SG-IOV: Socket-Granular I/O Virtualization for SmartNIC-Based Container Networks

Frank Chenxingyu Zhao[1], Hongtao Zhang[1], Jaehong Min[1], Shengkai Lin[2],
Wei Zhang[3], Kaiyuan Zhang[1], Ming Liu[4], Arvind Krishnamurthy[1]

University of Washington[1] , Shanghai Jiao Tong University[2]
University of Connecticut[3], University of Wisconsin-Madison[4]

ASPLOS 2026

Container Networks Underpin Modern Cloud

Container Networks Underpin Modern Cloud

Container Service

Container Networks Underpin Modern Cloud

Container Service

Google
GKE

AWS
Fargate

Azure
AKS

Container Network (CNI)

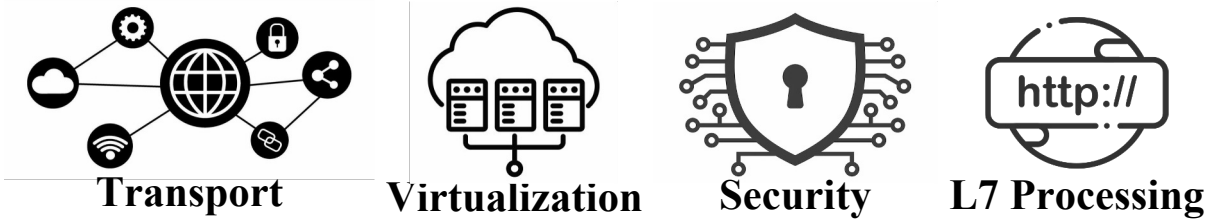
cilium

Container Networks Underpin Modern Cloud

Container Service

Container Network (CNI)

Rich Functionalities

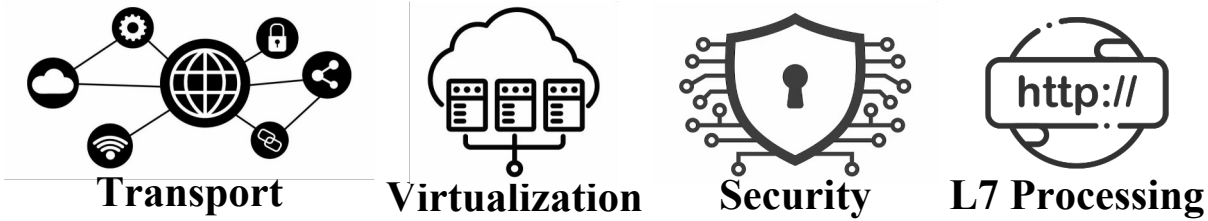


Container Networks Underpin Modern Cloud

Container Service

Container Network (CNI)

Rich Functionalities



More and more features—but what about the infrastructure cost?

Design Space (1/3): How to Build Efficient CNI?

Design Space (1/3): How to Build Efficient CNI?

Build Container Networks

Design Space (1/3): How to Build Efficient CNI?

- Choice#1: Software Solutions
  - *e.g.*, Cilium CNI

Build Container Networks

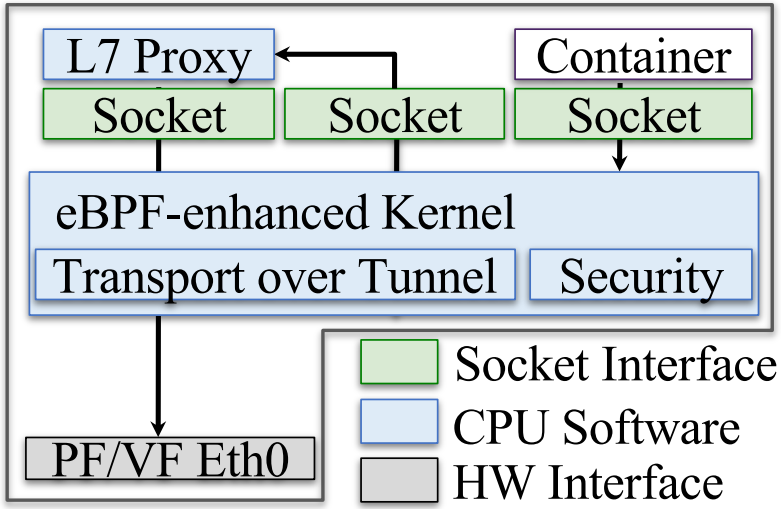
Software Solutions

Design Space (1/3): How to Build Efficient CNI?

- Choice#1: Software Solutions
  - *e.g.*, Cilium CNI

Build Container Networks

Software Solutions

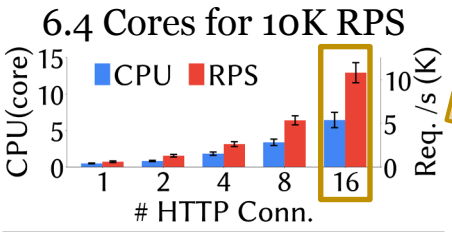


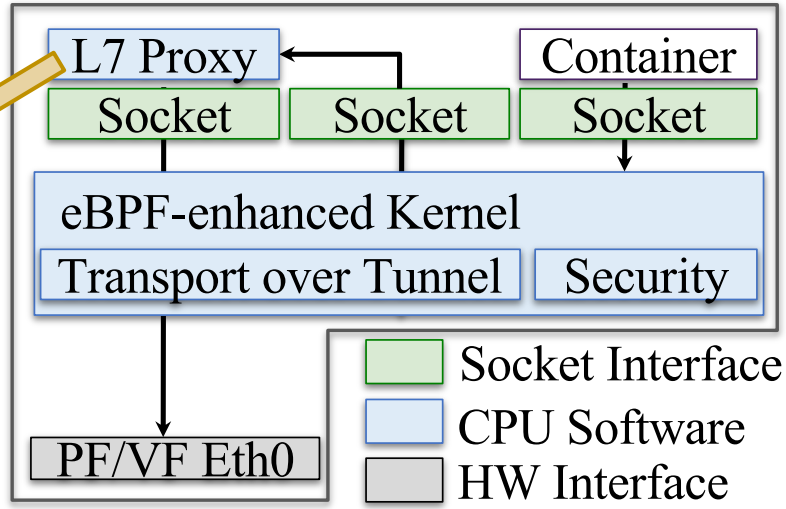
Design Space (1/3): How to Build Efficient CNI?

- Choice#1: Software Solutions
  - *e.g.*, Cilium CNI

Build Container Networks

Software Solutions



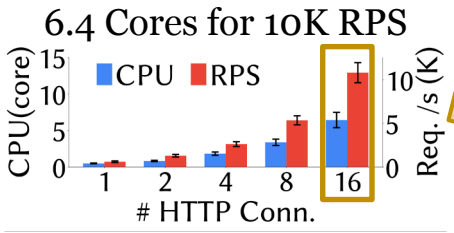


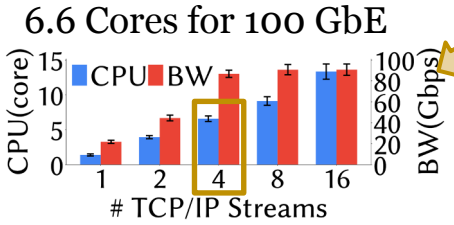
Design Space (1/3): How to Build Efficient CNI?

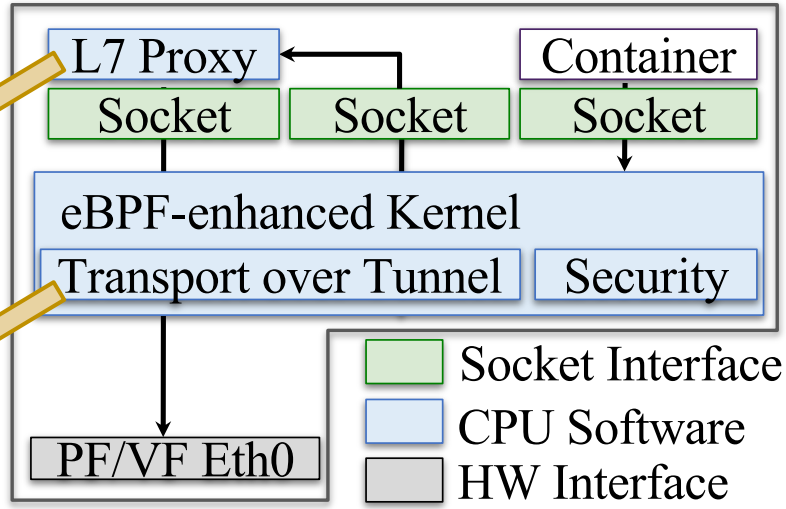
- Choice#1: Software Solutions
  - *e.g.*, Cilium CNI

Build Container Networks

Software Solutions





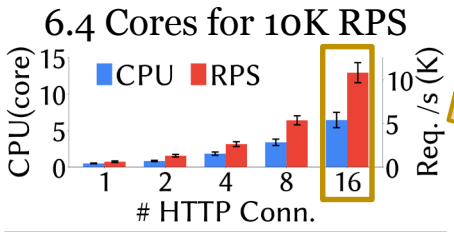


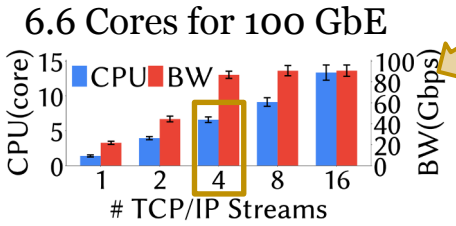
Design Space (1/3): How to Build Efficient CNI?

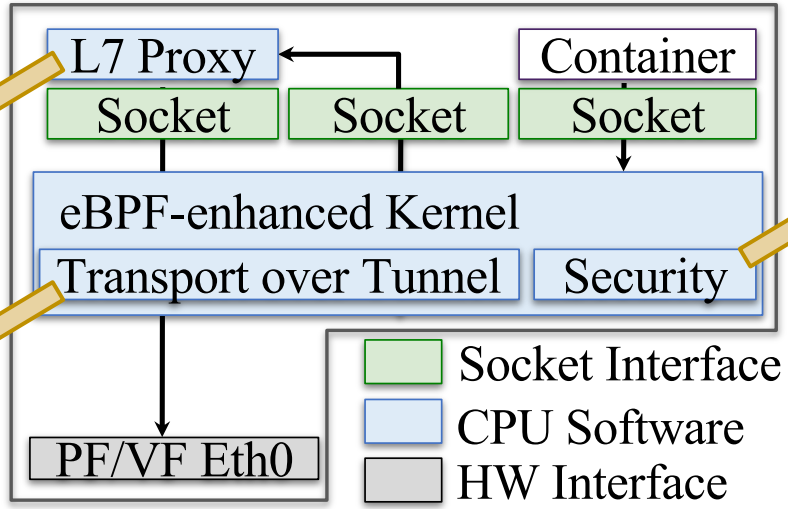
- Choice#1: Software Solutions
  - *e.g.*, Cilium CNI

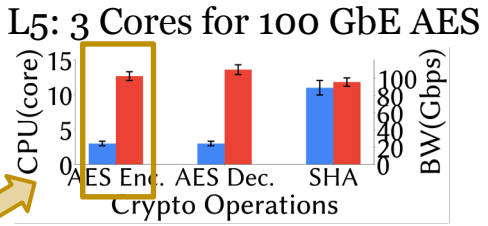
Build Container Networks

Software Solutions







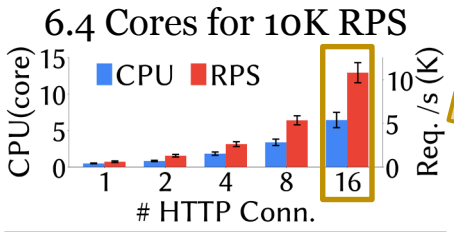


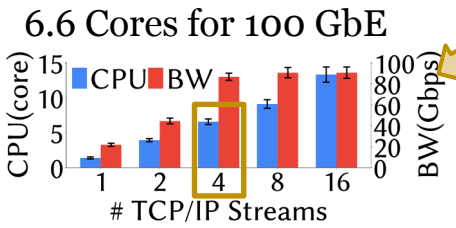
Design Space (1/3): How to Build Efficient CNI?

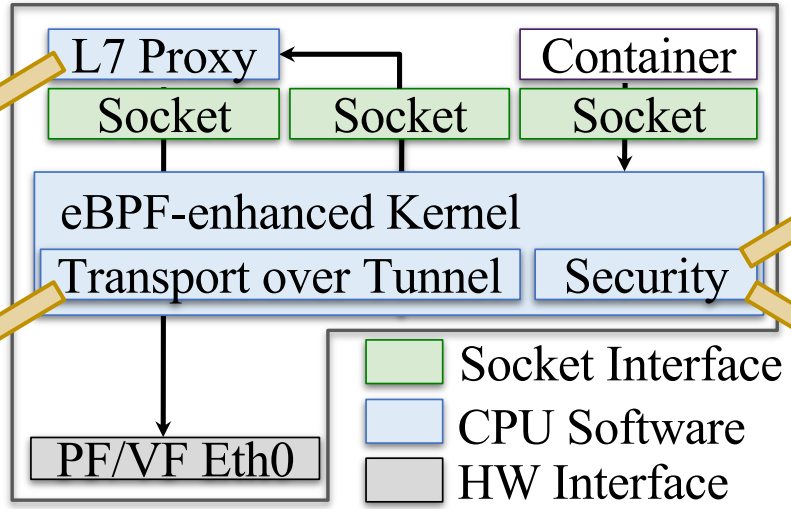
- Choice#1: Software Solutions
  - *e.g.*, Cilium CNI

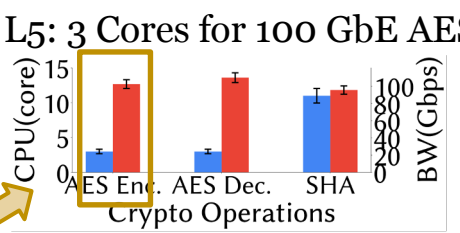
Build Container Networks

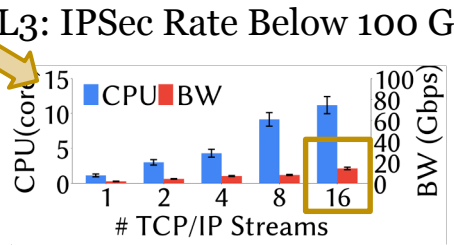
Software Solutions









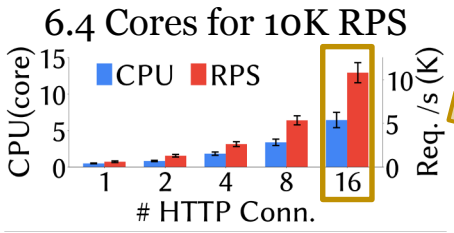


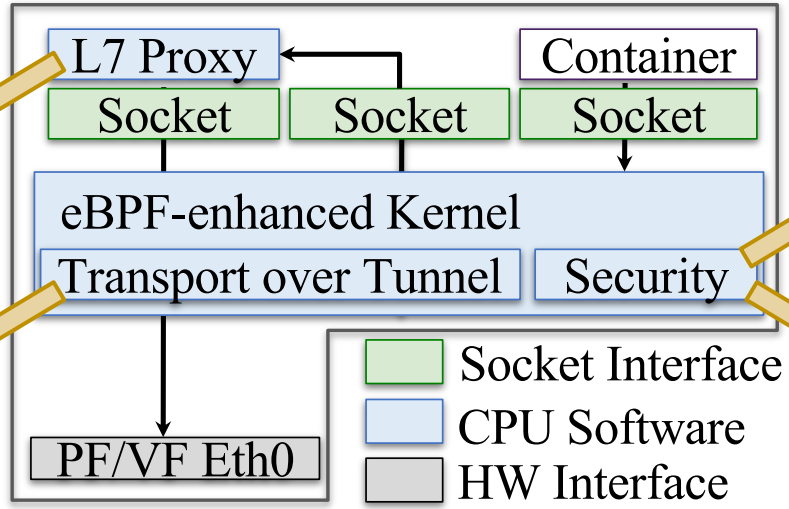
Design Space (1/3): How to Build Efficient CNI?

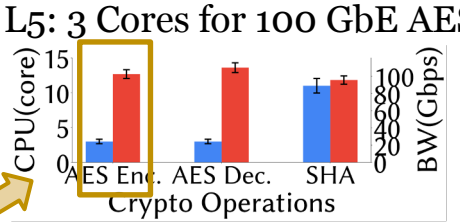
- Choice#1: Software Solutions
  - *e.g.*, Cilium CNI

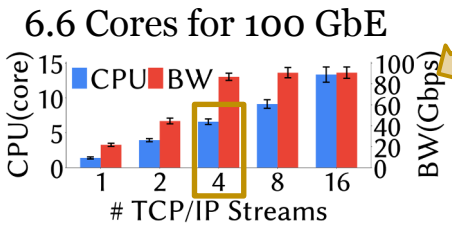
Build Container Networks

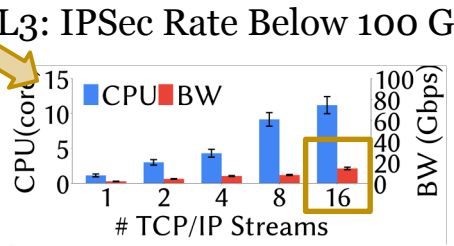
Software Solutions











Software CNI: high CPU tax, low efficiency

Design Space (2/3): How to Build Efficient CNI?

Build Container Networks

Software Solutions

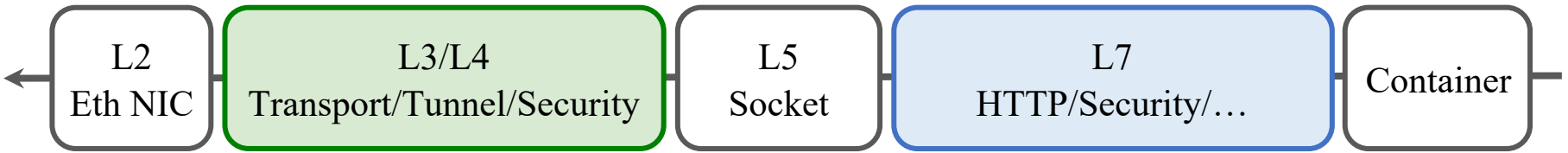
Design Space (2/3): How to Build Efficient CNI?

Design Space (2/3): How to Build Efficient CNI?

- Choice#2: Which Layer to Offload?

Build Container Networks

Software Solutions | **Hardware Offloading**

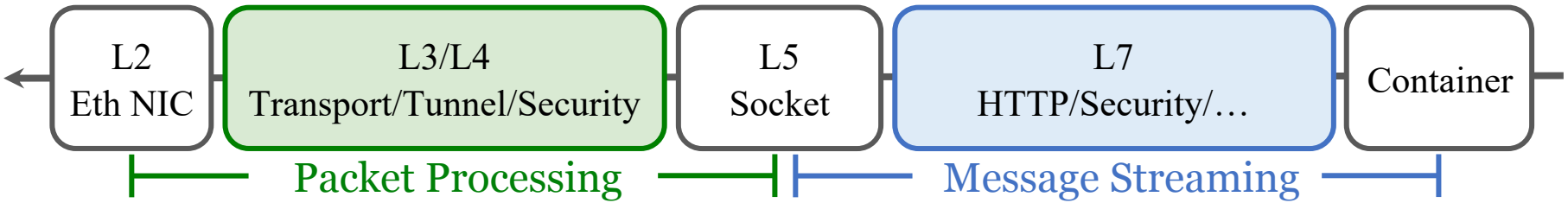


Design Space (2/3): How to Build Efficient CNI?

- Choice#2: Which Layer to Offload?
  - Key Boundary: L5 Socket

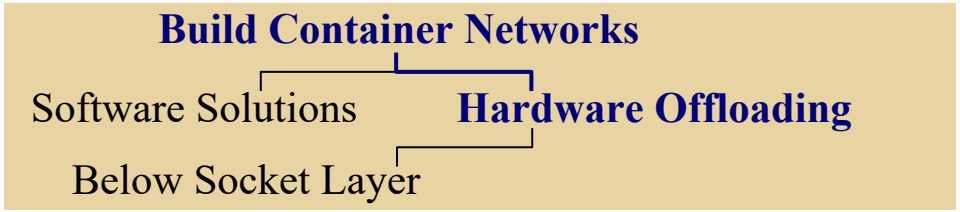
Build Container Networks

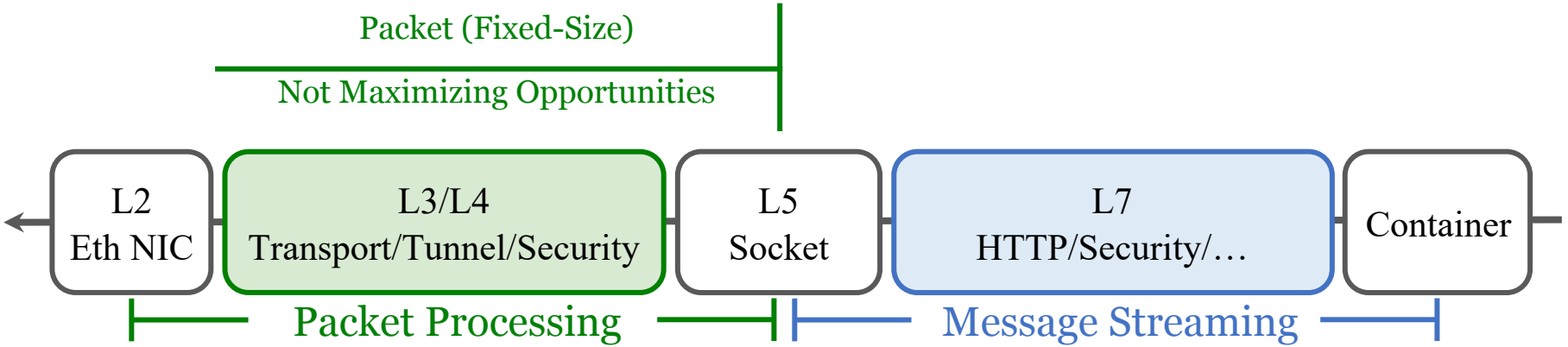
Software Solutions | **Hardware Offloading**



Design Space (2/3): How to Build Efficient CNI?

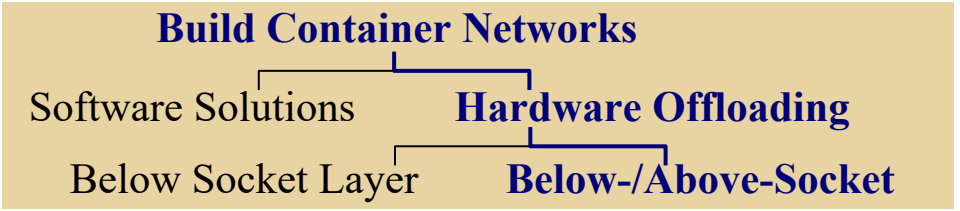
- Choice#2: Which Layer to Offload?
  - Key Boundary: L5 Socket

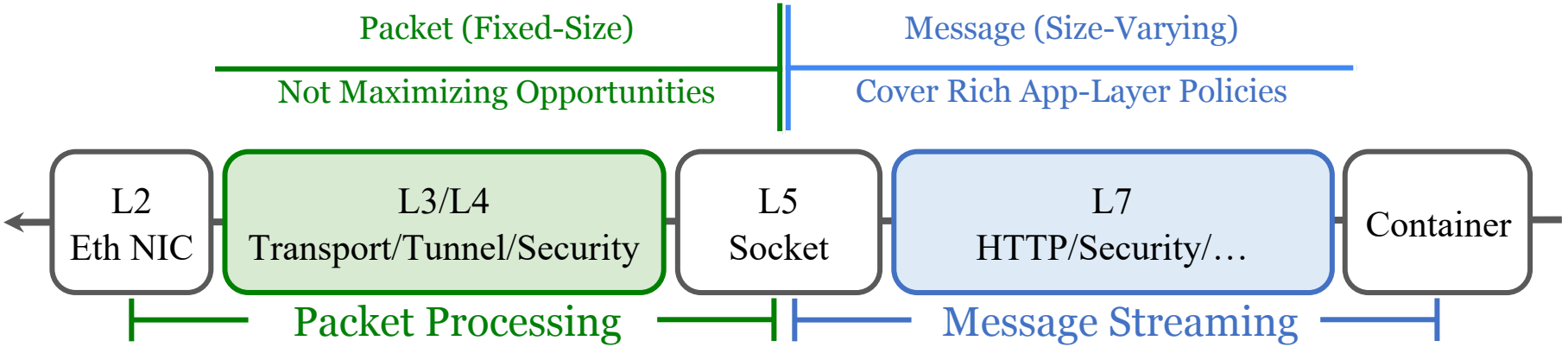




Design Space (2/3): How to Build Efficient CNI?

