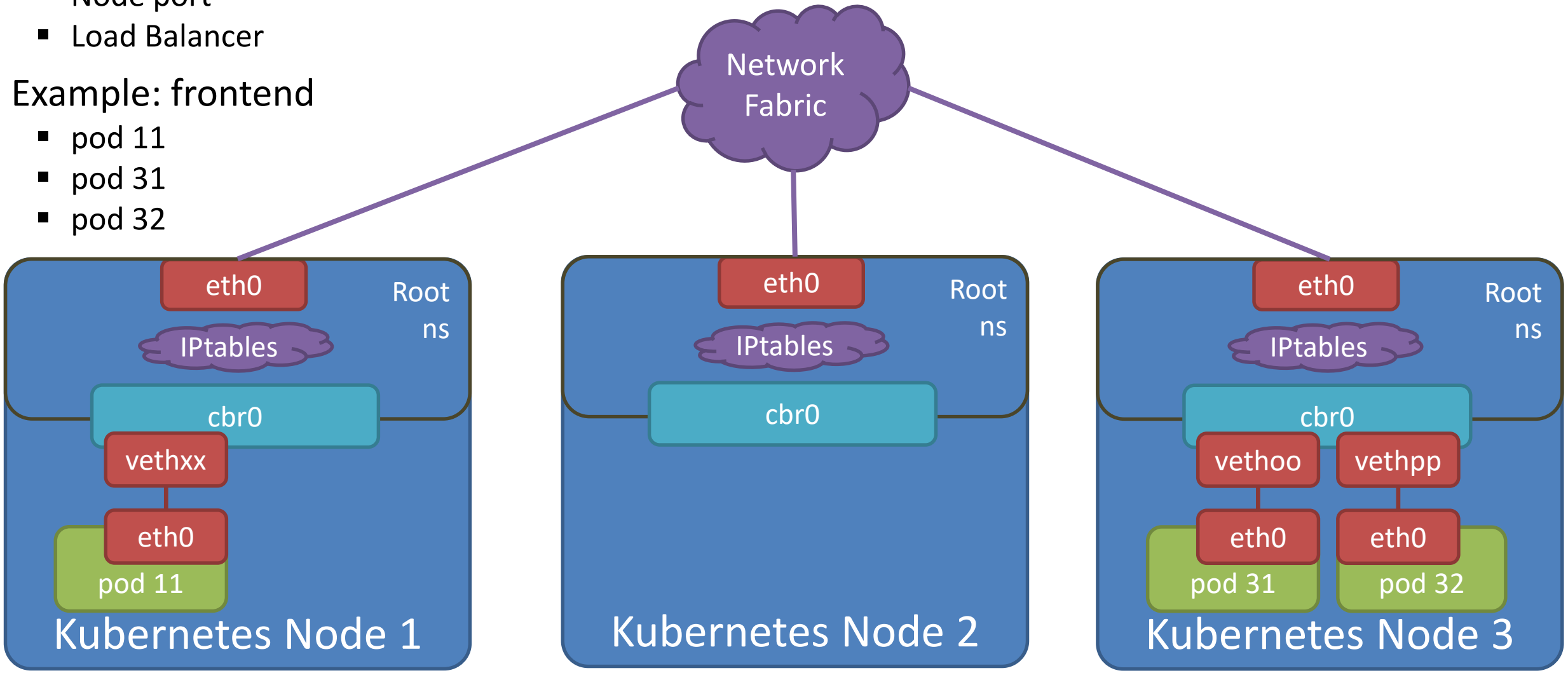
External-to-Internal Traffic

Serveices can be exposed to the outside by

- Node port
- Load Balancer

Example: frontend

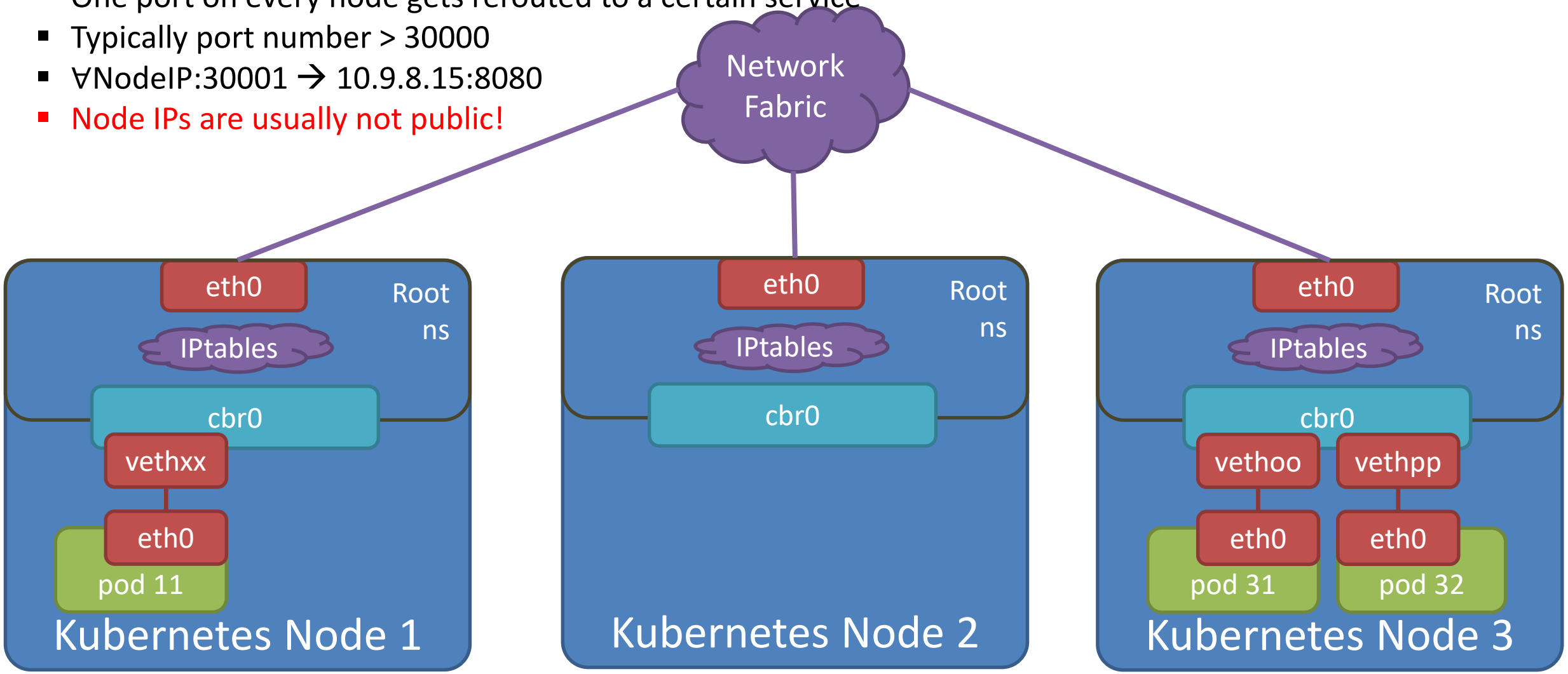
- pod 11
- pod 31
- pod 32



