Advanced Computer Networks

Endhost Network Stack in Data Center Networks (II)

CS740

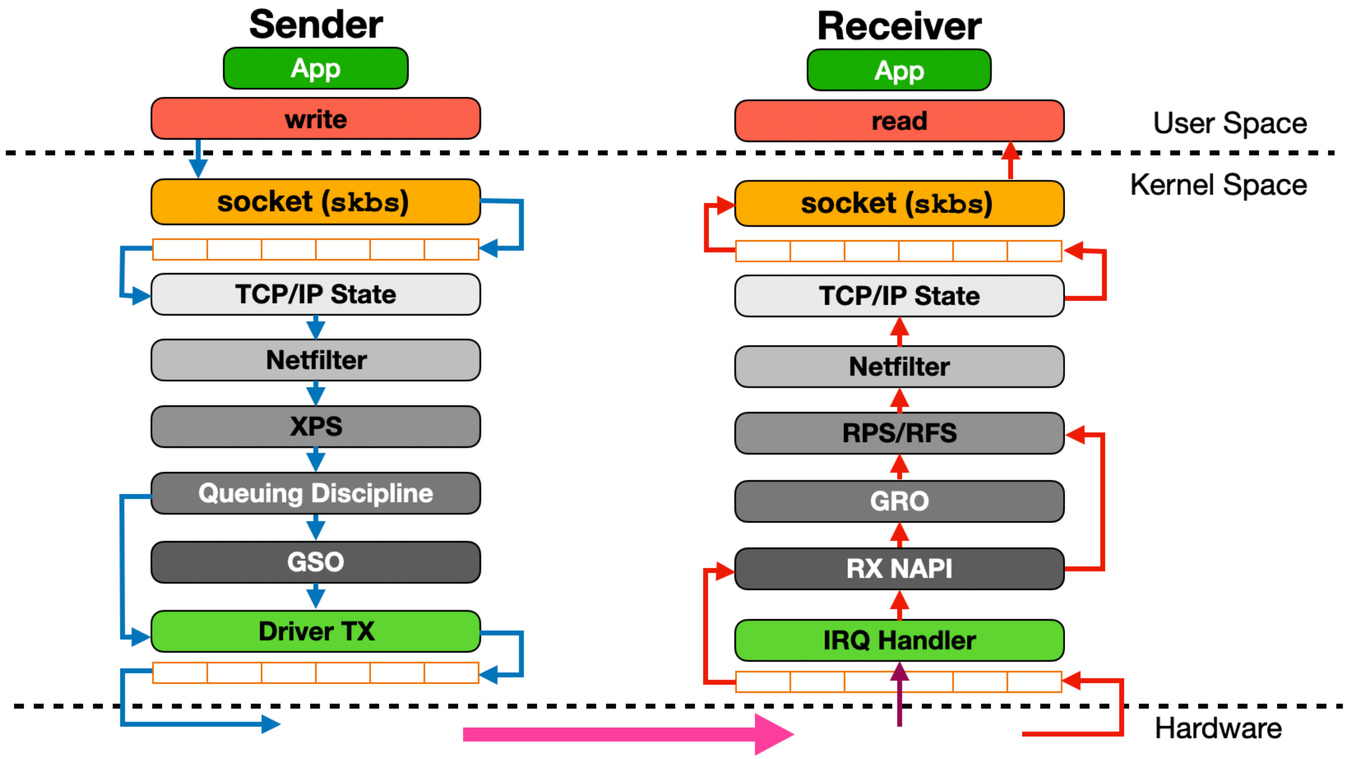
https://pages.cs.wisc.edu/~mgliu/CS740/F26/index.html

Ming Liu
mgliu@cs.wisc.edu

Outline

- Last lecture
  - Endhost Network Stack in Data Center Networks (I)
- Today
  - Endhost Network Stack in Data Center Networks (II)
- Announcements
  - In-class Exam 11/19/2026

Why not Linux networking stack?



It’s hard to develop and deploy new network functionality and performance optimizations.

Development Velocity

Problem: Lengthy Development and Release Cycles

- #1: Developing kernel code is slow
  - Rely on a small pool of software engineers

Problem: Lengthy Development and Release Cycles

- #1: Developing kernel code is slow
  - Rely on a small pool of software engineers

- #2: Runtime feature updates need disconnecting applications
  - (Sometimes) Require rebooting the machine
  - Pace kernel updates
  - Take 1-2 months to deploy new features

Problem: Lengthy Development and Release Cycles

- #1: Developing kernel code is slow
  - Rely on a small pool of software engineers
- #2: Runtime feature updates need disconnecting applications
  - (Sometimes) Require rebooting the machine
  - Pace kernel updates
  - Take 1-2 months to deploy new features
- #3: The broad generality of Linux makes optimization difficult
  - Easily broken by upstream changes

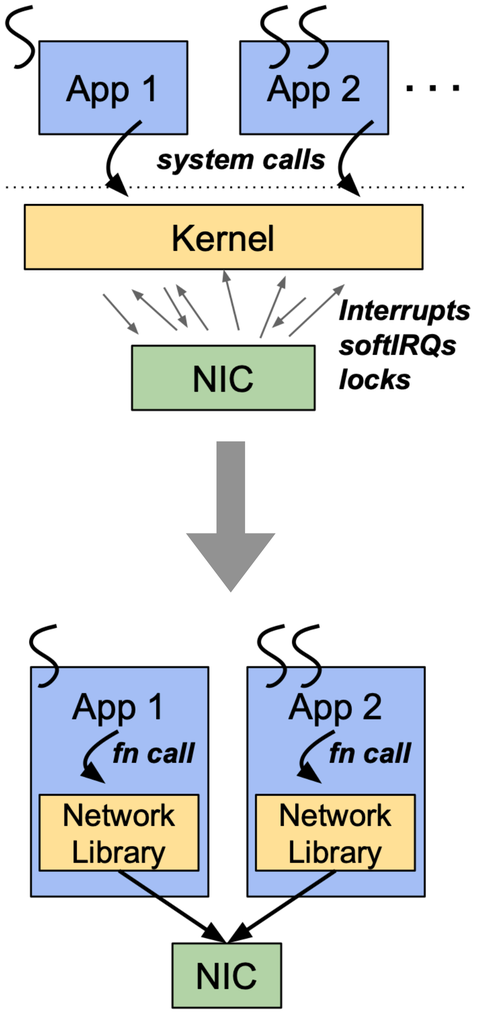
Problem: Lengthy Development and Release Cycles

- #1: Developing kernel code is slow
  - Rely on a small pool of software engineers
- #2: Runtime feature updates need disconnecting applications
  - (Sometimes) Require rebooting the machine
  - Pace kernel updates
  - Take 1-2 months to deploy new features
- #3: The broad generality of Linux makes optimization difficult
  - Easily broken by upstream changes

- **#4: Performance overheads from system calls, fine-grained synchronization, interrupts, and more.**

Prior Solution: OS Bypass and Library OS

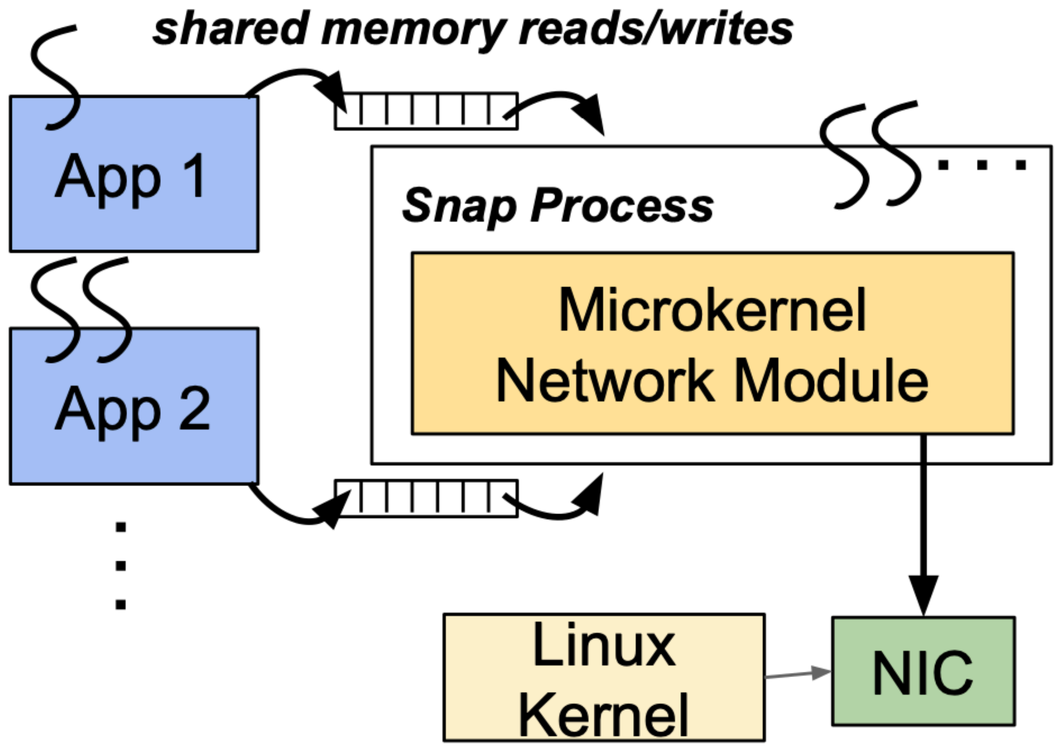
- Integrate networking logic in application libraries
  - E.g., Arrakis (OSDI'14), mTCP (NSDI'13), IX (OSDI'14)
  - Lab1
- Development velocity
  - Difficult to release changes to the fleet
  - App binaries may go months between releases
- Performance
  - Can be very fast
  - Require spin-polling in every application
  - No centralization



How does Snap tackle the problem?

