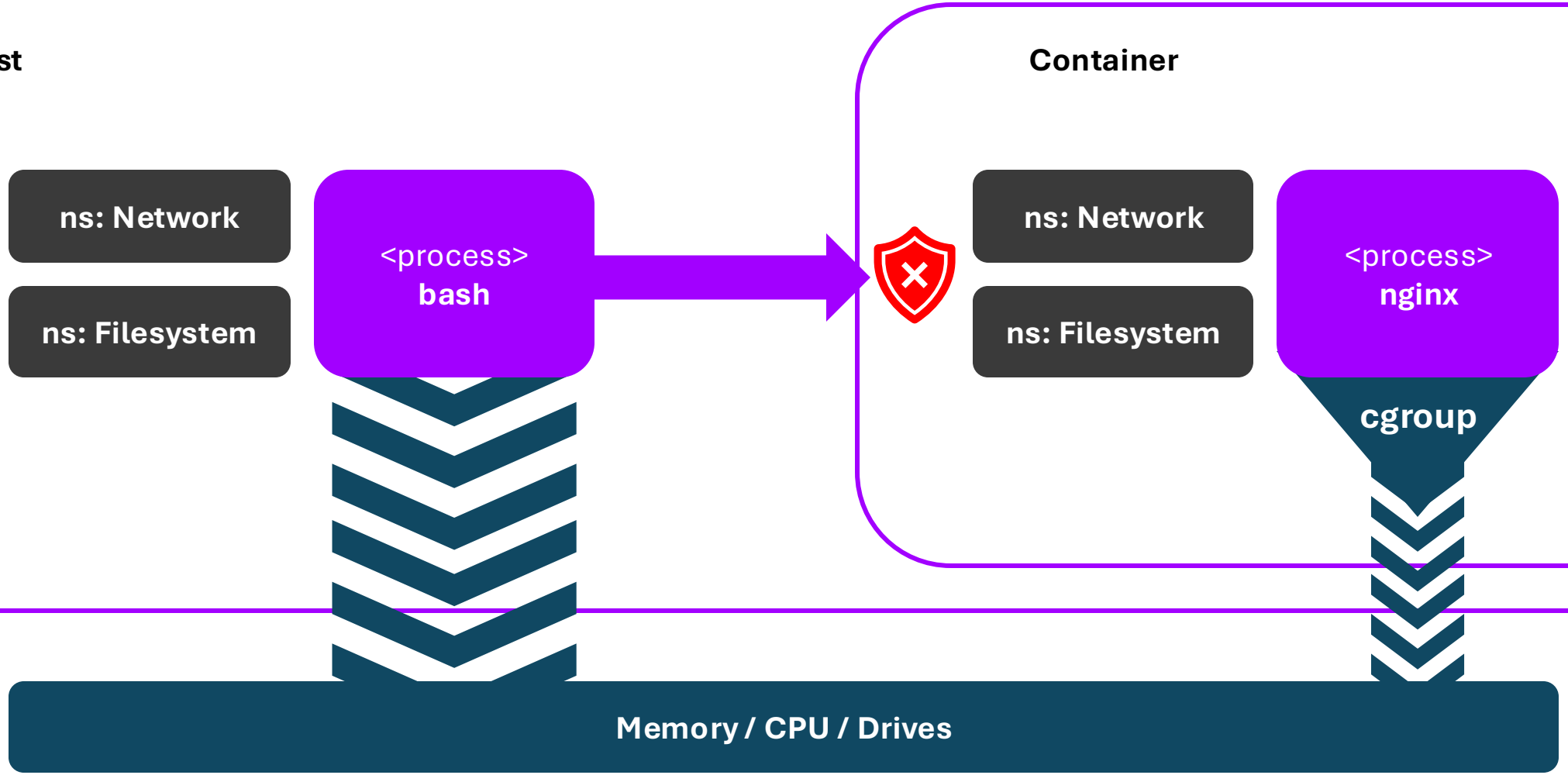
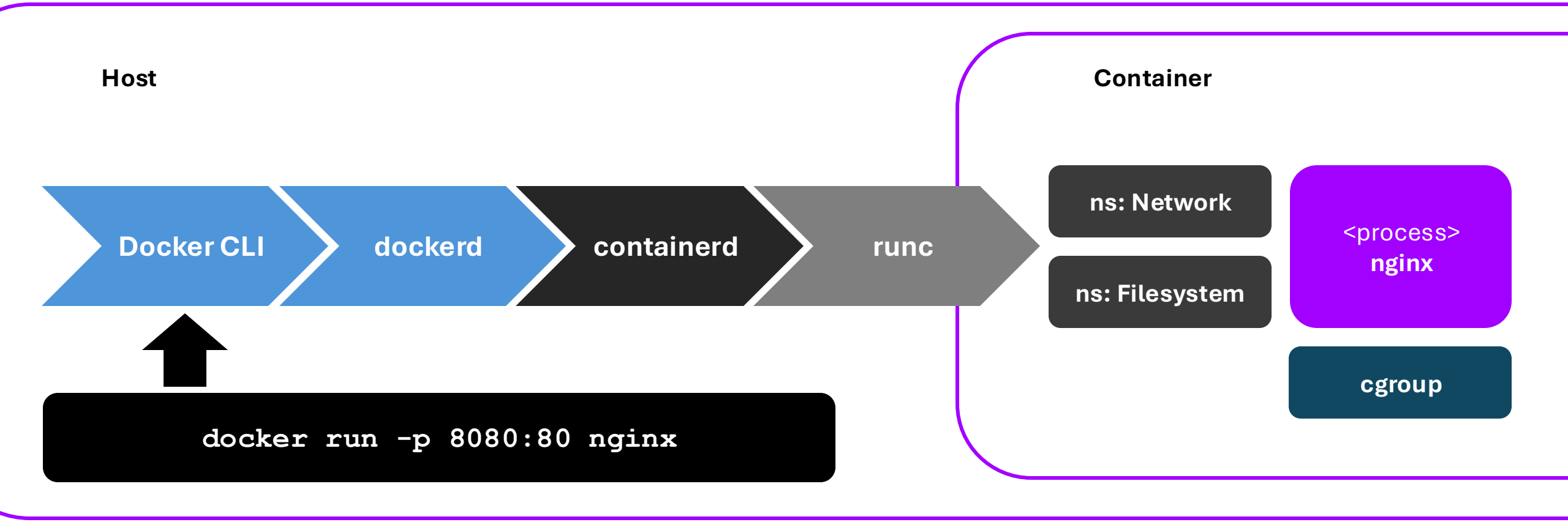
