Designing Customizable Abstractions for High Performance Network Functions

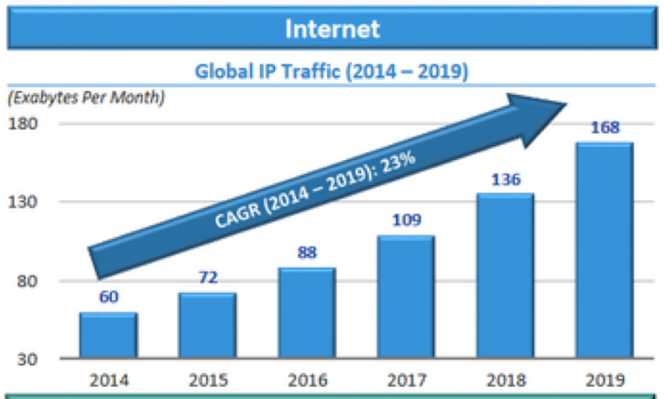
Guyue (Grace) Liu

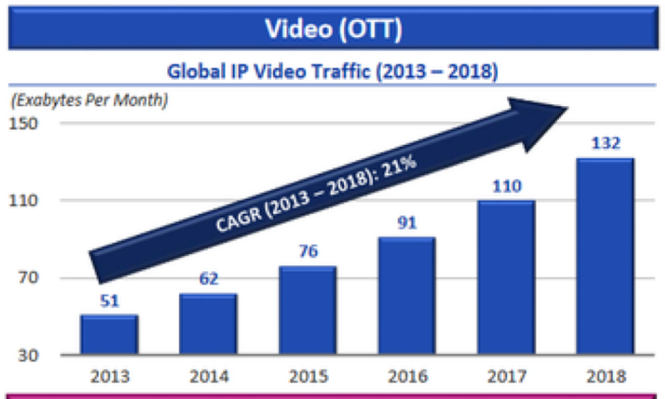
Committee:

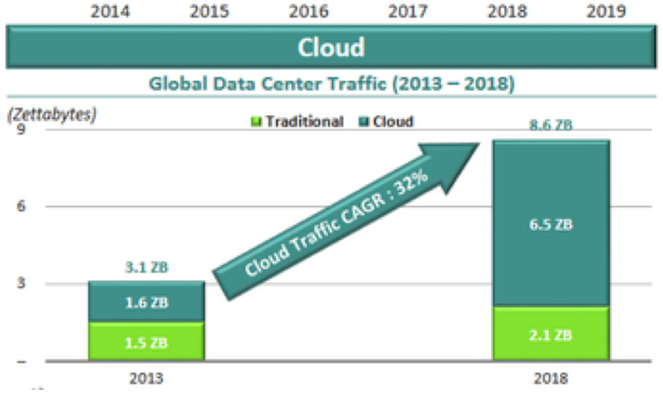
Timothy Wood
Gabriel Parmer
Xiuzhen Cheng
K. K. Ramakrishnan

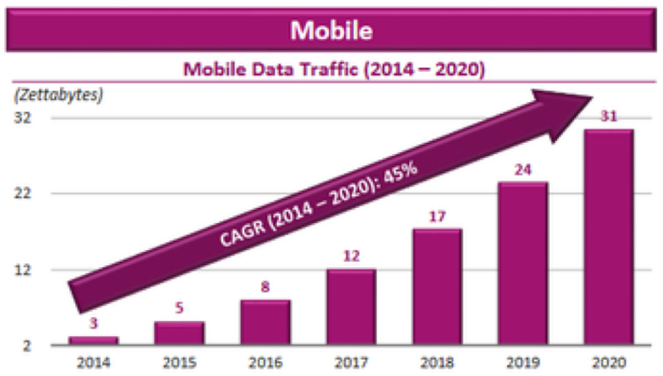
Cloud Data Centers

- Cloud Computing has been widely adopted by enterprises, startups, and individual users.
- Data Centers are growing BIG, FAST, and COMPLEX
  - 100,000+ servers in a single warehouse
  - 900+ trillion requests/day (Azure)





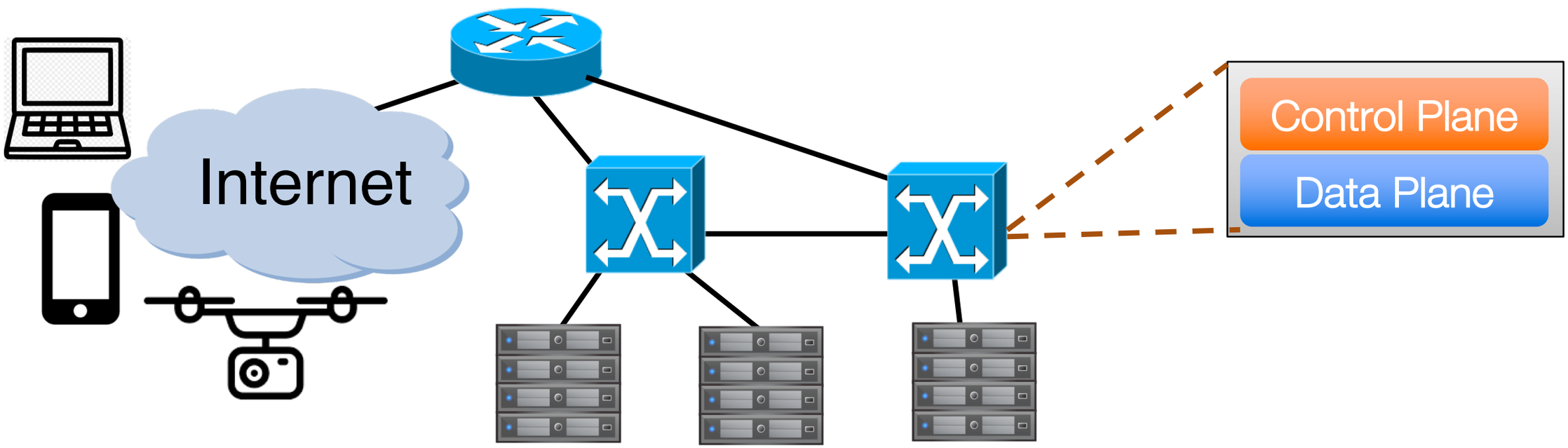




Alibaba Cloud
aliyun.com

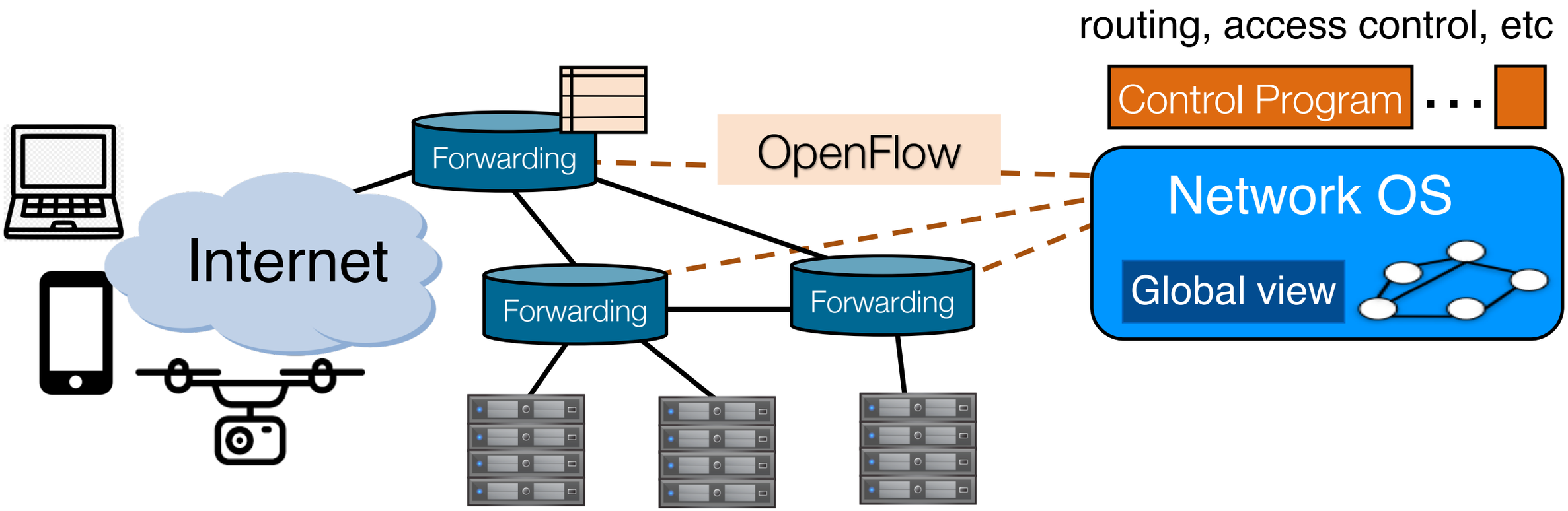
Networks are Changing

- Increasing data volume, speed, and diversity poses new challenges for the network infrastructure
- Networks are responsible for routing packets
- Traditional Networks are hard to manage and evolve
  - Control plane is tightly coupled with data plane in the hardware box
  - Closed proprietary with no open interface to manage routers and switches



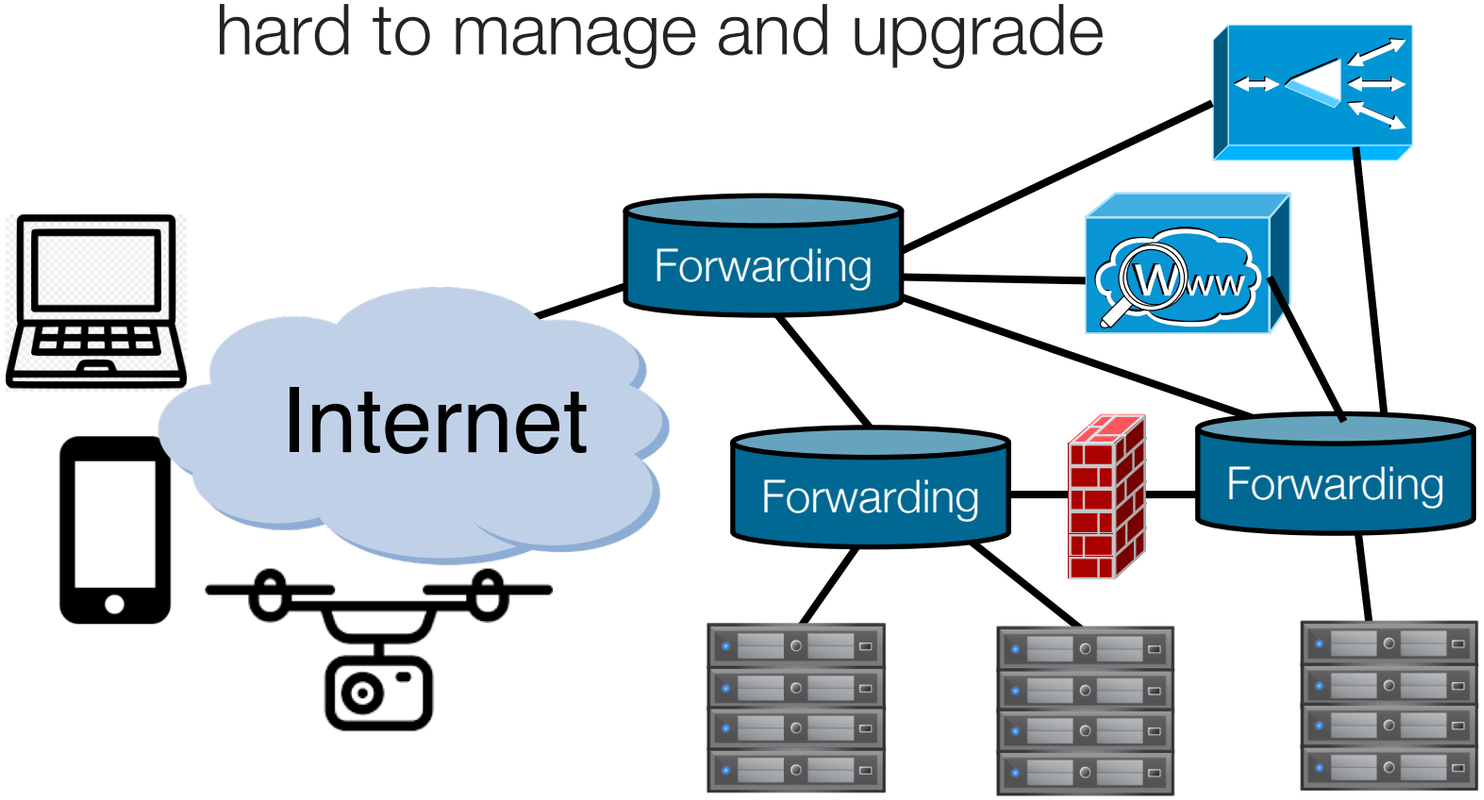
Software Defined Networking

- Software Defined Networking (SDN)
  - Decouples control plane from data plane
  - Control Programs: express operator goals and implemented on global network view abstraction
  - Network OS: gathers information for global network view
  - Switches/Routers: forward packets based on the rules given by NOS
  - Interface: open and vendor agnostic interface (e.g. OpenFlow)



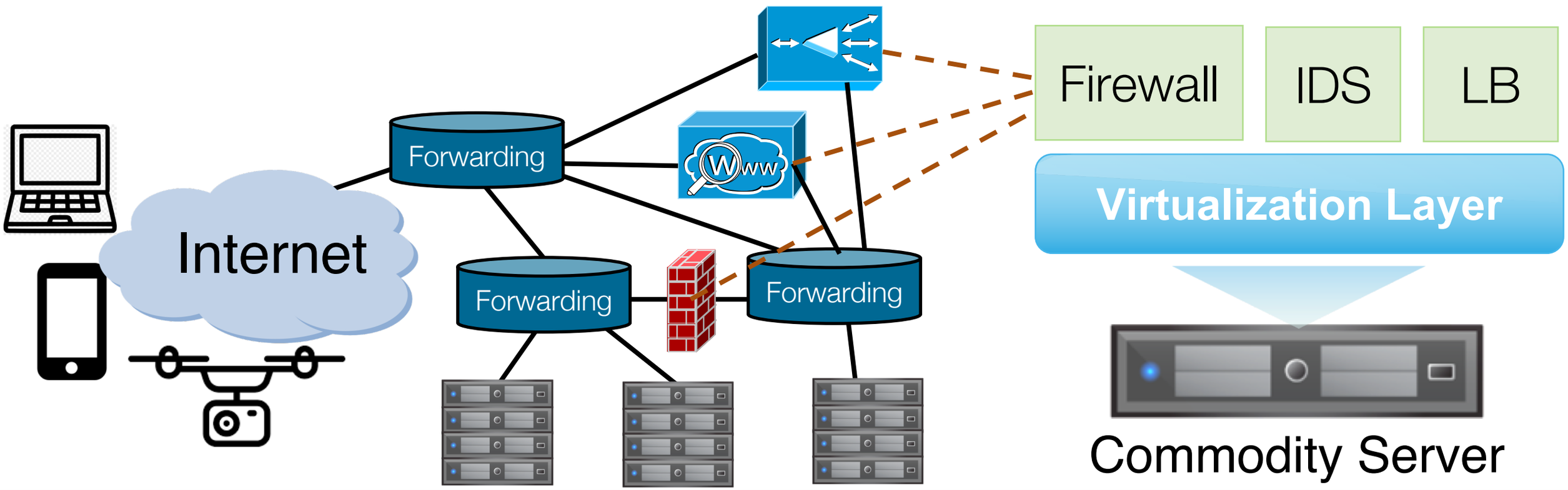
Network Functions

- SDN solves only half of the problem
- Networks does more than routing packets
  - **Network Functions (NFs) or Middleboxes:** perform specialized tasks to improve security and performance
  - Firewalls, DPI, Proxies, Caches, WAN Optimizers, Protocol Accelerators etc.
  - One in three devices in an enterprise network is a middlebox [SIGCOMM'12]
  - **Special-purpose hardware**: expensive, fixed functionality and deployment, hard to manage and upgrade



Network Function Virtualization

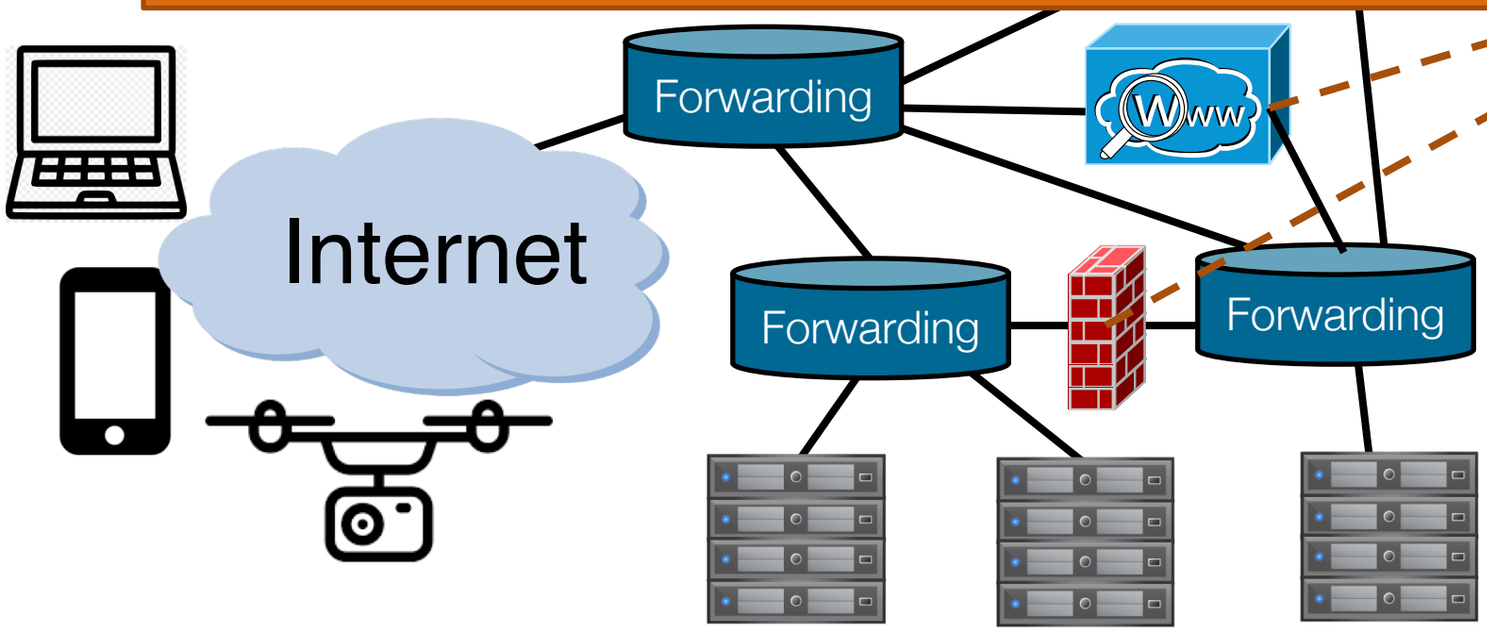
- Network Function Virtualization (NFV)
  - Run Network Functions in virtual machines on commodity servers
  - More flexible than hardware
  - Easy to deploy and manage
  - Improve scalability and better utilize network resources
  - Reduce both operational and capital expenditures



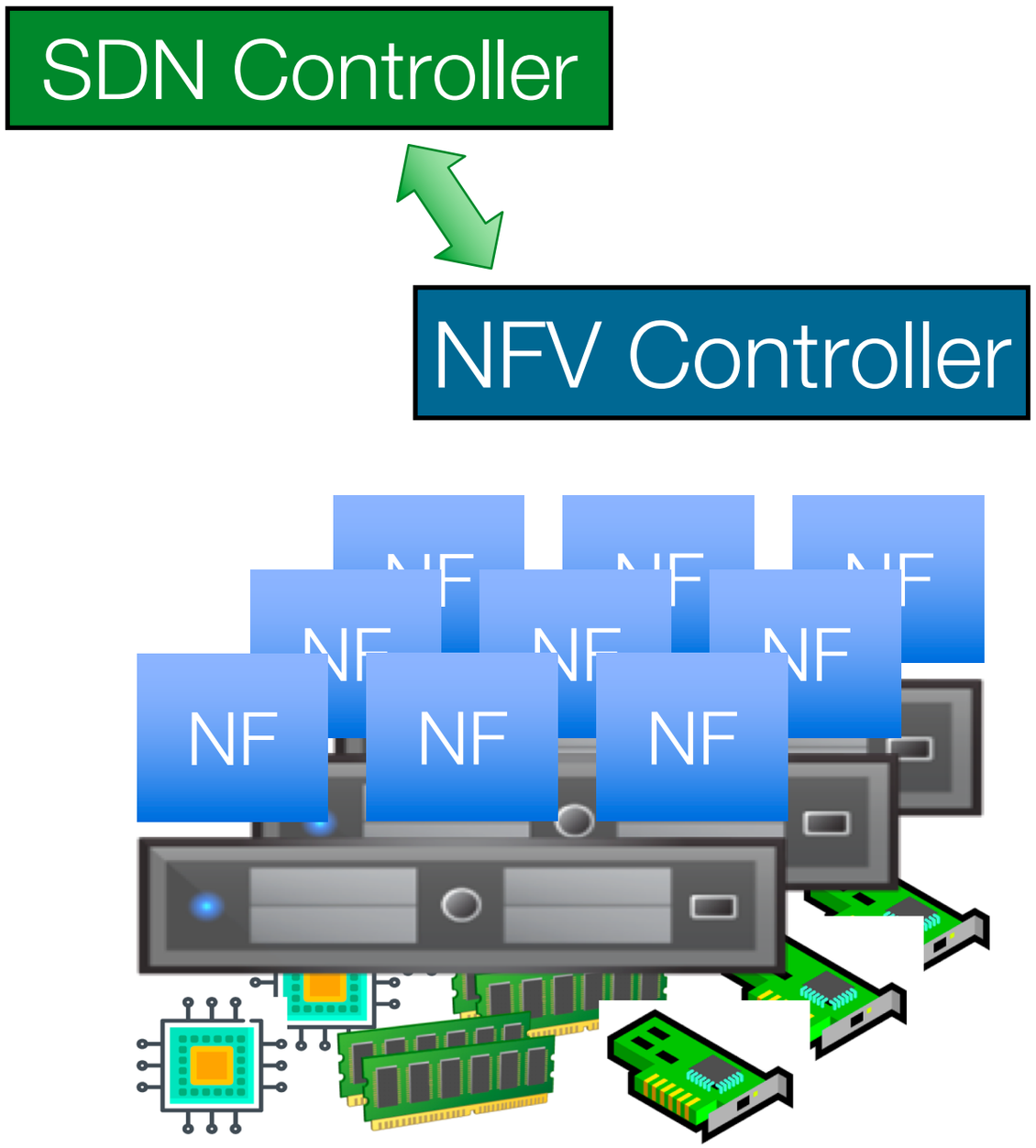
Network Function Virtualization

- Network Function Virtualization (NFV)
  - Run Network Functions in virtual machines on commodity servers
  - More flexible than hardware
  - Easy to deploy and manage
  - Improve scalability and better utilize network resources
  - Reduce both operational and capital expenditures

Challenges across system, application and service layers to realize this vision



Challenges

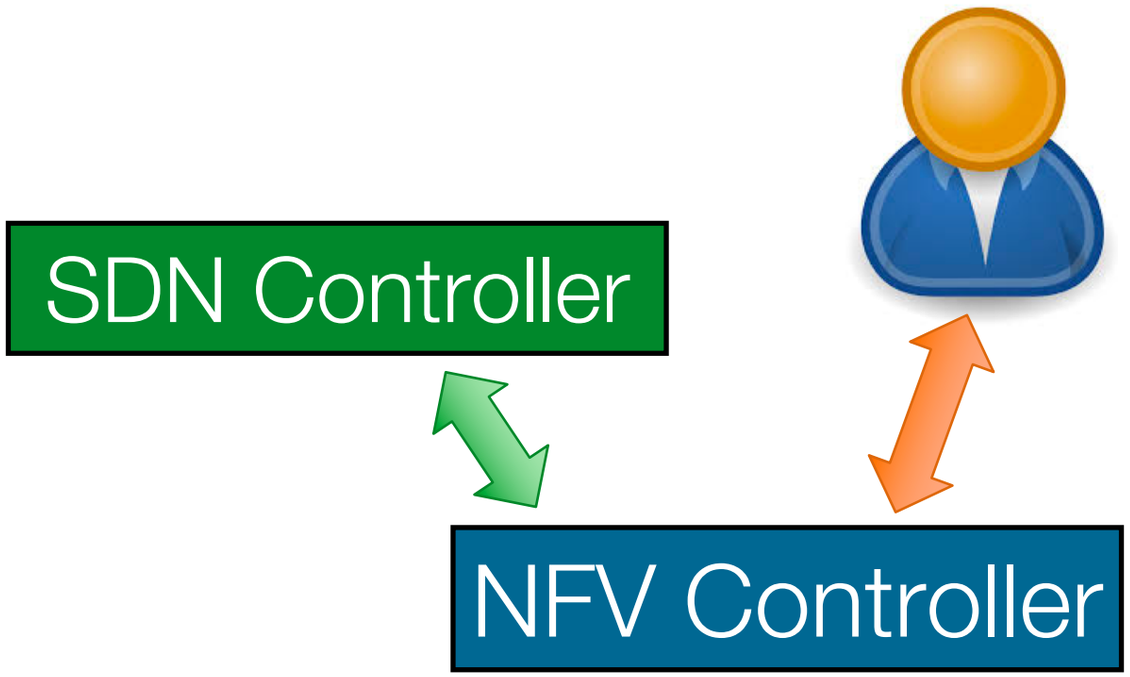


How to interact with the SDN controller?

How to manage NFs across multiple hosts?

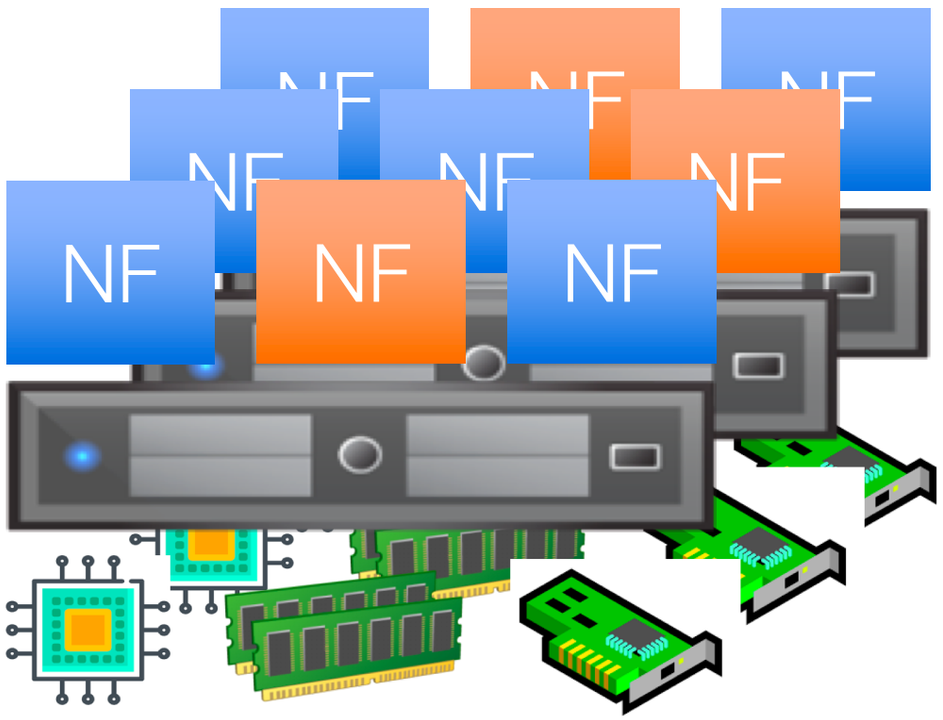
How to create a high performance data plane?

Challenges



Is it possible to customize the NFV framework?