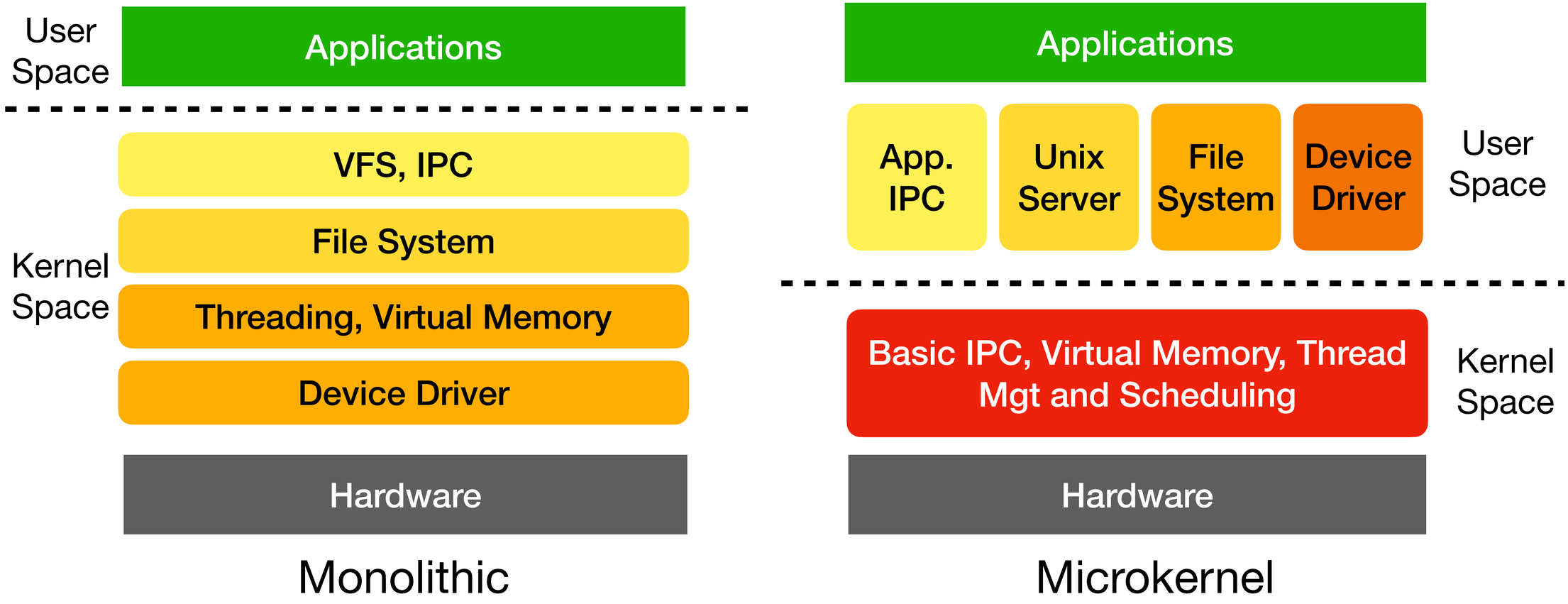
Snap: a Microkernel Approach

- Divide net functionalities into separated userspace processes



Microkernel

- A small privilege software abstracting the underlying hardware
  - Including virtual memory, IPC, thread management and scheduling, etc.
  - MINIX 3 microkernel: ~12K



Snap Benefits

- Development velocity
  - Decouple release cycles from application and kernel binaries
  - Transparent upgrade with iterative state transfer

- Performance
  - Fast due to kernel bypass and multi-core processing
  - Hold the centralization benefit and enrich scheduling/multiplexing policies

How does Snap work?

Snap End-to-End Data Path

Sender

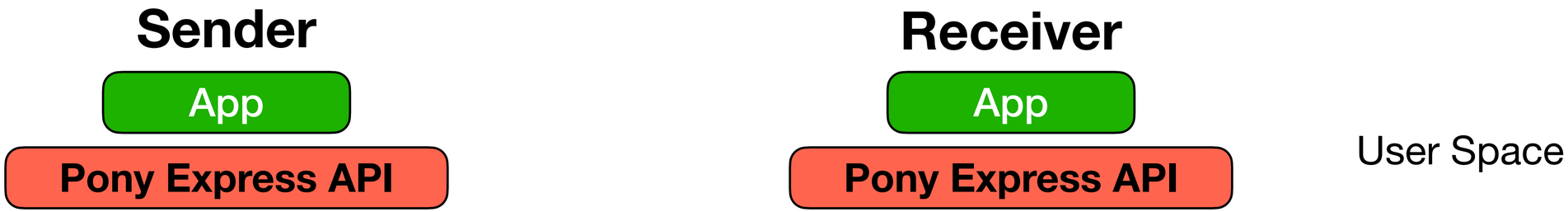
App

Receiver

App

User Space

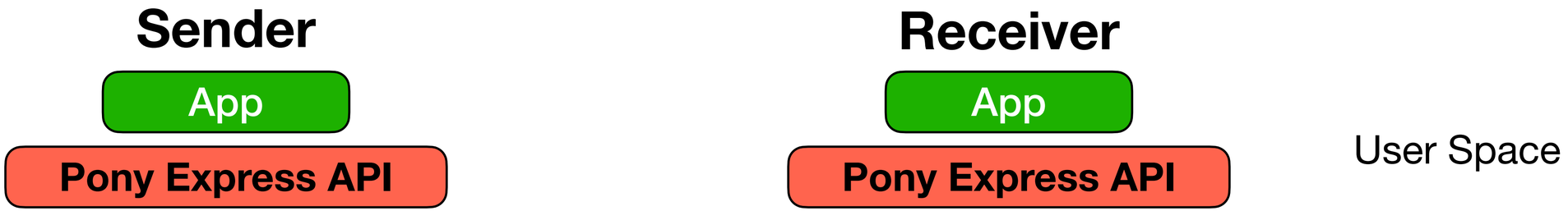
Snap End-to-End Data Path



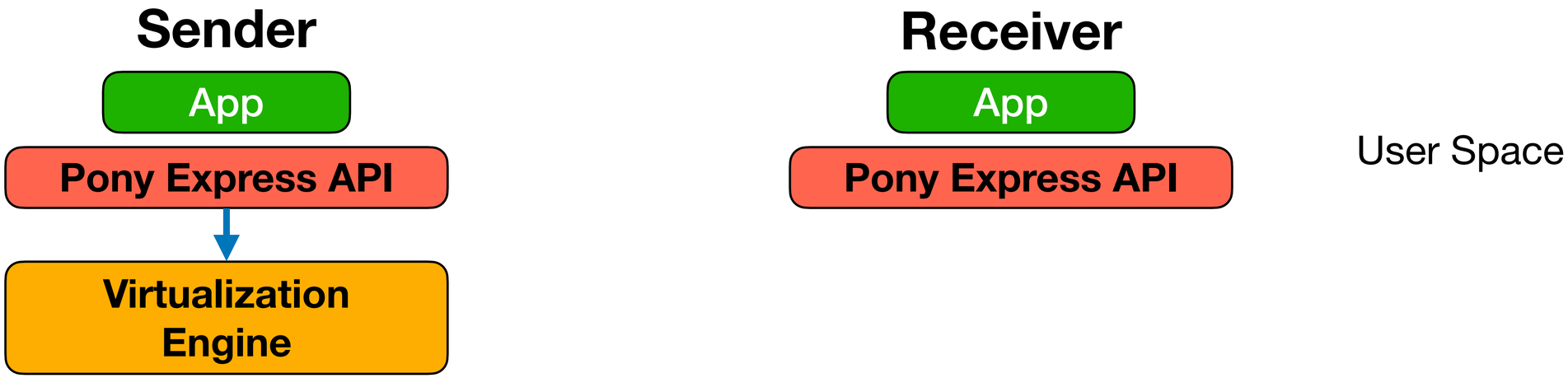
Pony Express

- A reliable transport and communication stack
  - Support messaging interface
  - Support one-sided operations (like RDMA)
- Zero-copy request/response payloads
  - Memory-mapped I/O
- Customized memory allocator
  - Like DPDK mbuf
- Completion delivery
  - Spin-poll the completion queue
  - Use a thread notification