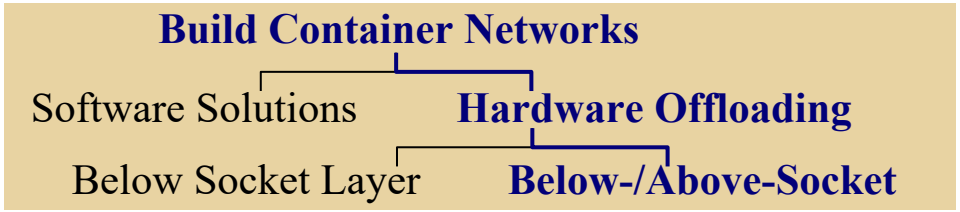
- Choice#2: Which Layer to Offload?
  - Key Boundary: L5 Socket

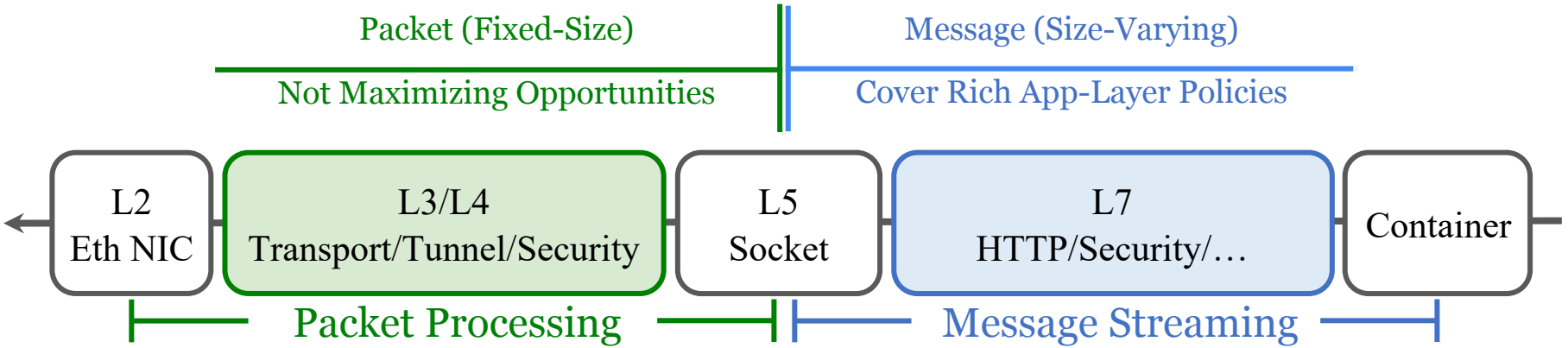




Design Space (2/3): How to Build Efficient CNI?

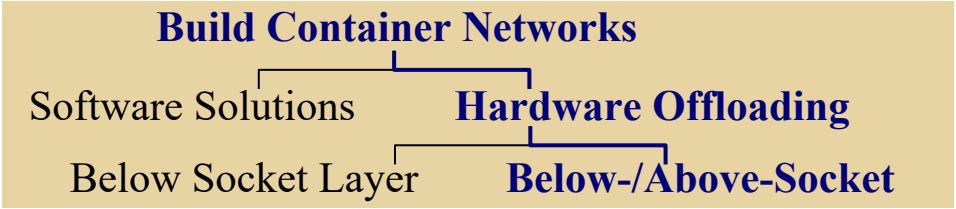
- Choice#2: Which Layer to Offload?
  - Key Boundary: L5 Socket





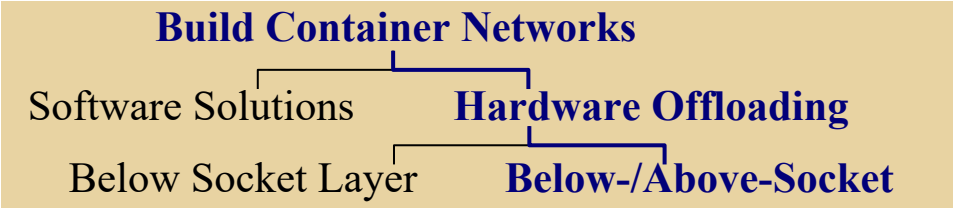
L5+ offload: more opportunities, but rich policies require streaming messages

Design Space (3/3): How to Build Efficient CNI?



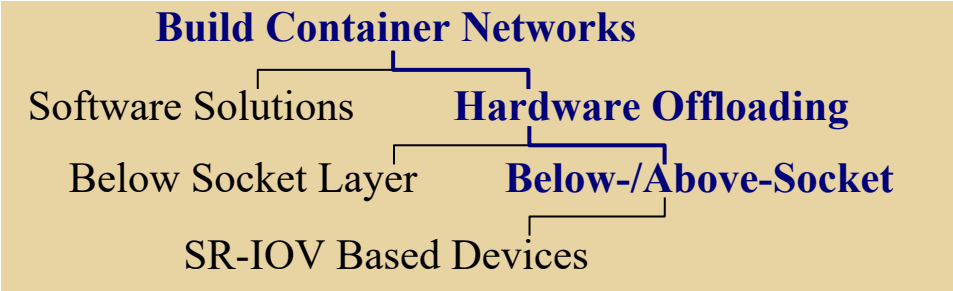
Design Space (3/3): How to Build Efficient CNI?

- Choice#3: How to Build an IOV Interface for L5+ Offloading?



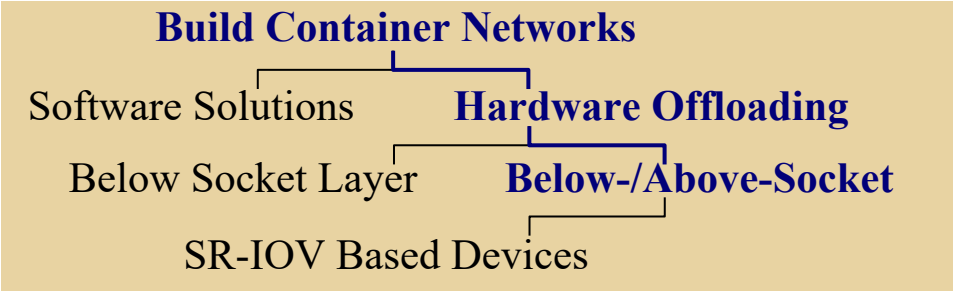
Design Space (3/3): How to Build Efficient CNI?

- Choice#3: How to Build an IOV Interface for L5+ Offloading?
  - Limitations of the *de facto* SR-IOV



Design Space (3/3): How to Build Efficient CNI?

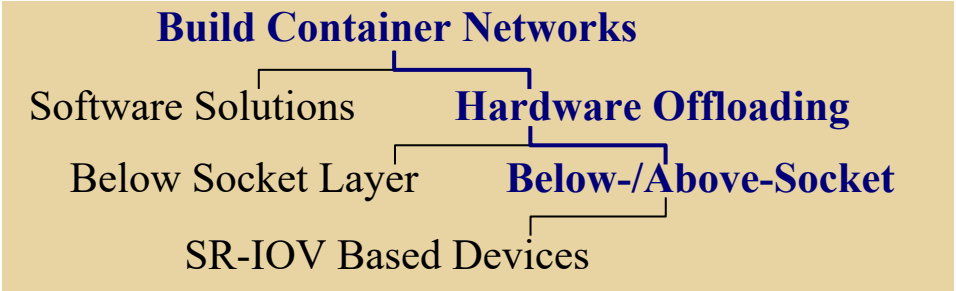
- Choice#3: How to Build an IOV Interface for L5+ Offloading?
  - Limitations of the *de facto* SR-IOV



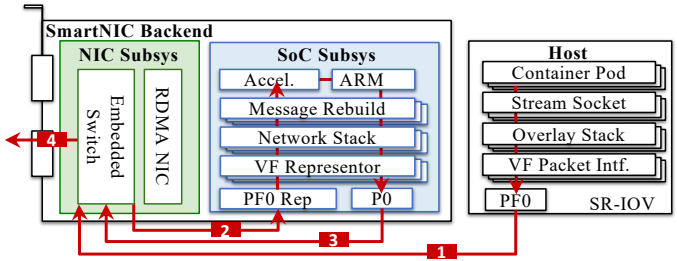
	SR-IOV	Container Needs
Scalability Gap	Low Device Count (100s VF/PF)	High Container Density (1k+ Containers/Node)

Design Space (3/3): How to Build Efficient CNI?

- Choice#3: How to Build an IOV Interface for L5+ Offloading?
  - Limitations of the *de facto* SR-IOV

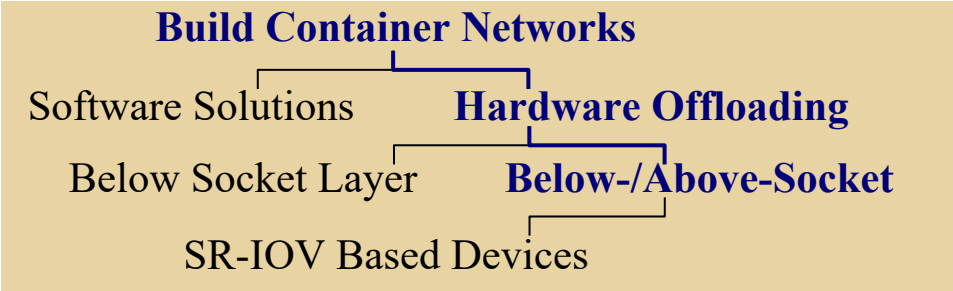


	SR-IOV	Container Needs
Scalability Gap	Low Device Count (100s VF/PF)	High Container Density (1k+ Containers/Node)
Stream-to-Packet Cyclic Transform.	Packet Orientation	Streaming Messages

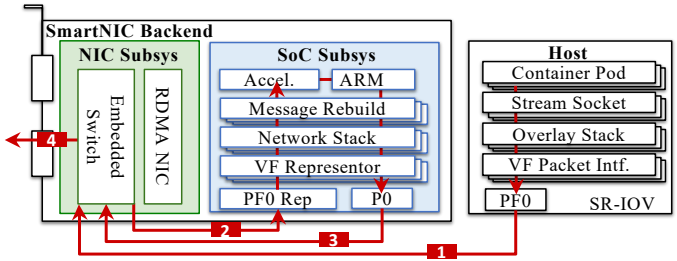


Design Space (3/3): How to Build Efficient CNI?

- Choice#3: How to Build an IOV Interface for L5+ Offloading?
  - Limitations of the *de facto* SR-IOV

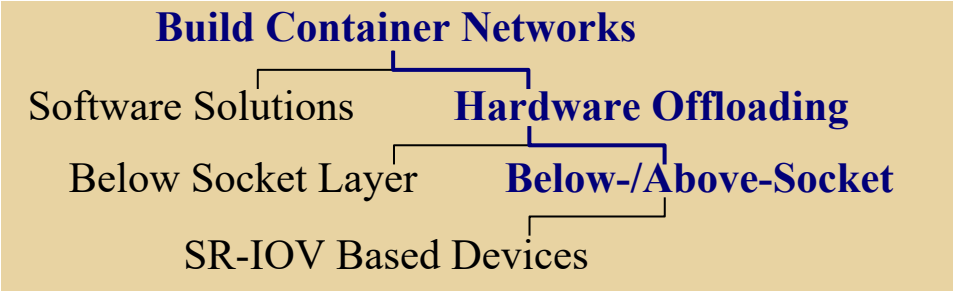


	SR-IOV	Container Needs
Scalability Gap	Low Device Count (100s VF/PF)	High Container Density (1k+ Containers/Node)
Stream-to-Packet Cyclic Transform.	Packet Orientation	Streaming Messages
Coarse-Grained Virtualization	Pre-Configured Rate Control	Per-Job Dynamic Scheduling

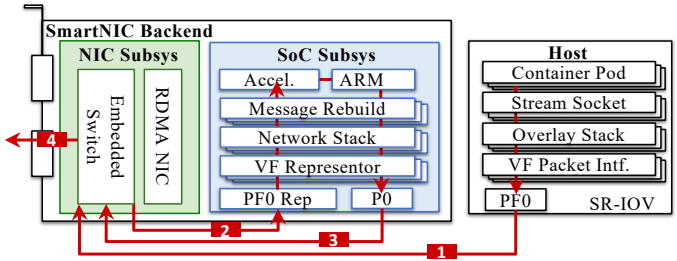


Design Space (3/3): How to Build Efficient CNI?

- Choice#3: How to Build an IOV Interface for L5+ Offloading?
  - Limitations of the *de facto* SR-IOV

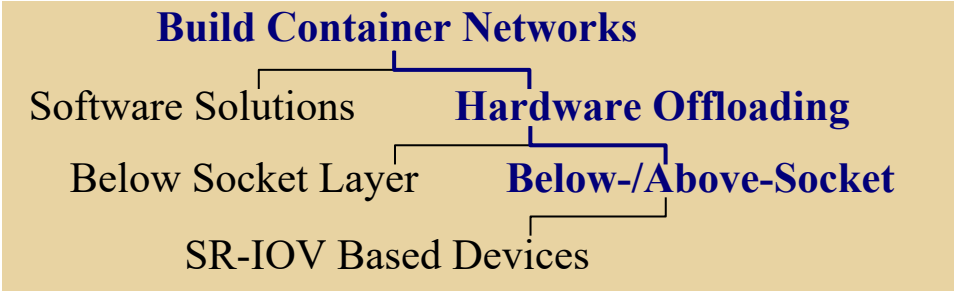


	SR-IOV	Container Needs
Scalability Gap	Low Device Count (100s VF/PF)	High Container Density (1k+ Containers/Node)
Stream-to-Packet Cyclic Transform.	Packet Orientation	Streaming Messages
Coarse-Grained Virtualization	Pre-Configured Rate Control	Per-Job Dynamic Scheduling



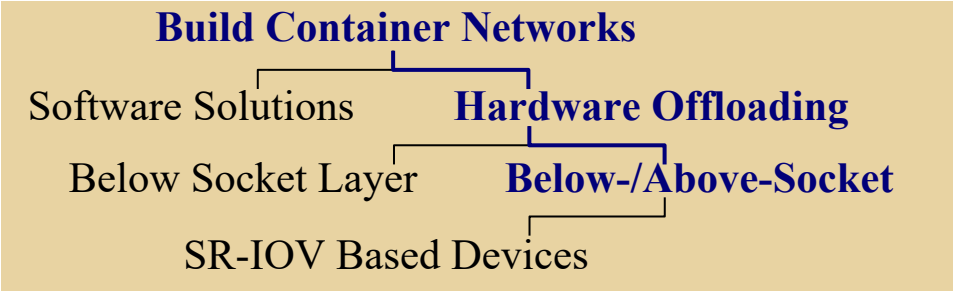
Existing SR-IOV is misaligned with container needs

Design Space (3/3): How to Build Efficient CNI?



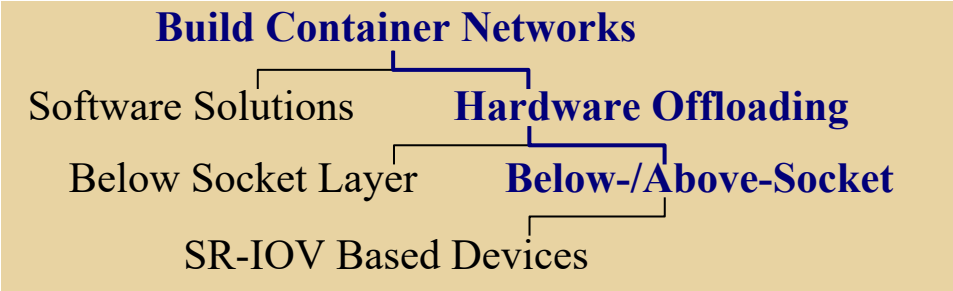
Design Space (3/3): How to Build Efficient CNI?

- Let's Rethink Hardware IOV



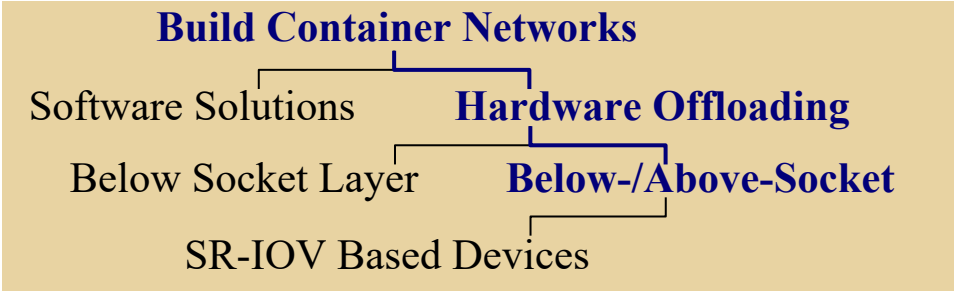
Design Space (3/3): How to Build Efficient CNI?

- Let's Rethink Hardware IOV



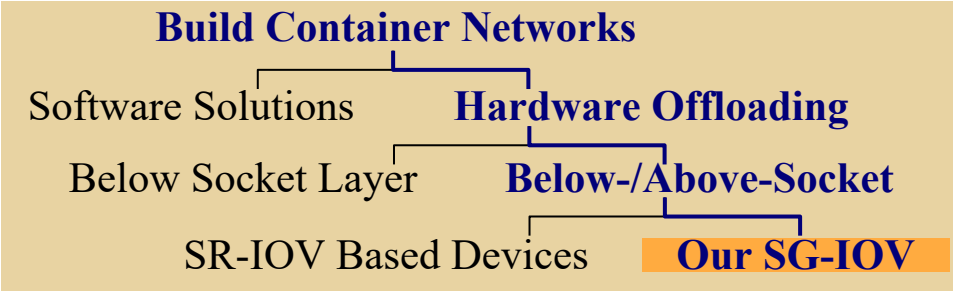
Can Hardware I/O Virtualization Operate at L5 Socket Granularity?

Ours: Socket-Granular I/O Virtualization



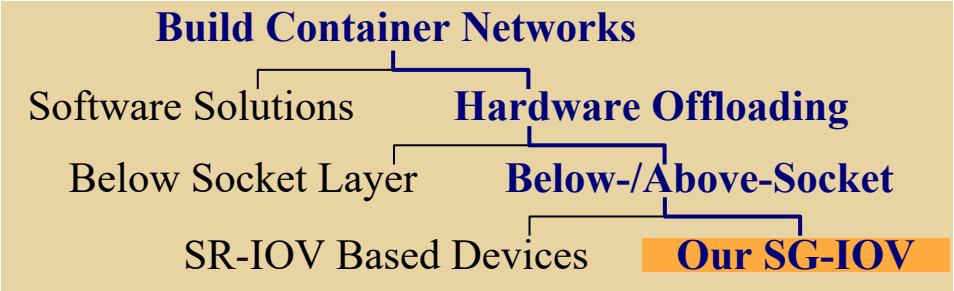
Ours: Socket-Granular I/O Virtualization

- SG-IOV: A New IOV Mechanism
  - Socket-Level Devices
  - Message-Stream Abstraction
  - Granular Virtualization



Ours: Socket-Granular I/O Virtualization

- SG-IOV: A New IOV Mechanism
  - Socket-Level Devices
  - Message-Stream Abstraction
  - Granular Virtualization



	SW-CNI	SR-IOV	Sub-Func.	TOE	**SG-IOV**
HW Offload	❌	✅	✅	✅	✅
Scalability	✅	❌	✅	✅	✅
Flexibility	✅	❌	❌	❌	✅
Granular HW Virtualization	❌	❌	❌	❌	✅

**SW-CNI: Software CNI; Sub-Func: NVIDIA Scalable Functions; TOE: TCP Offload Engine*

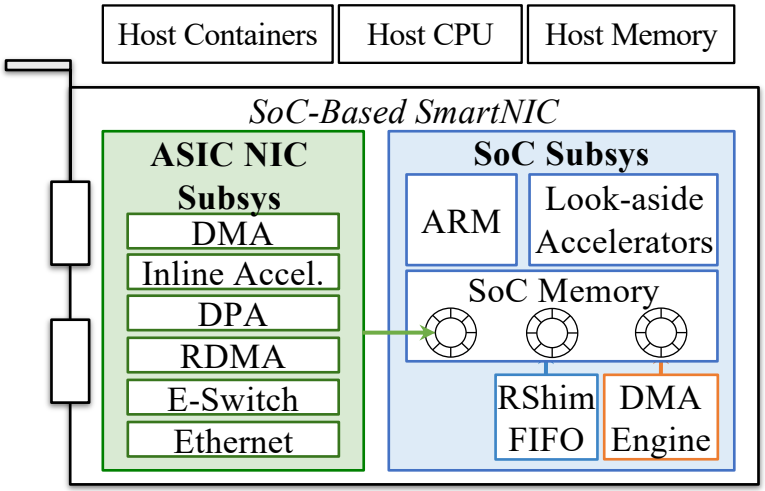
Enabler: SmartNIC Offers Diverse Processing Units

Enabler: SmartNIC Offers Diverse Processing Units

- End Host w/ SmartNIC Enclosing Heterogeneous Units

Enabler: SmartNIC Offers Diverse Processing Units

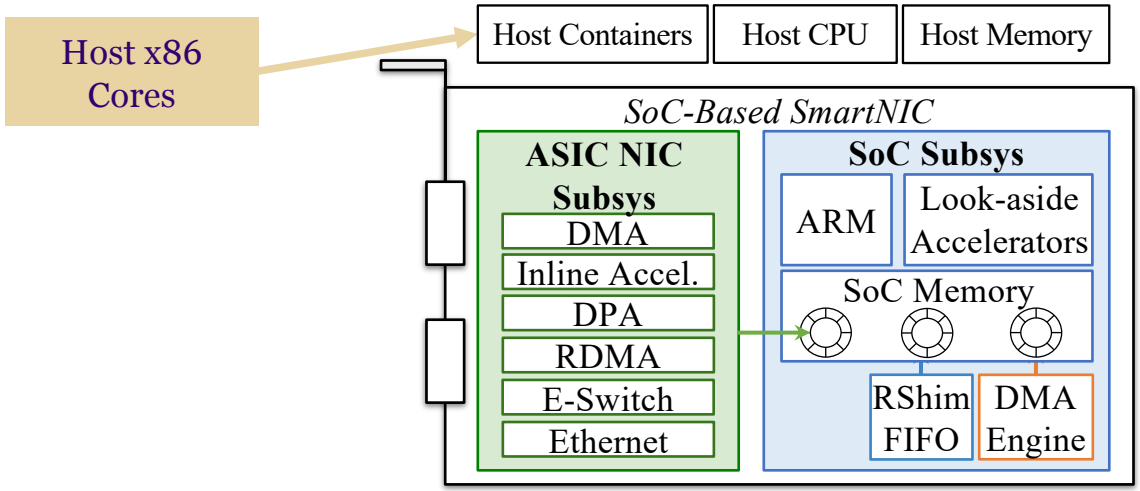
- End Host w/ SmartNIC Enclosing Heterogeneous Units



[NVIDIA BlueField-3 SmartNIC as an Example]

Enabler: SmartNIC Offers Diverse Processing Units

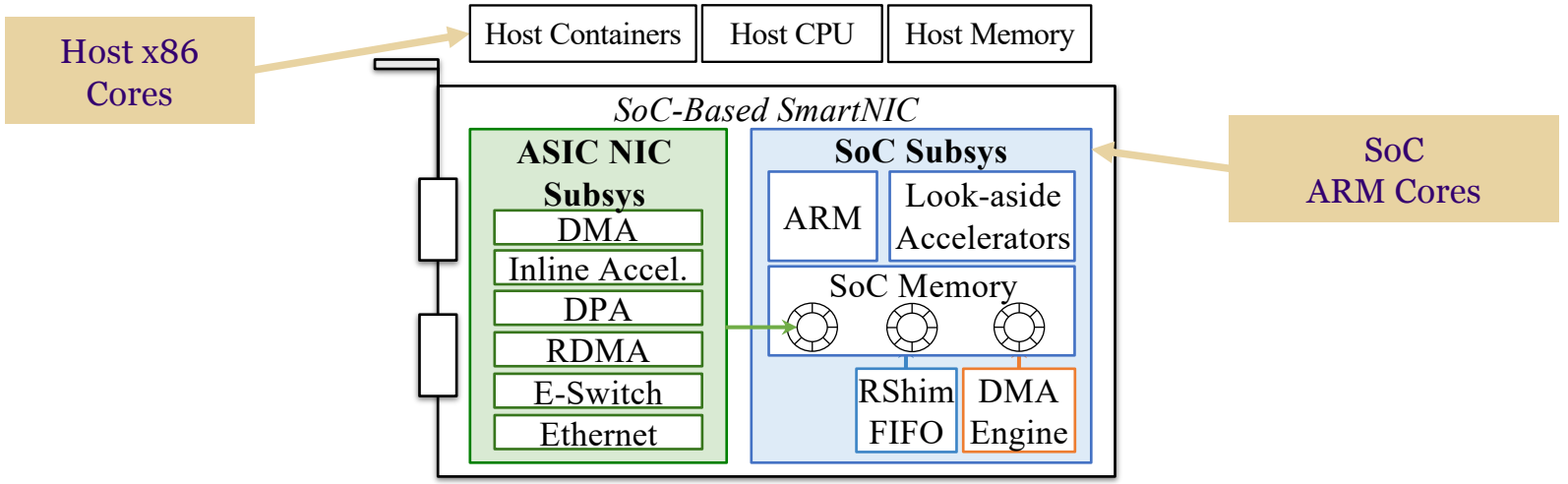
- End Host w/ SmartNIC Enclosing Heterogeneous Units



[NVIDIA BlueField-3 SmartNIC as an Example]

