ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Programming Software Network Stacks

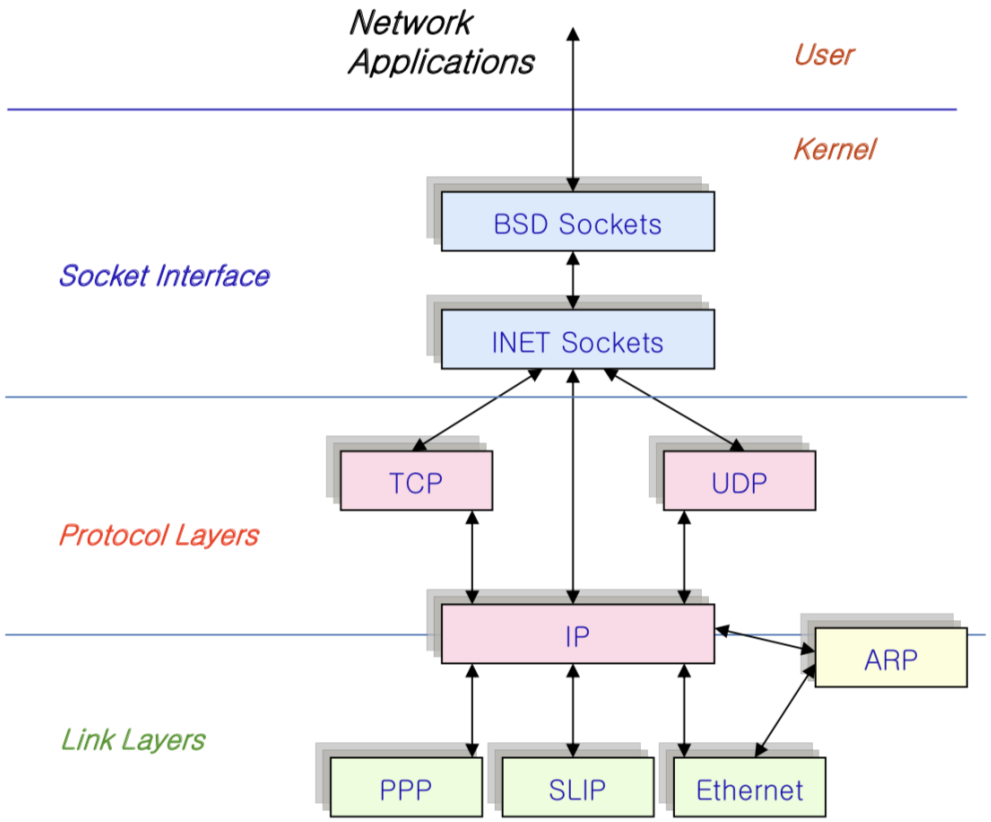
Khanh Nam Chu

ONE LOVE. ONE FUTURE.

CONTENTS

Programming Software Network Stacks

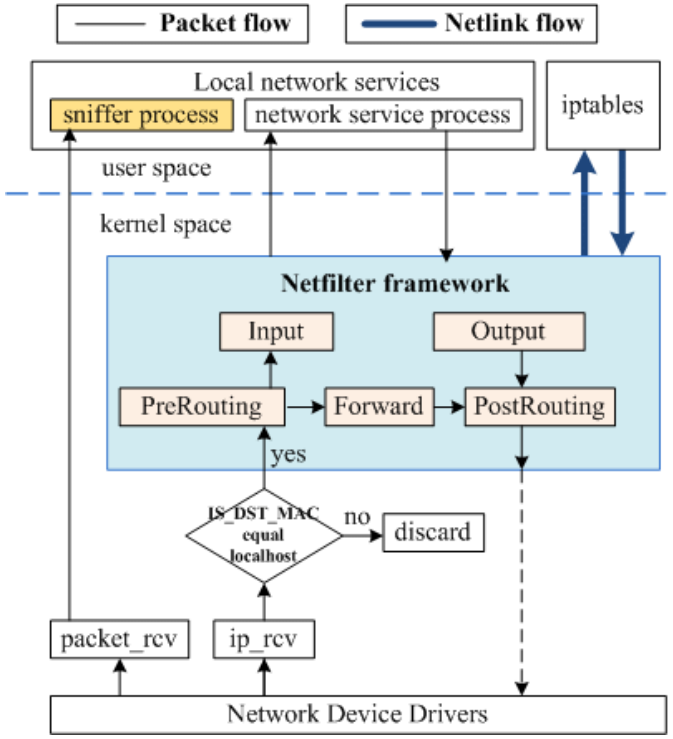
Programming software network stacks means writing or modifying the code that implements networking protocols in software—the part of the operating system (or a user-space library) that processes network packets according to rules like TCP, UDP, and IP.



1. Kernel Packet Processing

- Kernel packet processing is the set of steps the Linux kernel follows to receive, interpret, route, and deliver network packets (and also to send them) using the network stack.

