



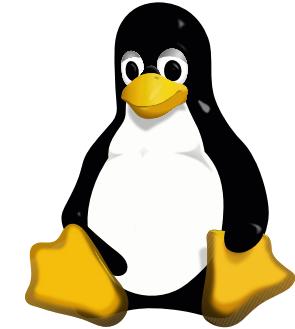
# Joyride: Rethinking Linux's Network Stack Design

Yanlin Du, Ruslan Nikolaev  
The Pennsylvania State University  
*KISV '25, Seoul, Republic of Korea*



PennState

# The Growing Network Performance Gap



**200-800 Gbps NICs**



PennState

# The Growing Network Performance Gap

Introducing  
The Broadcom  
400GbE RDMA NIC

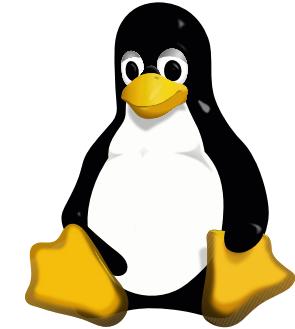
## Introducing Broadcom 400G NIC: AI Optimized NIC

- ✓ 400G high-performance NIC
- ✓ High-scale RDMA
- ✓ Industry's lowest power
- ✓ Longest reach 100G Serdes



5 | Broadcom Proprietary and Confidential. Copyright © 2024 Broadcom. All Rights Reserved. The term "Broadcom" refers to Broadcom Inc. and/or its subsidiaries.

 BROADCOM™



## 200-800 Gbps NICs



PennState

# The Growing Network Performance Gap

Introducing  
The Broadcom  
400GbE RDMA NIC

## Introducing Broadcom 400G NIC: AI Optimized NIC

- ✓ 400G high-performance NIC
- ✓ High-scale RDMA
- ✓ Industry's lowest power
- ✓ Longest reach 100G Serdes



## Intel® Ethernet E830 Controllers and Network Adapters

Performance networking for  
virtualization, cloud, telecom, and edge



### 200GbE Throughput at Higher Efficiency

2x bandwidth gen-on-gen and improved performance per watt

### Versatile Port Configurations

Enabled by broad port density, for efficient resource scaling

### Accurate & Advanced Timing Capabilities

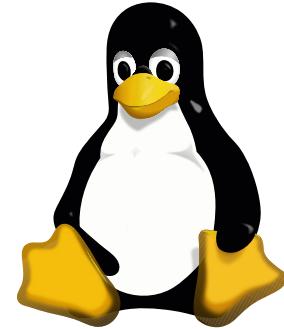
PTM with 1588 PTP, SyncE, GNSS for Telecom, FSI, AI applications

### Robust Security

Secure Boot, Secure FW Upgrade and Dual Hardware Root of Trust

 BROADCOM™

Intel E830 Launch Slide



## 200-800 Gbps NICs

# The Growing Network Performance Gap

Introducing  
The Broadcom  
400GbE RDMA NIC

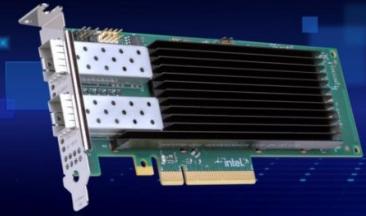
## Introducing Broadcom 400G NIC: AI Optimized NIC

- ✓ 400G high-performance NIC
- ✓ High-scale RDMA
- ✓ Industry's lowest power
- ✓ Longest reach 100G Serdes



## Intel® Ethernet E830 Controllers and Network Adapters

Performance networking for  
virtualization, cloud, telecom, and edge



### 200GbE Throughput at Higher Efficiency

2x bandwidth gen-on-gen and improved performance per watt

### Versatile Port Configurations

Enabled by broad port density, for efficient resource scaling

### Accurate & Advanced Timing Capabilities

PTM with 1588 PTP, SyncE, GNSS for Telecom, FSI, AI applications

### Robust Security

Secure Boot, Secure FW Upgrade and Dual Hardware Root of Trust

BROADCOM®

Intel E830 Launch Slide



## 200-800 Gbps NICs

ConnectX-8 Isn't Just Another NIC – It's a SuperNIC!



ConnectX-8 800G SuperNIC  
Redefining networking for AI factory

RDMA technology deployed over millions of GPUs

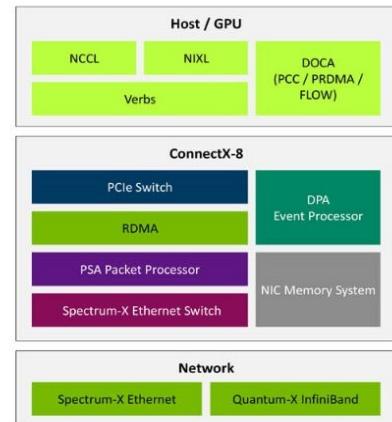
800G RDMA hardware pipeline, designed for AI bandwidth, latency and scale

Integrated load balancing, congestion control and reliability

Deep data-path programmability for AI workload and data center versatility

Tight integration to system architecture

Enterprise-class security



NVIDIA ConnectX 8 At Hot Chips 2025 Page 03



PennState

# The Growing Network Performance Gap



PennState

# The Growing Network Performance Gap

## **The Challenge:**

Network hardware

10 Gbps → 25 Gbps → 50 Gbps → 100 Gbps → 200 Gbps → 400 Gbps → 800 Gbps



PennState

# The Growing Network Performance Gap

## **The Challenge:**

Network hardware

10 Gbps → 25 Gbps → 50 Gbps → 100 Gbps → 200 Gbps → 400 Gbps → 800 Gbps

Linux's network stack is not keeping up with this pace



# The Growing Network Performance Gap

## **The Challenge:**

Network hardware

10 Gbps → 25 Gbps → 50 Gbps → 100 Gbps → 200 Gbps → 400 Gbps → 800 Gbps

Linux's network stack is not keeping up with this pace

Why cannot Linux keep up?



PennState

# The Growing Network Performance Gap

## The Challenge:

Network hardware

10 Gbps → 25 Gbps → 50 Gbps → 100 Gbps → 200 Gbps → 400 Gbps → 800 Gbps

Linux's network stack is not keeping up with this pace

## Why cannot Linux keep up?

At 100 Gbps with 1500-byte MTU

- 8+ million packets per second



# The Growing Network Performance Gap

## The Challenge:

Network hardware

10 Gbps → 25 Gbps → 50 Gbps → 100 Gbps → 200 Gbps → 400 Gbps → 800 Gbps

Linux's network stack is not keeping up with this pace

## Why cannot Linux keep up?

At 100 Gbps with 1500-byte MTU

- 8+ million packets per second
- Packets trigger: interrupt → mode switch → kernel processing → memory copy



# The Growing Network Performance Gap

## The Challenge:

Network hardware

10 Gbps → 25 Gbps → 50 Gbps → 100 Gbps → 200 Gbps → 400 Gbps → 800 Gbps

Linux's network stack is not keeping up with this pace

## Why cannot Linux keep up?

At 100 Gbps with 1500-byte MTU

- 8+ million packets per second
- Packets trigger: interrupt → mode switch → kernel processing → memory copy
- Result: 4-8 cores needed for 100 Gbps



# The Growing Network Performance Gap

## The Challenge:

Network hardware

10 Gbps → 25 Gbps → 50 Gbps → 100 Gbps → 200 Gbps → 400 Gbps → 800 Gbps

Linux's network stack is not keeping up with this pace

## Why cannot Linux keep up?

At 100 Gbps with 1500-byte MTU

- 8+ million packets per second
- Packets trigger: interrupt → mode switch → kernel processing → memory copy
- Result: 4-8 cores needed for 100 Gbps

**Kernel processing, not hardware, is now the bottleneck**



# Limitations of Existing Solutions

## **Kernel Improvements:**

BIG TCP => Bigger TCP window => Enable larger segments, fewer packets to process.

Generic Receive Offload (GRO) / Generic Segmentation Offload (GSO) => Packet coalescing to reduce per-packet overhead

Performance gains are incremental

## **Kernel-Bypass Solutions:**

IX [OSDI'14], TAS [EuroSys'19], Junction [NSDI'24], F-stack, among others

## **The Fundamental Gap:**

No existing system provides a transparent kernel bypass for unmodified applications on standard Linux



# Kernel Bypass: IX [OSDI'14]

- An independent TCP implementation that uses DPDK code partially
- **Pros:** Scalable performance, reducing lock overheads
- **Cons:**
  - No POSIX compatibility, custom interfaces
  - No support of numerous TCP extensions
  - Packet delivery reliability issues
  - Not being actively developed, not tested with modern NICs



# Kernel Bypass: TAS [EuroSys'19]

- A more recent effort that runs on top of DPDK
- **Pros:**
  - Scalable and CPU efficient stack
  - TAS tries to optimize performance for datacenter applications which helps to reduce costs => faster than IX
- **Cons:**
  - No POSIX compatibility
  - Like IX, does not provide full TCP stack that can replace Linux's TCP
  - Not being updated recently, uses an older DPDK version
  - Data-center specific assumptions: no IP fragmentation, reliable in order delivery, rare timeouts



# Kernel Bypass: RDMA

- Enables direct memory-to-memory communication between machines, bypassing the CPU and kernel
- **Pros:**
  - Very low-latency and high-throughput communication
  - Zero-copy data transfer, reducing CPU overhead
  - Offloads tasks to specialized NICs
- **Cons:**
  - Specialized NIC is required => not a suitable general-purpose replacement
  - No TCP/POSIX support



# Kernel Bypass: Junction [NSDI'24]

- A datacenter OS with a specific focus on Mellanox/NVIDIA NICs
- **Pros:**
  - Very good performance and scalability
  - Unmodified applications (mostly)
- **Cons:**
  - Cannot replace Linux's standard stack
  - Great performance but unclear how that works outside of Mellanox/NVIDIA NICs
  - These NICs still typically depend on a kernel-level driver



# Kernel Bypass: F-Stack

- Uses FreeBSD code to build TCP on top of DPDK
- **Pros:**
  - Good-quality code due to BSD-based TCP implementation
  - Support various TCP extensions
  - POSIX compatibility (but only partial due to lack of multithreading)
- **Cons:**
  - No support for multithreading
  - Limited LibC integration
  - No NIC sharing across different programs