Snap End-to-End Data Path

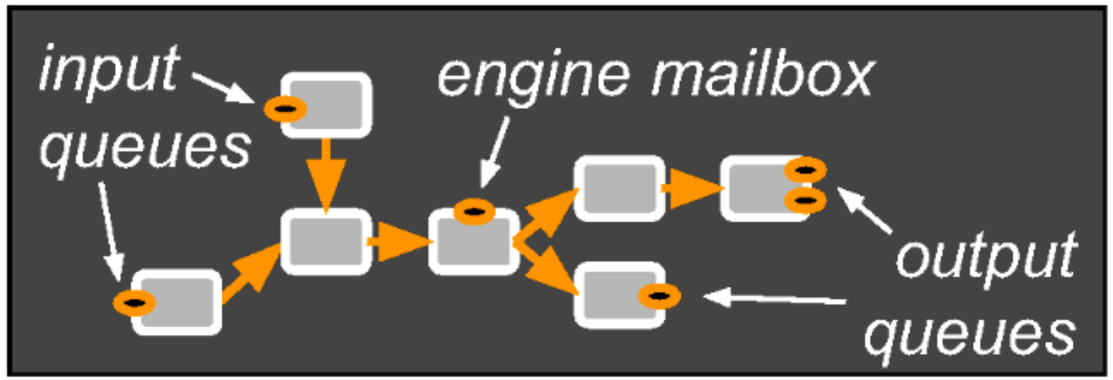


Snap End-to-End Data Path



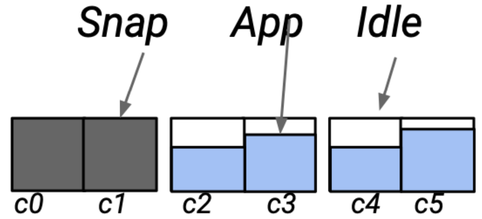
Snap Engine

- Stateful and single-threaded tasks run by a scheduling runtime
  - E.g., protocol processing, ACL enforcement, rate limiting, etc.
  - Implement a `Run()` method
- Key data plane element to build a packet processing pipeline
  - Lock-free communication
  - Unit of CPU scheduling and scaling



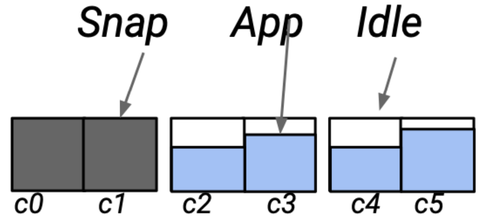
Snap Engine Scheduling

- #1: Dedicated Cores
  - Static provisioning of N cores to run engines
  - Simple and best for some situations



Snap Engine Scheduling

- #1: Dedicated Cores
  - Static provisioning of N cores to run engines
  - Simple and best for some situations

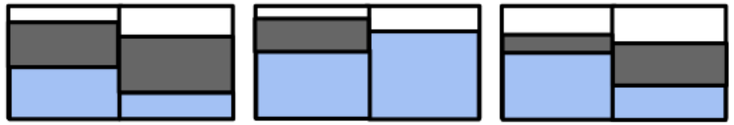


- Drawback: hard to provision
  - Provisioning for the worst case is wasteful
  - Provisioning for the average case leads to high tail latency

Snap Engine Scheduling

- #2: Spreading Engines
  - Bind each engine to a unique kernel thread
  - Interrupts triggered from NIC or application to schedule on-demand
  - Employ a new micro-quanta kernel scheduling class for low-latency

Snap Spreads

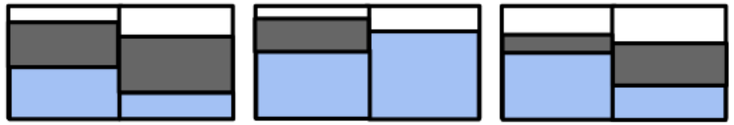


Snap Engine Scheduling

- #2: Spreading Engines
  - Bind each engine to a unique kernel thread
  - Interrupts triggered from NIC or application to schedule on-demand
  - Employ a new micro-quanta kernel scheduling class for low-latency

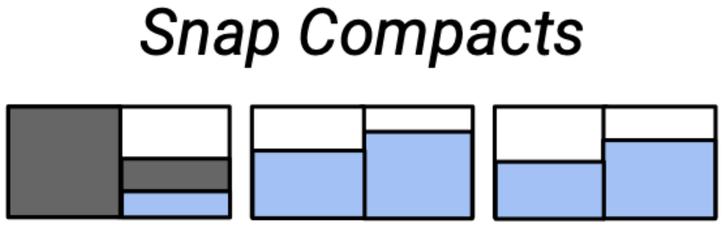
- Drawback
  - Scheduling overhead

Snap Spreads



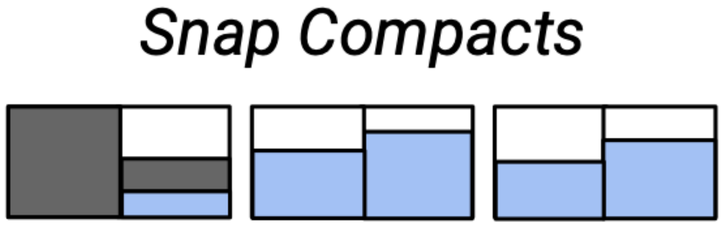
Snap Engine Scheduling

- #3: Compacting Engines
  - Consolidate engines to as fewer cores as possible
  - Periodic polling of queueing delays to re-balance engines to more cores



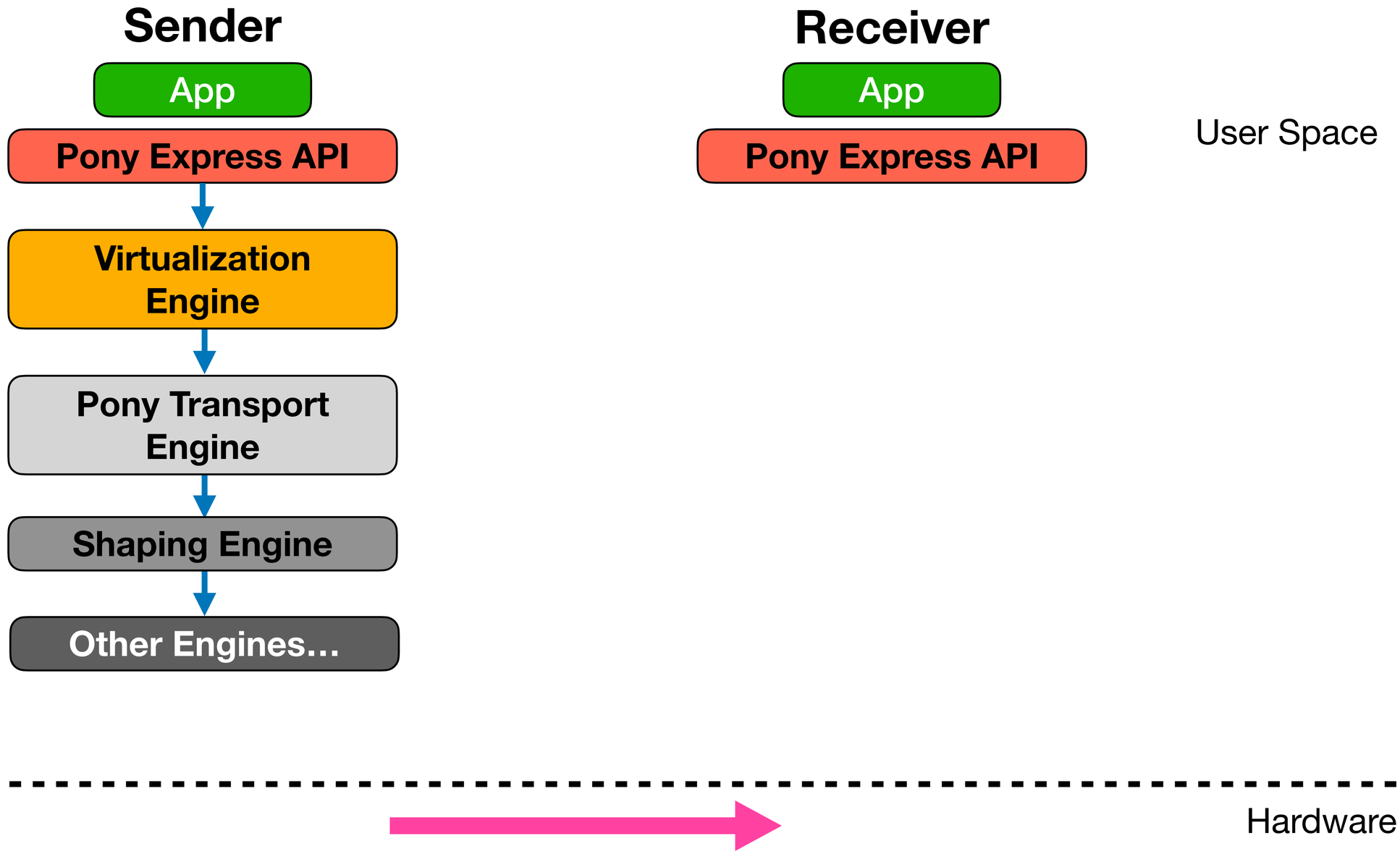
Snap Engine Scheduling

- #3: Compacting Engines
  - Consolidate engines to as fewer cores as possible
  - Periodic polling of queueing delays to re-balance engines to more cores

