2022 OFA Virtual Workshop

sPIN: High-performance streaming Processing in the Network

S. Di Girolamo, D. De Sensi, T. Hoefler, and the sPIN team @SPCL

NETFLIX

zoom

Workloads

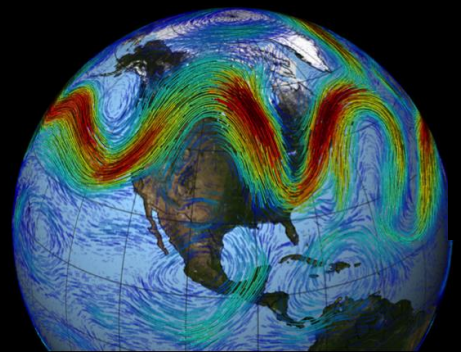
Systems

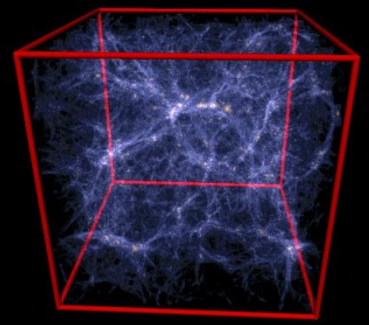
NETFLIX

zoom

Workloads

Systems



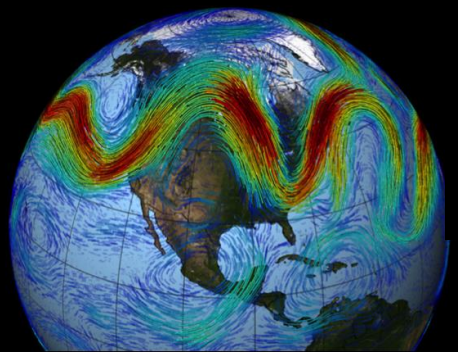


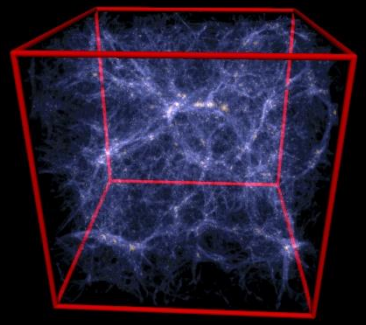
NETFLIX

zoom

Workloads

Systems

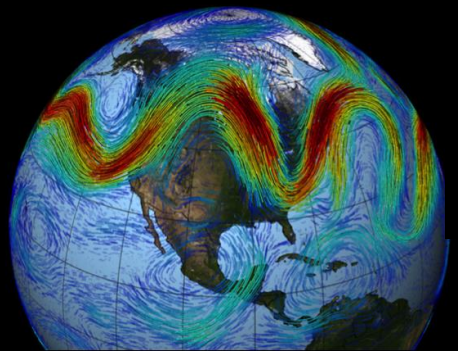


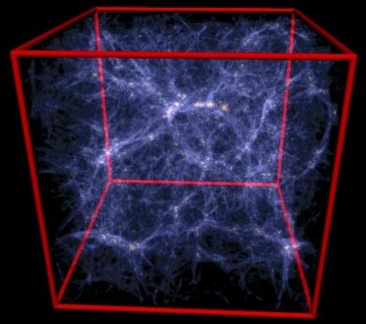


NETFLIX

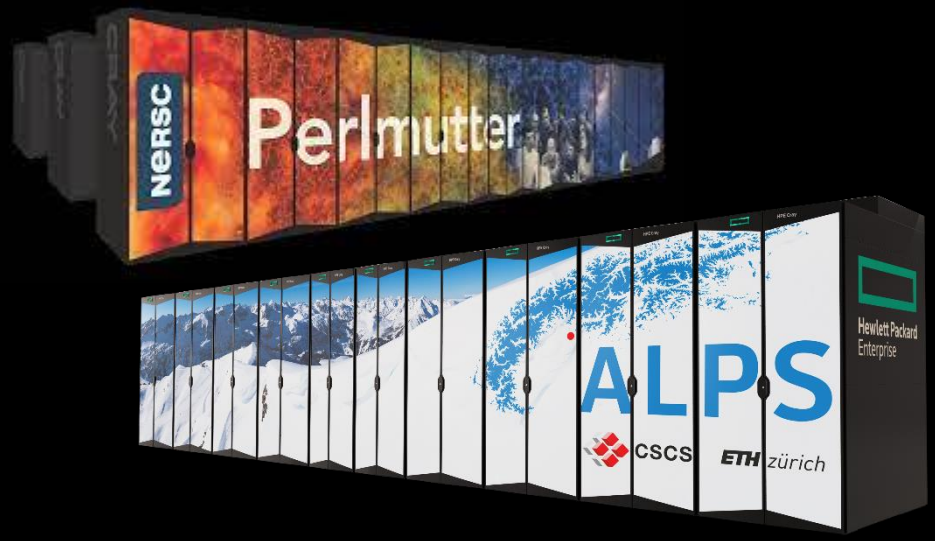
zoom

Workloads



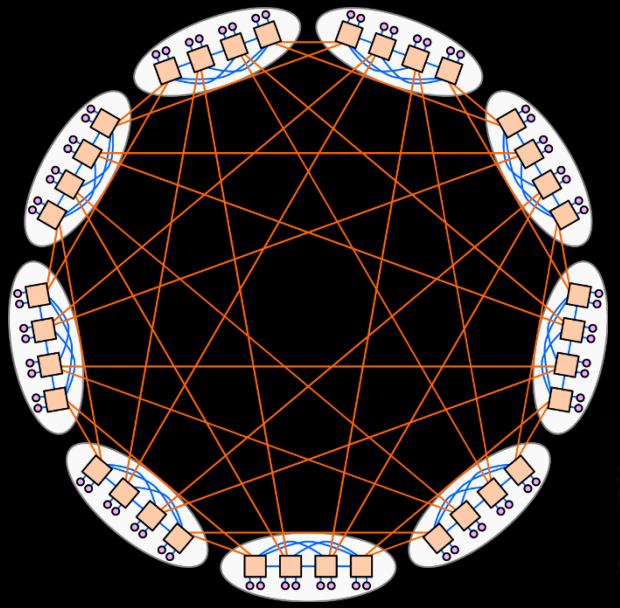


Systems

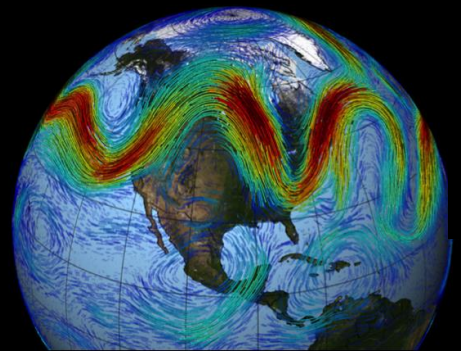


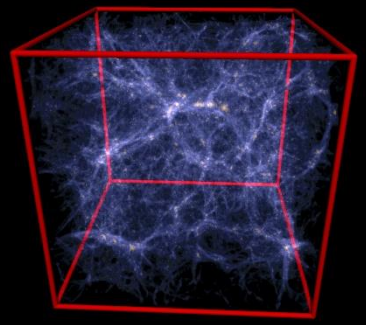
NETFLIX

zoom

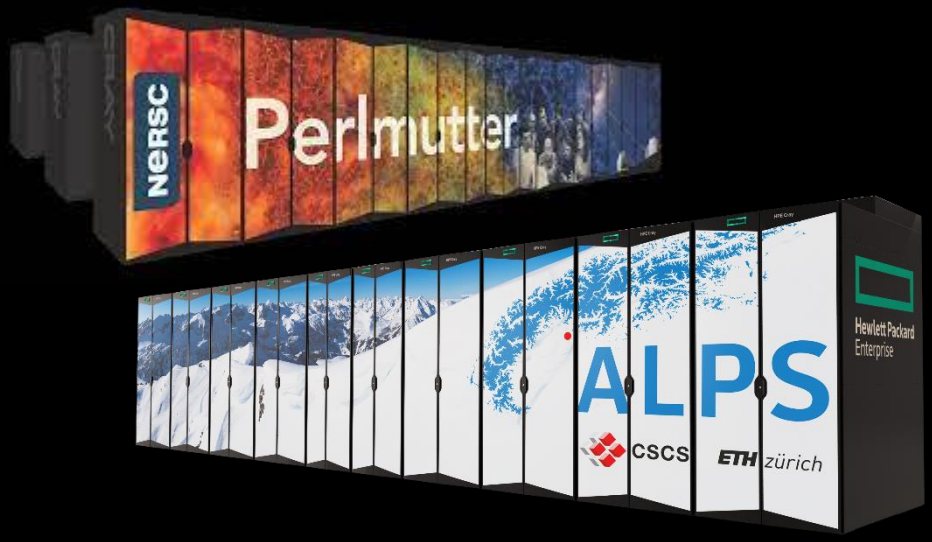


Workloads





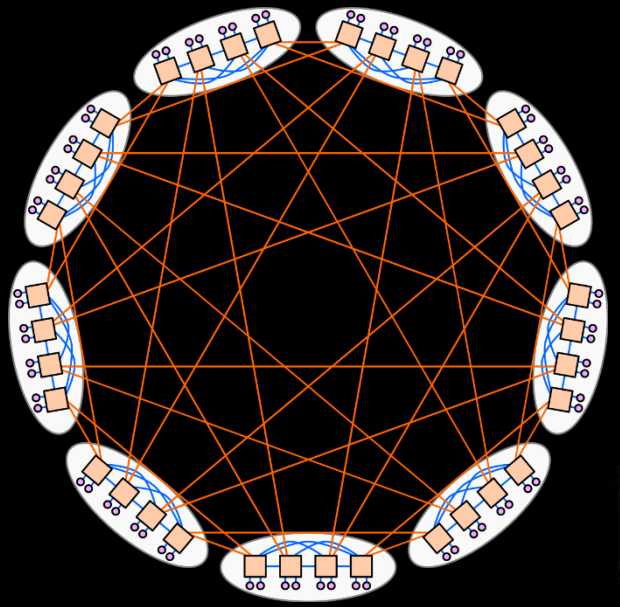
Systems



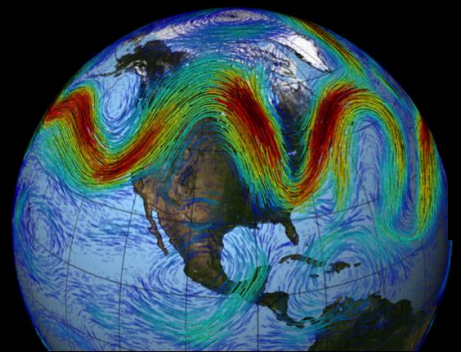
NETFLIX

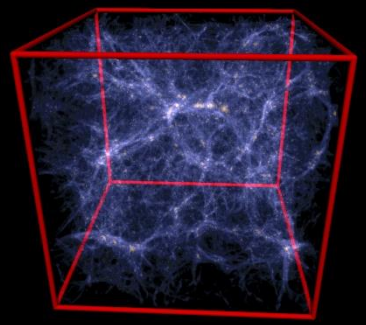
zoom

Low latency,
high throughput

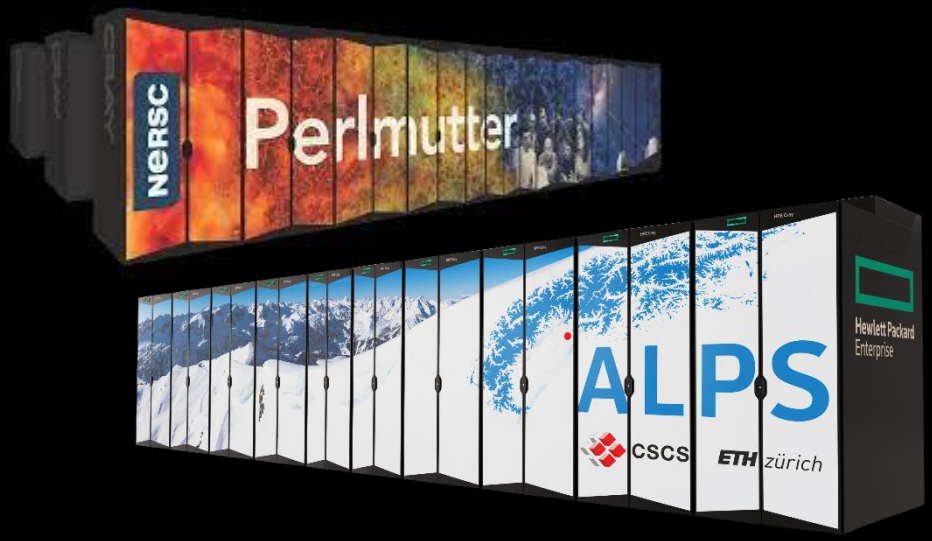


Workloads





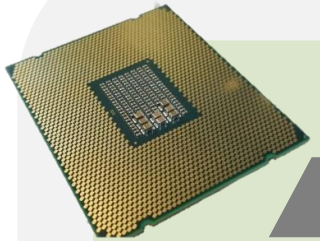
Systems



Data Processing in modern RDMA networks

Remote Nodes (via network)

Local Node



Core i7 Haswell

Regs	Regs
4 cycles ↕ ~1.3ns	4 cycles ↕ ~1.3ns
L1	L1
11 cycles ↕ ~ 3.6ns	11 cycles ↕ ~ 3.6ns
L2	L2
34 cycles ↕ ~11.3ns	34 cycles ↕ ~11.3ns

L3

125 cycles ↕ ~41.6ns

RDMA NIC

RDMA Processing

DMA Unit

PCIe bus

~ 250ns

Main Memory

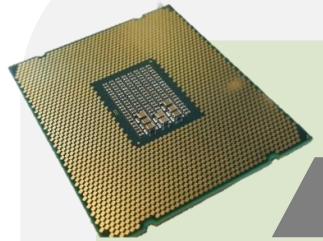
Input buffer

arriving packets

Data Processing in modern RDMA networks

Remote Nodes (via network)

Local Node



Core i7 Haswell

Regs	Regs
4 cycles ↕ ~1.3ns	4 cycles ↕ ~1.3ns
L1	L1
11 cycles ↕ ~ 3.6ns	11 cycles ↕ ~ 3.6ns
L2	L2
34 cycles ↕ ~11.3ns	34 cycles ↕ ~11.3ns

L3

125 cycles ↕ ~41.6ns

RDMA NIC

RDMA Processing

DMA Unit

PCIe bus

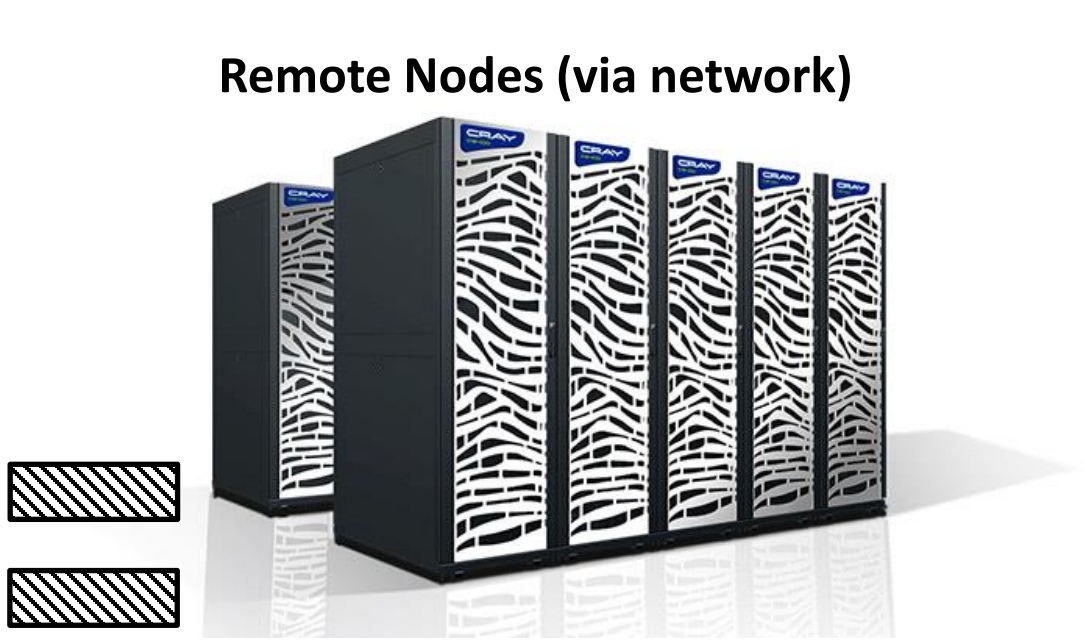
~ 250ns

Main Memory

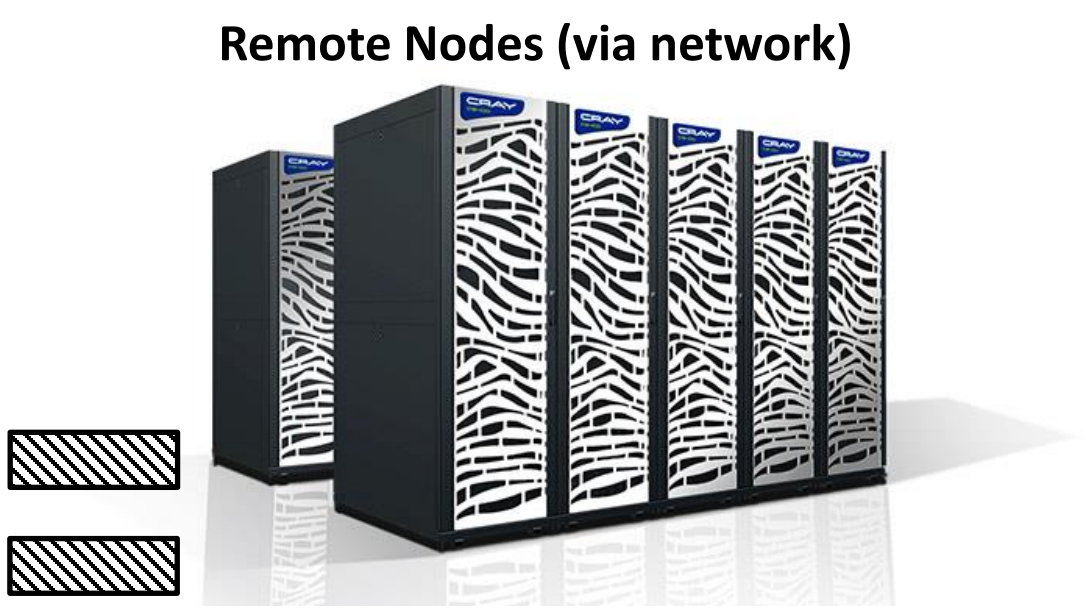
Input buffer

arriving packets

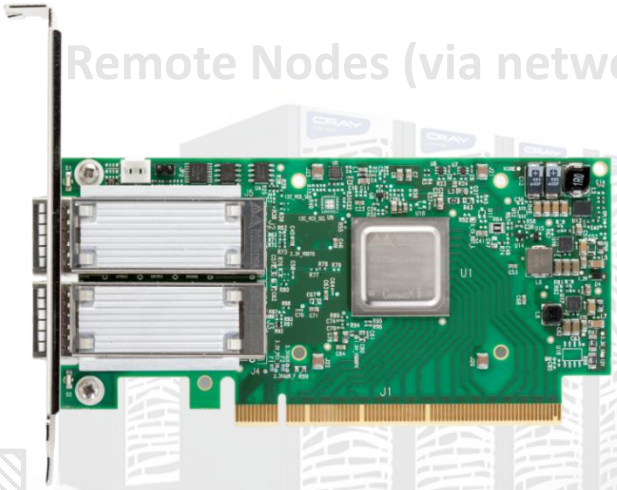
Data Processing in modern RDMA networks



Data Processing in modern RDMA networks



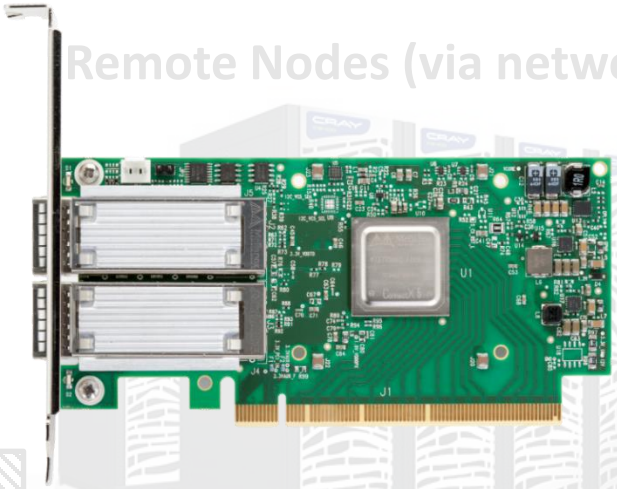
Data Processing in modern RDMA networks



Mellanox Connect-X5: 1 packet/5ns

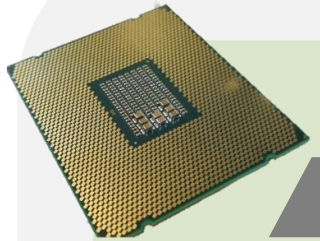
Mellanox Connect-X7 (400G): 1 packet/1.2ns

Data Processing in modern RDMA networks



Mellanox Connect-X5: 1 packet/5ns

Mellanox Connect-X7 (400G): 1 packet/1.2ns



Local Node

Core i7 Haswell

Regs	4 cycles	~1.3ns
L1	11 cycles	~ 3.6ns
L2	34 cycles	~11.3ns
L3		

125 cycles ~41.6ns

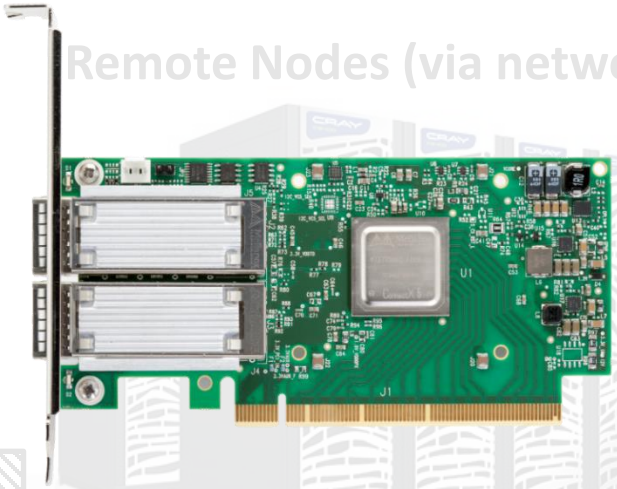
PCIe bus

~ 250ns

Main Memory

Input buffer

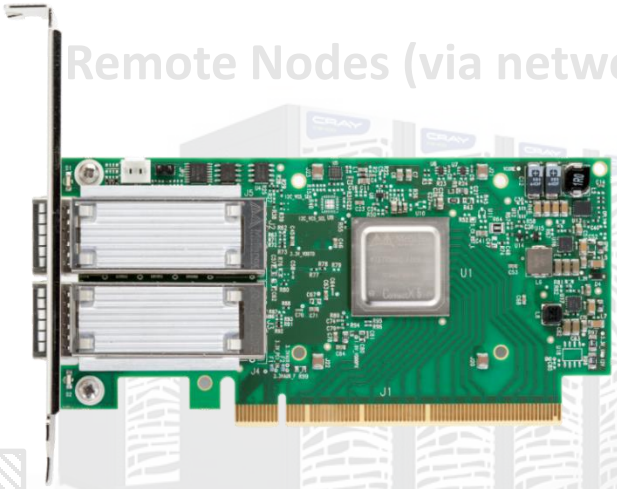
Data Processing in modern RDMA networks



Mellanox Connect-X5: 1 packet/5ns

Mellanox Connect-X7 (400G): 1 packet/1.2ns

Data Processing in modern RDMA networks



Mellanox Connect-X5: 1 packet/5ns

Mellanox Connect-X7 (400G): 1 packet/1.2ns

RDMA NIC

?

arriving packets

PCIe bus

~ 250ns

Remote Nodes (via network)

Local Node

Core i7 Haswell

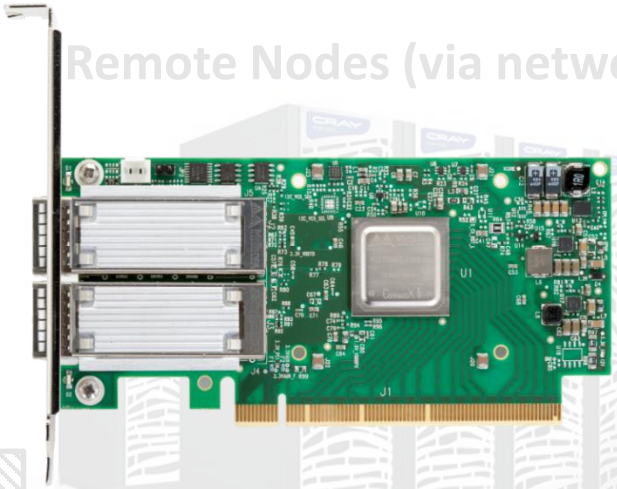
Regs
4 cycles ↕ ~1.3ns
L1
11 cycles ↕ ~ 3.6ns
L2
34 cycles ↕ ~11.3ns
L3

125 cycles ↕ ~41.6ns

Main Memory

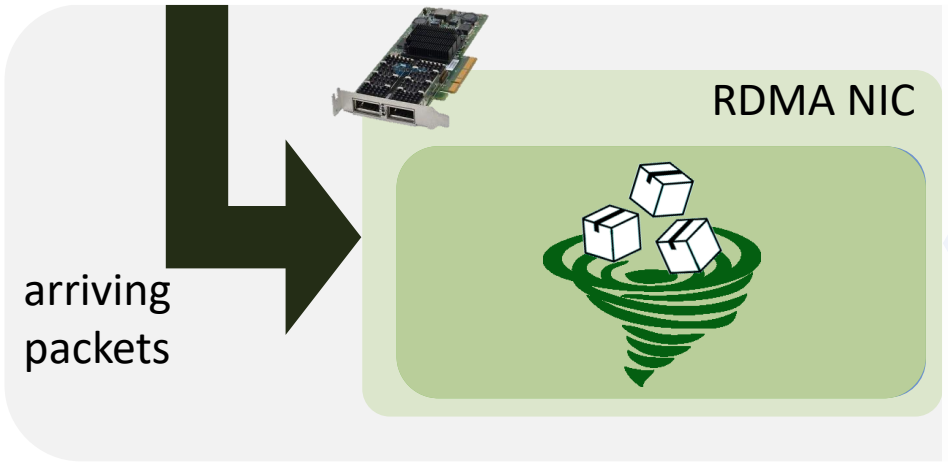
Input buffer

Data Processing in modern RDMA networks



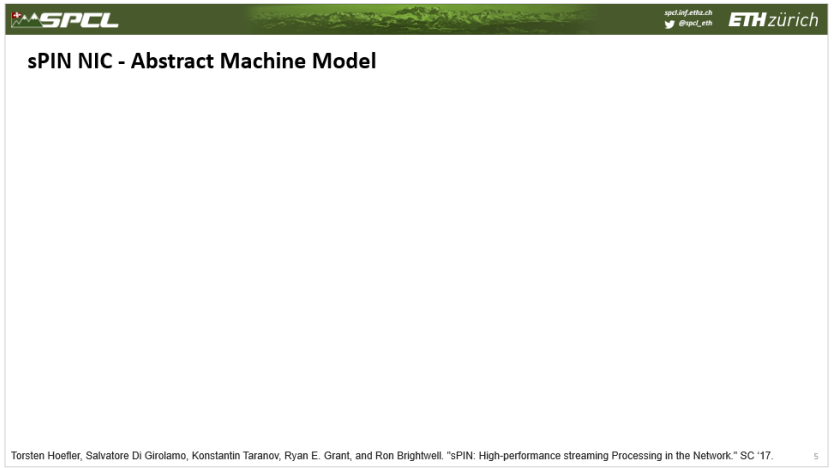
Mellanox Connect-X5: 1 packet/5ns

Mellanox Connect-X7 (400G): 1 packet/1.2ns

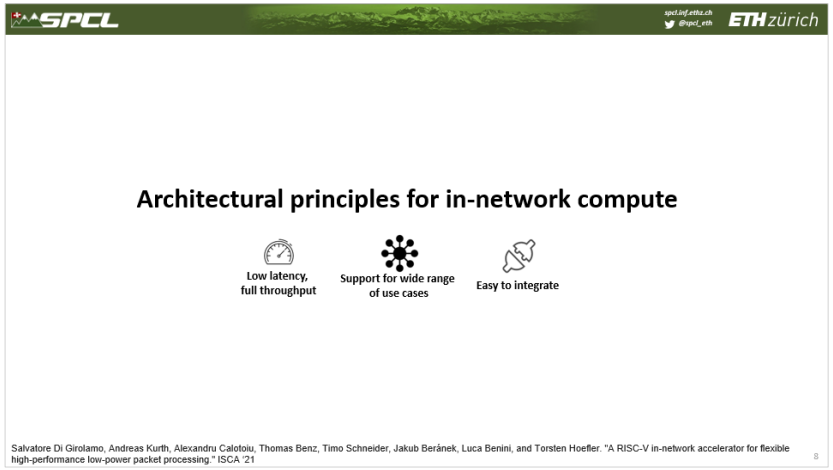


sPIN: High-performance streaming Processing in the Network

The programming model



The hardware accelerator



Use cases

sPIN NIC - Abstract Machine Model

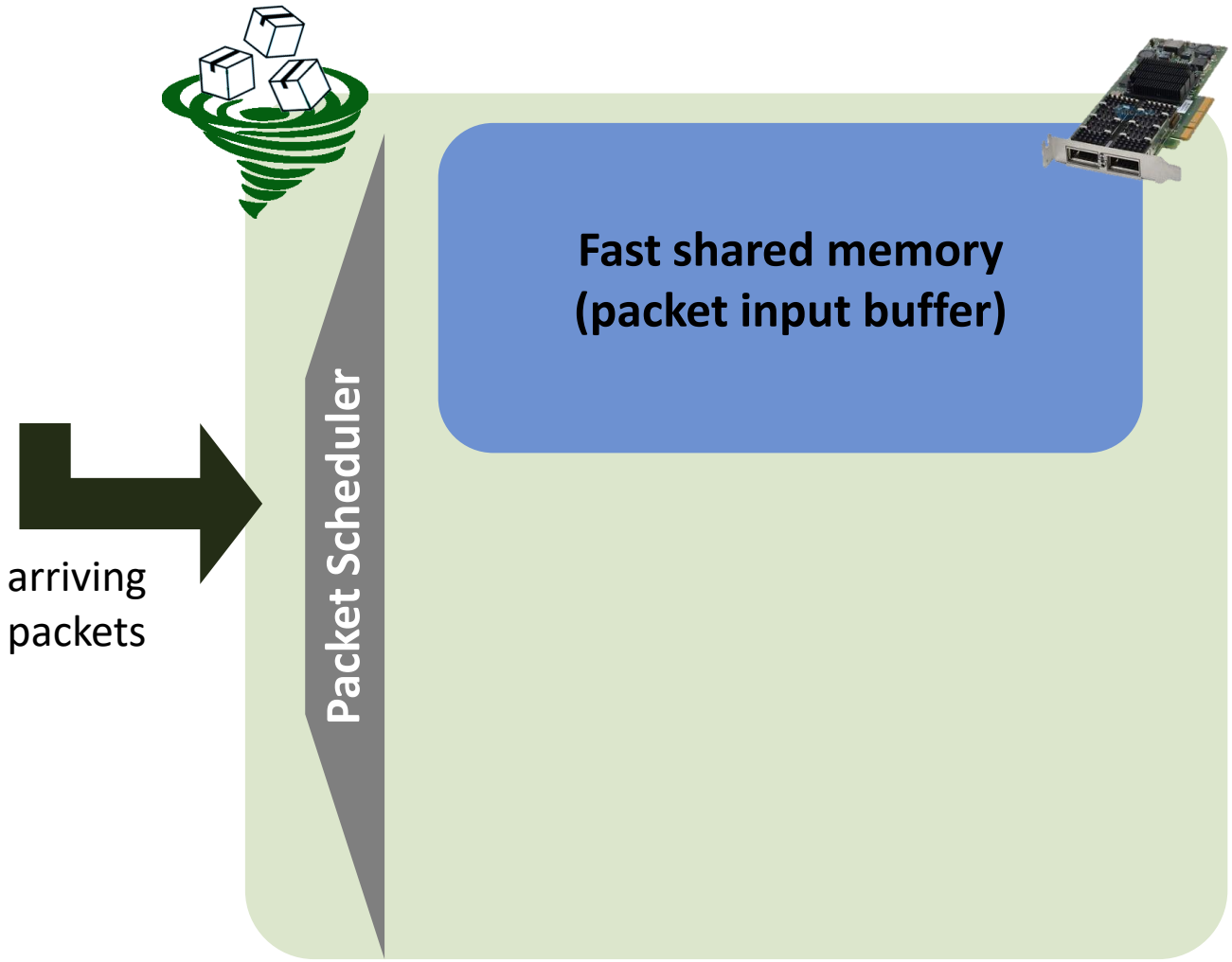
sPIN NIC - Abstract Machine Model

Torsten Hoefler, Salvatore Di Girolamo, Konstantin Taranov, Ryan E. Grant, and Ron Brightwell. "sPIN: High-performance streaming Processing in the Network." SC '17.

sPIN NIC - Abstract Machine Model

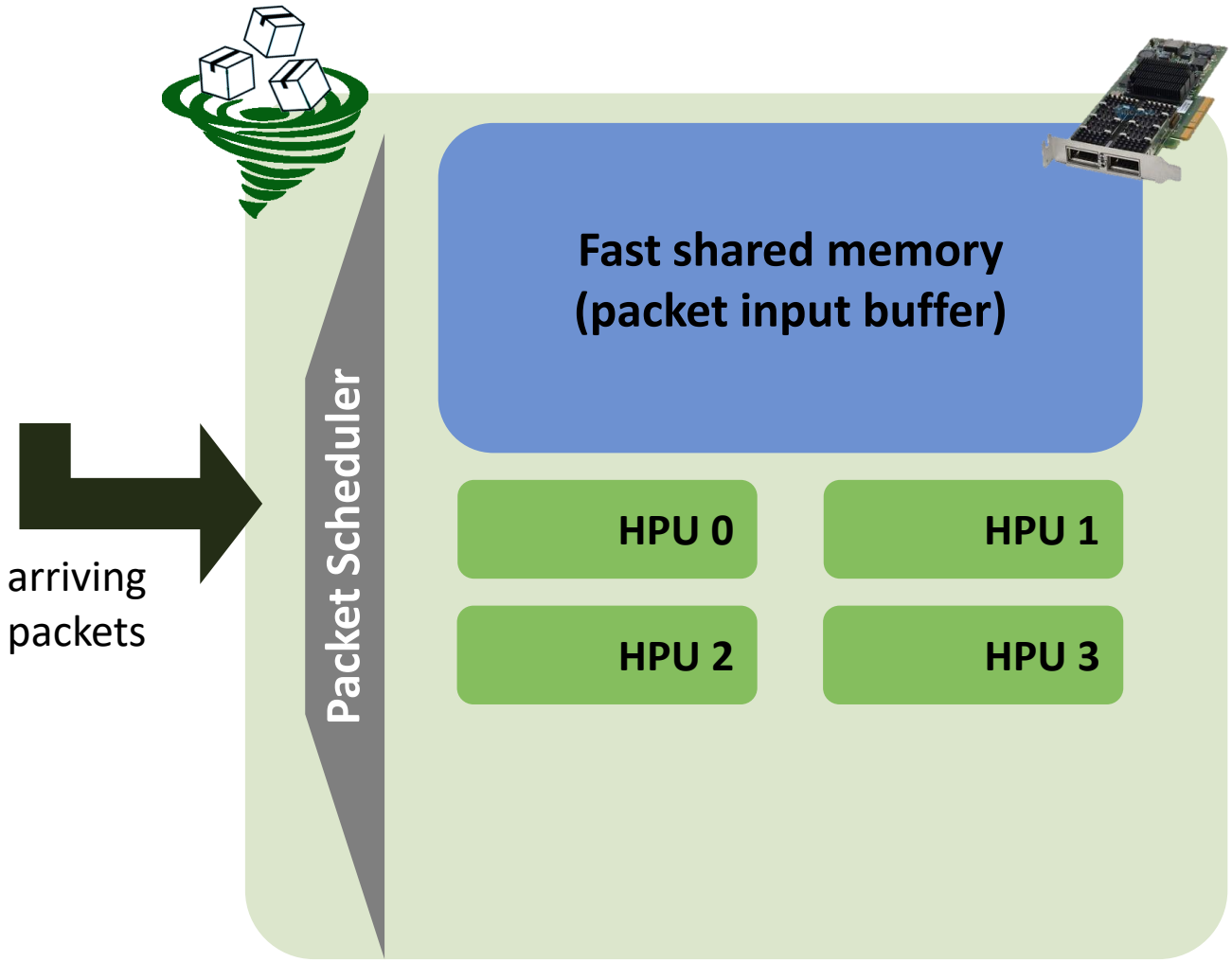
Fast shared memory
(packet input buffer)

sPIN NIC - Abstract Machine Model



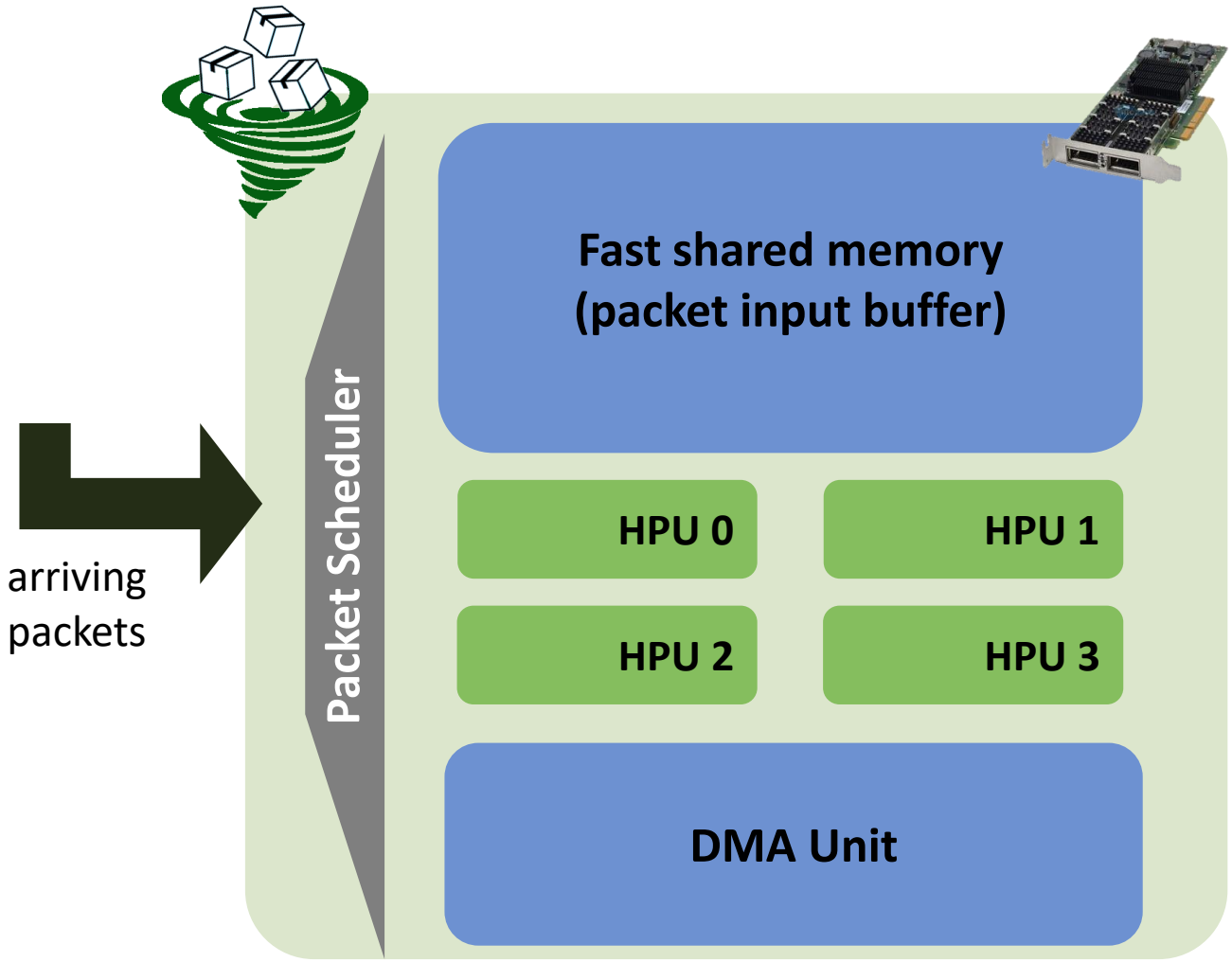
Torsten Hoefler, Salvatore Di Girolamo, Konstantin Taranov, Ryan E. Grant, and Ron Brightwell. "sPIN: High-performance streaming Processing in the Network." SC '17.

sPIN NIC - Abstract Machine Model



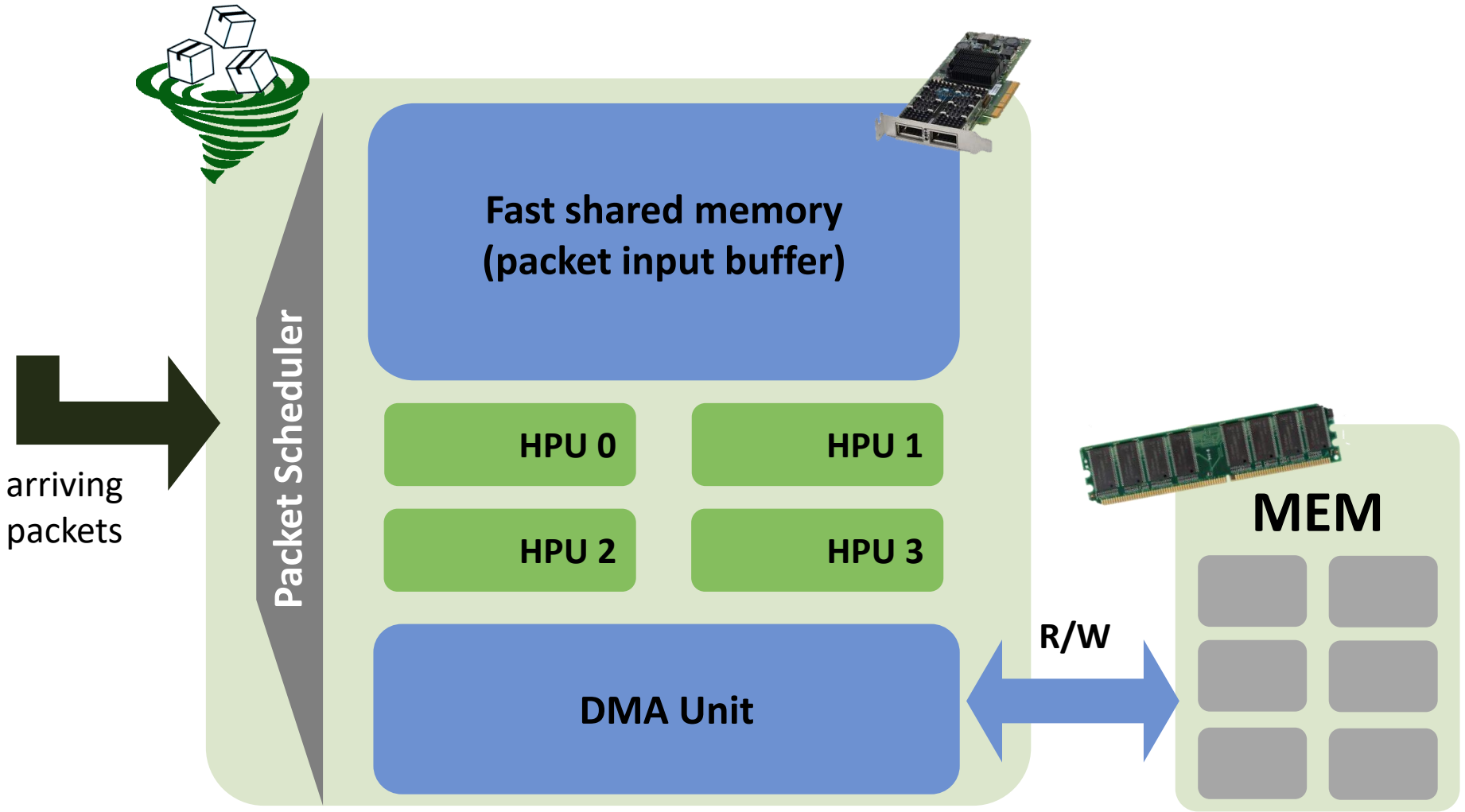
Torsten Hoefler, Salvatore Di Girolamo, Konstantin Taranov, Ryan E. Grant, and Ron Brightwell. "sPIN: High-performance streaming Processing in the Network." SC '17.

sPIN NIC - Abstract Machine Model

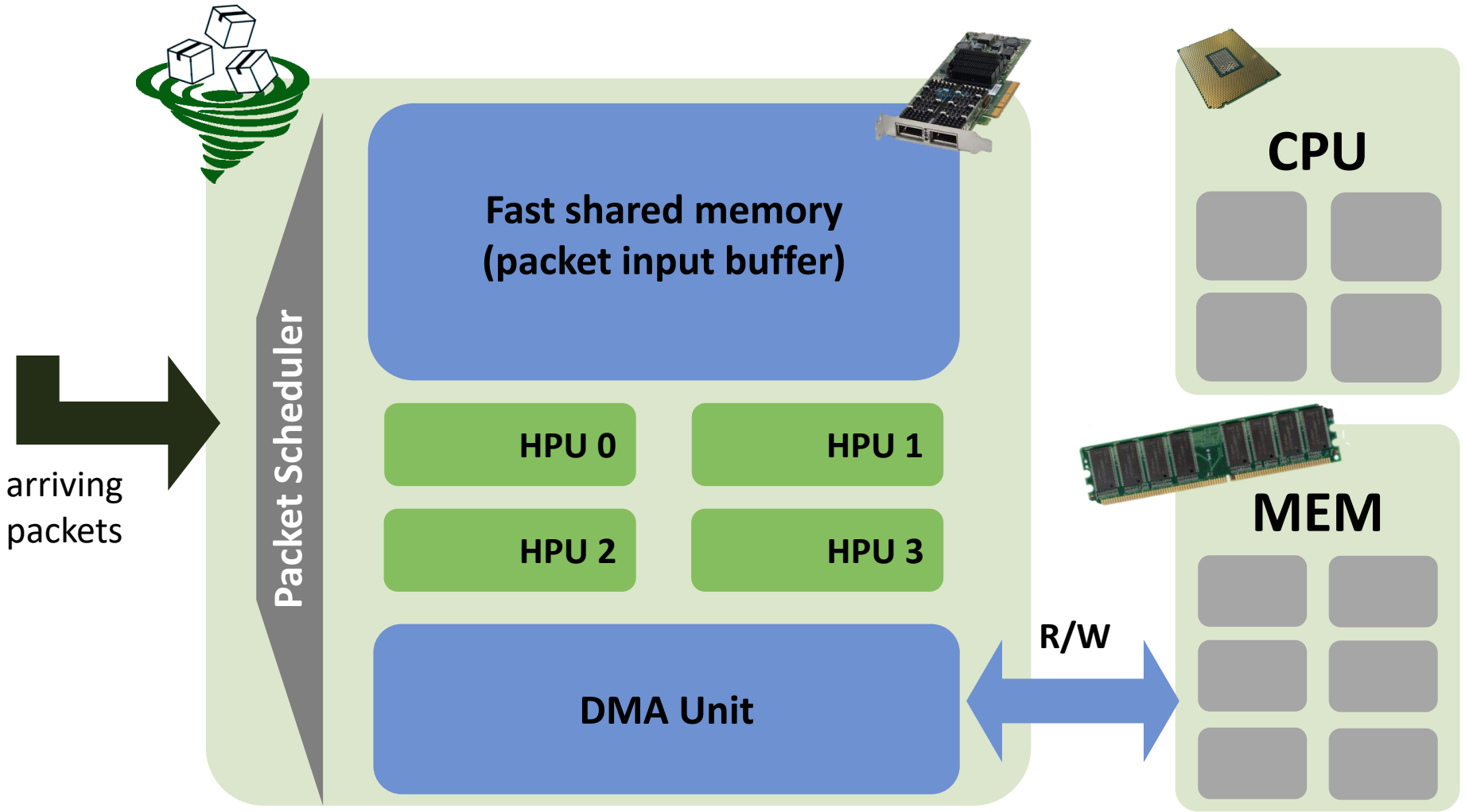


Torsten Hoefler, Salvatore Di Girolamo, Konstantin Taranov, Ryan E. Grant, and Ron Brightwell. "sPIN: High-performance streaming Processing in the Network." SC '17.

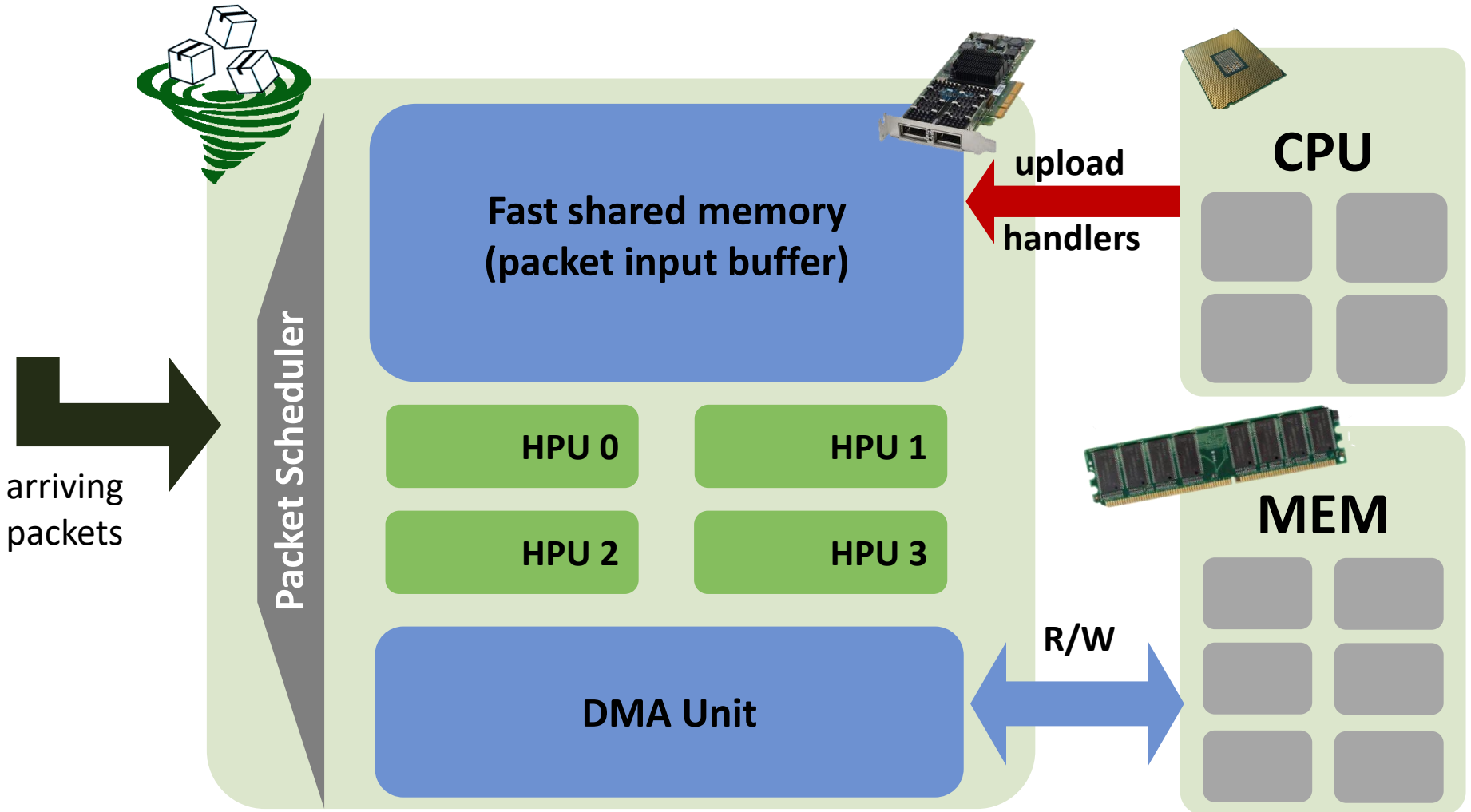
sPIN NIC - Abstract Machine Model



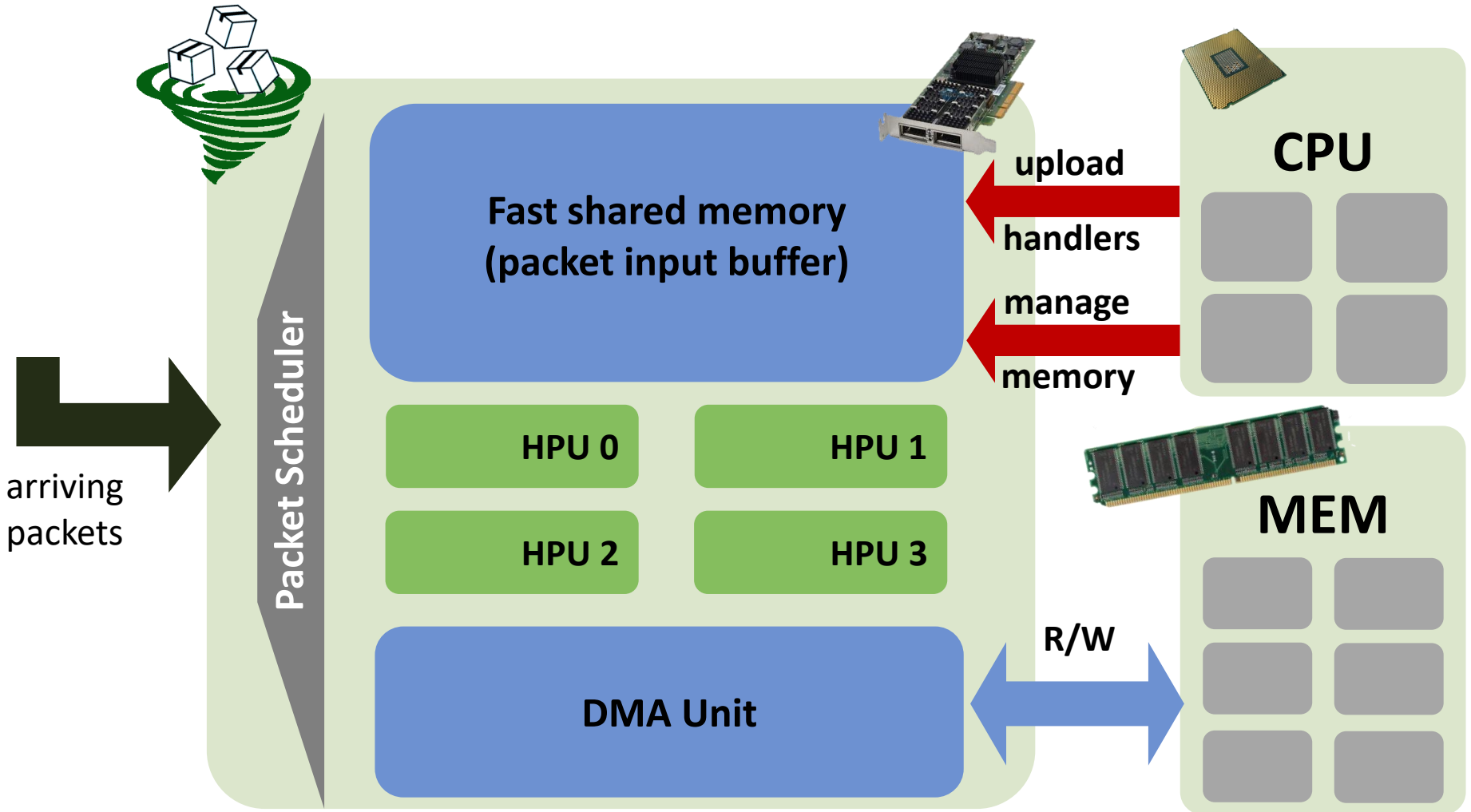
sPIN NIC - Abstract Machine Model



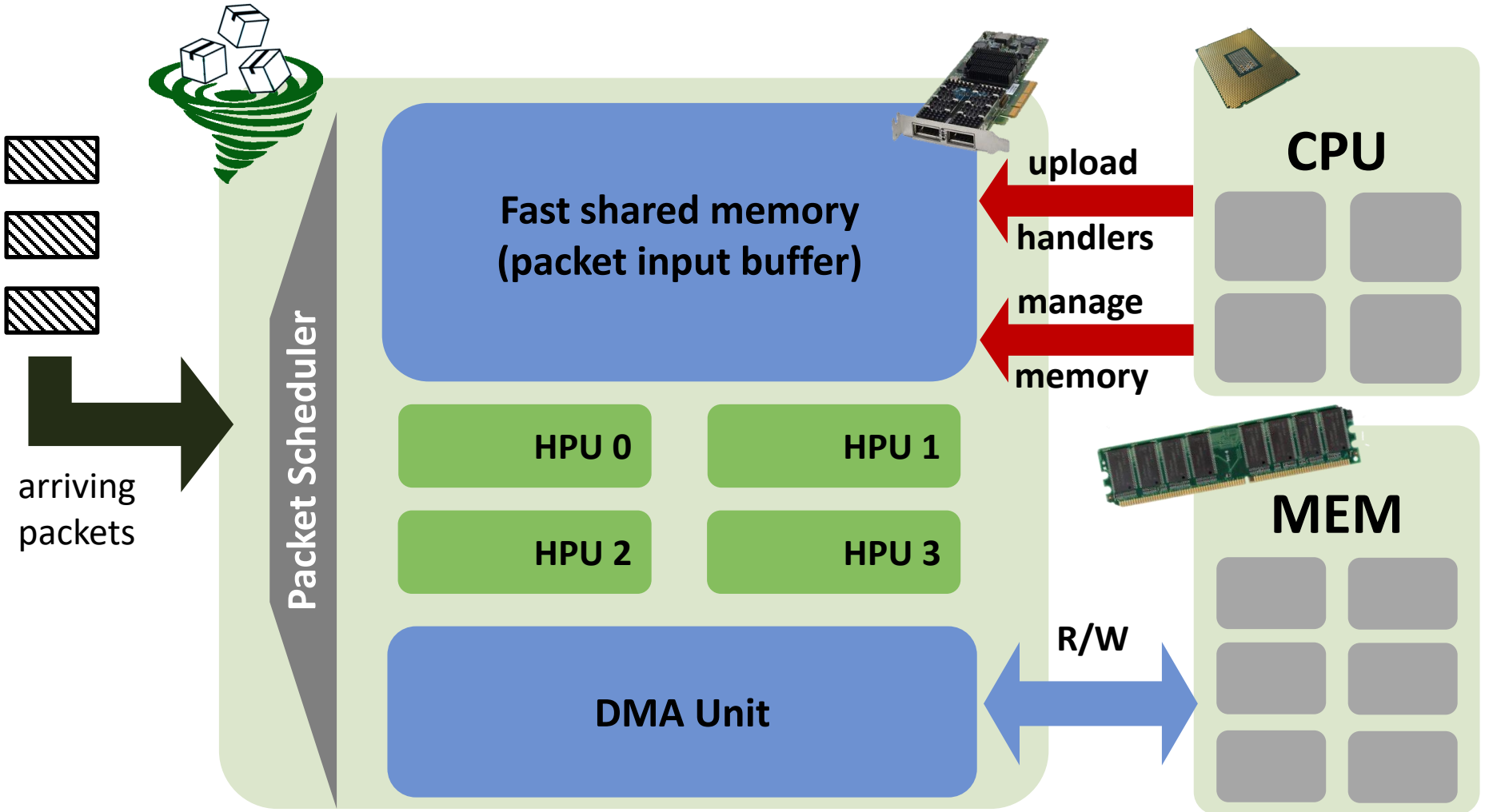
sPIN NIC - Abstract Machine Model



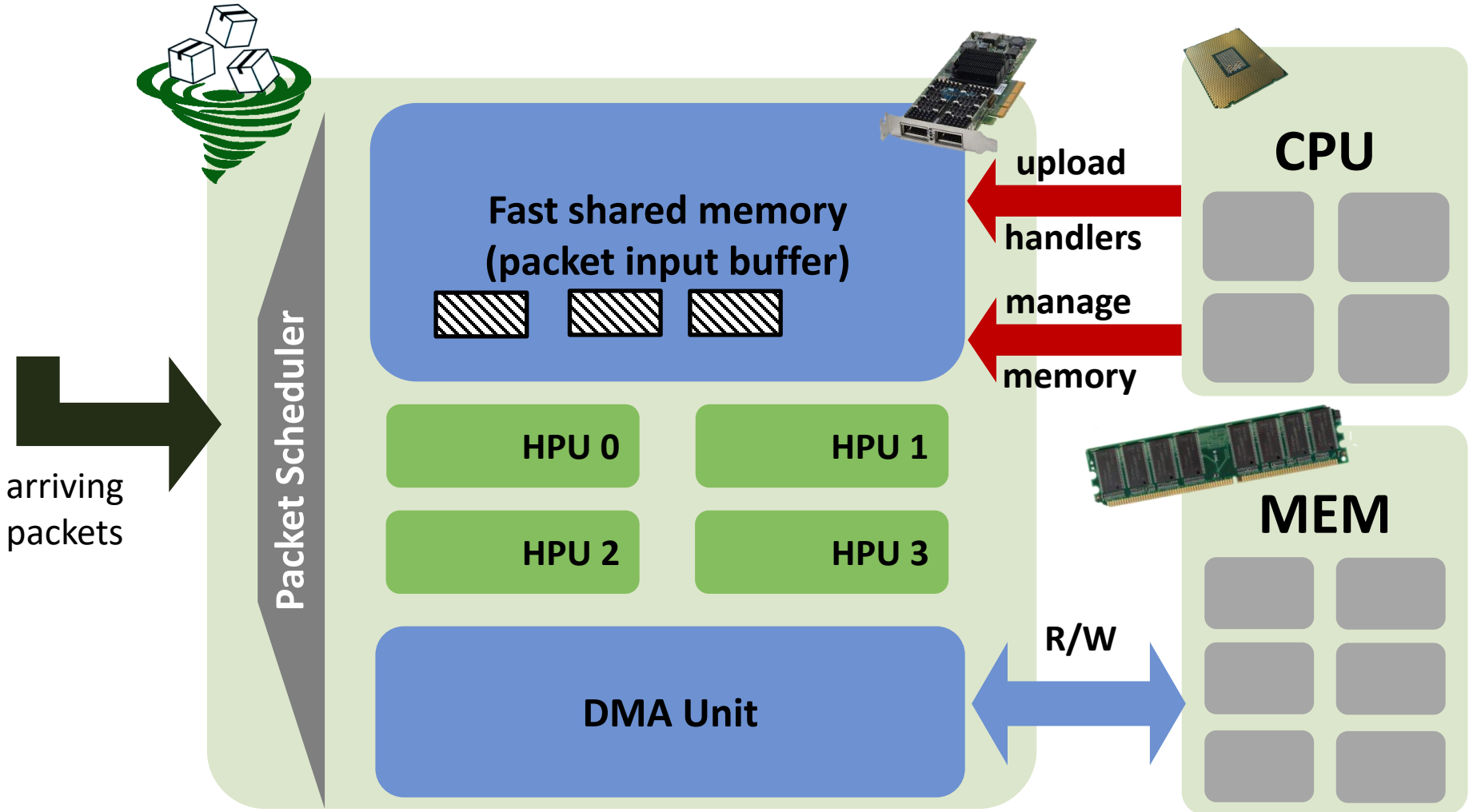
sPIN NIC - Abstract Machine Model



sPIN NIC - Abstract Machine Model

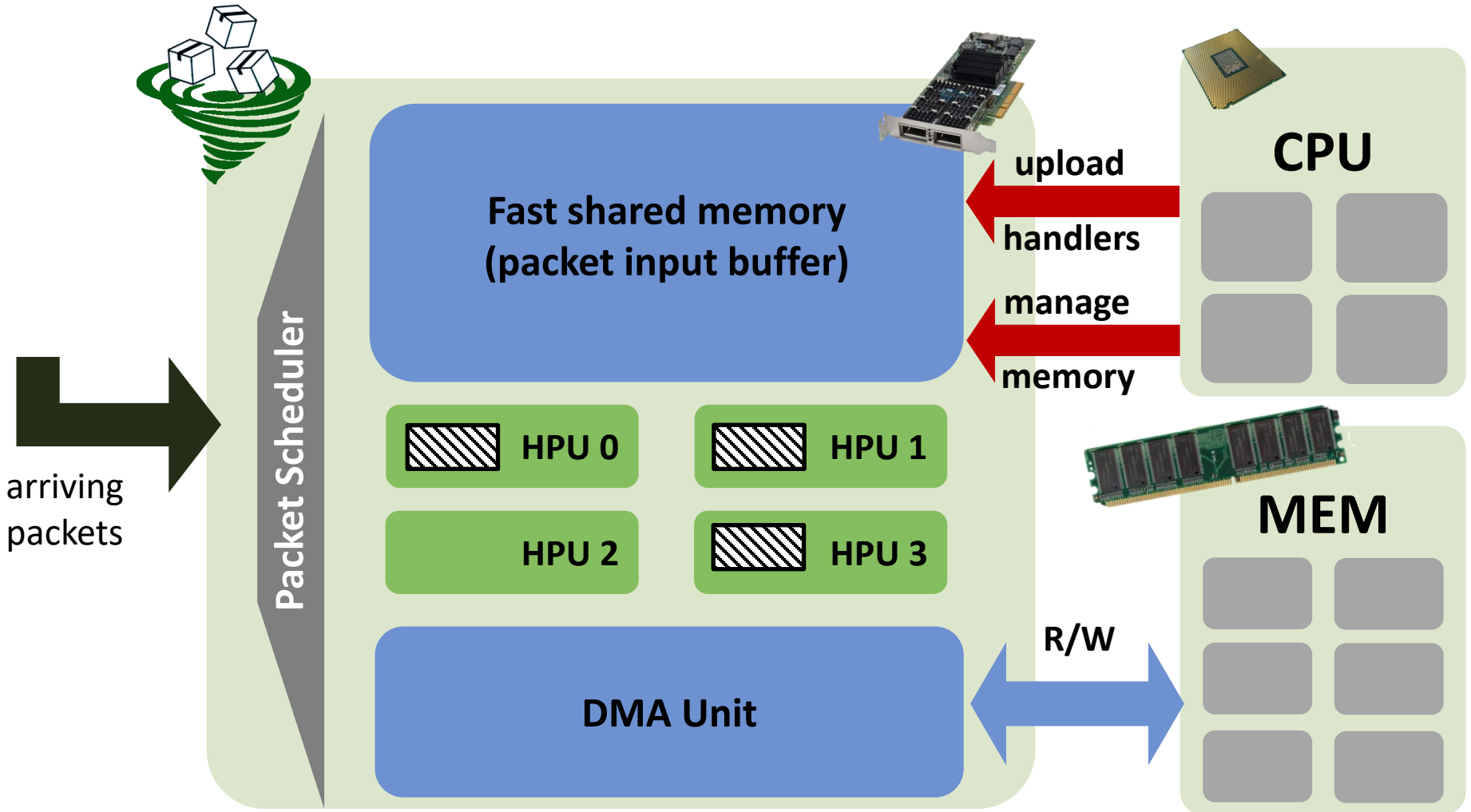


sPIN NIC - Abstract Machine Model

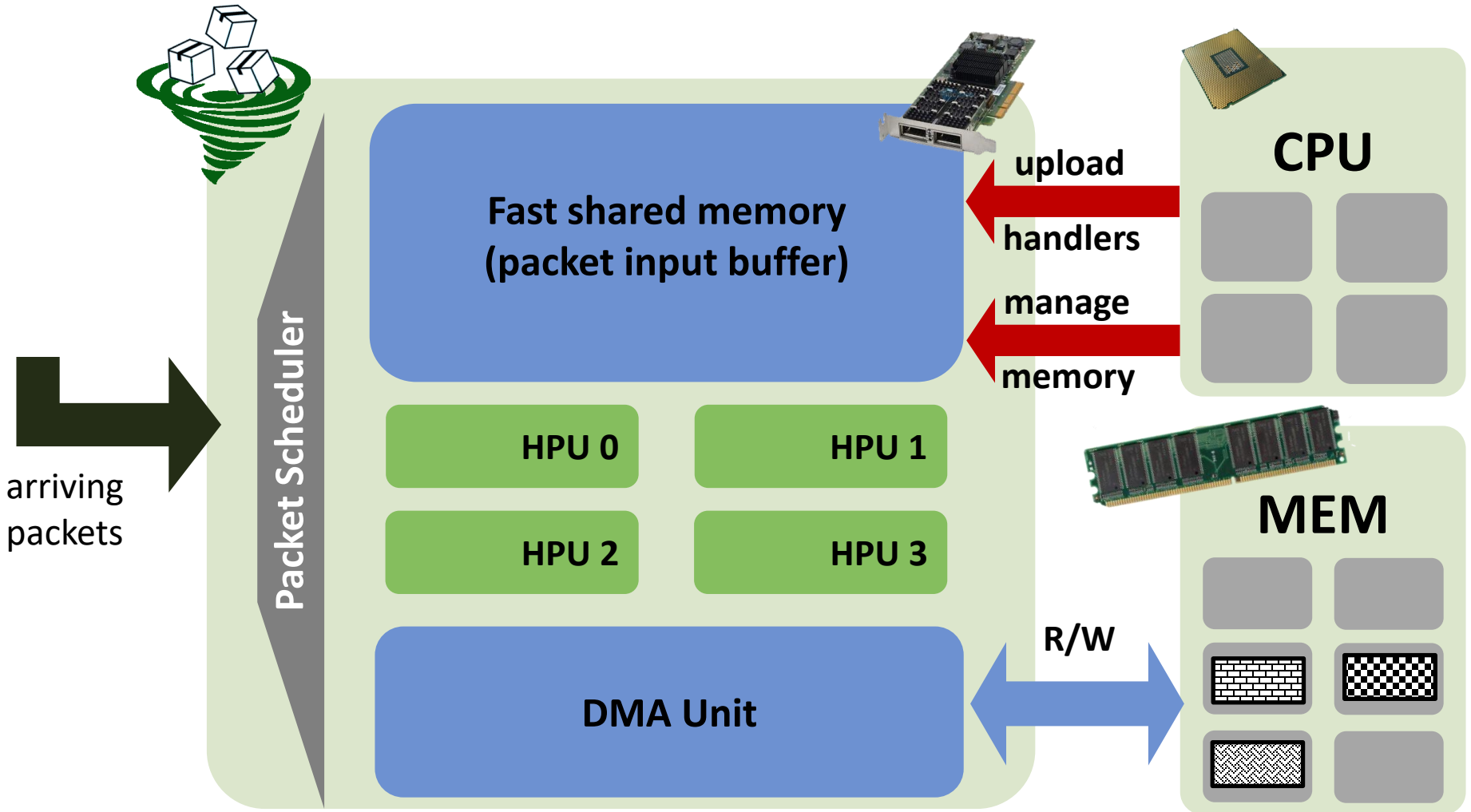


Torsten Hoefler, Salvatore Di Girolamo, Konstantin Taranov, Ryan E. Grant, and Ron Brightwell. "sPIN: High-performance streaming Processing in the Network." SC '17.

sPIN NIC - Abstract Machine Model

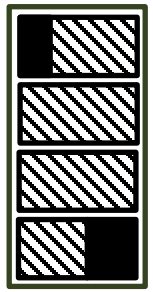


sPIN NIC - Abstract Machine Model

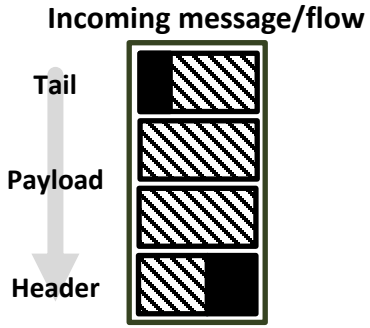


sPIN – Programming Interface

Incoming message/flow

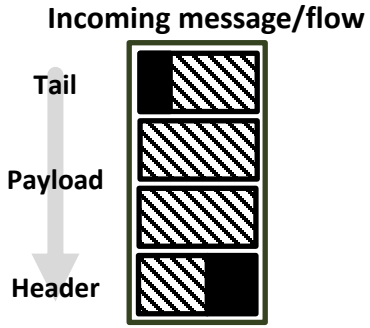


sPIN – Programming Interface



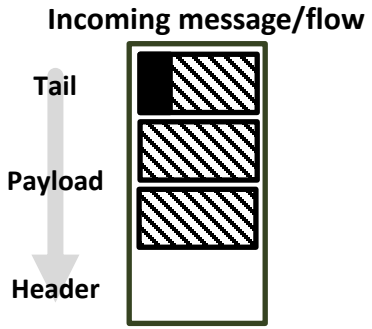
Torsten Hoefler, Salvatore Di Girolamo, Konstantin Taranov, Ryan E. Grant, and Ron Brightwell. "sPIN: High-performance streaming Processing in the Network." SC '17.

sPIN – Programming Interface



Torsten Hoefler, Salvatore Di Girolamo, Konstantin Taranov, Ryan E. Grant, and Ron Brightwell. "sPIN: High-performance streaming Processing in the Network." SC '17.

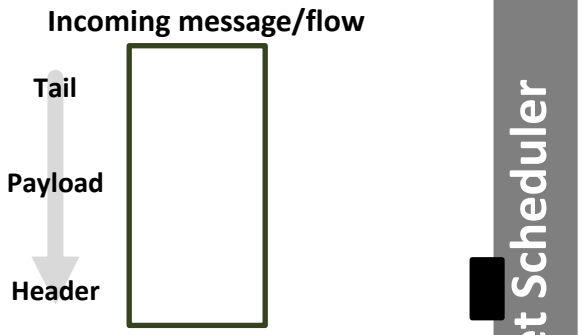
sPIN – Programming Interface

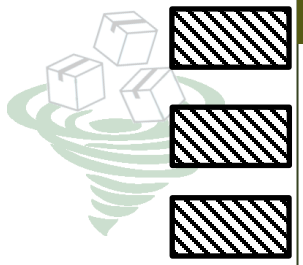


Header handler

```
int hh(const handler_args_t * args){
    return 0;
}
```

sPIN – Programming Interface





Header handler

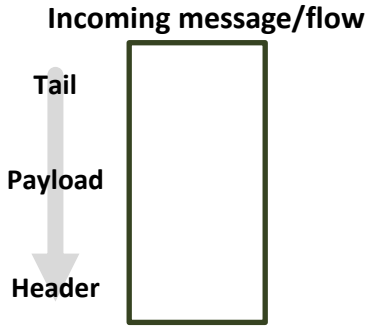
```
int hh(const handler_args_t * args){
    return 0;
}
```

Payload handler

```
int ph (const handler_args_t * args){
   spin_pkt_send(args->pkt_pld, 0, args->source, args->pkt_offset,
                 args->rlength, args->match_bits, args->pkt_pld_len);
    return 0;
}
```

Torsten Hoefler, Salvatore Di Girolamo, Konstantin Taranov, Ryan E. Grant, and Ron Brightwell. "sPIN: High-performance streaming Processing in the Network." SC '17.

sPIN – Programming Interface



Header handler

```
int hh(const handler_args_t * args){
    return 0;
}
```

Payload handler

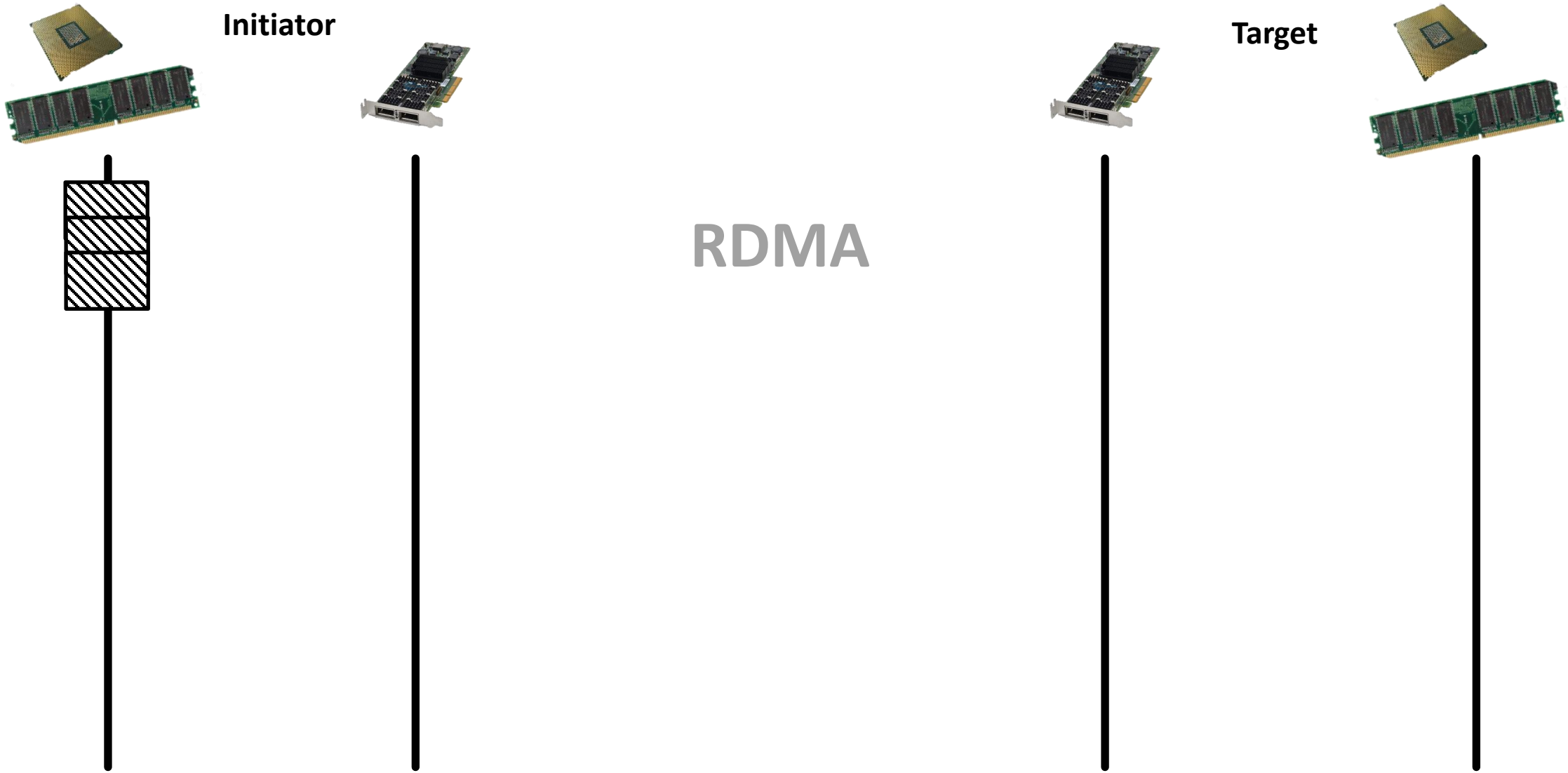
```
int ph (const handler_args_t * args){
   spin_pkt_send(args->pkt_pld, 0, args->source, args->pkt_offset,
                 args->rlength, args->match_bits, args->pkt_pld_len);
    return 0;
}
```

Completion handler

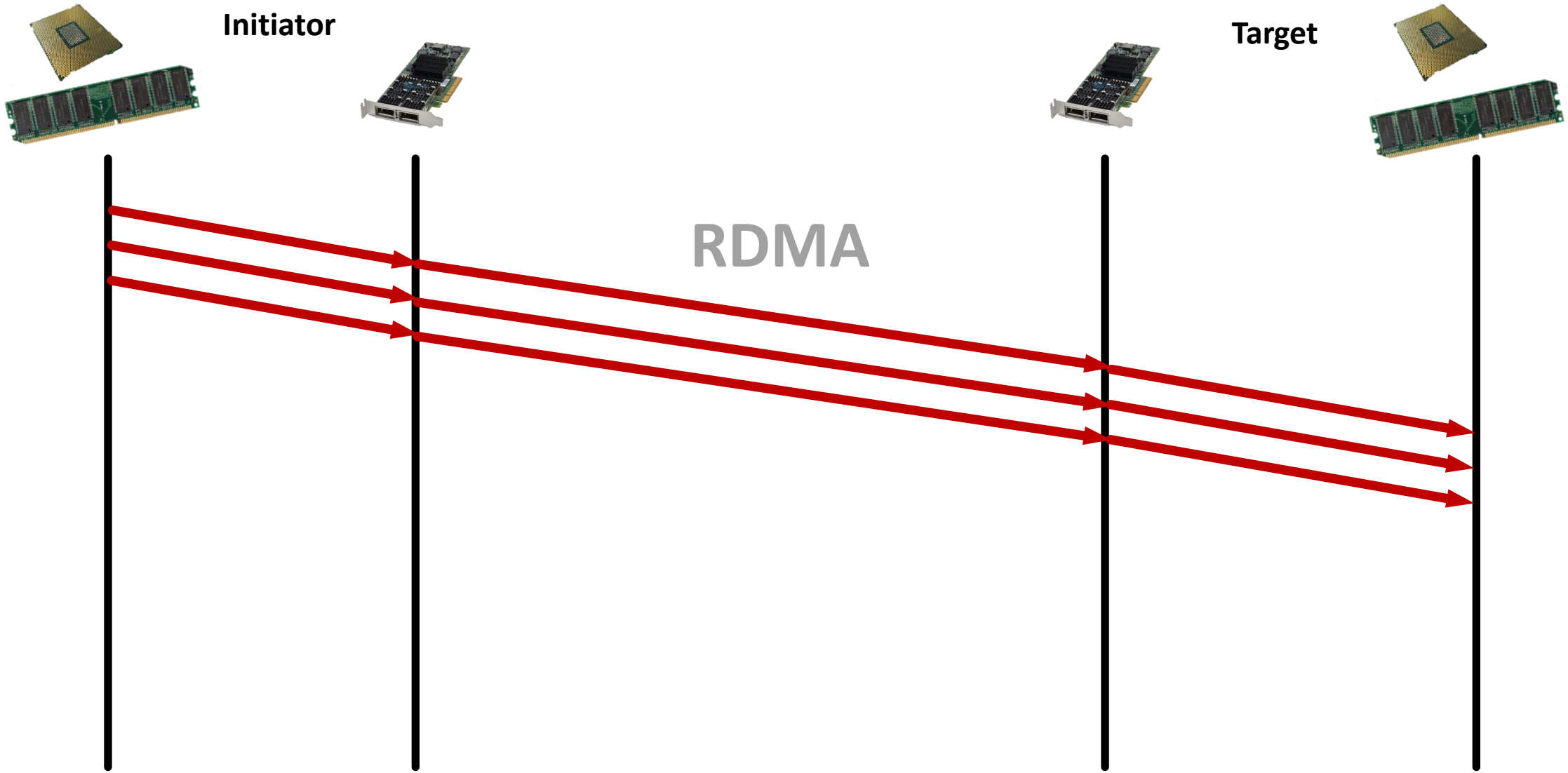
```
int th (const handler_args_t * args)
{
    return 0;
}
```

Torsten Hoefler, Salvatore Di Girolamo, Konstantin Taranov, Ryan E. Grant, and Ron Brightwell. "sPIN: High-performance streaming Processing in the Network." SC '17.

RDMA vs. sPIN in action: Streaming Ping Pong

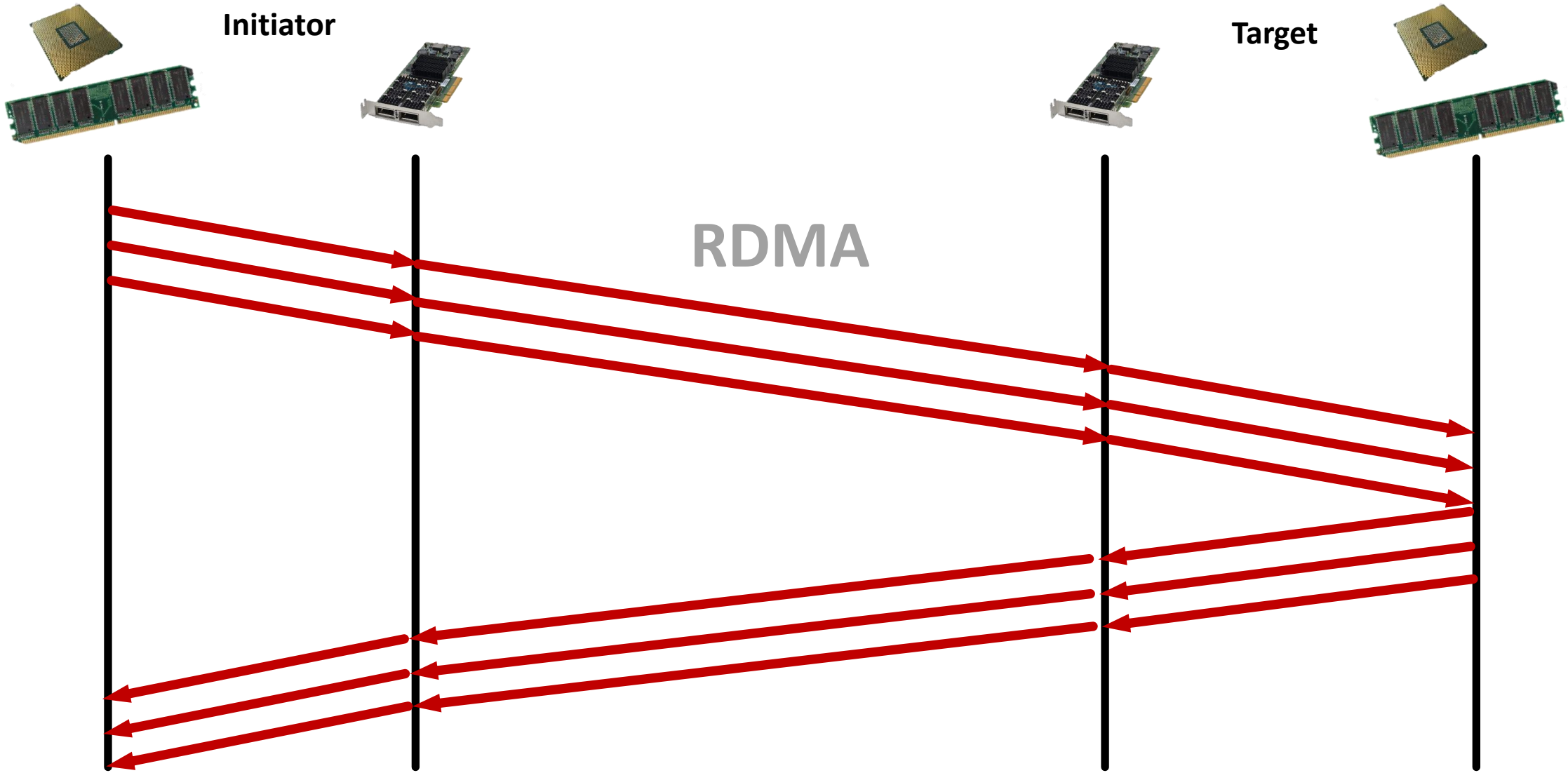


RDMA vs. sPIN in action: Streaming Ping Pong



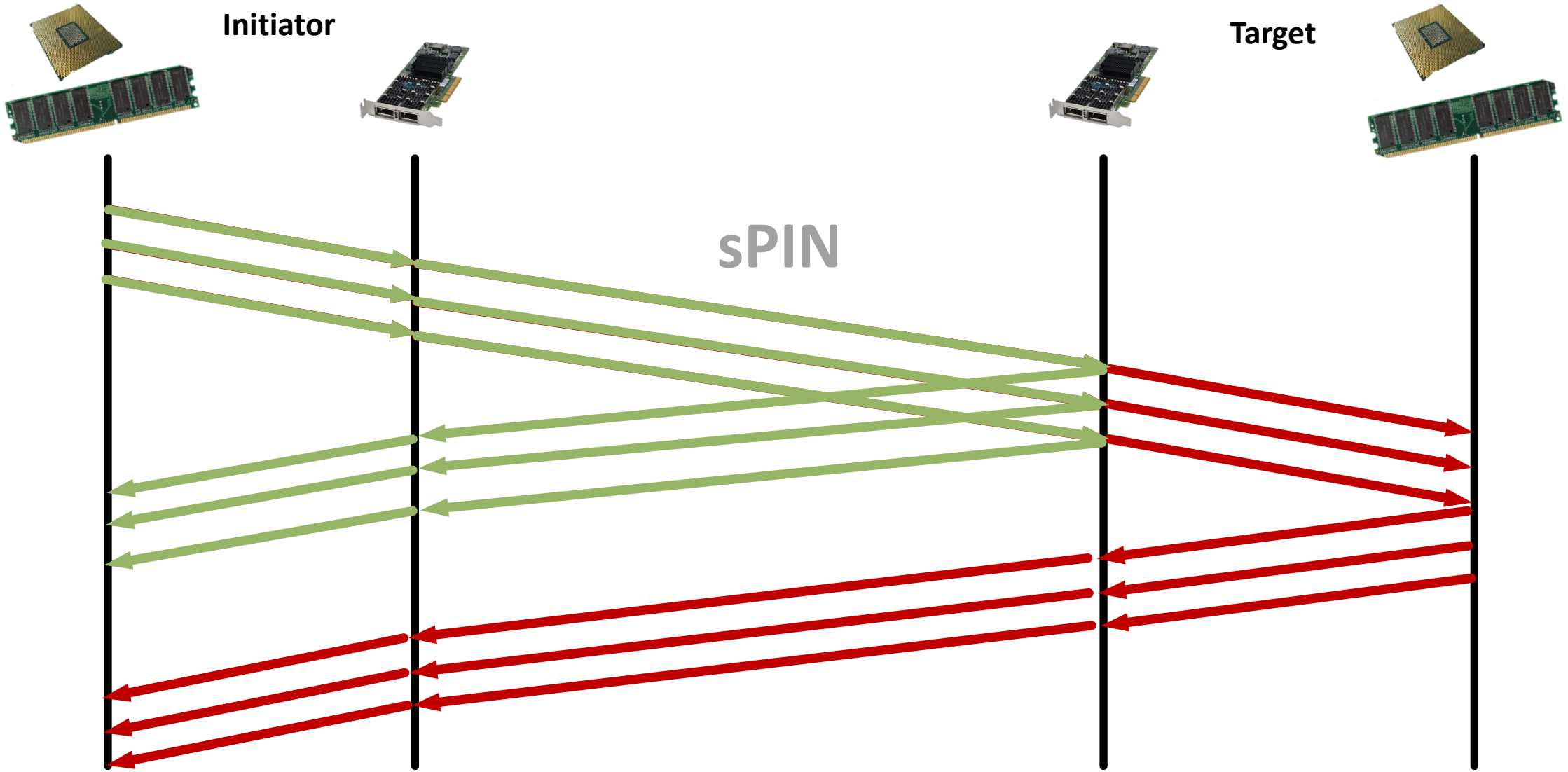
Torsten Hoefler, Salvatore Di Girolamo, Konstantin Taranov, Ryan E. Grant, and Ron Brightwell. "sPIN: High-performance streaming Processing in the Network." SC '17.

RDMA vs. sPIN in action: Streaming Ping Pong



Torsten Hoefler, Salvatore Di Girolamo, Konstantin Taranov, Ryan E. Grant, and Ron Brightwell. "sPIN: High-performance streaming Processing in the Network." SC '17.

RDMA vs. sPIN in action: Streaming Ping Pong



Torsten Hoefler, Salvatore Di Girolamo, Konstantin Taranov, Ryan E. Grant, and Ron Brightwell. "sPIN: High-performance streaming Processing in the Network." SC '17.

Architectural principles for in-network compute

Low latency, full throughput

Support for wide range of use cases

Easy to integrate

Salvatore Di Girolamo, Andreas Kurth, Alexandru Calotoiu, Thomas Benz, Timo Schneider, Jakub Beránek, Luca Benini, and Torsten Hoefler. "A RISC-V in-network accelerator for flexible high-performance low-power packet processing." ISCA '21

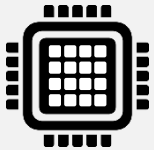
Architectural principles for in-network compute

Architectural principles for in-network compute

Low latency, full throughput

Architectural principles for in-network compute

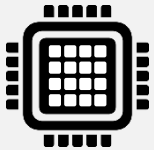
Low latency, full throughput



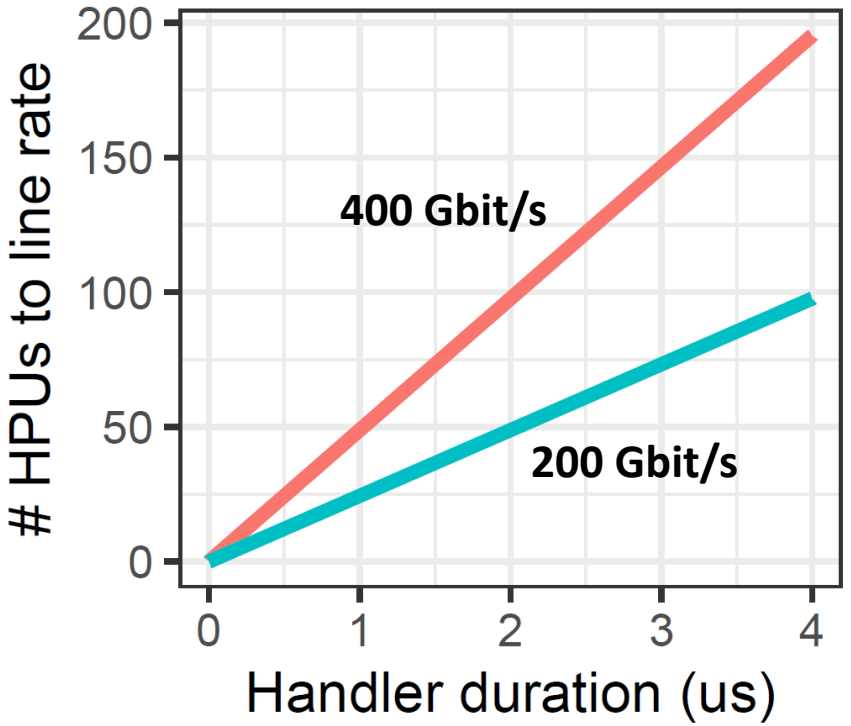
Highly parallel

Architectural principles for in-network compute

Low latency, full throughput

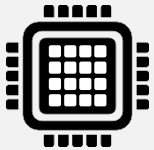


Highly parallel



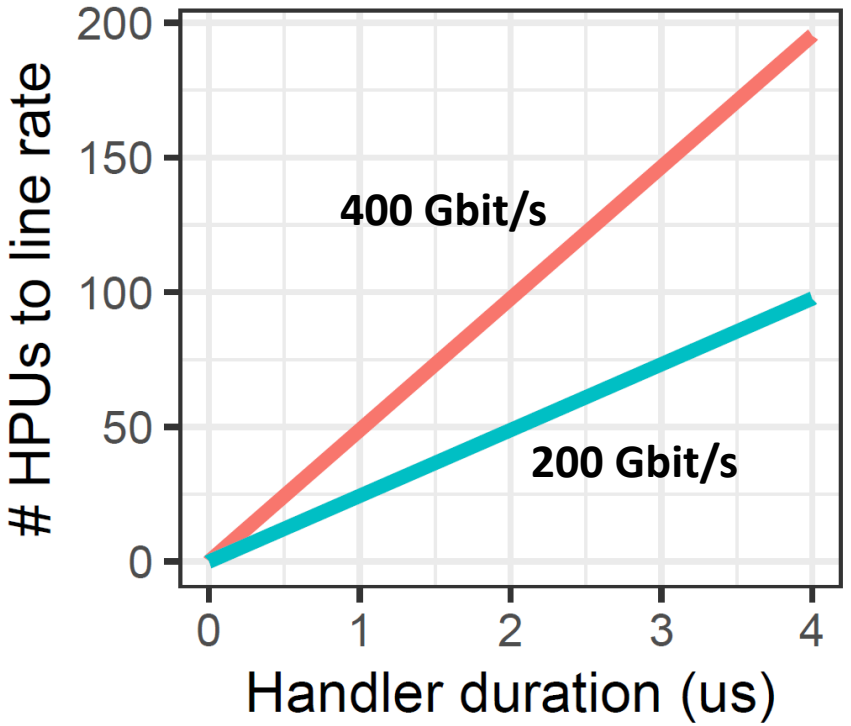
Architectural principles for in-network compute

Low latency, full throughput



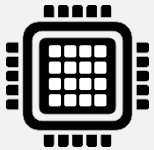
Highly parallel

Fast scheduling



Architectural principles for in-network compute

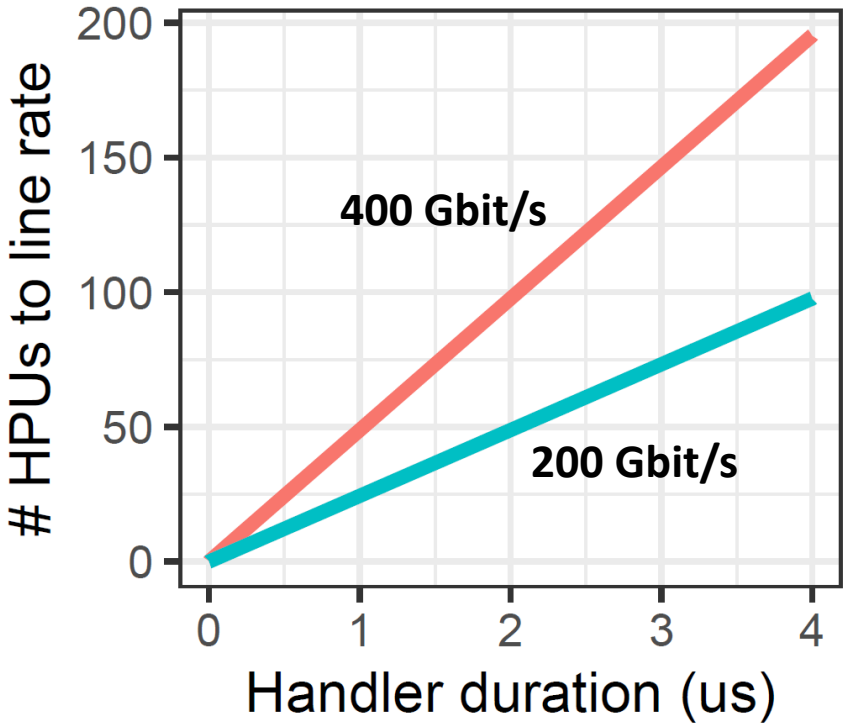
Low latency, full throughput



Highly parallel

Fast scheduling

Fast explicit
memory access



Architectural principles for in-network compute

Architectural principles for in-network compute

Support for wide range of use cases

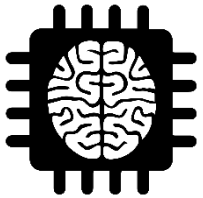
Architectural principles for in-network compute

Support for wide range of use cases

Network-accelerated datatypes

sPIN-FS

Zoo-sPINNER
consensus protocols



Quantization
Allreduce and other collectives

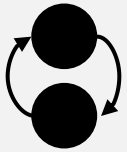
Packet classification and pattern matching

Serverless

Erasure coding

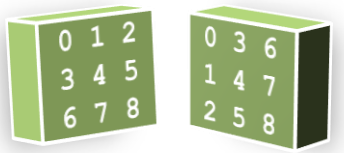
Architectural principles for in-network compute

Support for wide range of use cases



Stateful computation support

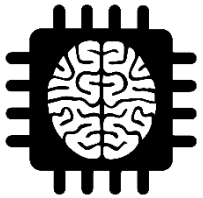
Handlers isolation



Network-accelerated datatypes

sPIN-FS

Zoo-sPINNER
consensus protocols



Quantization
Allreduce and other collectives

Packet classification and pattern matching

Serverless

Erasure coding

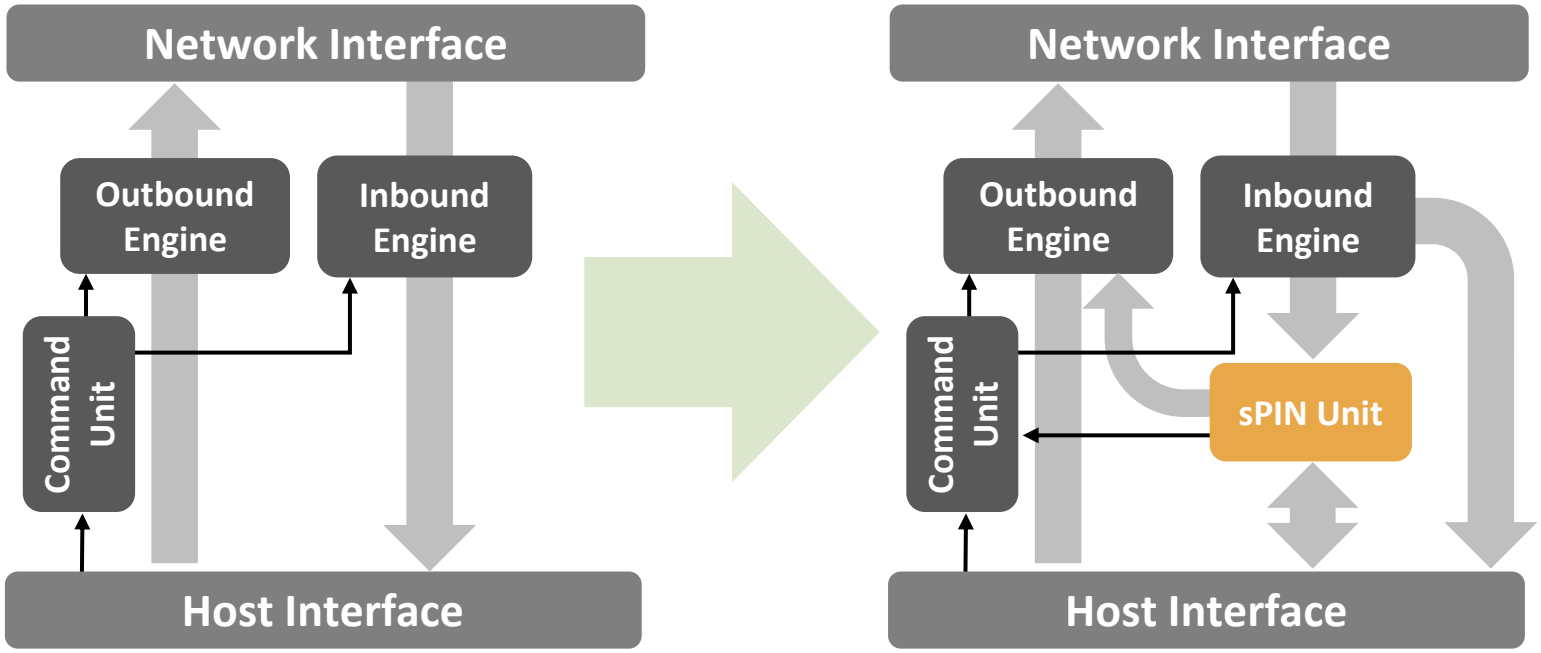
Architectural principles for in-network compute

Architectural principles for in-network compute

Easy to integrate

Architectural principles for in-network compute

Easy to integrate

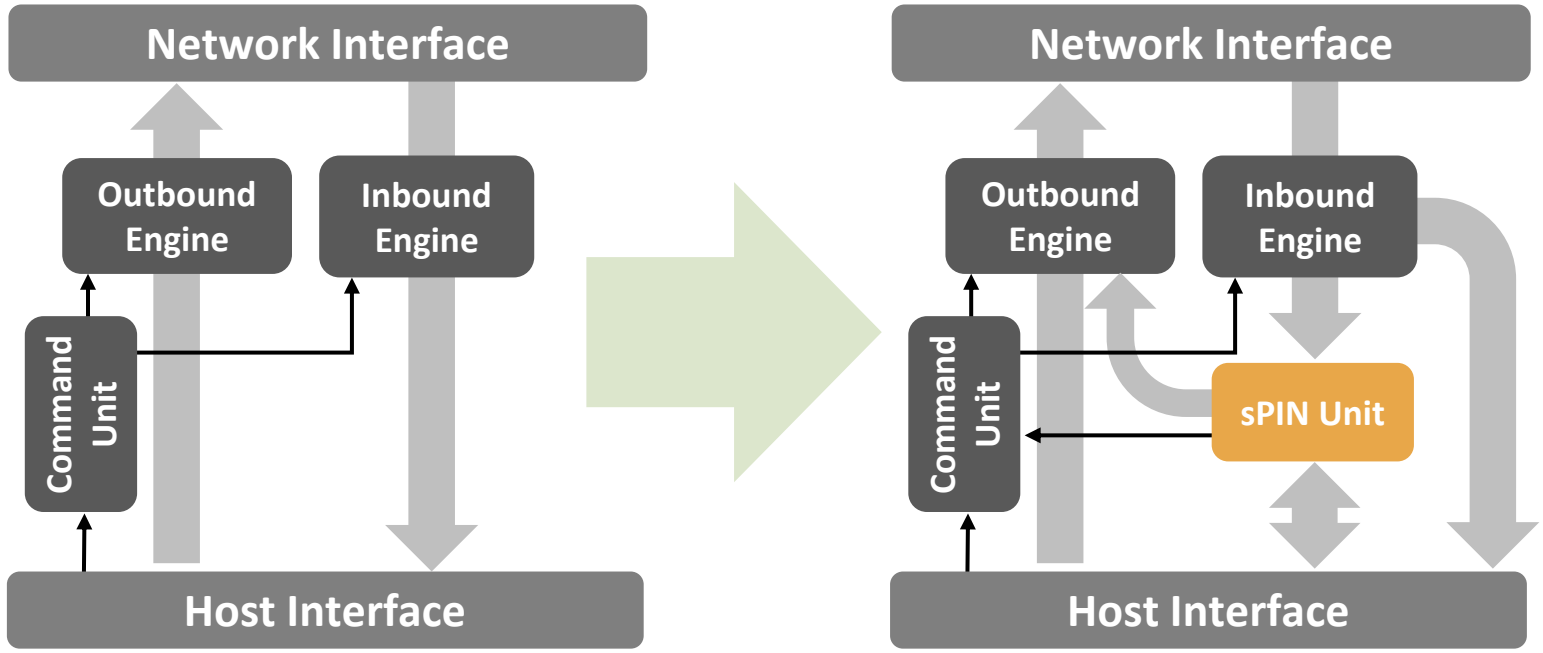


Architectural principles for in-network compute

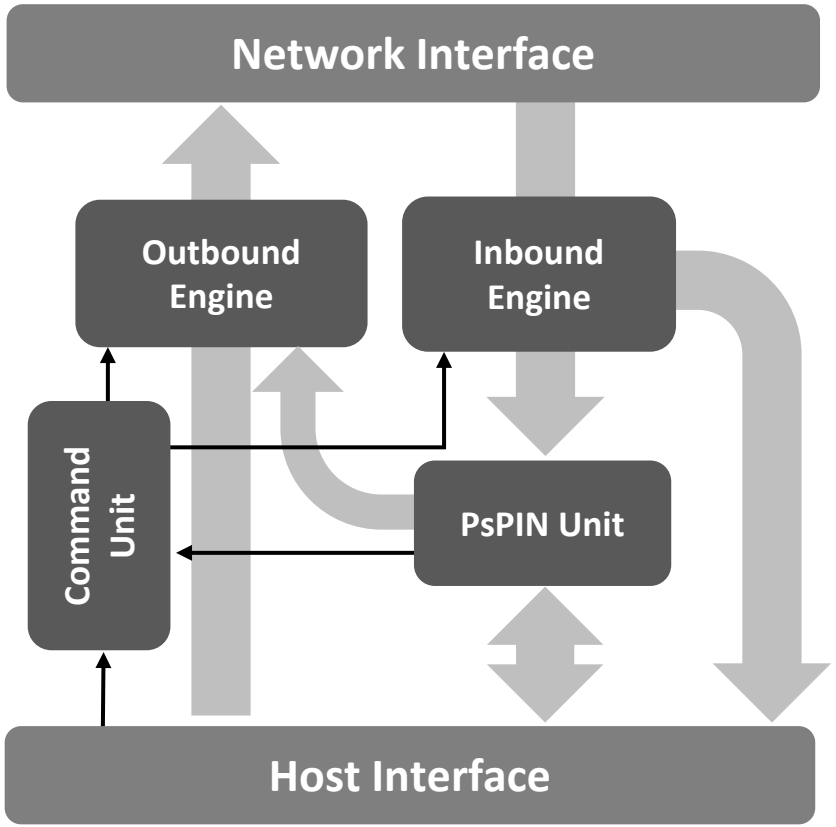
Easy to integrate

Area and power efficiency

Configurability

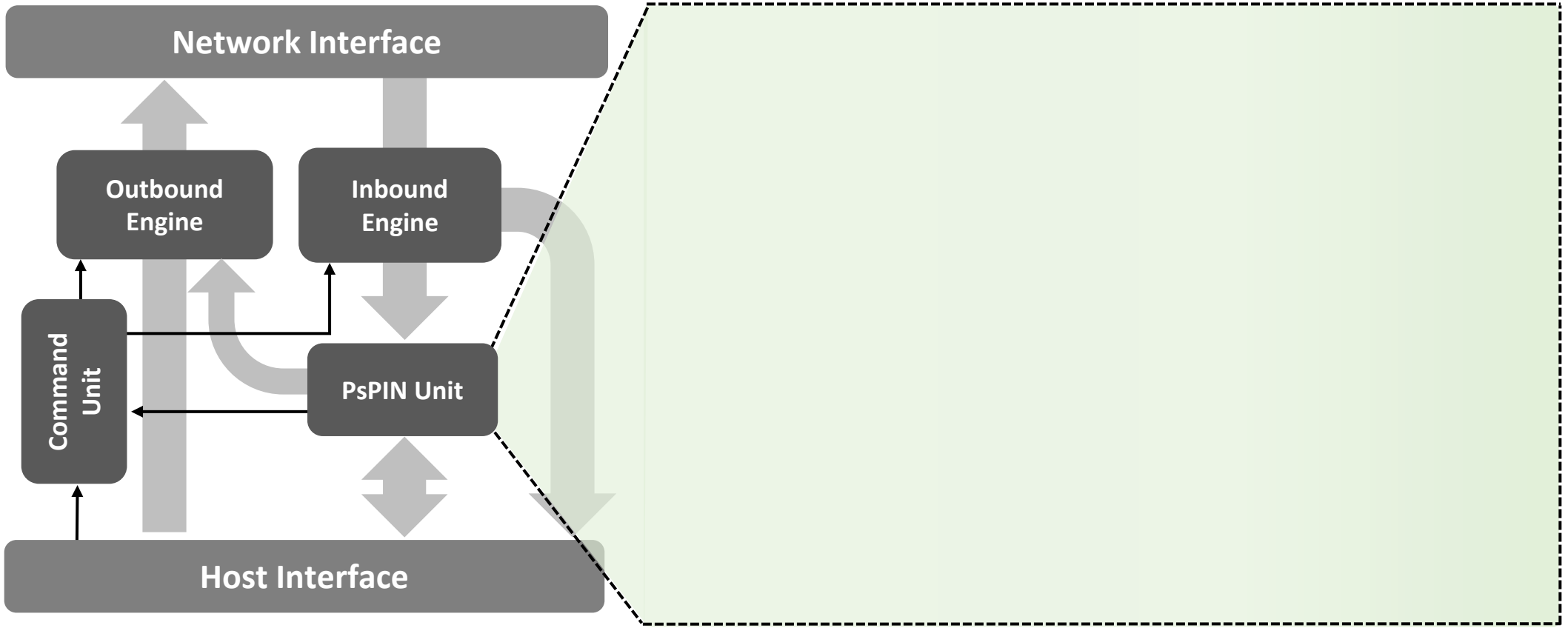


PsPIN: A PULP-powered implementation of sPIN



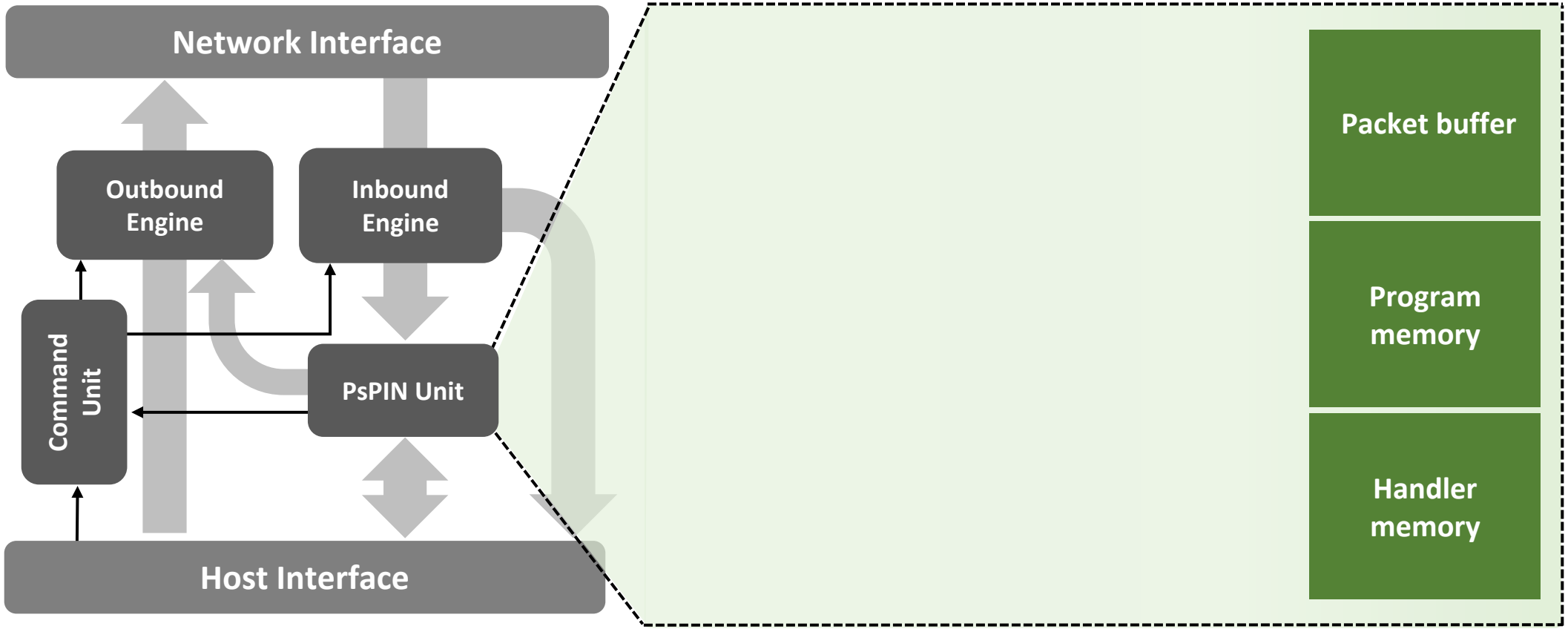
Salvatore Di Girolamo, Andreas Kurth, Alexandru Calotoiu, Thomas Benz, Timo Schneider, Jakub Beránek, Luca Benini, and Torsten Hoefler. "A RISC-V in-network accelerator for flexible high-performance low-power packet processing." ISCA '21

PsPIN: A PULP-powered implementation of sPIN



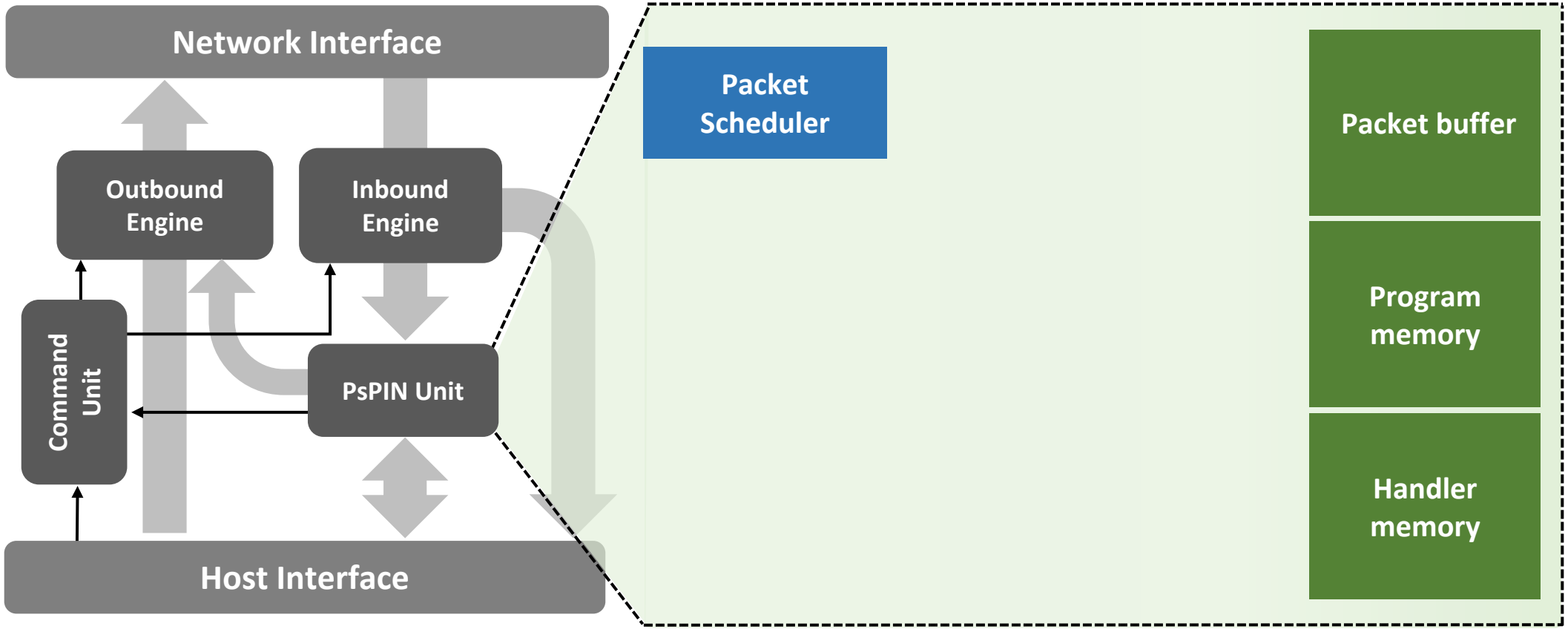
Salvatore Di Girolamo, Andreas Kurth, Alexandru Calotoiu, Thomas Benz, Timo Schneider, Jakub Beránek, Luca Benini, and Torsten Hoefler. "A RISC-V in-network accelerator for flexible high-performance low-power packet processing." ISCA '21

PsPIN: A PULP-powered implementation of sPIN



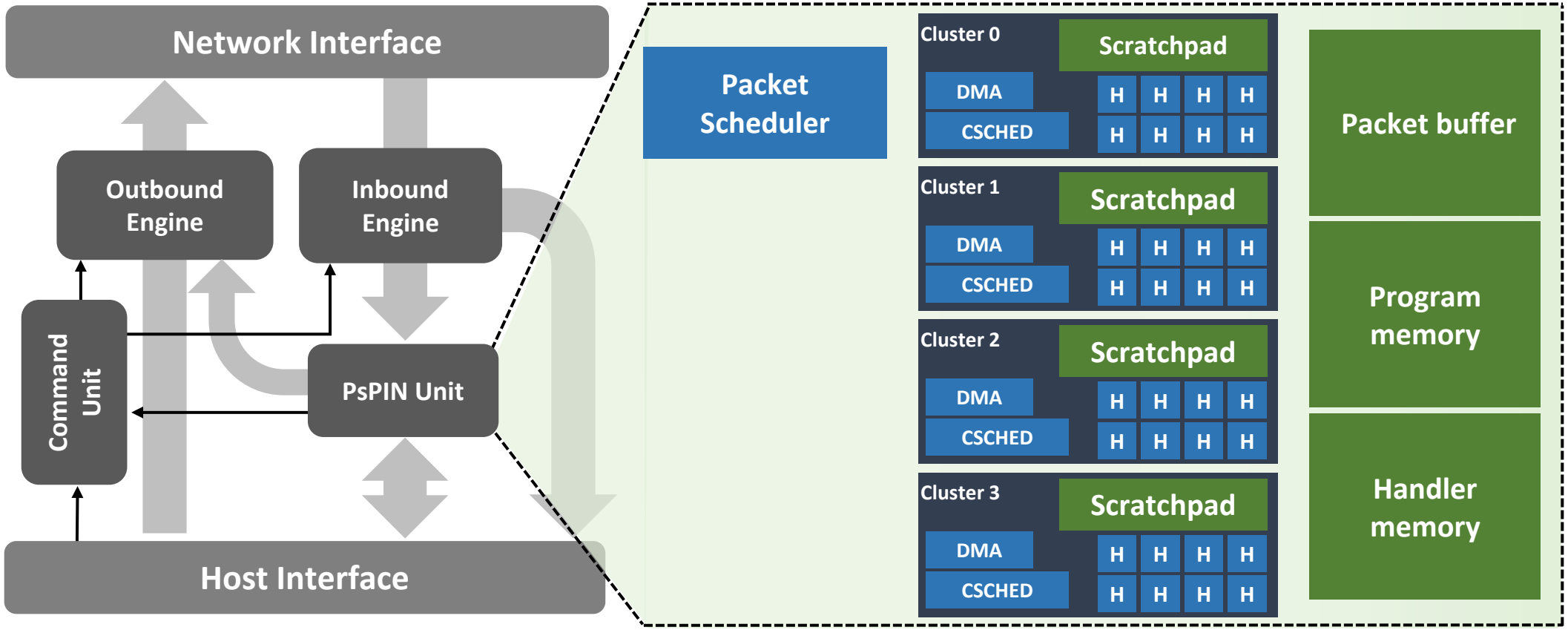
Salvatore Di Girolamo, Andreas Kurth, Alexandru Calotoiu, Thomas Benz, Timo Schneider, Jakub Beránek, Luca Benini, and Torsten Hoefler. "A RISC-V in-network accelerator for flexible high-performance low-power packet processing." ISCA '21

PsPIN: A PULP-powered implementation of sPIN



Salvatore Di Girolamo, Andreas Kurth, Alexandru Calotoiu, Thomas Benz, Timo Schneider, Jakub Beránek, Luca Benini, and Torsten Hoefler. "A RISC-V in-network accelerator for flexible high-performance low-power packet processing." ISCA '21

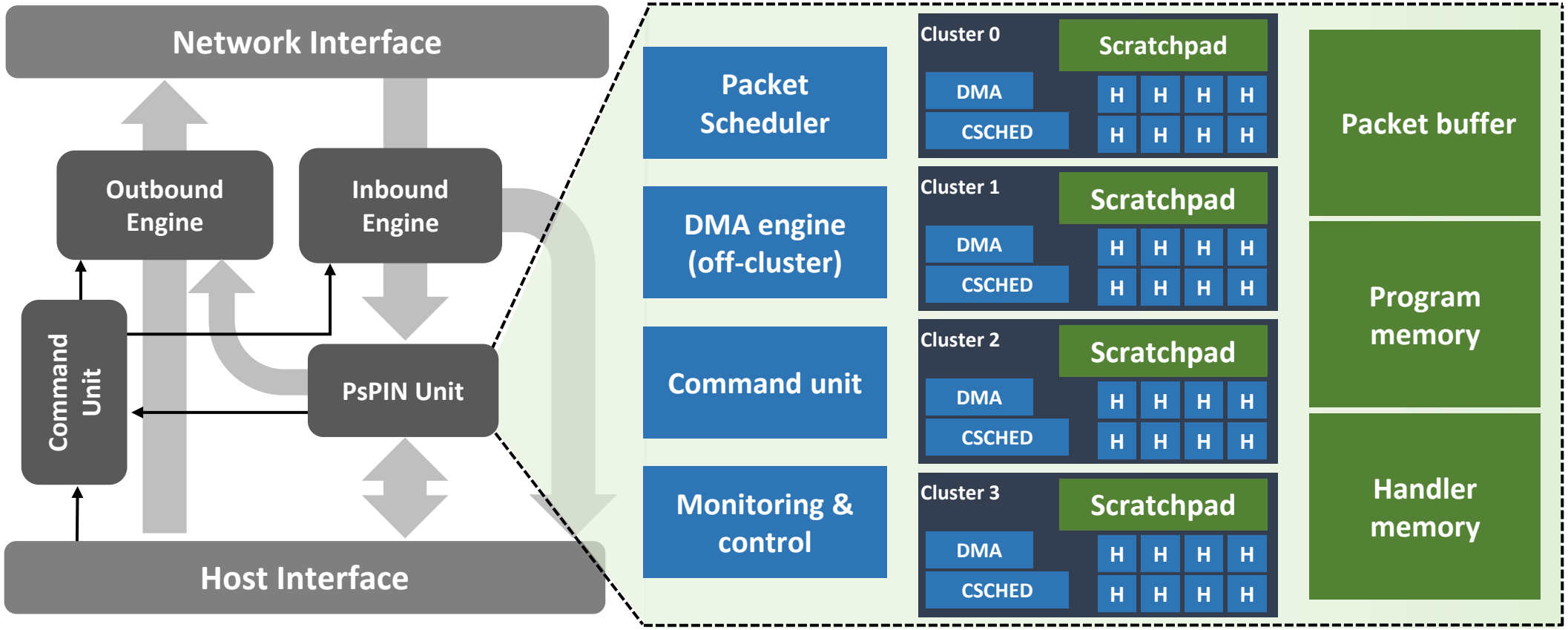
PsPIN: A PULP-powered implementation of sPIN



H cv32e40p (aka RI5CY): RISC-V, 4 stage pipeline, in-order, 32 bit (40 kGE)

Salvatore Di Girolamo, Andreas Kurth, Alexandru Calotoiu, Thomas Benz, Timo Schneider, Jakub Beránek, Luca Benini, and Torsten Hoefler. "A RISC-V in-network accelerator for flexible high-performance low-power packet processing." ISCA '21

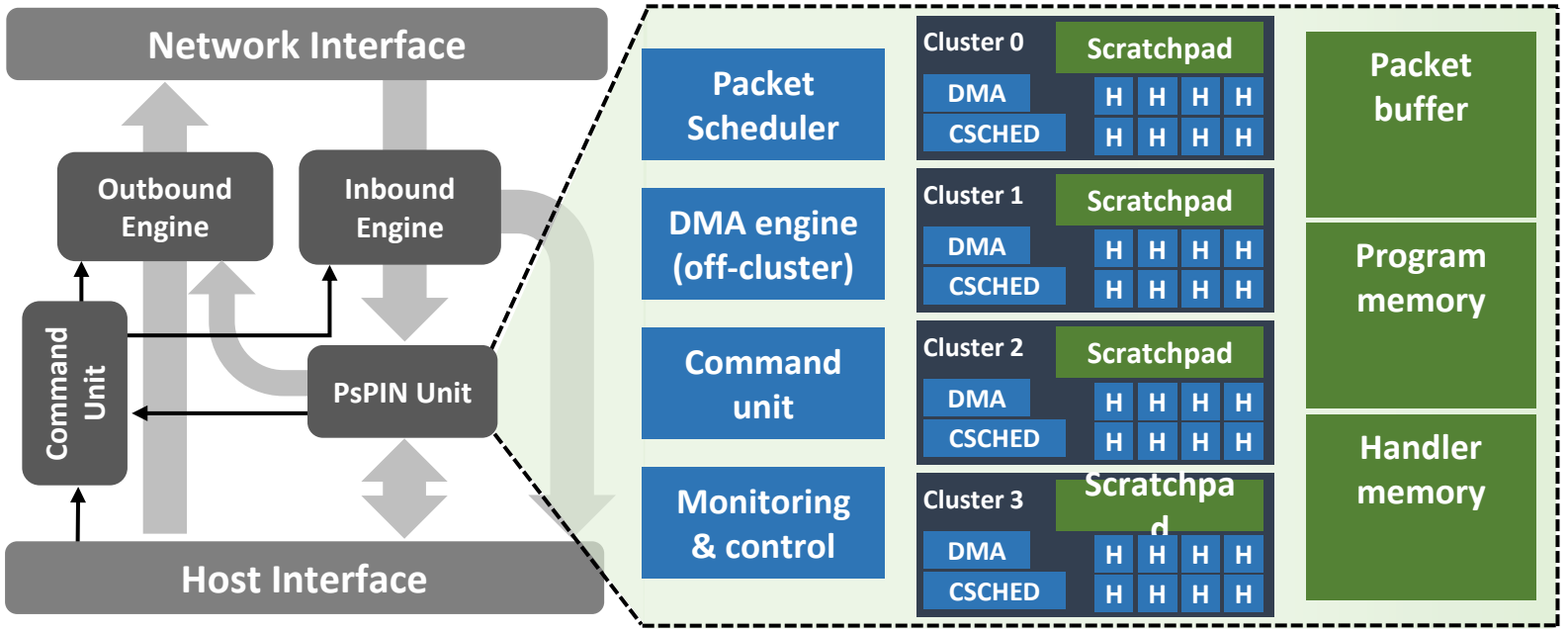
PsPIN: A PULP-powered implementation of sPIN



