

ESTR4120 Computer Networks

Lecture 9: Network Function Virtualization

Hong Xu

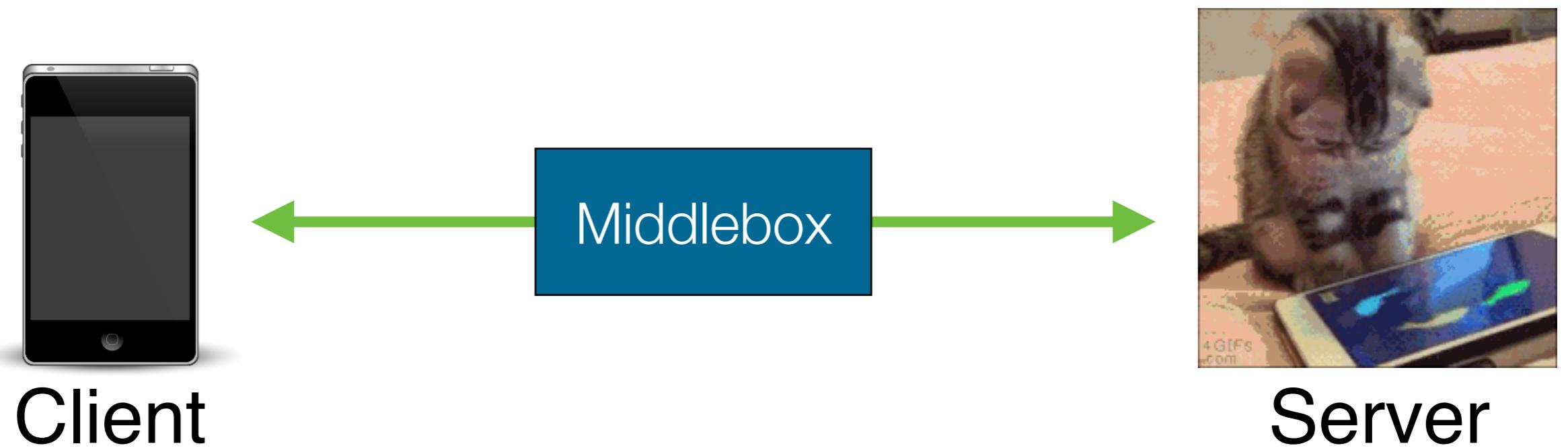
Materials from Timothy Wood's slides on CSCI 3907 at GWU

Advanced Networking and Distributed Systems

Module 3: Network Middleboxes

GW CSCI 3907/6907
Timothy Wood and Lucas Chaufournier

Middleboxes



Client

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



Compare

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 ▶](#)

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...

Router Firewall



Router

Switch

Virtualization Layer



Commodity Server

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]

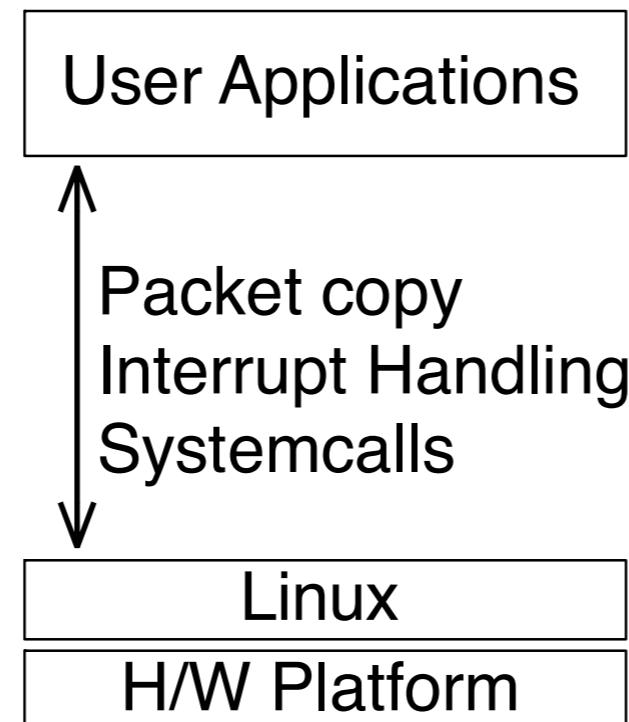
- VM based network services
- Flexible deployment and composition



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

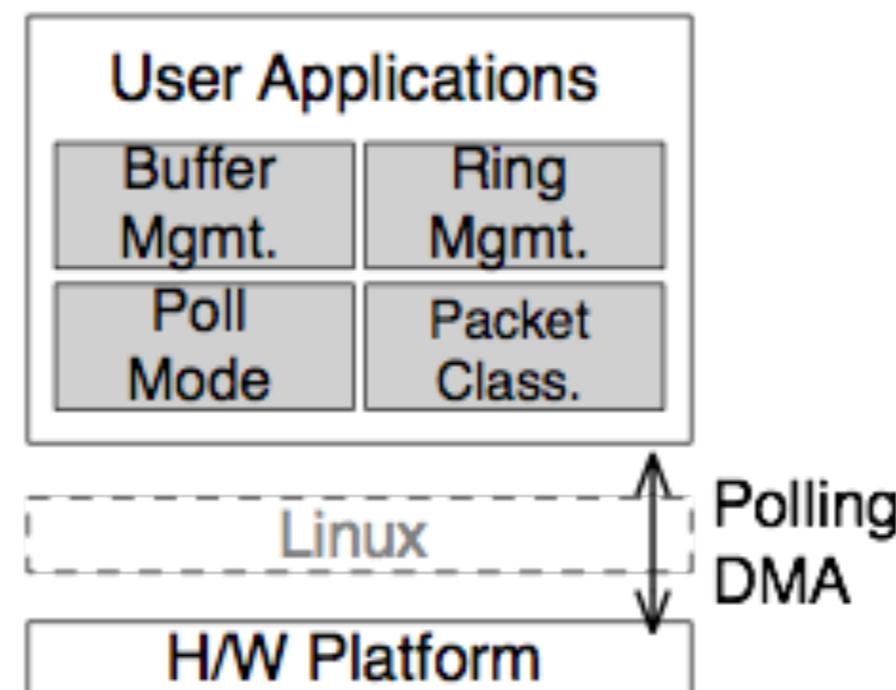


Can you handle being interrupted 60 million times per second?

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

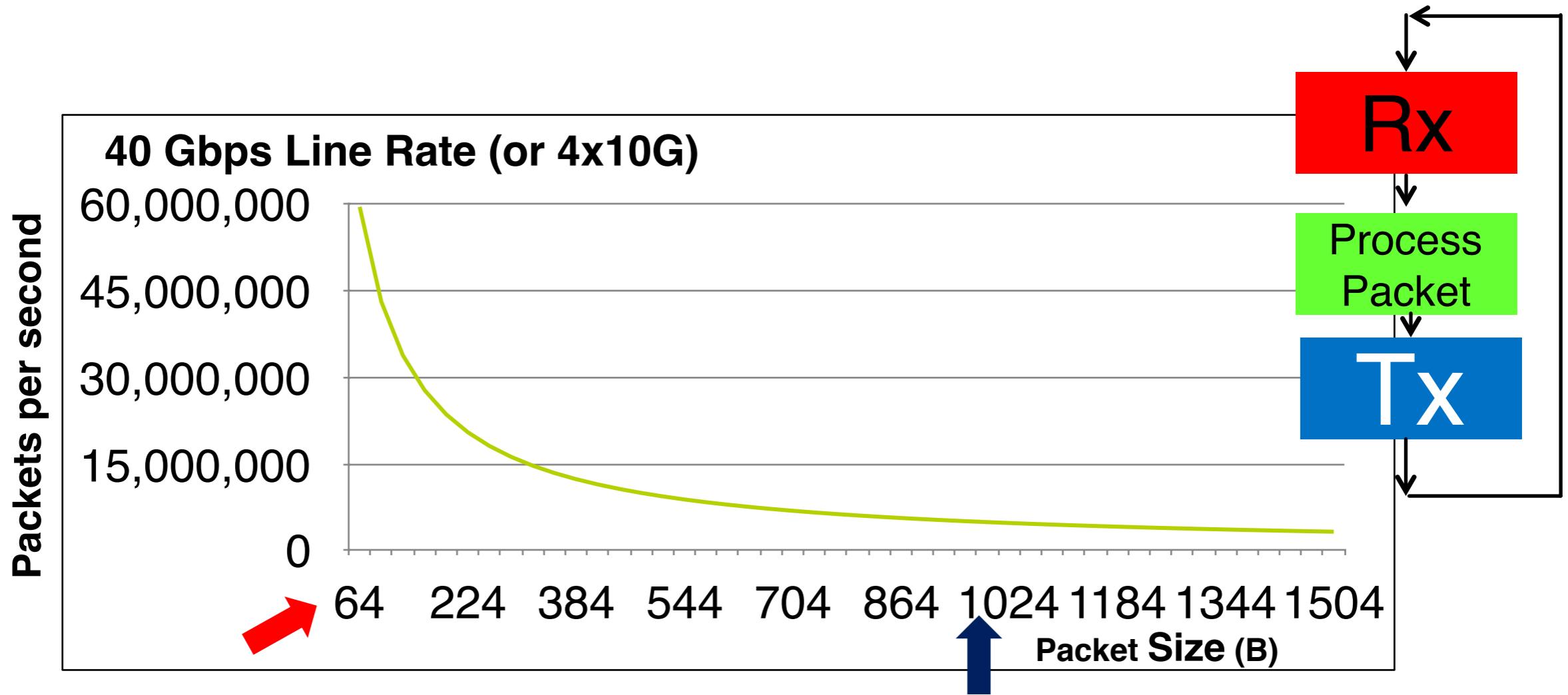
Why try it:

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

Alternatives:

- netmap
- PF_RING

What is “line rate”?



