High Performance Network Middleware with Intel DPDK and OpenNetVM

Tim Wood
George Washington University

Middleware 2017 Tutorial

Thanks to K.K. Ramakrishnan, Wei Zhang, Grace Liu, Jinho Hwang,
Neel Shah, Phil Lopreiato, CloudLab.us, and many others…
Some slides borrowed from Jay Muthurajan's University DPDK Lecture (Intel)

THE GEORGE WASHINGTON UNIVERSITY

UCR

Tim Wood

Associate Professor at George Washington University

- PhD from University of Massachusetts Amherst in 2011

Research Areas:

- Virtualization, operating systems
- Cloud management
- recently: software-based networks (SDN and NFV)

George Washington University

- Located in Washington DC, USA
- ~15 faculty, 80 PhD, 300 MS and 150 undergraduate students
- **We are hiring!**

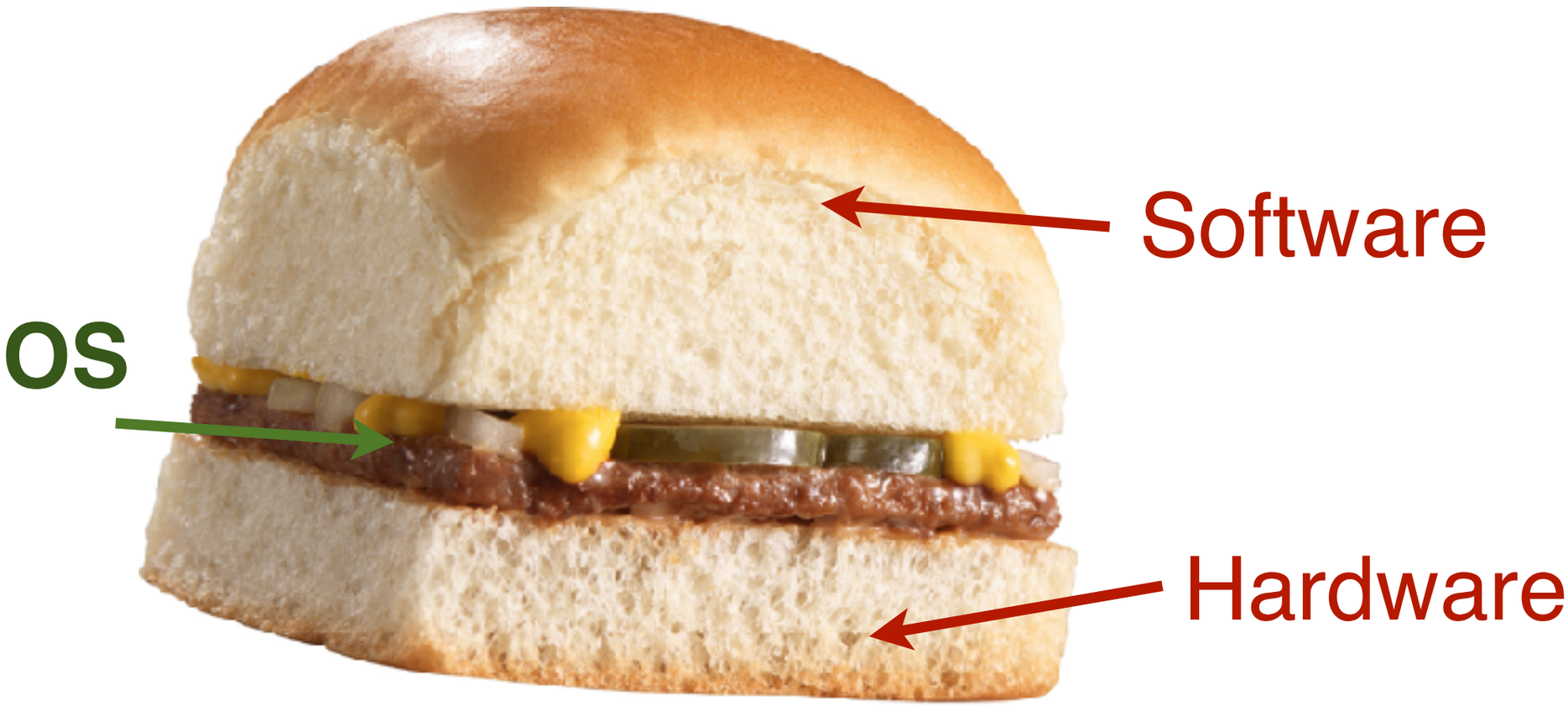
www.tim-wood.net

Operating Systems 101

abstraction layers and trade-offs

The OS

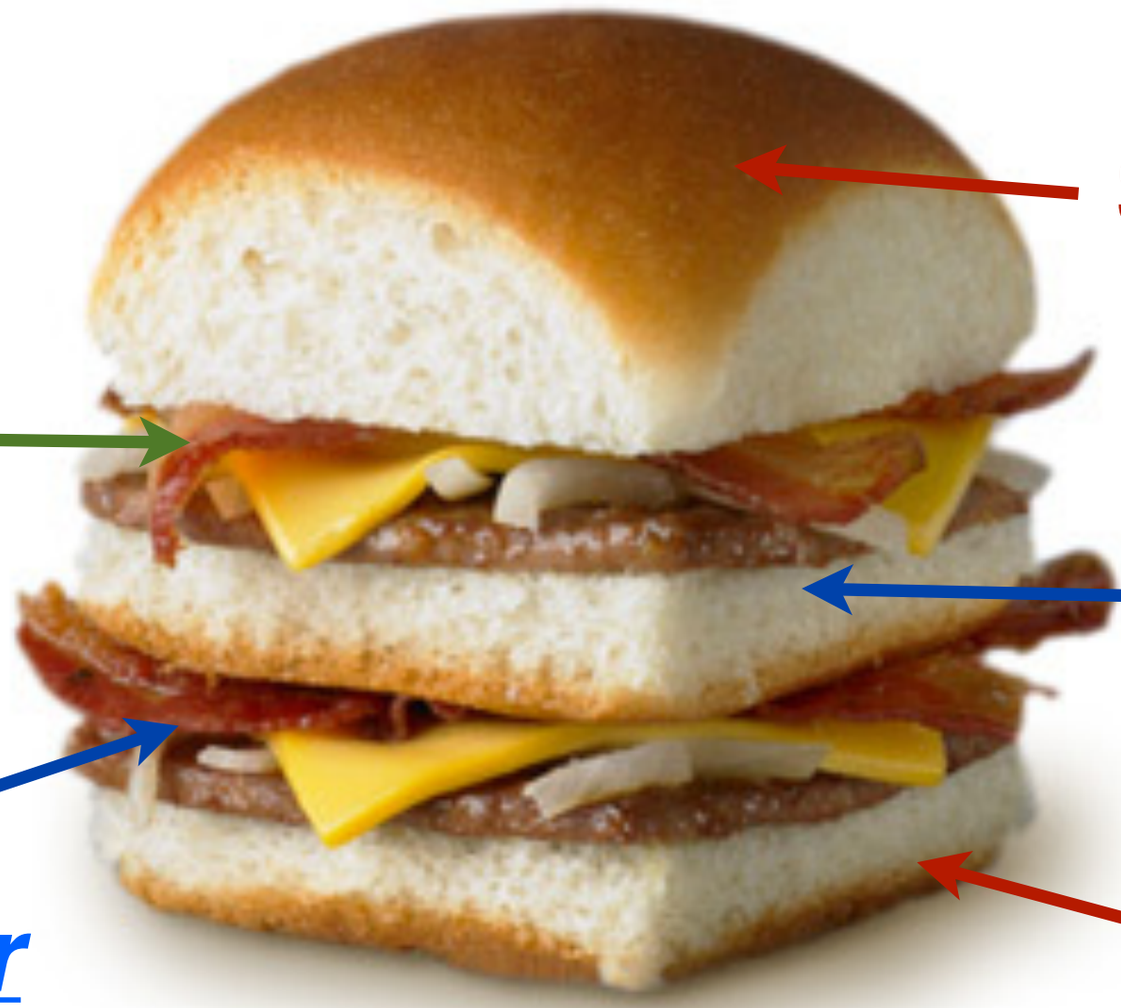
Delicious, with no fluff...



What could make this even better?

Virtualization Layers

Even more of all the good parts!



Software

OS

Virtual HW

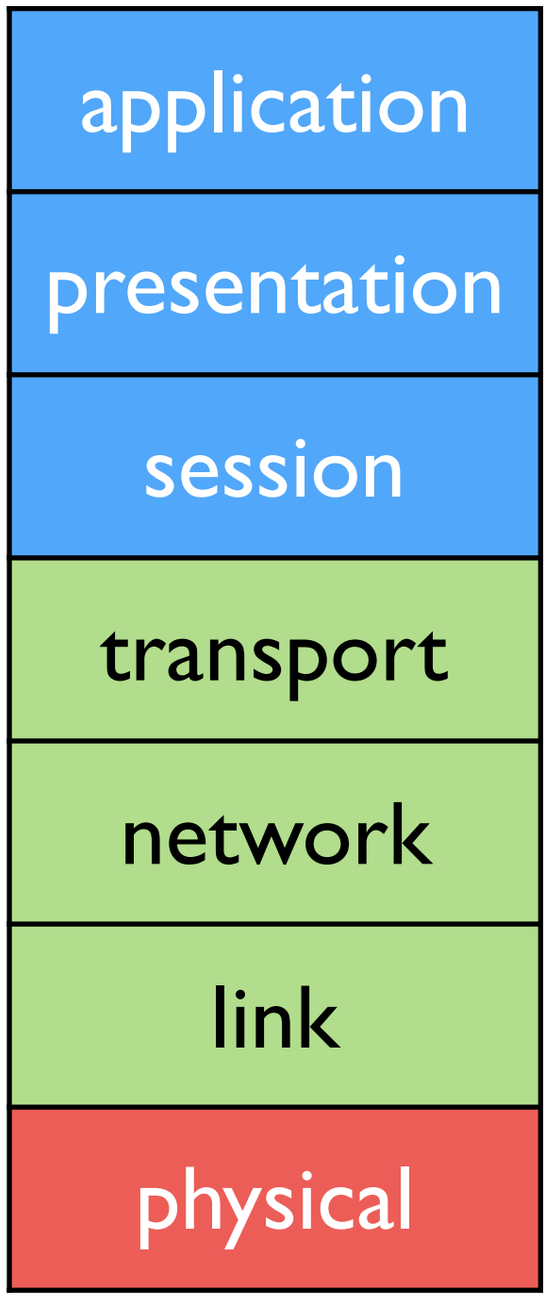
Hypervisor

(another operating system)

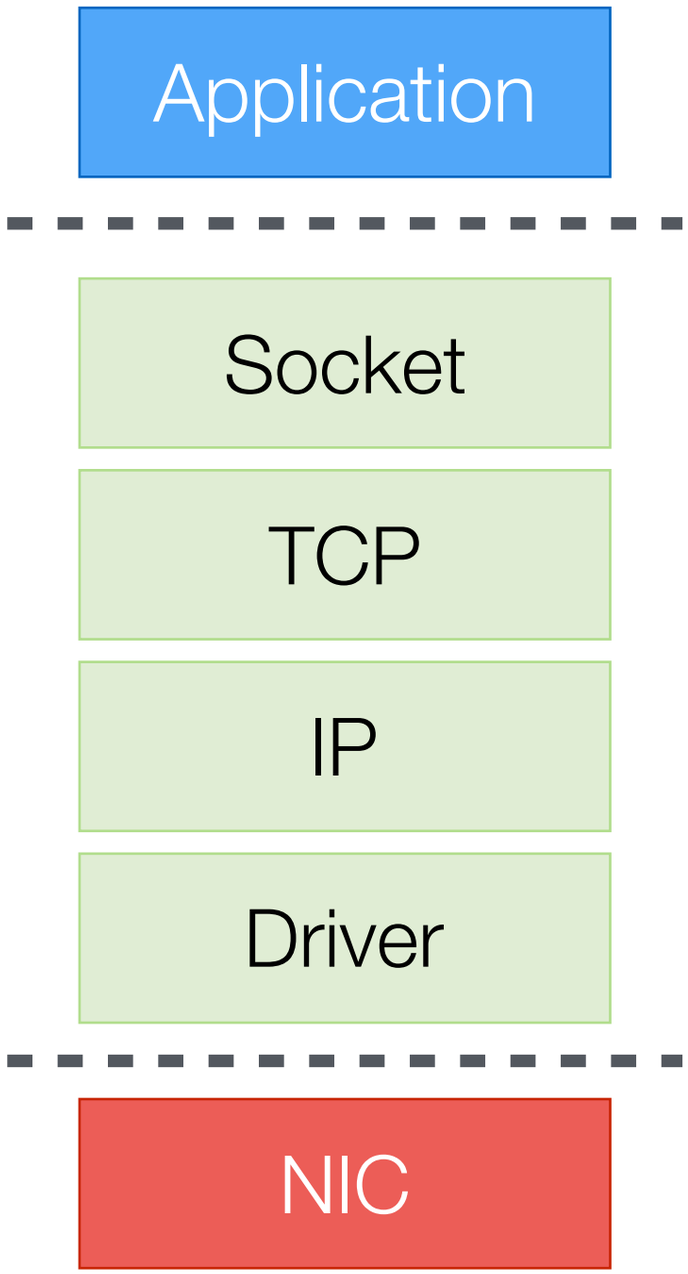
Hardware

Networks rely on layers

OSI Model



System Structure



Networks for Systems Folk

Topics for today:

Overview of software-based network tech
Systems challenges related to networking
Middleboxes = middleware
Opportunities for network middleware

DPDK and OpenNetVM

Systems Details!

DPDK: low-level network I/O library

OpenNetVM: an NFV platform optimized for high performance service chains built on DPDK

- Open source project, collaboration between GW and UCR

Goals:

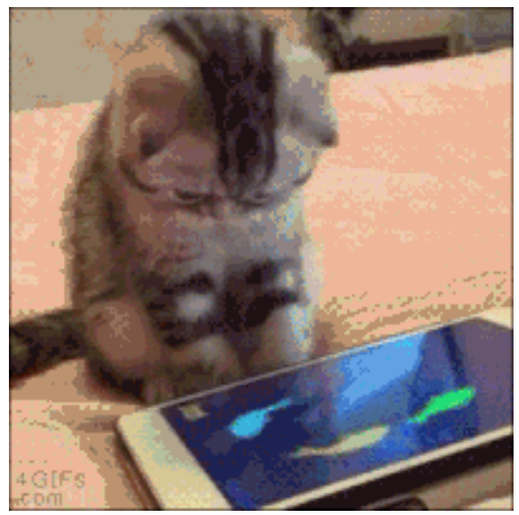
- Understand the design of DPDK and OpenNetVM
- Be able to run OpenNetVM's manager and network functions

Mix of presentation and tutorial

How does this work?

Me

??????????????????

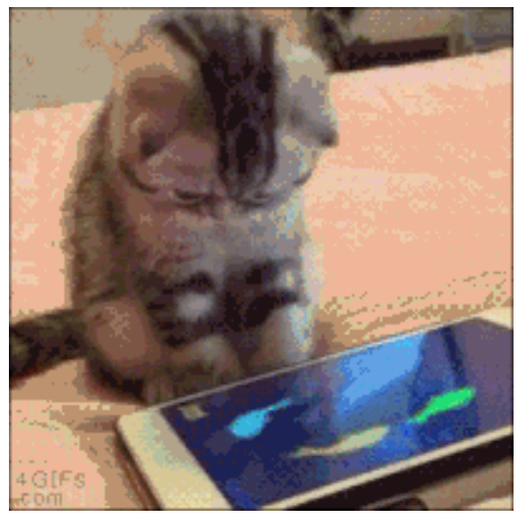


a cat

How does this work?

Me

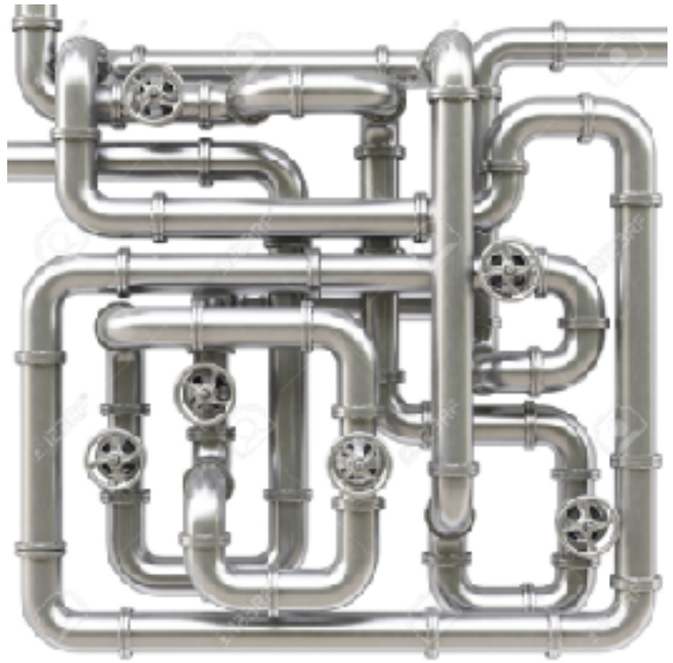
??????????????????

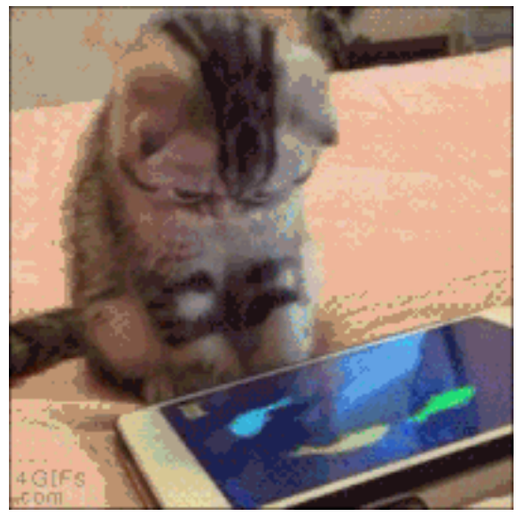


a cat

A series of tubes?

Me





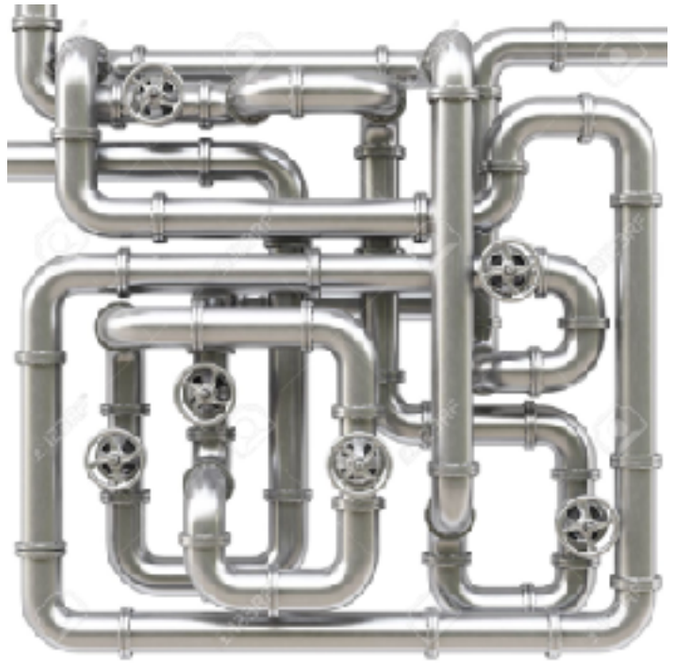
a cat

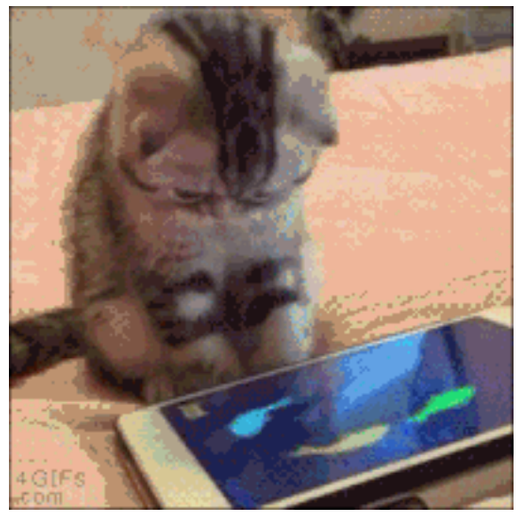
"*...the Internet is not something that you just dump something on.* ***It's not a big truck. It's a series of tubes****..."*

-- United States Senator Ted Stevens (R-Alaska)

A series of tubes?

Me





a cat

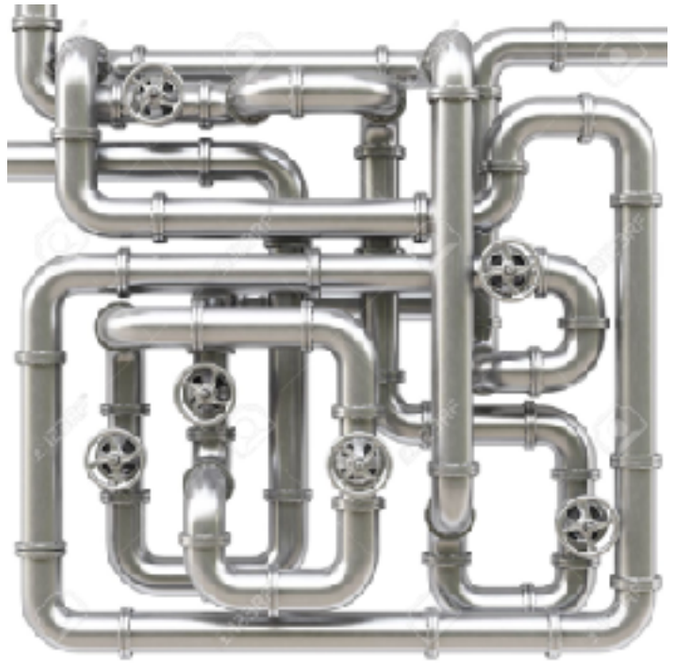
"*...the Internet is not something that you just dump something on.* ***It's not a big truck. It's a series of tubes****..."*

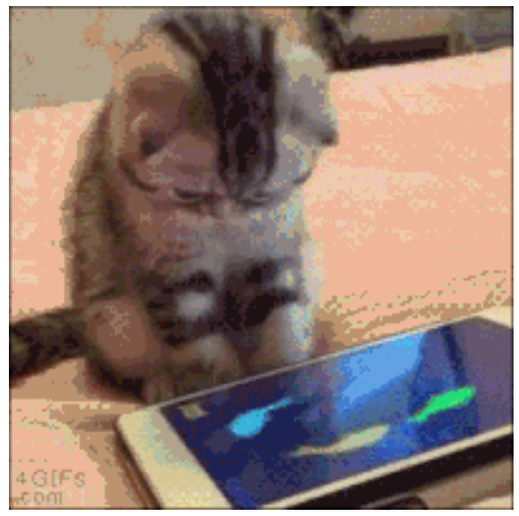
-- United States Senator Ted Stevens (R-Alaska)

Not a truck!

Not a truck! A series of tubes!

