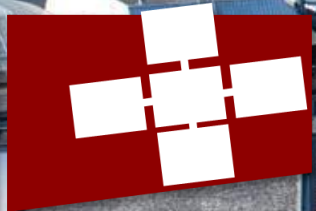


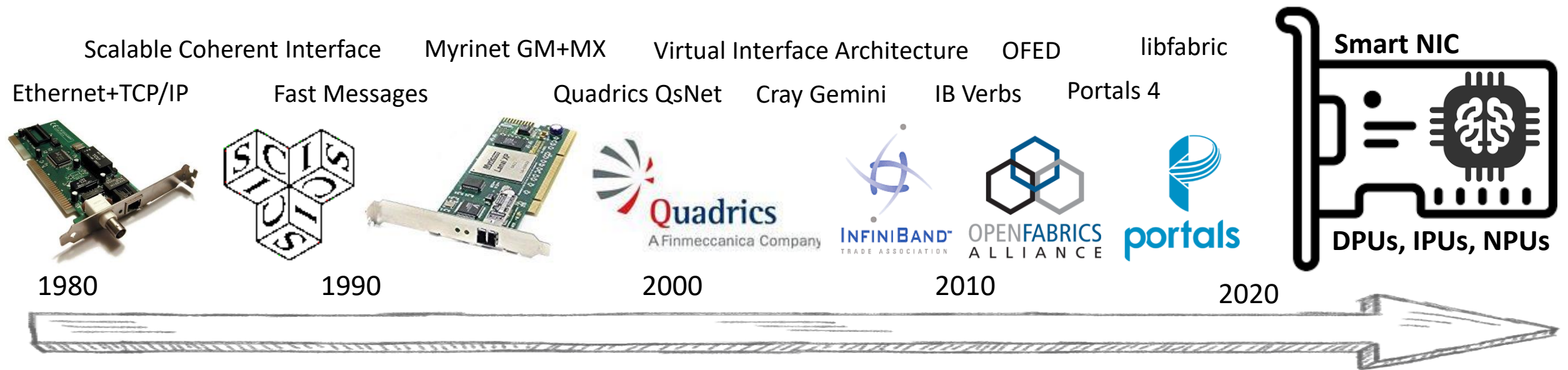
T. HOEFLER

WITH S. DI GIROLAMO, D. DE SENSI, K. TARANOV, L. BENINI, R. E. GRANT, R. BRIGHTWELL, A. KURTH, M. SCHAFFNER, T. SCHNEIDER, J. BERÁNEK, M. BESTA, D. ROWETH

New trends for sPIN-based in-network computing: from sparse reductions to RISC-V acceleration!



The Development of High-Performance Networking Interfaces



sockets

(active) message based

protocol offload

remote direct memory access (RDMA)

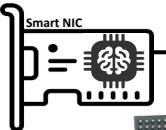


coherent memory access

OS bypass

zero copy

triggered operations



ARM cores
(with full OS, outside packet pipe)

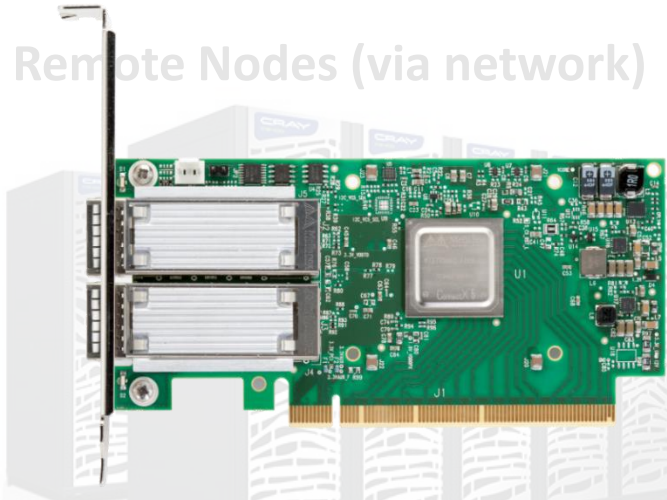
Flow Processors
(limited flexibility, P4)



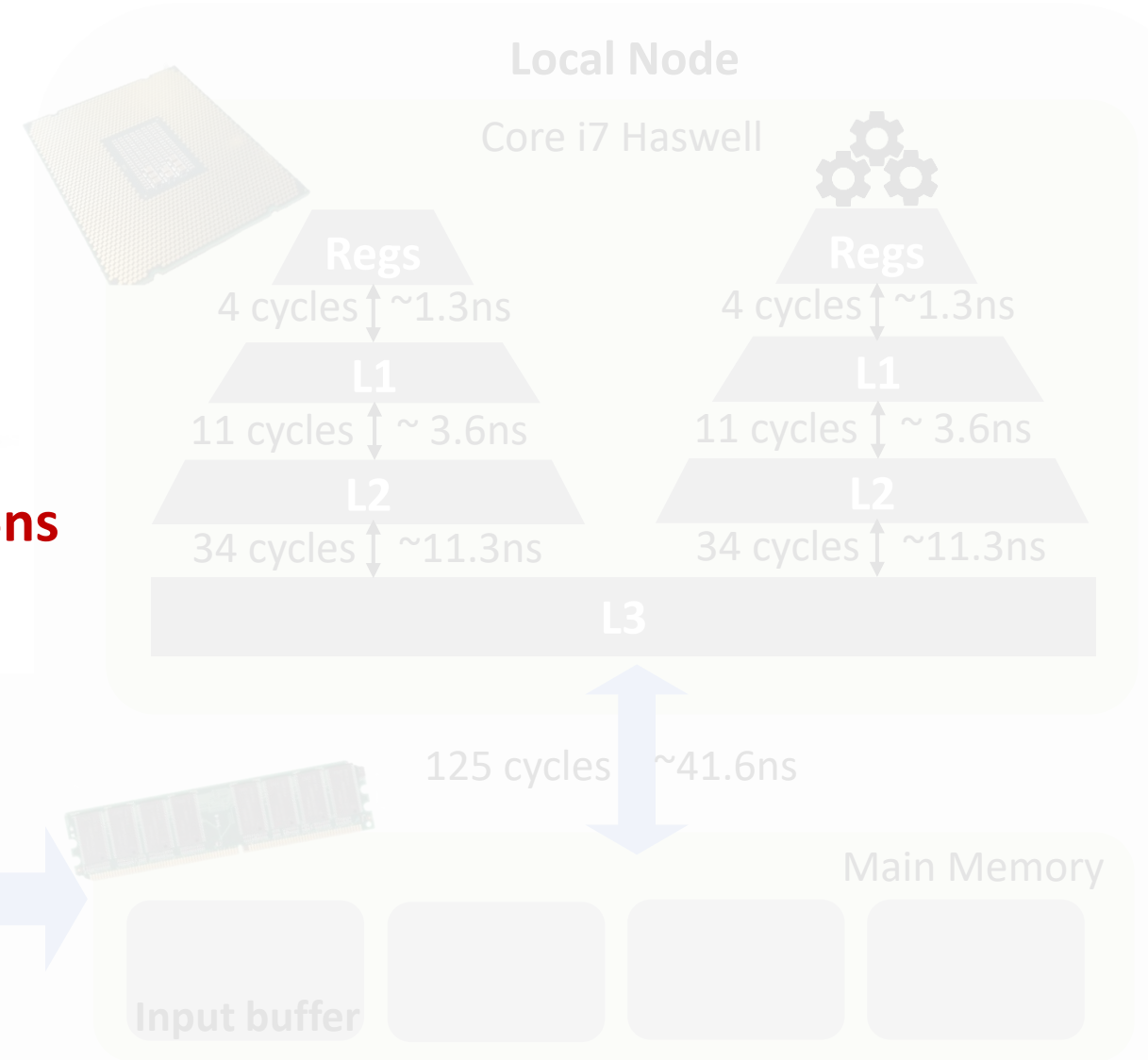
FPGAs
(limited productivity,
silicon efficiency)



Data Processing in modern RDMA networks

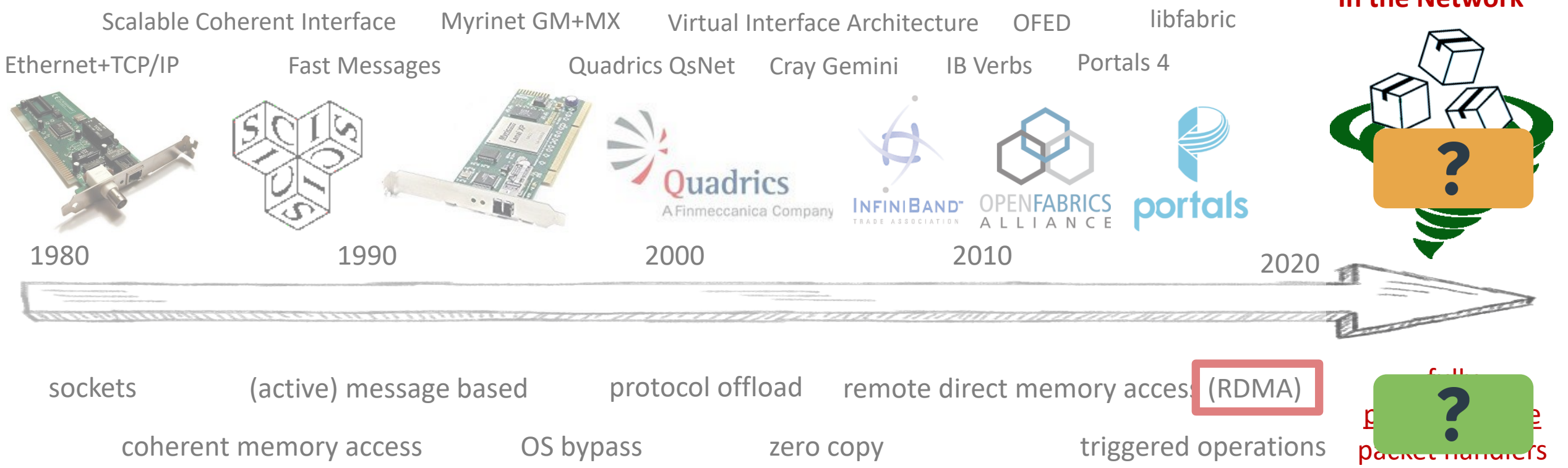


Mellanox ConnectX-6: 1 packet/2.5ns
Tomorrow (400G): 1 packet/1.2ns



The future of High-Performance Networking Interfaces

SPIN
Streaming Processing
In the Network



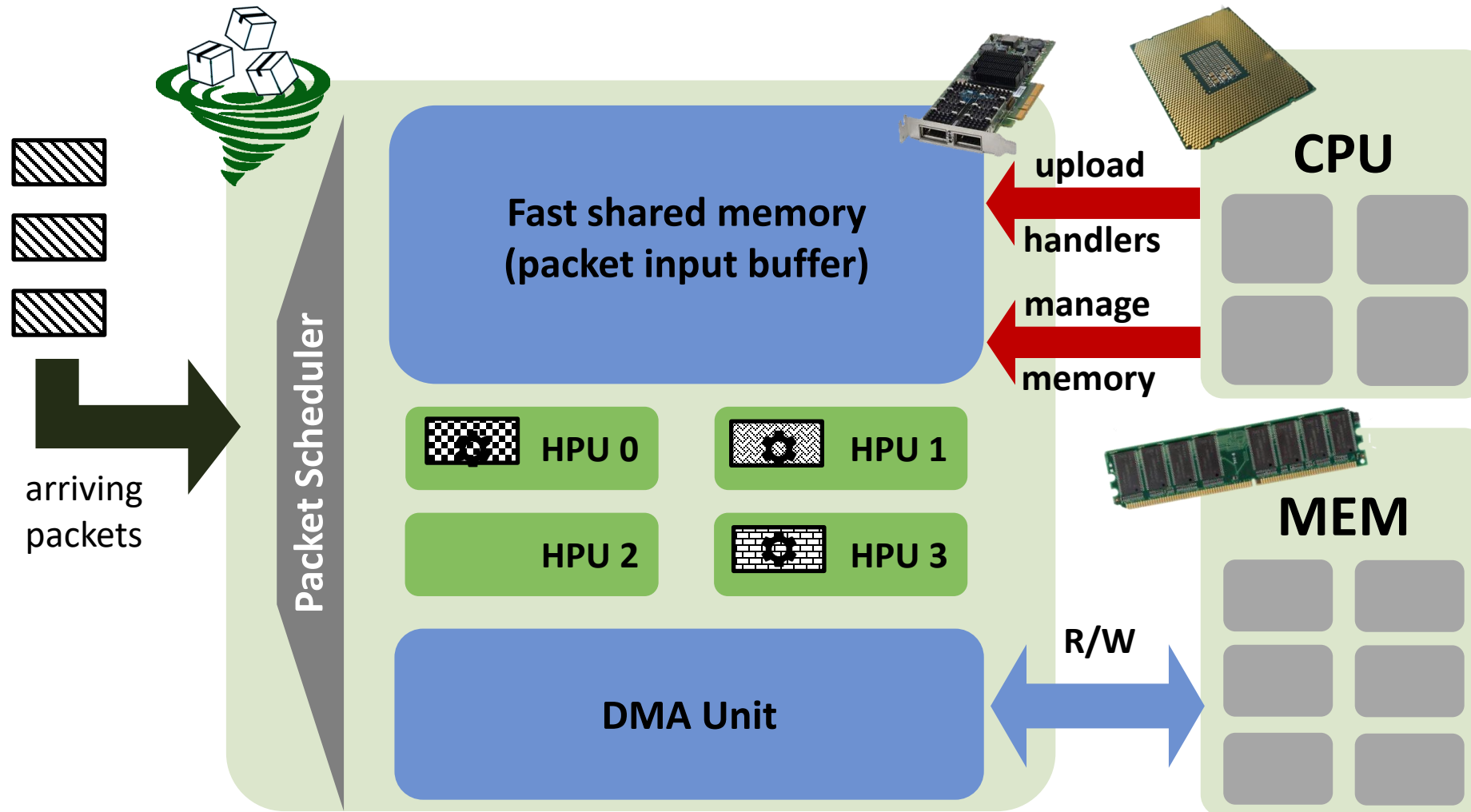
Established Principles for Compute Acceleration

OpenGL Microsoft DirectX11 Generalization Revolutionizes Acceleration NVIDIA CUDA OpenMP 4.0 OpenCL

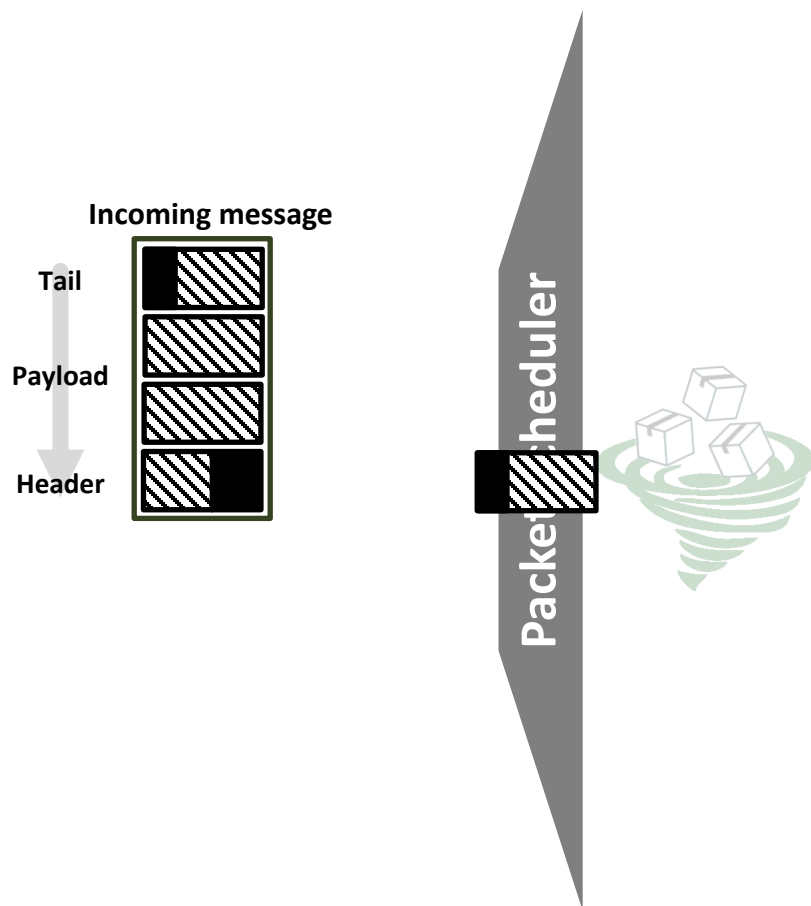
Where do we stand in Network Acceleration?

