Virtualization

What is virtualization?

- Mechanism to substitute a hardware environment
  - For another, emulated environment
  - By any means
- Key to modern Cloud environments
  - Base for automation

Drivers for virtualization

- Expanding hardware capabilities
  - Allowing each single machine to do more simultaneous work
  - Need to increment utilization
- Server consolidation
  - Control costs
  - Simplify management
- Need to control large cluster installations
  - Server and render farms
- Improving security, reliability, and device independence
- Ability to run complex, OS-dependent applications in different hardware or OS environments

Goals

- Easily share a pool of computing resources
  - Elimination of manual resource management in CPDs
    - Increase reliability
  - Increase utilization of those resources
- Use same hardware for simultaneous different OS
  - E.g., run Linux on-windows (vice-versa), or on OSX, …
  - Increase usefulness of computing resource
- Use same hardware to emulate even other hardware
  - Run ARM on i86 (or viceversa)

Isolation

- A zone's actions cannot influence another zone's actions
  - Derived from the fact that they share computing resources
  - Even its failures are isolated
- Two aspects:
  - Access:
    - What permissions can one get to manipulate a resource
    - How much information can one isolated "zone" capture from another isolated zone
  - Quotas:
    - How much of a resource can one zone consume

Access Isolation

- A zone cannot access a resource on another zone
  - File
  - User
  - Process
  - Core
  - Memory contents at an address
  - Messages being transmitted/received
- Common aspects for access aspect:
  - Resource can be named
  - They are identifiable
    - They are named somehow

Quota Isolation

- A zone cannot use more of a resource that it is given
  - Disk space
  - Memory
  - CPU usage
  - IOPs
  - Network bandwidth
- Common aspects for quota aspect:
  - No individual resource is named
  - They are NOT identifiable
    - E.g. we do not care what GB of ram we are given
  - Only magnitudes are important

Isolation Relaxations

- Establish policies for accessing individual resources among zones
  - ACLs
  - Capabilities based on name resolutions
  - Problem:
    - Ensure the policies can be clearly established and properly enforced
- Permit limited competition for shared resources:
  - Map multiple pools of resources to the same underlying hardware resource

Virtualization approaches

- Two major types
  - Hardware/Machine virtualization
  - OS/Light virtualization
- Also other kinds of virtualization
  - Network Virtualization

Machine/Hardware virtualization

- The hardware is "emulated" by a piece of software
  - Originally, literally an interpreter of machine code
    - With enormous loss of performance
    - With high degree of access isolation
    - With poor degree of quota isolation
    - Main Goal:
      - Run different environments (Windows/Unix,…)
      - Run different HW architectures (ARM on Intel,…)
- Eventually allows direct access to the CPU
  - Still, access to other hardware parts limited
  - Software in charge: Hypervisor
  - Enhancement: "enlightened" kernel modules, calling Hypervisor
- Eventually, hardware support with specific machine instructions
  - Run by the Hypervisor/Host system.
- Eventually, hierarchical virtualization.

Machine/Hardware virtualization

- Produces what are known as VMs (Virtual Machines)
  - Access: individual Memory addresses
    - According to quotas
  - Individual devices
  - Individual disks and/or partitions
  - Individual cores
    - According to quotas
- Perfect partition/Overcommitting:
  - Maps each hardware resource to each VM
    - VM has exclusive access to hardware resource
  - More simulated resources than hardware resources (quota relaxation)
    - Different VMs compete for resources in a limited way

Machine/Hardware virtualization

- Two types of hypervisors: Type 1 and Type 2
- Type-1: Directly on Hardware
  - Example Hypervisors:
    - VMWARE ESXi
    - HYPER-V
    - Xen
    - IBM CP/CMS (oldie), and z/VM
    - Oracle/SPARC VM Server
    - …
- Type-II: Hosted on OS
  - VirtualBox
  - Parallels
  - Qemu
  - WMware Desktop/Player
- Not always clear distinction: KVM (Linux kernel module)

Network virtualization

- At various layers
  - E.g., layer2 physical network is simulated
    - Various flavors of virtual switches
    - Example: various vswitches
    - Example: various veth interfaces
    - Example: zerotier-one
  - Also layer3
    - Example: Wireguard
- Typically uses encapsulation techniques
- Gives rise to soft networks

OS/Light virtualization

- Oldest technique for time sharing
  - Processes
  - Virtual memory
  - File systems
  - …
- Isolation imperfections abound
  - Shared name spaces
    - Files/Process IDs/users/sockets
    - ACLs to limit access, but still wide potential holes in isolation
  - Shared software stack & runtimes
    - Libraries
    - Daemons
    - …
  - Competition for resources
    - Memory/CPU/IO
    - Limited quota enforcement mechanisms

OS/Light virtualization

- Consequence:
  - Large API surface to protect!!!
  - Too much relaxation in access (naming)
  - Too much relaxation in resource usage (quotas)
- Requirements
  - Severely reduce API surface
  - Fully isolate name spaces from which to obtain access to specific resource
  - Effectively cap the amount of resources an environment can make use of.

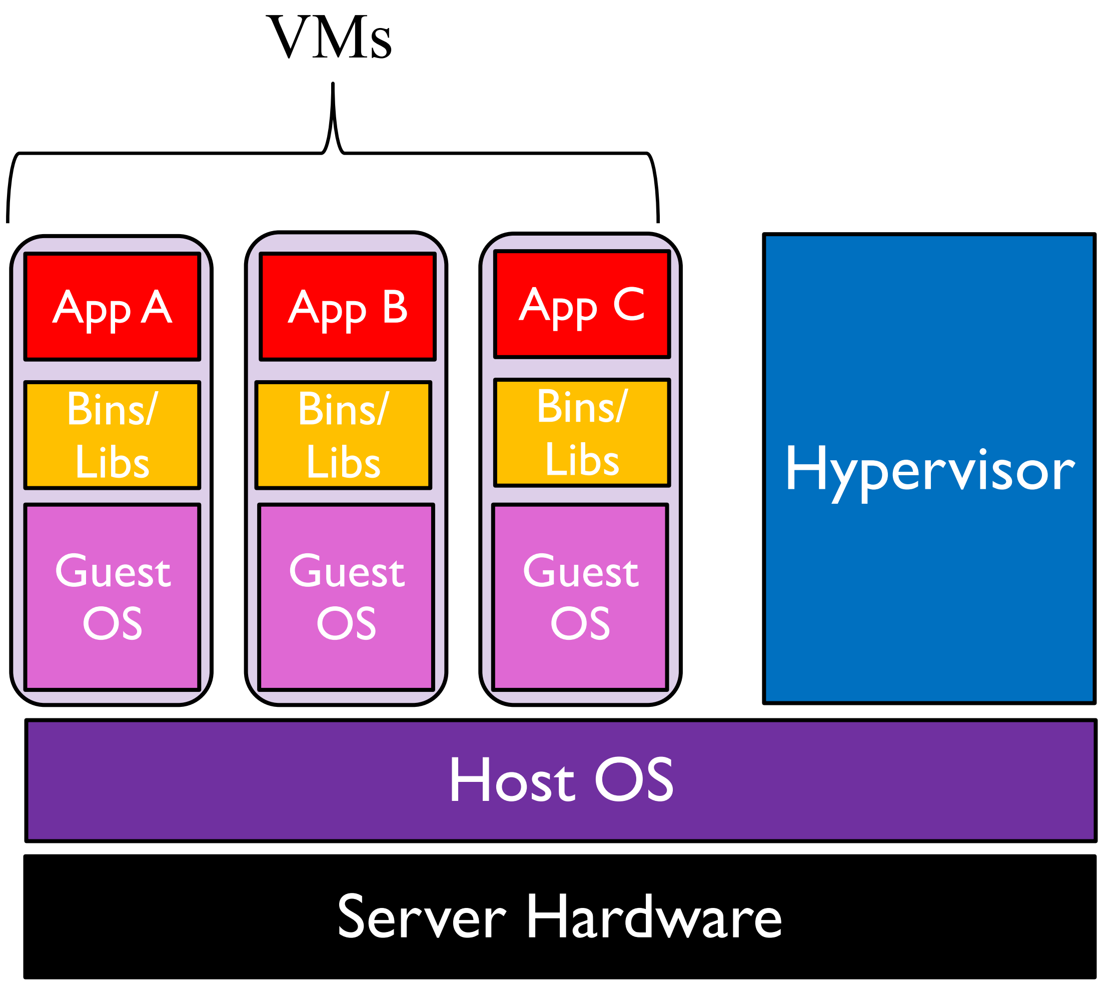
OS/Light virtualization

- Severely reduces surface AREA:
  - Only the kernel is shared
    - The rest of the software stack is **NOT** shared.
    - Kernel calls are the only surface area exposed.
- Kernel imposed namespace separation
  - Kernel is shared and it governs the rest of the OS
  - Kernel imposes the lowest level naming of objects
    - Mount points for file systems
    - Users, Groups, Process IDs, Network, devices, sockets,…
- Kernel imposed Resource consumption control
  - Quotas for memory/CPU/IOPS/Bandwidth
  - Mechanism to impose them on environments
- Environments built out of groups of processes on the OS.