- It’s basically what happens to a packet after the NIC hardware hands it to the kernel, until it reaches your application or is sent back out.



1. Kernel Packet Processing

Receive path (incoming packet → user application)

- Frame arrives at NIC (hardware): The NIC converts the physical signal to bytes and puts the frame into a DMA buffer in memory (Rx ring).

- Driver notices the completed Rx descriptor: Either via an interrupt or inside a poll function (NAPI). With NAPI, the driver disables per-packet IRQs and schedules a poll to avoid IRQ storms.

- Driver builds an sk_buff (skb): The driver wraps the DMA buffer with a kernel packet container (struct sk_buff), sets skn->dev (net_device), skb->data pointer, skb->len, and calls eth_type_trans() which parses and removes the Ethernet header and sets skb->protocol and header offsets.

1. Kernel Packet Processing

Receive path (incoming packet → user application)

- Checksum/ offload handling: If hardware provided checksum offload, the driver marks the skb so higher layers skip software checksum; otherwise, kernel will verify checksums.

- Submit the packet to the stack (NAPI / netif): The driver either calls a GRO-aware entry, which enqueue/process the skb in the RX softirq context. This hands the packet from driver context into kernel network processing.

- NET_RX_SOFTIRQ runs: Packet processing continues in softirq context (prevents long IRQ handlers). The softirq work dequeues packets and invokes the next layers.

- Link → Network layer transition & Netfilter PRE_ROUTING.

1. Kernel Packet Processing

Receive path (incoming packet → user application)

- Routing decision (is the packet for local delivery?): Kernel checks destination IP: if not local, it may forward or drop. If local, it proceeds to IP input.

- IP layer processing: IP header is validated (version, checksum if needed), fragments are reassembled if necessary, and the packet may traverse more Netfilter hooks (INPUT/LOCAL_IN) before being handed to transport.

- Transport layer (TCP / UDP / ICMP).

- Socket lookup and enqueue.

- User space reads the data.

1. Kernel Packet Processing

Transmit path (application → wire)

- Application writes to socket (send()/ write()).

- Transport layer forms segment.

- Routing & neighbor resolution: Kernel does a routing lookup (finds dst), ARP/ND resolution via the neighbor subsystem to obtain L2 (MAC) address. If answer not present, packet may be queued until resolution completes (or ARP request is initiated).

- Netfilter POSTROUTING / egress hooks: Packet may traverse NAT/POSTROUTING hooks, QoS (tc) qdiscs and filters are applied.

1. Kernel Packet Processing

Transmit path (application → wire)

- Link-layer header creation: Ethernet header (destination/source MAC, EtherType) is prepended; VLAN tags if necessary.

- Driver ndo_start_xmit() / dev_queue_xmit() : Kernel hands the skb to the netdevice transmit queue. The driver maps the skb to DMA descriptors, possibly using TSO/CSO (TCP segmentation offload / checksum offload) to let NIC do segmentation/checksum.

- NIC transmits the frame: Hardware reads descriptors, pulls data by DMA, and puts bits on the wire. TX completions are posted back (interrupts or polled) so driver can free buffers.

1. Kernel Packet Processing

How to make Kernel more programmable?

Solution #1:

- Identify which parts of the stack need to change more frequently.
- Separate out those parts of the code as a standalone "modules".
- Define interfaces for these modules to interact with the rest of the stack/kernel.

1. Kernel Packet Processing

How to make Kernel more programmable?

Solution #2:

- eBPF (extended Berkeley Packet Filter).

- Allows you to run your user-space programs in a "sandbox" in certain locations in the kernel.

- So, you can safely and efficiently extend the capabilities of the kernel without having to change the kernel.

2. User-Space Packet Processing

- User space packet processing means handling network packets outside the kernel, in a normal application's memory space, instead of letting the kernel's built-in network stack process them.

- NIC → kernel (driver + network stack) → application.