H cv32e40p (aka RI5CY): RISC-V, 4 stage pipeline, in-order, 32 bit (40 kGE)

Salvatore Di Girolamo, Andreas Kurth, Alexandru Calotoiu, Thomas Benz, Timo Schneider, Jakub Beránek, Luca Benini, and Torsten Hoefler. "A RISC-V in-network accelerator for flexible high-performance low-power packet processing." ISCA '21

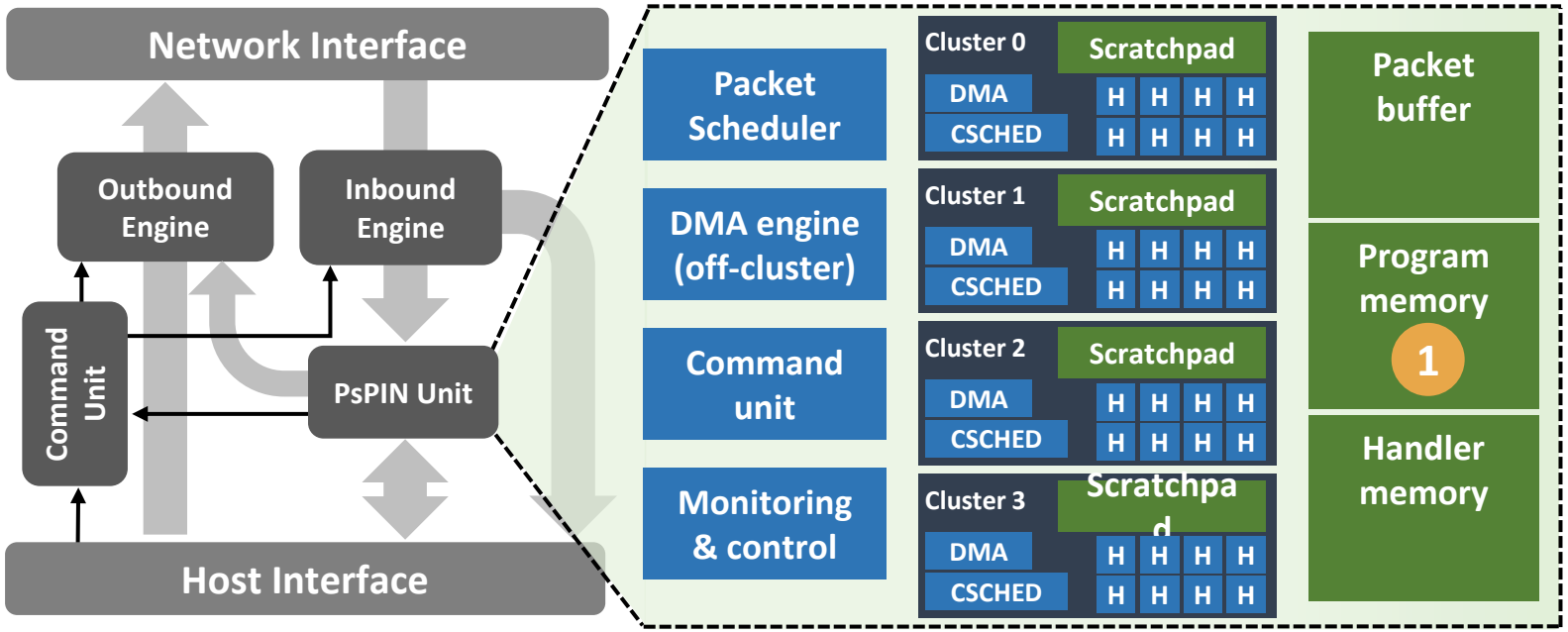
Application perspective



Application perspective

1 Define and offload handlers

Telemetry:
telemetry_hh(), telemetry_ph(), telemetry_th();
Filtering:
filter_hh(), filter_ph, filter_th();



Application perspective

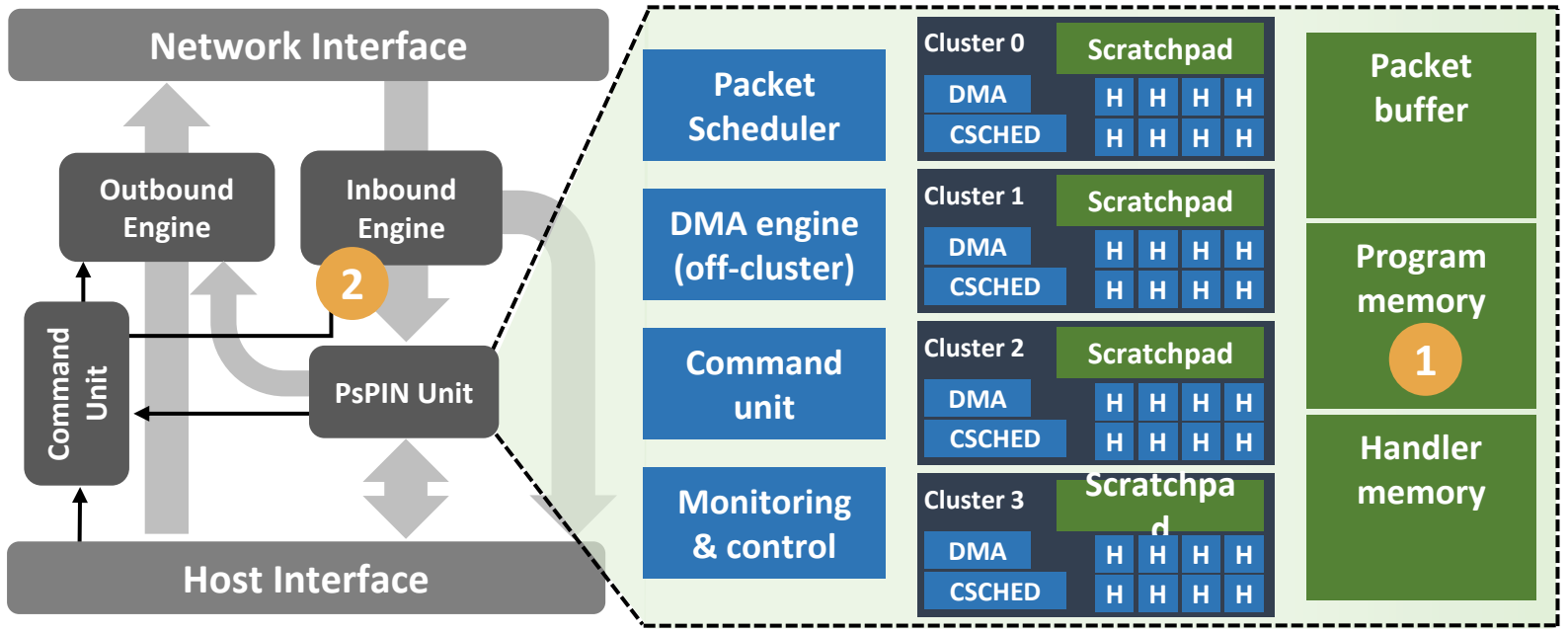
1 Define and offload handlers

Telemetry:
telemetry_hh(), telemetry_ph(), telemetry_th();
Filtering:
filter_hh(), filter_ph, filter_th();

2 Define an execution context

Execution context: EC_filter

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



Application perspective

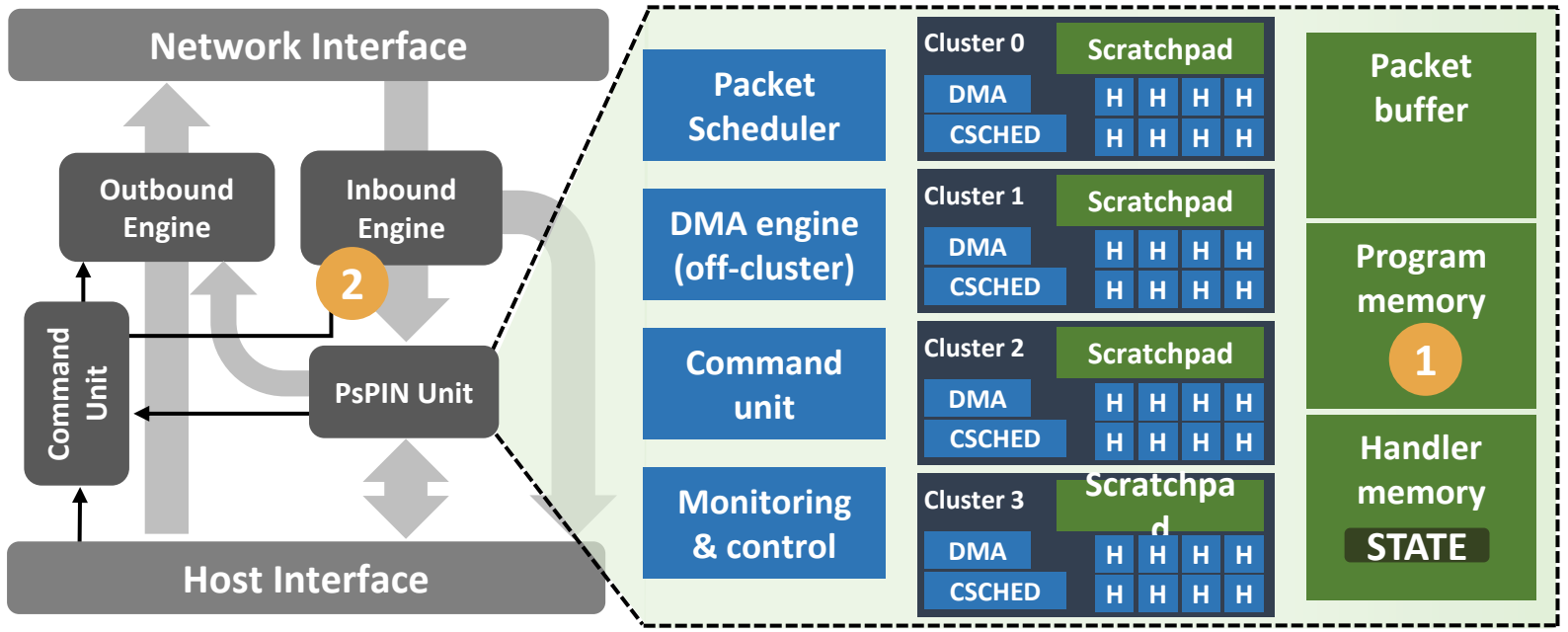
1 Define and offload handlers

Telemetry:
telemetry_hh(), telemetry_ph(), telemetry_th();
Filtering:
filter_hh(), filter_ph, filter_th();

2 Define an execution context

Execution context: EC_filter

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



Application perspective

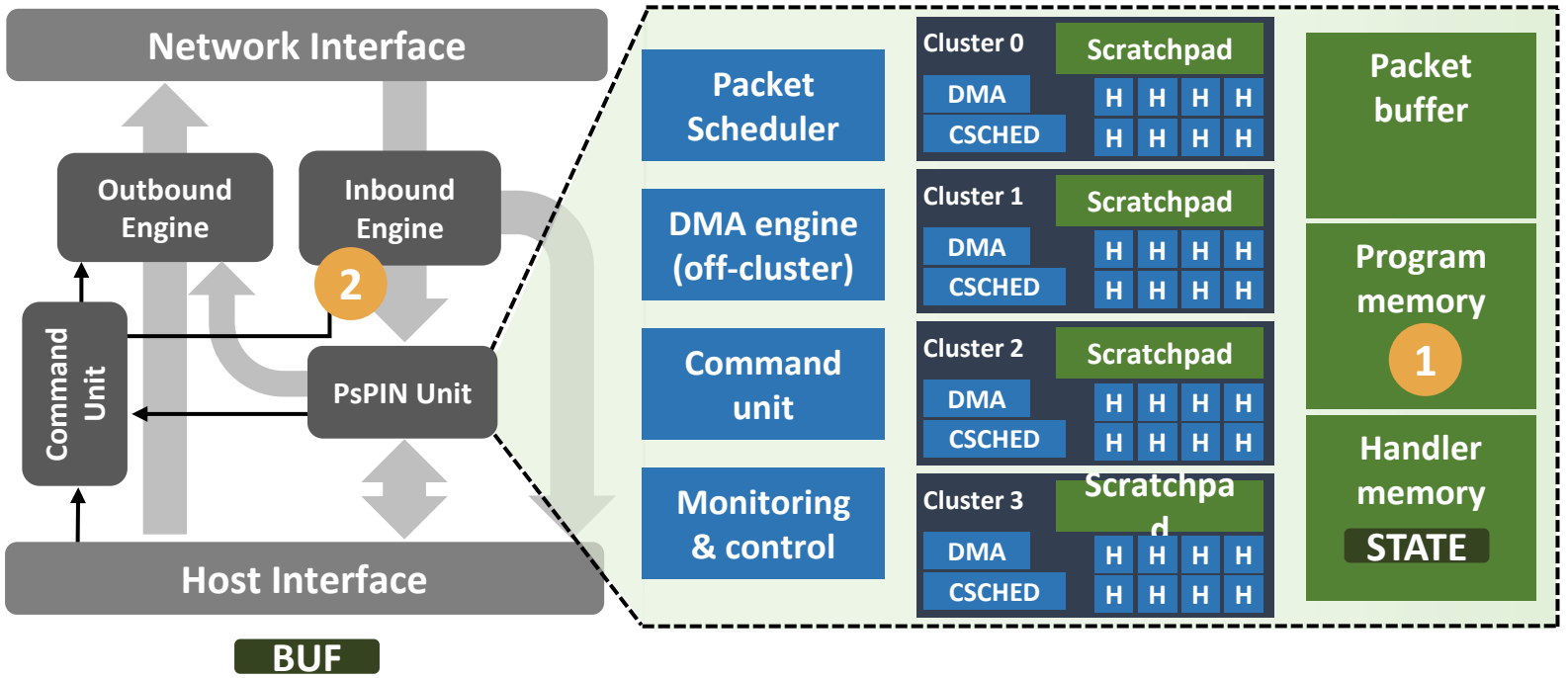
1 Define and offload handlers

Telemetry:
telemetry_hh(), telemetry_ph(), telemetry_th();
Filtering:
filter_hh(), filter_ph, filter_th();

2 Define an execution context

Execution context: EC_filter

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



Application perspective

1. Define and offload handlers

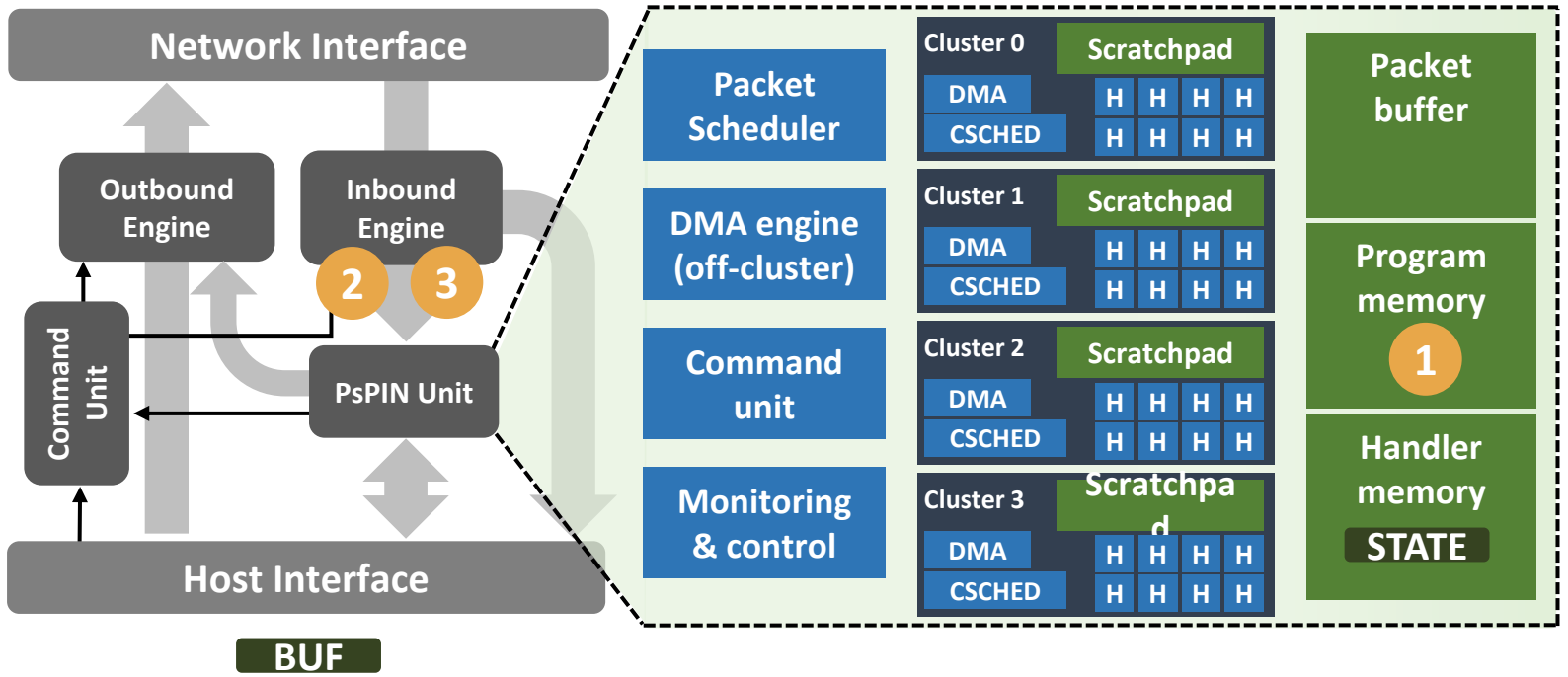
Telemetry:
telemetry_hh(), telemetry_ph(), telemetry_th();
Filtering:
filter_hh(), filter_ph, filter_th();

2. Define an execution context

Execution context: EC_filter

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

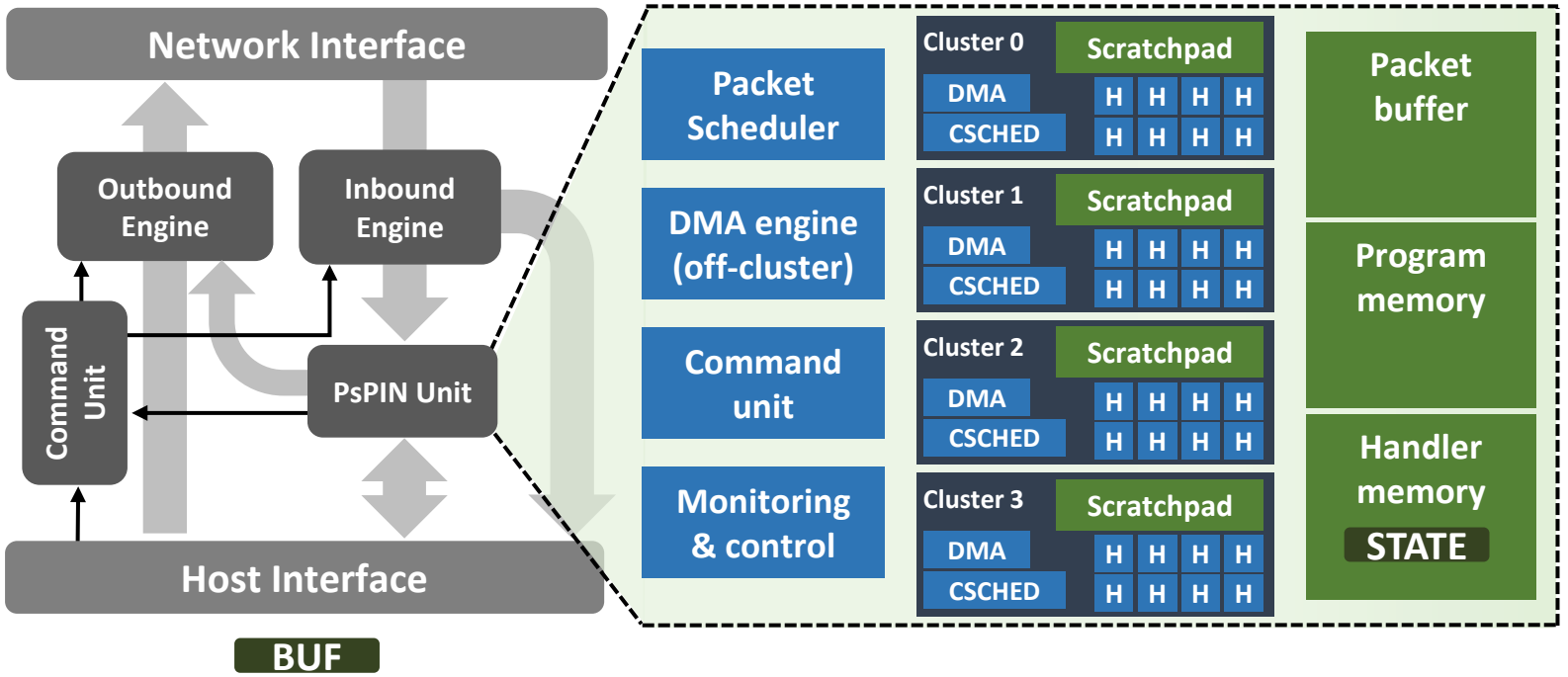
3. Define matching rule:
e.g., (IP packets) -> EC_filter



Network perspective

Execution context: EC_filter

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

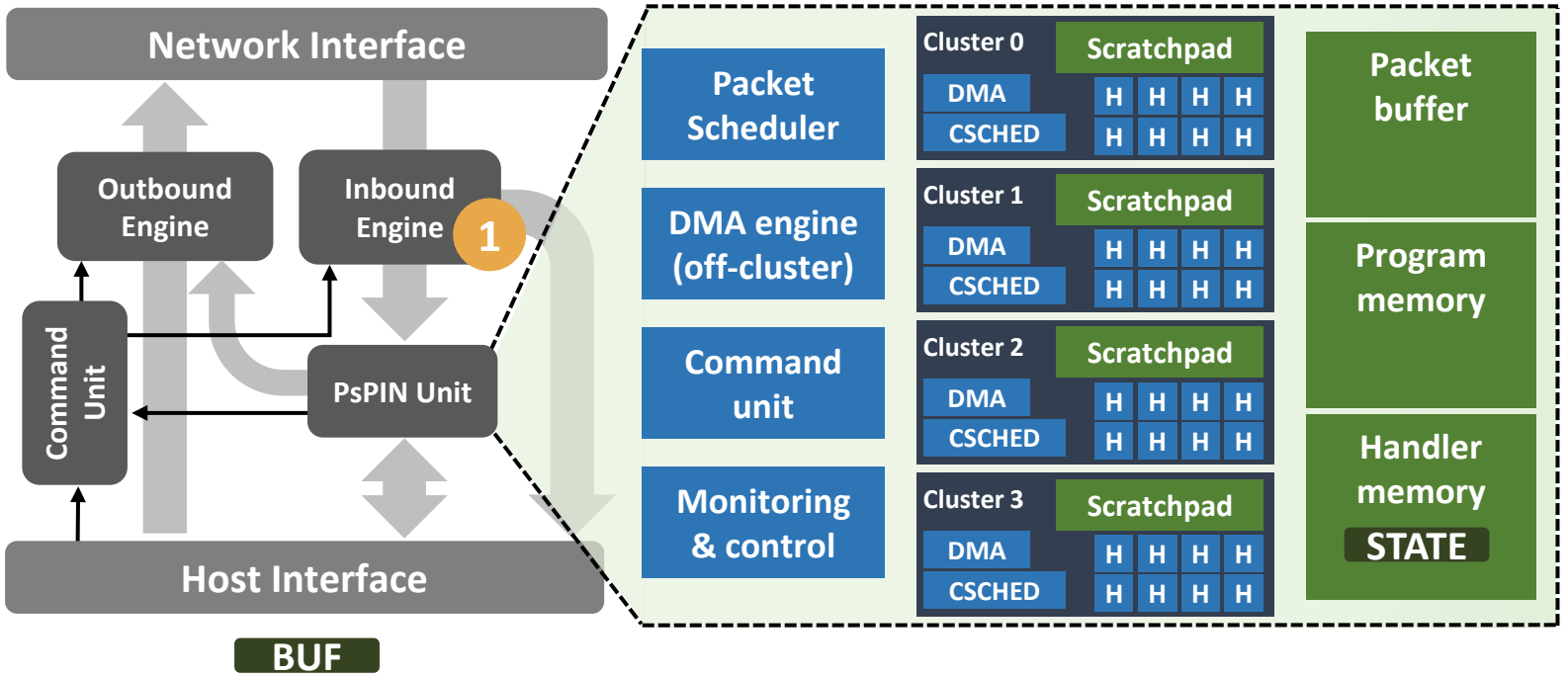


Network perspective

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

Execution context: EC_filter

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



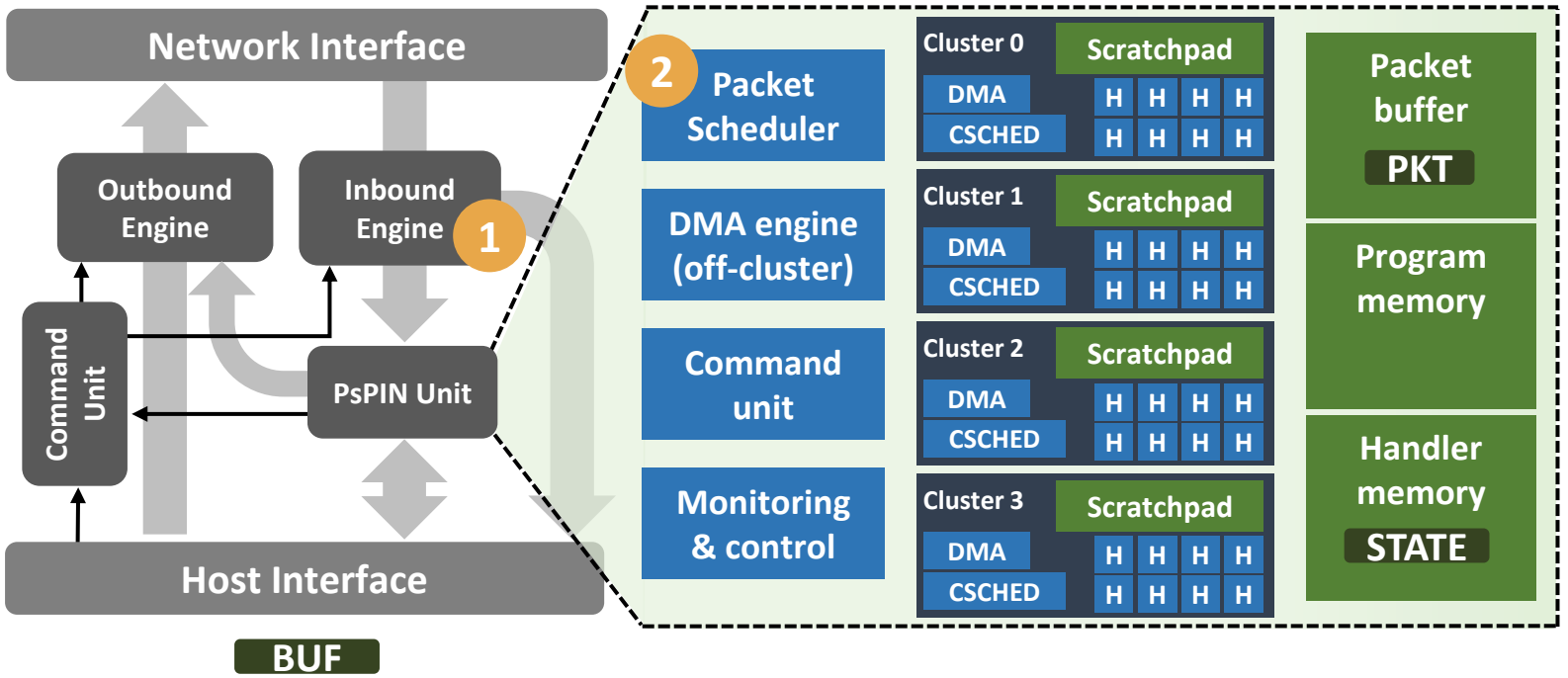
Network perspective

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

Execution context: EC_filter

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



Network perspective

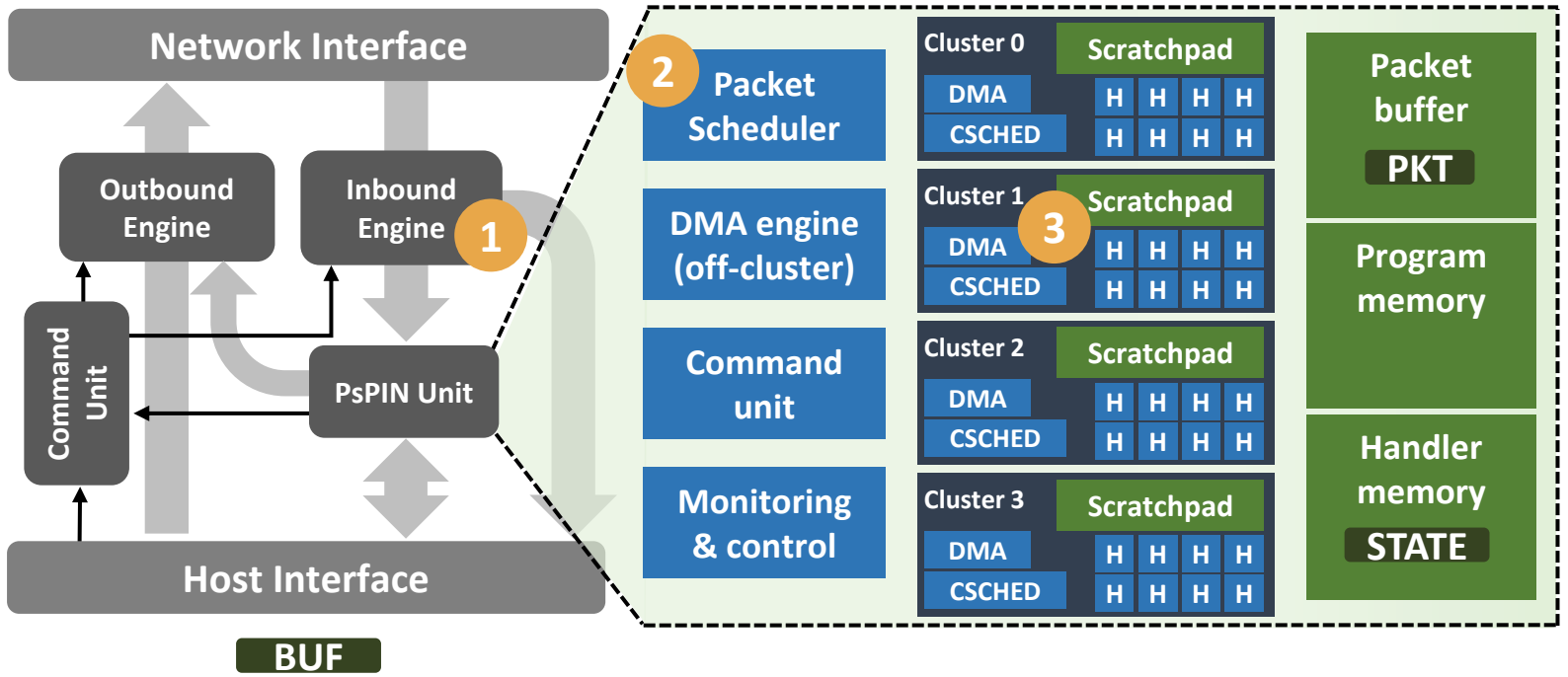
1. Match packet to execution context
   e.g., (IP packets) -> EC_filter

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)

Execution context: EC_filter

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



Network perspective

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

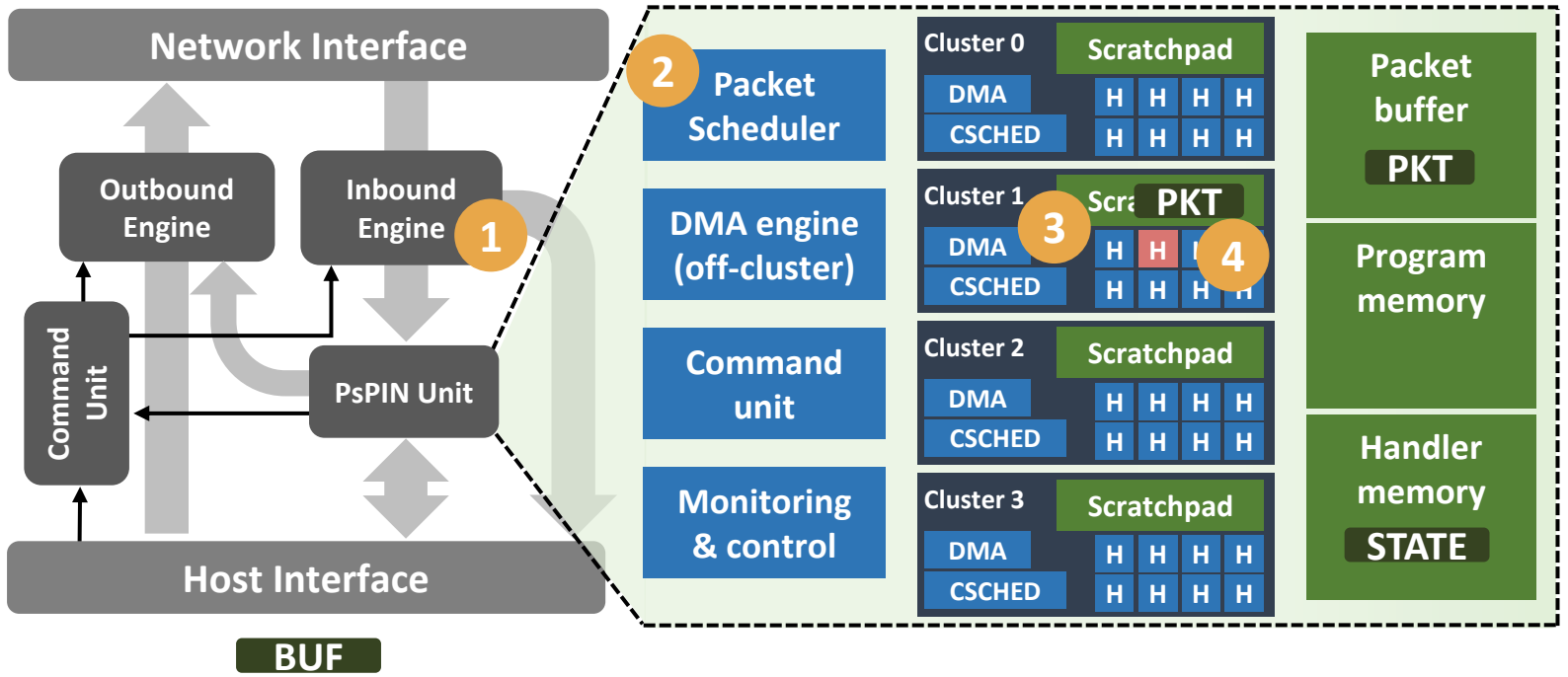
2. Write PKT to L2 pkt buffer and inform PsPIN of the new packet to process

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



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

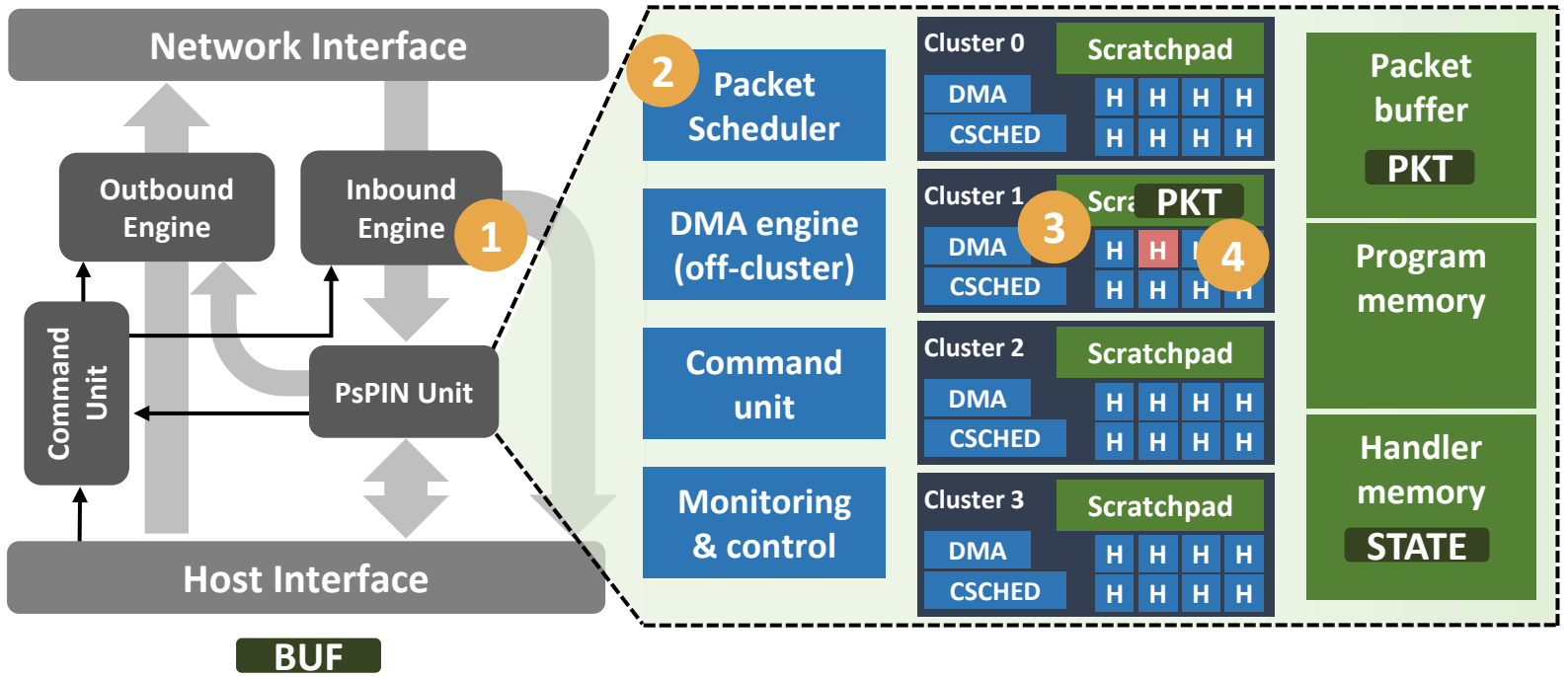
3. Schedule the packet to a cluster (task: pkt pointer, handler fun)

4. Copy packet to L1 and run the handler

Execution context: EC_filter

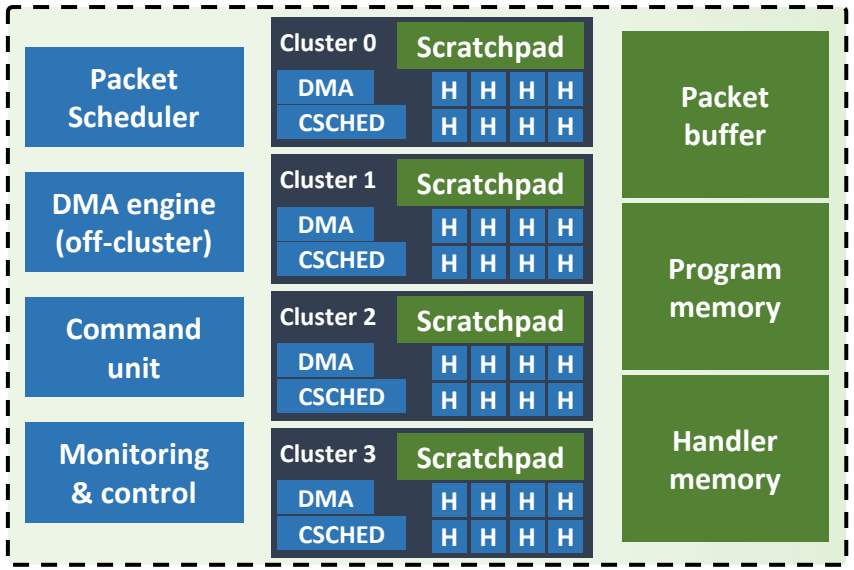
filter_hh(), filter_ph(), filter_th();
NIC memory: STATE
Host buffer: BUF

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



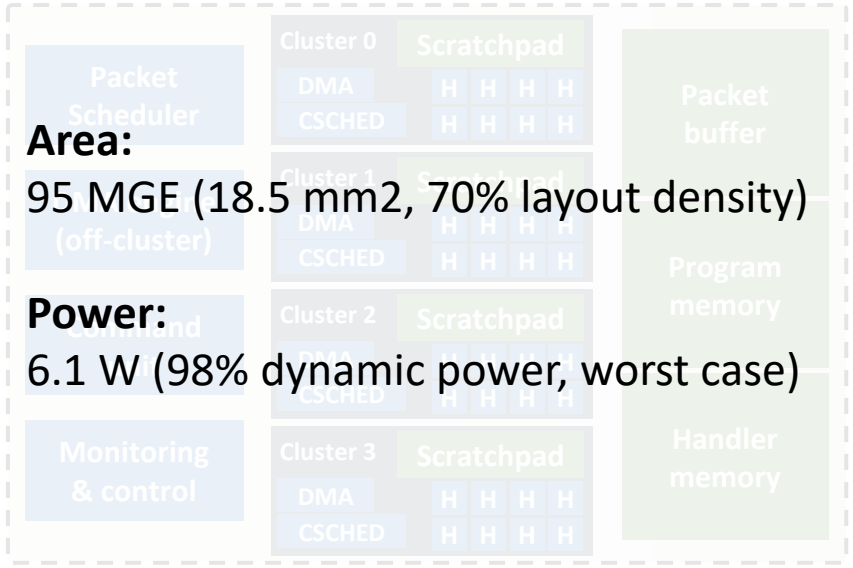
Circuit Complexity, Performance, and Efficiency

GlobalFoundries 22nm FDSOI @ 1GHz



Circuit Complexity, Performance, and Efficiency

GlobalFoundries 22nm FDSOI @ 1GHz



Area:
95 MGE (18.5 mm2, 70% layout density)

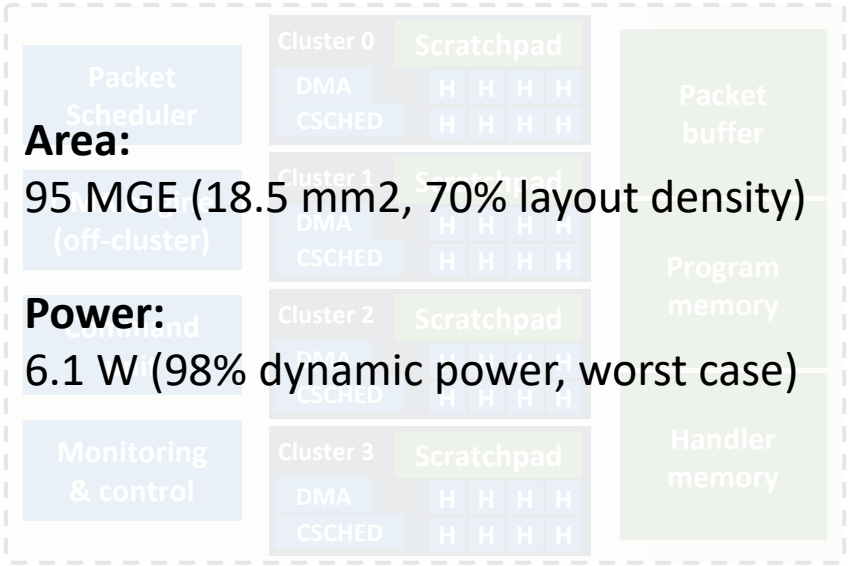
Power:
6.1 W (98% dynamic power, worst case)

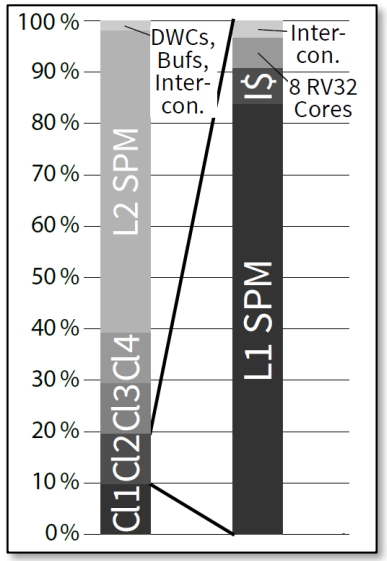
Circuit Complexity, Performance, and Efficiency

GlobalFoundries 22nm FDSOI @ 1GHz

Area:
95 MGE (18.5 mm2, 70% layout density)

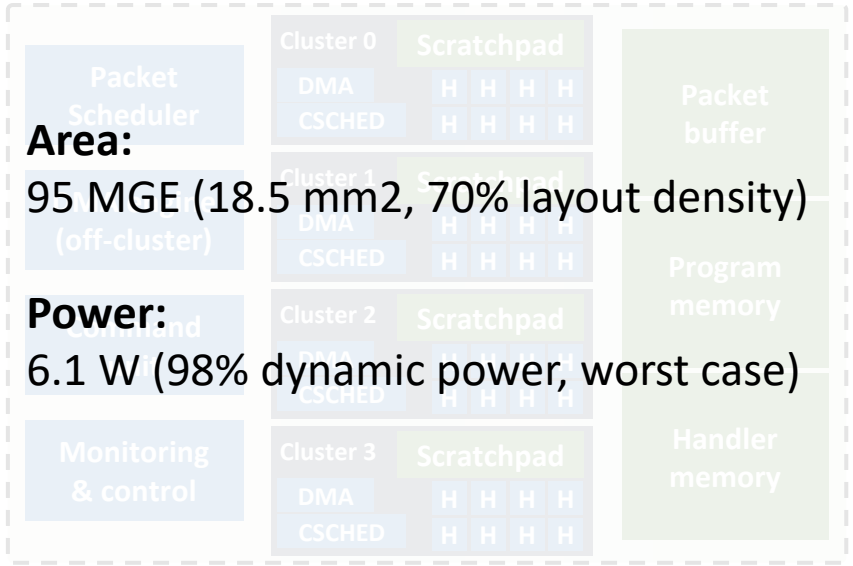
Power:
6.1 W (98% dynamic power, worst case)





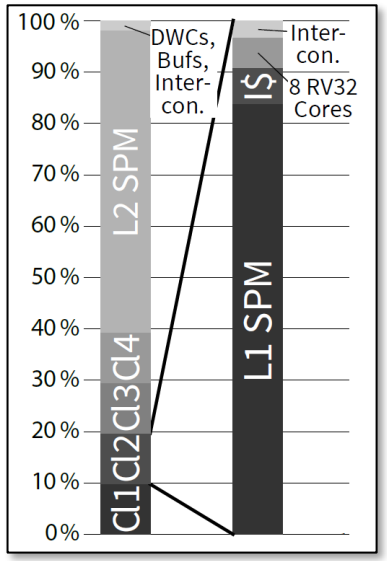
Circuit Complexity, Performance, and Efficiency

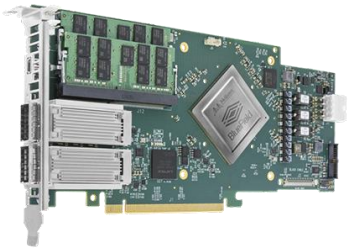
GlobalFoundries 22nm FDSOI @ 1GHz



Area:
95 MGE (18.5 mm2, 70% layout density)

Power:
6.1 W (98% dynamic power, worst case)

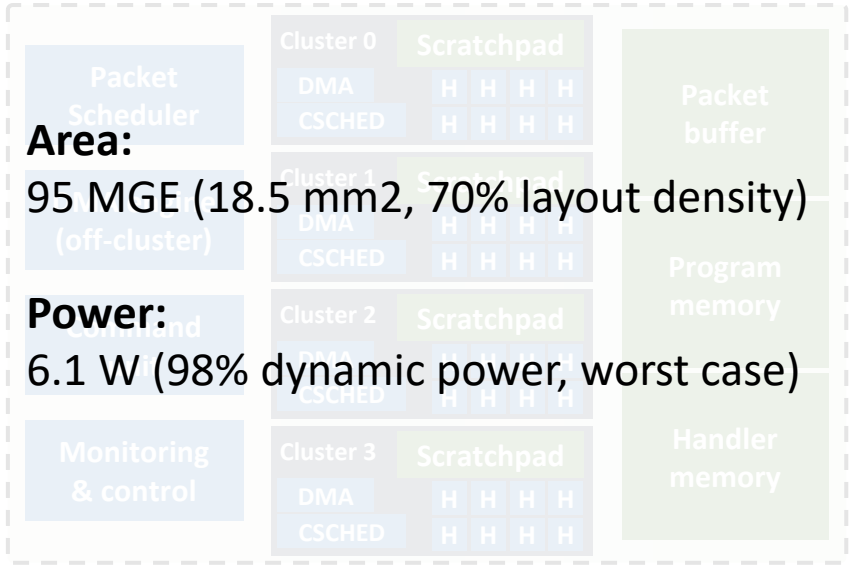




Mellanox BlueField: 16 A72 64bit cores
Estimated area: 51 mm2

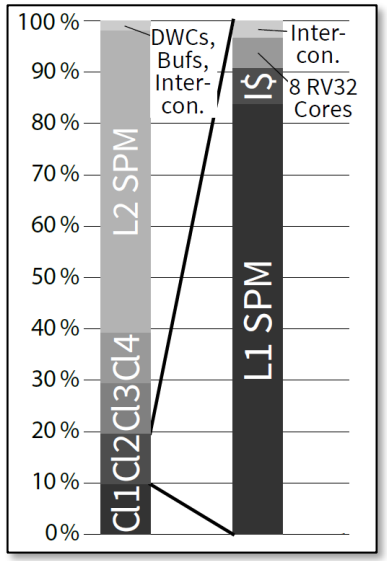
Circuit Complexity, Performance, and Efficiency

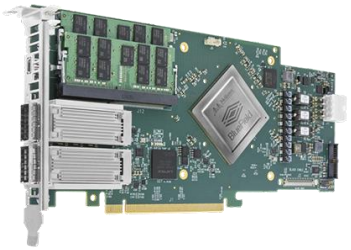
GlobalFoundries 22nm FDSOI @ 1GHz



Area:
95 MGE (18.5 mm2, 70% layout density)

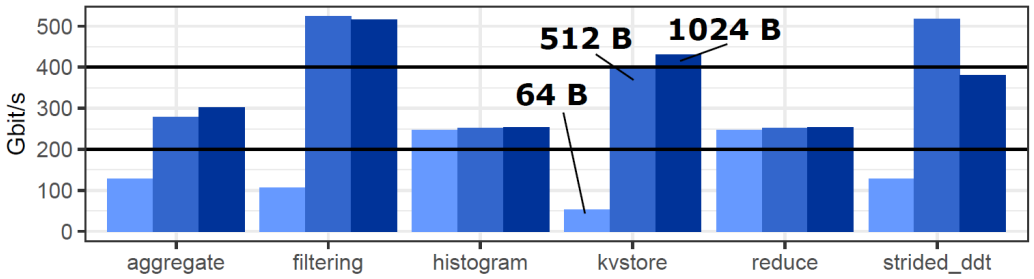
Power:
6.1 W (98% dynamic power, worst case)





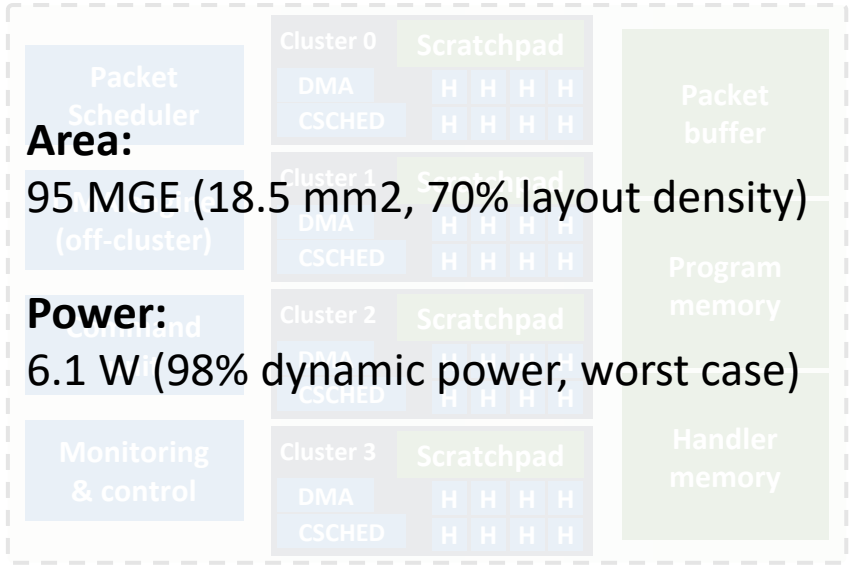
Mellanox BlueField: 16 A72 64bit cores
Estimated area: 51 mm2

Cycle-accurate simulations



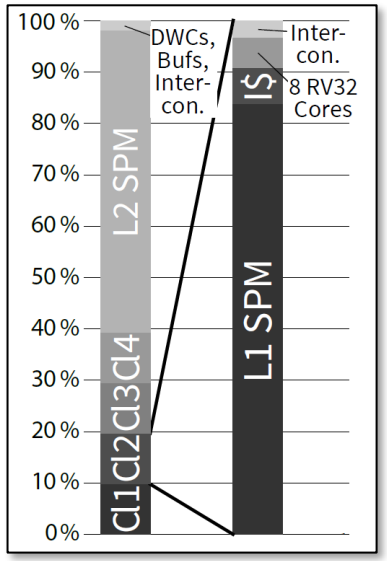
Circuit Complexity, Performance, and Efficiency

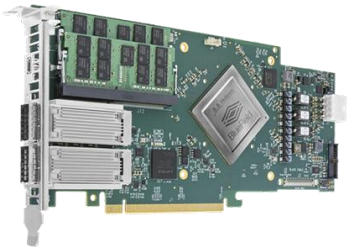
GlobalFoundries 22nm FDSOI @ 1GHz



Area:
95 MGE (18.5 mm2, 70% layout density)

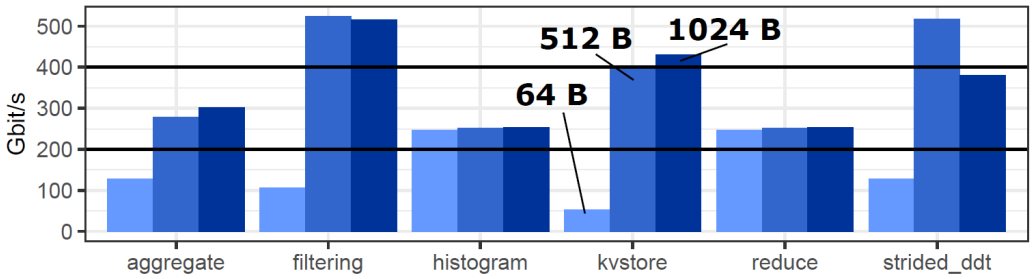
Power:
6.1 W (98% dynamic power, worst case)





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

Cycle-accurate simulations



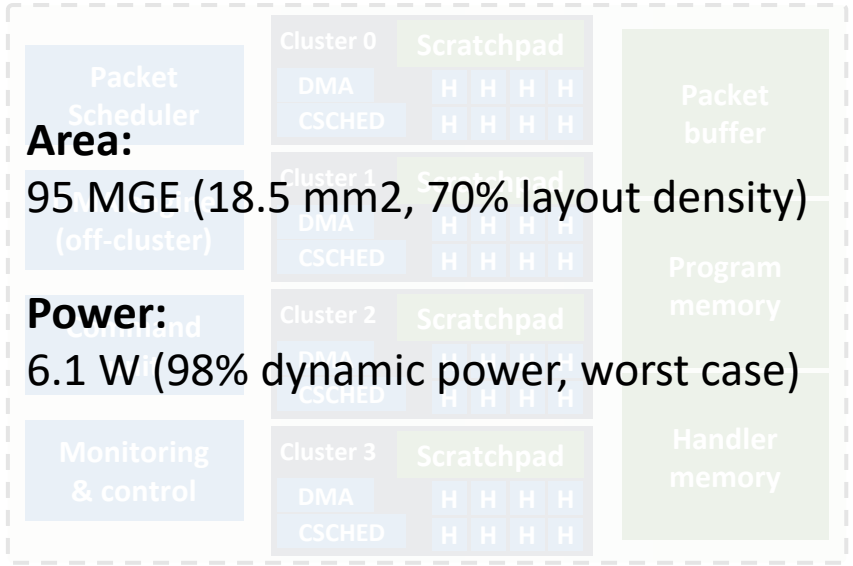
zynq

> 10x area efficiency (Gb/s/area)

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

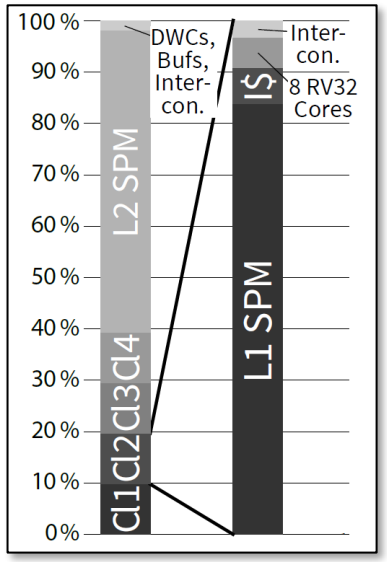
Circuit Complexity, Performance, and Efficiency

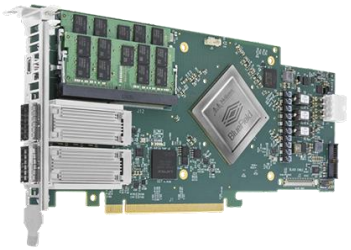
GlobalFoundries 22nm FDSOI @ 1GHz



Area:
95 MGE (18.5 mm2, 70% layout density)

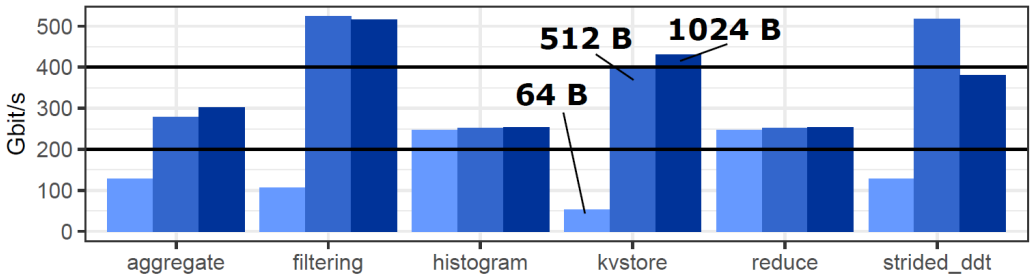
Power:
6.1 W (98% dynamic power, worst case)





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

Cycle-accurate simulations



zynq

> 10x area efficiency (Gb/s/area)

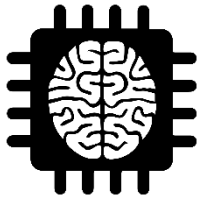
ARM Cortex-A53 @ 1.2 GHz
(4 cores, 2-way superscalar, 64-bit)

ault @ CSCS

> 100x area efficiency (Gb/s/area)

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

Network-accelerated datatypes

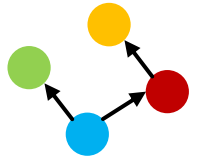


Quantization

Erasure coding

Distributed File Systems

Zoo-sPINNER
consensus on sPIN

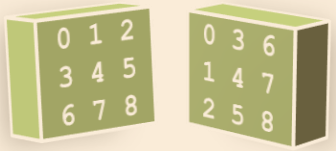


Network Group Communication

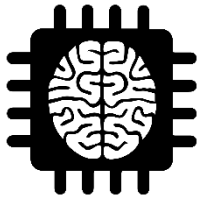
Packet classification and pattern matching

In-network allreduce

Serverless sPIN



Network-accelerated datatypes

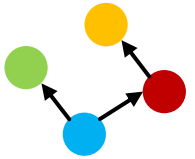


Quantization

Erasure coding

Distributed File Systems

Zoo-sPINNER
consensus on sPIN

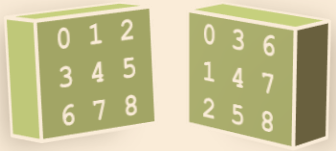


Network Group Communication

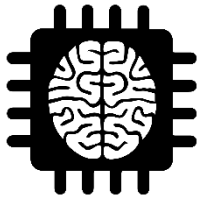
Packet classification and pattern matching

In-network allreduce

Serverless sPIN



Network-accelerated datatypes

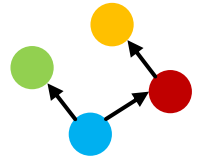


Quantization

Erasure coding

Distributed File Systems

Zoo-sPINNER
consensus on sPIN



Network Group Communication

Packet classification and pattern matching

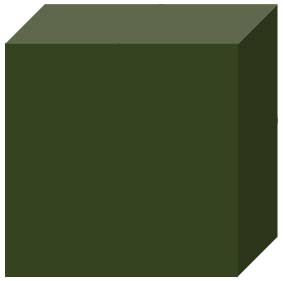
In-network allreduce

Serverless sPIN

Network-accelerated non-contiguous memory transfers

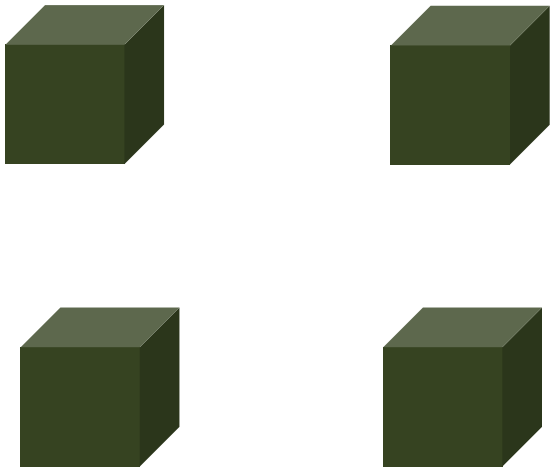
Salvatore Di Girolamo, Konstantin Taranov, Andreas Kurth, Michael Schaffner, Timo Schneider, Jakub Beránek, Maciej Besta, Luca Benini, Duncan Roweth, and Torsten Hoefler. "Network-accelerated non-contiguous memory transfers." SC ‘19

Network-accelerated non-contiguous memory transfers



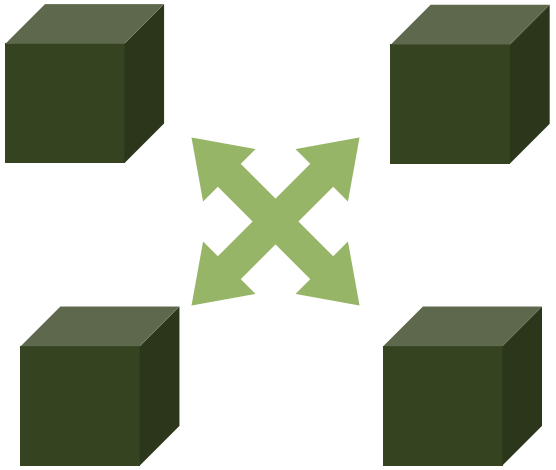
Salvatore Di Girolamo, Konstantin Taranov, Andreas Kurth, Michael Schaffner, Timo Schneider, Jakub Beránek, Maciej Besta, Luca Benini, Duncan Roweth, and Torsten Hoefler. "Network-accelerated non-contiguous memory transfers." SC ‘19

Network-accelerated non-contiguous memory transfers



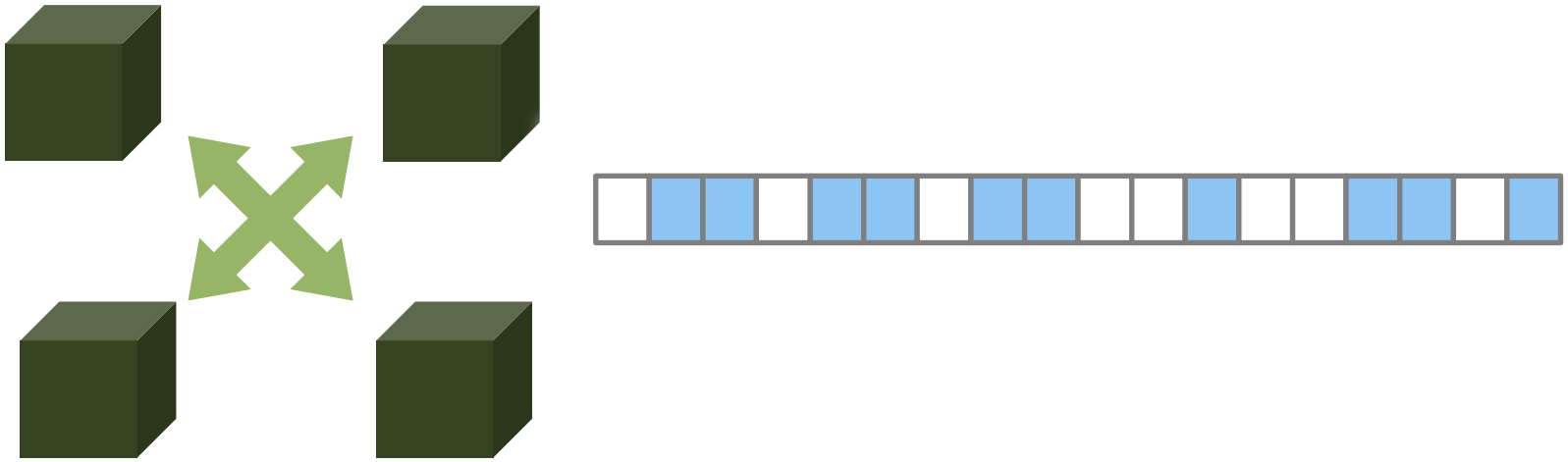
Salvatore Di Girolamo, Konstantin Taranov, Andreas Kurth, Michael Schaffner, Timo Schneider, Jakub Beránek, Maciej Besta, Luca Benini, Duncan Roweth, and Torsten Hoefler. "Network-accelerated non-contiguous memory transfers." SC ‘19

Network-accelerated non-contiguous memory transfers



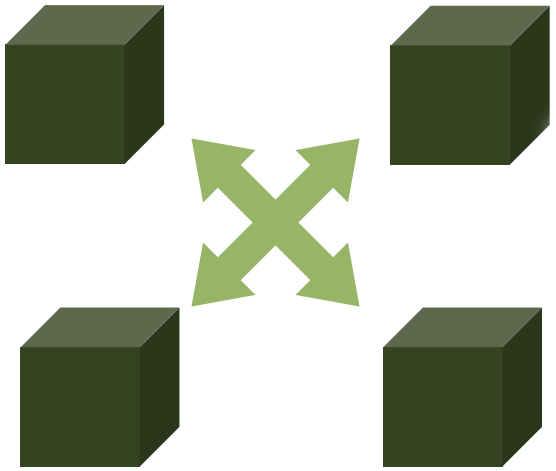
Salvatore Di Girolamo, Konstantin Taranov, Andreas Kurth, Michael Schaffner, Timo Schneider, Jakub Beránek, Maciej Besta, Luca Benini, Duncan Roweth, and Torsten Hoefler. "Network-accelerated non-contiguous memory transfers." SC ‘19

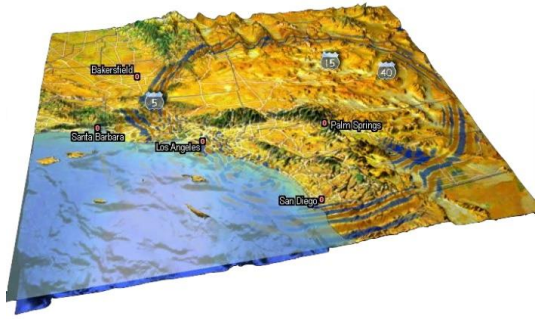
Network-accelerated non-contiguous memory transfers



Salvatore Di Girolamo, Konstantin Taranov, Andreas Kurth, Michael Schaffner, Timo Schneider, Jakub Beránek, Maciej Besta, Luca Benini, Duncan Roweth, and Torsten Hoefler. "Network-accelerated non-contiguous memory transfers." SC ‘19

Network-accelerated non-contiguous memory transfers

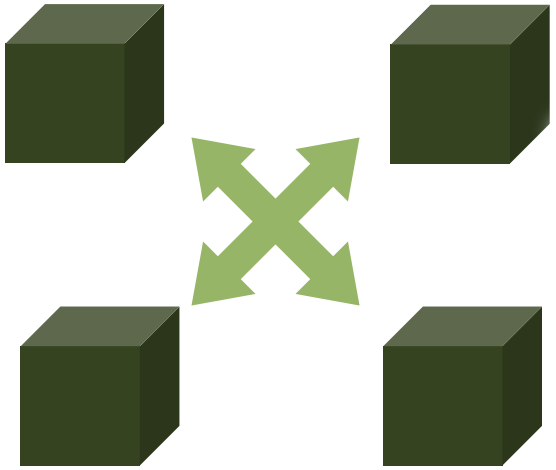


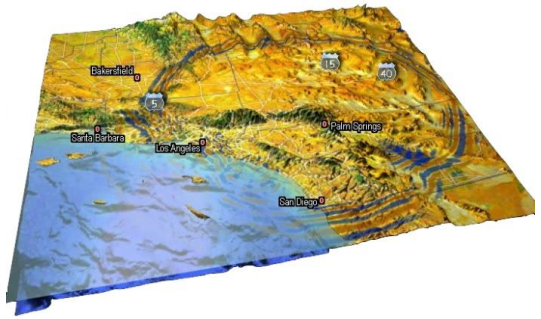


https://specfem3d.readthedocs.io/en/latest/

L. Carrington et al. High-frequency simulations of global seismic wave propagation using SPECFEM3D_GLOBE on 62K processors. SC 2008.

Network-accelerated non-contiguous memory transfers





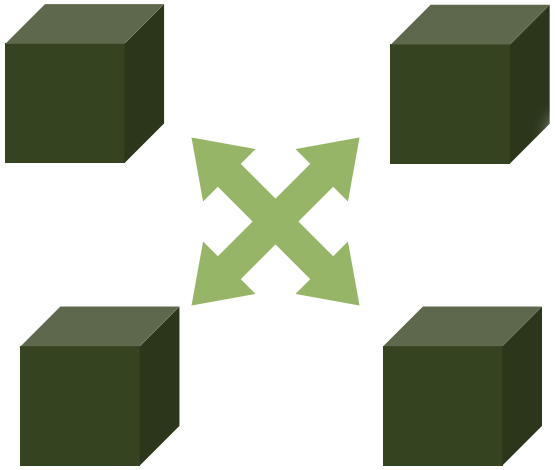
https://specfem3d.readthedocs.io/en/latest/

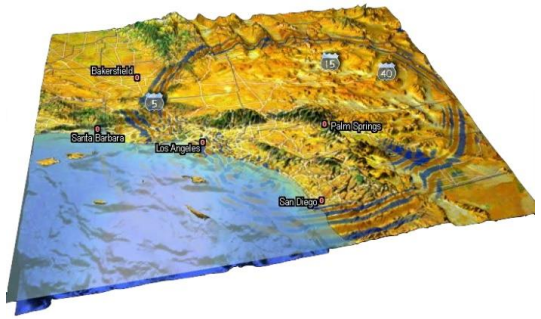
L. Carrington et al. High-frequency simulations of global seismic wave propagation using SPECFEM3D_GLOBE on 62K processors. SC 2008.

http://fourier.eng.hmc.edu/e161/lectures/fourier/node10.html

T. Hoefler et al. Parallel zero-copy algorithms for fast Fourier transform and conjugate gradient using MPI datatypes. EuroMPI 2010.

Network-accelerated non-contiguous memory transfers



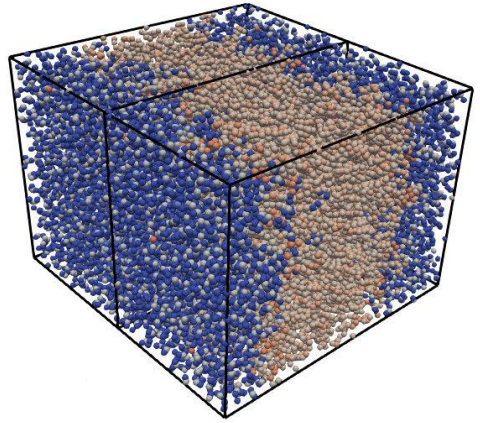


https://specfem3d.readthedocs.io/en/latest/

L. Carrington et al. High-frequency simulations of global seismic wave propagation using SPECFEM3D_GLOBE on 62K processors. SC 2008.

http://fourier.eng.hmc.edu/e161/lectures/fourier/node10.html

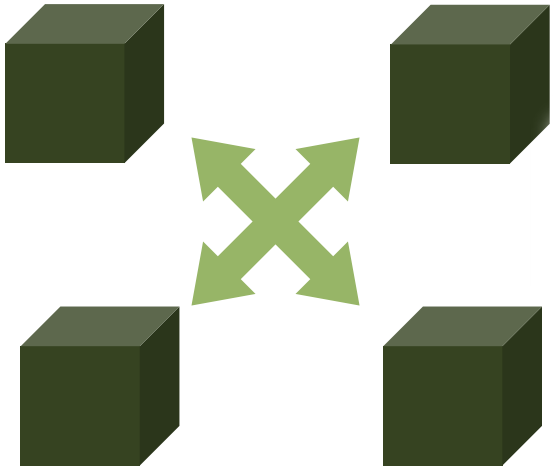
T. Hoefler et al. Parallel zero-copy algorithms for fast Fourier transform and conjugate gradient using MPI datatypes. EuroMPI 2010.



W. Usher et al. libIS: a lightweight library for flexible in transit visualization. ISAV 2018.

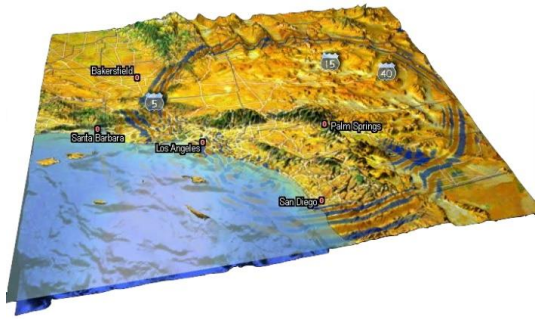
Salvatore Di Girolamo, Konstantin Taranov, Andreas Kurth, Michael Schaffner, Timo Schneider, Jakub Beránek, Maciej Besta, Luca Benini, Duncan Roweth, and Torsten Hoefler. "Network-accelerated non-contiguous memory transfers." SC '19

Network-accelerated non-contiguous memory transfers



MPI

Derived Datatypes

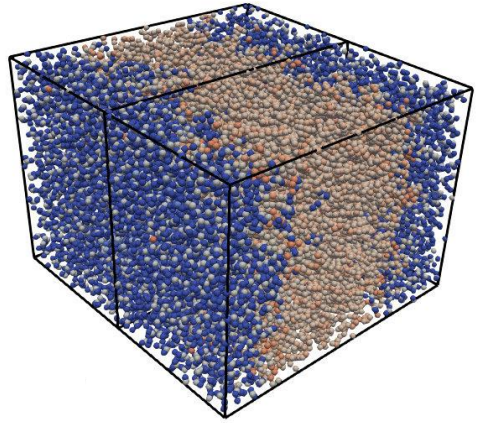


https://specfem3d.readthedocs.io/en/latest/

L. Carrington et al. High-frequency simulations of global seismic wave propagation using SPECFEM3D_GLOBE on 62K processors. SC 2008.

http://fourier.eng.hmc.edu/e161/lectures/fourier/node10.html

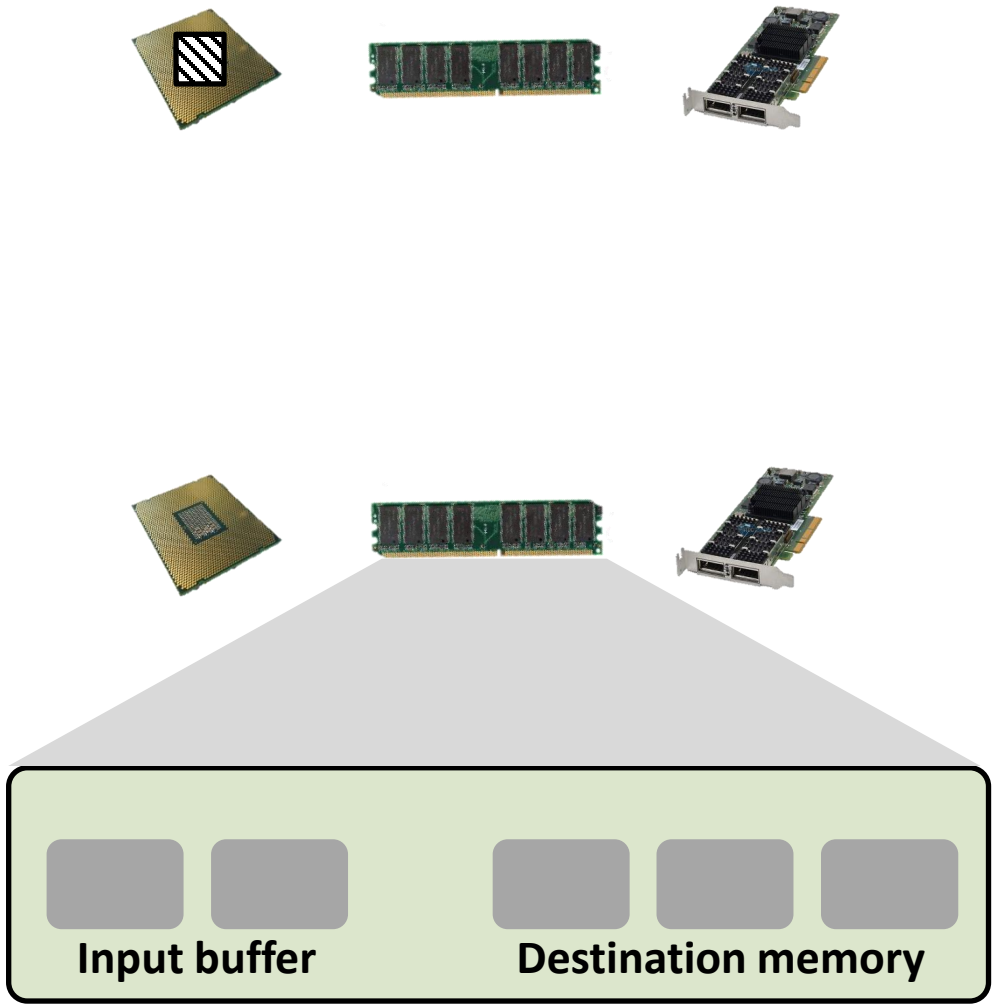
T. Hoefler et al. Parallel zero-copy algorithms for fast Fourier transform and conjugate gradient using MPI datatypes. EuroMPI 2010.



W. Usher et al. libIS: a lightweight library for flexible in transit visualization. ISAV 2018.

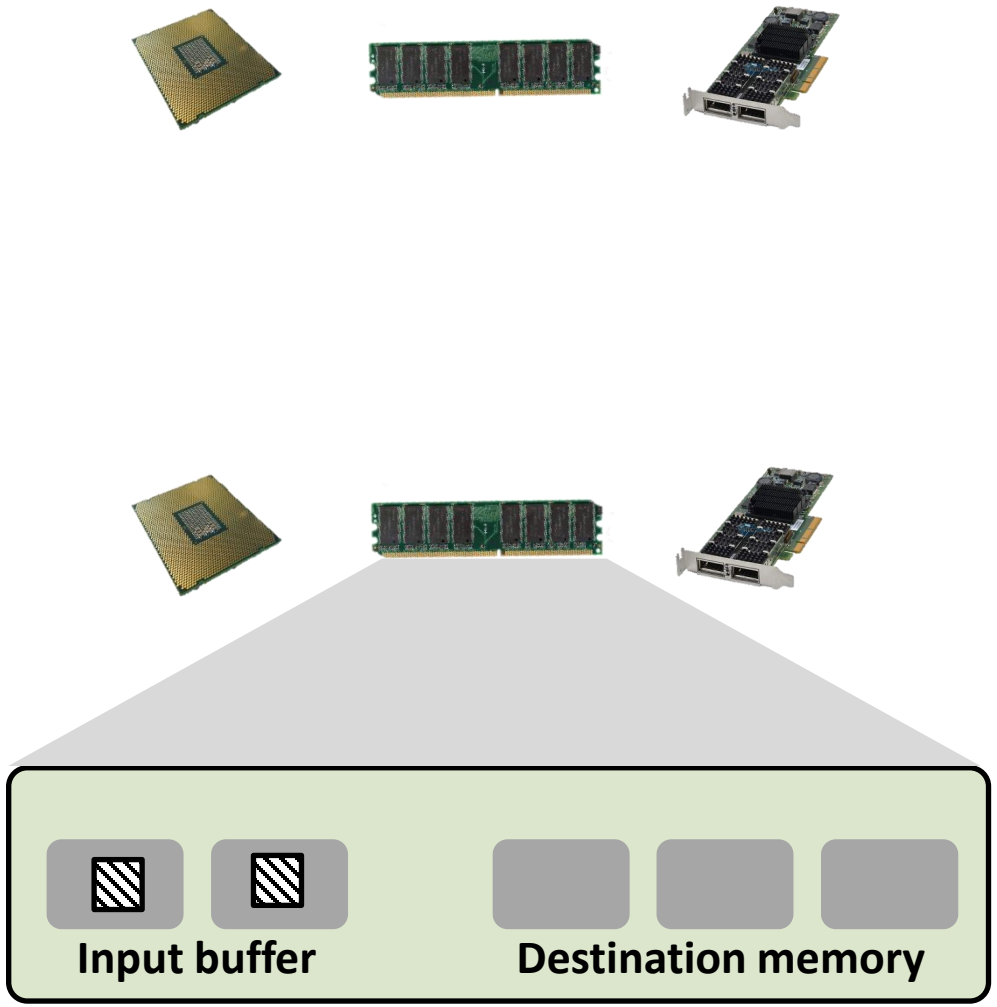
Salvatore Di Girolamo, Konstantin Taranov, Andreas Kurth, Michael Schaffner, Timo Schneider, Jakub Beránek, Maciej Besta, Luca Benini, Duncan Roweth, and Torsten Hoefler. "Network-accelerated non-contiguous memory transfers." SC '19

MPI Datatypes Processing



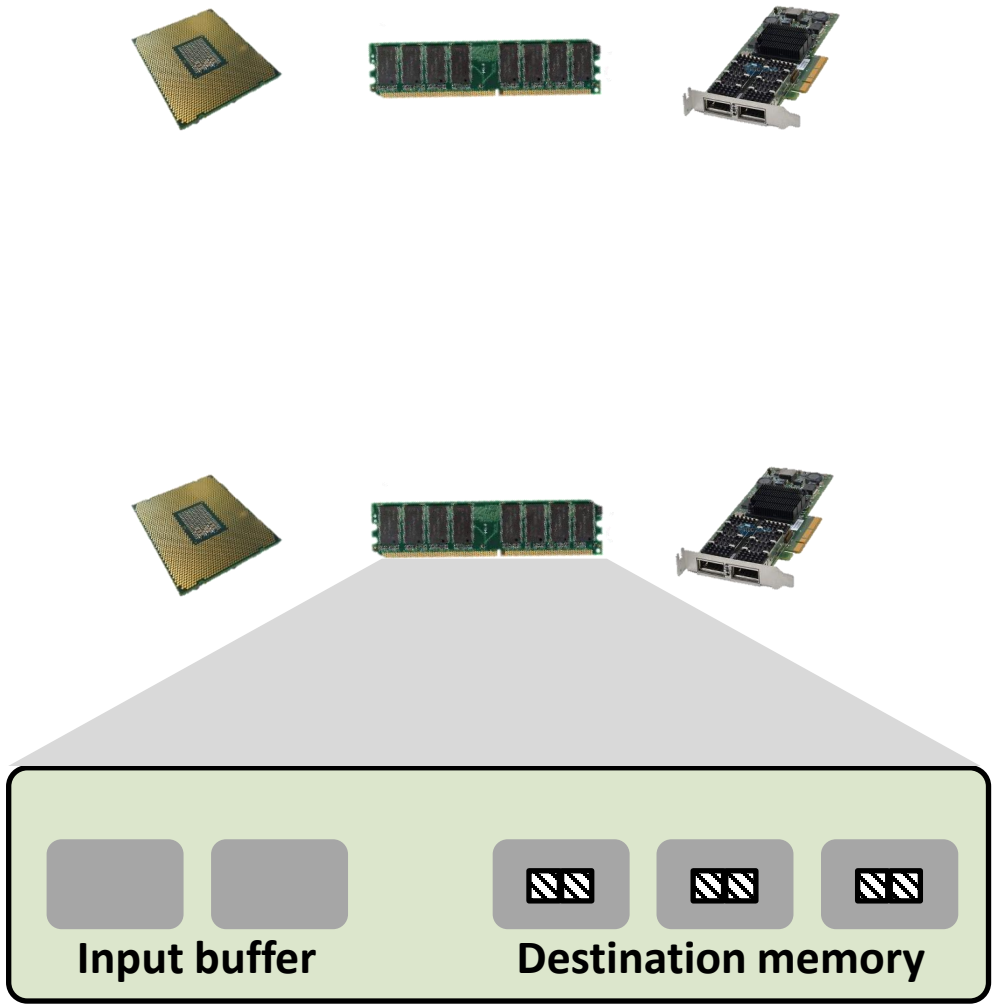
Gropp, W., et al., March. Improving the performance of MPI derived datatypes. *MPIDC'99*
Torsten Hoefler, Salvatore Di Girolamo, Konstantin Taranov, Ryan E. Grant, and Ron Brightwell. 2017. sPIN: High-performance streaming Processing In the Network. SC'17

MPI Datatypes Processing



Gropp, W., et al., March. Improving the performance of MPI derived datatypes. *MPIDC'99*
Torsten Hoefler, Salvatore Di Girolamo, Konstantin Taranov, Ryan E. Grant, and Ron Brightwell. 2017. sPIN: High-performance streaming Processing In the Network. SC'17

MPI Datatypes Processing

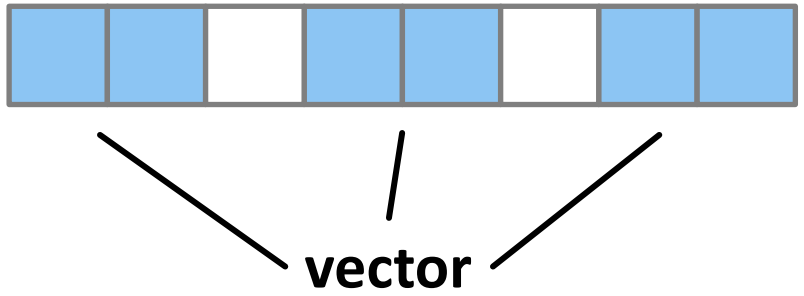


Gropp, W., et al., March. Improving the performance of MPI derived datatypes. *MPIDC'99*

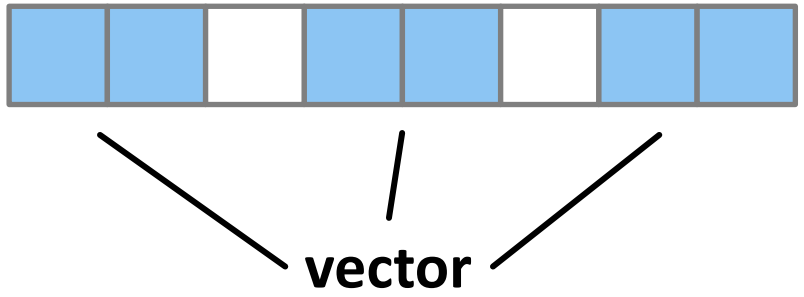
Torsten Hoefler, Salvatore Di Girolamo, Konstantin Taranov, Ryan E. Grant, and Ron Brightwell. 2017. sPIN: High-performance streaming Processing In the Network. SC'17

Specialized Handlers

Specialized Handlers

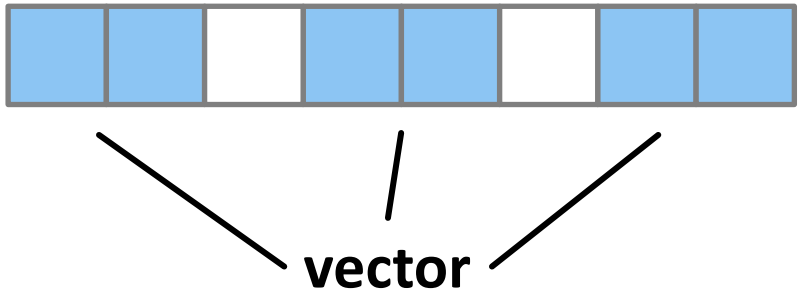


Specialized Handlers



spin_vec_t:
num_blocks: 3
block_size: 2
stride: 3
base_type: int

Specialized Handlers



spin_vec_t:
num_blocks: 3
block_size: 2
stride: 3
base_type: int

Specialized Handlers

NIC Memory

vector

spin_vec_t:
num_blocks: 3
block_size: 2
stride: 3
base_type: int

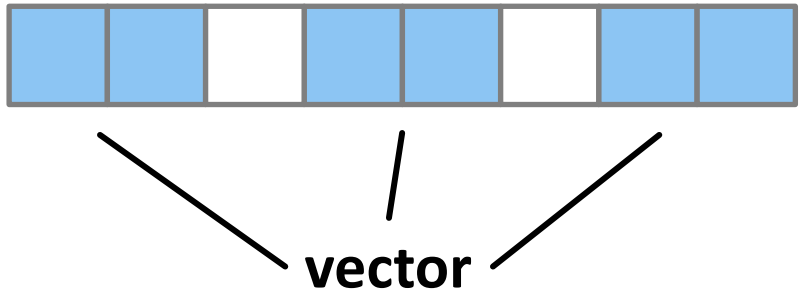
```
__handler vector_payload_handler(handler_args_t *args)
{
    spin_vec_t *ddt_descr = (spin_vec_t *)args->mem;
    uint32_t num_blocks = args->packet_len / ddt_descr->block_size;
    uint32_t stride = ddt_descr->stride;

    uint8_t *pkt_payload = args->pkt_payload_ptr;

    uint8_t *host_base_ptr = args->host_address;
    uint32_t host_offset = (ddt_descr->pkt_offset / ddt_descr->block_size) * stride;
    uint8_t *host_address = host_base_ptr + host_offset;

    for (uint32_t i=0; i<num_blocks; i++)
    {
        PtlHandlerDMAToHostNB(host_address, pkt_payload, block_size, DMA_NO_EVENT);
        pkt_payload += block_size;
        host_address += stride;
    }

    return SPIN_SUCCESS;
}
```

Handler

Specialized Handlers

NIC Memory

spin_vec_t:
num_blocks: 3
block_size: 2
stride: 3
base_type: int




```
__handler vector_payload_handler(handler_args_t *args)
{
    spin_vec_t *ddt_descr = (spin_vec_t *)args->mem;
    uint32_t num_blocks = args->packet_len / ddt_descr->block_size;
    uint32_t stride = ddt_descr->stride;

    uint8_t *pkt_payload = args->pkt_payload_ptr;

    uint8_t *host_base_ptr = args->host_address;
    uint32_t host_offset = ddt_descr->pkt_offset / (ddt_descr->block_size) * stride;
    uint8_t *host_address = host_base_ptr + host_offset;

    for (uint32_t i=0; i<num_blocks; i++)
    {
        PtlHandlerDMAToHostNB(host_address, pkt_payload, block_size, DMA_NO_EVENT);
        pkt_payload += block_size;
        host_address += stride;
    }

    return SPIN_SUCCESS;
}
```

Handler

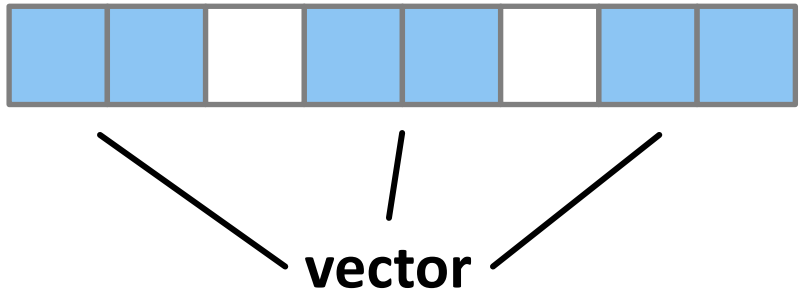
Specialized Handlers

NIC Memory

spin_vec_t:
num_blocks: 3
block_size: 2
stride: 3
base_type: int

```
__handler vector_payload_handler(handler_args_t *args)
{
    spin_vec_t *ddt_descr = (spin_vec_t *)args->mem;
    uint32_t num_blocks = args->packet_len / ddt_descr->block_size;
    uint32_t stride = ddt_descr->stride;

    uint8_t *pkt_payload = args->pkt_payload_ptr;

    uint8_t *host_base_ptr = args->host_address;
    uint32_t host_offset = (args->pkt_offset / ddt_descr->block_size) * stride;
    uint8_t *host_address = host_base_ptr + host_offset;

    for (uint32_t i=0; i<num_blocks; i++)
    {
        PtlHandlerDMAToHostNB(host_address, pkt_payload, block_size, DMA_NO_EVENT);
        pkt_payload += block_size;
        host_address += stride;
    }

    return SPIN_SUCCESS;
}
```

Handler



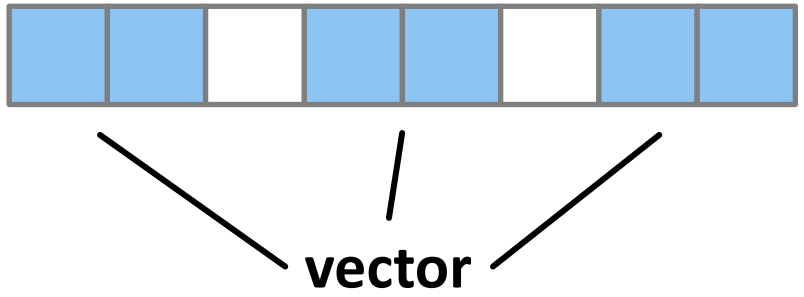
Specialized Handlers

NIC Memory

spin_vec_t:
num_blocks: 3
block_size: 2
stride: 3
base_type: int

```
__handler vector_payload_handler(handler_args_t *args)
{
    spin_vec_t *ddt_descr = (spin_vec_t *)args->mem;
    uint32_t num_blocks = args->packet_len / ddt_descr->block_size;
    uint32_t stride = ddt_descr->stride;

    uint8_t *pkt_payload = args->pkt_payload_ptr;

    uint8_t *host_base_ptr = args->host_address;
    uint32_t host_offset = (args->pkt_offset / ddt_descr->block_size) * stride;
    uint8_t *host_address = host_base_ptr + host_offset;

    for (uint32_t i=0; i<num_blocks; i++)
    {
        PtlHandlerDMAToHostNB(host_address, pkt_payload, block_size, DMA_NO_EVENT);
        pkt_payload += block_size;
        host_address += stride;
    }

    return SPIN_SUCCESS;
}
```

Handler



Specialized Handlers

NIC Memory

spin_vec_t:
num_blocks: 3
block_size: 2
stride: 3
base_type: int

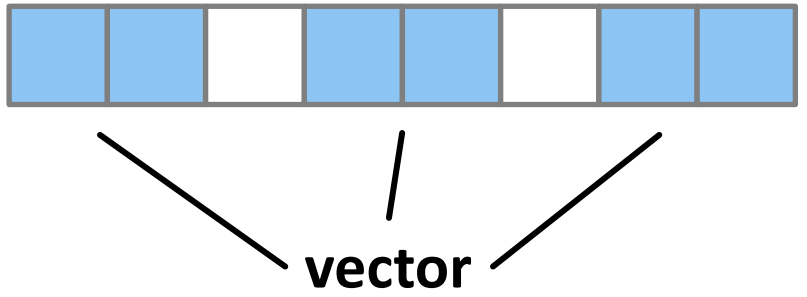
vector

```
__handler vector_payload_handler(handler_args_t *args)
{
    spin_vec_t *ddt_descr = (spin_vec_t *)args->mem;
    uint32_t num_blocks = args->packet_len / ddt_descr->block_size;
    uint32_t stride = ddt_descr->stride;

    uint8_t *pkt_payload = args->pkt_payload_ptr;

    uint8_t *host_base_ptr = args->host_address;
    uint32_t host_offset = (args->pkt_offset / ddt_descr->block_size;) * stride;
    uint8_t *host_address = host_base_addr + host_offset;

    for (uint32_t i=0; i<num_blocks; i++)
    {
        PtlHandlerDMAToHostNB(host_address, pkt_payload, block_size, DMA_NO_EVENT);
        pkt_payload += block_size;
        host_address += stride;
    }

    return SPIN_SUCCESS;
}
```

Specialized Handlers

NIC Memory

spin_vec_t:
num_blocks: 3
block_size: 2
stride: 3
base_type: int




```
__handler vector_payload_handler(handler_args_t *args)
{
    spin_vec_t *ddt_descr = (spin_vec_t *)args->mem;
    uint32_t num_blocks = args->packet_len / ddt_descr->block_size;
    uint32_t stride = ddt_descr->stride;

    uint8_t *pkt_payload = args->pkt_payload_ptr;

    uint8_t *host_base_ptr = args->host_address;
    uint32_t host_offset = (args->pkt_offset / ddt_descr->block_size;) * stride;
    uint8_t *host_address = host_base_addr + host_offset;

    for (uint32_t i=0; i<num_blocks; i++)
    {
        PtlHandlerDMAToHostNB(host_address, pkt_payload, block_size, DMA_NO_EVENT);
        pkt_payload += block_size;
        host_address += stride;
    }

    return SPIN_SUCCESS;
}
```

Load DDT info

Specialized Handlers

NIC Memory

spin_vec_t:
num_blocks: 3
block_size: 2
stride: 3
base_type: int

vector

```
__handler vector_payload_handler(handler_args_t *args)
{
    spin_vec_t *ddt_descr = (spin_vec_t *)args->mem;
    uint32_t num_blocks = args->packet_len / ddt_descr->block_size;
    uint32_t stride = ddt_descr->stride;

    uint8_t *pkt_payload = args->pkt_payload_ptr;

    uint8_t *host_base_ptr = args->host_address;
    uint32_t host_offset = (args->pkt_offset / ddt_descr->block_size) * stride;
    uint8_t *host_address = host_base_addr + host_offset;

    for (uint32_t i=0; i<num_blocks; i++)
    {
        PtlHandlerDMAToHostNB(host_address, pkt_payload, block_size, DMA_NO_EVENT);
        pkt_payload += block_size;
        host_address += stride;
    }

    return SPIN_SUCCESS;
}
```

Load DDT info

Compute host memory destination address

Specialized Handlers

NIC Memory

spin_vec_t:
num_blocks: 3
block_size: 2
stride: 3
base_type: int

vector

```
__handler vector_payload_handler(handler_args_t *args)
{
    spin_vec_t *ddt_descr = (spin_vec_t *)args->mem;
    uint32_t num_blocks = args->packet_len / ddt_descr->block_size;
    uint32_t stride = ddt_descr->stride;

    uint8_t *pkt_payload = args->pkt_payload_ptr;

    uint8_t *host_base_ptr = args->host_address;
    uint32_t host_offset = (args->pkt_offset / ddt_descr->block_size) * stride;
    uint8_t *host_address = host_base_addr + host_offset;

    for (uint32_t i=0; i<num_blocks; i++)
    {
        PtlHandlerDMAToHostNB(host_address, pkt_payload, block_size, DMA_NO_EVENT);
        pkt_payload += block_size;
        host_address += stride;
    }

    return SPIN_SUCCESS;
}
```

Load DDT info

Compute host memory destination address

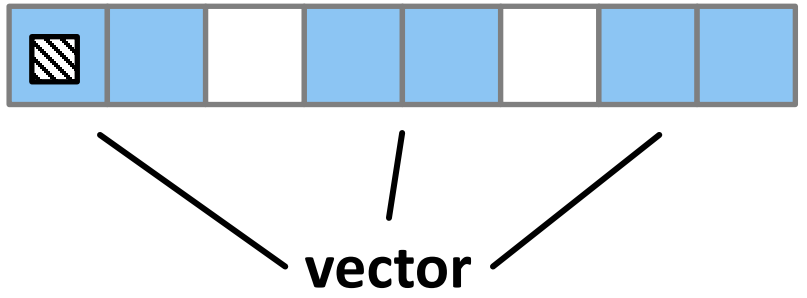
DMA all contig. regions contained in the packet

Specialized Handlers

NIC Memory

spin_vec_t:
num_blocks: 3
block_size: 2
stride: 3
base_type: int

Handler



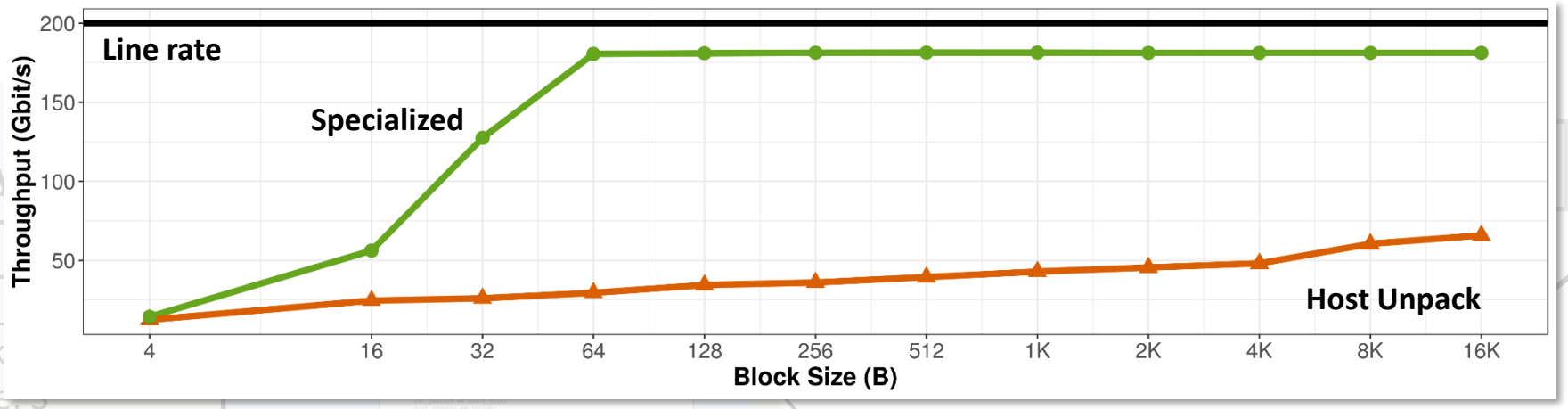

```
__handler vector_payload_handler(handler_args_t *args)
{
    spin_vec_t *ddt_descr = (spin_vec_t *)args->mem;
    uint32_t num_blocks = args->packet_len / ddt_descr->block_size;
    uint32_t stride = ddt_descr->stride;

    uint8_t *pkt_payload = args->pkt_payload_ptr;

    uint8_t *host_base_ptr = args->host_address;
    uint32_t host_offset = (args->pkt_offset / ddt_descr->block_size) * stride;
    uint8_t *host_address = host_base_addr + host_offset;

    for (uint32_t i=0; i<num_blocks; i++)
    {
        PtlHandlerDMAToHostNB(host_address, pkt_payload, block_size, DMA_NO_EVENT);
        pkt_payload += block_size;
        host_address += stride;
    }

    return SPIN_SUCCESS;
}
```

Load DDT info

Compute host memory destination address

DMA all contig. regions contained in the packet

Specialized Handlers

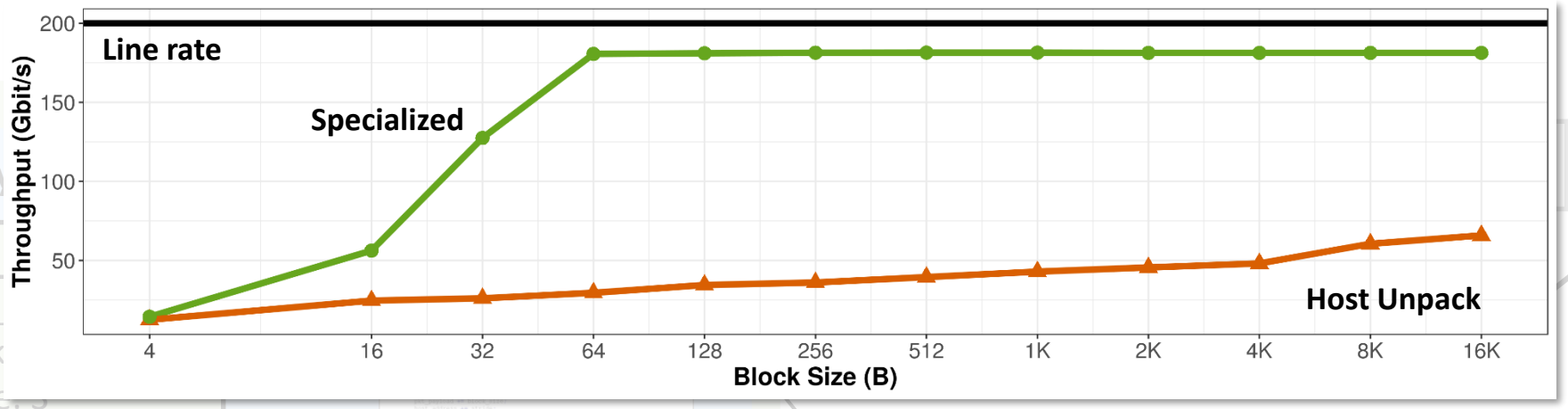


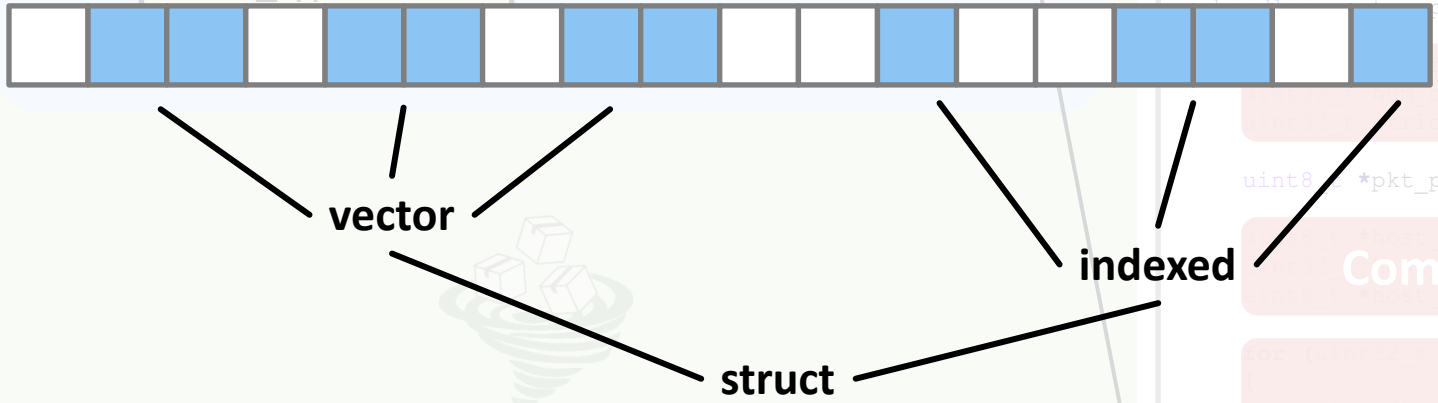
NIC Memory

```
spin_
num_
block
stride
base_type: int
```

```
__handler vector_payload_handler(handler_args_t *args)
{
    Load DDT info

    uint8_t *pkt_payload = args->pkt_payload_ptr;

    Compute host memory destination address

    DMA all contig. regions contained in the packet

    return SPIN_SUCCESS;
}
```

Specialized Handlers



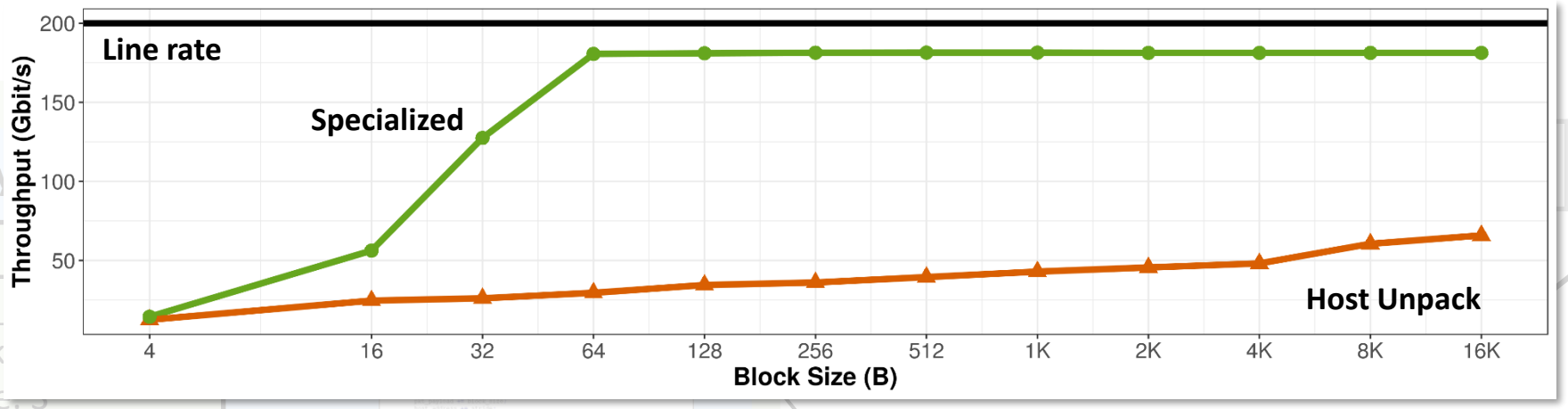


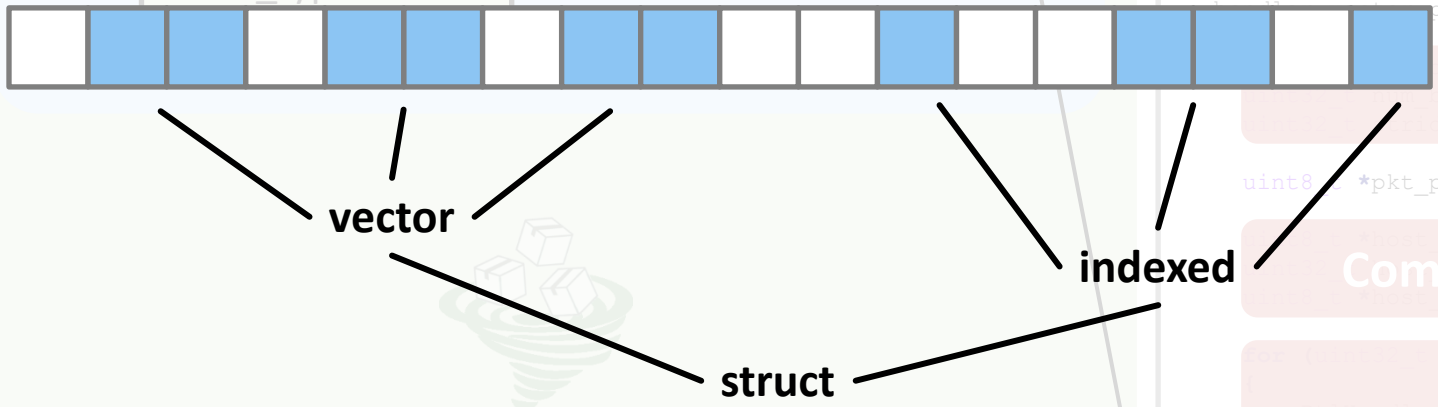
Load DDT info

Compute host memory destination address

DMA all contig. regions contained in the packet

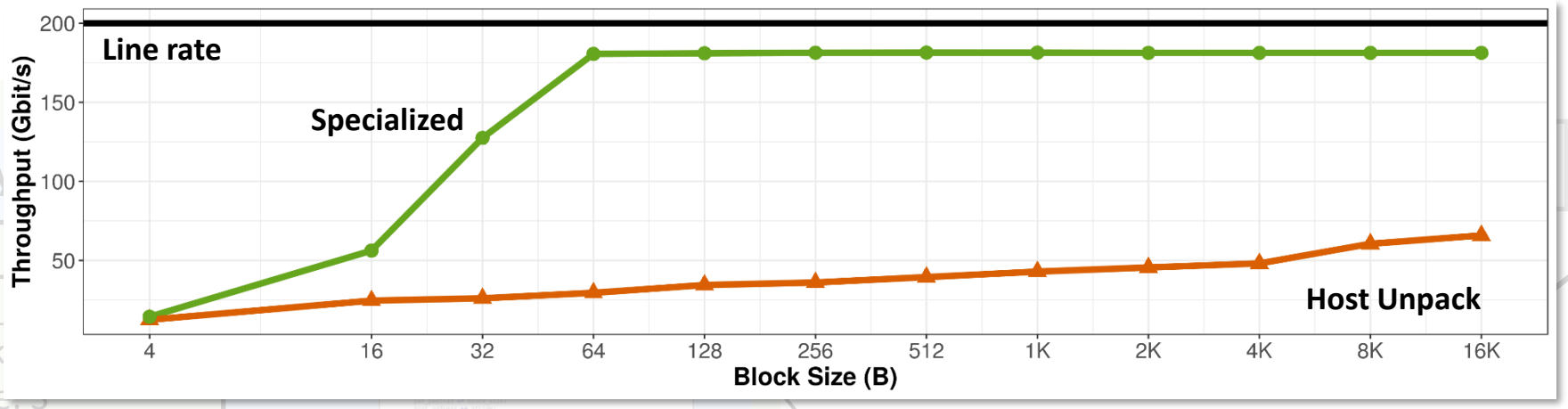
Specialized Handlers

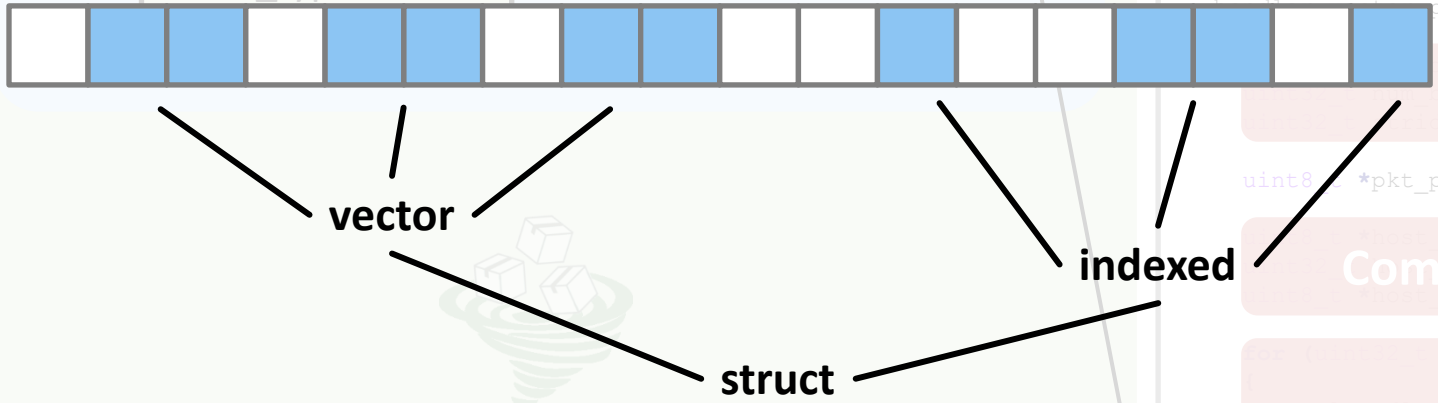


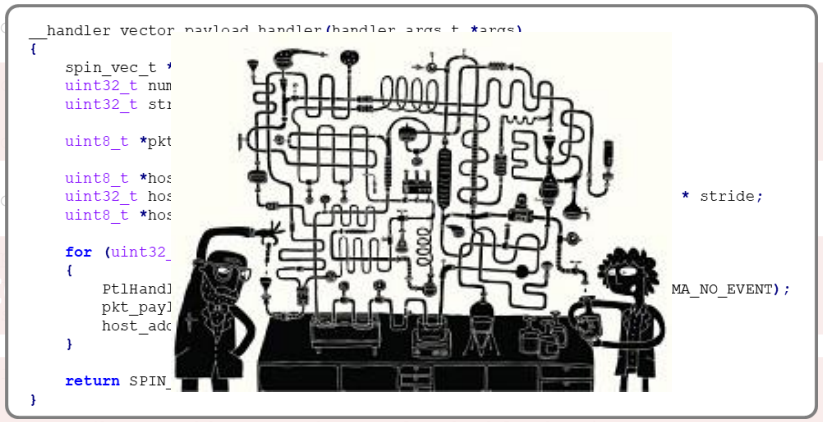



```
__handler vector_payload_handler(handler_args_t *args)
{
    spin_vec_t *ddt_descr = (spin_vec_t *)args->mem;
    uint32_t num_blocks = args->packet_len / ddt_descr->block_size;
    uint32_t stride = ddt_descr->stride;

    uint8_t *pkt_payload = args->pkt_payload_ptr;

    uint8_t *host_base_ptr = args->host_address;
    uint32_t host_offset = (args->pkt_offset / ddt_descr->block_size;) * stride;
    uint8_t *host_address = host_base_addr + host_offset;

    for (uint32_t i=0; i<num_blocks; i++)
    {
        PtlHandlerDMAToHostNB(host_address, pkt_payload, block_size, DMA_NO_EVENT);
        pkt_payload += block_size;
        host_address += stride;
    }

    return SPIN_SUCCESS;
}
```

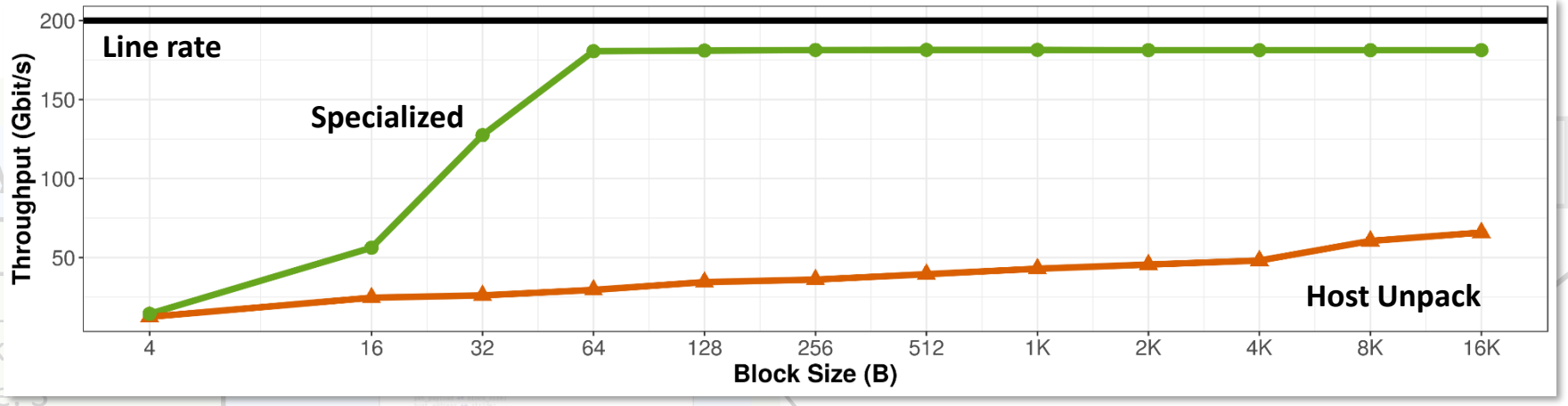
Specialized Handlers

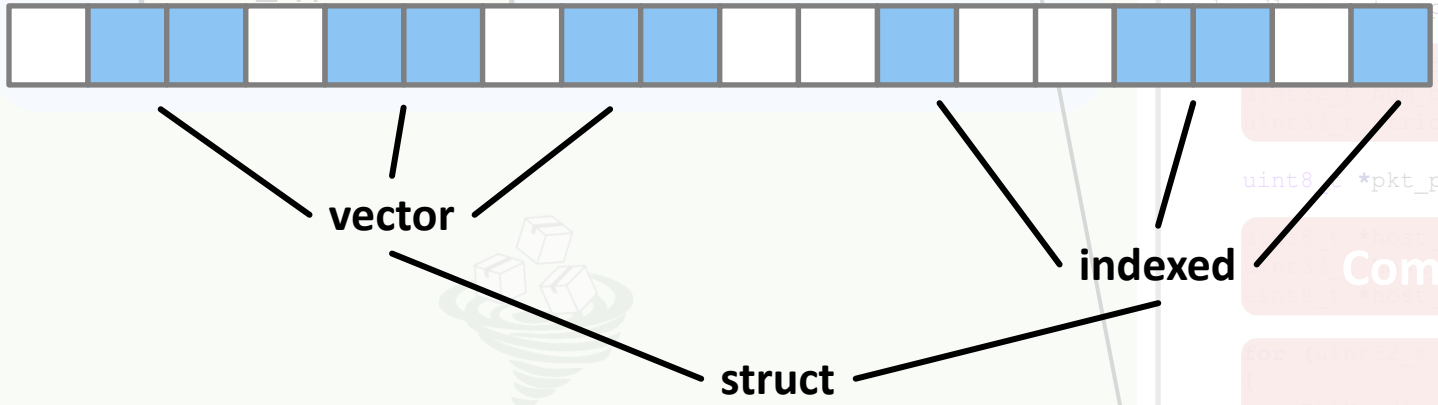


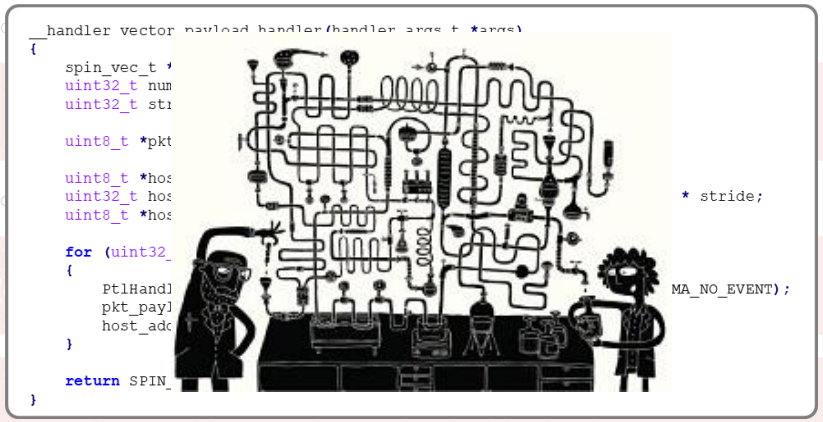




Specialized Handlers

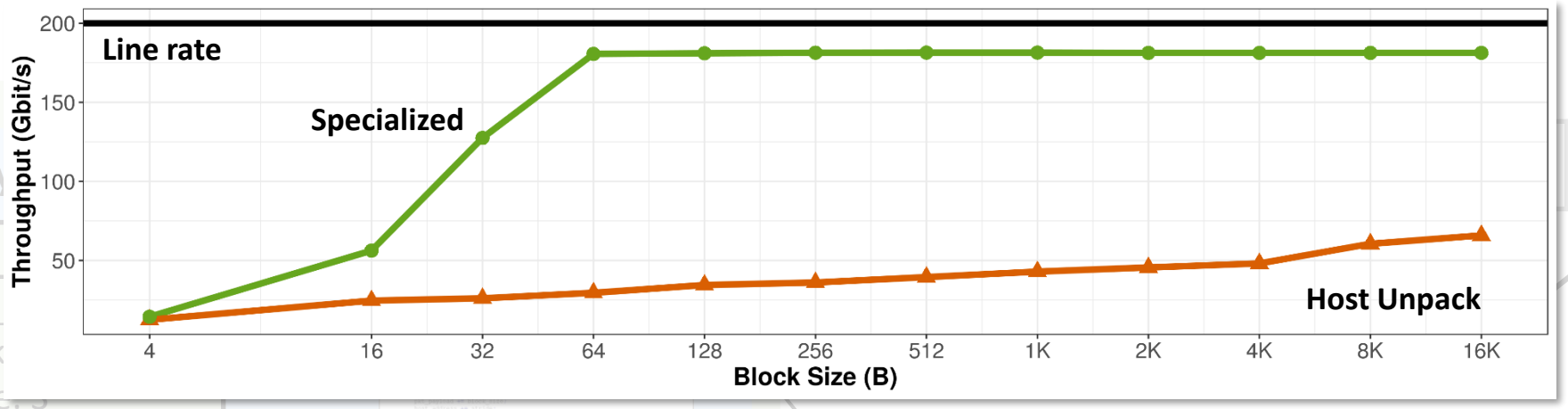


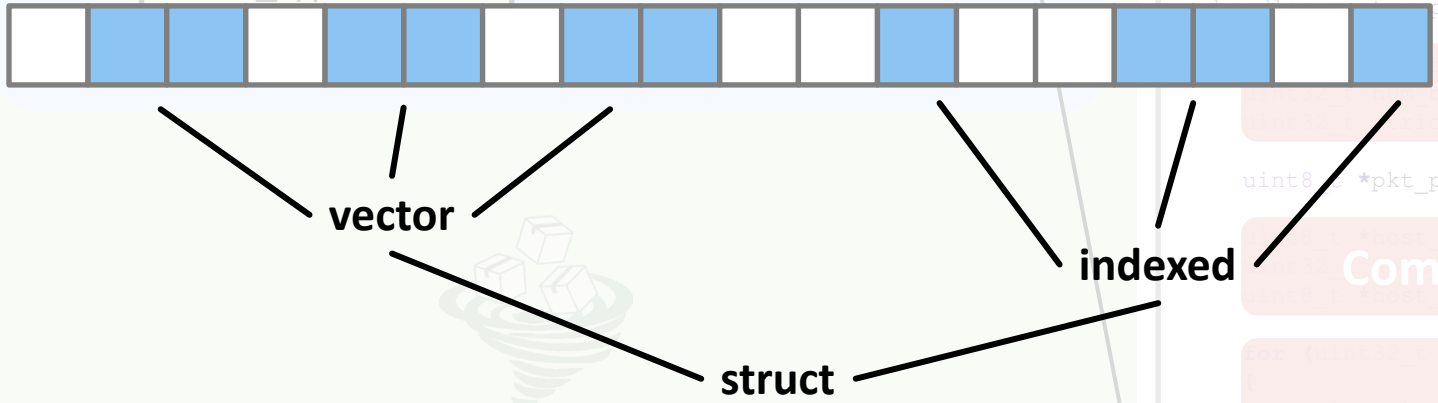


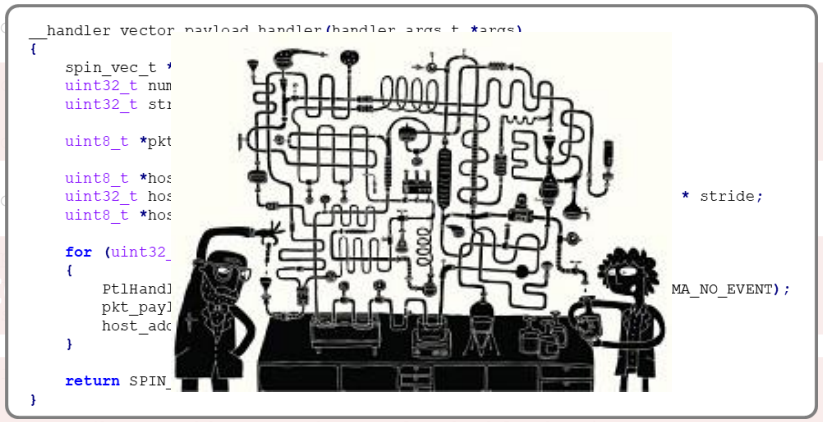


Need a different handlers for each possible derived datatype!