Machine vs Light virtualization

- VMs
  - Need to boot a complete OS (the guest OS)
    - Boot process identical to that performed by a physical machine
    - Takes time
    - Takes more resources to simulate the machine environment
    - In general, needs to start many processes to provide the machine environment
  - Host OS can be completely different from guest OS
    - E.g. Windows/Linux
      - Windows/MacOSX
      - MacOSX/Linux
      - MacOSX/Windows
      - Linux/*
  - Code cannot be shared among different guests

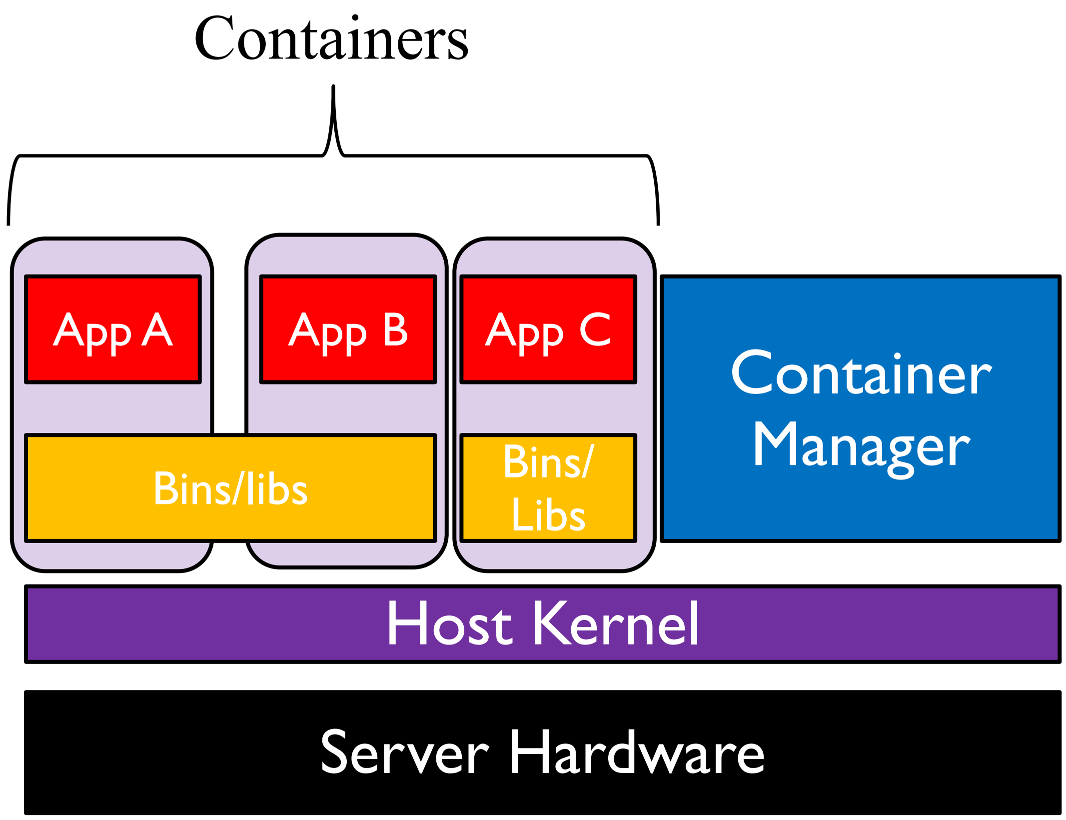
Machine vs Light virtualization



Machine vs Light virtualization

- Container-like entities (light virtualization)
  - "Simply" needs to set up the isolation environment
    - Needs support from the OS kernel
      - Namespace isolation
        - Thus communication primitives isolation
          - Including IPC mechanisms
          - Including virtualized network interfaces
      - User space isolation
        - Superuser (root in Linux) inside has nothing to do with root at the host
        - Is a special case of namespace isolation
      - Resource capping
        - CPU
        - Memory
        - …
    - Kernel code can be shared among containers
      - Other code can also be shared… at risk
  - Constraint: **guest must work on Host's kernel API**

