Advanced Networking and Distributed Systems

Module 3: Network Middleboxes

GW CSCI 3907/6907
Timothy Wood and Lucas Chaufournier

Upcoming

Tuesday 2/18

- Class: lecture on network middleboxes, info about midterm
- **DUE 11:59pm**: submit code review of another group's PR (don't need to fix anything yet, just comment on another group)

Thursday 2/20

- **DUE 11:59pm**: tech blog

Tuesday 2/25

- Class: intro to distributed systems and **MIDTERM**

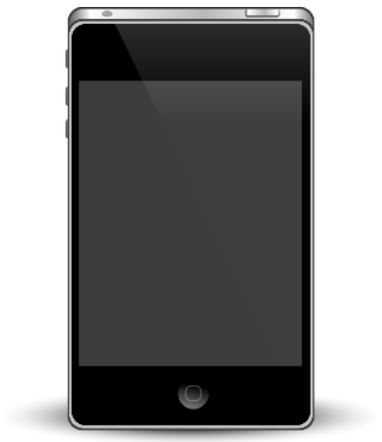
Sunday 3/1

- **DUE 11:59pm**: Corrected code for your PR due (fix your own PR by this date)

Tuesday 3/3

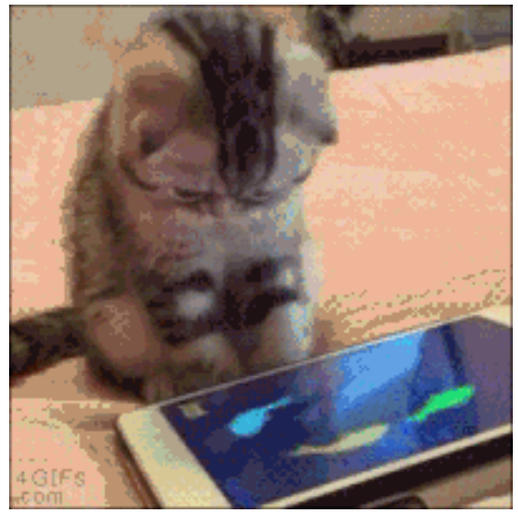
- Class: more on distributed systems

Middleboxes



Client

Middlebox



Server

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 (Evolved Packet Core - EPC, 5G)

- Mobility management, accounting, security, etc.

Proxies, caches, load balancers, etc.

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

$33,650.99

$5.99 Shipping

ADD TO CART ▶

Compare

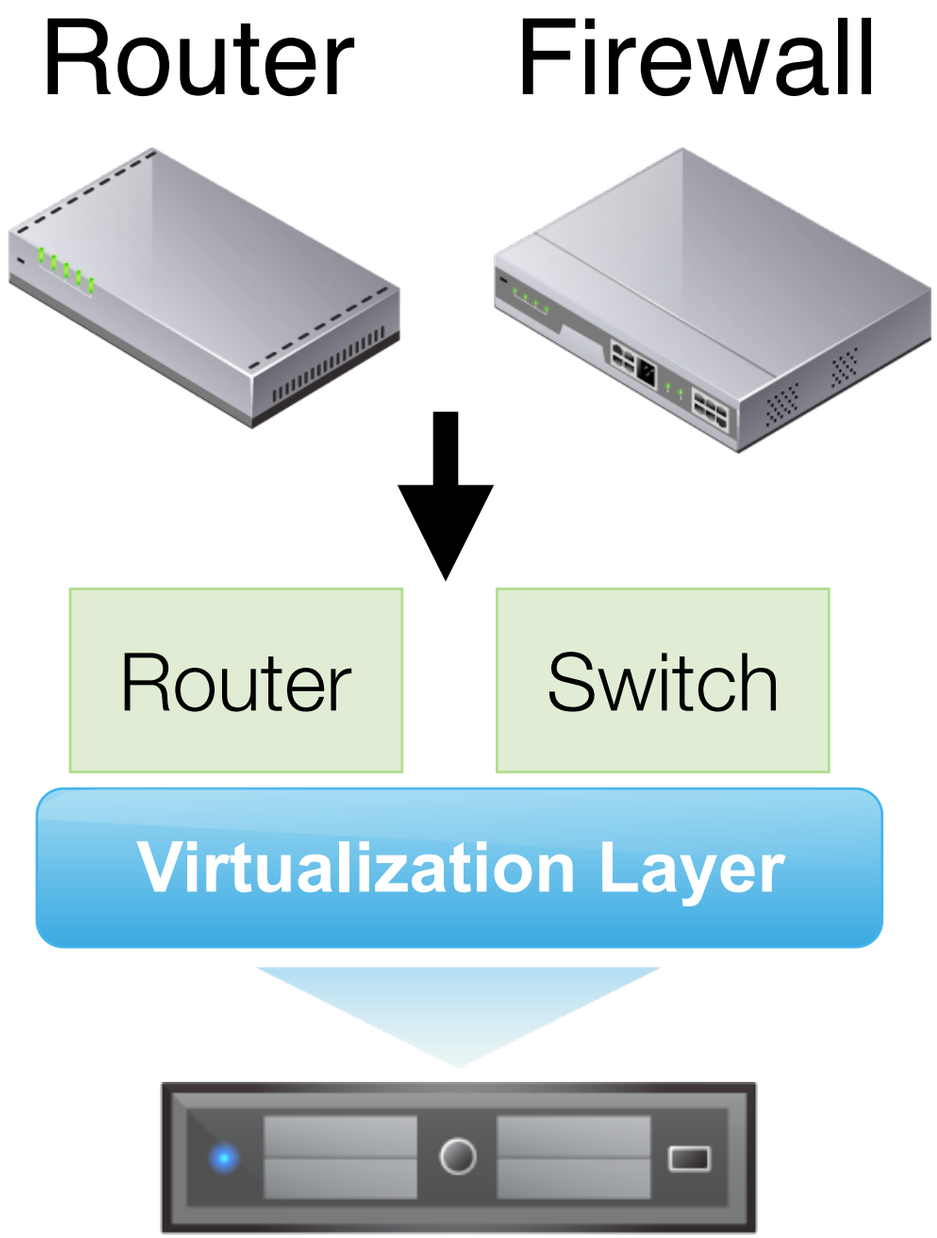
Network Function Virtualization

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 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

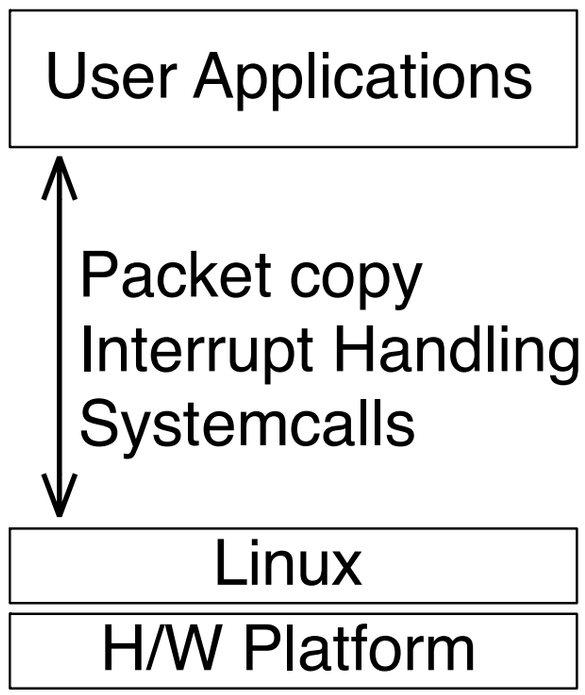
ClickOS [Martins, NSDI '14] and NetVM [Hwang, NSDI '14]

- VM based network services
- Flexible deployment and composition

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

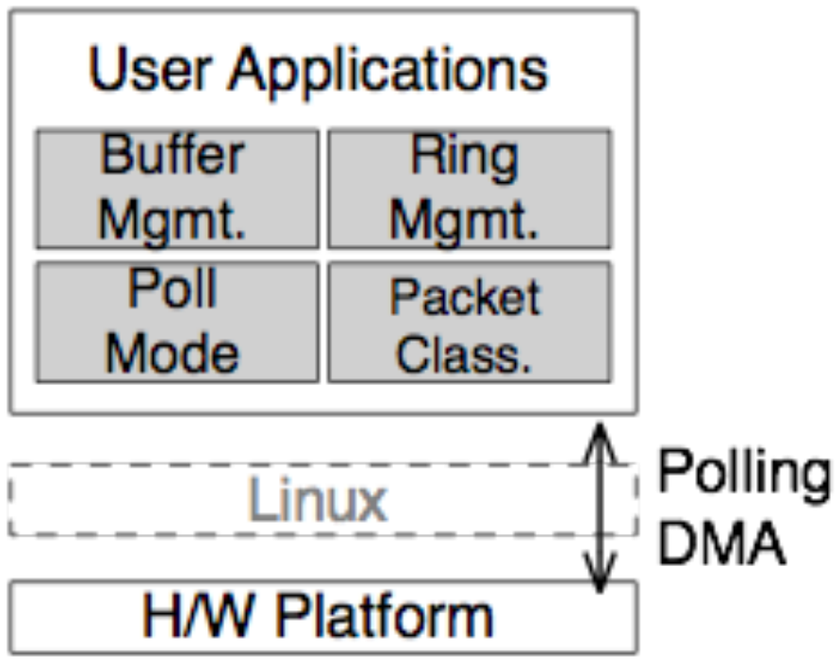


Can you handle being interrupted 60 million times per second?

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

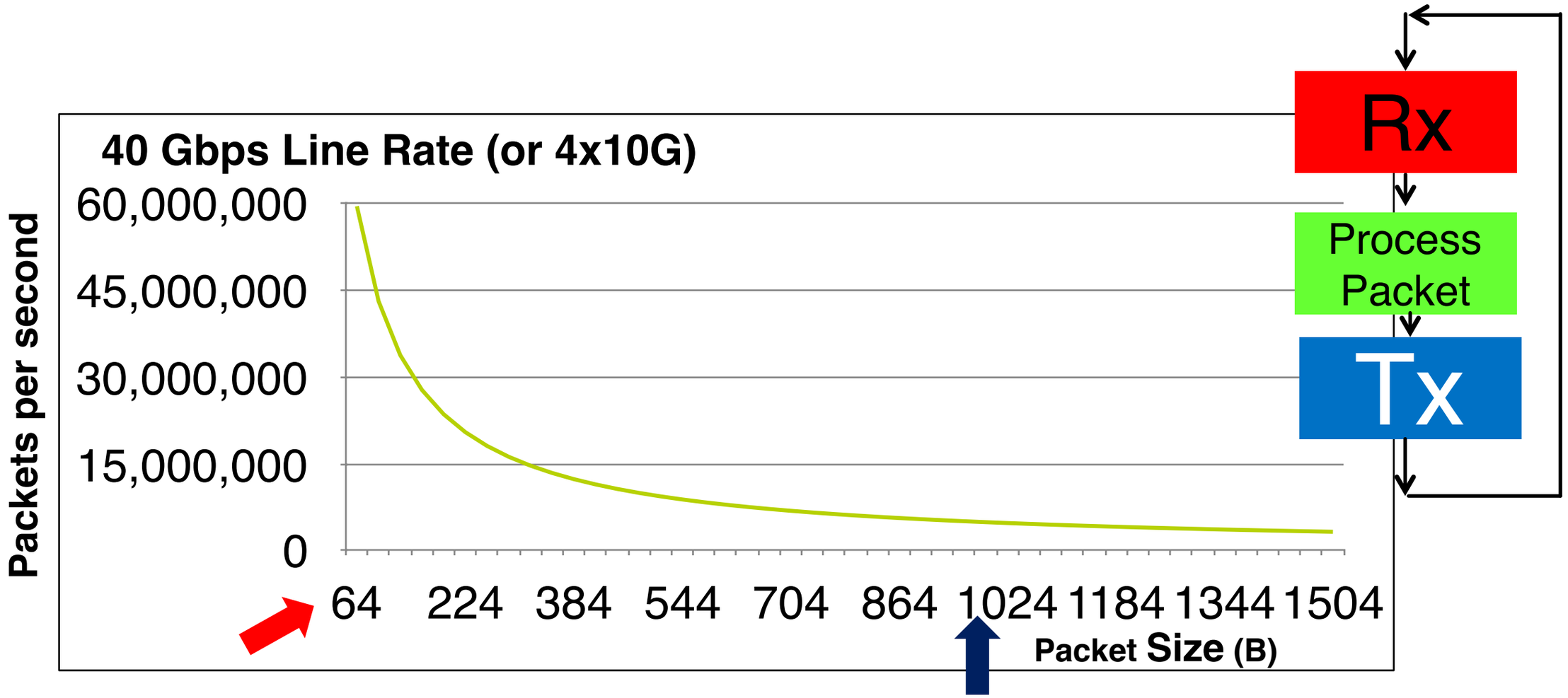
Why try it:

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

Alternatives:

- netmap
- PF_RING

What is “line rate”?