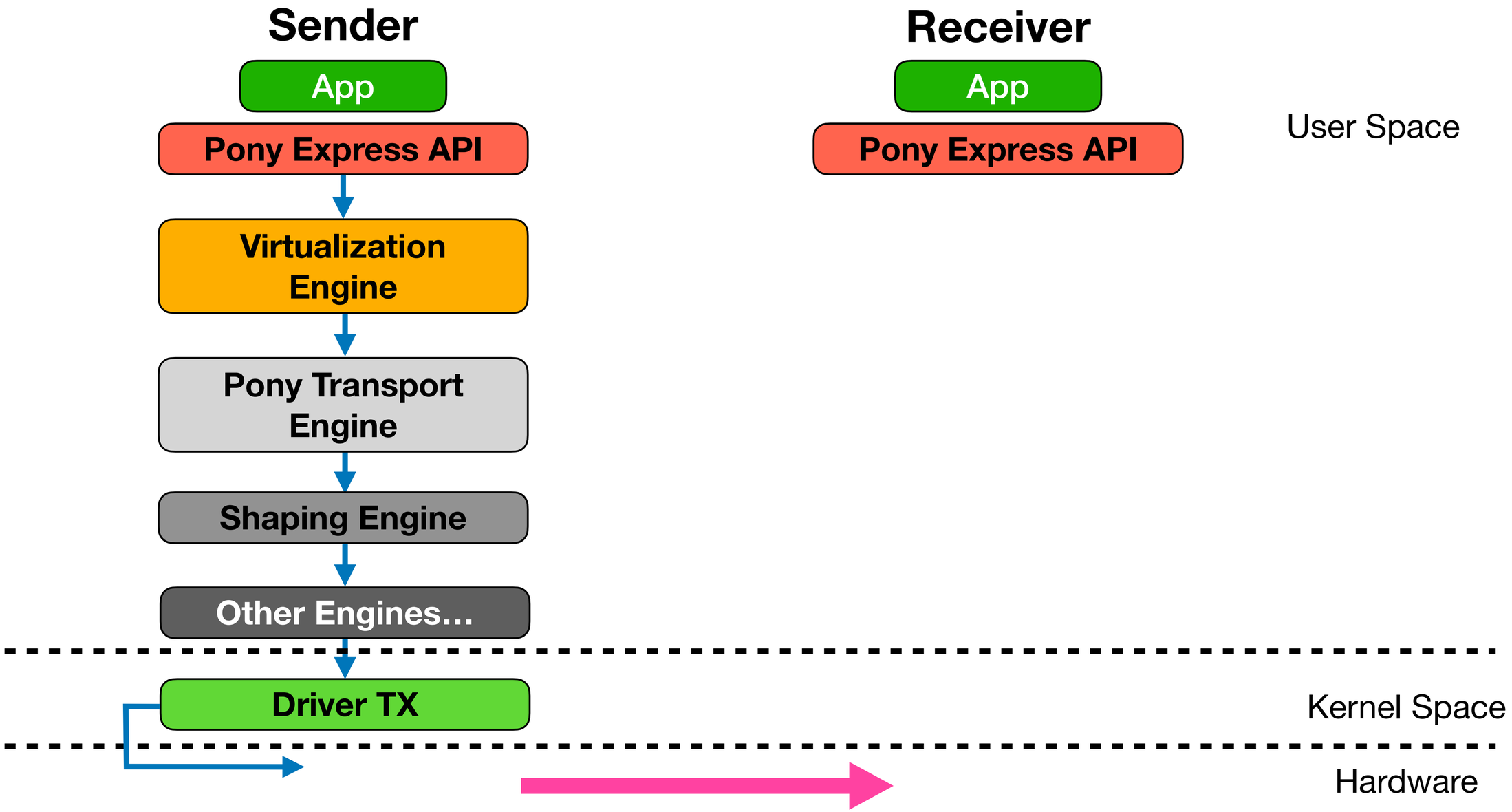


- Drawback
  - Hard to detect queue build-up when many engines

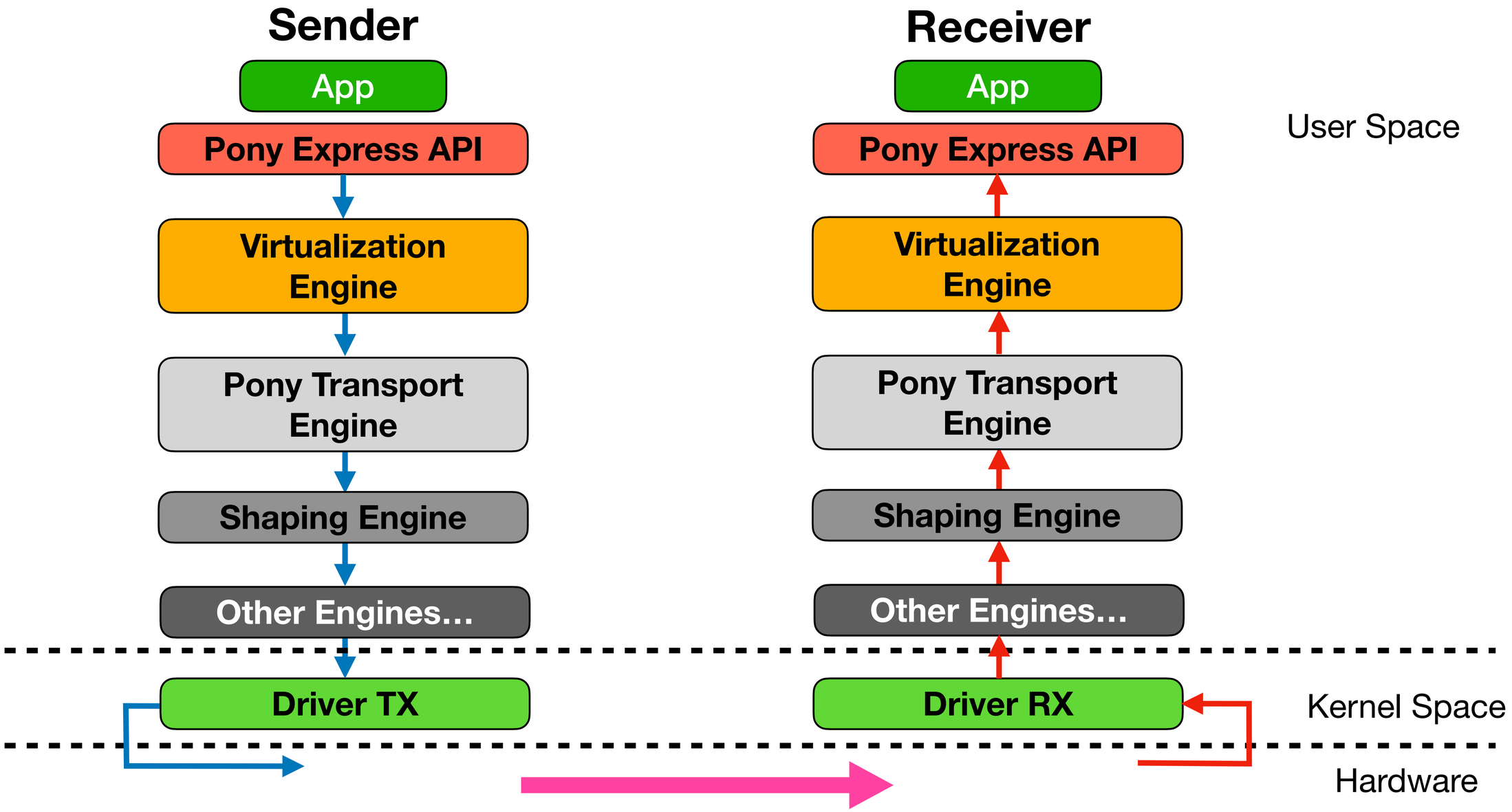
Snap End-to-End Data Path



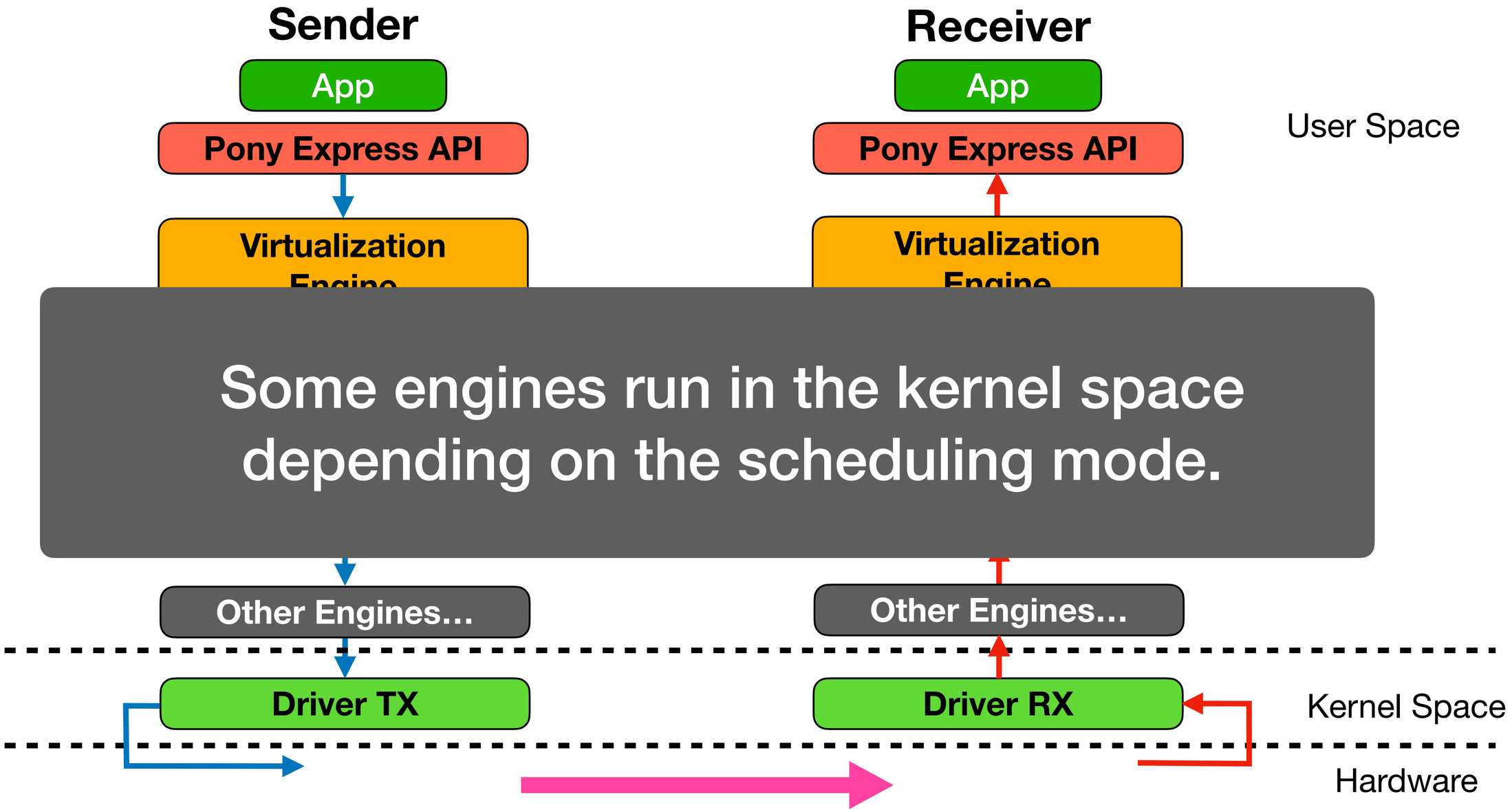
Snap End-to-End Data Path



