# G-NET: Effective GPU Sharing In NFV Systems

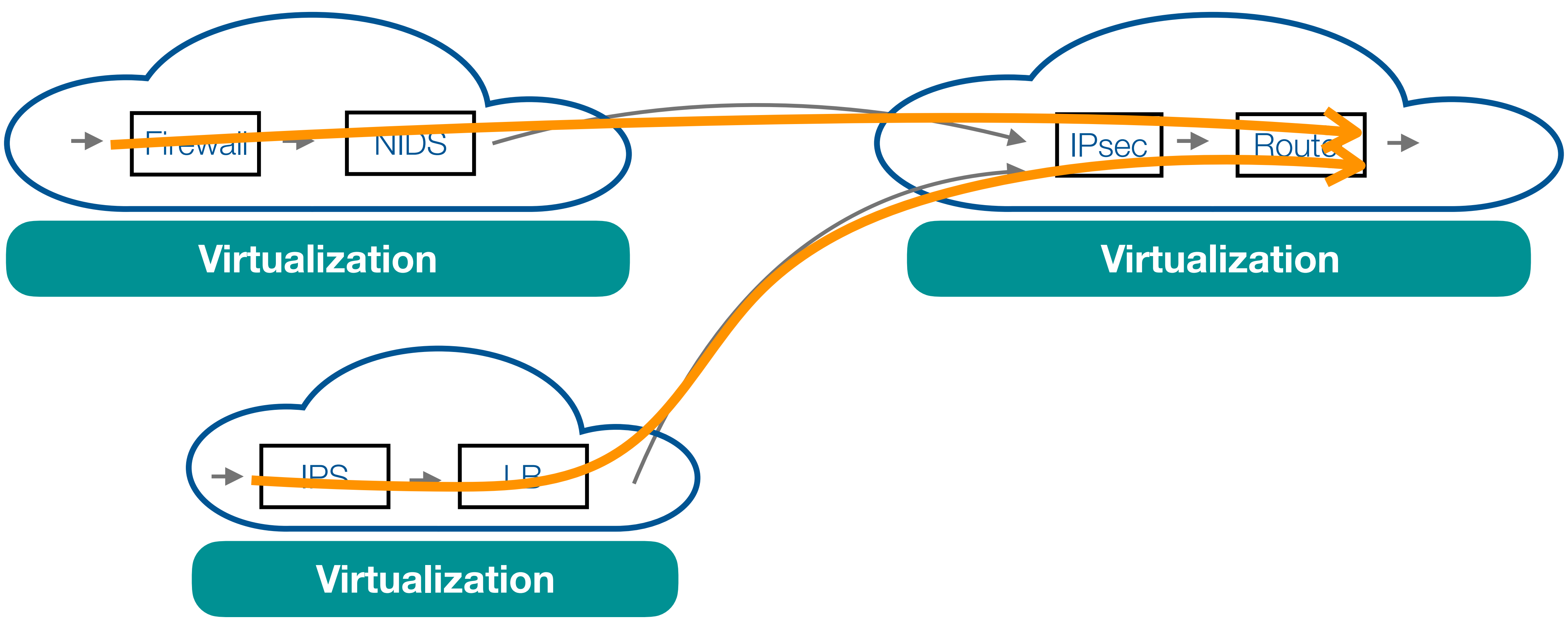
**Kai Zhang***, Bingsheng He^, Jiayu Hu#, Zeke Wang^,
Bei Hua#, Jiayi Meng#, Lishan Yang#

*Fudan University
^National University of Singapore
#University of Science and Technology of China

# Network Function Virtualization (NFV)

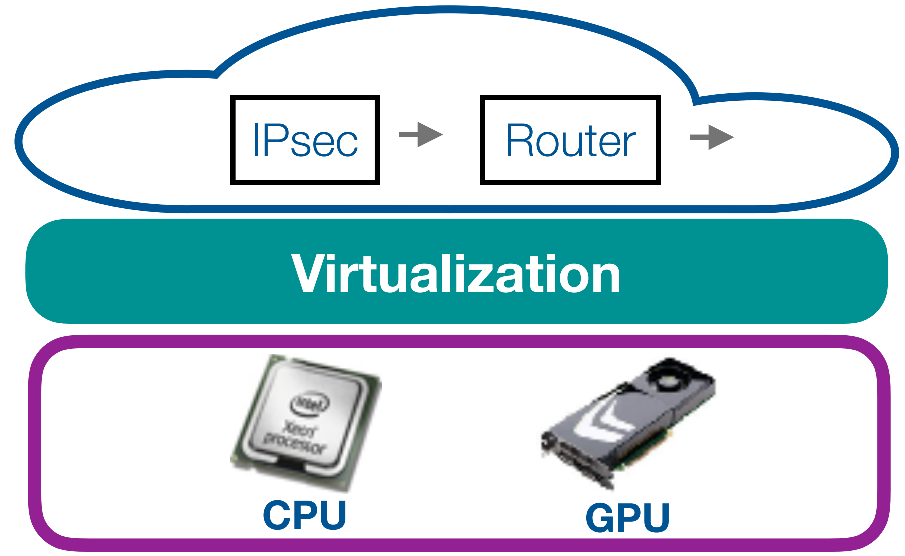
- **Network Functions**: nodes on the data path between a source host and a destination host
    - Firewall, NIDS, IPS, Gateway, VPNs, Load Balancers, etc.

- **NFV** is a network architecture concept: **hardware => software**
    - Based on **virtualization techniques**
    - **Easier to manage/deploy, higher flexibility, higher scalability, easier to validate,** etc.
    - Construct **service chains** to provide specific services to meet different demands

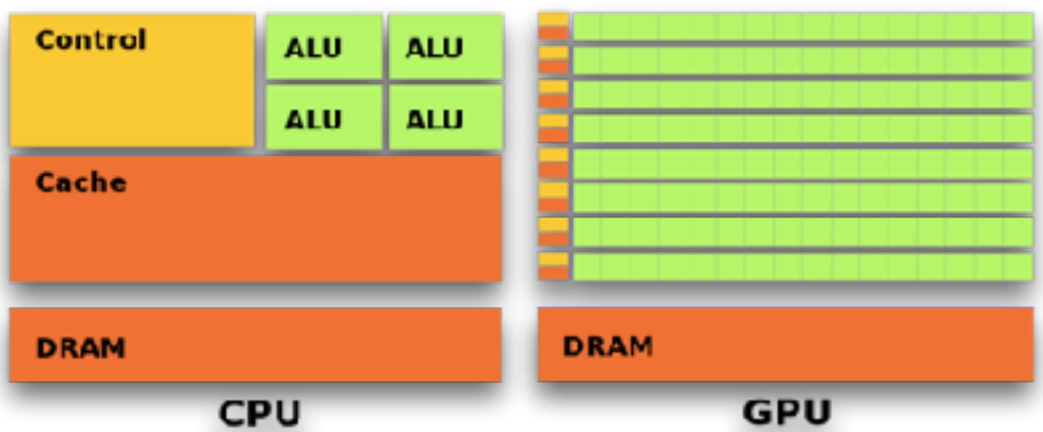


# GPUs in Accelerating Network Functions

- **GPUs** are proven to be a good candidate for accelerating network functions
  - **Router** - PacketShader [Sigcomm'10]
  - **SSL reverse proxy** - SSLShader [NSDI'11]
  - **NIDS** - Kargus [CCS'12], MIDeA [CCS'11]
  - **NDN Router** - [NSDI'13]



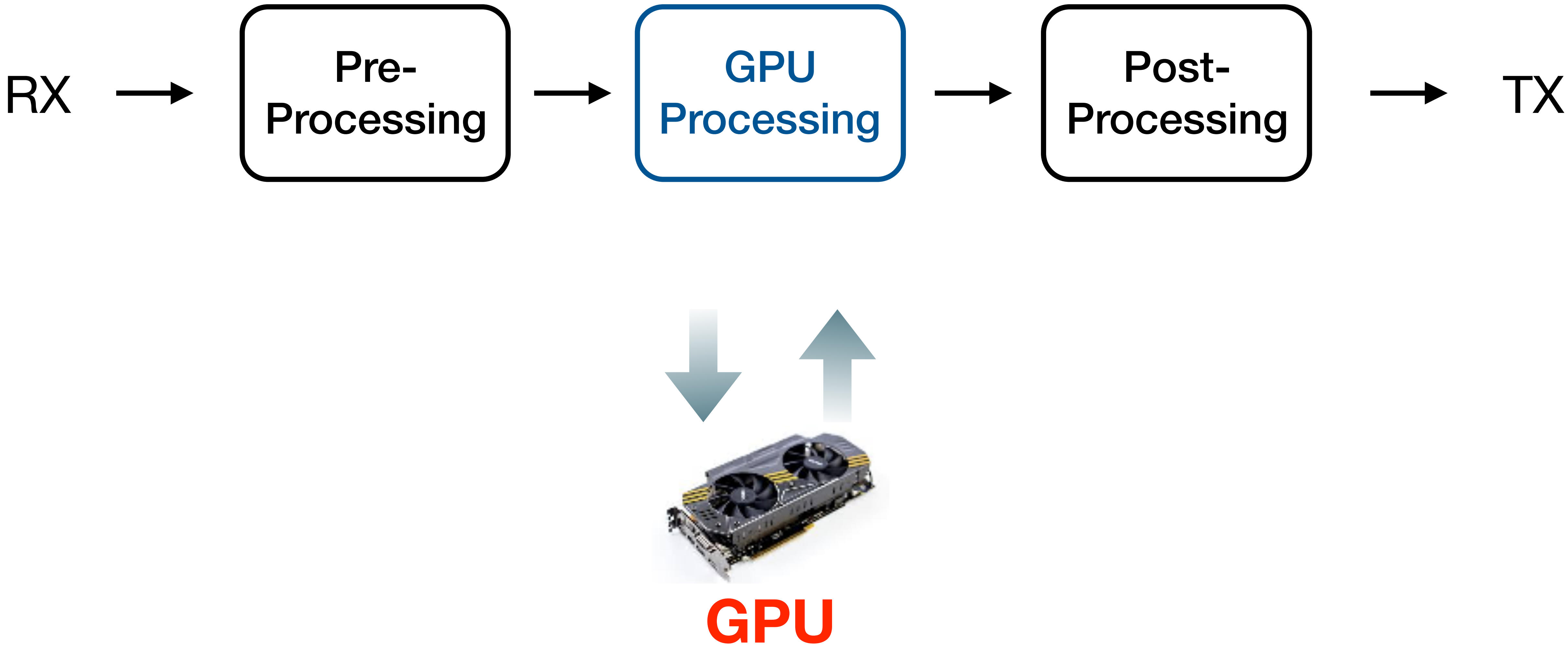

Intel Xeon E5-2697 v4: **18** Cores
**76.8** GB/s memory bandwidth
price: **$2702**