# Joyride's Vision

## A Microkernel-Inspired Network Architecture

- User-Space TCP/IP Processing

FreeBSD's mature stack + kernel-bypass (DPDK) performance

- Separate TCP stack instance for each application

- Transparent Application Support

Modified LibC will integrate network-syscall replacements

- System-Wide Deployment

One service for all apps, no per-application network configuration



# Joyride's Architecture



PennState

# Joyride's Architecture

- App 1  
High-performance network path via Joyride



PennState

# Joyride's Architecture

- App 1  
High-performance network path via Joyride

Joyride Path

User  
space

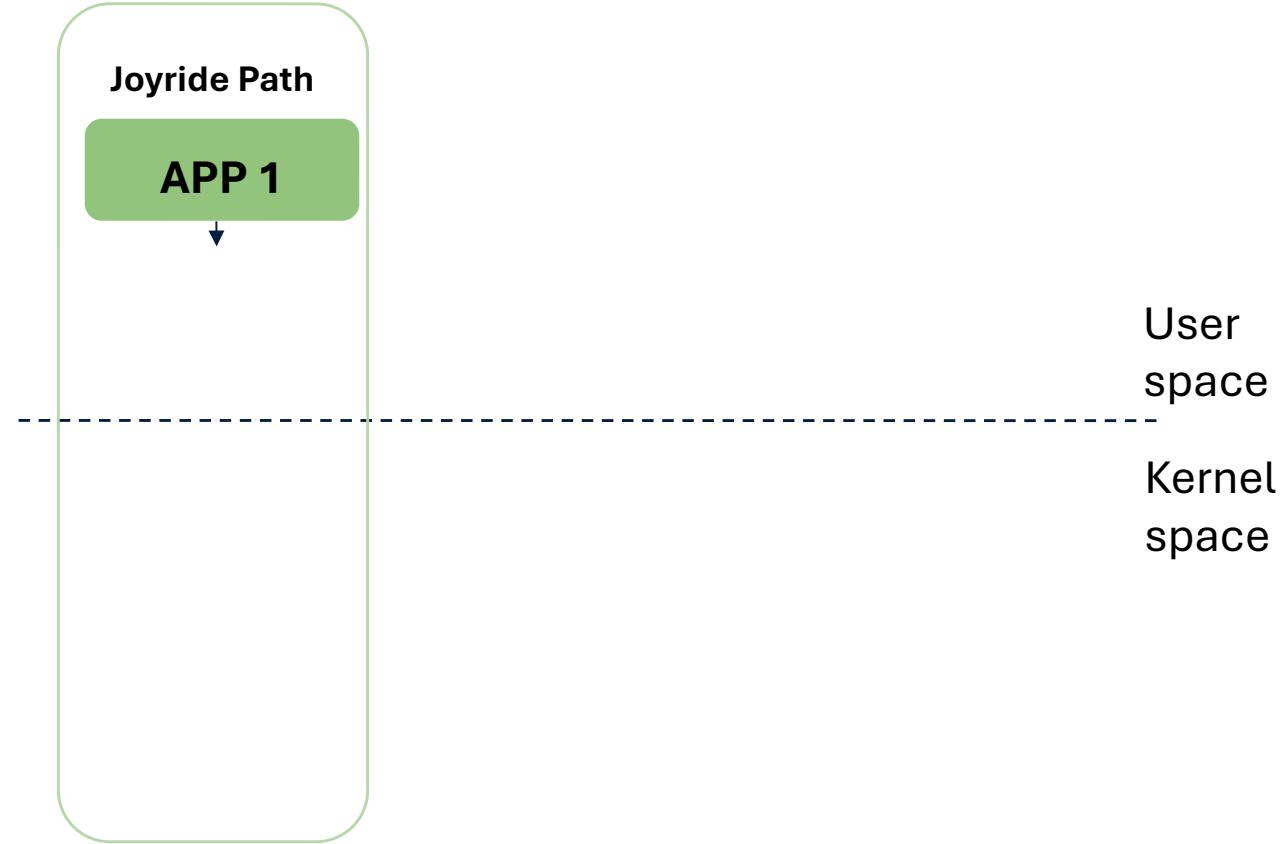
Kernel  
space



PennState

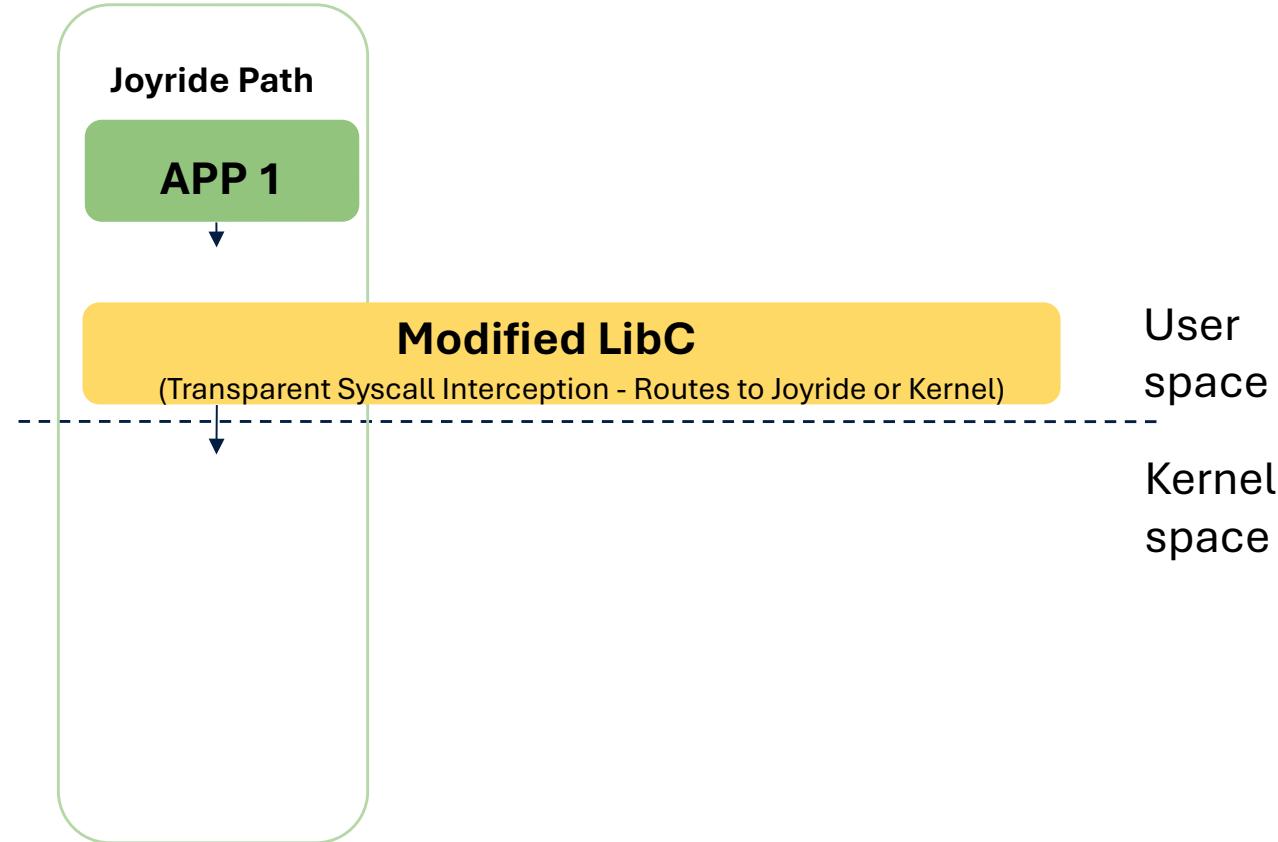
# Joyride's Architecture

- App 1  
High-performance network path via Joyride



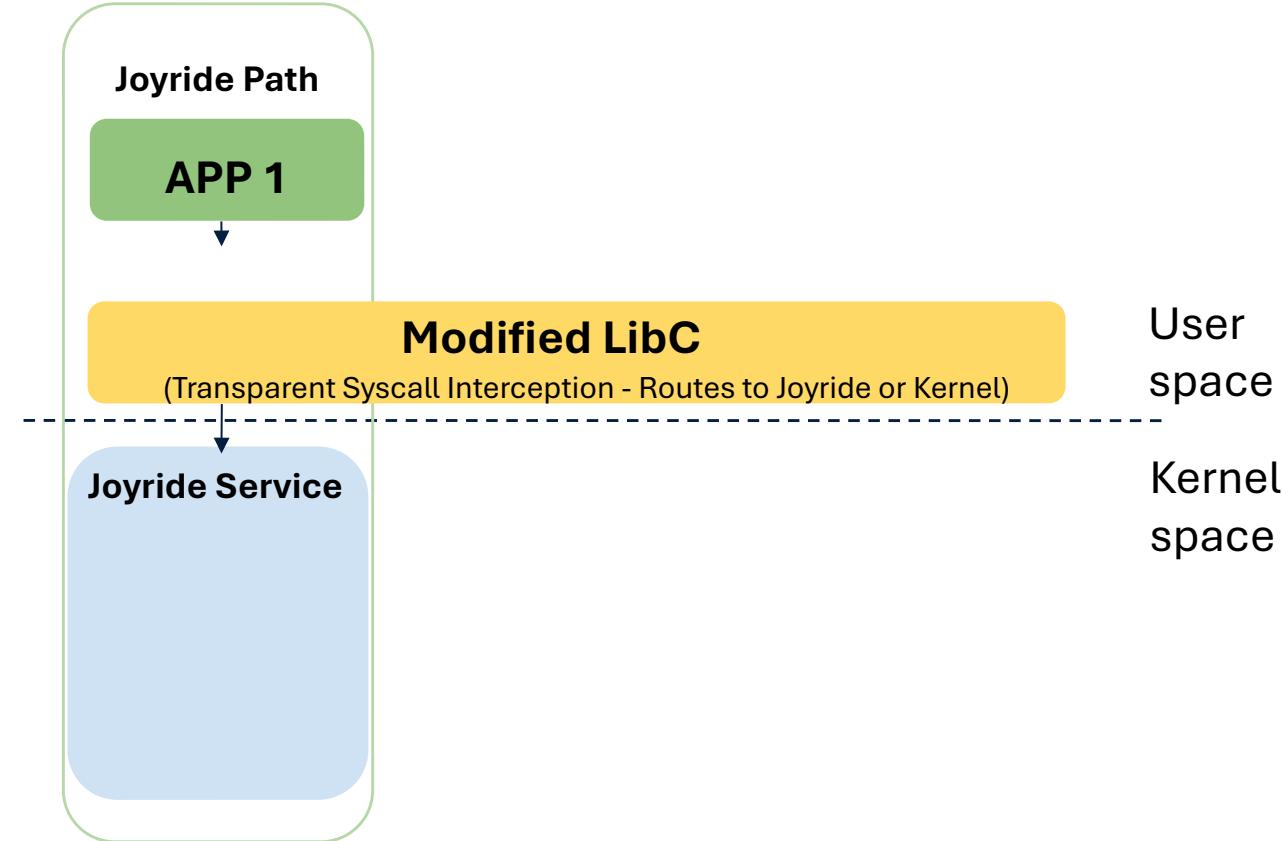
# Joyride's Architecture

- App 1  
High-performance network path via Joyride



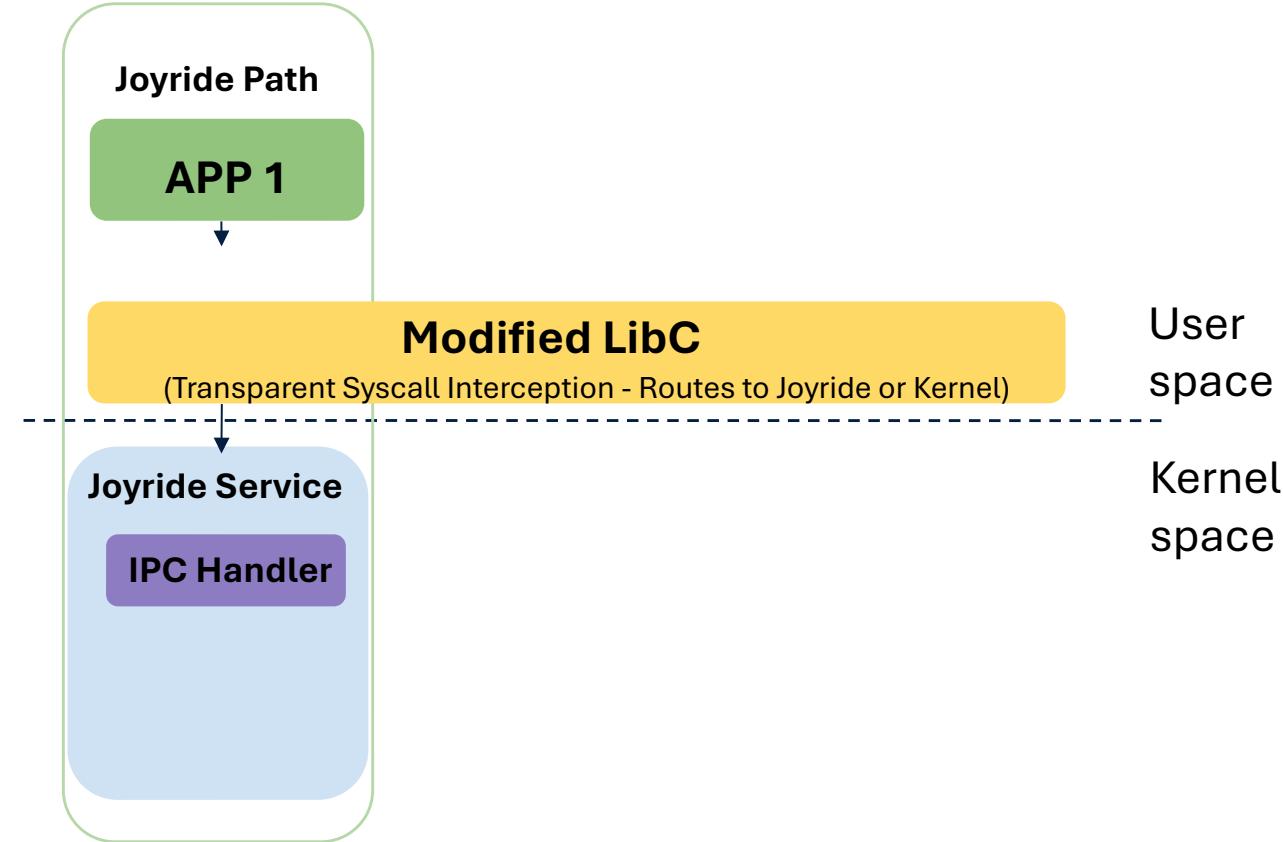
# Joyride's Architecture

- App 1  
High-performance network path via Joyride