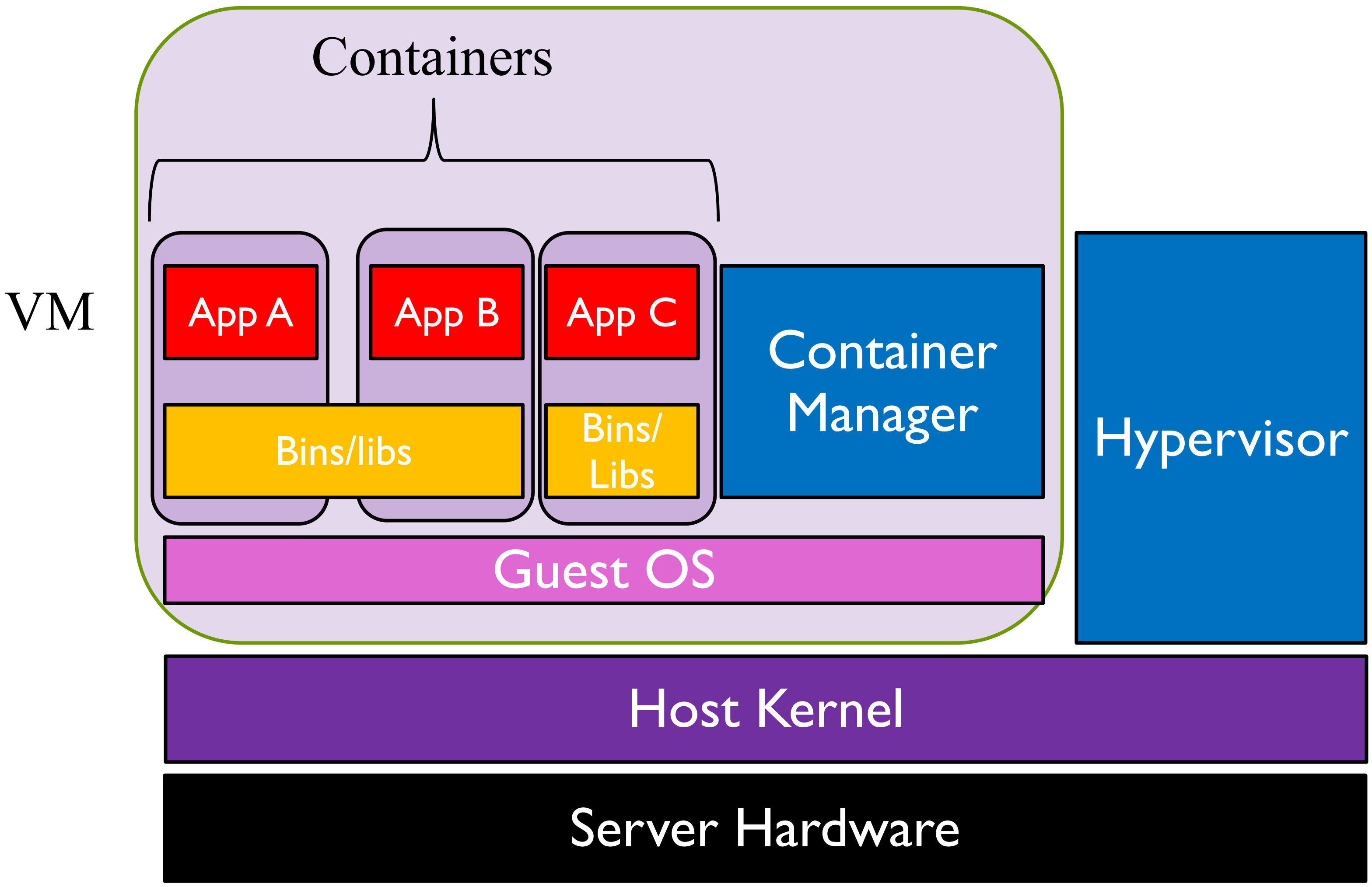
Machine vs Light virtualization



Machine vs Light virtualization

- Approaches can be mixed
  - Light virtualization inside Machine virtualization

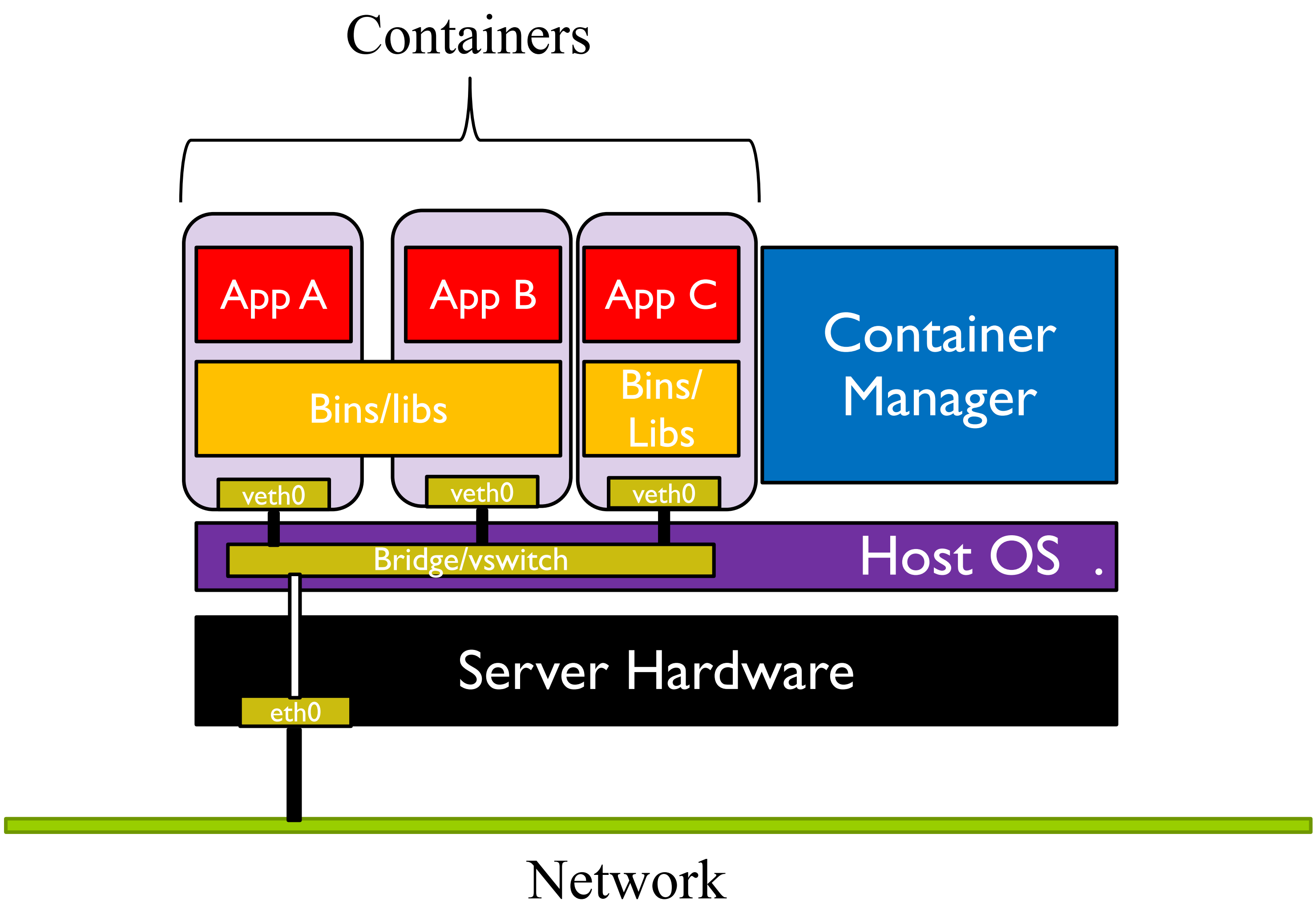
Machine vs Light virtualization



Machine vs Light virtualization

- Network virtualization
  - Similar in both cases
  - Uses the concept of a virtual switch at the host
  - The virtual switch is "connected" to virtual ethernet interfaces
    - In light virtualization, the OS kernel must support virtualized interfaces
- The virtual switch manages
  - Addressability
    - What IP networks are reachable
  - Available bandwidth: rate limiting
    - How many MB/s are avilable
  - In some fancier set-ups
    - Throtling

Machine vs Light virtualization



Containers: History

- Freeebsd Jails (~2000)
- Solaris zones (~2005)
  - Windows NT jails (unreleased) (2006)
- Linux mechanisms
  - 2006-2008
    - LXC libraries
    - Docker

Light virtualization on Linux: containers

- Supported by the kernel
  - Kernel version >= 3.6
- Based on kernel modules for
  - Namespace isolation
    - namespaces
  - Resource quota handling
    - cgroups

Linux cgroups

- cgroups == ***Control Groups***
- Kernel mechanism to limit resource consumption
  - By measuring consumption and configuring quotas
  - Applied to a group of processes
    - All bound to the same limits
    - Can be further restricted in the process tree
      - Hierarchical
      - Limits set for a group are inherited by subgroups
- Generic mechanism
  - Can be applied to any process in the system
    - Nothing special about "containers"
  - Very granular

Linux cgroups

- Provide the following functionalities
- **Resource limiting (as mentioned)**
  - A group can be configured not to exceed a specified memory limit or use more than the desired amount of processors or be limited to specific peripheral devices.
- **Prioritization:**
  - One or more groups may be configured to utilize fewer or more CPUs or disk I/O throughput.
- **Accounting:**
  - A group's resource usage is monitored and measured.
- **Control:**
  - Groups of processes can be frozen or stopped and restarted.

Linux cgroups

- Cgroup subsystems (controllers)
- Each manages a kind of resource
  - Responsible for distributing it to the processes
  - E.g. *memory* controller restricts memory usage
  - *E.g. cpuacct* controller restricts cpu usage
- Controlled through *cgroup* filesystem
  - Mounted at `/sys/fs/cgroup`
    - Controllers mounted within it
  - Each group is a subdir of a controller mount point

Linux cgroups: example memory

- Create a cgroup *example*
  - */sys/fs/cgroup/memory/example*
  - To limit it, we must add a file
    - $ echo 50000000 | sudo tee /sys/fs/cgroup/memory/foo/memory.limit_in_bytes
- Consider the app:
  - $ cat test.sh
    - #!/bin/sh
    - while [1]; do echo "hello world" sleep 60 done
  - Run it in the background
    - $ sh ~/test.sh &
  - Find its PID (e.g. 1234) and place it within the cgroup
    - Echo 1234 > /sys/fs/cgroup/memory/example/cgroup.procs
  - Try:
    - $ ps –o cgroup 1448

Linux namespaces

- Mechanism to limit visibility
  - Applied to a group of processes
- Types of resources covered:
  - Process trees (their ids)
  - Network interfaces
  - User Ids/Group Ids
  - File System mount points
    - Includes Unix Sockets
    - Devices
- Generic mechanism
  - Can be applied to any process in a system
  - Granular

Linux namespaces

- lsns
  - Command lists the set of available namespaces on a system
  - `4026531835 cgroup      5   938 ubuntu /lib/systemd/systemd --user`
  - `4026531836 pid         5   938 ubuntu /lib/systemd/systemd --user`
  - `4026531837 user        5   938 ubuntu /lib/systemd/systemd --user`
  - `4026531838 uts         5   938 ubuntu /lib/systemd/systemd --user`
  - `4026531839 ipc         5   938 ubuntu /lib/systemd/systemd --user`
  - `4026531840 mnt         5   938 ubuntu /lib/systemd/systemd --user`
  - `4026531992 net         5   938 ubuntu /lib/systemd/systemd --user`
- Each process has a PID within a PID namespace
  - Processes in different PID namespaces cannot interact with each other by PID
    - They make no sense across PID namespaces
    - Similar to the namespace of file descriptors
      - Each process has its own, and cannot transfer them to another process

Linux namespaces

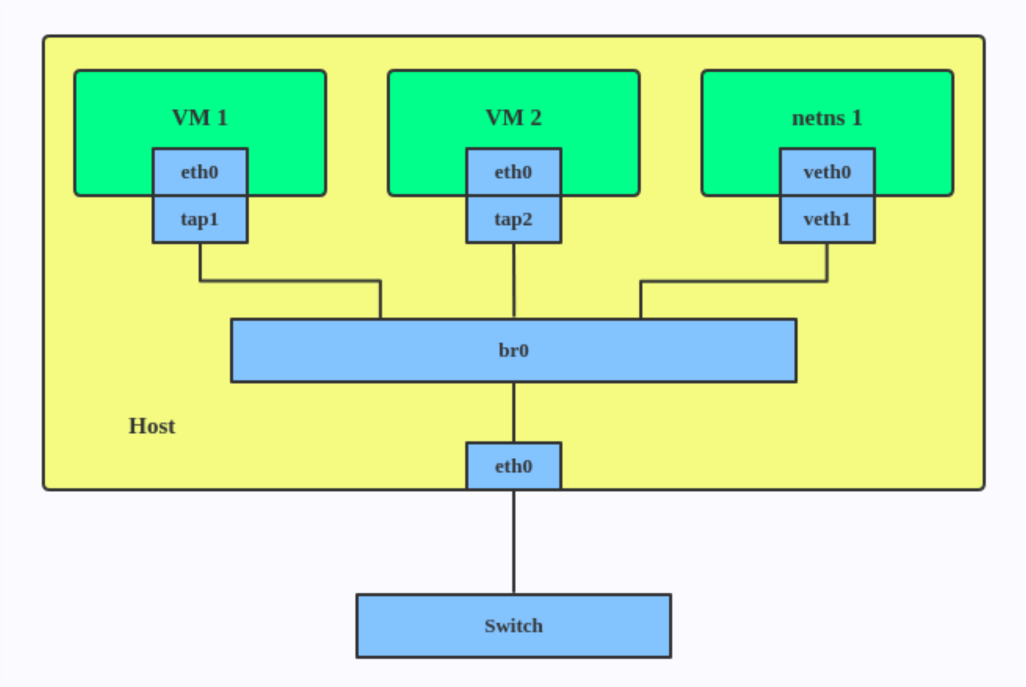
- The `unshare` command
  - It runs a subprocess with the given executable in their own namesapce
    - It indicates the namespaces that ran process MUST NOT share with the rest of the system
- Example
  - `sudo unshare --fork --pid --mount-proc zsh`
  - Runs a zsh subprocess, and offers a shell prompt
    - Any process launched from that prompt will be within the newly created namespace for zsh
    - Try: pidof zsh
      - You get 1
    - On another terminal, **pidof zsh**, the result is different
  - Many options available
    - E.g., mapping root /proc, …