P4 eBPF Data Acceleration Generalization RISC-V

sPIN NIC - Abstract Machine Model for Packet Processing



sPIN – Programming Interface



Header handler

```
__handler int pp_header_handler(const ptl_header_t h, void *state) {
    pingpong_info_t *i = state;
    i->source = h.source_id;
    return PROCESS_DATA; // execute payload handler to put from device
}
```

Payload handler

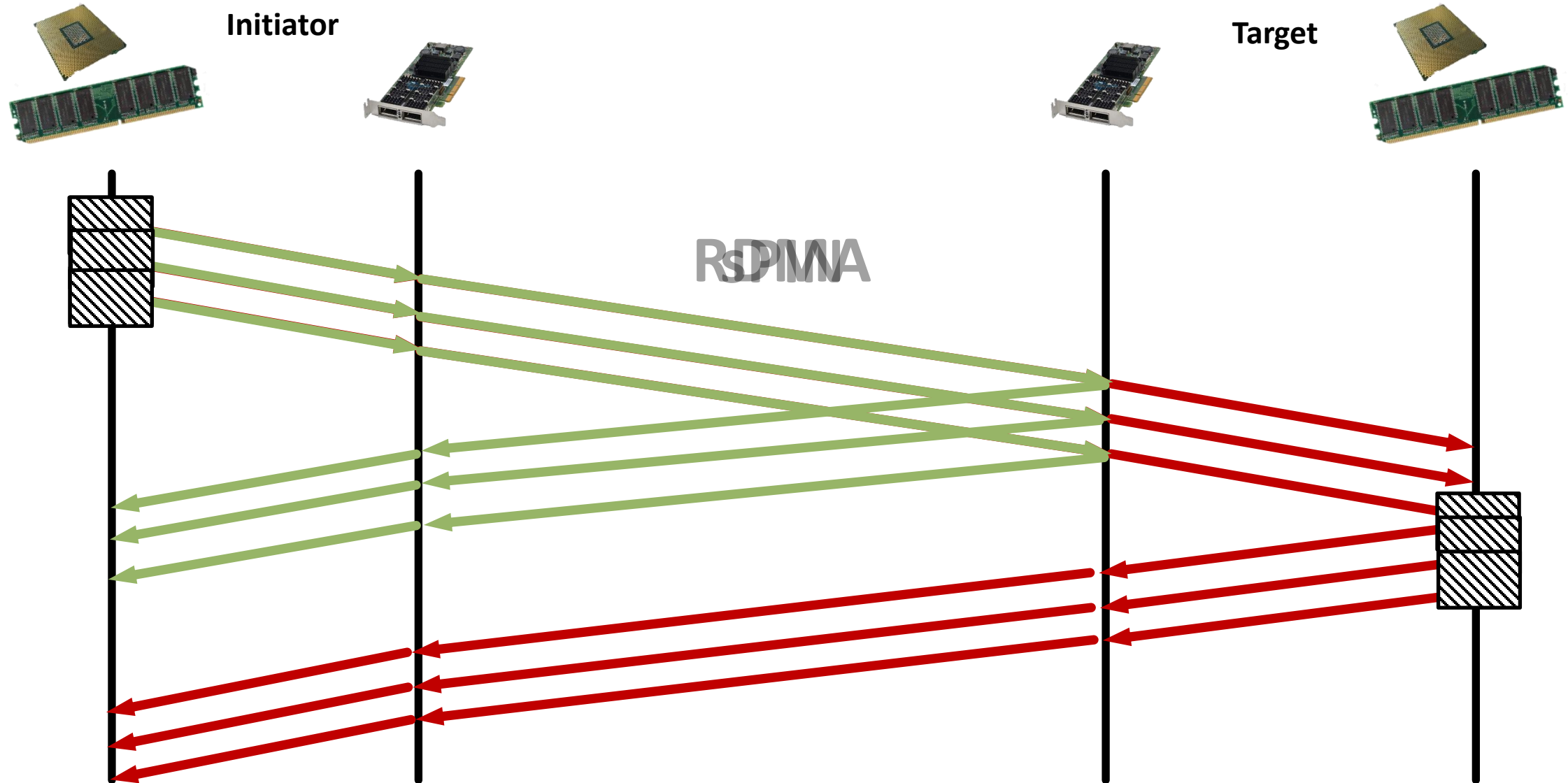
```
__handler int pp_payload_handler(const ptl_payload_t p, void *state) {
    pingpong_info_t *i = state;
    PtlHandlerPutFromDevice(p.base, p.length, 1, 0, i->source, 10, 0, NULL, 0);
    return SUCCESS;
}
```

Completion handler

```
__handler int pp_completion_handler(int dropped_bytes,
                                     bool flow_control_triggered, void *state) {
    return SUCCESS;
}
```

```
connect(peer, /* ... */, &pp_header_handler, &pp_payload_handler, &pp_completion_handler);
```

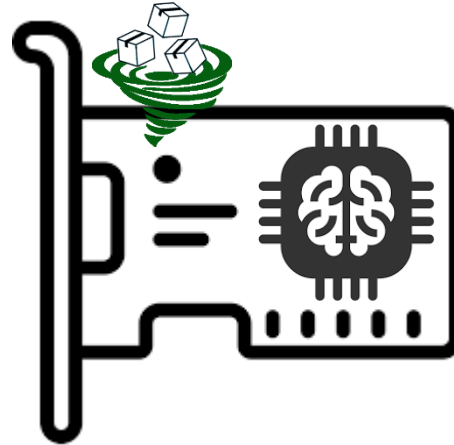
RDMA vs. sPIN in action: Streaming Ping Pong



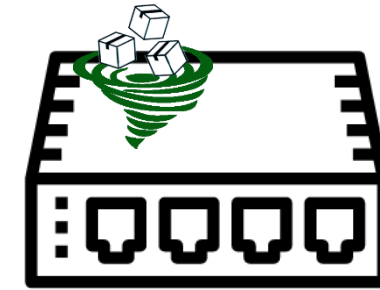
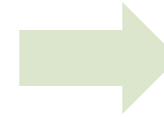
Talk roadmap



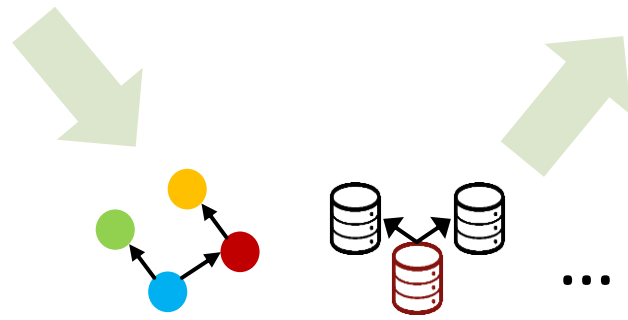
Motivation and Overview



Hardware Implementation



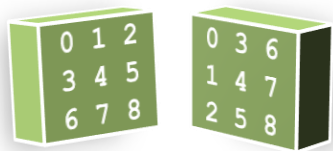
In-network reductions



Network Group Communication Distributed Data Management

Use cases

In-network compute use cases



Network-accelerated datatypes



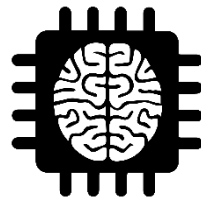
sPIN-FS



Zoo-sPINNER
consensus protocols



Erasure coding



Quantization
Allreduce and other collectives

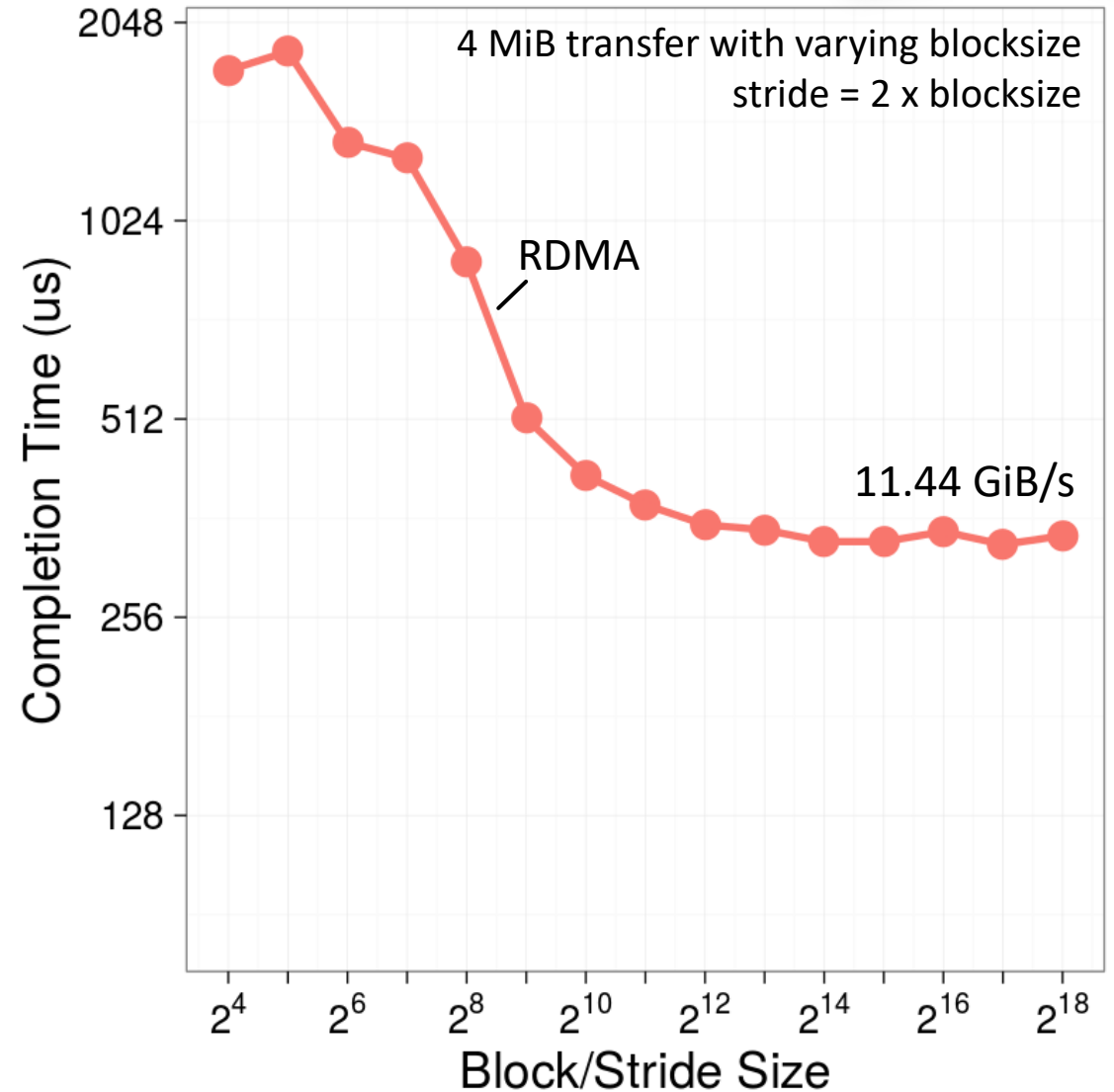
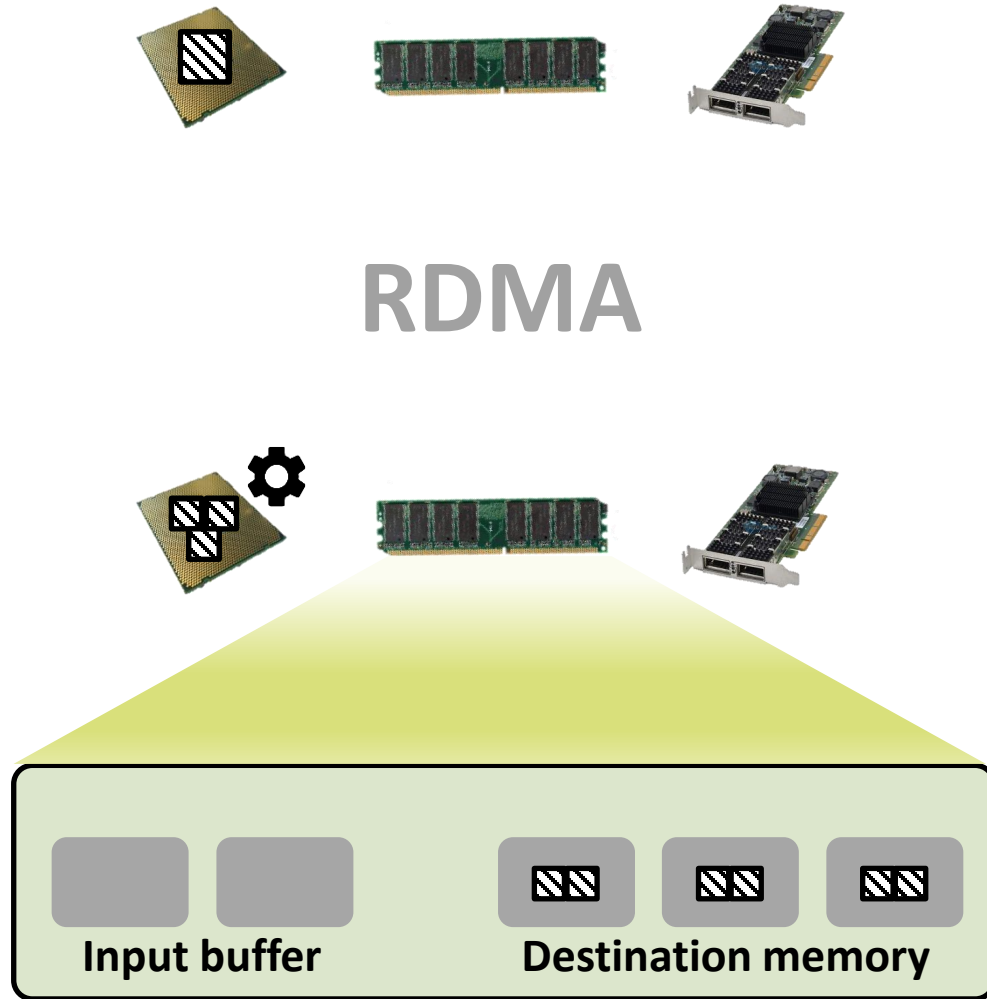
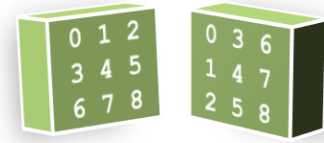