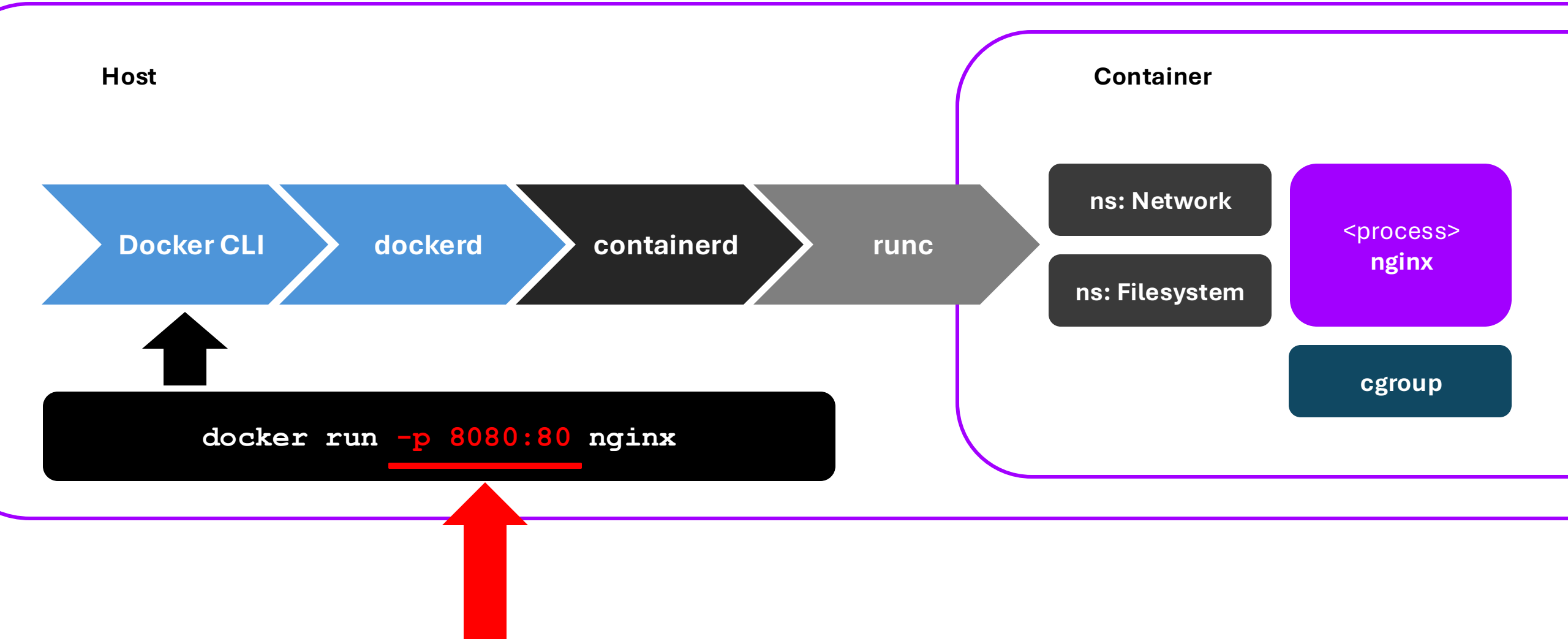
Docker network without Docker

