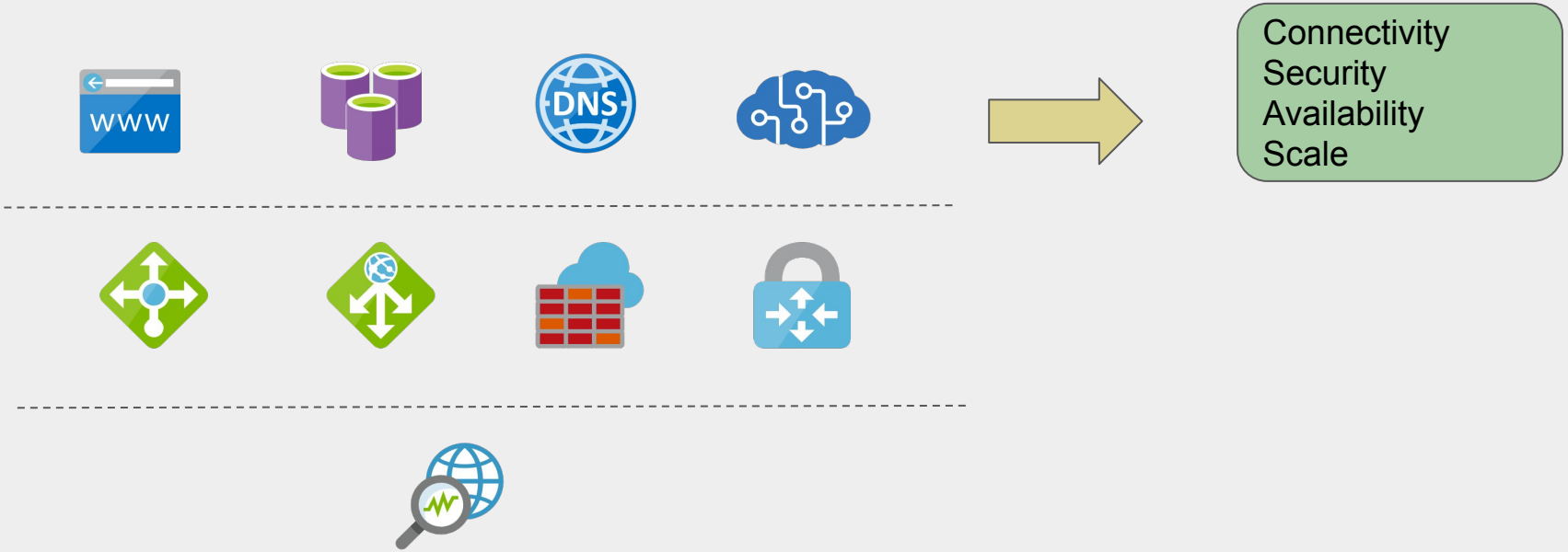
# Synergistic Server-Based Network Processing Stack

Dissertation Defense
by **Marcelo Abranches**
November 4th 2022

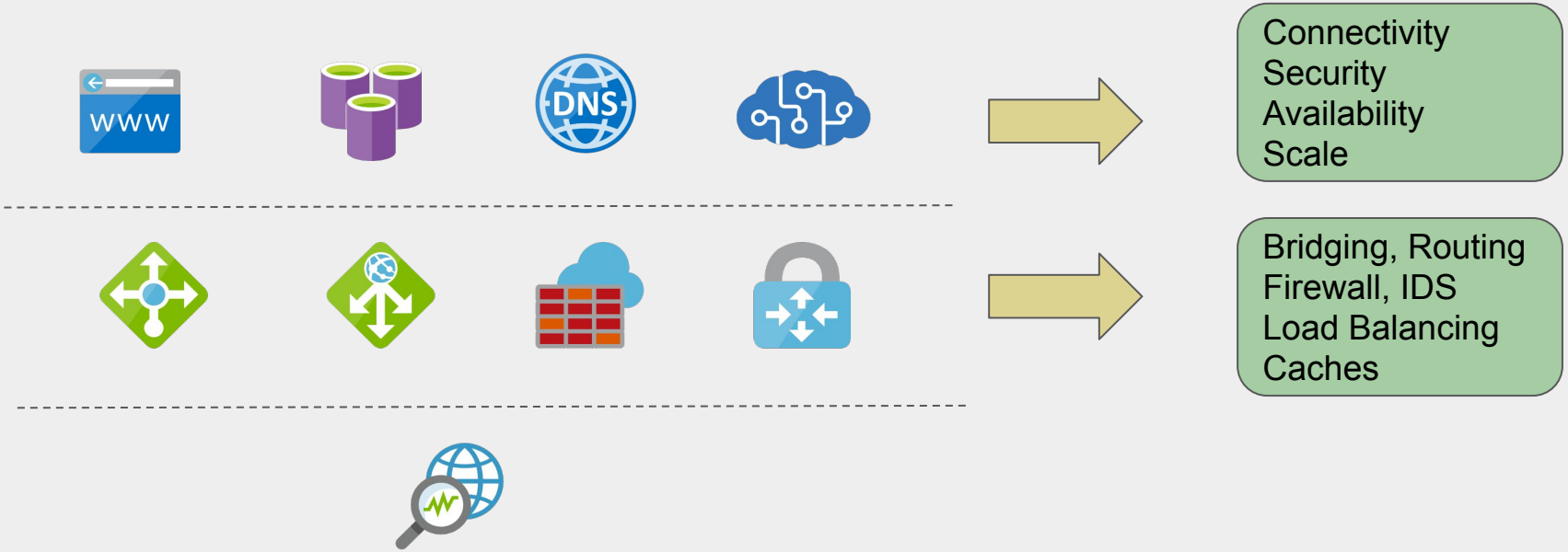
Committee

Eric Keller
Tamara Lehman
Joseph Izraelevitz
Eric Wustrow
Dirk Grunwald

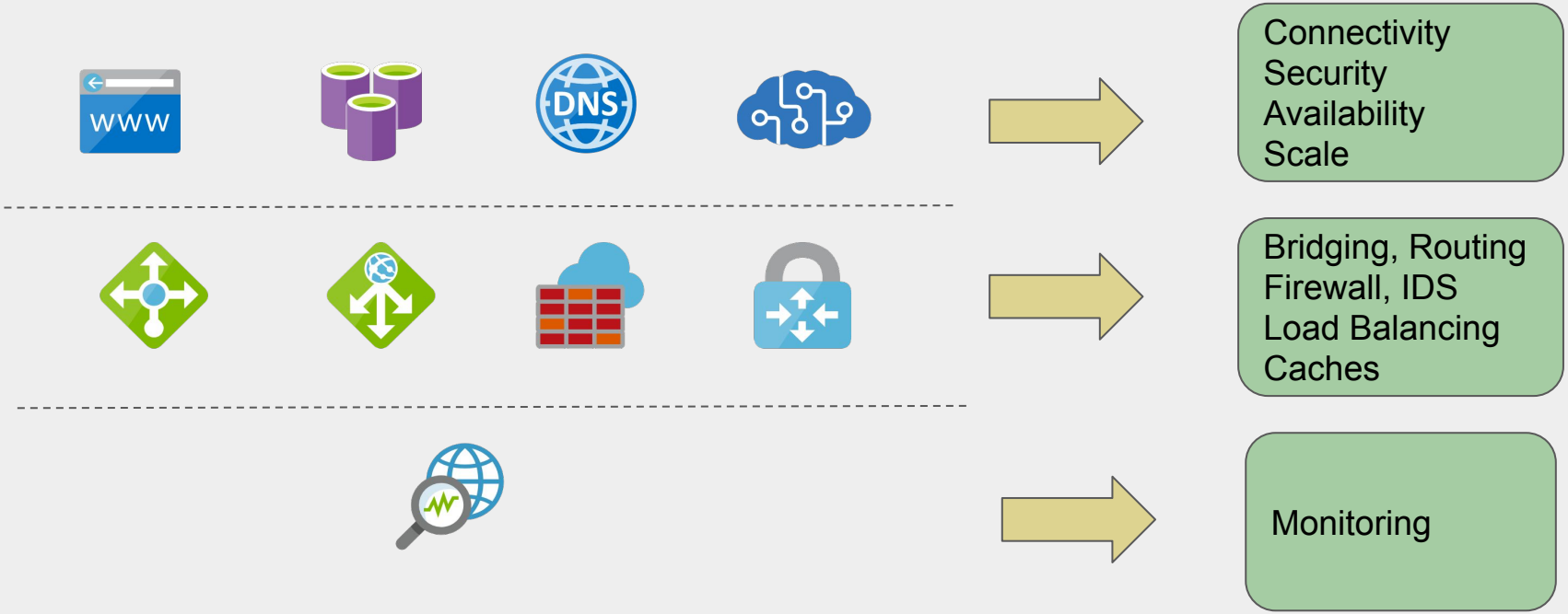
# Infrastructure needs



# Infrastructure needs

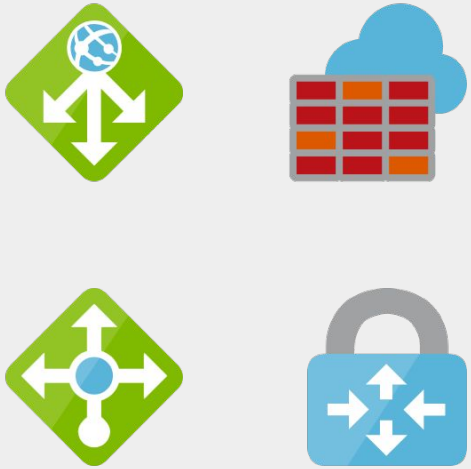


# Infrastructure needs

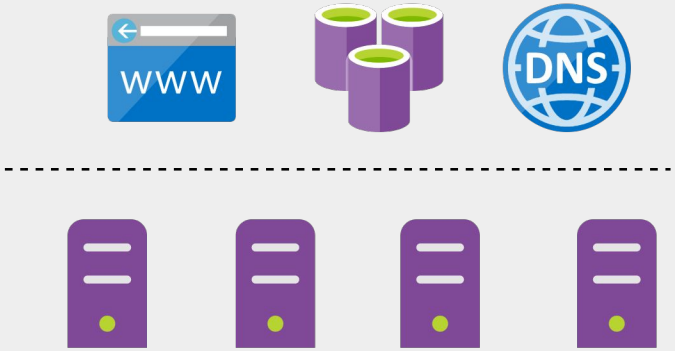


# Legacy Network Functions

Hardware appliances

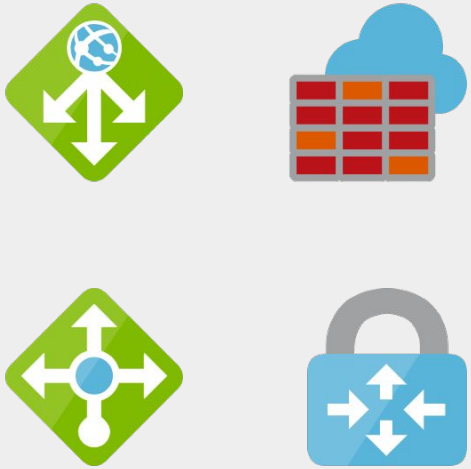


Mostly static infrastructure



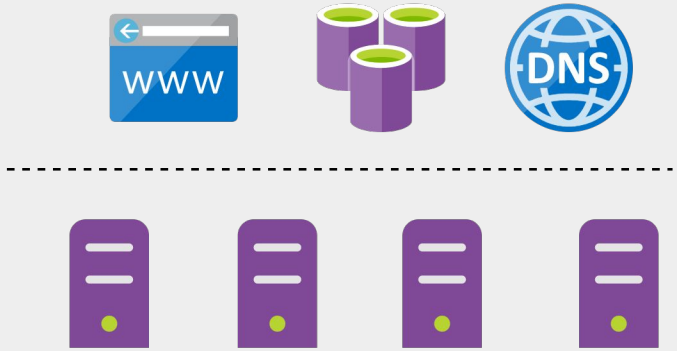
# Legacy Network Functions

Hardware appliances



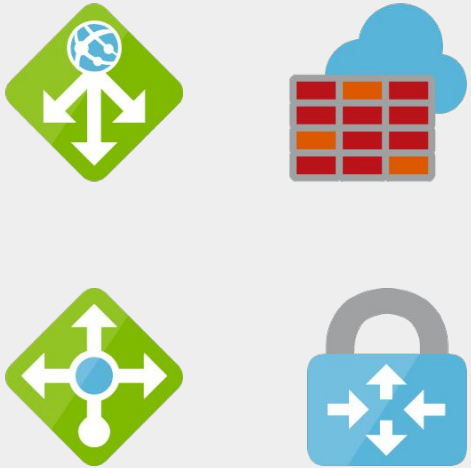
Good Fit!

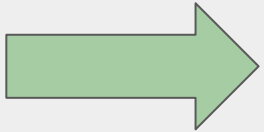
Mostly static infrastructure



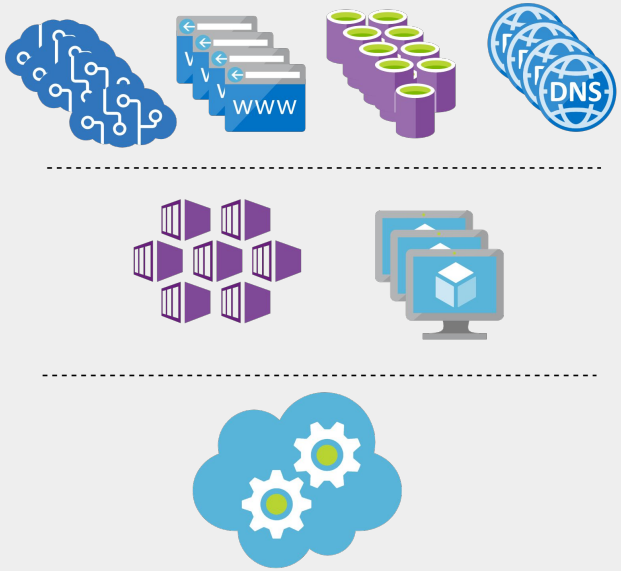
# Legacy Network Functions

Hardware appliances





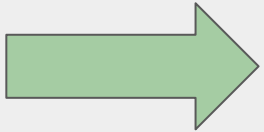
Dynamic Infrastructure



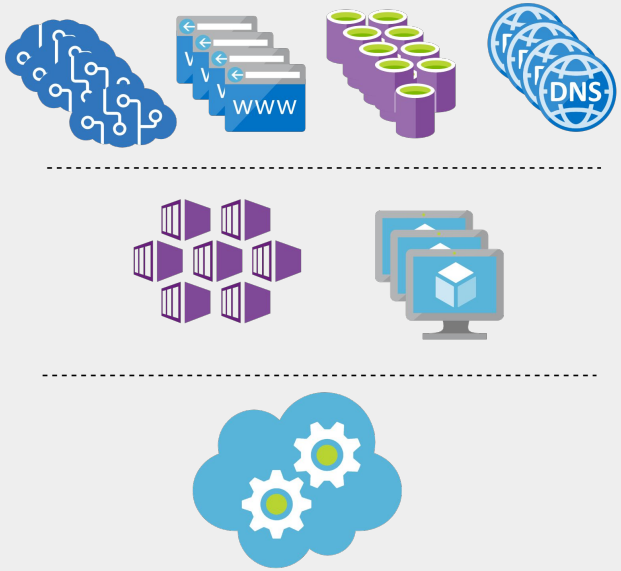
# Legacy Network Functions

Hardwar iances

Not ideal!

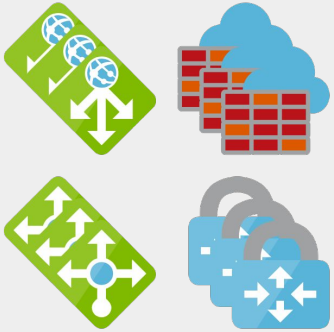


Dynamic Infrastructure

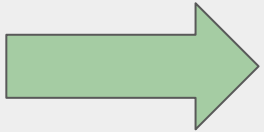


# Move to software

Software appliances



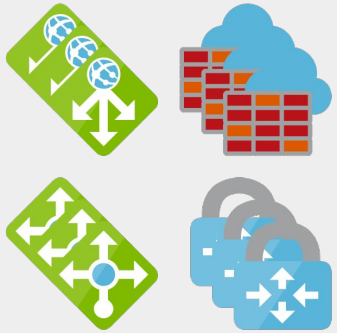
Agile, flexible and dynamic



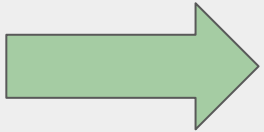
Dynamic Infrastructure


# Move to software

Software appliances



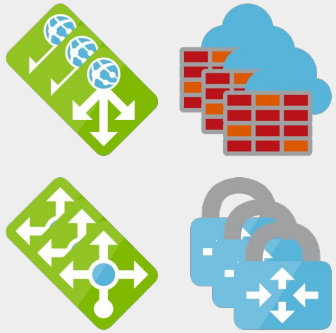
Good fit!



