# Container Networking

# About me

- Amr Metwally (24 years old)
- Backend developer at LYNQTECH
- Currently doing masters in distributed systems in TU Dortmund
- https://github.com/magdyamr542/container-networking-presentation

# Goal of the talk

- Understand basics of linux networking
- Get to know some (there are many) approaches of how containers talk to each other
  - On the same host
  - On different hosts (distributed system)

# Why care about container networking?

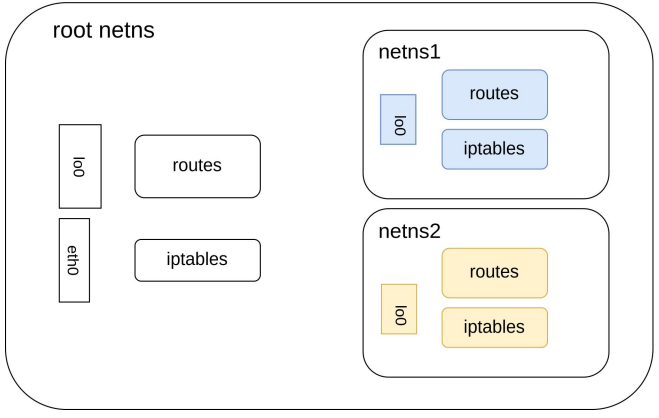
- Reach containers from the internet
  - Access the services they provide
- Enable containers to talk to the internet
  - They might depend on other software
- Inter-container connectivity (Microservices)
- Load balance traffic across different containers (Scaling)

# Agenda

- Some basics
  - OSI
  - NIC
  - Routing
  - NAT
- IPTABLES
- Network namespaces
  - Veth pairs
  - Virtual bridges
- K8s
  - Architecture
  - Pod to Pod networking
  - Service networking
  - DNS

# Theme of the talk

1. Explain some concept
2. Show a live demo

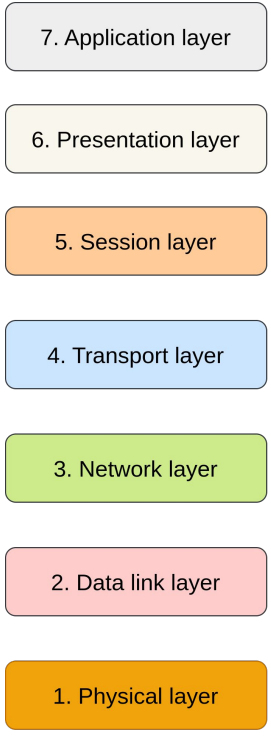



```
~/.config/config  main !2  sudo iptables -t nat -L
[sudo] password for amr:
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
DOCKER     all  --  anywhere             anywhere             ADDRTYPE match dst-type LOCAL

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
DOCKER     all  --  anywhere            !localhost/8          ADDRTYPE match dst-type LOCAL

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
MASQUERADE  all  --  172.21.0.0/16        anywhere
MASQUERADE  all  --  172.17.0.0/16        anywhere
MASQUERADE  all  --  172.18.0.0/16        anywhere

Chain DOCKER (2 references)
target     prot opt source               destination
RETURN     all  --  anywhere             anywhere
RETURN     all  --  anywhere             anywhere
~/.config/config  main !2
```

# OSI

# The Open Systems Interconnection (OSI Model)

- Model from the **International Organization for Standardization** (ISO)
  - Create standards for system interconnection
- 7 layers computers can use to communicate over network
- Interesting for us: L2, L3




Determine the transmission path of data using routing protocols

Format data on the network and send it from node to node

Transmit raw bit stream over physical medium

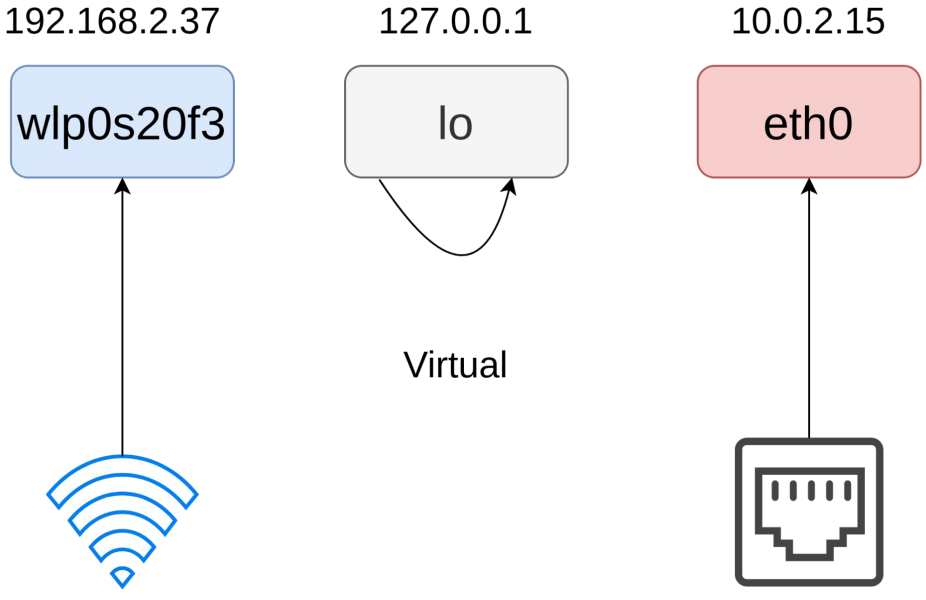
# NIC

# Network interface controller (NIC)

- Hardware device connecting a computer to a network
- Connection is either through cable or wireless
- Has a unique mac address assigned by manufacturer
- Can have an IP address

# Linux network interface

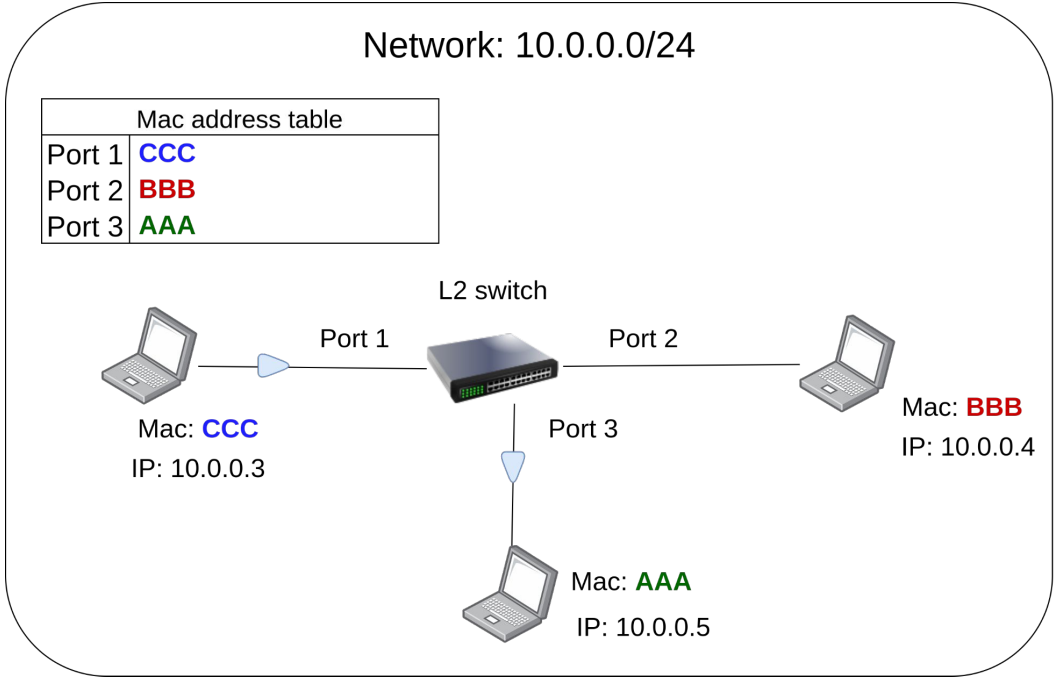
- Link between software and hardware side of networking
- Can be associated with a NIC
- Can be virtual
- Enables the kernel to talk to the NIC



# Routing

# L2 switches

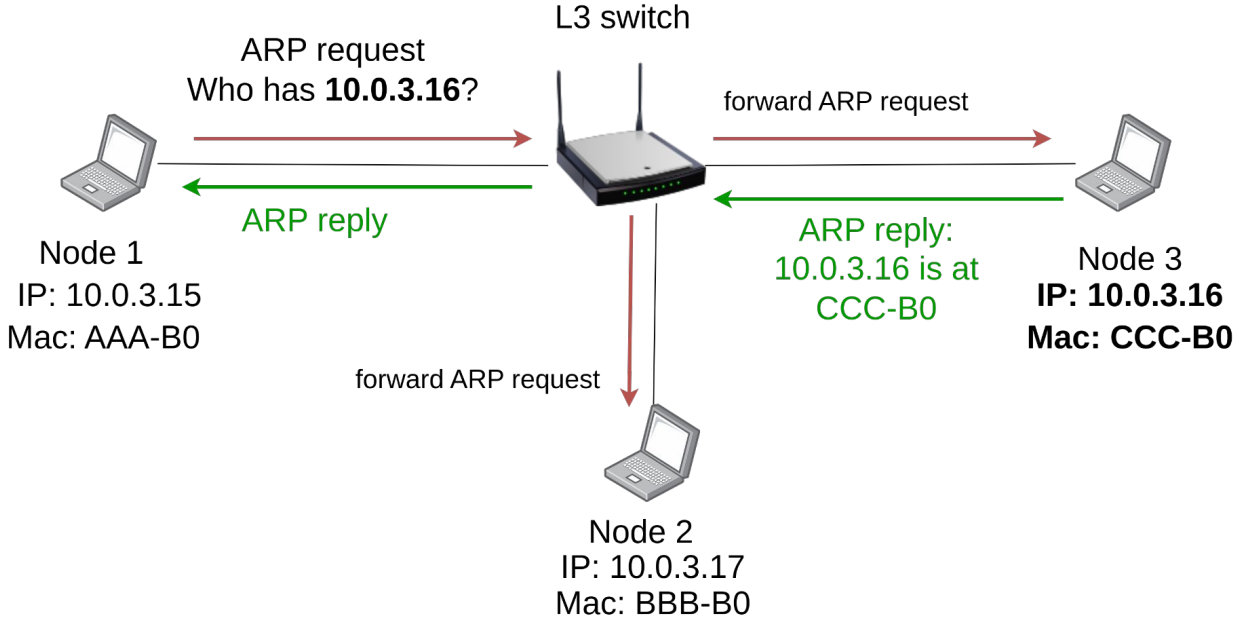
- A L2 switch forwards packets based on mac address
  - Mac address uniquely identifies a network device in a network
- Mac address table: maps a **port** to a **mac address**
- Enables connectivity in a local area network **LAN**
- Assumption: we know the mac address



# L3 switches: ARP

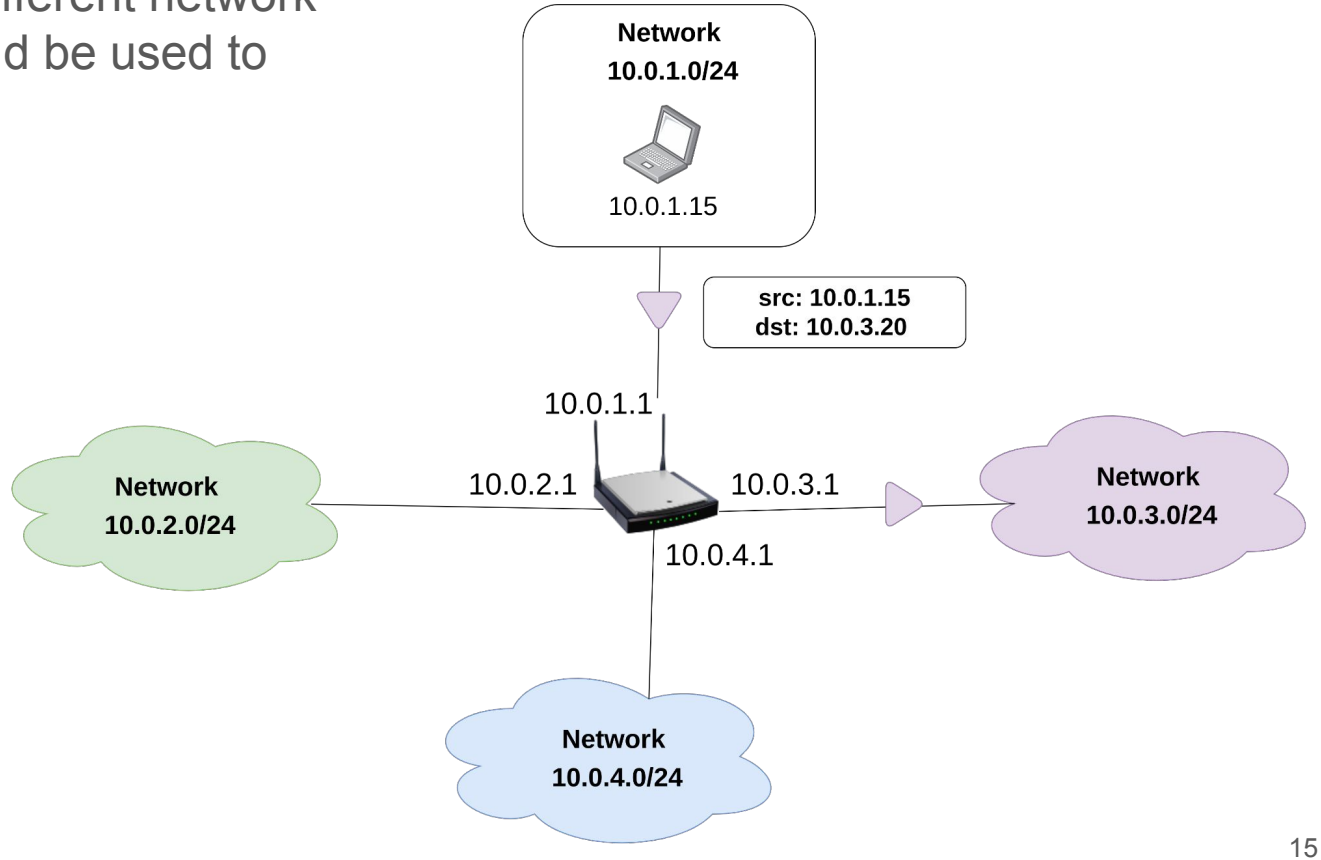
- IP address known, mac address unknown
- Address resolution protocol **ARP**
  - Get mac address for a certain IP address
  - ARP request: Broadcast who has mac for ip …?
  - ARP reply: ip … ist at this mac …
- L3 switches understand IPs

Network 10.0.3.0/24



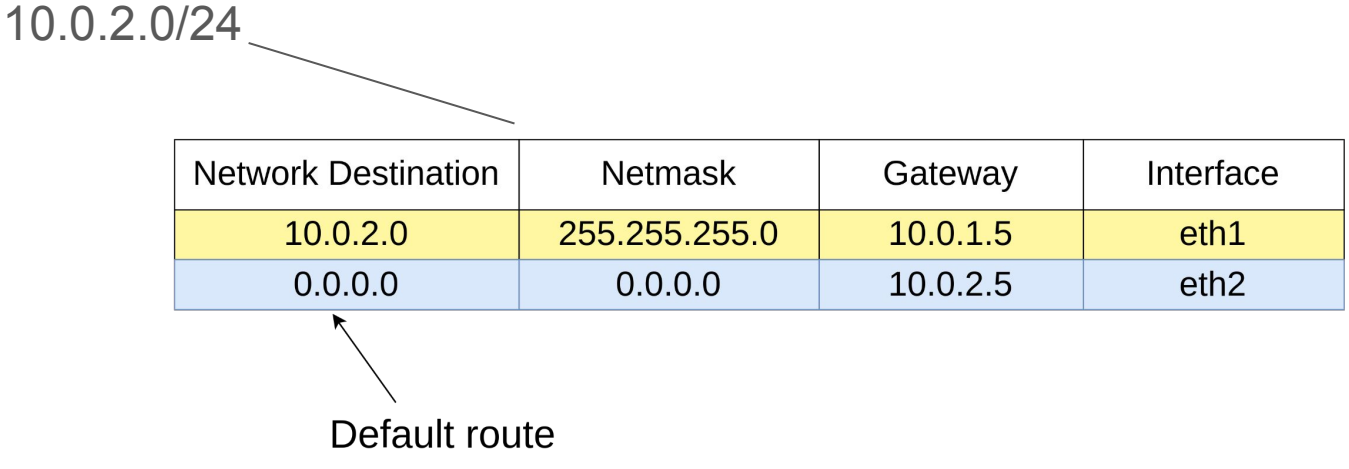
# Routing

- A router connects at least 2 networks
- Each interface is in different network
- Which interface should be used to deliver packets?



