Distributed Systems CS6421

Networking: SDN and NFV

Prof. Tim Wood

SDN + NFV

Networks are changing

- Trying to achieve the same level of customization, flexibility, and automation found in the cloud

Software-based Networks

- SDN: Software Defined Networking - control plane
- NFV: Network Function Virtualization - data plane

Software Defined Networks: Overview

Adapted from slides by

K. K. Ramakrishnan, UC Riverside

(with thanks to many people's material that he re-used: David Koll, Univ. of Goettingen, Germany, Jennifer Rexford, Princeton, Nick Mckeown, Stanford and others).

Cloud Scalability

"The average cloud environment might have 50 dedicated servers to one admin, and what you really need to get to is 500 servers to one admin, or what happened in the case of Microsoft, 10,000 servers. Without automation we don't have speed and scale - the very reason we want to go to the cloud." (Microsoft)

- Virtualization and automation software helped solve these problems for the cloud…
- What about the network?

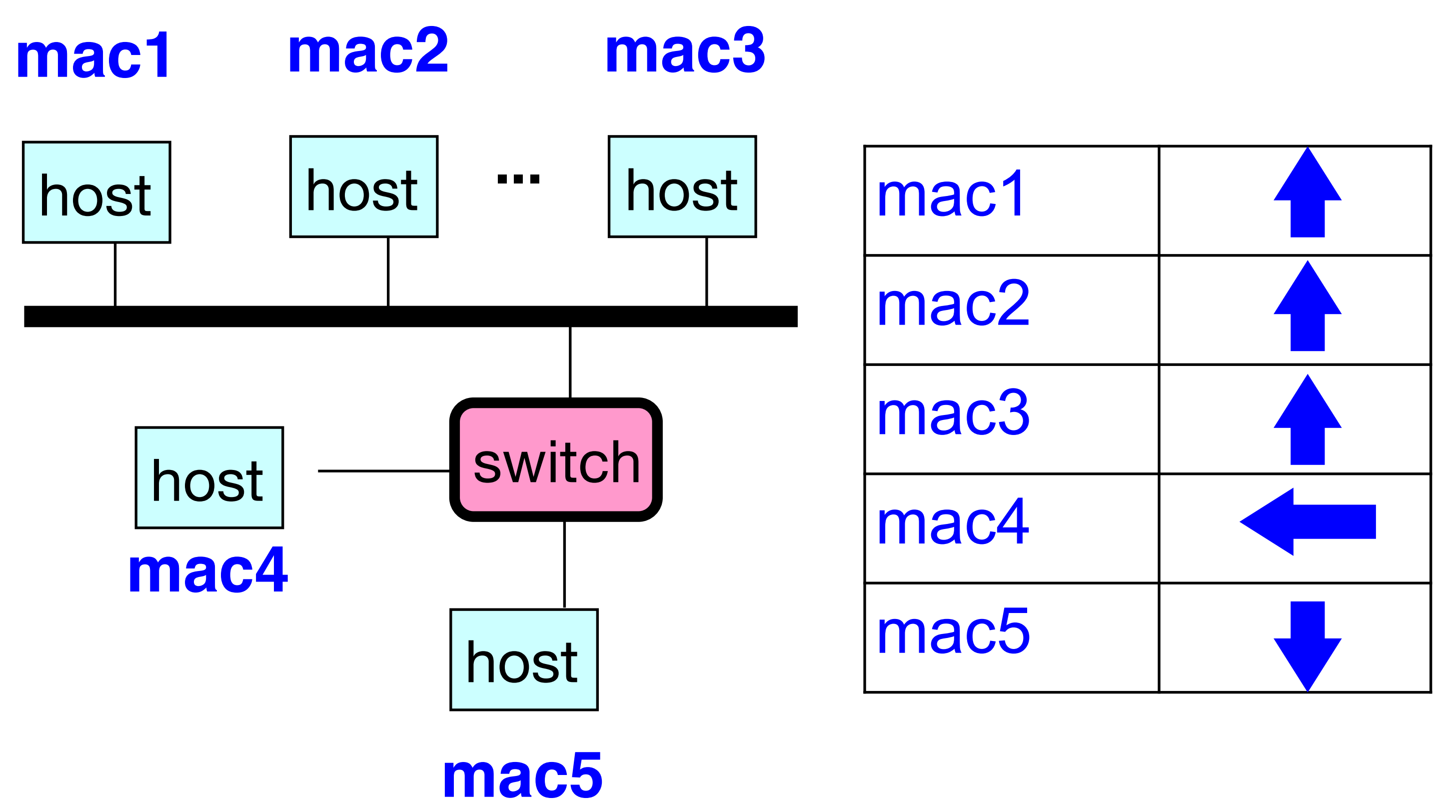
Network Scalability?

“Even simple topologies take days or weeks to create. Workload placement and mobility are restricted by physical network limitations and hardware dependencies require vendor-specific expertise. Network configuration is performed manually and maintenance is both expensive and resource-intensive.” (VMWare)

- Mainly a manual processes: have to manually configure each device with physical presence!

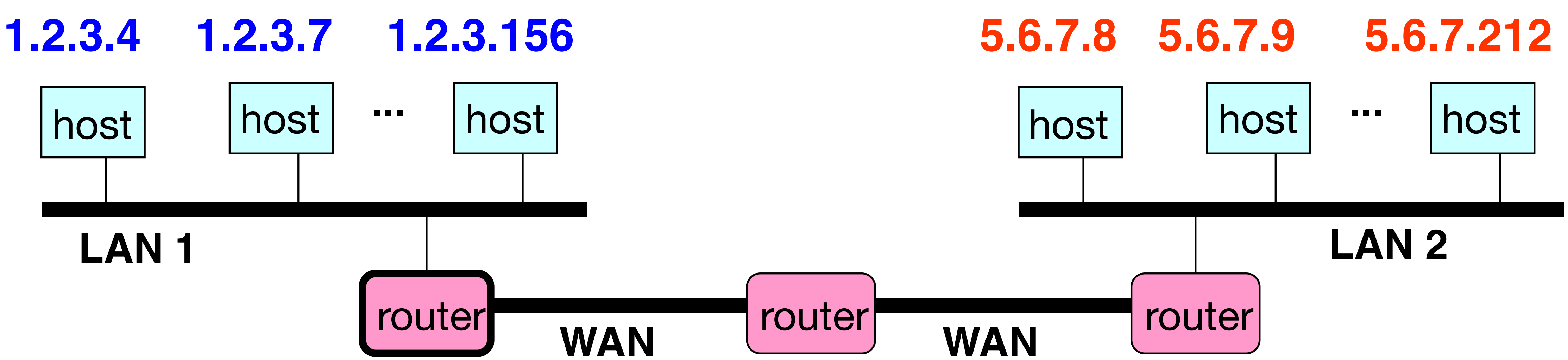
Switch: Match on Destination MAC

- MAC addresses are location independent
  - Assigned by the vendor of the interface card
  - Cannot be aggregated across hosts in LAN



Router: Match on IP Prefix

- IP addresses grouped into common subnets
  - Allocated by ICANN, regional registries, ISPs, and within individual organizations
  - Variable-length prefix identified by a mask length



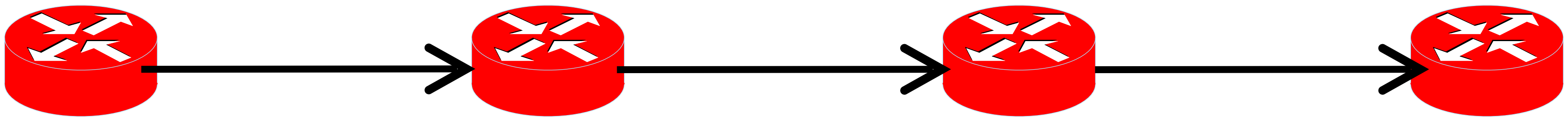
1.2.3.0/24	←
5.6.7.0/24	→

forwarding table

Prefixes may be nested. Routers identify the *longest matching* prefix.

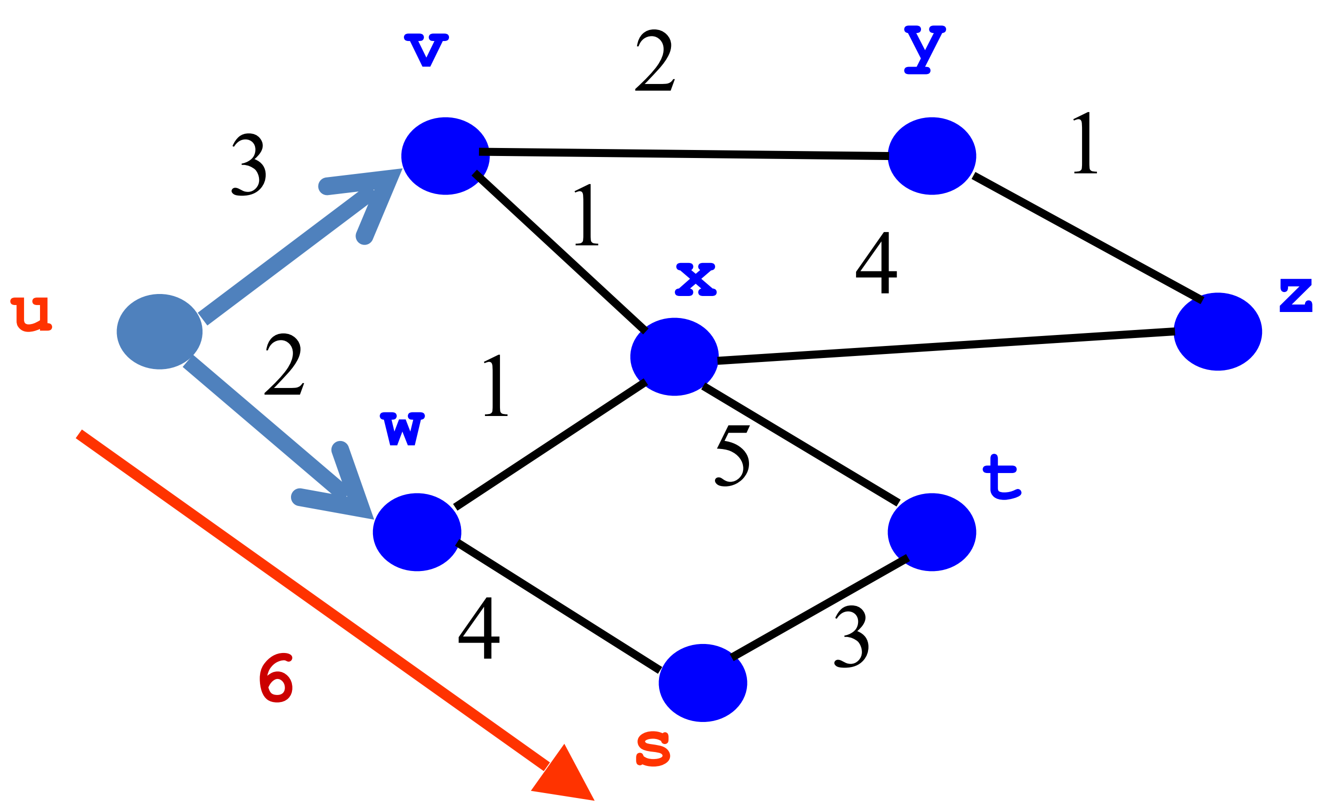
Forwarding vs. Routing

- Forwarding: data plane
  - Directing a data packet to an outgoing link
  - Individual router *using* a forwarding table
- Routing: control plane
  - Computing paths the packets will follow
  - Routers talking amongst themselves
  - Individual router *creating* a forwarding table



Example: Shortest-Path Routing

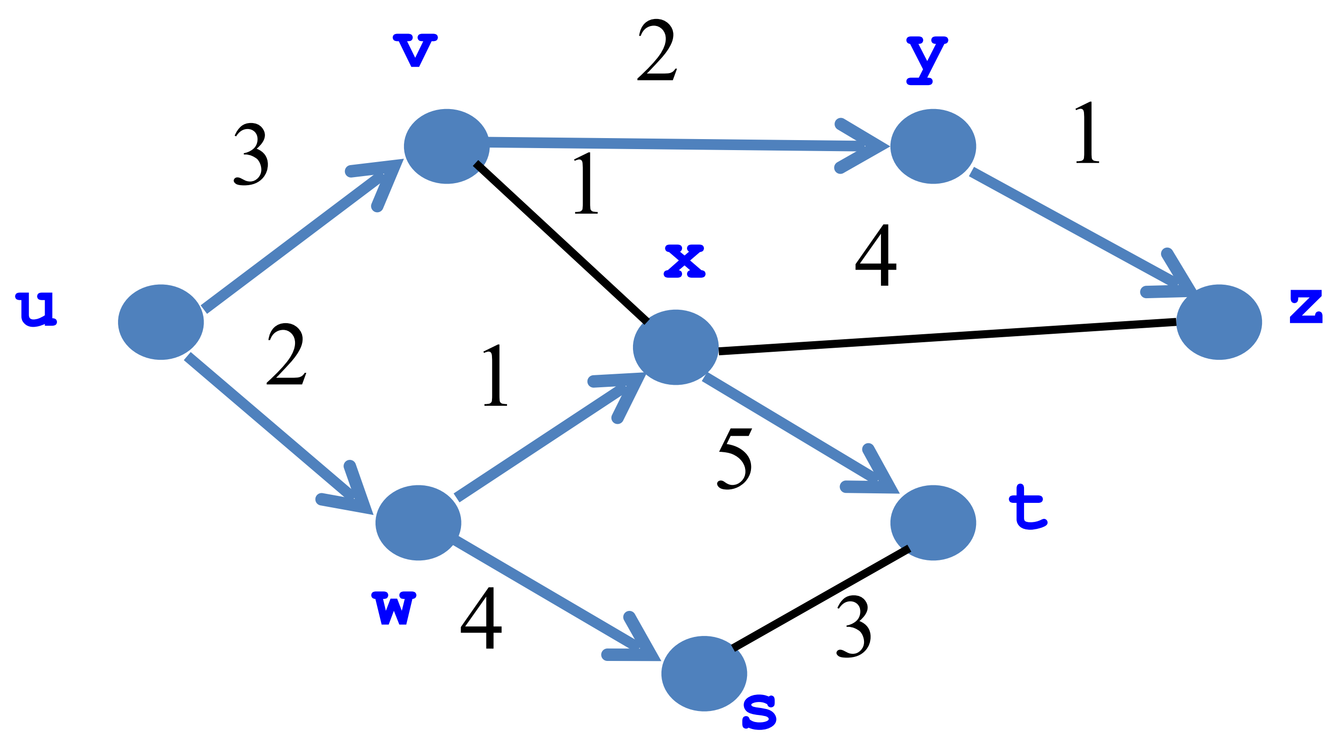
- Compute: *path costs* to all nodes
  - From a source u to all other nodes
  - Cost of the path through each link
  - Next hop along least-cost path to s



	link
v	(u,v)
w	(u,w)
x	(u,w)
y	(u,v)
z	(u,v)
s	(u,w)
t	(u,w)

Distributed Control Plane

- Link-state routing: OSPF, IS-IS
  - Flood the entire topology to all nodes
  - Each node computes shortest paths
  - Dijkstra's algorithm



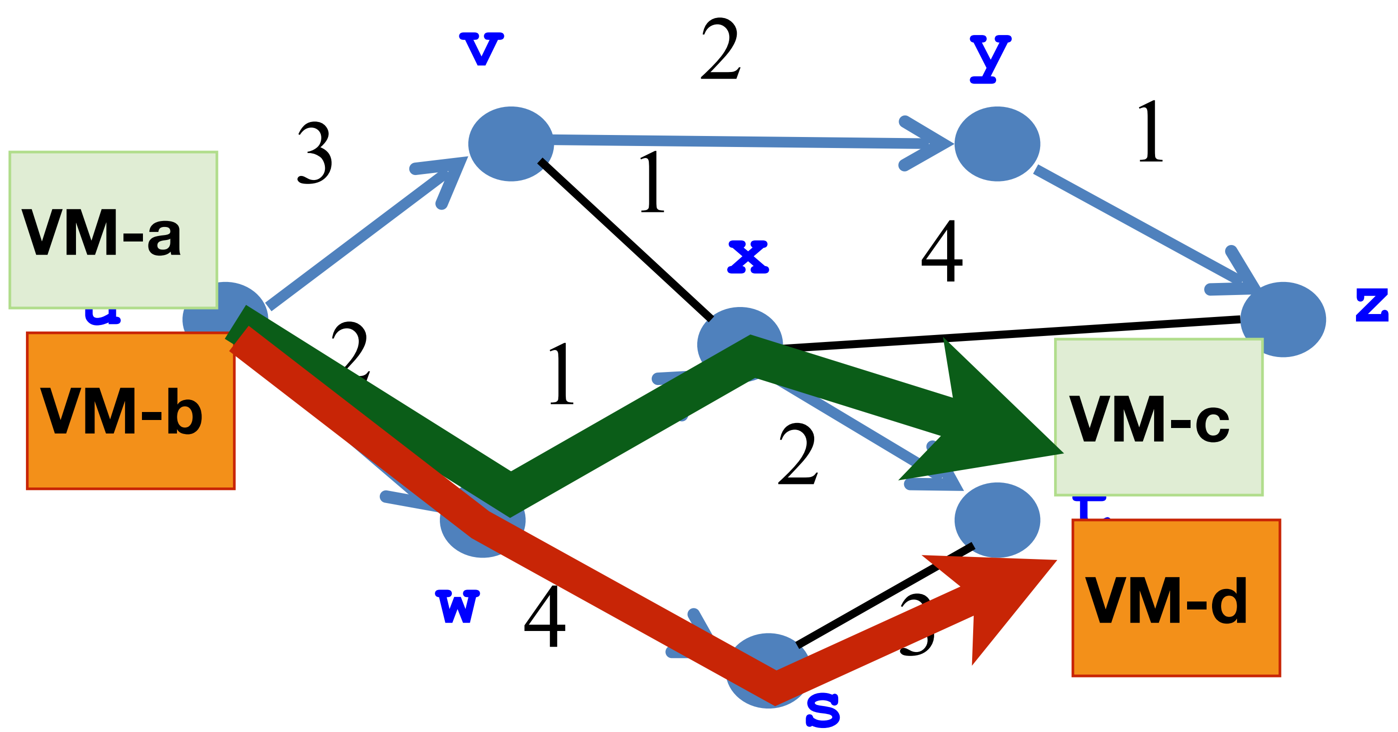
	link
v	(u,v)
w	(u,w)
x	(u,w)
y	(u,v)
z	(u,v)
s	(u,w)
t	(u,w)

Flexibility Problem

All packets arriving at a switch/router are treated the same

- Only consider the destination IP/MAC address to decide path

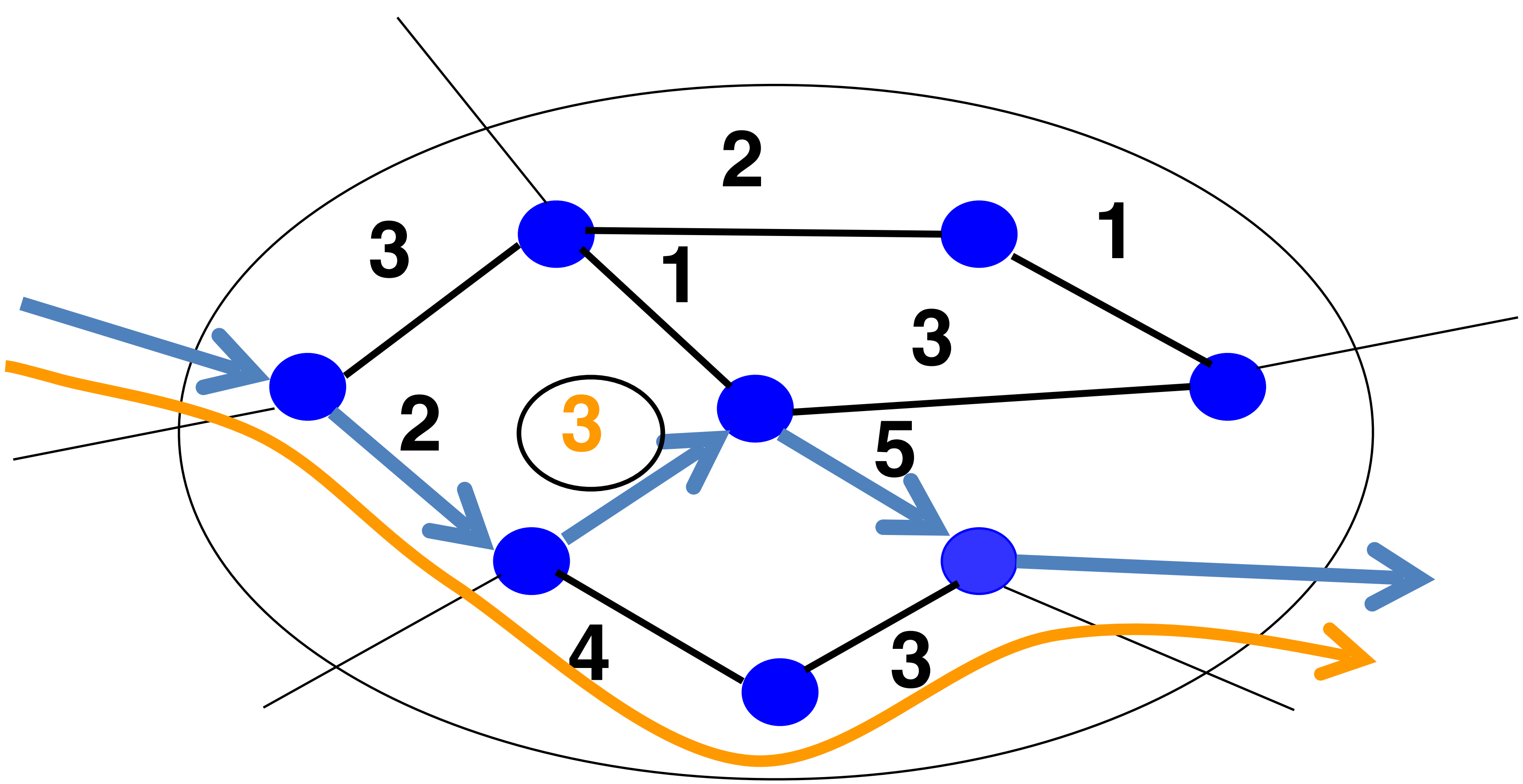
Prevents customizing the network for different cloud tenants!



Green VMs are paying more for higher bandwidth networking so we would like to support different paths!