External-to-Internal Traffic

Node port

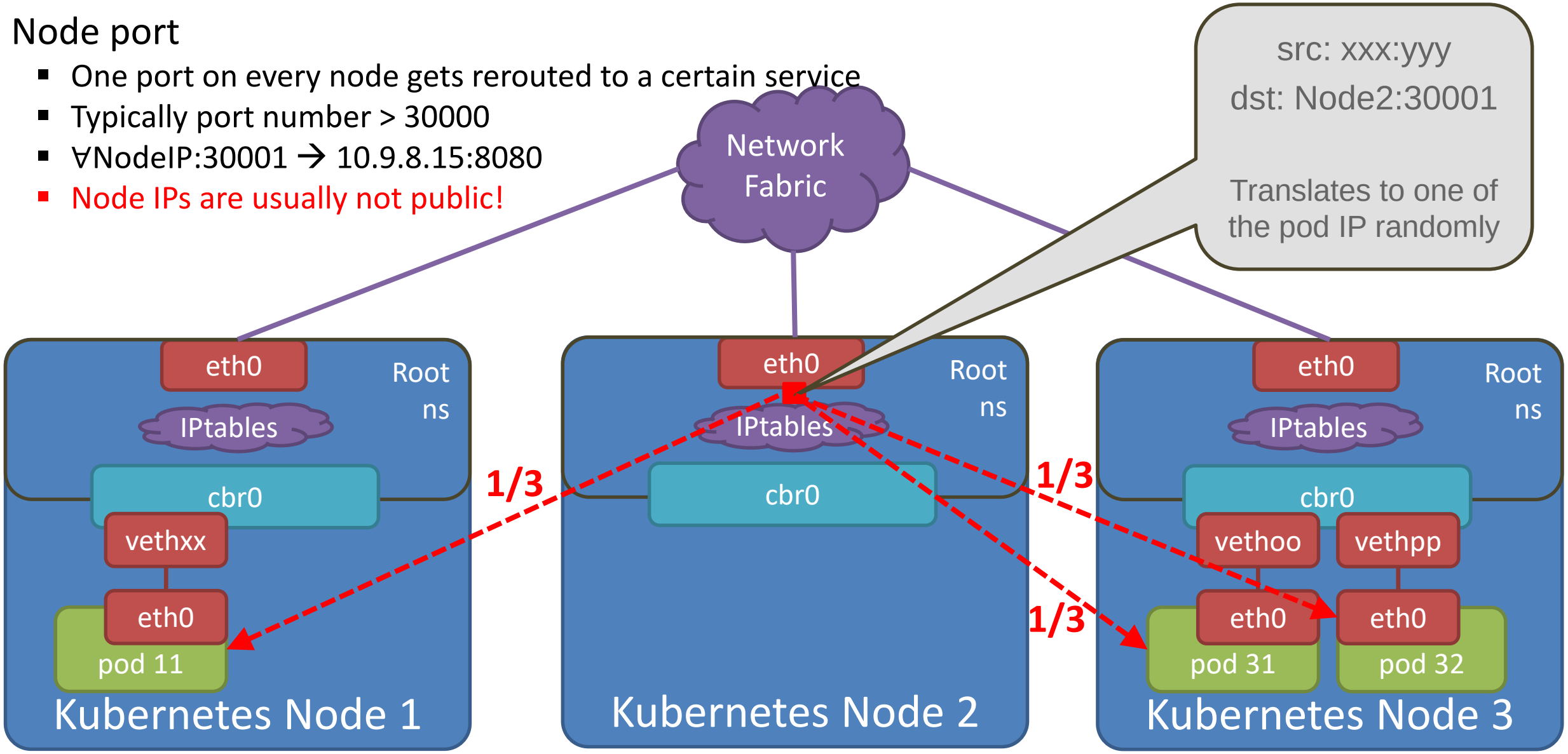
- One port on every node gets rerouted to a certain service
- Typically port number > 30000
- ∀NodeIP:30001 → 10.9.8.15:8080
- Node IPs are usually not public!