Enabler: SmartNIC Offers Diverse Processing Units

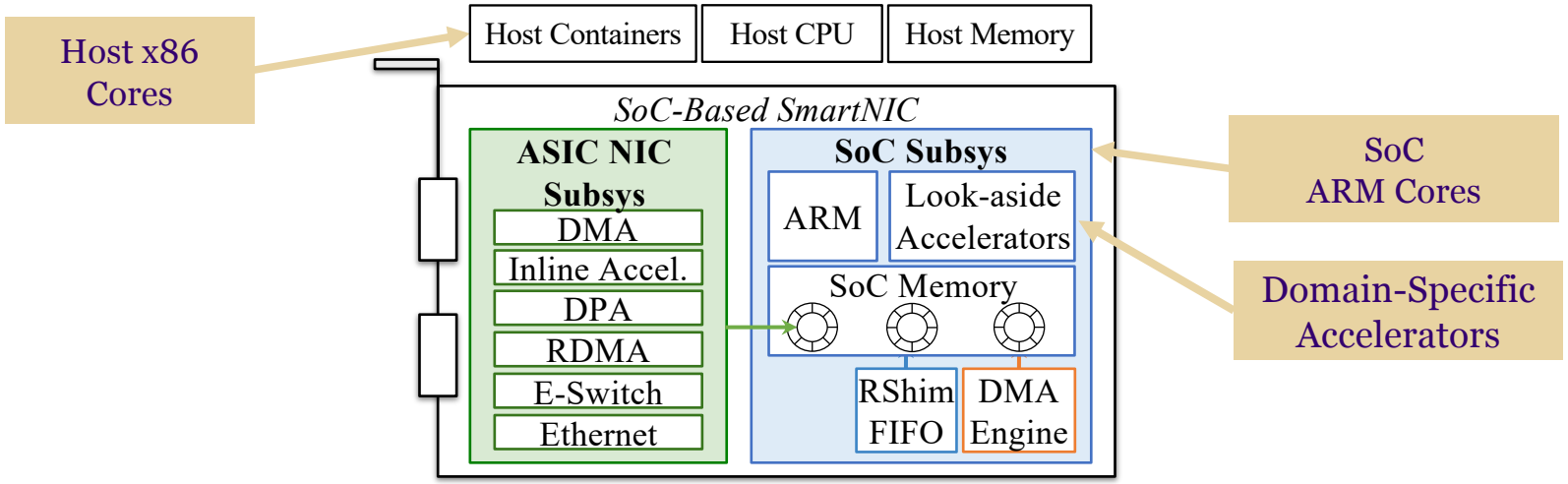
- End Host w/ SmartNIC Enclosing Heterogeneous Units



[NVIDIA BlueField-3 SmartNIC as an Example]

Enabler: SmartNIC Offers Diverse Processing Units

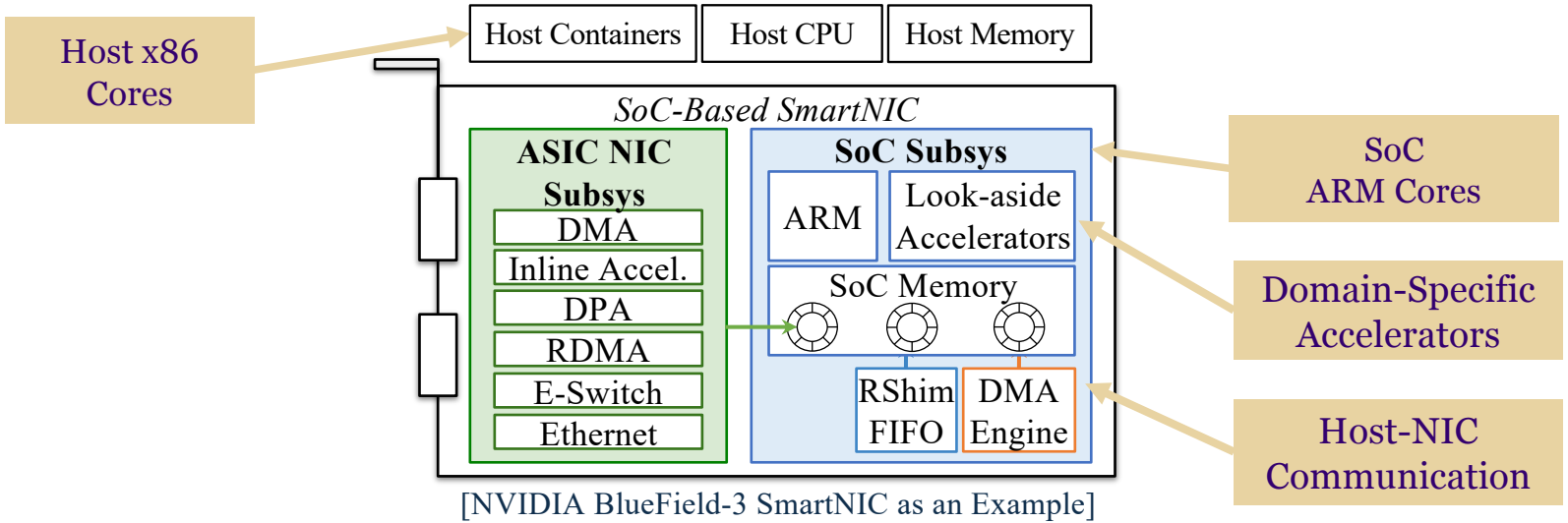
- End Host w/ SmartNIC Enclosing Heterogeneous Units



[NVIDIA BlueField-3 SmartNIC as an Example]

Enabler: SmartNIC Offers Diverse Processing Units

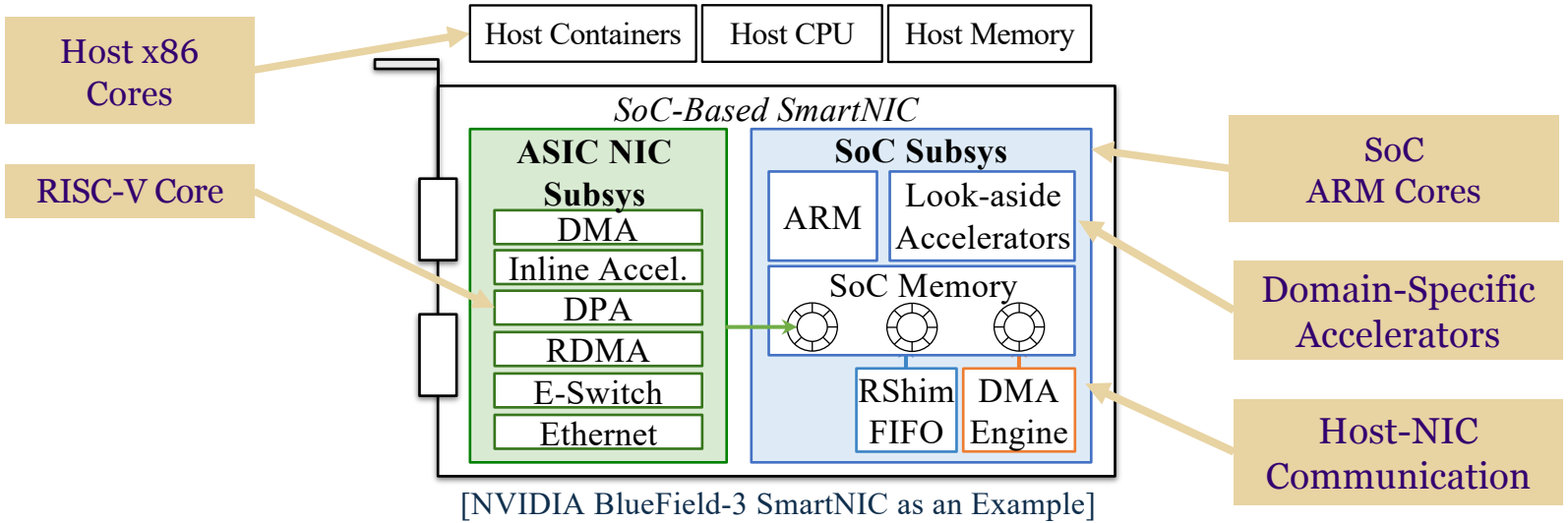
- End Host w/ SmartNIC Enclosing Heterogeneous Units



[NVIDIA BlueField-3 SmartNIC as an Example]

Enabler: SmartNIC Offers Diverse Processing Units

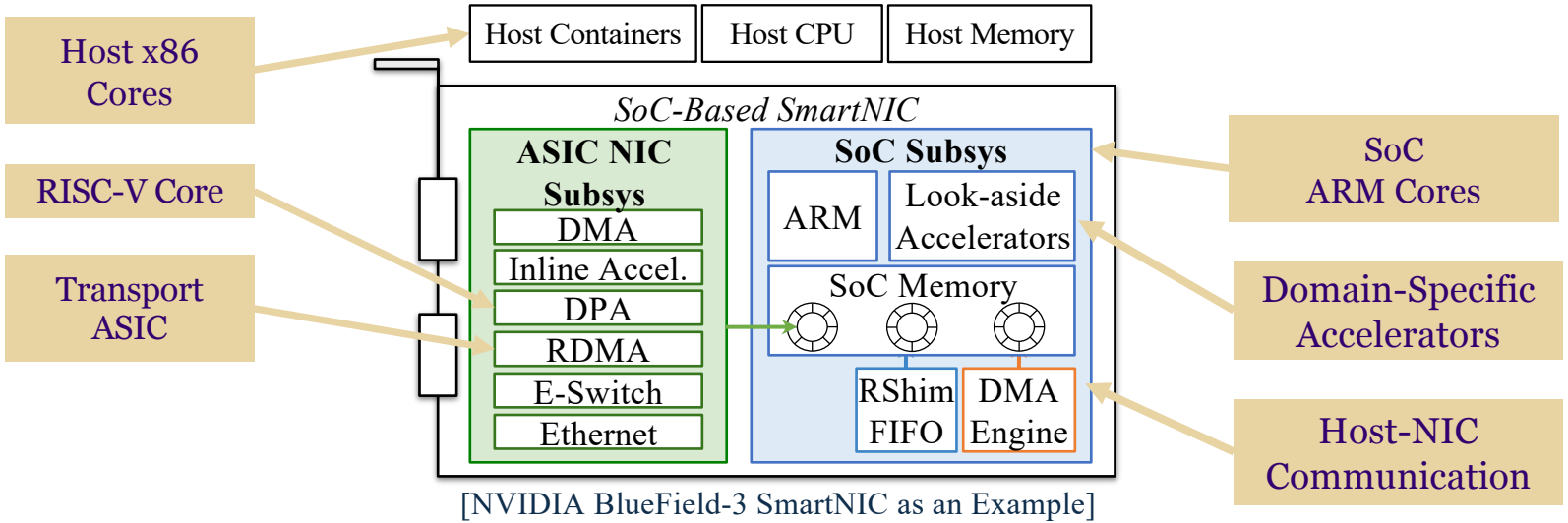
- End Host w/ SmartNIC Enclosing Heterogeneous Units



[NVIDIA BlueField-3 SmartNIC as an Example]

Enabler: SmartNIC Offers Diverse Processing Units

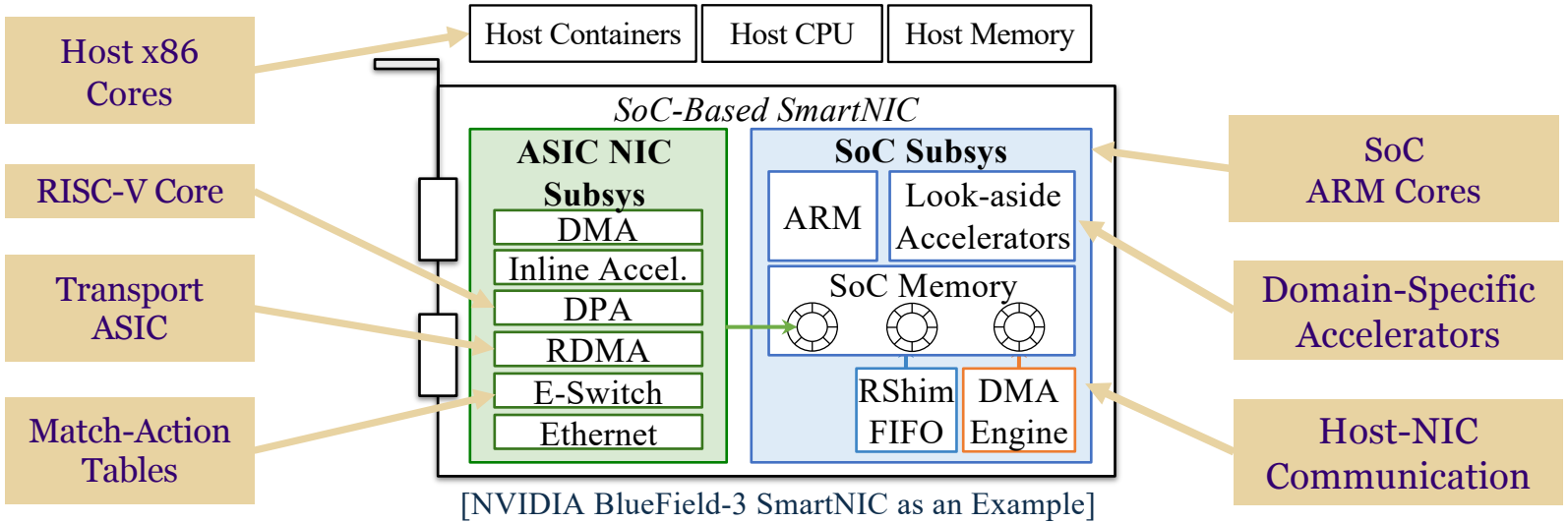
- End Host w/ SmartNIC Enclosing Heterogeneous Units



[NVIDIA BlueField-3 SmartNIC as an Example]

Enabler: SmartNIC Offers Diverse Processing Units

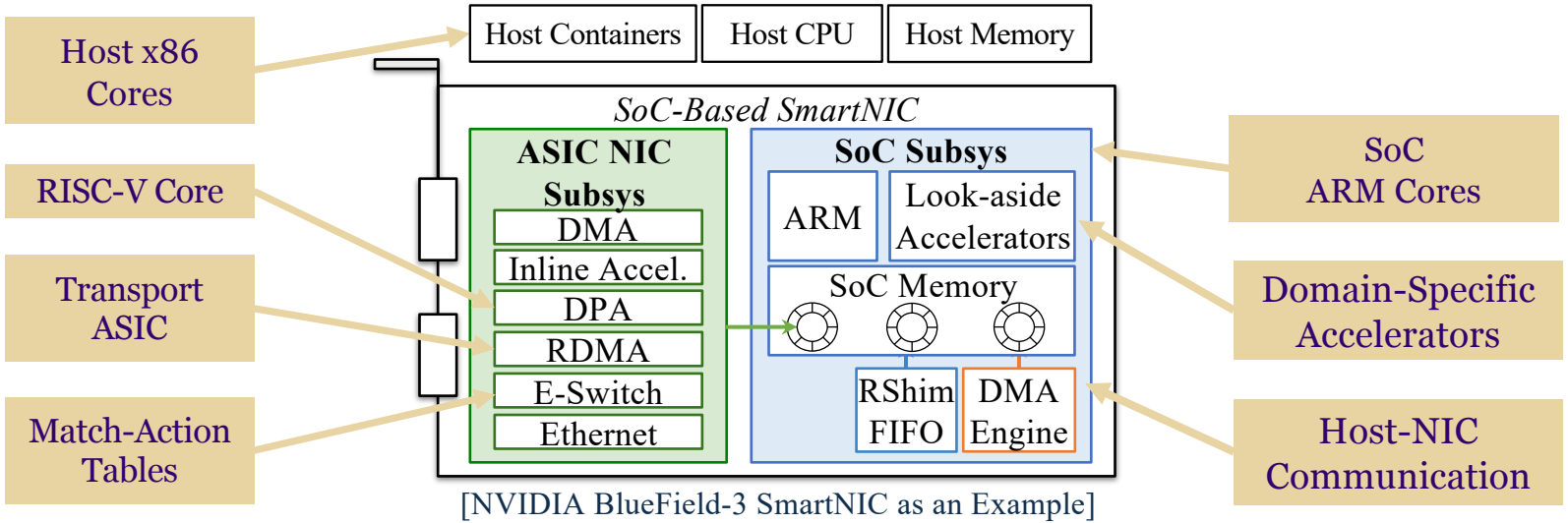
- End Host w/ SmartNIC Enclosing Heterogeneous Units



[NVIDIA BlueField-3 SmartNIC as an Example]

Enabler: SmartNIC Offers Diverse Processing Units

- End Host w/ SmartNIC Enclosing Heterogeneous Units



[NVIDIA BlueField-3 SmartNIC as an Example]

SR-IOV (2007 V1) was invented ~20 years ago
New devices and scenarios call for rethinking IOV

Overview (1/2): Signal and Data Plane Separation

Overview (1/2): Signal and Data Plane Separation

Overview (1/2): Signal and Data Plane Separation

- Signal Plane over SmartNIC-embedded Cores

Overview (1/2): Signal and Data Plane Separation

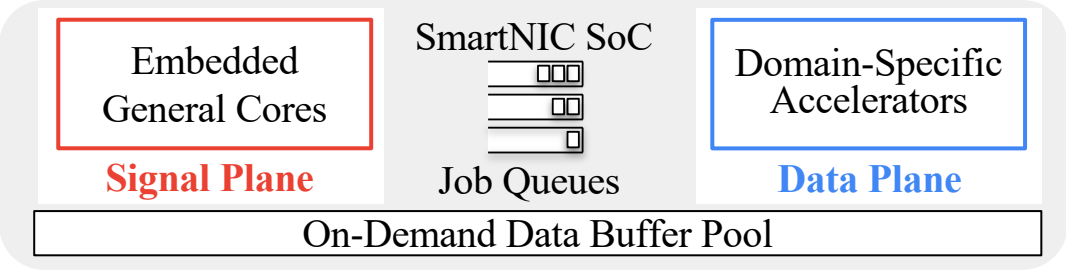
- Signal Plane over SmartNIC-embedded Cores
- Data Plane over SmartNIC-side Accelerators

Overview (1/2): Signal and Data Plane Separation

- Signal Plane over SmartNIC-embedded Cores
- Data Plane over SmartNIC-side Accelerators

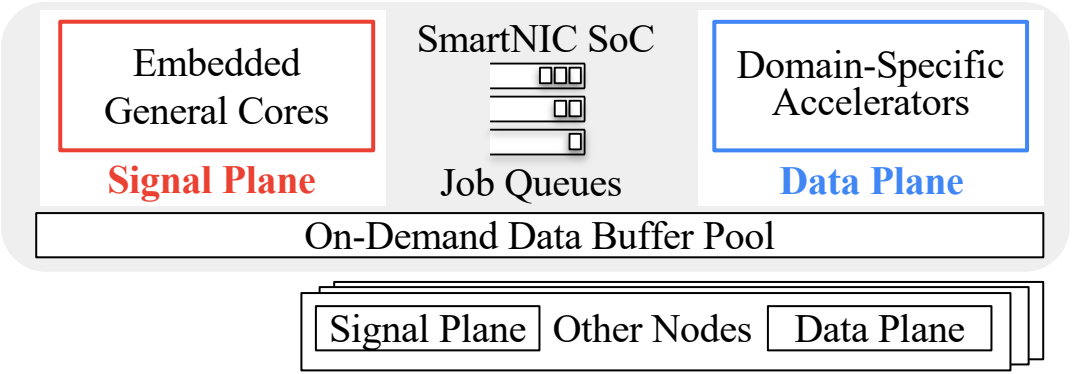
Overview (1/2): Signal and Data Plane Separation

- Signal Plane over SmartNIC-embedded Cores
- Data Plane over SmartNIC-side Accelerators



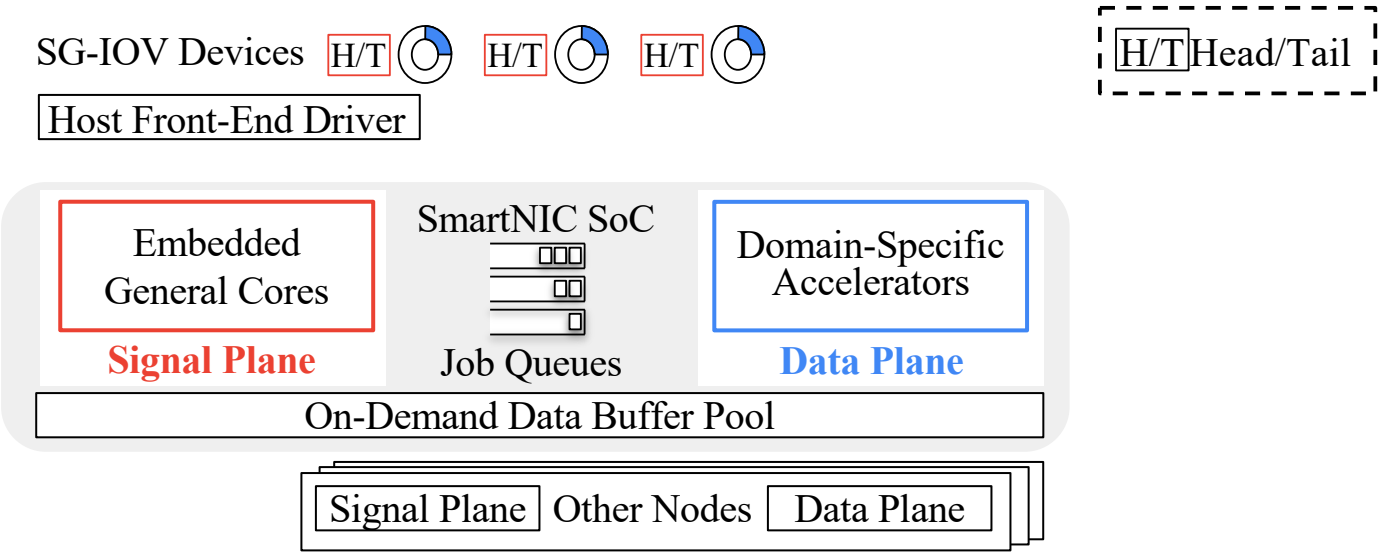
Overview (1/2): Signal and Data Plane Separation

- Signal Plane over SmartNIC-embedded Cores
- Data Plane over SmartNIC-side Accelerators



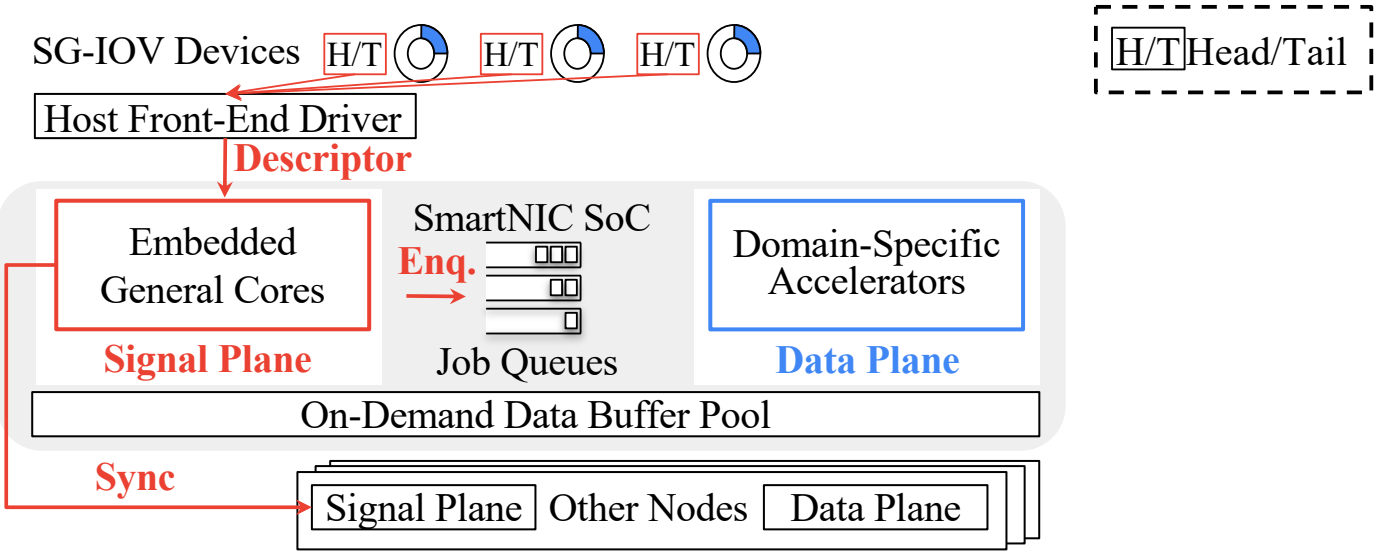
Overview (1/2): Signal and Data Plane Separation

- Signal Plane over SmartNIC-embedded Cores
- Data Plane over SmartNIC-side Accelerators



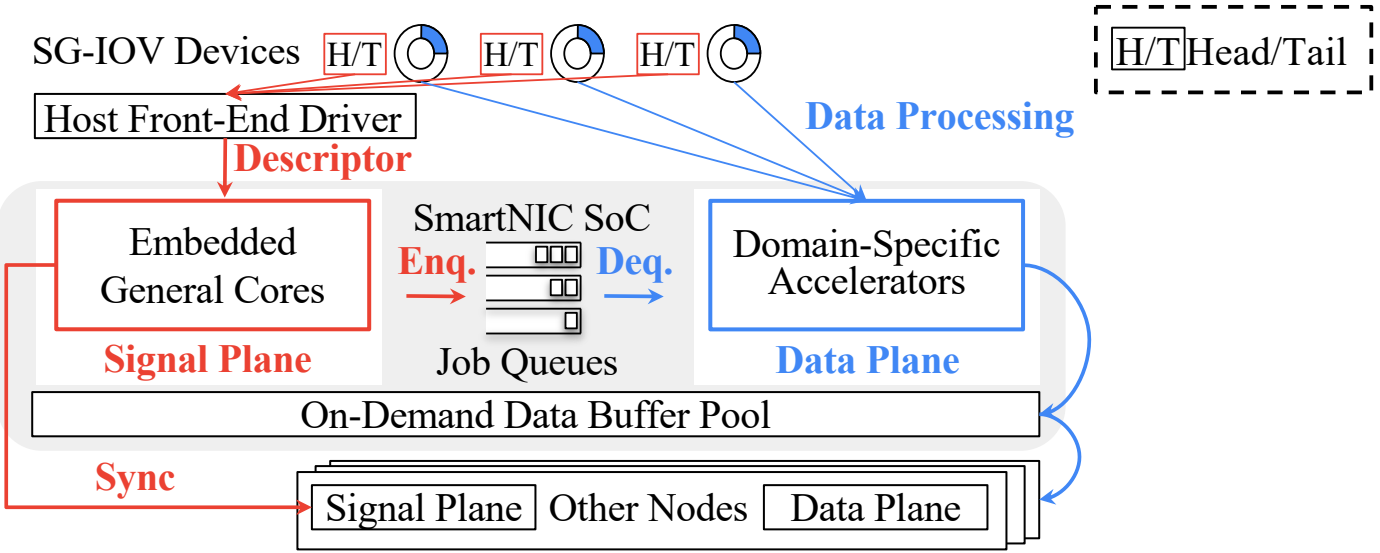
Overview (1/2): Signal and Data Plane Separation

