Mikhail Khalilov, Marcin Chrapek, Siyuan Shen, Alessandro Vezzu, Thomas Benz, Salvatore Di Girolamo, and Timo Schneider, ***ETH Zürich***
Daniele De Sensi, ***ETH Zürich and Sapienza University of Rome***
Luca Benini and Torsten Hoefler, ***ETH Zürich***

OSMOSIS: Enabling Multi-Tenancy in Datacenter SmartNICs

Why SmartNICs?

Gap between CPU and NIC bandwidth

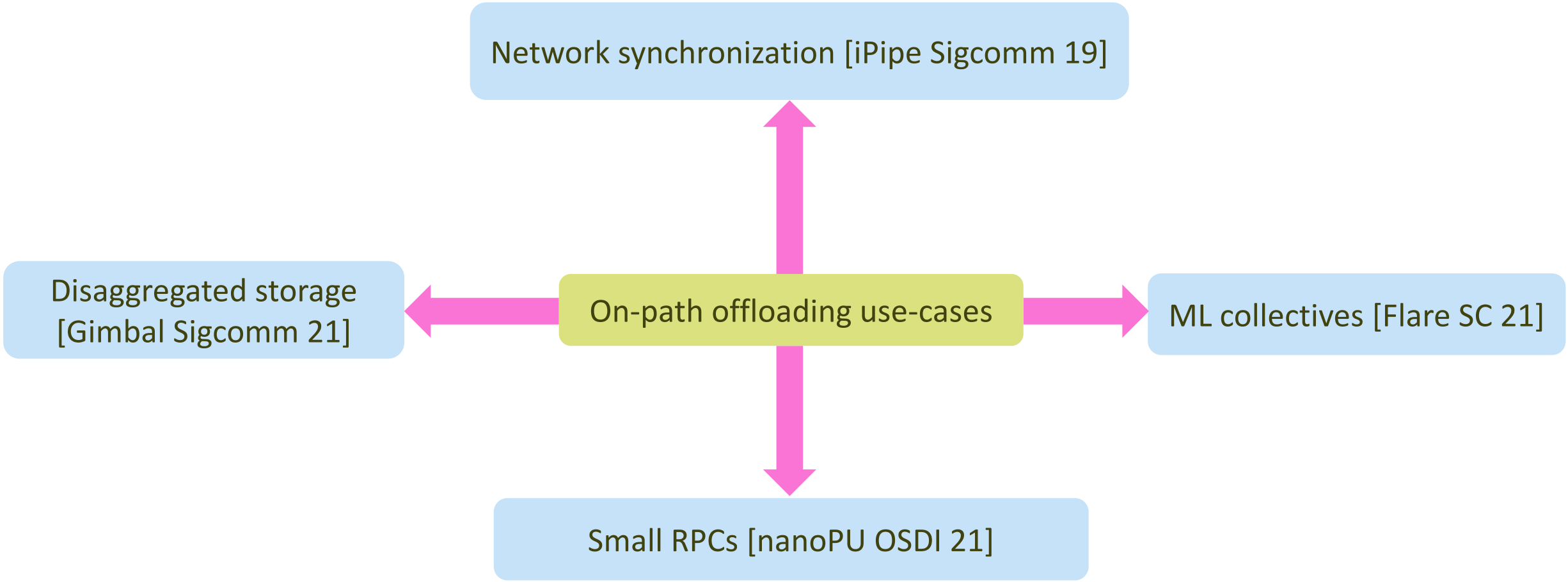
ConnectX-2 (2009): 10Gbps/port → 40x → **ConnectX-7** (2023): 400 Gbps/port

CPU bandwidth scales slower than the NIC bandwidth

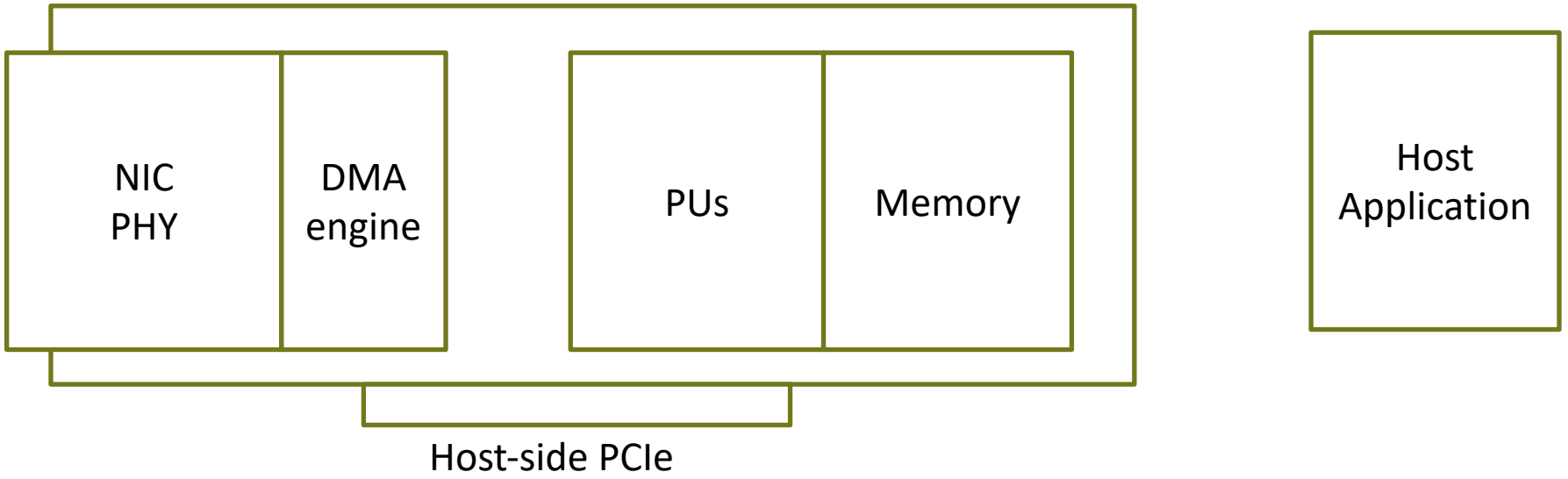
TX path
CPU cannot saturate NIC

RX path
CPU can't process packets at a line rate

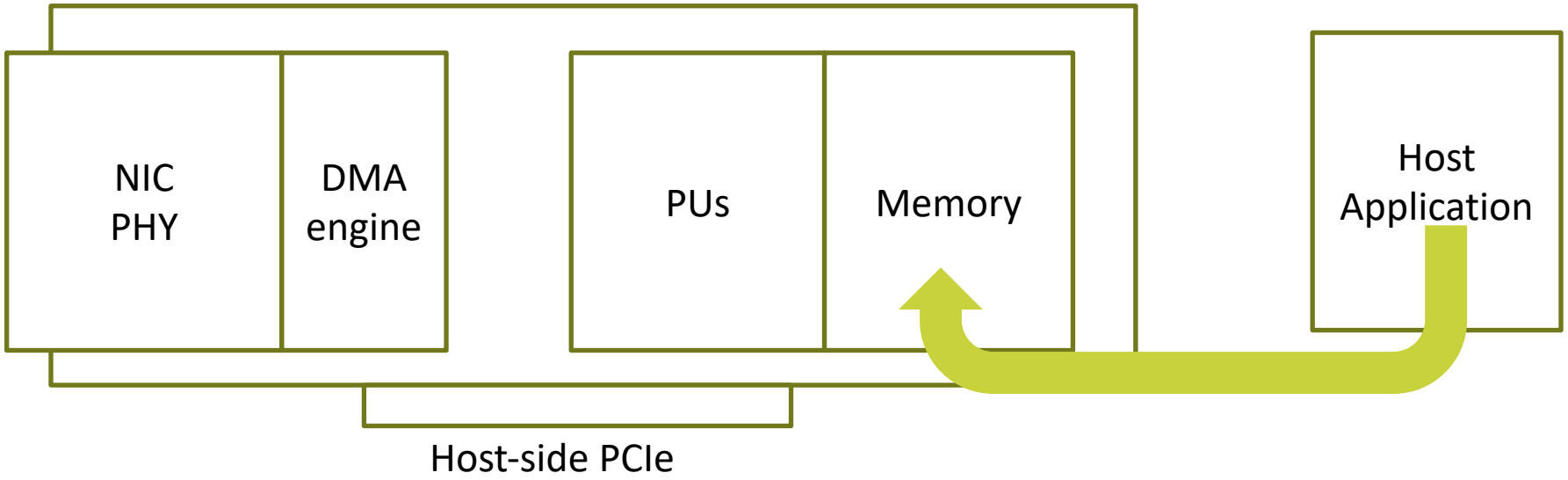
SmartNIC offloading approach:
free CPU cycles by offloading "hot" parts of the host-side stack to the NIC data path


Network synchronization [iPipe Sigcomm 19]
Disaggregated storage [Gimbal Sigcomm 21]
On-path offloading use-cases
ML collectives [Flare SC 21]
Small RPCs [nanoPU OSDI 21]