Specialized Handlers



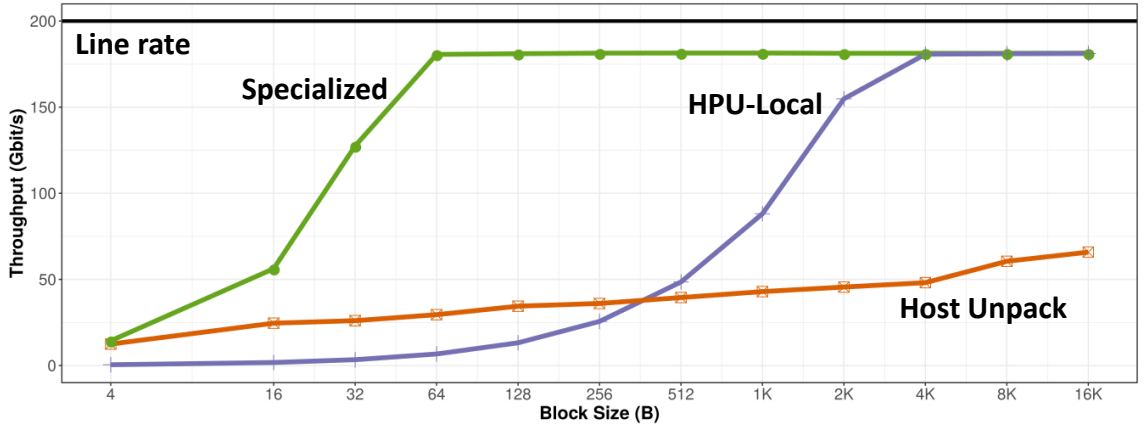




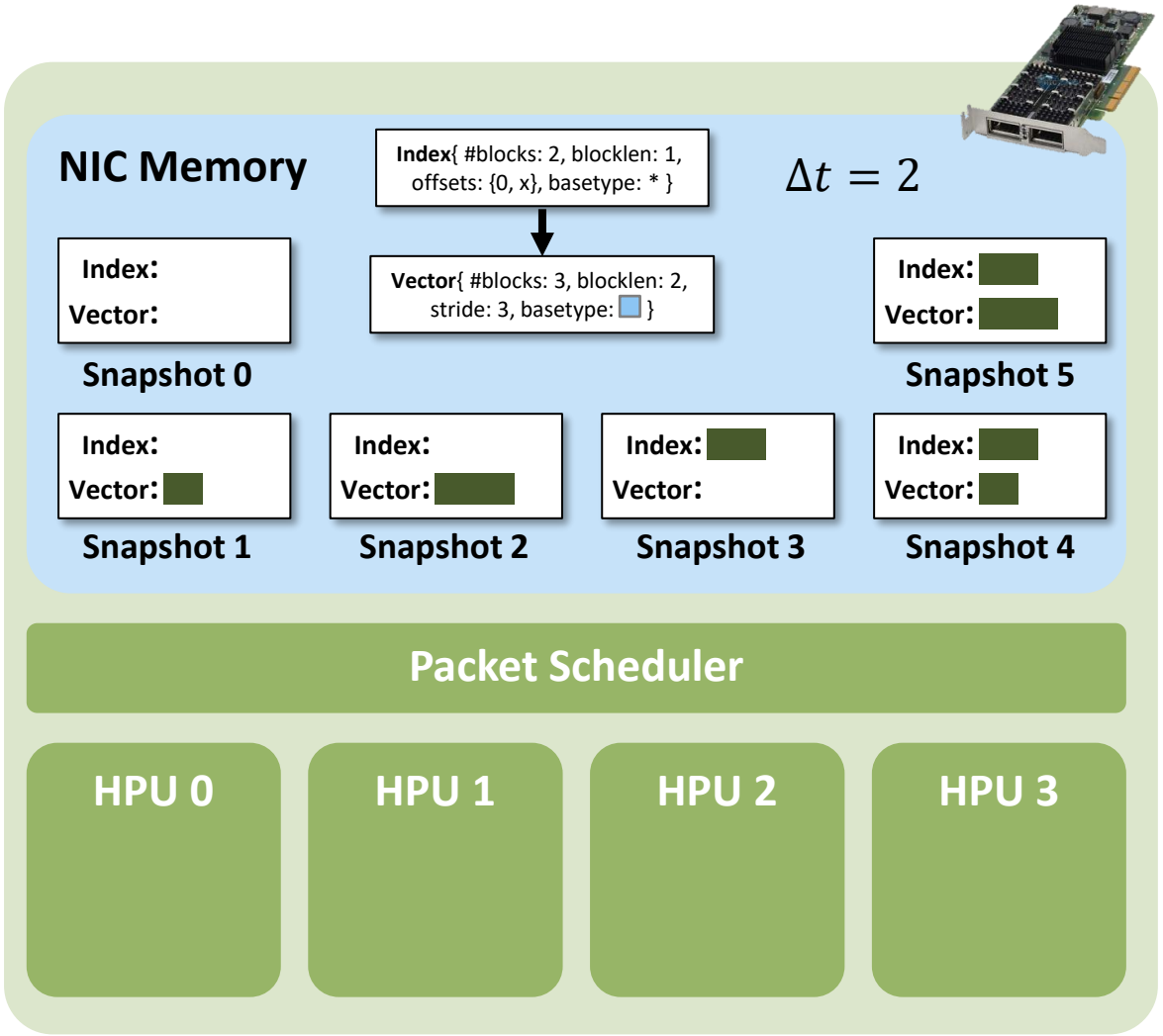
Can we define a general handler to process any datatype?

MPI Types Library on sPIN

HPU-Local: each HPU has its own state

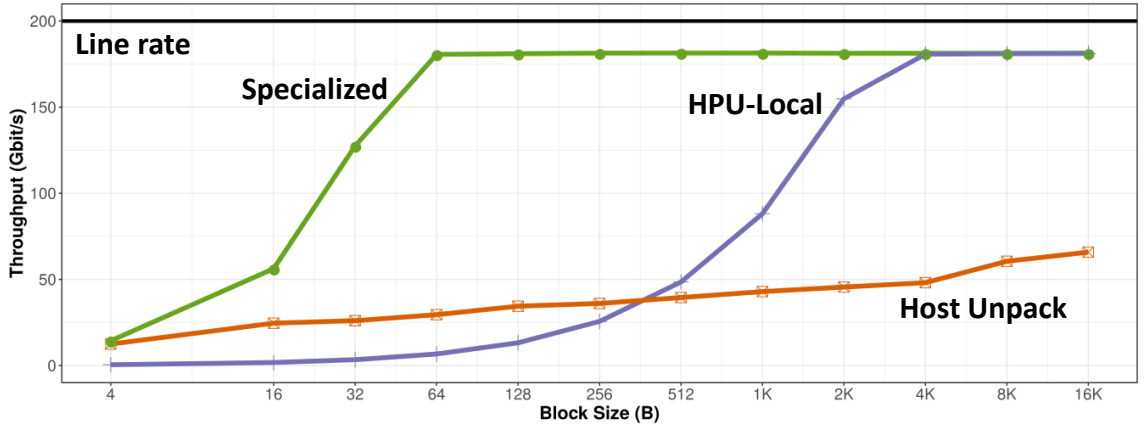


MPI Types Library on sPIN

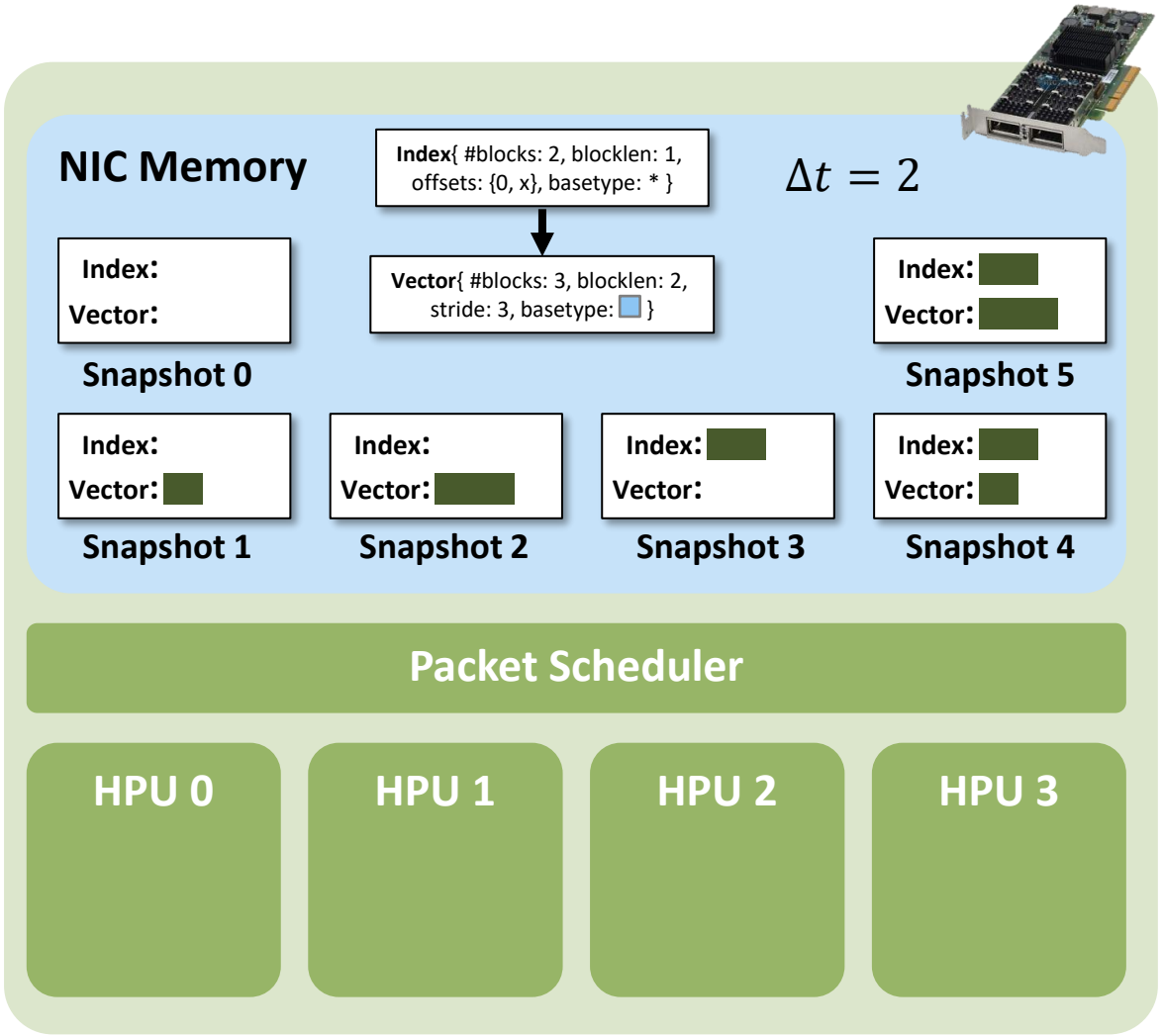


HPU-Local: each HPU has its own state

RO-checkpoints: pre-computed checkpoints shared by multiple HPUs (read-only)

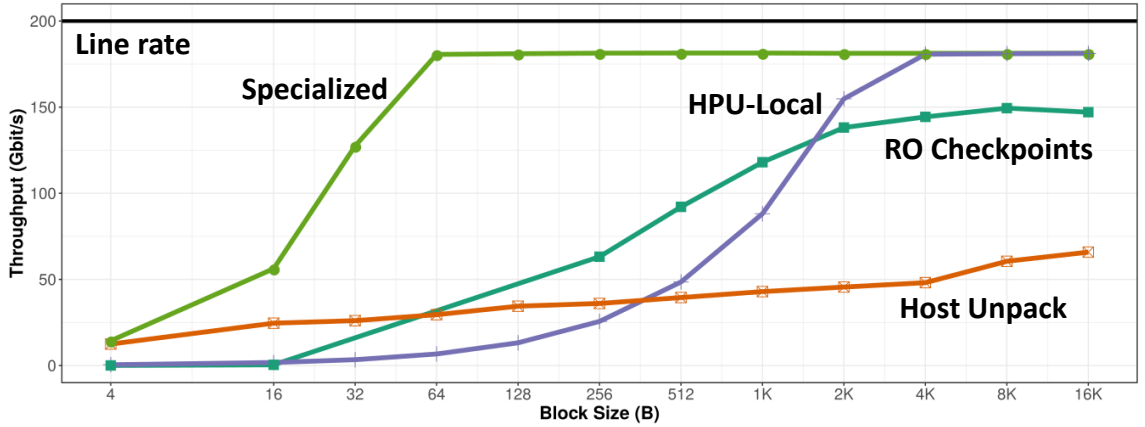


MPI Types Library on sPIN

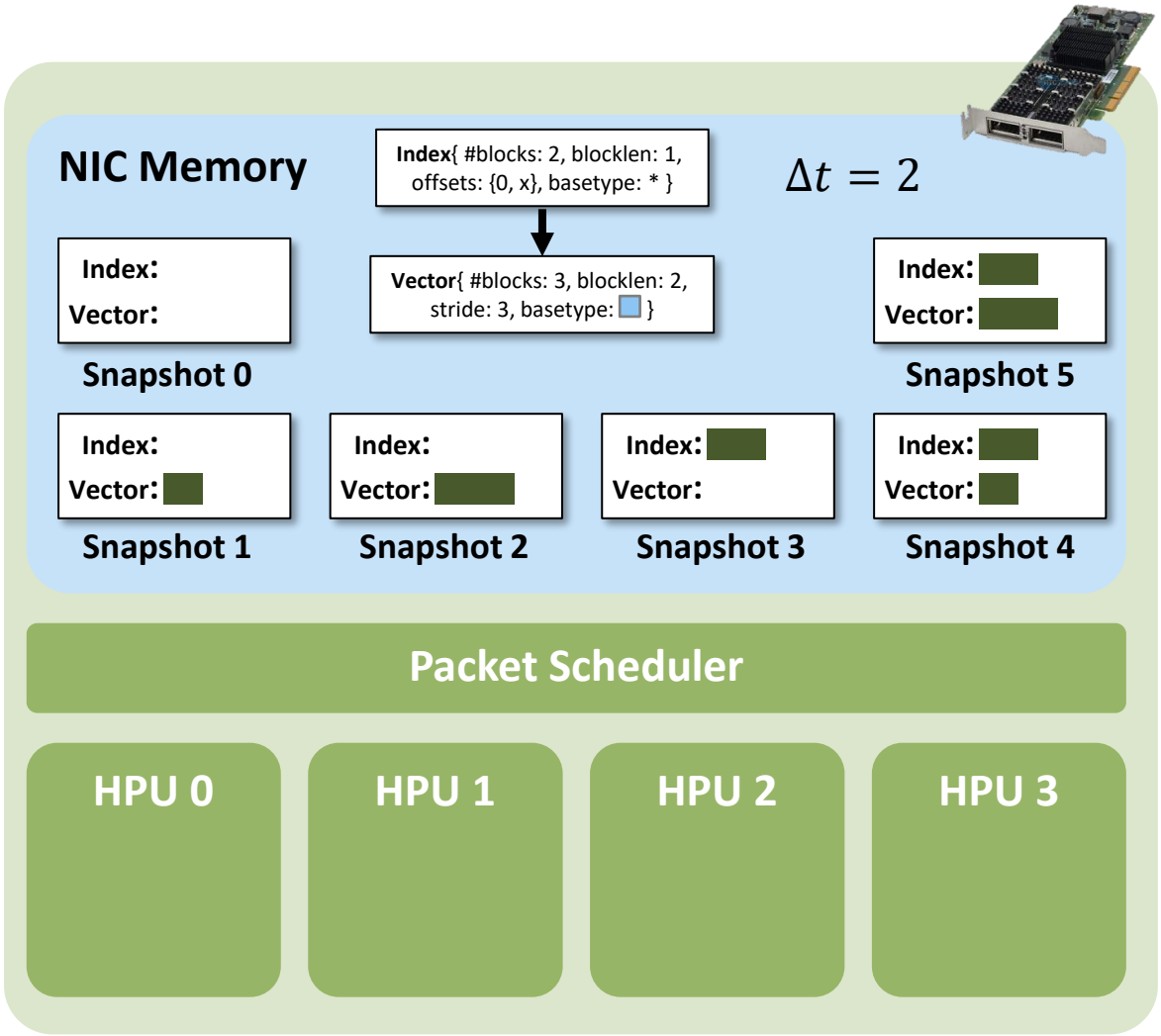


HPU-Local: each HPU has its own state

RO-checkpoints: pre-computed checkpoints shared by multiple HPUs (read-only)



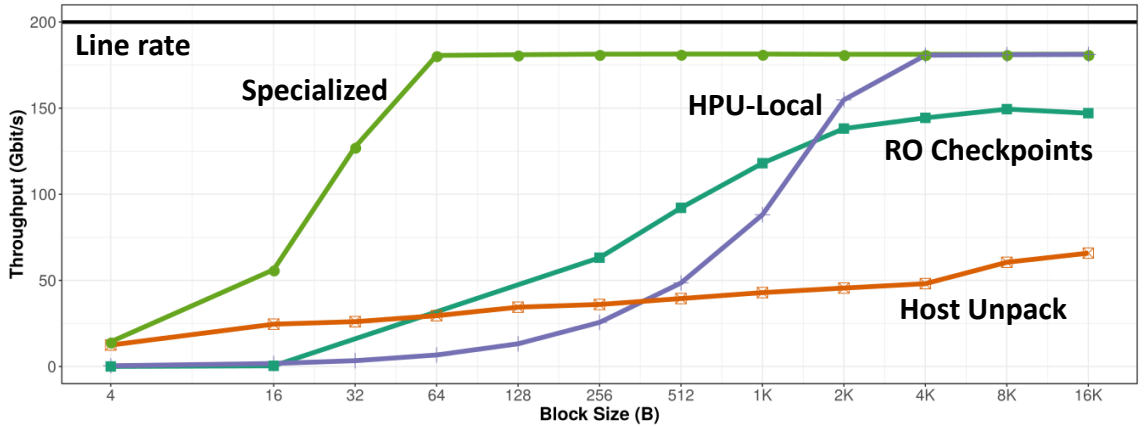
MPI Types Library on sPIN



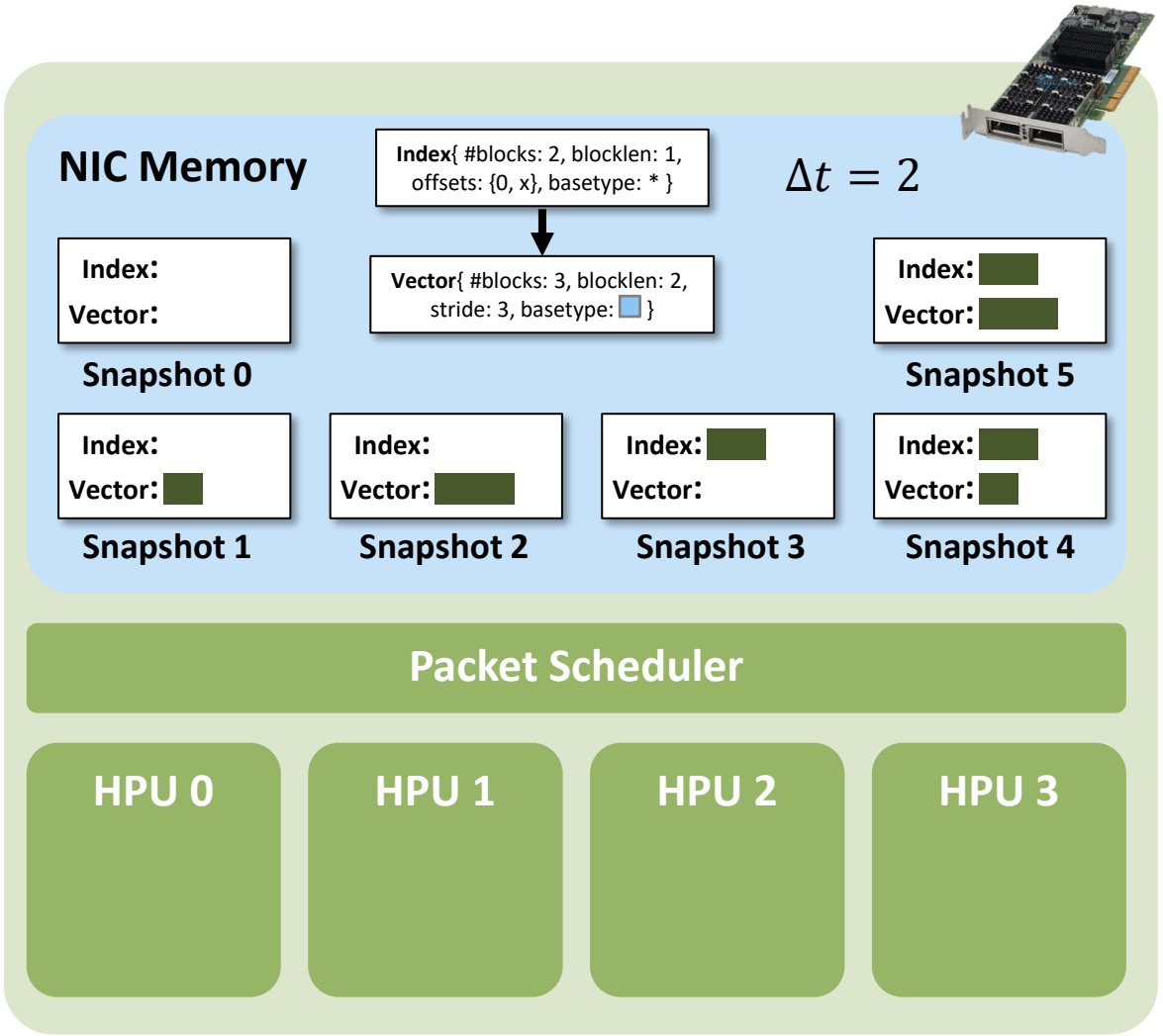
HPU-Local: each HPU has its own state

RO-checkpoints: pre-computed checkpoints shared by multiple HPUs (read-only)

RW-checkpoints: pre-computed checkpoints shared by multiple HPUs (read/write, fine-grain synchronization)



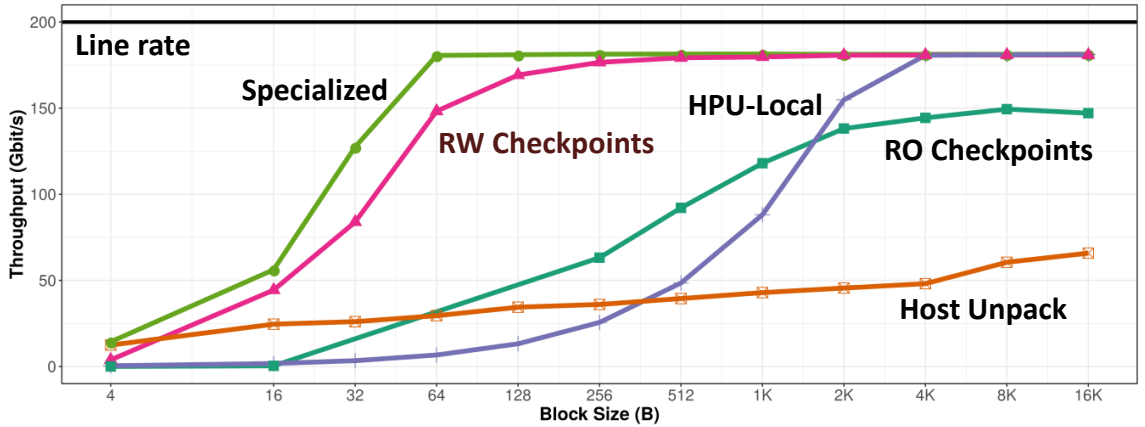
MPI Types Library on sPIN



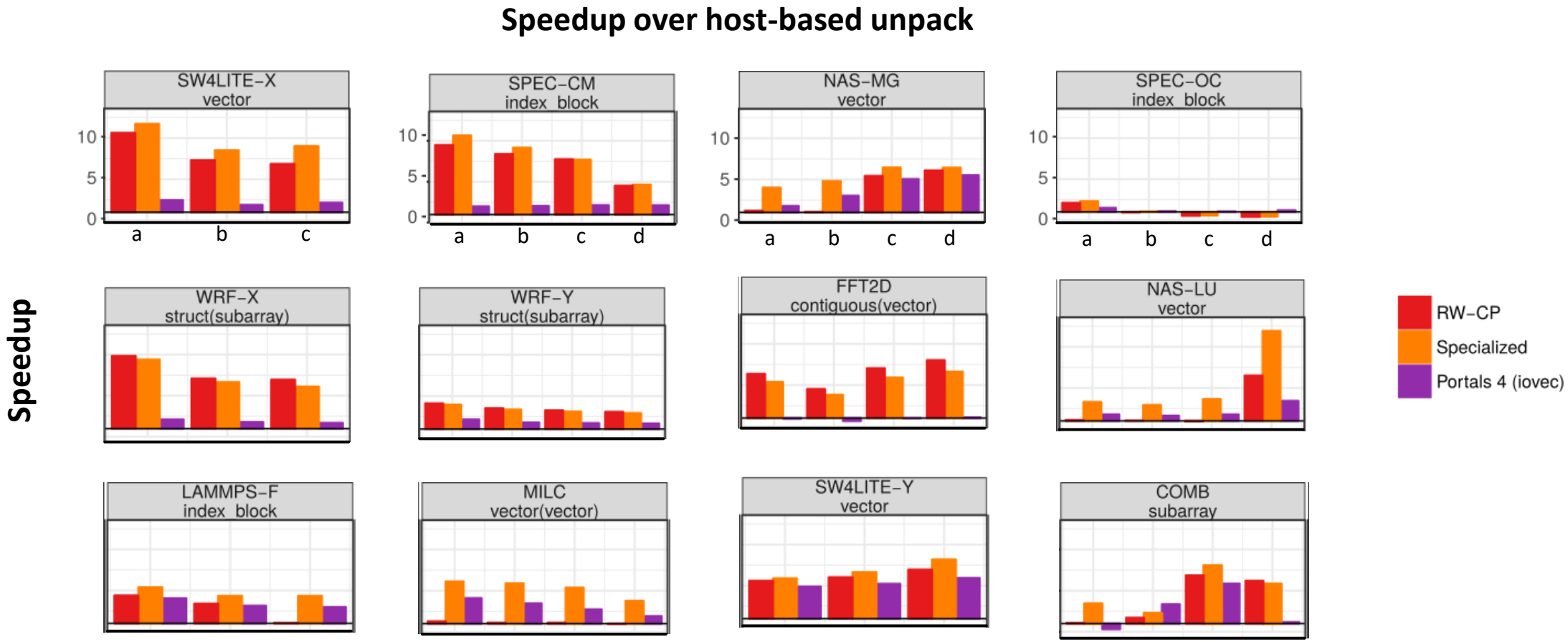
HPU-Local: each HPU has its own state

RO-checkpoints: pre-computed checkpoints shared by multiple HPUs (read-only)

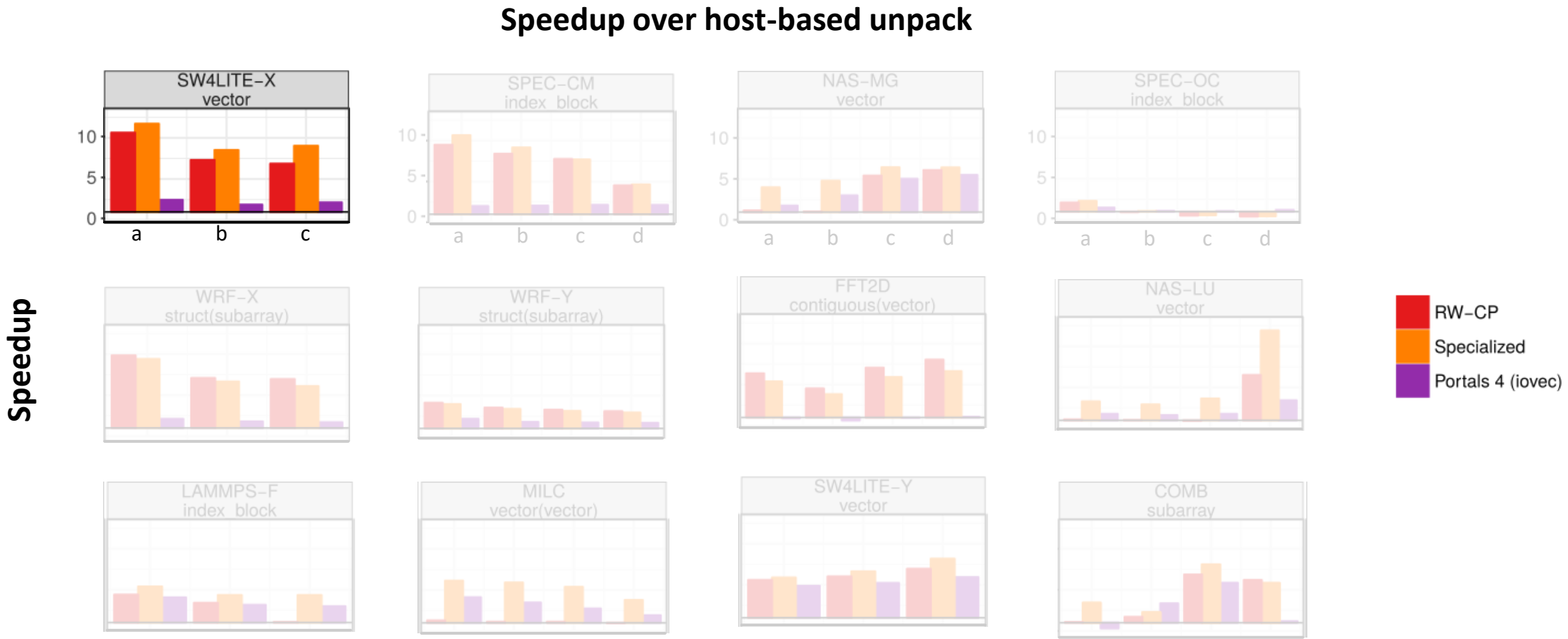
RW-checkpoints: pre-computed checkpoints shared by multiple HPUs (read/write, fine-grain synchronization)



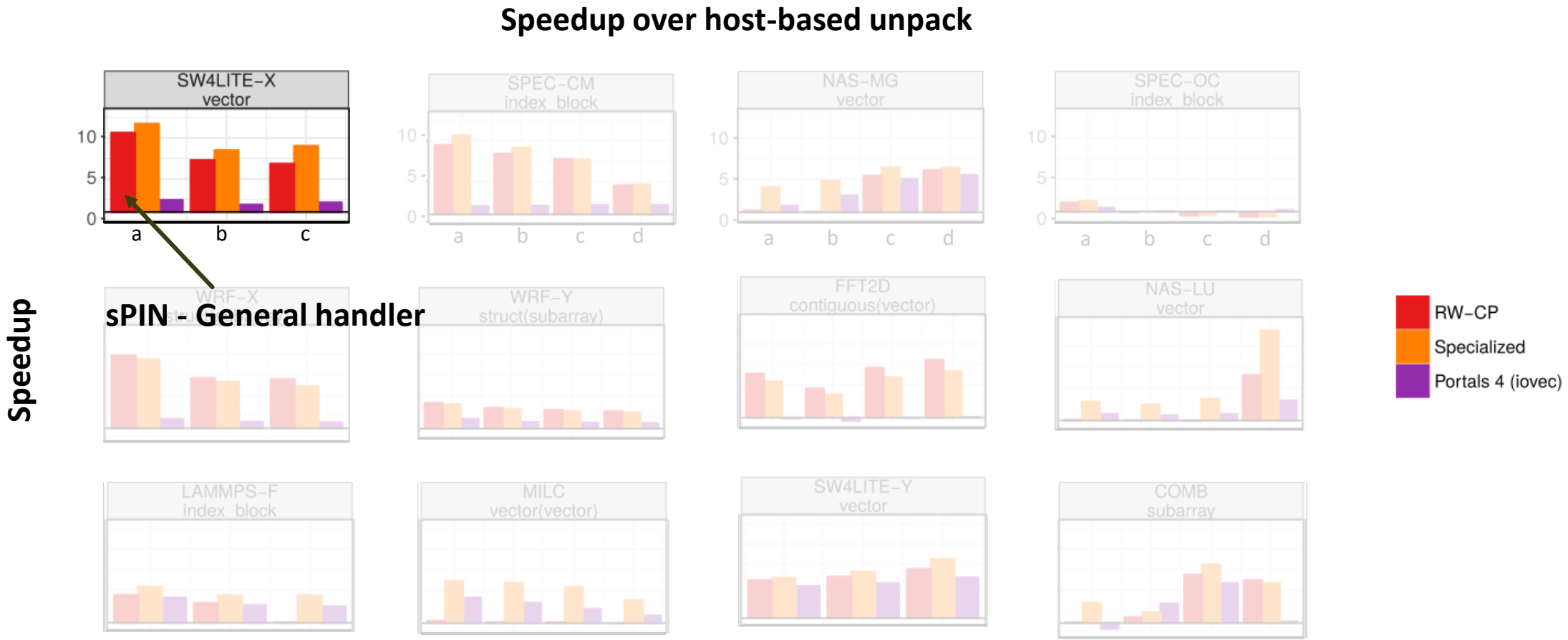
Real Applications DDTs



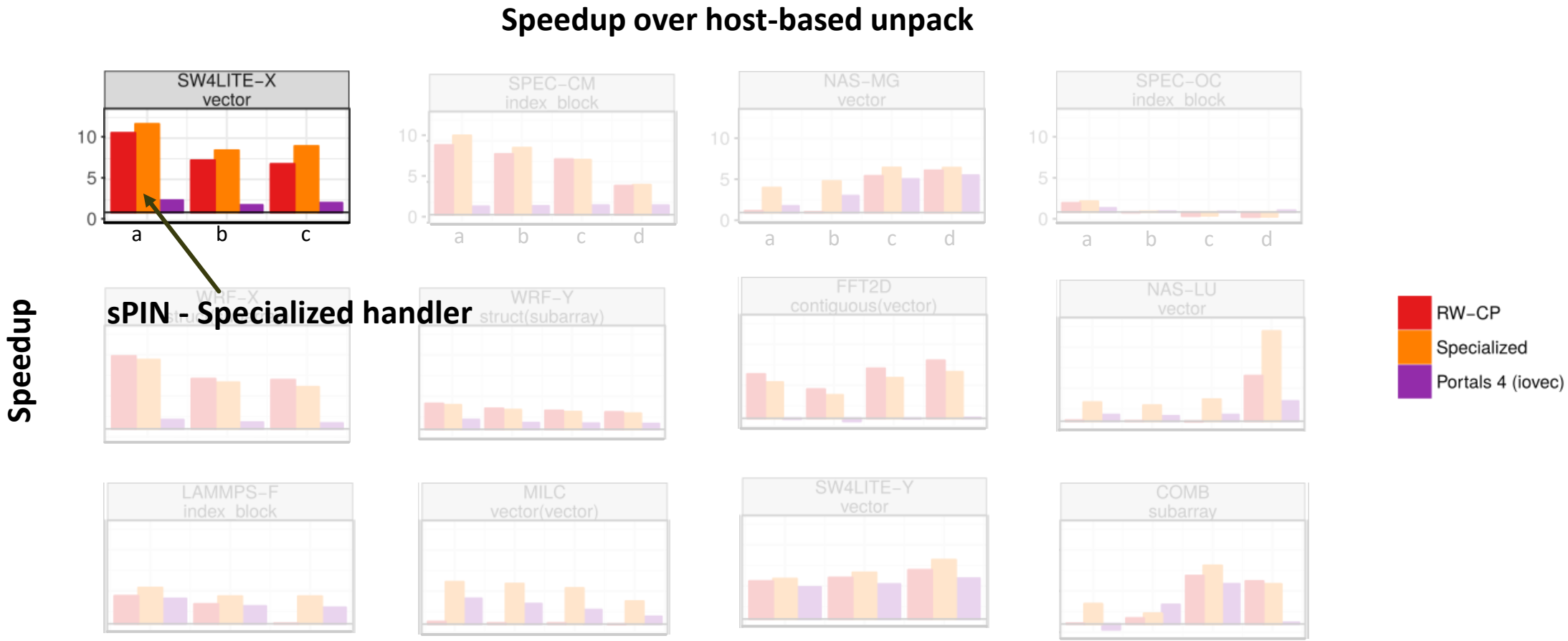
Real Applications DDTs



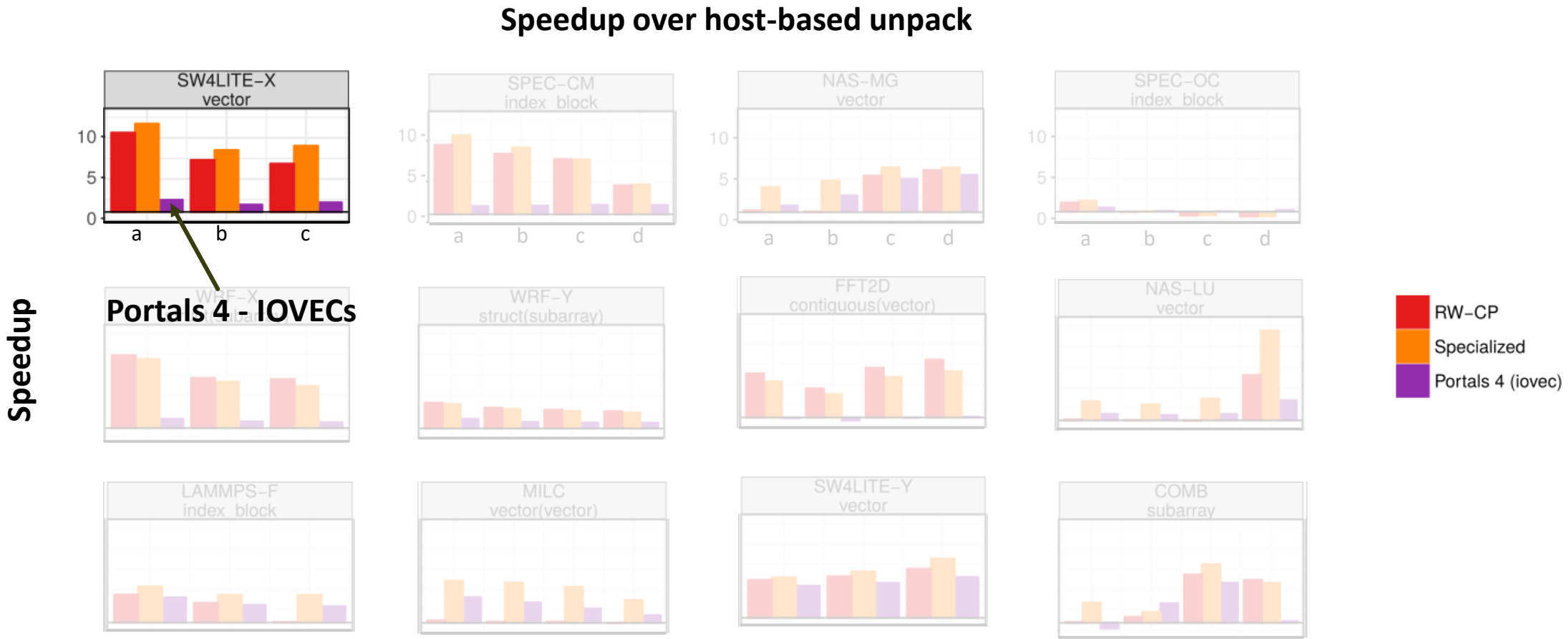
Real Applications DDTs



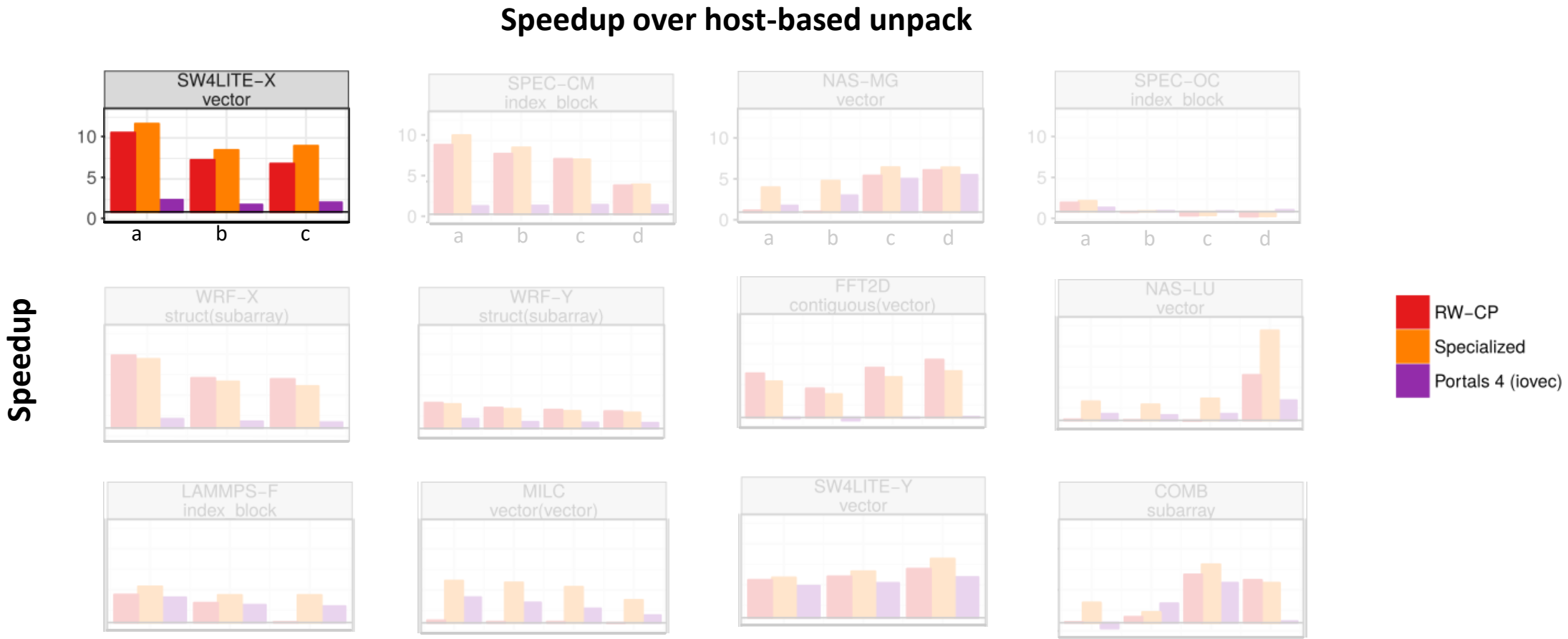
Real Applications DDTs



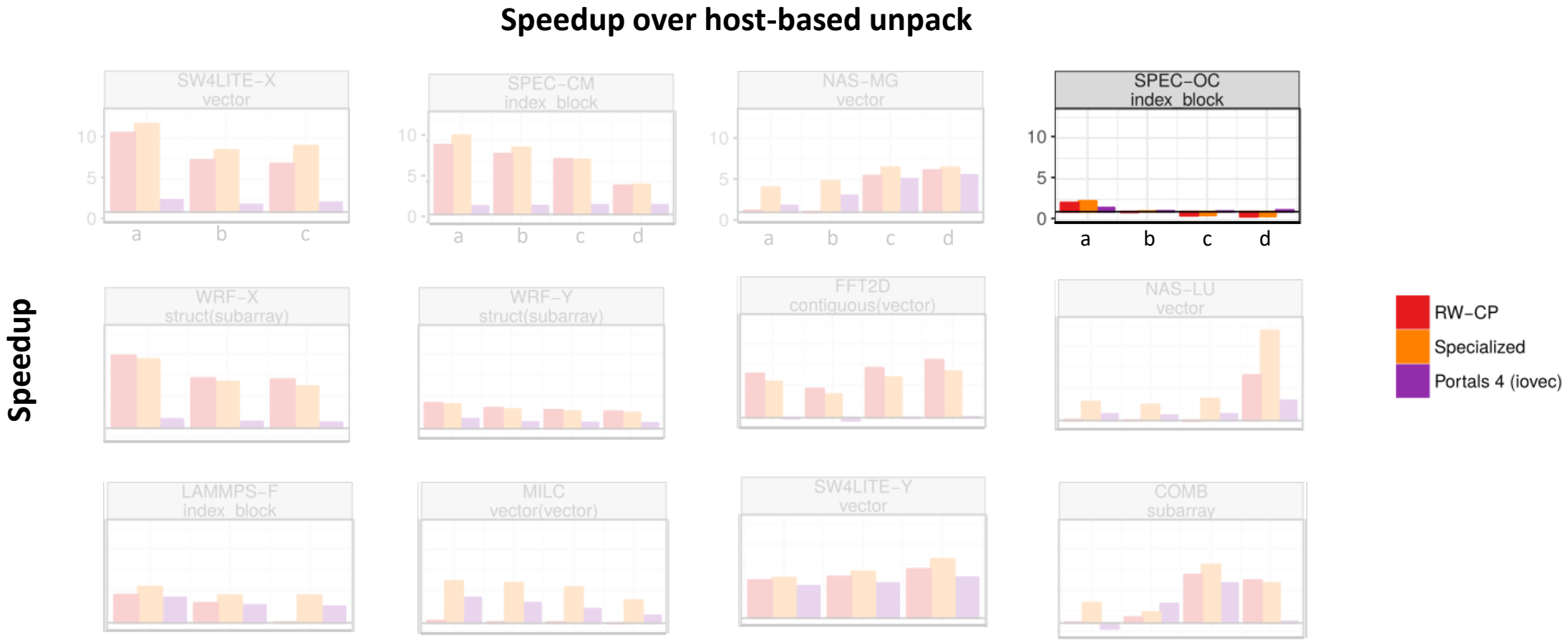
Real Applications DDTs



Real Applications DDTs

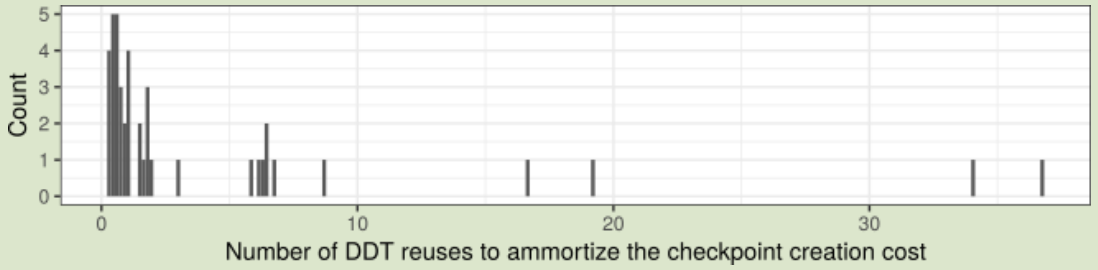


Real Applications DDTs



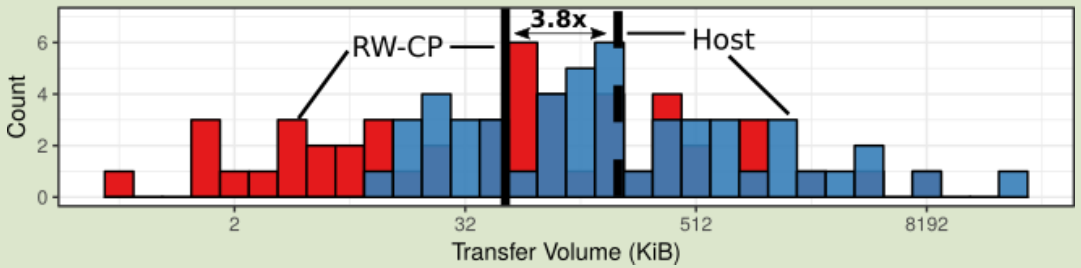
Real Applications DDTs

Checkpointing Overhead



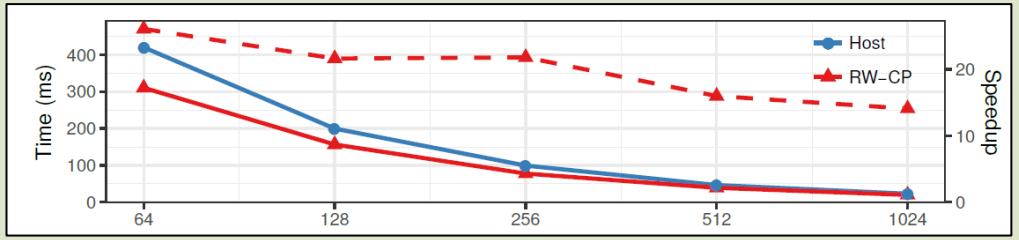
75% of the analyzed DDTs amortized after 4 reuses

Data Movement

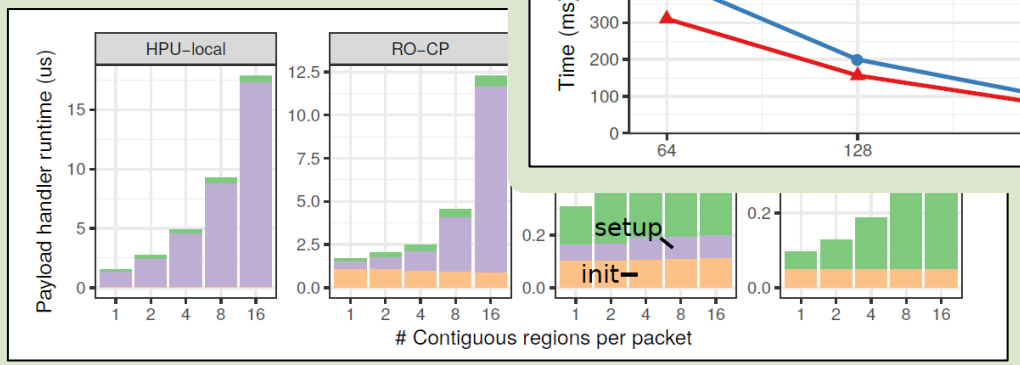


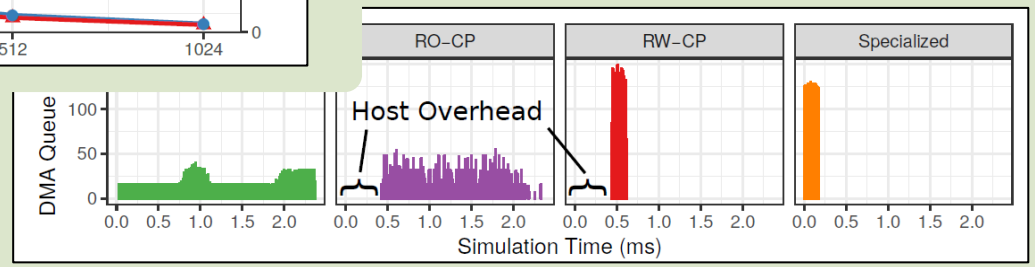
Up to 3.8x less moved data volume

Full app speedup (FFT2D)

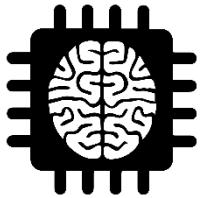


Handler Analysis





Network-accelerated datatypes

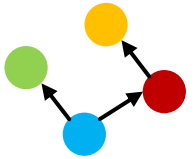


Quantization

Erasure coding

Distributed File Systems

Zoo-sPINNER
consensus on sPIN



Network Group Communication

Packet classification and pattern matching

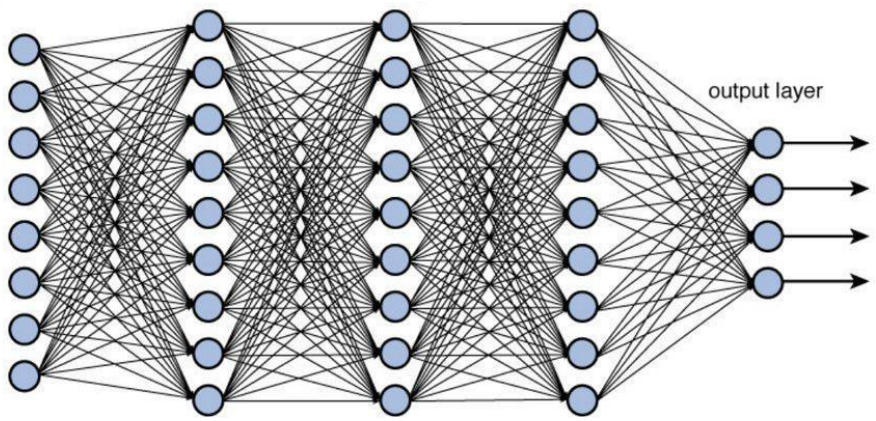
In-network allreduce

Serverless sPIN

Fast, scalable, and reliable storage is a first-class requirement of both HPC systems and datacenters.

Fast, scalable, and reliable storage is a first-class requirement of both HPC systems and datacenters.

Up to 60% I/O overhead [1].

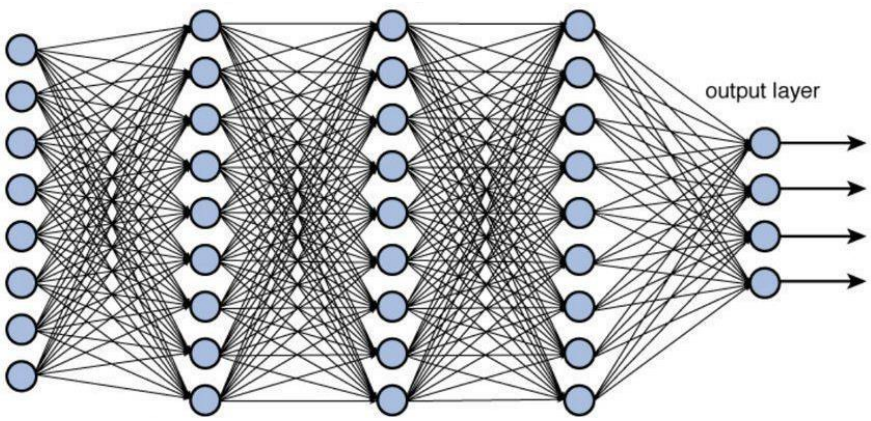


https://towardsdatascience.com/training-deep-neural-networks-9fdb1964b964

[1] Sarunya Pumma, Min Si, Wu-chun Feng, and Pavan Balaji. 2017. Parallel I/O optimizations for scalable deep learning. In 2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS). IEEE, 720–729.

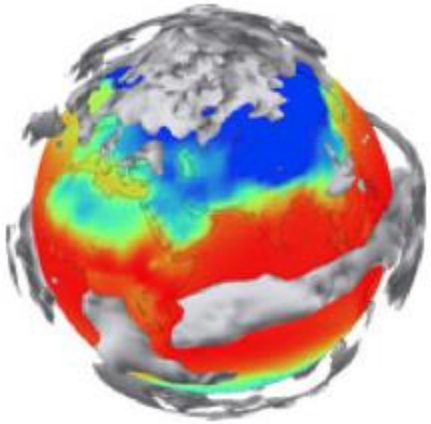
Fast, scalable, and reliable storage is a first-class requirement of both HPC systems and datacenters.

Up to 60% I/O overhead [1].



https://towardsdatascience.com/training-deep-neural-networks-9fdb1964b964

Up to 90% I/O overhead [2].

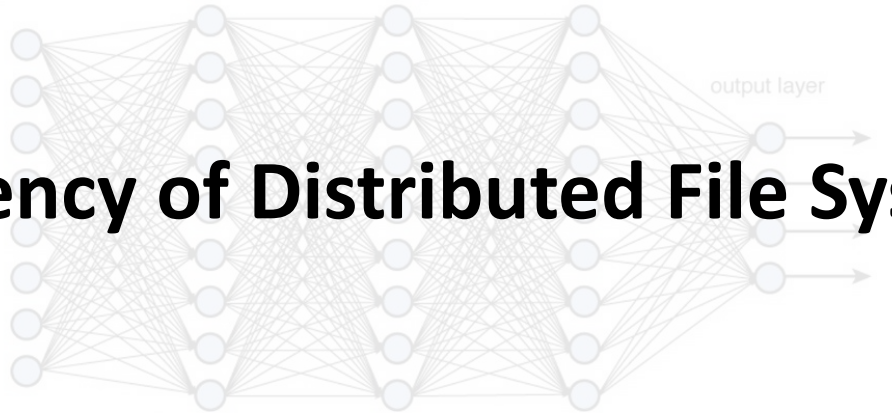


https://www2.cisl.ucar.edu/user-support/allocations/climate-simulation-laboratory-csl

[1] Sarunya Pumma, Min Si, Wu-chun Feng, and Pavan Balaji. 2017. Parallel I/O optimizations for scalable deep learning. In 2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS). IEEE, 720–729.
[2] Sriram Lakshminarasimhan, David A Boyuka, Saurabh V Pendse, Xiaocheng Zou, John Jenkins, Venkatram Vishwanath, Michael E Papka, and Nagiza F Samatova. 2013. Scalable in situ scientific data encoding for analytical query processing. In Proceedings of the 22nd international symposium on High-performance parallel and distributed computing. 1–12.

Fast, scalable, and reliable storage is a first-class requirement of both HPC systems and datacenters.

Up to 60% I/O overhead [1].



https://towardsdatascience.com/training-deep-neural-networks-9fdb1964b964

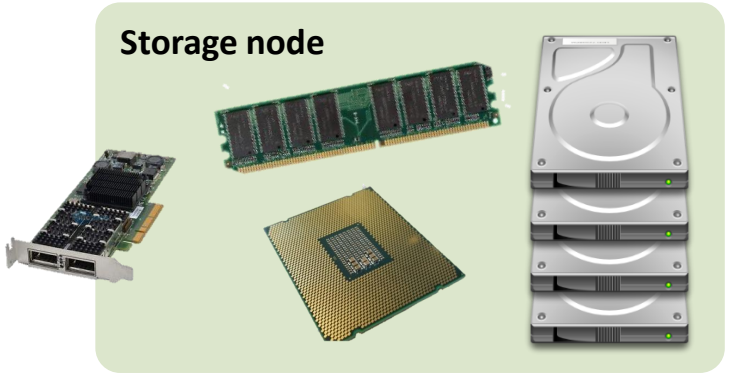
Up to 90% I/O overhead [2].

https://www2.cisl.ucar.edu/user-support/allocations/climate-simulation-laboratory-csl

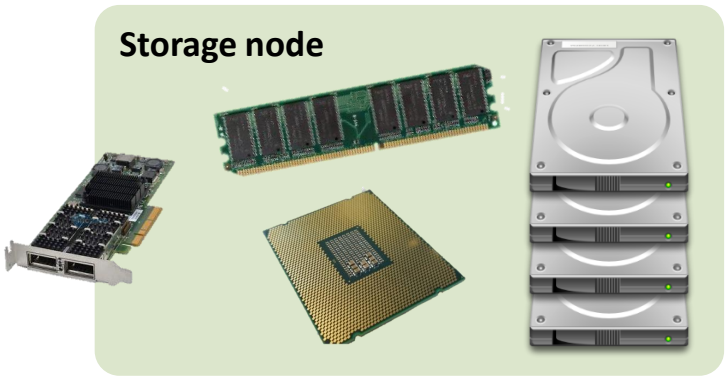
Efficiency of Distributed File Systems (DFS) is crucial in these systems

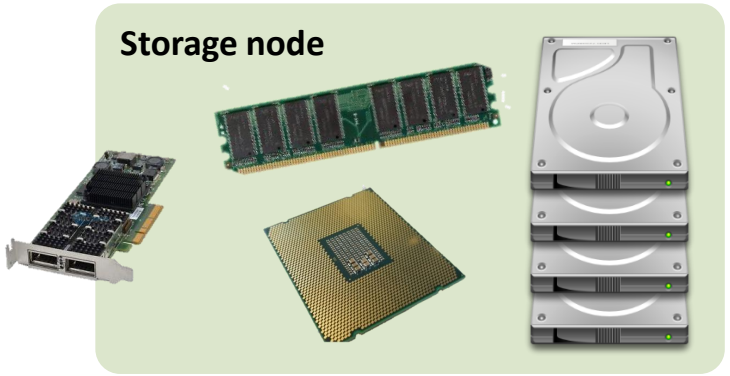
[1] Sarunya Pumma, Min Si, Wu-chun Feng, and Pavan Balaji. 2017. Parallel I/O optimizations for scalable deep learning. In 2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS). IEEE, 720–729.
[2] Sriram Lakshminarasimhan, David A Boyuka, Saurabh V Pendse, Xiaocheng Zou, John Jenkins, Venkatram Vishwanath, Michael E Papka, and Nagiza F Samatova. 2013. Scalable in situ scientific data encoding for analytical query processing. In Proceedings of the 22nd international symposium on High-performance parallel and distributed computing. 1–12.

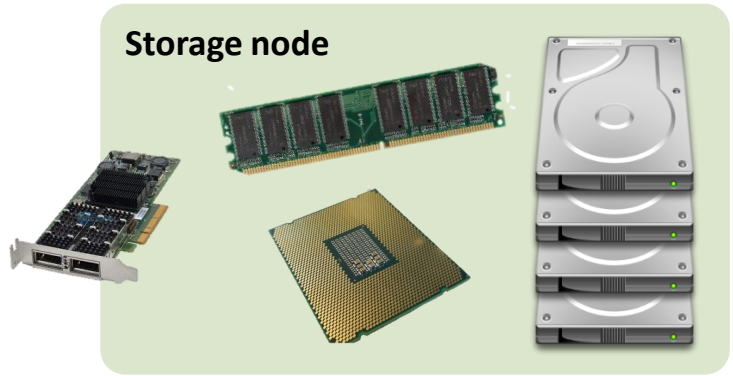
Distributed File Systems



Distributed File Systems



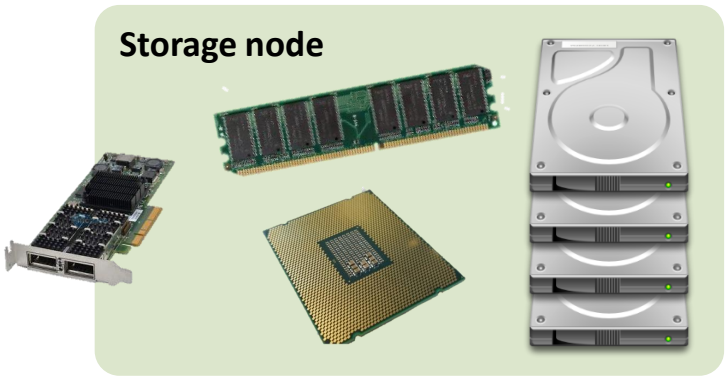


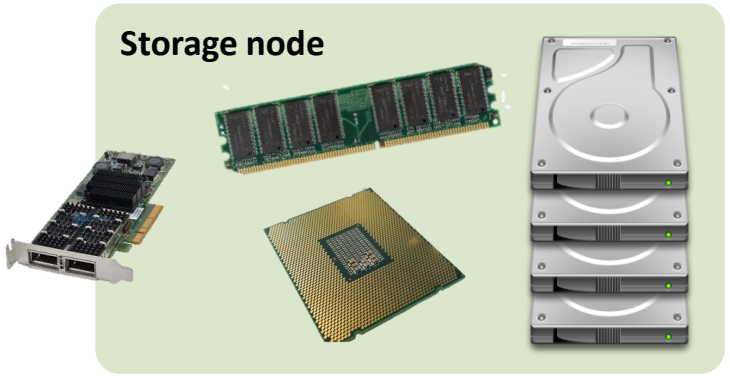


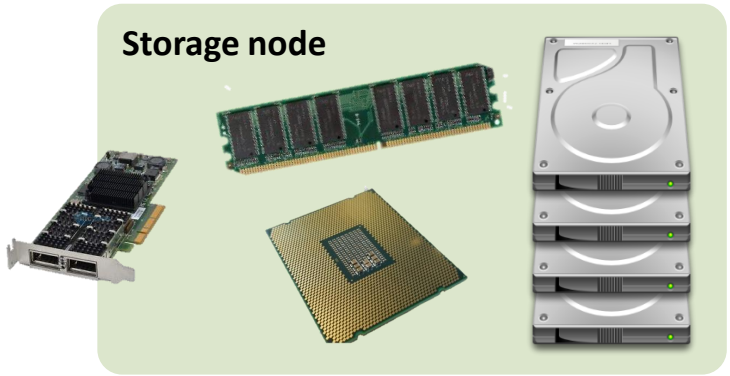
Distributed File Systems

Metadata nodes

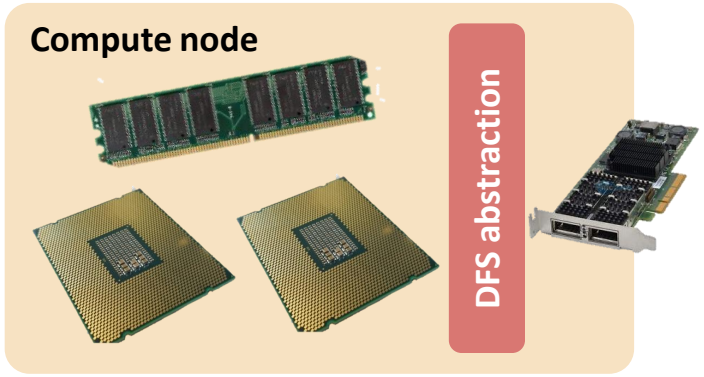
Management nodes

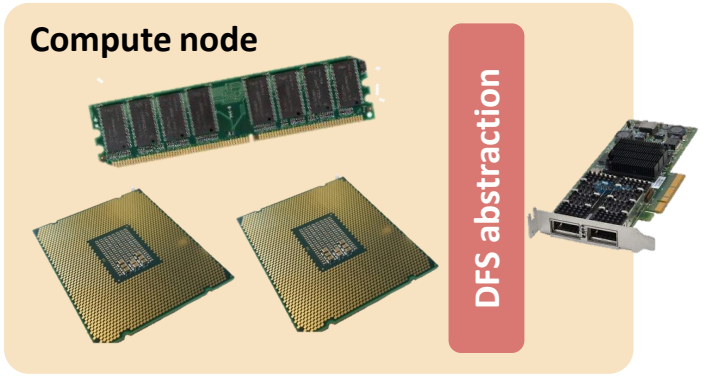






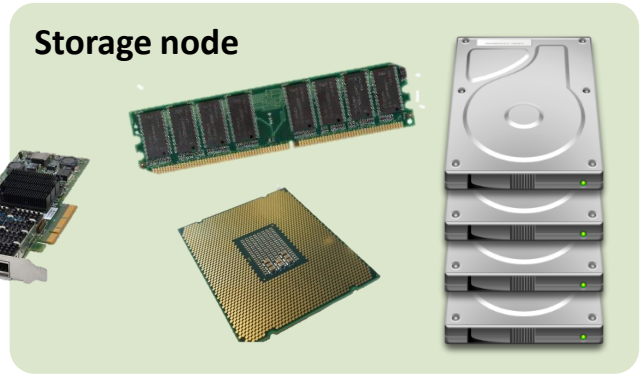
Distributed File Systems

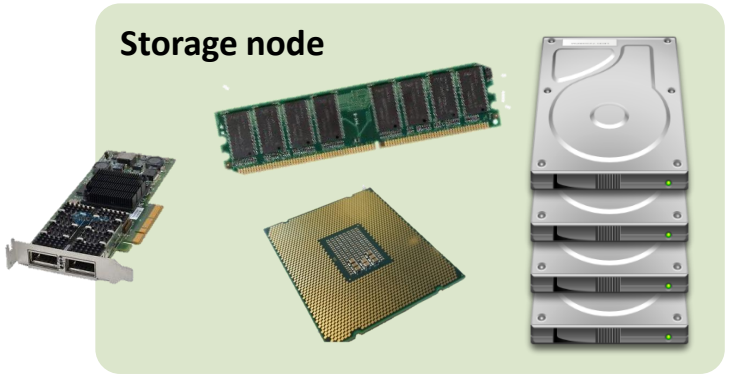


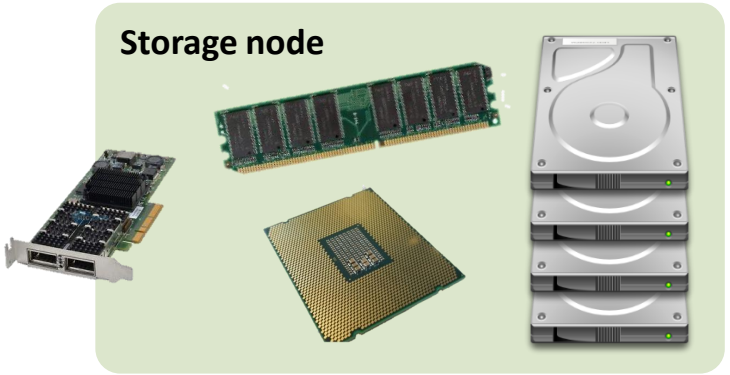


Metadata nodes

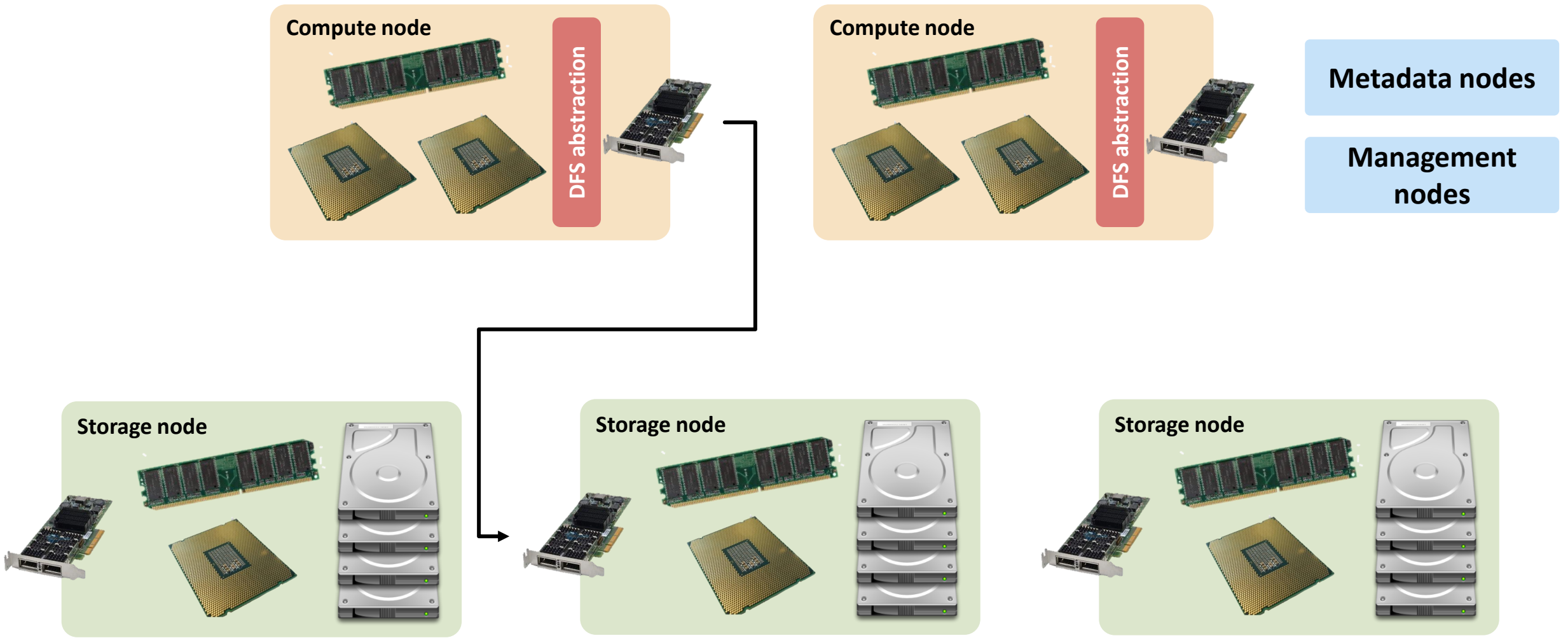
Management nodes



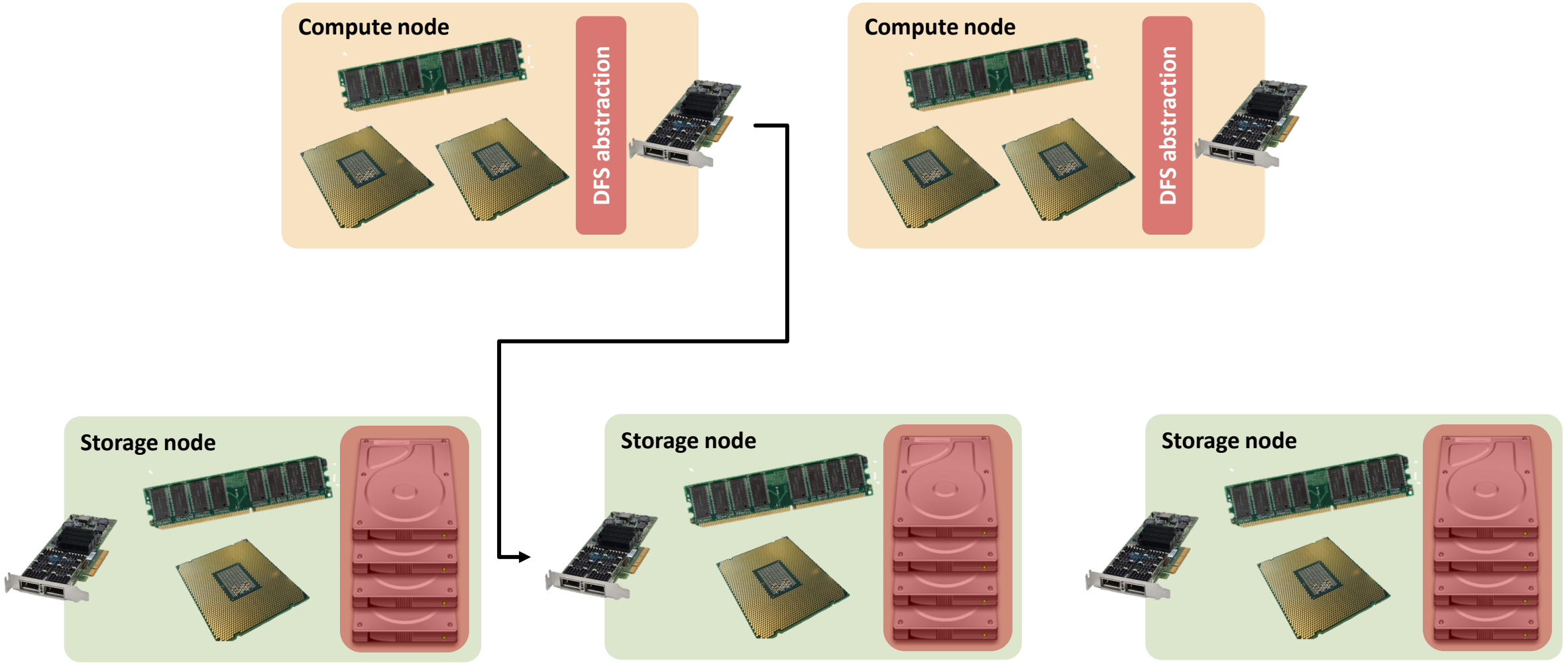




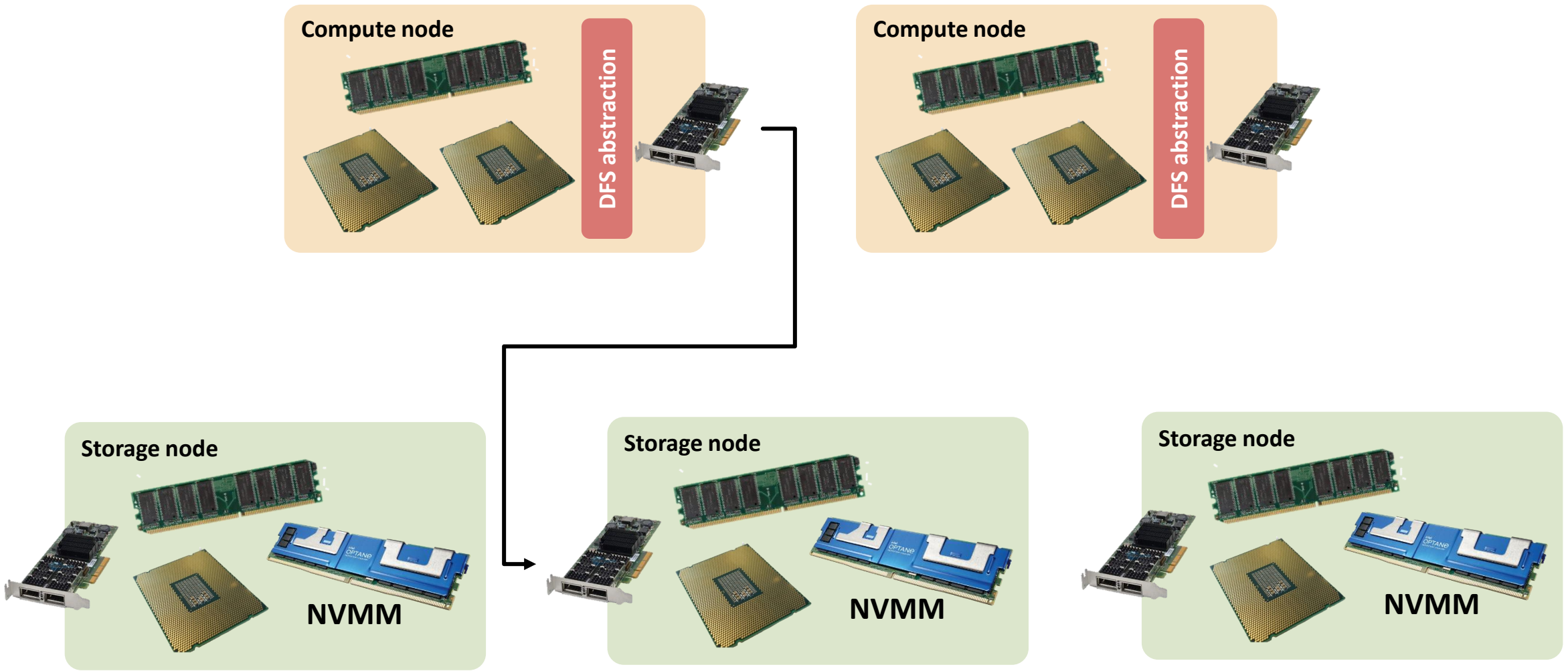
Distributed File Systems



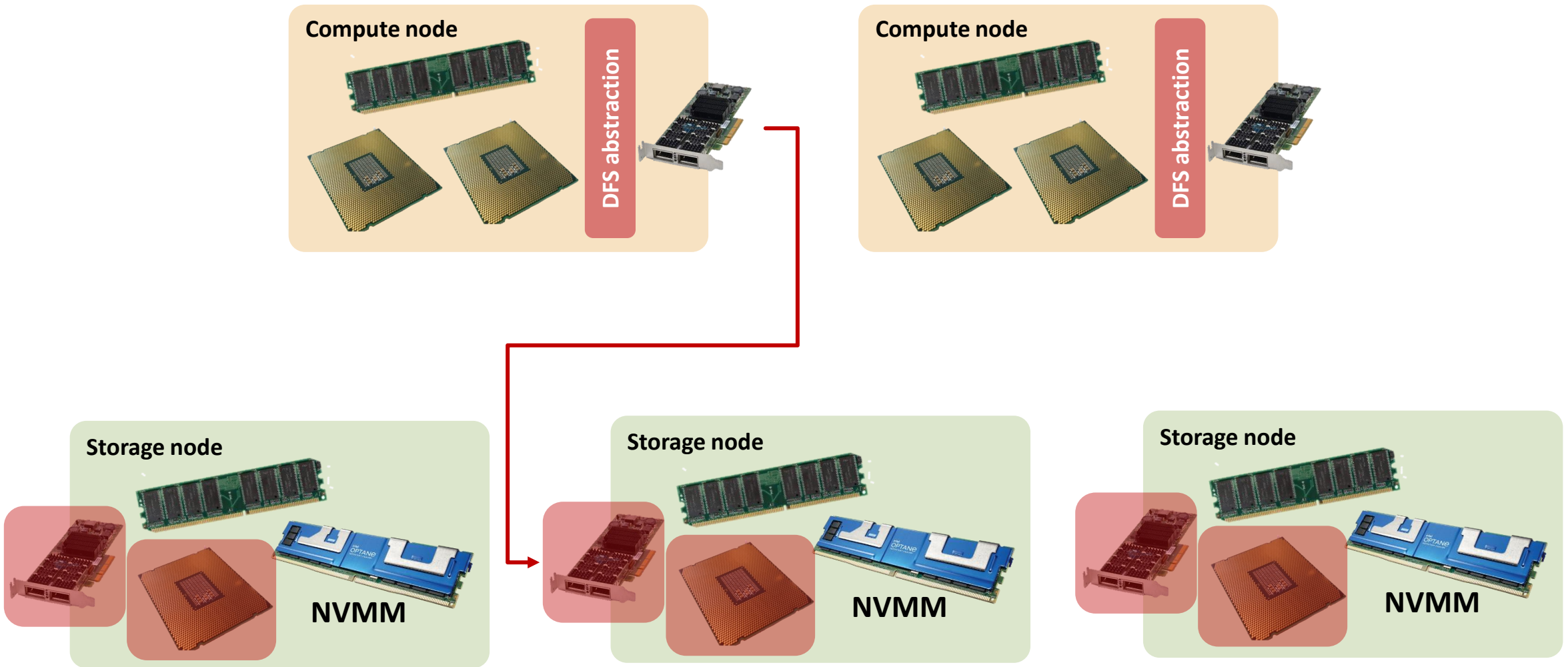
Distributed File Systems



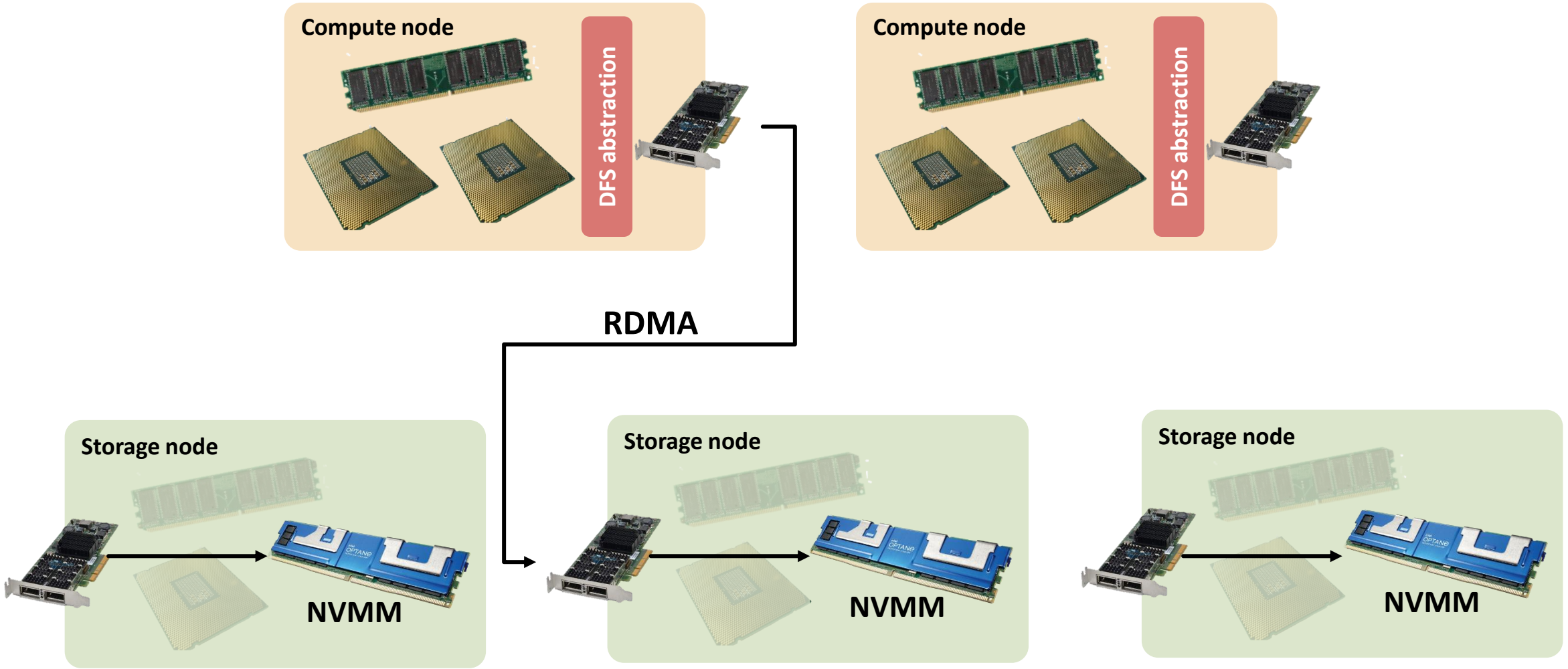
Distributed File Systems with NVMM



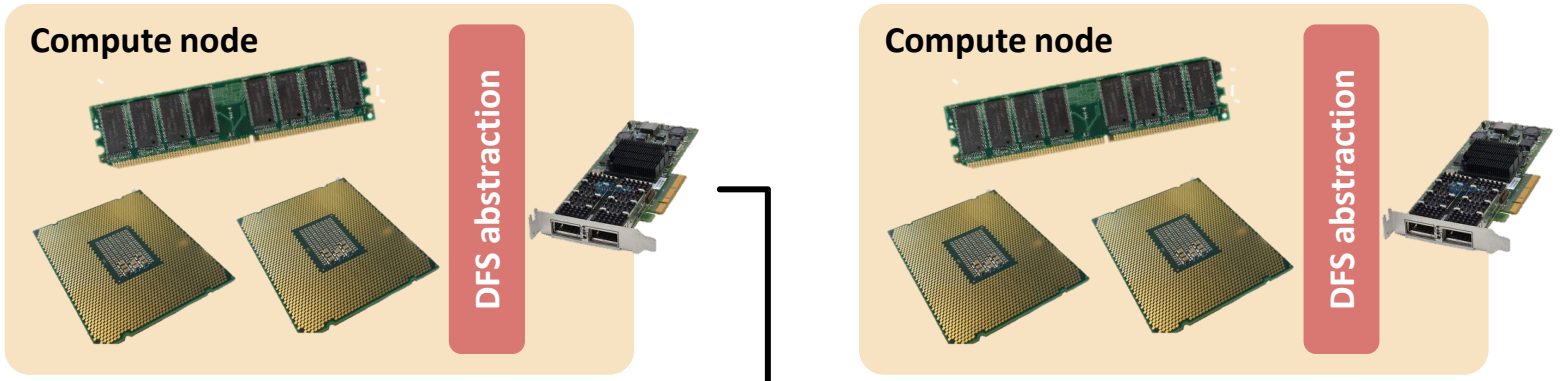
Distributed File Systems with NVMM

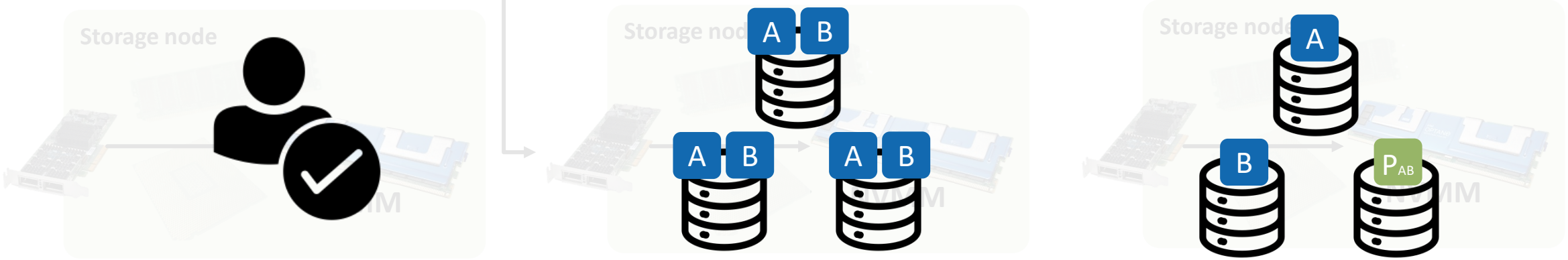


Distributed File Systems with NVMM



Distributed File Systems with NVMM



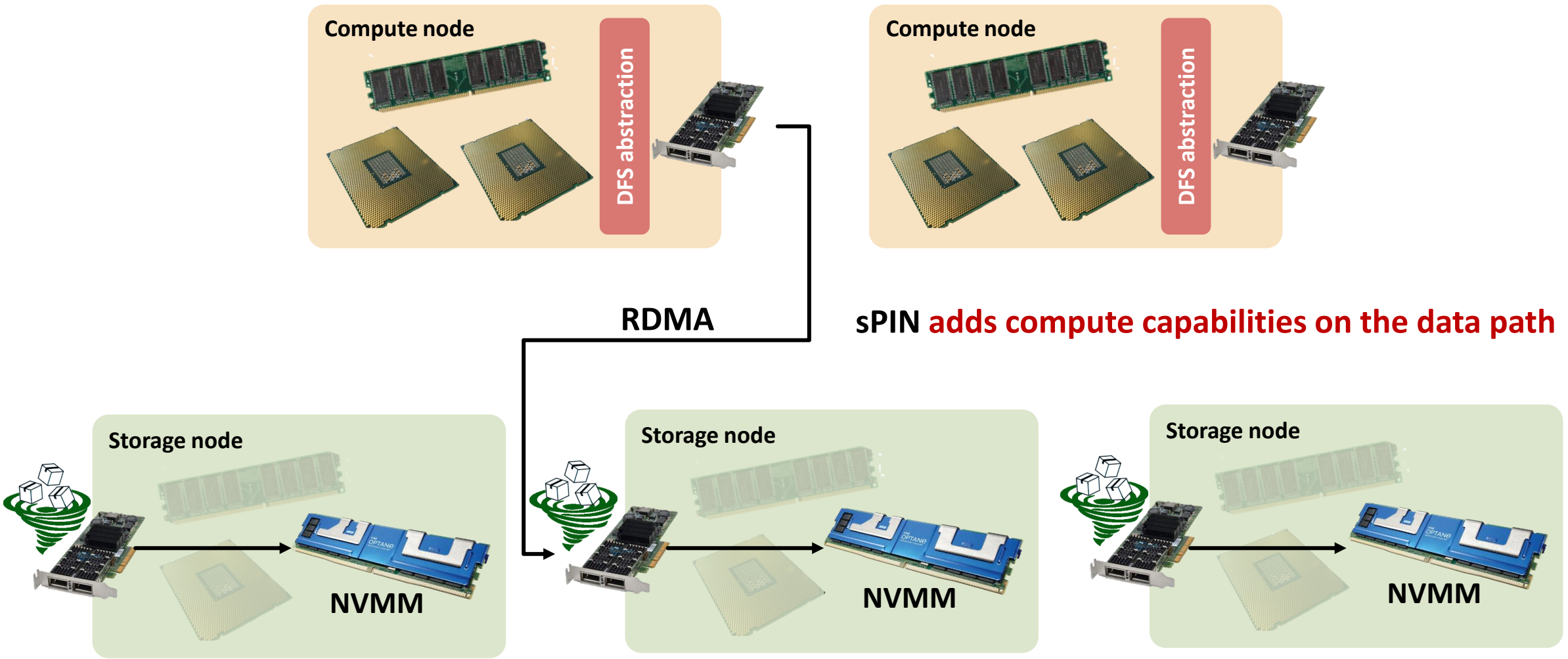


Client request validation

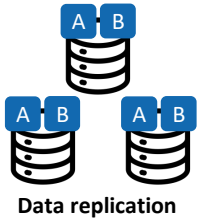
Data replication

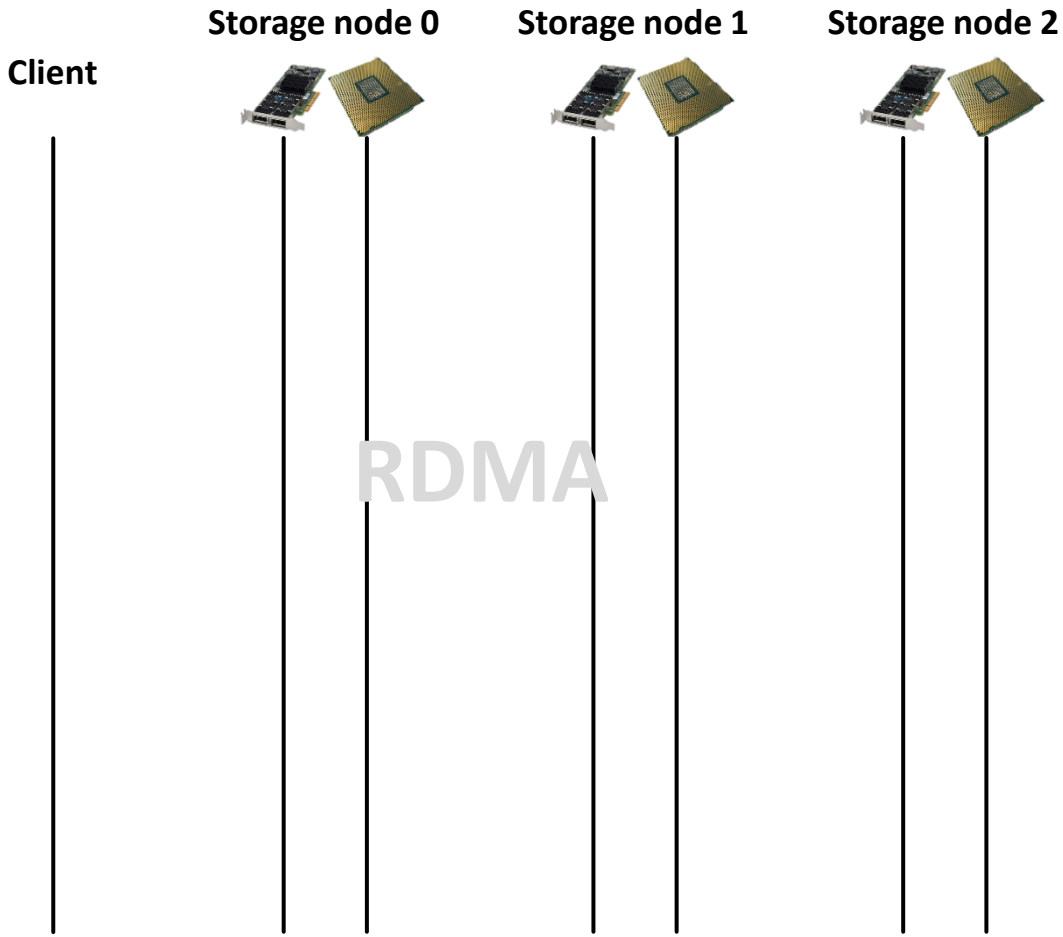
Erasure coding

Distributed File Systems with NVMM and sPIN



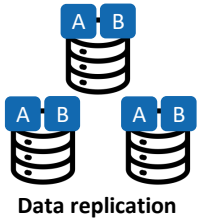
Offloaded data replication

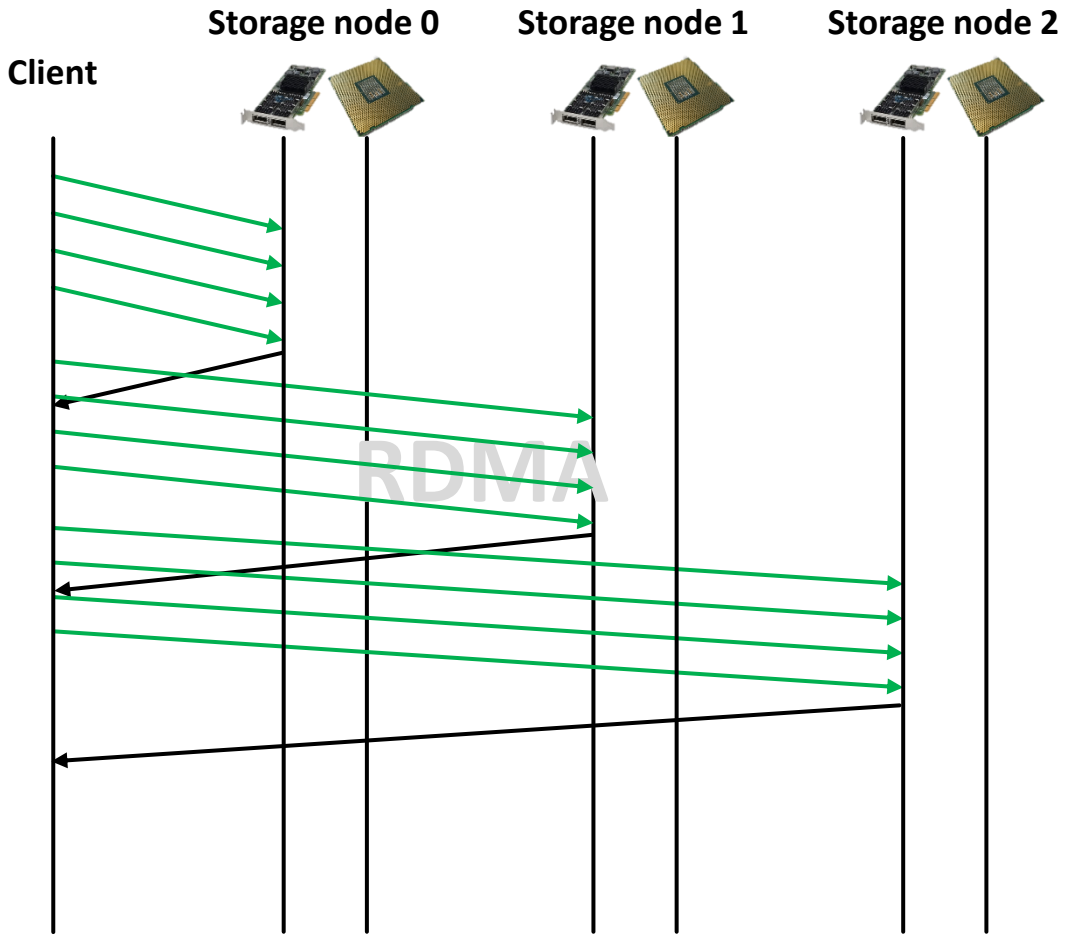




Yang, Jian, Joseph Izraelevitz, and Steven Swanson. "Orion: A distributed file system for non-volatile main memory and RDMA-capable networks." 17th {USENIX} Conference on File and Storage Technologies ({FAST} 19). 2019.

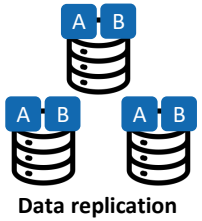
Offloaded data replication

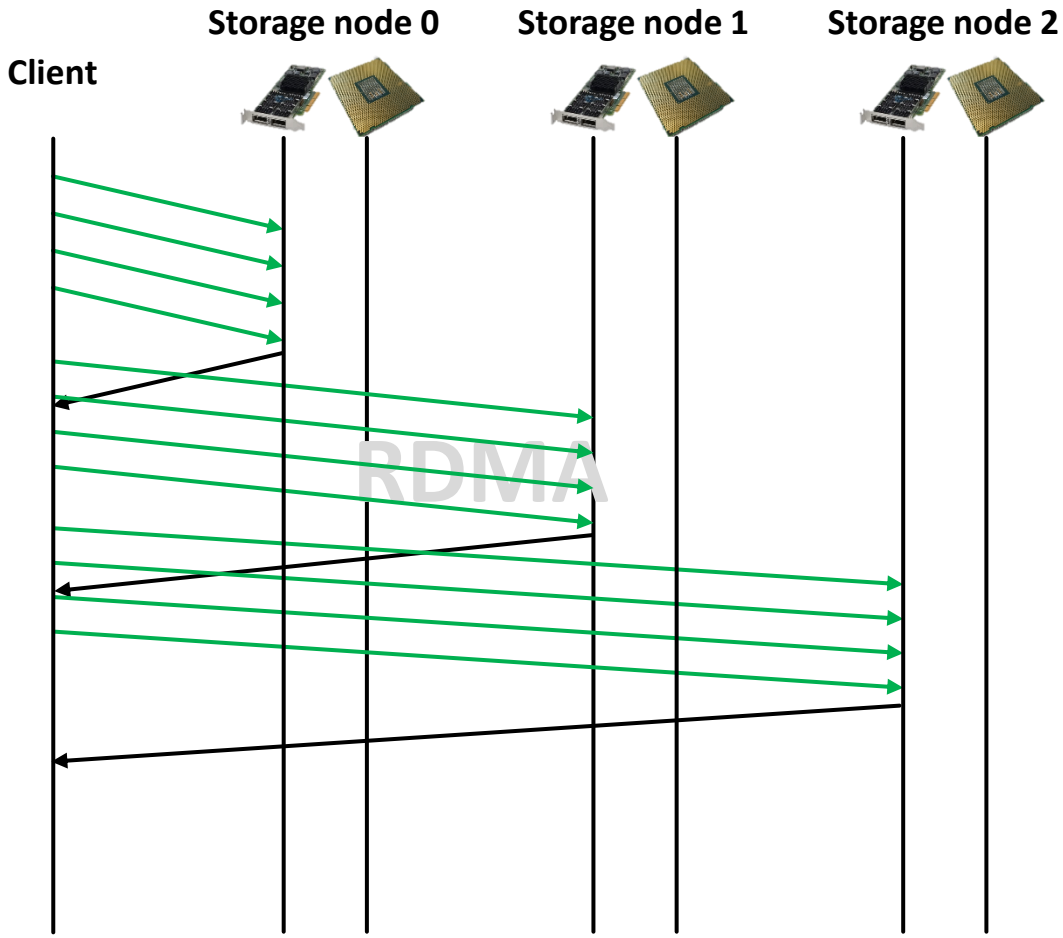


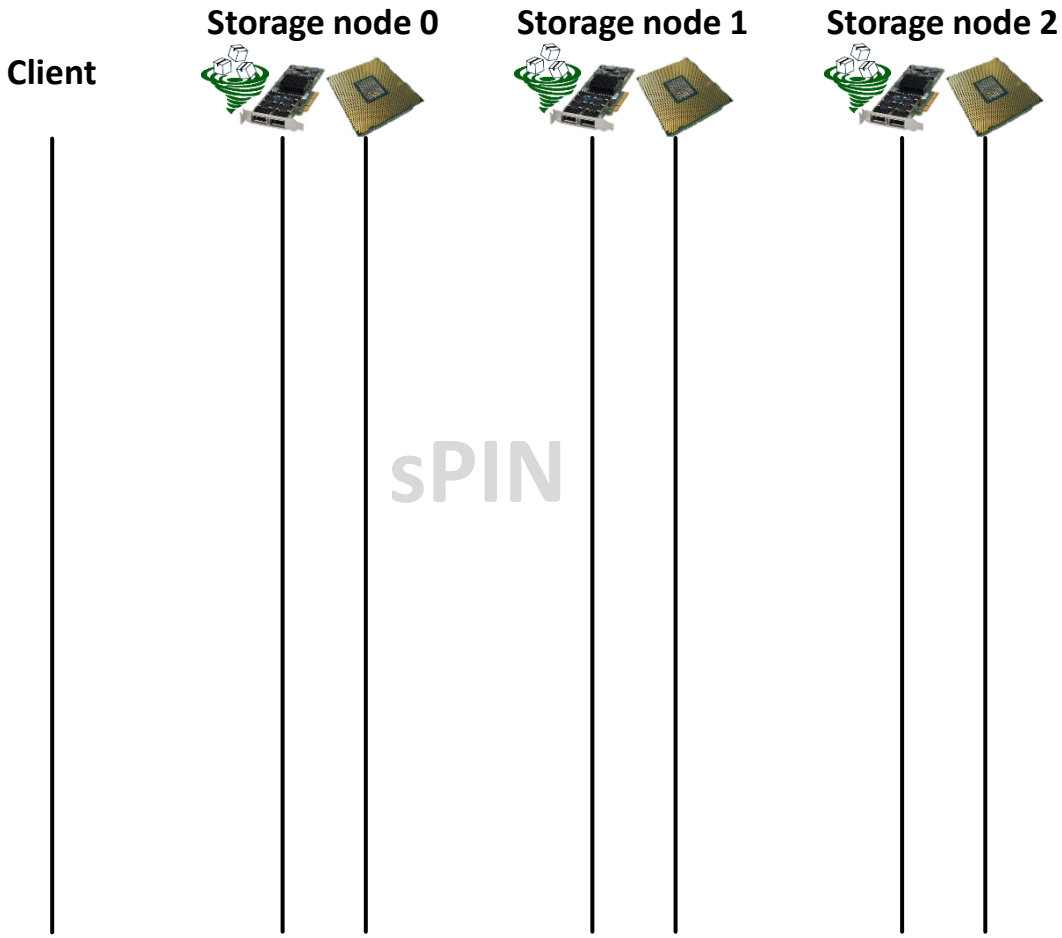


Yang, Jian, Joseph Izraelevitz, and Steven Swanson. "Orion: A distributed file system for non-volatile main memory and RDMA-capable networks." 17th {USENIX} Conference on File and Storage Technologies ({FAST} 19). 2019.

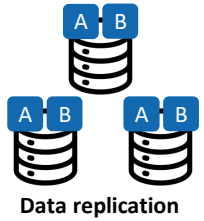
Offloaded data replication

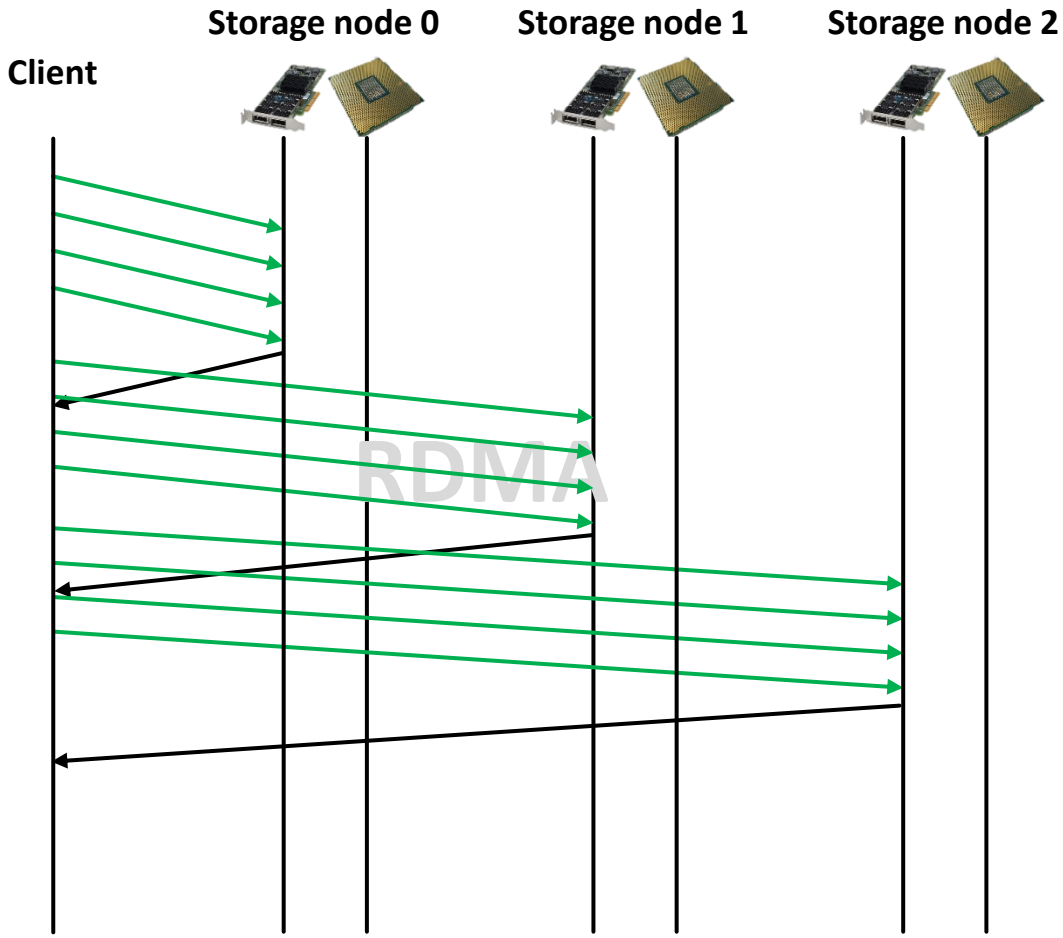


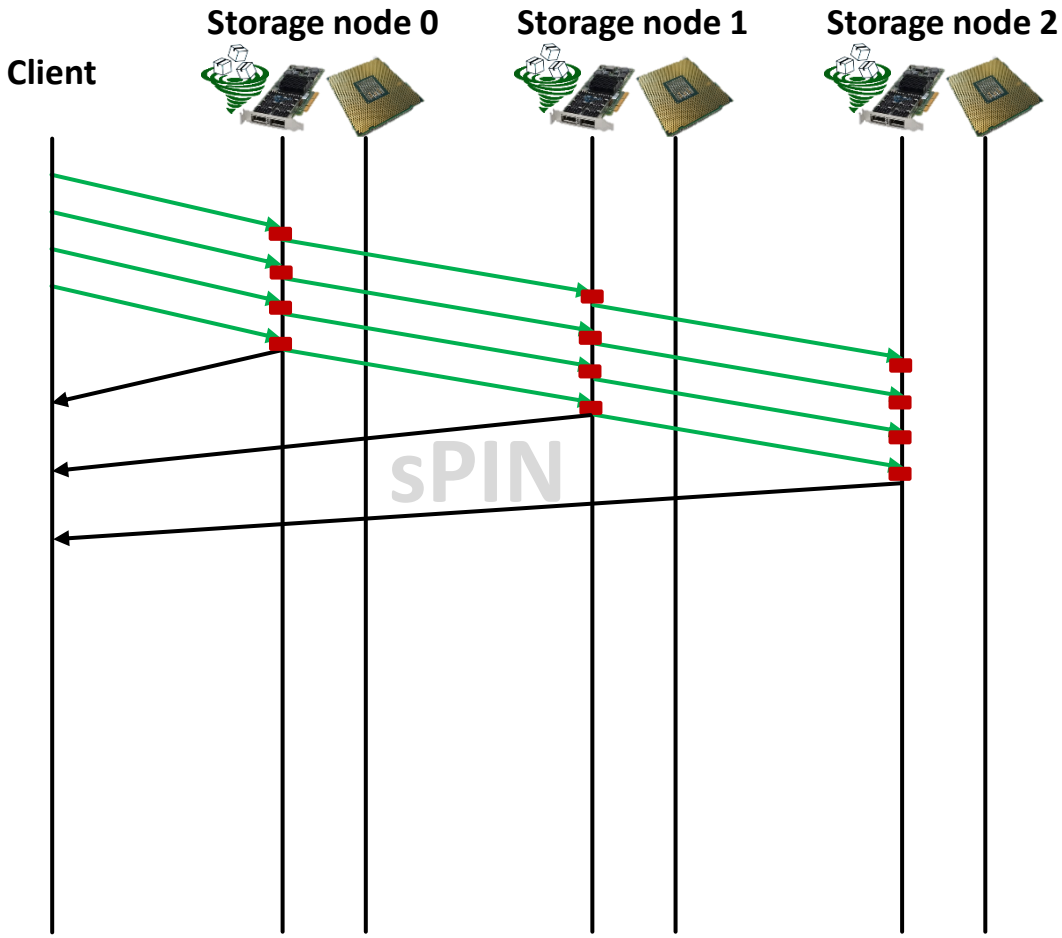




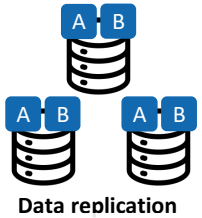
Offloaded data replication

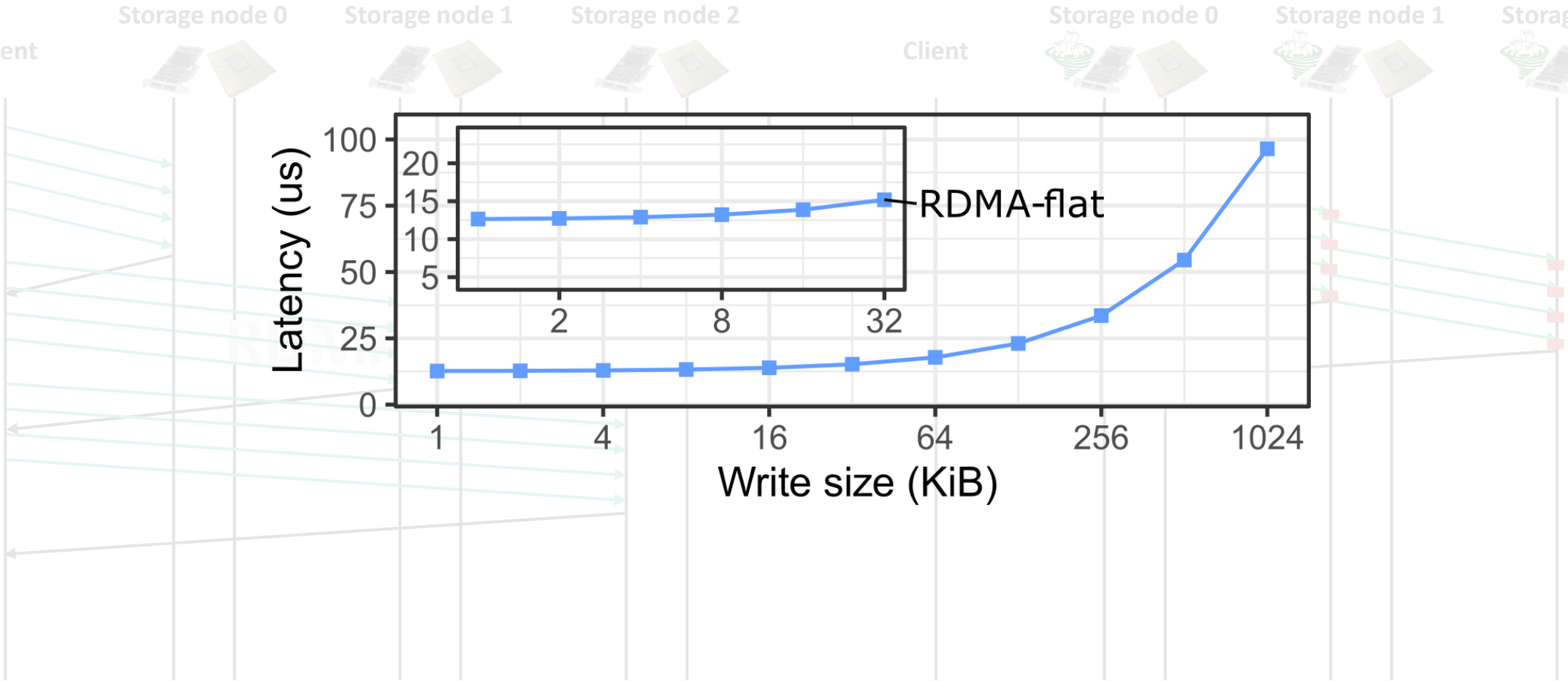




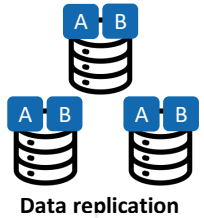


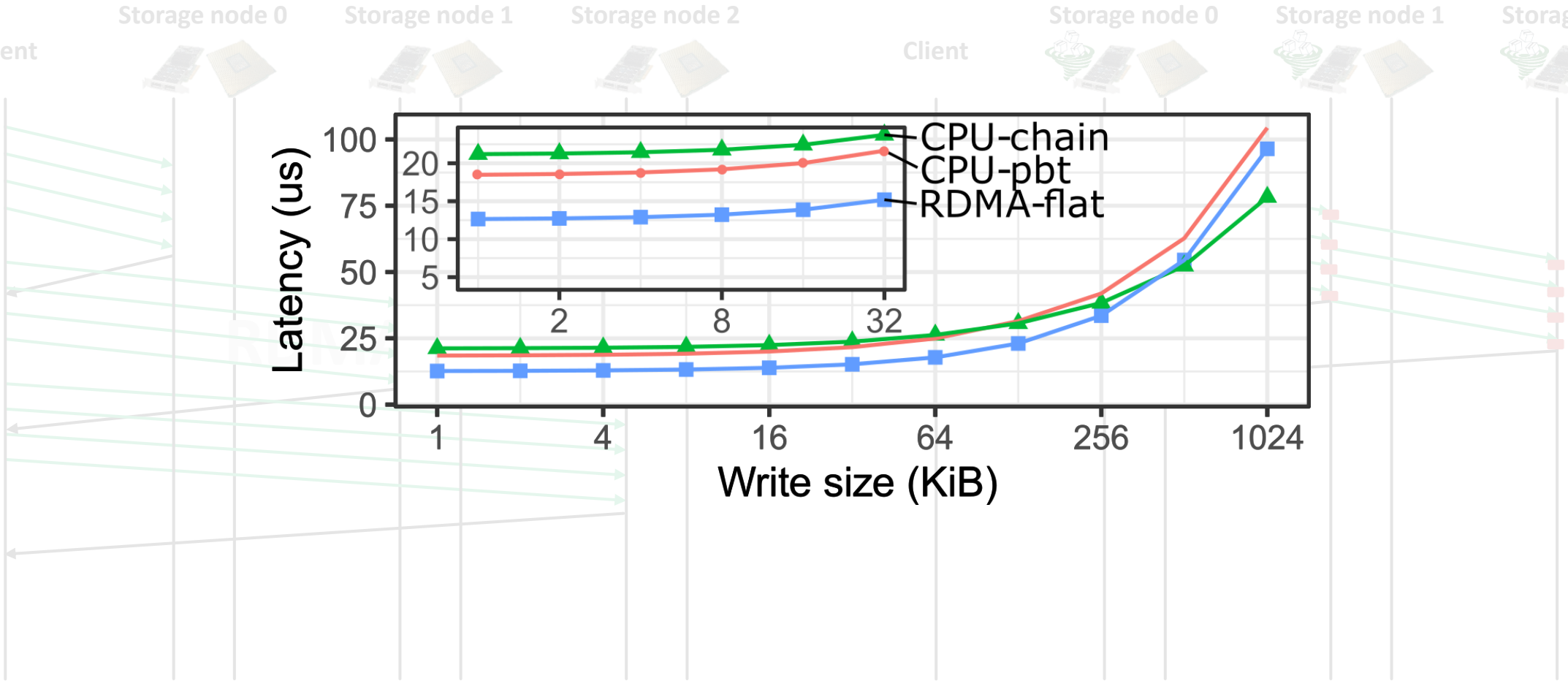
Offloaded data replication



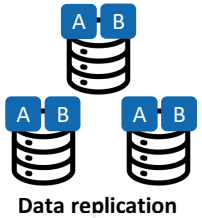


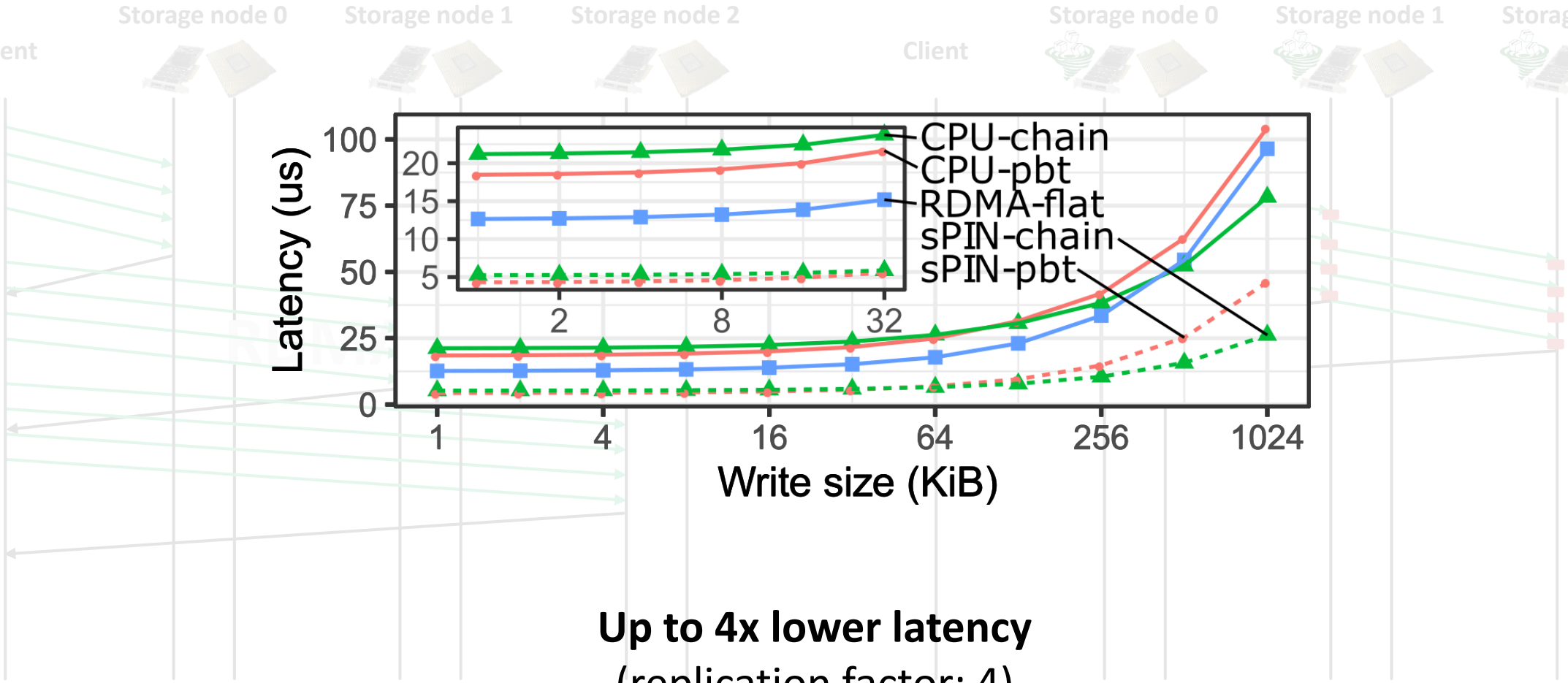
Offloaded data replication



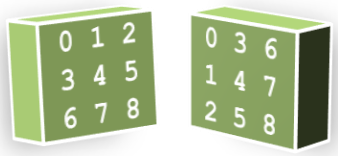


Offloaded data replication

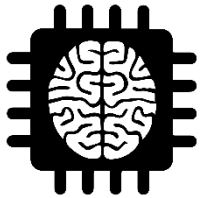




Up to 4x lower latency
(replication factor: 4)



Network-accelerated datatypes

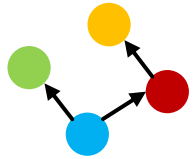


Quantization

Erasure coding

Distributed File Systems

Zoo-sPINNER
consensus on sPIN



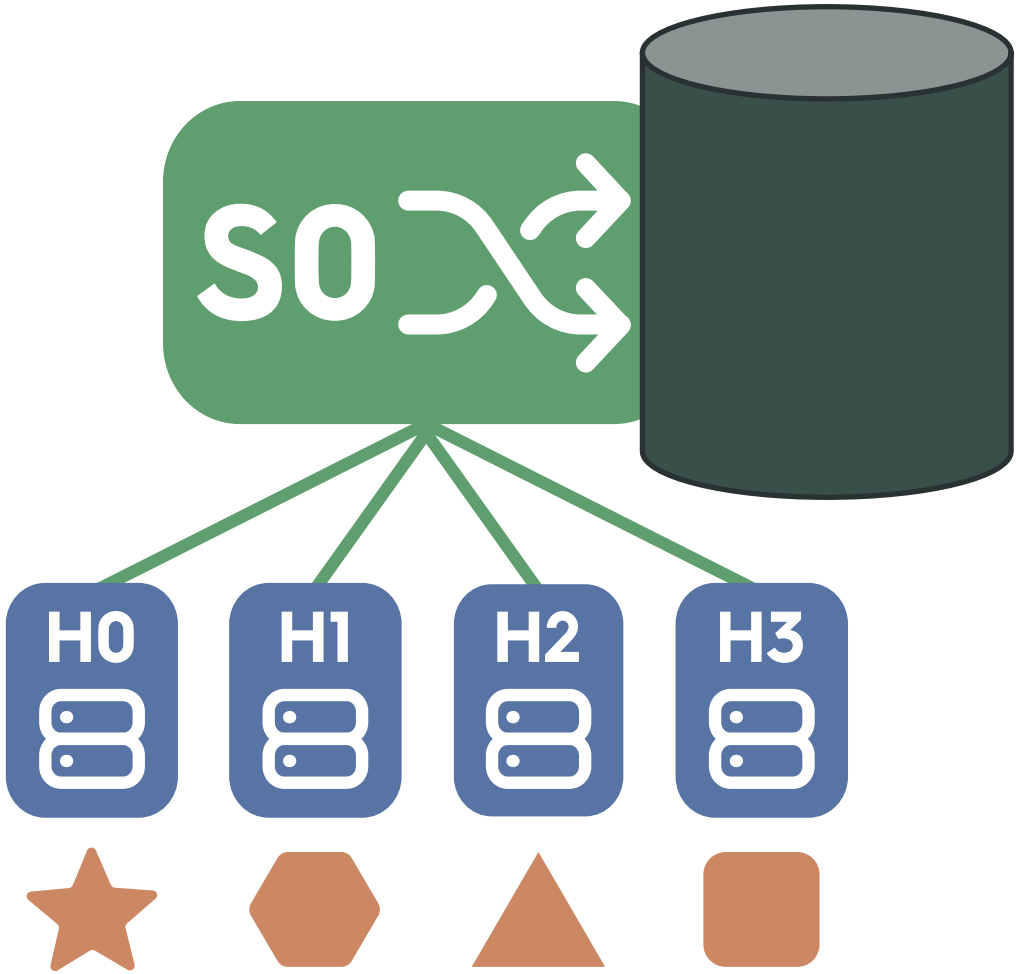
Network Group Communication

Packet classification and pattern matching

In-network allreduce

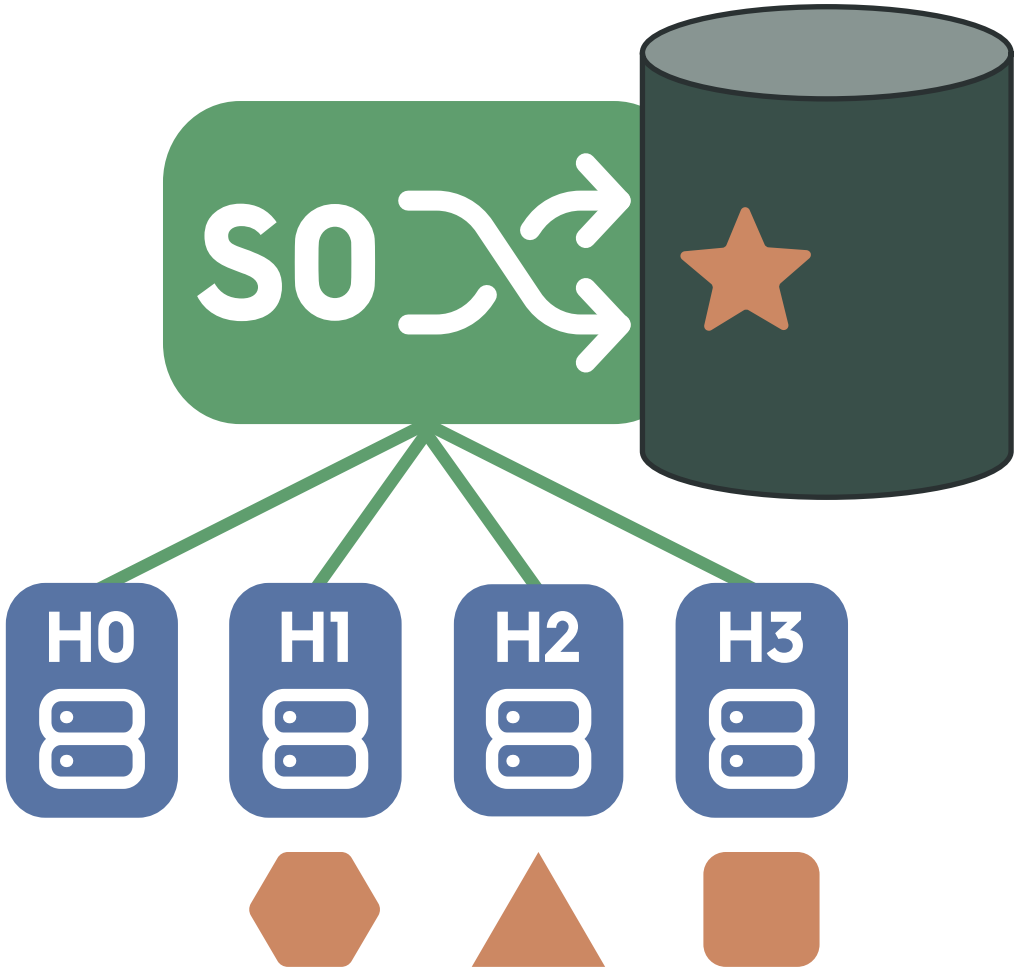
Serverless sPIN

In-network allreduce



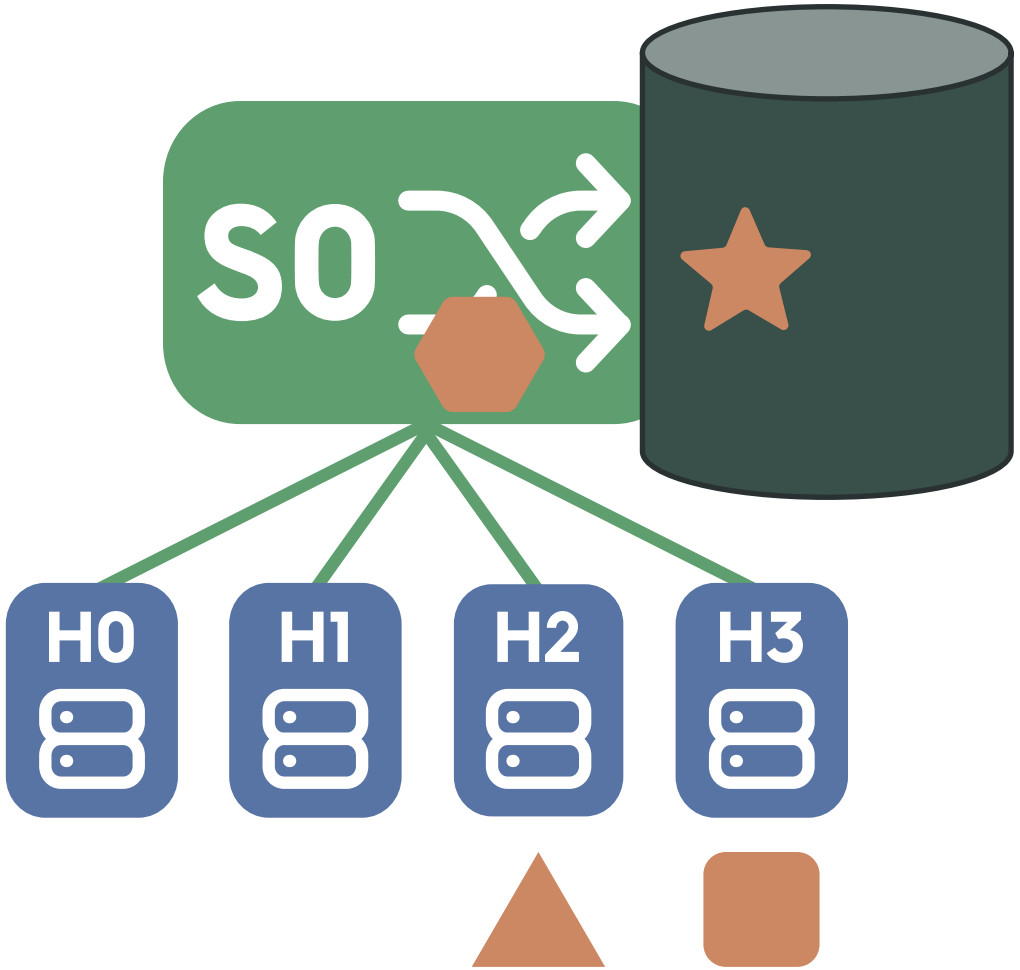
Daniele De Sensi, Salvatore Di Girolamo, Saleh Ashkboos, Shigang Li, and Torsten Hoefler. "Flare: flexible in-network allreduce." SC '21