Dynamic Infrastructure


# Software Packet Processing

Few nanoseconds to process each packet



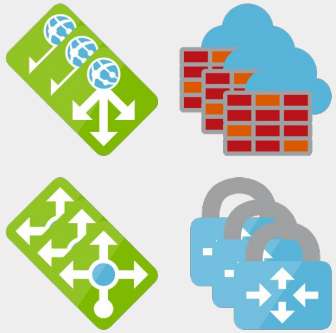

Low latency
High Throughput
High loads


# Software Packet Processing

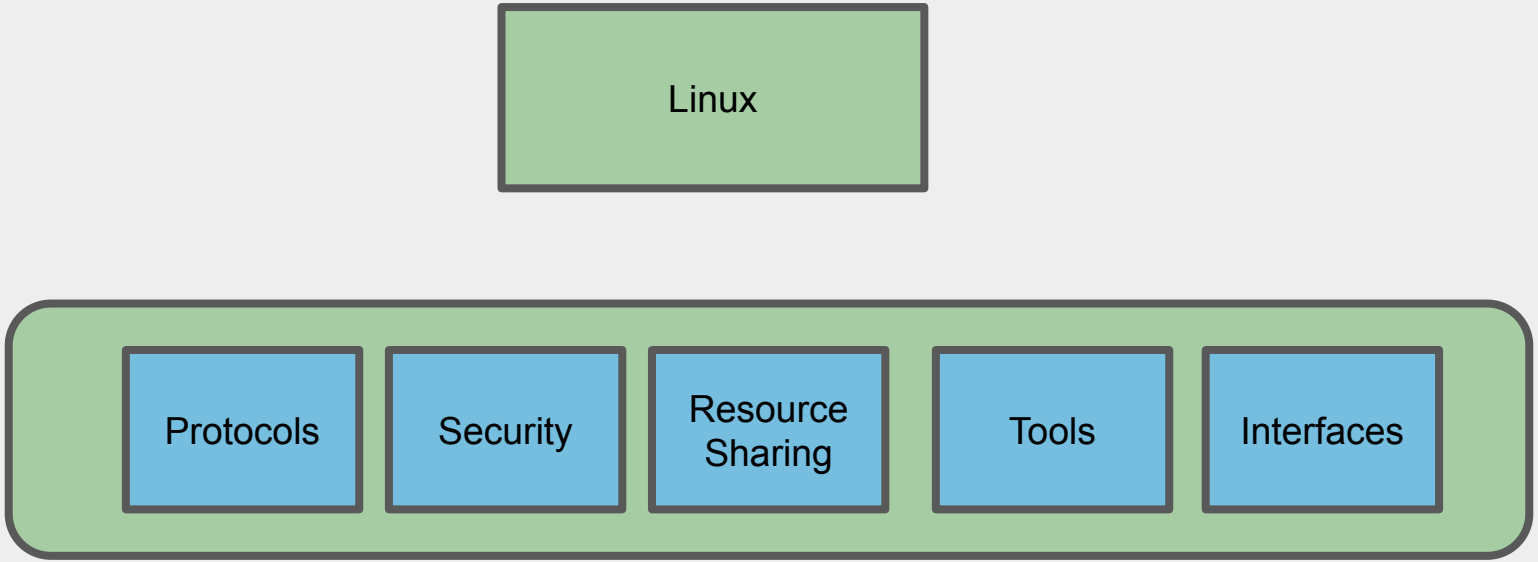
Few nanoseconds to process each packet

Avoid as much unnecessary processing as possible

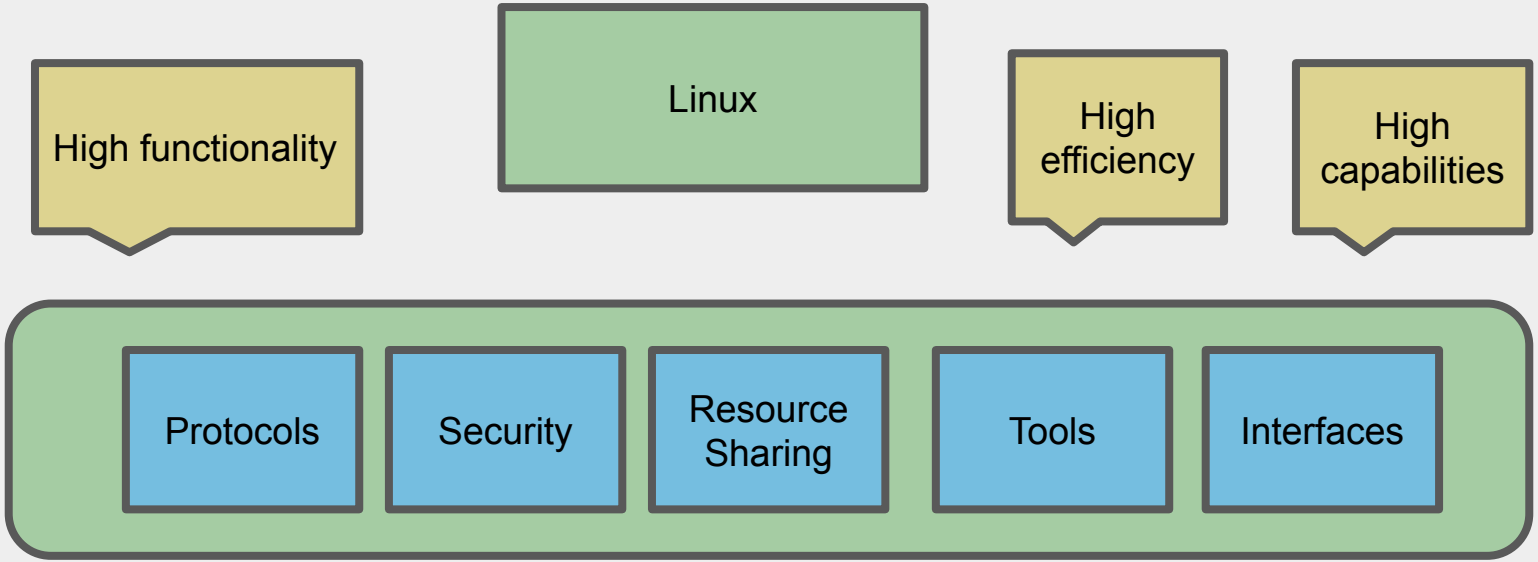
Low latency
High Throughput
High loads



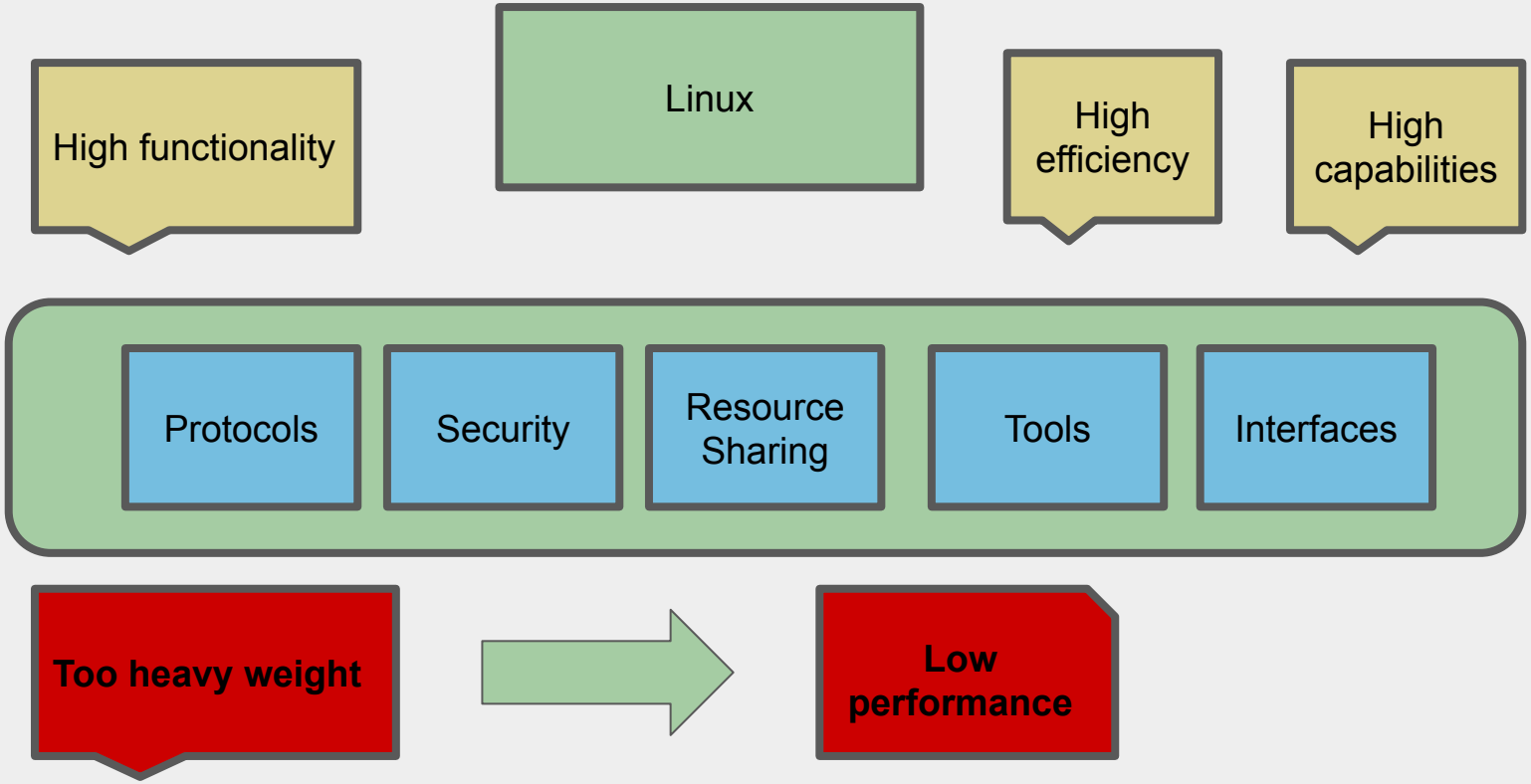

# Software Packet Processing



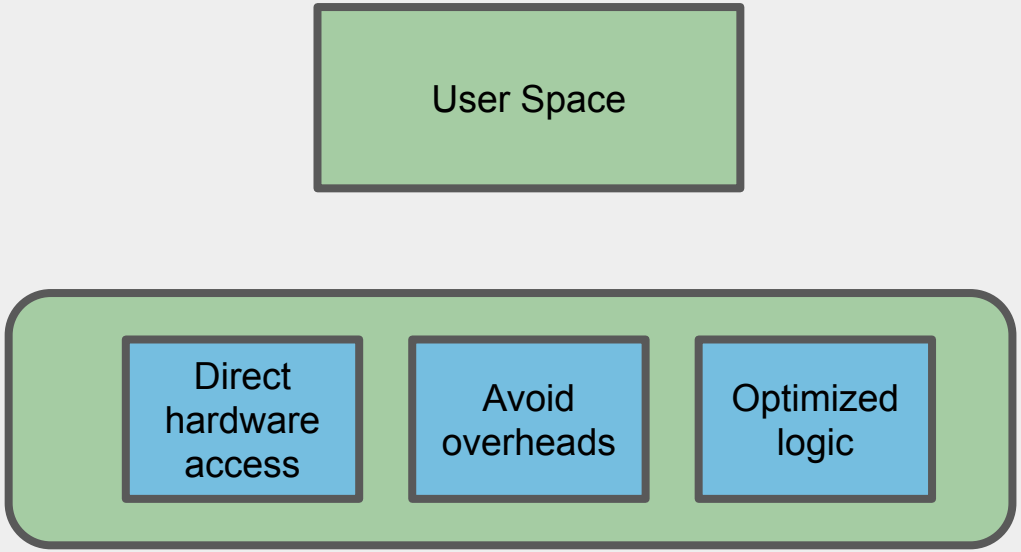
# Software Packet Processing



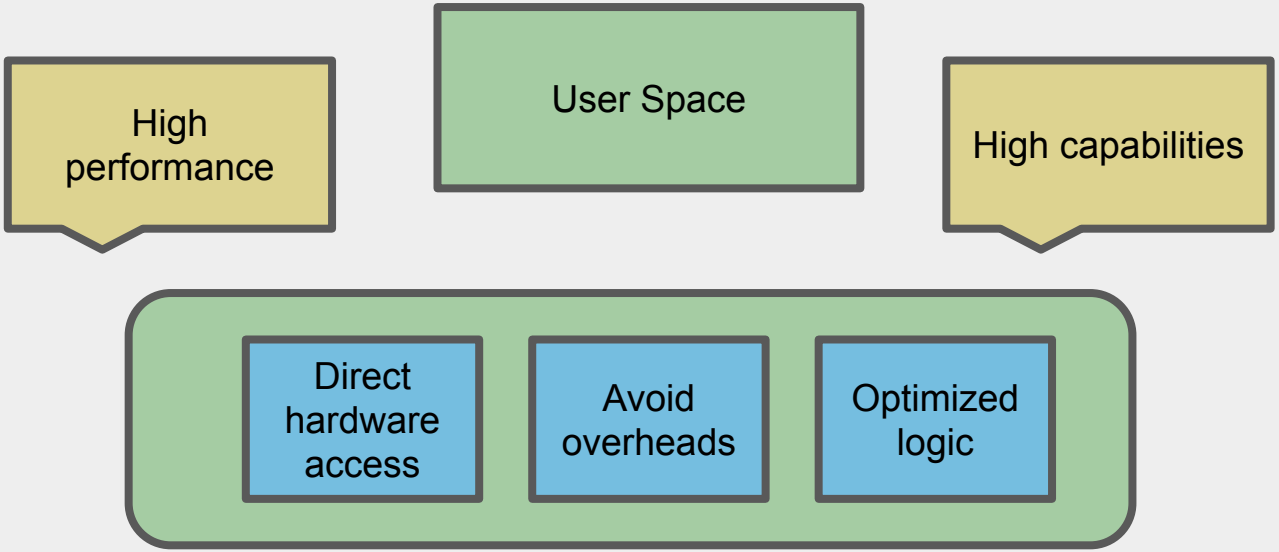
# Software Packet Processing



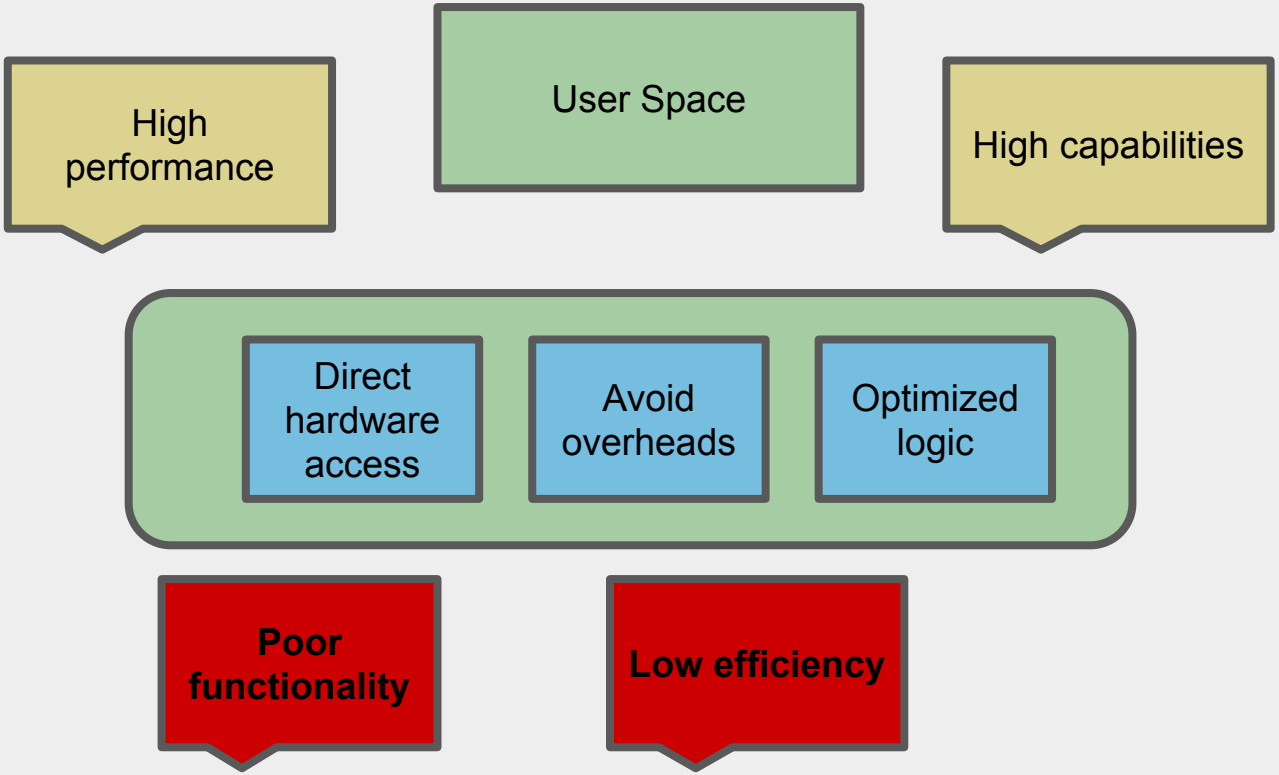
# Software Packet Processing



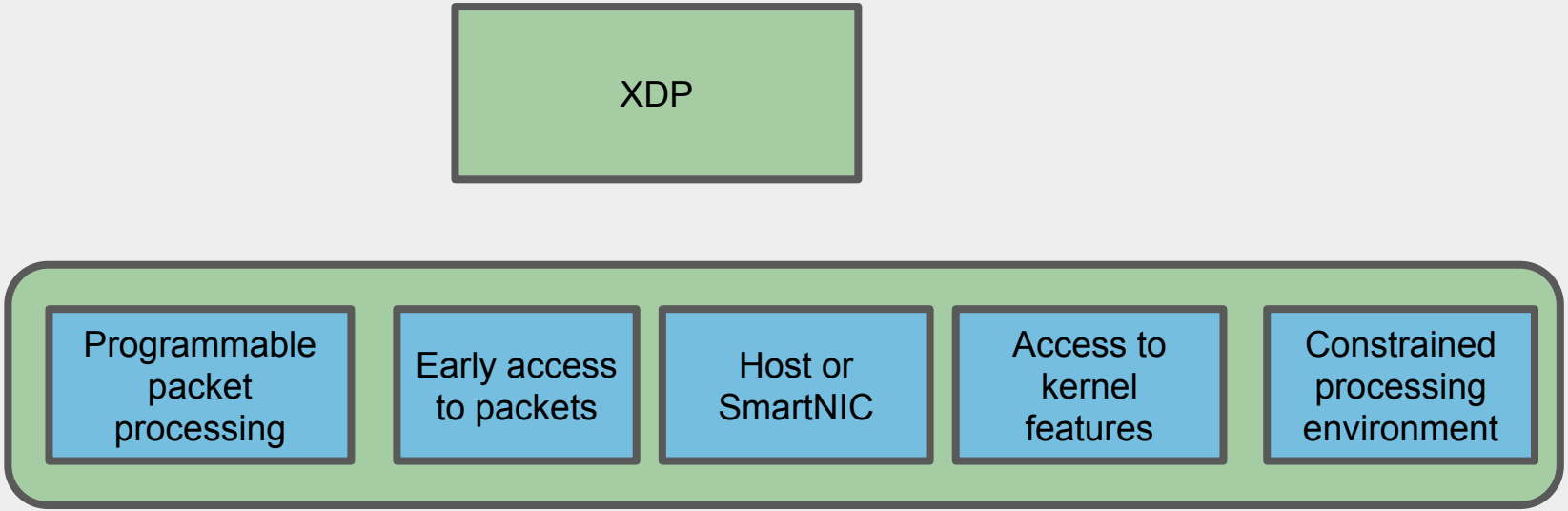
# Software Packet Processing



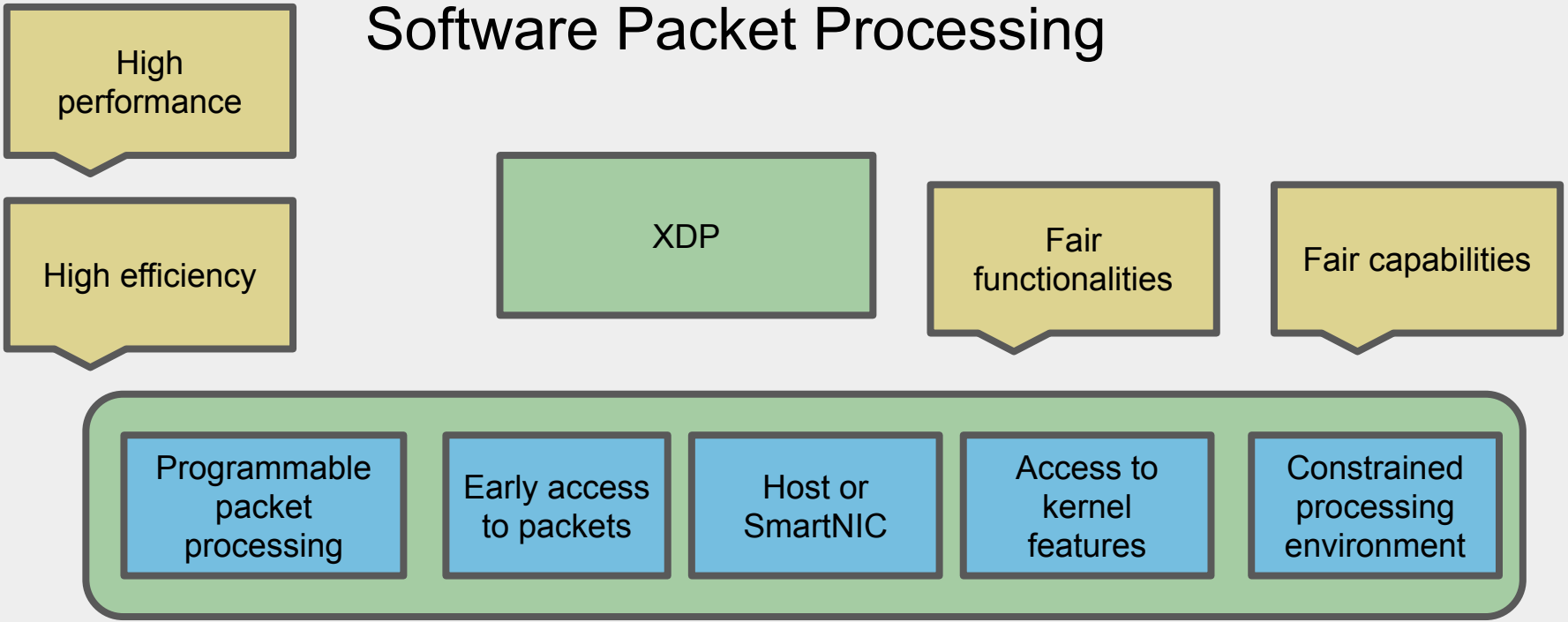
# Software Packet Processing



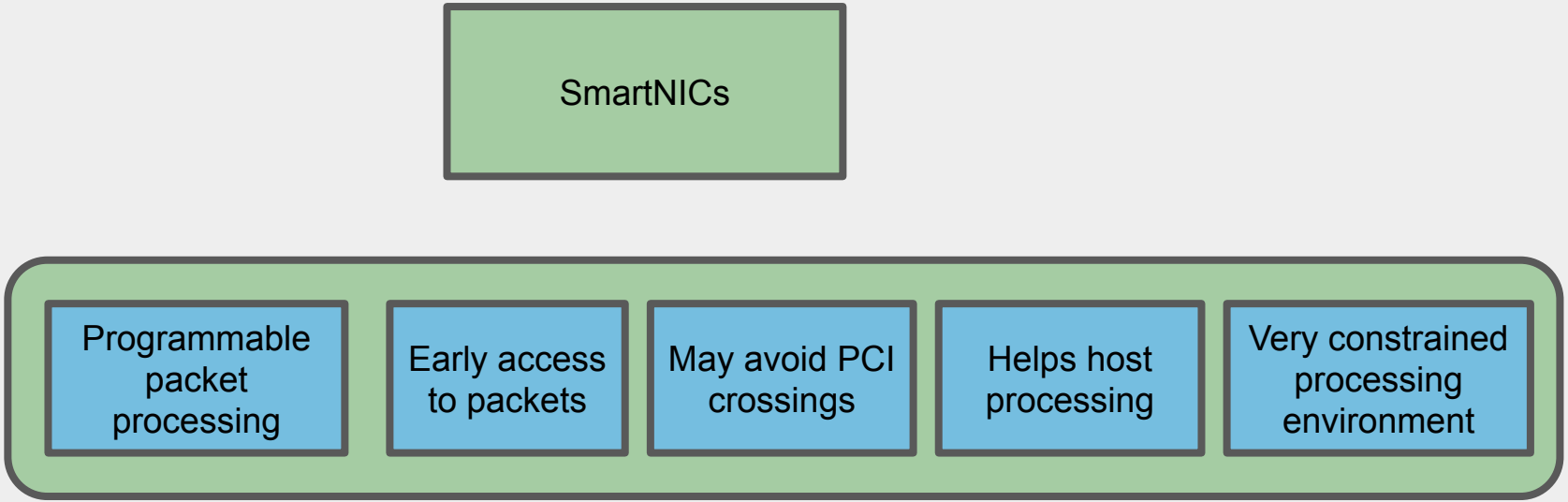
# Software Packet Processing



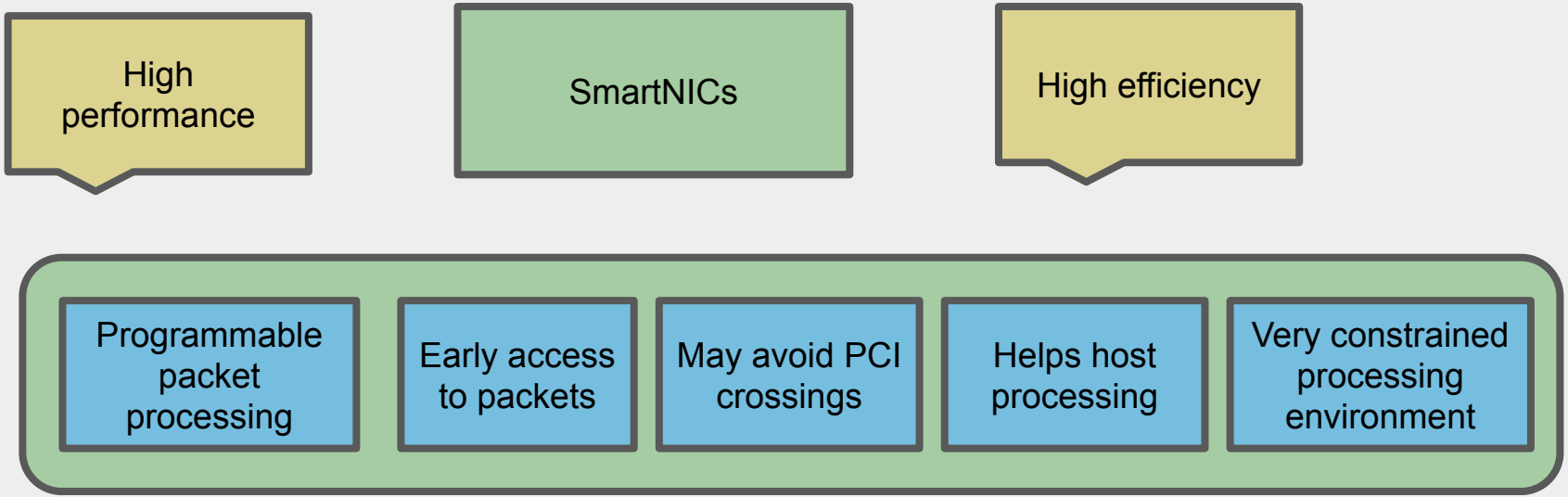
# Software Packet Processing



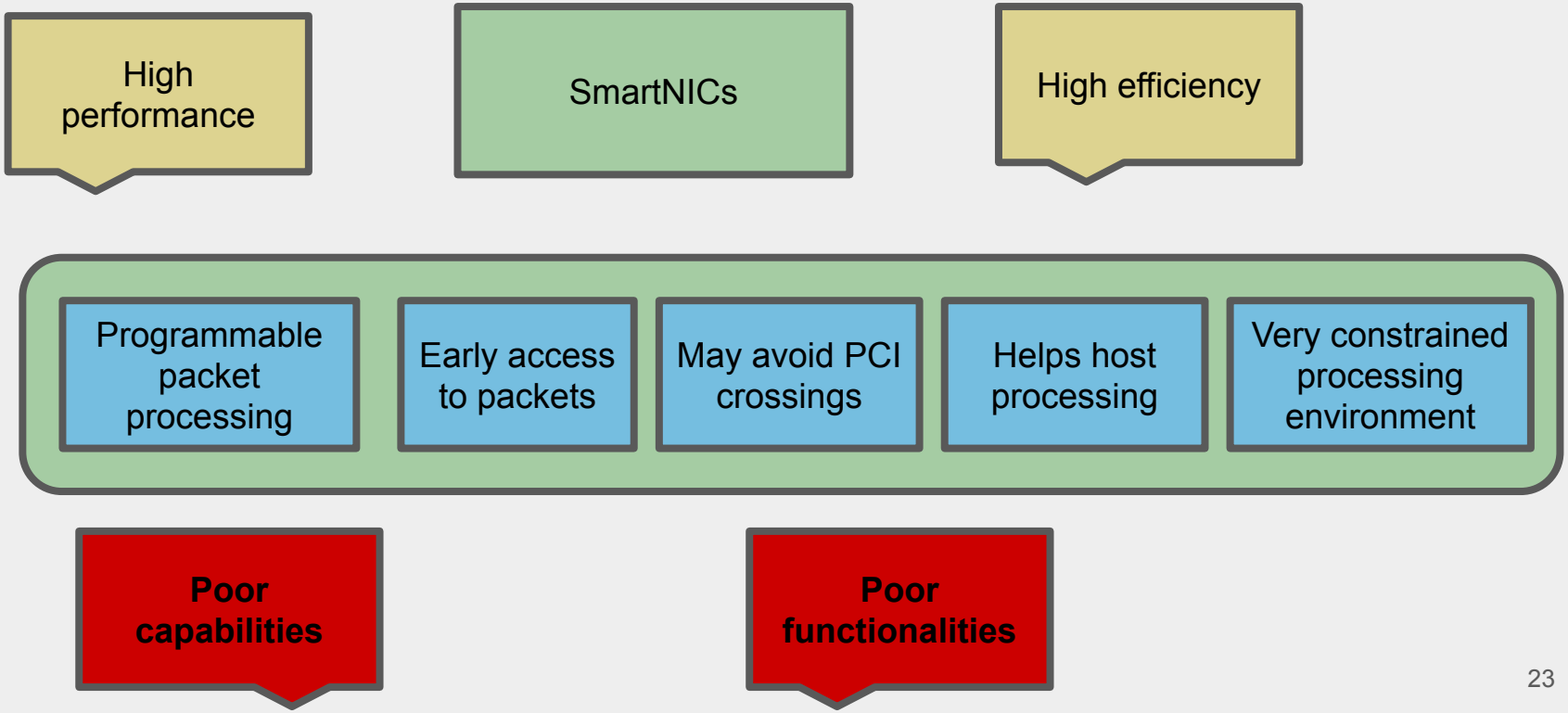
# Software Packet Processing



# Software Packet Processing



# Software Packet Processing



# Network Application Requirements

Performance + Efficiency + Functionality + Capability

# Network Application Requirements

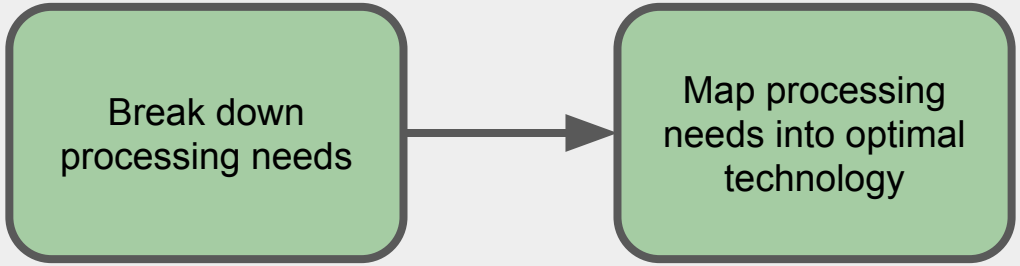
**No single packet processing technology can simultaneously provide all requirements**

Performance + Efficiency + Functionality + Capability

# Observation

**Different processing functions inside a network application have different processing requirements!**

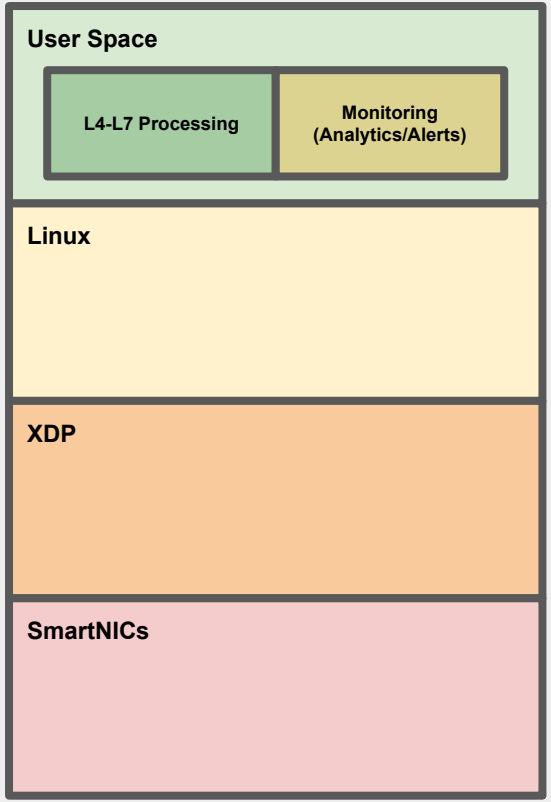
# Opportunity



# Mapping processing needs into optimal technology

| Packet Processing Needs | Network Monitoring Needs |
| --- | --- |
| **L4-L7 Processing**<br>Web Servers, Caches, Proxies, Stateful Firewalls, IDSs | **Analytics/Alerts**<br>Data mining, knowledge derivation, alert generation |
| **L2-L4 (Slow Path)**<br>Routing Protocols, Spanning Tree (STP) | **Routing, Sketches/stateful processing**<br>Routing packets through monitoring apps, HLL computation, Stateful processing |
| **L2-L4 (Fast Path)**<br>L2-L3 Forwarding, NAT, ACLs, Load Balancing | **Counters/pre-processing**<br>Compute/aggregate simple counters over packet fields, generate metadata |
| **System/Resource Management**<br>Resource Sharing, memory management, configuration tools | |