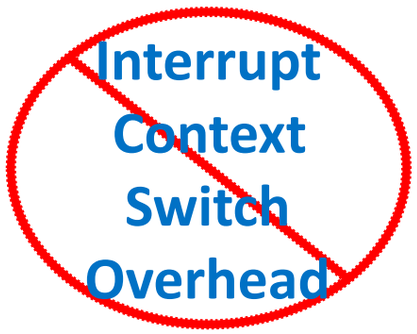
Network Infrastructure Packet Sizes

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

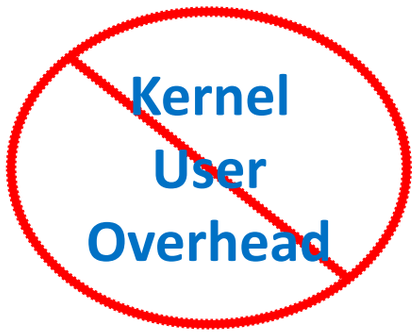
Packet Size	1024 bytes
40G Packets/second	4.8 Million each way
Packet arrival rate	208.8 ns
2 GHz Clock cycles	**417 cycles**

How to Eliminate / Hide Overheads?



Polling

Huge Pages



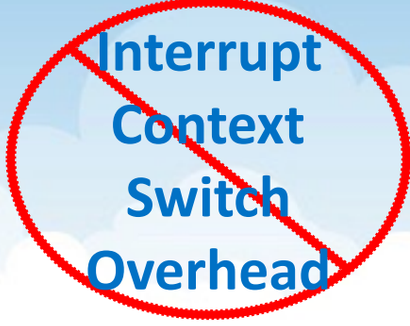
User Mode Driver

Lockless Inter-core Communication

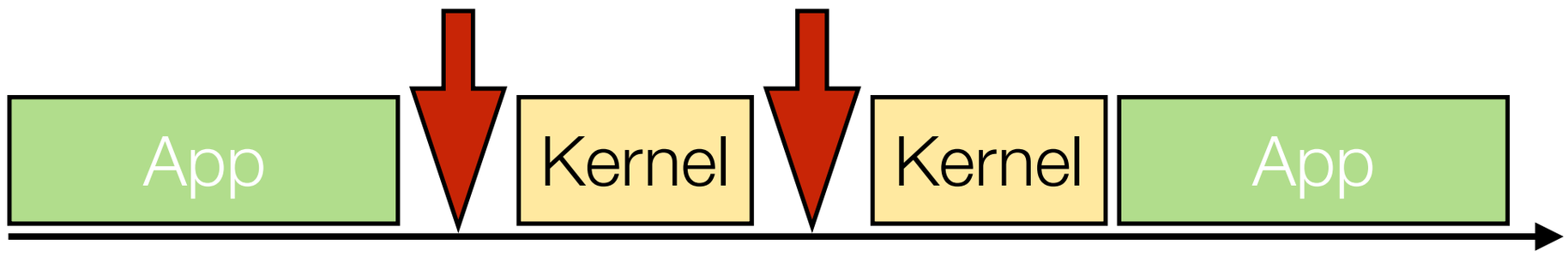
Pthread Affinity

High Throughput Bulk Mode I/O calls

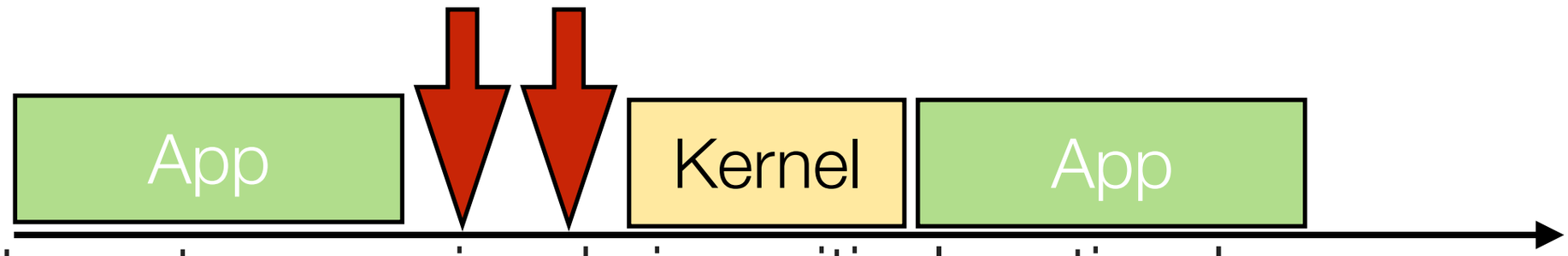
Network Interrupts



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

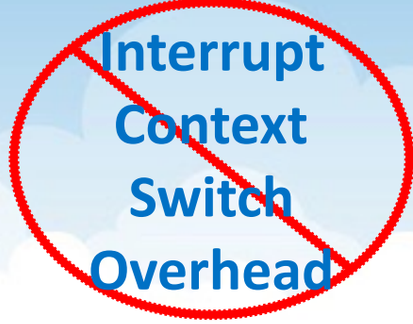


Coalescing interrupts helps, but still causes problems



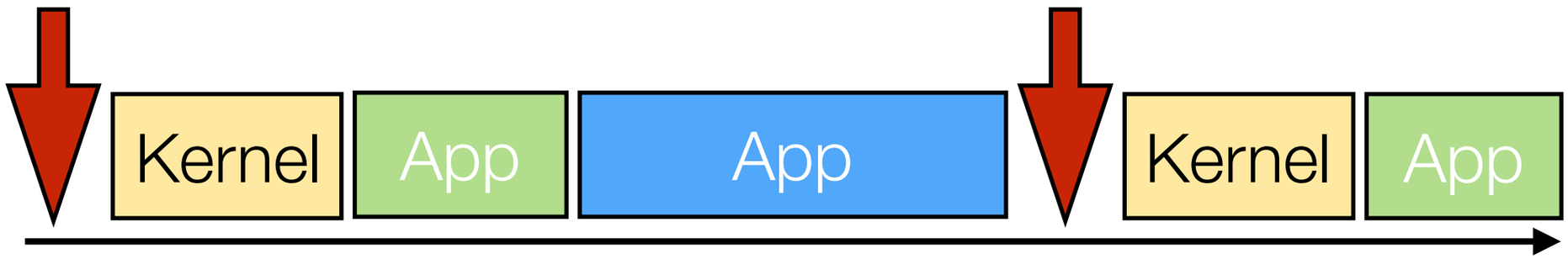
- Interrupts can arrive during critical sections!
- Interrupts can be delivered to the wrong CPU core!
- Still must pay context switch cost

Polling



Continuously loop looking for new packet arrivals

Trade-off?



Interrupts help share the CPU

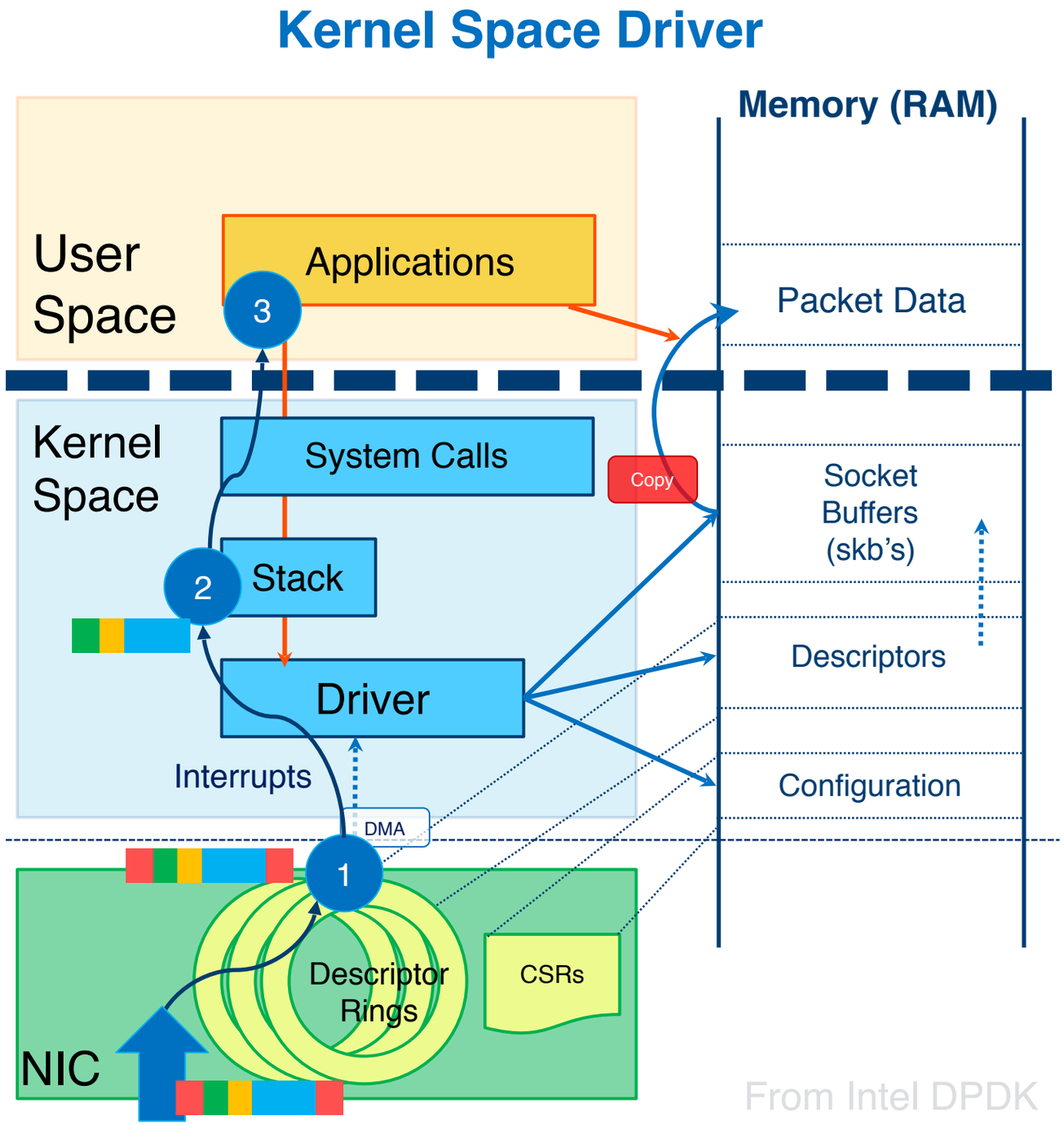
Polling can be wasteful

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?