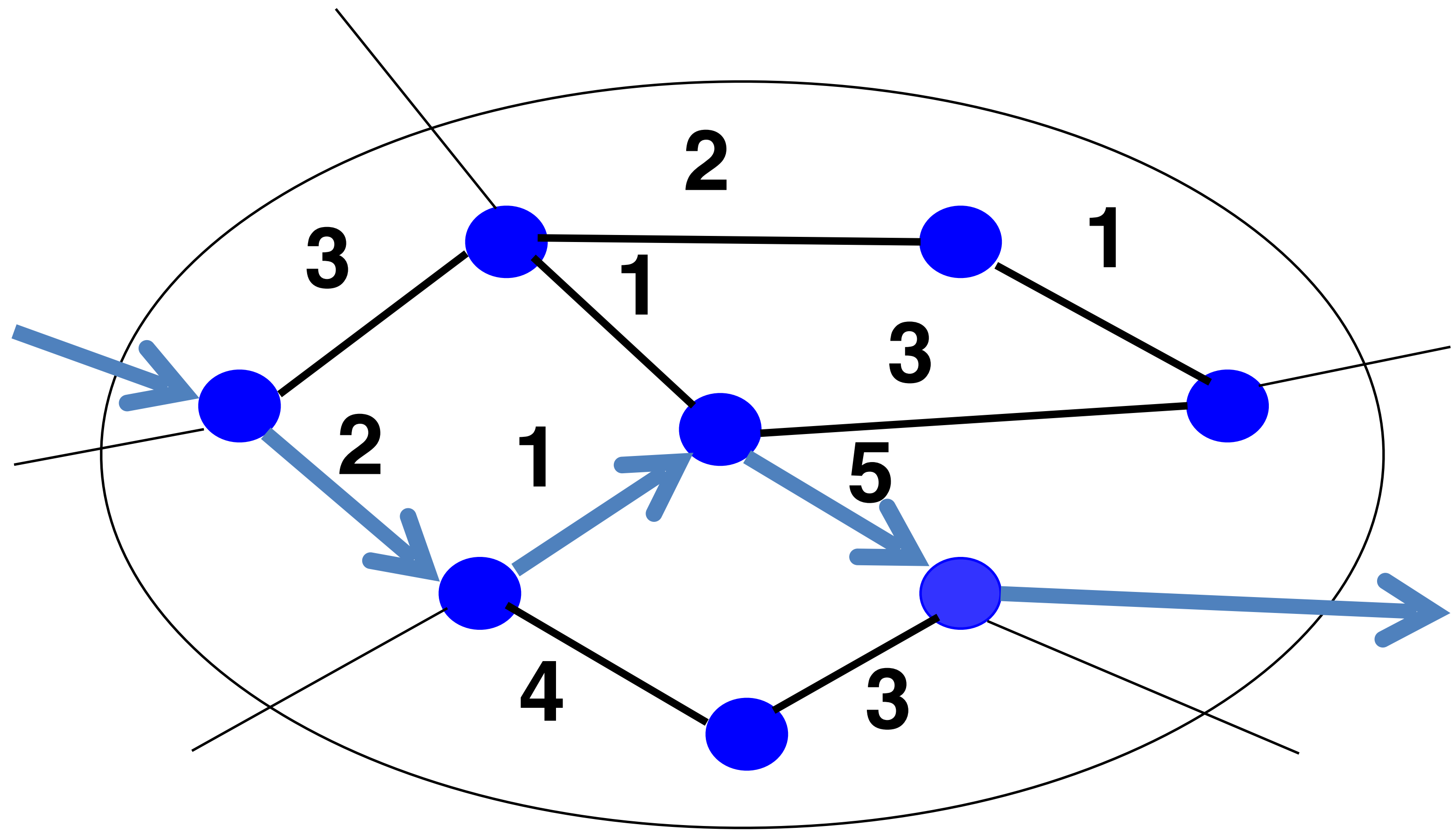
Traffic Engineering Problem

- Management plane: setting the weights
  - Inversely proportional to link capacity?
  - Proportional to propagation delay?
  - Network-wide optimization based on traffic?



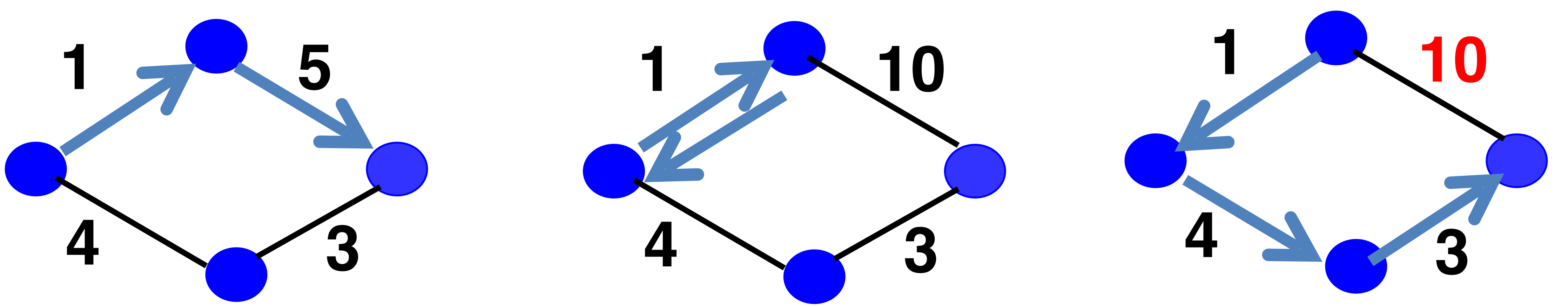
Traffic Engineering: Optimization

- Inputs
  - Network topology
  - Link capacities
  - Traffic matrix
- Output
  - Link weights
- Objective
  - Minimize max-utilized link
  - Or, minimize a sum of link congestion



Transient Routing Disruptions

- Topology changes
  - – Link weight change
  - – Node/link failure or recovery
- Routing convergence
  - – Nodes temporarily disagree how to route
  - – Leading to transient loops and blackholes

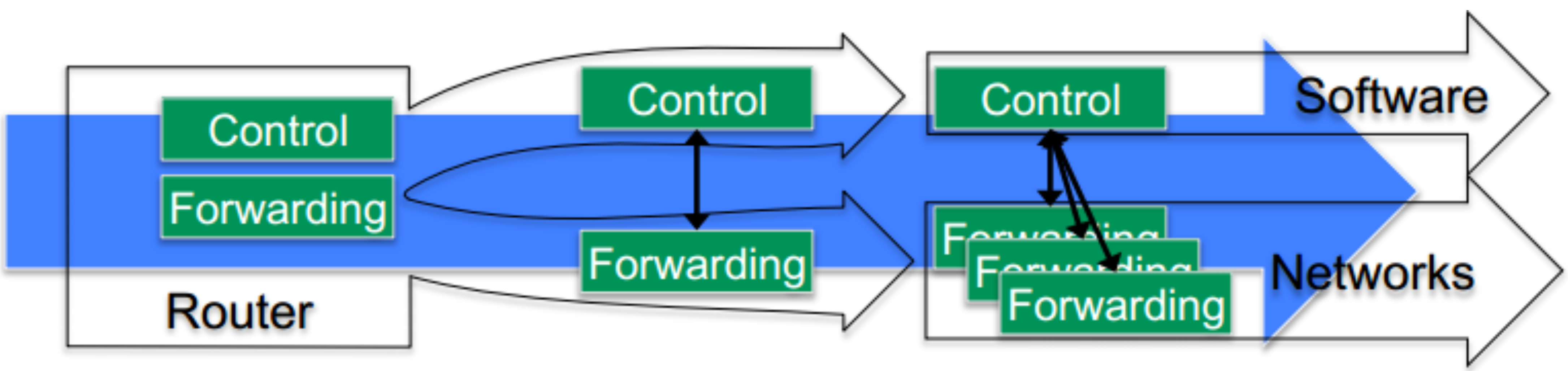


Management Plane Challenges

- Indirect control
  - Changing weights instead of paths
  - Complex optimization problem
- Uncoordinated control
  - Cannot control which router updates first
- Interacting protocols and mechanisms
  - Routing and forwarding
  - Naming and addressing
  - Access control
  - Quality of service
  - …

Software Defined Networking

- Solution: Software-Defined-Networking (SDN)
  - Decouples the control plane from the data plane



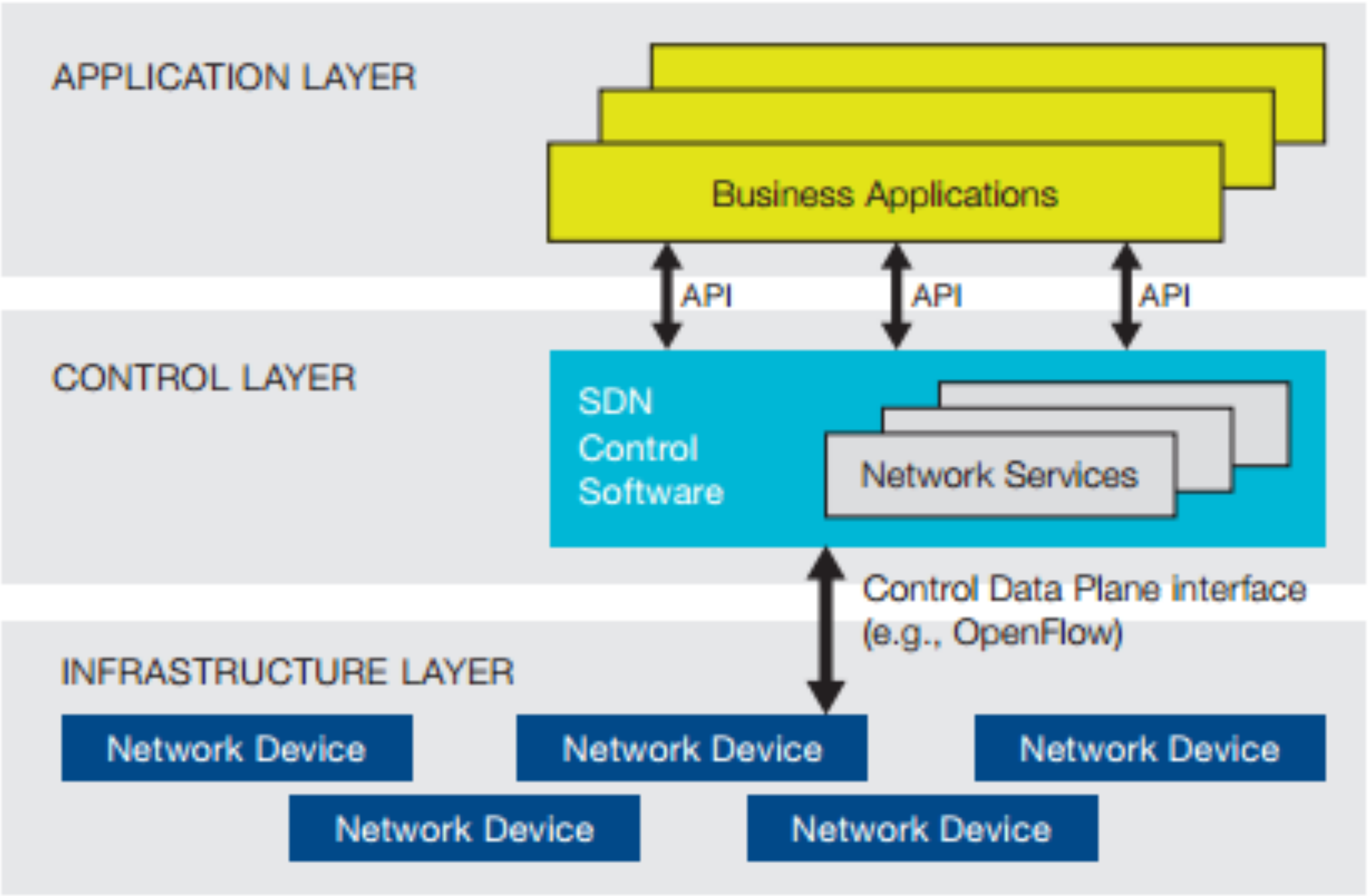
Images taken from materials of the Open Networking Foundation: https://www.opennetworking.org/

Software Defined Networking

- SDN makes the network *programmable*

- OSPF, DiffServ, IntServ, MPLS, RSVP?
  - All such protocols can be done in software, controlled by a central instance
  - Scalable, easily manageable, better interoperability

SDN Components at a glance



SDN Components at a glance

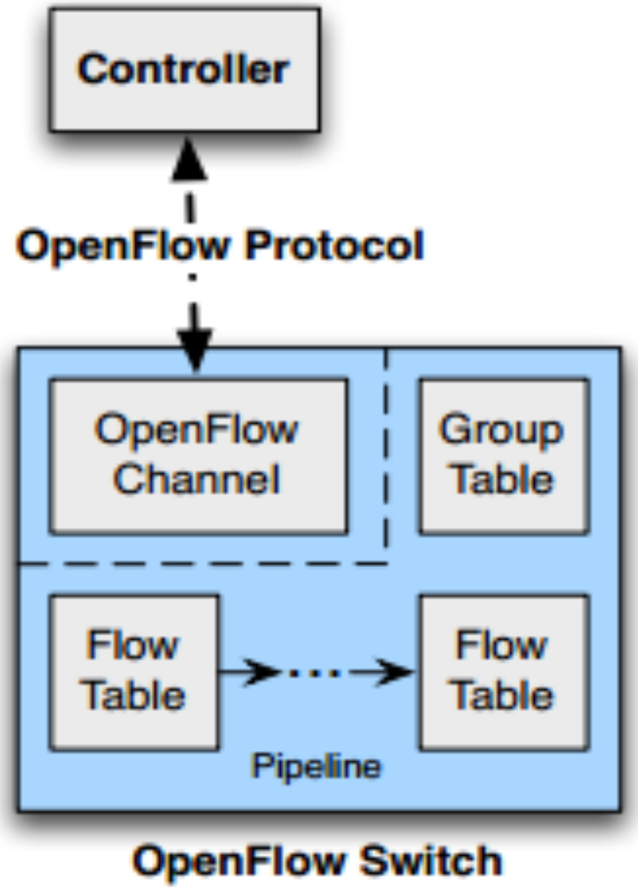
- Programmable Open APIs:
  - Connects applications with control plane
  - Allows for *programming* of routing, QoS, etc.
- Standard Communication Interface (e.g., OpenFlow):
  - Between control and data planes
  - Allows direct access to forwarding plane
- Network Controller (*logically* centralized):
  - Sets up rules, actions, etc. for the network devices
  - Core element of SDN

SDN Benefits

- SDN further allows for…
  - elastic resource allocation (e.g., to match QoS agreements)
  - distribution of the load on links (e.g., between backbone and application servers in SaaS)
  - scalability (no need to manually configure each of thousands (or even millions?) of devices)
  - overhead reduction
  - …and more

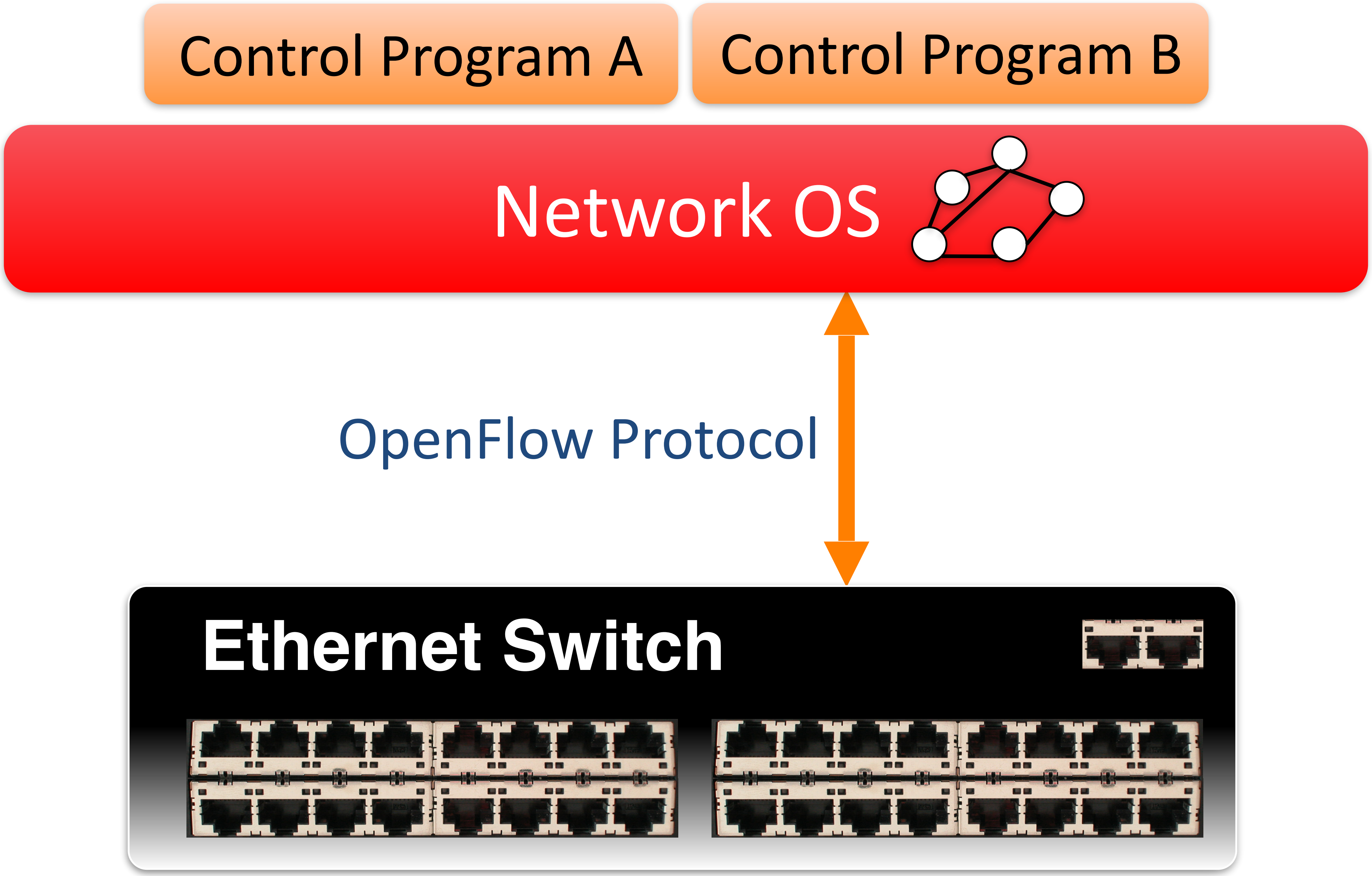
OpenFlow – The SDN Protocol

- Communication between the controller and the network devices (i.e., switches)

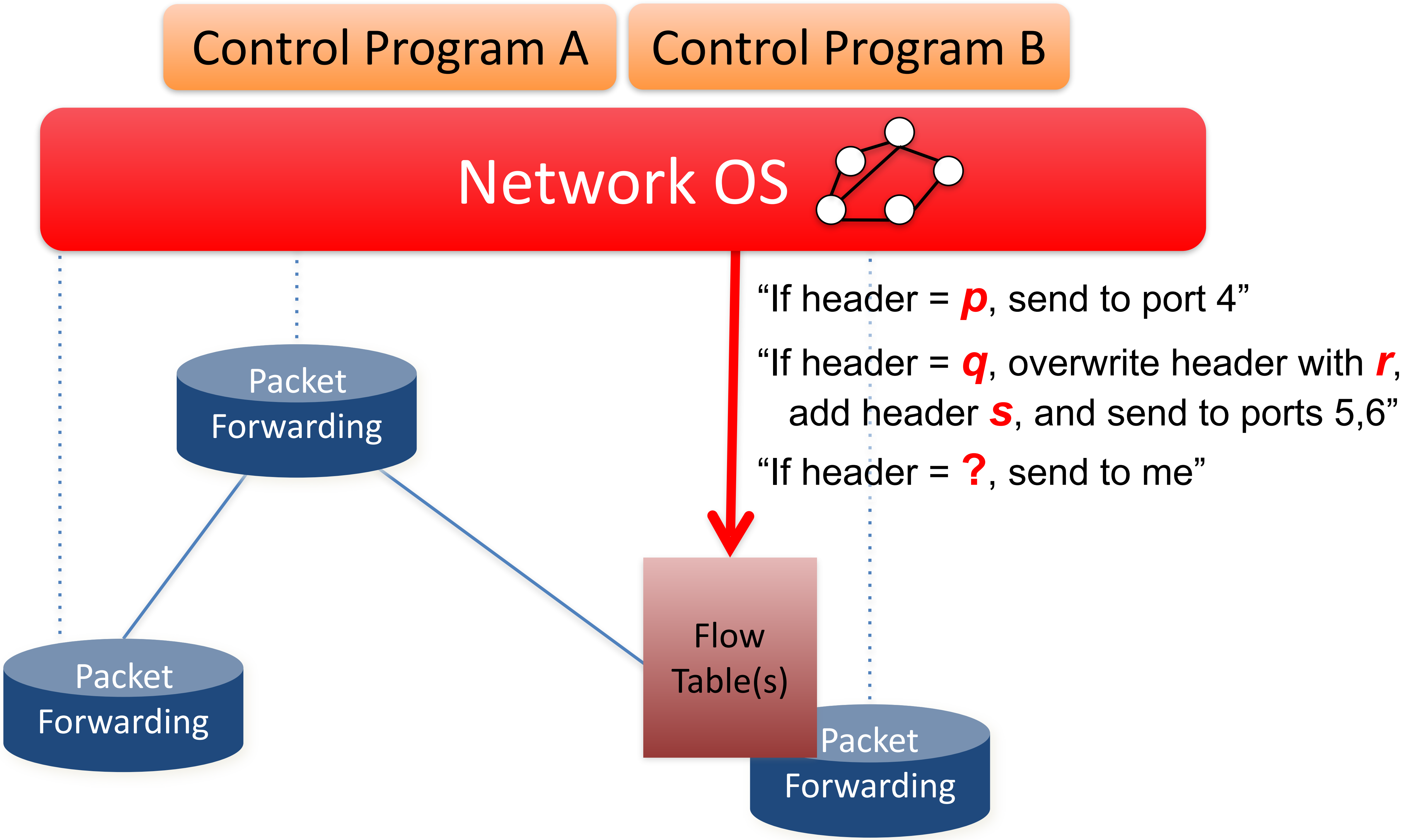


Specification by the Open Networking Foundation: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.4.pdf (March 2014)

OpenFlow Basics



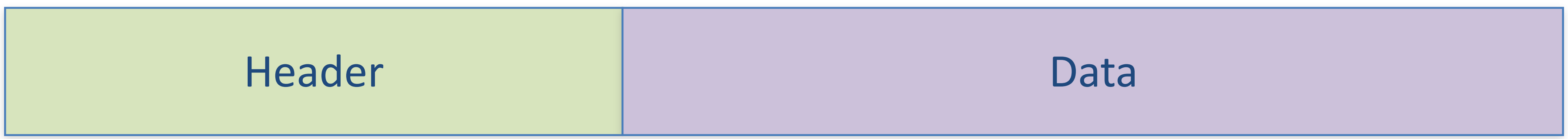
OpenFlow Basics



Plumbing Primitives

<Match, Action>

Match arbitrary bits in headers:



Match: 1000x01xx0101001x

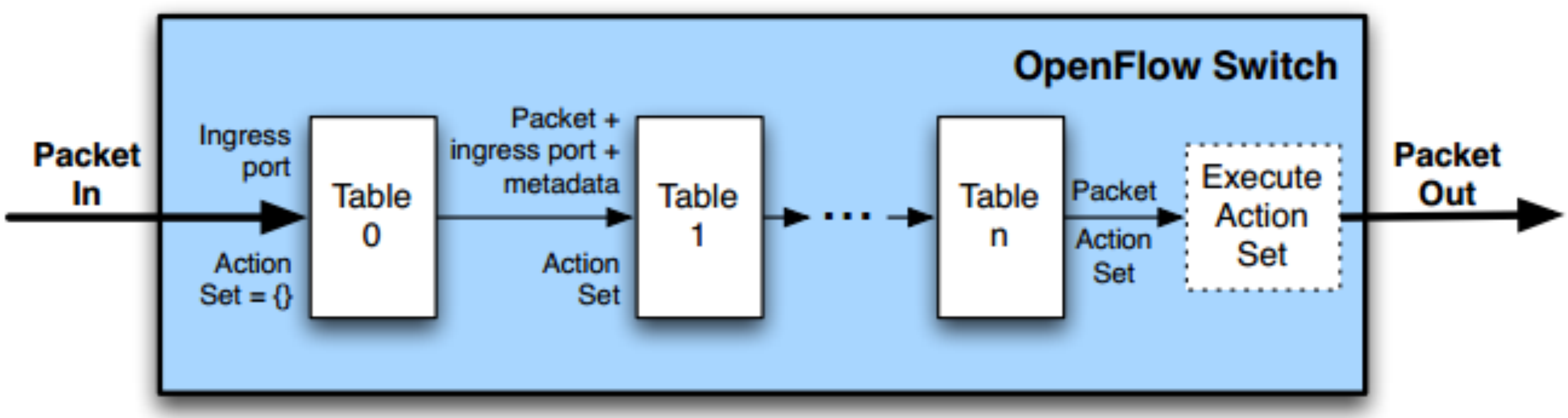
- Match on any header field, but not data
- Allows ‘any’ flow granularity

Action

- Forward to port(s), drop, send to controller
- Overwrite header with mask, push or pop
- Forward at specific bit-rate

OpenFlow – Switches

- Incoming packets are matched against rule tables
- Find highest priority match and execute actions
  - Send out port
  - Forward to another table
  - Drop
  - Rate limit
  - etc…



OpenFlow – Switches

- If no match in table: *table miss*
- Handling: depends on table configuration – might be *drop packet, forward to other table,* ***forward to controller***
- Forward to controller allows to set up a flow entry (i.e., at the beginning of a flow)
  - Based on a program!

