Linux Networking Explained

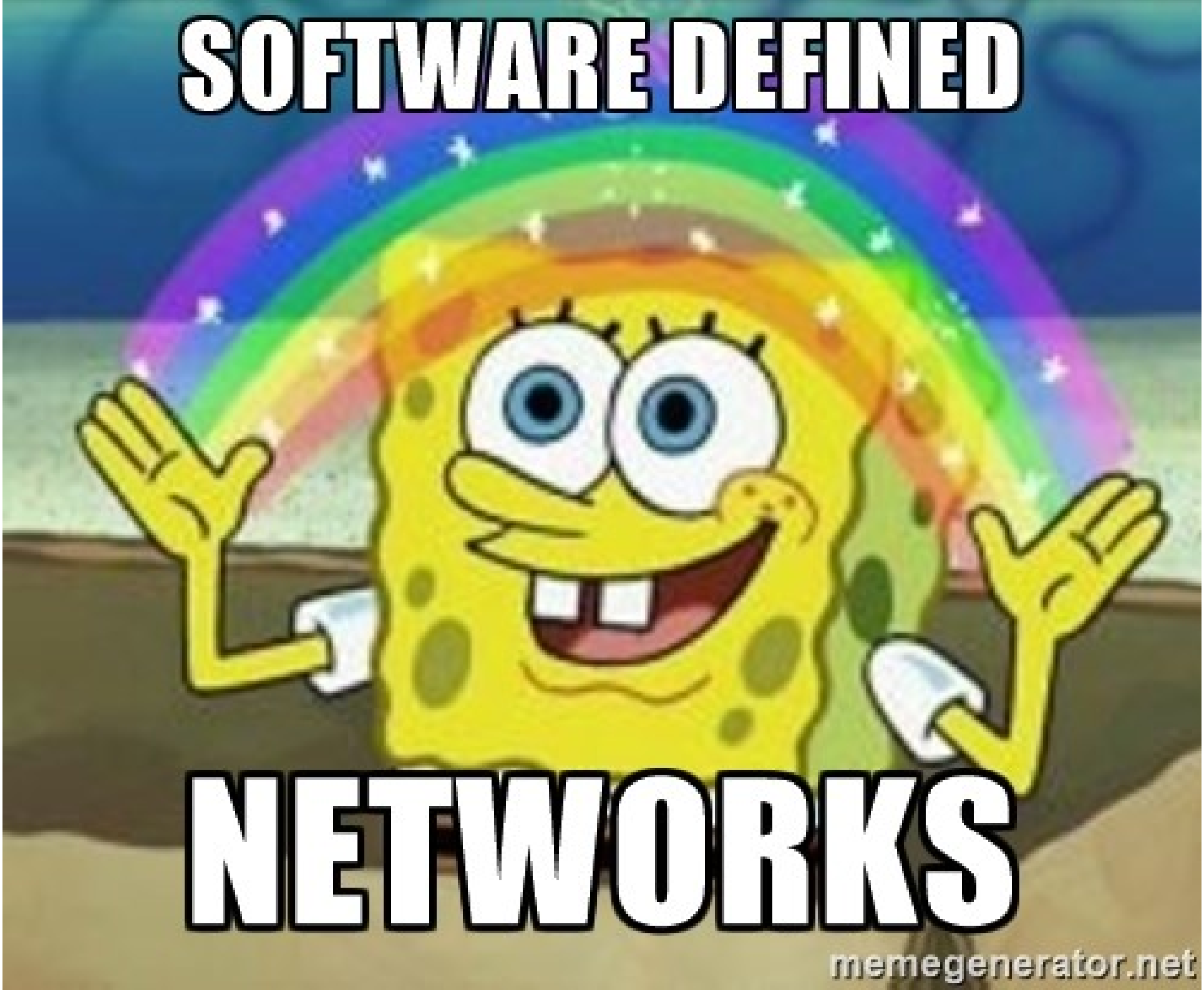
LinuxCon 2016, Toronto

Thomas Graf (@tgraf__)
Kernel, Cilium & Open vSwitch Team
Noiro Networks (Cisco)

Did you catch part I?

- Part II: LinuxCon, Toronto, 2016
  Linux Networking Explained
  Network devices, Namespaces, Routing, Veth, VLAN, IPVLAN, MACVLAN, MACVTAP, Bonding, Team, OVS, Bridge, BPF, IPSec

- Part I: LinuxCon, Seattle, 2015
  Kernel Networking Walkthrough
  The protocol stack, sockets, offloads, TCP fast open, TCP small queues, NAPI, busy polling, RSS, RPS, memory accounting
  http://goo.gl/ZKJpor