Virtual network interfaces

- Special virtualized network devices
- Interact with network namespace
- Many kinds in the Linux kernel (use `ip link help`)
  - Bridge
  - Bonded interface
  - Team device
  - VLAN (Virtual LAN)
  - VXLAN (Virtual eXtensible Local Area Network)
  - MACVLAN
  - IPVLAN
  - MACVTAP/IPVTAP
  - MACsec (Media Access Control Security)
  - VETH (Virtual Ethernet)
  - VCAN (Virtual CAN)
  - VXCAN (Virtual CAN tunnel)
  - IPOIB (IP-over-InfiniBand)
  - NLMON (NetLink MONitor)
  - Dummy interface
  - IFB (Intermediate Functional Block)
  - netdevsim

Network virtualization: Bridge

- Behaves like a network Switch
- Forwards packets among connected interfaces
- Use a bridge when you want to establish communication channels between VMs, containers, and your hosts.

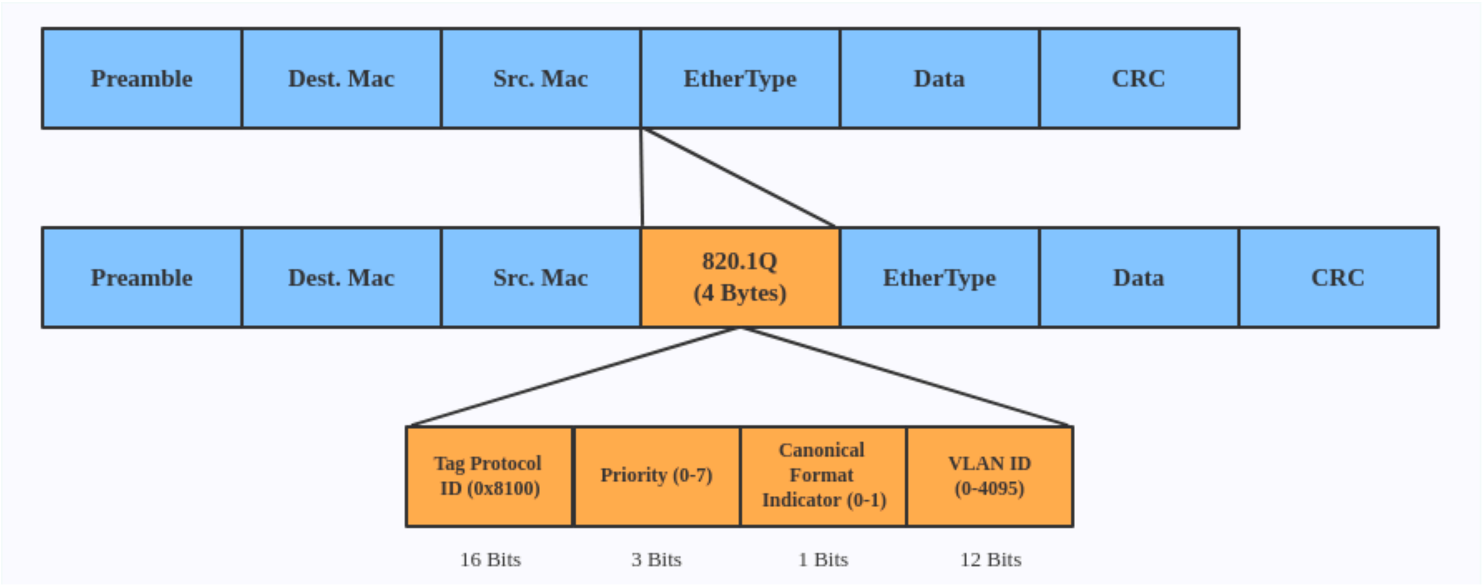


Network virtualization: Bridge

```
# ip link add br0 type bridge
# ip link set eth0 master br0
# ip link set tap1 master br0
# ip link set tap2 master br0
# ip link set veth1 master br0
```

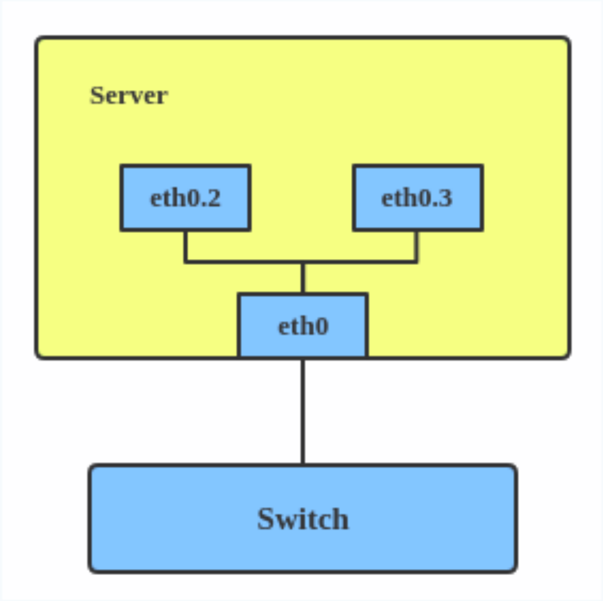
Network virtualization: VLAN

- "Virtual LAN"
- Separates L2 (e.g. ethernet) domains by adding tags to network packets
- Use same physical hardware to form multiple L2 domains
- Look as different ethernet interfaces



Network virtualization: VLAN

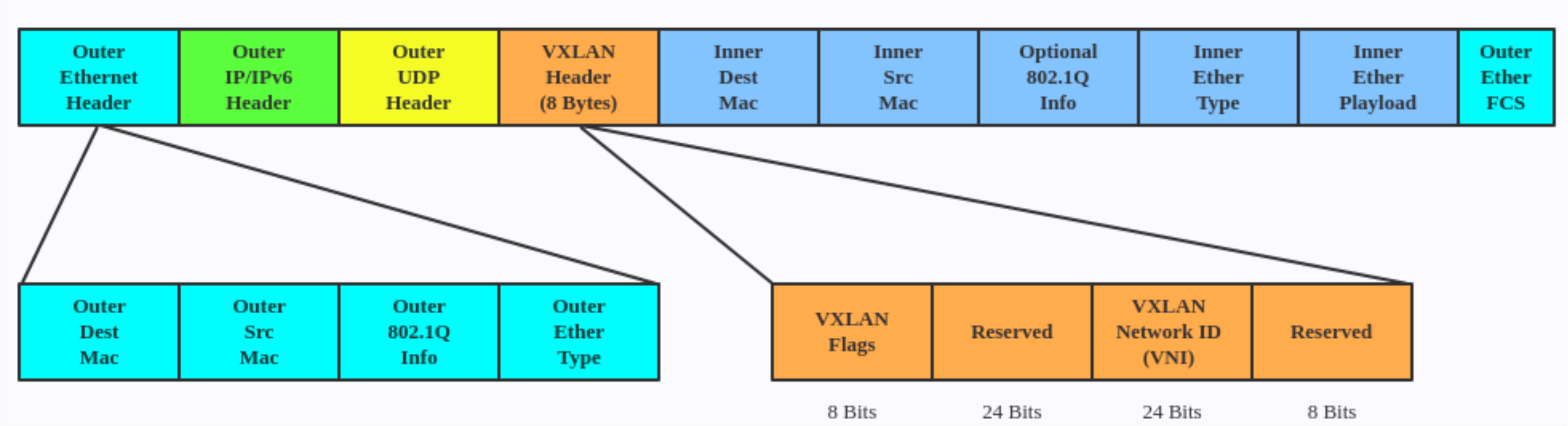
```
# ip link add link eth0 name eth0.2 type vlan id 2
# ip link add link eth0 name eth0.3 type vlan id 3
```