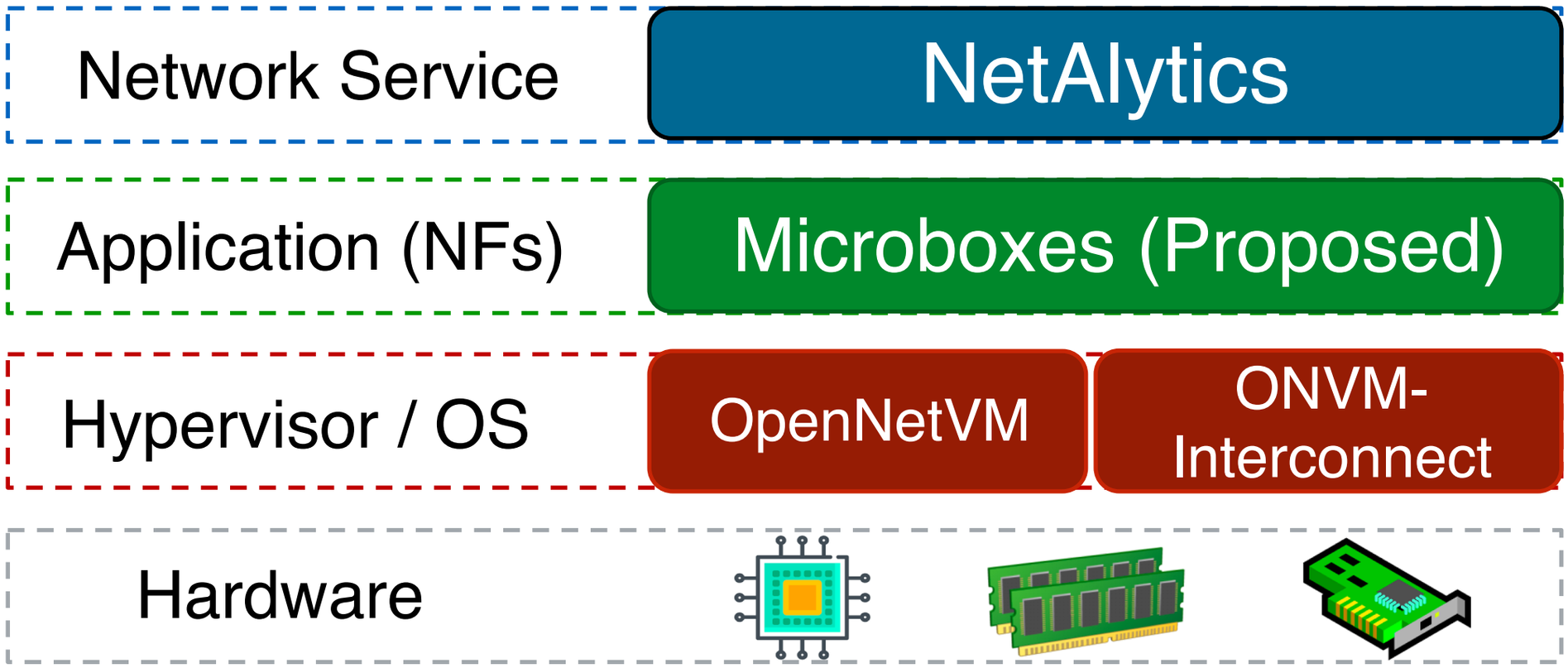
How to interact with the SDN controller?



How to manage NFs across multiple hosts?

How to create a high performance data plane?

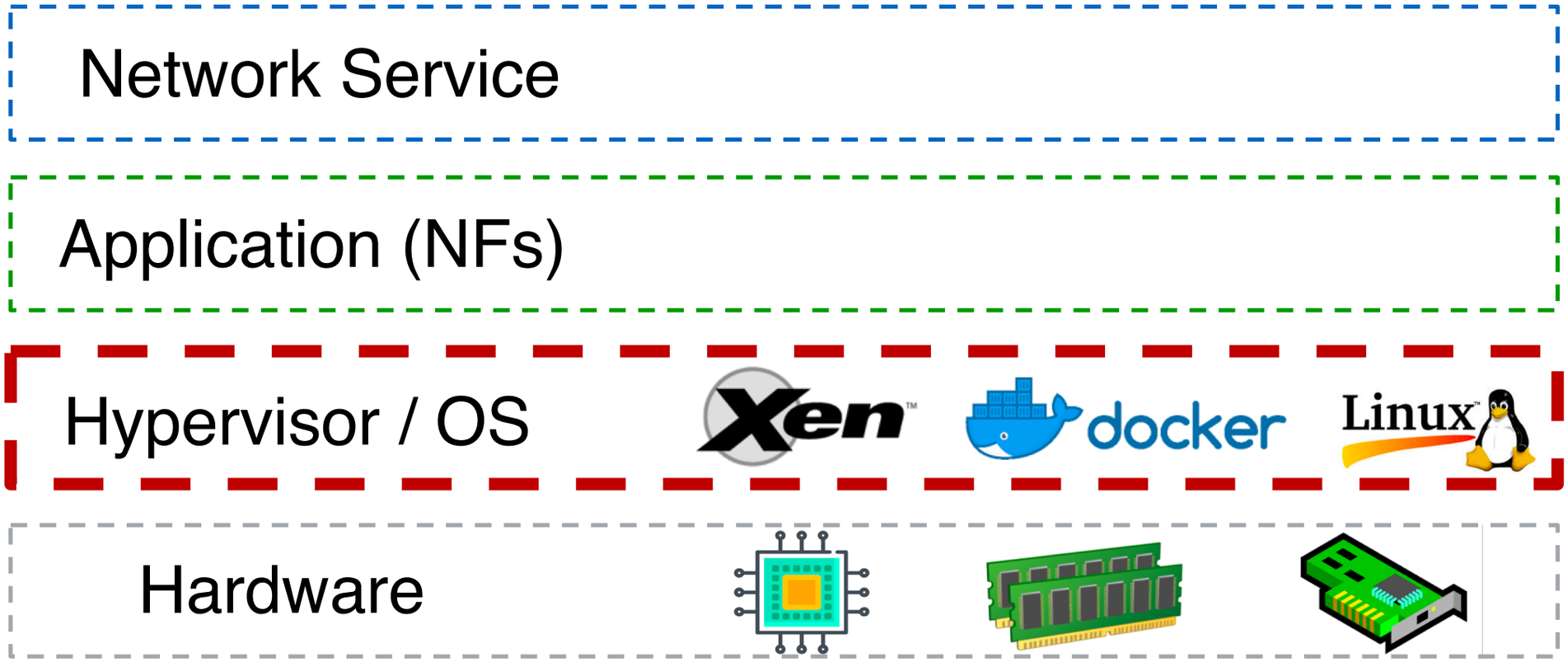
Contributions



- **OpenNetVM & ONVM-Interconnect**: A high-performance multi-host NFV platform
- **Microboxes**: A customizable service chaining abstraction
- **NetAlytics**: A non-intrusive monitoring platform

Outline

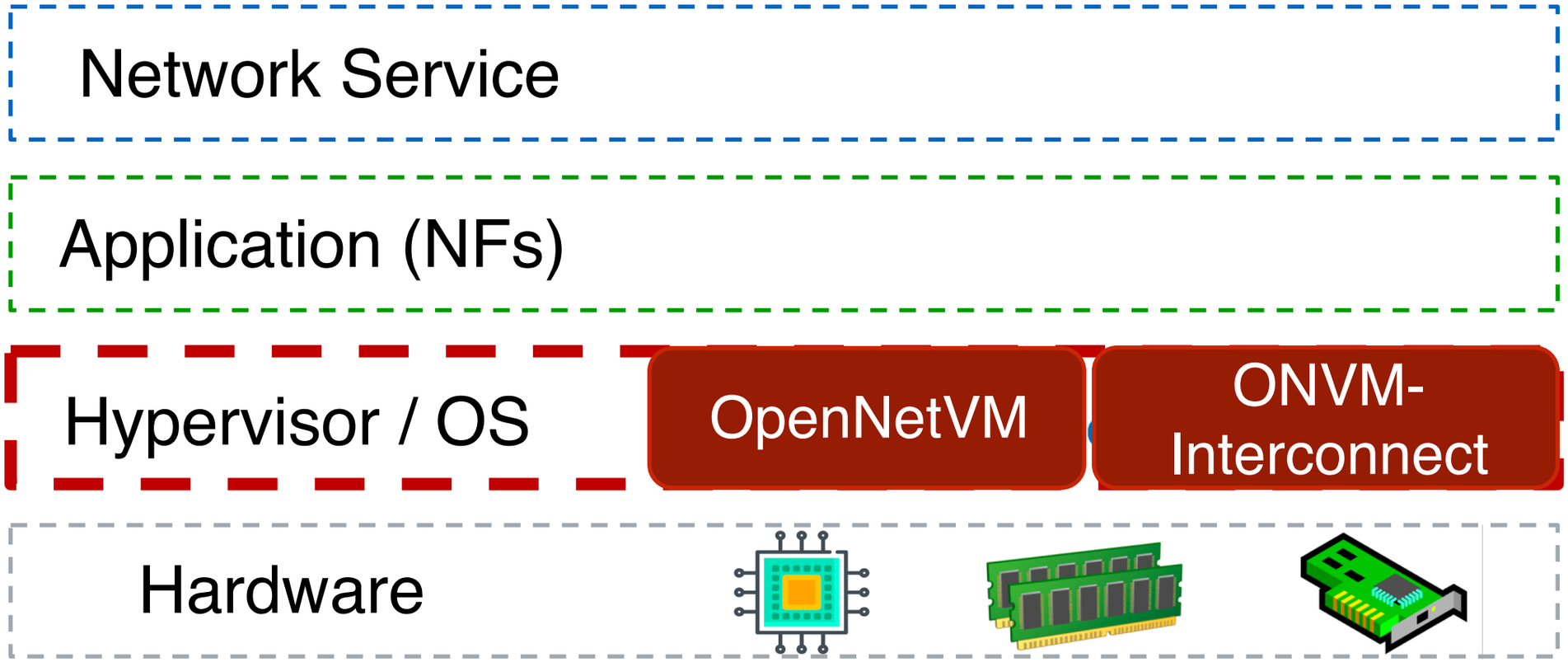
- Hypervisor / OS Layer:
  - virtualization enables resource sharing and simplifies automation
  - traditional kernel stack and virtualization layer are too expensive
- How can we design and optimize the system to achieve high performance packet processing?



Outline

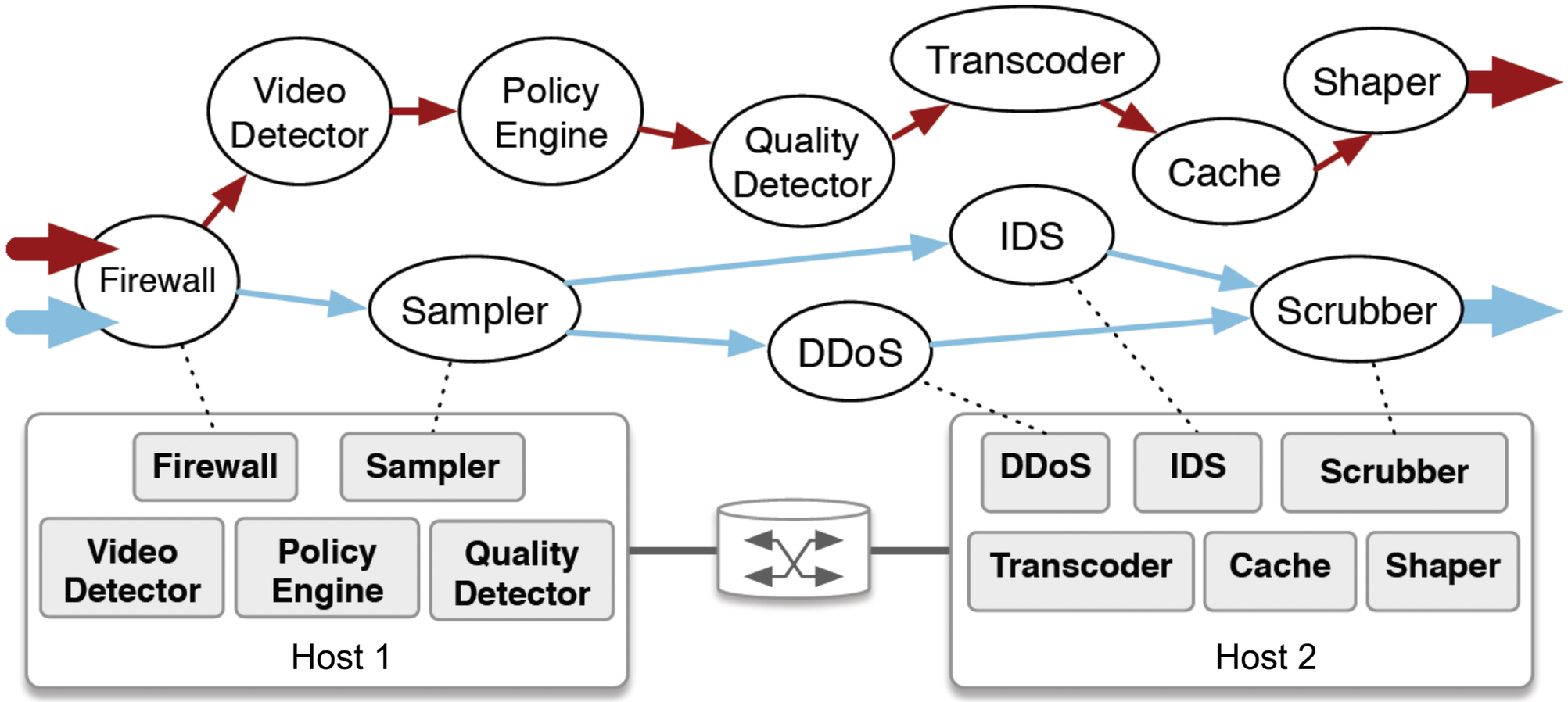
- **OpenNetVM** [HotMiddlebox'16] **& ONVM-Interconnect** [KBNets'17]: a high performance service chaining platform

Decouple NF management from packet processing



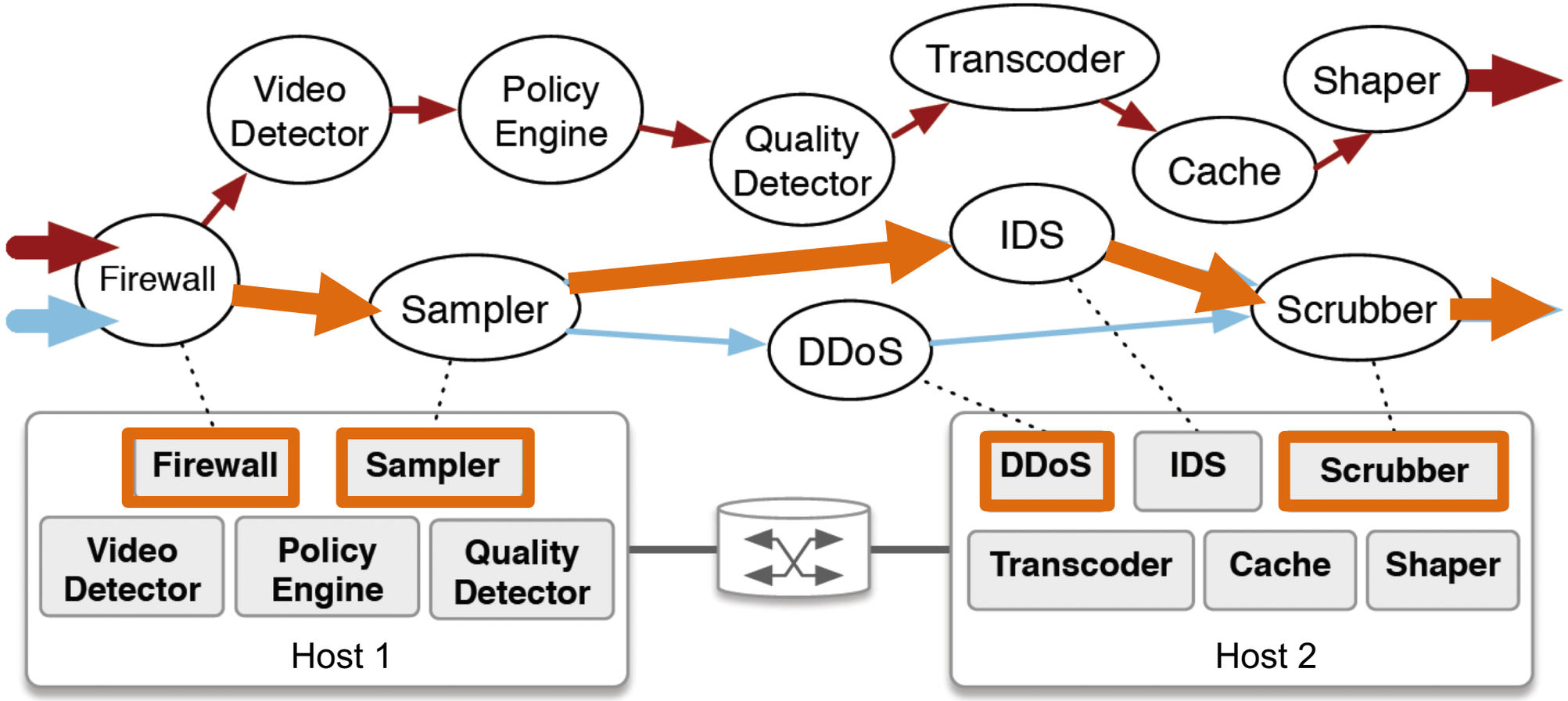
Complex Services

- A complex service needs a group of NFs stitched together and may require the use of multiple servers to hold the chain.
- video optimization (red), anomaly detection (blue)



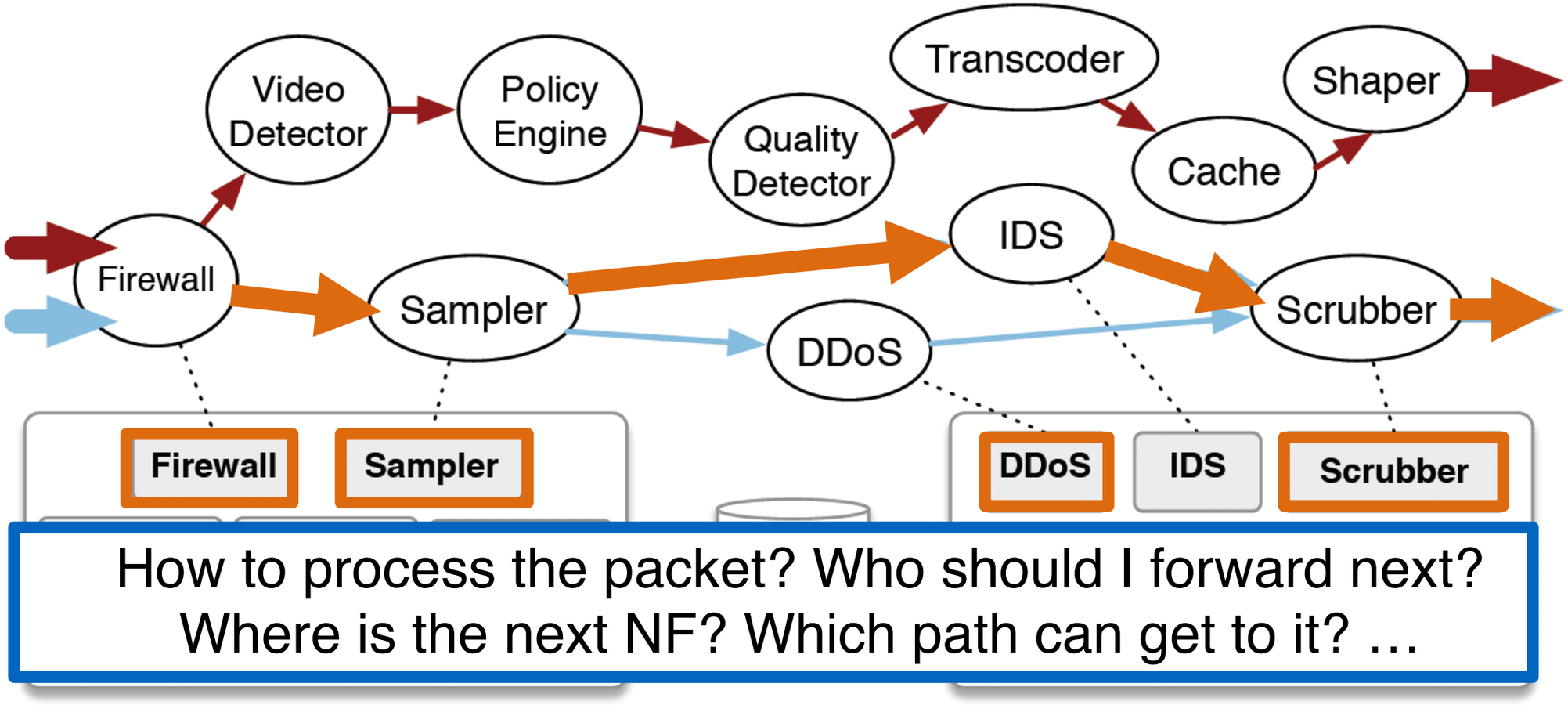
Service Chain

- Service Chain:
  - **NFs:** different functionality, capacity, and locate on multiple hosts
  - **Critical Path**: each NF processes packets and forwards to the next NF in order – run as fast as possible



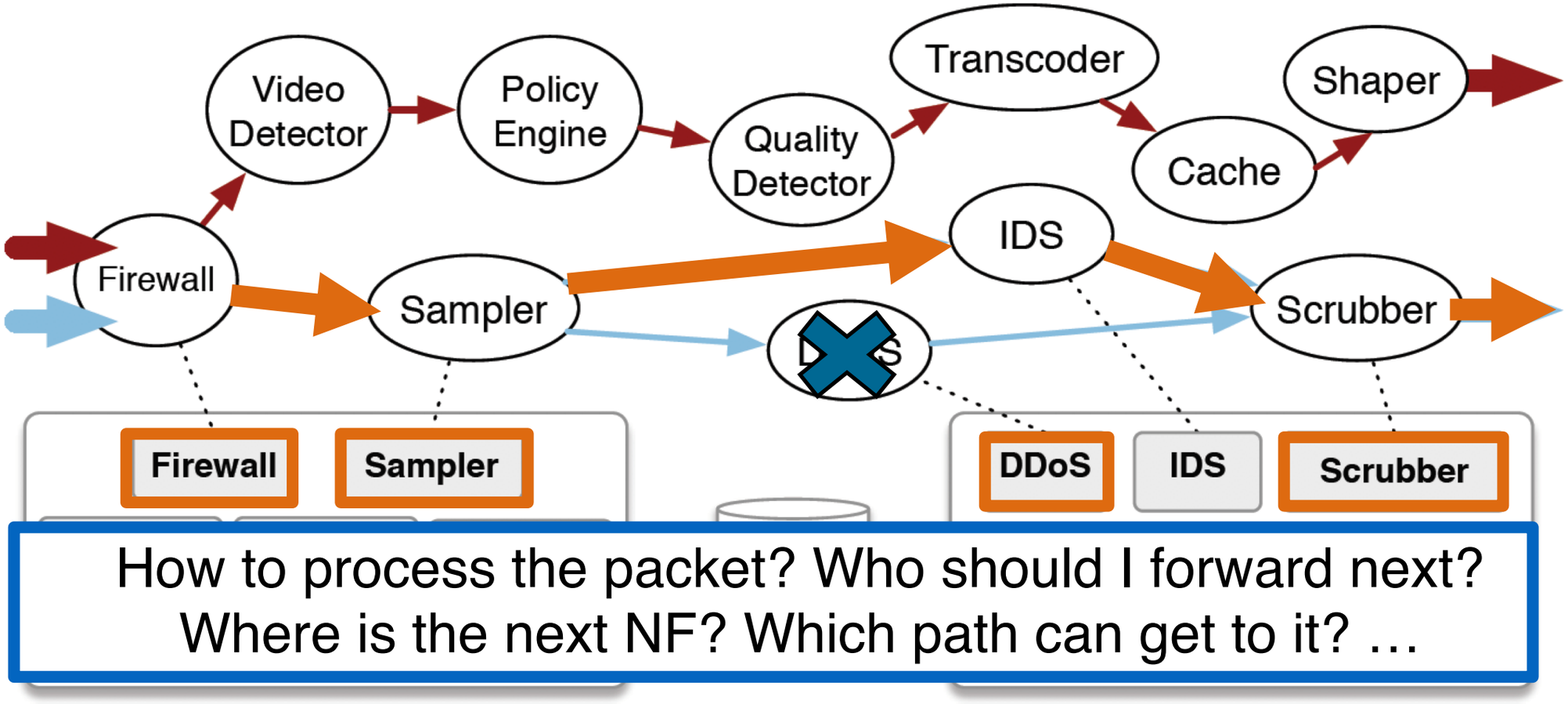
Service Chain

- Service Chain:
  - **NFs:** different functionality, capacity, and locate on multiple hosts
  - **Critical Path**: each NF processes packets and forwards to the next NF in order – run as fast as possible



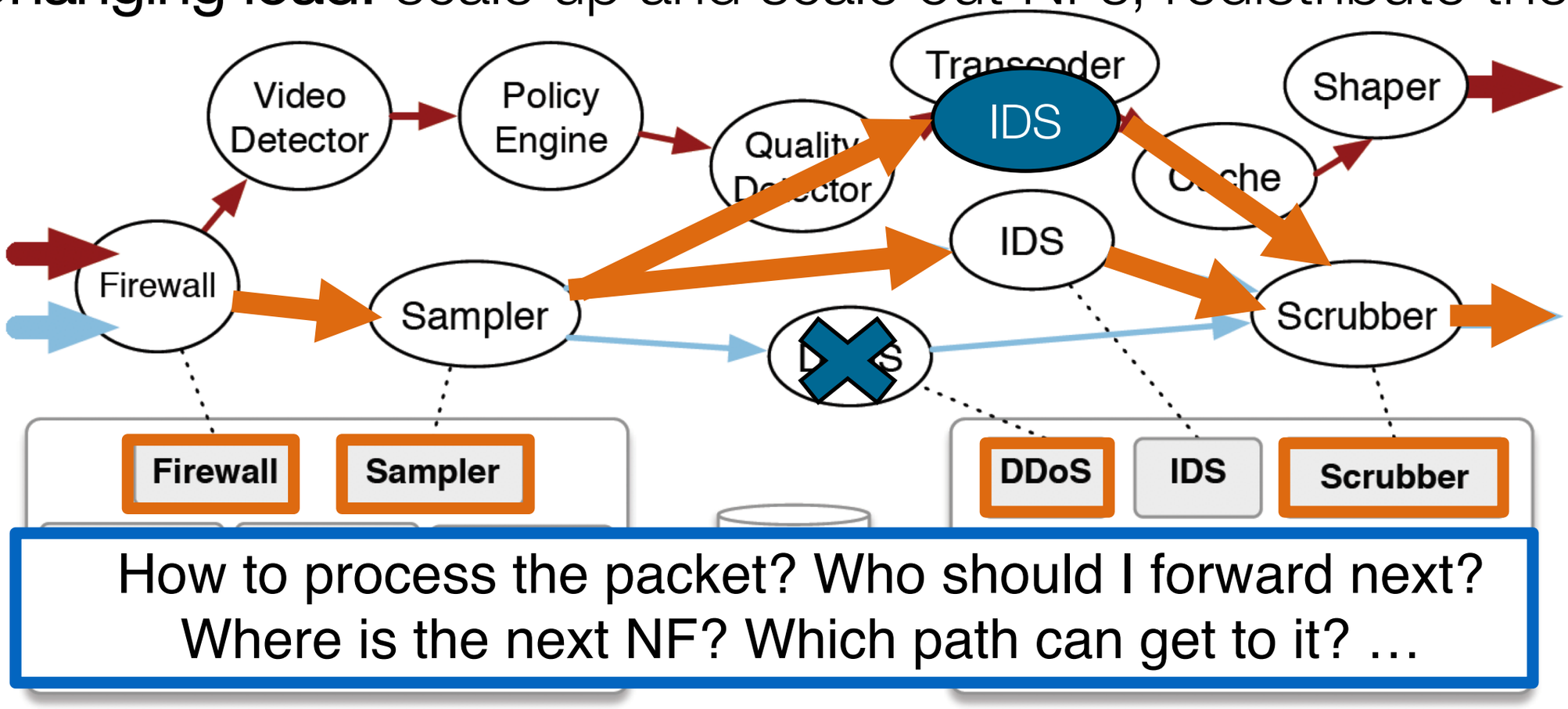
Service Chain

- Service Chain:
  - **NFs:** different functionality, capacity, and locate on multiple hosts
  - **Critical Path**: each NF processes packets and forwards to the next NF in order – run as fast as possible
  - **Not static**: add new functions, reroute existing flows