SOFTWARE DEFINED
NETWORKS
memegenerator.net

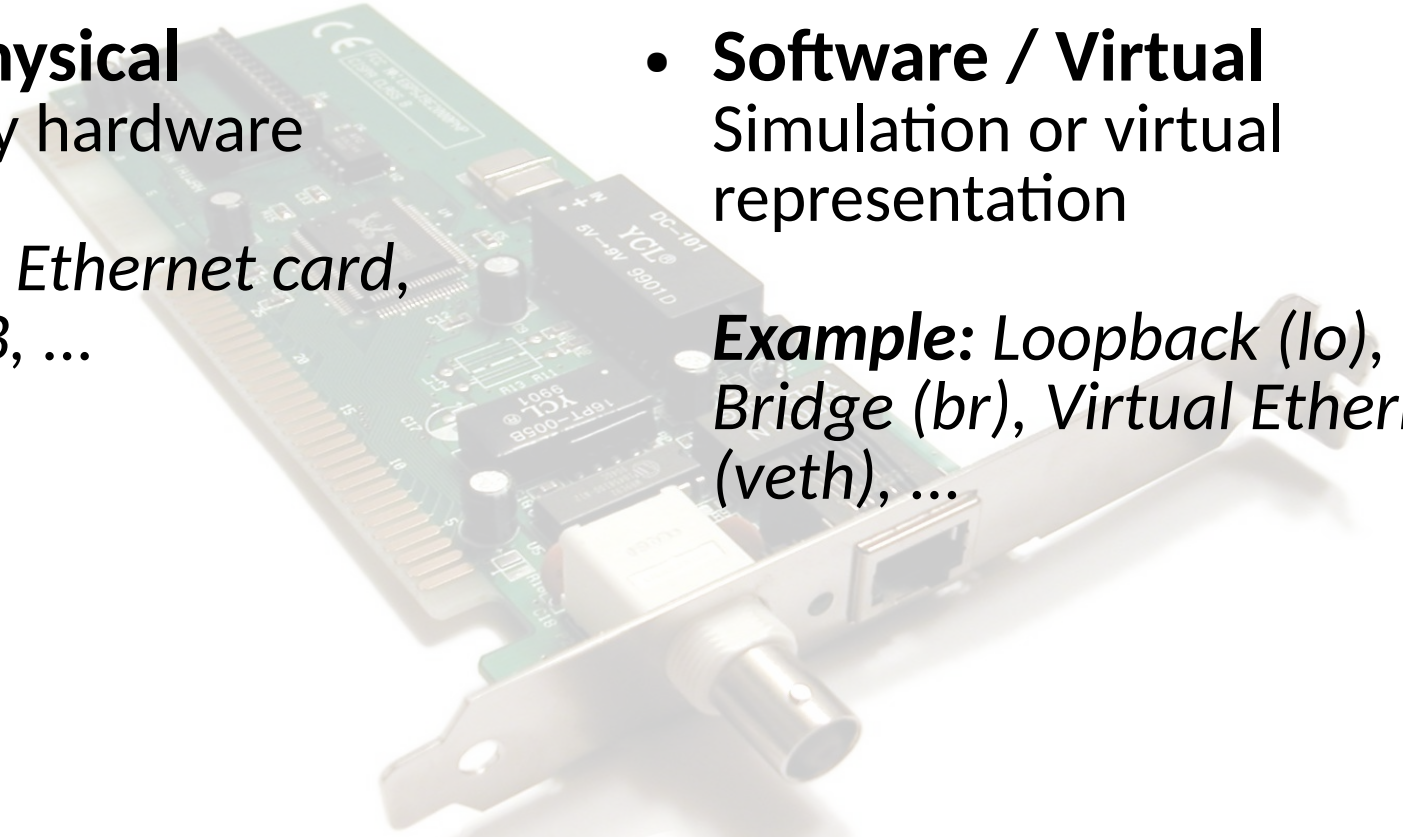
Network Devices

- **Real / Physical**
  Backed by hardware

  Example: *Ethernet card, WIFI, USB, ...*

- **Software / Virtual**
  Simulation or virtual representation

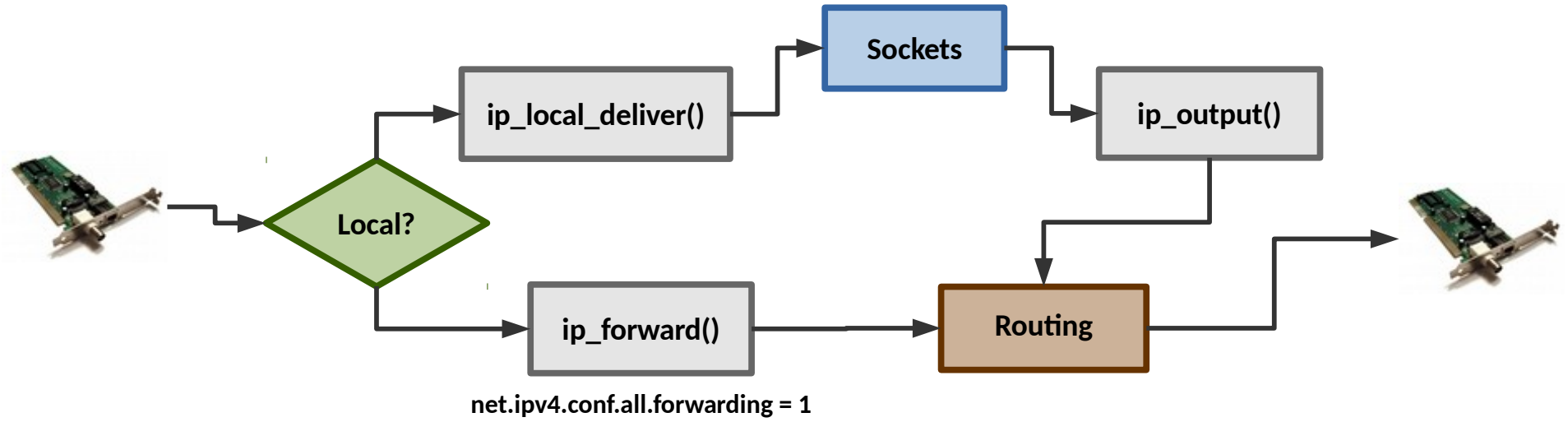
  Example: *Loopback (lo), Bridge (br), Virtual Ethernet (veth), ...*



```
$ ip link
[...]
$ ip link show enp1s0f1
4: enp1s0f1: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state [...]
    link/ether 90:e2:ba:61:e7:45 brd ff:ff:ff:ff:ff:ff
```

Addresses

Do we need to consider a packet for local sockets?




```
$ ip addr add 192.168.23.5/24 dev em1
$ ip address show dev em1
2: em1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP [...]
    link/ether 10:c3:7b:95:21:da brd ff:ff:ff:ff:ff:ff
    inet 192.168.23.5/24 brd 192.168.23.255 scope global em1
       valid_lft forever preferred_lft forever
    inet6 fe80::12c3:7bff:fe95:21da/64 scope link
       valid_lft forever preferred_lft forever
```

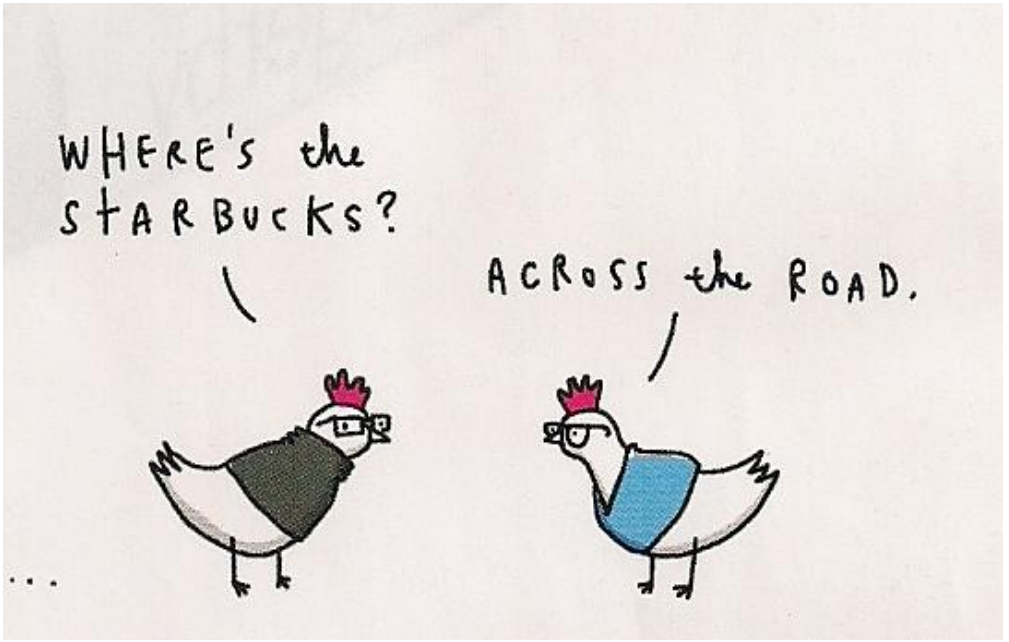
Pro Tip: The Local Table

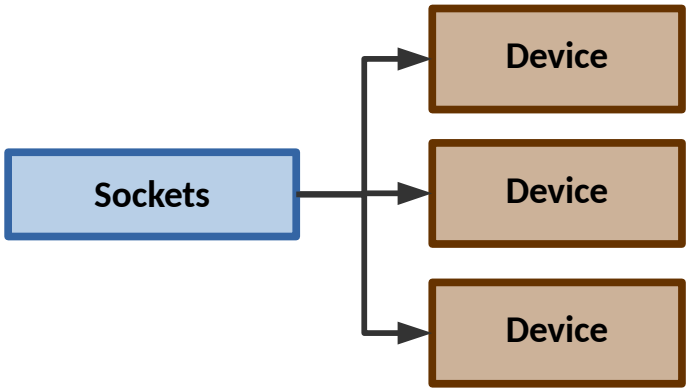
List all accepted local addresses:

```
$ ip route list table local type local
127.0.0.0/8 dev lo  proto kernel  scope host  src 127.0.0.1
127.0.0.1 dev lo  proto kernel  scope host  src 127.0.0.1
192.168.23.5 dev em1  proto kernel  scope host  src 192.168.23.5
192.168.122.1 dev virbr0  proto kernel  scope host  src 192.168.122.1
```

H4x0r Tip: You can also modify this table after the generated local routes have been inserted.