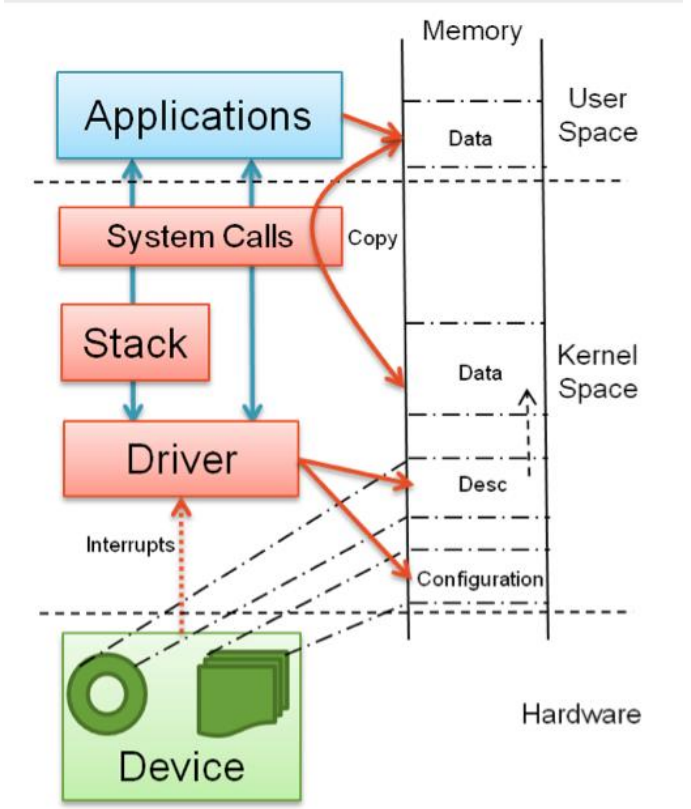


Figure 1: Kernel space network driver

2. User-Space Packet Processing

- User space packet processing means handling network packets outside the kernel, in a normal application's memory space, instead of letting the kernel's built-in network stack process them.

- NIC → kernel (driver + network stack) → application.

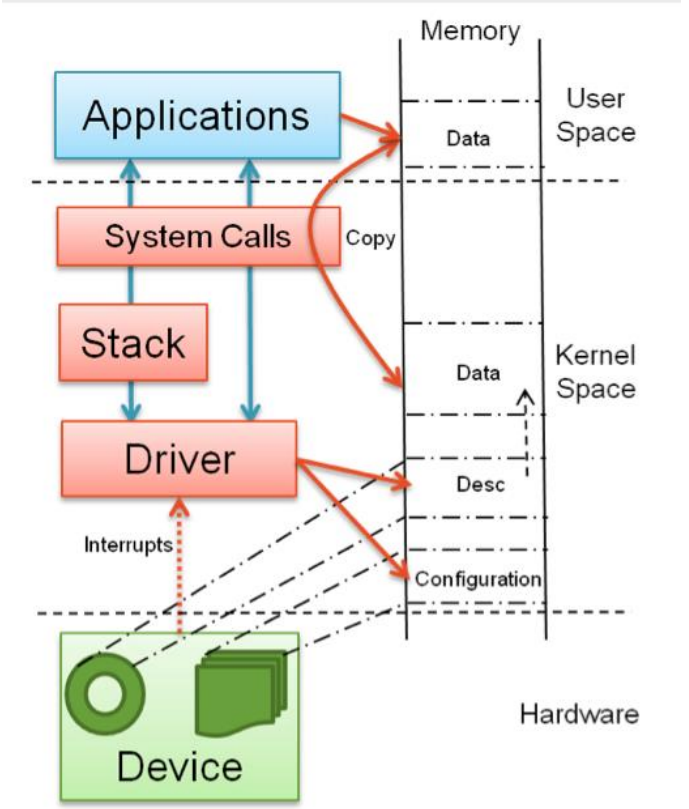


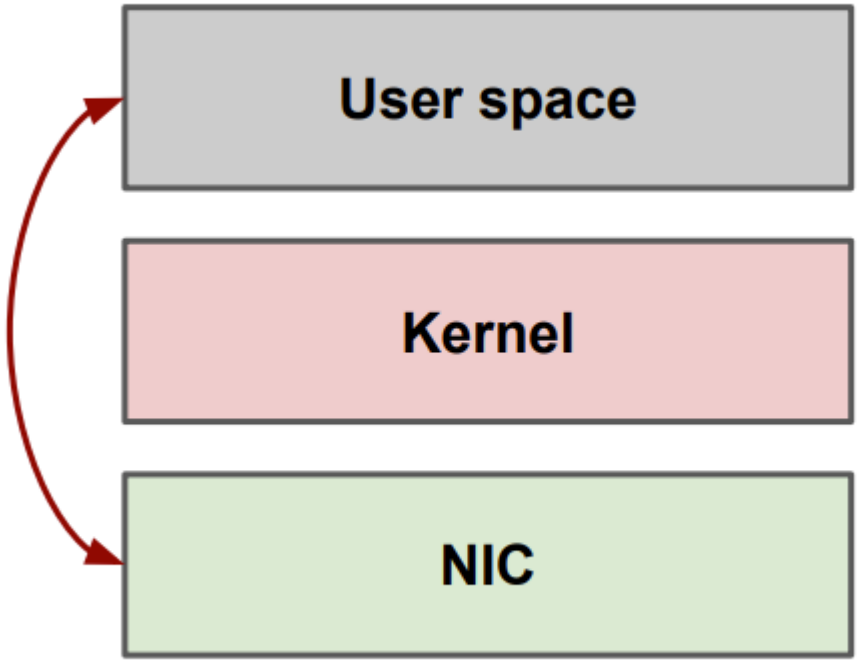
Figure 1: Kernel space network driver

2. User-Space Packet Processing

- The idea is to write a program in user space to process packets.
- Packets go directly from NIC to User-Space and vice versa.