Service Chain

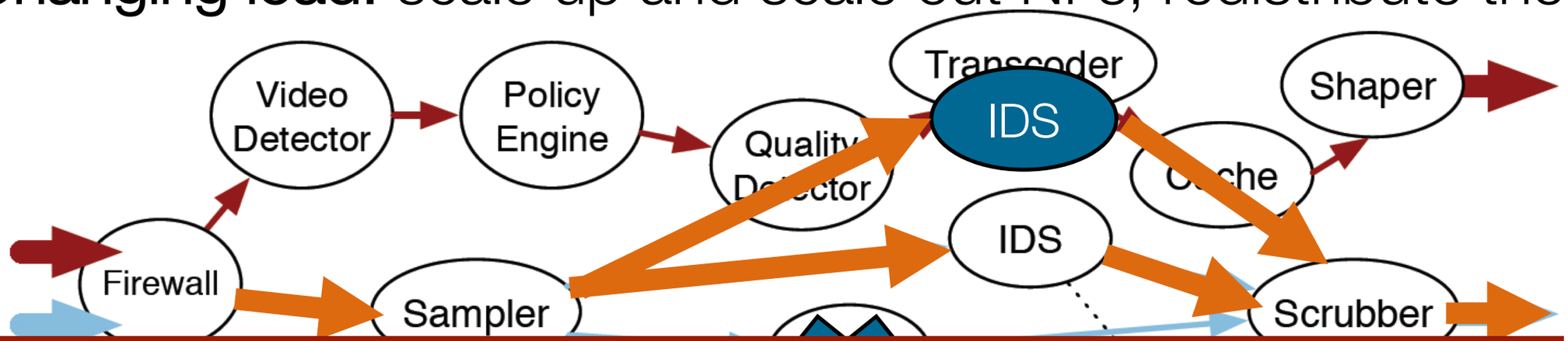
- Service Chain:
  - **NFs:** different functionality, capacity, and locate on multiple hosts
  - **Critical Path**: each NF processes packets and forwards to the next NF in order – run as fast as possible
  - **Not static**: add new functions, reroute existing flows
  - **Changing load:** scale up and scale out NFs, redistribute the load



How to process the packet? Who should I forward next? Where is the next NF? Which path can get to it? …

Service Chain

- Service Chain:
  - **NFs:** different functionality, capacity, and locate on multiple hosts
  - **Critical Path**: each NF processes packets and forwards to the next NF in order – run as fast as possible
  - **Not static**: add new functions, reroute existing flows
  - **Changing load:** scale up and scale out NFs, redistribute the load



Require a high performance, flexible and scalable NFV framework

How to process the packet? Who should I forward next? Where is the next NF? Which path can get to it? …

NFV Framework

Network Operator

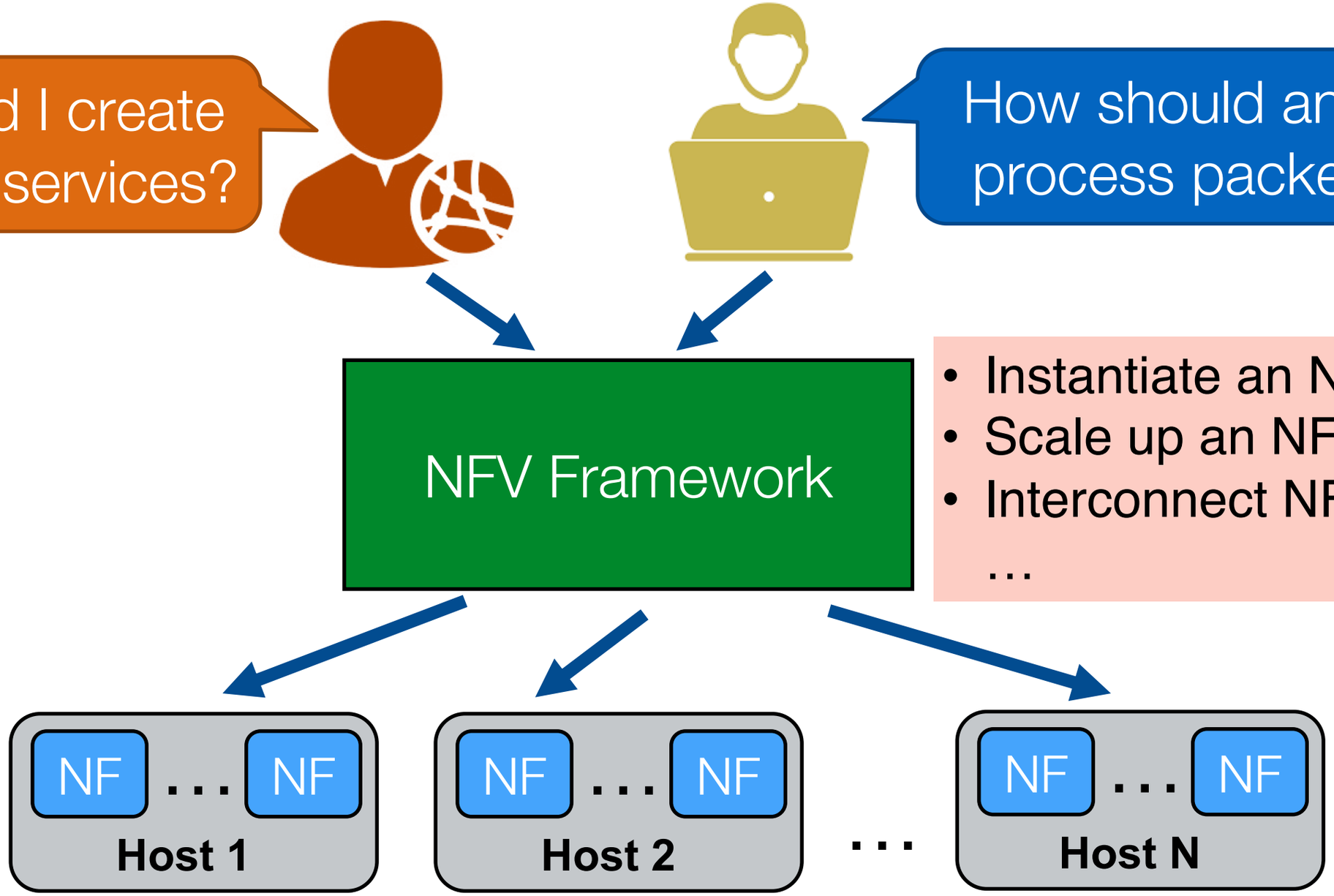
How should I create and deploy services?

NF Developer

How should an NF process packets?

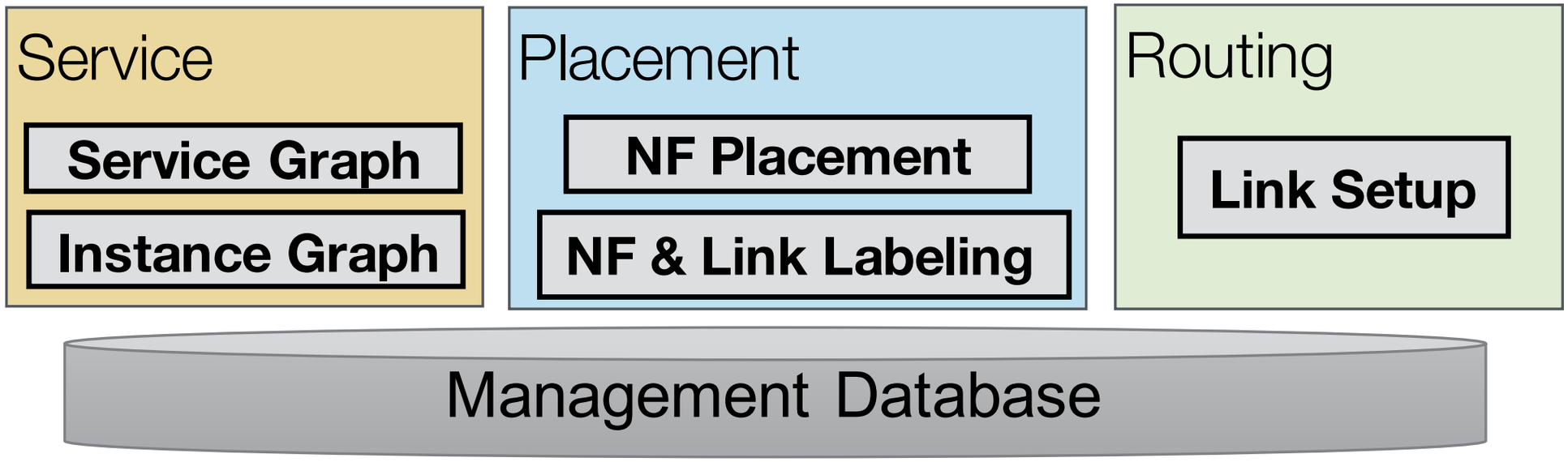
- Instantiate an NF
- Scale up an NF
- Interconnect NFs

…

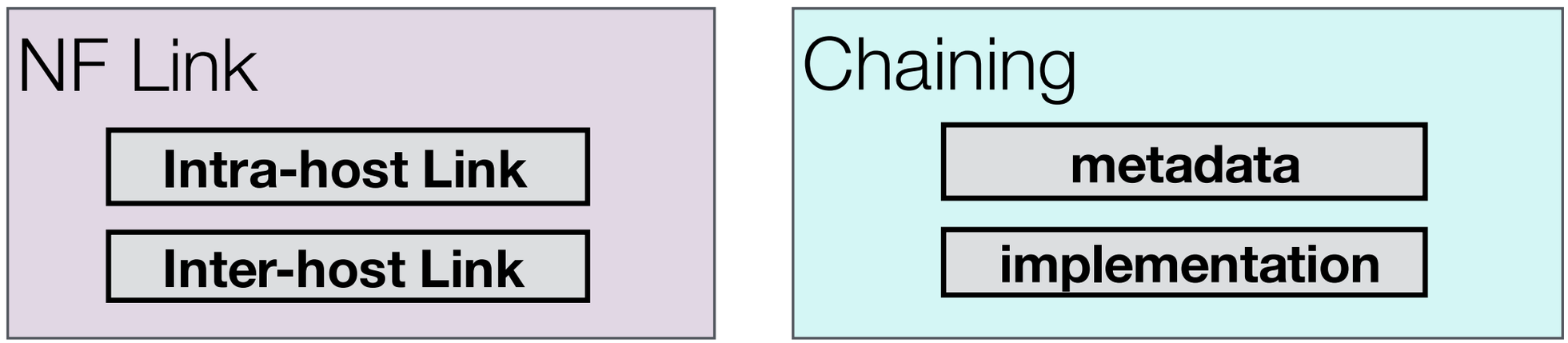


ONVM-Interconnect

Control Plane



Data Plane

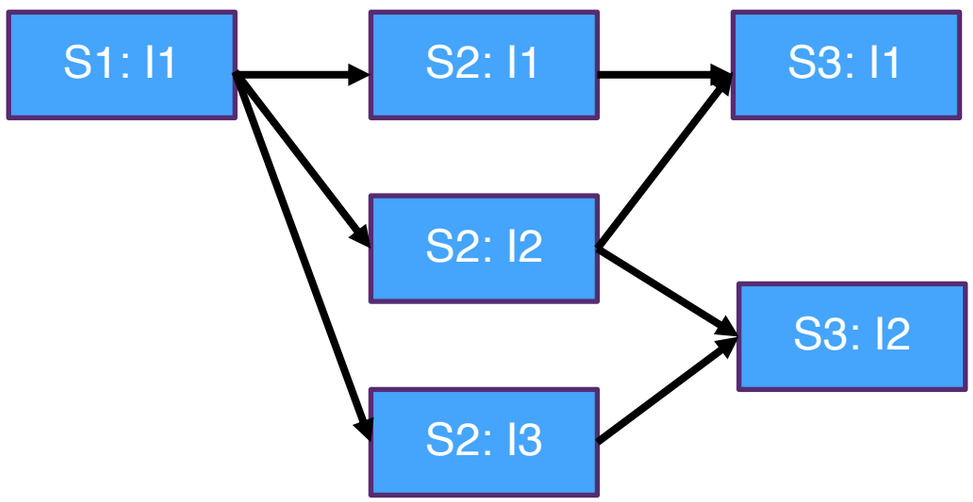


Control Plane

- Service Graph
  - Service NF:
    - Represent a specific packet processing function
    - NF template
    - Identified by Service ID
  - Virtual Link:
    - Stitch a sequence of services together to create a service chain

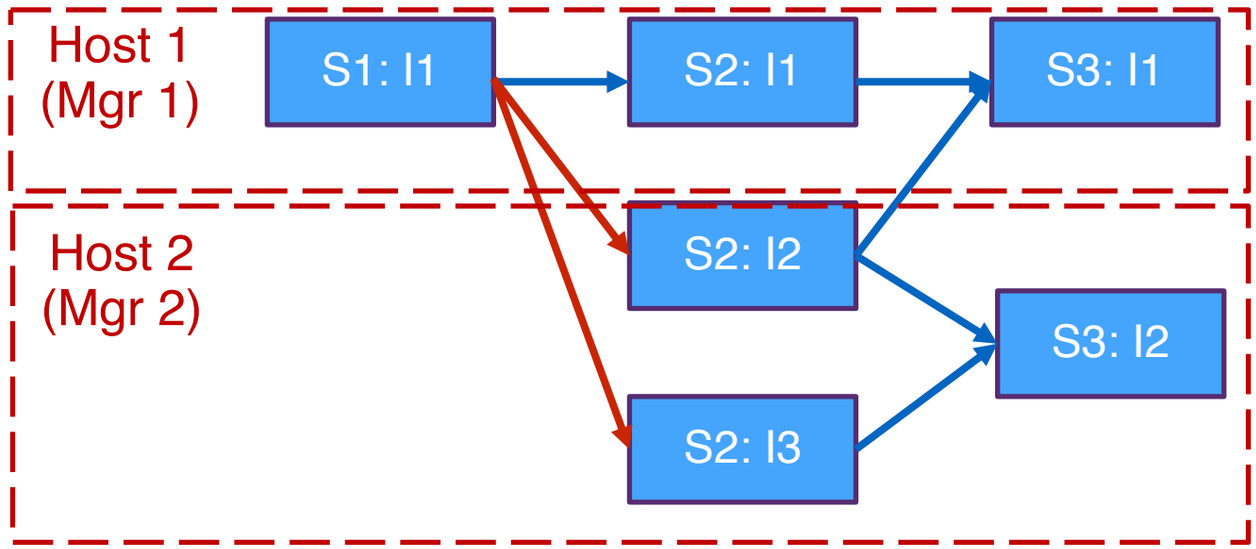
Control Plane

- Instance Graph
  - Instance NF:
    - Replicas of one service NF
    - NFs of the same service have a unique instance ID
  - Load balance traffic across instances
  - Scaling: Increase/decrease #instances for one service



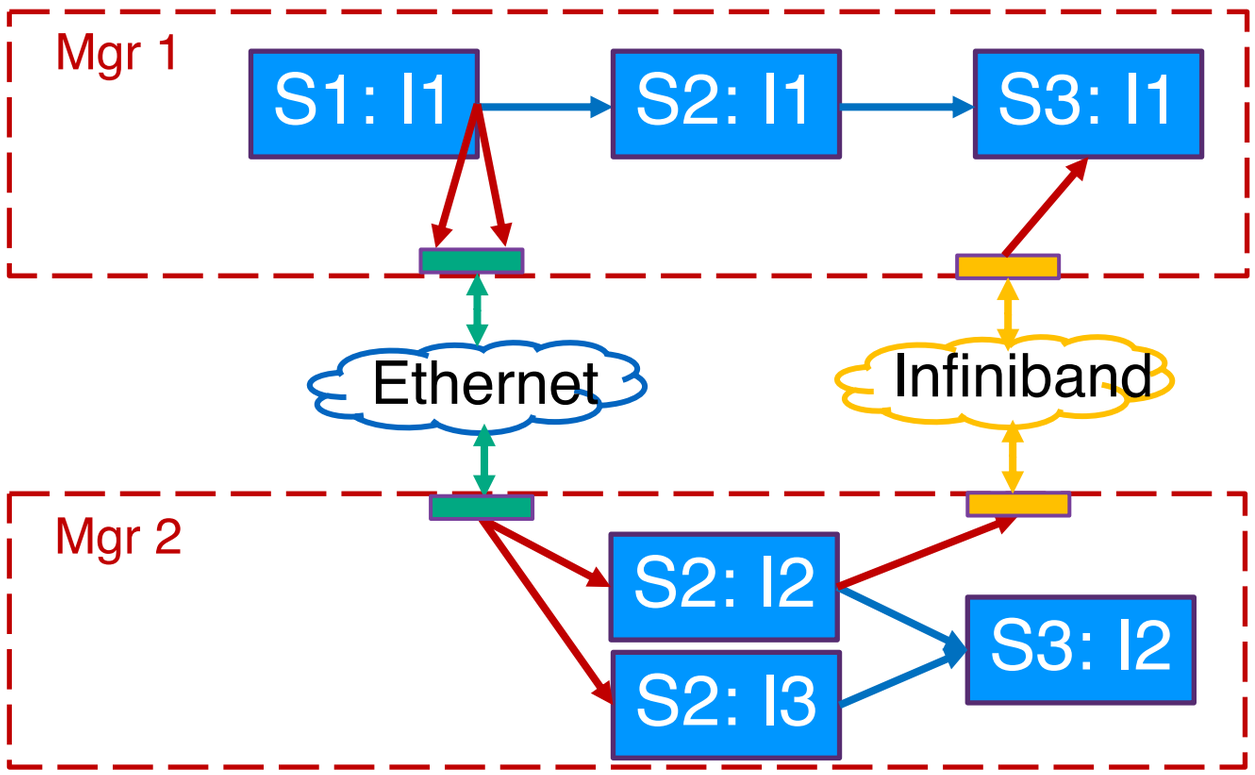
Control Plane

- Placement
  - NFs are placed on multiple hosts and managed by local NF managers.
  - NF labeling: (mgr_id, service_id, instance_id)
  - Link labeling: intra-host link, inter-host link

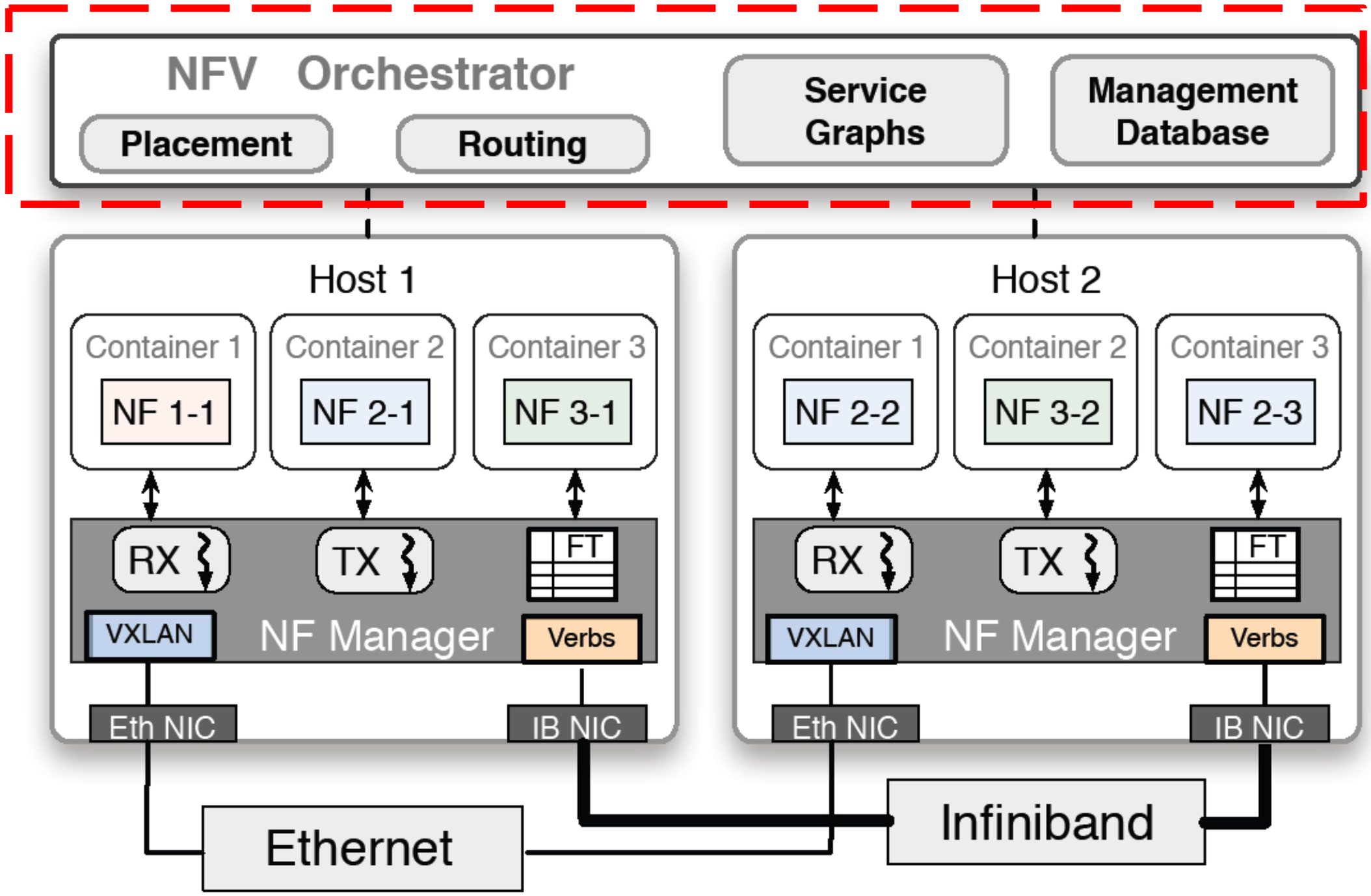


Control Plane

- Routing
  - Setup routes for inter-host link
  - Link addr: (src_mgr_id, dst_mgr_id, src_mgr_port, dst_mgr_port, link_type)



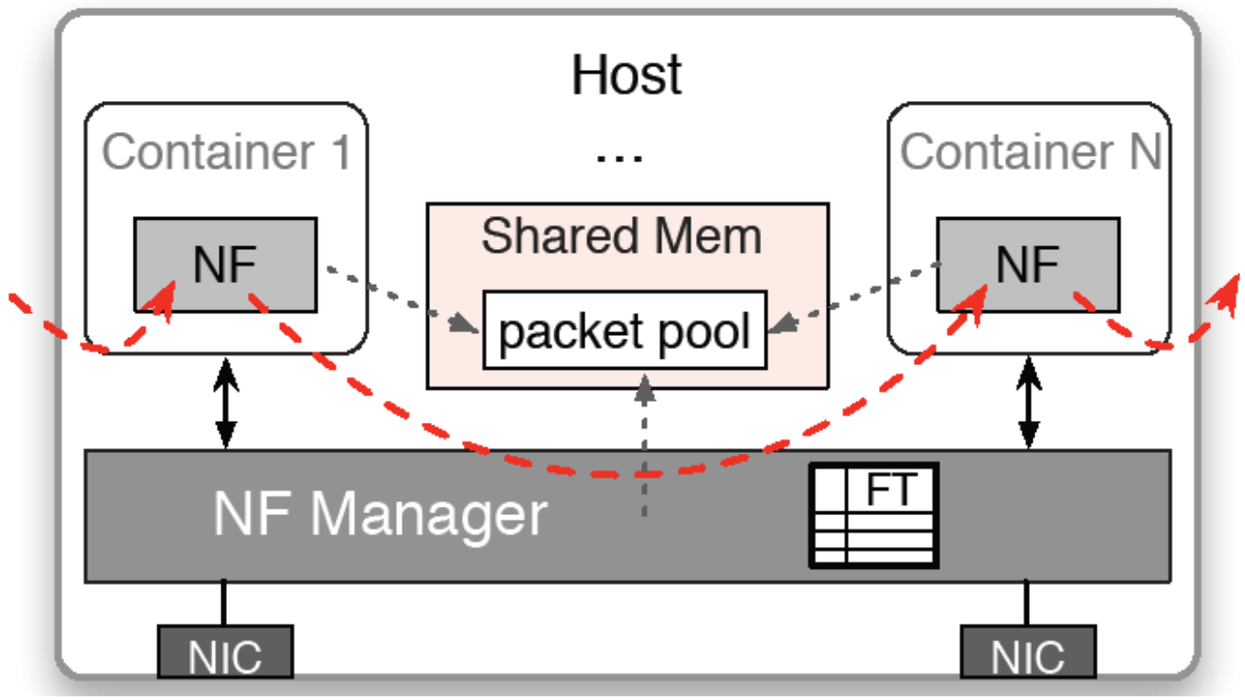
Control Plane



Data Plane

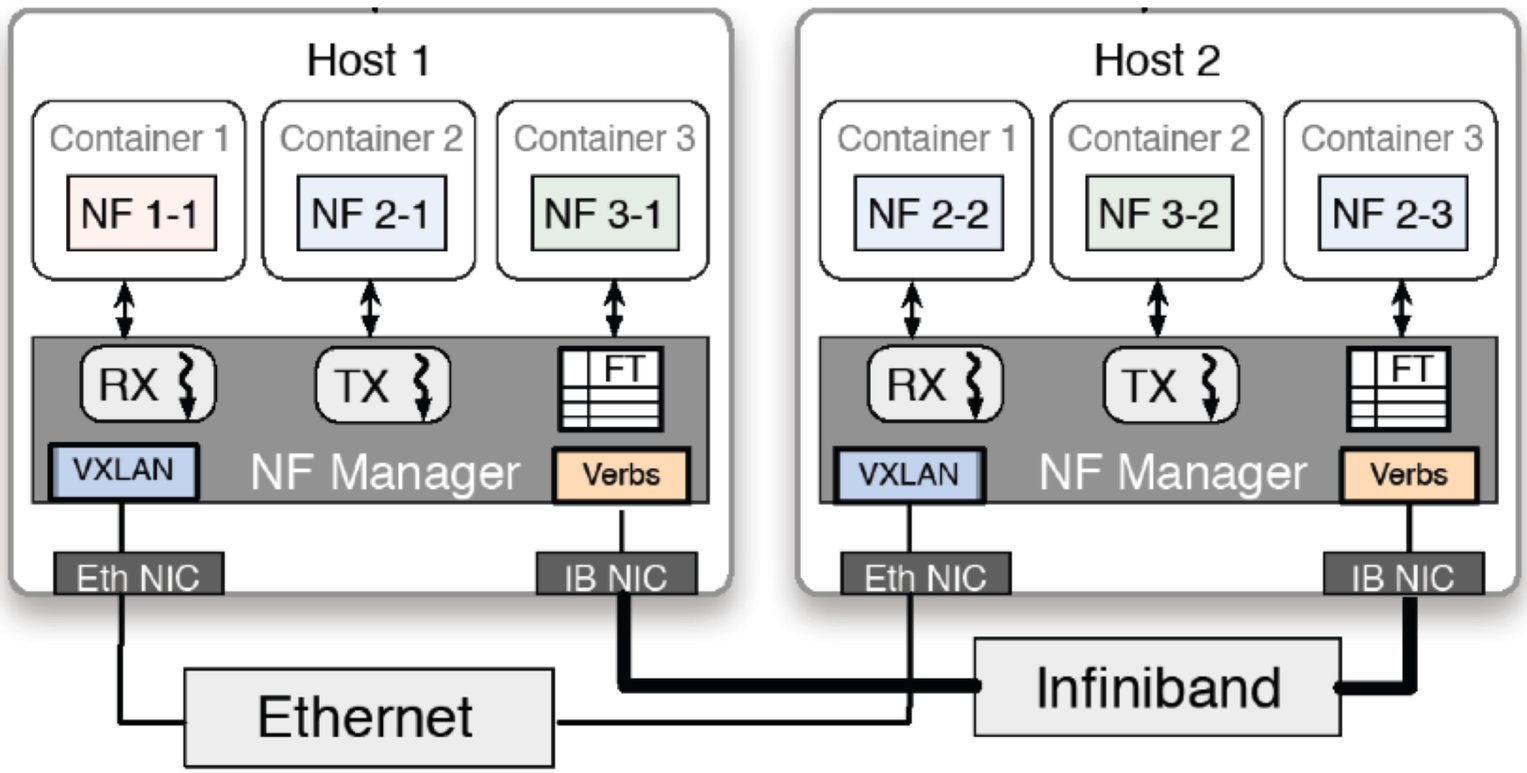
- OpenNetVM architecture on a single server
  - **DPDK:** provides underlying I/O engine
  - **NFs**: run insider docker containers, use NFlib APIs
  - **NF manager**: tracks which NFs are active, organizes chains
  - **Shared Memory**: efficient communication between NFs