Packet classification and pattern matching

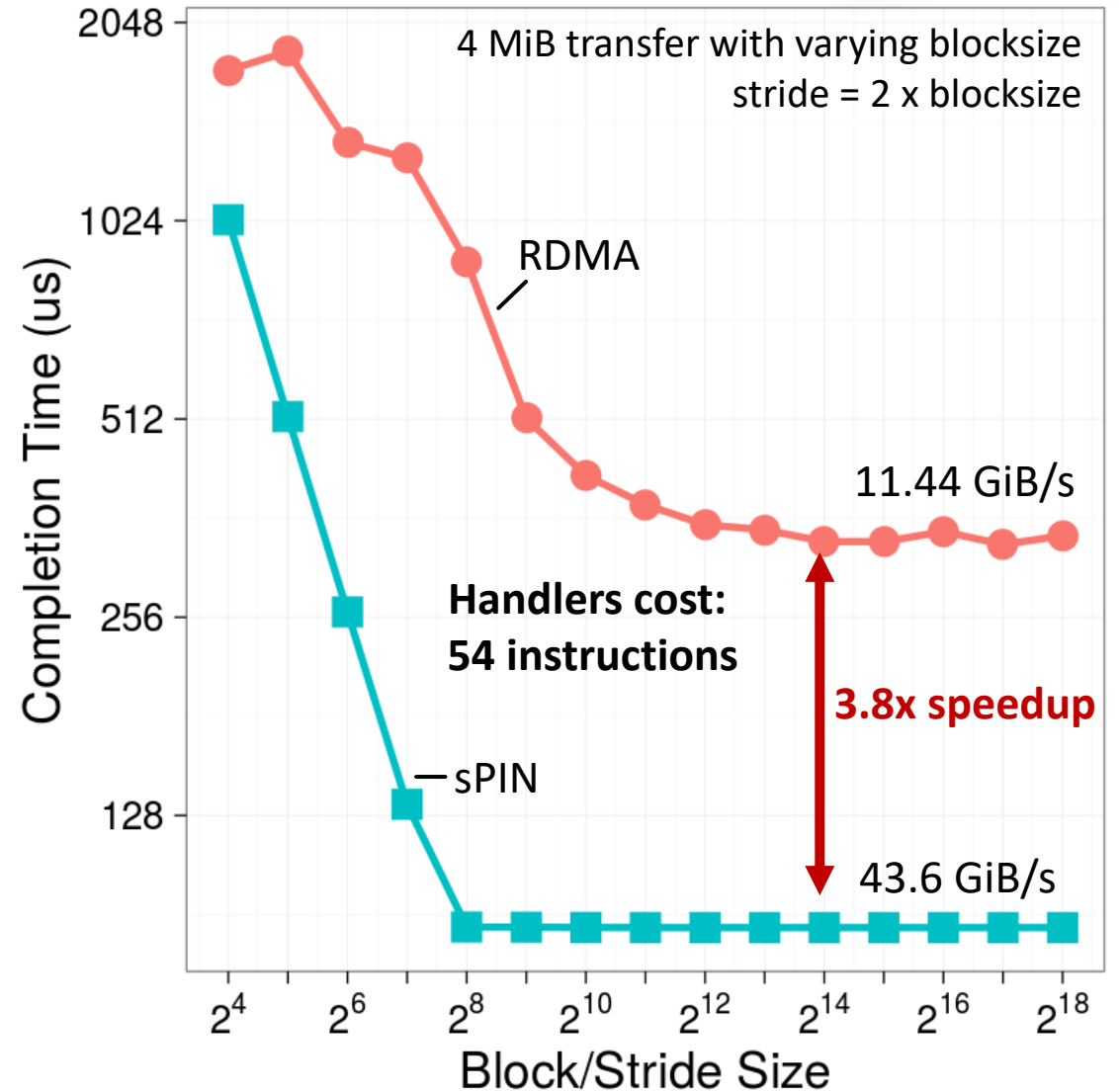
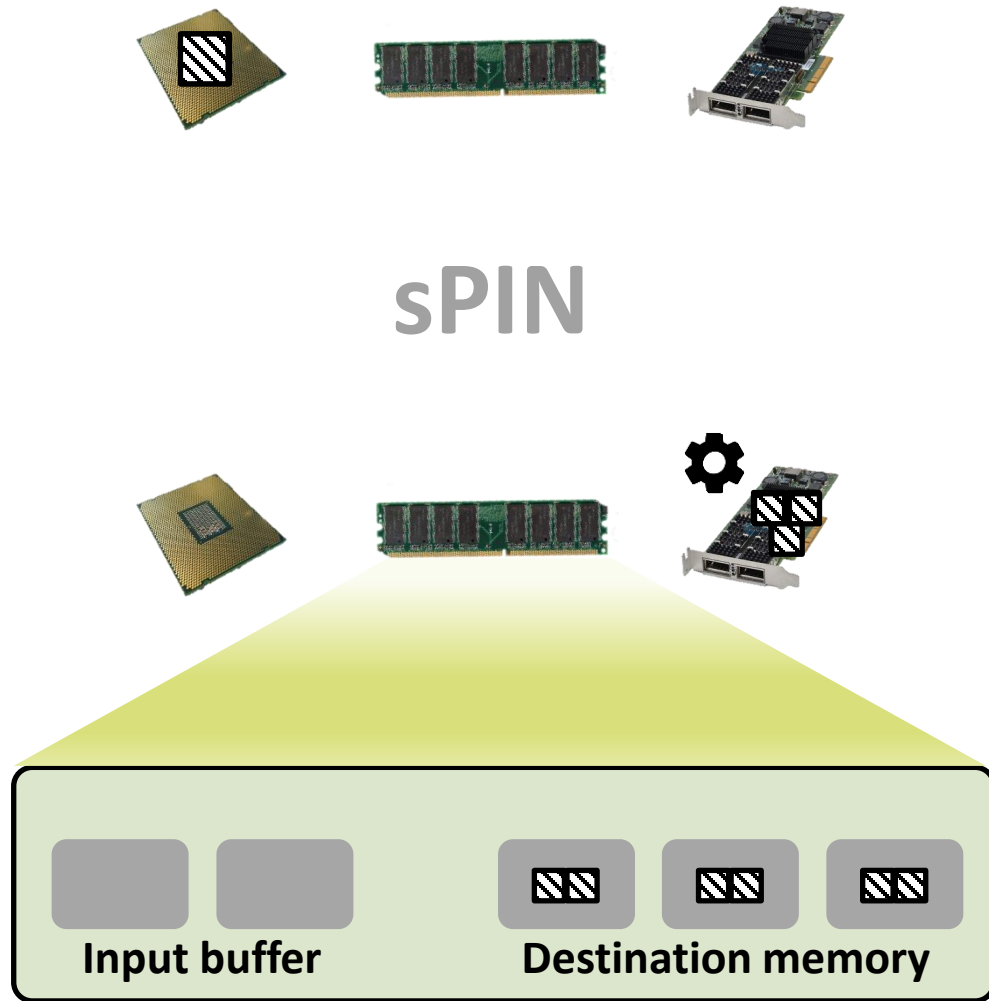
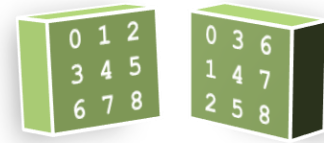


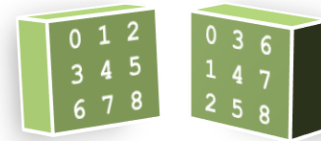
Serverless

MPI Datatypes acceleration

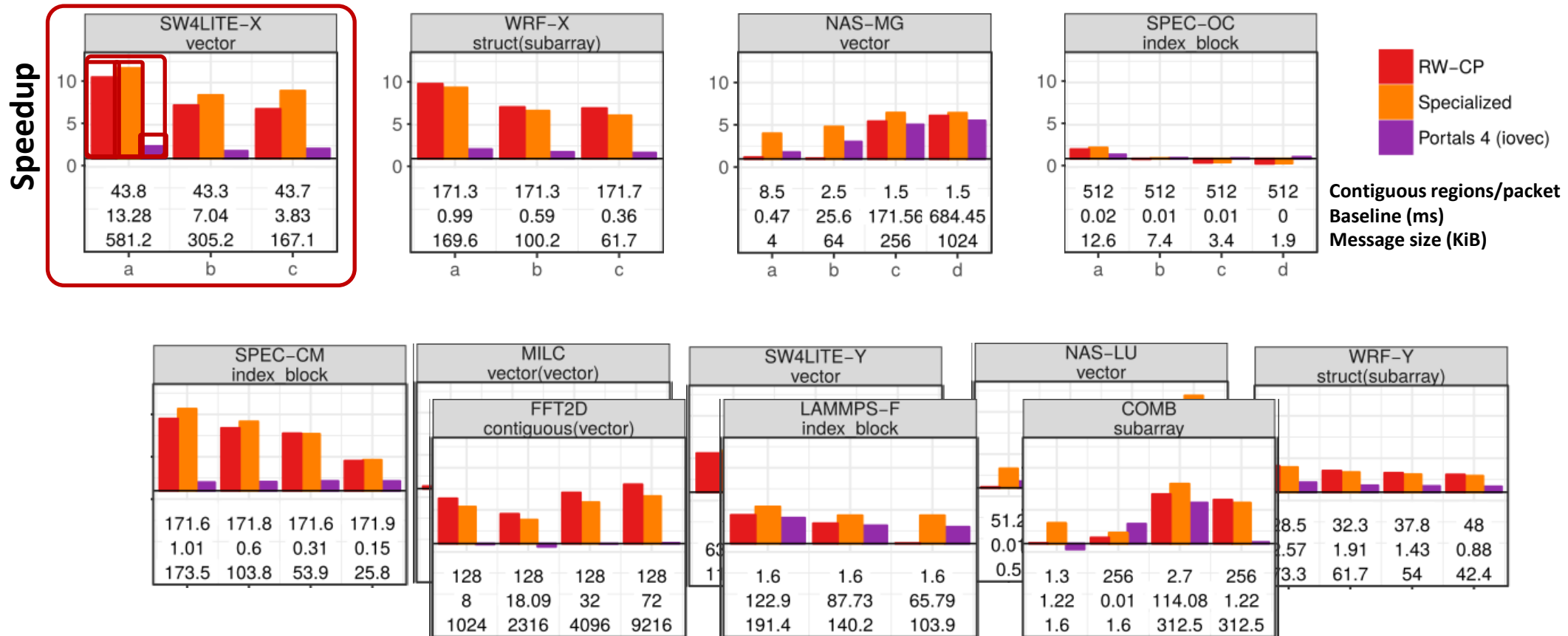


MPI Datatypes acceleration



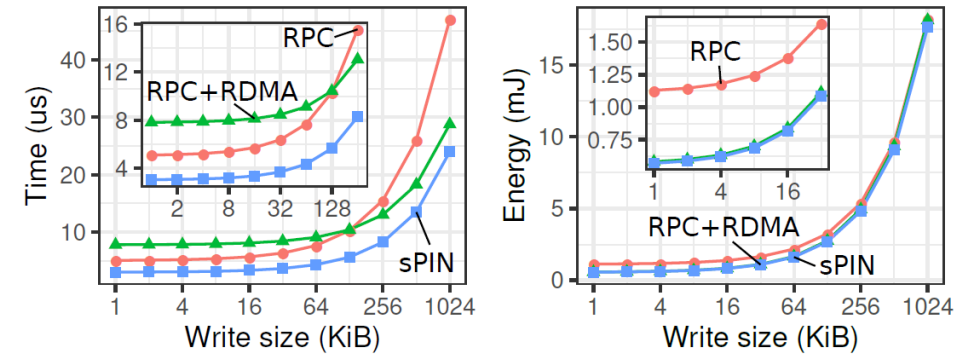


Real Applications DDTs



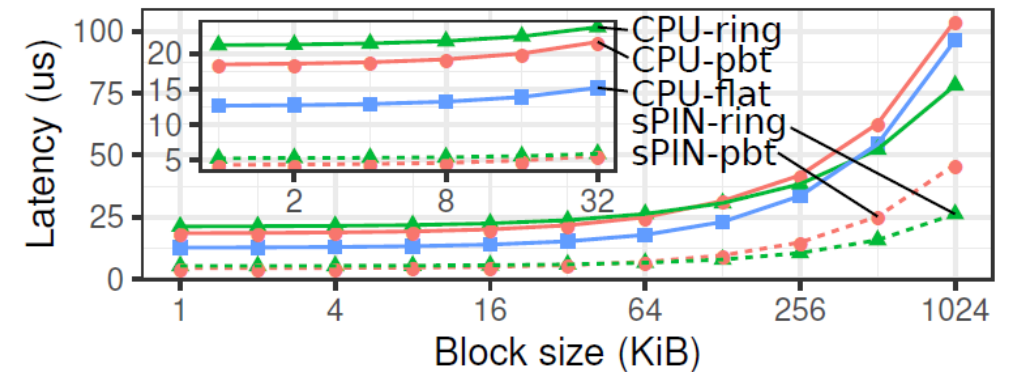
Fully one-sided & secure data access

up to 40% faster for small writes (1 KiB) and up to 16% for large ones (1 MiB)



Network-offloaded data replication

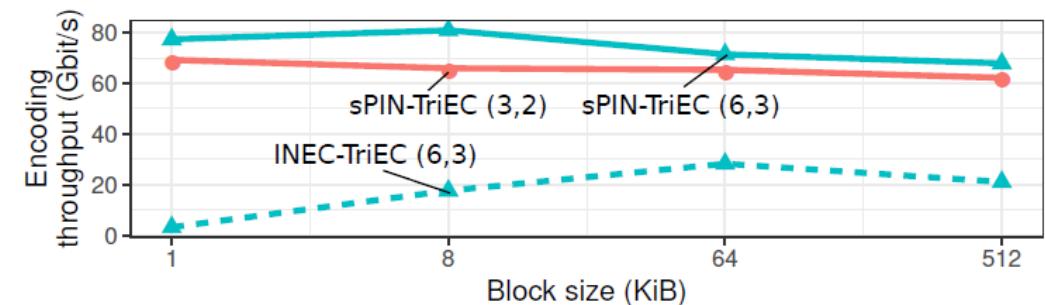
up to 4x for small writes (1 KiB) and up to 3.15x for large ones (1 MiB)



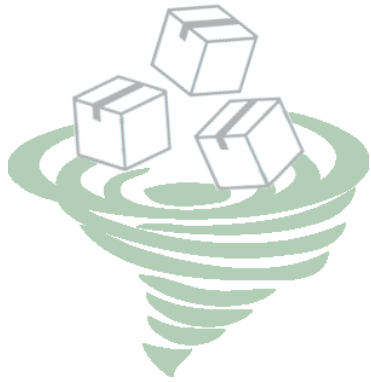
Network-offloaded erasure coding

up to 29x and 3.3x better bandwidth for 1 KiB and 512 KiB blocks w.r.t. INEC-TriEC

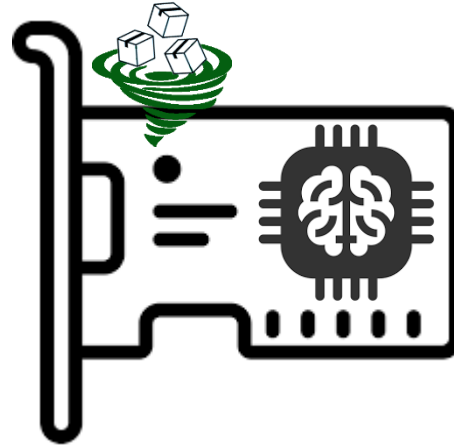
Bonus: sPIN can reduce time-to-market of new solutions like TriEC!



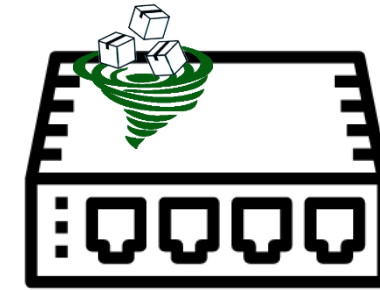
Talk roadmap



Motivation and Overview



Hardware Implementation



In-network reductions



Network Group
Communication

Distributed Data
Management

Use cases

Architectural principles for in-network compute



Low latency,
full throughput



Support for wide range
of use cases



Easy to integrate

Interconnection Design Considerations

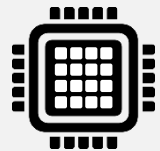
Requirements	Descriptions
Diversity	<ul style="list-style-type: none"> Various types of objects to connect : xPU, SSDs, NICs, HDDs, Various link distances : on die, on package, on board, inside box, in cluster, in data center Rich functions : cache coherency, peripheral semantics, network semantics, application semantics Cost and power consumption constraints: pJ/bit, \$/Gbps
Low Latency	<ul style="list-style-type: none"> Lowest latency under ideal conditions and Long Tail Effects in practical scenarios Network latency and latency inside box
High Bandwidth	<ul style="list-style-type: none"> NIC bandwidth , bandwidth inside box , cross-section bandwidth Complex network topology, routing, and congestion control Connection type and bandwidth determine the total cost
Low overhead	<ul style="list-style-type: none"> No processor resource occupation : software overhead, software delay, OS delay



Architectural principles for in-network compute



Low latency, full throughput



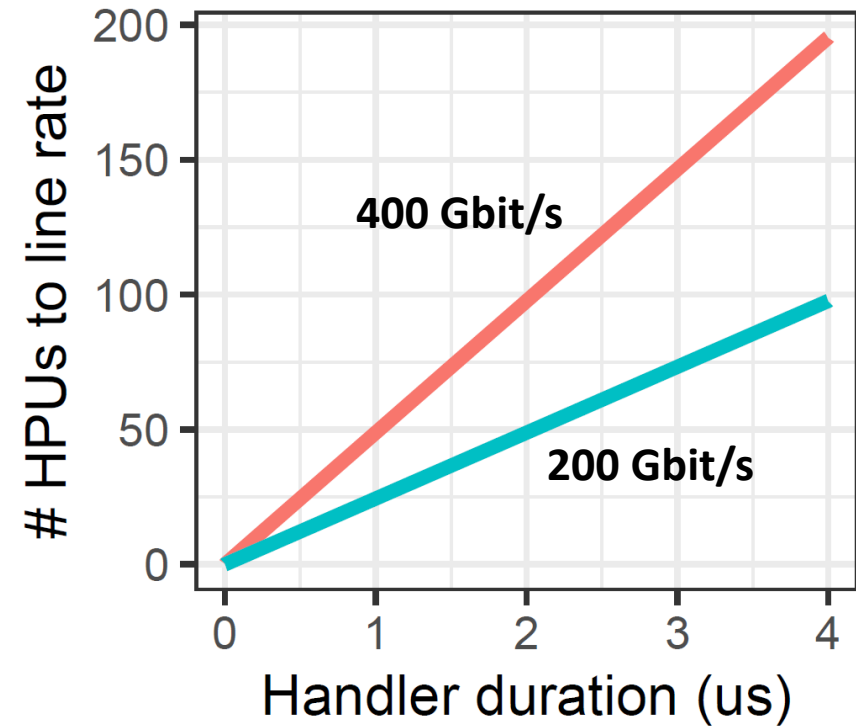
Highly parallel



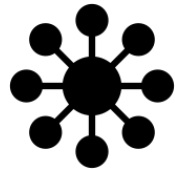
Fast scheduling



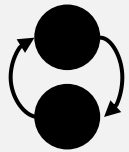
Fast explicit
memory access



Architectural principles for in-network compute



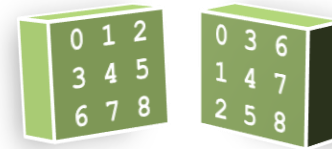
Support for wide range of use cases



Stateful computation support



Handlers isolation



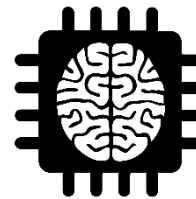
Network-accelerated datatypes [1]