- Signal Plane over SmartNIC-embedded Cores
- Data Plane over SmartNIC-side Accelerators



Overview (1/2): Signal and Data Plane Separation

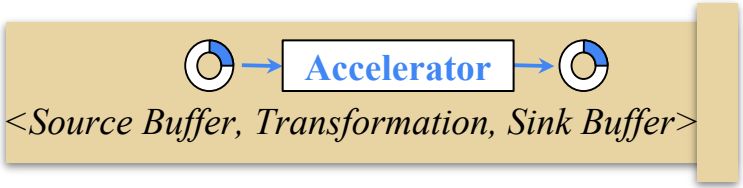
- Signal Plane over SmartNIC-embedded Cores
- Data Plane over SmartNIC-side Accelerators



Overview (2/2): Chainable Execution over Warp Pipes

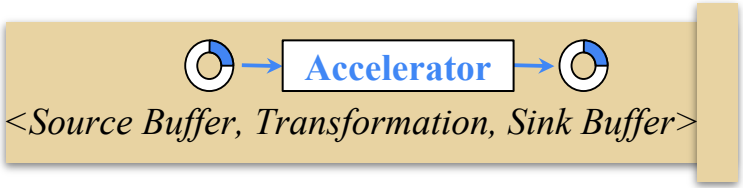
Overview (2/2): Chainable Execution over Warp Pipes

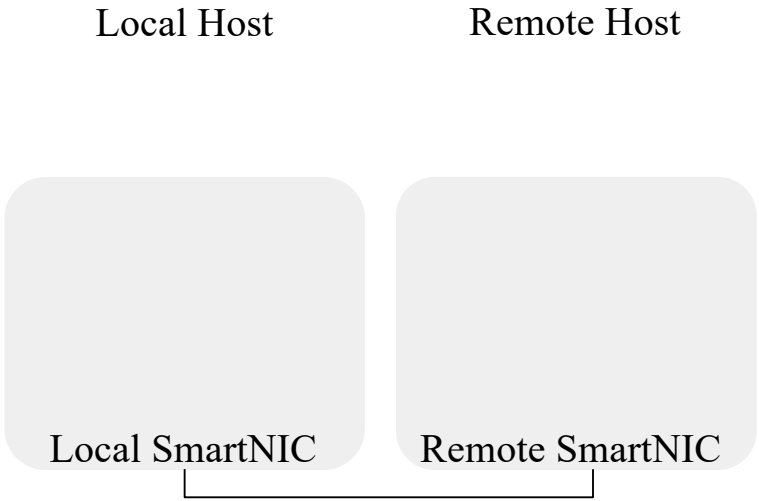
- *Warp Pipe:* Composition Unit for SG-IOV



Overview (2/2): Chainable Execution over Warp Pipes

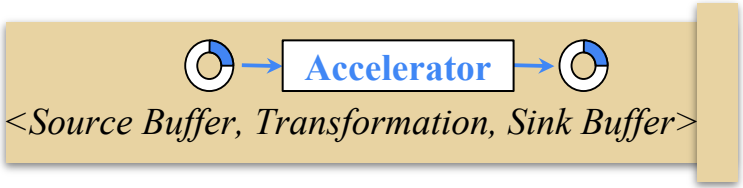
- *Warp Pipe:* Composition Unit for SG-IOV

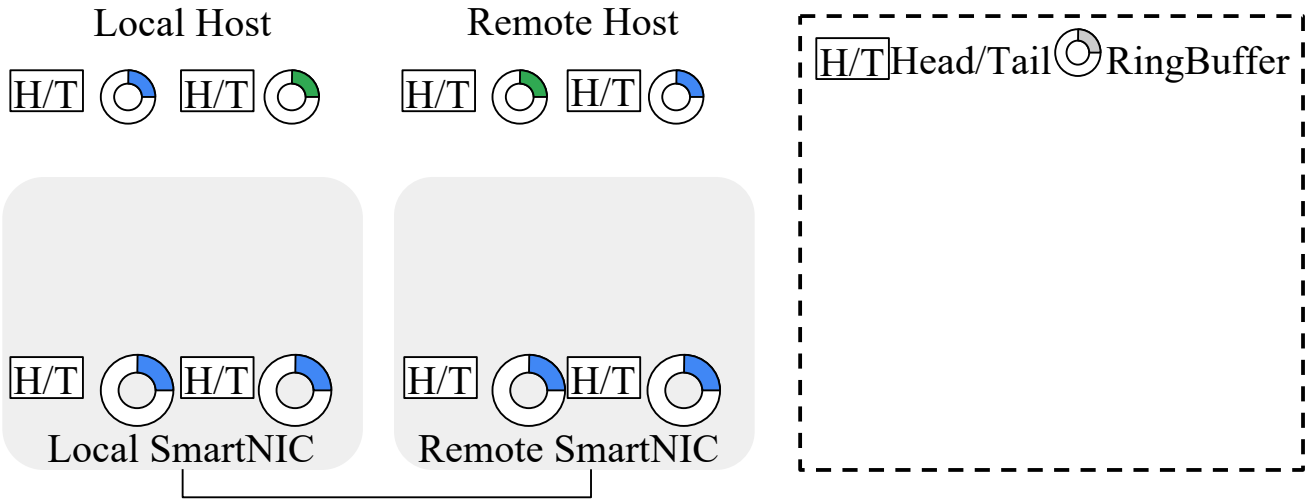




Overview (2/2): Chainable Execution over Warp Pipes

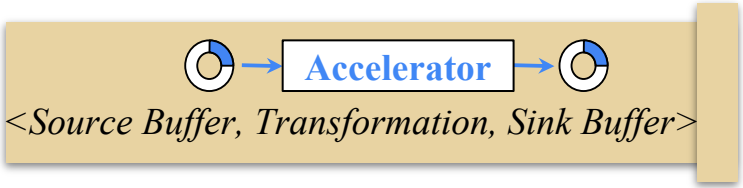
- *Warp Pipe:* Composition Unit for SG-IOV

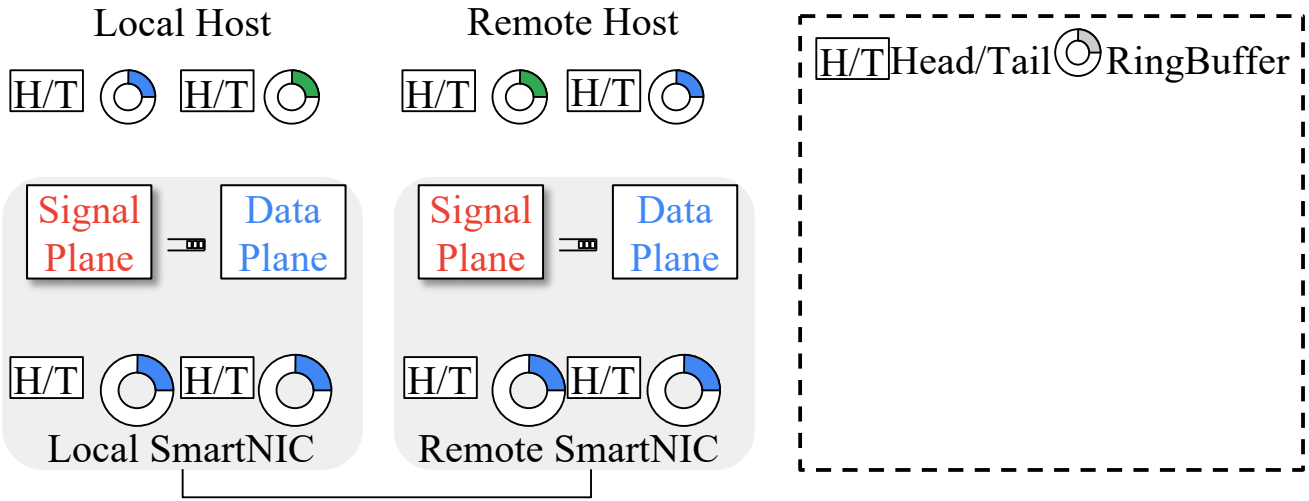




Overview (2/2): Chainable Execution over Warp Pipes

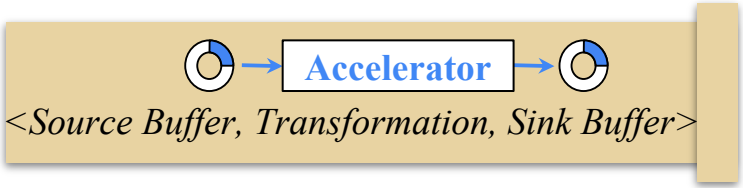
- *Warp Pipe:* Composition Unit for SG-IOV

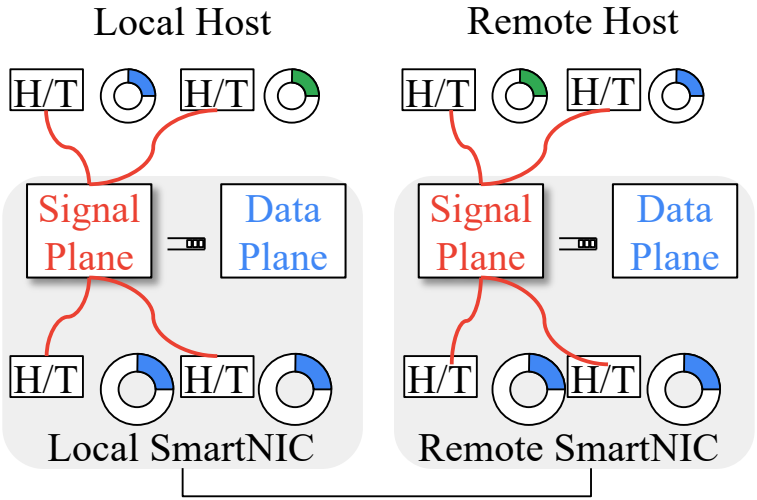


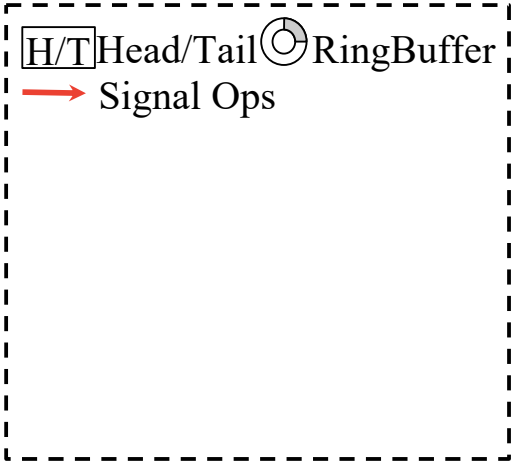


Overview (2/2): Chainable Execution over Warp Pipes

- *Warp Pipe:* Composition Unit for SG-IOV

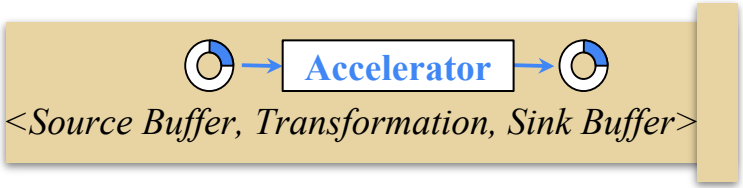


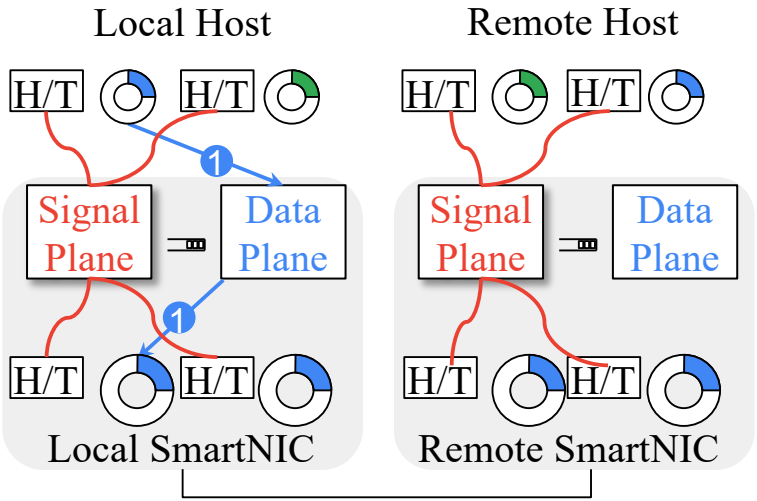


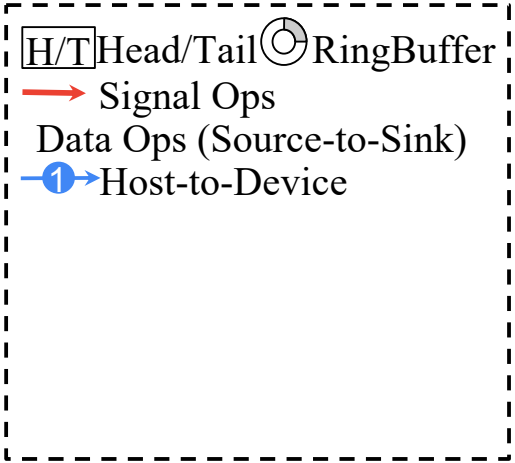


Overview (2/2): Chainable Execution over Warp Pipes

- *Warp Pipe:* Composition Unit for SG-IOV

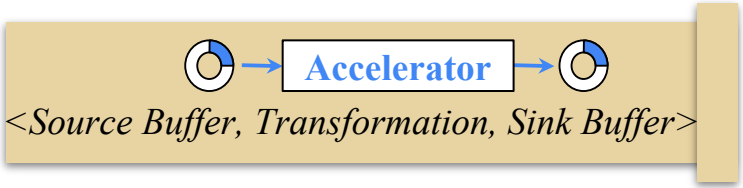


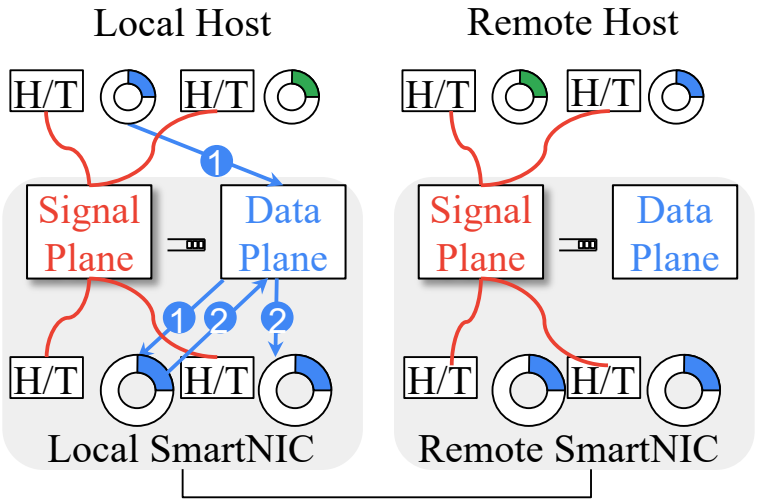


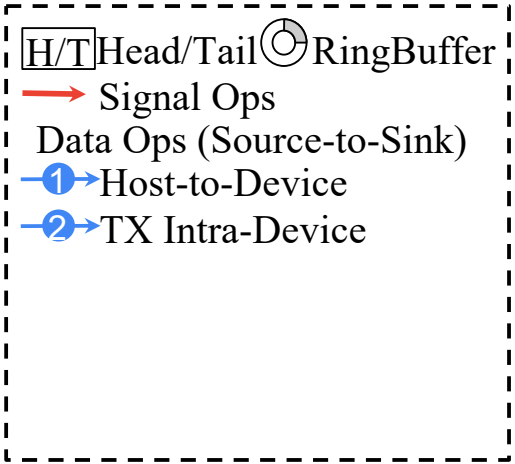


Overview (2/2): Chainable Execution over Warp Pipes

- *Warp Pipe:* Composition Unit for SG-IOV

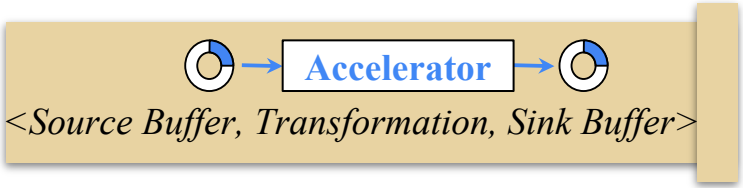


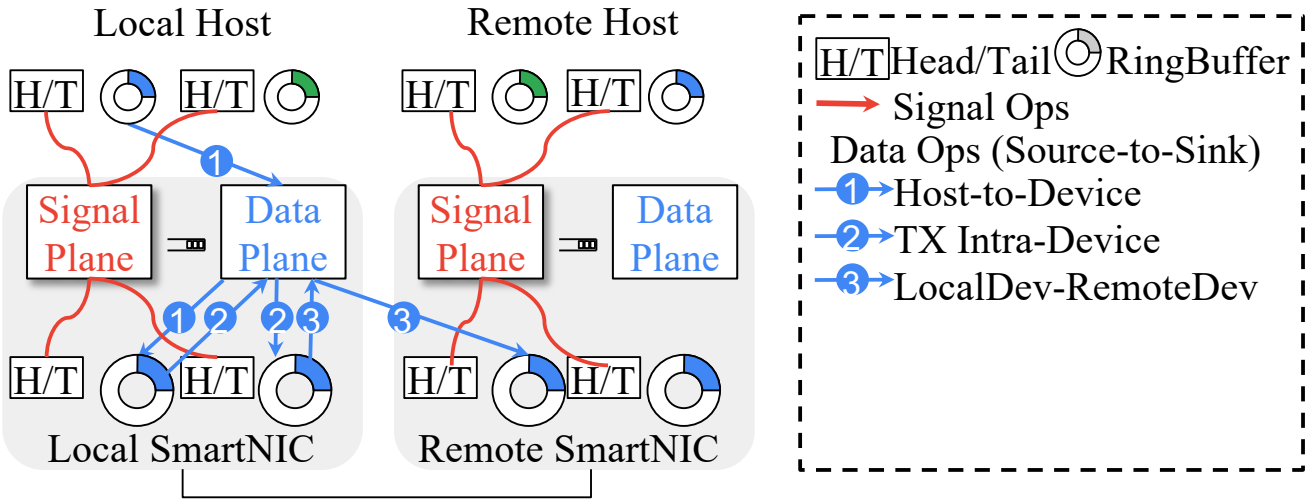




Overview (2/2): Chainable Execution over Warp Pipes

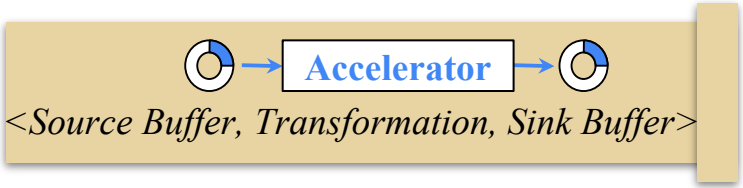
- *Warp Pipe:* Composition Unit for SG-IOV

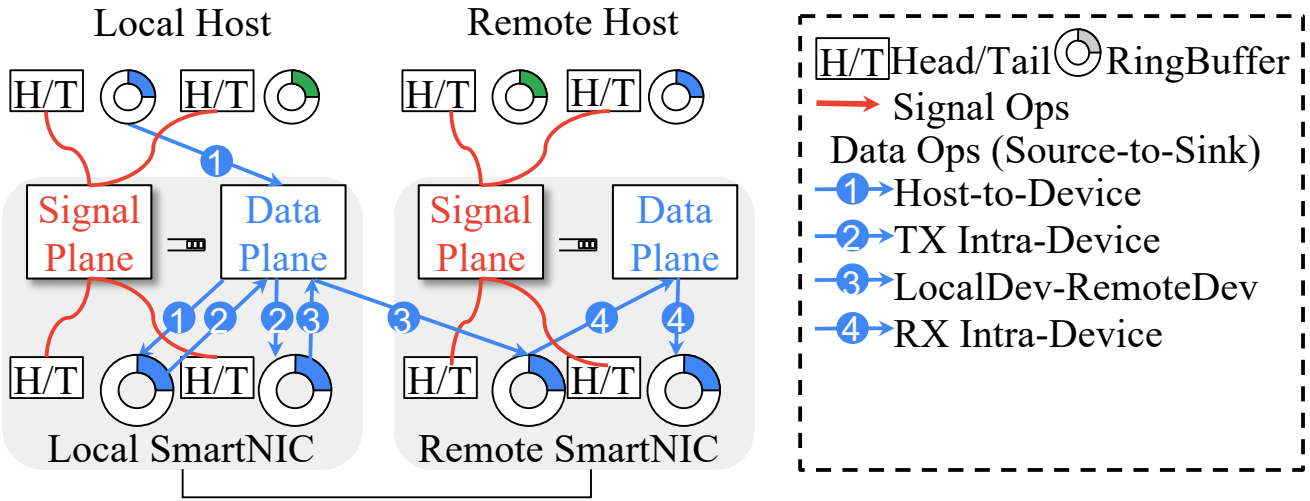




Overview (2/2): Chainable Execution over Warp Pipes

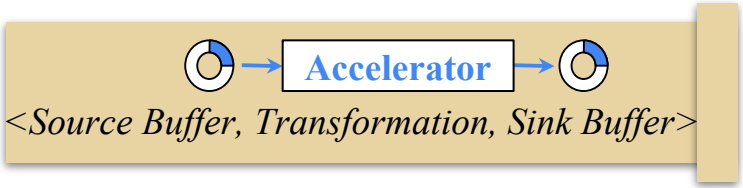
- *Warp Pipe:* Composition Unit for SG-IOV

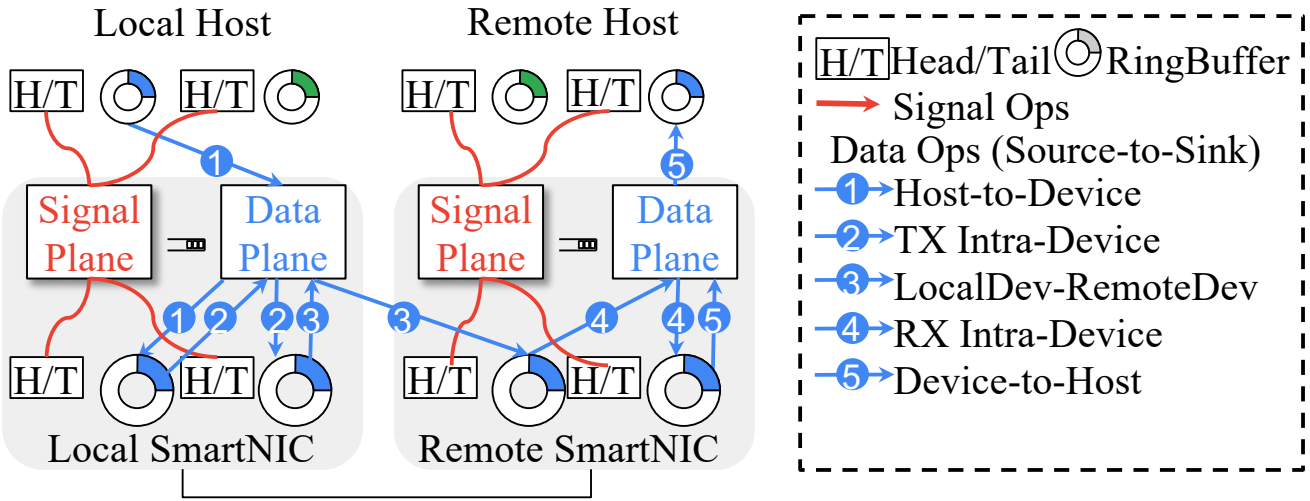




Overview (2/2): Chainable Execution over Warp Pipes

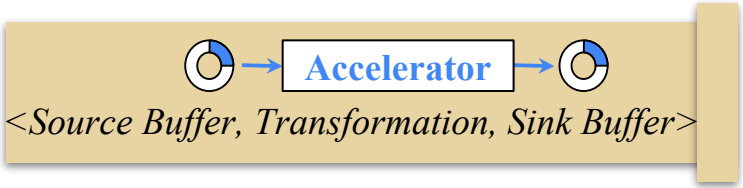
- *Warp Pipe:* Composition Unit for SG-IOV

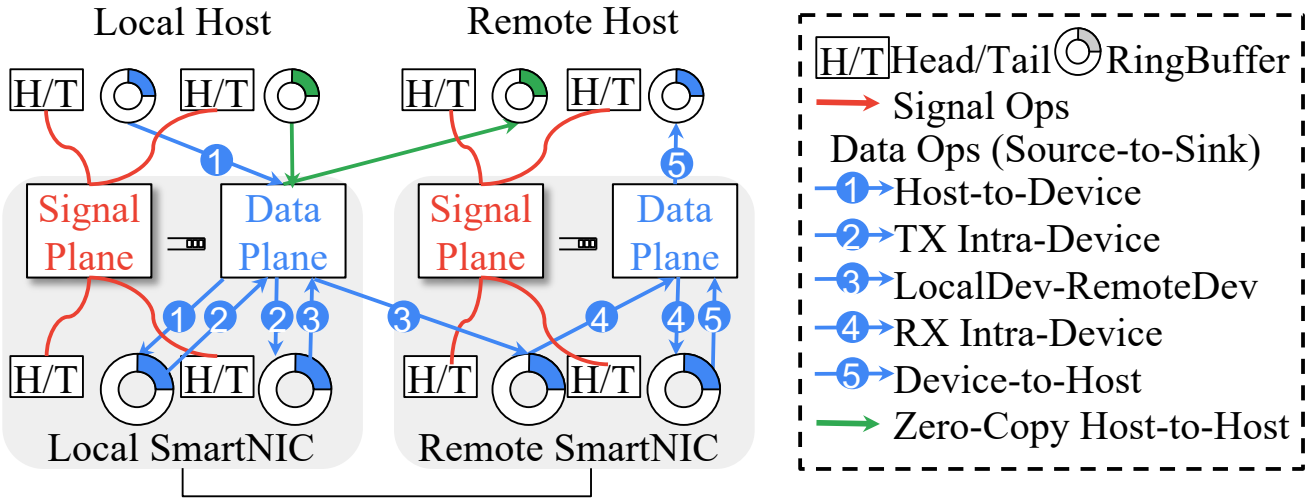




Overview (2/2): Chainable Execution over Warp Pipes

- *Warp Pipe:* Composition Unit for SG-IOV





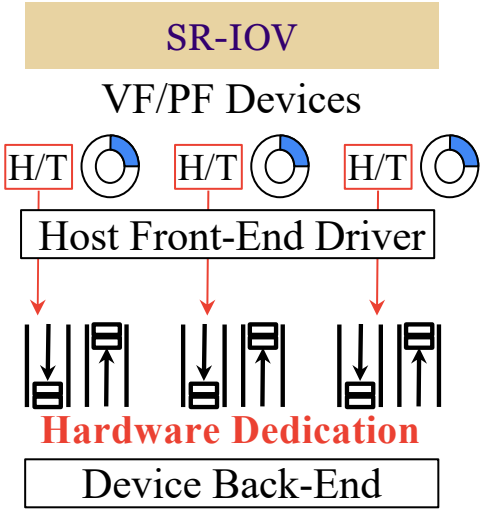
Technique #1: Scalable Signal Synchronization (1/2)

Technique #1: Scalable Signal Synchronization (1/2)

- How to support more devices?