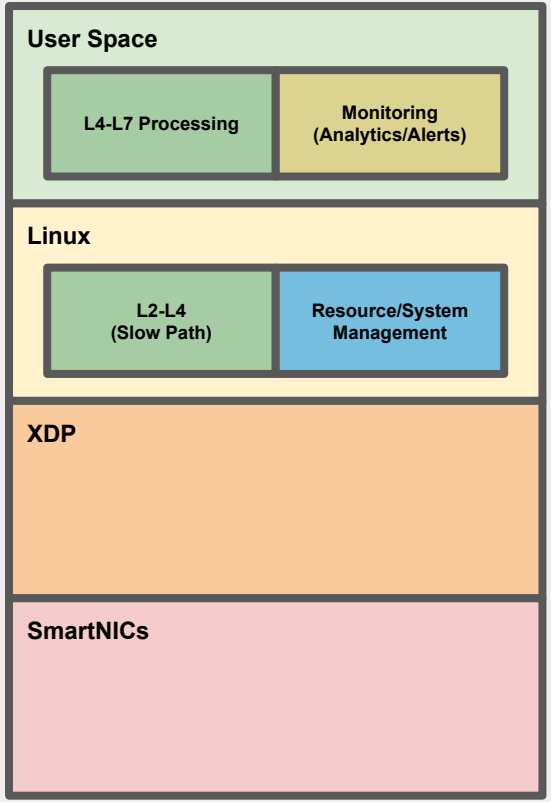
High performance and Capabilities



# Mapping processing needs into optimal technology

| Packet Processing Needs | Network Monitoring Needs |
| --- | --- |
| **L4-L7 Processing**<br>Web Servers, Caches, Proxies, Stateful Firewalls, IDSs | **Analytics/Alerts**<br>Data mining, knowledge derivation, alert generation |
| **L2-L4 (Slow Path)**<br>Routing Protocols, Spanning Tree (STP) | **Routing, Sketches/stateful processing**<br>Routing packets through monitoring apps, HLL computation, Stateful processing |
| **L2-L4 (Fast Path)**<br>L2-L3 Forwarding, NAT, ACLs, Load Balancing | **Counters/pre-processing**<br>Compute/aggregate simple counters over packet fields, generate metadata |
| **System/Resource Management**<br>Resource Sharing, memory management, configuration tools | |

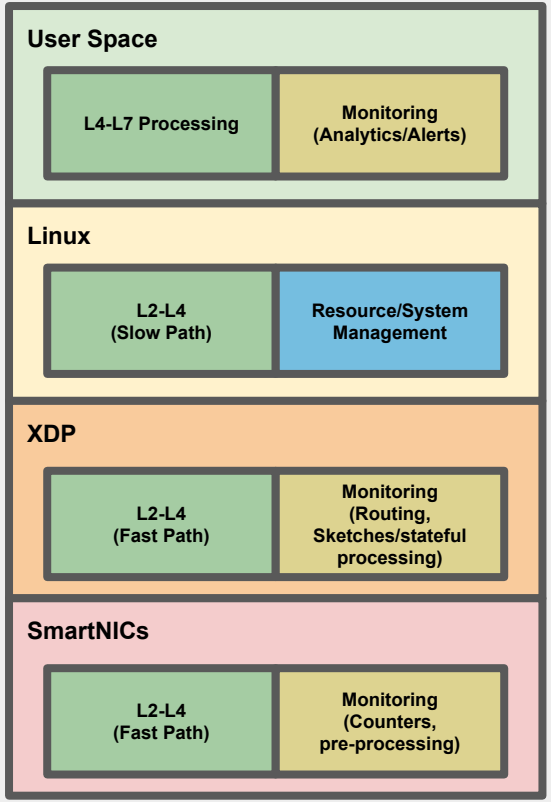
High efficiency and functionality



# Mapping processing needs into optimal technology

| Packet Processing Needs | Network Monitoring Needs |
| --- | --- |
| **L4-L7 Processing**<br>Web Servers, Caches, Proxies, Stateful Firewalls, IDSs | **Analytics/Alerts**<br>Data mining, knowledge derivation, alert generation |
| **L2-L4 (Slow Path)**<br>Routing Protocols, Spanning Tree (STP) | **Routing, Sketches/stateful processing**<br>Routing packets through monitoring apps, HLL computation, Stateful processing |
| **L2-L4 (Fast Path)**<br>L2-L3 Forwarding, NAT, ACLs, Load Balancing | **Counters/pre-processing**<br>Compute/aggregate simple counters over packet fields, generate metadata |
| **System/Resource Management**<br>Resource Sharing, memory management, configuration tools | |

Simple tasks and High performance



# High-level contributions

L4-L7 performance and feature richness

High-coverage monitoring

L2-L4 performance and feature richness

# Contribution I

L4-L7 performance and feature richness

# L4-L7 processing

App Firewalls

Web Services

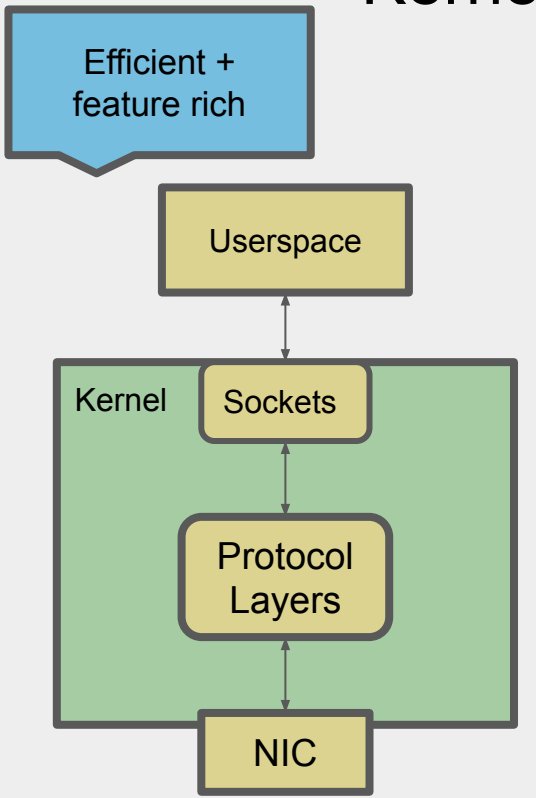

Proxies

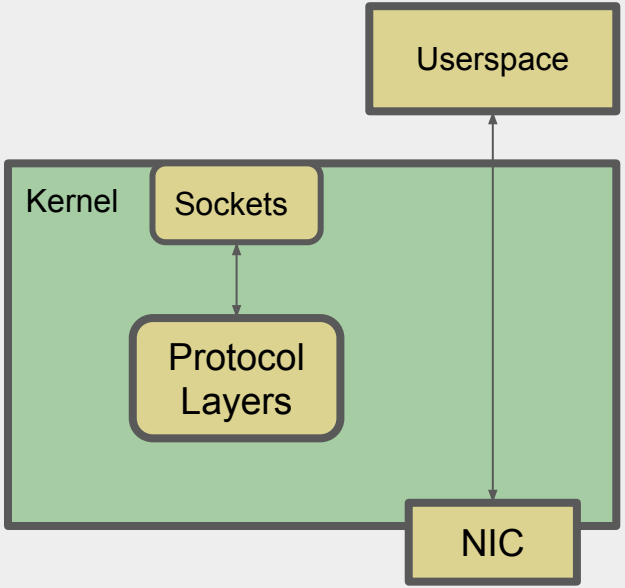
CDNs

Caches

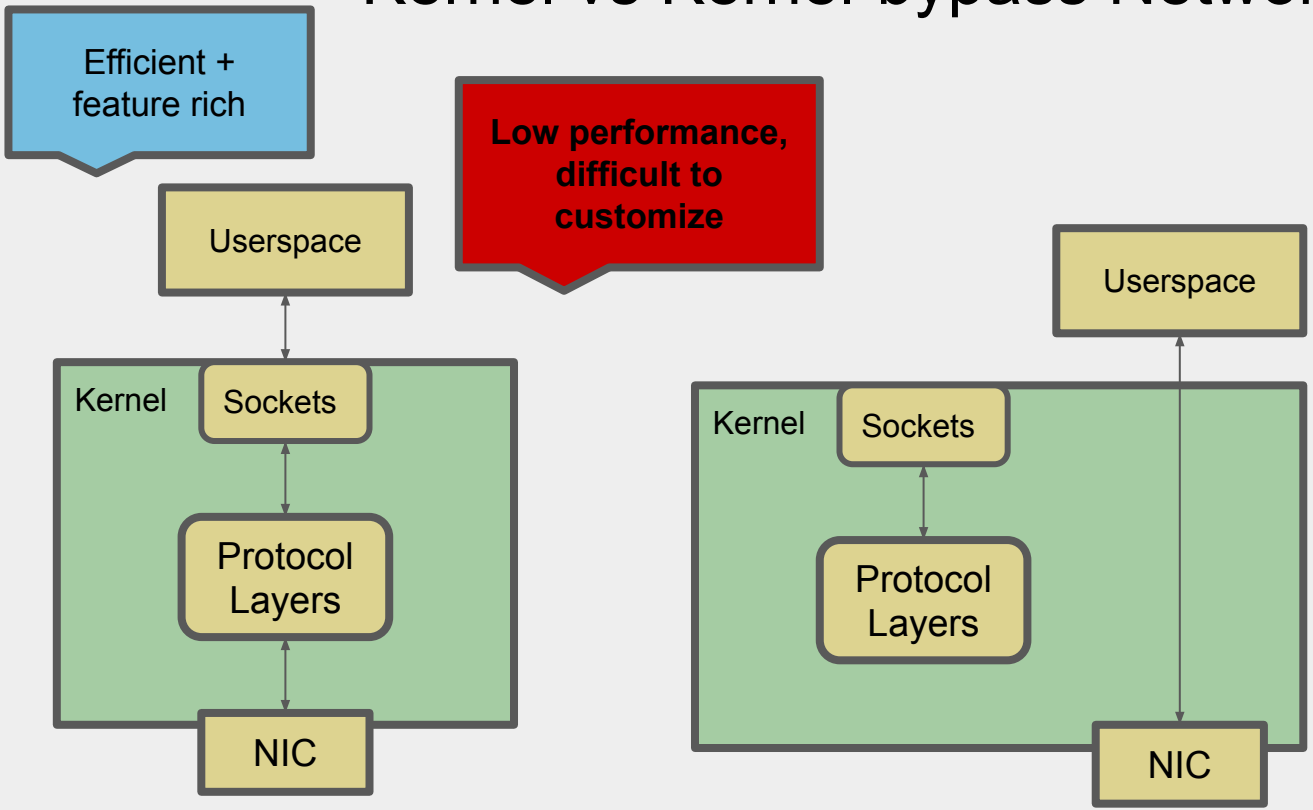
- Serve content
- Ensure performance and security

# Kernel vs Kernel-bypass Networking



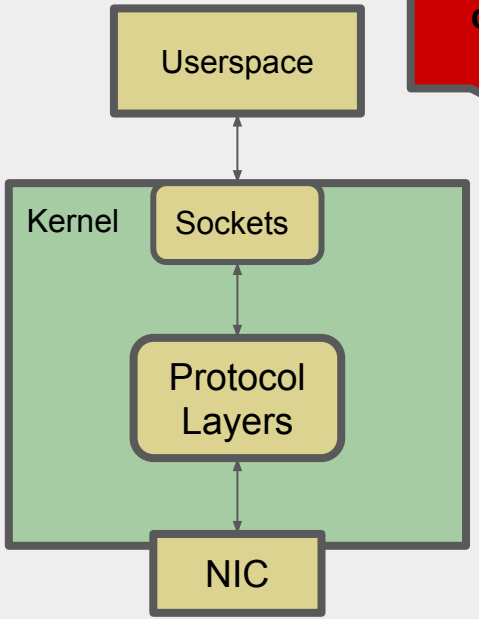
# Kernel vs Kernel-bypass Networking

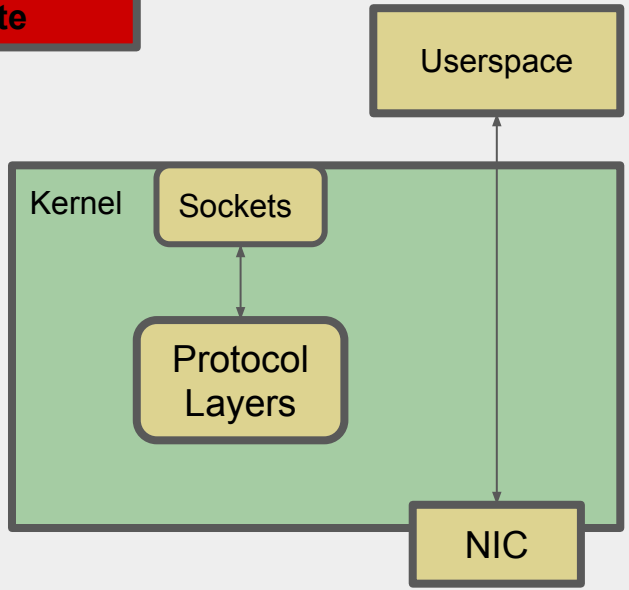


# Kernel vs Kernel-bypass Networking

Efficient + feature rich

**Low performance + difficult to customize and innovate**



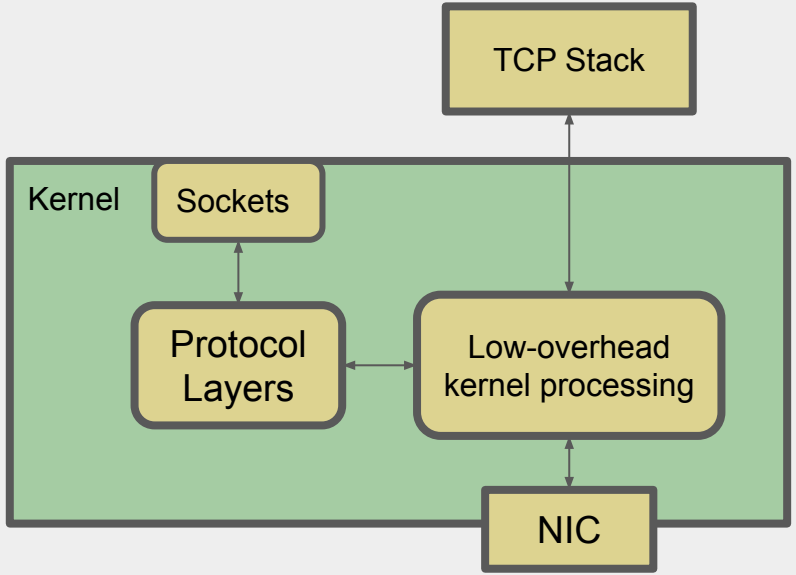




High performance + Easy to customize and Innovate

**Inefficient + Loses functionalities**

What if we leverage the good features provided by Linux Kernel to Enhance high-performance userspace L4-L7 Network Functions and applications?

# We propose a hybrid network stack

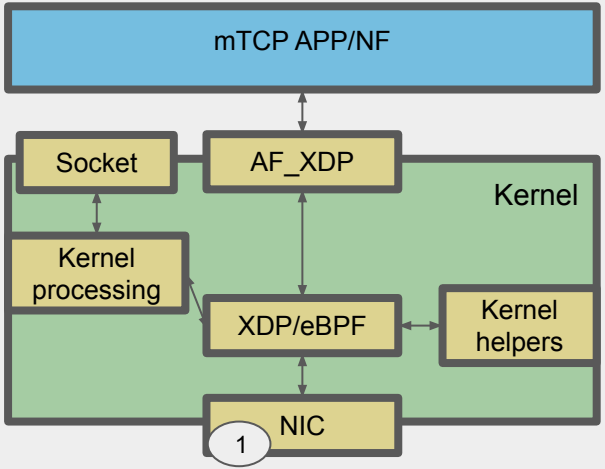



Leverage multicore TCP (mTCP) for scalability, performance and customization

Don't lose kernel functionality

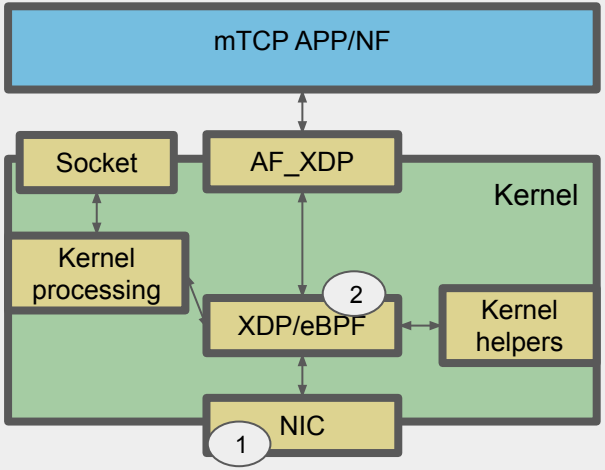
Build synergies between kernel and userspace stack

# Life of a packet in the hybrid stack




1) The packet arrives

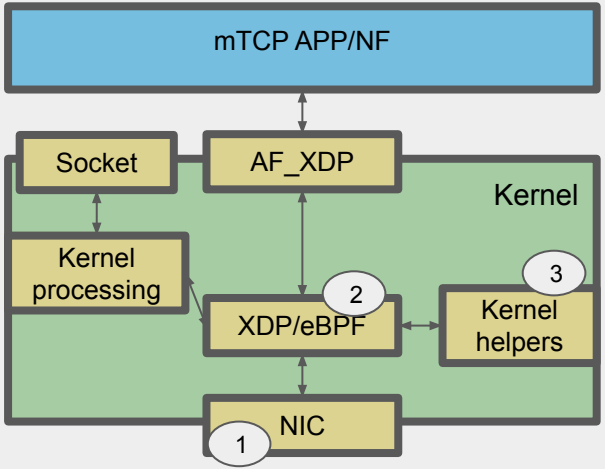
# Life of a packet in the hybrid stack




1) The packet arrives

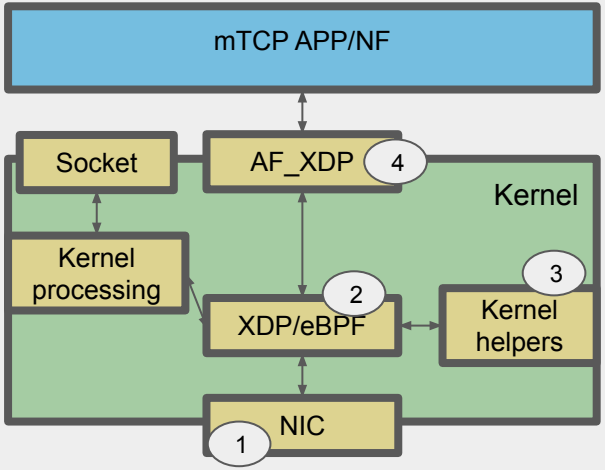
2) eBPF code is executed

# Life of a packet in the hybrid stack



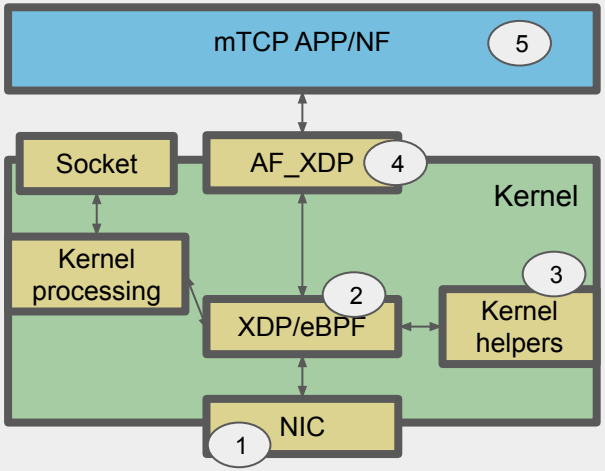

1) The packet arrives
2) eBPF code is executed
3) App can leverage kernel features

# Life of a packet in the hybrid stack



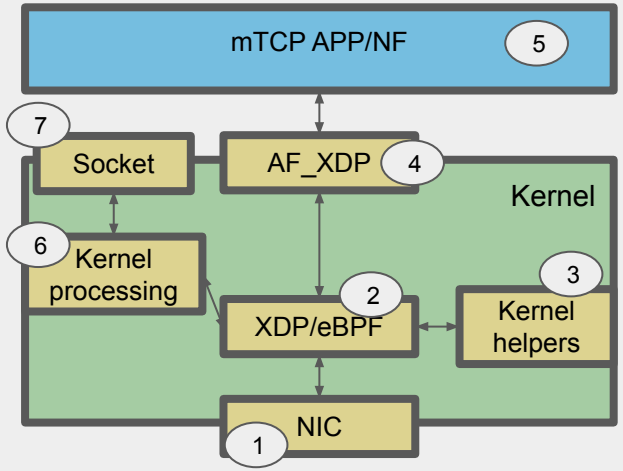

1) The packet arrives
2) eBPF code is executed
3) App can leverage kernel features
4) XDP sends packet to AF_XDP

# Life of a packet in the hybrid stack



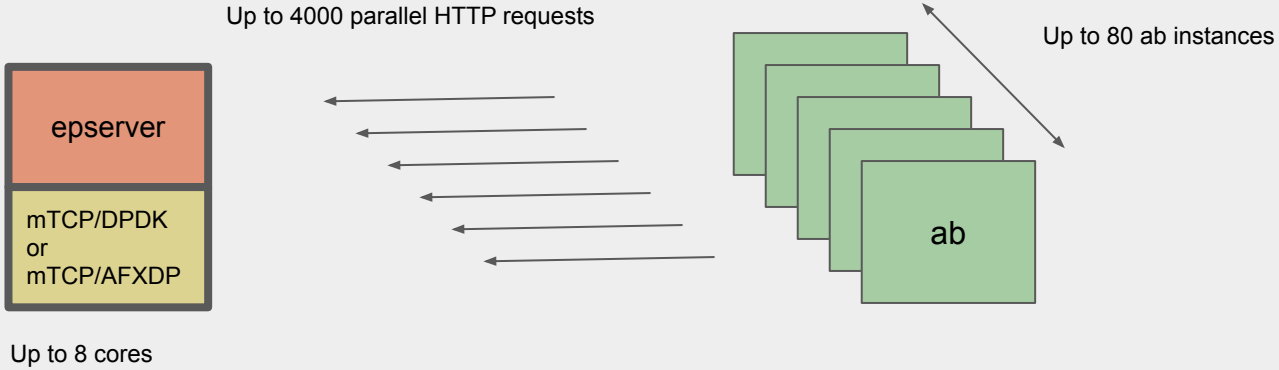

1) The packet arrives
2) eBPF code is executed
3) App can leverage kernel features
4) XDP sends packet to AF_XDP
5) mTCP app/NF thread produces/consumes packet

# Life of a packet in the hybrid stack

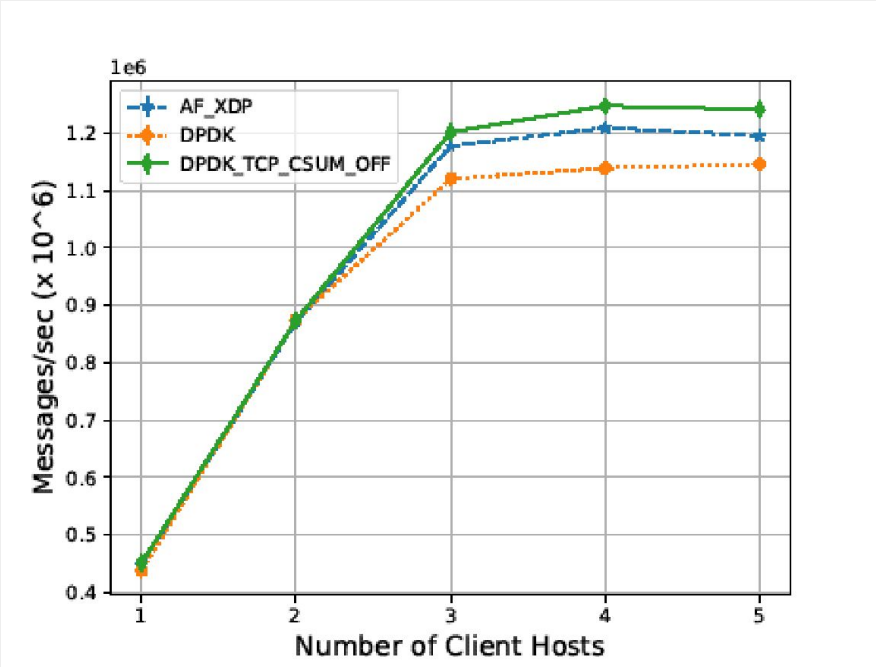


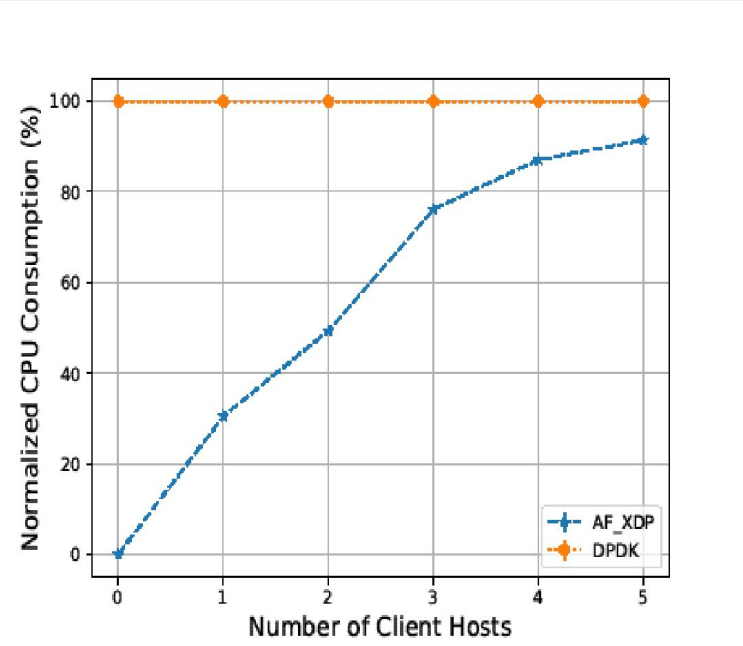

1) The packet arrives
2) eBPF code is executed
3) App can leverage kernel features
4) XDP sends packet to AF_XDP
5) mTCP app/NF thread produces/consumes packet
6) Packets can be sent to Kernel
7) Kernel based apps/NFs can produce/consume data/packets