- Look as different ethernet interfaces from within

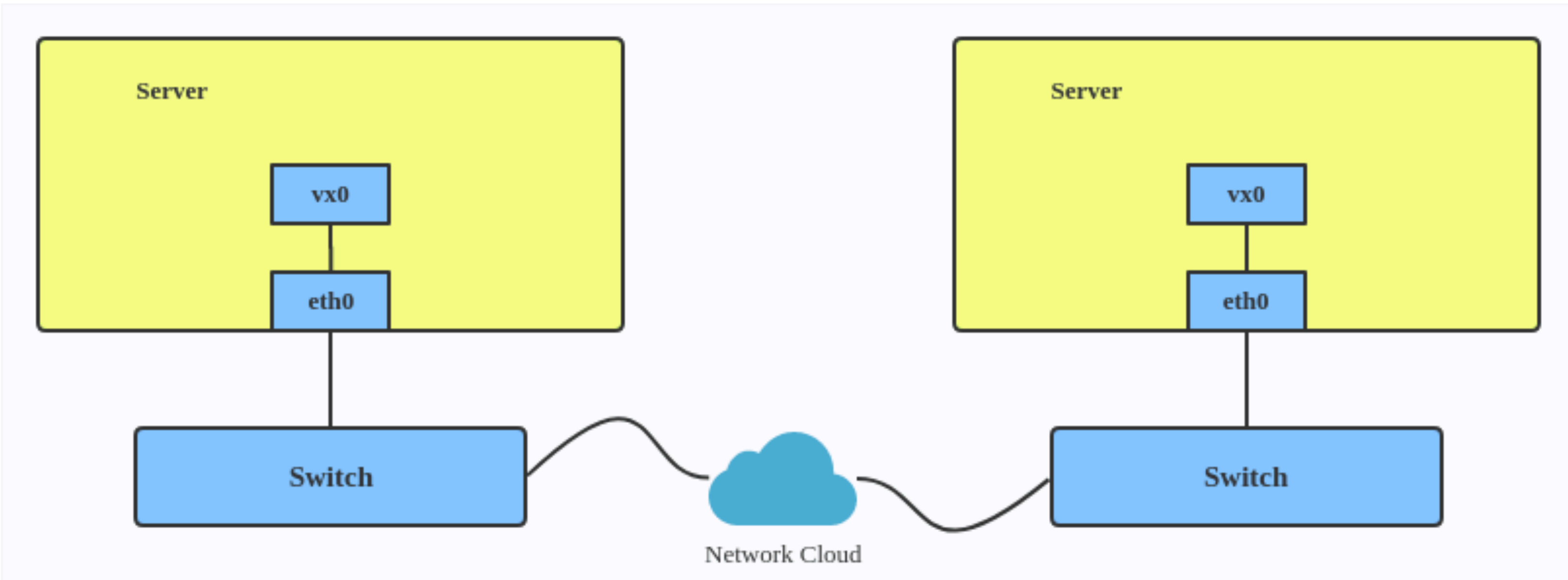
Network virtualization: VXLAN

- "Virtual eXtensible LAN"
- **TUNNELING** protocol
  - Solves limited VLAN ids
  - VXLAN ID (segment ID) is 24 bits long ➔ 16,777,216 virtual LANs
    - 4,096 times the VLAN capacity.
- Encapsulates Layer 2 frames with a VXLAN header into a UDP-IP packet



Network virtualization: VXLAN

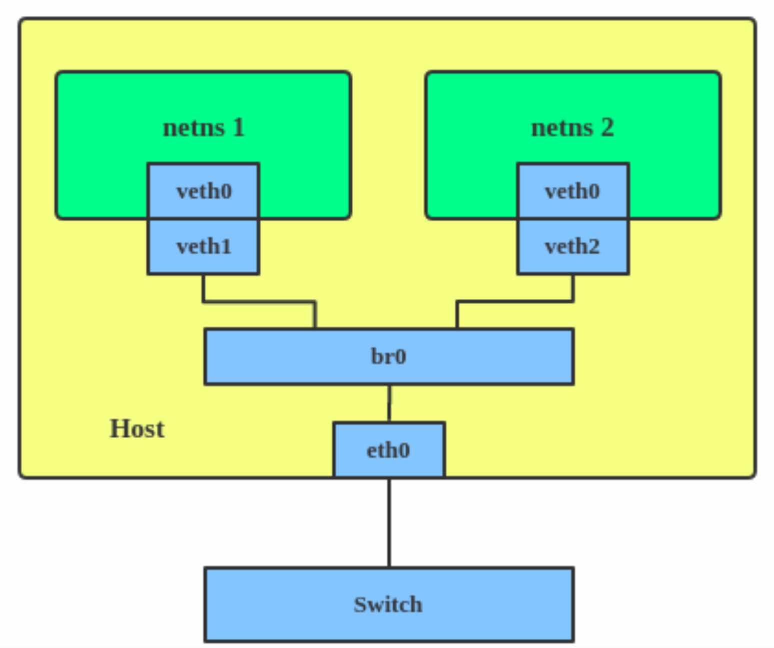
\# ip link add vx0 type vxlan id 100 local 1.1.1.1 remote 2.2.2.2 dev eth0 dstport 4789



- Typically deployed in data centers on virtualized hosts
  - May be spread across multiple racks.

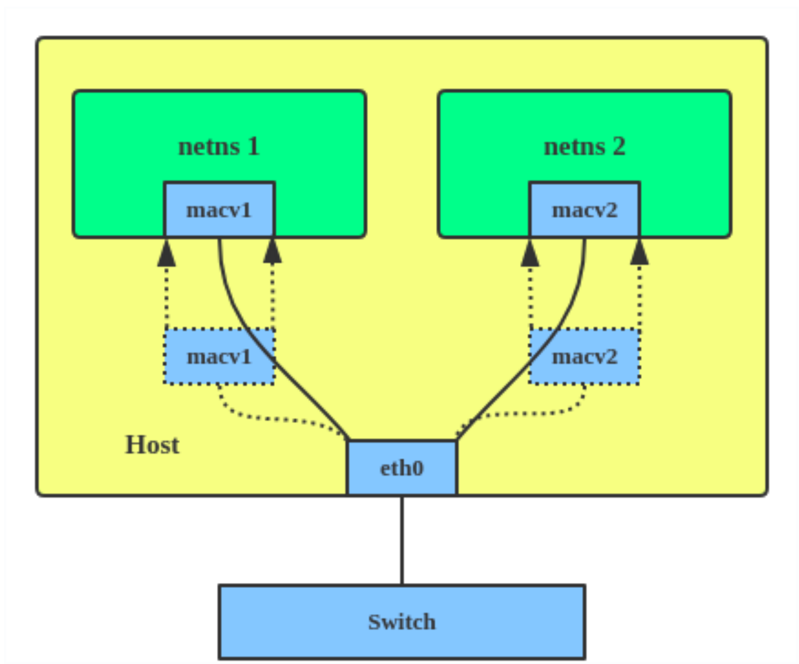
Network virtualization: MACVLAN

- Multiple interfaces on top of a physical one
  - Each has a different MAC address
- Without MACVLAN you need bridges to access the network
  - Plus, other virtualized interfaces (e.g. Veth)



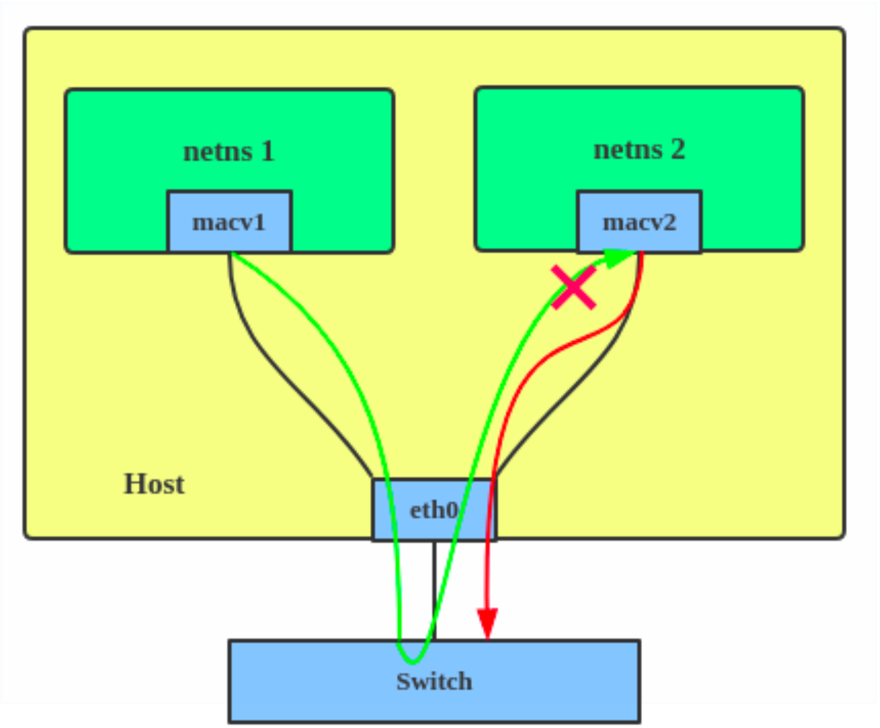
Network virtualization: MACVLAN

- With MACVLAN
- Directly attach the MACVLAN interface to the namespace wher it should be used