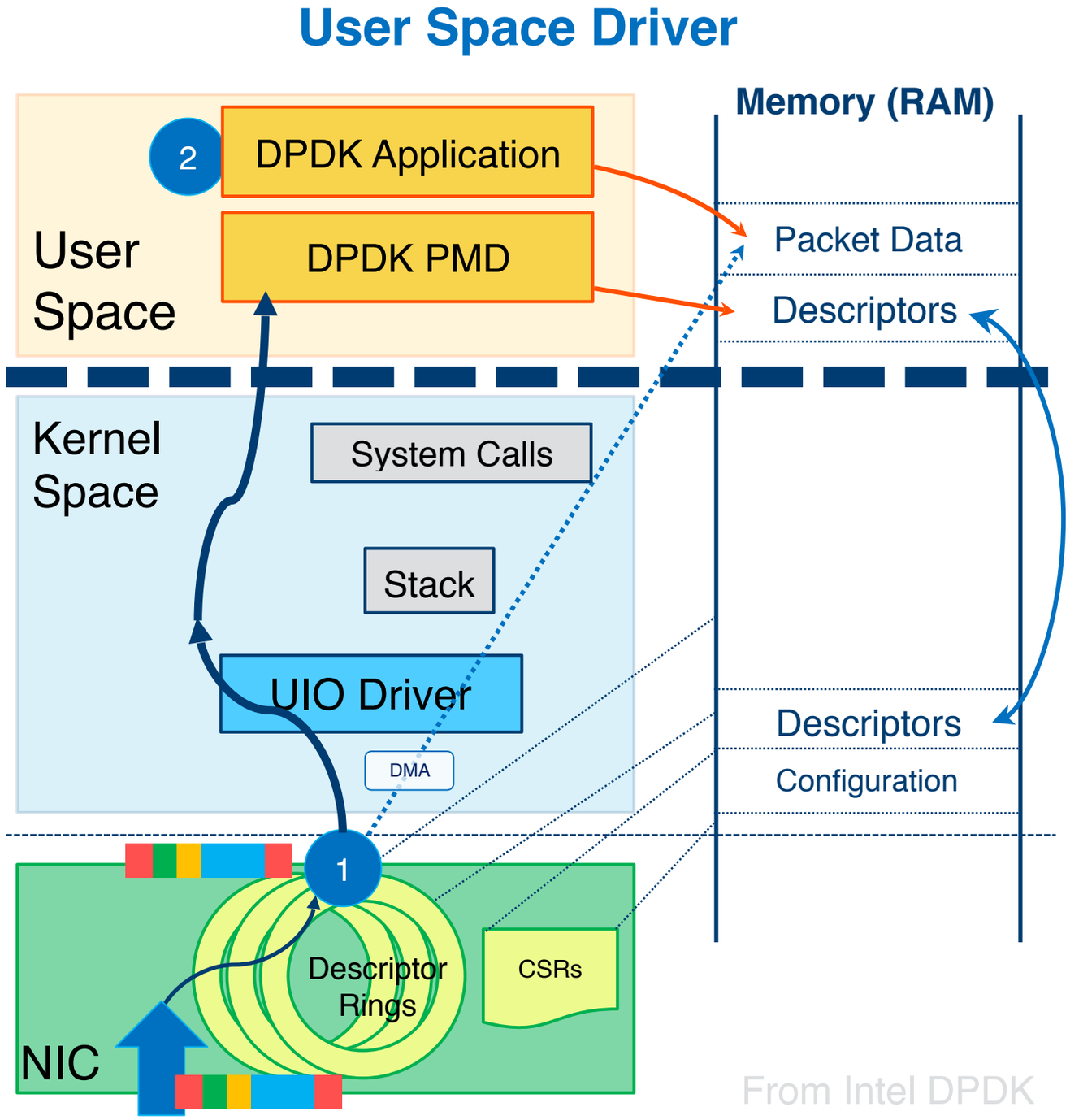
From Intel DPDK University Lecture

Kernel Bypass

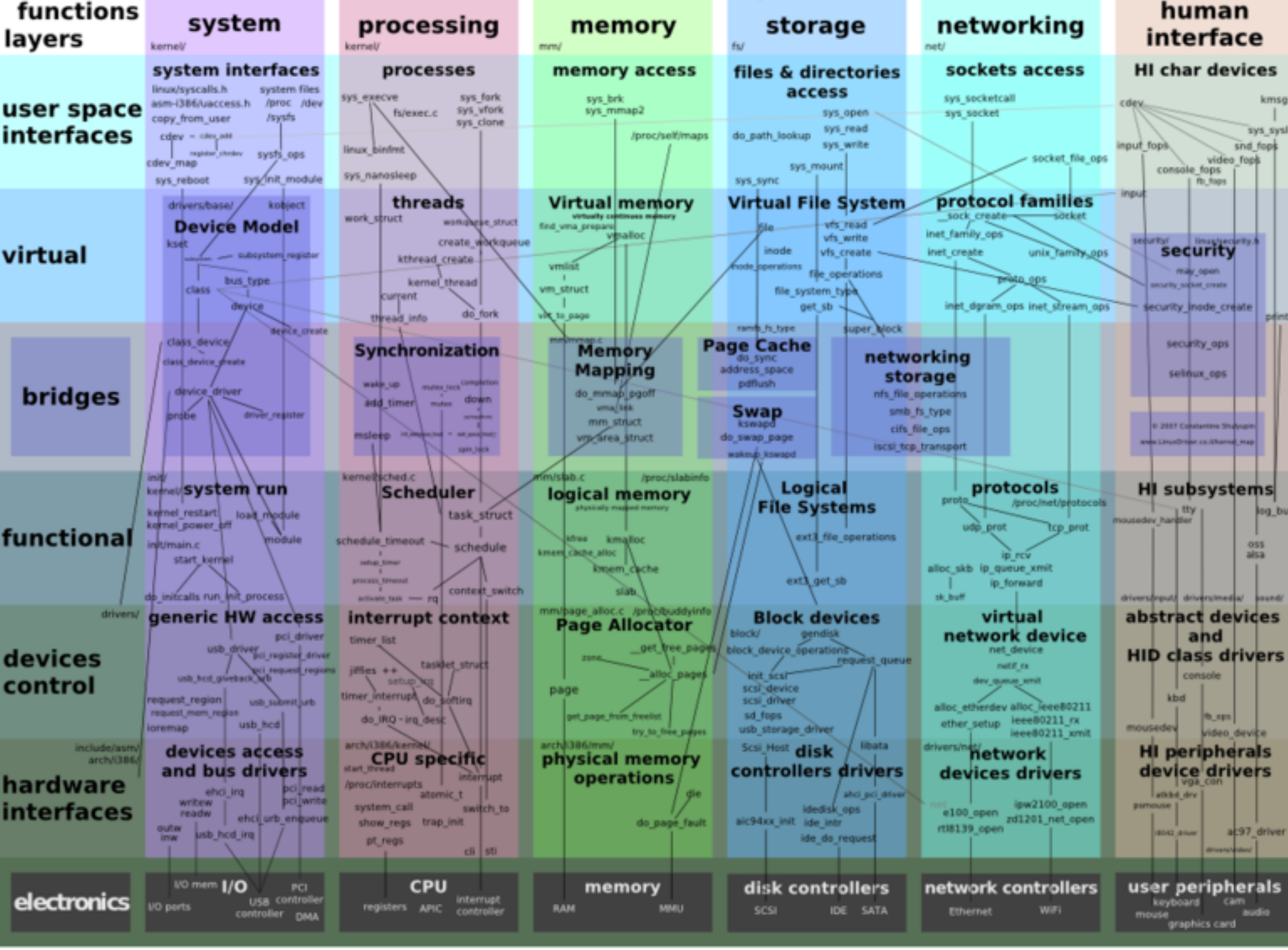
User-mode Driver

- 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
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
super_block
ramfs_fs_type
protocol families
__sock_create
socket
inet_family_ops
inet_create
unix_family_ops
proto_ops
inet_dgram_ops
inet_stream_ops
security/
linux/security.h
security
may_open
security_socket_create
security_inode_create
security_ops
selinux_ops
bridges
class_device
class_device_create
device_driver
probe
driver_register
Synchronization
wake_up
add_timer
msleep
mutex_lock
mutex
completion
down
spin_lock
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
functional
init/
kernel/
system run
kernel_restart
kernel_power_off
load_module
module
init/main.c
start_kernel
do_initcalls
run_init_process
kernel/sched.c
Scheduler
task_struct
schedule_timeout
schedule
setup_timer
process_timeout
activate_task
rq
context_switch
mm/slab.c
/proc/slabinfo
logical memory
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
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
do_IRQ
irq_desc
tasklet_struct
mm/page_alloc.c
/proc/buddyinfo
Page Allocator
__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
mousedev
fb_sps
video_device
hardware interfaces
include/asm/
arch/i386/
devices access and bus drivers
ehci_irq
writew
readw
outw
inw
usb_hcd_irq
pci_read
pci_write
ehci_urb_enqueue
arch/i386/kernel/
CPU specific
start_thread
/proc/interrupts
atomic_t
system_call
show_regs
trap_init
pt_regs
interrupt
switch_to
cli
sti
arch/i386/mm/
physical memory operations
die
do_page_fault
disk controllers drivers
Scsi_Host
libata
aic94xx_init
idedisk_ops
ide_intr
ide_do_request
ahci_pci_driver
network devices drivers
drivers/net/
e100_open
rtl8139_open
ipw2100_open
zd1201_net_open
HI peripherals device drivers
vga_con
atkbd_drv
psmouse
ac97_driver
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

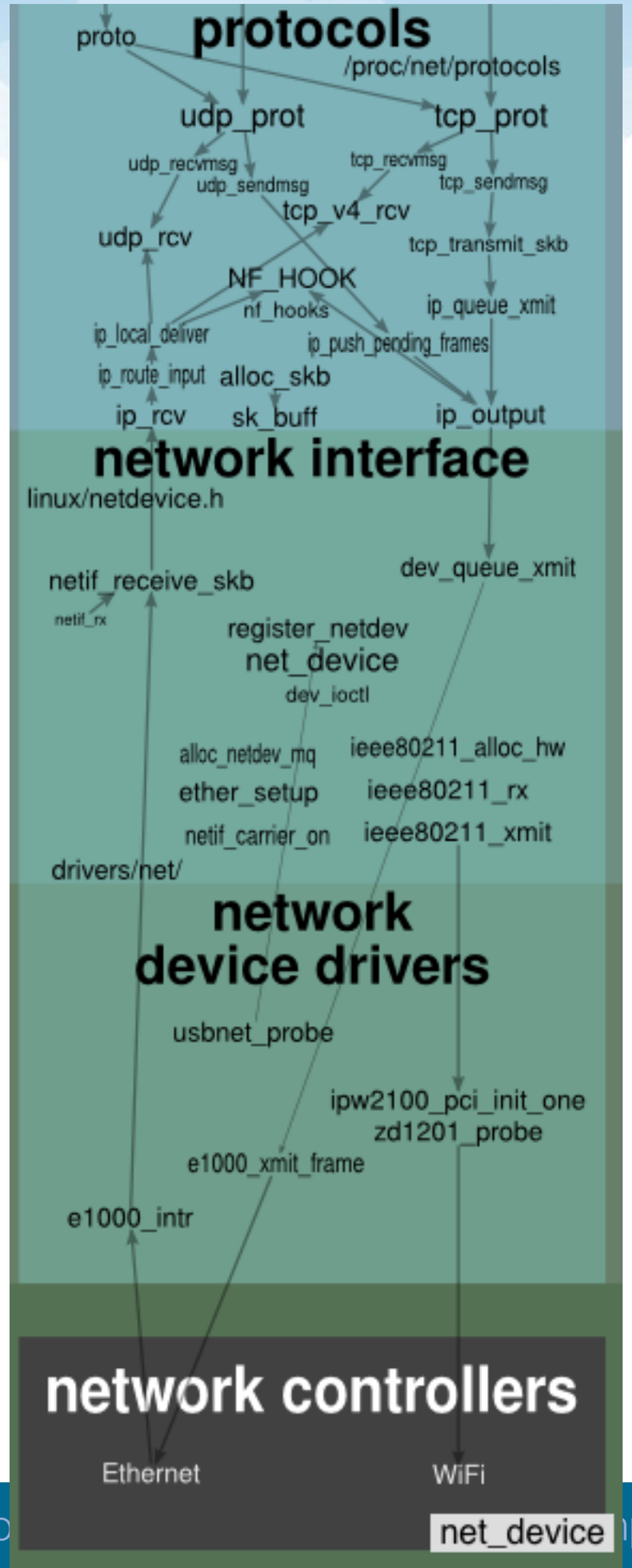
Linux networking stack has a lot of extra components

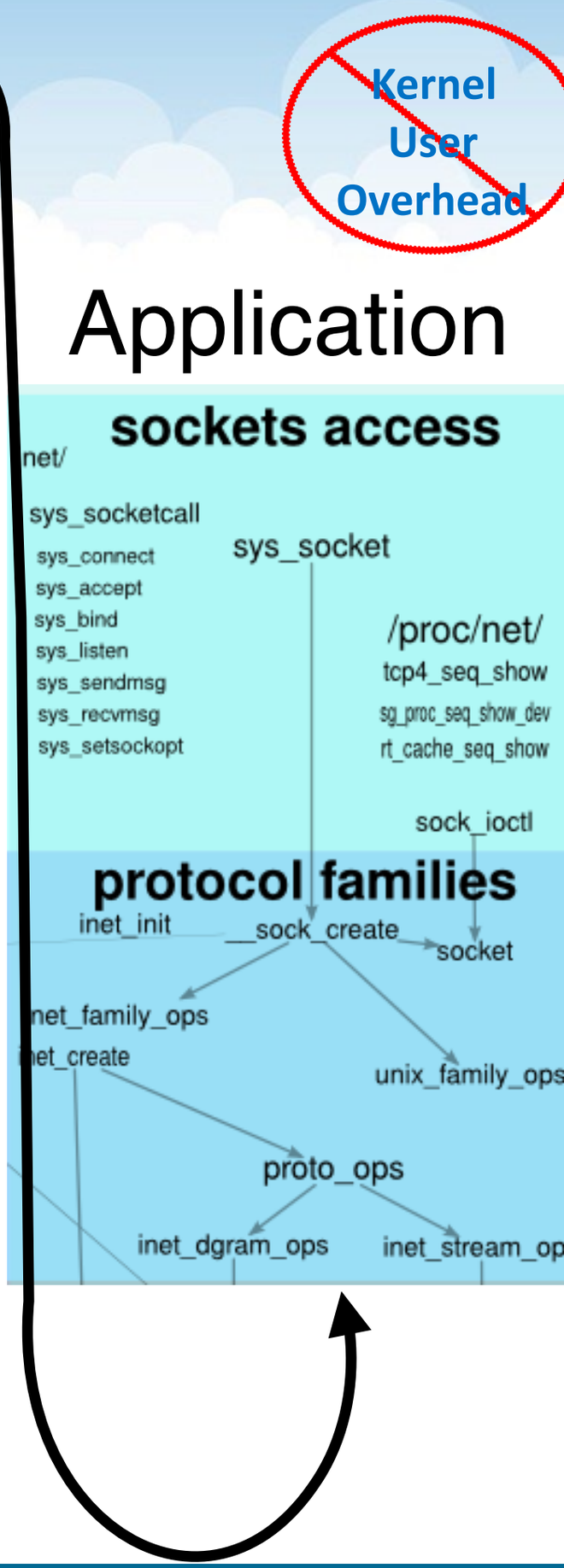
For NFV middlebox we don't use all of this:

- TCP, UDP, sockets

NFV middle boxes just need packet data

- Need it fast!