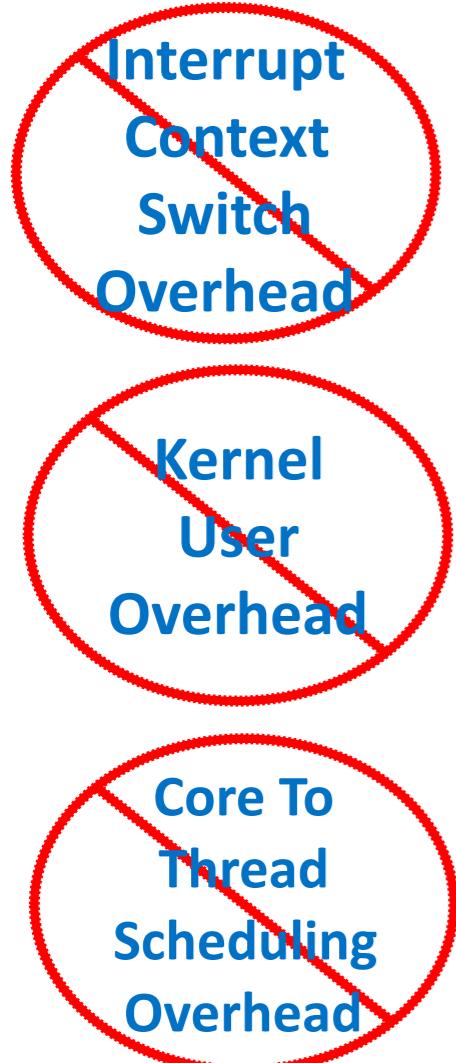
Network Infrastructure Packet Sizes

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

**User
Mode
Driver**

**Pthread
Affinity**



Huge Pages

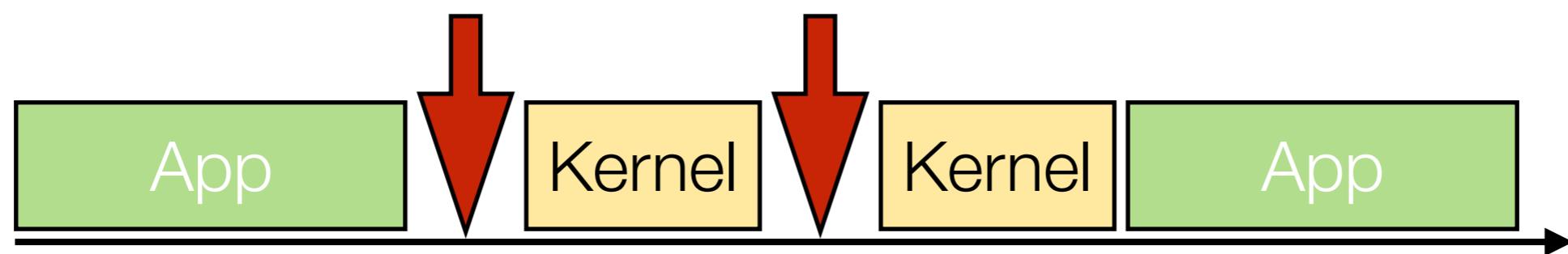
**Lockless Inter-core
Communication**

**High Throughput
Bulk Mode I/O calls**

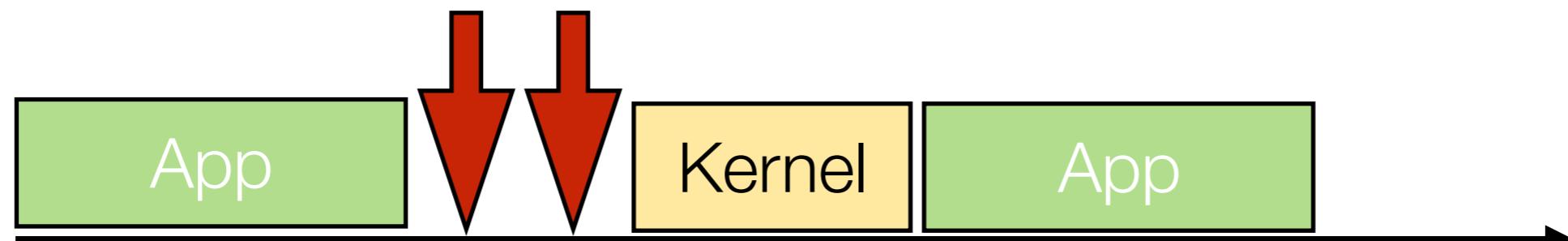
~~Interrupt
Context
Switch
Overhead~~

Network Interrupts

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



Coalescing interrupts helps, but still causes problems



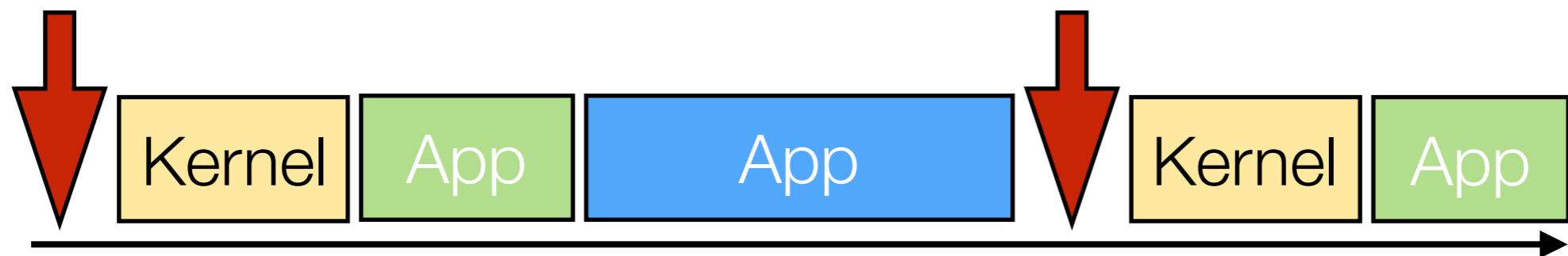
- Interrupts can arrive during critical sections!
- Interrupts can be delivered to the wrong CPU core!
- Still must pay context switch cost

Polling

Interrupt
Context
Switch
Overhead

Continuously loop looking for new packet arrivals

Trade-off?