Me



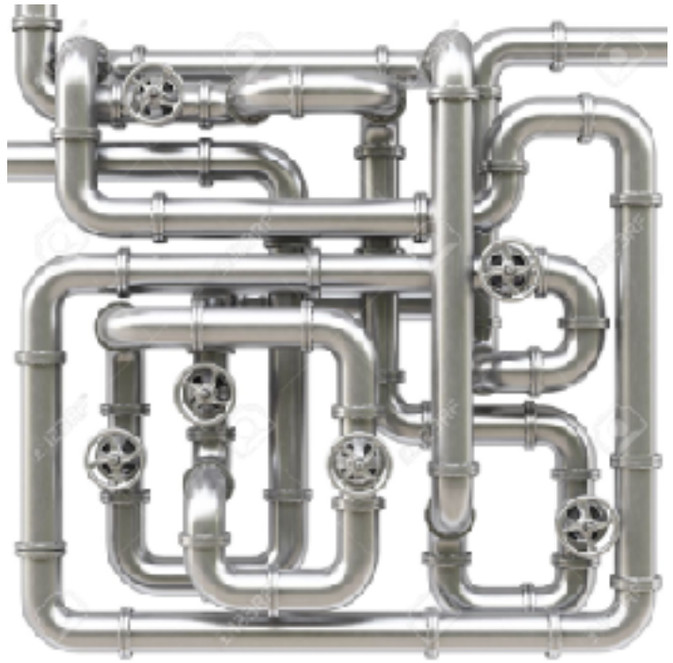


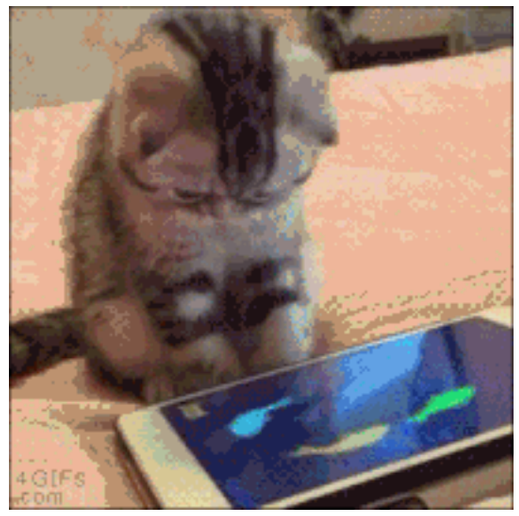
a cat

Even if it is all tubes... is it really that easy?

Not a truck! A series of tubes!

Me

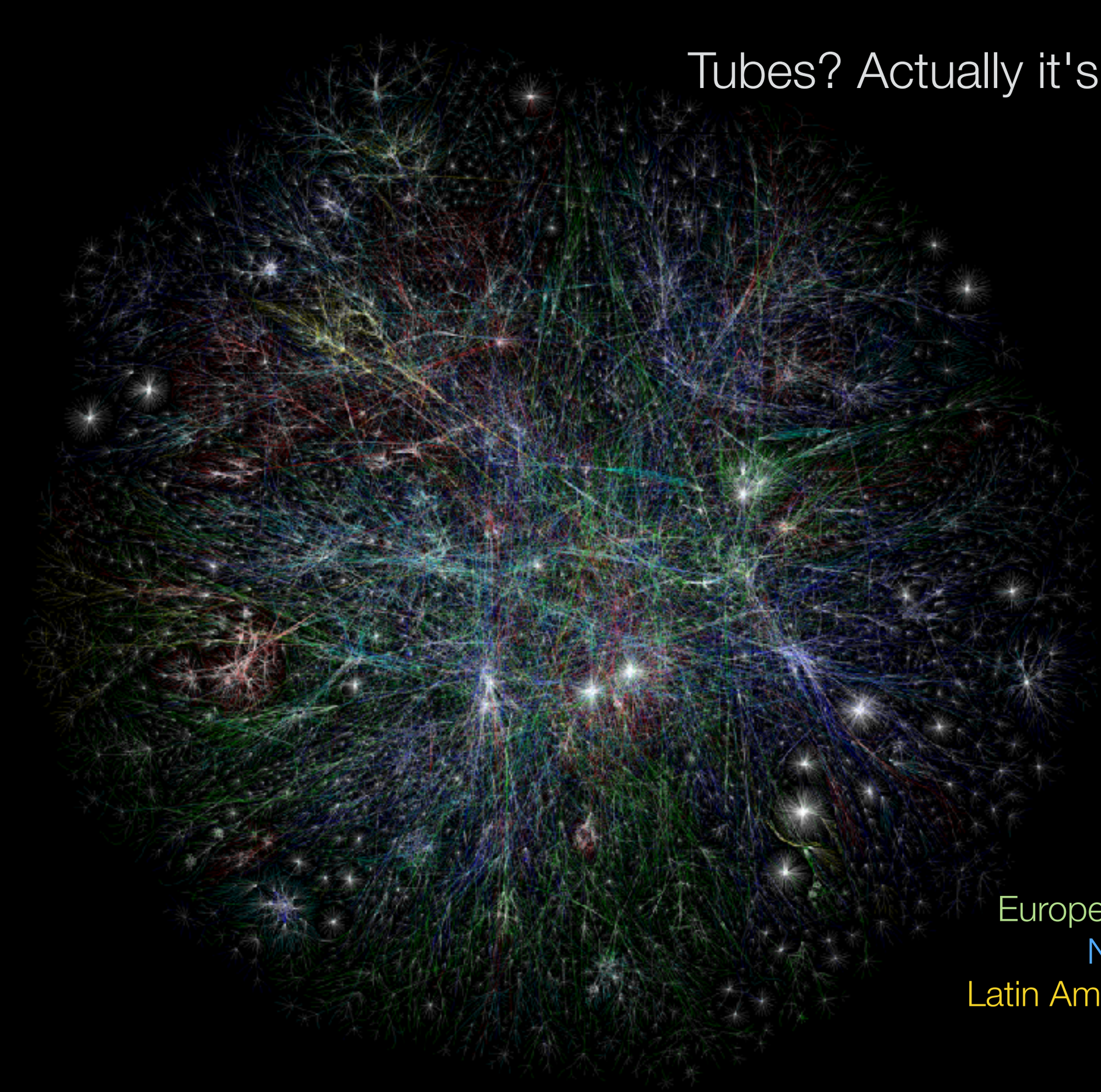




a cat

Even if it is all tubes... is it really that easy?

Tubes? Actually it's more like a web.



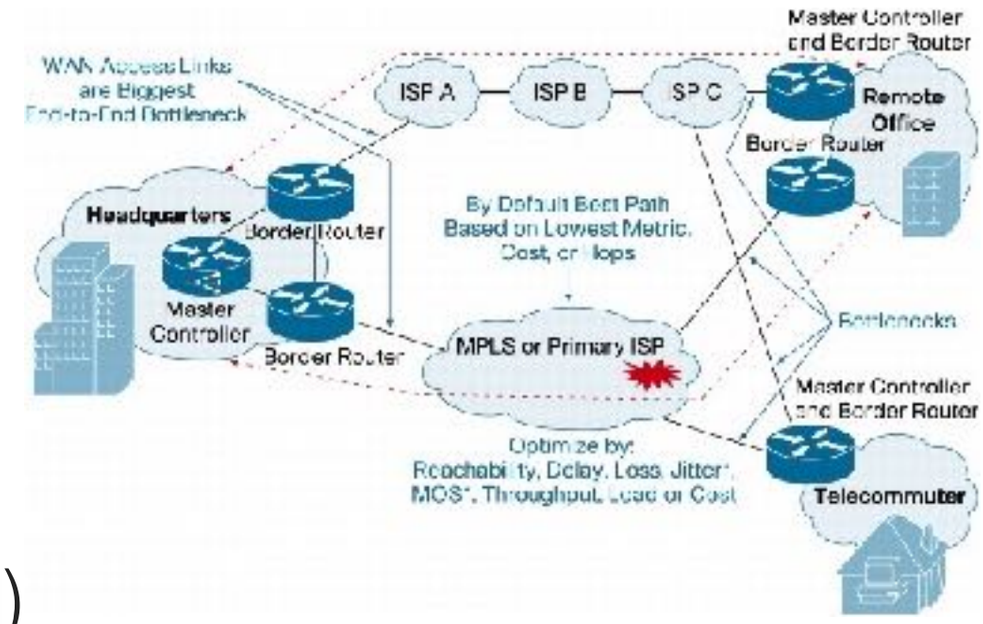
and it is
really
really BIG

Asia Pacific
Europe/Central Asia/Africa
North America
Latin American and Caribbean
Unknown

Networks are Changing

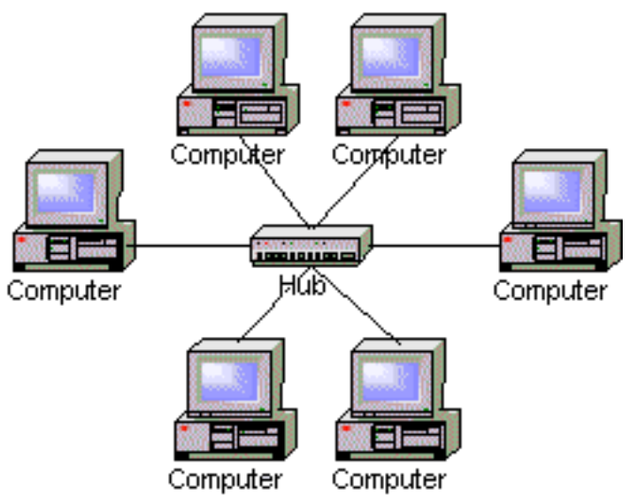
Scale and organization:

- Cloud data centers, mobile users
- Large-scale, highly dynamic

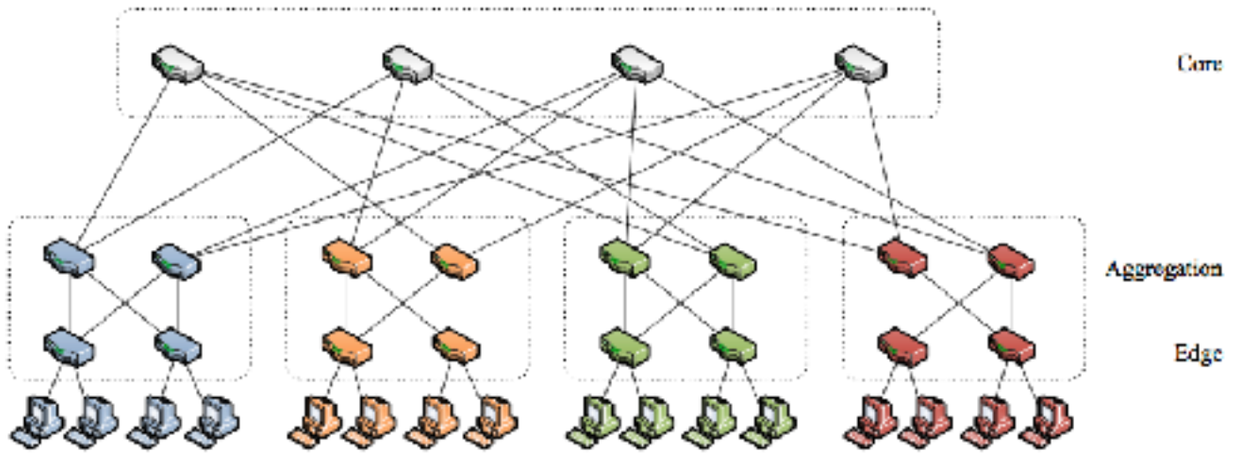


Hardware and control plane

- Software Defined Networking (SDN)
- Network Function Virtualization (NFV)



Then



Now…

Networks 101

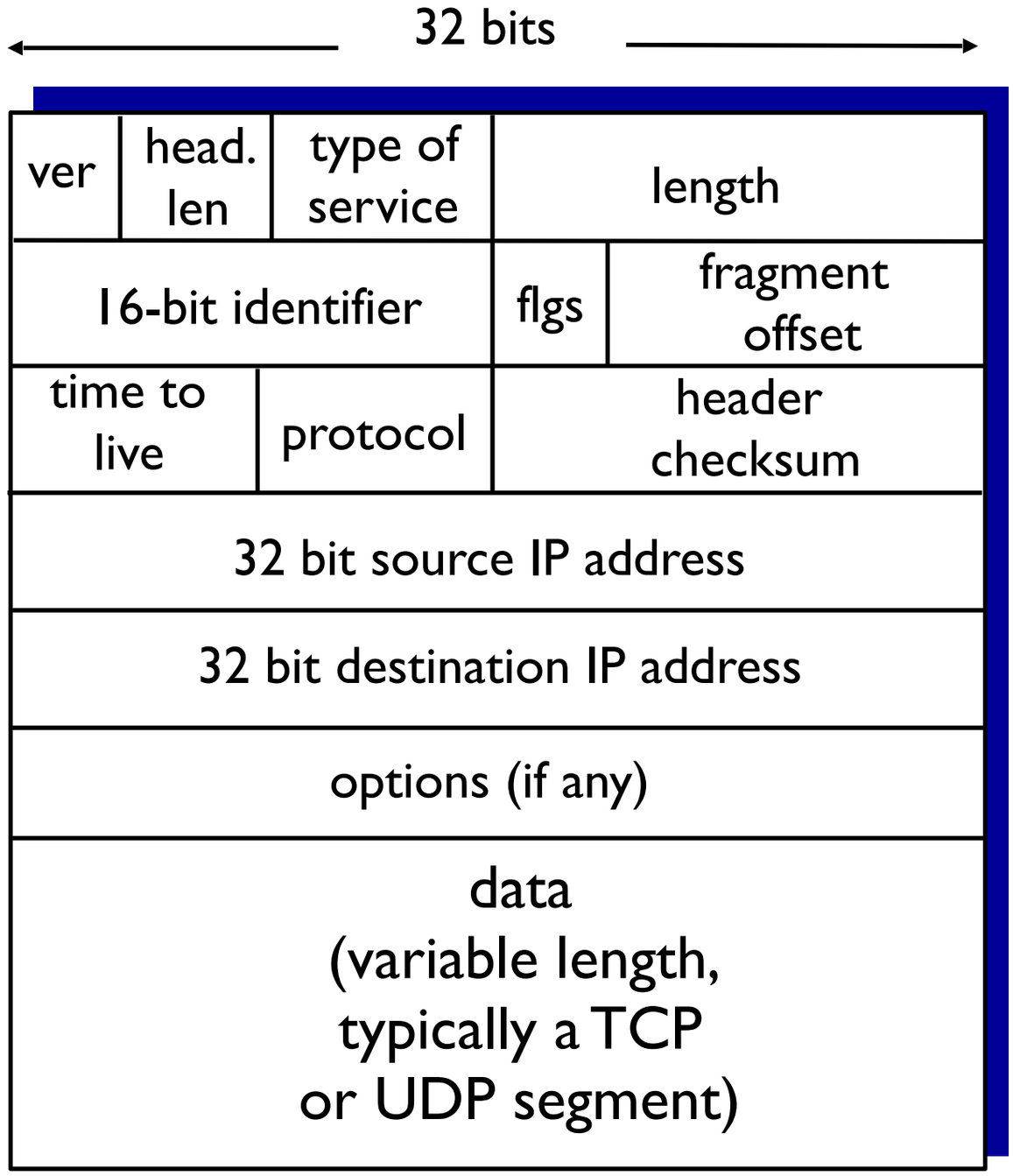
An application wants to send data…

Data gets broken into packets

Packets have a header to help find destination

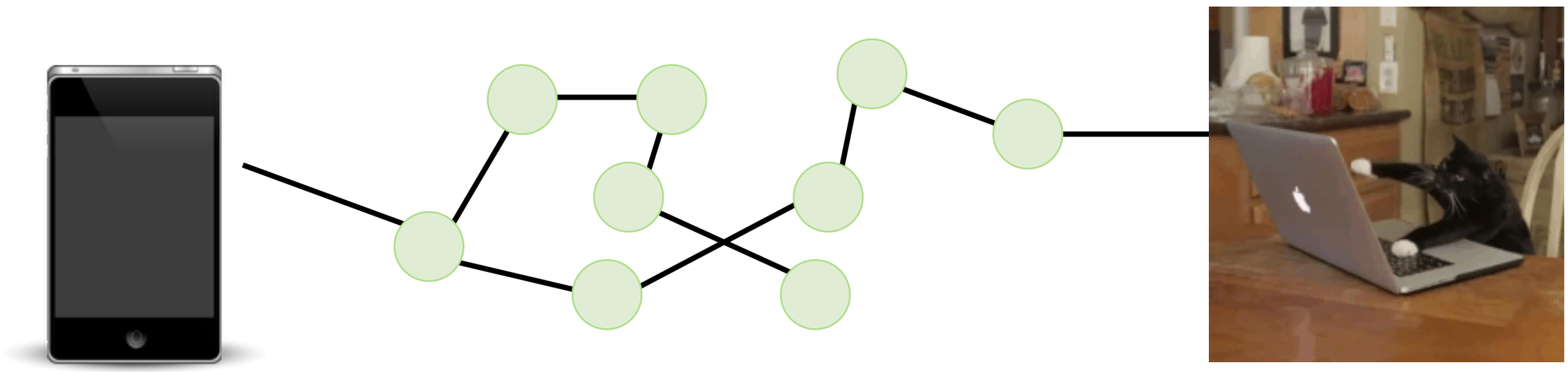
Headers for different protocols are layered

Network Interface Card (NIC) transmits packets out a wire



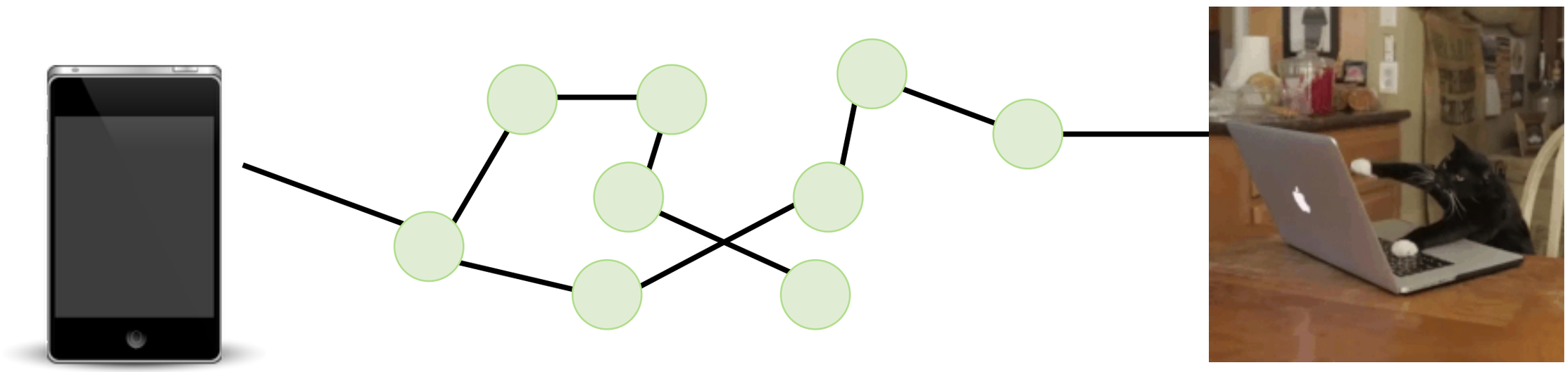
IP packet

Finding a Path



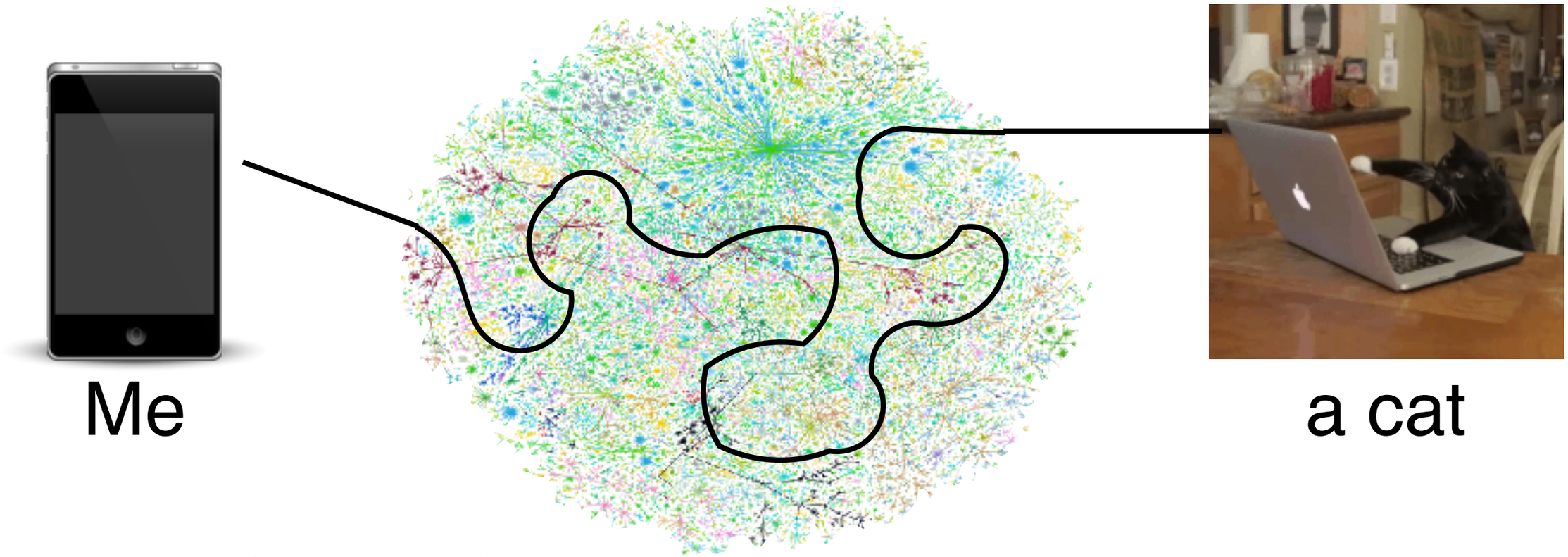
Can you solve this?

Finding a Path



Can you solve this?

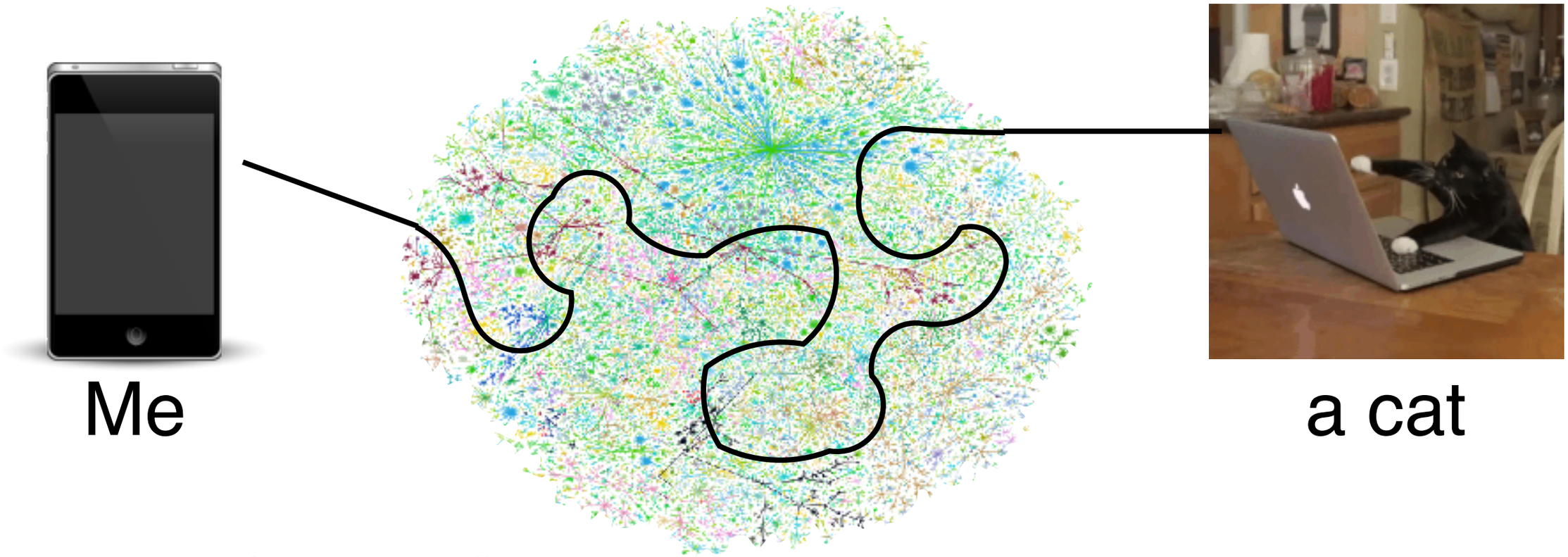
Finding a Path



Can you solve this?

- What information do you need?
- Is this practical?
- How does it scale?

Finding a Path



Can you solve this?