Containers isolate processes with namespaces and limit them with cgroups.

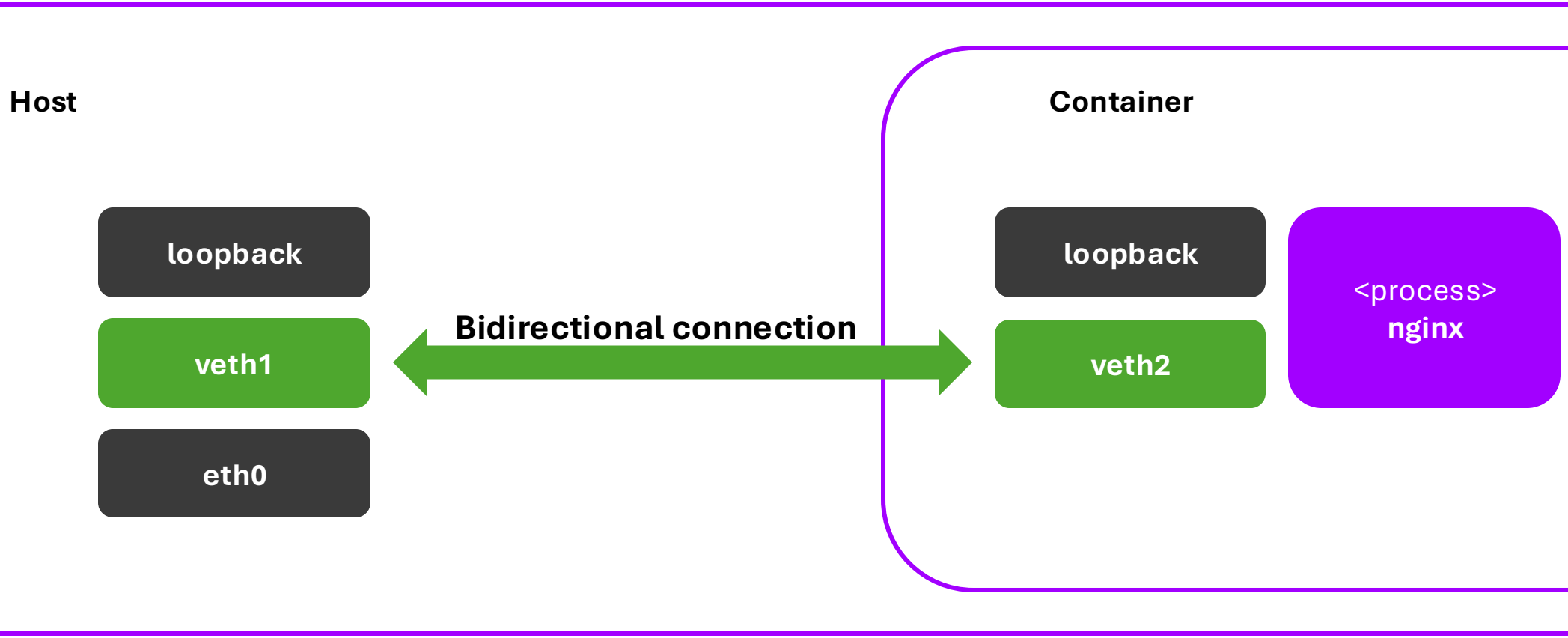

Host
Container
ns: Network
ns: Filesystem
<process>
bash
ns: Network
ns: Filesystem
<process>
nginx
cgroup
Memory / CPU / Drives

Docker is not magic; it's a configuration tool for namespaces and cgroups & ...


Host
Container
Docker CLI
dockerd
containerd
runc
ns: Network
ns: Filesystem
<process>
nginx
cgroup
docker run -p 8080:80 nginx


Host
Container
Docker CLI
dockerd
containerd
runc
ns: Network
ns: Filesystem
<process>
nginx
cgroup
docker run -p 8080:80 nginx

veth pairs link isolated network namespaces and enable fine grained access control.