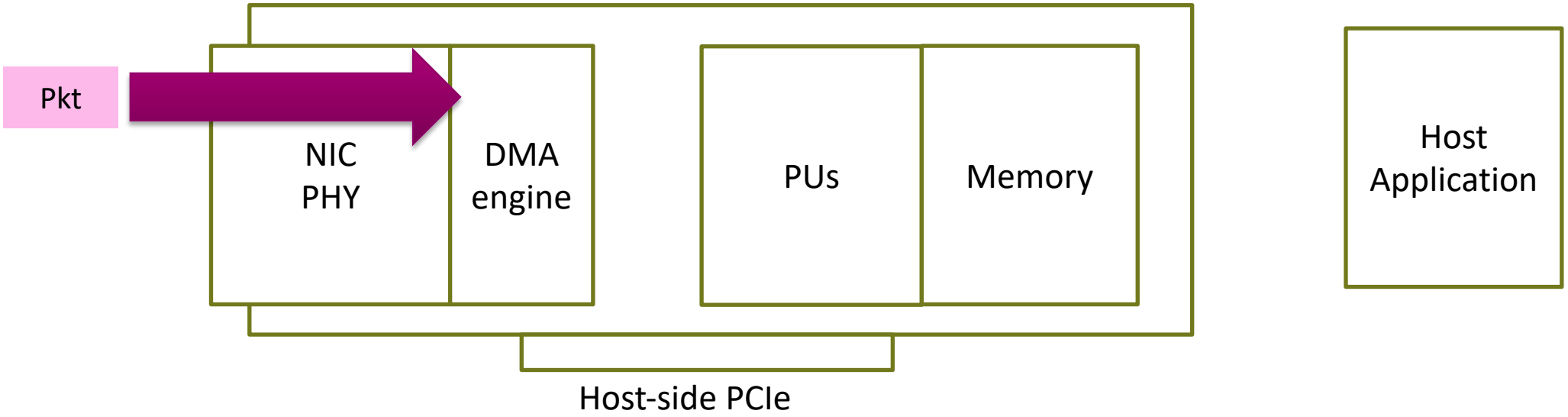
SmartNIC offloading



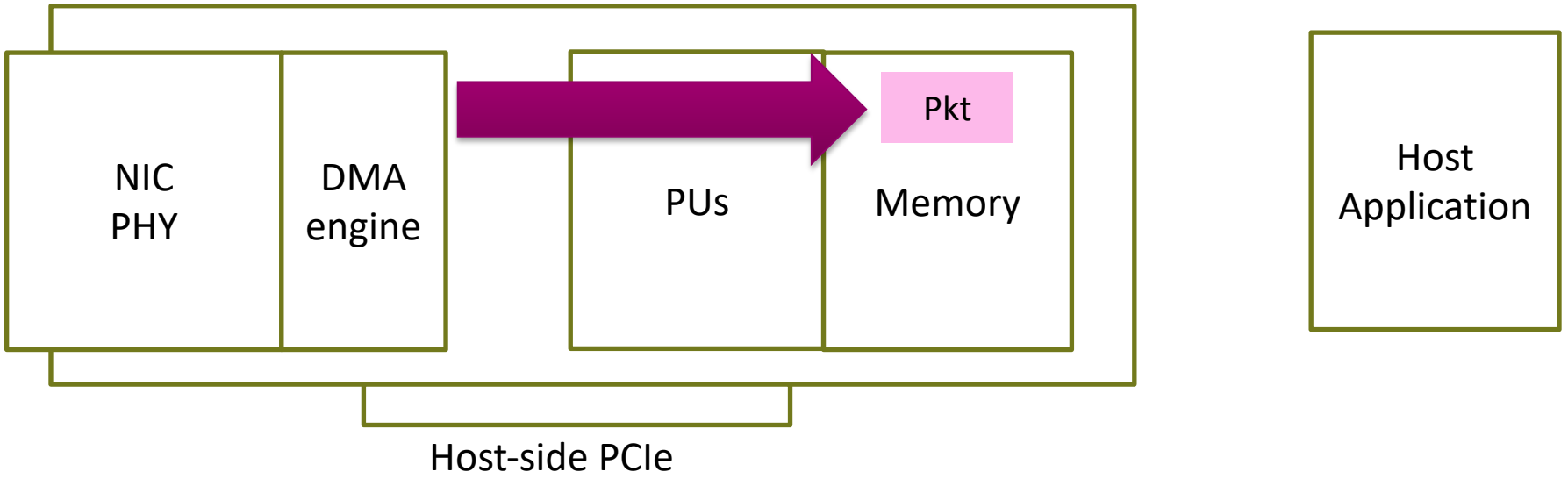
SmartNIC offloading



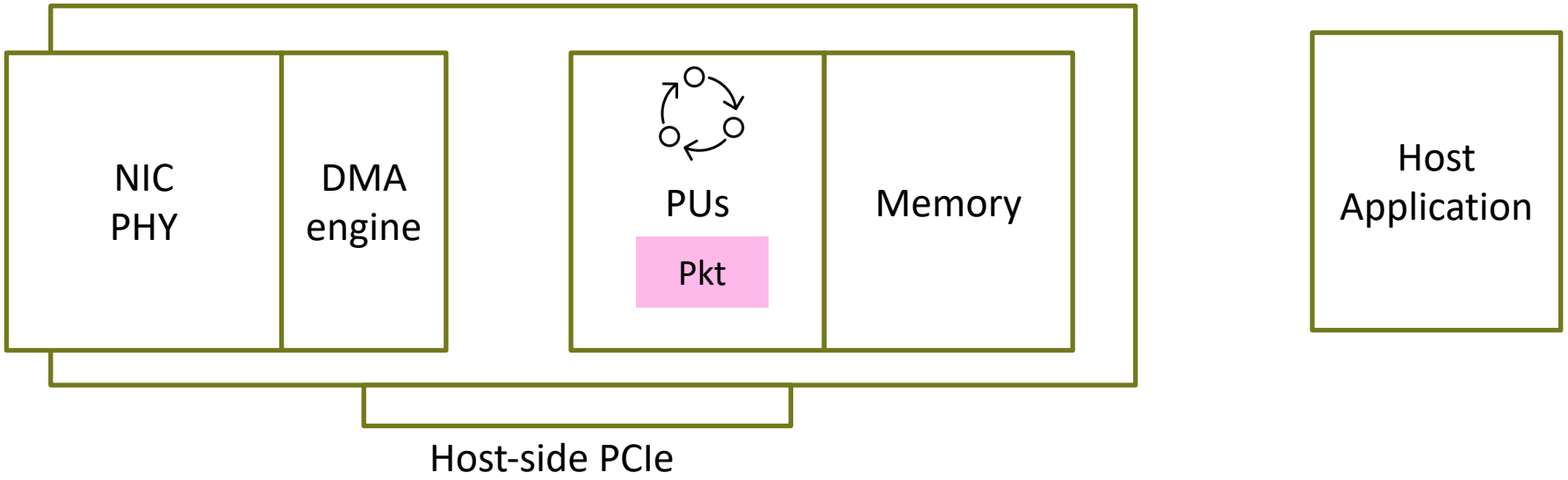
SmartNIC offloading



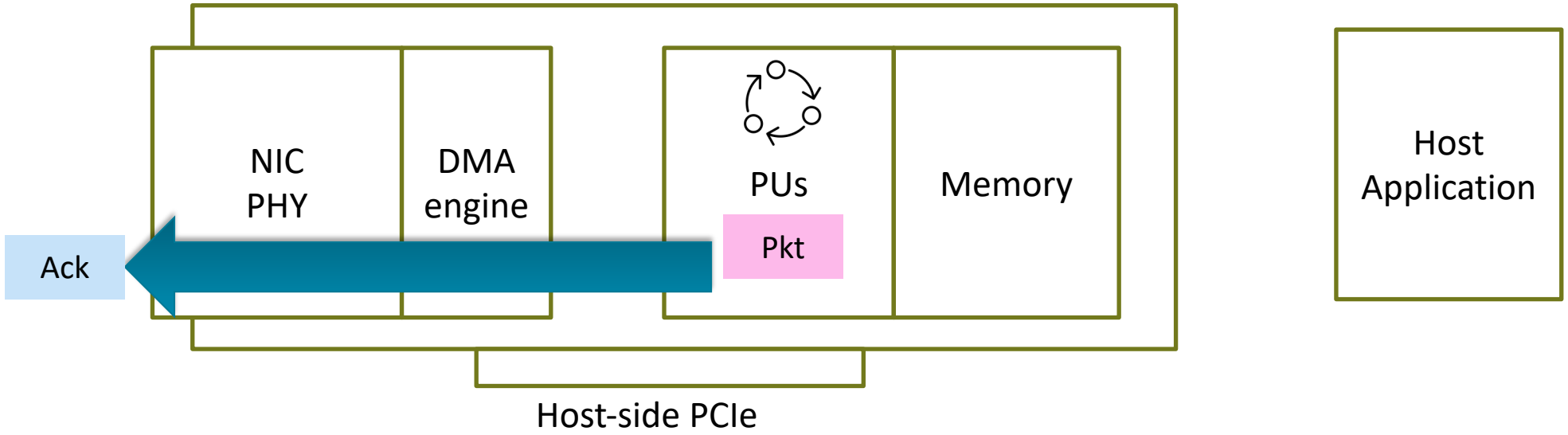
SmartNIC offloading



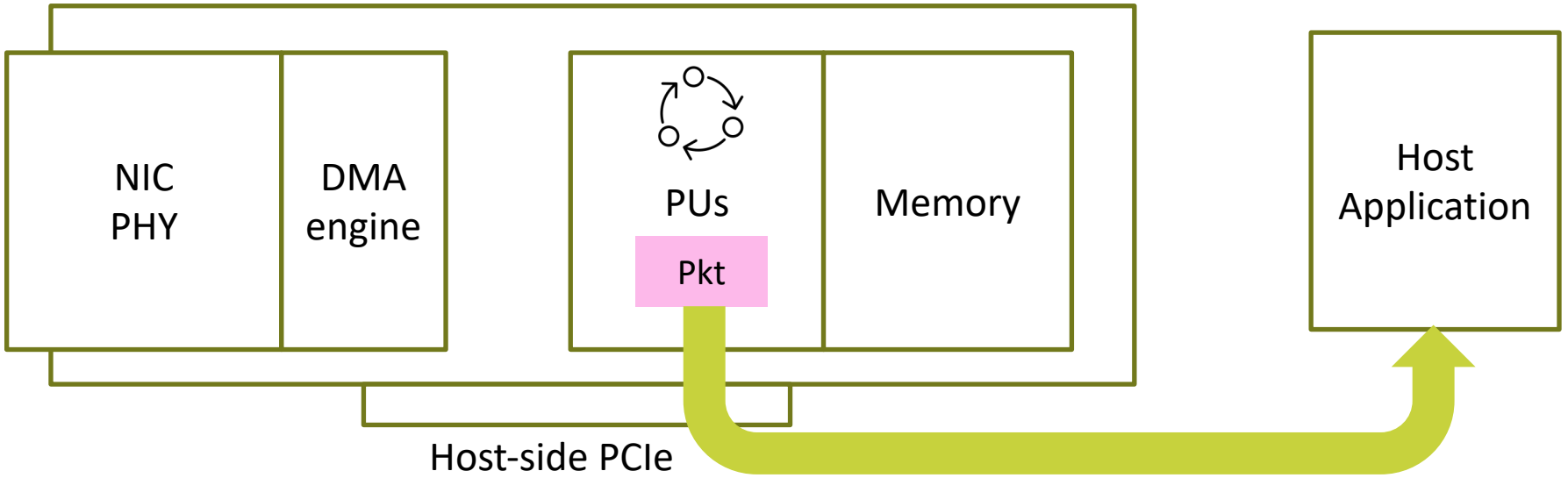
SmartNIC offloading



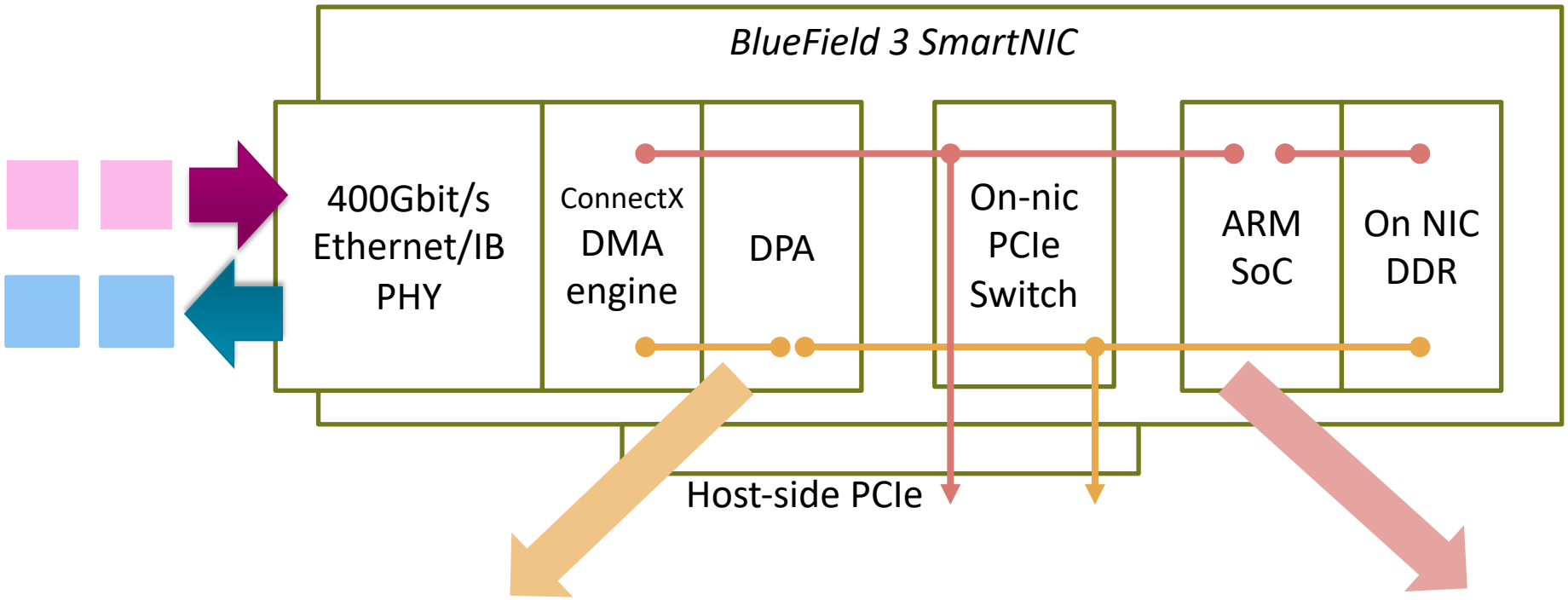
SmartNIC offloading



SmartNIC offloading



State-of-the-art: *on*-path vs *off*-path offloading



On-path: Data Path Accelerator (DPA)
16 hyper-threaded RISC-V cores: 256 hardware threads
Ideal for low-IPC highly-parallel workloads
C API to interact with DMA/accelerators

Off-path: ARM SoC