Routing





Direct Route - endpoints are direct neighbours (L2)

```
$ ip route add 10.0.0.0/8 dev em1
$ ip route show
10.0.0.0/8 dev em1 scope link
```

Nexthop Route - endpoints are behind another router (L3)

```
$ ip route add 20.10.0.0/16 via 10.0.0.1
$ ip route show
20.10.0.0/16 via 10.0.0.1 dev em1
```

Pro Trick: Simulating a Route Lookup

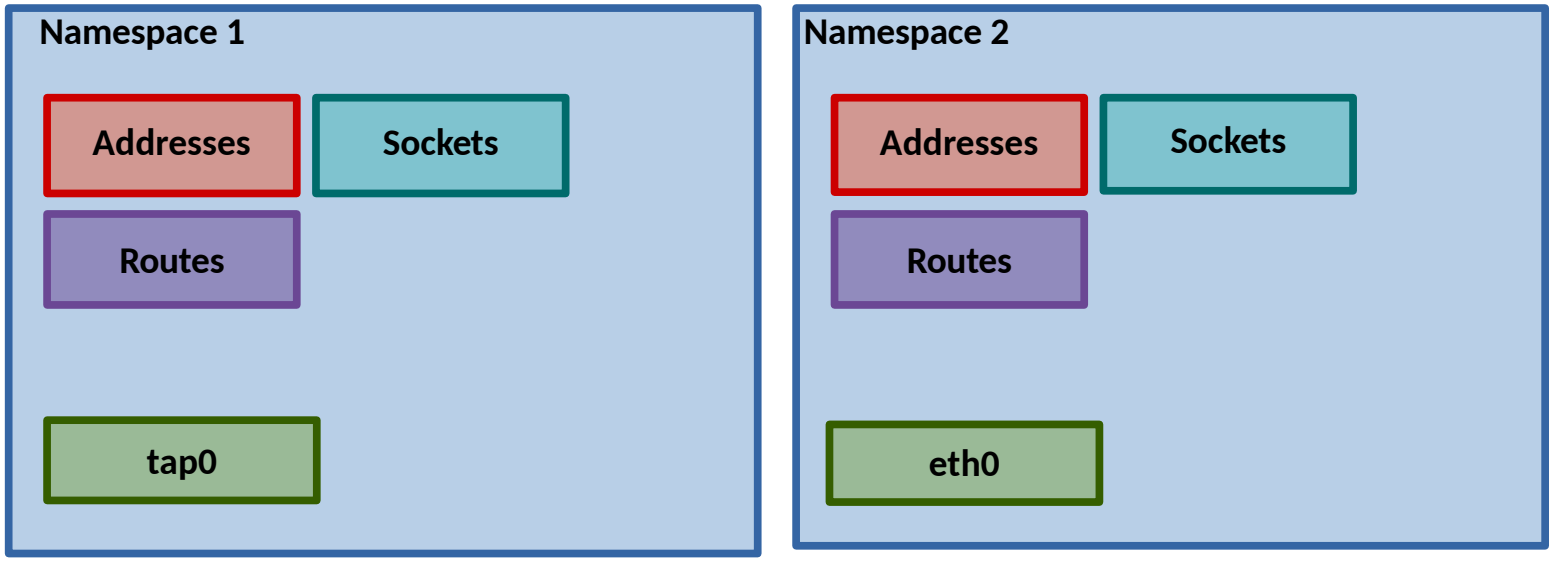
How will a packet to 20.10.3.3 get routed?

```
$ ip route get 20.10.3.3
20.10.3.3 via 10.0.0.1 dev em1  src 192.168.23.5
    cache
```

NOTE: This is not just **$(ip route show | grep)**. It performs an actual route lookup on the specified destination address in the kernel.

Network Namespaces

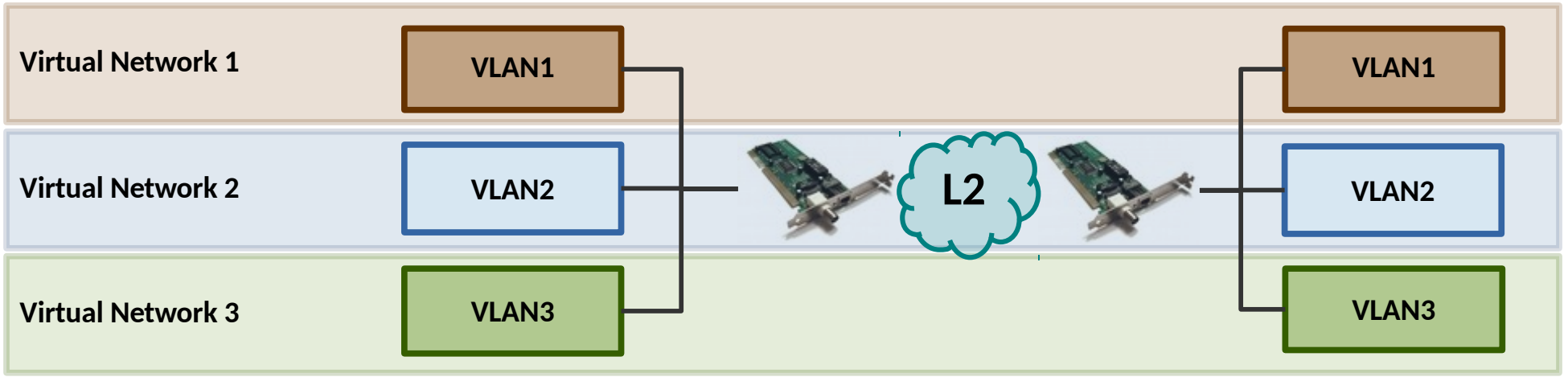
Linux maintains resources and data structures per namespace



NOTE: Not all data structures are namespace aware yet!

```
$ ip netns add blue
$ ip link set tap0 netns blue
$ ip netns exec blue ip address
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
19: tap0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether 42:ad:d0:10:e0:67 brd ff:ff:ff:ff:ff:ff
```

VLAN
Virtual Networks on Layer 2

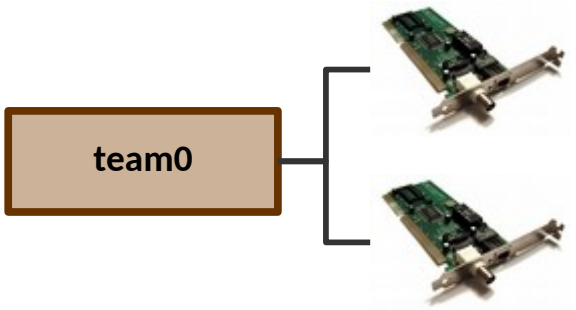


Packet Headers:


```
$ ip link add link em1 vlan1 type vlan id 1
$ ip link set vlan1 up
$ ip link show vlan1
15: vlan1@em1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP [...]
    link/ether 10:c3:7b:95:21:da brd ff:ff:ff:ff:ff:ff
```