Table Miss

- What can the controller do if there is a miss?
- What happens to subsequent packets?

- Why only send to controller on miss?
  - Why not every packet?

Examples

Switching - can customize based on known MAC addresses

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	00:1f:..	*	*	*	*	*	*	*	port6

Flow Switching - fine grained switching for each TCP connection

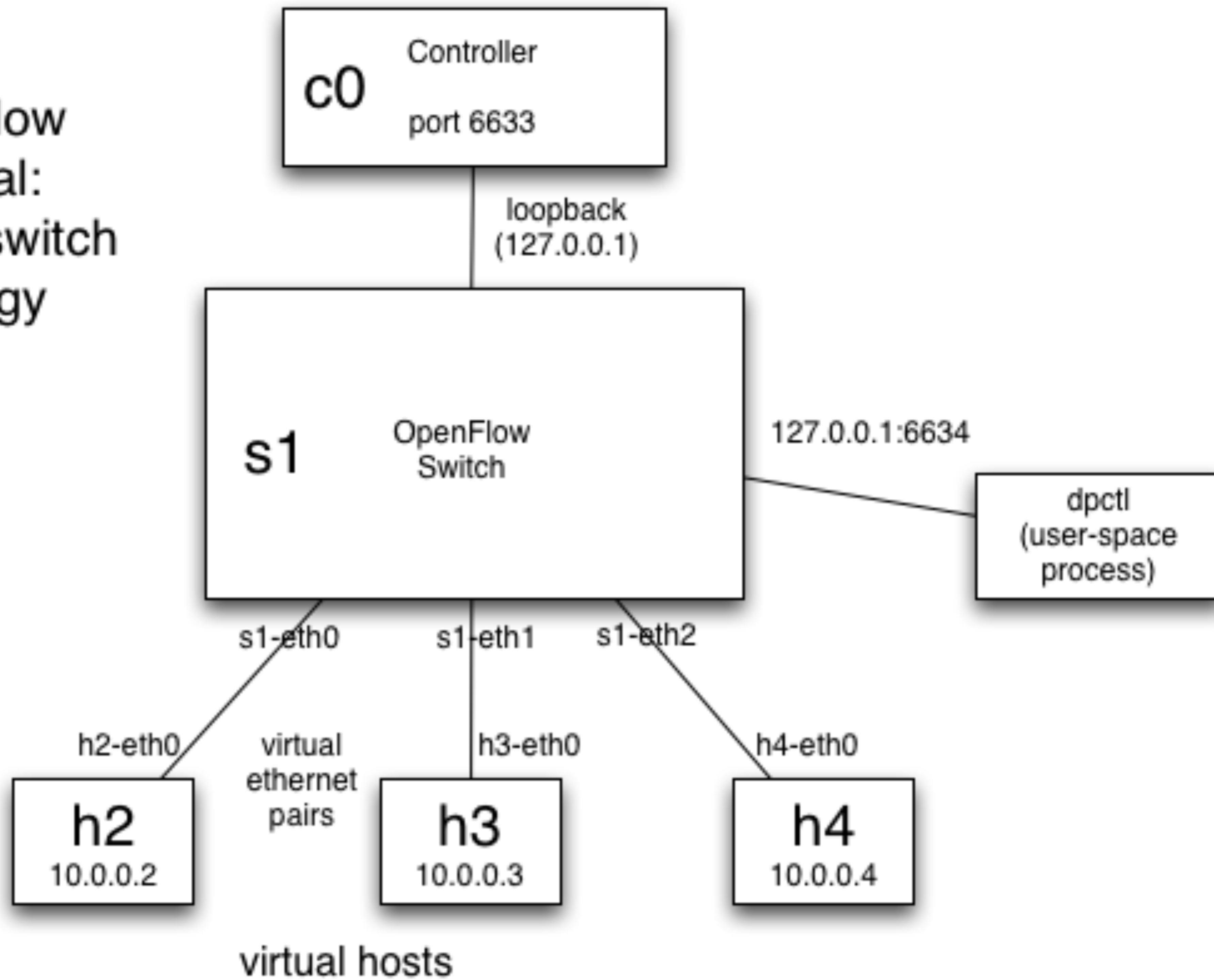
Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
port3	00:20..	00:1f..	0800	vlan1	1.2.3.4	5.6.7.8	4	17264	80	port6

Firewall - not just switching, but also dropping/rate limiting/etc

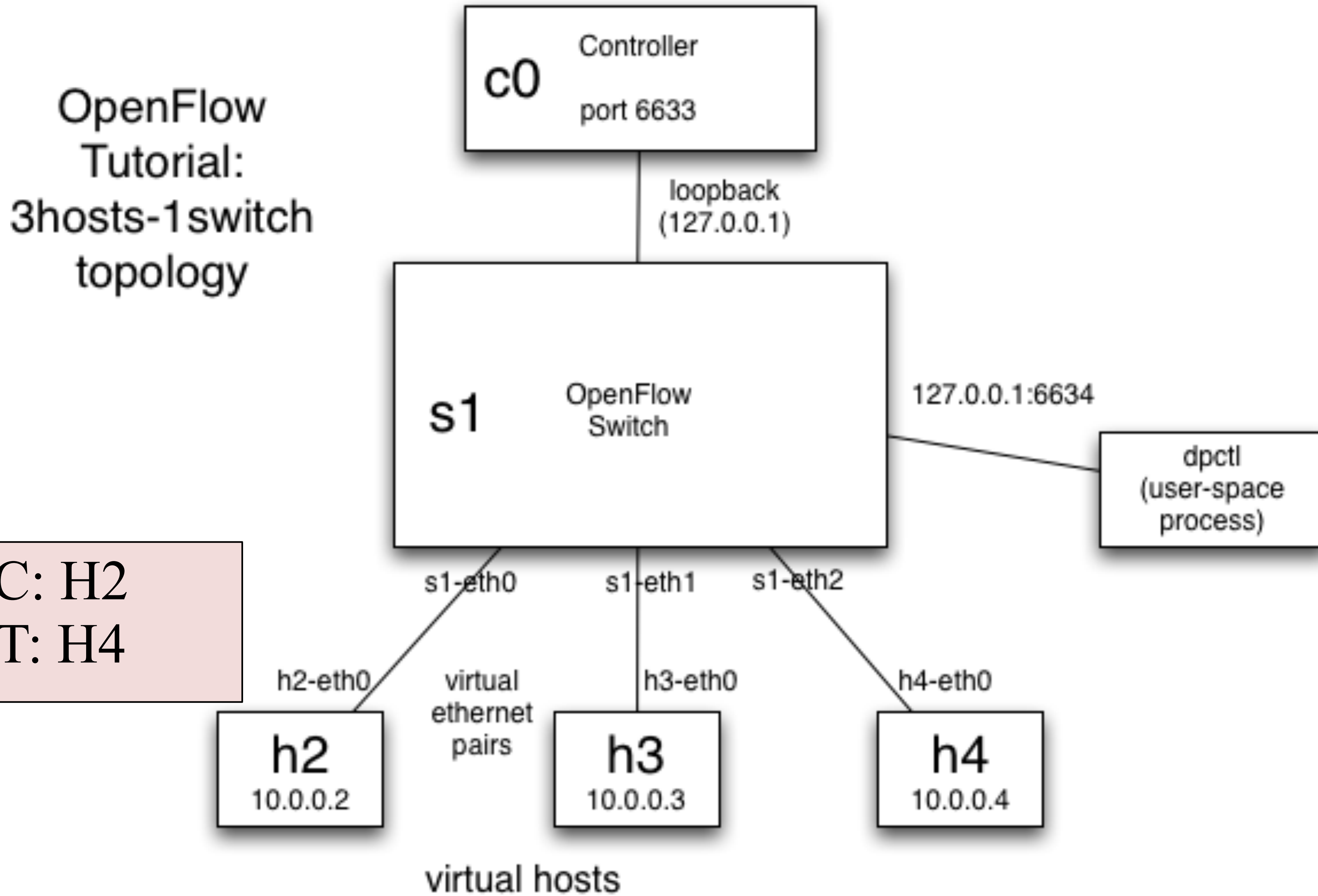
Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	*	*	*	22	drop