Snap End-to-End Data Path

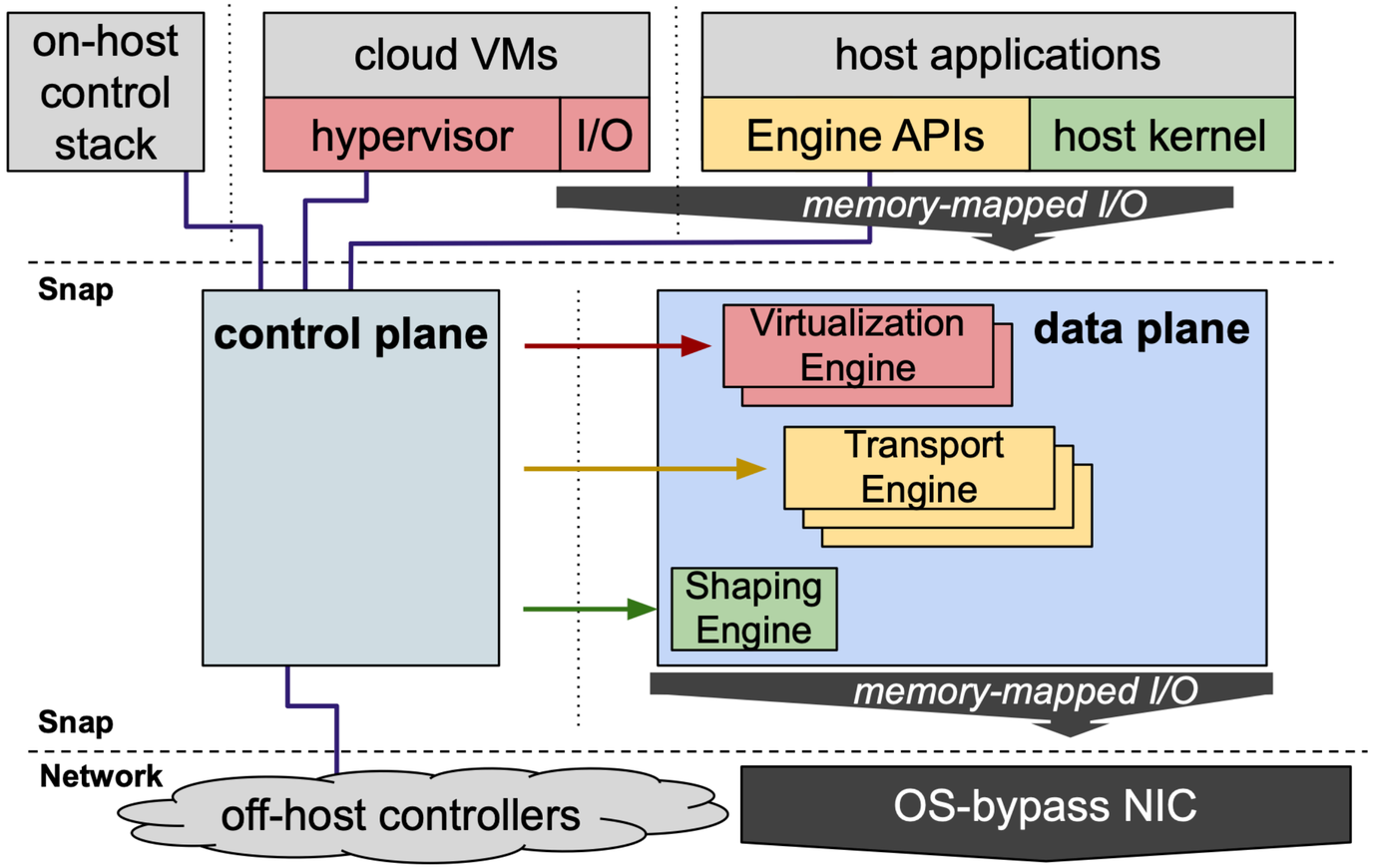


Snap End-to-End Data Path



Snap Architecture

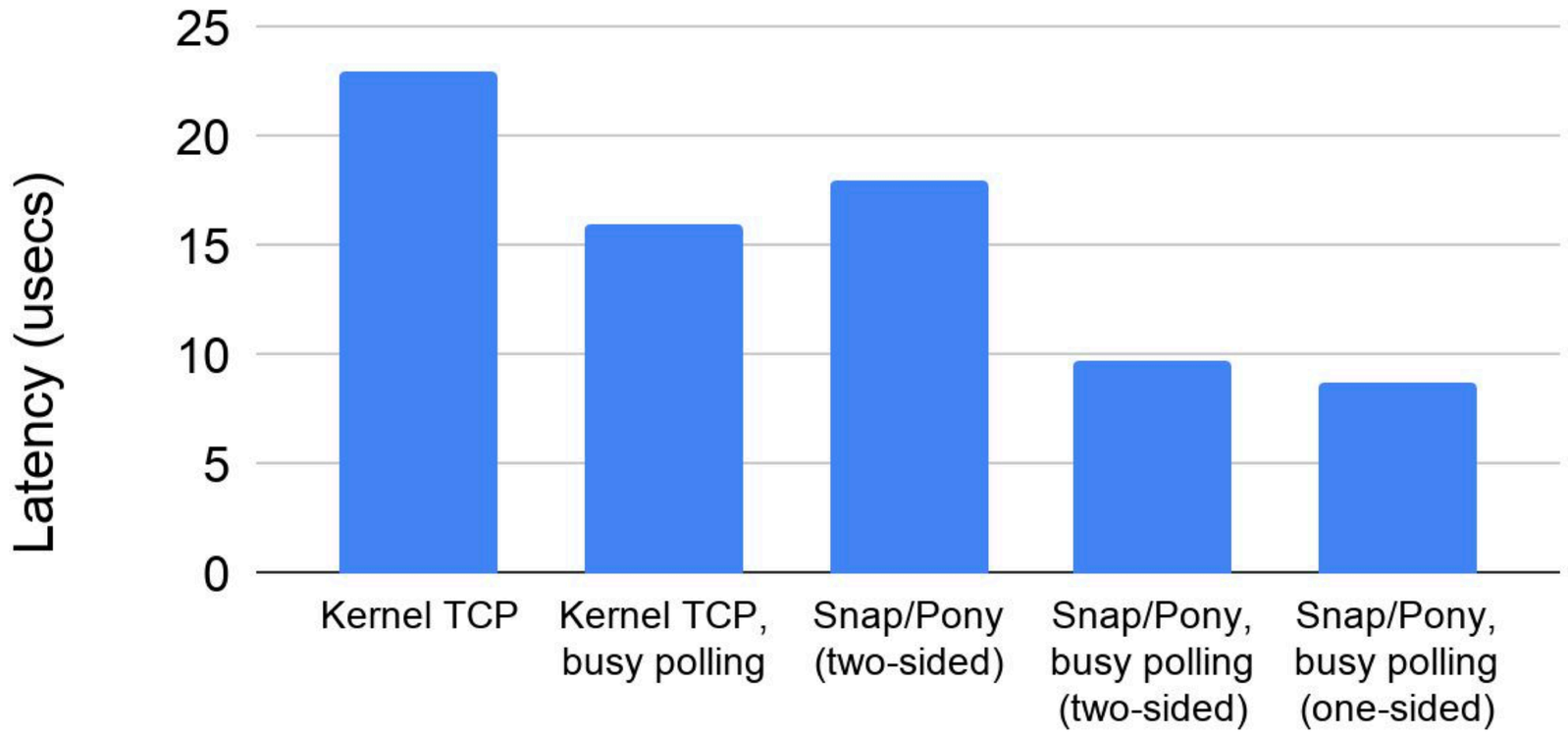
- The control plane sets up the engine pipeline



Why does Snap improve development velocity?