Interrupts help share the CPU



Polling can be wasteful

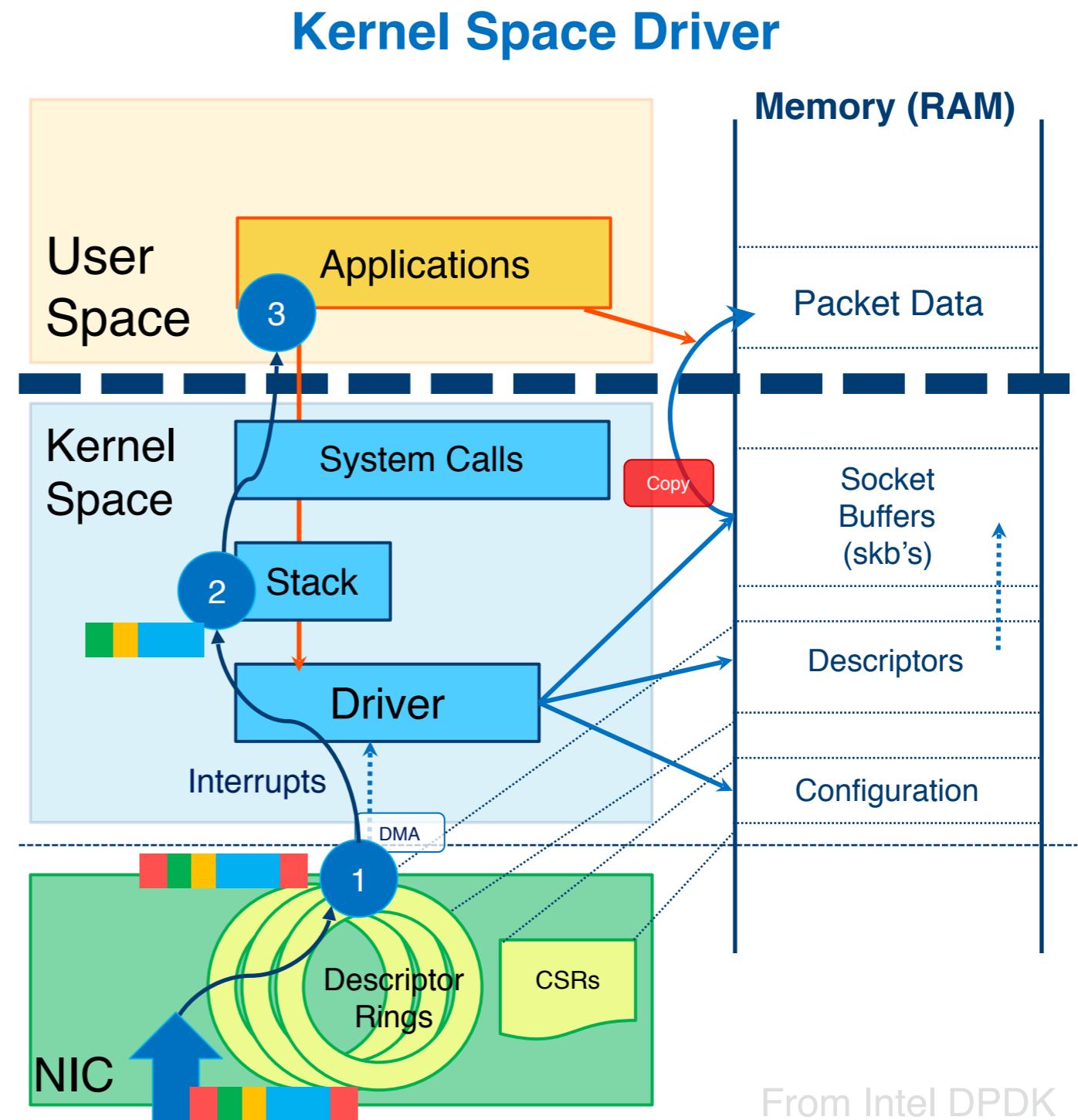
Kernel-User Overhead

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

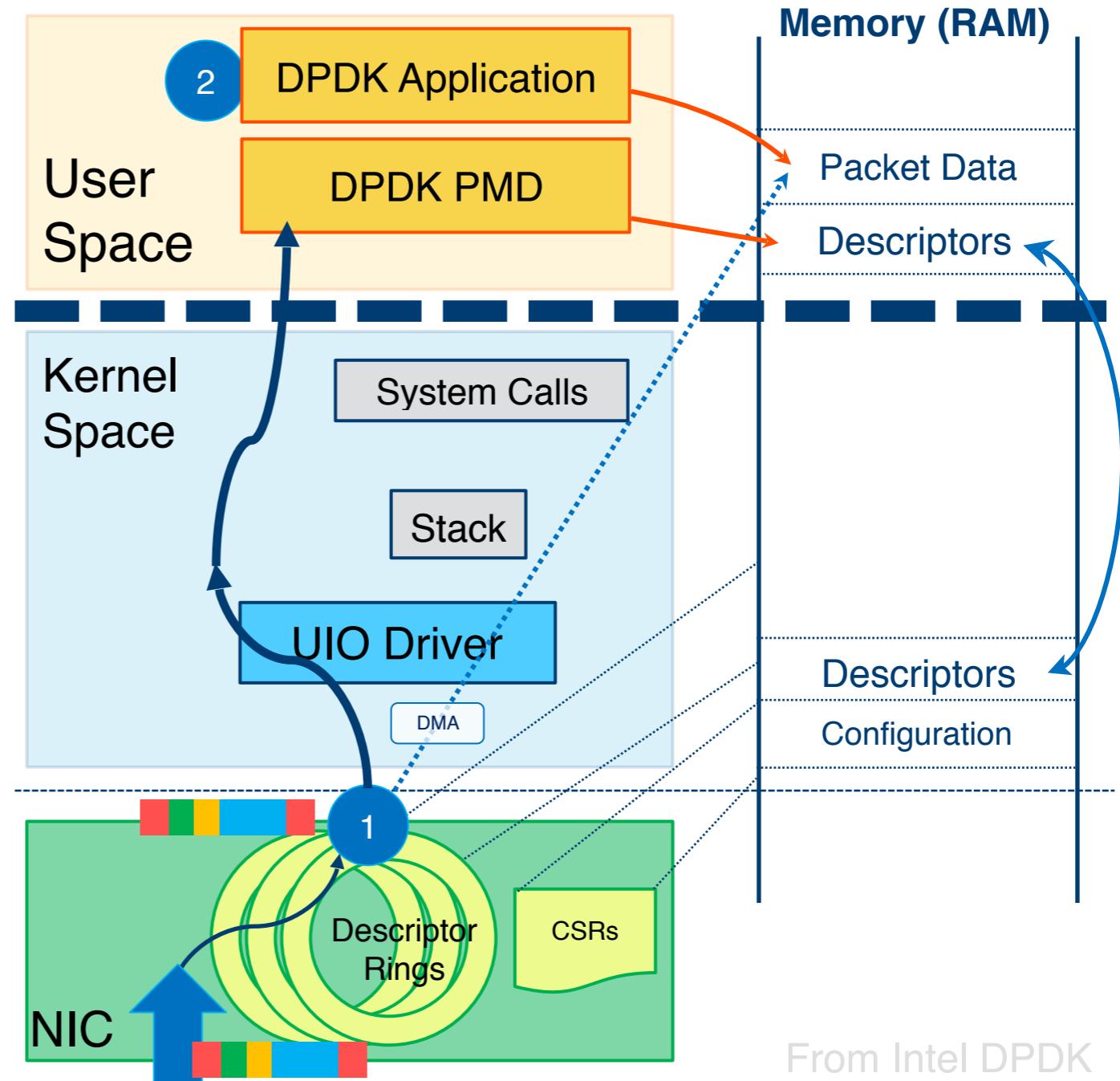
Kernel Bypass



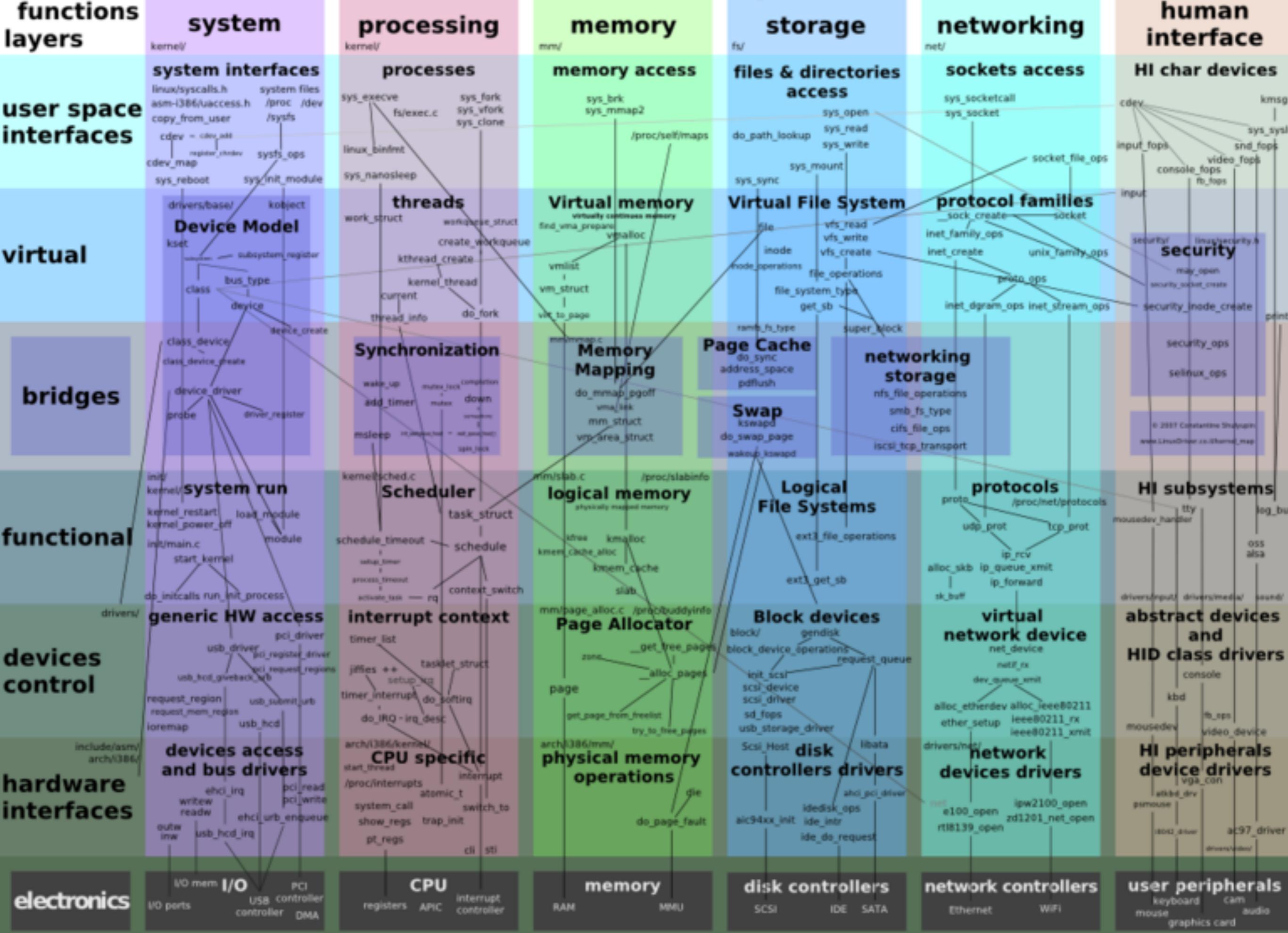
User-mode Driver

- 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

User Space Driver



From Intel DPDK
University Lecture



Networking

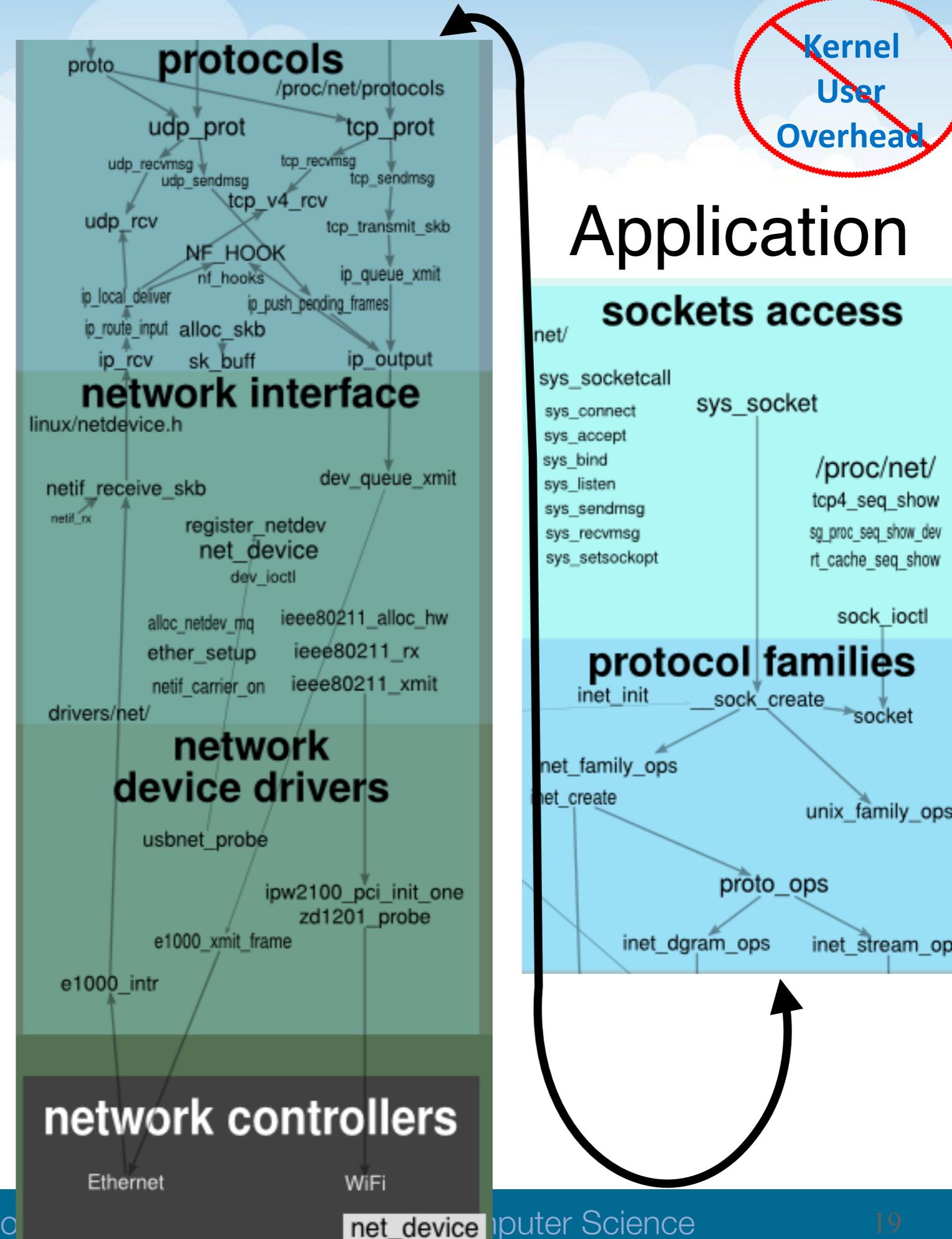
Linux networking stack has a lot of extra components

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

- TCP, UDP, sockets

NFV middle boxes just need packet data

- Need it fast!