OpenFlow - Example

OpenFlow
Tutorial:
3hosts-1switch
topology

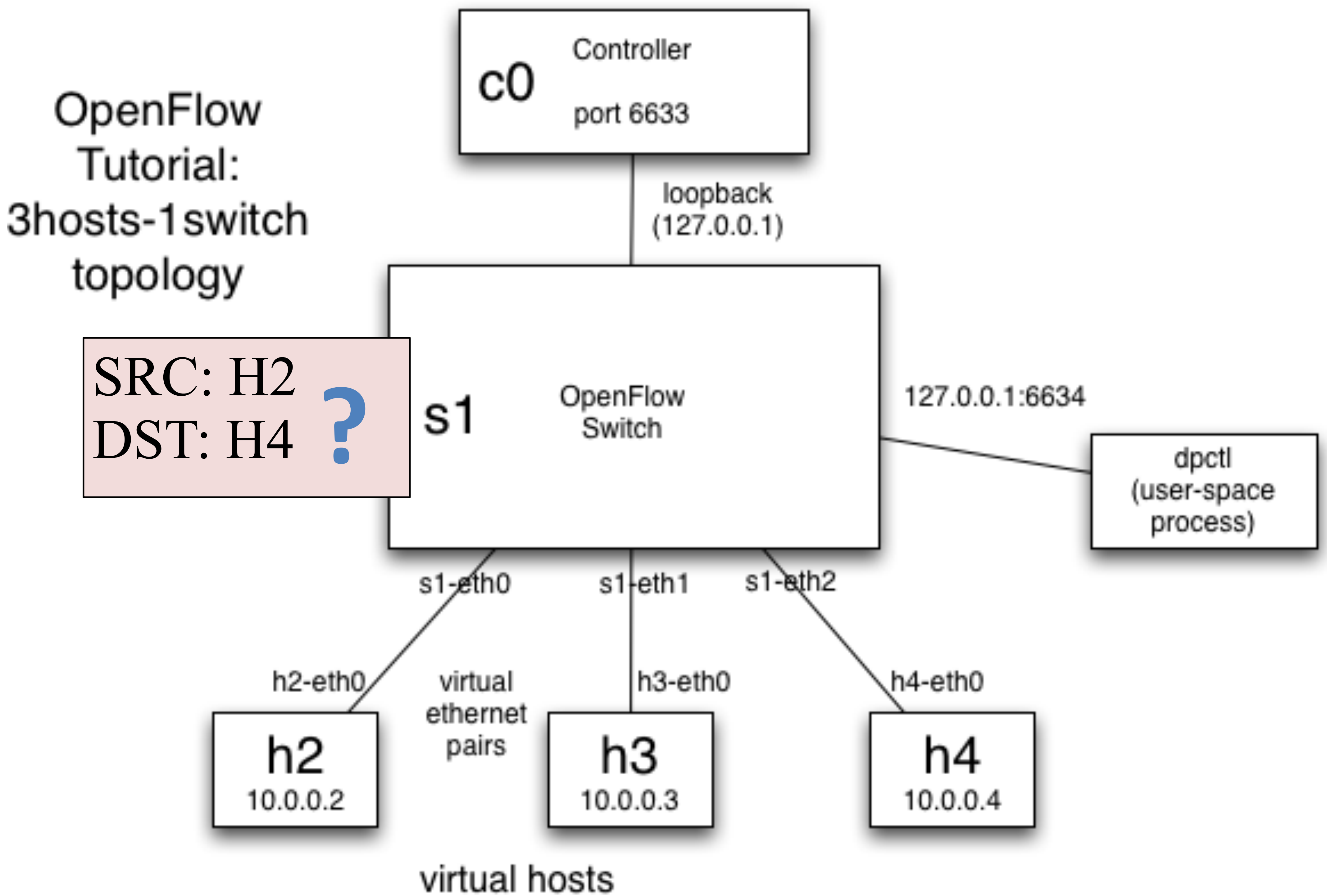


OpenFlow - Example

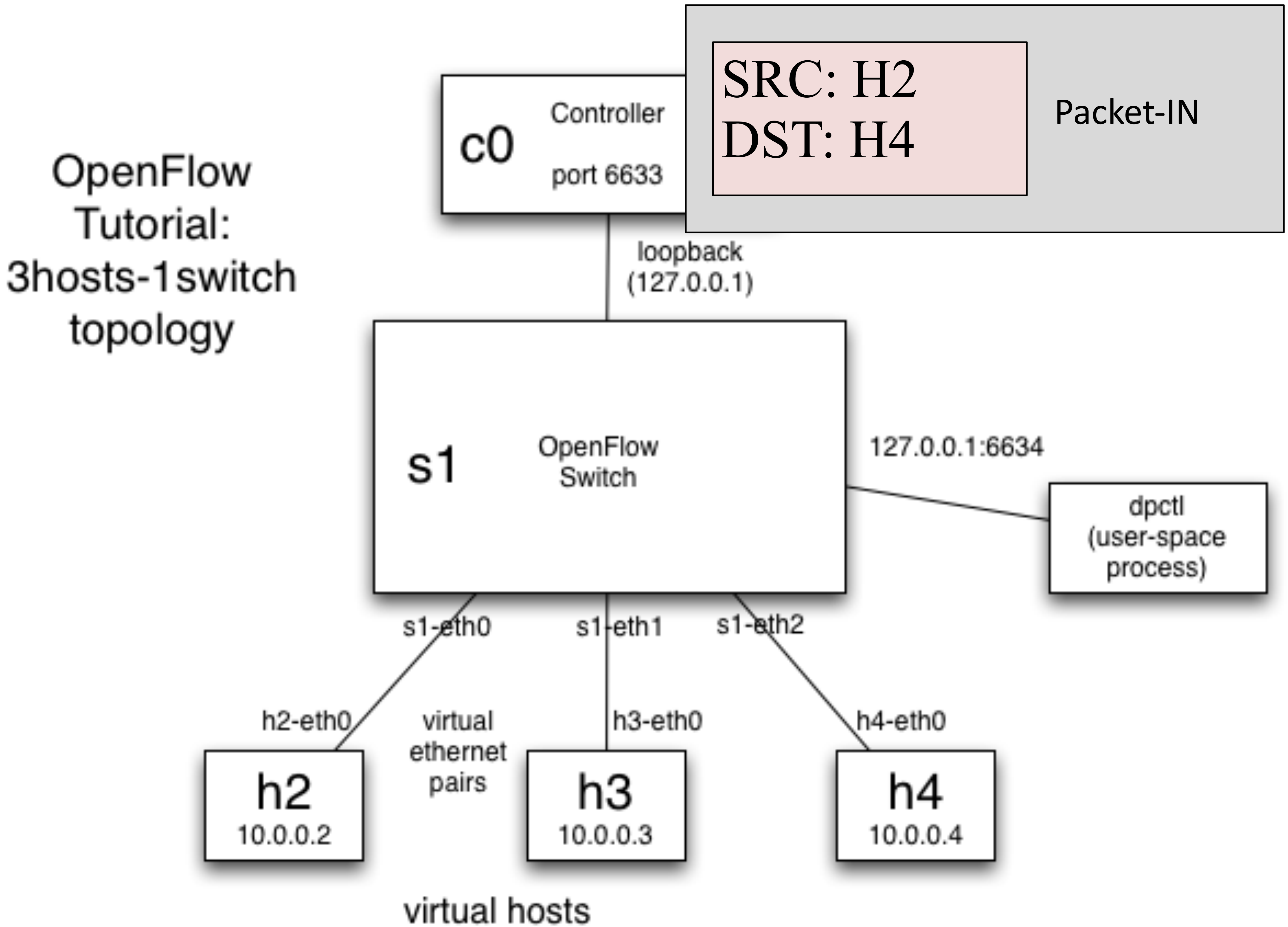


SRC: H2
DST: H4

OpenFlow - Example

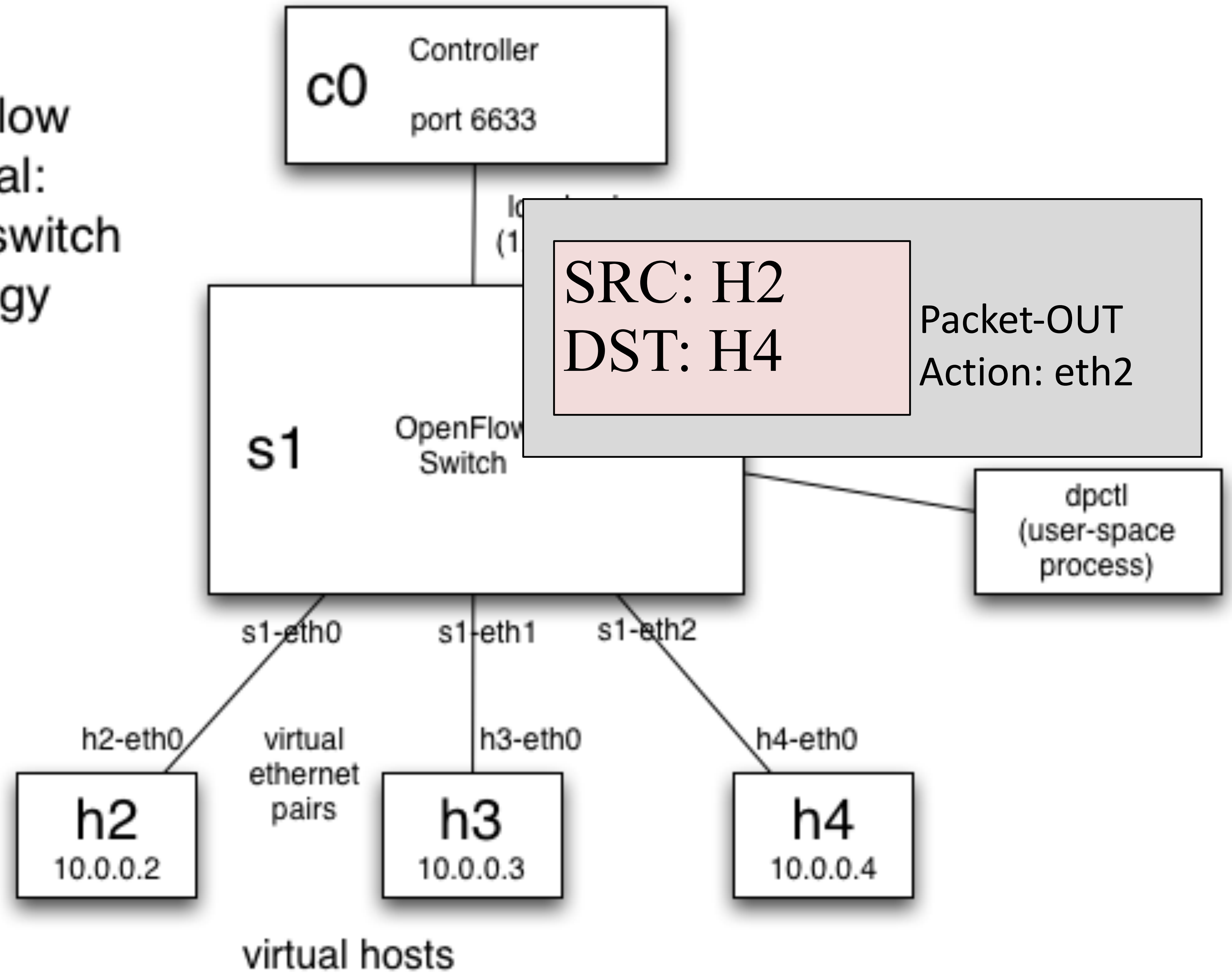


OpenFlow - Example

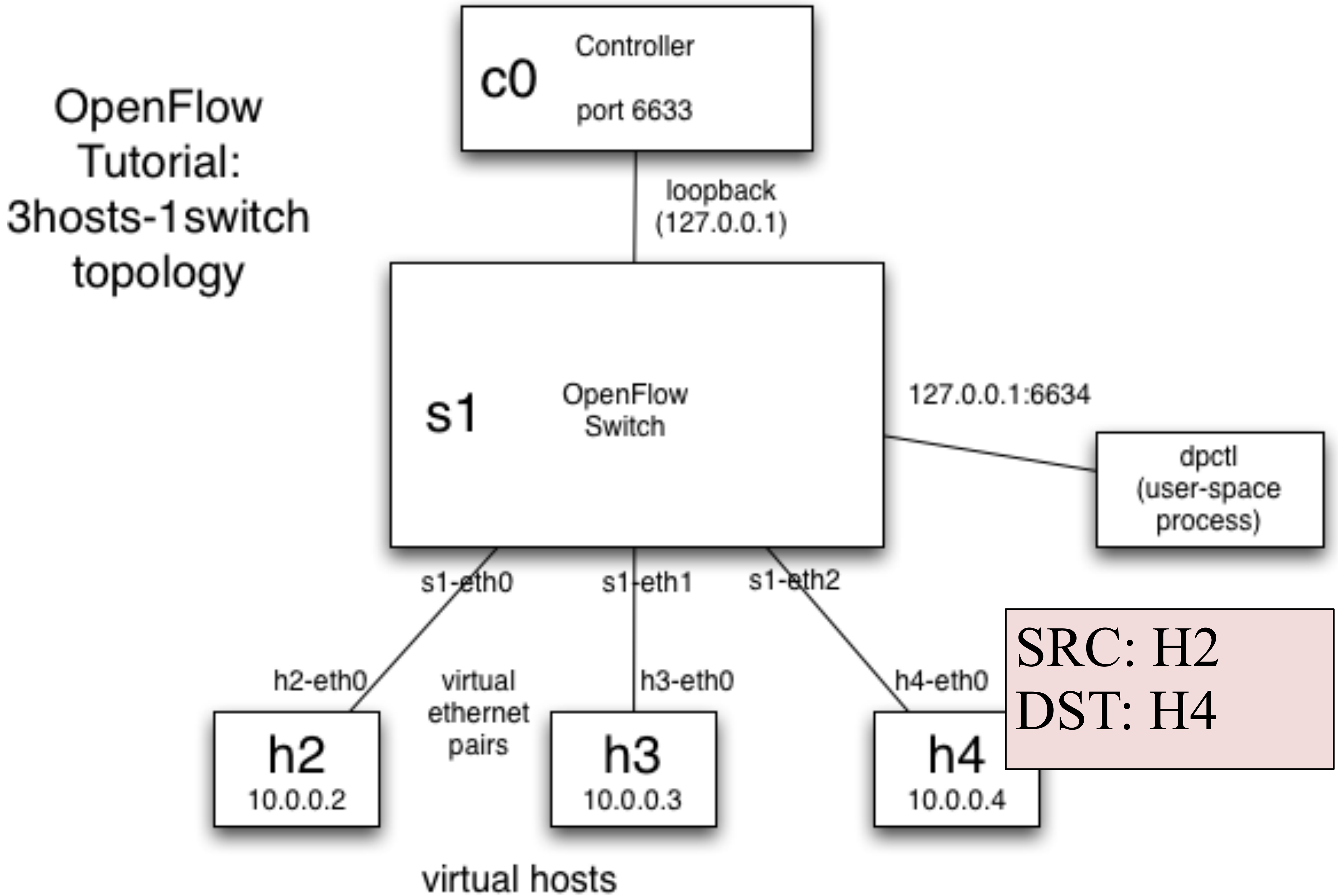


OpenFlow - Example

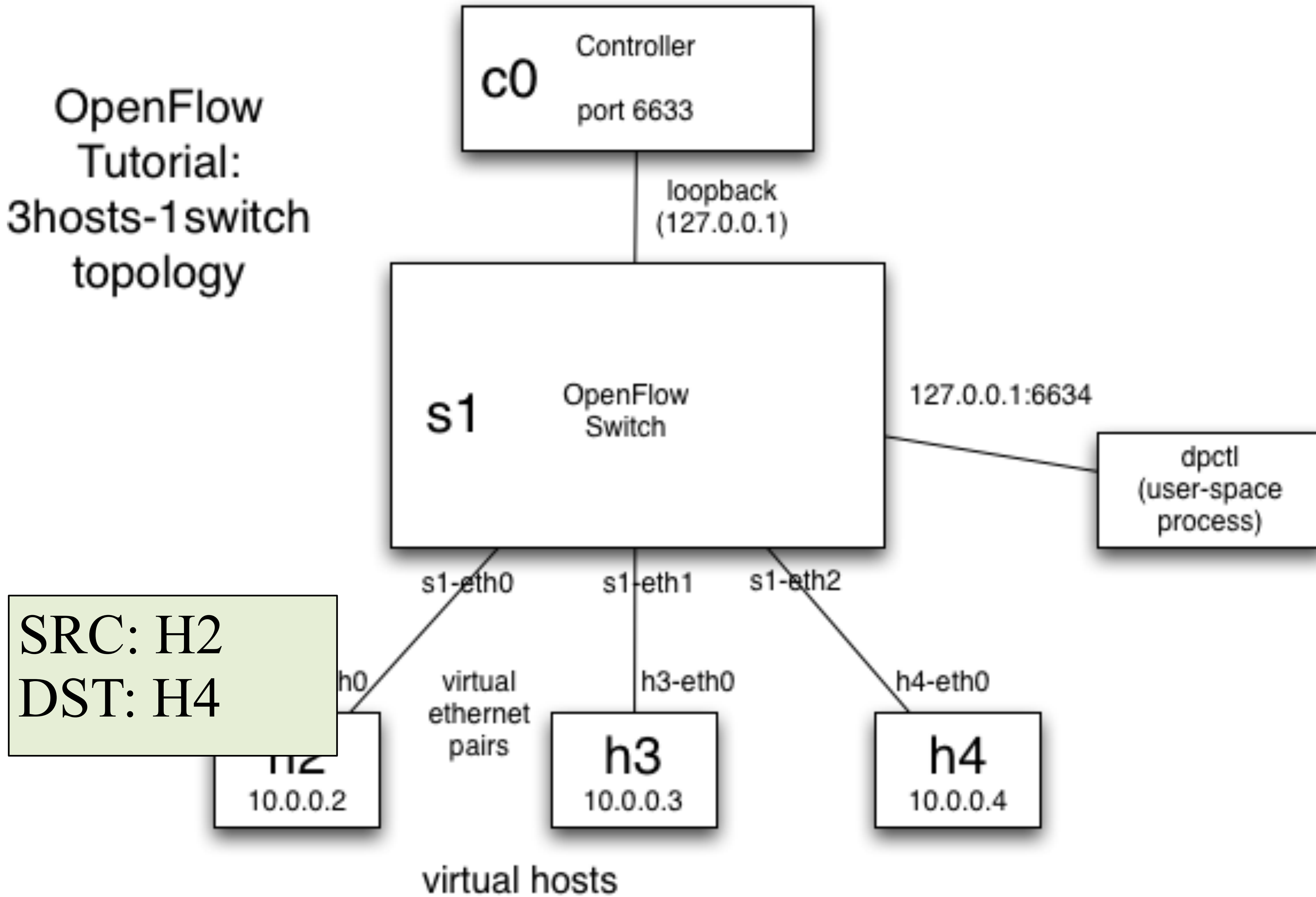
OpenFlow Tutorial: 3hosts-1switch topology



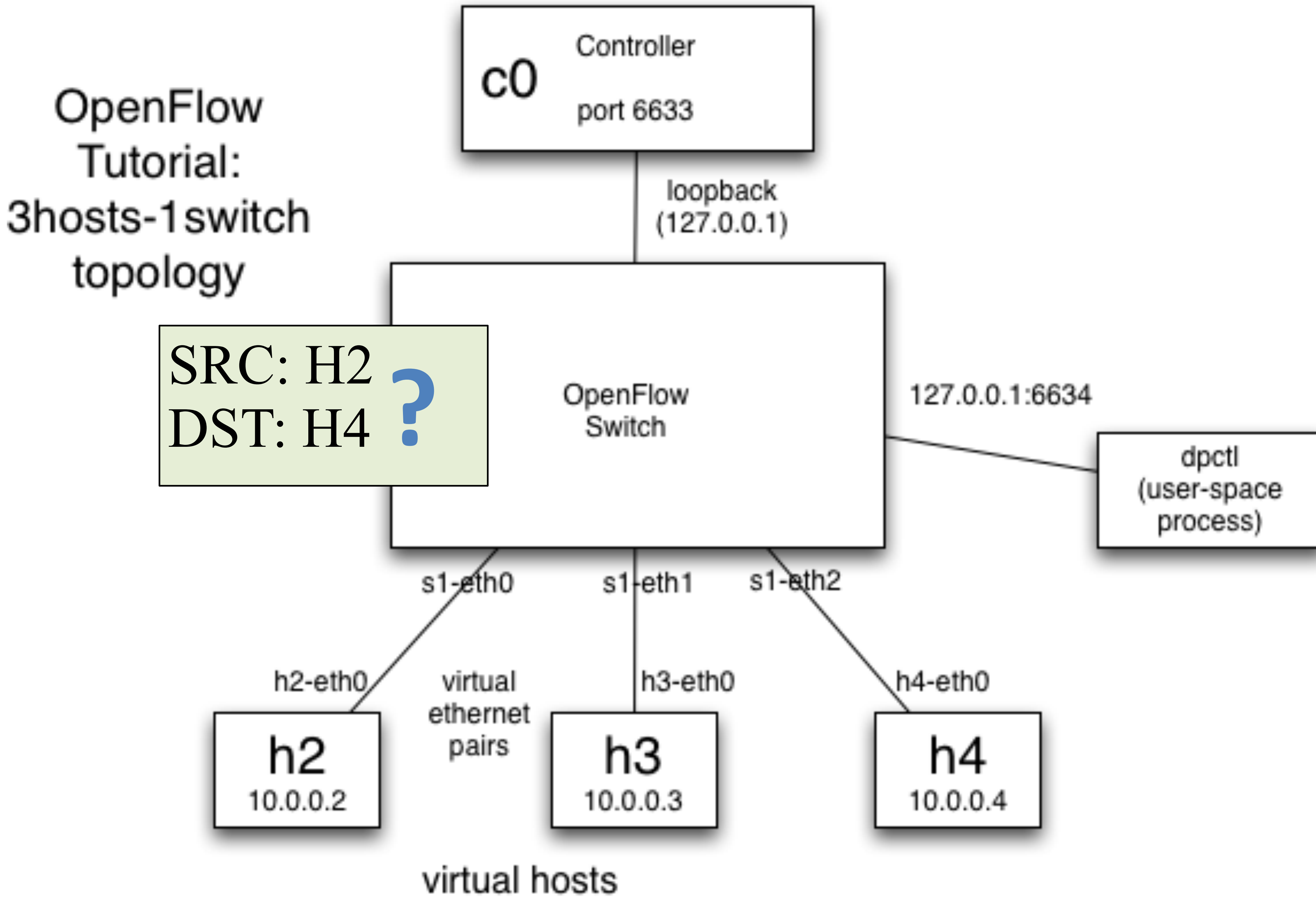
OpenFlow - Example



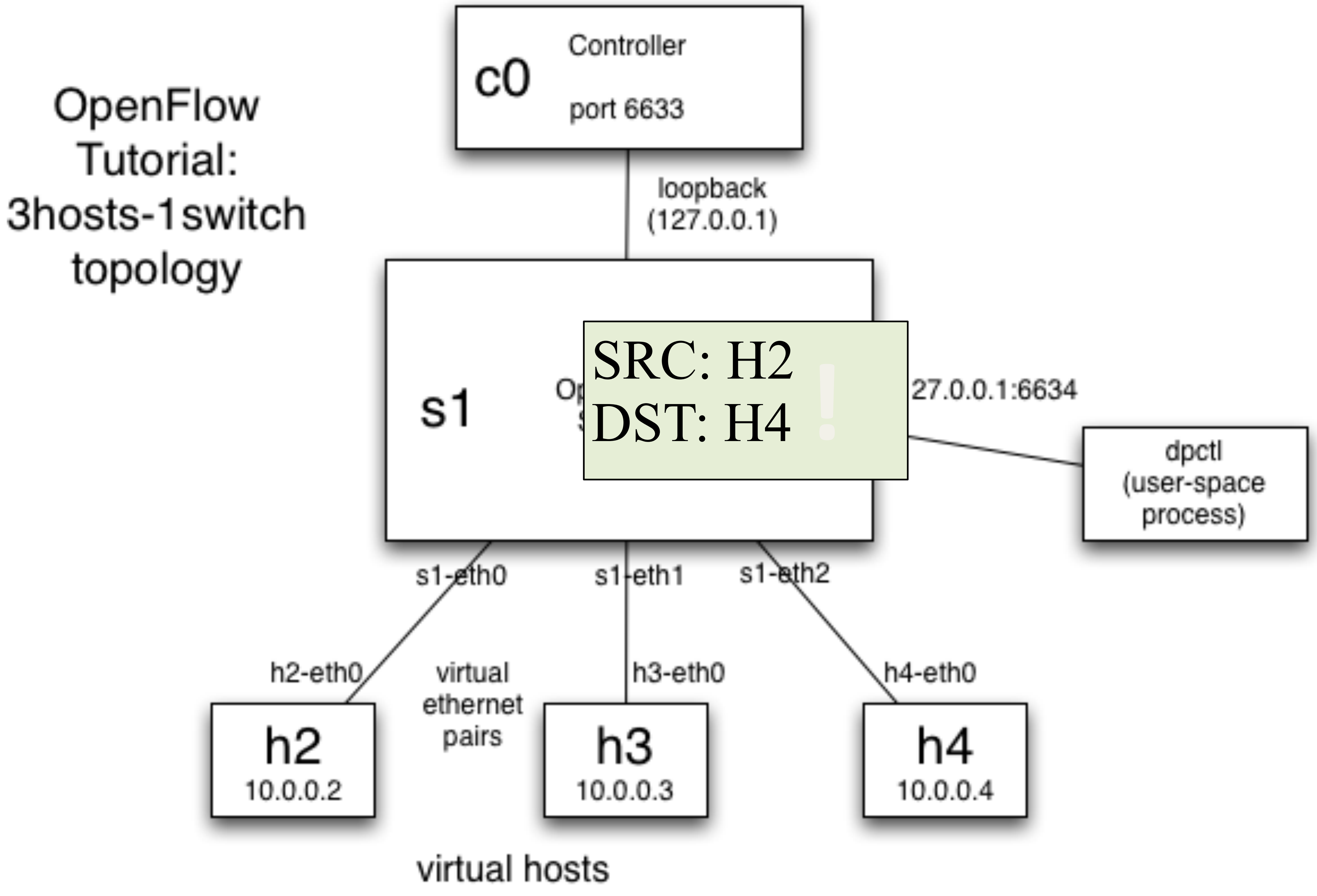
OpenFlow - Example



OpenFlow - Example

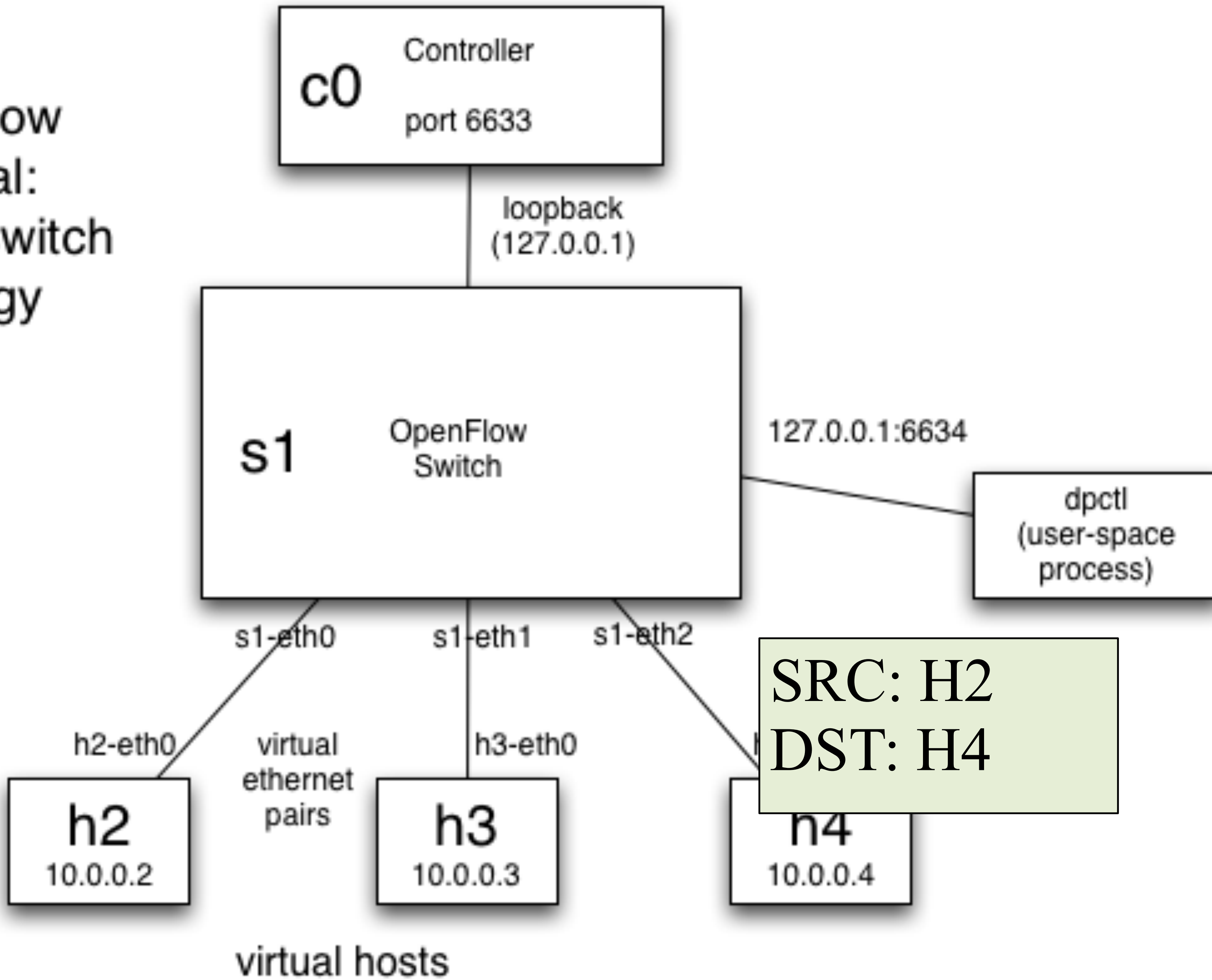


OpenFlow - Example



OpenFlow - Example

OpenFlow Tutorial: 3hosts-1switch topology



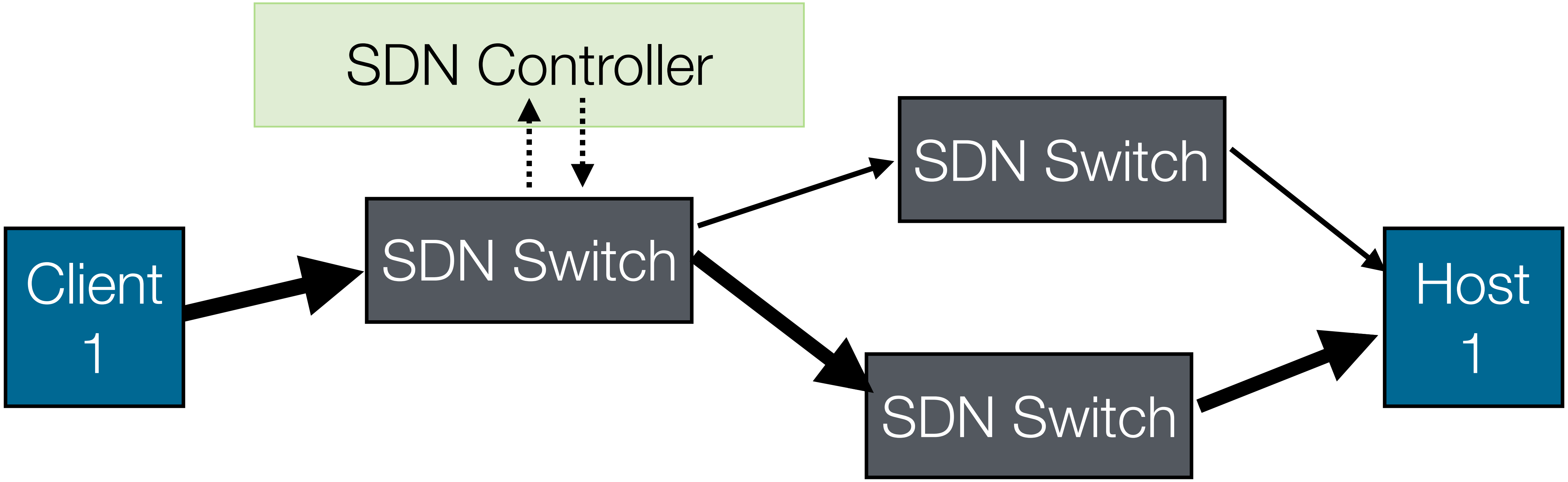
SDN Workflow

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: defines flow table rules for switches

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



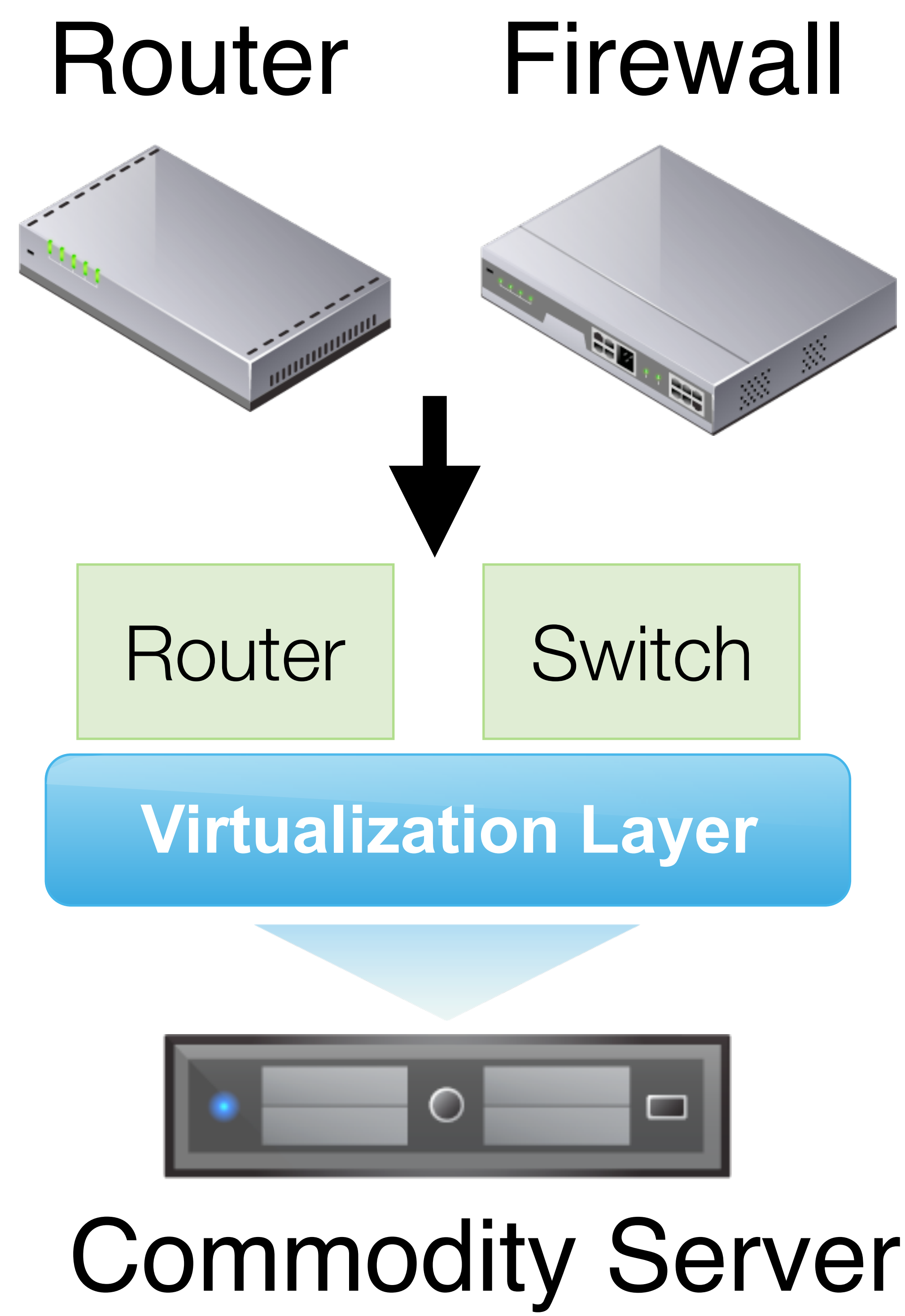
Network Function Virtualization

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…



Network Data Plane

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

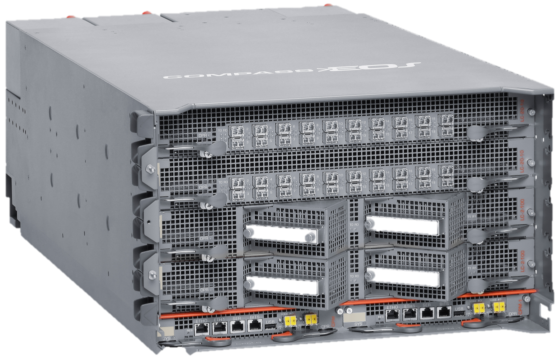
$5.99 Shipping

ADD TO CART ▶

Software-Based Data Plane

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)

AKA “middleboxes"

Switches, routers, firewalls, NAT

- 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 (Evolved Packet Core - EPC)

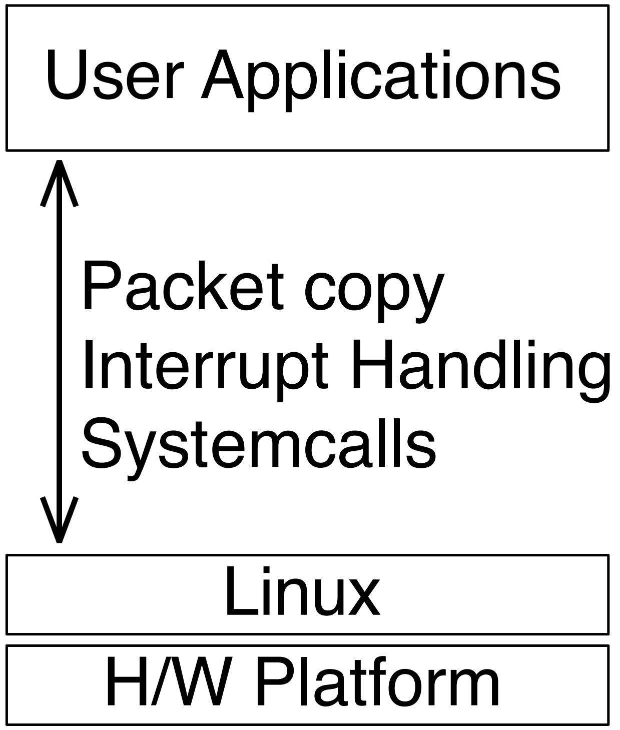
- Mobility management, accounting, security, etc.