# Evaluation

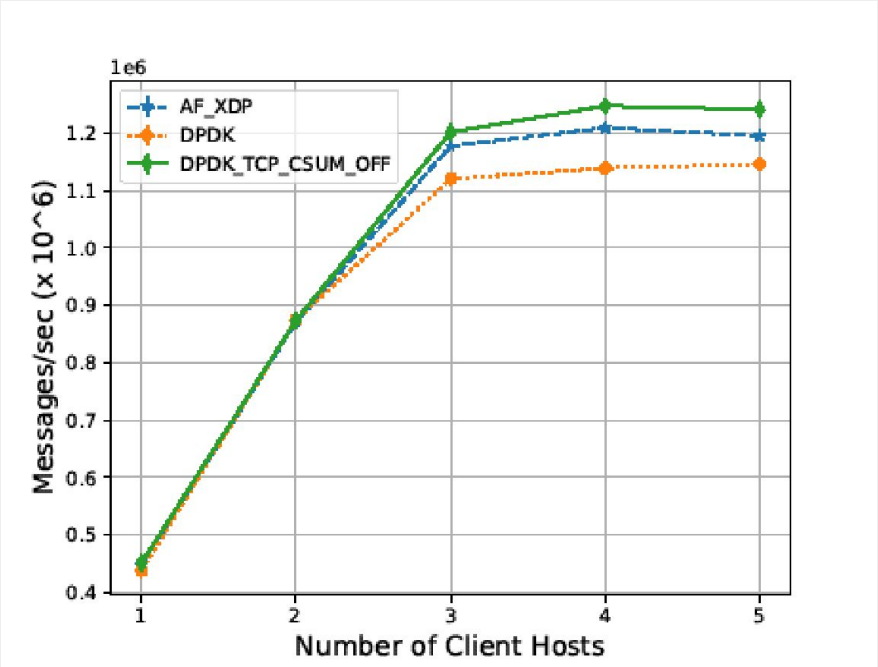

Up to 4000 parallel HTTP requests
Up to 80 ab instances
epserver
mTCP/DPDK
or
mTCP/AFXDP
ab
Up to 8 cores

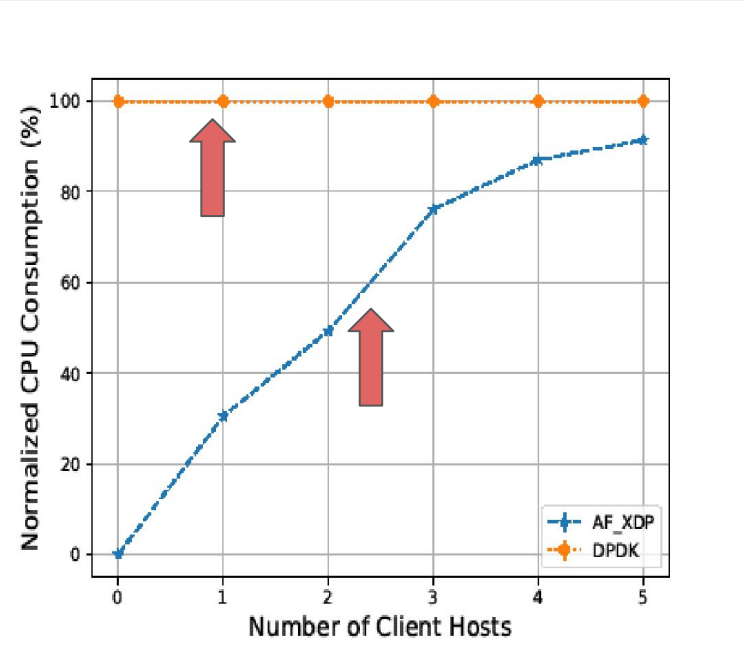
# CPU Efficiency



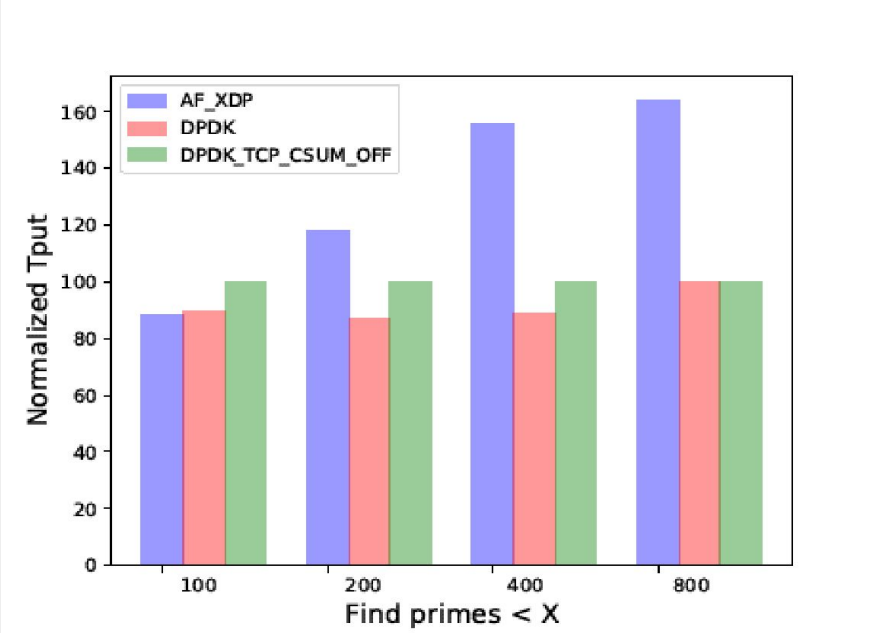



# CPU Efficiency

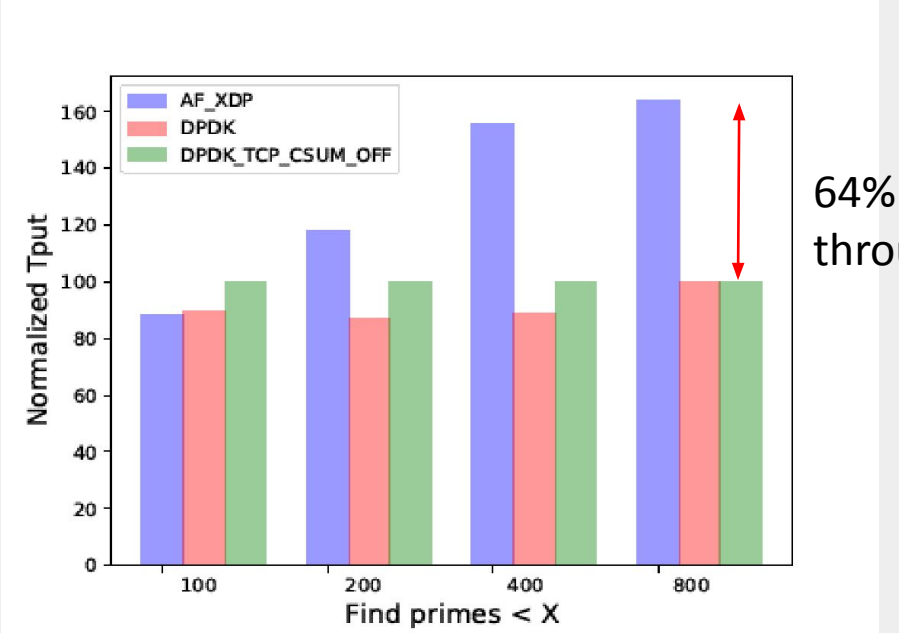


# CPU intensive workload



# CPU intensive workload



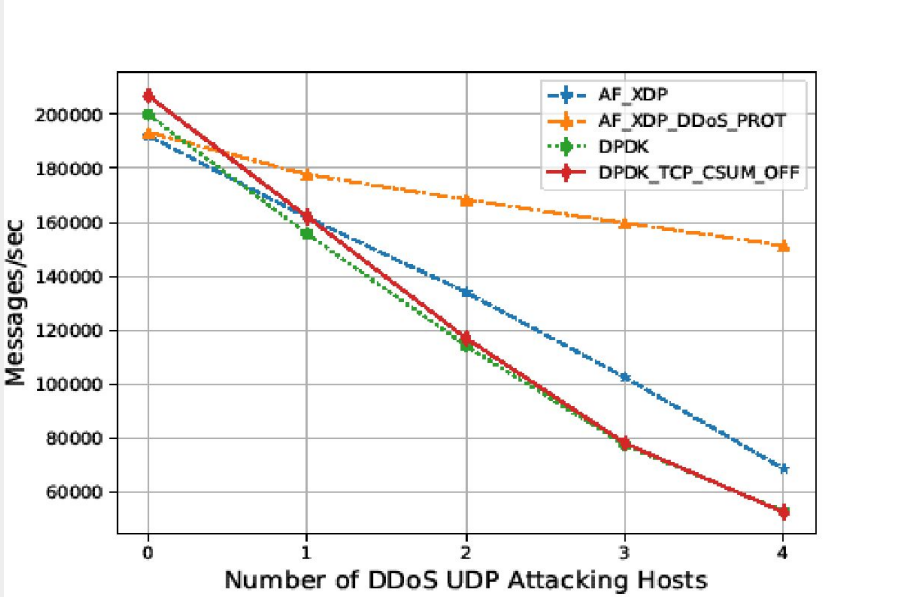

64% more throughput!

# DDoS Protection

4 of the 5 clients generate malicious UDP Traffic

1 client generates benign HTTP requests to the server

Sever runs on one core

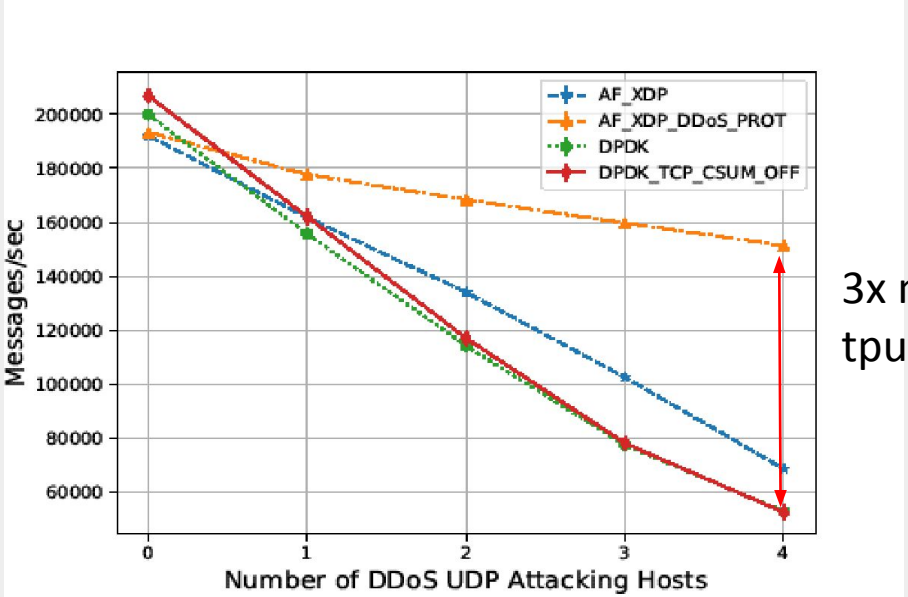


# DDoS Protection

4 of the 5 clients generate malicious UDP Traffic

1 client generates benign HTTP requests to the server

Sever runs on one core




3x more tput

# Contributions

New architecture for hybrid kernel/kernel bypass L4-L7 applications and network functions

We allowed a better CPU consumption profile for the userspace stack, making it more efficient and more suitable for running CPU intensive workloads

The hybrid processing allows cooperation mechanisms between high-performance TCP stack and kernel (e.g., DDoS protection)

# Contributions

New architecture for hybrid kernel/kernel bypass L4-L7 applications and network functions

We allowed a better CPU consumption profile for the userspace stack, making it more efficient and more suitable for running CPU intensive workloads

The hybrid processing allows cooperation mechanisms between high-performance TCP stack and kernel (e.g., DDoS protection)

Publication - "**A Userspace TCP Stack doesn't Have to Mean Losing Linux Processing**" IEEE NFV-SDN 2020

# Contribution II

High-coverage monitoring

# Overview

**Networks need monitoring**

Security issues

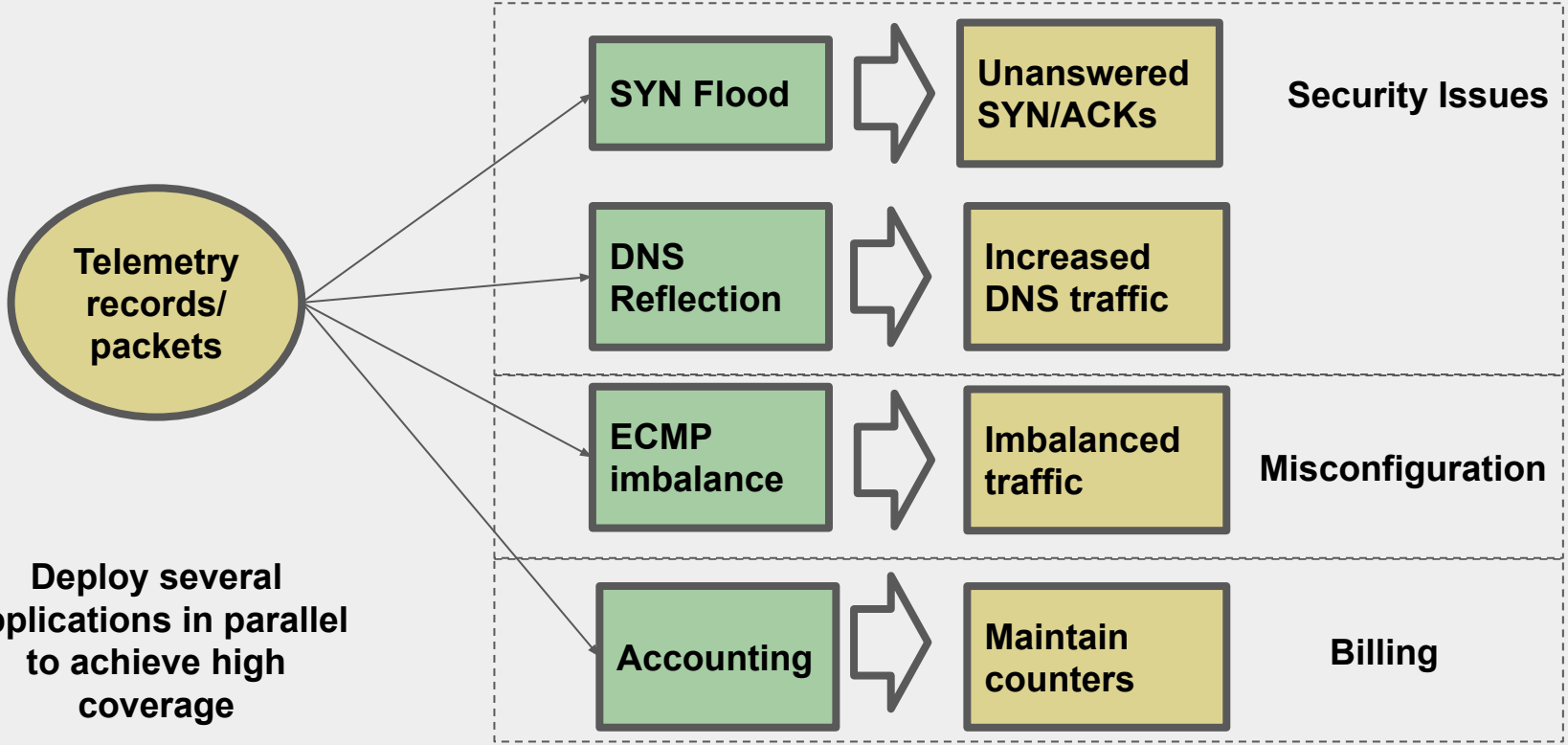
Performance issues

Traffic accounting

Misconfiguration

**Get insights to perform management actions**

# Need high-coverage monitoring




Deploy several applications in parallel to achieve high coverage

# Challenges with previous solutions

Several apps in parallel

High packet rates

Kernel bypass

**Inefficient**

Offload to network

Use efficient data path

# Challenges with previous solutions

Several apps in parallel

High packet rates

Kernel bypass

**Inefficient**

Offload to network

**Still needs software**

Use efficient data path

# Challenges with previous solutions

Several apps in parallel

High packet rates

Kernel bypass

**Inefficient**

Offload to network

**Still needs software**

Use efficient data path

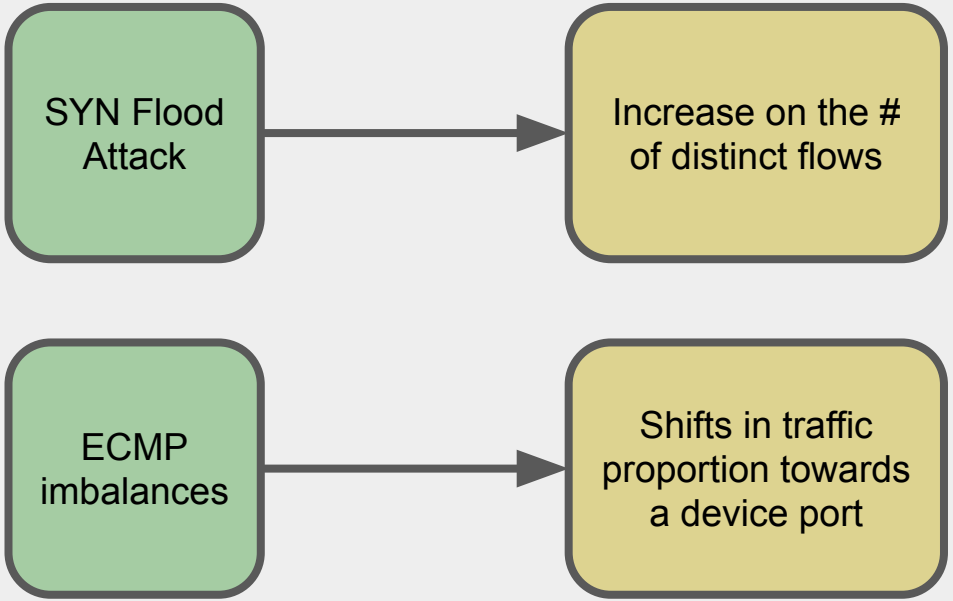
**Still processes too many packets**

# Key opportunities

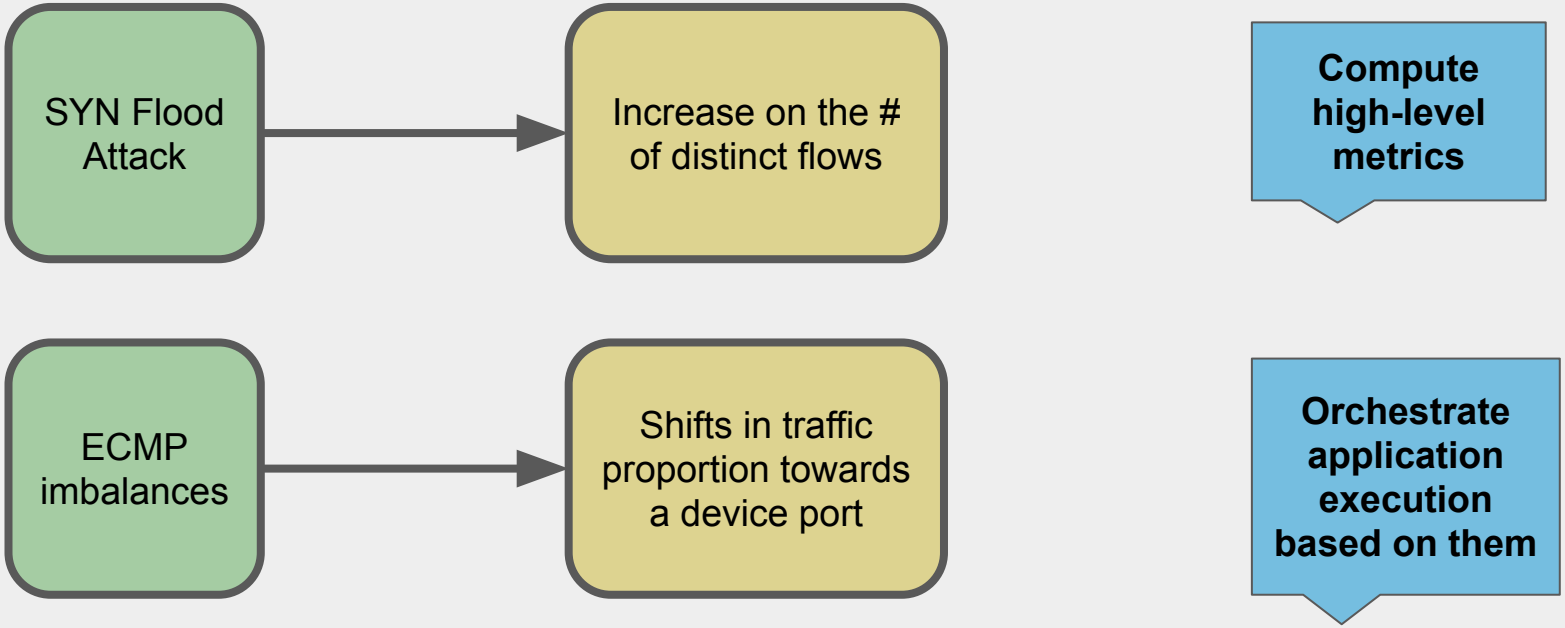
Only execute applications when needed

Consolidate tasks needed by all monitoring applications

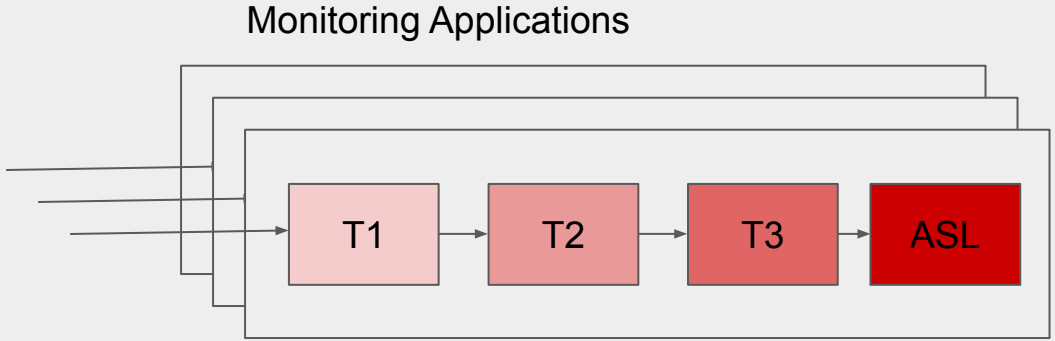
# Only execute applications when needed



# Only execute applications when needed



# Consolidate tasks needed by all applications



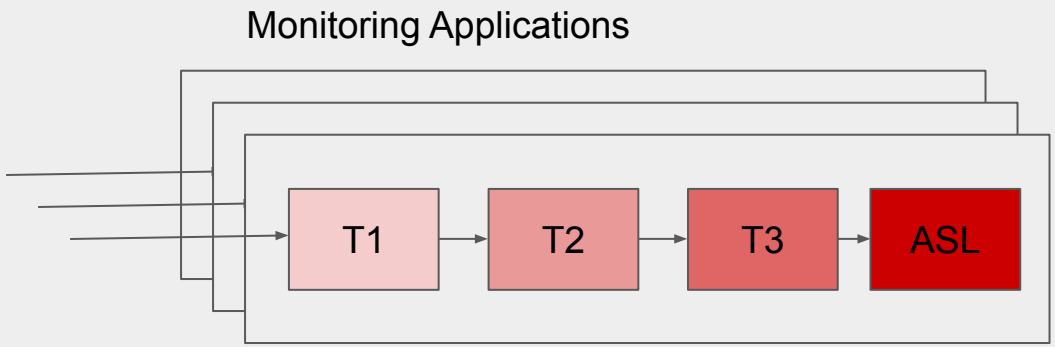

T1 -> Ingest, parse and select packets of interest

T2 -> Compute relevant metrics for the application

T3 -> If conditions are met, send packet for application processing

ASL -> Execute application specific logic, detect conditions and generate events

# Consolidate tasks needed by all applications




**Redundant logic!**
**All apps process all packets!**

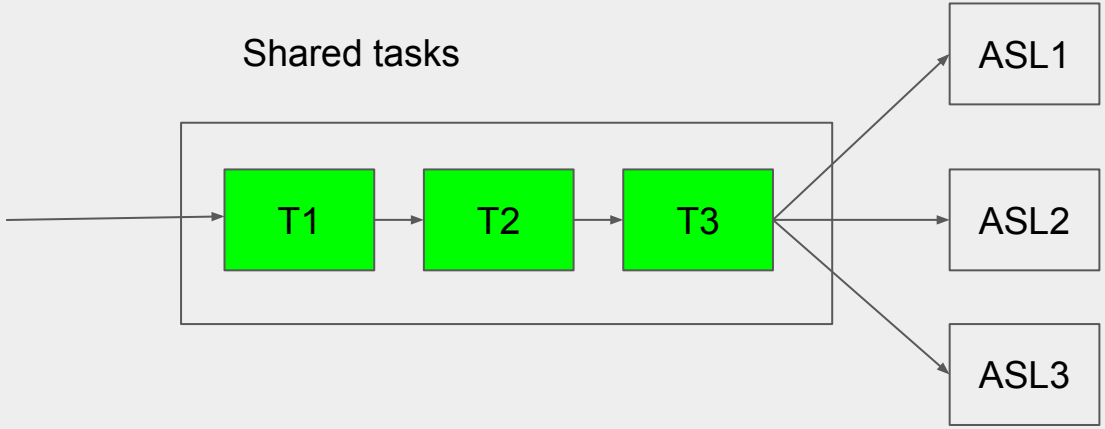
T1 -> Ingest, parse and select packets of interest

T2 -> Compute relevant metrics for the application

T3 -> If conditions are met, send packet for application processing

ASL -> Execute application specific logic, detect conditions and generate events

# Consolidate tasks in a shared novel primitive needed by all applications




No redundant processing!