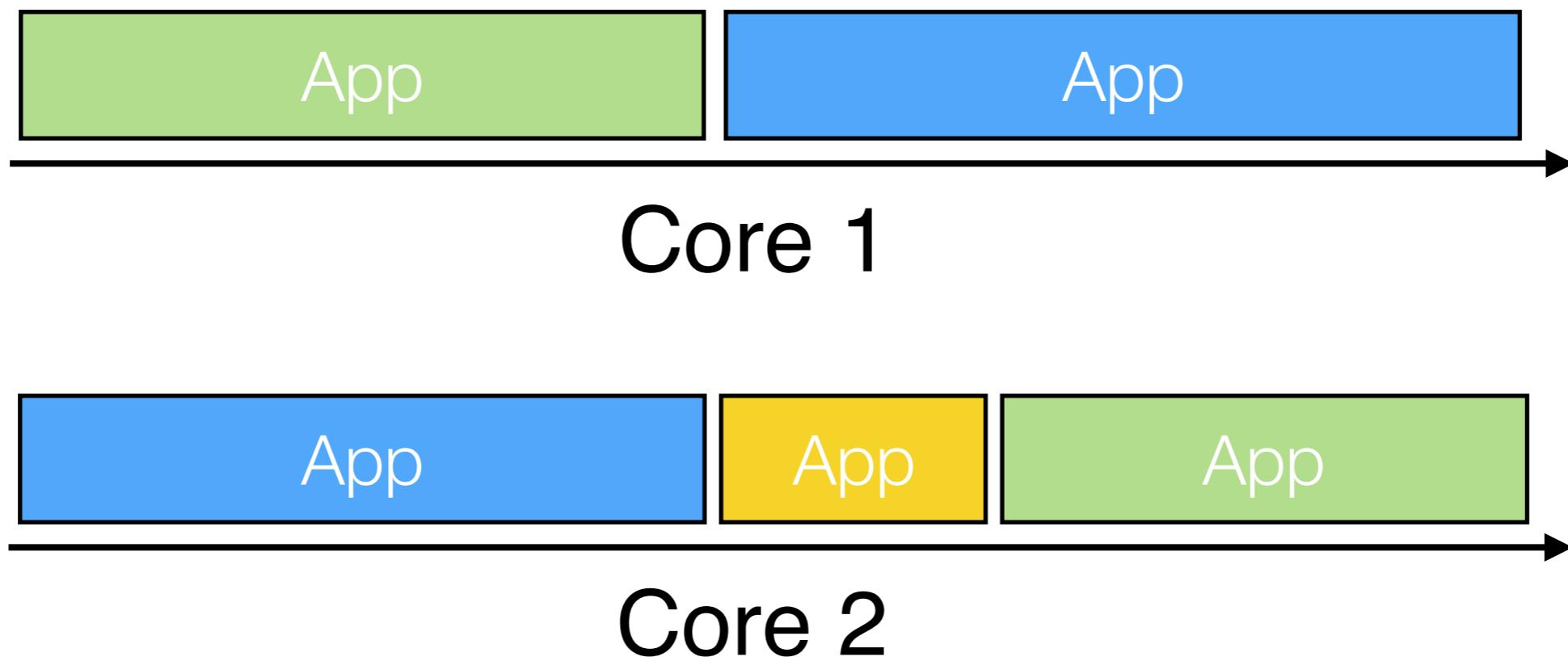


CPU Core Affinity



Linux Scheduler can move threads between cores

- Context switches :(
- Cache locality :(

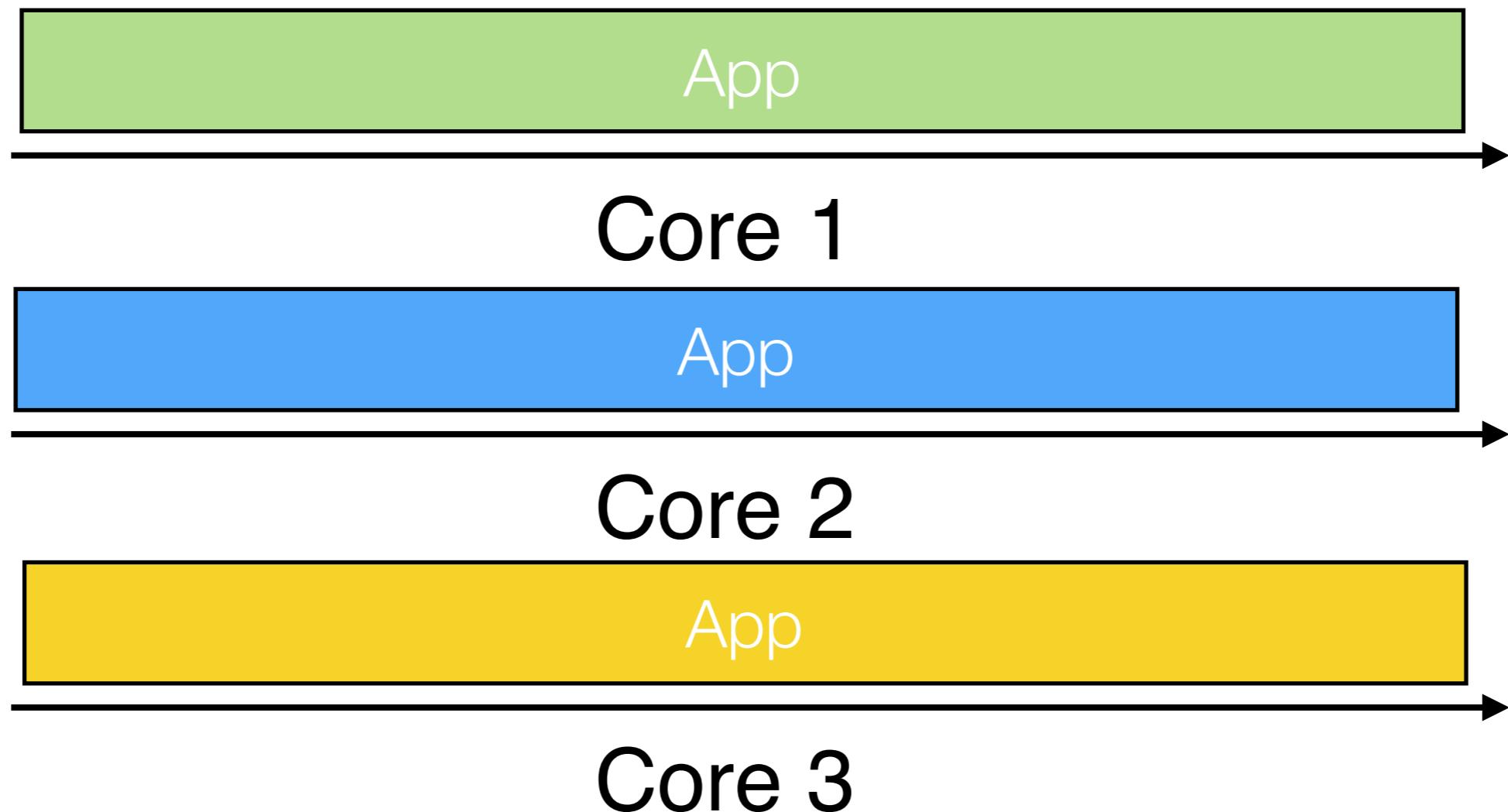


CPU Core Affinity



Pin threads and dedicate cores

- Trade-offs?



4K
Paging
Overhead

Paging Overhead

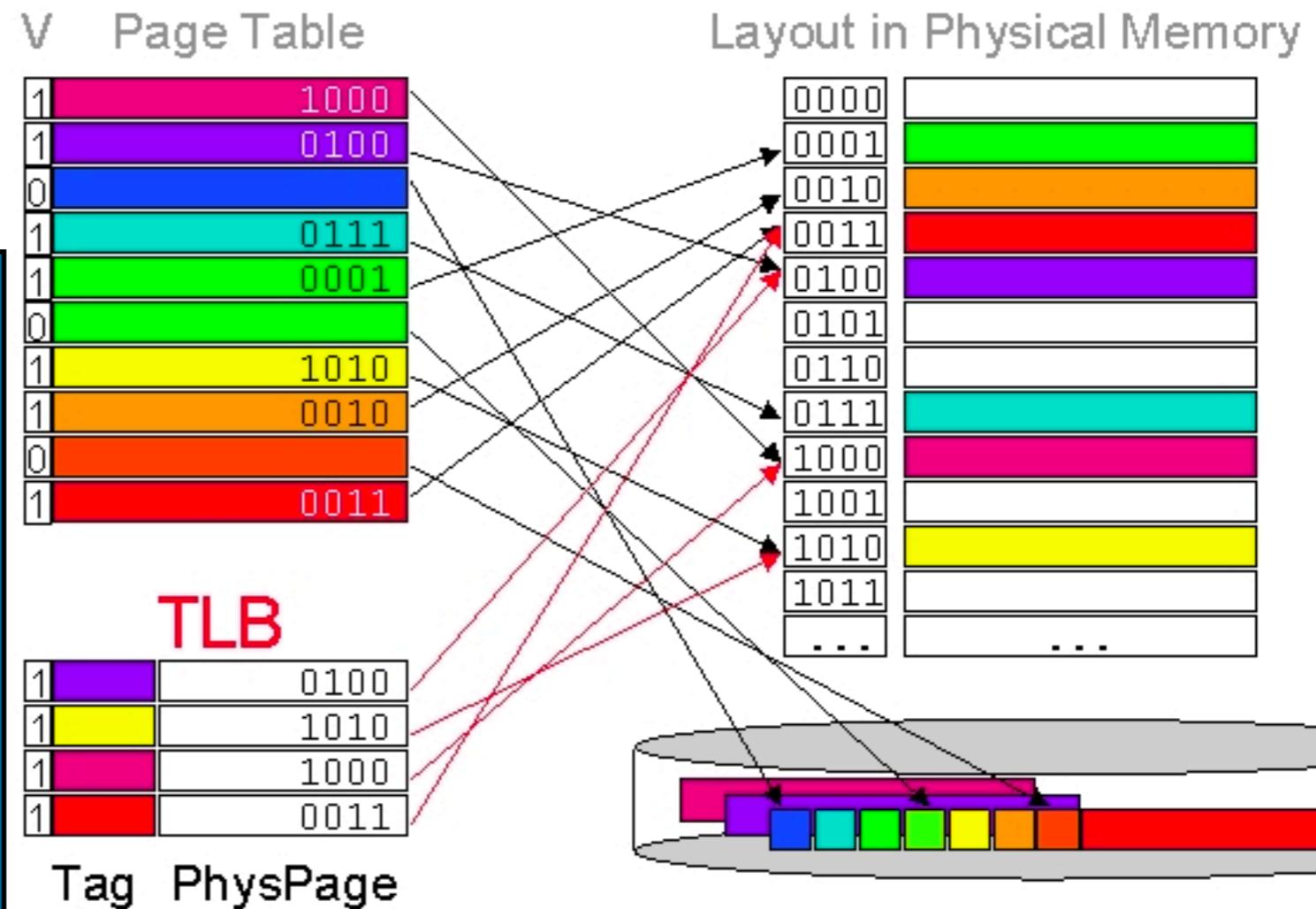
4KB Pages

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

How big is the TLB?

Packet ~ = 1KB

Translation Lookaside Buffer



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 batches of packet descriptors onto rings

Trade-offs?

Limitations

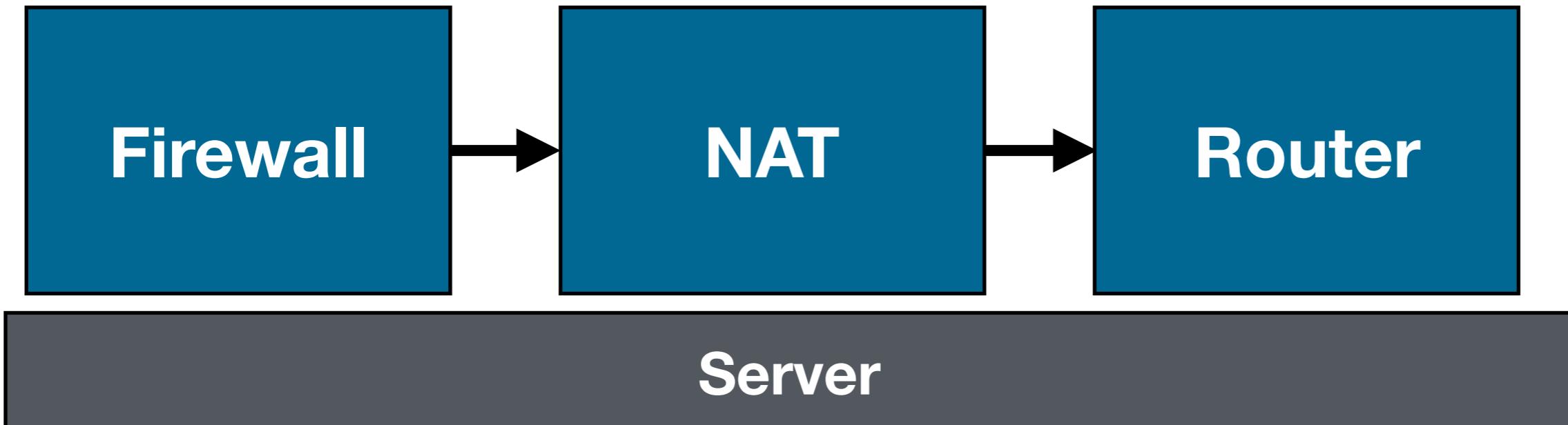
DPDK provides efficient I/O... but that's about it