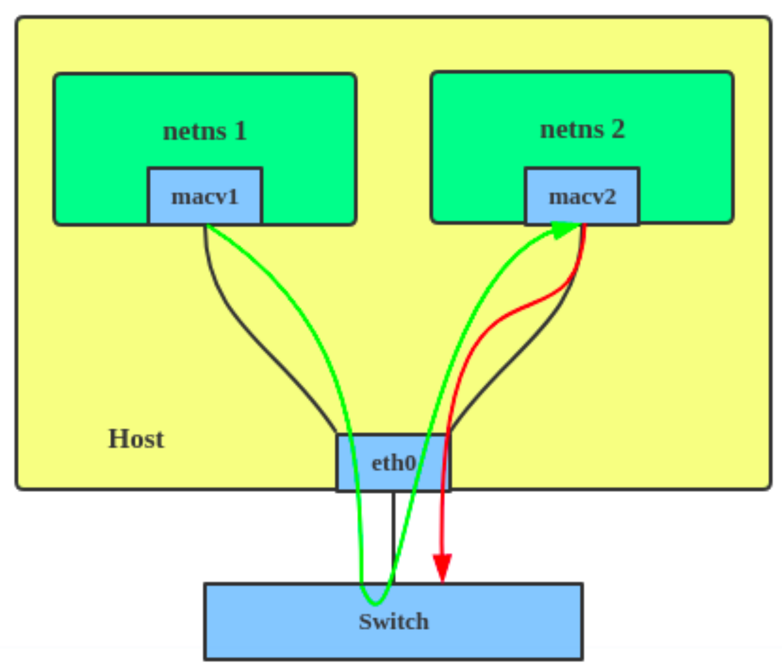
Network virtualization: Types of MACVLAN

- Private
  - Forbids communication among interfaces on same physical interface



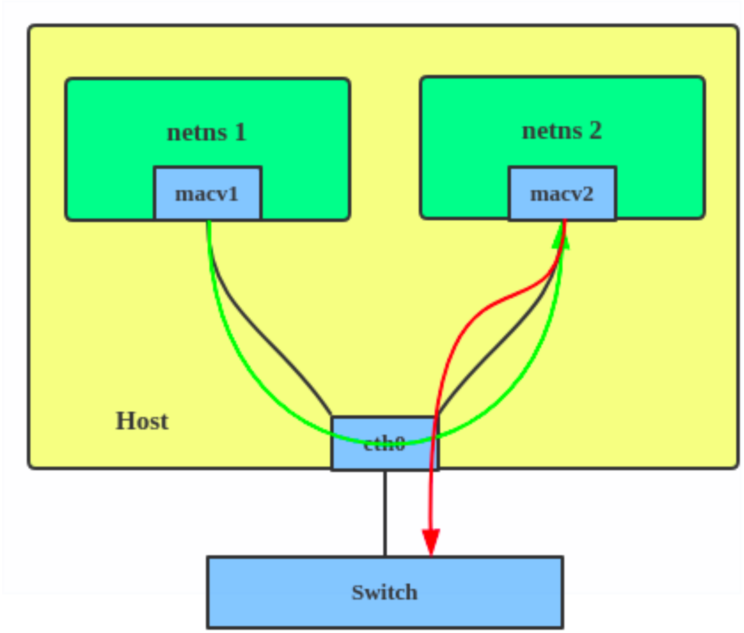
Network virtualization: Types of MACVLAN

- VEPA
  - Communication through the external switch
    - Must support *hairpin* mode
    - Or… There is a router that redirects packets back



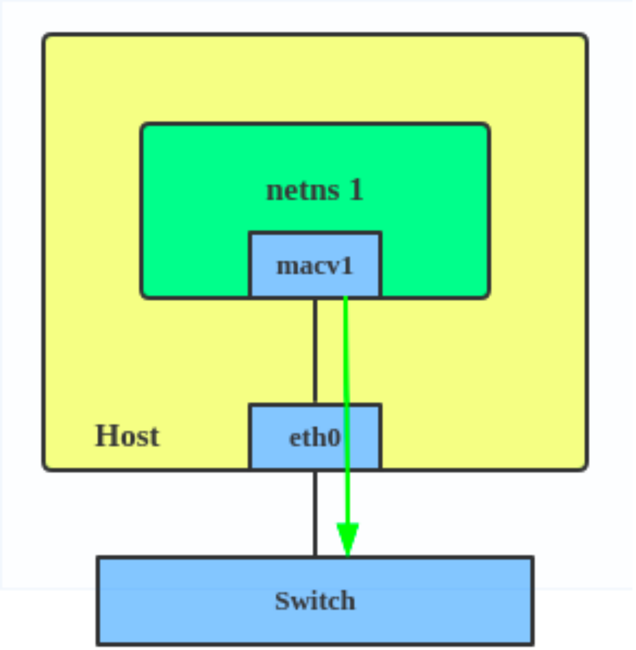
Network virtualization: Types of MACVLAN

- BRIDGE
  - Phys interface behaves as a simple bridge also for internal macvlans
  - Packets do not have to travel outside



Network virtualization: Types of MACVLAN

- PassThrough
  - Allows a single VM/namespace to be connected directly to the physical interface



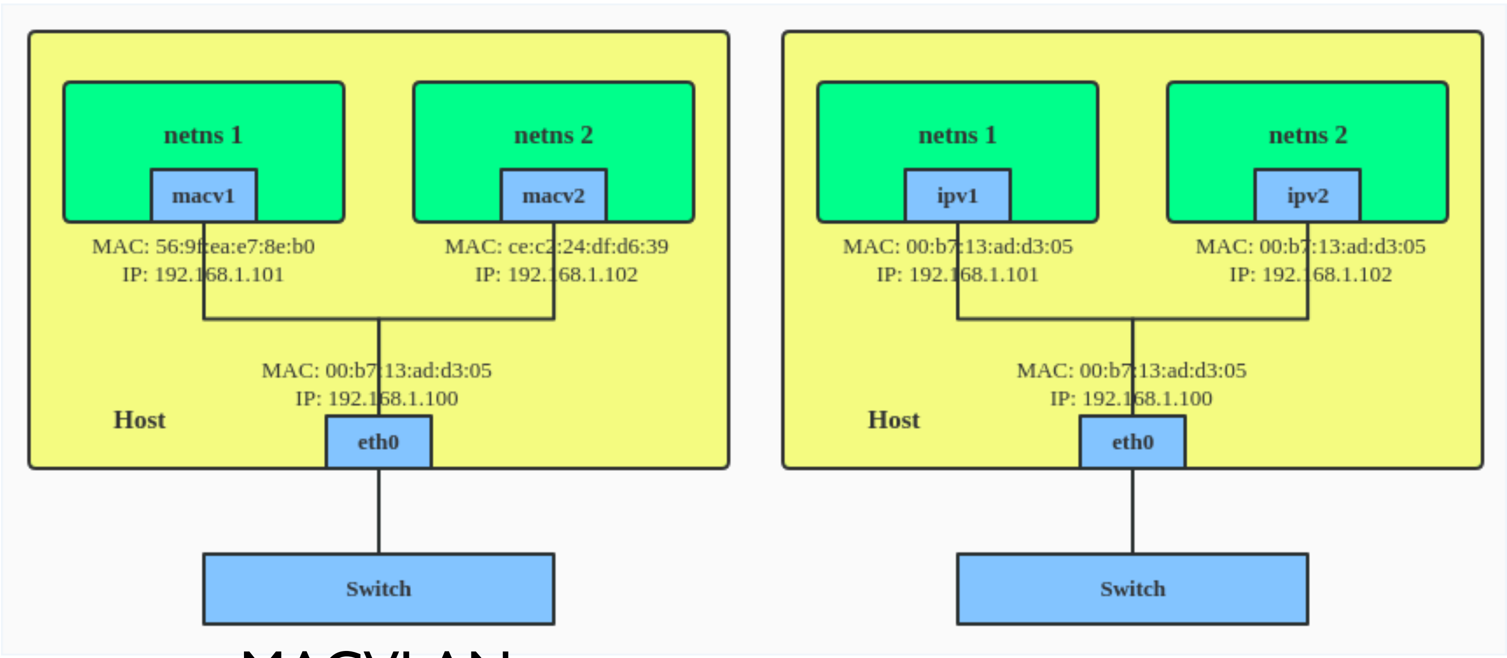
Network virtualization: Summary MACVLAN

- Use MACVLAN when connecting directly to a physical network from containers
- BRIDGE is the most common mode

```
# ip link add macvlan1 link eth0 type macvlan mode bridge
# ip link add macvlan2 link eth0 type macvlan mode bridge
# ip netns add net1
# ip netns add net2
# ip link set macvlan1 netns net1
# ip link set macvlan2 netns net2
```