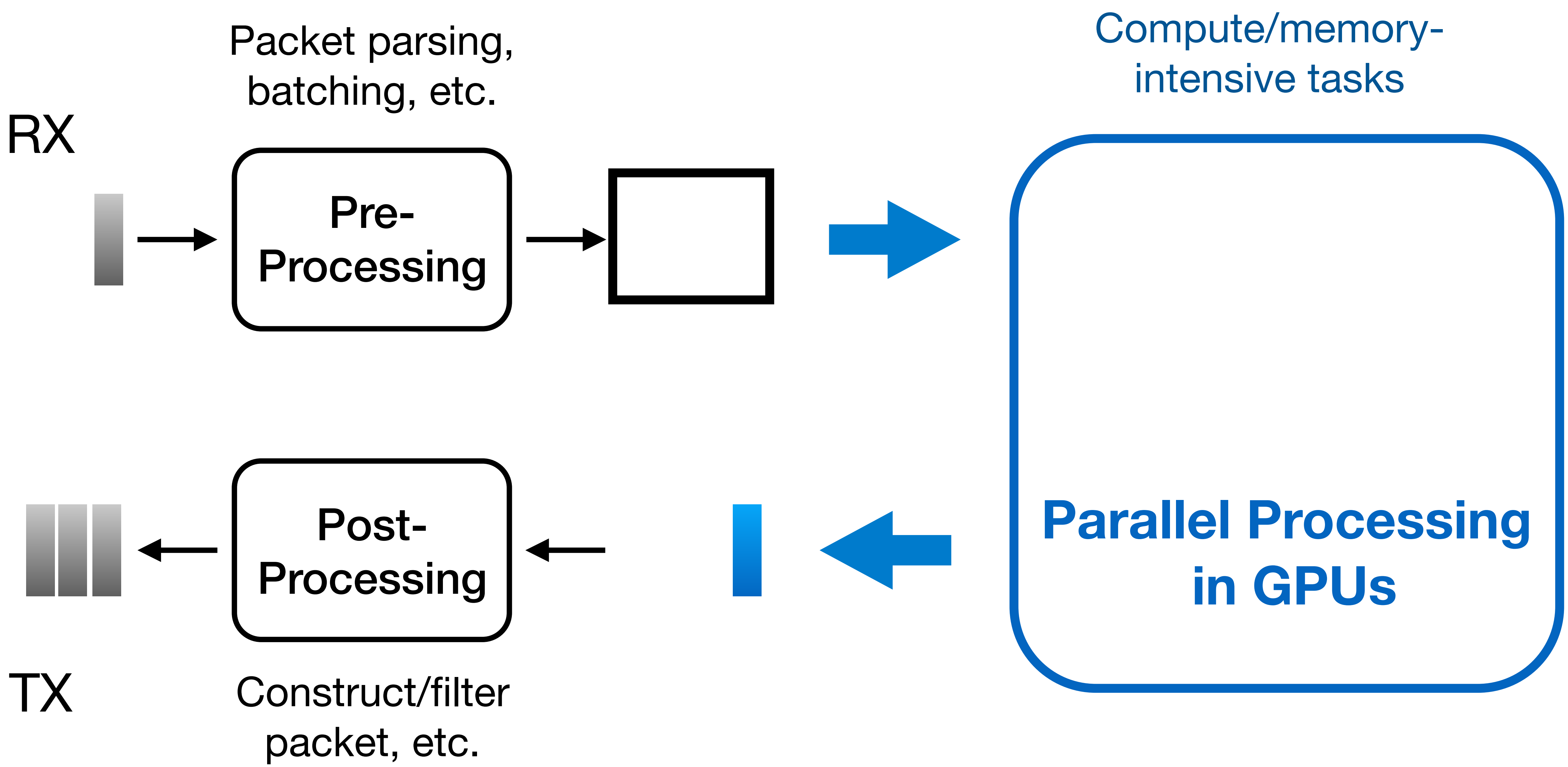

Nvidia Titan X: **3840** Cores
**550** GB/s memory bandwidth
price: **$1999**

1. **Massive Processing Units**
   - Network functions — large number of packets
   - GPU — thousands of cores for parallel processing
2. **Massively Hiding Memory Access Latency**
   - Network functions — large number of memory accesses in processing packets
   - GPUs can effectively hide memory access latency with massive hardware threads and zero-overhead thread scheduling (a GPU hardware support)

# GPU-Accelerated Network Functions

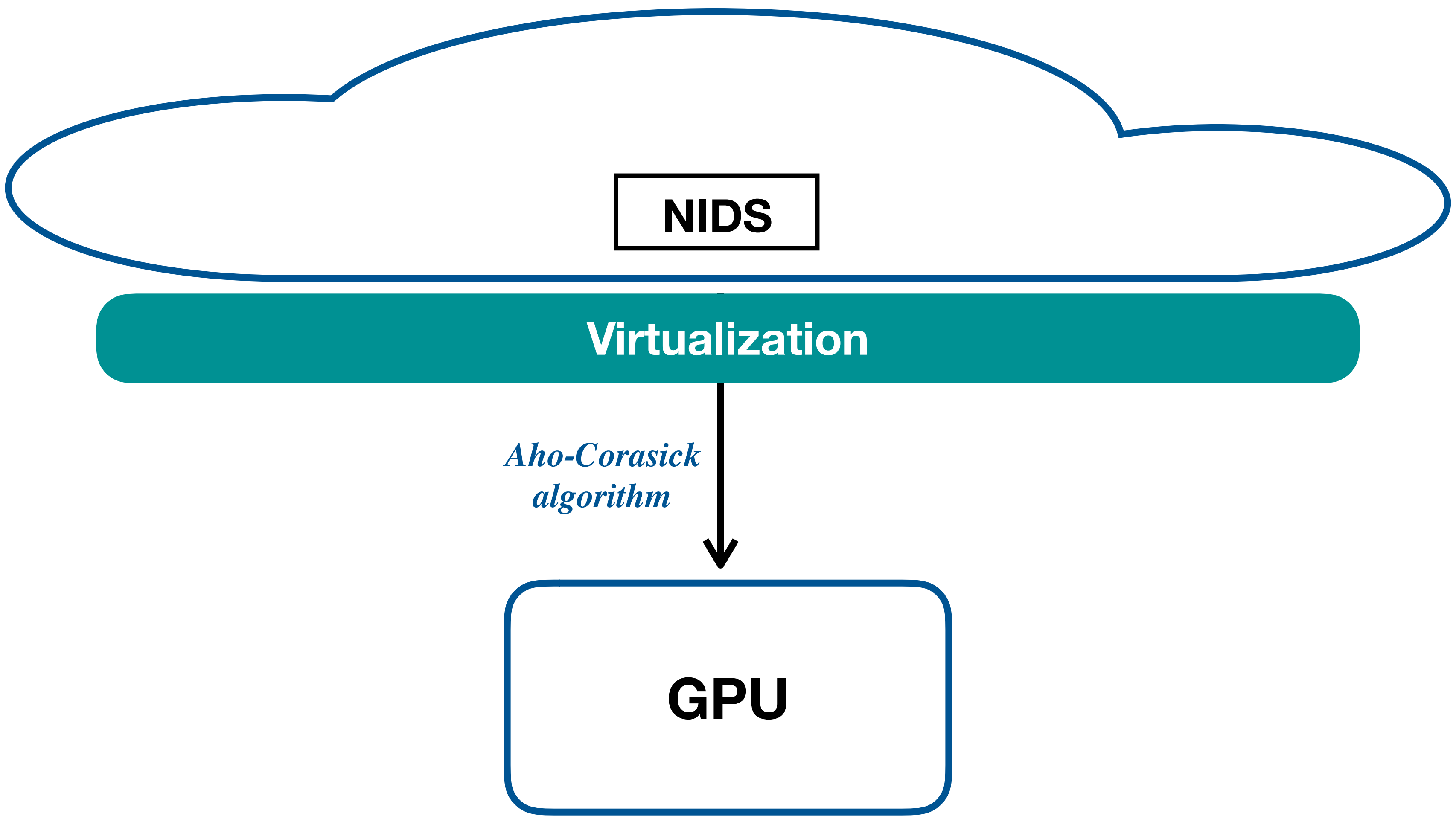


# GPU-Accelerated Network Functions



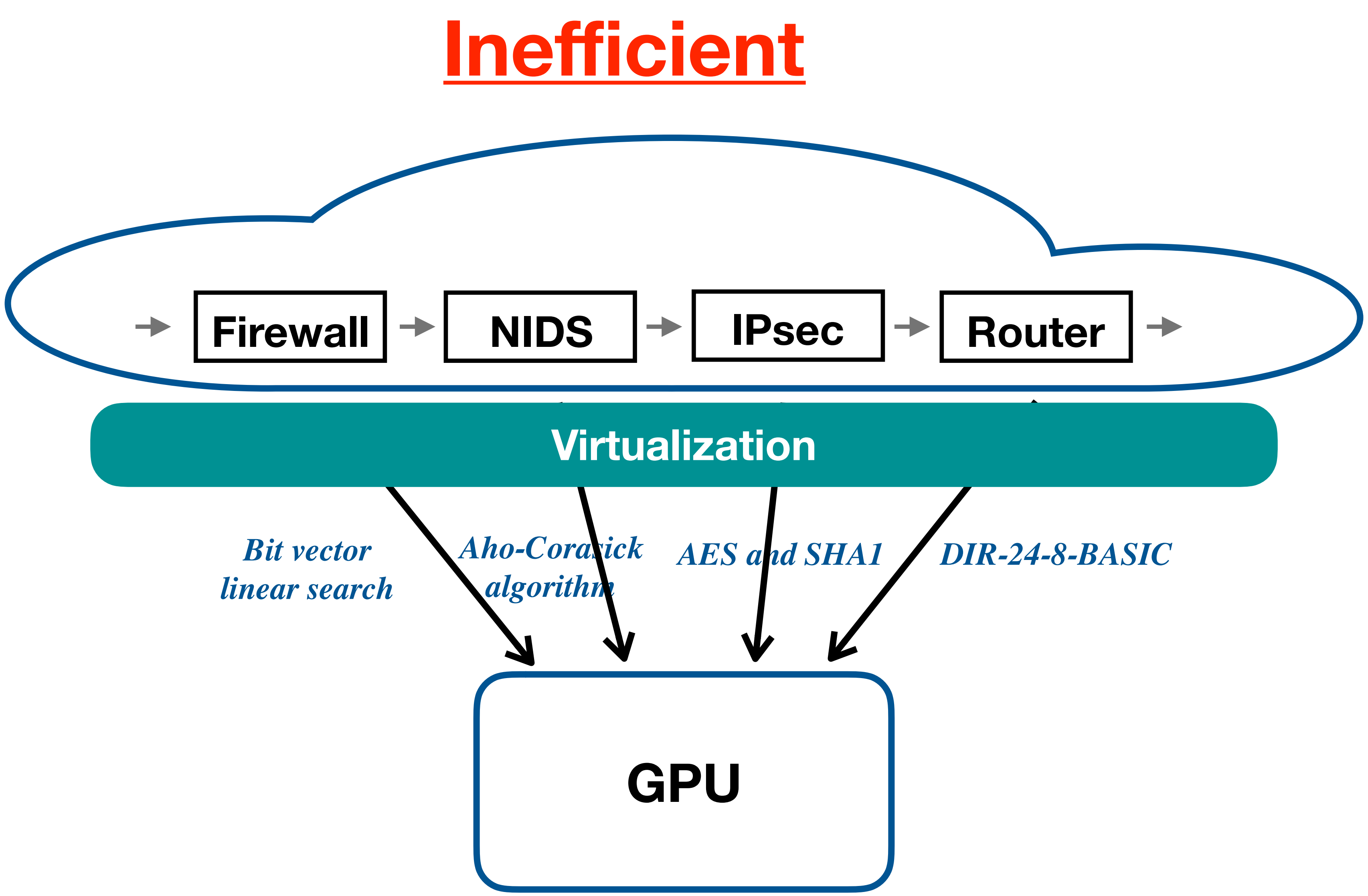
# Why GPUs Have not Been Utilized in NFV Systems?