# Routing table

- Decide next path for packets to reach the destination
- Components
  - **Network destination + Netmask**: Define the network
  - **Next hop/Gateway**: next immediate router in the path to reach the destination
  - **Interface**: Outgoing interface used to talk to the gateway
- **Default route**: The route used whl
- en no match is found

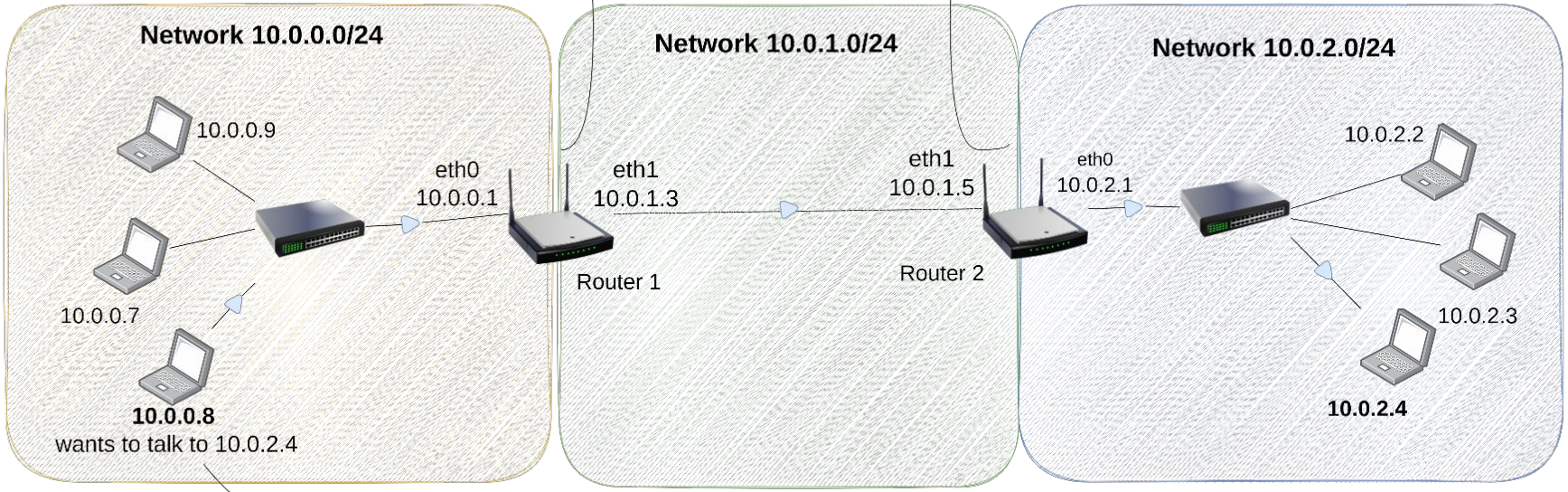


# L3 routing

Routers exchange network information with e.g **BGP**

| Network | Next Hop | Interface |
|---|---|---|
| 10.0.0.0/24 | Directly connected | eth0 |
| 10.0.1.0/24 | Directly connected | eth1 |
| **10.0.2.0/24** | **10.0.1.5** | **eth1** |

| Network | Next Hop | Interface |
|---|---|---|
| 10.0.0.0/24 | 10.0.1.3 | eth1 |
| 10.0.1.0/24 | Directly connected | eth1 |
| **10.0.2.0/24** | **Directly connected** | **eth0** |

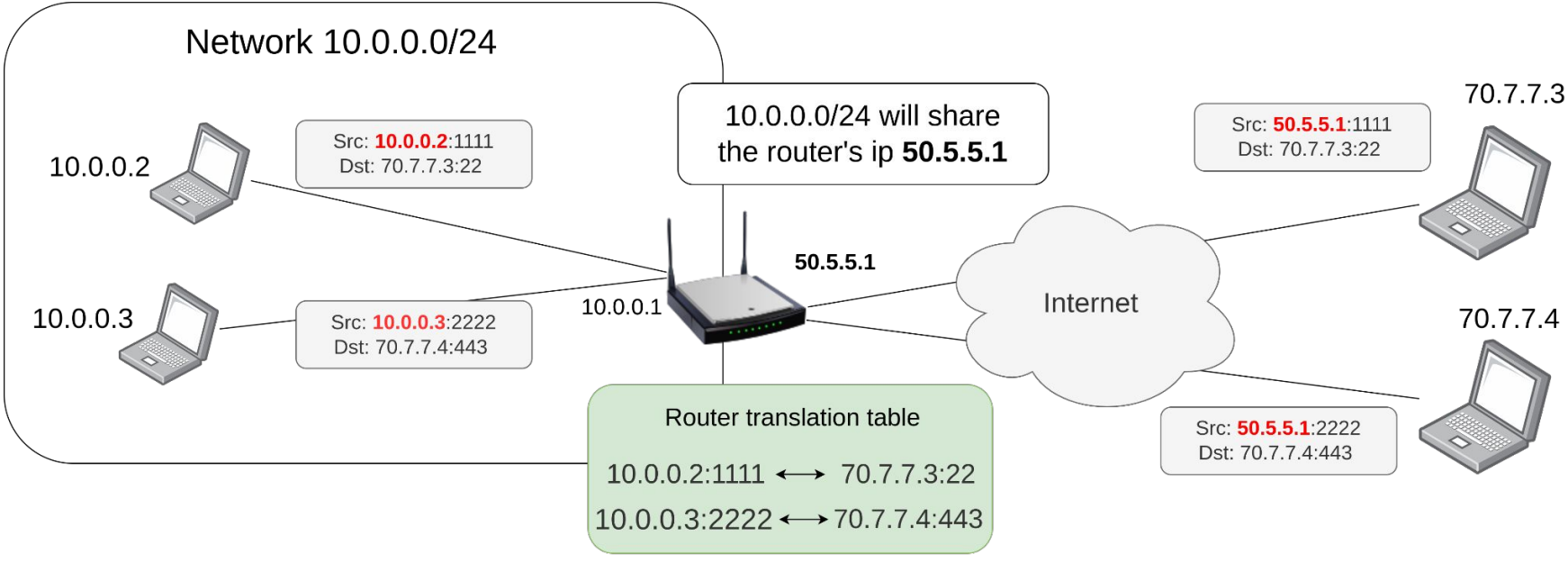



| Network Destination | Netmask | Gateway | Interface |
|---|---|---|---|
| 0.0.0.0 | 0.0.0.0 | 10.0.0.1 | eth0 |

# NAT

# Network address translation NAT

- Map multiple private addresses to a public address



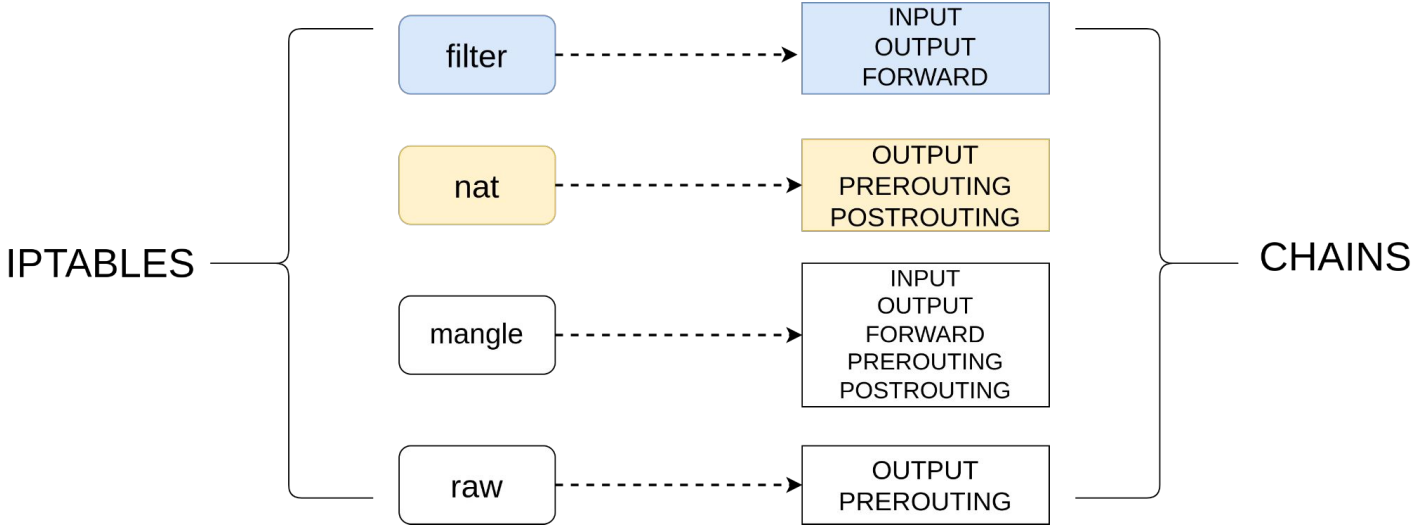
# Agenda

- Some basics ✅
  - OSI ✅
  - NIC ✅
  - Routing ✅
  - NAT ✅
- IPTABLES
- Network namespaces
  - Veth pairs
  - Virtual bridges
- K8s
  - Architecture
  - Pod to Pod networking
  - Service networking
  - DNS

# IPTABLES

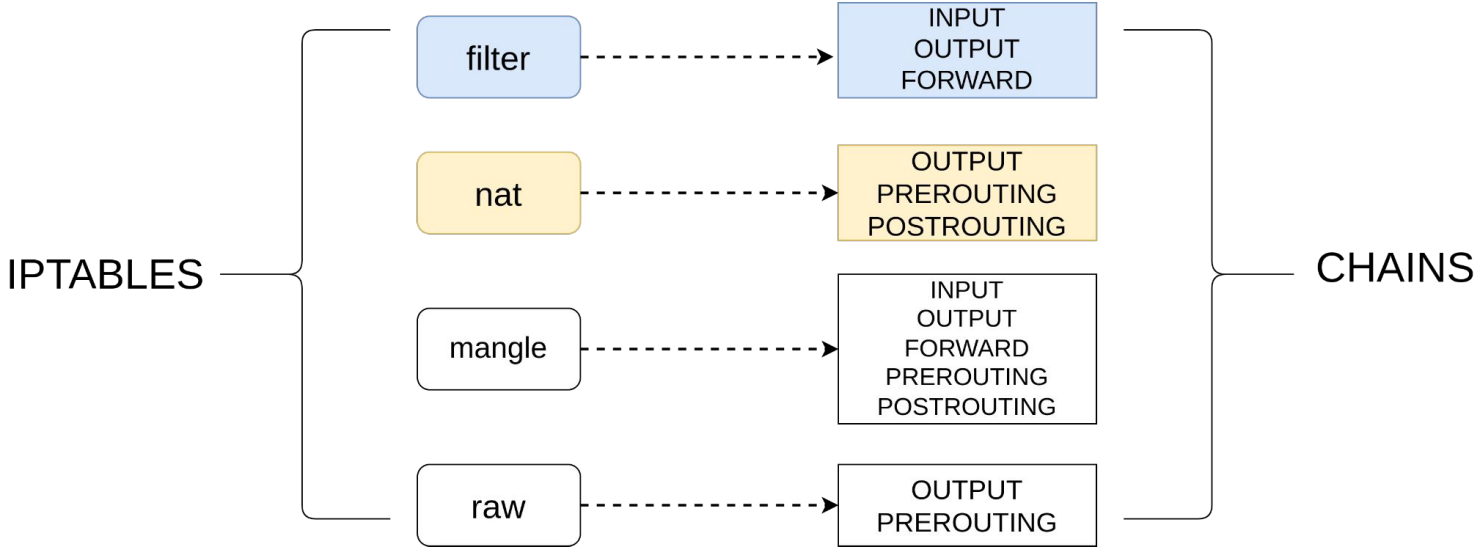
# IPTABLES: What does it do?

- Administration tool for IP packet filtering and NAT rules in the Linux kernel
- **Tables** contain built-in/user-defined **chains**
- Each **chain** is a list of **rules** that can match packets
- Each **rule** has a **target** specifying **what to do** with the packet
  - Built in targets
  - Accept: let packet through
  - Drop: ignore the packet
  - Jump: go to another chain
  - Return: stop traversing this chain

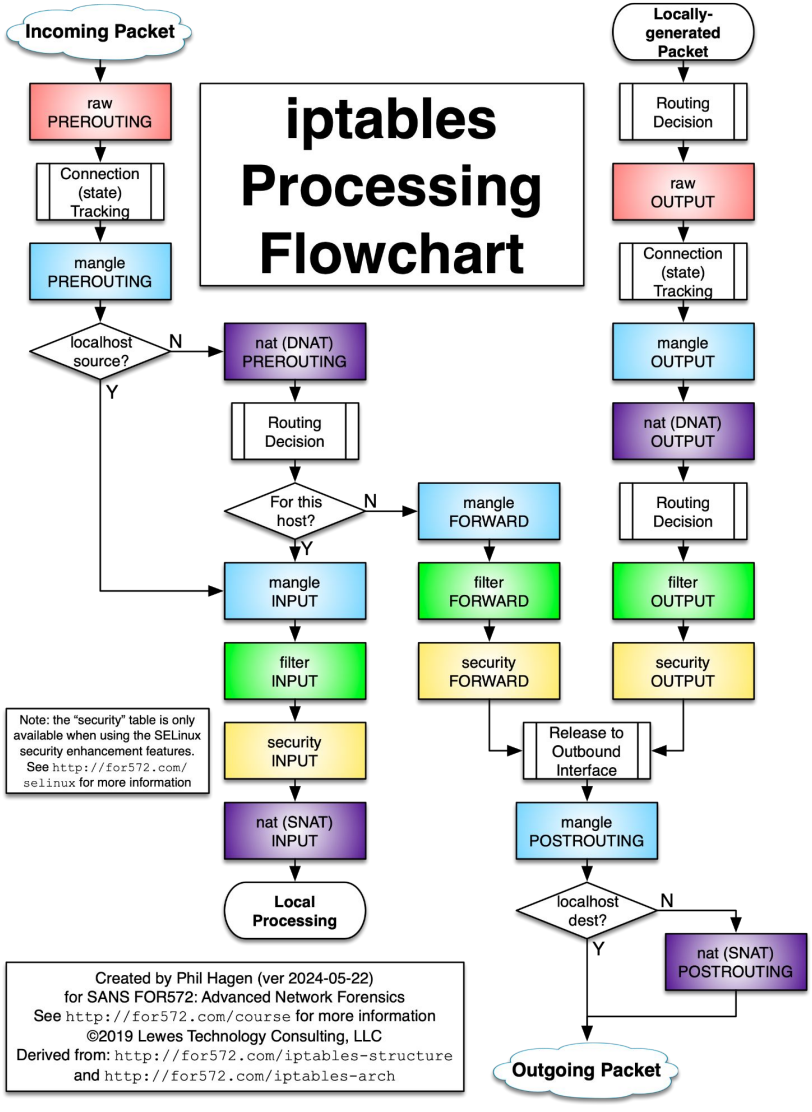


# IPTABLES: the tables

- **Filter**:  used to filter packets
  - Suitable for Firewall rules
- **Nat**: used when packet creates a new connection
  - Used for network address translation
- Mangle: special packet altering
- Raw: configures exemptions from connection tracking

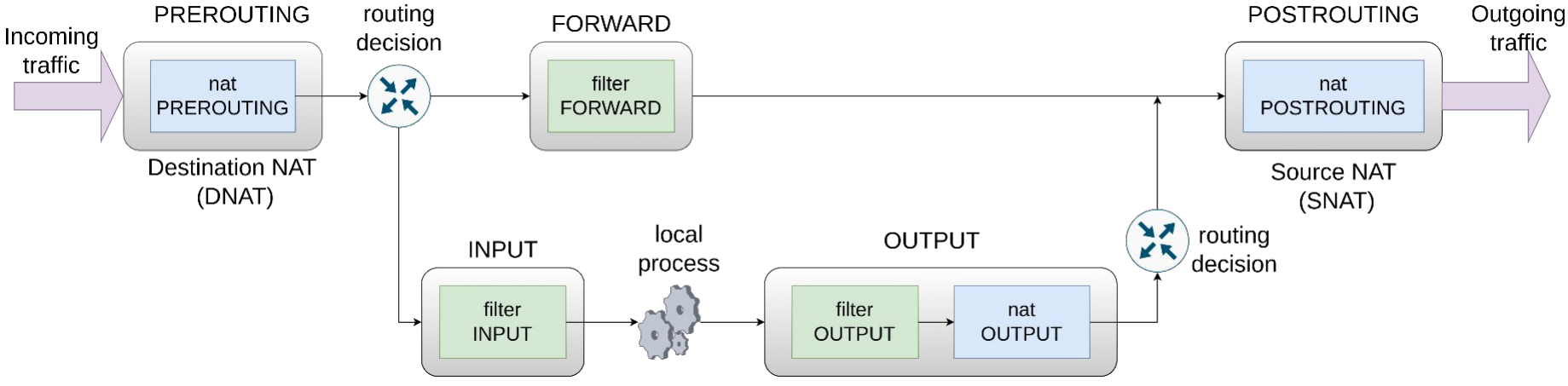


# IPTABLES packet flow



# IPTABLES packet flow

- PREROUTING: alter packets as soon as they reach the host
- POSTROUTING: alter packets before they leave the host
- INPUT: alter packets destined for local processes
- OUTPUT: alter packets generated by local processes
- FORWARD: alter packets forwarded by the host