=> eliminates virtualization & kernel overheads



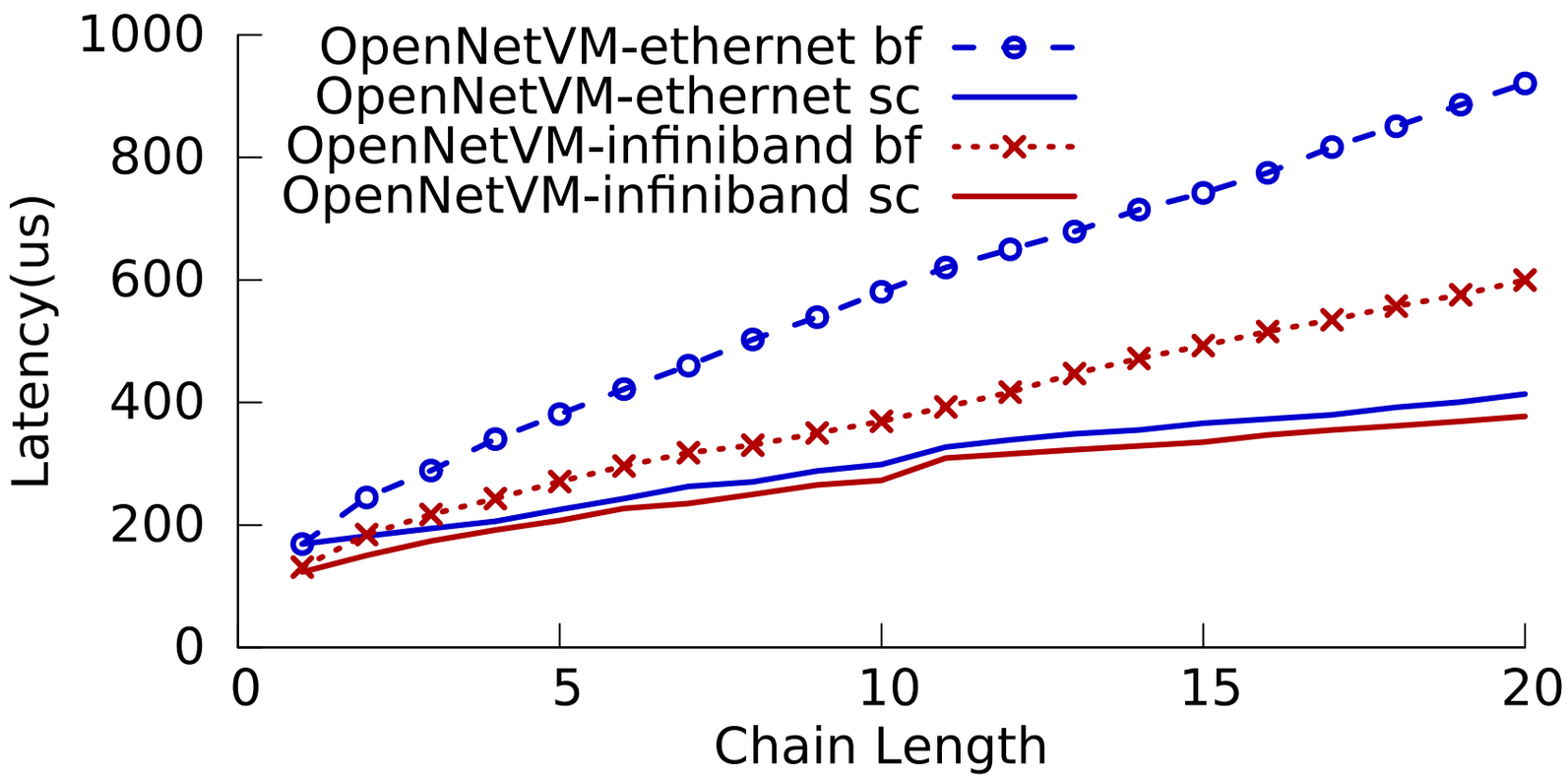
Data Plane

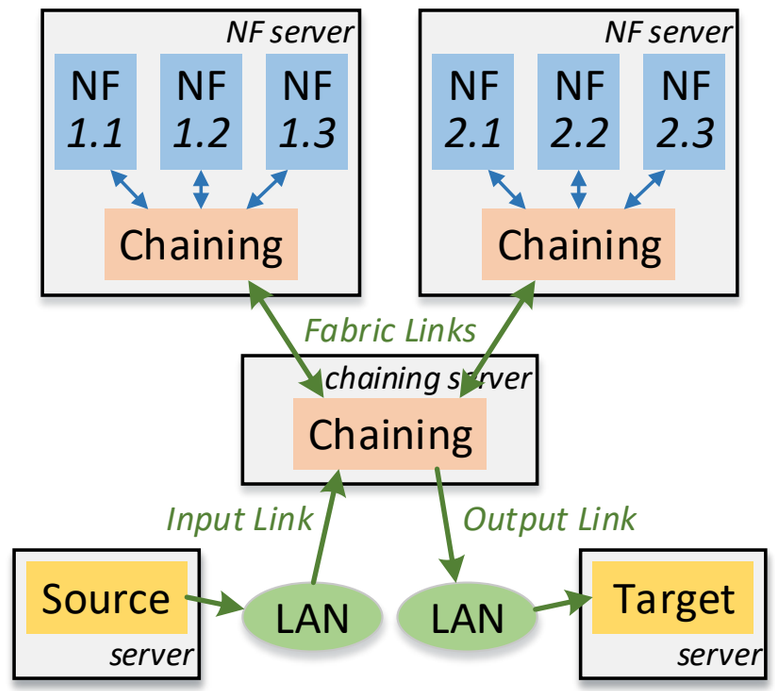
- NF Link connection between two NFs
  - Intra-host link: Shared Memory
  - Inter-host link:
    - Ethernet: vxlan encapsulation
    - InfiniBand (higher bandwidth and lower latency): verbs API, tune queue pair attributes to improve performance



Evaluation

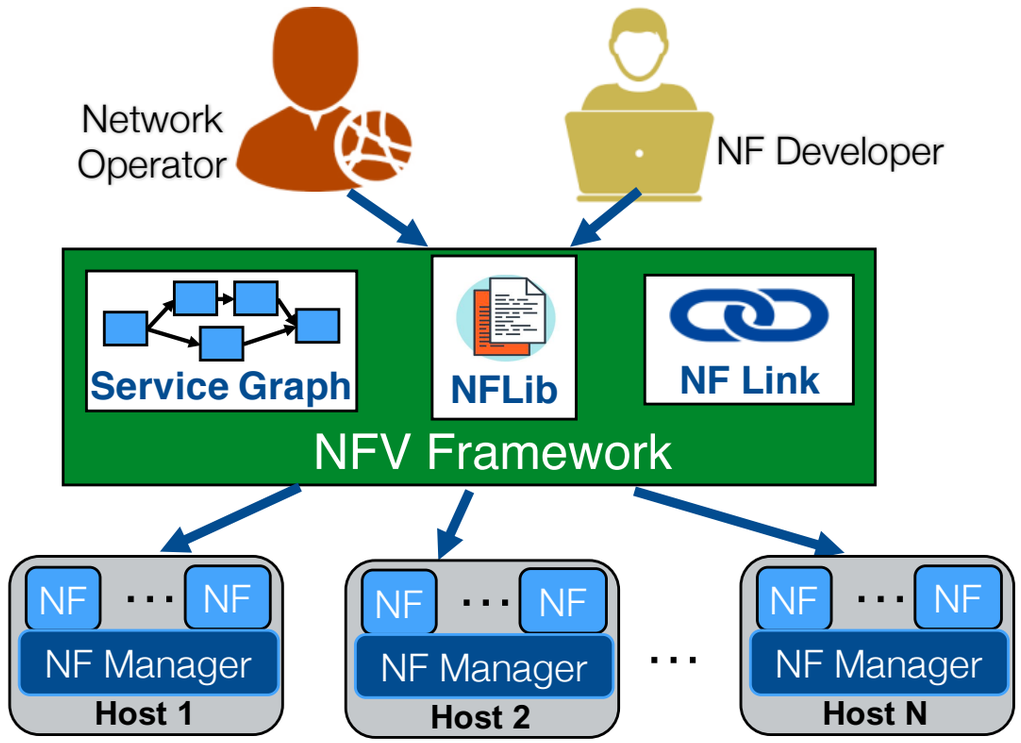
- Two placement strategies:
  - Chaining server distributes load between two servers running NFs
  - ShortCut (best case): traverse all NFs on server1 before being rerouted to server2
  - Back&Forth (worst case): bounce between NFs on the two hosts for every hop
- Both fabric type and placement strategy affect the performance
  - Infiniband can provide significant latency reductions, reducing the impact of NF placement decisions.





ONVM-Interconnect

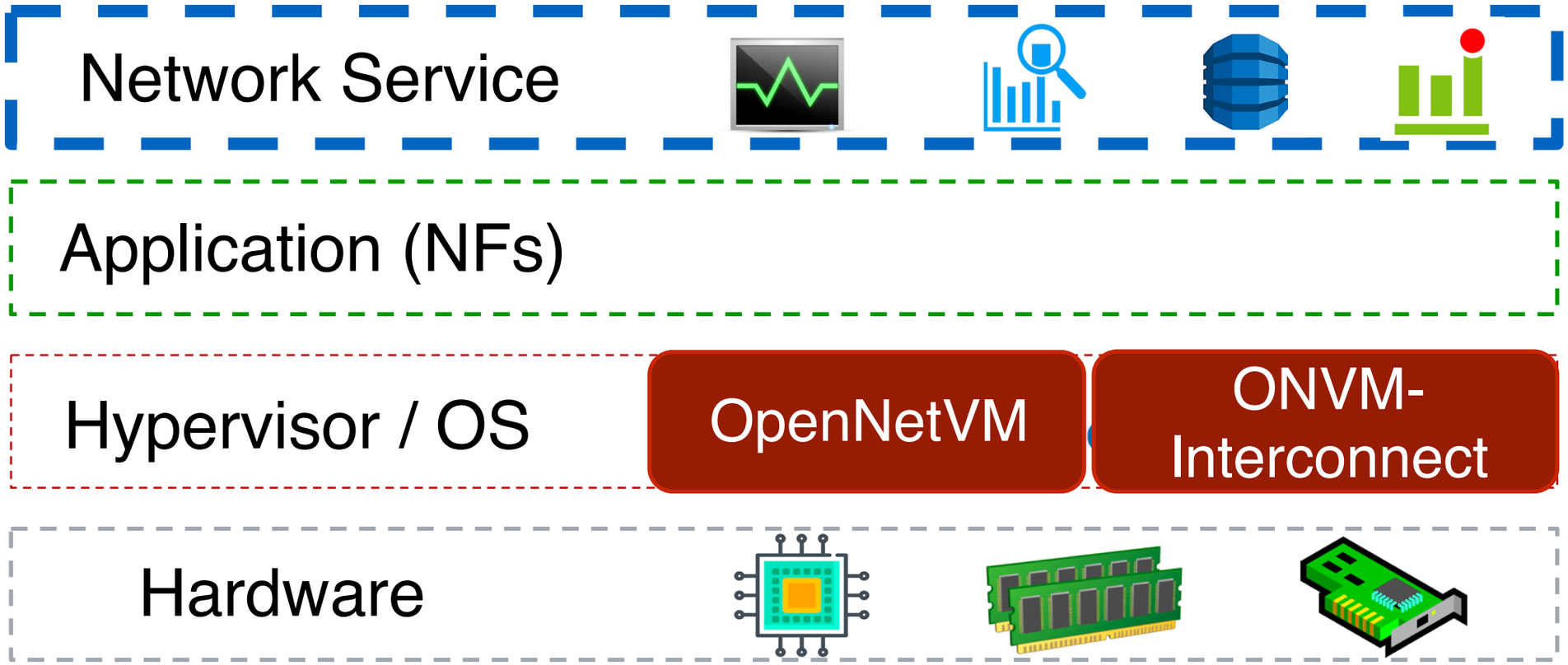
ONVM-Interconnect decouples NF management from packet processing while providing a high performance data plane



- **Scalable**: support both scale-up and scale-out NF execution
- **Performance**: NFs are interconnected with high performance links both in a single host and over multiple hosts
- **Flexible**: packets can be flexibly steered through service chains connected by different fabrics

Outline

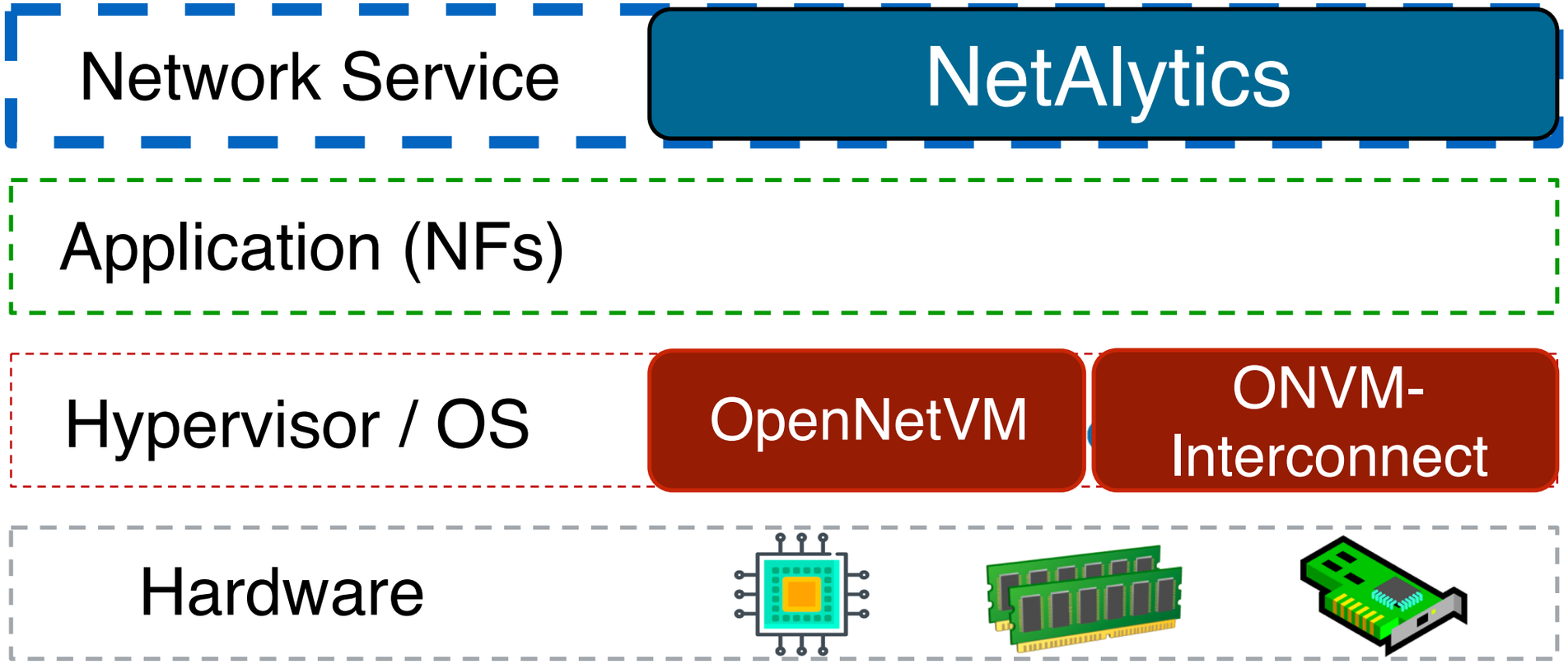
- Network Service Layer:
  - global network services include monitoring, analytics, traffic engineering etc.
  - the volume and variability of data keeps growing, and traditional tools can't process and analyze them in a timely manner
- How can we design network services that are **scalable, efficient, and transparent to customers**?



Outline

- **Netalytics** [Middleware'16]: Cloud-Scale Application Performance Monitoring with SDN and NFV

Turn the network from a communication pipe into a monitoring infrastructure



Common Questions

- Why is my multi-tier web application running slowly?
- Which types of requests are fast or slow?
- What is my most popular content?
- Which tier is my bottleneck?
- Where is network congestion affecting performance?

very efficiently

How to understand the **performance** and **behavior** of these large scale systems?

Existing Solutions

- #1 Software-logging: Need instrumentation, take time and space to process a large amount of data, may affect application performance
  - Example: 10Gbp/s for a hour requires 4.5TB of storage
- #2 Hardware-monitoring: Expensive, fixed deployment, limited monitoring fields and events analysis
  - Example: Cisco ASR 9001 Router $33,650.99

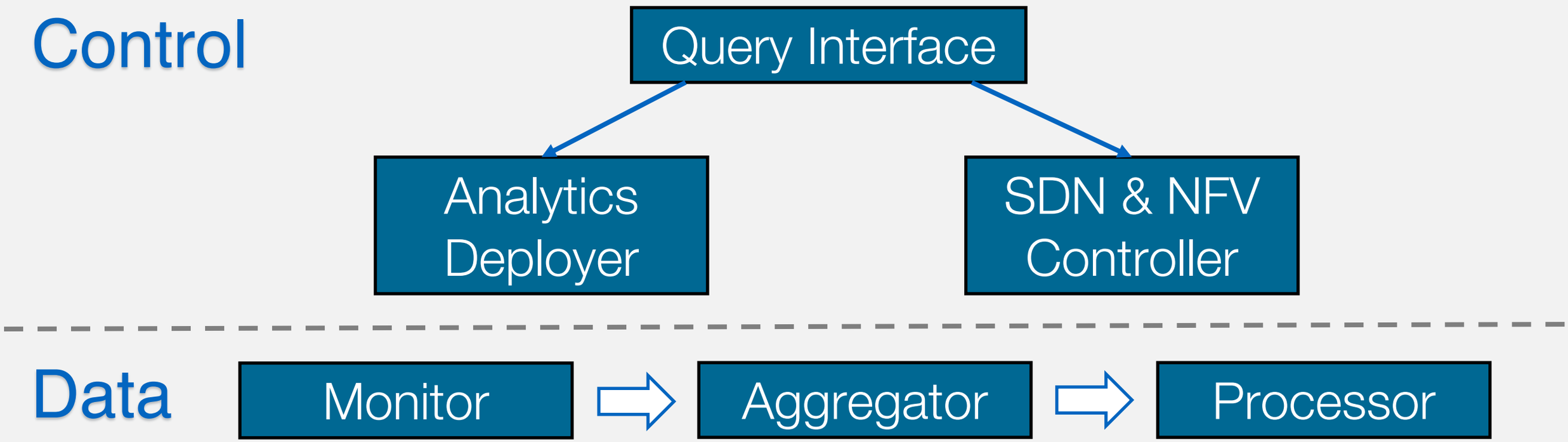
Emerging Trends

- Software-based Networks
  - Software Defined Networks (SDN): flexible control of packet flows
  - Network Function Virtualization (NFV): efficient packet processing software
- Big Data
  - Hadoop/Map Reduce: scalable batch processing
  - Real-time, streaming analytics

#1: Flexible network architectures that can gather network state -> Software Monitoring
#2: Tools to analyze large amounts of data with low latency –> Online Analysis

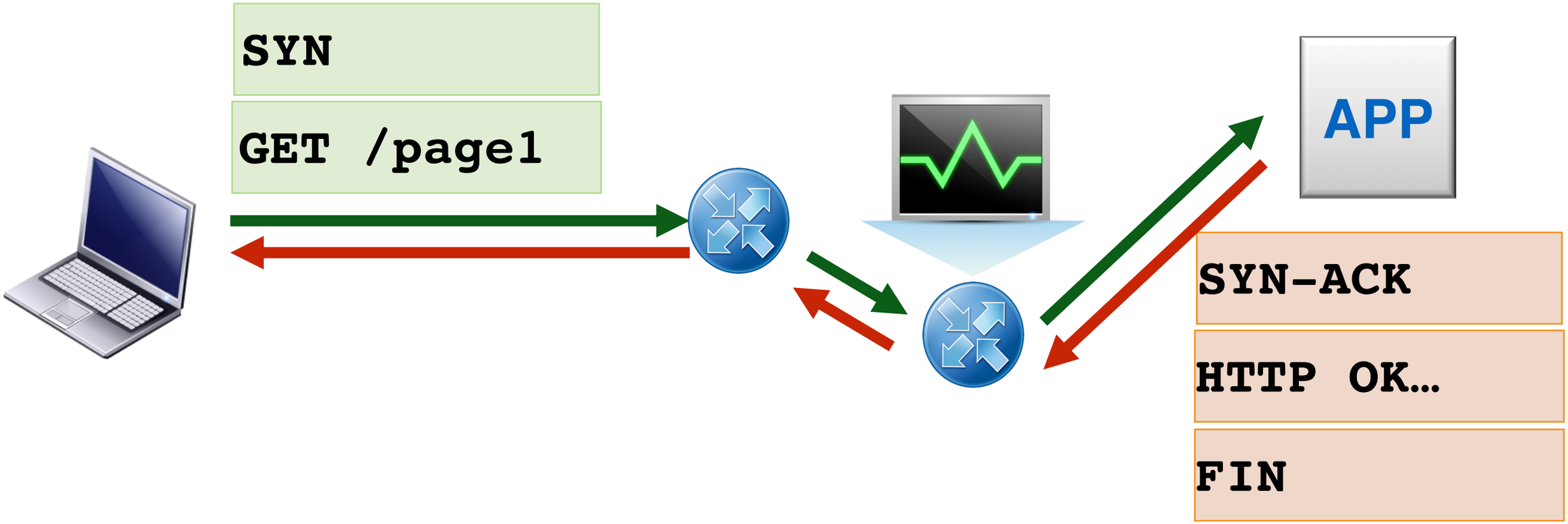
NetAlytics Overview

- A flexible control framework that uses SDN to split and redirect traffic, and automatic deploys data processors
- A scalable data pipeline that monitors packet flows unobtrusively and analyzes data in real-time



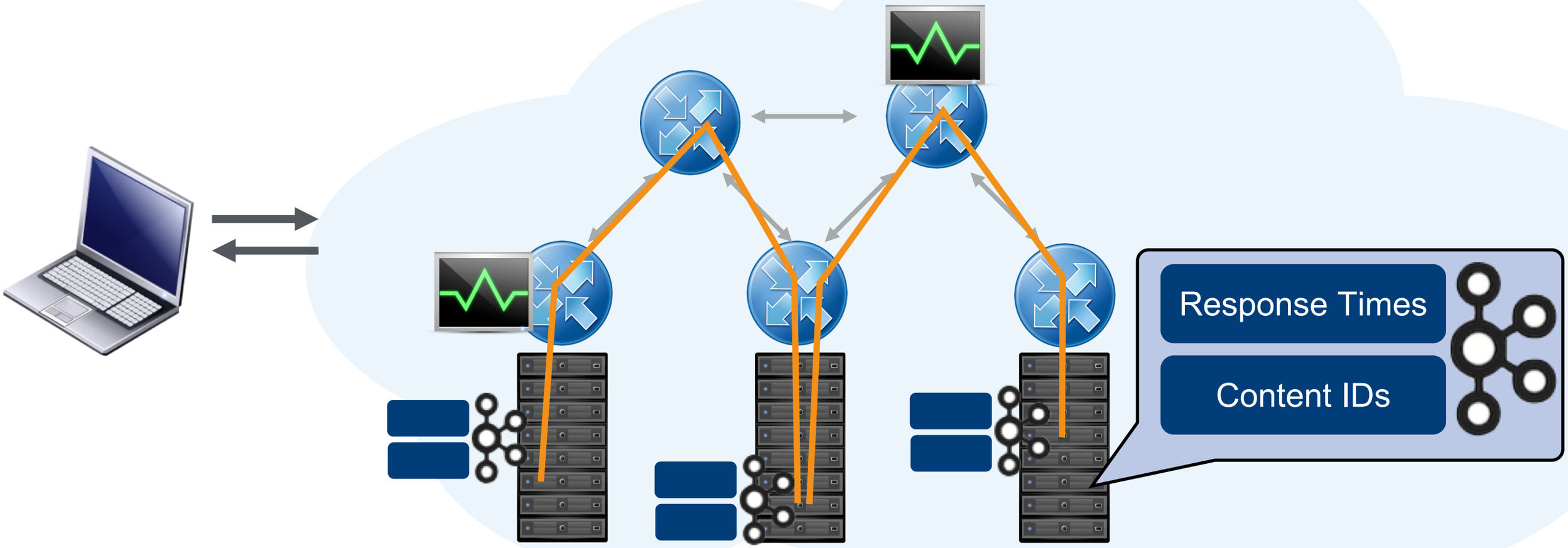
NetAlytics: Building Blocks

- **Monitors / Parsers**
  - NFV-based applications - incur minimal overhead
  - Observe packet flows sent by SDN controller
  - Parse protocols and extract important packet data - transparent to applications



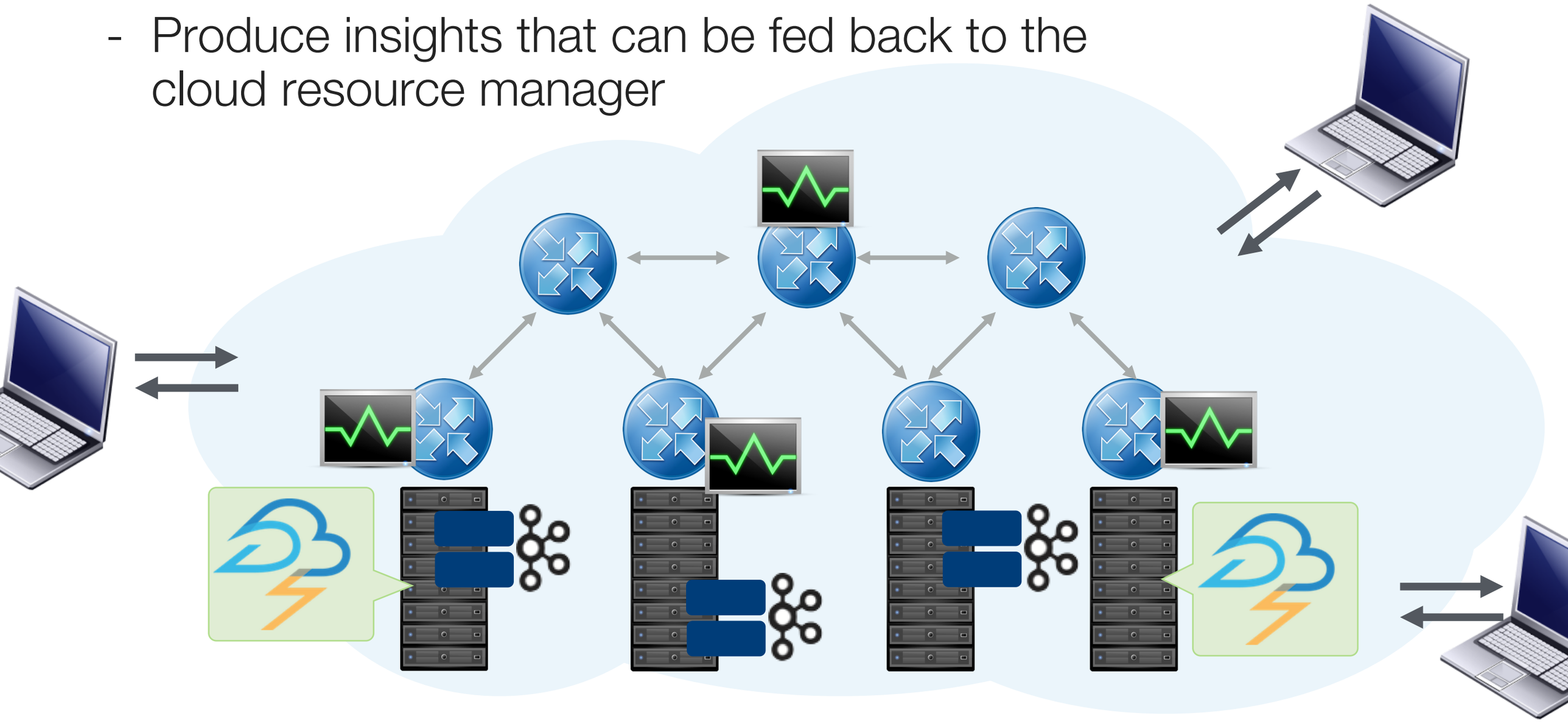
NetAlytics: Building Blocks

- **Aggregators**
  - Queuing services that store measurements from Monitors
  - One queue per topic
  - Hold data waiting to be processed



NetAlytics: Building Blocks

- **Processors**
  - Read and process data from aggregators in real time
  - Map-reduce style scalability
  - Produce insights that can be fed back to the cloud resource manager



Implementation

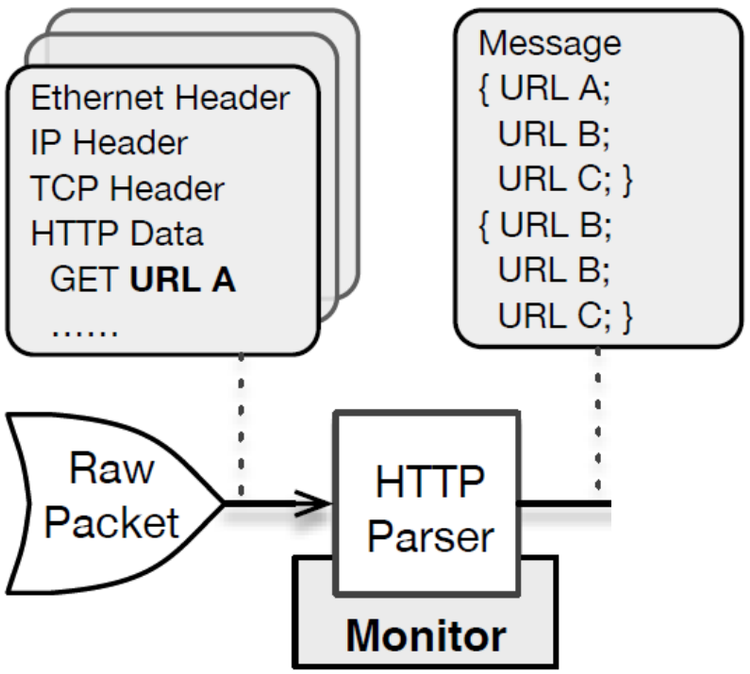
- **OpenNetVM** monitors
  - High performance NFV platform
  - Implement different parsers: tcp-connection-time, tcp_pkt_size, http_get, memcached_get

