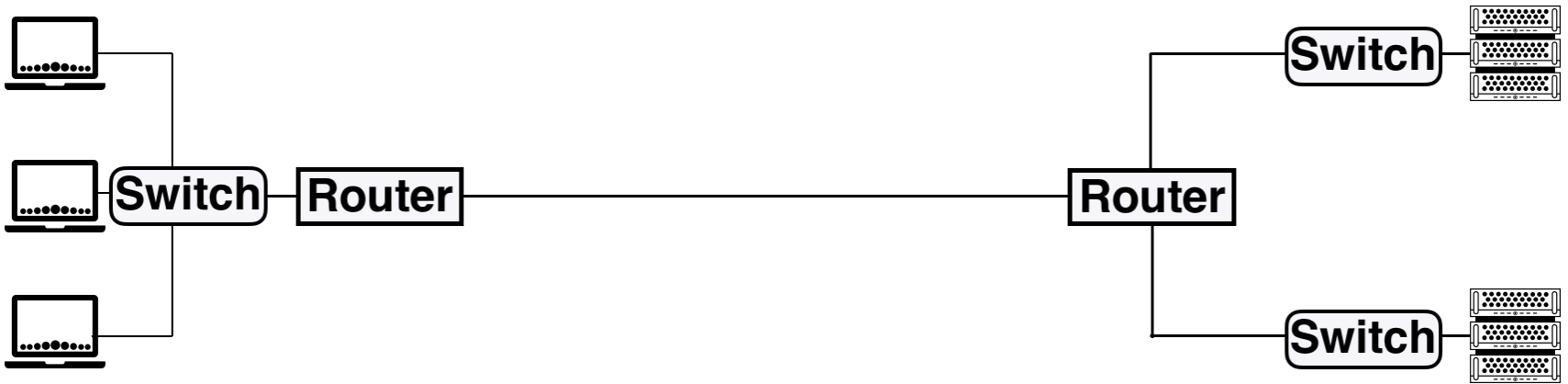


Network Functions

ECE/CS598HPN

Radhika Mittal

Conventional view of networks

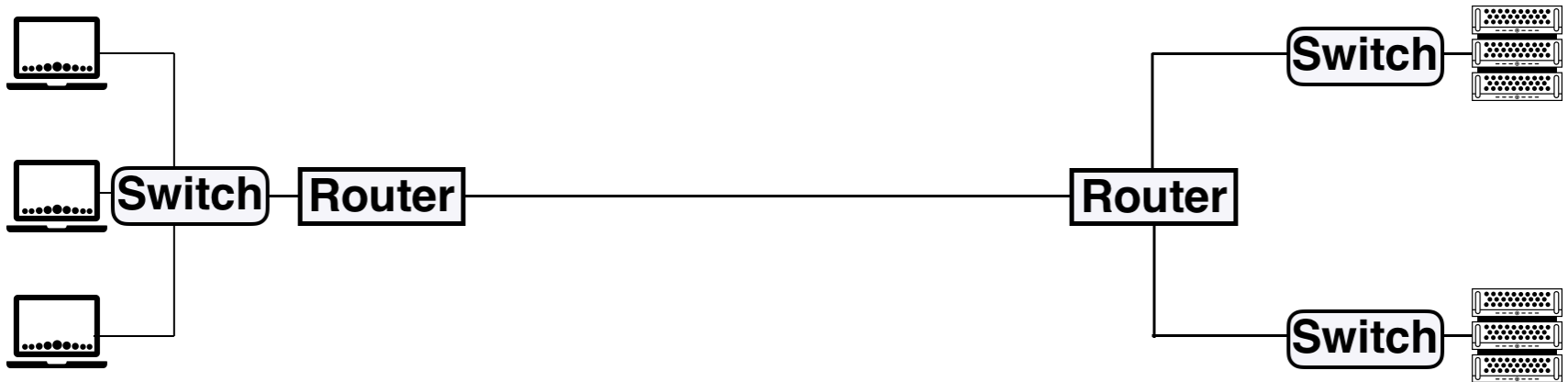


Data delivery is the only functionality provided by such a network.

**contents of the slide borrowed from talks given by Aurojit Panda, NYU*

Conventional view of networks

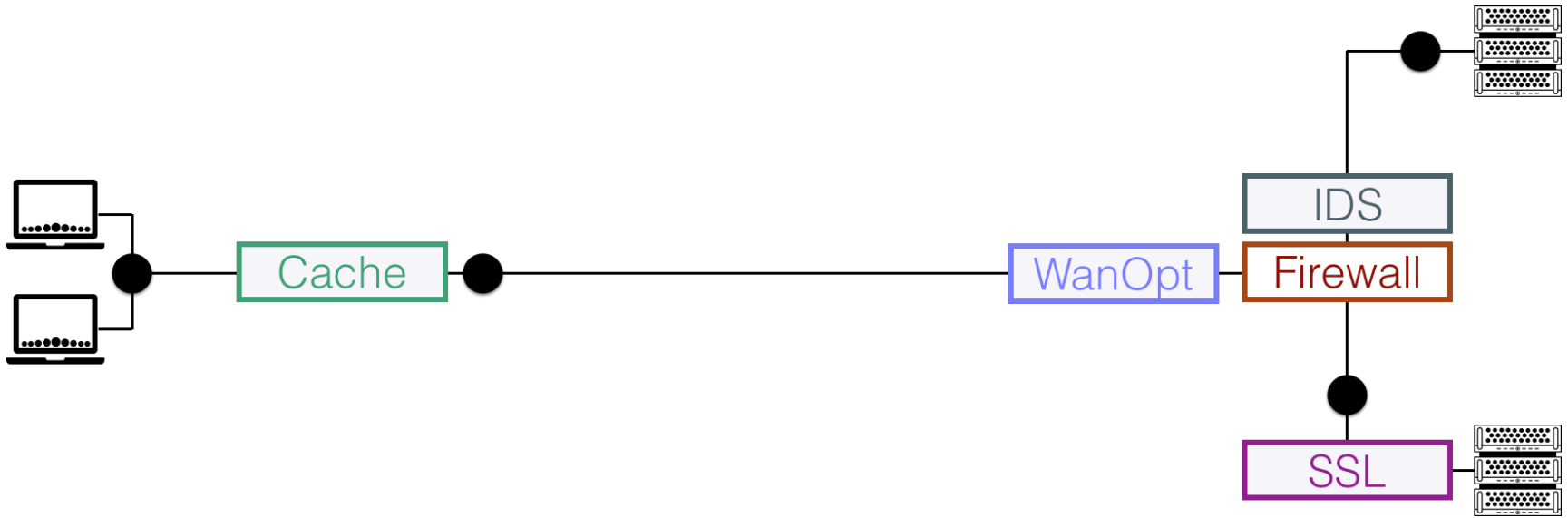
Data delivery is not the only required functionality.



**contents of the slide borrowed from talks given by Aurojit Panda, NYU*

Rise of middleboxes

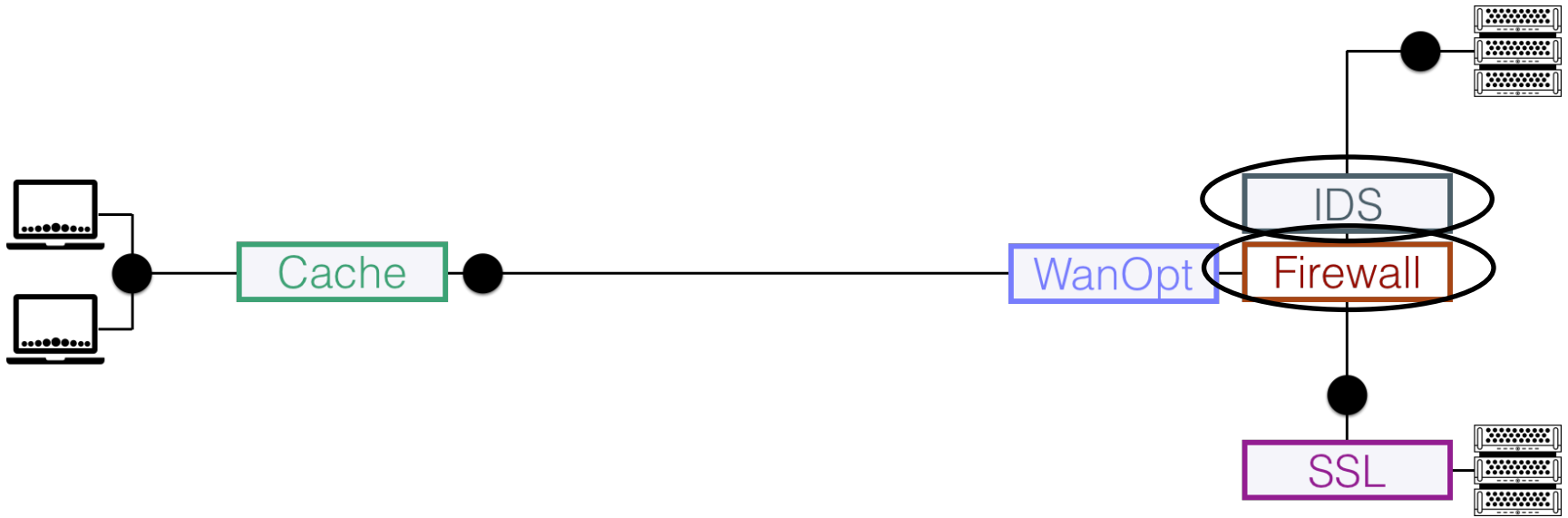
Data delivery is not the only required functionality.



**contents of the slide borrowed from talks given by Aurojit Panda, NYU*

Rise of middleboxes

Data delivery is not the only required functionality.

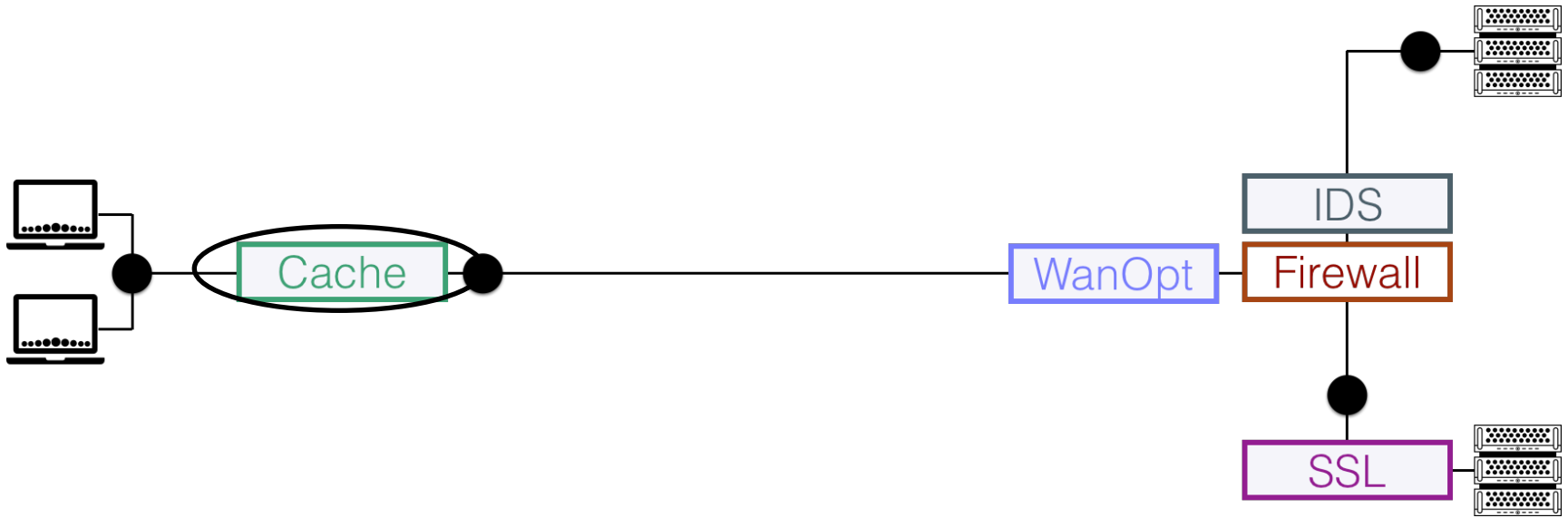


Security: identify and block unwanted traffic.

**contents of the slide borrowed from talks given by Aurojit Panda, NYU*

Rise of middleboxes

Data delivery is not the only required functionality.

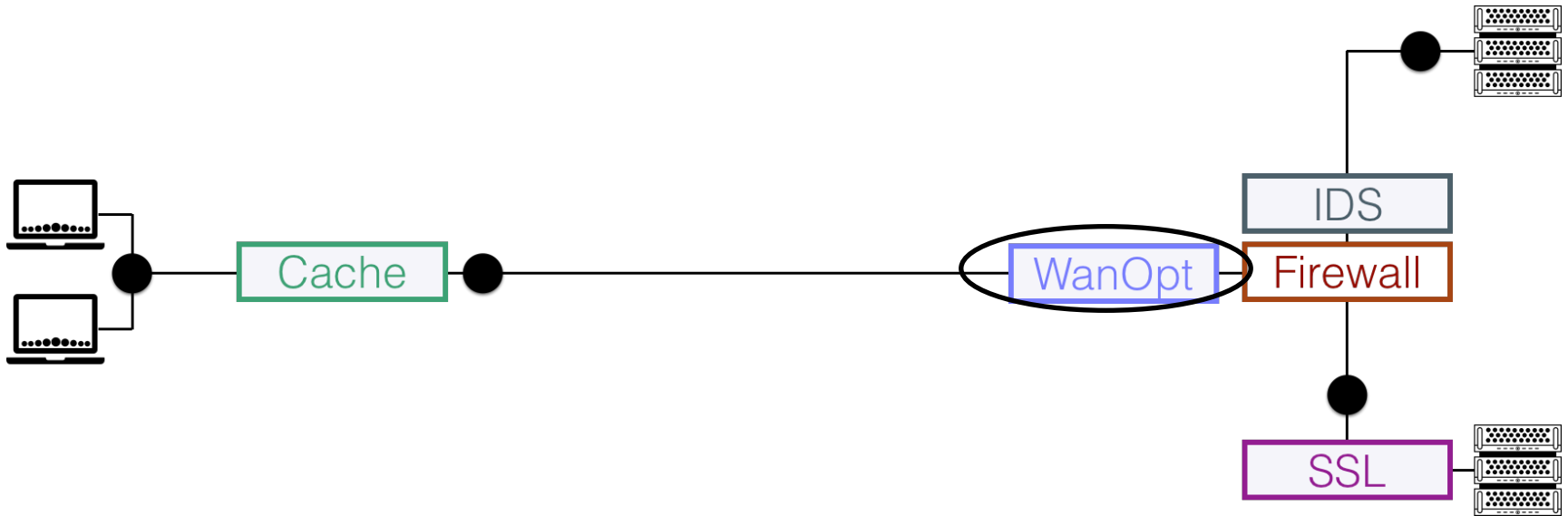


Performance: load content faster.

**contents of the slide borrowed from talks given by Aurojit Panda, NYU*

Rise of middleboxes

Data delivery is not the only required functionality.