- **Current GPU-based NFs** - ***Exclusive Access*** to GPU
    - The GPU is only accessed by **one network function**



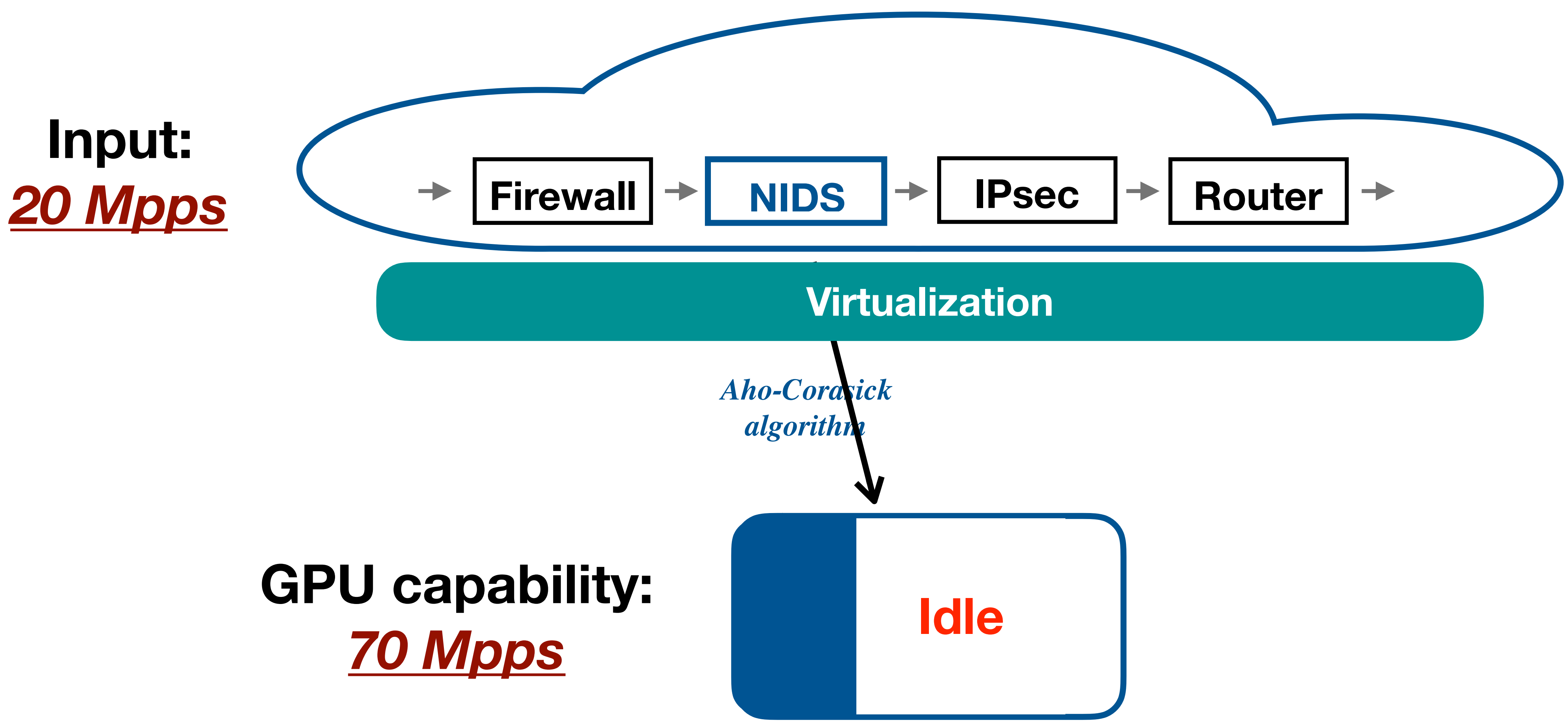
# Why GPUs Have not Been Utilized in NFV Systems?

- *Temporal GPU Sharing -* **Only kernels** from **one VM** can run on the GPU at a time



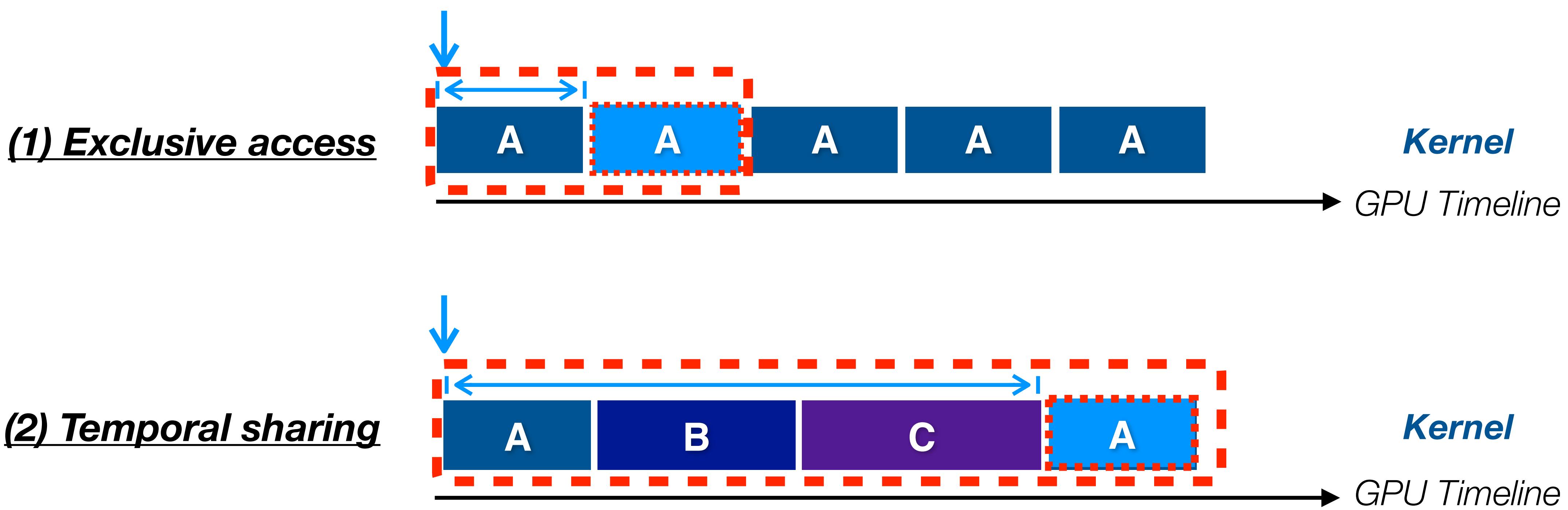
# Current Way of GPU Virtualization is Inefficient

- ***Temporal GPU Sharing* - Only kernels** from **one VM** can run on the GPU at a time
  - ***GPU underutilization***

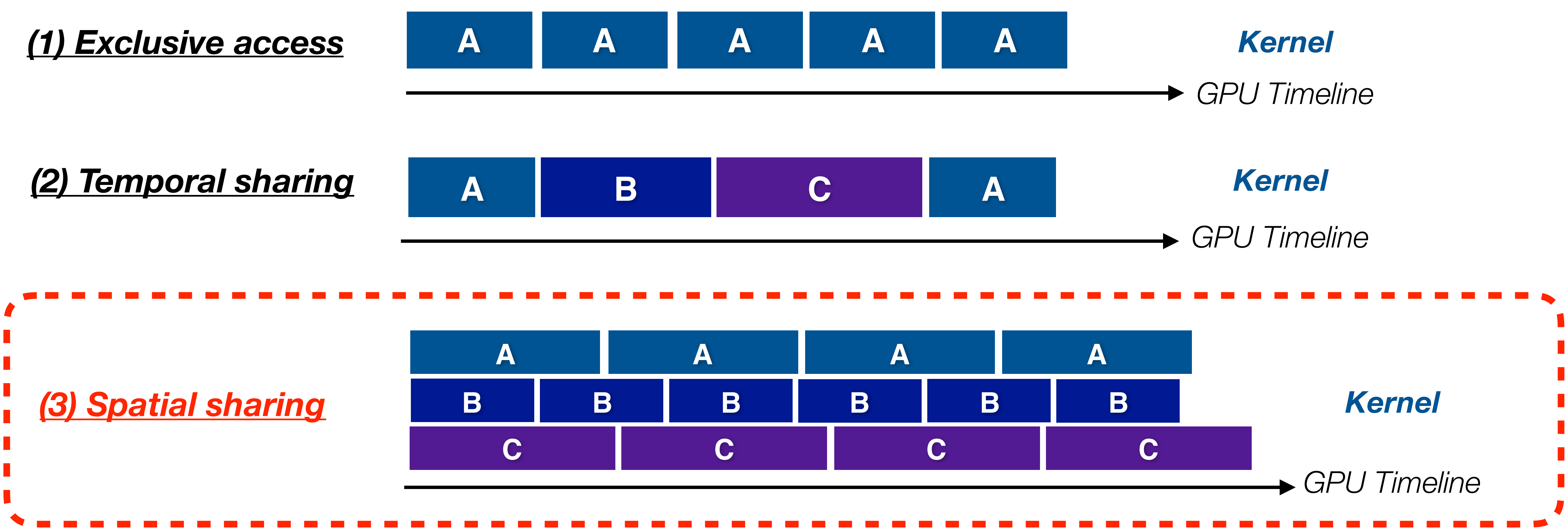


# Current Way of GPU Virtualization is Inefficient

- ***Temporal GPU Sharing*** **-** **Only kernels** from **one VM** can run on the GPU at a time
  - ***GPU underutilization***
  - ***Higher latency***



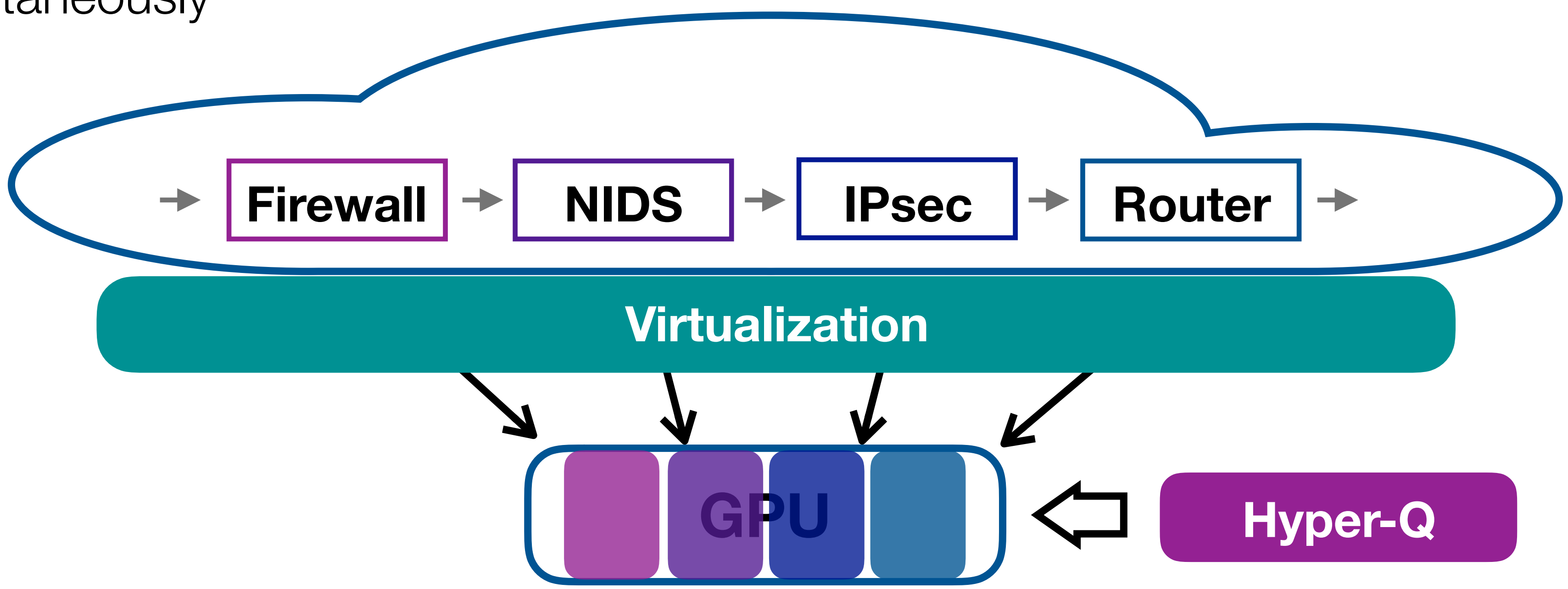
# Spatial GPU Sharing




- **Spatial GPU sharing** — multiple kernels run on the GPU **simultaneously**
  - **Minimize** the **interference** of kernel executions from other NFs (Latency)
  - **Enhance utilization** - Kernels from VMs can run on the GPU simultaneously (Throughput)