16 superscalar OOO ARM cores
Tailored for single-thread-bound workloads
Linux RDMA/DOCA stacks

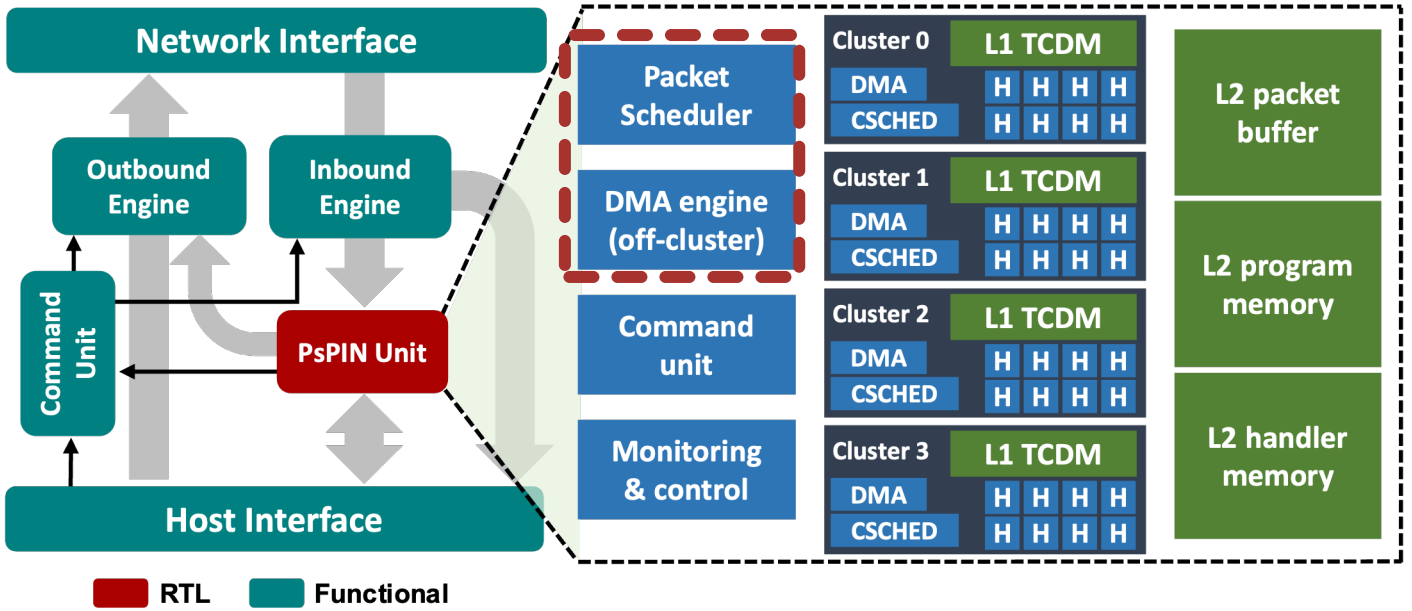
In our work we focus on the on-path offloading

Q: What will happen if we'll share SmartNIC resources between tenants?

A: Let's take an open-source SmartNIC and see!

Open-source PsPIN on-path SmartNIC

- Software/hardware stack based on energy-efficient RISC-V PUs
- Support for general-purpose per-packet processing with C
- Offloading is very similar to the NVIDIA BlueField Datapath Accelerator

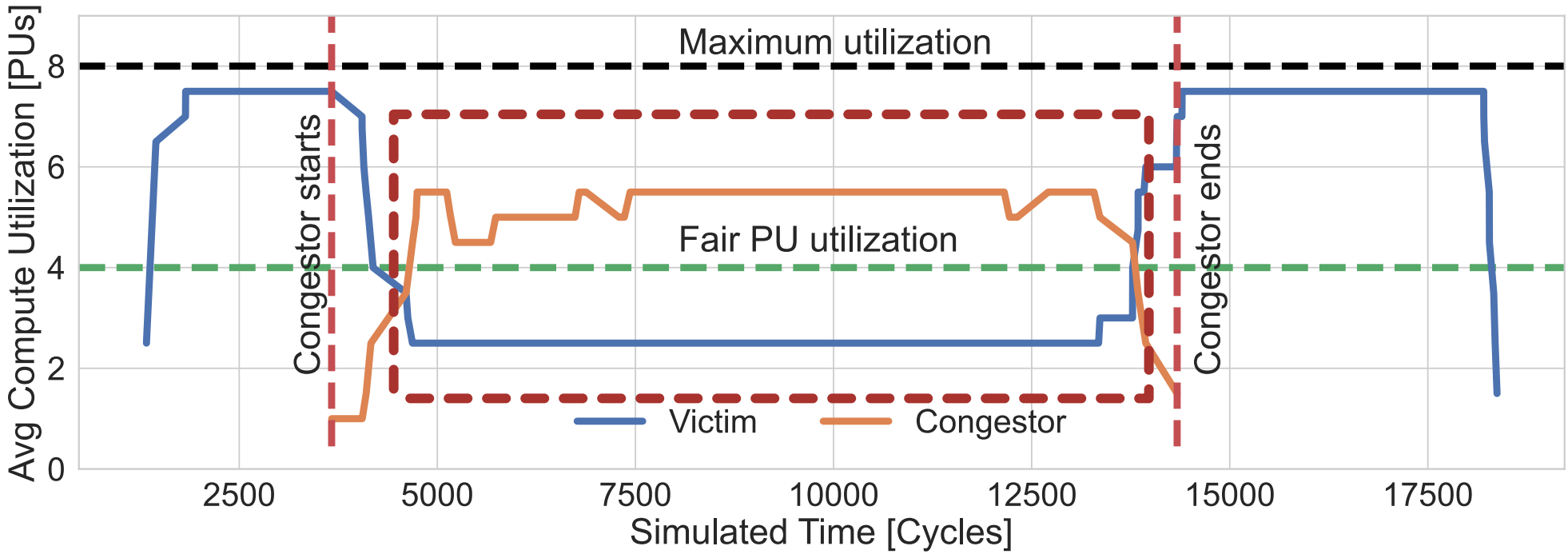


What are the implications of resource sharing for DMA engine and PUs between tenants?