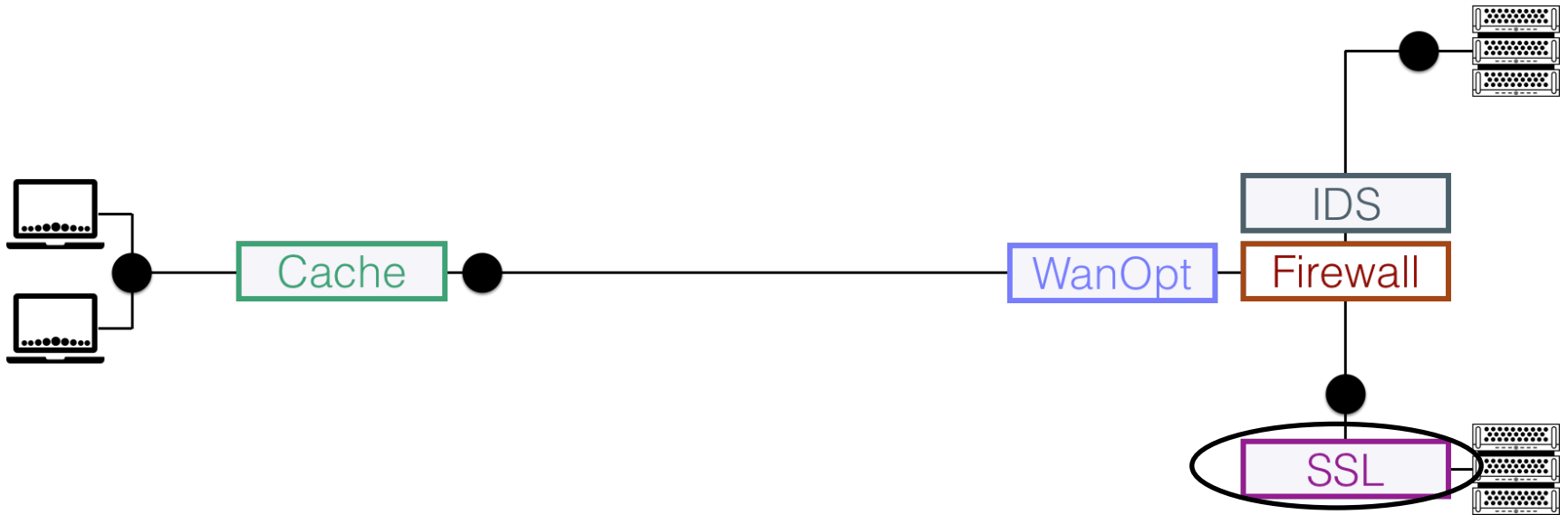


Performance: reduce bandwidth usage.

**contents of the slide borrowed from talks given by Aurojit Panda, NYU*

Rise of middleboxes

Data delivery is not the only required functionality.

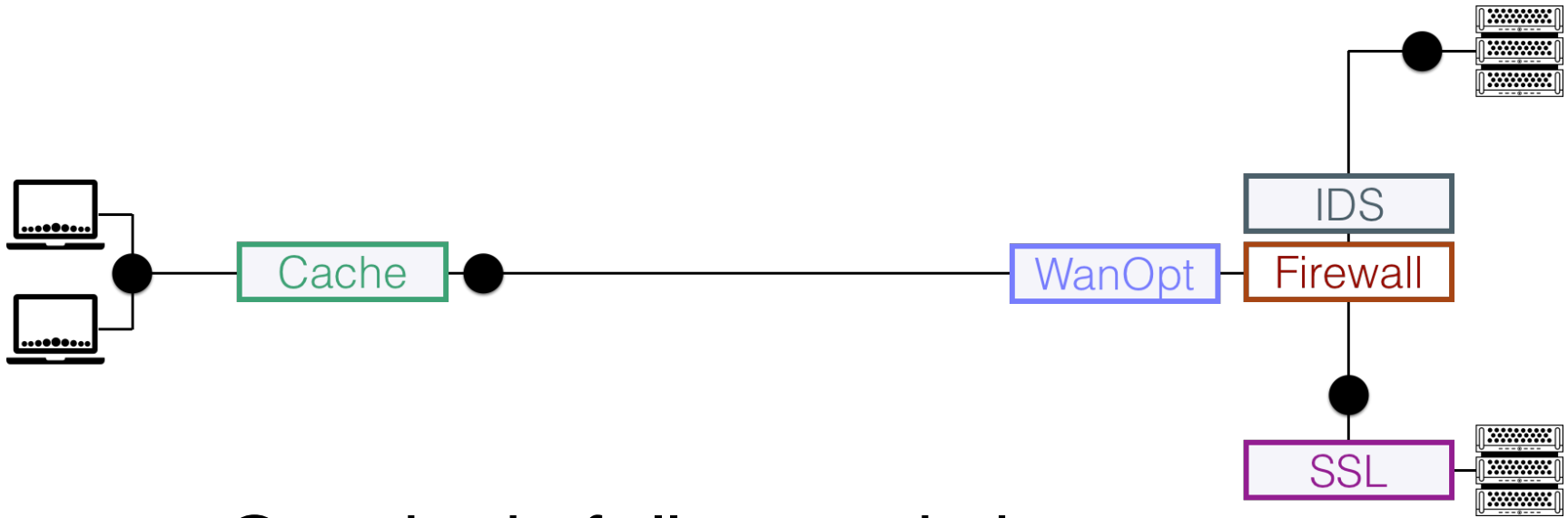


Application support: protocol for legacy application.

**contents of the slide borrowed from talks given by Aurojit Panda, NYU*

Rise of middleboxes

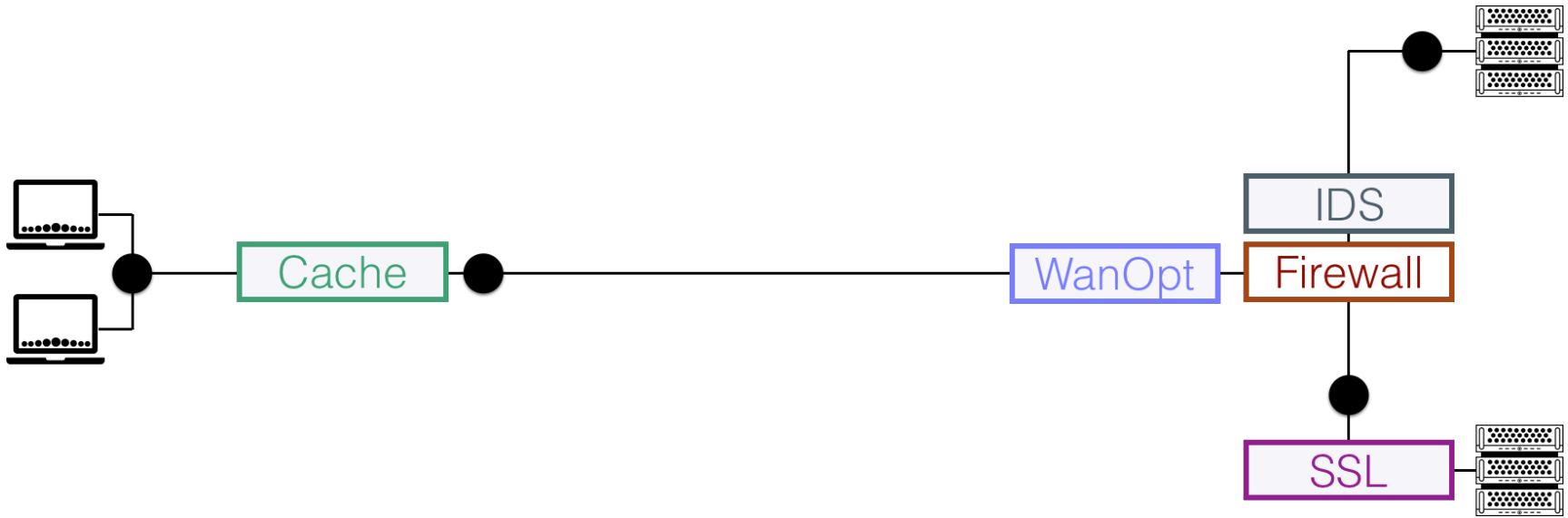
Data delivery is not the only required functionality.



One-third of all network devices in enterprises are middleboxes!
(source: Sherry et. al., SIGCOMM'12)

**contents of the slide borrowed from talks given by Aurojit Panda, NYU*

Rise of middleboxes

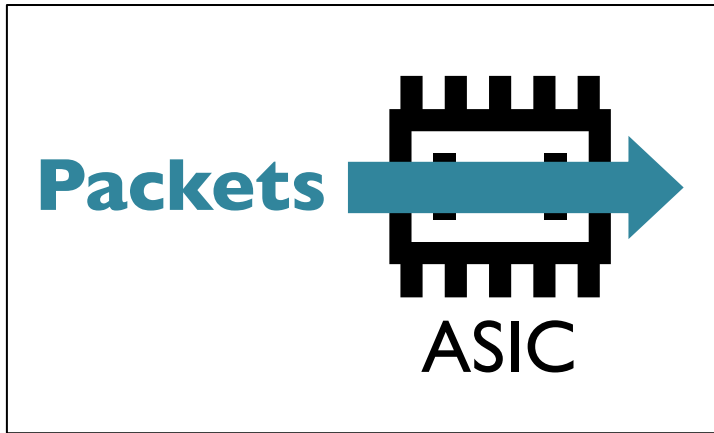


Stringent performance requirements:
process packets at line rate with
minimal latency overhead.

**contents of the slide borrowed from talks given by Aurojit Panda, NYU*

Evolution of middleboxes

Dedicated hardware

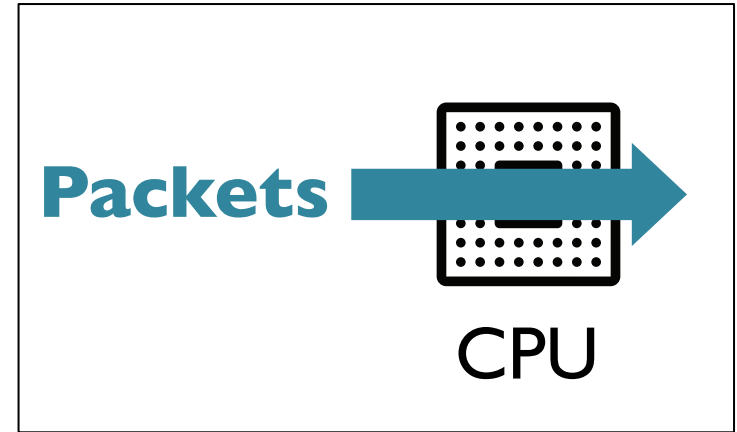


Middleboxes

*Need for
flexibility*



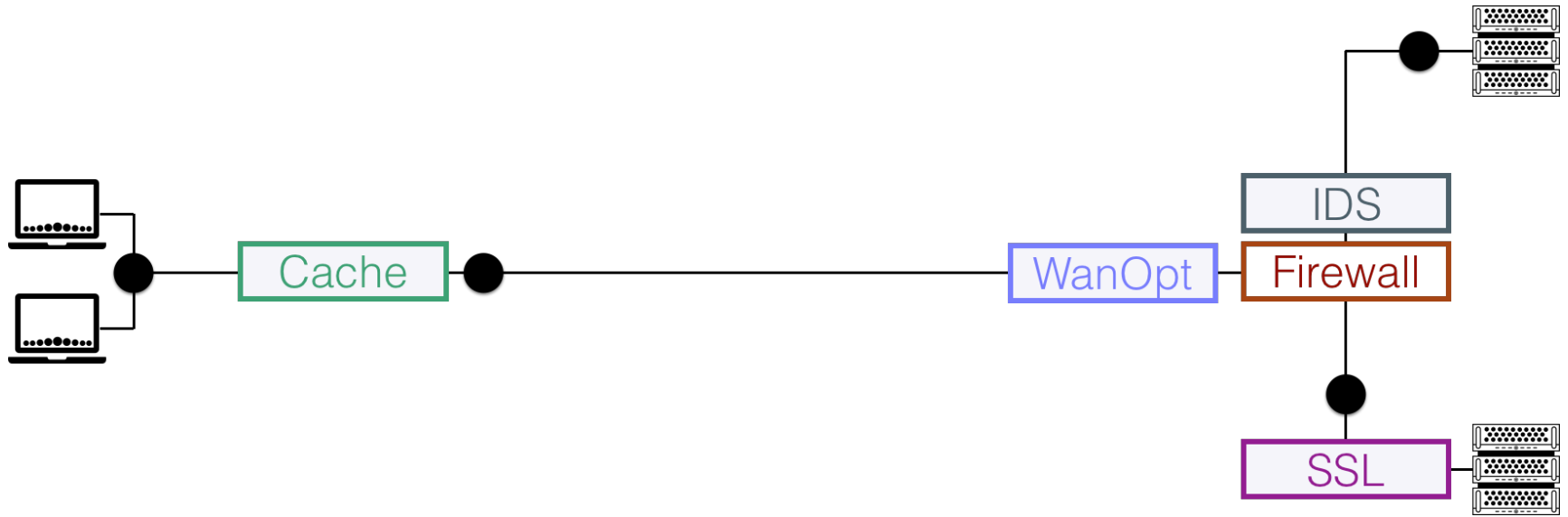
Software



Network functions

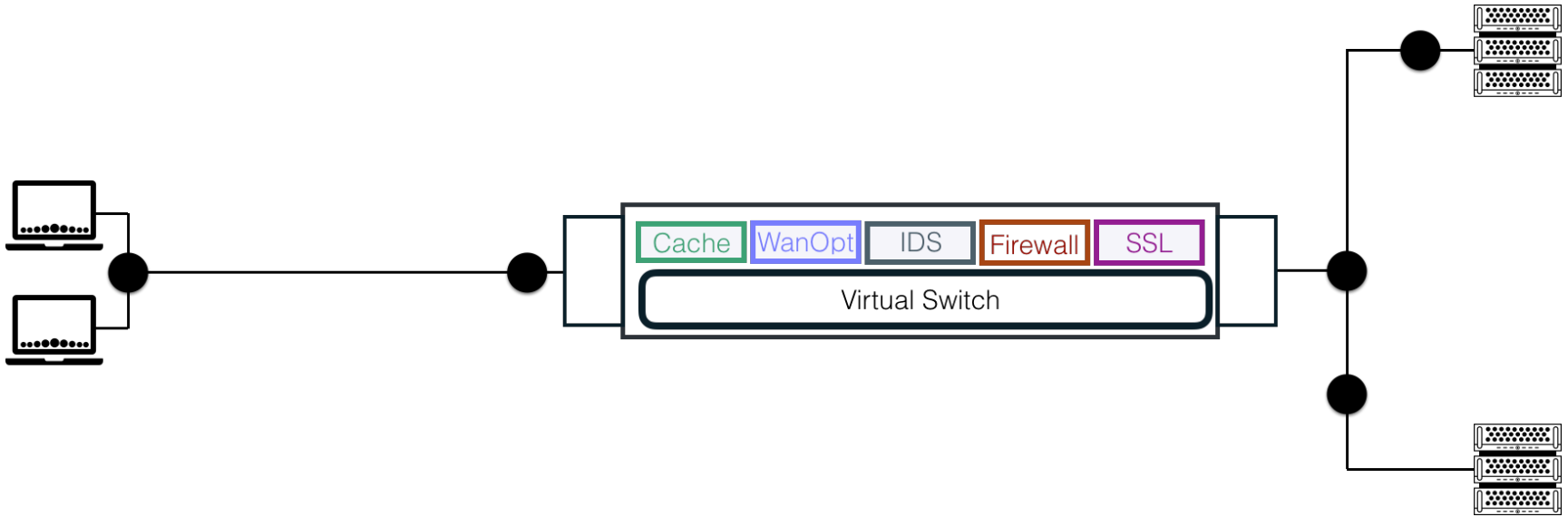
**contents of the slide borrowed from talks given by Aurojit Panda, NYU*

From hardware middleboxes....