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

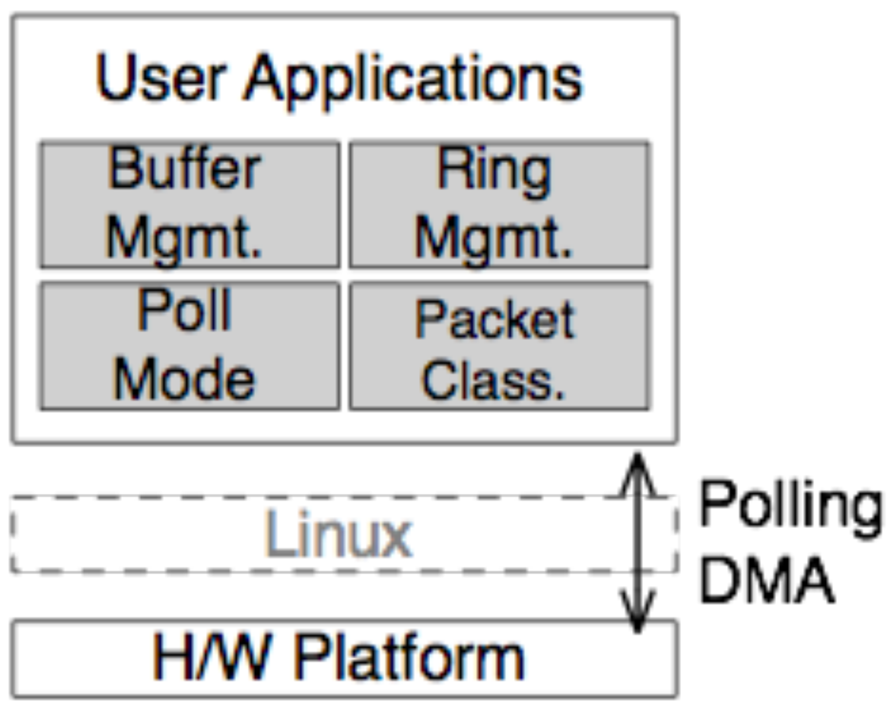


Can you handle being interrupted 60 million times per second?

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

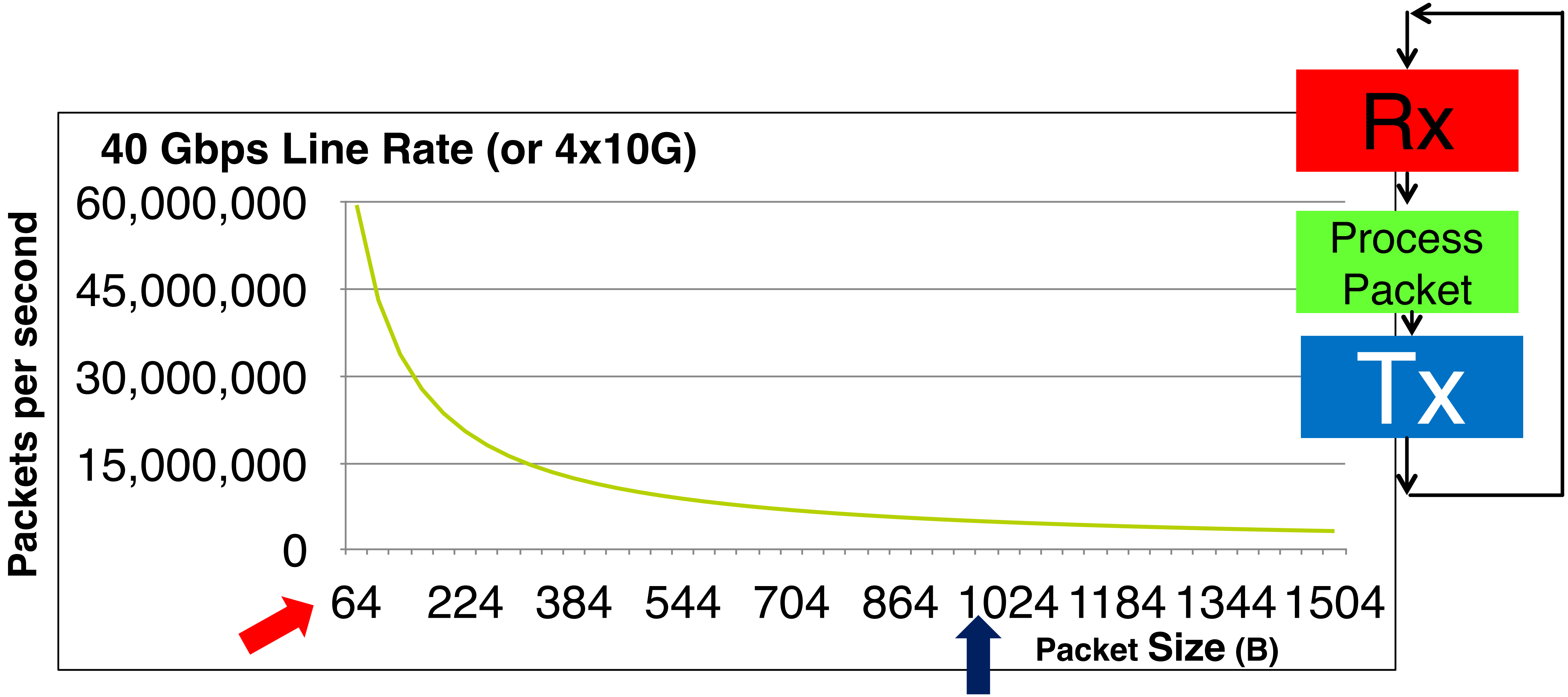
Why try it:

- 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?

Interrupt Context Switch Overhead

Kernel User Overhead

Core To Thread Scheduling Overhead

Polling

User Mode Driver

Pthread Affinity

4K Paging Overhead

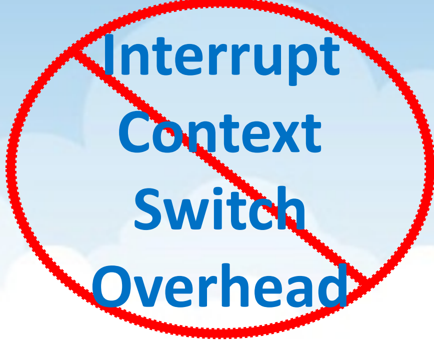
PCI Bridge I/O Overhead

Huge Pages

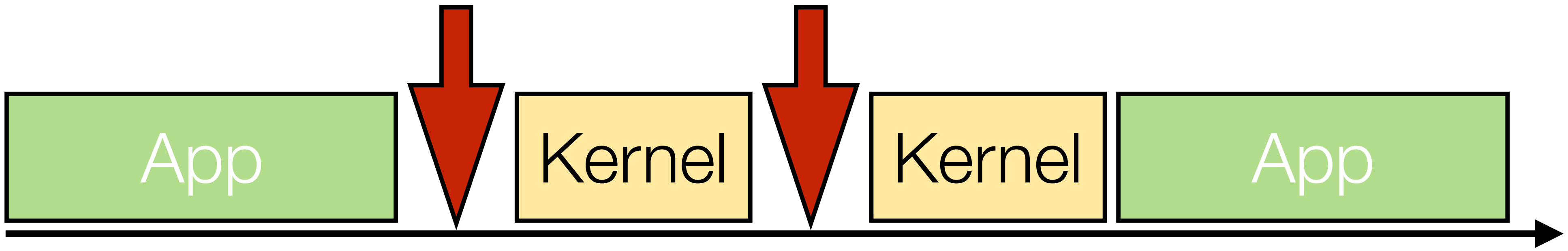
Lockless Inter-core Communication

High Throughput Bulk Mode I/O calls

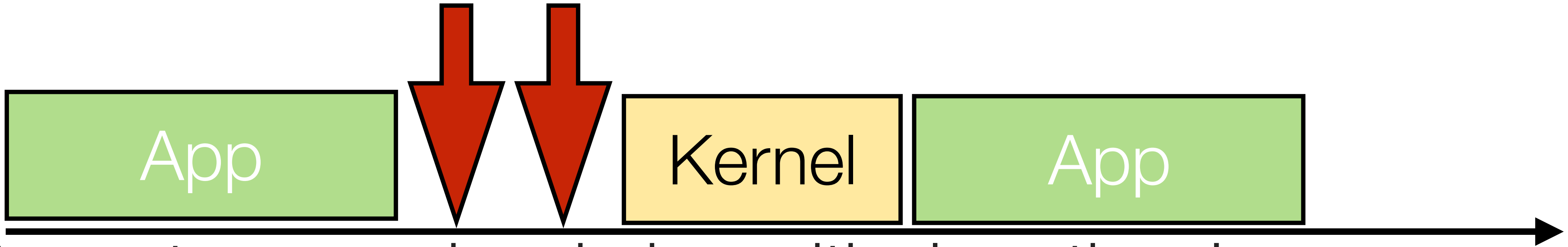
Network Interrupts



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

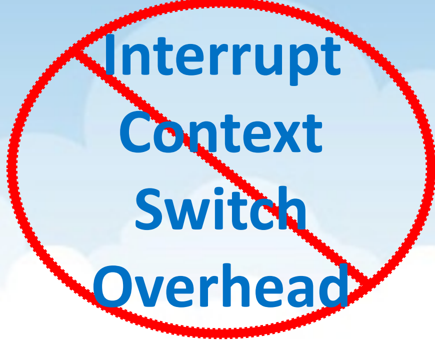


Coalescing interrupts helps, but still causes problems



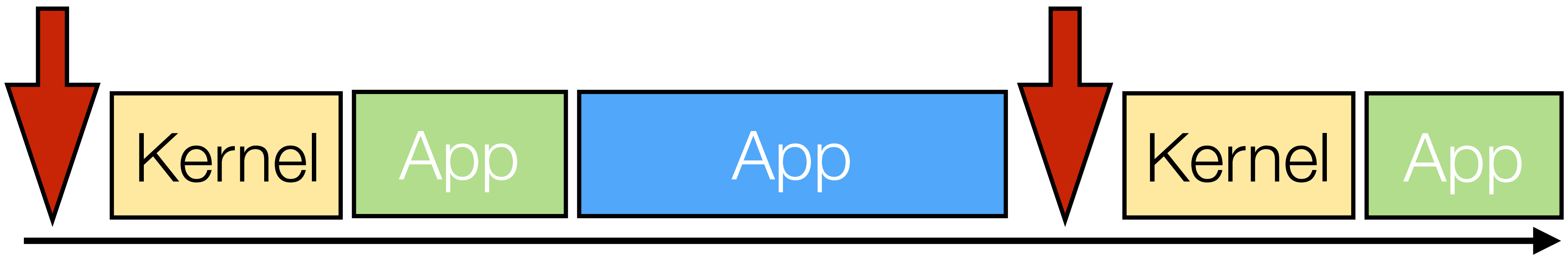
- Interrupts can arrive during critical sections!
- Interrupts can be delivered to the wrong CPU core!
- Still must pay context switch cost

Polling



Continuously loop looking for new packet arrivals

Trade-off?



Interrupts help share the CPU

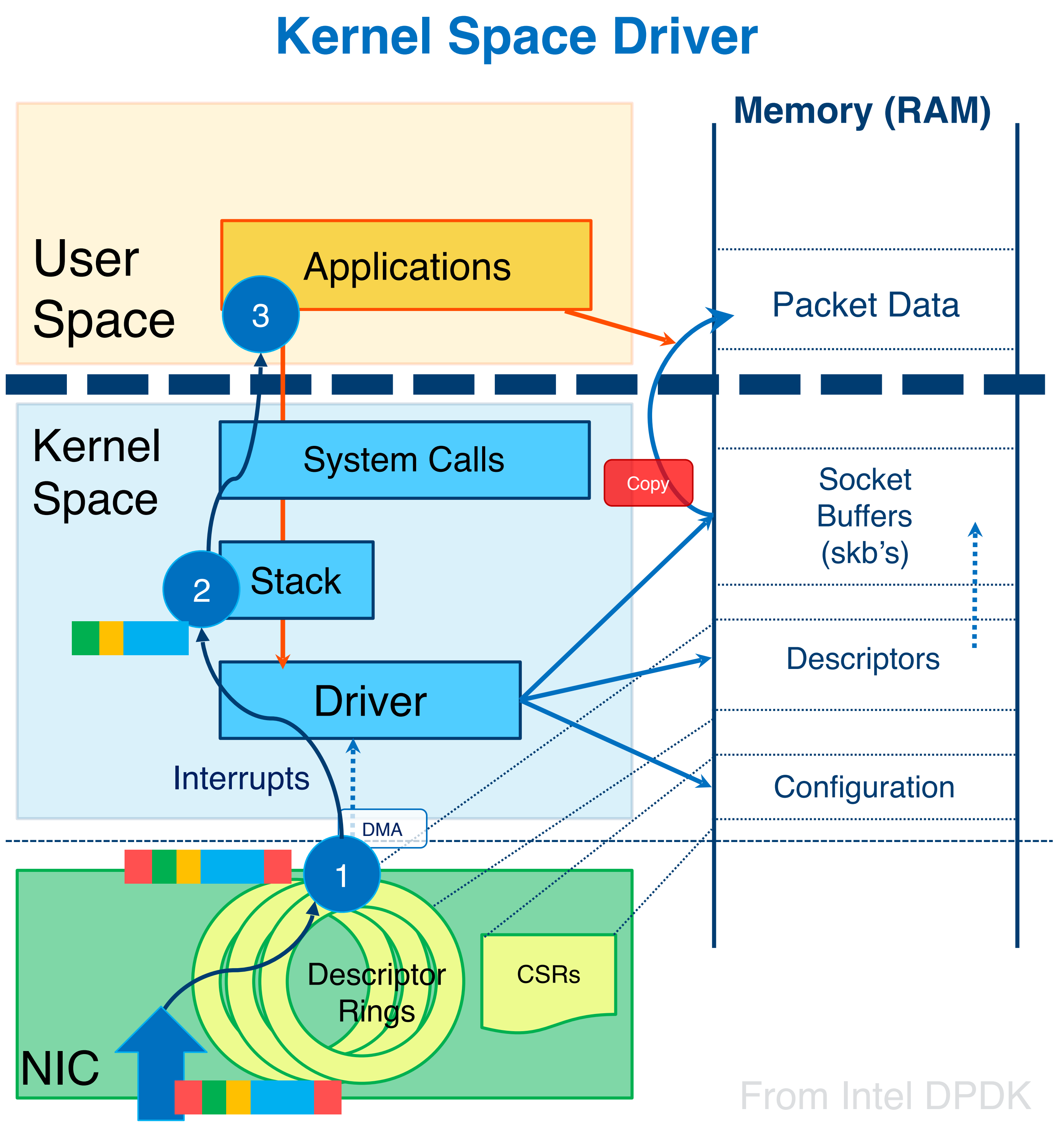
Polling can be wasteful

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?



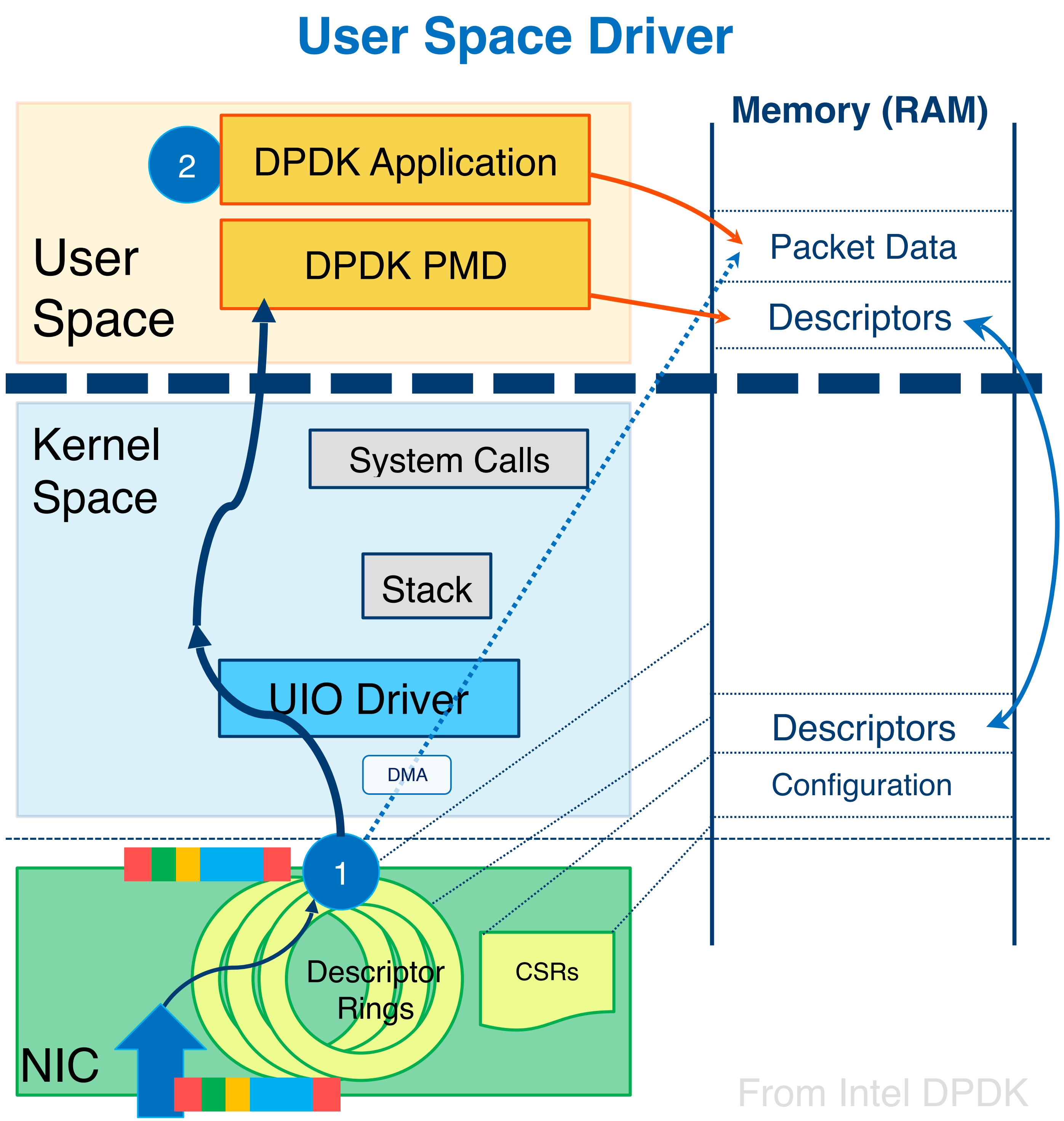
From Intel DPDK University Lecture

Kernel Bypass

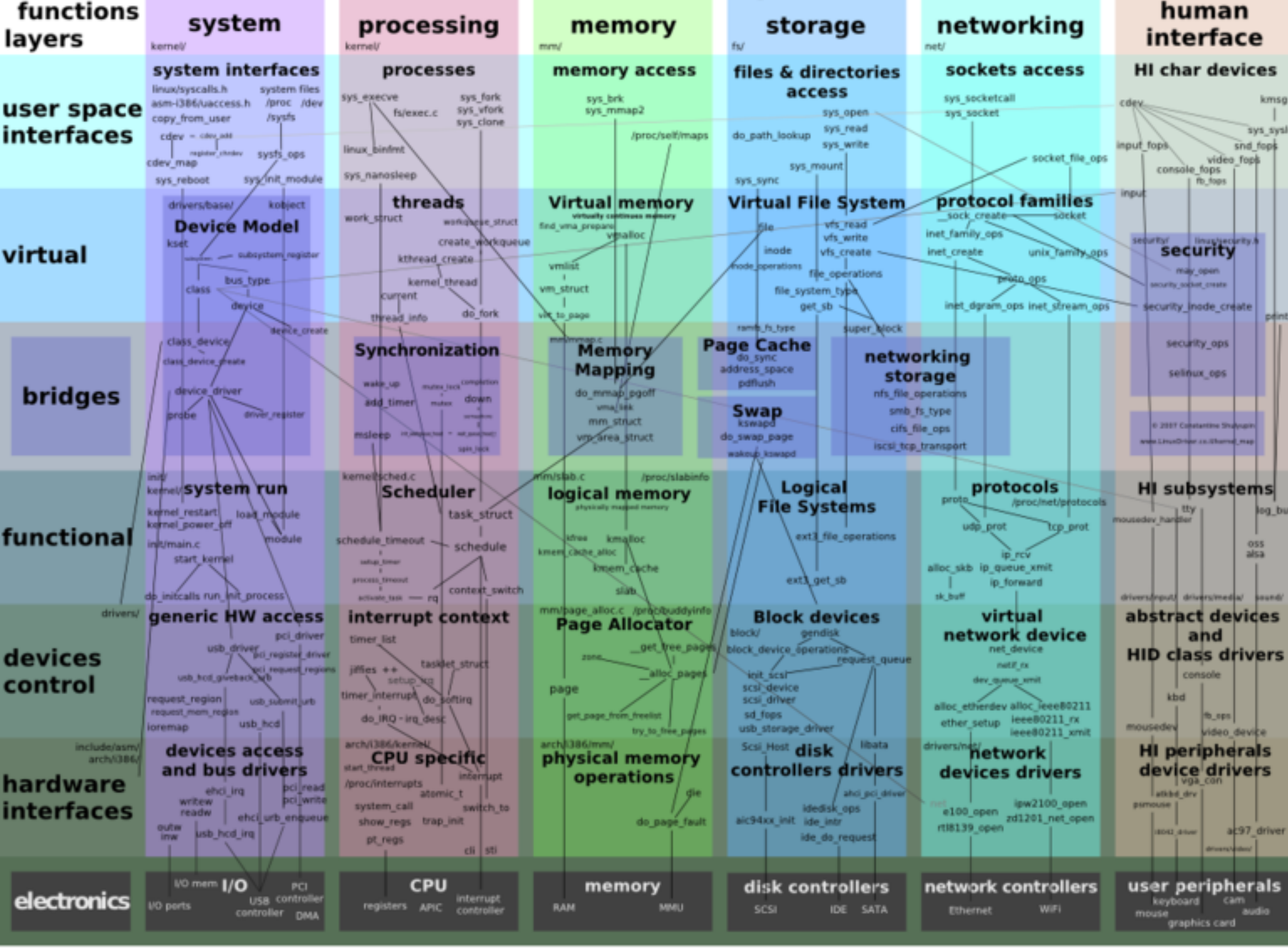
User-mode Driver

- 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
processes
memory access
files & directories access
sockets access
HI char devices
virtual
Device Model
threads
Virtual memory
Virtual File System
protocol families
security
bridges
Synchronization
Memory Mapping
Page Cache
Swap
networking storage
functional
system run
Scheduler
logical memory
Logical File Systems
protocols
HI subsystems
devices control
generic HW access
interrupt context
Page Allocator
Block devices
virtual network device
abstract devices and HID class drivers
hardware interfaces
devices access and bus drivers
CPU specific
physical memory operations
disk controllers drivers
network devices drivers
HI peripherals device drivers
electronics
I/O
CPU
memory
disk controllers
network controllers
user peripherals
sys_execve
sys_fork
sys_vfork
sys_clone
sys_brk
sys_mmap2
sys_open
sys_read
sys_write
sys_socketcall
sys_socket
kobject
bus_type
device
class_device
device_driver
task_struct
schedule
context_switch
kmalloc
kmem_cache
slab
alloc_pages
page
vfs_read
vfs_write
vfs_create
file_operations
inode
super_block
request_queue
tcp_prot
udp_prot
ip_rcv
net_device
do_page_fault
RAM
MMU
SCSI
IDE
SATA
Ethernet
WiFi
keyboard
mouse
graphics card
audio
registers
APIC
I/O ports
DMA