- **Apache Kafka** aggregators
  - High throughput, distributed messaging system

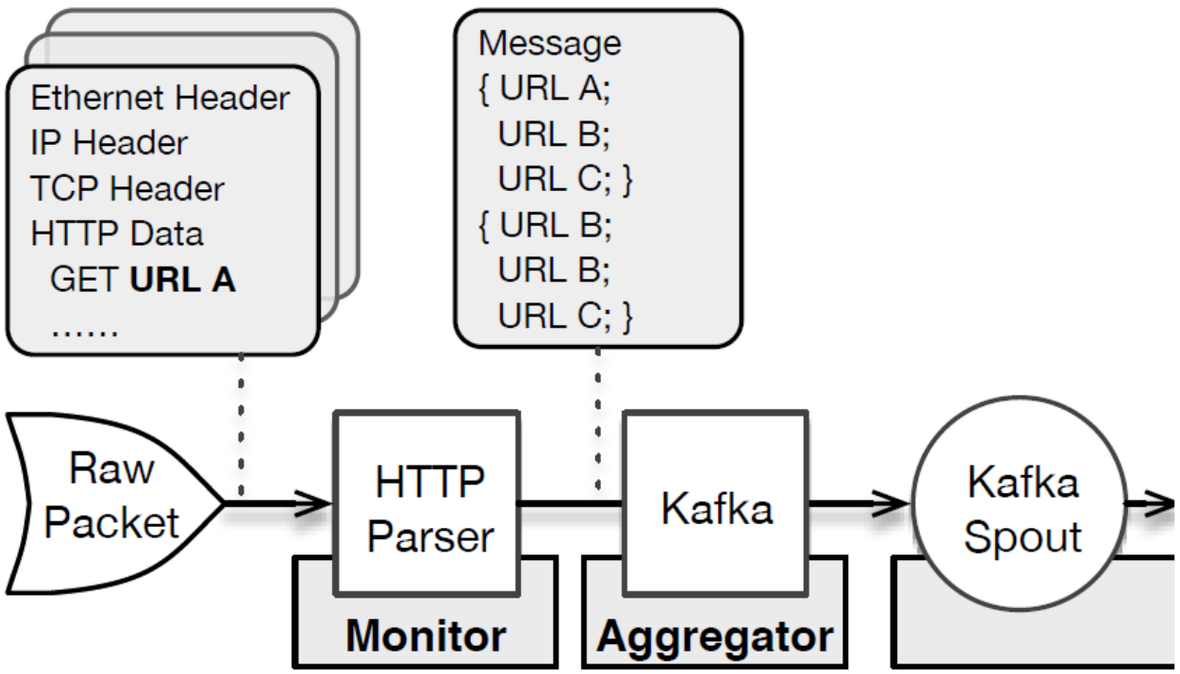
- **Apache Storm** processors
  - Distributed real time computation system

- Managed by **POX SDN** controller
  - Configures packet routing and port mirroring

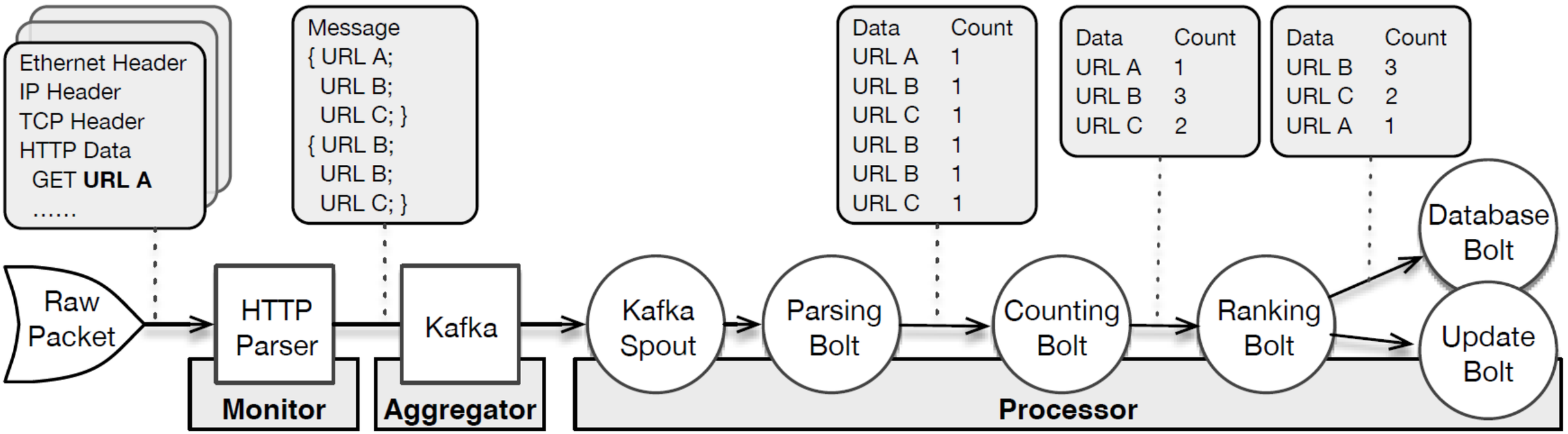
Example Pipeline



Example Pipeline



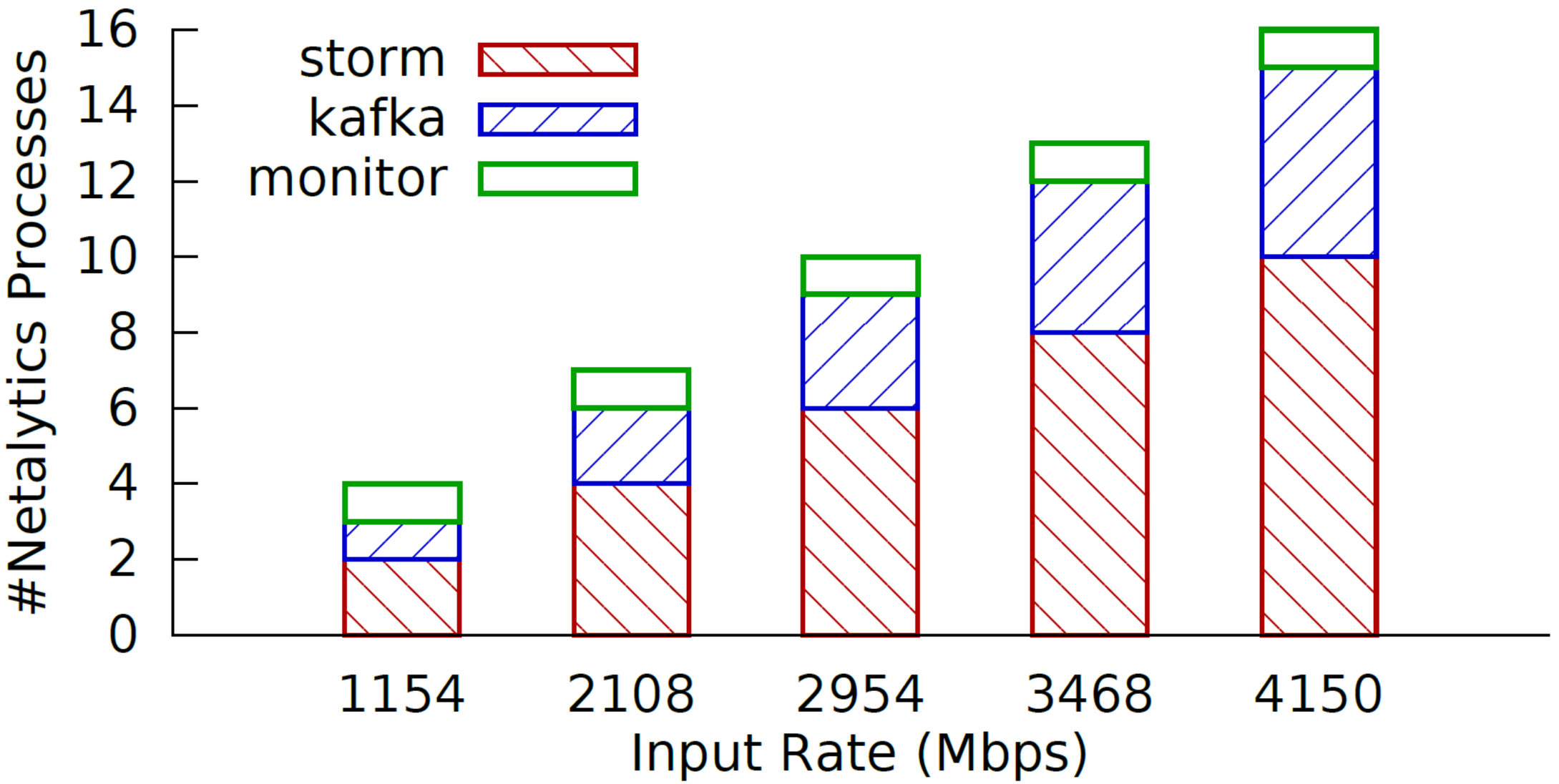
Example Pipeline



Scalable Network Analysis

A single core Monitor can process packets at 10Gbps

16 process cores can analyze packets at 4Gbps



NetAlytics

- Data Plane
  - Building Blocks
  - Implementation
  - Scalability Evaluation
- Control Plane
  - Query Interface
  - Placement Algorithms
- Use Cases

Management

Efficient data plane is necessary but not sufficient for achieving real-time performance analysis!

- What data to gather and how to process them?
  - What data to read from packets?
  - Which flows are important?
  - How to export the final results?
- Where to deploy data processors?
  - What are the task requirements?
  - What are the position constraints?
  - How to optimize resource utilization?

Query Interface

```
PARSE
FROM                          TO
LIMIT      SAMPLE
PROCESS
```

Query Interface

```
PARSE http_get
FROM S1:*,S2:*,S3:* TO D1:80,D2:80,D3:80
LIMIT 90s SAMPLE 0.1
PROCESS (top-k: k=10, w=10s)
```

- What data to gather?
  - PARSE **`http_get:`** start http_get parser for identified monitors.
  - FROM **`S1:*,S2:*,S3:*:`** monitor all flows originating from hosts S1, S2, S3
  - TO **`D1:80,D2:80,D3:80:`** monitor flows destined to port 80 of hosts D1, D2, D3

Query Interface

```
PARSE http_get
FROM S1:*,S2:*,S3:* TO D1:80,D2:80,D3:80
LIMIT 90s SAMPLE 0.1
PROCESS (top-k: k=10, w=10s)
```

- How to process the data?

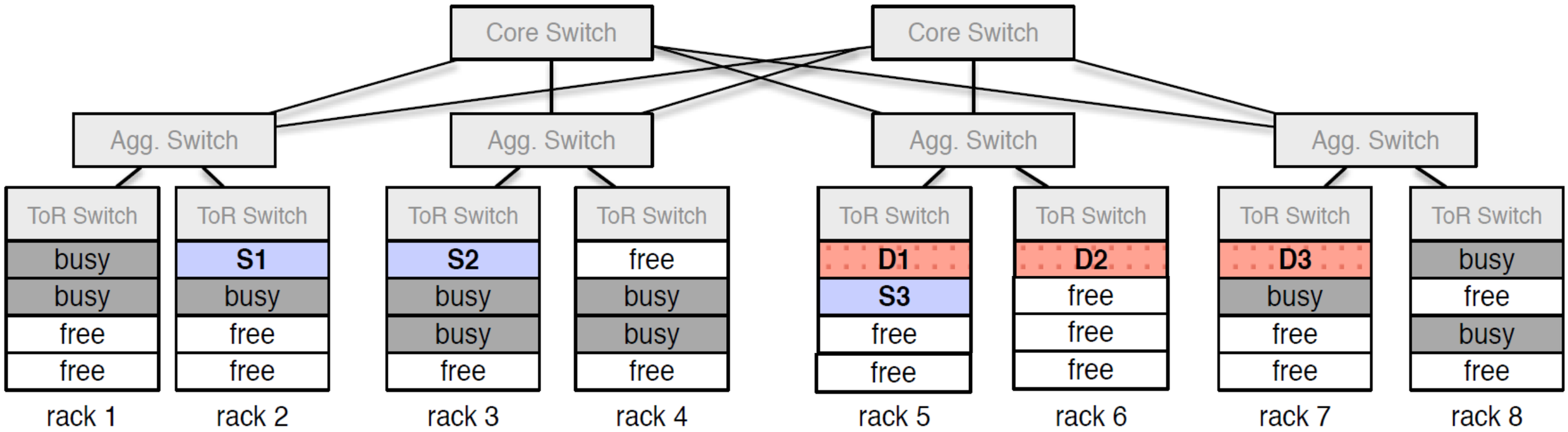
- LIMIT 90s: run this monitor task for 90 seconds.

- SAMPLE 0.1: sample 1 in 10 packets.

- PROCESS **`(top-k: k=10, w=10s)`**: start top-k processor with parameters k=10 and w=10s.

Placement Algorithms

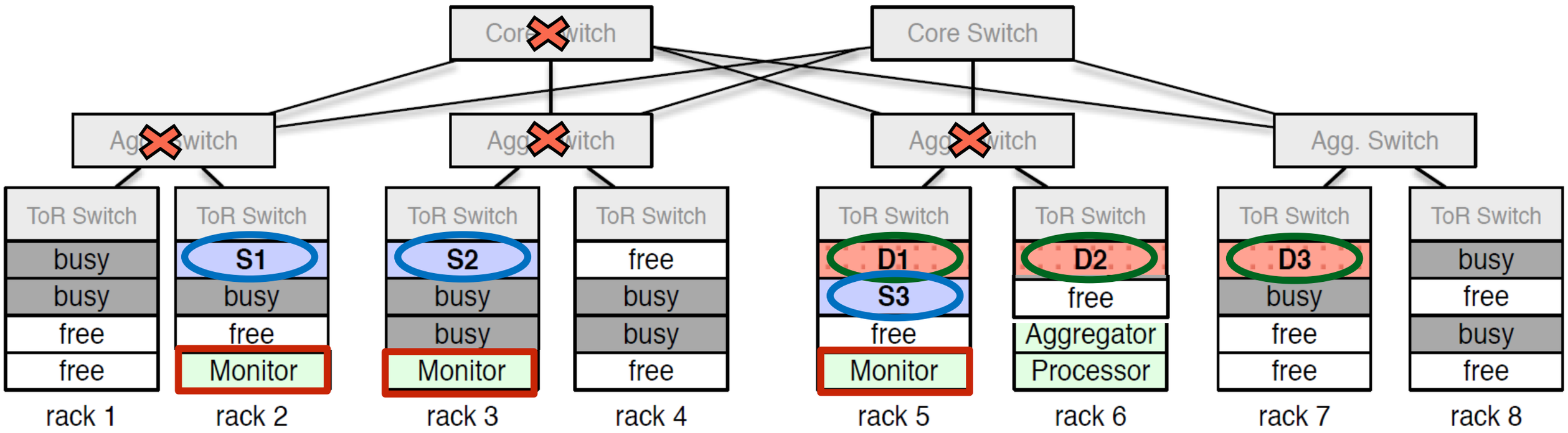
- Where to deploy monitors, aggregators and processors?
  - Steps:
    - Translate the query into a set of switches where monitors must be installed
    - Place aggregators and processors based on the monitors' positions and the task requirements
  - Task requirements:
    - Short-term or long-term query?
    - Network constrained or CPU constrained?



Placement Algorithms

- Where to deploy monitors, aggregators and processors?

```
PARSE http_get FROM S1:*,S2:*,S3:* TO D1:80,D2:80,D3:80
LIMIT 90s SAMPLE 0.1 PROCESS (top-k: k=10, w=10s)
```

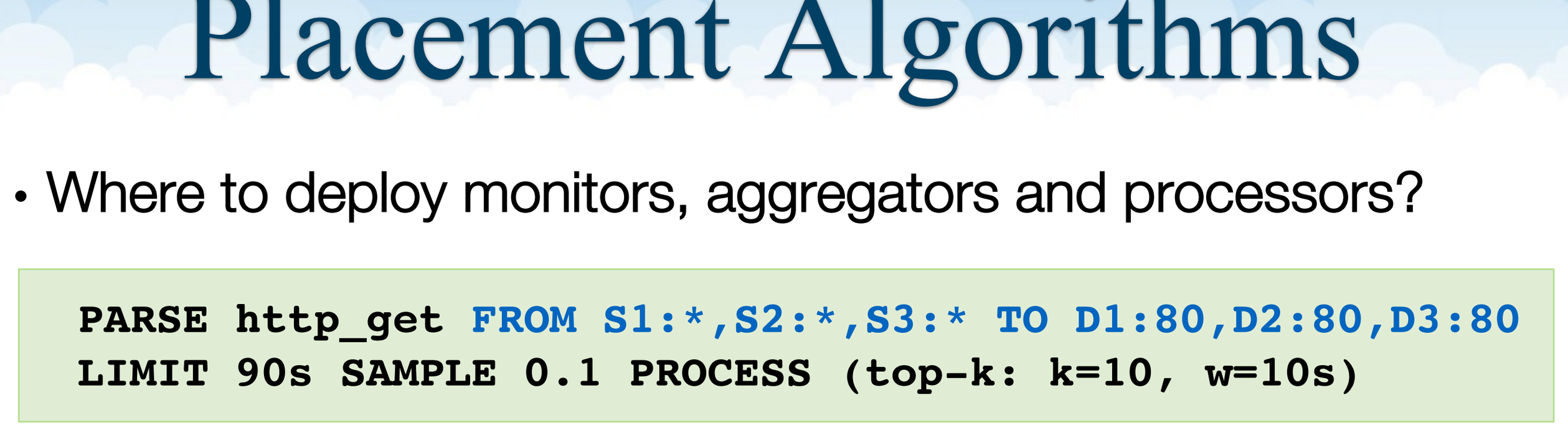


Placement Algorithms

- Where to deploy monitors, aggregators and processors?

```
PARSE http_get FROM S1:*,S2:*,S3:* TO D1:80,D2:80,D3:80
LIMIT 90s SAMPLE 0.1 PROCESS (top-k: k=10, w=10s)
```

- Trick: Localize monitoring traffic



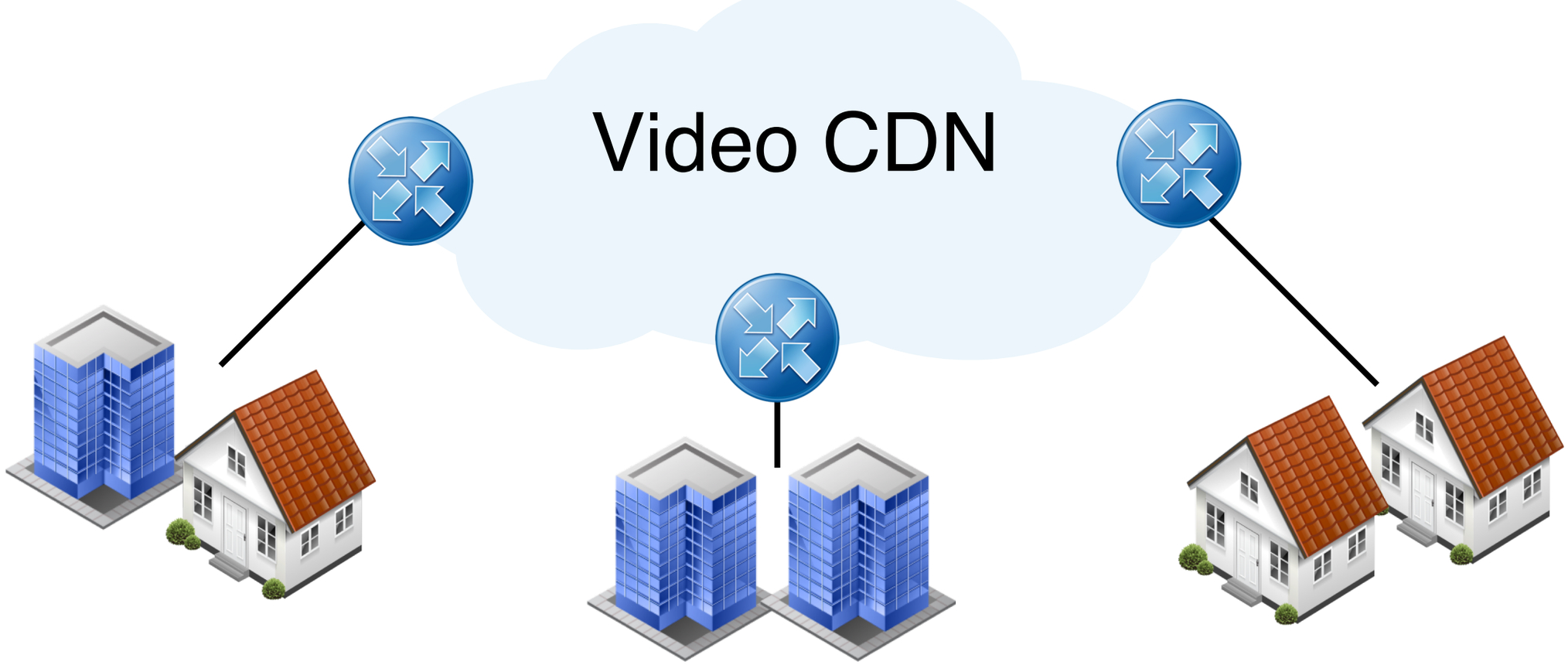
Netalytics

- Data Plane
- Control Plane
- Use Cases
  - Real-time popularity monitoring

Live Popularity Detection and Caching

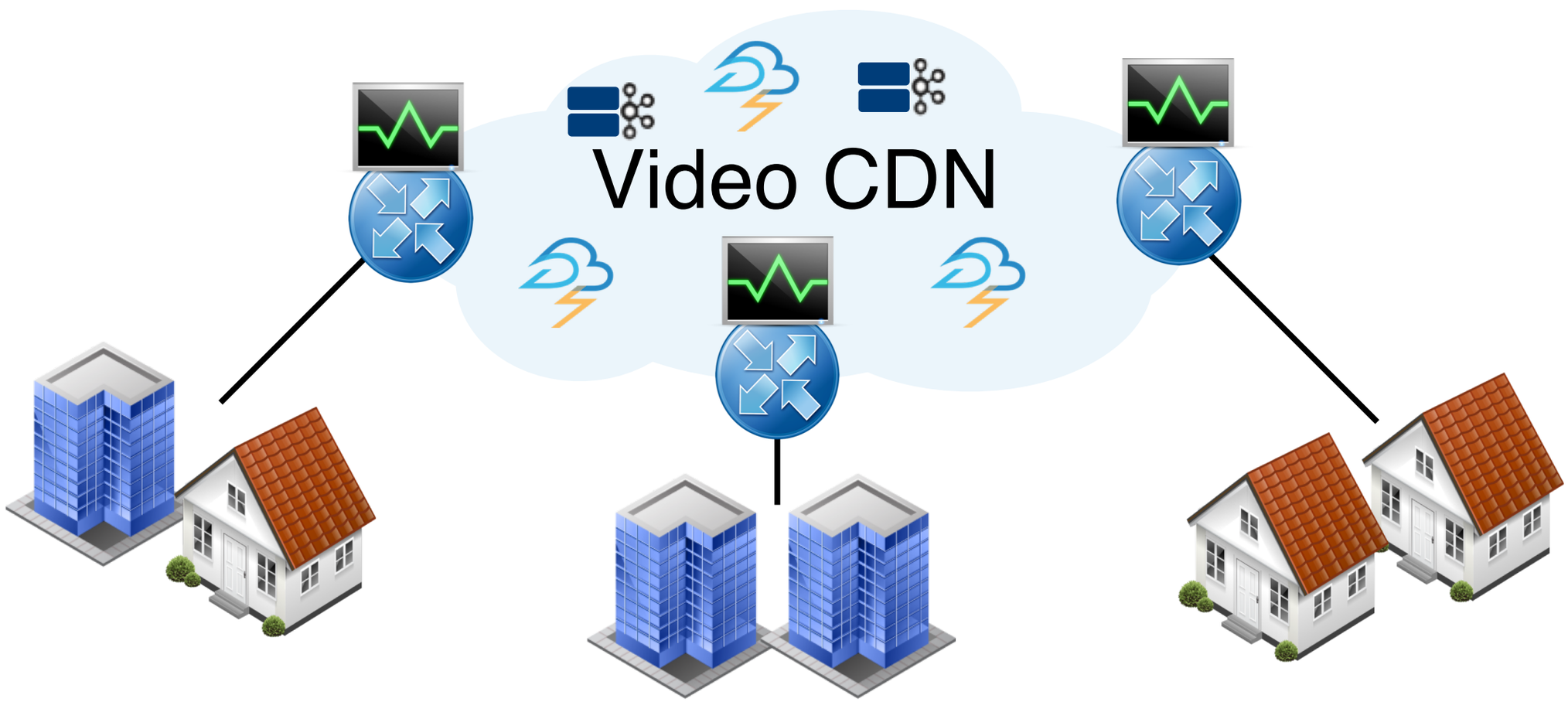
- What is the most popular content *right now*?
- How to allocate cache resources for this content?

```
PARSE http_get FROM * TO 10.1.1.2:80
SAMPLE 0.1 PROCESS (top-k: k=10, w=10s)
```



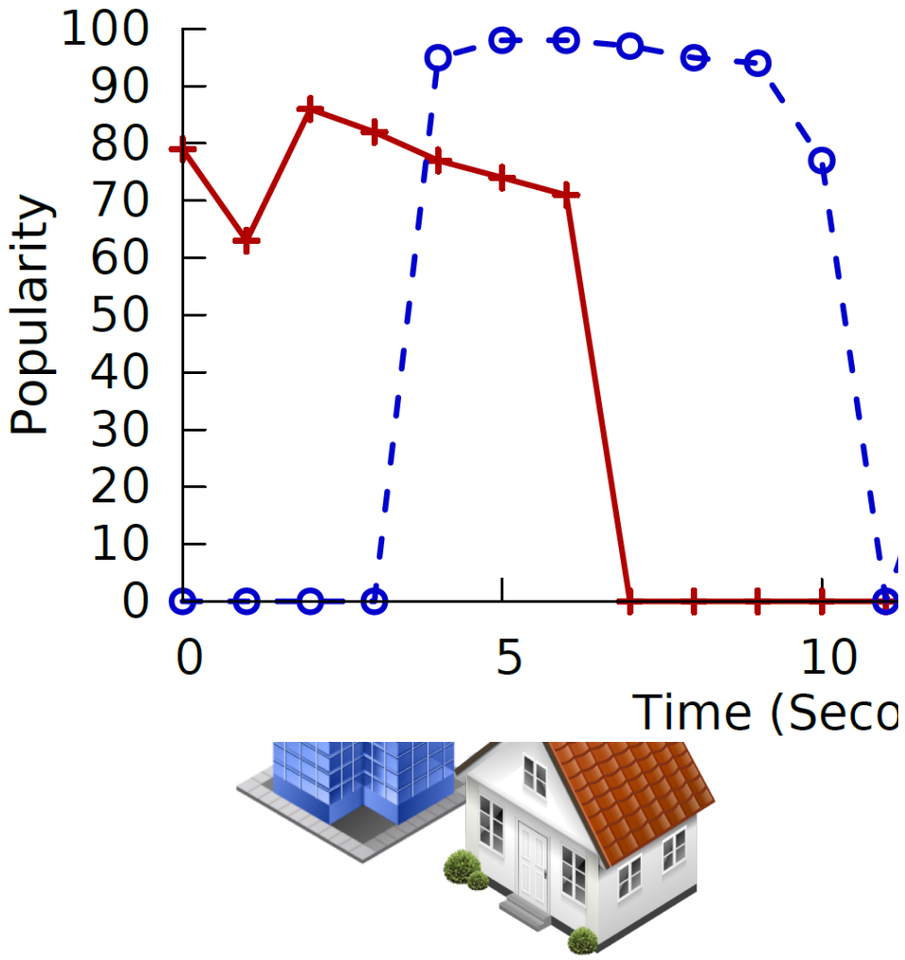
Live Popularity Detection and Caching

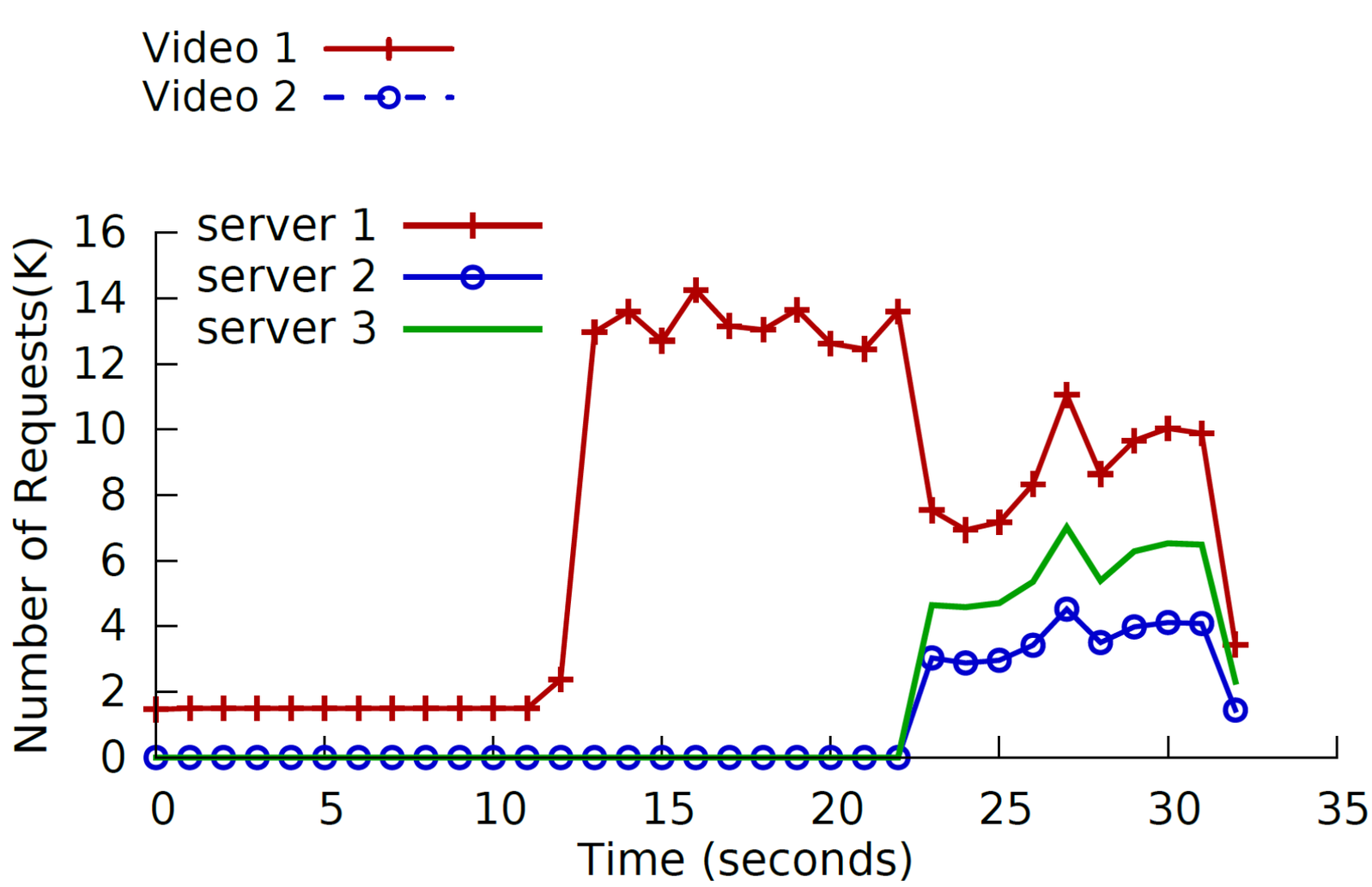
- Monitors parse HTTP requests for requested URL
- Processors use **top-k** streaming analytics algorithm
- Feedback video popularity to replication system