→ **Kernel Bypass**

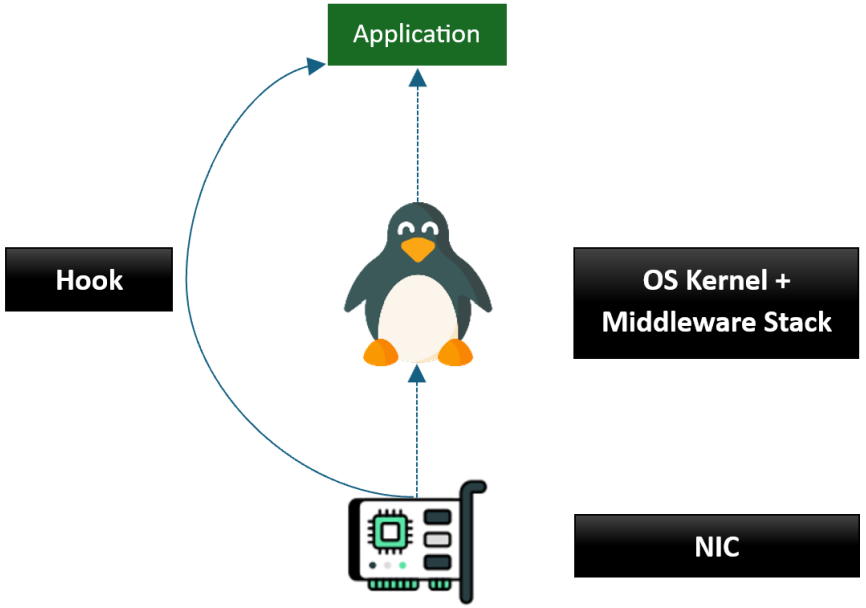
- Example frameworks: DPDK, Netmap.



2. User-Space Packet Processing

Kernel Bypass Pros:

- Fully customizable.
- High performance.
- Easier software to develop .
- Provides an opportunity to rethink how we design the network stack.



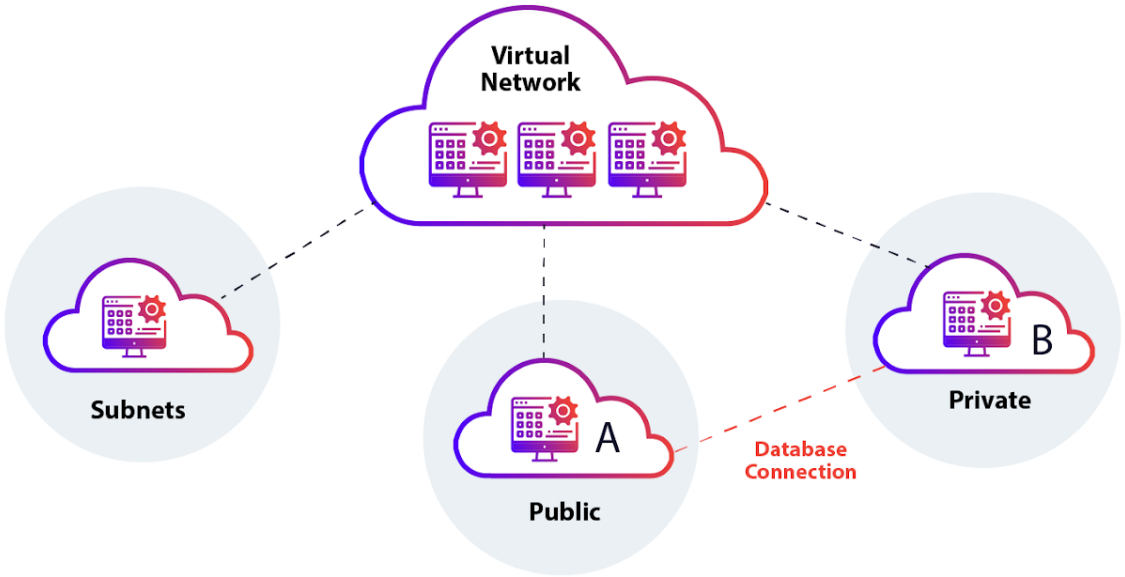
2. User-Space Packet Processing

Kernel Bypass Cons:

- Can't take advantage of the Kernel benefits, such as resource management, security, etc.
- Busy polling to get packets locks up CPU resources.