Bonding / Team
Link Aggregation

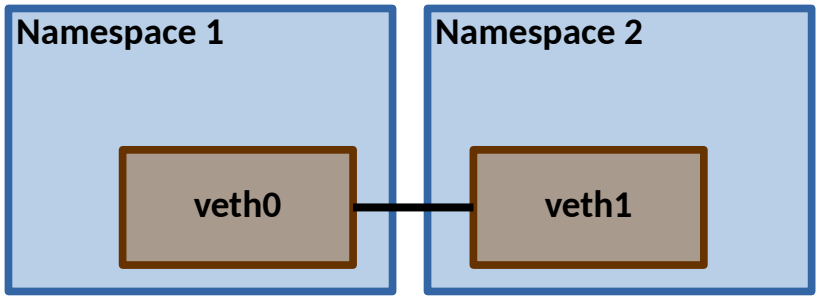
- **Uses**:
  - Redundant network cards (failover)
  - Connect to multiple ToR (LB)
- **Implementations**:
  - **Team** (new, user/kernel)
  - **Bonding** (old, kernel only)




```
$ cp /usr/share/doc/teamd-*/example_configs/activebackup_ethtool_1.conf .
$ teamd -g -f activebackup_ethtool_1.conf -d
[...]
$ teamdctl team0 state
[...]
```

Veth
Virtual Ethernet Cable

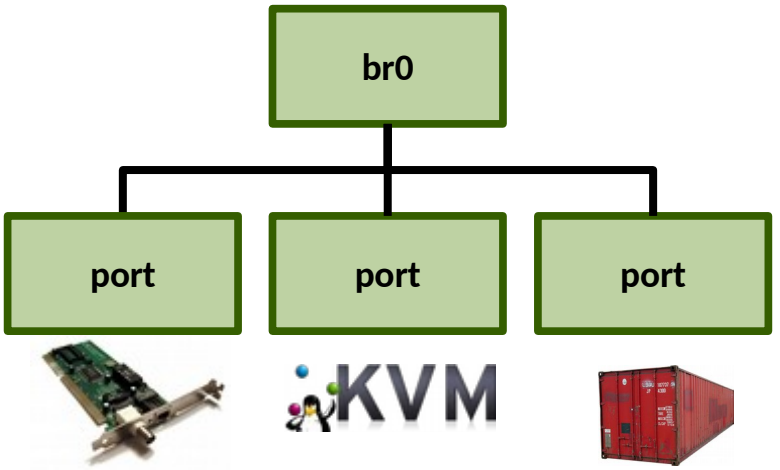
- Bidirectional FIFO
- Often used to cross namespaces



```
$ ip link add veth1 type veth peer name veth2
$ ip link set veth1 netns ns1
$ ip link set veth2 netns ns2
```

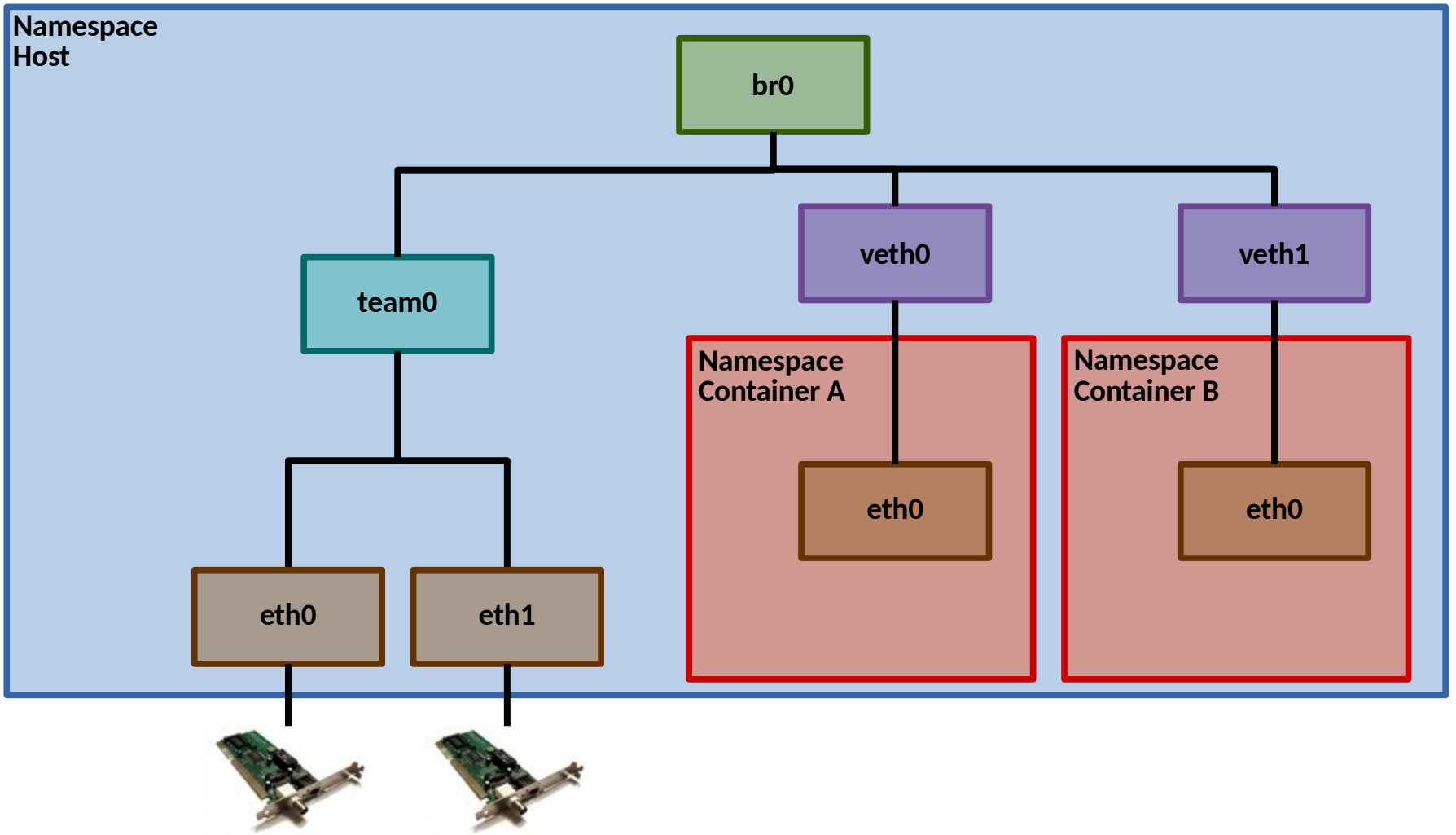
Bridge
Virtual Switch

- **Flooding:** Clone packets and send to all ports.
- **Learning:** Learn who's behind which port to avoid flooding
- **STP:** Detect wiring loops and disable ports
- **Native VLAN integration**
- **Offload:** Program HW based on FDB table