Doesn't help with NF management or orchestration

Service Chains

Chain together functionality to build more complex services

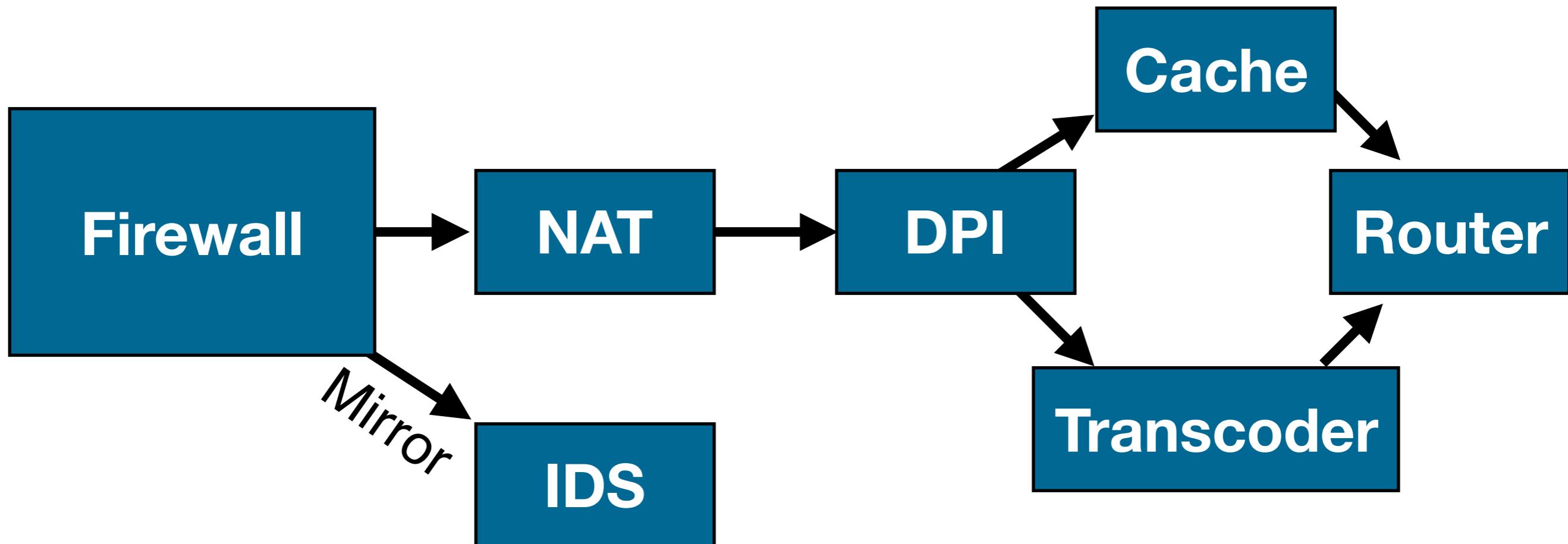
- Need to move packets through chain efficiently



Service Chains

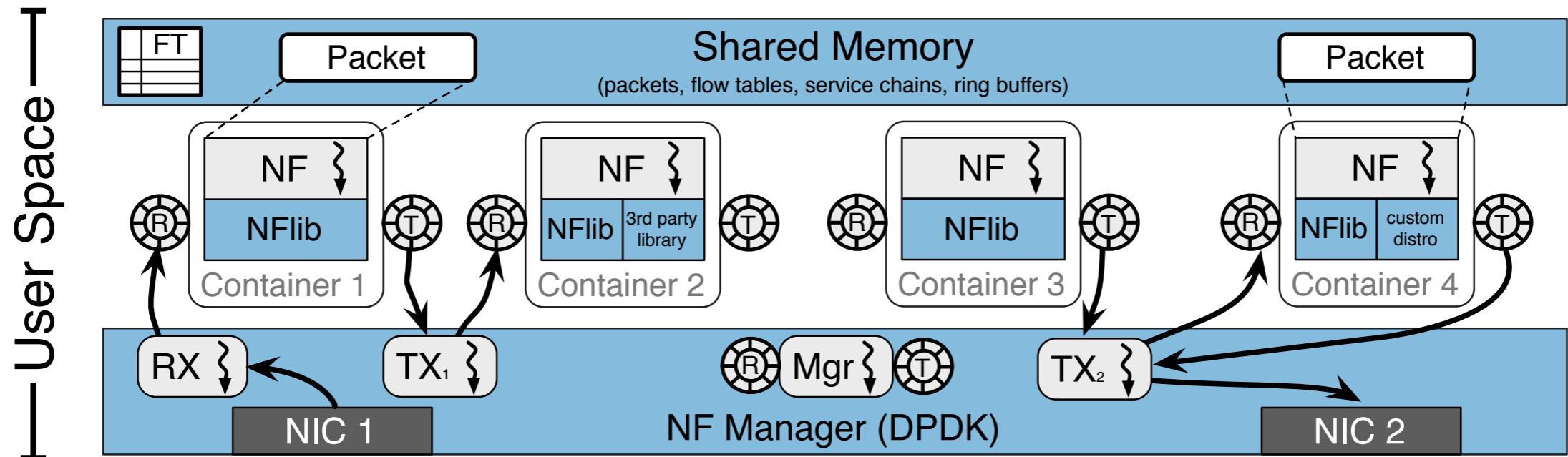
Chain together functionality to build more complex services

- Need to move packets through chain efficiently



Can be complex with multiple paths!

OpenNetVM NFV Platform



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

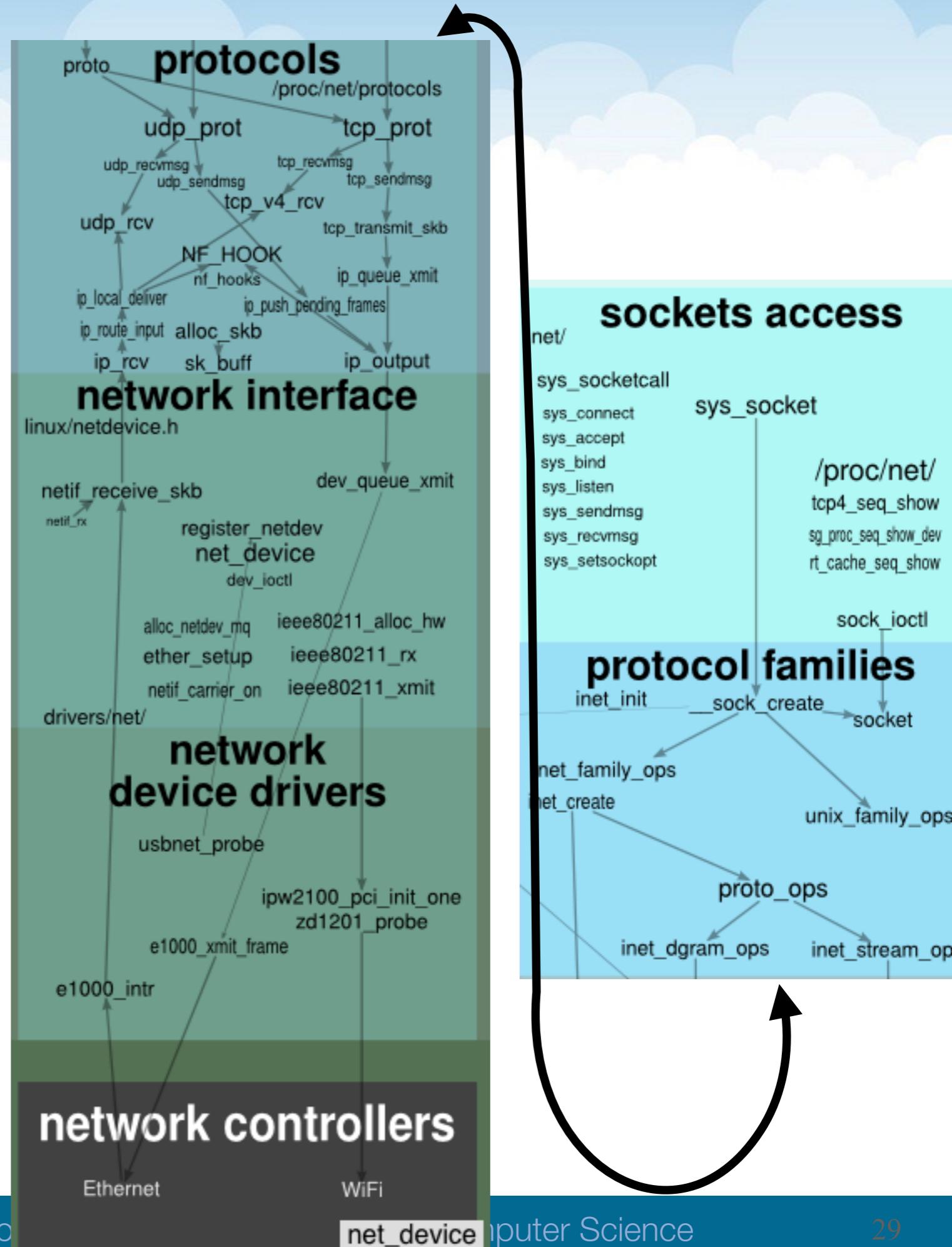
<http://sdnfv.github.io/>

Limitations

DPDK only helps
with raw packet IO

Doesn't provide any
protocol stacks!