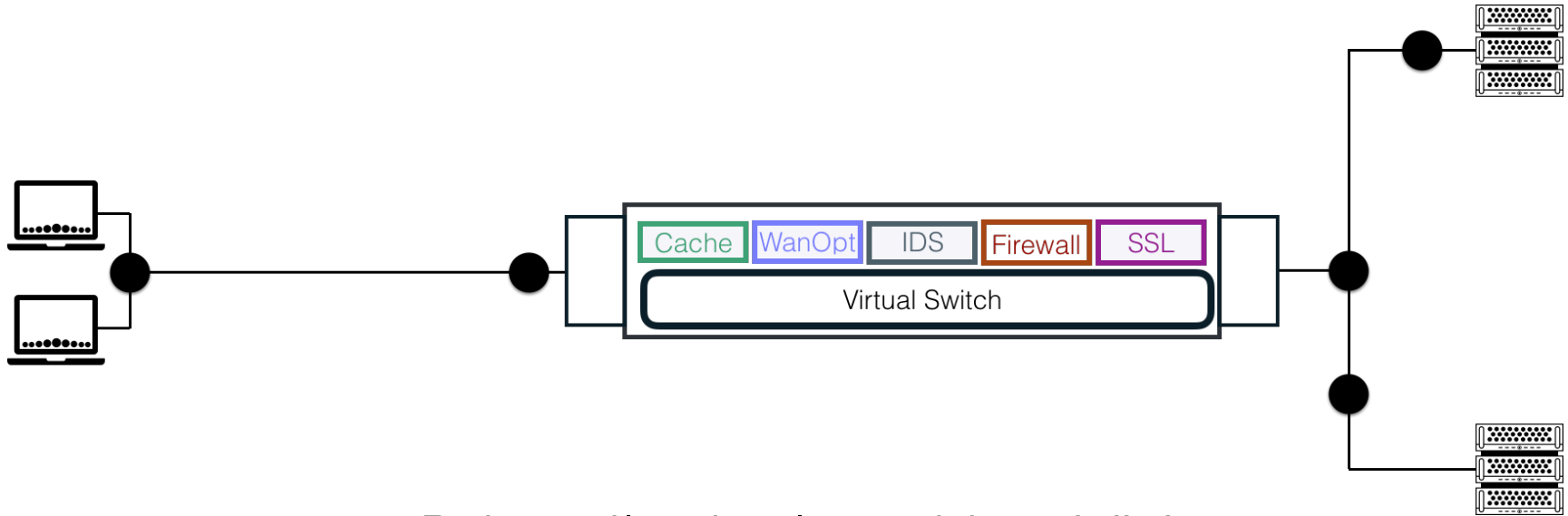
**contents of the slide borrowed from talks given by Aurojit Panda, NYU*

...to software network functions (NFs)



**contents of the slide borrowed from talks given by Aurojit Panda, NYU*

...to software network functions (NFs)

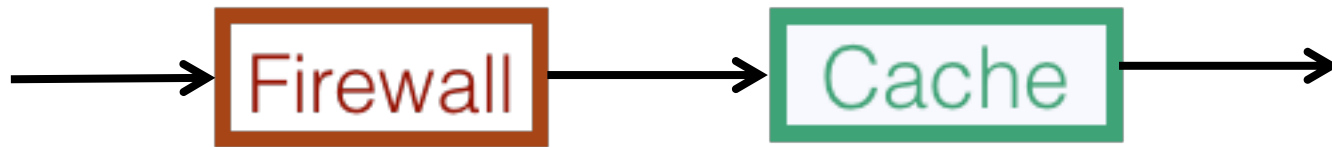


Primarily deployed in a VM
(Network Function Virtualization)

**contents of the slide borrowed from talks given by Aurojit Panda, NYU*

Key benefits of software network functions

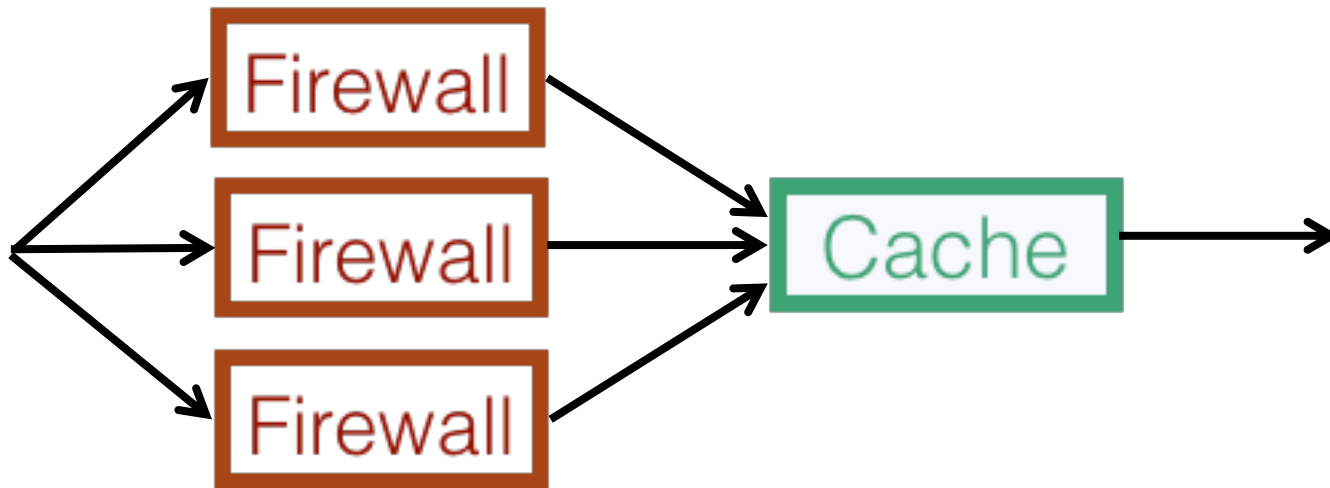
- Programmability
 - ability to update and create new NFs.
- Ease of deployment, configuration, and management.



NF Service Chain

Key benefits of software network functions

- Programmability
 - ability to update and create new NFs.
- Ease of deployment, configuration, and management.



Key benefits of software network functions

- Programmability
 - ability to update and create new NFs.
- Ease of deployment, configuration, and management.

Being adopted by both carriers and cloud providers.

Benefits of software NF come at a cost

- Complex and costly state management.
- Unpredictable performance.
- Performance degradation.

State management during scaling or failover

Split/Merge: System Support for Elastic Execution in Virtual Middleboxes

Shriram Rajagopalan^{†‡}, Dan Williams[†], Hani Jamjoom[†], and Andrew Warfield[‡]

[†]IBM T. J. Watson Research Center, Yorktown Heights, NY

[‡]University of British Columbia, Vancouver, Canada

Rollback-Recovery for Middleboxes

Justine Sherry*, Peter Xiang Gao*, Soumya Basu*, Aurojit Panda*,
Arvind Krishnamurthy*, Christian Maciocco†, Maziar Manesh†, João Martins‡,
Sylvia Ratnasamy*, Luigi Rizzo†, Scott Shenker*

* UC Berkeley • University of Washington † Intel Research ‡ NEC

ABSTRACT

OpenBox: A Software-Defined Framework for Developing, Deploying, and Managing Network Functions

Anat Bremner-Barr*, Yotam Harchol†, David Hay†,
bremner@idc.ac.il, yotamhc@cs.huji.ac.il, dhay@cs.huji.ac.il

* School of Computer Science
† School of Computer Science

ABSTRACT

We present OpenBox — a software-defined framework for network-wide development, deployment, and management of *network functions* (NFs). OpenBox actively decouples the control plane of network functions from the forwarding plane, similarly to SDN solutions. OpenBox consists of three log

Paving the Way for NFV: Simplifying Middlebox Modifications using StateAlyzr

Junaid Khalid, Aaron Gember-Jacobson, Roney Michael,
Anubhavnidhi Abhashkumar, Aditya Akella
University of Wisconsin-Madison

Abstract

central contribution of this paper is a novel, framework-

Elastic Scaling of Stateful Network Functions

Shinae Woo*, Justine Sherry†, Sangjin Han*, Sue Moon†, Sylvia Ratnasamy*, and Scott Shenker*‡

*University of California, Berkeley †KAIST ‡CMU §ICSI

Abstract

Elastic scaling is a central problem to realize in practice. The most Network Functions (NFs) need to be *shared* across NF instances while meeting requirements placed on NFs. No solution exists that meets the full spectrum of NFs.

S6 is a new framework that builds on the insight that abstraction is well-suited to the state as a distributed shared resource concept.

Stateless Network Functions: Breaking the Tight Coupling of State and Processing

Murad Kablan, Azzam Alsudais, Eric Keller
University of Colorado, Boulder

Franck Le
IBM Research

Pico Replication: A High Availability Framework for Middleboxes

Shriram Rajagopalan^{†‡}, Dan Williams[†], Hani Jamjoom[†]
[†]IBM T. J. Watson Research Center, Yorktown Heights
[‡]University of British Columbia

Abstract

Middleboxes are being rearchitected, composed, extensible, and level support for high availability. We propose *Pico Replication (PR)*, a structure to achieve low overhead, HA. Unlike generic (virtual machine) PR operates at the virtual machine

rewalls, intrusion detection systems, translators, and load balancers no longer rely on proprietary hardware, but can run in software on commodity servers, in a virtualized environment [25]. This shift away from hardware should bring several benefits including: (1) elastically scale the network function (2) recover from failures. Others have reported, achieving those that simple [44, 45, 73, 100, 73].

E2: A Framework for NFV Applications

Shoumik Palkar*
UC Berkeley
sppalkar@berkeley.edu

Chang Lan*
UC Berkeley
clan@eecs.berkeley.edu

Sangjin Han
UC Berkeley
sangjin@eecs.berkeley.edu

Keon Jang
Intel Labs
keon.jang@intel.com

Aurojit Panda
UC Berkeley
apanda@cs.berkeley.edu

Sylvia Ratnasamy
UC Berkeley
sylvia@eecs.berkeley.edu

Luigi Rizzo
Università di Pisa
rizzo@iet.unipi.it

Scott Shenker
UC Berkeley and ICSI
shenker@icsi.berkeley.edu

Understanding NF Performance

Backtracking Algorithmic Complexity Attacks Against a NIDS

Randy Smith Cristian Estan Somesh Jha
Computer Sciences Department

Automated Synthesis of Adversarial Workloads for Network Functions

Luis Pedrosa
EPFL
luis.pedrosa@epfl.ch

Rishabh Iyer
EPFL
rishabh.iyer@epfl.ch

Arseniy Zaostrovnykh
EPFL
arseniy.zaostrovnykh@epfl.ch

Jonas Fietz
EPFL
jonas.fietz@epfl.ch

Katerina Argyraki
EPFL
katerina.argyaki@epfl.ch

ABSTRACT

Software network management of network performance. However, they find that during deployment, performance of the network workloads. We challenge: it takes as much time to send and outputs packets over multiple paths. Under the condition with a sophisticated path that incur more memory-access paths functions that implement covered workloads cut throughput by 1

KEYWORDS

Network Function P

1 INTRODUCTION

This work is about software that runs on top of code, typically written in C, that handles processing functionality and network address translation. This functionality has been relegated to specialized middleboxes, often implemented in hardware. However, there has been a push to move this functionality to the potential to offer more flexibility and reduced capital and operational costs. This shift from hard

ACM acknowledges that this work was performed, in whole or in part, by an employee, contractor or subcontractor of the U.S. Government. The Government retains a non-exclusive, irrevocable, and exclusive right to reproduce this article, or to allow others to do so, for government purposes.

Denial of Service via Algorithmic Complexity Attacks

Scott A. Crosby
scrosby@cs.rice.edu

Dan S. Wallach
dwallach@cs.rice.edu

Department of Computer Science, Rice University

Abstract

We present a new class of low-bandwidth denial of service attacks that exploit algorithmic deficiencies in many common applications' data structures. Frequently used data structures have "average-case" expected running time that's far more efficient than the worst case. For example, both binary trees and hash tables can degenerate to linked lists with carefully chosen input. We show how an attacker can effectively compute such input, and we demonstrate attacks against the hash table implementations in two versions of Perl, the Squid web proxy, and the intrusion detection system. Using bandwidth

sume $O(n)$ time to insert n elements. However, if each element hashes to the same bucket, the hash table will also degenerate to a linked list, and it will take $O(n^2)$ time to insert n elements.

While balanced tree algorithms, such as red-trees [11], AVL trees [11], and treaps [17] can predictably input which causes worst-case behavior, and universal hash functions [5] can make hash functions that are not predictable to an attacker, many common applications use algorithms. If an attacker can control the inputs being used by these algorithms, the attacker may be able to induce the worst-case time, effectively causing a denial-of-service (DoS) attack.

Categories a
C.2 COMPUTER
work Operations

Keywords
data center network

3262

IEEE/ACM TRANSACTIONS ON NETWORKING, VOL. 24, NO. 6, DECEMBER 2016

Making DPI Engines Resilient to Algorithmic Complexity Attacks

Yehuda Afek, *Member, IEEE*, Anat Bremner-Barr, *Member, IEEE*, Yotam Harchol, *Member, IEEE*,
David Hay, *Member, IEEE*, and Yaron Koral, *Member, IEEE*

2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)

NFVPerf: Online Performance Monitoring and Bottleneck Detection for NFV

Priyanka Naik, Dilip Kumar Shaw, Mythili Vutukuru
Department of Computer Science and Engineering, Indian Institute of Technology, Bombay
Email: {ppnaik, dilip13, mythili}@cse.iitb.ac.in

The recent interest in NFV has been spurred by the advent of SD-WAN and SDN techniques for efficient packet processing in the network. NFV is expected to save costs

to add network appliances hardware. The level of operators is, to make

PerfSight: Performance Diagnosis for Software Dataplanes

Wenfei Wu, Kegiang He, Aditya Akella
University of Wisconsin-Madison

ABSTRACT

The advent of network functions virtualization (NFV) means that data planes are no longer simply composed of hardware. Instead they are very complex, and they are increasingly so.

Enforcing Network-Wide Consistency of Dynamic Middleboxes

Enforcing Network-Wide Policies in the Presence of Dynamic Middlebox Actions using FlowTags

Seyed Kaveh Fayazbakhsh* Luis Chiang† Vyas Sekar* Minlan Yu‡ Jeffrey C. Mogul*

*Carnegie Mellon University †Deutsche Telekom Labs ‡USC *Google

Abstract

Middleboxes provide key security and performance guarantees in networks. Unfortunately, the dynamic traffic modifications they induce make it difficult to reason

ing (SDN) to enforce and verify network-wide policies (e.g., [39, 40, 44]) does not extend to networks with middleboxes. Specifically, middlebox actions violate two key SDN tenets [24, 32]:

Providing guarantees about NF behavior

A Formally Verified NAT

Arseniy Zaozrovnykh
EPFL, Switzerland
arseniy.zaozrovnykh@epfl.ch

Solal Pirelli
EPFL, Switzerland
solal.pirelli@epfl.ch

Luis Pedrosa
EPFL, Switzerland
luis.pedrosa@epfl.ch

George Candea
EPFL, Switzerland
george.candea@epfl.ch

Katerina Argyraki
EPFL, Switzerland
katerina.argyaki@epfl.ch

There exists a lot of prior work on network verification, but, none that reasons about both the

Most
ough
oning
5, 59].
otion
-level
-) [35].
tevenful
e, even
ecution
at it is
out the

Software Dataplane Verification

Mihai Dobrescu and Katerina Argyraki
EPFL, Switzerland

Abstract

Software dataplanes are emerging as an alternative to traditional hardware switches and routers, promising programmability and short time to market. These advantages are set against the risk of disrupting the network with bugs, unpredictable performance, or security vulnerabilities. We explore the feasibility of verifying software dataplanes to ensure smooth network operation. For general programs, verifiability and performance are competing goals; we argue that software dataplanes are different—we can write them in a way that enables verification and preserves performance. We present a verification tool that takes as input a software dataplane, written in a way that meets a given set of conditions, and (dis)proves that the dataplane satisfies crash-freedom, bounded-execution, and filtering properties. We evaluate our tool on stateless and simple stateful Click pipelines; we perform complete and sound verification of these pipelines within tens of minutes, whereas a state-of-the-art general-purpose tool fails to complete the same task within several hours.

1 Introduction

Software dataplanes are emerging from both research [17, 26, 27, 37] and industry [2, 3] backgrounds as a more flexible alternative to traditional hardware switches and routers. They promise to cut network provisioning costs by half, by enabling dynamic allocation of packet-processing tasks to network devices [42]; or to turn the Internet into an evolvable architecture, by enabling continuous functionality update of devices located at strate-

The subject of this work is a verification tool that takes as input the executable binary of a software dataplane and proves that it does (or does not) satisfy a target property; if the target property is not satisfied, the tool should provide counter-examples, i.e., packet sequences that cause the property to be violated. Developers of packet-processing apps could use such a tool to produce software with guarantees, e.g., that never seg-faults or kernel-panics, no matter what traffic it receives. Network operators could use the tool to verify that a new packet-processing app they are considering for deployment will not destabilize their network, e.g., it will not introduce more than some known fixed amount of per-packet latency. One might even envision markets for packet-processing apps—similar to today's smartphone/tablet app markets—where network operators would shop for new code to “drop” into their network devices. The operators of such markets would need a verification tool to certify that their apps will not disrupt their customers' networks.