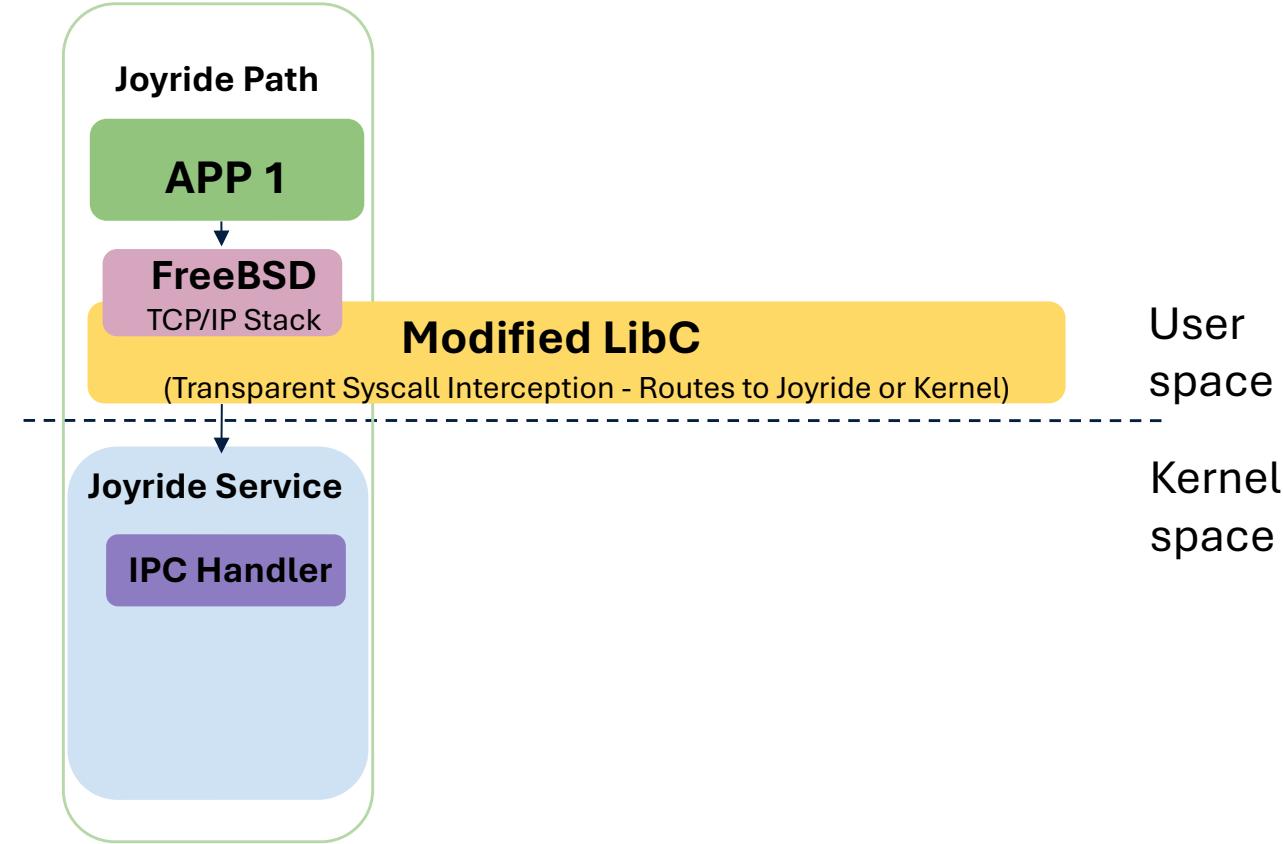
# Joyride's Architecture

- App 1  
High-performance network path via Joyride



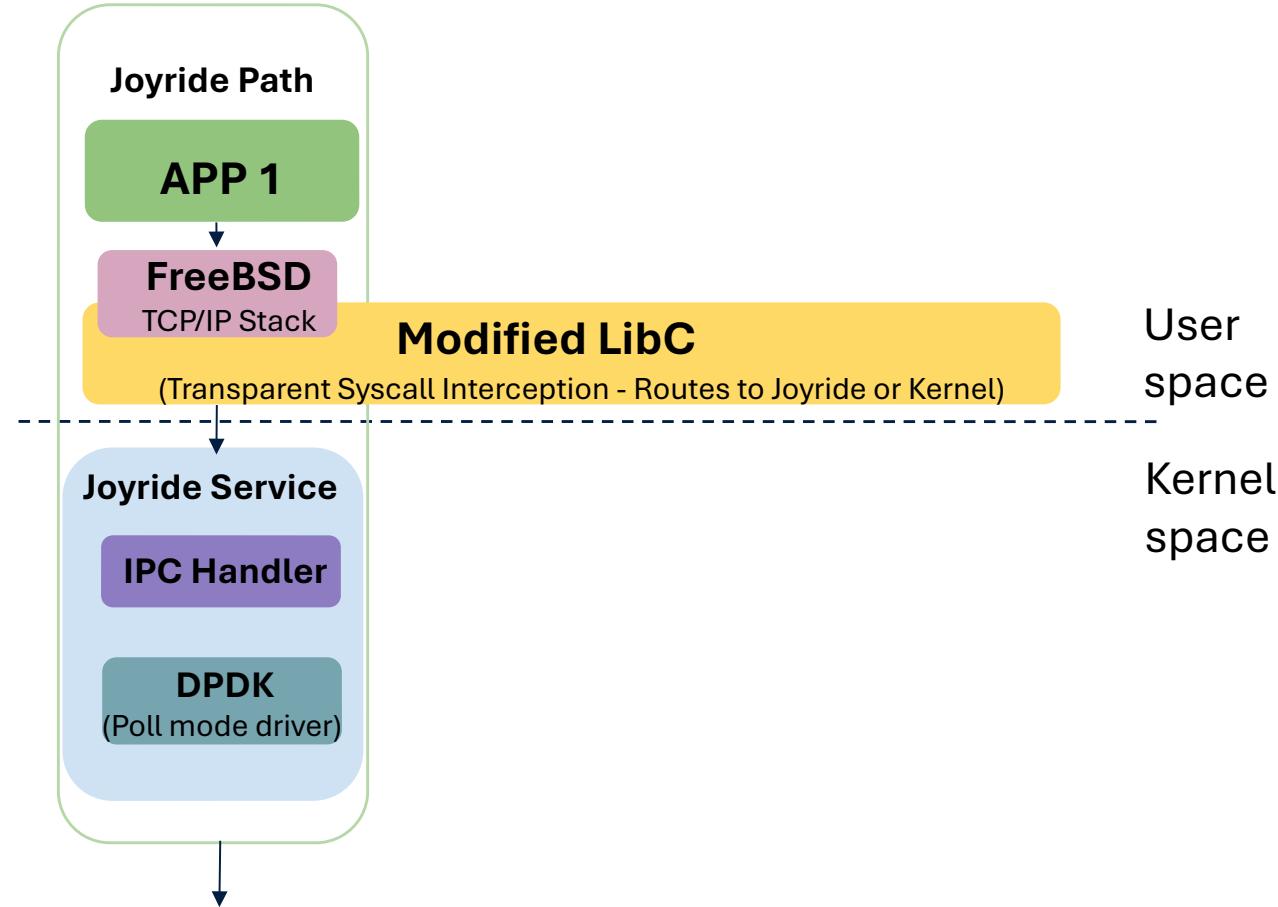
# Joyride's Architecture

- App 1  
High-performance network path via Joyride



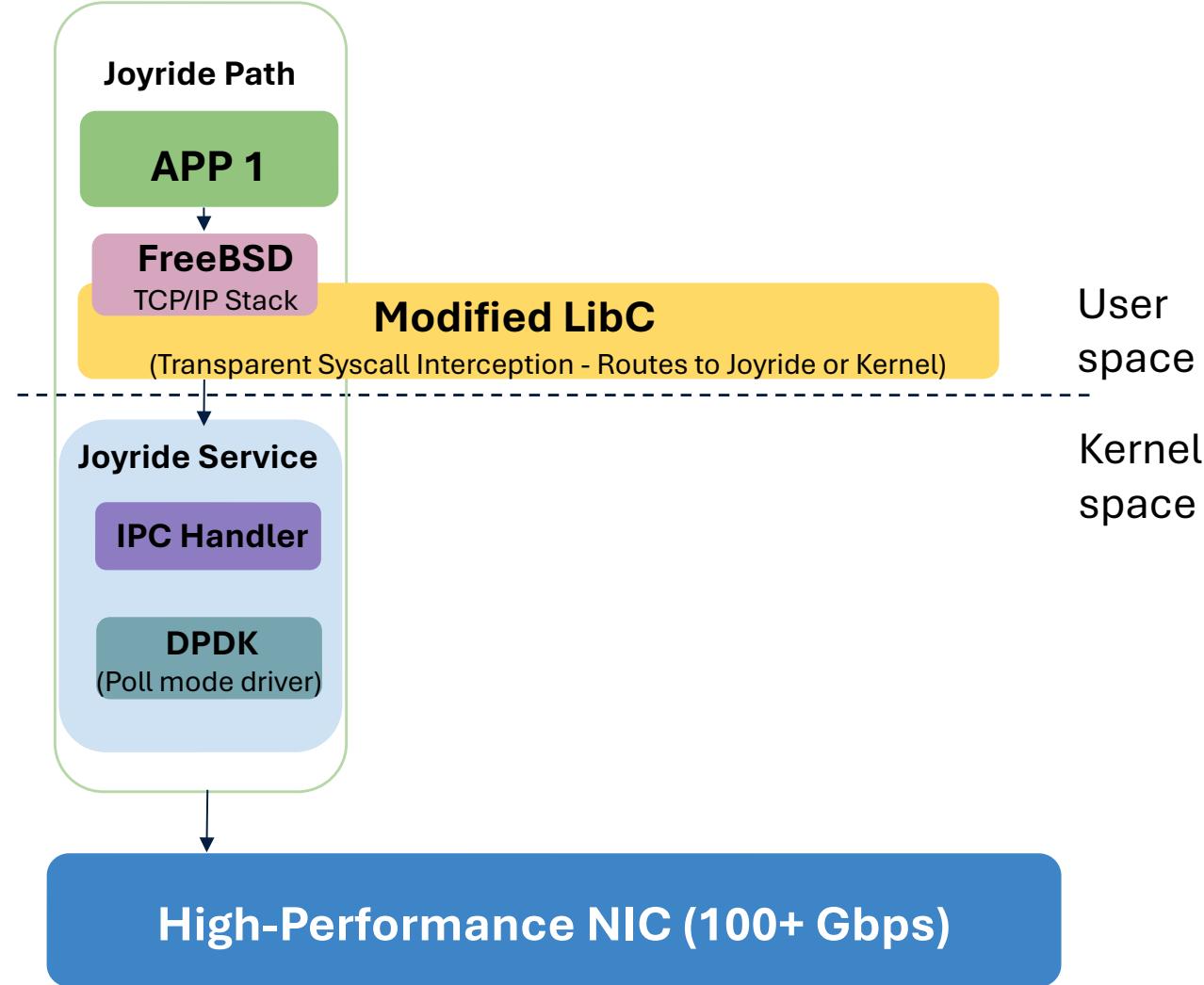
# Joyride's Architecture

- App 1  
High-performance network path via Joyride



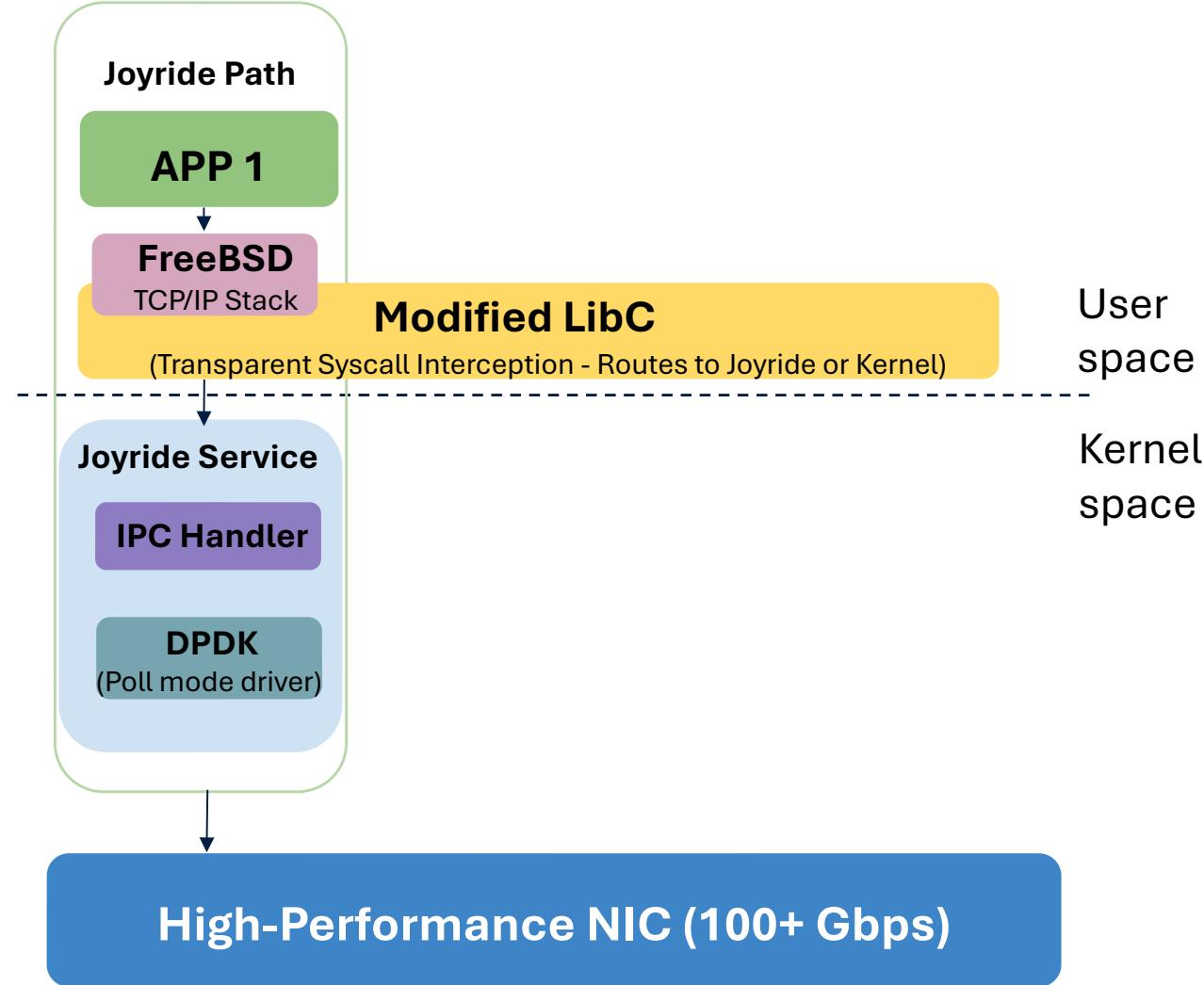
# Joyride's Architecture

- App 1  
High-performance network path via Joyride



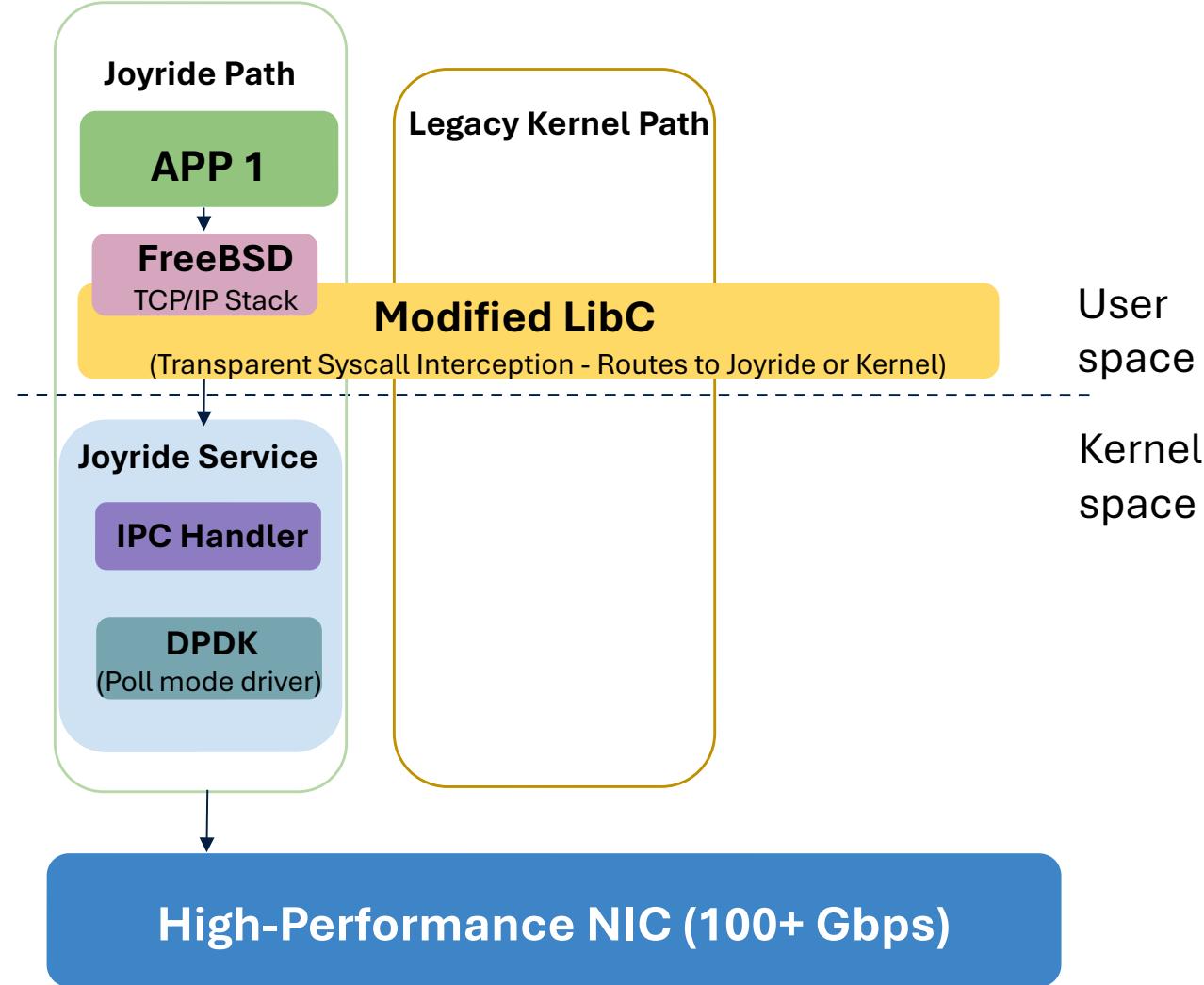
# Joyride's Architecture

- App 1  
