SC25 Tutorial: In-Network Computing with SmartNICs for HPC Applications

Presenters: Jeffrey Young, Elie Kfoury, Richard Graham, Oscar Hernandez, Antonio J. Peña, Jorge Crichigno, Mariam Kiran, Aaron Jezghani

Audience Survey

- Please follow presenter instructions to take a short survey on SmartNIC usage!

Tutorial Agenda

Agenda (Timing)

Time (CST)	Topic	Presenters
1:30 - 1:40	Introduction and Attendee Survey	
1:40 - 2:00	SmartNIC and DPU Background and Overview	Jeff, Elie
2:00-2:20	SmartNIC Applications and Use Cases	Jeff, Oscar
2:20 - 3:00	SmartNIC SW Infrastructure – HPC Programming Approaches: MPI and OpenMP Offload	Rich, Toni
3:00 - 3:30	**BREAK**	
3:30-4:15	SmartNIC SW Infrastructure – APIs and Frameworks: DPA Programming, DOCA and P4	Rich, Elie, Jorge
4:15-5:00	Hands on with DOCA and P4, DPA examples	All
4:45	Tutorial Survey / Wrap-up	

See the updated agenda at **https://github.com/gt-crnch-rg/smartnic-tutorial-sc25**

Introduction

Tutorial Objectives

- Describe how SmartNICs can be used as HPC/AI accelerators
- Describe how users can benefit from acceleration engines in their current and future applications
- Describe how one can use one family of SmartNICs to accelerate applications and communication libraries
- Share application and programming experiences using SmartNICs with canned examples and some hands-on examples

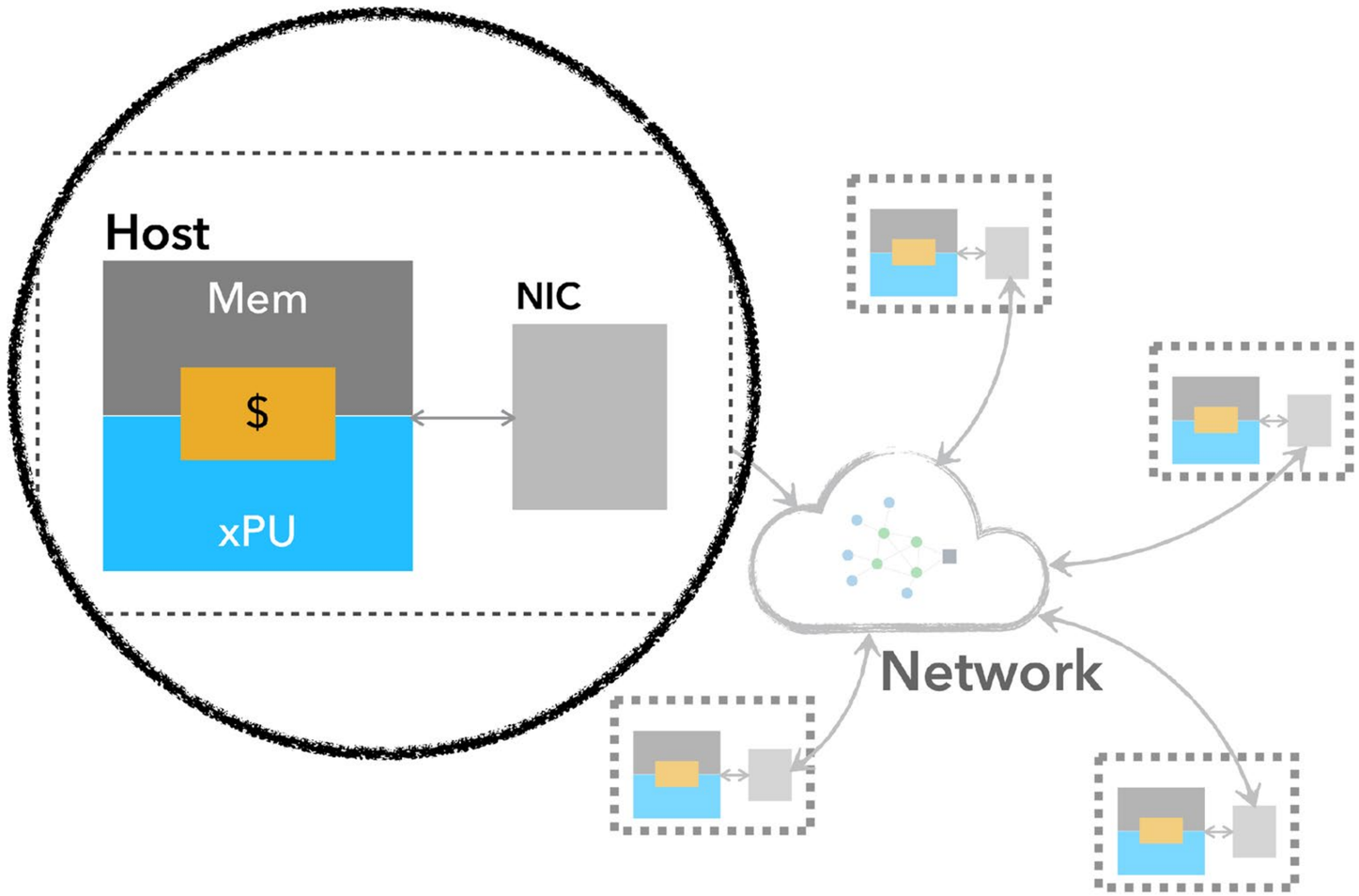
Anatomy of a supercomputer

The basic building block of a distributed-memory cluster or supercomputer is a node.

Each node includes a host, which is a processor (xPU) + memory hierarchy.

The host can communicate with other hosts via its NIC (network interface controller).

A network connects the nodes. The nodes may be arranged in some topology, which determines the network's carrying capacity and cost.



DPUs in modern clusters

The basic building block of a distributed-memory cluster or supercomputer is a node.

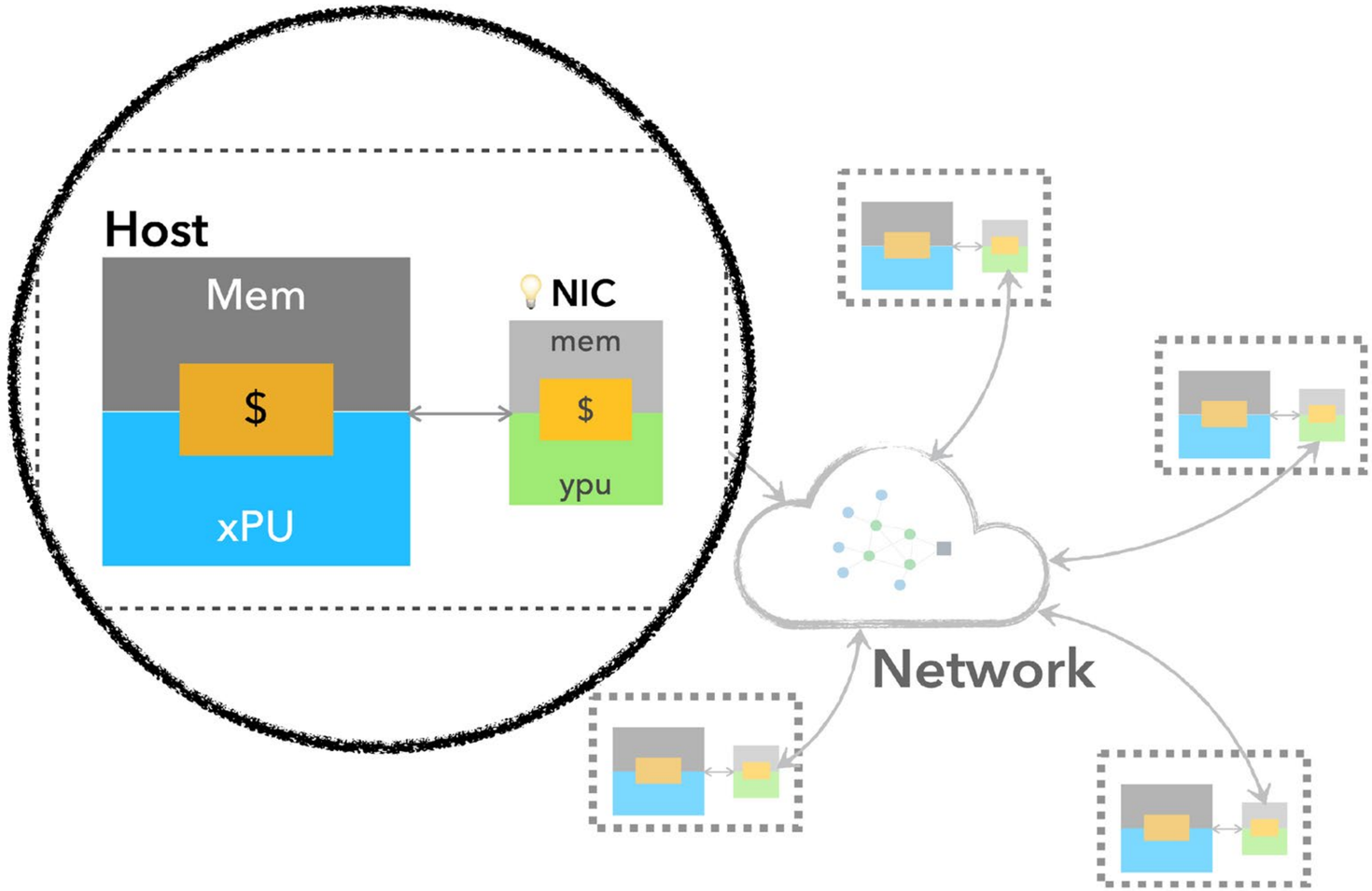
Each node includes a host, which is a processor (xPU) + memory hierarchy.

The host can communicate with other hosts via its NIC (network interface controller).

A network connects the nodes. The nodes may be arranged in some topology, which determines the network's carrying capacity and cost.

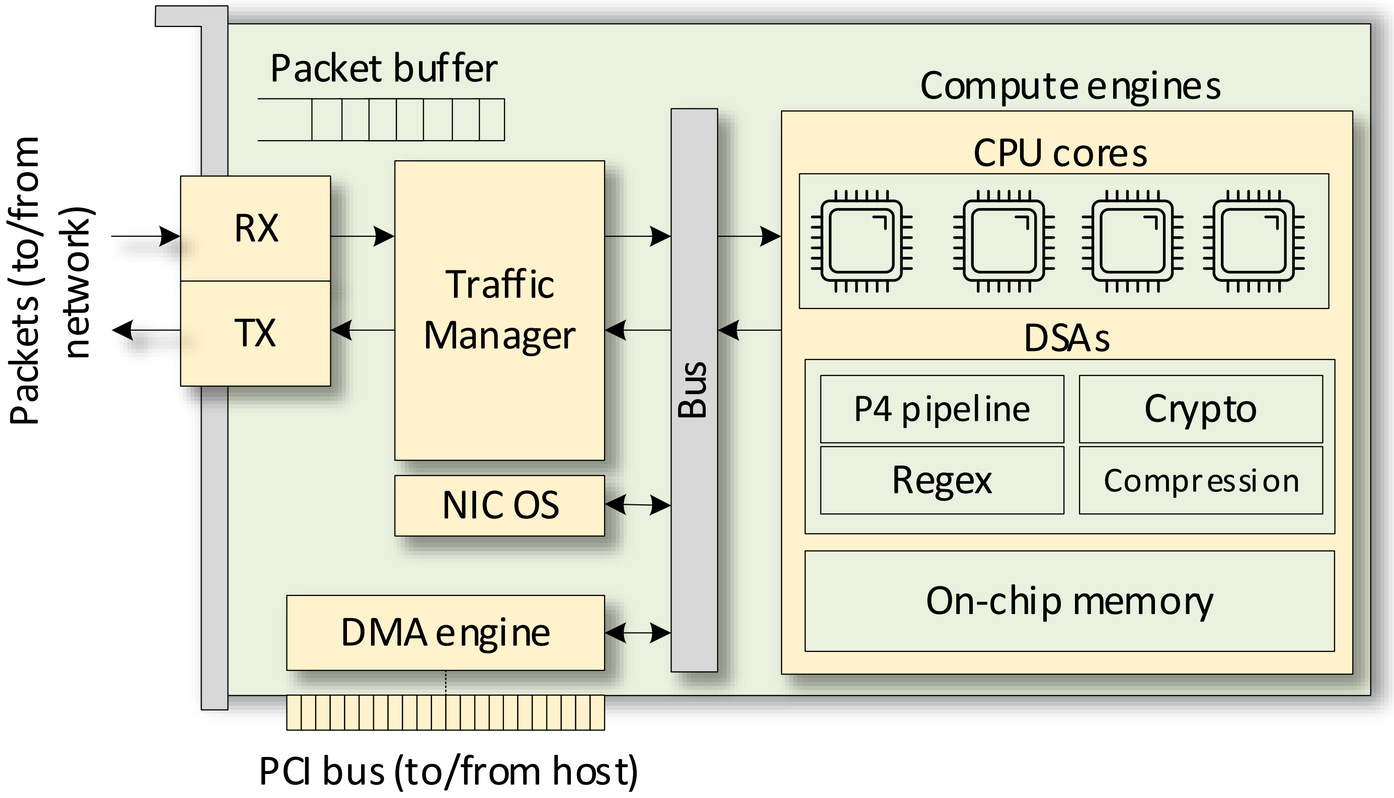
In a DPU, the NIC becomes "host-like" via the addition of processing (ypu), memory and other accelerations engines.

Node



Data Processing Units (DPUs)

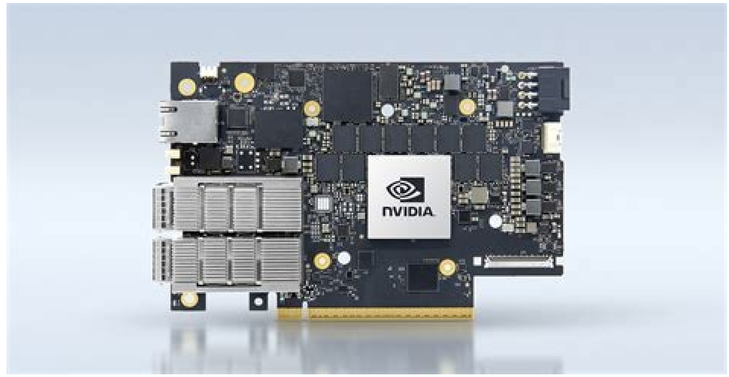
- DPUs use Domain Specific Accelerators (DSAs) to offload processing from general purpose CPUs
  - Programmable packet processing pipeline, regular expression, encryption/decryption, etc.
  - The DSAs are typically implemented on ASIC or FPGAs
- DPUs also include general-purpose CPU cores for managing the system

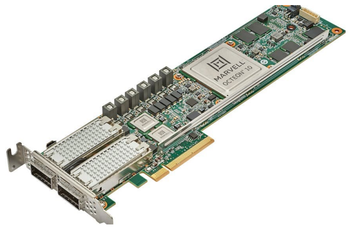


SmartNICs and Data Processing Unit (DPU) Background

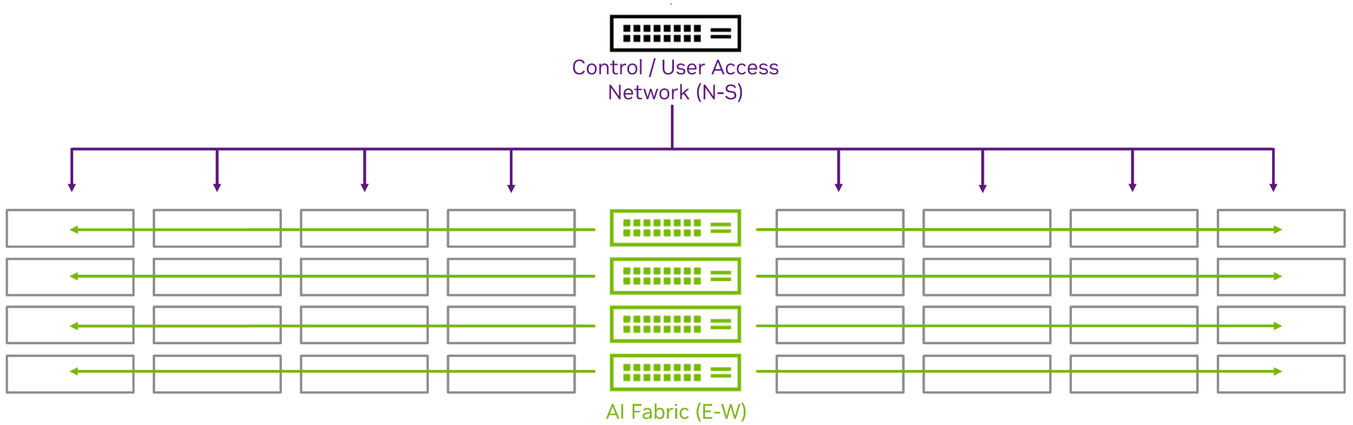
SmartNICs Available Today

- NVIDIA – BlueField family of Data Processing Units (ARM CPU)
- AMD – Pensando and Alveo FPGA
  - Mangoboost
- Intel Infrastructure Processing Units (IPUs)
- Intel FPGA variants (Silicom)
- Marvell Octeon (DPU with ARM based CPU)
- Research projects like sPIN



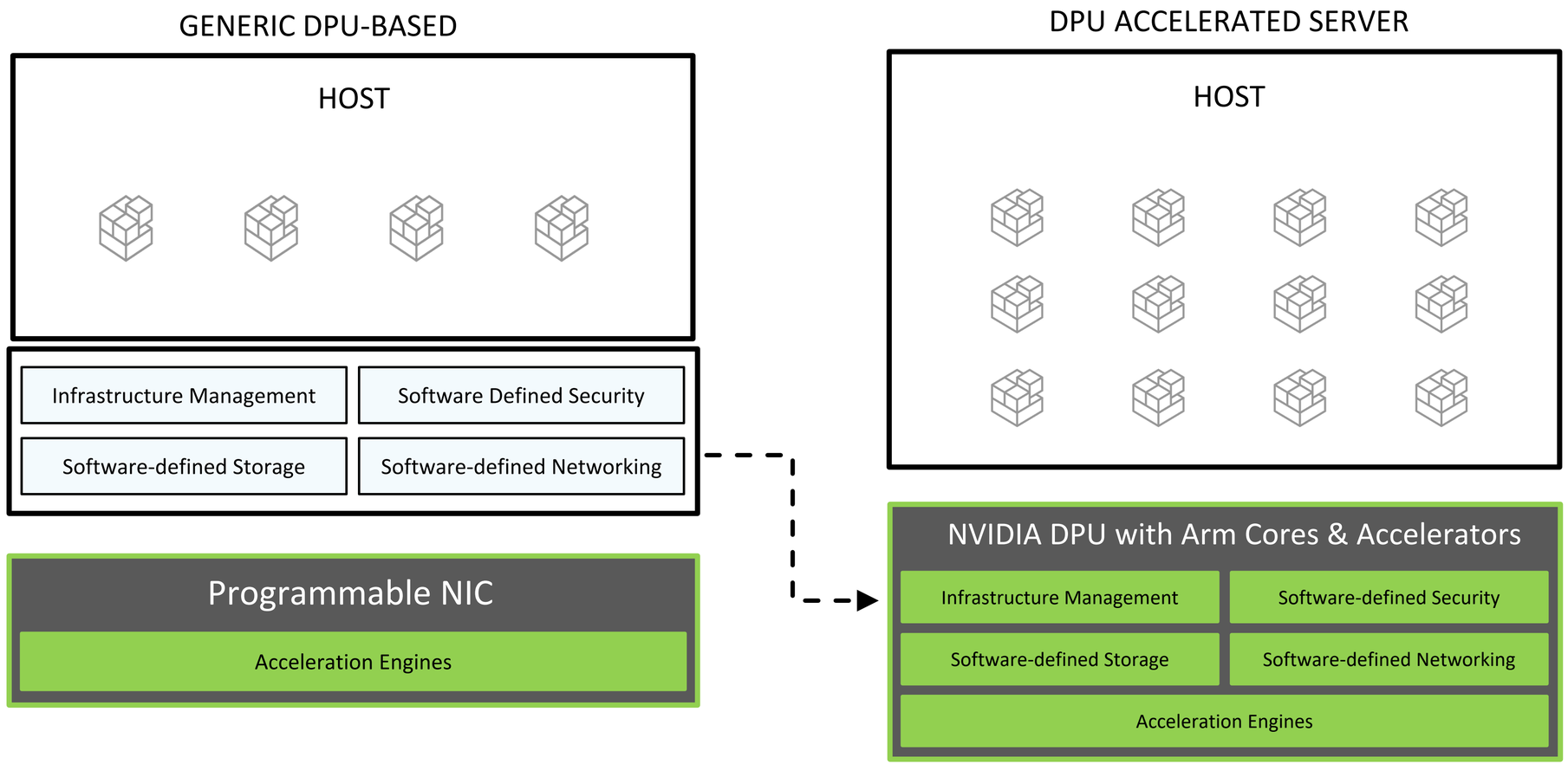


SmartNIC Networking Communication Types



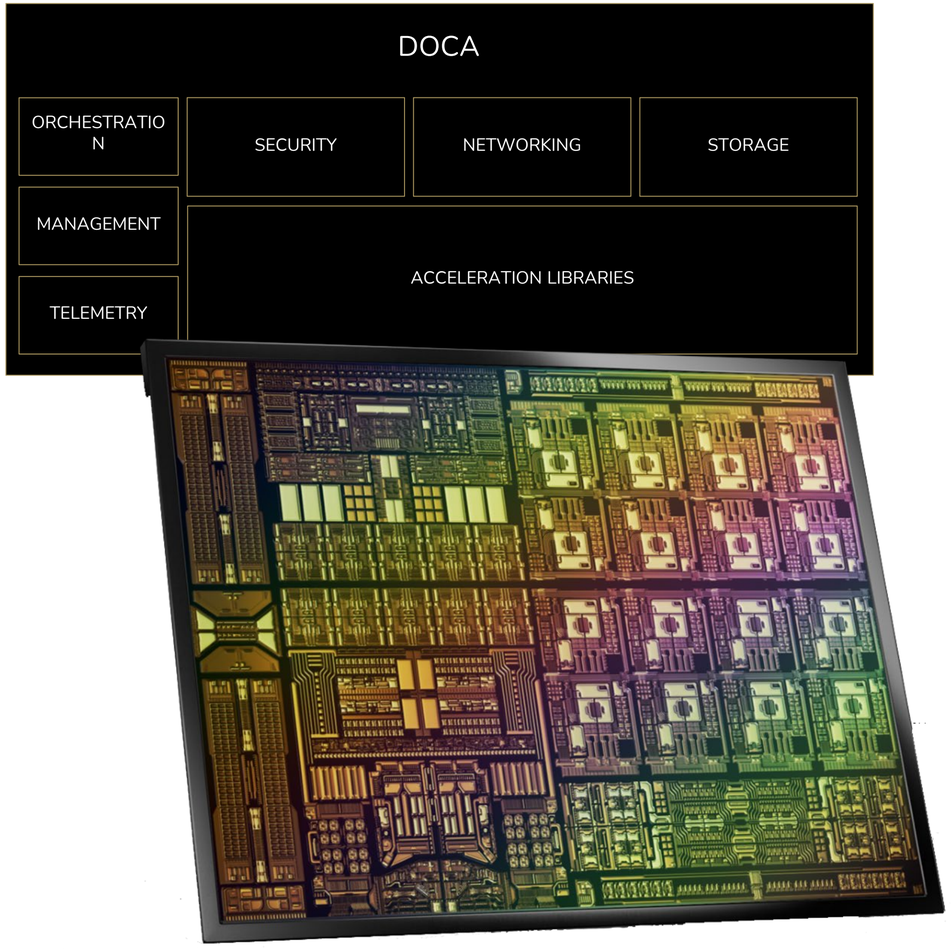
Control / User Access Network (N-S)		Computational Fabric (E-W)
Loosely-Coupled Applications, No Isolation Required		Tightly-Coupled Processes, Tenant Isolation Required
TCP (Low Bandwidth Flows and Utilization)		RDMA (High Bandwidth Flows and Utilization)
High Jitter Tolerance, average as key indicator		Low Jitter Tolerance, P99 is what matters
Heterogeneous Traffic, Statistical Multi-Pathing		Bursty Network Capacity, Predictive Performance
Low fabric utilization, forgiving of link failures		High link utilization, high penalty for failure and slow recovery

NVIDIA’s BlueField Data Processing Unit

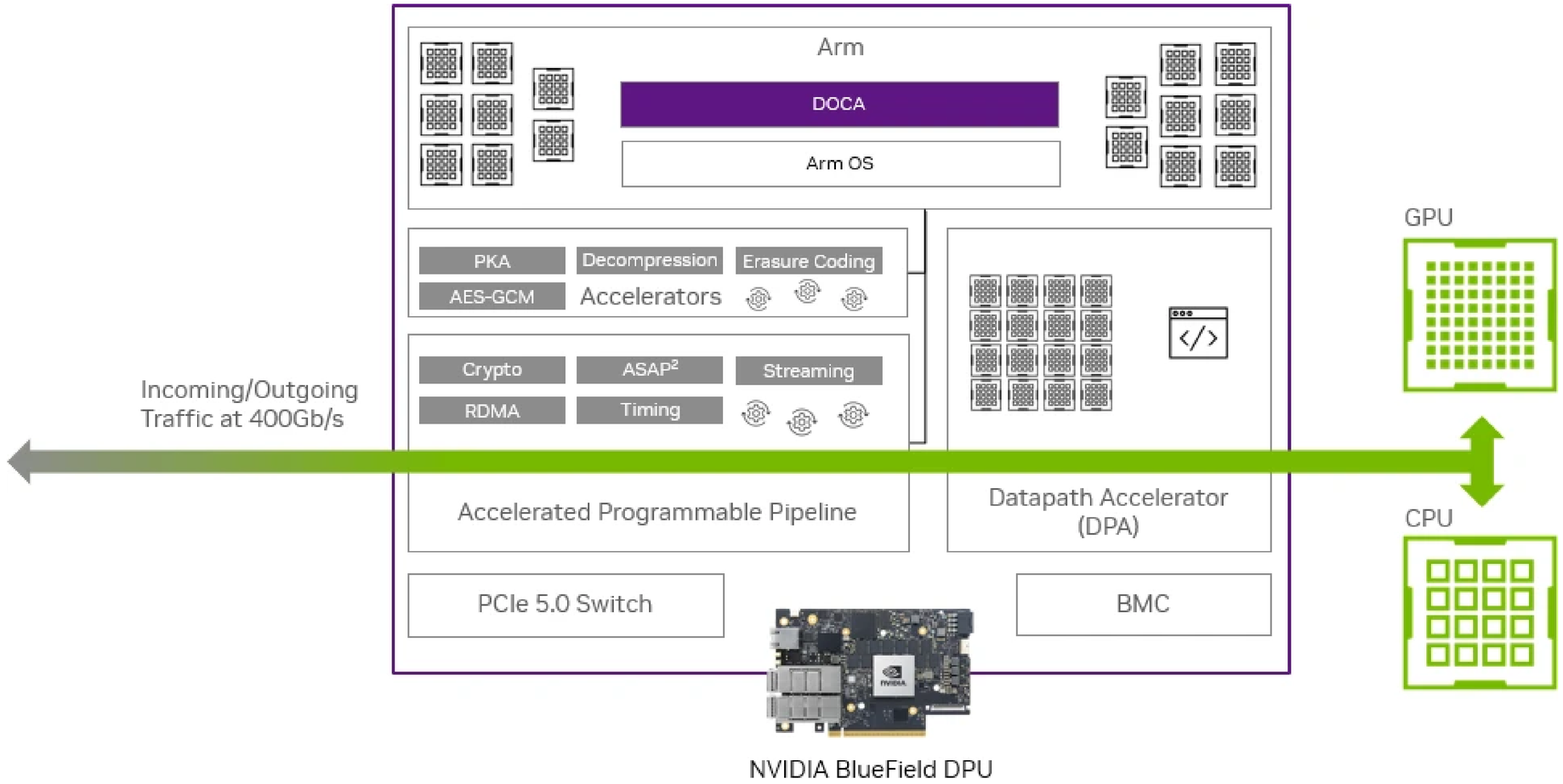


NVIDIA BlueField Specifications

	BlueField-2	BlueField-3
Network Bandwidth	200Gb/s	400Gb/s
RDMA max msg rate	215Mpps	370Mpps
Compute Cores	8	16
Compute	SPECINT2K6: 70	SPECINT2K6: 350
Memory Bandwidth	17GB/s	80GB/s
NVMe-OF	10M IOPs @ 4KB	18M IOPs@ 4KB
NVMe SNAP	5.4M IOPs @ 4KB	10M IOPs @ 4KB



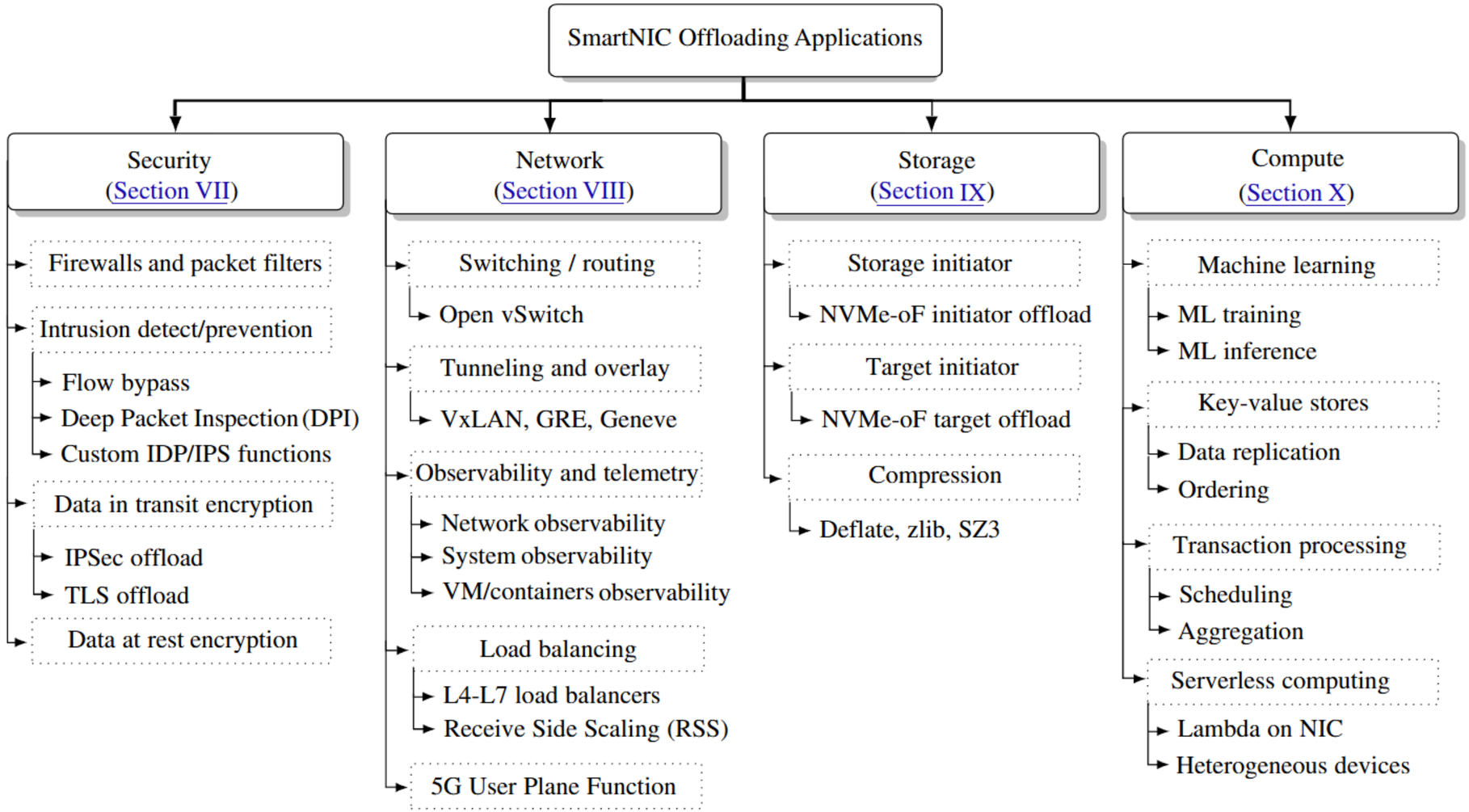
BlueField-3 Architecture



- The BlueField-3 includes programmable pipelines (P4), datapath accelerators (DPAs), Arm cores, and DSAs
- Different accelerators match better to north-south (Arm cores) or east-west communication (DPAs, DSAs)

Surveys on SmartNICs and DPUs

- DPUs have been used to offload a variety of applications and infrastructure functions



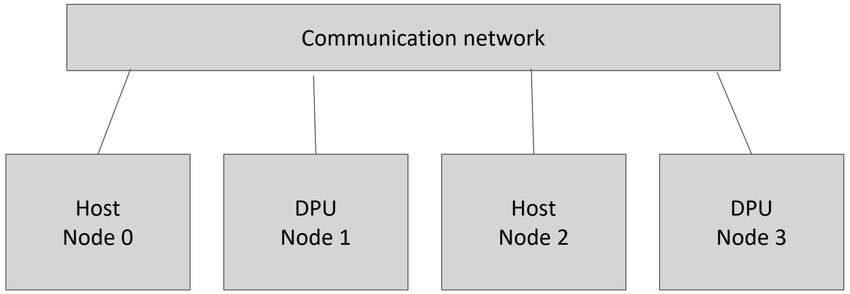
Kfoury, Elie F., et al. "A comprehensive survey on SmartNICs: Architectures, development models, applications, and research directions." *IEEE Access* (2024).