3. Network Processing in Virtualized Platforms

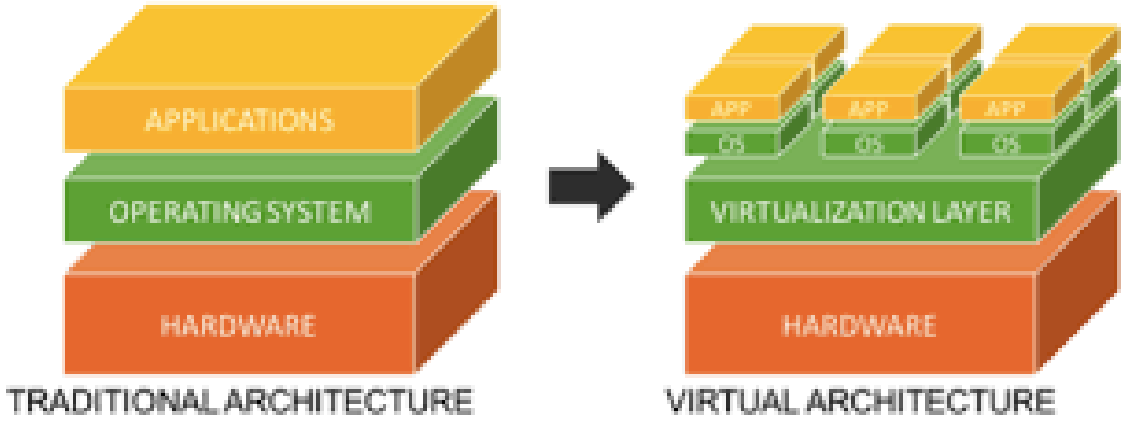
- Network processing in virtualized platforms is how network traffic is handled when your workloads are running inside virtual machines (VMs) or containers instead of directly on bare metal.



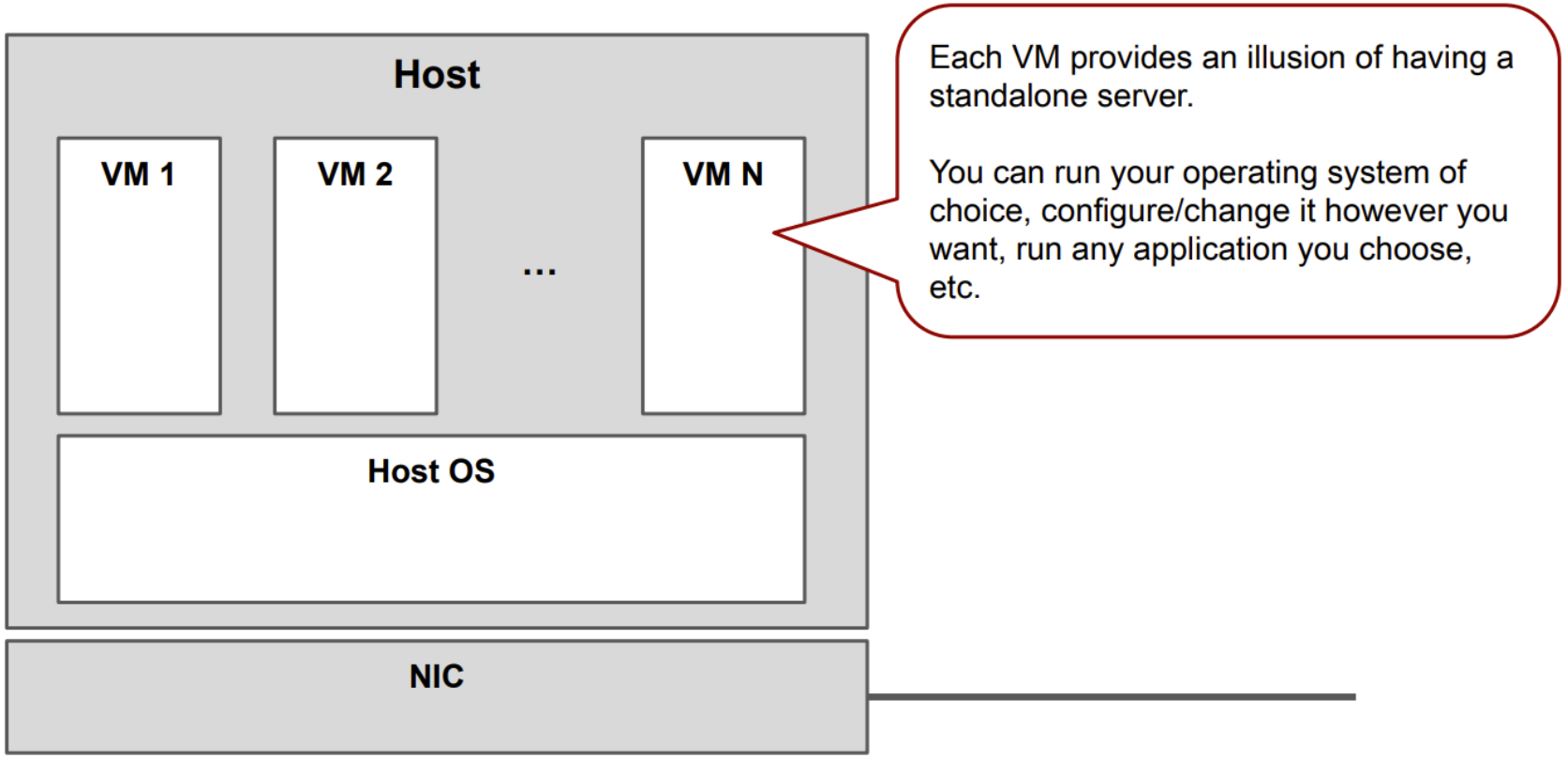
3. Network Processing in Virtualized Platforms