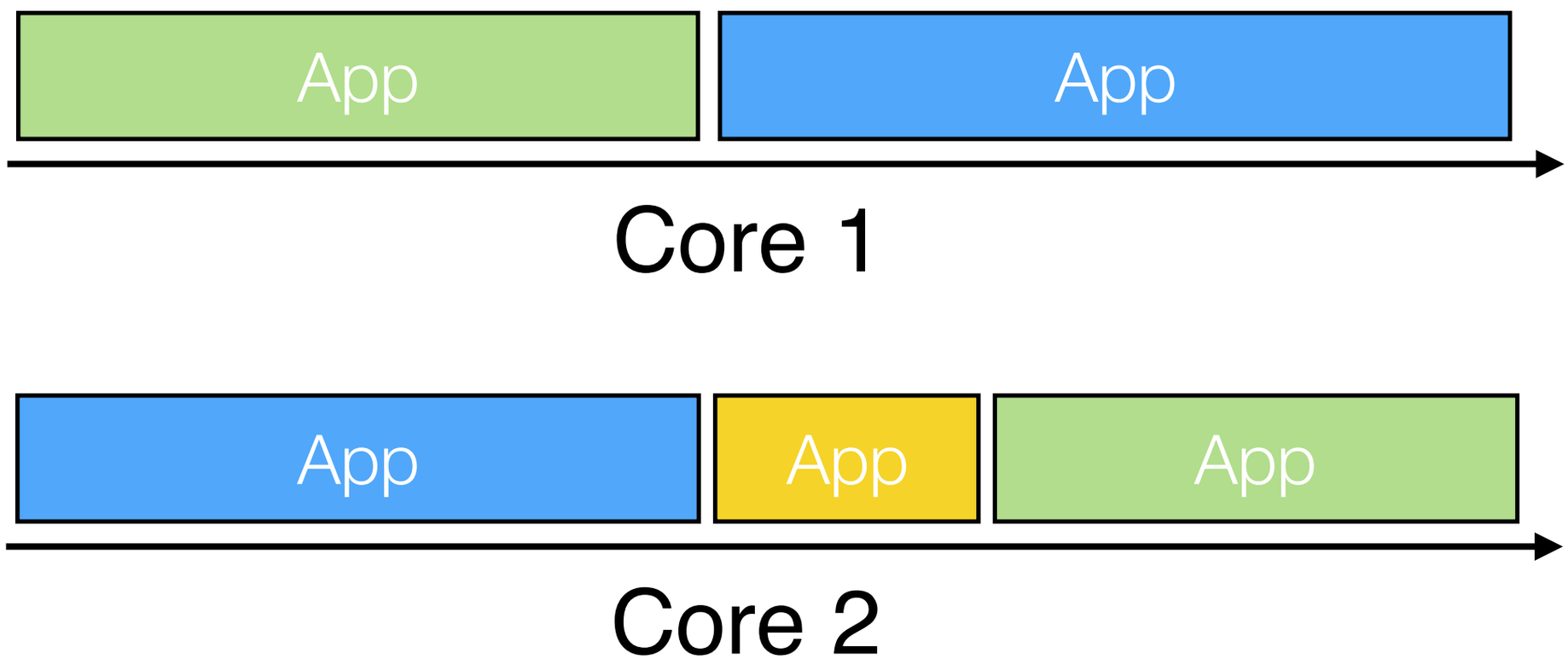




CPU Core Affinity

Linux Scheduler can move threads between cores

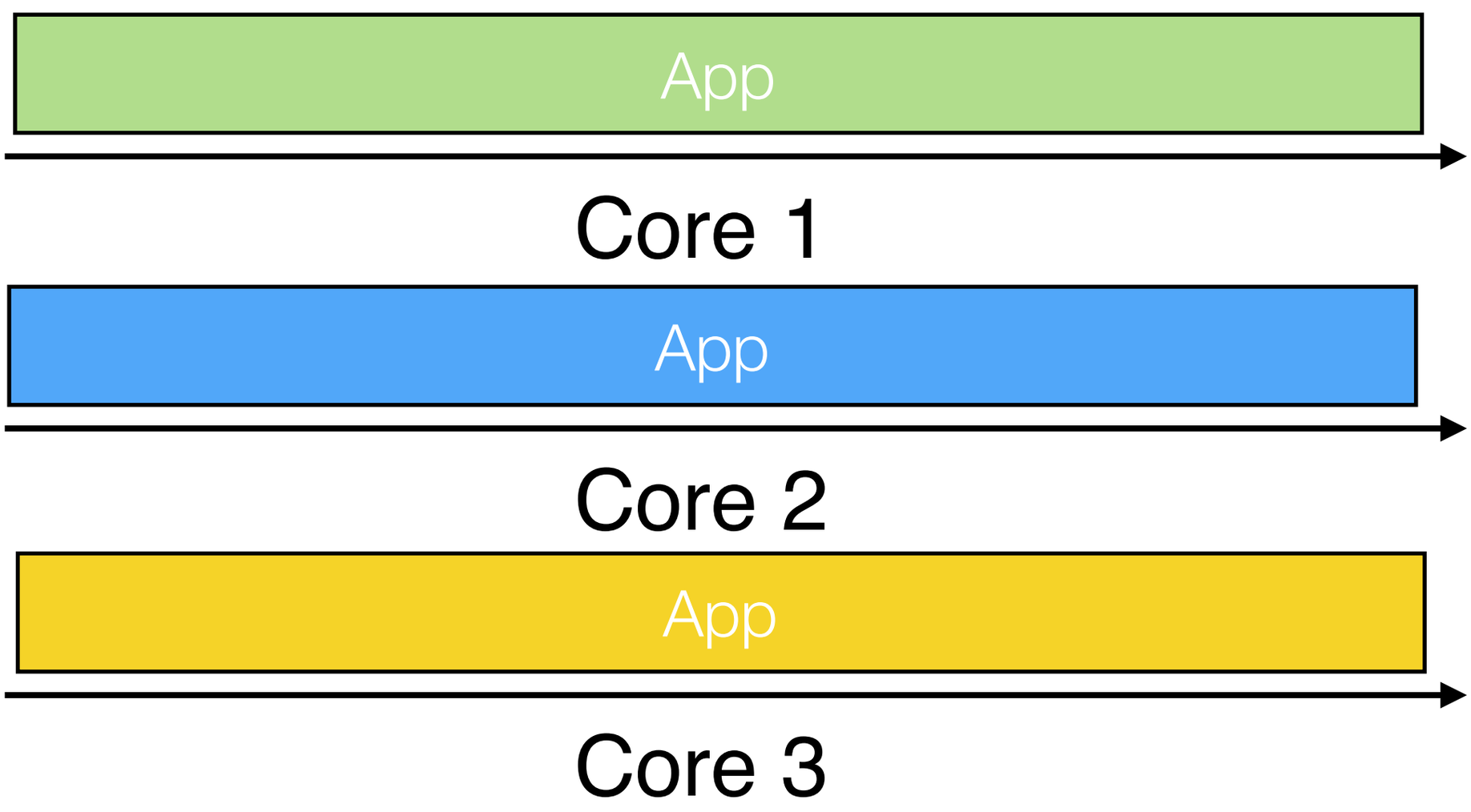
- Context switches :(
- Cache locality :(



CPU Core Affinity

Pin threads and dedicate cores

- **Trade-offs?**



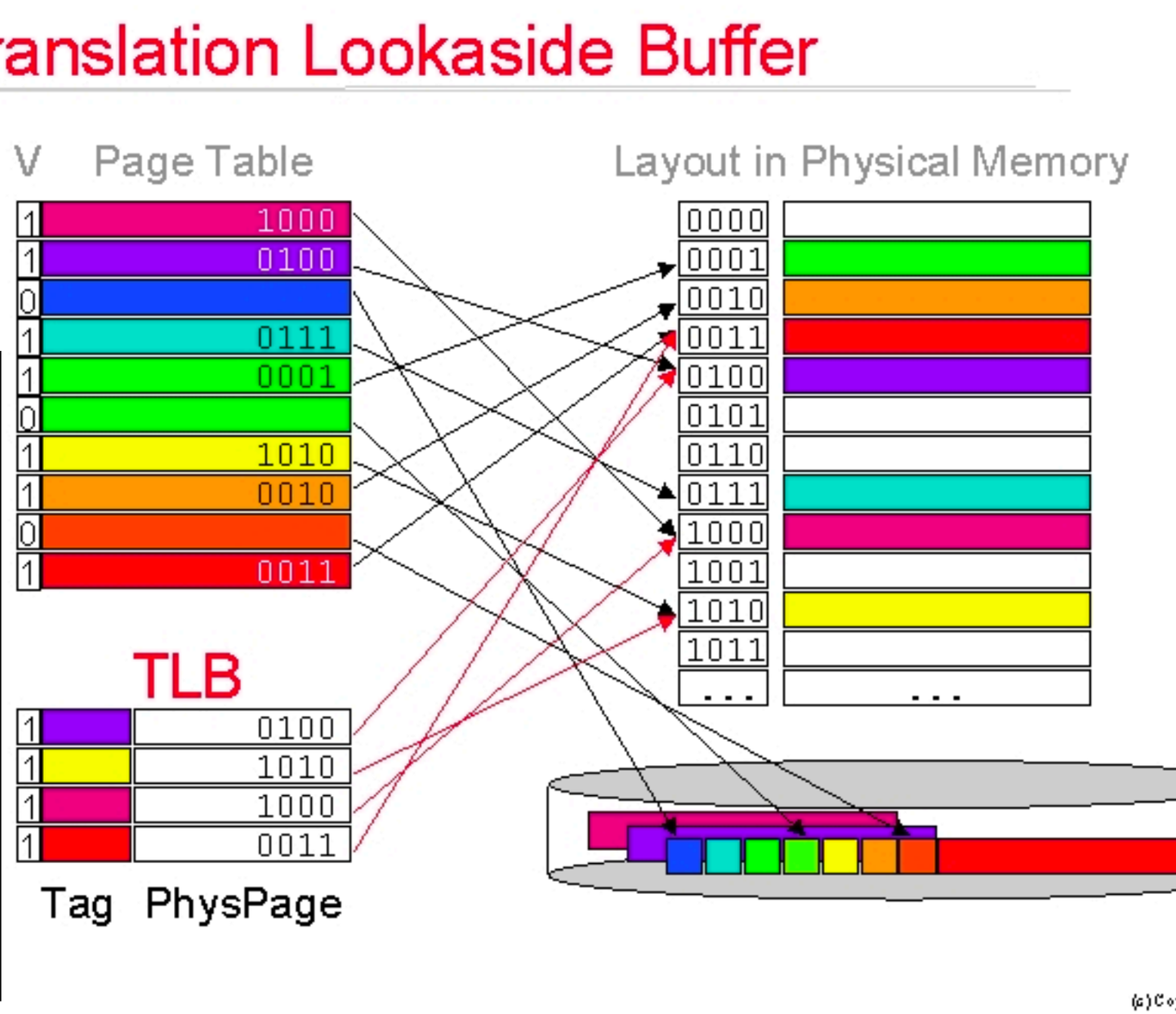
Paging Overhead

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

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?