In-network allreduce



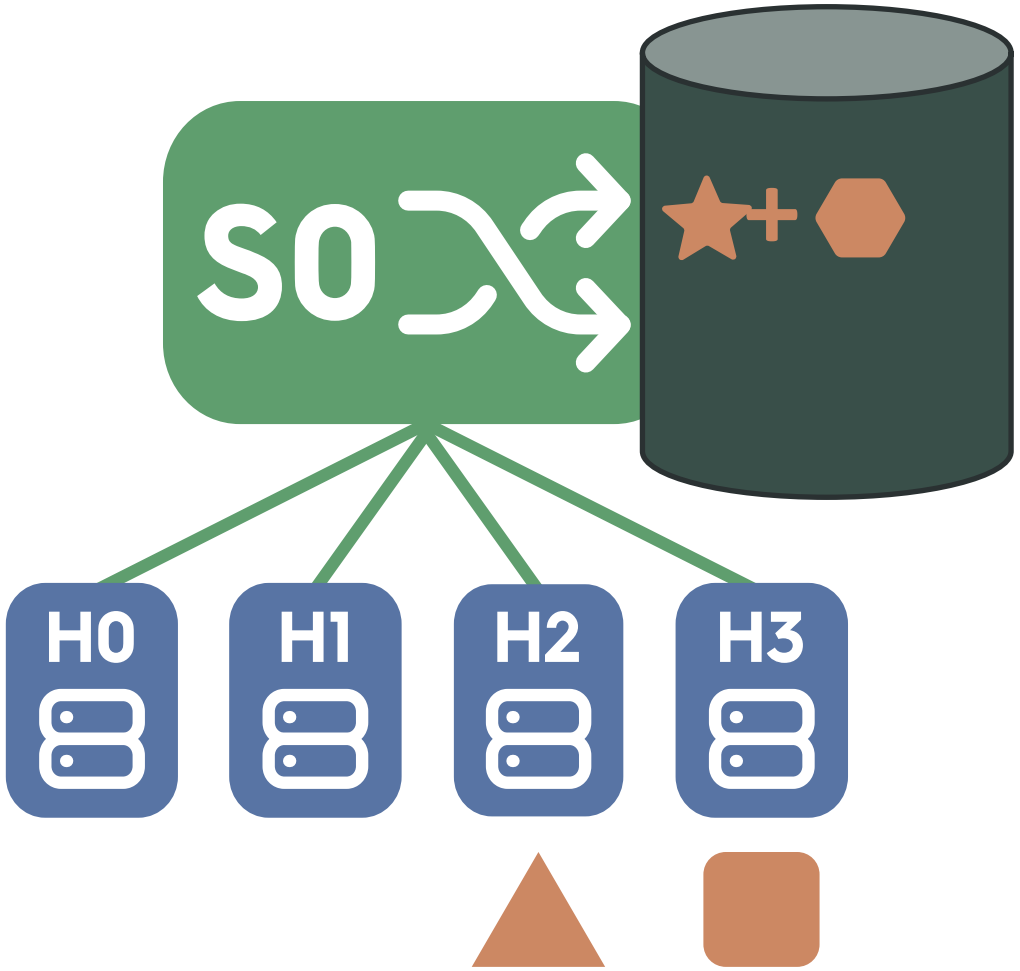
Daniele De Sensi, Salvatore Di Girolamo, Saleh Ashkboos, Shigang Li, and Torsten Hoefler. "Flare: flexible in-network allreduce." SC '21

In-network allreduce



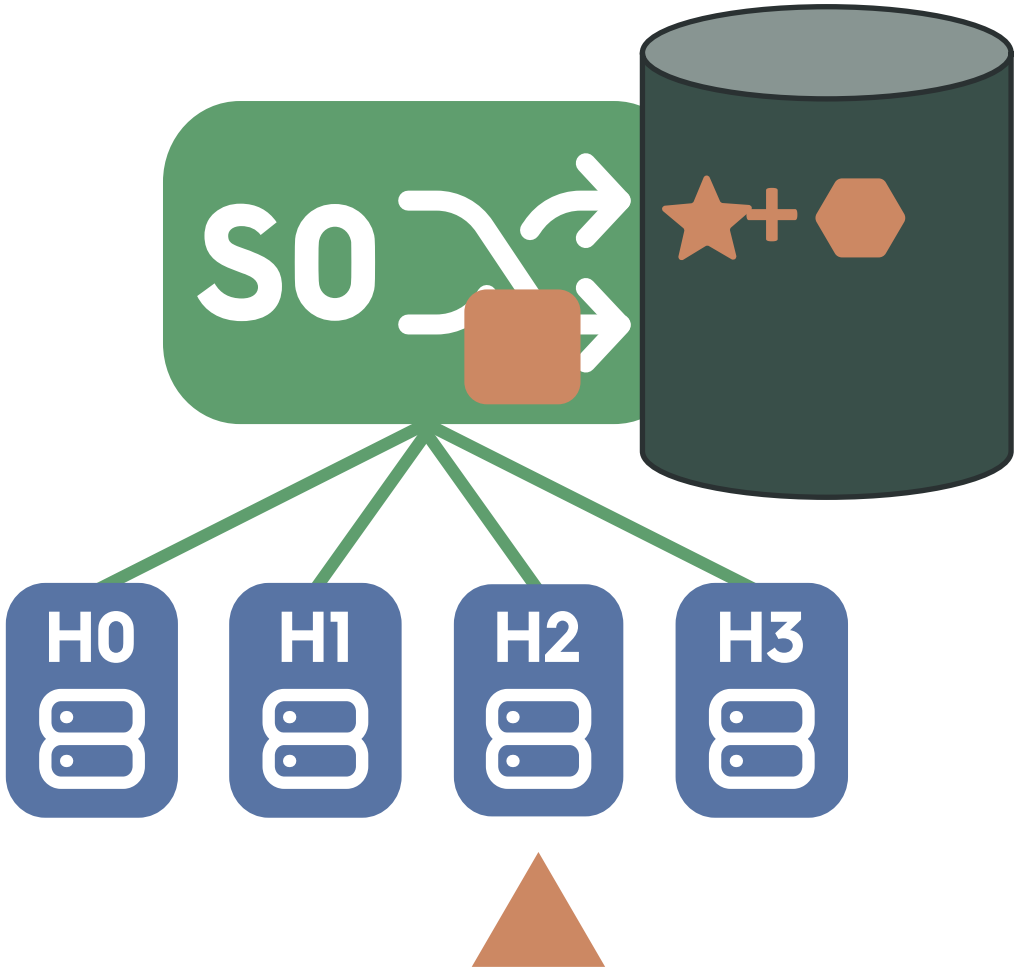
Daniele De Sensi, Salvatore Di Girolamo, Saleh Ashkboos, Shigang Li, and Torsten Hoefler. "Flare: flexible in-network allreduce." SC '21

In-network allreduce



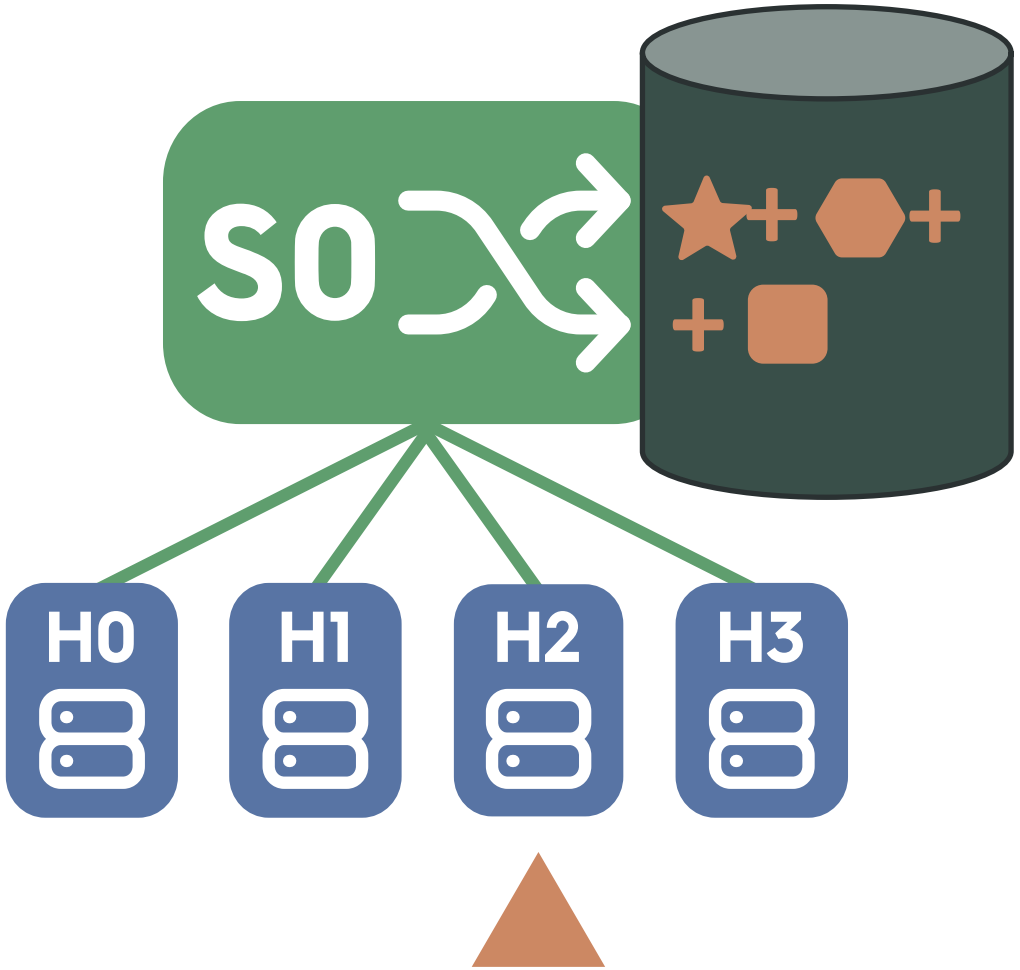
Daniele De Sensi, Salvatore Di Girolamo, Saleh Ashkboos, Shigang Li, and Torsten Hoefler. "Flare: flexible in-network allreduce." SC '21

In-network allreduce



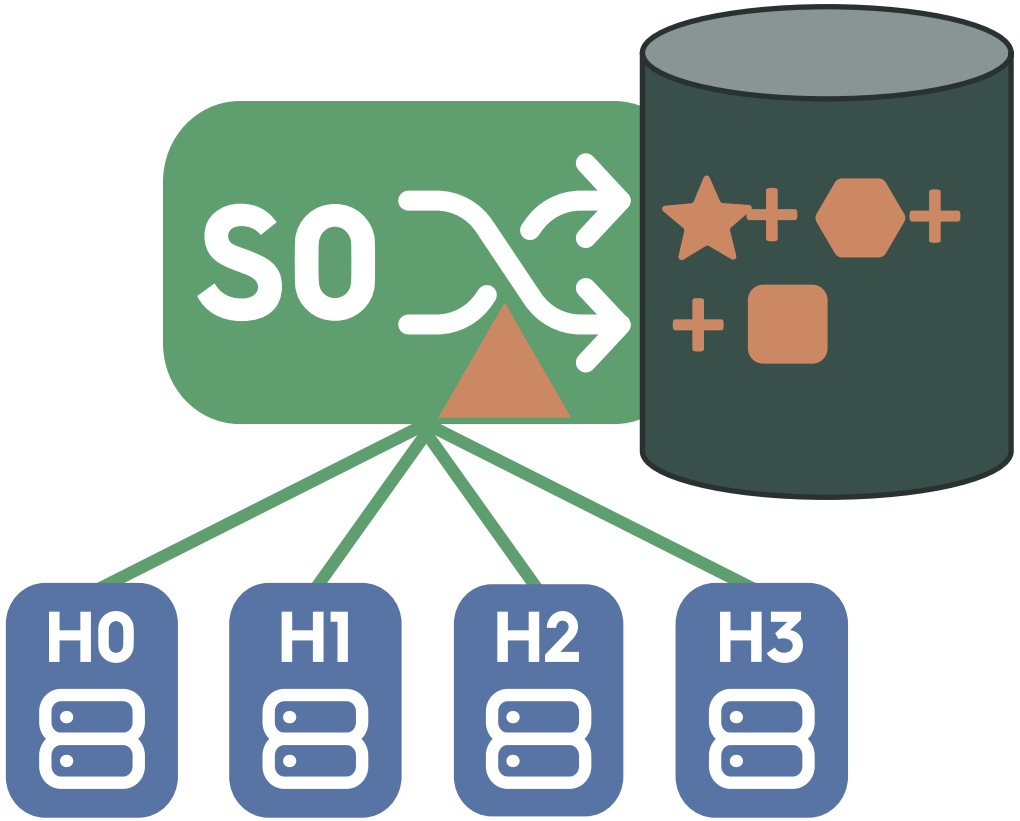
Daniele De Sensi, Salvatore Di Girolamo, Saleh Ashkboos, Shigang Li, and Torsten Hoefler. "Flare: flexible in-network allreduce." SC '21

In-network allreduce



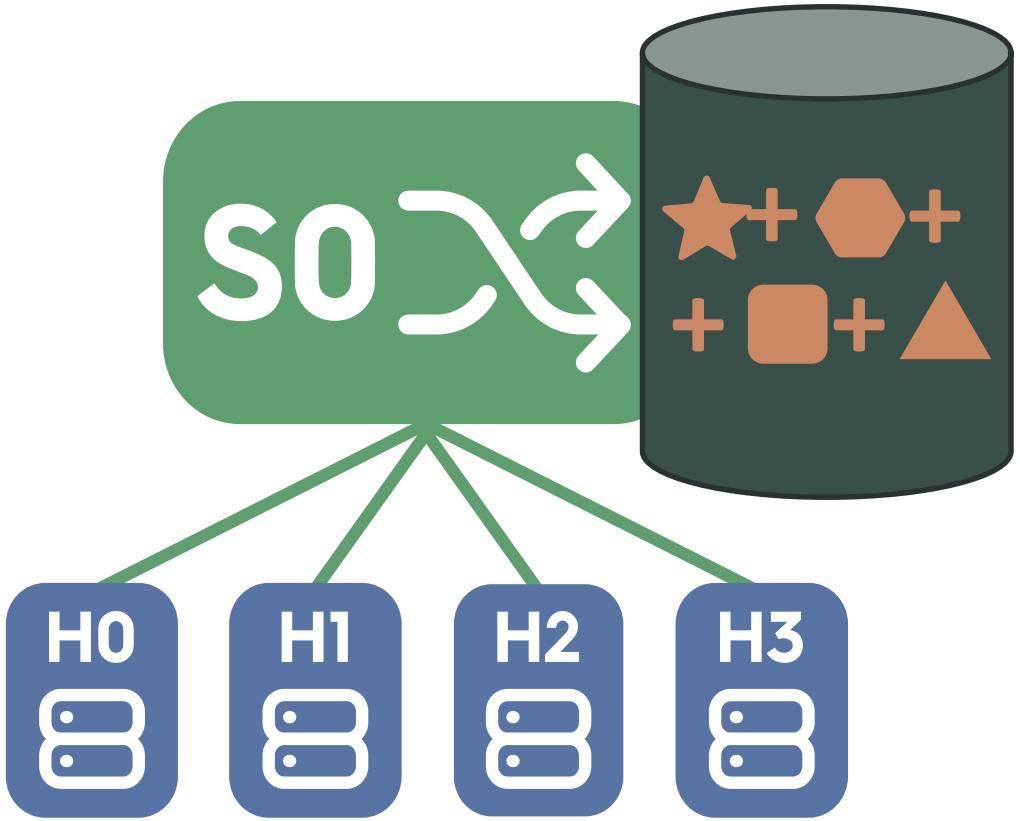
Daniele De Sensi, Salvatore Di Girolamo, Saleh Ashkboos, Shigang Li, and Torsten Hoefler. "Flare: flexible in-network allreduce." SC '21

In-network allreduce



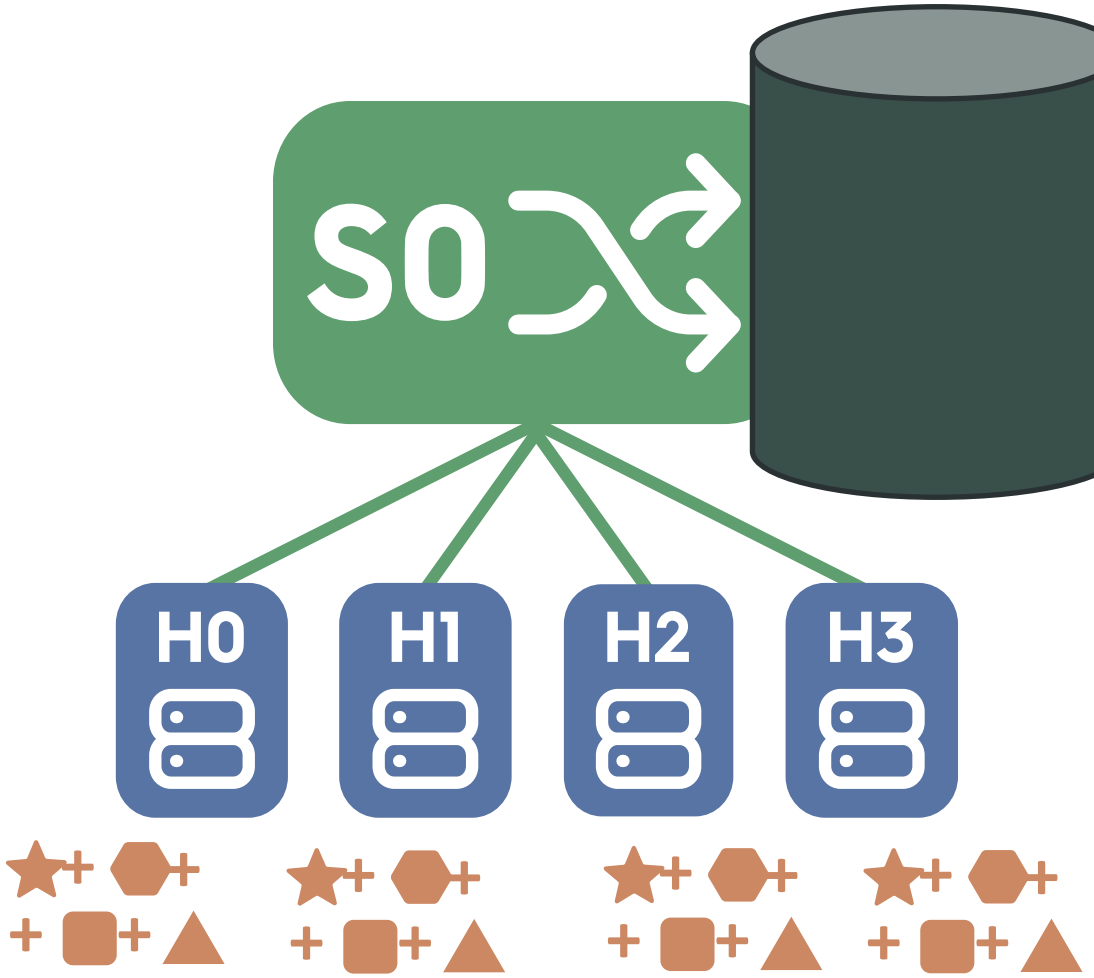
Daniele De Sensi, Salvatore Di Girolamo, Saleh Ashkboos, Shigang Li, and Torsten Hoefler. "Flare: flexible in-network allreduce." SC '21

In-network allreduce



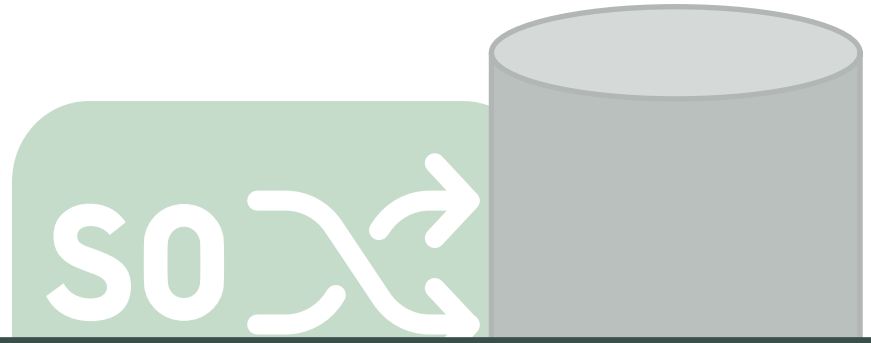
Daniele De Sensi, Salvatore Di Girolamo, Saleh Ashkboos, Shigang Li, and Torsten Hoefler. "Flare: flexible in-network allreduce." SC '21

In-network allreduce

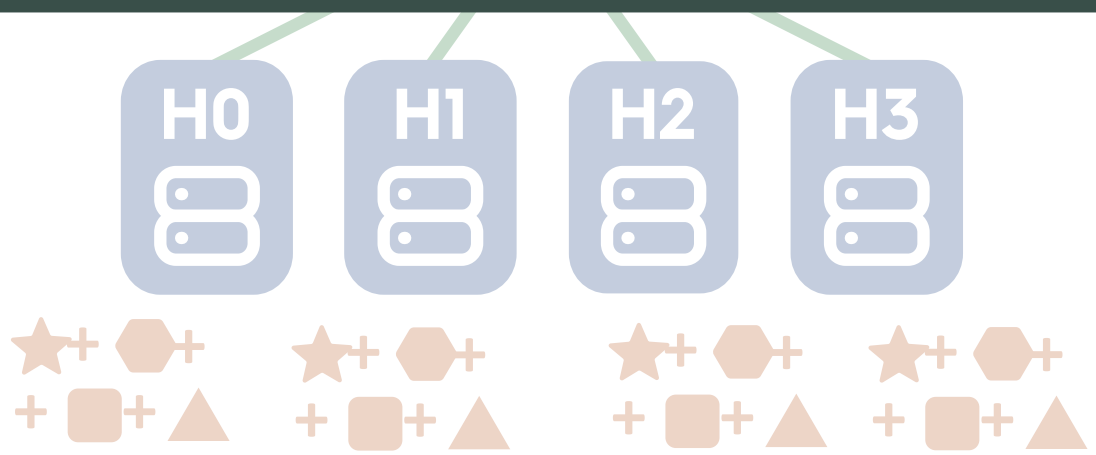


Daniele De Sensi, Salvatore Di Girolamo, Saleh Ashkboos, Shigang Li, and Torsten Hoefler. "Flare: flexible in-network allreduce." SC '21

In-network allreduce

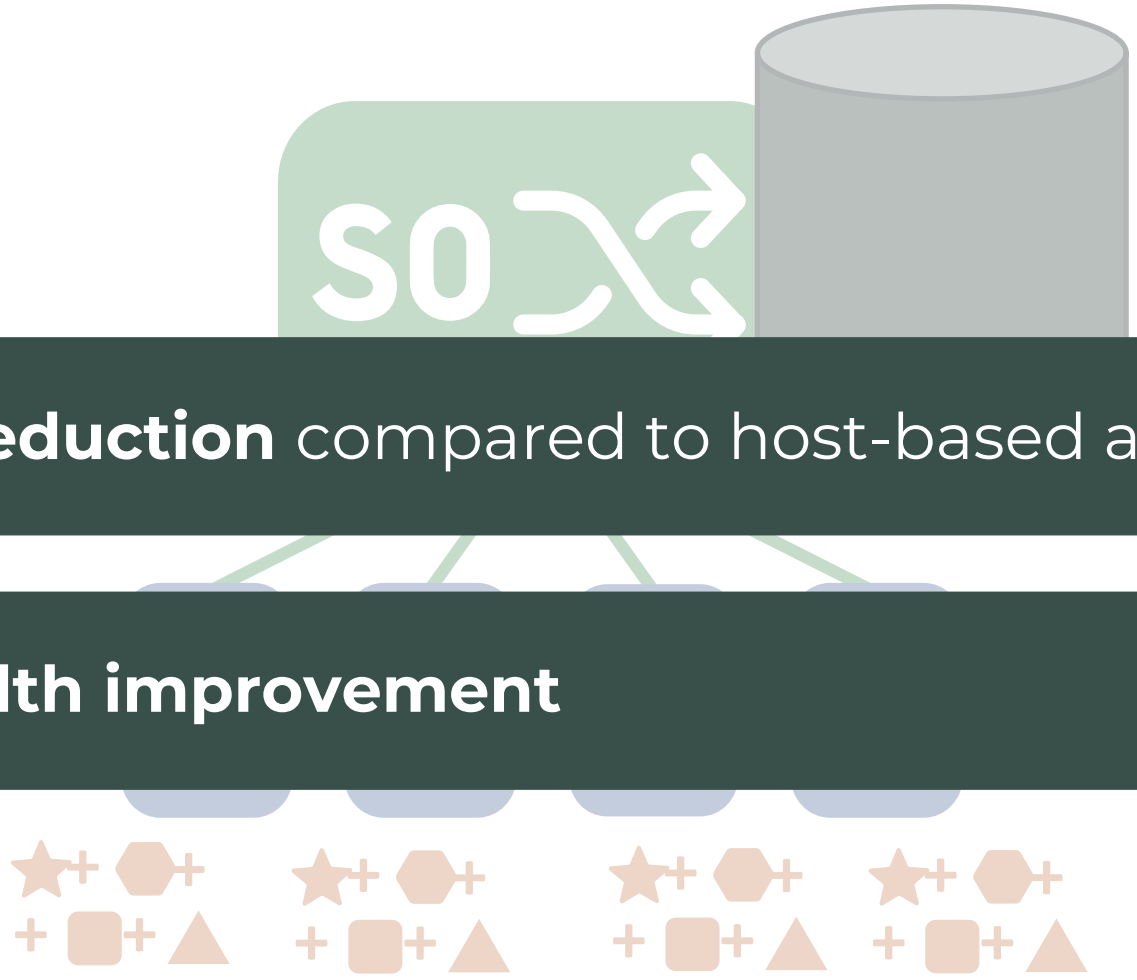


2x traffic reduction compared to host-based allreduce



Daniele De Sensi, Salvatore Di Girolamo, Saleh Ashkboos, Shigang Li, and Torsten Hoefler. "Flare: flexible in-network allreduce." SC '21

In-network allreduce



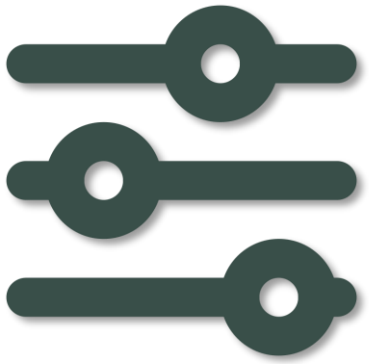
2x traffic reduction compared to host-based allreduce

2x bandwidth improvement

Daniele De Sensi, Salvatore Di Girolamo, Saleh Ashkboos, Shigang Li, and Torsten Hoefler. "Flare: flexible in-network allreduce." SC '21

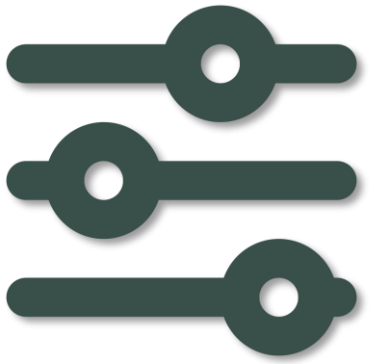
Missing features

Missing features



Custom
operators and
datatypes

Missing features



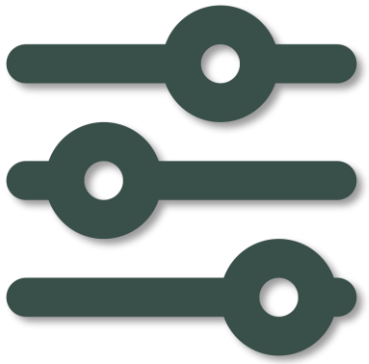
Custom **operators** and **datatypes**

int4

float16

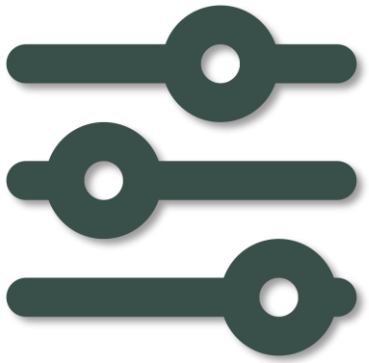
int8

Missing features

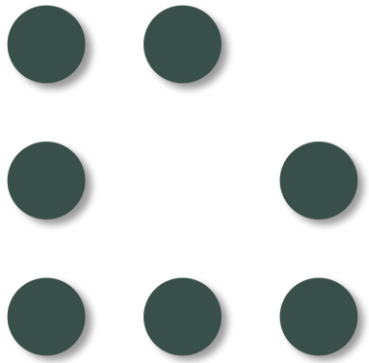


Custom
operators and
datatypes

Missing features

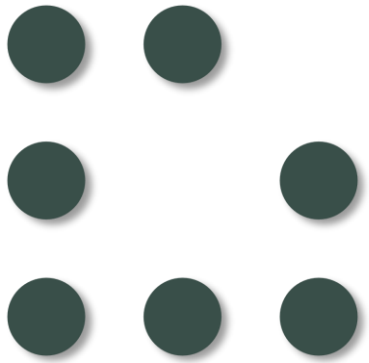


Custom **operators** and **datatypes**



Support for **sparse data**

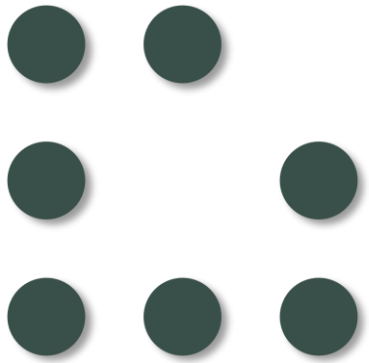
Missing features



2
0
8
0

Support for
sparse data

Missing features

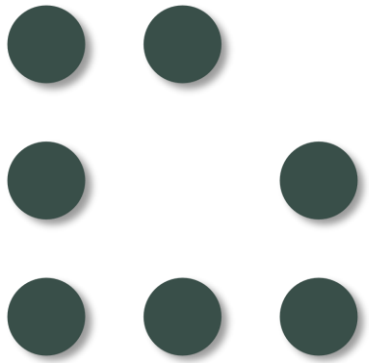


2

8

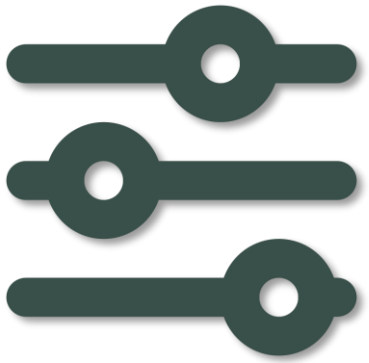
Support for
sparse data

Missing features

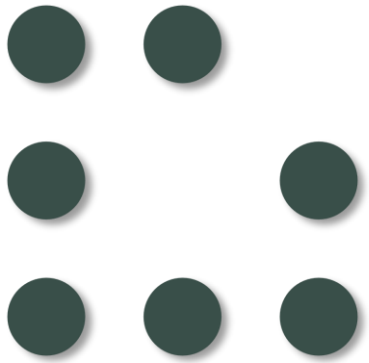


Support for
sparse data

Missing features



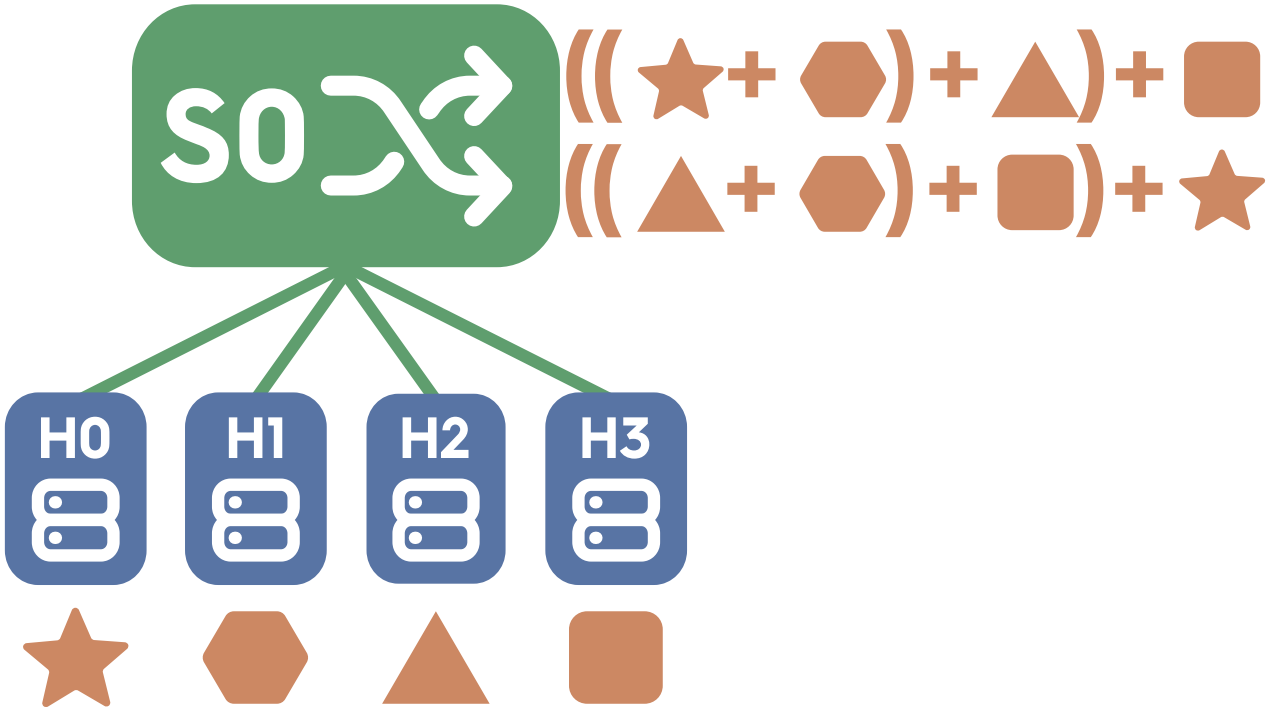
Custom **operators** and **datatypes**



Support for **sparse data**

Reproducibility

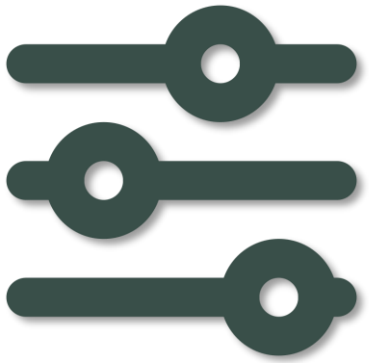
Missing features



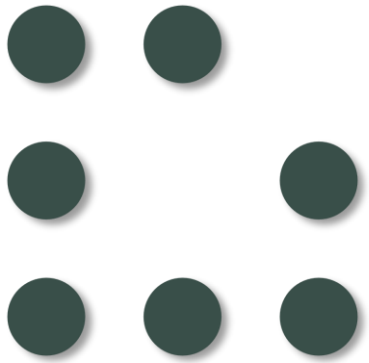
.10
.01

Reproducibility

Missing features



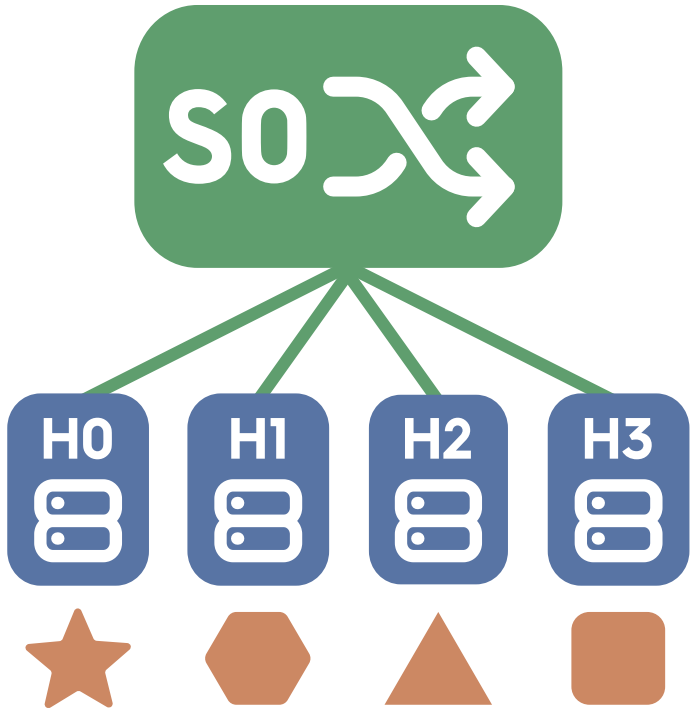
Custom **operators** and **datatypes**



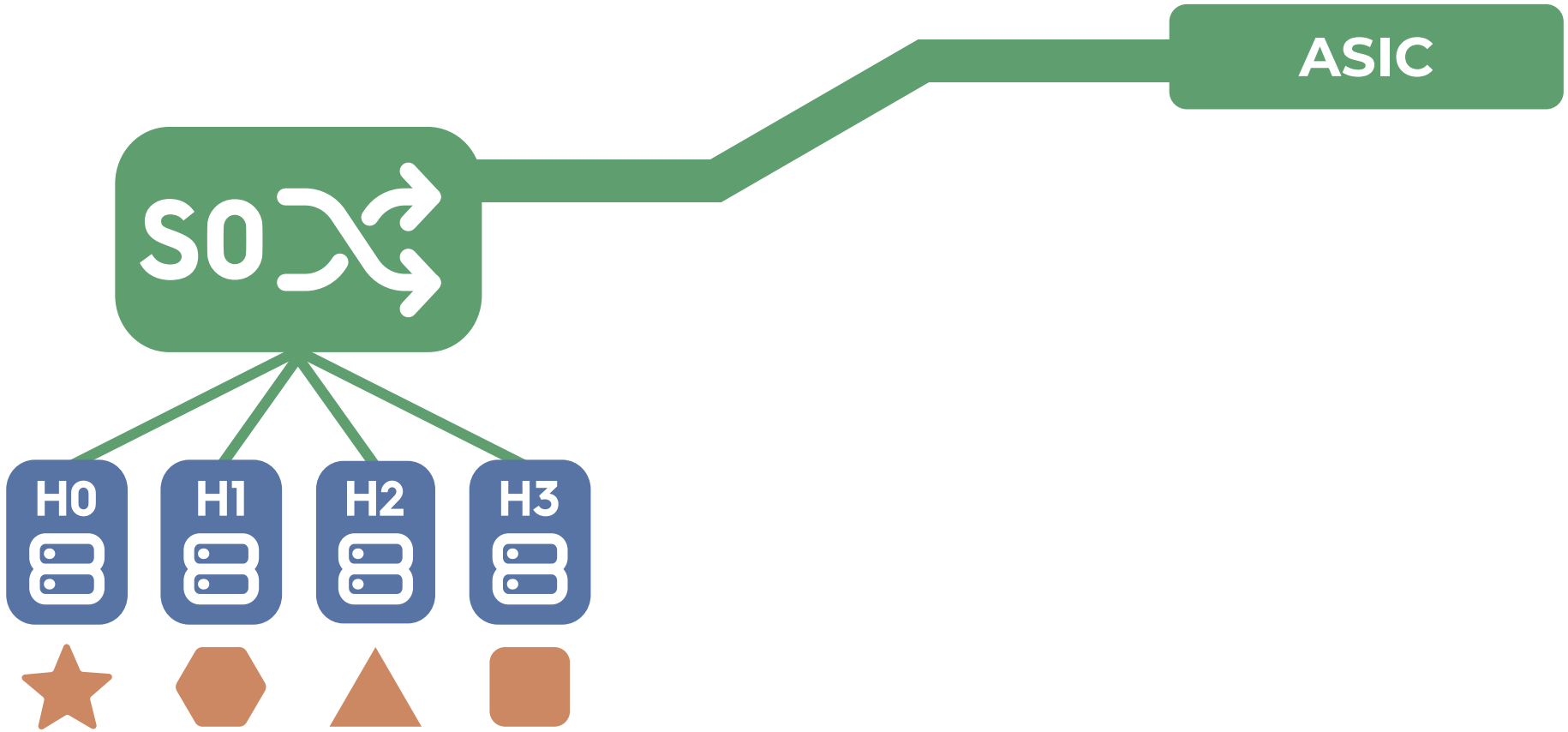
Support for **sparse data**

Reproducibility

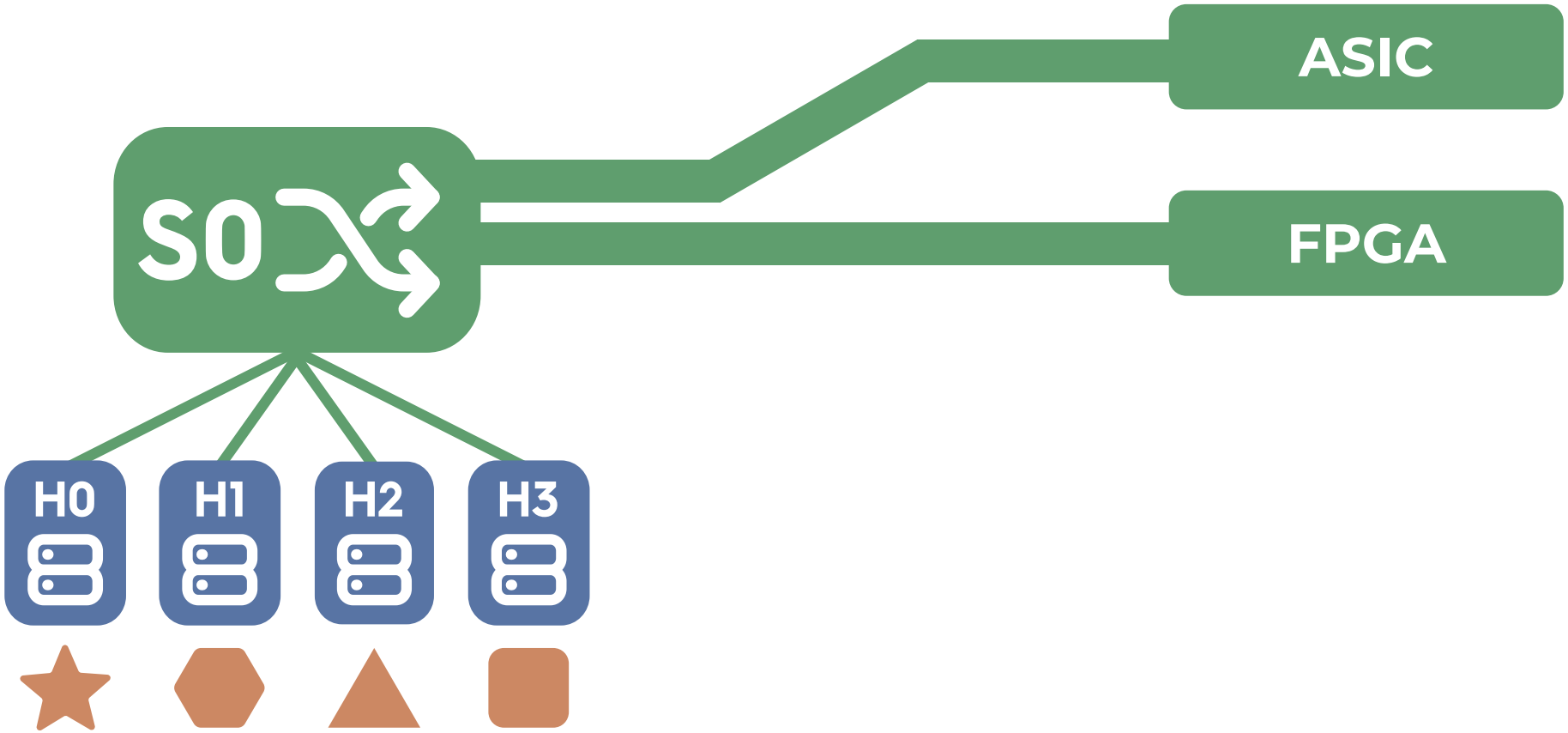
Existing switches architectures



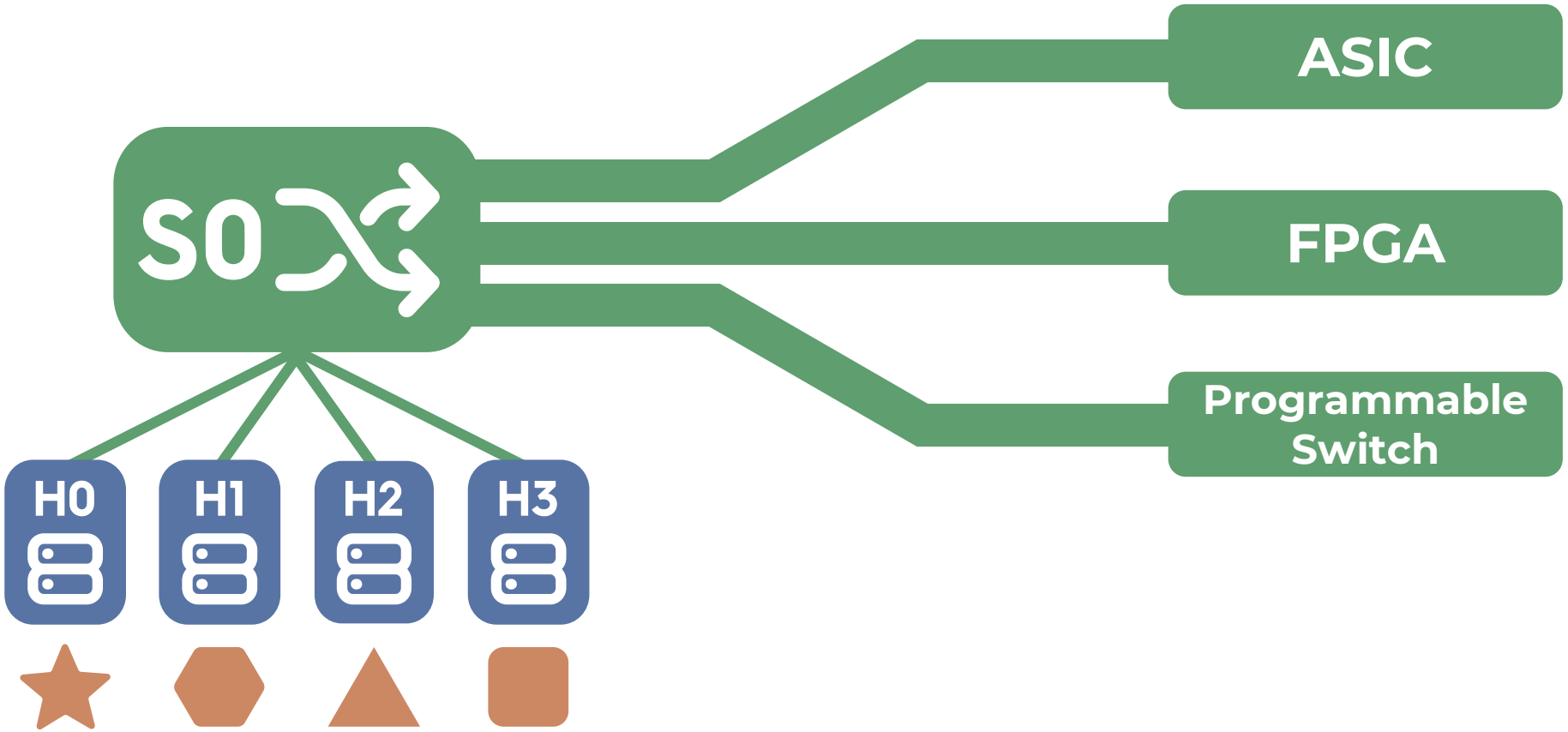
Existing switches architectures



Existing switches architectures

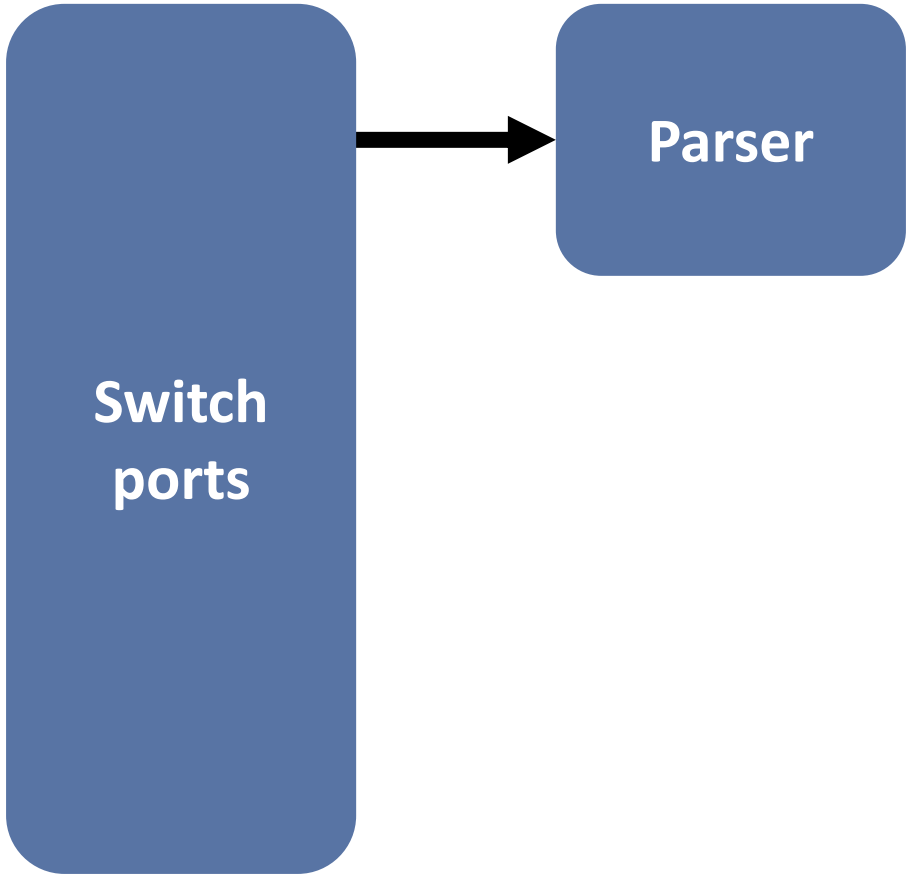


Existing switches architectures

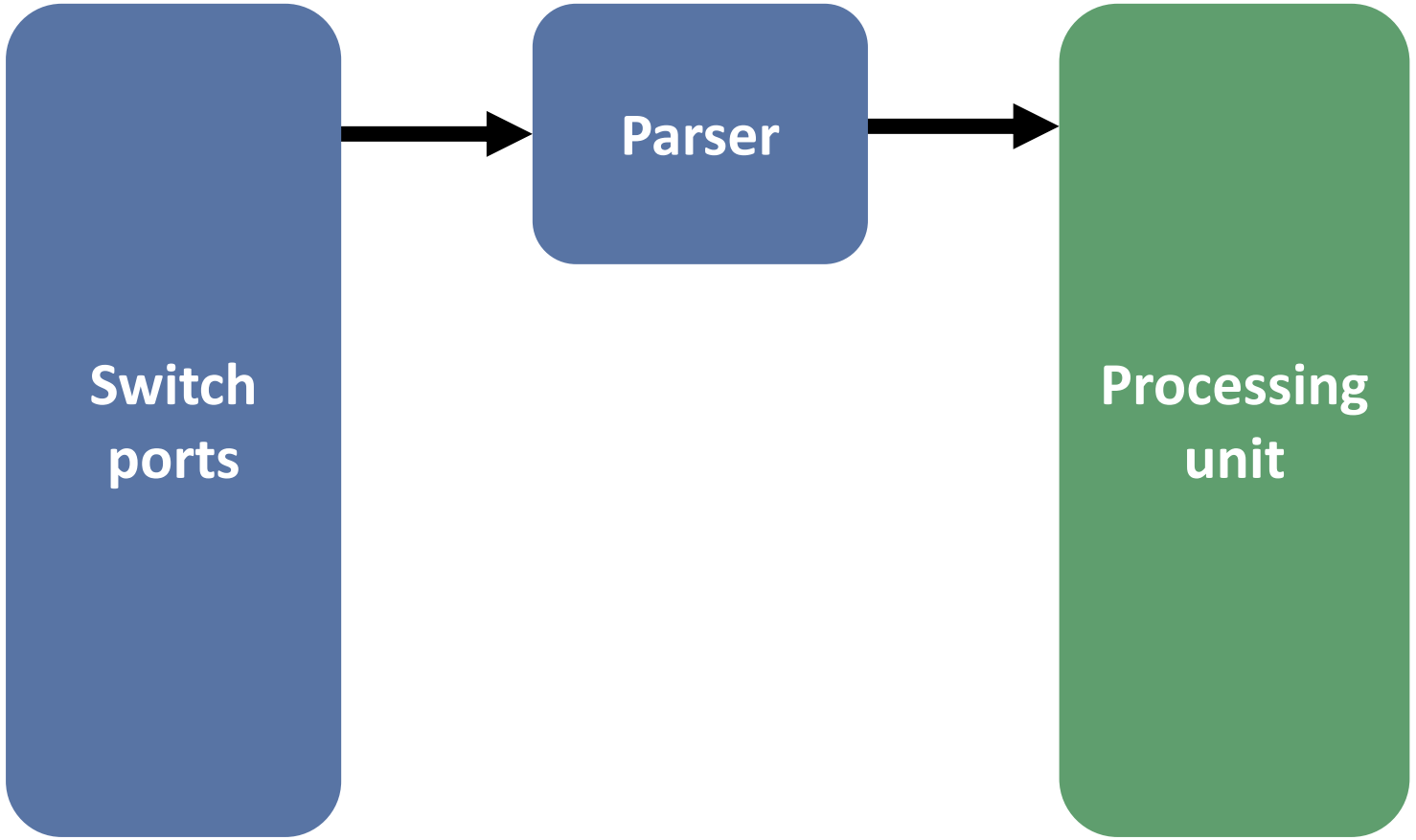


PsPIN-equipped switches

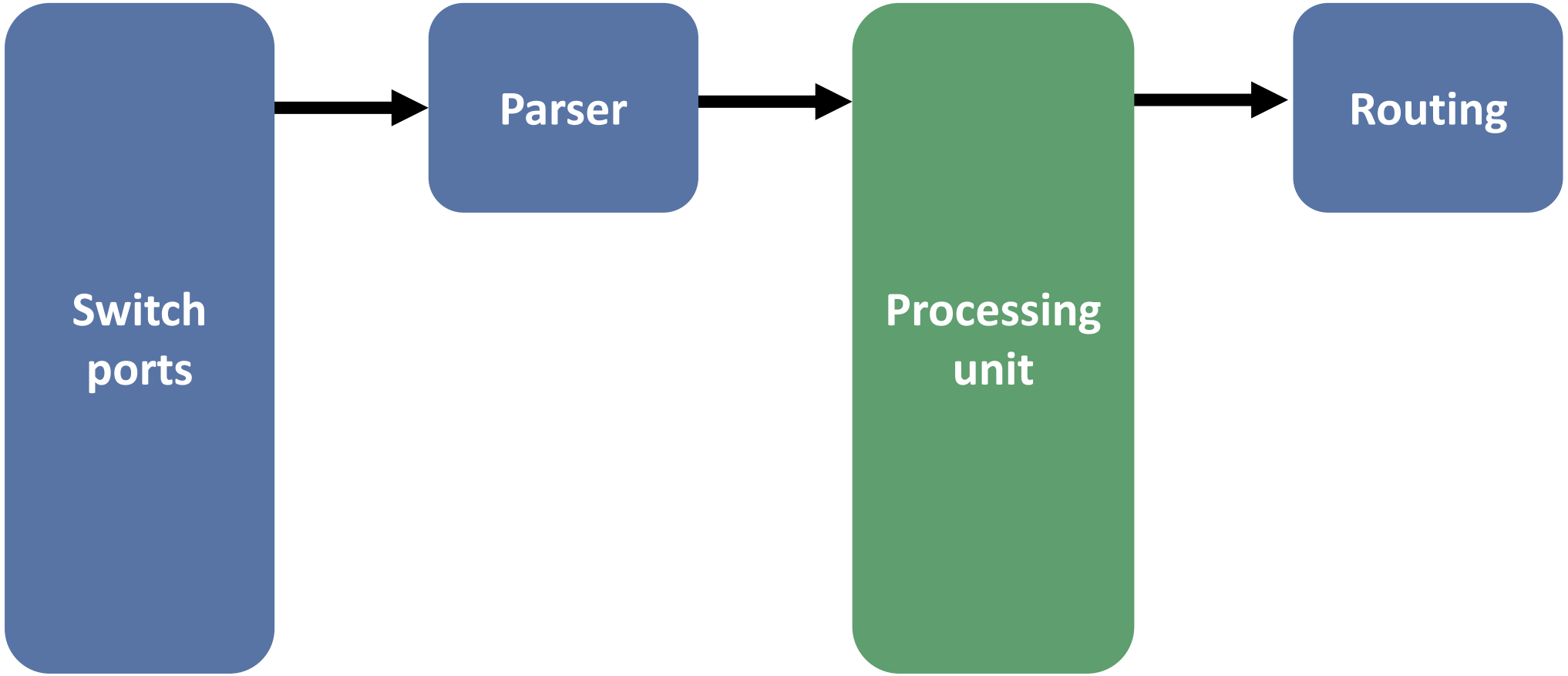
PsPIN-equipped switches



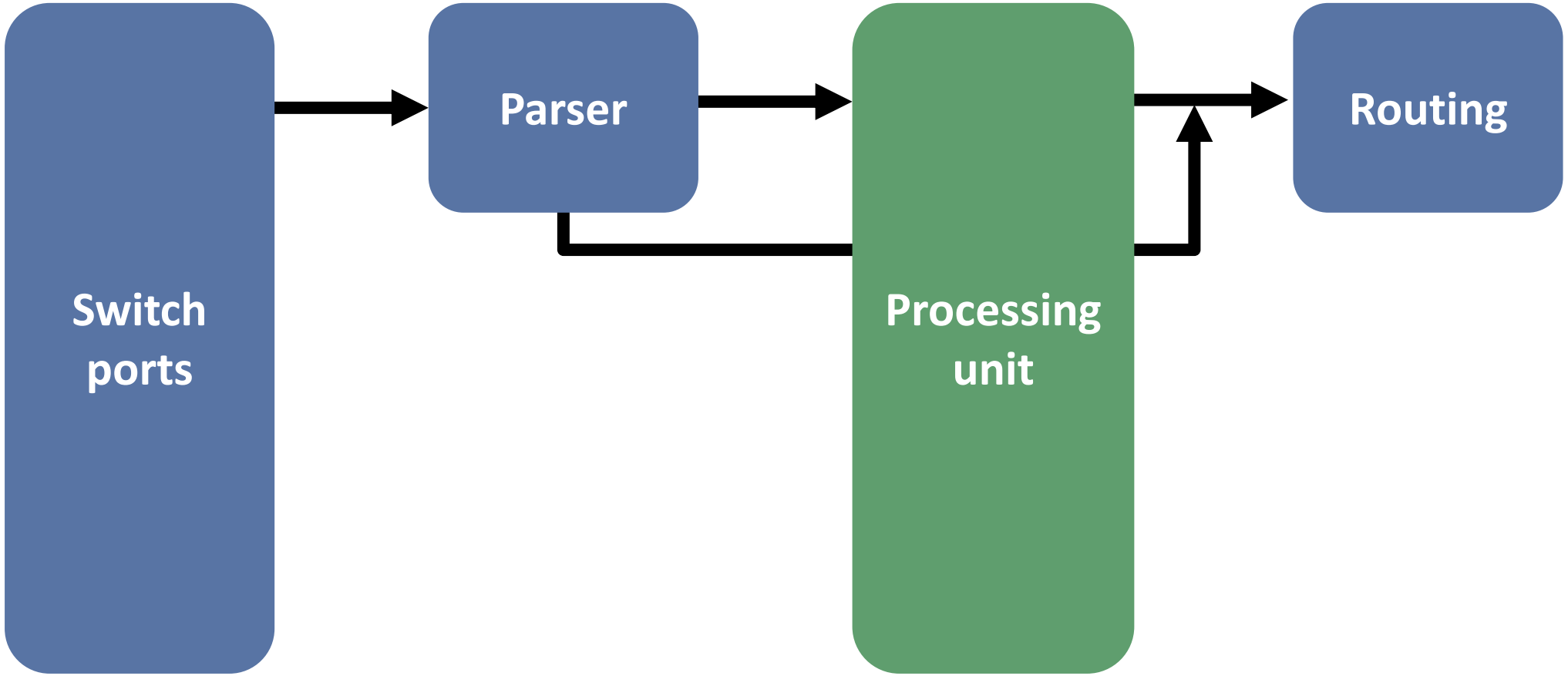
PsPIN-equipped switches



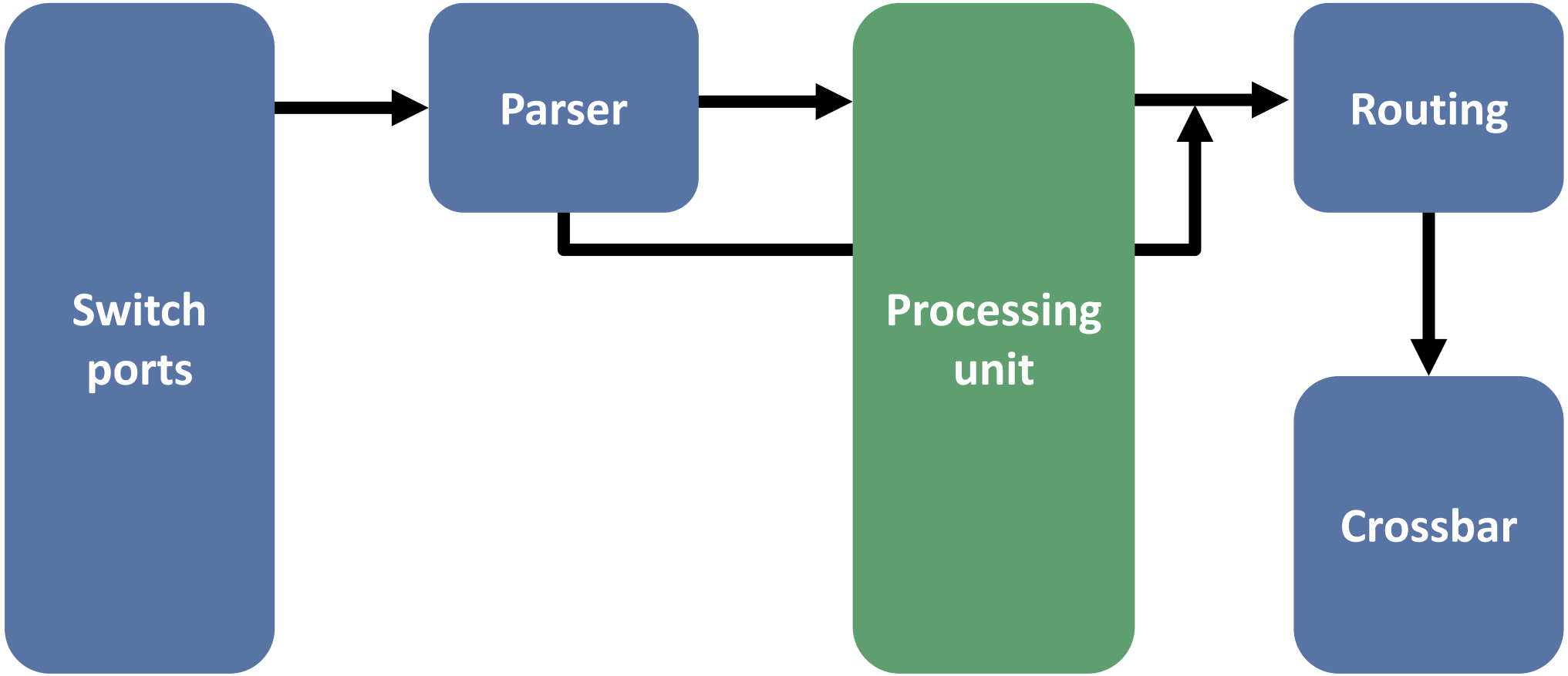
PsPIN-equipped switches



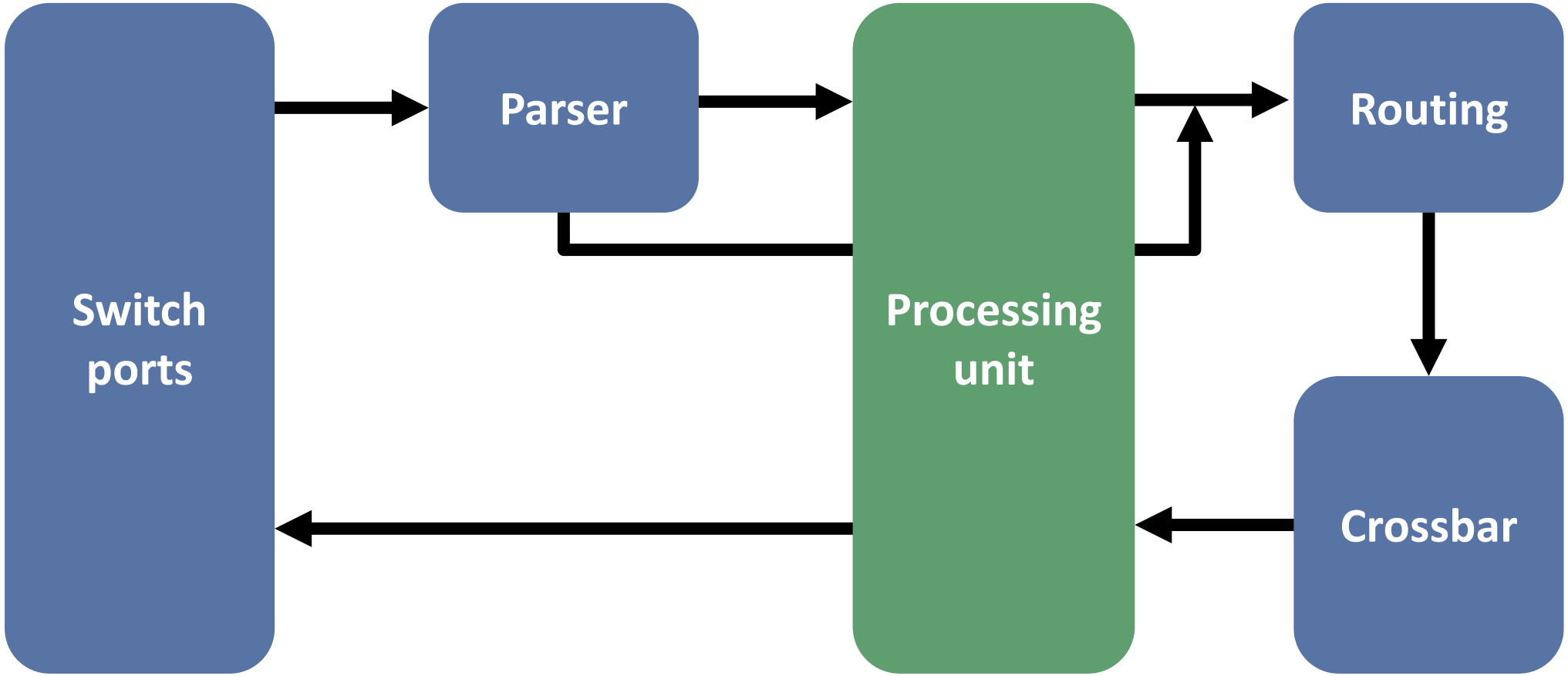
PsPIN-equipped switches



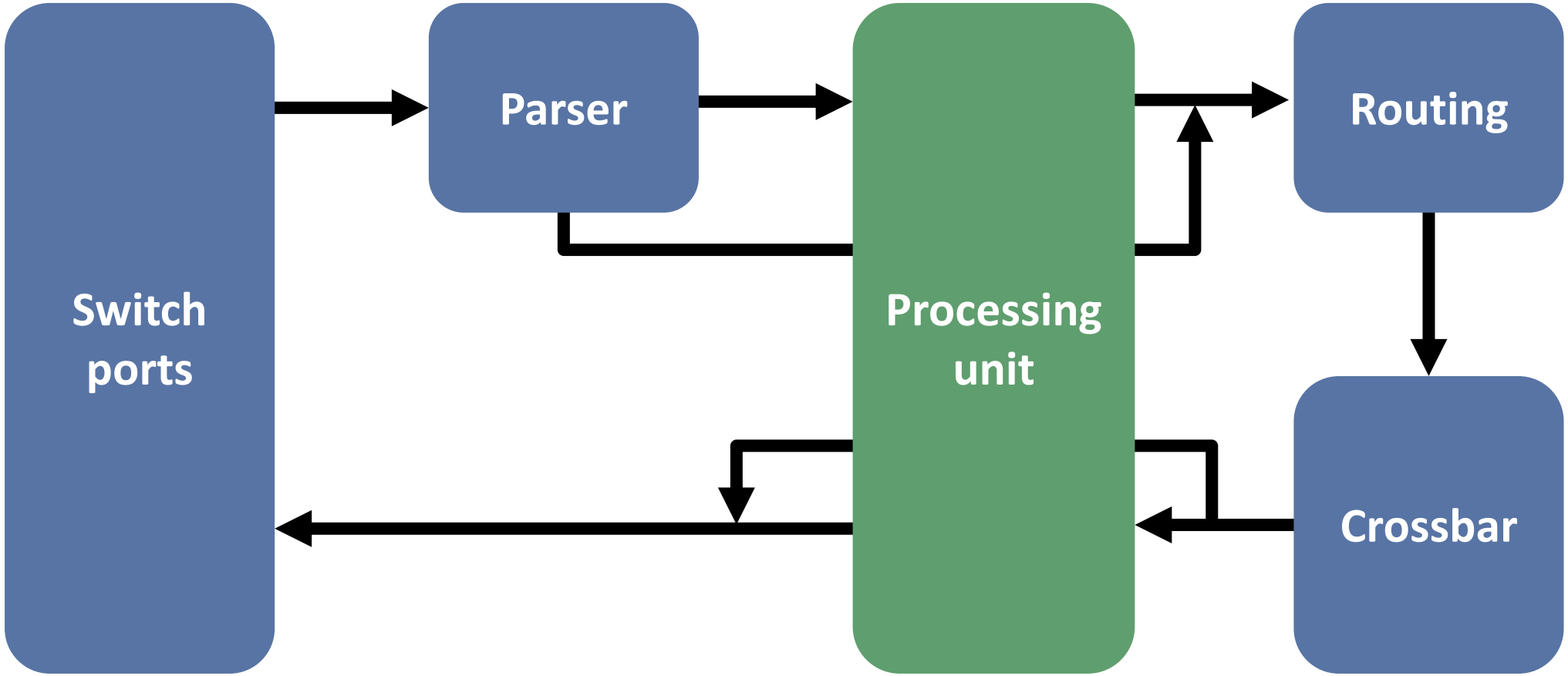
PsPIN-equipped switches



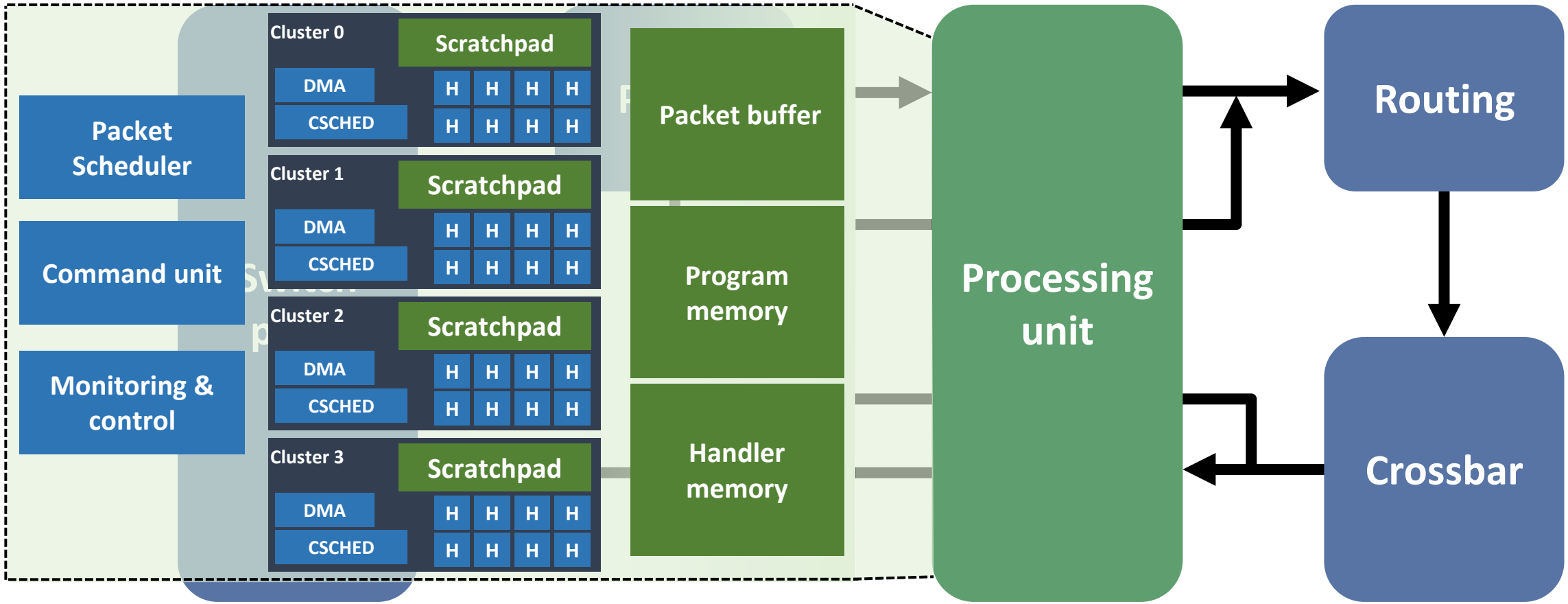
PsPIN-equipped switches



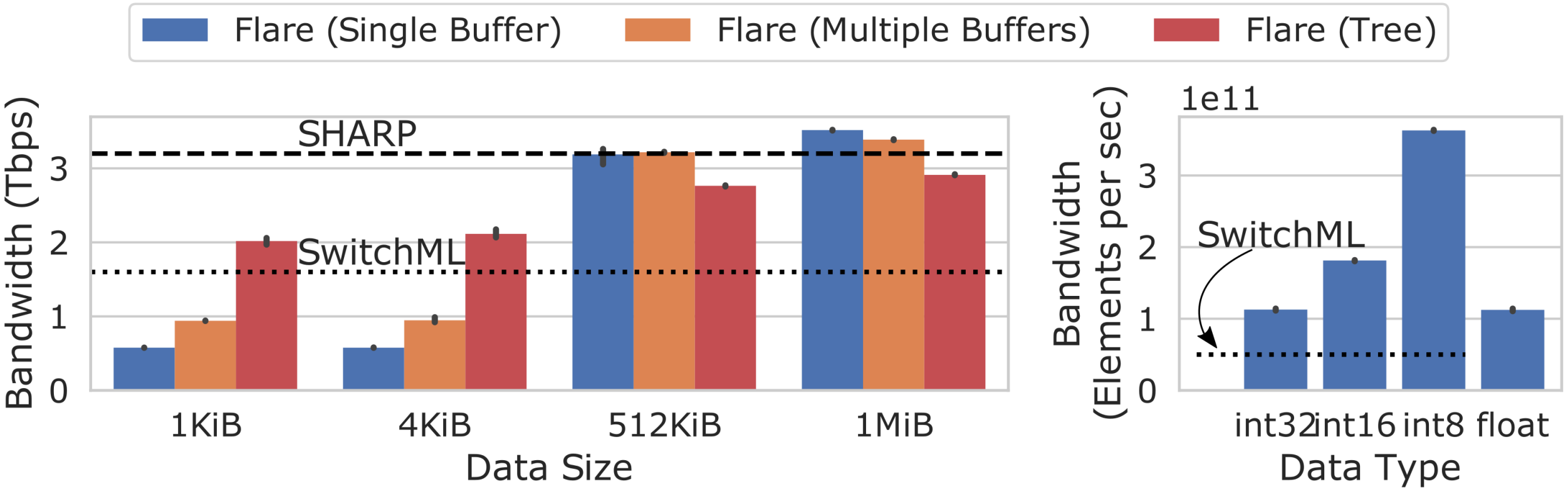
PsPIN-equipped switches



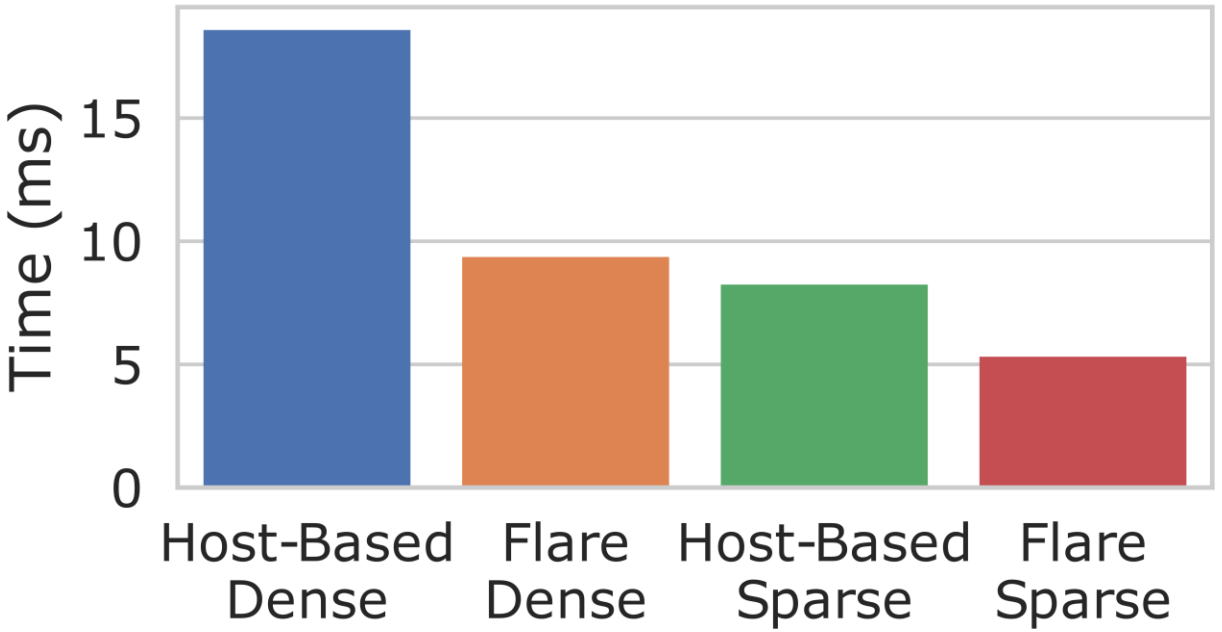
PsPIN-equipped switches

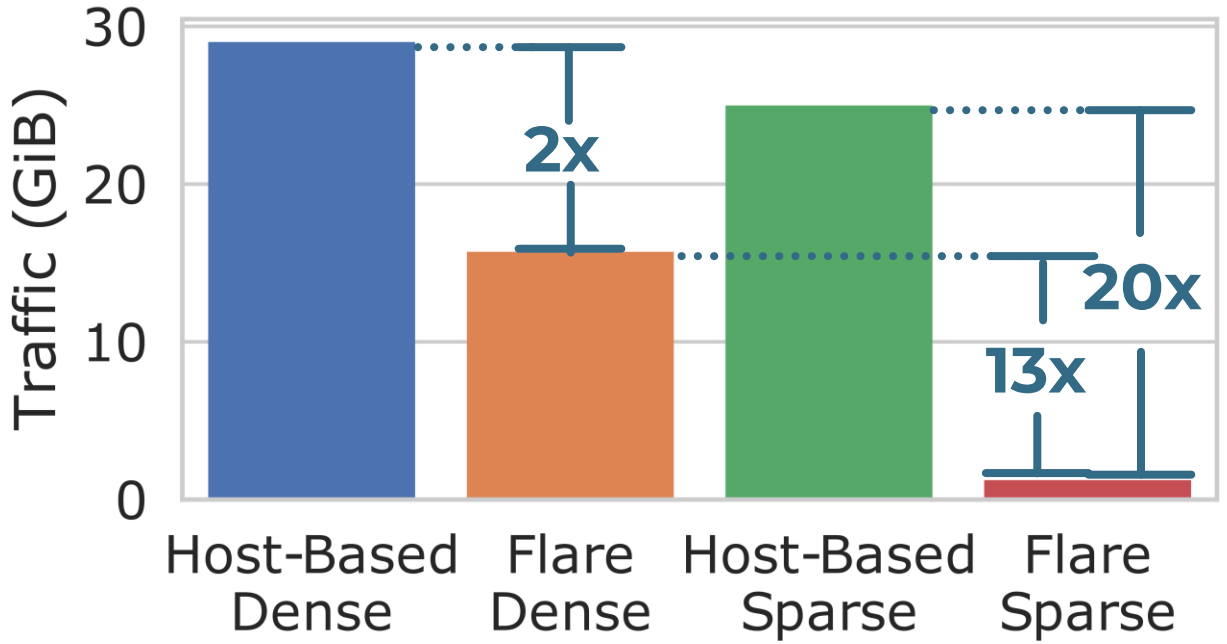


Results – single switch



Results – 64 nodes, 2-level fat tree



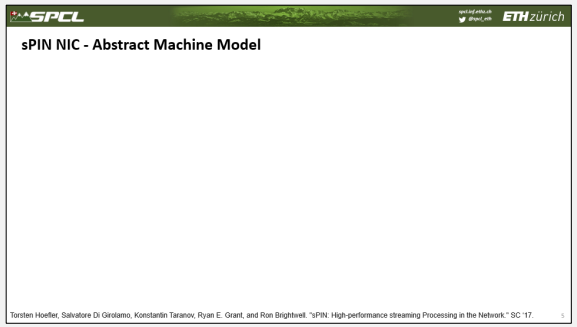


Communication time of a **ResNet50** iteration with **sparsified gradients** (0.2% density)

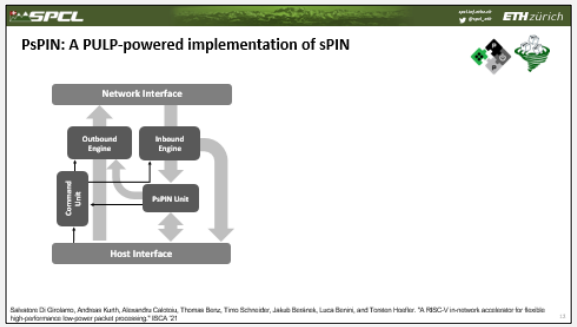
Wrapping up

Wrapping up

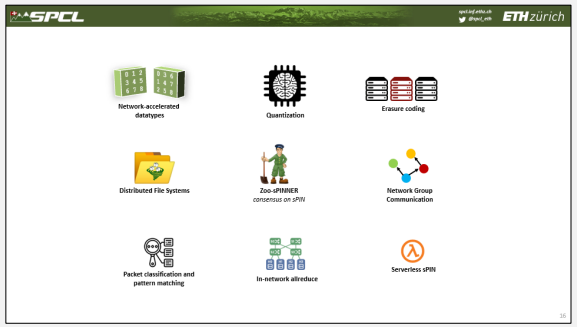
In this presentation



Programming model (SC ’17)



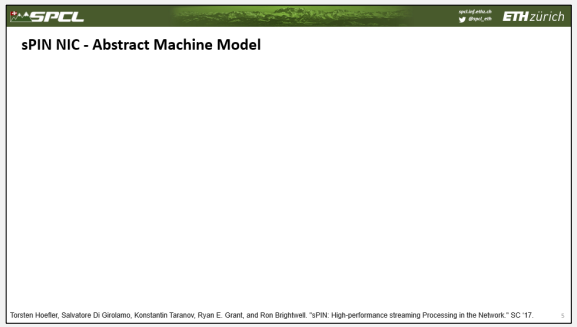
HW accelerator (ISCA ’21)



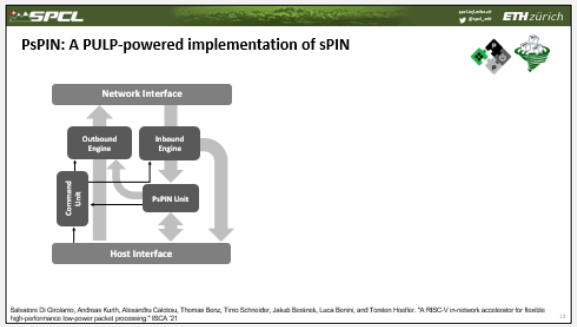
Use cases (SC ‘19, SC’21)

Wrapping up

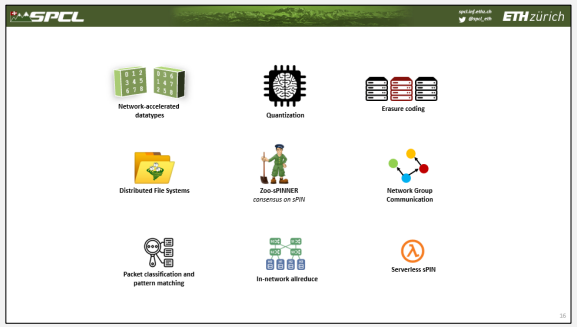
In this presentation



Programming model (SC ’17)



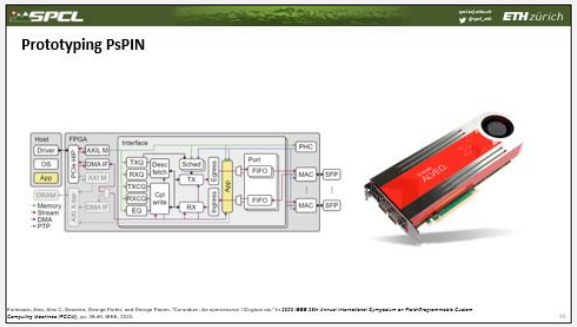
HW accelerator (ISCA ’21)



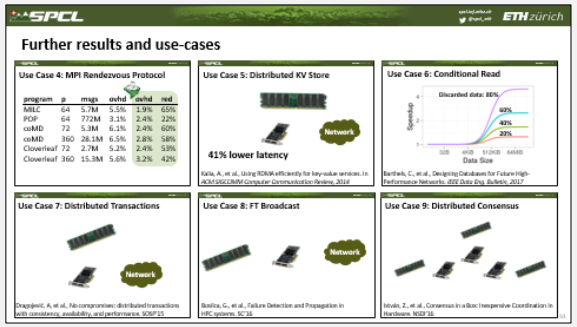
Use cases (SC ‘19, SC’21)

Our research directions

sPIN in the cloud



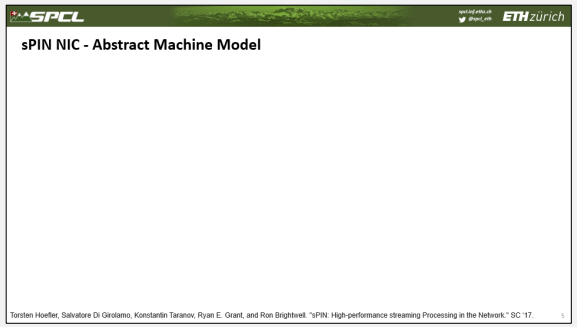
Prototyping PsPIN



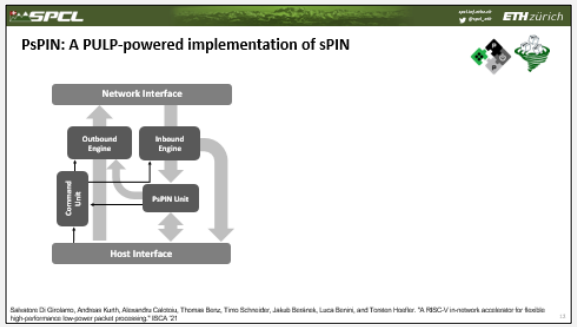
More use cases!

Wrapping up

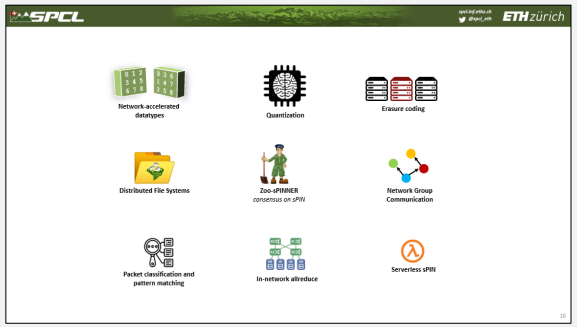
In this presentation



Programming model (SC ’17)



HW accelerator (ISCA ’21)



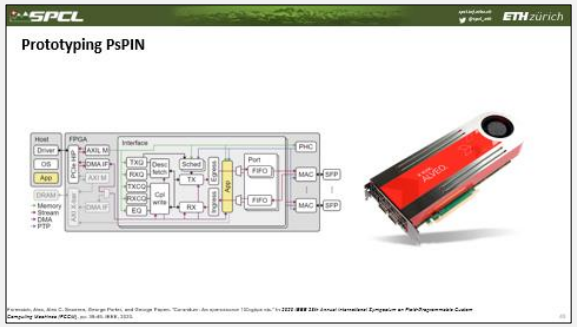
Use cases (SC ‘19, SC’21)

https://github.com/spcl/pspin

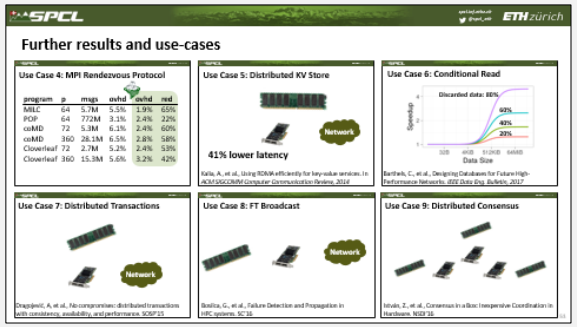
RTL, runtime, examples

Our research directions

sPIN in the cloud



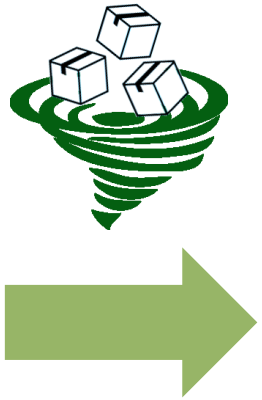
Prototyping PsPIN



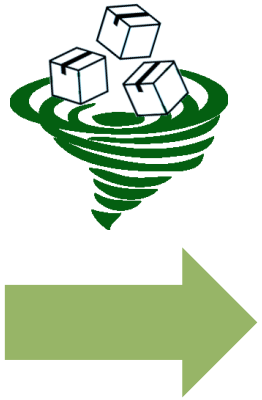
More use cases!

OSMOSIS

OSMOSIS



OSMOSIS

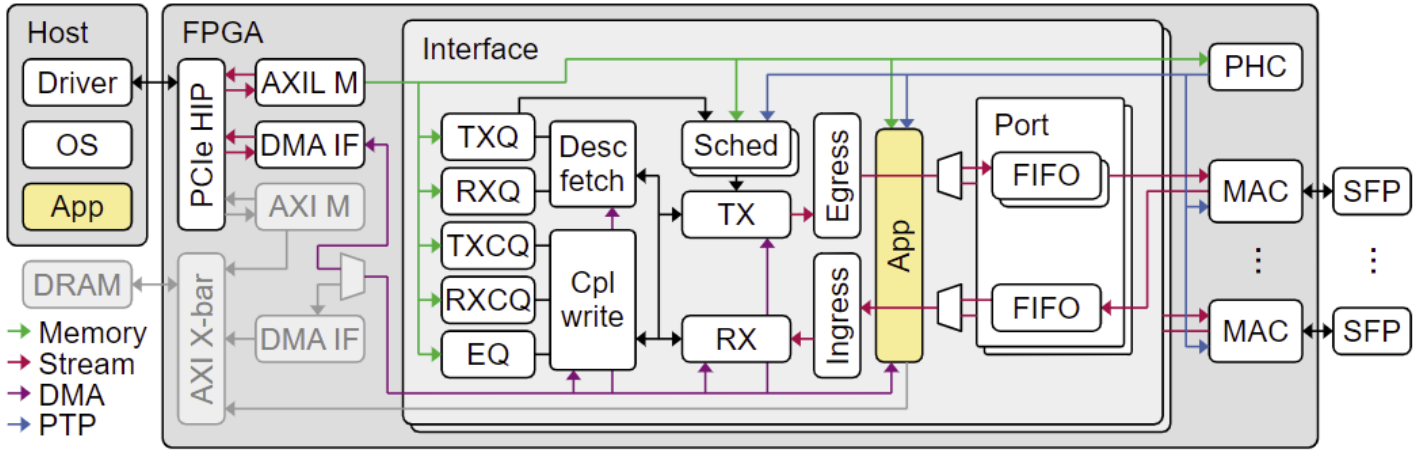


Isolation?

Multitenancy?

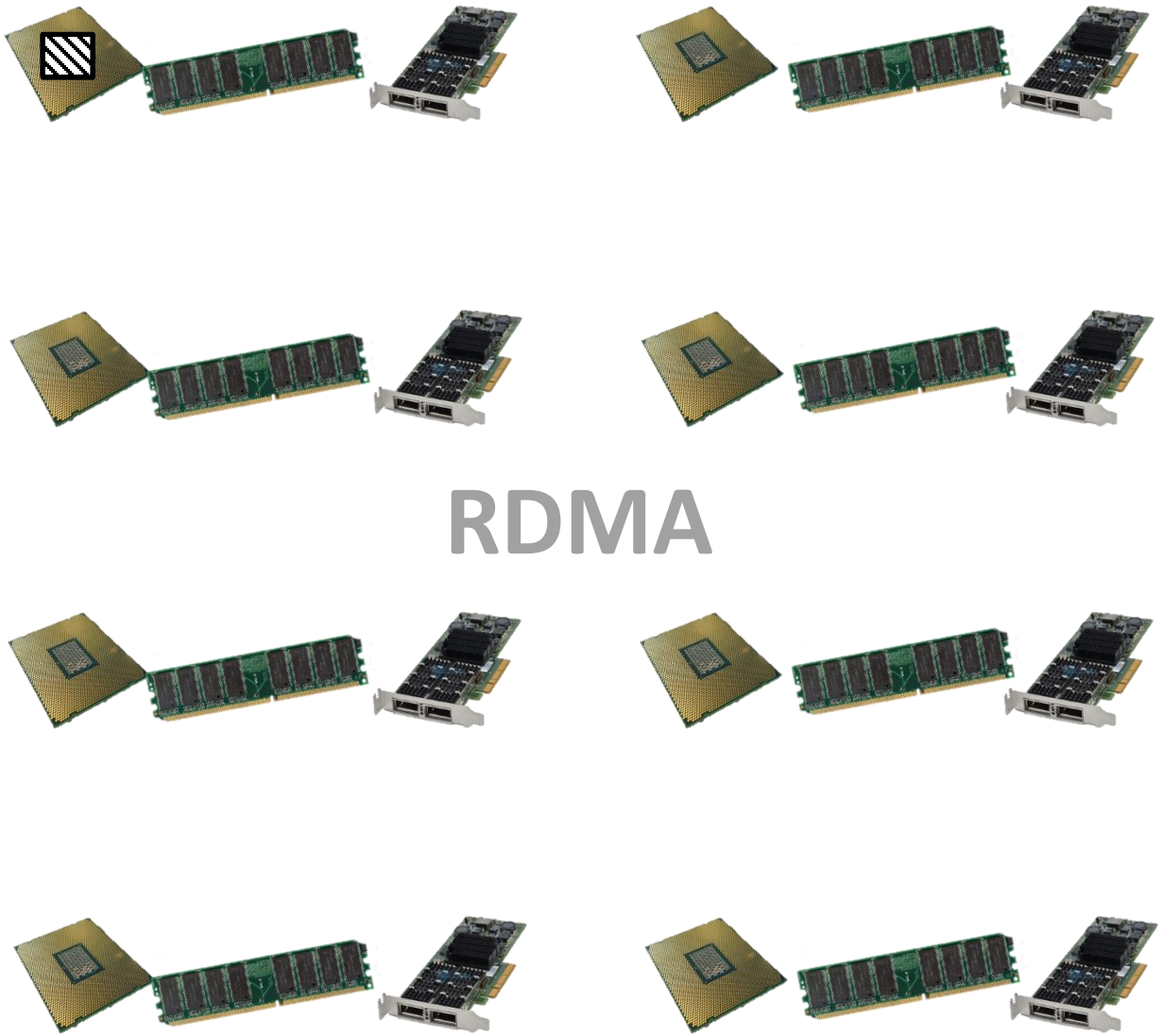
QoS?

Prototyping PsPIN



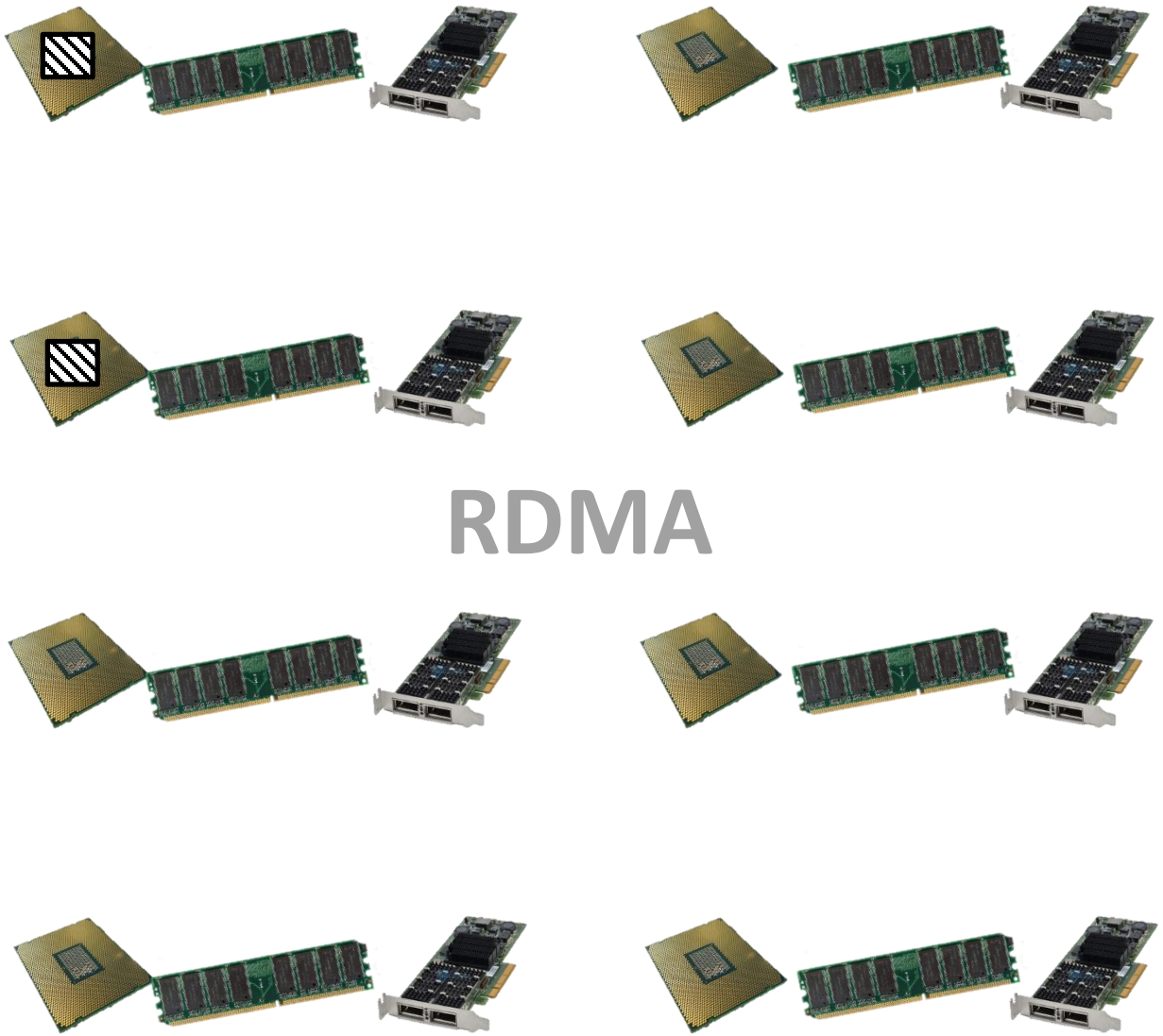
Forencich, Alex, Alex C. Snoeren, George Porter, and George Papen. "Corundum: An open-source 100-gbps nic." In *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 38-46. IEEE, 2020.

Use Case 1: Broadcast acceleration



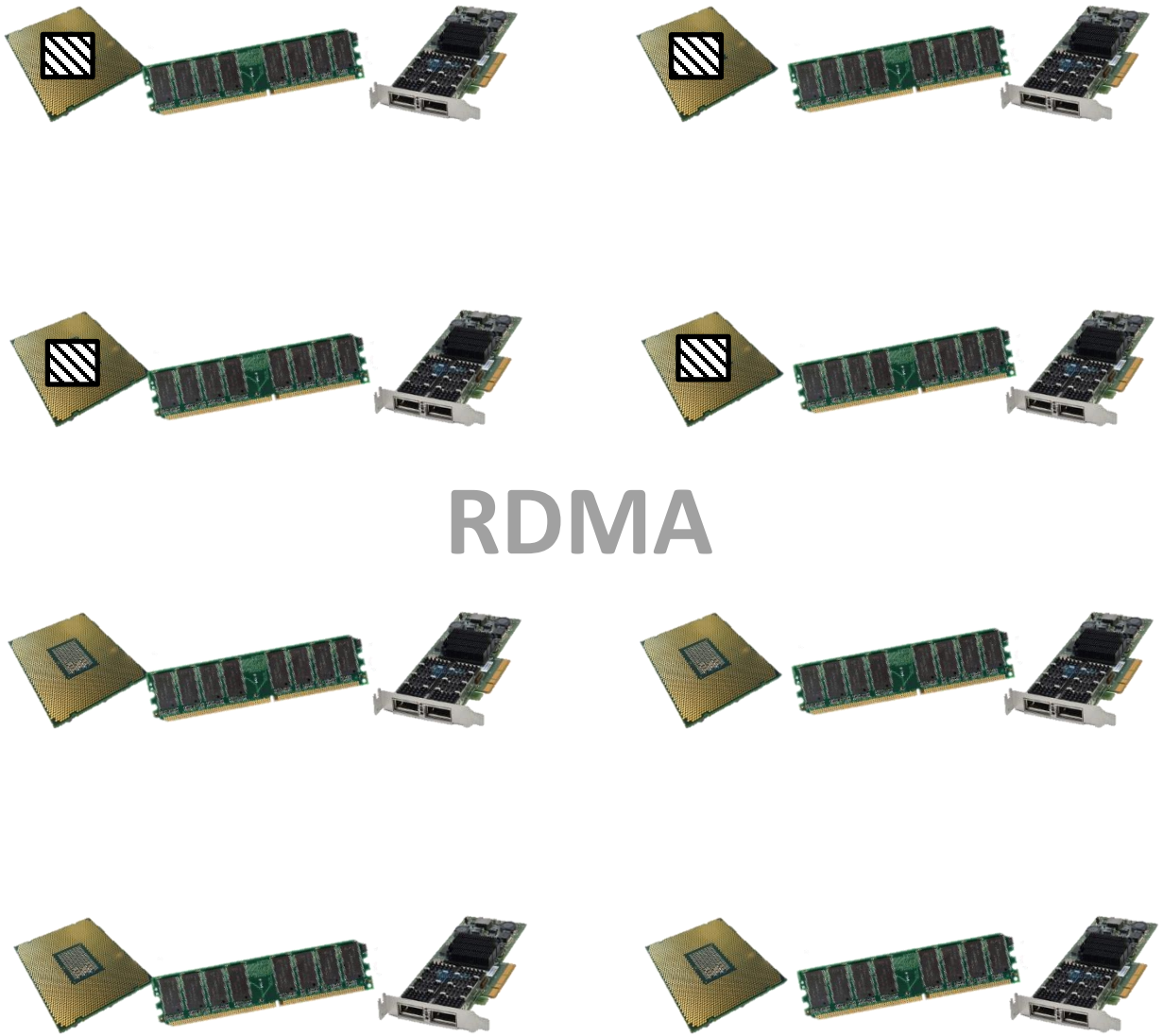
Liu, J., et al., High performance RDMA-based MPI implementation over InfiniBand. *International Journal of Parallel Programming* 2004

Use Case 1: Broadcast acceleration



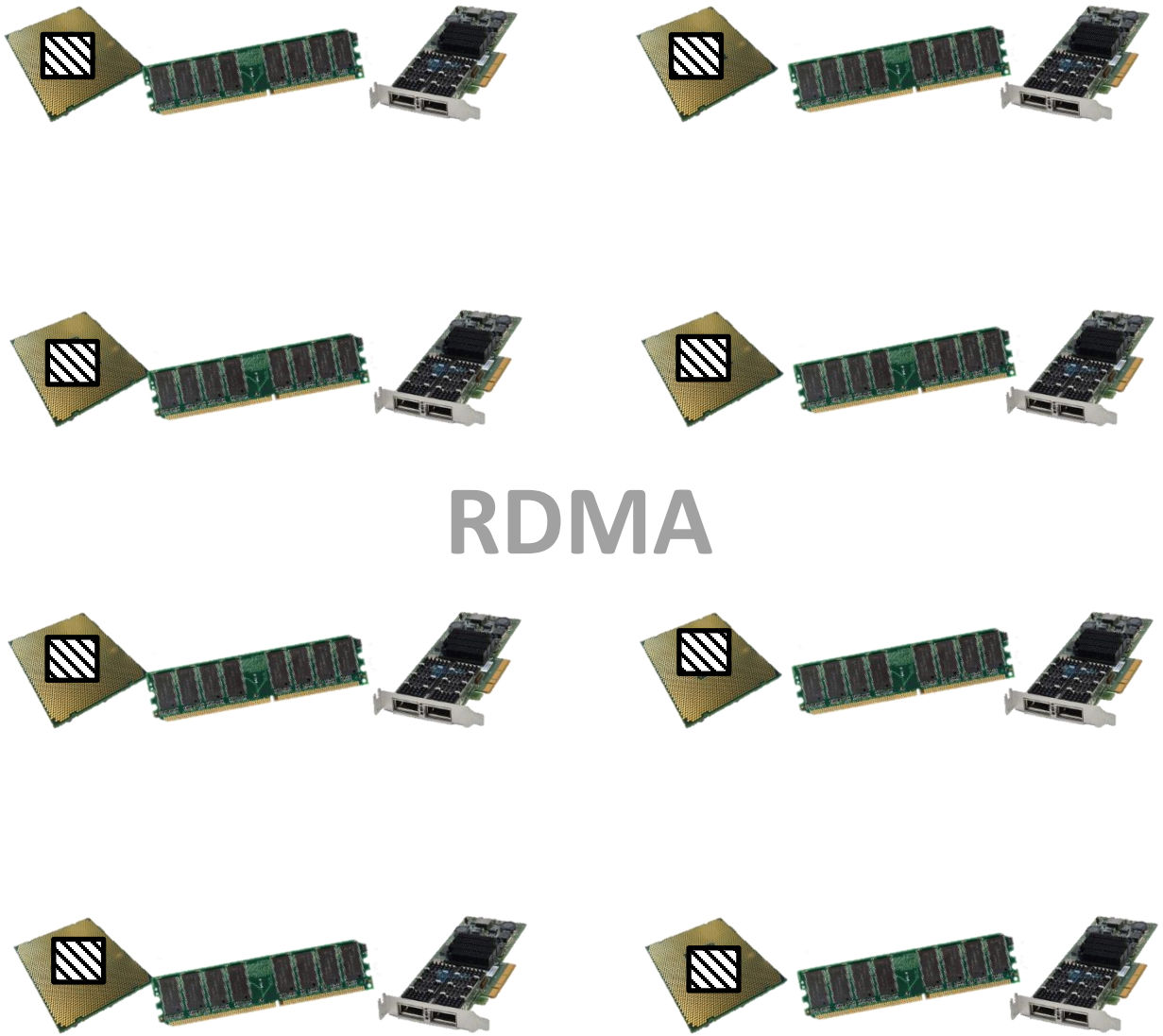
Liu, J., et al., High performance RDMA-based MPI implementation over InfiniBand. *International Journal of Parallel Programming* 2004

Use Case 1: Broadcast acceleration



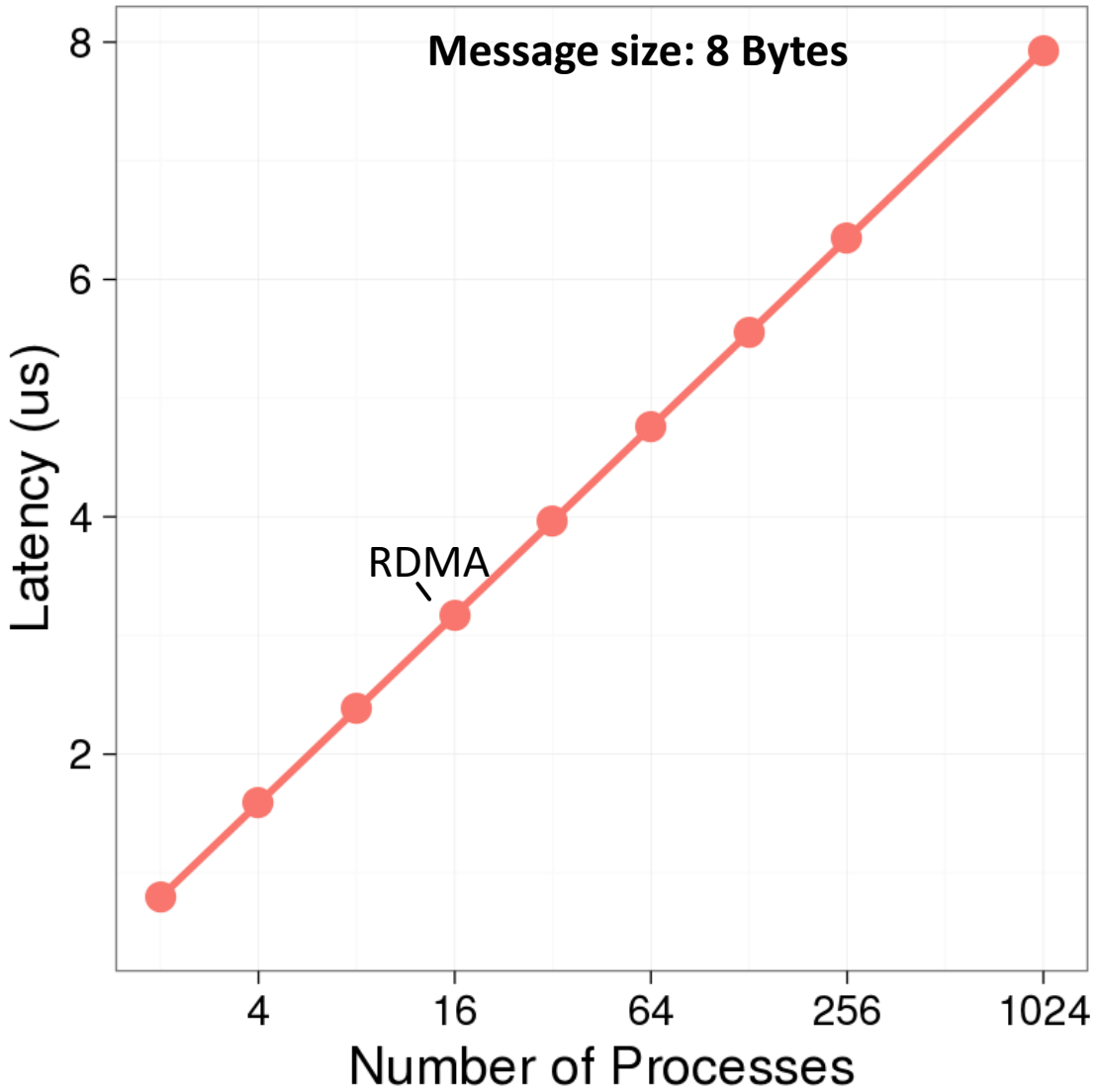
Liu, J., et al., High performance RDMA-based MPI implementation over InfiniBand. *International Journal of Parallel Programming* 2004

Use Case 1: Broadcast acceleration



Liu, J., et al., High performance RDMA-based MPI implementation over InfiniBand. *International Journal of Parallel Programming* 2004

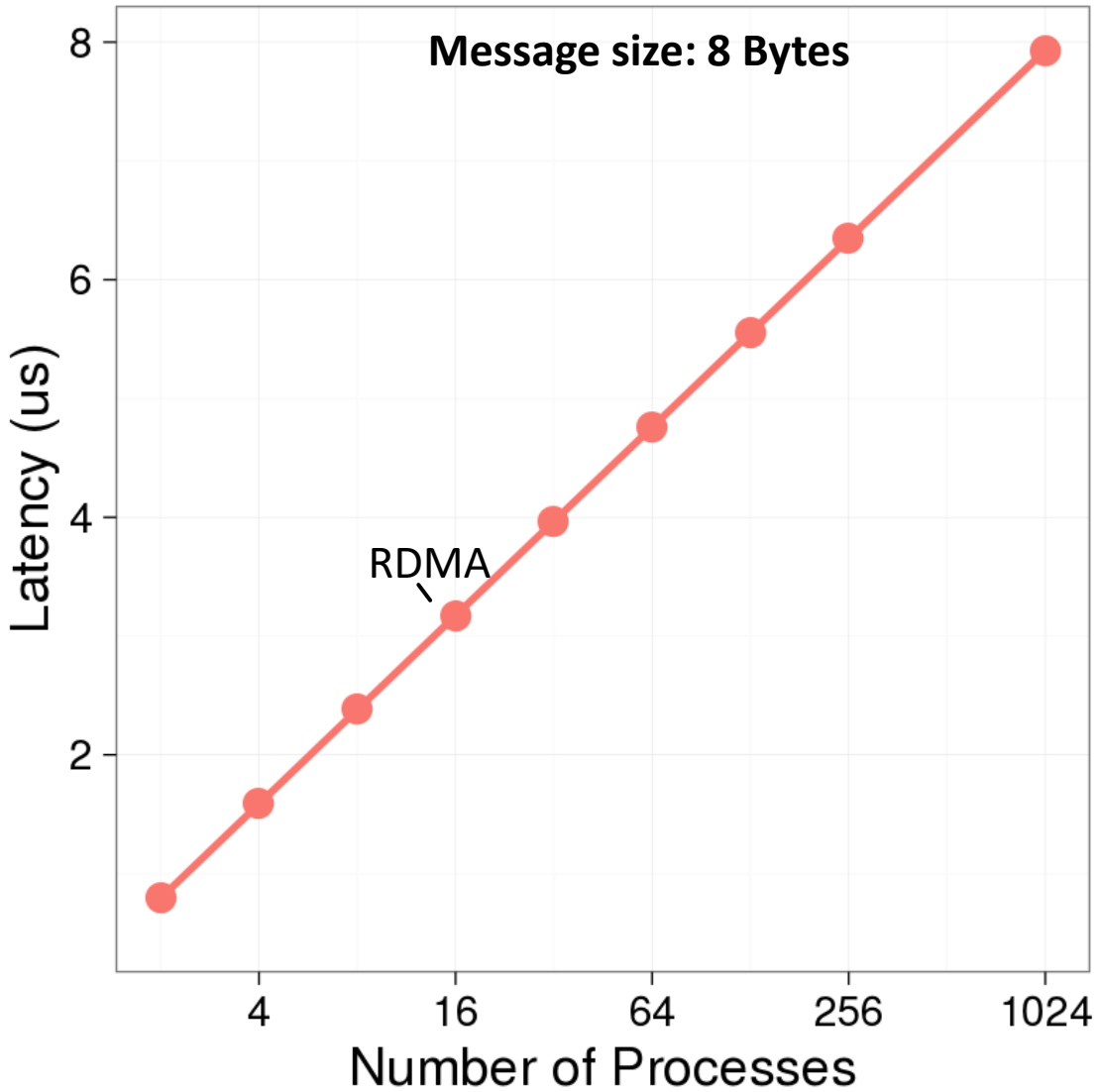
Use Case 1: Broadcast acceleration



Liu, J., et al., High performance RDMA-based MPI implementation over InfiniBand. *International Journal of Parallel Programming* 2004

Use Case 1: Broadcast acceleration

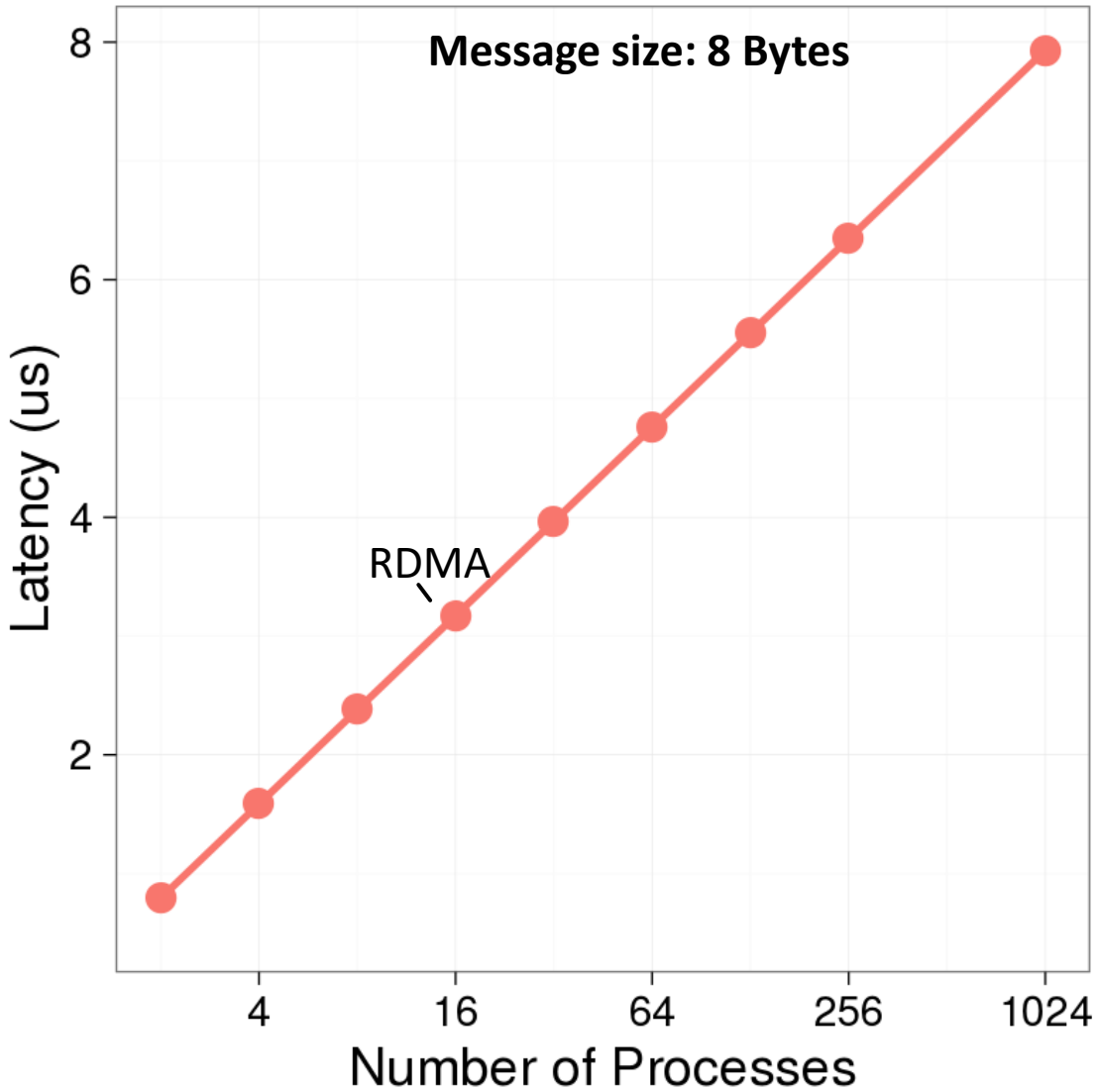
Network Group Communication



Underwood, K.D., et al., Enabling flexible collective communication offload with triggered operations. *HOTI'11*
Liu, J., et al., High performance RDMA-based MPI implementation over InfiniBand. *International Journal of Parallel Programming* 2004

Use Case 1: Broadcast acceleration

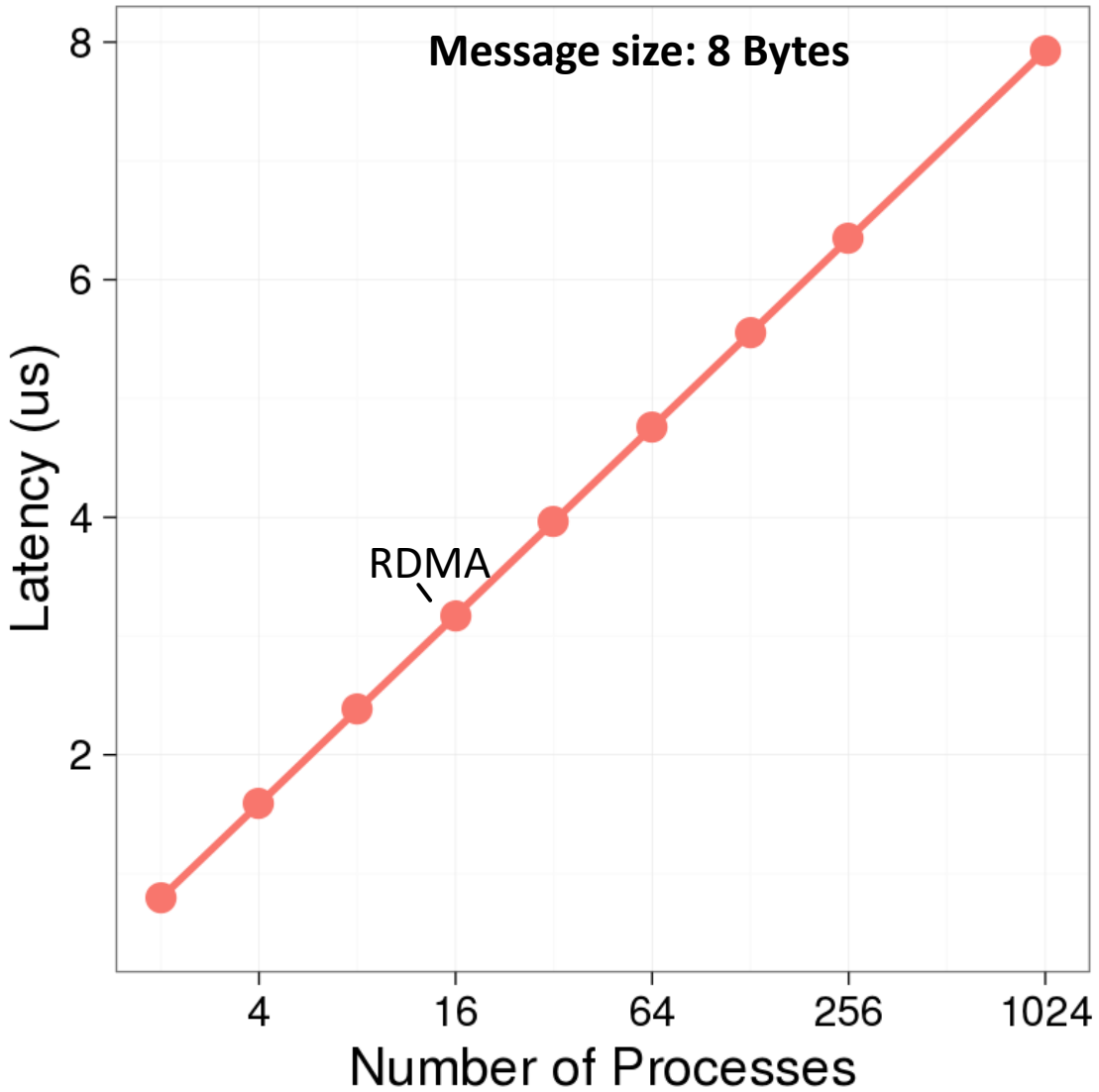
Network Group Communication



Underwood, K.D., et al., Enabling flexible collective communication offload with triggered operations. *HOTI'11*
Liu, J., et al., High performance RDMA-based MPI implementation over InfiniBand. *International Journal of Parallel Programming* 2004