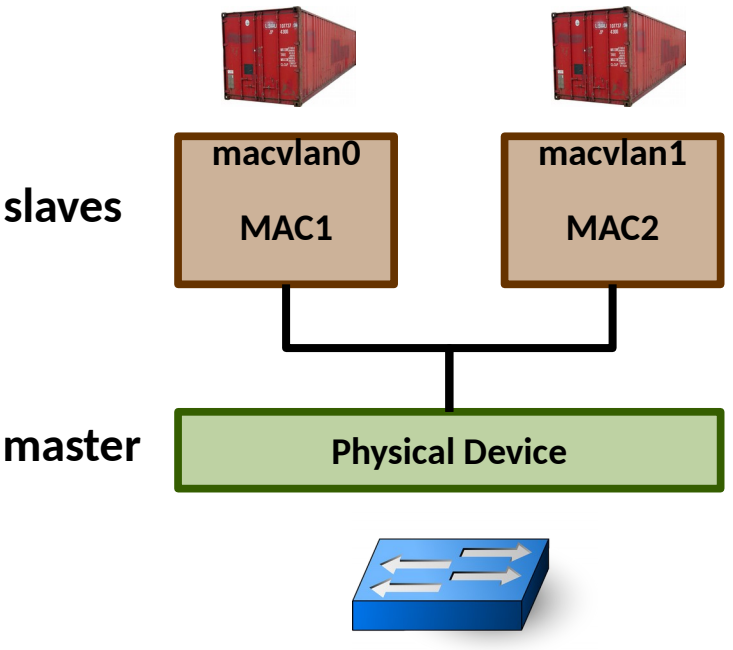

```
$ ip link add br0 type bridge
$ ip link set eth0 master br0
$ ip link set tap3 master br0
$ ip link set br0 up
```

Example
Bridge + Team + Veth



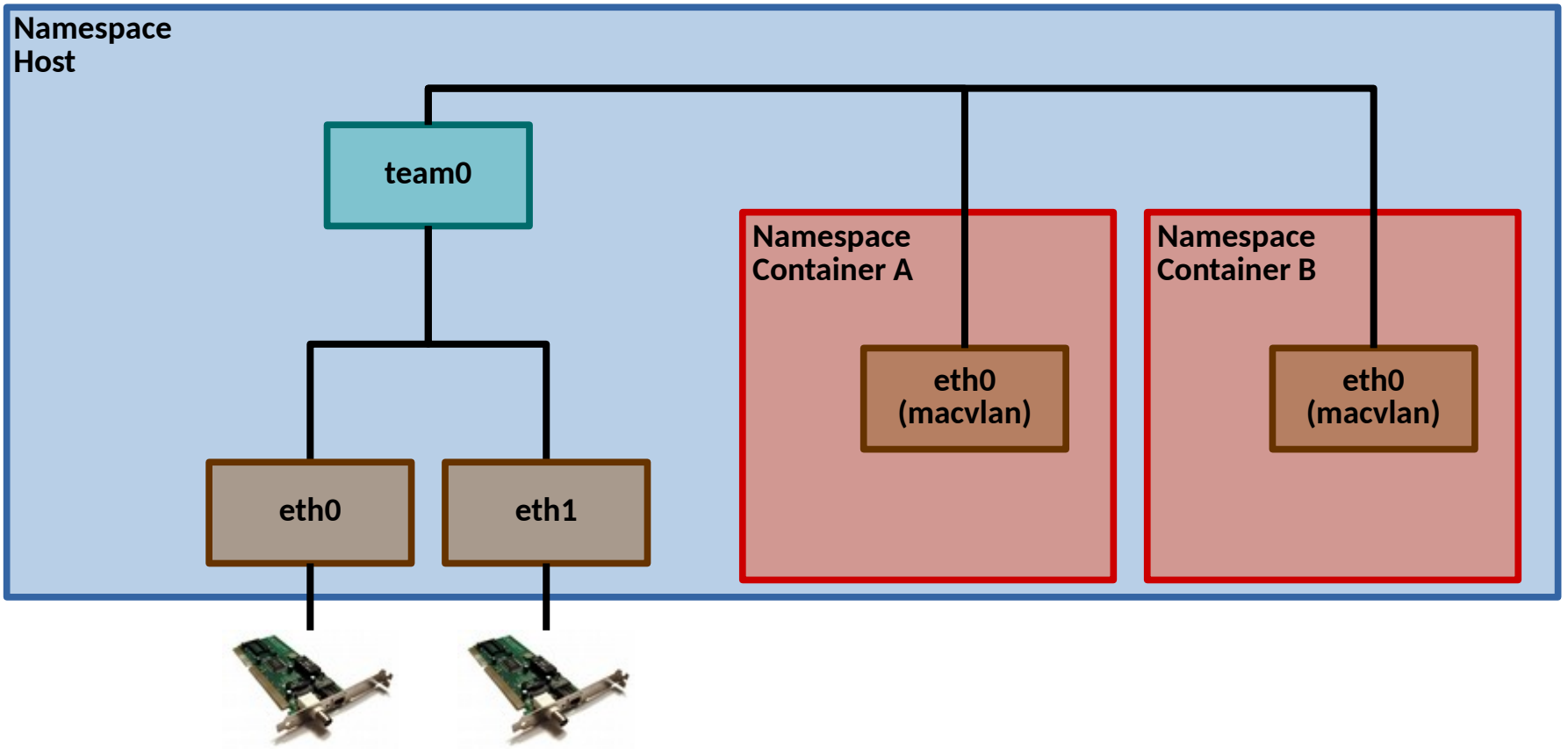
MACVLAN
Simplified bridging for guests

- **NOT** 802.1Q VLANs
- Multiple MAC addresses on single interface
- KISS - no learning, no STP
- Modes:
  - **VEPA (default):** Guest to guest done on ToR, L3 fallback possible
  - **Bridge:** Guest to guest in software
  - **Private:** Isolated, no guest to guest
  - **Passthrough:** Attaches VF (SR-IOV)



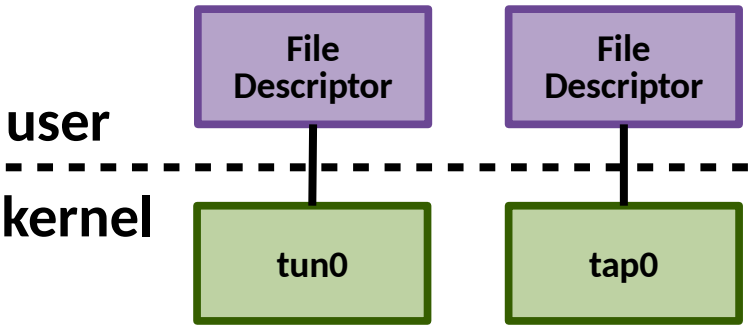

```
$ ip link add link em1 name macvlan0 type macvlan mode bridge
$ ip -d link show macvlan0
23: macvlan0@em1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN [...]
    link/ether f2:d8:91:54:d0:69 brd ff:ff:ff:ff:ff:ff promiscuity 0
    macvlan  mode bridge addrgenmode eui64
$ ip link set macvlan0 netns blue
```

Example
Team + MACVLAN



TUN/TAP
A gate to user space

- Character Device in user space
- Network device in kernel space
- L2 (TAP) or L3 (TUN)
- Uses: encryption, VPN, tunneling, virtual machines, …