[Di Girolamo et. al., ISCA 48]

In-network compute (INC) management

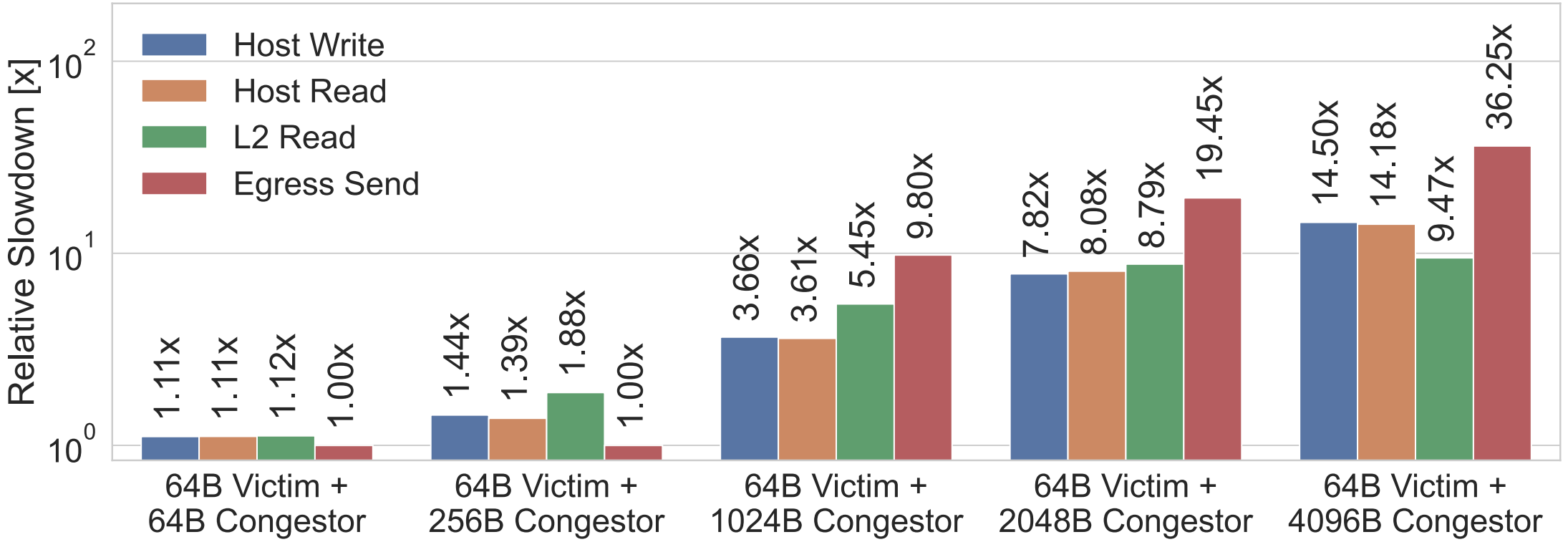
Victim tenant needs 2x less PU cycles to process single packet



Conventional round robin scheduling for compute engines is unfair

IO management

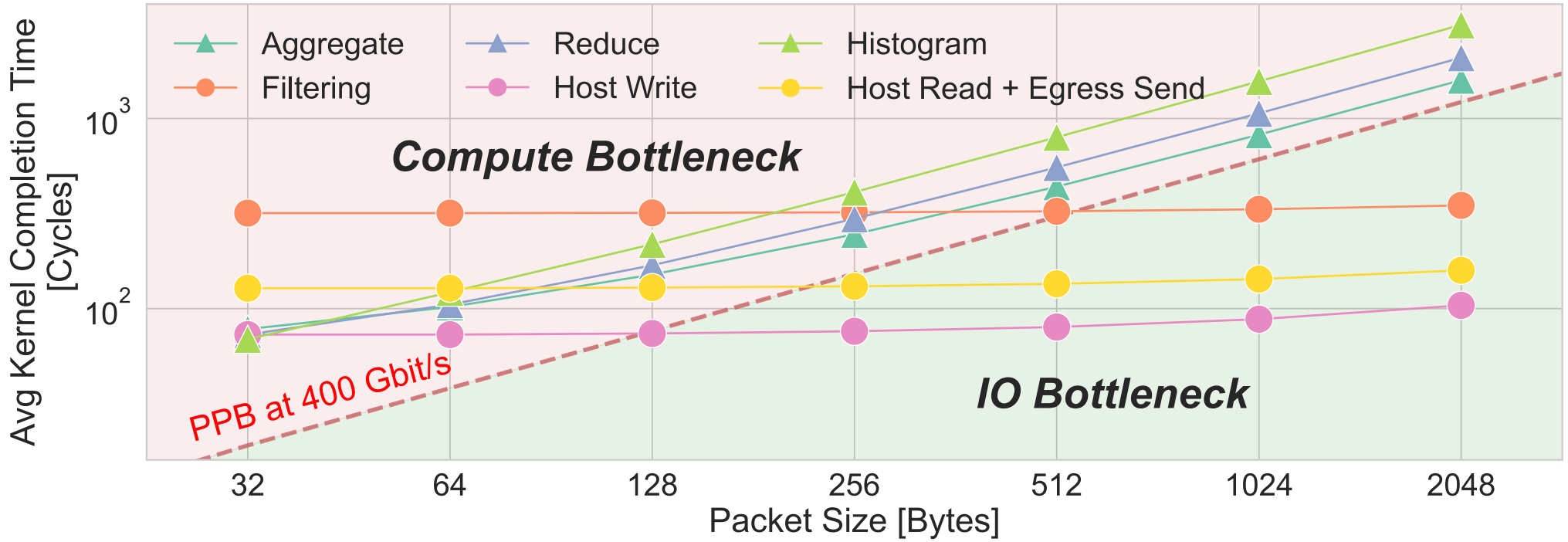
Two tenants offload kernels to serve IO RPCs of different sizes



FIFO processing of DMA requests results in HoL-blocking

… but why not use standard OS schedulers?

Packet processing deadlines at 400 Gbit/s and 1 GHz



<1 us to process packet

Software packet scheduling is not feasbile

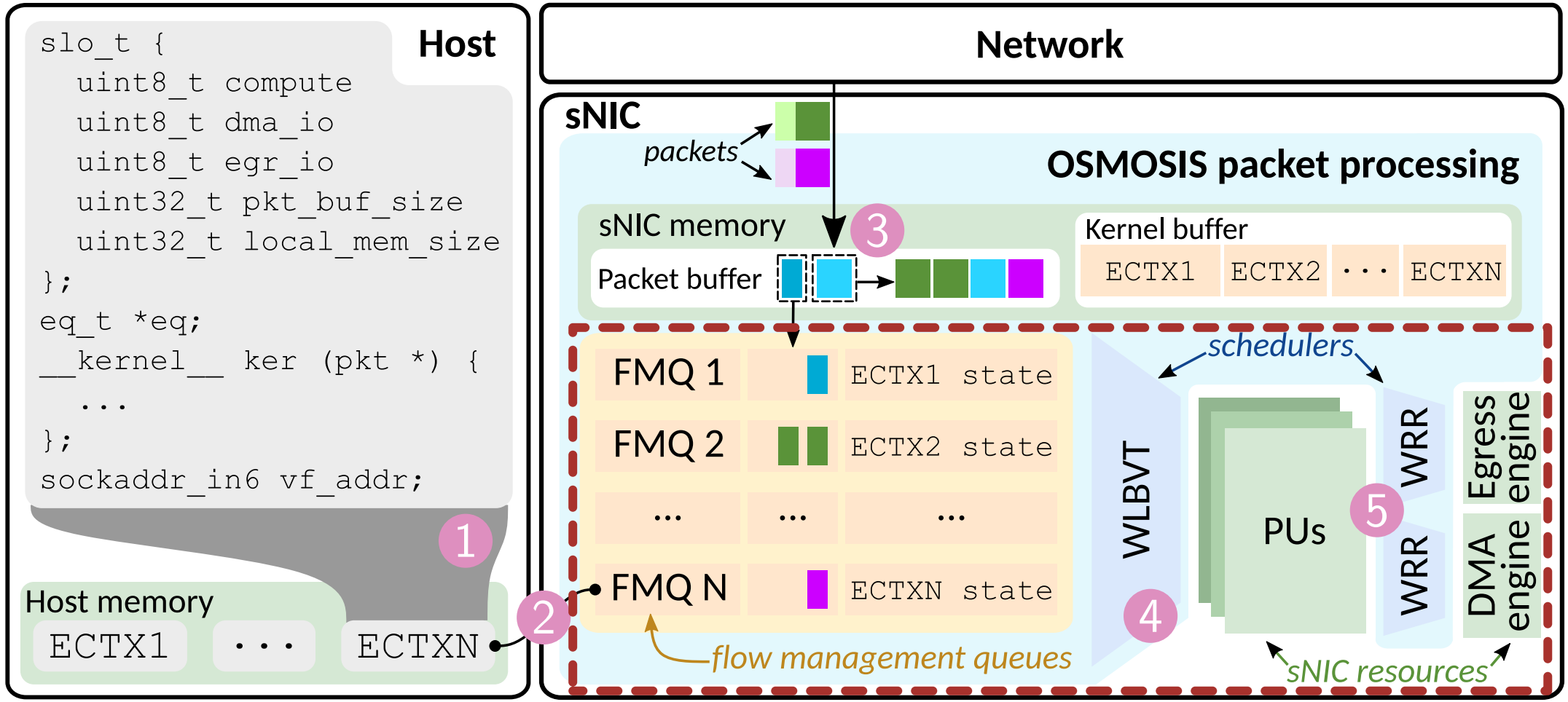
OSMOSIS

OSMOSIS: Operating System Support for Streaming In-Network Processing

- Hardware-based resource management designed for next-generation on-path SmartNICs
- Dynamic and work-conserving IO/compute tenant resource management
- Support for data-center SLOs and priority enforcement

	PUs	DMA	Egress	Memory
Scheduler	WLBVT	WRR	WRR	Static
SLO knob	Priority Kernel cycle limit	Priority	Priority	Allocation size