External-to-Internal Traffic

Node port

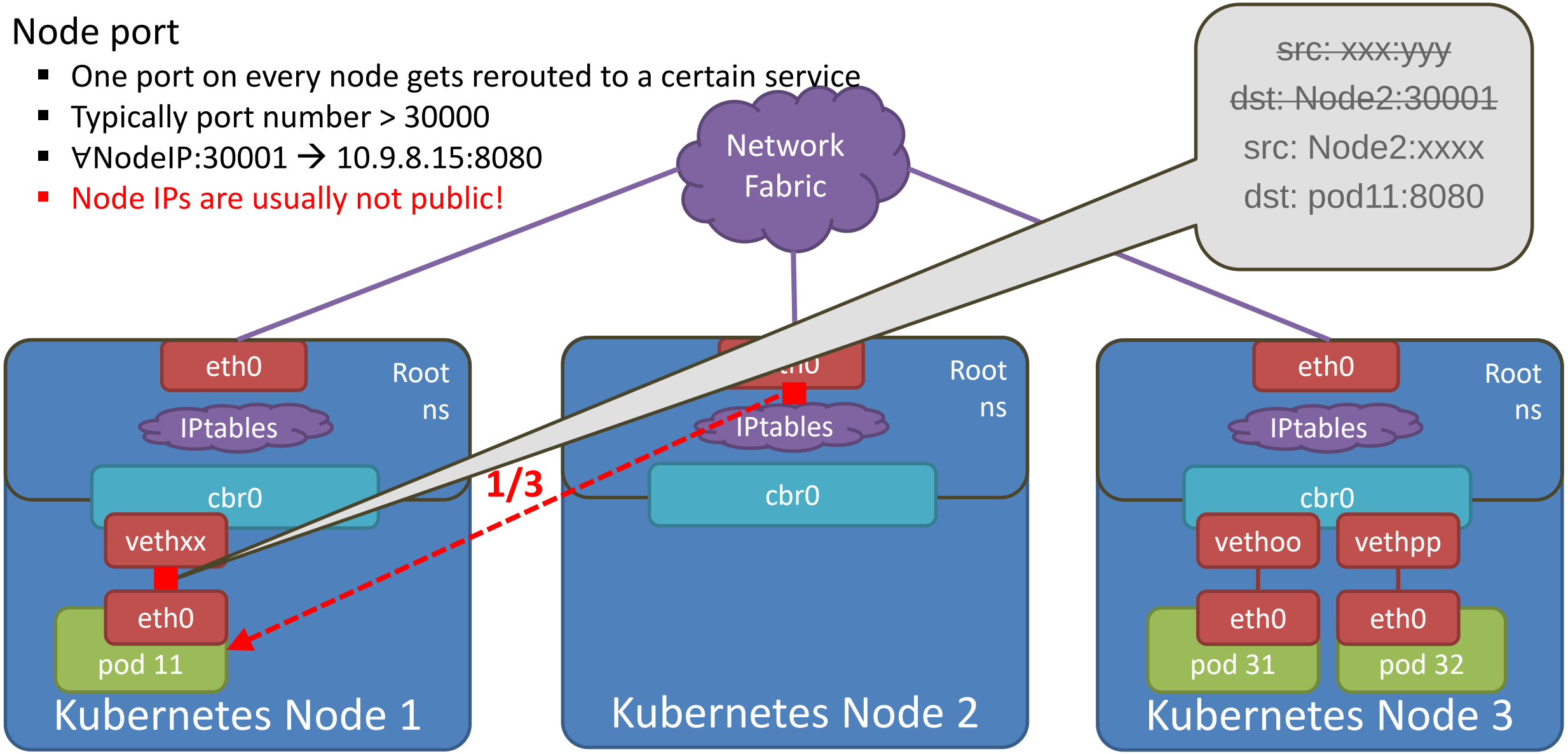
- One port on every node gets rerouted to a certain service
- Typically port number > 30000
- ∀NodeIP:30001 → 10.9.8.15:8080
- Node IPs are usually not public!



External-to-Internal Traffic

Node port

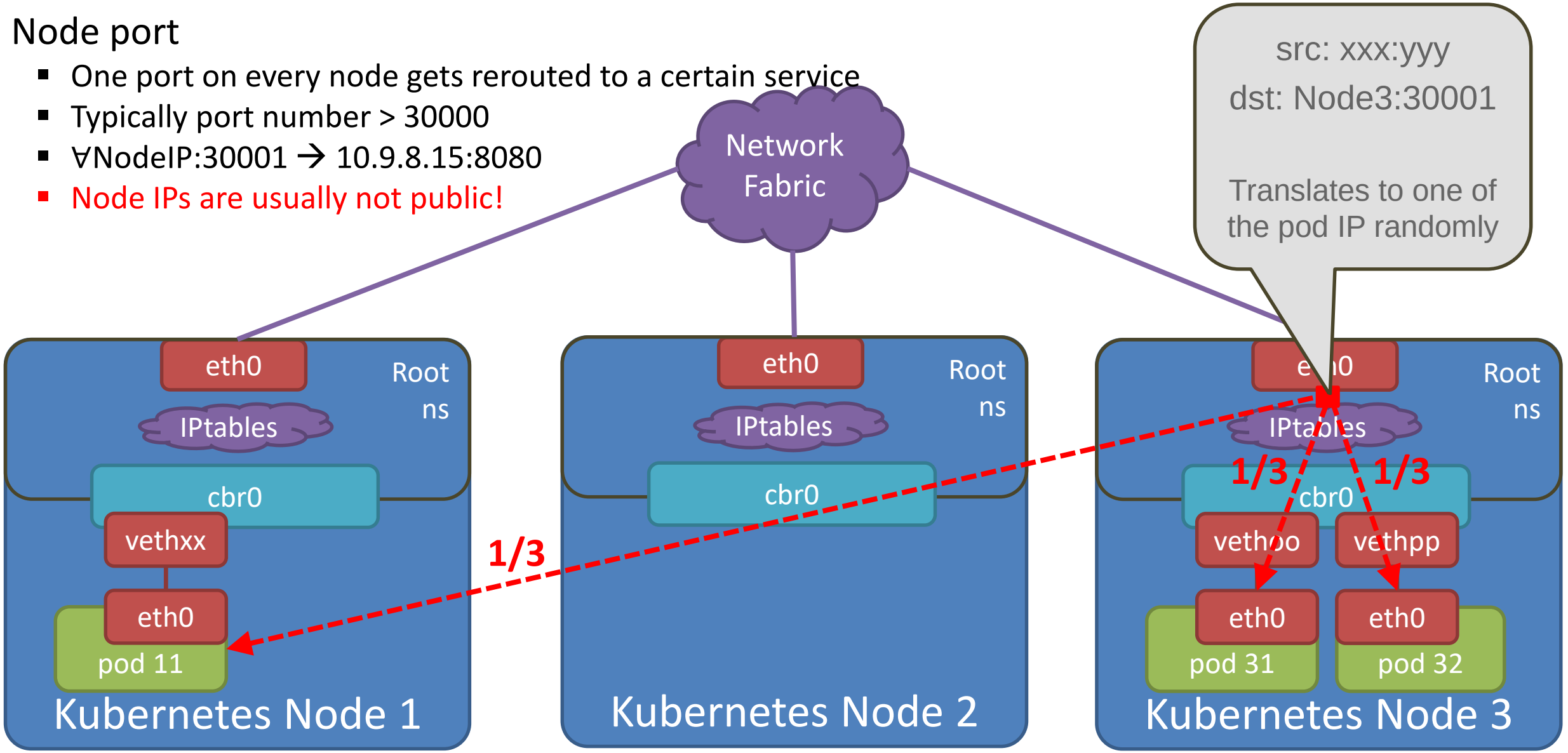
- One port on every node gets rerouted to a certain service
- Typically port number > 30000
- ∀NodeIP:30001 → 10.9.8.15:8080
- Node IPs are usually not public!