SPIN-FS



Zoo-sPINNER
consensus protocols



Quantization
Allreduce and other collectives



Packet classification and pattern matching



Serverless



Erasure coding

Architectural principles for in-network compute



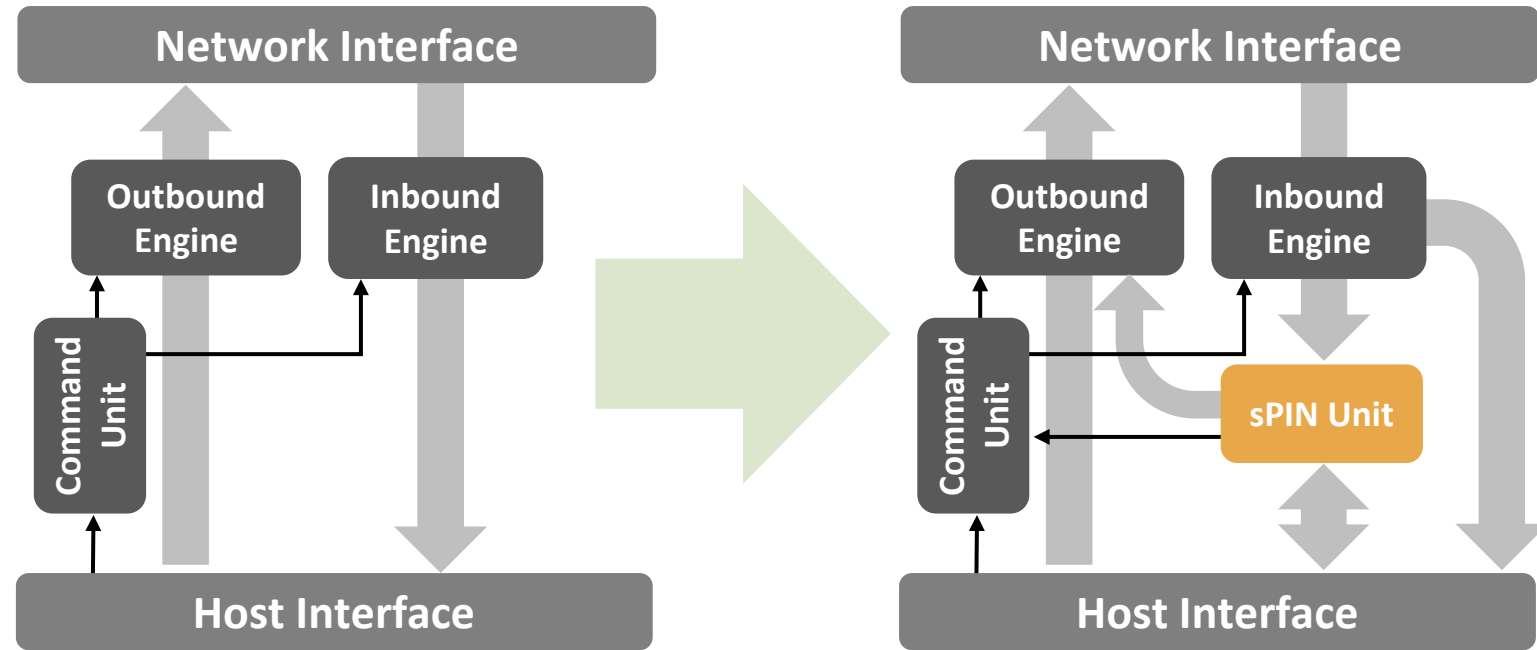
Easy to integrate



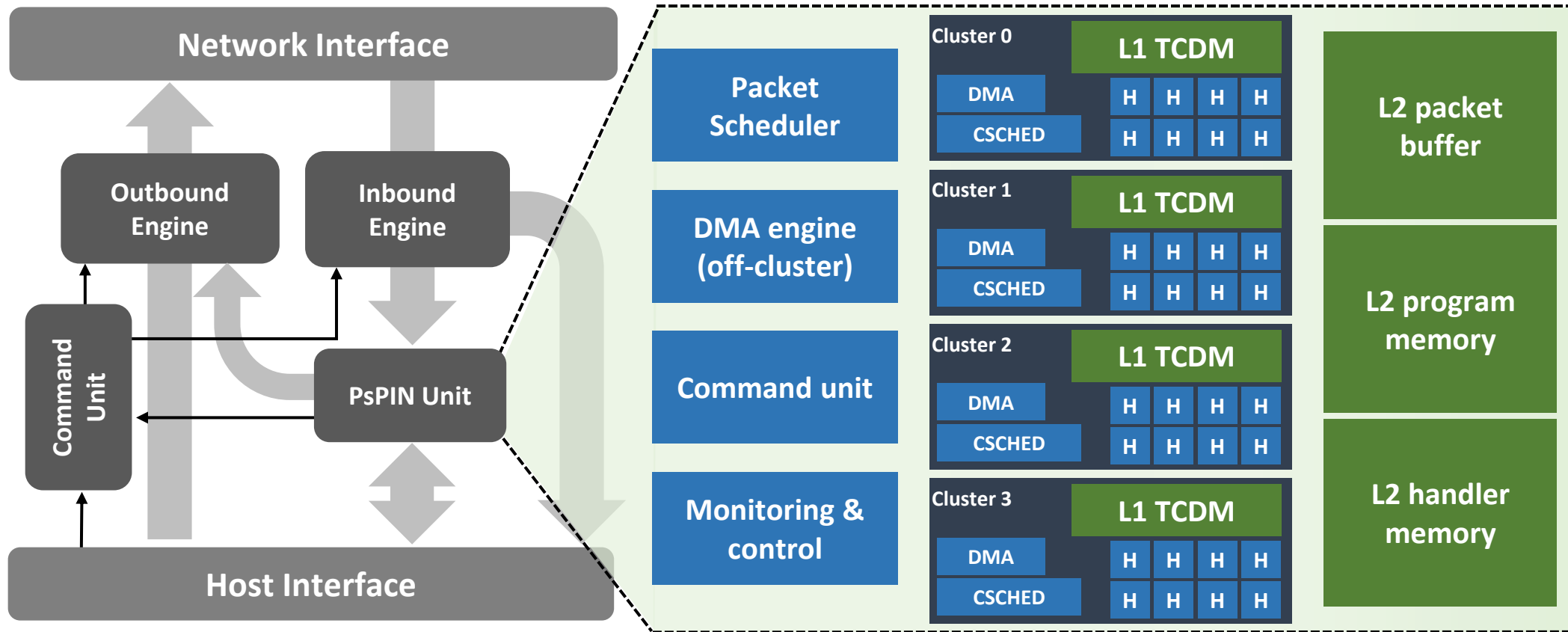
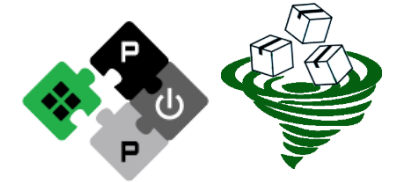
Area and power efficiency



Configurability



PsPIN: A PULP-powered implementation of sPIN



Application perspective

1 Define and offload handlers

2 Define an execution context

Execution context: EC_filter
 filter_hh(), filter_ph(), filter_th();
 NIC memory: **STATE**
 Host buffer: **BUF**

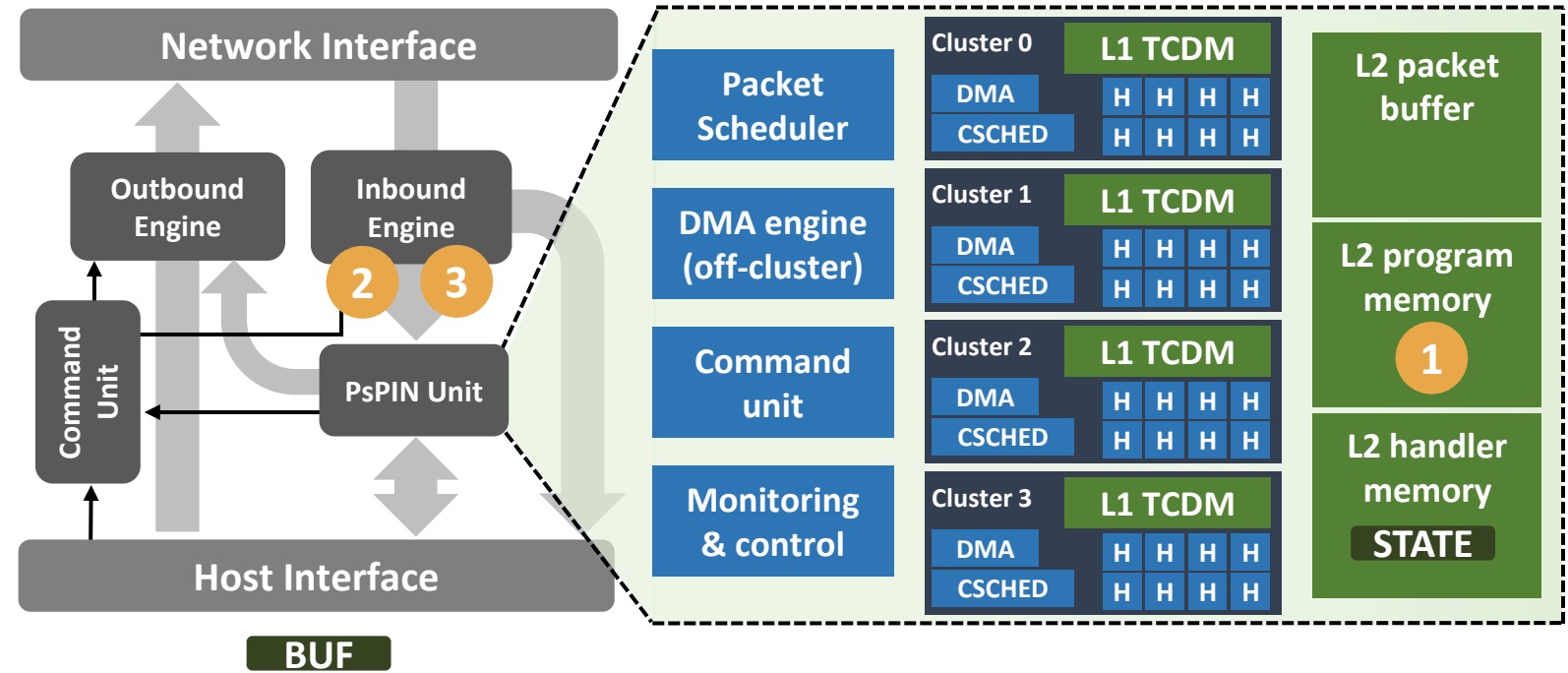
3 Define matching rule:
 e.g., (IP packets) -> EC_filter

Telemetry:

telemetry_hh(), telemetry_ph(), telemetry_th();

Filtering:

filter_hh(), filter_ph, filter_th();



Network perspective

- 1 Match packet to execution context
e.g., (IP packets) -> EC_filter

- 2 Write **PKT** to L2 pkt buffer
and inform PsPIN of the new
packet to process

- 3 Schedule the packet to a cluster
(task: pkt pointer, handler fun)

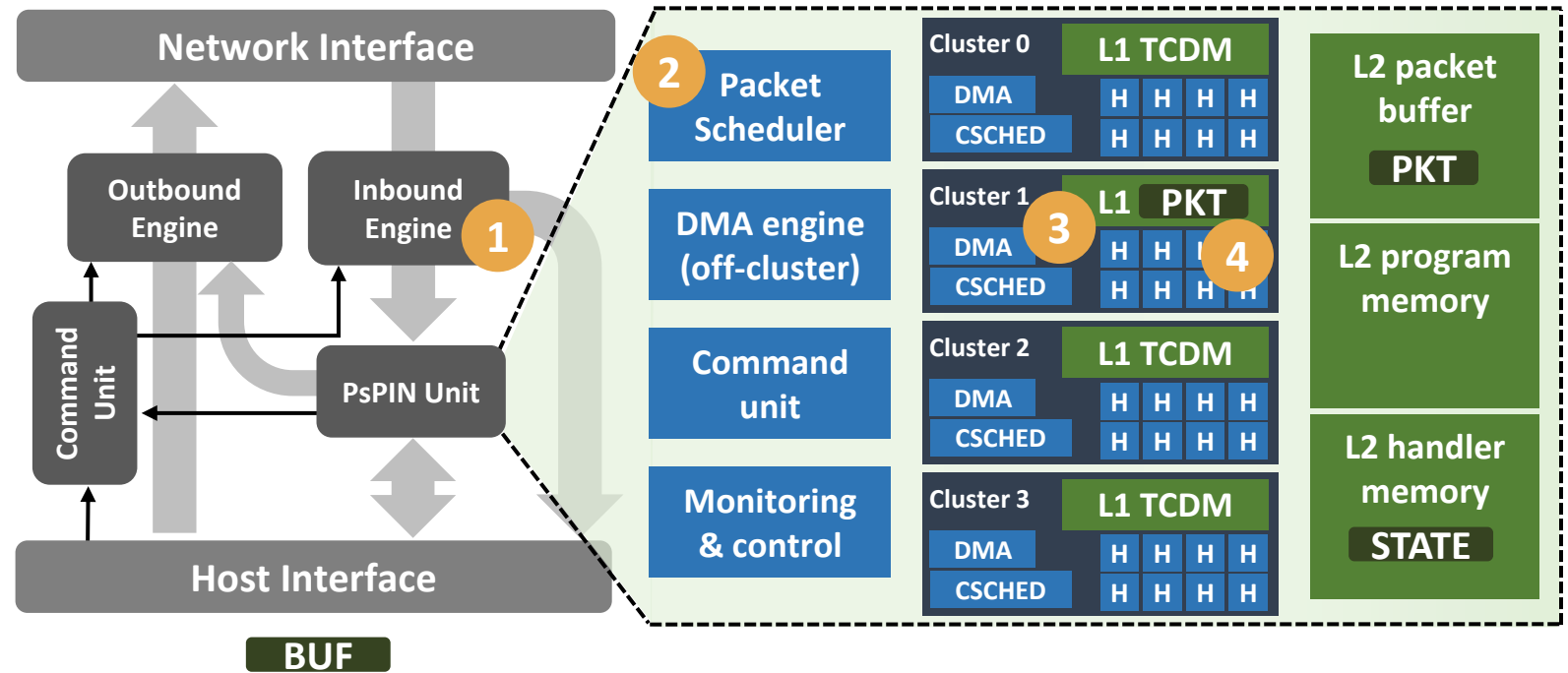
- 4 Copy packet to L1 and run
the handler

Execution context: EC_filter

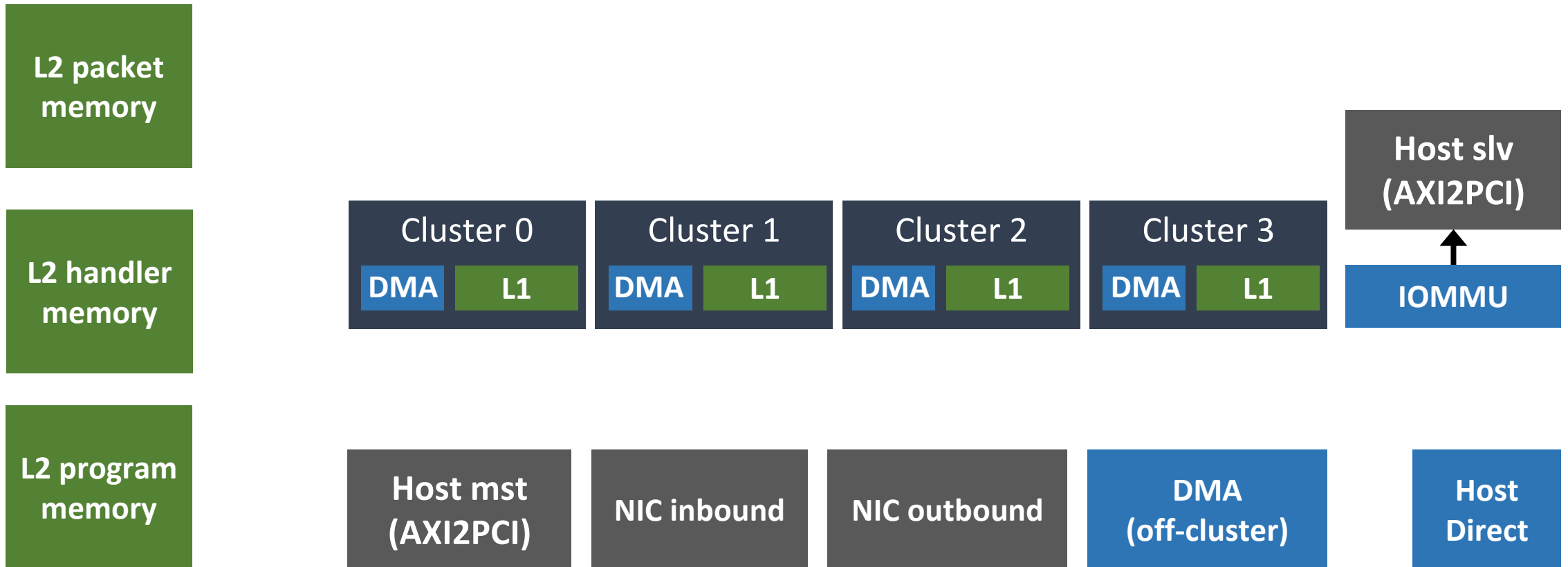
filter_hh(), filter_ph(), filter_th();
 NIC memory: **STATE**
 Host buffer: **BUF**