OSMOSIS overview



OSMOSIS compute management: WLBVT scheduler

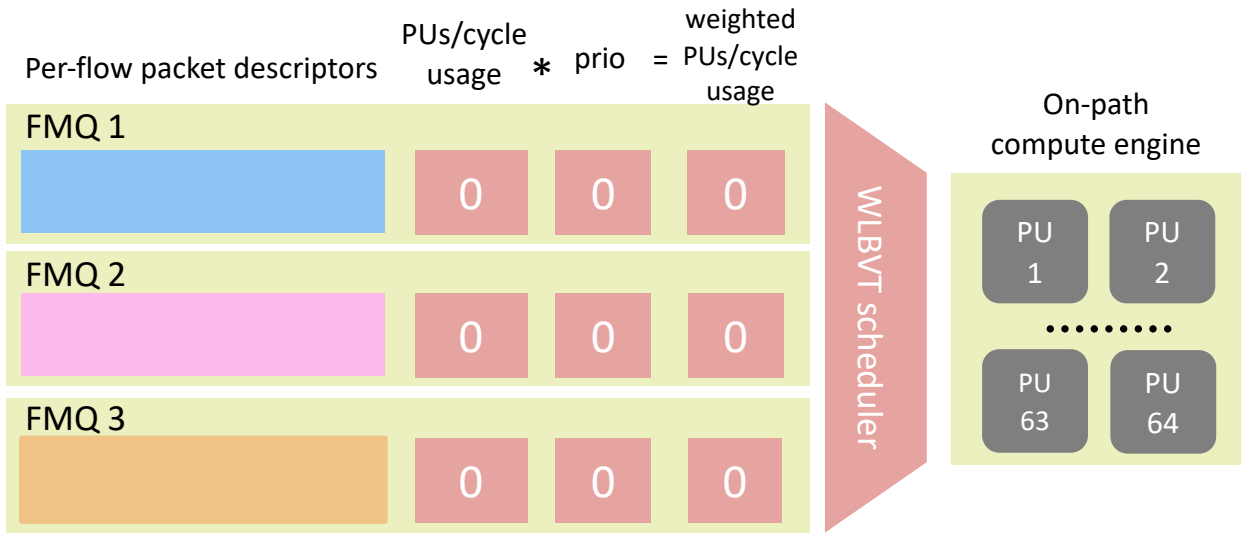
Weight-limited Borrowed Virtual Time

Inspired by the BVT [SOSP-17] and Shinjuku [NSDI '19] schedulers

- Track how many PUs/cycle the FMQs need
- Choose the FMQ with the smallest number of used cycles

Use WRR-like weights to support priorities

- Scale down number of used cycles according to priority



Initial system state:
All PUs are idle

OSMOSIS compute management: WLBVT scheduler

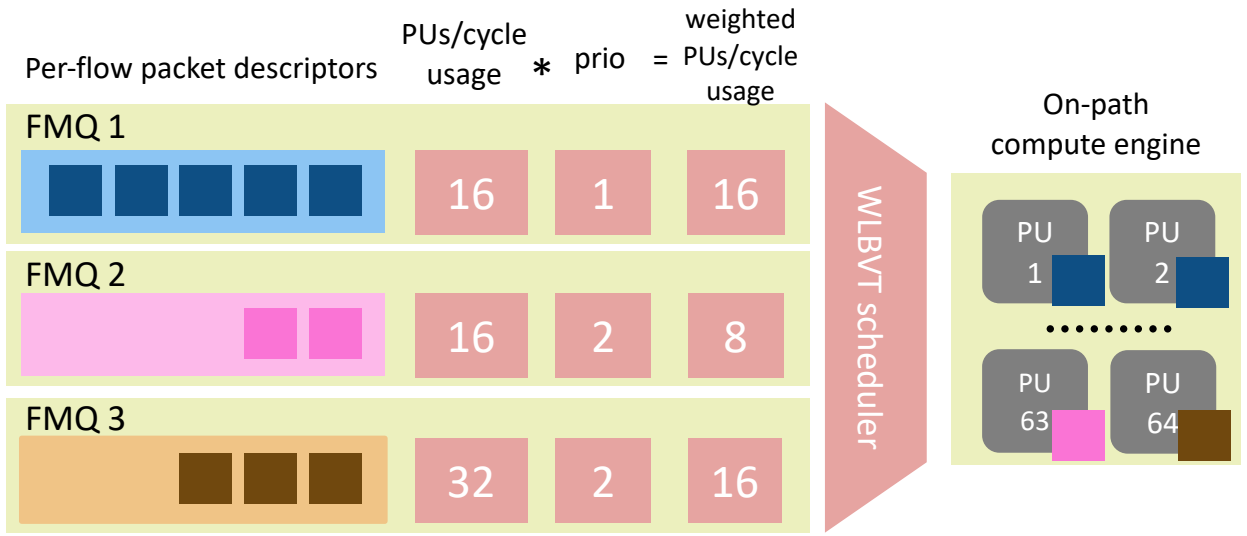
Weight-limited Borrowed Virtual Time

Inspired by the BVT [SOSP-17] and Shinjuku [NSDI '19] schedulers

- Track how many PUs/cycle the FMQs need
- Choose the FMQ with the smallest number of used cycles

Use WRR-like weights to support priorities

- Scale down number of used cycles according to priority



System state at cycle N:
All PUs are busy

OSMOSIS compute management: WLBVT scheduler

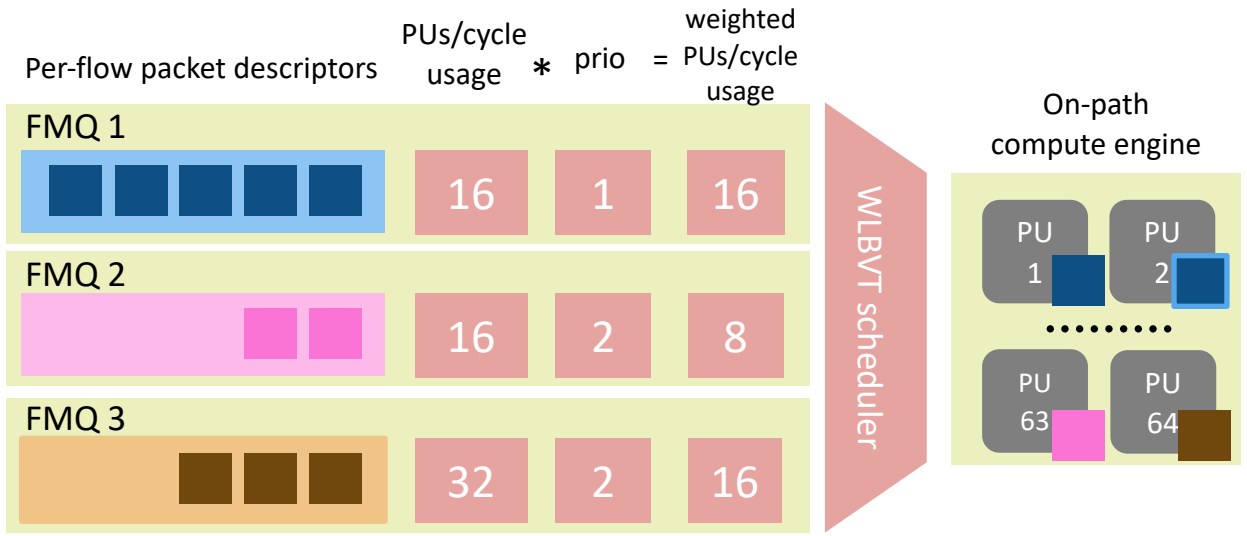
Weight-limited Borrowed Virtual Time

Inspired by the BVT [SOSP-17] and Shinjuku [NSDI '19] schedulers

- Track how many PUs/cycle the FMQs need
- Choose the FMQ with the smallest number of used cycles

Use WRR-like weights to support priorities

- Scale down number of used cycles according to priority



System state at cycle N + 1:
PU 2 finishes processing FMQ 1 packet

OSMOSIS compute management: WLBVT scheduler

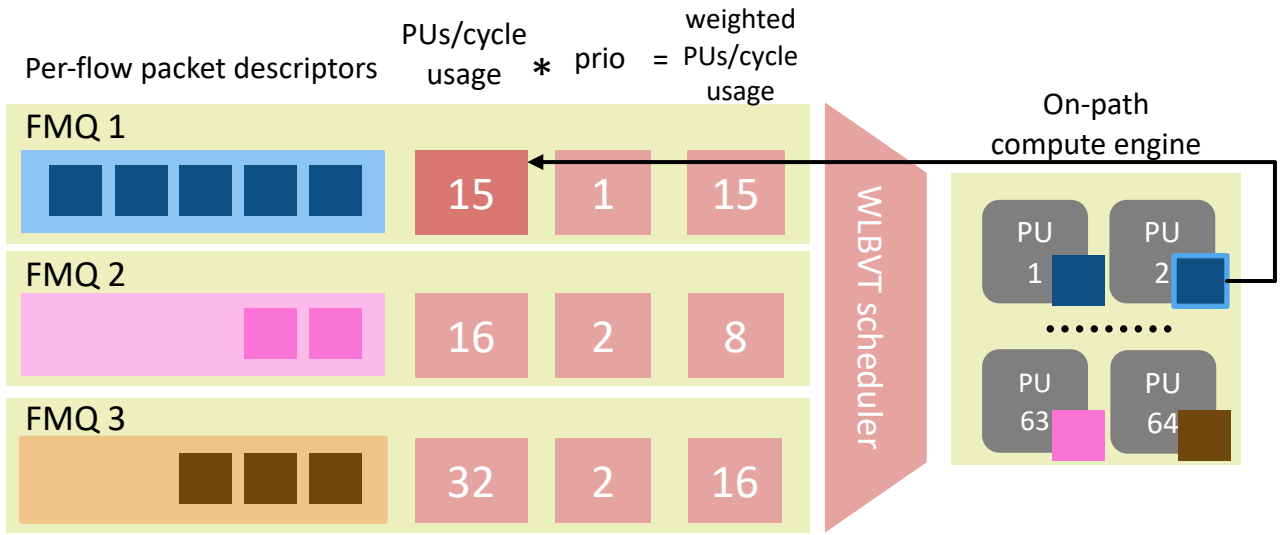
Weight-limited Borrowed Virtual Time

Inspired by the BVT [SOSP-17] and Shinjuku [NSDI '19] schedulers

- Track how many PUs/cycle the FMQs need
- Choose the FMQ with the smallest number of used cycles