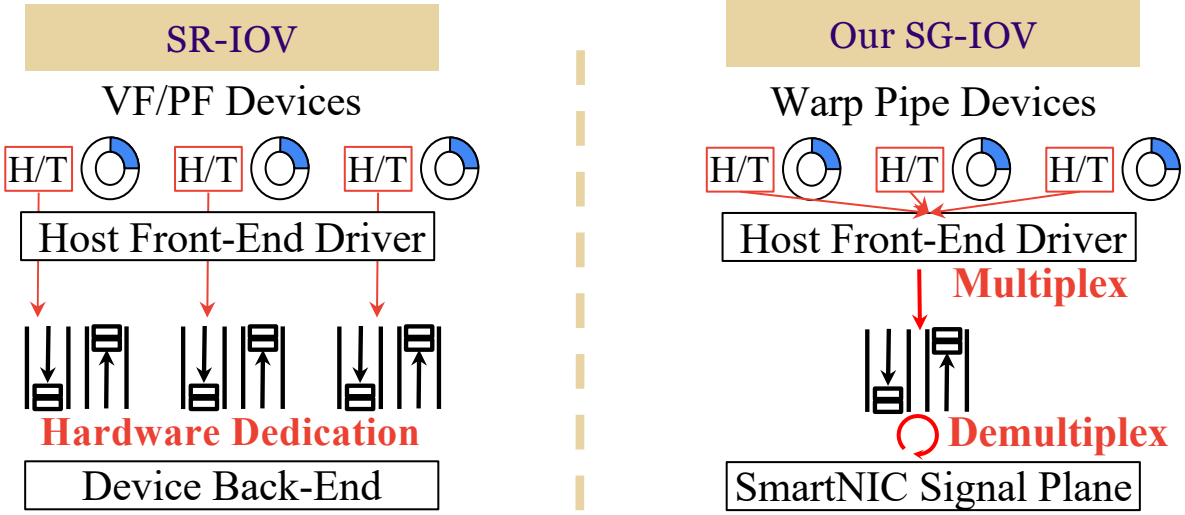
Technique #1: Scalable Signal Synchronization (1/2)

- How to support more devices?
  - SR-IOV: Device Count = Queue Pair Count



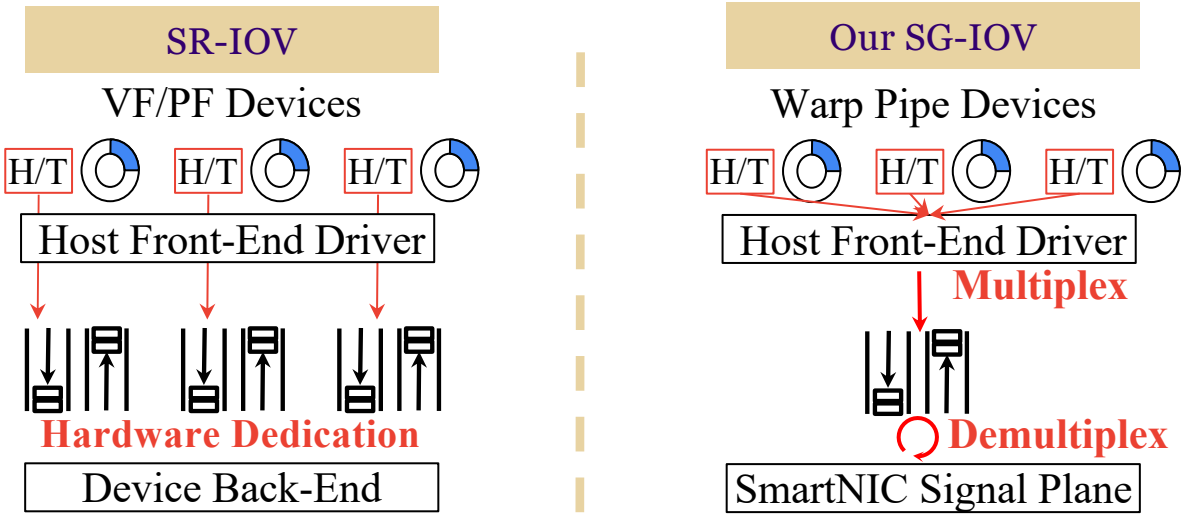
Technique #1: Scalable Signal Synchronization (1/2)

- How to support more devices?
  - SR-IOV: Device Count = Queue Pair Count
  - SG-IOV: Device Count ≫ Queue Pair Count



Technique #1: Scalable Signal Synchronization (1/2)

- How to support more devices?
  - SR-IOV: Device Count = Queue Pair Count
  - SG-IOV: Device Count ≫ Queue Pair Count



Approaching L5 socket scale, FIFO efficiency is crucial

Technique #1: Scalable Signal Synchronization (2/2)

Technique #1: Scalable Signal Synchronization (2/2)

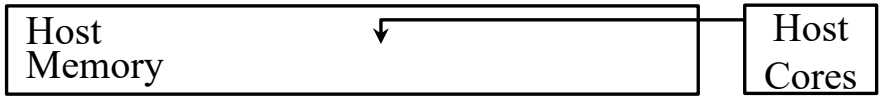
- *Cross-FIFO* Data Structure
  - Producer/Consumer Crossing PCIe

Technique #1: Scalable Signal Synchronization (2/2)

- *Cross-FIFO* Data Structure
  - Producer/Consumer Crossing PCIe
  - Hardware-Friendly Logic

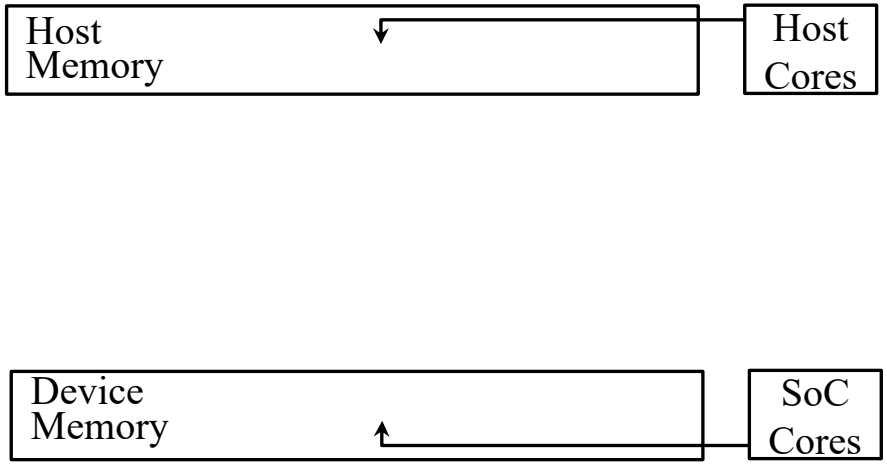
Technique #1: Scalable Signal Synchronization (2/2)

- *Cross-FIFO* Data Structure
  - Producer/Consumer Crossing PCIe
  - Hardware-Friendly Logic



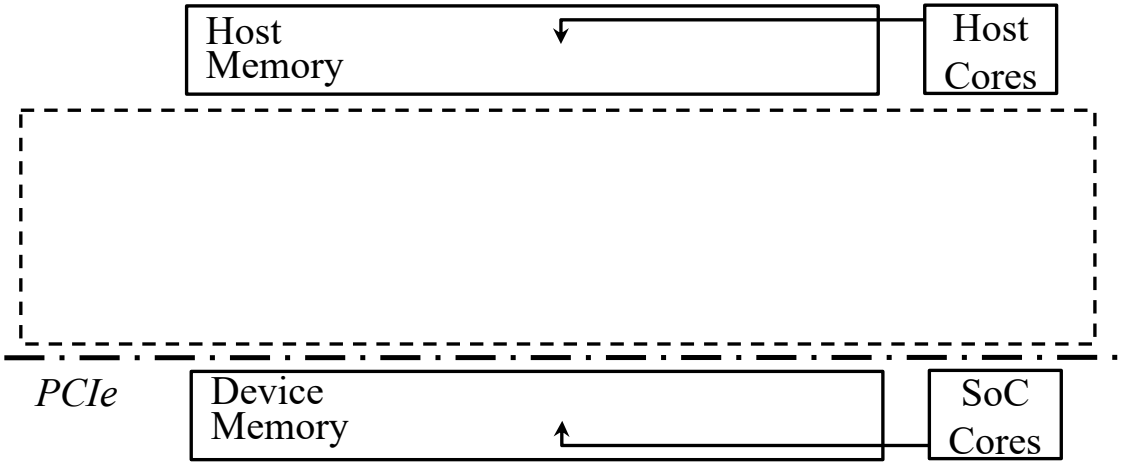
Technique #1: Scalable Signal Synchronization (2/2)

- *Cross-FIFO* Data Structure
  - Producer/Consumer Crossing PCIe
  - Hardware-Friendly Logic



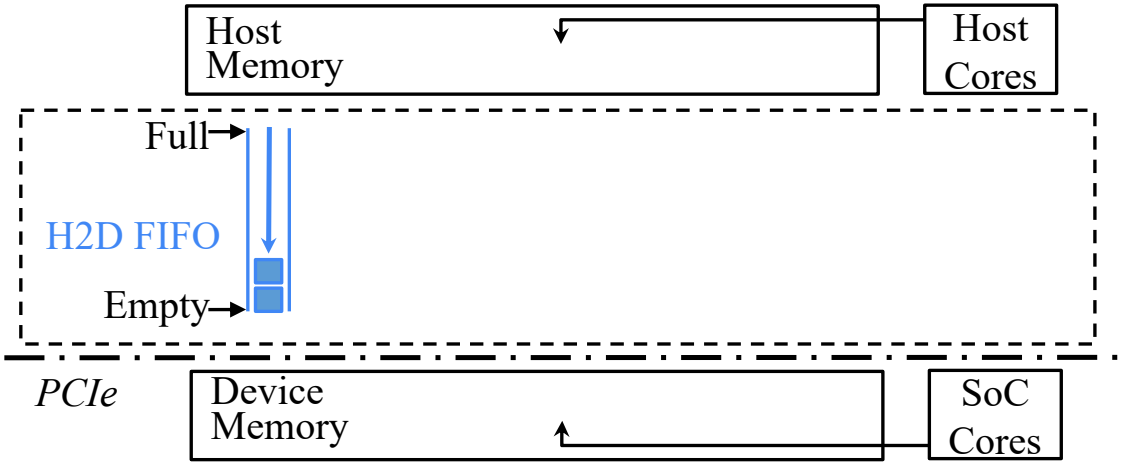
Technique #1: Scalable Signal Synchronization (2/2)

- *Cross-FIFO* Data Structure
  - Producer/Consumer Crossing PCIe
  - Hardware-Friendly Logic



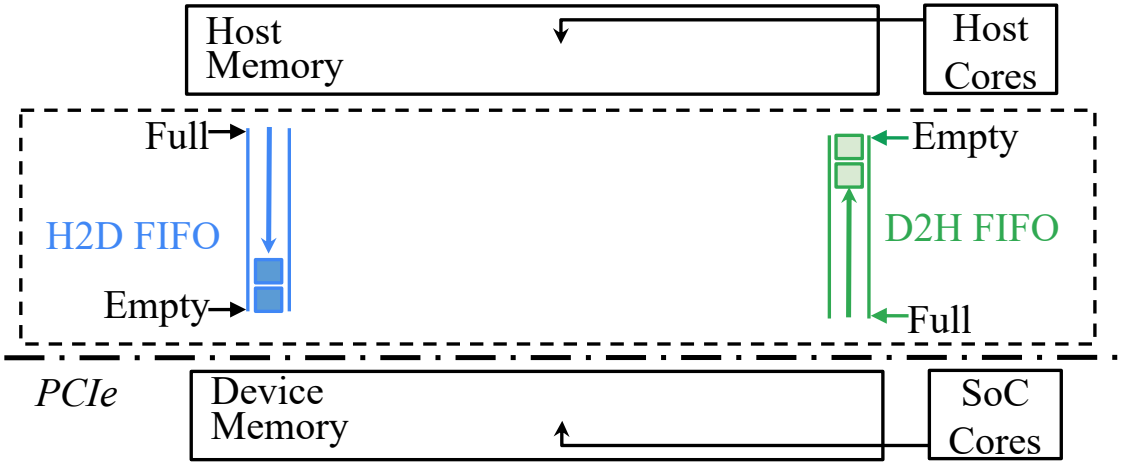
Technique #1: Scalable Signal Synchronization (2/2)

- *Cross-FIFO* Data Structure
  - Producer/Consumer Crossing PCIe
  - Hardware-Friendly Logic



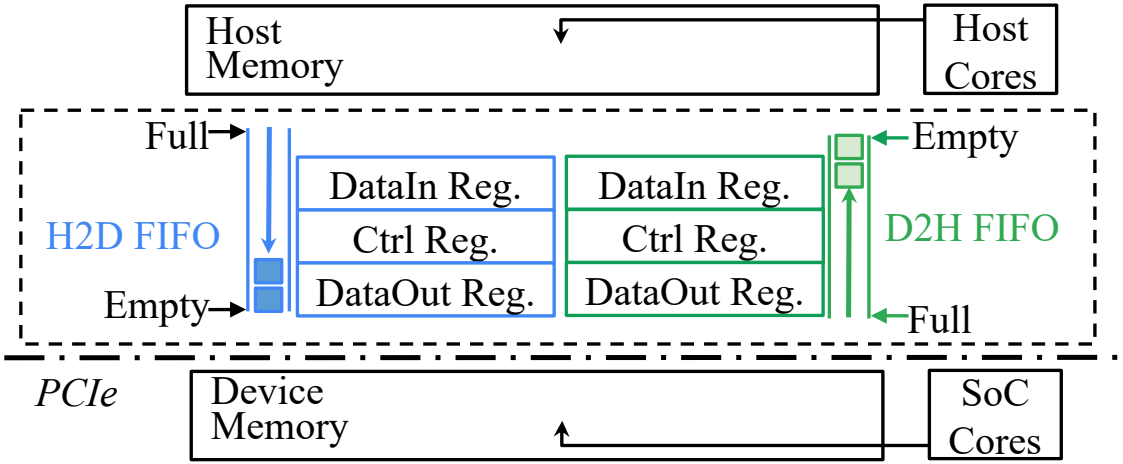
Technique #1: Scalable Signal Synchronization (2/2)

- *Cross-FIFO* Data Structure
  - Producer/Consumer Crossing PCIe
  - Hardware-Friendly Logic



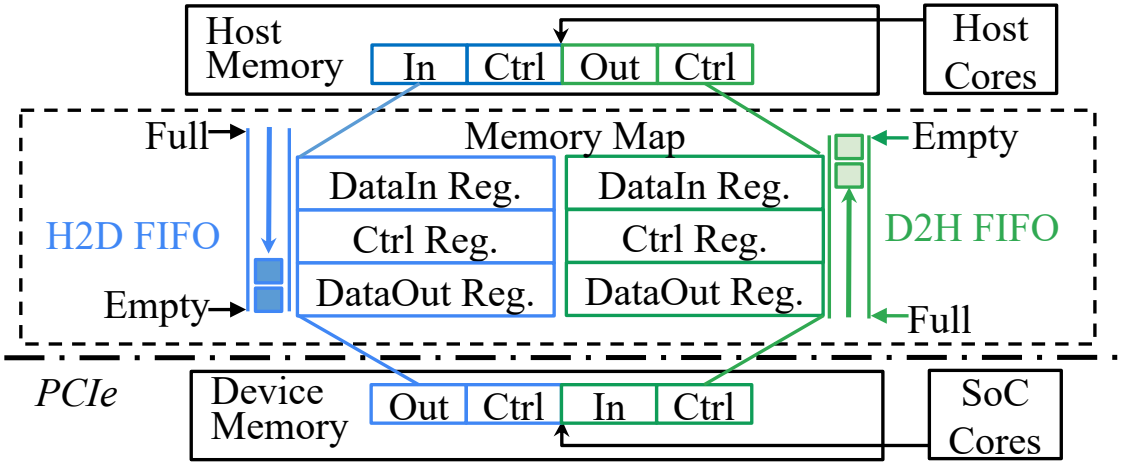
Technique #1: Scalable Signal Synchronization (2/2)

- *Cross-FIFO* Data Structure
  - Producer/Consumer Crossing PCIe
  - Hardware-Friendly Logic



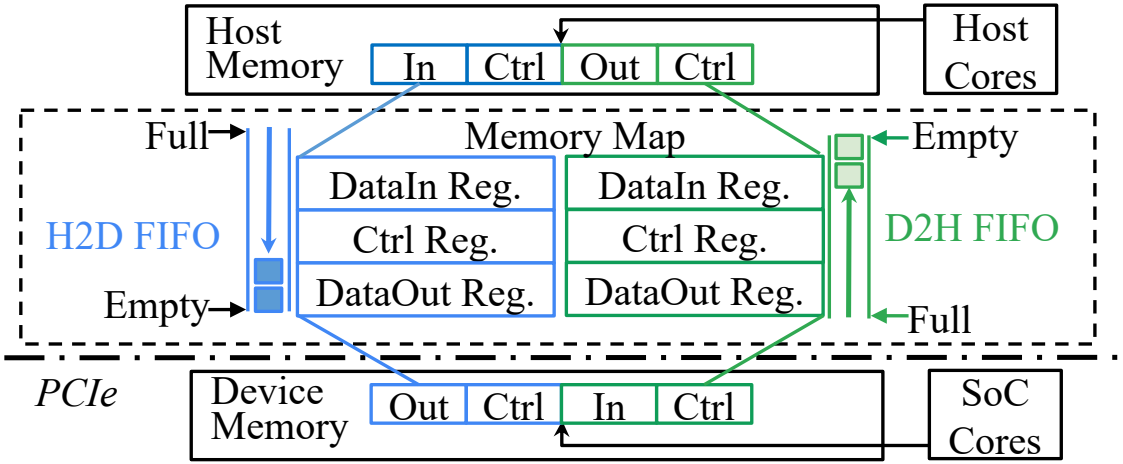
Technique #1: Scalable Signal Synchronization (2/2)

- *Cross-FIFO* Data Structure
  - Producer/Consumer Crossing PCIe
  - Hardware-Friendly Logic



Technique #1: Scalable Signal Synchronization (2/2)

- *Cross-FIFO* Data Structure
  - Producer/Consumer Crossing PCIe
  - Hardware-Friendly Logic



See the paper for detailed MMIO/RDMA primitives

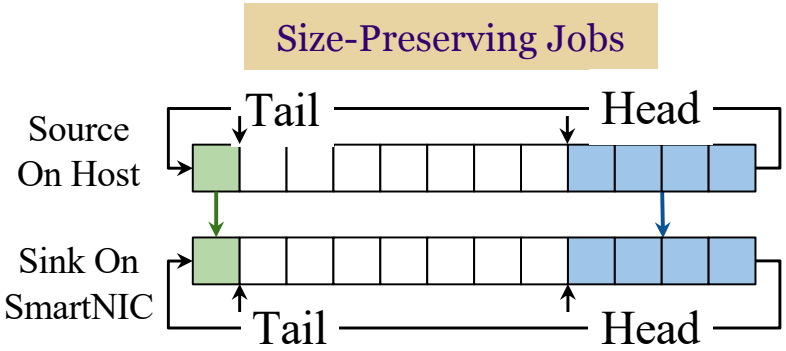
Technique #2: Flexible Job Generation

Technique #2: Flexible Job Generation

- Determine Source/Sink Chunks for Diverse Transformations (*e.g.*, DMA, Encrypt)

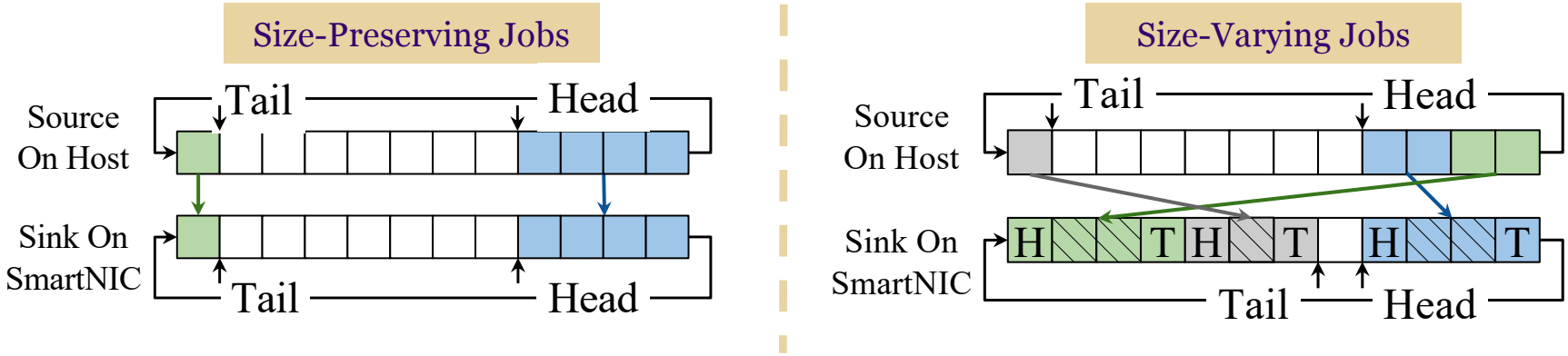
Technique #2: Flexible Job Generation

- Determine Source/Sink Chunks for Diverse Transformations (*e.g.*, DMA, Encrypt)



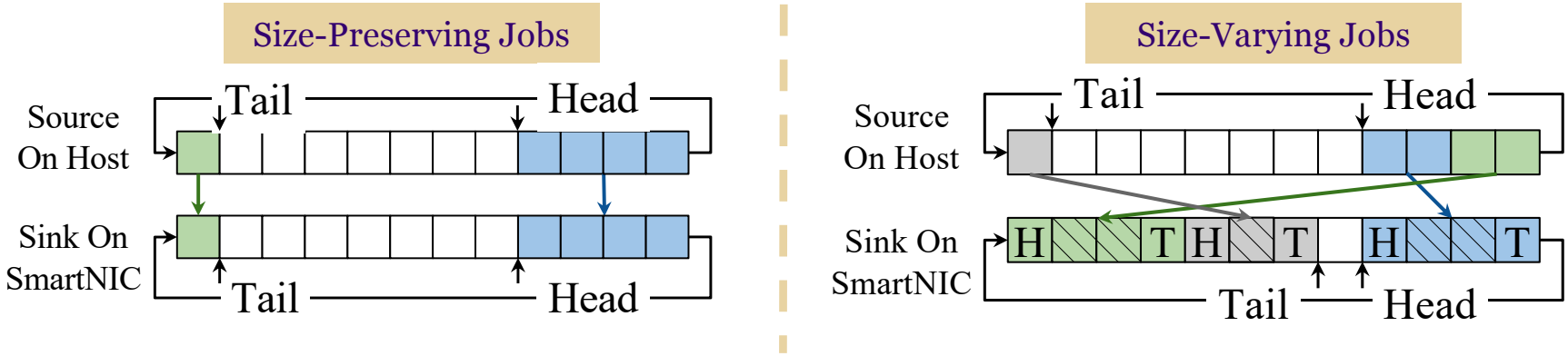
Technique #2: Flexible Job Generation

- Determine Source/Sink Chunks for Diverse Transformations (*e.g.*, DMA, Encrypt)
  - Size-varying message complicates buffer management
    - 2×2×2 = 8 cases: <source wrap or not × sink wrap or not × size change or not>



Technique #2: Flexible Job Generation

- Determine Source/Sink Chunks for Diverse Transformations (*e.g.*, DMA, Encrypt)
  - Size-varying message complicates buffer management
    - 2×2×2 = 8 cases: <source wrap or not × sink wrap or not × size change or not>