For general programs, verifiability and performance are competing goals. Proving properties of real programs (unlike searching for bugs) remains an elusive goal for the systems community, at least for programs that consist of more than a few hundred lines of code and are written in a low-level language like C++. A high-level language like Haskell can guarantee certain properties like the impossibility of buffer overflow by construction, but typically at the cost of performance.

For software dataplanes, it does not have to be this way: we will argue that we can write them in a way that enables verification and preserves performance. The key question then is: what defines a “software dataplane” and how much more restricted is it than a “general program”?

SymNet: scalable symbolic execution for modern networks

Radu Stoenescu, Matei Popovici, Lorina Negreanu, Costin Raiciu
University Politehnica of Bucharest
Splaiul Independentei 313, Bucharest, Romania
firstname.lastname@cs.pub.ro

Abstract

We present SymNet, a network static analysis tool based on symbolic execution. SymNet injects symbolic packets and tracks their evolution through the network. Our key novelty is SEFL, a language we designed for expressing data plane processing in a symbolic-execution friendly manner. SymNet statically analyzes an abstract data plane model that consists of the SEFL code for every node and the links between nodes. SymNet can check networks containing routers with hundreds of thousands of prefixes and NATs in seconds, while verifying packet header memory-safety and covering network functionality such as dynamic tunneling, stateful processing and encryption. We used SymNet to debug mid-

dlebox interactions from the literature, to check properties of our department's network and the Stanford backbone. Modeling network functionality is not easy. To aid users we have developed parsers that automatically generate SEFL of our department's network and the Stanford backbone. Modeling network functionality is not easy. To aid users we have developed parsers that automatically generate SEFL of our department's network and the Stanford backbone. Modeling network functionality is not easy. To aid users we have developed parsers that automatically generate SEFL of our department's network and the Stanford backbone.

Verifying Reachability in Networks with Mutable Datapaths

Aurojit Panda* Ori Lahav† Katerina Argyraki‡ Mooly Sagiv◇ Scott Shenker◆
*UC Berkeley †MPI-SWS ‡EPFL ◇TAU ◆ICSI

Abstract

Recent work has made great progress in verifying the forwarding correctness of networks [26–28, 35]. However, these approaches cannot be used to verify networks containing middleboxes, such as caches and firewalls, whose forwarding behavior depends on previously observed traffic. We explore how to verify reachability properties for networks that include such “mutable datapath” elements. The main challenge lies in handling large and complex networks. We achieve scaling by de-

boxes without changes in the physical infrastructure [13]. Given their complexity and prevalence, middleboxes are the cause of many network failures; for instance, 43% of a network provider's failure incidents involved middleboxes, and between 4% and 15% of middlebox incidents were the result of middlebox extensions.

or distributed firewall policy compliance is difficult before deploying the network configuration, and deployment can disrupt live traffic. Dynamic testing (packet generation and tracing) can only catch common issues (e.g. lack of connectivity) but does not scale to large networks. Static analysis of network data planes allows cheap, fast and exhaustive verification of deployed networks for packet reachability, absence of loops, bidirectional forwarding, etc. We do not aim to verify the control plane (e.g. routing protocols, SDN controllers etc.). Control plane verification is a hard problem that includes checking the correctness of the control plane configuration [8, 9, 4], proving convergence after link additions or failures and characterizing the transient behavior until convergence is reached [2, 11]. We assume the control plane configuration is stable and the control plane has converged and analyze the resulting data plane.

All static analysis tools take as input a model of the problem being solved and a snapshot of the forwarding state, and are able to answer queries about the network without resorting to dynamic testing [23, 14, 19, 20, 21]. What is the best simple use language for networks? If possible, we should simply use the implementation of network boxes (e.g. a C program), as this is the most accurate and is easiest to use. If we view packets as variables being passed between different network boxes, static network analysis becomes akin to software testing. This is a problem that has been studied for decades, and Symbolic execution is to use symbolic execution [3]. Symbolic execution is really powerful; it explores all possible values the leading approach is to use symbolic execution. In the context of Symbolic execution the program, providing possible values at every point. Symbolic execution is really powerful; it explores all possible values the leading approach is to use symbolic execution. In the context of Symbolic execution the program, providing possible values at every point.

High performance NF implementations

Microboxes: High Performance NFV with Customizable, Asynchronous TCP Stacks and Dynamic Subscriptions

Guyue Liu*, Yuxin Ren*, Mykola Yurchenko*,
K.K. Ramakrishnan†, Timothy Wood*

*George Washington University, †University of California, Riverside

FlowBlaze: Stateful Packet Processing in Hardware

Salvatore Pontarelli^{1,2}, Roberto Bifulco³, Marco Bonola^{1,2}, Carmelo Cascone⁴,
Marco Spaziani^{2,5}, Valerio Bruschi^{2,5}, Davide Sanvito⁶, Giuseppe Siracusano³,
Antonio Capone⁶, Michio Honda³, Felipe Huici³ and Giuseppe Bianchi^{2,5}

¹Axbryd, ²CNIT, ³NEC Laboratories Europe, ⁴Open Networking Foundation,
⁵University of Rome Tor Vergata, ⁶Politecnico di Milano

Abstract

While programmatic handle growing network yet simple abstraction in hardware remain problem with Flow stateful packet processing is based on produces the explicit Blaze to leverage flow expressive, supporting tions, and easy to utation issues from FlowBlaze on a Ne tency (in the order tively little power, thousands of flows, for even higher spe ware and software ically available.

1 Introduction

Network infrastructure network functions t and server load bal such as access con examples. Given the need to contin tors have turned to

NetBricks: Taking the V out of NFV

Aurojit Panda† Sangjin Han† Keon Jang‡ Melvin Walls† Sylvia Ratnasamy† Scott Shenker†*
† UC Berkeley ‡ Google * ICSI

Abstract

The move from hardware middleboxes to software network functions, as advocated by NFV, has proven more challenging than expected. Developing new NFs remains a tedious process, requiring that developers repeatedly rediscover and reapply the same set of optimizations, while current techniques for providing isolation between NFs (using VMs or containers) incur high performance overheads. In this paper we describe NetBricks, a new NFV framework that tackles both these problems. For building NFs we take inspiration from modern data analytics frameworks (e.g., Spark and Dryad) and build a small set of customizable network processing elements. We also embrace type checking and safe runtimes to provide isolation in software, rather than rely on hardware isolation. NetBricks provides the same memory isolation as containers and VMs, without incurring the same performance penalties. To improve I/O

standard tools for managing VMs; (c) faster development, which now requires writing software that runs on commodity hardware; and (d) reduced costs by consolidating several NFs on a single machine. However, despite these promised advances, there has been little progress towards large-scale NF deployments. Our discussions with three major carriers revealed that they are only just beginning small scale test deployments (with 10-100s of customers) using simple NFs e.g., firewalls and NATs.

The move from hardware middleboxes to software NFs was supposed to speed innovation, so why has progress been so slow? We believe this delay is because traditional approaches for both *building* and *running* NFs are a poor match for carrier networks, which have the following requirements: *performance*, NF deployments should be able to provide per-packet latencies on the order of 10s of μ s, and throughput on the order of 10s of Gbps; *efficiency*, it should be possible to consolidate several NFs on a sin-

mOS: A Reusable Networking Stack for Flow Monitoring Middleboxes

Muhammad Jamshed, YoungGyoun Moon, Donghwi Kim, Dongsu Han, and KyoungSoo Park
School of Electrical Engineering, KAIST

ClickNP: Highly Flexible and High Performance Network Processing with Reconfigurable Hardware

Bojie Li§† Kun Tan† Layong (Larry) Luo† Yanqing Peng*† Renqian Luo§†
Ningyi Xu† Yongqiang Xiong† Peng Cheng† Enhong Chen§
†Microsoft Research §USTC ‡Microsoft *SJTU

ABSTRACT

flexible software network functions (NFs) are crucial components to enable multi-tenancy in the clouds. Hardware packet processing on a commodity server has capacity and induces high latency. While software can scale out using more servers, doing so adds significant cost. This paper focuses on accelerating NFs with reconfigurable hardware, i.e., FPGA, which is now a mainstream and inexpensive for datacenters. However, it is predominately programmed using low-level hardware description languages (HDLs), which are hard to code and difficult to debug. More importantly, HDLs are almost unusable for most software programmers. This paper presents ClickNP, a FPGA-accelerated platform for highly flexible high-performance NFs with commodity servers. ClickNP is flexible as it is completely programmable using C-like languages, and exposes a modular programming abstraction that resembles Click Modular Router. ClickNP achieves high performance. Our prototype NFs show that they process traffic at up to 200 million packets per second at a low latency (< 2 μ s). Compared to existing soft-interpreters, with FPGA, ClickNP improves throughput by 10x, while reducing latency by 10x. To the best of our knowledge, ClickNP is the first FPGA-accelerated platform for NFs, written completely in high-level language and achieves 40 Gbps line rate at any packet size.

CONCEPTS

1. INTRODUCTION

Modern multi-tenant datacenters provide shared infrastructure for hosting many different types of services from different customers (i.e., tenants) at a low cost. To ensure security and performance isolation, each tenant is deployed in a *virtualized network* environment. Flexible network functions (NFs) are required for datacenter operators to enforce isolation while simultaneously guaranteeing Service Level Agreements (SLAs).

Conventional hardware-based network appliances are not flexible, and almost all existing cloud providers, e.g., Microsoft, Amazon and VMware, have been deploying software-based NFs on servers to maximize the flexibility [23, 30]. However, software NFs have two fundamental limitations – both stem from the nature of software packet processing. First, processing packets in software has limited capacity. Existing software NFs usually require multiple cores to achieve 10 Gbps rate [33, 43]. But the latest network links have scaled up to 40~100 Gbps [11]. Although one could add more cores in a server, doing so adds significant cost, not only in terms of capital expense, but also more operational expense as they are burning significantly more energy. Second, processing packets in software incurs large, and highly variable latency. This latency may range from tens of microseconds to milliseconds [22, 33, 39]. For many low latency applications (e.g., stock trading), this inflated latency is unacceptable.

High performance NF implementations

Microboxes: High Performance NFV with Customizable, Asynchronous TCP Stacks and Dynamic Subscriptions

Guyue Liu*, Yuxin Ren*, Mykola Yurchenko*,
K.K. Ramakrishnan†, Timothy Wood*

*George Washington University, †University of California, Riverside

FlowBlaze: Stateful Packet Processing in Hardware

Salvatore Pontarelli^{1,2}, Roberto Bifulco³, Marco Bonola^{1,2}, Carmelo Cascone⁴,
Marco Spaziani^{2,5}, Valerio Bruschi^{2,5}, Davide Sanvito⁶, Giuseppe Siracusano³,
Antonio Capone⁶, Michio Honda³, Felipe Huici³ and Giuseppe Bianchi^{2,5}

¹Axbryd, ²CNIT, ³NEC Laboratories Europe, ⁴Open Networking Foundation,
⁵University of Rome Tor Vergata, ⁶Politecnico di Milano

Abstract

While programmers handle growing network traffic, yet simple abstraction in hardware remains a problem with FlowBlaze. Stateful packet processing is based on FlowBlaze to leverage expressive, supporting functions, and easy to state transition issues from FlowBlaze on a Network (in the order of thousands of flows) for even higher performance and software flexibility available.

1 Introduction

Network infrastructure network functions and server load balance such as access control examples. Given the need to continue to have turned to

NetBricks: Taking the V out of NFV

Aurojit Panda† Sangjin Han† Keon Jang‡ Melvin Walls† Sylvia Ratnasamy† Scott Shenker†*
† UC Berkeley ‡ Google * ICSI

Abstract

The move from hardware middleboxes to software network functions, as advocated by NFV, has proven more challenging than expected. Developing new NFs remains a tedious process, requiring that developers repeatedly rediscover and reapply the same set of optimizations, while current techniques for providing isolation between NFs (using VMs or containers) incur high performance overheads. In this paper we describe NetBricks, a new NFV framework that takes both these problems. For building NFs we take inspiration from modern data analytics frameworks (e.g., Spark and Dryad) and build a small set of customizable network processing elements. We also embrace type checking and safe runtimes to provide isolation in software, rather than rely on hardware isolation. NetBricks provides the same memory isolation as containers and VMs, without incurring the same performance penalties. To improve I/O

standard tools for managing VMs; (c) faster development, which now requires writing software that runs on commodity hardware; and (d) reduced costs by consolidating several NFs on a single machine. However, despite these promised advances, there has been little progress towards large-scale NF deployments. Our discussions with three major carriers revealed that they are only just beginning small scale test deployments (with 10-100s of customers) using simple NFs e.g., firewalls and NATs.

The move from hardware middleboxes to software NFs was supposed to speed innovation, so why has progress been so slow? We believe this delay is because traditional approaches for both *building* and *running* NFs are a poor match for carrier networks, which have the following requirements: *performance*, NF deployments should be able to provide per-packet latencies on the order of 10s of μ s, and throughput on the order of 10s of Gbps; *efficiency*, it should be possible to consolidate several NFs on a sin-

mOS: A Reusable Networking Stack for Flow Monitoring Middleboxes

Muhammad Jamshed, YoungGyoun Moon, Donghwi Kim, Dongsu Han, and Kyoungsoo Park
School of Electrical Engineering, KAIST

ClickNP: Highly Flexible and High Performance Network Processing with Reconfigurable Hardware

Bojie Li^{§†} Kun Tan[†] Layong (Larry) Luo[†] Yanqing Peng^{*†} Renqian Luo^{§†}
Ningyi Xu[†] Yongqiang Xiong[†] Peng Cheng[†] Enhong Chen[§]
[†]Microsoft Research [§]USTC [‡]Microsoft ^{*}SJTU

ABSTRACT

Flexible software network functions (NFs) are crucial elements to enable multi-tenancy in the clouds. However, packet processing on a commodity server has capacity and induces high latency. While software can scale out using more servers, doing so adds significant cost. This paper focuses on accelerating NFs with reconfigurable hardware, i.e., FPGA, which is now a mature and inexpensive for datacenters. However, predominately programmed using low-level hardware description languages (HDLs), which are hard to code and difficult to debug. More importantly, HDLs are almost impossible for most software programmers. This paper presents a FPGA-accelerated platform for highly flexible performance NFs with commodity servers. ClickNP is flexible as it is completely programmable using C-like languages, and exposes a modular programming abstraction that resembles Click Modular Router. ClickNP achieves high performance. Our prototype NFs show that they process traffic at up to 200 million packets per second with low latency (< 2 μ s). Compared to existing software NFs, with FPGA, ClickNP improves throughput by 10x, while reducing latency by 10x. To the best of our knowledge, ClickNP is the first FPGA-accelerated platform for NFs, written completely in high-level language and achieves 40 Gbps line rate at any packet size.

1. INTRODUCTION

Modern multi-tenant datacenters provide shared infrastructure for hosting many different types of services from different customers (i.e., tenants) at a low cost. To ensure security and performance isolation, each tenant is deployed in a *virtualized network* environment. Flexible network functions (NFs) are required for datacenter operators to enforce isolation while simultaneously guaranteeing Service Level Agreements (SLAs).

Conventional hardware-based network appliances are not flexible, and almost all existing cloud providers, e.g., Microsoft, Amazon and VMware, have been deploying software-based NFs on servers to maximize the flexibility [23, 30]. However, software NFs have two fundamental limitations – both stem from the nature of software packet processing. First, processing packets in software has limited capacity. Existing software NFs usually require multiple cores to achieve 10 Gbps rate [33, 43]. But the latest network links have scaled up to 40~100 Gbps [11]. Although one could add more cores in a server, doing so adds significant cost, not only in terms of capital expense, but also more operational expense as they are burning significantly more energy. Second, processing packets in software incurs large, and highly variable latency. This latency may range from tens of microseconds to milliseconds [22, 33, 39]. For many low latency applications (e.g., stock trading), this inflated latency is unacceptable.