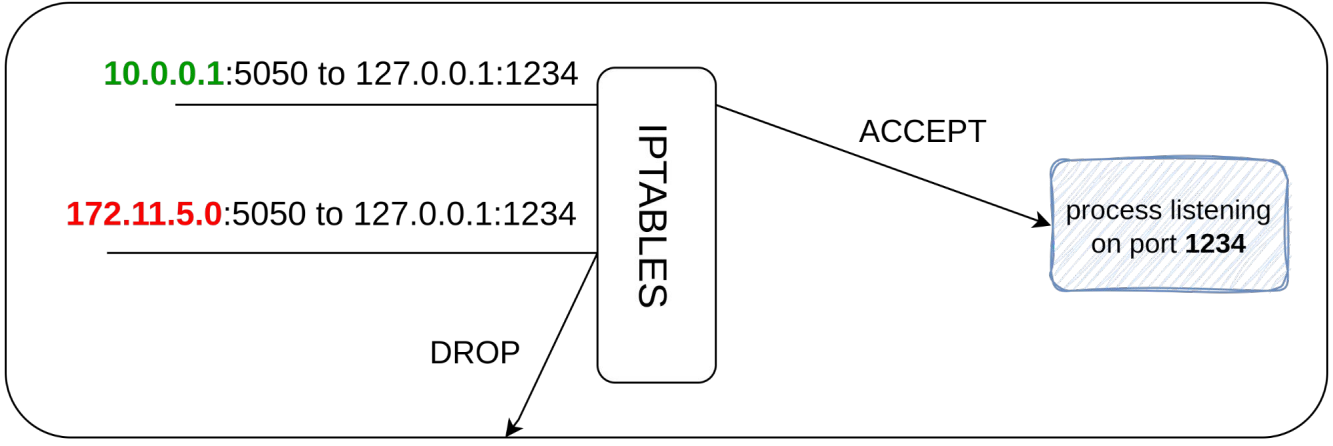
# IPTABLES
# Use Cases

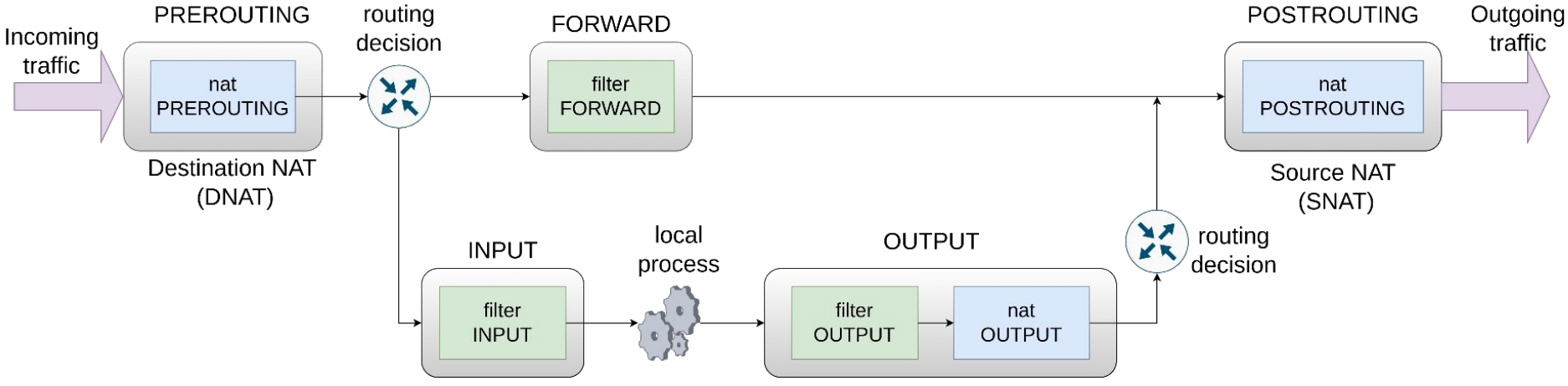
# IPTABLES as a Firewall

- Accept packets from 10.0.0.1
- Reject otherwise (target **DROP**)

```
iptables -A INPUT -p tcp -s 10.0.0.1 --dport 1234 -j ACCEPT
```

```
iptables -A INPUT -p tcp ! -s 10.0.0.1 --dport 1234 -j DROP
```

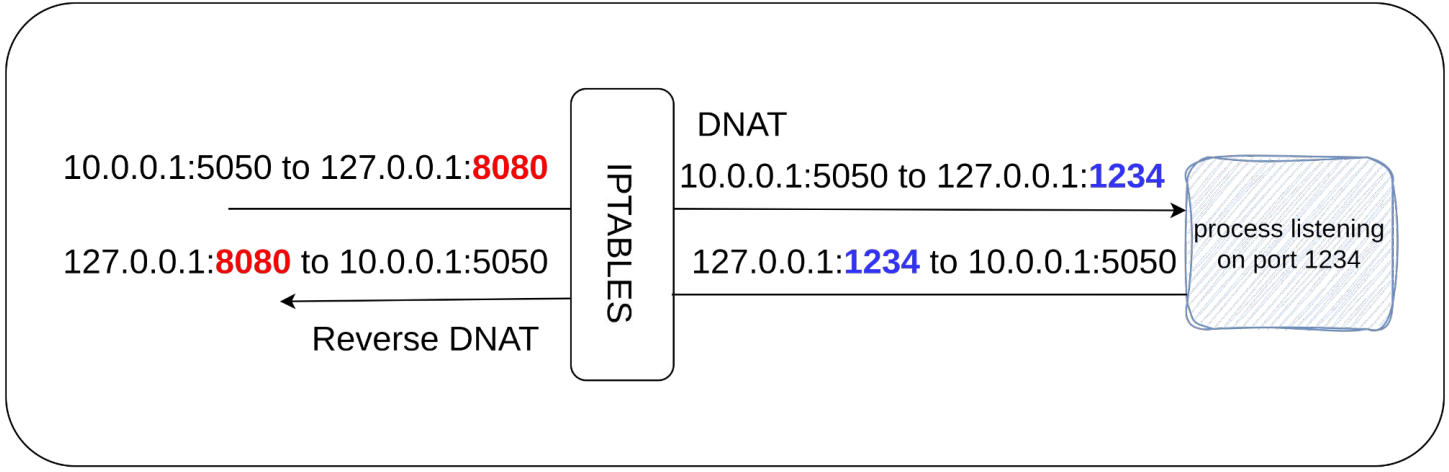


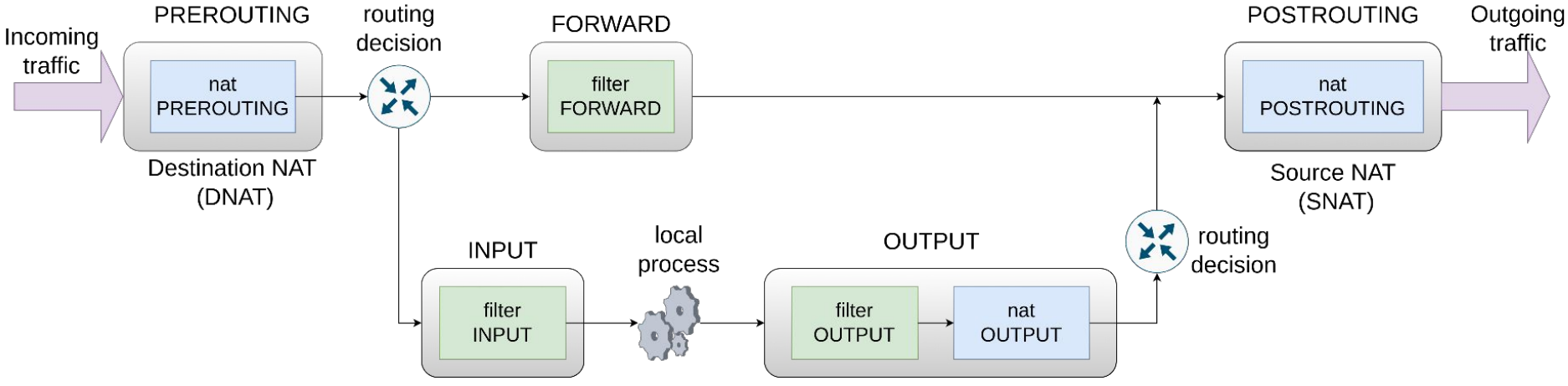

Incoming traffic
PREROUTING
nat PREROUTING
Destination NAT (DNAT)
routing decision
FORWARD
filter FORWARD
POSTROUTING
nat POSTROUTING
Source NAT (SNAT)
Outgoing traffic
INPUT
filter INPUT
local process
OUTPUT
filter OUTPUT
nat OUTPUT
routing decision

# IPTABLES as NAT

- **OUTPUT** chain (packet generated by a local process)
- Change destination to 127.0.0.1:1234

```
iptables -t nat \
 -A OUTPUT \
-p tcp \
--dport 8080 \
-j DNAT --to-destination 127.0.0.1:1234
```




Incoming traffic
PREROUTING
nat PREROUTING
Destination NAT (DNAT)
routing decision
FORWARD
filter FORWARD
POSTROUTING
nat POSTROUTING
Source NAT (SNAT)
Outgoing traffic
INPUT
filter INPUT
local process
OUTPUT
filter OUTPUT
nat OUTPUT
routing decision

# IPTABLES as Load Balancer

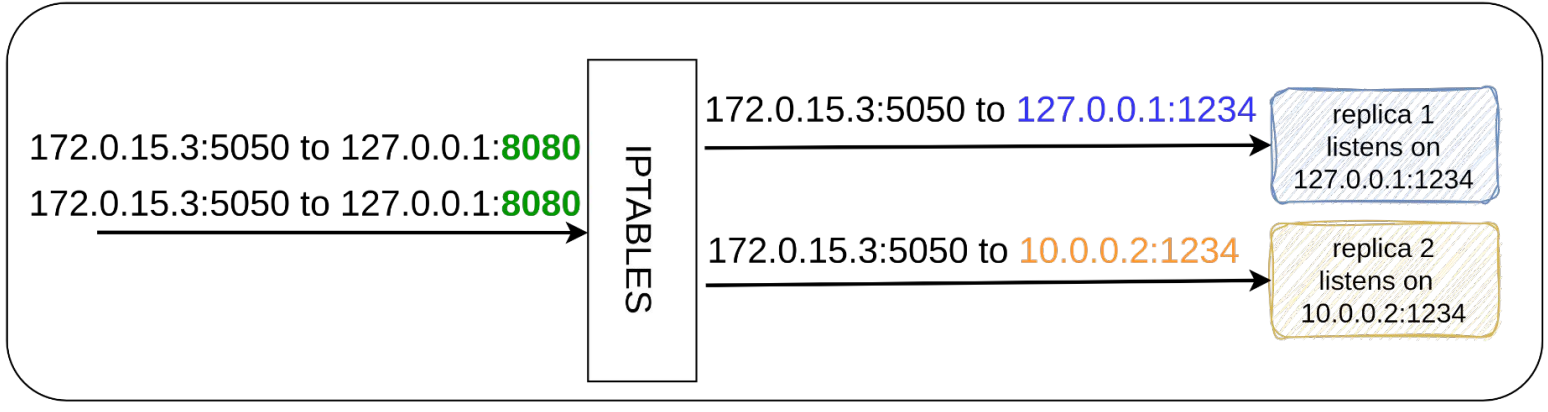
- **OUTPUT** chain (packet coming from a local process)
- Random probability load balancing
- Match each rule with 50% chance

```
iptables -t nat \
 -A OUTPUT \
-p tcp \
--dport 8080 \
-m statistic --mode random --probability 0.5 \
-j DNAT --to-destination 127.0.0.1:1234
```

Use this rule
with 50% probability!

```
iptables -t nat \
 -A OUTPUT \
-p tcp \
--dport 8080 \
-j DNAT --to-destination 10.0.0.2:1234
```

Use this rule otherwise



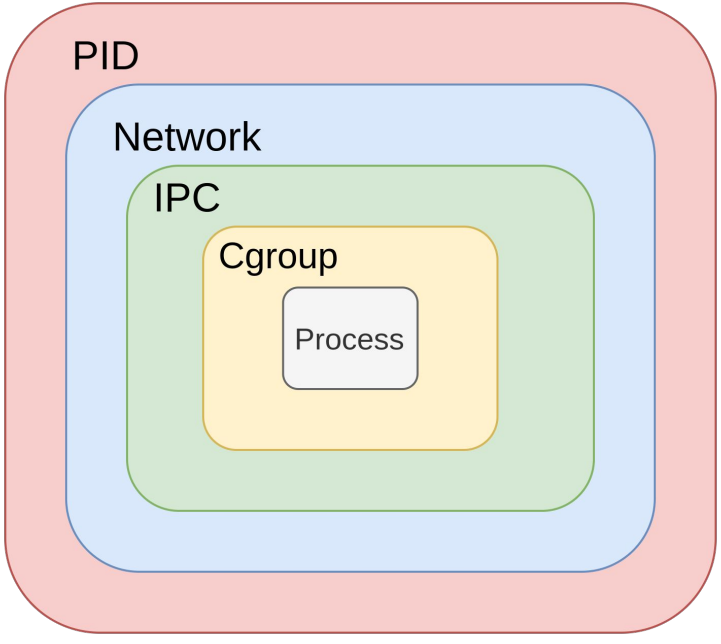
# Agenda

- Some basics ✅
  - OSI ✅
  - NIC ✅
  - Routing ✅
  - NAT ✅
- IPTABLES ✅
- Network namespaces
  - Veth pairs
  - Virtual bridges
- K8s
  - Architecture
  - Pod to Pod networking
  - Service networking
  - DNS

# Network namespaces

# Linux namespaces

- A container is a process inside of one or more namespaces



# Linux namespaces

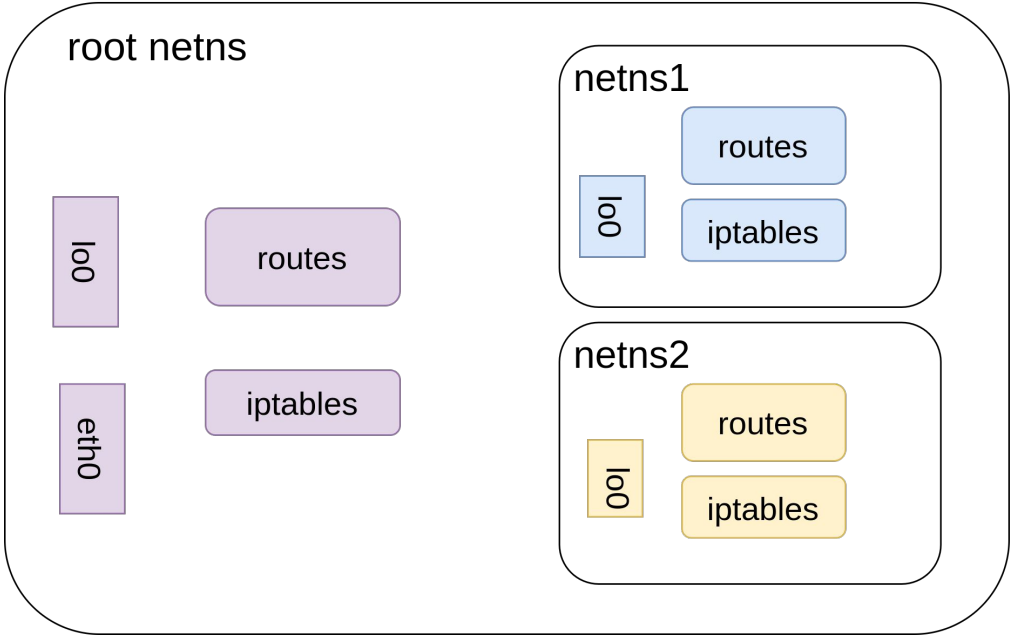
The linux manual says: “A namespace wraps a global system resource in an abstraction that makes it appear to the processes within the namespace that they have their own isolated instance of the global resource. Changes to the global resource are visible to other processes that are members of the namespace, but are invisible to other processes”

Example namespaces:

- Cgroup
  - Virtualize the view of cgroups. Limit usage of resources
- Mount
  - Isolation of mounts seen by processes. Processes in different mount namespace see different directory hierarchies
- **Network**
  - Provide isolation of the system resources associated with networking
  - I.e: network devices, routing tables, firewall rules etc.