Use WRR-like weights to support priorities

- Scale down number of used cycles according to priority



System state at cycle N + 2:
Feedback update to FMQ 1 PU usage

OSMOSIS compute management: WLBVT scheduler

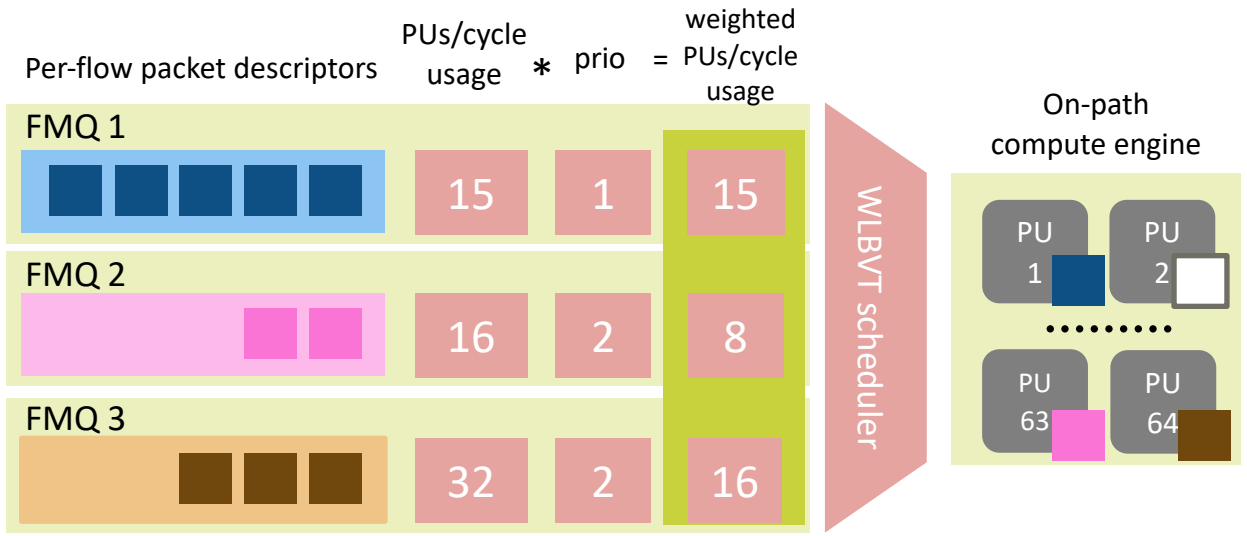
Weight-limited Borrowed Virtual Time

Inspired by the BVT [SOSP-17] and Shinjuku [NSDI '19] schedulers

- Track how many PUs/cycle the FMQs need
- Choose the FMQ with the smallest number of used cycles

Use WRR-like weights to support priorities

- Scale down number of used cycles according to priority



System state at cycle N + 3:
Find FMQ with minimal PU usage

OSMOSIS compute management: WLBVT scheduler

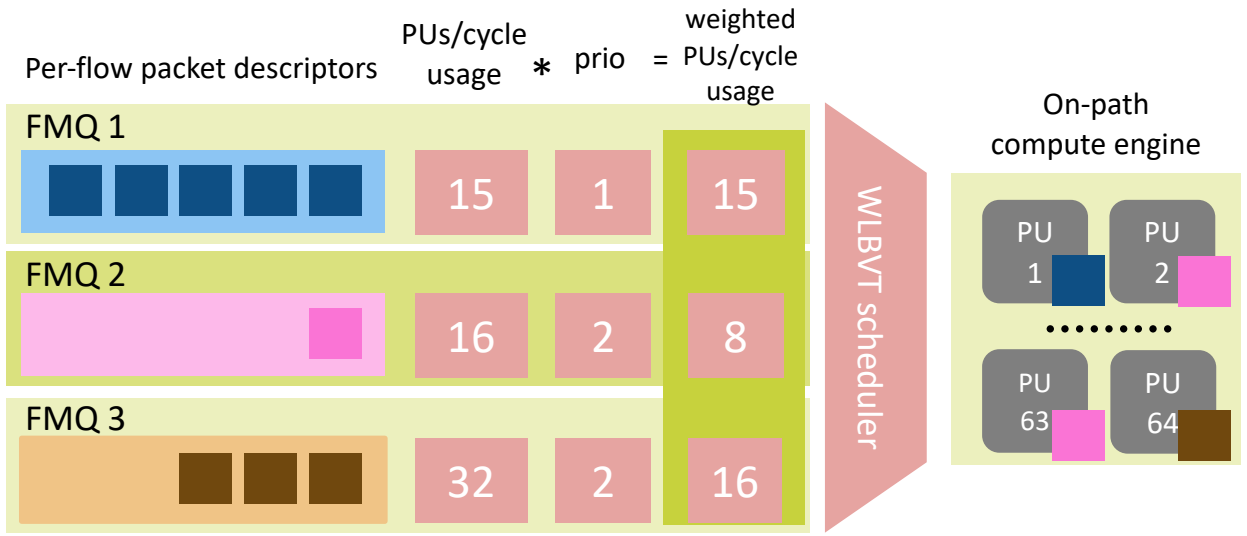
Weight-limited Borrowed Virtual Time

Inspired by the BVT [SOSP-17] and Shinjuku [NSDI '19] schedulers

- Track how many PUs/cycle the FMQs need
- Choose the FMQ with the smallest number of used cycles

Use WRR-like weights to support priorities

- Scale down number of used cycles according to priority



System state at cycle N + 4:
Schedule packet from FMQ 2

OSMOSIS compute management: WLBVT scheduler

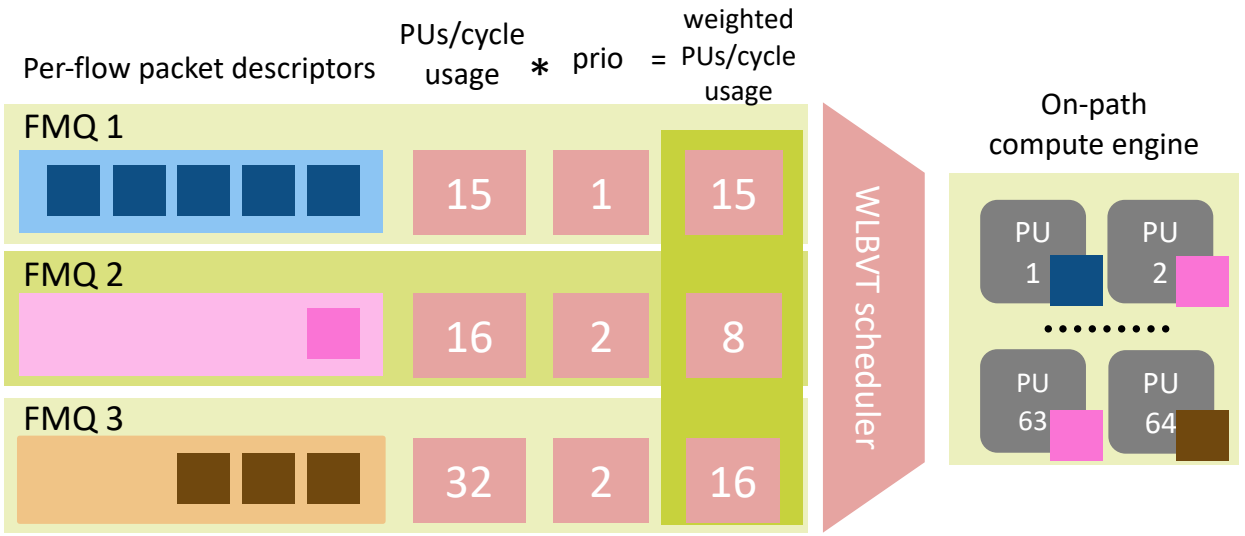
Weight-limited Borrowed Virtual Time

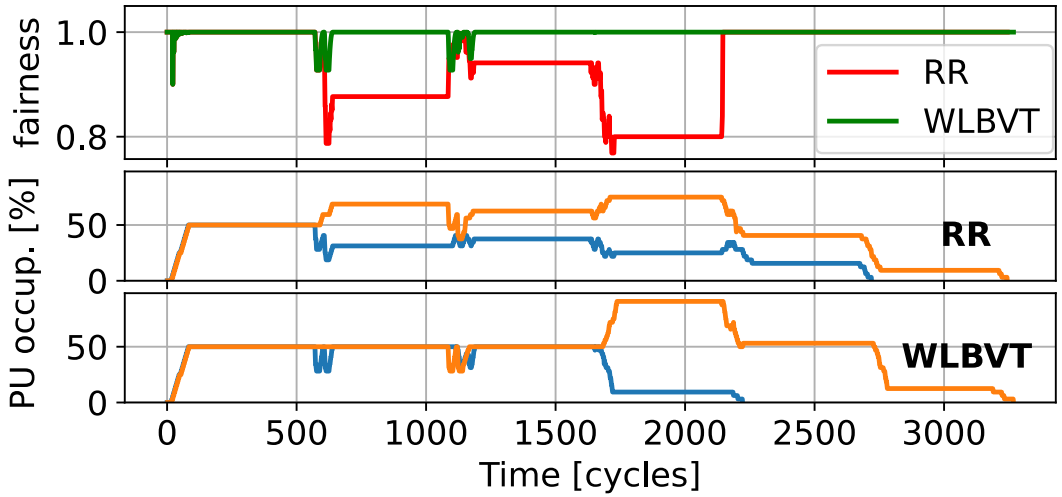
Inspired by the BVT [SOSP-17] and Shinjuku [NSDI '19] schedulers

- Track how many PUs/cycle the FMQs need
- Choose the FMQ with the smallest number of used cycles