- What information do you need?
- Is this practical?
- How does it scale?

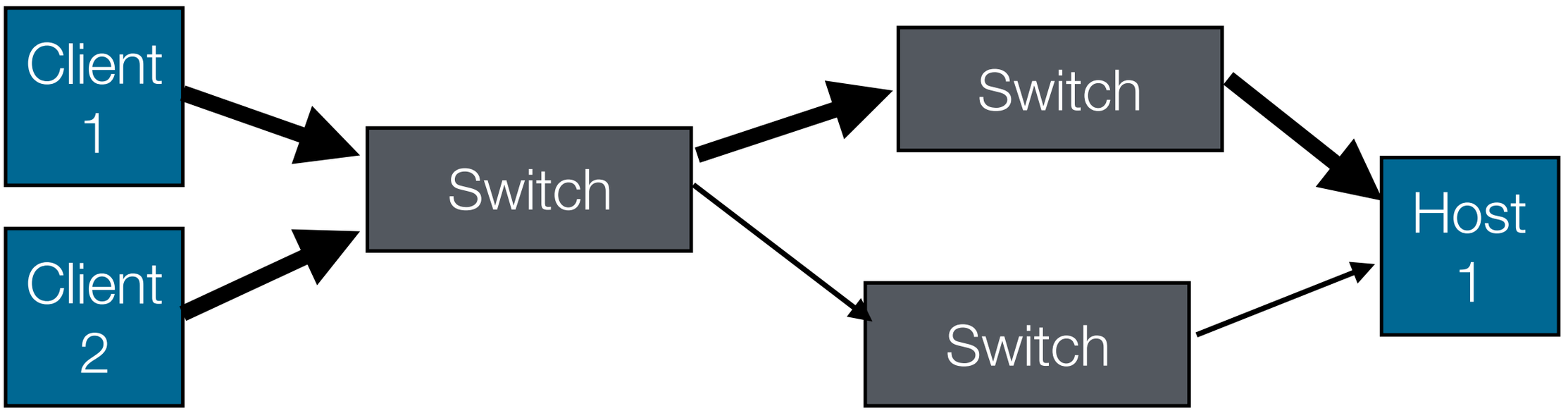
Networks 101

Routers and switches direct packets

- Router: operates between local networks
- Switch: connects hosts within a local network

Use distributed protocols to find shortest paths

- Always follow same path
- **Trade-offs?**



Software Defined Networks

Create a centralized network **control plane**

- Routing rules for each flow

Simple, fast data plane

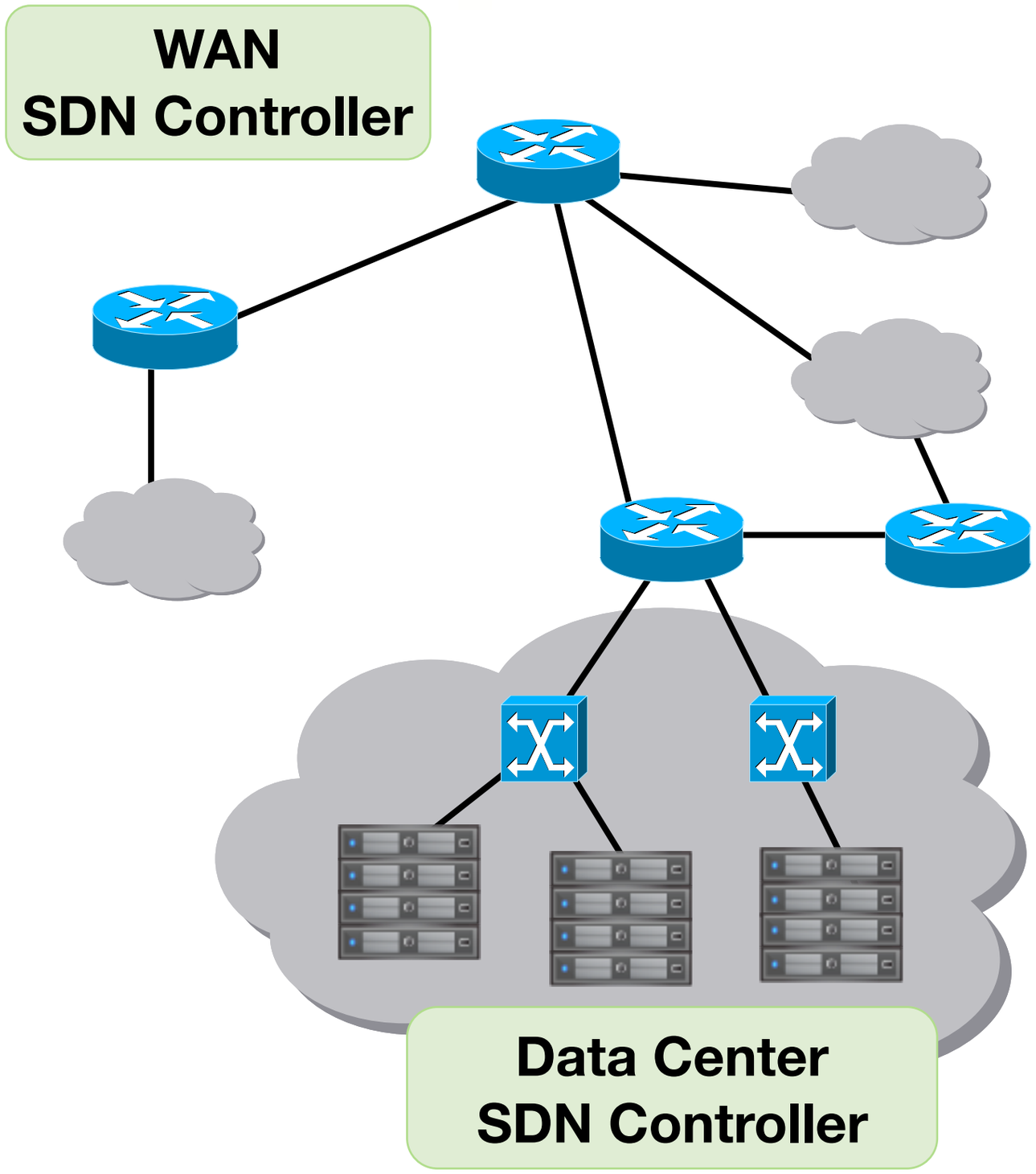
- Routers, switches, etc
- Ask SDN Controller for help

SDN lookup

- Send first packet of unknown flow to controller

Match/action rules

- in: n-tuple (header info)
- out: next hop destination



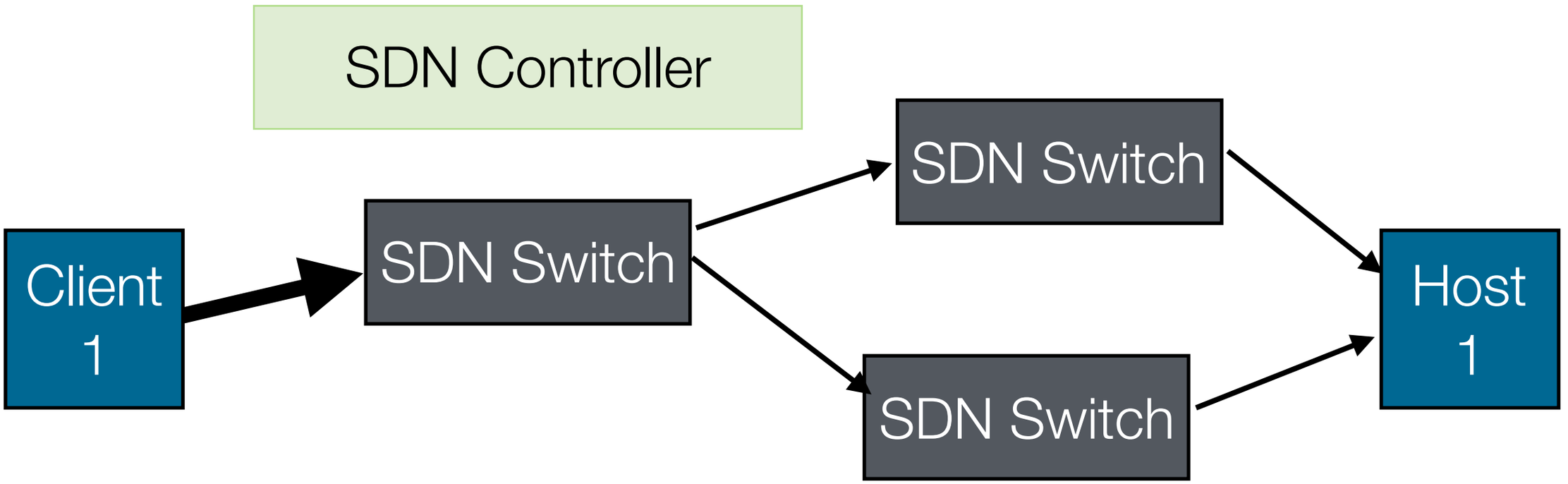
SDN Workflow

Data plane (switches): maintains a flow table

- Flow = one point-to-point connection (Src/Dest IP and Port)
- Action = how switch should process the packet

Control plane: populates flow table rules on switches

- Can be based on business logic
- Select next hop, drop, mirror, etc.



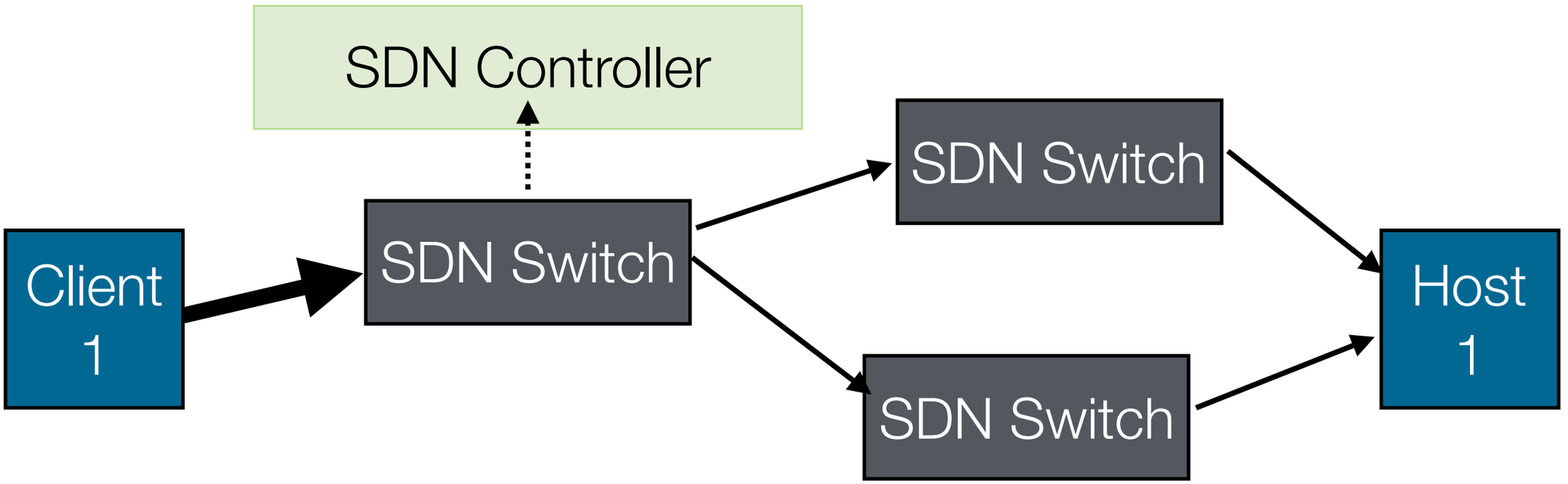
SDN Workflow

Data plane (switches): maintains a flow table

- Flow = one point-to-point connection (Src/Dest IP and Port)
- Action = how switch should process the packet

Control plane: populates flow table rules on switches

- Can be based on business logic
- Select next hop, drop, mirror, etc.



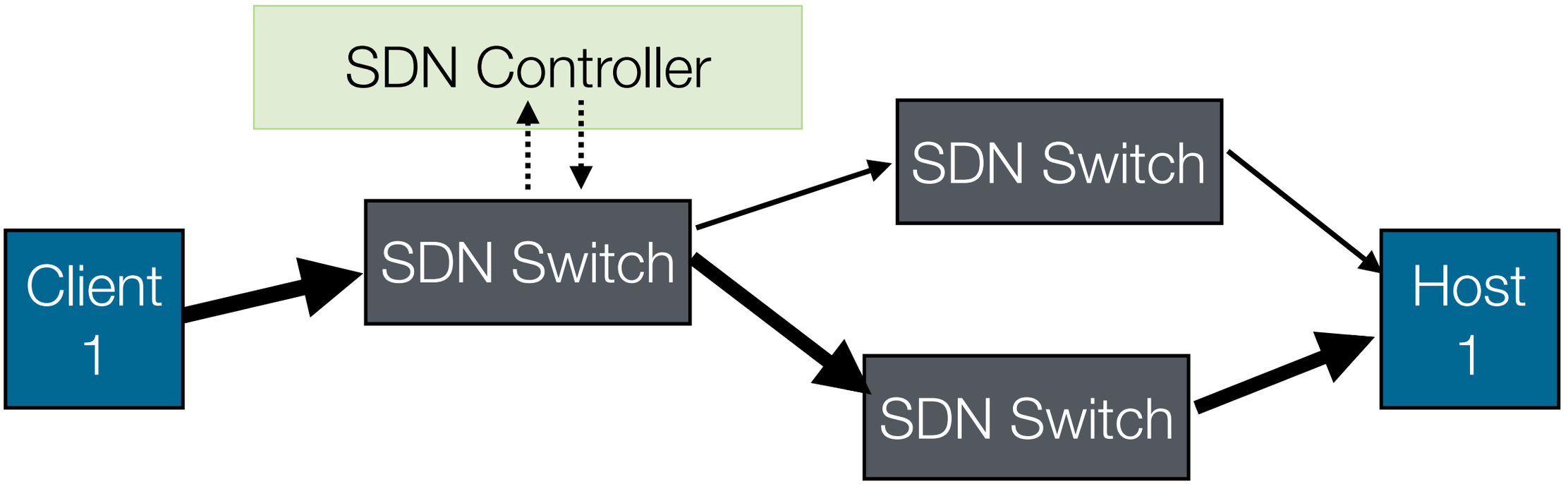
SDN Workflow

Data plane (switches): maintains a flow table

- Flow = one point-to-point connection (Src/Dest IP and Port)
- Action = how switch should process the packet

Control plane: populates flow table rules on switches

- Can be based on business logic
- Select next hop, drop, mirror, etc.



SDN Resources

SDN Controllers

- OpenDaylight
- Nox/Pox
- Project Floodlight
- Ryu

SDN controlled switches

- Your favorite HW vendor
- Open vSwitch (software)

Mininet

- Network emulator for experimenting with SDN control

SDN/NFV Conferences

- Sigcomm, NSDI, SOSR, HotSDN/HotNets/HotMiddlebox

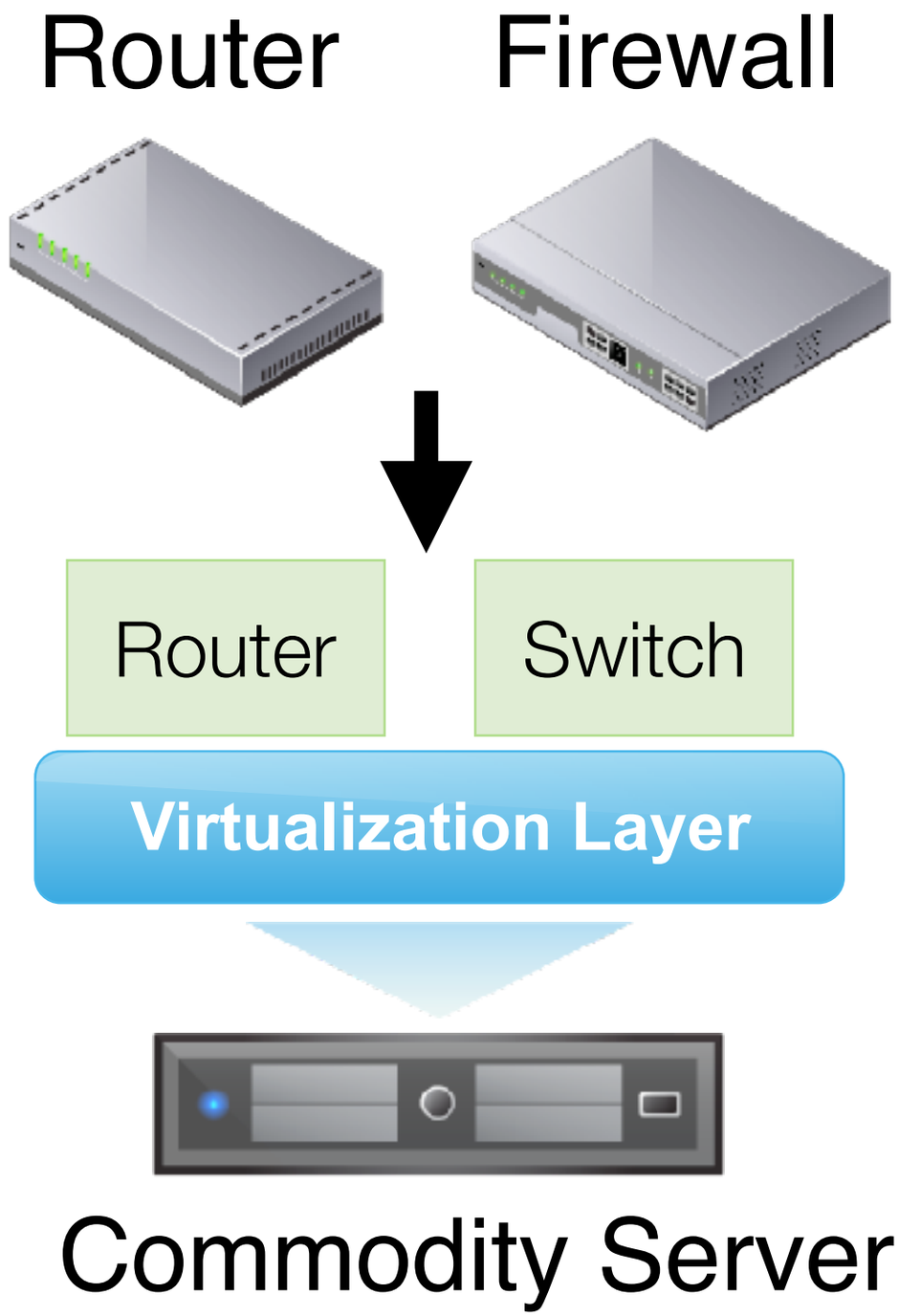
Network Function Virtualization

Make an efficient, customizable **data plane**

- routers, switches, firewalls, proxies, IDS, DPI, etc

Run network functions (NFs) in virtual machines

- More flexible than hardware
- Isolates functionality, easy to deploy and manage
- Slower than hardware…



Software-based Networking

Network is more than just transport

- Security, QoS, accounting, caching, transcoding

Systems-level support for network software is lacking

- Needs careful resource management
- Maximizing parallelism and minimizing latency are crucial

Opportunity to redesign!

Convergence of research areas:

- Operating Systems
- Networking
- Real Time Computing
- Security
- Modeling
- Stream Processing

Network Hardware

Perform network functionality on custom ASICs

Fast, **expensive**, **inflexible**

Cisco ASR 9001 Router

- **Dimensions:** Height:3.5" Width:17.4" Depth:18.5"
- **Weight:** 30.20 lb
- **Features:** Product Type:Router Chassis Number of Total Expansion Slots:7 Form Factor:Rack-mountable Compatible Rack Unit:2U VoIP Supported:No Expansion Slot Type:Port Adapter SFP+ Product Name:ASR 9001 Router Standard Memory:8 GB
- **Model #:** ASR 9001
- **Item #:** N82E16833420947
- **Return Policy:** Standard Return Policy

Compare

$33,650.99

$5.99 Shipping

ADD TO CART ▶

Software-Based Networking

Hardware Routers and Switches

- Expensive, single purpose
- Controllable with SDNs, but not flexible

PacketShader [Han, SIGCOMM '10]

- Use commodity servers and GPUs
- 39 Gbps processing rates

Netmap [Rizzo, ATC '12] and DPDK

- Libraries to provide zero-copy network processing on commodity 10gbps NICs

ClickOS [Martins, NSDI '14] and NetVM [Hwang, NSDI '14]

- VM based network services
- Flexible deployment and composition

Network Functions (NFs)

Switches, routers, firewalls, NAT

AKA “middleboxes"

- Simple packet header analysis and forwarding

Intrusion Detection Systems (IDS)

- Deep packet inspection (DPI) beyond header to detect threats
- Must have high scalability to observe full packet flows

Intrusion Prevention Systems (IPS)

- Similar to IDS, but deployed in-line, so it can actively manipulate traffic flows
- Must be efficient to avoid adding delay

Cellular functions (Extended Packet Core - EPC)

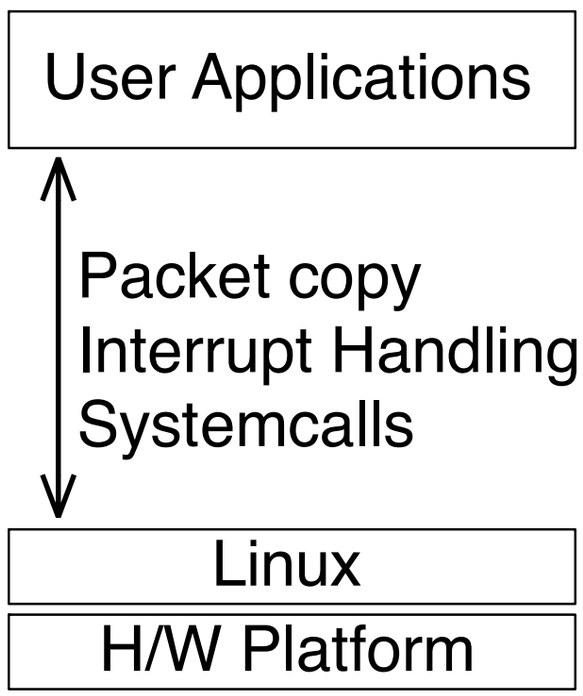
- Mobility management, accounting, security, etc.

Proxies, caches, load balancers, etc.

Linux Packet Processing

Traditional networking:

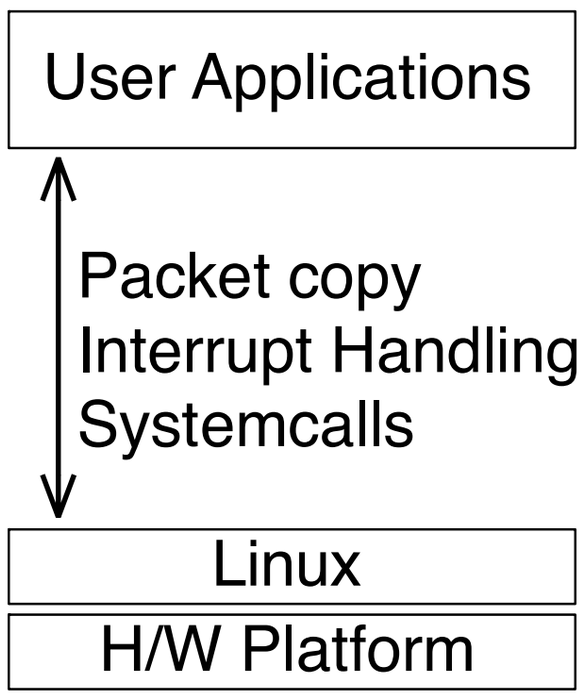
- NIC uses DMA to copy data into kernel buffer
- Interrupt when packets arrive
- Copy packet data from kernel space to user space
- Use system call to send data from user space



Linux Packet Processing

Traditional networking:

- NIC uses DMA to copy data into kernel buffer
- Interrupt when packets arrive
- Copy packet data from kernel space to user space
- Use system call to send data from user space

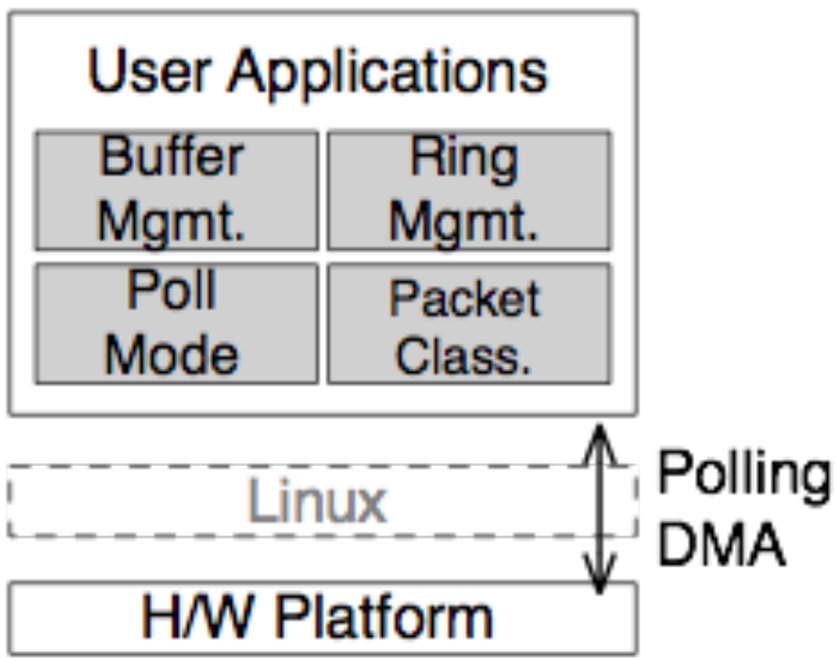


Can you handle being interrupted 60 million times per second?