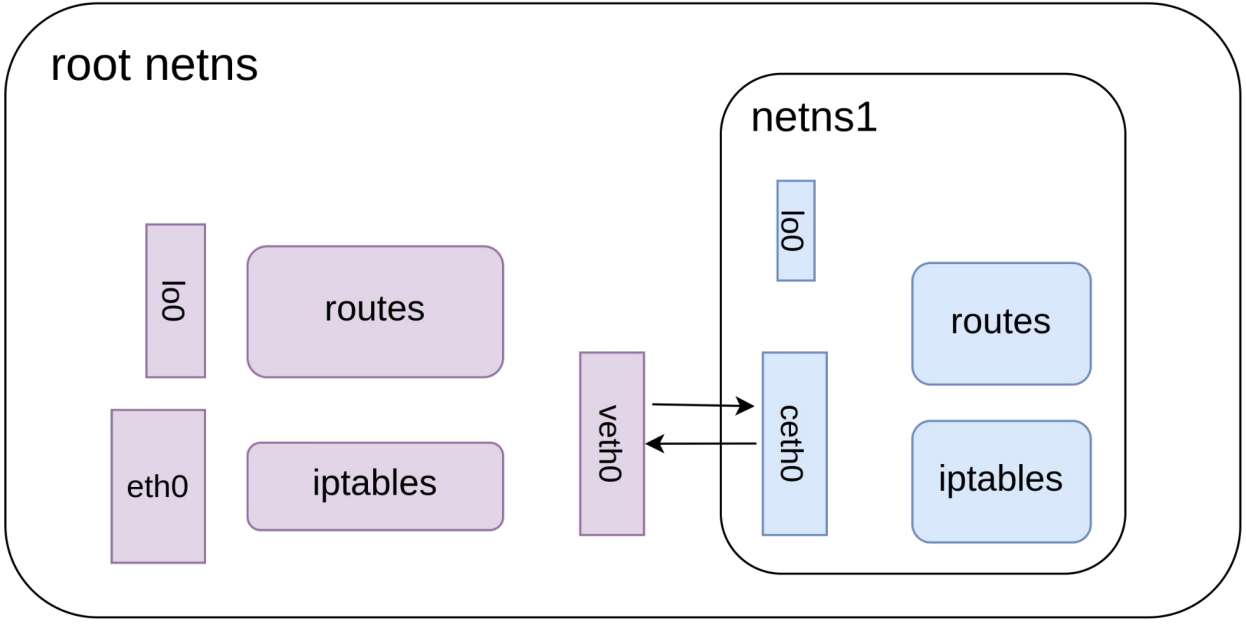
# Isolated network namespaces

- netns1 and netns2 have their own:
  - Routing tables
  - IPTABLES rules
  - Network devices/interfaces
  - Port numbers
- Unreachable from the root ns (Isolation)

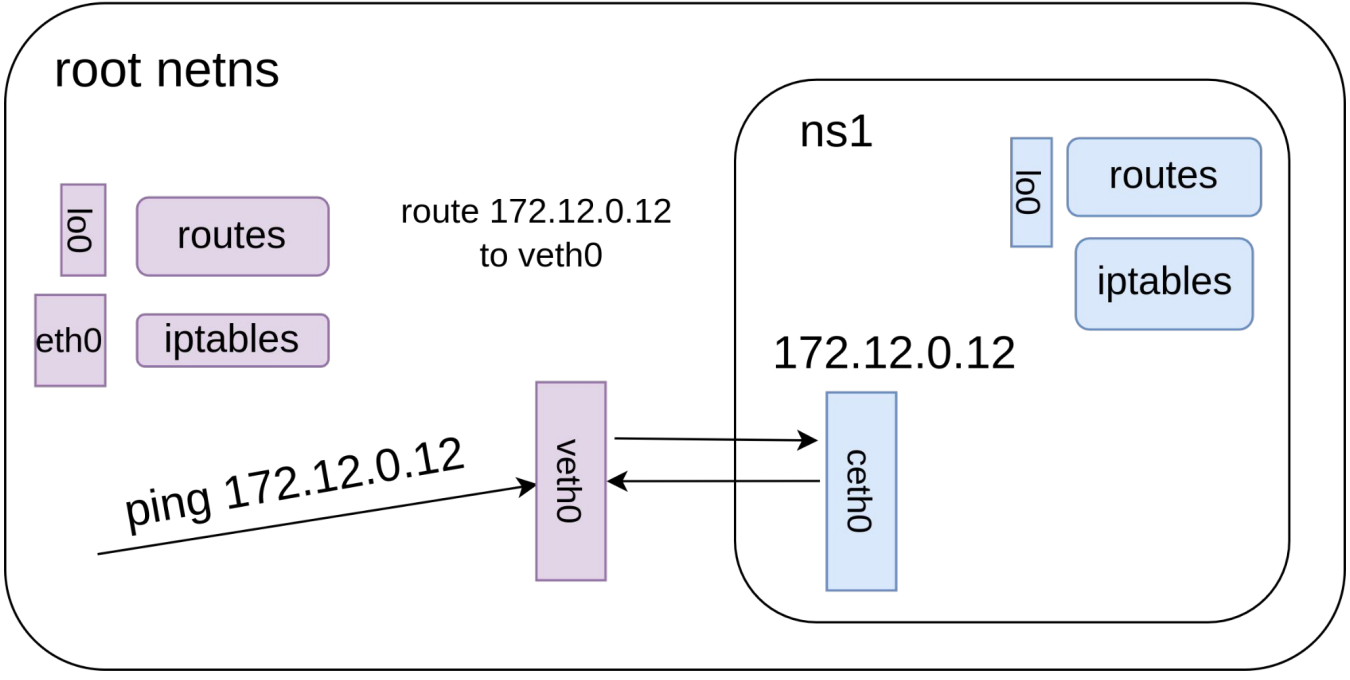


# Virtual ethernet devices

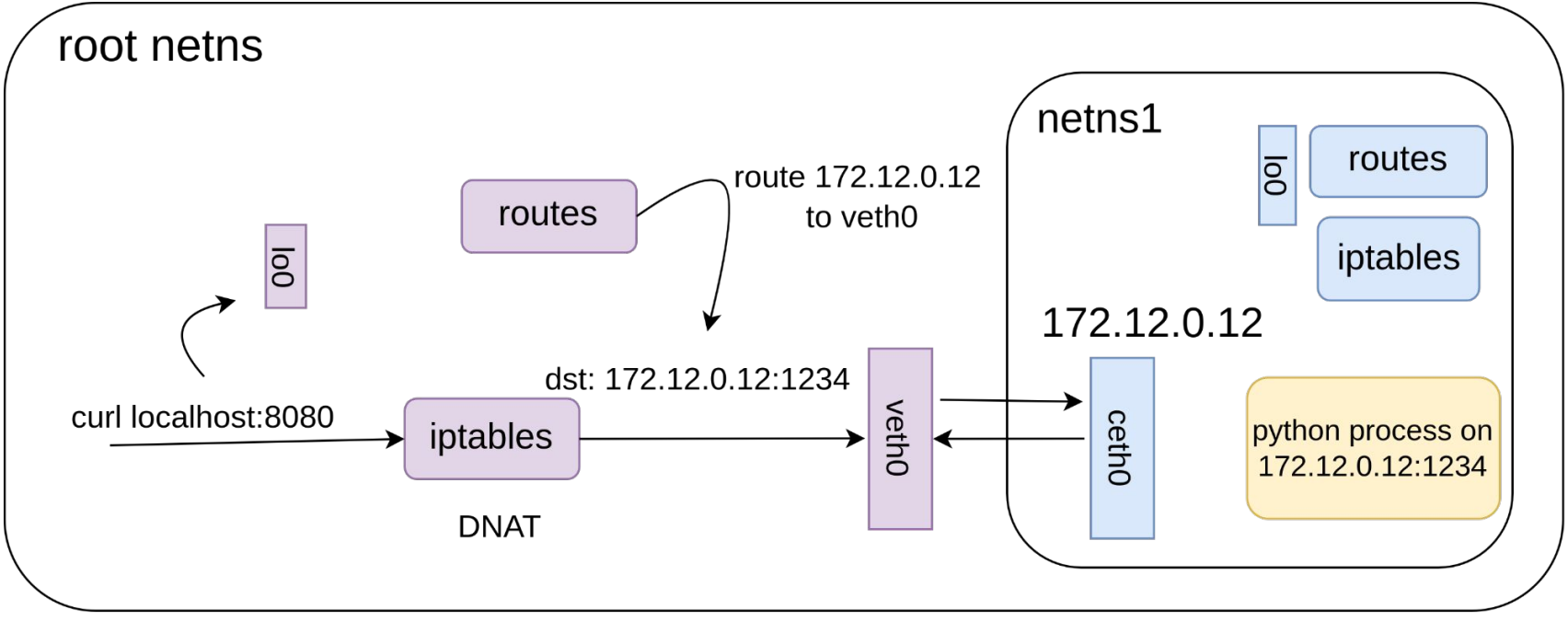
- Also called veth pairs
- Tunnel between namespaces
- Packets transmitted on one device are received in the other device



# Virtual ethernet devices

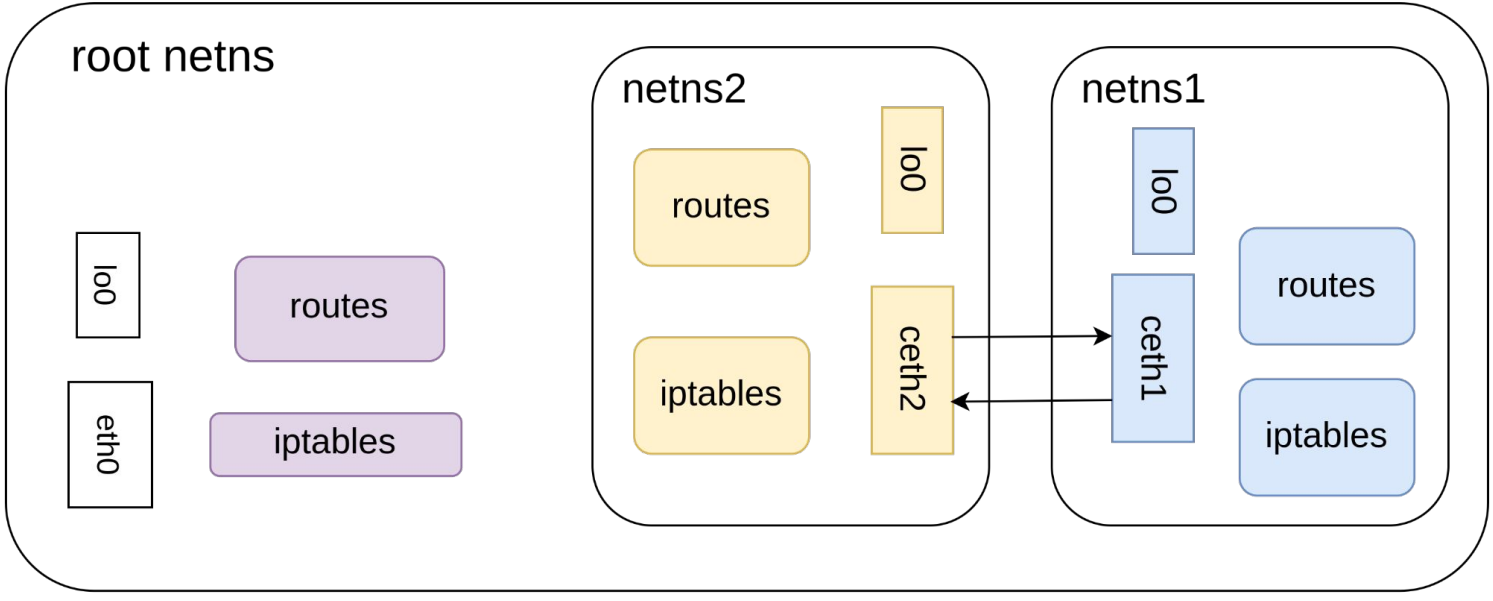


# Port mapping: veth + iptables



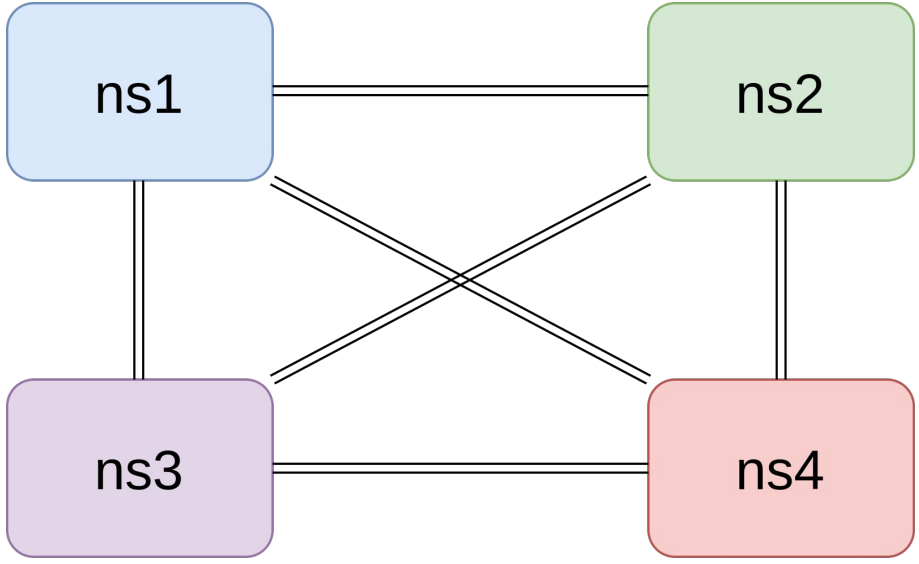
# Connect 2 namespaces

- One veth pair end in netns1
- Other end in netns2
- Classical way to connect namespaces
- Does this scale??



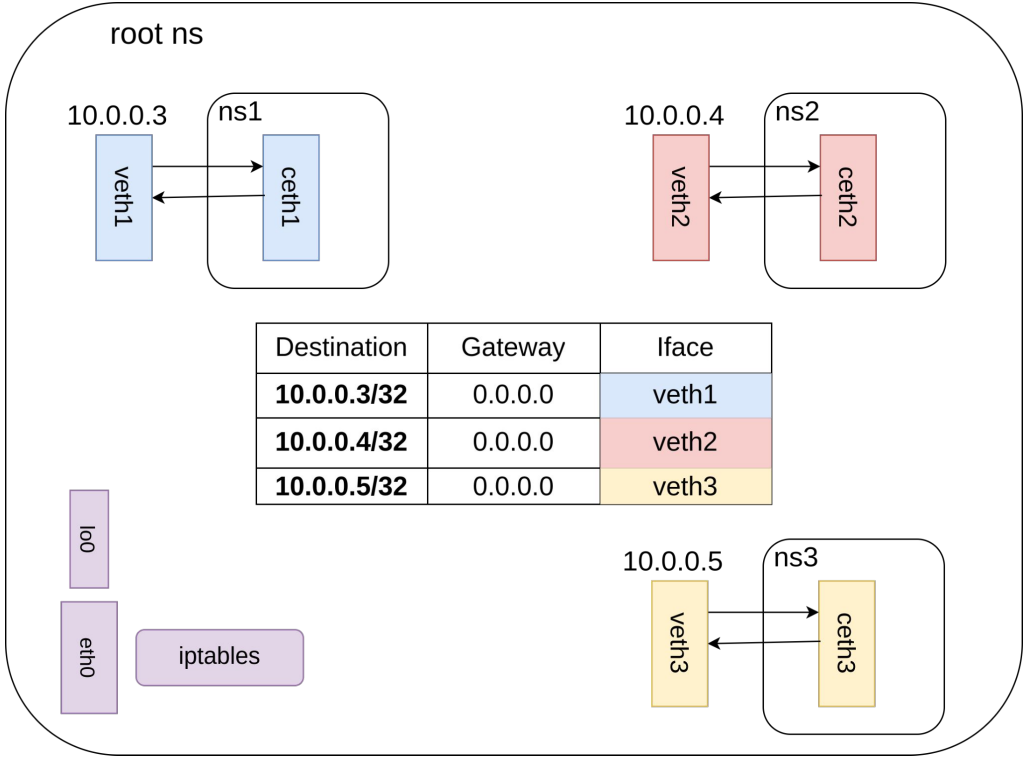
# Connect 4 namespaces

- For each new namespace, we need N-1 new veth pairs
- Need a better approach!!