Use WRR-like weights to support priorities

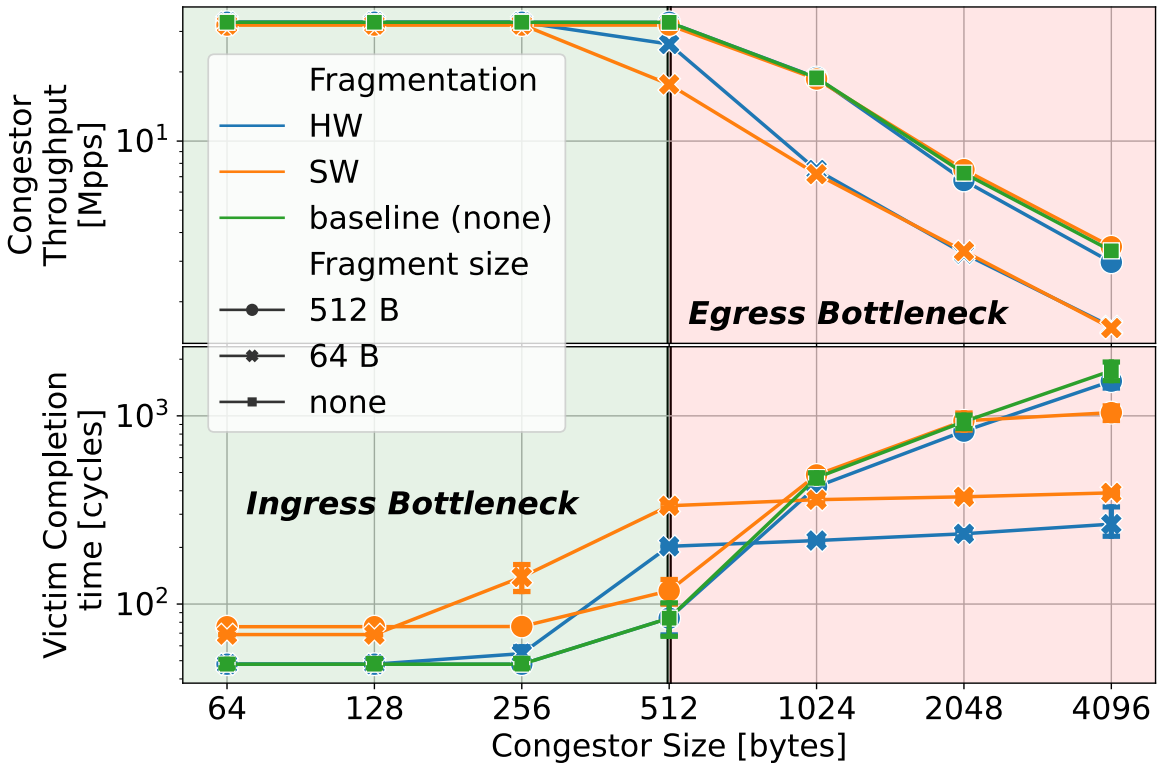
- Scale down number of used cycles according to priority





OSMOSIS IO management

- **WRR scheduling**
- **DMA request fragmentation**
  - Software: split large requests into smaller ones
  - Hardware: keep per-AXI-stream state



Evaluation

Experimental testbed

Synthesis with Synopsys Design Compiler NXT

- GlobalFoundries 22nm process

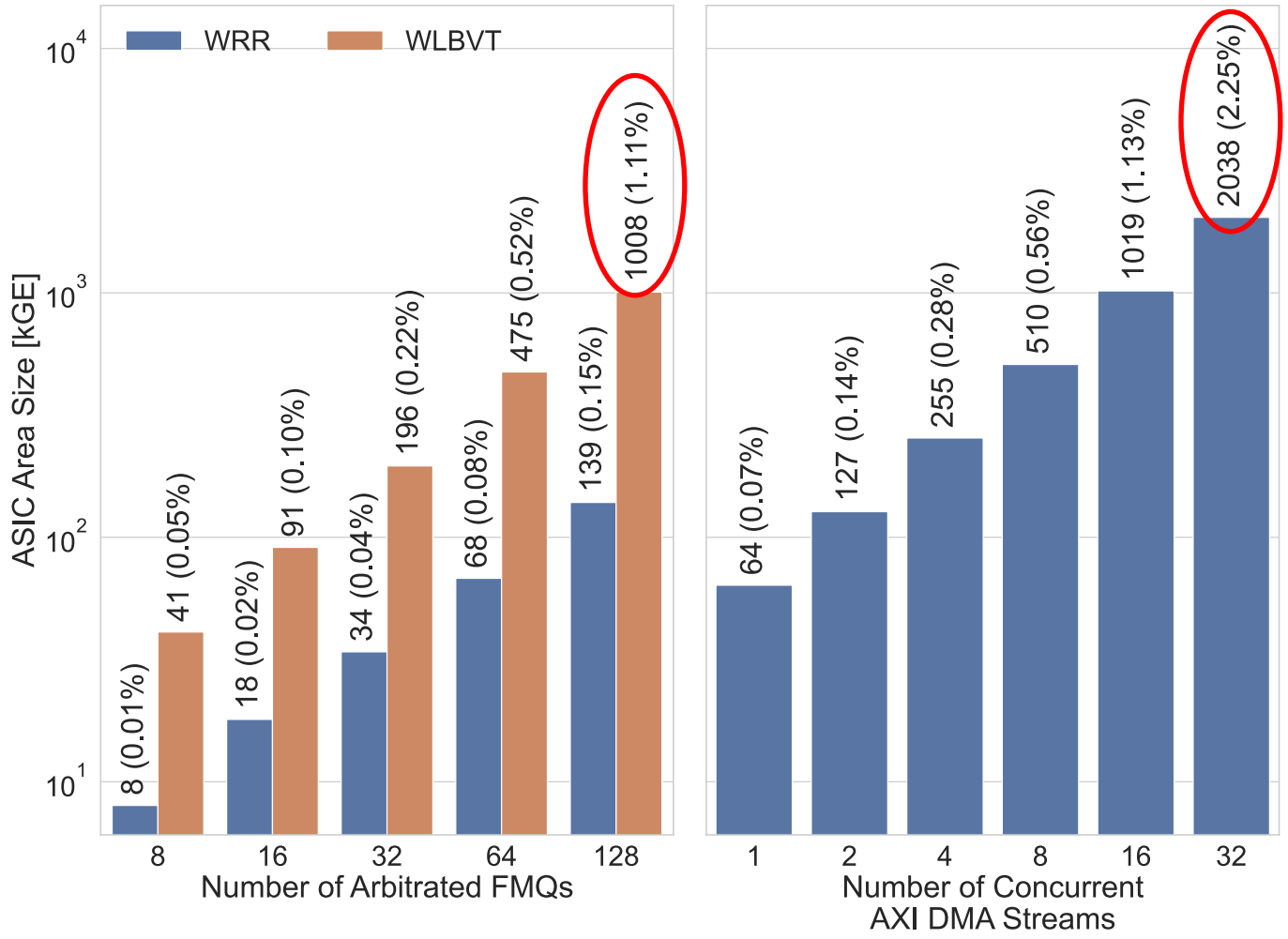
Cycle-accurate simulation with Verilator