Use Case 1: Broadcast acceleration

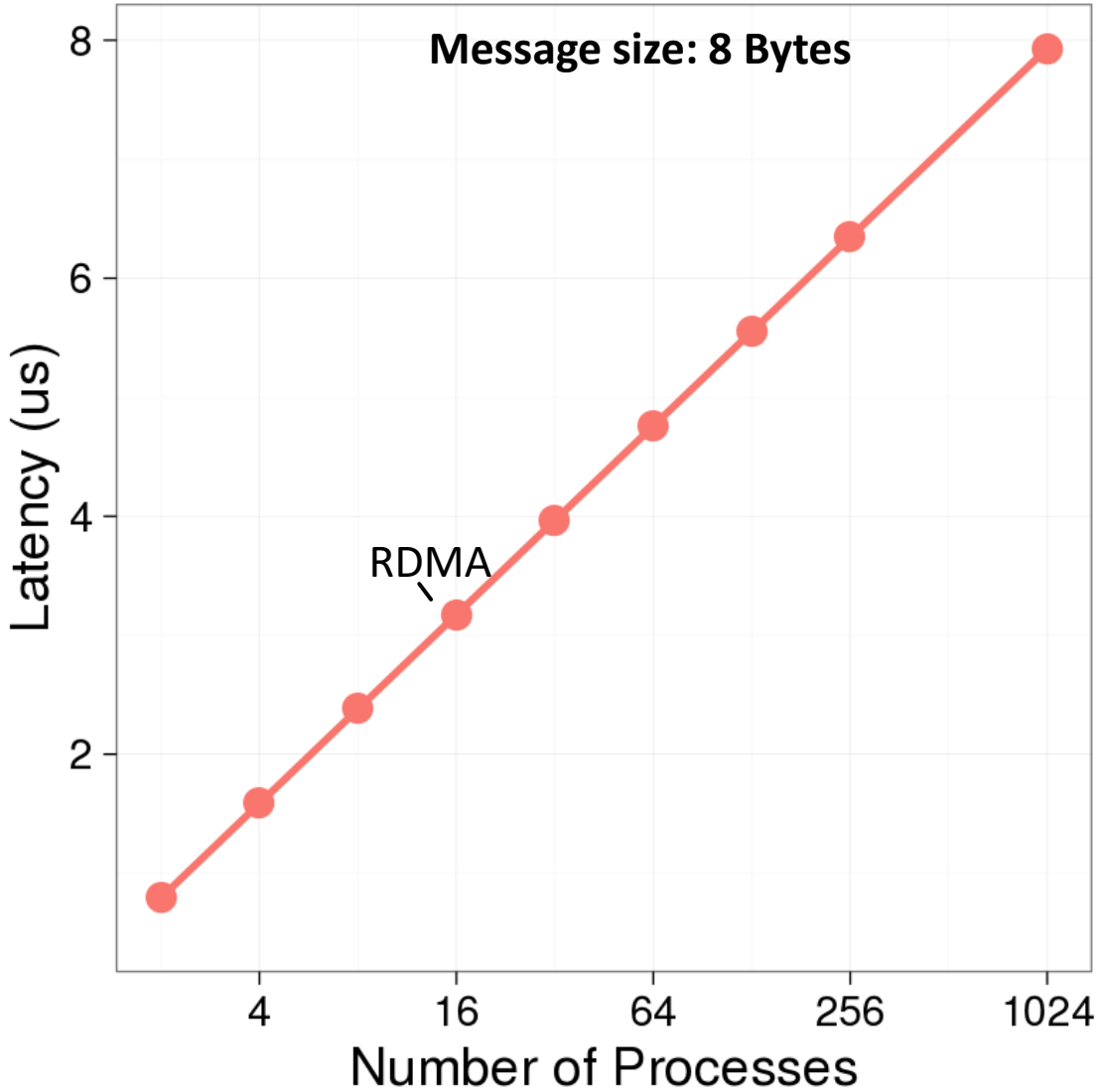
Network Group Communication



Underwood, K.D., et al., Enabling flexible collective communication offload with triggered operations. *HOTI'11*
Liu, J., et al., High performance RDMA-based MPI implementation over InfiniBand. *International Journal of Parallel Programming* 2004

Use Case 1: Broadcast acceleration

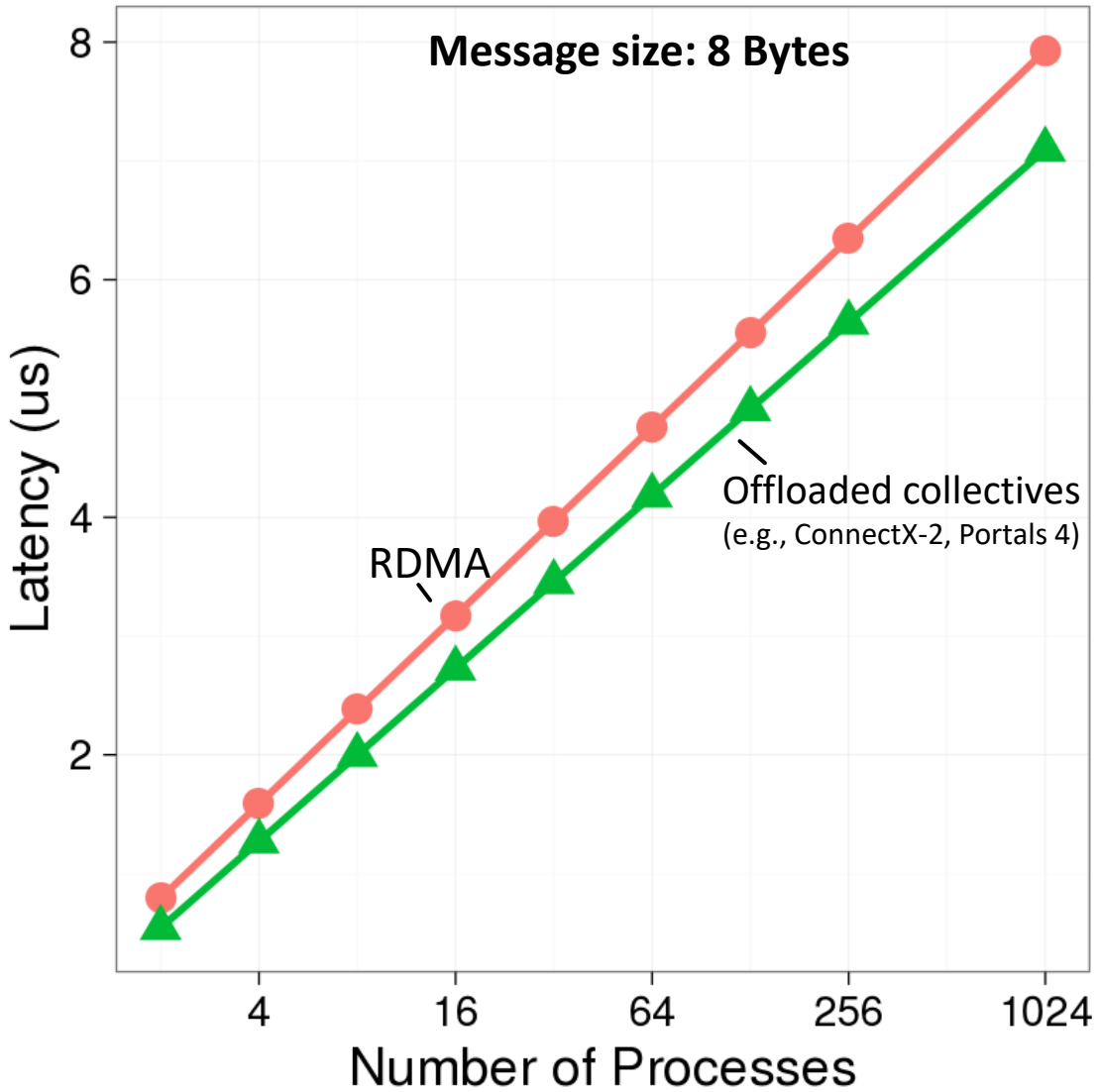
Network Group Communication



Underwood, K.D., et al., Enabling flexible collective communication offload with triggered operations. *HOTI'11*
Liu, J., et al., High performance RDMA-based MPI implementation over InfiniBand. *International Journal of Parallel Programming* 2004

Use Case 1: Broadcast acceleration

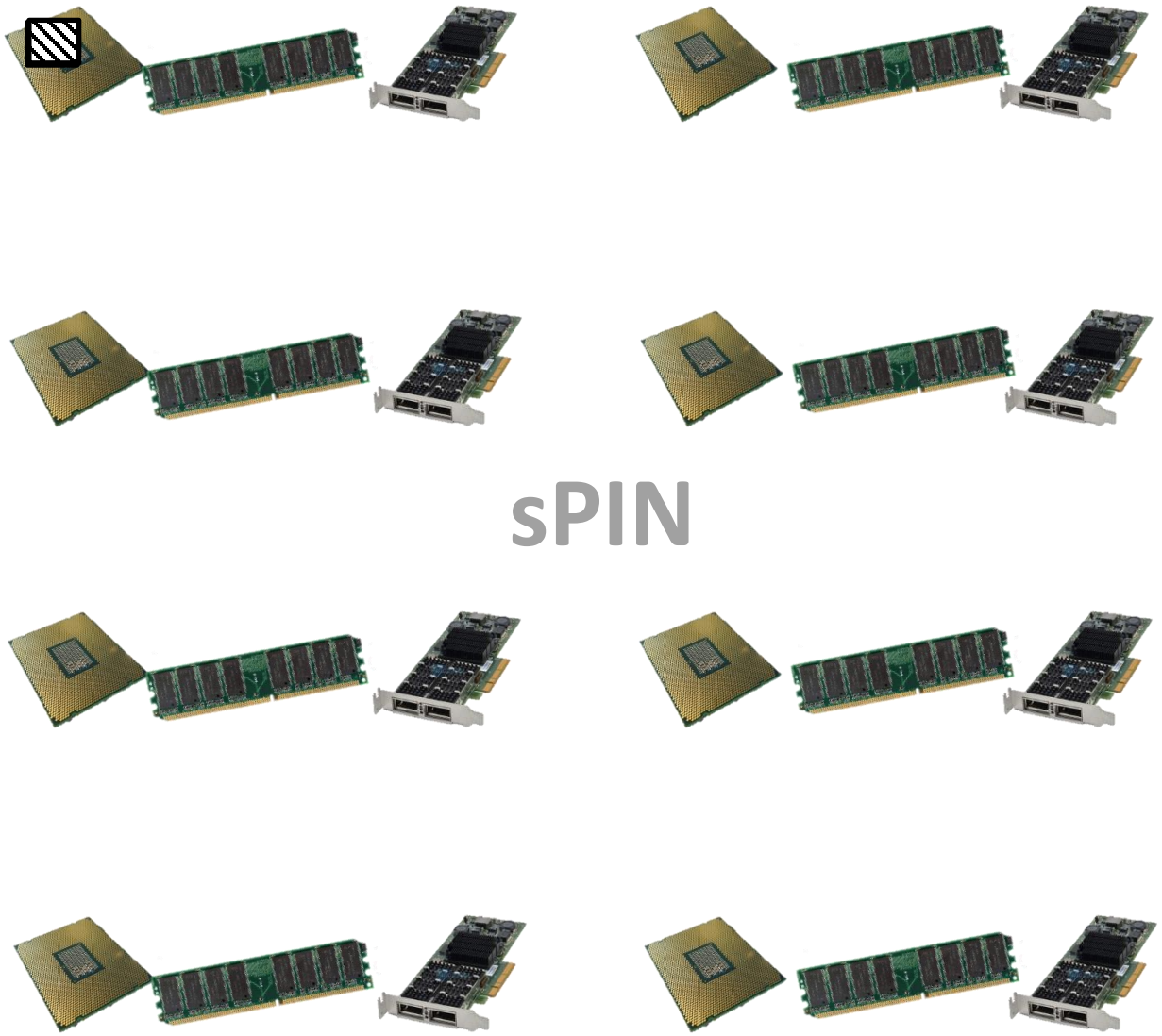
Network Group Communication

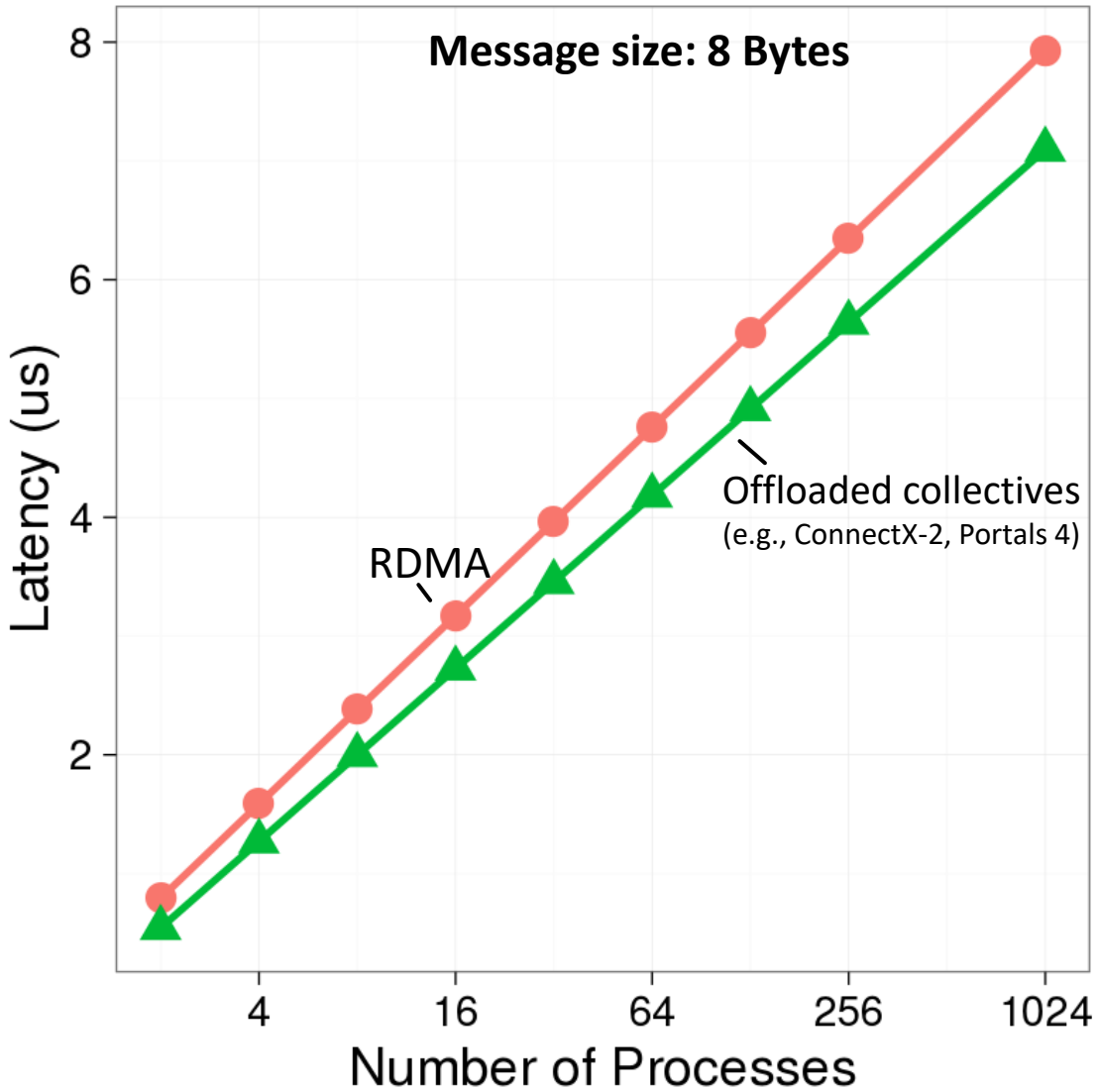


Underwood, K.D., et al., Enabling flexible collective communication offload with triggered operations. *HOTI'11*
Liu, J., et al., High performance RDMA-based MPI implementation over InfiniBand. *International Journal of Parallel Programming* 2004

Use Case 1: Broadcast acceleration

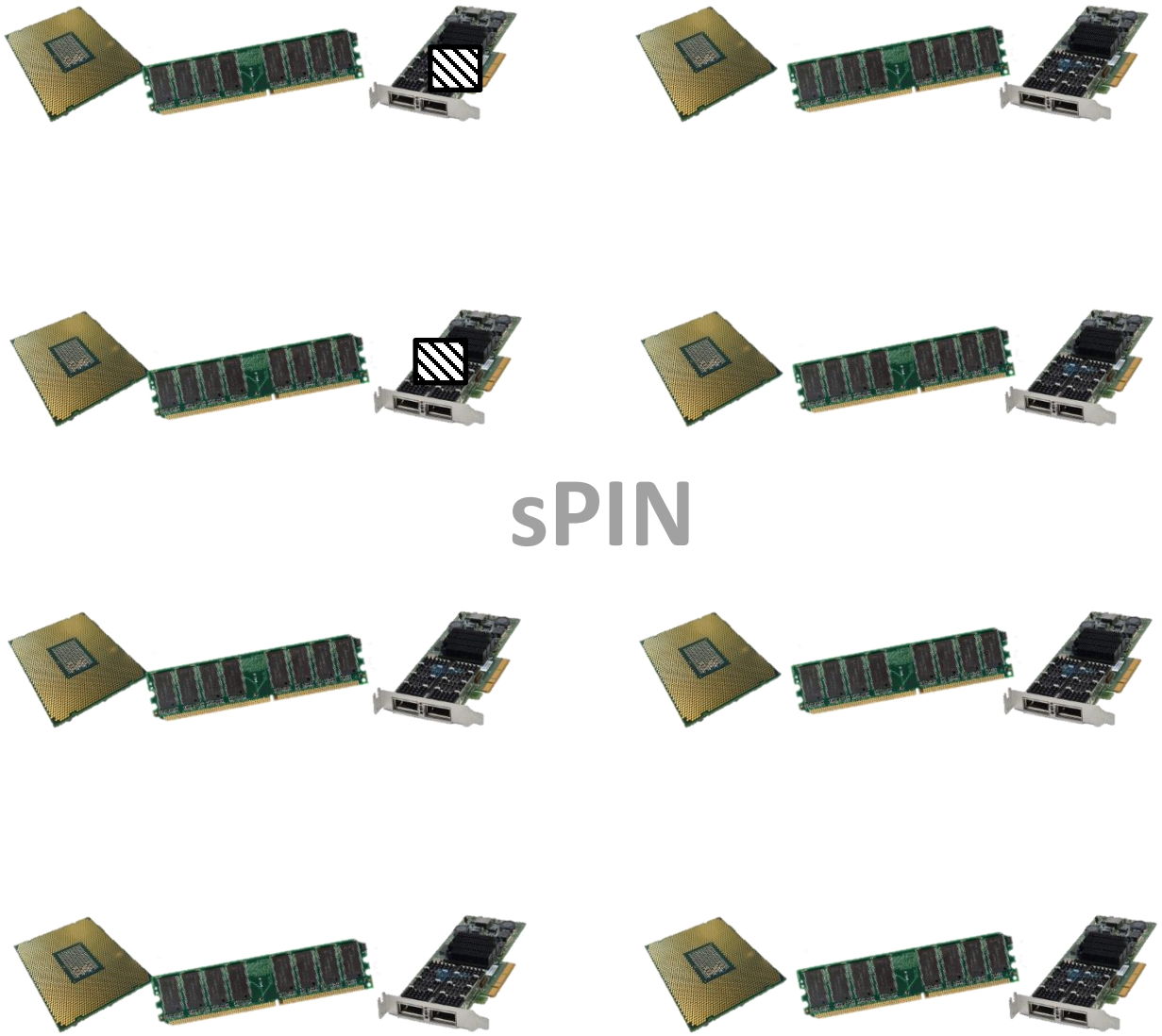
Network Group Communication



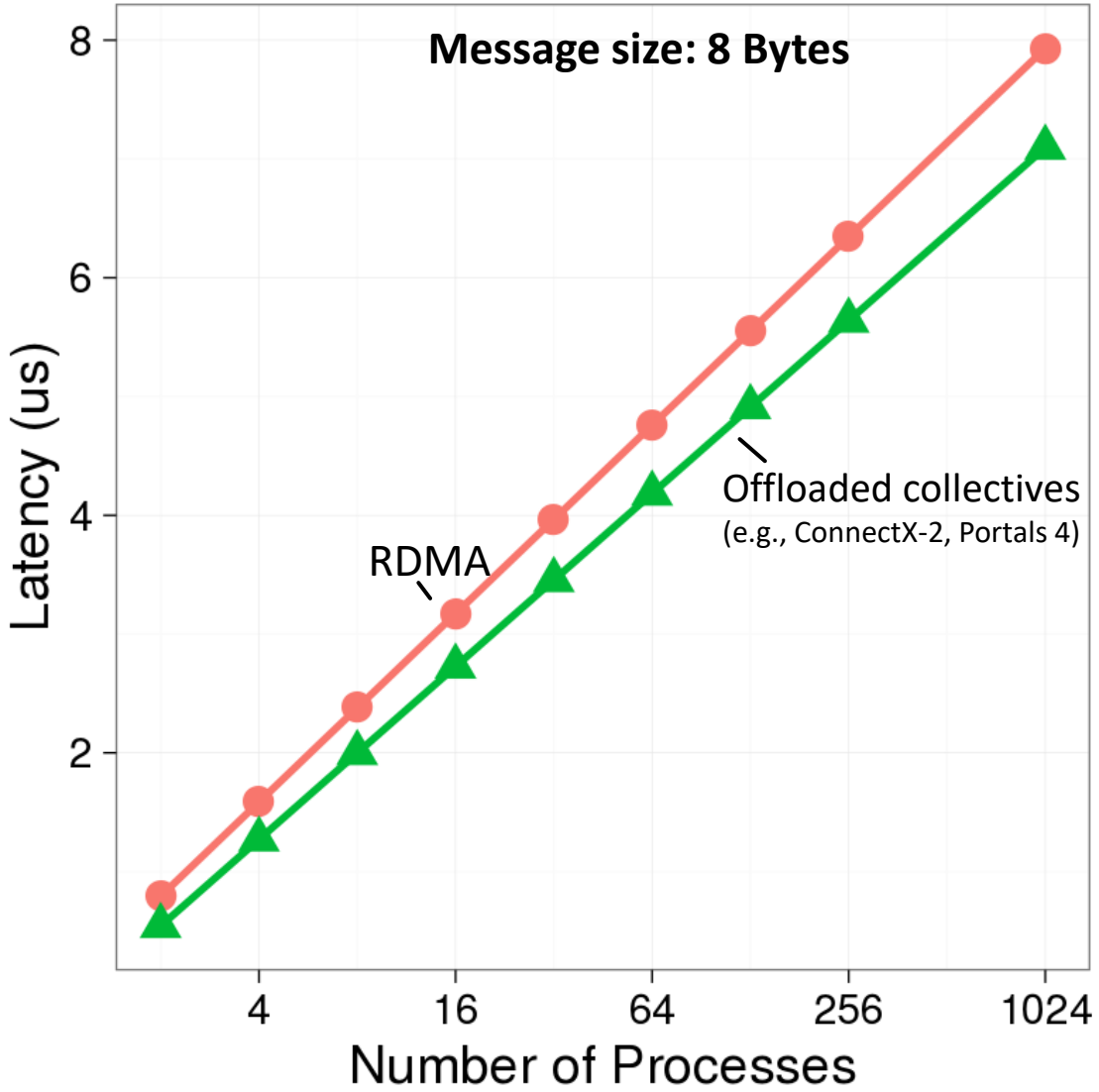


Underwood, K.D., et al., Enabling flexible collective communication offload with triggered operations. *HOTI'11*
Liu, J., et al., High performance RDMA-based MPI implementation over InfiniBand. *International Journal of Parallel Programming* 2004

Use Case 1: Broadcast acceleration

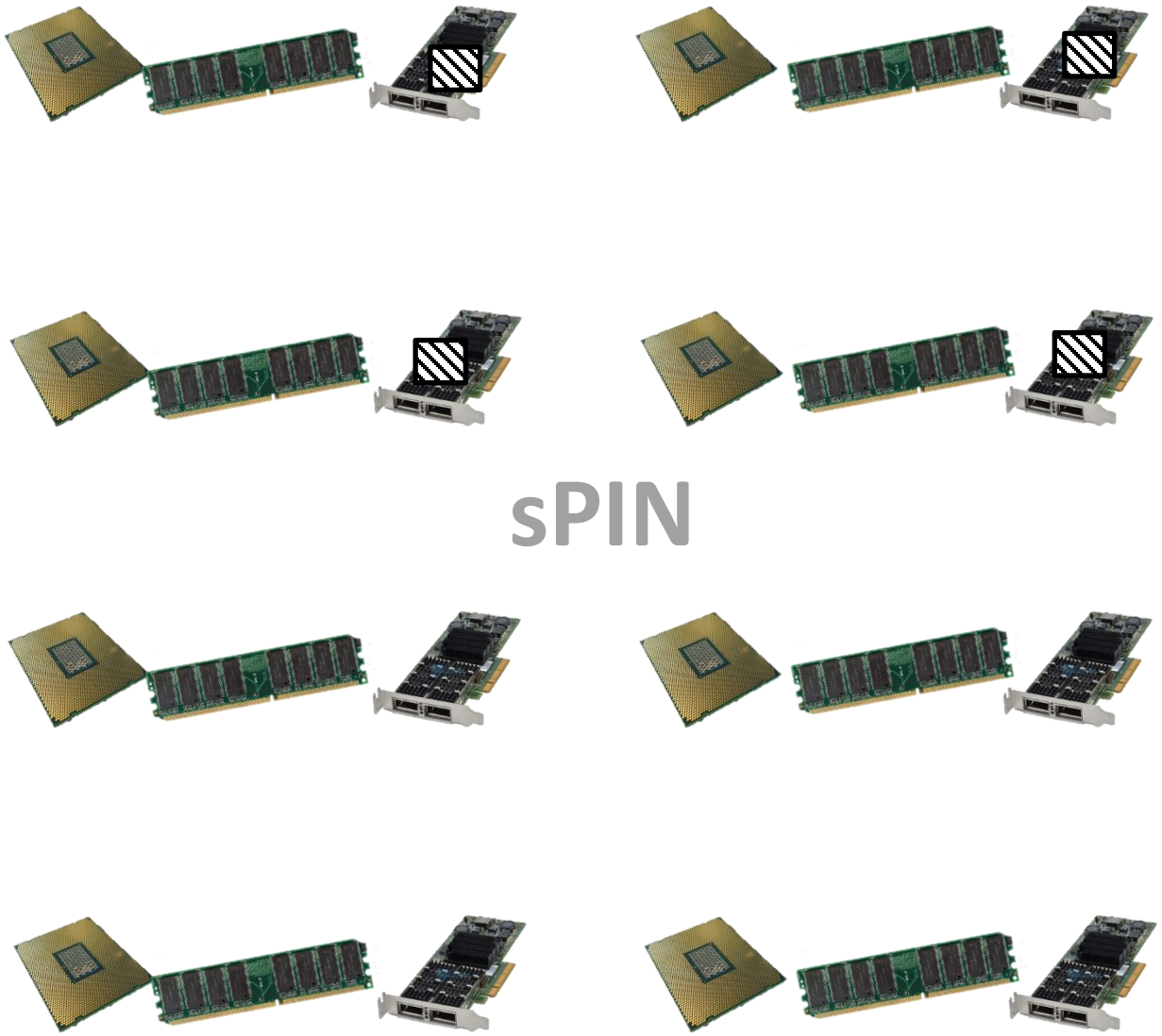


Network Group Communication

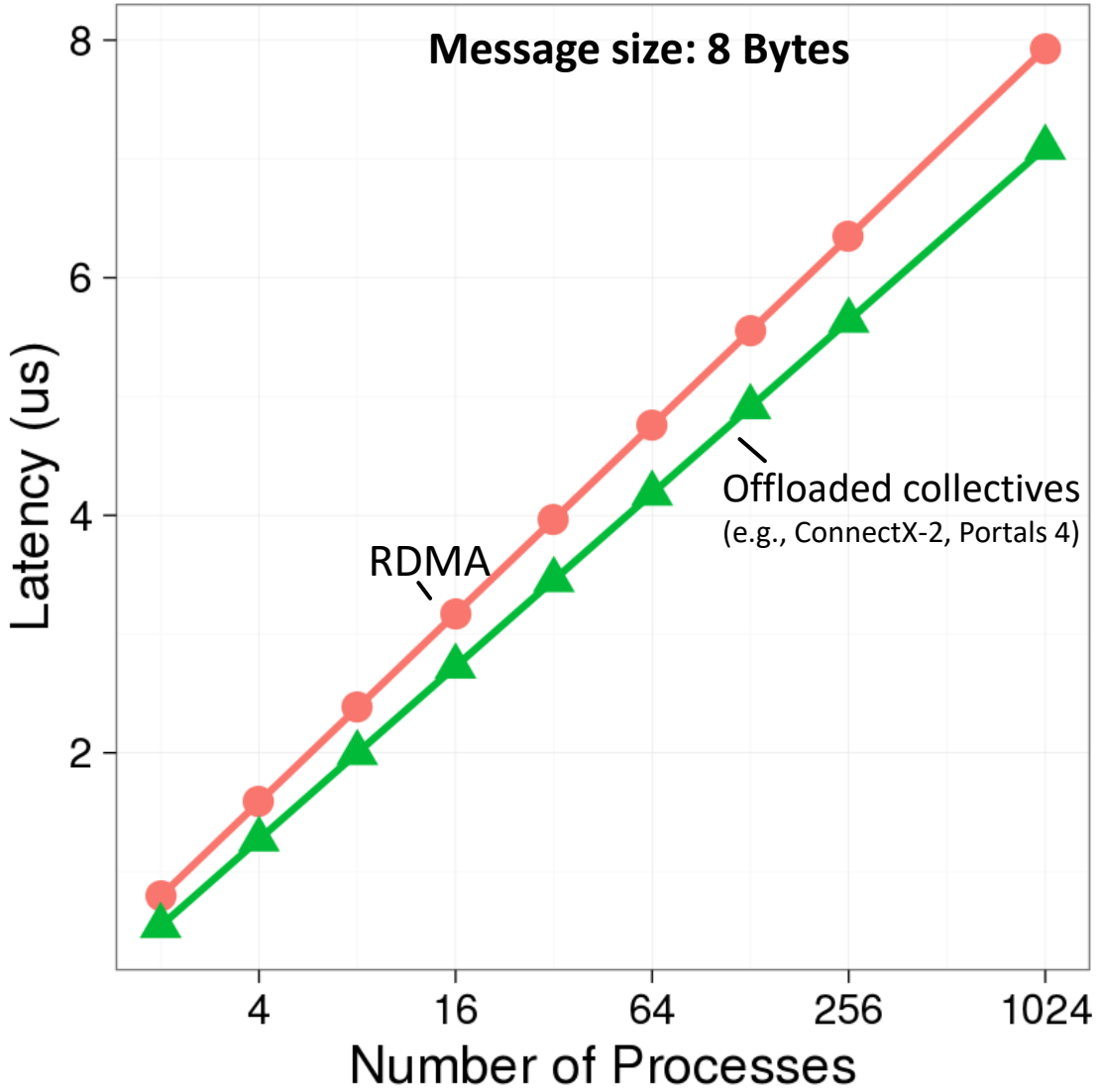


Underwood, K.D., et al., Enabling flexible collective communication offload with triggered operations. *HOTI'11*
Liu, J., et al., High performance RDMA-based MPI implementation over InfiniBand. *International Journal of Parallel Programming* 2004

Use Case 1: Broadcast acceleration

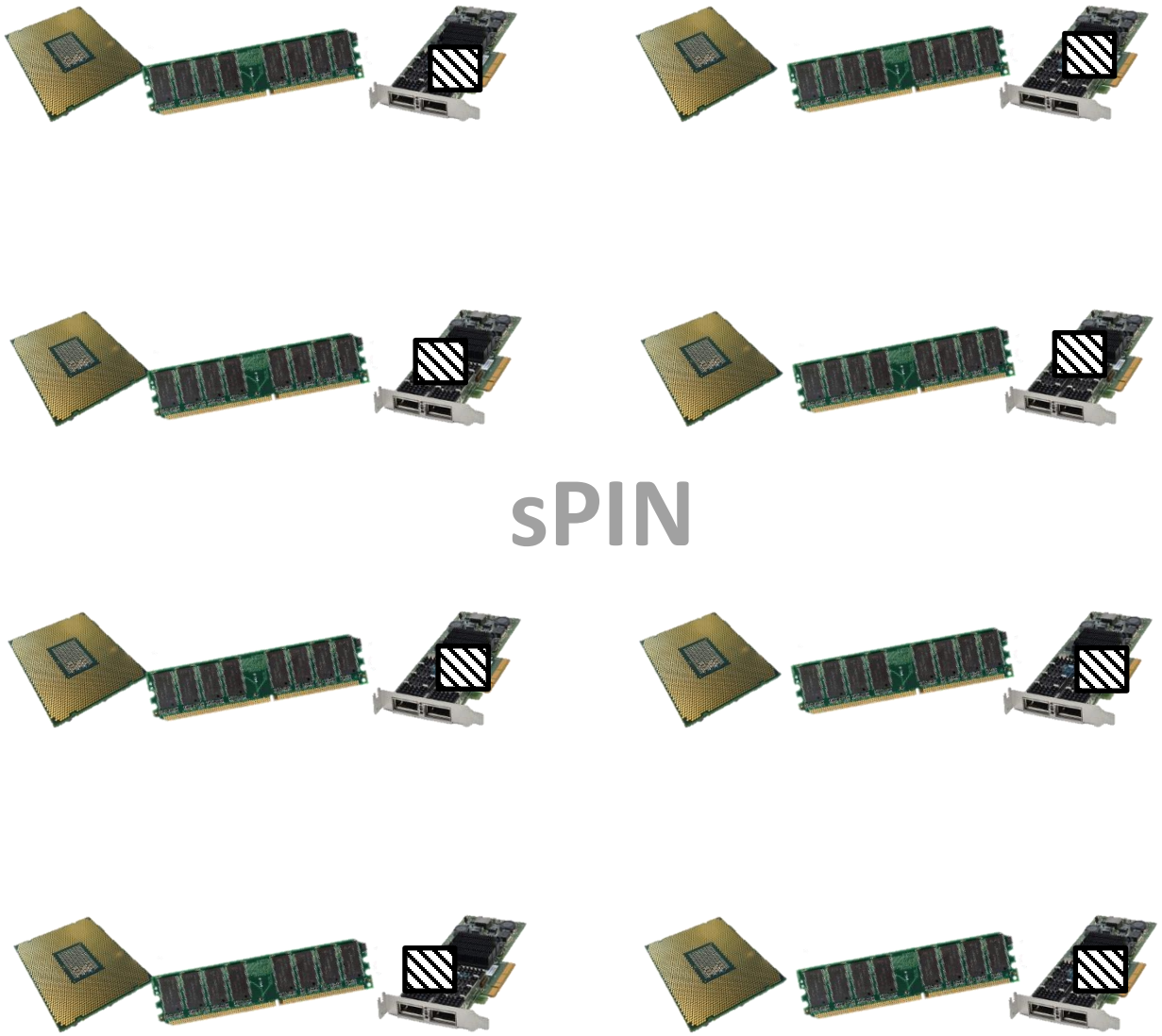


Network Group Communication

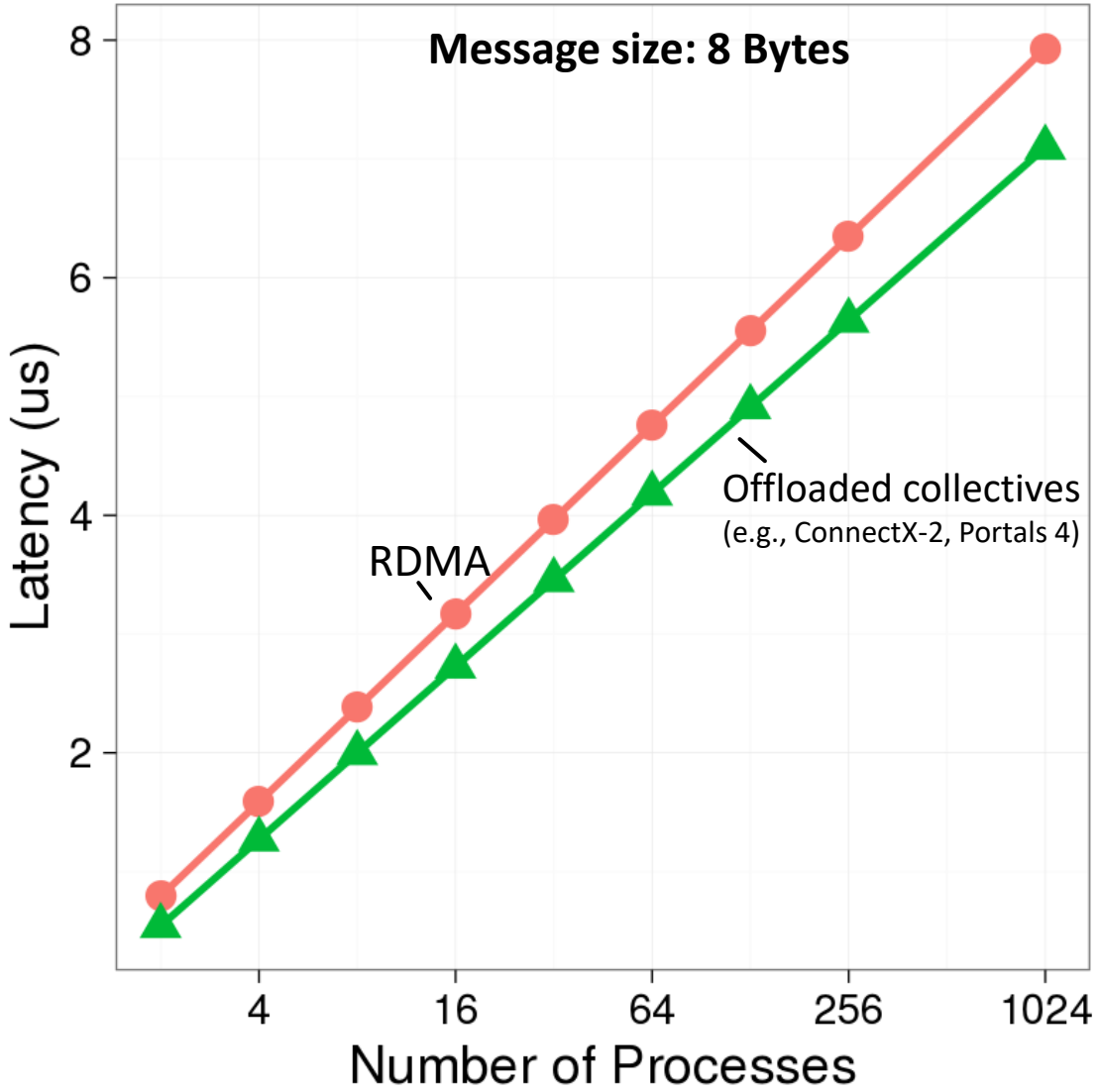


Underwood, K.D., et al., Enabling flexible collective communication offload with triggered operations. *HOTI'11*
Liu, J., et al., High performance RDMA-based MPI implementation over InfiniBand. *International Journal of Parallel Programming* 2004

Use Case 1: Broadcast acceleration

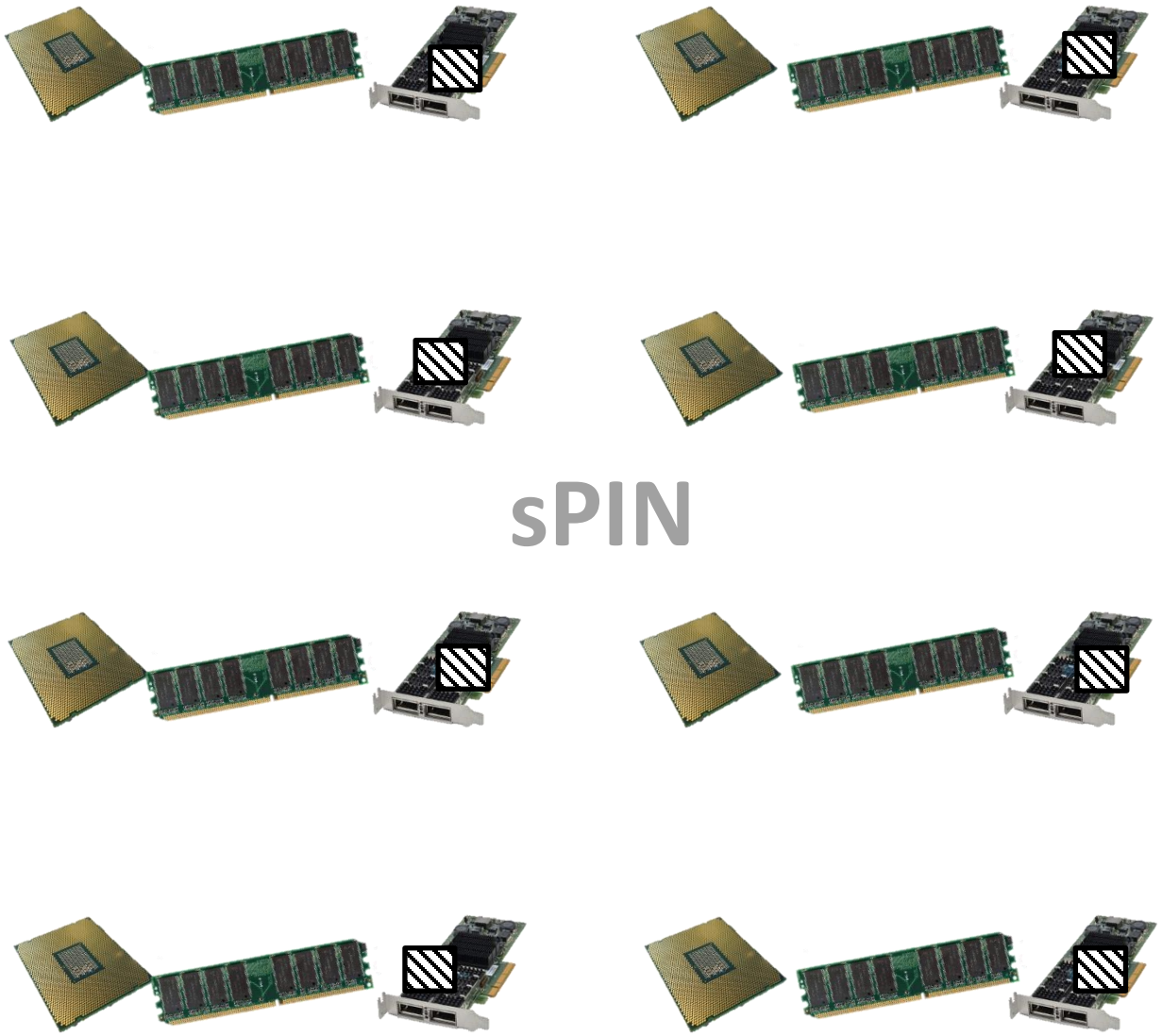


Network Group Communication

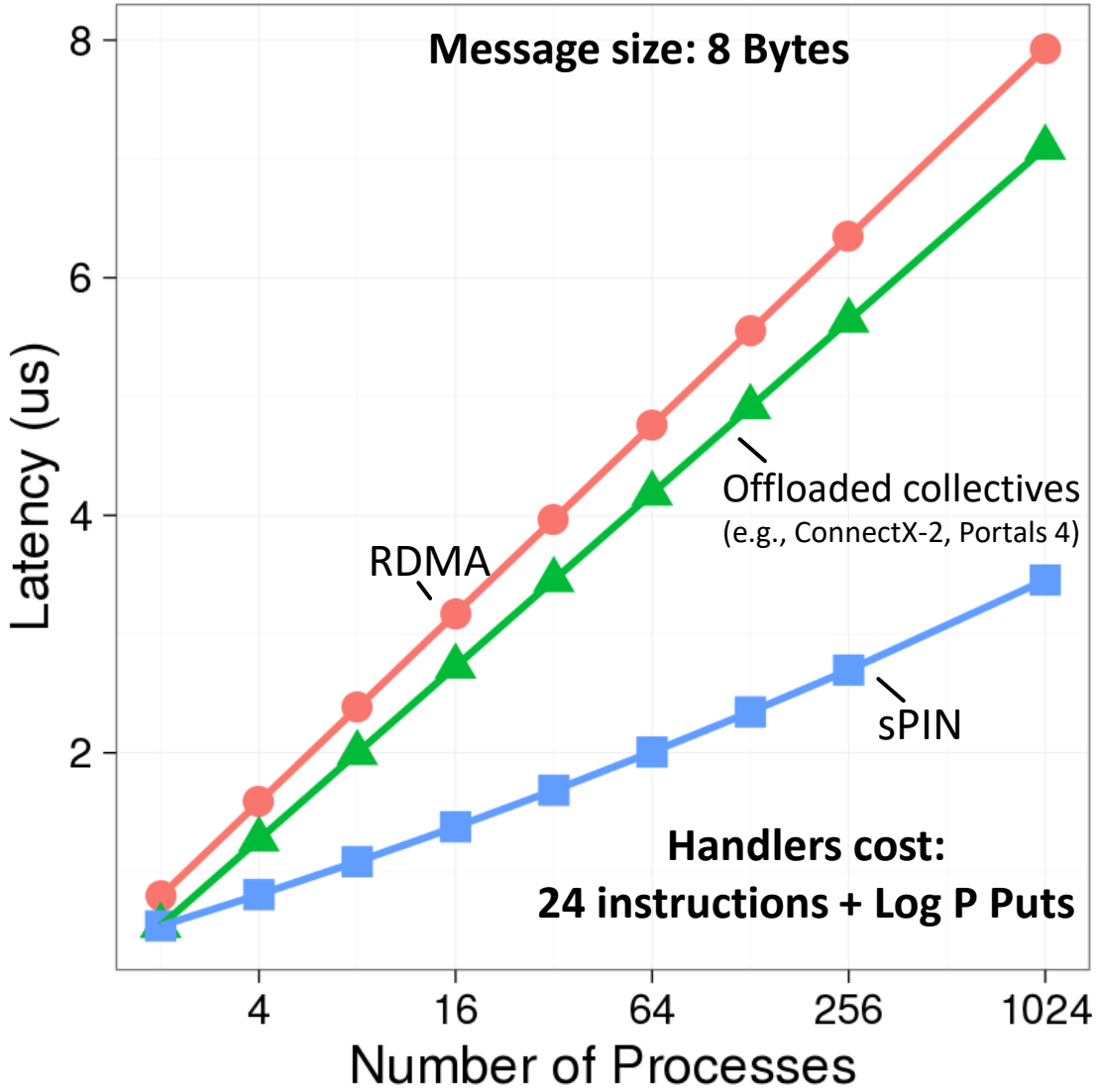


Underwood, K.D., et al., Enabling flexible collective communication offload with triggered operations. *HOTI'11*
Liu, J., et al., High performance RDMA-based MPI implementation over InfiniBand. *International Journal of Parallel Programming* 2004

Use Case 1: Broadcast acceleration



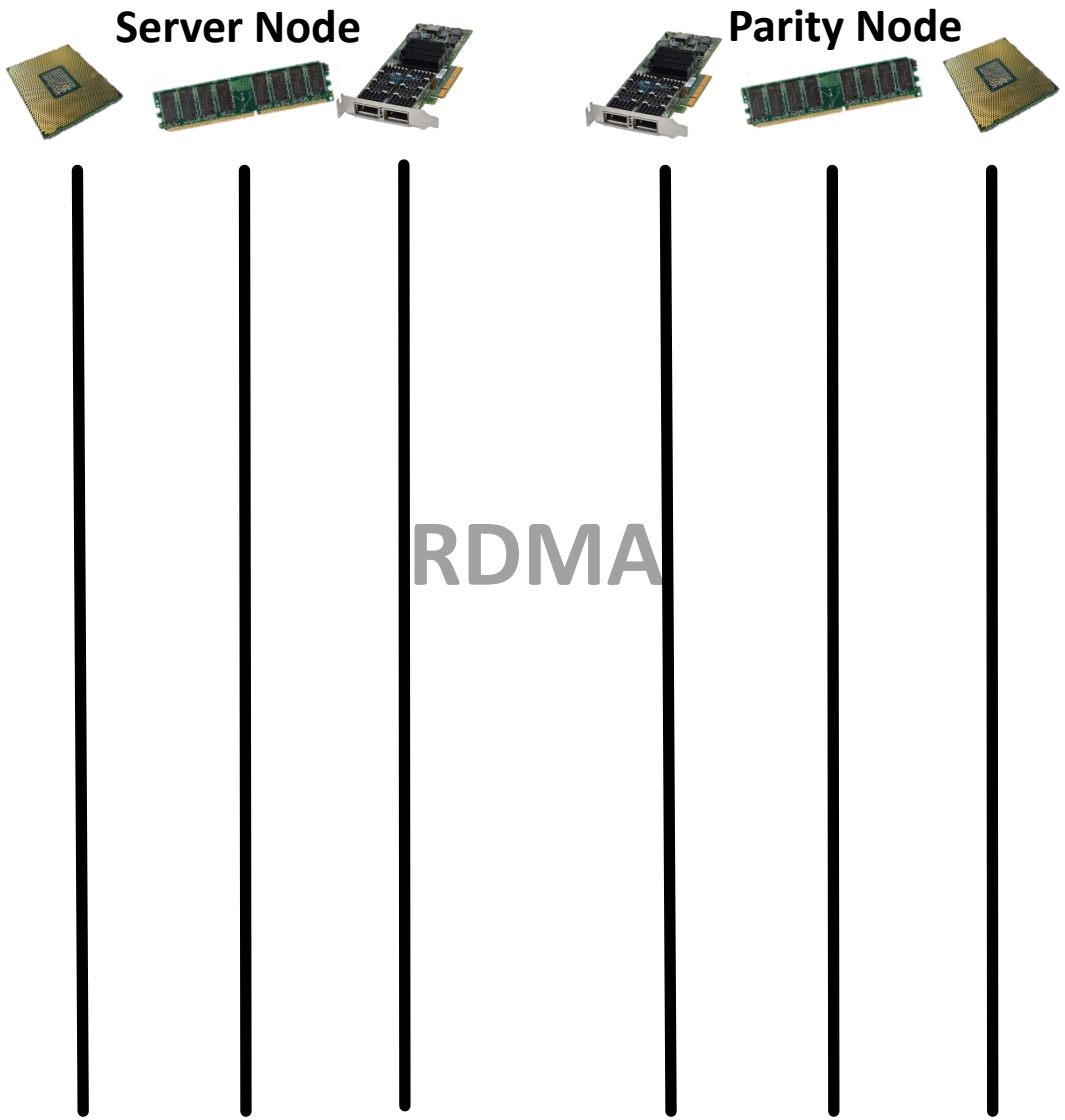
Network Group Communication



Underwood, K.D., et al., Enabling flexible collective communication offload with triggered operations. *HOTI'11*
Liu, J., et al., High performance RDMA-based MPI implementation over InfiniBand. *International Journal of Parallel Programming* 2004

Use Case 2: RAID acceleration

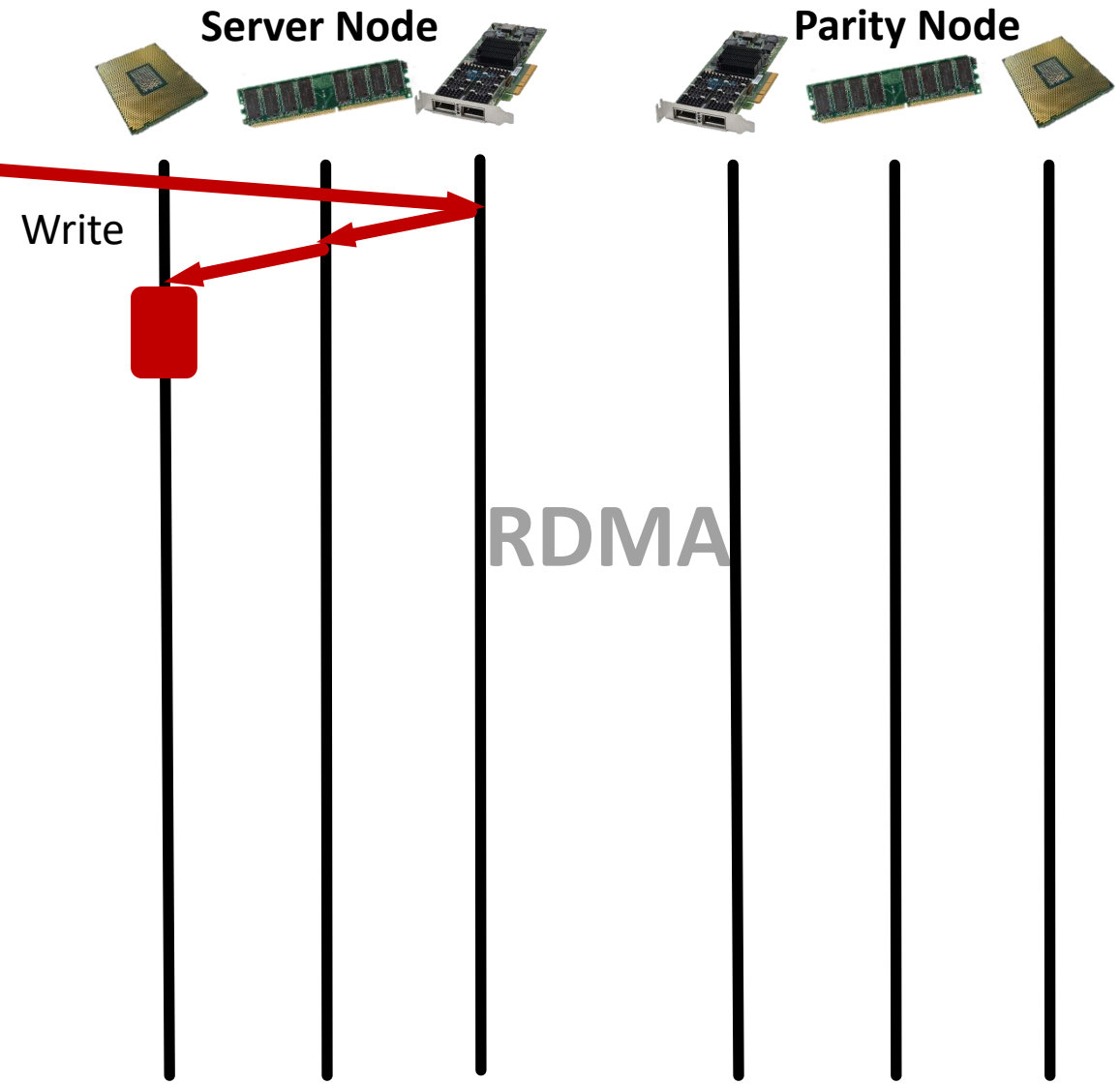
Distributed Data Management



Shankar D. et al., High-performance and Resilient Key-Value Store with Online Erasure Coding for Big Data Workloads. ICDCS'17

Use Case 2: RAID acceleration

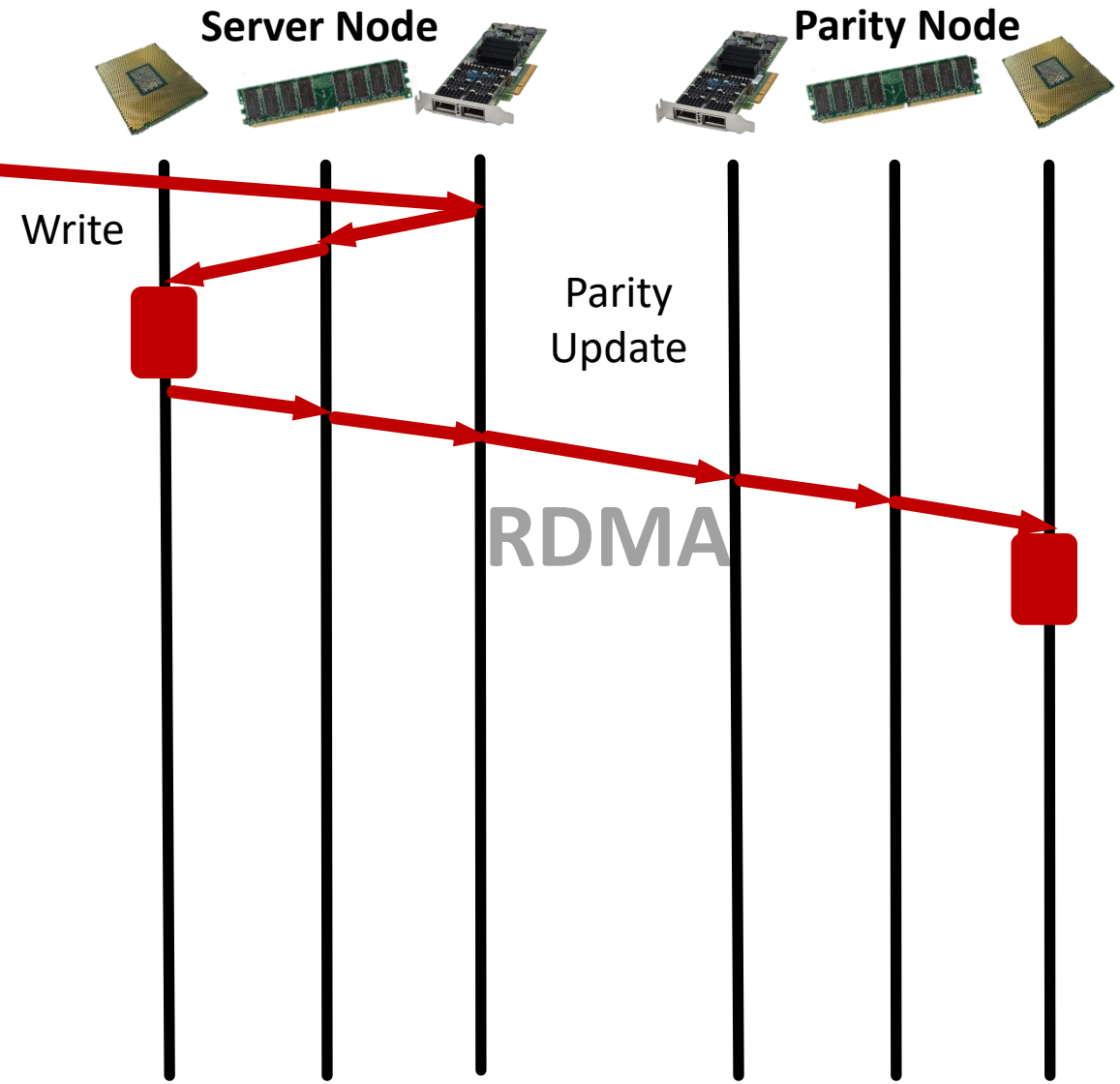
Distributed Data Management



Shankar D. et al., High-performance and Resilient Key-Value Store with Online Erasure Coding for Big Data Workloads. ICDCS'17

Use Case 2: RAID acceleration

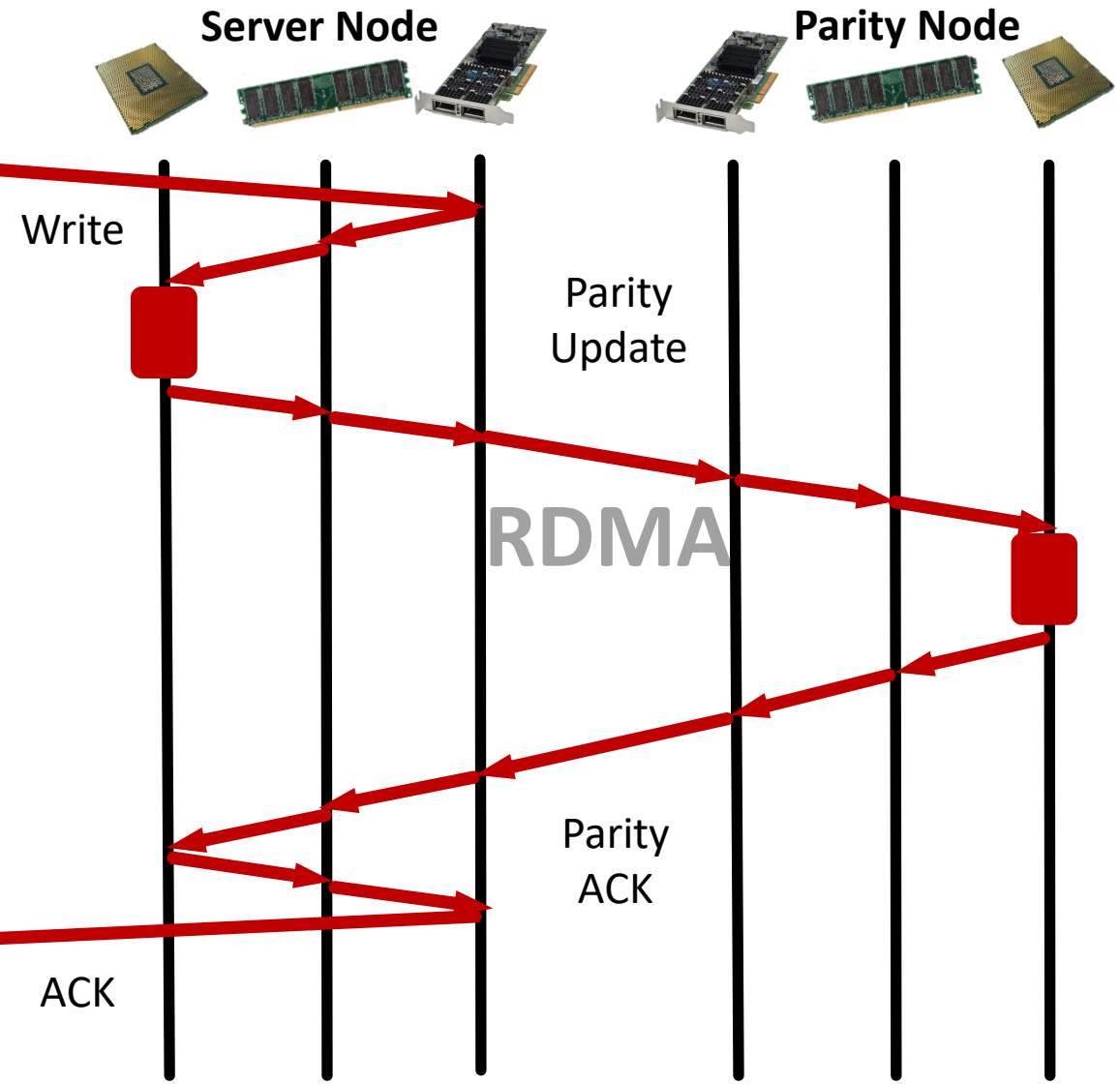
Distributed Data Management



Shankar D. et al., High-performance and Resilient Key-Value Store with Online Erasure Coding for Big Data Workloads. ICDCS'17

Use Case 2: RAID acceleration

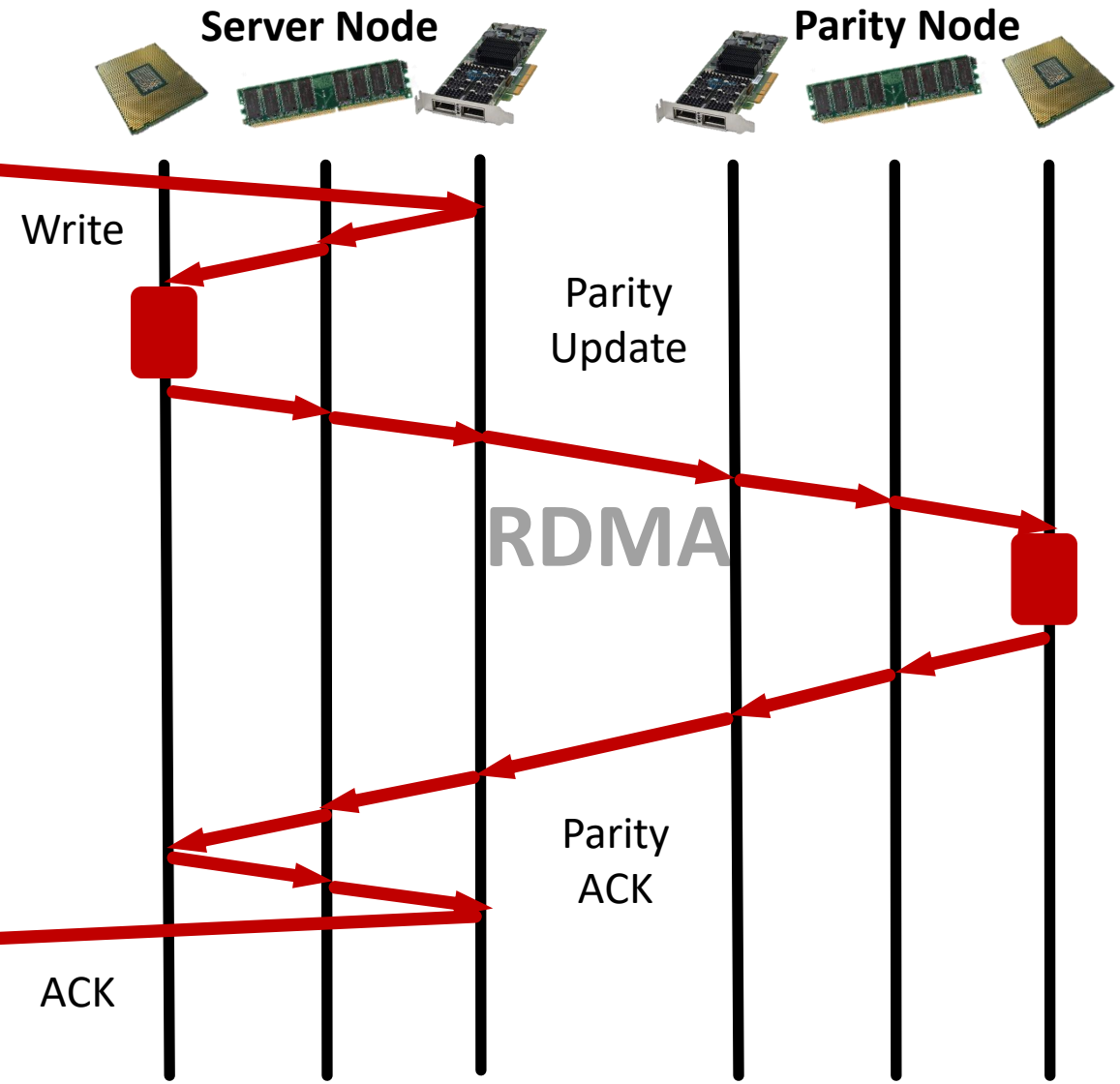
Distributed Data Management

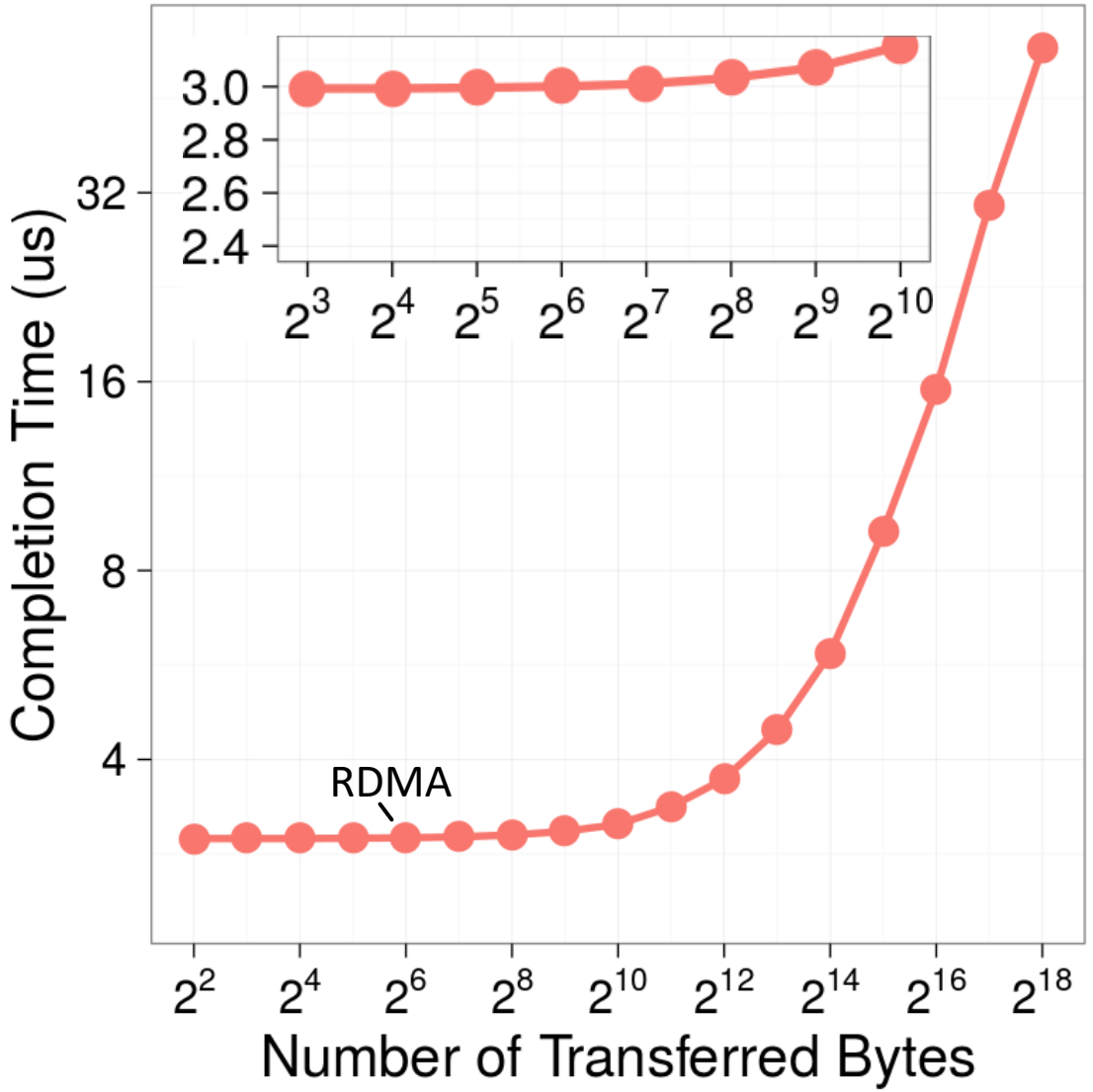


Shankar D. et al., High-performance and Resilient Key-Value Store with Online Erasure Coding for Big Data Workloads. ICDCS'17

Use Case 2: RAID acceleration

Distributed Data Management

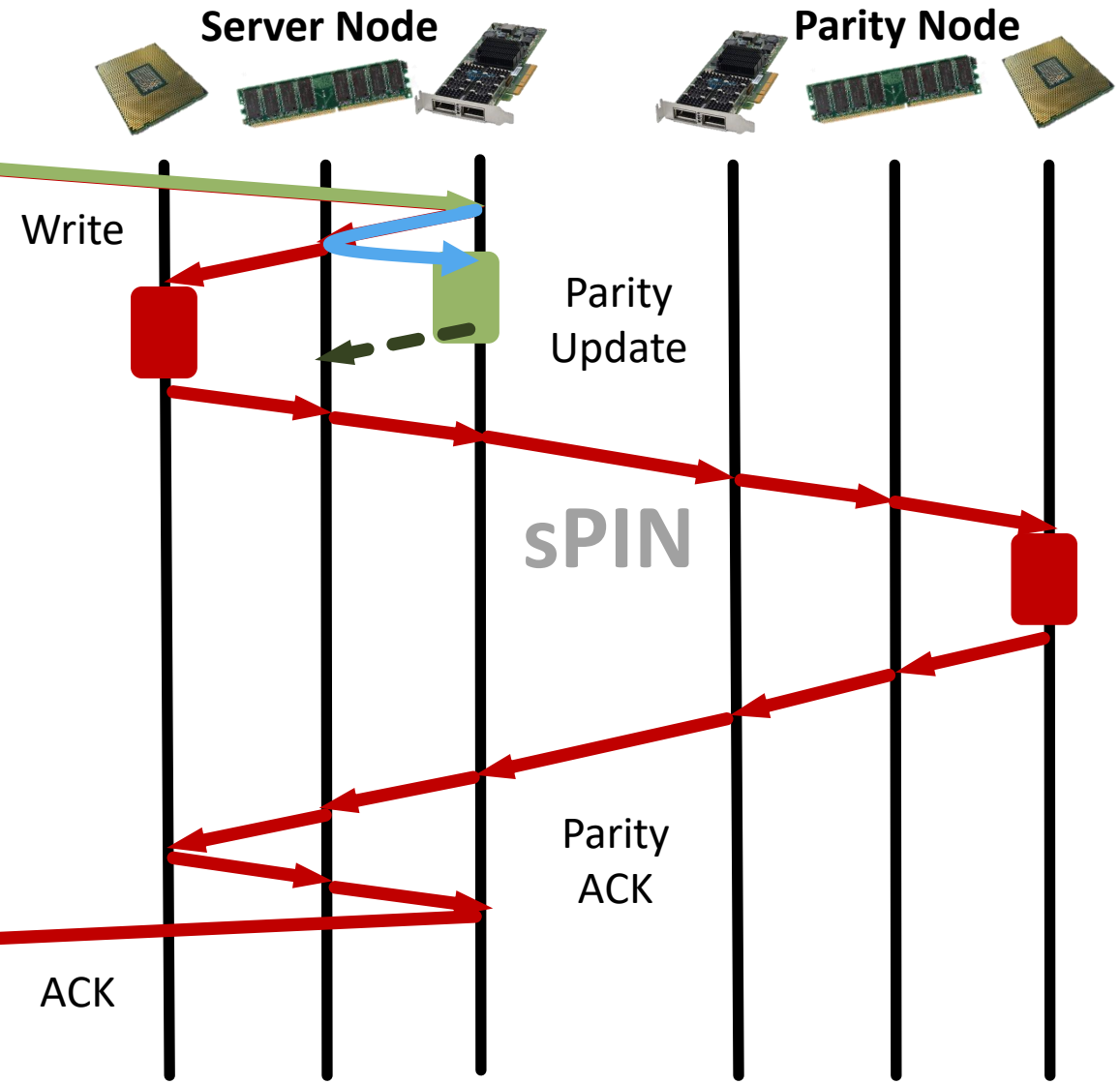


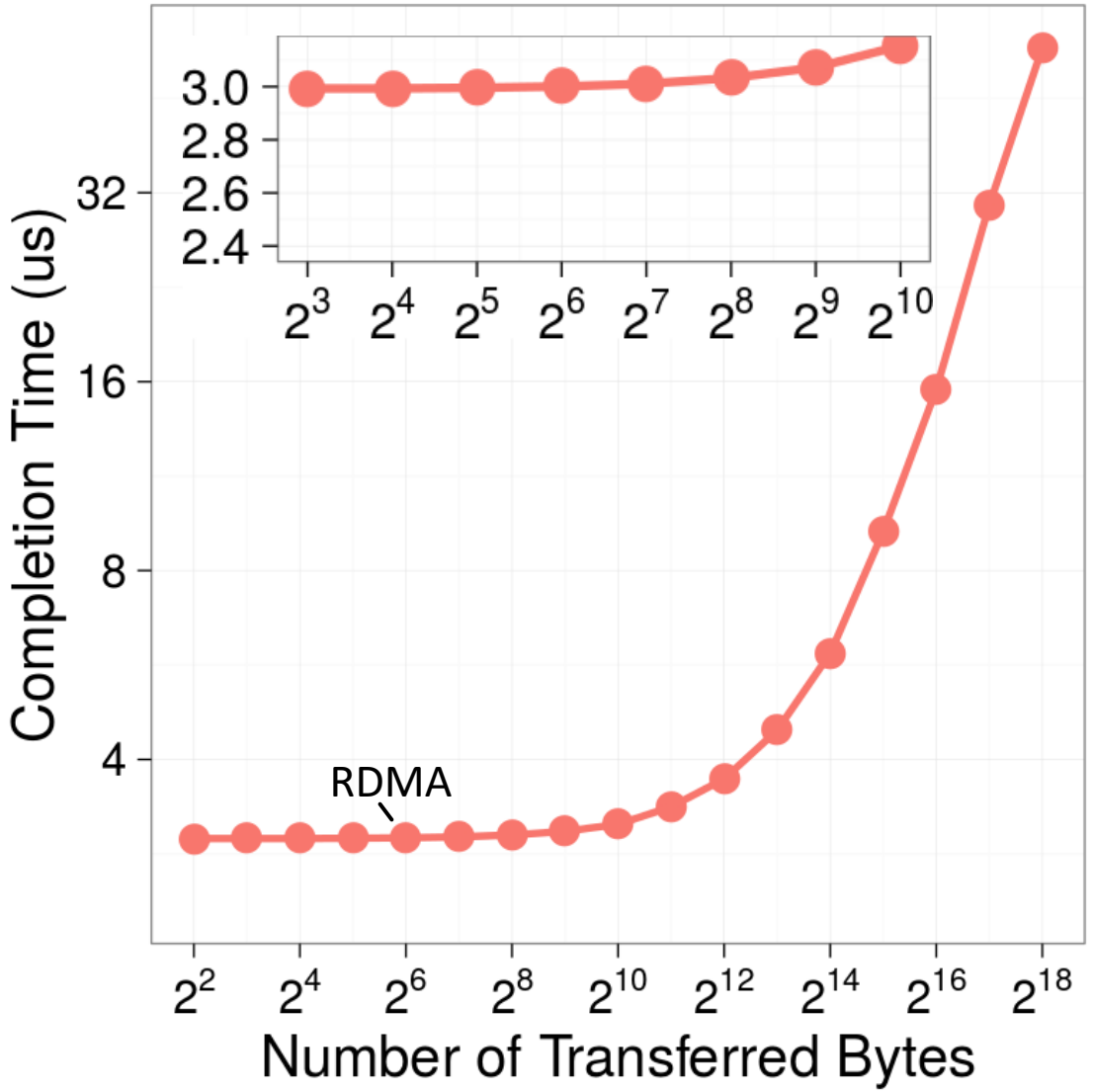


Shankar D. et al., High-performance and Resilient Key-Value Store with Online Erasure Coding for Big Data Workloads. ICDCS'17

Use Case 2: RAID acceleration

Distributed Data Management

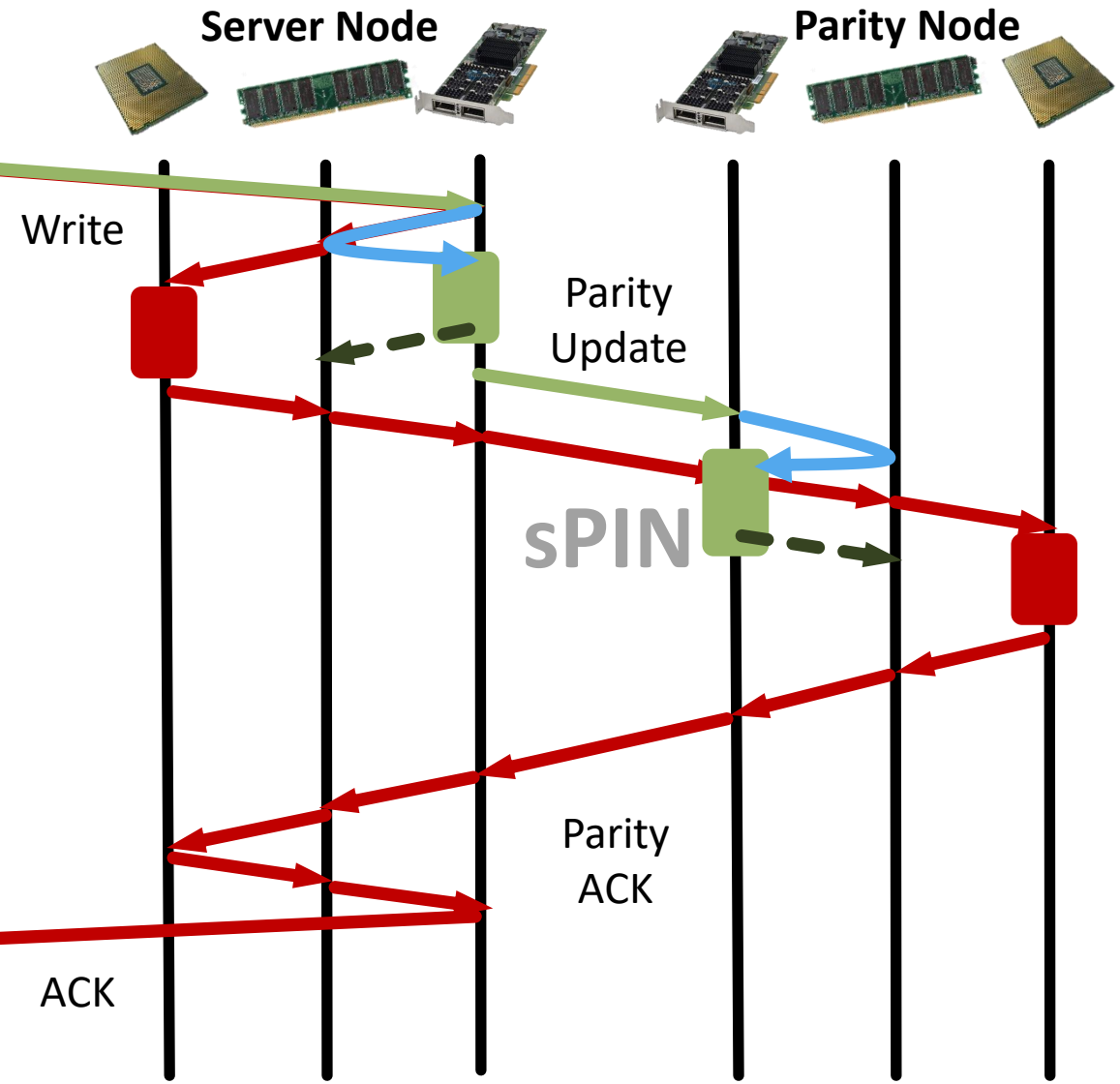


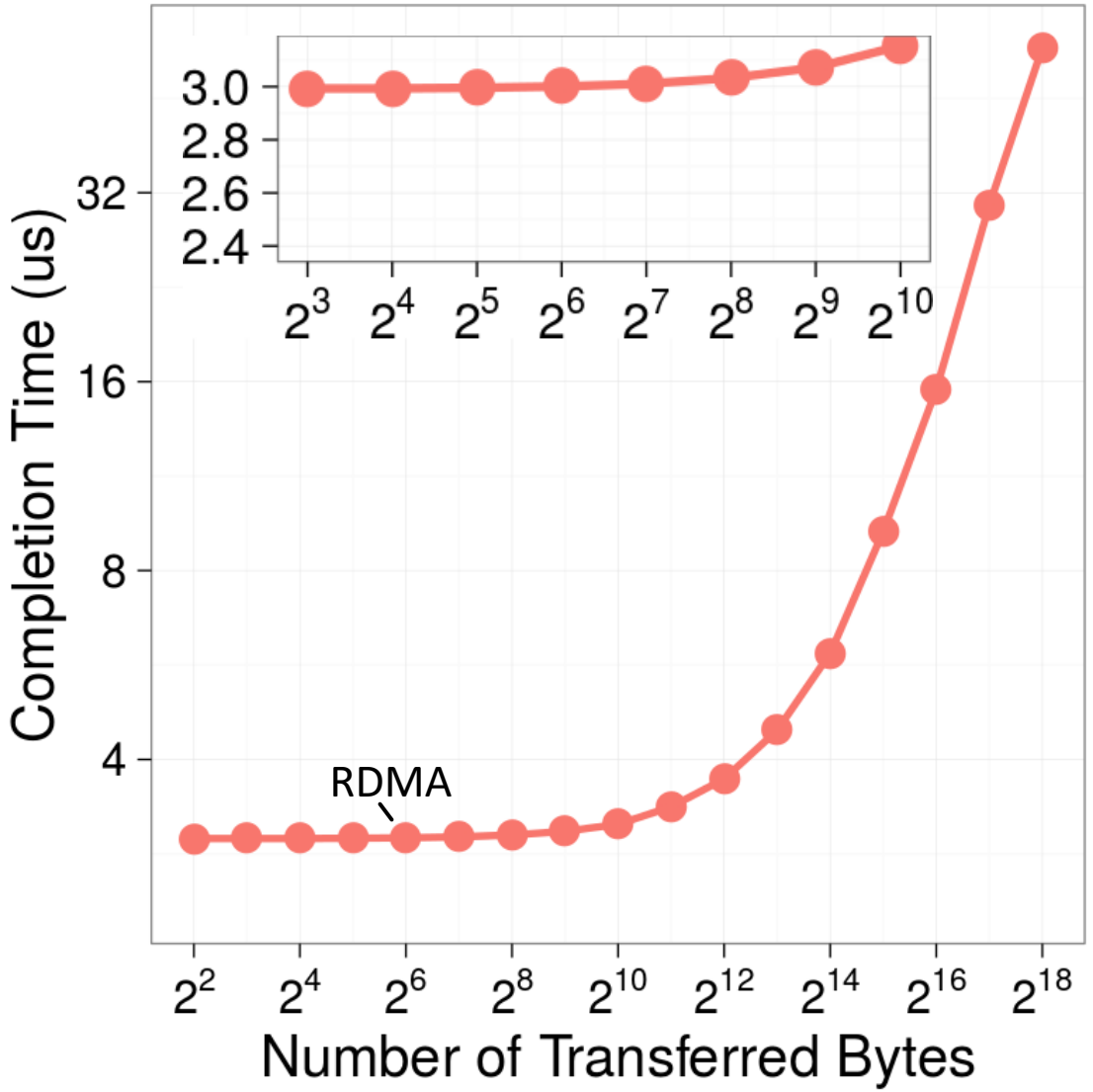


Shankar D. et al., High-performance and Resilient Key-Value Store with Online Erasure Coding for Big Data Workloads. ICDCS'17

Use Case 2: RAID acceleration

Distributed Data Management

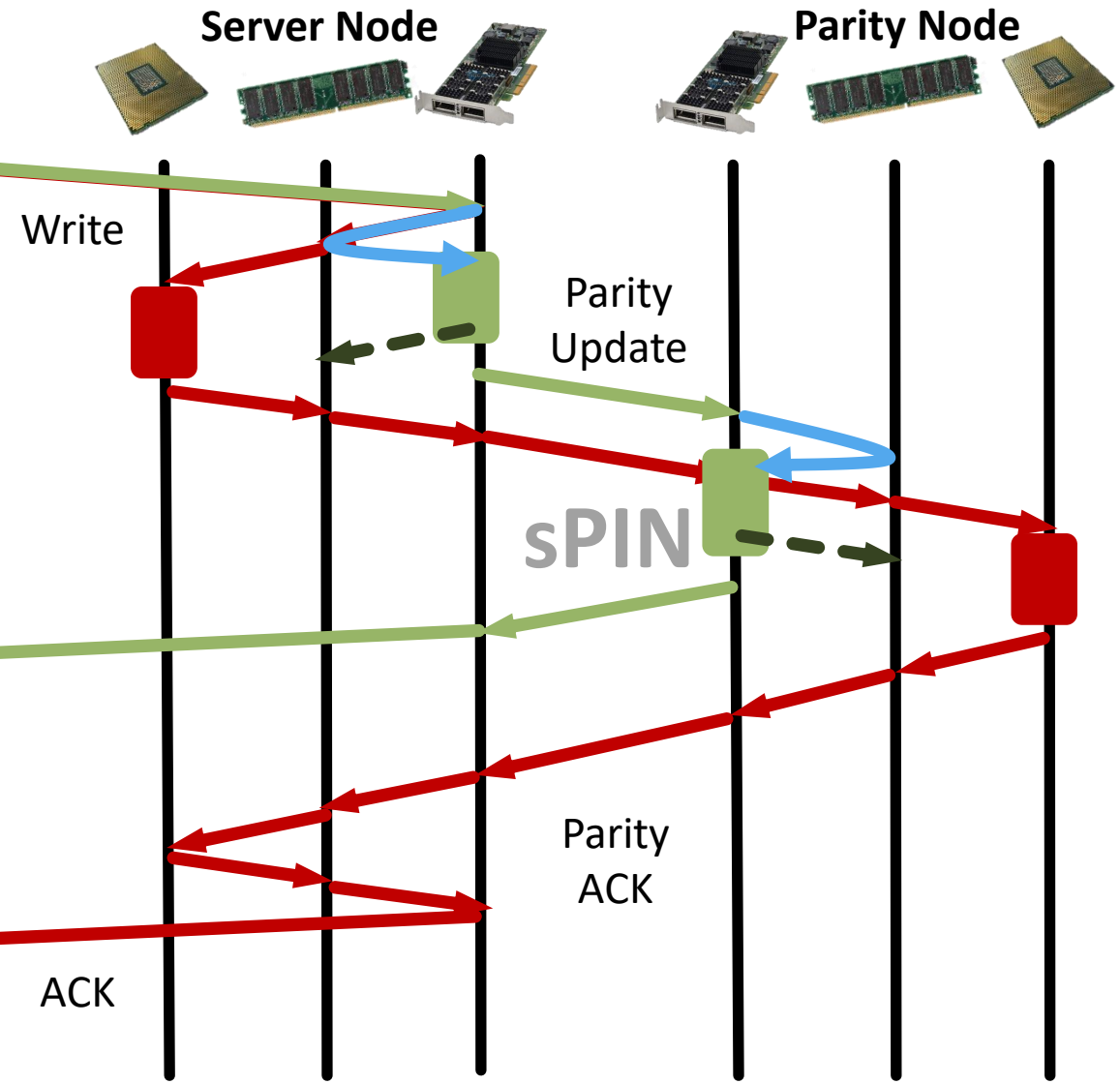


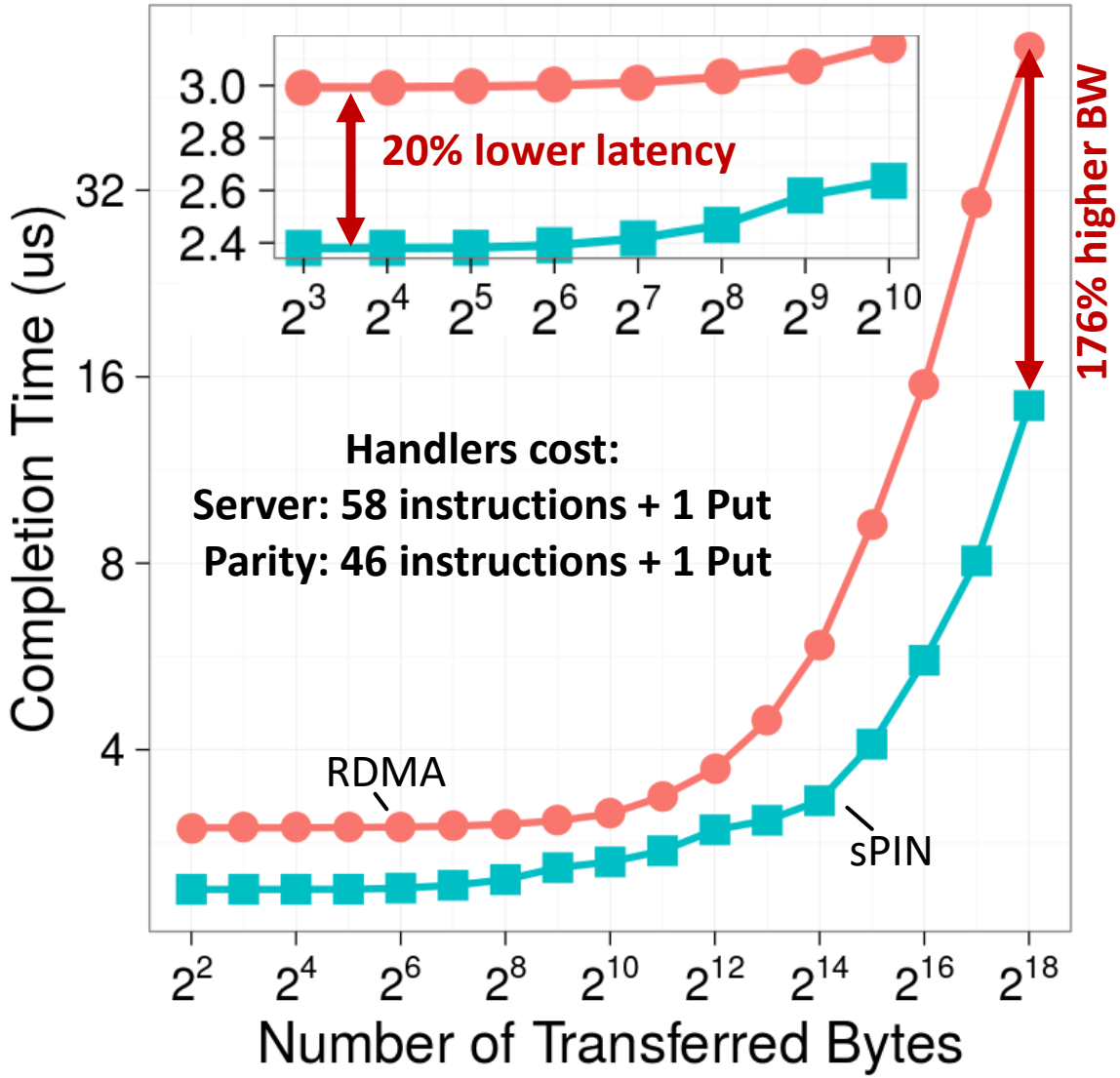


Shankar D. et al., High-performance and Resilient Key-Value Store with Online Erasure Coding for Big Data Workloads. ICDCS'17

Use Case 2: RAID acceleration

Distributed Data Management





Shankar D. et al., High-performance and Resilient Key-Value Store with Online Erasure Coding for Big Data Workloads. ICDCS'17

Further results and use-cases

Use Case 4: MPI Rendezvous Protocol

program	p	msgs	ovhd	ovhd	red
MILC	64	5.7M	5.5%	1.9%	65%
POP	64	772M	3.1%	2.4%	22%
coMD	72	5.3M	6.1%	2.4%	60%
coMD	360	28.1M	6.5%	2.8%	58%
Cloverleaf	72	2.7M	5.2%	2.4%	53%
Cloverleaf	360	15.3M	5.6%	3.2%	42%

Further results and use-cases

Use Case 4: MPI Rendezvous Protocol

program	p	msgs	ovhd	ovhd	red
MILC	64	5.7M	5.5%	1.9%	65%
POP	64	772M	3.1%	2.4%	22%
coMD	72	5.3M	6.1%	2.4%	60%
coMD	360	28.1M	6.5%	2.8%	58%
Cloverleaf	72	2.7M	5.2%	2.4%	53%
Cloverleaf	360	15.3M	5.6%	3.2%	42%

Further results and use-cases

Use Case 4: MPI Rendezvous Protocol

program	p	msgs	ovhd	ovhd	red
MILC	64	5.7M	5.5%	1.9%	65%
POP	64	772M	3.1%	2.4%	22%
coMD	72	5.3M	6.1%	2.4%	60%
coMD	360	28.1M	6.5%	2.8%	58%
Cloverleaf	72	2.7M	5.2%	2.4%	53%
Cloverleaf	360	15.3M	5.6%	3.2%	42%

Further results and use-cases

Use Case 4: MPI Rendezvous Protocol

program	p	msgs	ovhd	ovhd	red
MILC	64	5.7M	5.5%	1.9%	65%
POP	64	772M	3.1%	2.4%	22%
coMD	72	5.3M	6.1%	2.4%	60%
coMD	360	28.1M	6.5%	2.8%	58%
Cloverleaf	72	2.7M	5.2%	2.4%	53%
Cloverleaf	360	15.3M	5.6%	3.2%	42%

Use Case 5: Distributed KV Store

Kalia, A., et al., Using RDMA efficiently for key-value services. In *ACM SIGCOMM Computer Communication Review, 2014*

Further results and use-cases

Use Case 4: MPI Rendezvous Protocol

program	p	msgs	ovhd	ovhd	red
MILC	64	5.7M	5.5%	1.9%	65%
POP	64	772M	3.1%	2.4%	22%
coMD	72	5.3M	6.1%	2.4%	60%
coMD	360	28.1M	6.5%	2.8%	58%
Cloverleaf	72	2.7M	5.2%	2.4%	53%
Cloverleaf	360	15.3M	5.6%	3.2%	42%

Use Case 5: Distributed KV Store

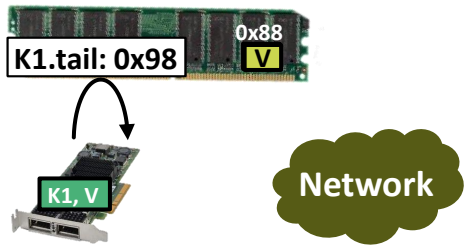
Kalia, A., et al., Using RDMA efficiently for key-value services. In *ACM SIGCOMM Computer Communication Review, 2014*

Further results and use-cases

Use Case 4: MPI Rendezvous Protocol

program	p	msgs	ovhd	ovhd	red
MILC	64	5.7M	5.5%	1.9%	65%
POP	64	772M	3.1%	2.4%	22%
coMD	72	5.3M	6.1%	2.4%	60%
coMD	360	28.1M	6.5%	2.8%	58%
Cloverleaf	72	2.7M	5.2%	2.4%	53%
Cloverleaf	360	15.3M	5.6%	3.2%	42%

Use Case 5: Distributed KV Store



41% lower latency

Kalia, A., et al., Using RDMA efficiently for key-value services. In *ACM SIGCOMM Computer Communication Review, 2014*

Further results and use-cases

Use Case 4: MPI Rendezvous Protocol

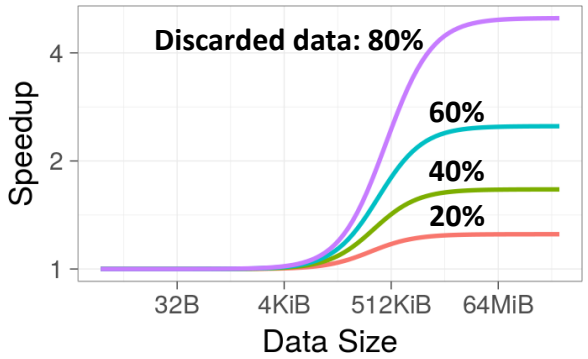
program	p	msgs	ovhd	ovhd	red
MILC	64	5.7M	5.5%	1.9%	65%
POP	64	772M	3.1%	2.4%	22%
coMD	72	5.3M	6.1%	2.4%	60%
coMD	360	28.1M	6.5%	2.8%	58%
Cloverleaf	72	2.7M	5.2%	2.4%	53%
Cloverleaf	360	15.3M	5.6%	3.2%	42%

Use Case 5: Distributed KV Store

41% lower latency

Kalia, A., et al., Using RDMA efficiently for key-value services. In *ACM SIGCOMM Computer Communication Review, 2014*

Use Case 6: Conditional Read



Barthels, C., et al., Designing Databases for Future High-Performance Networks. *IEEE Data Eng. Bulletin, 2017*

Further results and use-cases

Use Case 4: MPI Rendezvous Protocol

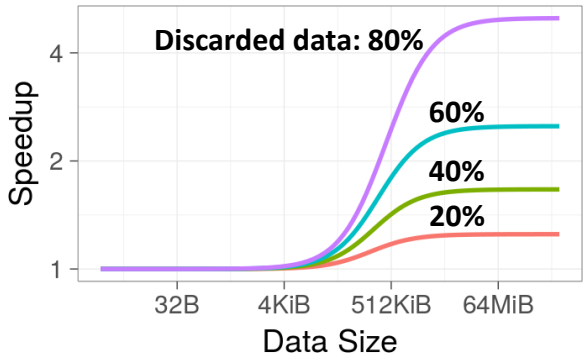
program	p	msgs	ovhd	ovhd	red
MILC	64	5.7M	5.5%	1.9%	65%
POP	64	772M	3.1%	2.4%	22%
coMD	72	5.3M	6.1%	2.4%	60%
coMD	360	28.1M	6.5%	2.8%	58%
Cloverleaf	72	2.7M	5.2%	2.4%	53%
Cloverleaf	360	15.3M	5.6%	3.2%	42%

Use Case 5: Distributed KV Store

41% lower latency

Kalia, A., et al., Using RDMA efficiently for key-value services. In *ACM SIGCOMM Computer Communication Review, 2014*

Use Case 6: Conditional Read



Barthels, C., et al., Designing Databases for Future High-Performance Networks. *IEEE Data Eng. Bulletin, 2017*

Further results and use-cases

Use Case 4: MPI Rendezvous Protocol

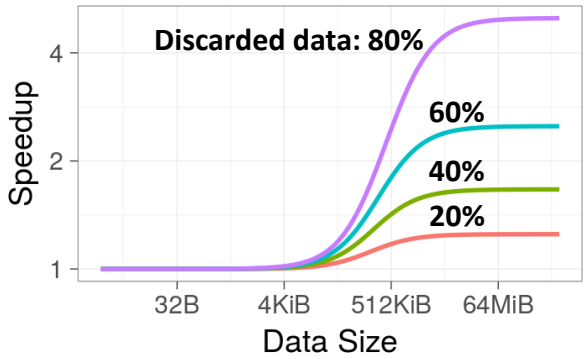
program	p	msgs	ovhd	ovhd	red
MILC	64	5.7M	5.5%	1.9%	65%
POP	64	772M	3.1%	2.4%	22%
coMD	72	5.3M	6.1%	2.4%	60%
coMD	360	28.1M	6.5%	2.8%	58%
Cloverleaf	72	2.7M	5.2%	2.4%	53%
Cloverleaf	360	15.3M	5.6%	3.2%	42%

Use Case 5: Distributed KV Store

41% lower latency

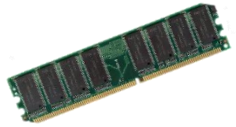
Kalia, A., et al., Using RDMA efficiently for key-value services. In *ACM SIGCOMM Computer Communication Review, 2014*

Use Case 6: Conditional Read



Barthels, C., et al., Designing Databases for Future High-Performance Networks. *IEEE Data Eng. Bulletin, 2017*

Use Case 7: Distributed Transactions



Dragojević, A, et al., No compromises: distributed transactions with consistency, availability, and performance. SOSP'15

Further results and use-cases

Use Case 4: MPI Rendezvous Protocol

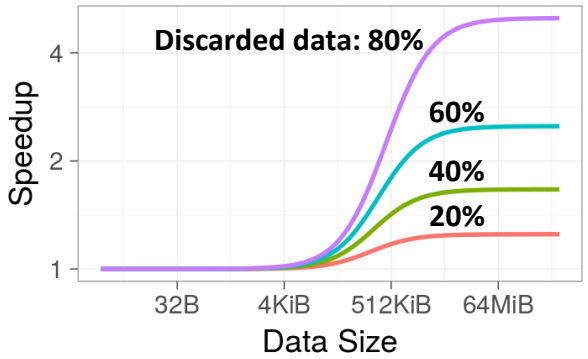
program	p	msgs	ovhd	ovhd	red
MILC	64	5.7M	5.5%	1.9%	65%
POP	64	772M	3.1%	2.4%	22%
coMD	72	5.3M	6.1%	2.4%	60%
coMD	360	28.1M	6.5%	2.8%	58%
Cloverleaf	72	2.7M	5.2%	2.4%	53%
Cloverleaf	360	15.3M	5.6%	3.2%	42%

Use Case 5: Distributed KV Store

41% lower latency

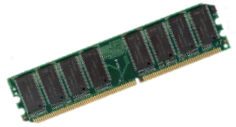
Kalia, A., et al., Using RDMA efficiently for key-value services. In *ACM SIGCOMM Computer Communication Review, 2014*

Use Case 6: Conditional Read



Barthels, C., et al., Designing Databases for Future High-Performance Networks. *IEEE Data Eng. Bulletin, 2017*

Use Case 7: Distributed Transactions



Dragojević, A, et al., No compromises: distributed transactions with consistency, availability, and performance. SOSP'15

Further results and use-cases

Use Case 4: MPI Rendezvous Protocol

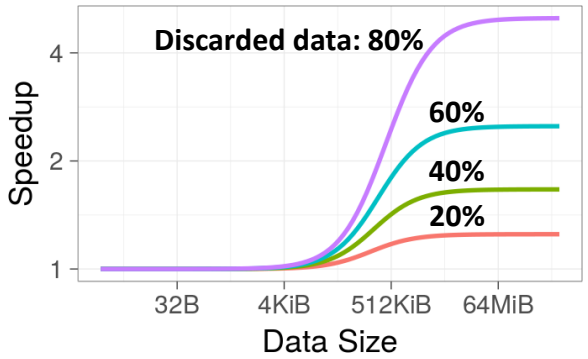
program	p	msgs	ovhd	ovhd	red
MILC	64	5.7M	5.5%	1.9%	65%
POP	64	772M	3.1%	2.4%	22%
coMD	72	5.3M	6.1%	2.4%	60%
coMD	360	28.1M	6.5%	2.8%	58%
Cloverleaf	72	2.7M	5.2%	2.4%	53%
Cloverleaf	360	15.3M	5.6%	3.2%	42%

Use Case 5: Distributed KV Store

41% lower latency

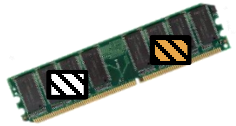
Kalia, A., et al., Using RDMA efficiently for key-value services. In *ACM SIGCOMM Computer Communication Review, 2014*

Use Case 6: Conditional Read



Barthels, C., et al., Designing Databases for Future High-Performance Networks. *IEEE Data Eng. Bulletin, 2017*

Use Case 7: Distributed Transactions



log pkts

data pkts

Dragojević, A, et al., No compromises: distributed transactions with consistency, availability, and performance. SOSP'15

Further results and use-cases

Use Case 4: MPI Rendezvous Protocol

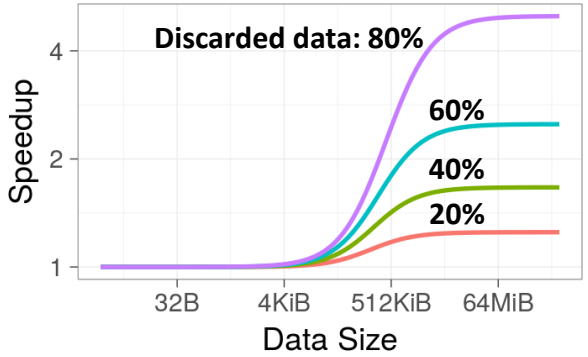
program	p	msgs	ovhd	ovhd	red
MILC	64	5.7M	5.5%	1.9%	65%
POP	64	772M	3.1%	2.4%	22%
coMD	72	5.3M	6.1%	2.4%	60%
coMD	360	28.1M	6.5%	2.8%	58%
Cloverleaf	72	2.7M	5.2%	2.4%	53%
Cloverleaf	360	15.3M	5.6%	3.2%	42%

Use Case 5: Distributed KV Store

41% lower latency

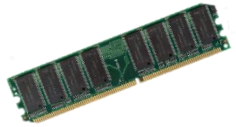
Kalia, A., et al., Using RDMA efficiently for key-value services. In *ACM SIGCOMM Computer Communication Review, 2014*

Use Case 6: Conditional Read



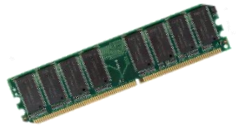
Barthels, C., et al., Designing Databases for Future High-Performance Networks. *IEEE Data Eng. Bulletin, 2017*

Use Case 7: Distributed Transactions



Dragojević, A, et al., No compromises: distributed transactions with consistency, availability, and performance. SOSP'15

Use Case 8: FT Broadcast



Bosilca, G., et al., Failure Detection and Propagation in HPC systems. SC'16

Further results and use-cases

Use Case 4: MPI Rendezvous Protocol

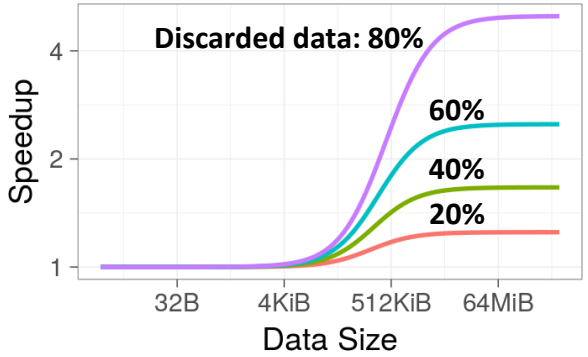
program	p	msgs	ovhd	ovhd	red
MILC	64	5.7M	5.5%	1.9%	65%
POP	64	772M	3.1%	2.4%	22%
coMD	72	5.3M	6.1%	2.4%	60%
coMD	360	28.1M	6.5%	2.8%	58%
Cloverleaf	72	2.7M	5.2%	2.4%	53%
Cloverleaf	360	15.3M	5.6%	3.2%	42%

Use Case 5: Distributed KV Store

41% lower latency

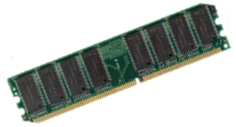
Kalia, A., et al., Using RDMA efficiently for key-value services. In *ACM SIGCOMM Computer Communication Review, 2014*

Use Case 6: Conditional Read



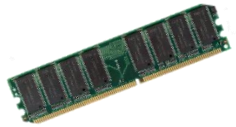
Barthels, C., et al., Designing Databases for Future High-Performance Networks. *IEEE Data Eng. Bulletin, 2017*

Use Case 7: Distributed Transactions



Dragojević, A, et al., No compromises: distributed transactions with consistency, availability, and performance. SOSP'15

Use Case 8: FT Broadcast



Bosilca, G., et al., Failure Detection and Propagation in HPC systems. SC'16

Further results and use-cases

Use Case 4: MPI Rendezvous Protocol

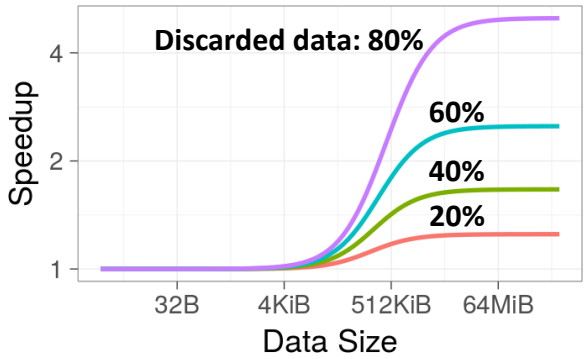
program	p	msgs	ovhd	ovhd	red
MILC	64	5.7M	5.5%	1.9%	65%
POP	64	772M	3.1%	2.4%	22%
coMD	72	5.3M	6.1%	2.4%	60%
coMD	360	28.1M	6.5%	2.8%	58%
Cloverleaf	72	2.7M	5.2%	2.4%	53%
Cloverleaf	360	15.3M	5.6%	3.2%	42%

Use Case 5: Distributed KV Store

41% lower latency

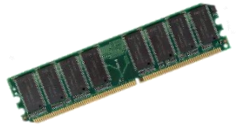
Kalia, A., et al., Using RDMA efficiently for key-value services. In *ACM SIGCOMM Computer Communication Review, 2014*

Use Case 6: Conditional Read



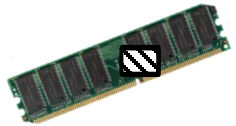
Barthels, C., et al., Designing Databases for Future High-Performance Networks. *IEEE Data Eng. Bulletin, 2017*

Use Case 7: Distributed Transactions



Dragojević, A, et al., No compromises: distributed transactions with consistency, availability, and performance. SOSP'15

Use Case 8: FT Broadcast



Bosilca, G., et al., Failure Detection and Propagation in HPC systems. SC'16

Further results and use-cases

Use Case 4: MPI Rendezvous Protocol

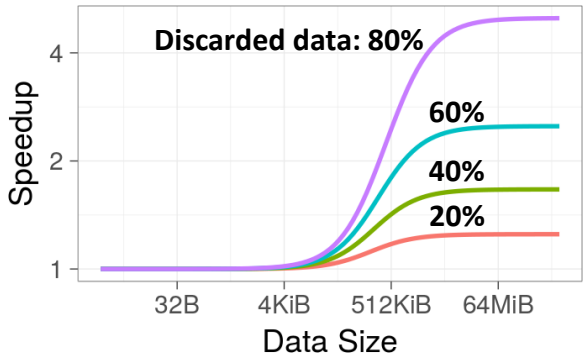
program	p	msgs	ovhd	ovhd	red
MILC	64	5.7M	5.5%	1.9%	65%
POP	64	772M	3.1%	2.4%	22%
coMD	72	5.3M	6.1%	2.4%	60%
coMD	360	28.1M	6.5%	2.8%	58%
Cloverleaf	72	2.7M	5.2%	2.4%	53%
Cloverleaf	360	15.3M	5.6%	3.2%	42%

Use Case 5: Distributed KV Store

41% lower latency

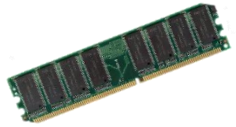
Kalia, A., et al., Using RDMA efficiently for key-value services. In *ACM SIGCOMM Computer Communication Review, 2014*

Use Case 6: Conditional Read



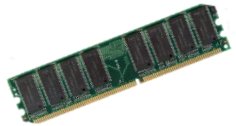
Barthels, C., et al., Designing Databases for Future High-Performance Networks. *IEEE Data Eng. Bulletin, 2017*

Use Case 7: Distributed Transactions



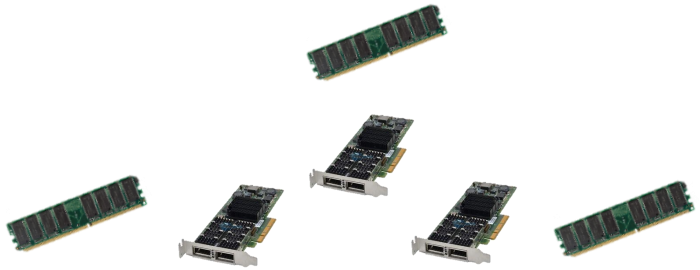
Dragojević, A, et al., No compromises: distributed transactions with consistency, availability, and performance. SOSP'15

Use Case 8: FT Broadcast



Bosilca, G., et al., Failure Detection and Propagation in HPC systems. SC'16

Use Case 9: Distributed Consensus



István, Z., et al., Consensus in a Box: Inexpensive Coordination in Hardware. NSDI'16

Further results and use-cases

Use Case 4: MPI Rendezvous Protocol

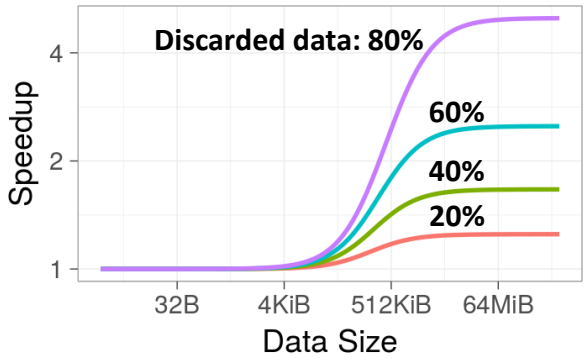
program	p	msgs	ovhd	ovhd	red
MILC	64	5.7M	5.5%	1.9%	65%
POP	64	772M	3.1%	2.4%	22%
coMD	72	5.3M	6.1%	2.4%	60%
coMD	360	28.1M	6.5%	2.8%	58%
Cloverleaf	72	2.7M	5.2%	2.4%	53%
Cloverleaf	360	15.3M	5.6%	3.2%	42%

Use Case 5: Distributed KV Store

41% lower latency

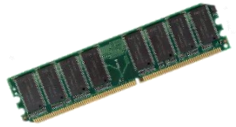
Kalia, A., et al., Using RDMA efficiently for key-value services. In *ACM SIGCOMM Computer Communication Review, 2014*

Use Case 6: Conditional Read



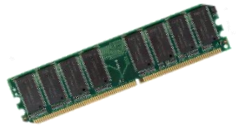
Barthels, C., et al., Designing Databases for Future High-Performance Networks. *IEEE Data Eng. Bulletin, 2017*

Use Case 7: Distributed Transactions



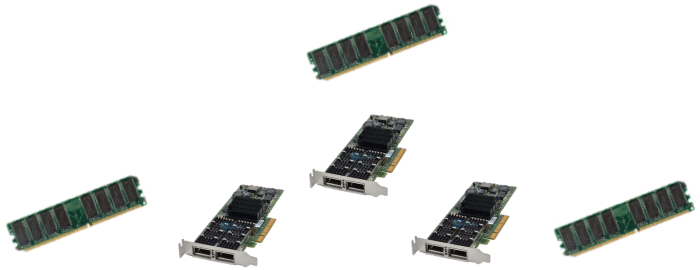
Dragojević, A, et al., No compromises: distributed transactions with consistency, availability, and performance. SOSP'15

Use Case 8: FT Broadcast



Bosilca, G., et al., Failure Detection and Propagation in HPC systems. SC'16

Use Case 9: Distributed Consensus



István, Z., et al., Consensus in a Box: Inexpensive Coordination in Hardware. NSDI'16

Further results and use-cases

Use Case 4: MPI Rendezvous Protocol

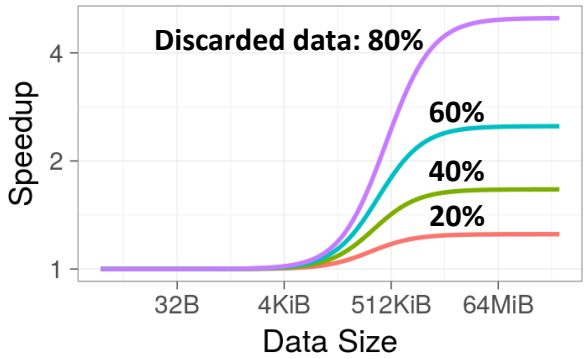
program	p	msgs	ovhd	ovhd	red
MILC	64	5.7M	5.5%	1.9%	65%
POP	64	772M	3.1%	2.4%	22%
coMD	72	5.3M	6.1%	2.4%	60%
coMD	360	28.1M	6.5%	2.8%	58%
Cloverleaf	72	2.7M	5.2%	2.4%	53%
Cloverleaf	360	15.3M	5.6%	3.2%	42%

Use Case 5: Distributed KV Store

41% lower latency

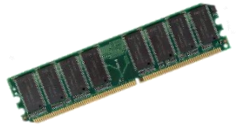
Kalia, A., et al., Using RDMA efficiently for key-value services. In *ACM SIGCOMM Computer Communication Review, 2014*

Use Case 6: Conditional Read



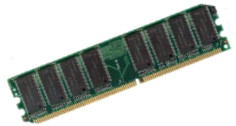
Barthels, C., et al., Designing Databases for Future High-Performance Networks. *IEEE Data Eng. Bulletin, 2017*

Use Case 7: Distributed Transactions



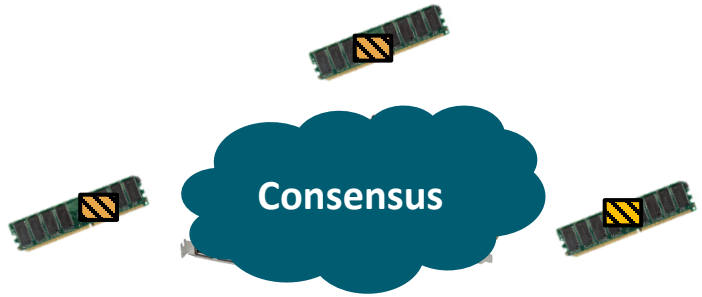
Dragojević, A, et al., No compromises: distributed transactions with consistency, availability, and performance. SOSP'15

Use Case 8: FT Broadcast



Bosilca, G., et al., Failure Detection and Propagation in HPC systems. SC'16

Use Case 9: Distributed Consensus



István, Z., et al., Consensus in a Box: Inexpensive Coordination in Hardware. NSDI'16

Further results and use-cases

Use Case 4: MPI Rendezvous Protocol

program	p	msgs	ovhd	ovhd	red
MILC	64	5.7M	5.5%	1.9%	65%
POP	64	772M	3.1%	2.4%	22%
coMD	72	5.3M	6.1%	2.4%	60%
coMD	360	28.1M	6.5%	2.8%	58%
Cloverleaf	72	2.7M	5.2%	2.4%	53%
Cloverleaf	360	15.3M	5.6%	3.2%	42%

Use Case 5: Distributed KV Store

The Next 700 sPIN use-cases

41% lower latency

Kalia, A., et al., Using RDMA efficiently for key-value services. In *ACM SIGCOMM Computer Communication Review, 2014*

Use Case 6: Conditional Read

Barthels, C., et al., Designing Databases for Future High-Performance Networks. *IEEE Data Eng. Bulletin, 2017*

Use Case 7: Distributed Transactions

Dragojević, A, et al., No compromises: distributed transactions with consistency, availability, and performance. SOSP'15

Use Case 8: FT Broadcast

Bosilca, G., et al., Failure Detection and Propagation in HPC systems. SC'16

Use Case 9: Distributed Consensus

István, Z., et al., Consensus in a Box: Inexpensive Coordination in Hardware. NSDI'16

Further results and use-cases

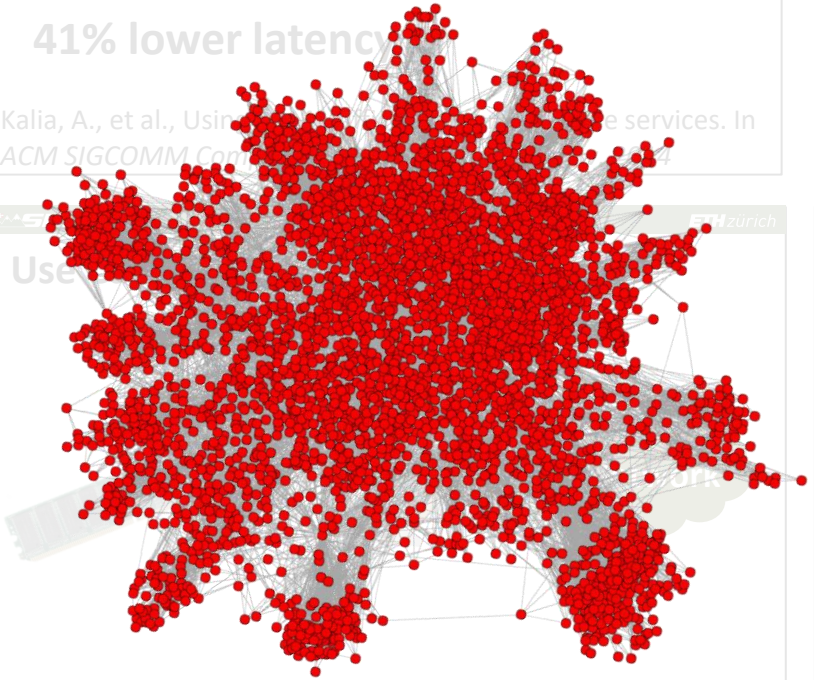
Use Case 4: MPI Rendezvous Protocol

program	p	msgs	ovhd	ovhd	red
MILC	64	5.7M	5.5%	1.9%	65%
POP	64	772M	3.1%	2.4%	22%
coMD	72	5.3M	6.1%	2.4%	60%
coMD	360	28.1M	6.5%	2.8%	58%
Cloverleaf	72	2.7M	5.2%	2.4%	53%
Cloverleaf	360	15.3M	5.6%	3.2%	42%

Use Case 5: Distributed KV Store

The Next 700 sPIN use-cases

… just think about sPIN graph kernels ….