Scheduling overhead:

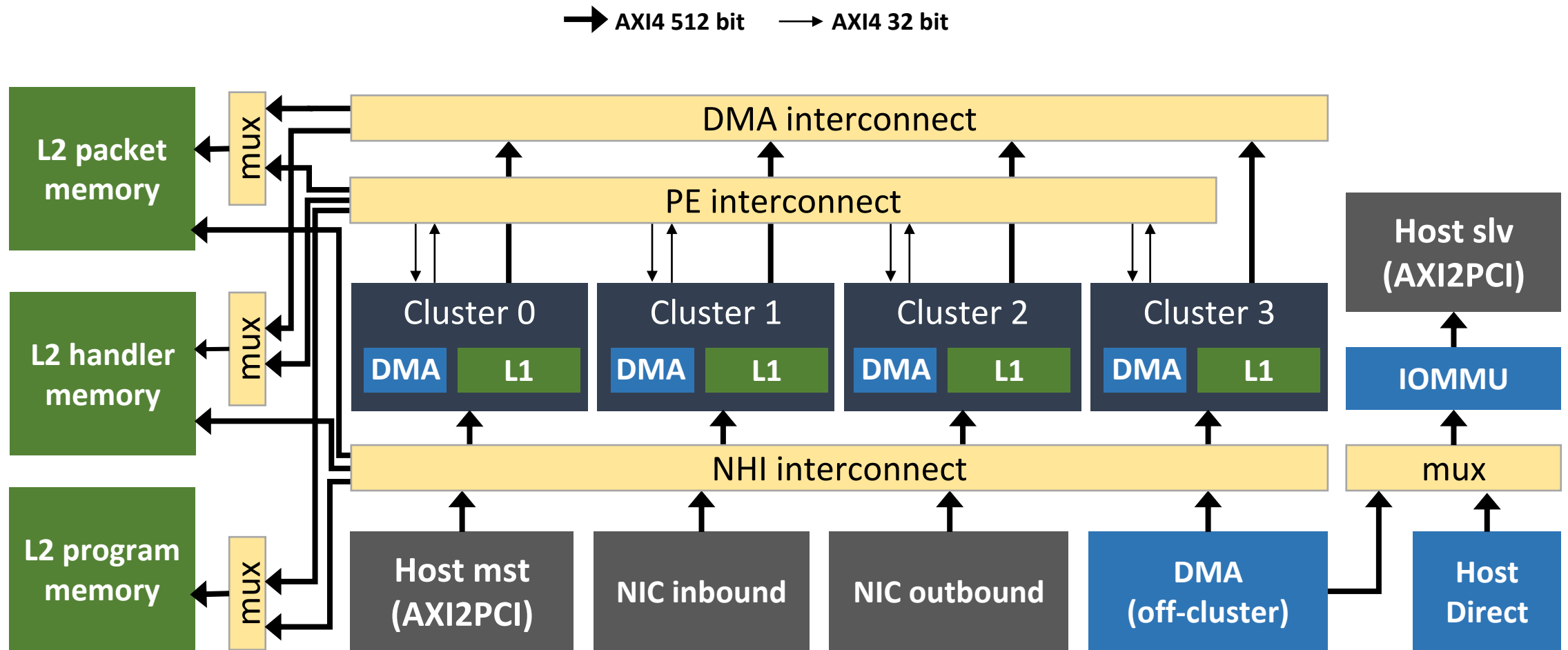
- 64 B packets: 12 ns
- 1 KiB packets: 26 ns