Live Popularity Detection and Caching

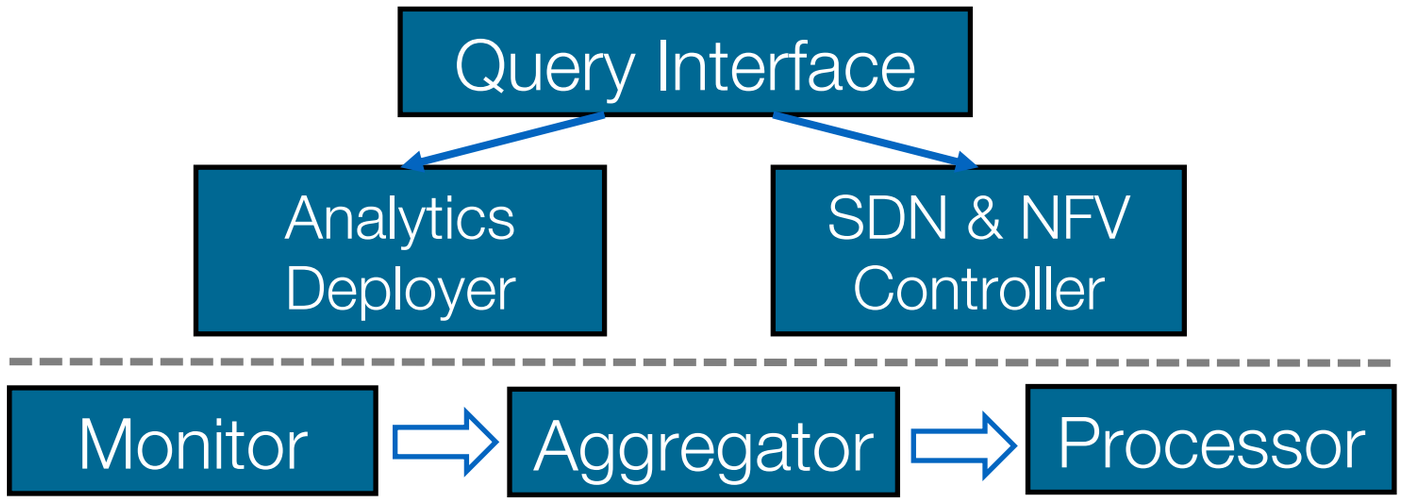
NetAlytics can gather complex information lively across a cloud and use it to drive management





Conclusion

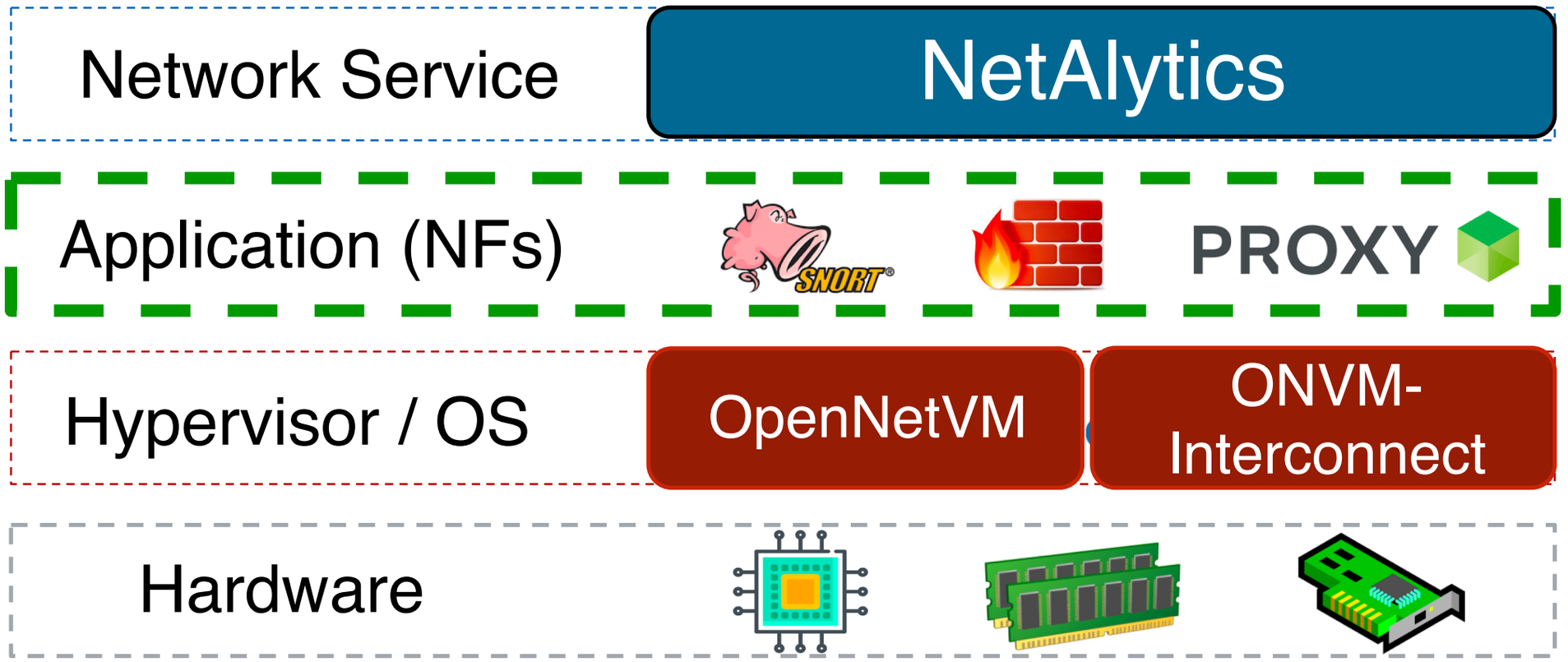
NetAlytics turns the network from a communication pipe into a customizable monitoring infrastructure



- Flexible **query interface** to manage data parsing and processing
- Fast **placement algorithms** to place processes and minimize resource consumption
- **SDNs** redirect certain traffic flows so they can be monitored
- **Monitors** use **NFV** to provide efficient, software measurements
- **Aggregators** and **Processors** provide scalable message queuing and real time stream processing

Outline

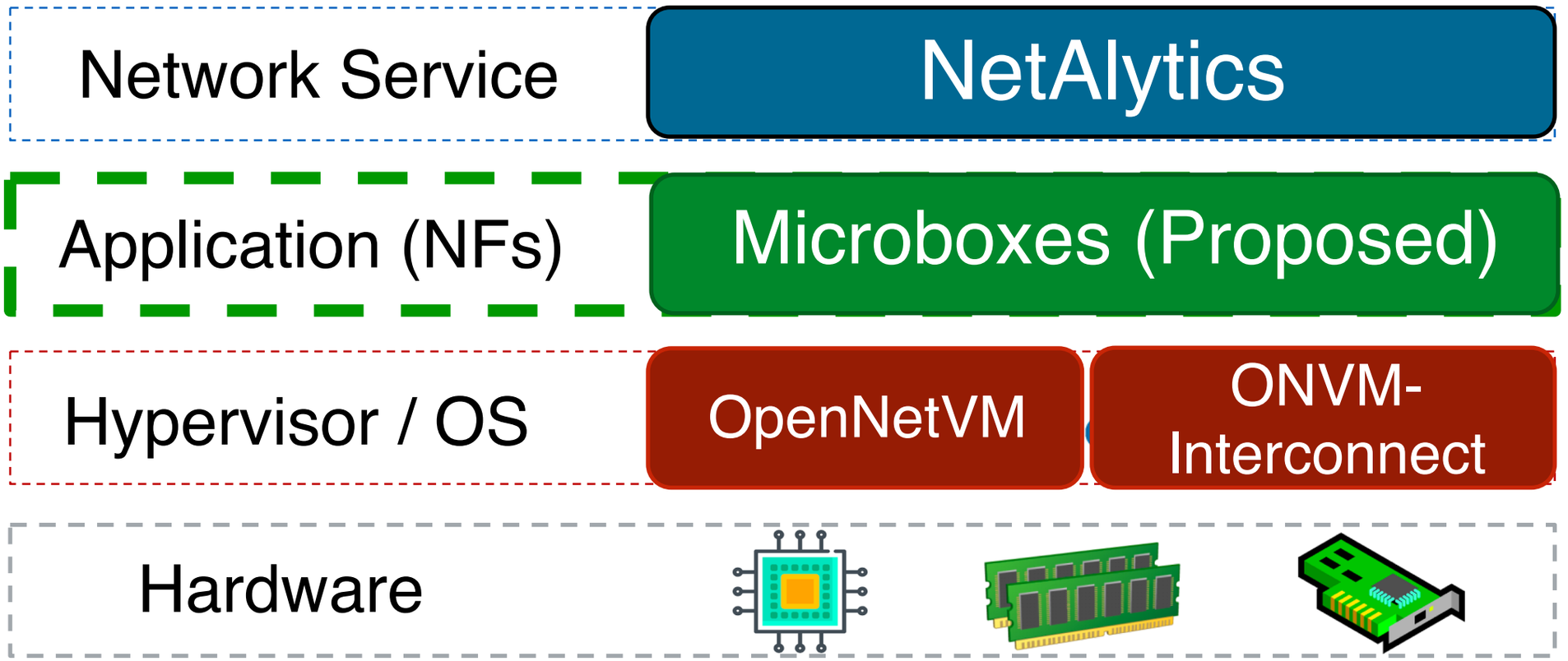
- Application Layer:
  - existing legacy applications are not designed to coordinate with other applications and don't have unified APIs
- How can we provide **flexible abstractions** to enable modular and customizable applications?



Outline

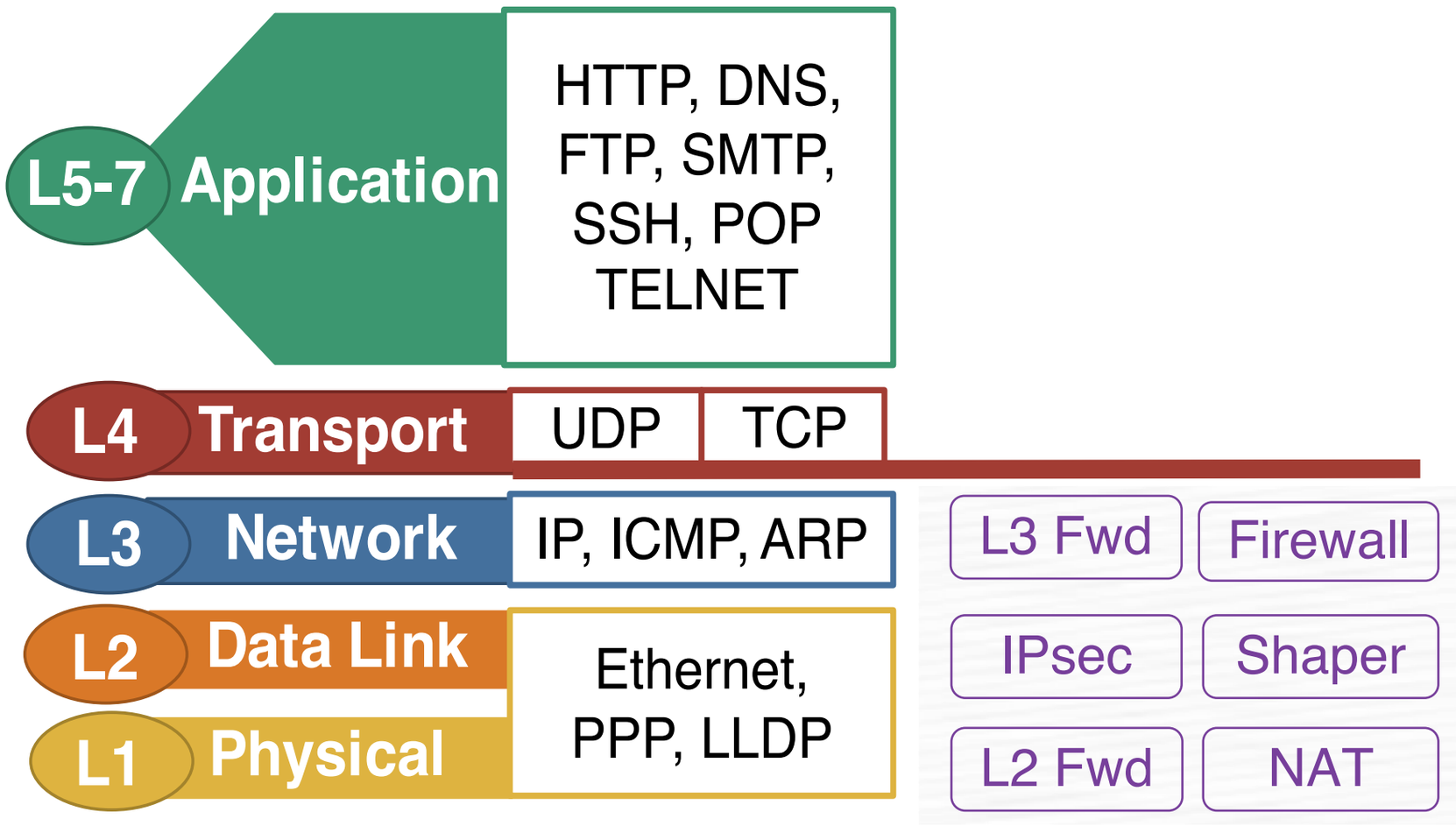
- **Microboxes** [SIGCOMM'18]: a customizable TCP stack for transport and application layer network functions

Replace the monolithic TCP stack with modular and composable stacks



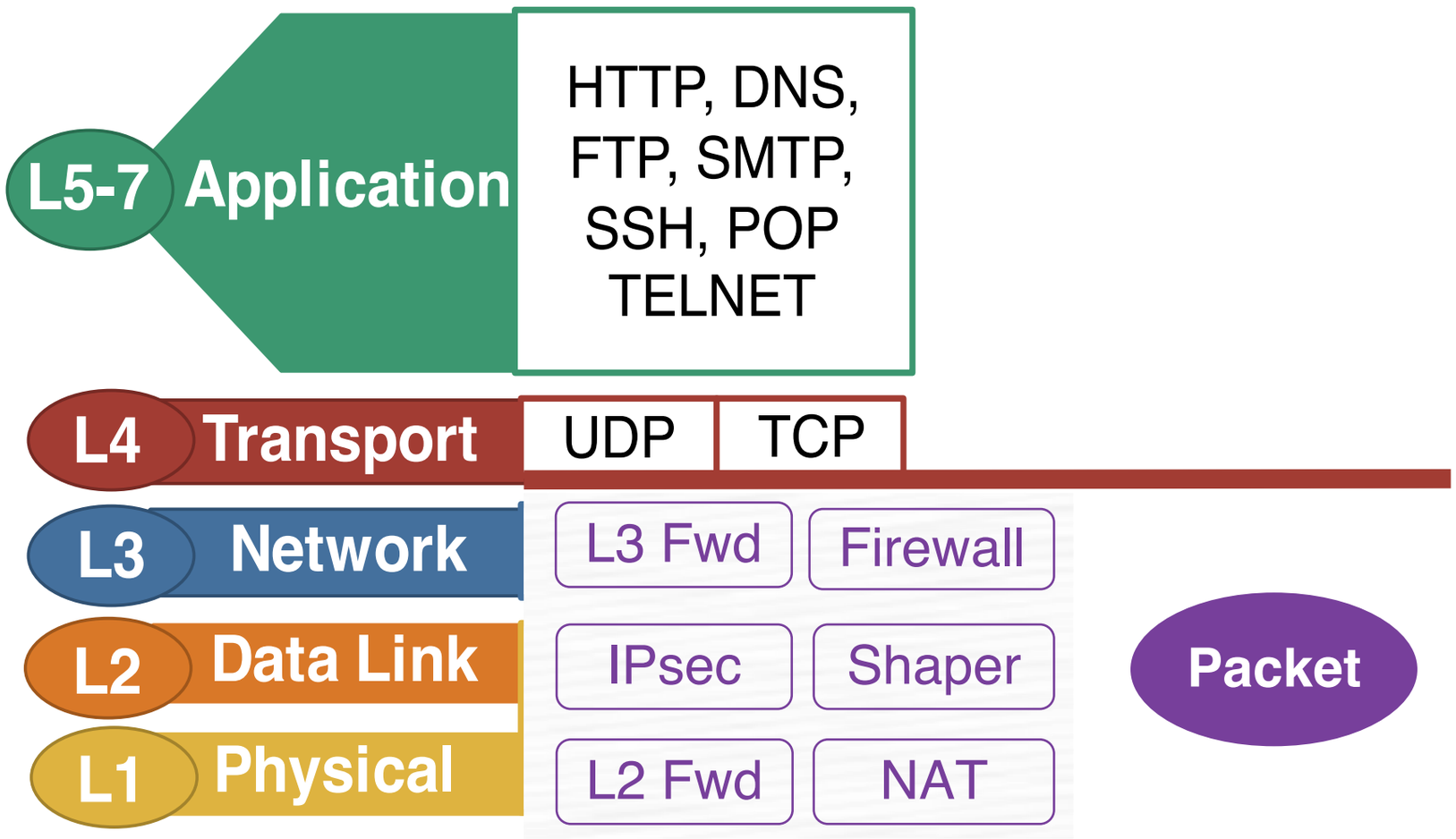
Why Improve Existing NFV Frameworks?

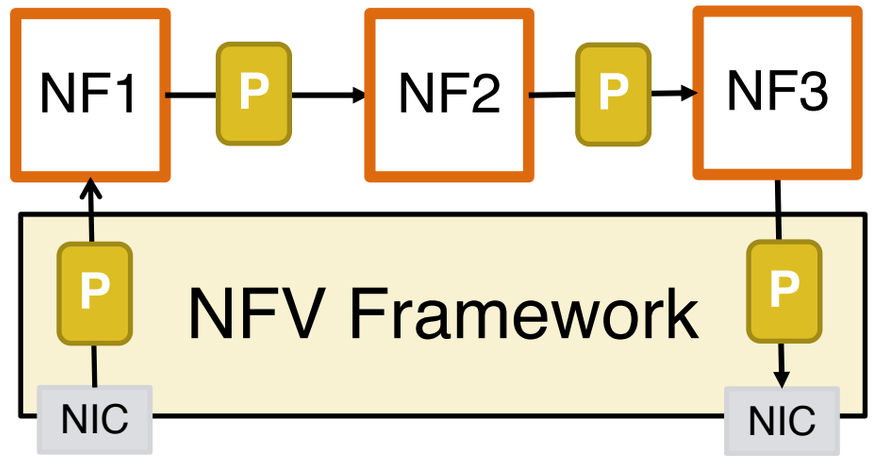
- Existing NFV frameworks focus on L2/L3 processing
- Parse protocols up to the network layer
- Limits the types of NFs that can support



Why Improve Existing NFV Frameworks?

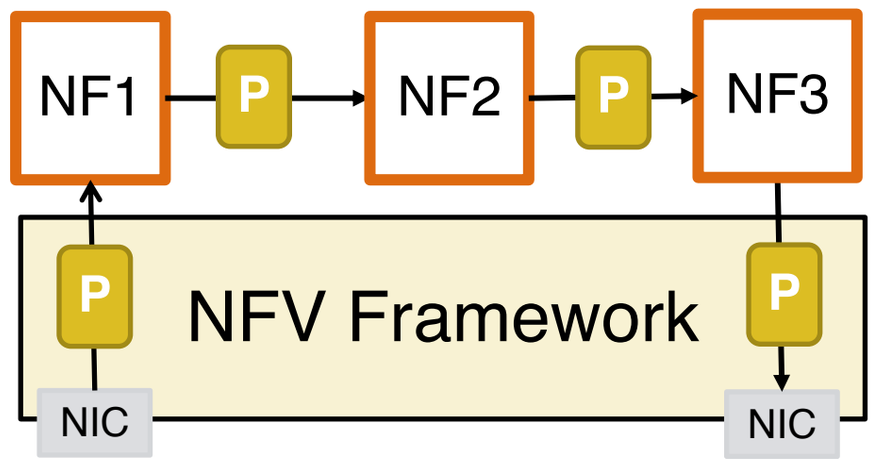
- Based on a packet-centric model: each NF in the chain is given every packet for processing





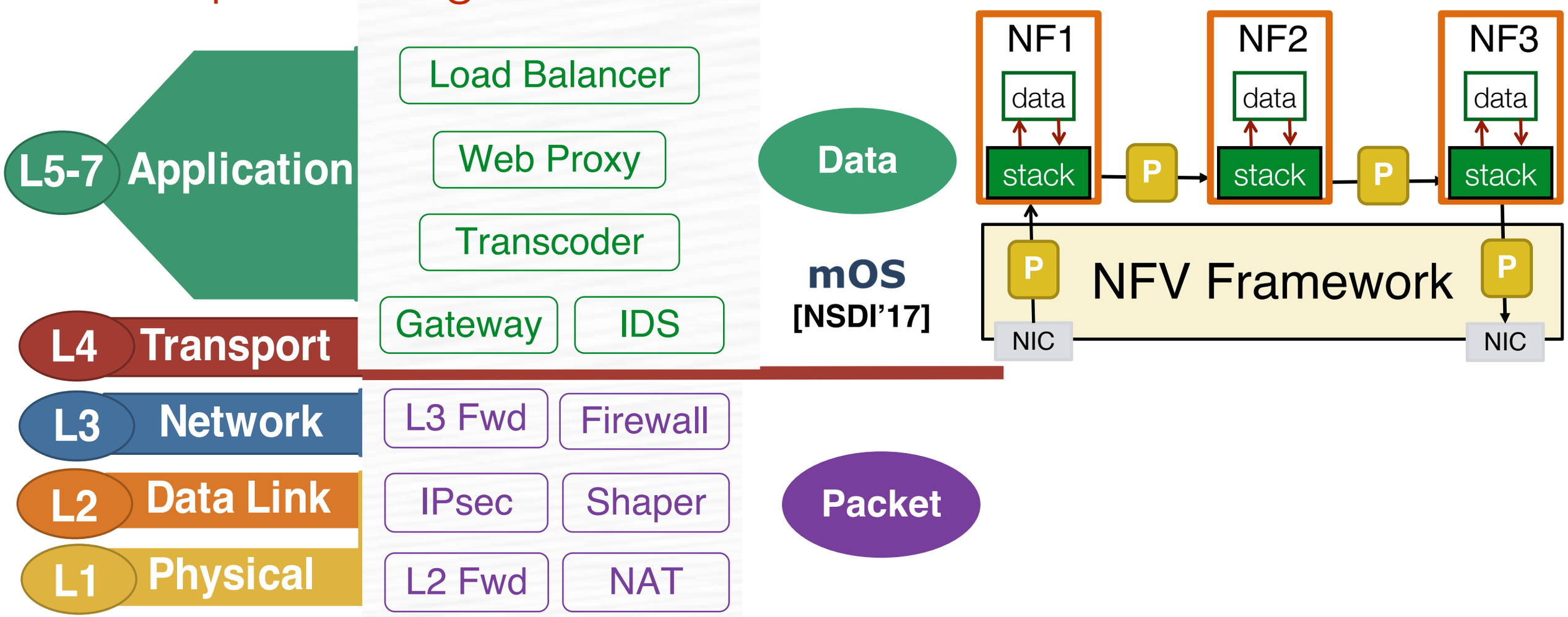
Why Improve Existing NFV Frameworks?

- Application level protocols care about data instead of individual packet
- Rely on the transport layer to deliver correct data



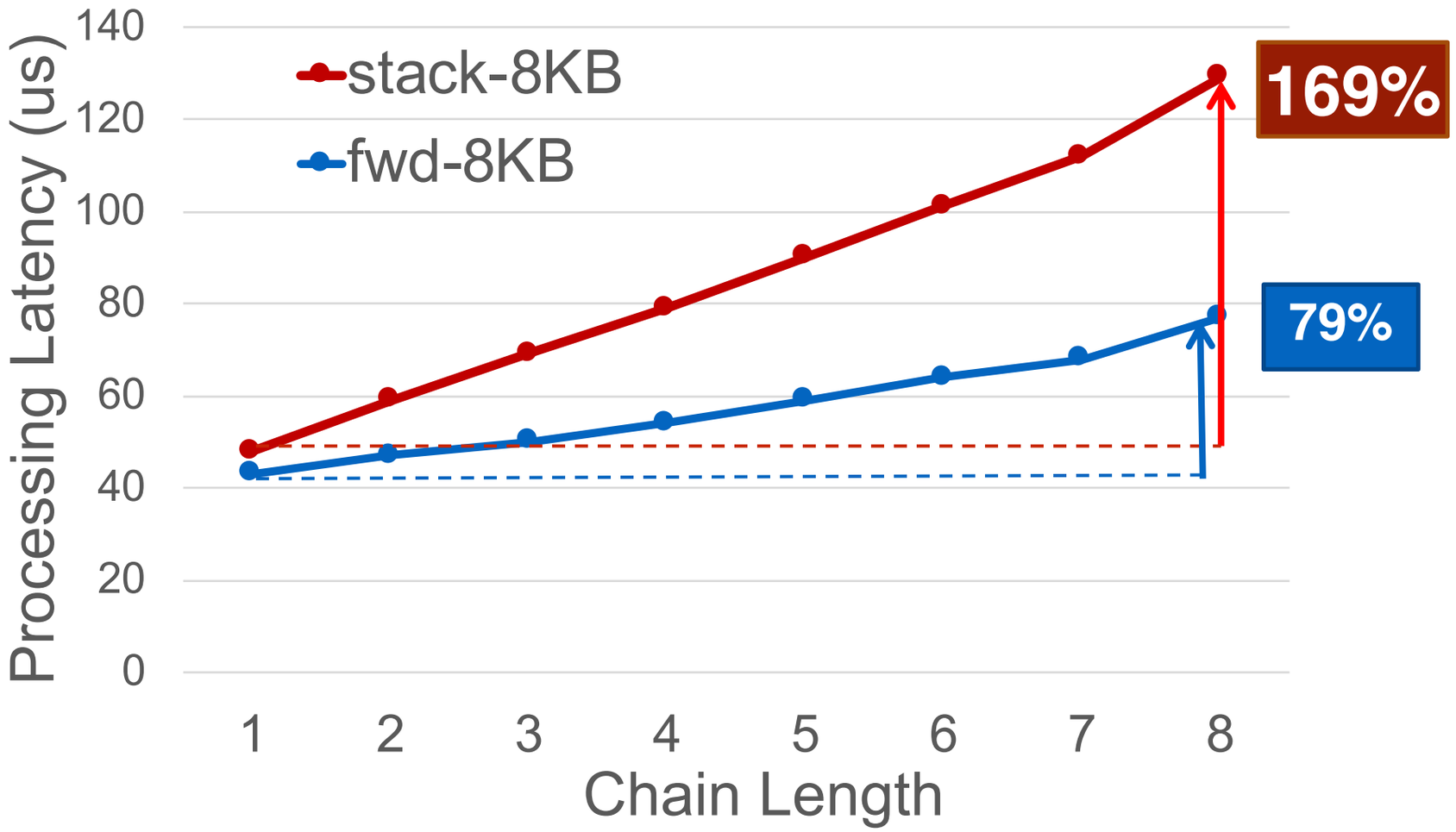
Why Improve Existing NFV Frameworks?