External-to-Internal Traffic

Node port

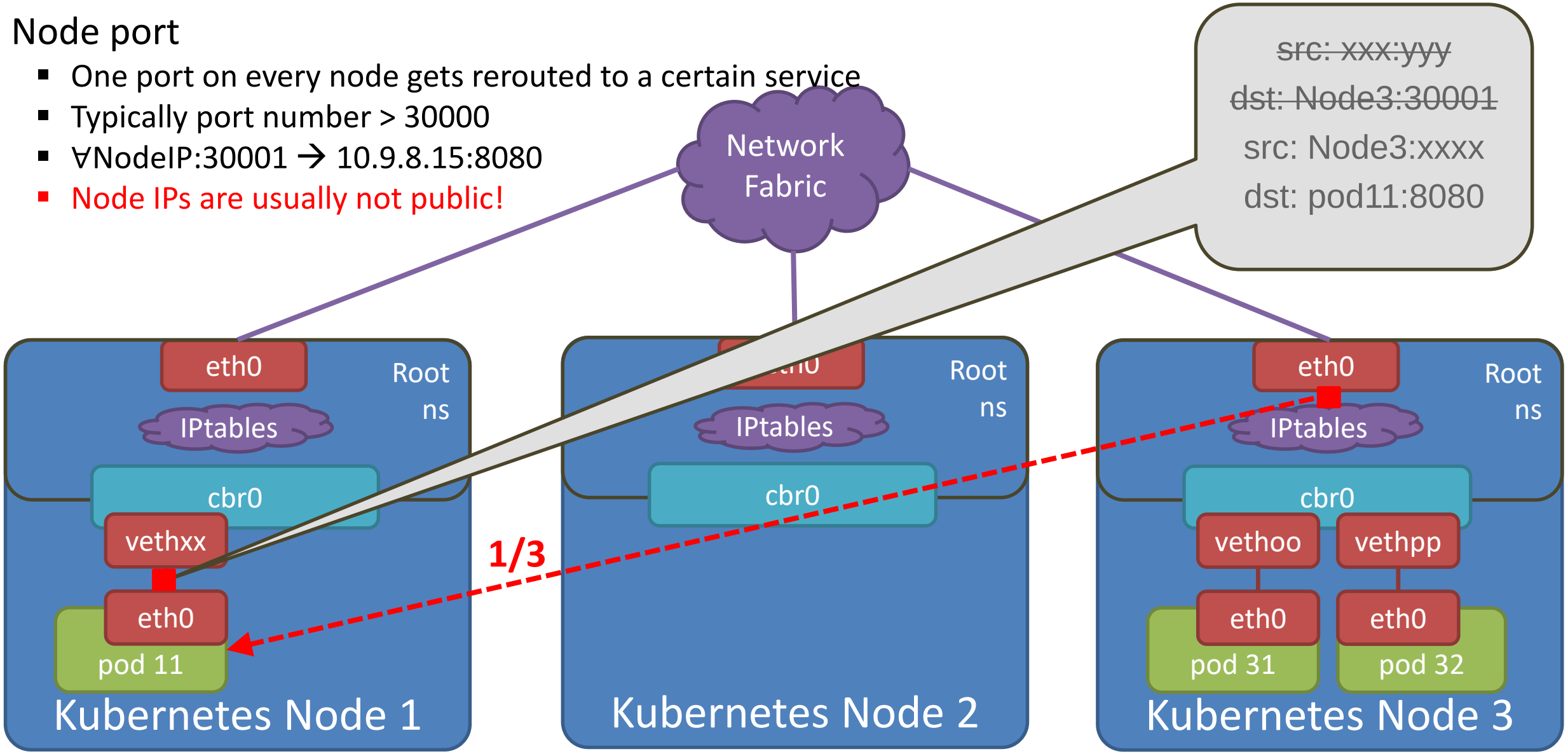
- One port on every node gets rerouted to a certain service
- Typically port number > 30000
- ∀NodeIP:30001 → 10.9.8.15:8080
- Node IPs are usually not public!



External-to-Internal Traffic

Node port

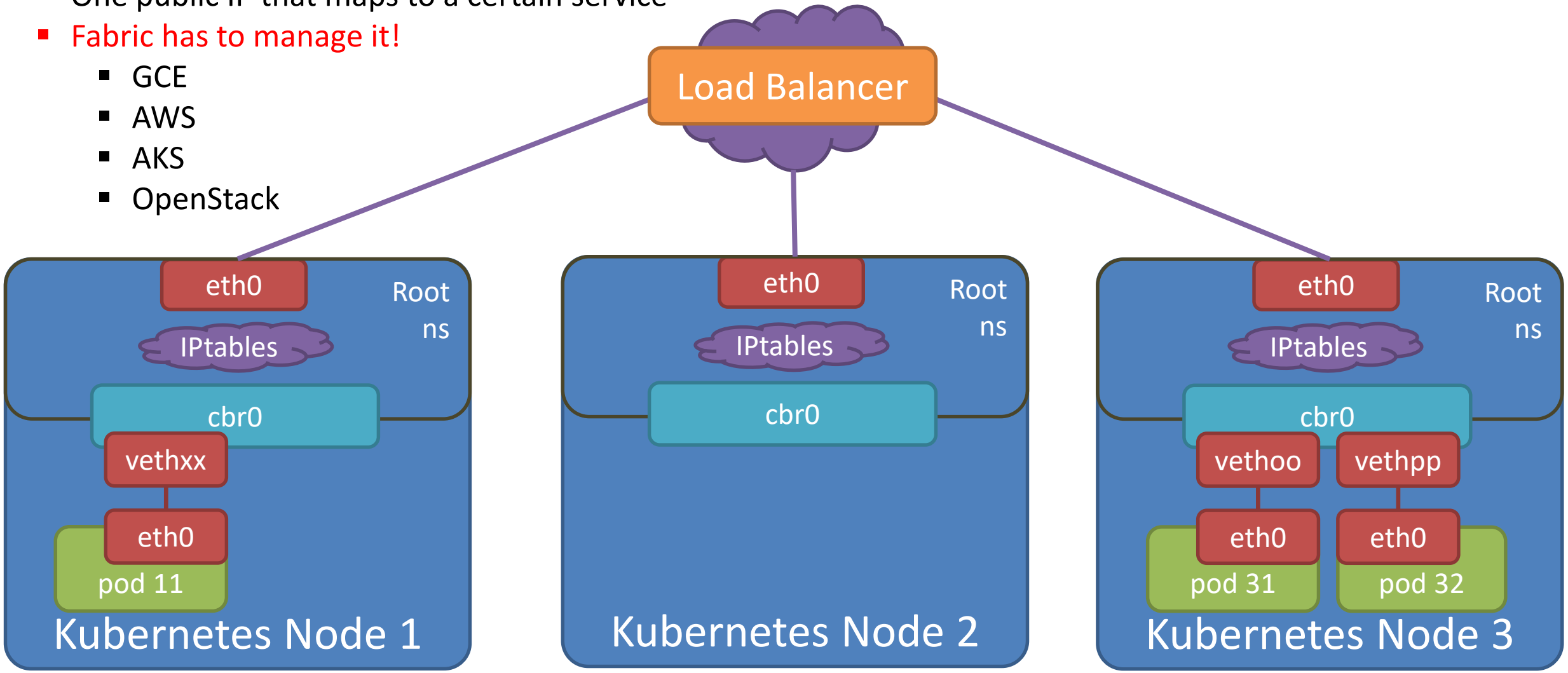
- One port on every node gets rerouted to a certain service
- Typically port number > 30000
- ∀NodeIP:30001 → 10.9.8.15:8080
- Node IPs are usually not public!