NetBricks: Taking the V out of NFV

OSDI'16

Slides borrowed from the OSDI talk

NFV Requirements

- **High Packet Rates:** Must keep up with line rate which is $> 10\text{MPPS}$
- **Low Latency:** Used for applications like VoIP and video conferencing
- **Support NF Chaining:** Packets go through sequence of NFs



Challenges for NFV

- **Running NFs:**
 - Isolation and Performance
- **Building NFs:**
 - High-level Programming and Performance

Challenges for NFV

- **Running NFs:**
 - Isolation and Performance
- **Building NFs:**
 - High-level Programming and Performance

Isolation

- **Memory Isolation:**
 - Each NF's memory cannot be accessed by other NFs.
- **Packet Isolation:**
 - When chained, each NF processes packets in isolation.
- **Performance Isolation:**
 - One NF does not affect another's performance.

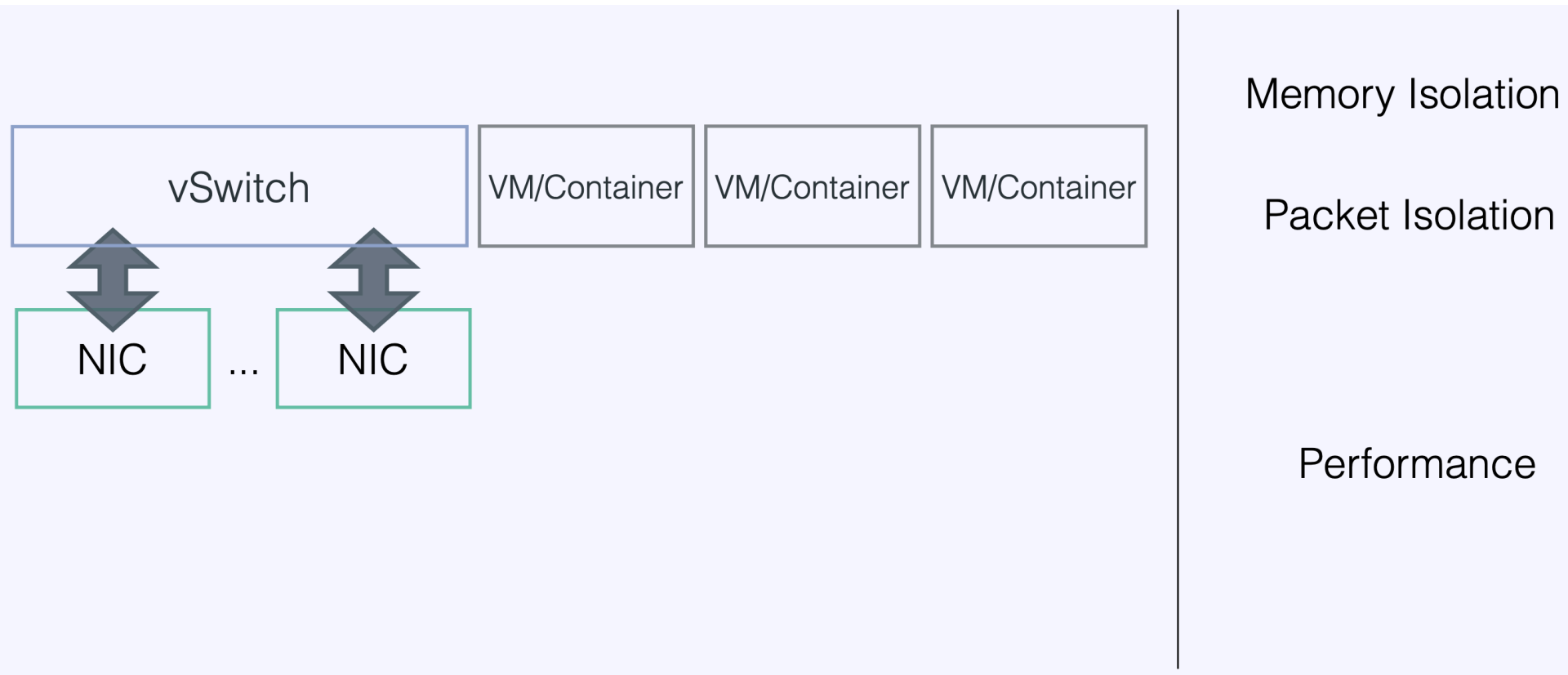
Isolation

- **Memory Isolation:**
 - Each NF's memory cannot be accessed by other NFs.
- **Packet Isolation:**
 - When chained, each NF processes packets in isolation.
- **Performance Isolation:**
 - One NF does not affect another's performance.
 - *Achieved by scheduling policies (left largely to future work)*

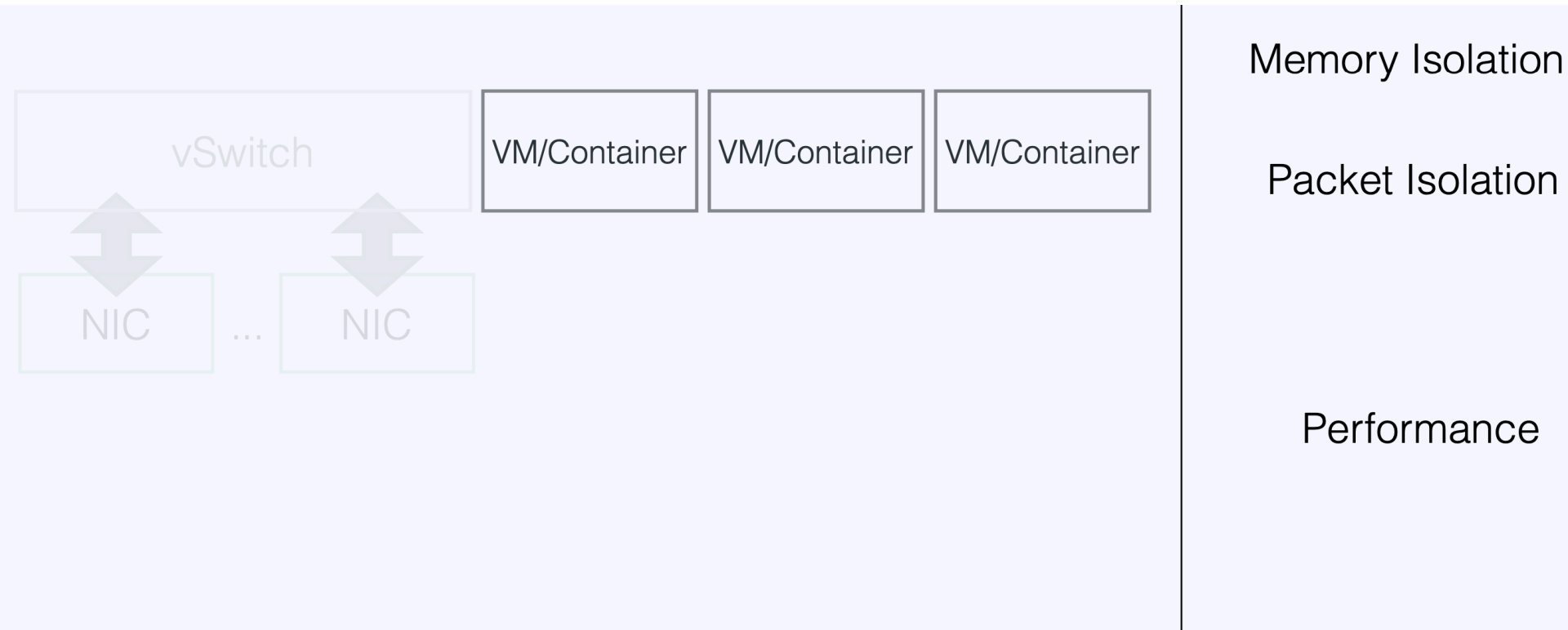
Isolation

- **Memory Isolation:**
 - Each NF's memory cannot be accessed by other NFs.
- **Packet Isolation:**
 - When chained, each NF processes packets in isolation.
- **Performance Isolation:**
 - One NF does not affect another's performance.
 - *Achieved by scheduling policies (left largely to future work)*