- Current solution with packet model needs to incorporate a TCP stack for each NF
- Protocol processing is part of the NF - repeated protocol stack processing within a chain. Redundant!



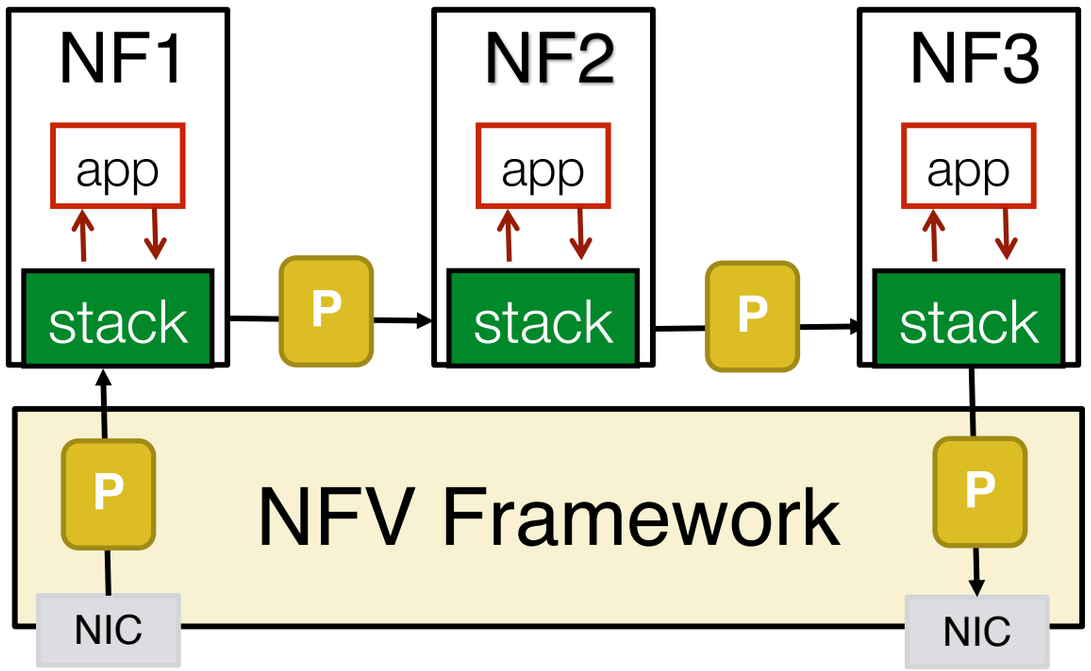
Redundant Stack Processing

- Compare the processing latency between NF doing stack processing and simple forwarding NF
- As the chain length increases, the overhead grows significantly when going through stack processing multiple times



Consolidate Stack Processing

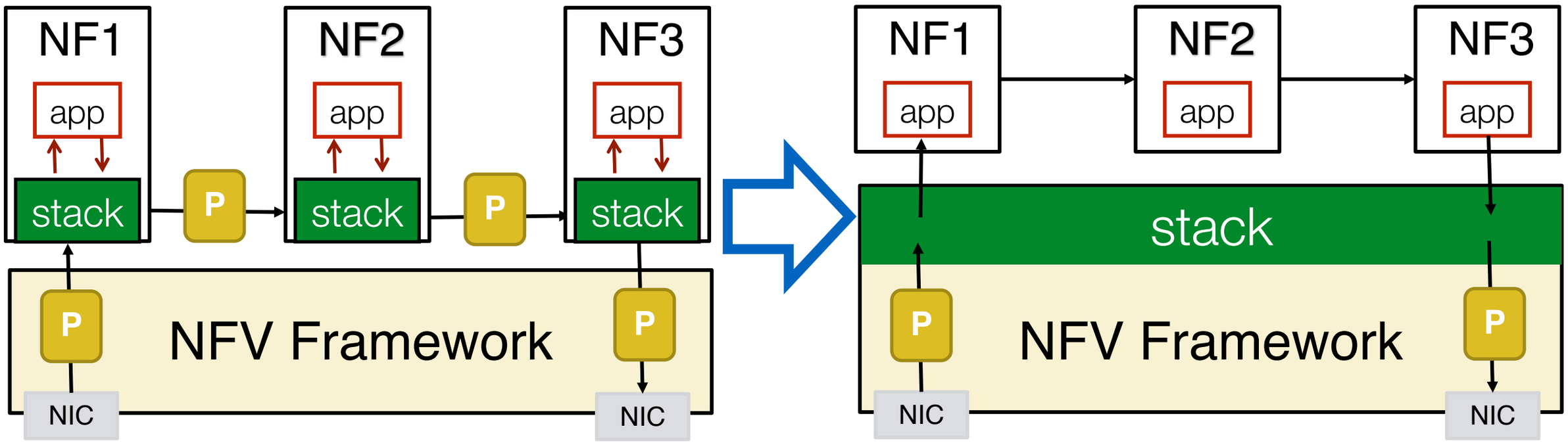
- How can we remove the redundancy within a chain?



Consolidate Stack Processing

- How can we remove the redundancy within a chain?

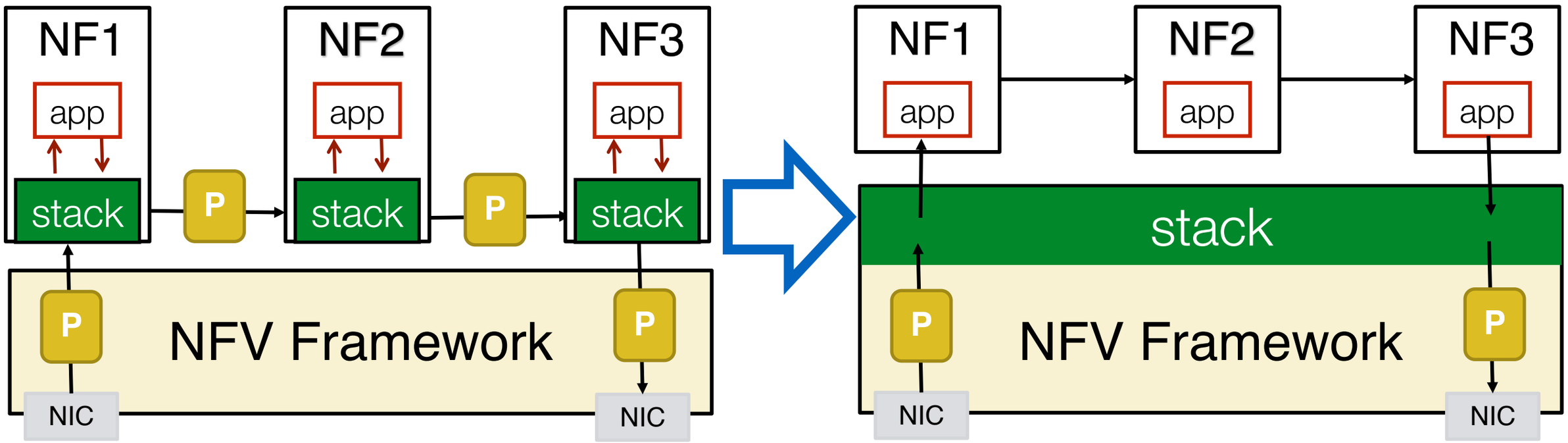
Move stack processing from NF to the NFV framework



Consolidate Stack Processing

- How can we remove the redundancy within a chain?

Move stack processing from NF to the NFV framework

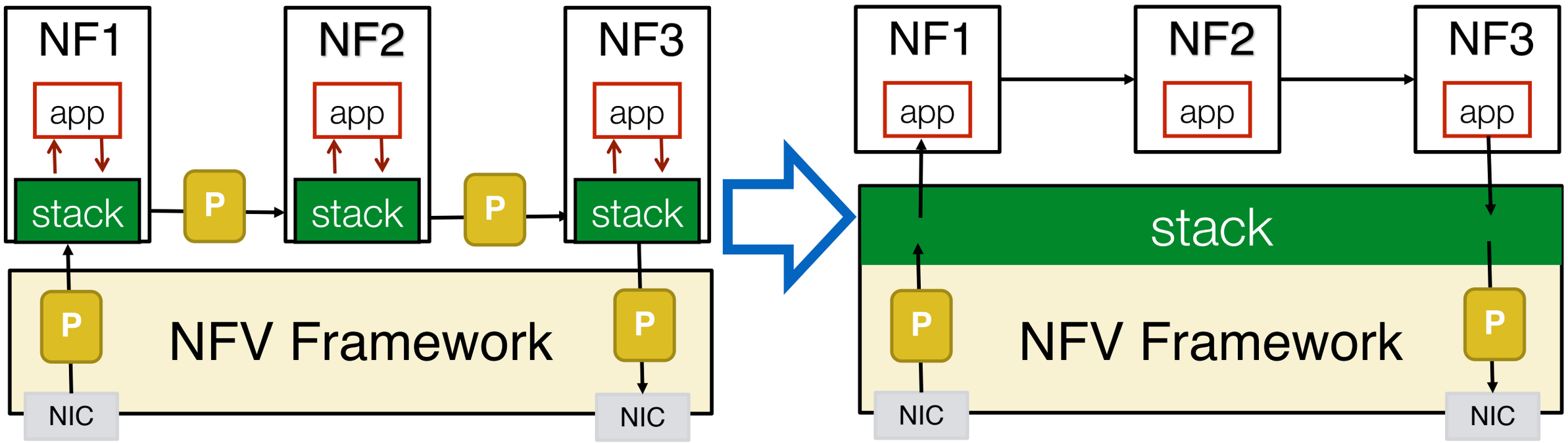


How to avoid unnecessary processing in the stack?

Consolidate Stack Processing

- How can we remove the redundancy within a chain?

Move stack processing from NF into NFV framework



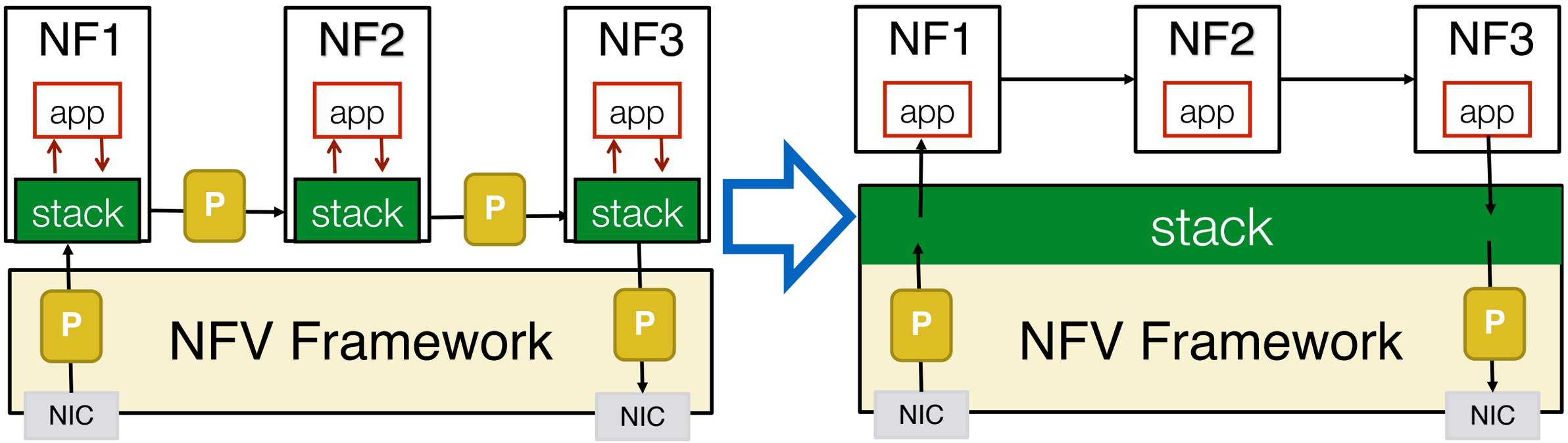
How to avoid u… processing in t…

How to ensure consistency while maximizing the performance ?

Consolidate Stack Processing

- How can we remove the redundancy within a chain?

Move stack processing from NF into NFV framework



How to avoid u
processing in t

How to ensure consi
maximizing the per

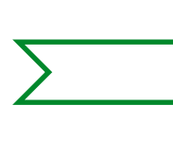
What is the right interface between the stack and NFs?

Microboxes Preview

x Redundant Stack Processing

x A Monolithic Stack

x Separate Stacks/Interfaces



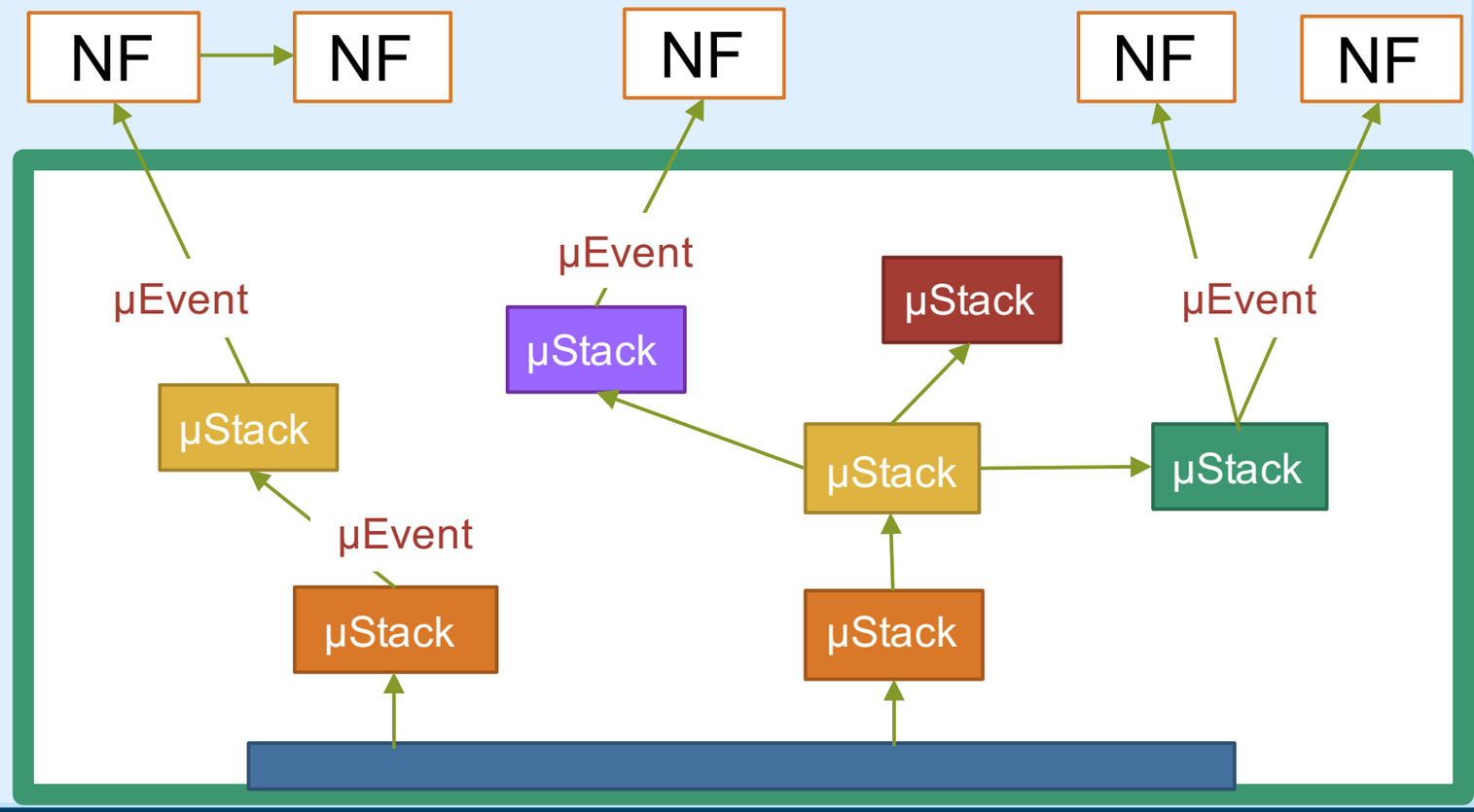
✓ Consolidate Stack Processing

✓ Customizable Stack Modules

✓ Unified Event Interface

Microboxes

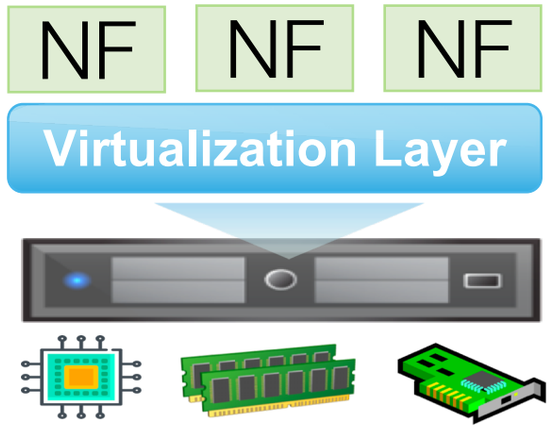
= *μ*Stack + *μ*Event



Conclusion

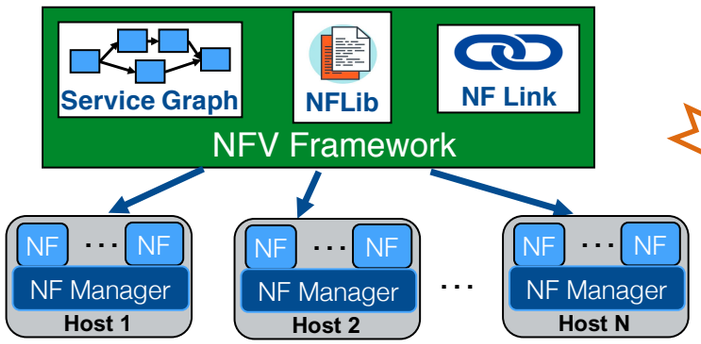
Network Devices

OpenNetVM



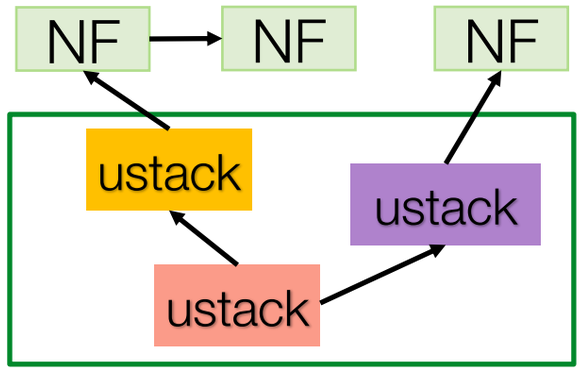
Performance

ONVM-Interconnect



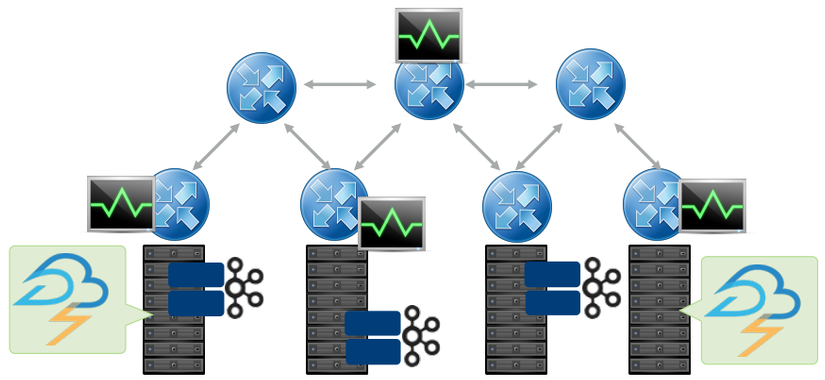
Management

Microboxes



Flexibility

NetAlytics



Scalability

Replace hardware devices with virtual NFs

Separate NF management from packet processing

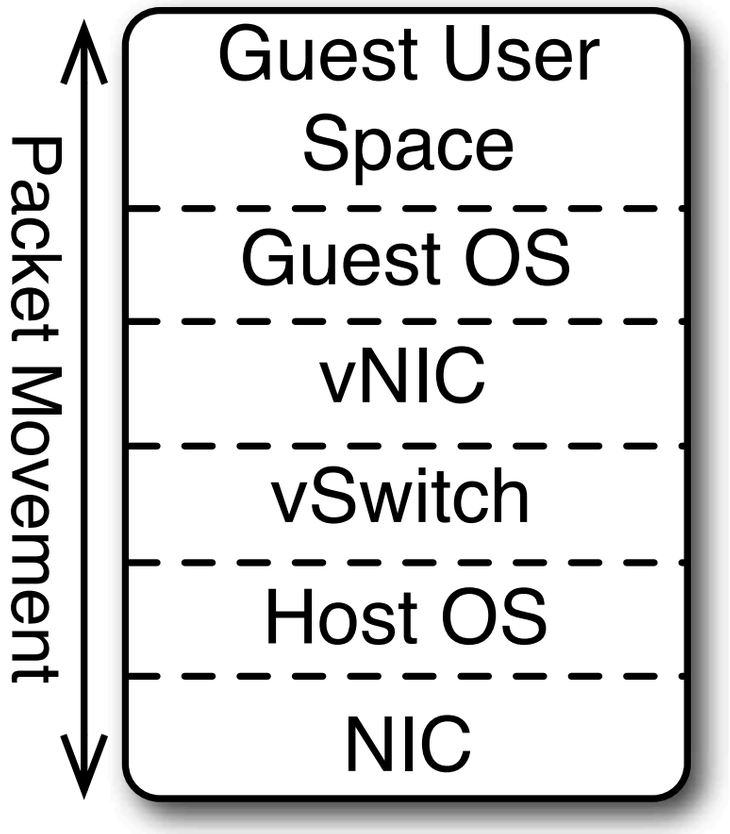
Decouple stack processing from NFs to the NFV framework

Embed intelligence into the network and provide scalable services

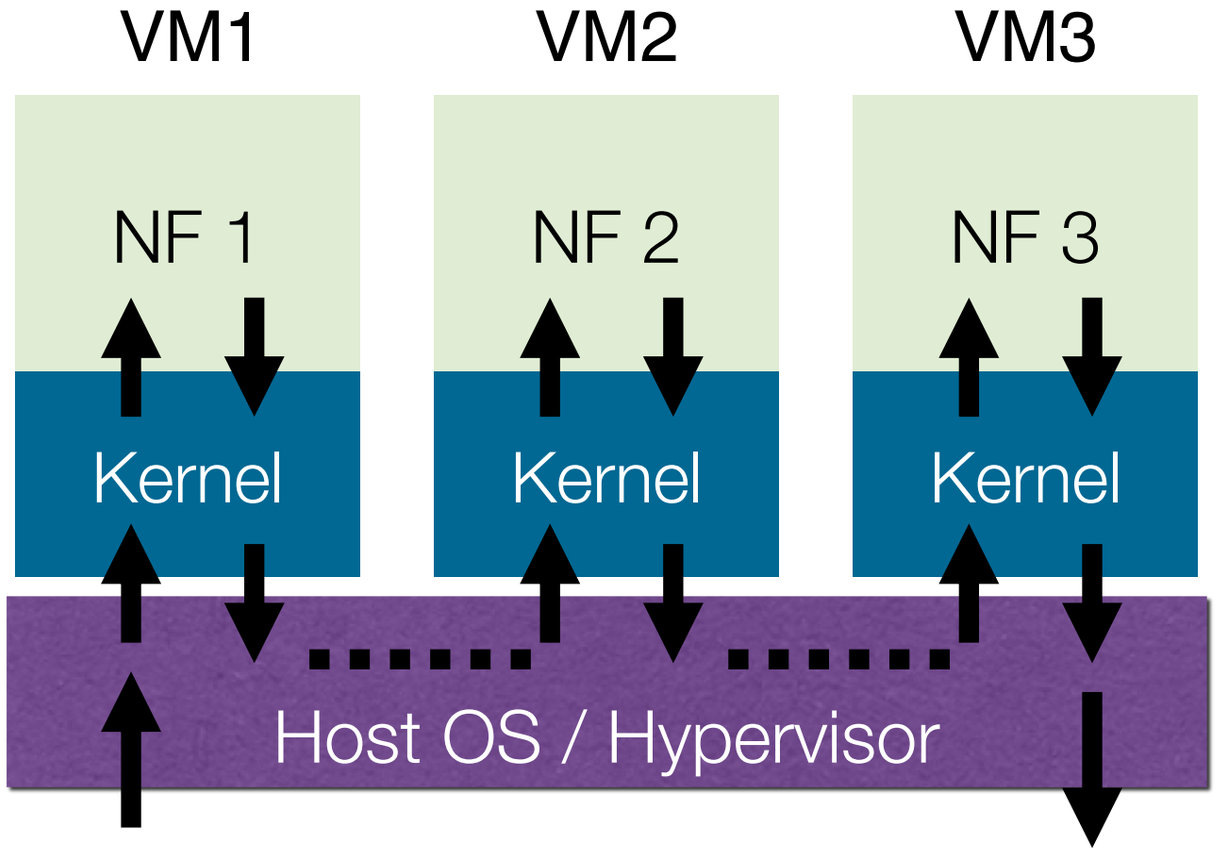
Backup Slides

Performance Challenge

An NFV platform must maximize I/O performance and be optimized to avoid bottlenecks in the management layer.