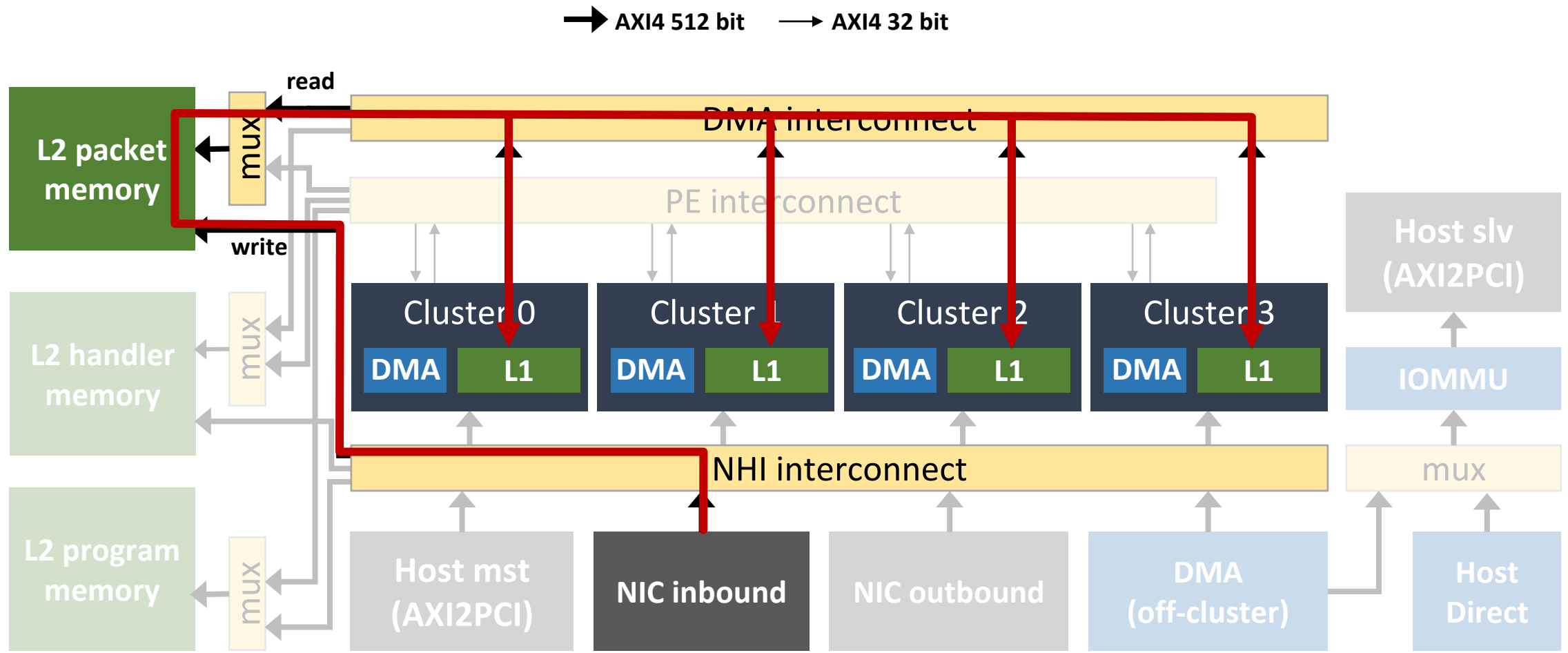
400G Data Path



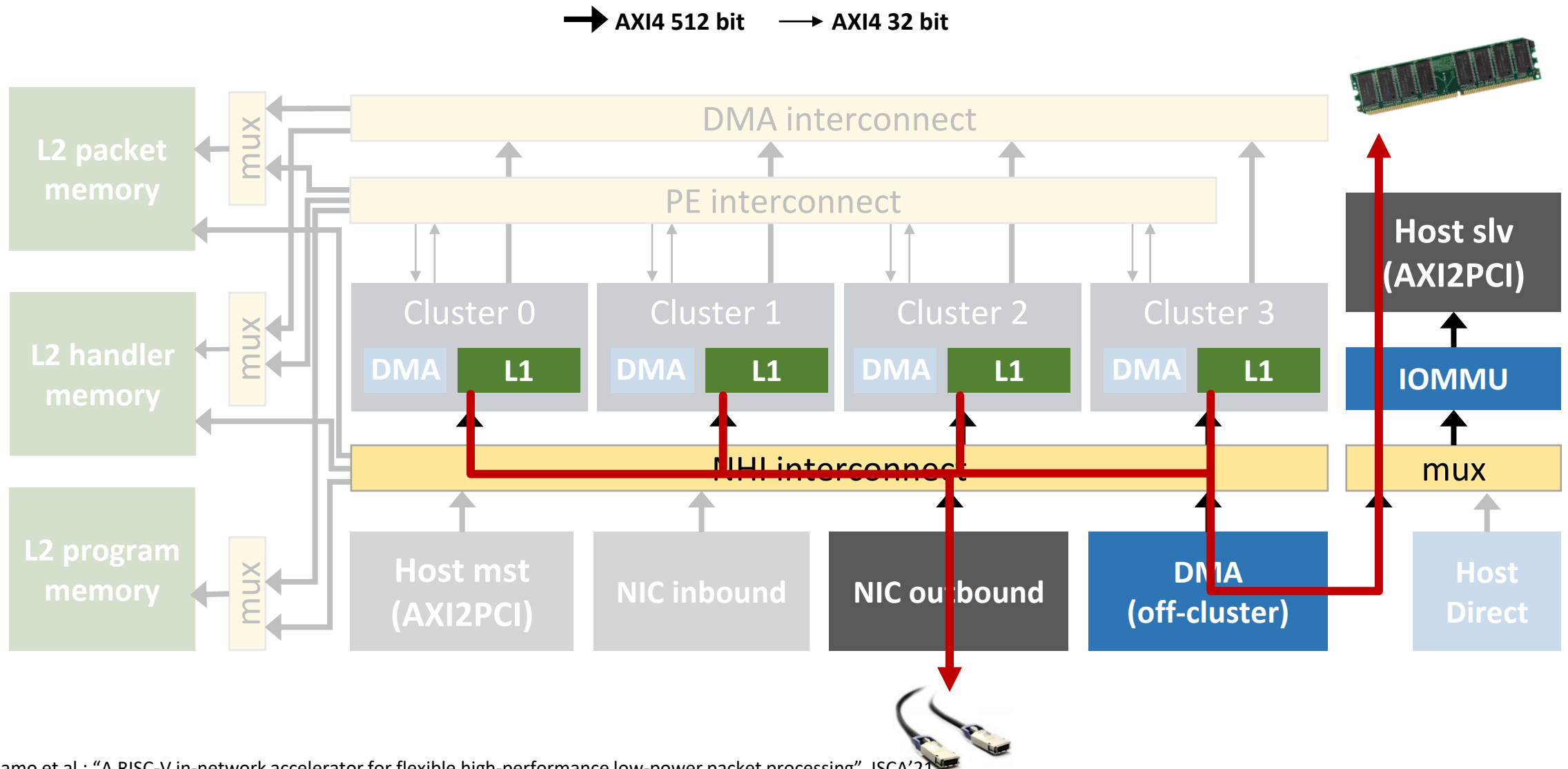
400G Data Path



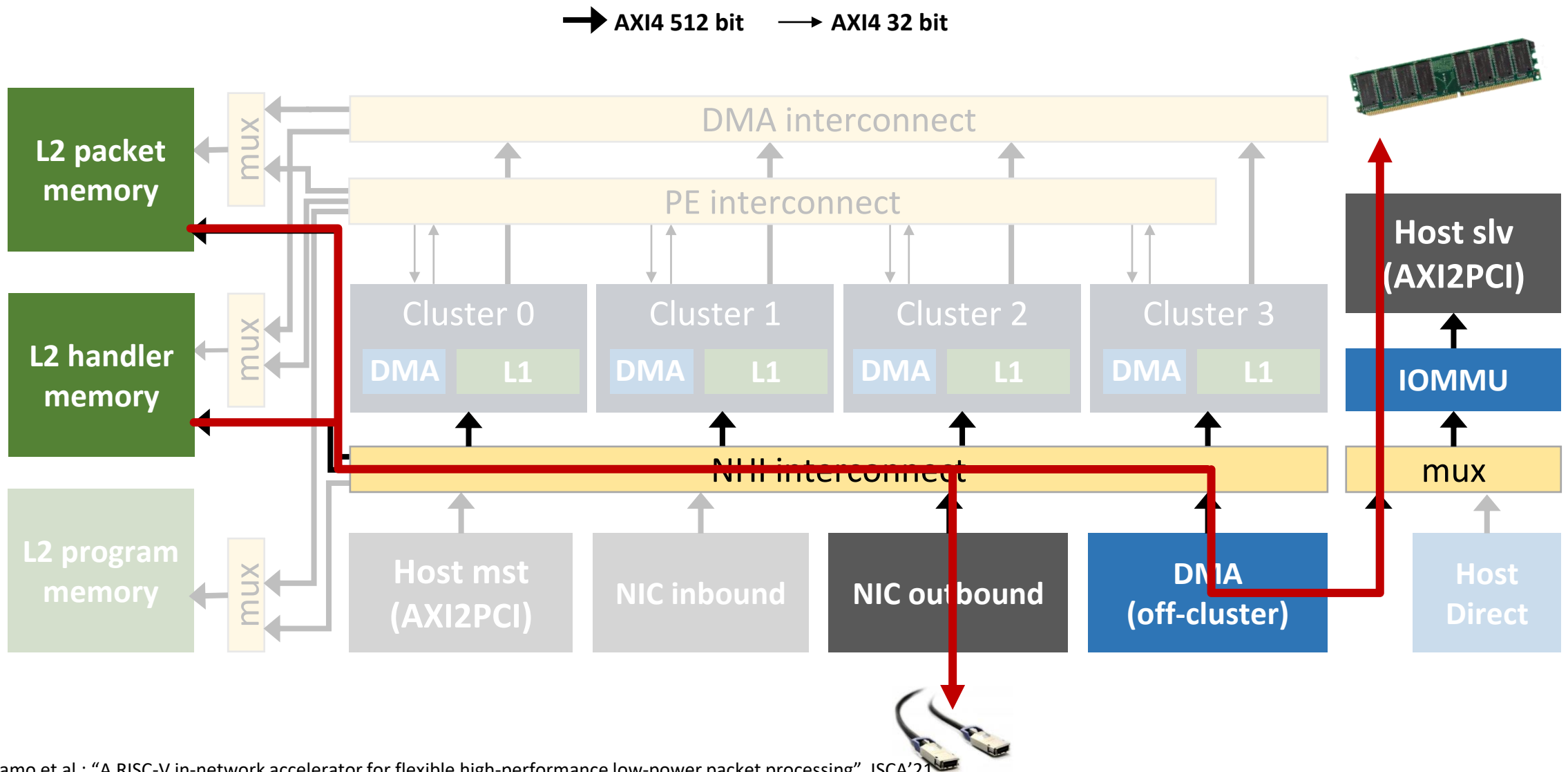
400G Data Path



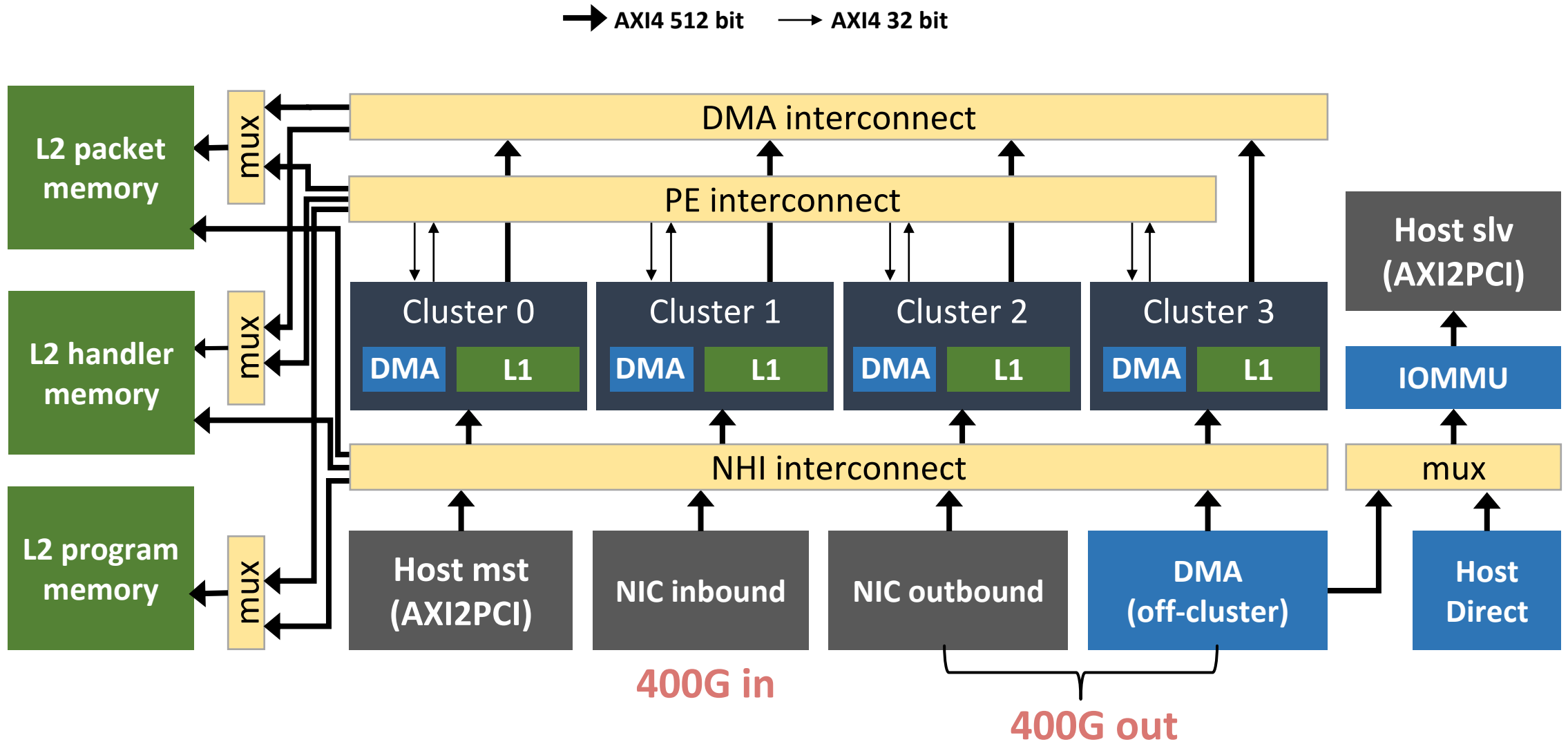
400G Data Path



400G Data Path

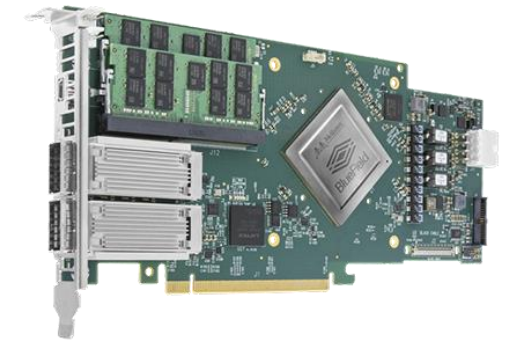
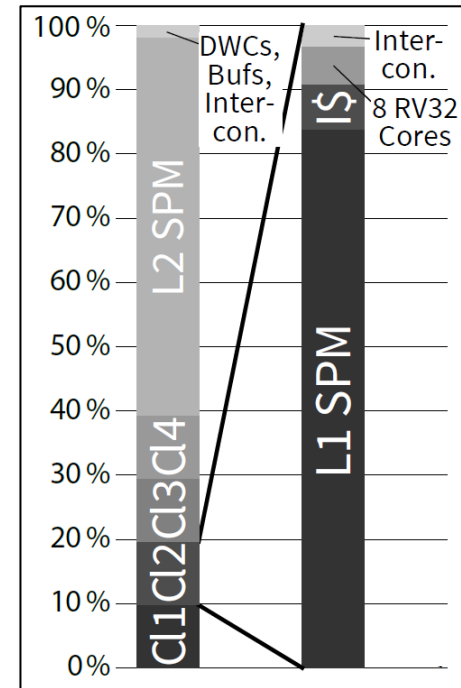
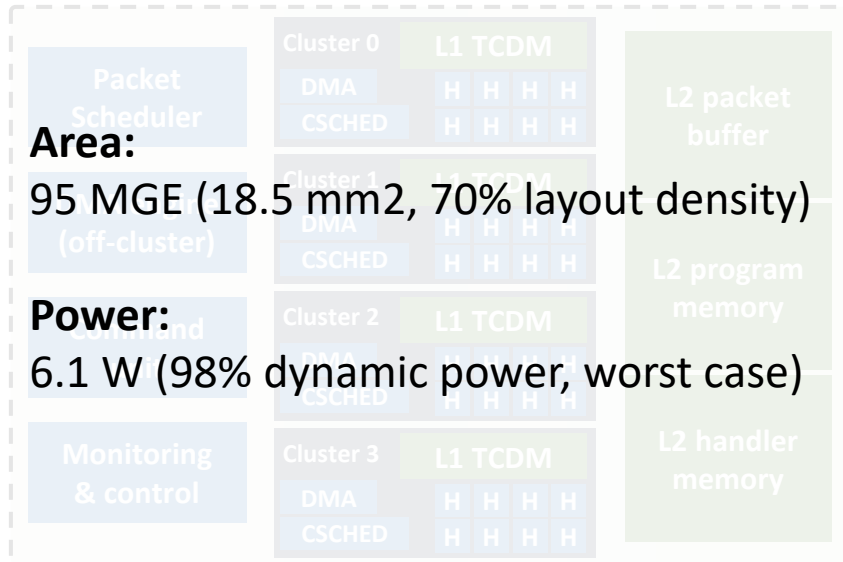


400G Data Path



Circuit Complexity and Power Estimations

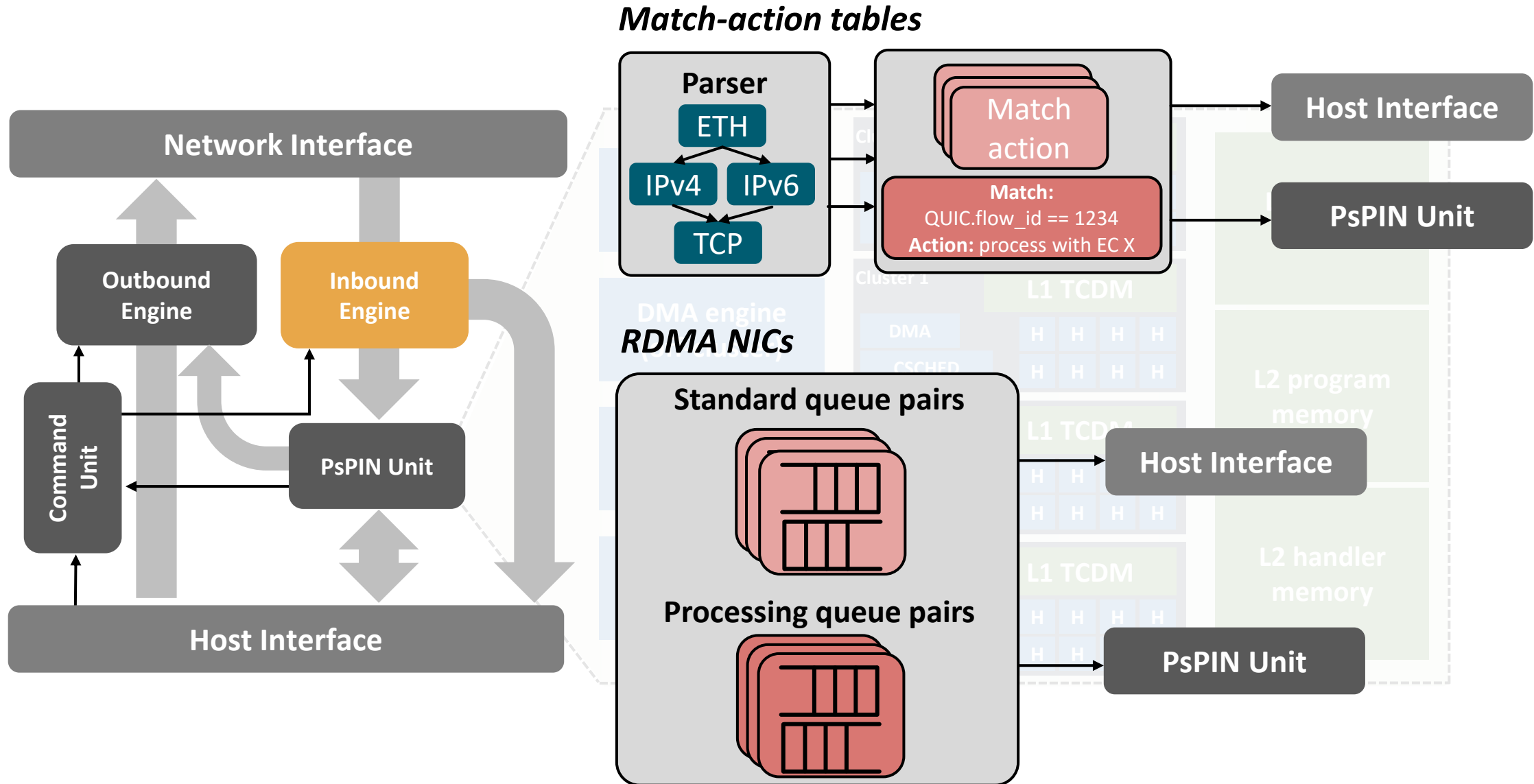
GlobalFoundries 22nm FDSOI @ 1GHz


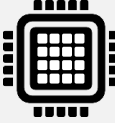

















Mellanox BlueField: 16 A72 64bit cores
Estimated area: 51 mm²

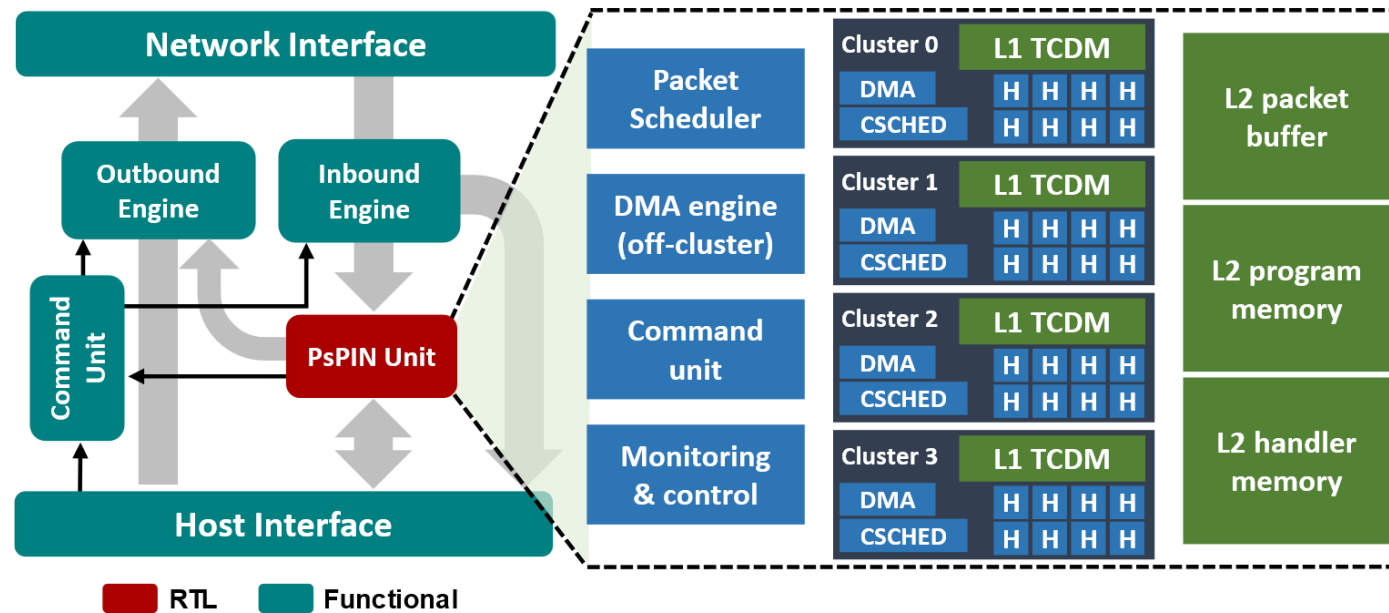
Component	Area (mm ²)			Power (W)		
	Unit	Total	Perc.	Unit	Total	Perc.
PsPIN	18.47	18.47	100.0%	6.08	6.08	100.0%
↳ L2 memories (×1)	9.48	9.48	51.3%	1.09	1.10	18.1%
↳ Interconnect (×1)	0.57	0.57	3.0%	0.71	0.71	11.7%
↳ Cluster (×4)	1.99	7.95	43.0%	0.94	3.77	62.0%
↳ L1 (×1)	1.65	1.65	82.9%	0.52	0.52	55.3%
↳ Core (×8)	0.01	0.08	4.0%	0.02	0.14	15.3%
↳ Instr. cache (×1)	0.08	0.08	4.0%	0.14	0.14	15.1%
↳ Interconnect (×1)	0.06	0.06	3.0%	0.11	0.11	11.3%

NIC integration



 Low latency, full throughput	 Highly parallel  Fast scheduling  Fast explicit memory access	   32 cores, higher core-count configurations are possible with more clusters Tens of nanoseconds to get handlers started Single-cycle L1 memory
 Support for wide range of use cases	 Stateful computation support  Handlers isolation	  Implicit in the sPIN programming model HW-configured (1 cycle) RISC-V PMP
 Easy to integrate	 Area and power efficiency  Configurability	  18.5 mm², 6.1 W Configurable number of clusters and cores/cluster

Experimental results



Handlers Characterization

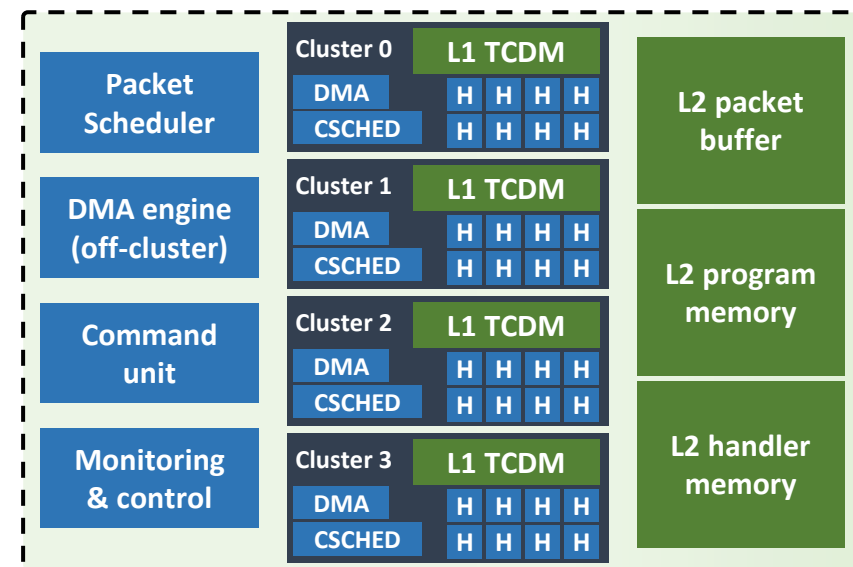
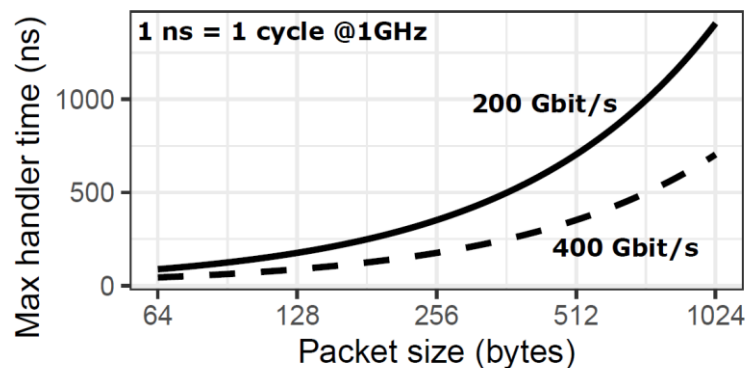
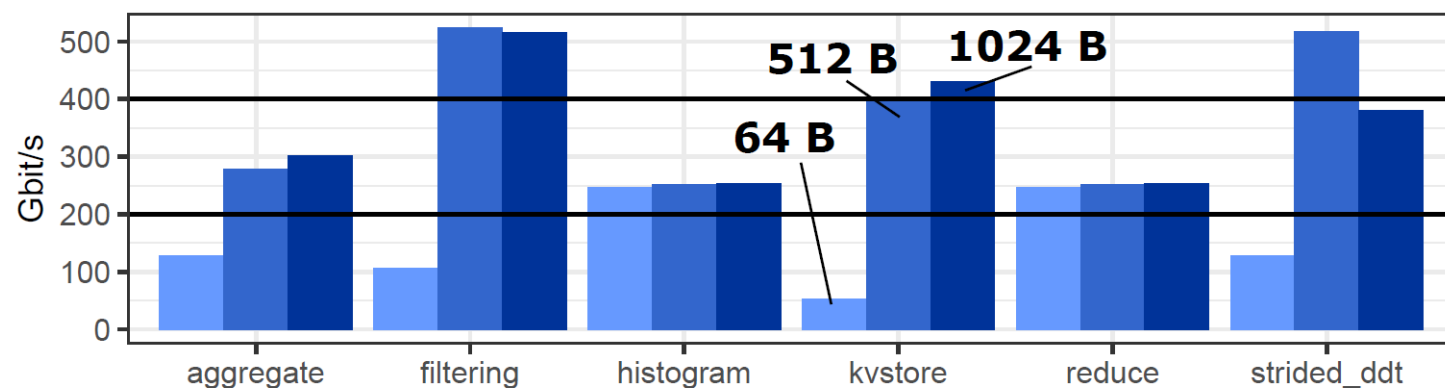
Packet steering
filtering, strided datatypes