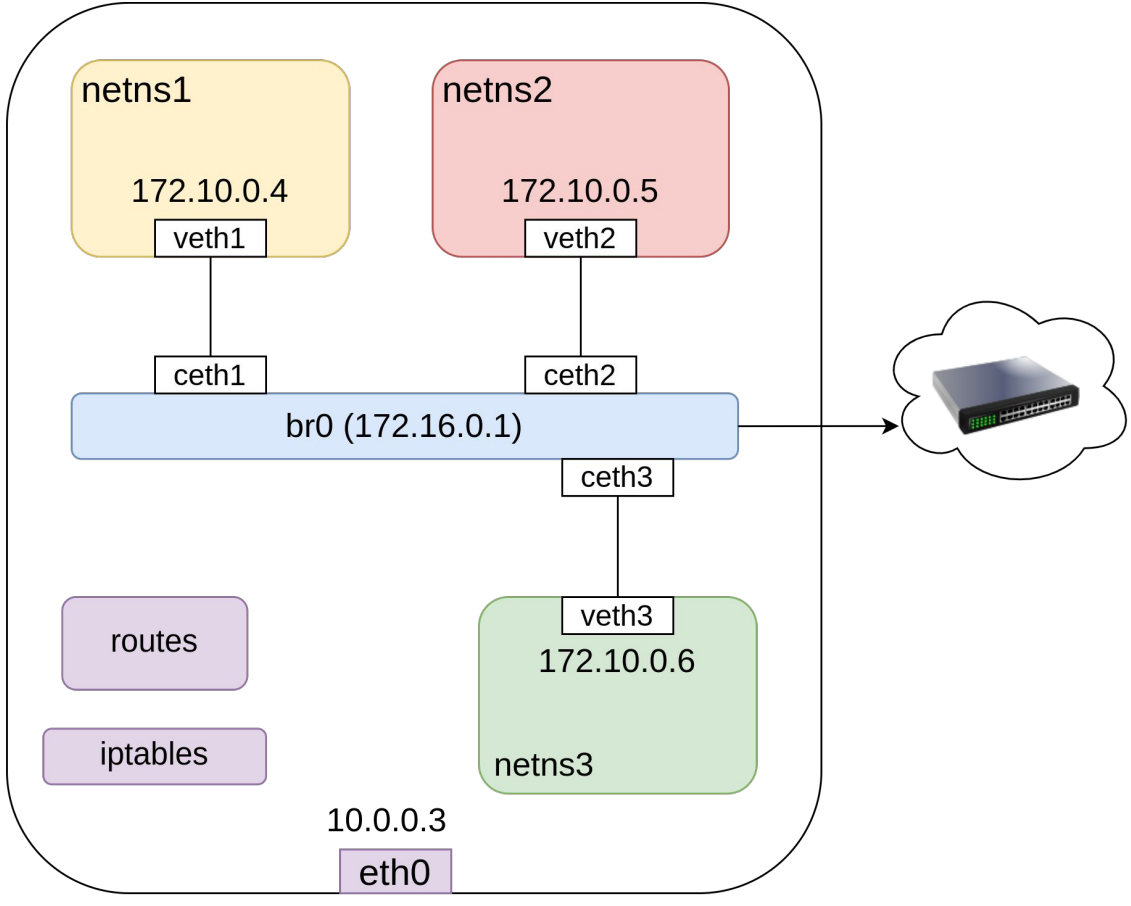
# Connect multiple namespaces

- Option 1: through routing table in the root namespace
- Calico in k8s does this
- Downsides??

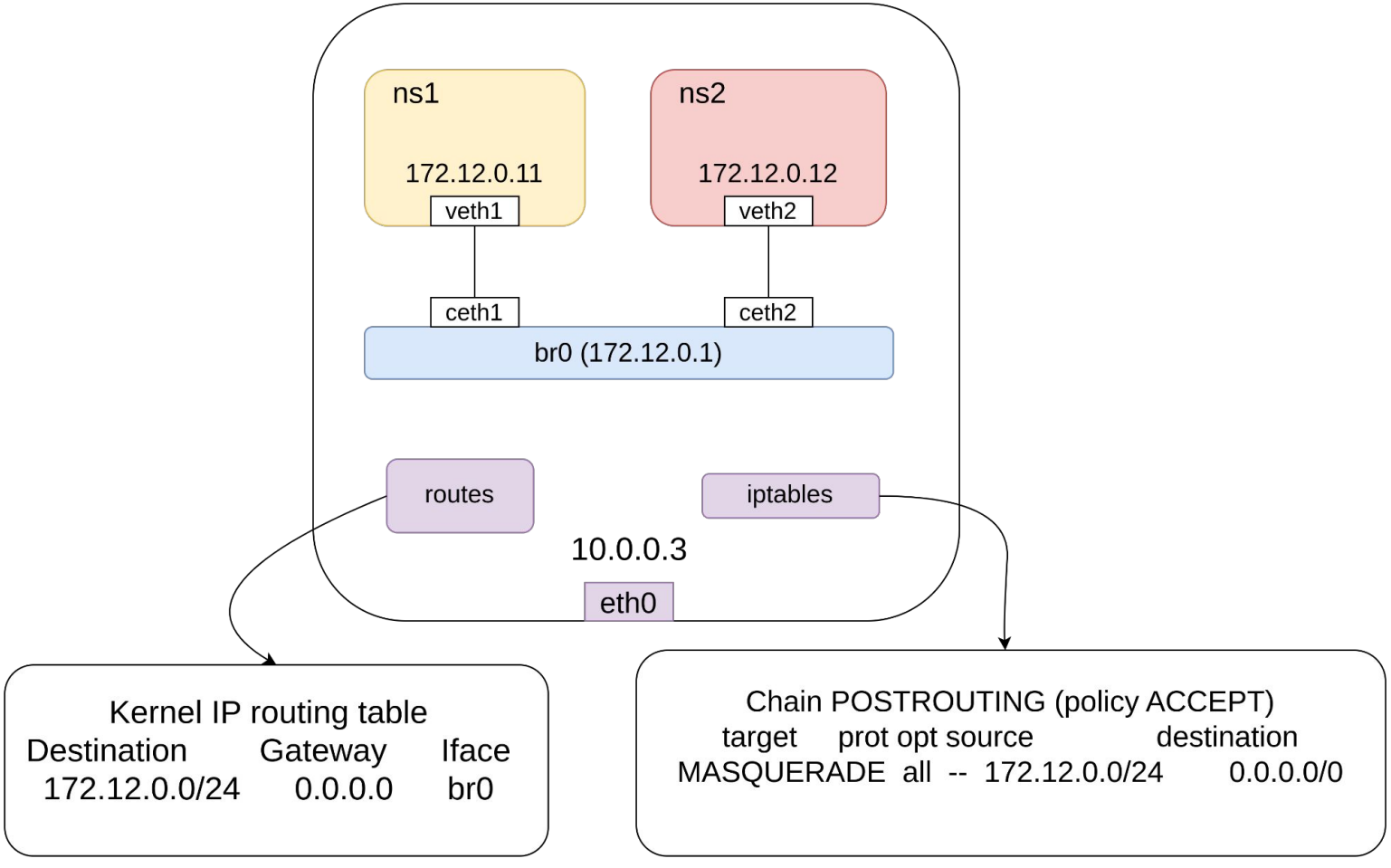


# Connect multiple namespaces

- Option 2: use a virtual bridge
- Acts as **L3 switch**
- Docker does this (default docker0) bridge



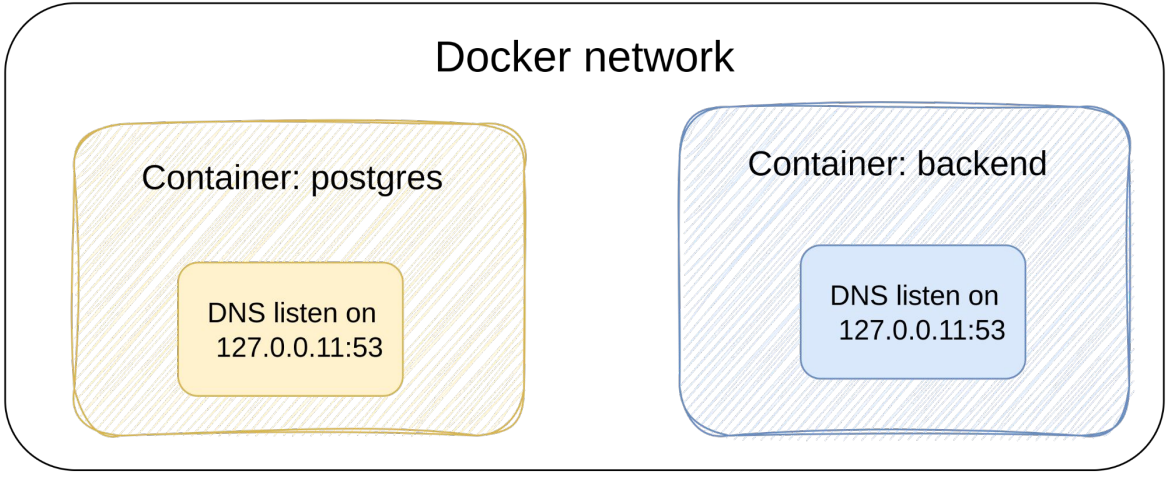
# Connect multiple namespaces



# Docker DNS

- DNS resolver inside each container

```
1  services:
2    backend:
3      build: .
4      ports:
5        - "8000:5000"
6    postgres:
7      image: "postgres:13.15"
8
```



# Agenda

- Some basics ✅
  - OSI ✅
  - NIC ✅
  - Routing ✅
  - NAT ✅
- IPTABLES ✅
- Network namespaces ✅
  - Veth pairs ✅
  - Virtual bridges ✅
- K8s
  - Architecture
  - Pod to Pod networking
  - Service networking
  - DNS

# K8s

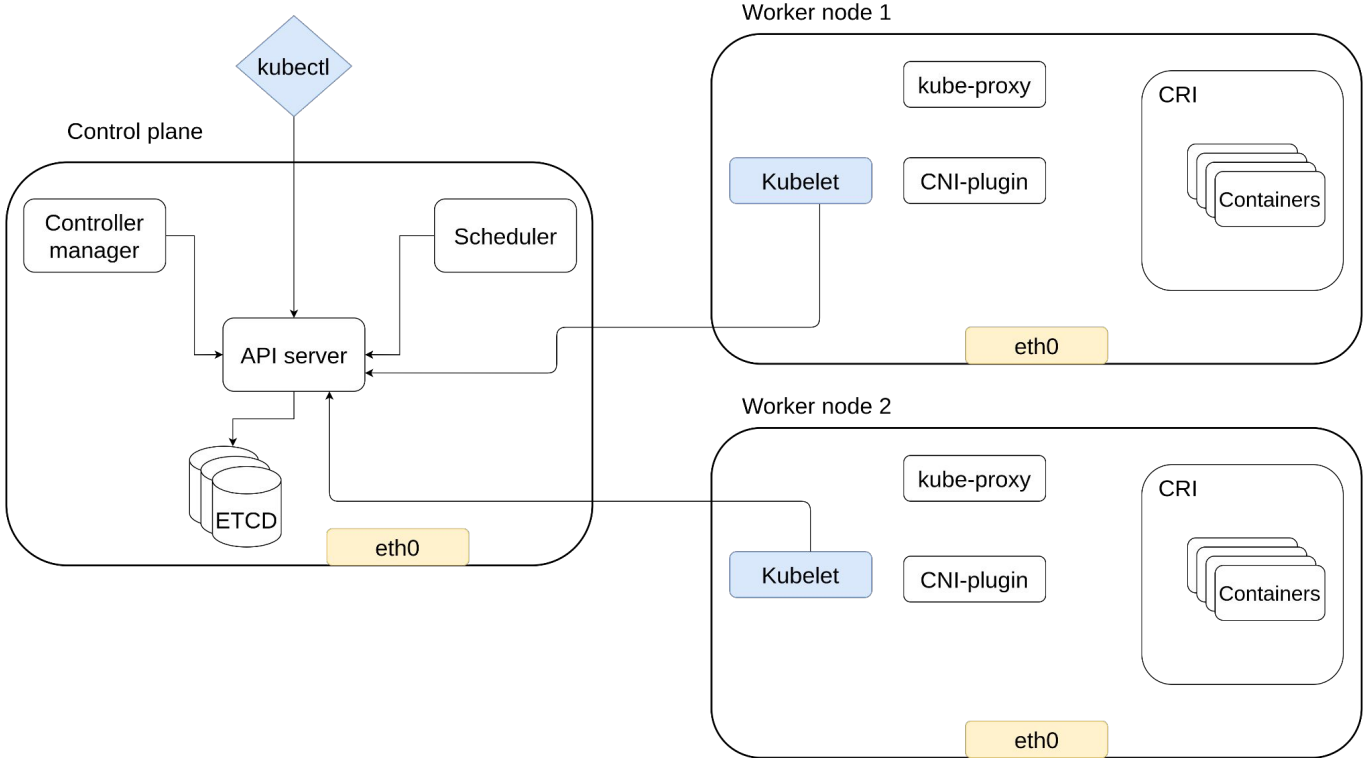
## K8s: networking model

- All pods have their own IPs
- All pods can talk to all other pods
- All nodes can talk to all pods

How to achieve these goals?

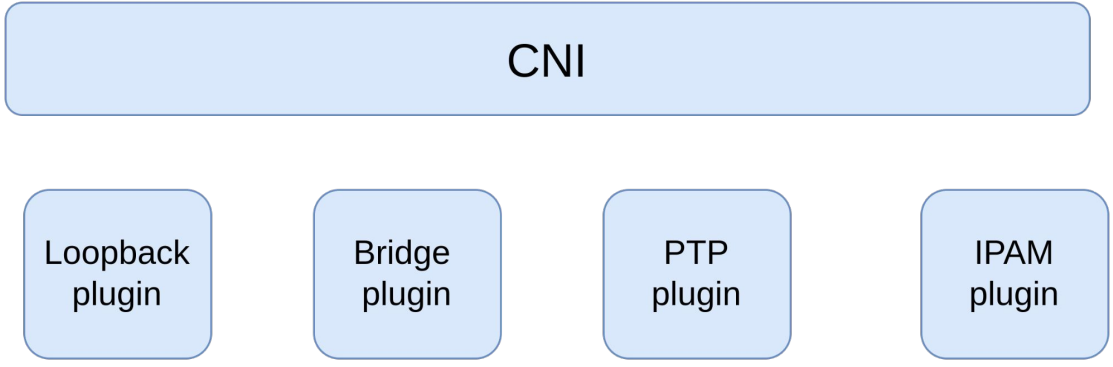
# K8s architecture

- kubelet: agent running on each node to create containers
- controller manager: monitor the state of the cluster (current state, desired state)
- kube-proxy: programs iptables on each node
- CRI: container runtime interface (e.g Podman)
- **CNI-plugin**: container network interface (e.g Calico)



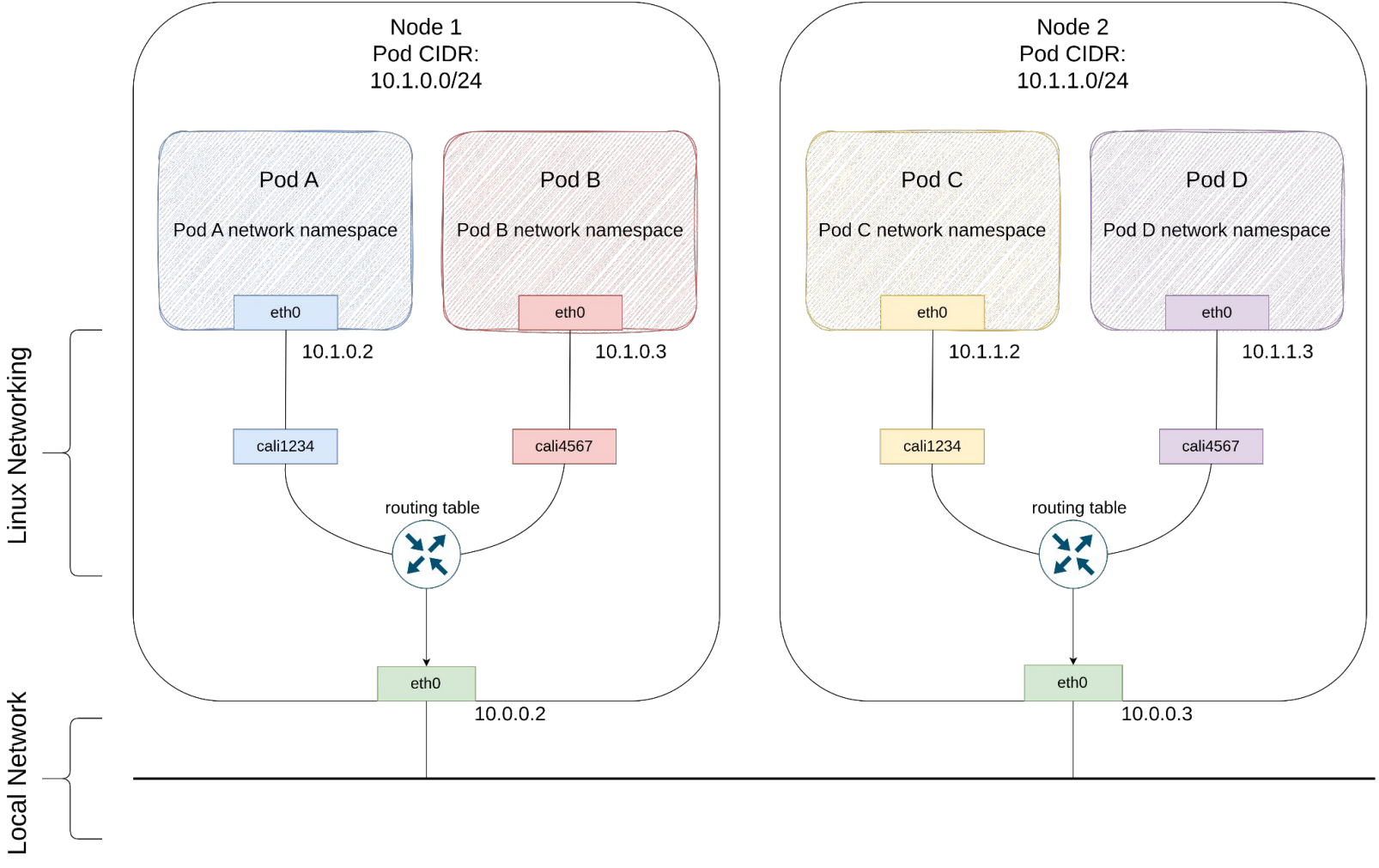
# K8s: Container network interface CNI

- Specification and libraries to write plugins
- Plugins configure network resources
  - Manages network connectivity of containers
  - Provide Pods with IP addresses
  - Delete network resources when pods are deleted
- E.g: Flannel and **Calico**