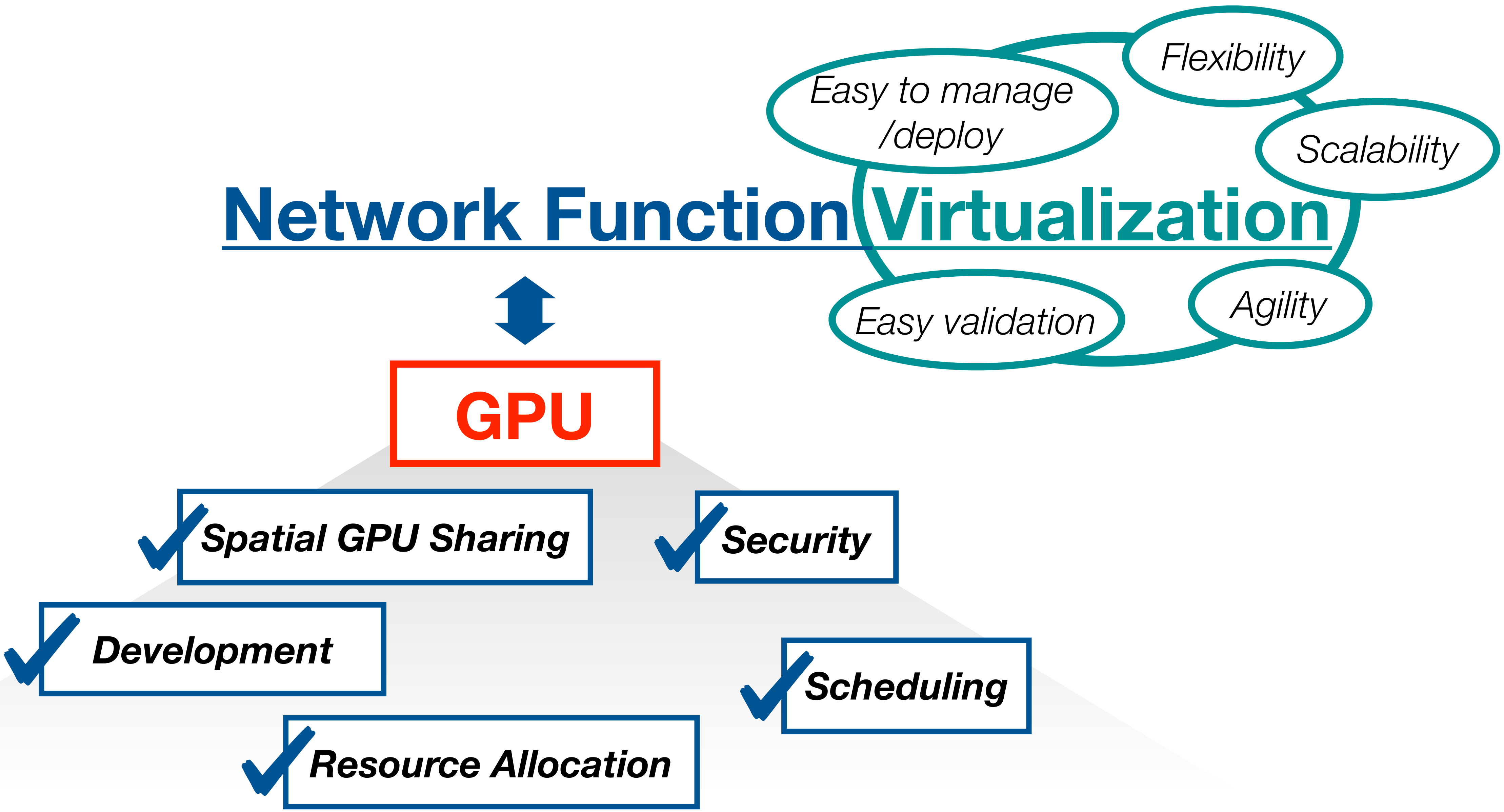
# Hyper-Q for Spatial GPU Sharing

- **Hyper-Q** for spatial GPU sharing
  - A technique that enables GPU kernels from the **same GPU context** to execute on the GPU simultaneously



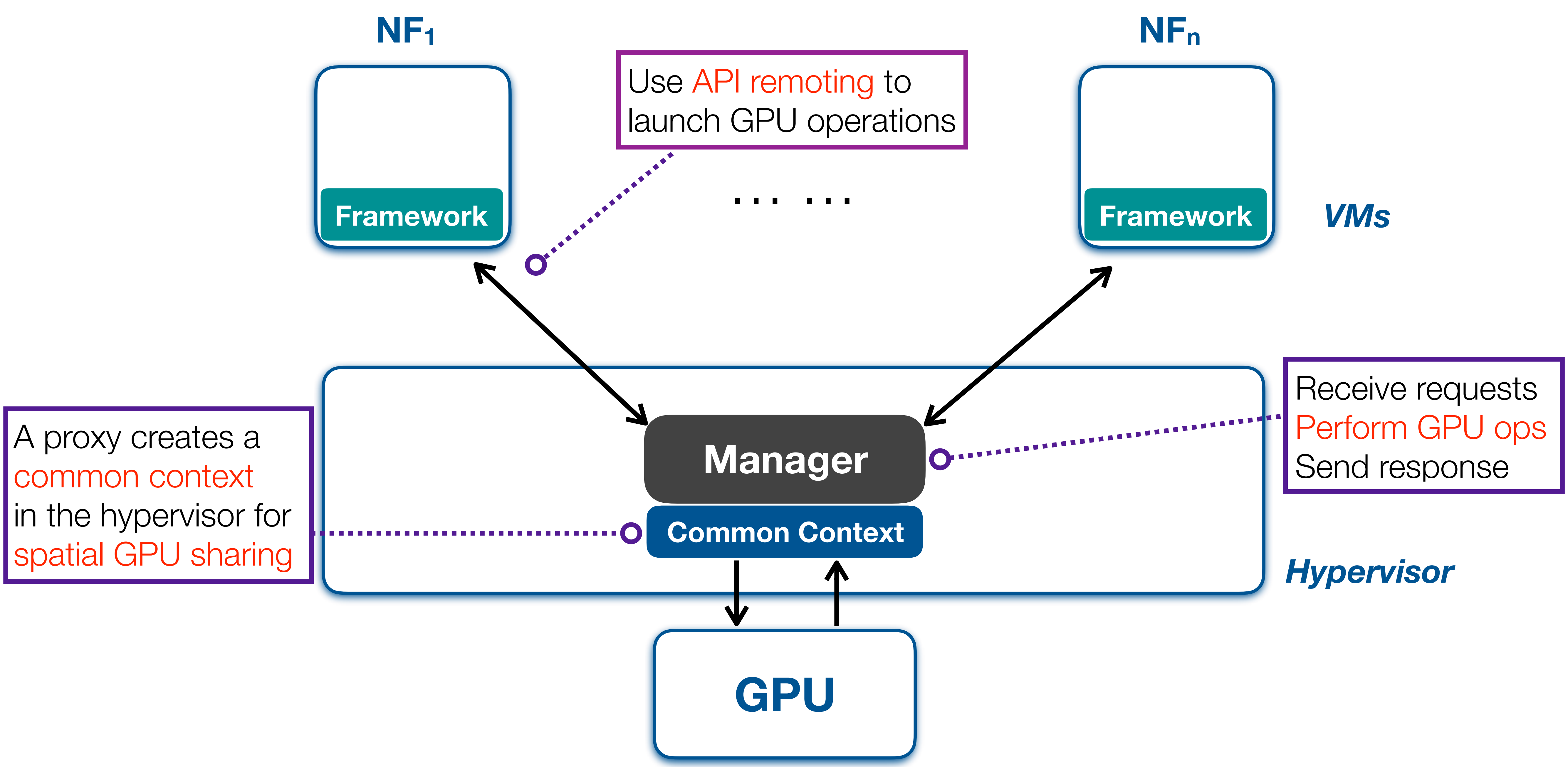

- **Challenges**
  1. VMs have **different GPU context** **=>** **Cannot** utilize Hyper-Q directly
  2. Kernels utilizing Hyper-Q can **access the entire memory space** **=>** **Security issue**
  3. NFs are **not aware of** the existence of other NFs; An NF tries to maximize its resources would influence other NFs **=>** Demanding **scheduling** and **resource allocation** schemes