Small resource footprint!

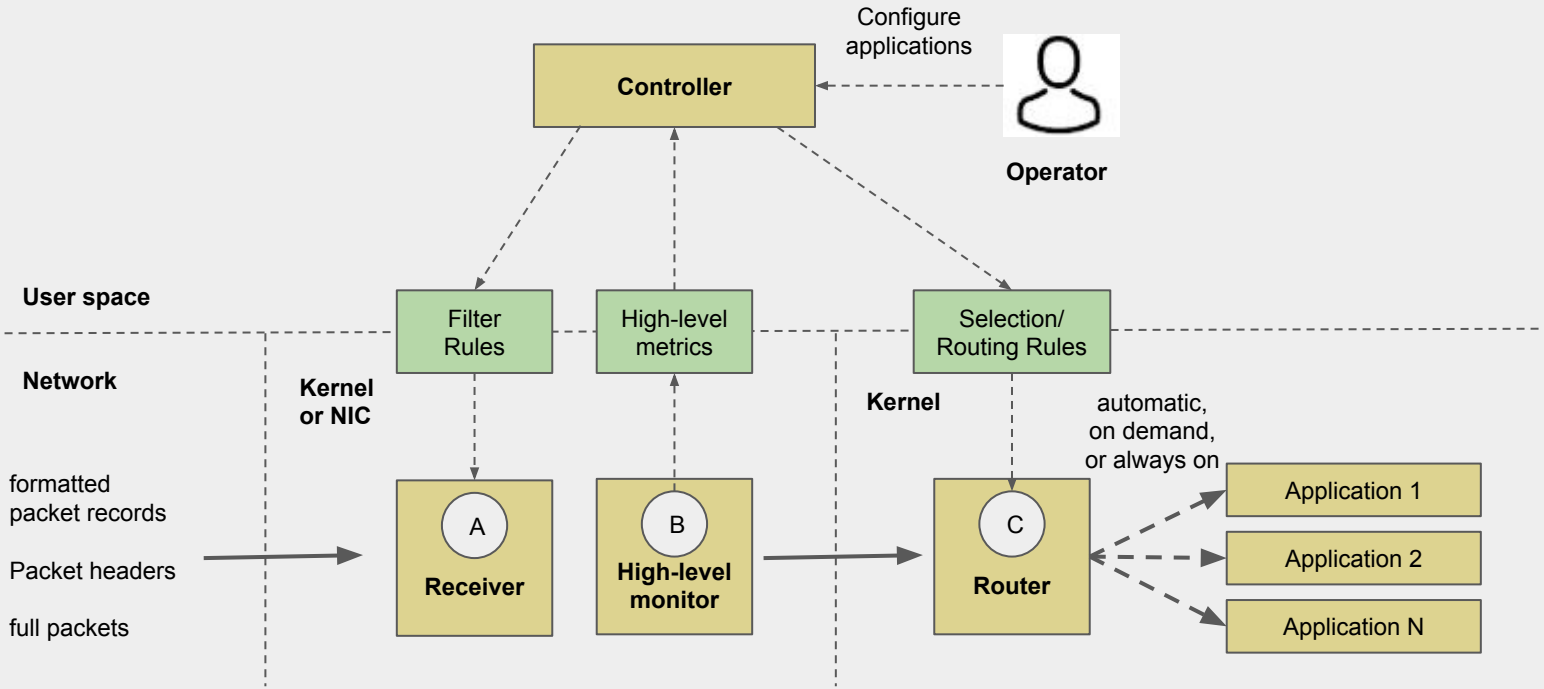
Developers can write slimmer applications!

# What orchestration and task consolidation enables?

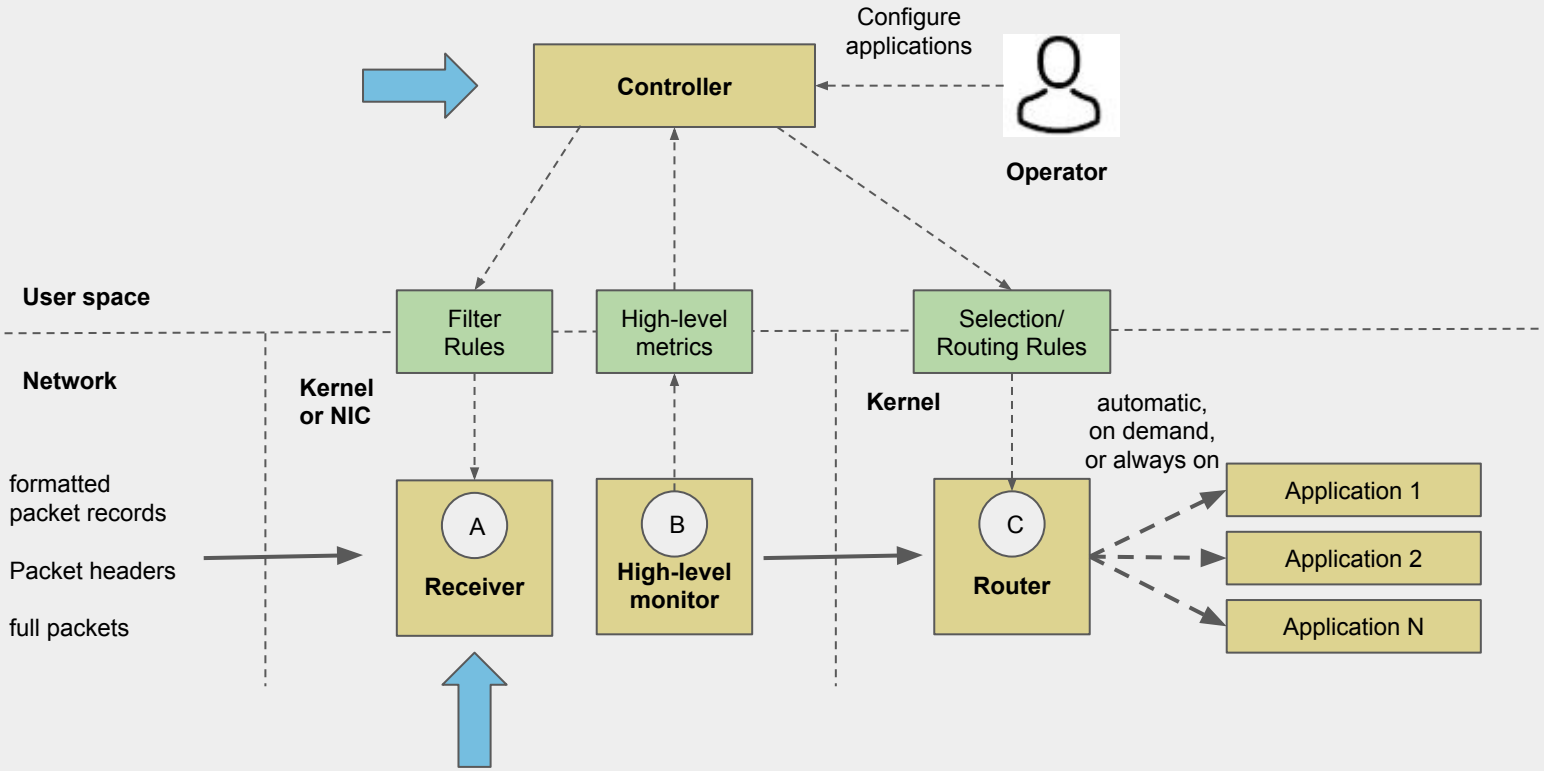
Use processing power to obtain knowledge about critical issues quicker

Avoid extra delays due to in-band monitoring

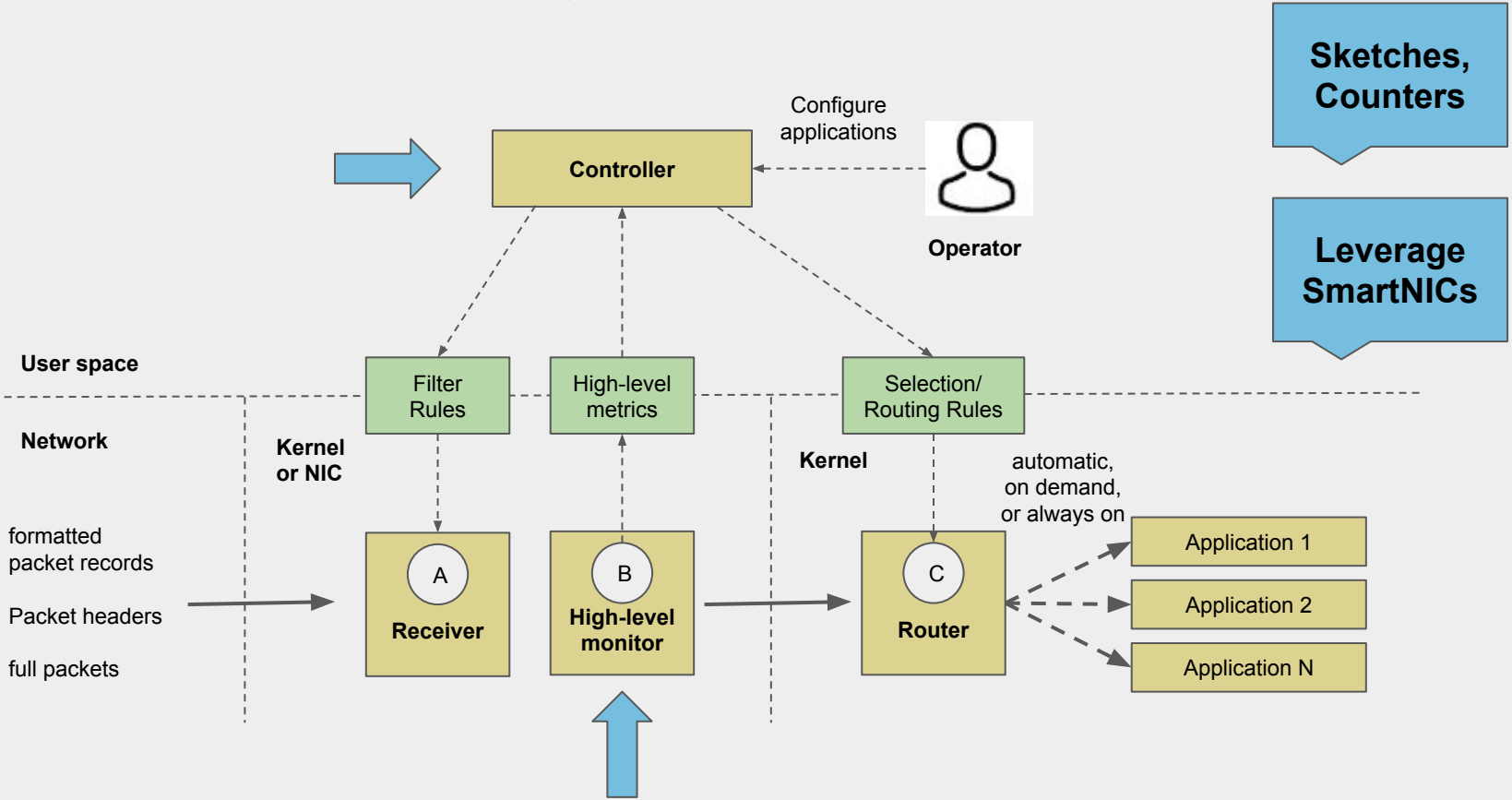
# Our system architecture



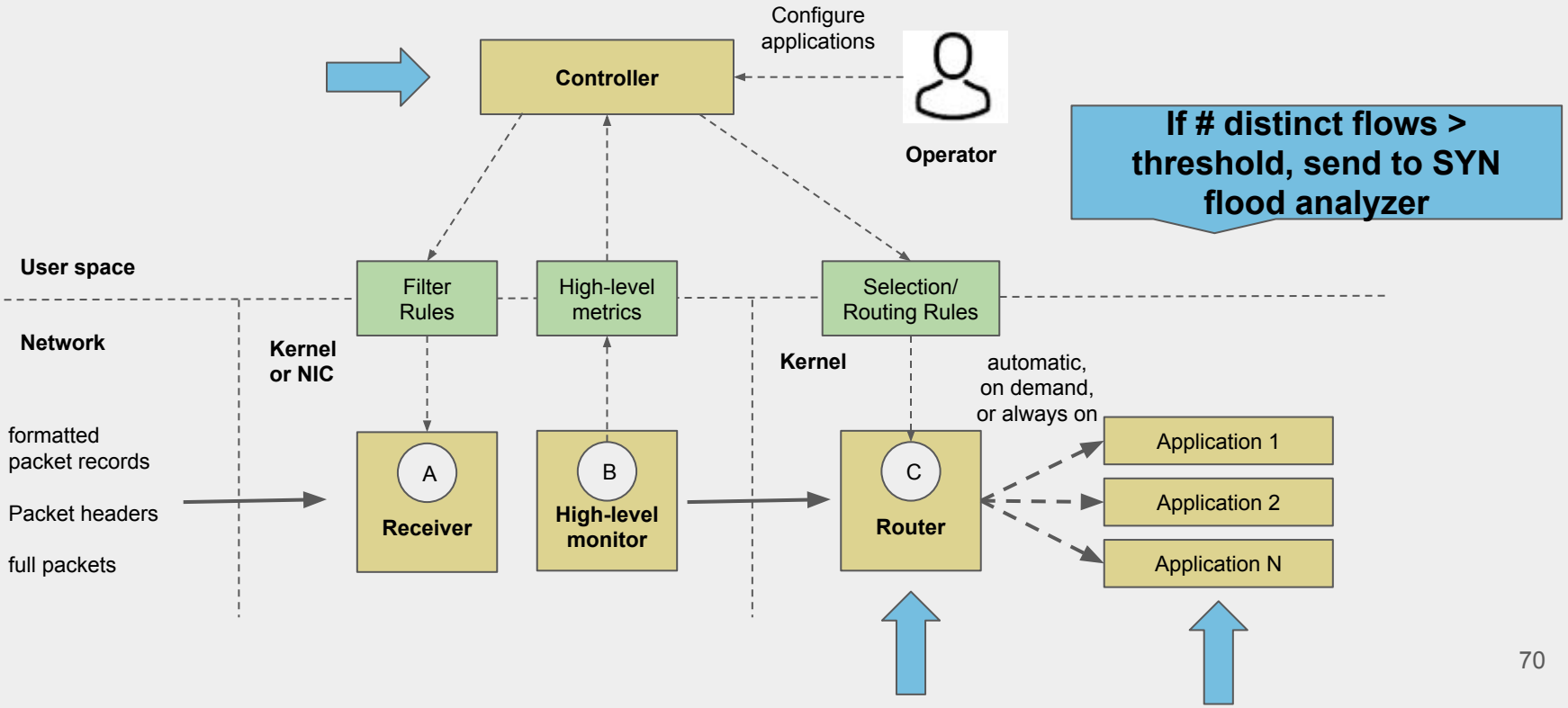
# Our system architecture



# Our system architecture



# Our system architecture



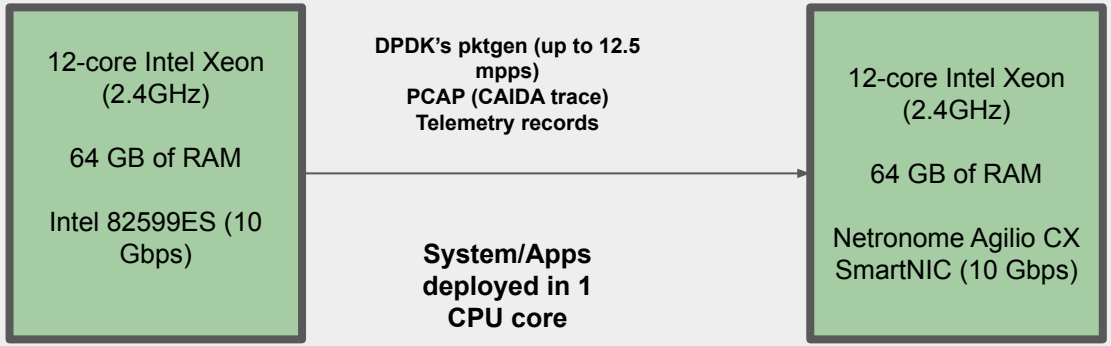
# Example applications

| | |
|---|---|
| Traffic accounting | Counts # of packets and bytes per IP destination |
| Half open TCP connections analyzer | Tracks incomplete TCP handshakes |
| DNS flow analyzer | Maintains per flow statistics of DNS traffic |

# Evaluation

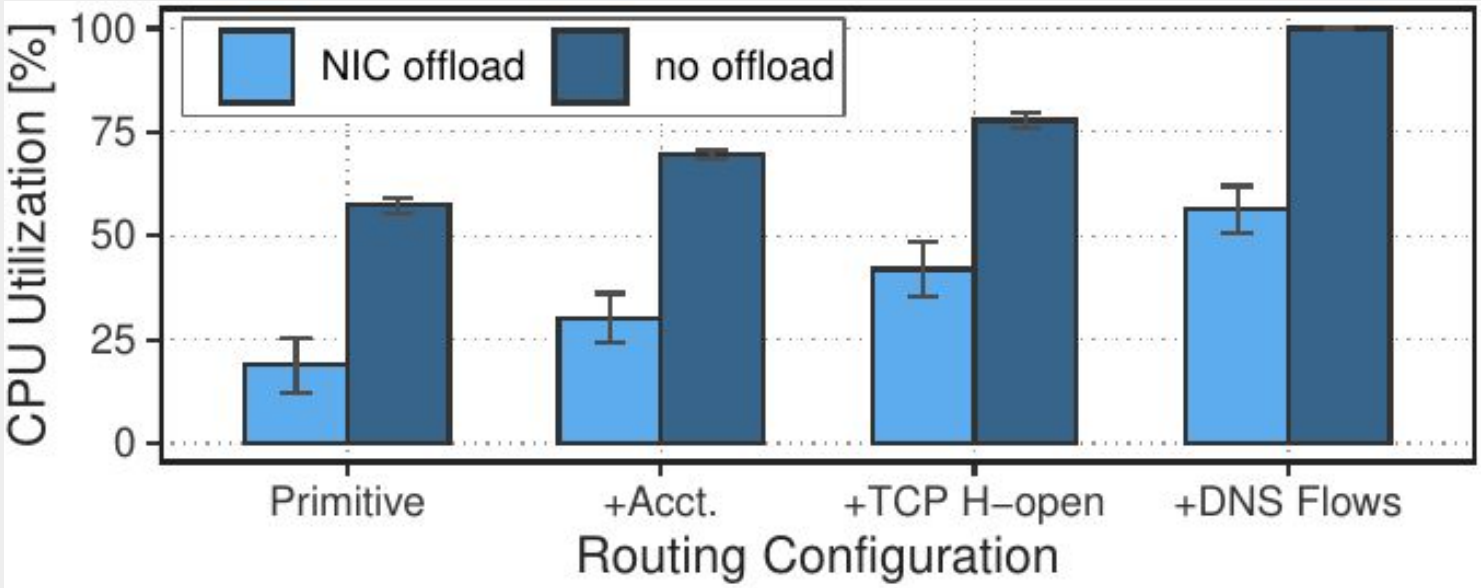
CPU efficiency

Scalability



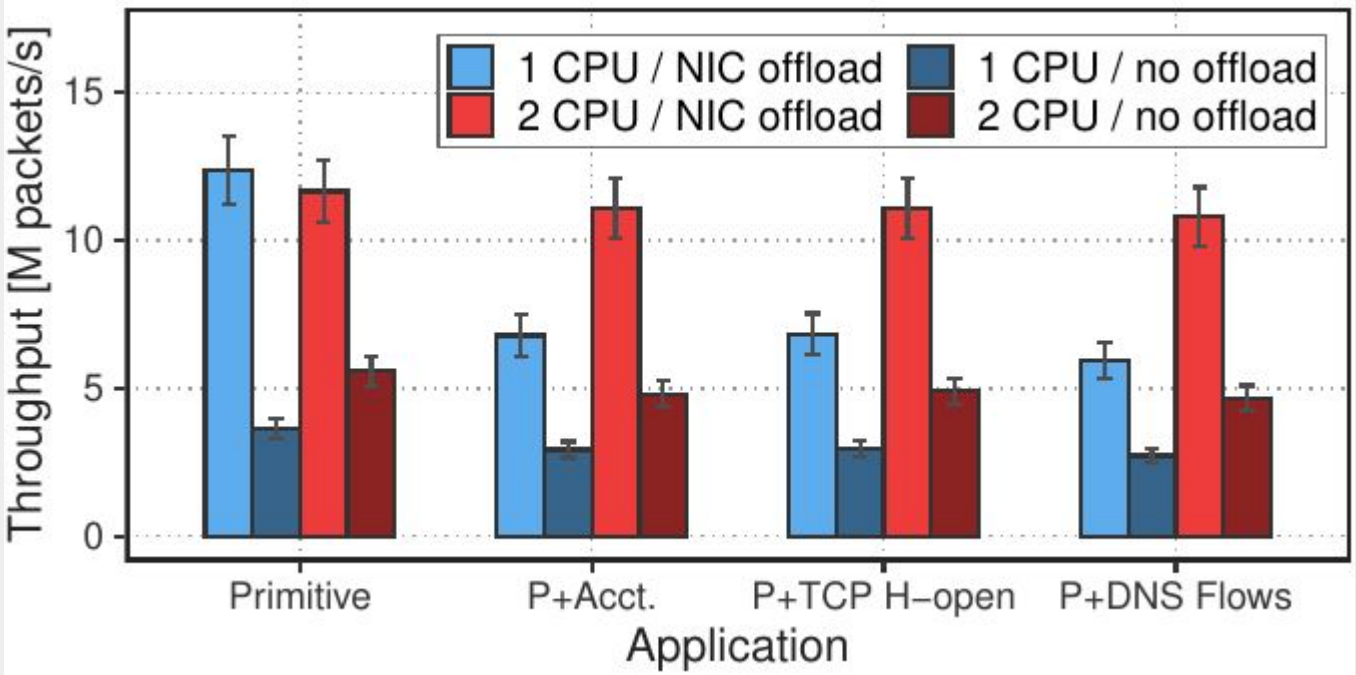
# Efficiency in CPU utilization

**Offered load 2 mpps**



# Scalability as we add resources

**Offered load 12.5 mpps**



# Contributions

Consolidate monitoring tasks in novel monitoring primitives

Dynamic orchestration of monitoring applications based on high-level metrics

Focus on monitoring what is important improving monitoring  and avoiding performance impacts on the network

# Contributions

Consolidate monitoring tasks in novel monitoring primitives

Dynamic orchestration of monitoring applications based on high-level metrics

Focus on monitoring what is important improving monitoring  and avoiding performance impacts on the network

Publication - “**Efficient Network Monitoring Applications in the Kernel with eBPF and XDP**” IEEE NFV-SDN 2021

Best paper award!