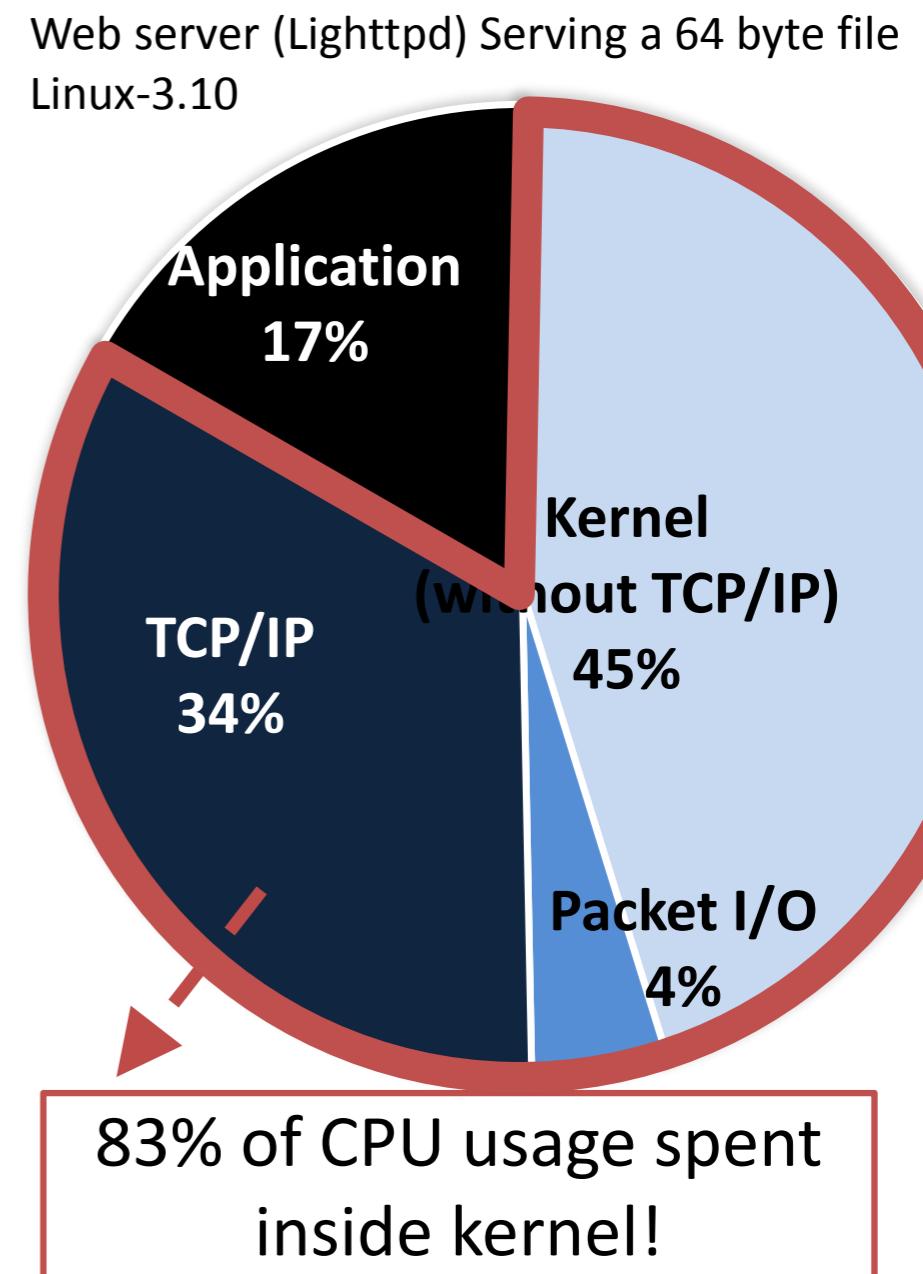
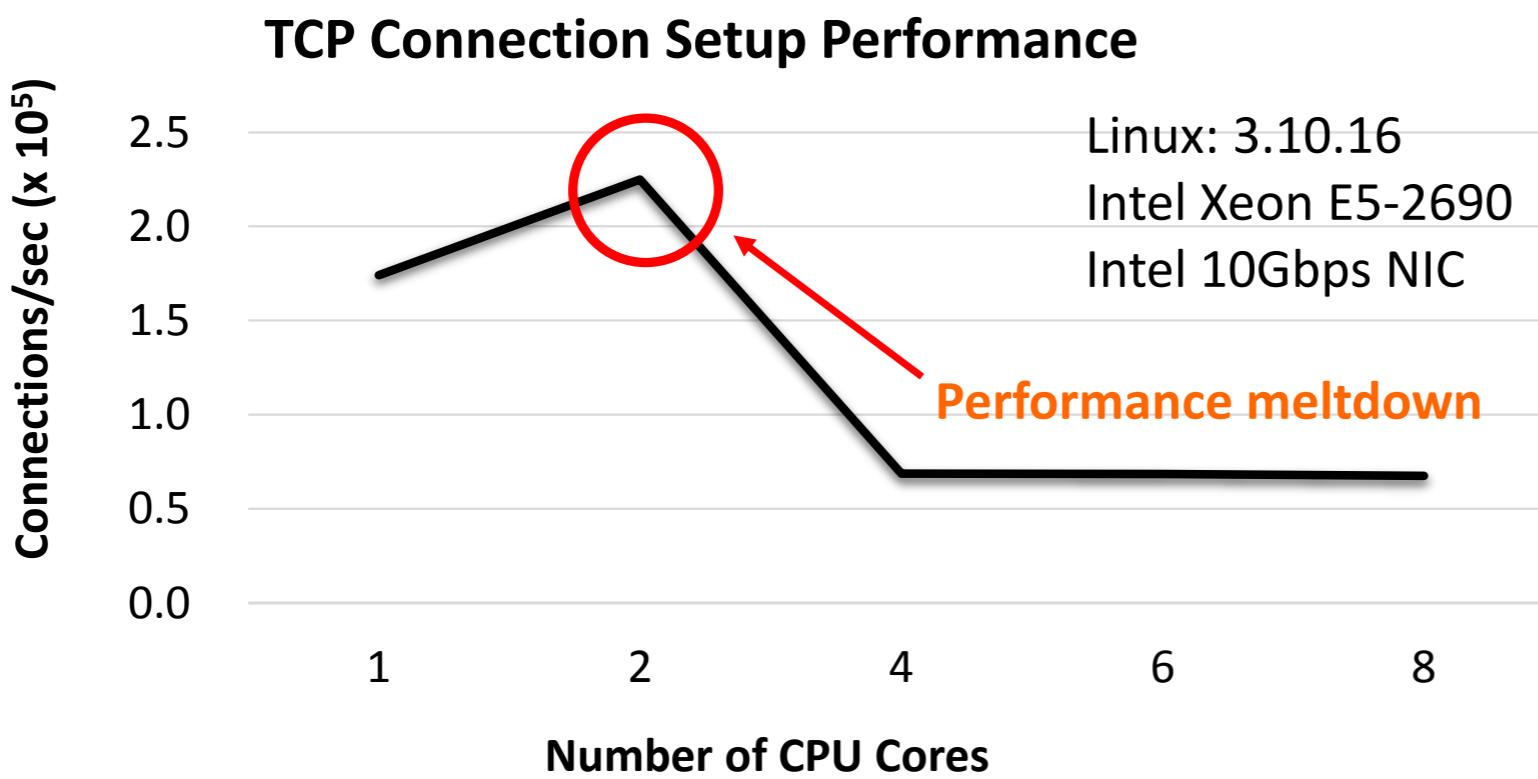
- No IP
- No TCP or UDP
- No socket interface