# The Goal of G-NET

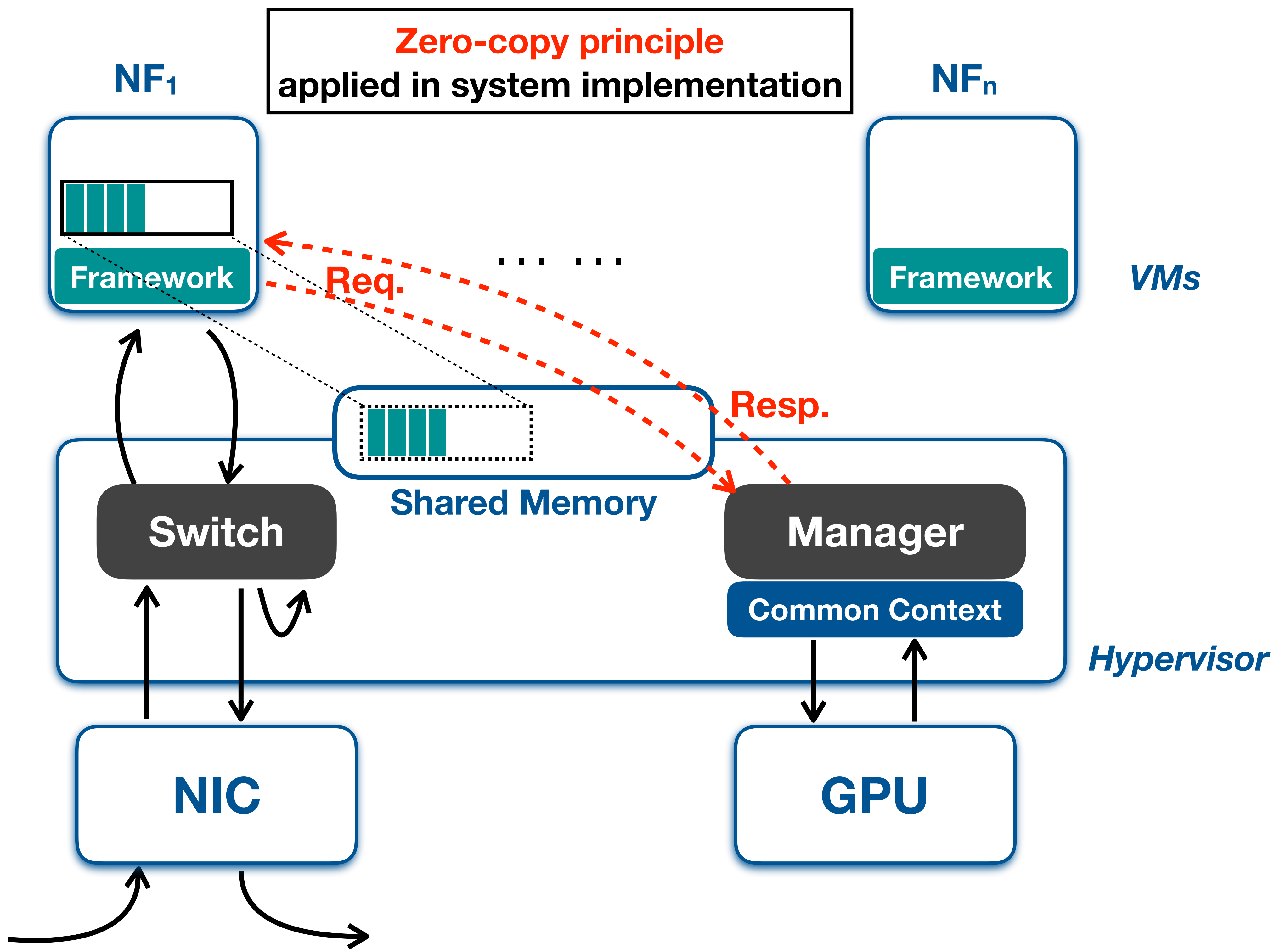



**G-NET: *NF-Hypervisor Co-Design***

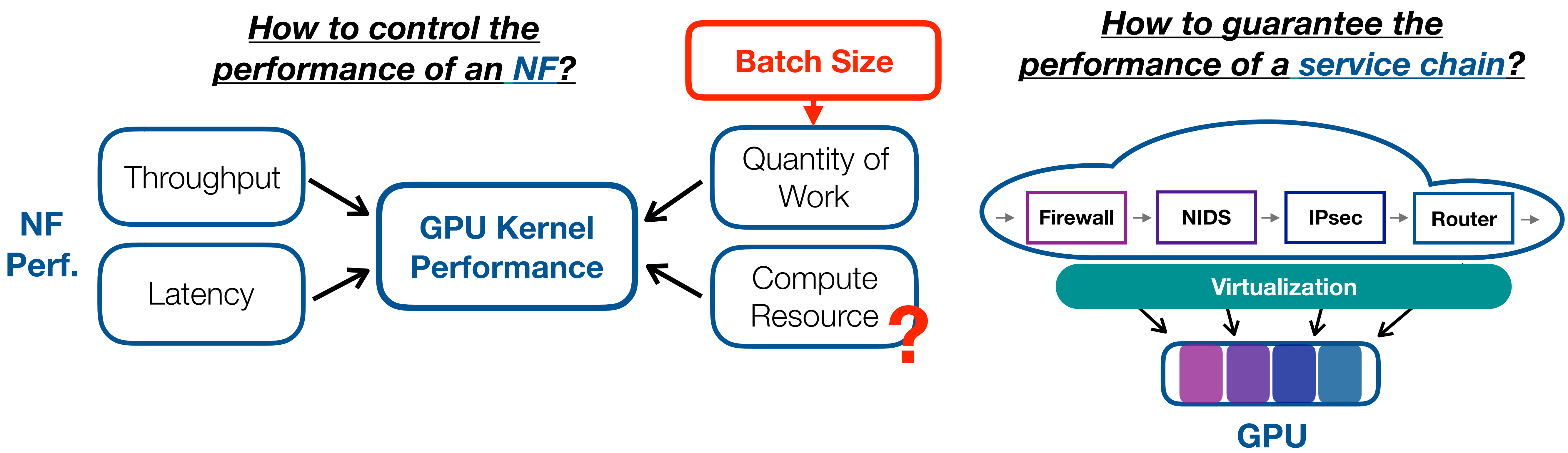
# G-NET: GPU Virtualization



# G-NET: System Workflow



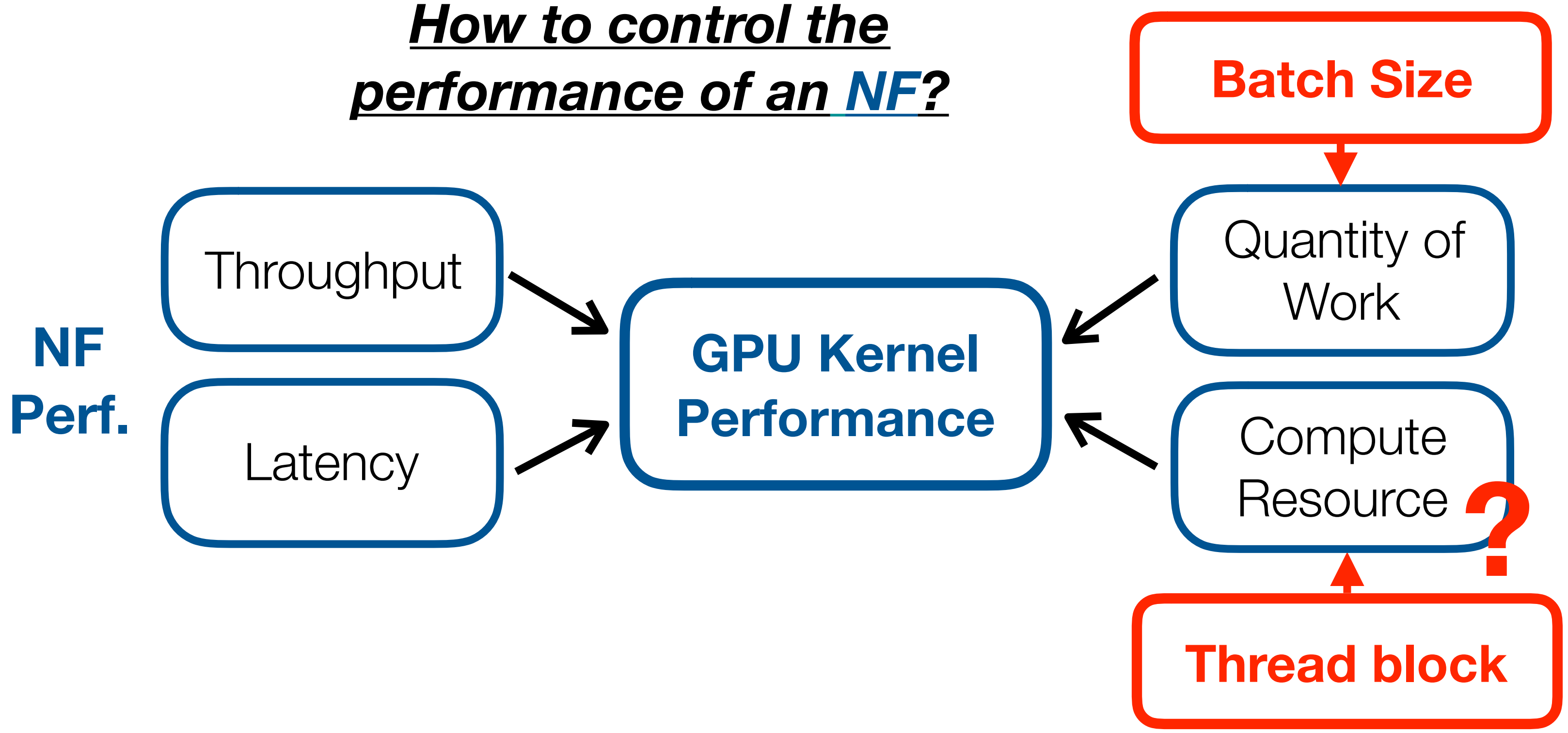
# Achieve Predictable Performance



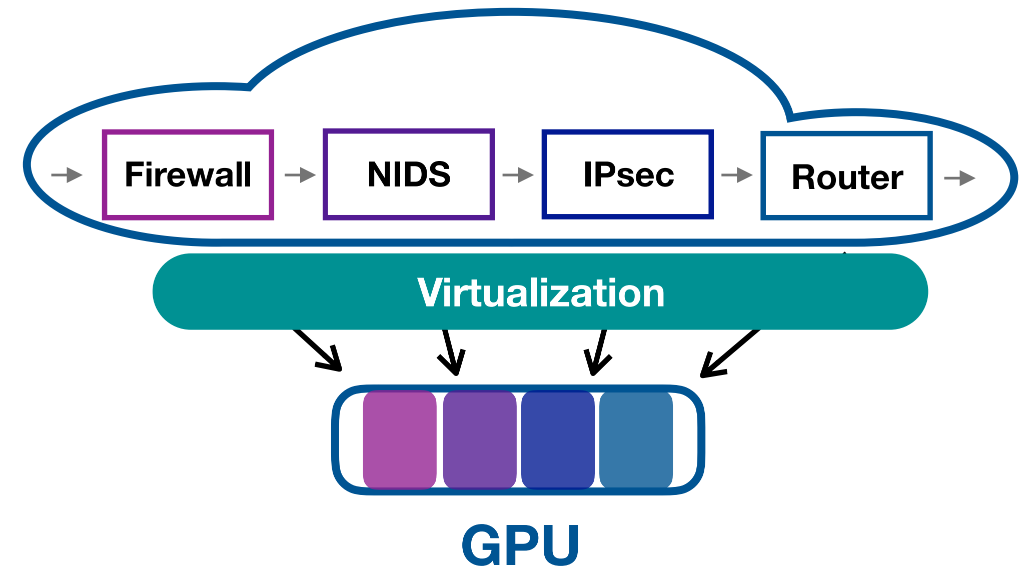

- **How to allocate GPU resources?**
  - GPUs utilize fast **thread switching** to enhance hardware utilization
  - GPUs have **massive hardware threads** (**#thread >> #core**)
  - Unlike CPUs, a GPU thread is **unable** to be bond to a specific core

# Achieve Predictable Performance

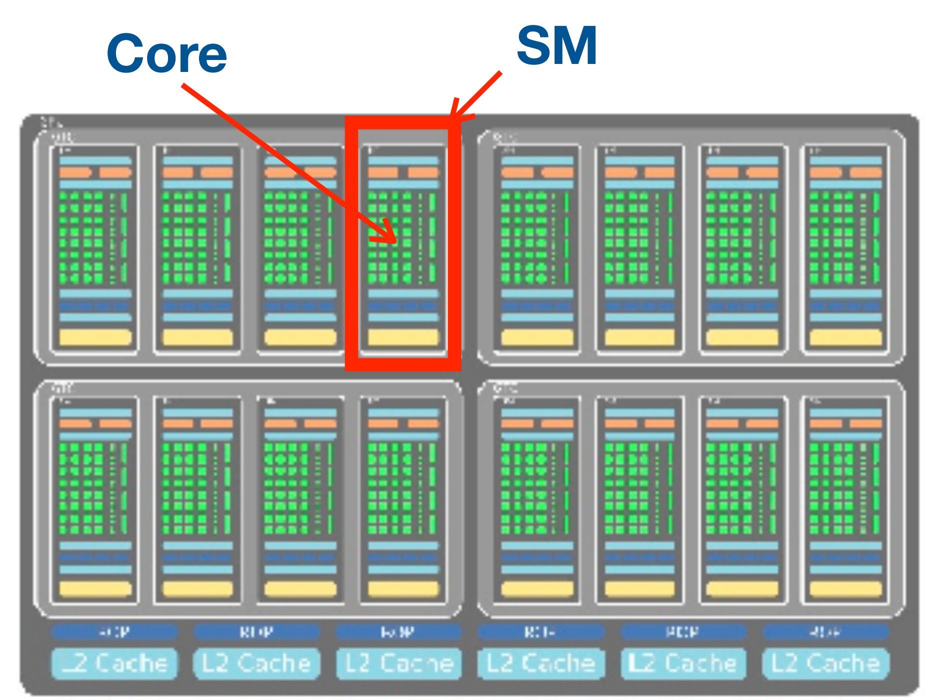
***How to control the performance of an NF?***




***How to guarantee the performance of a service chain?***




- Our Approach
  - **Streaming Multiprocessor (SM)** as the basic unit for resource allocation
  - One thread block can only run on one SM;
    A thread block would be scheduled to run on an idle SM when there are available ones
  - A thread block is allocated with one SM when
    ***Total #thread blocks <= Total #SMs***



# Service Chain Based Scheduling

- How to optimize the performance of a service chain with limited compute resources
  - NFs have **different** processing tasks

| | Firewall | NIDS | IPsec | Router | → 3 |
|---|---|---|---|---|---|
| #SM | 5 | 5 | 5 | 5 | ~~Fairness~~ |
| Throughput (Mpps) | 6 | 3 | 4 | 10 | |