High-performance network path via Joyride
- App 2  
Corner-case applications: fall back to the traditional TCP/IP stack



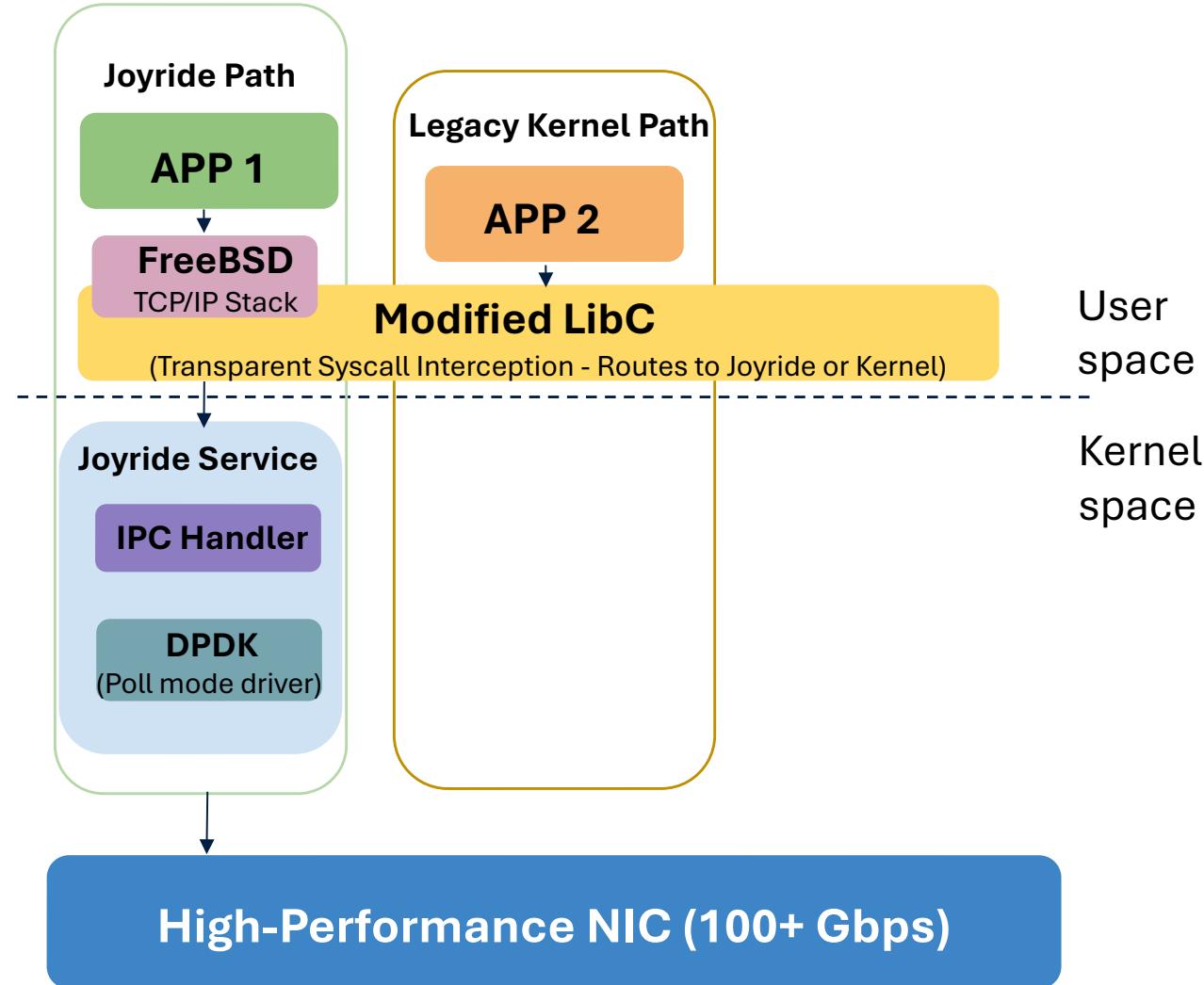
# Joyride's Architecture

- App 1  
High-performance network path via Joyride
- App 2  
Corner-case applications: fall back to the traditional TCP/IP stack



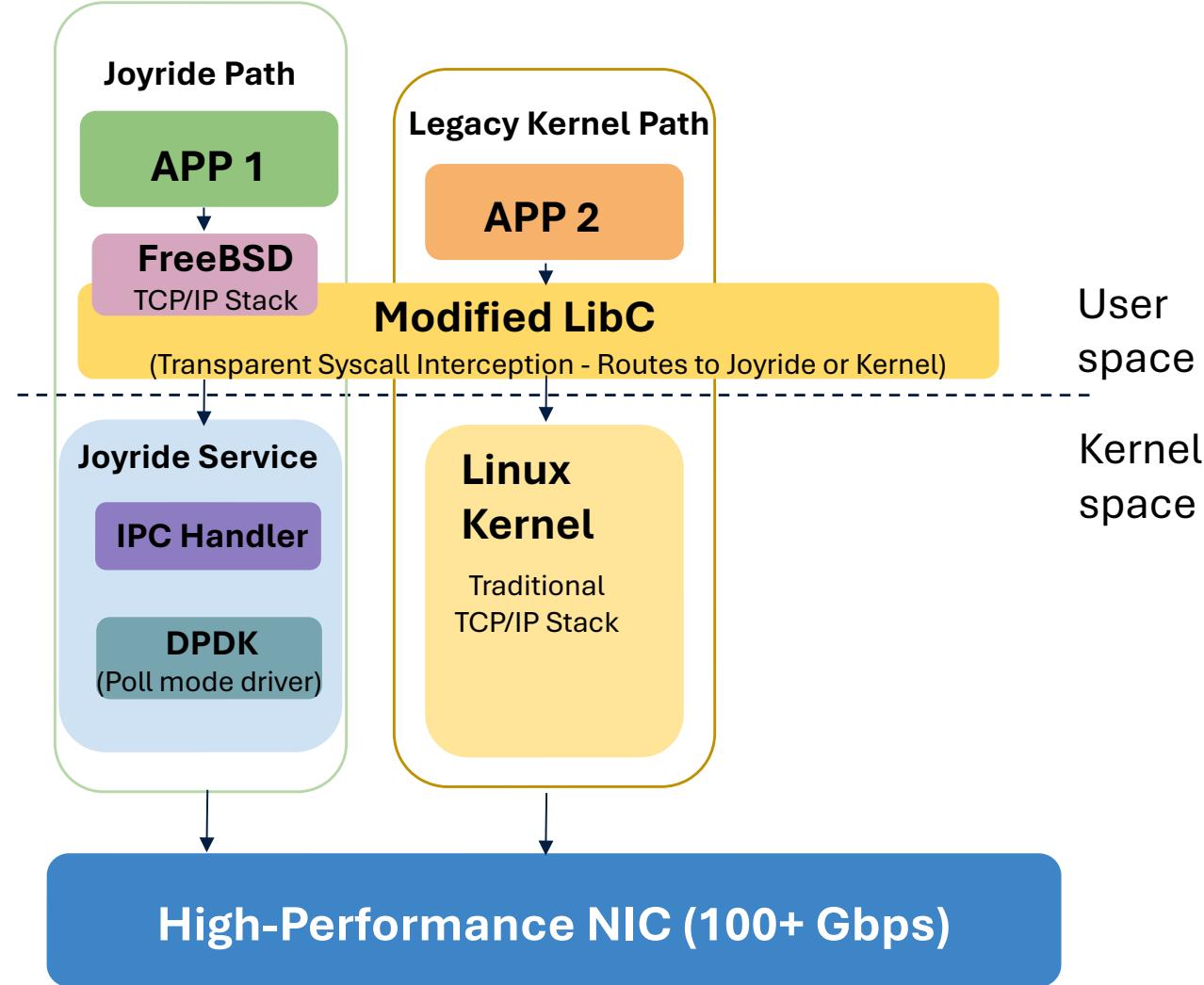
# Joyride's Architecture

- App 1  
High-performance network path via Joyride
- App 2  
Corner-case applications: fall back to the traditional TCP/IP stack