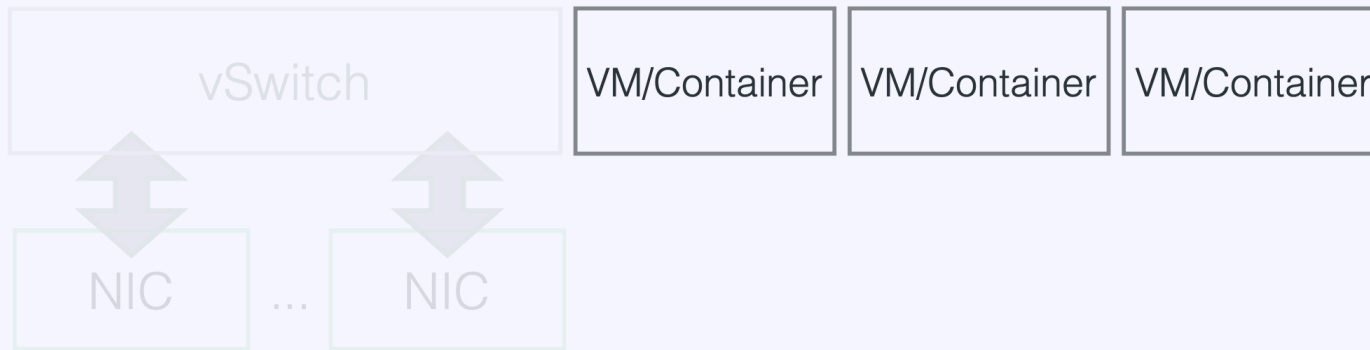
Current Solution



Current Solution



Current Solution

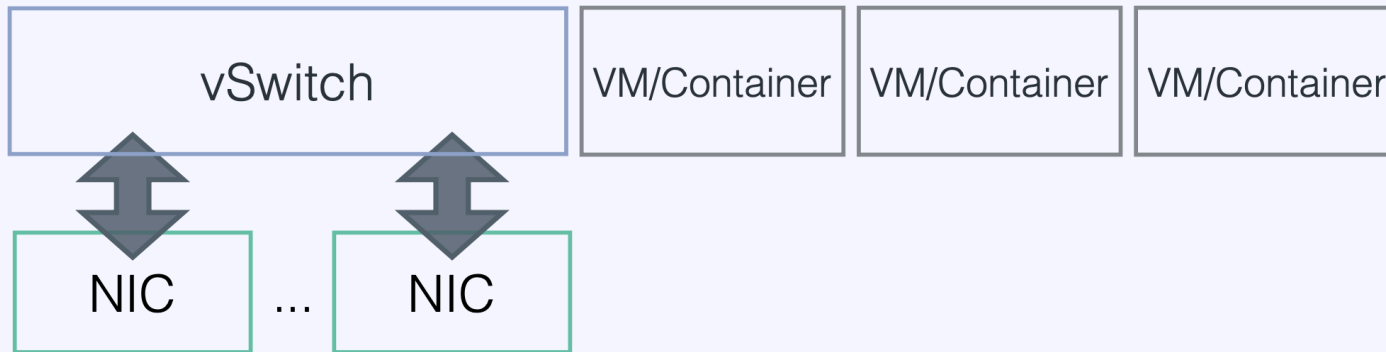


✓ Memory Isolation

Packet Isolation

Performance

Current Solution

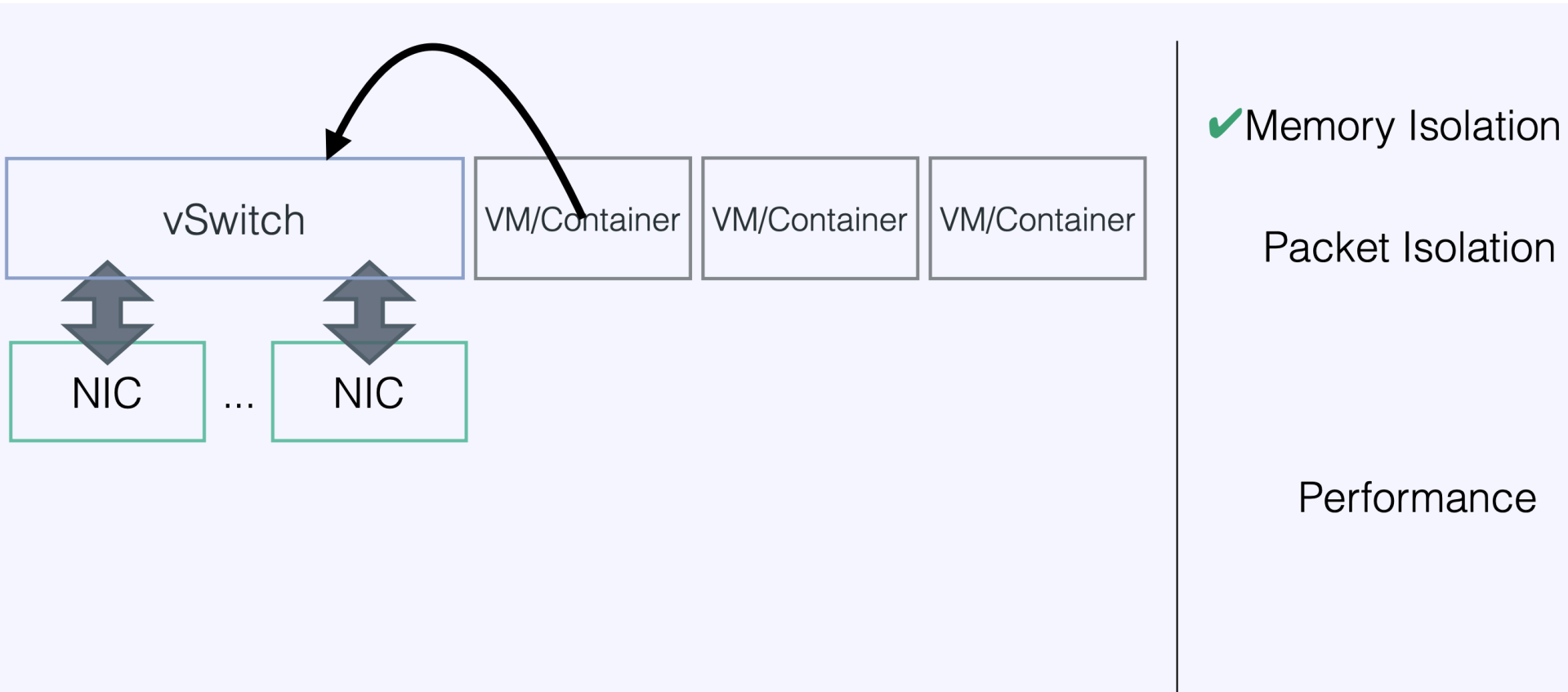


✓ Memory Isolation

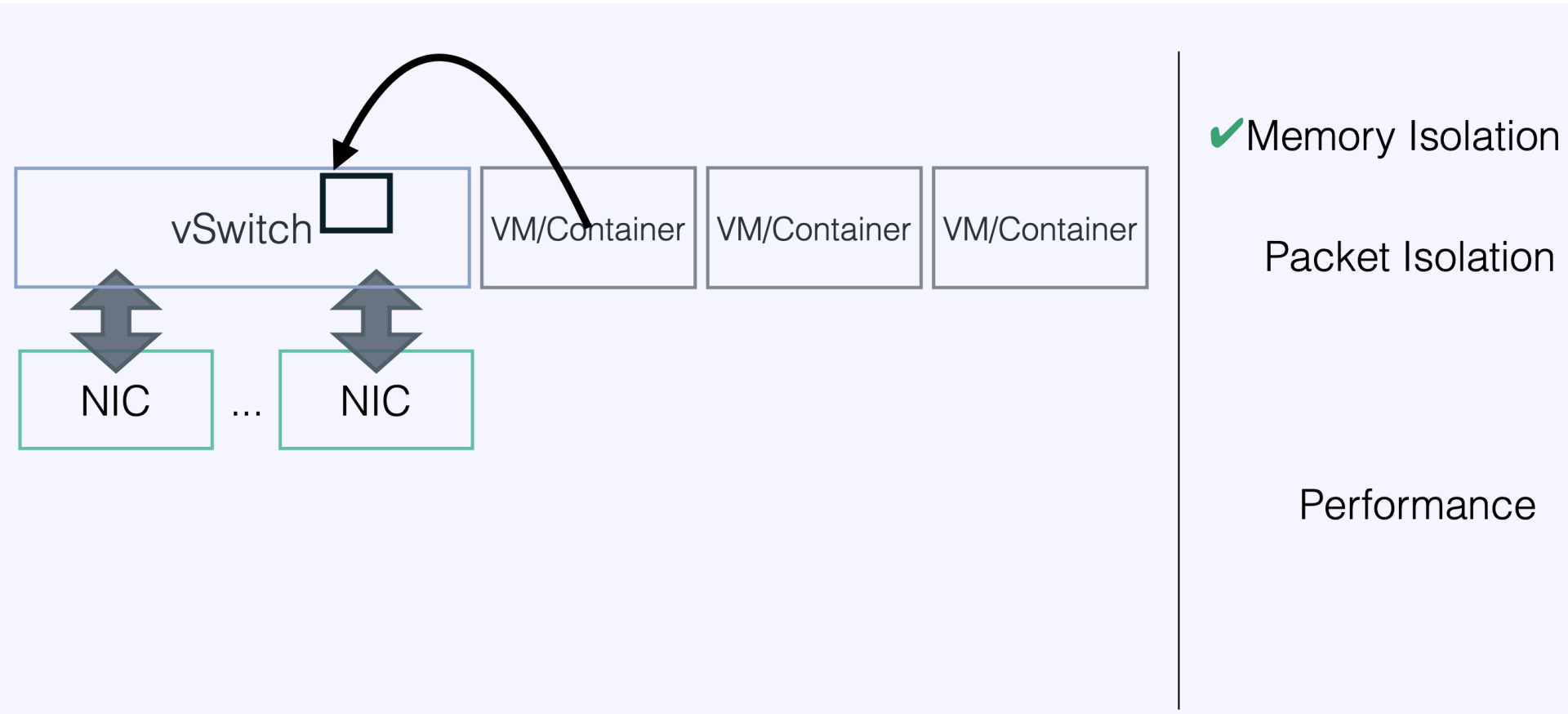
Packet Isolation

Performance

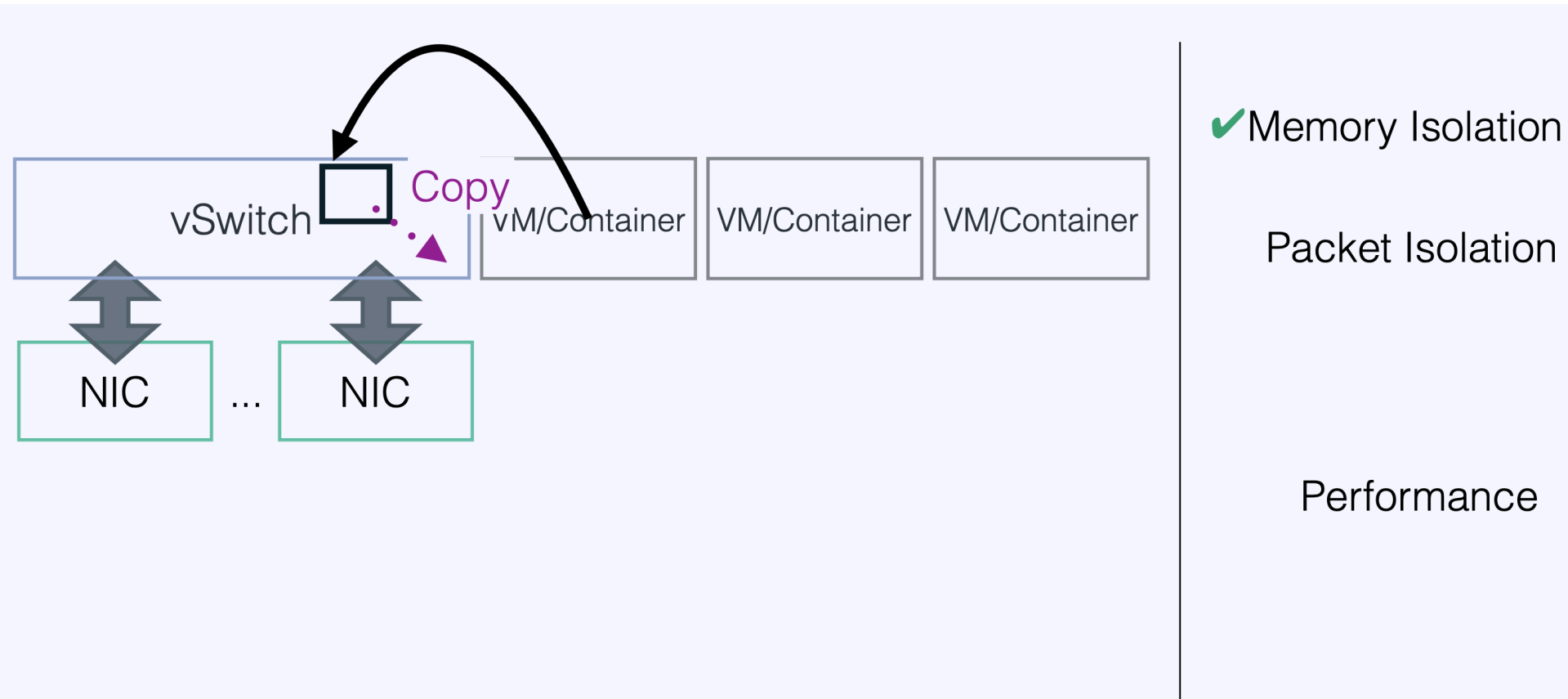
Current Solution



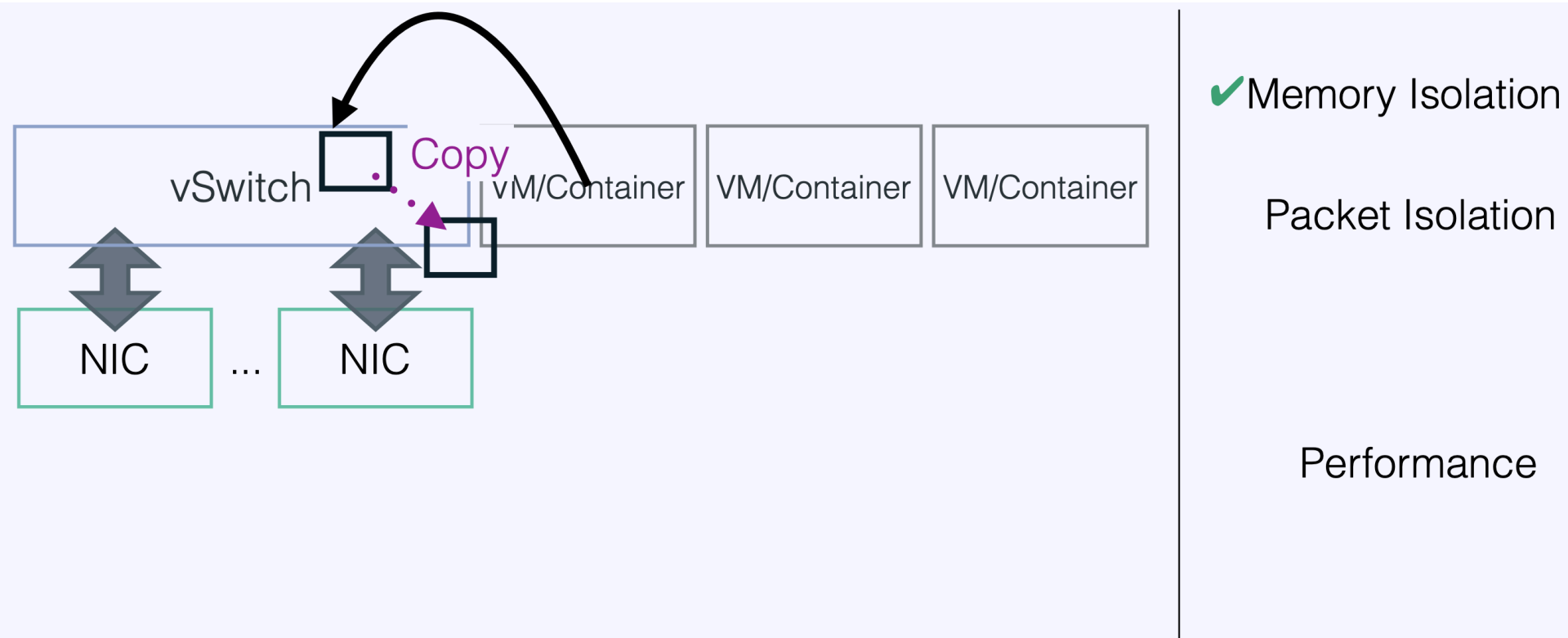
Current Solution



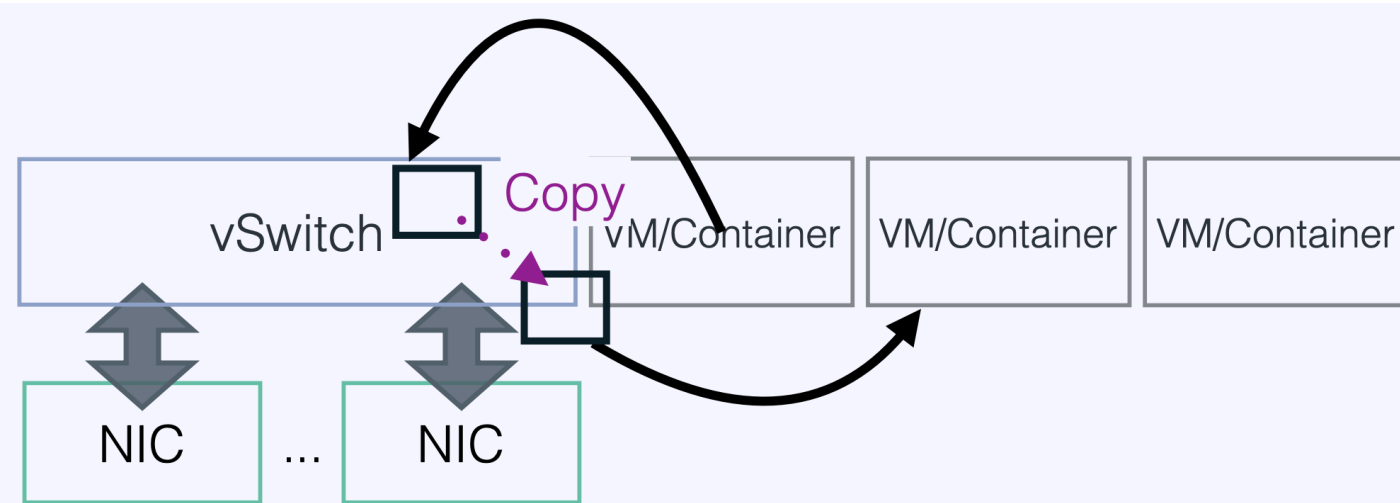
Current Solution



Current Solution



Current Solution

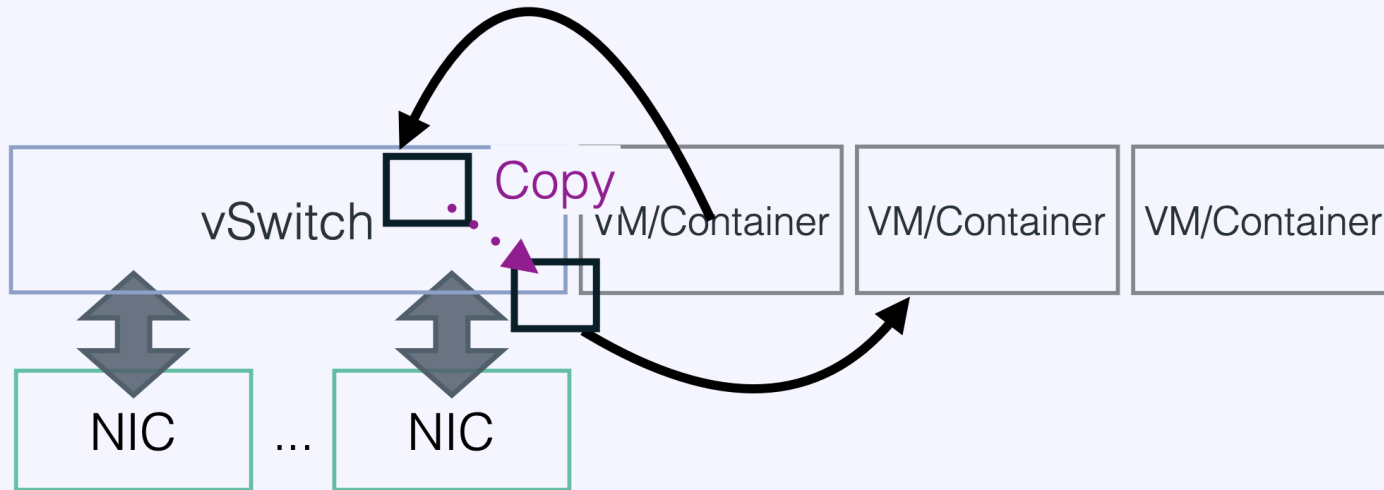


✓ Memory Isolation

Packet Isolation

Performance

Current Solution

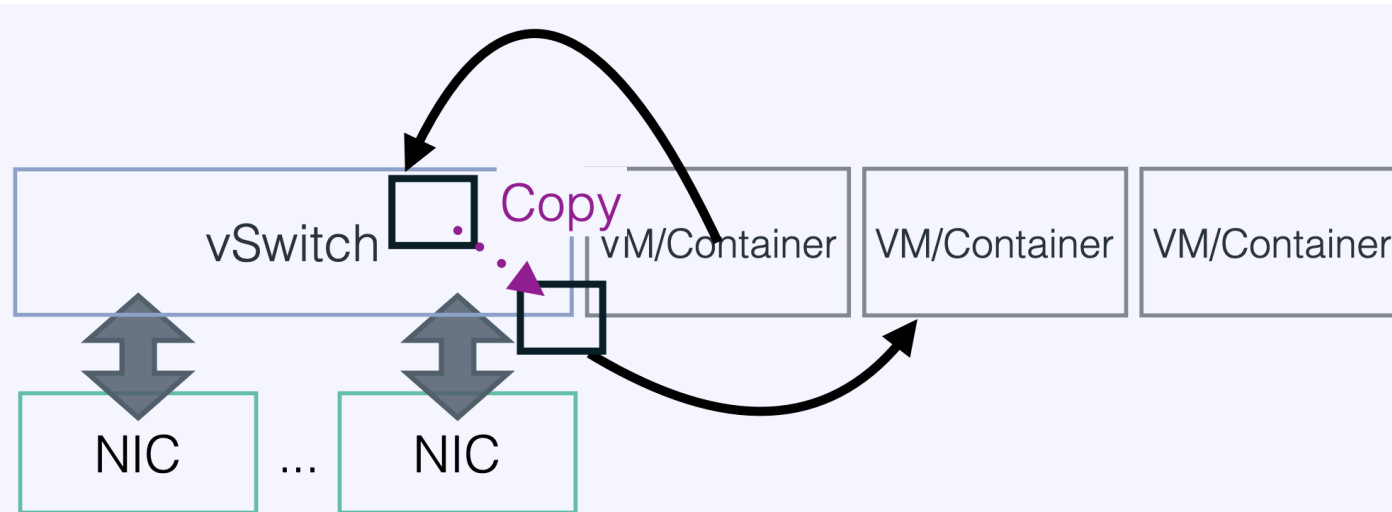


✓ Memory Isolation

✓ Packet Isolation

Performance

Current Solution

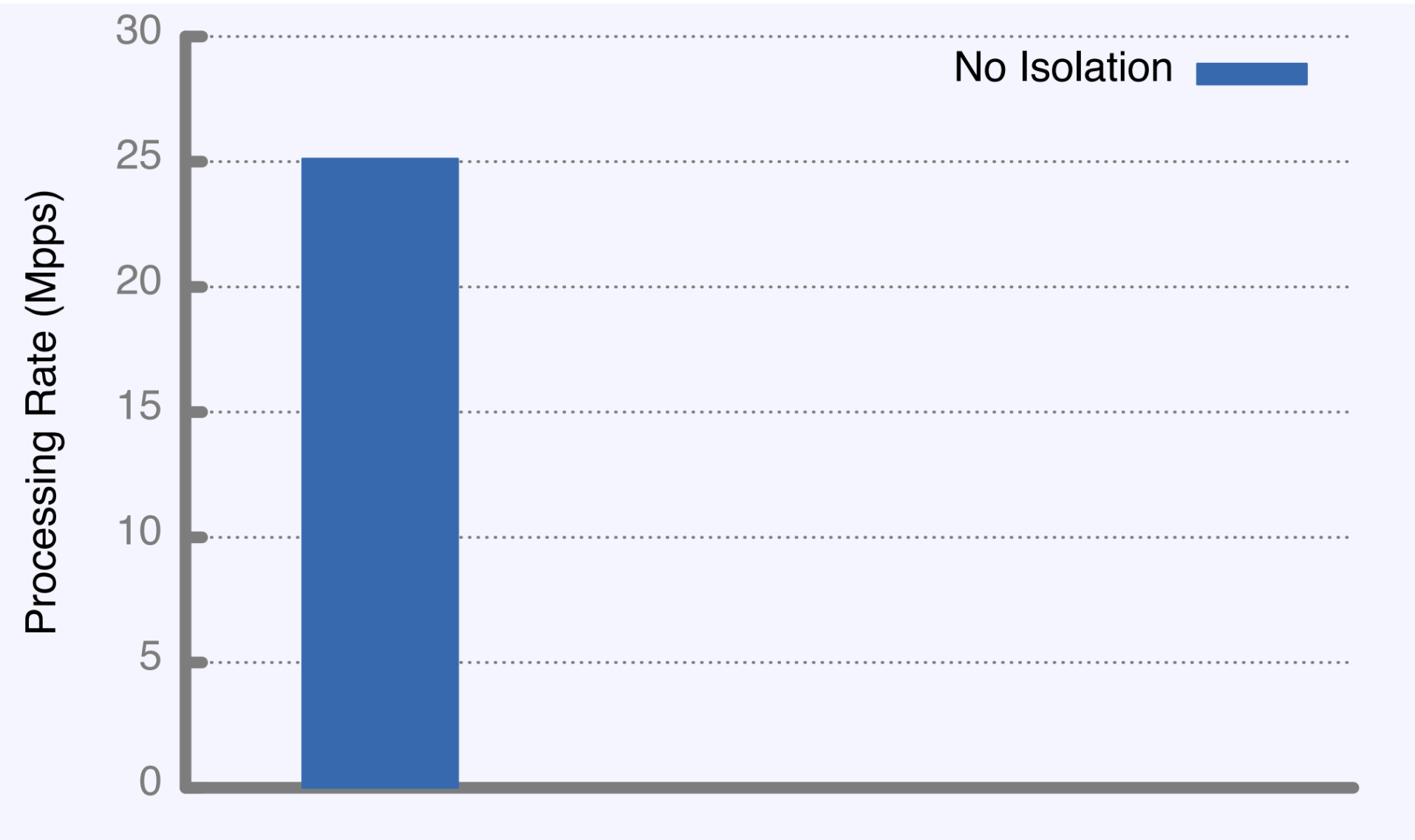


✓ Memory Isolation

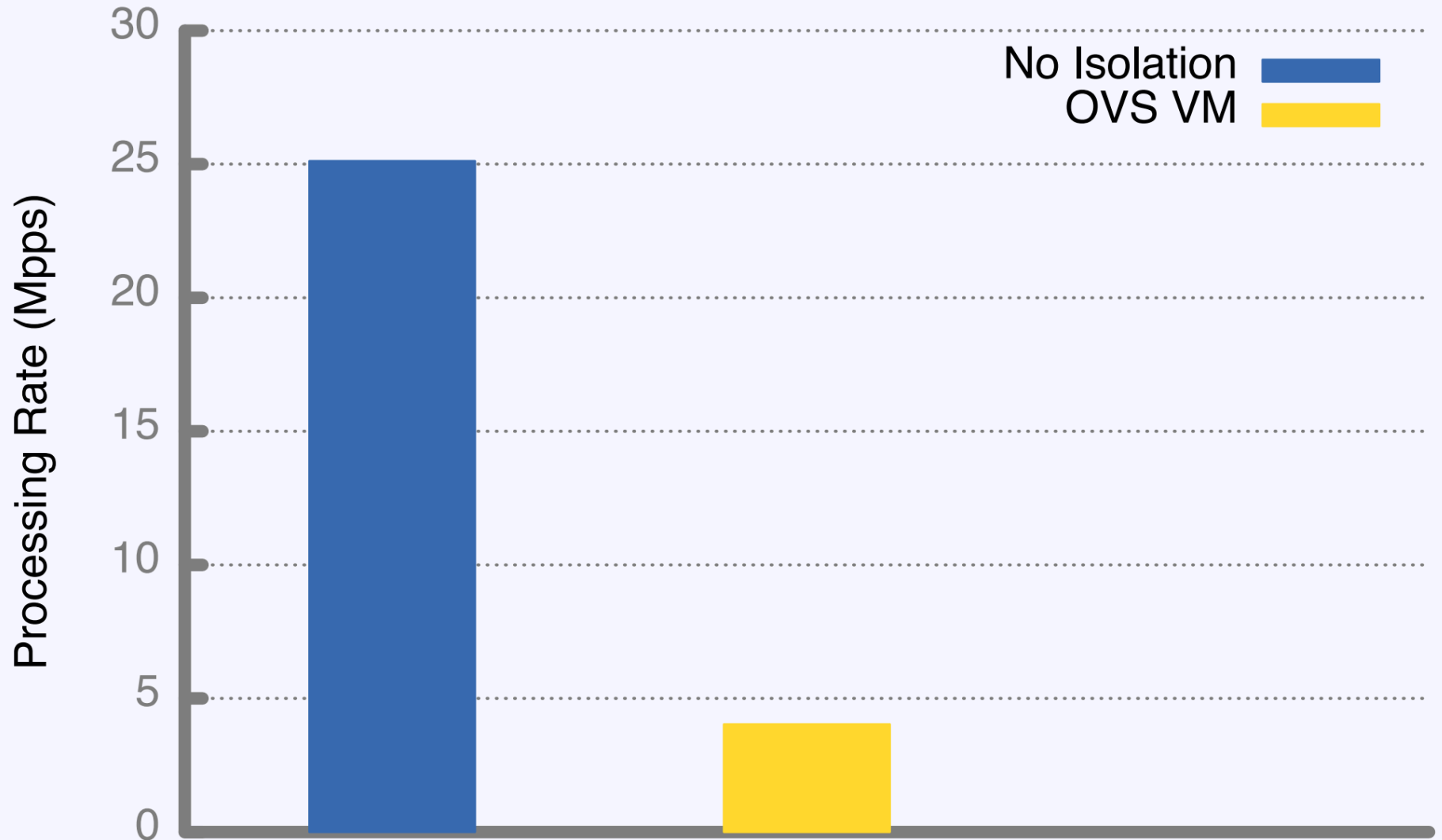
✓ Packet Isolation

✗ Performance

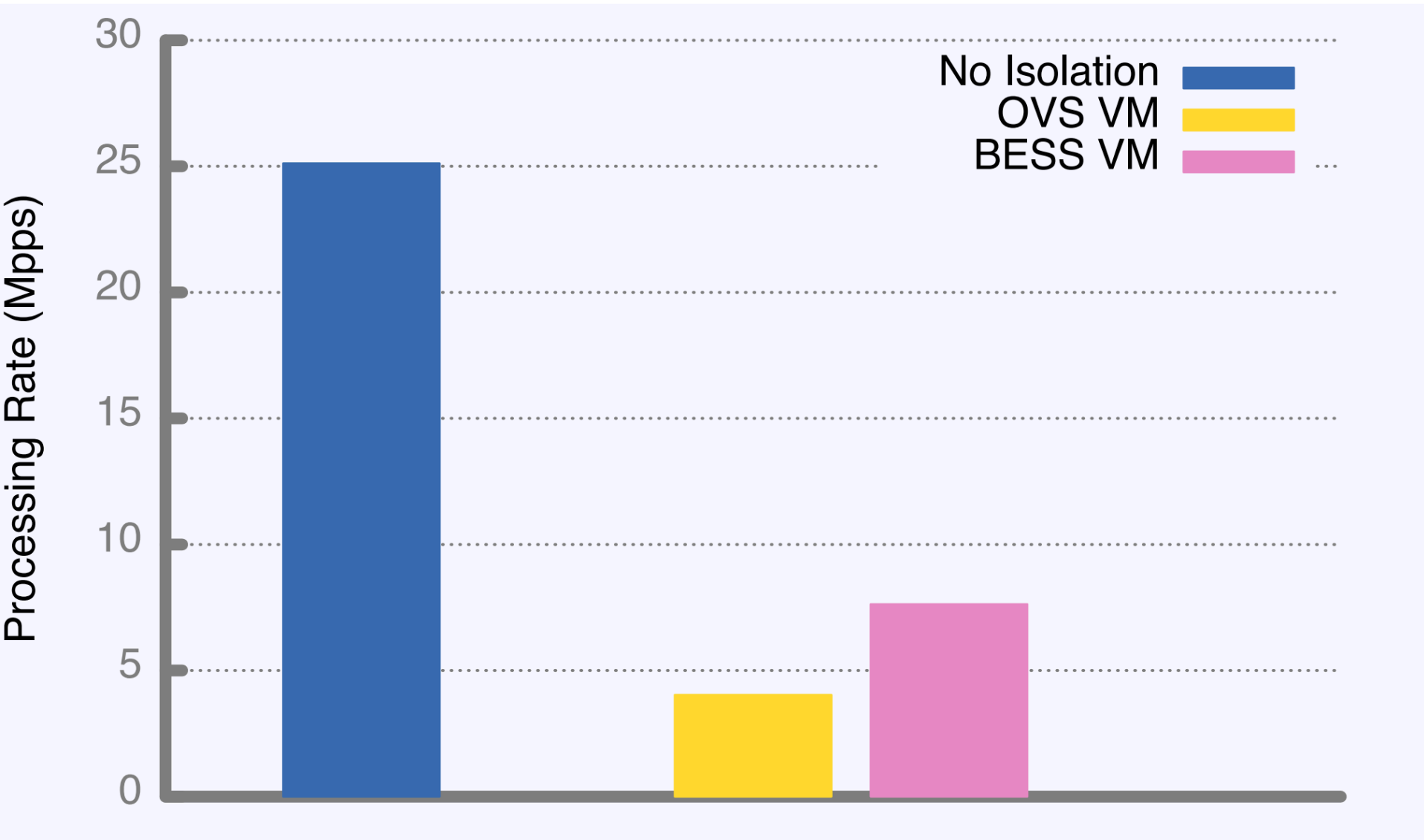
Isolation costs Performance



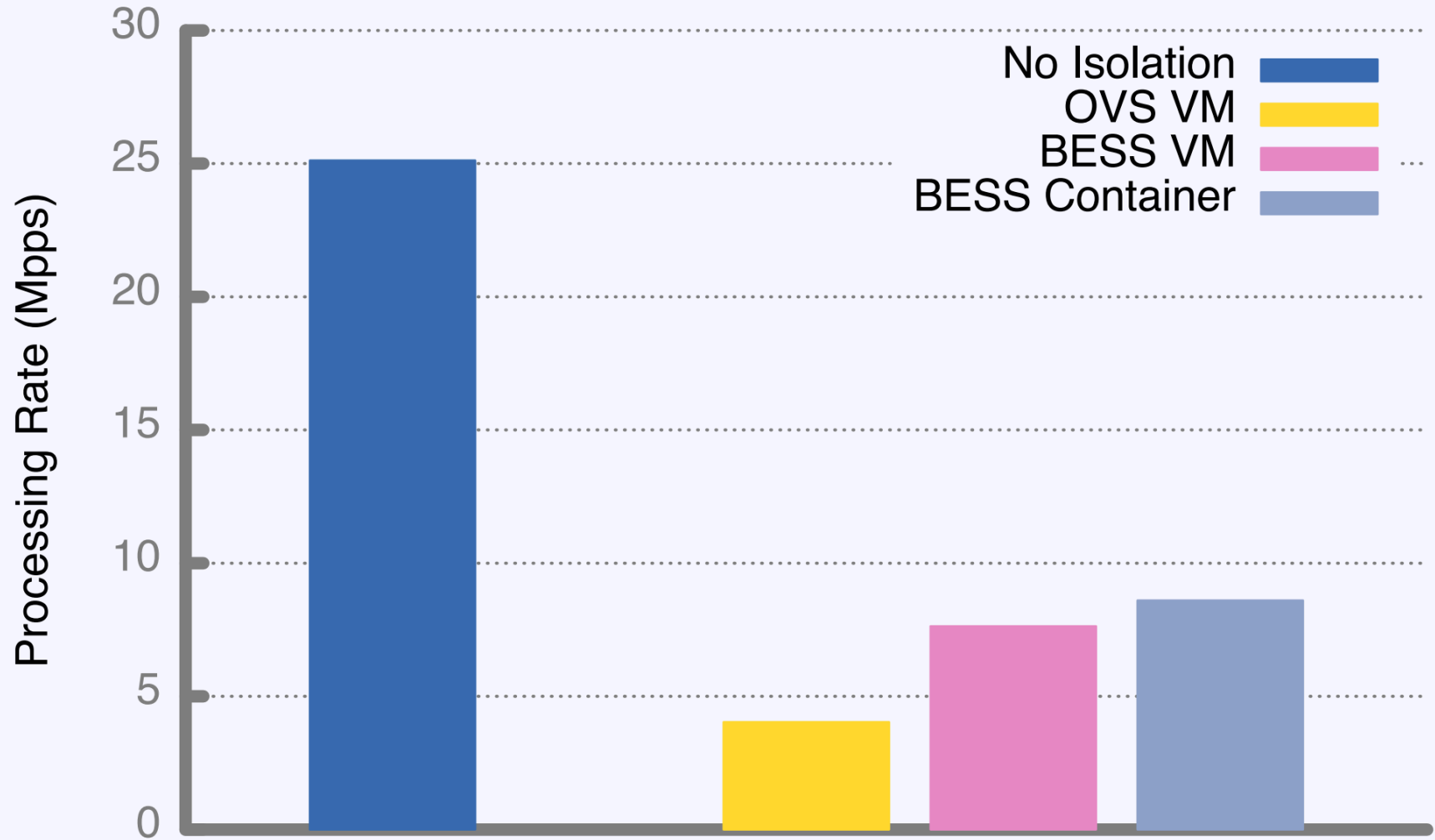
Isolation costs Performance



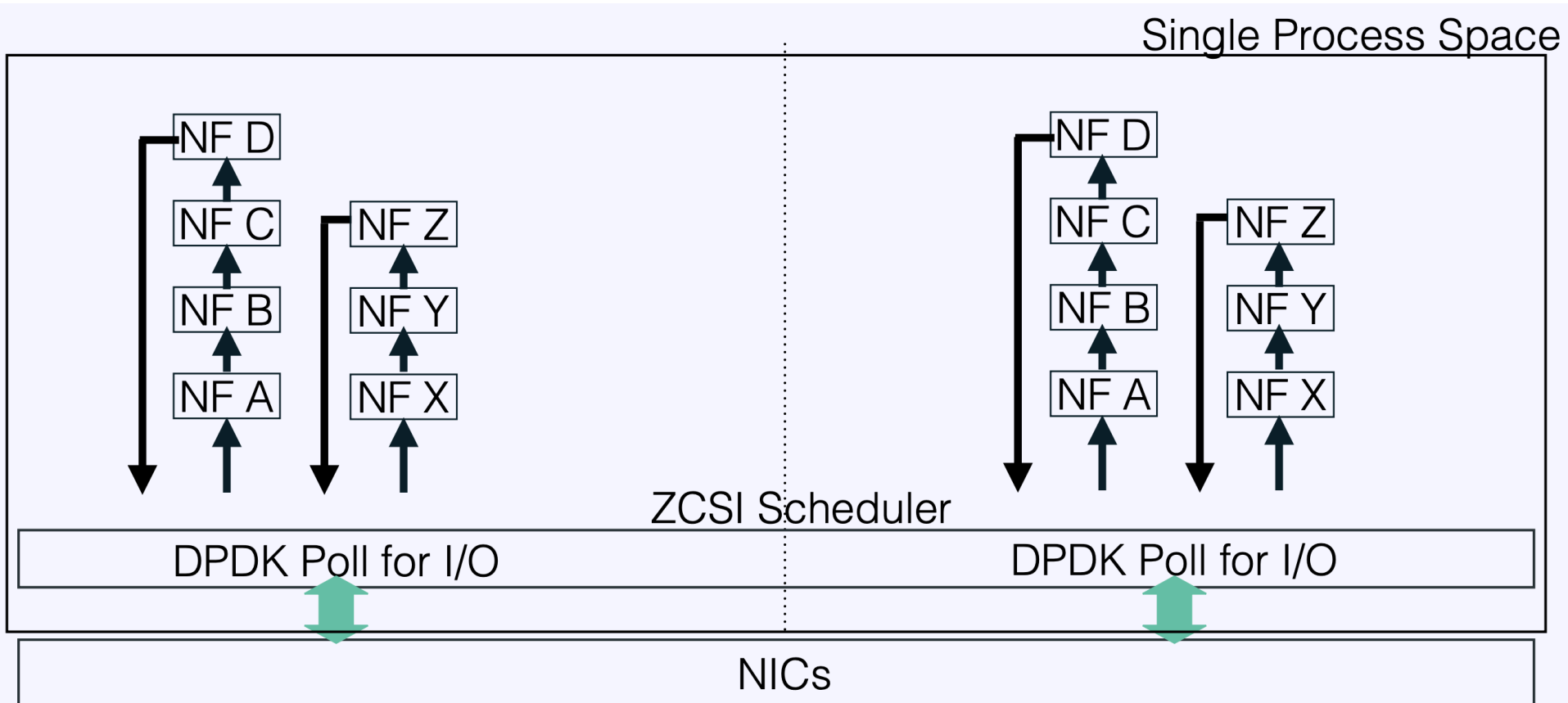
Isolation costs Performance



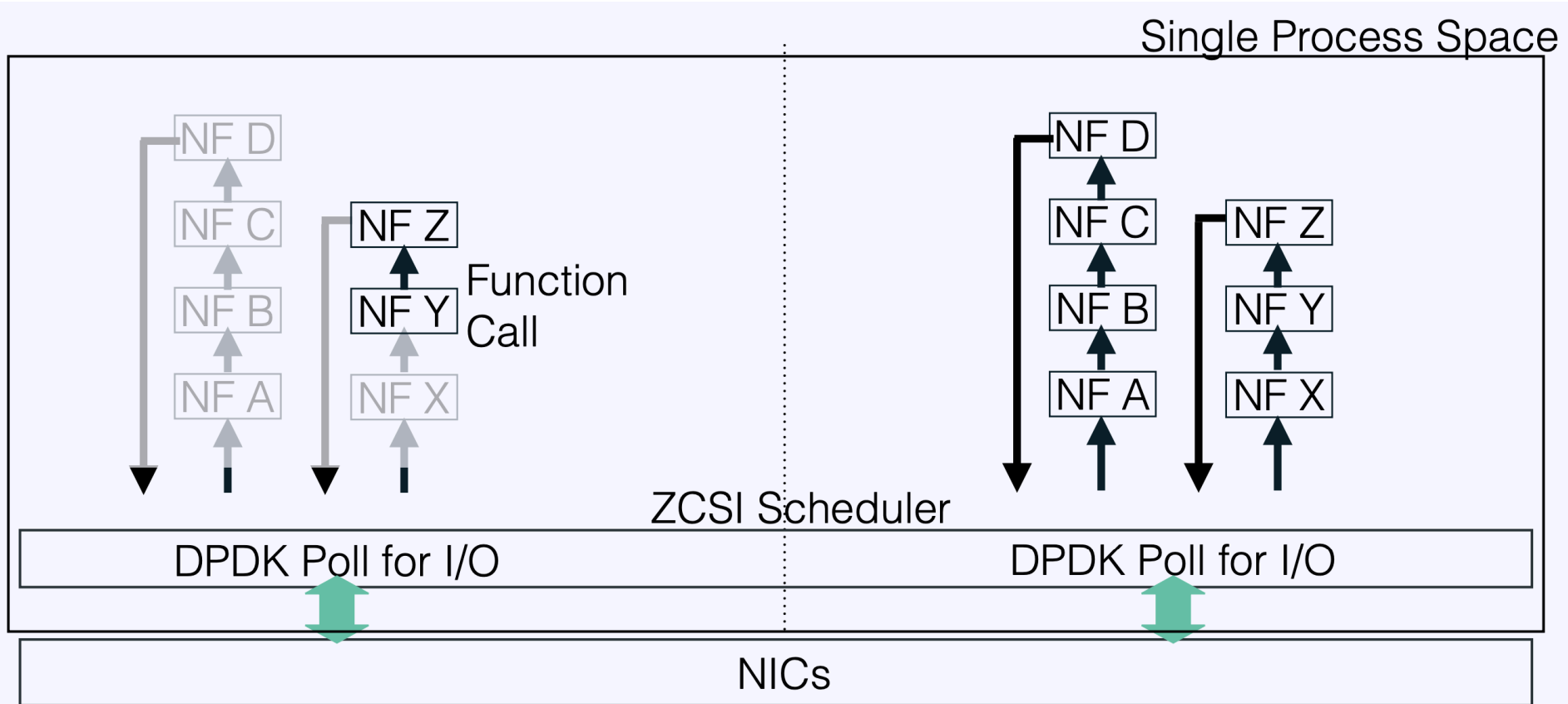
Isolation costs Performance



NetBricks Runtime Architecture



NetBricks Runtime Architecture



ZCSI: Zero Copy Soft Isolation

- VMs and containers impose cost on packets crossing isolation boundaries.
- Insight: Use type checking (compile time) and runtime checks for isolation.