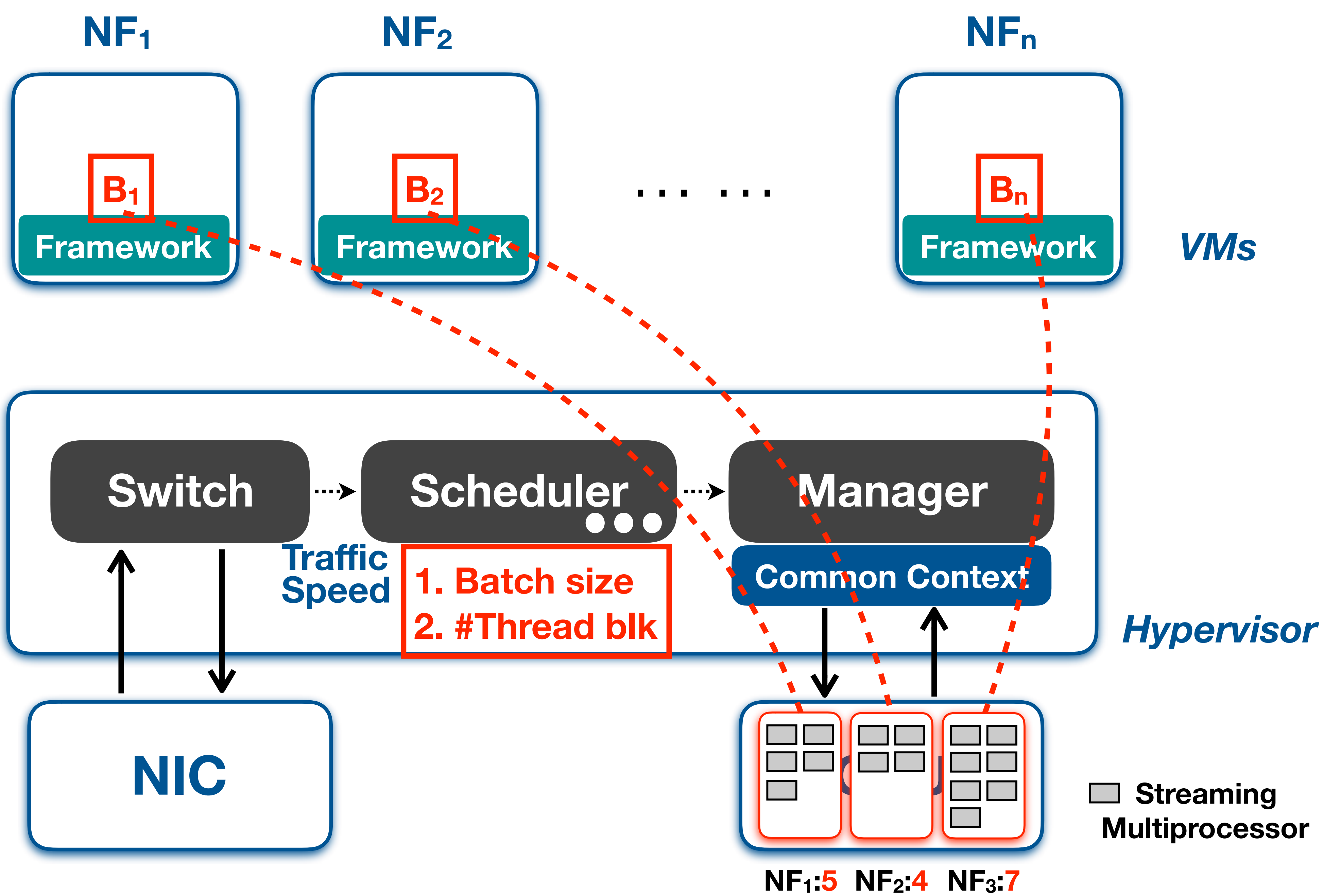
- **Service chain based** scheduling and resource allocation
  - Locate the **bottleneck NF** (the NF with the lowest throughput **T**)
  - Allocate **resources** for all NFs in the **service chain** to achieve throughput **T * (1+P)** *(0<P<1)*

→ Firewall → NIDS → IPsec → Router →

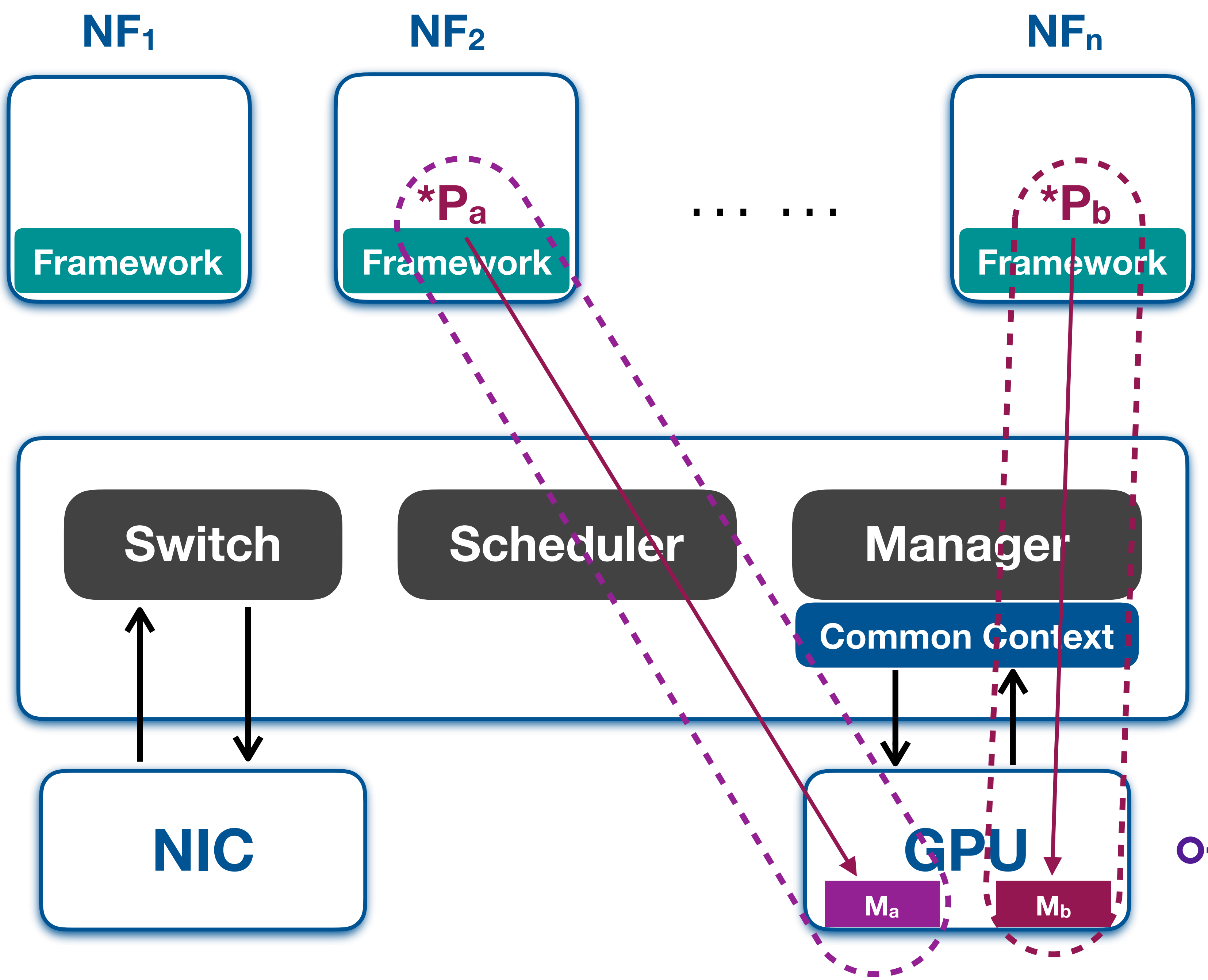
**Throughput improves by P in each round**

SMs

# Service Chain Based Scheduling



# IsoPointer for GPU Memory Isolation



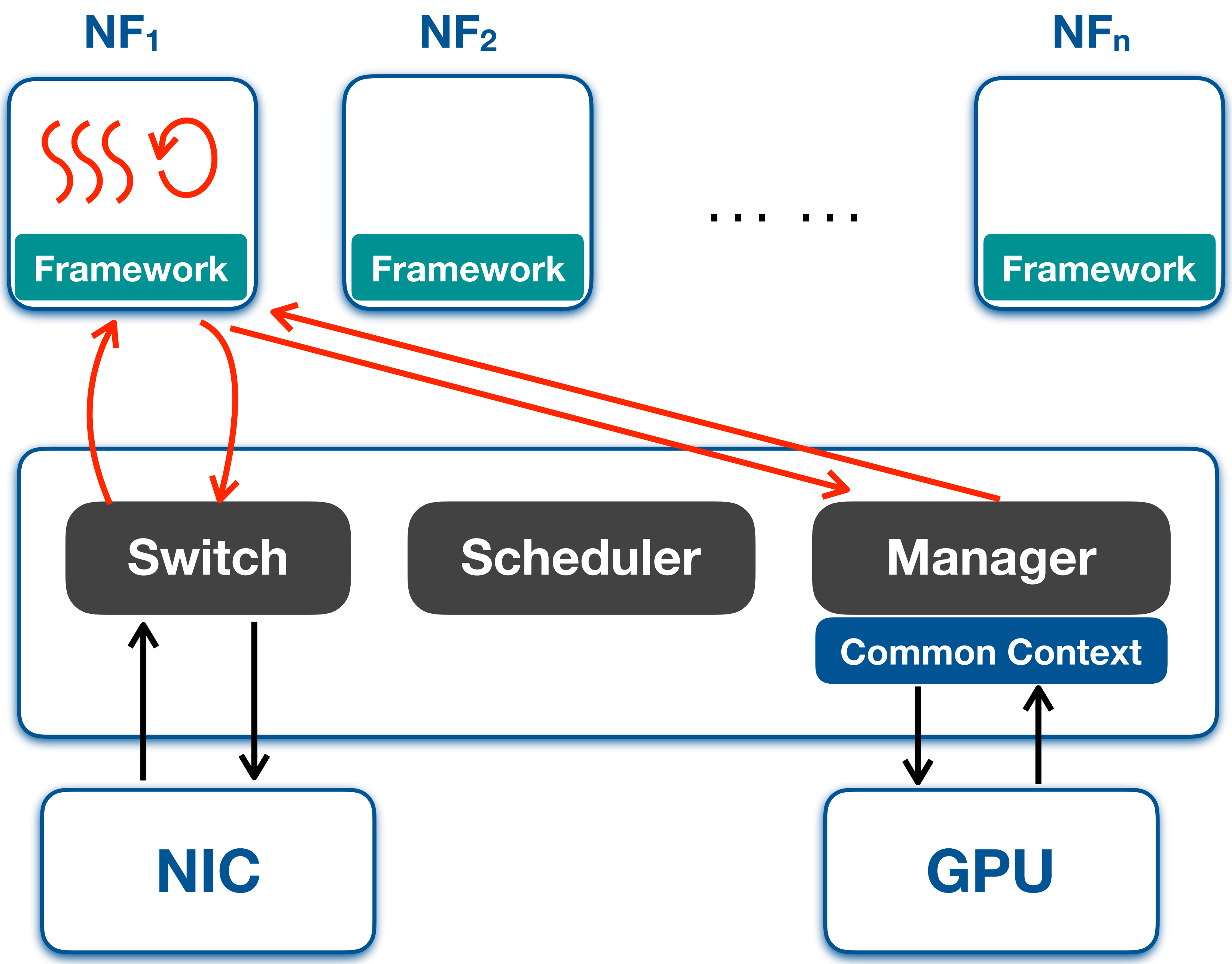

*P —> Memory Region

- Base Address (B)
- Memory Size (S)