User Space Packet Processing

Recent NICs and OS support allow user space apps to directly access packet data

- NIC uses DMA to copy data into ~~kernel~~ **user space** buffer
- ~~Interrupt~~ **use polling to find** when packets arrive
- ~~Copy packet data from kernel space to user space~~
- Use ~~system~~ **regular function** call to send data from user space



Data Plane Development Kit

High performance I/O library

Poll mode driver reads packets from NIC

Packets bypass the OS and are copied directly into user space memory

Low level library... does not provide:

- Support for multiple network functions
- SDN-based control
- Interrupt-driven NFs
- State management
- TCP stack

Data Plane Development Kit

Where to find it:

- http://dpdk.org/

What to use it for:

- Applications that need high speed access to low-level packet data

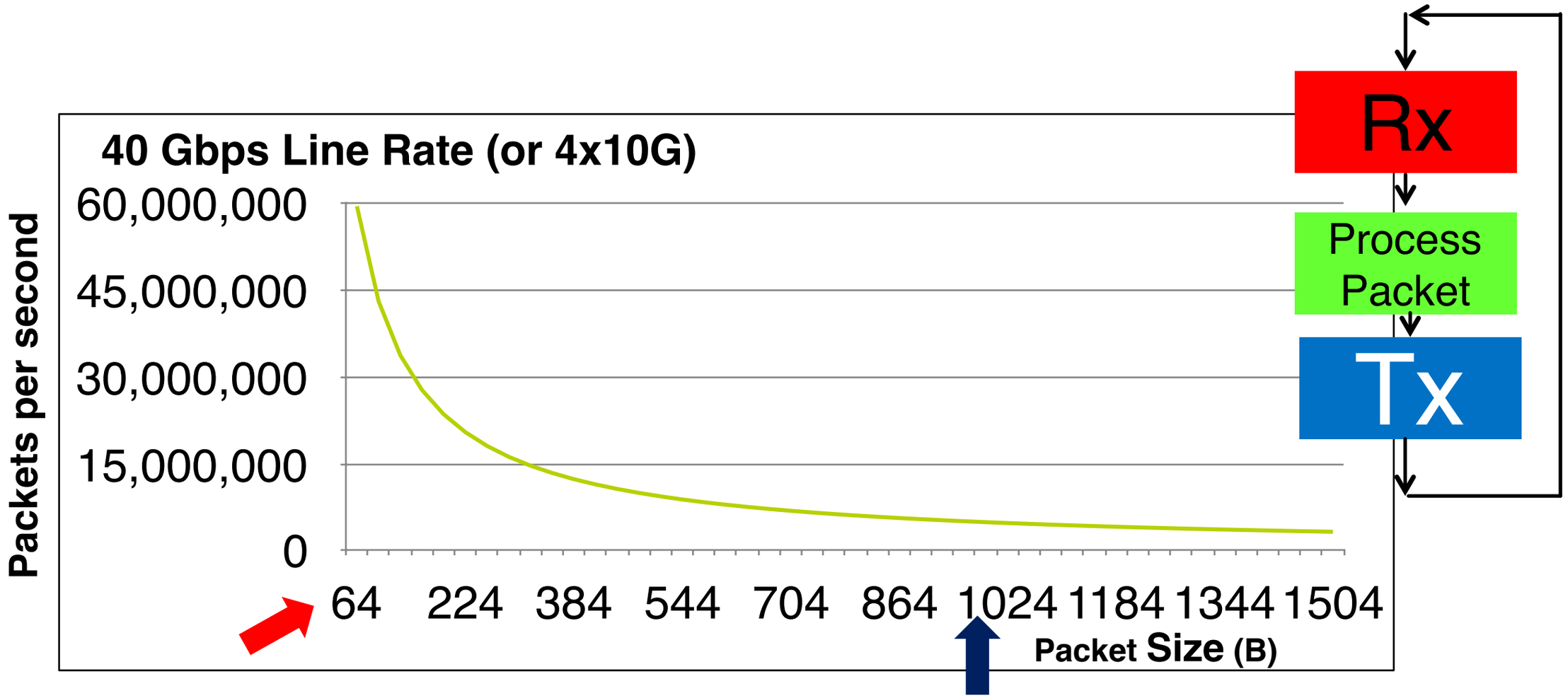
Why try it:

- One of the best documented open source projects I've ever seen

Alternatives:

- netmap
- PF_RING

What is “line rate”?



Network Infrastructure Packet Sizes

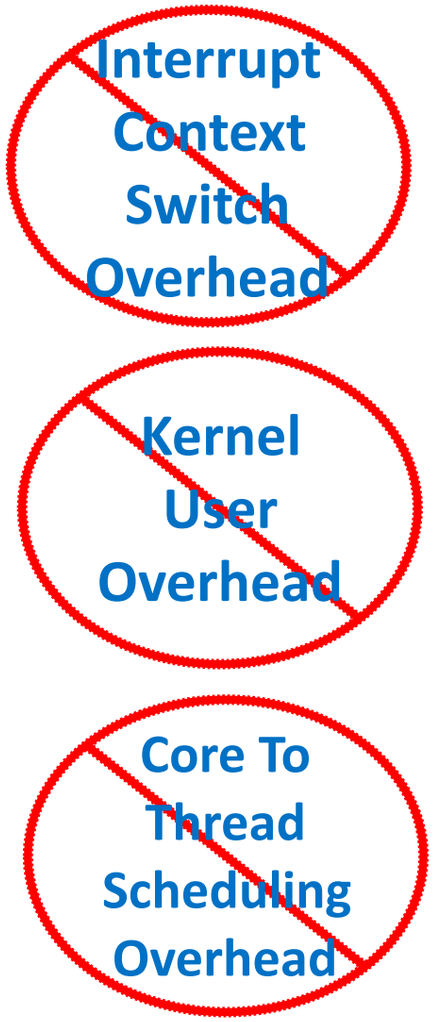
Packet Size	64 bytes
40G Packets/second	59.5 Million each way
Packet arrival rate	16.8 ns
2 GHz Clock cycles	**33 cycles**

Typical Server Packet Sizes

Packet Size	1024 bytes
40G Packets/second	4.8 Million each way
Packet arrival rate	208.8 ns
2 GHz Clock cycles	**417 cycles**

How to Eliminate / Hide Overheads?

How to Eliminate / Hide Overheads?

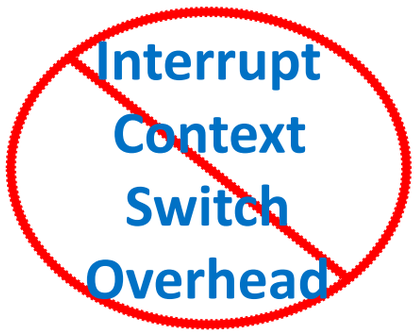


Polling

User Mode Driver

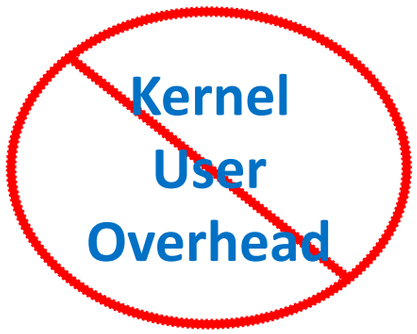
Pthread Affinity

How to Eliminate / Hide Overheads?



Polling

Huge Pages



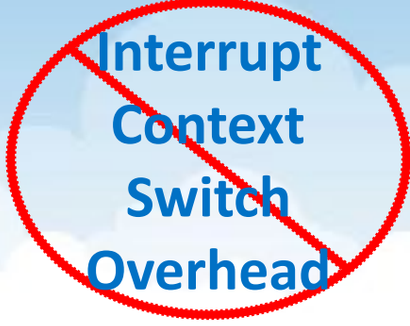
User Mode Driver

Lockless Inter-core Communication

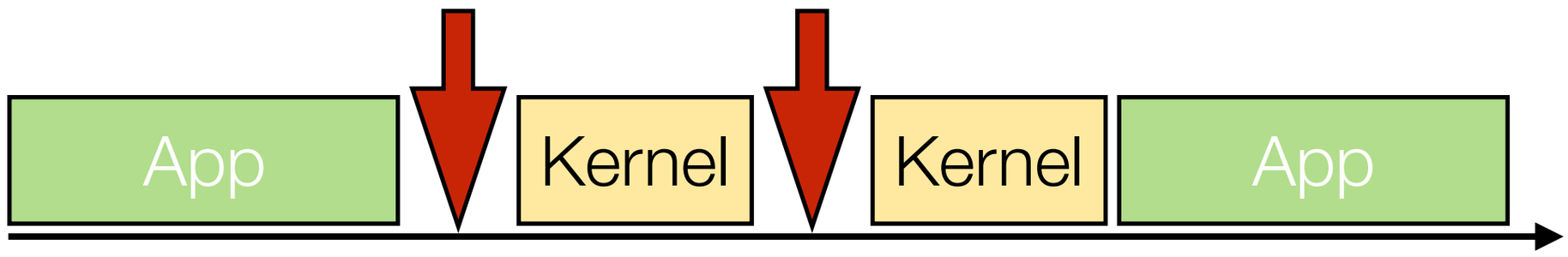
Pthread Affinity

High Throughput Bulk Mode I/O calls

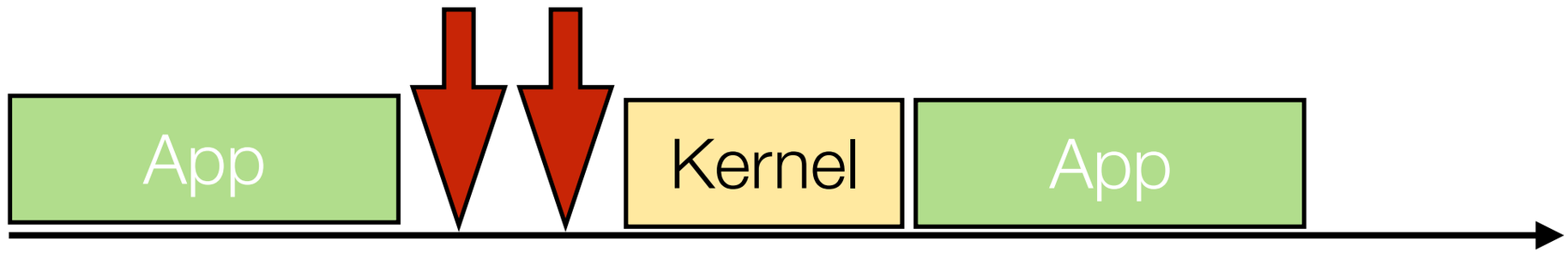
Network Interrupts



Very distracting! Have to stop doing useful work to handle incoming packets

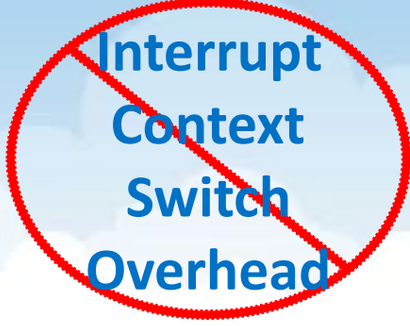


Coalescing interrupts helps, but still causes problems



- Interrupts can arrive during critical sections!
- Interrupts can be delivered to the wrong CPU core!
- Still must pay context switch cost

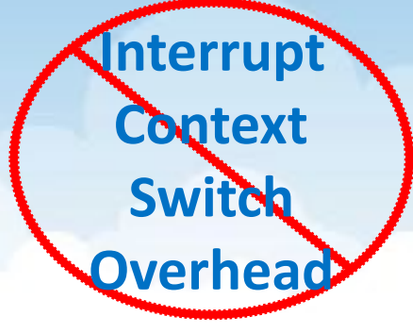
Polling



Continuously loop looking for new packet arrivals

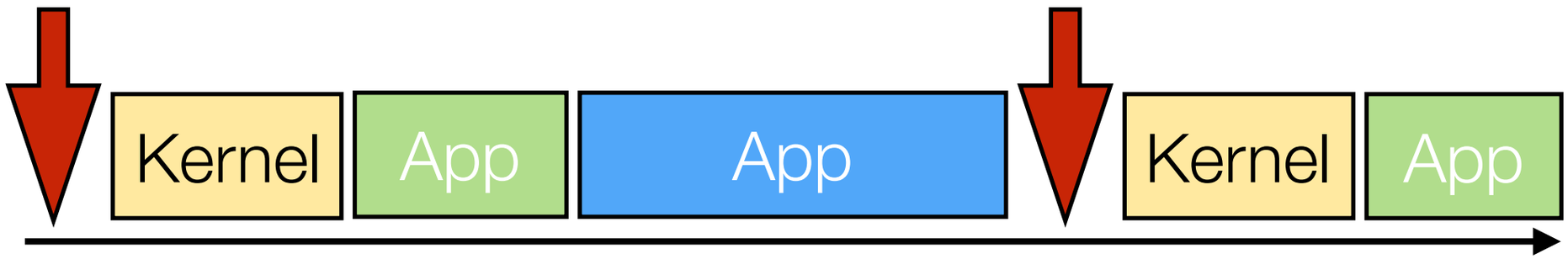
Trade-off?

Polling



Continuously loop looking for new packet arrivals

Trade-off?



Interrupts help share the CPU

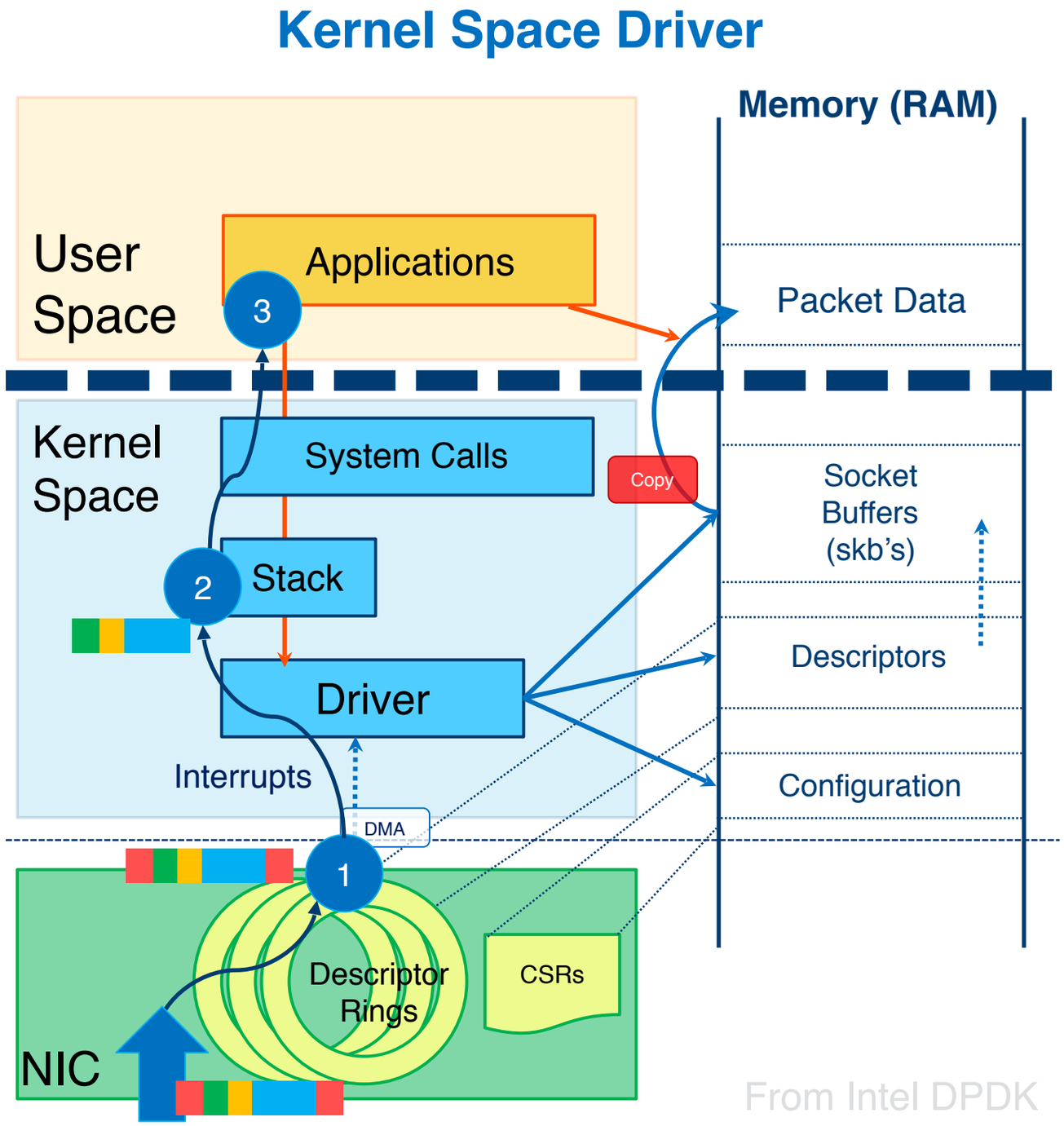
Polling can be wasteful

Kernel-User Overhead

NIC Driver operates in kernel mode

- Reads packets into kernel memory
- Stack pulls data out of packets
- Data is copied into user space for application
- Application uses system calls to interface with OS

Why is copying so bad?



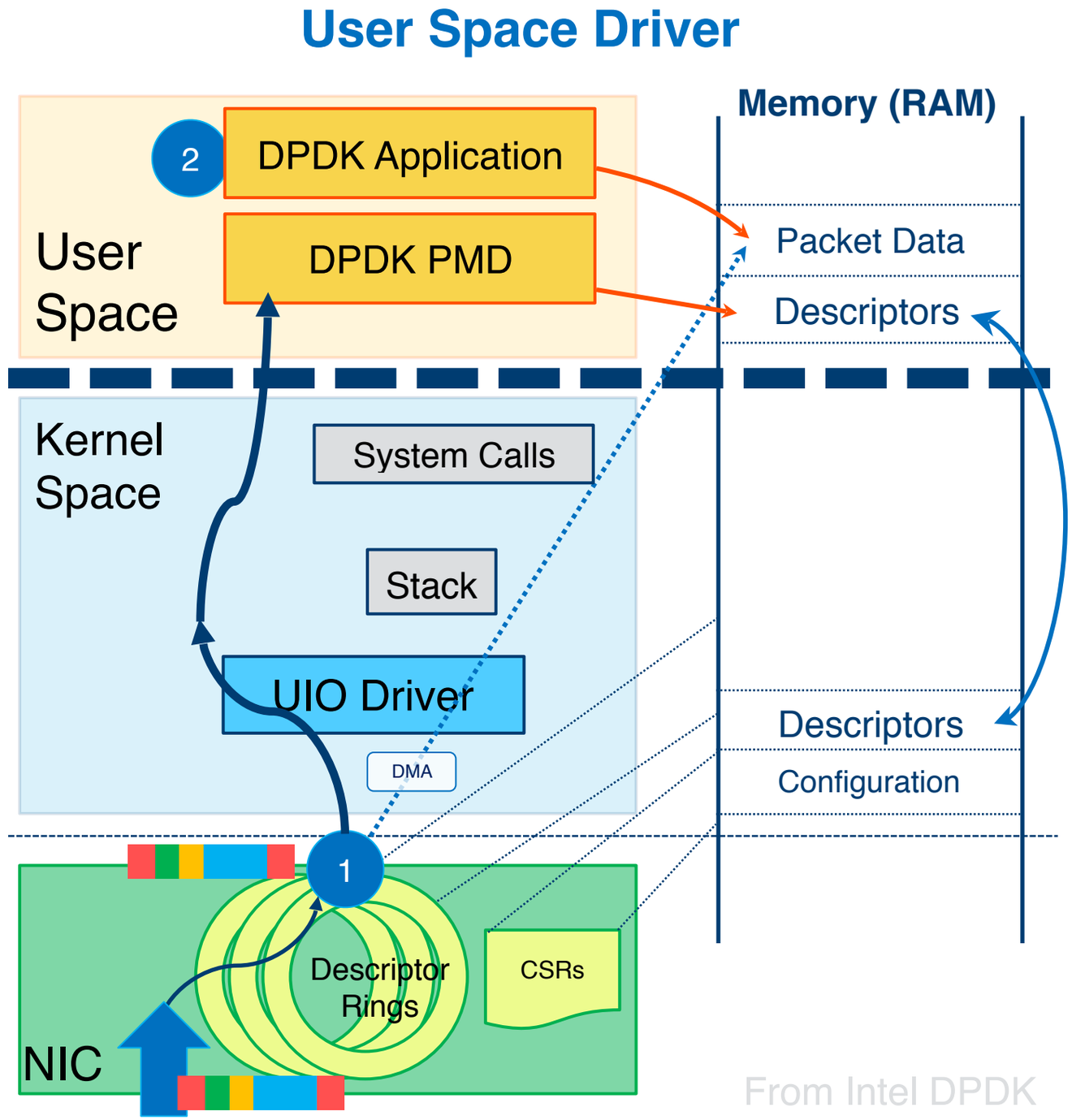
From Intel DPDK University Lecture

Kernel Bypass

User-mode Driver

- Kernel only sets up basic access to NIC
- User-space driver tells NIC to DMA data directly into user-space memory

- No extra copies
- No in-kernel processing
- No context switching



From Intel DPDK University Lecture

functions
layers

system
kernel/

processing
kernel/

memory
mm/

storage
fs/

networking
net/

human interface

user space interfaces

system interfaces
linux/syscalls.h
asm-i386/uaccess.h
copy_from_user
system files
/proc
/dev
/sysfs
cdev
cdev_add
register_chrdev
cdev_map
sysfs_ops
sys_reboot
sys_init_module

processes
sys_execve
fs/exec.c
sys_fork
sys_vfork
sys_clone
linux_binfmt
sys_nanosleep

memory access
sys_brk
sys_mmap2
/proc/self/maps

files & directories access
sys_open
sys_read
sys_write
do_path_lookup
sys_mount
sys_sync

sockets access
sys_socketcall
sys_socket
socket_file_ops

HI char devices
cdev
kmsg
sys_sysl
input_fops
snd_fops
video_fops
console_fops
fb_fops
input

virtual

drivers/base/
kobject
Device Model
kset
subsystem
subsystem_register
bus_type
class
device
device_create
class_device
class_device_create
device_driver
probe
driver_register

threads
work_struct
workqueue_struct
create_workqueue
kthread_create
kernel_thread
current
thread_info
do_fork