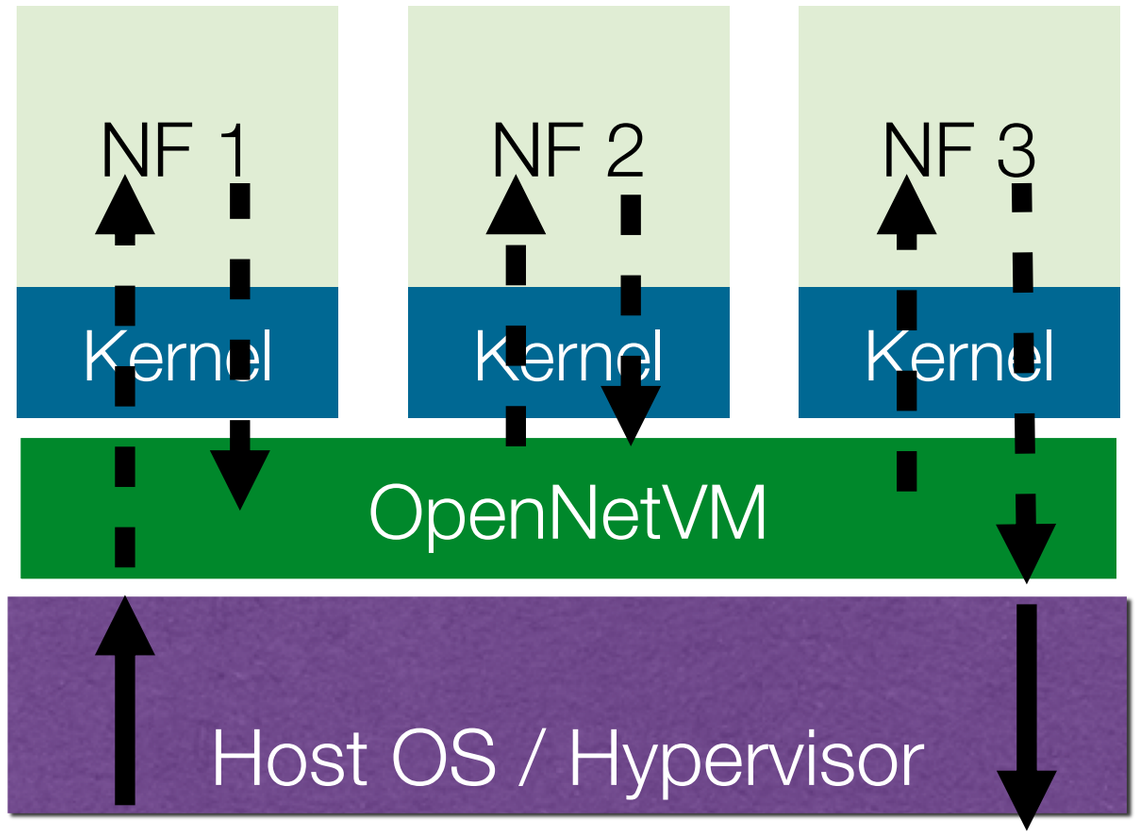


Traditional: 2 DMAs + 12 copies



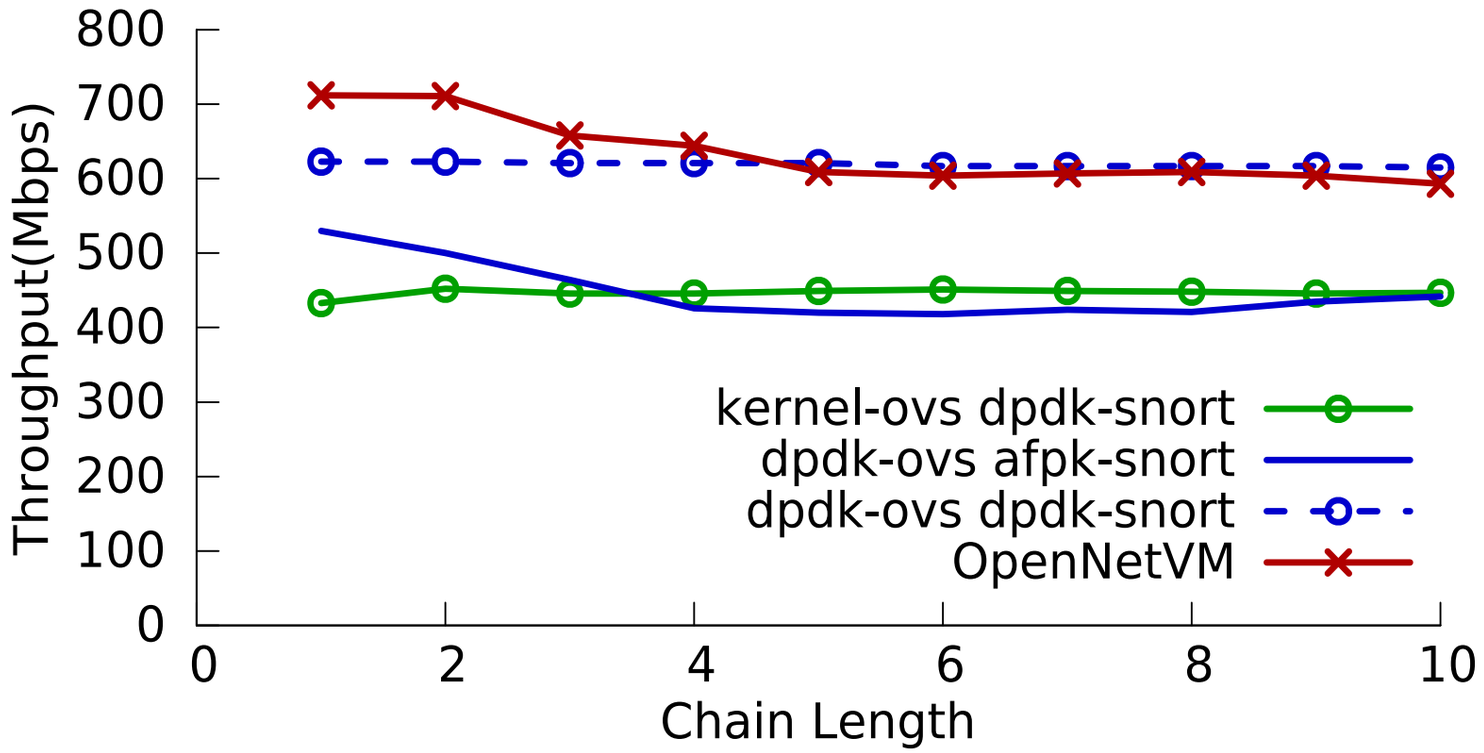
Zero Copy I/O

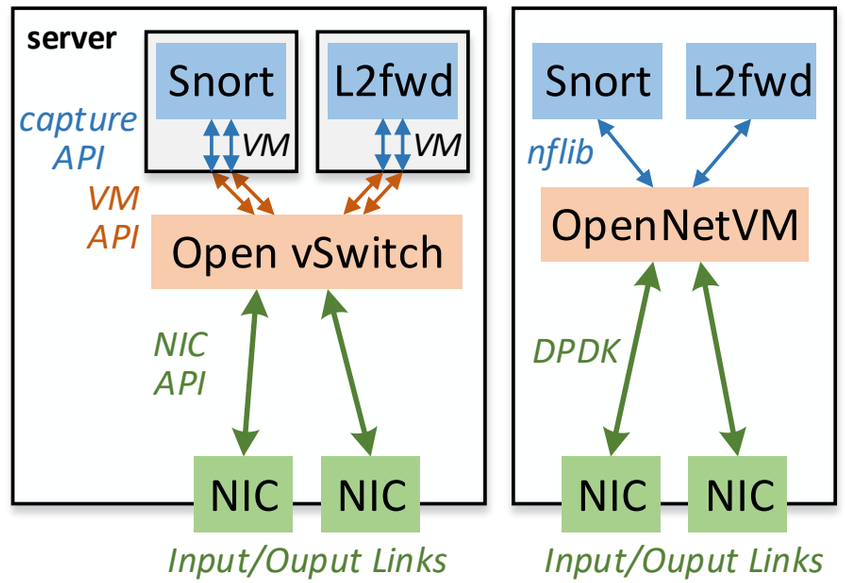
- OpenNetVM is designed for NF service chains
  - Don't want to copy packet multiple times to move between NFs
  - DMA packets into shared memory, directly accessible to NFs!



OpenNetVM Evaluation

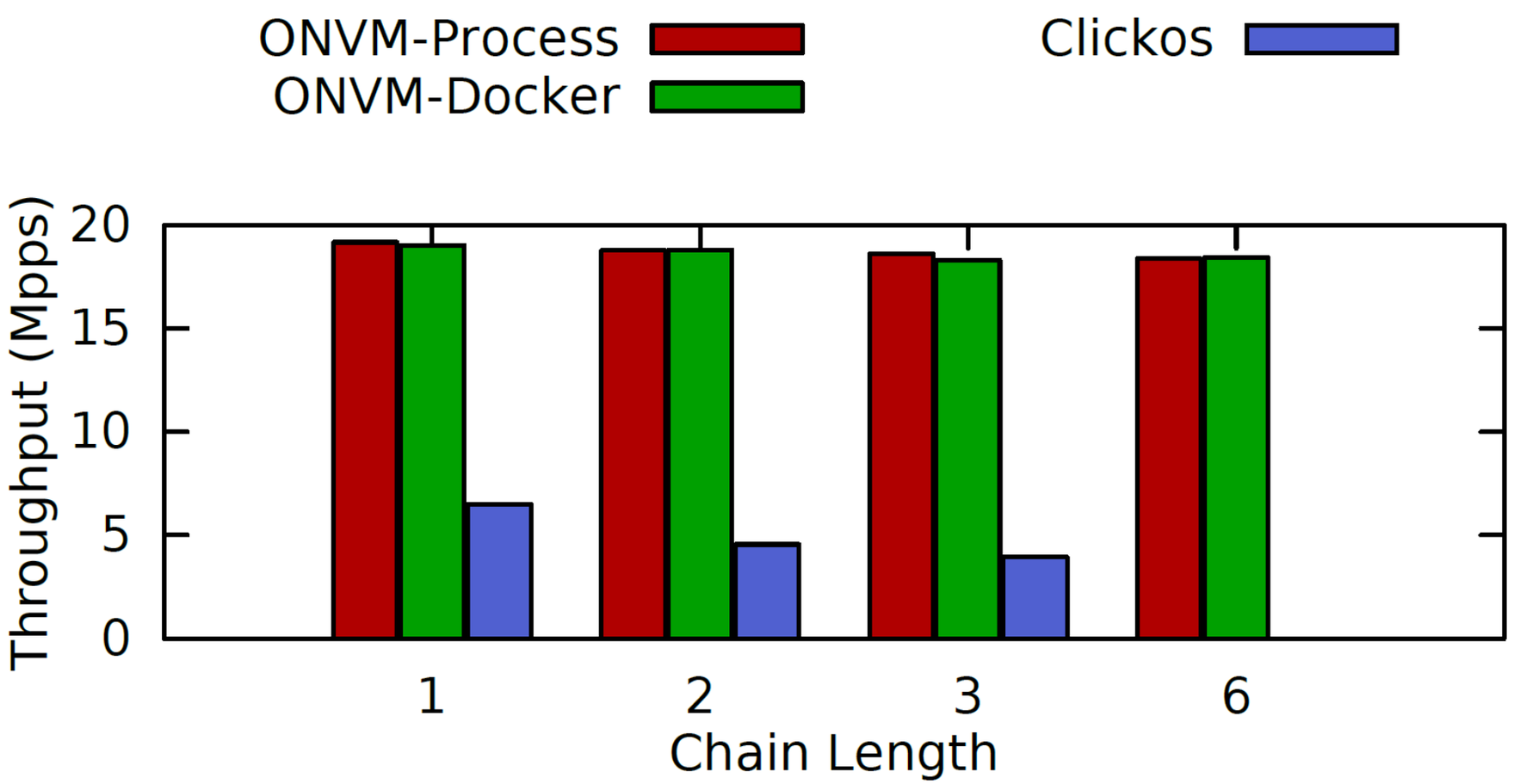
- Snort: Network Intrusion Detection
  - snort-afpk: kernel L2 redirect
  - snort-dpdk: dpdk (bypass VM kernel)
- Open vSwitch (ovs): software switch used in industry
  - ovs-kernel: Linux kernel module + vhost-net
  - ovs-dpdk: dpdk (bypass host kernel) + vhost-user
- OpenNetVM: pointer rings + packet shared memory





Service Chain Performance

- Negligible performance difference between processes and containers.
- OpenNetVM sees only a 4% drop in throughput for a six NFs chain, while ClickOS falls by 39% with a chain of three NFs.

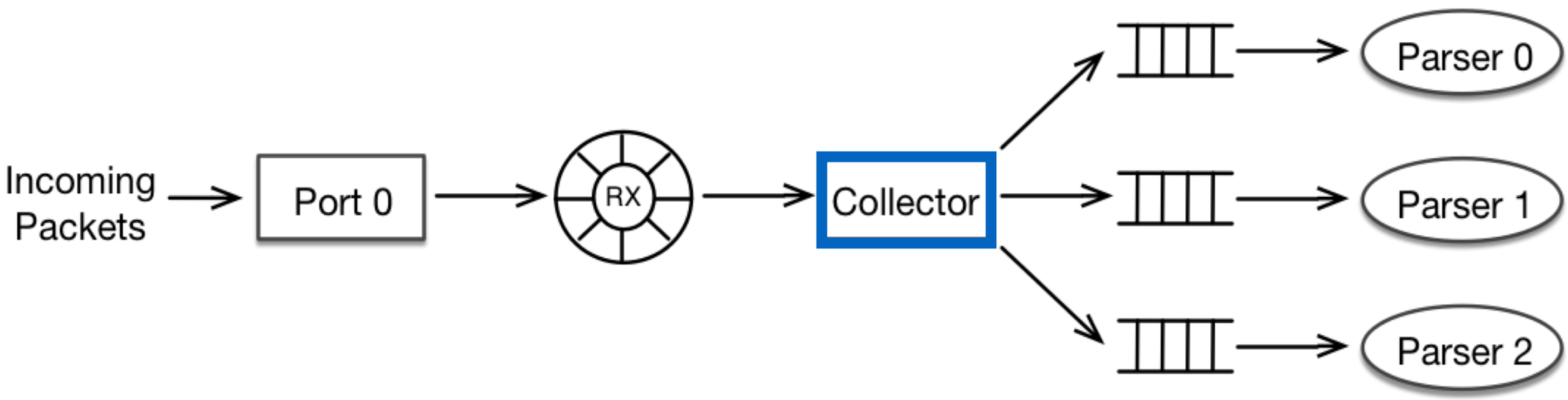


New Monitoring Solution

	Hardware-Monitoring	**Softeare-Logging**
Monitor	Hardware	Software
Analyze	Online	Offline
Deployment Cost	✗	✓
Storage Cost	✓	✗
Flexibility	✗	✓
Performance	✓	✗
Responsiveness	✓	✗

Monitor Architecture

- Collector:
  - receives packets from NIC queue
  - puts a packet pointer into parser queues
    (does not copy packets!)
  - use a reference count to track

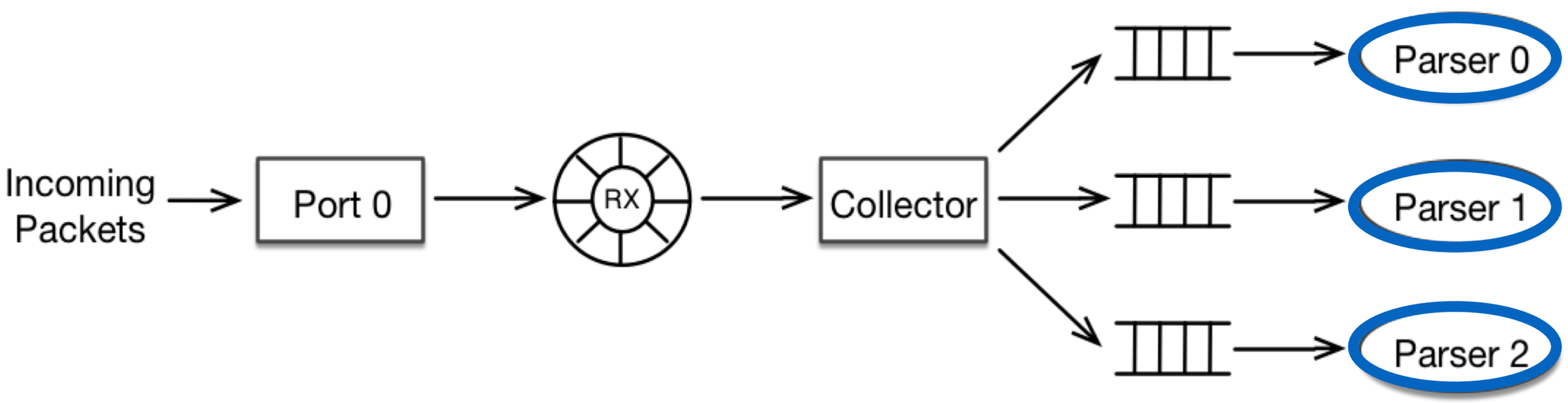


Monitor Architecture

- Parser:
  - contains the protocol logic to extract important packet data
  - protocolLib: implements common functions to analyze protocols

Example parsers:

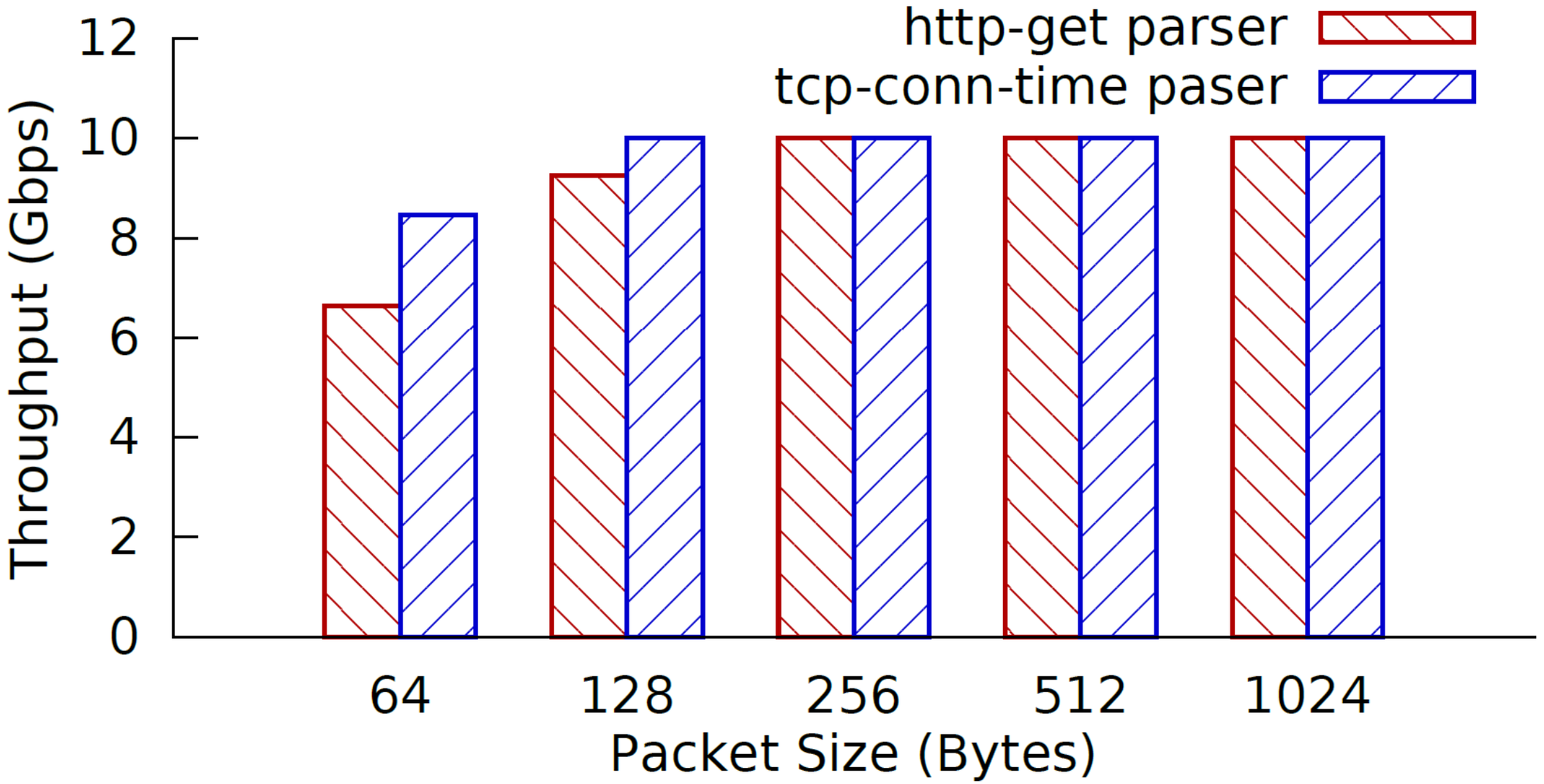
- tcp-connection-time
- tcp-flow-key
- tcp-pkt-size
- http-get
- memcached-get



Efficient Packet Parsing

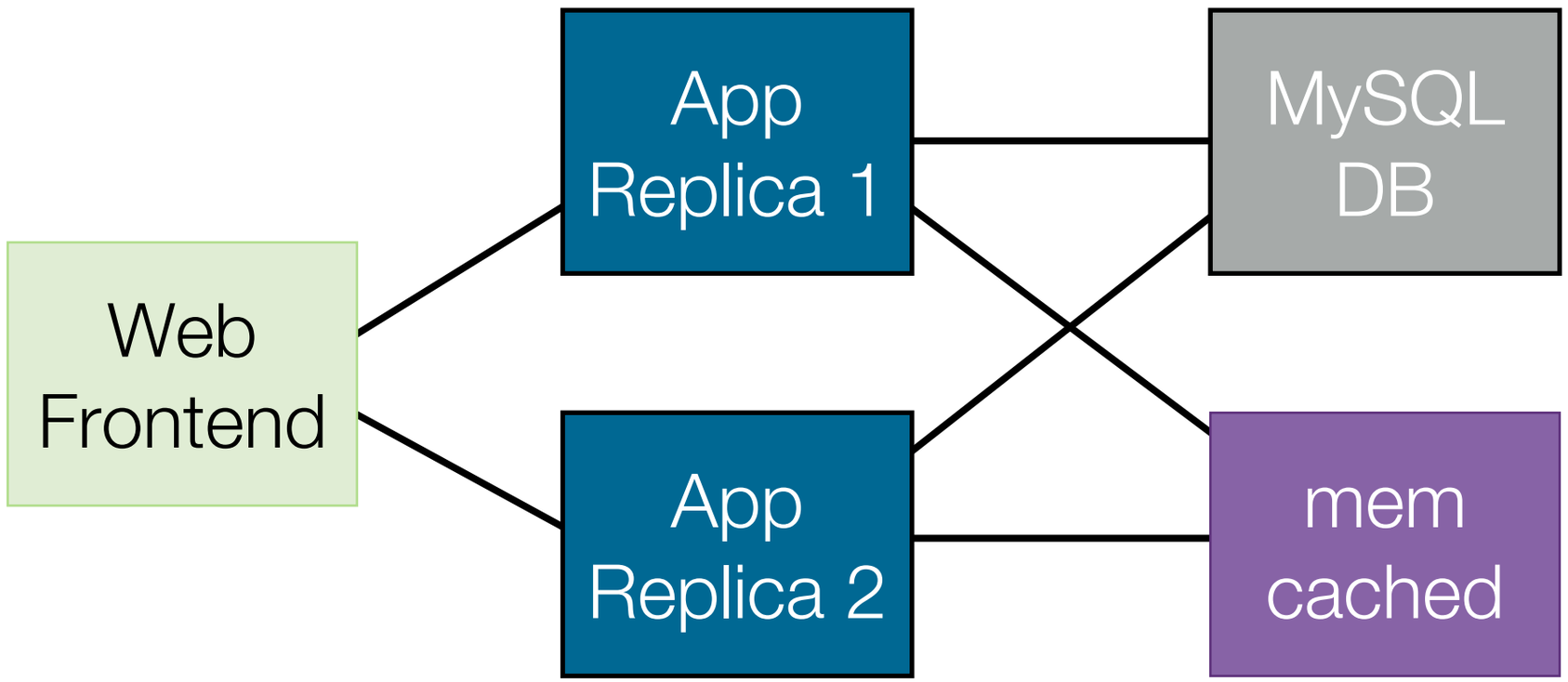
A single core Monitor can process packets at 10Gbps

- Trick: No kernel overheads, No interrupts



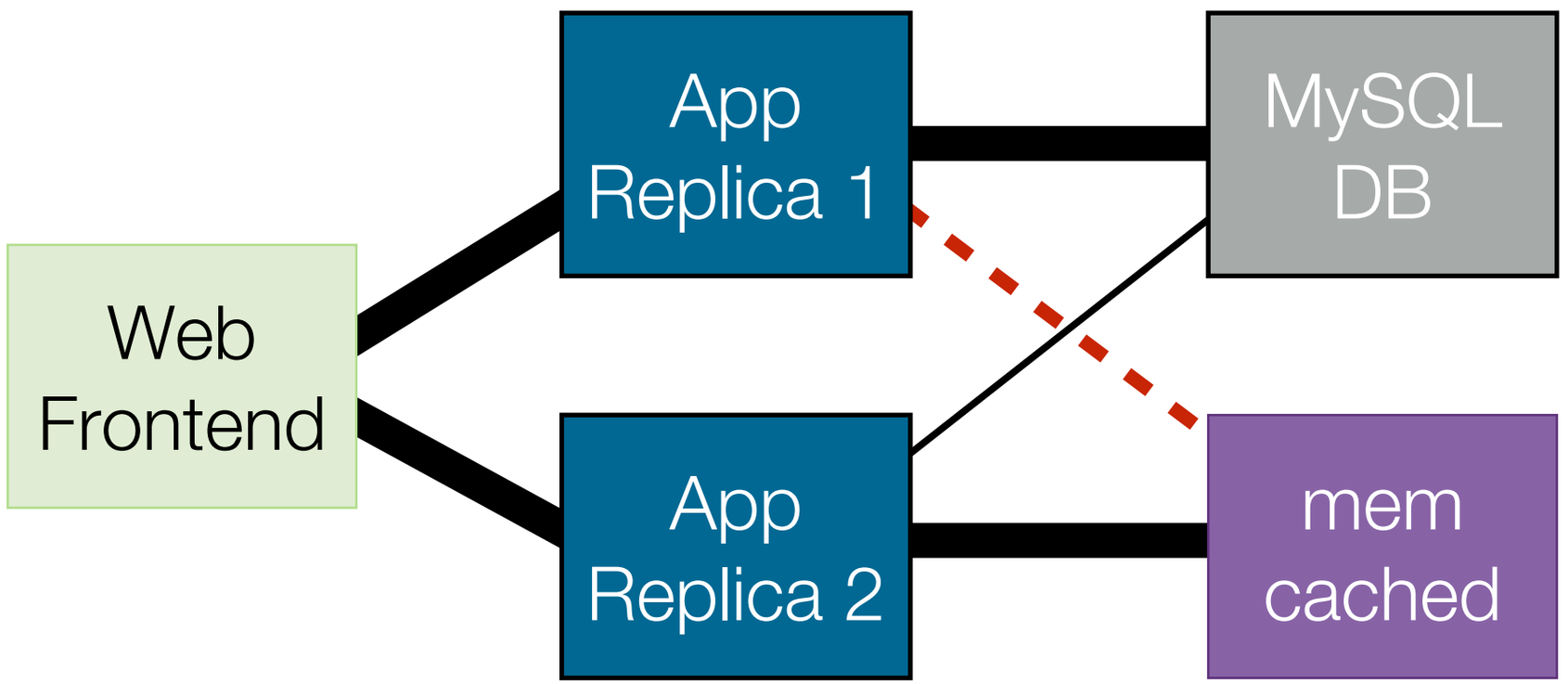
Web Performance Analysis

- Average response time is ~200ms. Should be < 20!
- Why??



Web Performance Analysis

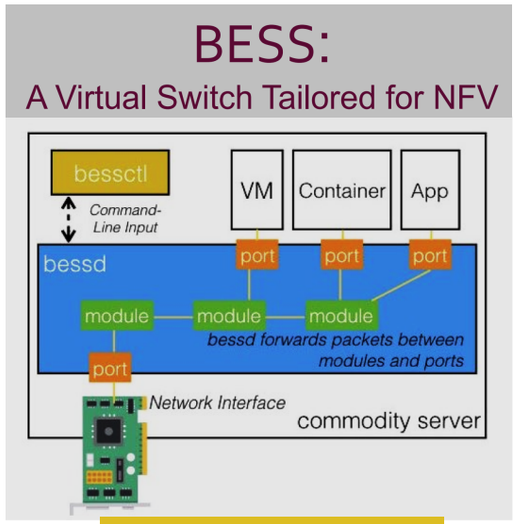
- Average response time is ~200ms. Should be < 20!
- Why??



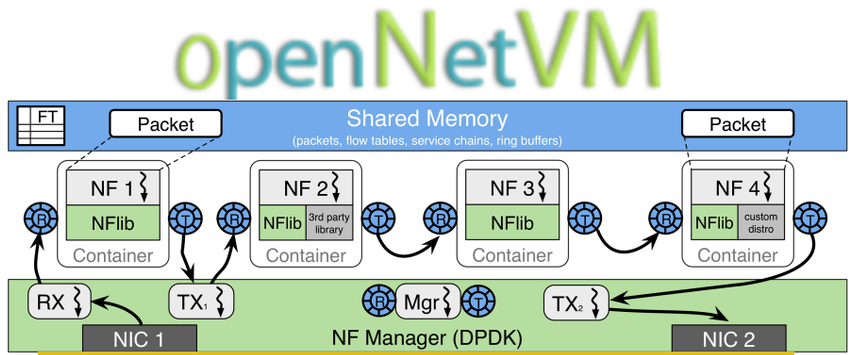
NetAlytics queries make it easy to find problems across a distributed application

Why Improve Existing NFV Frameworks?

- Existing NFV frameworks focus on L2/L3 processing



E2 [SOSP'15]



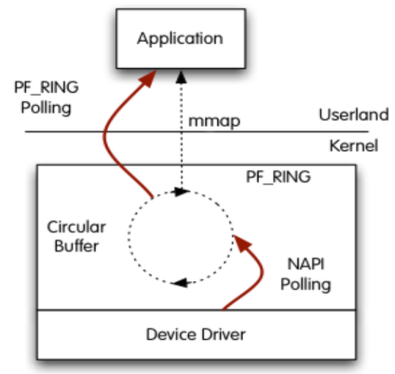
OpenNetVM [Hotmiddlebox'16]

ClickOS [NSDI'14]

netmap - the fast packet I/O framework

netmap [Usenix ATC '12]

PF_RING™



Vanilla PF_RING

PF_RING [SANE'04]

Networks are Changing

- Traditional Networks are hard to manage and evolve
  - Compute the configuration of each physical device
  - Operate without communication guarantees
  - Operate within given network-level protocol
- Root Cause: Control plane is tightly coupled with data plane