# Contribution III

L2-L4 performance and feature richness

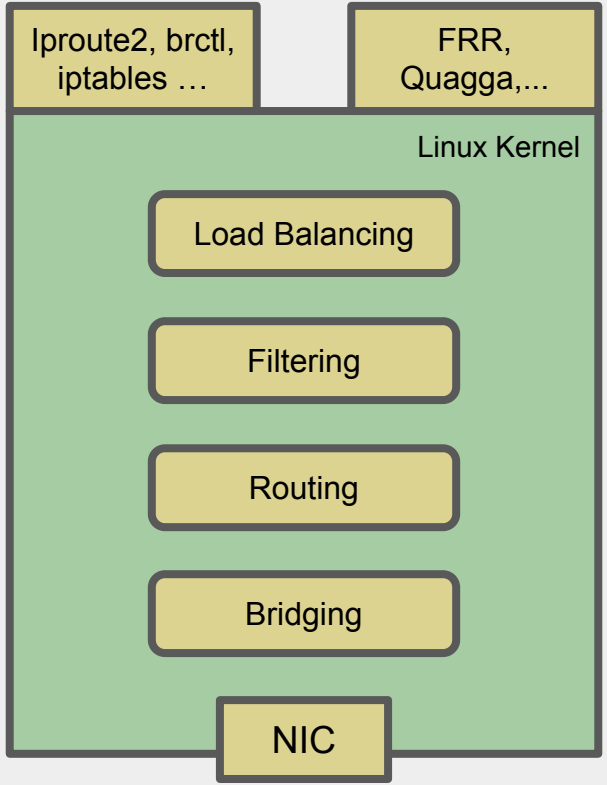
# Linux Ecosystem

Interconnectivity

Performance

Scalability

Security



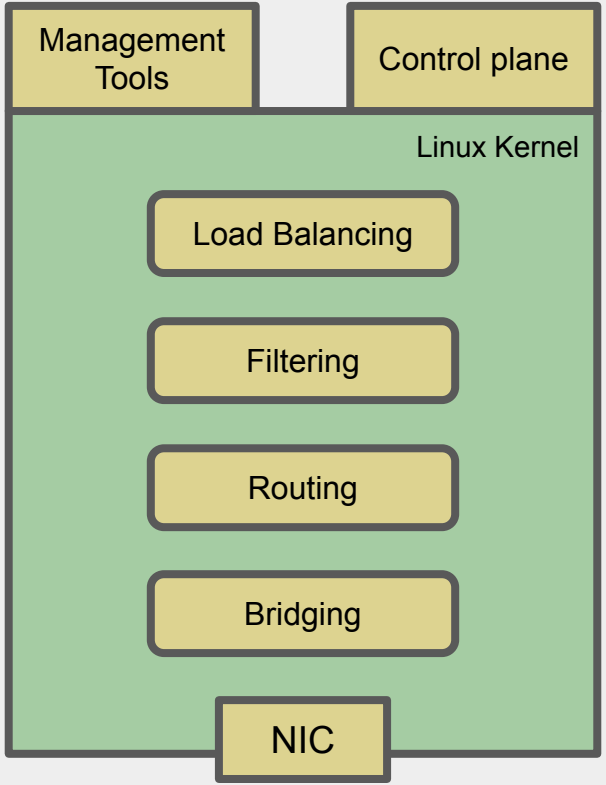
# Linux Ecosystem

Interconnectivity

Performance

Scalability

Security




Heavyweight data structures

Heavyweight protocol implementations

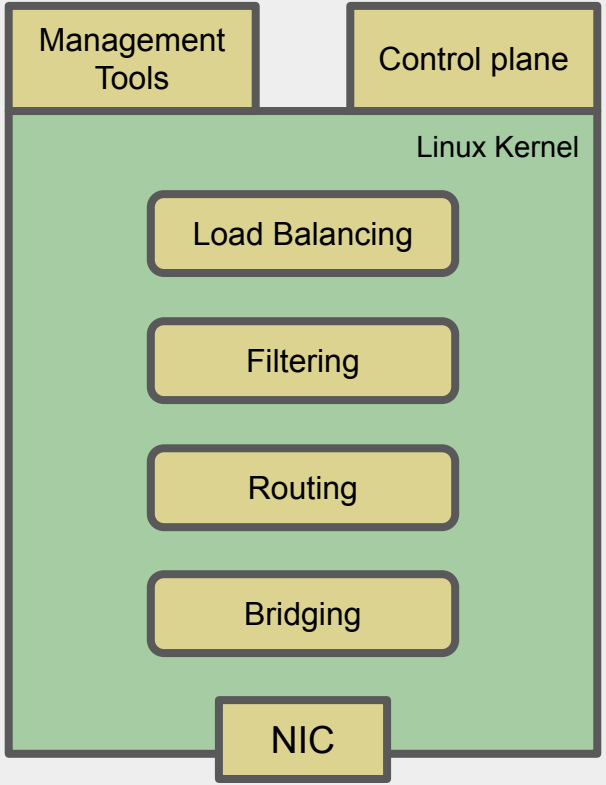
# Linux Ecosystem

Interconnectivity

Performance

Scalability

Security



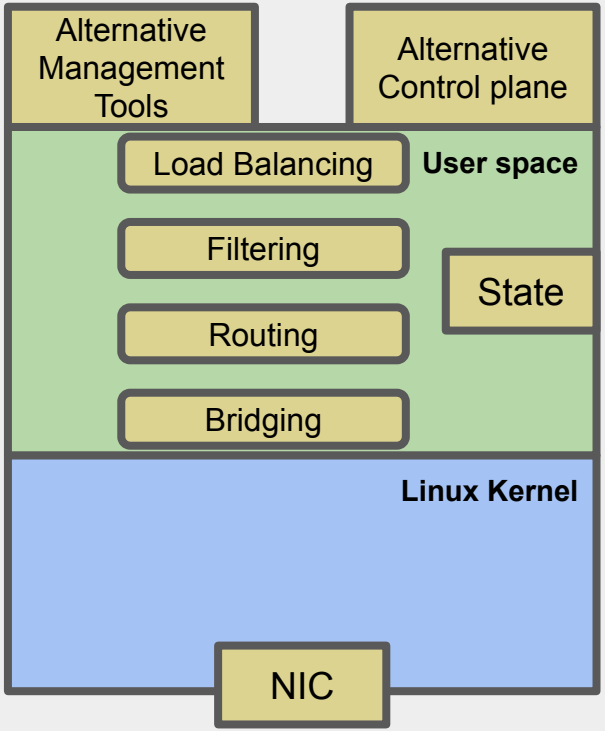

Heavyweight data structures

Heavyweight protocol implementations

**Long critical path**

**Poor performance**

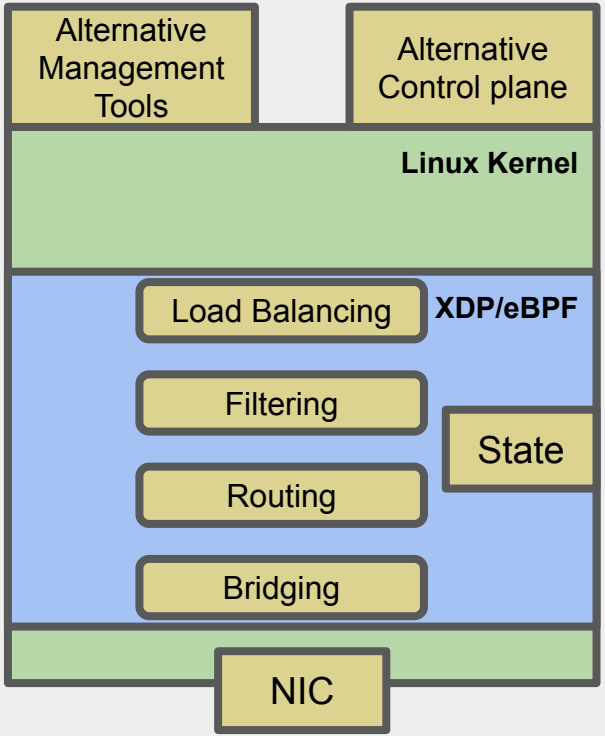
# Alternative processing pipelines




Kernel bypass

Reimplement several NFs in user space

# Alternative processing pipelines




In kernel network stack bypass

Reimplement several NFs in eBPF

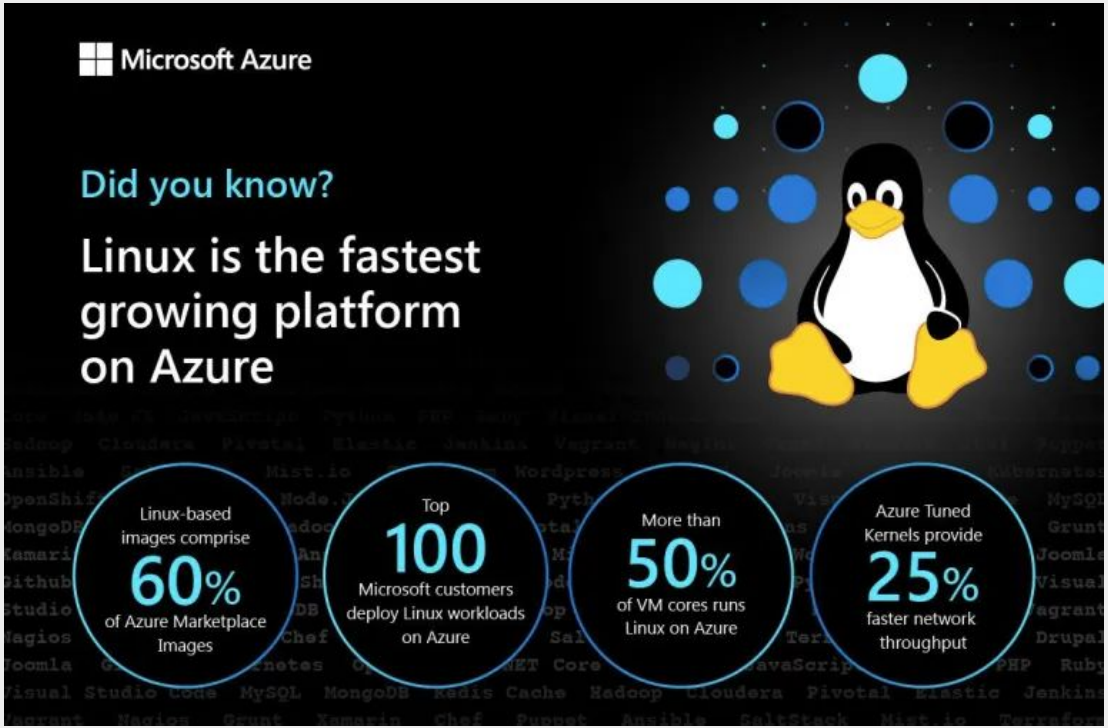
# Alternative processing pipelines

**Lose the opportunity to leverage Linux’s rich ecosystem**

# Our Approach

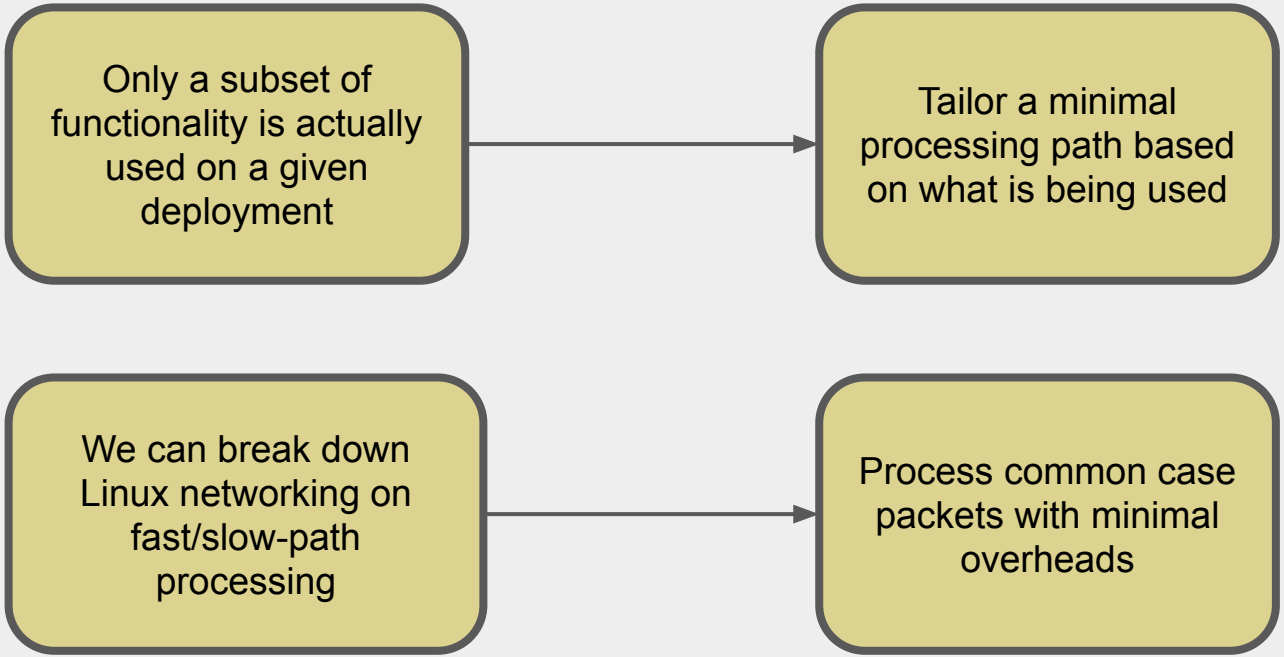
Redesign the Linux network stack to address its shortcomings

# Why is this important?

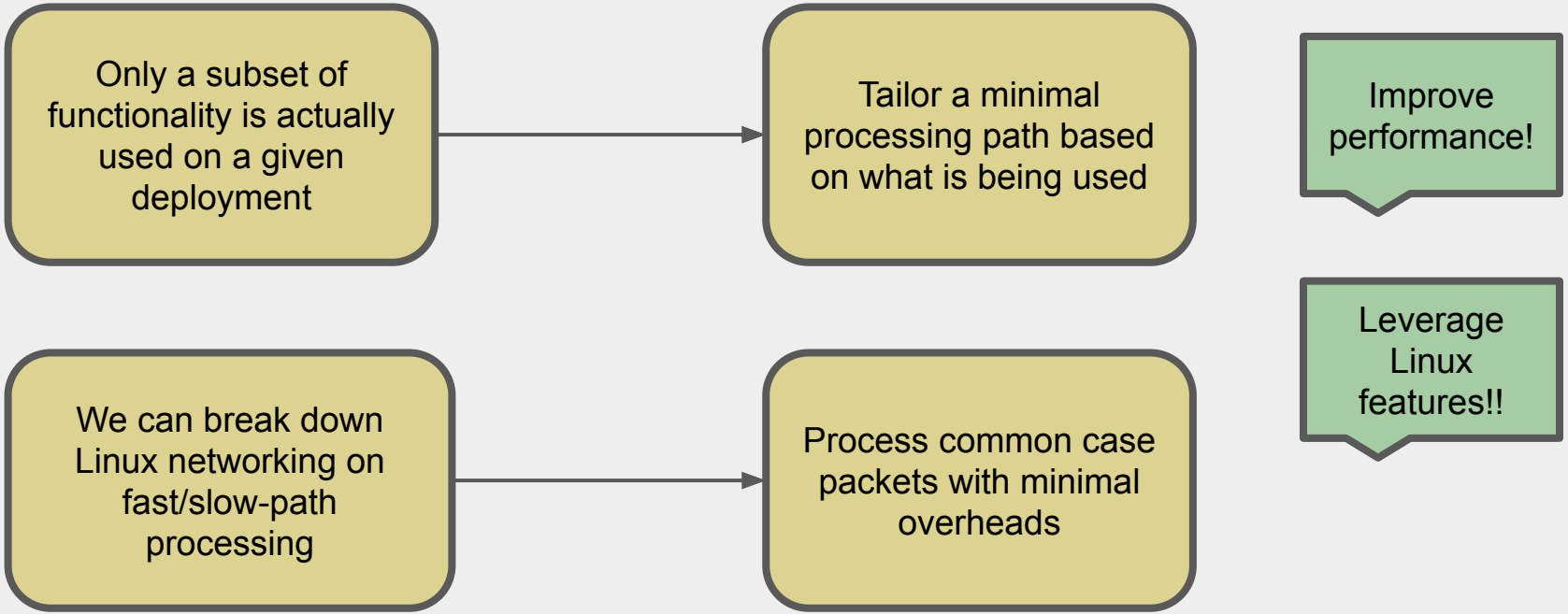



Source: https://build5nines.com/linux-is-most-used-os-in-microsoft-azure-over-50-percent-fo-vm-cores/

# Two Key Insights and Opportunities



# Two Key Insights and Opportunities



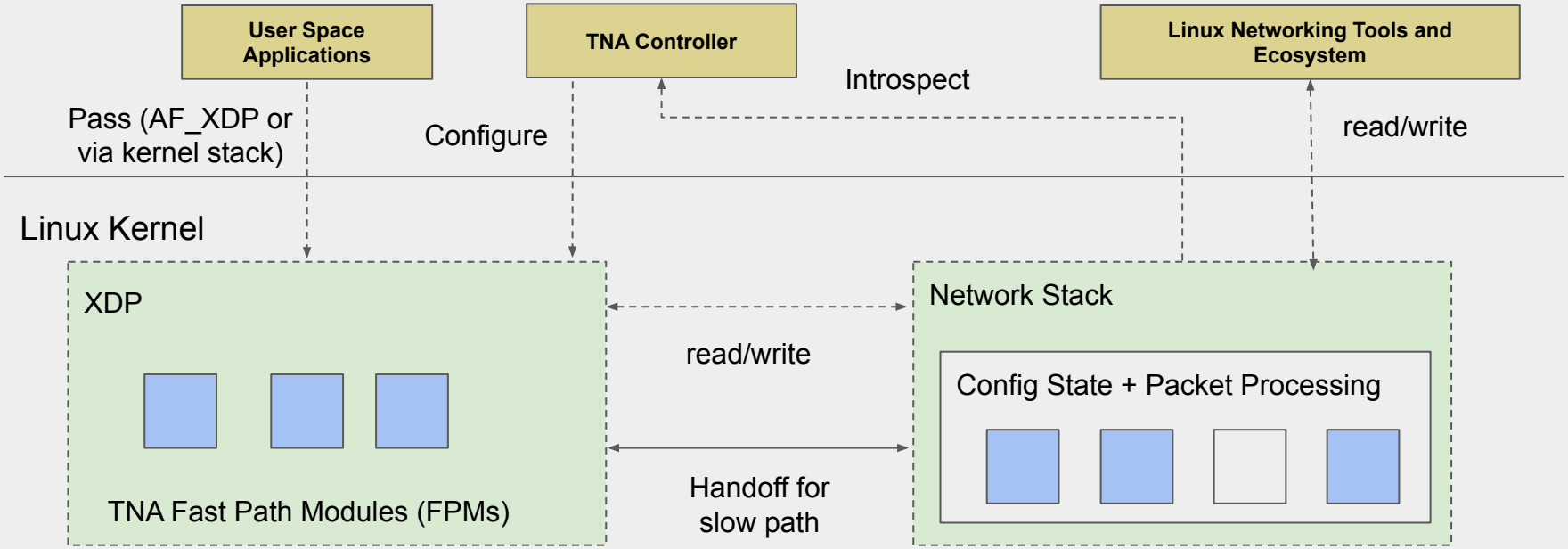
# Challenges

How to break down Linux network into slow/fast path processing?

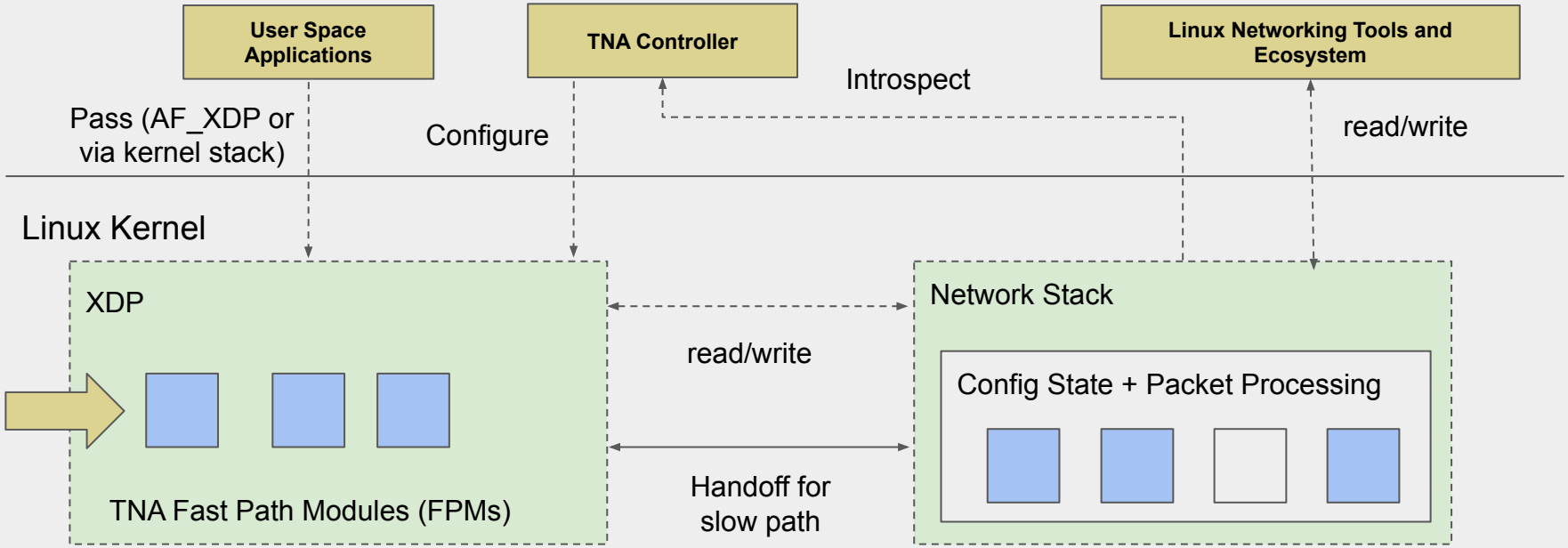
How to dynamically instantiate a minimal fast path to support what is being used?

How to make this transparent to the system?