41% lower latency

Kalia, A., et al., Using … services. In *ACM SIGCOMM Con…*

Use Case 6: Conditional Read

Barthels, C., et al., Designing Databases for Future High-Performance Networks. *IEEE Data Eng. Bulletin, 2017*

Use Case 7: Distributed Transactions

Dragojević, A, et al., No compromises: distributed transactions with consistency, availability, and performance. SOSP'15

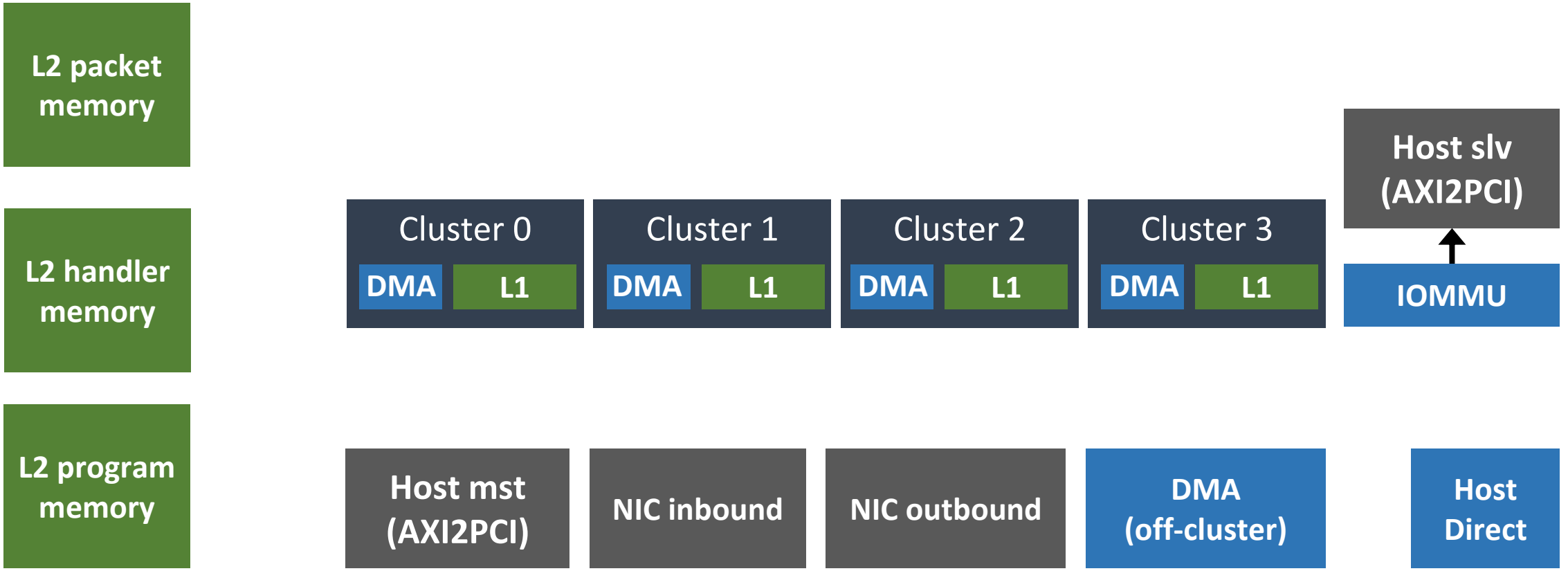
Use Case 8: …

Bosilca, G., et al., Failure Detection and Propagation in HPC systems. SC'16

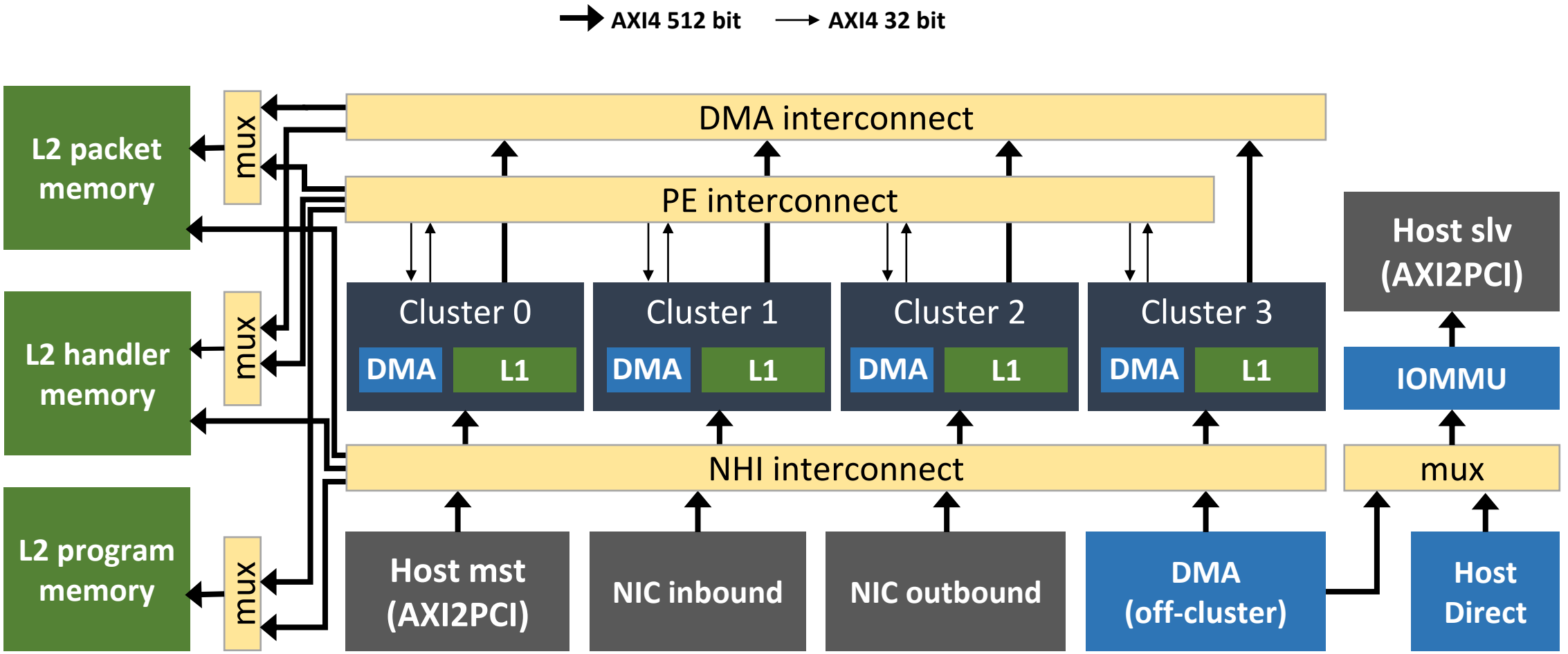
Use Case 9: Distributed Consensus

István, Z., et al., Consensus in a Box: Inexpensive Coordination in Hardware. NSDI'16

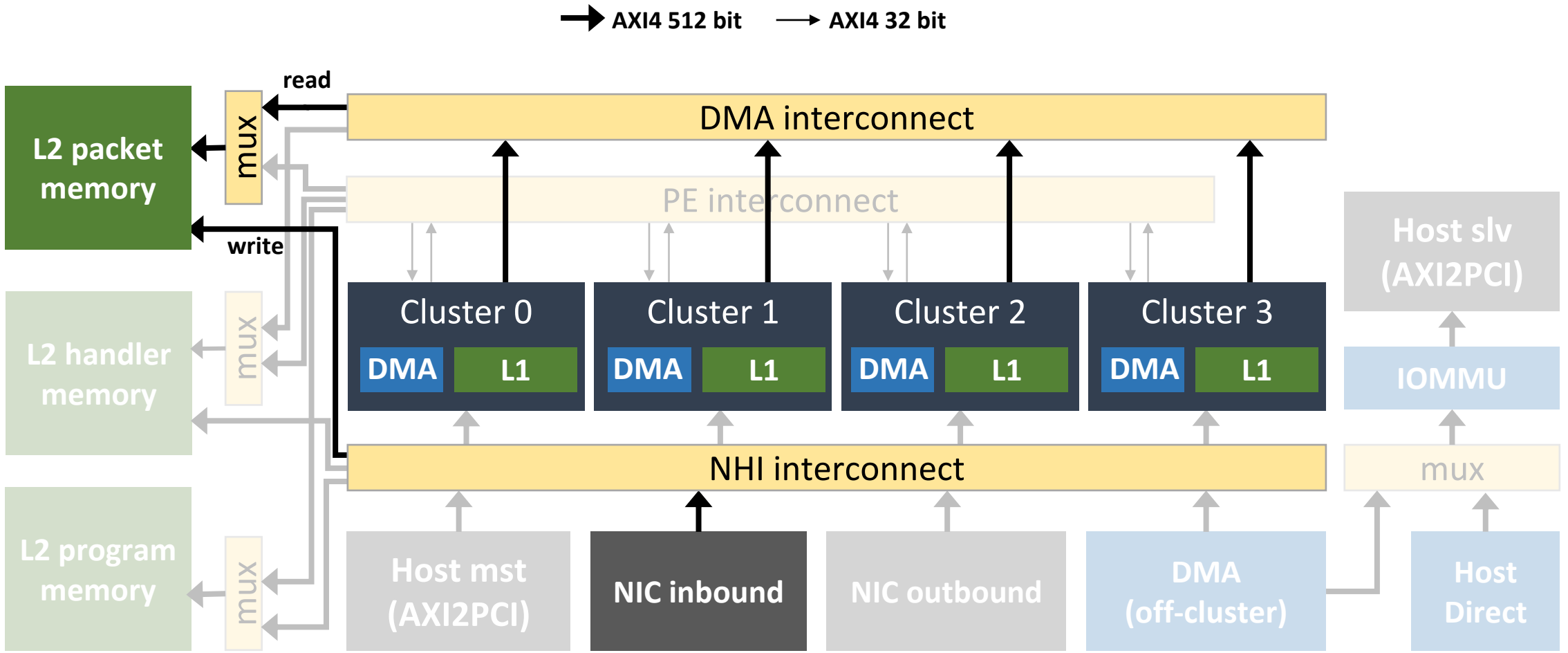
400G Data Path



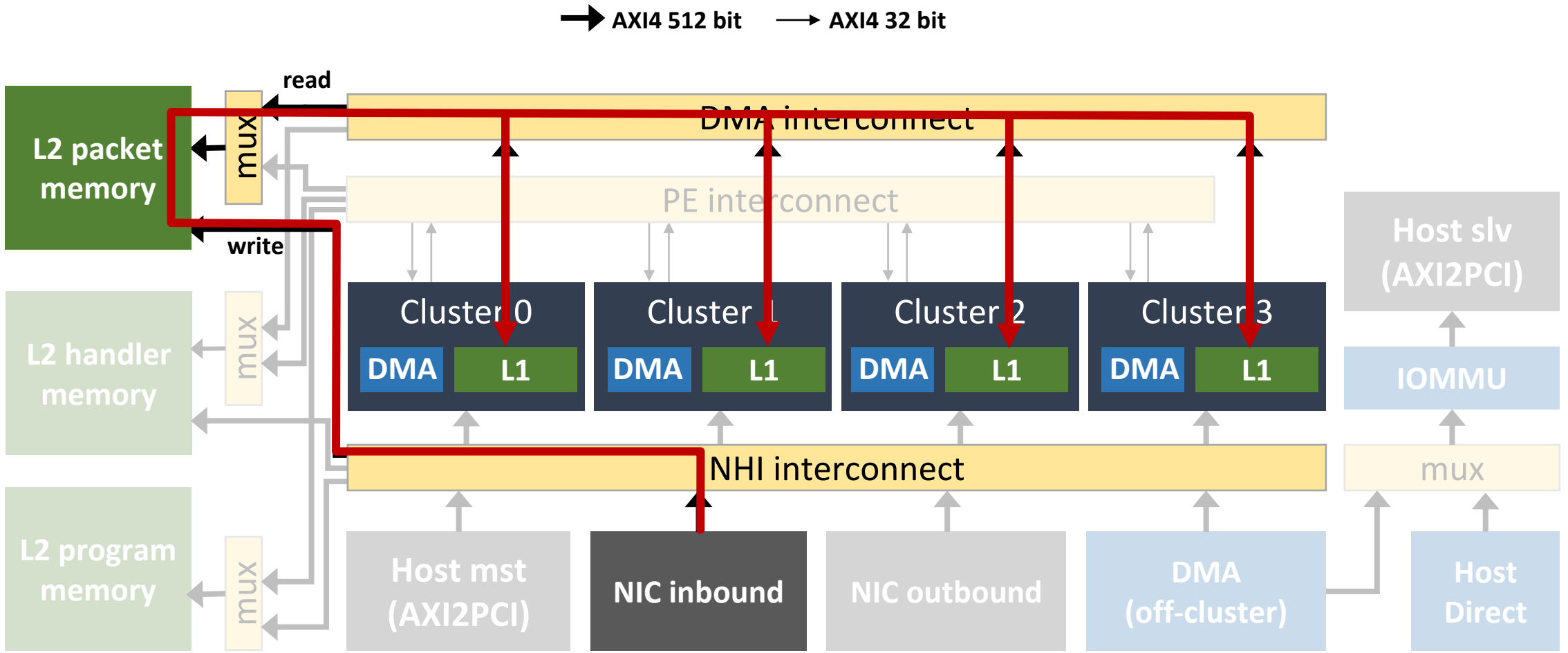
400G Data Path



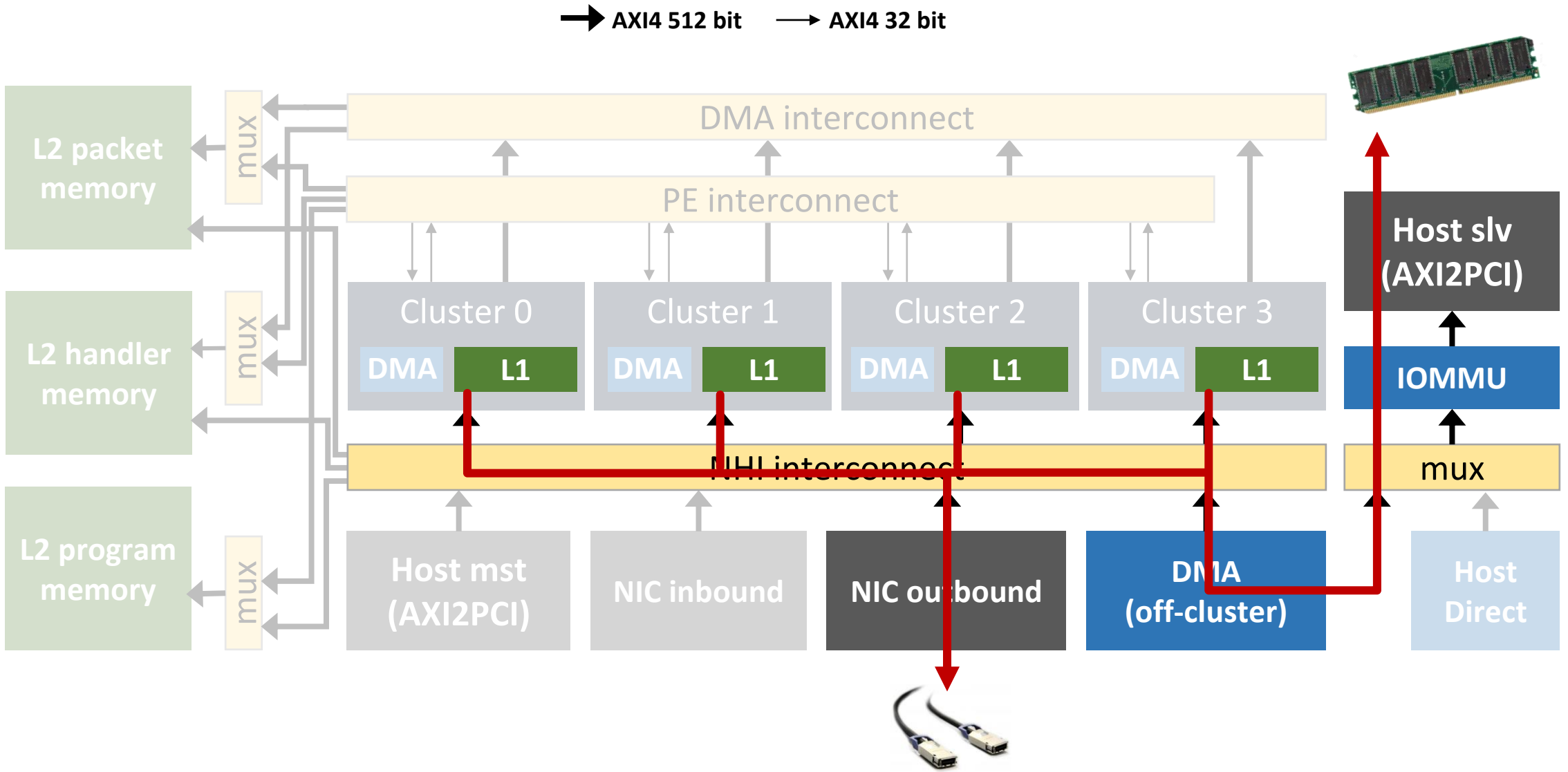
400G Data Path



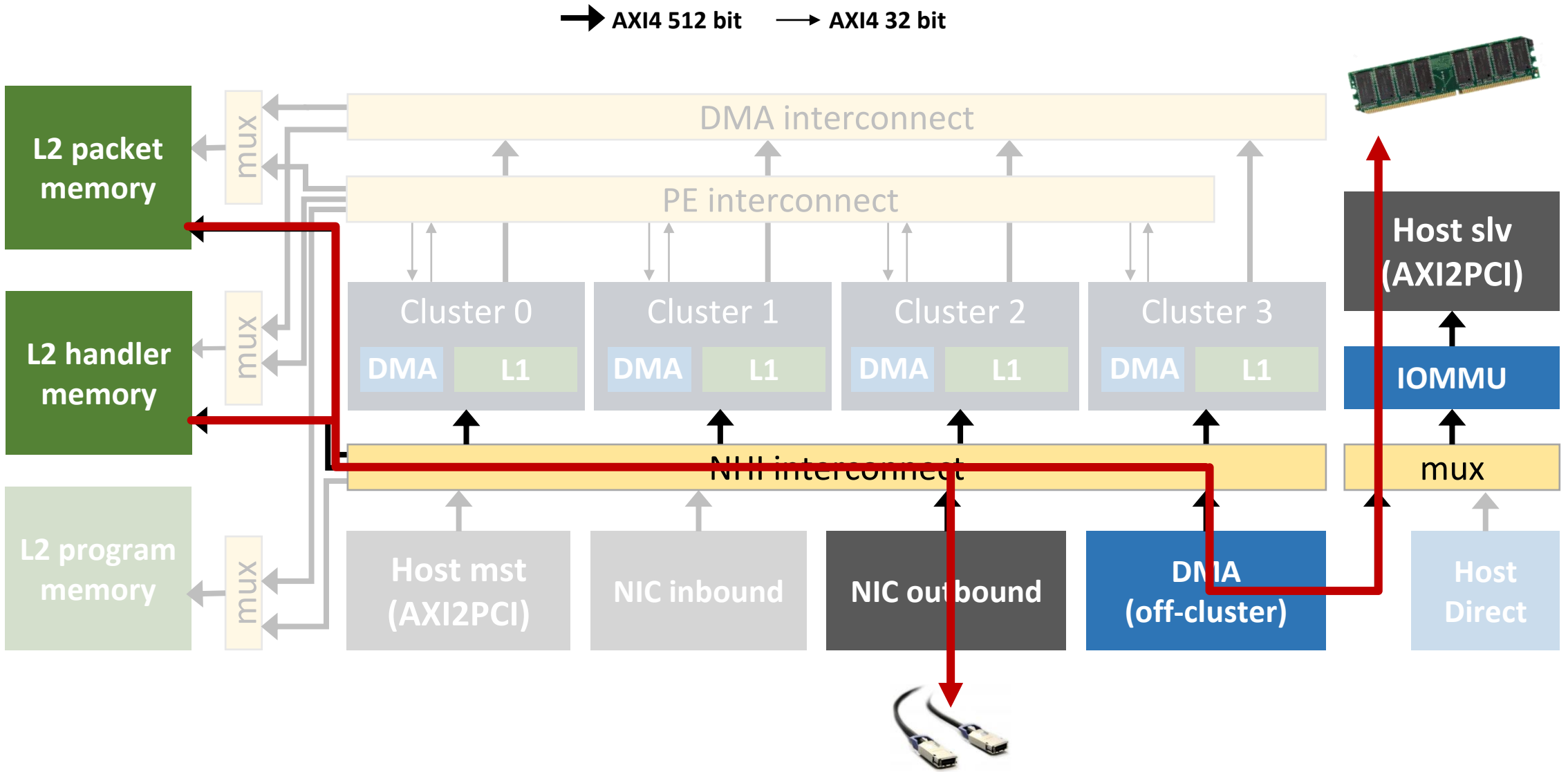
400G Data Path



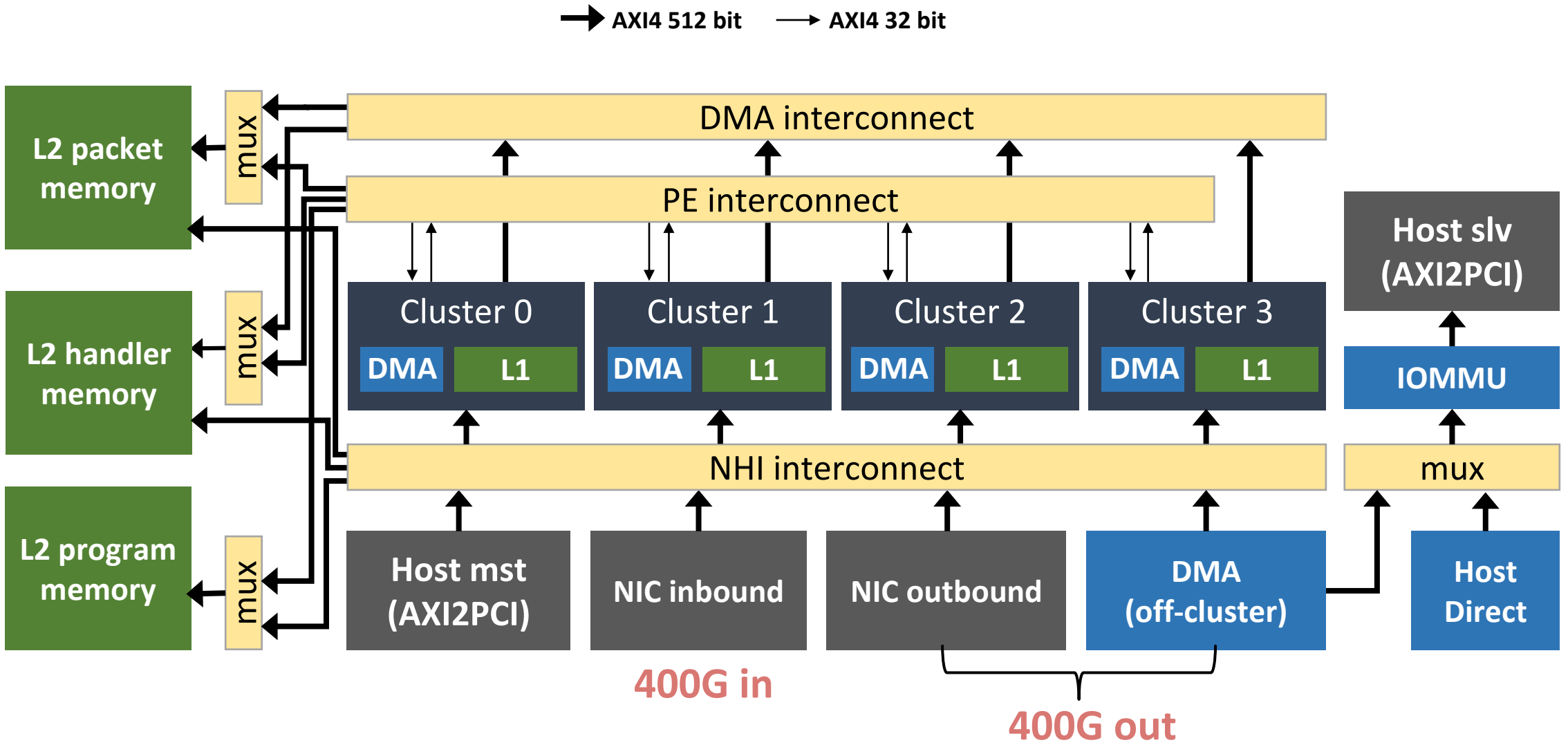
400G Data Path



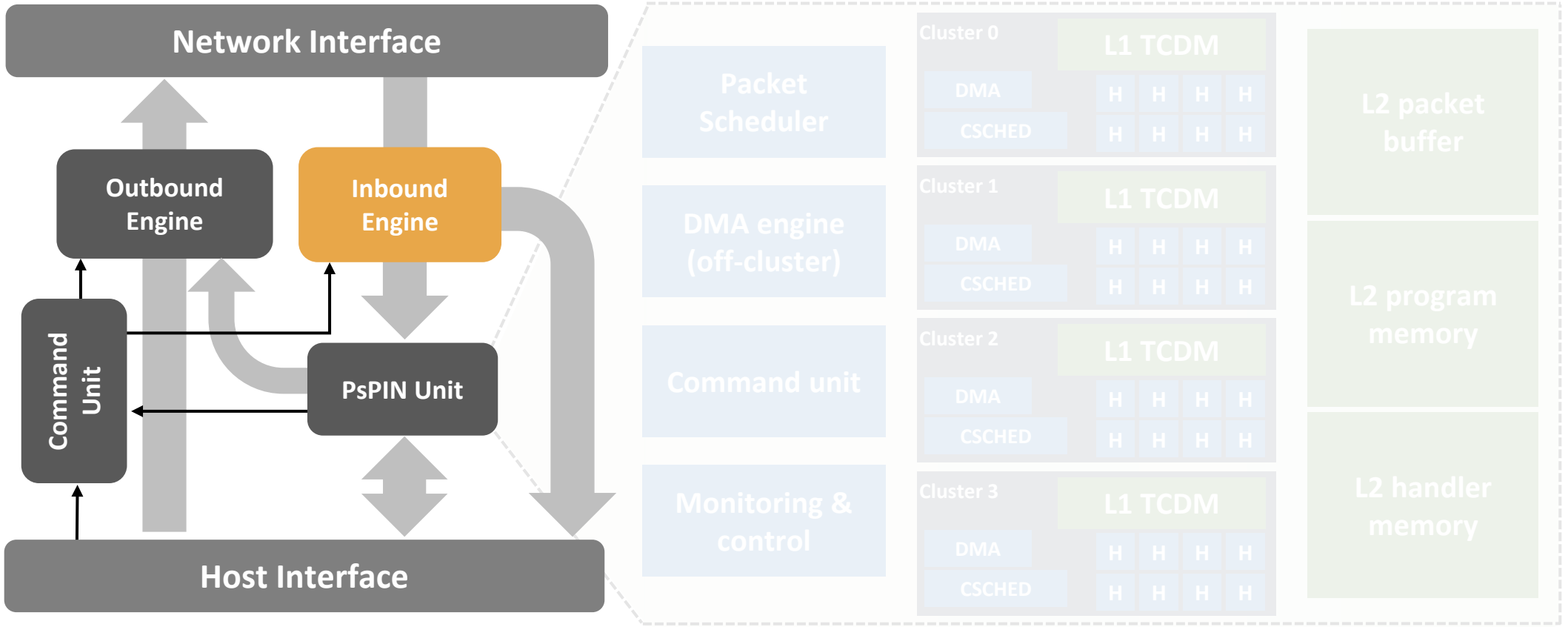
400G Data Path



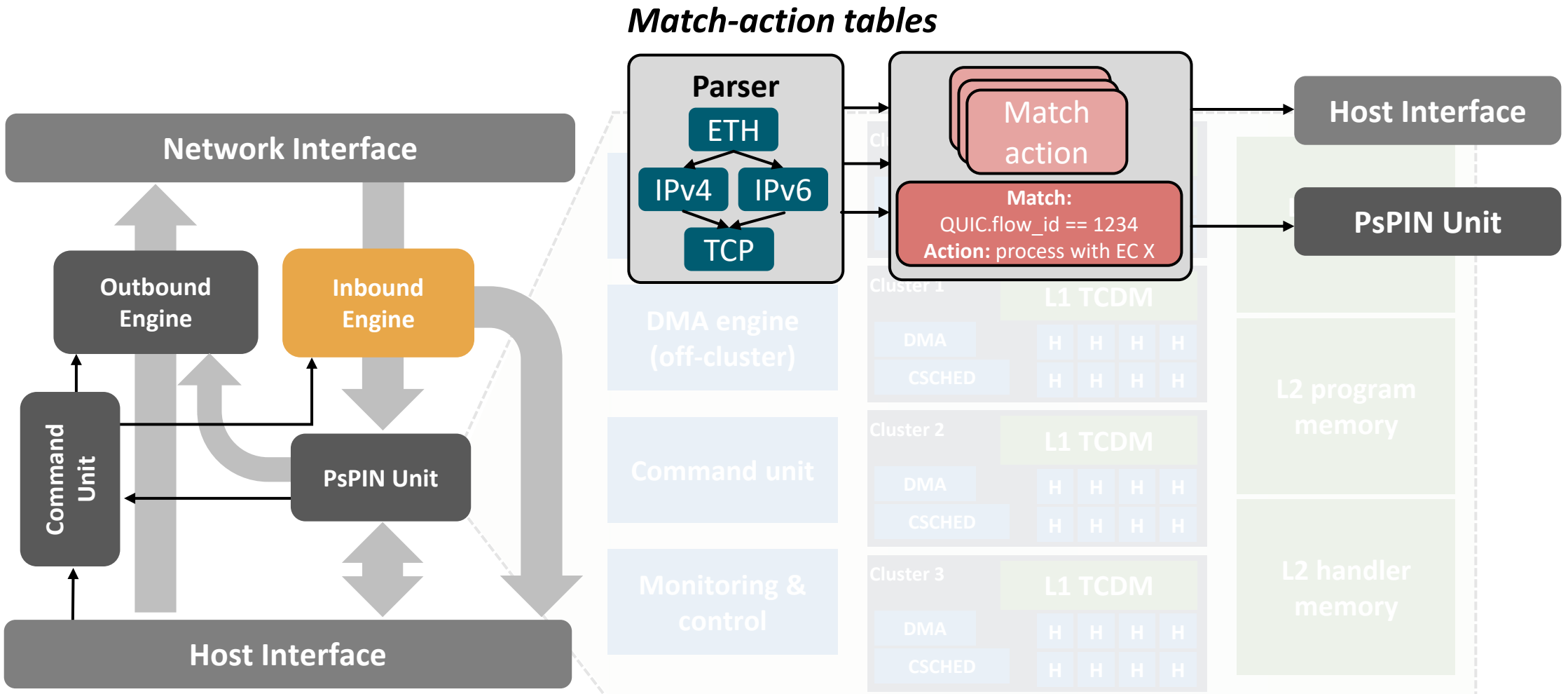
400G Data Path



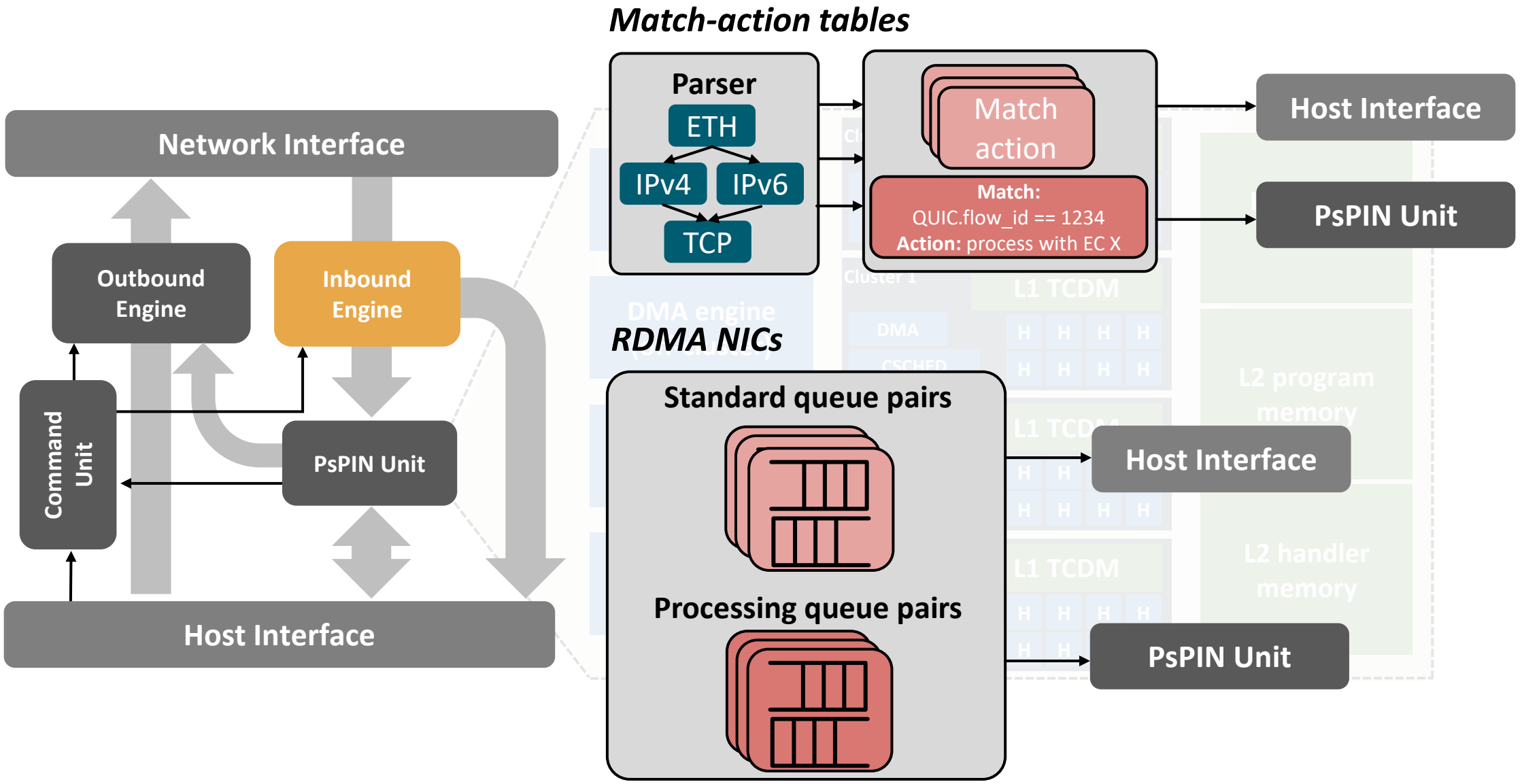
NIC integration



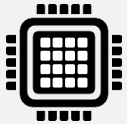
NIC integration



NIC integration



Low latency,
full throughput



Highly parallel

Fast scheduling

Fast explicit
memory access

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

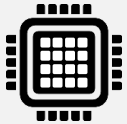
Low latency, full throughput

Highly parallel	✓	**32 cores, higher core-count configurations are possible with more clusters**
Fast scheduling	✓	**Tens of nanoseconds to get handlers started**
Fast explicit memory access	✓	**Single-cycle L1 memory**

Support for wide range of use cases

- Stateful computation support
- Handlers isolation

Low latency, full throughput



Highly parallel

Fast scheduling

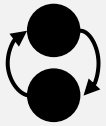
Fast explicit memory access

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

Low latency, full throughput

- Highly parallel ✓ — **32 cores, higher core-count configurations are possible with more clusters**
- Fast scheduling ✓ — **Tens of nanoseconds to get handlers started**
- Fast explicit memory access ✓ — **Single-cycle L1 memory**

Support for wide range of use cases

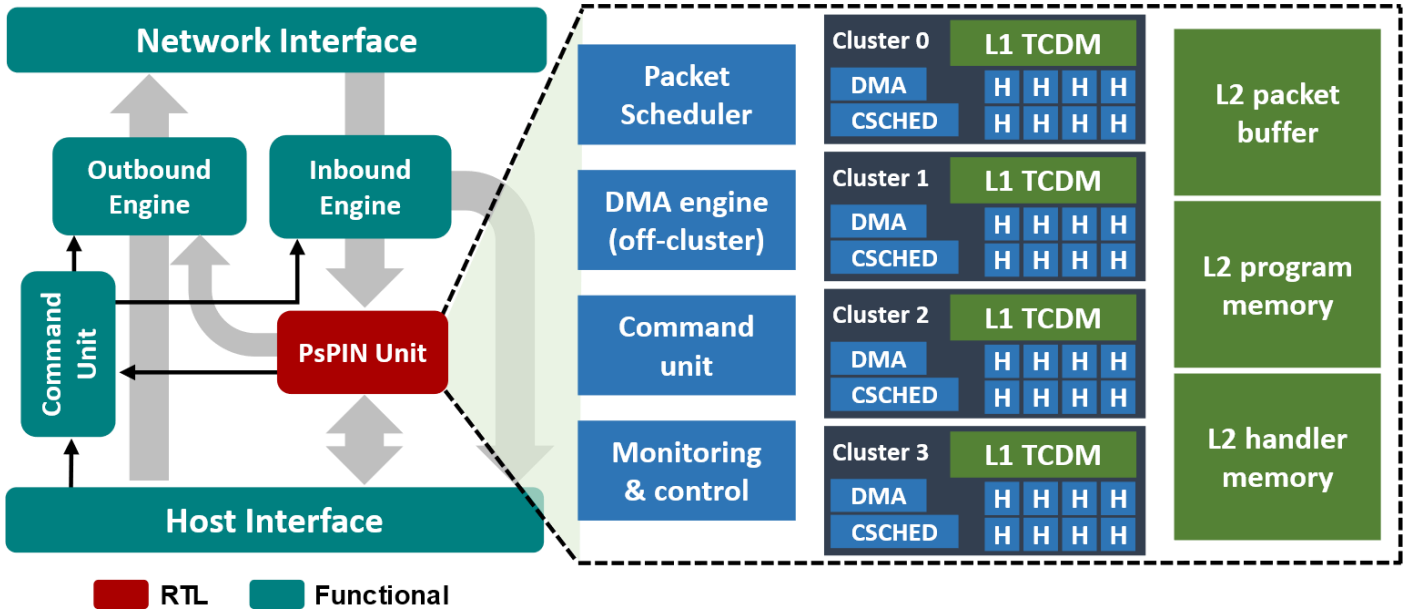
- Stateful computation support ✓ — **Implicit in the sPIN programming model**
- Handlers isolation ✓ — **HW-configured (1 cycle) RISC-V PMP**

Easy to integrate

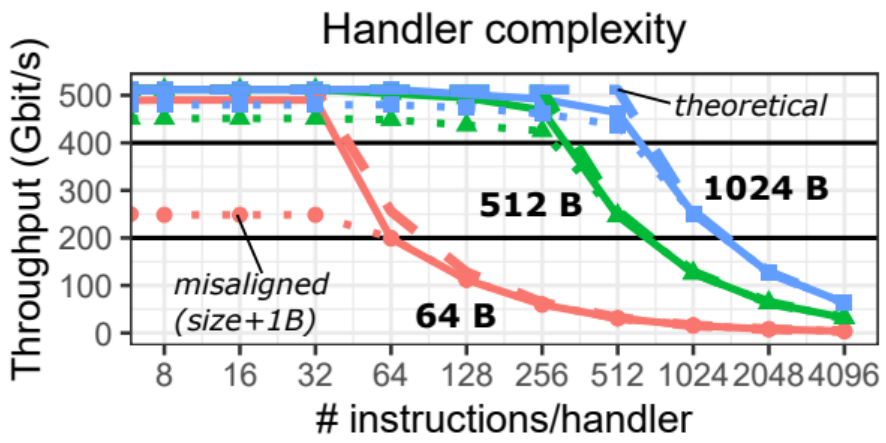
- Area and power efficiency
- Configurability

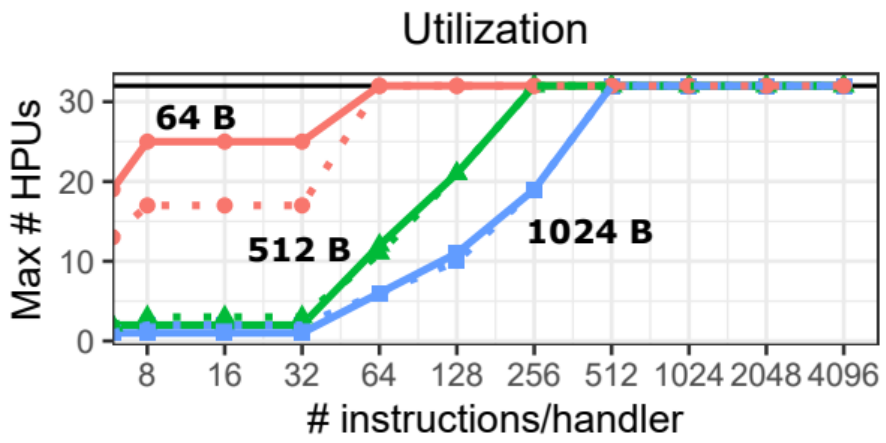
Low latency, full throughput	Highly parallel	✓	**32 cores, higher core-count configurations are possible with more clusters**
	Fast scheduling	✓	**Tens of nanoseconds to get handlers started**
	Fast explicit memory access	✓	**Single-cycle L1 memory**
Support for wide range of use cases	Stateful computation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

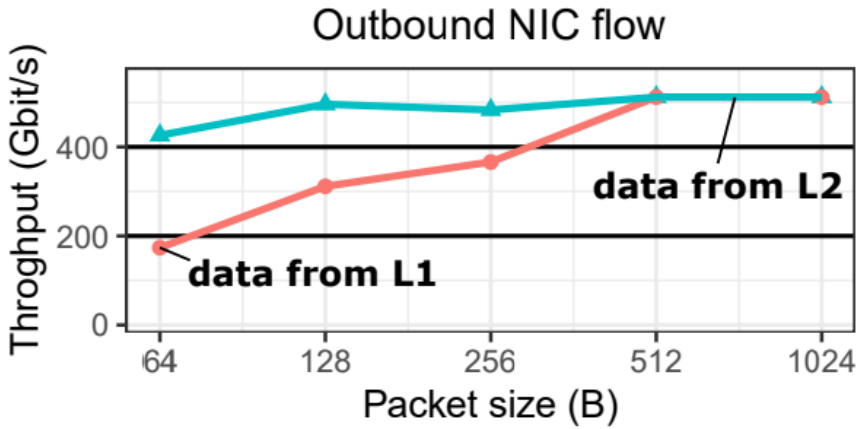
Experimental results

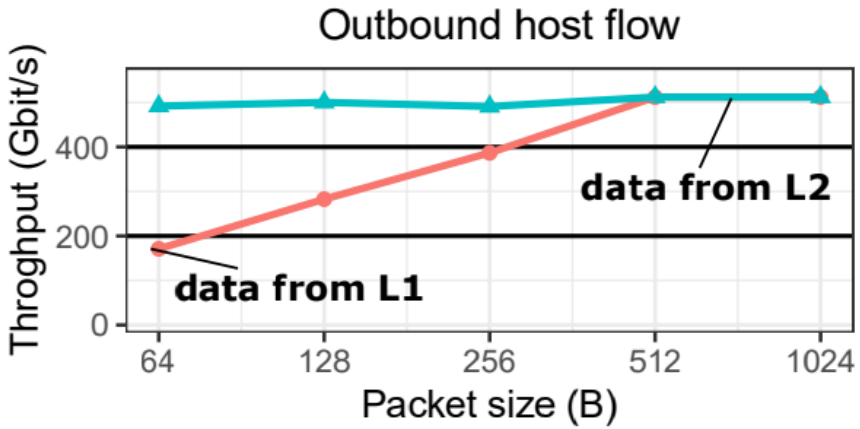


PsPIN Throughput and utilization









Handlers Characterization

Handlers Characterization

Packet steering
filtering, strided datatypes

Data movement
key-value store

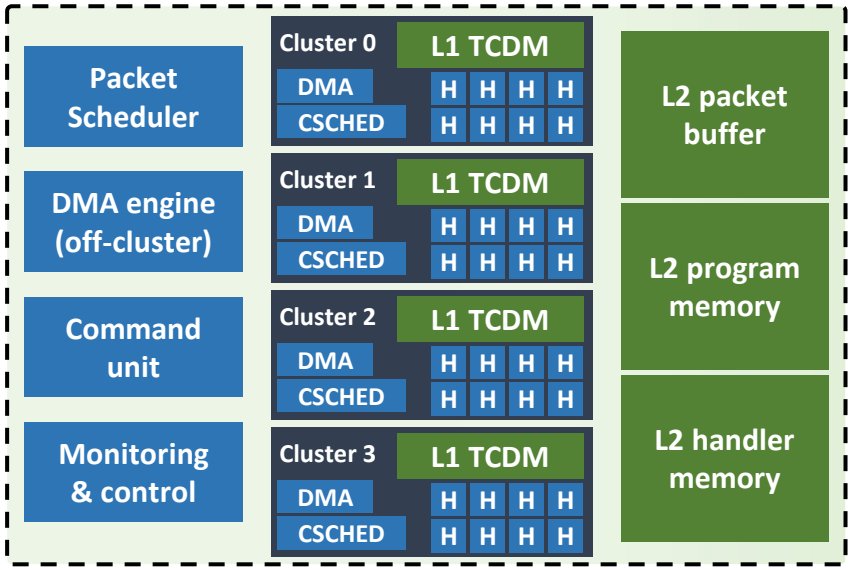
Full packet processing
aggregate, histogram, reduce

Handlers Characterization

Packet steering
filtering, strided datatypes

Data movement
key-value store

Full packet processing
aggregate, histogram, reduce

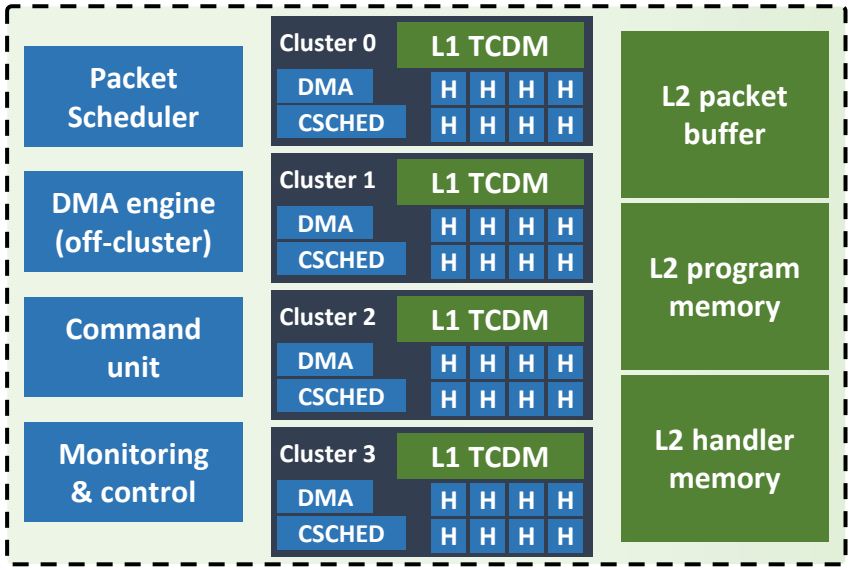


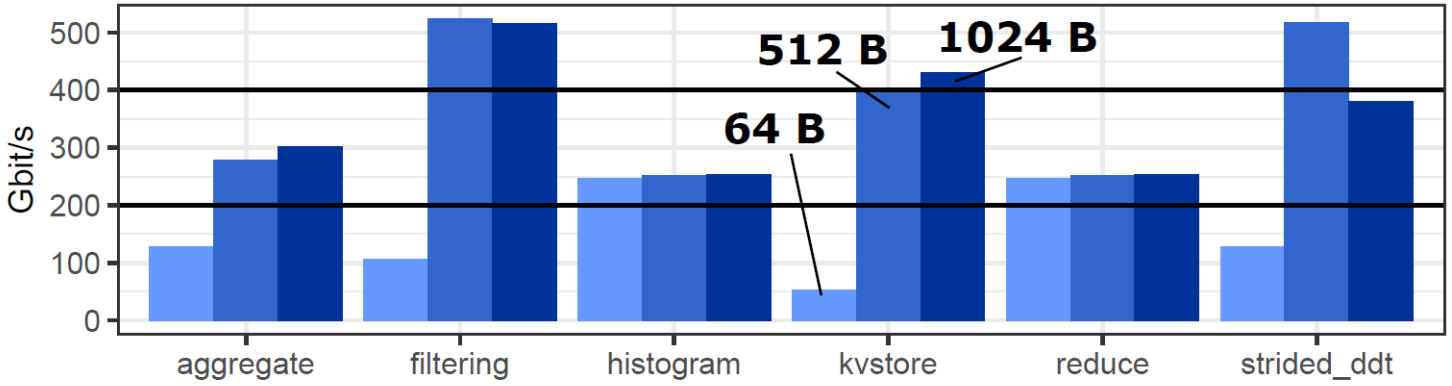
Handlers Characterization

Packet steering
filtering, strided datatypes

Data movement
key-value store

Full packet processing
aggregate, histogram, reduce



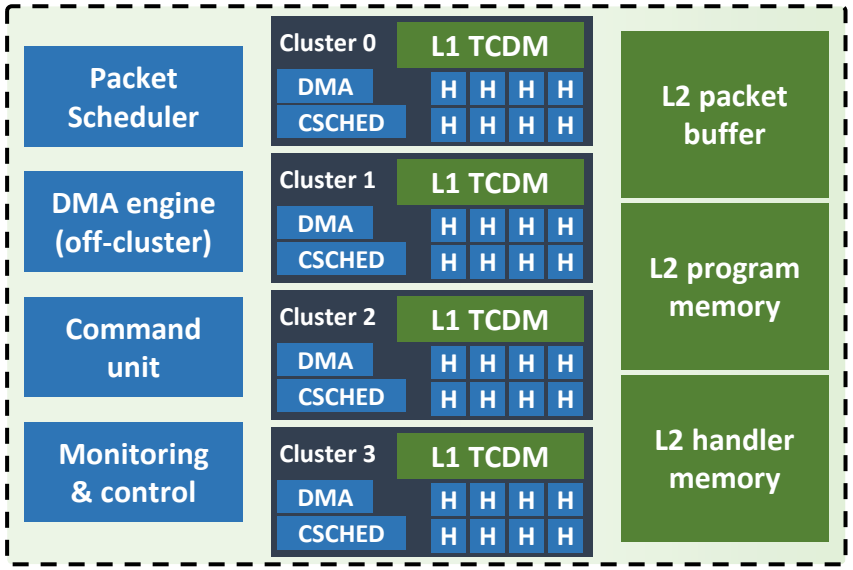


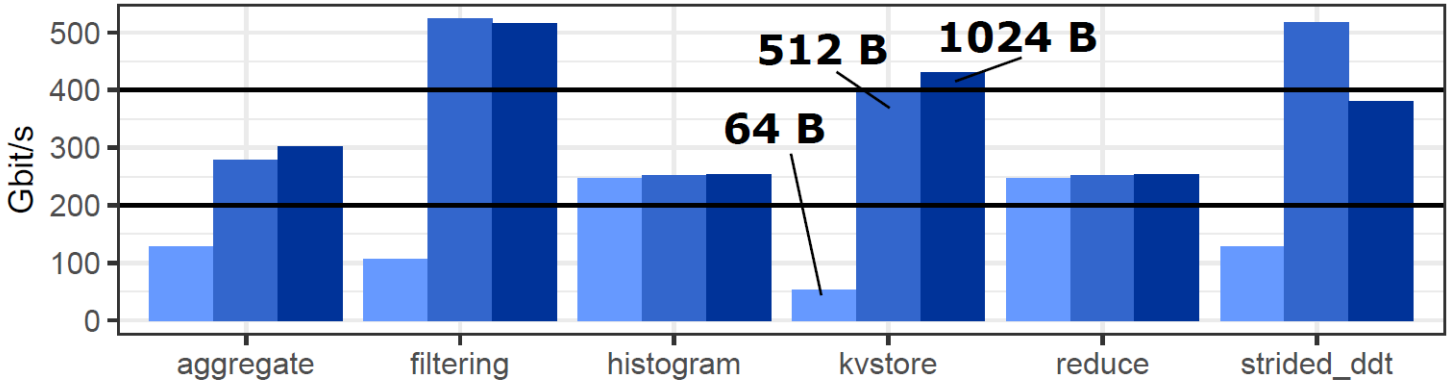
Handlers Characterization

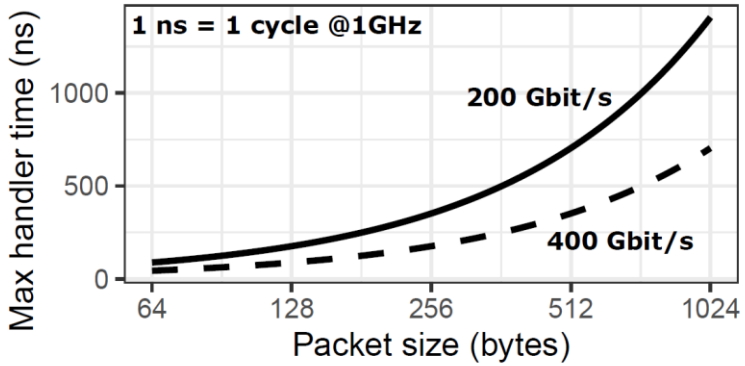
Packet steering
filtering, strided datatypes

Data movement
key-value store

Full packet processing
aggregate, histogram, reduce







How about other architectures?

How about other architectures?

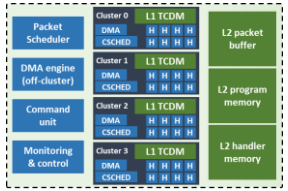
ault @ CSCS

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



RI5CY (RISC-V) @ 1 GHz
(32 cores, single-issue, in-order, 32-bit)

How about other architectures?

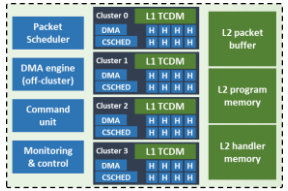
ault @ CSCS

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

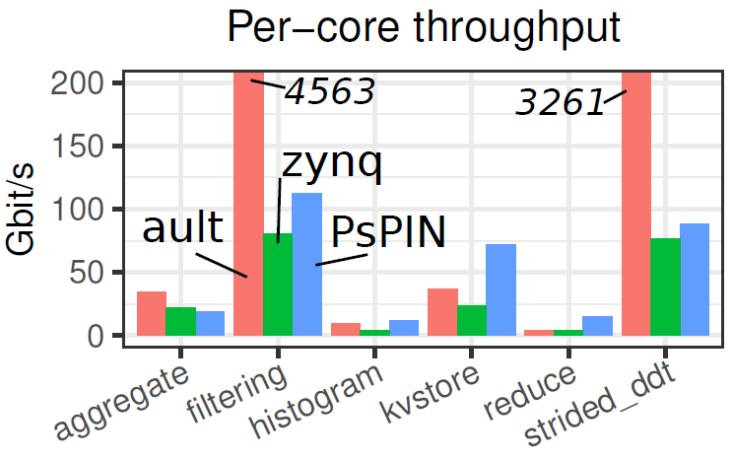
zynq

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

PsPIN



RI5CY (RISC-V) @ 1 GHz
(32 cores, single-issue, in-order, 32-bit)



How about other architectures?

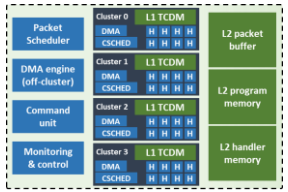
ault @ CSCS

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

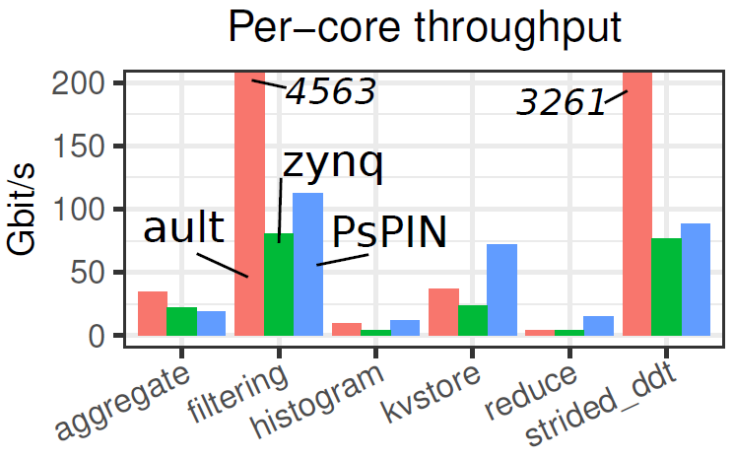
zynq

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

PsPIN



RI5CY (RISC-V) @ 1 GHz
(32 cores, single-issue, in-order, 32-bit)



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?

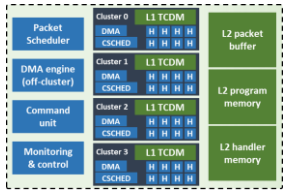
ault @ CSCS

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

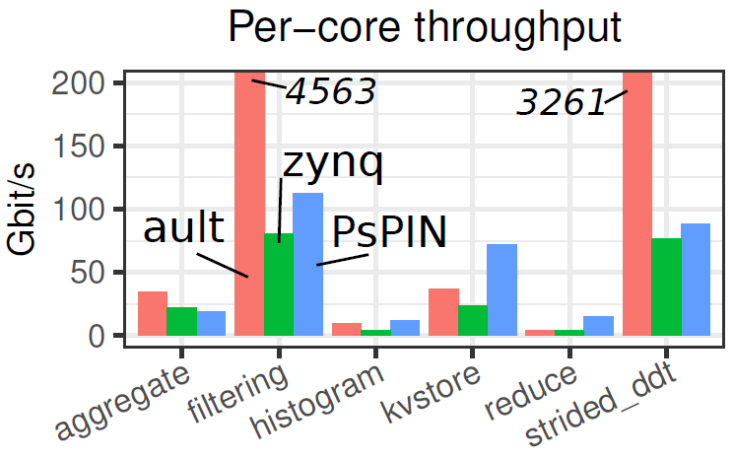
zynq

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

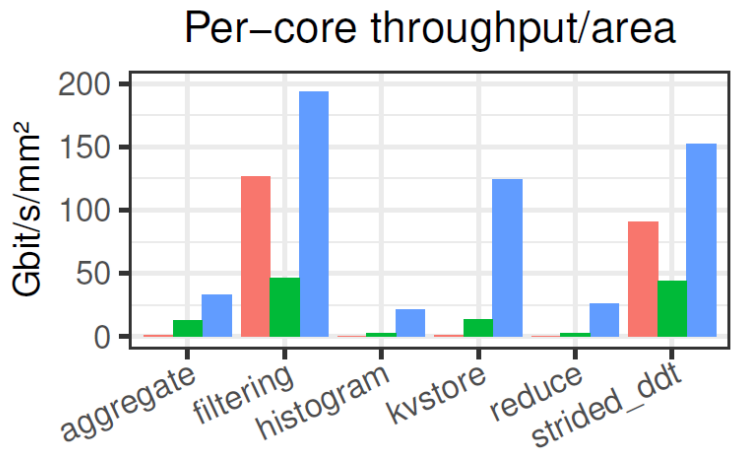
PsPIN



RI5CY (RISC-V) @ 1 GHz
(32 cores, single-issue, in-order, 32-bit)



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



Comparison vs other architectures

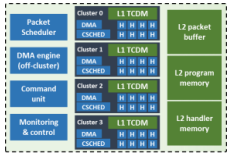
ault @ CSCS

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

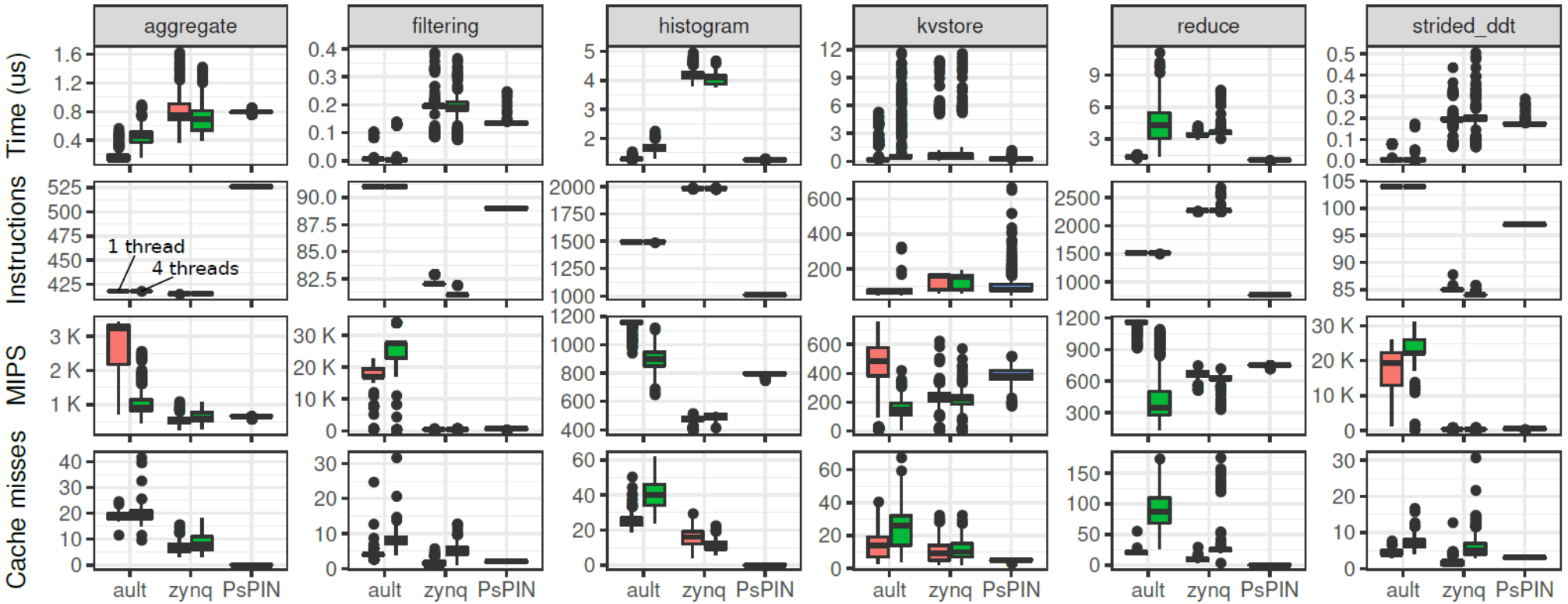
zynq

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

PsPIN



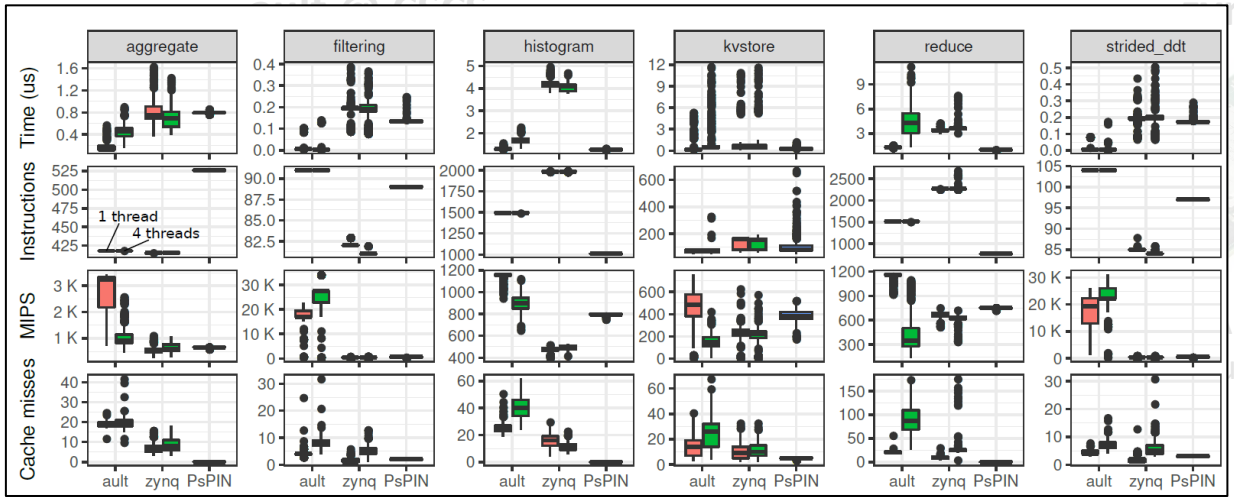
RI5CY (RISC-V) @ 1 GHz
(32 cores, single-issue, in-order, 32-bit)

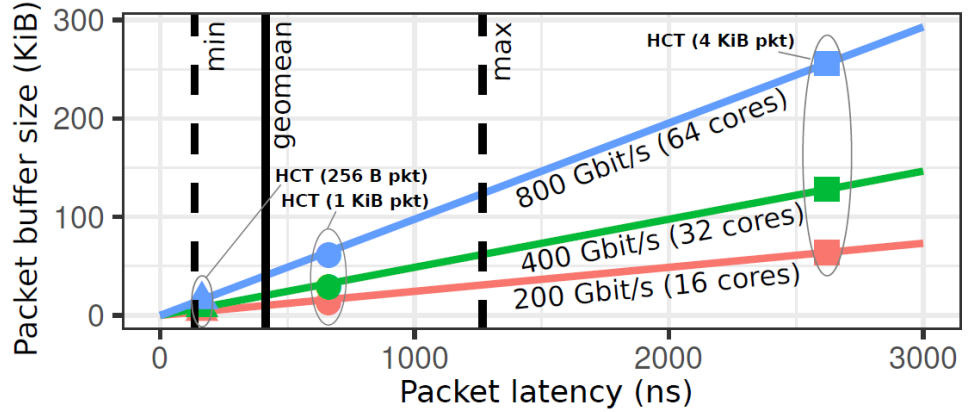


Other SmartNICs

- **Netronome/P4-based NICs**
  - Different philosophy
  - Offloaded computation is not per-application (vs sPIN per-application packet handlers)
    No isolation (computation on the NIC sees all packets)
  - Introduce limitation on the offloaded computation
    e.g., XDP is not Turing complete
  - Not open source ☹
- **INCA/off-path SmartNICs**
  - Complementary to sPIN/PsPIN

How about other architectures?





PsPIN

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*, Jakub 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 mm^2 (22 nm FDSOI).**

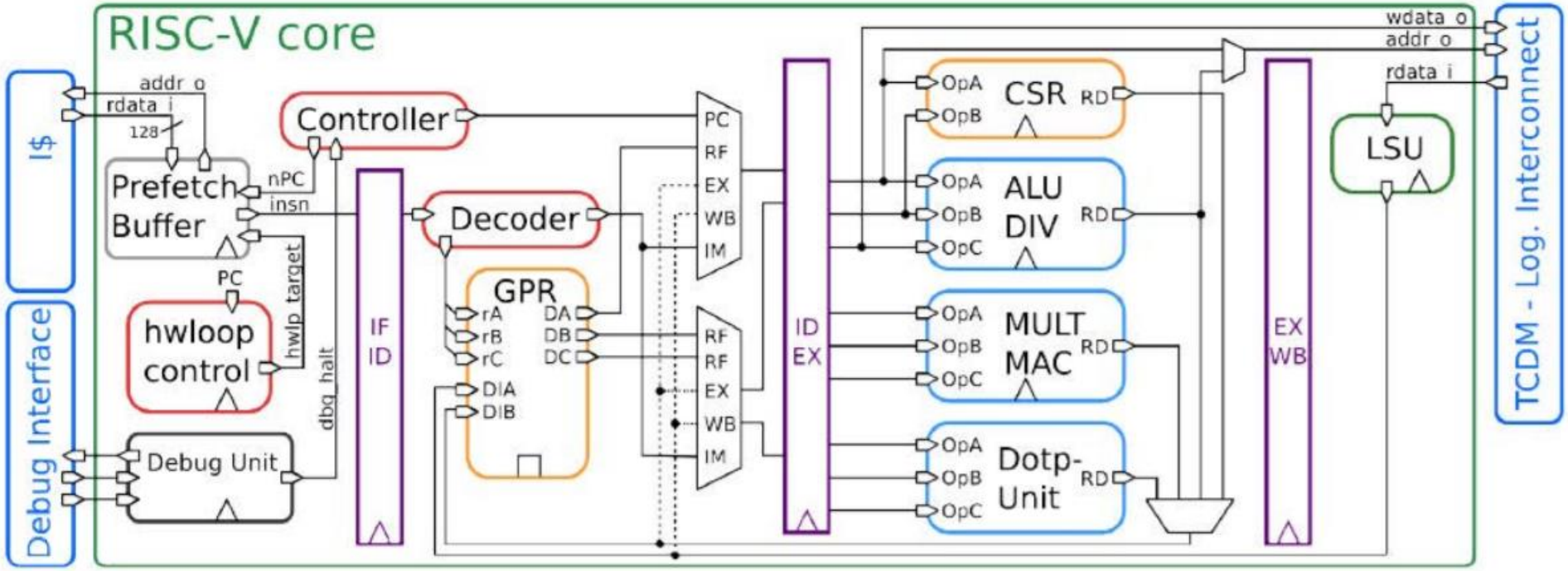
***Index Terms*—in-network compute, packet processing, special-**

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

RI5CY Core



- **4-stage pipeline, in-order, optimized for energy efficiency**
- **Area: 40 kGE**
- **Critical path: 30 logic levels of critical path**
- **CoreMark/MHZ 3.19**
- **Includes various extensions (X) to RISC-V (SIMD, Fixed point, bit manipulation, hw loops)**

Options:
- FPU: IEEE 754 single precision (+ 40-70 kGE)
- U/M privileges

https://pulp-platform.org/docs/hipeac/pulp_intro_kgf.pdf

But why PULP/RISC-V?

But why PULP/RISC-V?

- RISC-V is an open source ISA
  - Allows and supports extensions

    Doing this in ARM may be complex and expensive

But why PULP/RISC-V?

- RISC-V is an open source ISA
  - Allows and supports extensions

    Doing this in ARM may be complex and expensive

- PULP

But why PULP/RISC-V?

- RISC-V is an open source ISA
  - Allows and supports extensions

    Doing this in ARM may be complex and expensive

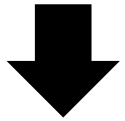
- PULP
  - Open source!
  - Energy efficient
  - Provides tight control over compute and data movement schedule
  - Fits well the sPIN abstract machine model (e.g., removing cache coherency on ARM could be painful)
  - Actively researched

Custom ISA Extensions

```
for (i = 0; i < 100; i++)
   d[i] = a[i] + b[i];
```

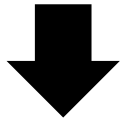
Custom ISA Extensions

```
for (i = 0; i < 100; i++)
   d[i] = a[i] + b[i];
```



Custom ISA Extensions

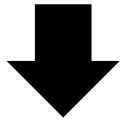
```
for (i = 0; i < 100; i++)
   d[i] = a[i] + b[i];
```



```
mv x5, 0
mv x4, 100
Lstart:
   lb x2, 0(x10)
   lb x3, 0(x11)
   addi x10, x10, 1
   addi x11, x11, 1
   add x2, x3, x2
   sb x2, 0(x12)
   addi x4, x4, -1
   addi x12, x12, 1
bne x4, x5, Lstart
```

Custom ISA Extensions

```
for (i = 0; i < 100; i++)
   d[i] = a[i] + b[i];
```

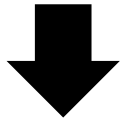


```
mv x5, 0
mv x4, 100
Lstart:
   lb x2, 0(x10)
   lb x3, 0(x11)
   addi x10, x10, 1
   addi x11, x11, 1
   add x2, x3, x2
   sb x2, 0(x12)
   addi x4, x4, -1
   addi x12, x12, 1
bne x4, x5, Lstart
```

11 cycles/output

Custom ISA Extensions

```
for (i = 0; i < 100; i++)
   d[i] = a[i] + b[i];
```

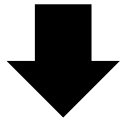


```
mv x5, 0
mv x4, 100
Lstart:
   lb x2, 0(x10)
   lb x3, 0(x11)
   addi x10, x10, 1
   addi x11, x11, 1
   add x2, x3, x2
   sb x2, 0(x12)
   addi x4, x4, -1
   addi x12, x12, 1
bne x4, x5, Lstart
```

11 cycles/output

Custom ISA Extensions

```
for (i = 0; i < 100; i++)
   d[i] = a[i] + b[i];
```



```
mv x5, 0
mv x4, 100
Lstart:
   lb x2, 0(x10)
   lb x3, 0(x11)
   addi x10, x10, 1
   addi x11, x11, 1
   add x2, x3, x2
   sb x2, 0(x12)
   addi x4, x4, -1
   addi x12, x12, 1
bne x4, x5, Lstart
```

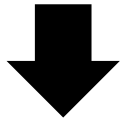
```
mv x5, 0
mv x4, 100
Lstart:
   lb x2, 0(x10!)
   lb x3, 0(x11!)
   addi x4, x4, -1
   add x2, x3, x2
   sb x2, 0(x12!)
bne x4, x5, Lstart
```

Auto-incr load/store

11 cycles/output

Custom ISA Extensions

```
for (i = 0; i < 100; i++)
   d[i] = a[i] + b[i];
```



```
mv x5, 0
mv x4, 100
Lstart:
   lb x2, 0(x10)
   lb x3, 0(x11)
   addi x10, x10, 1
   addi x11, x11, 1
   add x2, x3, x2
   sb x2, 0(x12)
   addi x4, x4, -1
   addi x12, x12, 1
bne x4, x5, Lstart
```

11 cycles/output

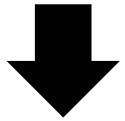
```
mv x5, 0
mv x4, 100
Lstart:
   lb x2, 0(x10!)
   lb x3, 0(x11!)
   addi x4, x4, -1
   add x2, x3, x2
   sb x2, 0(x12!)
bne x4, x5, Lstart
```

Auto-incr load/store

8 cycles/output

Custom ISA Extensions

```
for (i = 0; i < 100; i++)
   d[i] = a[i] + b[i];
```



```
mv x5, 0
mv x4, 100
Lstart:
   lb x2, 0(x10)
   lb x3, 0(x11)
   addi x10, x10, 1
   addi x11, x11, 1
   add x2, x3, x2
   sb x2, 0(x12)
   addi x4, x4, -1
   addi x12, x12, 1
bne x4, x5, Lstart
```

```
mv x5, 0
mv x4, 100
Lstart:
   lb x2, 0(x10!)
   lb x3, 0(x11!)
   addi x4, x4, -1
   add x2, x3, x2
   sb x2, 0(x12!)
bne x4, x5, Lstart
```

Auto-incr load/store

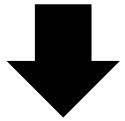
```
lp.setupi 100, Lend
   lb x2, 0(x10!)
   lb x3, 0(x11!)
   add x2, x3, x2
Lend: sb x2, 0(x12!)
```

HW loop

11 cycles/output **8 cycles/output**

Custom ISA Extensions

```
for (i = 0; i < 100; i++)
   d[i] = a[i] + b[i];
```



```
mv x5, 0
mv x4, 100
Lstart:
   lb x2, 0(x10)
   lb x3, 0(x11)
   addi x10, x10, 1
   addi x11, x11, 1
   add x2, x3, x2
   sb x2, 0(x12)
   addi x4, x4, -1
   addi x12, x12, 1
bne x4, x5, Lstart
```

11 cycles/output

```
mv x5, 0
mv x4, 100
Lstart:
   lb x2, 0(x10!)
   lb x3, 0(x11!)
   addi x4, x4, -1
   add x2, x3, x2
   sb x2, 0(x12!)
bne x4, x5, Lstart
```

Auto-incr load/store

8 cycles/output

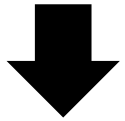
```
lp.setupi 100, Lend
   lb x2, 0(x10!)
   lb x3, 0(x11!)
   add x2, x3, x2
Lend: sb x2, 0(x12!)
```

HW loop

5 cycles/output

Custom ISA Extensions

```
for (i = 0; i < 100; i++)
   d[i] = a[i] + b[i];
```



```
mv x5, 0
mv x4, 100
Lstart:
   lb x2, 0(x10)
   lb x3, 0(x11)
   addi x10, x10, 1
   addi x11, x11, 1
   add x2, x3, x2
   sb x2, 0(x12)
   addi x4, x4, -1
   addi x12, x12, 1
bne x4, x5, Lstart
```

11 cycles/output

```
mv x5, 0
mv x4, 100
Lstart:
   lb x2, 0(x10!)
   lb x3, 0(x11!)
   addi x4, x4, -1
   add x2, x3, x2
   sb x2, 0(x12!)
bne x4, x5, Lstart
```

Auto-incr load/store

8 cycles/output

```
lp.setupi 100, Lend
   lb x2, 0(x10!)
   lb x3, 0(x11!)
   add x2, x3, x2
Lend: sb x2, 0(x12!)
```

HW loop

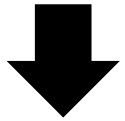
5 cycles/output

```
lp.setupi 25, Lend
   lw x2, 0(x10!)
   lw x3, 0(x11!)
   pv.add.b x2, x3, x2
Lend: sw x2, 0(x12!)
```

Packed SIMD

Custom ISA Extensions

```
for (i = 0; i < 100; i++)
   d[i] = a[i] + b[i];
```



```
mv x5, 0
mv x4, 100
Lstart:
   lb x2, 0(x10)
   lb x3, 0(x11)
   addi x10, x10, 1
   addi x11, x11, 1
   add x2, x3, x2
   sb x2, 0(x12)
   addi x4, x4, -1
   addi x12, x12, 1
bne x4, x5, Lstart
```

11 cycles/output

```
mv x5, 0
mv x4, 100
Lstart:
   lb x2, 0(x10!)
   lb x3, 0(x11!)
   addi x4, x4, -1
   add x2, x3, x2
   sb x2, 0(x12!)
bne x4, x5, Lstart
```

Auto-incr load/store

8 cycles/output

```
lp.setupi 100, Lend
   lb x2, 0(x10!)
   lb x3, 0(x11!)
   add x2, x3, x2
Lend: sb x2, 0(x12!)
```

HW loop

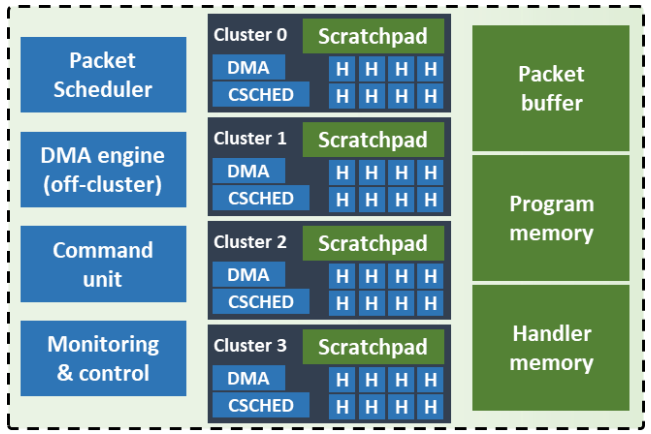
5 cycles/output

```
lp.setupi 25, Lend
   lw x2, 0(x10!)
   lw x3, 0(x11!)
   pv.add.b x2, x3, x2
Lend: sw x2, 0(x12!)
```

Packed SIMD

1.25 cycles/output

Scalability



Area:
95 MGE (18.5 mm2, 70% layout density)

Power:
6.1 W (98% dynamic power, worst case)

$$C = \frac{P \cdot N \cdot f}{B}$$

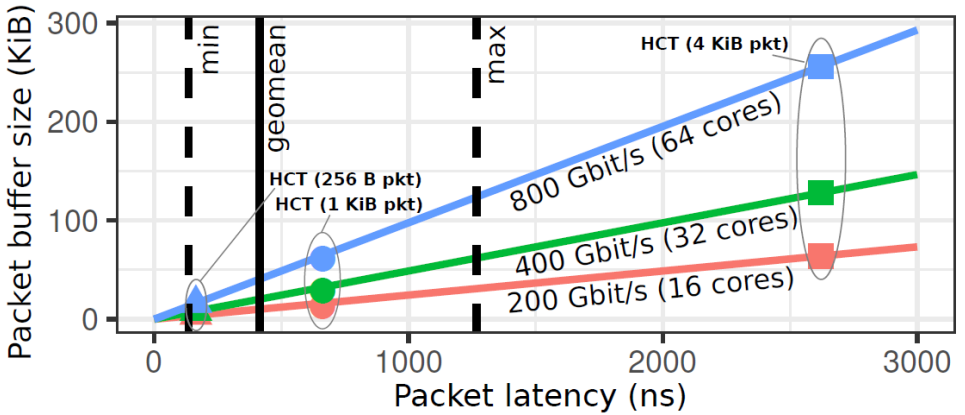
Heavier handlers

- Number of clusters
  - (number of cores/cluster is more challenging)
  - Simple cores (e.g., Snitch)
- Number of PsPIN units
- Frequency
- Accelerators (FPGAs?), specialized instructions

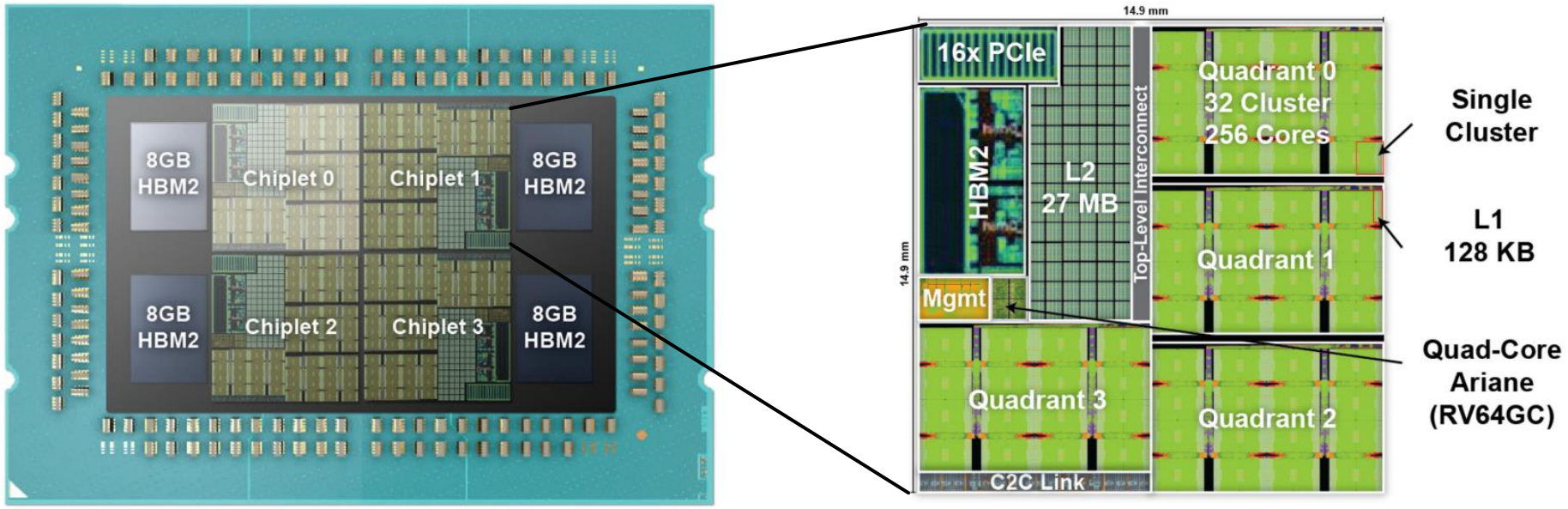
22nm -> 14 nm-> ~50% area reduction -> 20/30% frequency increase

Higher network bandwidth

- Need to rebalance
- Need to scale data path as well!
- Only packet buffer is affected!



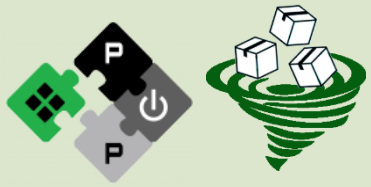
A 4096-Core RISC-V Architecture



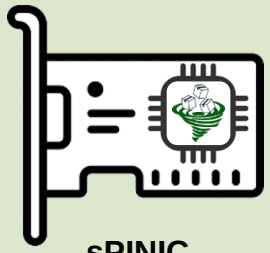
222 mm^2 @ 22nm

Zaruba, Florian, Fabian Schuiki, and Luca Benini. "Manticore: A 4096-Core RISC-V Chiplet Architecture for Ultraefficient Floating-Point Computing." *IEEE Micro* 41, no. 2 (2020): 36-42.

A sPINning ecosystem

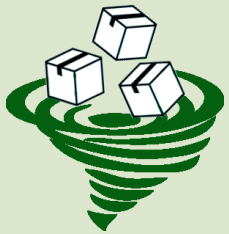


PsPIN (ISCA '21)
Power-efficient sPIN accelerator



sPINIC

Flare (SC '21)
Offloading all-reduce to sPIN switches



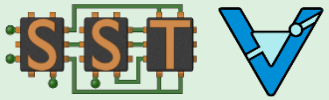
sPIN
(SC '17, best paper fin.)
programming model

architecture

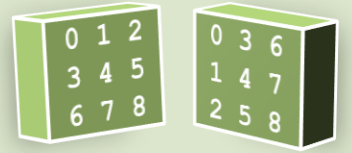
use cases

Simulations

Verilator support (Github)
Running PsPIN in an open-source cycle-accurate simulator



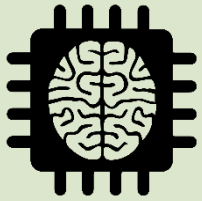
SST support (in progress)
Large scale network simulations mixed with cycle accurate ones



Network-accelerated datatypes (SC '19)

Zoo-sPINNER (MSc)
consensus on sPIN

sPIN-FS



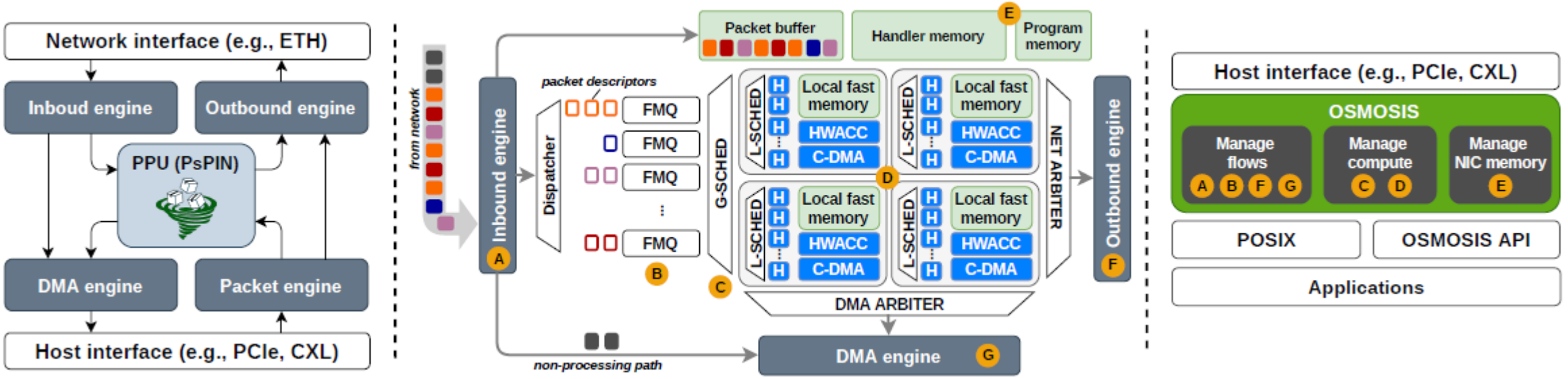
Quantization (MSc)
Allreduce and other collectives

Erasure coding (MSc)

Serverless sPIN (BSc)

Packet classification and pattern matching (BSc/MSc)

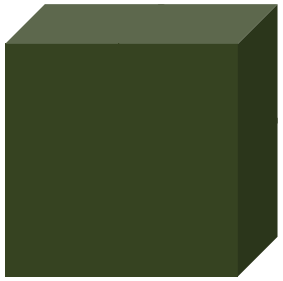
Multitenancy



Network-accelerated non-contiguous memory transfers

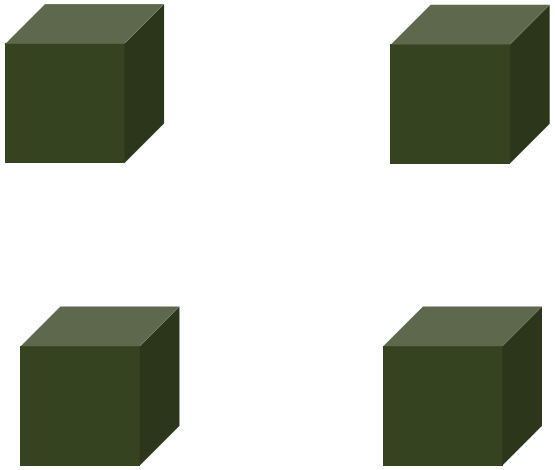
Di Girolamo, Salvatore, Konstantin Taranov, Andreas Kurth, Michael Schaffner, Timo Schneider, Jakub Beránek, Maciej Besta, Luca Benini, Duncan Roweth, and Torsten Hoefler. "Network-accelerated non-contiguous memory transfers." **SC '19**

Network-accelerated non-contiguous memory transfers



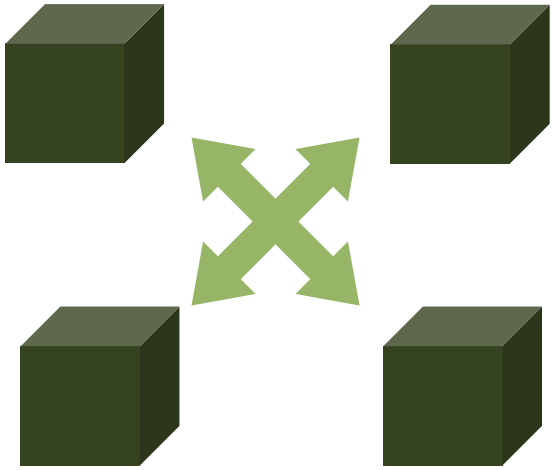
Di Girolamo, Salvatore, Konstantin Taranov, Andreas Kurth, Michael Schaffner, Timo Schneider, Jakub Beránek, Maciej Besta, Luca Benini, Duncan Roweth, and Torsten Hoefler. "Network-accelerated non-contiguous memory transfers." **SC '19**

Network-accelerated non-contiguous memory transfers



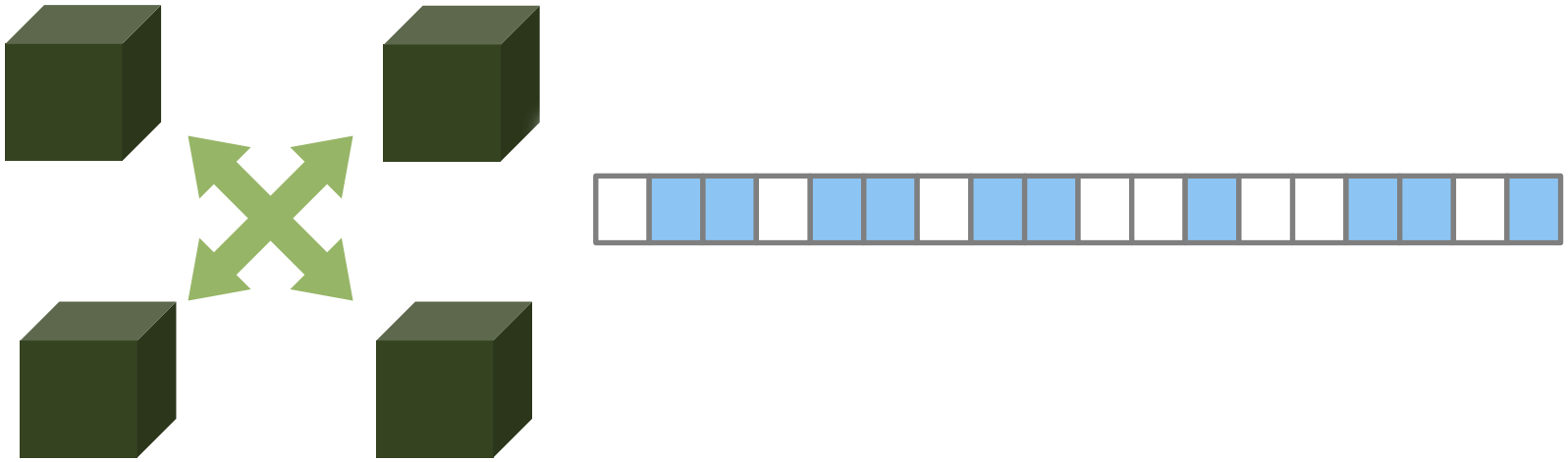
Di Girolamo, Salvatore, Konstantin Taranov, Andreas Kurth, Michael Schaffner, Timo Schneider, Jakub Beránek, Maciej Besta, Luca Benini, Duncan Roweth, and Torsten Hoefler. "Network-accelerated non-contiguous memory transfers." **SC '19**

Network-accelerated non-contiguous memory transfers



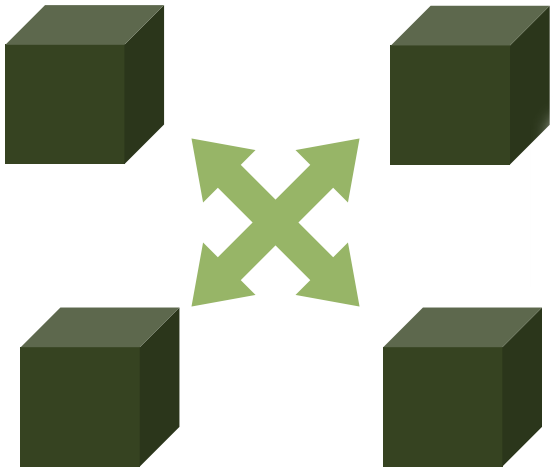
Di Girolamo, Salvatore, Konstantin Taranov, Andreas Kurth, Michael Schaffner, Timo Schneider, Jakub Beránek, Maciej Besta, Luca Benini, Duncan Roweth, and Torsten Hoefler. "Network-accelerated non-contiguous memory transfers." **SC '19**

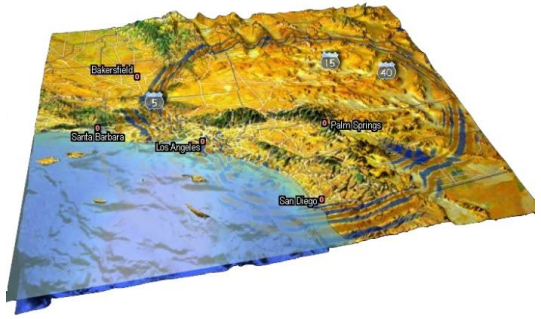
Network-accelerated non-contiguous memory transfers



Di Girolamo, Salvatore, Konstantin Taranov, Andreas Kurth, Michael Schaffner, Timo Schneider, Jakub Beránek, Maciej Besta, Luca Benini, Duncan Roweth, and Torsten Hoefler. "Network-accelerated non-contiguous memory transfers." **SC '19**

Network-accelerated non-contiguous memory transfers

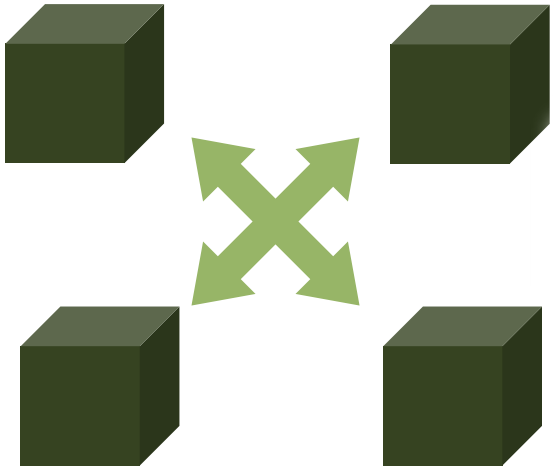


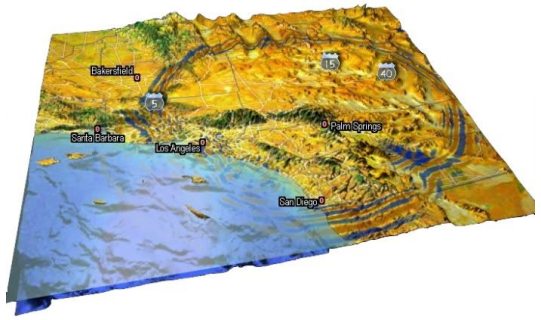


https://specfem3d.readthedocs.io/en/latest/

L. Carrington et al. High-frequency simulations of global seismic wave propagation using SPECFEM3D_GLOBE on 62K processors. SC 2008.

Network-accelerated non-contiguous memory transfers





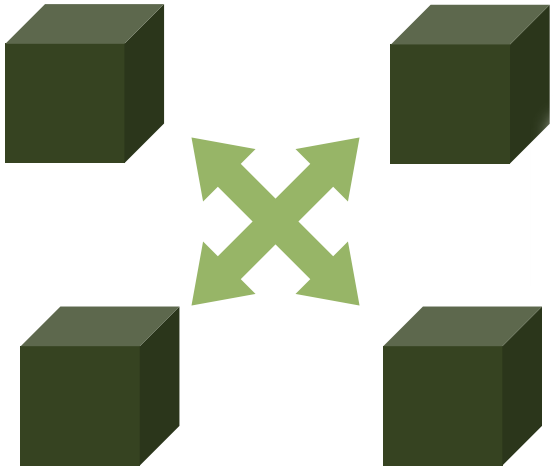
https://specfem3d.readthedocs.io/en/latest/

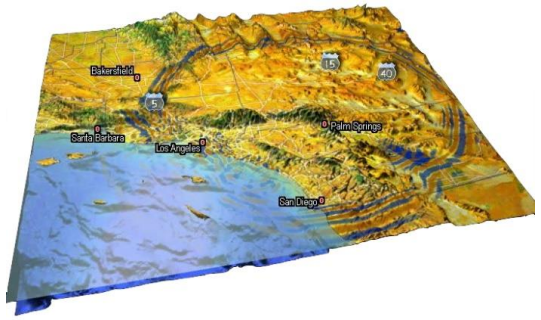
L. Carrington et al. High-frequency simulations of global seismic wave propagation using SPECFEM3D_GLOBE on 62K processors. SC 2008.

http://fourier.eng.hmc.edu/e161/lectures/fourier/node10.html

T. Hoefler et al. Parallel zero-copy algorithms for fast Fourier transform and conjugate gradient using MPI datatypes. EuroMPI 2010.

Network-accelerated non-contiguous memory transfers



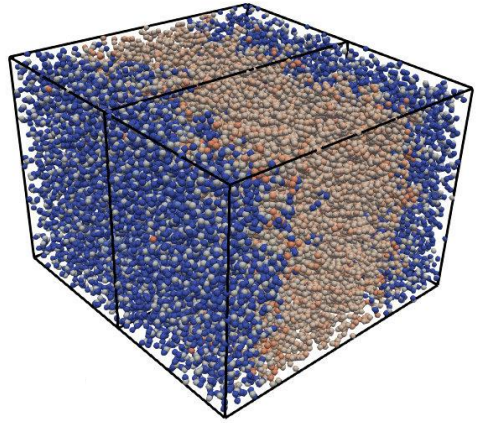


https://specfem3d.readthedocs.io/en/latest/

L. Carrington et al. High-frequency simulations of global seismic wave propagation using SPECFEM3D_GLOBE on 62K processors. SC 2008.

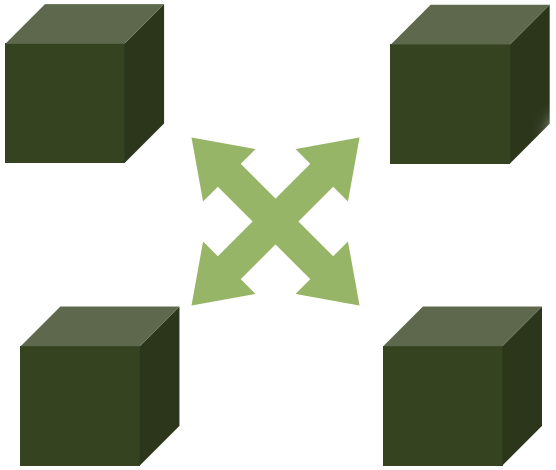
http://fourier.eng.hmc.edu/e161/lectures/fourier/node10.html

T. Hoefler et al. Parallel zero-copy algorithms for fast Fourier transform and conjugate gradient using MPI datatypes. EuroMPI 2010.



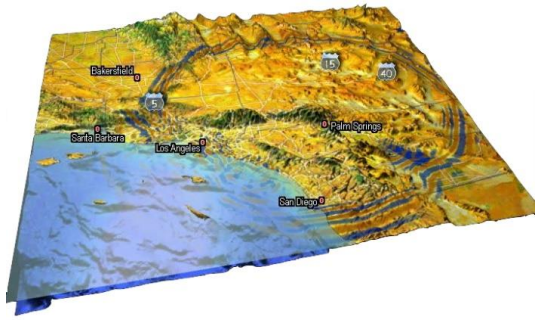
W. Usher et al. libIS: a lightweight library for flexible in transit visualization. ISAV 2018.

Network-accelerated non-contiguous memory transfers



MPI

Derived Datatypes

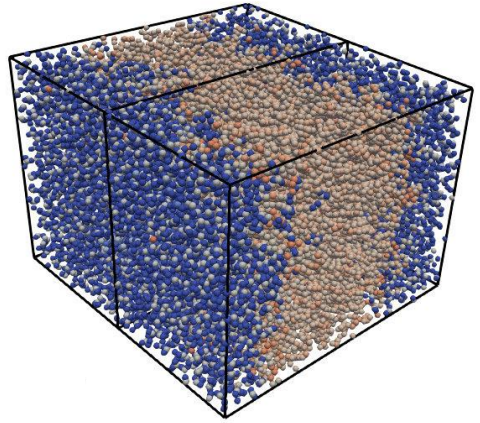


https://specfem3d.readthedocs.io/en/latest/

L. Carrington et al. High-frequency simulations of global seismic wave propagation using SPECFEM3D_GLOBE on 62K processors. SC 2008.

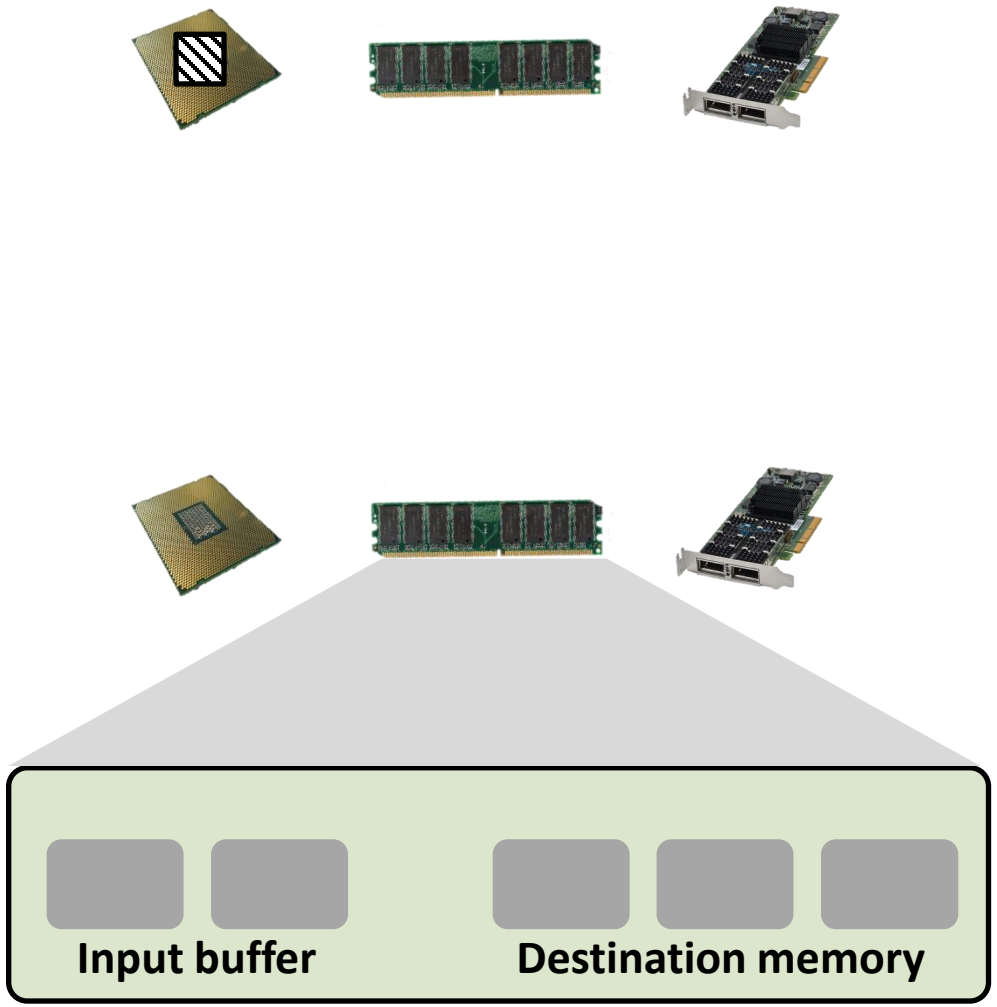
http://fourier.eng.hmc.edu/e161/lectures/fourier/node10.html

T. Hoefler et al. Parallel zero-copy algorithms for fast Fourier transform and conjugate gradient using MPI datatypes. EuroMPI 2010.



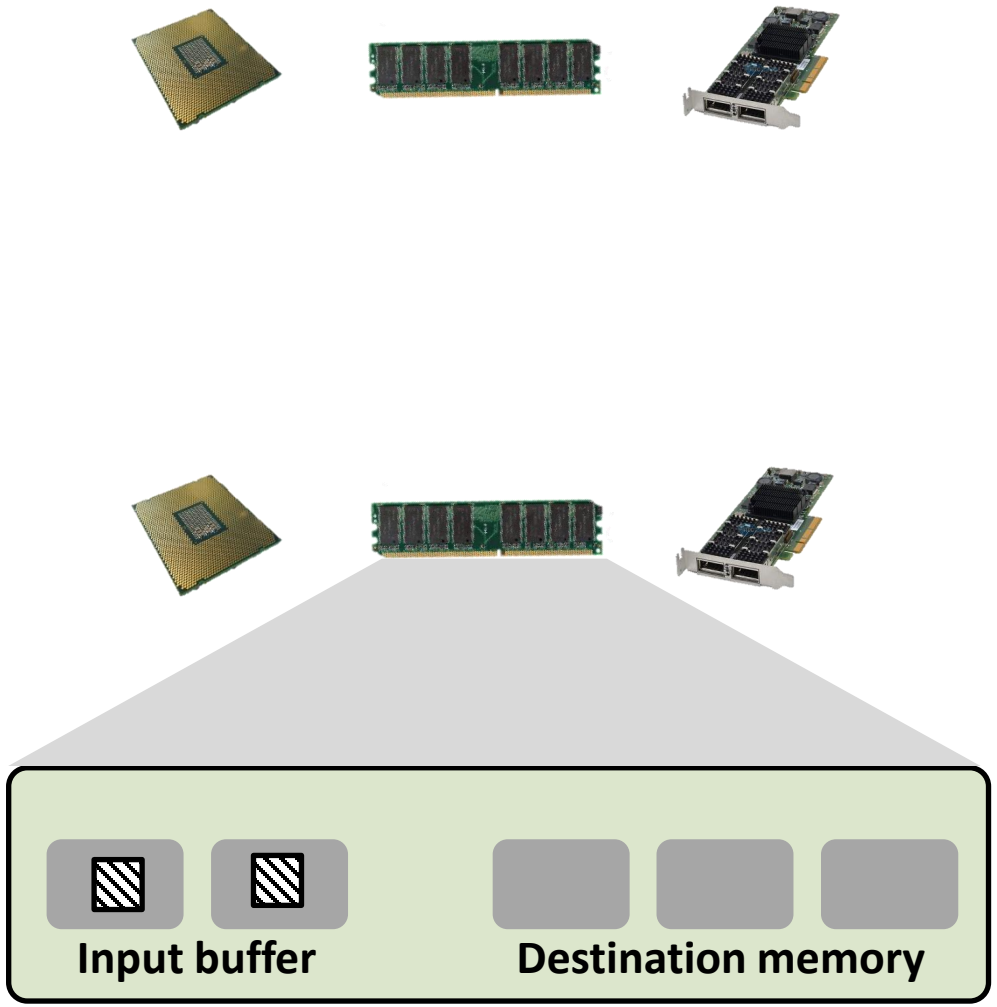
W. Usher et al. libIS: a lightweight library for flexible in transit visualization. ISAV 2018.

MPI Datatypes Processing



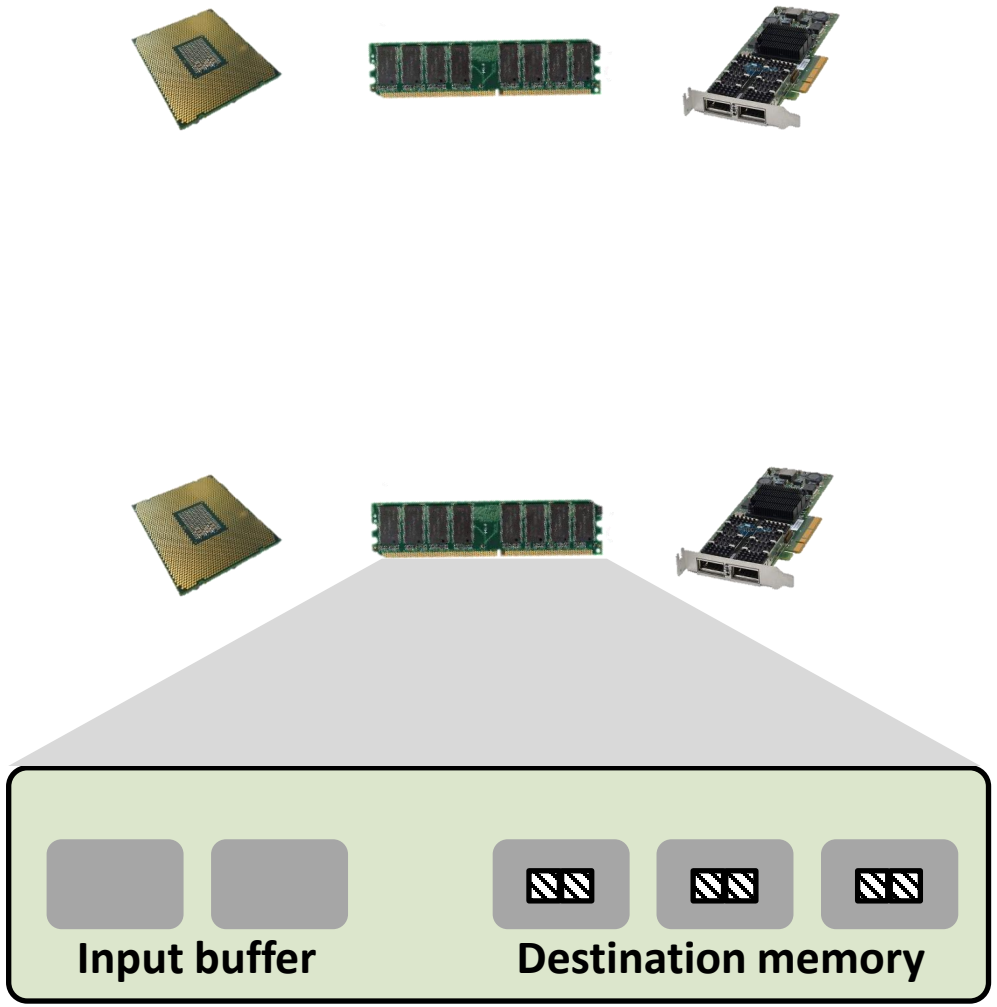
Gropp, W., et al., March. Improving the performance of MPI derived datatypes. *MPIDC'99*
Torsten Hoefler, Salvatore Di Girolamo, Konstantin Taranov, Ryan E. Grant, and Ron Brightwell. 2017. sPIN: High-performance streaming Processing In the Network. SC'17

MPI Datatypes Processing



Gropp, W., et al., March. Improving the performance of MPI derived datatypes. *MPIDC'99*
Torsten Hoefler, Salvatore Di Girolamo, Konstantin Taranov, Ryan E. Grant, and Ron Brightwell. 2017. sPIN: High-performance streaming Processing In the Network. SC'17

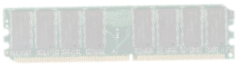
MPI Datatypes Processing

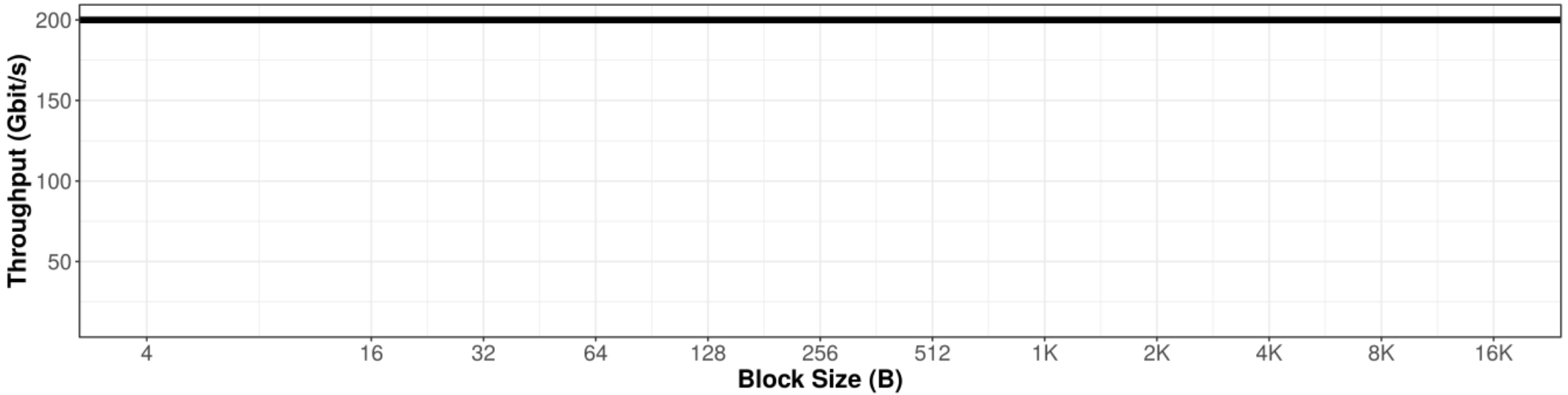


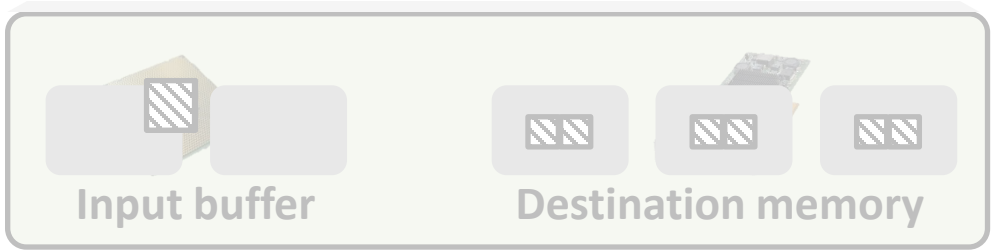
Gropp, W., et al., March. Improving the performance of MPI derived datatypes. *MPIDC'99*

Torsten Hoefler, Salvatore Di Girolamo, Konstantin Taranov, Ryan E. Grant, and Ron Brightwell. 2017. sPIN: High-performance streaming Processing In the Network. SC'17

MPI Datatypes Processing

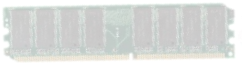


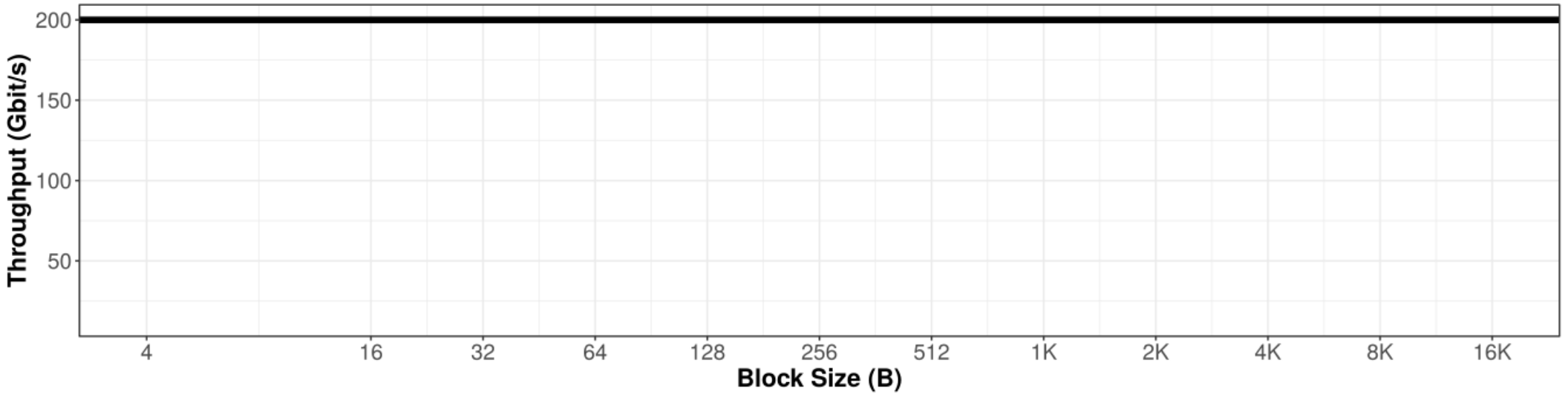


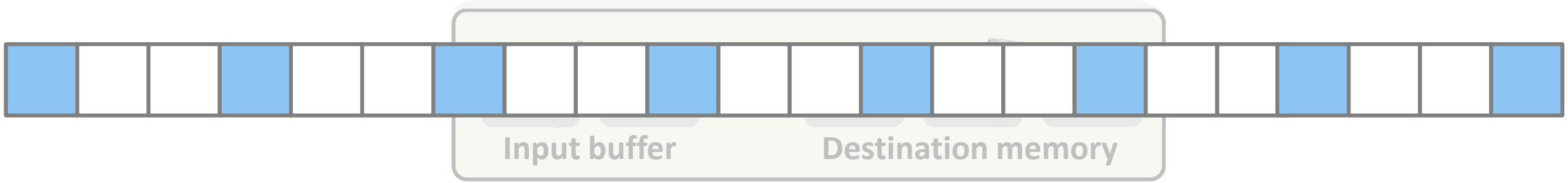


Gropp, W., et al., March. Improving the performance of MPI derived datatypes. *MPIDC'99*

MPI Datatypes Processing

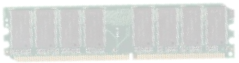


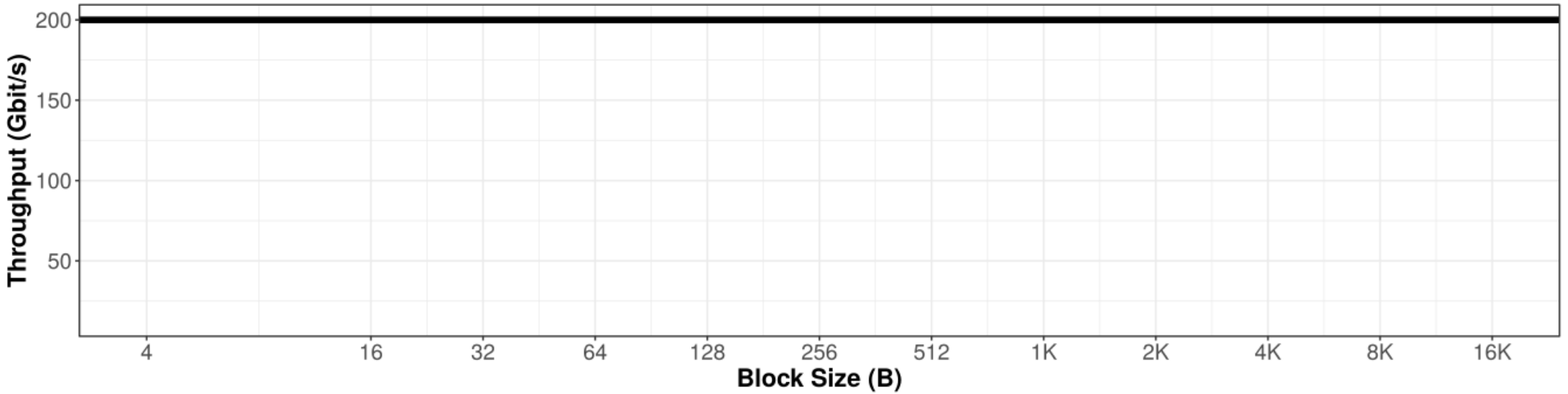




Gropp, W., et al., March. Improving the performance of MPI derived datatypes. *MPIDC'99*

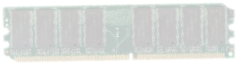
MPI Datatypes Processing

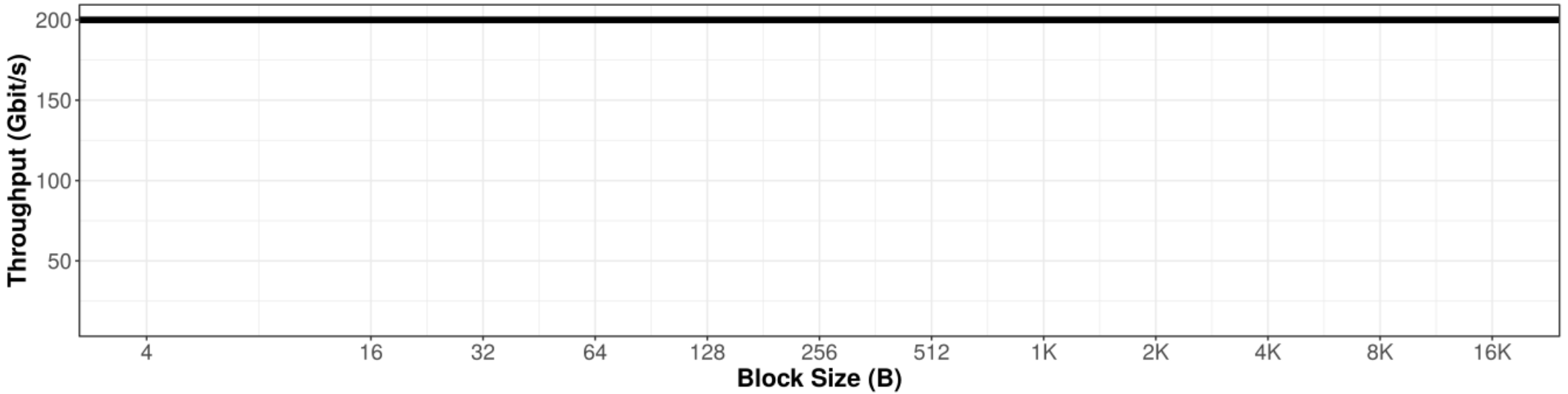




Gropp, W., et al., March. Improving the performance of MPI derived datatypes. *MPIDC'99*

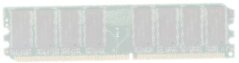
MPI Datatypes Processing

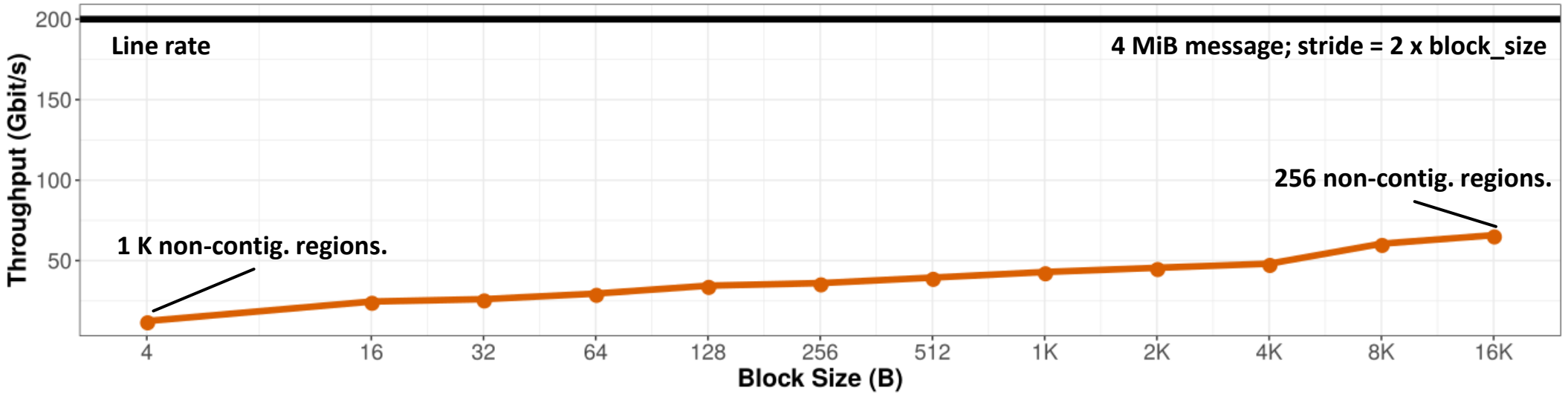




Gropp, W., et al., March. Improving the performance of MPI derived datatypes. *MPIDC'99*

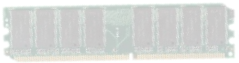
MPI Datatypes Processing

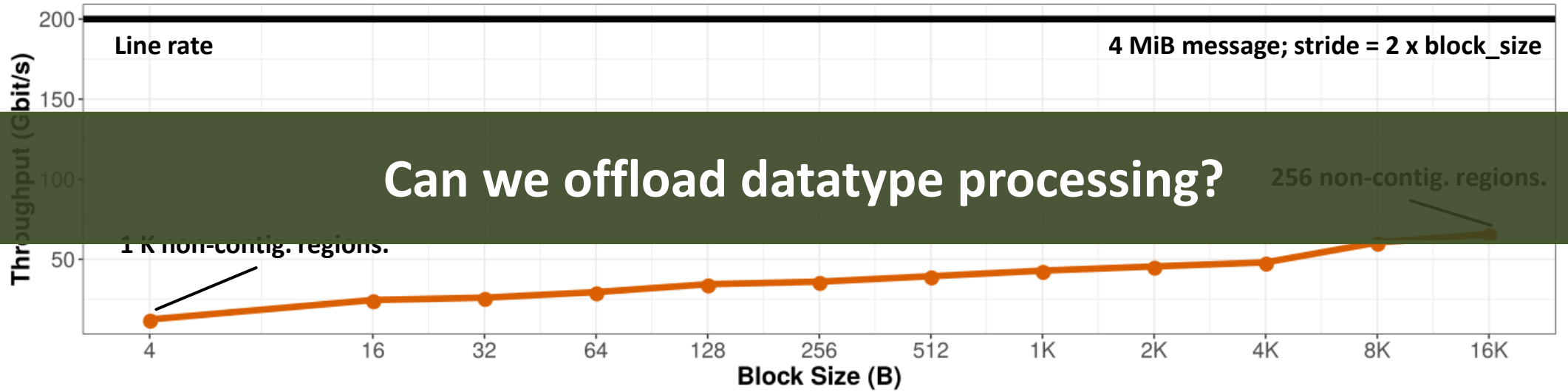




Gropp, W., et al., March. Improving the performance of MPI derived datatypes. *MPIDC'99*

MPI Datatypes Processing



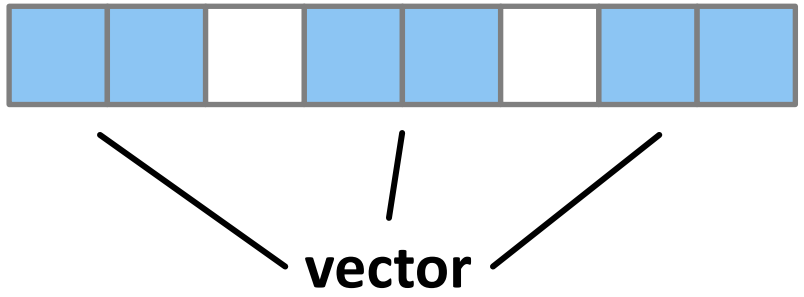


Can we offload datatype processing?