- 32 1 GHz RISC-V PUs
- 400 Gbit/s ingress/egress link

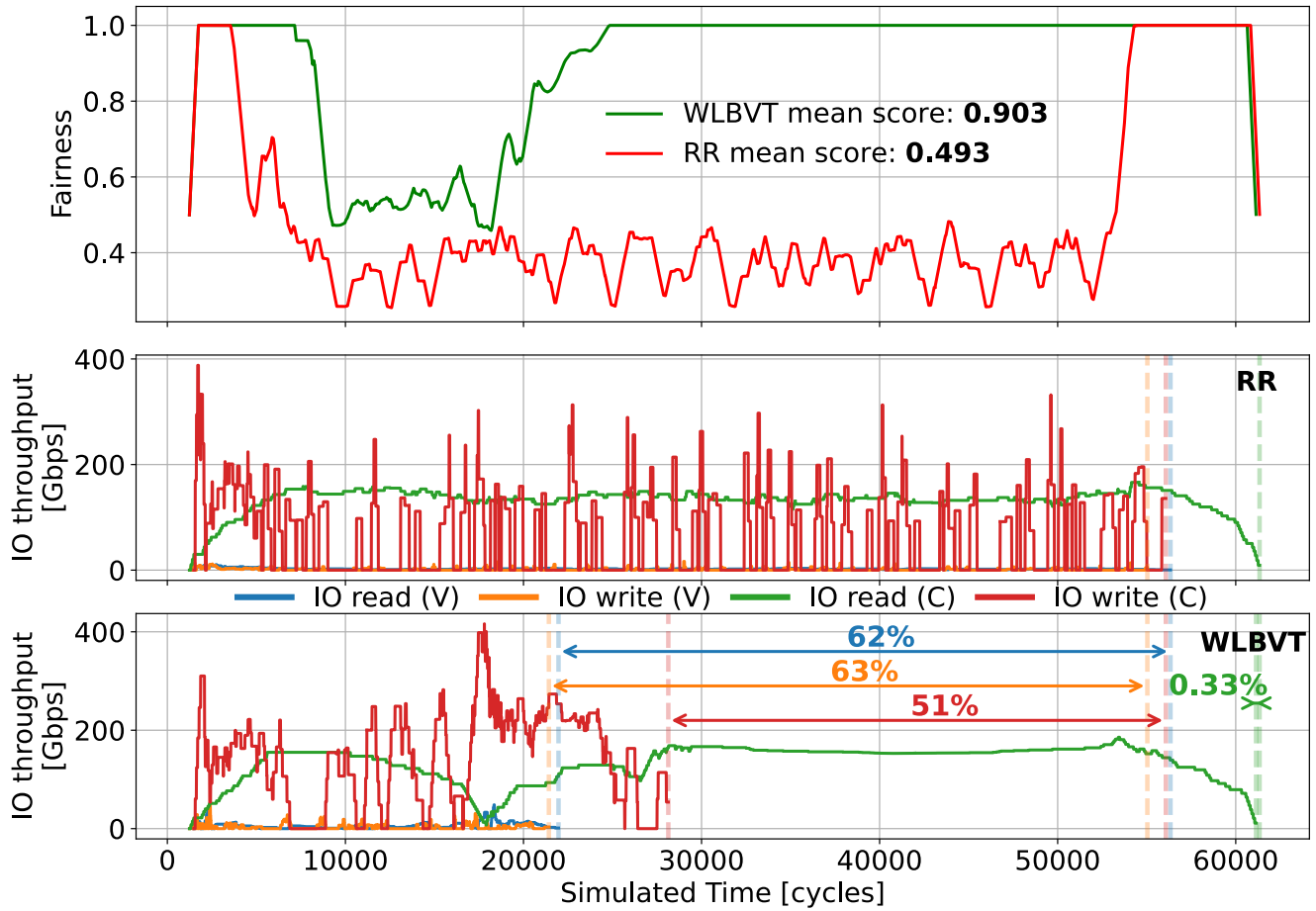
Compared systems

- Baseline vanilla PsPIN SmartNIC
- OSMOSIS-enhanced PsPIN

Hardware footprint analysis



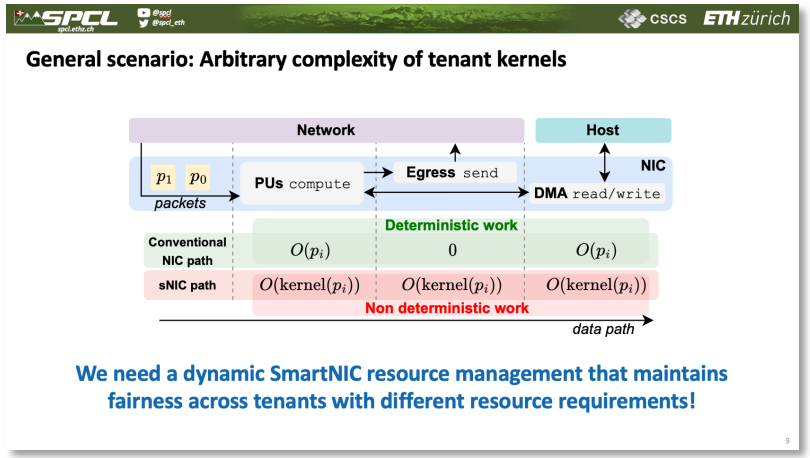
Resource isolation performance

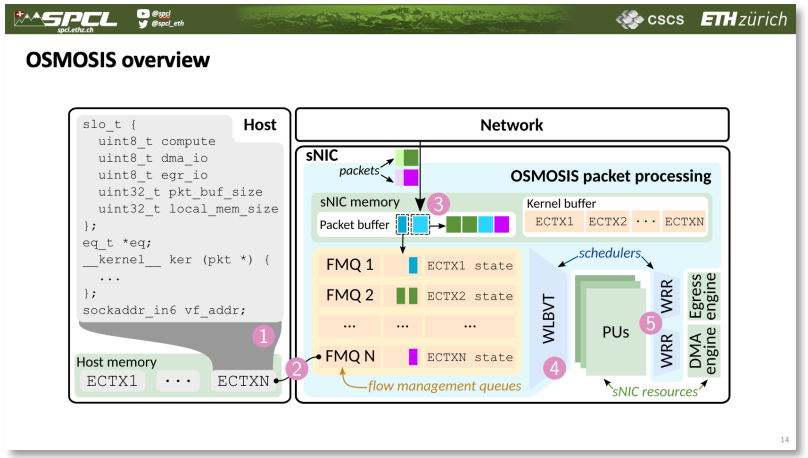


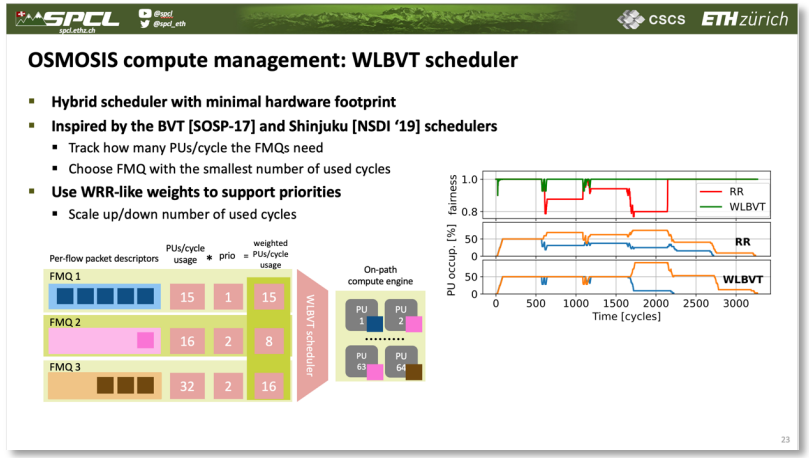
IO read = DMA read + Egress send
IO write = DMA write

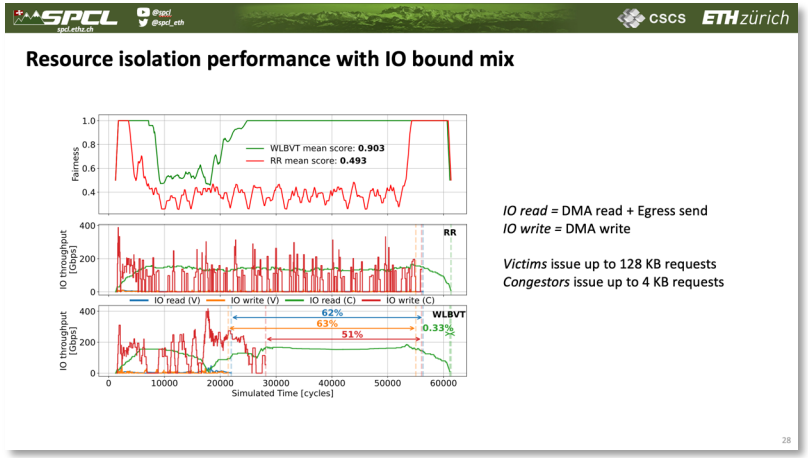
Victims issue up to 128 KB requests
Congestors issue up to 4 KB requests

Conclusions









More of SPCL's research:

youtube.com/@spcl — 180+ Talks

twitter.com/spcl_eth — 1.4K+ Followers

github.com/spcl — 3.8K+ Stars

… or spcl.ethz.ch

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

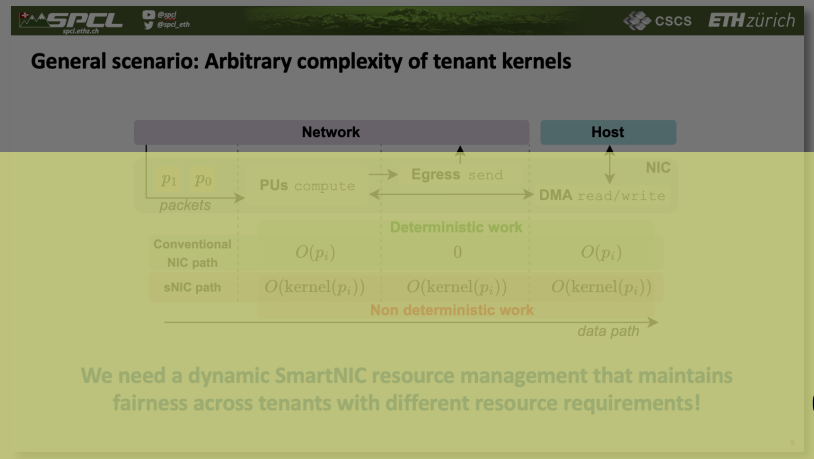
ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

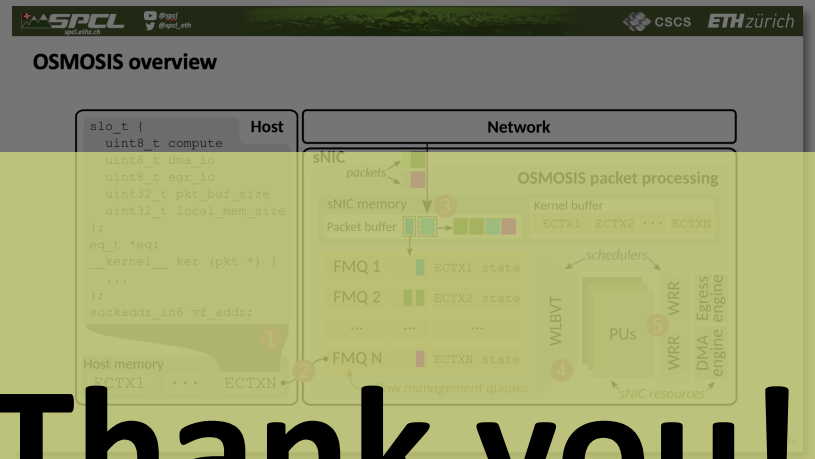
ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

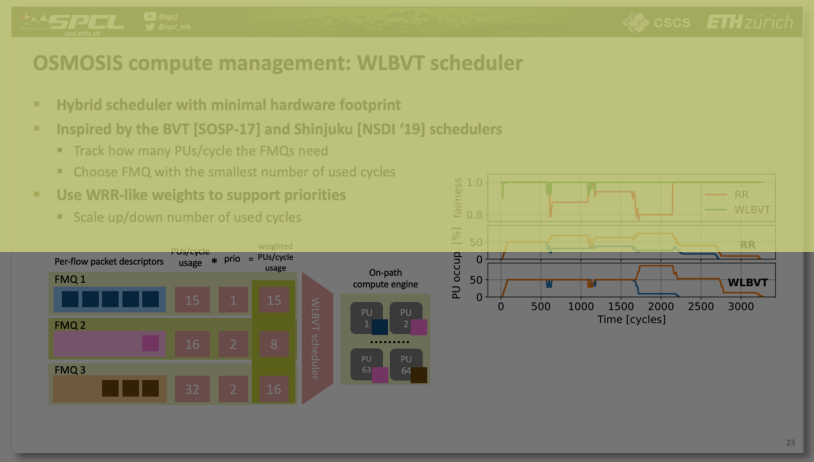
https://spclgitlab.ethz.ch/mkhalilov/pspin-osmosis

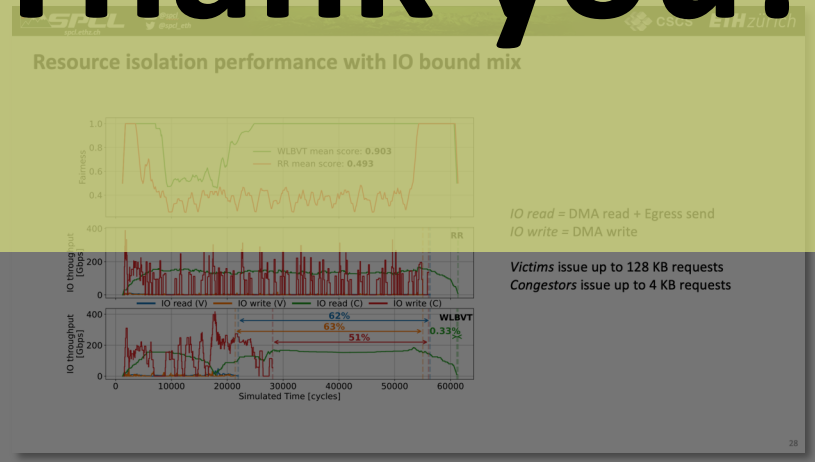
mikhail.khalilov@inf.ethz.ch

Conclusions









Thank you!

More of SPCL's research:

youtube.com/@spcl — 180+ Talks

twitter.com/spcl_eth — 1.4K+ Followers

github.com/spcl — 3.8K+ Stars

... or spcl.ethz.ch

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

https://spclgitlab.ethz.ch/mkhalilov/pspin-osmosis

mikhail.khalilov@inf.ethz.ch