DPU Programming Models and Frameworks

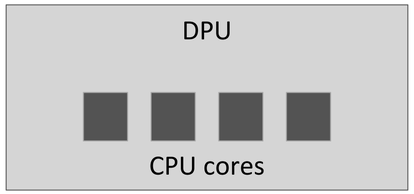
High-level HPC Programming Models

Three types of HPC programming models are available to program DPUs:

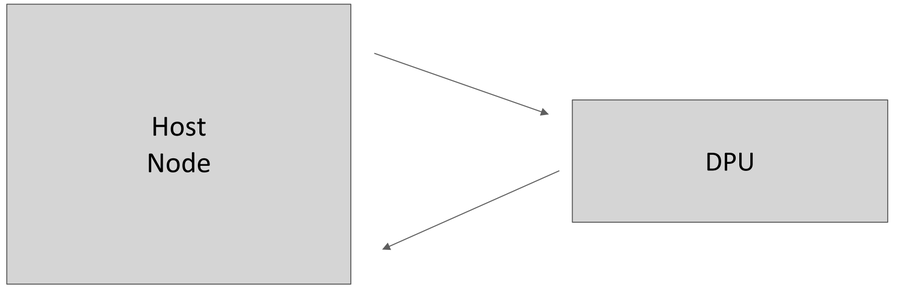
Distributed and distributed-shared memory
- **MPI – send/receive based**
- OpenSHMEM – put/get based



Shared memory
- OpenMP (shared memory)

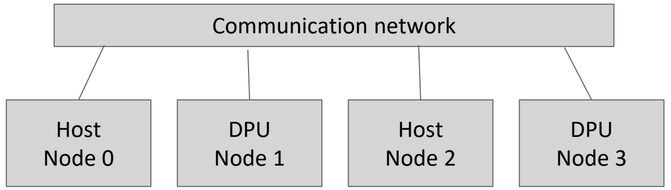


Accelerator offloading
- MPI, OpenSHMEM
- **OpenMP offload**



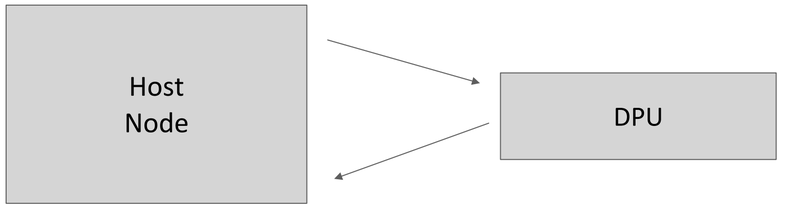
"Offloading network operations"

Distributed Programming Model



- Ranks are mapped to compute nodes and DPUs.
- A DPU is treated as another node in the system and its rank can be part of a communicator or sub communicator.
- DPU ranks will have different computing capabilities
- Potential use-cases:
  - Application is decomposed to use DPUs for computation or I/O
    - Examples:
      - DPUs cores are used for computation
      - DPUs handle the I/O ranks of an application

Accelerator Offload Model



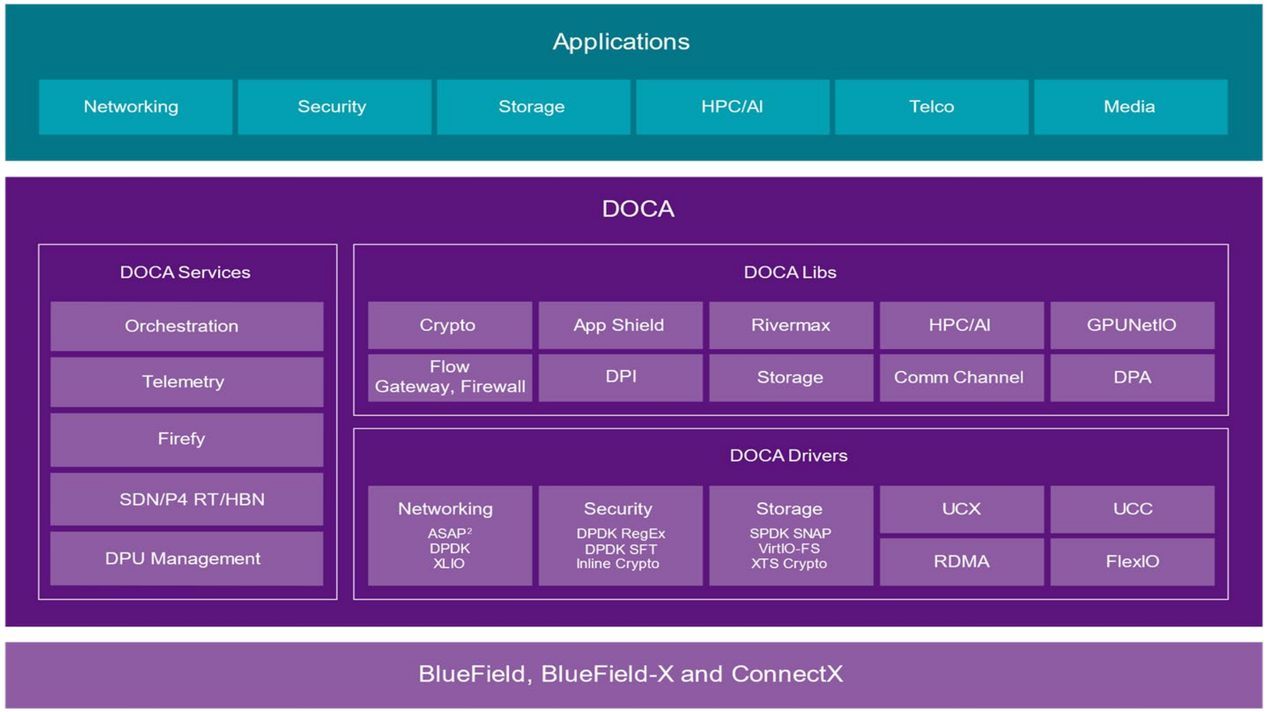
"Offloading network operations"

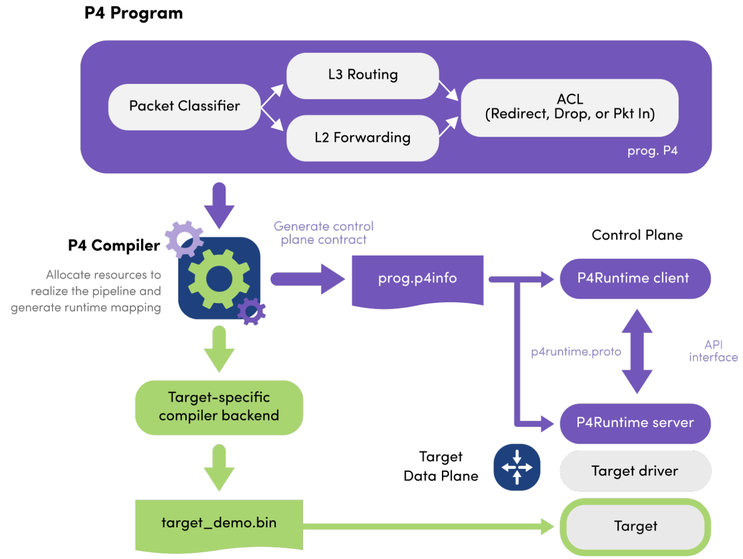
Two model views:

- Programming model exposes the accelerator
  - When accelerator is exposed
  - Program explicitly interacts with the accelerator, such as with an MPI library implementation

- Programming model does not expose the accelerator
  - Accelerator is used by services the application uses such as MPI, OpenSHMEM and storage
  - Services explicitly interact with the accelerator

Other Frameworks for DPUs/DPAs





DOCA
- NVIDIA-specific SDK for application developers
- Supports Data Plane Development Kit (DPDK), RDMA, compression, etc.

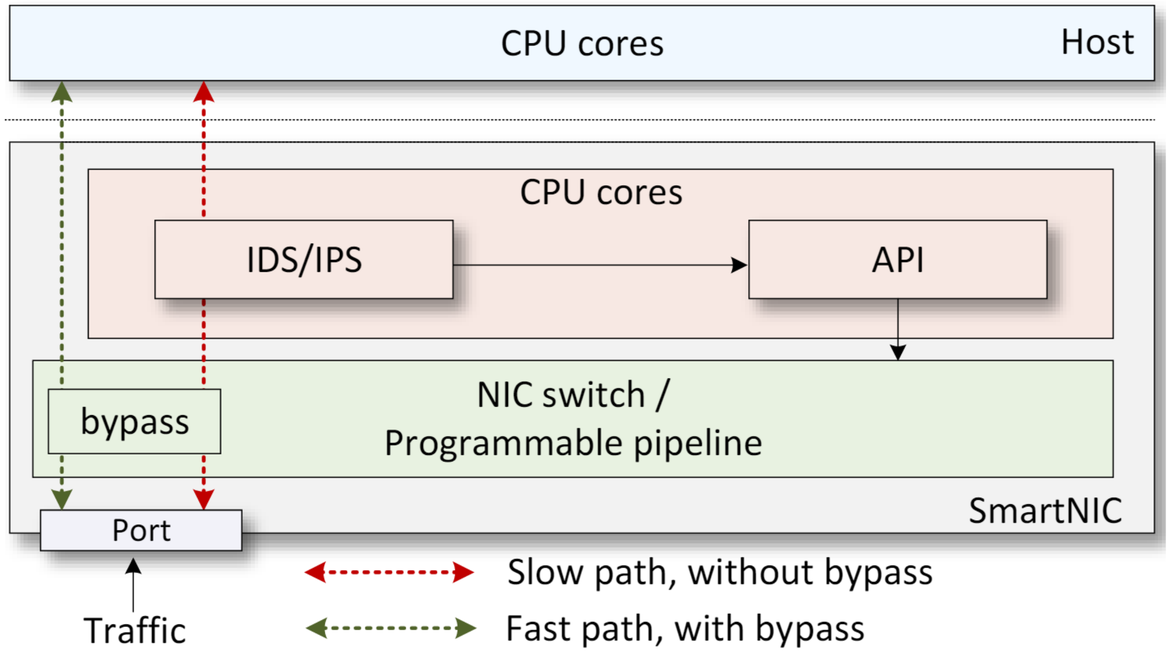
P4
- Provides a vendor-neutral specification for packet processing and manipulation

SmartNIC Applications and Use Cases

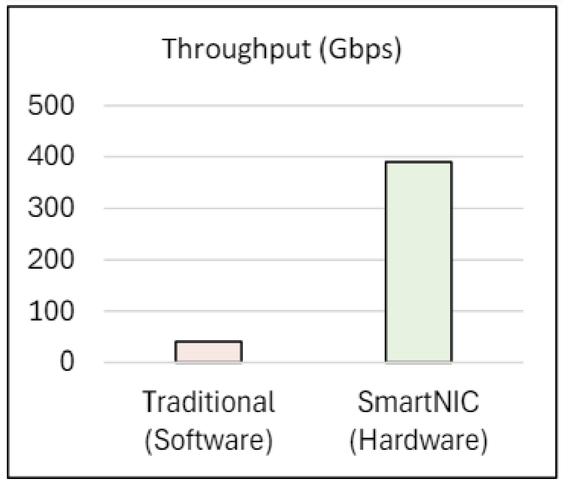
Accelerating IDS/IPS Functions

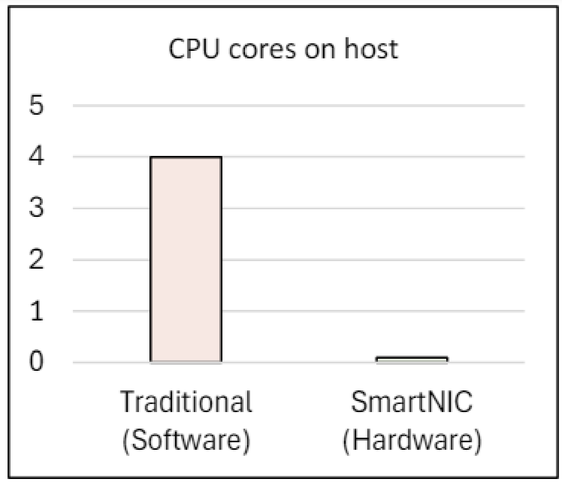
Intrusion Detection/Prevention System (IDS/IPS) functions can be offloaded to DPUs

- Traffic bypass, Deep Packet Inspection (DPI), signature matching, etc.



Suricata bypass function

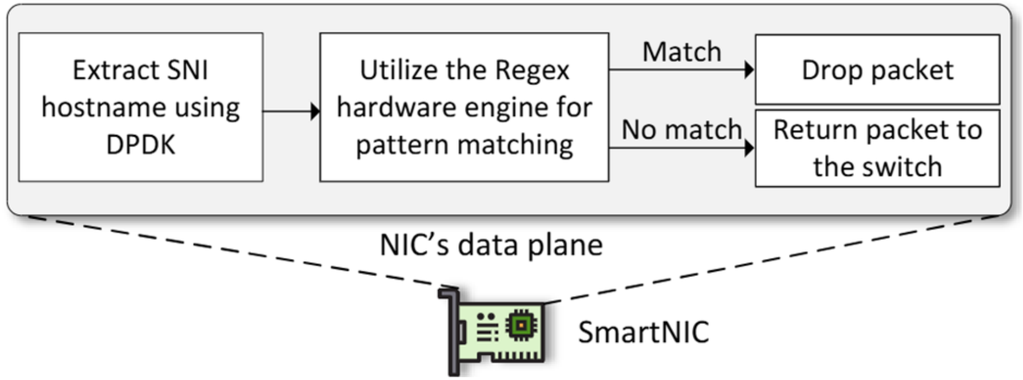


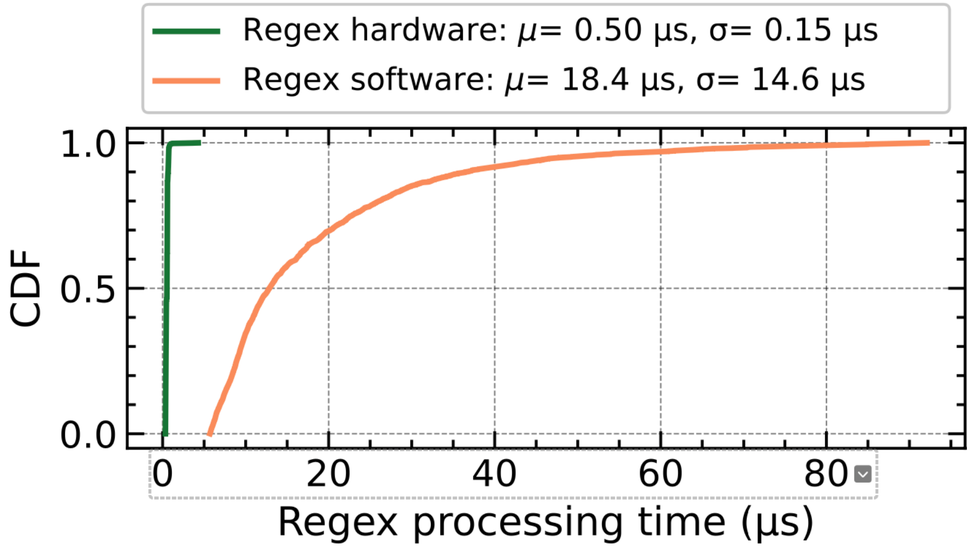


M. Gonen. Accelerating the Suricata IDS/IPS With NVIDIA BlueField DPUs. Available: https://tinyurl.com/ys8n6mmz

Deep Packet Inspection Offload

DPUs are accelerating pattern matching through regular expression DSAs





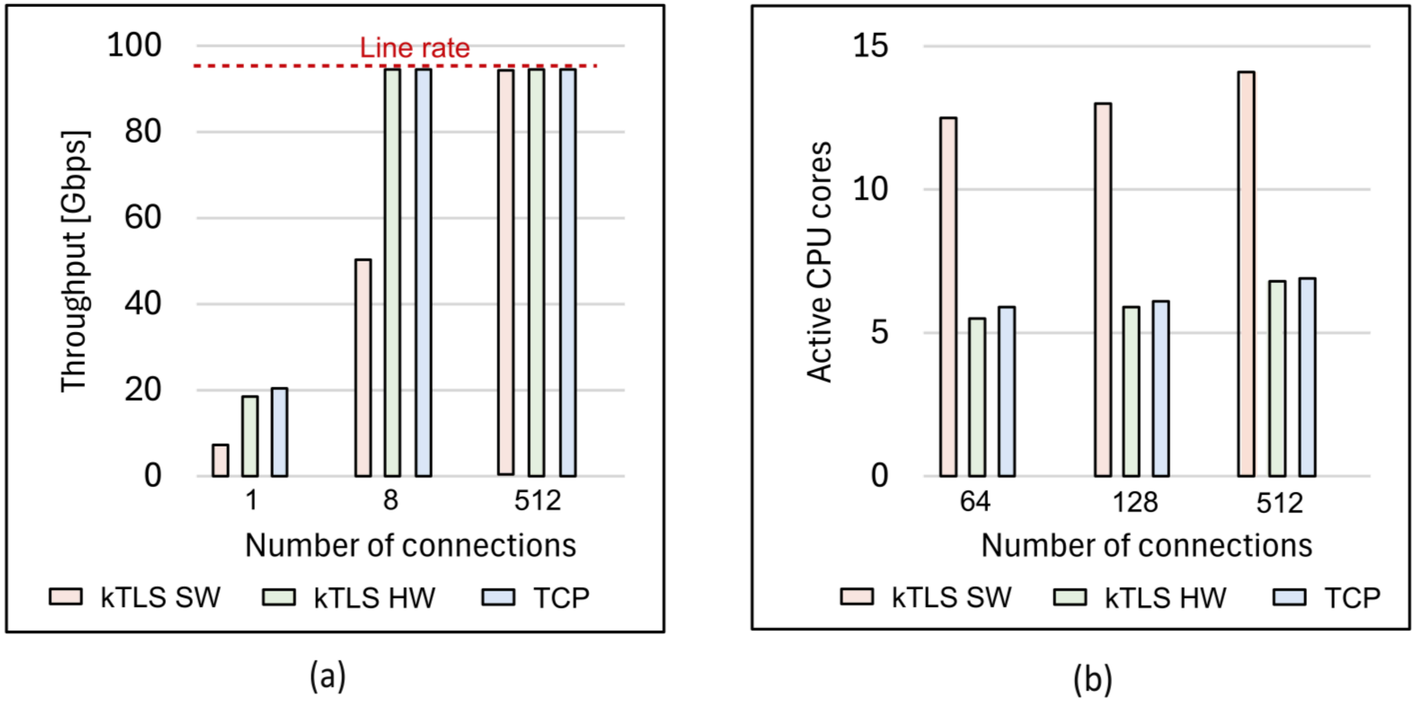
Mazloum, A., et al. "Toward Fingerprinting Encrypted C2 Traffic in the Data Plane.“, in submission.

Cryptographic Functions Offload (1/2)

SmartNICs support hardware accelerators for offloading cryptographic operations:

- Symmetric encryption
- Asymmetric encryption
- True Random Number Generator (TRNG)

TLS hardware offload



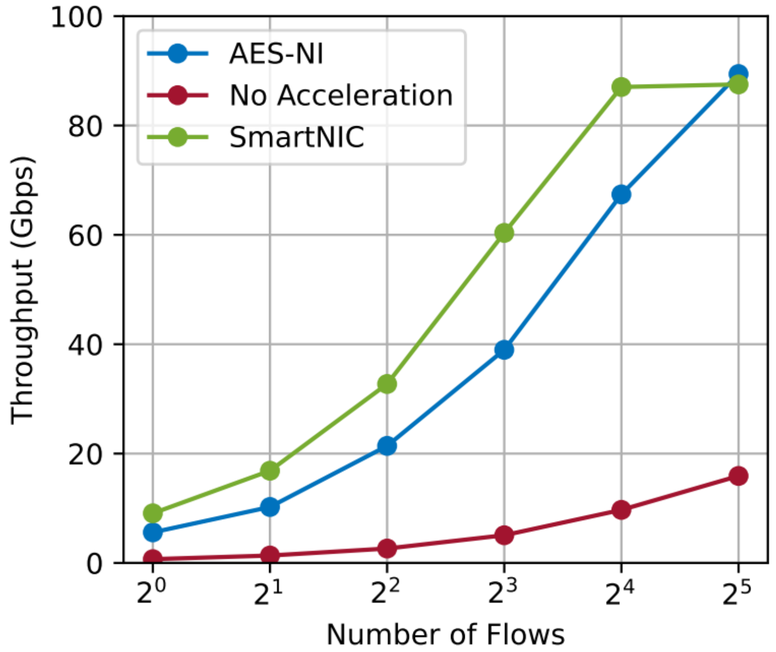
B. Tuaf, T. Gilboa, T. Toukan. kTLS Offload Performance Enhancements for Real-life Applications. Available: https://tinyurl.com/bdewpxh5

Cryptographic Functions Offload (2/2)

SmartNICs support hardware accelerators for offloading cryptographic operations:

- Symmetric encryption
- Asymmetric encryption
- True Random Number Generator (TRNG)
- Etc.

Large data transfers encryption (TLS offload)

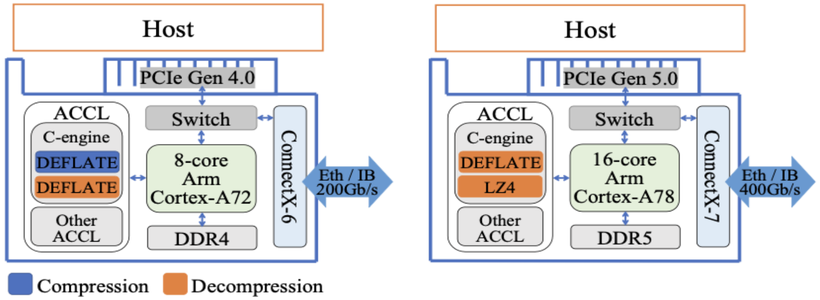


Kfoury, E., et al. "Evaluating Encryption Overhead and Offload Techniques in High-Speed Data Transfers.“, 2025 IEEE Black Sea Conference on Communications and Networking (BlackSeaCom) https://doi.org/10.1109/BlackSeaCom65655.2025.11193901.

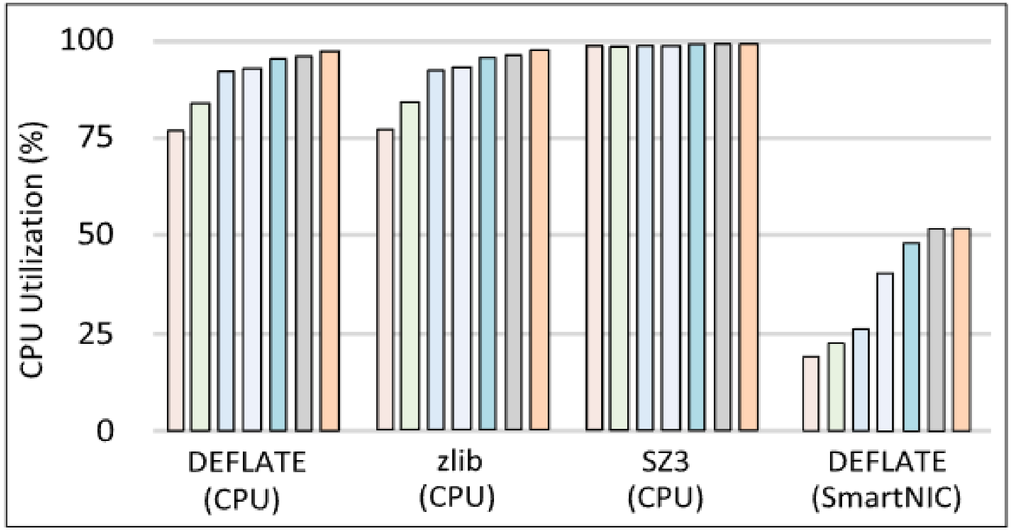
(De)Compression Offload

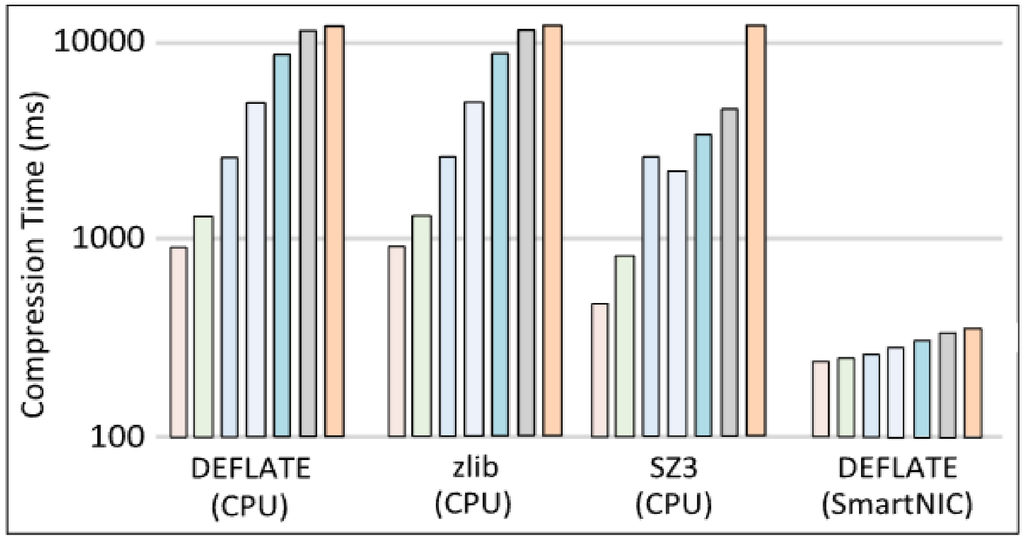
SmartNICs can be used to accelerate file compression and decompression operations

- The experiment shows results for compression algorithms over seven datasets
  - Speedups in this work made use of DSAs on the BlueField platform



(a) BlueField-2 DPU Arch. (b) BlueField-3 DPU Arch.

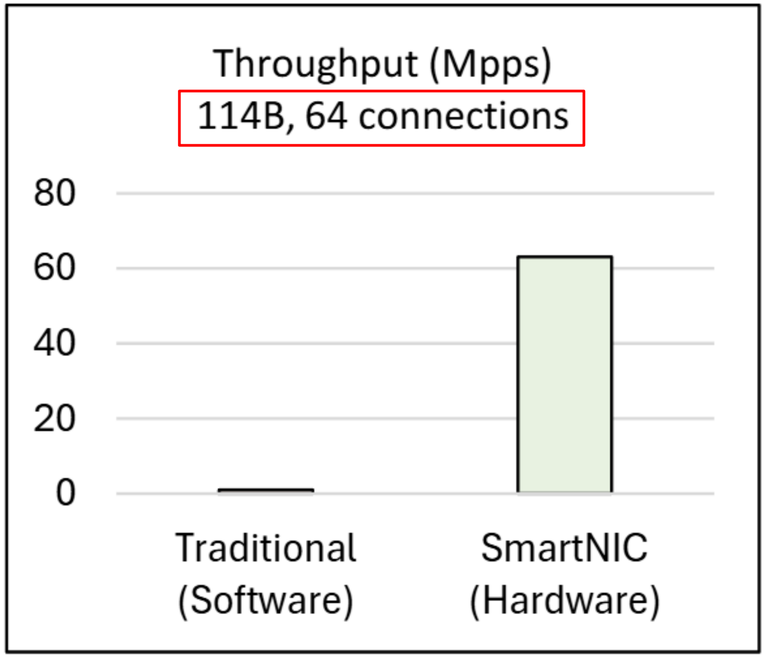


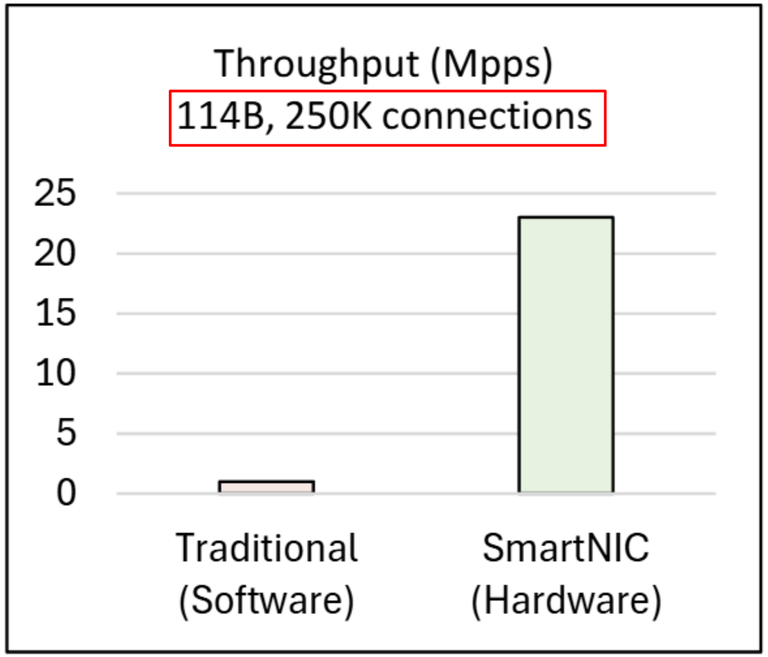


Li, Yuke, et al. "Characterizing lossy and lossless compression on emerging BlueField DPU architectures." 2023 IEEE Symposium on High-Performance Interconnects (HOTI). IEEE, 2023.

Tunneling

SmartNICs are accelerating tunneling operations (VxLAN + Connection Tracking)





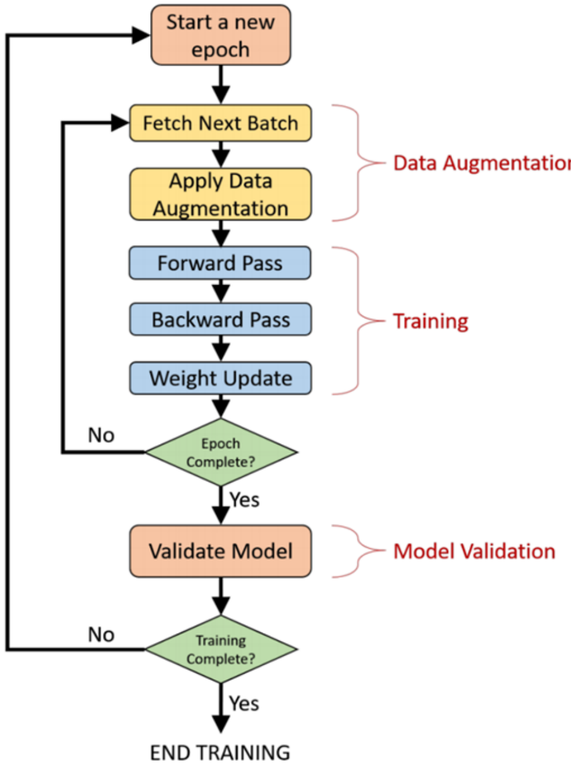
Idan Burstein. "Nvidia Data Center Processing Unit (DPU) Architecture." 2021 IEEE Hot Chips 33 Symposium (HCS)

Accelerating MPI and Deep Learning Applications with the DPU Technology

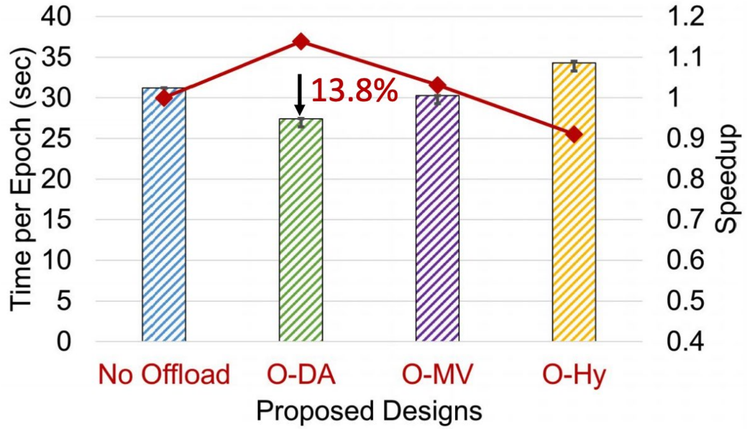
Nick Sarkauskas, Arpan Jain, Nawras Alnaasan, Tu Tran, Bharat Ramesh, Aamir Shafi, Hari Subramoni, and Dhabaleswar K. (DK) Panda

EXPLOITING DPUS FOR DEEP NEURAL NETWORK TRAINING

- **There are several phases in Deep Neural Network Training**
  - Fetching Training Data
  - Data Augmentation
  - Forward Pass
  - Backward Pass
  - Weight Update
  - Model Validation
- **Different phases can be offloaded to DPUs to accelerate the training.**



- **Speedup**
  - Single node: O-DA (13.8%) and O-MV (3.1%)
  - Multi-node: Achieves average 13.9% speedup on 1-16 nodes

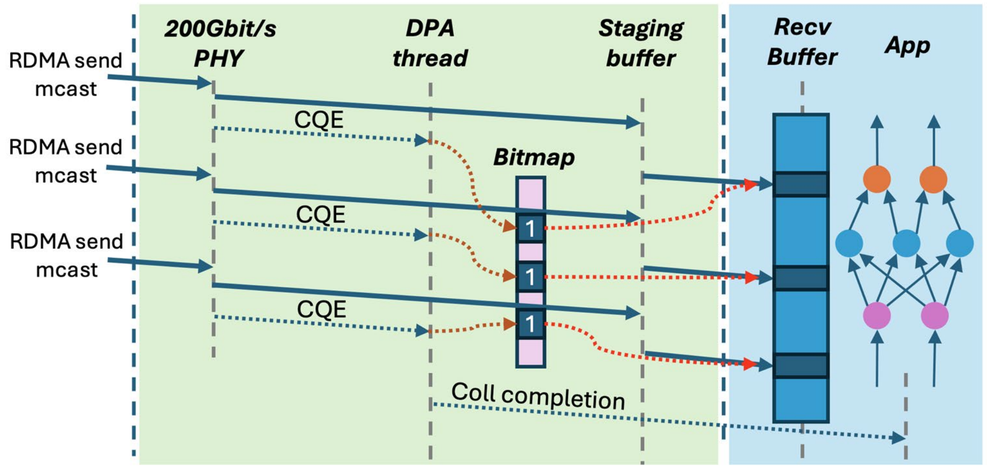


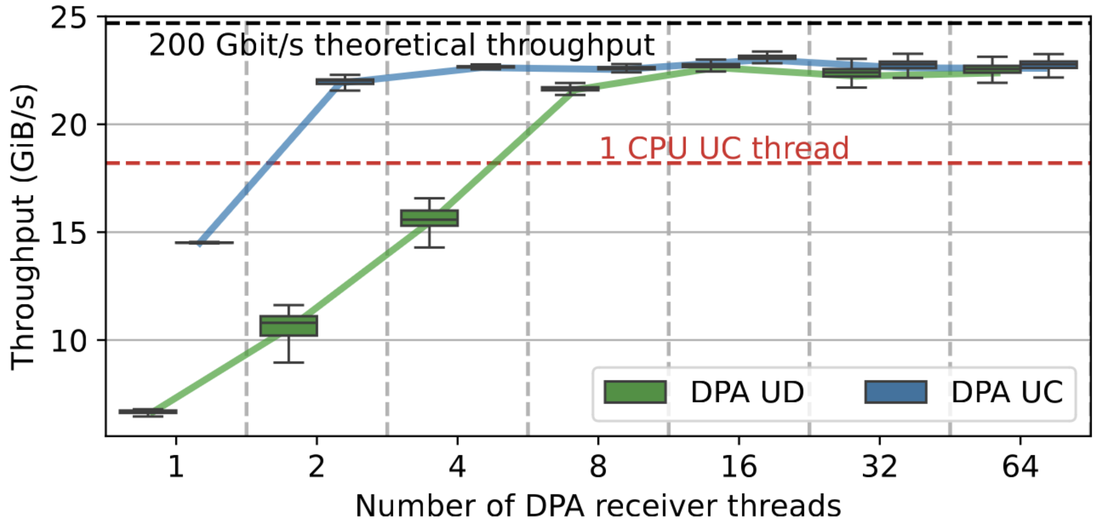
Single Node Experiments

Source: https://www.openfabrics.org/wp-content/uploads/2022-workshop/2022-workshop-presentations/101_DKPanda.pdf