See the paper for the *Recursive Strategy* that covers all cases

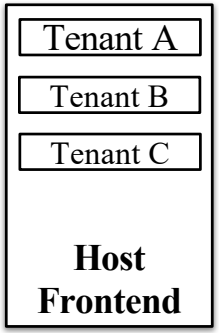
Technique #3: Fine-Grained HW Virtualization (1/2)

Technique #3: Fine-Grained HW Virtualization (1/2)

- Per-Tenant, Per-Accelerator Queuing System

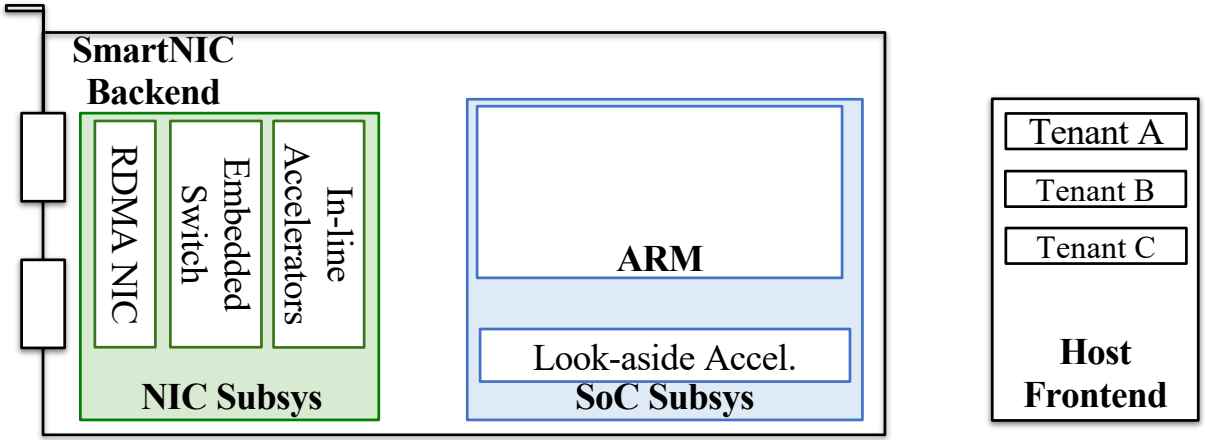
Technique #3: Fine-Grained HW Virtualization (1/2)

- Per-Tenant, Per-Accelerator Queuing System



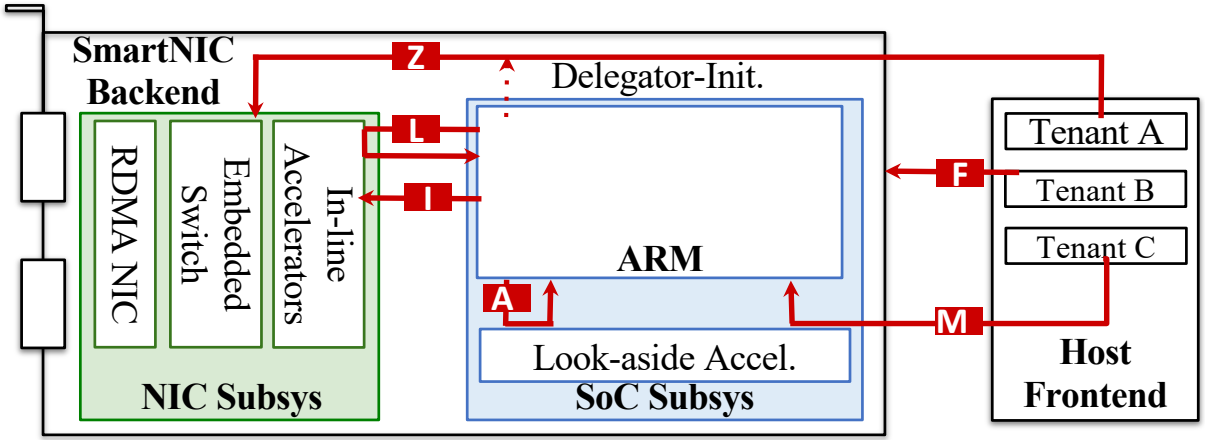
Technique #3: Fine-Grained HW Virtualization (1/2)

- Per-Tenant, Per-Accelerator Queuing System



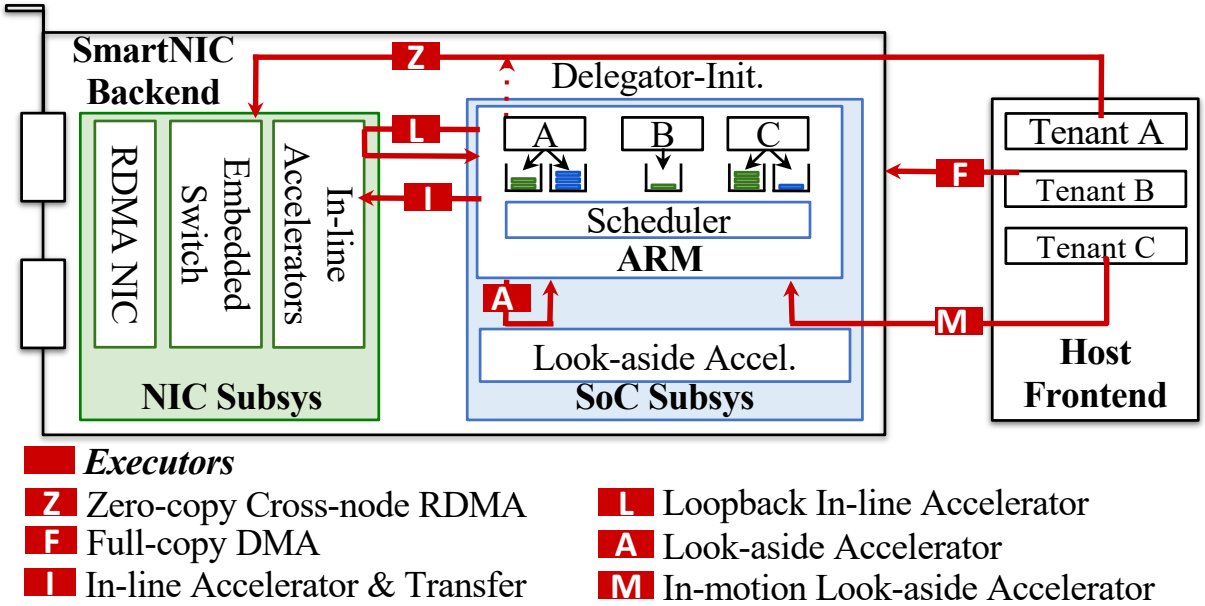
Technique #3: Fine-Grained HW Virtualization (1/2)

- Per-Tenant, Per-Accelerator Queuing System



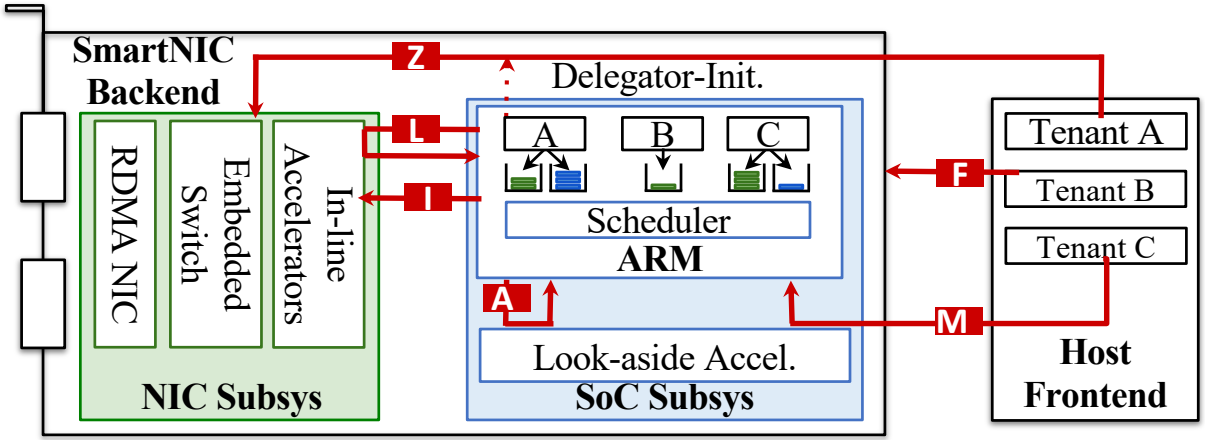
Technique #3: Fine-Grained HW Virtualization (1/2)

- Per-Tenant, Per-Accelerator Queuing System



Technique #3: Fine-Grained HW Virtualization (1/2)

- Per-Tenant, Per-Accelerator Queuing System



Next, how do we use such a queuing system to schedule jobs?

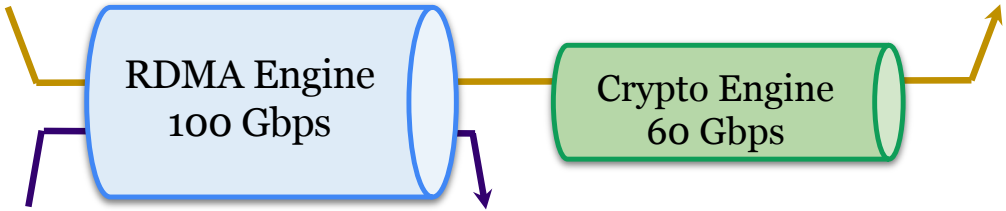
Technique #3: Fine-Grained HW Virtualization (2/2)

Technique #3: Fine-Grained HW Virtualization (2/2)

- Dominant Resource Fairness (DRF) Policy Specialized by SG-IOV

Technique #3: Fine-Grained HW Virtualization (2/2)

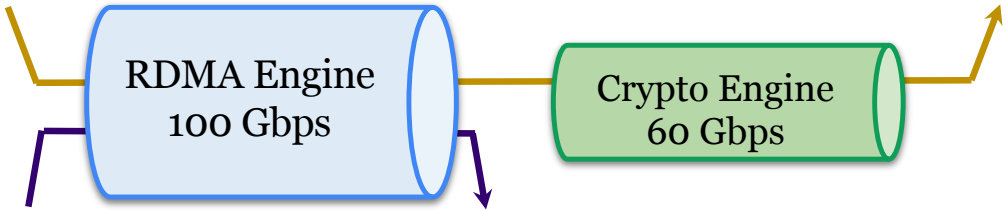
- Dominant Resource Fairness (DRF) Policy Specialized by SG-IOV



Dominant Resource
Tenant-A: {RDMA, Crypto}
Tenant-B: {RDMA}

Technique #3: Fine-Grained HW Virtualization (2/2)

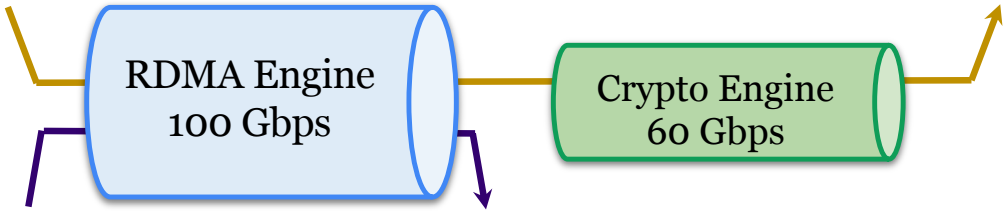
- Dominant Resource Fairness (DRF) Policy Specialized by SG-IOV
  - ***Equal-Bandwidth Demand*:** Multi-accelerator tenants consume the same BW on all accelerators



Dominant Resource
Tenant-A: {RDMA, Crypto}
Tenant-B: {RDMA}

Technique #3: Fine-Grained HW Virtualization (2/2)

- Dominant Resource Fairness (DRF) Policy Specialized by SG-IOV
  - ***Equal-Bandwidth Demand*:** Multi-accelerator tenants consume the same BW on all accelerators



DRF under Equal-Bandwidth Demand: *Consider K accelerators with processing bandwidths $C_1, \ldots, C_K$. Tenant t uses a subset $S_t \subseteq \{1, \ldots, K\}$ at equal bandwidth on each resource in S_t. For tenant t, let the bottleneck capacity be $Z_{S_t} \triangleq \min_{j \in S_t} C_j$.*

Per-tenant allocation (per used resource):

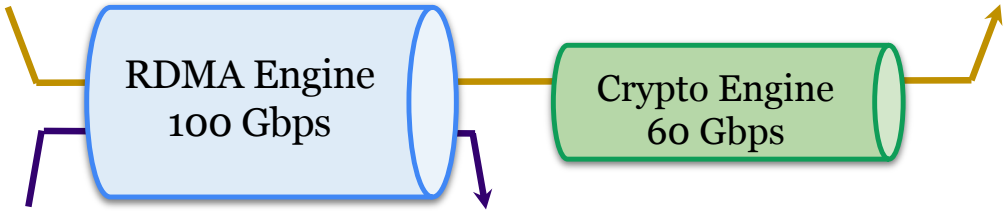
$$a_t = s^* \cdot Z_{S_t} \quad \text{where} \quad s^* = \min_j \frac{C_j}{\sum_{t: j \in S_t} Z_{S_t}},$$

where the sum $\sum_{t: j \in S_t} Z_{S_t}$ is taken over all tenants that use resource j, adding each tenant's bottleneck capacity Z_{S_t}.

Dominant Resource
Tenant-A: {RDMA, Crypto}
Tenant-B: {RDMA}

Technique #3: Fine-Grained HW Virtualization (2/2)

- Dominant Resource Fairness (DRF) Policy Specialized by SG-IOV
  - ***Equal-Bandwidth Demand*:** Multi-accelerator tenants consume the same BW on all accelerators



DRF under Equal-Bandwidth Demand: *Consider K accelerators with processing bandwidths $C_1, \ldots, C_K$. Tenant t uses a subset $S_t \subseteq \{1, \ldots, K\}$ at equal bandwidth on each resource in S_t. For tenant t, let the bottleneck capacity be $Z_{S_t} \triangleq \min_{j \in S_t} C_j$.*

Per-tenant allocation (per used resource):

$$a_t = s^* \cdot Z_{S_t} \quad \text{where} \quad s^* = \min_j \frac{C_j}{\sum_{t: j \in S_t} Z_{S_t}},$$

where the sum $\sum_{t: j \in S_t} Z_{S_t}$ is taken over all tenants that use resource j, adding each tenant's bottleneck capacity Z_{S_t}.

Dominant Resource

Tenant-A: {RDMA, Crypto}

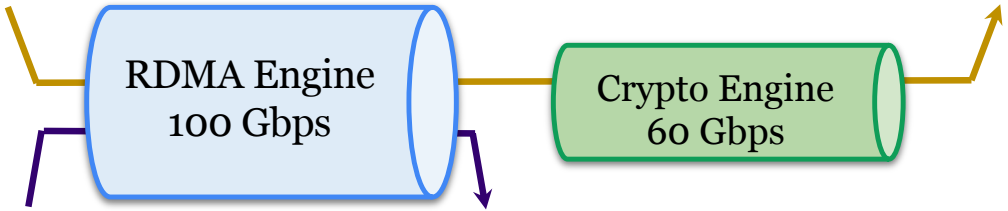
Tenant-B: {RDMA}

Dominant Share

$$s^* = \min\left(\frac{100}{100+60}, \frac{60}{60}\right) = 0.625$$

Technique #3: Fine-Grained HW Virtualization (2/2)

- Dominant Resource Fairness (DRF) Policy Specialized by SG-IOV
  - ***Equal-Bandwidth Demand*:** Multi-accelerator tenants consume the same BW on all accelerators



DRF under Equal-Bandwidth Demand: *Consider K accelerators with processing bandwidths $C_1, \ldots, C_K$. Tenant t uses a subset $S_t \subseteq \{1, \ldots, K\}$ at equal bandwidth on each resource in S_t. For tenant t, let the bottleneck capacity be $Z_{S_t} \triangleq \min_{j \in S_t} C_j$.*

Per-tenant allocation (per used resource):

$$a_t = s^* \cdot Z_{S_t} \quad \text{where} \quad s^* = \min_j \frac{C_j}{\sum_{t: j \in S_t} Z_{S_t}},$$

where the sum $\sum_{t: j \in S_t} Z_{S_t}$ is taken over all tenants that use resource j, adding each tenant's bottleneck capacity Z_{S_t}.

Dominant Resource

Tenant-A: {RDMA, Crypto}

Tenant-B: {RDMA}

Dominant Share

$$s^* = \min\left(\frac{100}{100+60}, \frac{60}{60}\right) = 0.625$$

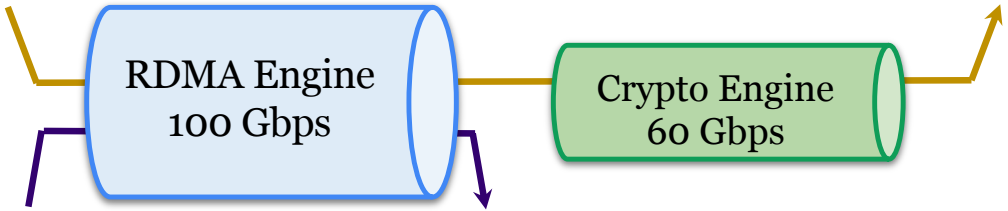
Bandwidth Allocation

Tenant-A: 0.625 x 60 = 37.5 Gbps

Tenant-B: 0.625 x 100 = 62.5 Gbps

Technique #3: Fine-Grained HW Virtualization (2/2)

- Dominant Resource Fairness (DRF) Policy Specialized by SG-IOV
  - ***Equal-Bandwidth Demand*:** Multi-accelerator tenants consume the same BW on all accelerators



DRF under Equal-Bandwidth Demand: *Consider K accelerators with processing bandwidths $C_1, \ldots, C_K$. Tenant t uses a subset $S_t \subseteq \{1, \ldots, K\}$ at equal bandwidth on each resource in S_t. For tenant t, let the bottleneck capacity be $Z_{S_t} \triangleq \min_{j \in S_t} C_j$.*

Per-tenant allocation (per used resource):

$$a_t = s^* \cdot Z_{S_t} \quad \text{where} \quad s^* = \min_j \frac{C_j}{\sum_{t: j \in S_t} Z_{S_t}},$$

where the sum $\sum_{t: j \in S_t} Z_{S_t}$ is taken over all tenants that use resource j, adding each tenant's bottleneck capacity Z_{S_t}.

Dominant Resource

Tenant-A: {RDMA, Crypto}

Tenant-B: {RDMA}

Dominant Share

$$s^* = \min\left(\frac{100}{100+60}, \frac{60}{60}\right) = 0.625$$

Bandwidth Allocation

Tenant-A: 0.625 x 60 = 37.5 Gbps

Tenant-B: 0.625 x 100 = 62.5 Gbps

See the paper for the formal mathematical analysis

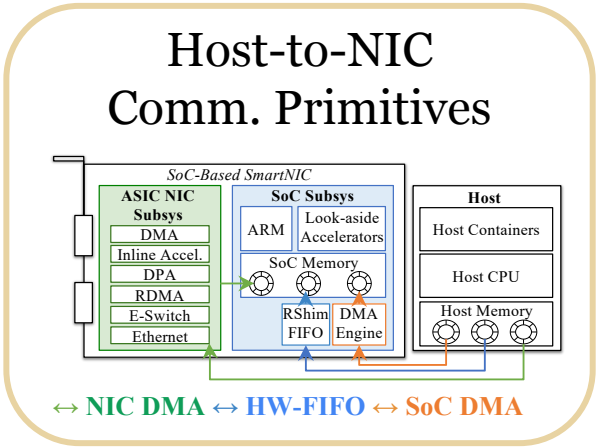
Implementation (1/2): SG-IOV Device

Implementation (1/2): SG-IOV Device

- Prototype SG-IOV with NVIDIA BlueField-3

Implementation (1/2): SG-IOV Device

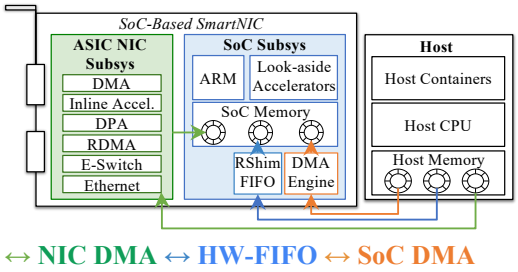
- Prototype SG-IOV with NVIDIA BlueField-3



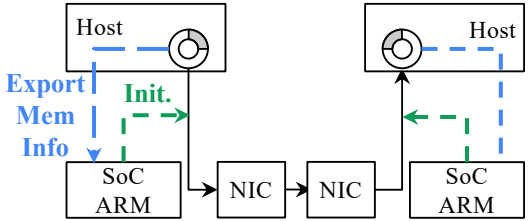
Implementation (1/2): SG-IOV Device

- Prototype SG-IOV with NVIDIA BlueField-3

Host-to-NIC Comm. Primitives



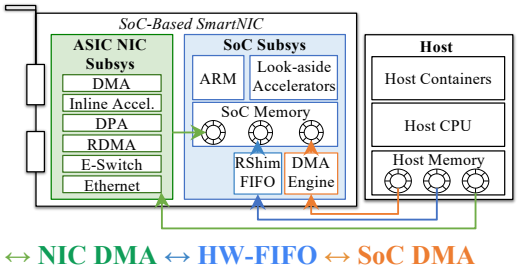
SoC-Delegated Zero-Copy



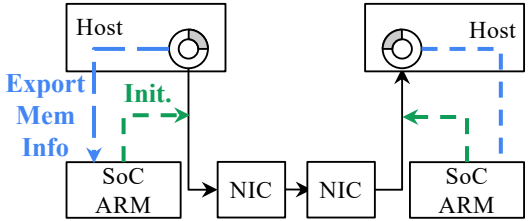
Implementation (1/2): SG-IOV Device

- Prototype SG-IOV with NVIDIA BlueField-3

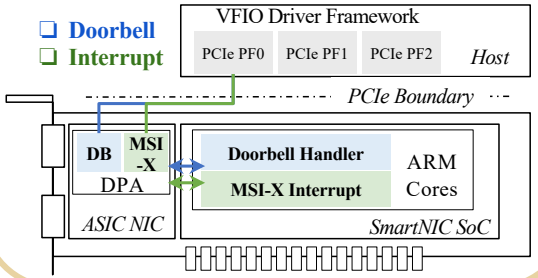
Host-to-NIC Comm. Primitives



SoC-Delegated Zero-Copy



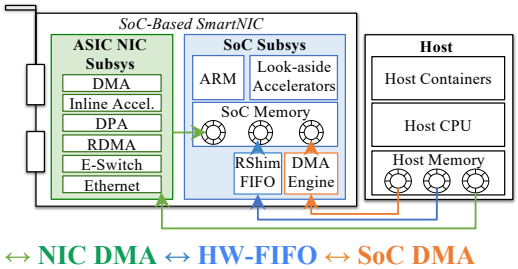
PCIe Device Emulation



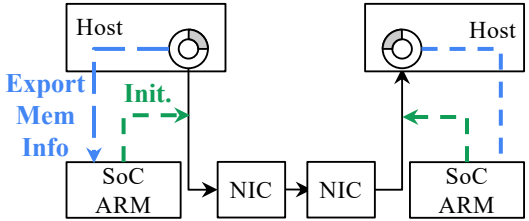
Implementation (1/2): SG-IOV Device

- Prototype SG-IOV with NVIDIA BlueField-3

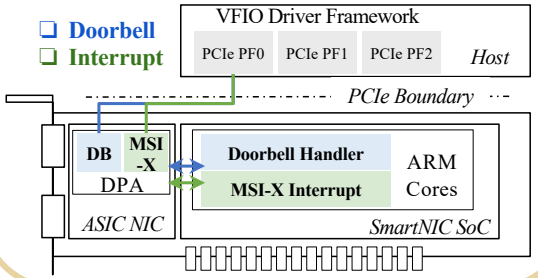
Host-to-NIC Comm. Primitives



SoC-Delegated Zero-Copy



PCIe Device Emulation



See the paper for detailed characterization

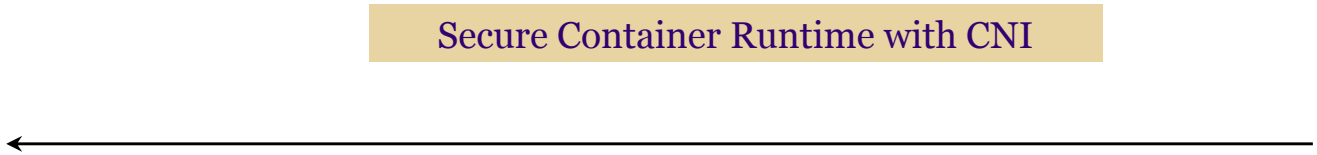
Implementation (2/2): SG-IOV Container Network Intf.

Implementation (2/2): SG-IOV Container Network Intf.

- SGIOV-CNI Complying with Secure Container Runtime

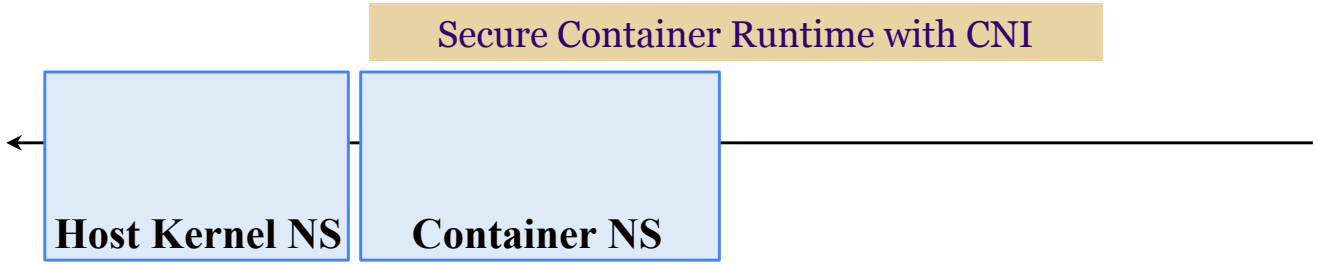
Implementation (2/2): SG-IOV Container Network Intf.