Virtual memory
virtually continuous memory
find_vma_prepare
vmalloc
vmlist
vm_struct
vir_to_page
mm/mmap.c

Virtual File System
file
inode
inode_operations
vfs_read
vfs_write
vfs_create
file_operations
file_system_type
get_sb
ramfs_fs_type
super_block

protocol families
__sock_create
socket
inet_family_ops
inet_create
unix_family_ops
proto_ops
inet_dgram_ops
inet_stream_ops

security
security/
linux/security.h
security
may_open
security_socket_create
security_inode_create
security_ops
selinux_ops

functional

system run
init/
kernel/
kernel_restart
kernel_power_off
load_module
module
init/main.c
start_kernel
do_initcalls
run_init_process

Scheduler
kernel/sched.c
task_struct
schedule_timeout
schedule
setup_timer
process_timeout
activate_task
rq
context_switch

logical memory
mm/slab.c
/proc/slabinfo
physically mapped memory
kfree
kmalloc
kmem_cache_alloc
kmem_cache
slab

Logical File Systems
ext3_file_operations
ext3_get_sb

protocols
proto
/proc/net/protocols
udp_prot
tcp_prot
ip_rcv
alloc_skb
ip_queue_xmit
ip_forward
sk_buff

HI subsystems
tty
mousedev_handler
oss
alsa
drivers/input/
drivers/media/
sound/

devices control

drivers/

generic HW access
pci_driver
usb_driver
pci_register_driver
pci_request_regions
usb_hcd_giveback_urb
request_region
request_mem_region
ioremap
usb_submit_urb
usb_hcd

interrupt context
timer_list
jiffies ++
setup_irq
timer_interrupt
do_softirq
tasklet_struct
do_IRQ
irq_desc

Page Allocator
mm/page_alloc.c
/proc/buddyinfo
__get_free_pages
zone
__alloc_pages
page
get_page_from_freelist
try_to_free_pages

Block devices
block/
gendisk
block_device_operations
request_queue
init_scsi
scsi_device
scsi_driver
sd_fops
usb_storage_driver

virtual network device
net_device
netif_rx
dev_queue_xmit
alloc_etherdev
alloc_ieee80211
ether_setup
ieee80211_rx
ieee80211_xmit

abstract devices and HID class drivers
console
kbd
fb_sps
mousedev
video_device

hardware interfaces

include/asm/
arch/i386/

devices access and bus drivers
ehci_irq
writew
readw
outw
inw
usb_hcd_irq
ehci_urb_enqueue
pci_read
pci_write

CPU specific
arch/i386/kernel/
start_thread
/proc/interrupts
atomic_t
system_call
show_regs
trap_init
pt_regs
interrupt
switch_to
cli
sti

physical memory operations
arch/i386/mm/
die
do_page_fault

disk controllers drivers
Scsi_Host
libata
ahci_pci_driver
idedisk_ops
ide_intr
ide_do_request
aic94xx_init

network devices drivers
drivers/net/
net
e100_open
ipw2100_open
rtl8139_open
zd1201_net_open

HI peripherals device drivers
vga_con
atkbd_drv
psmouse
ac97_driver

bridges

Synchronization
wake_up
add_timer
msleep
mutex_lock
mutex
completion
down

Memory Mapping
do_mmap_pgoff
vma_link
mm_struct
vm_area_struct

Page Cache
do_sync
address_space
pdflush

Swap
kswapd
do_swap_page
wakeup_kswapd

networking storage
nfs_file_operations
smb_fs_type
cifs_file_ops
iscsi_tcp_transport

electronics

I/O
I/O mem
I/O ports
USB controller
PCI controller
DMA

CPU
registers
APIC
interrupt controller

memory
RAM
MMU

disk controllers
SCSI
IDE
SATA

network controllers
Ethernet
WiFi

user peripherals
keyboard
cam
mouse
audio
graphics card

Networking

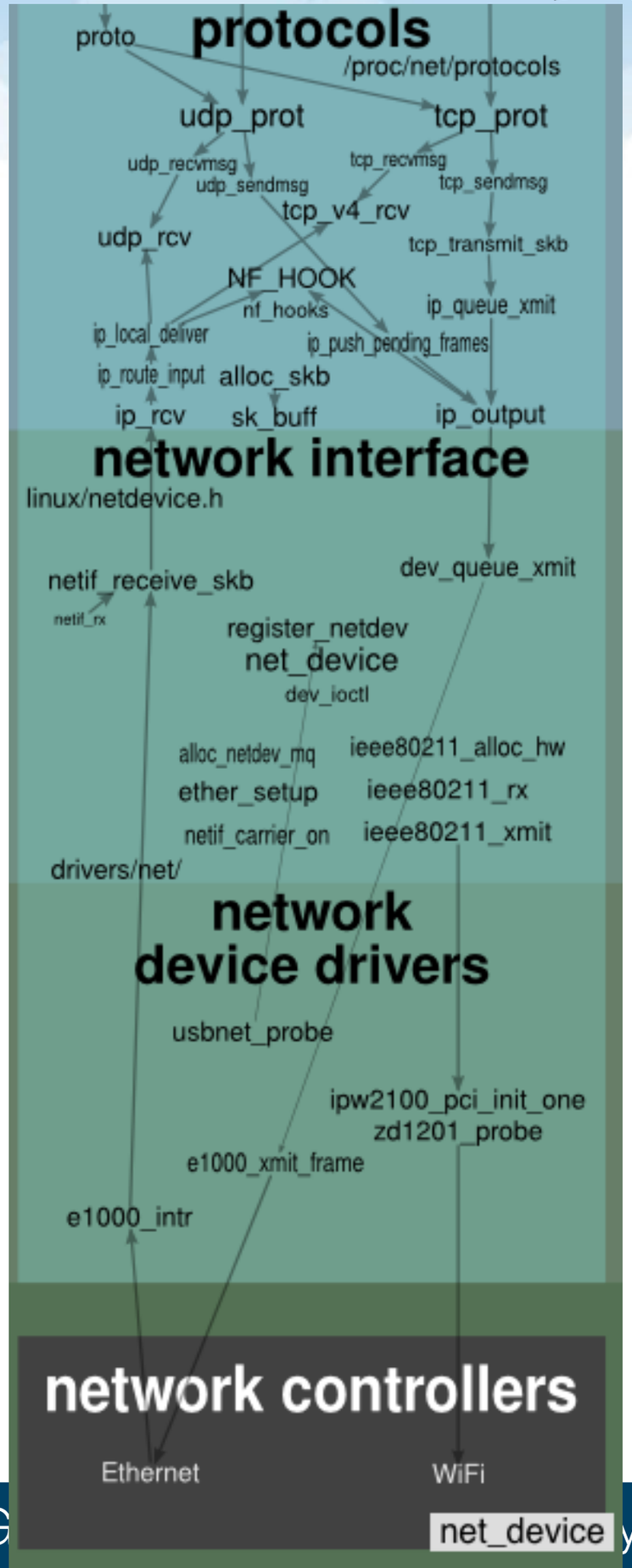
Linux networking stack has a lot of extra components

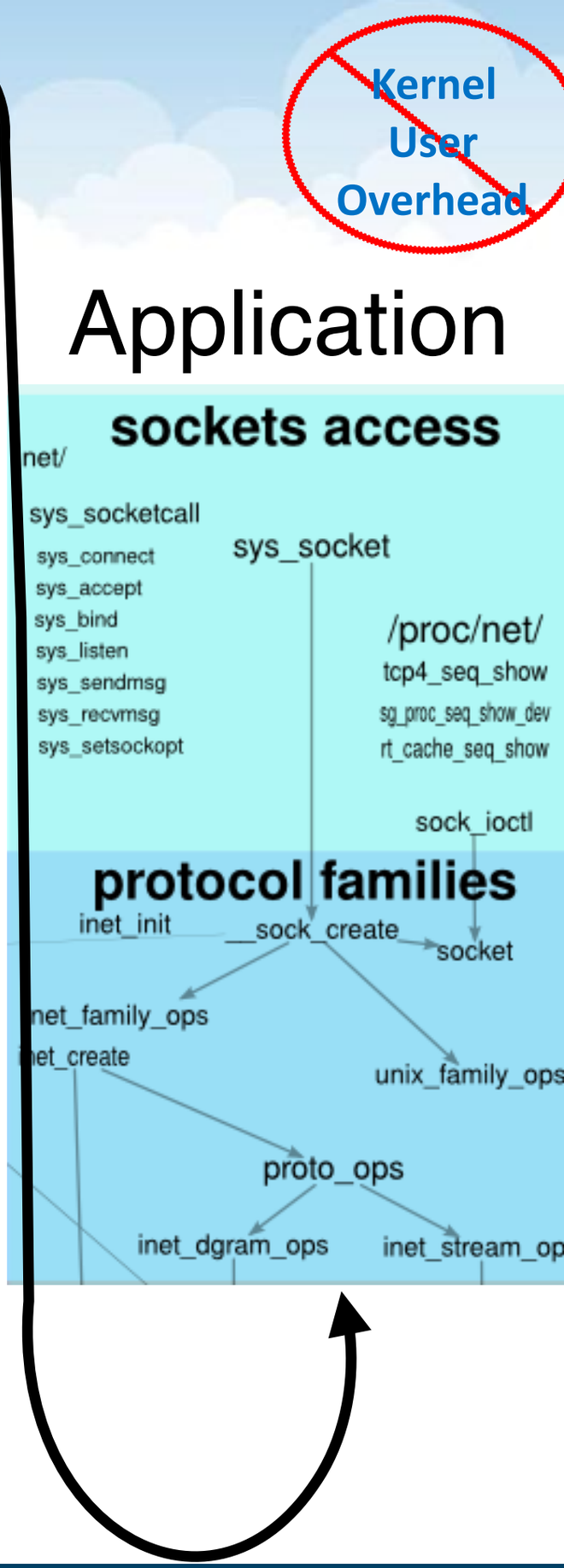
For NFV middlebox we don't use all of this:

- TCP, UDP, sockets

NFV middle boxes just need packet data

- Need it fast!

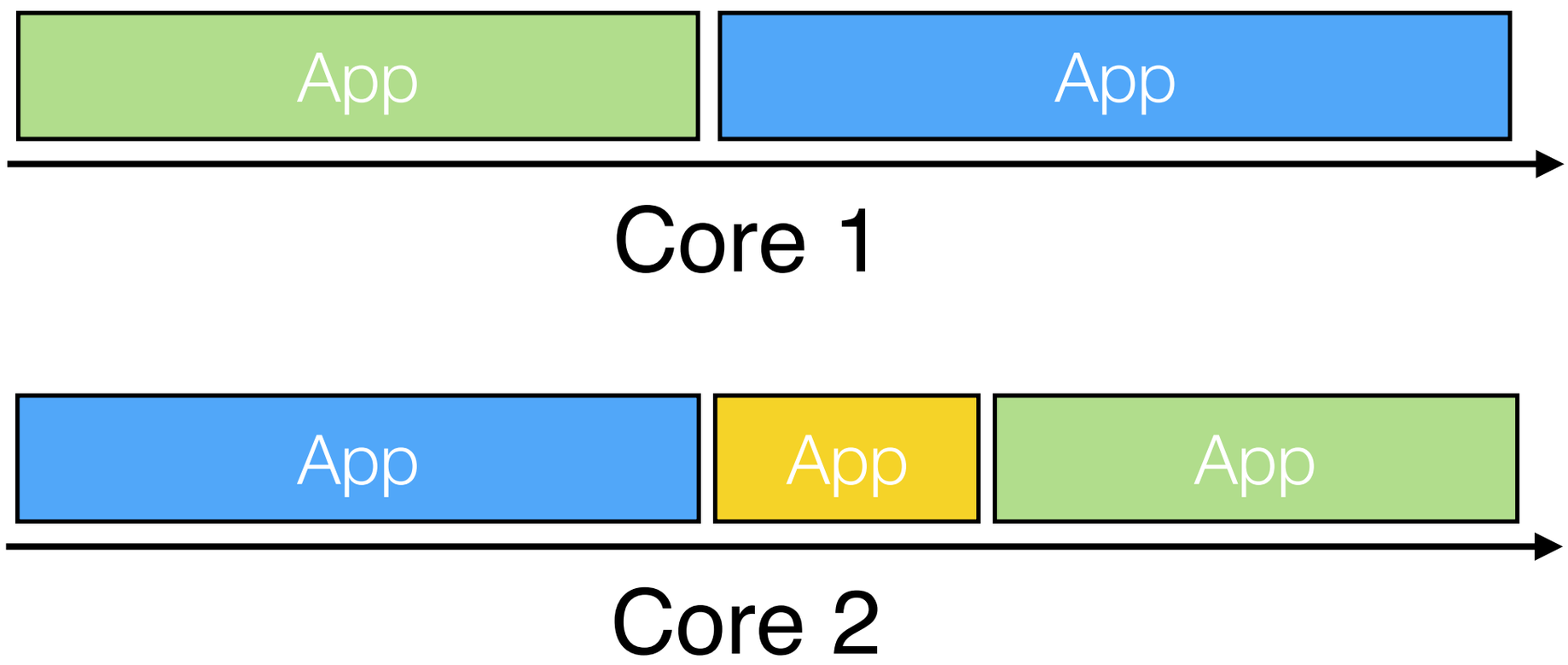




CPU Core Affinity

Linux Scheduler can move threads between cores

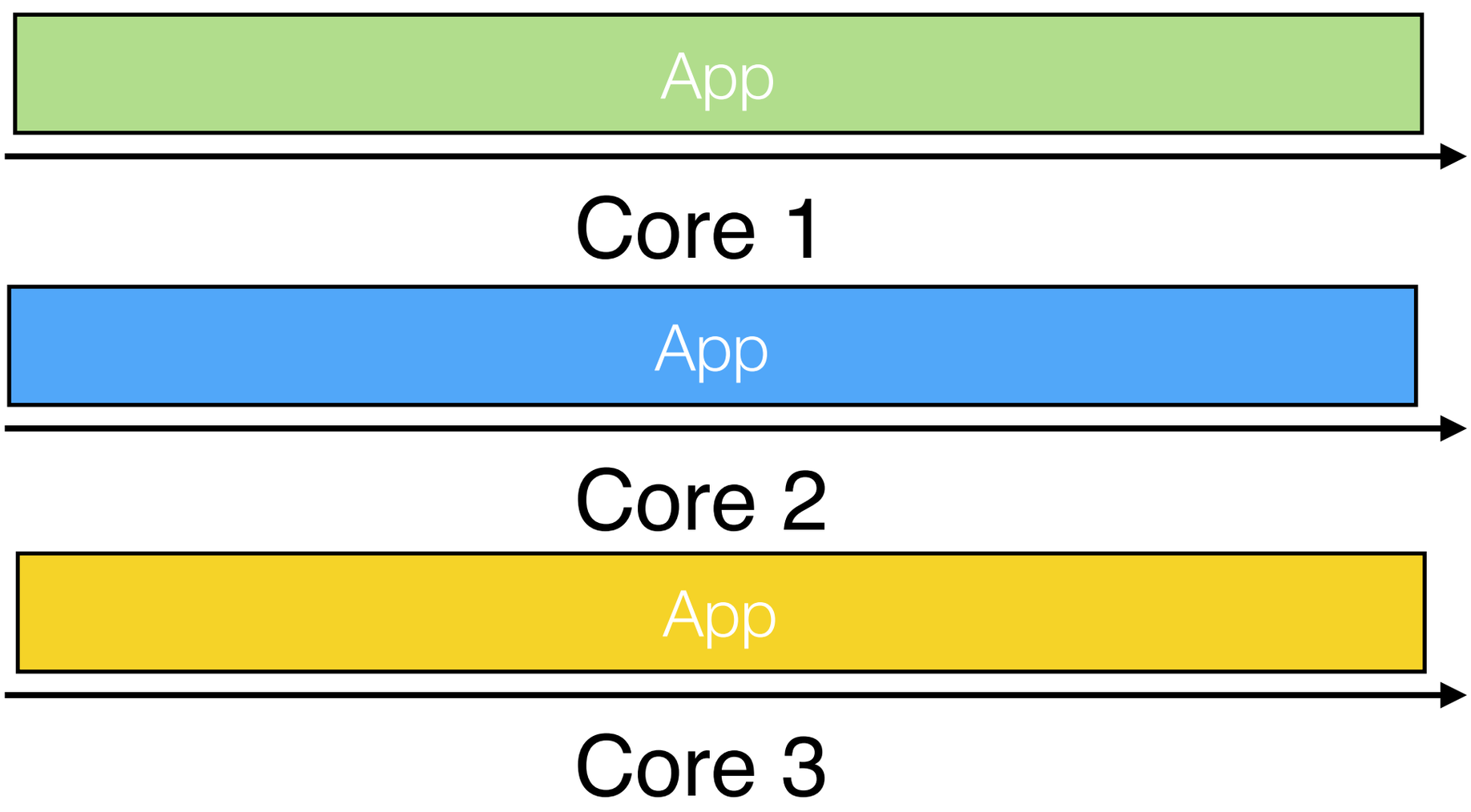
- Context switches :(
- Cache locality :(



CPU Core Affinity

Pin threads and dedicate cores

- **Trade-offs?**



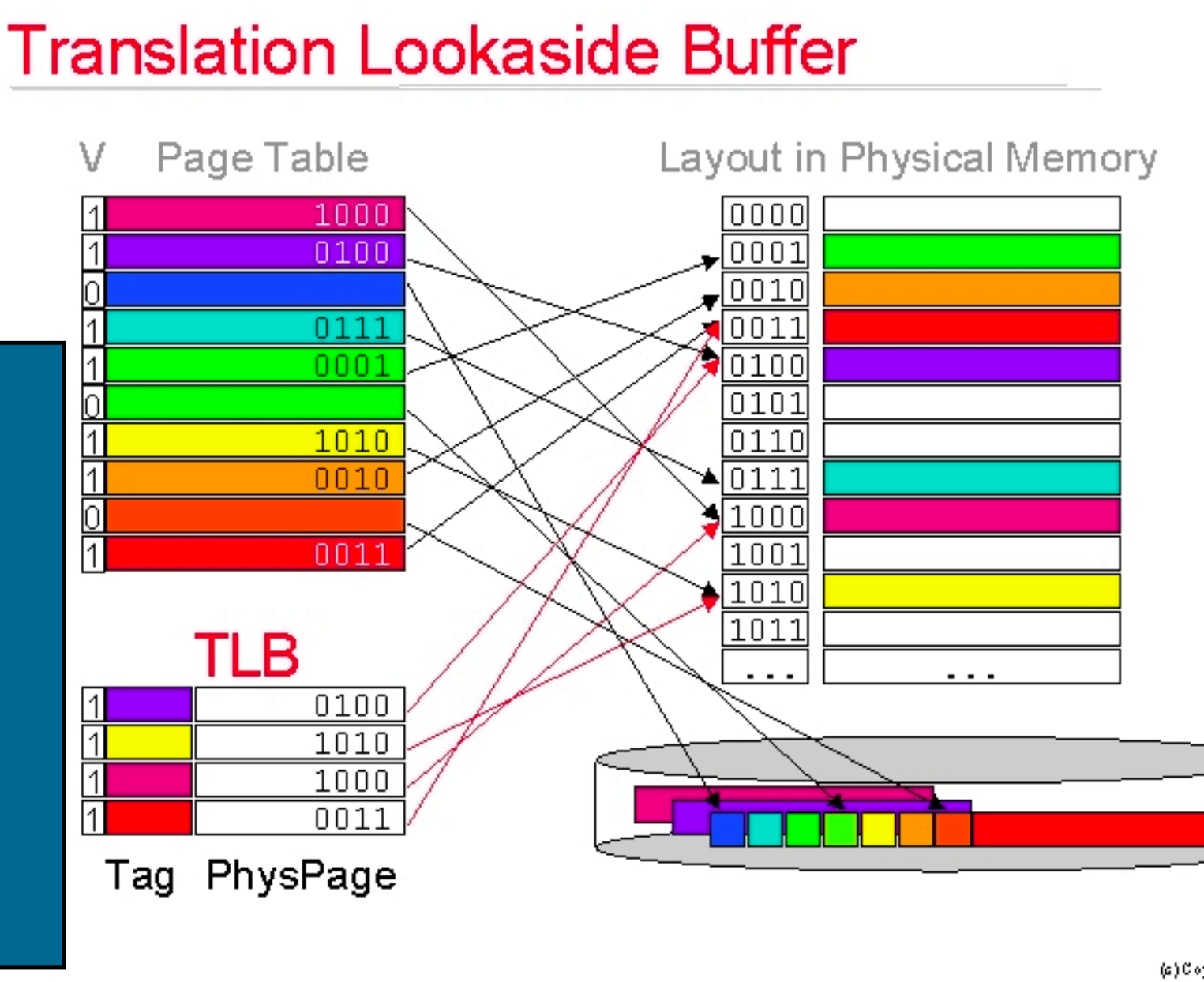
Paging Overhead

4KB Pages

- 4 packets per page
- 14 million pps
- 3.6 million page table entries every second

Packet ~= 1KB

How big is the TLB?



https://courses.cs.washington.edu/courses/cse378/00au/CSE378-00.Lec28/sld004.htm

Paging Overhead

4KB Pages

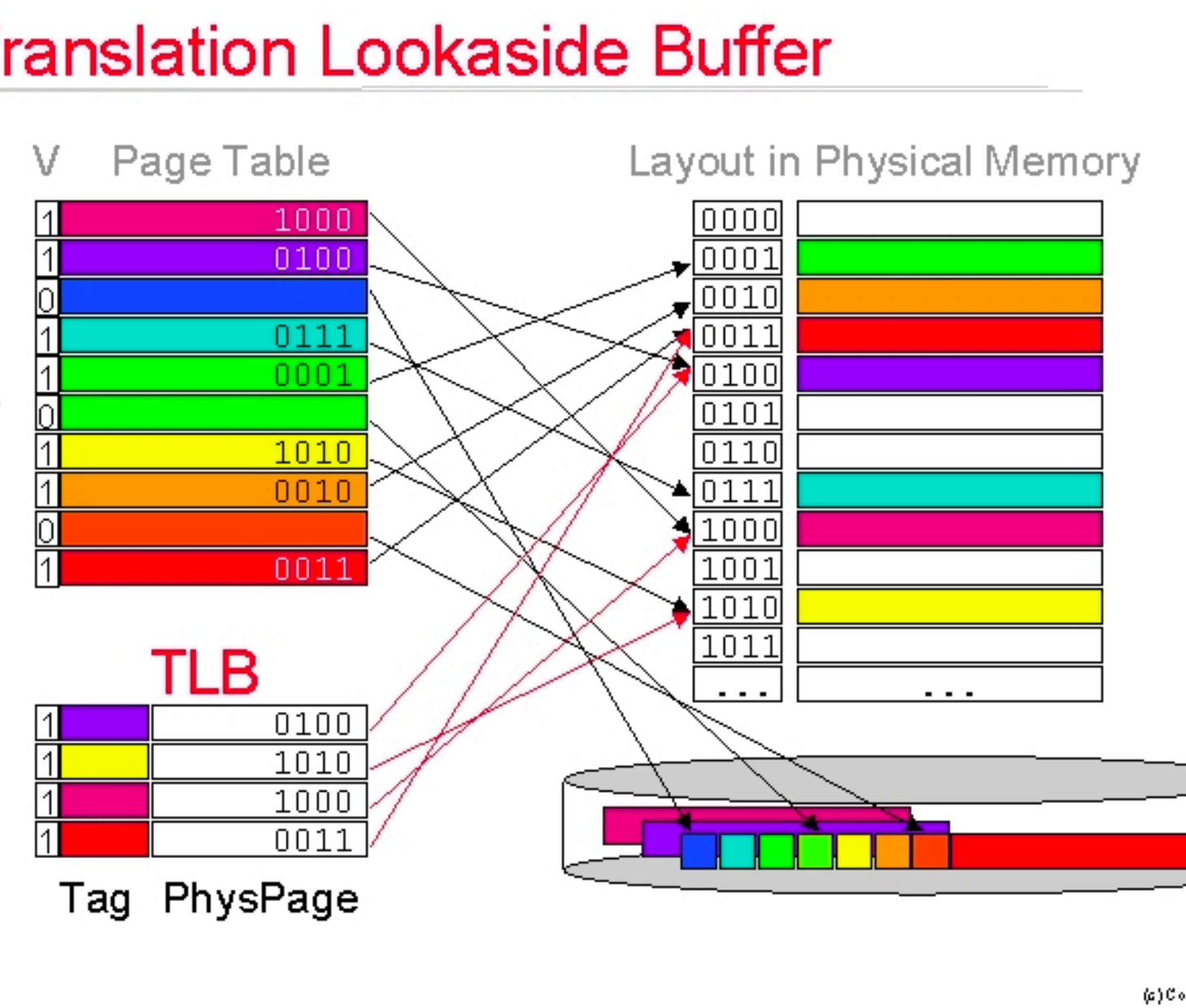
Packet ~= 1KB

- 4 packets per page
- 14 million pps
- 3.6 million page table entries every second
- Sandy Bridge: 64 entries

1GB Huge Pages

- 1 million packets/page
- 14 page table entries every second!