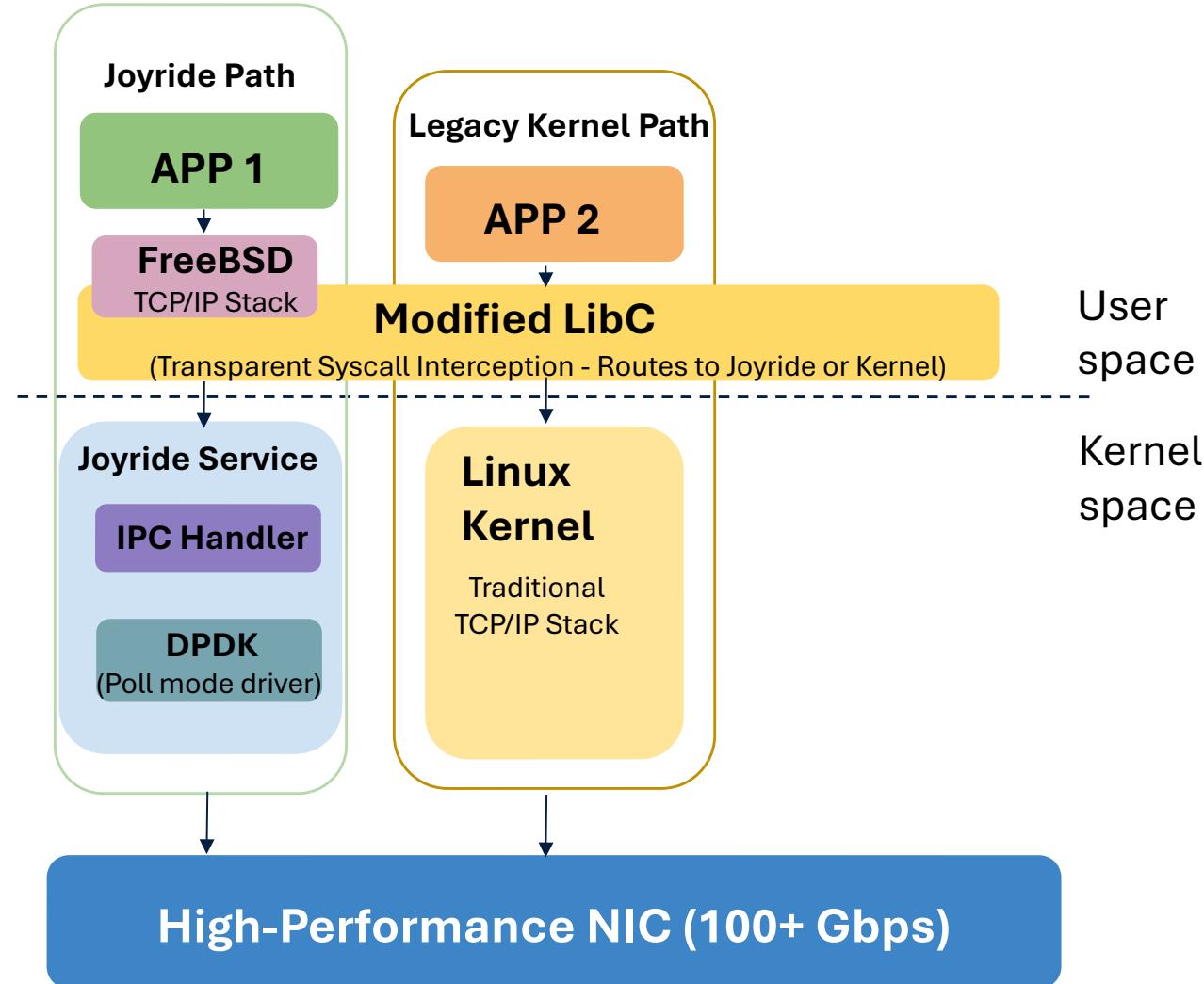
# Joyride's Architecture

- App 1  
High-performance network path via Joyride
- App 2  
Corner-case applications: fall back to the traditional TCP/IP stack



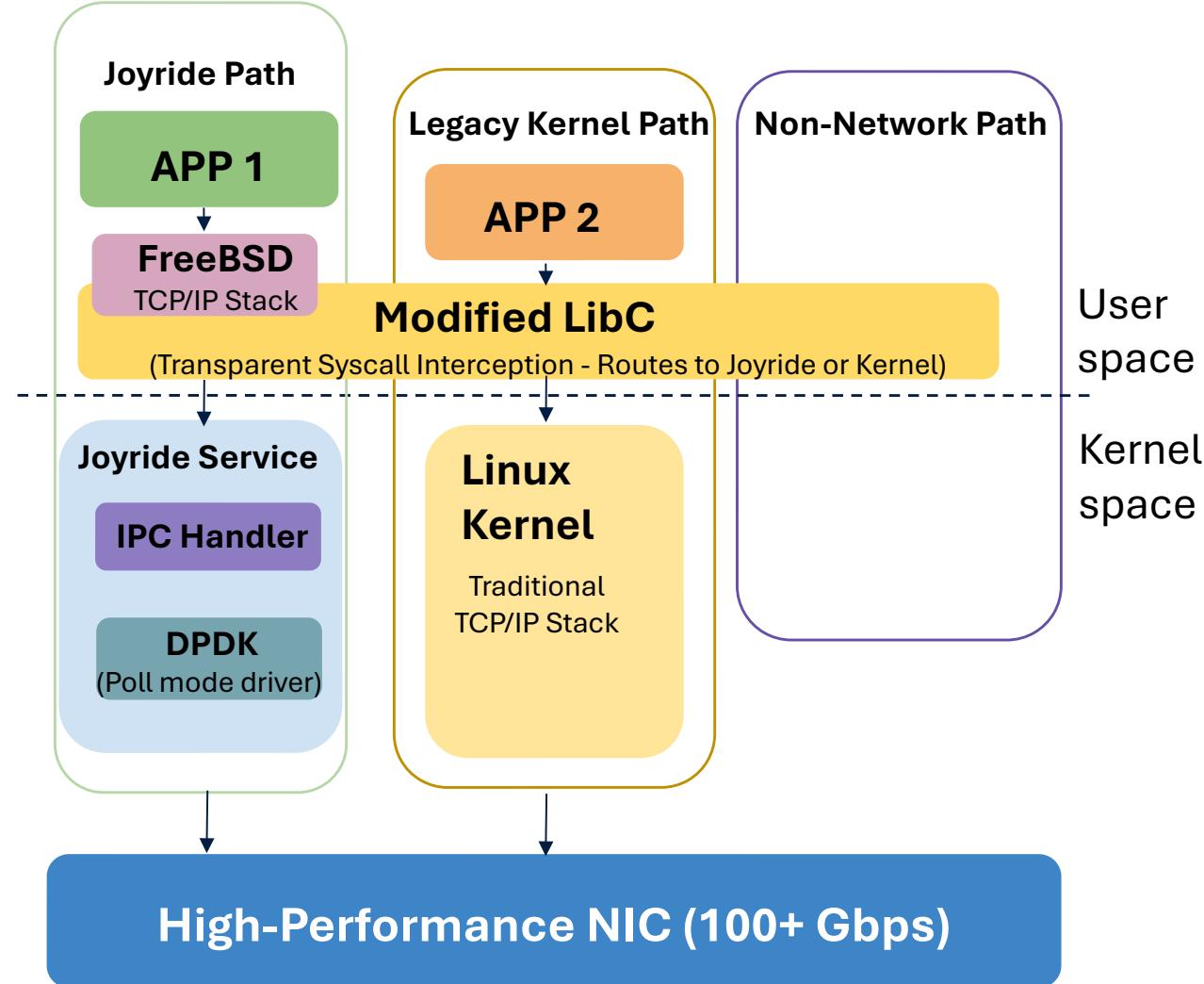
# Joyride's Architecture

- App 1  
High-performance network path via Joyride
- App 2  
Corner-case applications: fall back to the traditional TCP/IP stack
- App 3  
Program not related to network