How it works:

- Physical NIC (pNIC) receives a packet.
- Hypervisor (e.g., KVM, VMware ESXi) intercepts it.
- Virtual switch (like Open vSwitch, Linux bridge, or VMware vSwitch) decides which VM or container it belongs to.
- Packet goes to the vNIC (virtual NIC) of the target VM.
- Guest OS kernel processes the packet via its own network stack.
- Application in the VM gets the data.



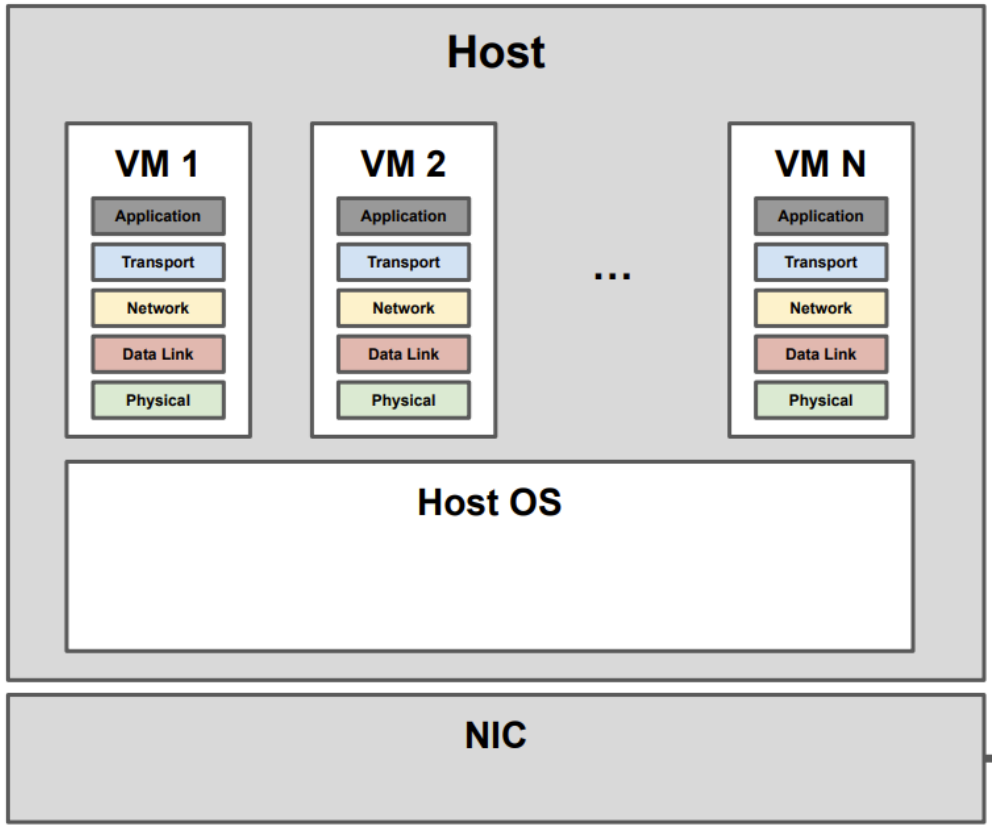
Server Virtualization



3. Network Processing in Virtualized Platforms

Server Virtualization

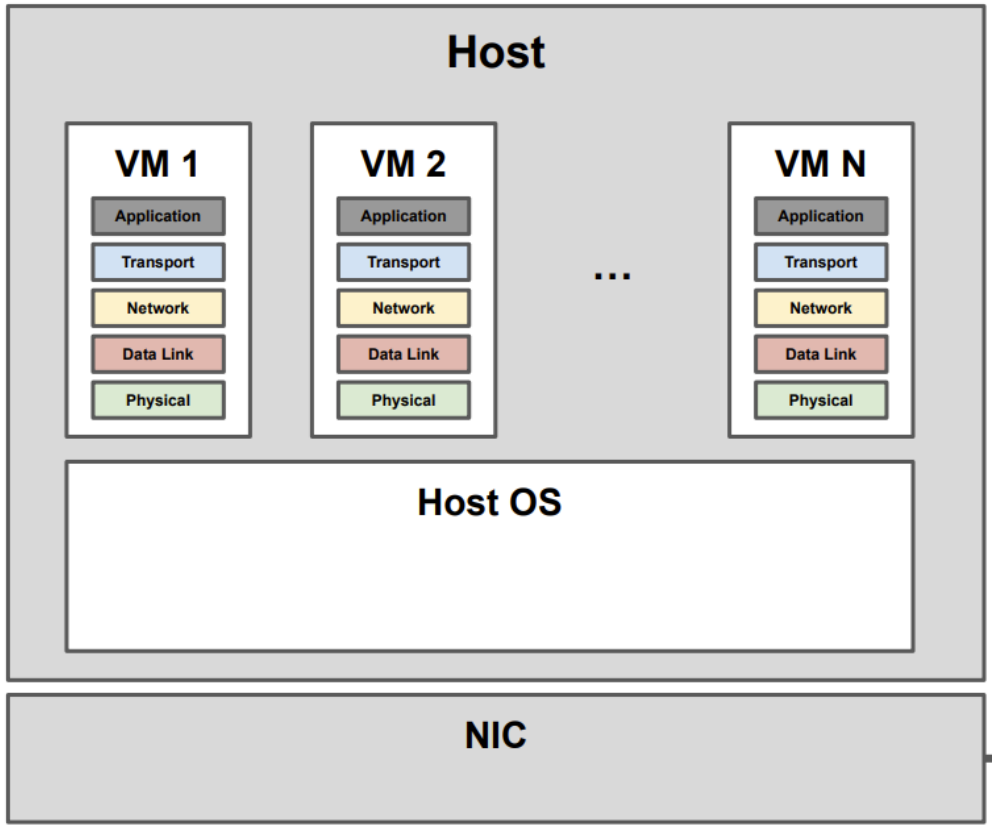
- Each VM has its own network stack (with a virtualized NIC)
- The provider of the virtualized platform needs to:
  - forward traffic between VMs or VMs and the NIC.
  - manage how to share network resource between VMs.