- SGIOV-CNI Complying with Secure Container Runtime



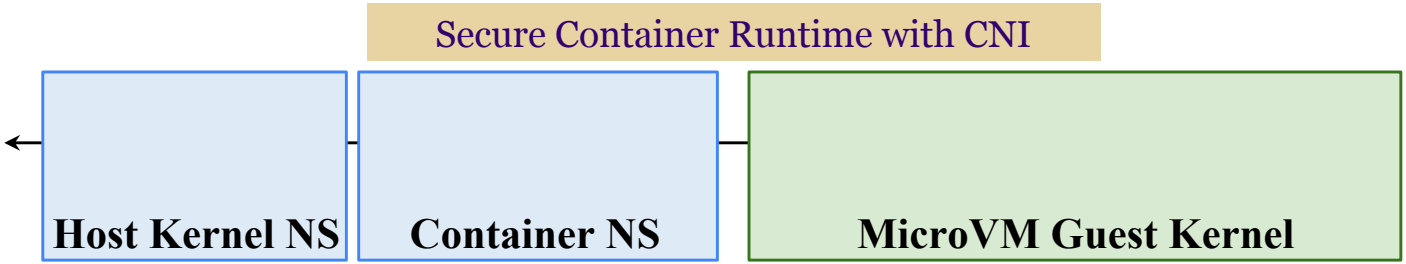
Implementation (2/2): SG-IOV Container Network Intf.

- SGIOV-CNI Complying with Secure Container Runtime



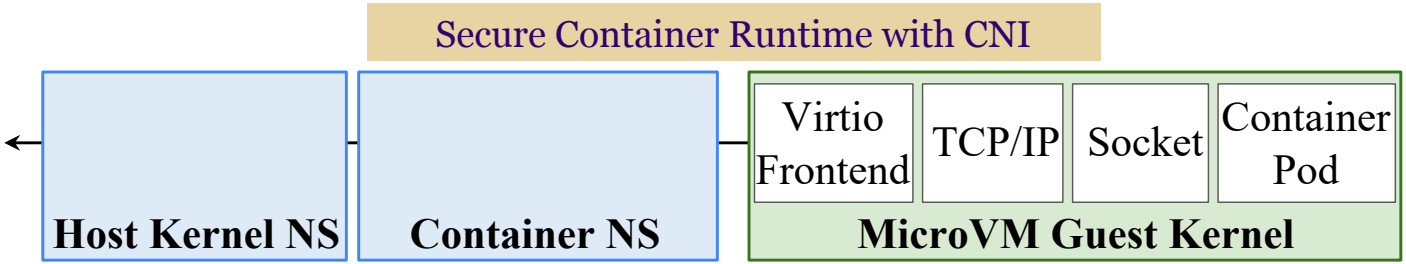
Implementation (2/2): SG-IOV Container Network Intf.

- SGIOV-CNI Complying with Secure Container Runtime



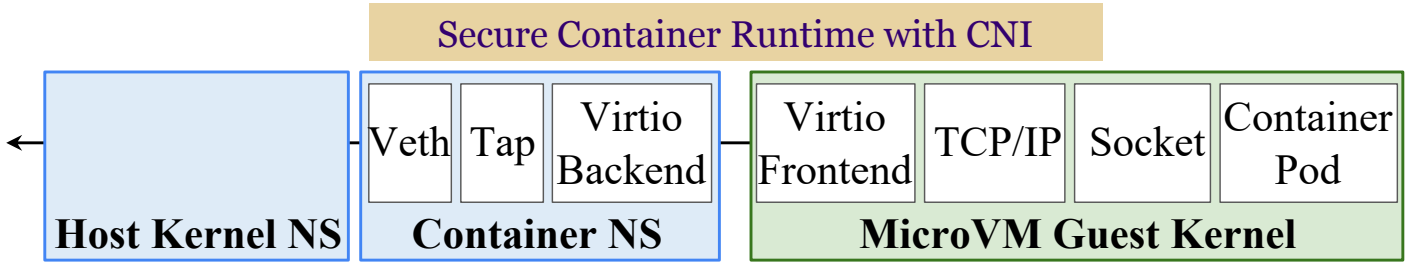
Implementation (2/2): SG-IOV Container Network Intf.

- SGIOV-CNI Complying with Secure Container Runtime



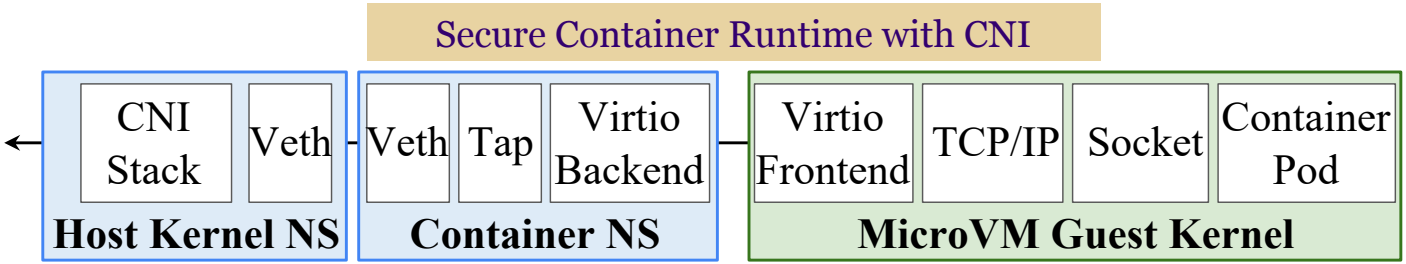
Implementation (2/2): SG-IOV Container Network Intf.

- SGIOV-CNI Complying with Secure Container Runtime



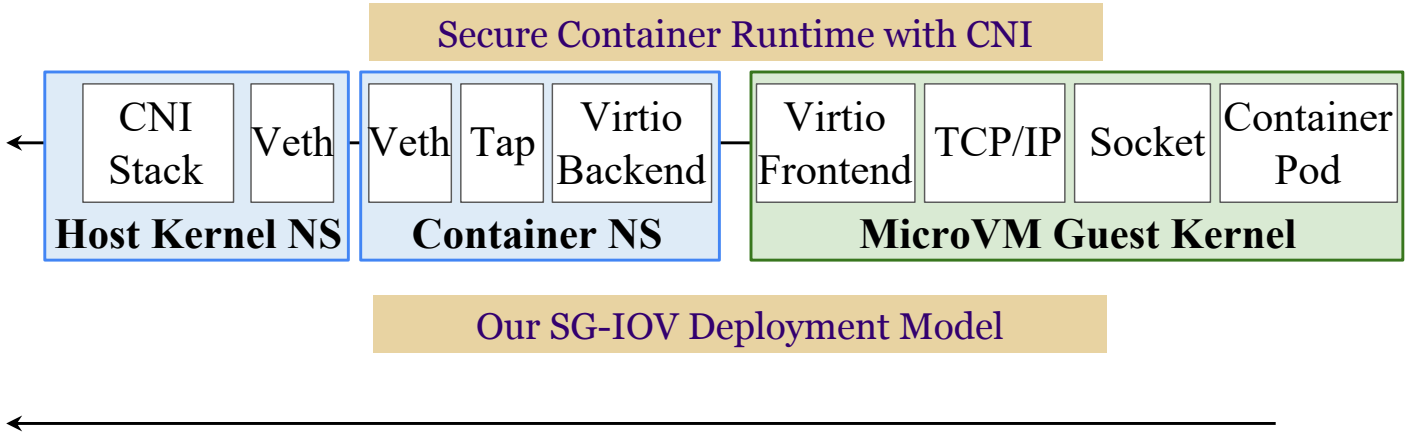
Implementation (2/2): SG-IOV Container Network Intf.

- SGIOV-CNI Complying with Secure Container Runtime



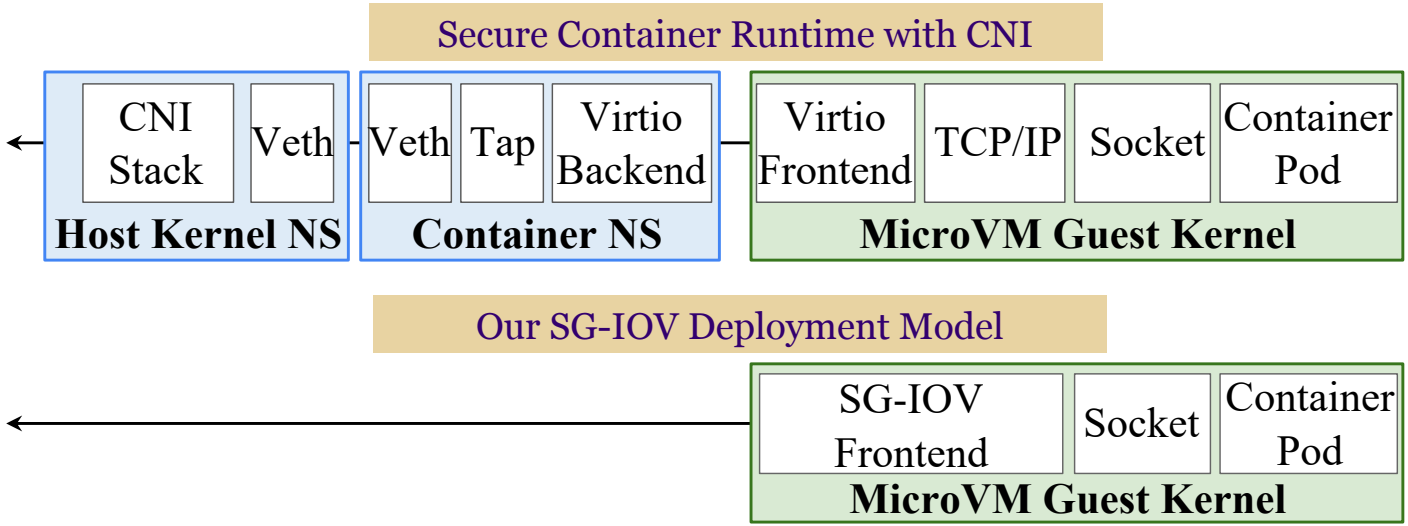
Implementation (2/2): SG-IOV Container Network Intf.

- SGIOV-CNI Complying with Secure Container Runtime



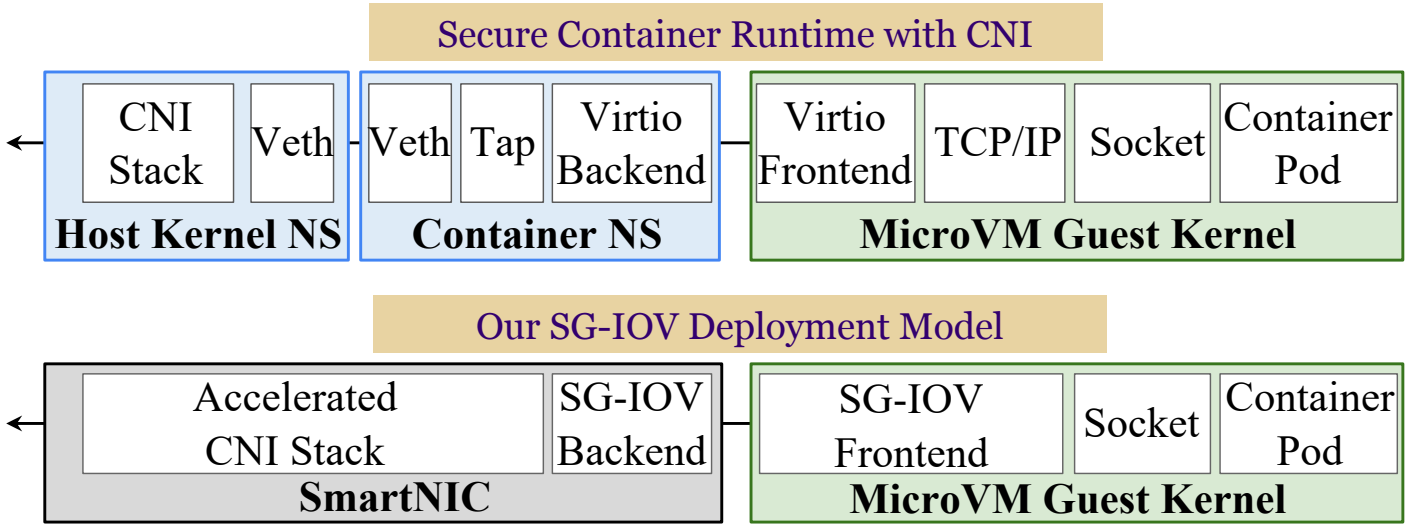
Implementation (2/2): SG-IOV Container Network Intf.

- SGIOV-CNI Complying with Secure Container Runtime



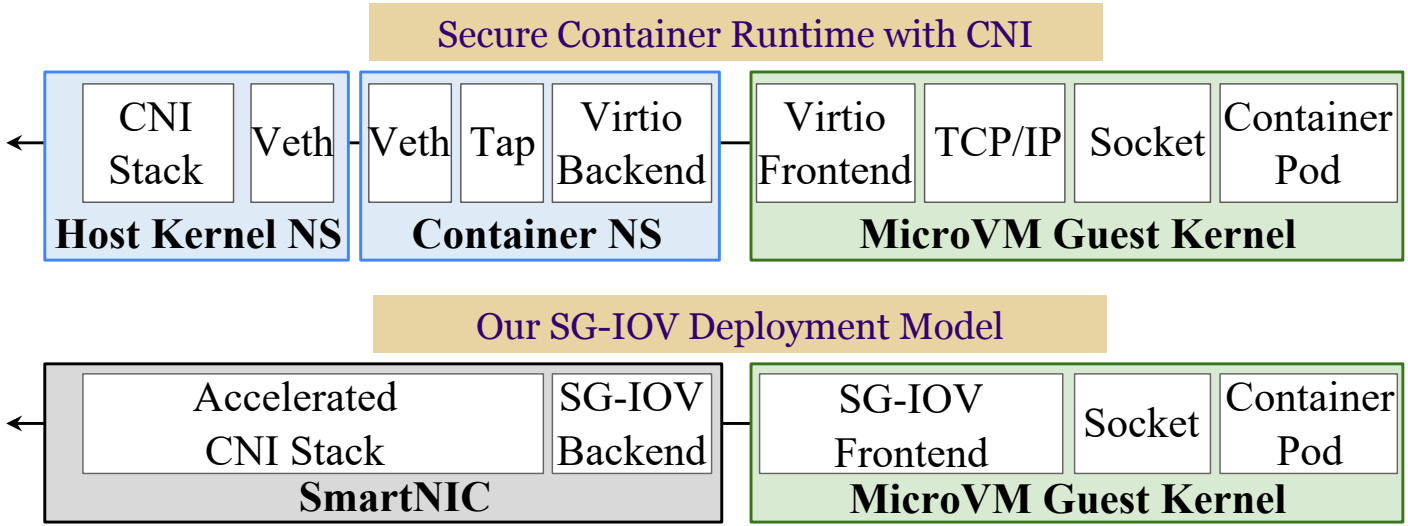
Implementation (2/2): SG-IOV Container Network Intf.

- SGIOV-CNI Complying with Secure Container Runtime



Implementation (2/2): SG-IOV Container Network Intf.

- SGIOV-CNI Complying with Secure Container Runtime



See the paper for SGIOV-CNI support for traditional containers

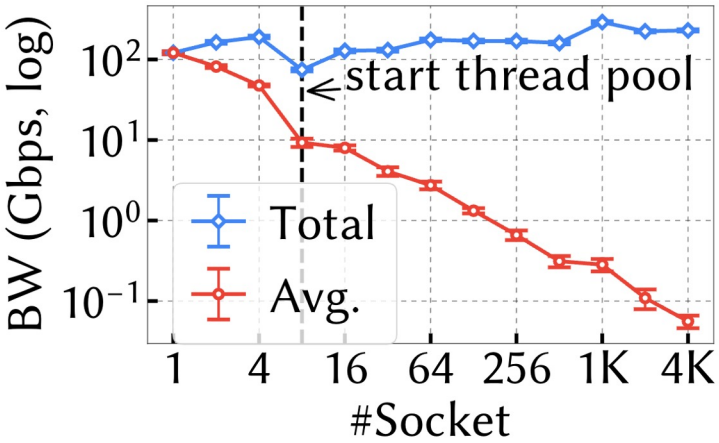
Evaluation: Micro-Benchmarking SG-IOV

Evaluation: Micro-Benchmarking SG-IOV

- Test setup
  - Two connected servers (NVIDIA BlueField-3 with a 100GbE switch)
  - Standard POSIX send()/recv() over Warp Pipes
  - 128KB messages for bandwidth; 4KB messages for latency

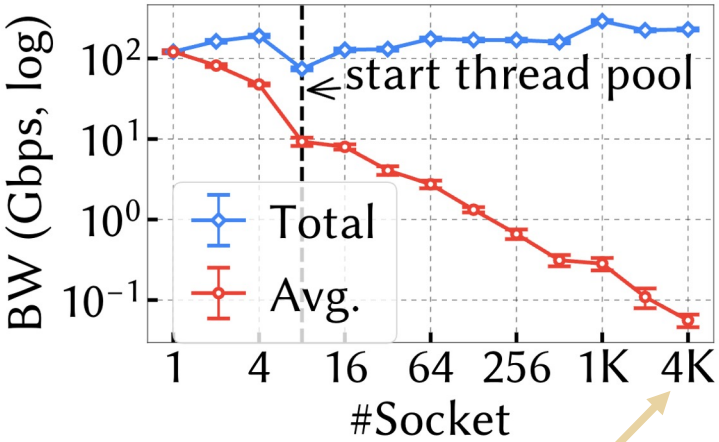
Evaluation: Micro-Benchmarking SG-IOV

- Test setup
  - Two connected servers (NVIDIA BlueField-3 with a 100GbE switch)
  - Standard POSIX send()/recv() over Warp Pipes
  - 128KB messages for bandwidth; 4KB messages for latency



Evaluation: Micro-Benchmarking SG-IOV

- Test setup
  - Two connected servers (NVIDIA BlueField-3 with a 100GbE switch)
  - Standard POSIX send()/recv() over Warp Pipes
  - 128KB messages for bandwidth; 4KB messages for latency

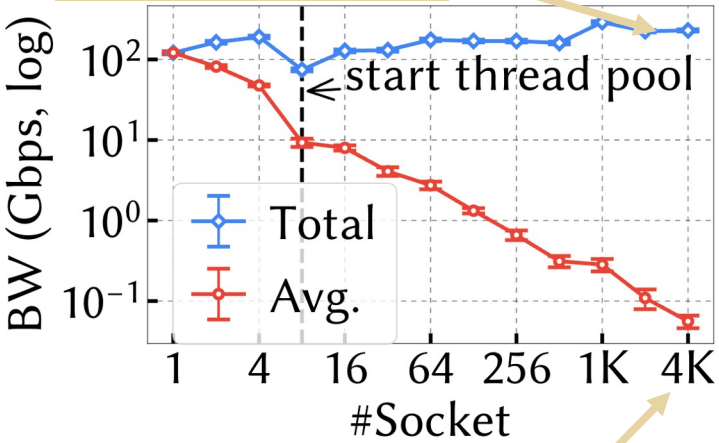


Scales up to 4K devices, approaching socket scale

Evaluation: Micro-Benchmarking SG-IOV

- Test setup
  - Two connected servers (NVIDIA BlueField-3 with a 100GbE switch)
  - Standard POSIX send()/recv() over Warp Pipes
  - 128KB messages for bandwidth; 4KB messages for latency

Sustaining line rate

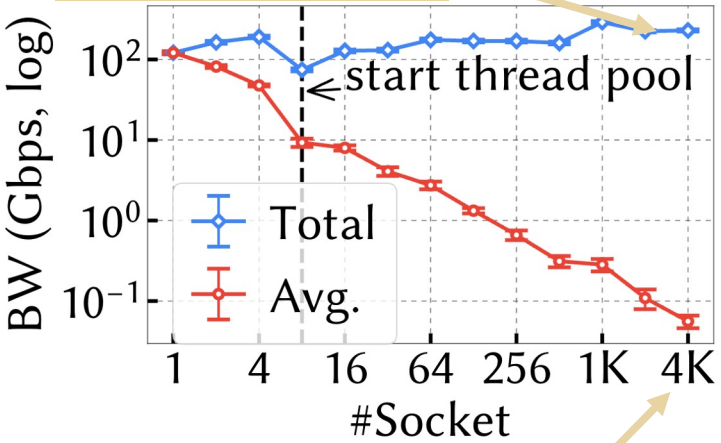


Scales up to 4K devices, approaching socket scale

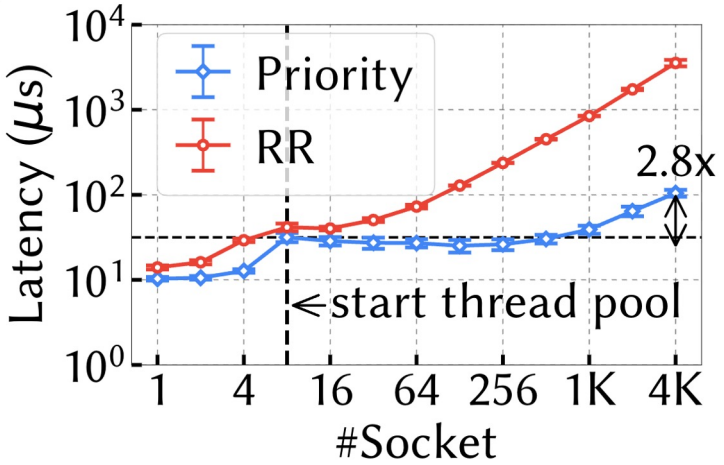
Evaluation: Micro-Benchmarking SG-IOV

- Test setup
  - Two connected servers (NVIDIA BlueField-3 with a 100GbE switch)
  - Standard POSIX send()/recv() over Warp Pipes
  - 128KB messages for bandwidth; 4KB messages for latency

Sustaining line rate



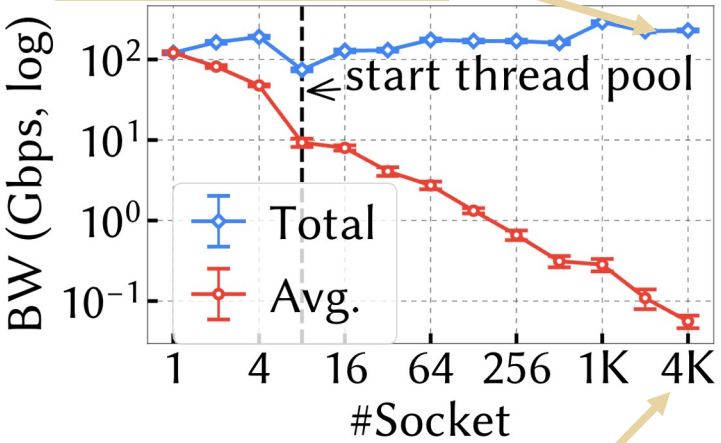
Scales up to 4K devices, approaching socket scale



Evaluation: Micro-Benchmarking SG-IOV

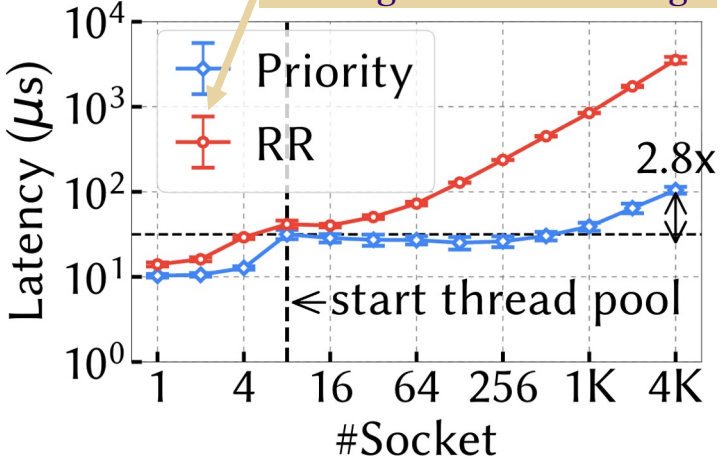
- Test setup
  - Two connected servers (NVIDIA BlueField-3 with a 100GbE switch)
  - Standard POSIX send()/recv() over Warp Pipes
  - 128KB messages for bandwidth; 4KB messages for latency

Sustaining line rate



Scales up to 4K devices, approaching socket scale

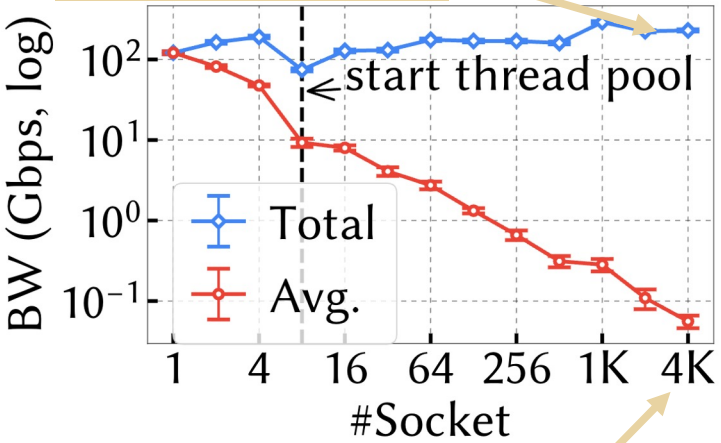
Configurable scheduling



Evaluation: Micro-Benchmarking SG-IOV

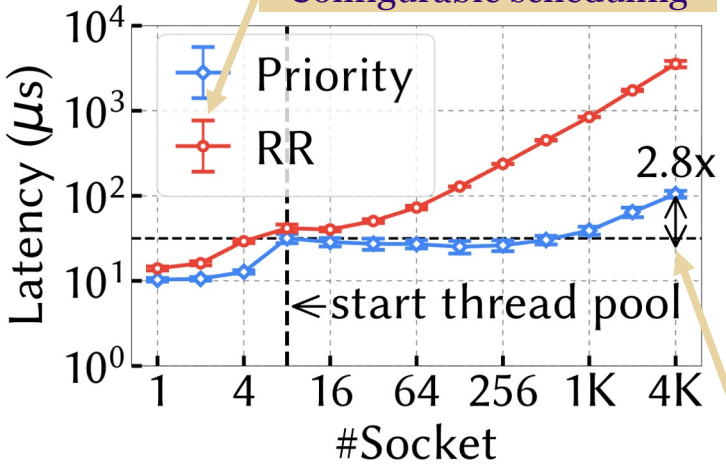
- Test setup
  - Two connected servers (NVIDIA BlueField-3 with a 100GbE switch)
  - Standard POSIX send()/recv() over Warp Pipes
  - 128KB messages for bandwidth; 4KB messages for latency