Data movement
key-value store



Full packet processing
aggregate, histogram, reduce



How about other architectures?

ault @ CSCS



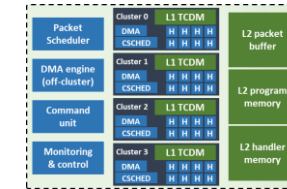
Xeon Gold @ 3 GHz
 (18-core, 4-way superscalar, OOO, 64-bit)

zynq



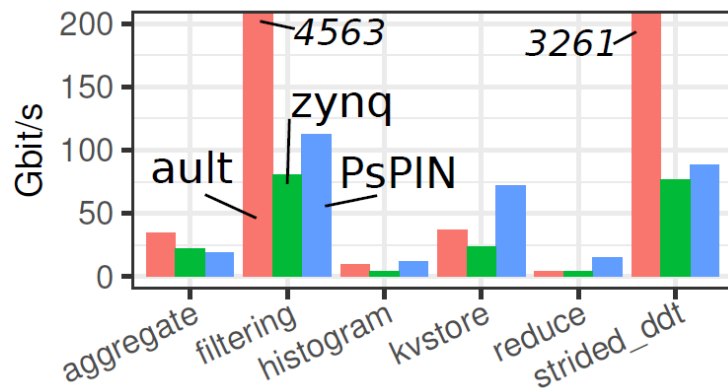
ARM Cortex-A53 @ 1.2 GHz
 (4 cores, 2-way superscalar, 64-bit)

PsPIN

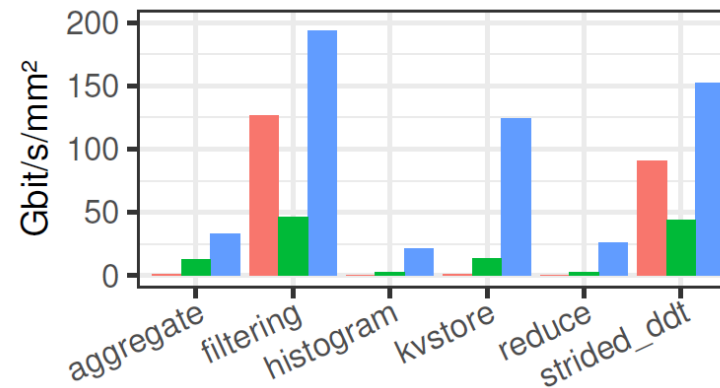


RISCV (RISC-V) @ 1 GHz
 (32 cores, single-issue, in-order, 32-bit)

Per-core throughput

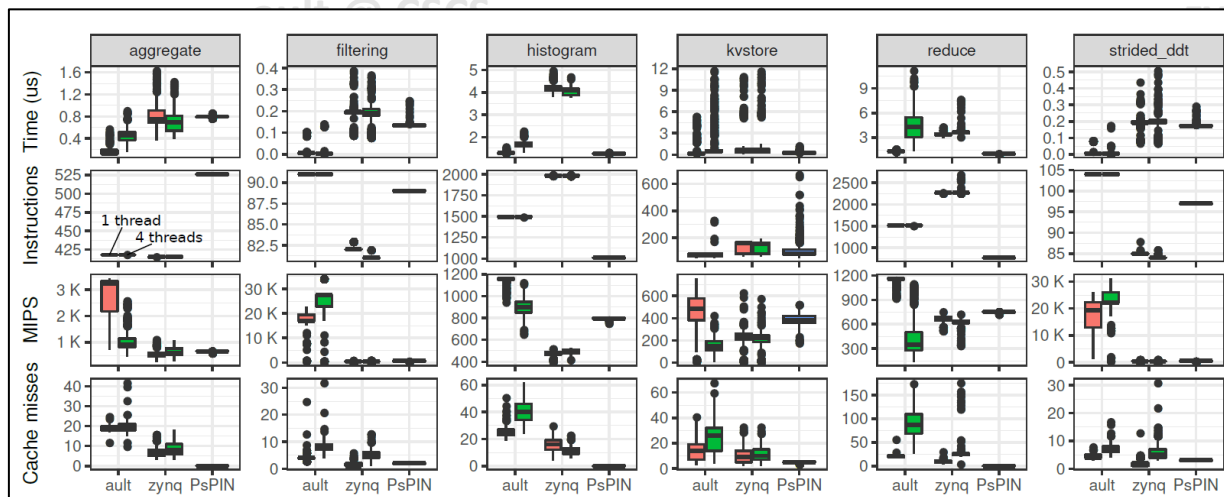


Per-core throughput/area



Arch.	Tech.	Die area	PEs	Memory	Area/PE	Area/PE (scaled)
ault	14 nm	485 mm ² [4]	18	43.3 MiB	17.978 mm ²	35.956 mm ²
zynq	16 nm	3.27 mm ² [3]	4	1.125 MiB	0.876 mm ²	1.752 mm ²
PsPIN	22 nm	18.5 mm ²	32	12 MiB	0.578 mm ²	0.578 mm ²

How about other architectures?



A RISC-V in-network accelerator for flexible high-performance low-power packet processing

Salvatore Di Girolamo*, Andreas Kurth†, Alexandru Calotoiu*, Thomas Benz†, Timo Schneider*, Jakob Beránek‡, Luca Benini†, Torsten Hoefler*

*Dept. of Computer Science, ETH Zürich, Switzerland

†Integrated System Laboratory, ETH Zürich, Switzerland

‡IT4Innovations, VŠB - Technical University of Ostrava

*{first.lastname}@inf.ethz.ch, †{first.lastname}@iis.ee.ethz.ch, ‡jakub.beranek@vsb.cz

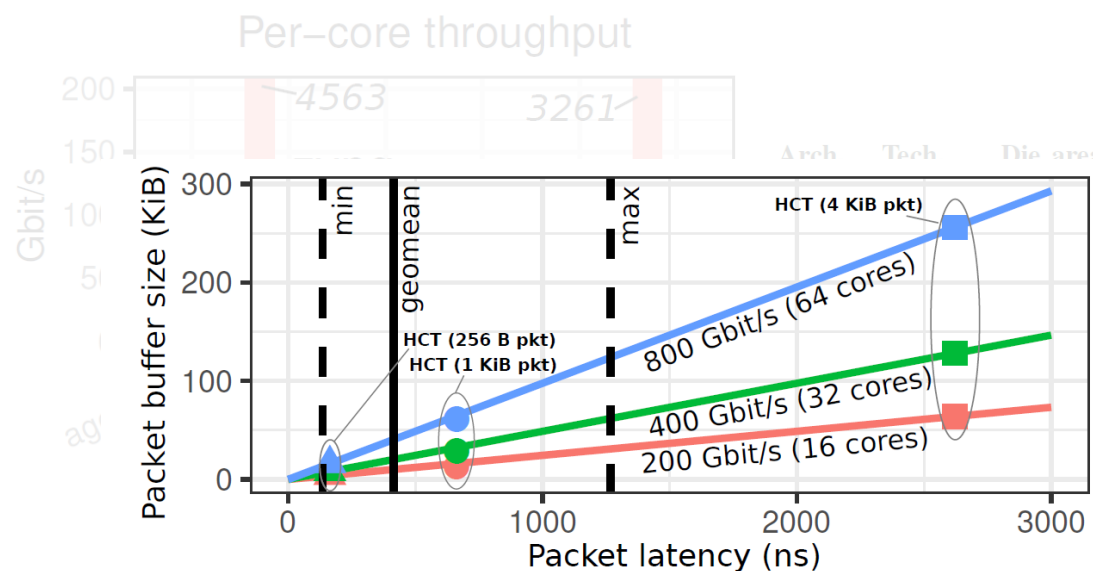
Abstract—The capacity of offloading data and control tasks to the network is becoming increasingly important, especially if we consider the faster growth of network speed when compared to CPU frequencies. In-network compute alleviates the host CPU load by running tasks directly in the network, enabling additional computation/communication overlap and potentially improving overall application performance. However, sustaining bandwidths provided by next-generation networks, e.g., 400 Gbit/s, can become a challenge. sPIN is a programming model for in-NIC compute, where users specify handler functions that are executed on the NIC, for each incoming packet belonging to a given message or flow. It enables a CUDA-like acceleration, where the NIC is equipped with lightweight processing elements that process network packets in parallel. We investigate the architectural specialties that a sPIN NIC should provide to enable high-performance, low-power, and flexible packet processing. We introduce PsPIN, a first open-source sPIN implementation, based on a multi-cluster RISC-V architecture and designed according to the identified architectural specialties. We investigate the performance of PsPIN with cycle-accurate simulations, showing that it can process packets at 400 Gbit/s for several use cases, introducing minimal latencies (26 ns for 64 B packets) and occupying a total area of 18.5 mm² (22 nm FDSOI).

Index Terms—in-network compute, packet processing, special-

data into user-space memory. Even though this greatly reduces packet processing overheads on the CPU, the incoming data must still be processed. A flurry of specialized technologies exists to move additional parts of this processing into network cards, e.g., FPGAs virtualization support [22], P4 simple rewriting rules [13], or triggered operations [9].

Streaming processing in the network (sPIN) [28] defines a unified programming model and architecture for network acceleration beyond simple RDMA. It provides a user-level interface, similar to CUDA for compute acceleration, considering the specialties and constraints of low-latency line-rate packet processing. It defines a flexible and programmable network instruction set architecture (NISA) that not only lowers the barrier of entry but also supports a large set of use-cases [28]. For example, Di Girolamo et al. demonstrate up to 10x speedups for serialization and deserialization (marshalling) of non-consecutive data [20].

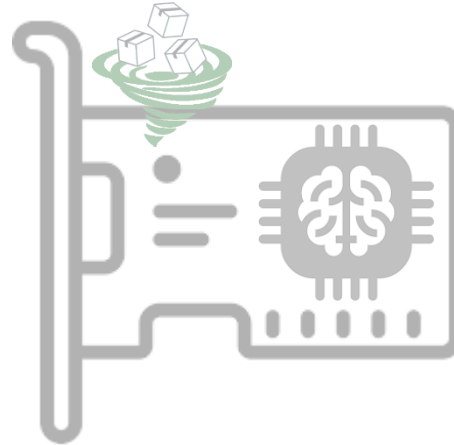
While the NISA defined by sPIN can be implemented on existing SmartNICs [1], their microarchitecture (often standard ARM SoCs) is not optimized for packet-processing tasks. In



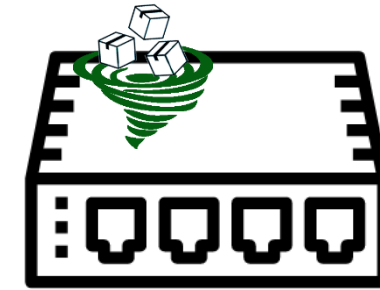
Talk roadmap



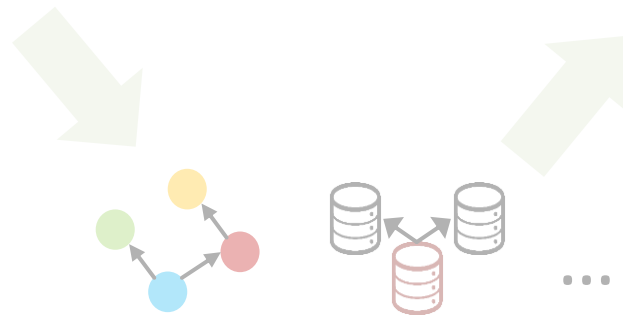
Motivation and Overview



Hardware Implementation



In-network reductions



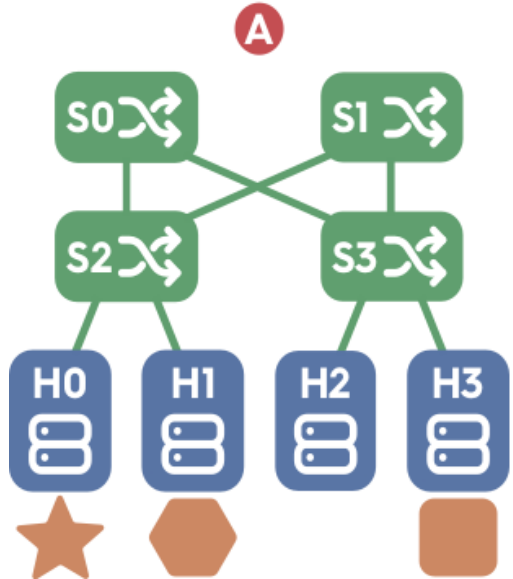
Network Group
Communication

Distributed Data
Management

Use cases

sPIN switch motivation

We can reduce the required network bandwidth for allreduce by 2x if we **offload the operation into the network**



State of the art

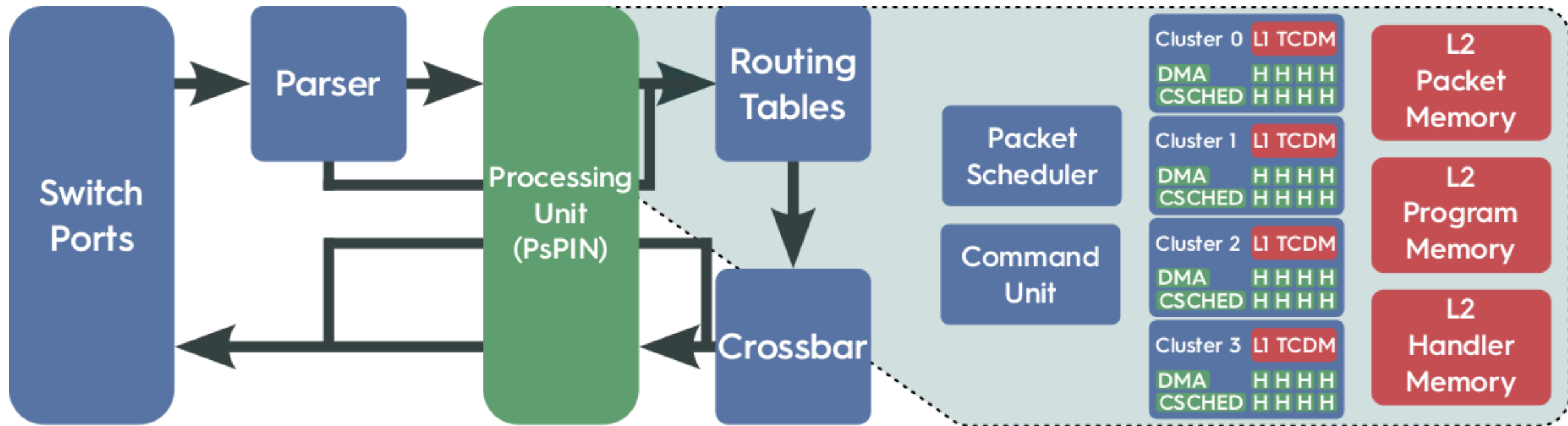
Mellanox SHARP

SwitchML

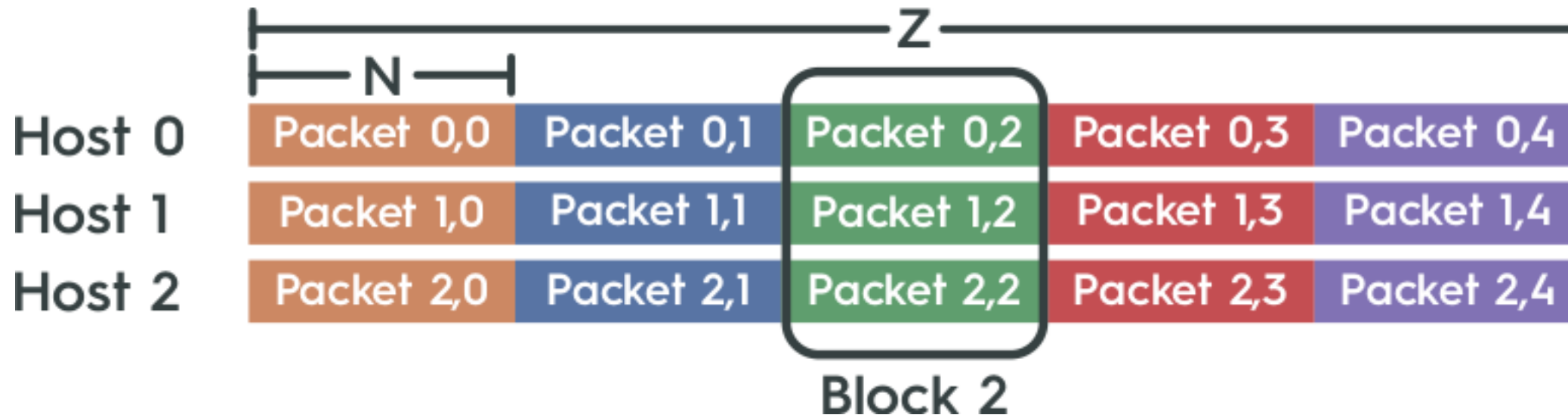
	FIXED-FUNCTION SWITCHES							FPGAs		PROGR. SWITCHES			
	[9]	[16]	[17]	[18]	[19]	[21]	[10]	[22]	[23]	[24]	[11]	[25]	FLARE
F1	👎	👎	👎	👎	👎	👎	👎	👎	👎	👍	👍	👍	👍
F2	👎	👎	👎	👎	👎	👎	👎	👎	👎	👎	👎	👍	👍
F3	?	👍	?	?	👍	?	👎	👍	👍	👎	👎	👎	👍

- F1 – custom operators and types**
- F2 – unstructured and sparse data**
- F3 – determinism/reproducibility**

sPIN switch architecture – same core sPIN system design!