External-to-Internal Traffic

Load Balancer

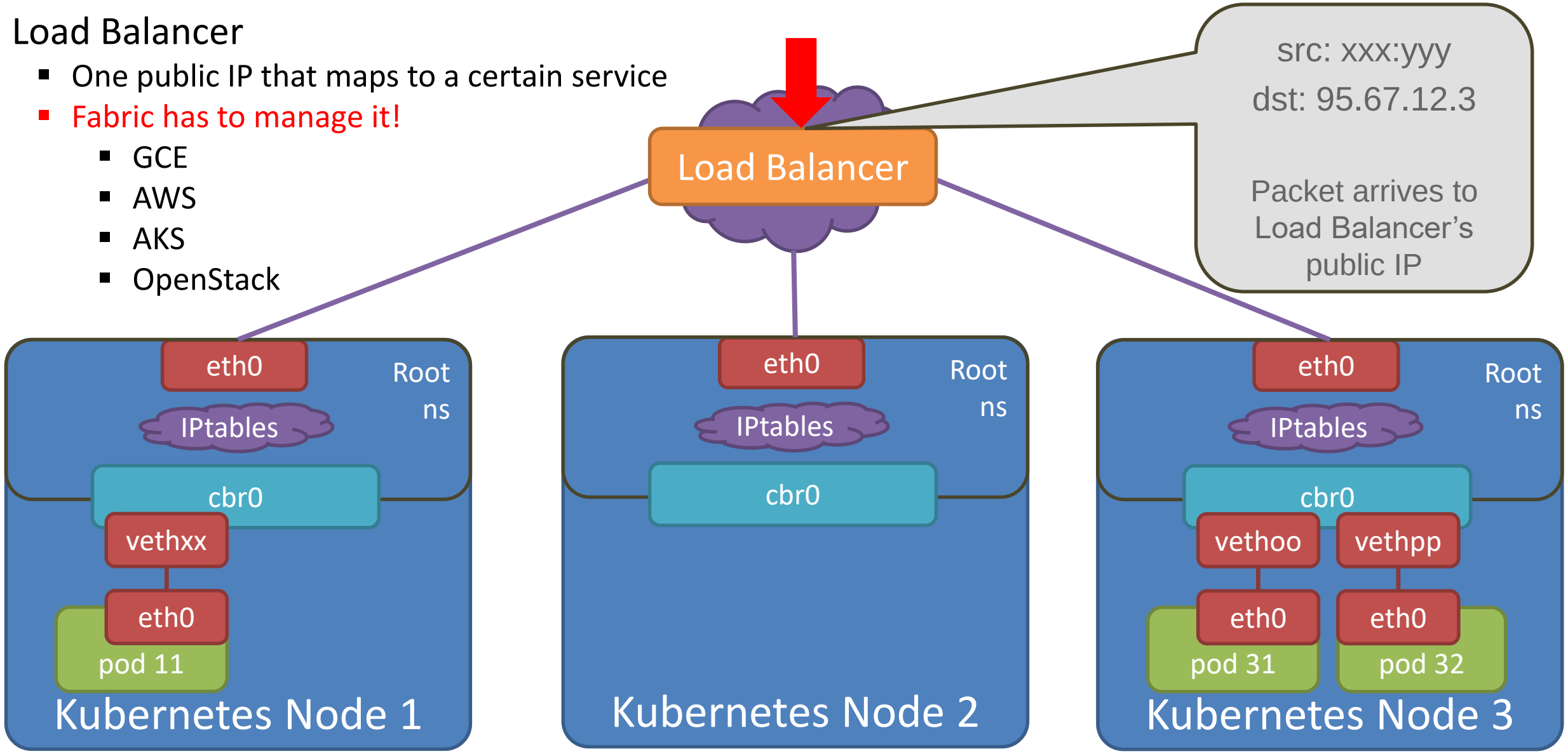
- One public IP that maps to a certain service
- Fabric has to manage it!
  - GCE
  - AWS
  - AKS
  - OpenStack