Sustaining line rate



Scales up to 4K devices, approaching socket scale

Configurable scheduling



Strong isolation; for comparison, 128 SR-IOV VFs yield 3.7×

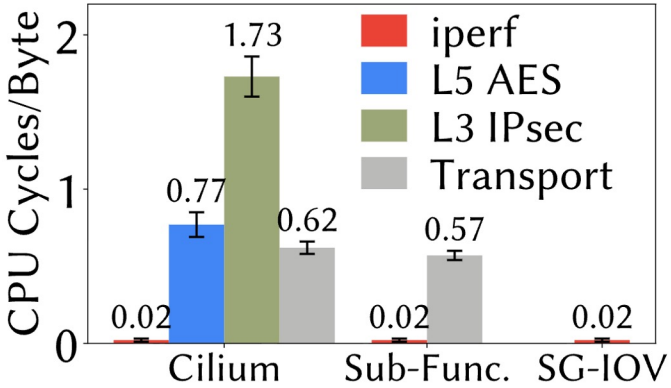
Evaluation: End-to-End Results for SGIOV-CNI (1/2)

Evaluation: End-to-End Results for SGIOV-CNI (1/2)

- Saving Host CPU Cycles
  - Baselines: 1) Cilium; 2) CNI over NVIDIA Sub-Functions
  - Applications:
    - 1) Containerized *iperf3* (128KB message) with VxLAN, L3 IPSec, and L5 TLS
    - 2) Containerized *wrk* (HTTP requests) → *Envoy* proxy → *Nginx* server

Evaluation: End-to-End Results for SGIOV-CNI (1/2)

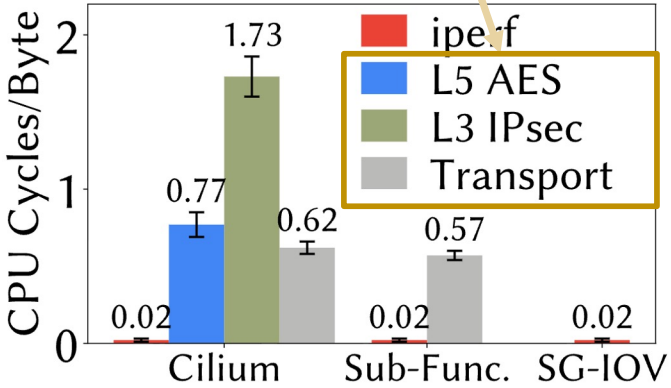
- Saving Host CPU Cycles
  - Baselines: 1) Cilium; 2) CNI over NVIDIA Sub-Functions
  - Applications:
    - 1) Containerized *iperf3* (128KB message) with VxLAN, L3 IPSec, and L5 TLS
    - 2) Containerized *wrk* (HTTP requests) → *Envoy* proxy → *Nginx* server



Evaluation: End-to-End Results for SGIOV-CNI (1/2)

- Saving Host CPU Cycles
  - Baselines: 1) Cilium; 2) CNI over NVIDIA Sub-Functions
  - Applications:
    - 1) Containerized *iperf3* (128KB message) with VxLAN, L3 IPSec, and L5 TLS
    - 2) Containerized *wrk* (HTTP requests) → *Envoy* proxy → *Nginx* server

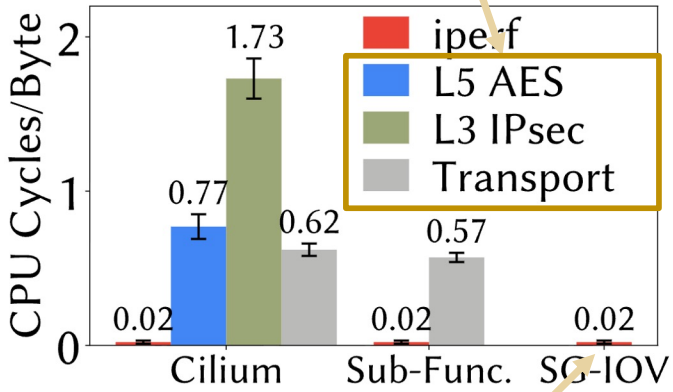
SG-IOV offloads these onto SmartNICs



Evaluation: End-to-End Results for SGIOV-CNI (1/2)

- Saving Host CPU Cycles
  - Baselines: 1) Cilium; 2) CNI over NVIDIA Sub-Functions
  - Applications:
    - 1) Containerized *iperf3* (128KB message) with VxLAN, L3 IPSec, and L5 TLS
    - 2) Containerized *wrk* (HTTP requests) → *Envoy* proxy → *Nginx* server

SG-IOV offloads these onto SmartNICs

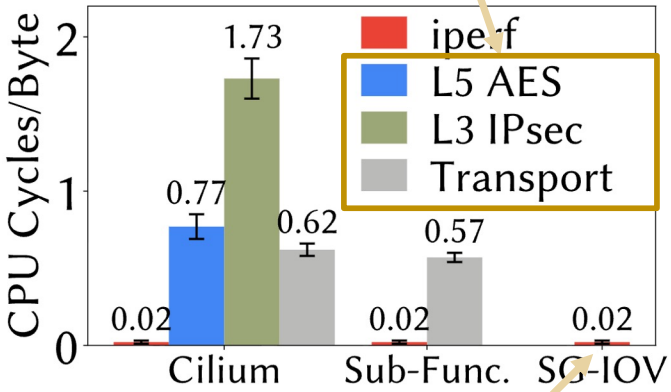


Only app on host; saves 1.9 cores/10Gbps

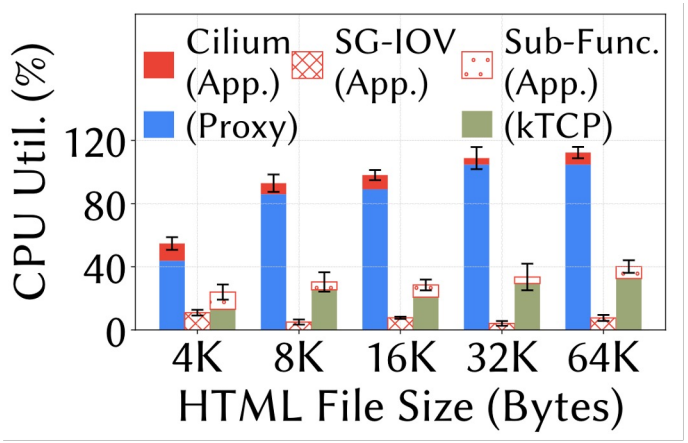
Evaluation: End-to-End Results for SGIOV-CNI (1/2)

- Saving Host CPU Cycles
  - Baselines: 1) Cilium; 2) CNI over NVIDIA Sub-Functions
  - Applications:
    - 1) Containerized *iperf3* (128KB message) with VxLAN, L3 IPSec, and L5 TLS
    - 2) Containerized *wrk* (HTTP requests) → *Envoy* proxy → *Nginx* server

SG-IOV offloads these onto SmartNICs



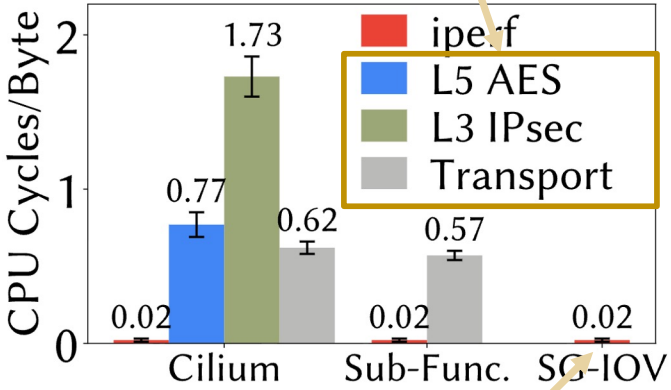
Only app on host; saves 1.9 cores/10Gbps



Evaluation: End-to-End Results for SGIOV-CNI (1/2)

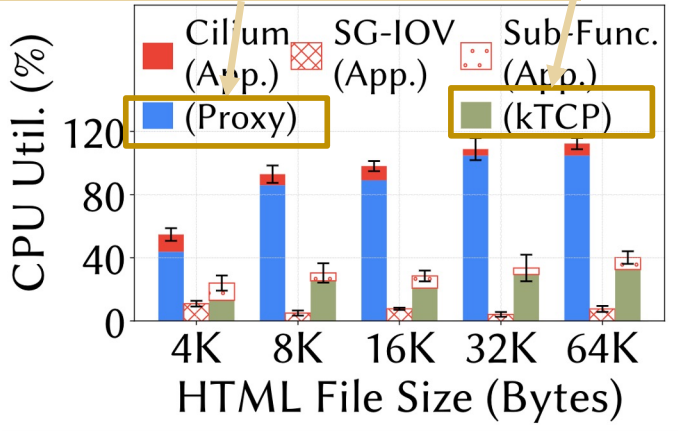
- Saving Host CPU Cycles
  - Baselines: 1) Cilium; 2) CNI over NVIDIA Sub-Functions
  - Applications:
    - 1) Containerized *iperf3* (128KB message) with VxLAN, L3 IPSec, and L5 TLS
    - 2) Containerized *wrk* (HTTP requests) → *Envoy* proxy → *Nginx* server

SG-IOV offloads these onto SmartNICs



Only app on host; saves 1.9 cores/10Gbps

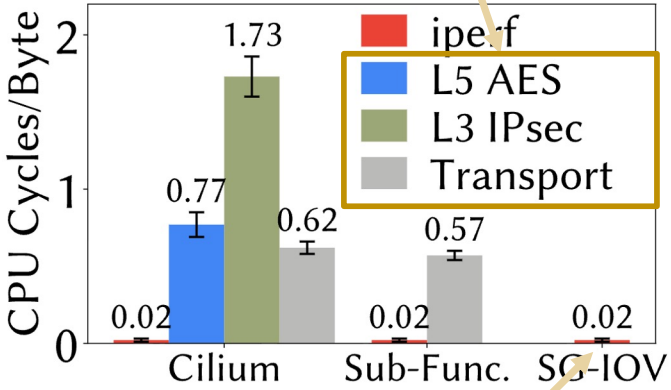
SG-IOV offloads these onto SmartNICs



Evaluation: End-to-End Results for SGIOV-CNI (1/2)

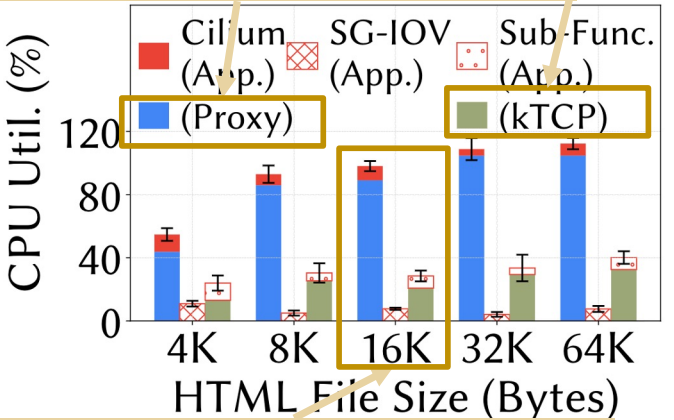
- Saving Host CPU Cycles
  - Baselines: 1) Cilium; 2) CNI over NVIDIA Sub-Functions
  - Applications:
    - 1) Containerized *iperf3* (128KB message) with VxLAN, L3 IPSec, and L5 TLS
    - 2) Containerized *wrk* (HTTP requests) → *Envoy* proxy → *Nginx* server

SG-IOV offloads these onto SmartNICs



Only app on host; saves 1.9 cores/10Gbps

SG-IOV offloads these onto SmartNICs



Save 1.06 CPU cores per connection

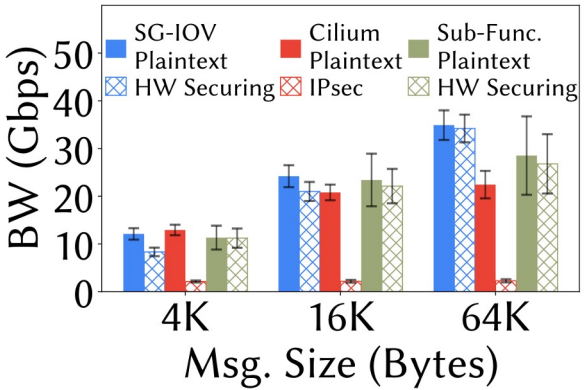
Evaluation: End-to-End Results for SGIOV-CNI (2/2)

Evaluation: End-to-End Results for SGIOV-CNI (2/2)

- Increasing Throughput and Reducing Latency
  - Applications:
    - 1) For throughput, containerized iperf3 (128KB Message), plaintext or IPSec
    - 2) For latency, containerized *NPtcp*, Zero-Copy mode

Evaluation: End-to-End Results for SGIOV-CNI (2/2)

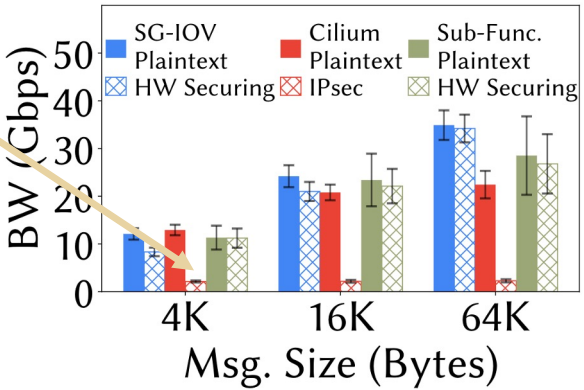
- Increasing Throughput and Reducing Latency
  - Applications:
    - 1) For throughput, containerized iperf3 (128KB Message), plaintext or IPSec
    - 2) For latency, containerized *NPtcp*, Zero-Copy mode



Evaluation: End-to-End Results for SGIOV-CNI (2/2)

- Increasing Throughput and Reducing Latency
  - Applications:
    - 1) For throughput, containerized iperf3 (128KB Message), plaintext or IPSec
    - 2) For latency, containerized *NPtcp*, Zero-Copy mode

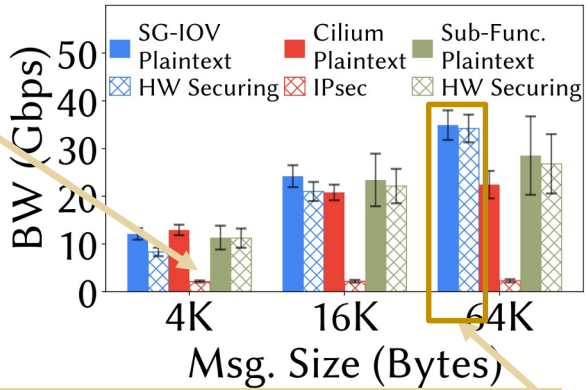
CPU SW is inefficient for secure traffic



Evaluation: End-to-End Results for SGIOV-CNI (2/2)

- Increasing Throughput and Reducing Latency
  - Applications:
    - 1) For throughput, containerized iperf3 (128KB Message), plaintext or IPSec
    - 2) For latency, containerized *NPtcp*, Zero-Copy mode

CPU SW is inefficient for secure traffic

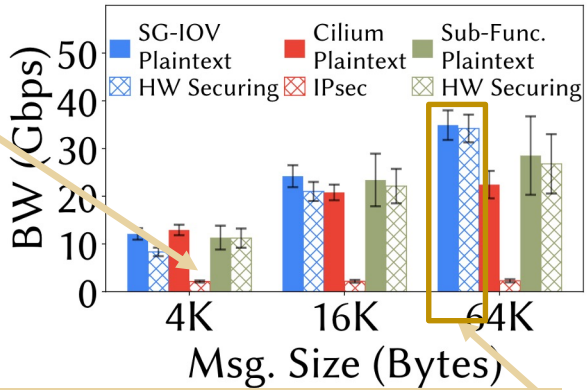


SG-IOV increases BW for plaintext/securing

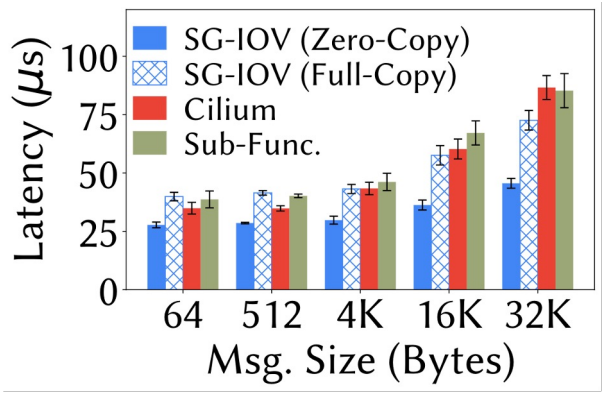
Evaluation: End-to-End Results for SGIOV-CNI (2/2)

- Increasing Throughput and Reducing Latency
  - Applications:
    - 1) For throughput, containerized iperf3 (128KB Message), plaintext or IPSec
    - 2) For latency, containerized *NPtcp*, Zero-Copy mode

CPU SW is inefficient for secure traffic



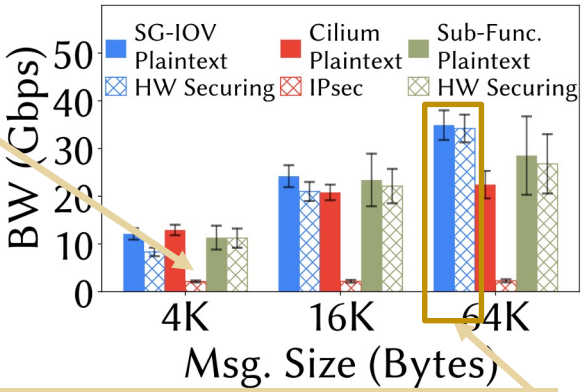
SG-IOV increases BW for plaintext/securing



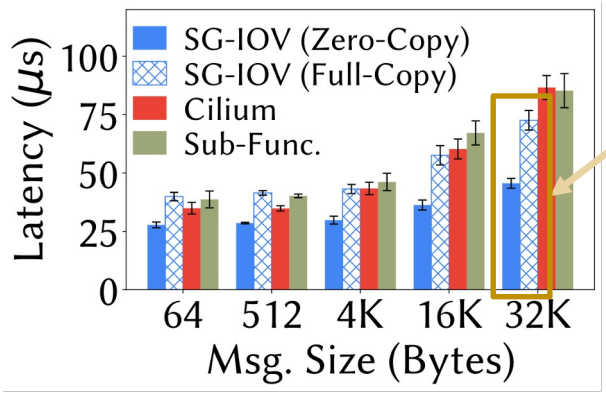
Evaluation: End-to-End Results for SGIOV-CNI (2/2)

- Increasing Throughput and Reducing Latency
  - Applications:
    - 1) For throughput, containerized iperf3 (128KB Message), plaintext or IPSec
    - 2) For latency, containerized *NPtcp*, Zero-Copy mode

CPU SW is inefficient for secure traffic



SG-IOV increases BW for plaintext/securing

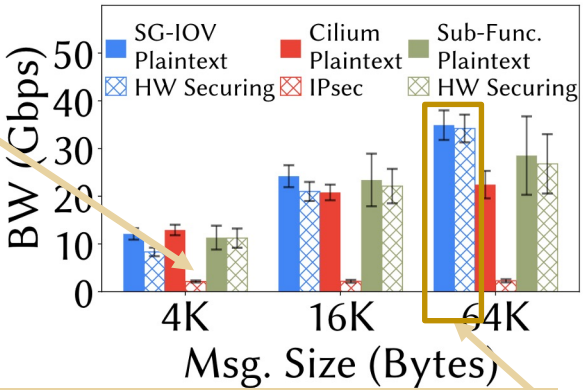


Reduces latency, especially with zero-copy

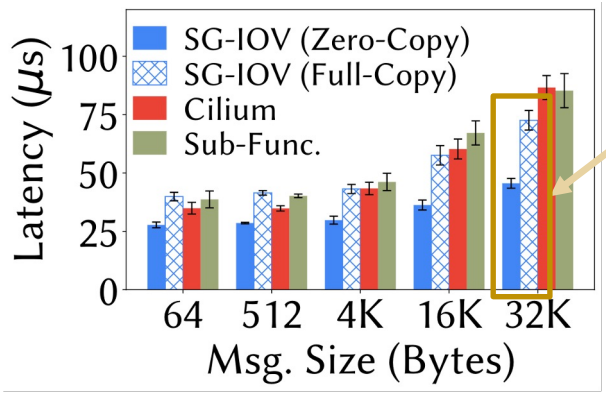
Evaluation: End-to-End Results for SGIOV-CNI (2/2)

- Increasing Throughput and Reducing Latency
  - Applications:
    - 1) For throughput, containerized iperf3 (128KB Message), plaintext or IPSec
    - 2) For latency, containerized *NPtcp*, Zero-Copy mode

CPU SW is inefficient for secure traffic



SG-IOV increases BW for plaintext/securing



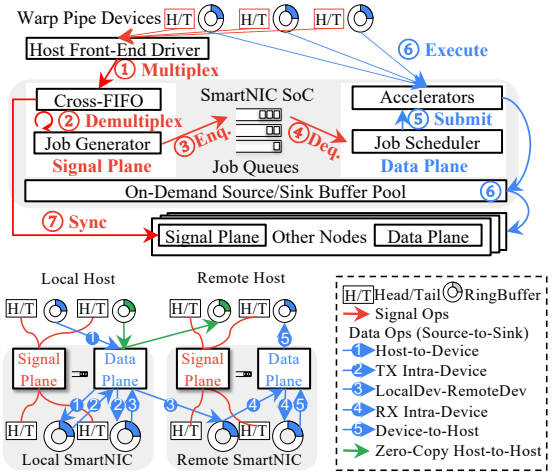
Reduces latency, especially with zero-copy

See the paper for more results on functionality, scalability, and cost

Conclusion

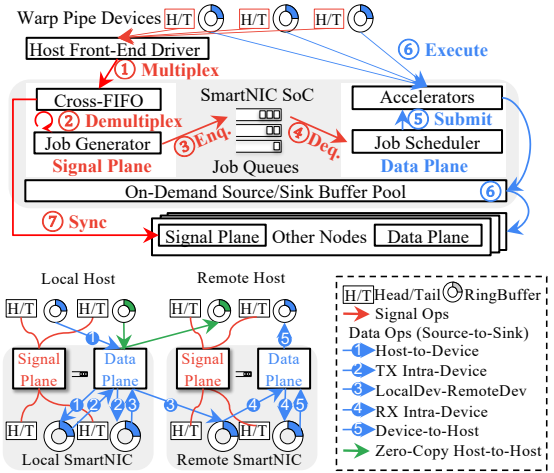
Conclusion

- How to Build Efficient Container Networks
  - SmartNIC-based CNI
- Socket-Granular I/O Virtualization
  - Scalability
  - Flexibility
  - Granular Virtualization
- SGIOV-CNI
  - Save Host CPU Cores
  - Boost Performance



Conclusion

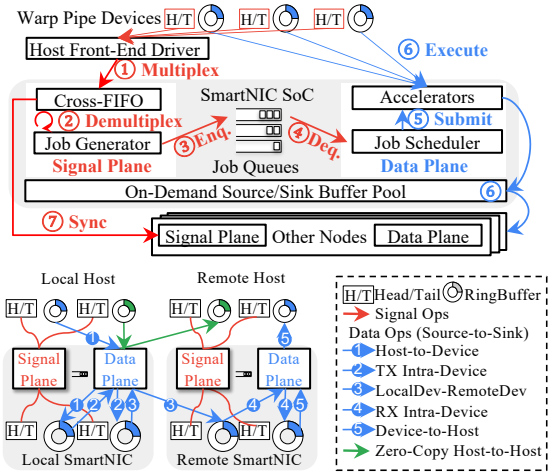
- How to Build Efficient Container Networks
  - SmartNIC-based CNI
- Socket-Granular I/O Virtualization
  - Scalability
  - Flexibility
  - Granular Virtualization
- SGIOV-CNI
  - Save Host CPU Cores
  - Boost Performance



SG-IOV: Lifting Hardware IOV to Operate at L5 Socket Granularity

Conclusion

- How to Build Efficient Container Networks
  - SmartNIC-based CNI
- Socket-Granular I/O Virtualization
  - Scalability
  - Flexibility
  - Granular Virtualization
- SGIOV-CNI
  - Save Host CPU Cores
  - Boost Performance



SG-IOV: Lifting Hardware IOV to Operate at L5 Socket Granularity

ASPLOS

Thanks for listening!

Frank Chenxingyu Zhao
cxyzhao@cs.washington.edu
University of Washington

UNIVERSITY of WASHINGTON