Trade-off?



https://courses.cs.washington.edu/courses/cse378/00au/CSE378-00.Lec28/sld004.htm

Locks

Thread synchronization is expensive

- Tens of nanoseconds to take an uncontested lock
- 10Gbps -> 68ns per packet

Producer/Consumer architecture

- Gather packets from NIC (producer) and ask worker to process them (consumer)

Lock-free communication

- Ring-buffer based message queues

Bulk Operations

PCIe bus uses messaging protocols for CPU to interact with devices (NICs)

Each message incurs some overhead

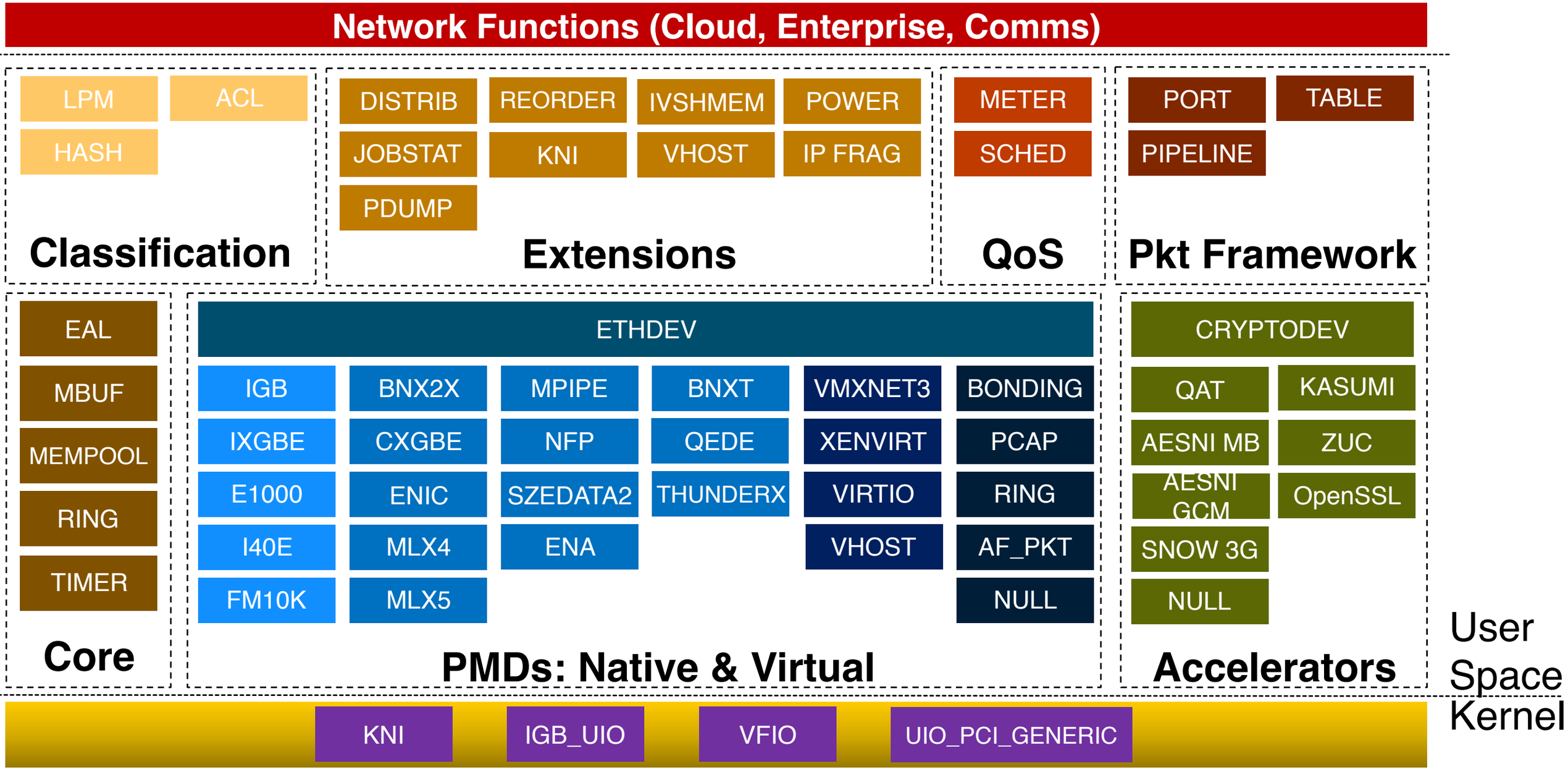
Better to make larger bulk requests over PCIe

DPDK helps batch requests into bulk operations

- Retrieve a batch (32) of packet descriptors received by NIC
- Enqueue/dequeue beaches of packet descriptors onto rings

Trade-offs?

DPDK Framework



DPDK Sample Apps

Using DPDK

Lots of examples! Great docs!

- But still fairly complex…
- http://dpdk.readthedocs.io/

Basic forwarding with DPDK

- dpdk/examples/skeleton/basicfwd.c (docs)
- Create a memory pool to store packets
- Initialize NIC ports and configure with mem pool
- Initialize ring buffer for receiving packets from NIC
- Initialize ring buffer for transmitting packets to NIC
- For each port:
  - Dequeue a batch of packets
  - Transmit each packet out the opposite port

DPDK Limitations

Barebones I/O library

- No protocol processing (e.g., TCP stack)

Focused on running a single NF

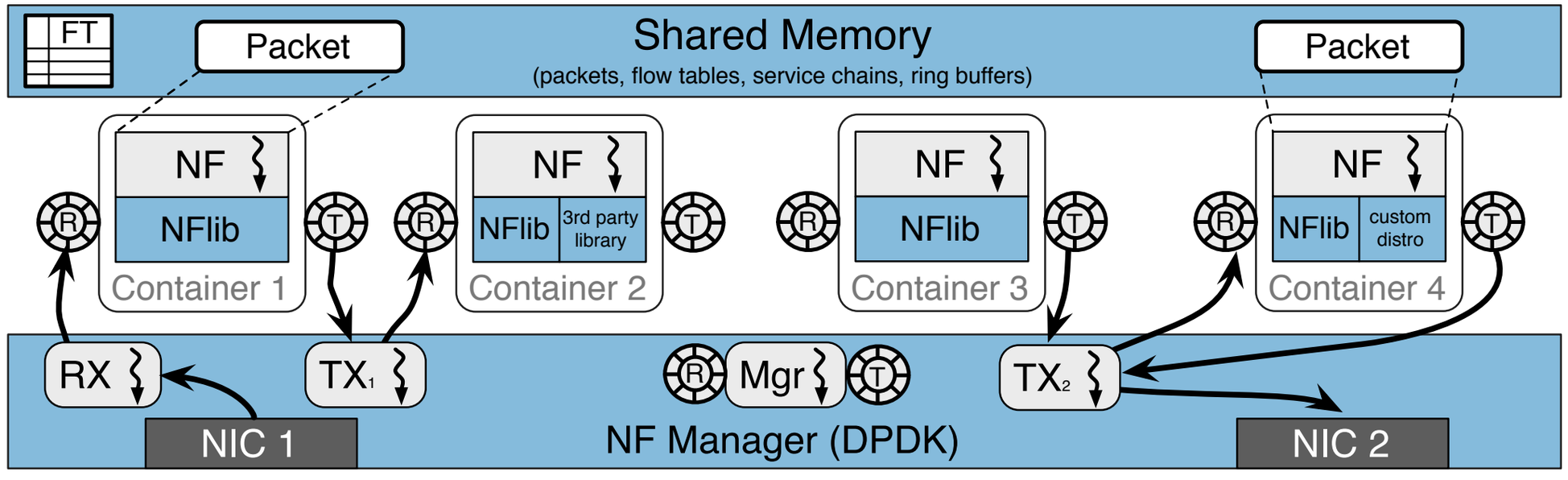
- Need to specially design NFs to be able to run multiple functions at once
- NF monopolizes entire NIC port(s)

Extensive use of polling means high CPU usage

No management layer

- addressing, reliability, auto scaling, fault tolerance

OpenNetVM* NFV Platform



Extends DPDK to offer higher level NF management

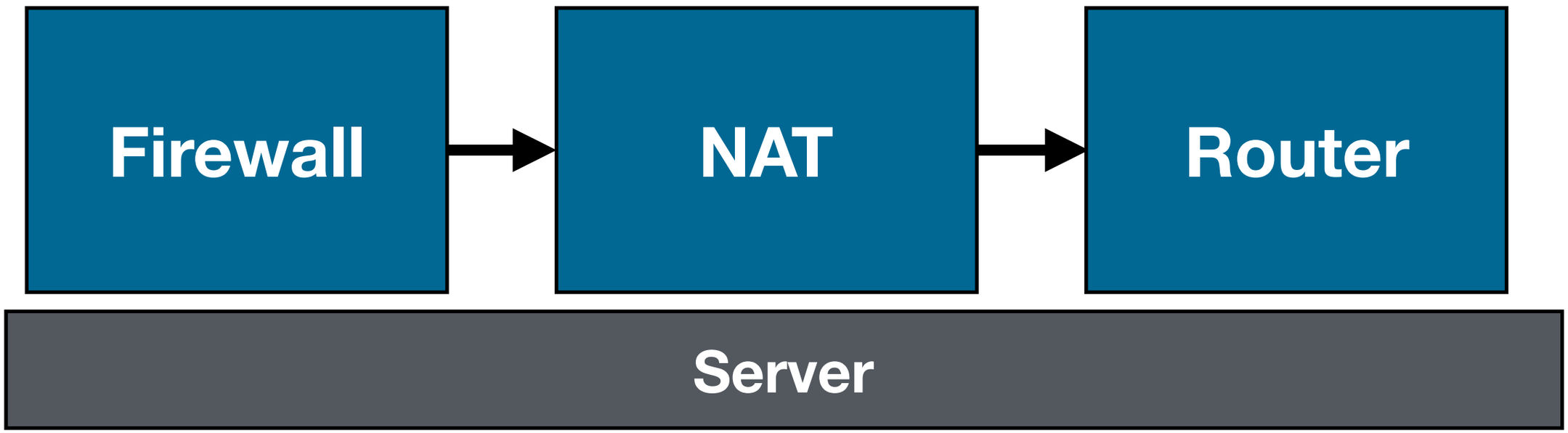
- Service chaining
- Flexible packet routing
- Load balancing
- Flow state management

*Uses containers not VMs (original system used KVM)

Service Chains

Chain together functionality to build more complex services

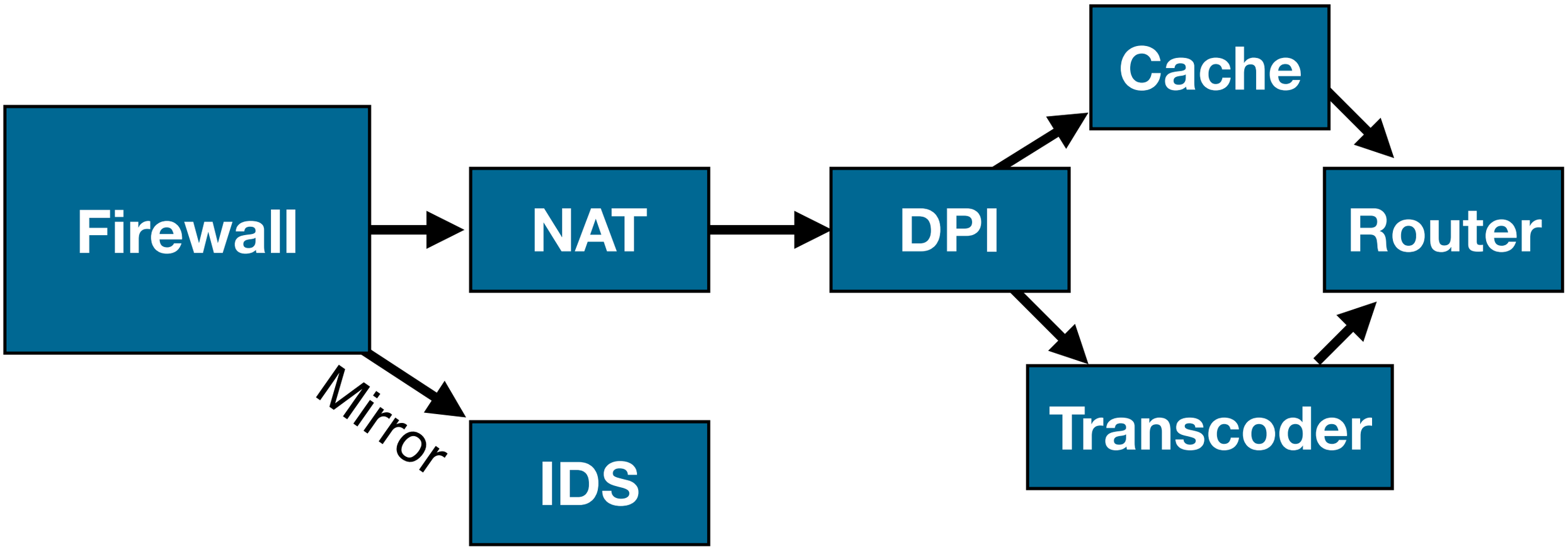
- Need to move packets through chain efficiently



Service Chains

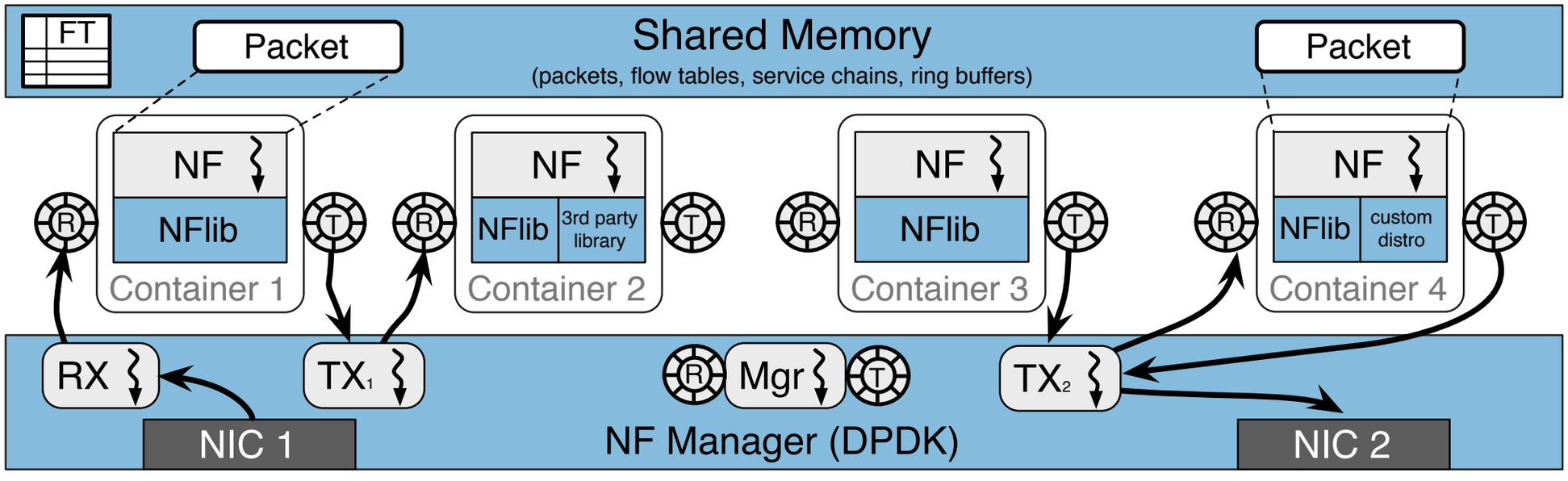
Chain together functionality to build more complex services

- Need to move packets through chain efficiently



Can be complex with multiple paths!

OpenNetVM Architecture



DPDK: provides underlying I/O engine

NFs: run inside Docker container, use NFlib API

Service chains: connect multiple NFs together

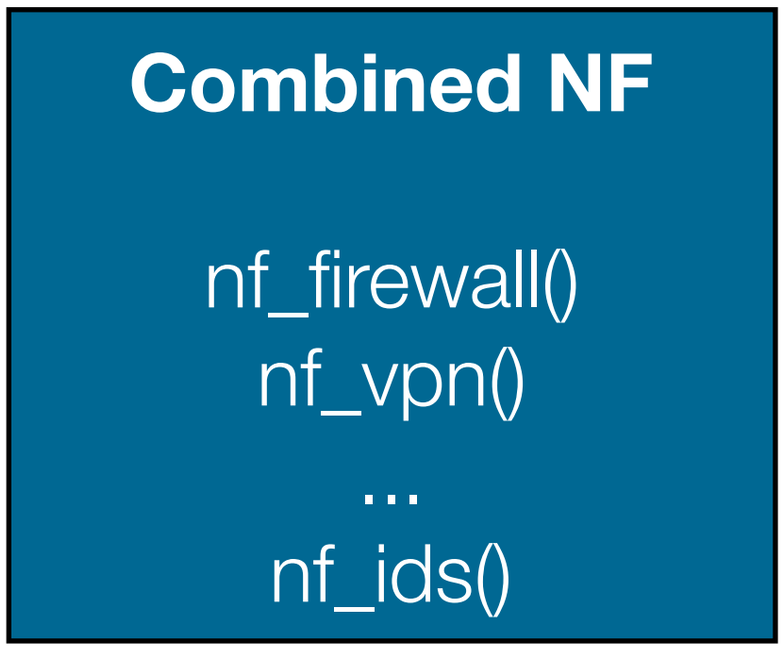
NF Manager: tracks which NFs are active, organizes chains

Shared memory: allows zero-copy packet transfer in chains

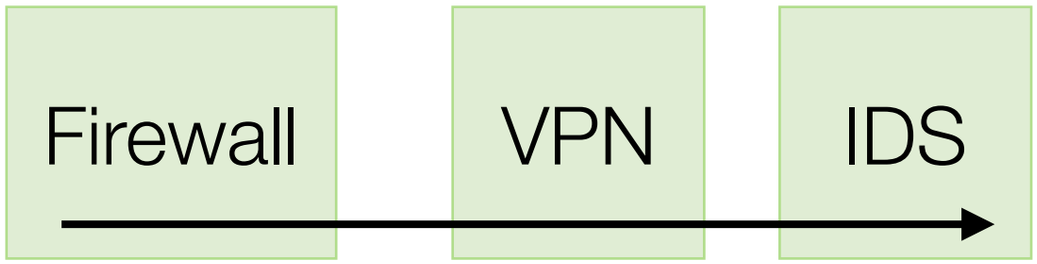
See [NSDI ’14] and [HotMiddlebox ’16] for full details

Design Principle 1

An NFV platform should focus on providing efficient IO for **modular** NFs which can be **dynamically** deployed and managed



VS



Fast calls to NFs

NFs must be compiled together and known in advance

Tight code dependencies

VM or container per NF provides isolation

Can be dynamically instantiated

Greater control over resource allocation

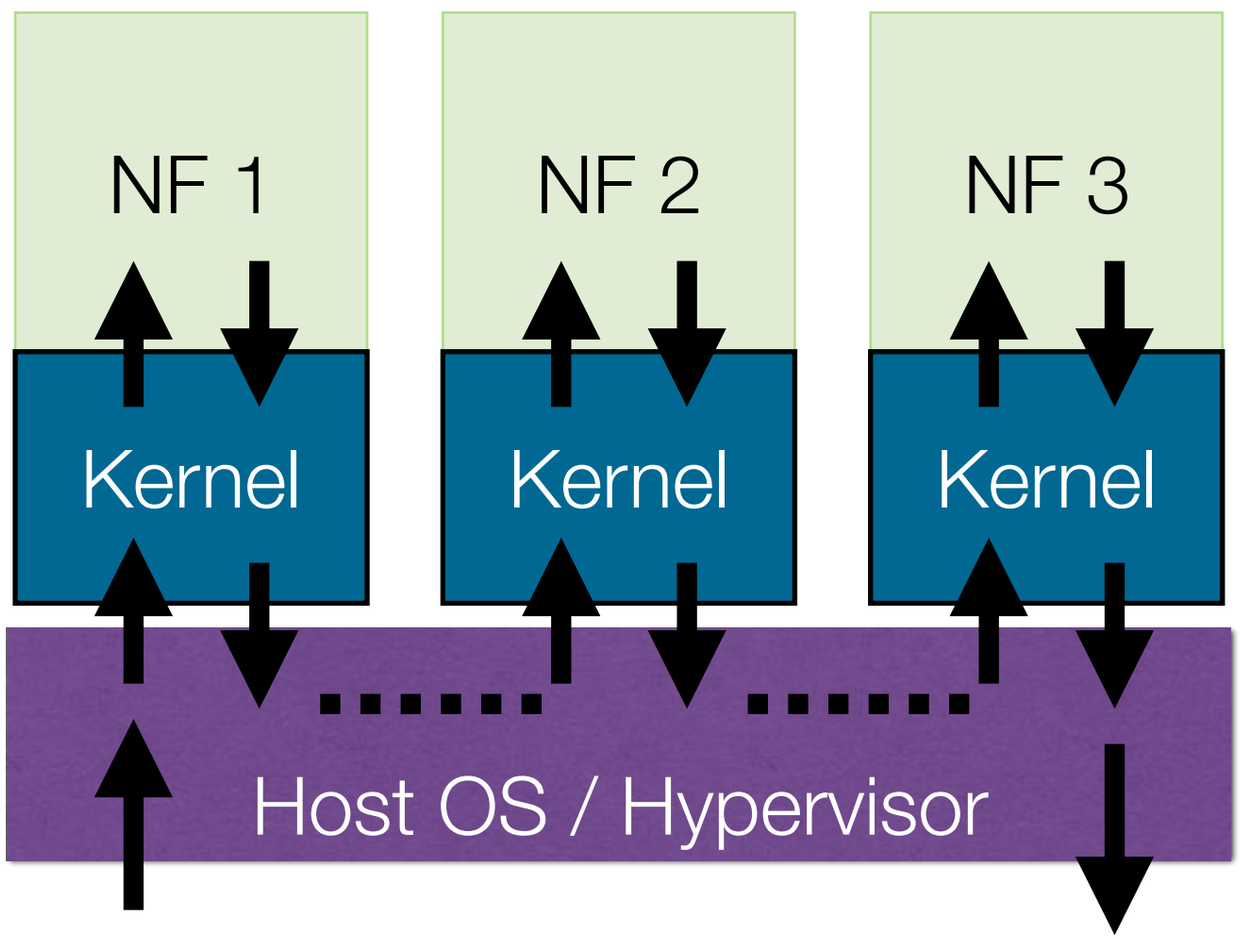
Higher cost when moving data between NFs

Packet Copies

OpenNetVM is designed for NF service chains

- Don't want to copy packet multiple times to move between NFs

Existing systems can be very inefficient!