DPA use case: AI Collectives Acceleration

- Fully Sharded Data Parallel (FSDP) training used for LLAMA 3 training
- FSDP relies on Allgather that can be accelerated with hardware RDMA multicast
- Multicast 4KB datagram processing introduces high CPU overheads at >= 200 Gbit/s
- DPA is an ideal offloading Multicast Send/Receive

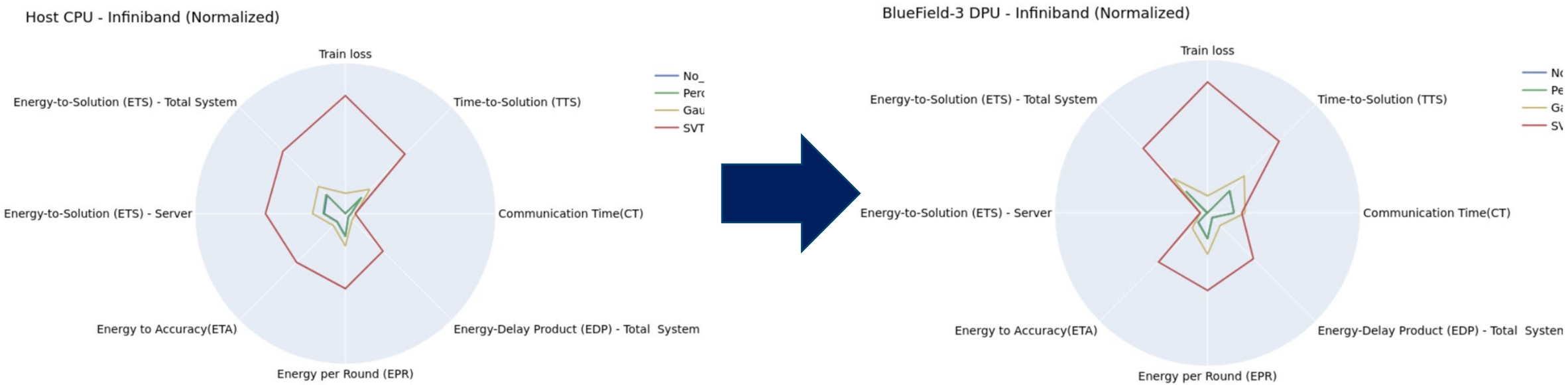




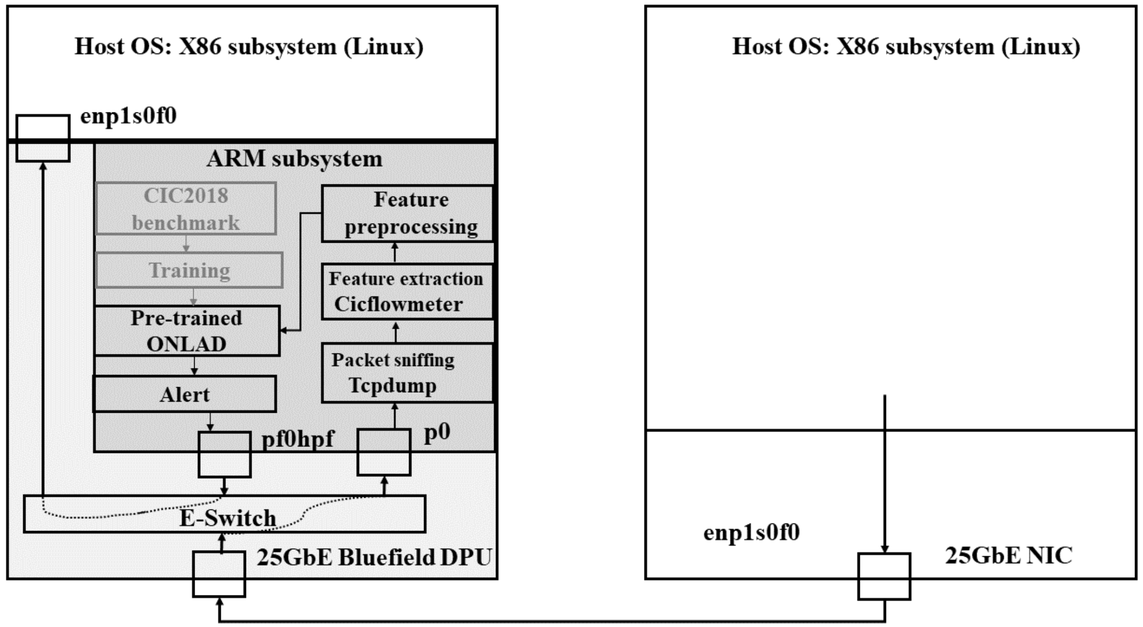
Mikhail Khalilov, Salvatore Di Girolamo, Marcin Chrapek, Rami Nudelman, Gil Bloch, and Torsten Hoefler, ***"Bandwidth-optimal, Fully-Offloaded Collectives"***, Supercomputing 2024.

SmartNIC-Accelerated Federated Learning Energy & Privacy Trade-offs

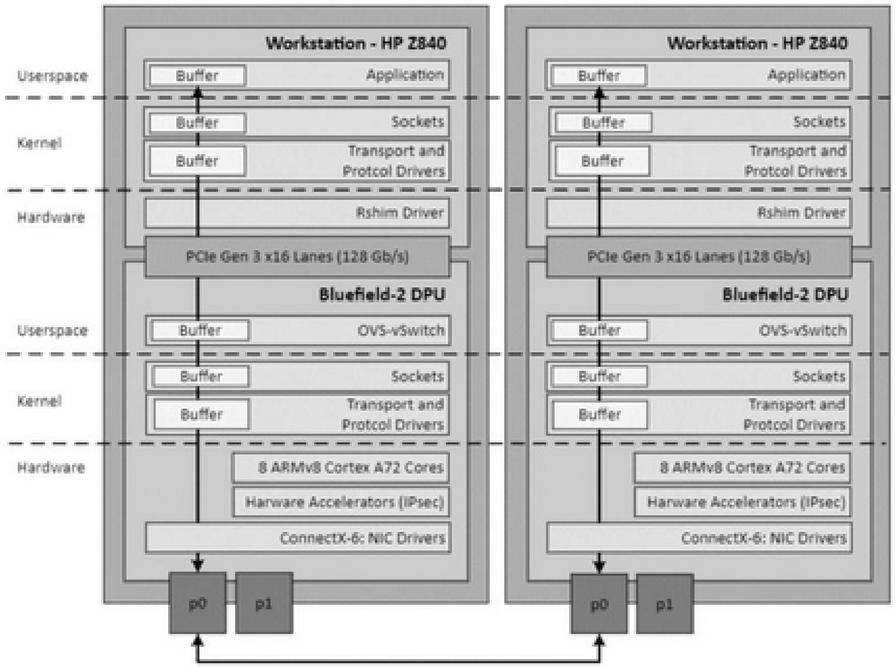
- BF3 reduces server-side energy use by up to 4.5× for Gaussian and Percentile methods.
- DPU favors lightweight privacy methods in energy-constrained settings.
- Setup:
  - FL Framework: NVFLARE
  - 5 clients – GH200
  - 1 server - BF-3
- Future work: accelerate compression and security



Security Related Research on DPUs



Man Wu, Hiroki Matsutani, and Masaaki Kondo. "ONLAD-IDS: ONLAD-Based Intrusion Detection System Using SmartNIC." In *2022 IEEE 24th Int Conf on High Performance Computing & Communications; (HPCC/DSS/SmartCity/DependSys)*, pp. 546-553. IEEE, 2022.



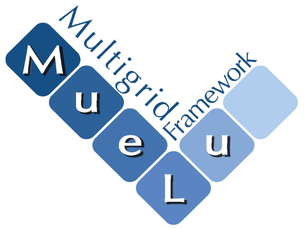
From Noah Diamon Scott Graham, and Gilbert Clark. "Securing InfiniBand networks with the Bluefield-2 data processing unit." In *International Conference on Cyber Warfare and Security*, vol. 17, no. 1; 2022

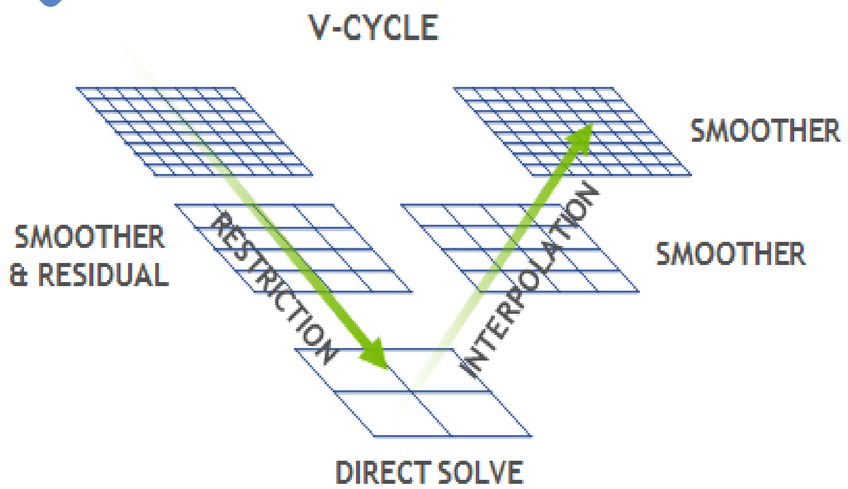
- Recent work has looked at using frameworks like DOCA to implement zero trust architectures, but the addition of processing cores and a full OS stack to the DPU allows for the design of sophisticated IDS and secure network stacks

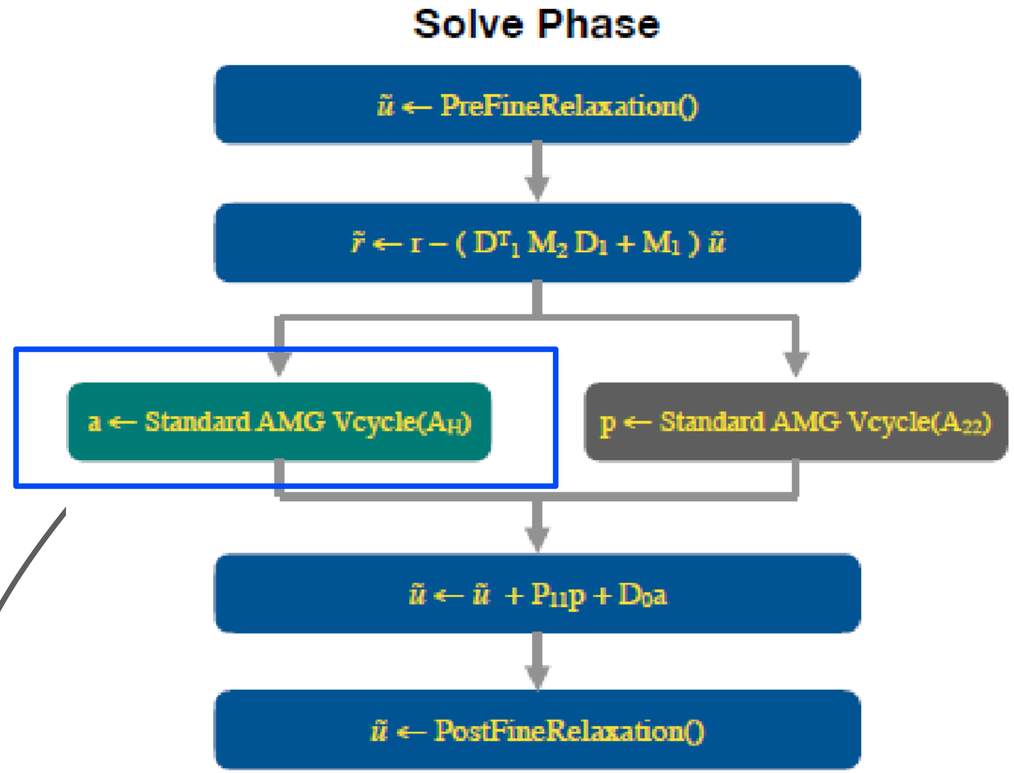
DPU Offload Work for HPC Libraries

HPC applications such as asymmetric multigrid can also potentially be offloaded to the DPU using MPI and and the onboard Arm cores.

- For example, Maxwell Equation Solvers as implemented in the MueLu library represents a compelling distributed and shared memory parallel algorithm. These implementations can be formulated using adaptive multigrid approaches and possible also parallelized to DPUs!
- Results show that this offload can result in both a small speedup and greenup (efficiency gain) for a DPU offload enabled algorithm.







Possible Offload Candidate

Bochev, Pavel B., et al. "An algebraic multigrid approach based on a compatible gauge reformulation of Maxwell's equations." *SIAM Journal on Scientific Computing* 31.1 (2008)

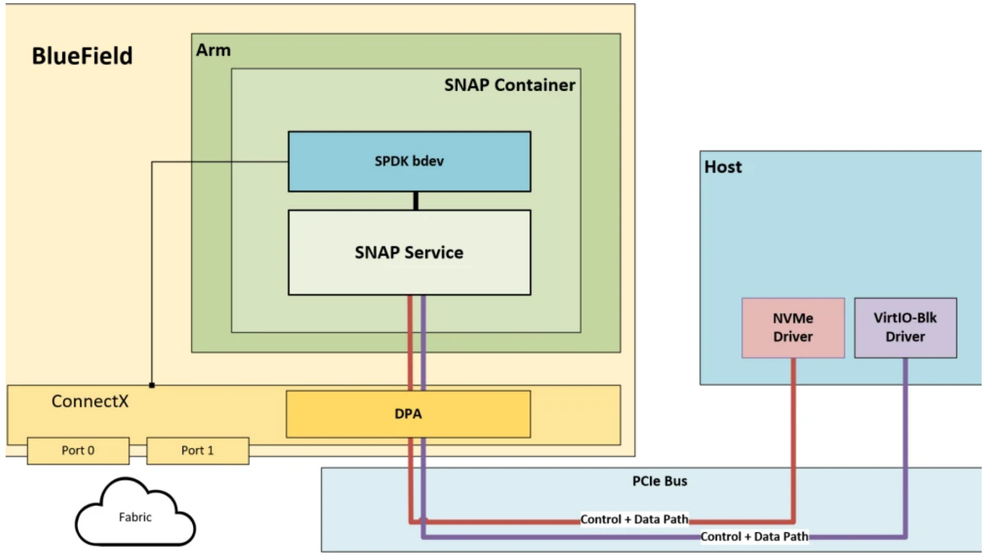
Karamati, et al. *work under submission*.

Offloading Storage with DPUs

DPU resources can be used to address practical problems with storage device access:

DPU offload for NVMe over Fabrics (VirtIO Block)

- NVMe natively supports namespace-attach over network transports (Verbs, TCP/IP)
  - Essentially, a network block device
  - However, NVMe relies on shared secret **per device, not per namespace!**
- NVIDIA SNAP can access NVMe devices securely from the DPU
- Enables on-demand, elastic storage without sacrificing security



https://docs.nvidia.com/networking/software/snap/index.html

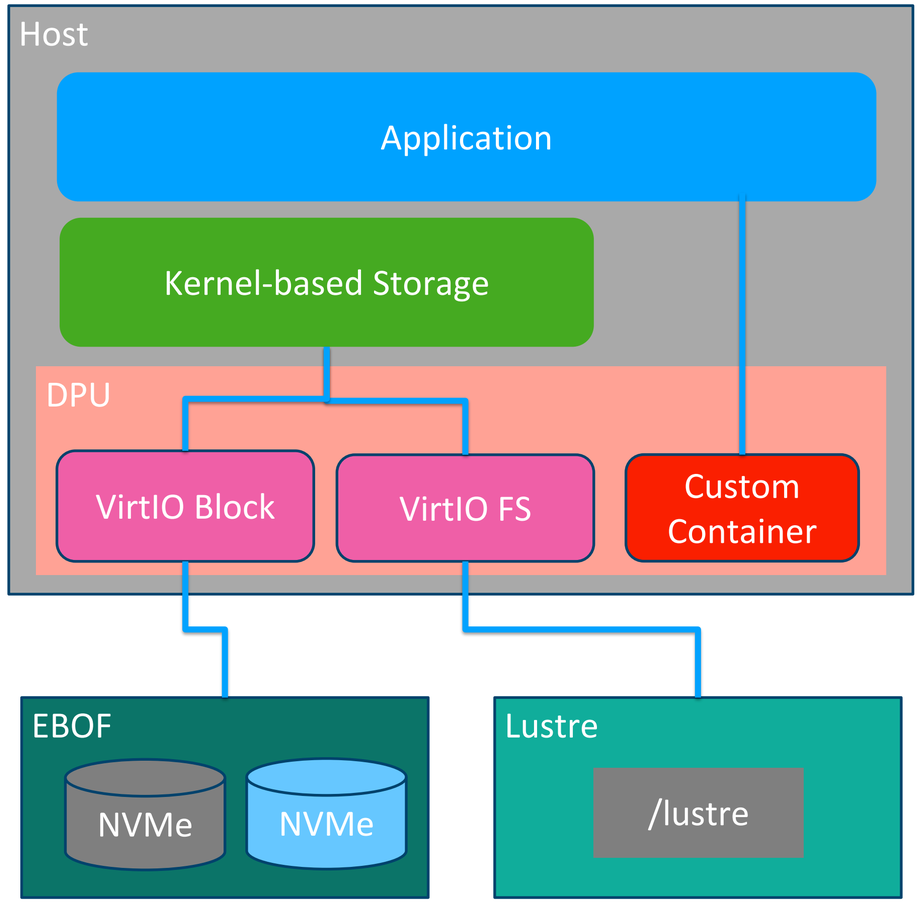
File System Client Offload to DPU (VirtIO FS)

- SNAP also provides a virtioFS driver that allows a file system client (e.g. Lustre) to run on the DPU with access to a local FS provided to the host as a local mount
- Reduce jitter associated with file system clients and async I/O

Advanced Storage Offloads for DPU

Containers running on the DPU can also provide higher level services

- I/O proxy ranks running on DPUs
- Key-value services running on DPUs (work by UC Riverside)
- Storage system services for data compression, checksum, etc running on the DPU



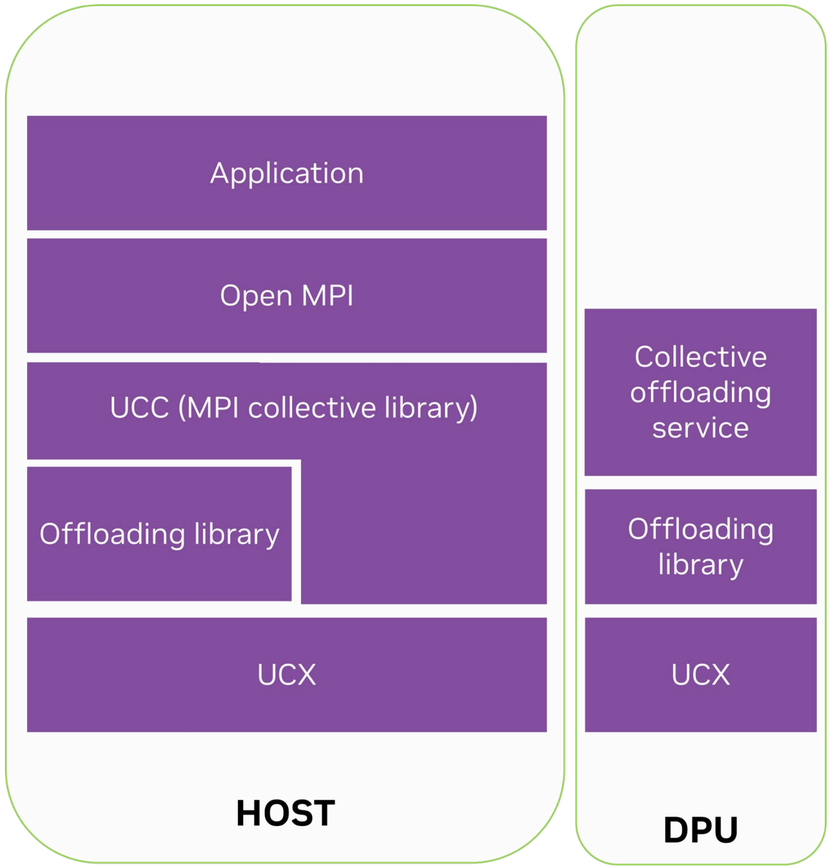
More Information - SmartNIC Research Directions

- *SIAM PP 2024 mini-symposium at https://crnch.gatech.edu/siam-pp-2024/*
- *ISC'2023 DPU Workshop https://dpu.ornl.gov*
- *SmartNIC surveys like Kfoury, et al. “A comprehensive survey on SmartNICs: Architectures, development models, applications, and research directions." IEEE Access (2024).*

SmartNIC SW Infrastructure – HPC Programming Approaches

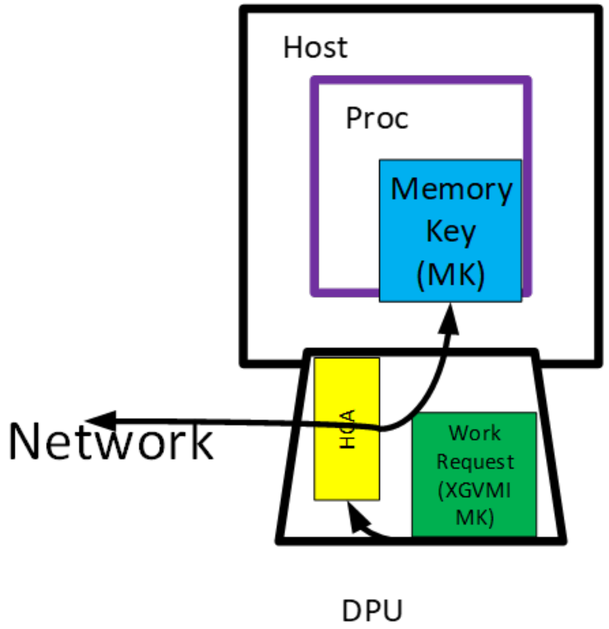
DPU Software Components for HPC Programming

- UCC
  - Collective algorithms
- Service Processes (SP)
  - Run on the BlueField
  - Single SP services several host-side processes
- DOCA UROM – Run-time library used that serves the offloaded algorithms
  - Control path – provides coordination services, active message based
  - Services provided:
    - Offload engines
    - Offload service bootstrapping
    - Group and topology management
    - UCX endpoint cache
    - Execution context
    - Notifications
- UCX
  - Data-path services
  - Extended to support the alias memory keys (XGVMI)
- BlueField optimized collectives



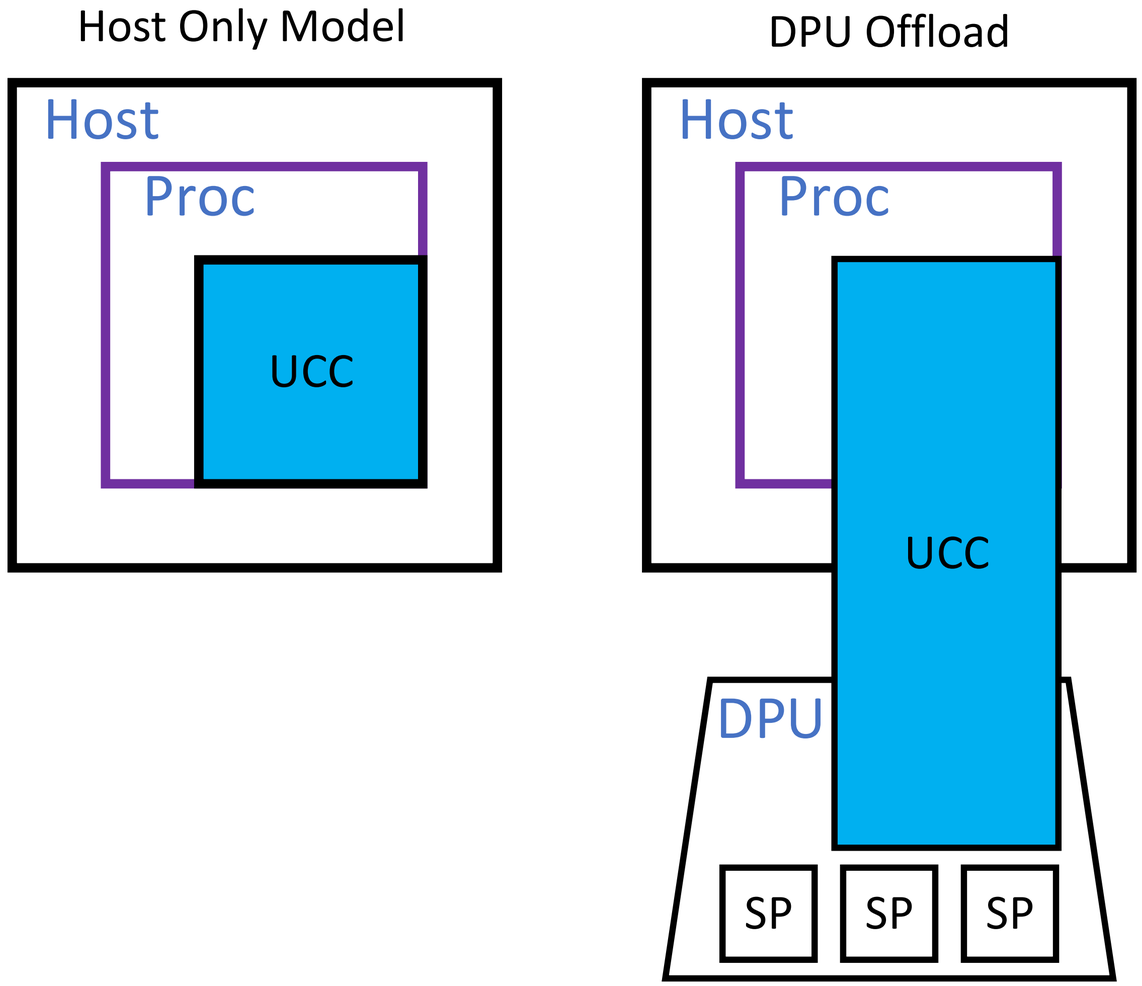
Cross-GVMI Memory on DPUs

- Memory keys that allow DPU based work-requests to reference host-side memory

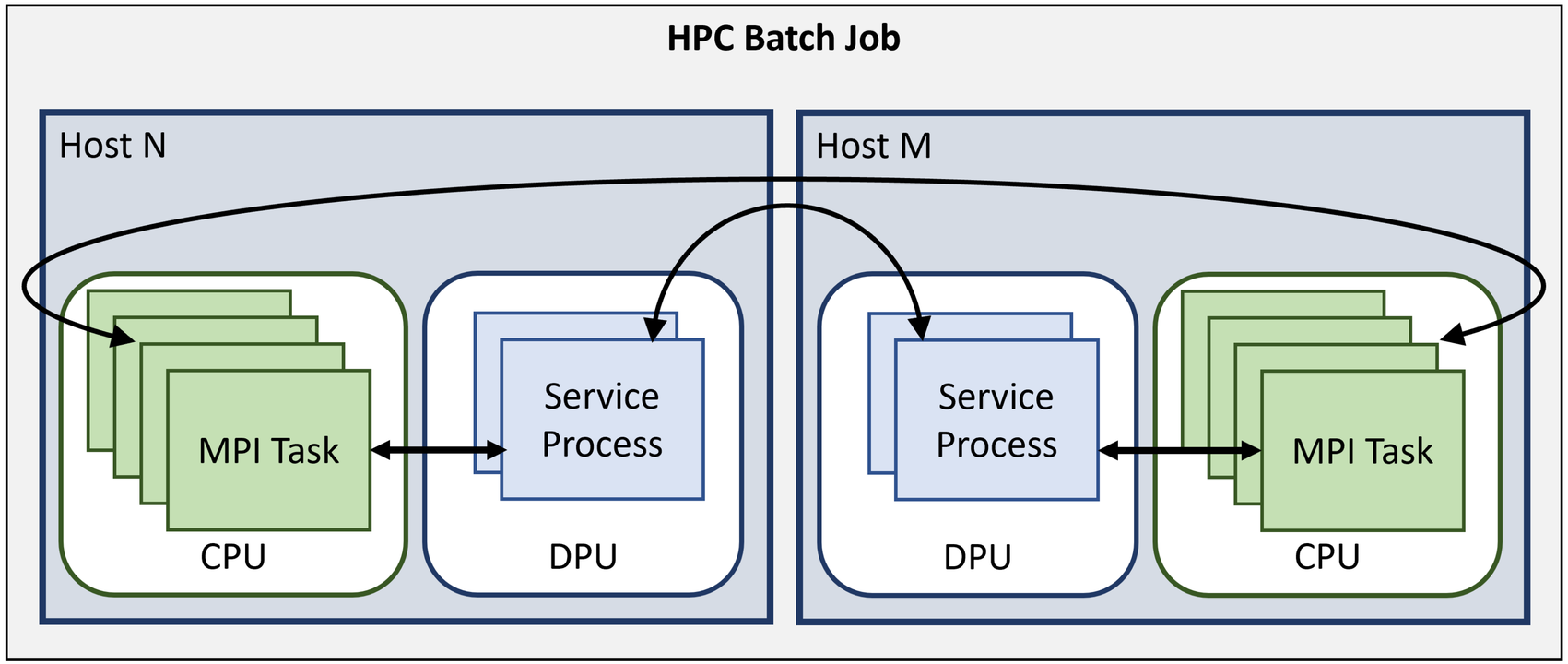


- DPU can initiate data transfer on behalf of the host, with no host involvement
  - Do not need to move data to the DPU before the DPU can send it
  - Can post receive work requests on behalf of memory that is host resident

UCC DPU Offload Model

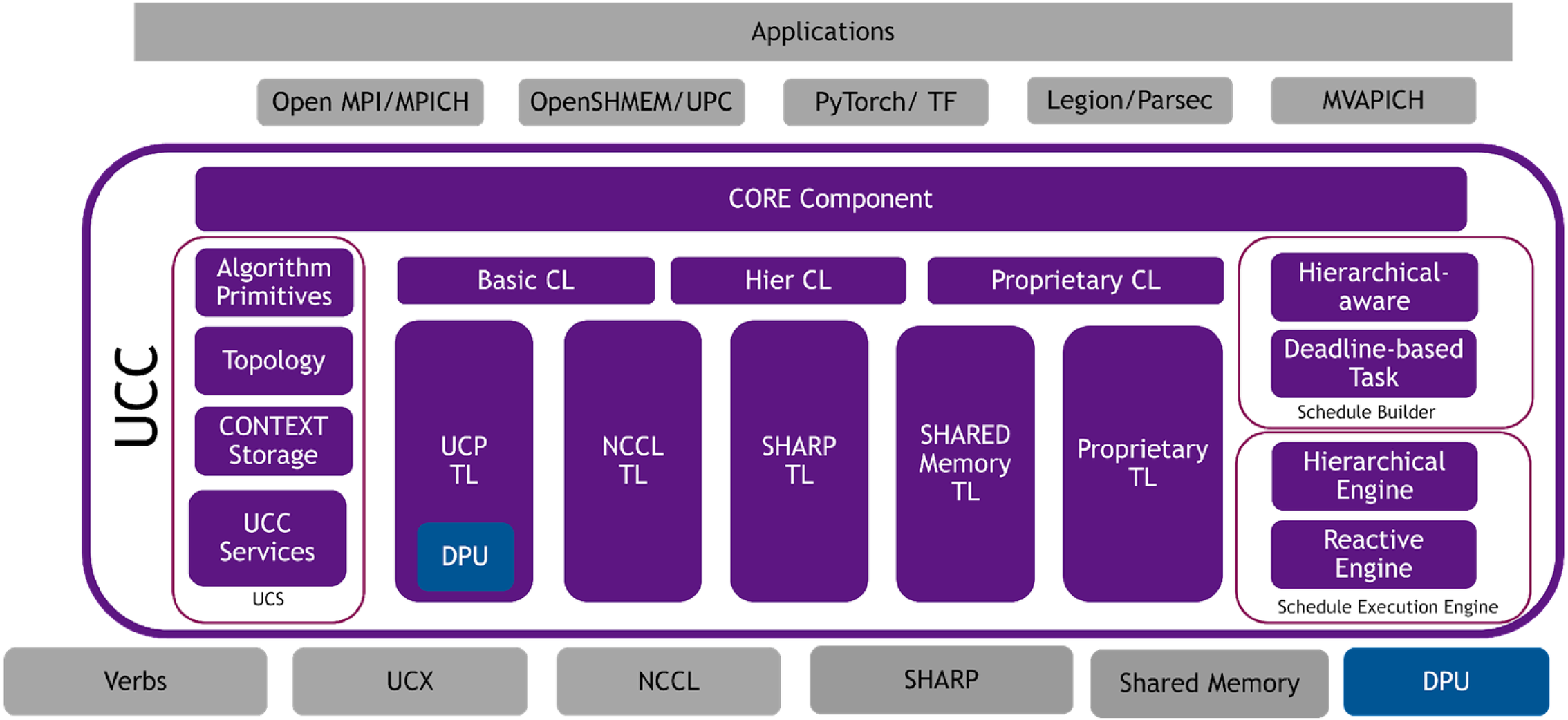


Job Process Layout

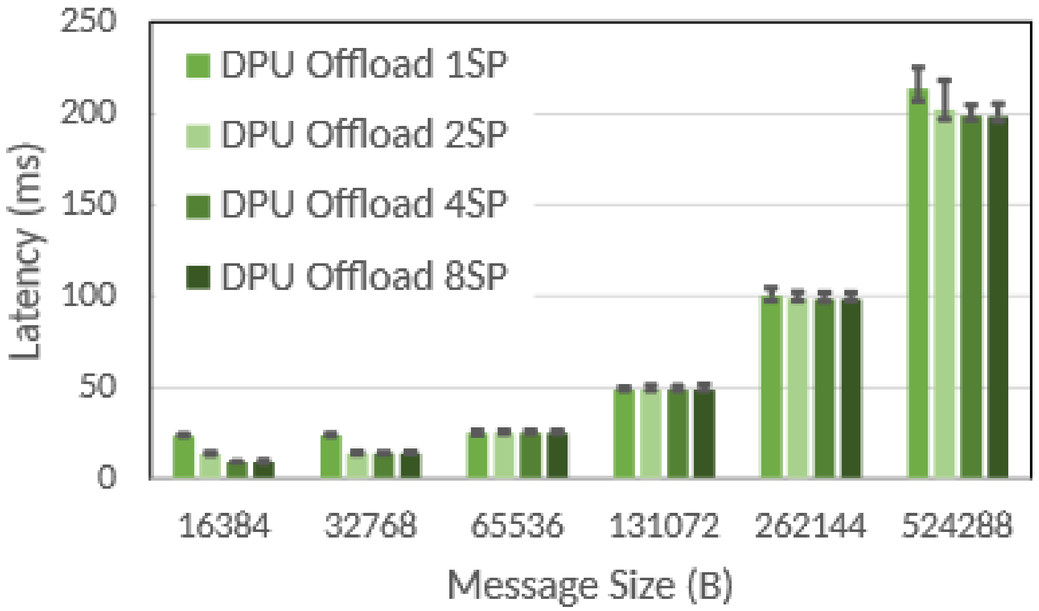


- Service Processes on the DPU enable seamless MPI jobs that make use of DPU and host CPU resources