Networking

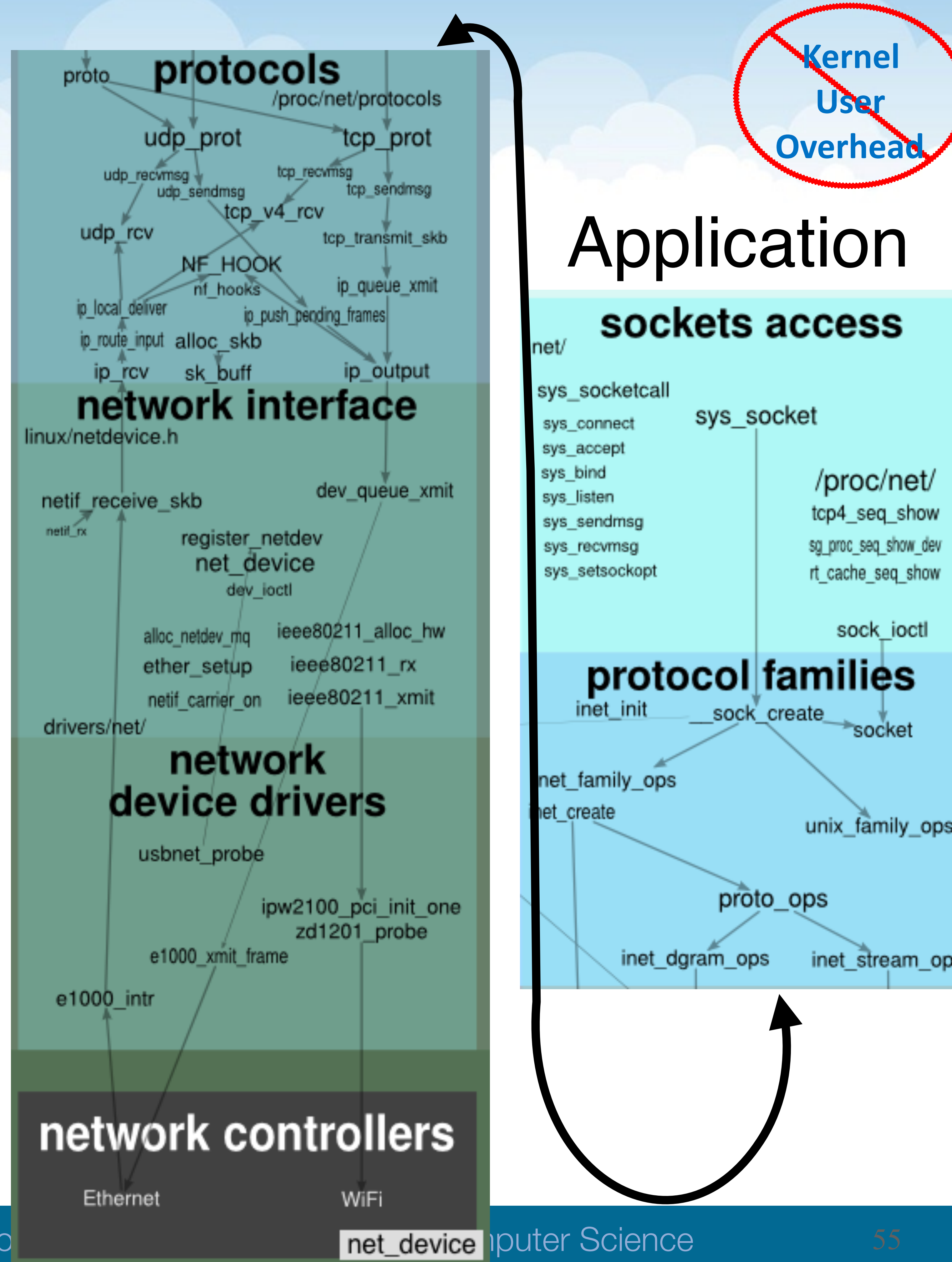
Linux networking stack has a lot of extra components

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

- TCP, UDP, sockets

NFV middle boxes just need packet data

- Need it fast!

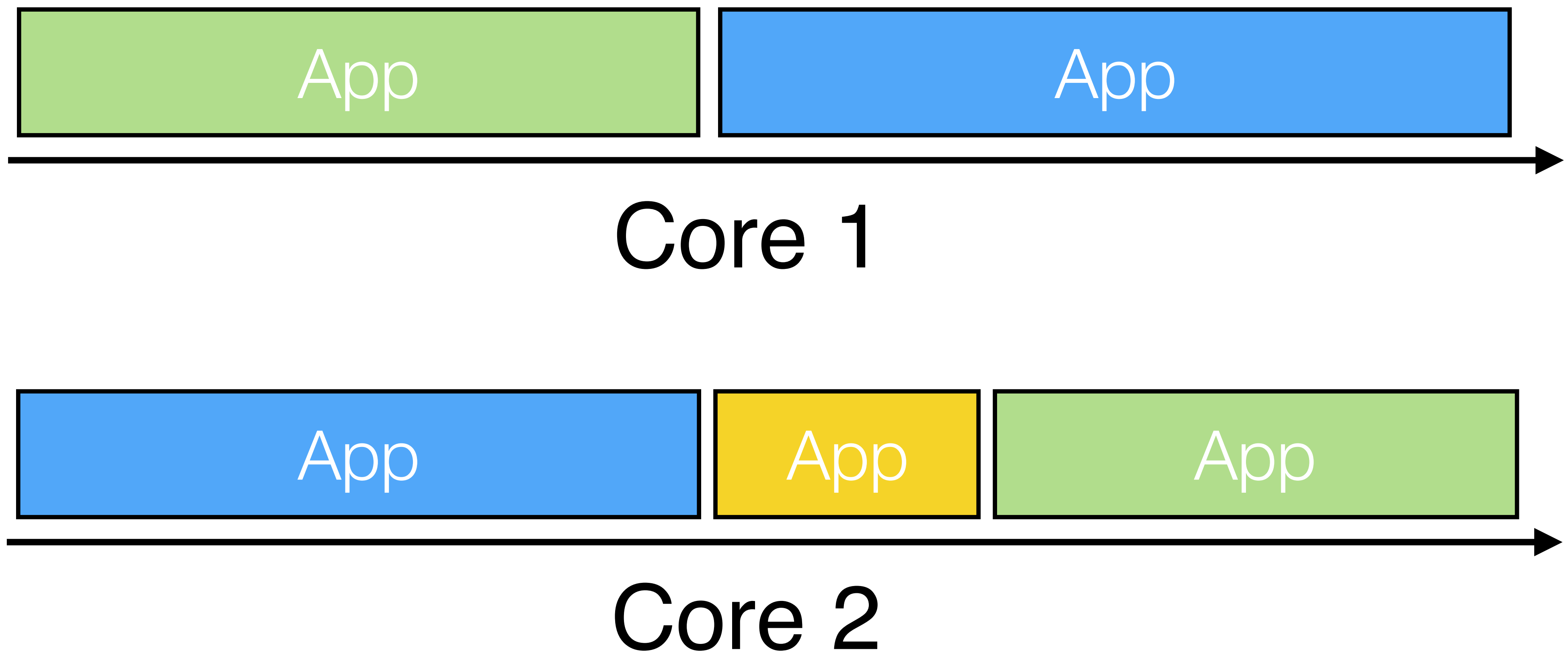


CPU Core Affinity

Core To Thread Scheduling Overhead

Linux Scheduler can move threads between cores

- Context switches :(
- Cache locality :(

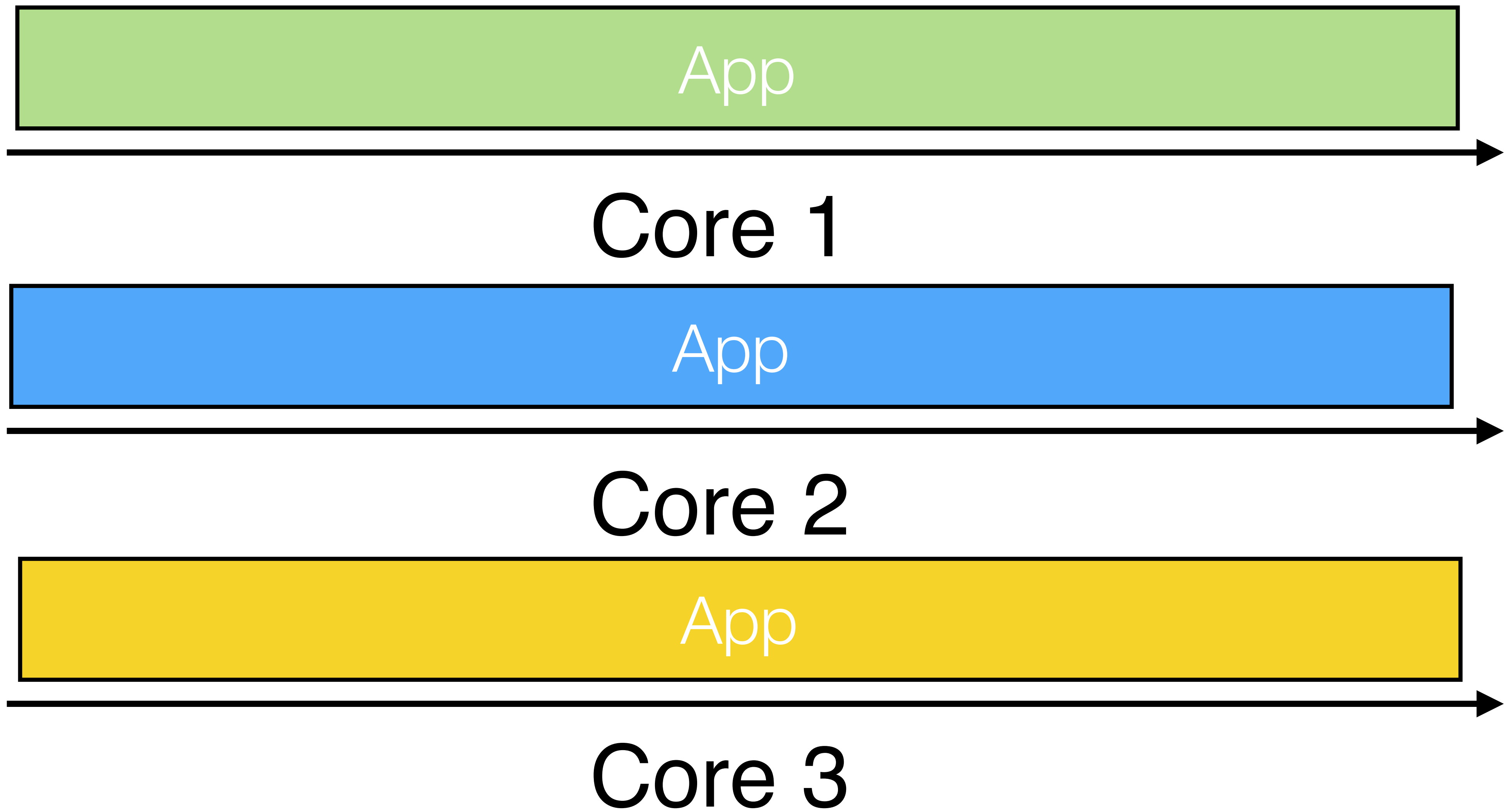


CPU Core Affinity

Core To Thread Scheduling Overhead

Pin threads and dedicate cores

- **Trade-offs?**



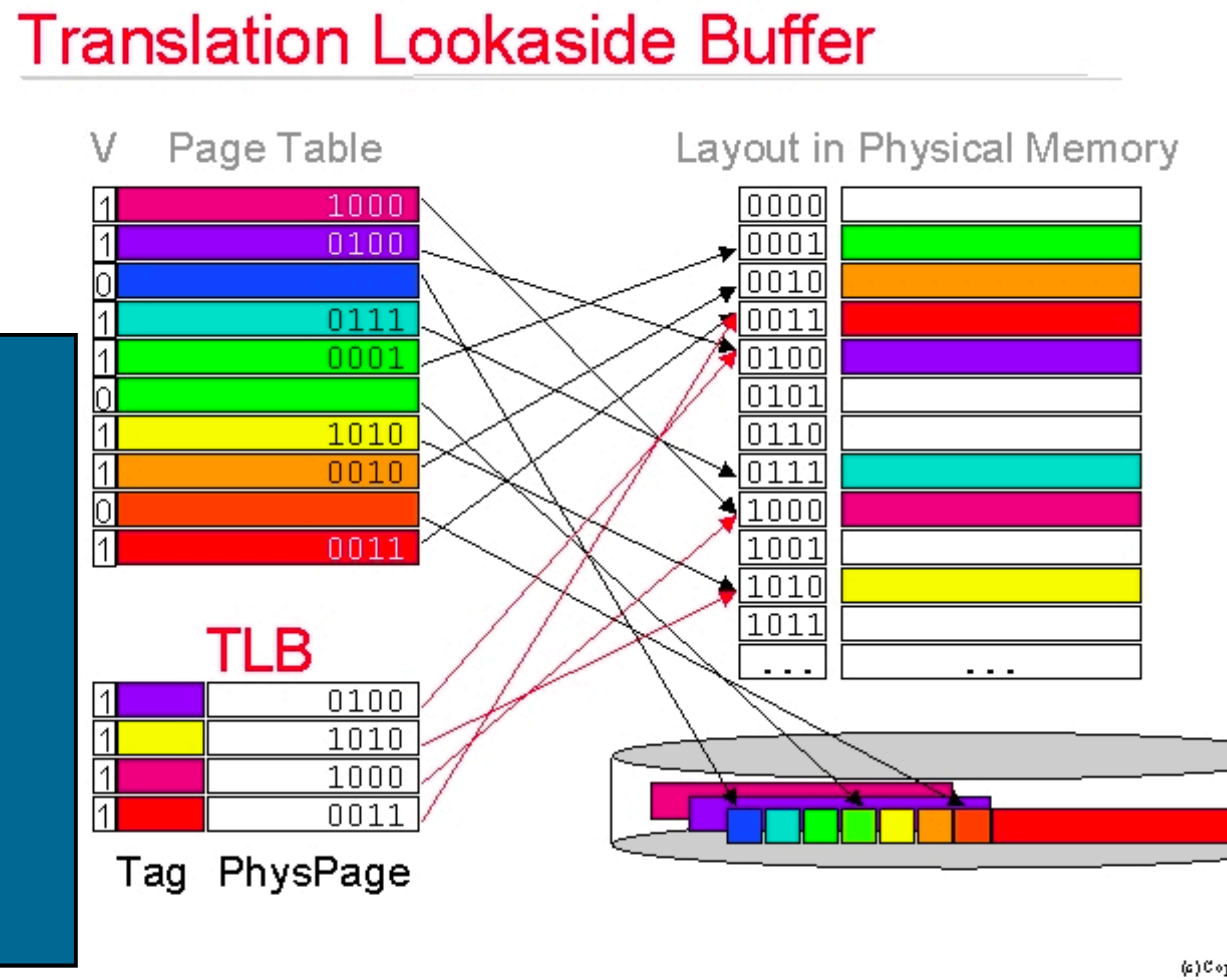
Paging Overhead

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

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

PCI Bridge I/O Overhead

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?

Limitations

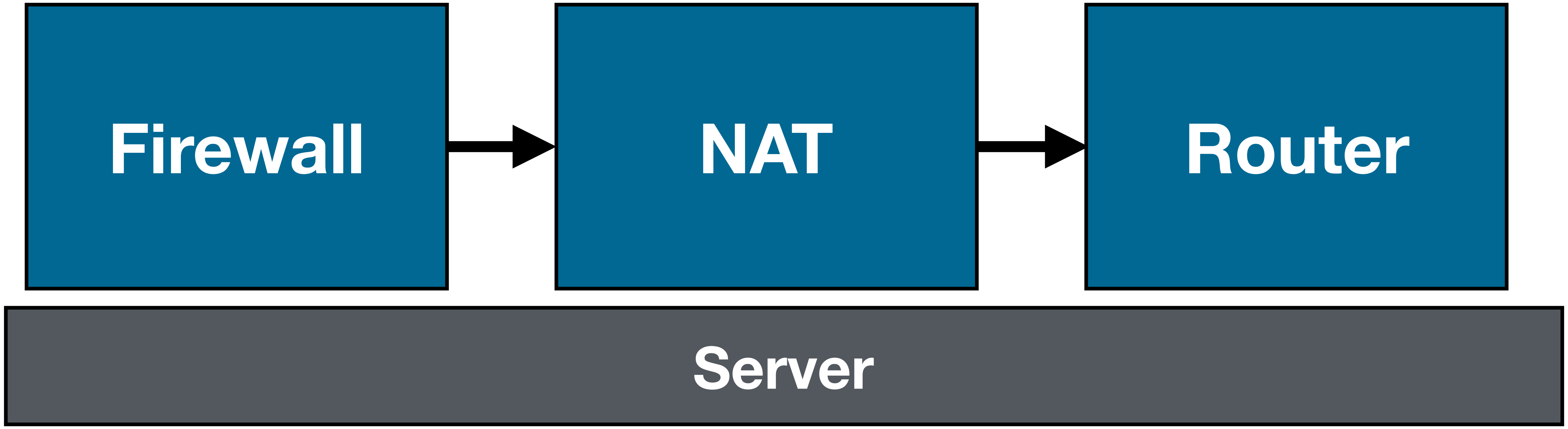
DPDK provides efficient I/O… but that's about it

Doesn't help with NF management or orchestration

Service Chains

Chain together functionality to build more complex services

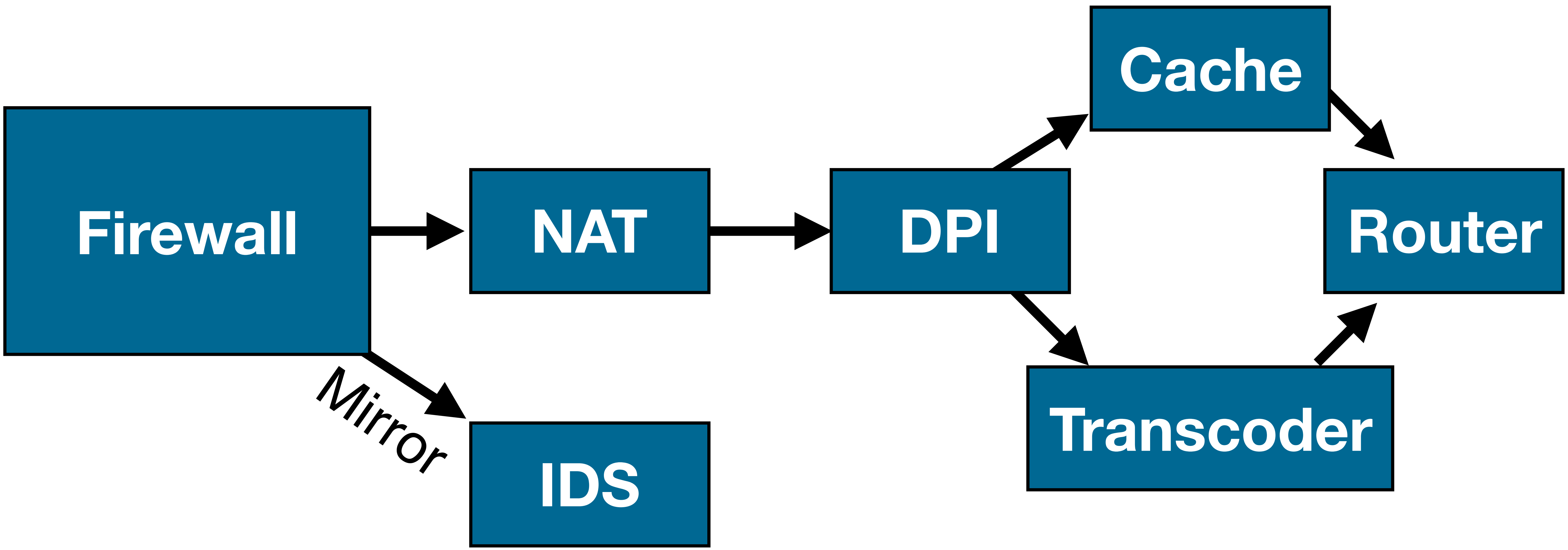
- Need to move packets through chain efficiently



Service Chains

Chain together functionality to build more complex services

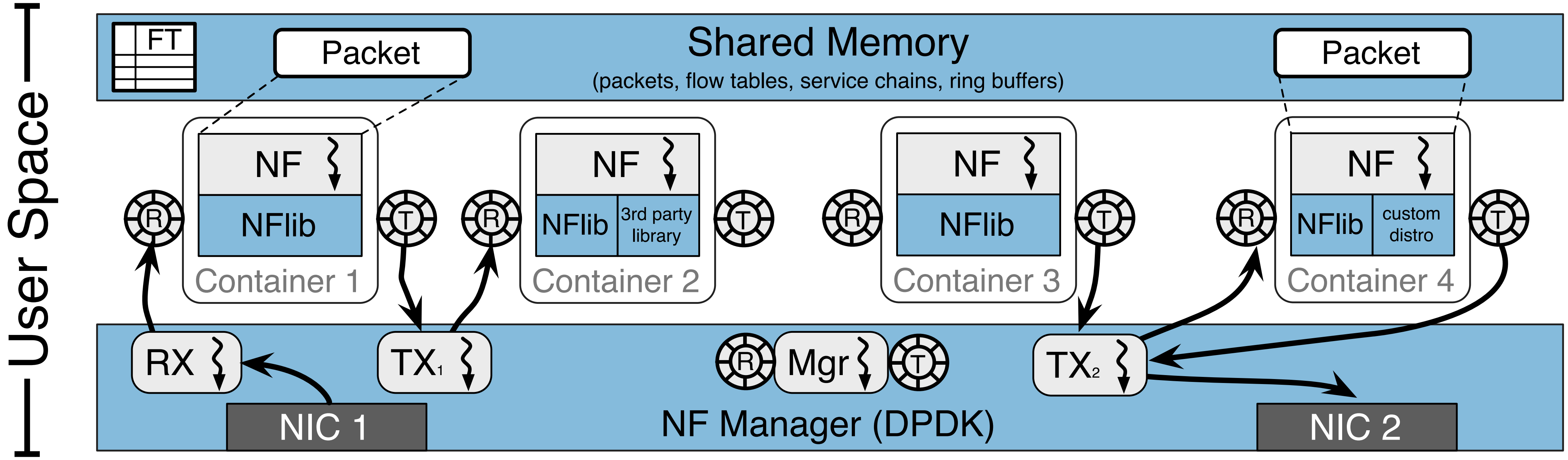
- Need to move packets through chain efficiently



Can be complex with multiple paths!