External-to-Internal Traffic

Load Balancer

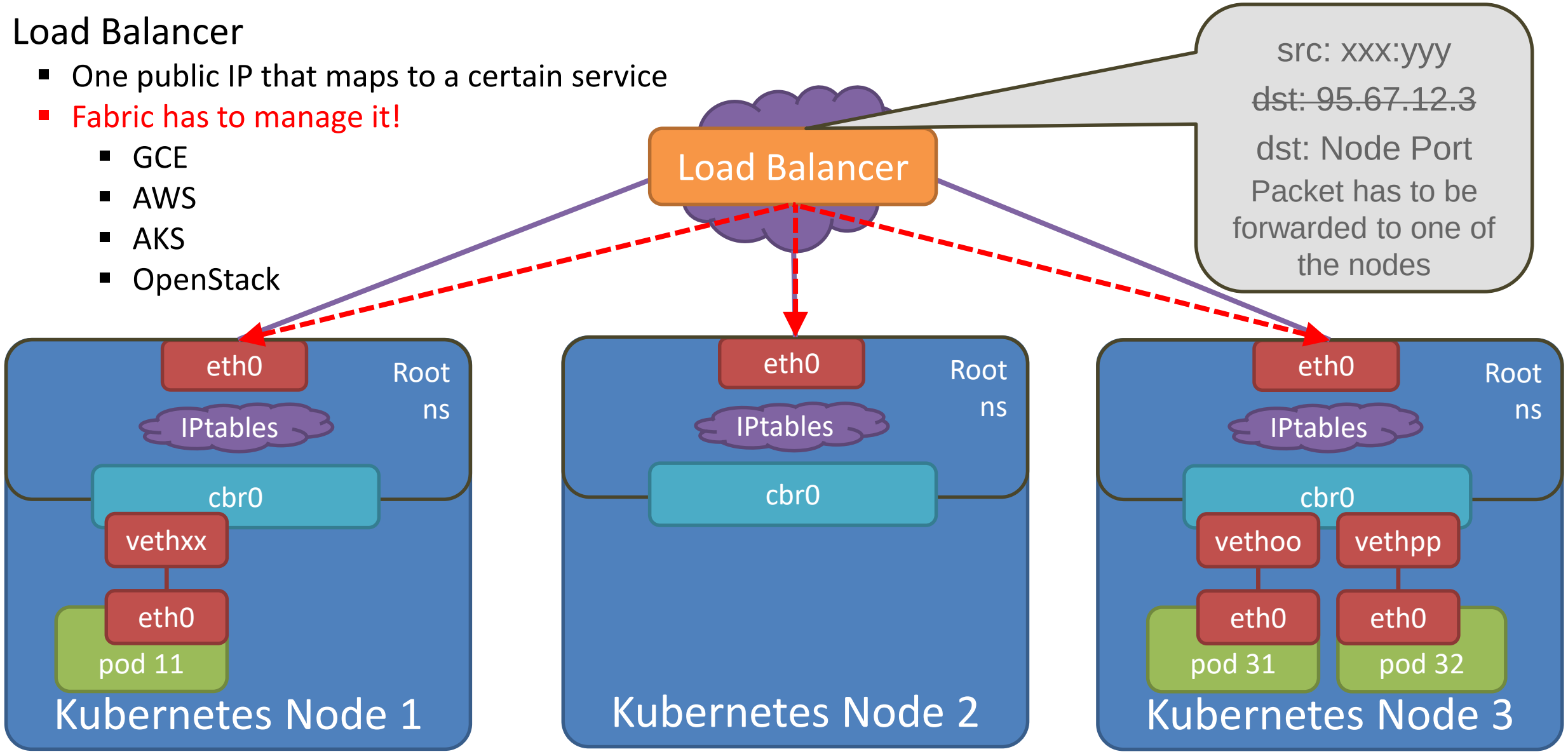
- One public IP that maps to a certain service
- Fabric has to manage it!
  - GCE
  - AWS
  - AKS
  - OpenStack



External-to-Internal Traffic

Load Balancer

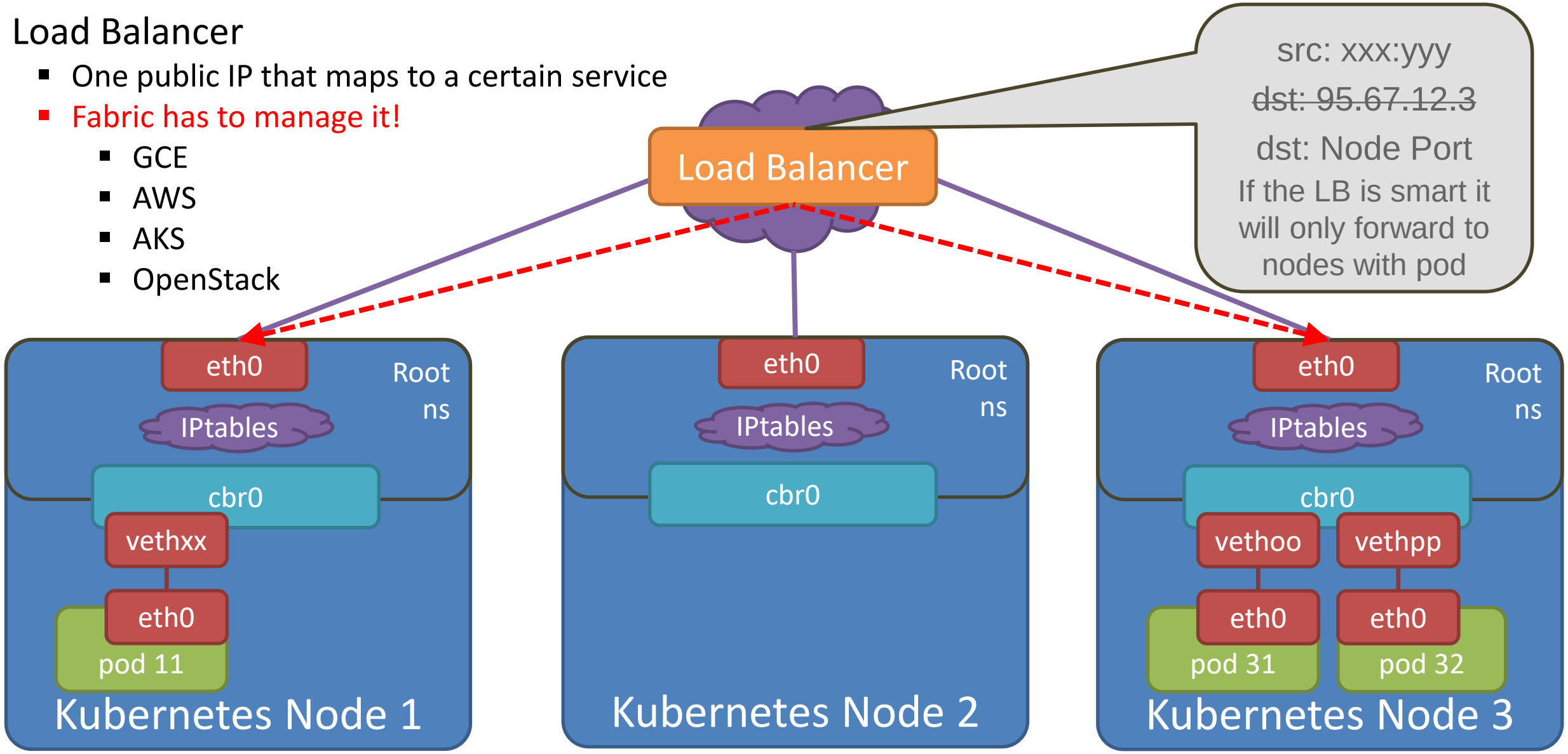
- One public IP that maps to a certain service
- Fabric has to manage it!
  - GCE
  - AWS
  - AKS
  - OpenStack