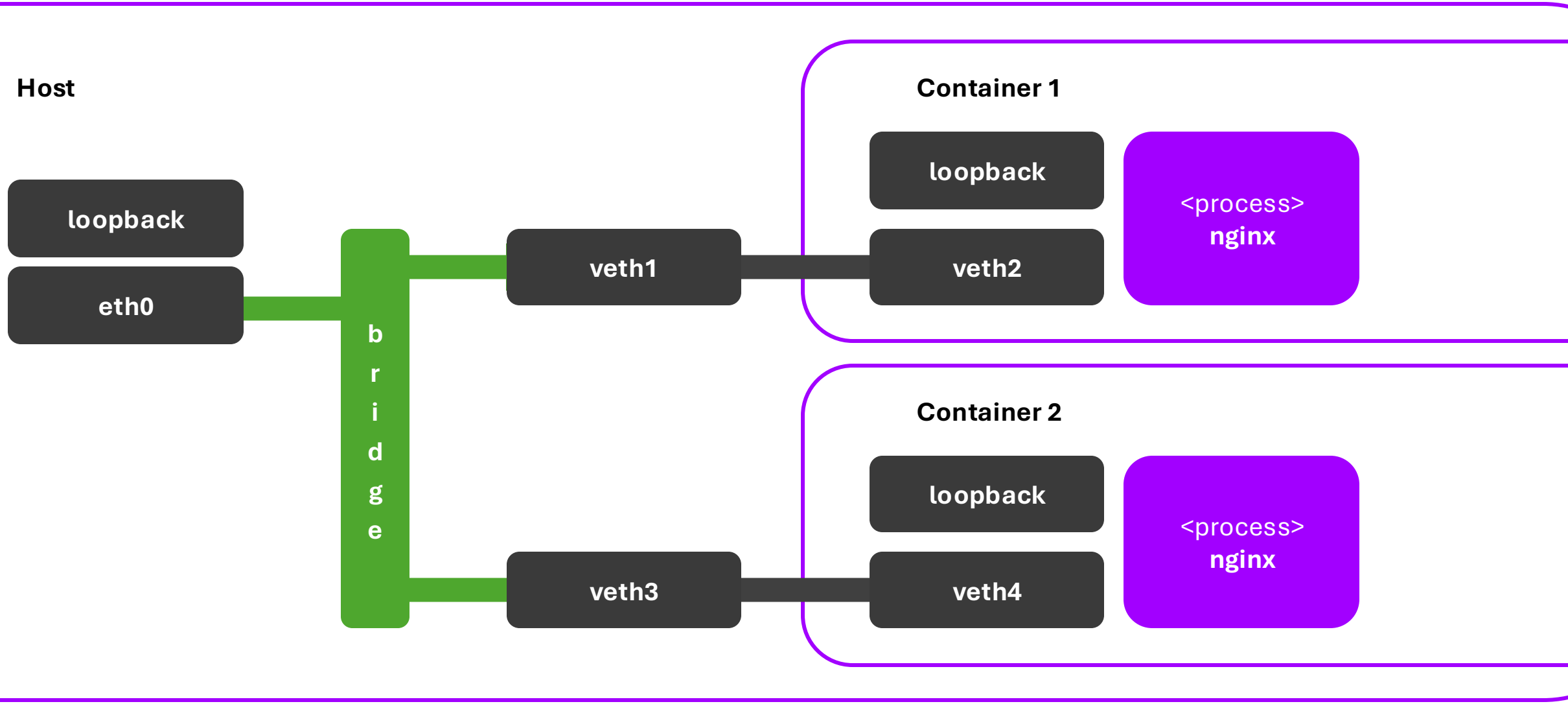

Host
Container
loopback
veth1
eth0
Bidirectional connection
loopback
veth2
<process>
nginx

Demo 1

Let’s connect custom namespaces to the host – no Docker required.

Just like handshakes, the number of veth pairs grows with the number of connections.

Bridges are like switches, **connecting a number of network links.**


Host
loopback
eth0
bridge
veth1
veth3
Container 1
loopback
veth2
<process>
nginx
Container 2
loopback
veth4
<process>
nginx