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

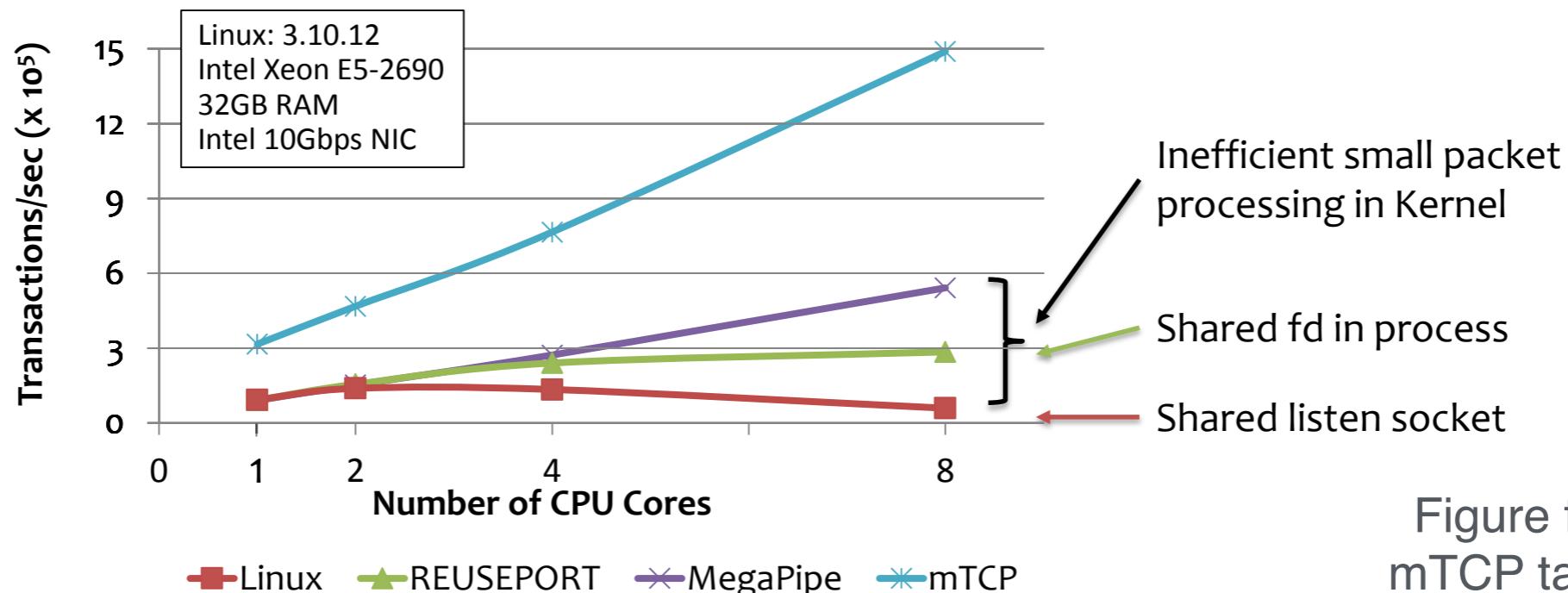


Figure from Jeong's
mTCP talk at NSDI 14

mTCP Kernel Bypass

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

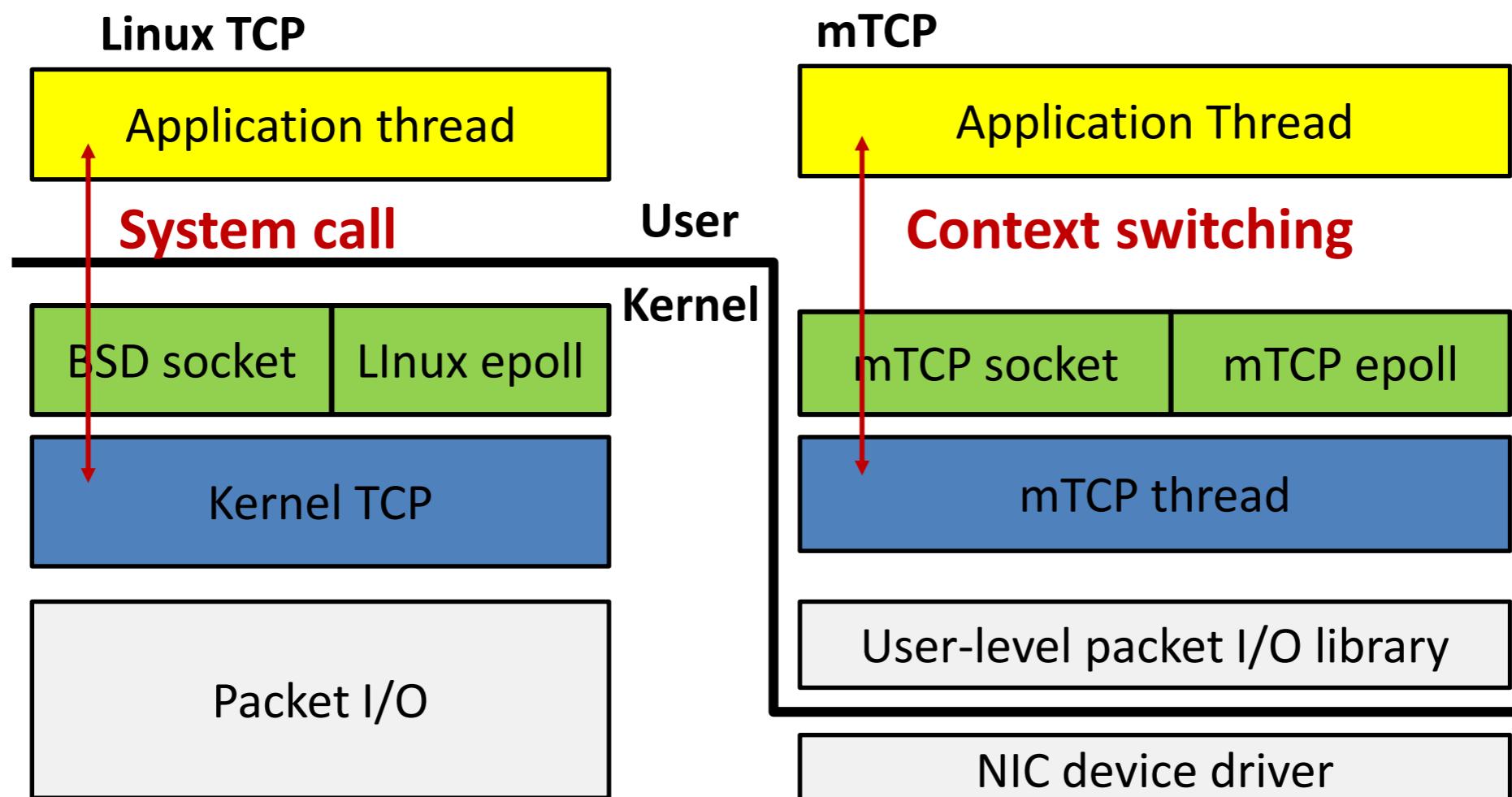
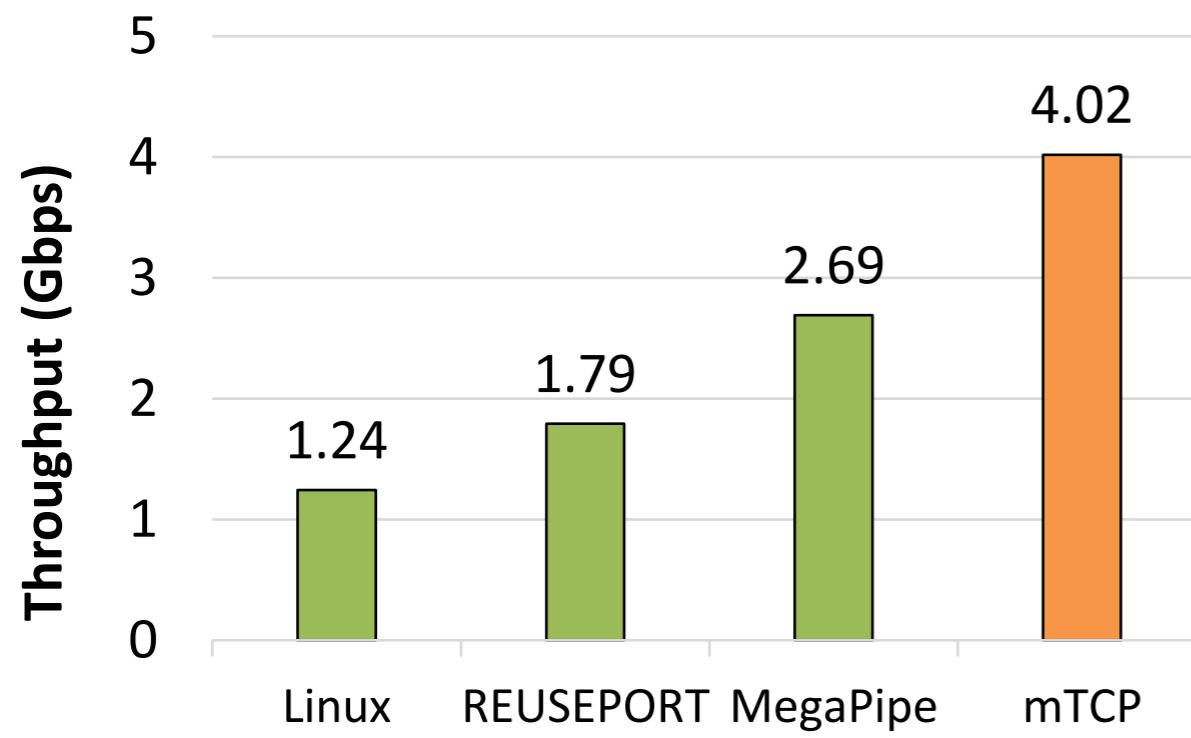


Figure from Jeong's
mTCP talk at NSDI 14

Performance Improvement on Ported Applications

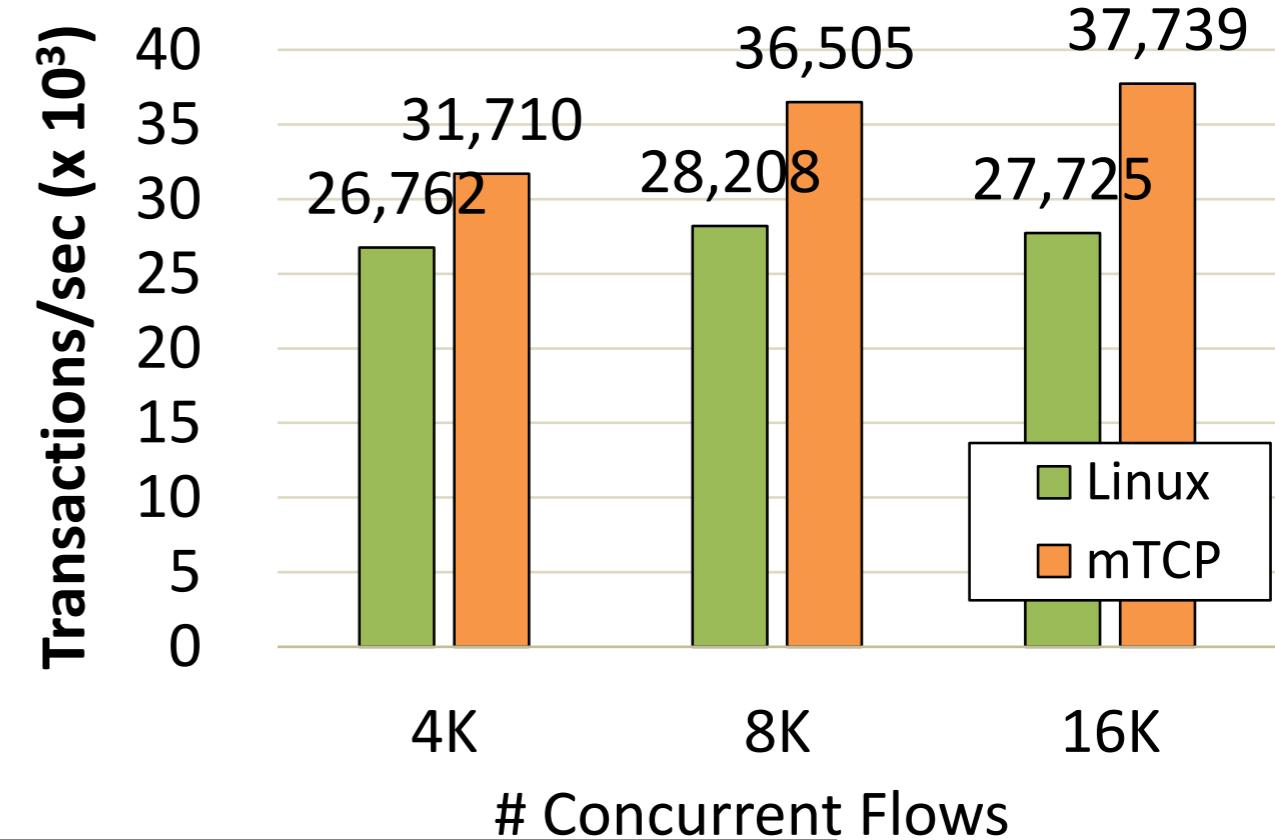
Web Server (Lighttpd)

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



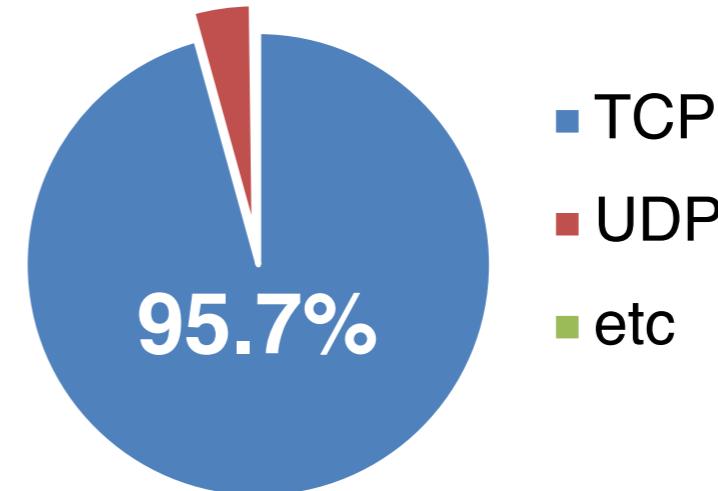
SSL Proxy (SSLS shader)

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



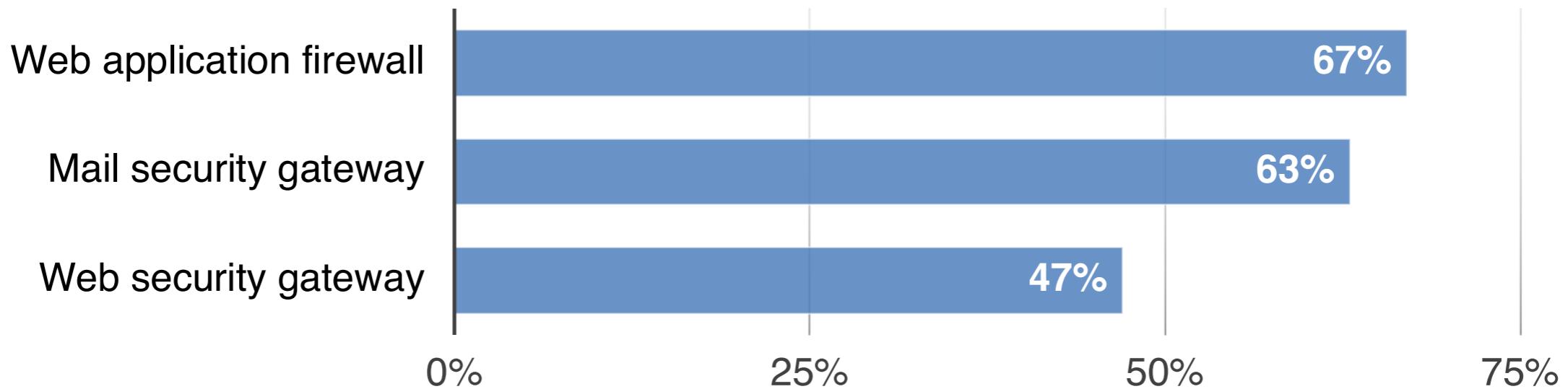
Most Middleboxes Deal with TCP Traffic

- 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.