External-to-Internal Traffic

Load Balancer

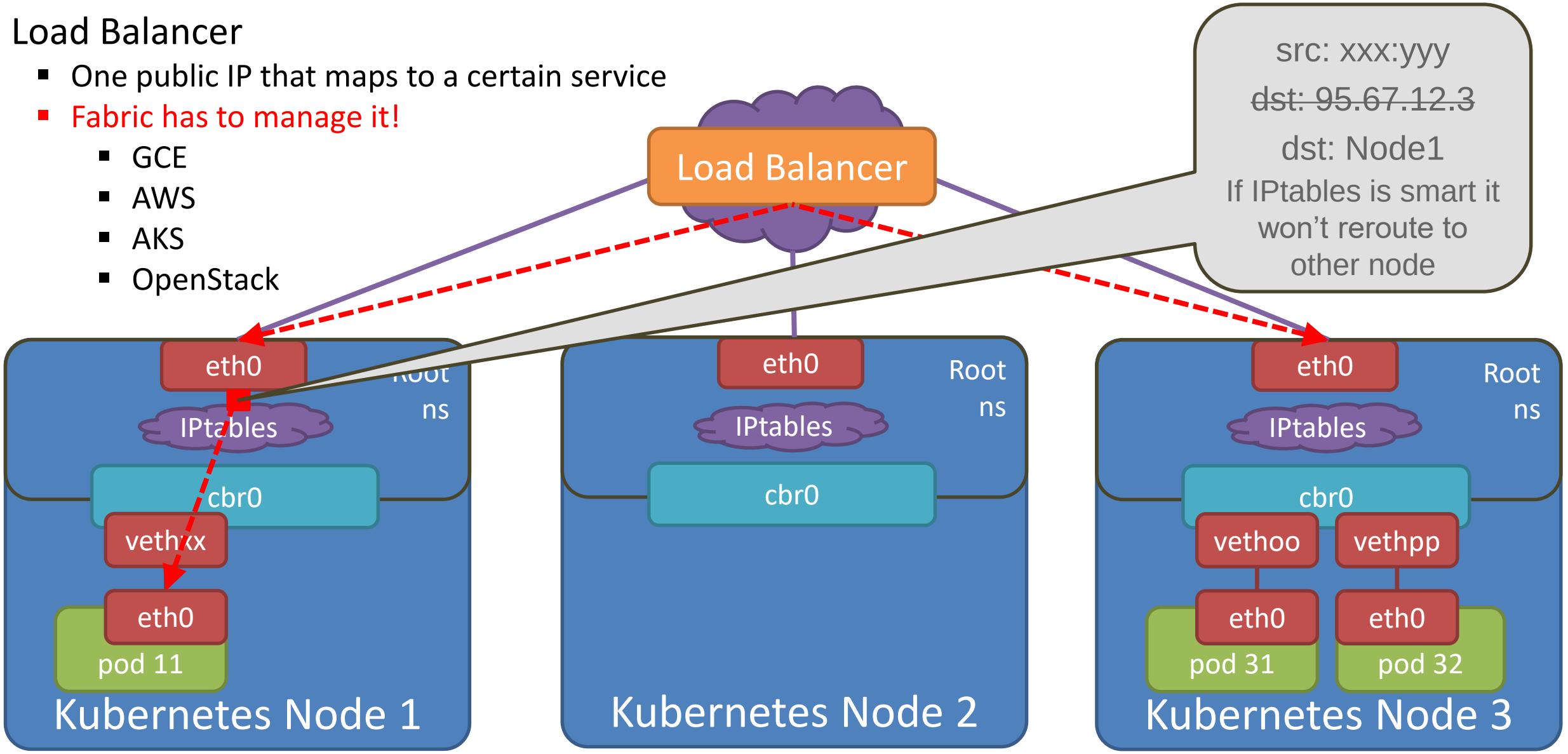
- One public IP that maps to a certain service
- Fabric has to manage it!
  - GCE
  - AWS
  - AKS
  - OpenStack



External-to-Internal Traffic

Load Balancer

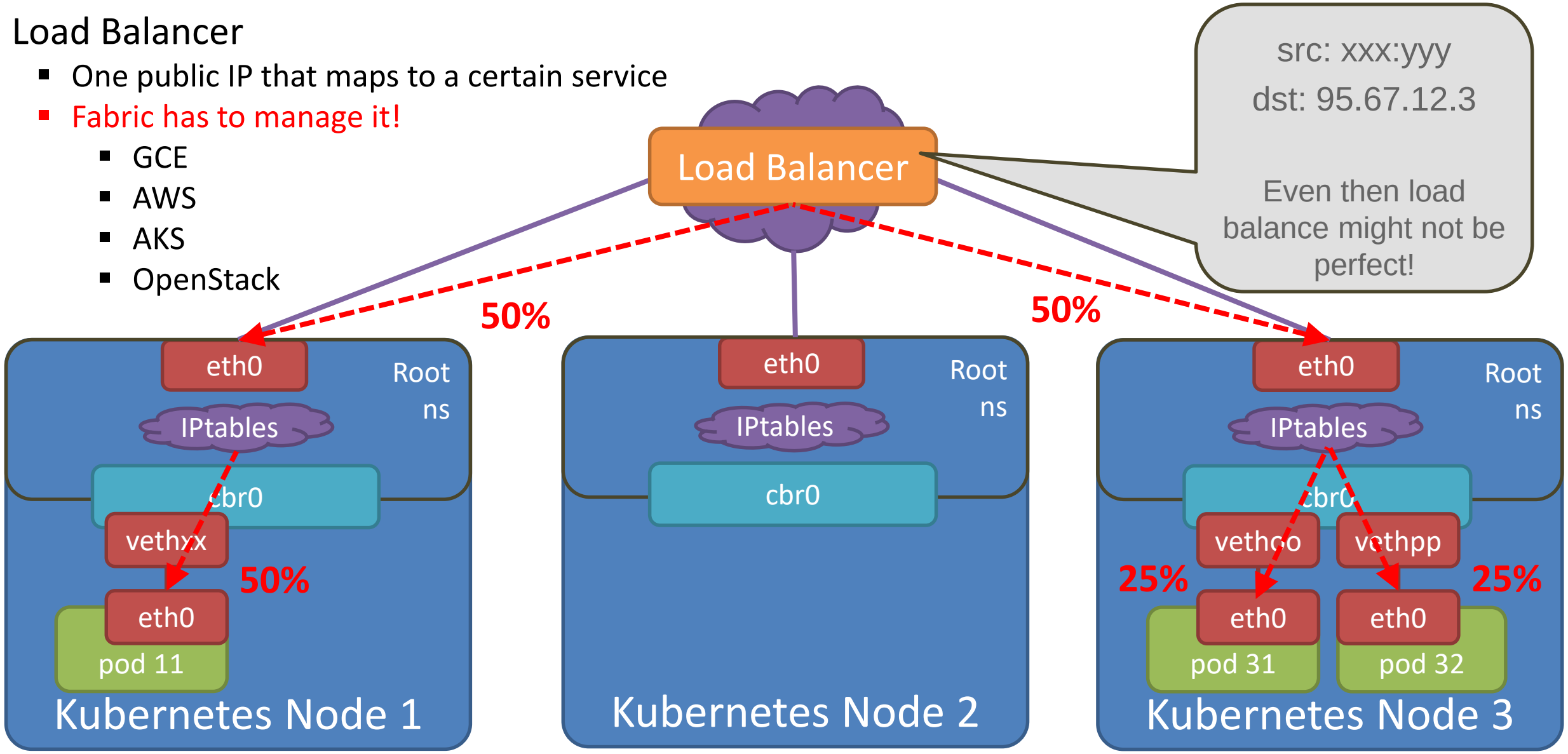
- One public IP that maps to a certain service
- Fabric has to manage it!
  - GCE
  - AWS
  - AKS
  - OpenStack