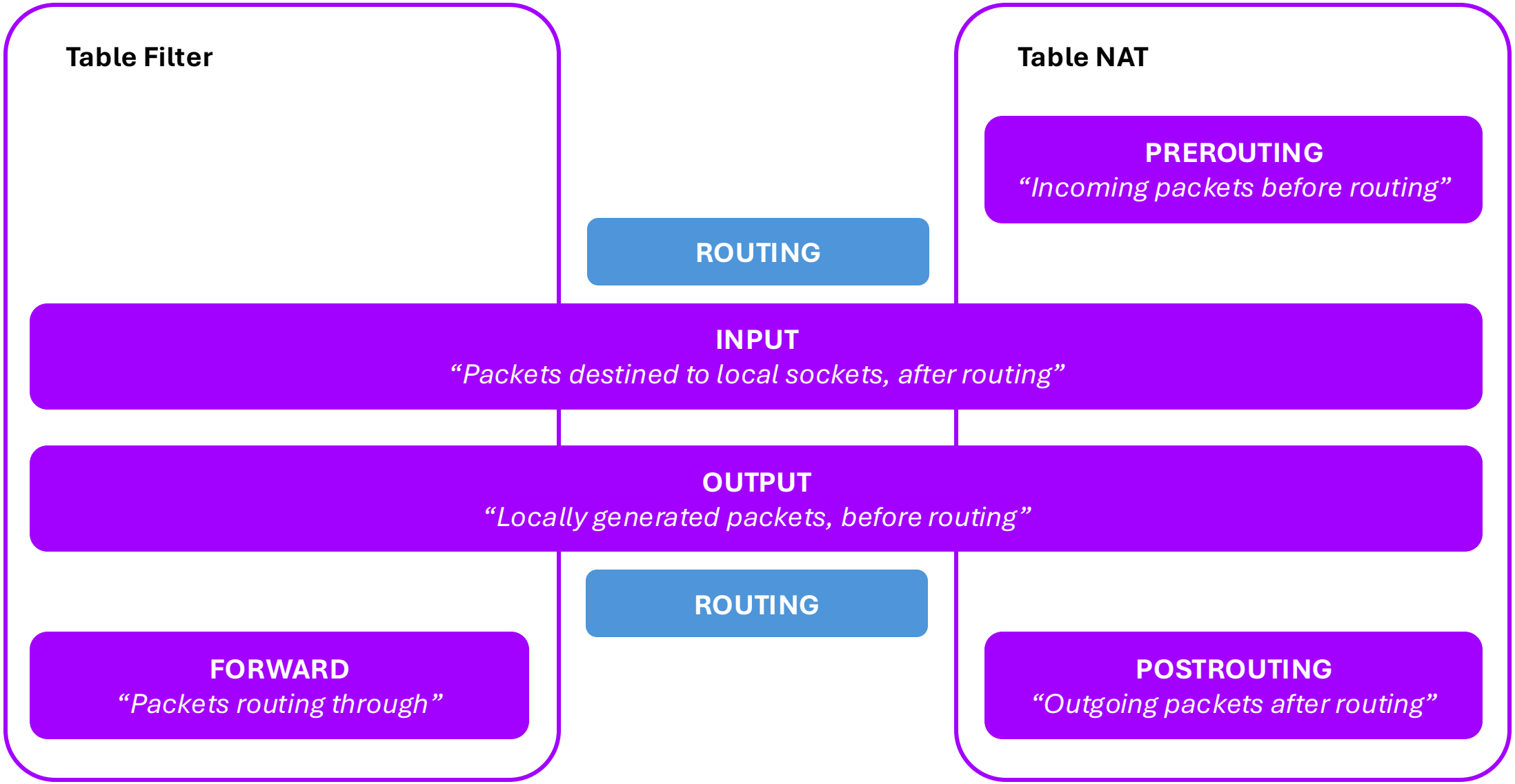
Demo 2

Let’s wire up two namespaces using veth pairs and a bridge – still no Docker.

Linux offers iptables to filter and NAT packets and iproute2 for routing.

NAT changes packet addresses to allow access between separate networks.

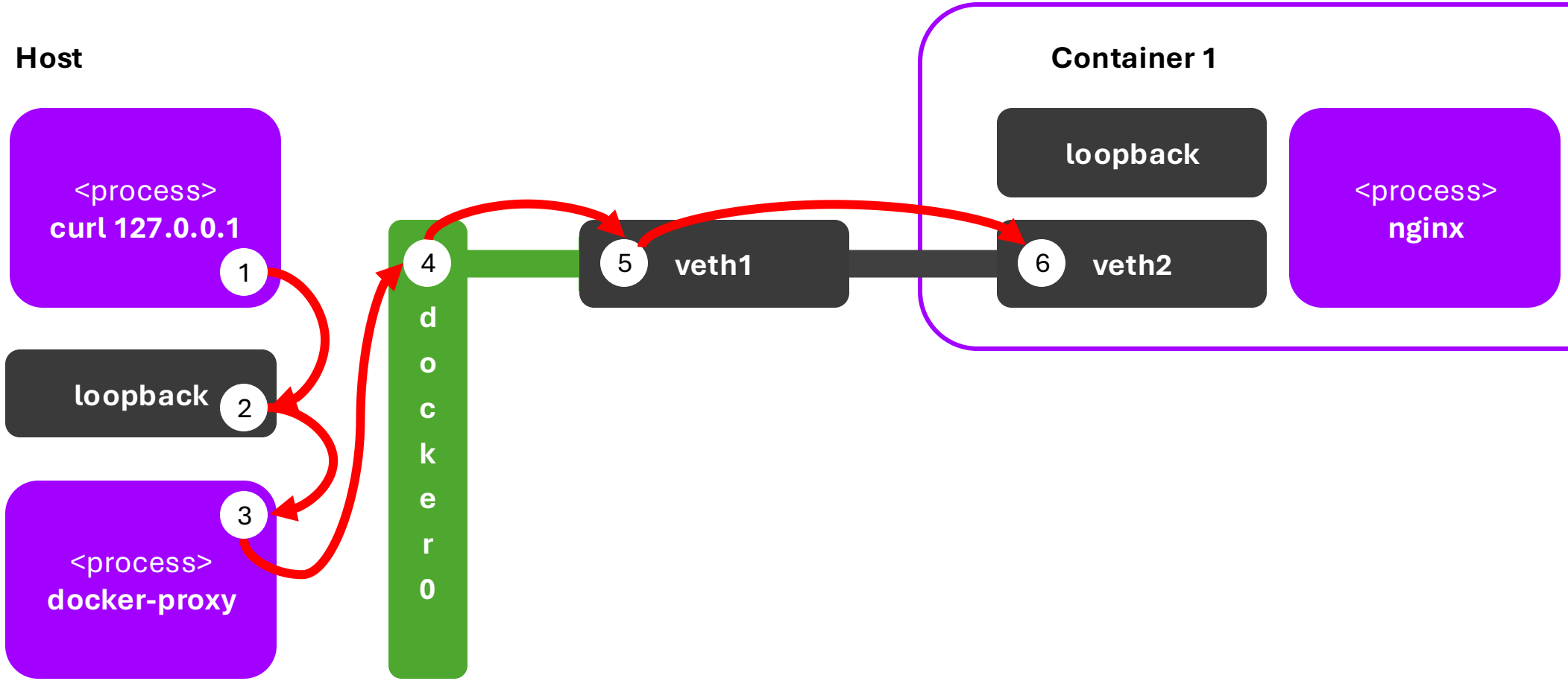
Routing is the process of **selecting the next hop** for a packet to reach its destination.