- Isolation costs largely paid at compile time (small runtime costs).

NetBricks Approach

- Disallow pointer arithmetic in NF code: use safe subset of languages.
- Type checks + array bounds checking provide memory isolation.
- Build on unique types for packet isolation.
 - Unique types ensure references destroyed after certain calls.
 - Ensure only one NF has a reference to a packet.
 - Enables zero copy packet I/O.
- All of these features implemented on top of **Rust**.

Software Isolation

- Provides memory and packet isolation.
- Improved consolidation: multiple NFs can share a core.
 - Function call to NF (\sim few cycles) vs context switch ($\sim 1\mu\text{s}$).
- Reduce memory and cache pressure.
 - Zero copy I/O \Rightarrow do not need to copy packets around.

Challenges for NFV

- Running NFs:
 - Isolation and Performance
- **Building NFs:**
 - High-level Programming and Performance

How to write NFs?

- **Current:** NF writers concerned about meeting performance targets
 - Low level abstractions (I/O, cache aware data structures) and low level code.
- Spend lots of time optimizing how abstractions are used to get performance.
- **Observation:** NFs exhibit common patterns: abstract and optimize these.
- Analogous to what happened in other areas.
 - MPI to Map Reduce, etc.

Abstractions

Packet Processing

Parse/Deparse
Transform
Filter

Control Flow

Group By
Shuffle