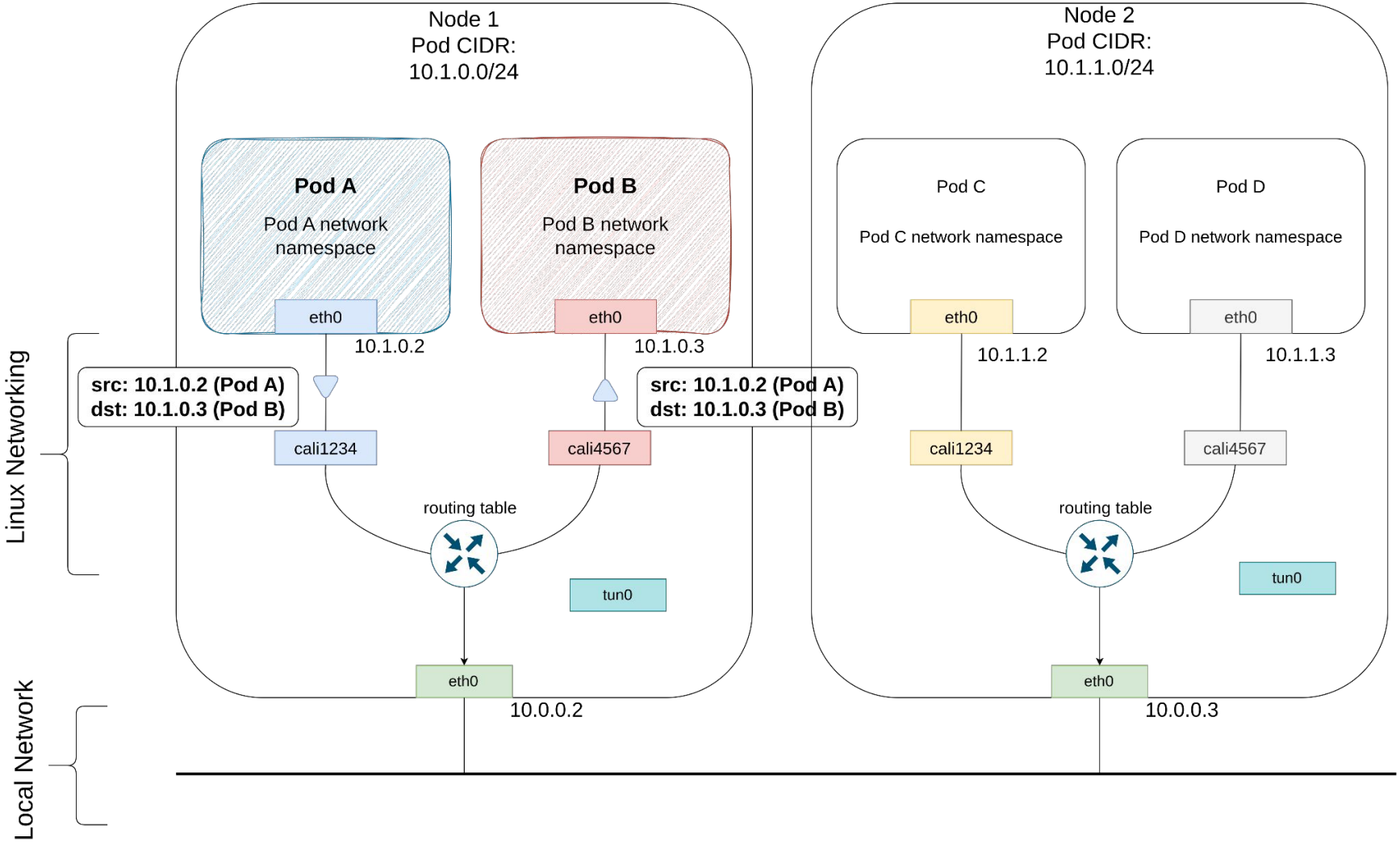


# K8s networking

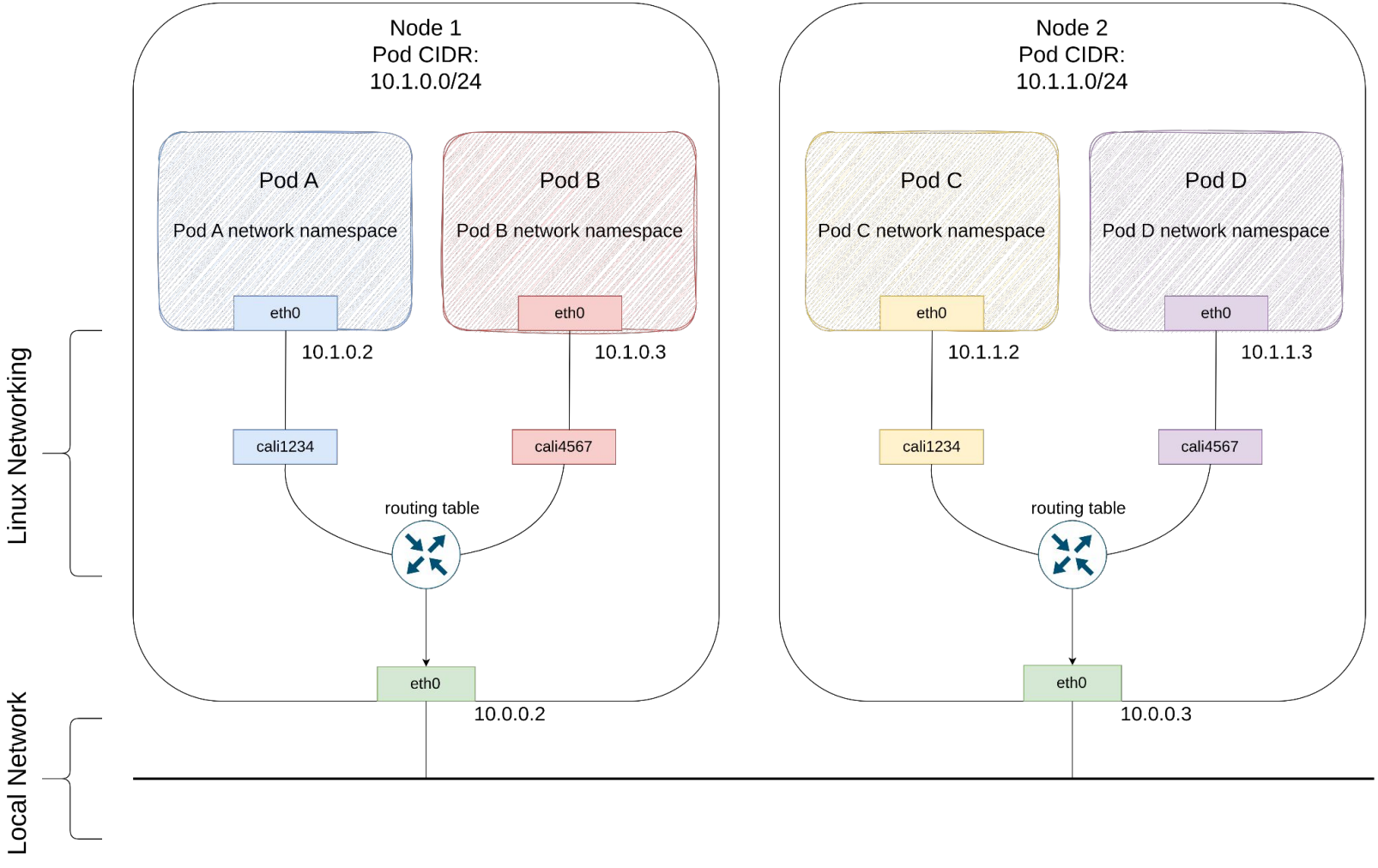
- Goal: pod-to-pod and pod-to-service connectivity



# Pod to Pod networking: Same Node

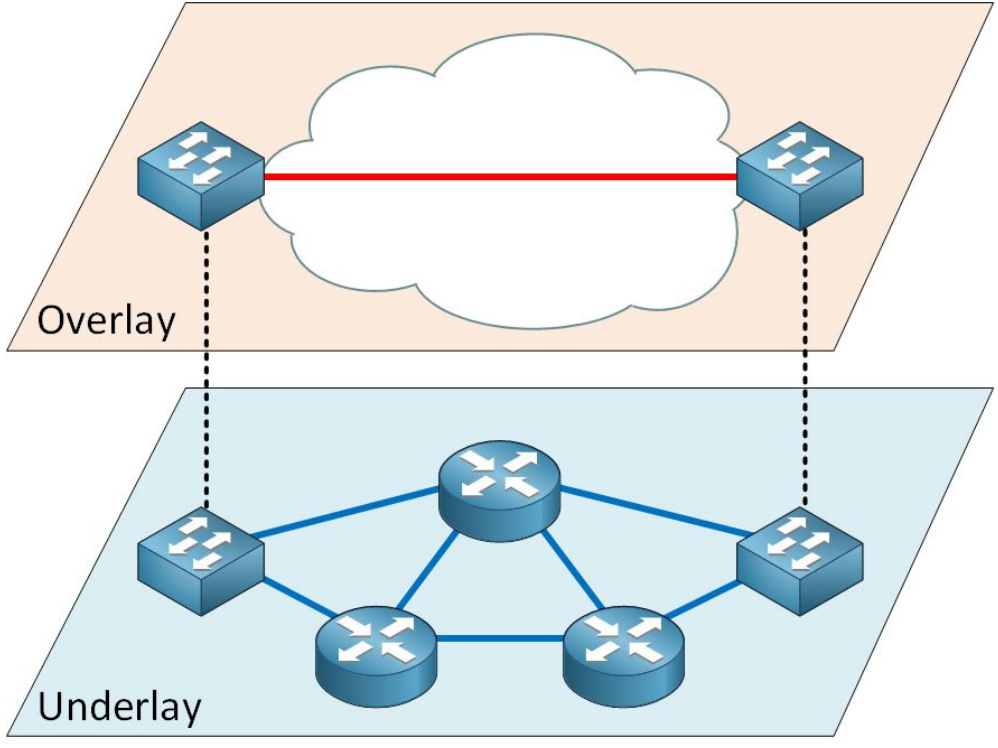


# Pod to Pod networking: Different nodes



# Overlay network

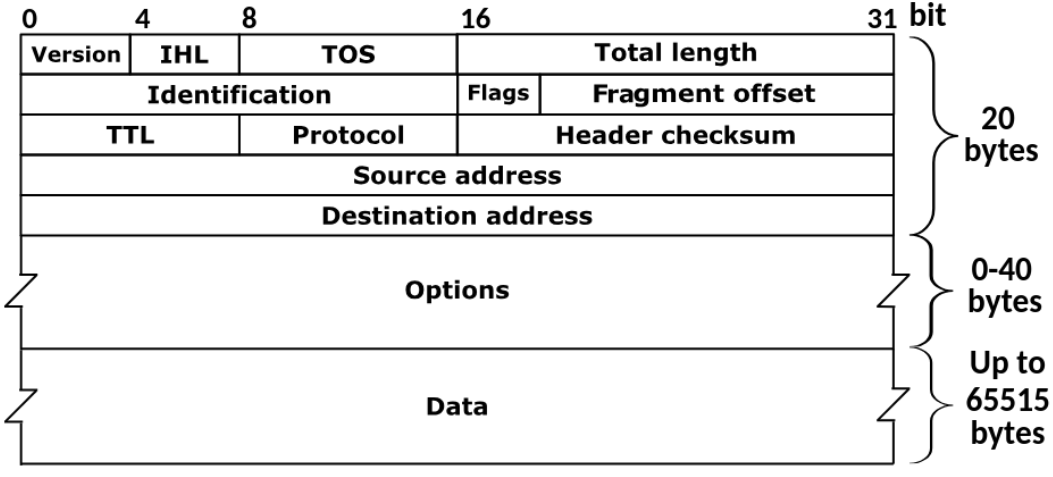
- **Virtual** network running on top of a **physical** network (called underlay network)
- Overlay for pods
- Underlay for nodes

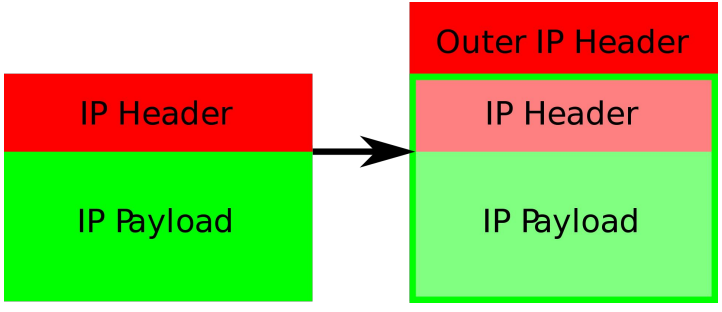


# Ip in Ip

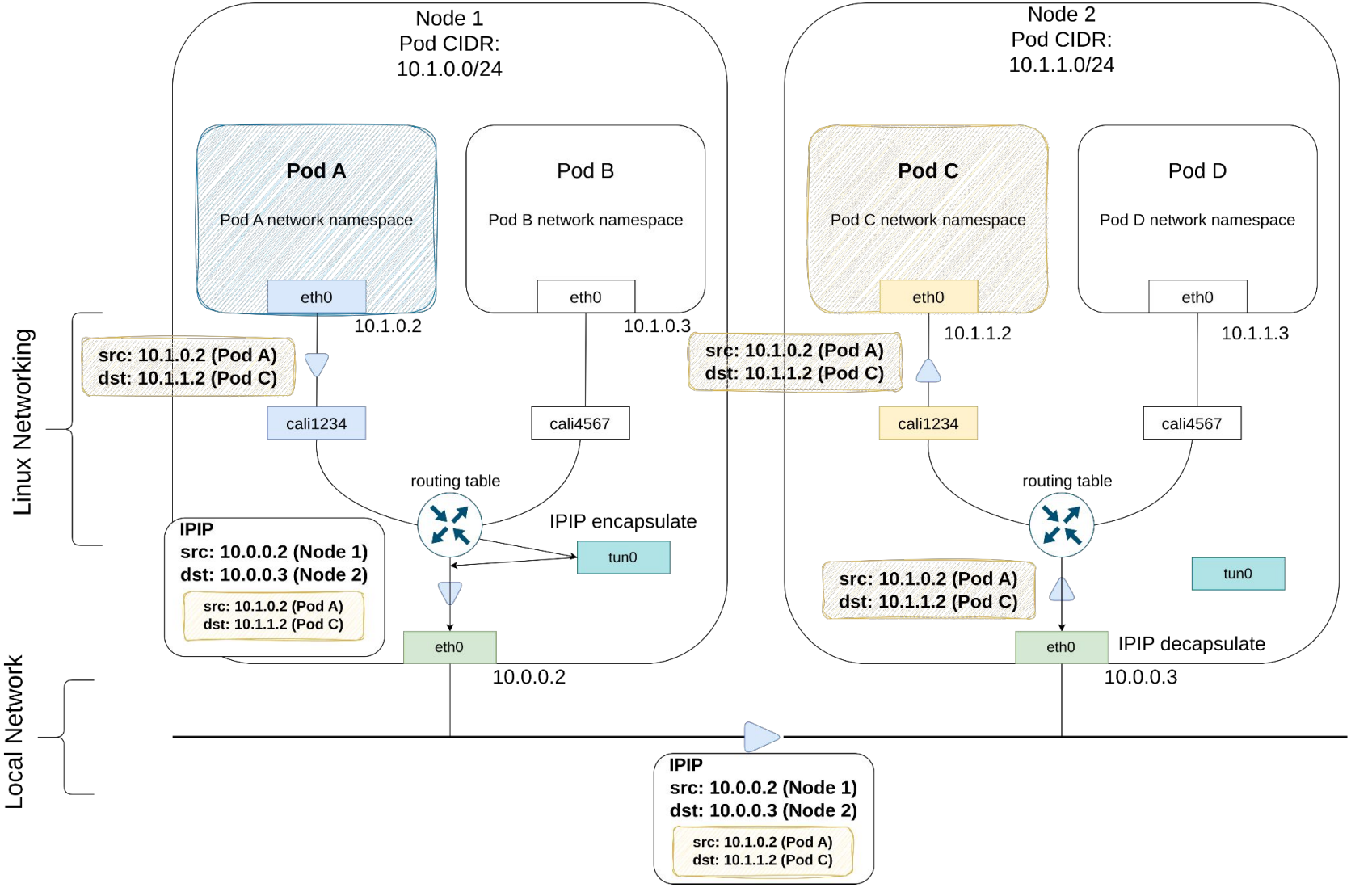
- IP tunneling protocol
- Encapsulate one IP packet in another IP packet
  - Tunnel **entry** does the **encapsulation**
  - Tunnel **end** does the **decapsulation**