Limitations

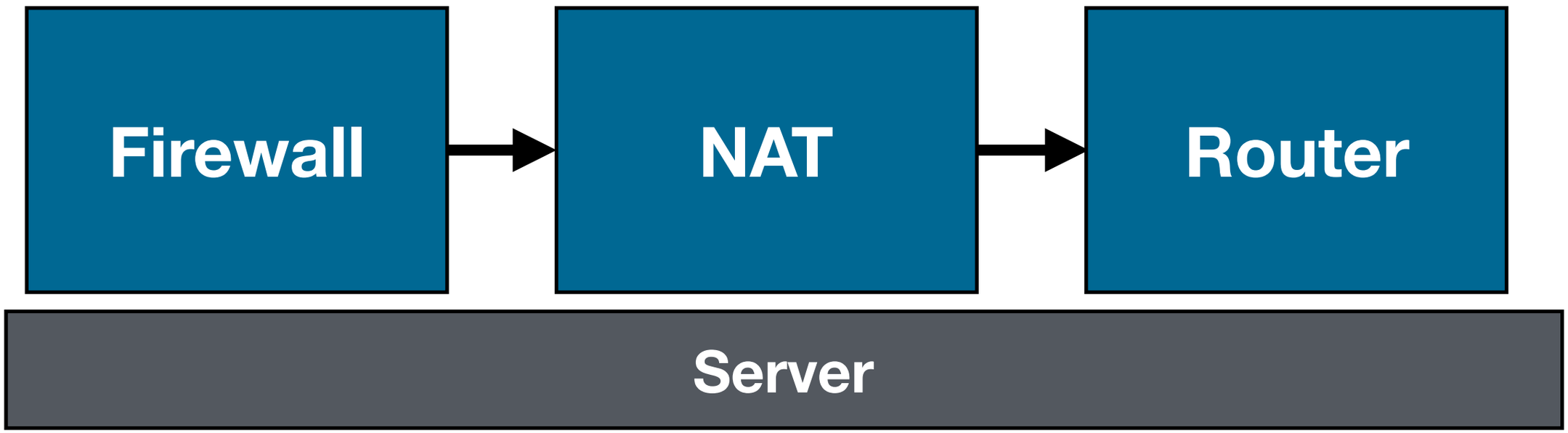
DPDK provides efficient I/O… but that's about it

Doesn't help with NF management or orchestration

Service Chains

Chain together functionality to build more complex services

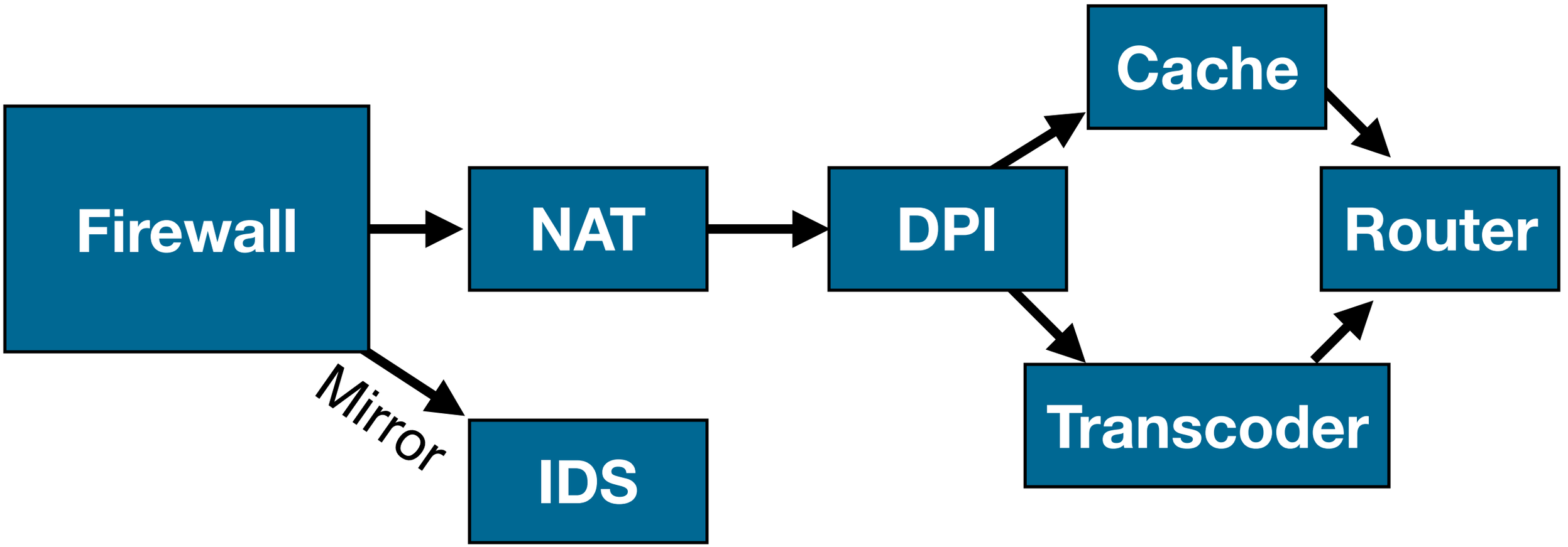
- Need to move packets through chain efficiently



Service Chains

Chain together functionality to build more complex services

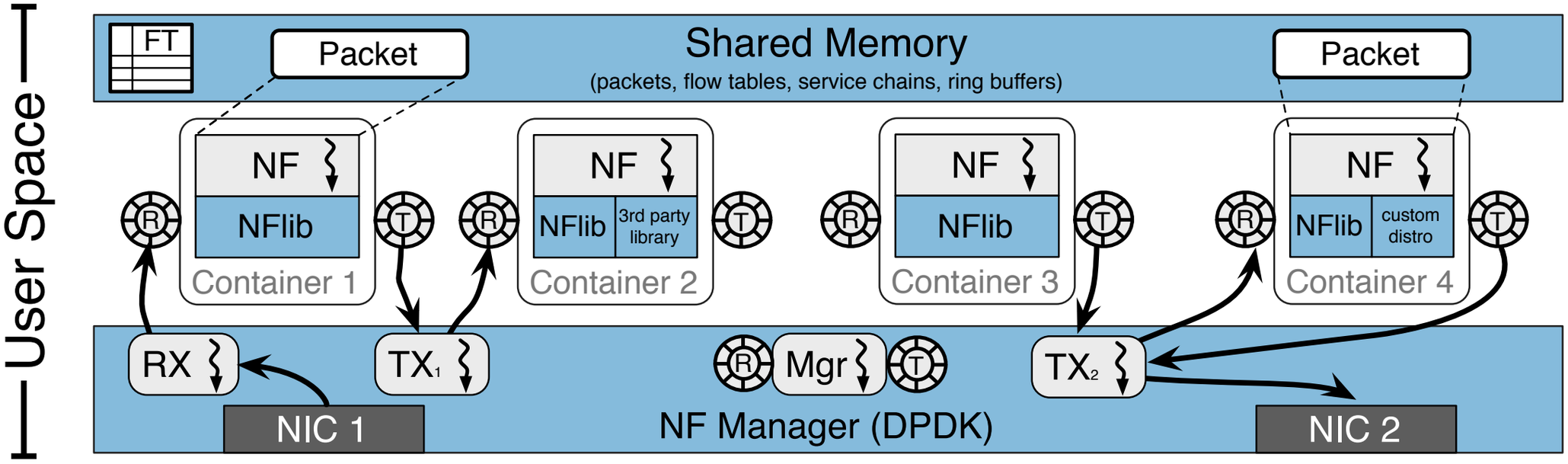
- Need to move packets through chain efficiently



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

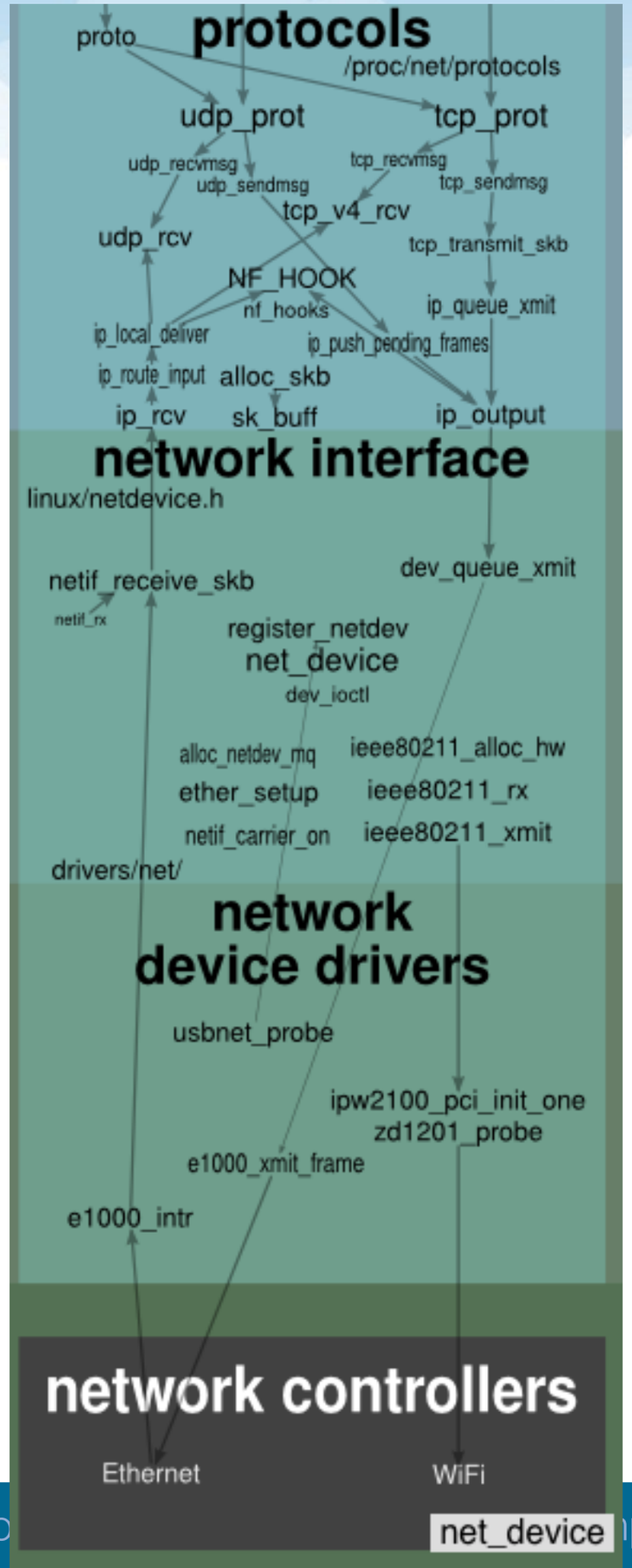
http://sdnfv.github.io/

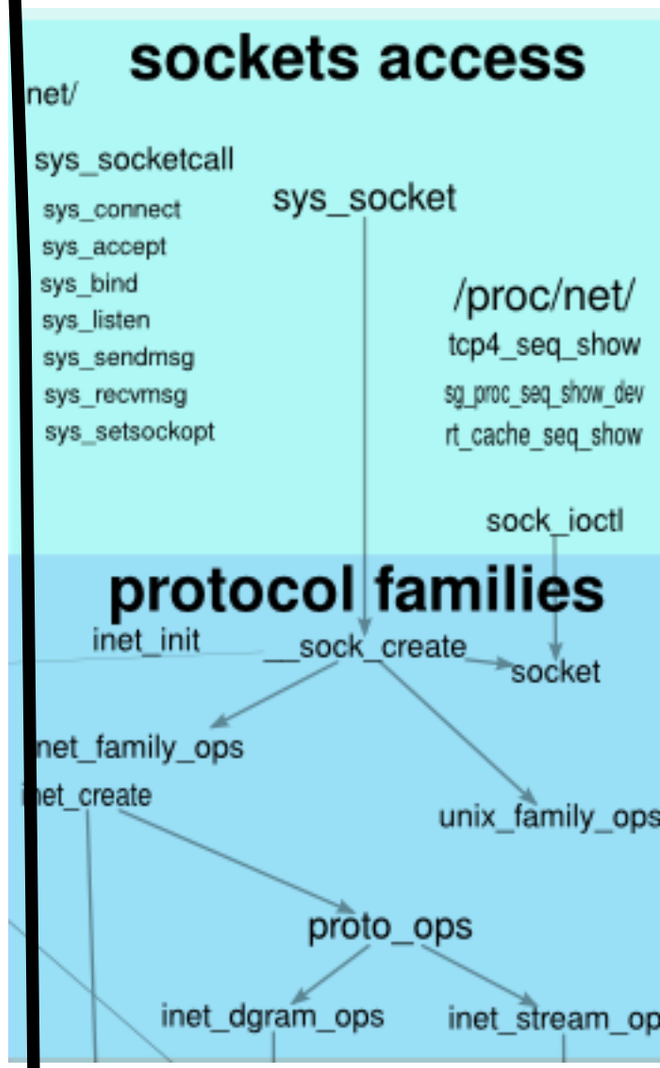
Limitations

DPDK only helps with raw packet IO

Doesn't provide any protocol stacks!