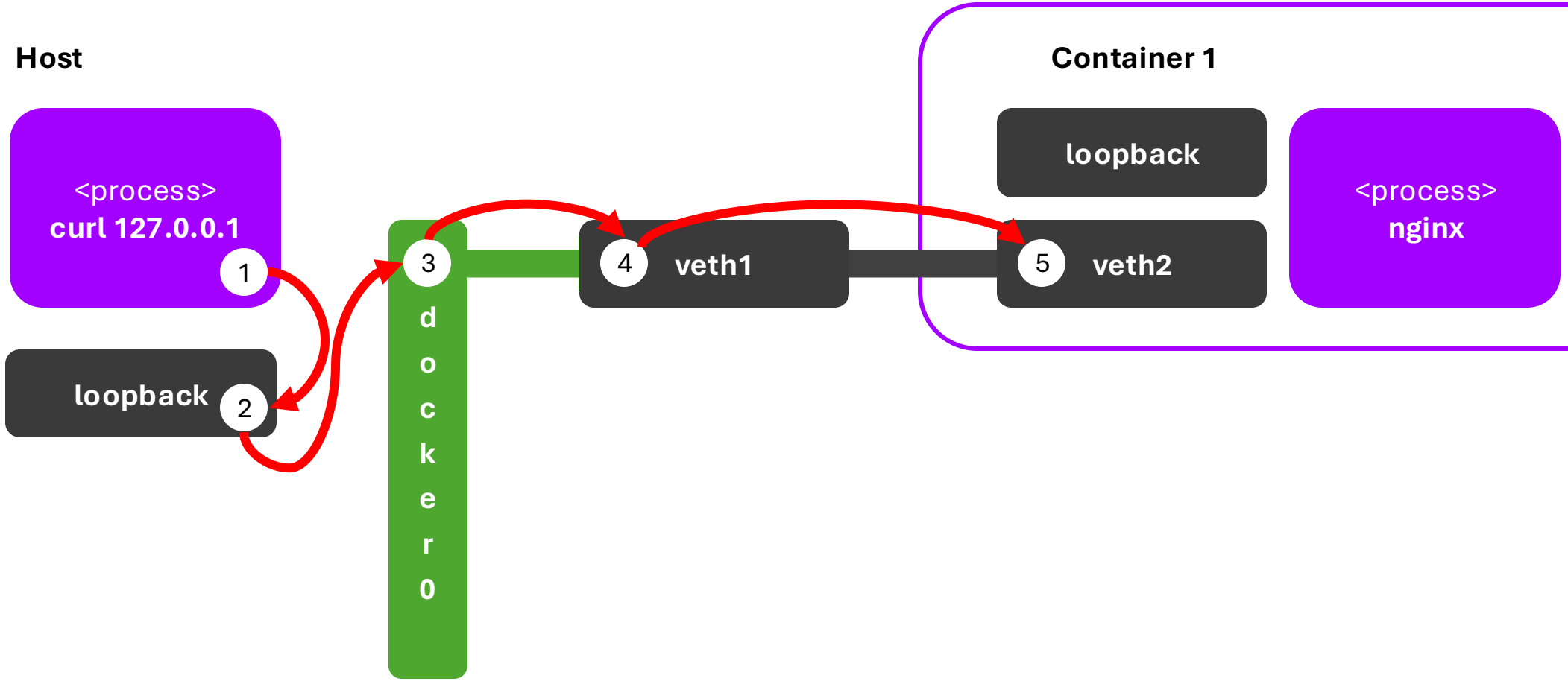


Note: Routing decision is a single step, shown twice here for clarity; bridges with br_netfilter behave differently; Path of packet depends on whether it is incoming, outgoing or being routed.