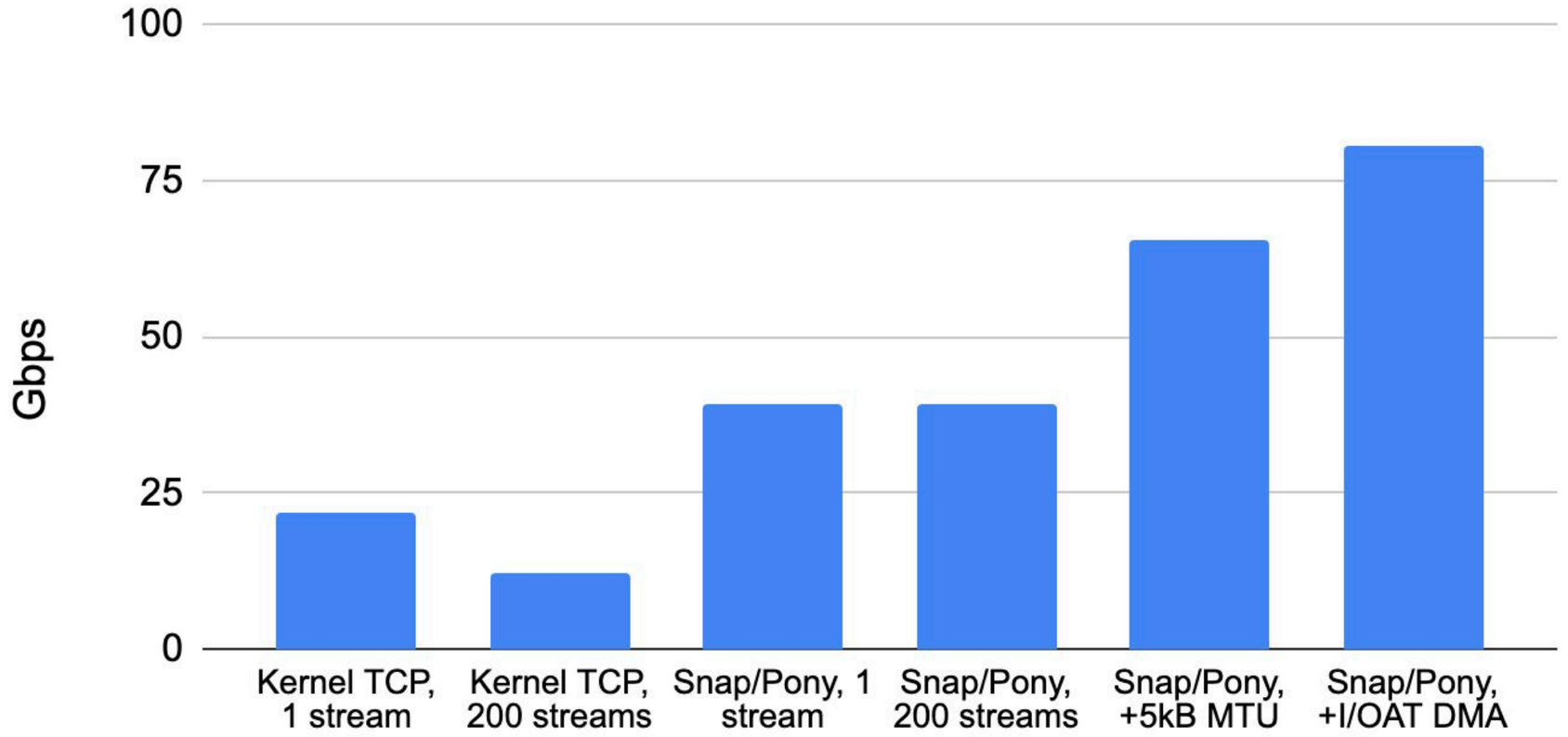
Snap Unloaded Latency

- 2-node ping pong latency



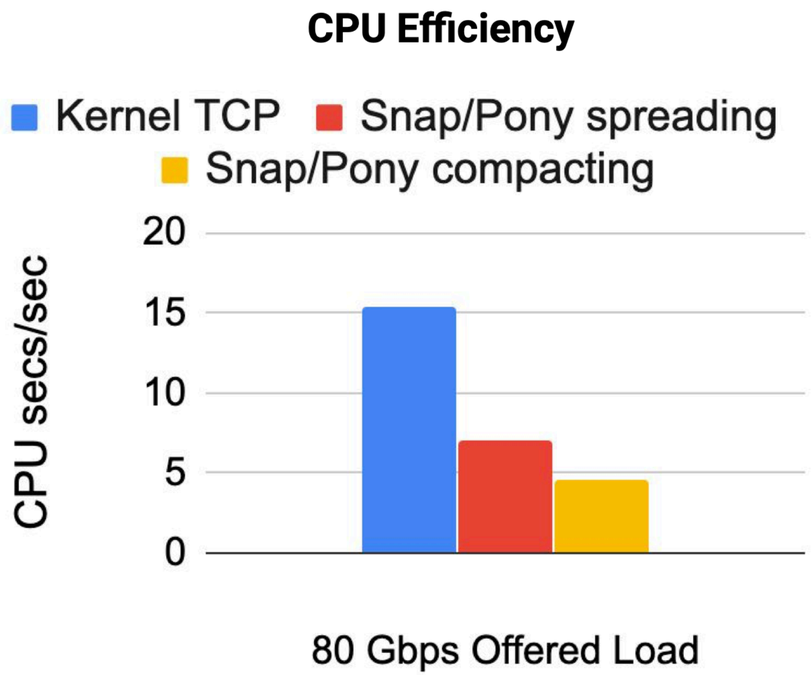
Snap Throughput

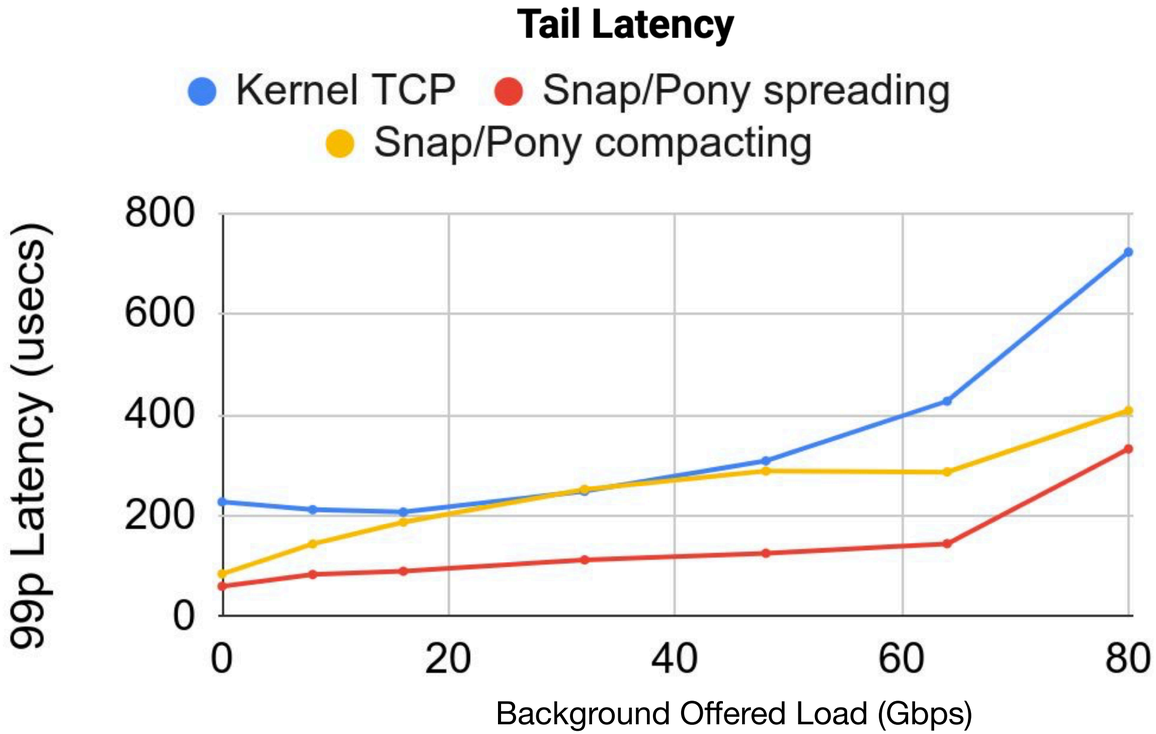
- 2-node throughput
  - Single pony engine + dedicated core