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

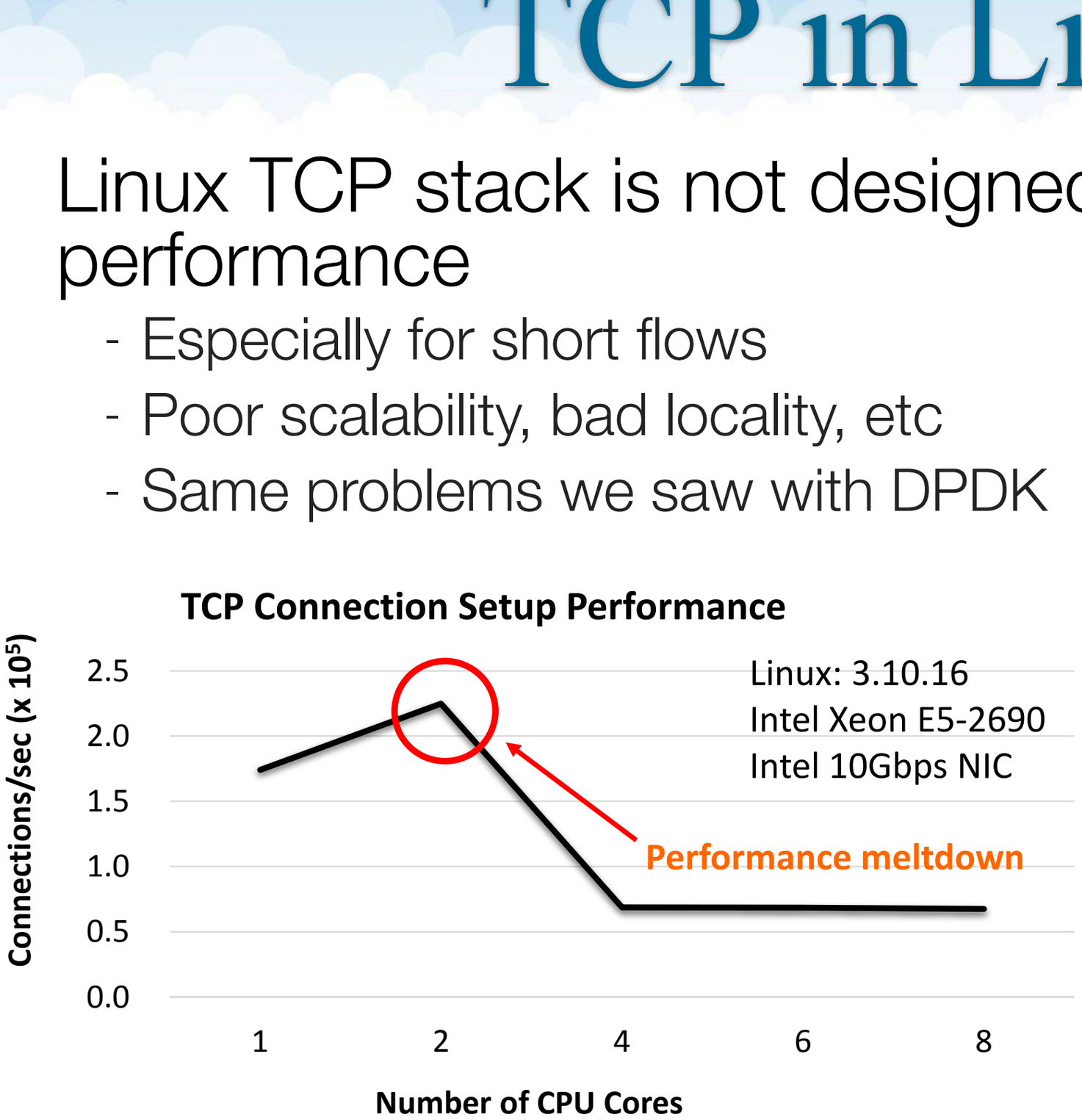


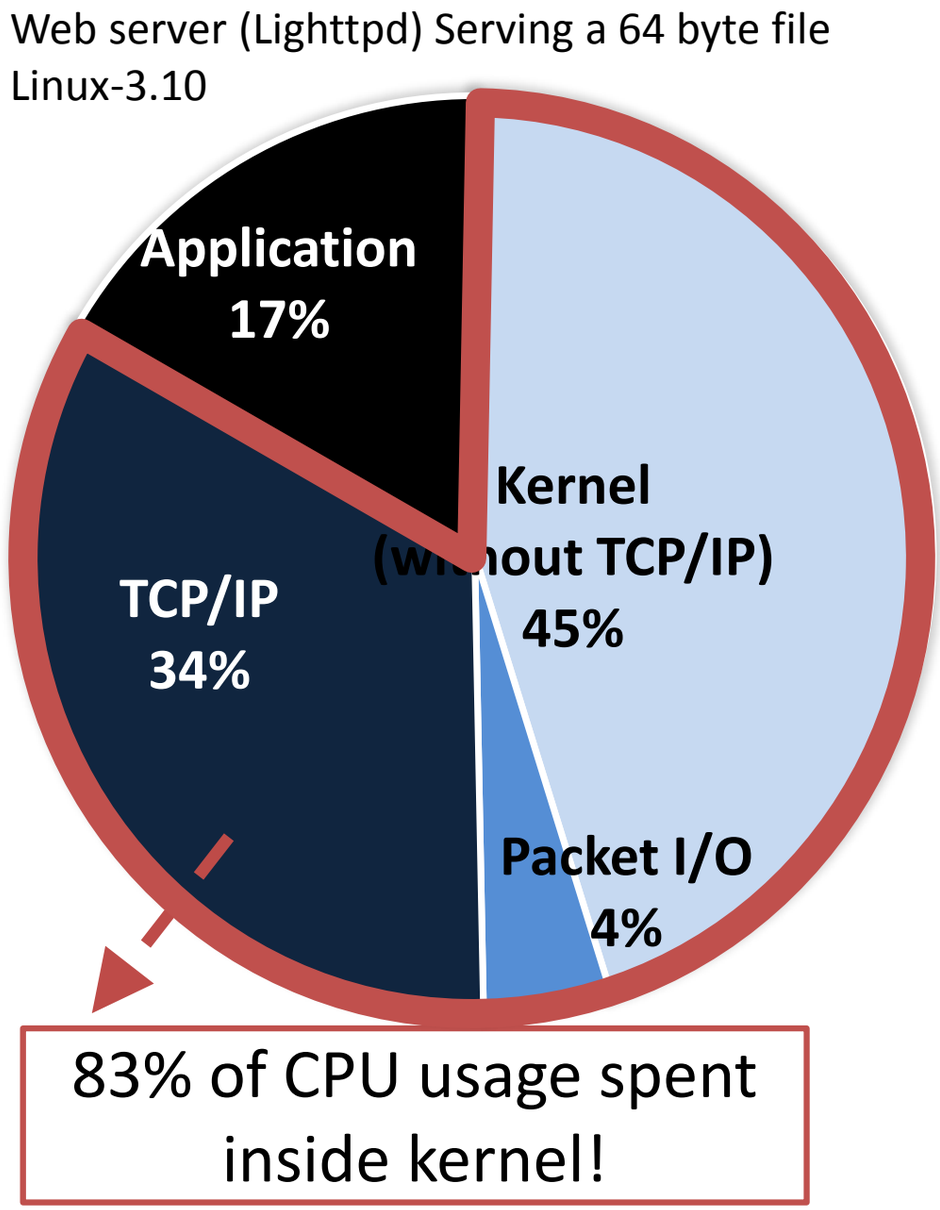


TCP in Linux

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

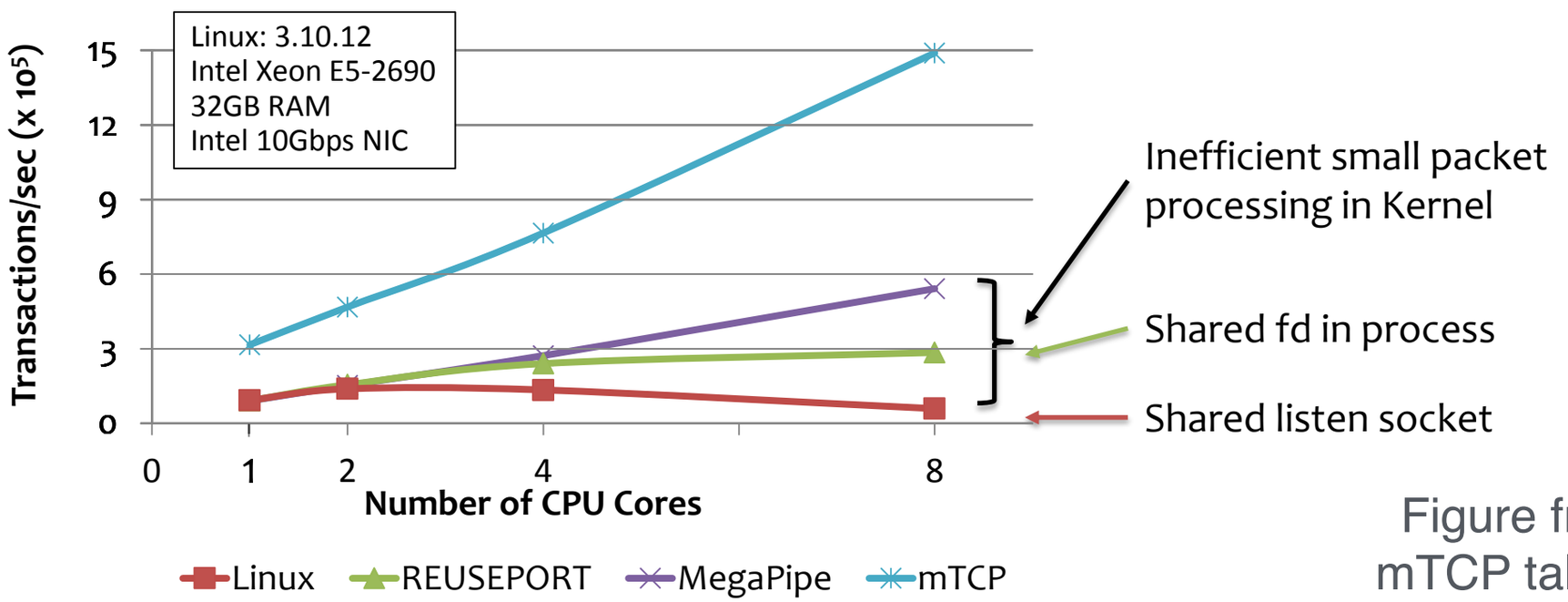


Figure from Jeong's mTCP talk at NSDI 14

mTCP Kernel Bypass

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

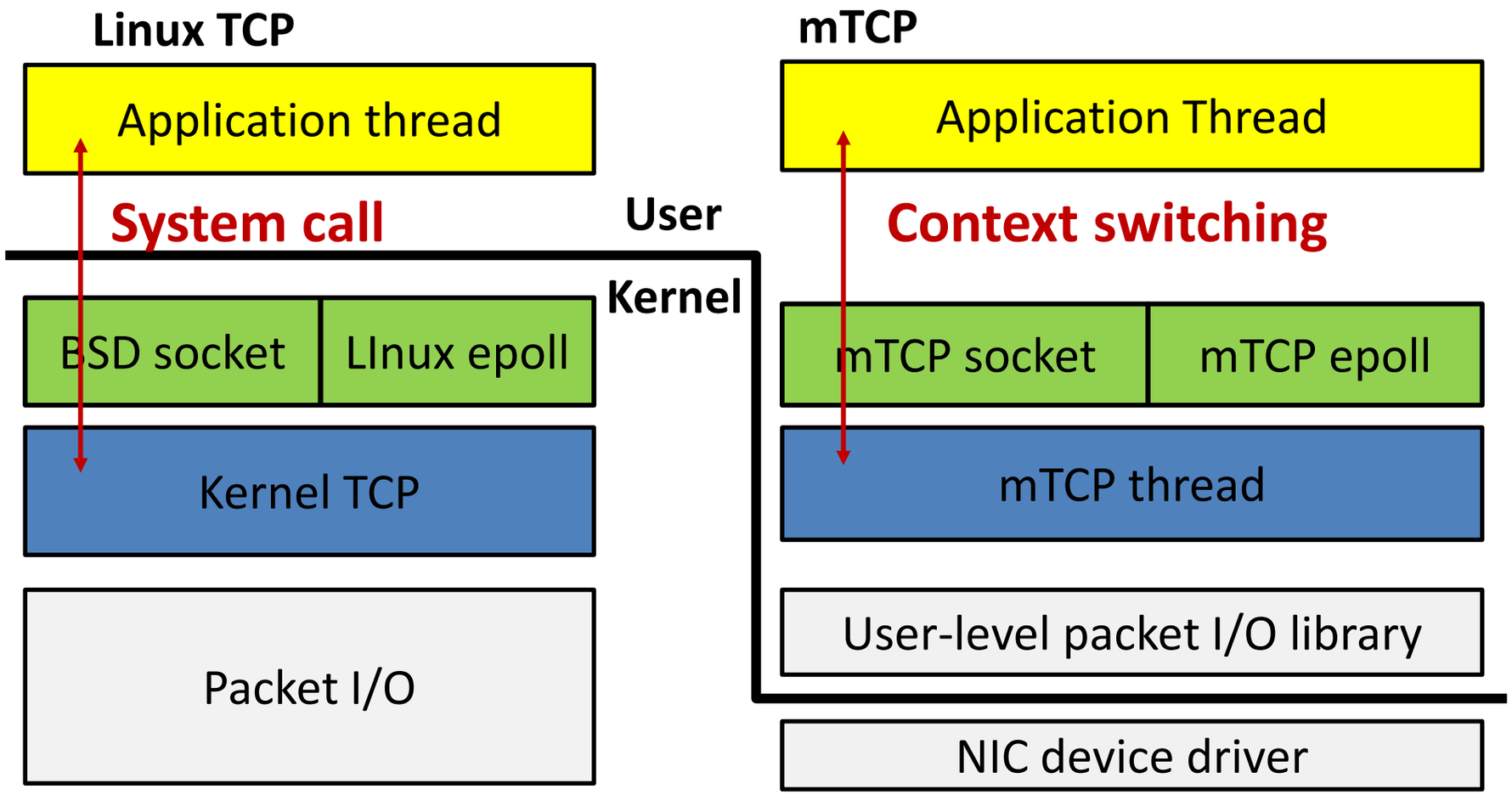
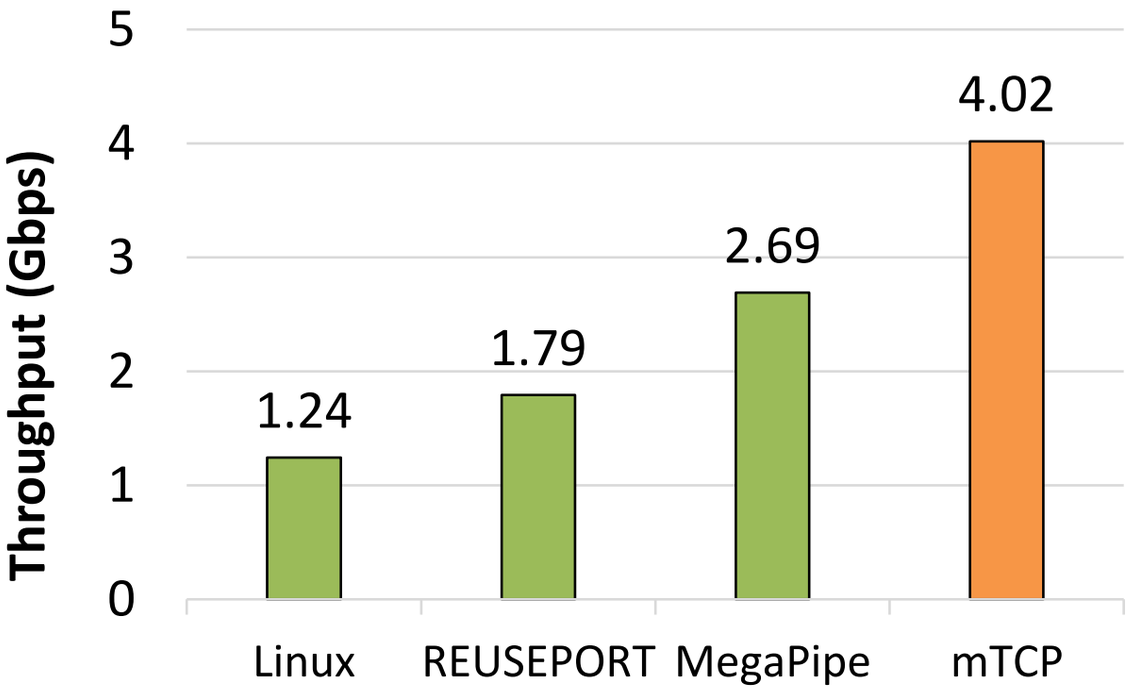


Figure from Jeong's mTCP talk at NSDI 14

Performance Improvement on Ported Applications