Docker deploys a **userland proxy** that handles some networking parts, **we disable it.**

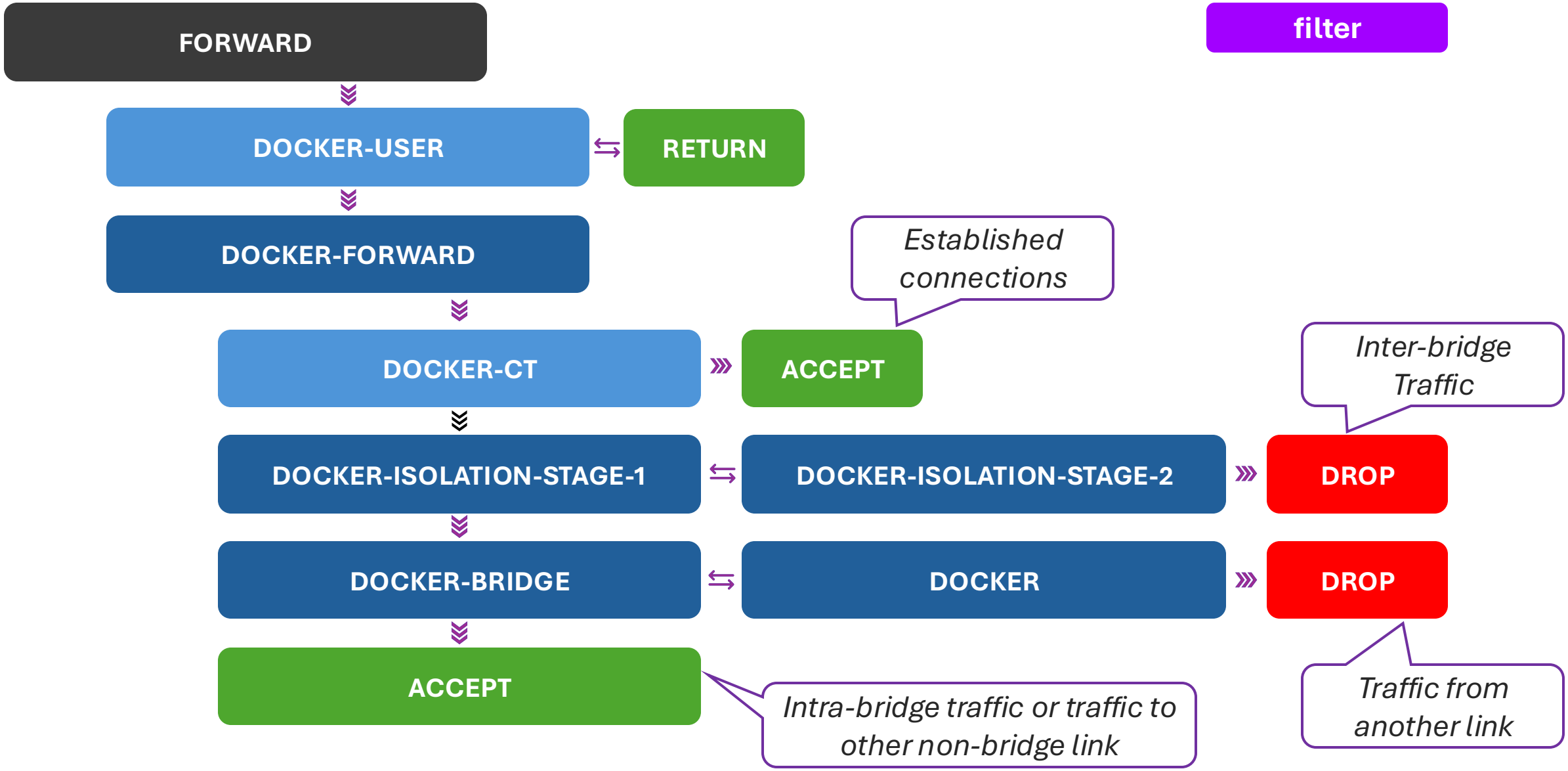
```
bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 8080 -container-ip 172.17.0.2 -conta
bin/docker-proxy -proto tcp -host-ip :: -host-port 8080 -container-ip 172.17.0.2 -container
```