Packet must be copied:

- when going from host to VM
- when going from VM kernel to VM guest

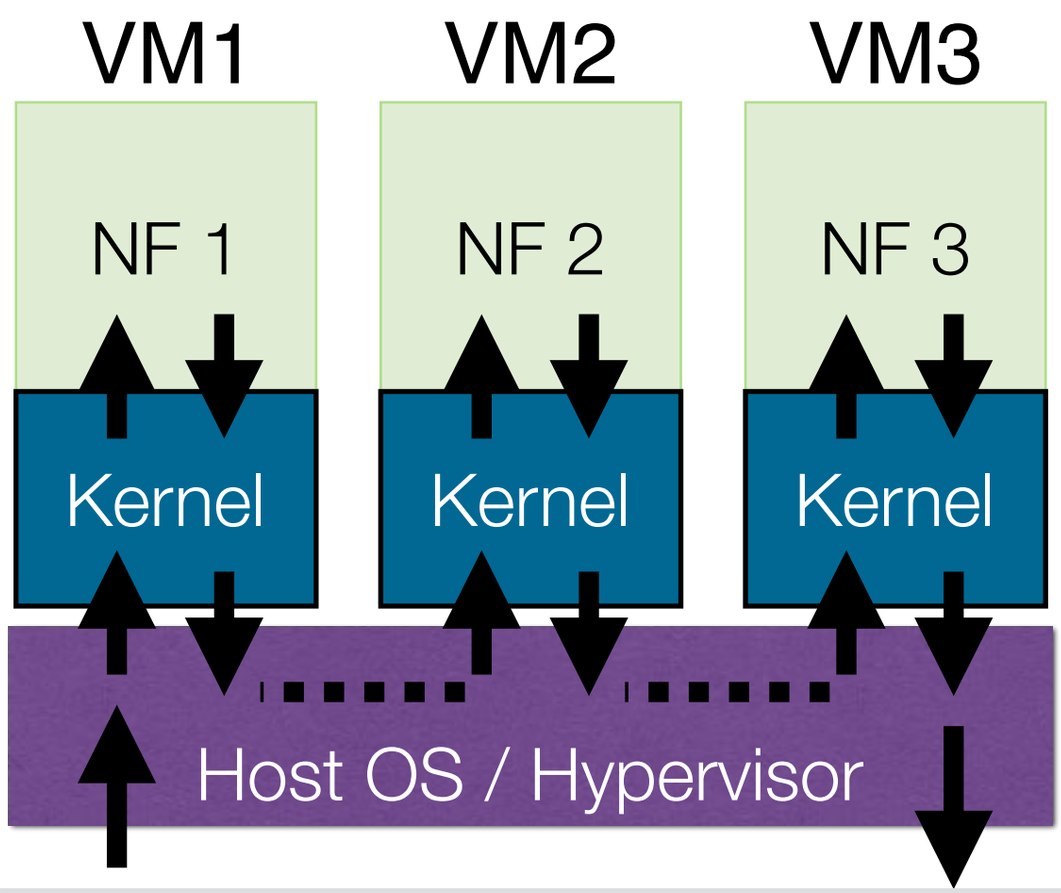
2 DMAs + 12 copies!

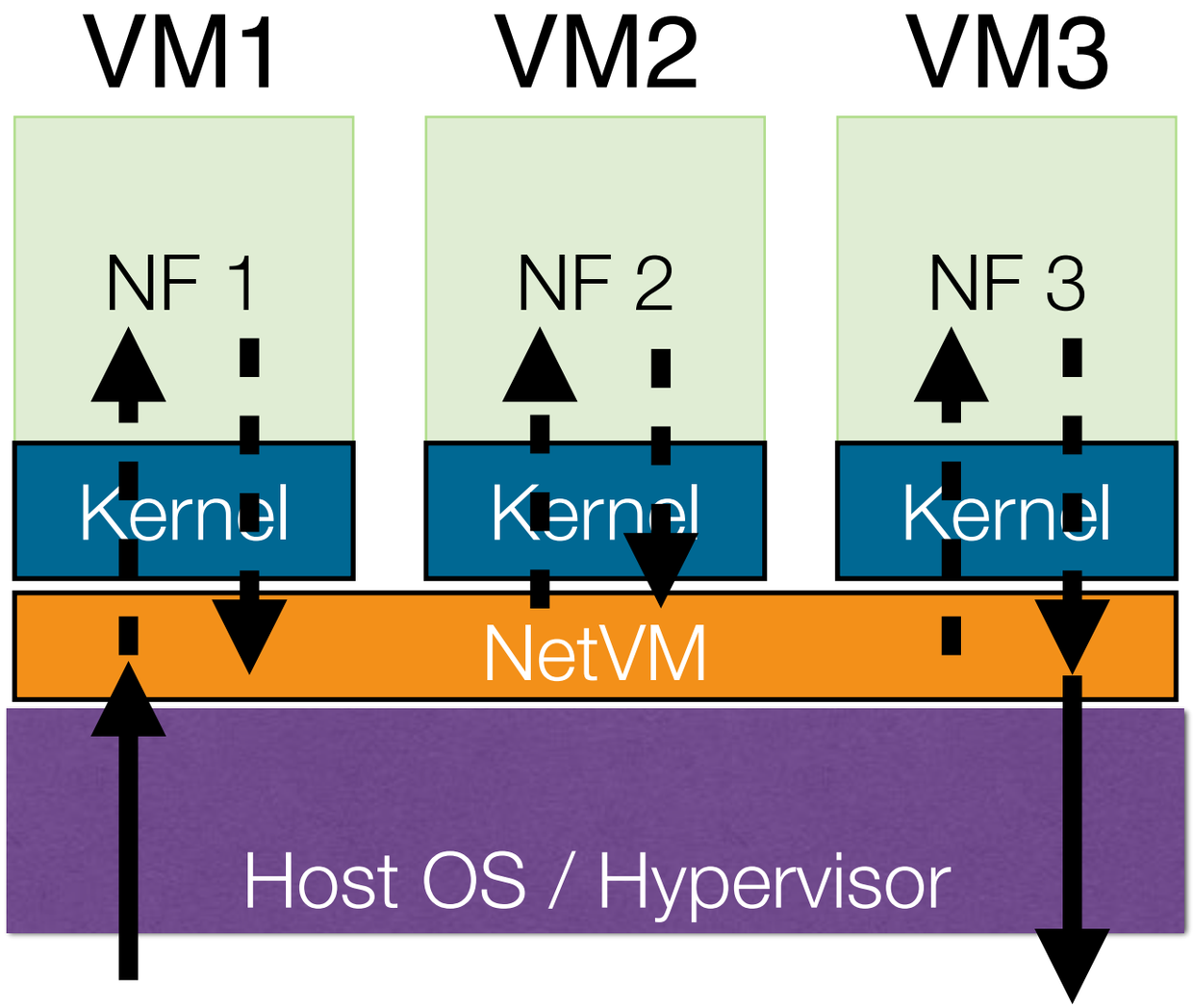
Zero Copy I/O

OpenNetVM is designed for NF service chains

- Don't want to copy packet multiple times to move between NFs

DMA into shared memory, directly accessible to NFs!



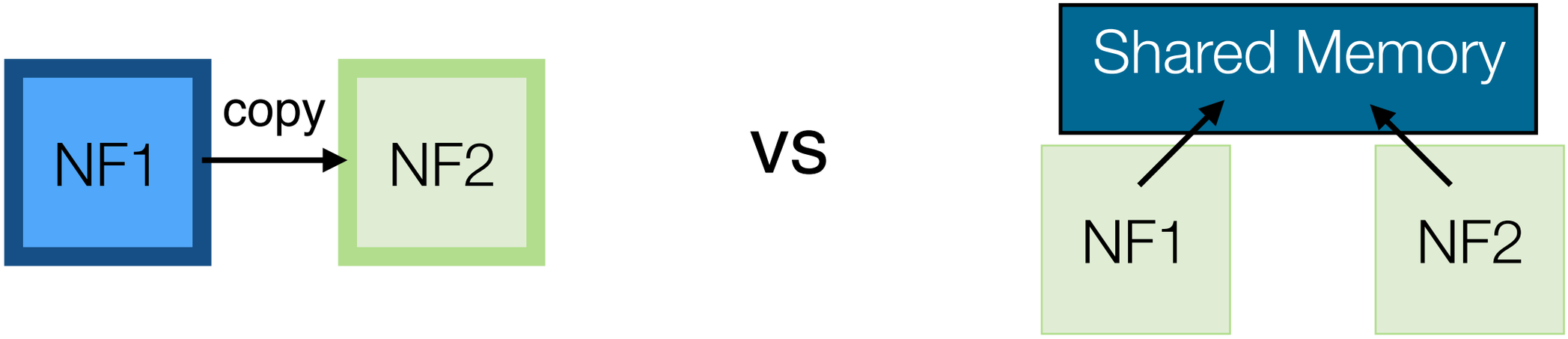


Traditional: 2 DMAs + 12 copies
NetVM: 2 DMAs, 0 copies

Design Principle 2

Eliminate abstraction layers when they incur overhead or provide protection that is not critical

NFs are developed by trusted vendors; security is less critical than performance and fault isolation



Protects against packet snooping

High cost in long chains

Eliminates copies while retaining process isolation

Light Weight VMs

OpenNetVM uses Docker Containers to run NFs

- Less overhead than full virtualization
- Very fast to start new containers (< 0.5 seconds)
- Easier to share memory

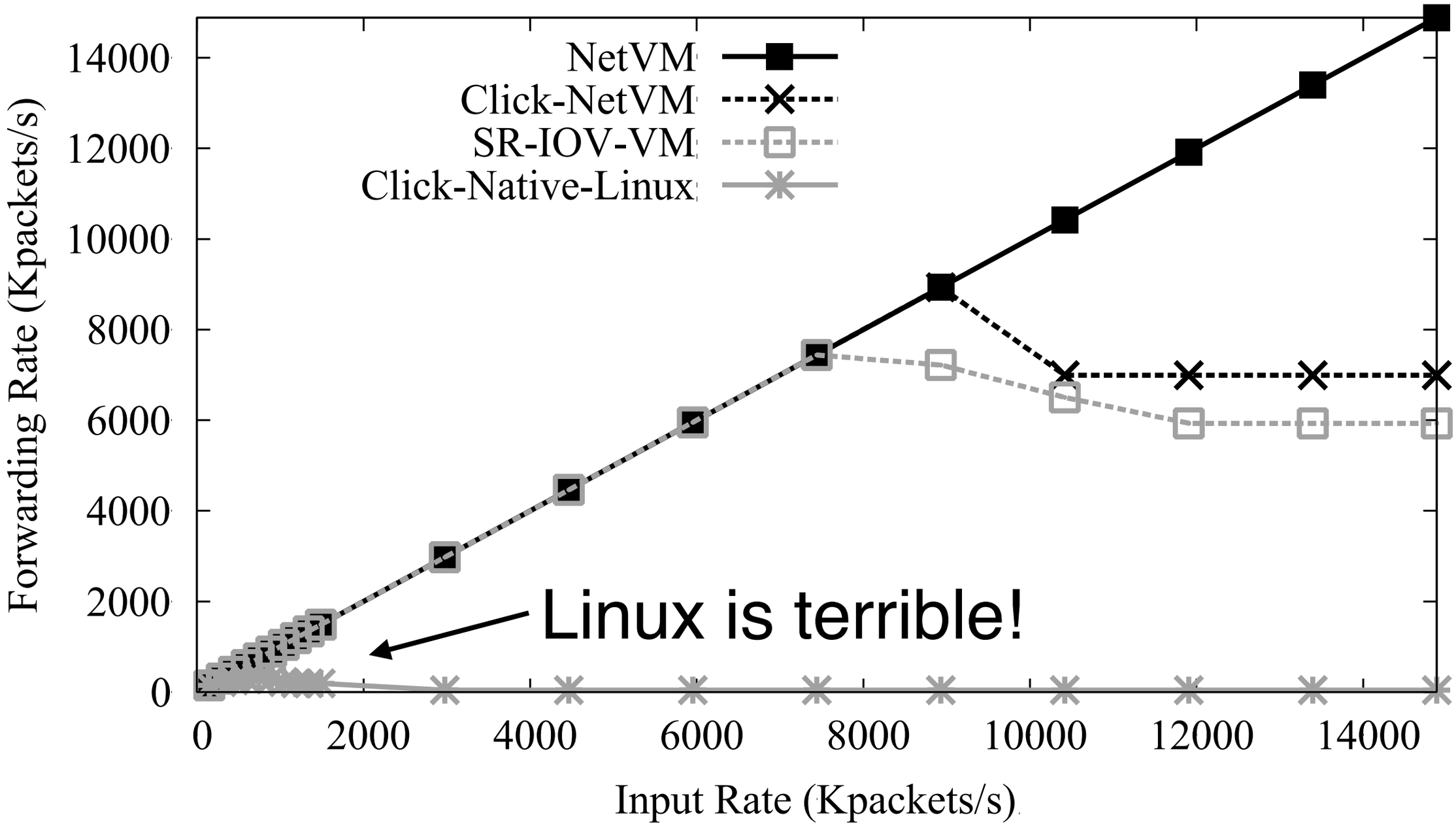
Much faster service chains compared to ClickOS

- Negligible overhead from Docker containers

Chain length:	1	2	3	6
ONVM-proc	19.1 Mpps	18.7 Mpps	18.6 Mpps	18.3 Mpps
ONVM-Docker	18.6 Mpps	18.8 Mpps	18.3 Mpps	18.4 Mpps
ClickOS	6.5 Mpps	4.5 Mpps	3.9 Mpps	---

High Speed Packet Delivery

64-byte packets, 10Gbps = 14,880,952 packets/s

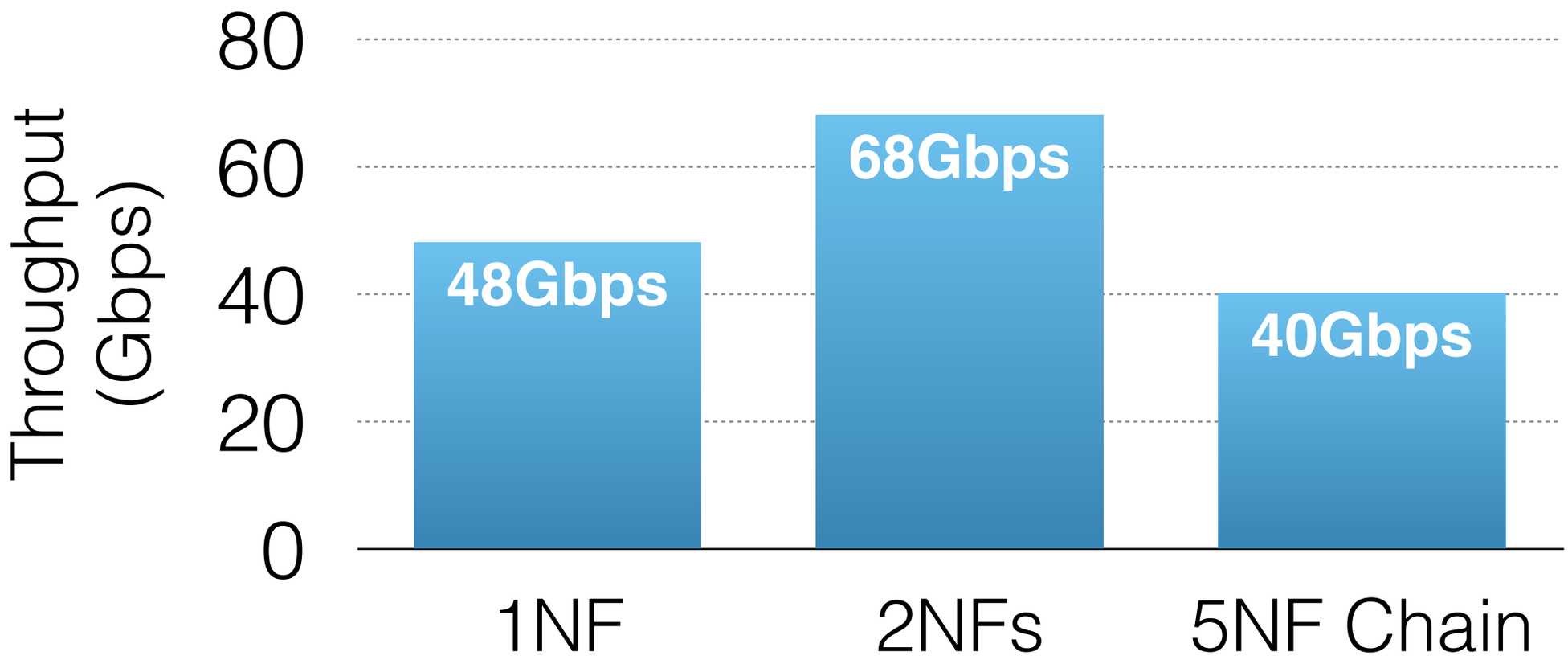


Real Traffic

Send HTTP traffic through OpenNetVM

- 1 RX thread, 1 TX thread, 1 NF = 50Gbps
- 2 RX thread, 2 TX thread, 1 NF = 68Gbps (NIC bottleneck)
- 2 RX threads, 5 TX threads, chain of 5 NFs = 38Gbps

Fast enough to run a software-based core router



OpenNetVM Development

Recent/Upcoming Features

- Bimonthly updates
- TCP stack support with mTCP
- Faster NF-to-NF communication (and uses fewer cores)
- More NFs: nDPI, Snort, L3/L7 load balancers

Interest/Users at 16 universities, several industry labs

- **NFVnice: Dynamic Backpressure and Scheduling for NFV Service Chains.** *Sameer G Kulkarni, Wei Zhang, Jinho Hwang, Shriram Rajagopalan, K.K. Ramakrishnan, Timothy Wood, Mayutan Arumaithurai, and Xiaoming Fu.* ACM SIGCOMM, August 2017.
- **Design Challenges for High Performance, Scalable NFV Interconnects.** *Guyue Liu, K. K. Ramakrishnan, Mike Schlansker, Jean Tourrilhes, and Timothy Wood.* ACM Workshop on Kernel Bypass Networks, August 2017.
- **Flurries: Countless Fine-Grained NFs for Flexible Per-Flow Customization.** *Wei Zhang, Jinho Hwang, Shriram Rajagopalan, K. K. Ramakrishnan, Timothy Wood.* ACM Co-NEXT, December 2016.
- **SDNFV: Flexible and Dynamic Software Defined Control of an Application- and Flow-Aware Data Plane.** *Wei Zhang, Guyue Liu, Timothy Wood, K. K. Ramakrishnan, Jinho Hwang.* ACM/IFIP/USENIX Middleware, December 2016.
- **Netalytics: Cloud-Scale Application Performance Monitoring with SDN and NFV.** *Guyue Liu, Michael Trotter, Yuxin Ren, Timothy Wood.* ACM/IFIP/USENIX Middleware, December 2016.
- **OpenNetVM: A Platform for High Performance Network Service Chains.** *Wei Zhang, Guyue Liu, Wenhui Zhang, Neel Shah, Phil Lopreiato, Gregoire Todeschi, K.K. Ramakrishnan, and Timothy Wood.* ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization, August 2016.
- **Towards a Software-Based Network: Integrating Software Defined Networking and Network Function Virtualization.** *Timothy Wood, K. K. Ramakrishnan, Jinho Hwang, Grace Liu, and Wei Zhang.* IEEE Networks Magazine, June 2015.
- **Virtual Function Placement and Traffic Steering in Flexible and Dynamic Software Defined Networks.** *Ali Mohammadkhan, Sheida Ghapani, Guyue Liu, Wei Zhang, K. K. Ramakrishnan and Timothy Wood.* IEEE International Workshop on Local and Metropolitan Area Networks (LANMAN), April 2015.
- **Cloud-Scale Application Performance Monitoring with SDN and NFV.** *Guyue Liu, Timothy Wood.* IEEE International Workshop on Cloud Analytics (IWCA15), March 2015.
- **SmartSwitch: Blurring the Line Between Network Infrastructure & Cloud Applications.** *Wei Zhang, Timothy Wood, K.K. Ramakrishnan, Jinho Hwang.* Usenix Workshop on Hot Topics in Cloud Computing (HotCloud), June 2014.
- **Load Balancing of Heterogeneous Workloads in Memcached Clusters.** *Wei Zhang, Timothy Wood, H. Howie Huang, Jinho Hwang, K.K. Ramakrishnan.* Usenix International Workshop on Feedback Computing, June 2014.
- **NetVM: High Performance and Flexible Networking using Virtualization on Commodity Platforms.** *Jinho Hwang, K.K. Ramakrishnan, Timothy Wood.* Usenix Symposium on Networked System Design and Implementation (NSDI), April 2014. **Extended TNSM journal version.**

ONVM Basics

Clone the repository: http://github.com/sdnfv/openNetVM

Build and install DPDK

- Included with the repository; may work with other versions

Build ONVM manager and examples

Run the NF Manager

Run one or more NFs

Send some packets!

Help me out!

I need **five** volunteers to help run openNetVM

- Find info at: **`https://github.com/sdnfv/onvm-tutorial`**
- Use nodes 3, 4, 5, 6, and 7

Connect to your server in **two** terminal windows

- user=**`tutorial`** pw=**`[redacted]`**

```
# become root
sudo -s
# change to ONVM main directory
cd $ONVM_HOME
# configure DPDK to use NICs
./scripts/setup_nics.sh dpdk
```

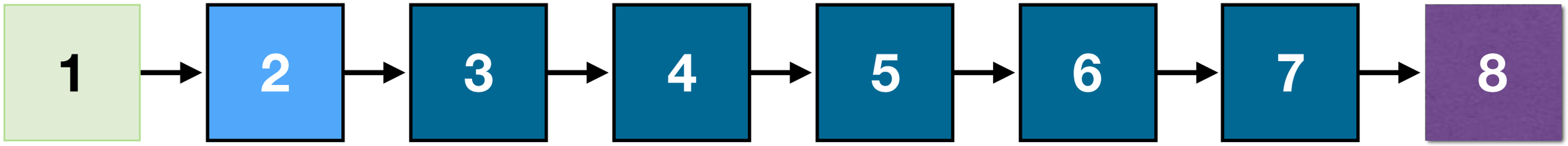
Legal stuff: *You may only use these servers for running code related to the tutorial!*

Testbed Setup

Servers are courtesy of Cloudlab.us NSF testbed

8 servers connected in a chain

Already have ONVM/DPDK installed and configured



Our goal:

- Send traffic from node 1 to node 8
- Forward all packets through 6 OpenNetVM servers
- Hopefully it will work...!

Setting up Environment

(All of this has been done already)

Export shell variables for important paths

Allocate memory for huge page region

Bind network interfaces to the DPDK driver

ONVM provides a **`setup_environment.sh`** script to automate most of these tasks

ONVM Code Structure

`openNetVM/onvm/onvm_mgr`

- NF Manager code

`openNetVM/onvm/onvm_nf`

- NFLib API used by NFs

`openNetVM/onvm/shared`

- Code shared by both Manager and NFs

`openNetVM/dpdk/`

- Git submodule with DPDK library

`openNetVM/examples/`

- Sample NF code

NF Manager

NF Manager

- Receives packets using DPDK (RX threads)
- Distributes packets based on a flow table (RX/TX threads)
- Tracks active NFs (Stats thread)

```
cd $ONVM_HOME/onvm
./go.sh  0,1,2 3 -s stdout
# usage:  ./go.sh CORE_LIST PORT_LIST
```

Core list: set of cores used for manager threads

- **0**: stats/managerment, **1**: TX thread, **2**: RX thread
- Current release assumes all cores are on same CPU socket
- Can adjust number of RX threads with compile time macro

Port list: bitmap specifying ports (e.g., 3=0b11=ports 0 and 1)

Speed Tester NF

A simple throughput tester NF

Creates a batch of packets and sends them to an NF

```
cd $ONVM_HOME/examples/speed_tester
./go.sh 4 1 1
# usage:  ./go.sh CORE_LIST NF_ID DEST_ID
# (be sure the manager is already running)
Total packets: 170000000
TX pkts per second:  21526355
Packets per group: 128
```

Send to self to benchmark manager's TX threads

(Doesn't use networking, just local packet processing)

Performance Characteristics

What affects NF performance?