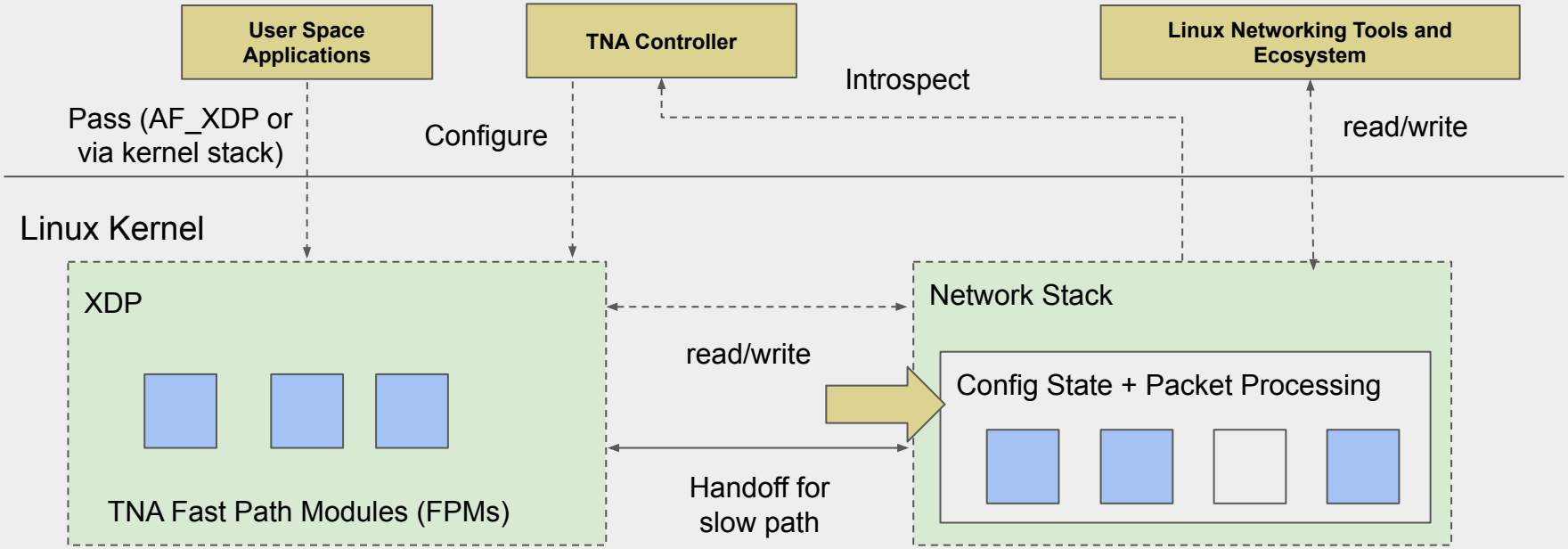
# Introducing TNA



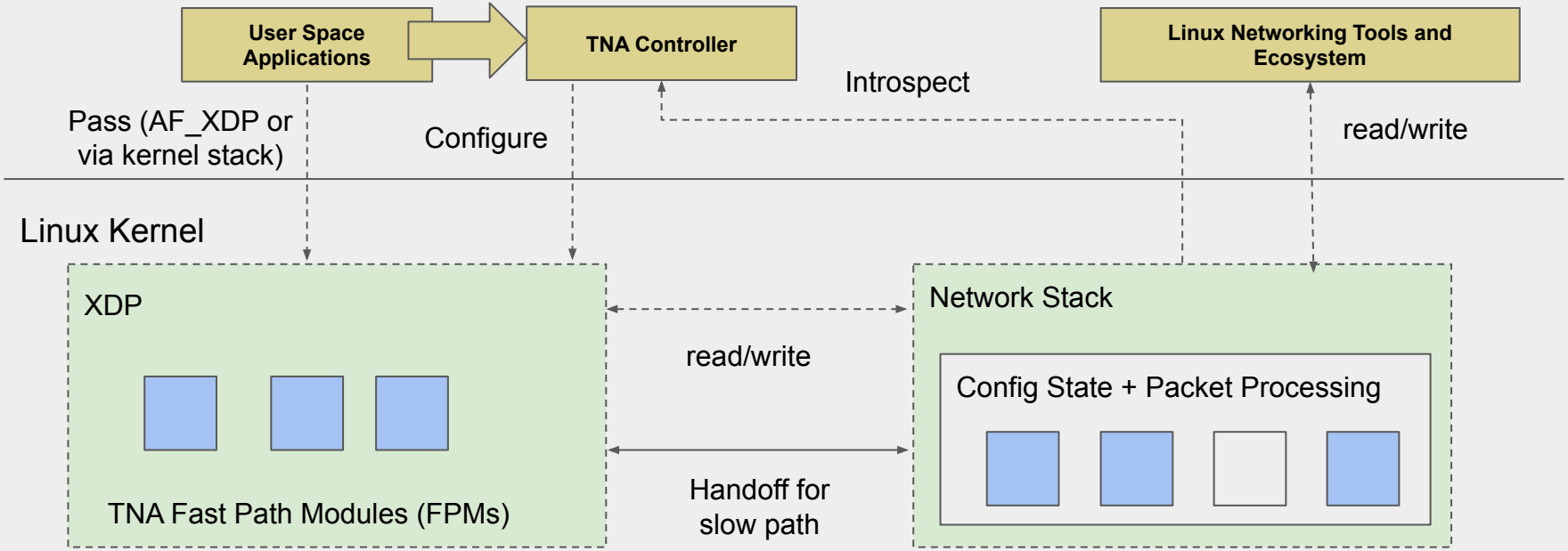
# Introducing TNA



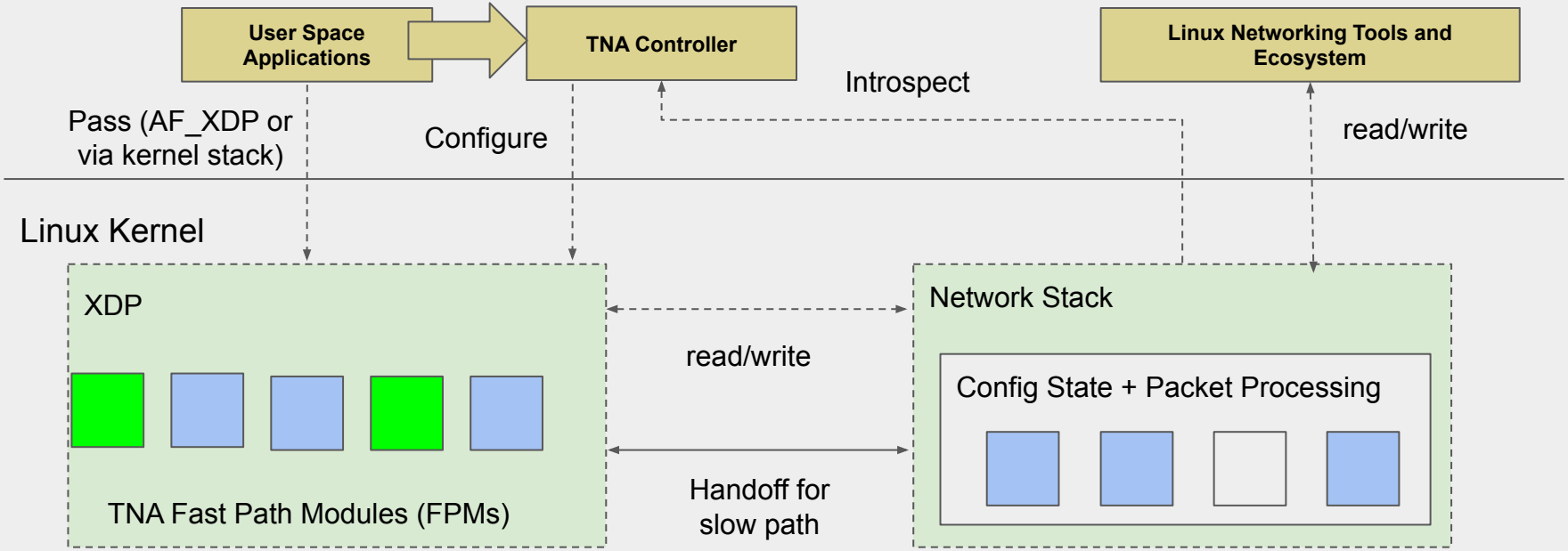
# Introducing TNA



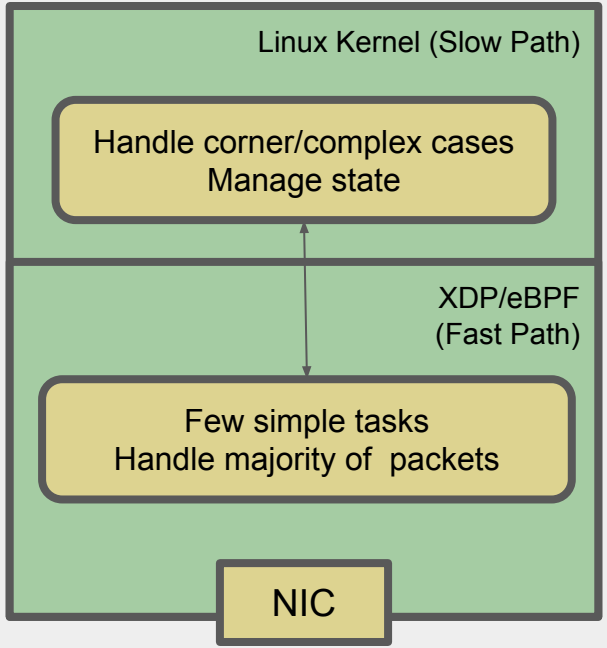
# Introducing TNA



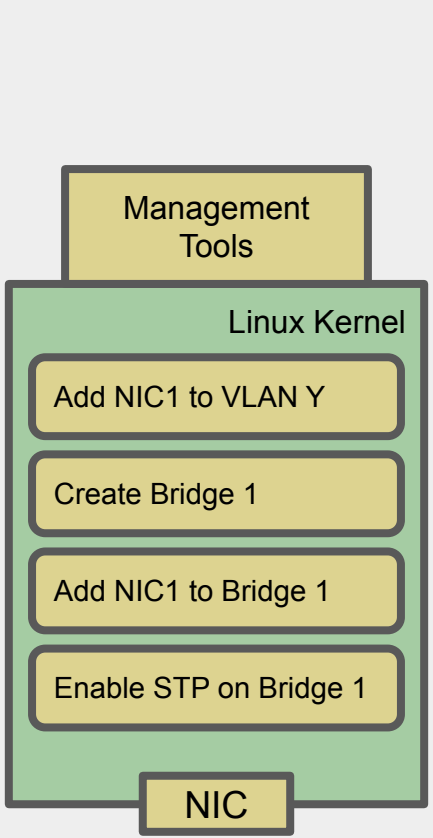
# Introducing TNA



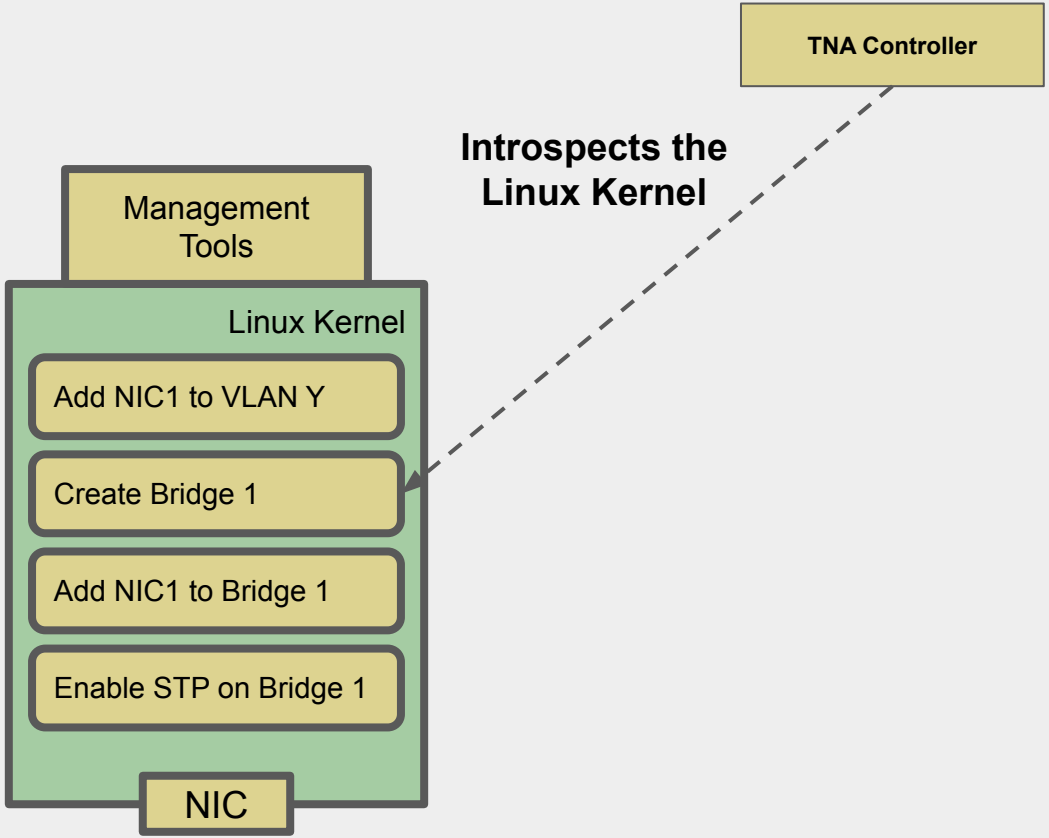
# Designing fast-path modules (FPMs)



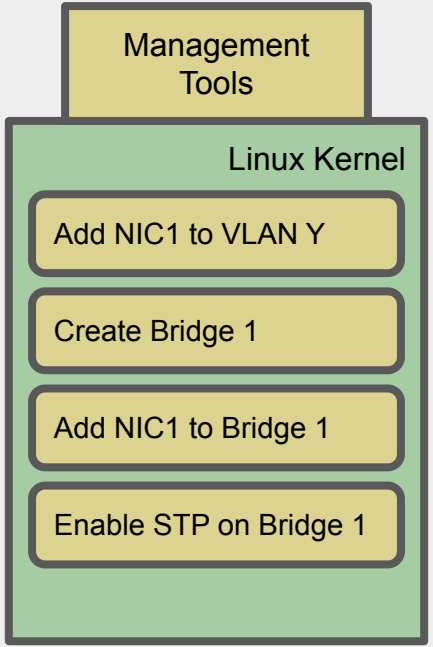
# Automated Fast-Path Data plane Creation