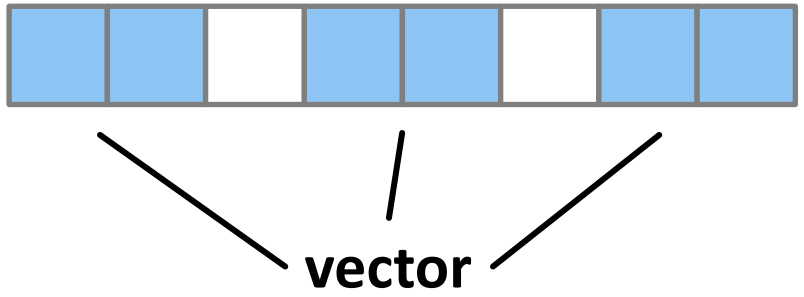
Gropp, W., et al., March. Improving the performance of MPI derived datatypes. *MPIDC'99*

Specialized Handlers

Specialized Handlers

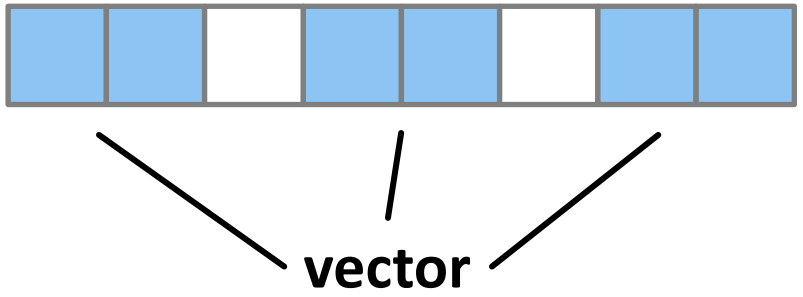


Specialized Handlers



spin_vec_t:
num_blocks: 3
block_size: 2
stride: 3
base_type: int

Specialized Handlers



spin_vec_t:
num_blocks: 3
block_size: 2
stride: 3
base_type: int

```
__handler vector_payload_handler(handler_args_t *args)
{
    spin_vec_t *ddt_descr = (spin_vec_t *)args->mem;
    uint32_t num_blocks = args->packet_len / ddt_descr->block_size;
    uint32_t stride = ddt_descr->stride;

    uint8_t *pkt_payload = args->pkt_payload_ptr;

    uint8_t *host_base_ptr = args->host_address;
    uint32_t host_offset = args->pkt_offset / (ddt_descr->block_size) * stride;
    uint8_t *host_address = host_base_ptr + host_offset;

    for (uint32_t i=0; i<num_blocks; i++)
    {
        PtlHandlerDMAToHostNB(host_address, pkt_payload, block_size, DMA_NO_EVENT);
        pkt_payload += block_size;
        host_address += stride;
    }

    return SPIN_SUCCESS;
}
```

Handler

Specialized Handlers

NIC Memory

vector

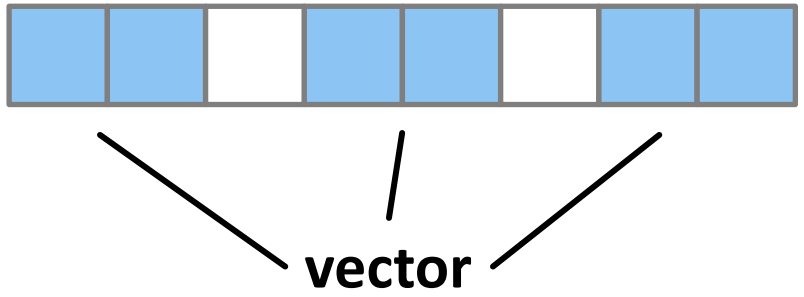
spin_vec_t:
num_blocks: 3
block_size: 2
stride: 3
base_type: int

Handler

Specialized Handlers

NIC Memory

spin_vec_t:
num_blocks: 3
block_size: 2
stride: 3
base_type: int




```
__handler vector_payload_handler(handler_args_t *args)
{
    spin_vec_t *ddt_descr = (spin_vec_t *)args->mem;
    uint32_t num_blocks = args->packet_len / ddt_descr->block_size;
    uint32_t stride = ddt_descr->stride;

    uint8_t *pkt_payload = args->pkt_payload_ptr;

    uint8_t *host_base_ptr = args->host_address;
    uint32_t host_offset = ddt_descr->pkt_offset / (ddt_descr->block_size) * stride;
    uint8_t *host_address = host_base_ptr + host_offset;

    for (uint32_t i=0; i<num_blocks; i++)
    {
        PtlHandlerDMAToHostNB(host_address, pkt_payload, block_size, DMA_NO_EVENT);
        pkt_payload += block_size;
        host_address += stride;
    }

    return SPIN_SUCCESS;
}
```

Handler

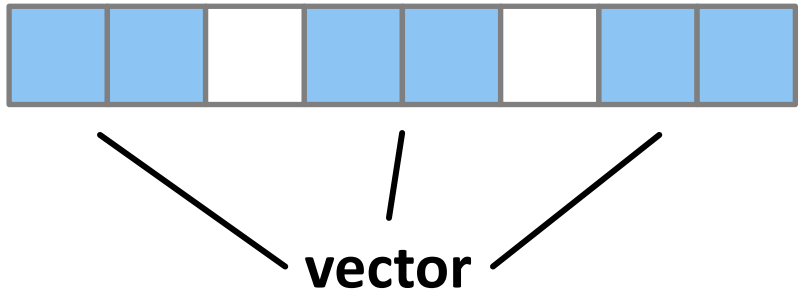
Specialized Handlers

NIC Memory

spin_vec_t:
num_blocks: 3
block_size: 2
stride: 3
base_type: int

```
__handler vector_payload_handler(handler_args_t *args)
{
    spin_vec_t *ddt_descr = (spin_vec_t *)args->mem;
    uint32_t num_blocks = args->packet_len / ddt_descr->block_size;
    uint32_t stride = ddt_descr->stride;

    uint8_t *pkt_payload = args->pkt_payload_ptr;

    uint8_t *host_base_ptr = args->host_address;
    uint32_t host_offset = (args->pkt_offset / ddt_descr->block_size) * stride;
    uint8_t *host_address = host_base_ptr + host_offset;

    for (uint32_t i=0; i<num_blocks; i++)
    {
        PtlHandlerDMAToHostNB(host_address, pkt_payload, block_size, DMA_NO_EVENT);
        pkt_payload += block_size;
        host_address += stride;
    }

    return SPIN_SUCCESS;
}
```

Handler



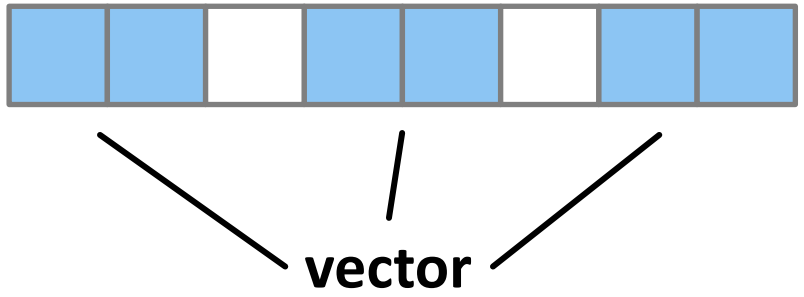
Specialized Handlers

NIC Memory

spin_vec_t:
num_blocks: 3
block_size: 2
stride: 3
base_type: int

```
__handler vector_payload_handler(handler_args_t *args)
{
    spin_vec_t *ddt_descr = (spin_vec_t *)args->mem;
    uint32_t num_blocks = args->packet_len / ddt_descr->block_size;
    uint32_t stride = ddt_descr->stride;

    uint8_t *pkt_payload = args->pkt_payload_ptr;

    uint8_t *host_base_ptr = args->host_address;
    uint32_t host_offset = (args->pkt_offset / ddt_descr->block_size) * stride;
    uint8_t *host_address = host_base_ptr + host_offset;

    for (uint32_t i=0; i<num_blocks; i++)
    {
        PtlHandlerDMAToHostNB(host_address, pkt_payload, block_size, DMA_NO_EVENT);
        pkt_payload += block_size;
        host_address += stride;
    }

    return SPIN_SUCCESS;
}
```

Handler

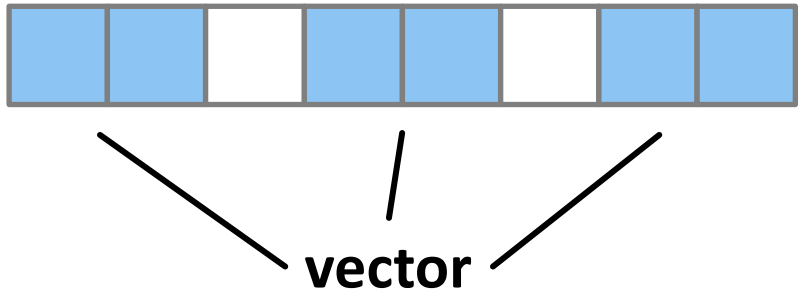


Specialized Handlers

NIC Memory

spin_vec_t:
num_blocks: 3
block_size: 2
stride: 3
base_type: int

Handler

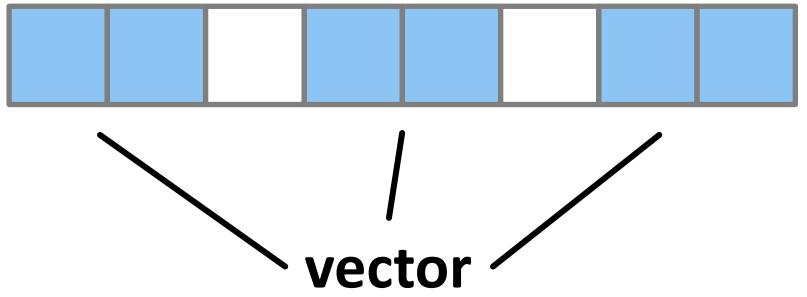



```
__handler vector_payload_handler(handler_args_t *args)
{
    spin_vec_t *ddt_descr = (spin_vec_t *)args->mem;
    uint32_t num_blocks = args->packet_len / ddt_descr->block_size;
    uint32_t stride = ddt_descr->stride;

    uint8_t *pkt_payload = args->pkt_payload_ptr;

    uint8_t *host_base_ptr = args->host_address;
    uint32_t host_offset = (args->pkt_offset / ddt_descr->block_size;) * stride;
    uint8_t *host_address = host_base_addr + host_offset;

    for (uint32_t i=0; i<num_blocks; i++)
    {
        PtlHandlerDMAToHostNB(host_address, pkt_payload, block_size, DMA_NO_EVENT);
        pkt_payload += block_size;
        host_address += stride;
    }

    return SPIN_SUCCESS;
}
```

Specialized Handlers

NIC Memory

spin_vec_t:
num_blocks: 3
block_size: 2
stride: 3
base_type: int




```
__handler vector_payload_handler(handler_args_t *args)
{
    spin_vec_t *ddt_descr = (spin_vec_t *)args->mem;
    uint32_t num_blocks = args->packet_len / ddt_descr->block_size;
    uint32_t stride = ddt_descr->stride;

    uint8_t *pkt_payload = args->pkt_payload_ptr;

    uint8_t *host_base_ptr = args->host_address;
    uint32_t host_offset = (args->pkt_offset / ddt_descr->block_size;) * stride;
    uint8_t *host_address = host_base_addr + host_offset;

    for (uint32_t i=0; i<num_blocks; i++)
    {
        PtlHandlerDMAToHostNB(host_address, pkt_payload, block_size, DMA_NO_EVENT);
        pkt_payload += block_size;
        host_address += stride;
    }

    return SPIN_SUCCESS;
}
```

Load DDT info

Specialized Handlers

NIC Memory

spin_vec_t:
num_blocks: 3
block_size: 2
stride: 3
base_type: int

Handler

vector

```
__handler vector_payload_handler(handler_args_t *args)
{
    spin_vec_t *ddt_descr = (spin_vec_t *)args->mem;
    uint32_t num_blocks = args->packet_len / ddt_descr->block_size;
    uint32_t stride = ddt_descr->stride;

    uint8_t *pkt_payload = args->pkt_payload_ptr;

    uint8_t *host_base_ptr = args->host_address;
    uint32_t host_offset = (args->pkt_offset / ddt_descr->block_size) * stride;
    uint8_t *host_address = host_base_addr + host_offset;

    for (uint32_t i=0; i<num_blocks; i++)
    {
        PtlHandlerDMAToHostNB(host_address, pkt_payload, block_size, DMA_NO_EVENT);
        pkt_payload += block_size;
        host_address += stride;
    }

    return SPIN_SUCCESS;
}
```

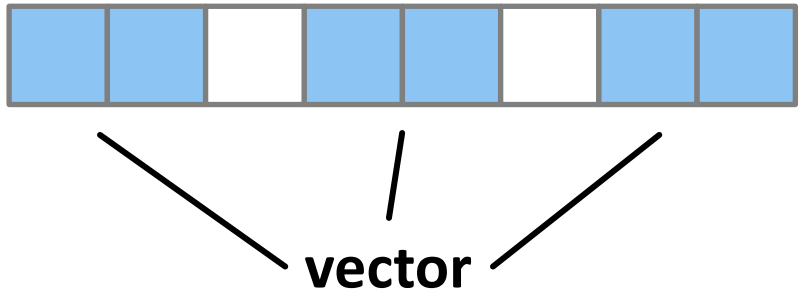
Load DDT info

Compute host memory destination address

Specialized Handlers

NIC Memory

spin_vec_t:
num_blocks: 3
block_size: 2
stride: 3
base_type: int




```
__handler vector_payload_handler(handler_args_t *args)
{
    spin_vec_t *ddt_descr = (spin_vec_t *)args->mem;
    uint32_t num_blocks = args->packet_len / ddt_descr->block_size;
    uint32_t stride = ddt_descr->stride;

    uint8_t *pkt_payload = args->pkt_payload_ptr;

    uint8_t *host_base_ptr = args->host_address;
    uint32_t host_offset = (args->pkt_offset / ddt_descr->block_size) * stride;
    uint8_t *host_address = host_base_addr + host_offset;

    for (uint32_t i=0; i<num_blocks; i++)
    {
        PtlHandlerDMAToHostNB(host_address, pkt_payload, block_size, DMA_NO_EVENT);
        pkt_payload += block_size;
        host_address += stride;
    }

    return SPIN_SUCCESS;
}
```

Load DDT info

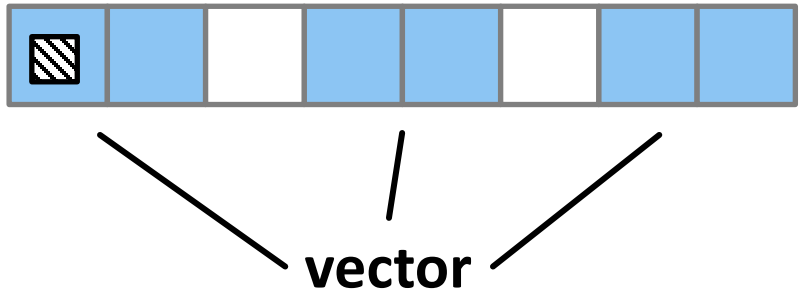
Compute host memory destination address

DMA all contig. regions contained in the packet

Specialized Handlers

NIC Memory

spin_vec_t:
num_blocks: 3
block_size: 2
stride: 3
base_type: int



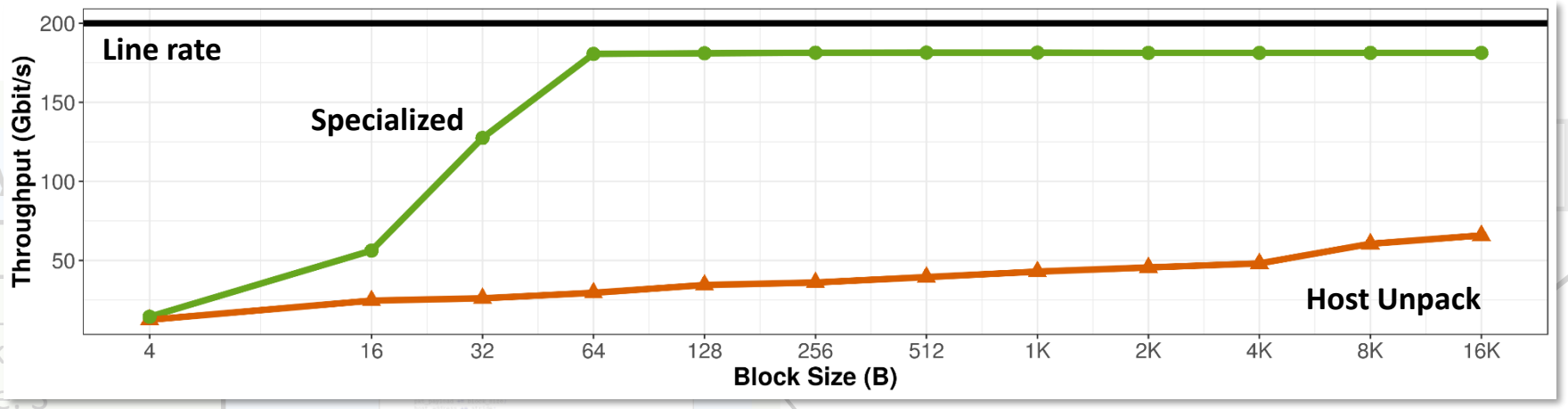

```
__handler vector_payload_handler(handler_args_t *args)
{
    spin_vec_t *ddt_descr = (spin_vec_t *)args->mem;
    uint32_t num_blocks = args->packet_len / ddt_descr->block_size;
    uint32_t stride = ddt_descr->stride;

    uint8_t *pkt_payload = args->pkt_payload_ptr;

    uint8_t *host_base_ptr = args->host_address;
    uint32_t host_offset = (args->pkt_offset / ddt_descr->block_size) * stride;
    uint8_t *host_address = host_base_addr + host_offset;

    for (uint32_t i=0; i<num_blocks; i++)
    {
        PtlHandlerDMAToHostNB(host_address, pkt_payload, block_size, DMA_NO_EVENT);
        pkt_payload += block_size;
        host_address += stride;
    }

    return SPIN_SUCCESS;
}
```

Load DDT info

Compute host memory destination address

DMA all contig. regions contained in the packet

Specialized Handlers

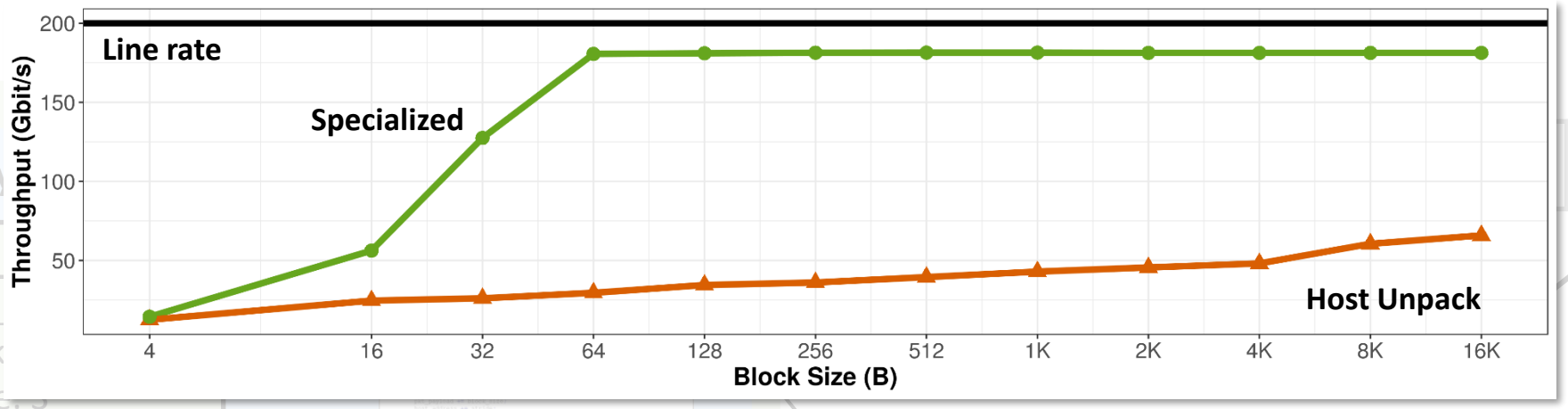


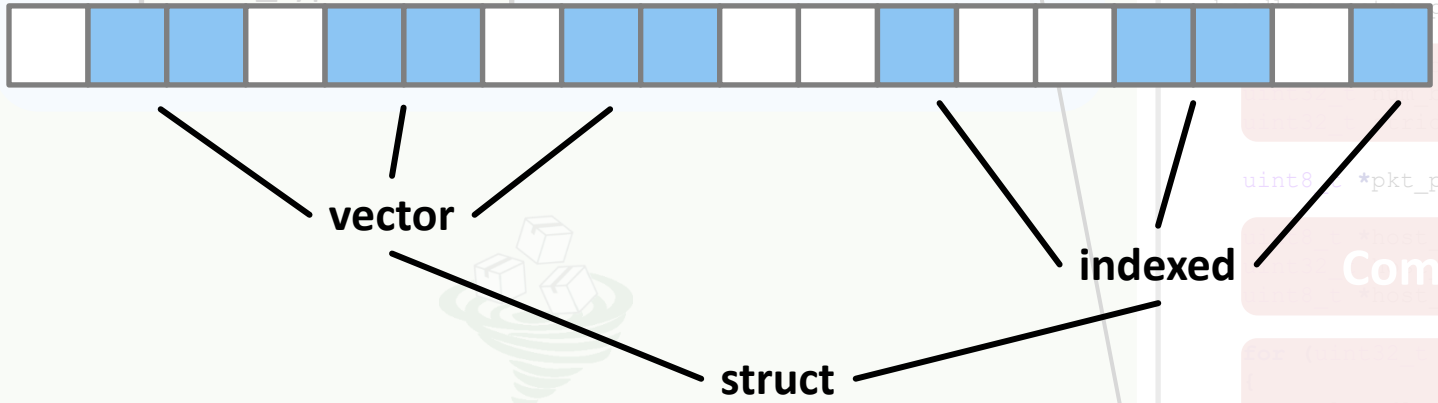
NIC Memory

```
spin_
num_
block
stride
base_type: int
```

```
__handler vector_payload_handler(handler_args_t *args)
{
    Load DDT info

    uint8_t *pkt_payload = args->pkt_payload_ptr;

    Compute host memory destination address

    DMA all contig. regions contained in  the packet

    return SPIN_SUCCESS;
}
```

Specialized Handlers



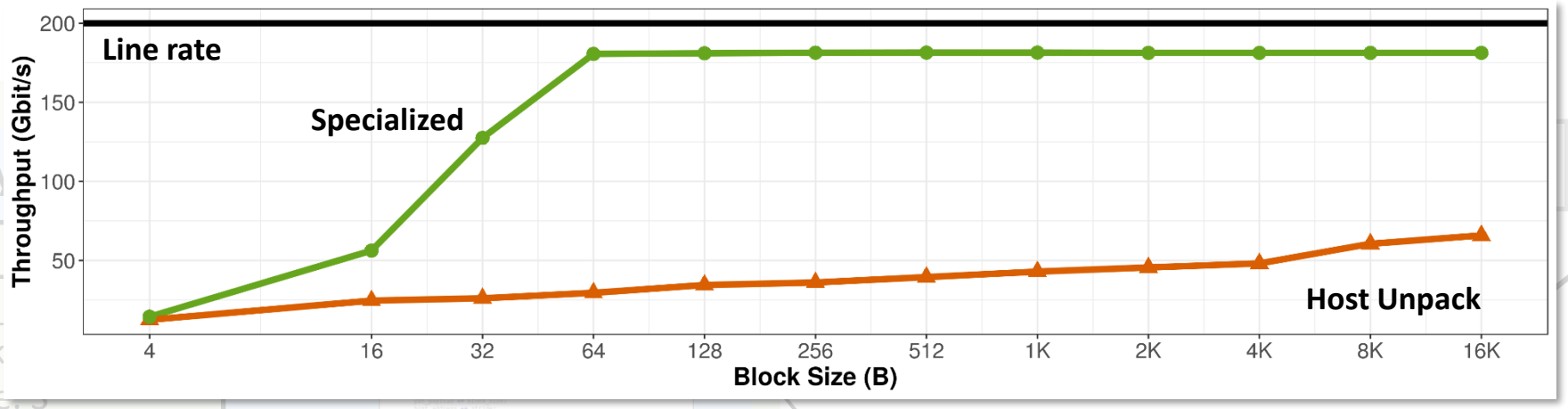


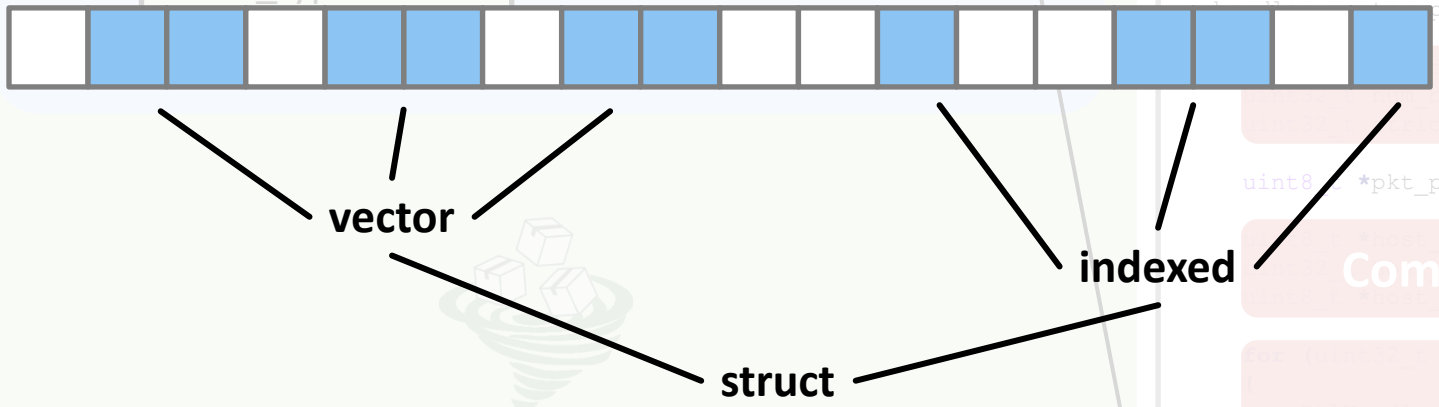
Load DDT info

Compute host memory destination address

DMA all contig. regions contained in the packet

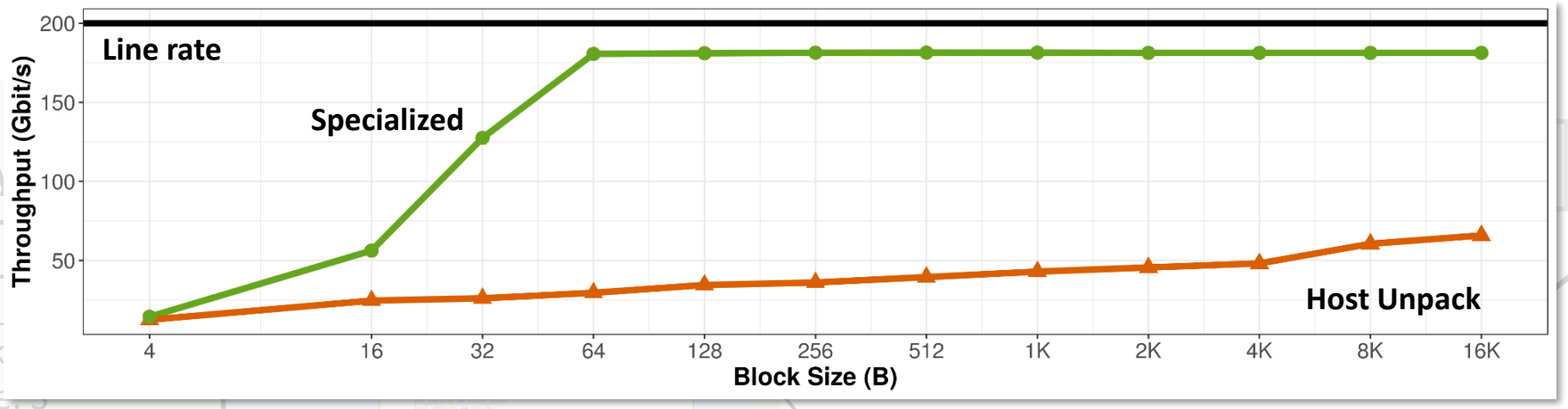
Specialized Handlers

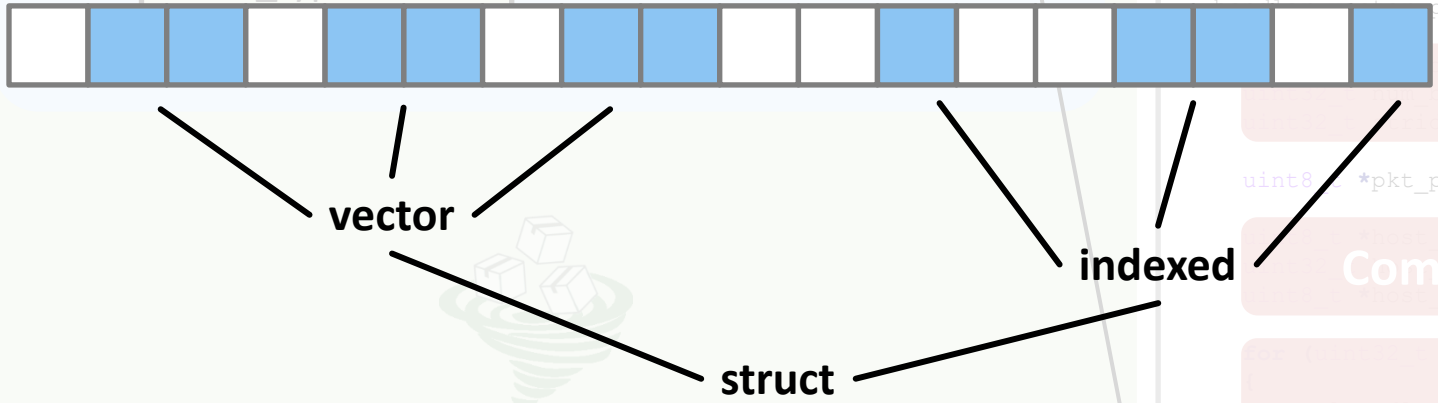


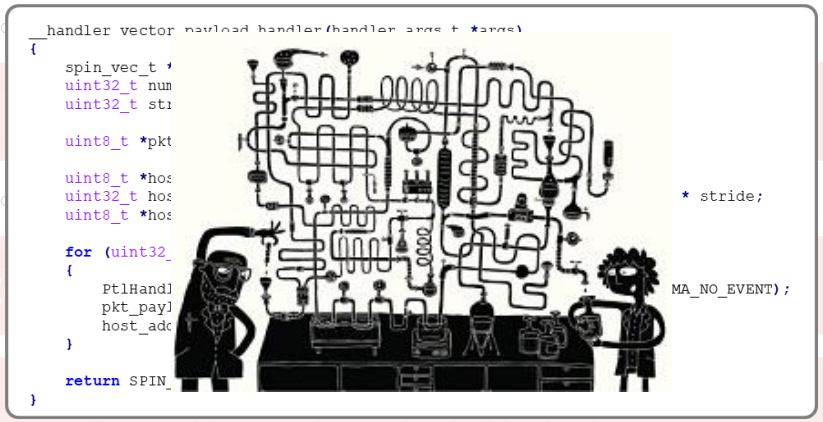



```
__handler vector_payload_handler(handler_args_t *args)
{
    spin_vec_t *ddt_descr = (spin_vec_t *)args->mem;
    uint32_t num_blocks = args->packet_len / ddt_descr->block_size;
    uint32_t stride = ddt_descr->stride;

    uint8_t *pkt_payload = args->pkt_payload_ptr;

    uint8_t *host_base_ptr = args->host_address;
    uint32_t host_offset = (args->pkt_offset / ddt_descr->block_size;) * stride;
    uint8_t *host_address = host_base_addr + host_offset;

    for (uint32_t i=0; i<num_blocks; i++)
    {
        PtlHandlerDMAToHostNB(host_address, pkt_payload, block_size, DMA_NO_EVENT);
        pkt_payload += block_size;
        host_address += stride;
    }

    return SPIN_SUCCESS;
}
```

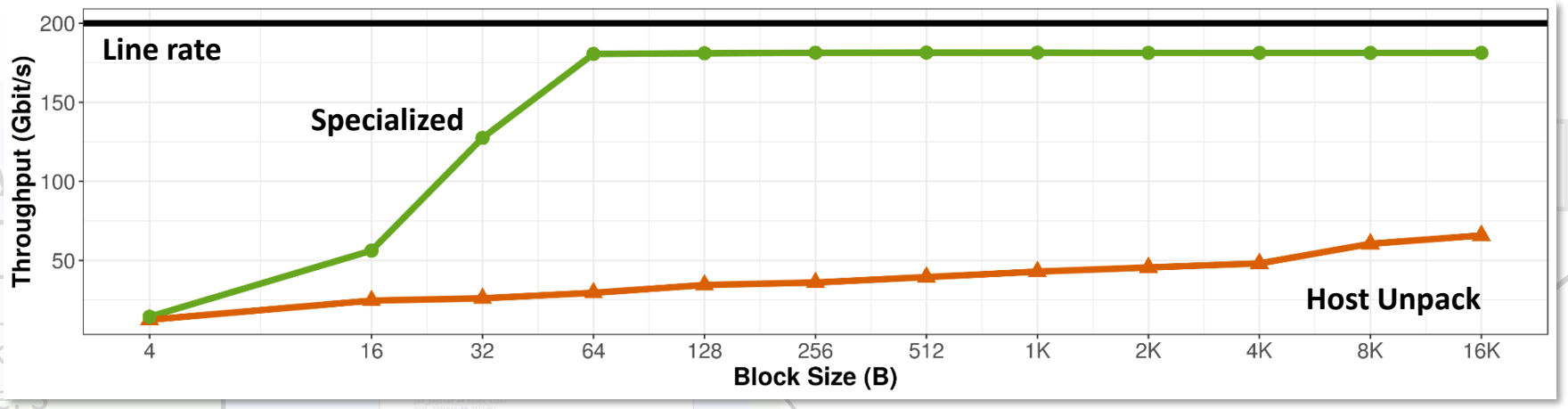
Specialized Handlers

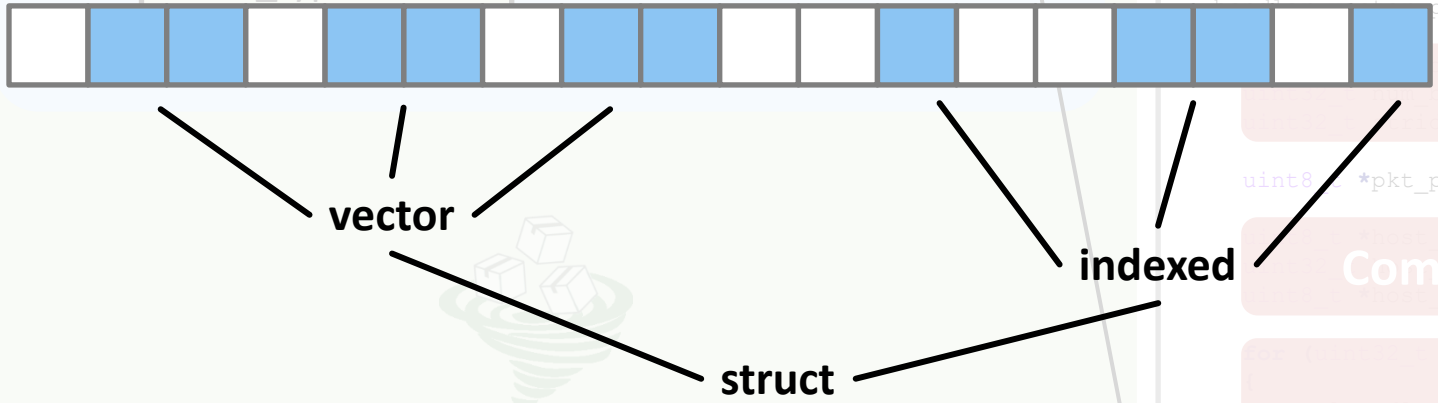


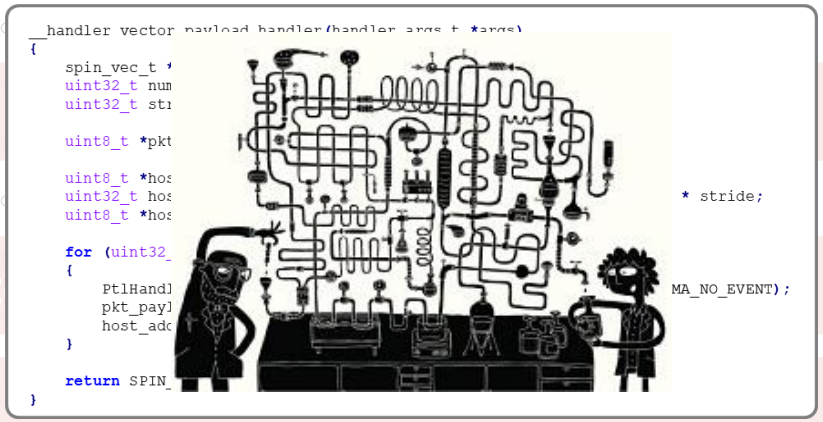




Specialized Handlers

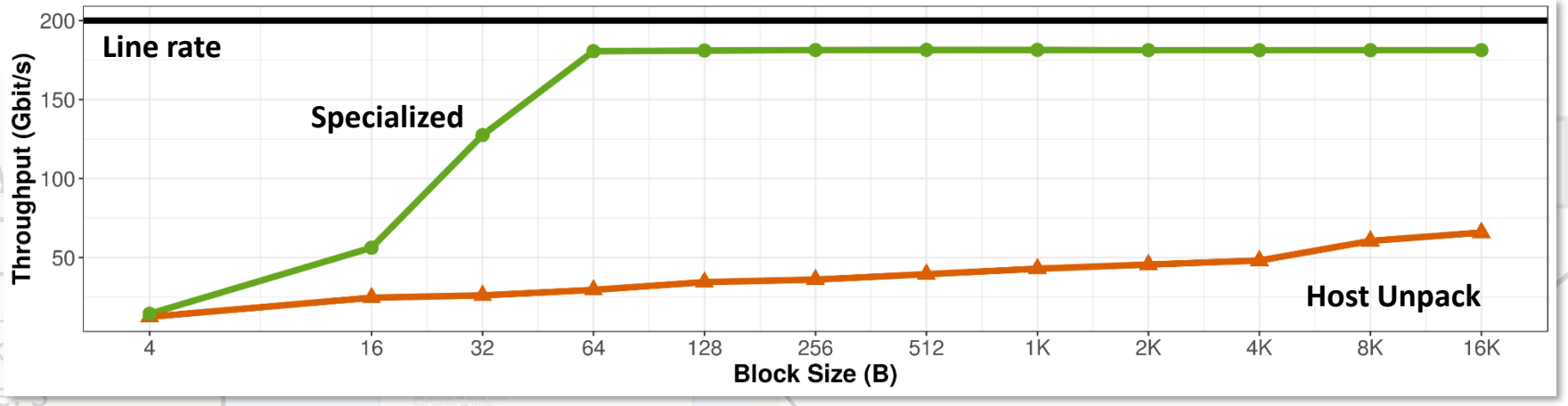


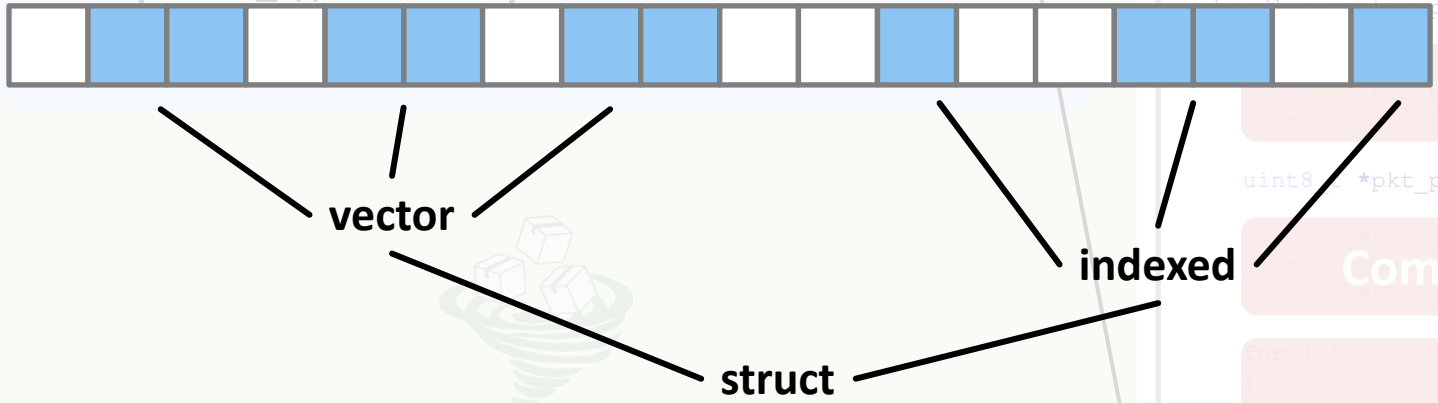


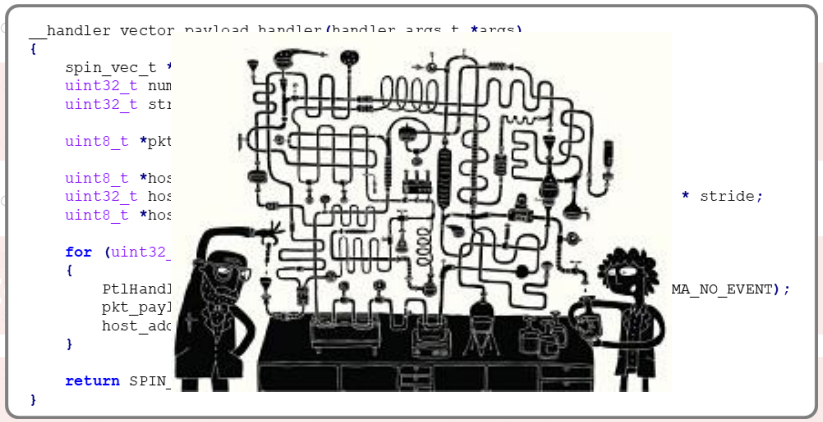


Need a different handlers for each possible derived datatype!

Specialized Handlers







Can we define a general handler to process any datatype?

MPI Types Library on sPIN

Ross, Robert, et al. "Processing MPI datatypes outside MPI." European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting. Springer, Berlin, Heidelberg, 2009.

MPI Types Library on sPIN

Ross, Robert, et al. "Processing MPI datatypes outside MPI." European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting. Springer, Berlin, Heidelberg, 2009.

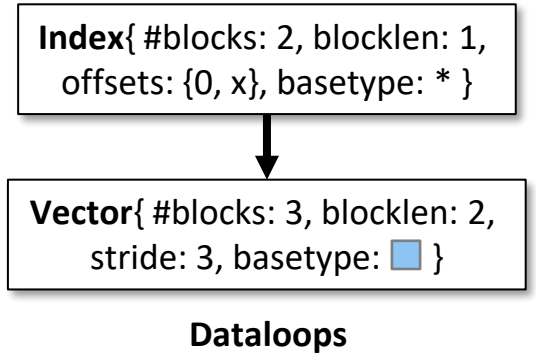
MPI Types Library on sPIN

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: ▭ }

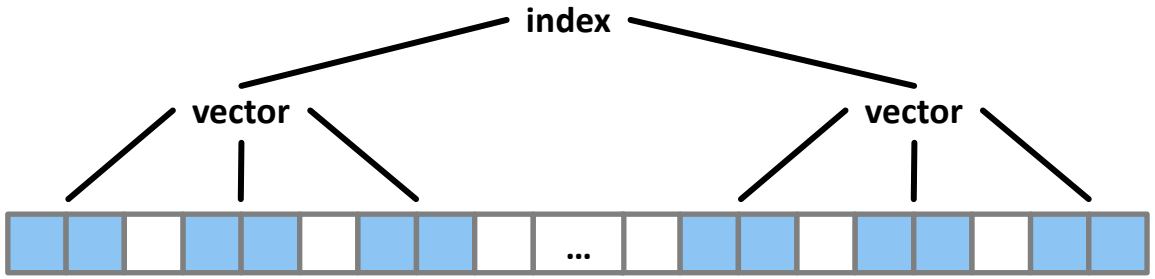
Dataloops

Ross, Robert, et al. "Processing MPI datatypes outside MPI." European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting. Springer, Berlin, Heidelberg, 2009.

MPI Types Library on sPIN

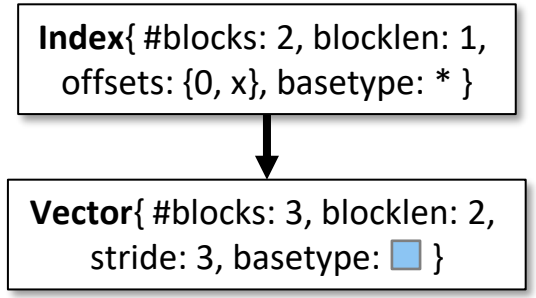


Dataloops



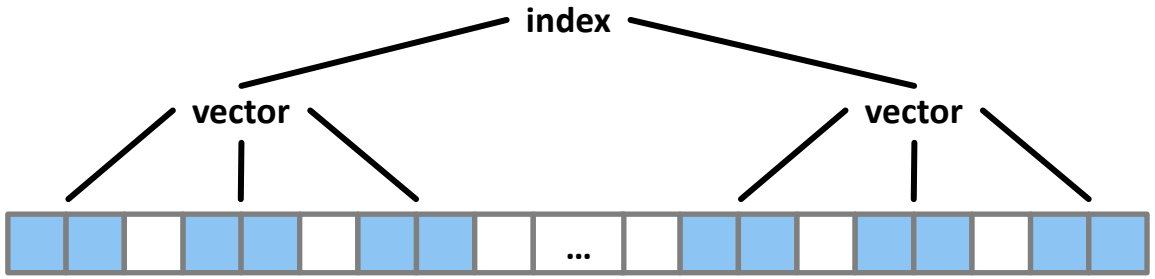
Ross, Robert, et al. "Processing MPI datatypes outside MPI." European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting. Springer, Berlin, Heidelberg, 2009.

MPI Types Library on sPIN



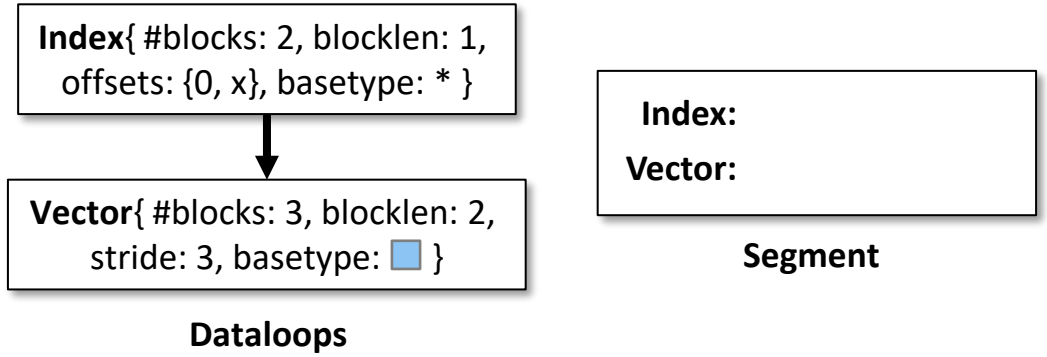
Dataloops

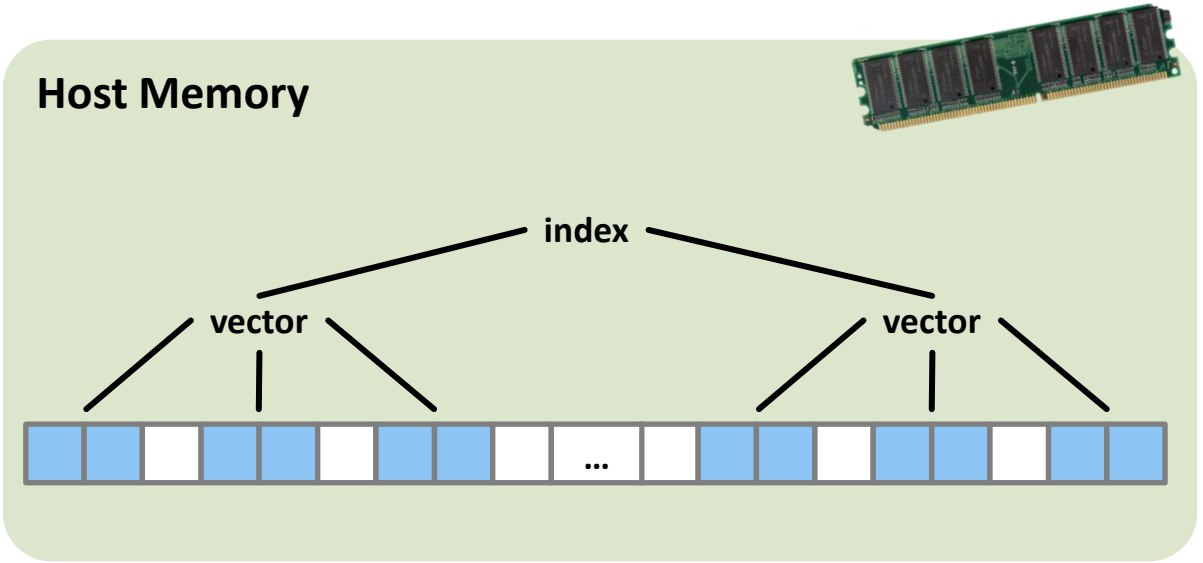
Segment



Ross, Robert, et al. "Processing MPI datatypes outside MPI." European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting. Springer, Berlin, Heidelberg, 2009.

MPI Types Library on sPIN





Ross, Robert, et al. "Processing MPI datatypes outside MPI." European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting. Springer, Berlin, Heidelberg, 2009.

MPI Types Library on sPIN

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: ▢ }

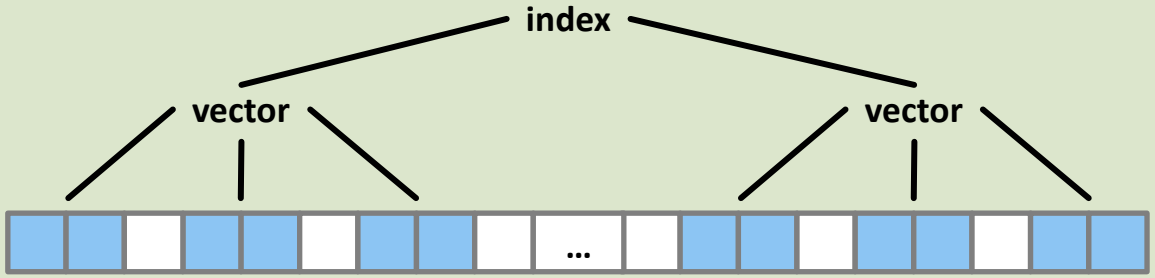
Dataloops

Index:

Vector:

Segment

Host Memory



Ross, Robert, et al. "Processing MPI datatypes outside MPI." European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting. Springer, Berlin, Heidelberg, 2009.

MPI Types Library on sPIN

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: ▢ }

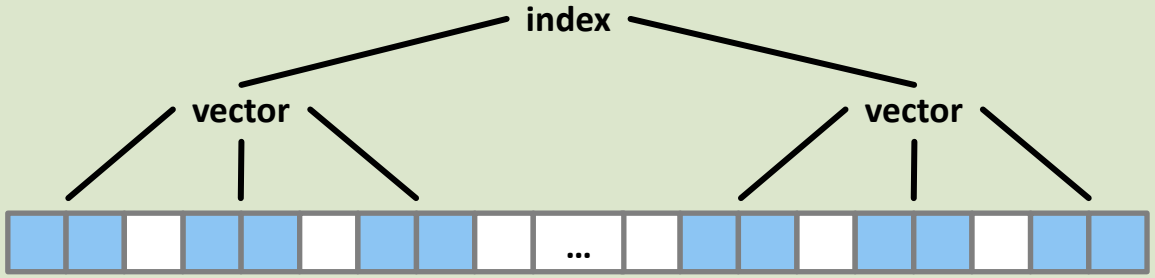
Dataloops

Index:

Vector:

Segment

Host Memory



Ross, Robert, et al. "Processing MPI datatypes outside MPI." European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting. Springer, Berlin, Heidelberg, 2009.

MPI Types Library on sPIN

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

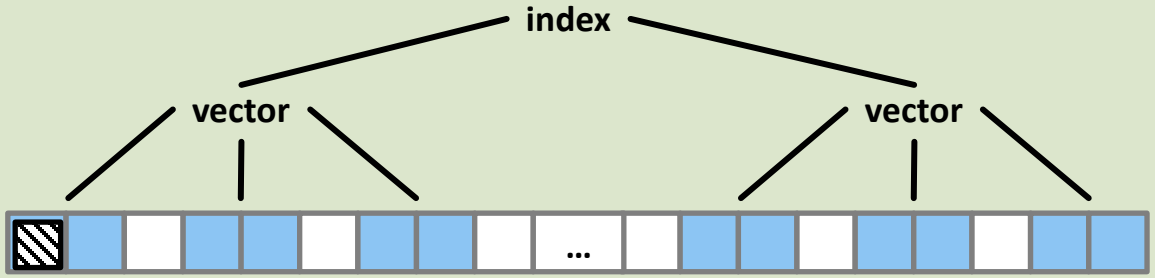
Dataloops

Index:

Vector:

Segment

Host Memory



Ross, Robert, et al. "Processing MPI datatypes outside MPI." European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting. Springer, Berlin, Heidelberg, 2009.

MPI Types Library on sPIN

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

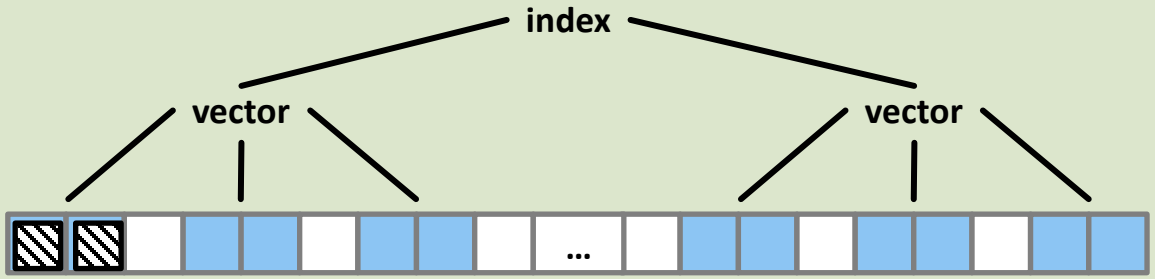
Dataloops

Index:

Vector:

Segment

Host Memory



Ross, Robert, et al. "Processing MPI datatypes outside MPI." European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting. Springer, Berlin, Heidelberg, 2009.

MPI Types Library on sPIN

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

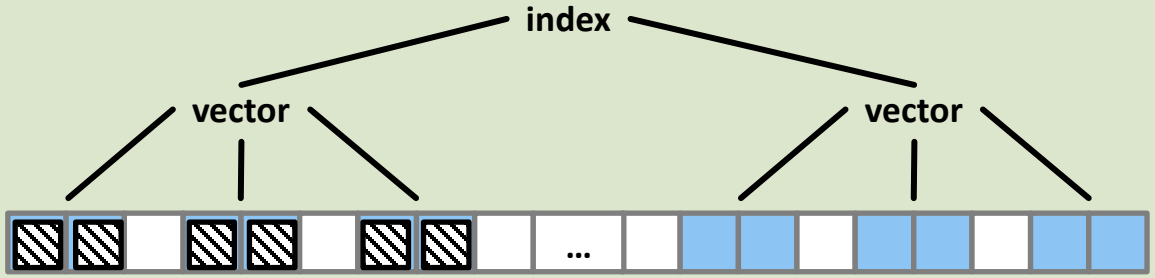
Dataloops

Index:

Vector:

Segment

Host Memory



Ross, Robert, et al. "Processing MPI datatypes outside MPI." European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting. Springer, Berlin, Heidelberg, 2009.

MPI Types Library on sPIN

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

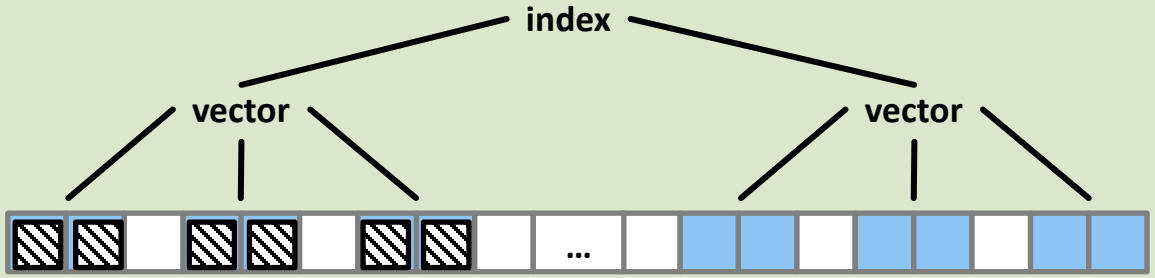
Dataloops

Index:

Vector:

Segment

Host Memory



Ross, Robert, et al. "Processing MPI datatypes outside MPI." European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting. Springer, Berlin, Heidelberg, 2009.

MPI Types Library on sPIN

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: ▢ }

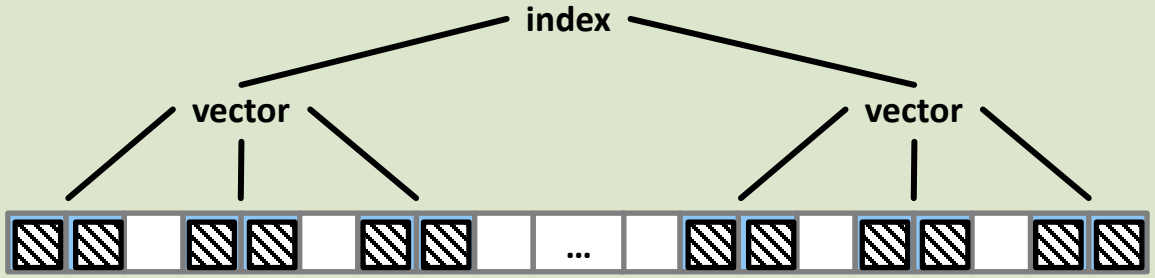
Dataloops

Index:

Vector:

Segment

Host Memory



Ross, Robert, et al. "Processing MPI datatypes outside MPI." European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting. Springer, Berlin, Heidelberg, 2009.

MPI Types Library on sPIN

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: ▢ }

Dataloops

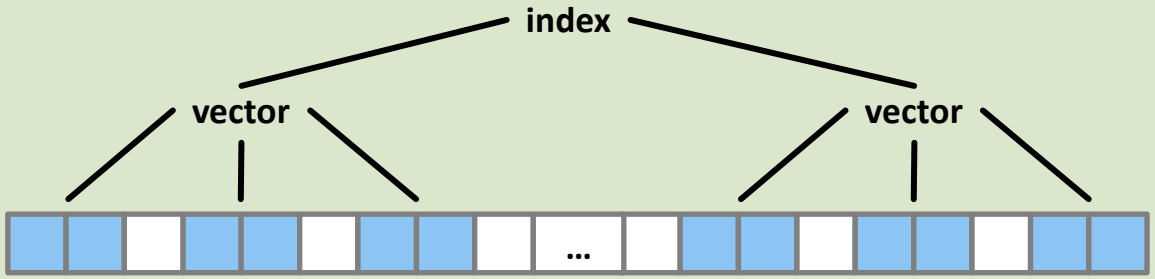
Index:
Vector:

Segment

Packet Scheduler

HPU 0 | **HPU 1** | **HPU 2** | **HPU 3**

Host Memory



MPI Types Library on sPIN

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

Dataloops

Index:

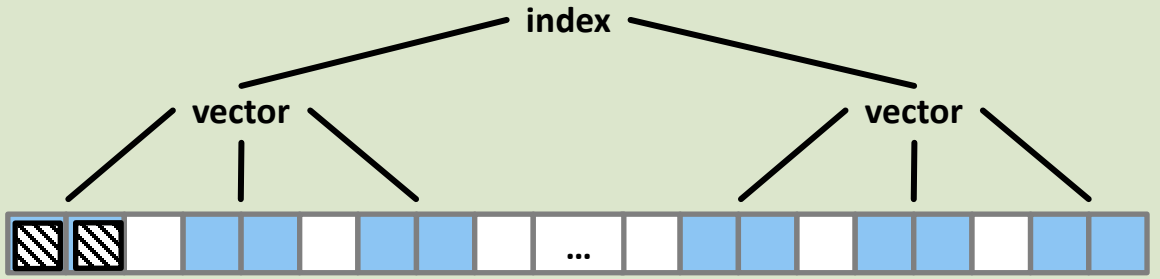
Vector:

Segment

Packet Scheduler

HPU 0 | **HPU 1** | **HPU 2** | **HPU 3**

Host Memory



MPI Types Library on sPIN

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

Dataloops

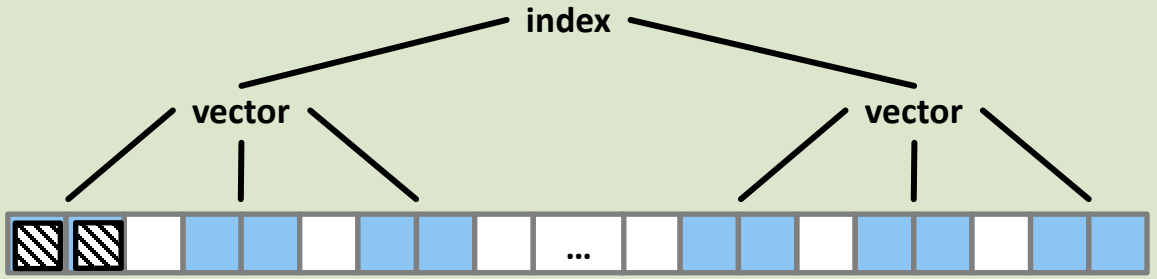
Index:
Vector:

Segment

Packet Scheduler

HPU 0 | HPU 1 | HPU 2 | HPU 3

Host Memory



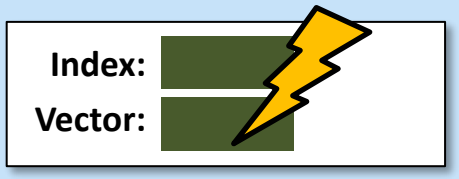
MPI Types Library on sPIN

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

Dataloops

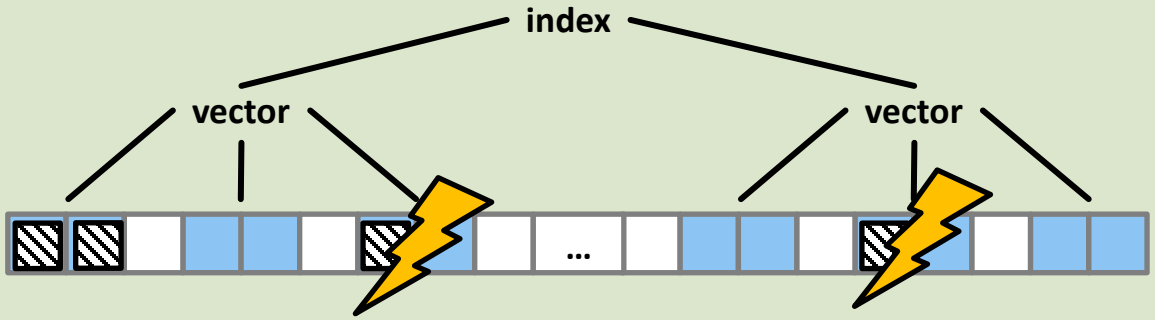


Segment

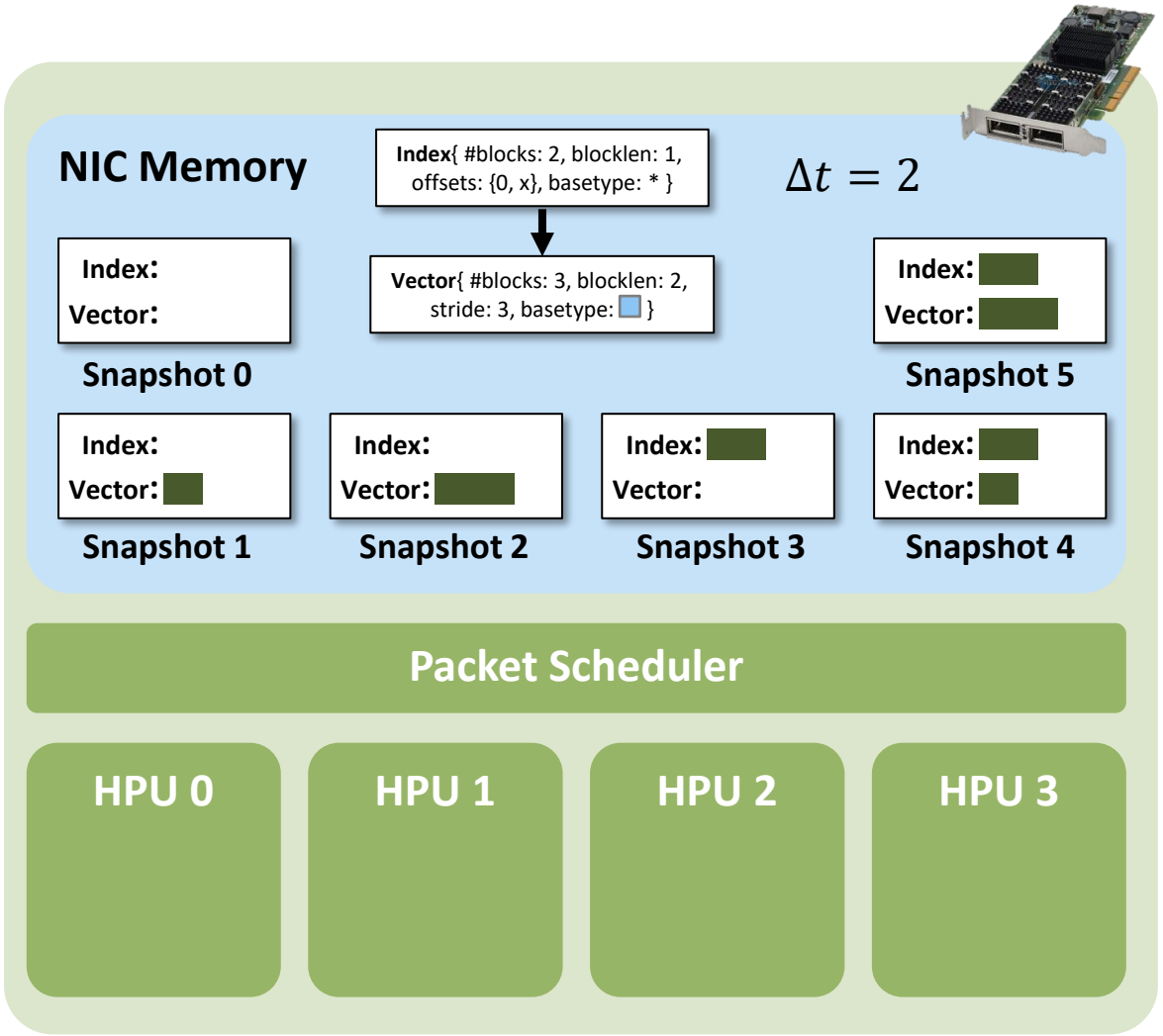
Packet Scheduler

HPU 0 | HPU 1 | HPU 2 | HPU 3

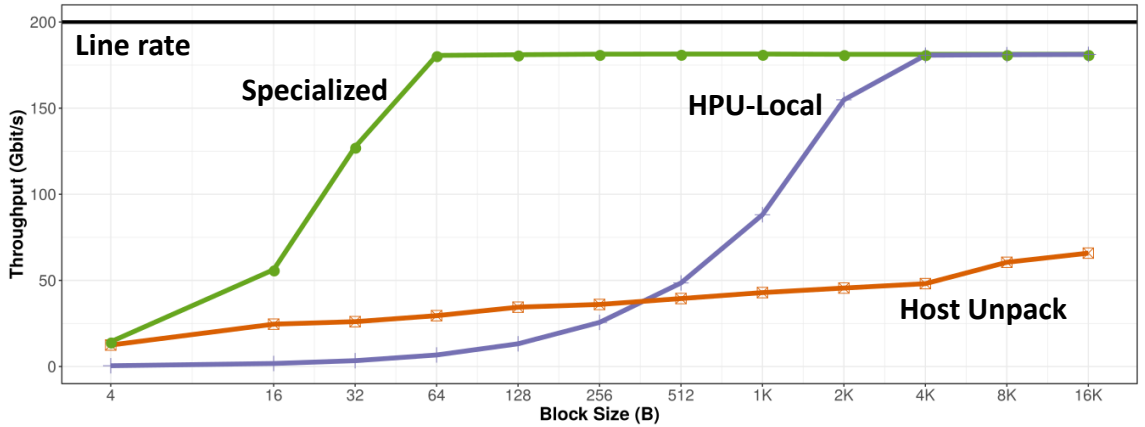
Host Memory



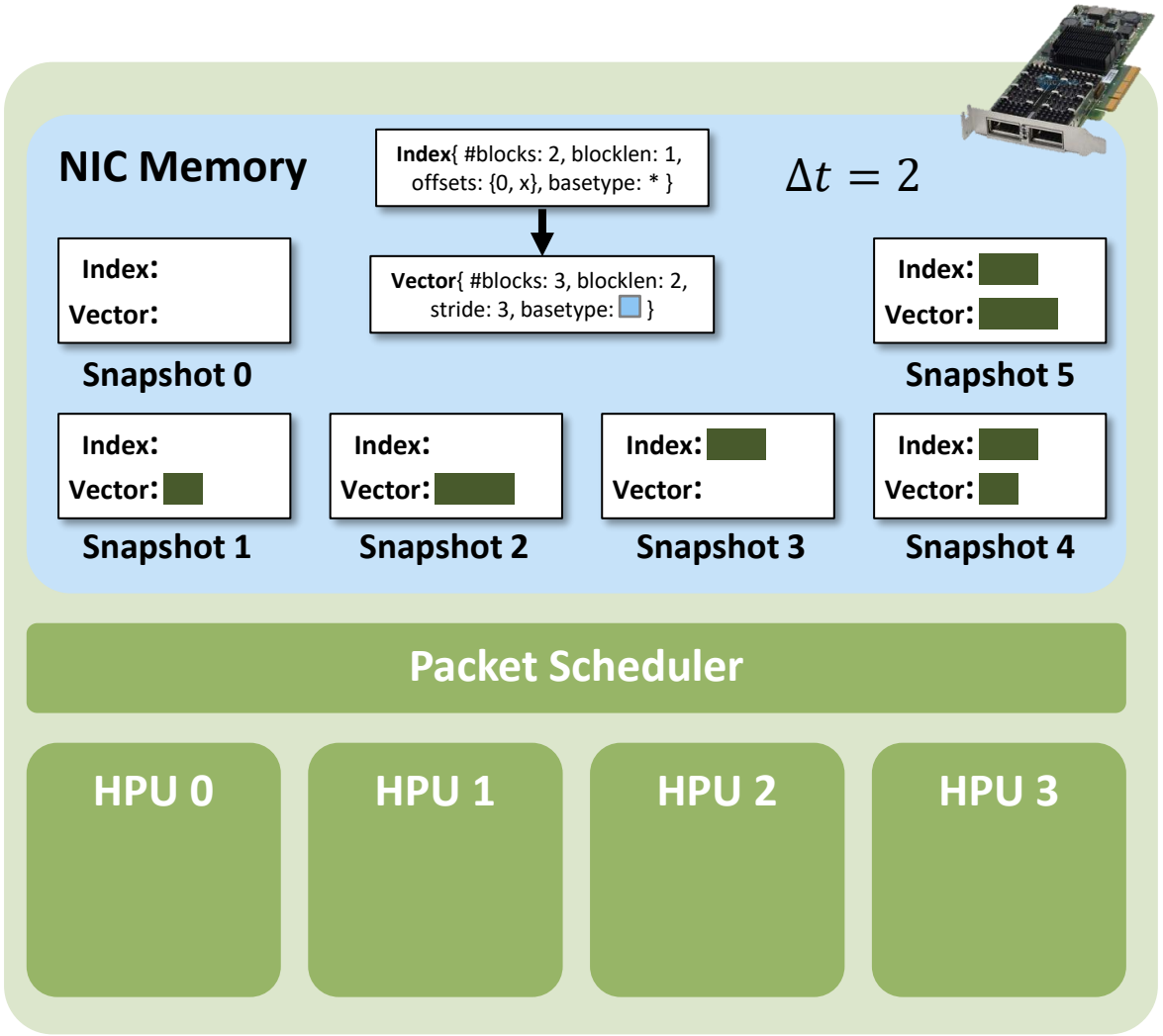
MPI Types Library on sPIN



HPU-Local: each HPU has its own state

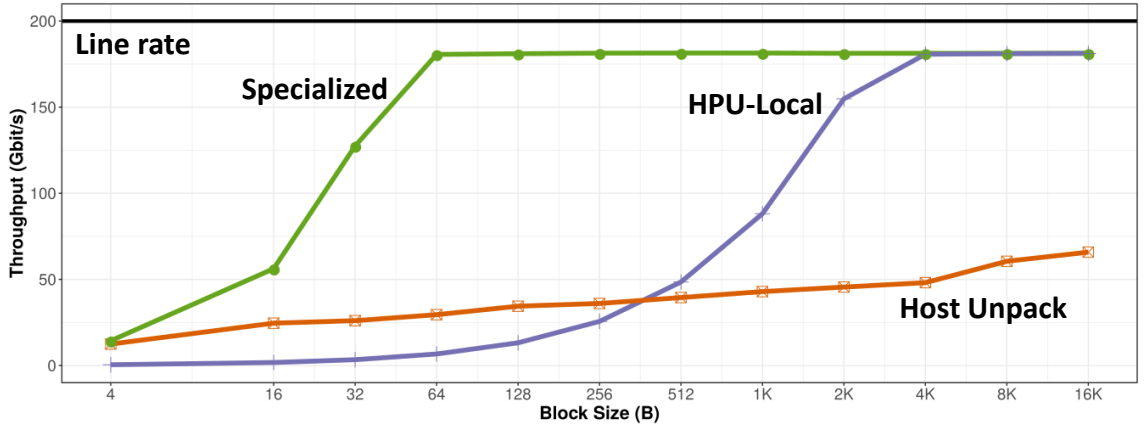


MPI Types Library on sPIN

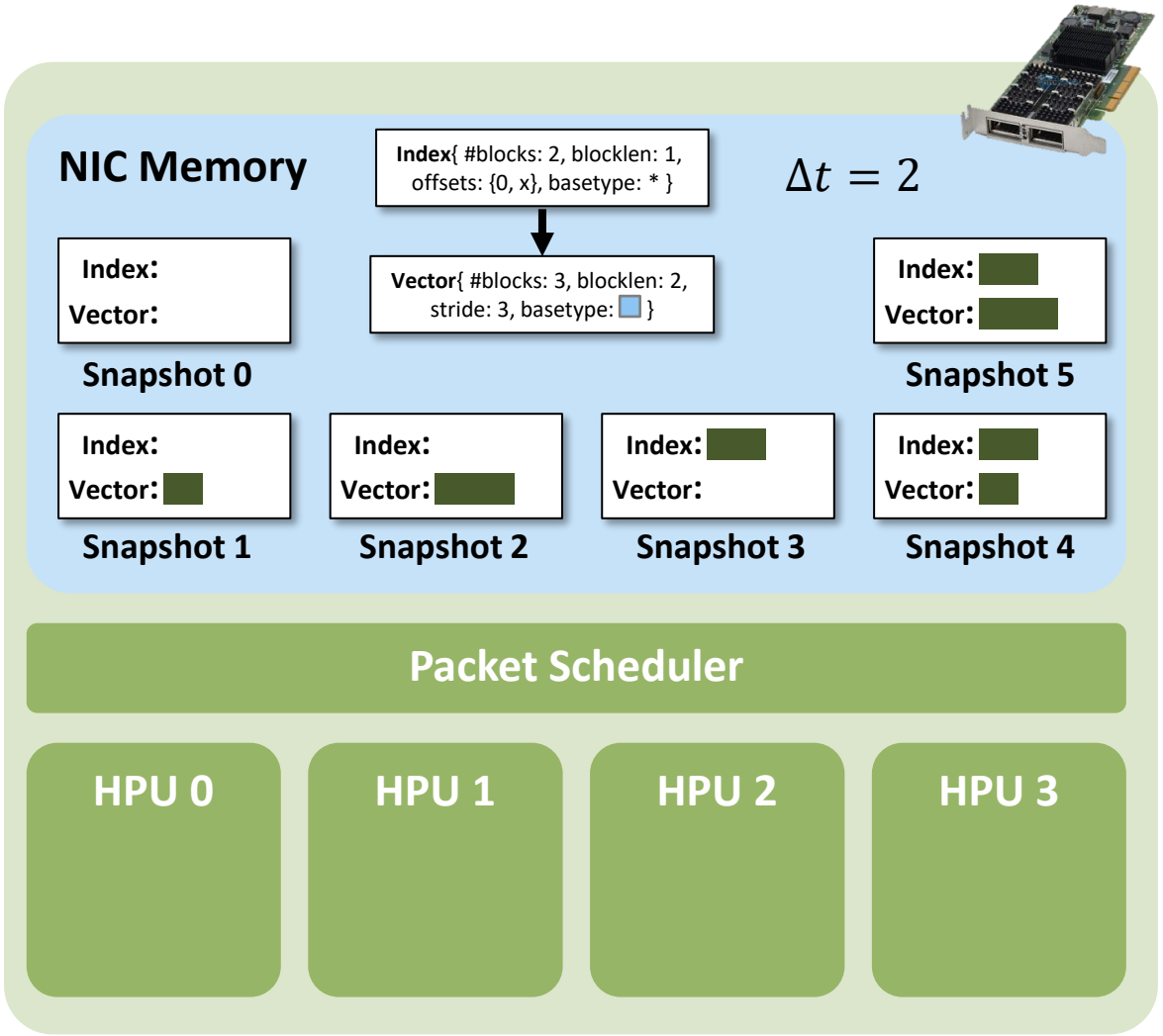


HPU-Local: each HPU has its own state

RO-checkpoints: pre-computed checkpoints shared by multiple HPUs (read-only)

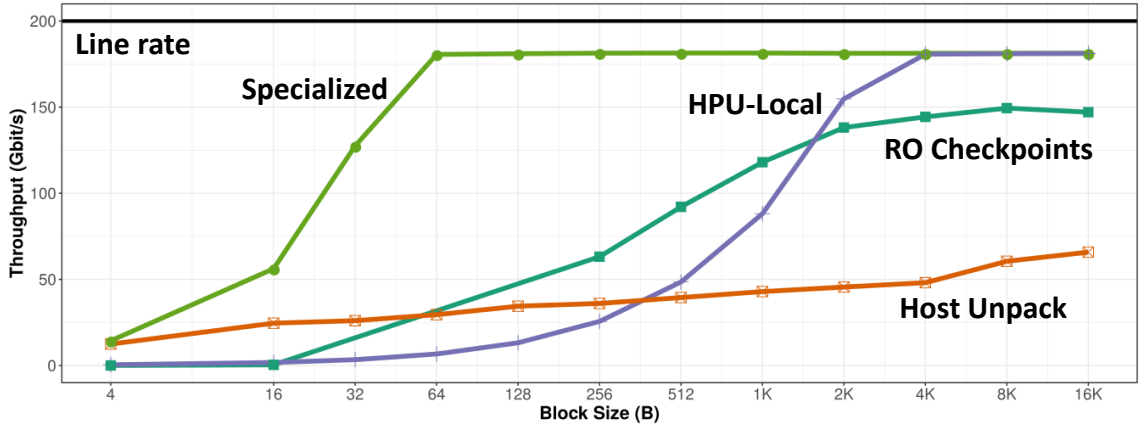


MPI Types Library on sPIN

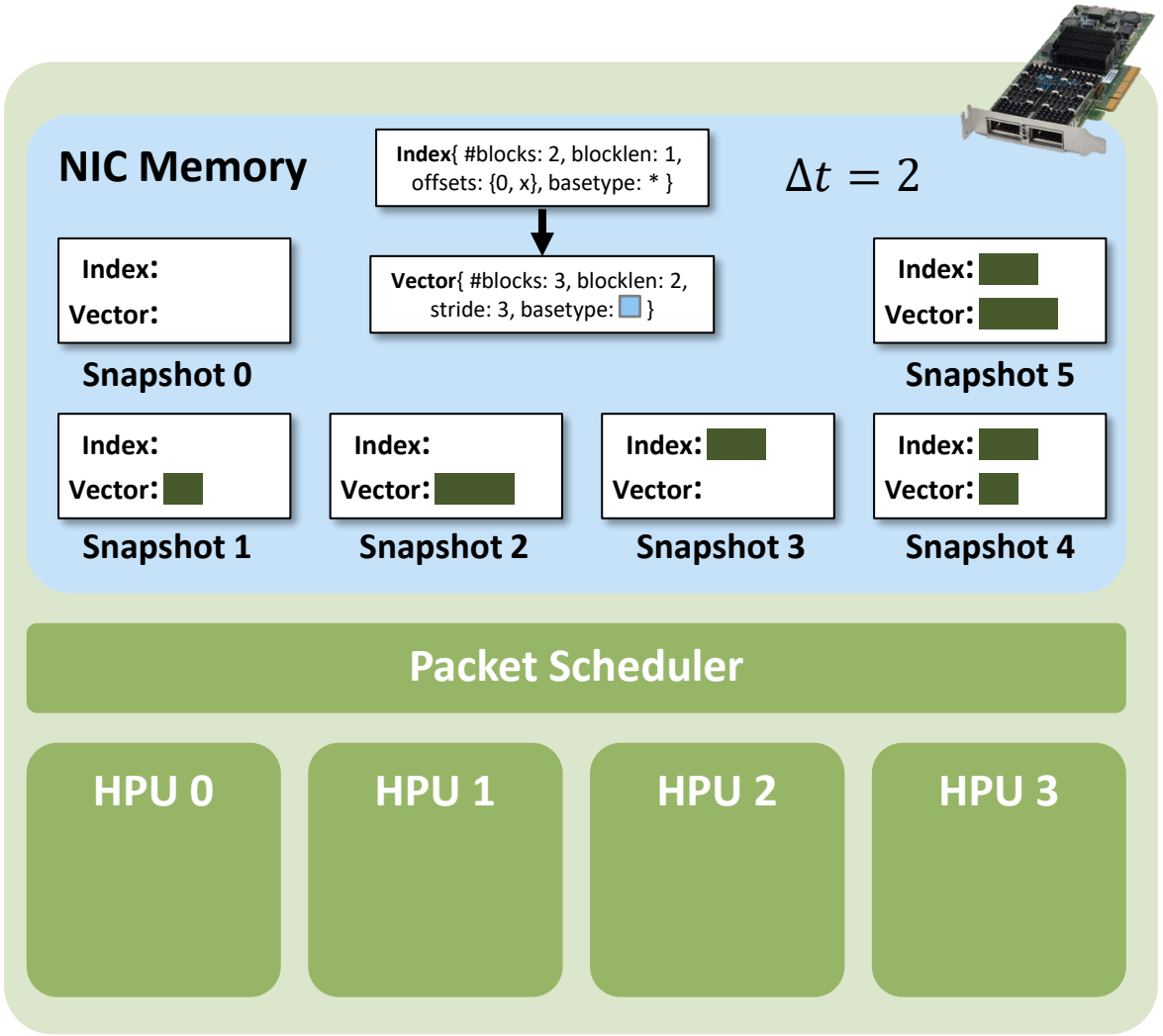


HPU-Local: each HPU has its own state

RO-checkpoints: pre-computed checkpoints shared by multiple HPUs (read-only)



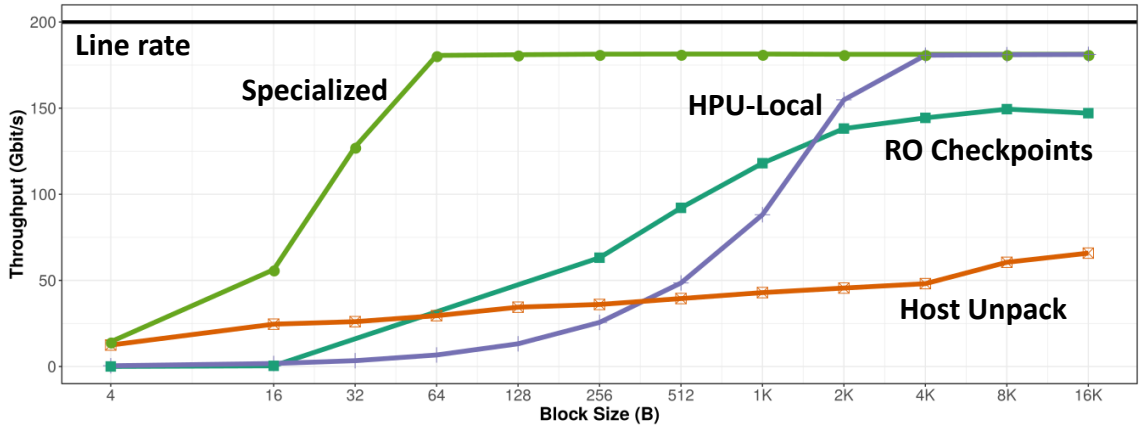
MPI Types Library on sPIN



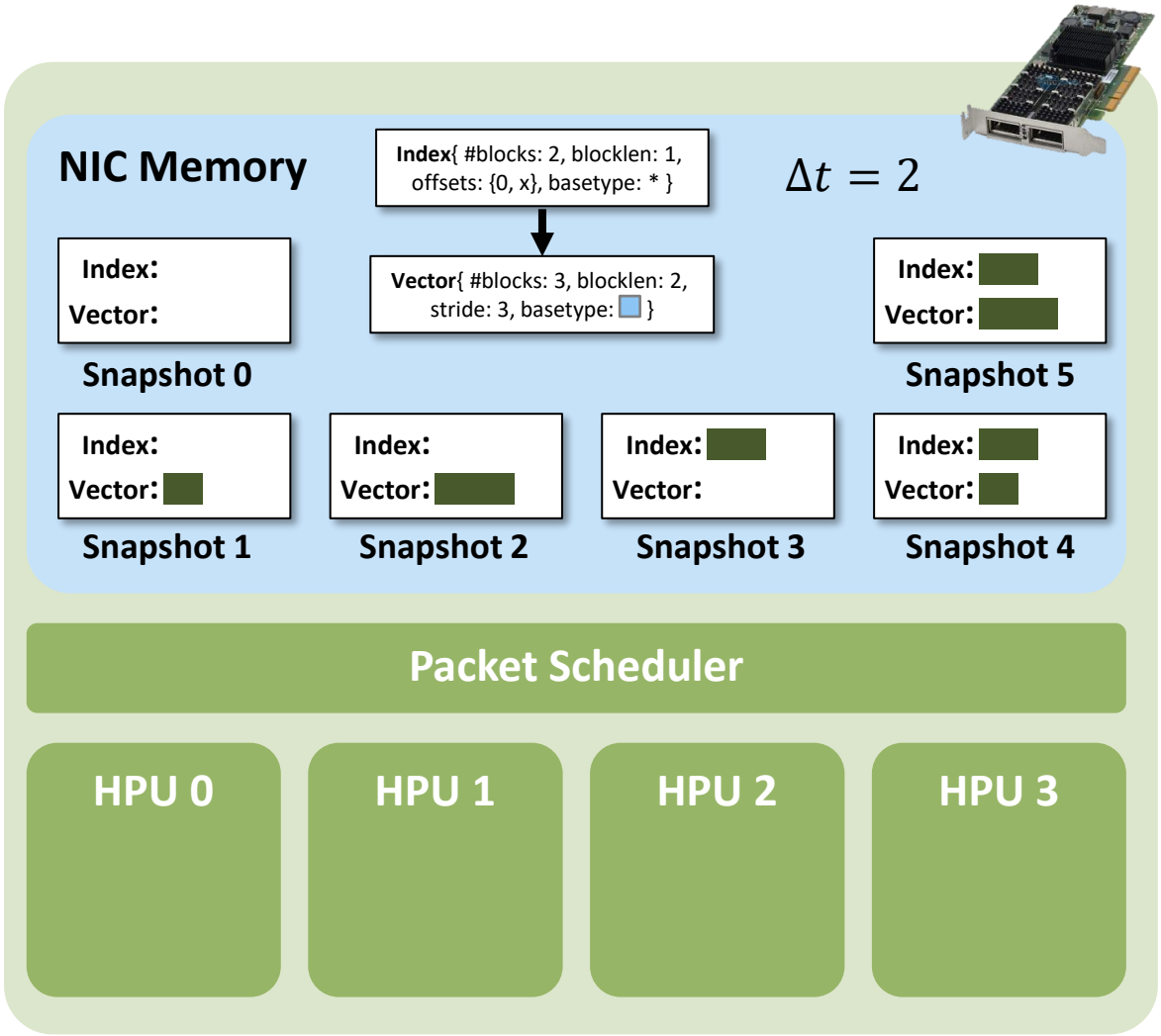
HPU-Local: each HPU has its own state

RO-checkpoints: pre-computed checkpoints shared by multiple HPUs (read-only)

RW-checkpoints: pre-computed checkpoints shared by multiple HPUs (read/write, fine-grain synchronization)



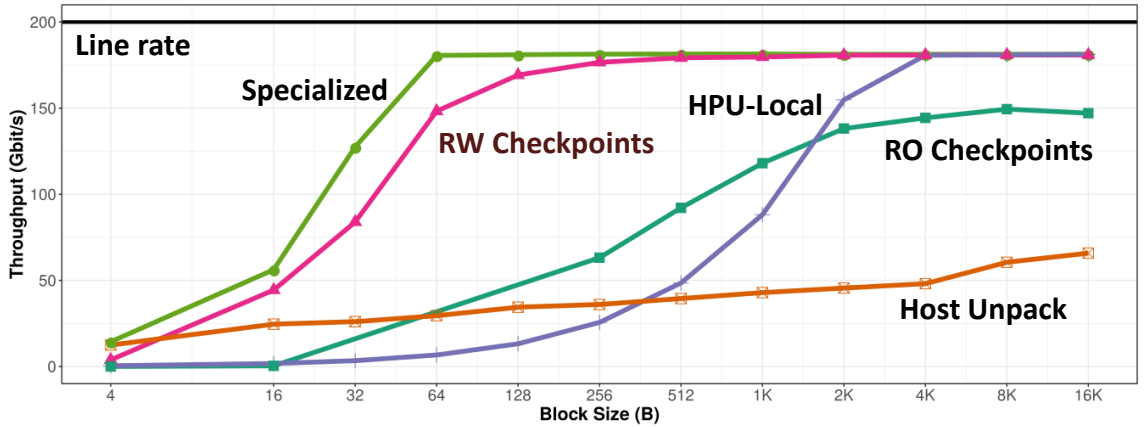
MPI Types Library on sPIN



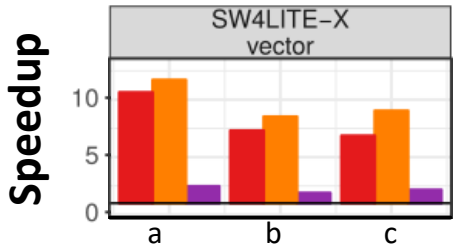
HPU-Local: each HPU has its own state

RO-checkpoints: pre-computed checkpoints shared by multiple HPUs (read-only)

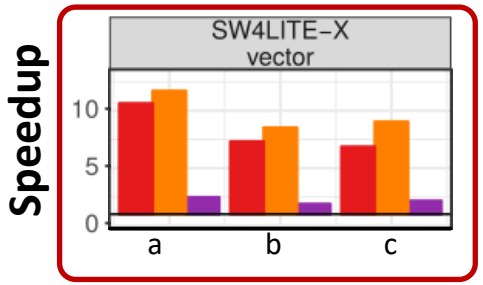
RW-checkpoints: pre-computed checkpoints shared by multiple HPUs (read/write, fine-grain synchronization)



Real Applications DDTs

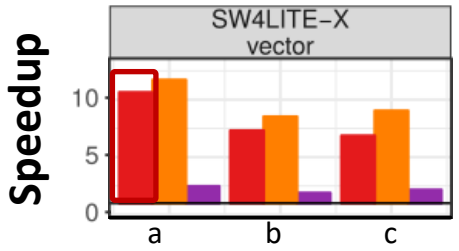


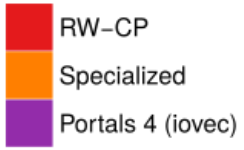
Real Applications DDTs



Real Applications DDTs

Real Applications DDTs



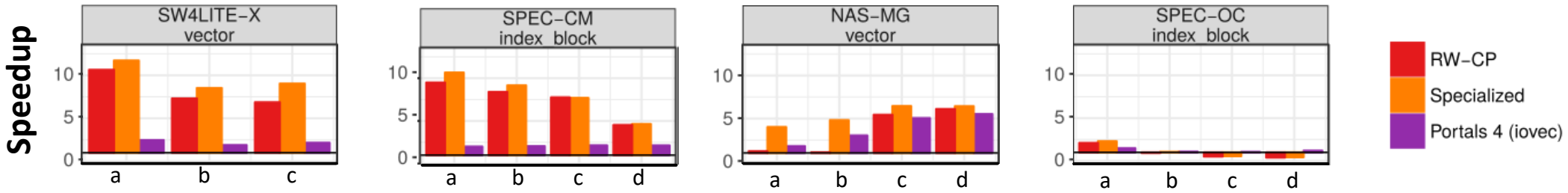


Real Applications DDTs

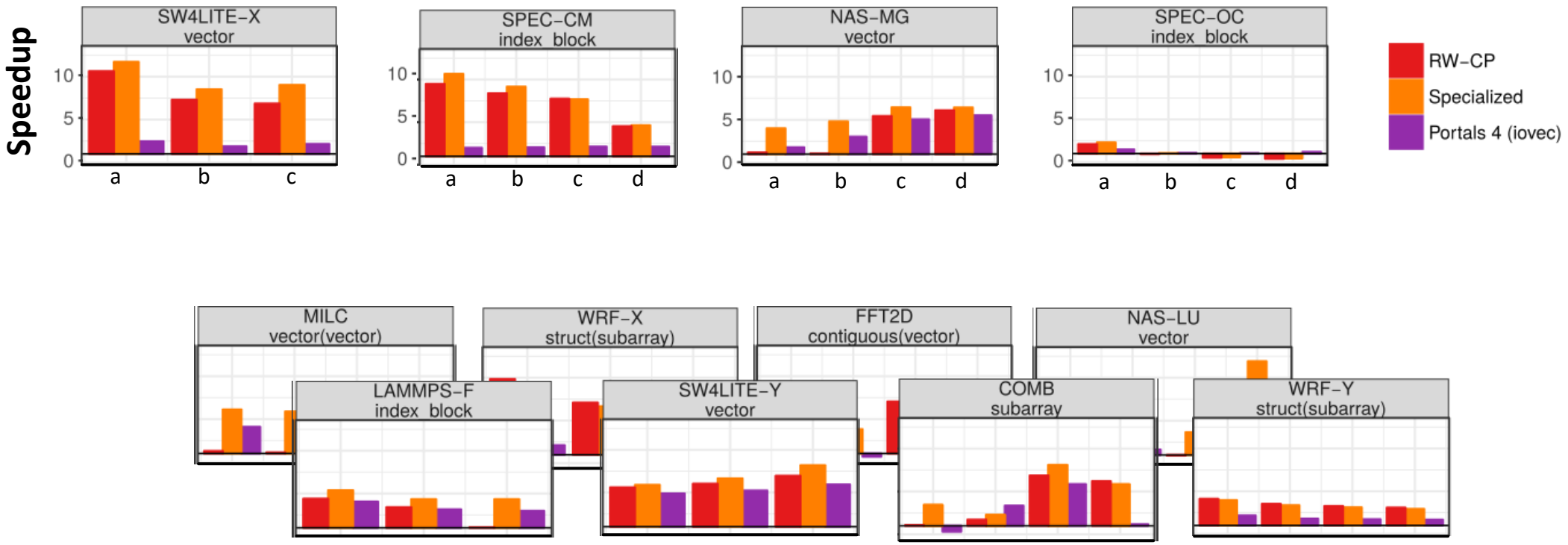
Real Applications DDTs

Real Applications DDTs

Real Applications DDTs

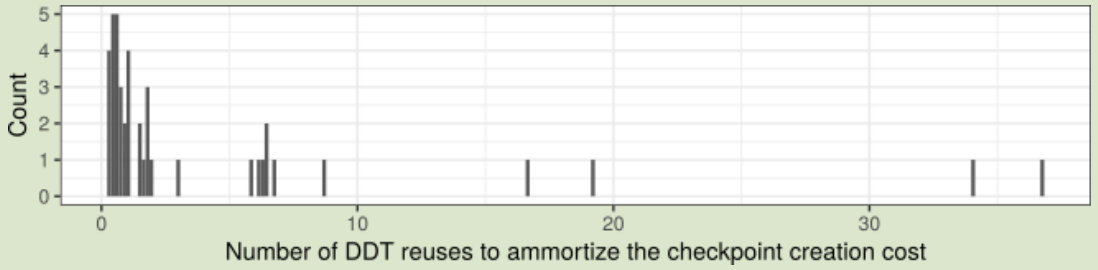


Real Applications DDTs



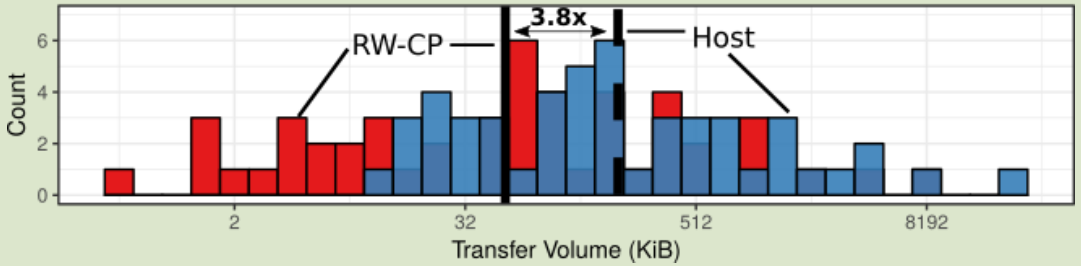
Real Applications DDTs

Checkpointing Overhead



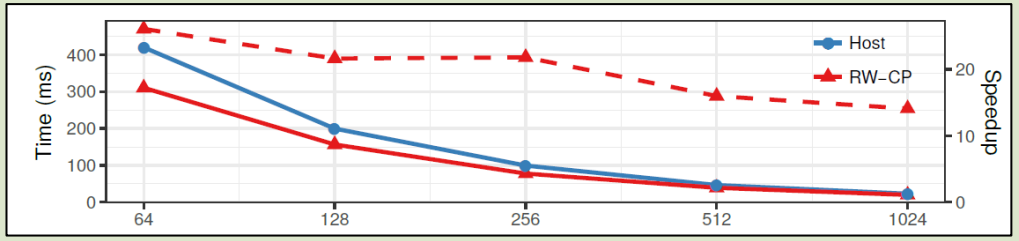
75% of the analyzed DDTs amortized after 4 reuses

Data Movement

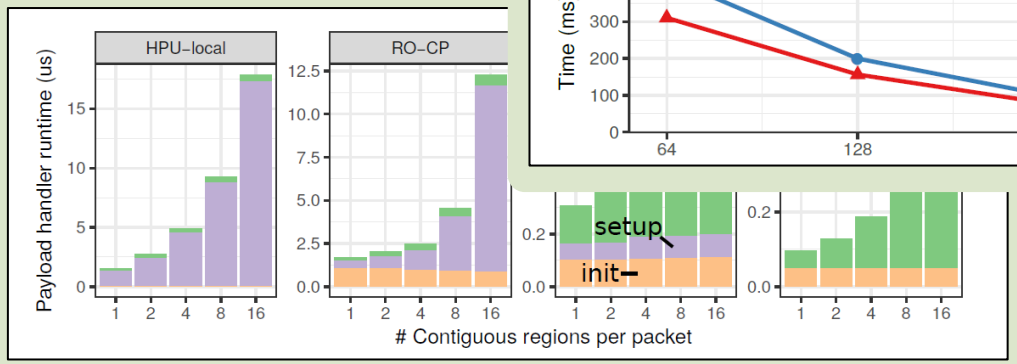


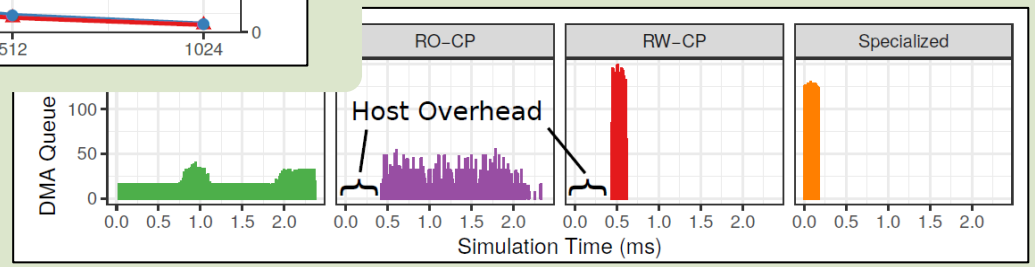
Up to 3.8x less moved data volume

Full app speedup (FFT2D)



Handler Analysis





Support for Non-Contiguous Transfers

Support for Non-Contiguous Transfers

ARMCI

SHMEM

CAF

UPC

Chapel

X10

Portals 4

MPI

Support for Non-Contiguous Transfers

ARMCI **CAF** **Chapel** **Portals 4**

SHMEM **UPC** **X10** **MPI**

I/O Vectors Strided transfers Compiler-Assisted Aggregation

Support for multiple strides (e.g., 3D faces)

Support for Non-Contiguous Transfers

ARMCI **CAF** **Chapel** **Portals 4**

SHMEM **UPC** **X10**

MPI

I/O Vectors

Strided transfers

Compiler-Assisted Aggregation

Support for multiple strides (e.g., 3D faces)

Derived Datatypes

Support for Non-Contiguous Transfers

ARMCI **CAF** **Chapel** **Portals 4**

SHMEM **UPC** **X10**

I/O Vectors Strided transfers Compiler-Assisted Aggregation

Support for multiple strides (e.g., 3D faces)

MPI

Derived Datatypes

Support for Non-Contiguous Transfers

ARMCI **CAF** **Chapel** **Portals 4**

SHMEM **UPC** **X10**

I/O Vectors Strided transfers Compiler-Assisted Aggregation

Support for multiple strides (e.g., 3D faces)

MPI

Derived Datatypes

Support for Non-Contiguous Transfers

ARMCI **CAF** **Chapel** **Portals 4**

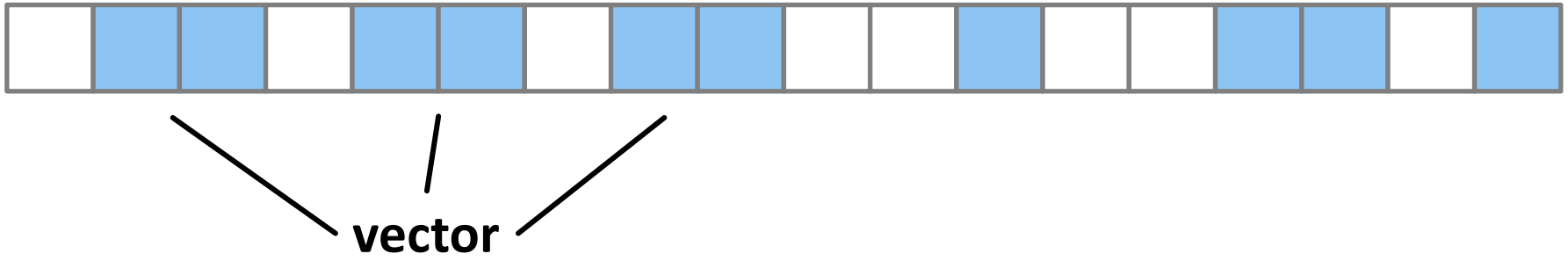
SHMEM **UPC** **X10**

I/O Vectors Strided transfers Compiler-Assisted Aggregation

Support for multiple strides (e.g., 3D faces)

MPI

Derived Datatypes



Support for Non-Contiguous Transfers

ARMCI **CAF** **Chapel** **Portals 4**

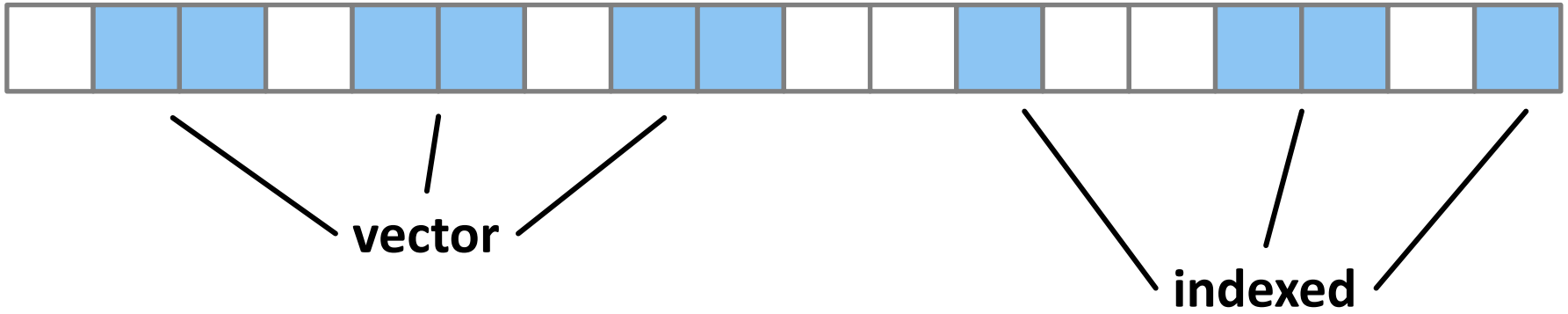
SHMEM **UPC** **X10**

I/O Vectors Strided transfers Compiler-Assisted Aggregation

Support for multiple strides (e.g., 3D faces)

MPI

Derived Datatypes



Support for Non-Contiguous Transfers

ARMCI **CAF** **Chapel** **Portals 4**

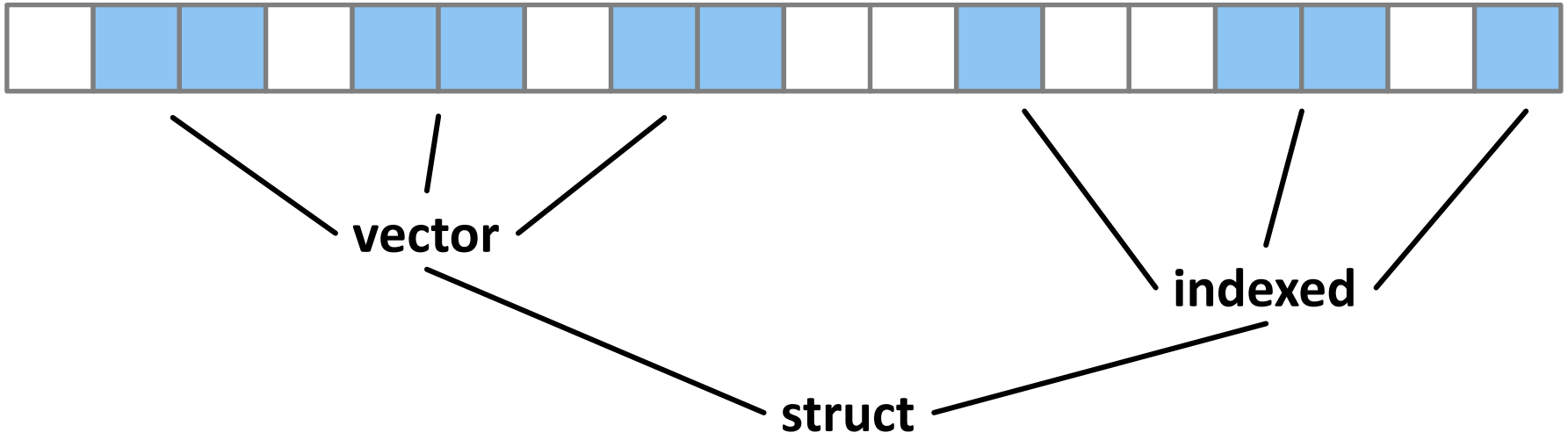
SHMEM **UPC** **X10**

I/O Vectors Strided transfers Compiler-Assisted Aggregation

Support for multiple strides (e.g., 3D faces)

MPI

Derived Datatypes



MPI Types Library on sPIN: HPU-Local

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

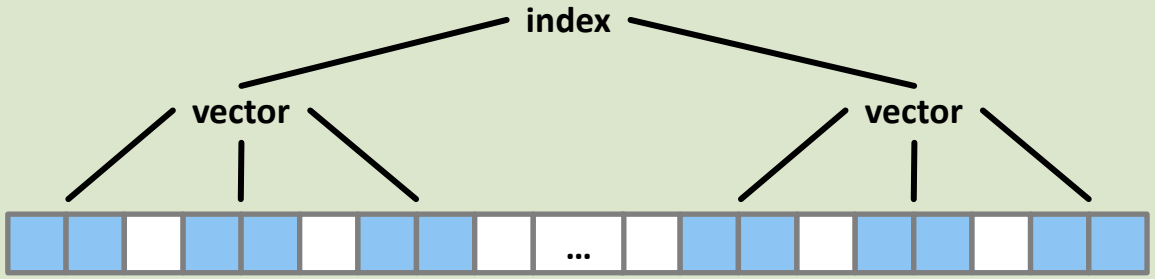
Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

Segment 0	Segment 1	Segment 2	Segment 3
Index: Vector:	Index: Vector:	Index: Vector:	Index: Vector:

Packet Scheduler

HPU 0 | HPU 1 | HPU 2 | HPU 3

Host Memory



MPI Types Library on sPIN: HPU-Local

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

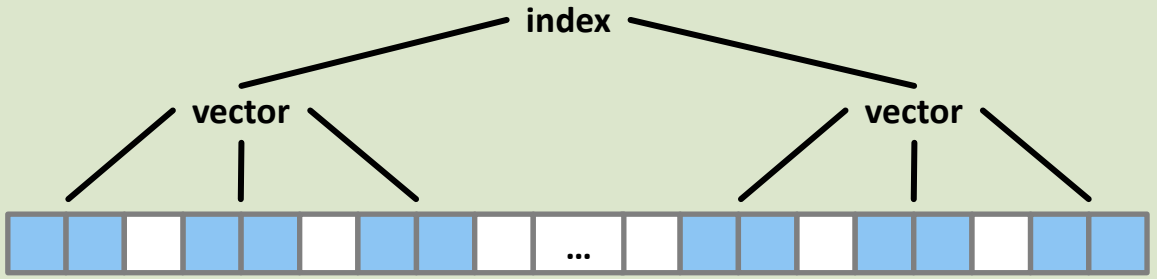
Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

Segment 0	Segment 1	Segment 2	Segment 3
Index: Vector:	Index: Vector:	Index: Vector:	Index: Vector:

Packet Scheduler

HPU 0 | HPU 1 | HPU 2 | HPU 3

Host Memory



MPI Types Library on sPIN: HPU-Local

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

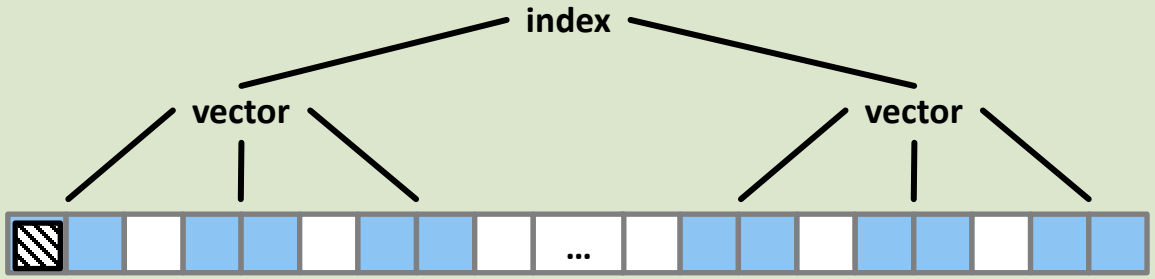
Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

Segment 0	Segment 1	Segment 2	Segment 3
Index: Vector:	Index: Vector:	Index: Vector:	Index: Vector:

Packet Scheduler

HPU 0 | HPU 1 | HPU 2 | HPU 3

Host Memory



MPI Types Library on sPIN: HPU-Local

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

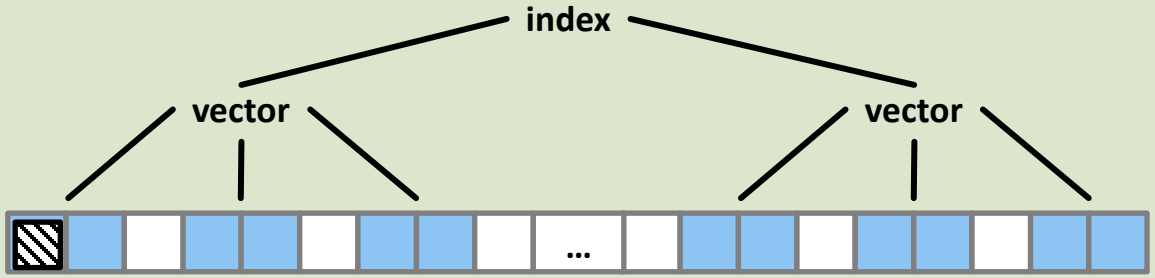
Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

Segment 0	Segment 1	Segment 2	Segment 3
Index: Vector:	Index: Vector:	Index: Vector:	Index: Vector:

Packet Scheduler

HPU 0	HPU 1	HPU 2	HPU 3

Host Memory



MPI Types Library on sPIN: HPU-Local

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

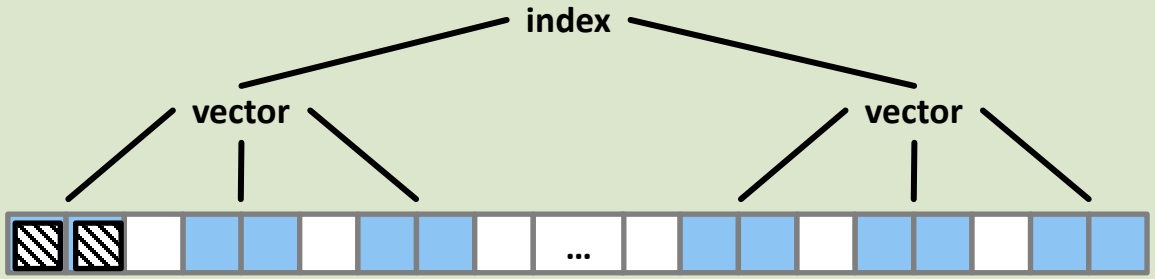
Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

Index: Vector:	Index: Vector:	Index: Vector:	Index: Vector:
Segment 0	Segment 1	Segment 2	Segment 3

Packet Scheduler

HPU 0 | HPU 1 | HPU 2 | HPU 3

Host Memory



MPI Types Library on sPIN: HPU-Local

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

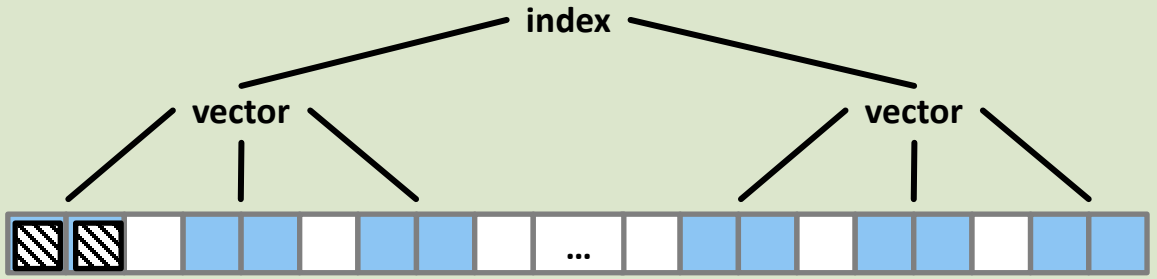
Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

Segment 0	Segment 1	Segment 2	Segment 3
Index: Vector:	Index: Vector:	Index: Vector:	Index: Vector:

Packet Scheduler

HPU 0 | HPU 1 | HPU 2 | HPU 3

Host Memory



MPI Types Library on sPIN: HPU-Local

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

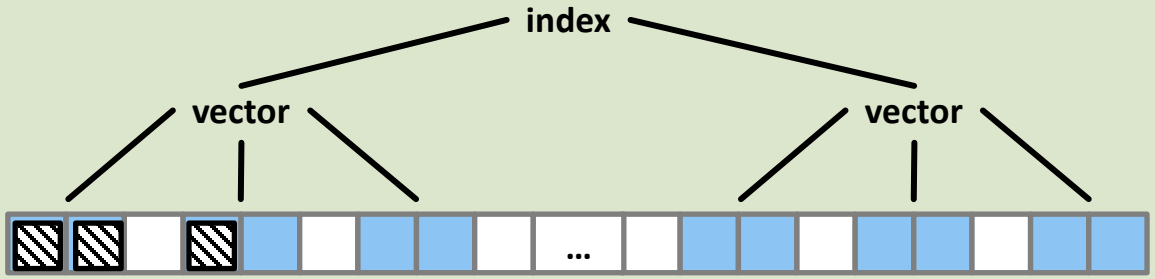
Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

Index: Vector:	Index: Vector:	Index: Vector:	Index: Vector:
Segment 0	Segment 1	Segment 2	Segment 3

Packet Scheduler

HPU 0 | HPU 1 | HPU 2 | HPU 3

Host Memory



MPI Types Library on sPIN: HPU-Local

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

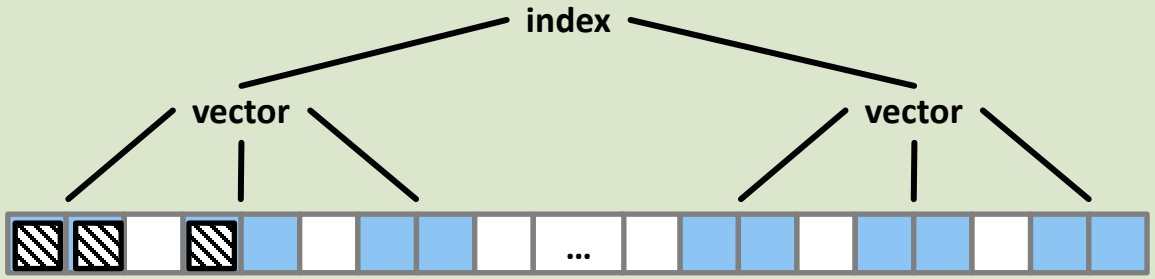
Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

Segment 0	Segment 1	Segment 2	Segment 3
Index: Vector:	Index: Vector:	Index: Vector:	Index: Vector:

Packet Scheduler

HPU 0 | HPU 1 | HPU 2 | HPU 3

Host Memory



MPI Types Library on sPIN: HPU-Local

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

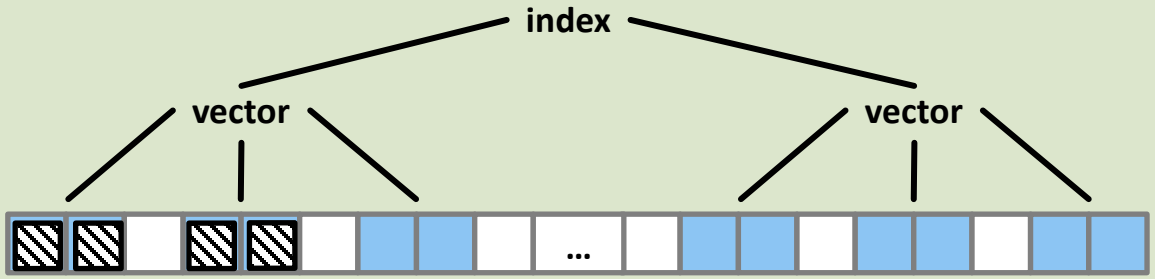
Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

Index: Vector:	Index: Vector:	Index: Vector:	Index: Vector:
Segment 0	Segment 1	Segment 2	Segment 3

Packet Scheduler

HPU 0 | HPU 1 | HPU 2 | HPU 3

Host Memory



MPI Types Library on sPIN: HPU-Local

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

Index: Vector: Segment 0

Index: Vector: Segment 1

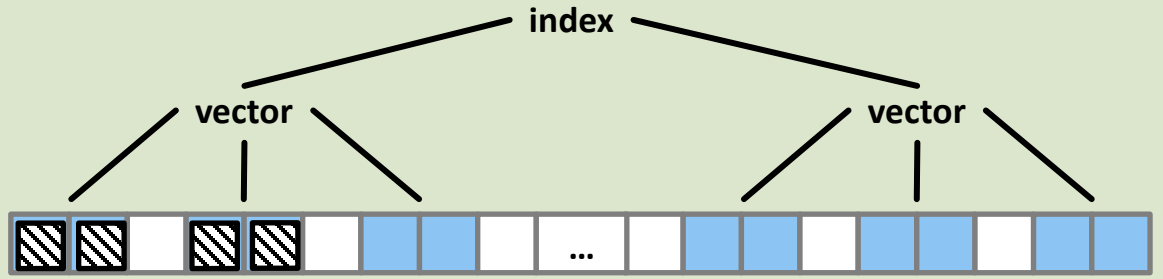
Index: Vector: Segment 2

Index: Vector: Segment 3

Packet Scheduler

HPU 0 | HPU 1 | HPU 2 | HPU 3

Host Memory



MPI Types Library on sPIN: HPU-Local

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

Index: Vector: Segment 0

Index: Vector: Segment 1

Index: Vector: Segment 2

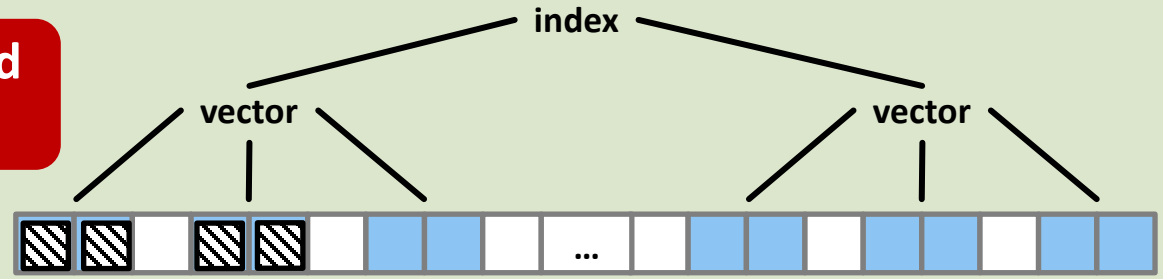
Index: Vector: Segment 3

Fast-forward overhead

Packet Scheduler

HPU 0 | HPU 1 | HPU 2 | HPU 3

Host Memory



MPI Types Library on sPIN: HPU-Local

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

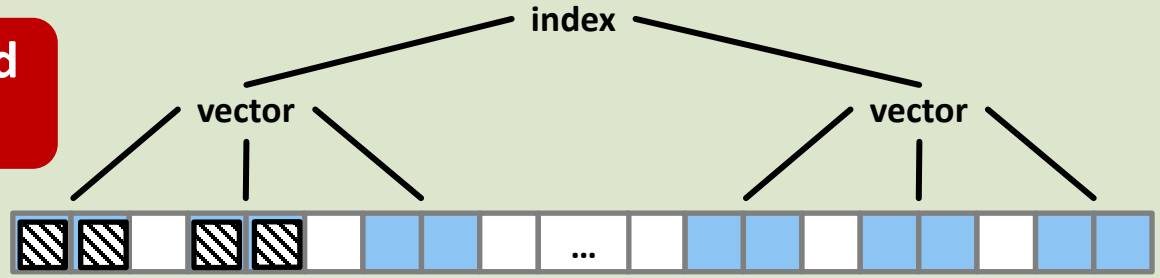
Fast-forward overhead

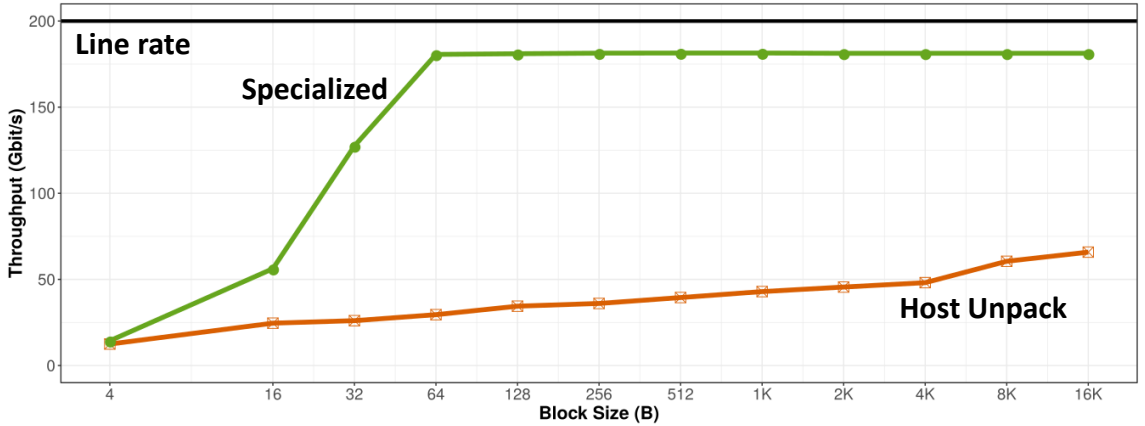
Segment 0	Segment 1	Segment 2	Segment 3
Index: Vector:	Index: Vector:	Index: Vector:	Index: Vector:

Packet Scheduler

HPU 0 | HPU 1 | HPU 2 | HPU 3

Host Memory





MPI Types Library on sPIN: HPU-Local

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

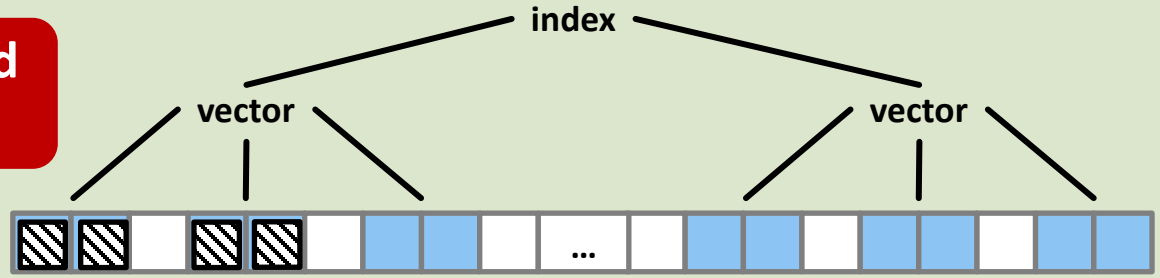
Fast-forward overhead

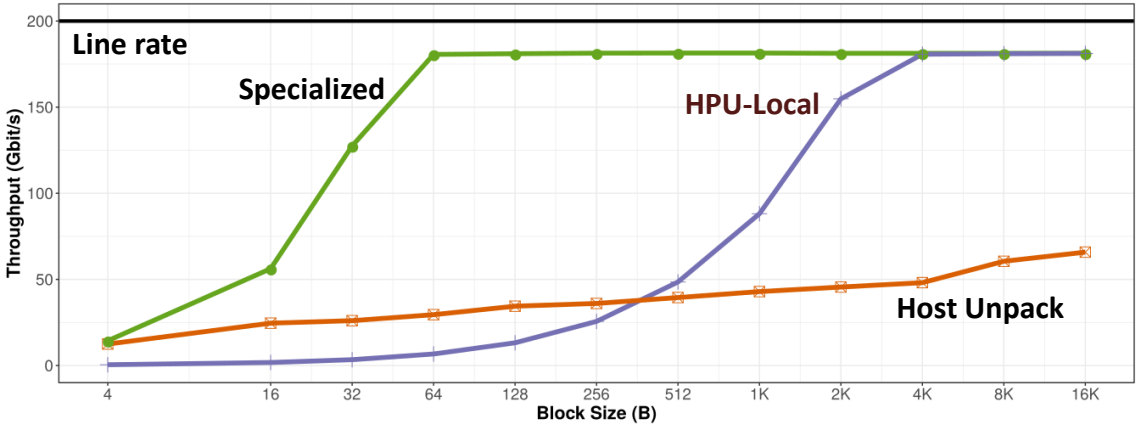
Segment 0	Segment 1	Segment 2	Segment 3
Index: Vector:	Index: Vector:	Index: Vector:	Index: Vector:

Packet Scheduler

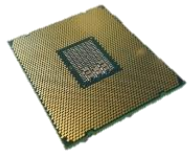
HPU 0 | HPU 1 | HPU 2 | HPU 3

Host Memory





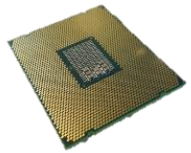
MPI Types Library on sPIN: Checkpoints



Snapshot the state on the host to bootstrap the handlers

Segment

MPI Types Library on sPIN: Checkpoints

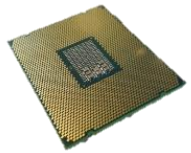


Snapshot the state on the host to bootstrap the handlers

Segment

$$\Delta t = 2$$

MPI Types Library on sPIN: Checkpoints



Snapshot the state on the host to bootstrap the handlers

Index:
Vector:

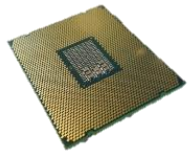
Segment

$\Delta t = 2$

Index:
Vector:

Checkpoint 0

MPI Types Library on sPIN: Checkpoints



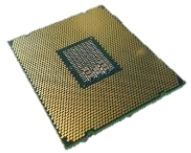
Snapshot the state on the host to bootstrap the handlers

Segment

$\Delta t = 2$

Checkpoint 0

MPI Types Library on sPIN: Checkpoints



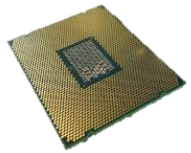
Snapshot the state on the host to bootstrap the handlers

Segment

$\Delta t = 2$

Checkpoint 0

MPI Types Library on sPIN: Checkpoints



Snapshot the state on the host to bootstrap the handlers

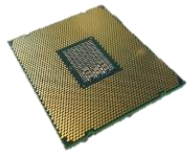
Segment

$\Delta t = 2$

Checkpoint 0

Checkpoint 1

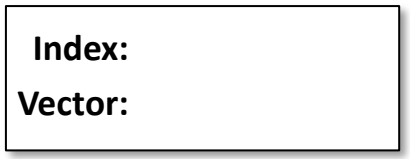
MPI Types Library on sPIN: Checkpoints



Snapshot the state on the host to bootstrap the handlers

Segment

$\Delta t = 2$



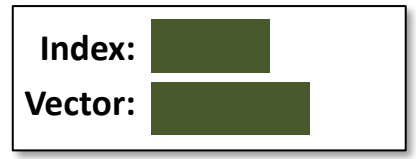
Checkpoint 0

Checkpoint 1

Checkpoint 2

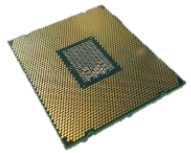
Checkpoint 3

Checkpoint 4



Checkponit 5

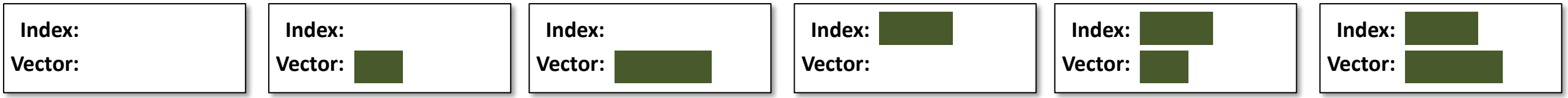
MPI Types Library on sPIN: Checkpoints



Snapshot the state on the host to bootstrap the handlers

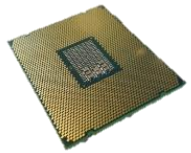
Segment

$\Delta t = 2$



Checkpoint 0 | Checkpoint 1 | Checkpoint 2 | Checkpoint 3 | Checkpoint 4 | Checkponit 5

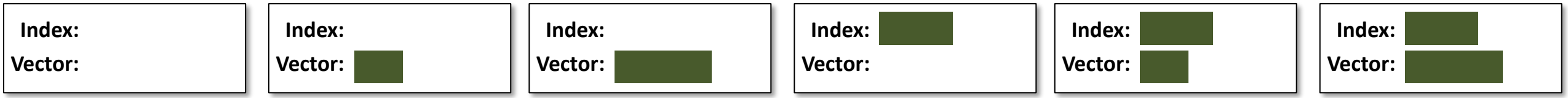
MPI Types Library on sPIN: Checkpoints



Snapshot the state on the host to bootstrap the handlers

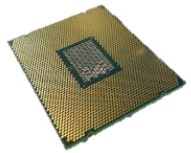
Segment

$\Delta t = 2$



Checkpoint 0 | Checkpoint 1 | Checkpoint 2 | Checkpoint 3 | Checkpoint 4 | Checkponit 5