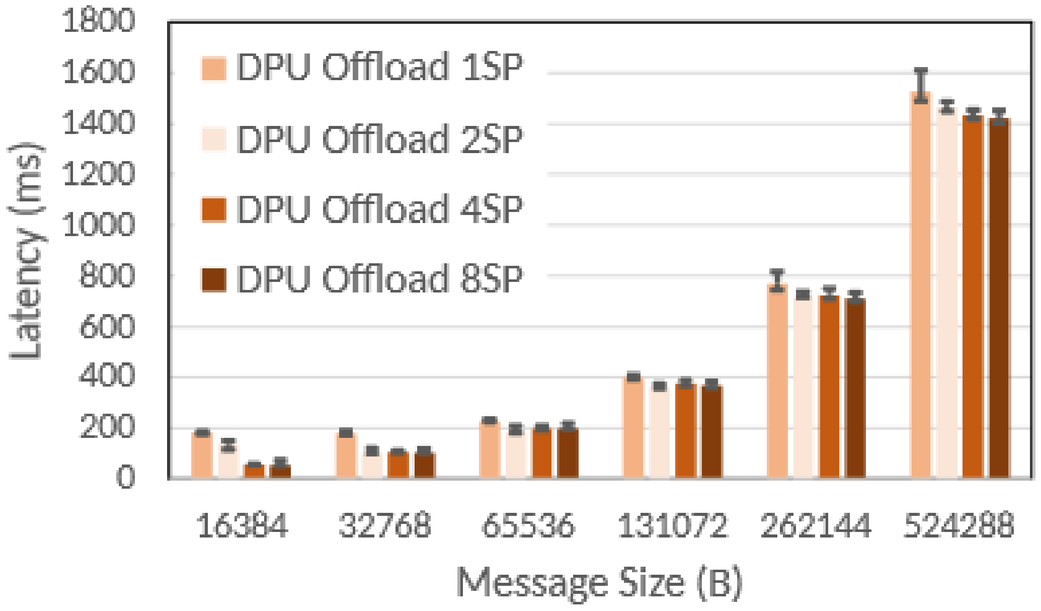
Unified Collective Communication (UCC) Architecture



Collective Latency (OSU) as a Function of Service Process Count

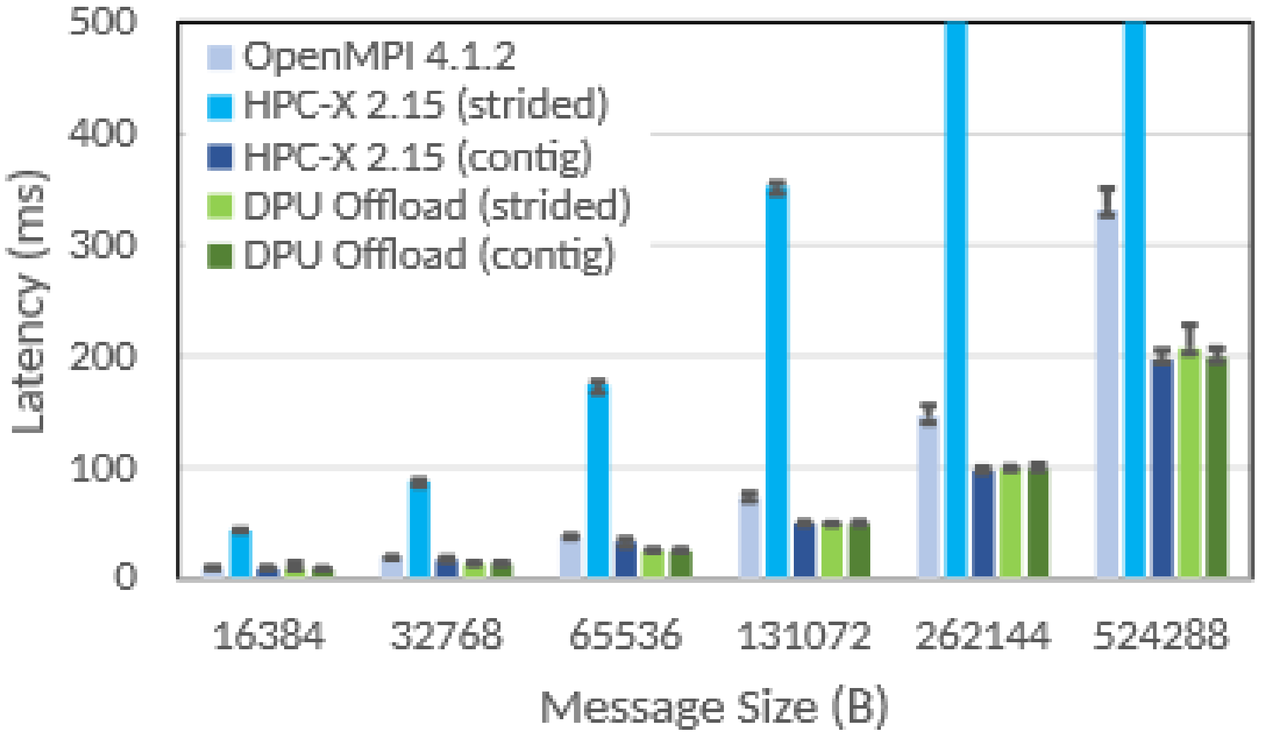


(a) MPI_Allgatherv

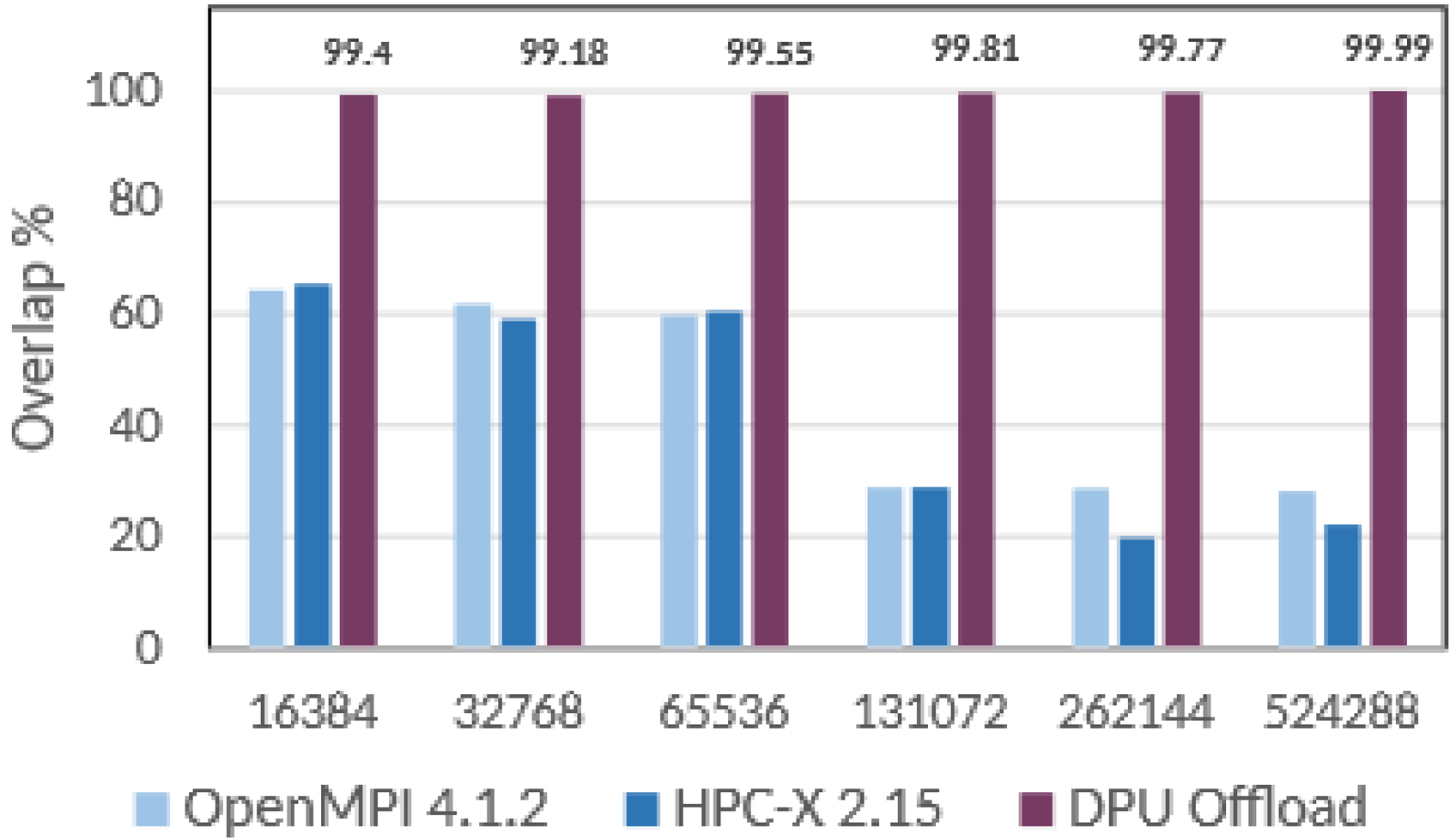


(b) MPI_Ialltoallv

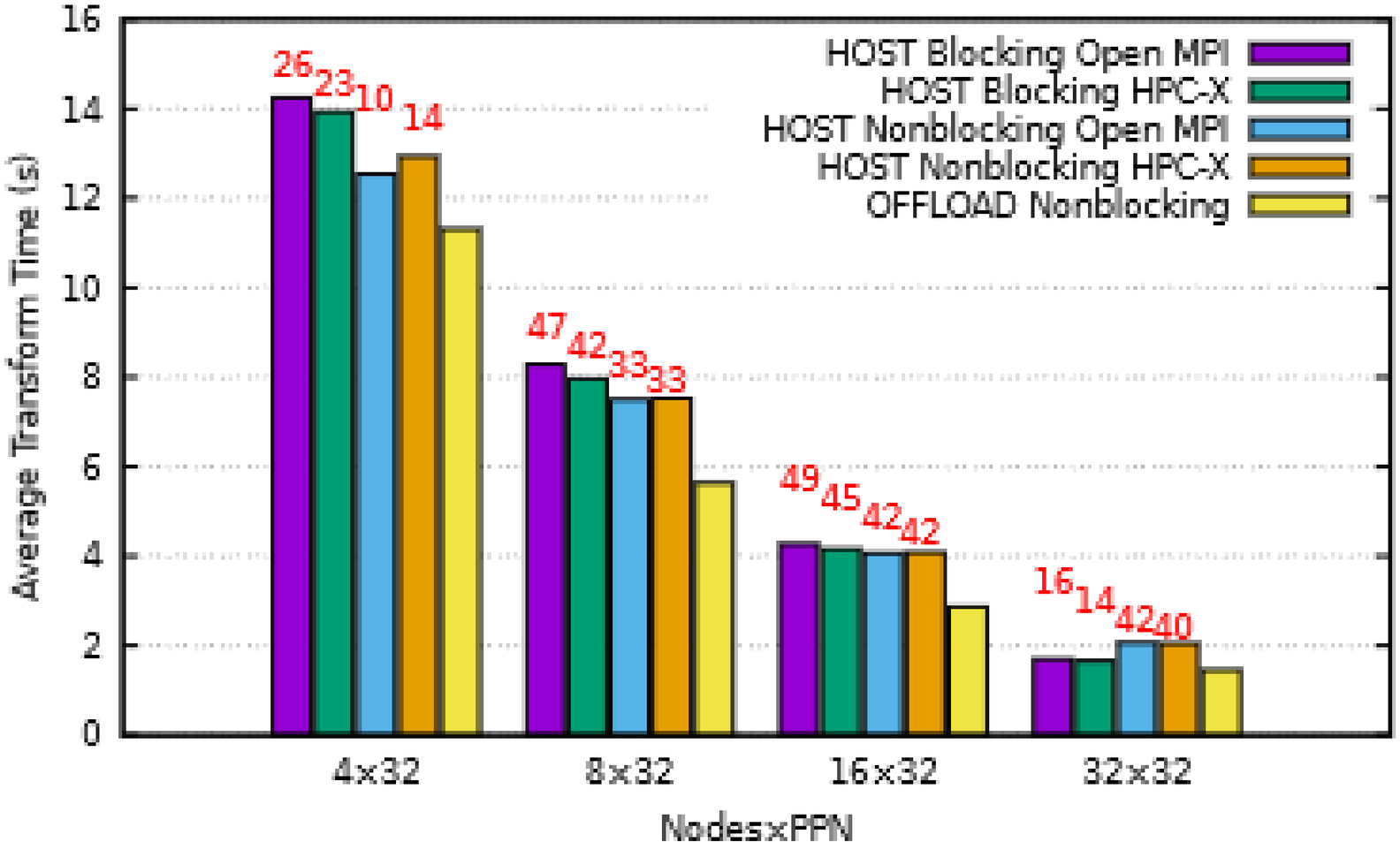
MPI_AllgatherV OSU Latency – Implementation Comparison



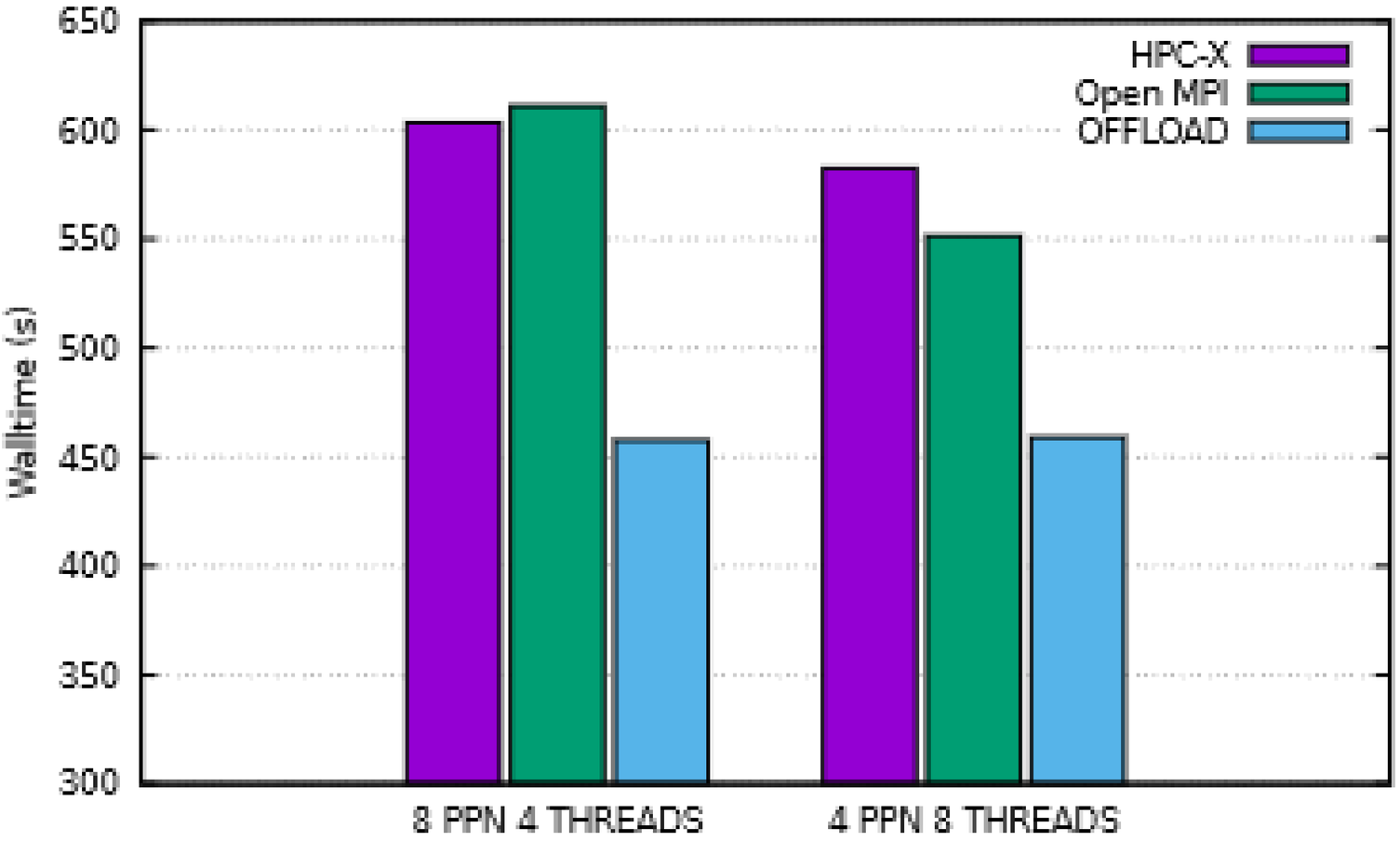
MPI_iAlltoallv OSU overlap as Function - Implementation comparison



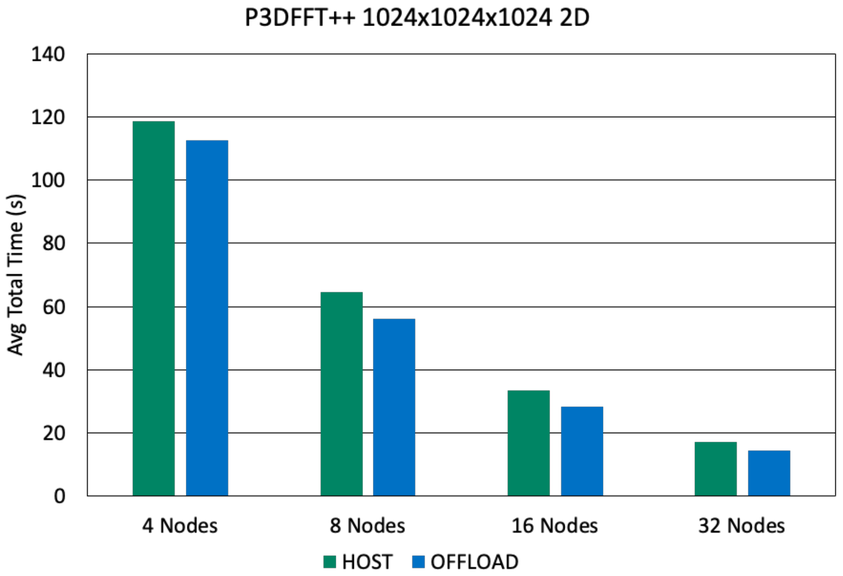
P3dFFT++ FFT Transform



Octopus Performance

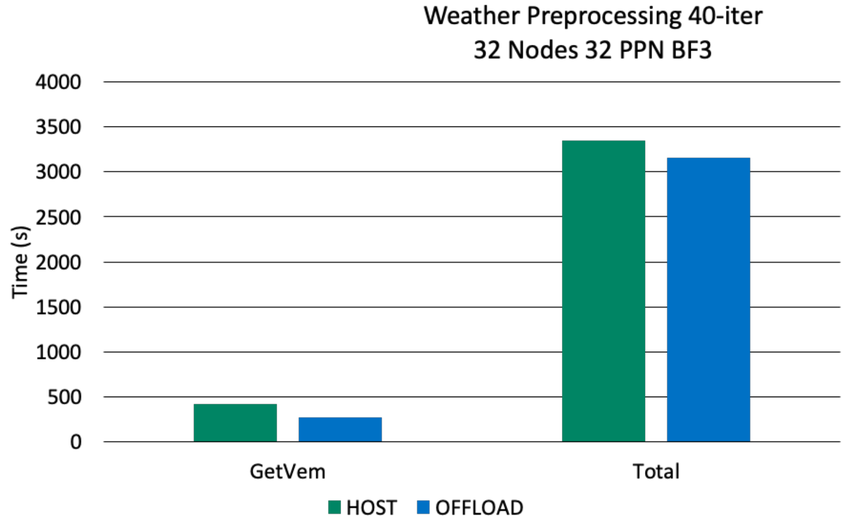


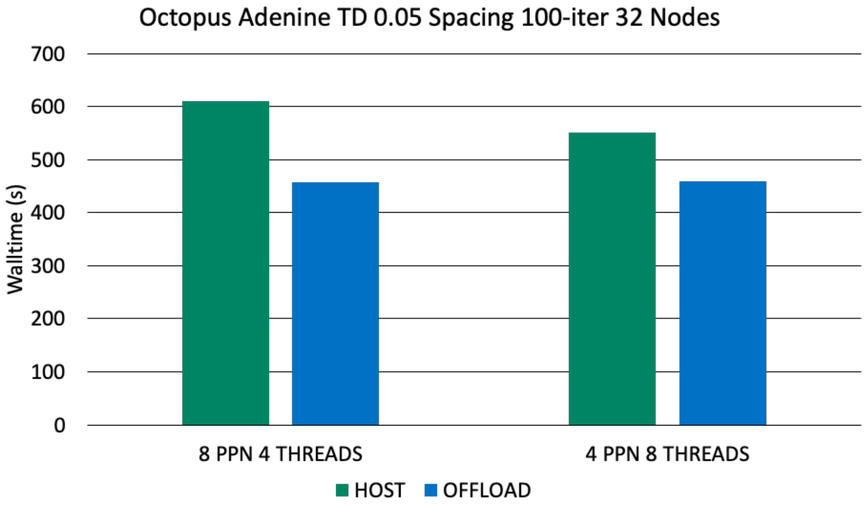
BlueField-3 Application Acceleration



+17%

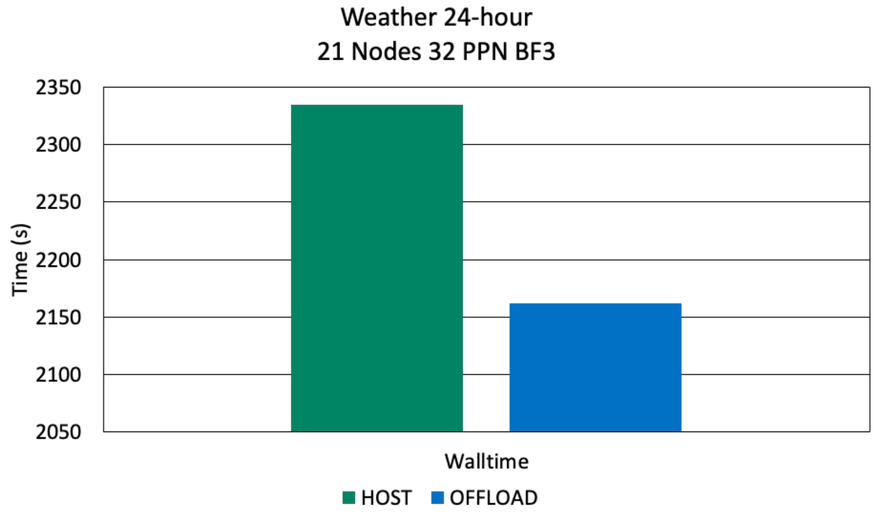
+7%





+17%

+6%

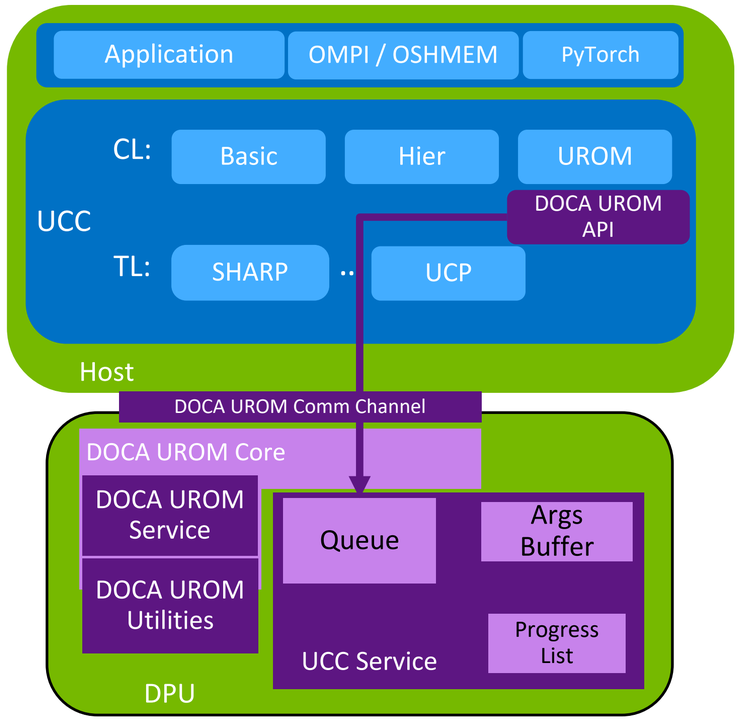


DPU UCC Offload with One-sided Collectives

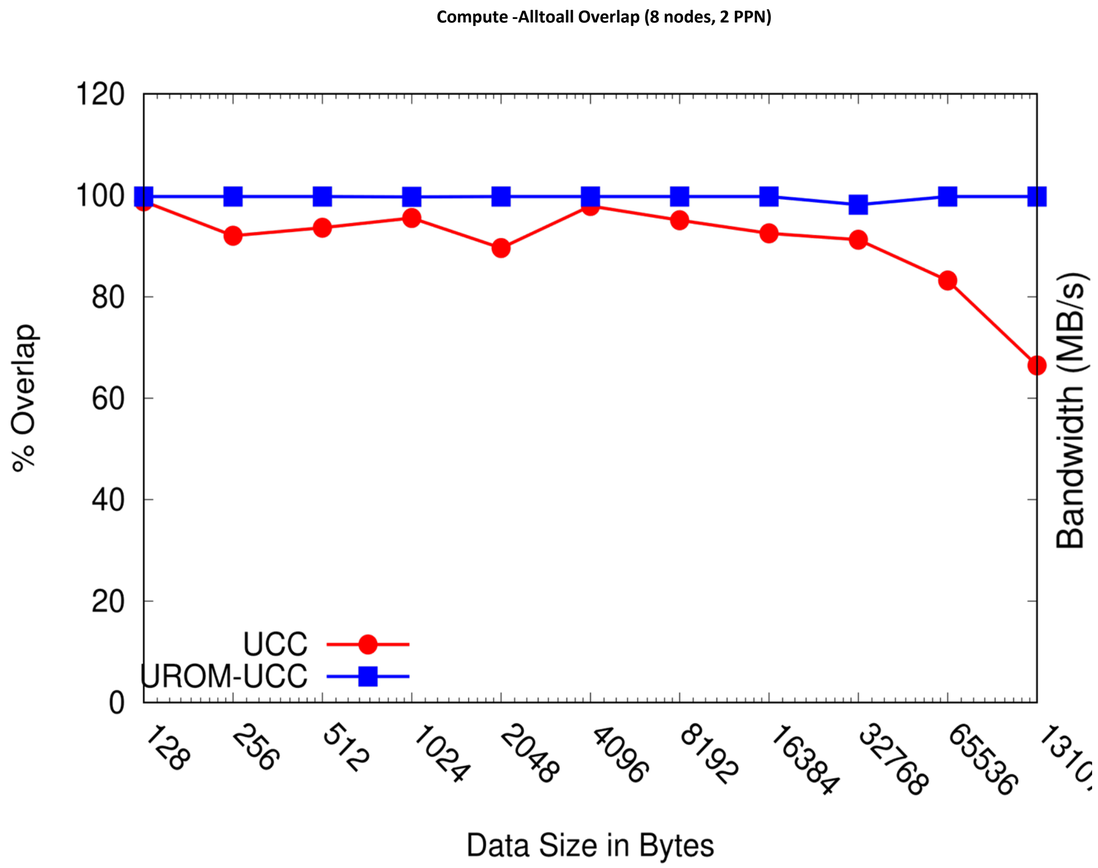
- Complete Offload of UCC library to DPU
  - Offloaded Collective Progress
    - Capable of 100% overlap
  - Capable of smart collective progress to minimize congestion
    - Leverage network telemetry and make informed decisions
- Enabled by UCC UROM CL
  - Execution of unmodified applications (e.g., OSHMEM / MPI applications)
  - Example:

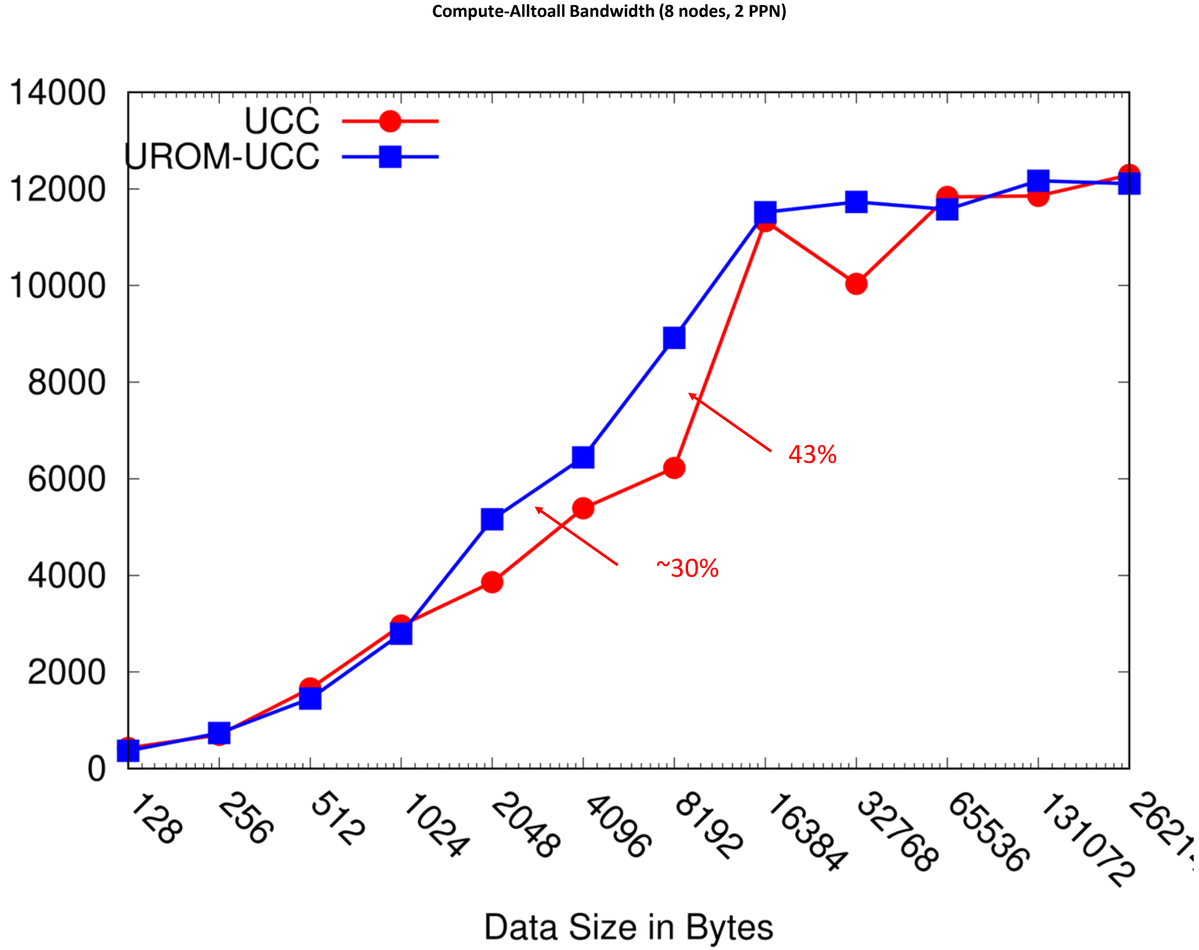
```
oshrun -np 4 -mca scoll_ucc_enable 1 --mca scoll_ucc_priority 100
--mca scoll_ucc_cls urom,basic ./my_ucc_app
```

- Evaluated using Alltoall with compute micro-benchmark
  - Near 100% overlap with increasing data sizes
    - 25% improved overlap vs host due to offloaded progress
  - Up to 43% performance improvement for bandwidth
    - GET-based algorithm for small / medium messages
    - PUT-based algorithm for large messages