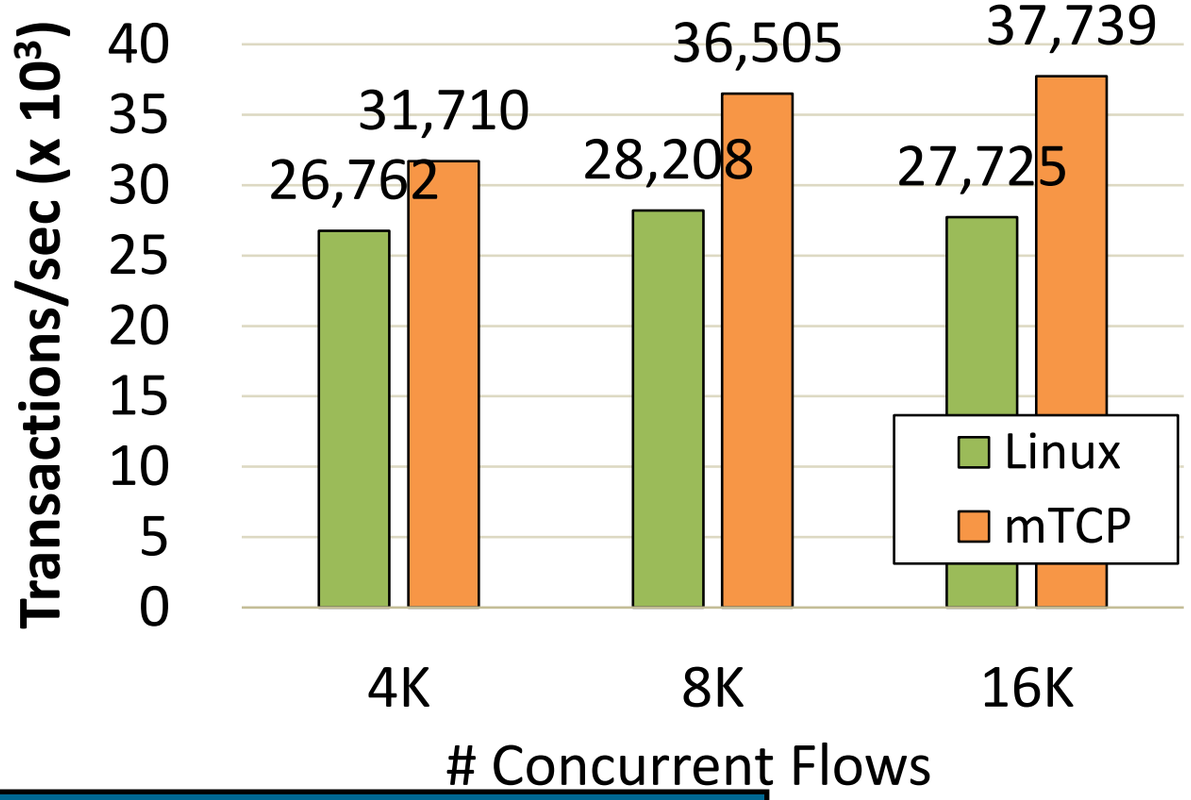
Web Server (Lighttpd)

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



SSL Proxy (SSLShader)

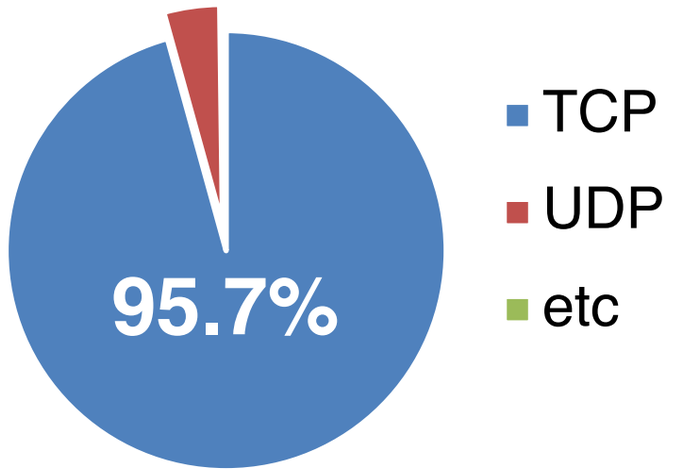
- Performance Bottleneck in TCP
- Cipher suite
  1024-bit RSA, 128-bit AES, HMAC-SHA1
- Download 1-byte object via HTTPS



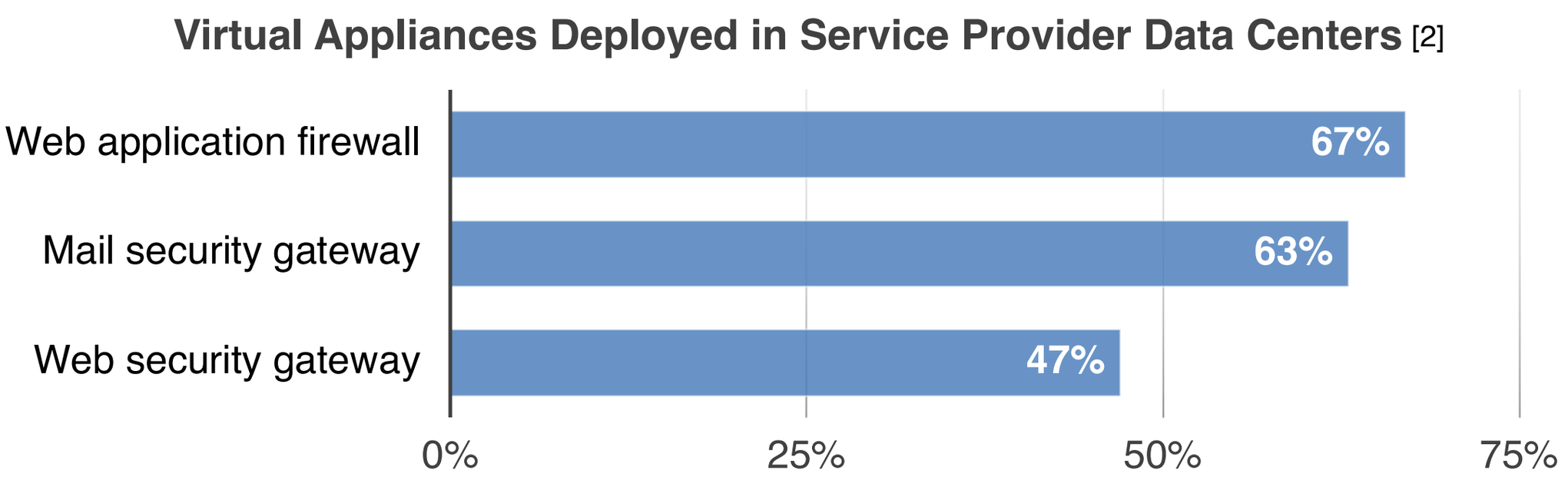
Slide from Jeong's mTCP talk, NSDI '14

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



[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.