Network virtualization: IPVLAN

- Similar to MACVLAN
  - But Interfaces share MAC address

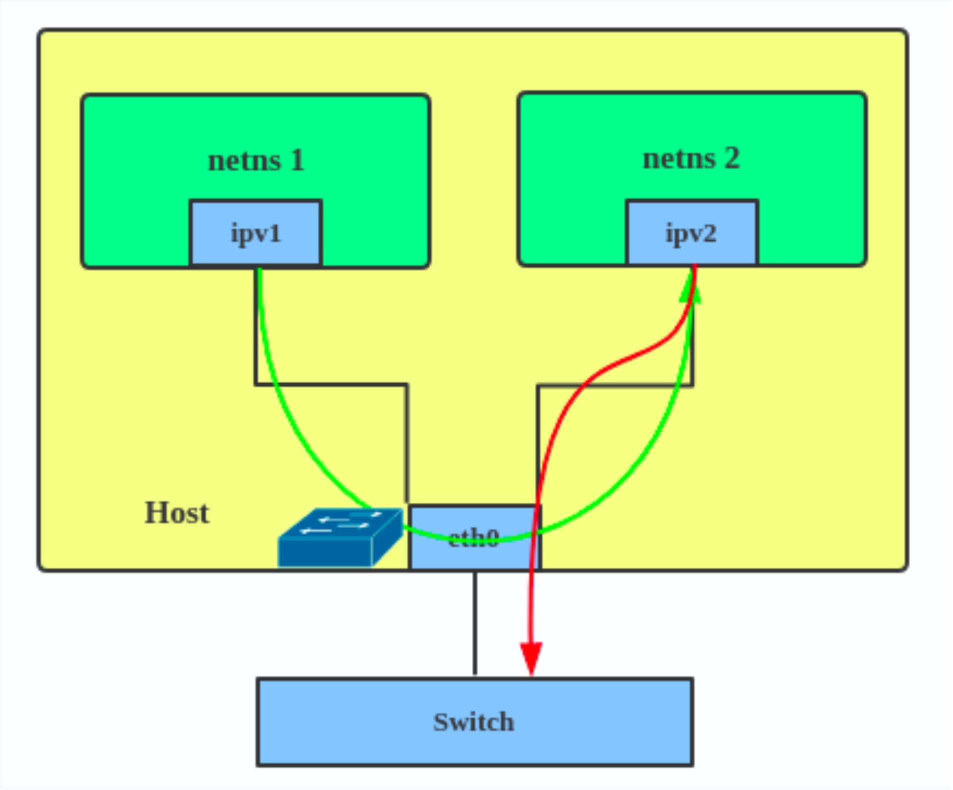


MACVLAN
Different MAC addresses

IPVLAN
Same MAC addresses

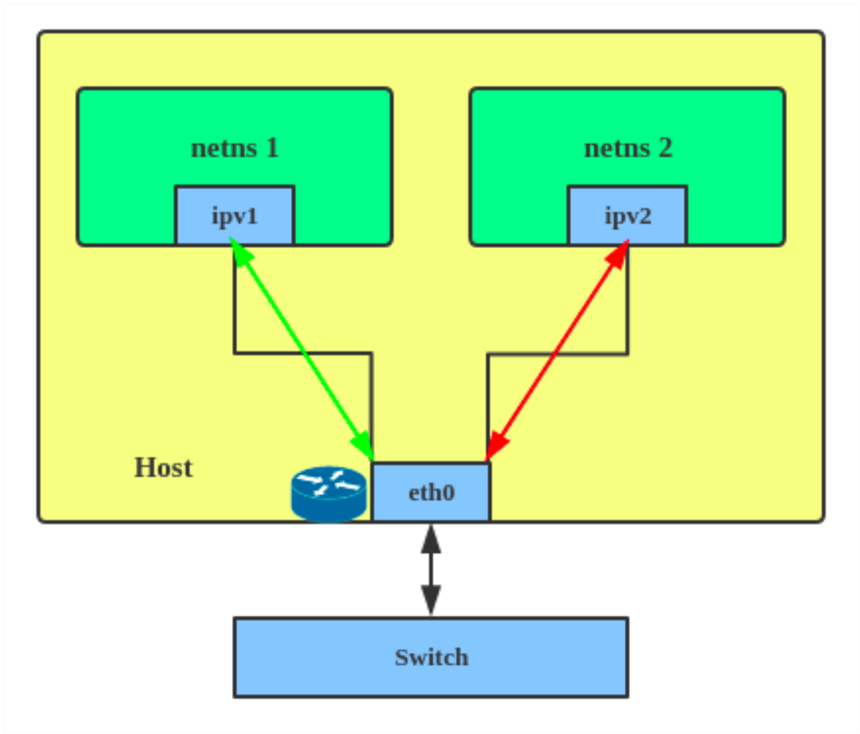
Network virtualization: IPVLAN

- Two modes: Mode L2 or Mode L3
- In mode L2, it behaves like MACVLAN BRIDGE
  - Parent interface behaves like a simple bridge



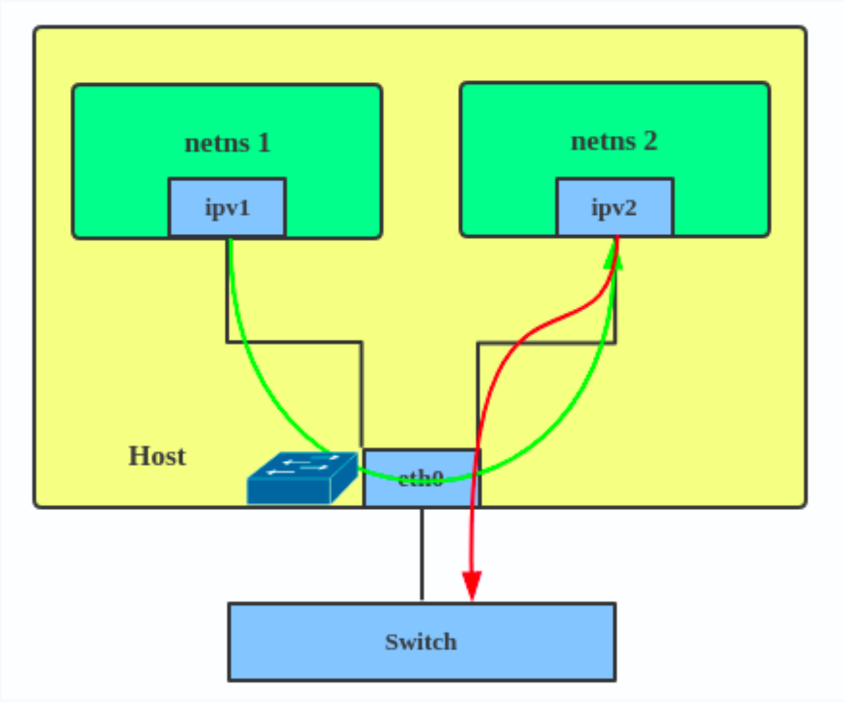
Network virtualization: IPVLAN

- In mode L3,
  - Parent interface behaves like a router
  - Potentially better scalability



Network virtualization: IPVLAN

```
# ip netns add netns1
# ip link add ipv1 link eth0 type ipvlan mode l2
# ip link set dev ipv1 netns netns1
# ip netns add netns2
# ip link add ipv2 link eth0 type ipvlan mode l2
# ip link set dev ipv2 netns netns2
```



Network virtualization: MACVLAN or IPVLAN

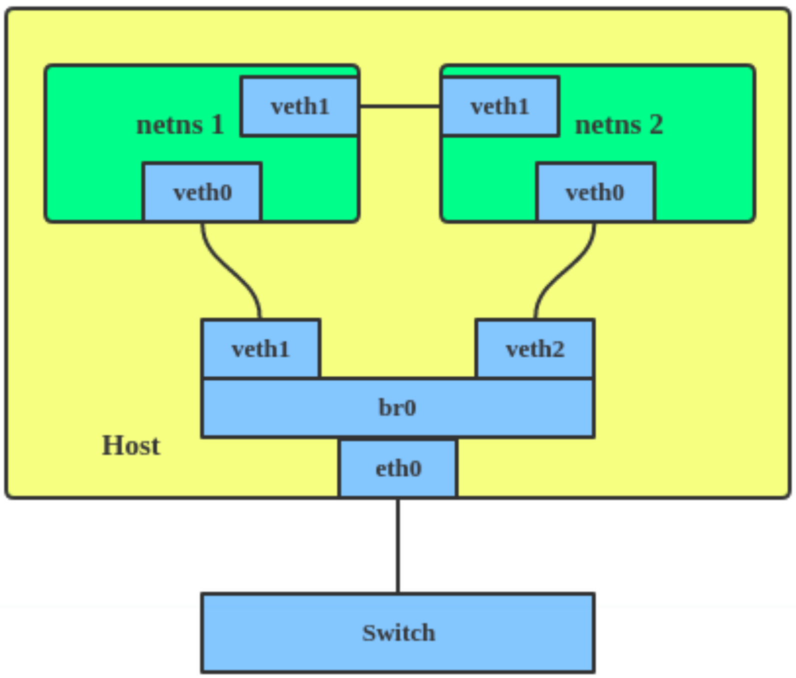
- When to select IPVLAN over MACVLAN:
  - The host connected to the external switch/router has policy configured that allows only one mac per port.
  - Number of virtual devices created on a master exceed the MAC capacity
    - Forces NIC in promiscuous mode
      - Impact on performance
  - The slave device is to be put into an untrusted namespace
    - L2 on the slave could be misused.