Merge

Byte Stream

Window
Packetize

State

Bounded
Consistency

Behavior of these abstractions dictated by user-defined functions (UDFs)

Example NF

- Maglev: Load balancer from Google (NSDI'16).
- NetBricks implementation: 105 lines, 2 hours of time.
- Comparable performance to optimized code

Conclusion

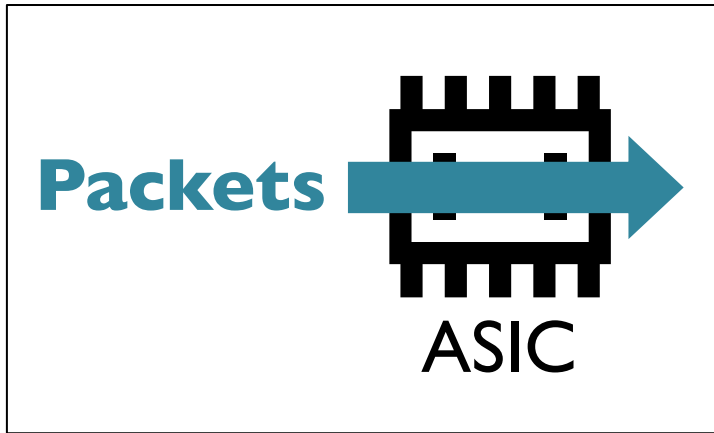
- Software isolation is necessary for high performance NFV.
 - Type checking + bound checking + unique types.
- Performance is not anathema to high-level programming
 - Abstract operators + UDF simplify development.

Your thoughts?

- What did you like about the paper?
- What are its limitations?

Evolution of middleboxes

Dedicated hardware

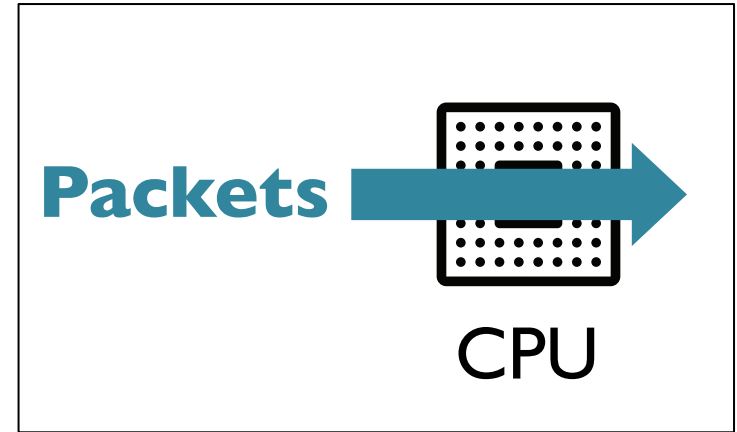


Middleboxes

*Need for
flexibility*



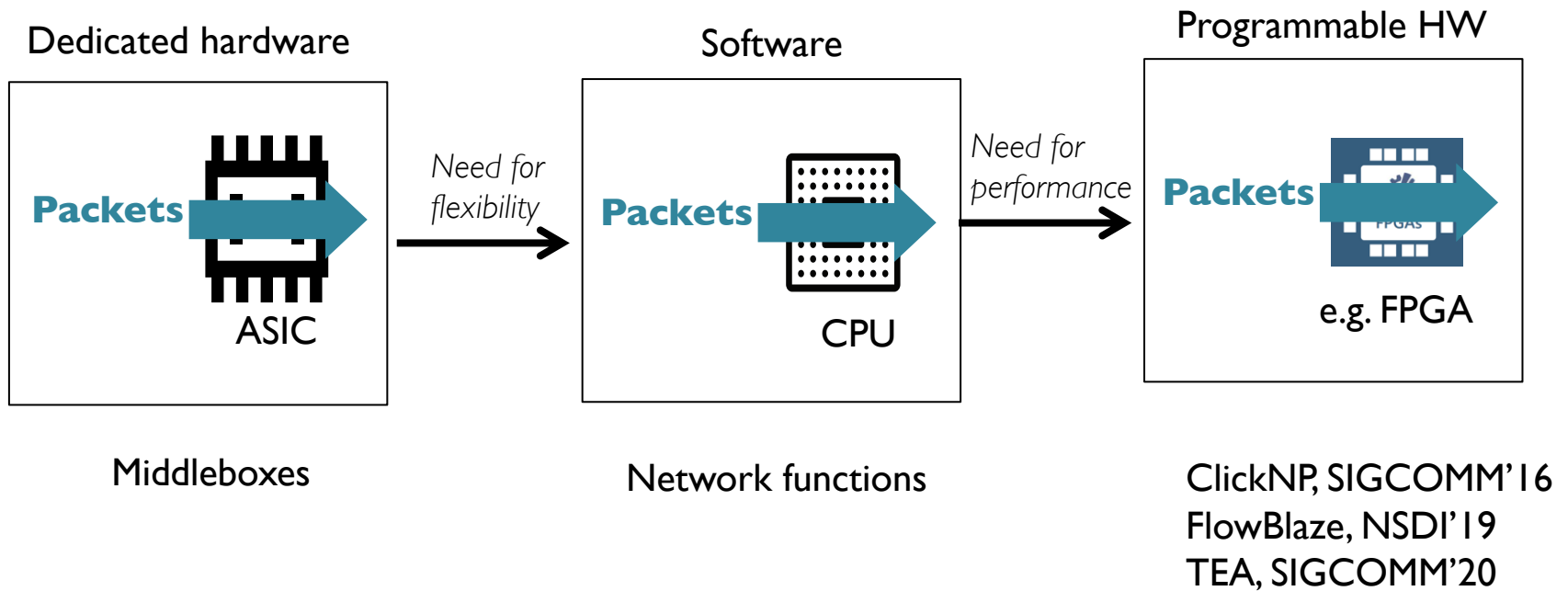
Software



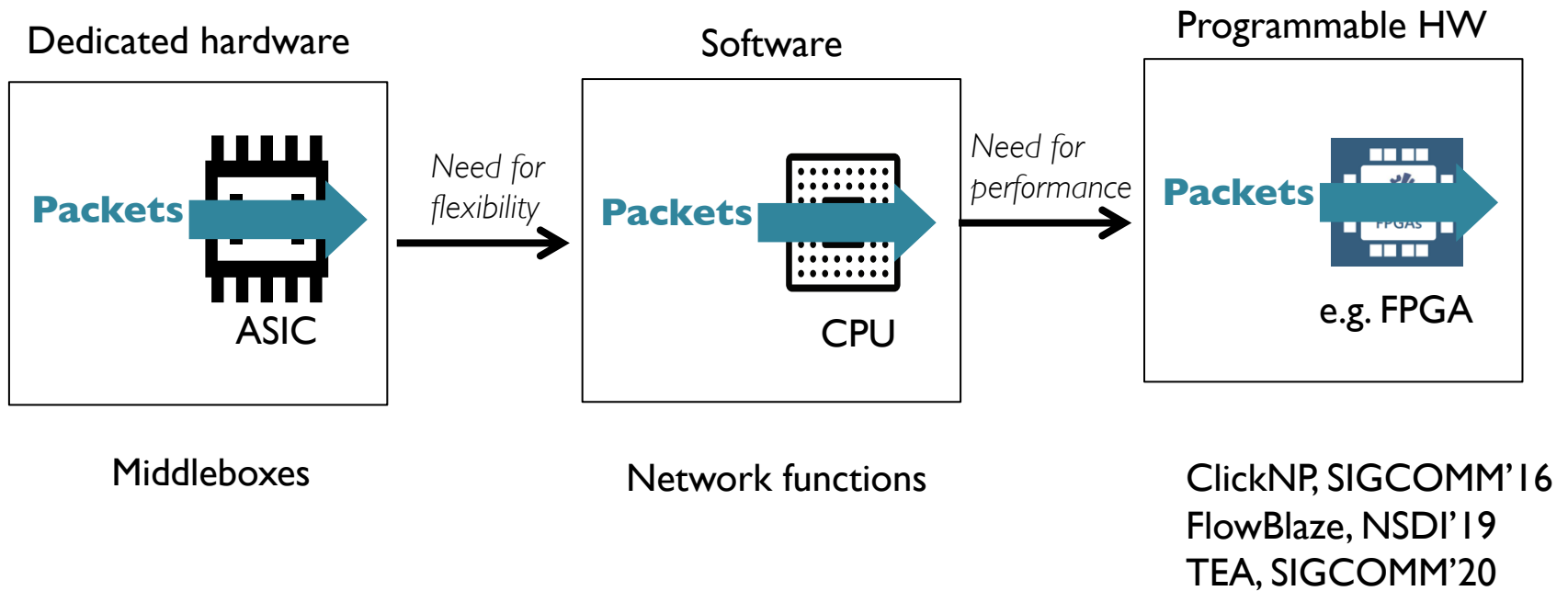
Network functions

**contents of the slide borrowed from talks given by Aurojit Panda, NYU*

Evolution of middleboxes



Evolution of middleboxes



Upcoming classes

- No class on Nov 16th.
 - Optional reading on network edge.
 - 5G systems microbook is a useful resource (linked on the webpage).
- Nov 18 and Nov 30: your presentations (each up to 6mins long).
- Friday, Dec 2nd:
 - Wrapping up
 - Quiz for bonus class participation scores.
- Monday, Dec 5th: Final projects due (will update on course website).
- Dec 7th: Final project presentation. Details TBA.