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

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

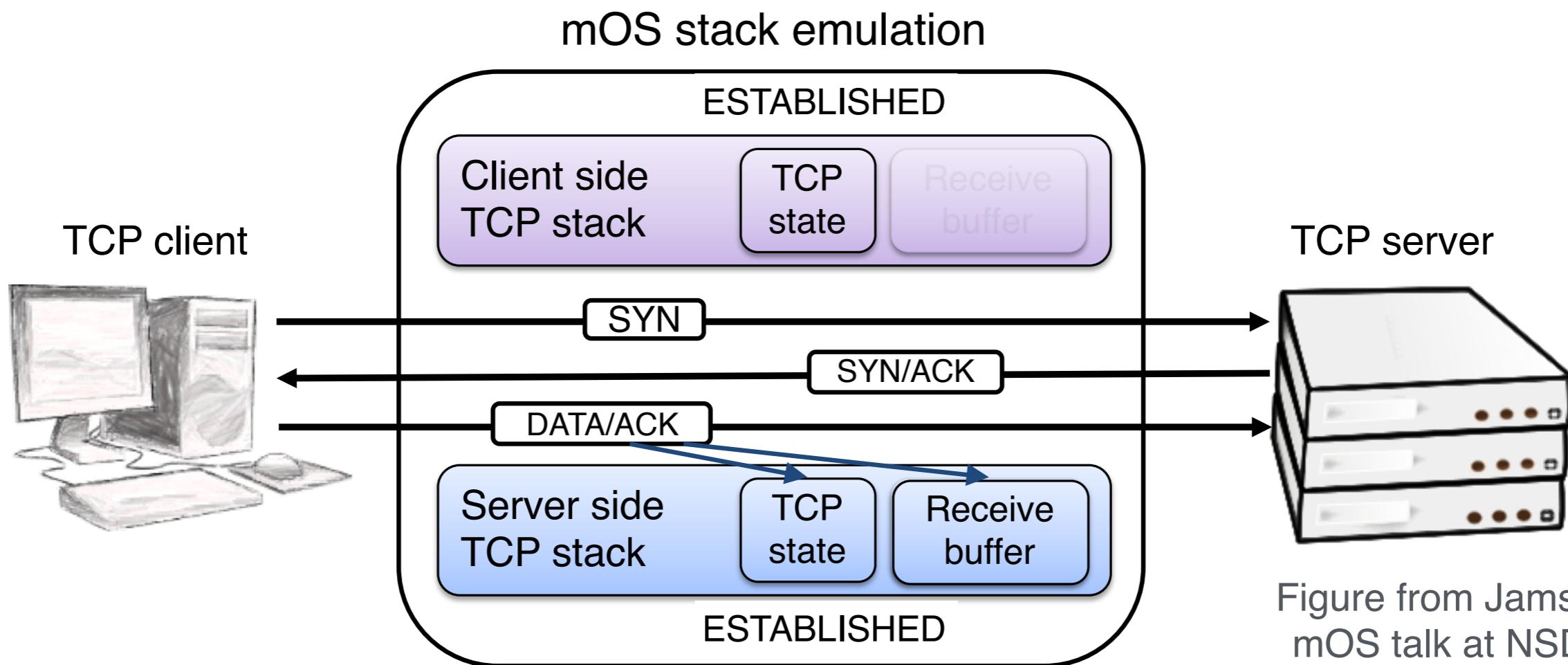


Figure from Jamshed's
mOS talk at NSDI 17

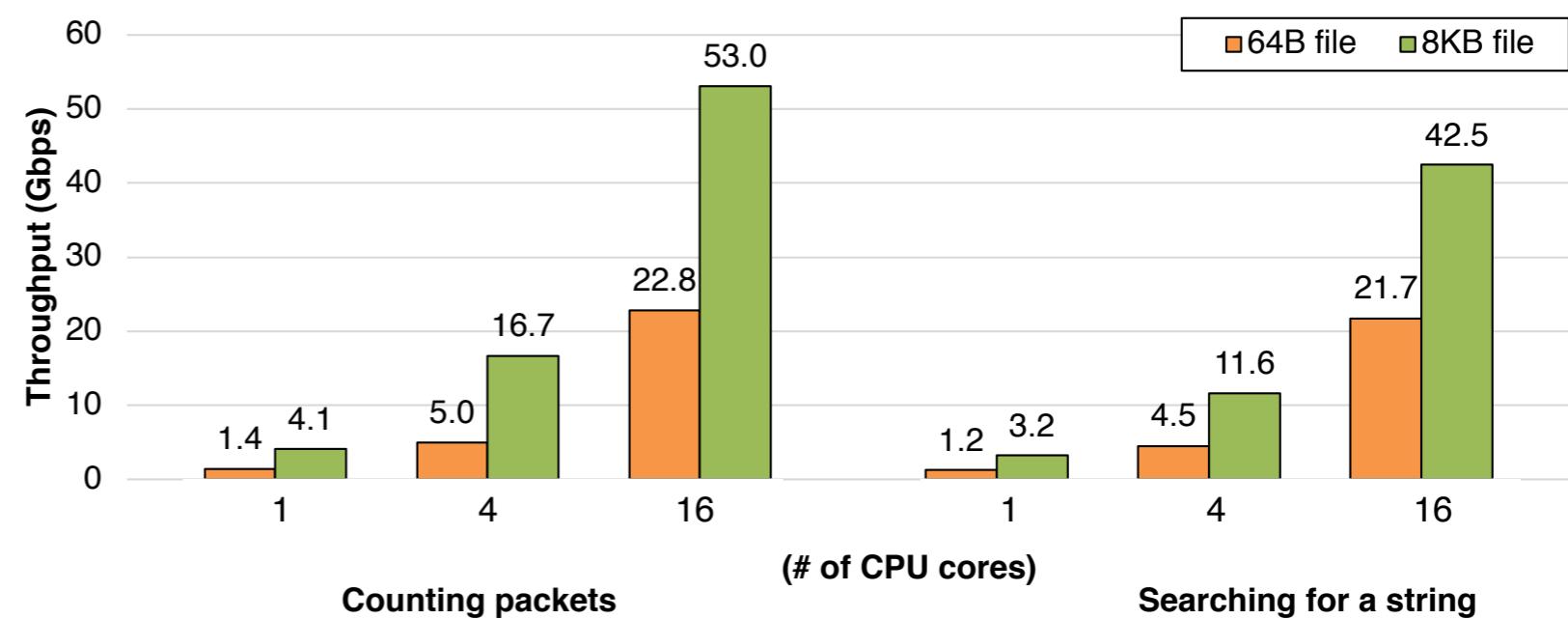
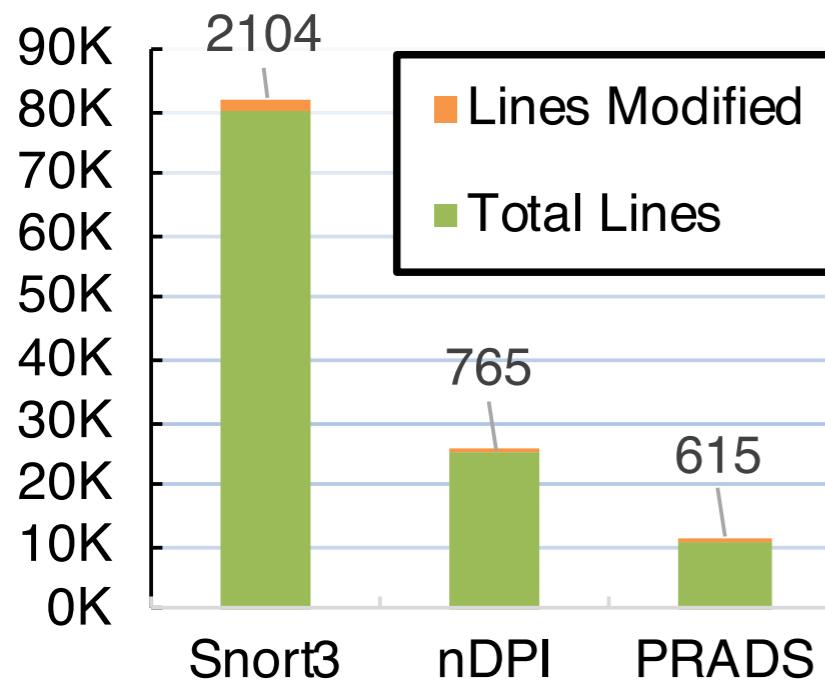
mOS [Jamshed, NSDI '17]

Base Events

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

User Events

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



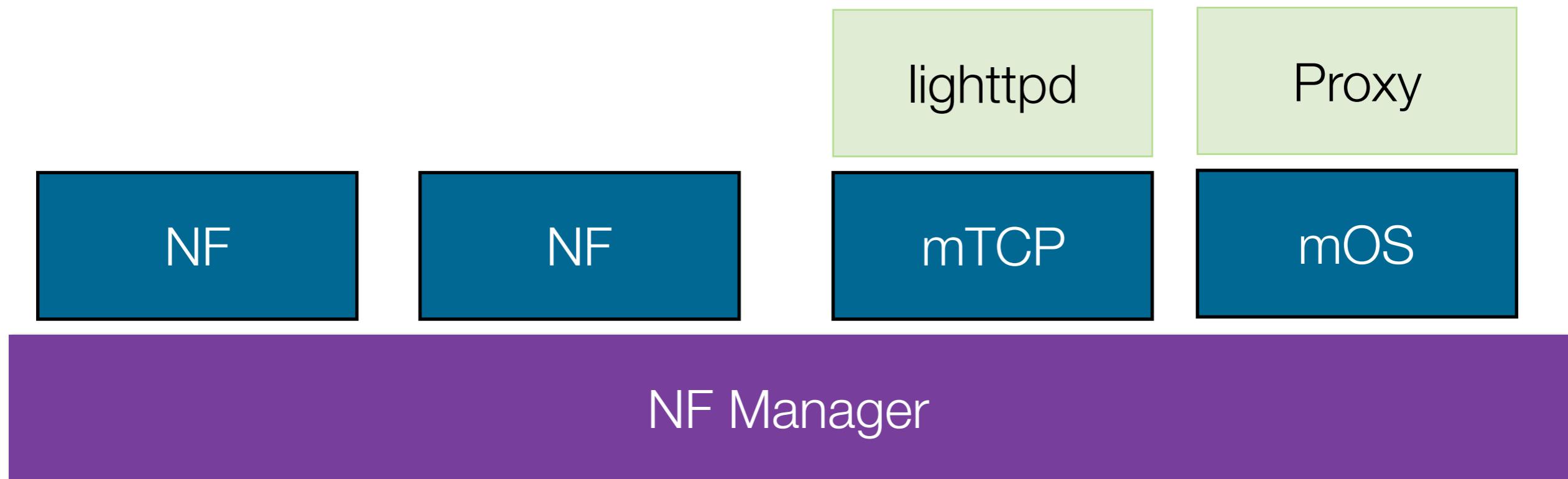
Figures from Jamshed's
mOS talk at NSDI 17

TCP + OpenNetVM

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

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

- NF services could expose APIs to work with endpoints