PERFORMANCE OF COMPUTE-ALLTOALL BENCHMARK



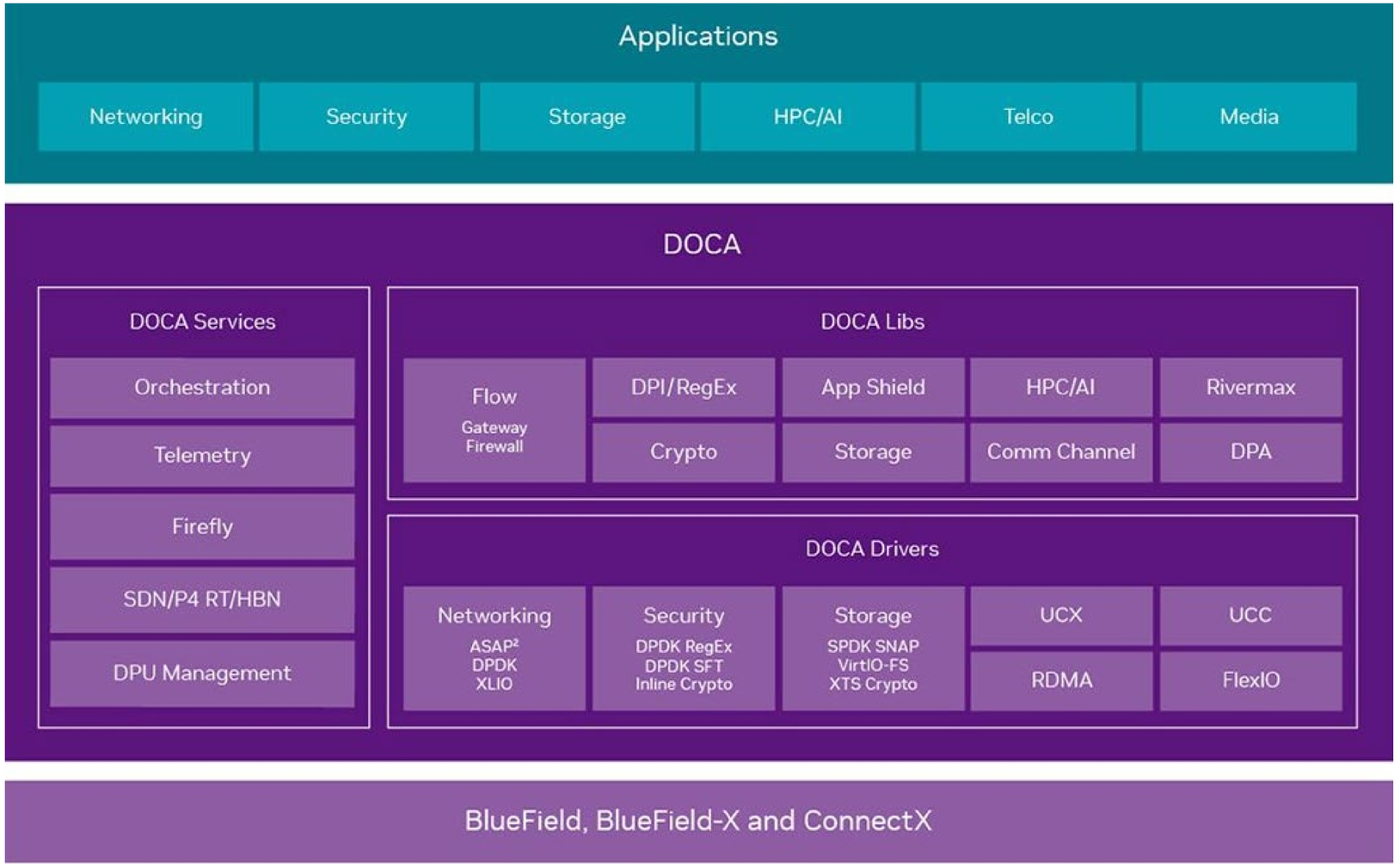


OpenMP Offload to DPUs

Brought to you by the AccelCom Group at BSC
M. Usman, M. Benito, S. Iserte, **A. J. Peña** - accelcom@bsc.es

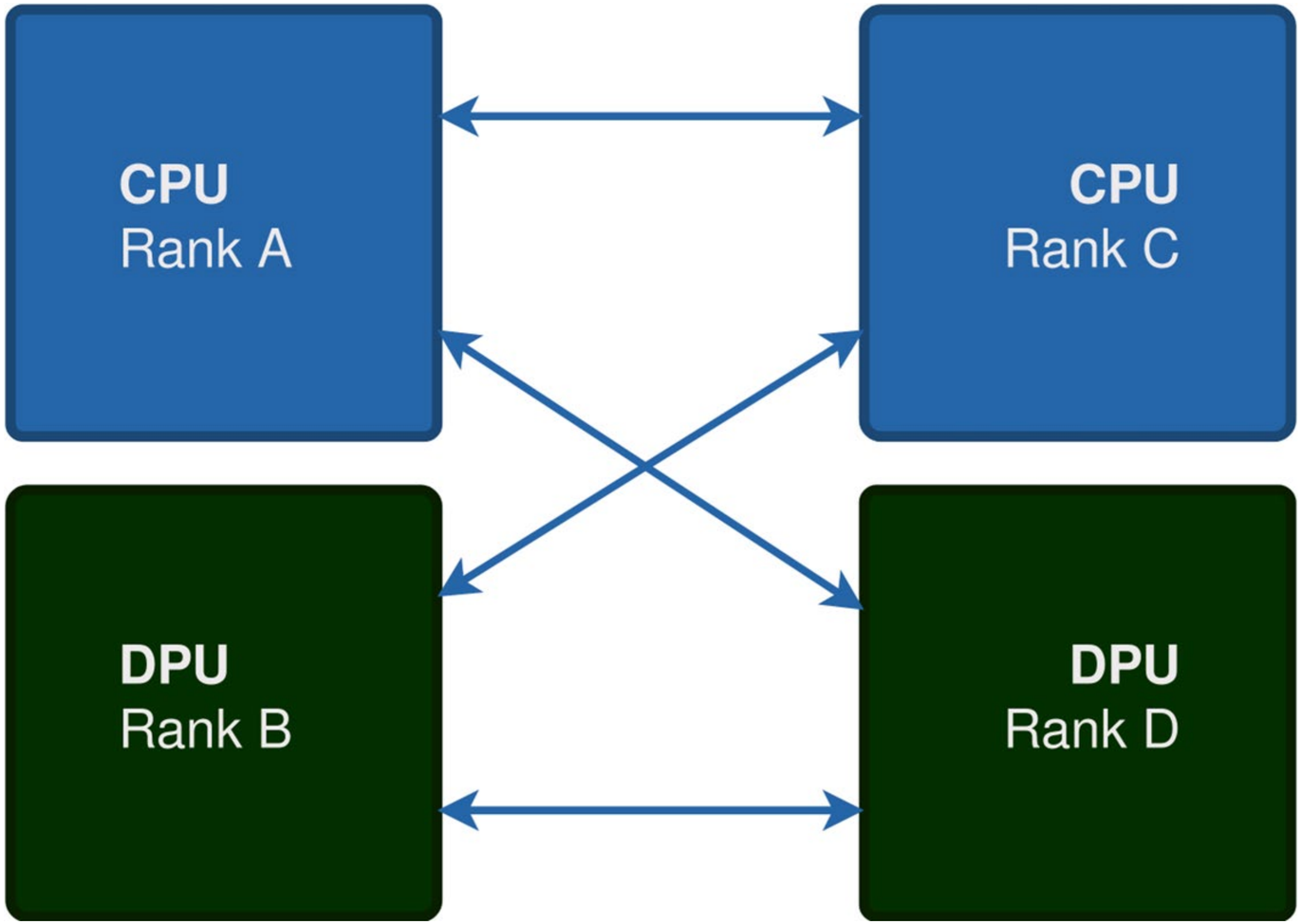
Outline

- Nvidia BlueField DPU Programming
- OpenMP DPU Offloading Support (ODOS)
  - Demo #1
- ODOS MPI Support
  - Demo #2
- Use case: Halo Exchange
- Performance



BlueField DPU Programming Low Level API

- DOCA is to DPUs what CUDA is to GPUs
- DOCA is not an offloading API
- Too low level for domain scientists



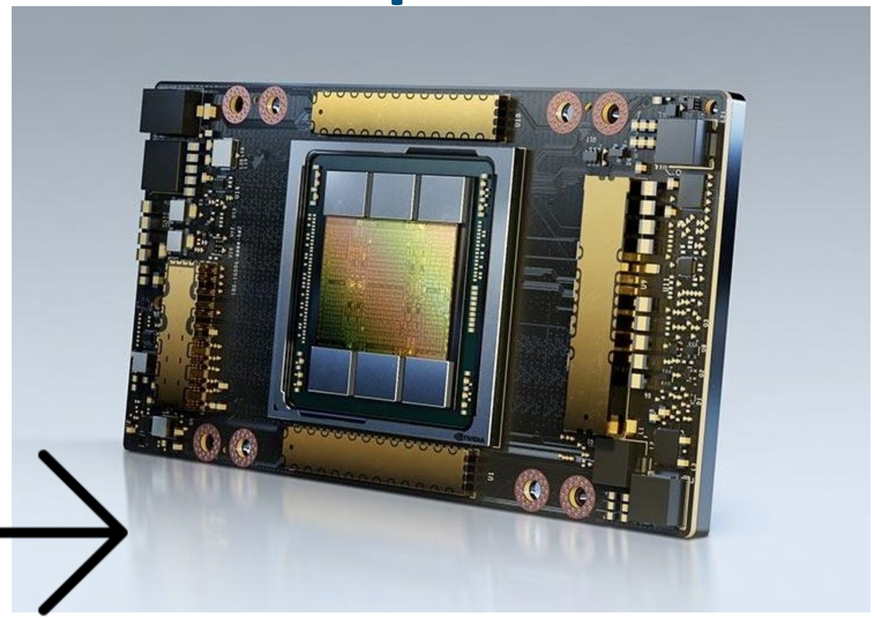
BlueField DPU MPMD Programming

- Multiple program multiple data execution model in MPI
- Poor productivity because the processors are widely different

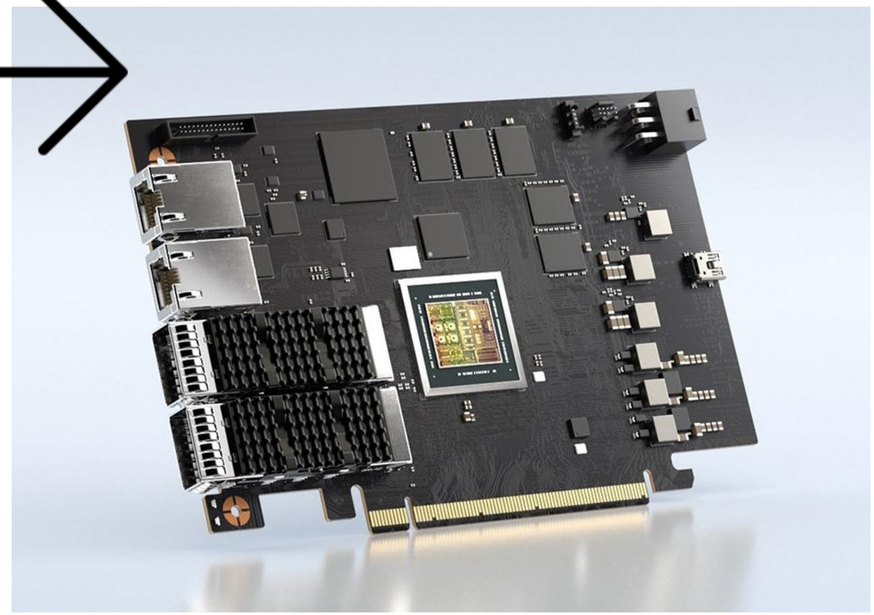
DPU Offloading Programming with standard OpenMP

```
int main ()
{
    /*************/
    /* exec in host */
    /*************/

    #pragma omp target device (  )
    {
        /***************/
        /* exec in target */
        /***************/
    }
}
```

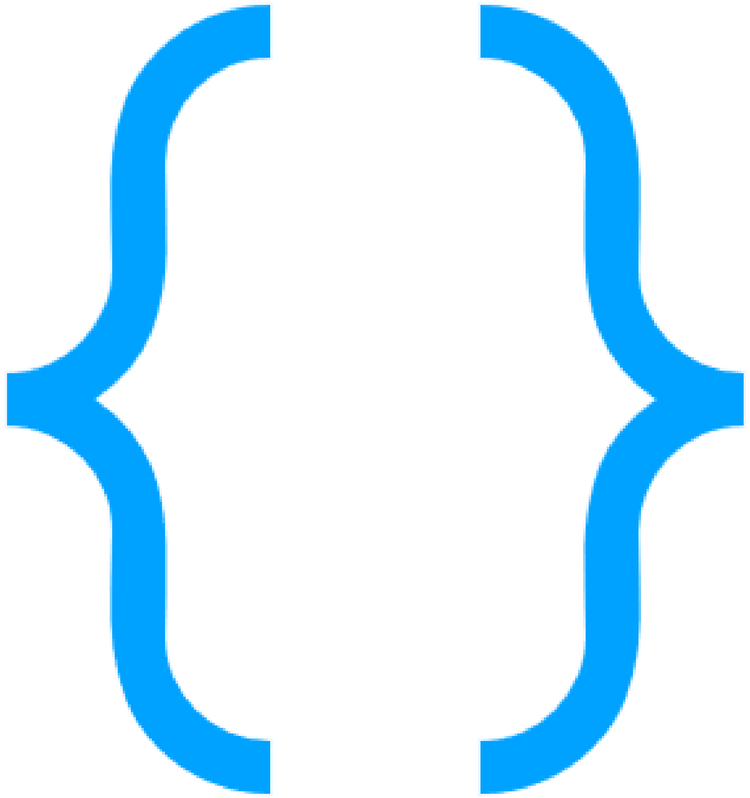
GPU



DPU

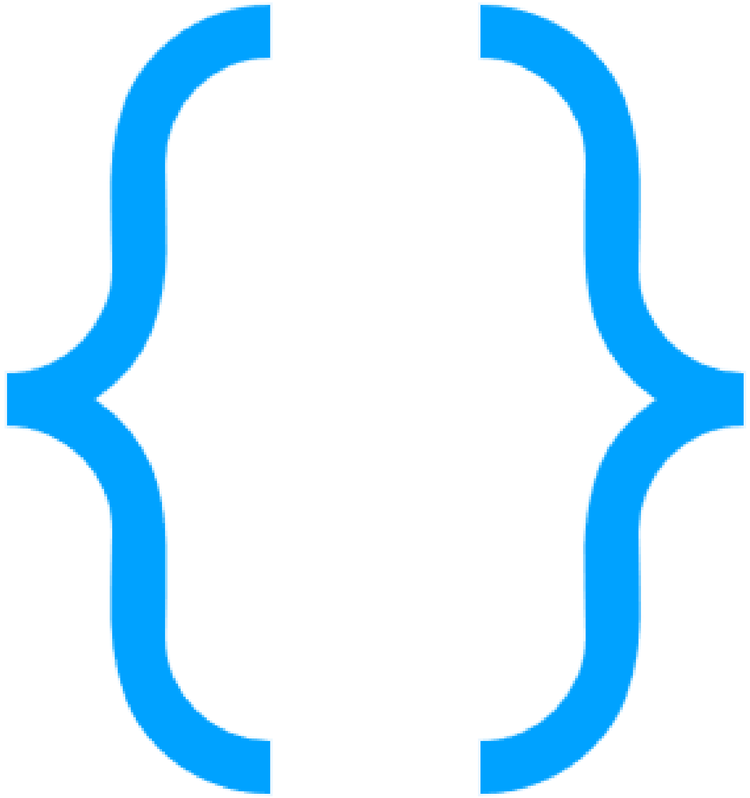


Basic Syntax



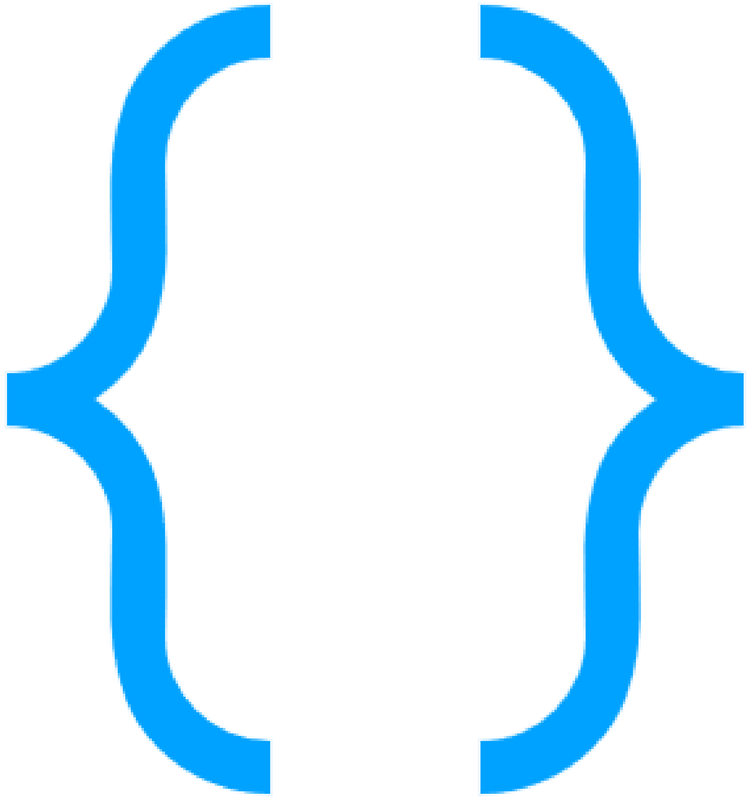
```
// exec in host

#pragma omp target
{
   // exec in target
}
```

How do I Know the Device Number?



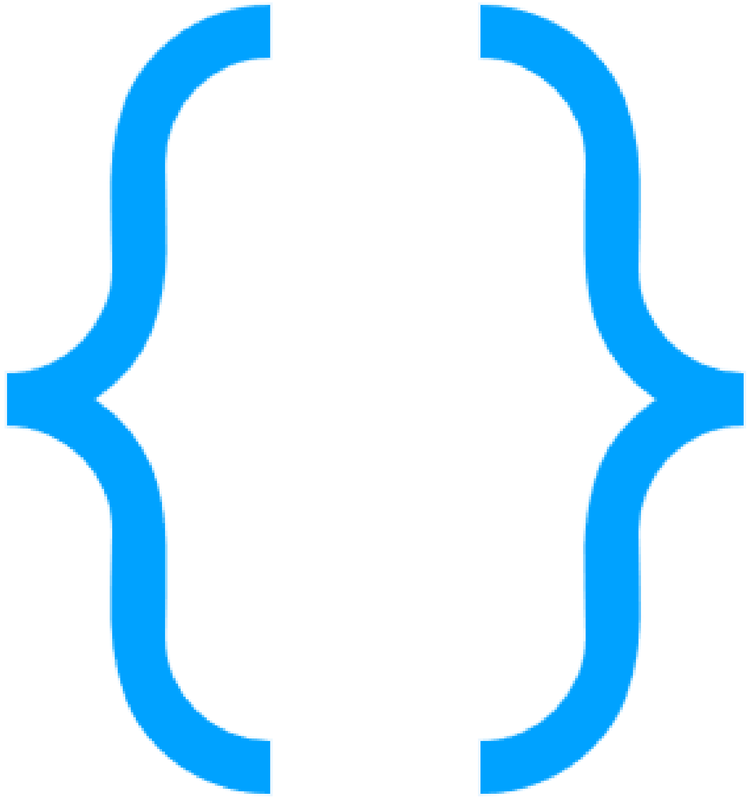
```
$ llvm-omp-device-info

Device (4):
    CUDA Driver Version:                    10020
    CUDA Device Number:                     0
    Device Name:                            Tesla V100-SXM2-16GB
    Global Memory Size:                     16911433728 bytes
    Number of Multiprocessors:              80
    Concurrent Copy and Execution:          Yes
    Total Constant Memory:                  65536 bytes
    Max Shared Memory per Block:            49152 bytes
    Registers per Block:                    65536
    ...
    Compute Capabilities:                   70
```

How do I Know the Device Number?



```
$ llvm-omp-device-info

...
Device (7):
    BlueField DPU device
        iface      :  ib0
        ib dev     :  mlx5_2
        doca id    :  2
        pci addr   :  42:00:0
        comm channel:
            max msg size              :   4080
            max send    queue size    :   8192
            max receive queue size    :   8192
...
```

How to Choose a DPU?



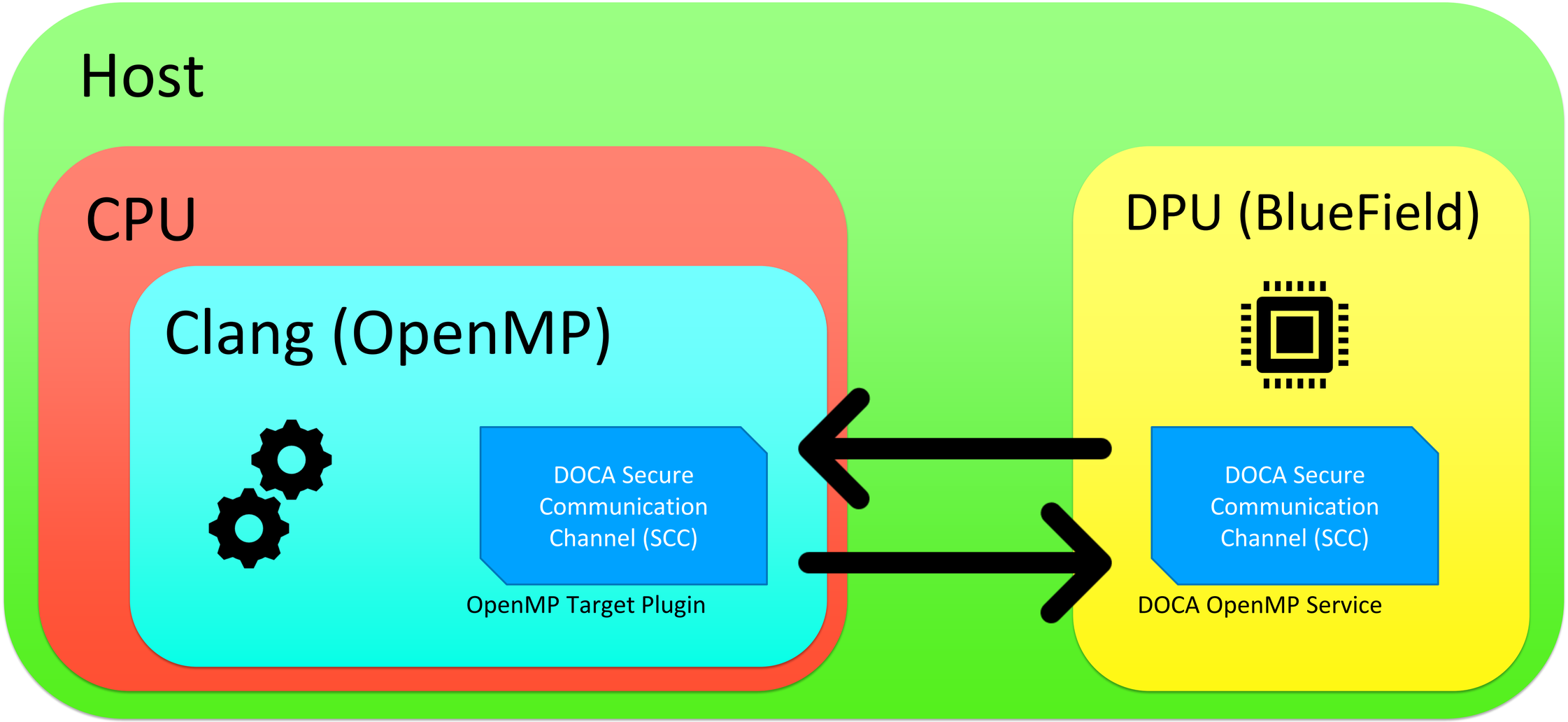
```
// exec in host
#define __DPUID__ 7
#pragma omp target device(__DPUID__)
{
   // exec in target
}
```

```
void main( int argc, char* argv[] ) {
  // Data initialization
  MPI_Init( &argc, &argv );
  MPI_Comm_rank( MPI_COMM_WORLD, &dpurank );
  if ( dpurank )  // DPU rank receives data
    MPI_Recv( &buf, SIZE, MPI_INT, 0, … );
  else            // Host rank sends data
    MPI_Send( &buf, SIZE, MPI_INT, dpurank, … );
  for( int t = 1; t <= TIMESTEPS; t++ ) {
    if ( dpurank );     // Offloaded compute
    else;               // Host compute
  }
  if ( dpurank )  // DPU rank send data
    MPI_Send( &buf, SIZE, MPI_INT, 0, … );
  else            // Host rank receive data
    MPI_Recv( &buf, SIZE, MPI_INT, dpurank, … );
  MPI_Finalize();
}
```

```
void main( int argc, char* argv[] ) {
  // Data initialization
  #pragma omp target
            data map(tofrom:buf[0:SIZE]))
  {
    for( int t = 1; t <= TIMESTEPS; t++ ) {
      #pragma omp target
      {
        // Offloaded compute
      }
      // Host compute
    }
  }
}
```

OpenMP DPU Offloading Support

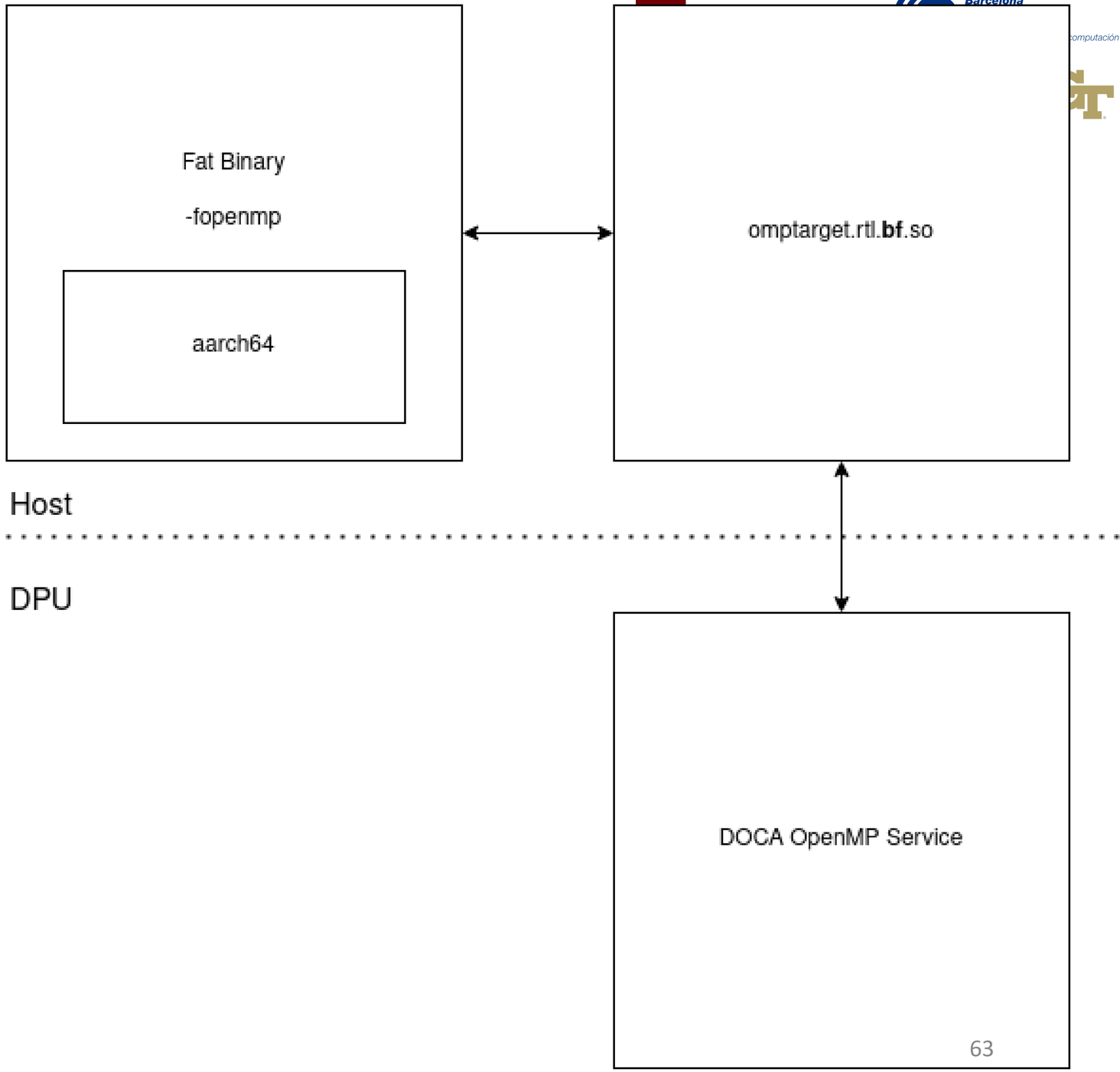
ODOS Architecture



M. Usman, S. Iserte, R. Ferrer, and A. J. Peña, "DPU Offloading Programming with the OpenMP API." *LLVM-HPC23 co-located with SC23*, https://dl.acm.org/doi/10.1145/3624062.3624165

ODOS Modules

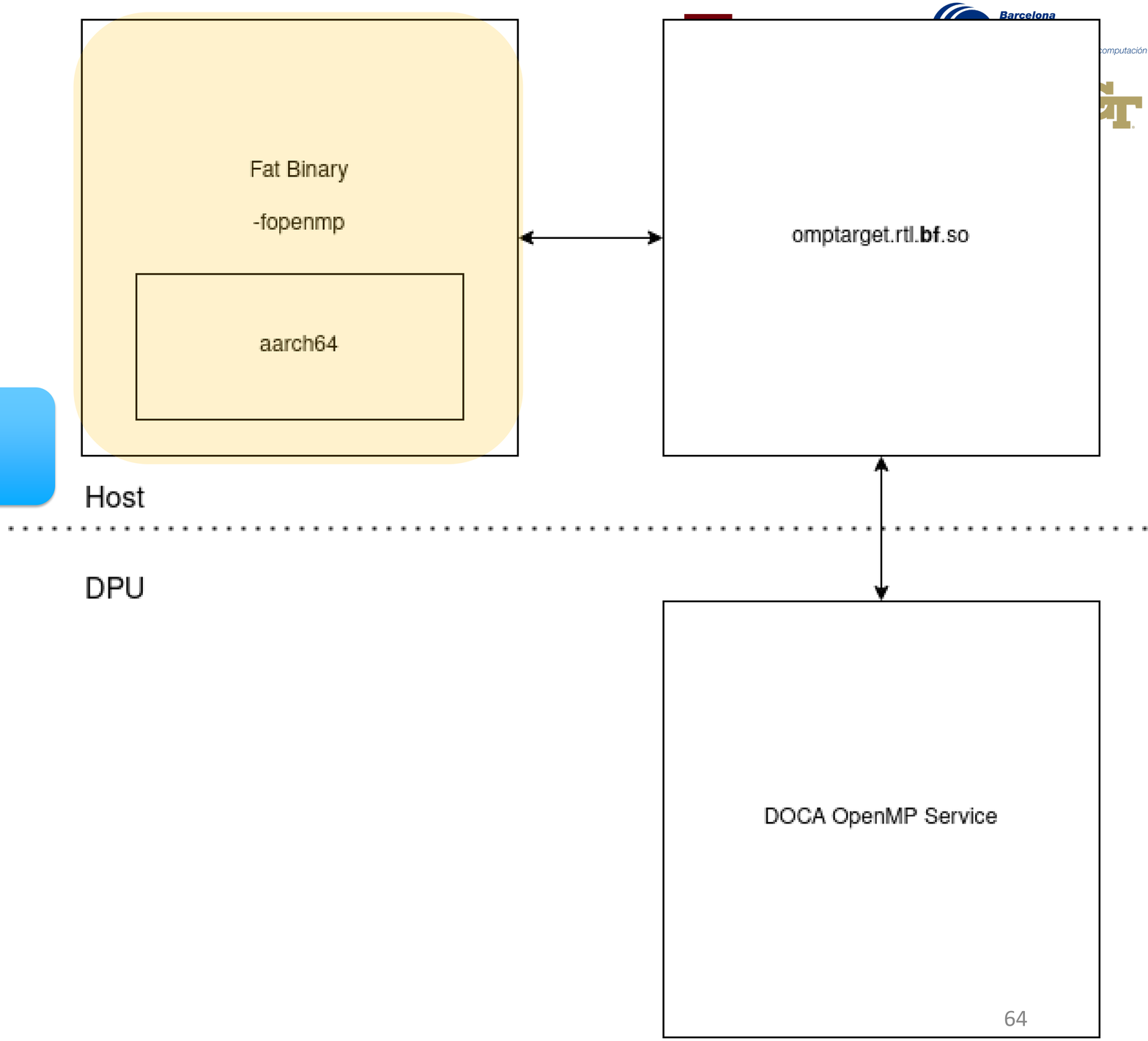
- Fat Binary Generation
- OpenMP Runtime (host)
- OpenMP Service (device)