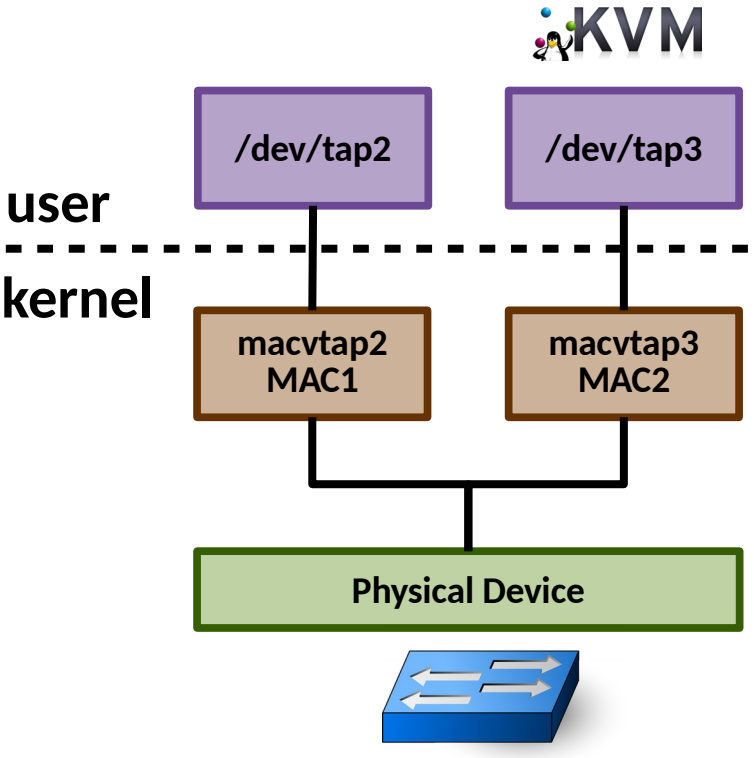

```
$ ip tuntap add tun0 mode tun
$ ip link set tun0 up
$ ip link show tun0
18: tun0: <NO-CARRIER,POINTOPOINT,MULTICAST,NOARP,UP> mtu 1500 qdisc fq_codel [...]
    link/none
$ ip route add 10.1.1.0/24 dev tun0
```

user.c:

```
fd = open("/dev/net/tun", O_RDWR);
strncpy(ifr.ifr_name, “tap0” , IFNAMSIZ);
ioctl(fd, TUNSETIFF, (void *) &ifr);
```

MACVTAP
Bridge + TAP = MACVTAP

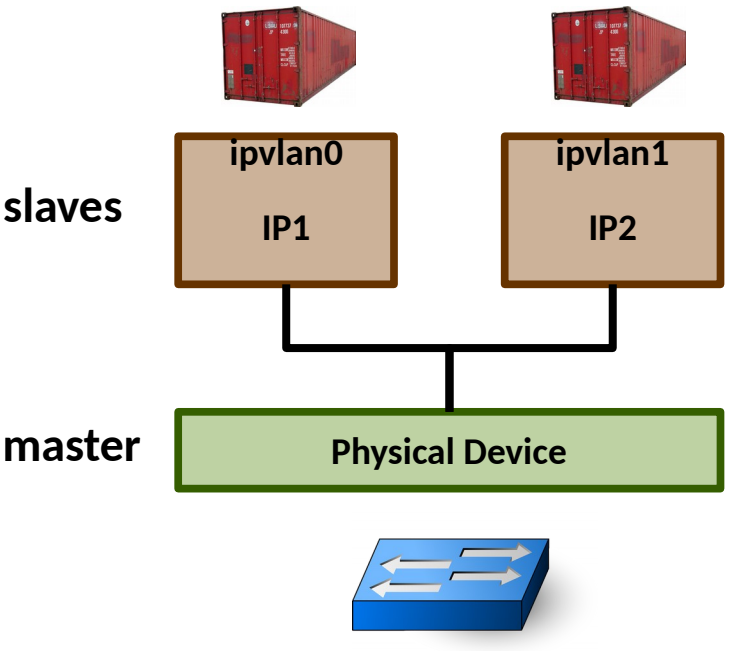
- A TAP with an integrated bridge
- Connects VM/container via L2
- Same modes as MACVLAN




```
$ ip link add link em1 name macvtap0 type macvtap mode vepa
$ ip -d link show macvtap
20: macvtap0@em1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP [...]
    link/ether 3e:cb:79:61:8c:4b brd ff:ff:ff:ff:ff:ff
    macvtap  mode vepa addrgenmode eui64
$ ls -l /dev/tap20
crw-------. 1 root root 241, 1 Aug  8 21:08 /dev/tap20
```

IPVLAN
MACVLAN for Layer 3 (L3)

- Can hide many containers behind a single MAC address.
- Shared L2 among slaves
- Mode:
  - L2: Like MACVLAN w/ single MAC
  - L3: L2 deferred to master namespace, no multicast/broadcast

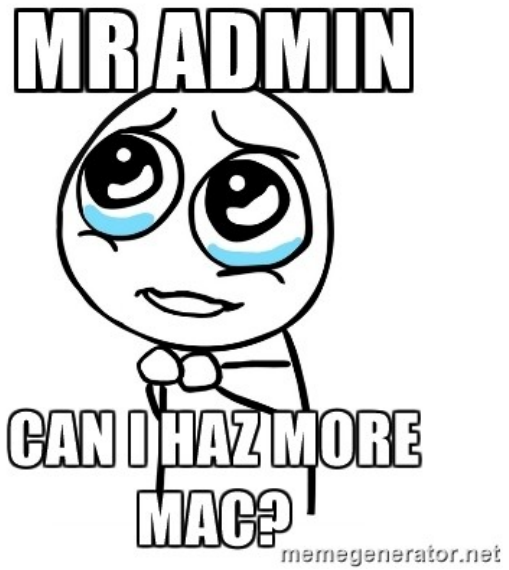



```
$ ip netns add blue
$ ip link add link eth0 ipvl0 type ipvlan mode l3
$ ip link set dev ipvl0 netns blue
$ ip netns exec blue ip link set dev ipvl0 up
$ ip netns exec blue ip addr add 10.1.1.1/24 dev ipvl0
```

MACVLAN vs IPVLAN

MACVLAN

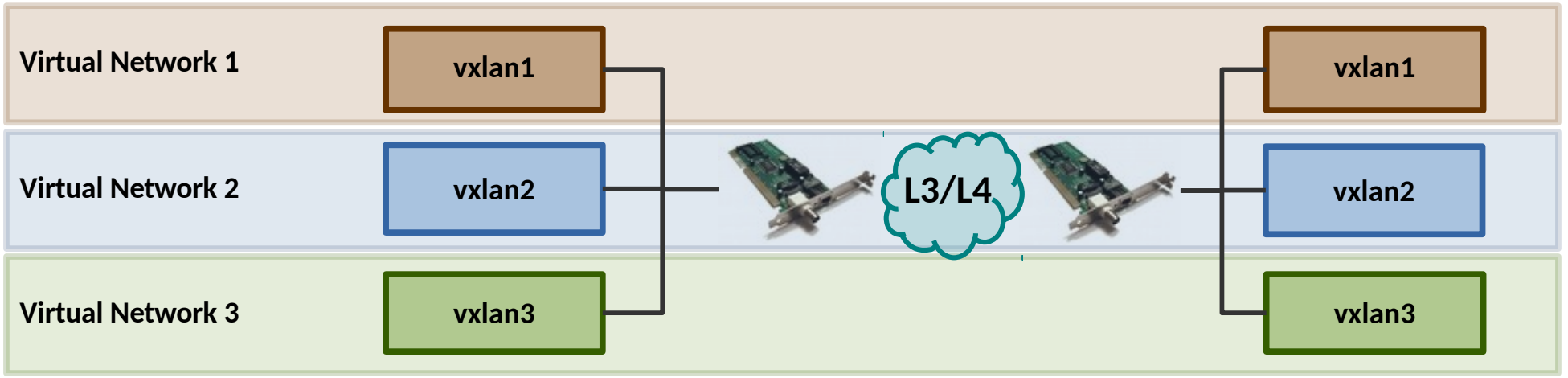
- ToR or NIC may have maximum MAC address limit
- Doesn't work well with 802.11 (wireless)