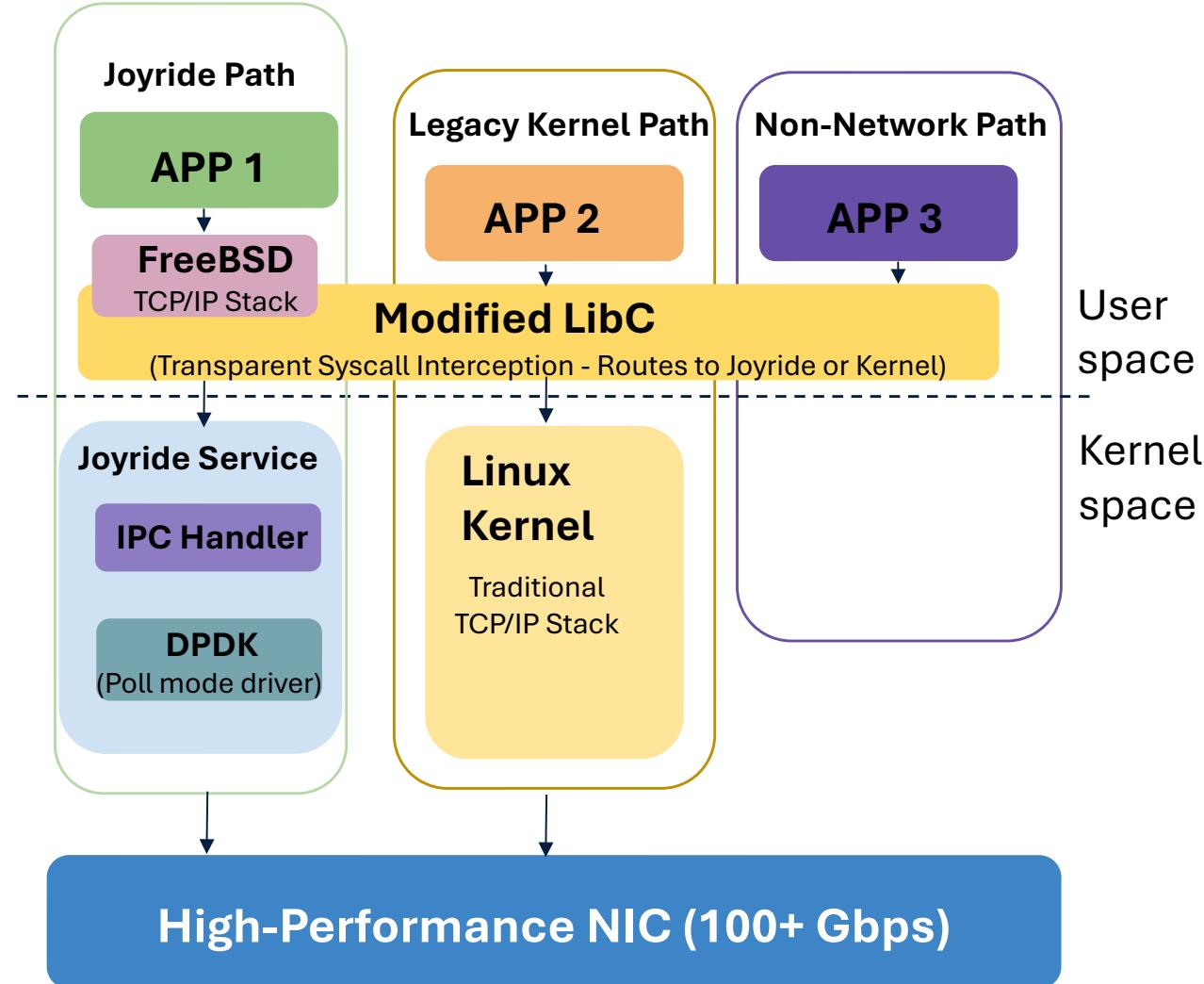
# Joyride's Architecture

- App 1  
High-performance network path via Joyride
- App 2  
Corner-case applications: fall back to the traditional TCP/IP stack
- App 3  
Program not related to network



# Joyride's Architecture

- App 1  
High-performance network path via Joyride
- App 2  
Corner-case applications: fall back to the traditional TCP/IP stack
- App 3  
Program not related to network

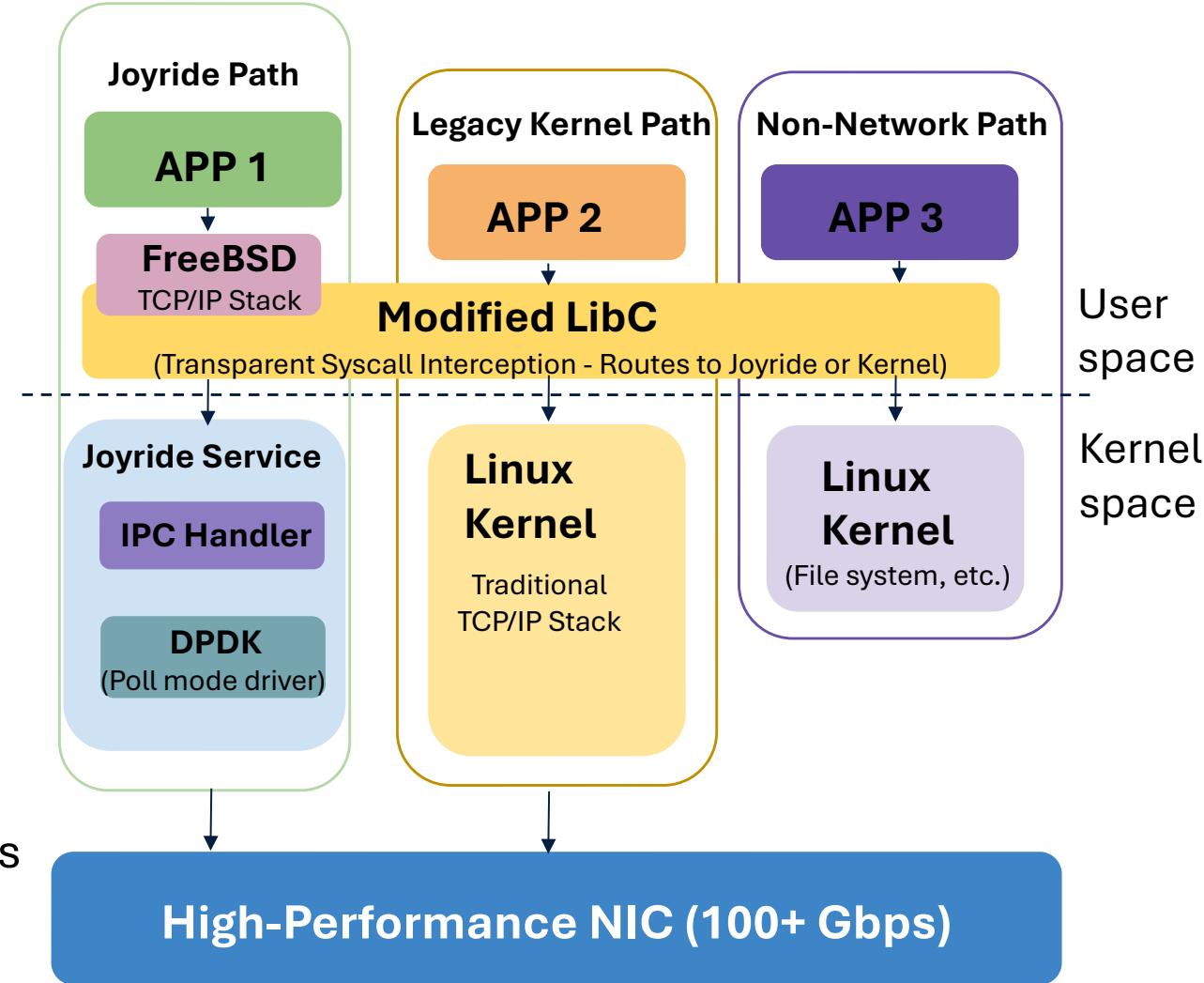


# Joyride's Architecture

- App 1  
High-performance network path via Joyride
- App 2  
Corner-case applications: fall back to the traditional TCP/IP stack
- App 3  
Program not related to network

Single NIC shared transparently across all paths using SR-IOV and VF (Virtual Functions)

*Potentially multiple isolated Joyride Services*

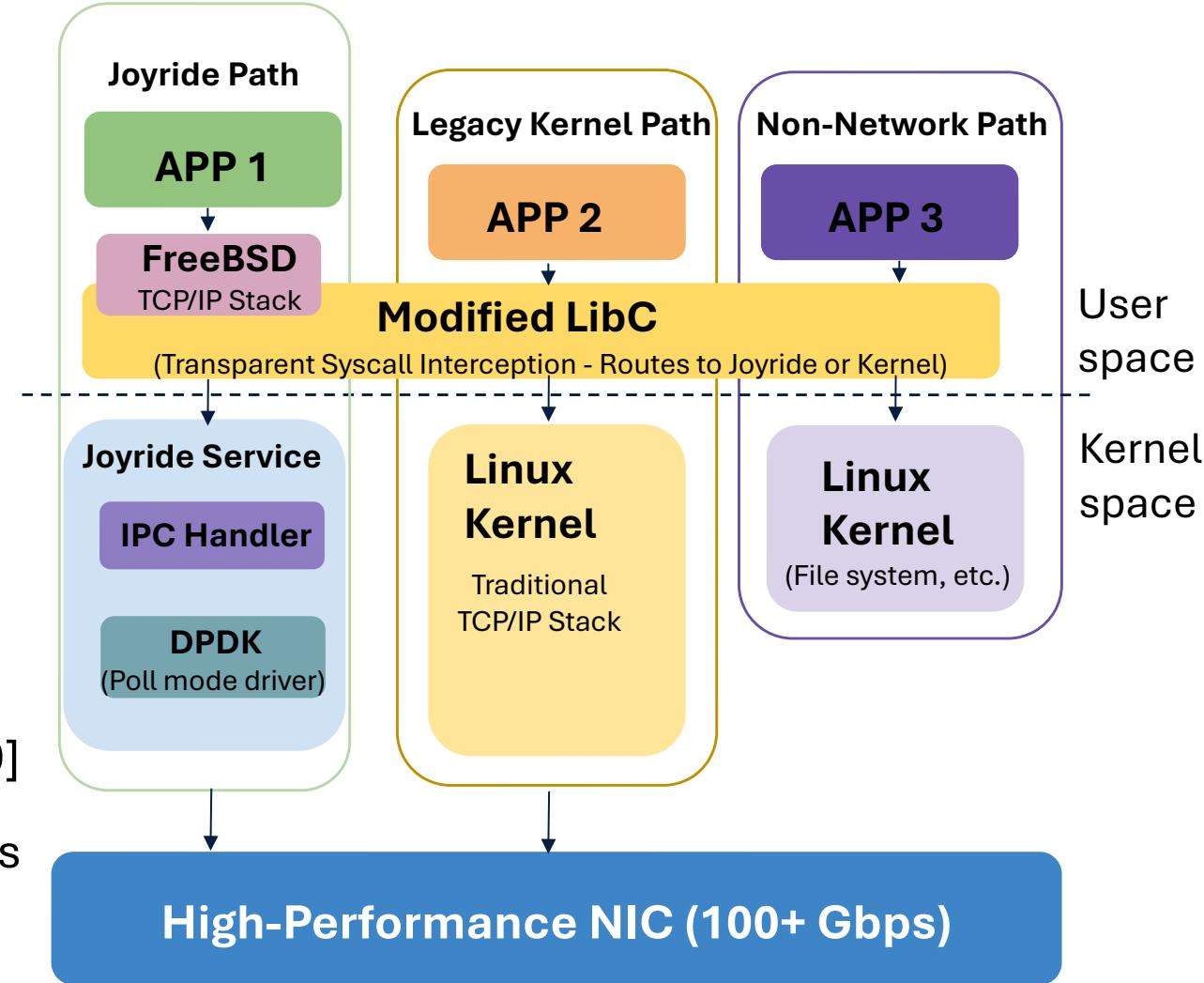


# Joyride's Architecture

- App 1  
High-performance network path via Joyride
- App 2  
Corner-case applications: fall back to the traditional TCP/IP stack
- App 3  
Program not related to network
- Potentially a **hybrid** architecture for high-demanding applications, LibrettOS [VEE'20]