Fat Binary Generation

Cross-compilation

- I.e., x86-64 & aarch64
- Default Linux GNU GCC compiler

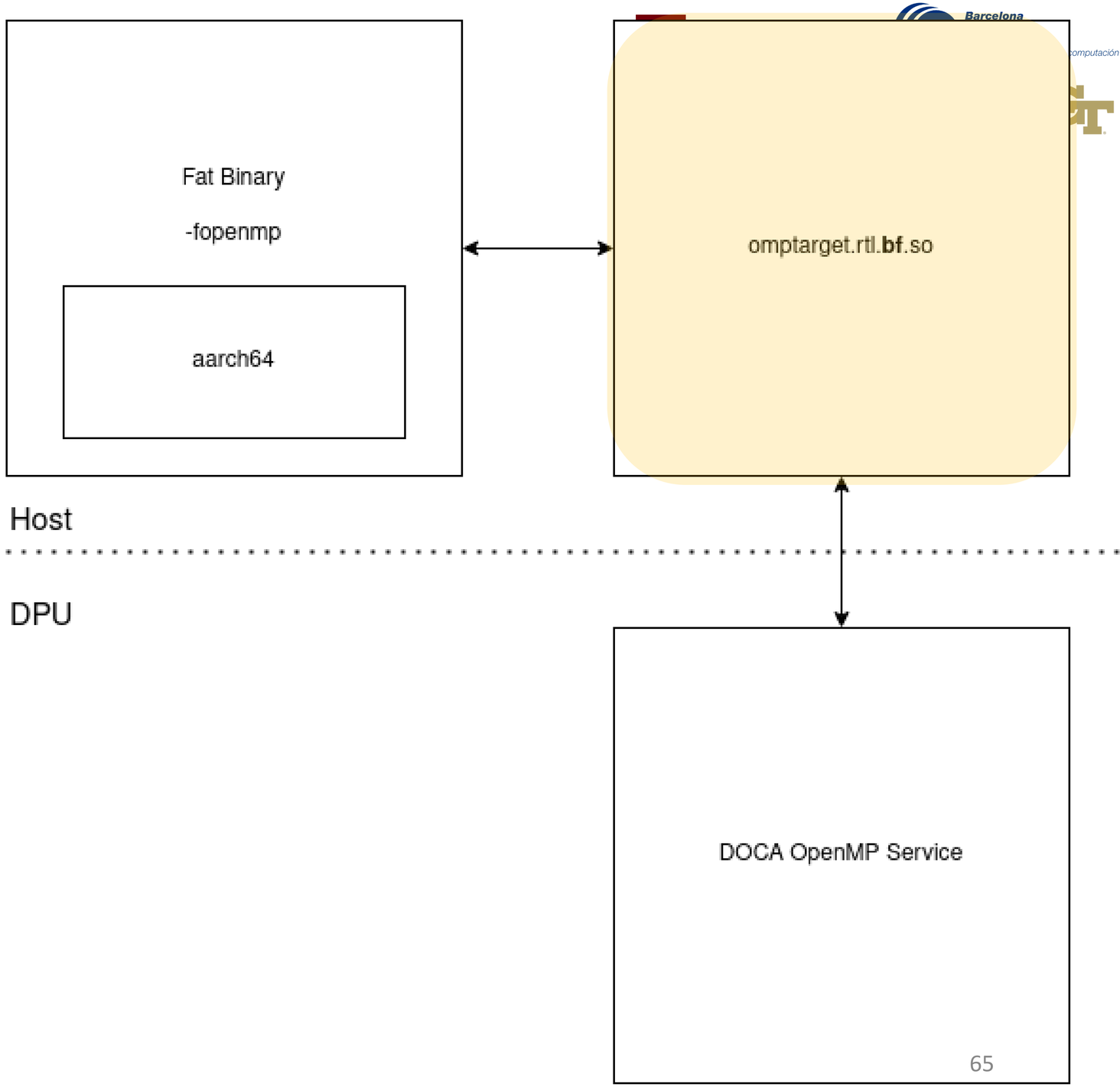


Target Plugin

Communication with the DPU using DOCA SCC

Send commands to

- Load binary
- Allocate and exchange data
- Execute target blocks



DOCA OpenMP Service

Receive commands using DOCA SCC

Load binary

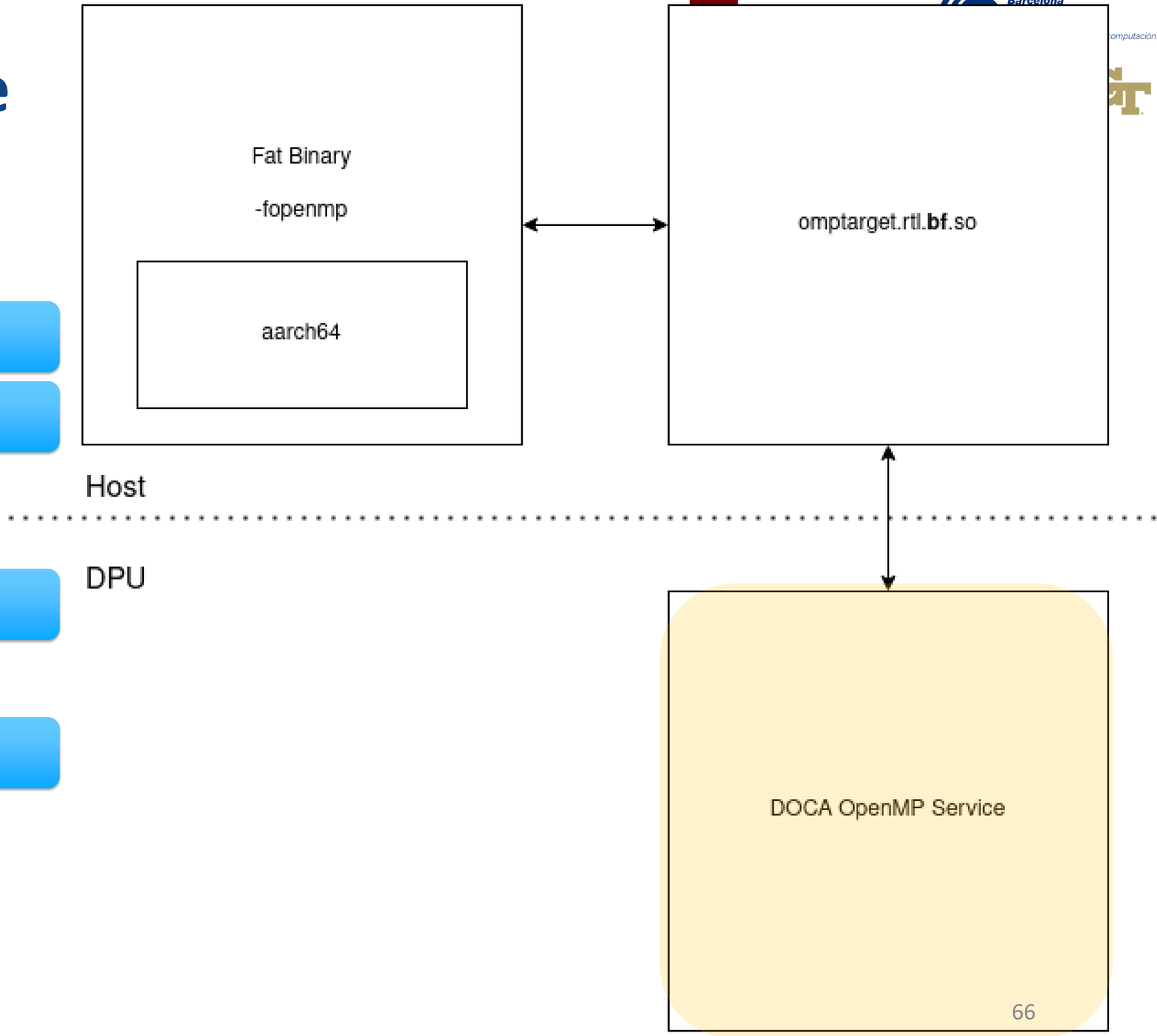
- Save image as a shared object
- Open the image [dlopen]
- Send back handles of symbols to host [dlsym]

Data operations

- Allocate/Delete [malloc/free]
- Send/Retrieve [DOCA SCC]

Target block execution

- Run target handle [ffi_prep_cif, ffi_call]



DEMO #1

- How to compile ODOS-enabled applications
- Basic example
- OpenMP "Parallel" feature
- Shared libraries
- Asynchronous TCP

How to Compile ODOS-enabled Applications

ODOS-enabled LLVM

Enable OpenMP and specify offloading targets

```
$ clang -fopenmp -fopenmp-targets=aarch64-unknown-linux app.c -o app
```

```c
#include <omp.h>
#include <stdio.h>

int main()
{
#pragma omp target
      puts("Hi Folks!");

      return 0;
}
```

Host

```
uthmanhere@thor013:~/omp_exp/labs_sc23/task_a/build$
```

DPU

```
uthmanhere@thorbf3a013:~$
```

```c
#include <omp.h>
#include <stdio.h>

int main()
{
#pragma omp target
#pragma omp parallel
        puts("Hey! OpenMP even work in DPU...");

        return 0;
}
```

Host

```
uthmanhere@thor013:~/omp_exp/labs_sc23/task_b/build$
```

DPU

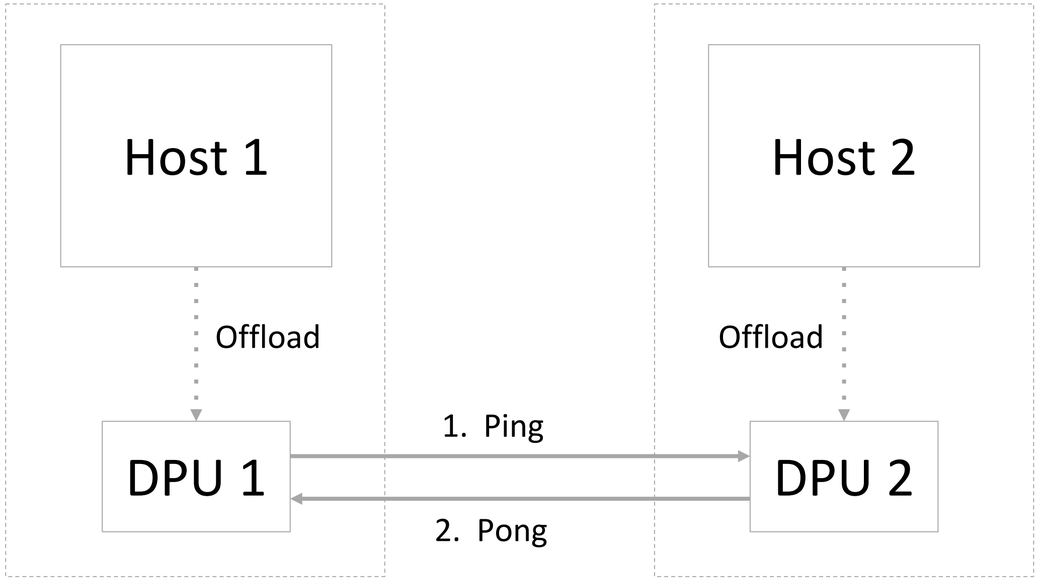
```
uthmanhere@thorbf3a013:~
```

Shared Libraries

```
$ #Compile shared object
$ LLVM/bin/clang log.c -shared -o liblog_x86.so

$ #Cross-compile shared object
$ LLVM/bin/clang log.c -shared -target aarch64-unknown-linux -o liblog_aarch64.so

$ #Link the objects and generate the fat binary
$ LLVM/bin/clang -fopenmp -fopenmp-targets=aarch64-unknown-linux  code.c -o code -L.
-llog_x86 -Wl,--device-linker=-L.,--device-linker=-llog_aarch64
```

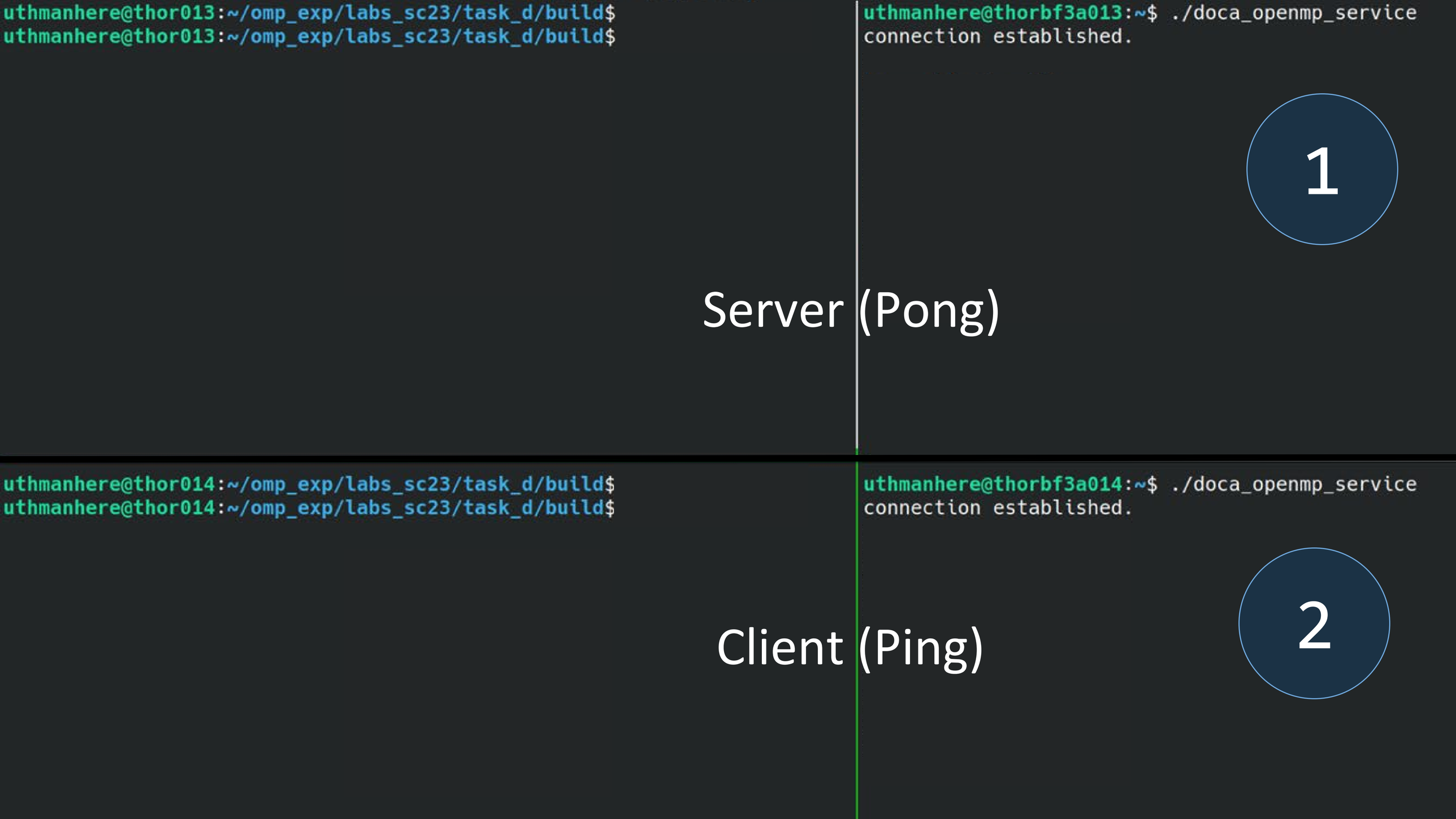
Asynchronous TCP I/O in DPU using OpenMP (1/2)

```
#pragma omp target nowait
{
    // initialize tcp sockets, listen, bind, connect, and accept.
    fd = tcp_init(mode);

    // compute and communicate asynchronously on DPU
    ping_pong(fd, mode);

    close(fd);
}
#pragma omp taskwait
```



Asynchronous TCP I/O in DPU using OpenMP (2/2)

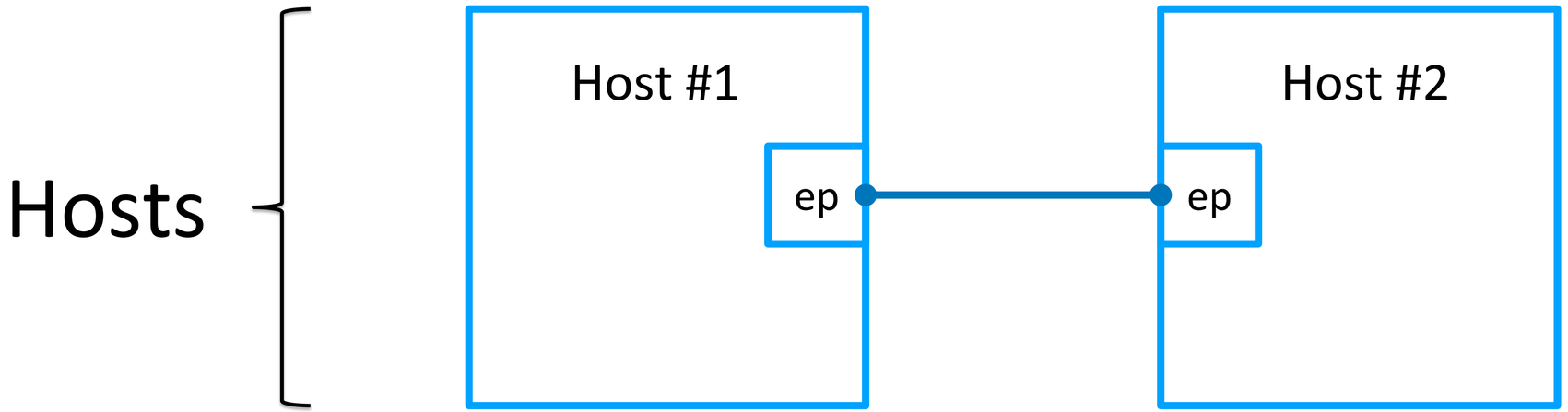
```
void ping_pong(int fd, char mode) {
        int i, m = 0;
        for (i = 0; i < _ITERATIONS_; ++i) {
                if (mode == 'c') {
                        printf("to   server > %02d\n", m);
                        send(fd, &m, sizeof(m), 0);
                        recv(fd, &m, sizeof(m), 0);
                        ++m;
                } else {
                        recv(fd, &m, sizeof(m), 0);
                        ++m;
                        printf("to   client > %02d\n", m);
                        send(fd, &m, sizeof(m), 0);
                }
        }
}
```


uthmanhere@thor013:~/omp_exp/labs_sc23/task_d/build$
uthmanhere@thor013:~/omp_exp/labs_sc23/task_d/build$
uthmanhere@thorbf3a013:~$ ./doca_openmp_service
connection established.
1
Server (Pong)
uthmanhere@thor014:~/omp_exp/labs_sc23/task_d/build$
uthmanhere@thor014:~/omp_exp/labs_sc23/task_d/build$
uthmanhere@thorbf3a014:~$ ./doca_openmp_service
connection established.
Client (Ping)
2

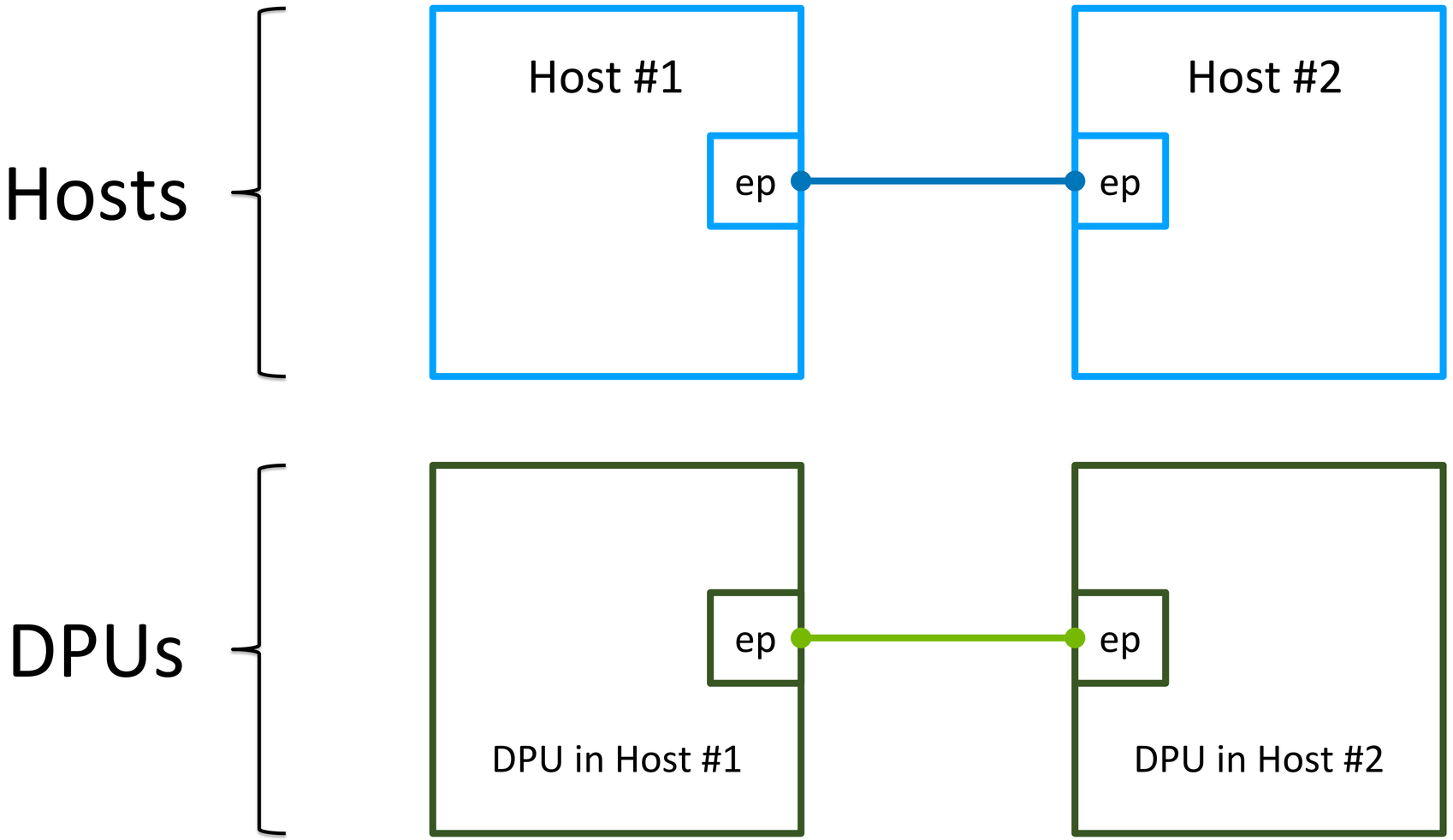
ODOS MPI

M. Usman, M. Benito, S. Iserte, and A. J. Peña, "ODOS-MPI: HPC-Friendly SmartNIC Offloading of Computation/Communication Kernels", in *SC25*
Wed. 10:30am Room 275

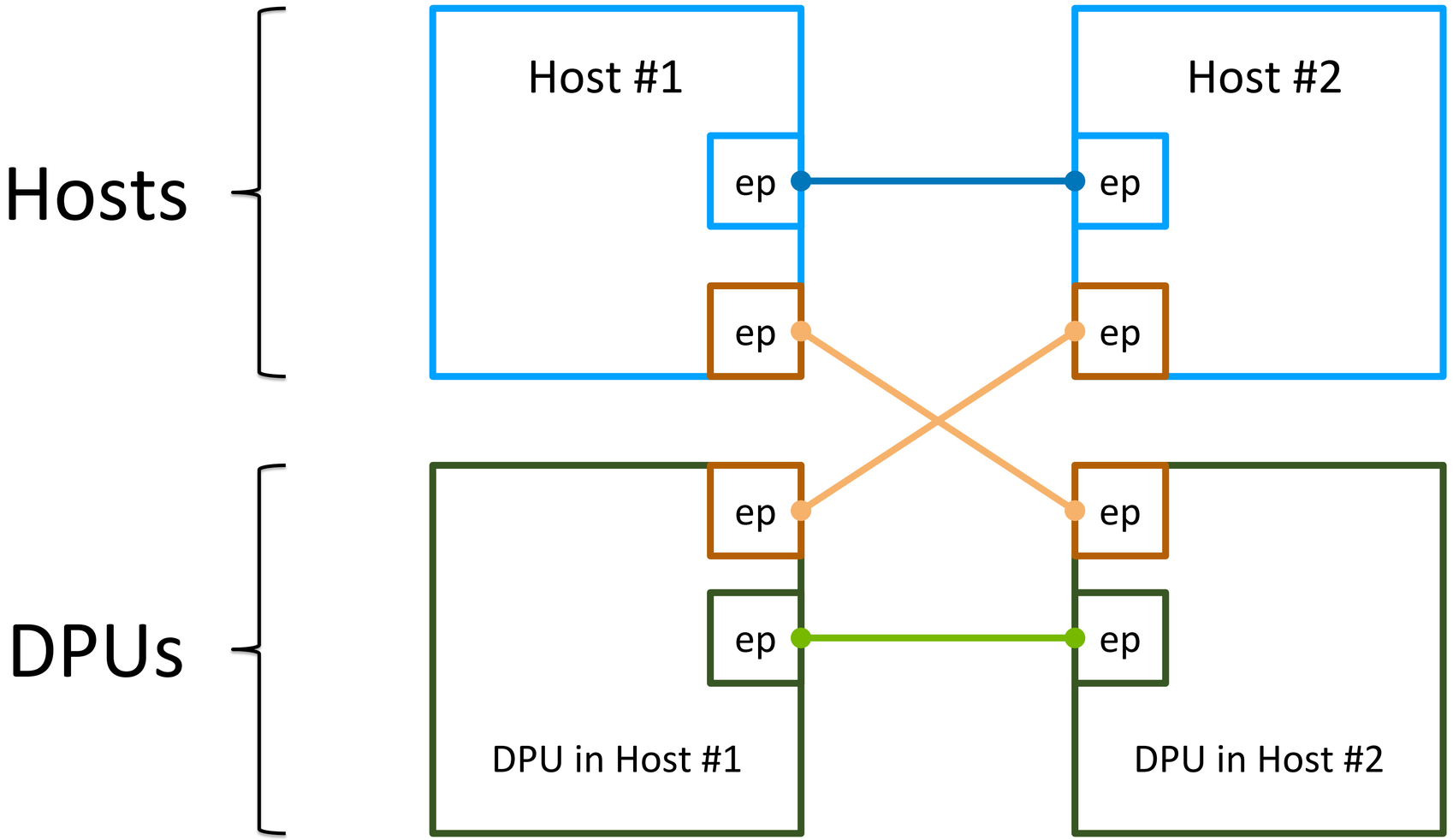
End-Points Links (1/3)



End-Points Links (2/3)



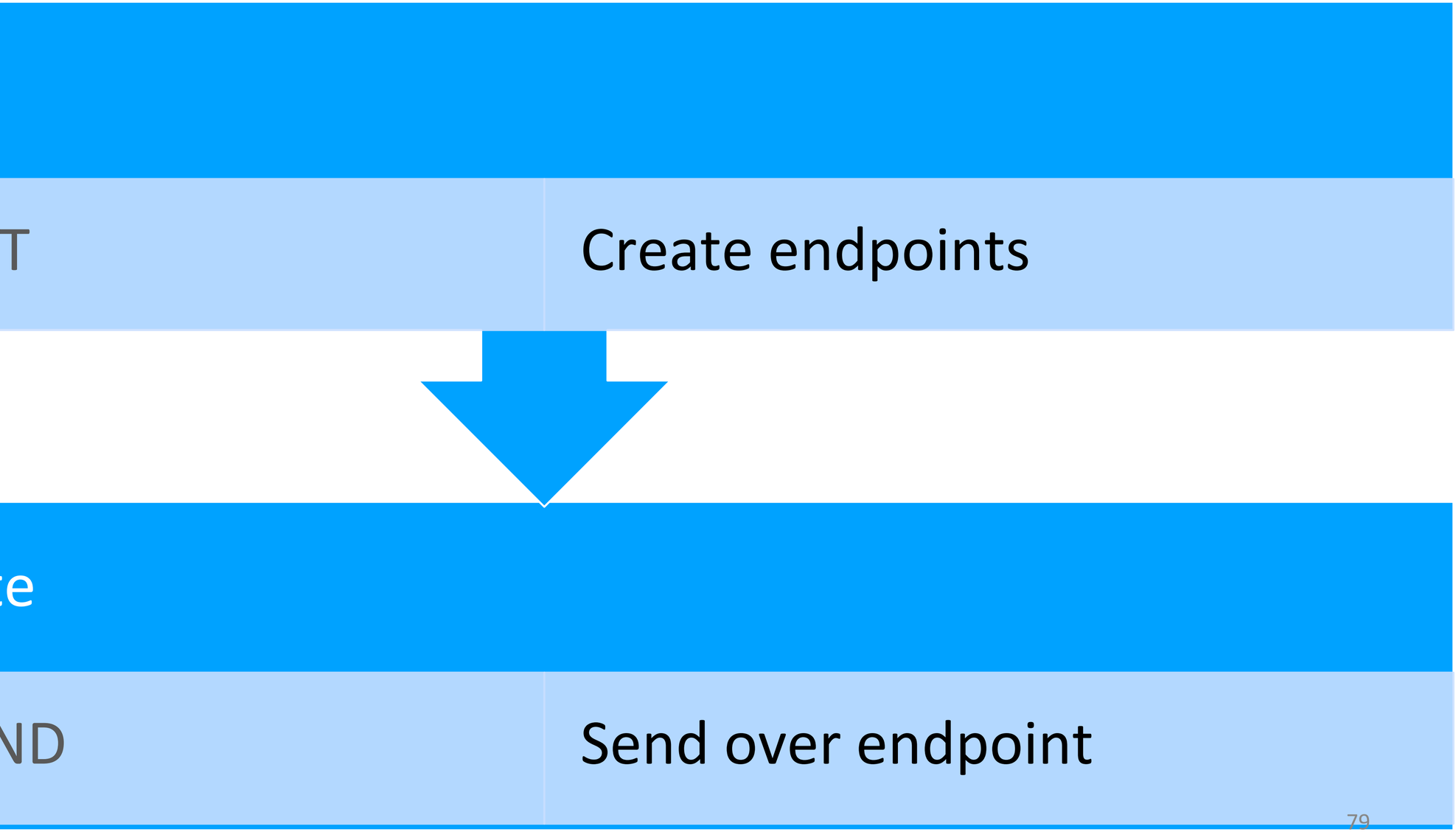
End-Points Links (3/3)



DPU as a local co-processor

- DPUs don’t have their own MPI rank number ☺

Open MPI for ODOS



DEMO #2

- MPI P2P
- MPI RMA

- **2 Nodes**
- **1 rank per node**
- **Ranks can execute commands in the host and in the DPU**

MPI P2P DPU-to-DPU

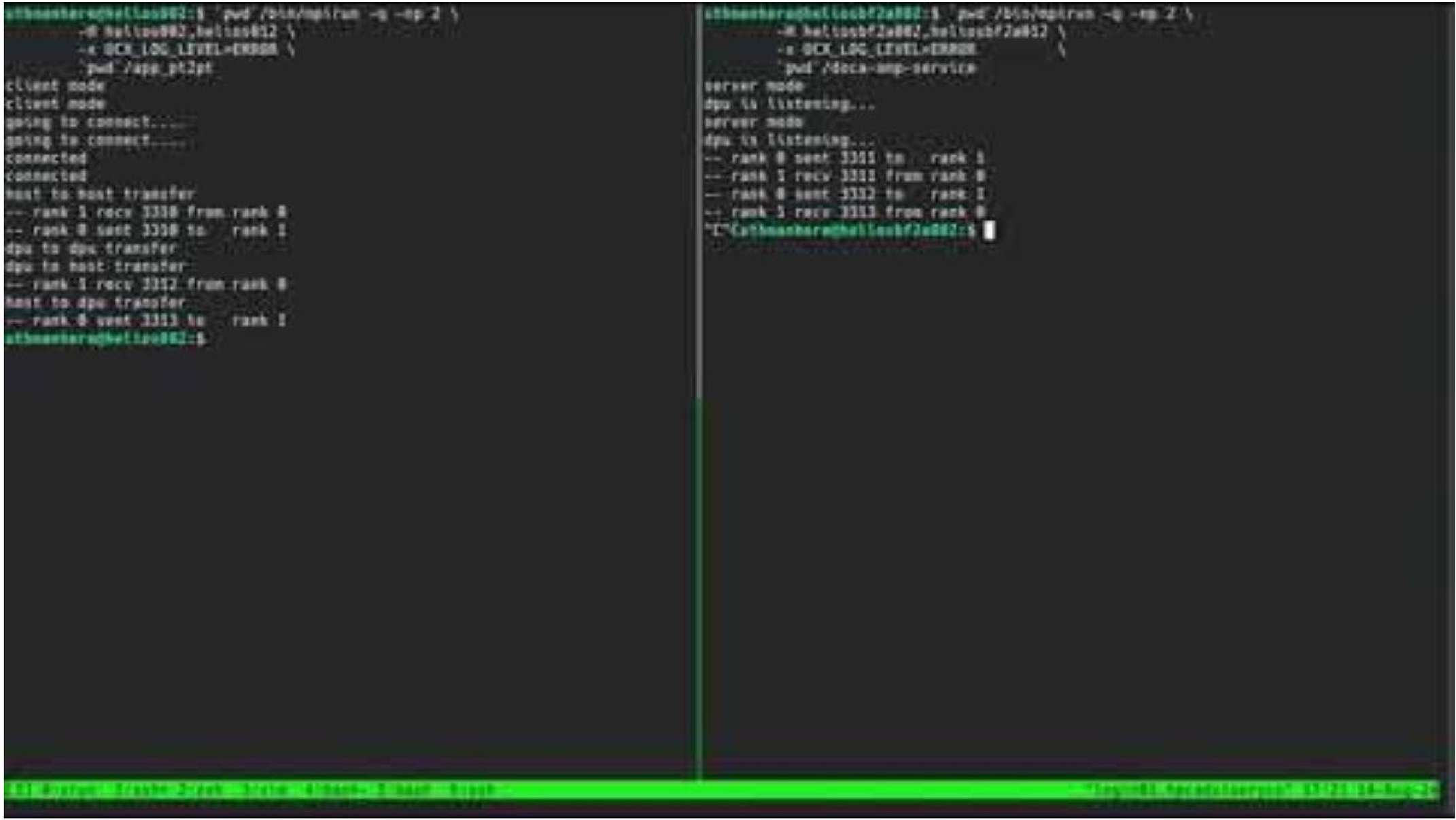
```
#pragma omp target
{
  if ( rank == 0 ) {
   MPI_Send( &buf, 1, MPI_INT, dst, 0, MPI_COMM_WORLD );
  } else {
   MPI_Recv( &recv_data, 1, MPI_INT, src, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE );
  }
}
```

MPI P2P DPU-to-Host

```
if ( rank == 0 ) {
    #pragma omp target
    {
        MPI_Info info;
        MPI_Info_create( &info );
        MPI_Info_set( info, "REMOTE_DEST_HETERO", "1" );
        MPI_Comm_set_info( MPI_COMM_WORLD, info );
        MPI_Send( &buf, 1, MPI_INT, dst, 0, MPI_COMM_WORLD );
        MPI_Info_delete( info, "REMOTE_DEST_HETERO" );
        MPI_Info_free( &info );
    }
} else {
    MPI_Recv( &recv_data, 1, MPI_INT, src, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE );
}
```

MPI P2P Example

- Host to host: "3310"
- Device to device: "3311"
- Device to host: "3312"
- Host to device: "3313"



```
uthmanhere@helios002:$ `pwd`/bin/mpirun -q -np 2 \
        -H helios002,helios012 \
        -x UCX_LOG_LEVEL=ERROR \
        `pwd`/app_pt2pt
client mode
client mode
going to connect....
going to connect....
connected
connected
host to host transfer
-- rank 1 recv 3310 from rank 0
-- rank 0 sent 3310 to   rank 1
dpu to dpu transfer
dpu to host transfer
-- rank 1 recv 3312 from rank 0
host to dpu transfer
-- rank 0 sent 3313 to   rank 1
uthmanhere@helios002:$
```

```
uthmanhere@heliosbf2a002:$ `pwd`/bin/mpirun -q -np 2 \
        -H heliosbf2a002,heliosbf2a012 \
        -x UCX_LOG_LEVEL=ERROR         \
        `pwd`/doca-omp-service
server mode
dpu is listening...
server mode
dpu is listening...
-- rank 0 sent 3311 to   rank 1
-- rank 1 recv 3311 from rank 0
-- rank 0 sent 3312 to   rank 1
-- rank 1 recv 3313 from rank 0
^C^Cuthmanhere@heliosbf2a002:$
```