Amount of computation performed per packet

- Hopefully very very little

RX threads - read packets from the NIC

- By default only use one RX thread
- One thread can handle ~7Gbps of 64 byte packets
- Two RX threads can handle ~70Gbps of "real" traffic

NFs - Process packets and pass to next NF or to TX thread

- Speed tester is best case: ~50 Million packets per second

TX threads - send packets out NIC

Lifecycle of an NF

onvm_nf_init - register NF with manager

- NF specifies its Service ID
- Manager returns an Instance ID

onvm_nf_run - start packet processing loop

- NF will poll a shared ring buffer until an RX or TX thread gives it a batch of packets

packet_handler - custom packet processing

- Called for each incoming packet
- Implements the application logic for the NF

Packet Handler

Read or write packet data

Read or write packet meta data

- Action, service chain position, flags, user data

Packet actions - specified by NF for each packet

- **TONF**: send to another NF
- **OUT**: send out a NIC port
- **NEXT**: use action in flow table
- **DROP**: discard packet

Service Types

Each NF starts with a user specified Service ID

- e.g., 1=Firewall, 2=Proxy, 3=...

Each NF is assigned a unique Instance ID

Service types are used for addressing (TONF action)

The manager uses the Instance ID to load balance across replicas with the same Service ID

- The RSS hash of the packet (based on n-tuple) is used to pick an instance in a consistent way for all packets in a flow

By default: all RX packets are delivered to Service 1

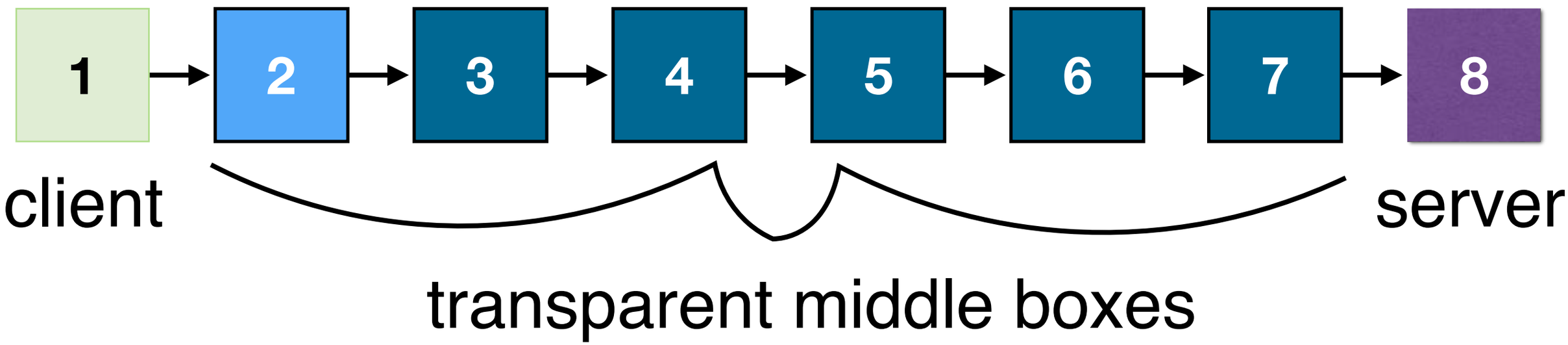
- Can be changed by modifying the default Service Chain

Bridge NF

Bridges two NIC ports

Useful for chaining servers

```
cd $ONVM_HOME/examples/bridge
./go.sh 4 1
# usage:  ./go.sh CORE_LIST NF_ID
# (be sure the manager is already running)
```



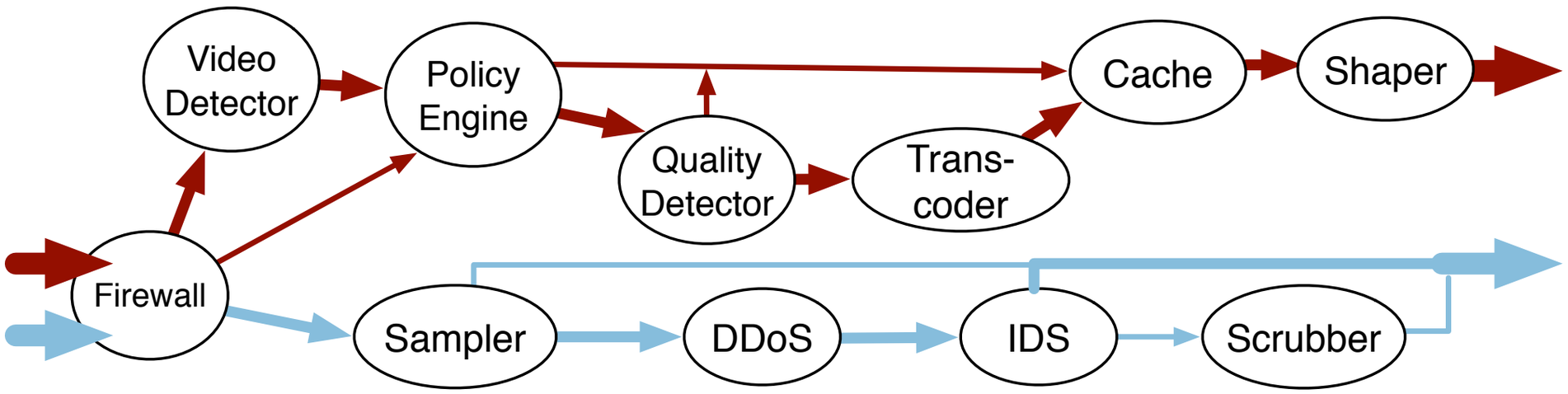
Packet Steering

NF Controlled

- NFs can specify an action (to NF, out port, drop, etc)

Manager Controlled

- Flow table in manager defines a Service Chain
- List of actions (to NF, out port, drop, etc)
- Flow table rules are installed by NFs
- SDN controller example NF uses OpenFlow to lookup service chain rules for each flow

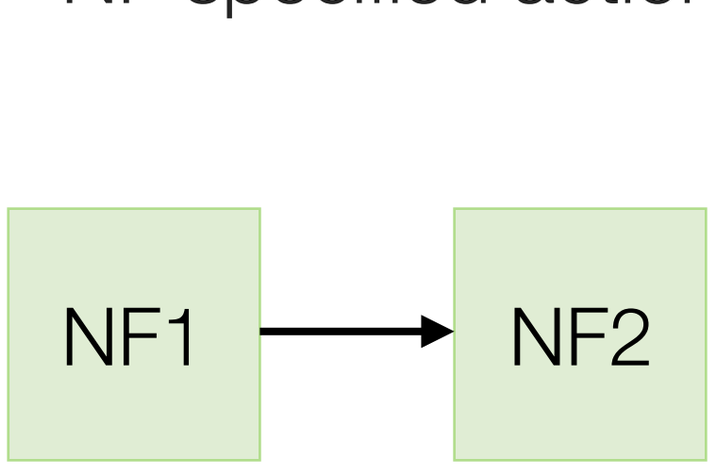


Design Principle 3

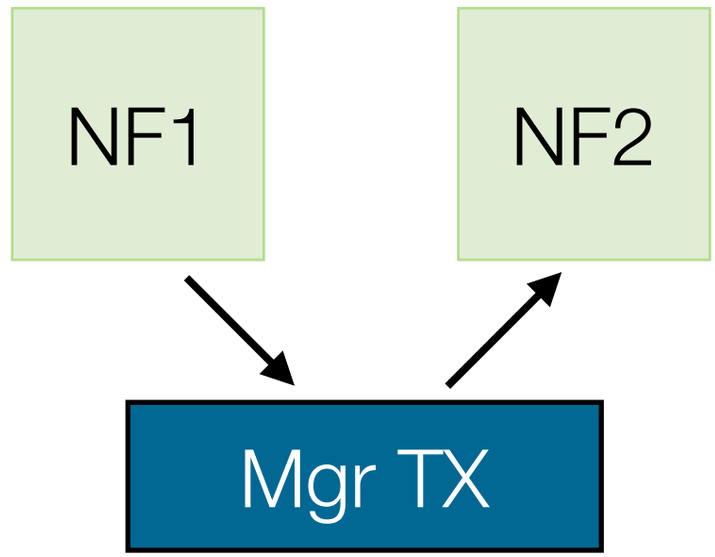
NFs should have **significant** control over how packets are routed...

But they should not have **complete** control

- NF specified action is only a hint



vs



Fast, uses minimal cores

Assumes NFs always know how to direct packets

Manager abstraction provides flexible steering, load balancing, and composition

Consumes greater resources

Flow Director

Data structure and API to define service chains

- Maps a flow (5-tuple) to an array of actions

Code is in shared/onvm_flow_dir.c

- High speed concurrent hash table based on Cuckoo Hash
- Re-uses RSS packet hash calculations to lower overhead
- Manager provides flow table implementation, but relies of NFs to configure the rules

Flow	Action 1	Action 2	Action 3	...
f1	ToNF 3	ToNF 4	Out 1	
f2	ToNF 3	ToNF 4	ToNF 7	...
...				

Design Summary

NFs must be modular and easy to compose

- Isolated processes run in Docker containers
- Simple development, deployment, and chaining

Break abstractions when necessary

- Bypass OS and network stack to get raw packets
- Shared memory between processes for faster chains

Balance control across the hierarchy

- SDN controller provides flow table rules
- NFs can make custom decisions based on individual packets
- Manager can override decisions if necessary

THE GEORGE WASHINGTON UNIVERSITY

open source code at **http://sdnfv.github.io**

UCR

NFV as Middleware

"[Middlware 2017] is a major forum for the discussion of innovations and recent scientific advances of middleware systems with a focus on the ***design, implementation, deployment, and evaluation of distributed systems, platforms and architectures for computing, storage, and communication****"*

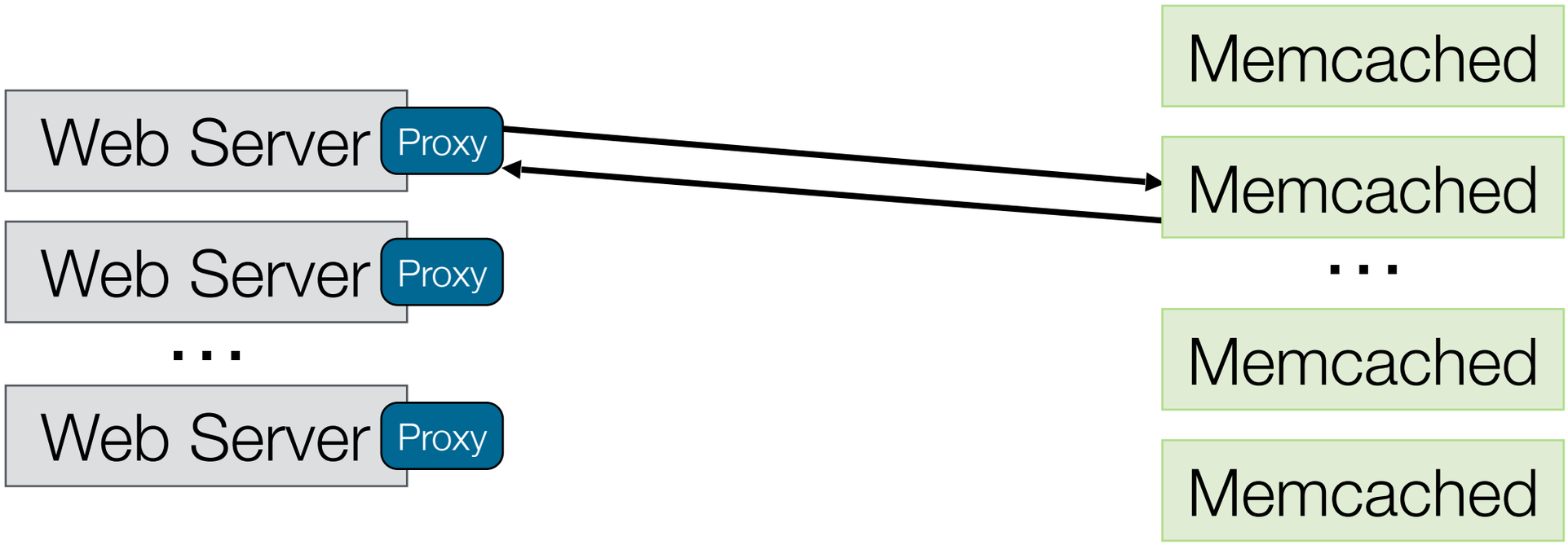
Middleboxes are an appealing form of middleware

- Completely transparent to end points
- Can perform arbitrary transformations, monitoring
- Can deploy without modifying end hosts

Load Balancing Middleware

Load balancing proxies for memcached

- Commonly deployed middleware

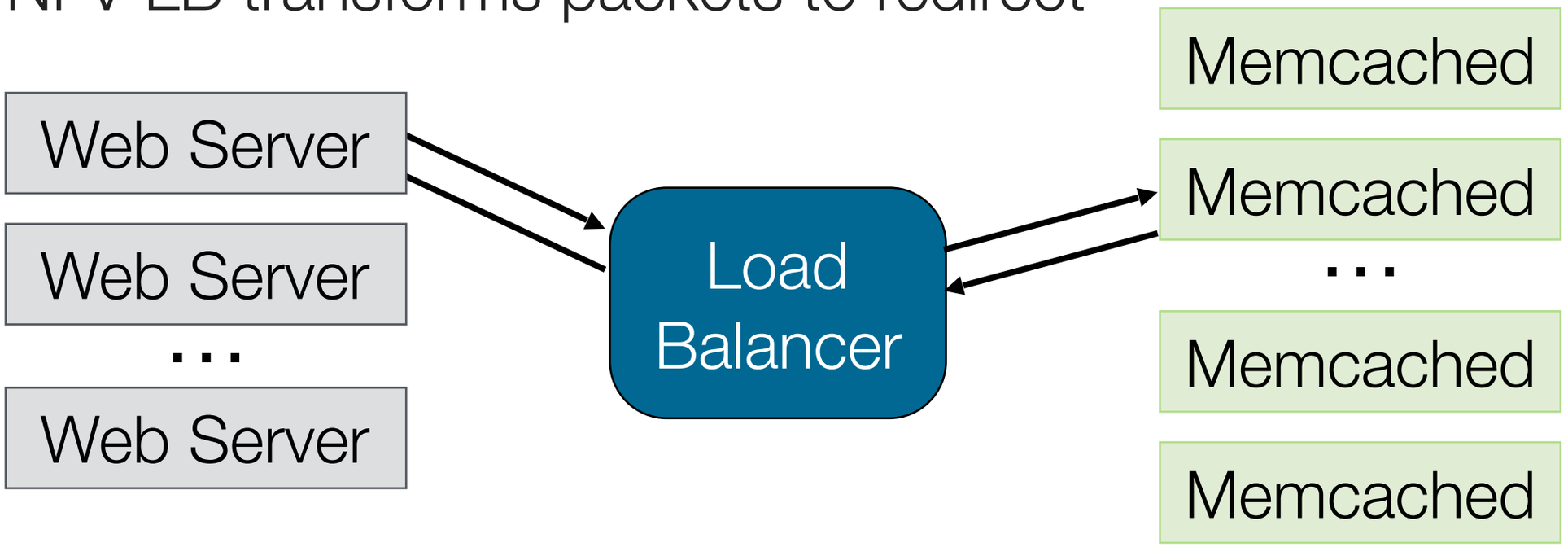


Web application interacts with memcached servers via proxy middleware that balances requests across multiple servers and handles key replication

NFV Load Balancing

Use NFV to build a centralized load balancer

- Web servers are oblivious to LB
- NFV LB transforms packets to redirect

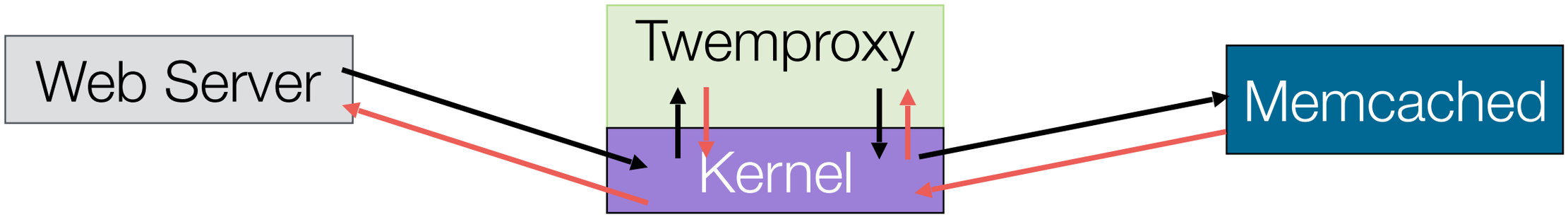


Load Balancer must have efficient
control path (to monitor workload and tune replication)
and **data path** (to forward to the best replica)

Load Balancing @ Twitter

TwemProxy

- Uses two connections with web server and memcached nodes

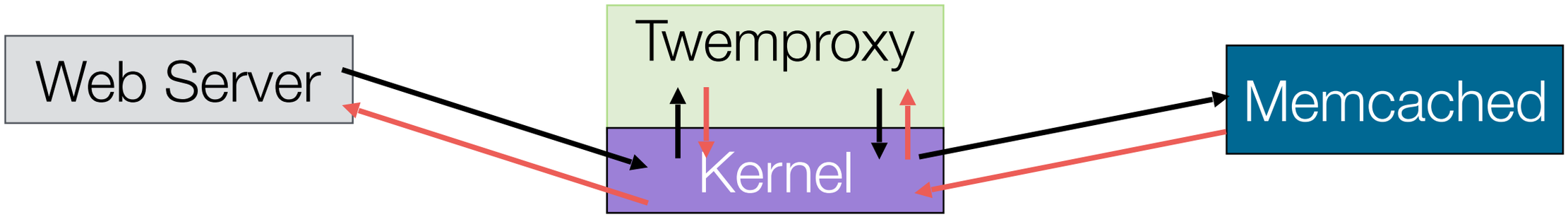


- 4 packet copies + 4 DMAs

NetKV Scalable Load Balancer

TwemProxy

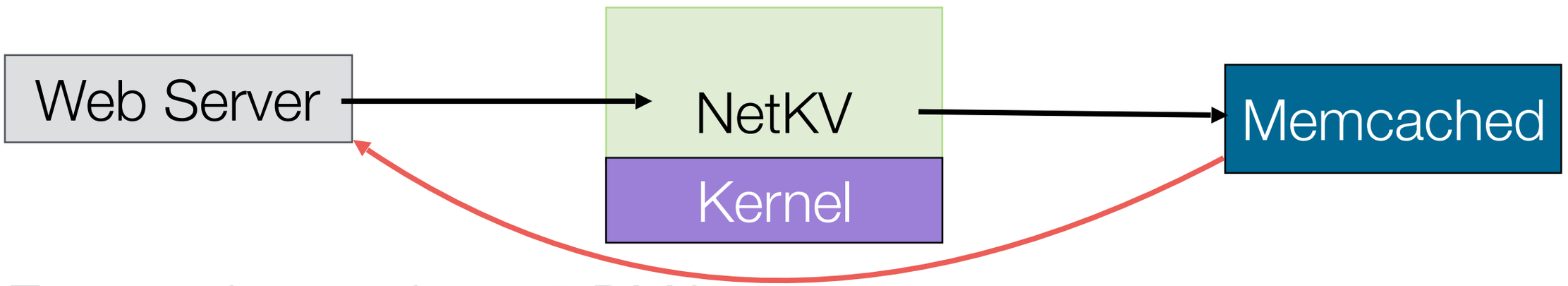
- Uses two connections with web server and memcached nodes



- 4 packet copies + 4 DMAs

NetKV uses openNetVM for fast I/O

- Bypass kernel and rewrite packet to redirect, UDP only



- Zero packet copies + 2 DMAs
- Reply directly to web server

NetKV Overview

ICAC 16

Handle special cases

Write Handler

Local Cache

Hot Writes

Very Hot Keys

Detect and manage hot keys

Replication Engine

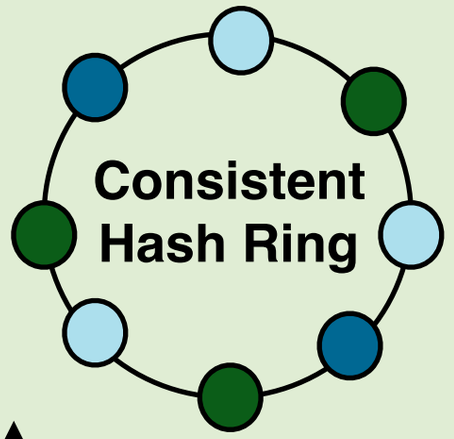
Hot Key Detector

Separate lookup paths for hot and cold keys

Replica Table

Key	Replica Factor

Hot



Cold

Dispatcher: OpenNetVM-based I/O platform

Incoming Requests

Data Center Qs?

Why is my multi-tier web application running slowly?

Which types of requests are fast or slow?

What is my most popular content?

Which tier is my bottleneck?

Where is network congestion affecting performance?

How to understand the performance and behavior of these large scale systems?

Data Center Qs?

Why is my multi-tier web application running slowly?

Which types of requests are fast or slow?

What is my most popular content?

Which tier is my bottleneck?

Where is network congestion affecting performance?

very efficiently

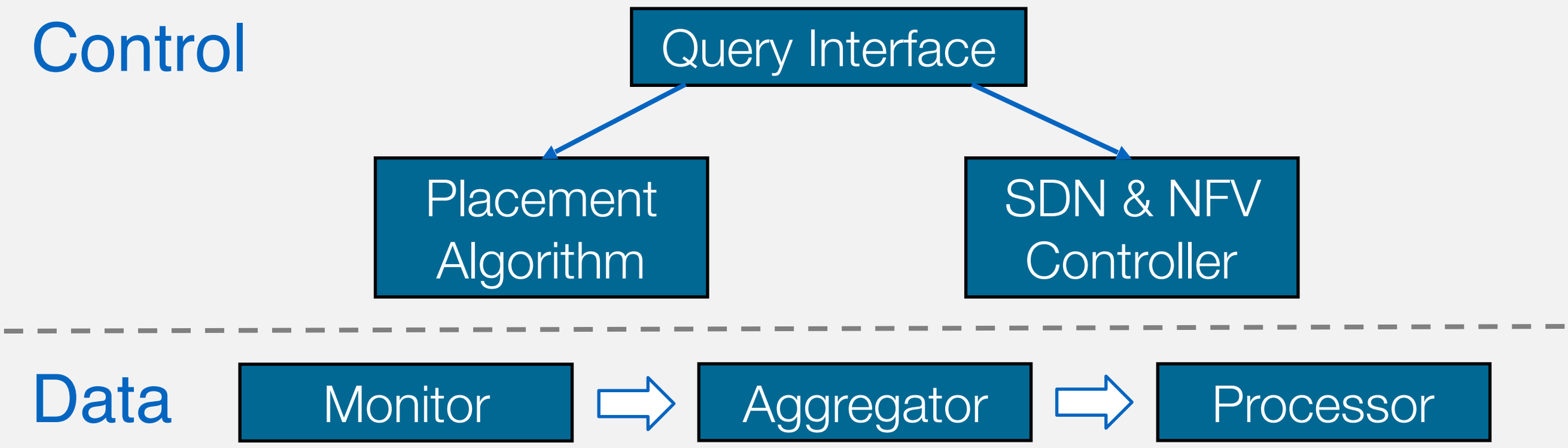
How to understand the performance and behavior of these large scale systems?

NetAlytics

Use SDN **control plane** to redirect data that matches monitoring queries

NFV **data pipeline** monitors packet flows unobtrusively and analyzes data in real-time.

Analytics can then drive cloud management systems



NFV Middleware Limits

Challenges for using NFV as middleware

Little semantic information is sent along with packets

- Per-packet processing is very low level
- May need to do byte stream reconstruction, TCP or application level protocol processing, etc.

Performance challenges

- Even if I/O platform can handle millions of packets per second, NFs may not be able to
- Can be very resource intensive

Reliability and security concerns

- Adding a server in line may cause single point of failure
- WAN traffic is often encrypted

SDN / NFV Summary

SDN: a centralized network control plane

- Routing, control over new flows as they arrive

NFV: efficient software data plane

- Per packet analysis and transformation

DPDK: high performance user-space network I/O

OpenNetVM: service chaining and NF management