Pointer access checking:
**B <= P < B+S**

IsoPointer: guarantee GPU memory isolation

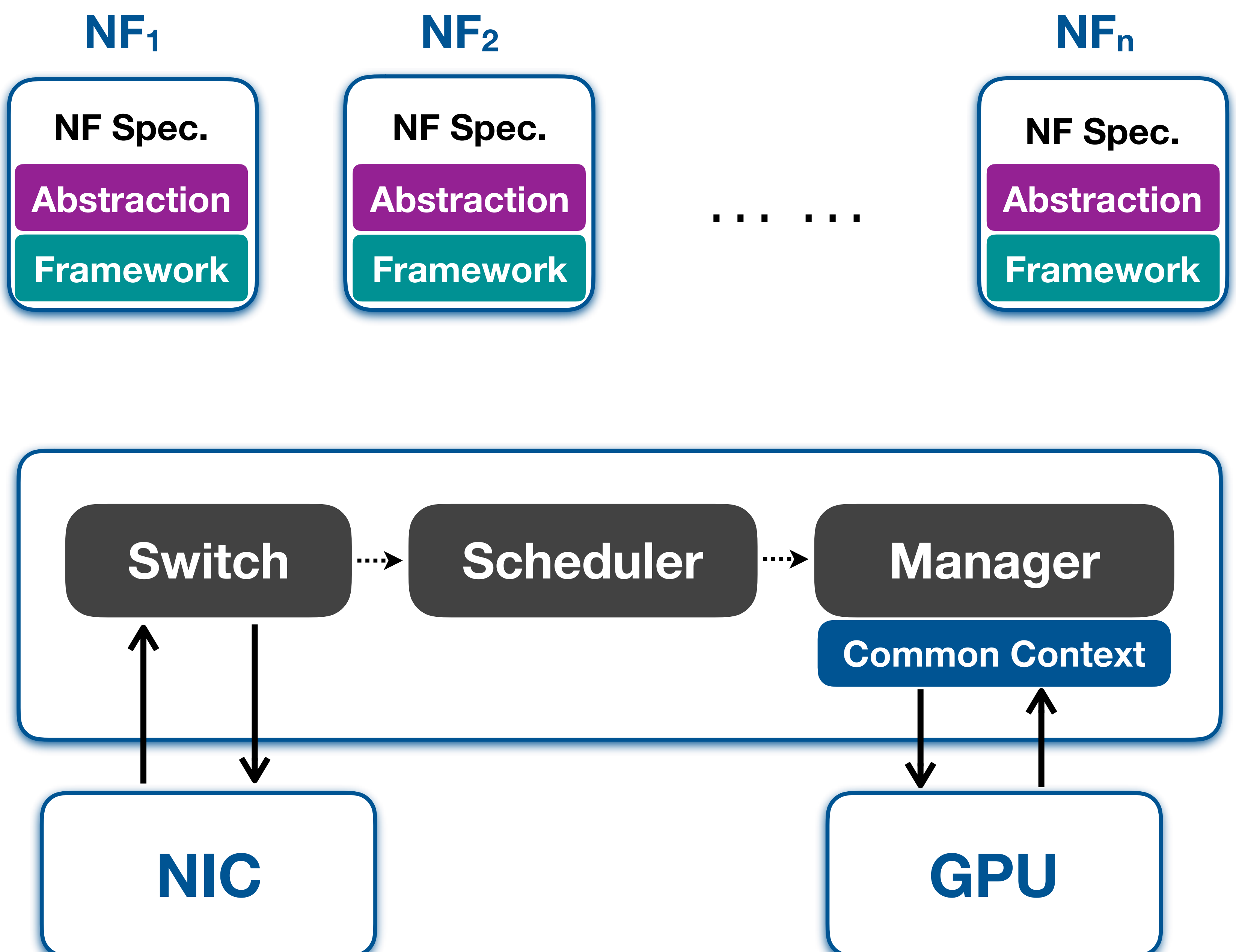
# NF Development




- **Repetitive development efforts**
  - CPU-GPU pipeline
  - Manage CPU threads
  - Communicate with Manager
  - Packet I/O with Switch
  - … …

- **Framework** handles all common operations

# NF Development




- **Repetitive development efforts**
  - CPU-GPU pipeline
  - Manage CPU threads
  - Communicate with Manager
  - Packet I/O with Switch
  - … …

- **Framework** handles all common operations
- **Abstraction** to simplify NF development

# NF Development

**NF₁**

NF Spec.

Abstraction

Framework

| Pre-Processing | GPU Processing | Post-Processing |
|---|---|---|
| • *pre_pkt_handler* | • *memcpy_htod*<br>• *set_kernel_args*<br>• *memcpy_dtoh* | • *post_pkt_handler* |
| called for each pkt | called for each kernel | called for each pkt |

**CPU code (Router)**

```
struct my_batch {
    uint64_t job_num;
    isoPtr<uint32_t> host_ip;
    isoPtr<uint32_t> dev_ip;
    isoPtr<uint8_t> host_port;
    isoPtr<uint8_t> dev_port;
}
void kernel_init(void) {
    gInstallKernel("/pathto/router.cu", "iplookup");
    build_routing_table();
}
...
    gInstallArg(batch->dev_ip);
    gInstallArg(batch->dev_port);
    gInstallArg(batch->job_num);
    gInstallArg(dev_lookup_table);
}
void memcpy_dtoh(batch) {
    gMemcpyDtoH(batch->host_port, batch->dev_port,
        batch->job_num * PORT_SIZE);
}
void post_pkt_handler(batch, pkt, pkt_idx) {
    pkt->port = batch->host_port[pkt_idx];
}
```

**Implementation = Significantly reduces development efforts + GPU Kernel**

Core functions

# Evaluation

- Hardware

**CPU**: Intel Xeon E5-2650 v4

**GPU**: NVIDIA GTX TITAN X

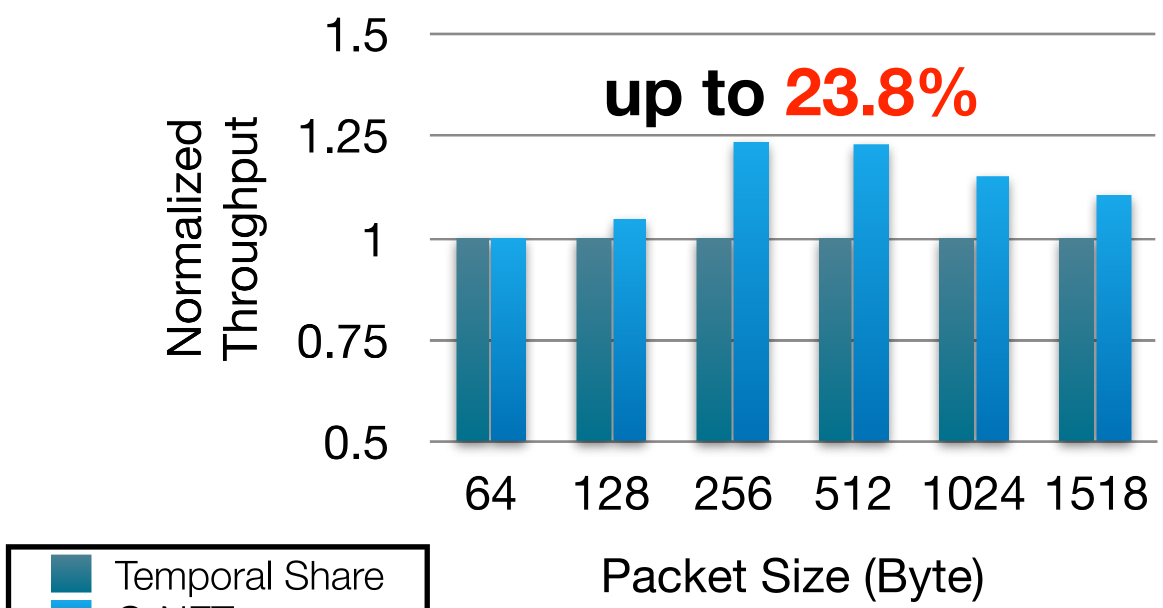
**NIC**: Intel XL710 40Gbps

- Software
  - **Virtualization**: Docker 17.03.0-ce
  - **NIC Driver & Library**: DPDK 17.02.1
  - **OS**: CentOS 7.3.1611, Linux kernel 3.8.0-30

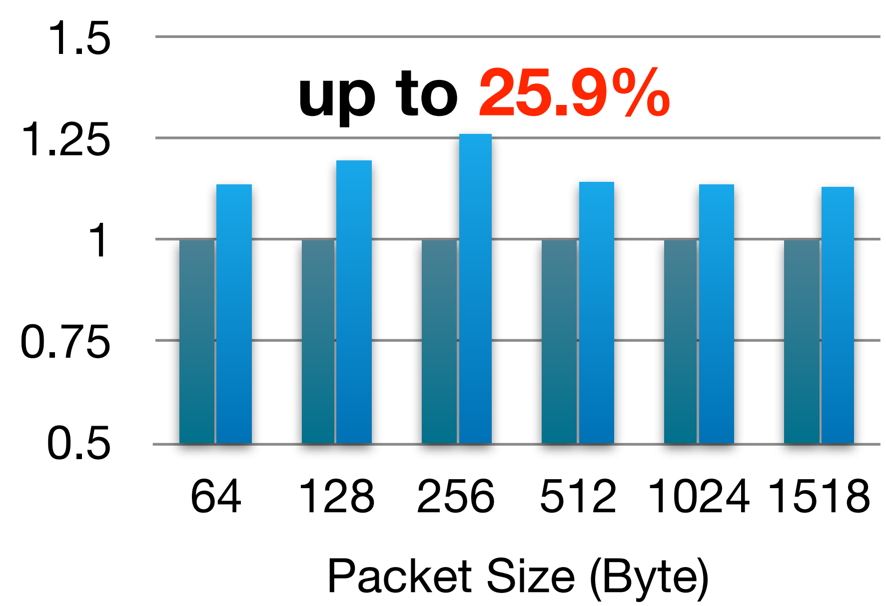
- Service Chains
  - **2 NFs**: IPsec → NIDS
  - **3 NFs**:
    - Firewall → IPsec → NIDS
    - IPsec → NIDS → Router
  - **4 NFs**: Firewall → IPsec → NIDS → Router

# Throughput

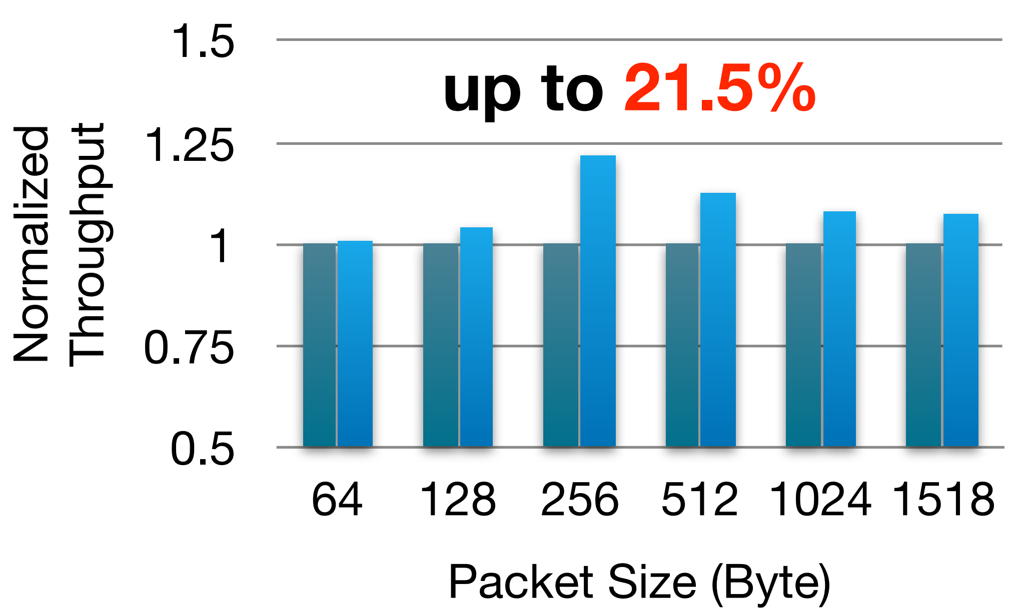
- Comparison with **Temporal GPU Sharing**