Snap Scaling

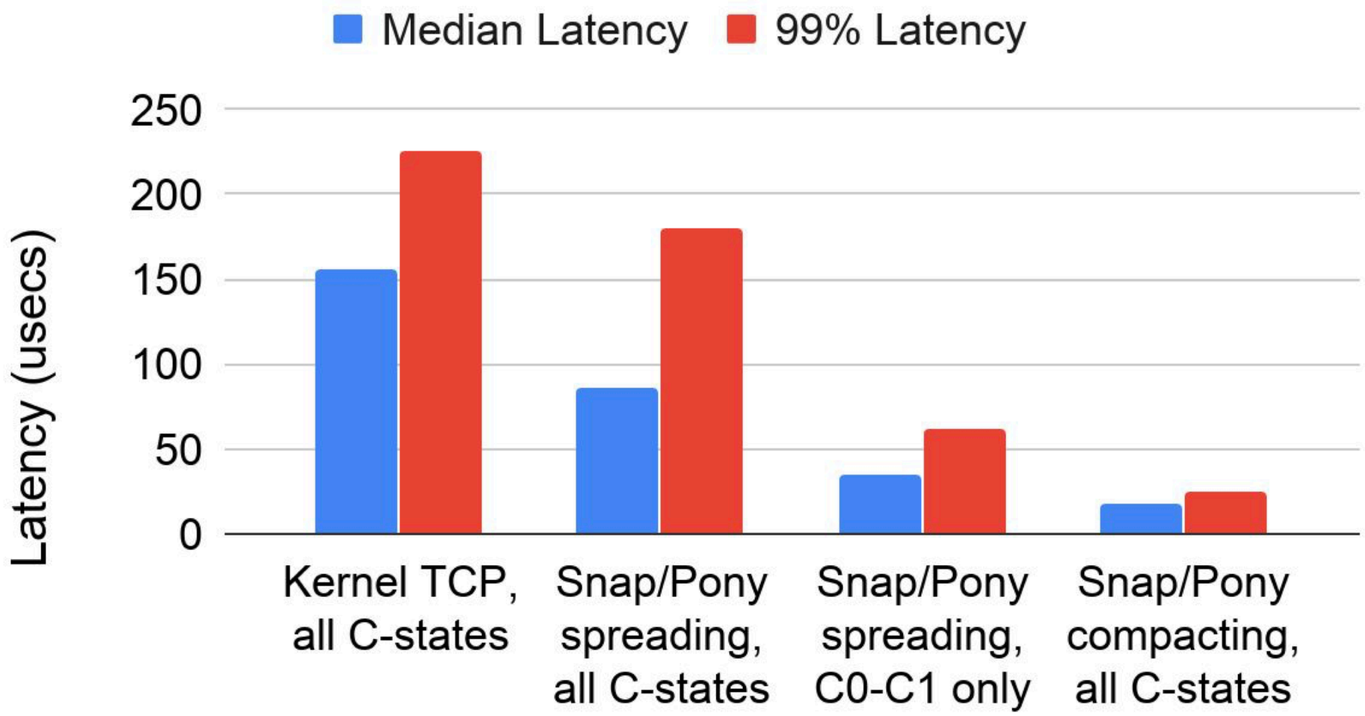
- 10 concurrent Pony express engine





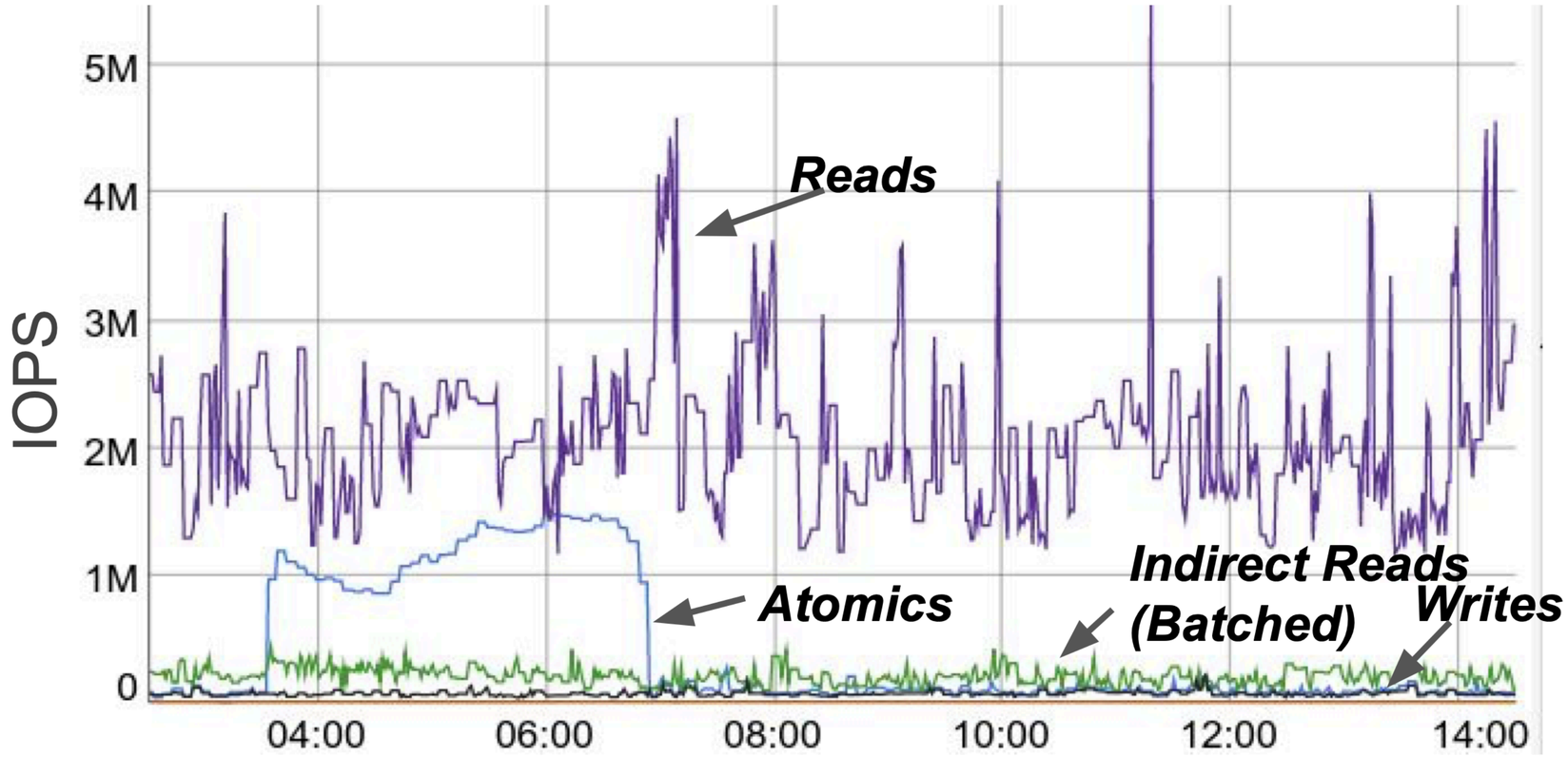
Snap Performance Impact

- C-states and non-preemptible kernel activity
  - Spreading engines



Snap @Production

- Heavy-loaded machine
  - Single pony engine + dedicated core



A Few Words About the Exam

- Goal: solidify your understanding of data center networks

A Few Words About the Exam

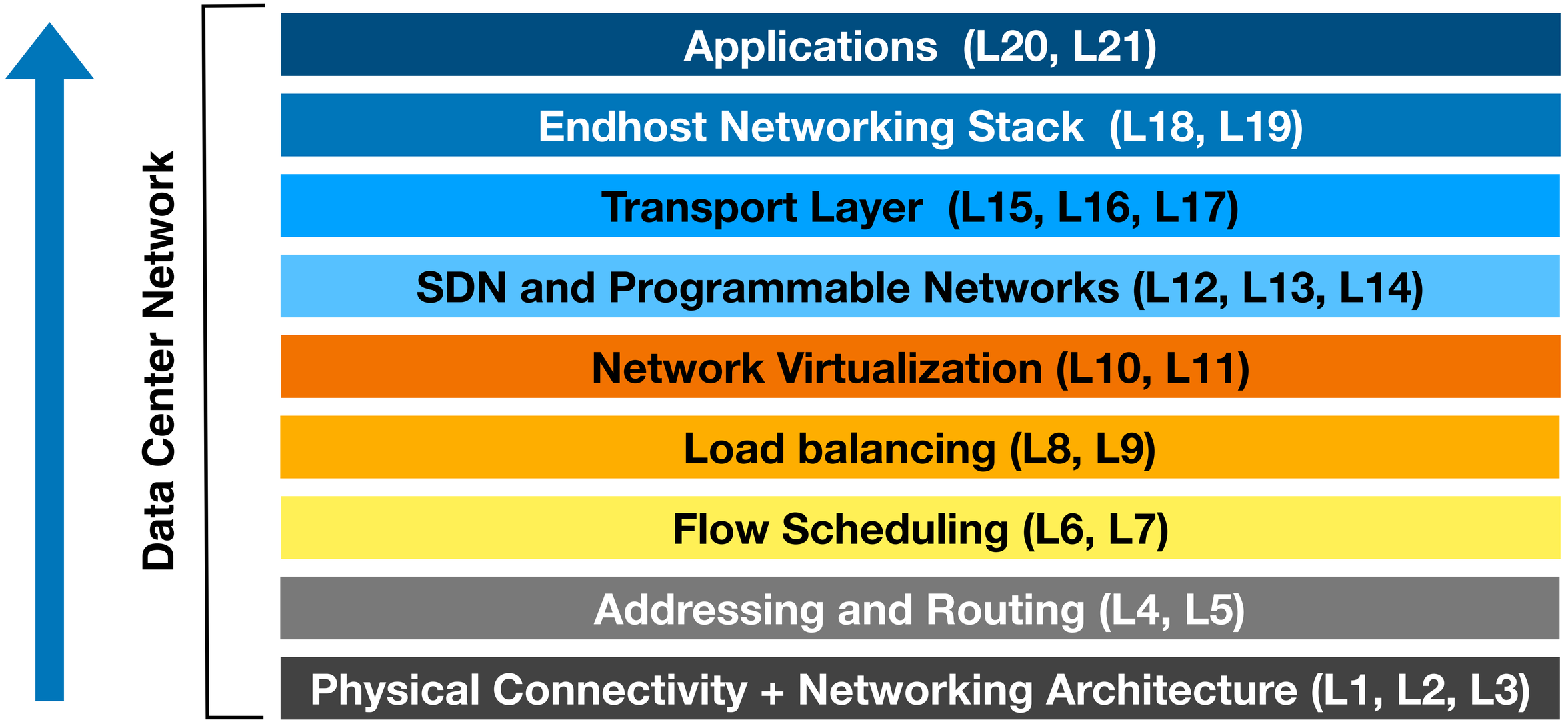
- Goal: solidify your understanding of data center networks

- When and Where
  - 11/19 9:30 am — 11:00 am
  - MH 6618

A Few Words About the Exam

- Goal: solidify your understanding of data center networks
- When and Where
  - 11/19 9:30 am — 11:00 am
  - MH 6618
- Must attend
  - No make-up

Exam Scope


Data Center Network
Applications (L20, L21)
Endhost Networking Stack (L18, L19)
Transport Layer (L15, L16, L17)
SDN and Programmable Networks (L12, L13, L14)
Network Virtualization (L10, L11)
Load balancing (L8, L9)
Flow Scheduling (L6, L7)
Addressing and Routing (L4, L5)
Physical Connectivity + Networking Architecture (L1, L2, L3)

Exam Style

CS 740: Exam, Fall 2024

Please ***write*** your name and student ID below.

Last Name:
First Name:
Student ID:

Instructions:

1. This is an **open-book, open-notes** exam. You can use any material provided in this course or on the Internet or ChatGPT-like AI assistant.
2. You are **not allowed** to discuss the exam questions and solutions with others during the exam.
3. You have **90 mins** to complete this exam.
4 Please choose **4 questions out of 5**.
5. Make sure your answers are **precise**, **complete**, and **clear**.

(For Instructors Use)

Q1 (25 points): Networking Architecture, Addressing, Routing	
Q2 (25 points): Flow Scheduling and Load Balancing	
Q3 (25 points): SDN and Programmable Networks	
Q4 (25 points): Congestion Control	
Q5 (25 points): Network Virtualization and Endhost NStack	
Total (100 points)	

Exam Style

CS 740: Exam, Fall 2024

Please ***write*** your name and student ID below.