OpenNetVM NFV Platform

Made at GW!



DPDK: provides underlying I/O engine

NFs: run inside Docker container, use NFlib API

Manager: tracks which NFs are active, organizes chains

Shared memory: efficient communication between NFs

SDN-aware: Controller can dictate flow rules for NFs

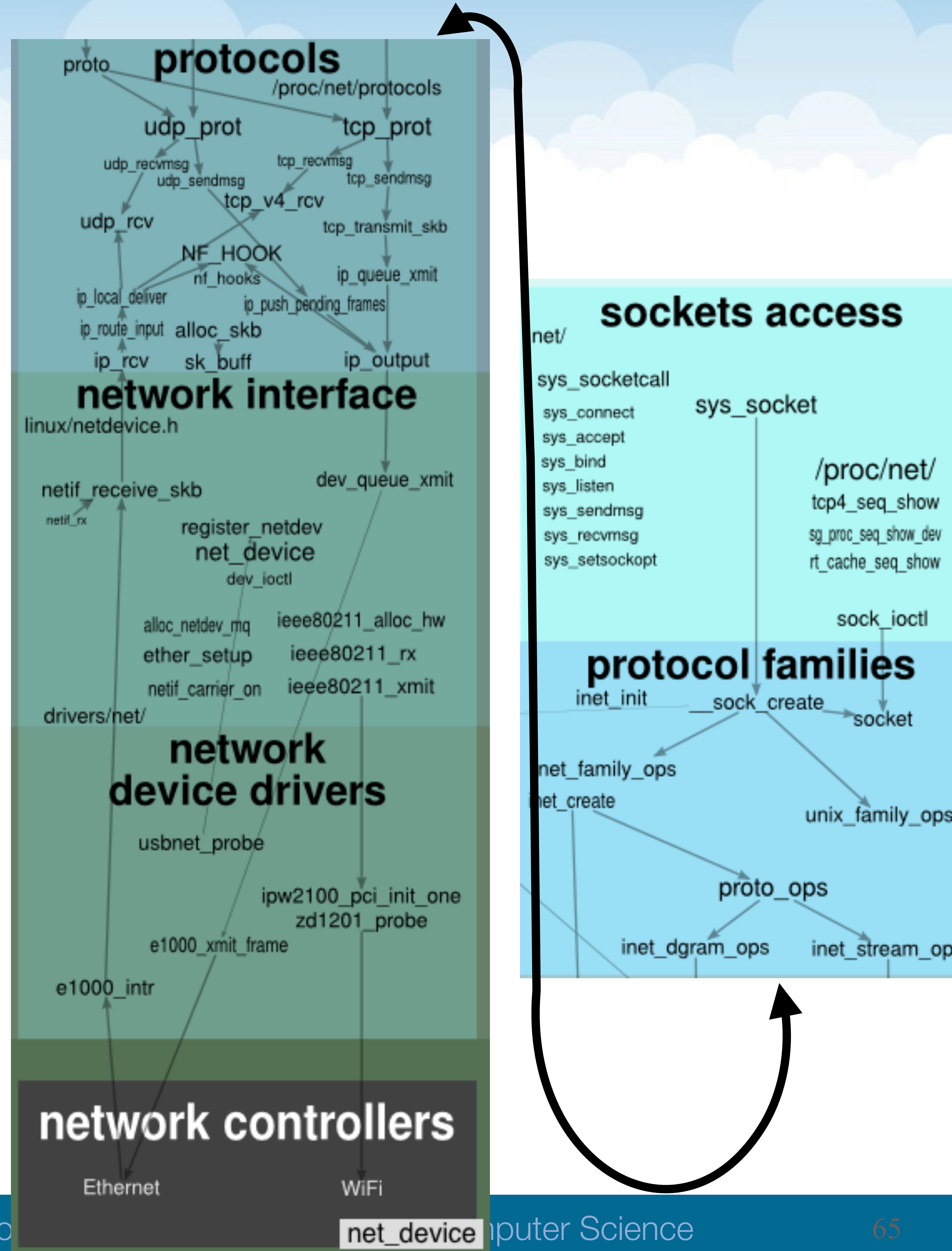
http://sdnfv.github.io/

Limitations

DPDK only helps with raw packet IO

Doesn't provide any protocol stacks!

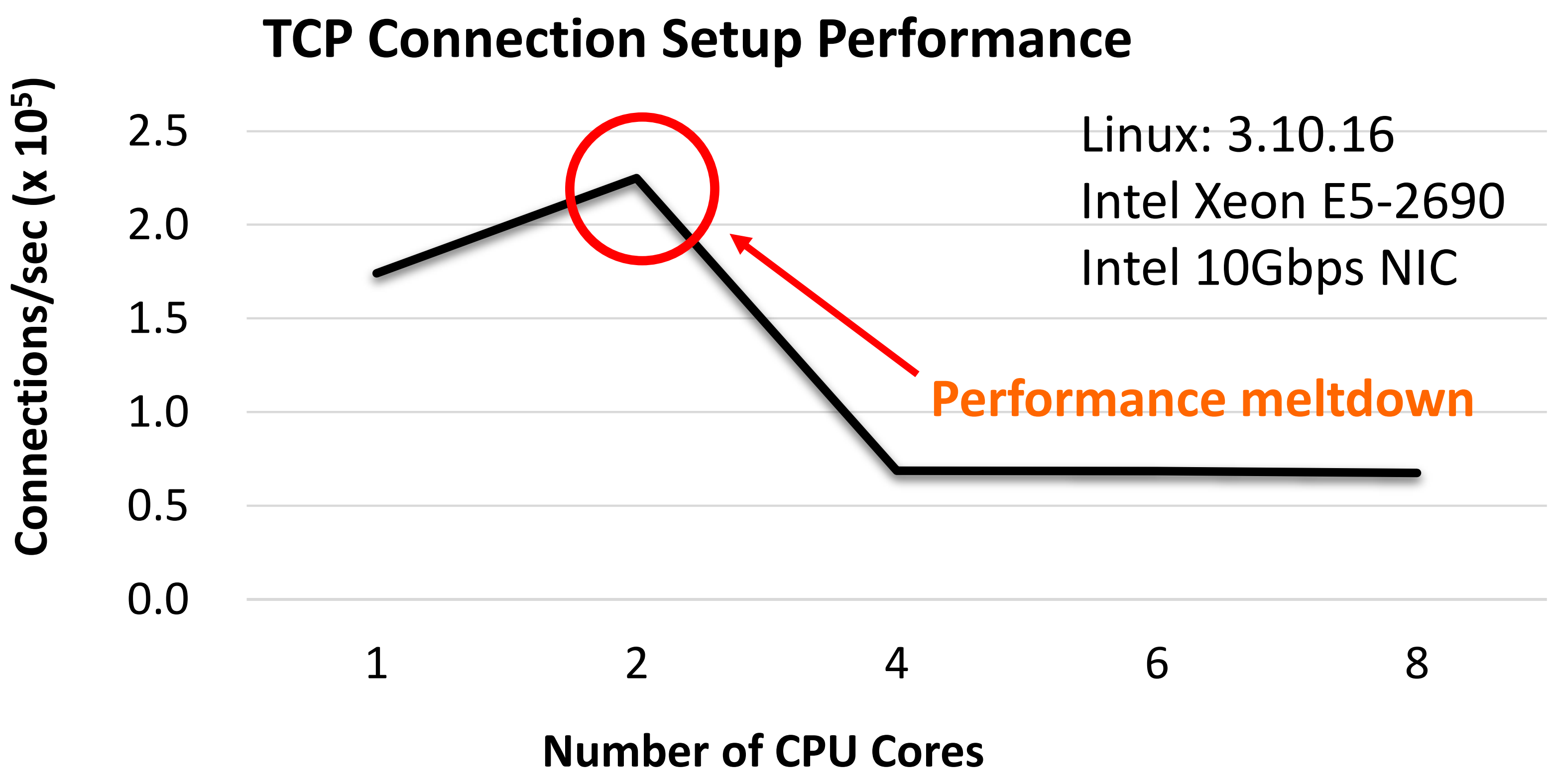
- No IP
- No TCP or UPD
- No socket interface

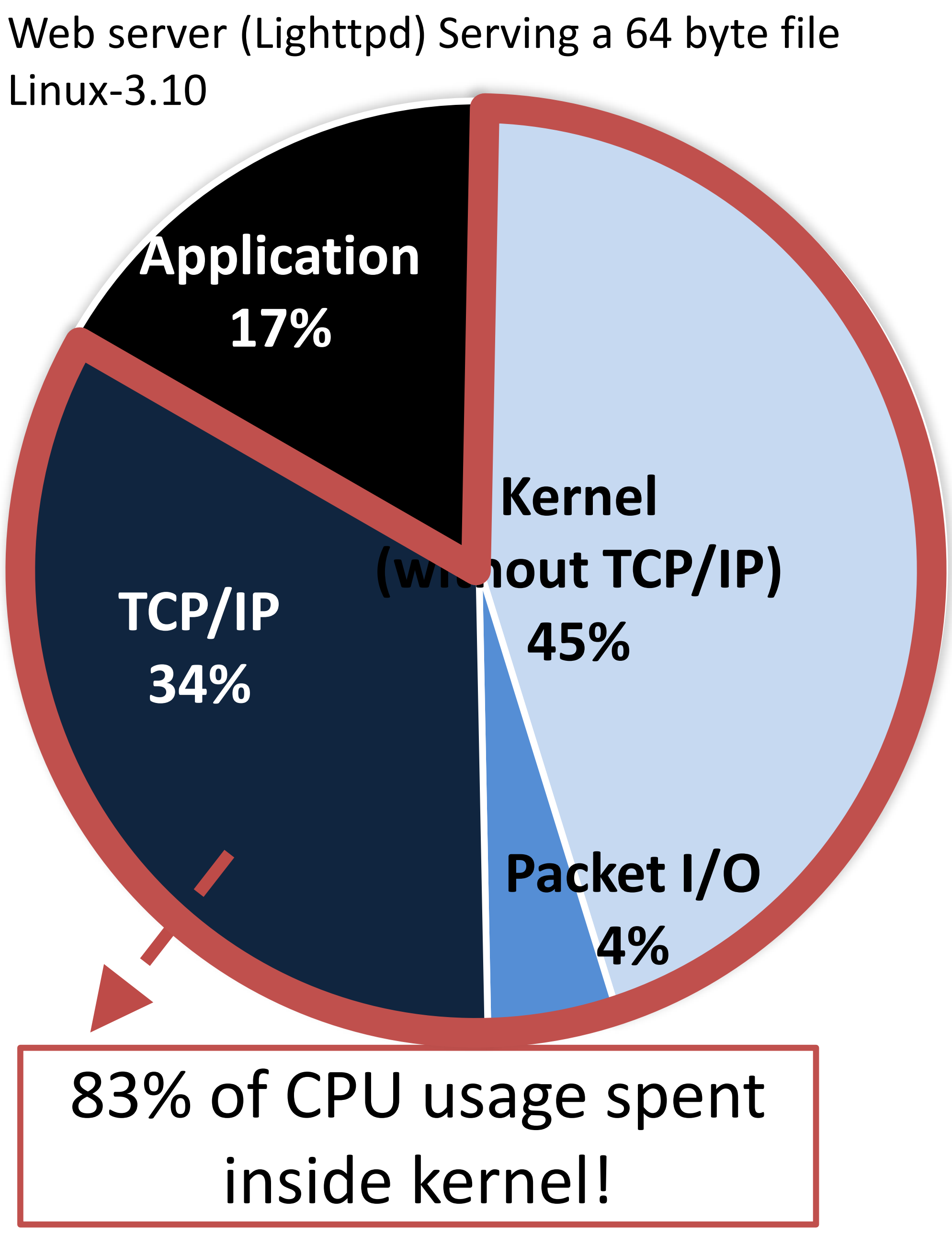


TCP in Linux

Linux TCP stack is not designed for high performance

- Especially for short flows
- Poor scalability, bad locality, etc
- Same problems we saw with DPDK





Figures from Jeong's mTCP talk at NSDI 14

mTCP [Jeong, NSDI '14]

User space TCP stack

- Built on DPDK/netmap (and now OpenNetVM!)

Key Ideas:

- Eliminate shared resources by partitioning flows to independent threads
- Use batching to minimize overheads
- Epoll interface to support existing end-point applications

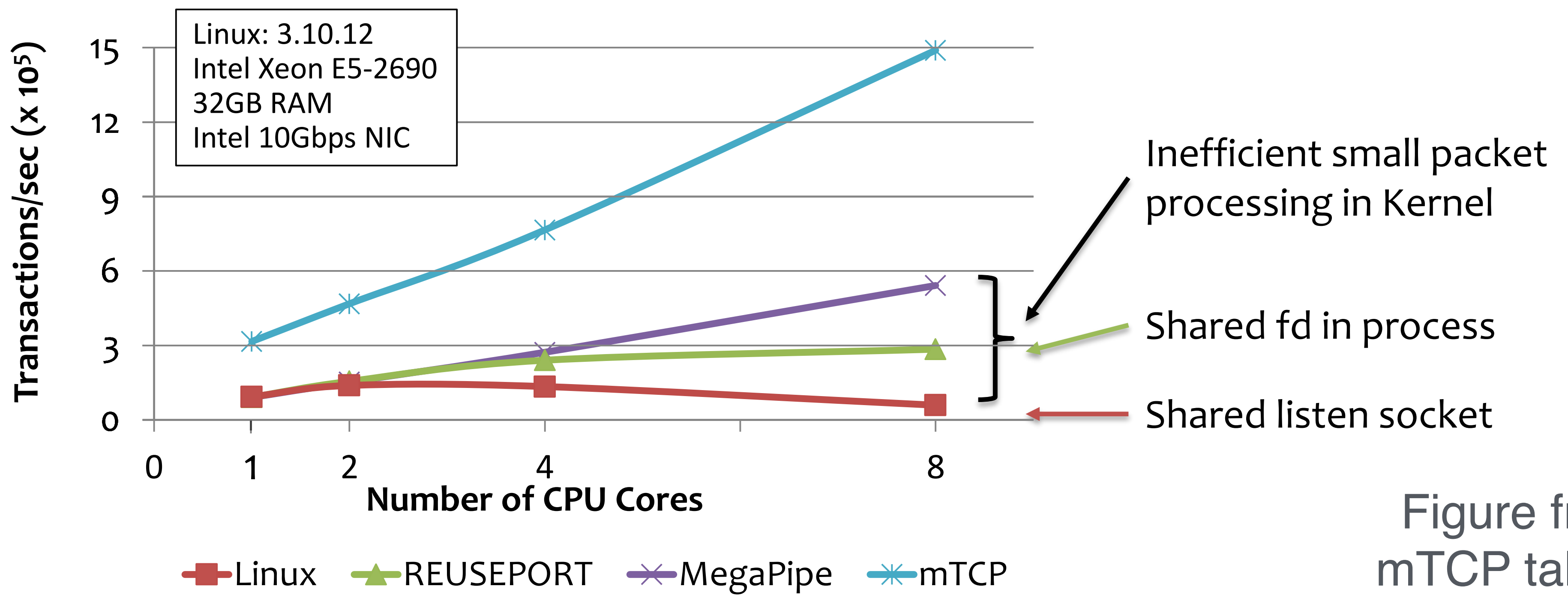


Figure from Jeong's mTCP talk at NSDI 14

mTCP Kernel Bypass

Responding to a packet arrival only incurs a context switch, not a full system call

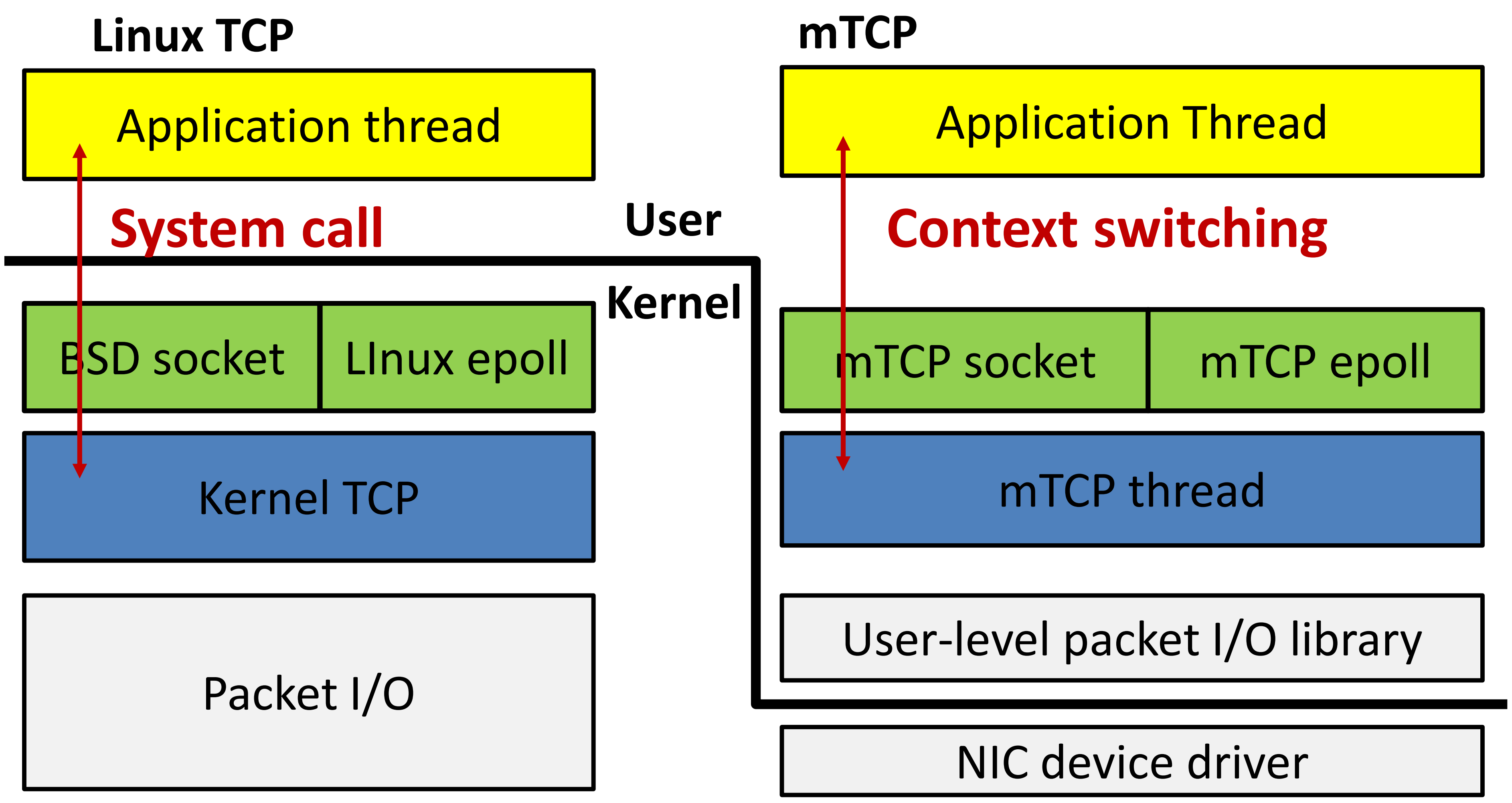


Figure from Jeong's mTCP talk at NSDI 14

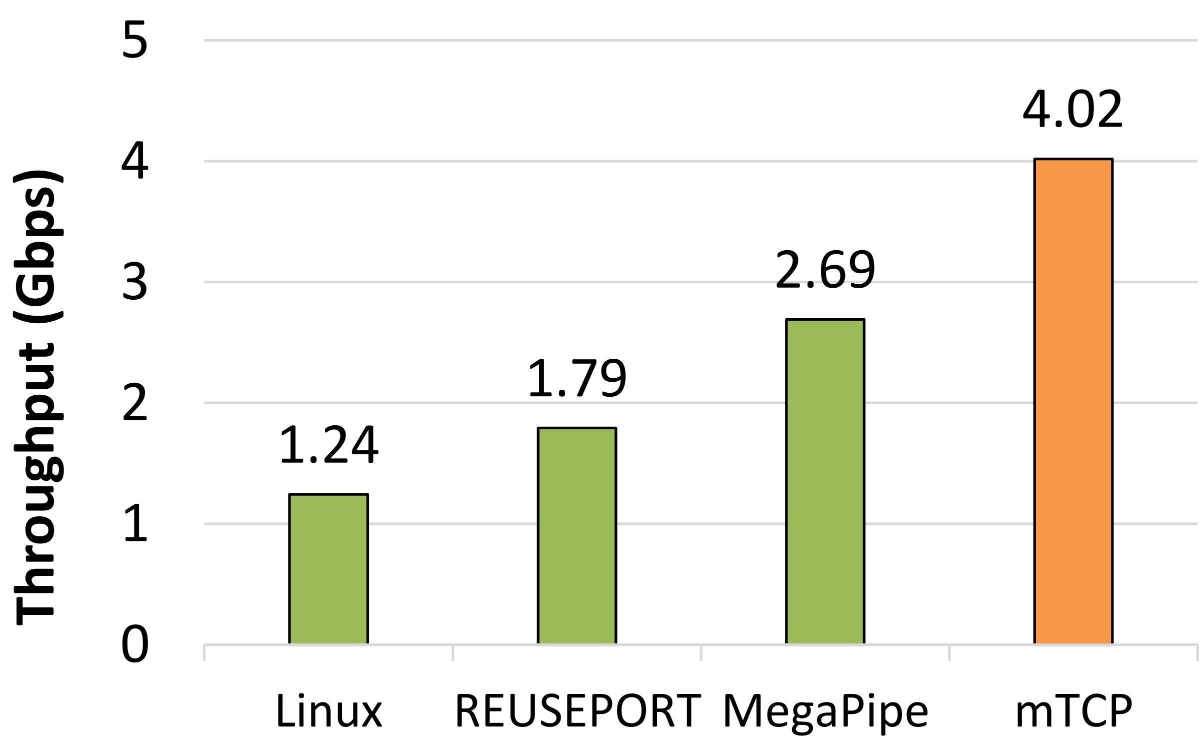
Performance Improvement on Ported Applications