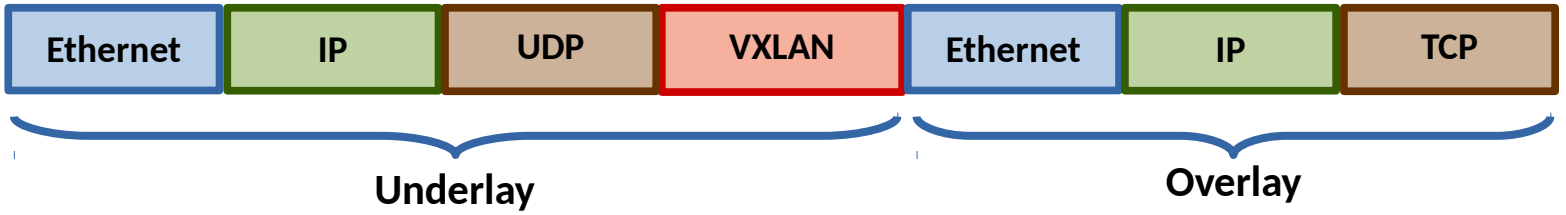
IPVLAN

- DHCP based on MAC doesn't work, must use client ID
- EUI-64 IPv6 addresses generation issues
- No broadcast/multicast in L3 mode

Encapsulation (Tunnels)
Virtual Networks on Layer 3/4

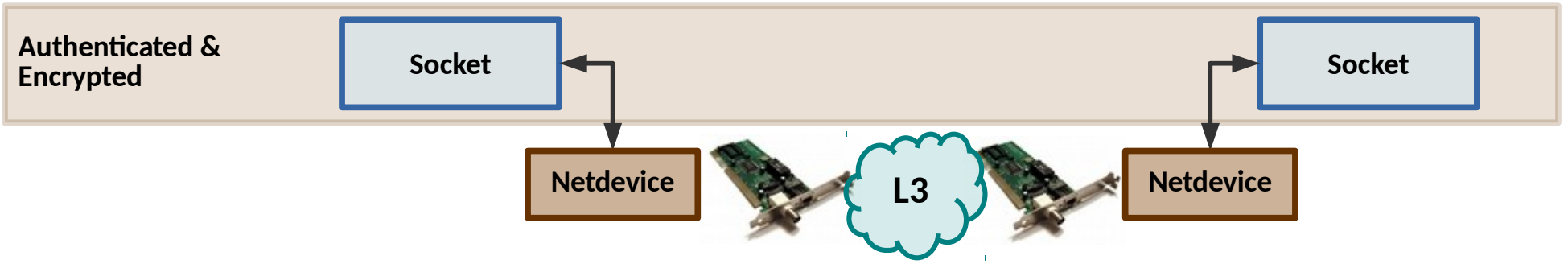


VXLAN Headers example:




```
$ ip link add vxlan42 type vxlan id 42 group 239.1.1.1 dev em1 dstport 4789
$ ip link set vxlan42 up
$ ip link show vxlan42
31: vxlan42: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UNKNOWN [...]
    link/ether e6:fc:c8:7e:07:83 brd ff:ff:ff:ff:ff:ff
```

IPSec

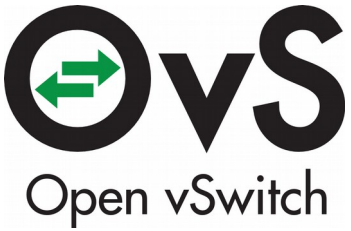


Transport Mode

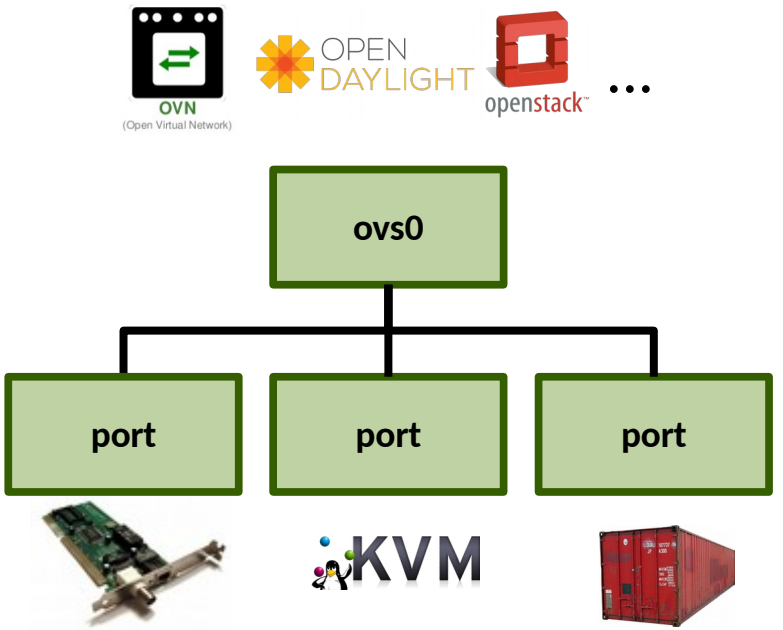
Tunnel Mode

- **AH:** Authentication
- **ESP:** Authenication + encryption

```
$ ip xfrm state add src 192.168.211.138 dst 192.168.211.203 proto esp \
     spi 0x53fa0fdd mode transport reqid 16386 replay-window 32 \
     auth "hmac(sha1)" 0x55f01ac07e15e437115dde0aedd18a822ba9f81e \
     enc "cbc(aes)" 0x6aed4975adf006d65c76f63923a6265b \
     sel src 0.0.0.0/0 dst 0.0.0.0/0
```