IP packet structure



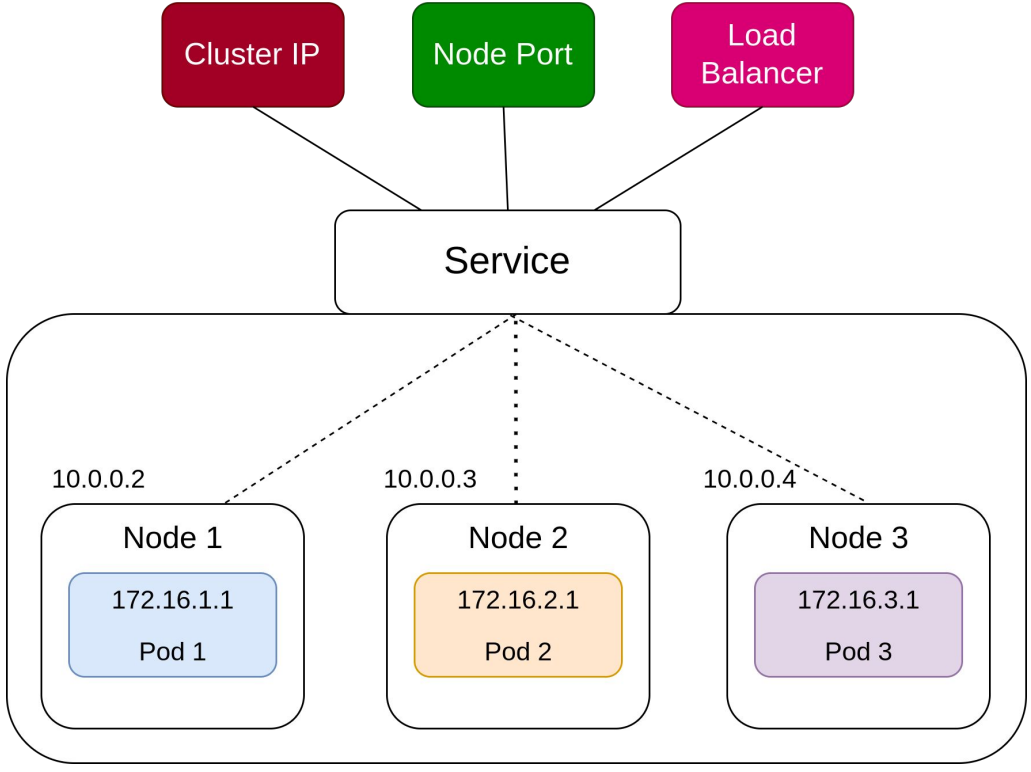



# Pod to Pod networking: Different Nodes

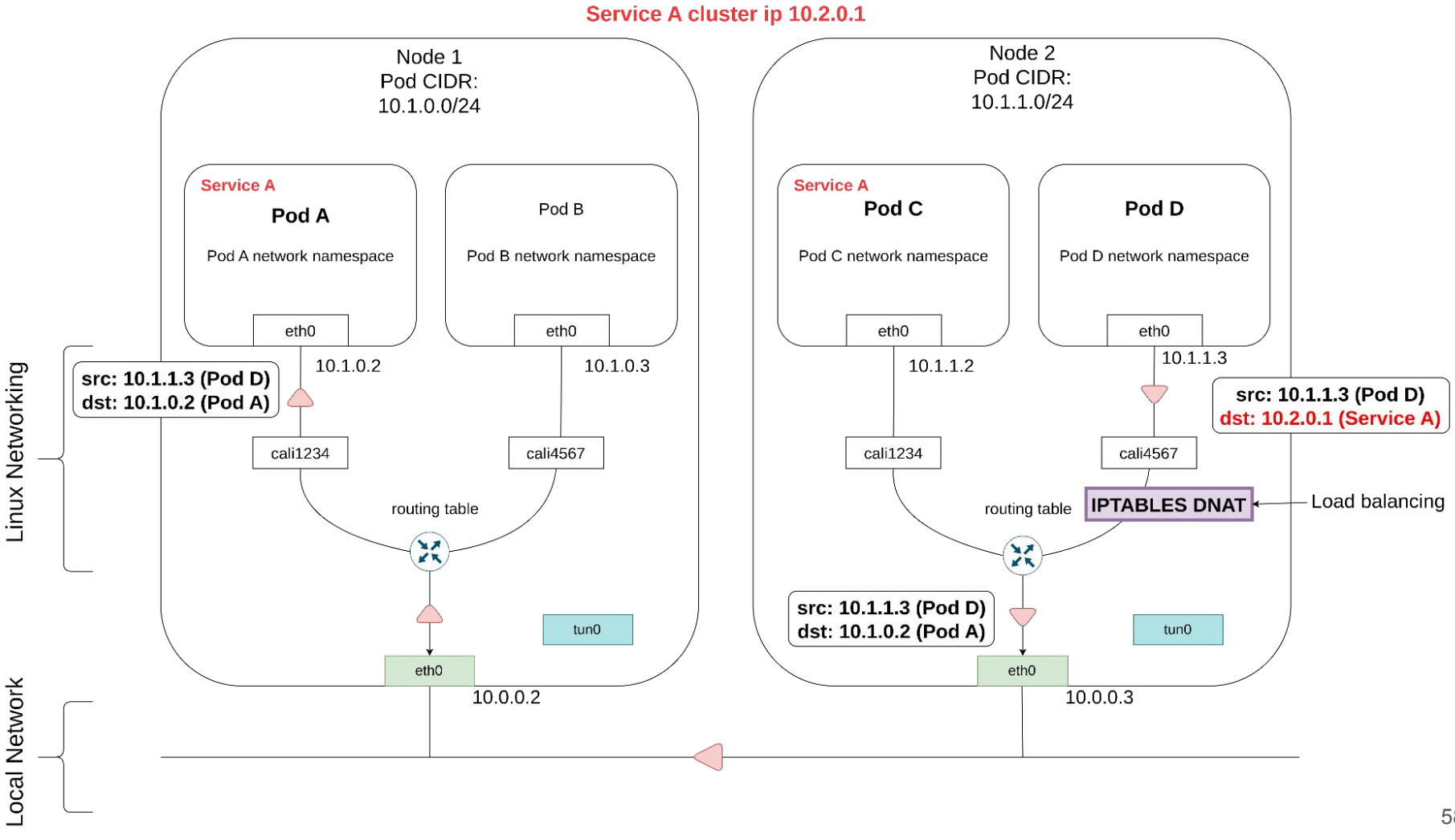


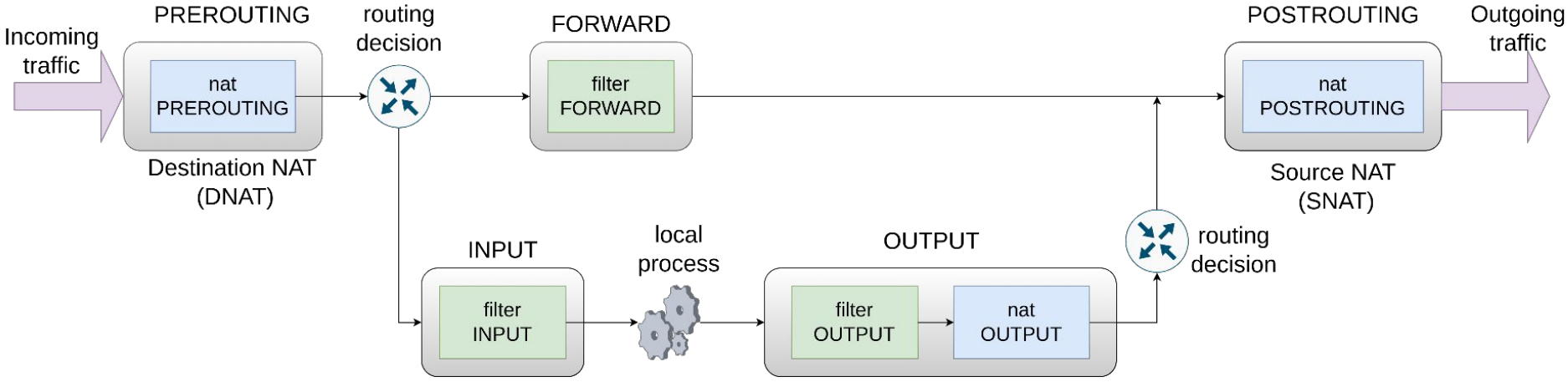
# K8s services overview

- A service balances load between pods
- Cluster IP
  - Service gets a virtual ip
- NodePort
  - Listen on a port on all nodes
  - Based on ClusterIP internally

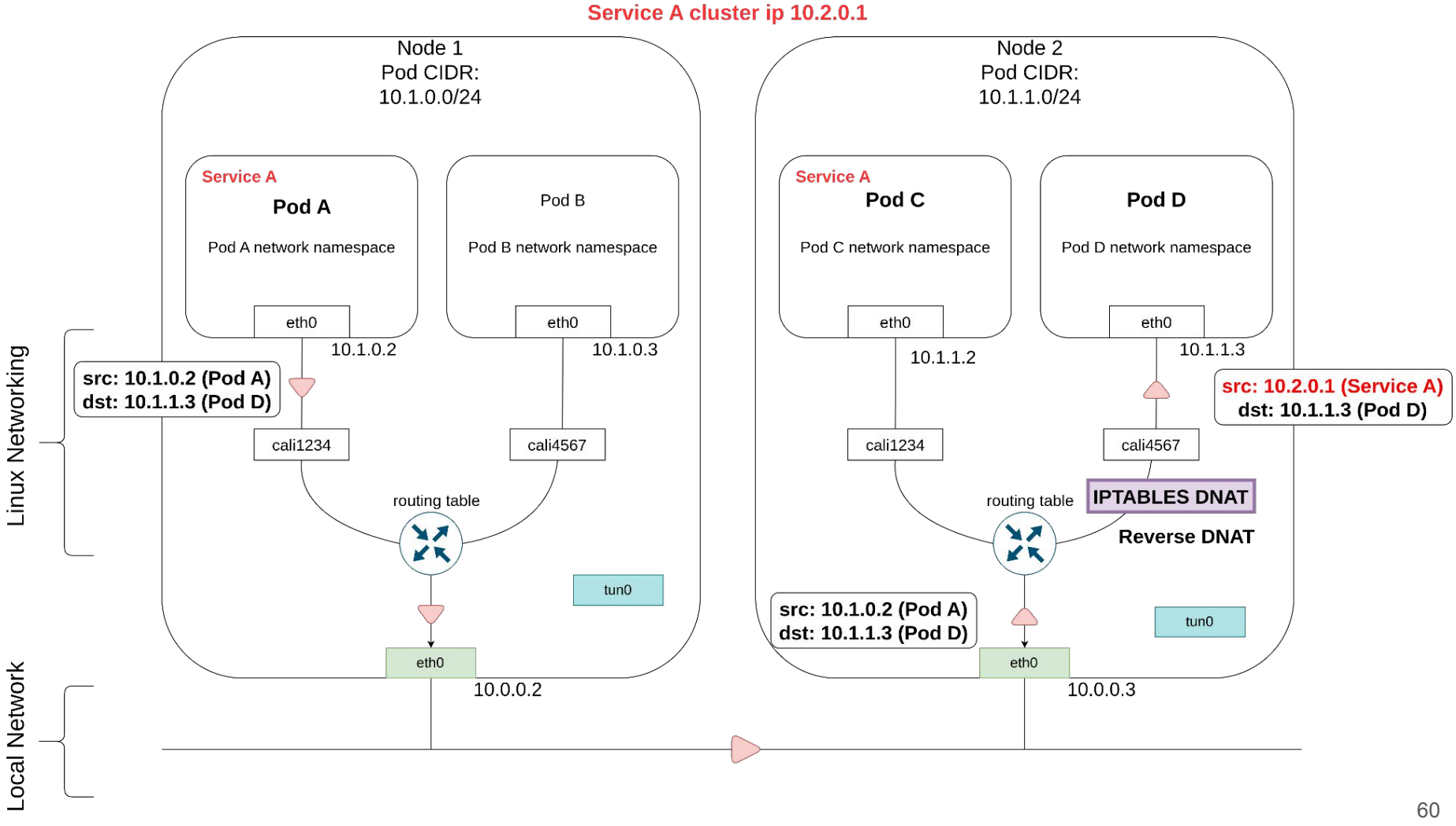


# ClusterIP request




Incoming traffic
PREROUTING
nat PREROUTING
Destination NAT (DNAT)
routing decision
FORWARD
filter FORWARD
POSTROUTING
nat POSTROUTING
Source NAT (SNAT)
Outgoing traffic
INPUT
filter INPUT
local process
OUTPUT
filter OUTPUT
nat OUTPUT
routing decision

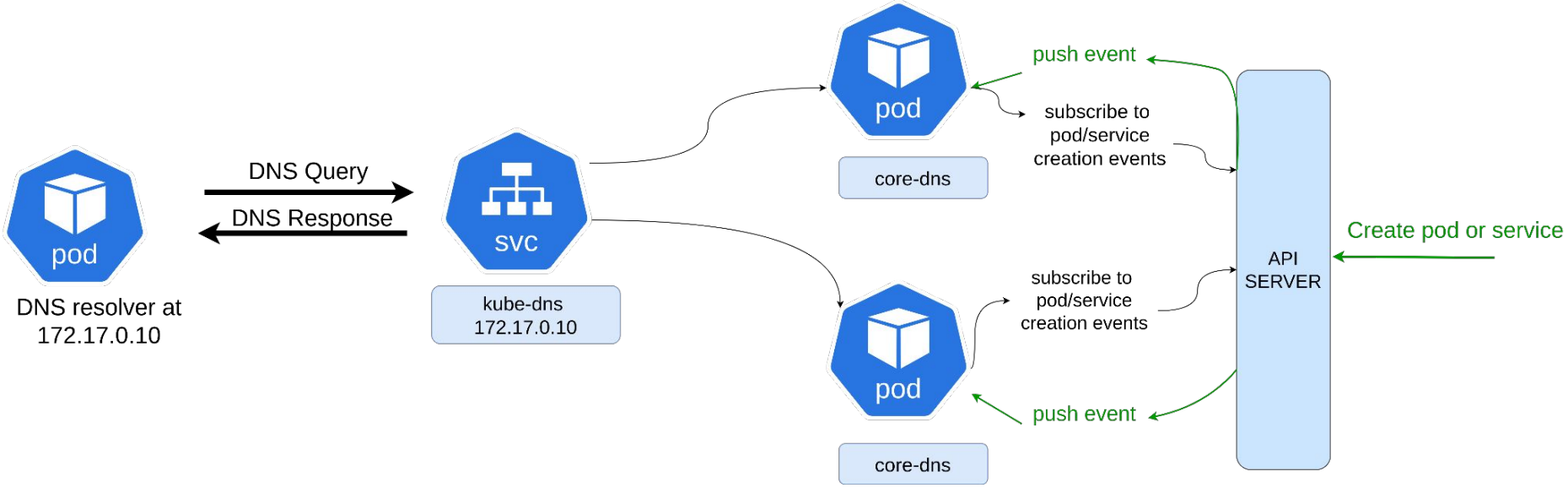
# ClusterIP response



# DNS

# K8s dns

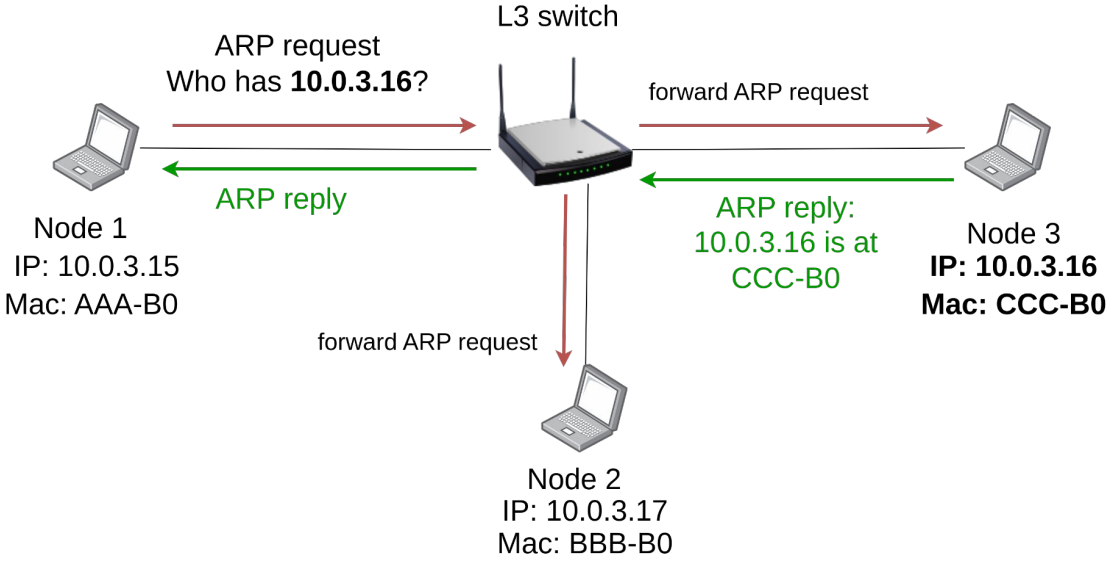
- Each pod has a dns resolver in `/etc/resolv.conf`
- The resolver points to ClusterIP of `kube-dns` service
- Core-dns pods consult the service registry
- Service registry contains mapping of service names to pod ips