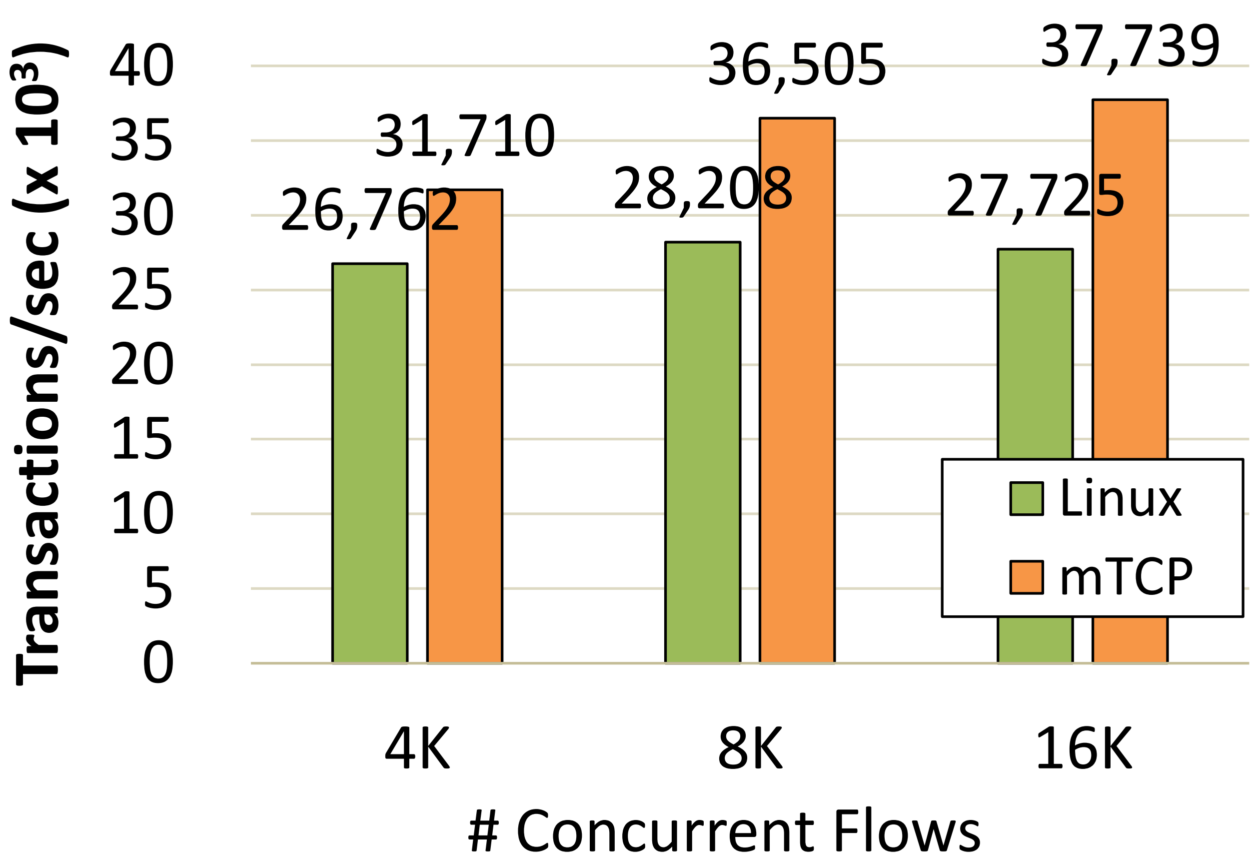
Web Server (Lighttpd)

- Real traffic workload: Static file workload from SpecWeb2009 set
- **3.2x** faster than Linux
- **1.5x** faster than MegaPipe

SSL Proxy (SSLShader)

- Performance Bottleneck in TCP
- Cipher suite
  1024-bit RSA, 128-bit AES, HMAC-SHA1
- Download 1-byte object via HTTPS

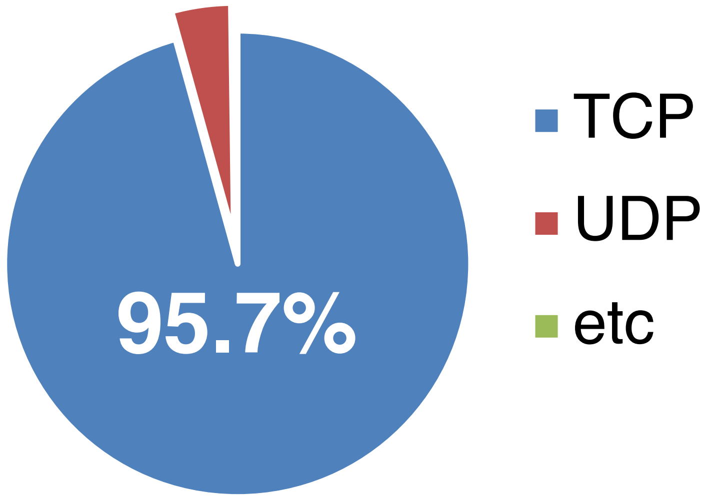




Slide from Jeong's mTCP talk, NSDI '14

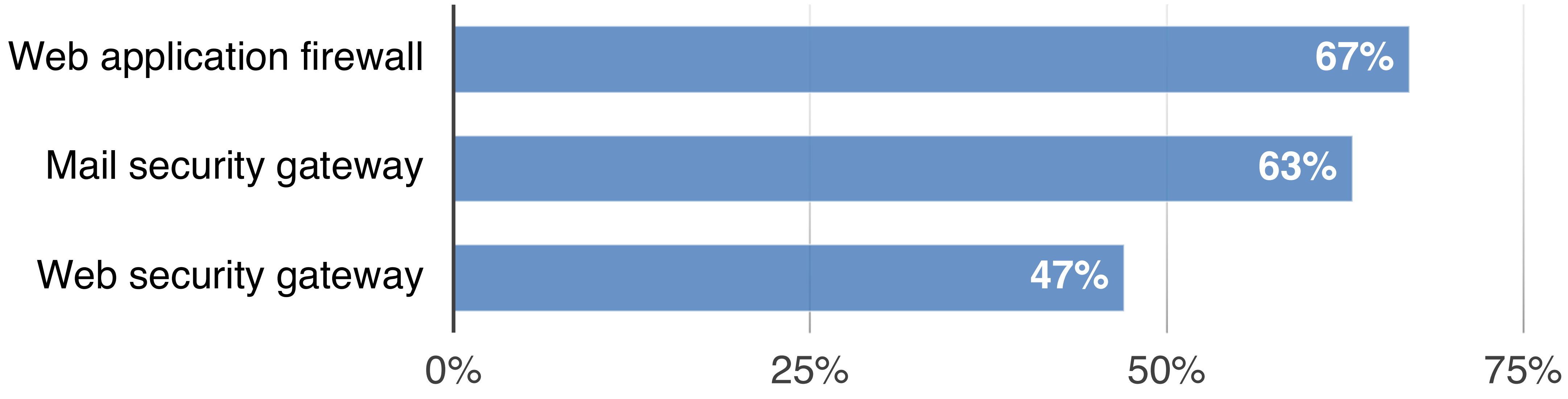
Most Middleboxes Deal with TCP Traffic

- TCP dominates the Internet
  - 95+% of traffic is TCP [1]



- Top 3 middleboxes in service providers rely on L4/L7 semantics

Virtual Appliances Deployed in Service Provider Data Centers [2]



[1] "Comparison of Caching Strategies in Modern Cellular Backhaul Networks", ACM MobiSys 2013.
[2] IHS Infonetics Cloud & Data Center Security Strategies & Vendor Leadership: Global Service Provider Survey, Dec. 2014.

2

Slide from Jamshed's mOS talk, NSDI '17

mOS [Jamshed, NSDI ‘17]

What if your middle box (not end point server) needs TCP processing?

Proxies, L4/L7 load balancers, DPI, IDS, etc

- TCP state transitions
- Byte stream reconstruction

Borrow code from open-source IDS (e.g., snort, suricata)	• 50K~100K code lines tightly coupled with their IDS logic
Borrow code from open-source kernel (e.g., Linux/FreeBSD)	• Designed for TCP end host • Different from middlebox semantics
Implement your own flow management code	• Complex and error-prone • ***Repeat*** it for every custom middlebox

Table from Jamshed’s mOS talk, NSDI ‘17

mOS [Jamshed, NSDI '17]

Reusable protocol stack for middle boxes

Key Idea: Allow customizable processing based on flow-level "events"

Separately track client and server side state

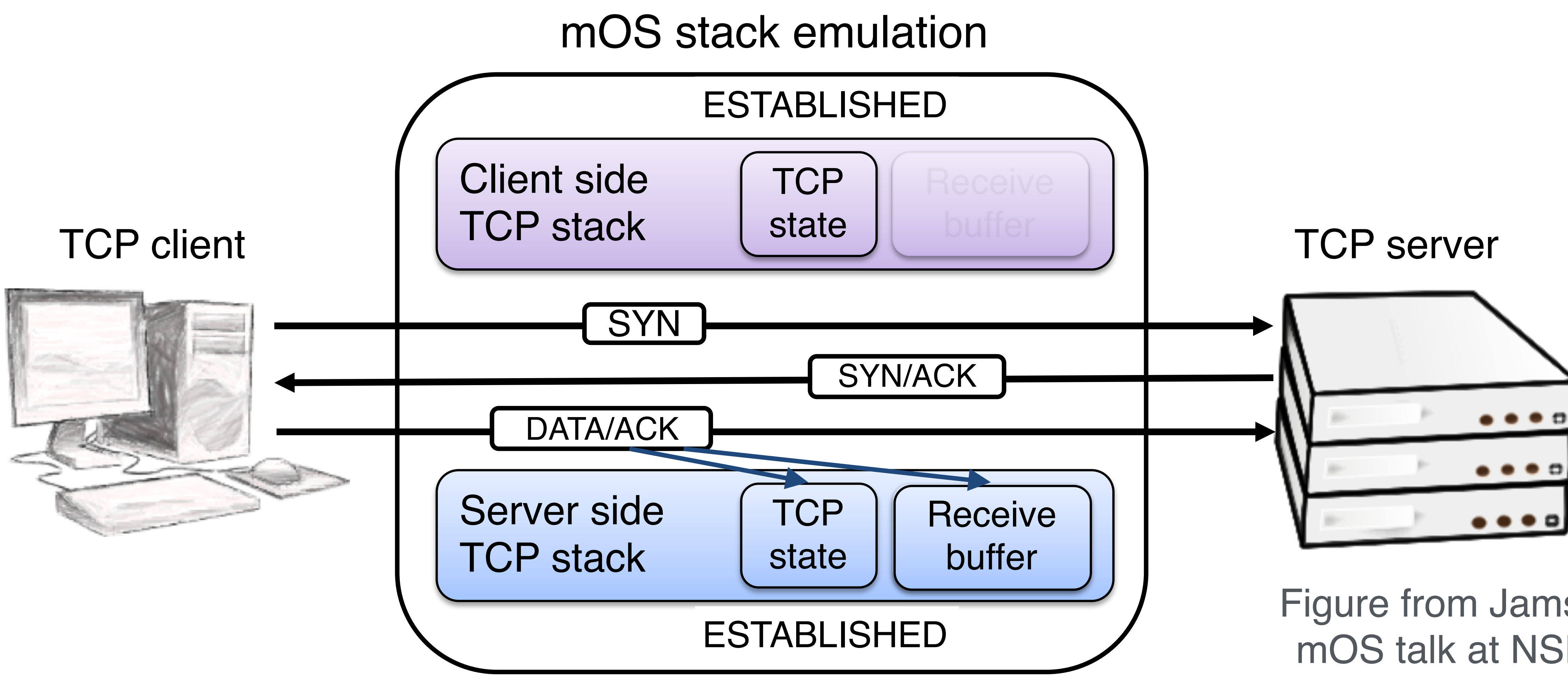


Figure from Jamshed's mOS talk at NSDI 17

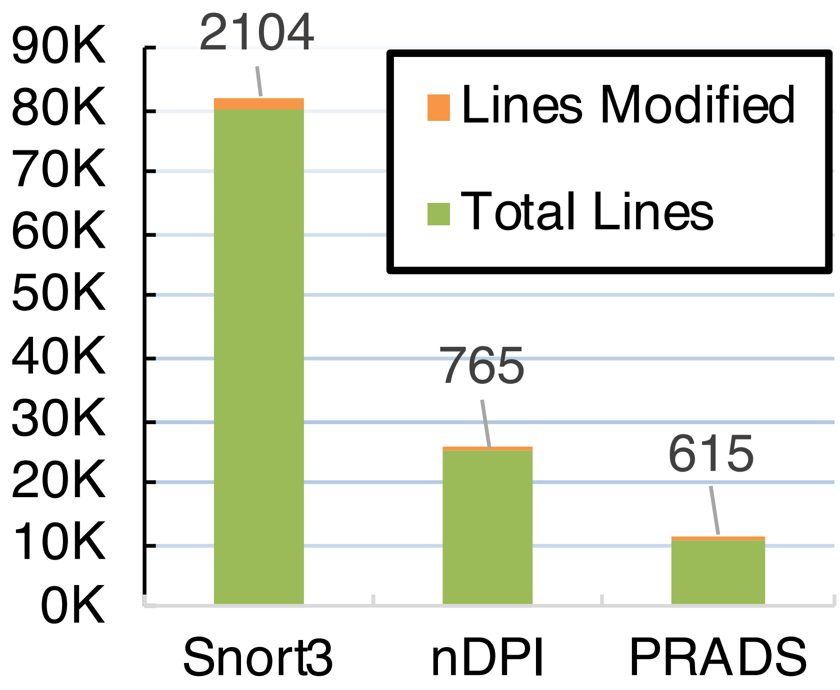
mOS [Jamshed, NSDI '17]

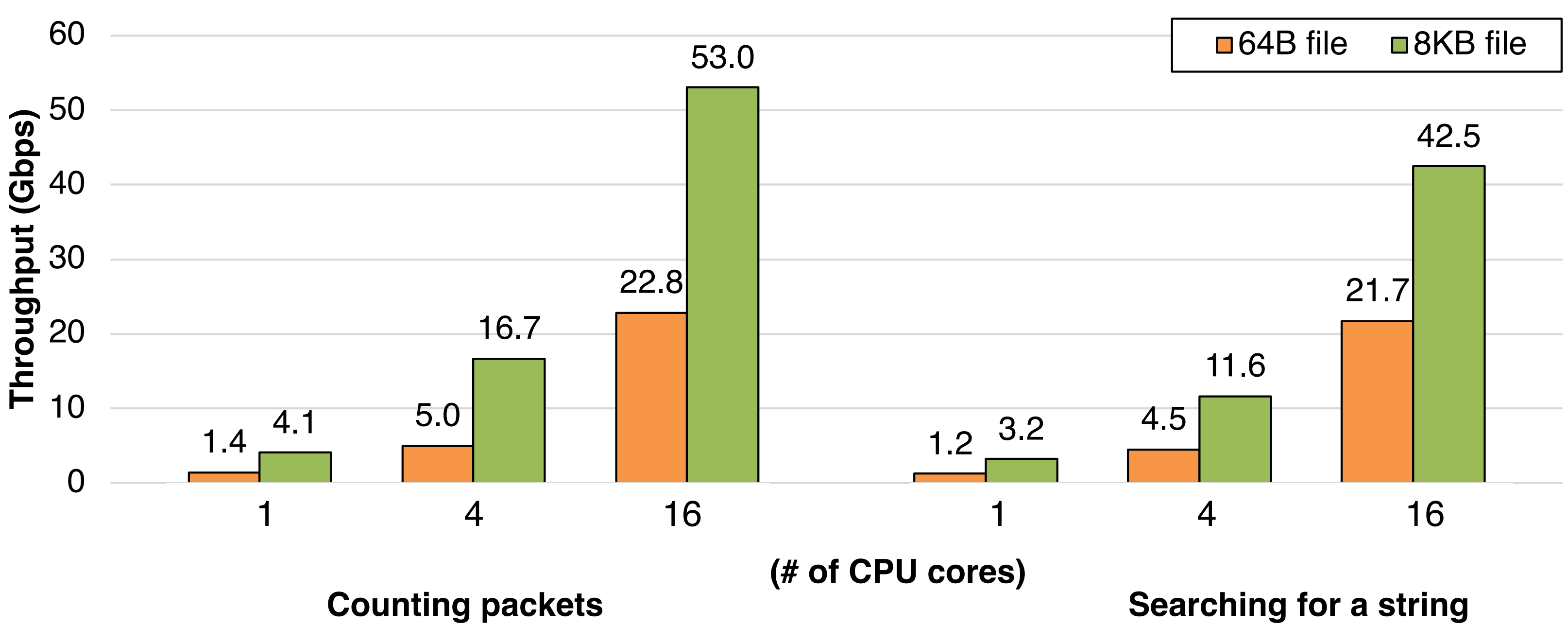
Base Events

- TCP connection start/end, packet arrival, retransmission, etc

User Events

- Base event + a filter function (executable code) run in mOS stack





Figures from Jamshed's mOS talk at NSDI 17

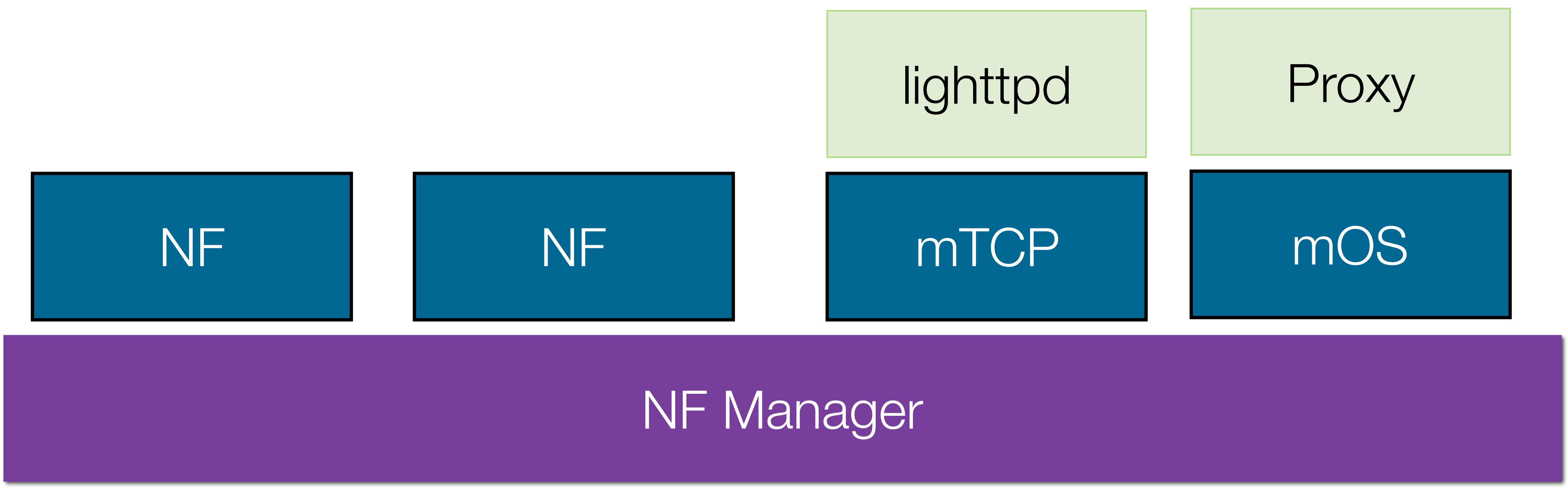
TCP + OpenNetVM

Made at GW!

We have ported mOS/mTCP to run on OpenNetVM

Allows deployment of mixed NFs and endpoints

Allows several different mTCP endpoints on same host

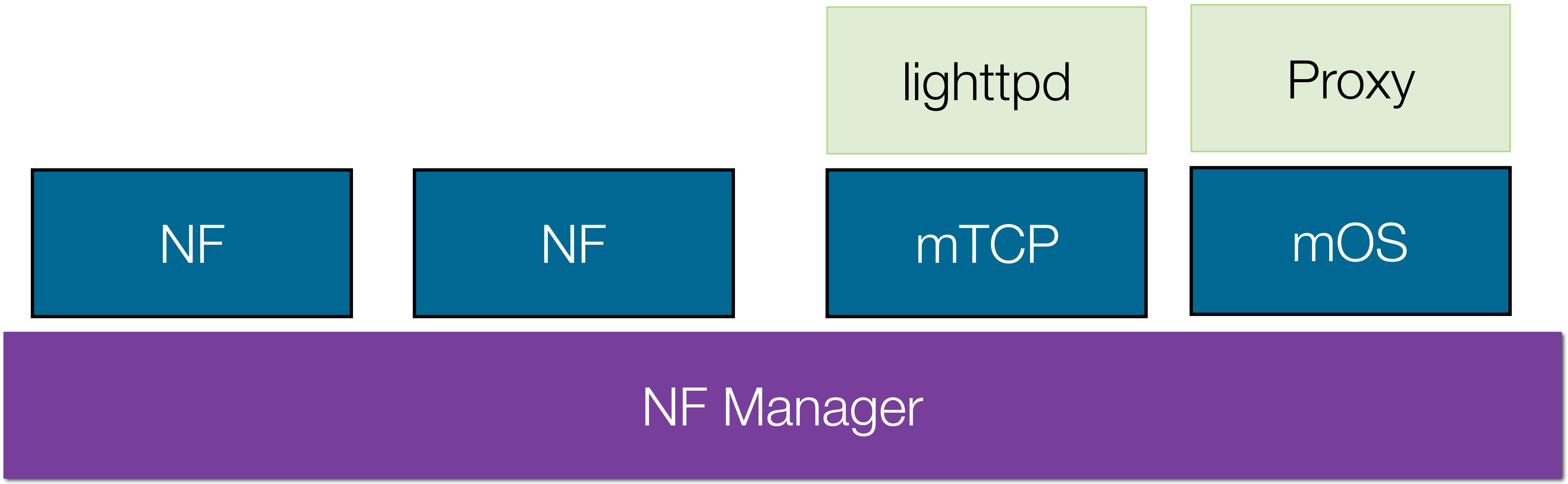


TCP + OpenNetVM

Made at GW!

Mixed NFs + endpoints blurs the line of the application and the network

- NF services could expose APIs to work with endpoints



Microboxes

Made at GW!

- × Redundant Stack Processing
- × A Monolithic Stack
- × Separate Stacks/Interfaces

⇒

- ✓ Consolidate Stack Processing
- ✓ Customizable Stack Modules
- ✓ Unified Event Interface

Microboxes

= µStack + µEvent

= stack snapshot + parallel stacks
 + parallel events + event hierarchy
 + publish/subscribe interface