External-to-Internal Traffic

Load Balancer

- One public IP that maps to a certain service
- Fabric has to manage it!
  - GCE
  - AWS
  - AKS
  - OpenStack



Ingress

- A LoadBalancer type Service alternative
- Separate API Object (`kind: Ingress`)
- Manages external access to the Services in a cluster, typically HTTP.
  - A LoadBalancer Service does it only at the TCP level
- May provide load balancing, SSL termination and name-based virtual hosting.
- Must have an ingress controller to satisfy an Ingress:
  - There are a number of which you can choose (e.g. ingress-nginx)
  - All fit the reference specification, but in reality there are some differences
- Ingress rules:
  - Optional Host (if not specified, rule applied to all inbound HTTP)
  - List of paths (e.g. /testpath) and their associated backends
  - Backend: Service name and port
- Several Ingress Types:
  - Single Service Ingress:
    - specifying a *default backend* with no rules
  - Simple fanout:
    - single IP address to more than one Service
  - Name based virtual hosting:
    - routing HTTP traffic to multiple host names at the same IP address

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: test-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path: /testpath
        pathType: Prefix
        backend:
          serviceName: test
          servicePort: 80
```

Ingress Just Hit Stable Status in 1.19

1.14 - 1.18

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: test-ingress
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path: /testpath
        pathType: Prefix
        backend:
          serviceName: test
          servicePort: 80
```

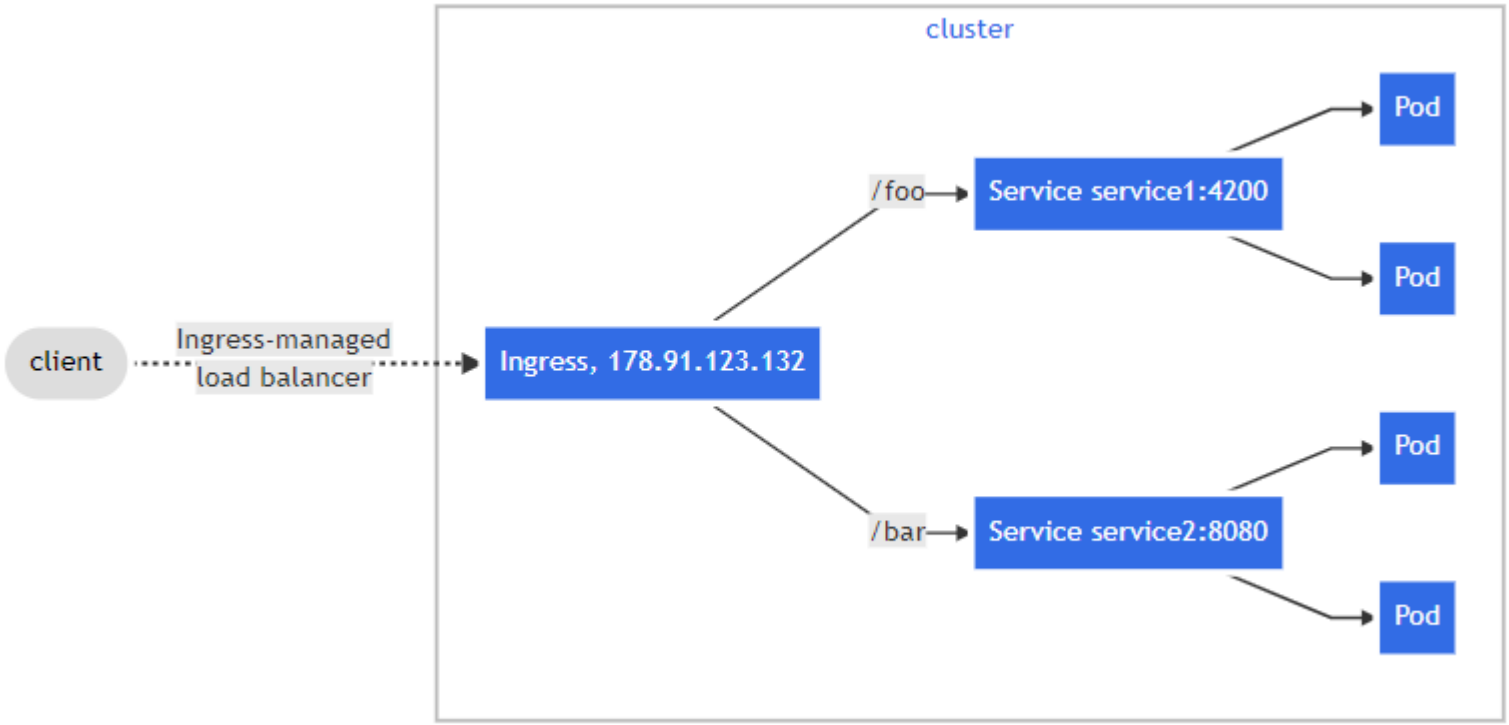
1.19 and after

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: test-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx
  rules:
  - http:
      paths:
      - path: /testpath
        pathType: Prefix
        backend:
          service:
            name: test
            port:
              number: 80
```

Nginx specific annotations:
https://kubernetes.github.io/ingress-nginx/user-guide/nginx-configuration/annotations/

Simple Fanout with Ingress

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: simple-fanout-example
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - path: /foo
        pathType: Prefix
        backend:
          service:
            name: service1
            port:
              number: 4200
      - path: /bar
        pathType: Prefix
        backend:
          service:
            name: service2
            port:
              number: 8080
```



Name Based Virtual Hosting

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: name-virtual-host-ingress
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - pathType: Prefix
        path: "/"
        backend:
          service:
            name: service1
            port:
              number: 80
  - host: bar.foo.com
    http:
      paths:
      - pathType: Prefix
        path: "/"
        backend:
          service:
            name: service2
            port:
              number: 80
```