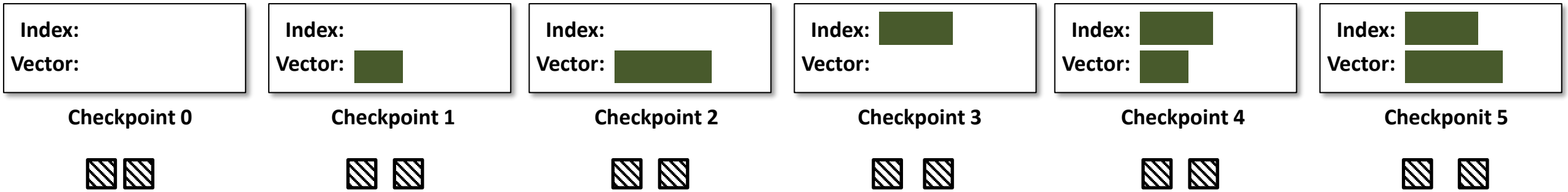
MPI Types Library on sPIN: Checkpoints



Snapshot the state on the host to bootstrap the handlers

Segment

$\Delta t = 2$



MPI Types Library on sPIN: Checkpoints

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

$\Delta t = 2$

Index:
Vector:
Snapshot 0

Index:
Vector:
Snapshot 5

Index:
Vector:
Snapshot 1

Index:
Vector:
Snapshot 2

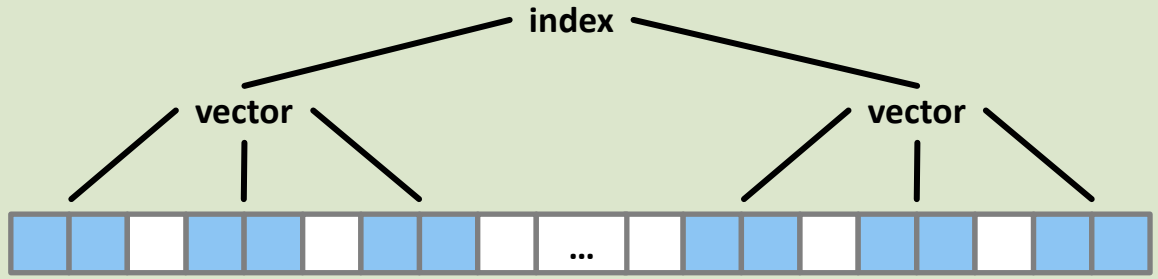
Index:
Vector:
Snapshot 3

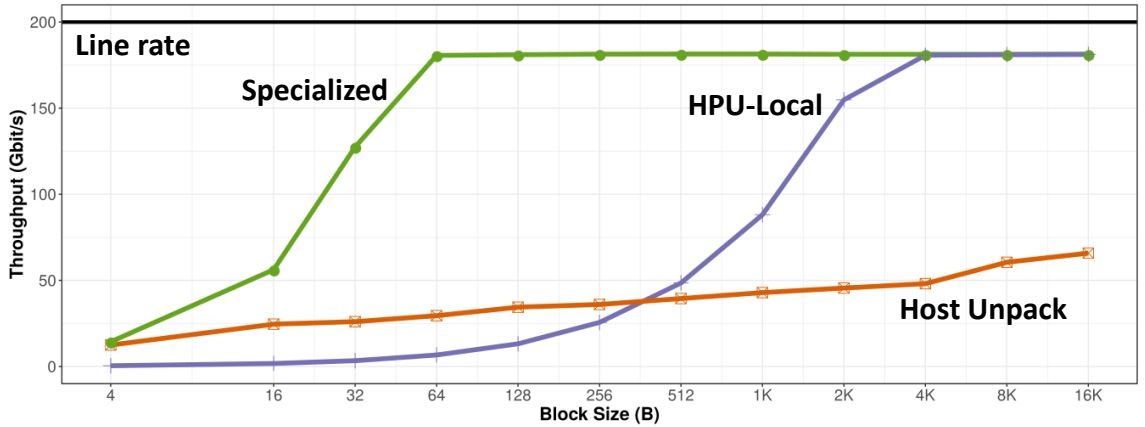
Index:
Vector:
Snapshot 4

Packet Scheduler

HPU 0 | HPU 1 | HPU 2 | HPU 3

Host Memory





MPI Types Library on sPIN: Checkpoints

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

$\Delta t = 2$

Index:
Vector:
Snapshot 0

Index:
Vector:
Snapshot 5

Index:
Vector:
Snapshot 1

Index:
Vector:
Snapshot 2

Index:
Vector:
Snapshot 3

Index:
Vector:
Snapshot 4

Packet Scheduler

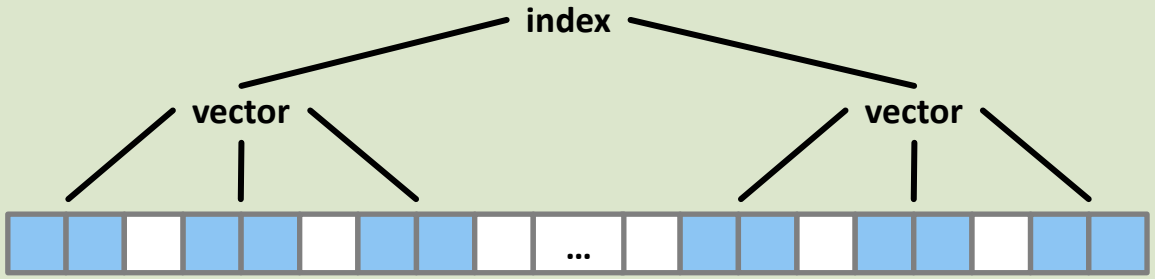
HPU 0

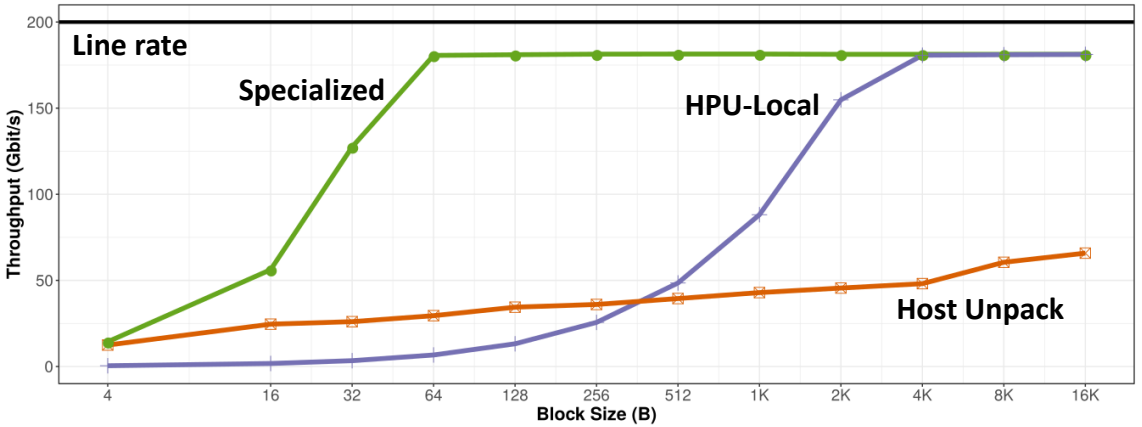
HPU 1

HPU 2

HPU 3

Host Memory





MPI Types Library on sPIN: Checkpoints

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

$\Delta t = 2$

Index: Vector: Snapshot 0

Index: Vector: Snapshot 5

Index: Vector: Snapshot 1

Index: Vector: Snapshot 2

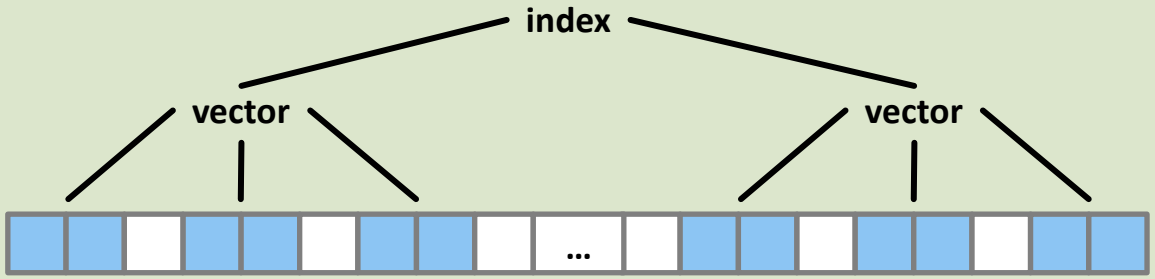
Index: Vector: Snapshot 3

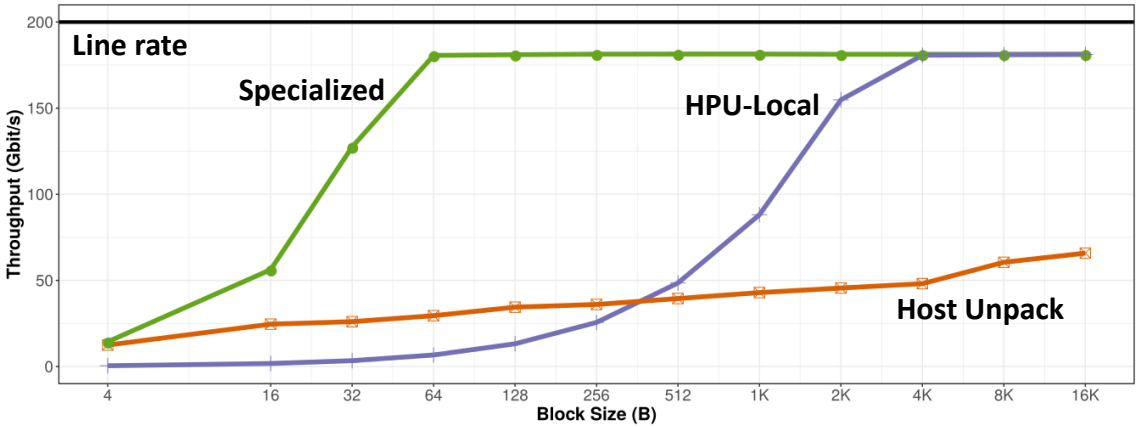
Index: Vector: Snapshot 4

Packet Scheduler

HPU 0 | HPU 1 | HPU 2 | HPU 3

Host Memory





MPI Types Library on sPIN: Checkpoints

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

$\Delta t = 2$

Index:
Vector:
Snapshot 0

Index:
Vector:
Snapshot 5

Index:
Vector:
Snapshot 1

Index:
Vector:
Snapshot 2

Index:
Vector:
Snapshot 3

Index:
Vector:
Snapshot 4

Packet Scheduler

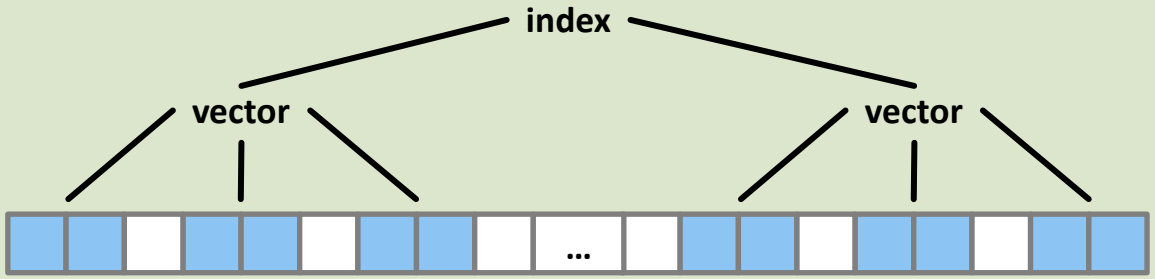
HPU 0

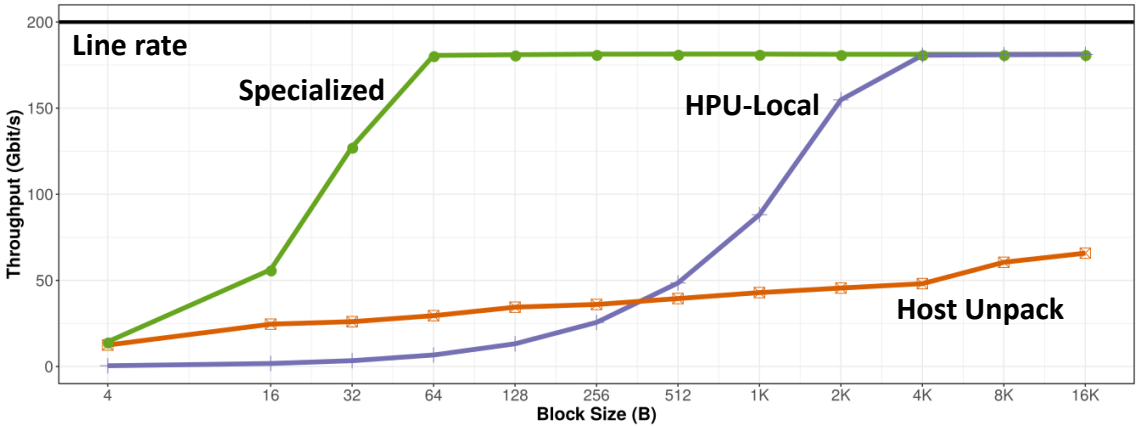
HPU 1

HPU 2

HPU 3

Host Memory





MPI Types Library on sPIN: Checkpoints

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

$\Delta t = 2$

Index:
Vector:
Snapshot 0

Index:
Vector:
Snapshot 5

Index:
Vector:
Snapshot 1

Index:
Vector:
Snapshot 2

Index:
Vector:
Snapshot 3

Index:
Vector:
Snapshot 4

Packet Scheduler

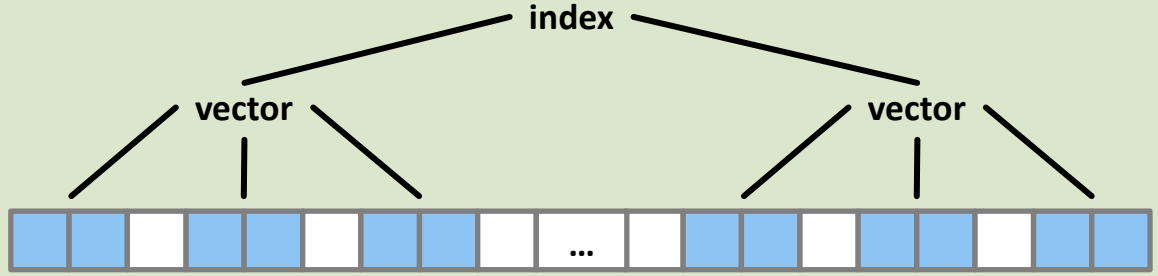
HPU 0

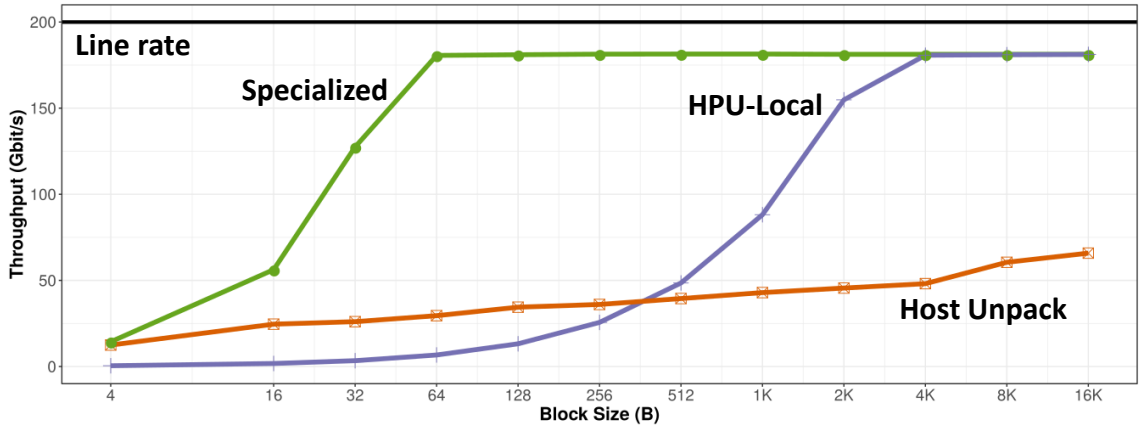
HPU 1

HPU 2

HPU 3

Host Memory





MPI Types Library on sPIN: Read-Only Checkpoints

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

$\Delta t = 2$

Snapshot 0: Index: Vector:

Snapshot 1: Index: Vector:

Snapshot 2: Index: Vector:

Snapshot 3: Index: Vector:

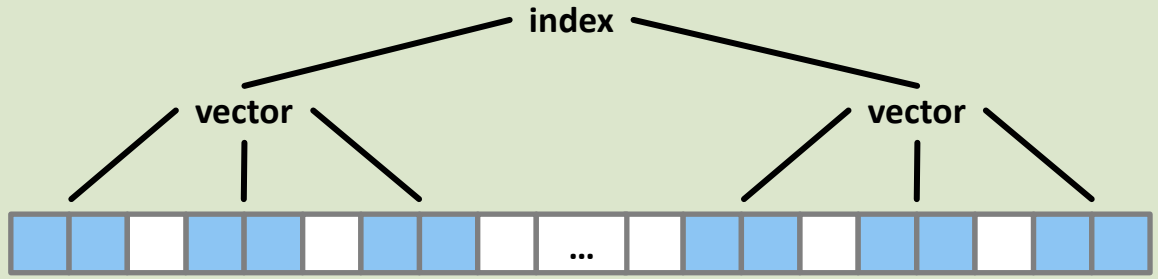
Snapshot 4: Index: Vector:

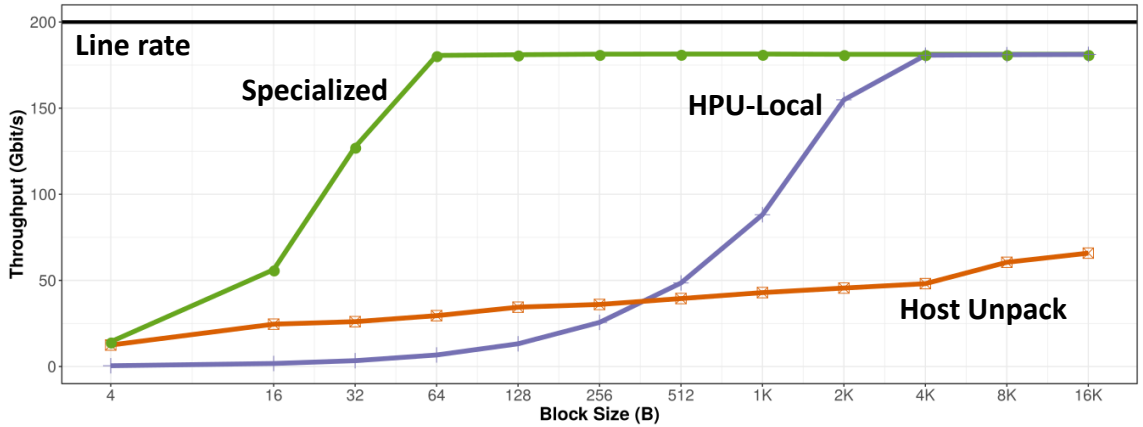
Snapshot 5: Index: Vector:

Packet Scheduler

HPU 0 | HPU 1 | HPU 2 | HPU 3

Host Memory





MPI Types Library on sPIN: Read-Only Checkpoints

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

$\Delta t = 2$

Snapshot 0: Index: Vector:

Snapshot 1: Index: Vector:

Snapshot 2: Index: Vector:

Snapshot 3: Index: Vector:

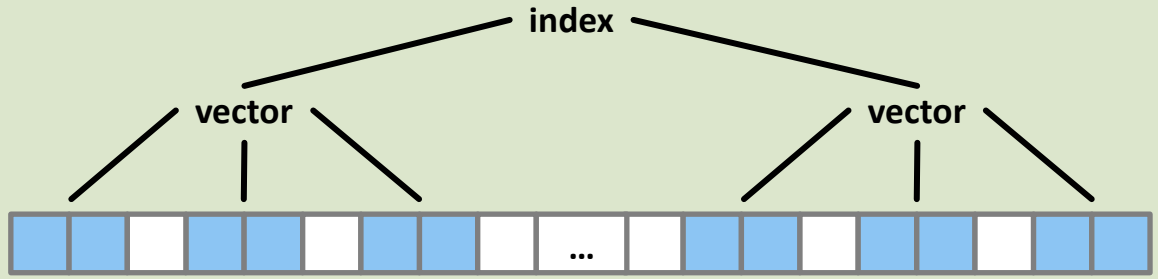
Snapshot 4: Index: Vector:

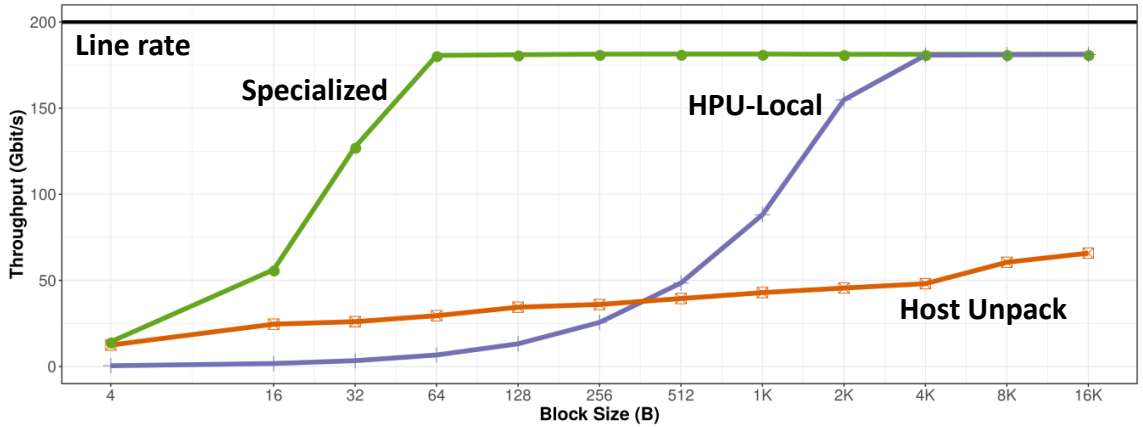
Snapshot 5: Index: Vector:

Packet Scheduler

HPU 0 | HPU 1 | HPU 2 | HPU 3

Host Memory





MPI Types Library on sPIN: Read-Only Checkpoints

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

$\Delta t = 2$

Snapshot 0: Index: Vector:

Snapshot 1: Index: Vector:

Snapshot 2: Index: Vector:

Snapshot 3: Index: Vector:

Snapshot 4: Index: Vector:

Snapshot 5: Index: Vector:

Packet Scheduler

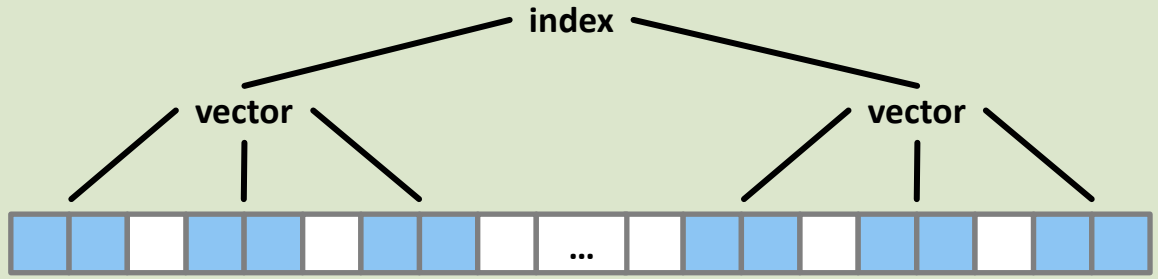
HPU 0: Index: Vector:

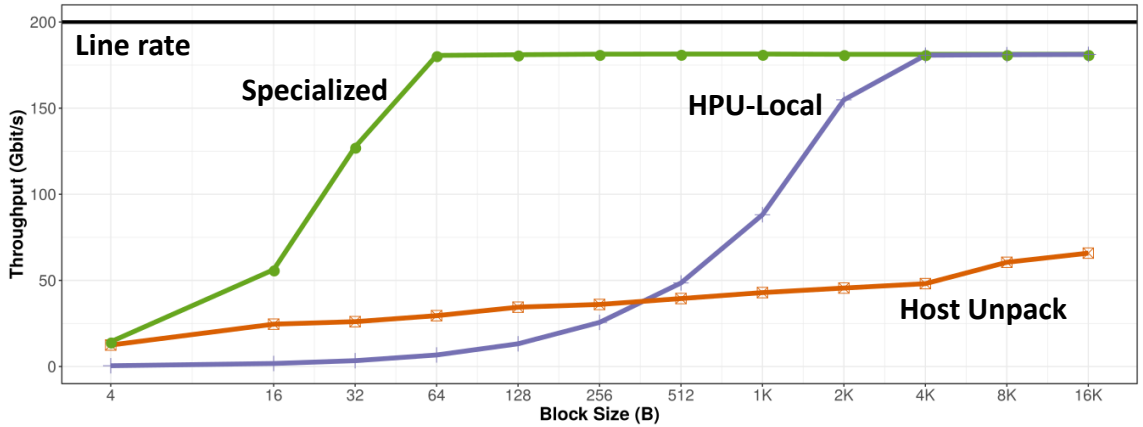
HPU 1: Index: Vector:

HPU 2: Index: Vector:

HPU 3: Index: Vector:

Host Memory





MPI Types Library on sPIN: Read-Only Checkpoints

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

$\Delta t = 2$

Snapshot 0: Index: Vector:

Snapshot 1: Index: Vector:

Snapshot 2: Index: Vector:

Snapshot 3: Index: Vector:

Snapshot 4: Index: Vector:

Snapshot 5: Index: Vector:

Packet Scheduler

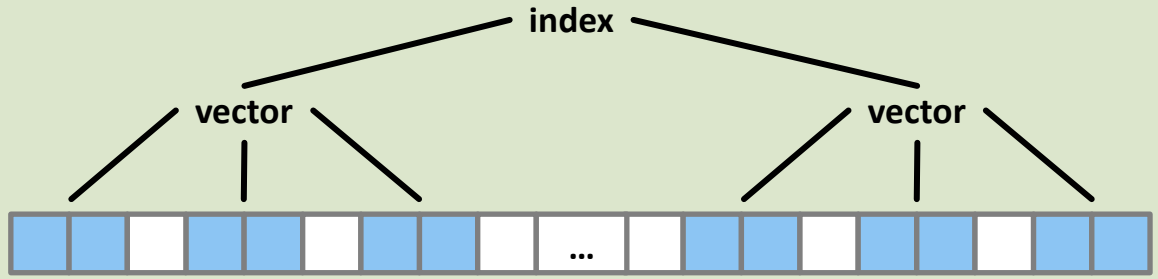
HPU 0: Index: Vector:

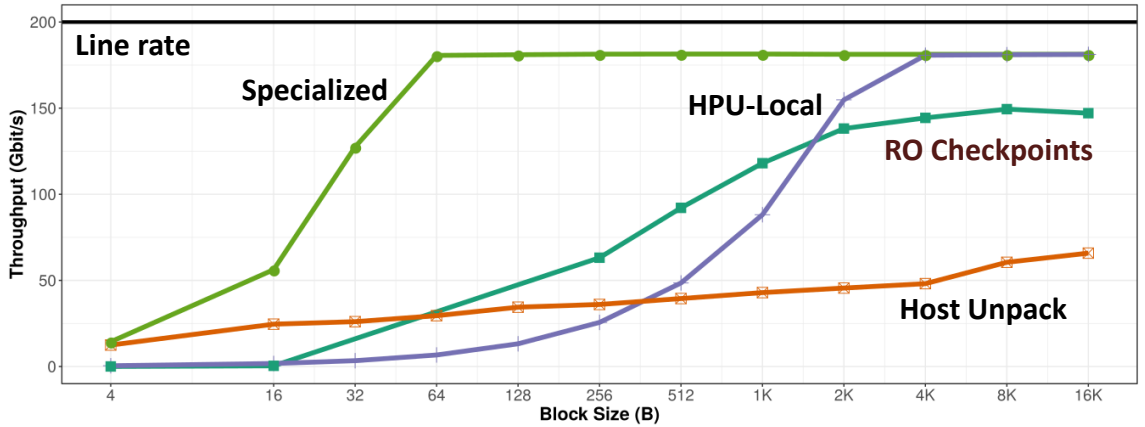
HPU 1: Index: Vector:

HPU 2: Index: Vector:

HPU 3: Index: Vector:

Host Memory





MPI Types Library on sPIN: Read-Only Checkpoints

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

$\Delta t = 2$

Snapshot 0: Index: Vector:

Snapshot 1: Index: Vector:

Snapshot 2: Index: Vector:

Snapshot 3: Index: Vector:

Snapshot 4: Index: Vector:

Snapshot 5: Index: Vector:

Packet Scheduler

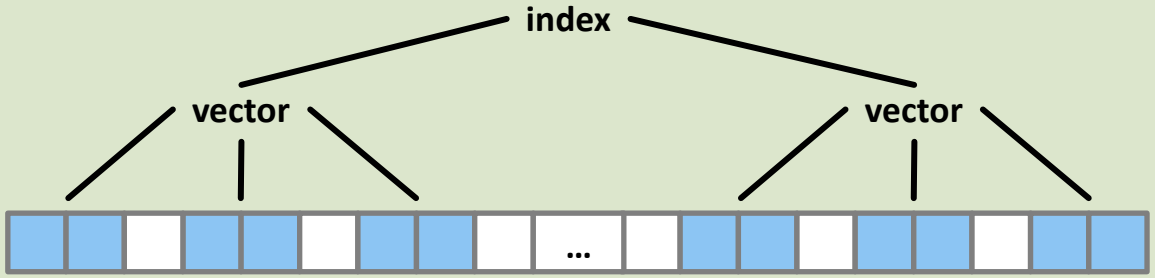
HPU 0: Index: Vector:

HPU 1: Index: Vector:

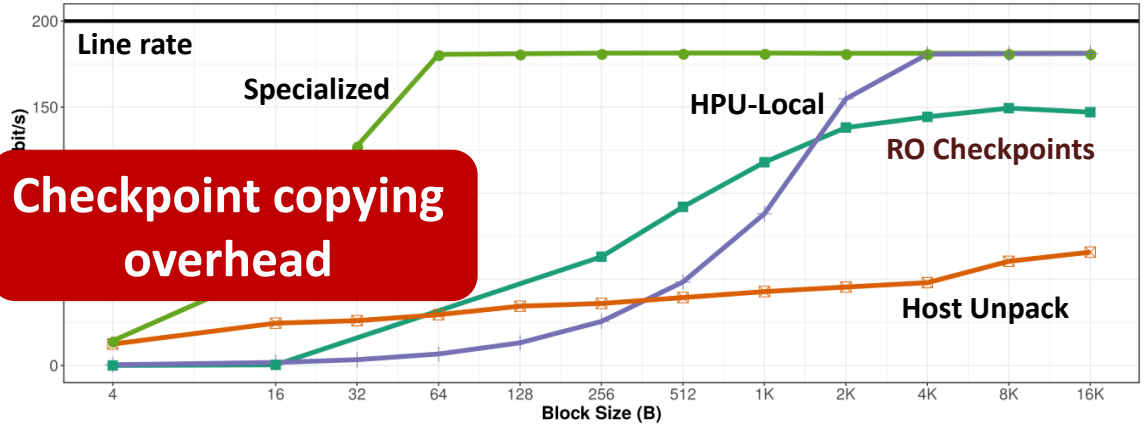
HPU 2: Index: Vector:

HPU 3: Index: Vector:

Host Memory



Checkpoint copying overhead



MPI Types Library on sPIN: Read-Write Checkpoints

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

$\Delta t = 2$

Index:
Vector:

Index:
Vector:

Index:
Vector:

Index:
Vector:

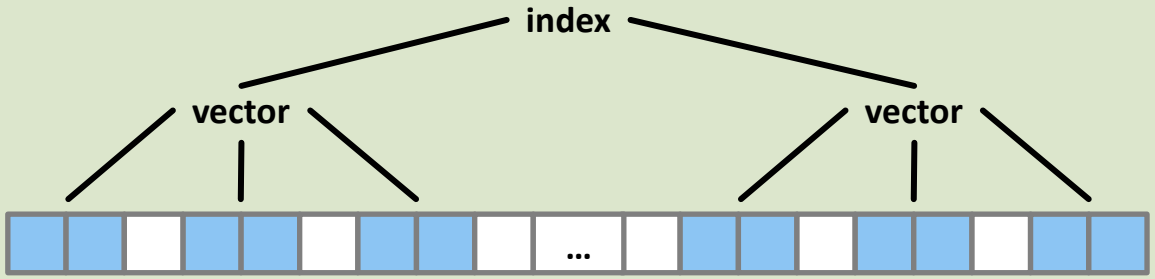
Index:
Vector:

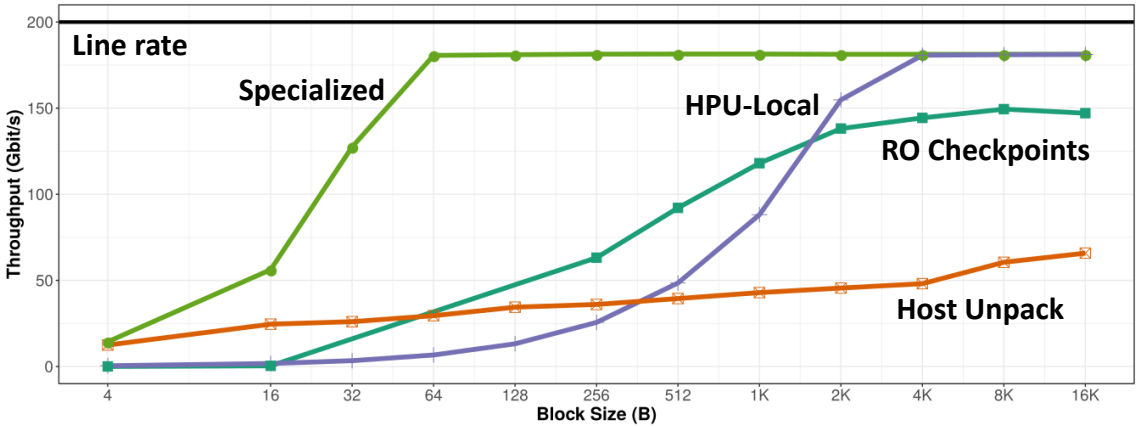
Index:
Vector:

Packet Scheduler

HPU 0 | HPU 1 | HPU 2 | HPU 3

Host Memory





MPI Types Library on sPIN: Read-Write Checkpoints

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

$\Delta t = 2$

V-HPU 0
Index:
Vector:

V-HPU 5
Index:
Vector:

V-HPU 1
Index:
Vector:

V-HPU 2
Index:
Vector:

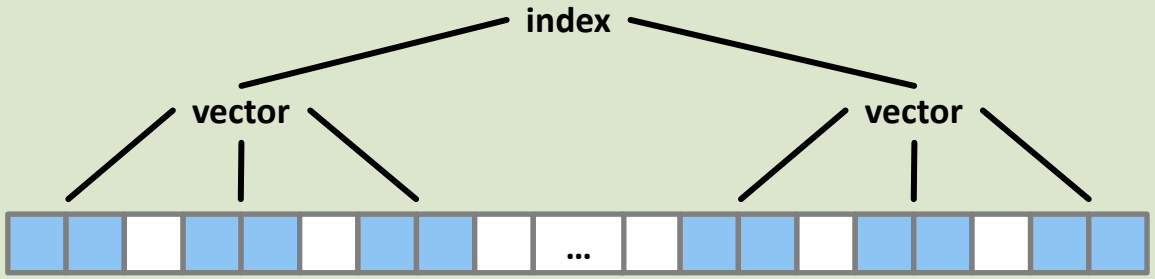
V-HPU 3
Index:
Vector:

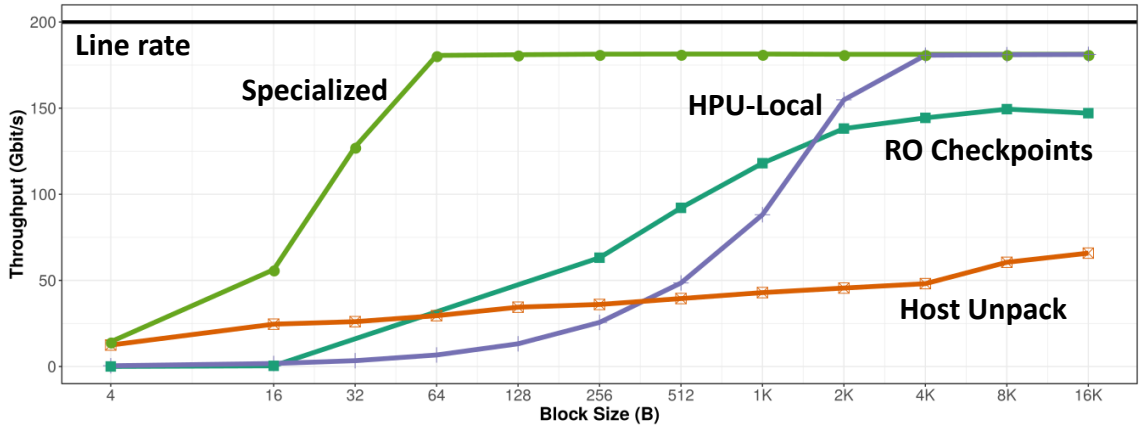
V-HPU 4
Index:
Vector:

Packet Scheduler

HPU 0 | HPU 1 | HPU 2 | HPU 3

Host Memory





MPI Types Library on sPIN: Read-Write Checkpoints

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

$\Delta t = 2$

V-HPU 0 — Index: Vector:

V-HPU 1 — Index: Vector:

V-HPU 2 — Index: Vector:

V-HPU 3 — Index: Vector:

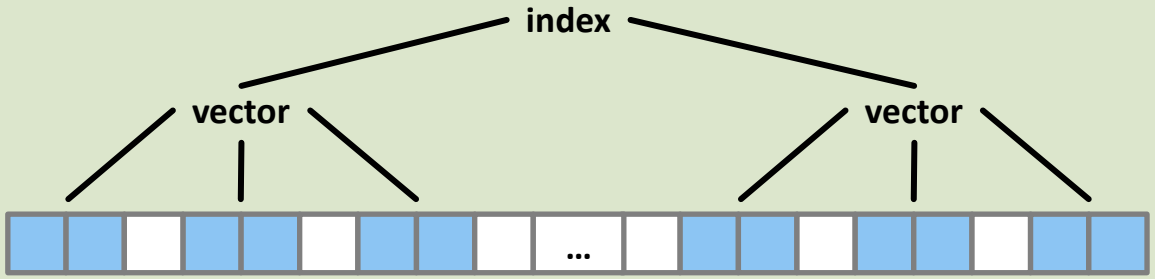
V-HPU 4 — Index: Vector:

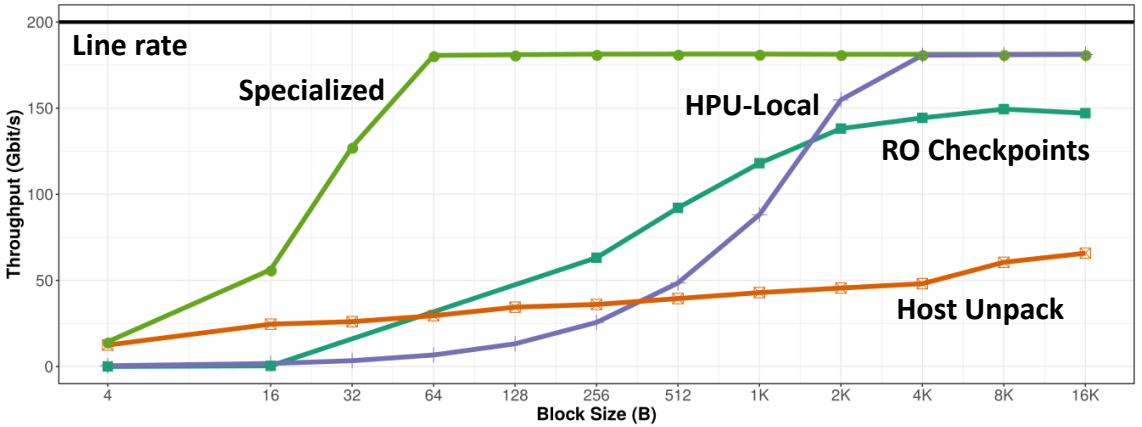
V-HPU 5 — Index: Vector:

Packet Scheduler

HPU 0 | HPU 1 | HPU 2 | HPU 3

Host Memory





MPI Types Library on sPIN: Read-Write Checkpoints

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

$\Delta t = 2$

Index:
Vector:

V-HPU 5
Index:
Vector:

V-HPU 1
Index:
Vector:

V-HPU 2
Index:
Vector:

V-HPU 3
Index:
Vector:

V-HPU 4
Index:
Vector:

Packet Scheduler

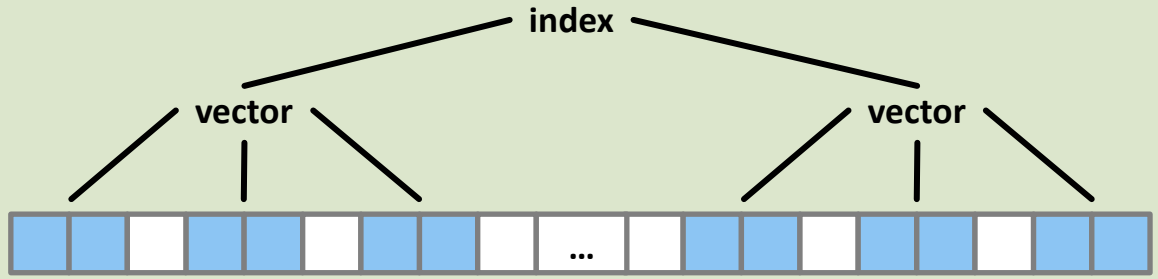
HPU 0
V-HPU 0

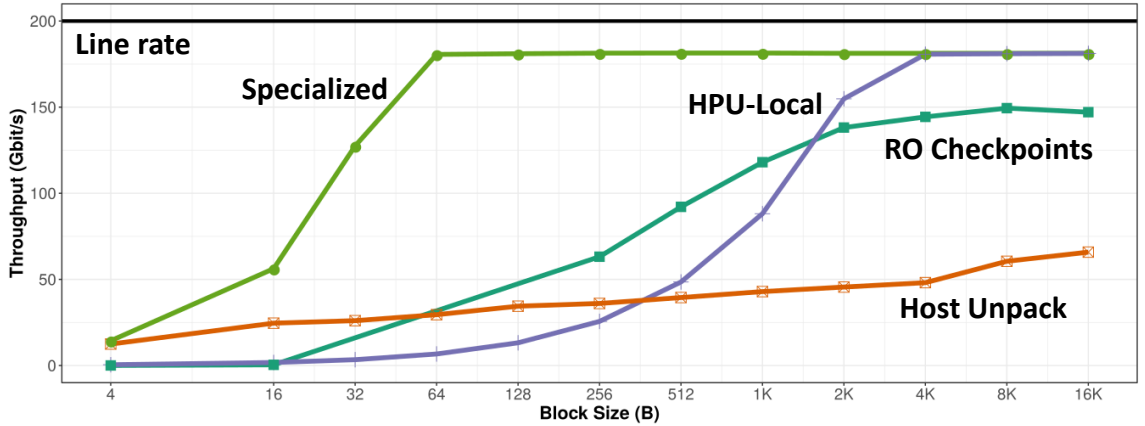
HPU 1

HPU 2

HPU 3

Host Memory





MPI Types Library on sPIN: Read-Write Checkpoints

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

$\Delta t = 2$

Index:
Vector:

V-HPU 5
Index:
Vector:

V-HPU 1
Index:
Vector:

V-HPU 2
Index:
Vector:

V-HPU 3
Index:
Vector:

V-HPU 4
Index:
Vector:

Packet Scheduler

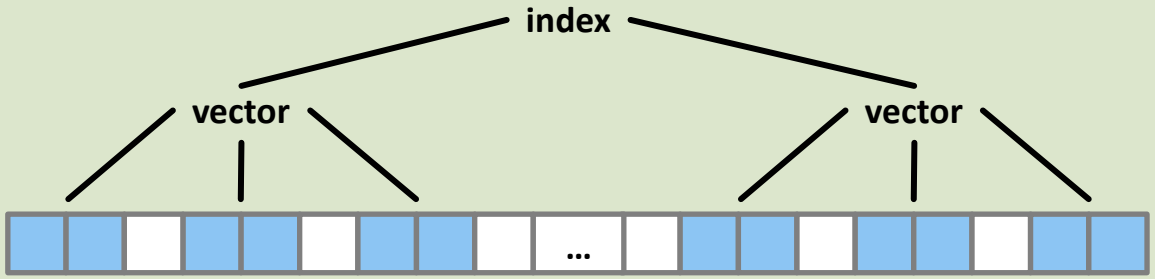
HPU 0
V-HPU 0

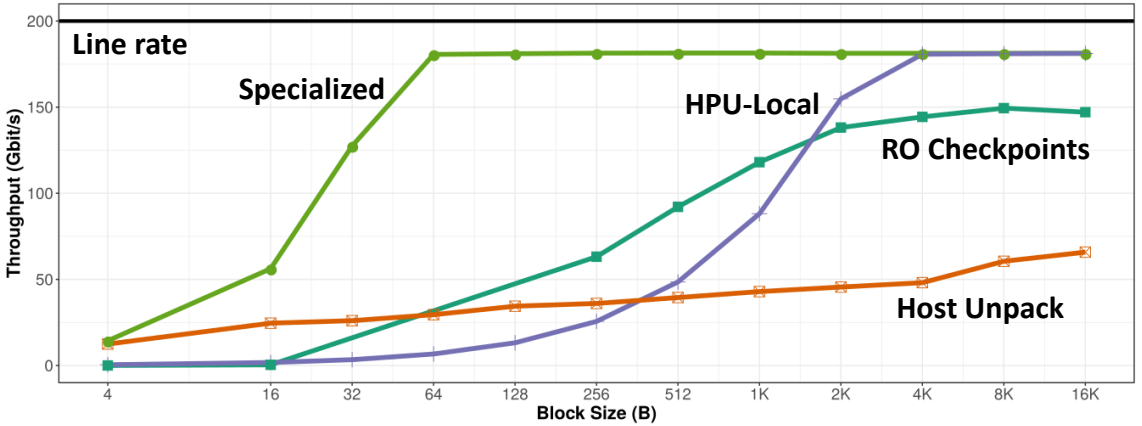
HPU 1

HPU 2

HPU 3

Host Memory





MPI Types Library on sPIN: Read-Write Checkpoints

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

$\Delta t = 2$

Index:
Vector:

V-HPU 5
Index:
Vector:

V-HPU 1
Index:
Vector:

V-HPU 2
Index:
Vector:

V-HPU 3
Index:
Vector:

V-HPU 4
Index:
Vector:

Packet Scheduler

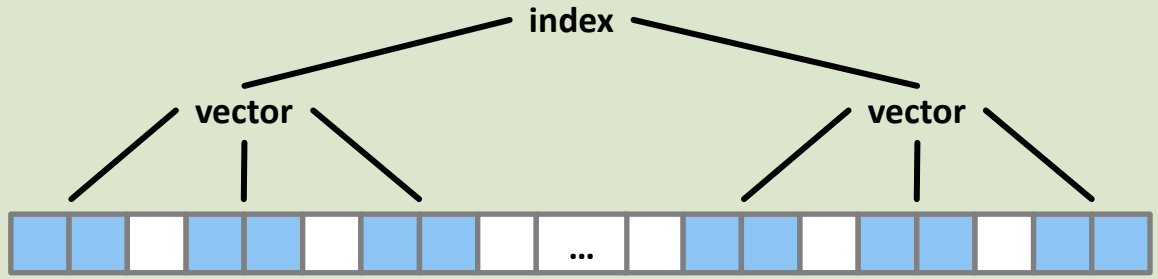
HPU 0
V-HPU 0

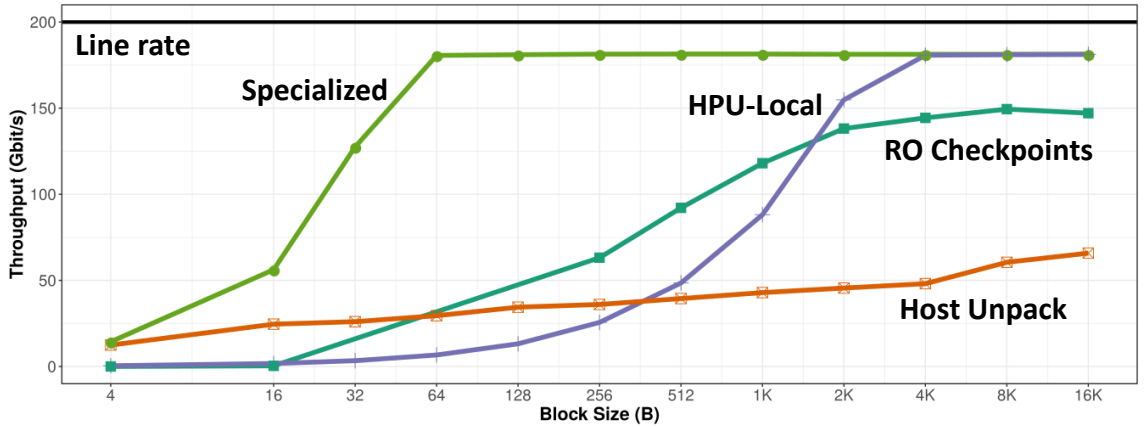
HPU 1

HPU 2

HPU 3

Host Memory





MPI Types Library on sPIN: Read-Write Checkpoints

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

$\Delta t = 2$

Index:
Vector:

V-HPU 5
Index:
Vector:

Index:
Vector:

V-HPU 2
Index:
Vector:

V-HPU 3
Index:
Vector:

V-HPU 4
Index:
Vector:

Packet Scheduler

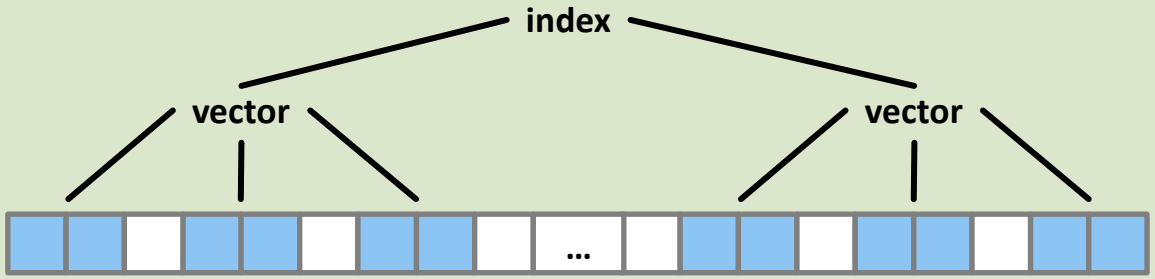
HPU 0
V-HPU 0

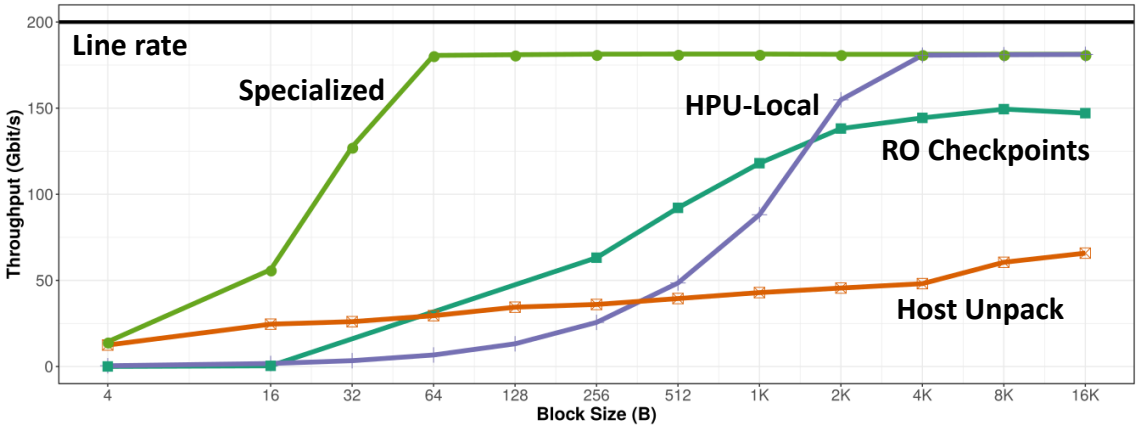
HPU 1

HPU 2
V-HPU 1

HPU 3

Host Memory





MPI Types Library on sPIN: Read-Write Checkpoints

NIC Memory

Index{ #blocks: 2, blocklen: 1, offsets: {0, x}, basetype: * }

Vector{ #blocks: 3, blocklen: 2, stride: 3, basetype: }

$\Delta t = 2$

Index:
Vector:

V-HPU 5
Index:
Vector:

Index:
Vector:

V-HPU 2
Index:
Vector:

V-HPU 3
Index:
Vector:

V-HPU 4
Index:
Vector:

Packet Scheduler

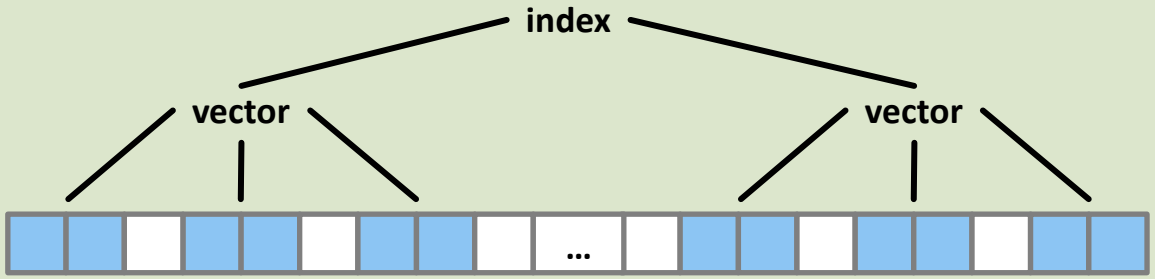
HPU 0
V-HPU 0

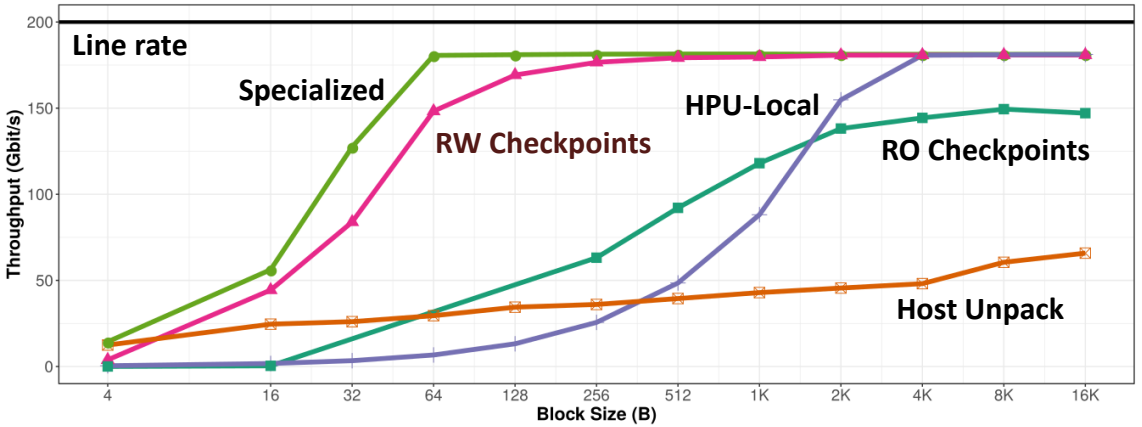
HPU 1

HPU 2
V-HPU 1

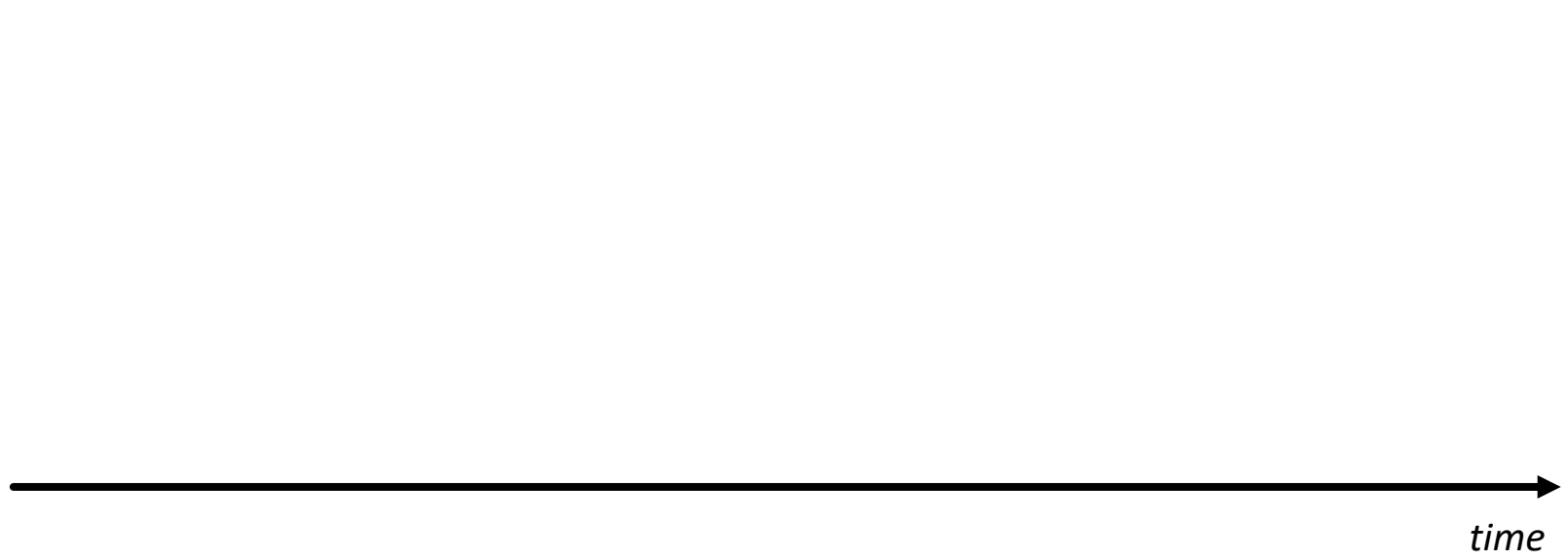
HPU 3

Host Memory





Checkpoint Interval Selection

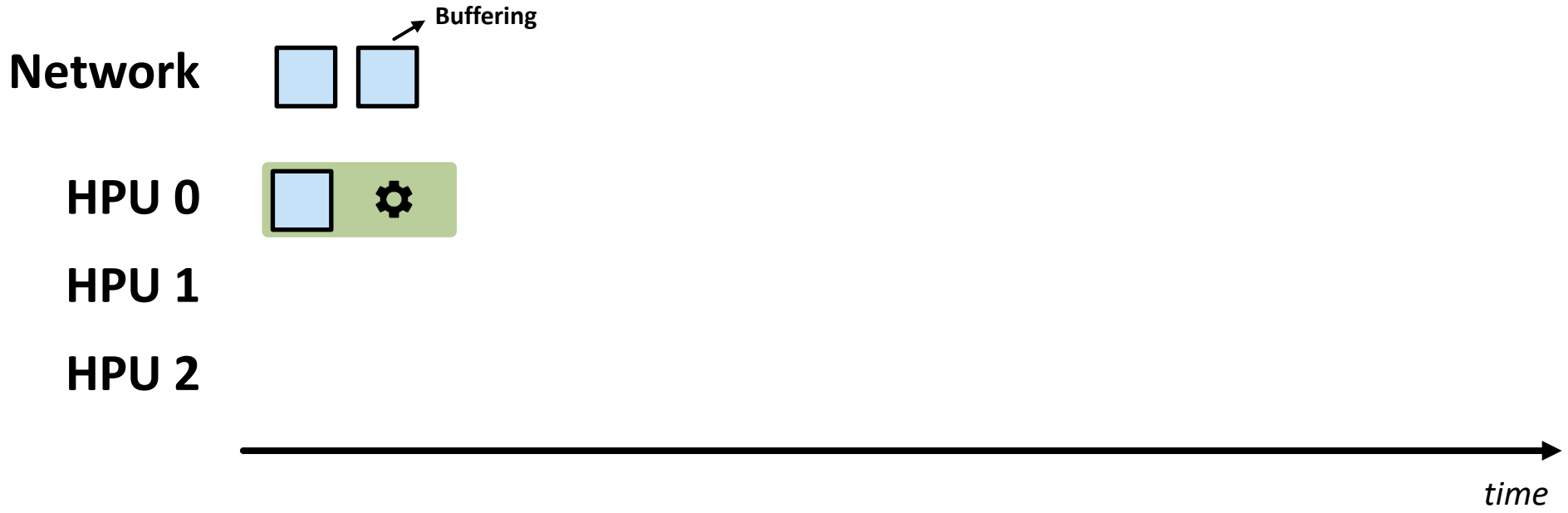


Checkpoint Interval Selection

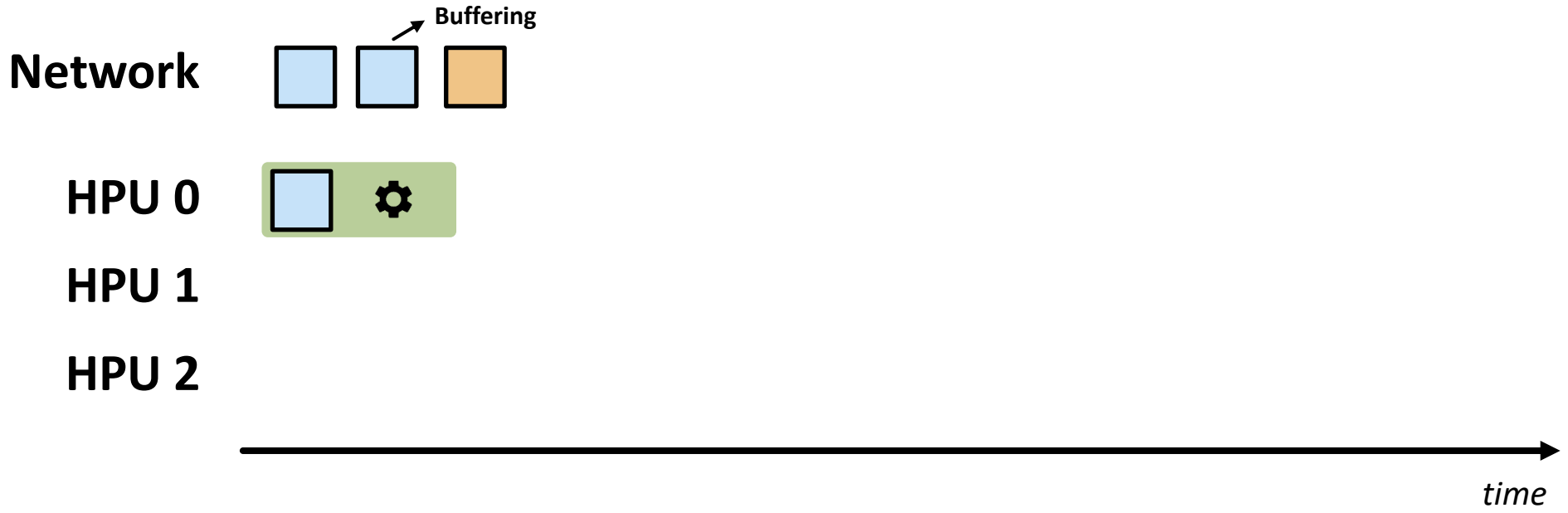
Checkpoint Interval Selection

Checkpoint Interval Selection

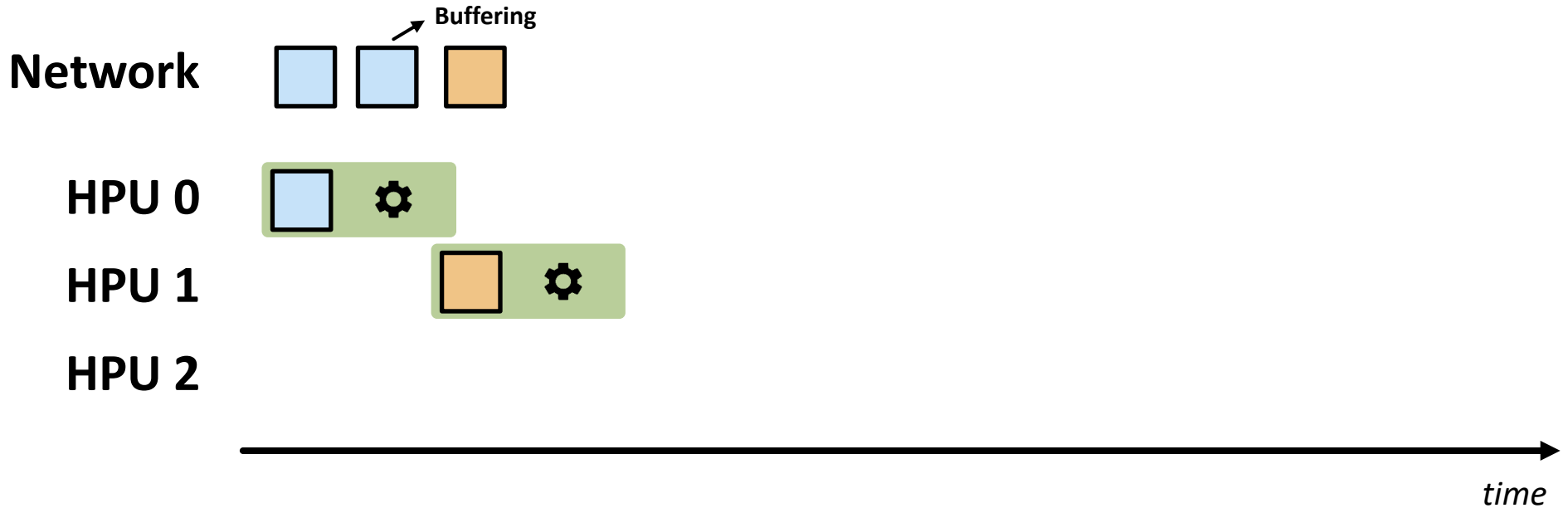
Checkpoint Interval Selection



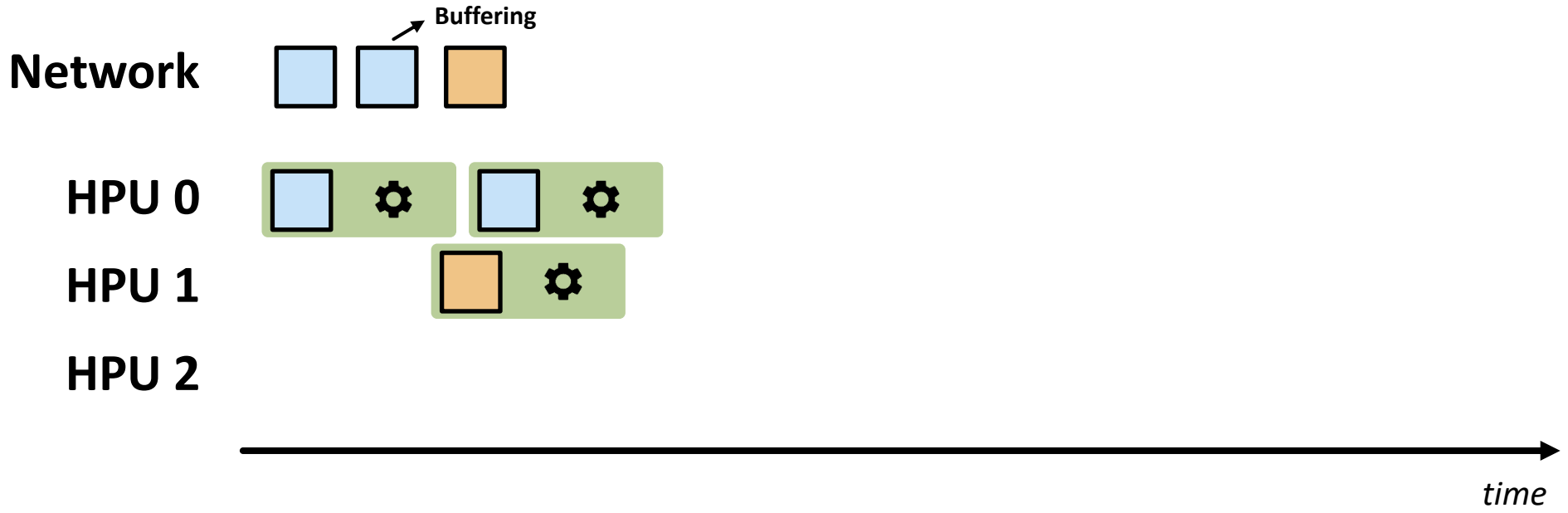
Checkpoint Interval Selection



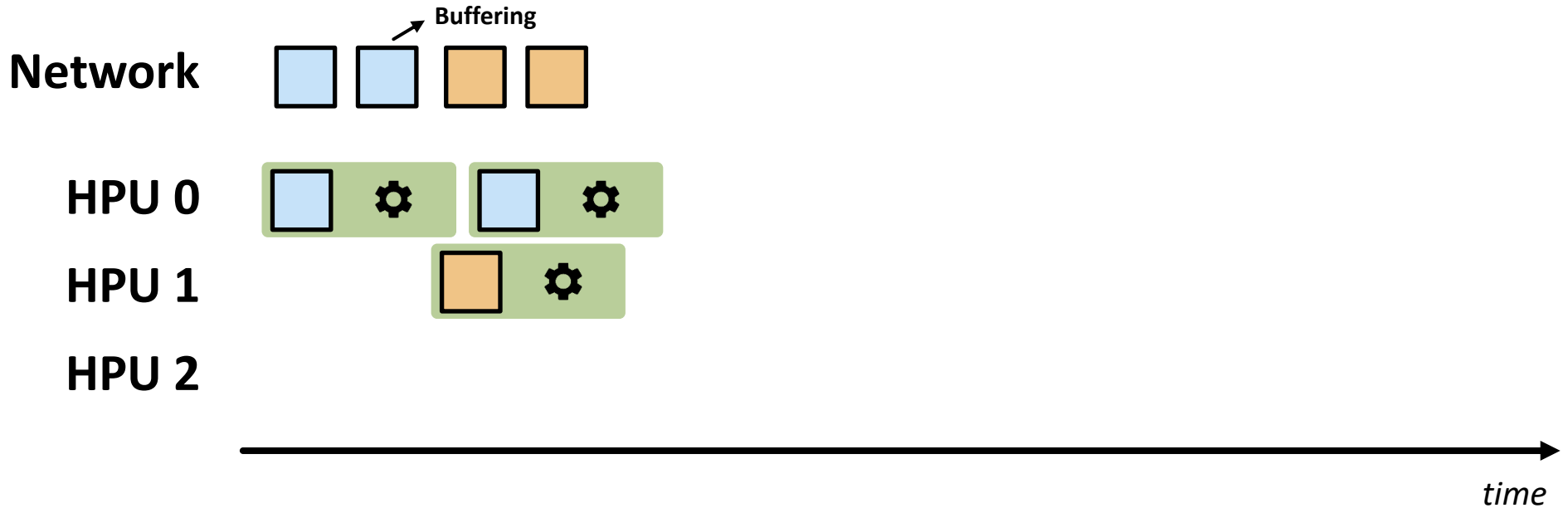
Checkpoint Interval Selection



Checkpoint Interval Selection

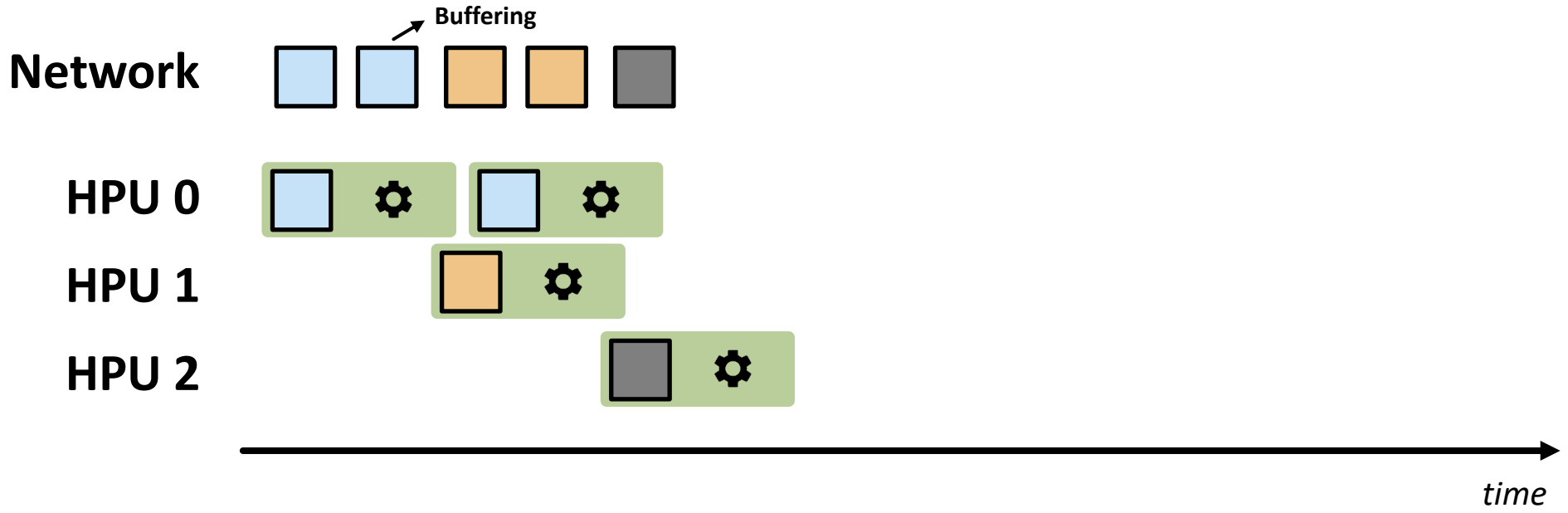


Checkpoint Interval Selection

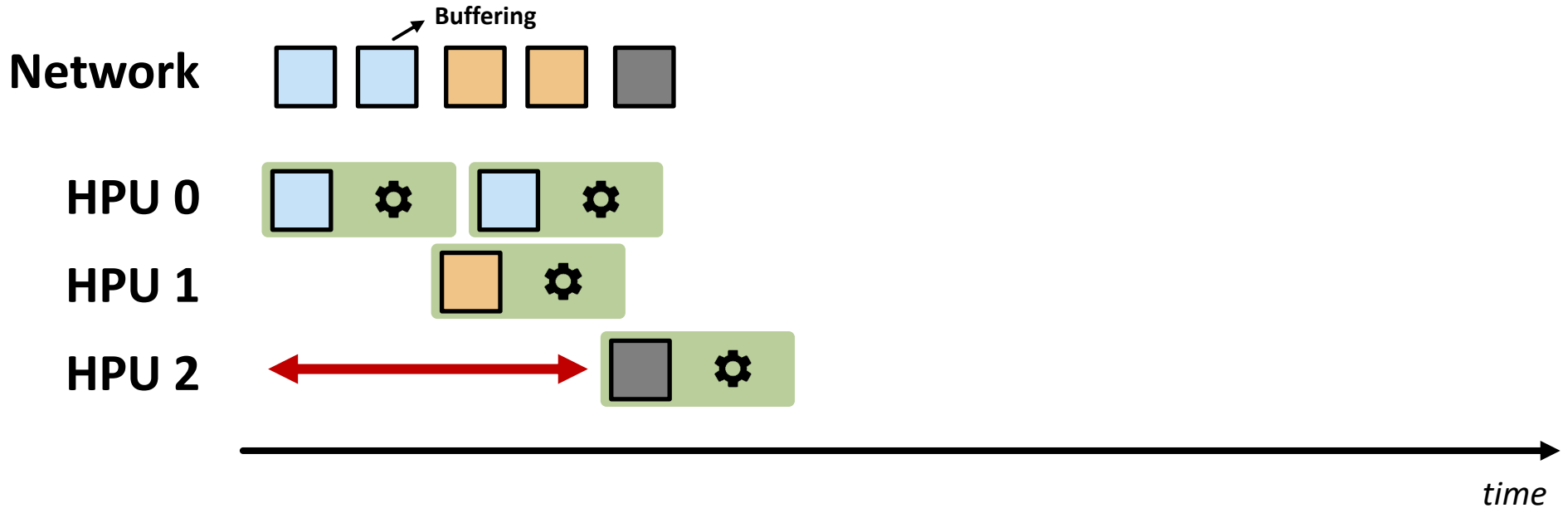


Checkpoint Interval Selection

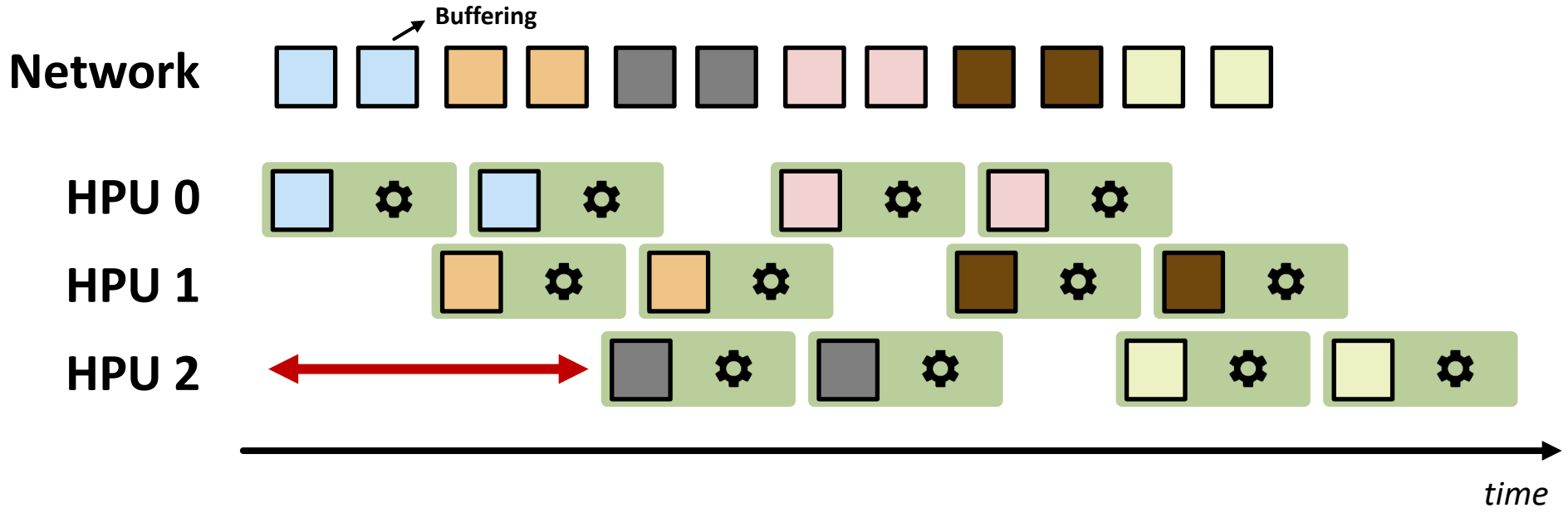
Checkpoint Interval Selection



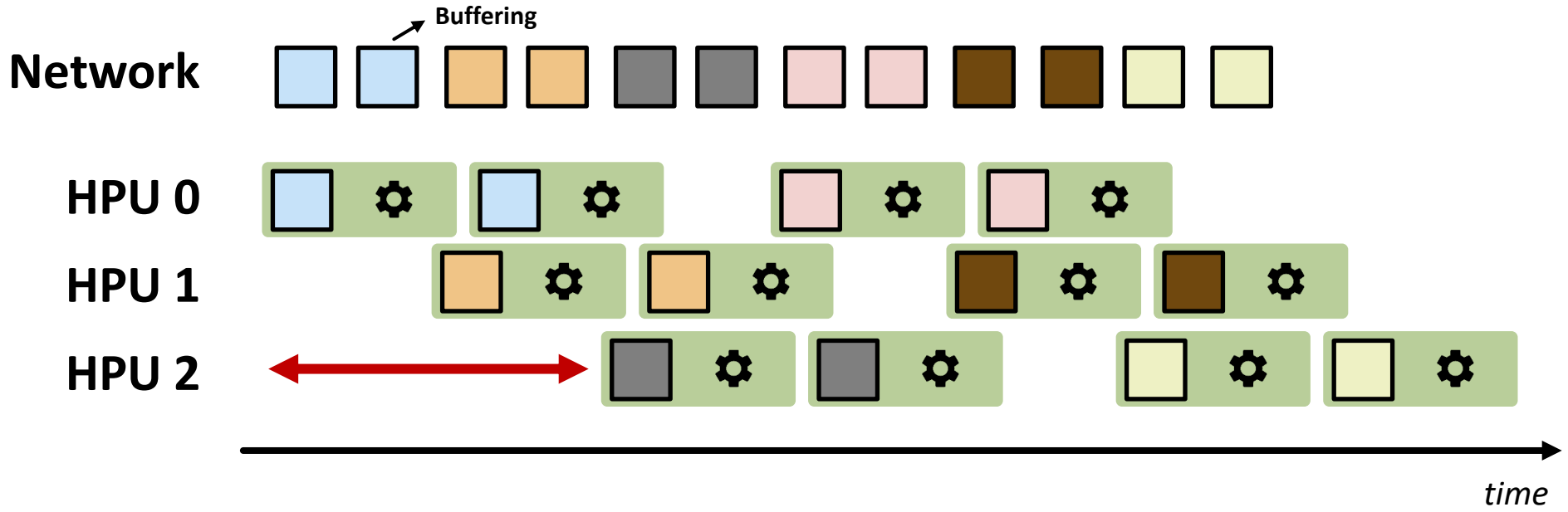
Checkpoint Interval Selection



Checkpoint Interval Selection

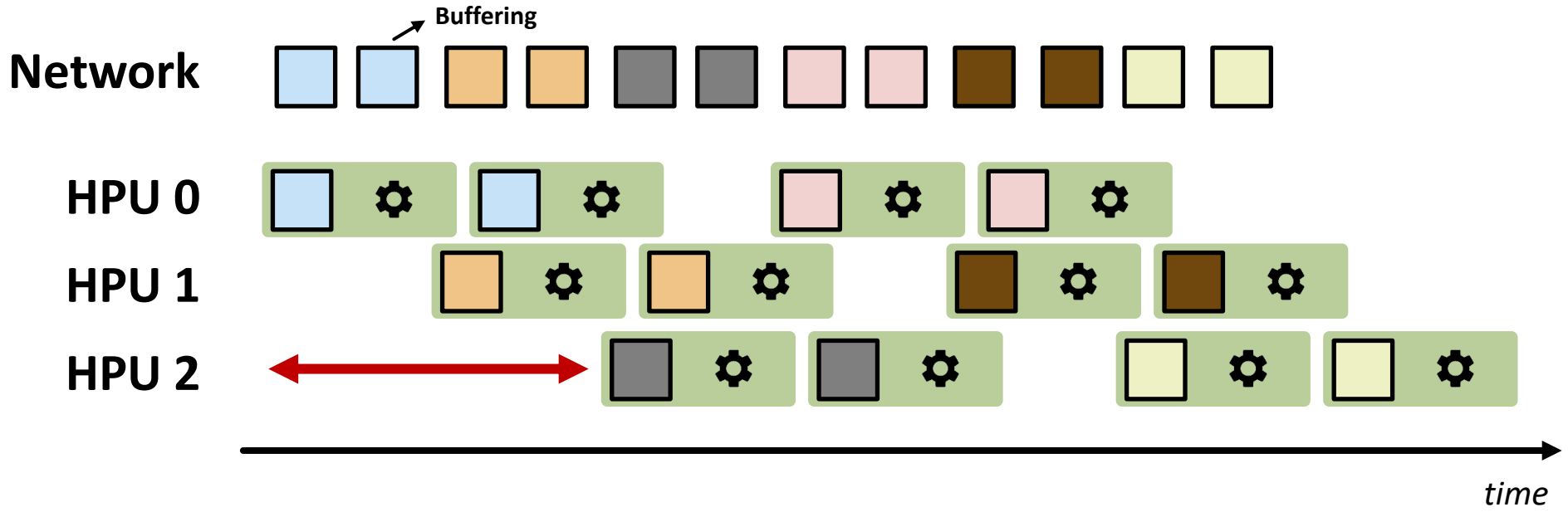


Checkpoint Interval Selection



$$T_C = T_{pkt} + \left\lceil \frac{\Delta r}{k} \right\rceil \cdot (P-1) \cdot T_{pkt} + \left\lceil \frac{n_{pkt}}{P} \right\rceil \cdot T_{PH}(\gamma)$$

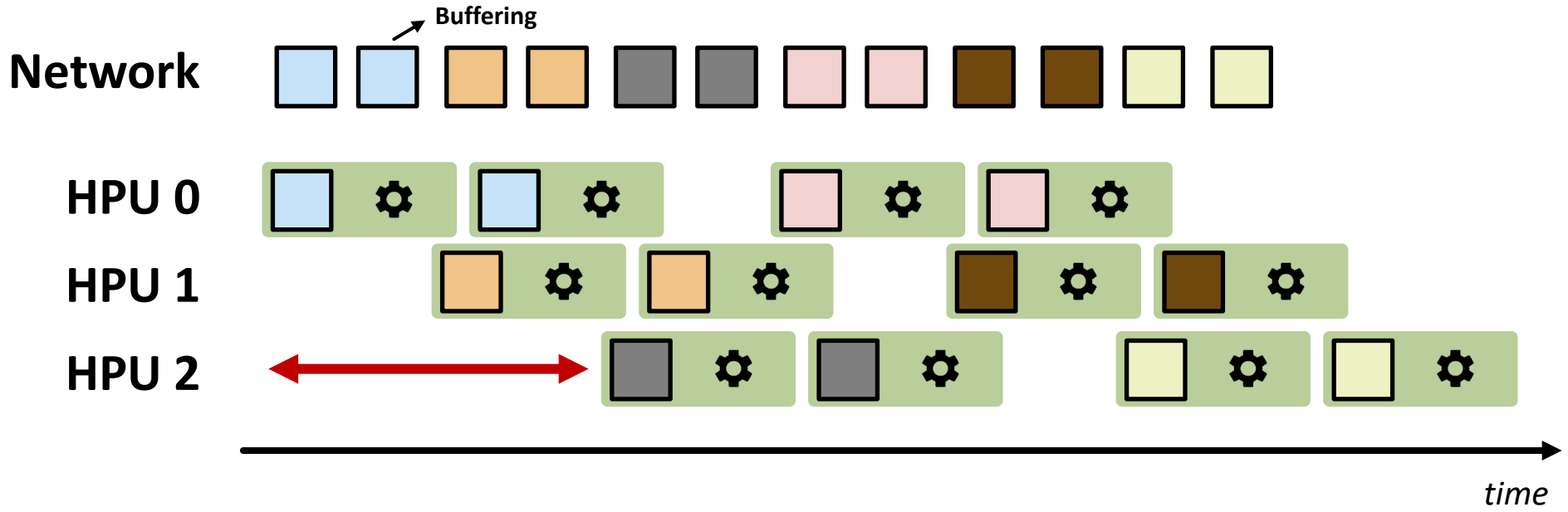
Checkpoint Interval Selection



$$T_C = T_{pkt} + \left\lceil \frac{\Delta r}{k} \right\rceil \cdot (P-1) \cdot T_{pkt} + \left\lceil \frac{n_{pkt}}{P} \right\rceil \cdot T_{PH}(\gamma)$$

1 **Limit the impact of the scheduling overhead**

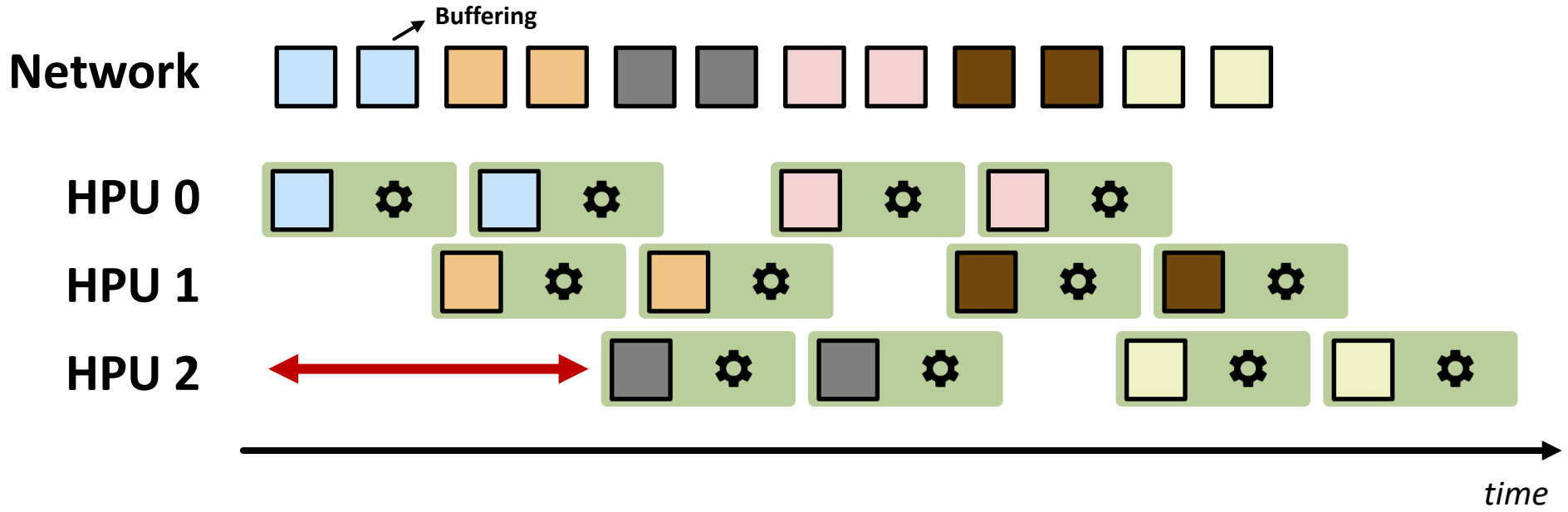
Checkpoint Interval Selection



$$T_C = T_{pkt} + \left\lceil \frac{\Delta r}{k} \right\rceil \cdot (P-1) \cdot T_{pkt} + \left\lceil \frac{n_{pkt}}{P} \right\rceil \cdot T_{PH}(\gamma)$$

1. Limit the impact of the scheduling overhead
2. Do not saturate NIC memory wih checkpoints

Checkpoint Interval Selection

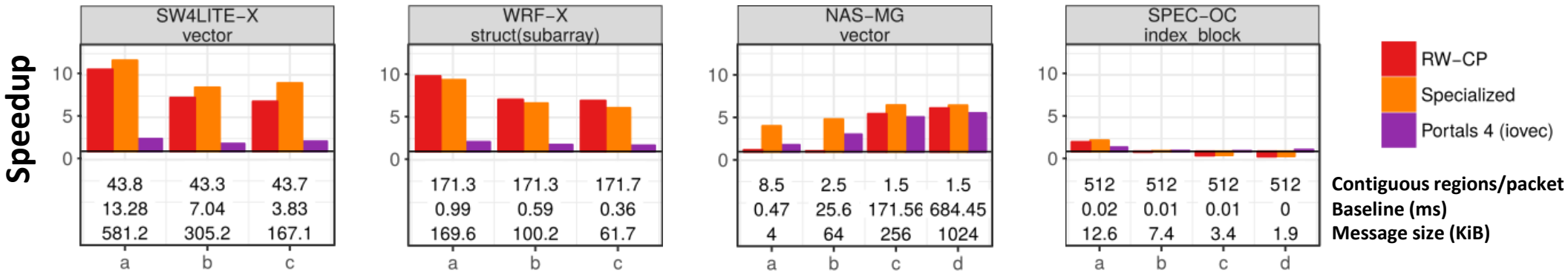


$$T_C = T_{pkt} + \left\lceil \frac{\Delta r}{k} \right\rceil \cdot (P-1) \cdot T_{pkt} + \left\lceil \frac{n_{pkt}}{P} \right\rceil \cdot T_{PH}(\gamma)$$

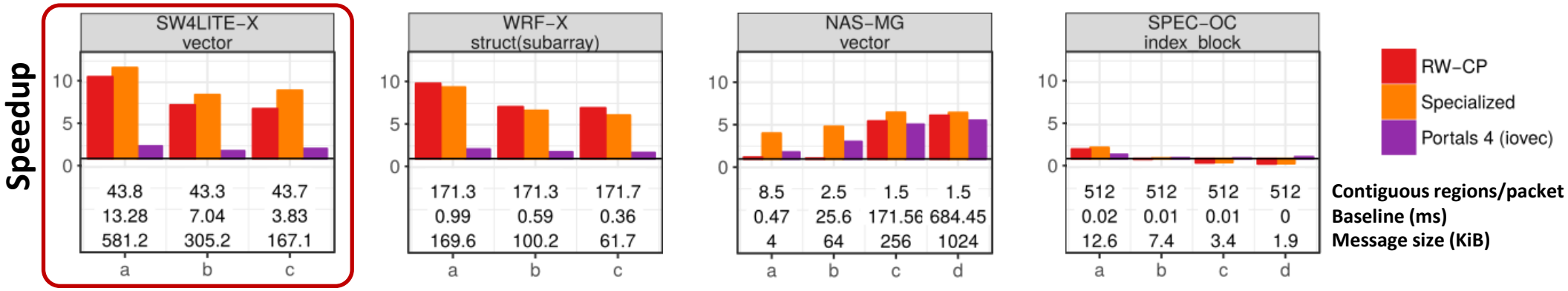
1. Limit the impact of the scheduling overhead
2. Do not saturate NIC memory wih checkpoints
3. Do not saturate the packet buffer

Real Applications DDTs

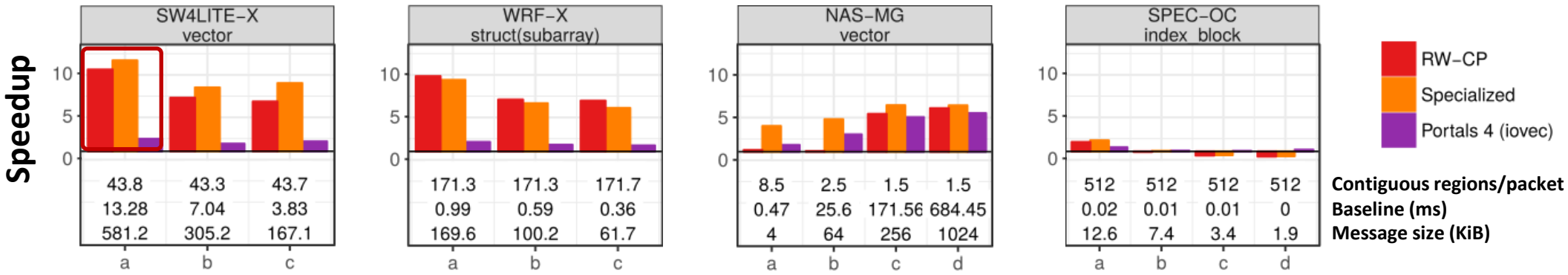
Real Applications DDTs



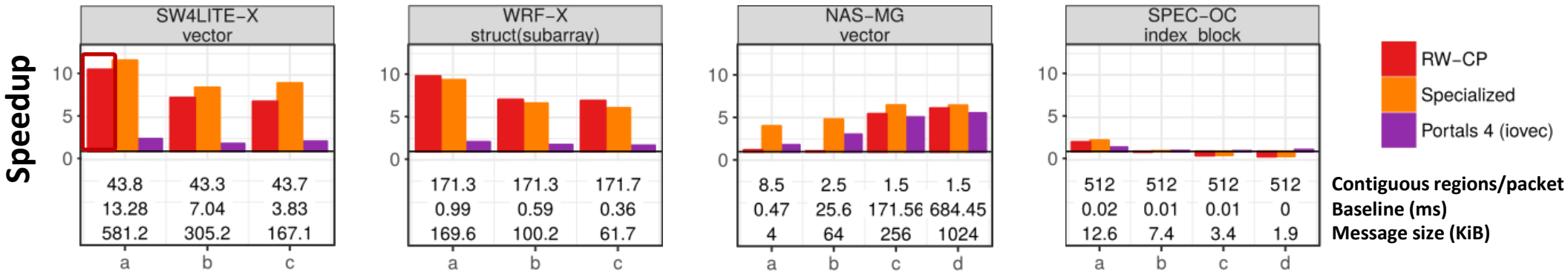
Real Applications DDTs



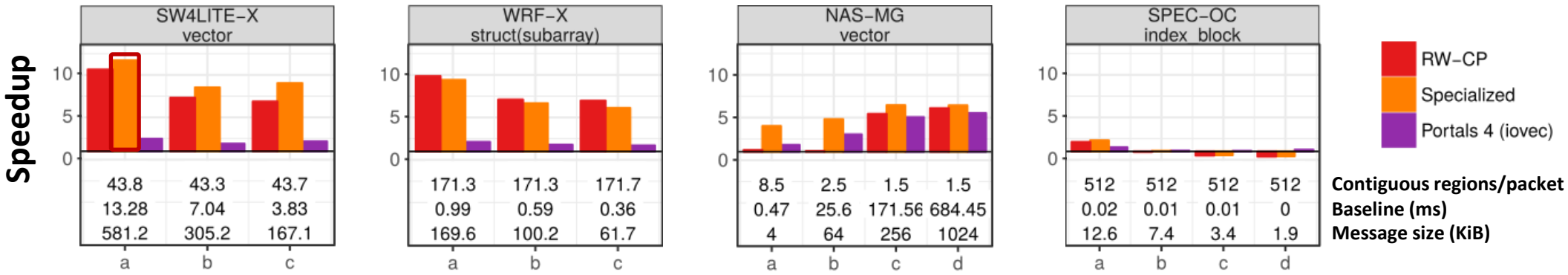
Real Applications DDTs



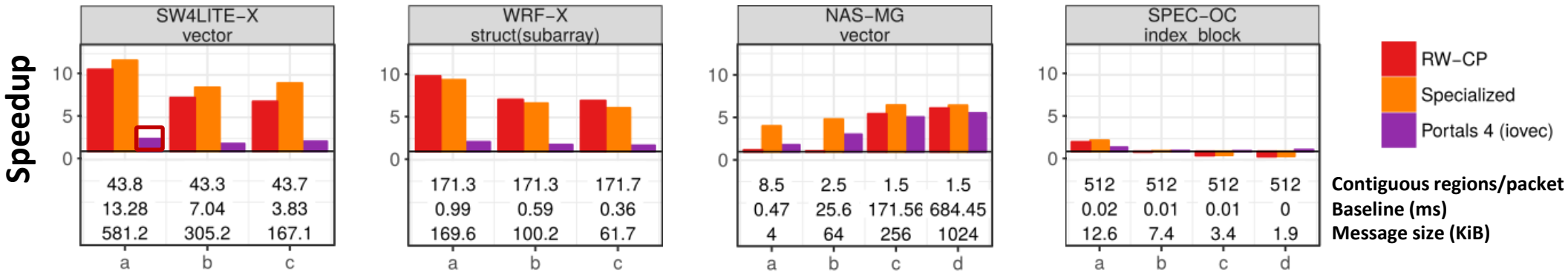
Real Applications DDTs



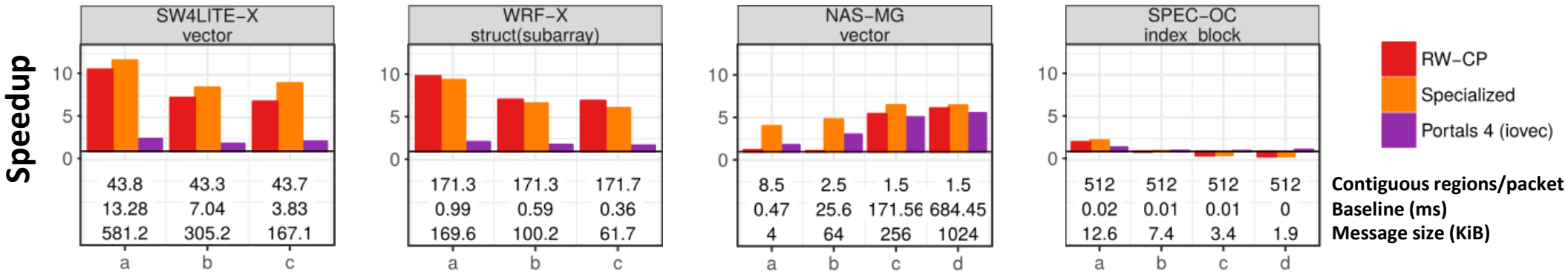
Real Applications DDTs



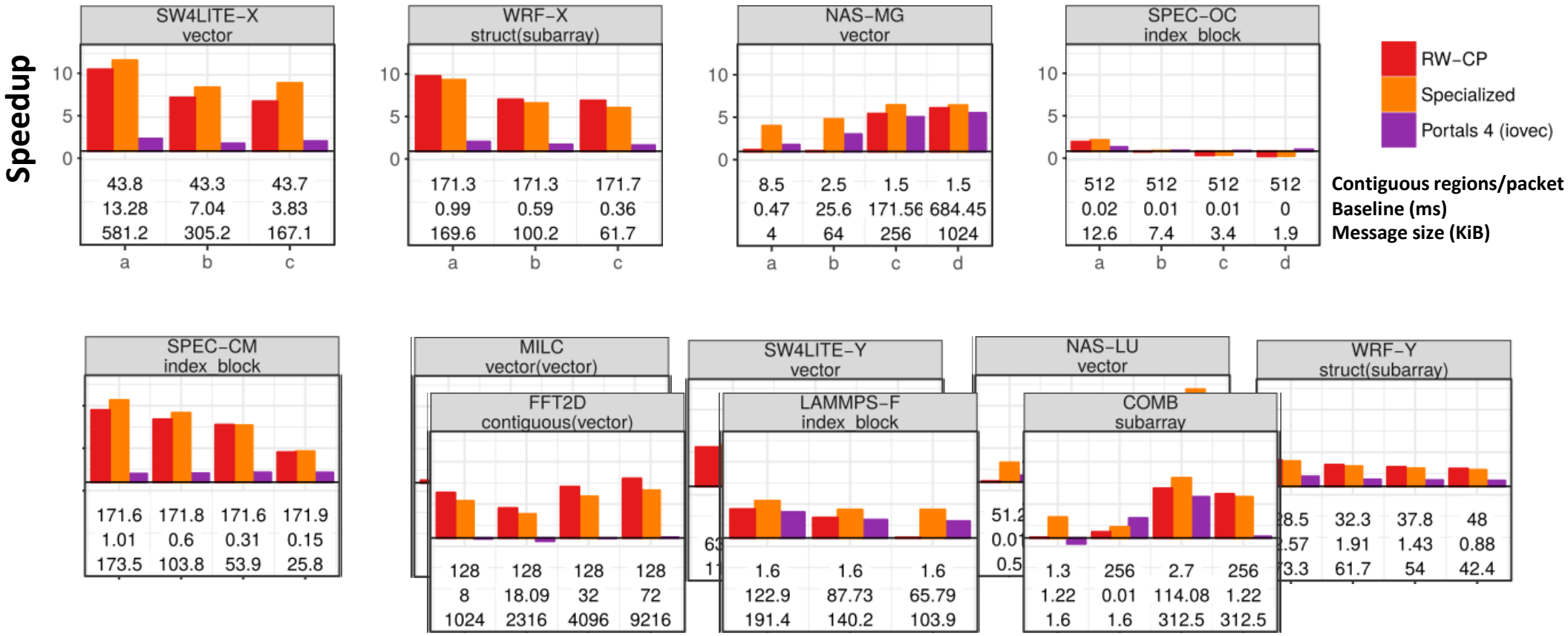
Real Applications DDTs



Real Applications DDTs

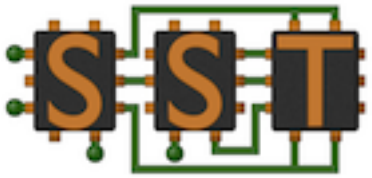


Real Applications DDTs



Cray Slingshot Simulator

Cray Slingshot Simulator



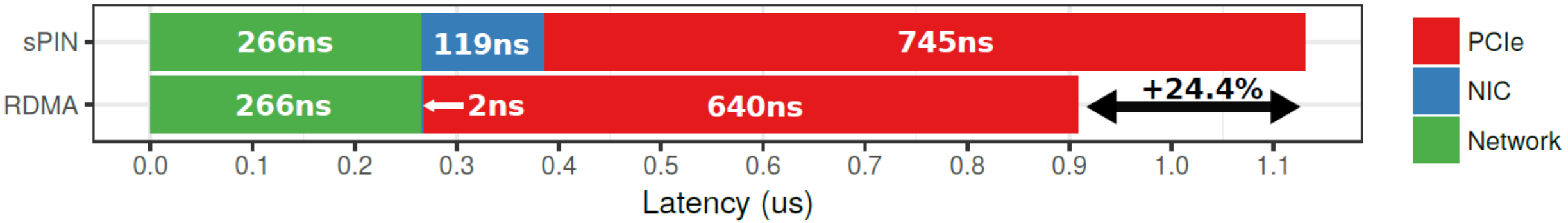
Cray Slingshot Simulator

+

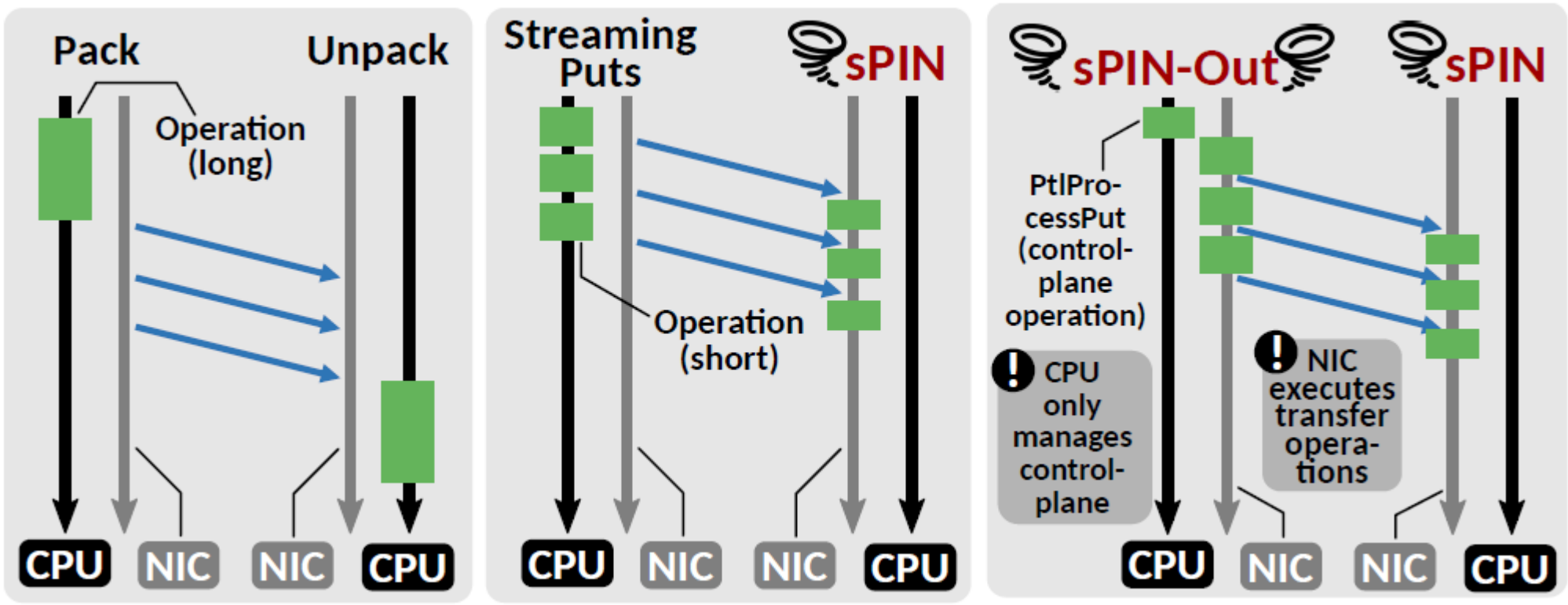
gem5

arm

32 Cortex A15 @800 MHz, single-cycle access memory

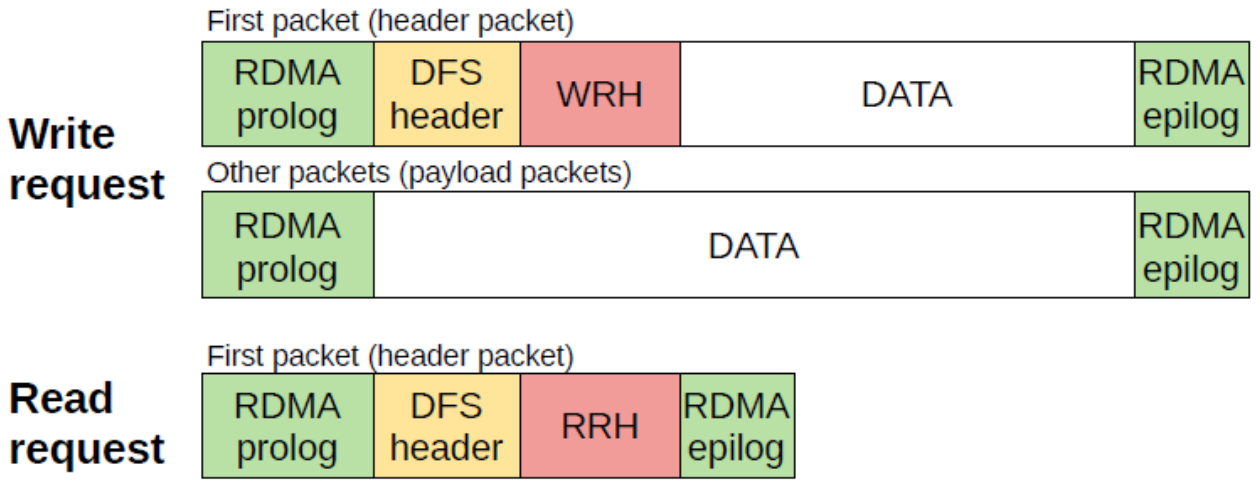

sPIN
RDMA
266ns
119ns
745ns
266ns
2ns
640ns
+24.4%
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
1.1
Latency (us)
PCIe
NIC
Network

Network-accelerated DDT processing strategies

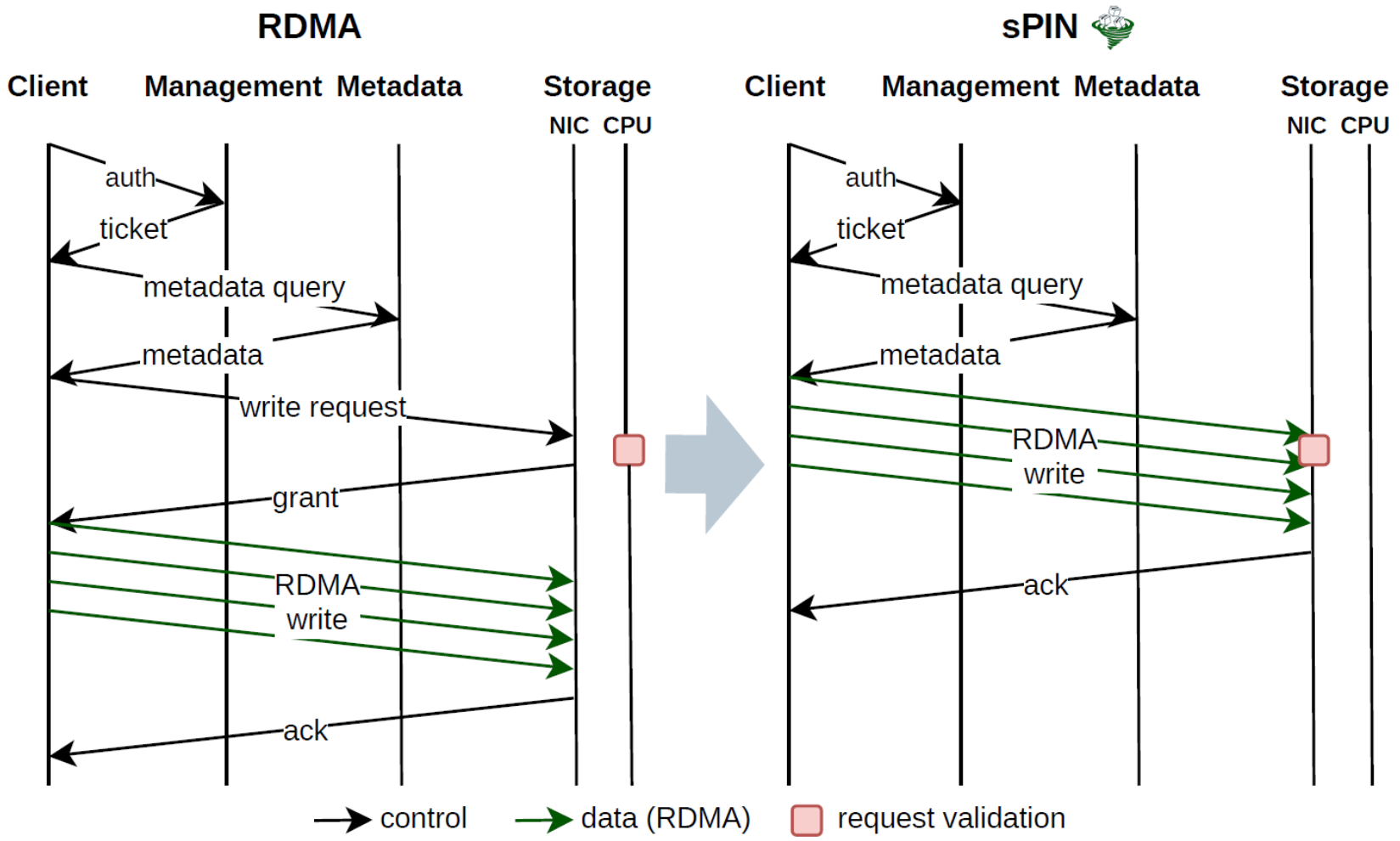


DFS handlers

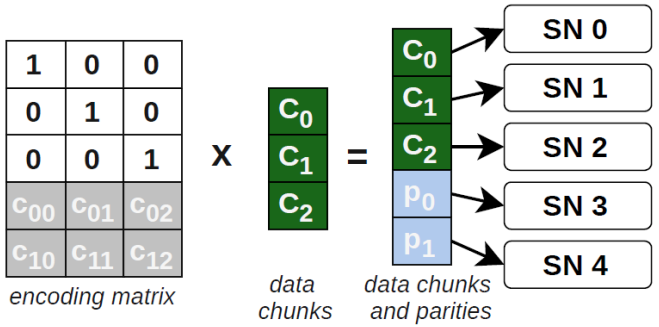
```
1  void header_handler(spin_task_t* task, pkt_t* pkt) {
2    dfs_state_t* state = (dfs_state_t*) task->mem;
3    bool accept_next_pkts = DFS_request_init(state, pkt);
4
5    int req_idx = task->flow_id;
6    state->req_table[req_idx].greq_id = pkt->dfs.greq_id;
7    state->req_table[req_idx].accept = accept_next_pkts;
8  }
9
10 void payload_handler(spin_task_t* task, pkt_t* pkt) {
11   dfs_state_t* state = (dfs_state_t*) task->mem;
12   int req_idx = task->flow_id;
13
14   if (state->req_table[req_idx].accept)
15     DFS_request_process_pkt(state, pkt);
16 }
17
18 void tail_handler(spin_task_t* task, pkt_t* pkt) {
19   dfs_state_t* state = (dfs_state_t*) task->mem;
20   int req_idx = task->flow_id;
21
22   if (state->req_table[req_idx].accept)
23     DFS_request_fini(state, pkt);
24 }
```

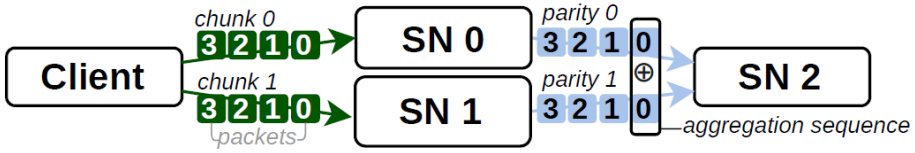


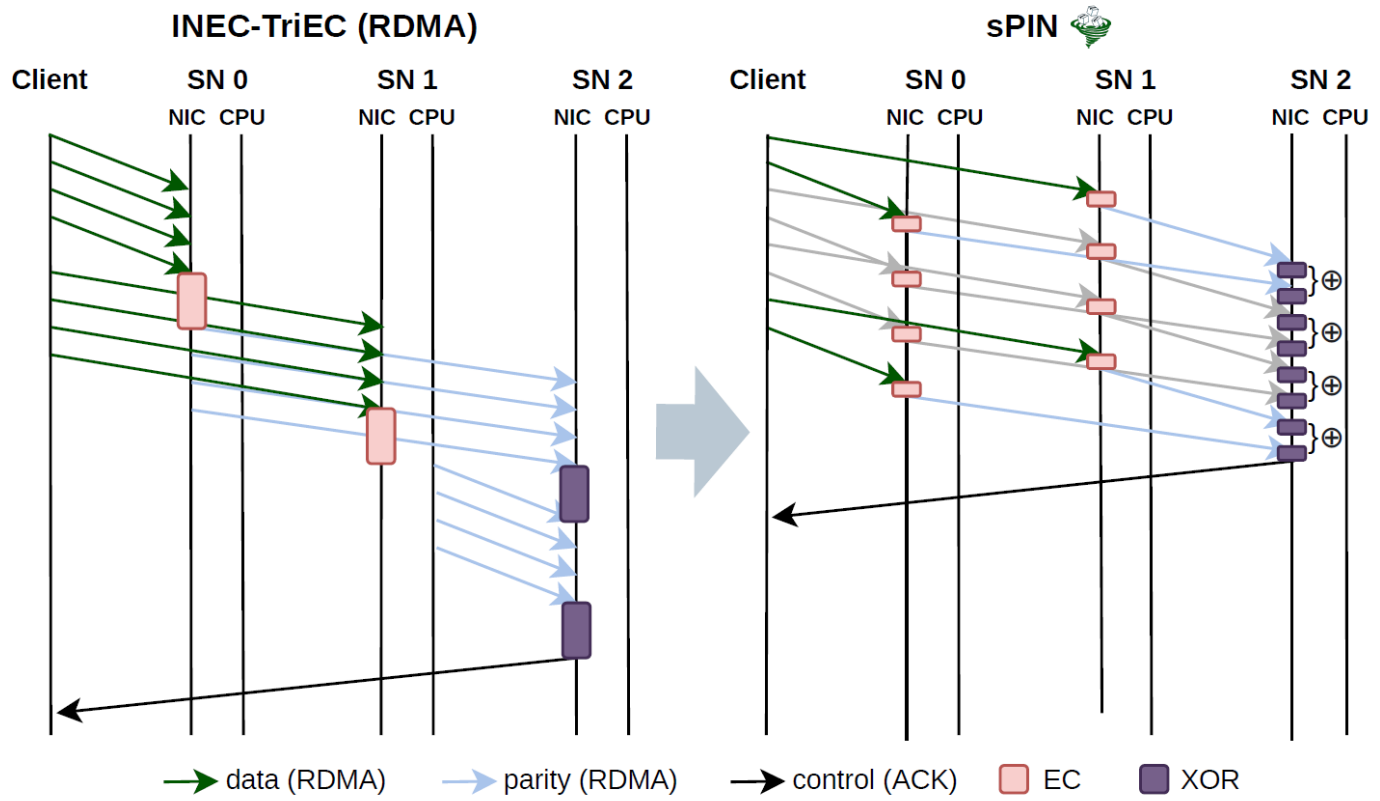
Request validation



Erasure coding







Simple write latency

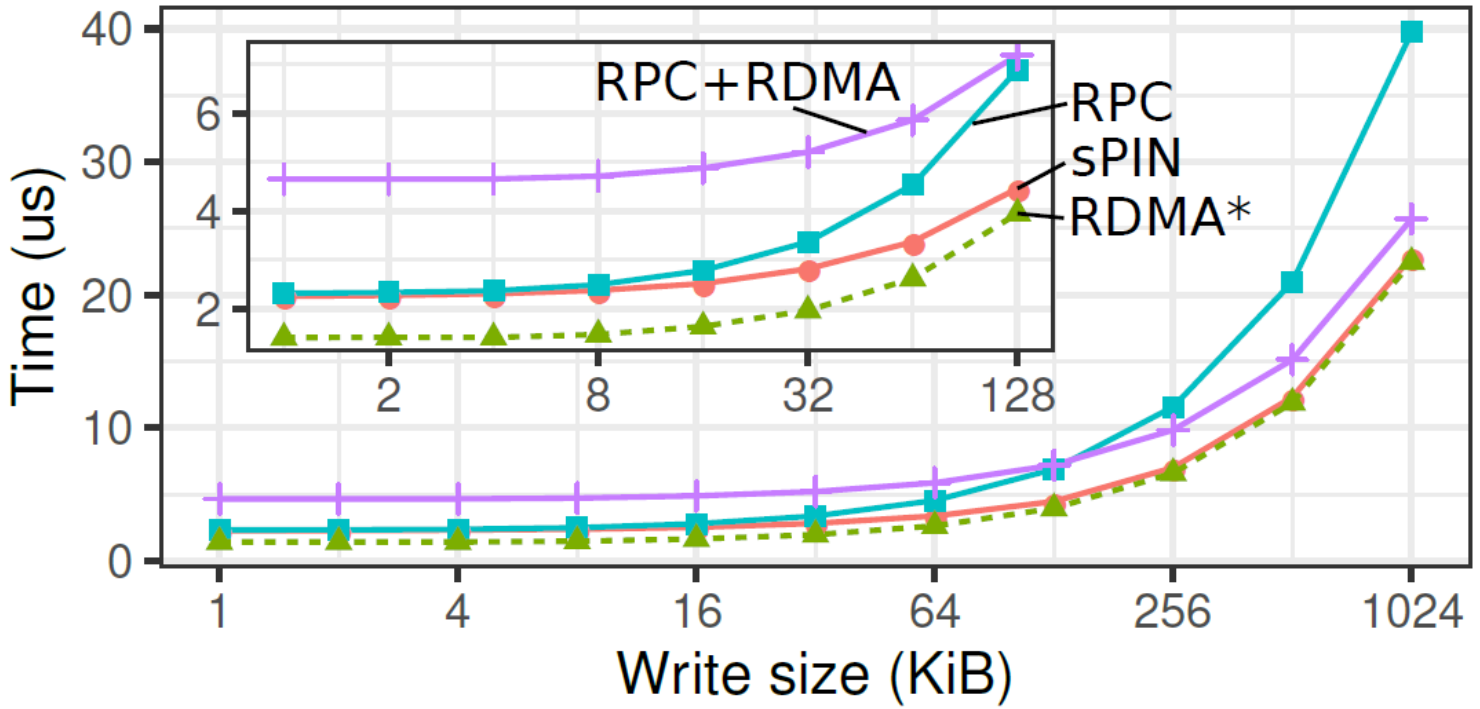


Figure 8: Write latencies with different protocols and write sizes. RDMA writes are reported as baseline and do not implement request validation.

Writes with erasure coding

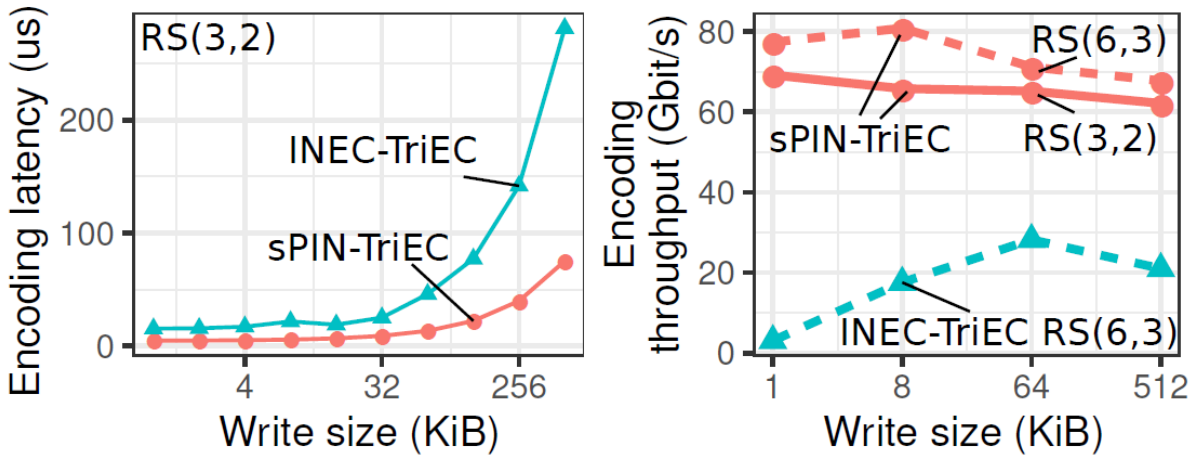


Figure 12: Encoding latency throughput.

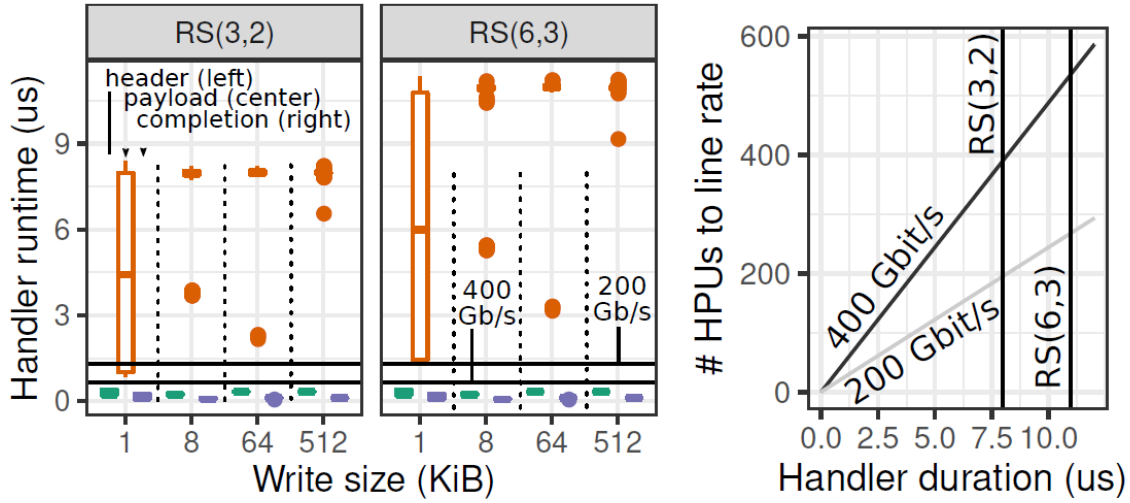


Figure 13: Left: handler running times for RS(3,2) and RS(6,3). Right: HPUs needed to sustain 400 Gbit/s and 200 Gbit/s (with 1 KiB packets) vs average handler duration.

Writes with replication

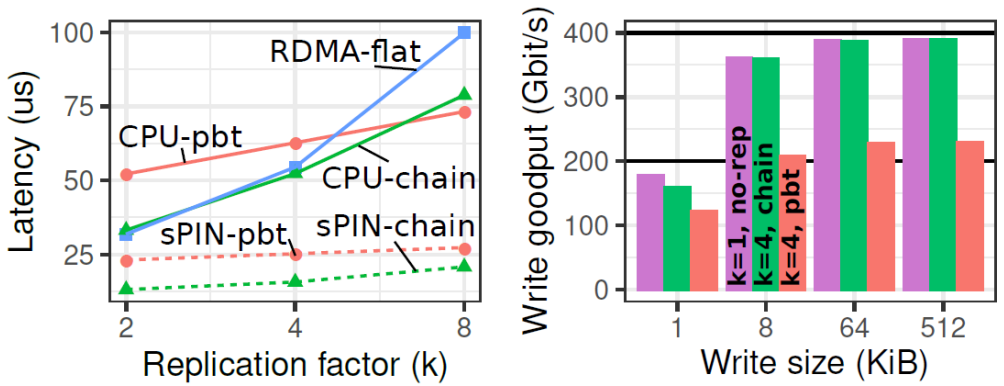


Figure 10: Left: 512 KiB write latency for different replication factors (k). Right: goodput of sPIN-accelerated writes.

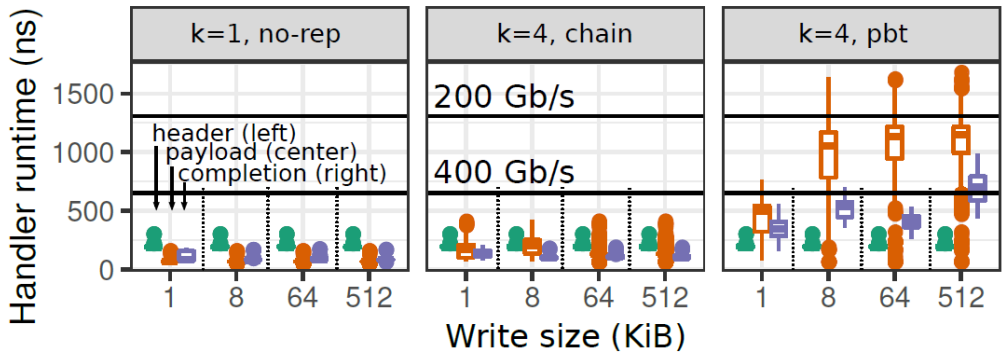


Figure 11: Left: 512 KiB write latency for different replication factors factor. Right: replicated writes goodput.