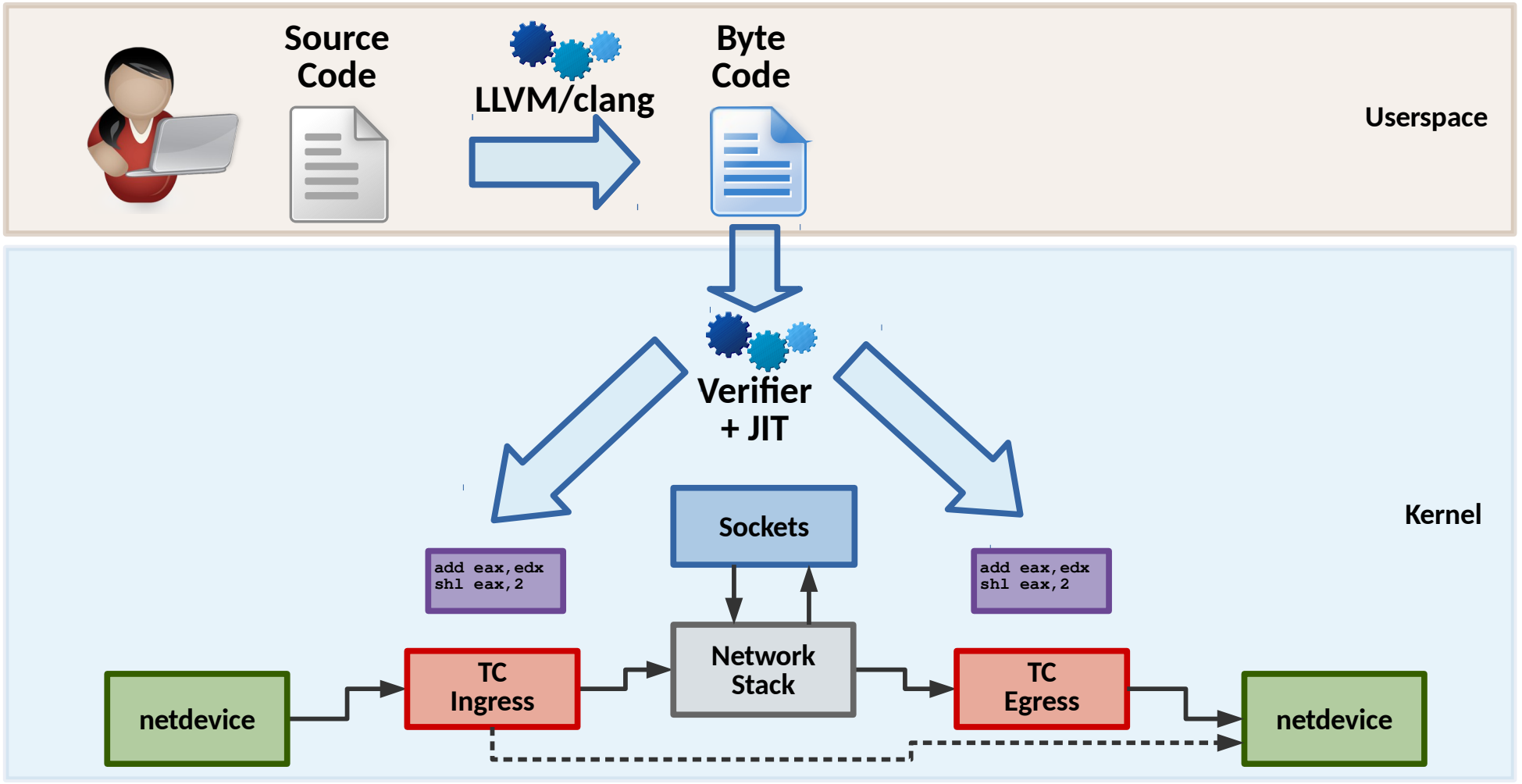


- Fully programmable L2-L4 virtual switch with APIs: OpenFlow and OVSDB
- Split into a user and kernel component
- Multiple control plane integrations:
  - OVN, ODL, Neutron, CNI, Docker, …




```
$ ovs-vsctl add-br ovs0
$ ovs-vsctl add-port ovs0 em1
$ ovs-ofctl add-flow ovs0 in_port=1,actions=drop
$ ovs-vsctl show
a425a102-c317-4743-b0ba-79d59ff04a74
    Bridge "ovs0"
        Port "em1"
            Interface "em1"
[...]
```

BPF



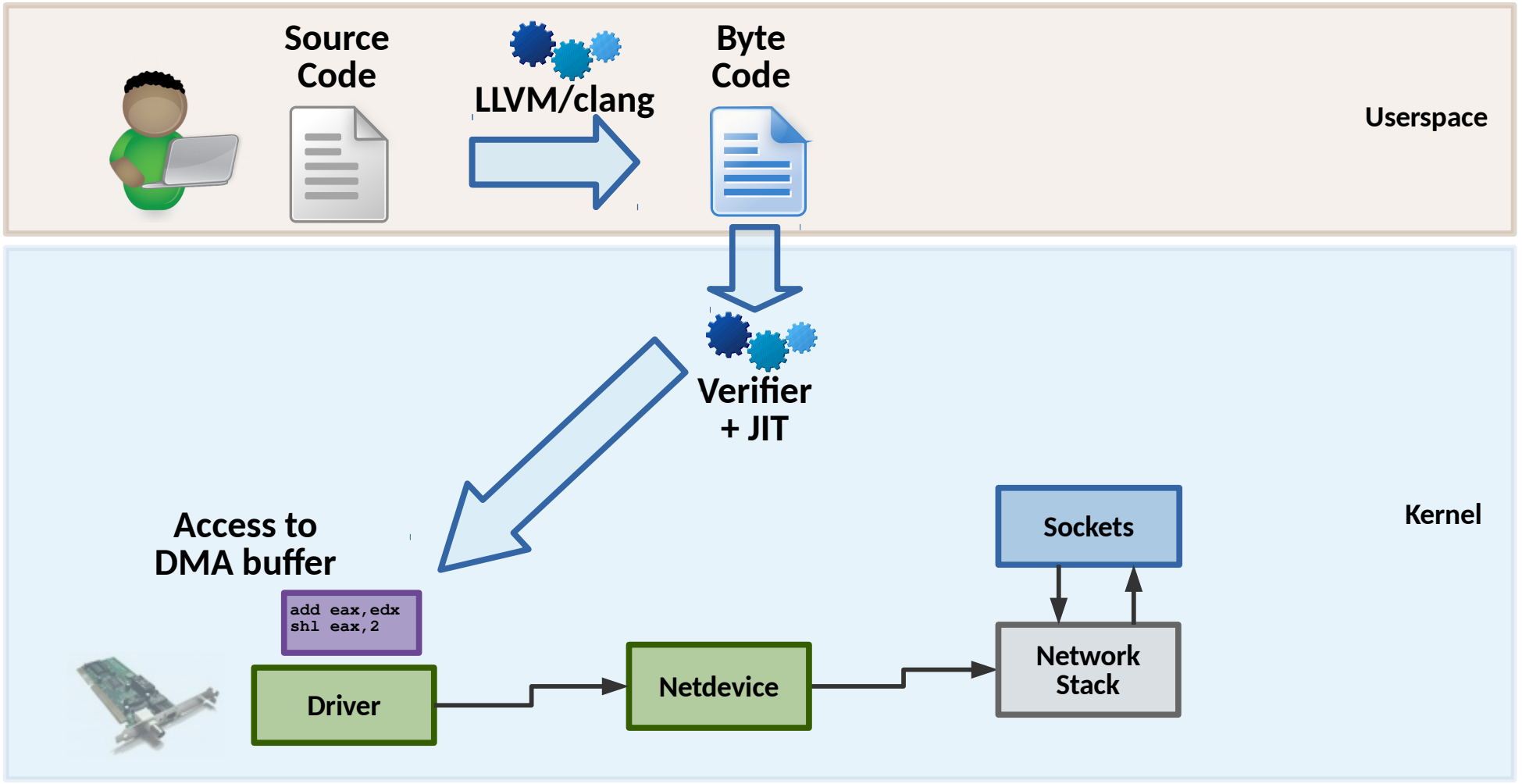
Attaching a BPF program to eth0 at ingress:

```
$ clang -O2 -target bpf -c code.c -o code.o
$ tc qdisc add dev eth0 clsact
$ tc filter add dev eth0 ingress bpf da obj code.o sec my-section1
$ tc filter add dev eth0  egress bpf da obj code.o sec my-section2
```

BPF Features
(As of Aug 2016)

- Maps
  - Arrays (per CPU), hashtables (per CPU)
- Packet mangling
- Redirect to other device
- Tunnel metadata (encapsulation)
- Cgroups integration
- Event notifications via perf ring buffer

XDP – Express Data Path



Q&A

Learn more about networking with BPF:
Fast IPv6-only Networking for Containers Based on BPF and XDP

Wednesday August 24, 2016 4:35pm – 5:35pm, Queen's Quay

Contact:

- **Twitter:** @tgraf__ **Mail:** tgraf@tgraf.ch

Image Sources:

- **Cover (Toronto)**
  Rick Harris (https://www.flickr.com/photos/rickharris/)
- **The Invisible Man**
  Dr. Azzacov (https://www.flickr.com/photos/drazzacov/)
- **Chicken**
  JOHN LLOYD (https://www.flickr.com/photos/hugo90/)