MPI Collectives across DPUs

```
#pragma omp target
{
    MPI_Alltoall(send_buffer, 1, MPI_INT, recv_buffer, 1, MPI_INT, MPI_COMM_WORLD);
}
```

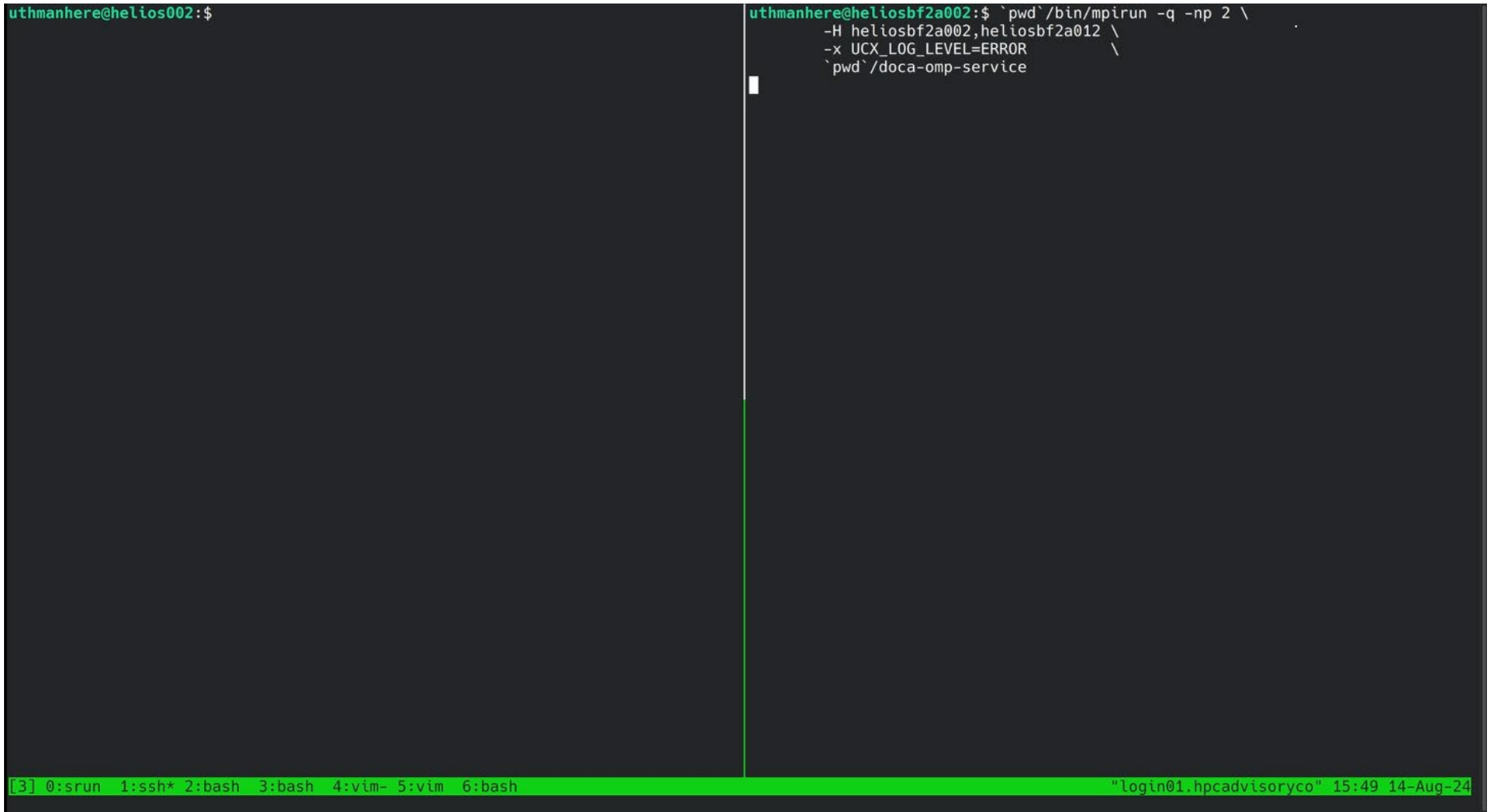
MPI RMA Host-to-DPU

```
#pragma omp target nowait
{
  MPI_Win_create(&window_data_dpu, sizeof(int), sizeof(int), MPI_INFO_NULL, MPI_COMM_WORLD, &win);
  MPI_Win_fence(0, win);
  MPI_Win_free(&win);
}

MPI_Info info;
MPI_Info_create(&info);
MPI_Win_create(&window_data, sizeof(int), sizeof(int), info, MPI_COMM_WORLD, &win);

if (rank == src) {
    MPI_Info_set(info, "REMOTE_DEST_HETERO", "1");
    MPI_Win_set_info(win, info);
    MPI_Put(&data, 1, MPI_INT, dst, 0, 1, MPI_INT, win);
    MPI_Info_delete(info, "REMOTE_DEST_HETERO");
 }

 MPI_Win_fence(0, win);
 MPI_Win_free(&win);
 MPI_Info_free(&info);
```

Host-DPU Simultaneous RMA Communication (1/4)

Simultaneous one-sided cross-communication:

- Rank#0_Host to Rank#1_DPU
- Rank#0_DPU to Rank#1_Host



Host-DPU Simultaneous RMA Communication (2/4)

```
uthmanhere@helios002:$ `pwd`/bin/mpirun -q -np 2 \
        -H helios002,helios012 \
        -x UCX_LOG_LEVEL=ERROR \
        `pwd`/app_osc
client mode
going to connect....
client mode
going to connect....
connected
connected
```

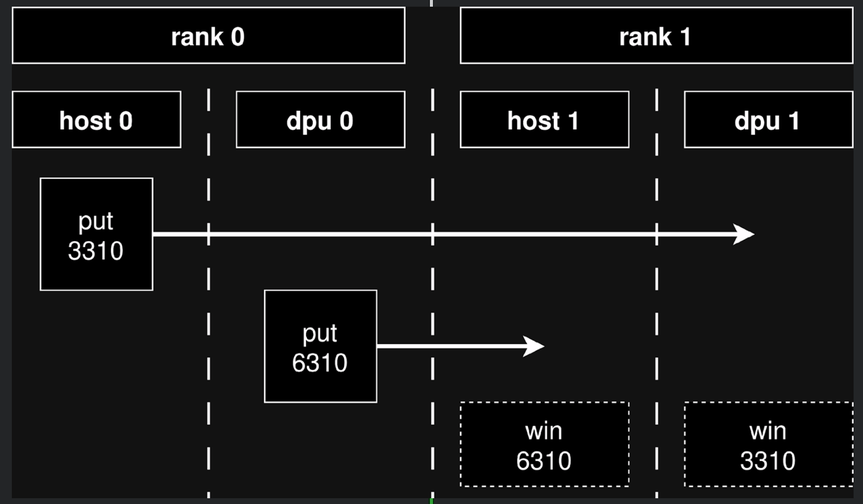
```
uthmanhere@heliosbf2a002:$ `pwd`/bin/mpirun -q -np 2 \
        -H heliosbf2a002,heliosbf2a012 \
        -x UCX_LOG_LEVEL=ERROR         \
        `pwd`/doca-omp-service
server mode
dpu is listening...
server mode
dpu is listening...
```

```
[3] 0:srun  1:ssh* 2:bash  3:bash  4:vim- 5:vim  6:bash                    "login01.hpcadvisoryco" 15:49 14-Aug-24
```

Host-DPU Simultaneous RMA Communication (3/4)

```
uthmanhere@helios002:$ `pwd`/bin/mpirun -q -np 2 \
        -H helios002,helios012 \
        -x UCX_LOG_LEVEL=ERROR \
        `pwd`/app_osc
client mode
going to connect....
client mode
going to connect....
connected
connected
>>> process 1 got 6310 from rank 0 by put
```

```
uthmanhere@heliosbf2a002:$ `pwd`/bin/mpirun -q -np 2 \
        -H heliosbf2a002,heliosbf2a012 \
        -x UCX_LOG_LEVEL=ERROR         \
        `pwd`/doca-omp-service
server mode
dpu is listening...
server mode
dpu is listening...
>>> process 1 got 3310 from rank 0 by put
```

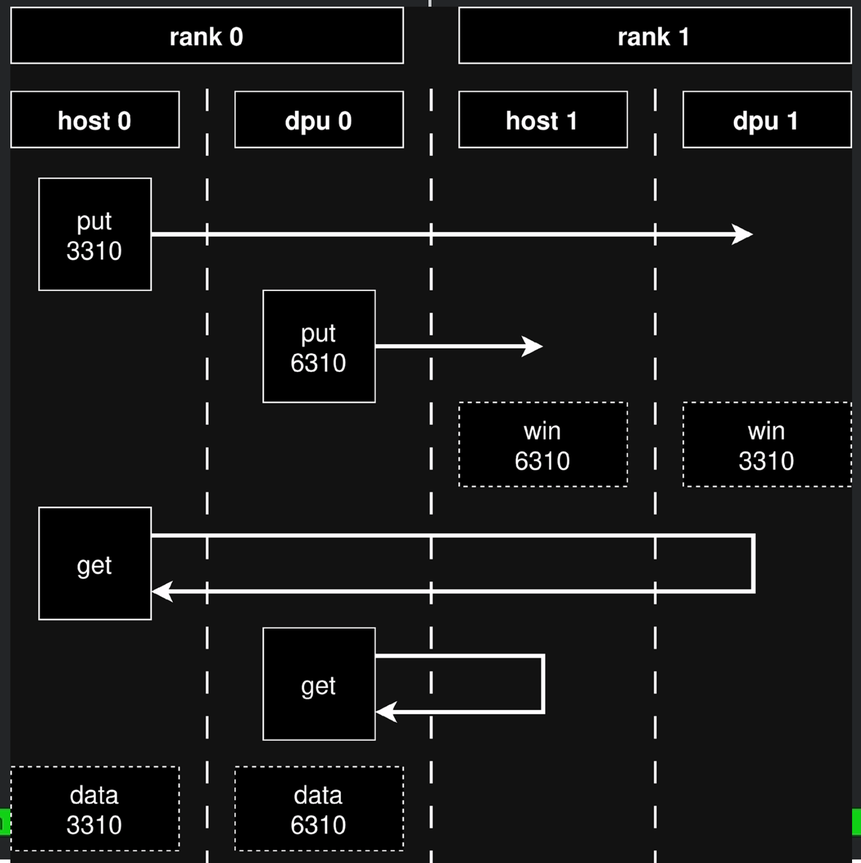



```
[3] 0:srun  1:ssh* 2:bash  3:bash  4:vim- 5:vim  6:bash                    "login01.hpcadvisoryco" 15:49 14-Aug-24
```

Host-DPU Simultaneous RMA Communication (4/4)

```
uthmanhere@helios002:$ `pwd`/bin/mpirun -q -np 2 \
        -H helios002,helios012 \
        -x UCX_LOG_LEVEL=ERROR \
        `pwd`/app_osc
client mode
going to connect....
client mode
going to connect....
connected
connected
>>> process 1 got 6310 from rank 0 by put
>>> process 0 got 3310 from rank 1 by get
uthmanhere@helios002:$
```

```
uthmanhere@heliosbf2a002:$ `pwd`/bin/mpirun -q -np 2 \
        -H heliosbf2a002,heliosbf2a012 \
        -x UCX_LOG_LEVEL=ERROR         \
        `pwd`/doca-omp-service
server mode
dpu is listening...
server mode
dpu is listening...
>>> process 1 got 3310 from rank 0 by put
>>> process 0 got 6310 from rank 1 by get
```

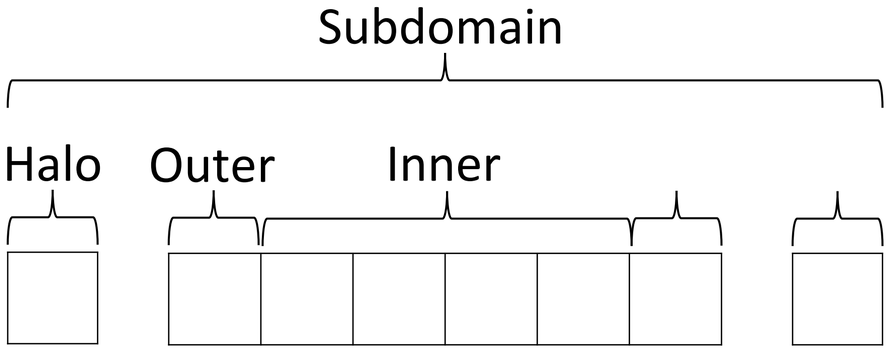


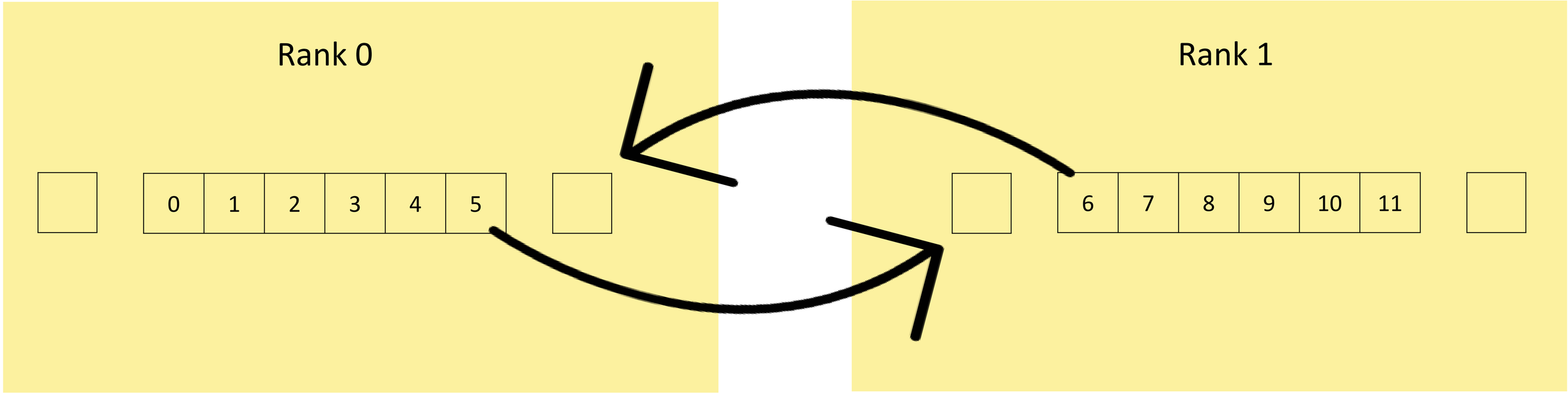

```
[3] 0:srun  1:ssh* 2:bash  3:bash  4:vim- 5:vim                    "login01.hpcadvisoryco" 15:49 14-Aug-24
```

Example: Halo Exchange

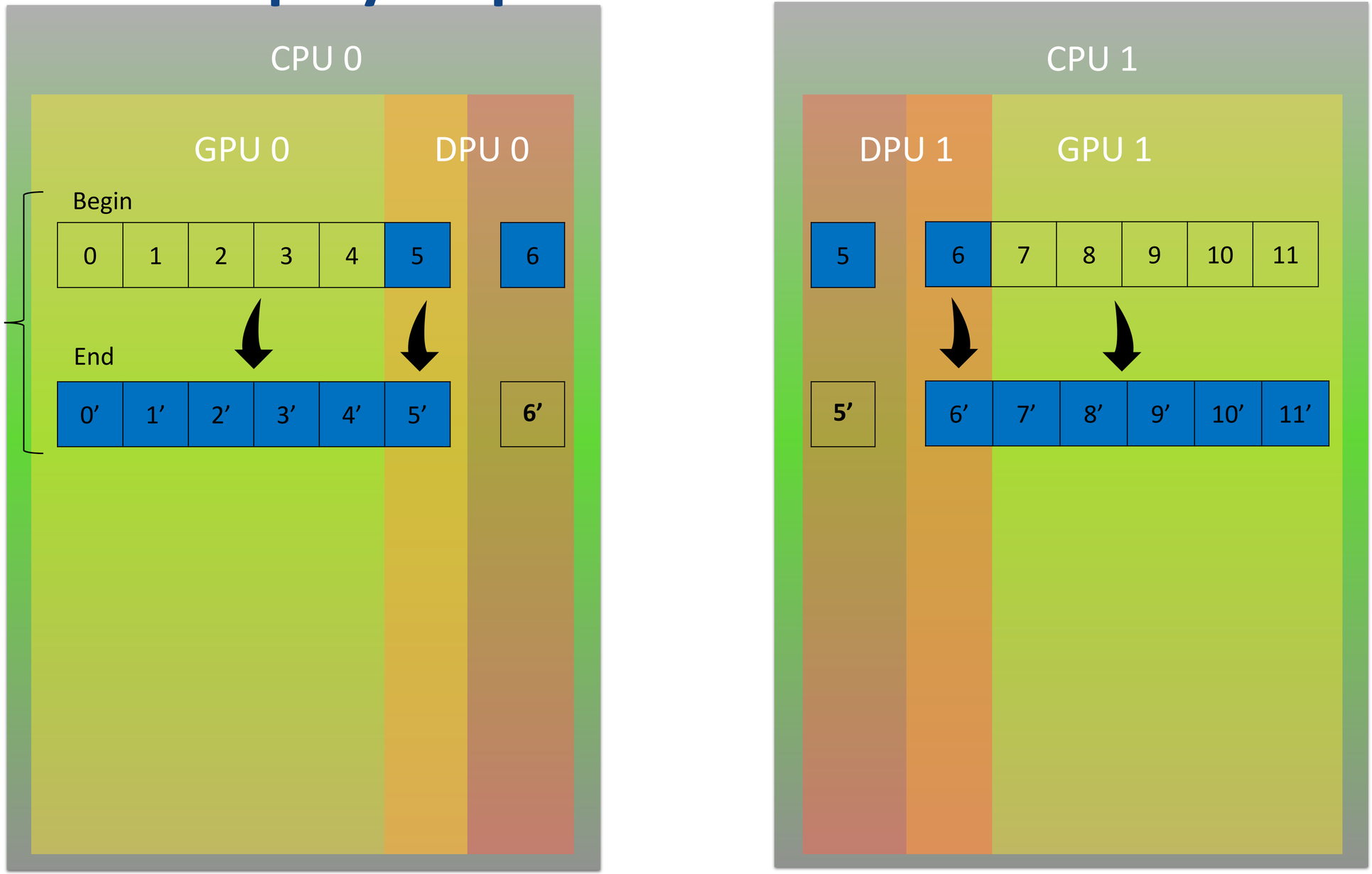
Halo Exchange Offloading

SIZE = 8
THICKNESS = 1

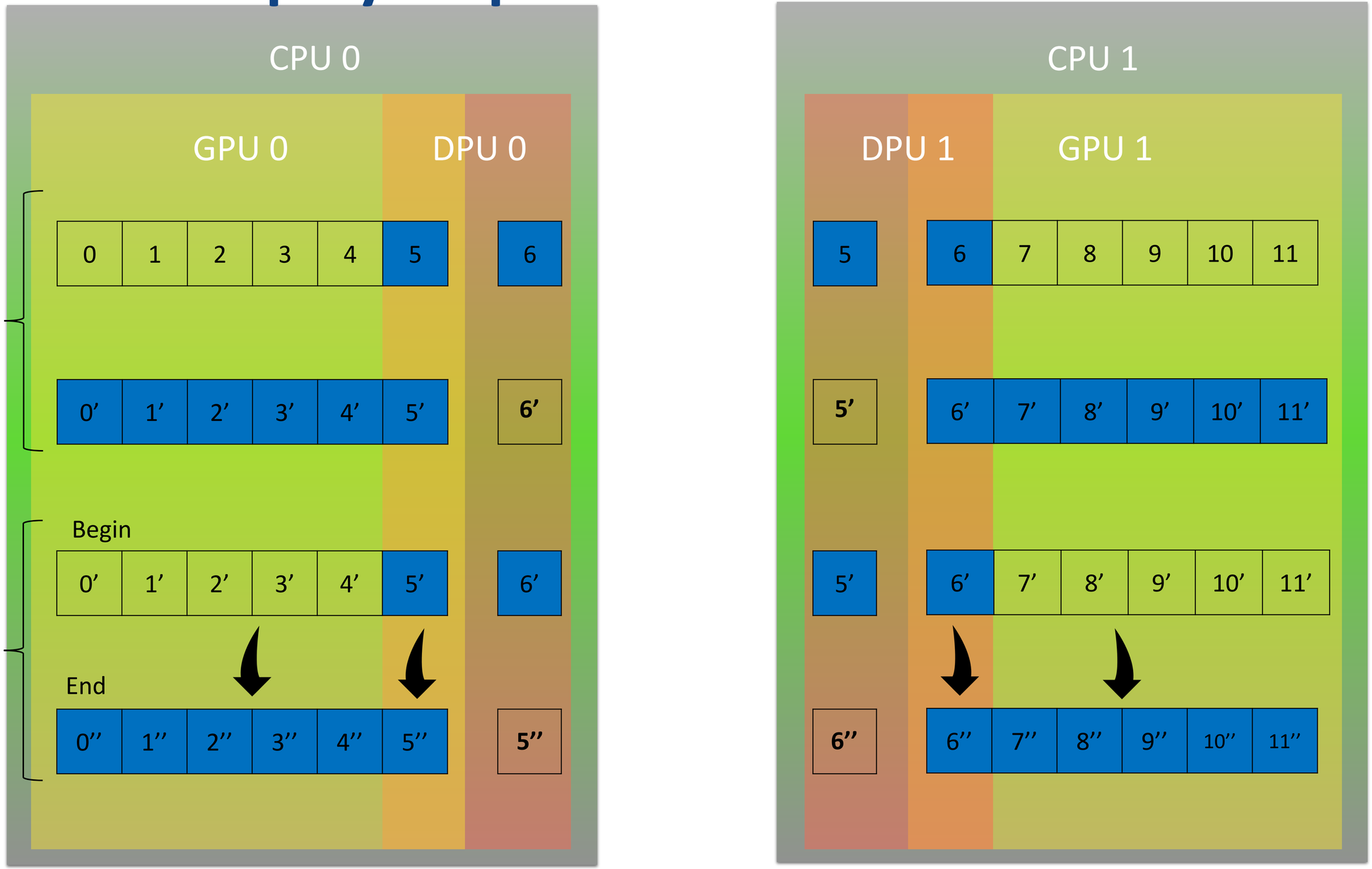




Execution step by step



Execution step by step



```
int curr_subdomain[SIZE + THICKNESS * 2];
int next_subdomain[SIZE + THICKNESS * 2];
#pragma omp target data map(to: curr_subdomain[THICKNESS:SIZE-THICKNESS])
                        map(from: next_subdomain[THICKNESS:SIZE-THICKNESS])
                        device(GPU_ID) nowait
{
    #pragma omp target teams num_teams(2)
    {
        #pragma omp parallel
        {
            next_subdomain[THICKNESS] = compute_inner(curr_subdomain[THICKNESS:SIZE-THICKNESS])
        }
    }
}
#pragma omp target data map(to: curr_subdomain[0:THICKNESS*2], curr_subdomain[SIZE-THICKNESS*2:SIZE]
                        map(from: next_subdomain[0:THICKNESS*2], next_subdomain[SIZE-THICKNESS*2:SIZE])
                        device(DPU_ID) nowait
{
    #pragma omp parallel
    #pragma omp single
    {
        #pragma omp task
        {
            next_subdomain[THICKNESS] = compute_outer(curr_subdomain[0:THICKNESS*2], left);
        }
        #pragma omp task
        {
            next_subdomain[SIZE-THICKNESS*2] = compute_outer(curr_subdomain[SIZE-THICKNESS*2:SIZE], right);
        }
    }
}
#pragma omp barrier
curr_subdomain = copy_domain(next_subdomain);
```

GPU

DPU

Offloaded Function to the DPU

```
int* compute_outer(int* data, int rank_id)
{
    int halo[THICKNESS], outer_domain[THICKNESS], tag_id = 0;
    MPI_Request request;
    MPI_Status status;

    MPI_Irecv(halo, THICKNESS, MPI_INT, rank_id, tag_id, MPI_COMM_WORLD, &request);
    outer_domain = compute(data);
    MPI_Send(outer_domain, THICKNESS, MPI_INT, rank_id, tag_id, MPI_COMM_WORLD);
    MPI_Wait(&request, &status);
    return outer_domain;
}
```

Performance Evaluation

Testbed

Jupiter

- Intel Xeon 10-core E5-2680, 64GB DD3
- NVIDIA BlueField-2 HDR 100Gbps: 8 ARMv8 A72 cores, 16GB DDR4

Thor

- 2x Intel Xeon 16-core E5-2697, 256GB DD4
- NVIDIA BlueField-3 NDR 200Gbps: 16 ARMv8.2 A78 cores, 16GB DDR5

HPC-AI ADVISORY COUNCIL

NETWORK OF EXPERTISE

Software Stack

OpenMP 5.0

LLVM 14.0.6

DOCA 2.0.2

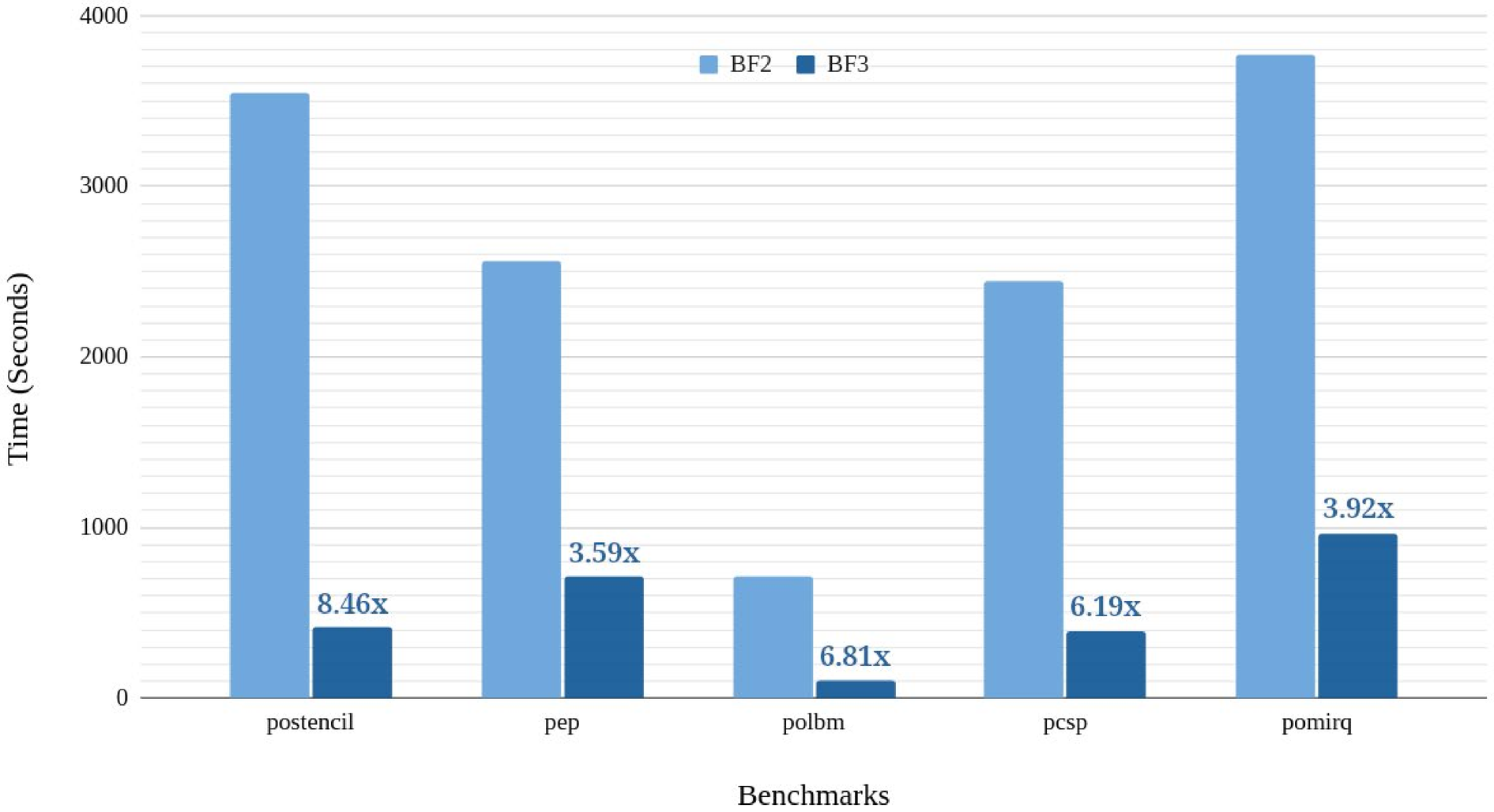
OpenMPI 4.1.6

SPEC ACCEL Benchmark Suite (OpenMP target Offloading)

- Removed OpenMP “distribute” construct for GPUs
- Added “simd” construct for enabling SIMD instructions

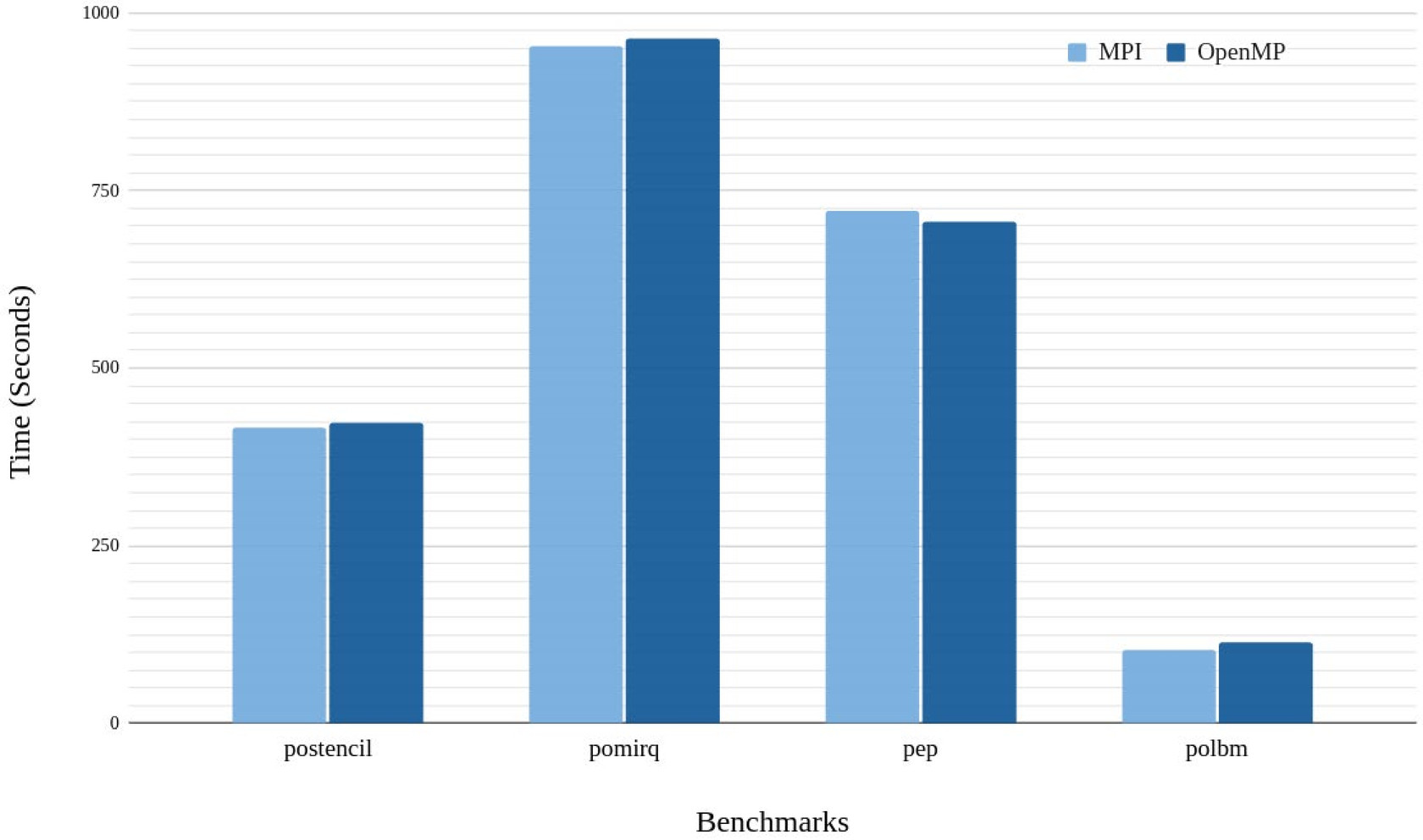
OSU Microbenchmarks (variation supporting DOCA)

BlueField-2 and BlueField-3 Performance Comparison



M. Usman, S. Iserte, R. Ferrer, and A. J. Peña, "DPU Offloading Programming with the OpenMP API." *LLVM-HPC23 co-located with SC23,* https://dl.acm.org/doi/10.1145/3624062.3624165