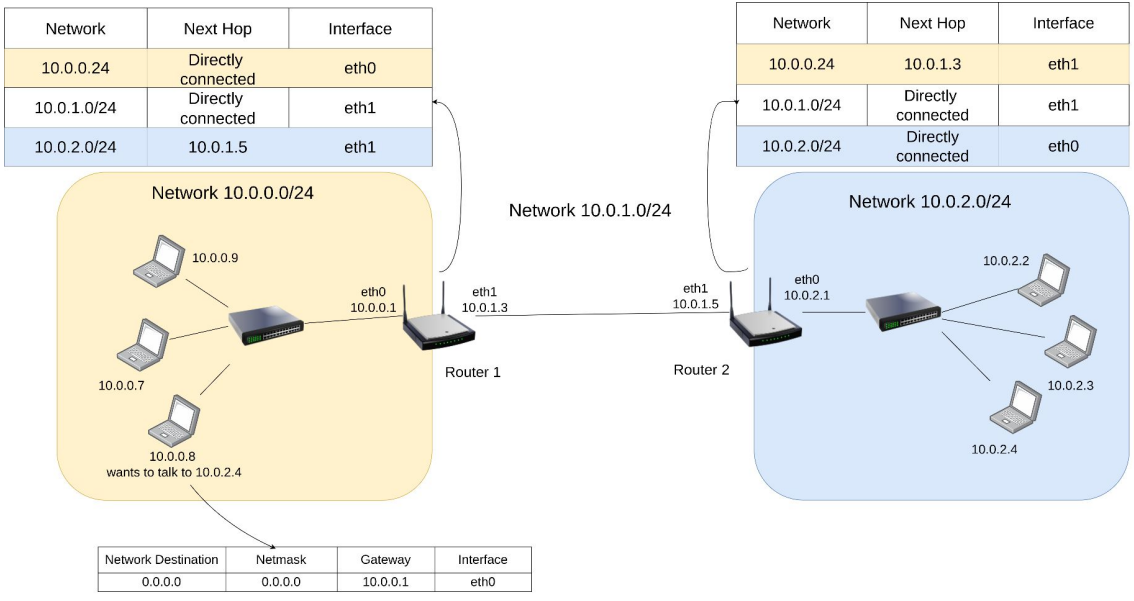


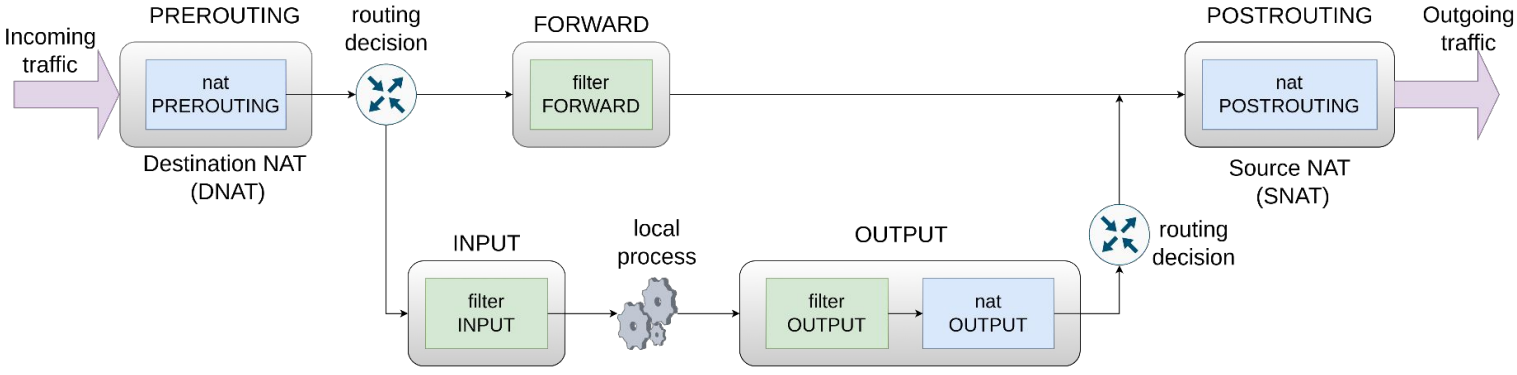
# Agenda

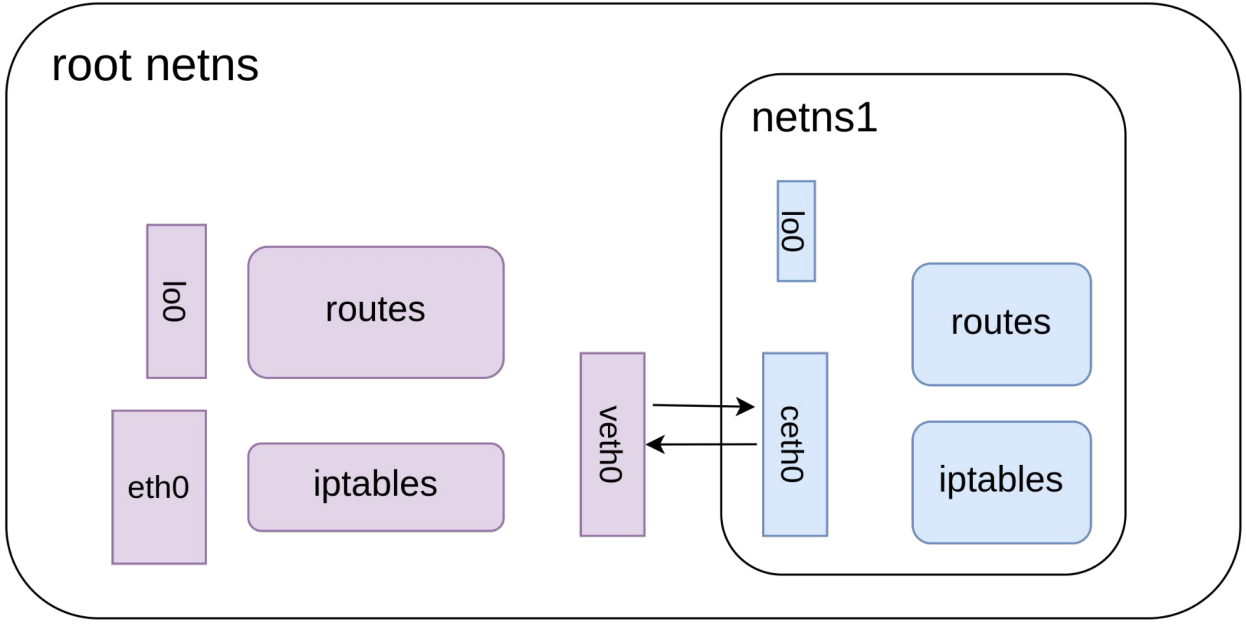
- Some basics ✅
  - OSI ✅
  - NIC ✅
  - Routing ✅
  - NAT ✅
- Network namespaces ✅
  - Veth pairs ✅
  - Virtual bridges ✅
- IPTABLES ✅
- K8s ✅
  - Architecture ✅
  - Pod to Pod networking ✅
  - Service networking ✅
  - DNS ✅

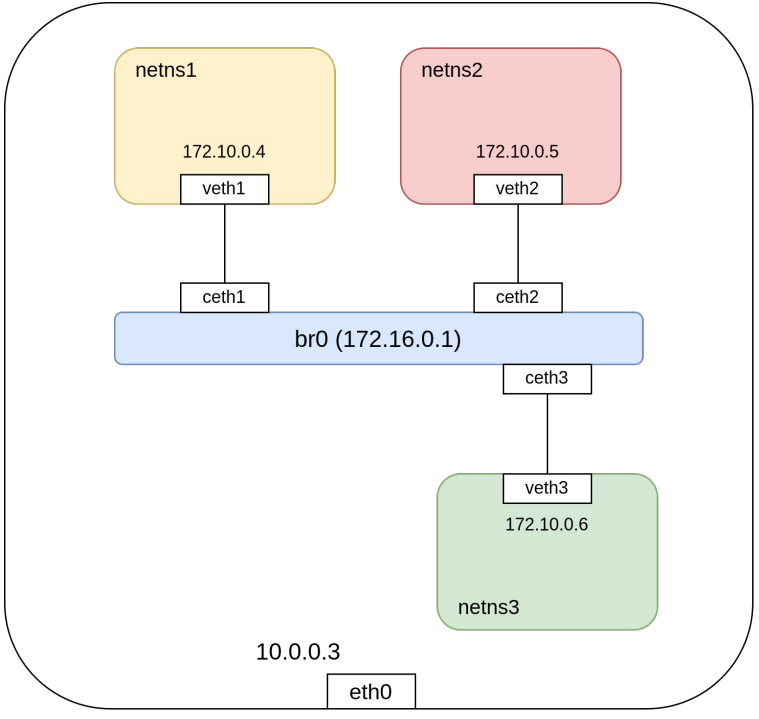
# Recap

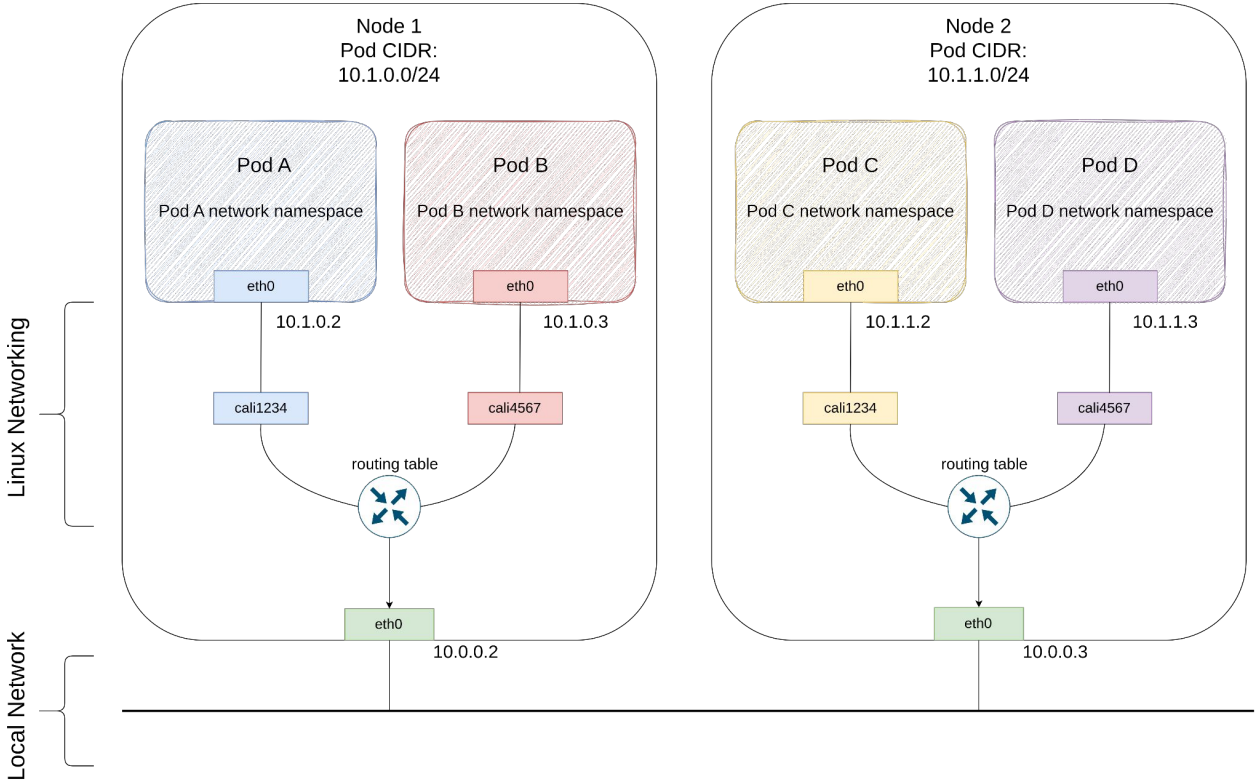

L3 switch
ARP request
Who has **10.0.3.16**?
forward ARP request
ARP reply
ARP reply:
10.0.3.16 is at
CCC-B0
Node 1
IP: 10.0.3.15
Mac: AAA-B0
Node 3
**IP: 10.0.3.16**
**Mac: CCC-B0**
forward ARP request
Node 2
IP: 10.0.3.17
Mac: BBB-B0

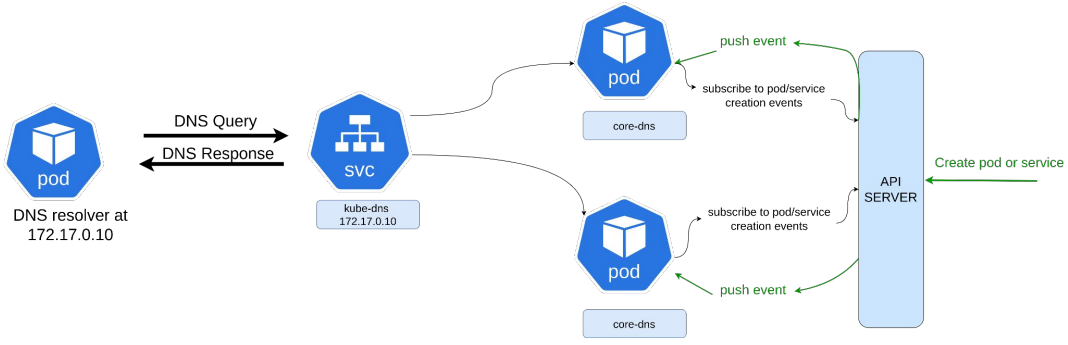

Network
Next Hop
Interface
10.0.0.24
Directly connected
eth0
10.0.1.0/24
Directly connected
eth1
10.0.2.0/24
10.0.1.5
eth1
Network
Next Hop
Interface
10.0.0.24
10.0.1.3
eth1
10.0.1.0/24
Directly connected
eth1
10.0.2.0/24
Directly connected
eth0
Network 10.0.0.0/24
Network 10.0.1.0/24
Network 10.0.2.0/24
10.0.0.9
10.0.0.7
10.0.0.8
wants to talk to 10.0.2.4
eth0
10.0.0.1
eth1
10.0.1.3
Router 1
eth1
10.0.1.5
Router 2
eth0
10.0.2.1
10.0.2.2
10.0.2.3
10.0.2.4
Network Destination
Netmask
Gateway
Interface
0.0.0.0
0.0.0.0
10.0.0.1
eth0


Incoming traffic
PREROUTING
nat PREROUTING
Destination NAT (DNAT)
routing decision
FORWARD
filter FORWARD
POSTROUTING
nat POSTROUTING
Source NAT (SNAT)
Outgoing traffic
INPUT
filter INPUT
local process
OUTPUT
filter OUTPUT
nat OUTPUT
routing decision


root netns
lo0
routes
eth0
iptables
veth0
netns1
lo0
routes
ceth0
iptables


netns1
172.10.0.4
veth1
netns2
172.10.0.5
veth2
ceth1
ceth2
br0 (172.16.0.1)
ceth3
veth3
172.10.0.6
netns3
10.0.0.3
eth0


Node 1
Pod CIDR:
10.1.0.0/24
Pod A
Pod A network namespace
eth0
10.1.0.2
Pod B
Pod B network namespace
eth0
10.1.0.3
cali1234
cali4567
routing table
eth0
10.0.0.2
Node 2
Pod CIDR:
10.1.1.0/24
Pod C
Pod C network namespace
eth0
10.1.1.2
Pod D
Pod D network namespace
eth0
10.1.1.3
cali1234
cali4567
routing table
eth0
10.0.0.3
Linux Networking
Local Network


pod
DNS resolver at
172.17.0.10
DNS Query
DNS Response
svc
kube-dns
172.17.0.10
pod
core-dns
pod
core-dns
push event
subscribe to pod/service
creation events
subscribe to pod/service
creation events
push event
API
SERVER
Create pod or service

# Thank you