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

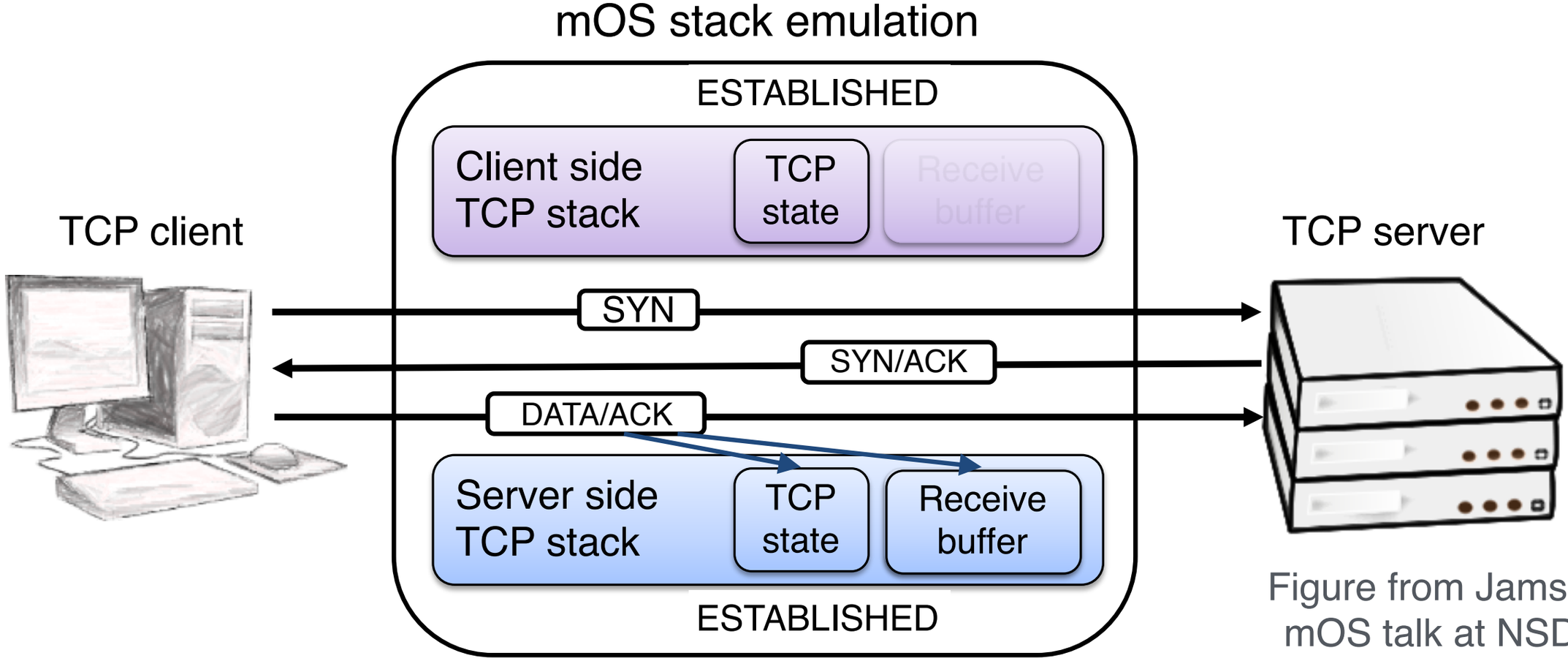


Figure from Jamshed's mOS talk at NSDI 17

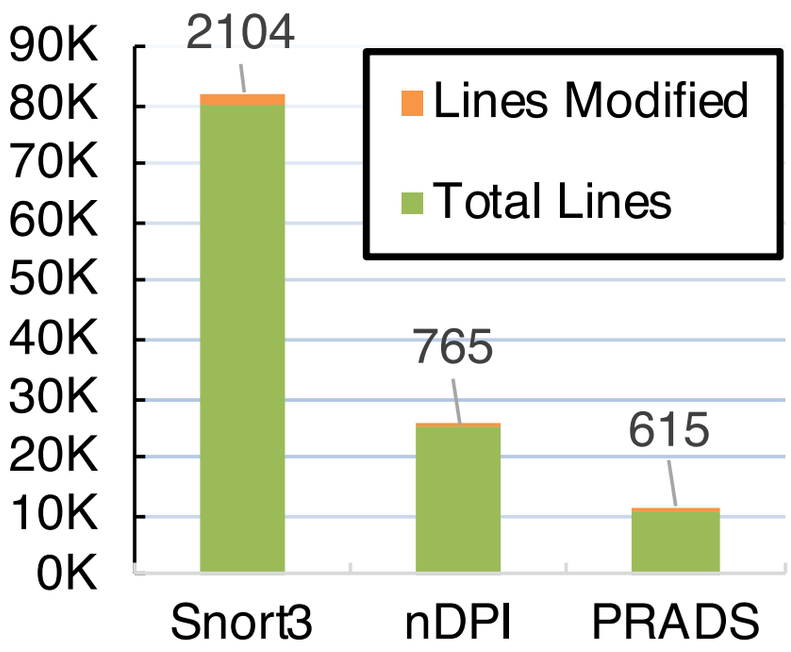
mOS [Jamshed, NSDI '17]

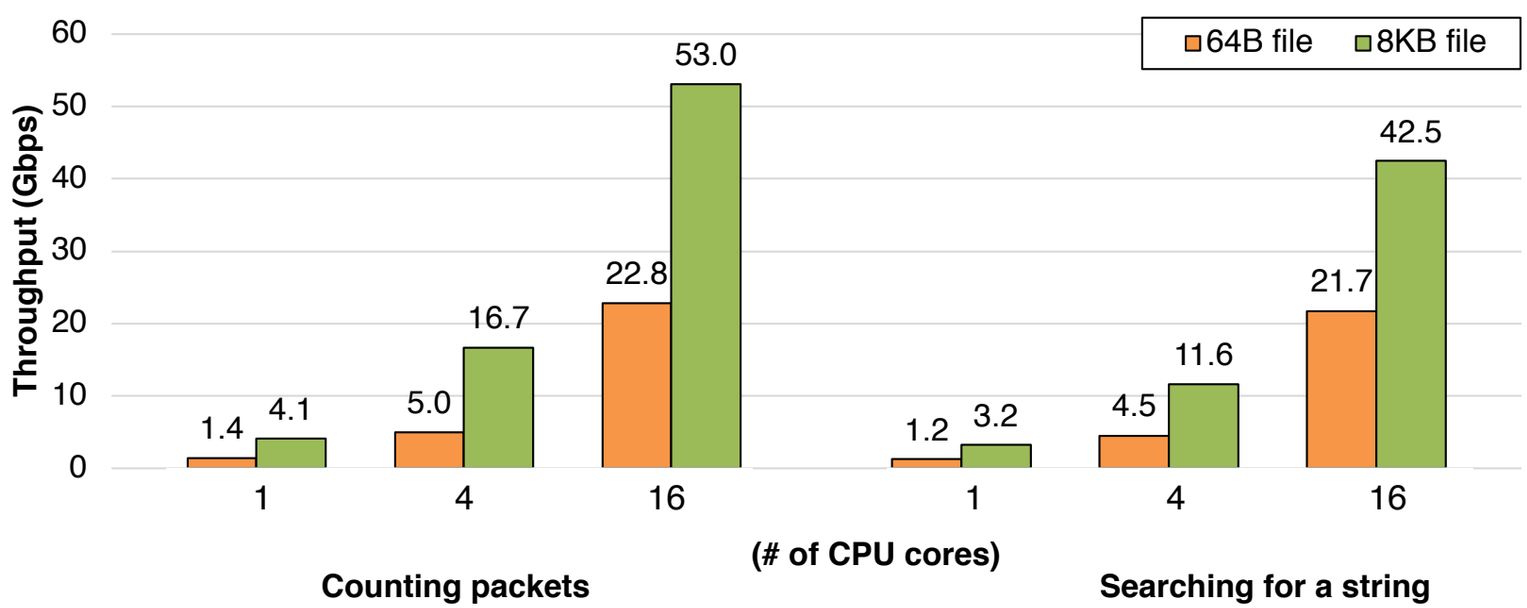
Base Events

- 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

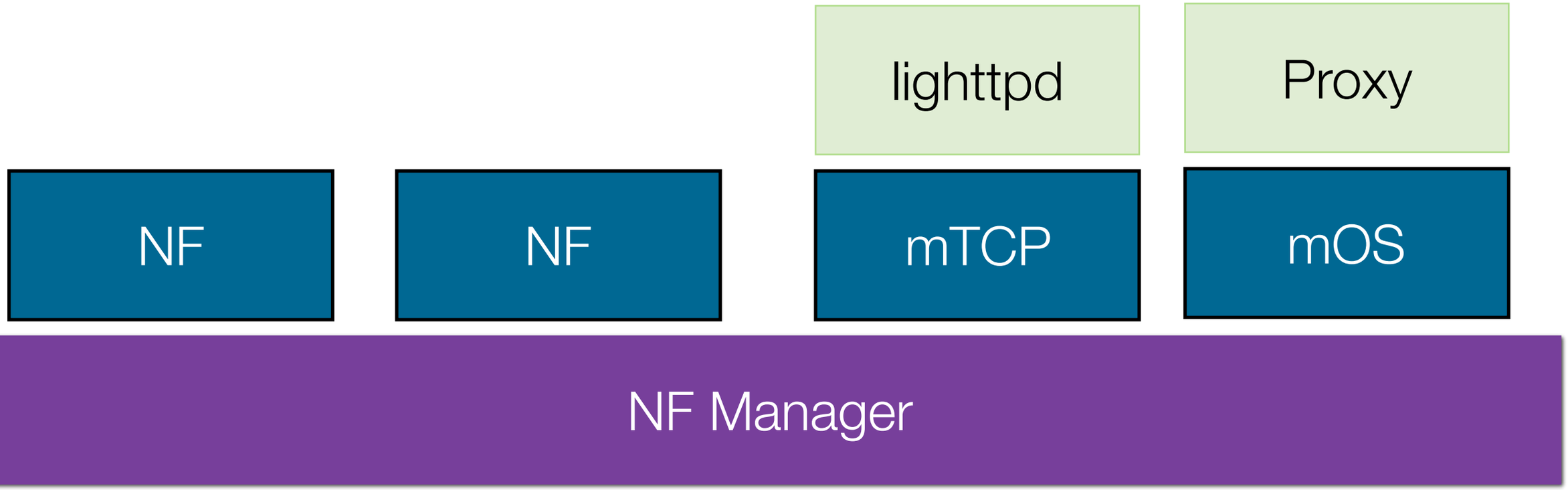
TCP + OpenNetVM

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

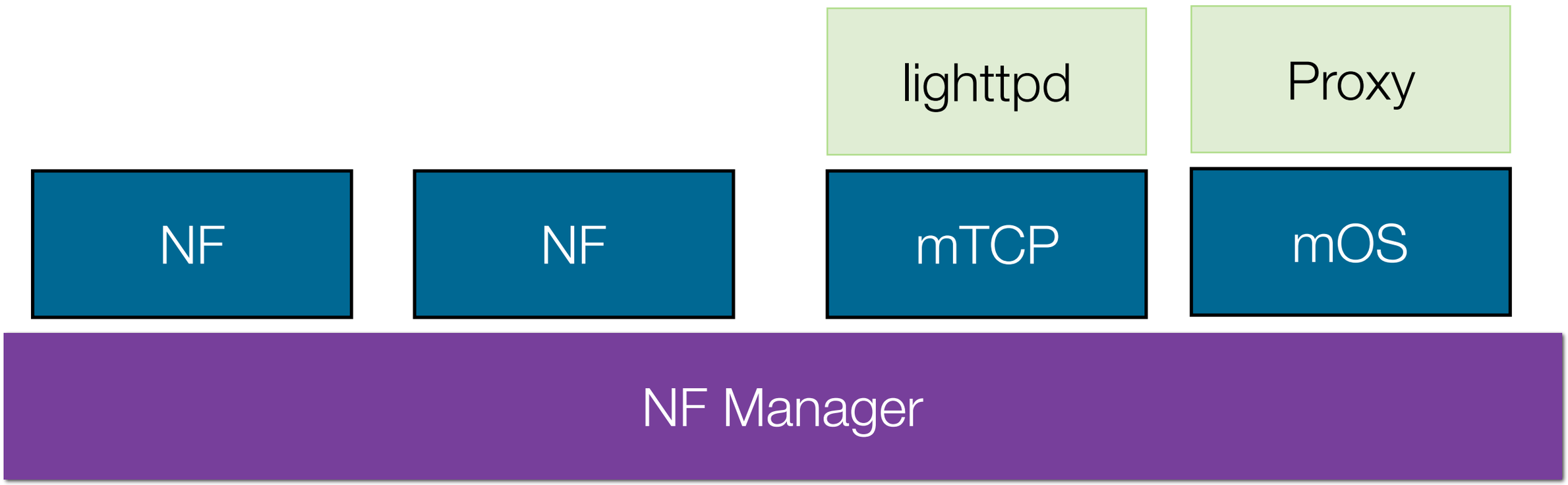


TCP + OpenNetVM

Made at GW!

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

- NF services could expose APIs to work with endpoints



Networking Exam

What to expect

Midterm

What have we covered?

Course Outline

Network Layering

- Protocol layers, software layers, etc

Socket APIs

- Don’t need to know code, but should be able to read it

UDP and TCP

- Pros and cons, basic principles

Threading Architectures

- Thread pools, go routines, non-blocking / event based

Performance Metrics

- Latency vs Throughput, what affects each, basic equations

Middleboxes

- Kernel bypass principles

Midterm

Questions to test your understanding

- Apply principles, not memorize them

Closed book, closed notes

You may bring:

- 1 double sided sheet of 8.5x11 paper
- with **handwritten** notes