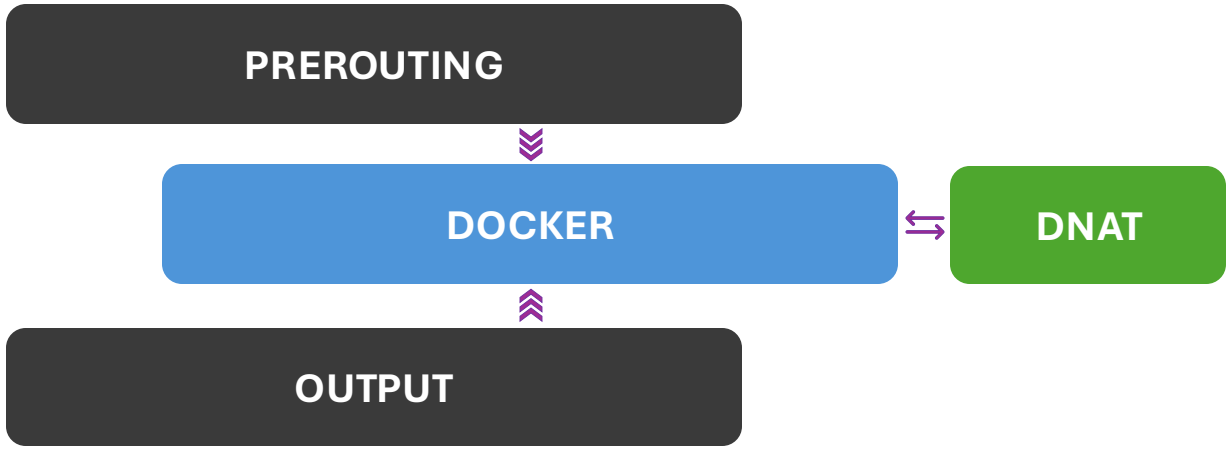

Host
Container 1
<process>
curl 127.0.0.1
1
loopback
2
<process>
docker-proxy
3
4
docker0
5
veth1
6
veth2
loopback
<process>
nginx


Host
Container 1
<process>
curl 127.0.0.1
1
loopback
2
3
d
o
c
k
e
r
0
4
veth1
loopback
5
veth2
<process>
nginx

Demo 3

Let’s explore the Docker networking setup.


FORWARD
filter
DOCKER-USER
RETURN
DOCKER-FORWARD
Established connections
DOCKER-CT
ACCEPT
Inter-bridge Traffic
DOCKER-ISOLATION-STAGE-1
DOCKER-ISOLATION-STAGE-2
DROP
DOCKER-BRIDGE
DOCKER
DROP
ACCEPT
Intra-bridge traffic or traffic to other non-bridge link
Traffic from another link



nat

Demo 4

Let’s setup a port forwarding after the container was created.

Thank you,

If you enjoyed the talk, follow me on LinkedIn

DEMO 1 (BARE VETH COMMUNICATION)

host

--> to ns1

```
ip link add veth1 type veth peer name veth2

ip link set veth1 netns ns1

ip addr add 10.0.0.2/24 dev veth2

ip link set dev veth2 up
```

--> to ns1

```
nc 10.0.0.1 8080
```

ns1

```
ip netns exec ns1 bash

ip link set dev lo up

ping <host> # not reachable
```

--> to host

```
ip addr add 10.0.0.1/24 dev veth1

ip link set dev veth1 up

nc -l 8080
```

--> to host

DEMO 1.5 (SECOND VETH)

host

--> to ns2

ns1

```
ping 10.0.0.3
```

ns2

```
ip netns add ns3

ip link add veth3 type veth peer name veth4

ip link set veth3 netns ns3

ip addr add 10.0.0.4/24 dev veth4

ip link set dev veth4 up

ip link show veth4 # show LOWERLAYERDOWN

ip netns exec ns3 bash

ip link set dev lo up

ip addr add 10.0.0.3/24 dev veth3

ip link set dev veth3 up

ping 10.0.0.1
```

--> to ns1

DEMO 2 (BRIDGE)

host

```
ip addr flush dev veth2

ip addr flush dev veth4

ip link add bridge0 type bridge

ip addr add 10.0.0.10/24 dev bridge0

ip link set bridge0 up

ip link set veth2 master bridge0

ip link set veth4 master bridge0

ip route # should be `10.0.0.0/24 dev bridge0 proto kernel scope link src 10.0.0.10`
```

--> to ns1

```
iptables -L FORWARD # show policy drop

iptables -I FORWARD -o bridge0 -j ACCEPT
```

--> to ns2

ns1

```
ping 10.0.0.10
```

--> to ns2

```
nc 10.0.0.3 8080
```

ns2

```
ping 10.0.0.10

ping 10.0.0.1 # unreachable
```

--> to host

```
ping 10.0.0.1

nc -l 8080
```

--> to ns1

DEMO 3 (DOCKER NETWORK ISOLATION)

host

```
ip netns del ns1

ip netns del ns3

ip link del bridge0

docker network ls

ip link

docker network create dummy1

docker network create dummy2

ip addr

docker run --network=dummy1 --name=dummy1 -it jonlabelle/network-tools

docker run --network=dummy2 --name=dummy2 -it jonlabelle/network-tools
```

--> to container 2

```
iptables -t filter -L -v # show filtered packages in isolation stage
```

container 1

```
ip addr

ping 172.19.0.2
```

--> to host

container 2

```
ip addr

ping 172.18.0.2
```

--> to container 1

DEMO 4 (DOCKER PORT FORWARDING)

host

```
docker stop dummy1 dummy2

docker rm dummy1 dummy2

docker network rm dummy1 dummy2

docker run -p 8888:80 --name=nginx -d nginx

curl localhost:8888

iptables -t nat -L -v

docker stop nginx

docker rm nginx

docker run -d nginx

docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' <container>

curl 172.17.0.2:80

curl localhost:8888

iptables -t nat -I DOCKER -p tcp -m tcp --dport 8888 -j DNAT --to-destination 172.17.0.2:80
```