Single NIC shared transparently across all paths using SR-IOV and VF (Virtual Functions)

**Potentially multiple isolated Joyride Services**



# Discussion: Security and Isolation



PennState

# Discussion: Security and Isolation

- **Privilege Separation:**

Network stack runs entirely in user space



PennState

# Discussion: Security and Isolation

- **Privilege Separation:**

- Network stack runs entirely in user space

- Faults and breaches are contained in the network process (Joyride service)



# Discussion: Security and Isolation

- **Privilege Separation:**

- Network stack runs entirely in user space

- Faults and breaches are contained in the network process (Joyride service)

- Reduced trusted computing base (potentially fully eliminating Linux's network stack)



# Discussion: Security and Isolation

- **Privilege Separation:**

- Network stack runs entirely in user space

- Faults and breaches are contained in the network process (Joyride service)

- Reduced trusted computing base (potentially fully eliminating Linux's network stack)

- **Hardware Sharing:**

- Virtual Functions (VFs) provide hardware-level isolation for multiple instances

- No direct NIC access from applications



# Discussion: Security and Isolation

- **Privilege Separation:**

- Network stack runs entirely in user space

- Faults and breaches are contained in the network process (Joyride service)

- Reduced trusted computing base (potentially fully eliminating Linux's network stack)

- **Hardware Sharing:**

- Virtual Functions (VFs) provide hardware-level isolation for multiple instances

- No direct NIC access from applications

- **Performance:**

- Will use fast inter-process communication mechanisms based on shared memory and scalable data structures



# Preliminary Results: Experimental Setup

## **Hardware:**

AMD EPYC 9005 series (both client and server)

Intel E810 100 Gbps NICs

## **System:**

Ubuntu 22.04

Linux kernel 6.2

**Benchmark:** ttcp (Linux), a built-in throughput test (DPDK)

Standard ttcp version for blocking socket tests

Custom non-blocking ttcp variant implemented for comparison

(Note: the original ttcp tool did not support the non-blocking mode, we implemented our own ttcp variant to test performance under the non-blocking mode)

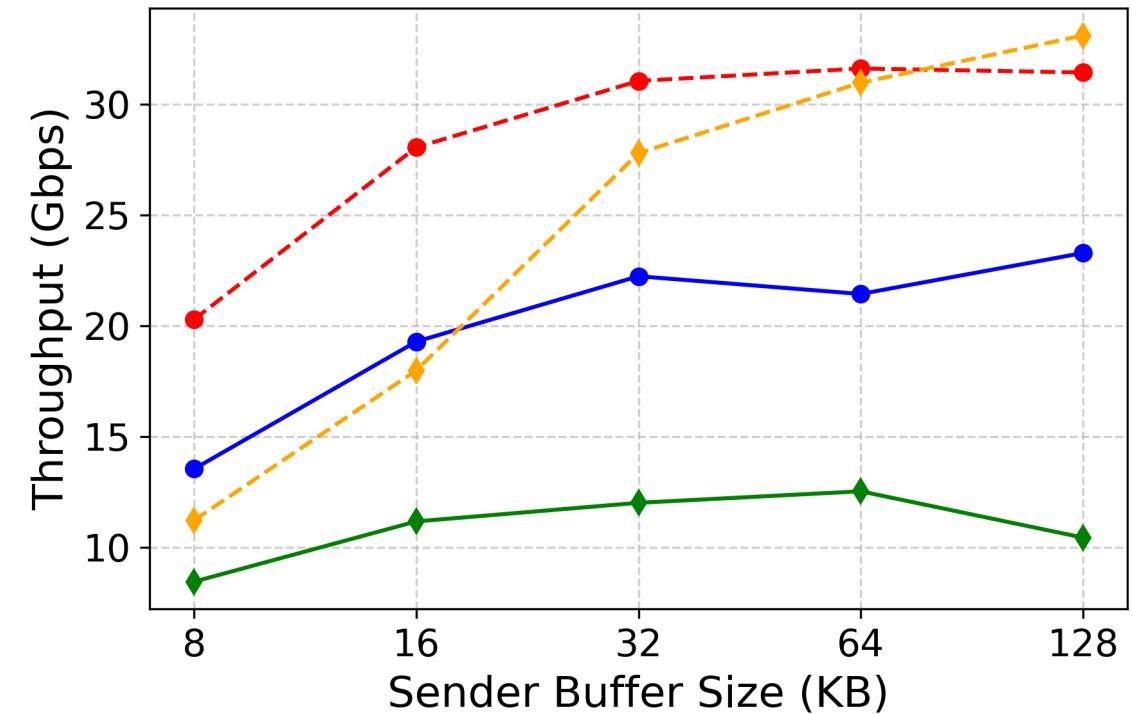


# Preliminary Results

## Single-Process Throughput vs. Buffer Size (Linux):

- Performance is greatly affected by buffer size and whether it is blocking vs. non-blocking
- Both implementations plateau far below link capacity

—●— Blocking Recv 8 KB Buffer  
-●--- Blocking Recv 256KB Buffer  
—◆— Non-Blocking Recv 8 KB Buffer  
—◆--- Non-Blocking Recv 256KB Buffer

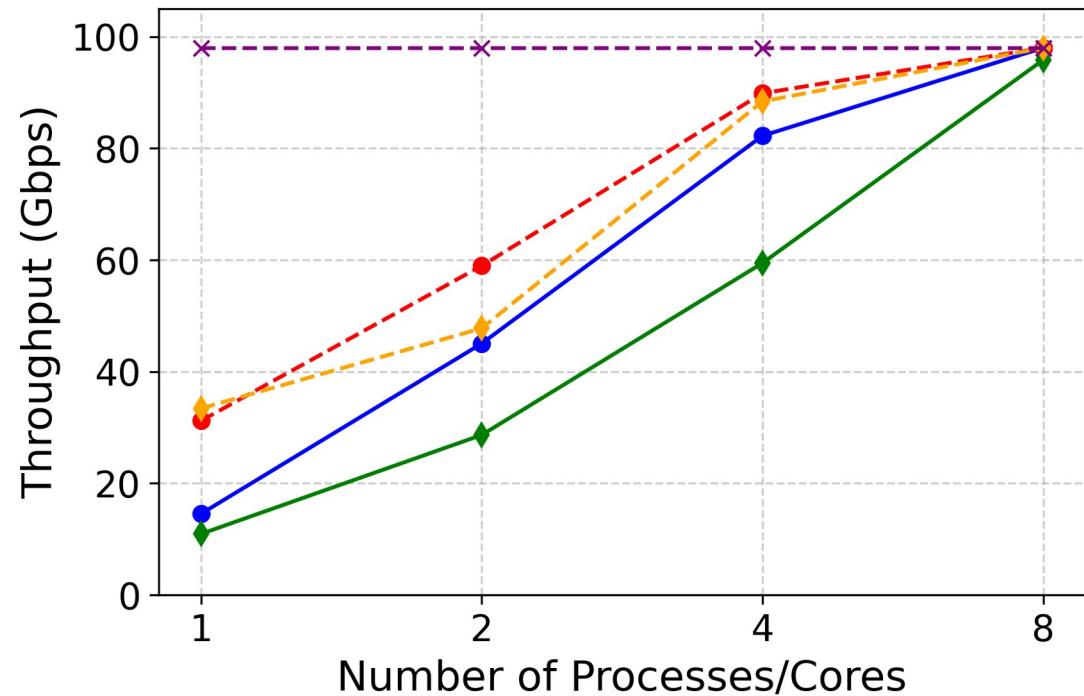


# Preliminary Results

## Aggregate Network Throughput:

- 4-8 cores saturate the link in Linux vs. just 1 core with DPDK
- Linux's network stack and mode switches have a visible performance overhead
- Buffer-size increases help but provide only a sublinear improvement
- Linux's non-blocking version is not helping much to reduce the overhead

—●— Blocking Send 8KB Buffer  
-●--- Blocking Send 256KB Buffer  
—◆— Non-Blocking Send 8KB Buffer  
—◆--- Non-Blocking Send 256KB Buffer  
-×--- DPDK Reference



# Future Work

## ● Design a New TCP Stack and User-Space Server:

Port the most recent FreeBSD TCP/IP code to run over DPDK

Avoid shortcuts made in F-stack

## ● LibC Replacement and Integration Layer:

Complete POSIX socket API coverage with poll/select/epoll/etc

## ● Real-Life Tests:

Web servers (Nginx, Apache)

Databases (PostgreSQL, Redis)

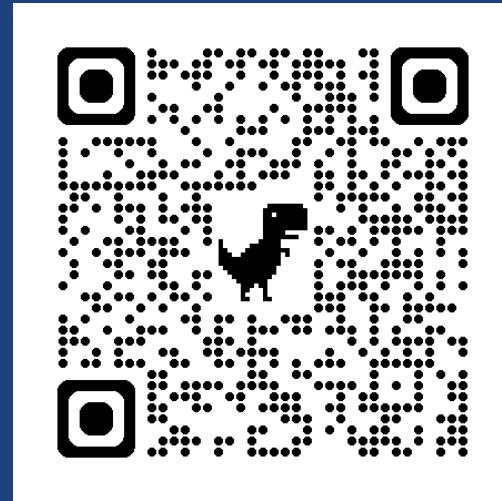


# Thank you!

Q&A?

Yanlin Du  
duyanlin@psu.edu

Ruslan Nikolaev  
rnikola@psu.edu



PennState