Last Name:
First Name:
Student ID:

Instructions:

1. This is an **open-book, open-notes** exam. You can use any material provided in this course or on the Internet or ChatGPT-like AI assistant.
2. You are **not allowed** to discuss the exam questions and solutions with others during the exam.
3. You have **90 mins** to complete this exam.
4 Please choose **4 questions out of 5**.
5. Make sure your answers are **precise**, **complete**, and **clear**.

(For Instructors Use)

Q1 (25 points): Networking Architecture, Addressing, Routing	
Q2 (25 points): Flow Scheduling and Load Balancing	
Q3 (25 points): SDN and Programmable Networks	
Q4 (25 points): Congestion Control	
Q5 (25 points): Network Virtualization and Endhost NStack	
Total (100 points)	

- Just pick 4 questions
- If you answer 5, we will grade based on the top 4
- If the question is unclear, write your assumptions first

A Question Example

- Each question will have 3~4 sub-questions

Question 3. [30 points] Resource Sharing

Efficient resource sharing is important to any multi-tenant computing systems. It ensures performance isolation and mitigates execution interference.

(a). [**15 points**] DRF generalizes the max-min fairness to multiple resource types. Considering an ideal environment with the following configurations:

- Resource pool <20 CPUs, 24GB Memory, 13 GPUs>
- User A's request demand vector <1 CPUs, 4GB, 1 GPU>
- User B's request demand vector<4 CPUs, 1GB, 1 GPUs>
- User C's request demand vector<1 CPUs, 1GB, 4 GPUs>

Assume (1) each user has an unlimited amount of incoming tasks; (2) the allocation starts from scratch; (3) the scheduler performs the first two allocations to users A and B. Please fill up the following scheduling table based on the DRF algorithm.

Schedule	User A's Dom. Share	User B's Dom. Share	User C's Dom. Share	CPU Total Allocation	RAM Total Allocation	GPU Total Allocation
User A						
User B						

Summary

- Today
  - SNAP (SOSP'19)

- Next Topic: Data Center Applications
  - Atlas (SIGCOMM'17)
  - QUIC (SIGCOMM'17)