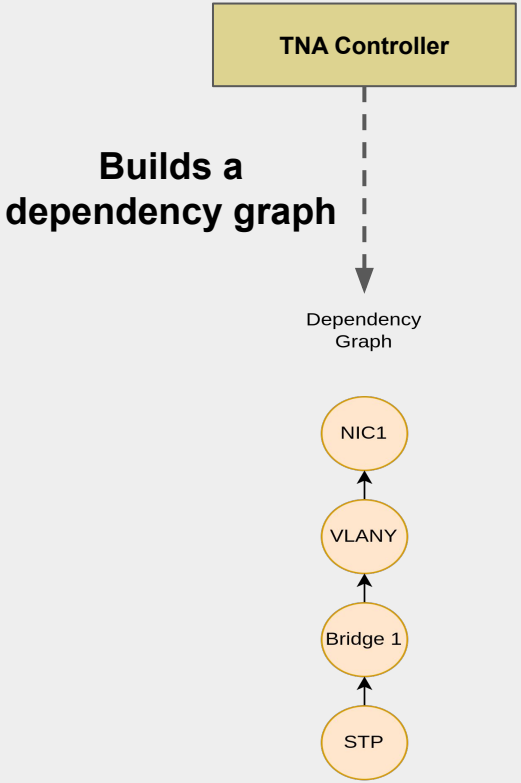


# Automated Fast-Path Data plane Creation

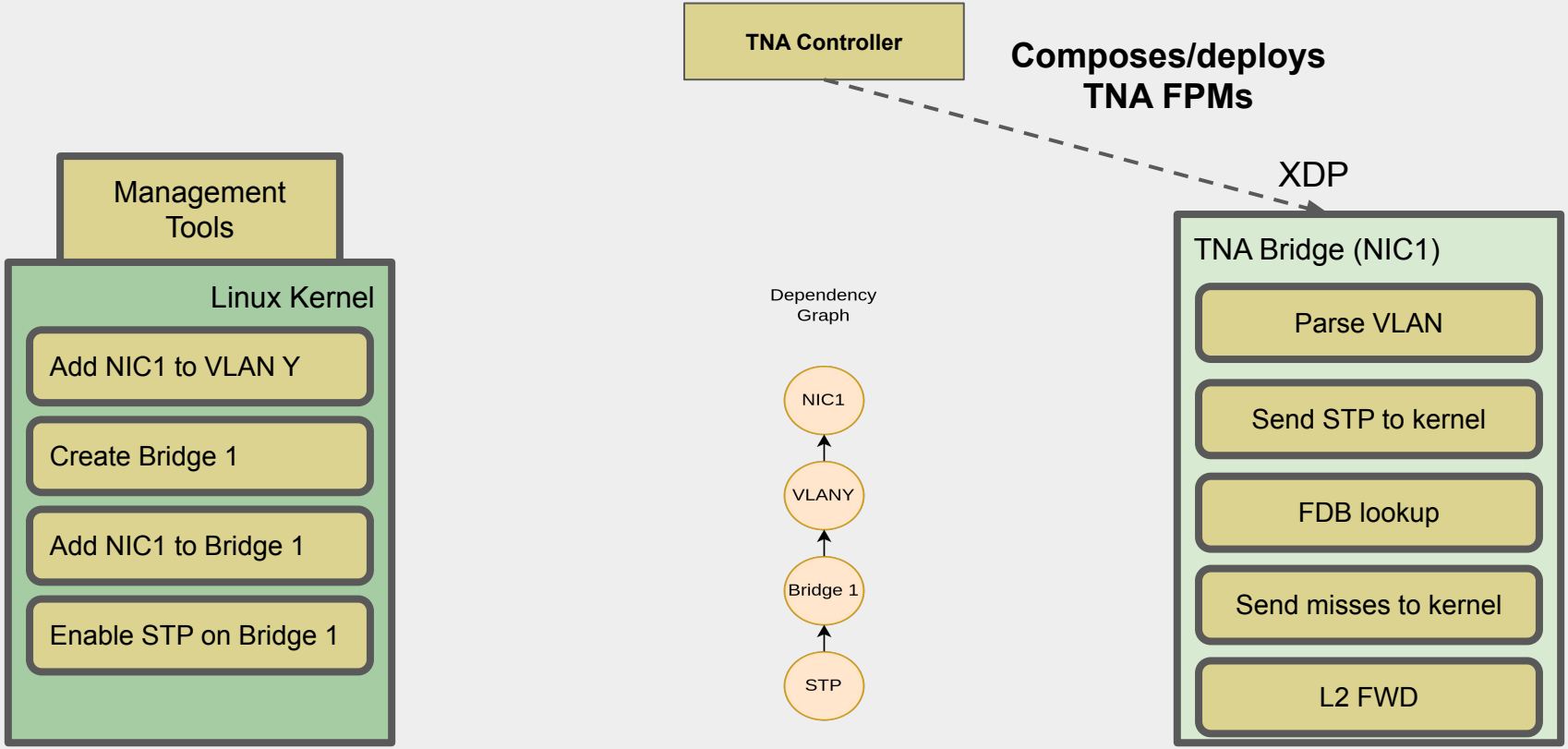


# Automated Fast-Path Data plane Creation



# Automated Fast-Path Data plane Creation



# Example Applications

Bridging

Routing

iptables Filtering

FPMs

Helpers

Controller code

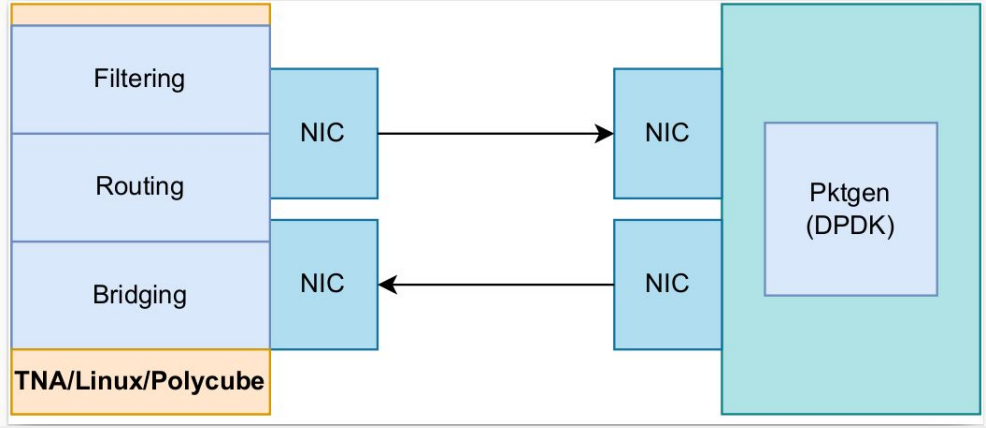
# Evaluation

Scalability

Throughput

Up to 6 cores

10 Gbps NICs

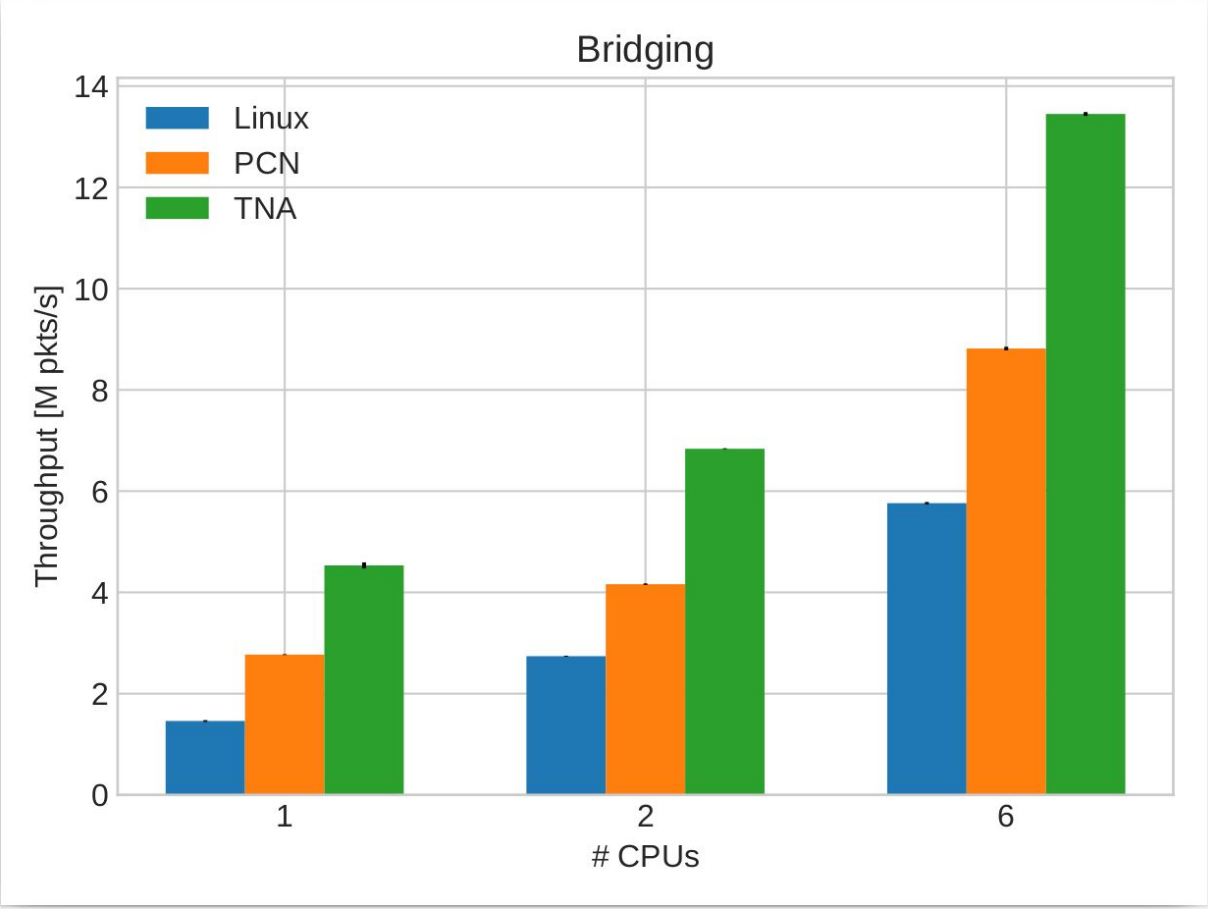


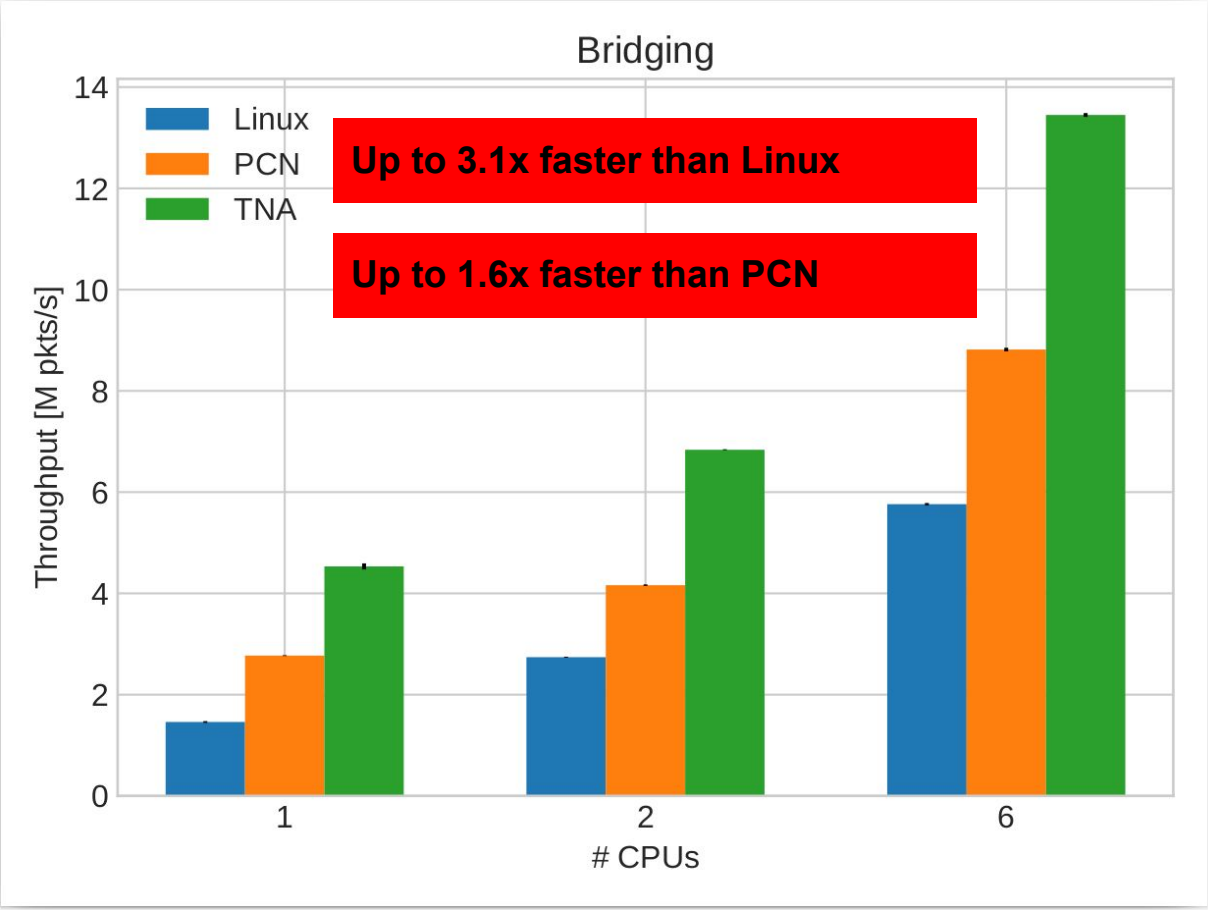

1 core generates

64B packets at
15 mpps

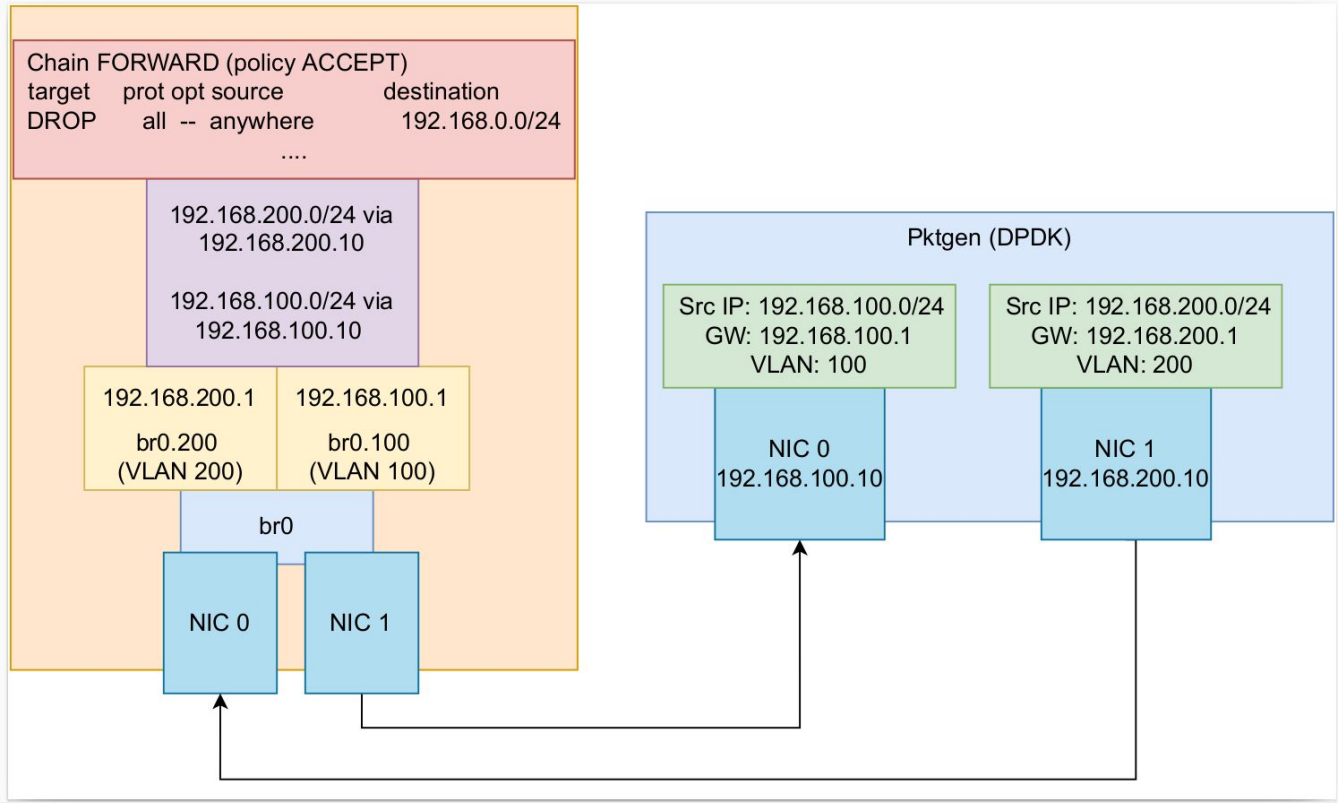
1 core receives

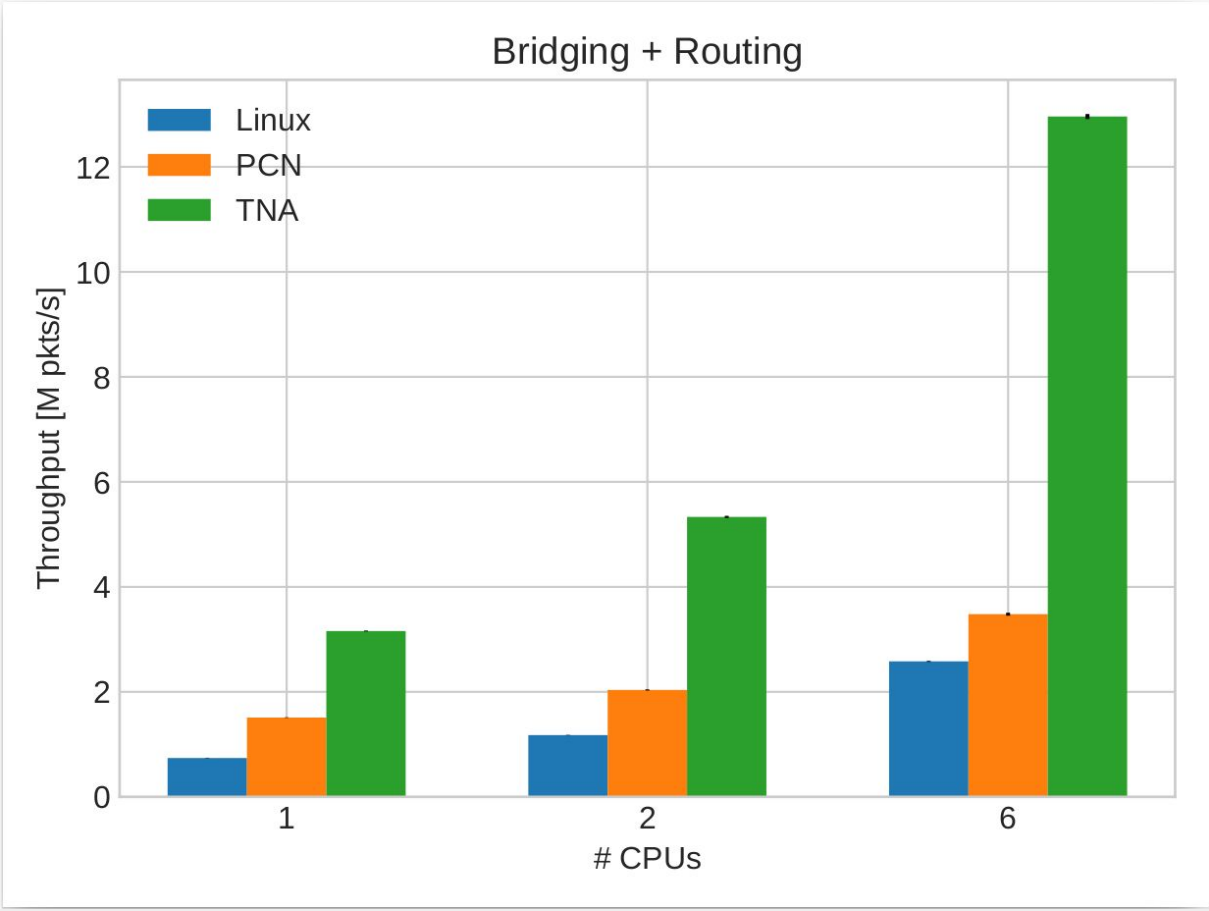
10 Gbps NICs

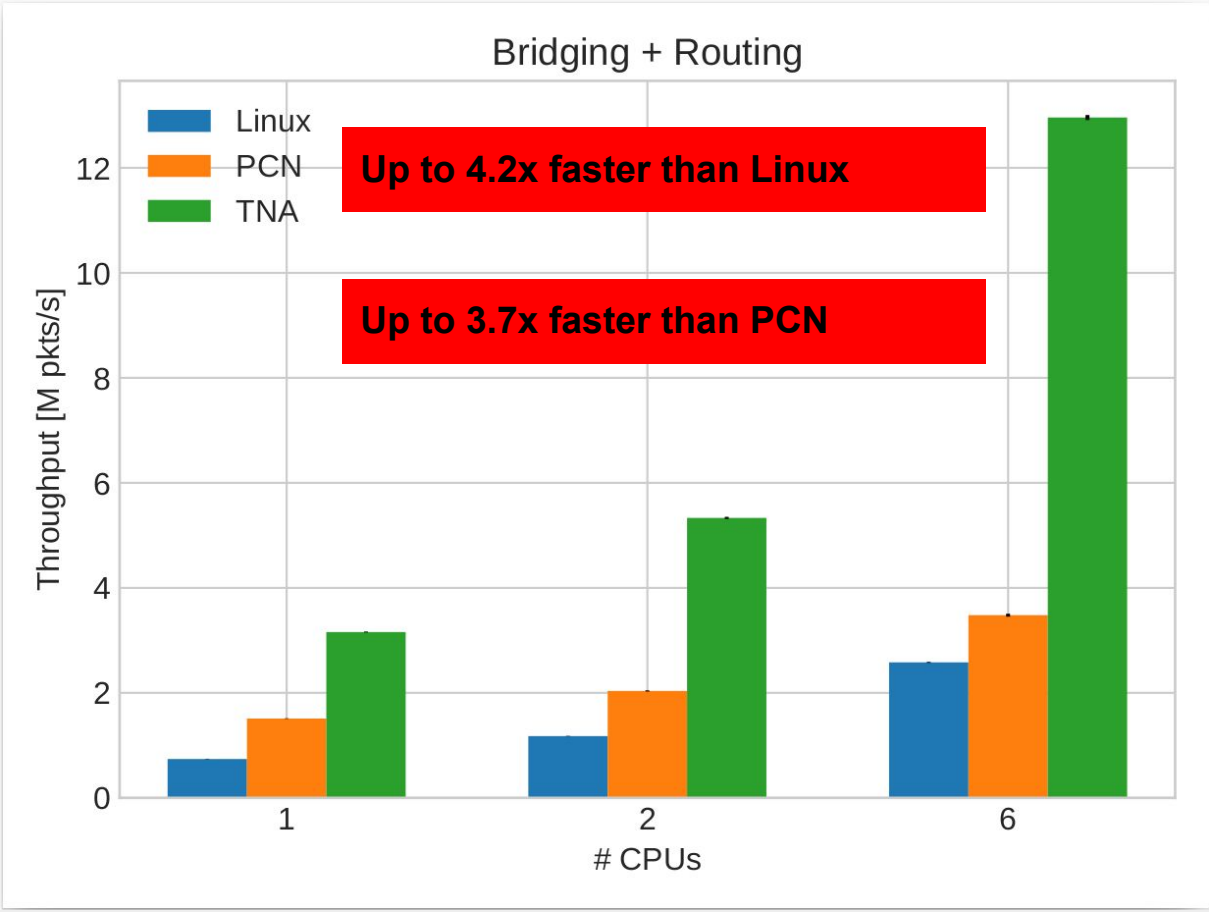

Bridging
Linux
PCN
TNA
Throughput [M pkts/s]
0
2
4
6
8
10
12
14
1
2
6
# CPUs

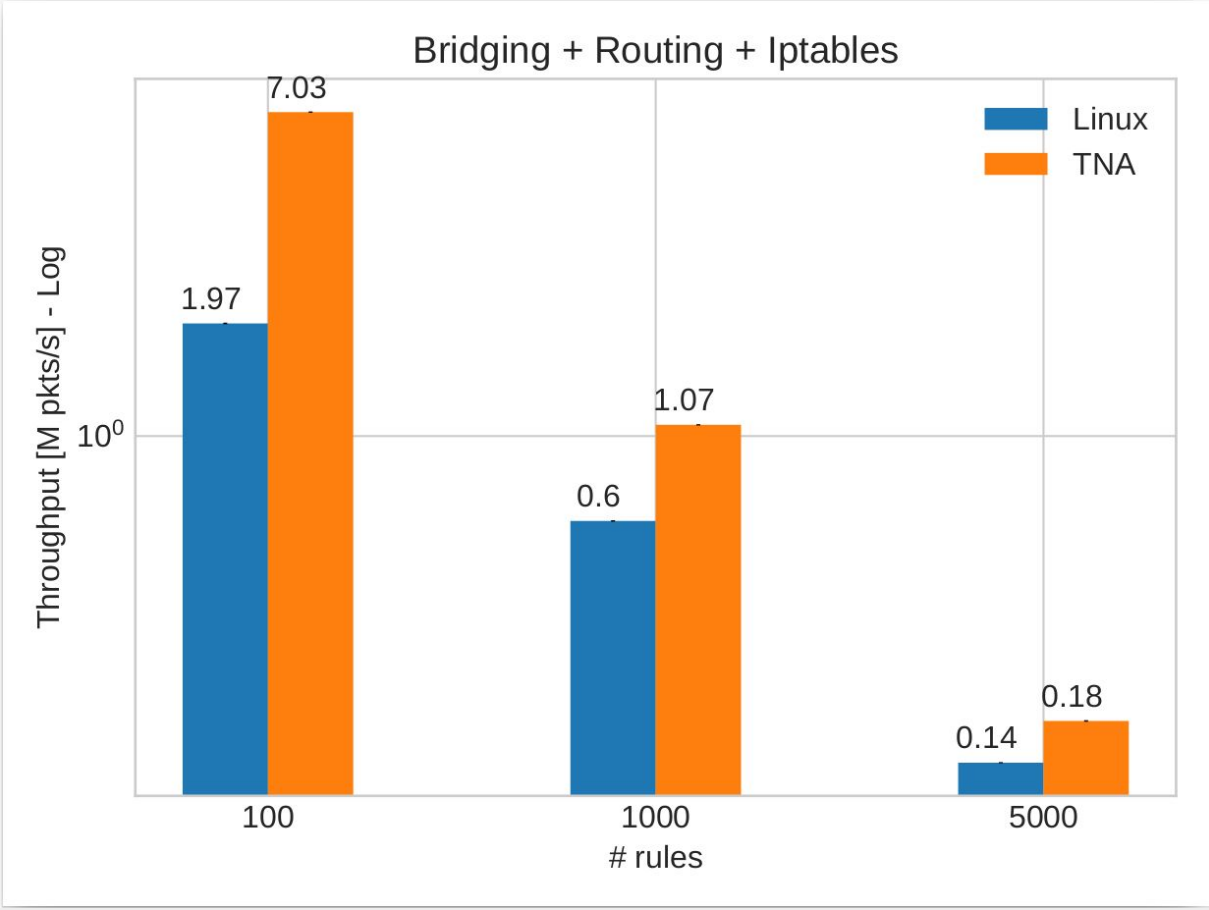

Bridging
Linux
PCN
TNA
Up to 3.1x faster than Linux
Up to 1.6x faster than PCN
Throughput [M pkts/s]
0
2
4
6
8
10
12
14
1
2
6
# CPUs

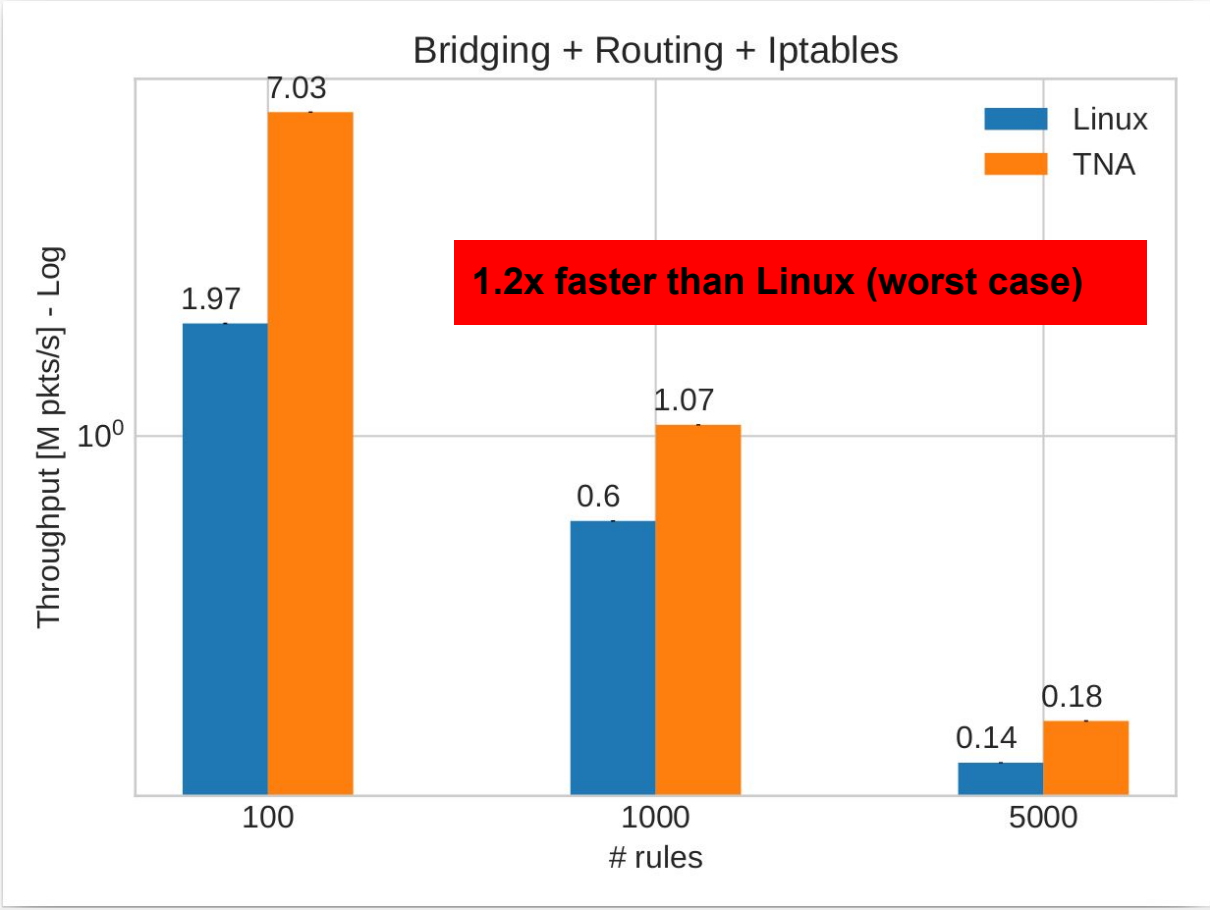
# Stacking Applications




Bridging + Routing
Linux
PCN
TNA
Throughput [M pkts/s]
0
2
4
6
8
10
12
1
2
6
# CPUs


Bridging + Routing
Linux
PCN
TNA
Up to 4.2x faster than Linux
Up to 3.7x faster than PCN
Throughput [M pkts/s]
0
2
4
6
8
10
12
1
2
6
# CPUs


Bridging + Routing + Iptables
Linux
TNA
Throughput [M pkts/s] - Log
$10^0$
1.97
7.03
0.6
1.07
0.14
0.18
100
1000
5000
# rules


Bridging + Routing + Iptables
Linux
TNA
1.2x faster than Linux (worst case)
Throughput [M pkts/s] - Log
$10^0$
7.03
1.97
1.07
0.6
0.18
0.14
100
1000
5000
# rules

# Contributions

Rethink Linux network stack

Make it fast, extensible, transparently to the rest of the system.

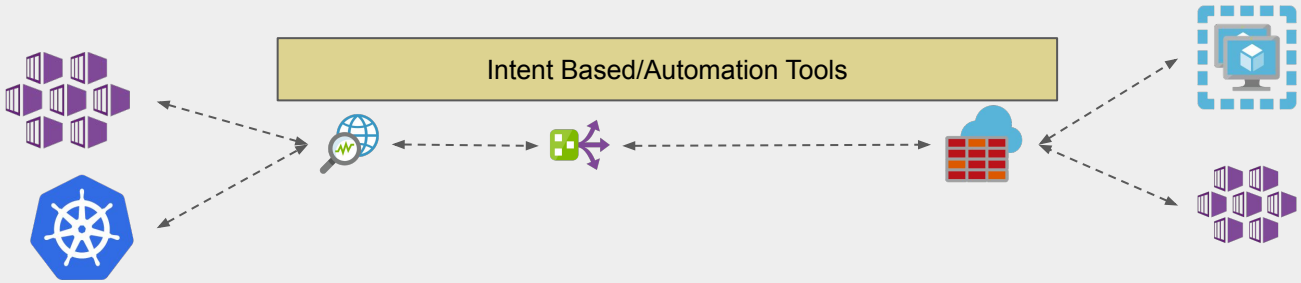
Publication: “Getting back what was lost in the era of high-speed software packet processing” Hotnets’22

# A vision for the future of modern networks …

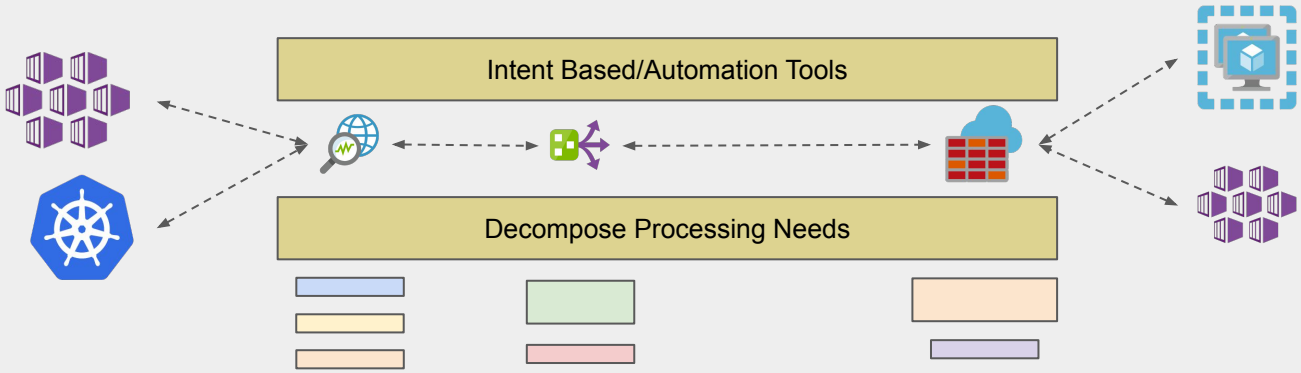
# A vision for the future of modern networks …

Intent Based/Automation Tools

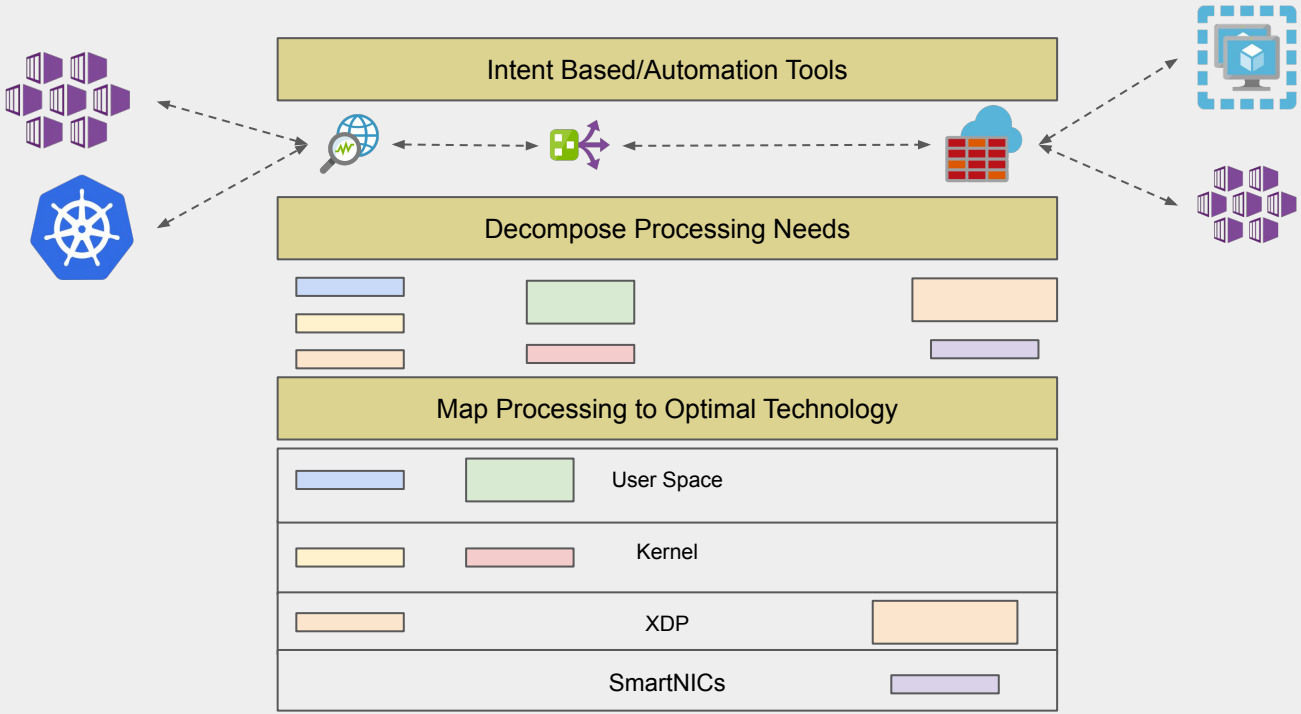
# A vision for the future of modern networks …



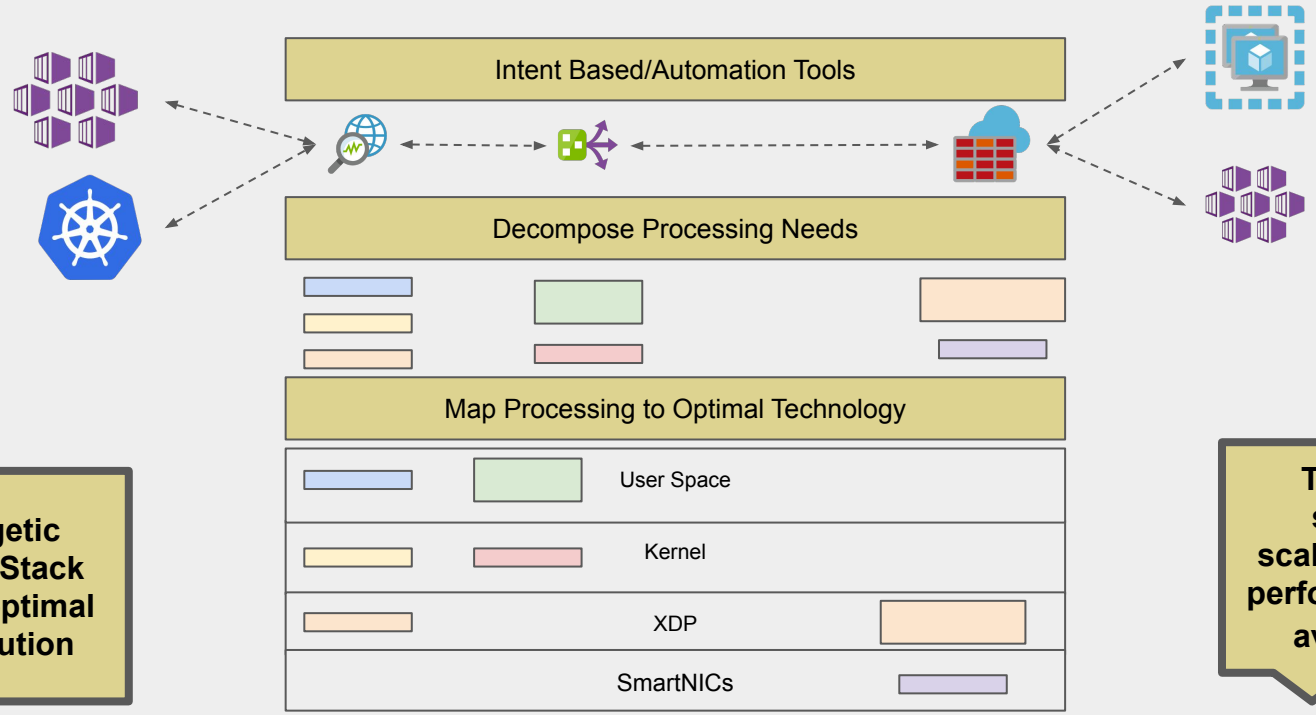
# A vision for the future of modern networks …



# A vision for the future of modern networks …



# A vision for the future of modern networks …




**Synergetic Network Stack enables optimal NF execution**

**To ensure security, scalability, high performance and availability**

# Thank you!

# Conclusion

Address needs and challenges of L2-L7 network applications and monitoring

Leverage synergies among multiple packet processing technologies

Build systems with high performance, efficiency and great functionality