Network virtualization: VETH

- Virtual ETHernet: local ethernet tunnel
  - No tunnel traffic exits the host
- Created as pairs
  - Traffic entering one device immediately exits the other device
  - Like a cable
- When either device is down, the whole link is down
- Use VETH when network namespaces need to communicate
  - With the host
  - Among themselves

Network virtualization: VETH

```
# ip netns add netns1
# ip netns add netns2
# ip link add veth1 netns netns1 type veth peer name veth1
netns netns2
# ip link add veth0 netns netns1 type veth peer name veth1
# ip link add veth0 netns netns2 type veth peer name veth2
```



Network virtualization: DUMMY

- Fully virtual
  - E.g. The loopback interface is also fully virtual
- For routing packets through
  - With no transmission
- Used for testing/debugging

```
# ip link add dummy1 type dummy
# ip addr add 192.168.1.1/24 dev dummy1
# ip link set dummy1 up
```

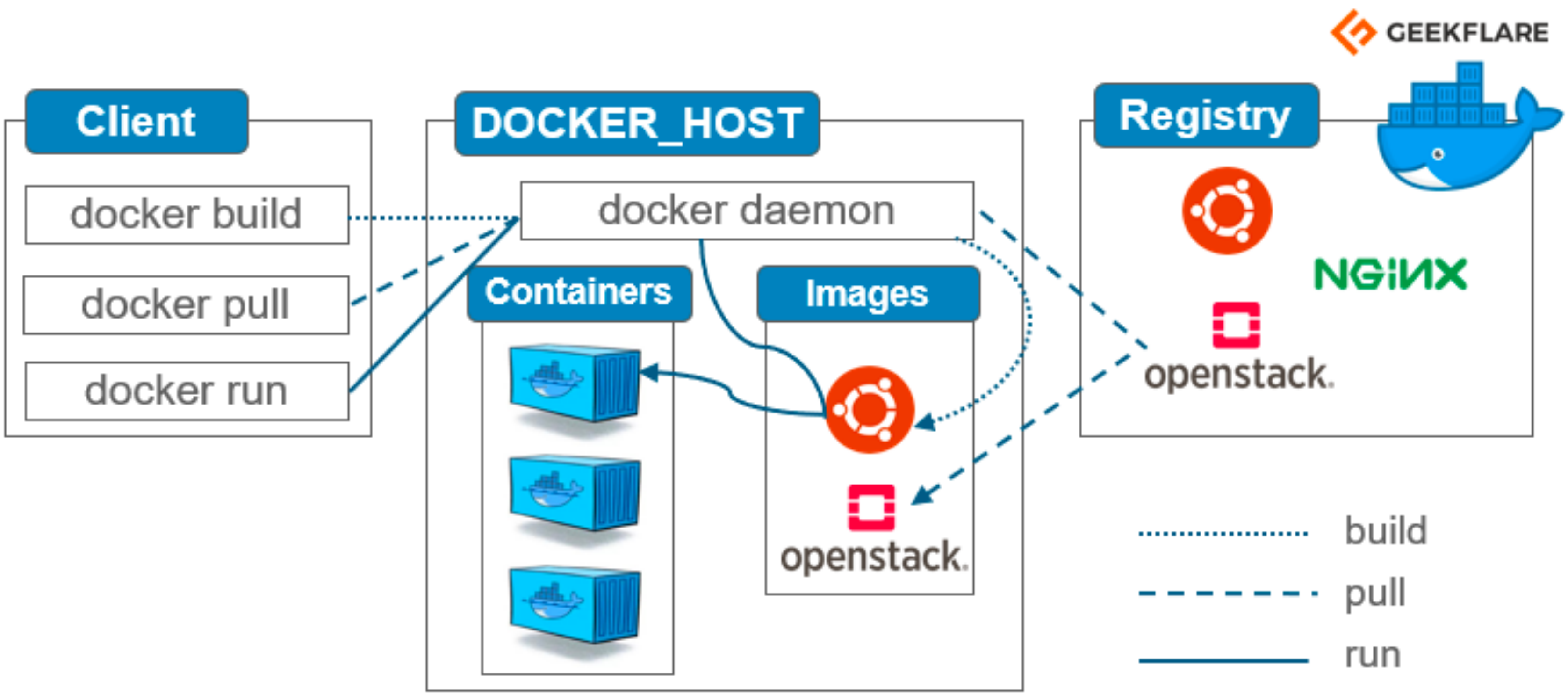
Linux Containers

- Group of processes subjected to
  - Full set of cgroup policies
    - All resources are contemplated
  - Full set of namespace policies
    - All namespaces are contemplated
- Full cgroup/namespace policies isolate a container
  - Specifically from other containers
    - Two-way isolation
  - One-way isolation of the host
    - Containers cannot access the host but host can access containers
      - Host is privileged wrt kernel: full access to kernel API
- Various isolation relaxations
  - E.g. Share named resources
  - E.g. non-quotaed resources

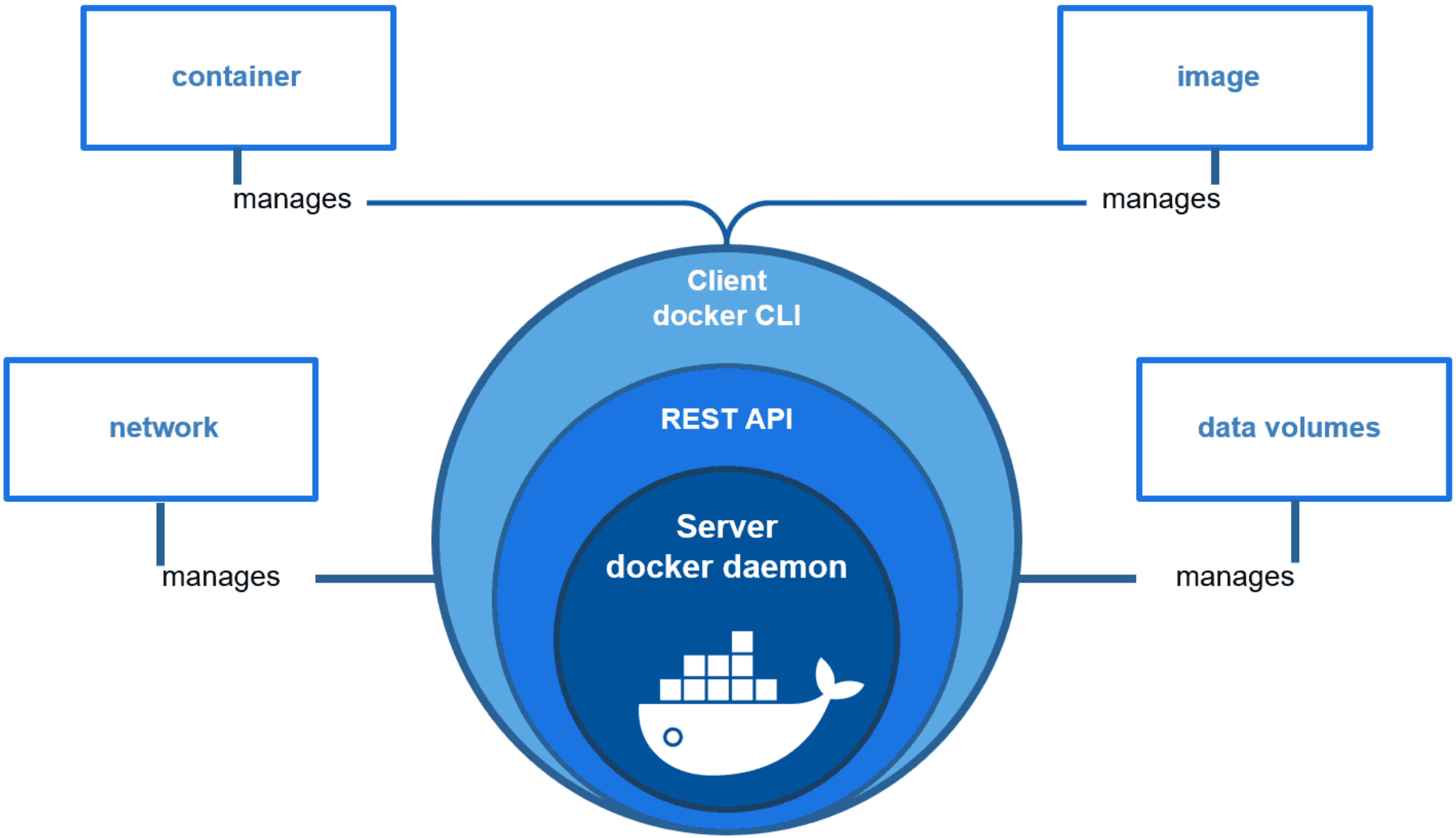
Docker

- A container management system
- Flexible architecture
  - Daemon (dockerd)
  - CLI command (docker)
  - Various plug-ins
    - Storage
    - Networking
  - An image standard
    - Based on a layered file system
    - Leveraged to port software stacks among hosts
    - Caches images on local host

Docker High level architecture



Docker Conceptual structure



Docker warm up exercises

- Go to

https://github.com/eficode-academy/docker-katas/tree/master/labs