3. Network Processing in Virtualized Platforms

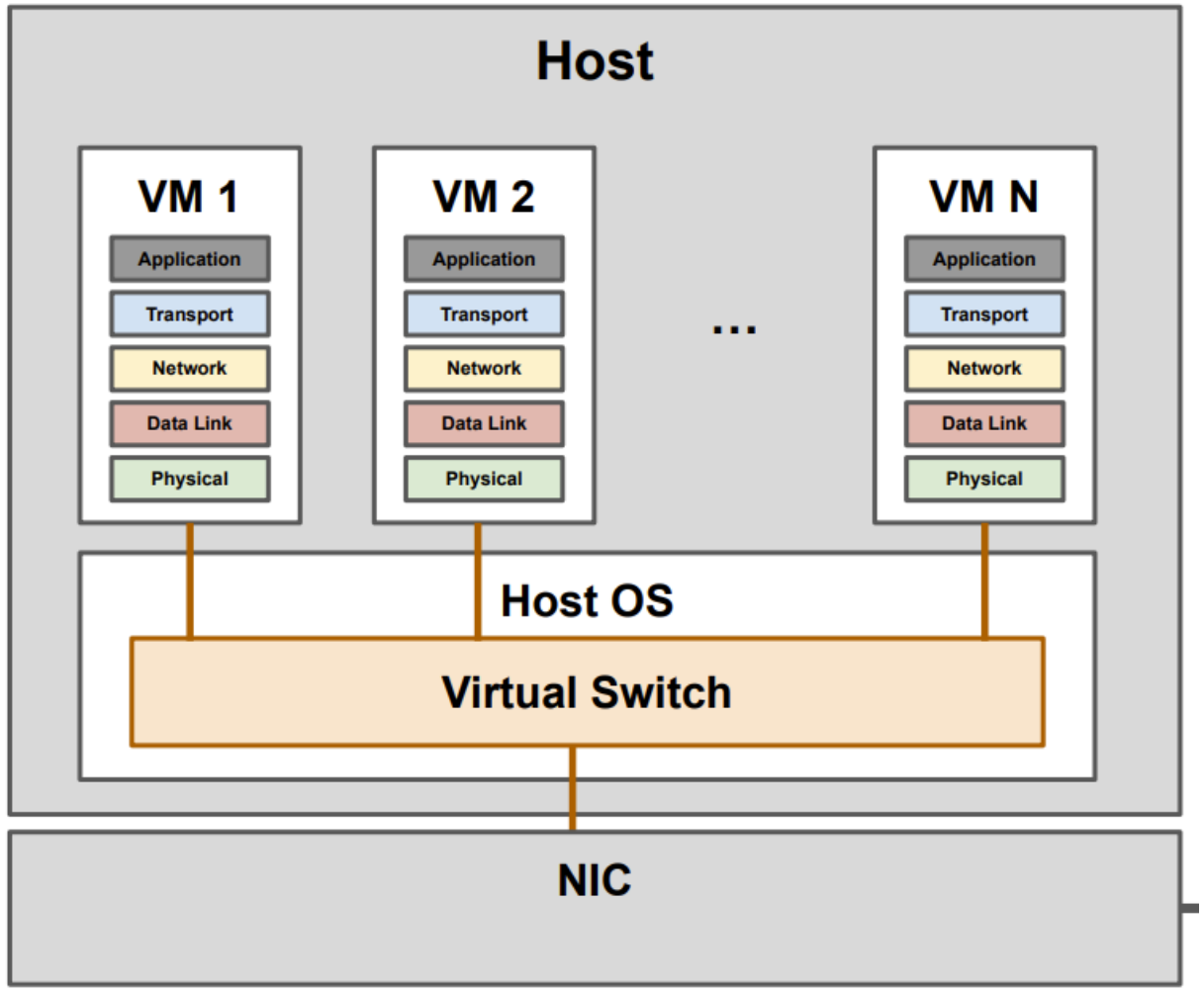
Server Virtualization

- Each VM has its own network stack (with a virtualized NIC)
- The provider of the virtualized platform needs to:
  - forward traffic between VMs or VMs and the NIC.
  - manage how to share network resource between VMs.



3. Network Processing in Virtualized Platforms

Virtual Switch (vSwitch)



3. Network Processing in Virtualized Platforms

Network Function

- Traditionally, switches and routers only do packet processing up to and including layer 3 (the network layer) to do forwarding.