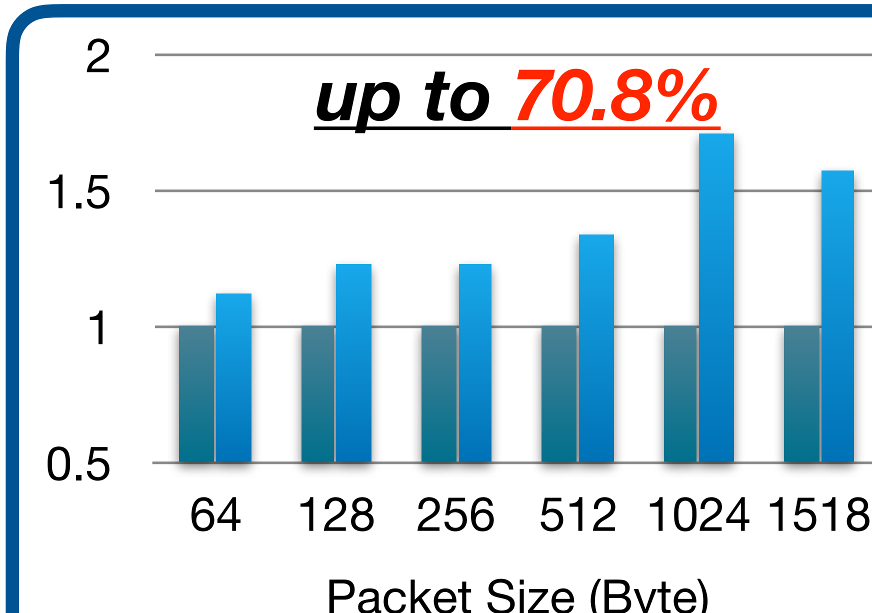

***(a) IPSec+NIDS***




***(b) Firewall+IPSec+NIDS***




***(c) IPSec+NIDS+Router***

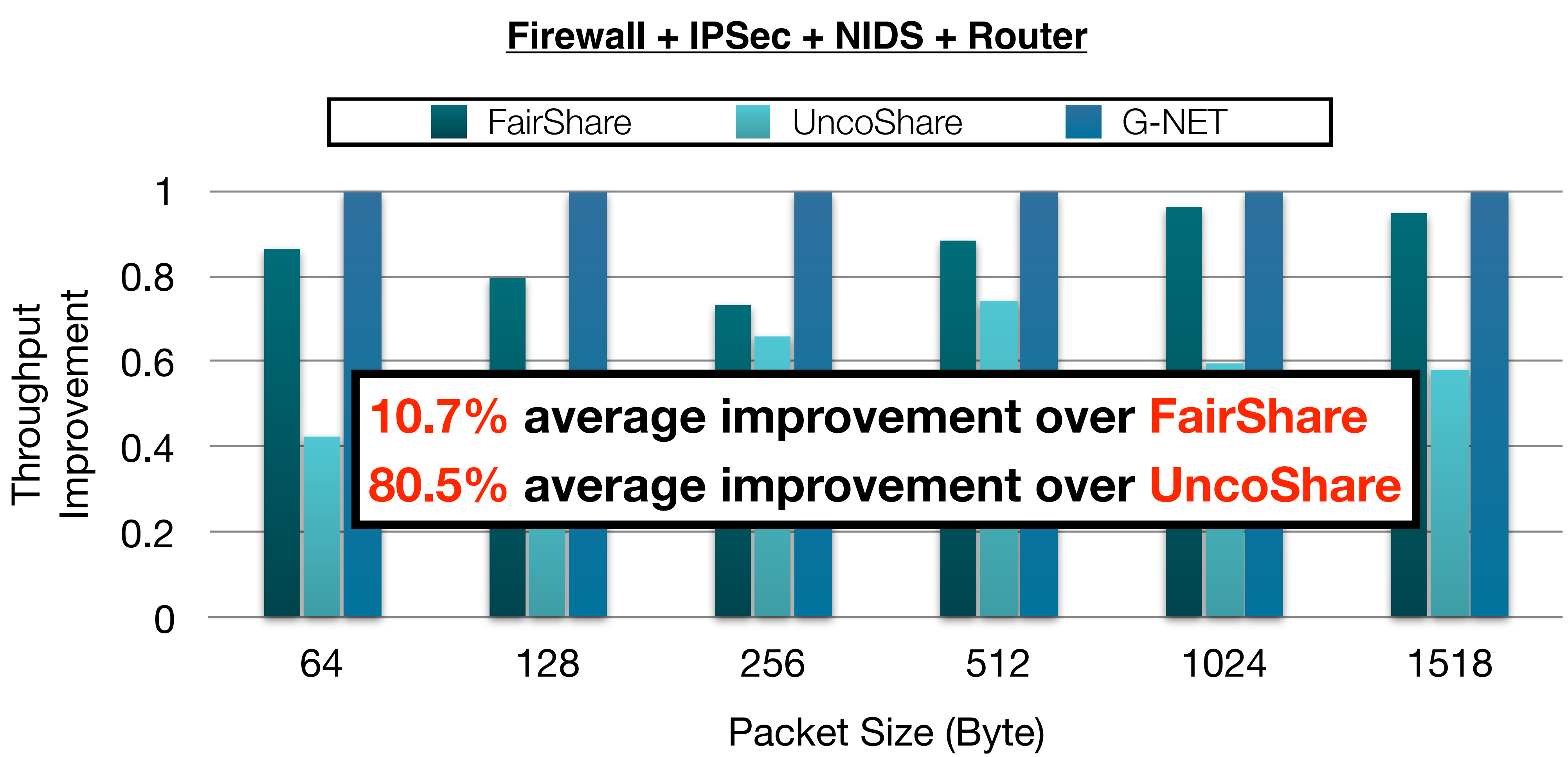



***(d) Firewall+IPSec+NIDS+Router***

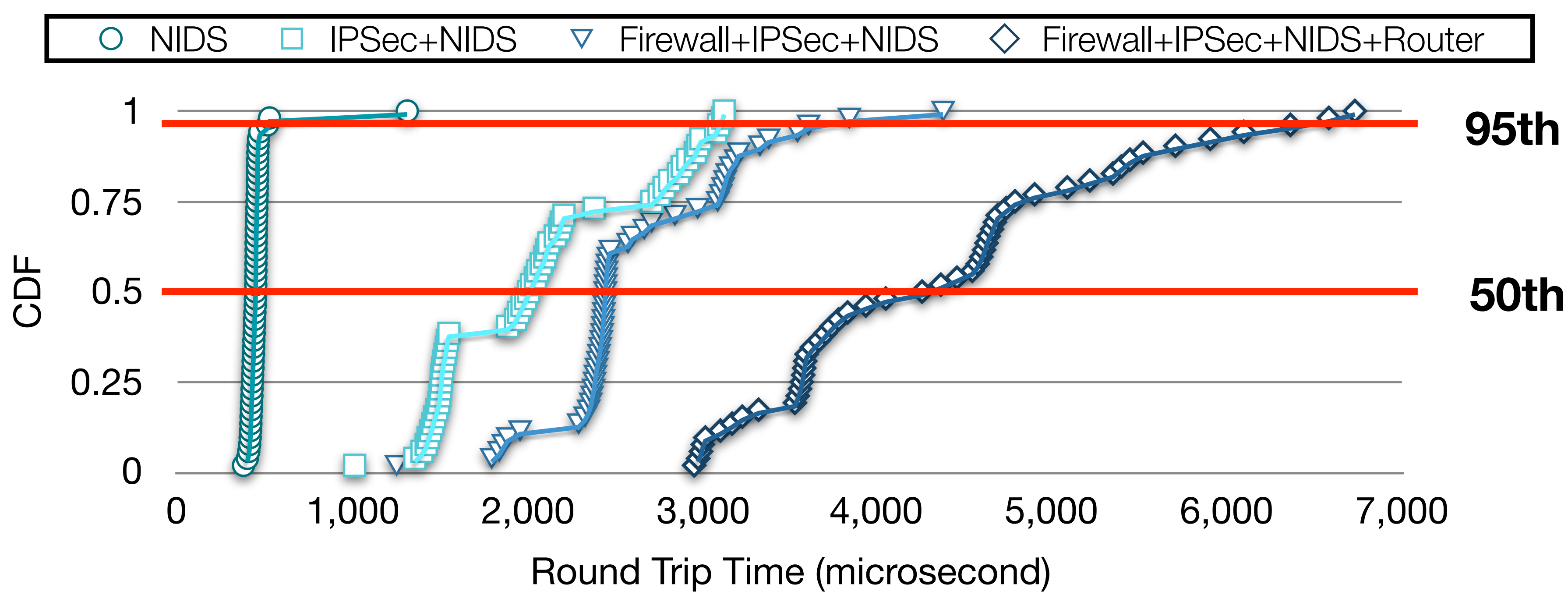
***More Resource Competition with four NFs***

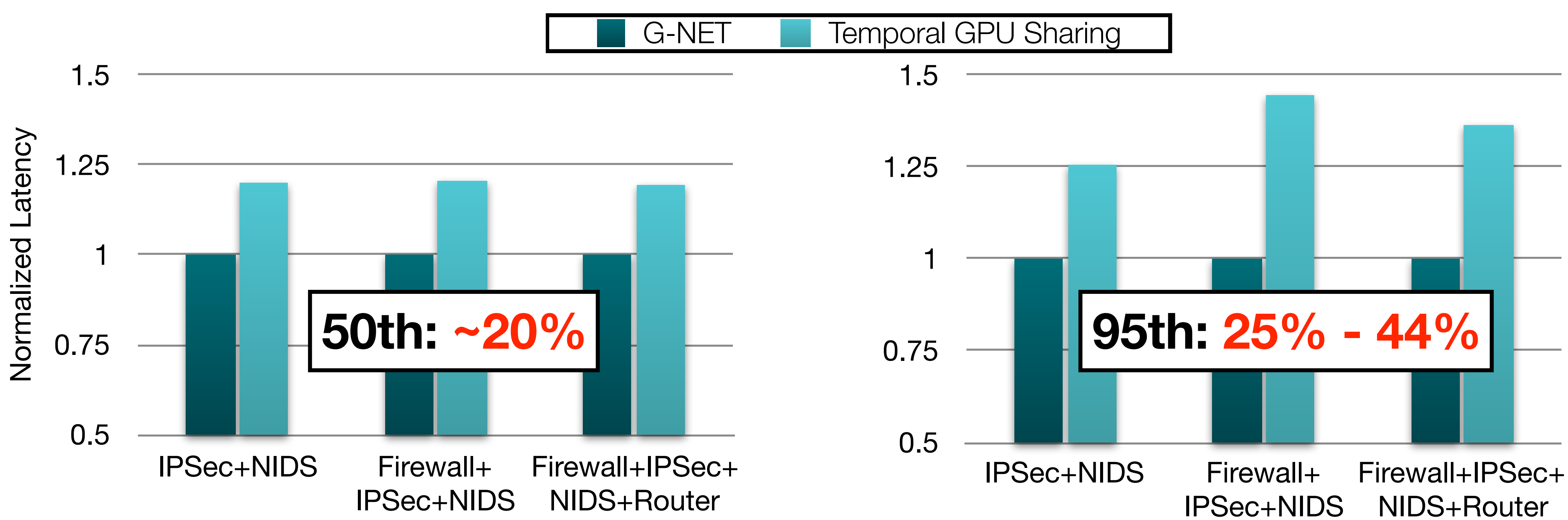
# Scheduling

- Service chain scheduling scheme comparison
  - **G-NET**: optimize the performance of the service chain
  - **FairShare**: Evenly partition compute resources among NFs
  - **UncoShare**: Each NF tries to maximize its resource allocation



# Latency




(a) 50th percentile latency

(b) 95th percentile latency

# Conclusion

- G-NET:
  - An NFV system that **efficiently** utilizes GPUs with **spatial GPU sharing**
  - **Service chain** based scheduling and resource allocation scheme
  - **Memory isolation** with IsoPointer
  - **Abstraction** that simplifies building GPU-accelerated NFs

- Experimental Results （Compare with temporal GPU sharing）
  - Enhances throughput by up to **70.8%**
  - Reduces latency by up to **44.3%**

**G-NET**

High-efficient platform for building GPU-based NFV systems