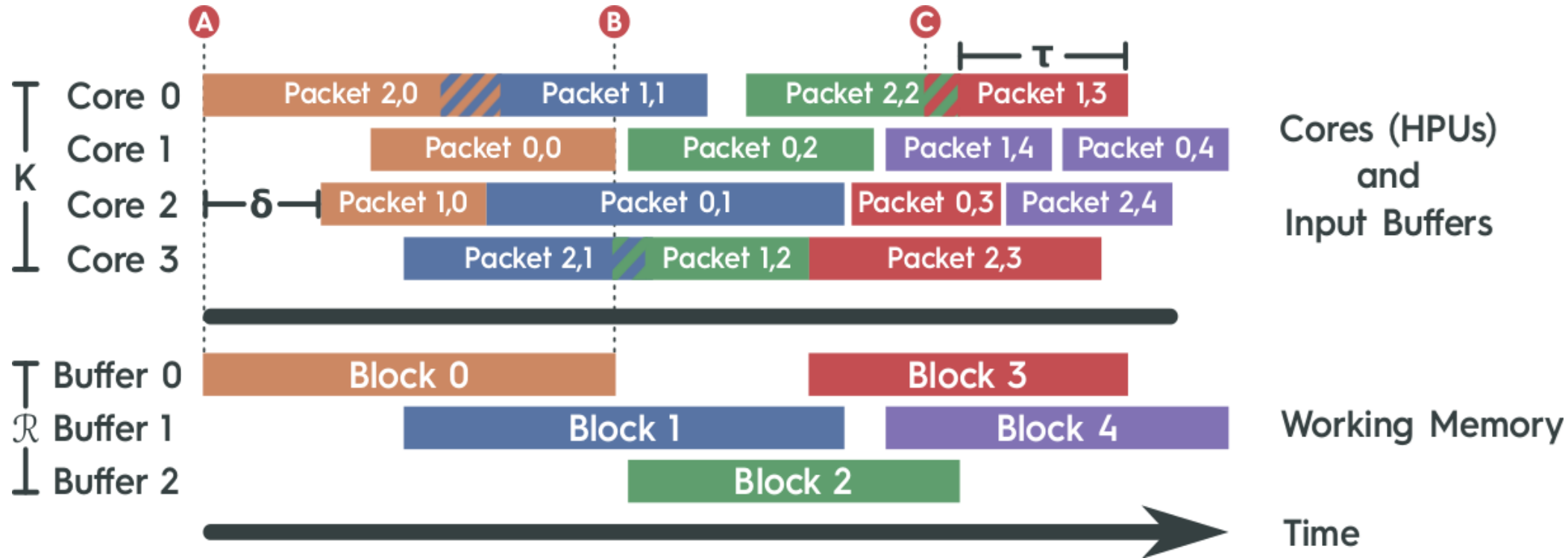


Allreduce basics



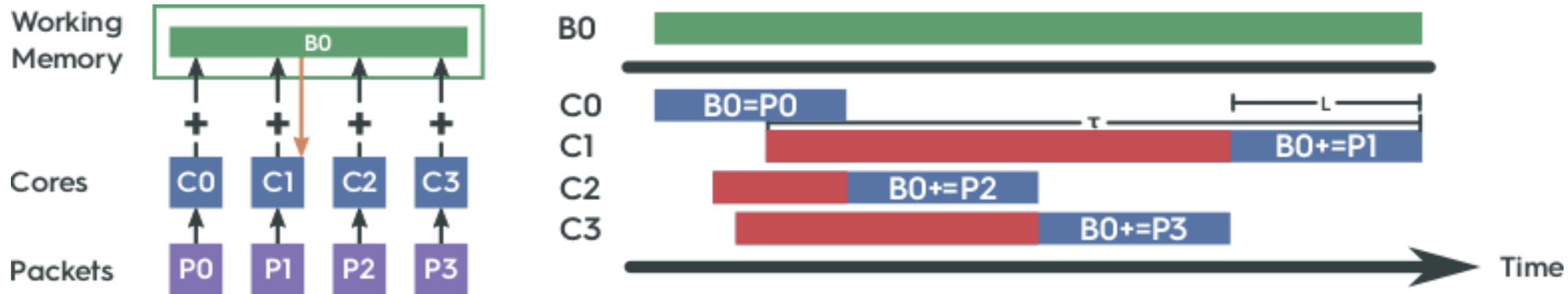
Packets in a **block** need to be **aggregated** together

An example timeline in a switch



Buffering and staggered sending

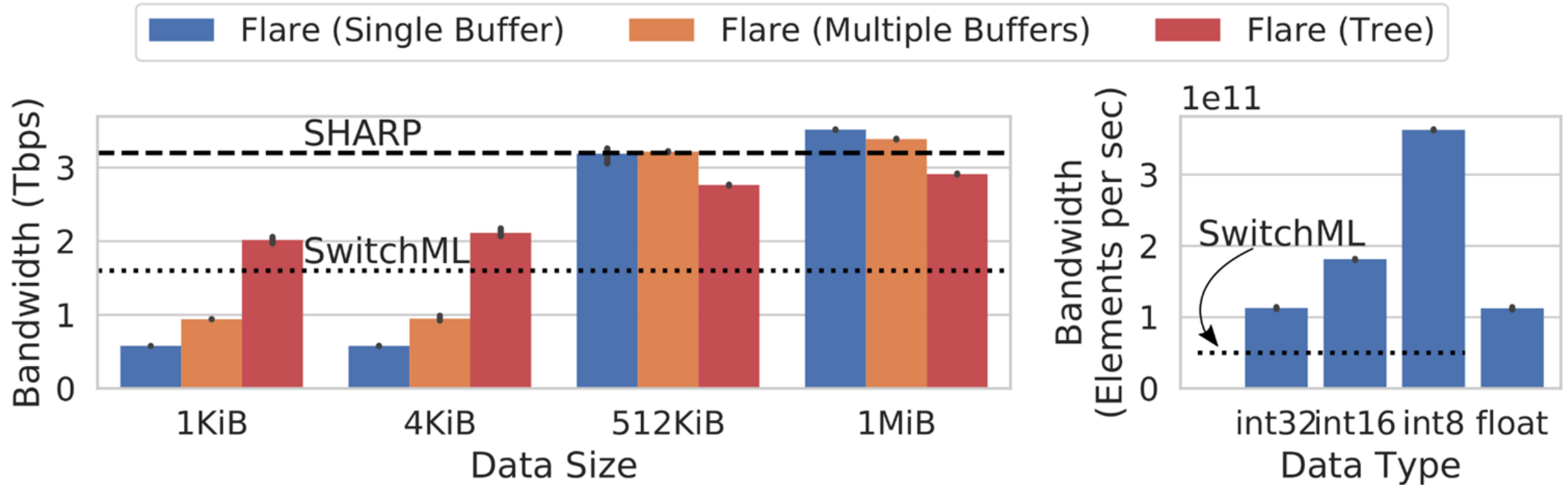
Single buffer requires locking (or atomics)



Staggered block sending mitigates locking – requires some buffer memory

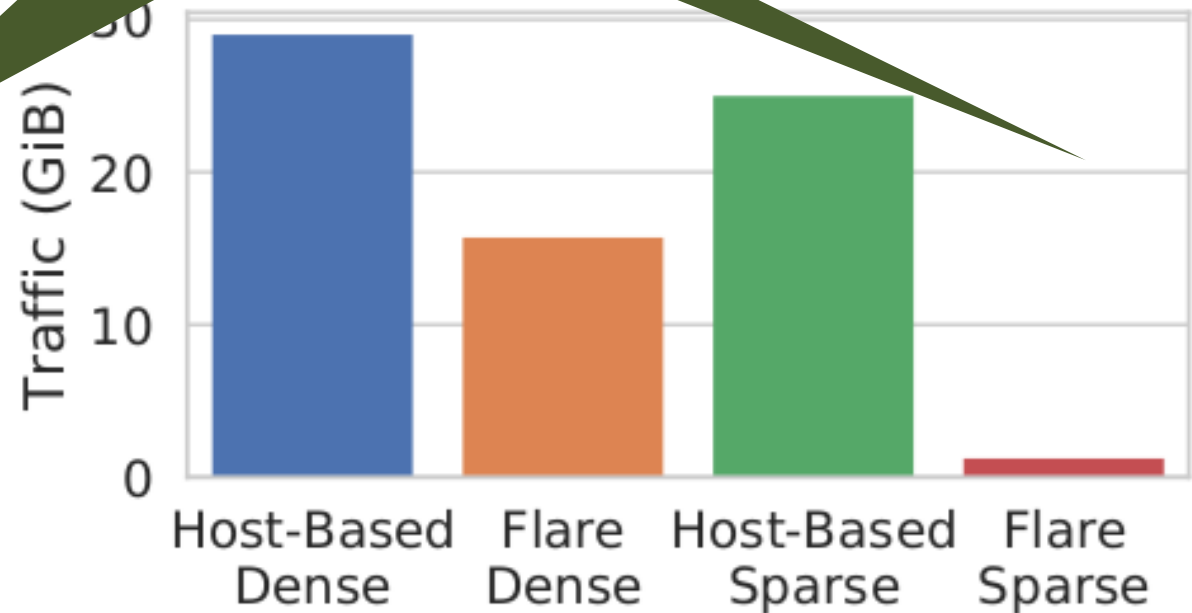
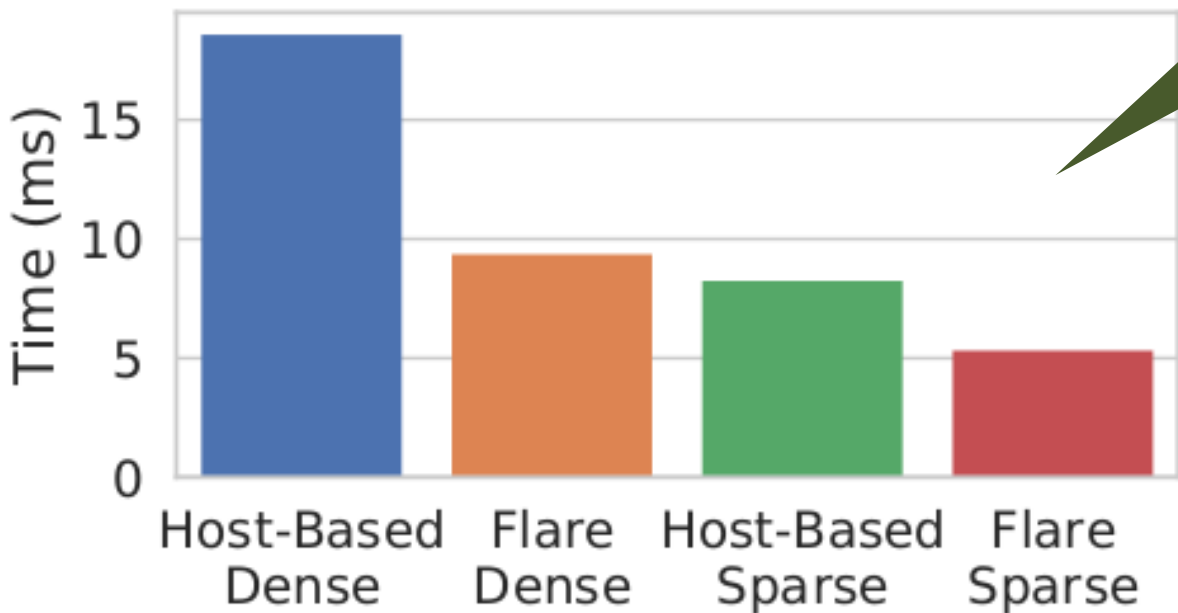


Results – cycle accurate simulations



ResNet-50 training example

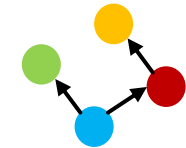
Bound by computation / microarchitecture!
Consider iSSR (DATE'21).



Communication time of a ResNet50 iteration with sparsified gradients (99.8% sparse)



Motivation and Overview



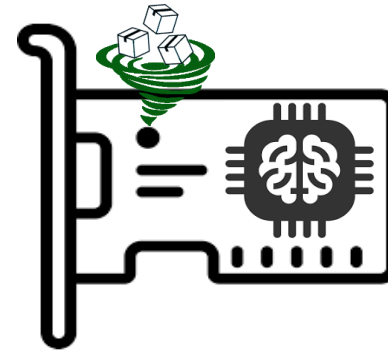
Network Group Communication



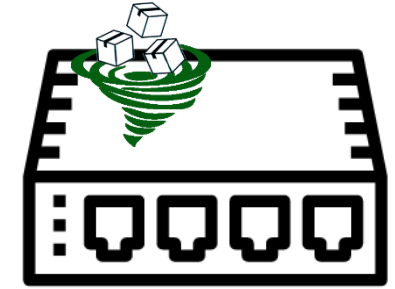
Distributed Data Management

...

Use cases



Hardware Implementation

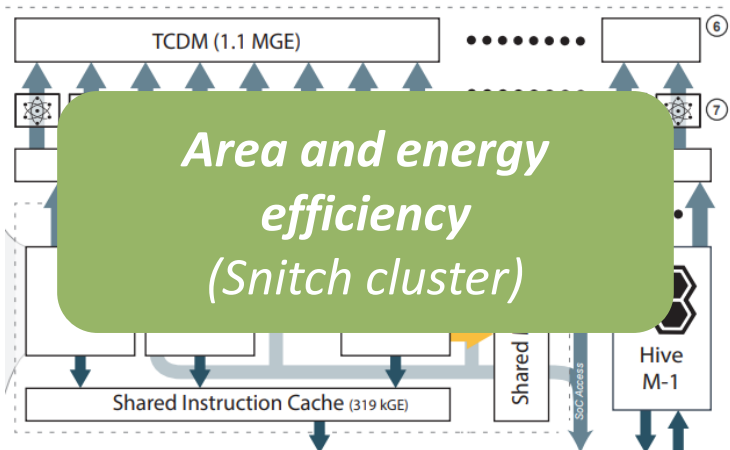


In-network reductions



<https://github.com/spcl/pspin>

RTL, runtime, examples



Area and energy efficiency (Snitch cluster)

Large scale simulations (SST + Verilator)

Programming interface (performance modeling and handler offloading)

`bar()`

SPCL is hiring PhD students and highly-qualified postdocs to reach new heights!