- But soon, it became apparent we may need to do more than just forwarding in the middle of the network and may need to look further into packets (i.e., high layers of the stack).

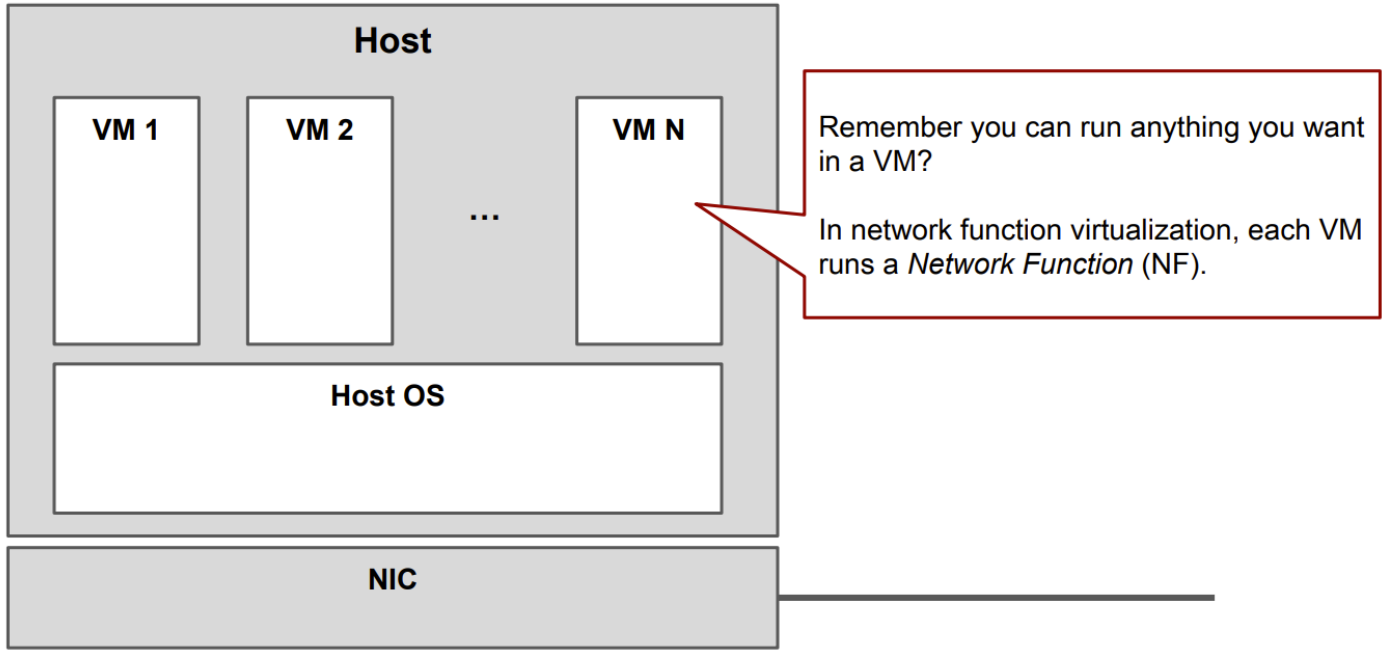
- They were called middleboxes.

- Network function is a generic term to describe any kind of network processing, specially the more advanced middlebox-like packet processing.

3. Network Processing in Virtualized Platforms

Network Function

If network function virtualization (NFV), network functions are as software inside VMs instead of each having a separate (specialized) physical device.



HUST

hust.edu.vn fb.com/dhbkhn

THANK YOU !