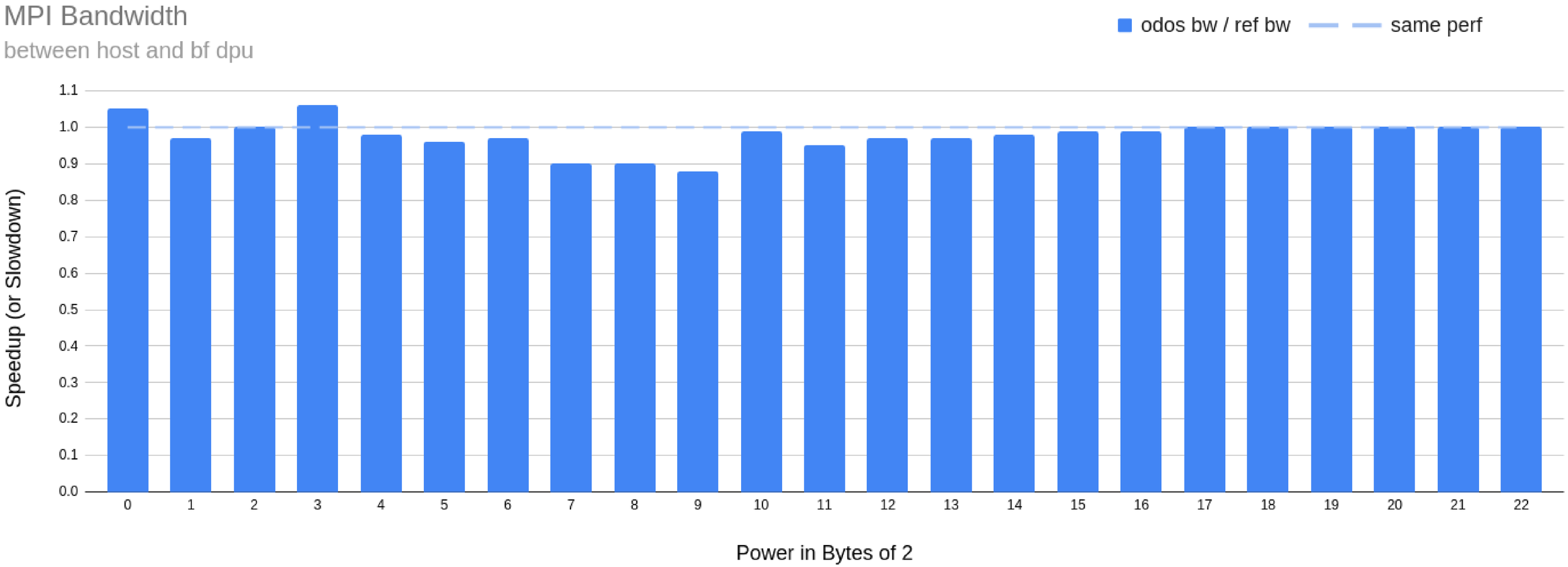
OpenMP and MPI Performance Comparison



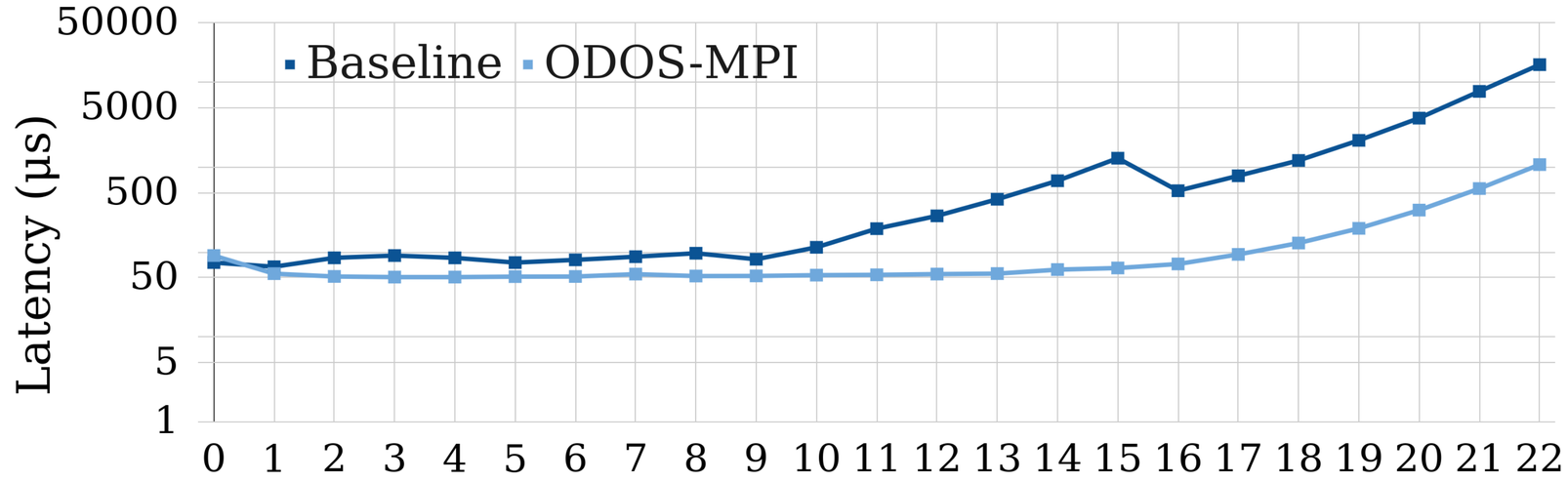
M. Usman, S. Iserte, R. Ferrer, and A. J. Peña, "DPU Offloading Programming with the OpenMP API." *LLVM-HPC23 co-located with SC23,* https://doi.org/10.1145/3624062.3624165

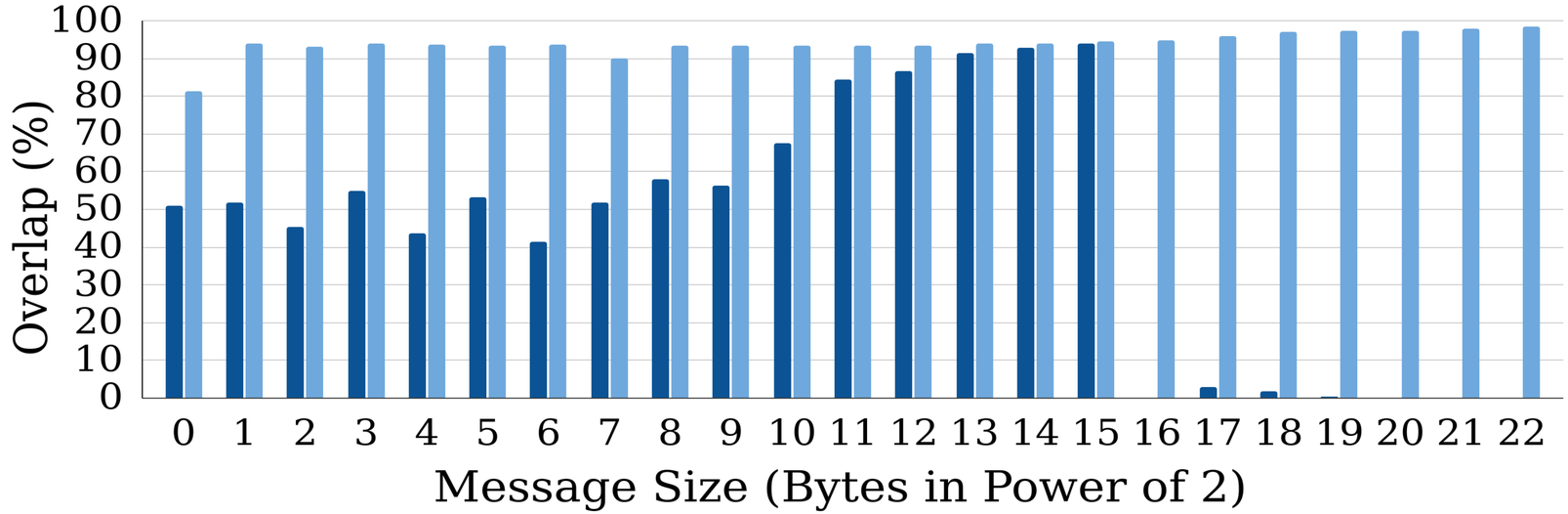
MPI vs ODOS MPI Offloading

Custom version of OSU to offload MPI to DPU

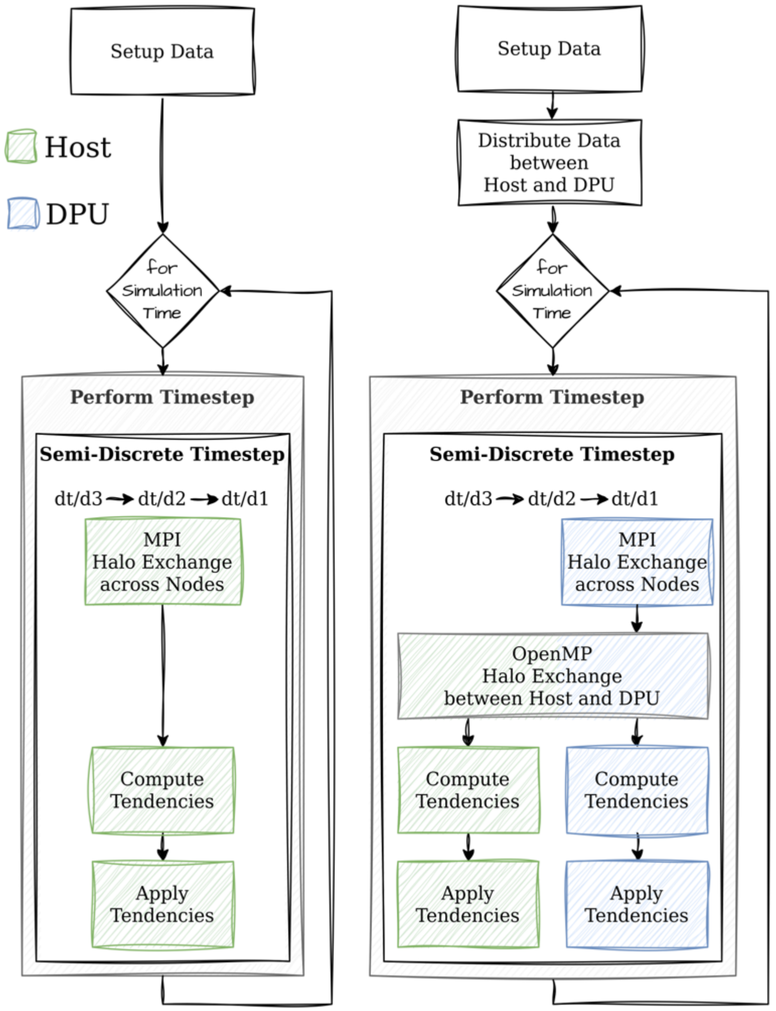


OSU MB: Computation-Communication Overlap

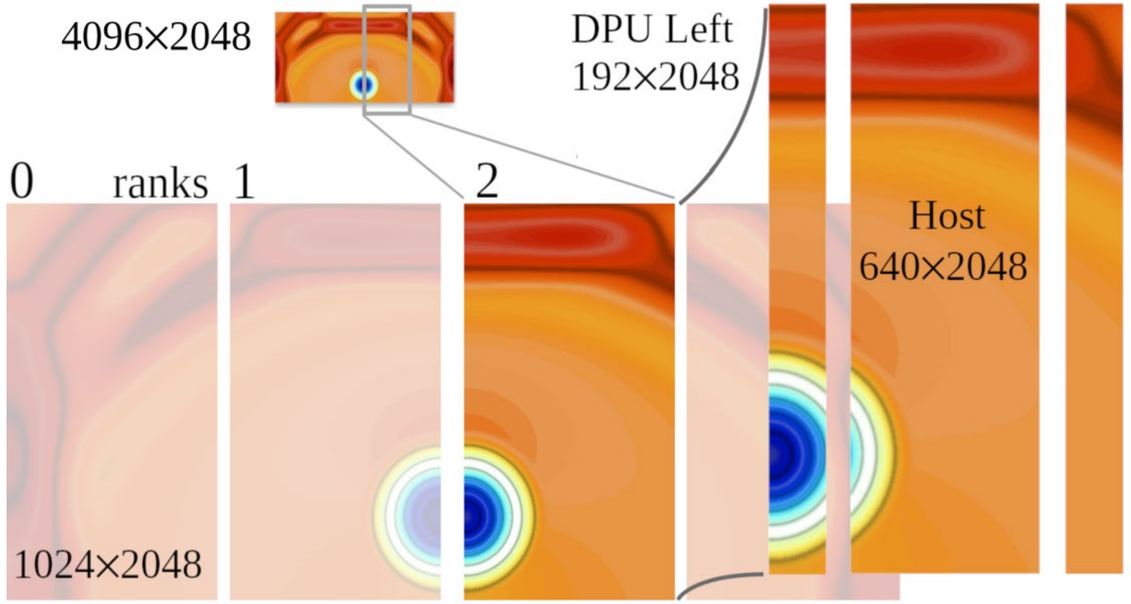




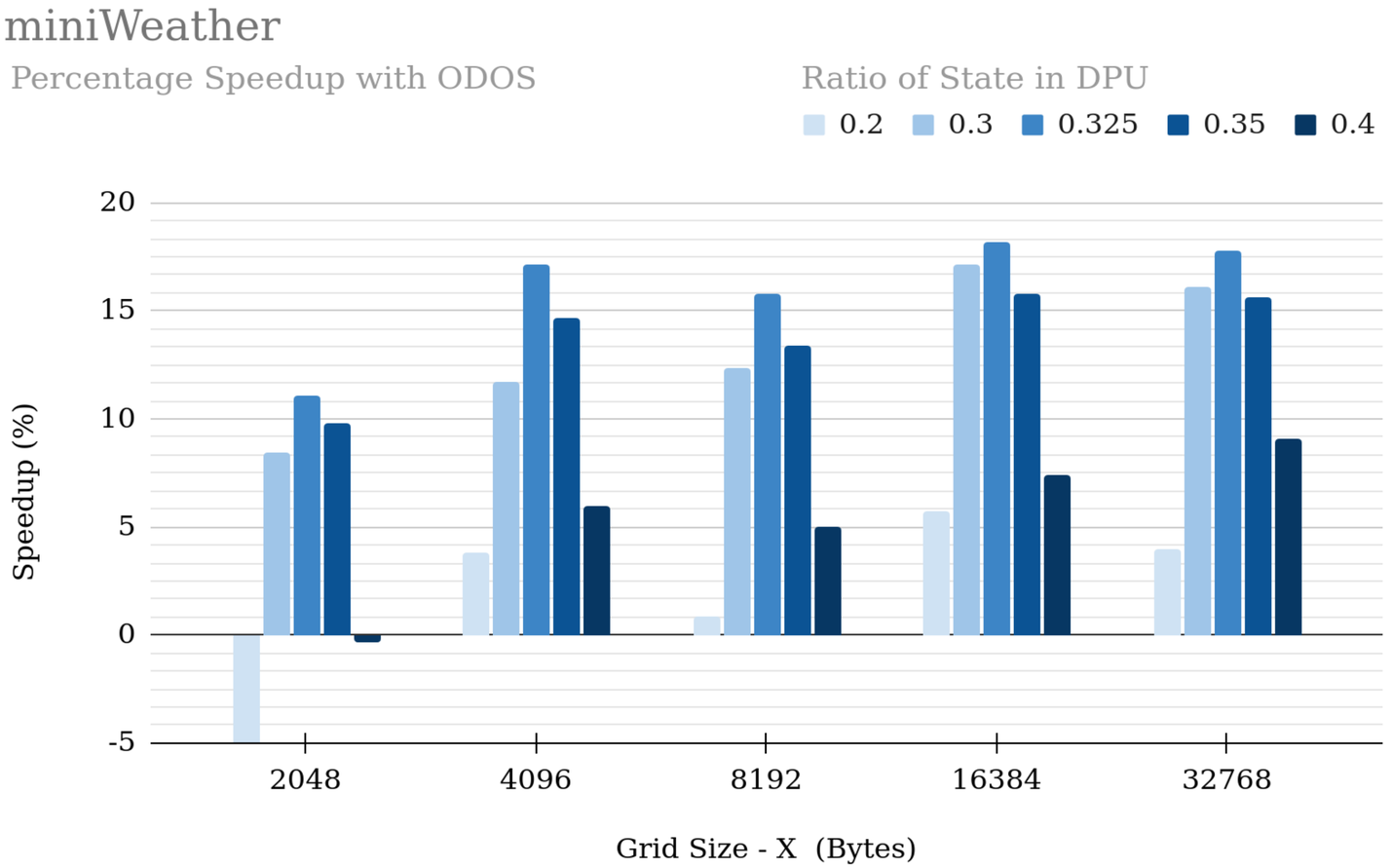
miniWeather Application



- Simulates 2-D inviscid Euler equations for stratified fluid dynamics
- Using Runge-Kutta Time Integration method



miniWeather: Percentage Speedup with ODOS-MPI



M. Usman, M. Benito, S. Iserte, and A. J. Peña, "ODOS-MPI: HPC-Friendly SmartNIC Offloading of Computation/Communication Kernels", in *SC25*

Wed. 10:30am Room 275

Break

Time (CST)	Topic	Presenters
1:30 - 1:40	Introduction and Attendee Survey	
1:40 - 2:00	SmartNIC and DPU Background and Overview	Jeff, Elie
2:00-2:20	SmartNIC Applications and Use Cases	Jeff, Oscar
2:20 - 3:00	SmartNIC SW Infrastructure – HPC Programming Approaches: MPI and OpenMP Offload	Rich, Toni
3:00 - 3:30	**BREAK**	
3:30-4:15	SmartNIC SW Infrastructure – APIs and Frameworks: DPA Programming, DOCA and P4	Rich, Elie, Jorge
4:15-5:00	Hands on with DOCA and P4, DPA examples	All
4:45	Tutorial Survey / Wrap-up	

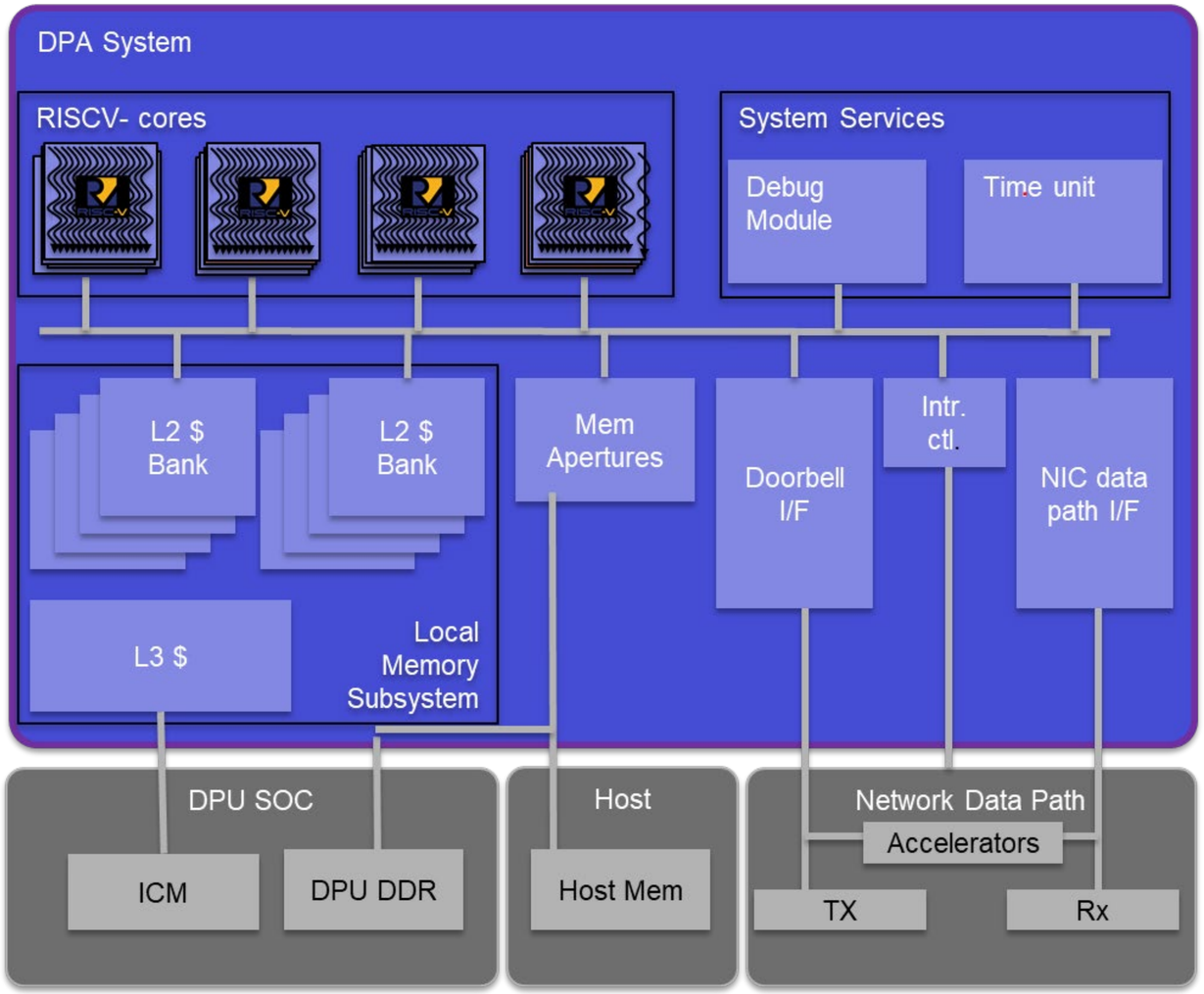
See the updated agenda at **https://github.com/gt-crnch-rg/smartnic-tutorial-sc25**

SmartNIC SW Infrastructure – APIs and Frameworks

NVIDIA’s Data Path Accelerator (DPA)

Data Path Accelerator (DPA)

- RV64IMAC(B) + NVIDIA extensions (e.g. arithmetic ops)
- Part of the Connect-X network core
- High BW Dedicated local cluster cache
- Access to Host system memory
- NIC direct access to DPA caches
- NIC interrupts and doorbells
- Full access to accelerators
- Standard Debug Module and timer unit
- DPA-RTOS
  - Low memory footprint, highly threaded
  - Low latency scalable HW based scheduling
  - Process isolation enforced



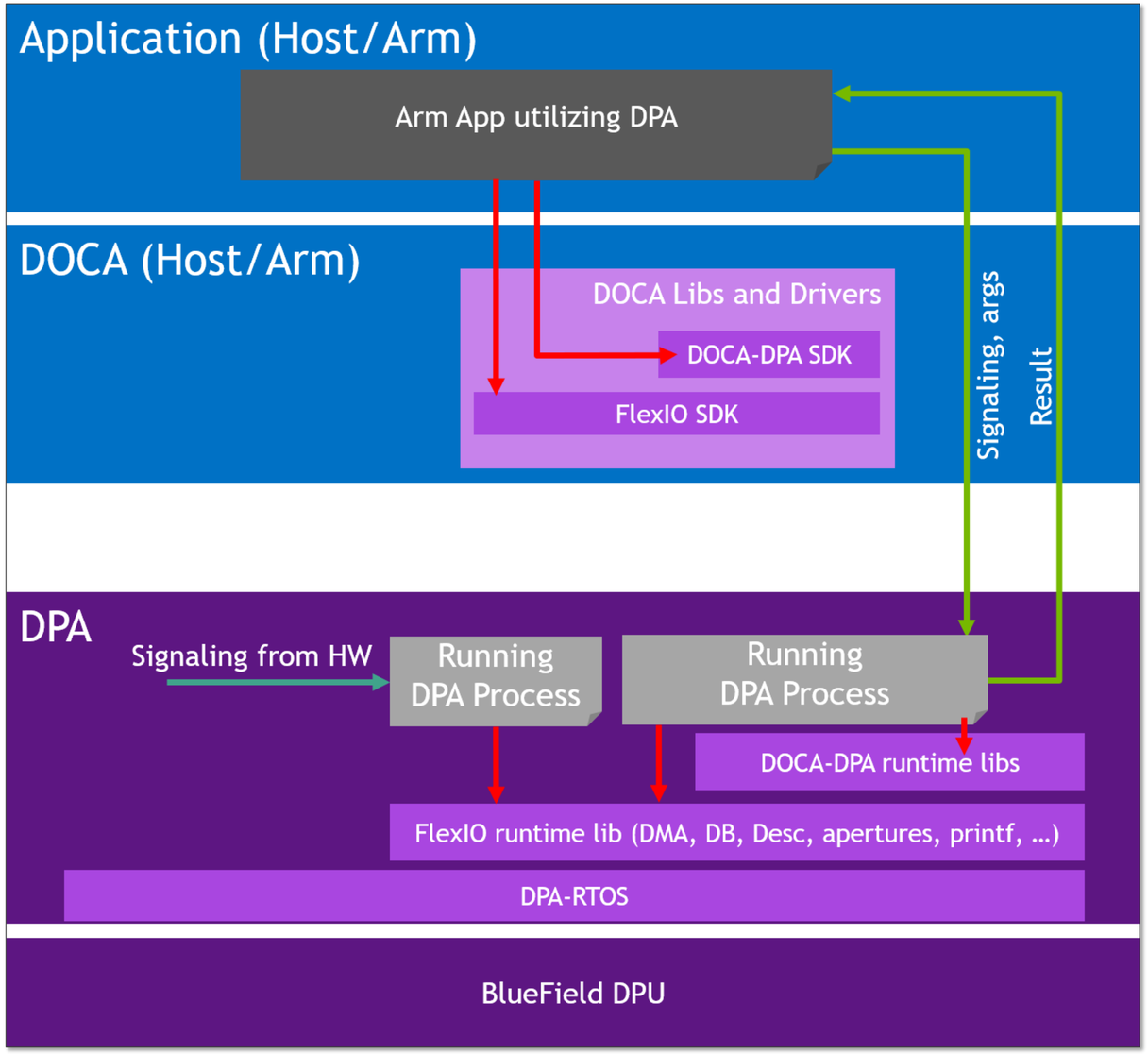
DPA highlights – Bluefield-3 (ConnectX-7 core)

DPA			
Architecture		Standard RISC-V RV64IMAC(B)-USM	
Pipeline Width		1 Instruction / Clock	
Threads per Core		16	
Number of Cores		16	
Frequency Target		Core 1.8 GHz / System 505 MHz	
Caches		Capacity	Access / Clock
	L1 Code / Core	8 KB (2-4K inst.)	1 (per Core)
	L1 Data / Thread	1 KB	1 (per Core)
	L2 / Cluster	1.5 MB	8
	L3 / Cluster	3 MB	2
	ICMC / NIC	4 MB	2

DPAs – More Details

- DPAs have a limited amount of memory
- DPA threads cannot run for more than a few seconds
- Custom micro-kernel
- No libc
  - No dynamic memory allocation from device
  - No standard mechanisms that most users rely on

- FlexIO is the lowest-level library for programming DPAs
  - Processes on the host are responsible for allocating device memory
  - Processes on the host start kernels (viaDOCA FlexIO)
- It is possible to activate DPA threads on demand



DPA Execution Model

- Collective execution initiated on host (MPI ranks) but executed on the DPAs

- Collective algorithms consist of control messages and data transfers
  - Control messages: coordination between ranks running on the hosts and DPA threads required by the algorithms
  - Data transfer: move the application's data

- Short-lived DPA threads
  - Send & receive completions trigger the execution of a thread on the DPA
  - Rely on persistent data accessible on/from the DPA to track & manage the state of the local collective operations
  - When messages generated by the implementation of the collective algorithms complete, the local memory is updated to track the state of the said collective
  - When the collective completes, a completion is generated for all the ranks

Programming DPAs

Two software packages are available

- FlexIO: low-level library supporting both the host and device programming
  - https://docs.nvidia.com/doca/api/3.0.0/flexio-sdk-api/index.html
- DOCA DPA: based on FlexIO but provides useful abstractions for the implementation of new collective algorithms
  - https://docs.nvidia.com/doca/sdk/doca+dpa/index.html

Similar to GPU programming in some ways:

- A kernel is started on the GPU
- The kernel code must be static (e.g., no dynamic memory allocation)
- The kernel code is “compiled in” the host-code using the dpacc compiler and linking all binaries together
- FlexIO handles the bootstrapping and execution of kernel codes

DPA specificities

- Kernels can be implemented to activate threads only upon completion of network operations
- The limited amount of memory imposes limitations on designing collective algorithms

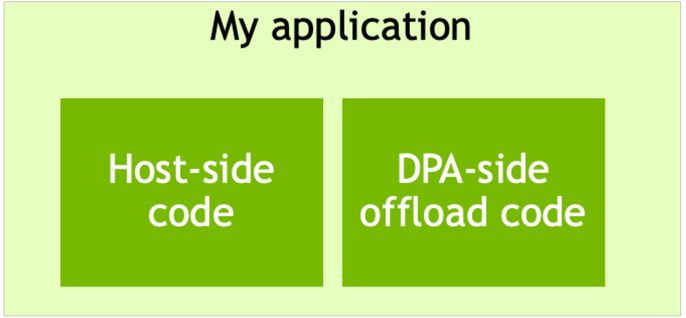


Figure 1: illustration of software DPA offloading

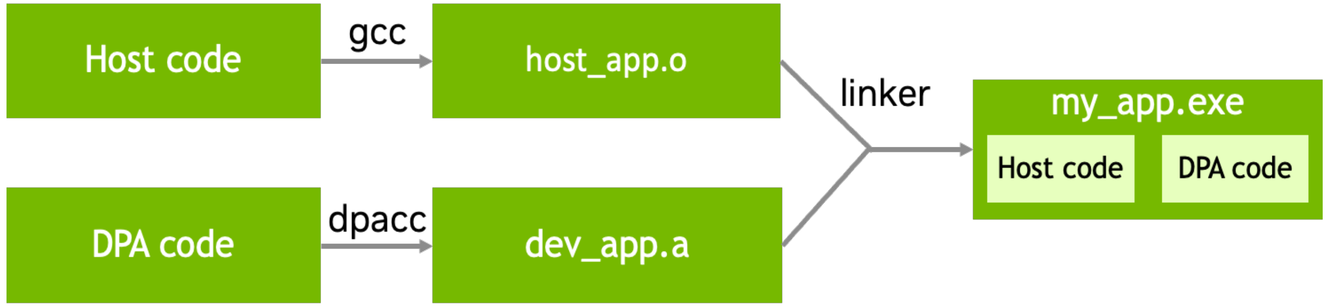
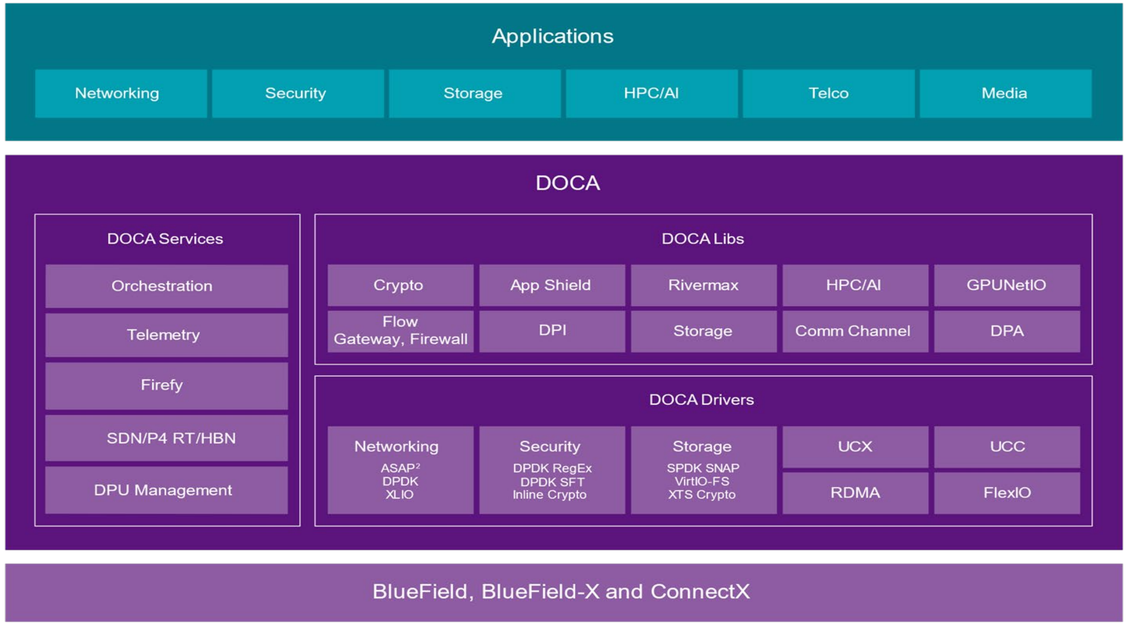


Figure 2: example of how the offload code is compiled and packaged in a single binary

DOCA Programming

DOCA Programming

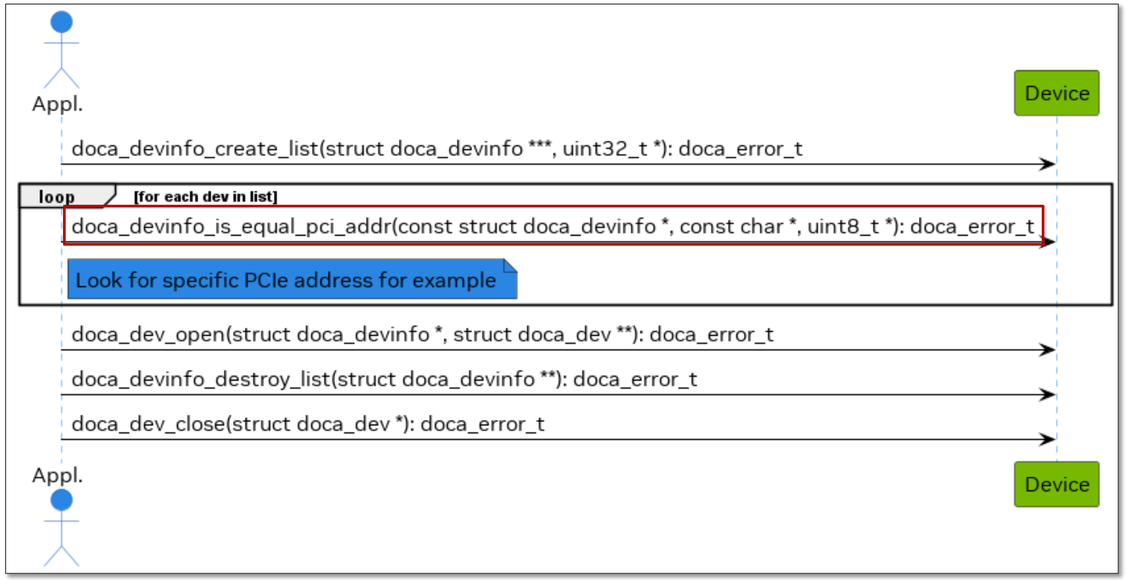
- DOCA exposes APIs to invoke the accelerators (e.g., decompression, direct memory access (DMA), crypto, etc.)
- The workflow consists of:
  1. Device discovery
  2. Create memory map (MMAP)
  3. Create & start a buffer inventory
  4. Create a Progress Engine (PE)
  5. Create and configure the accelerator instance
  6. Allocate and submit a task
  7. Drive progress & handle completion
  8. Cleanup



Device Discovery

To work with DOCA libraries, an application must open a device available on the DPU

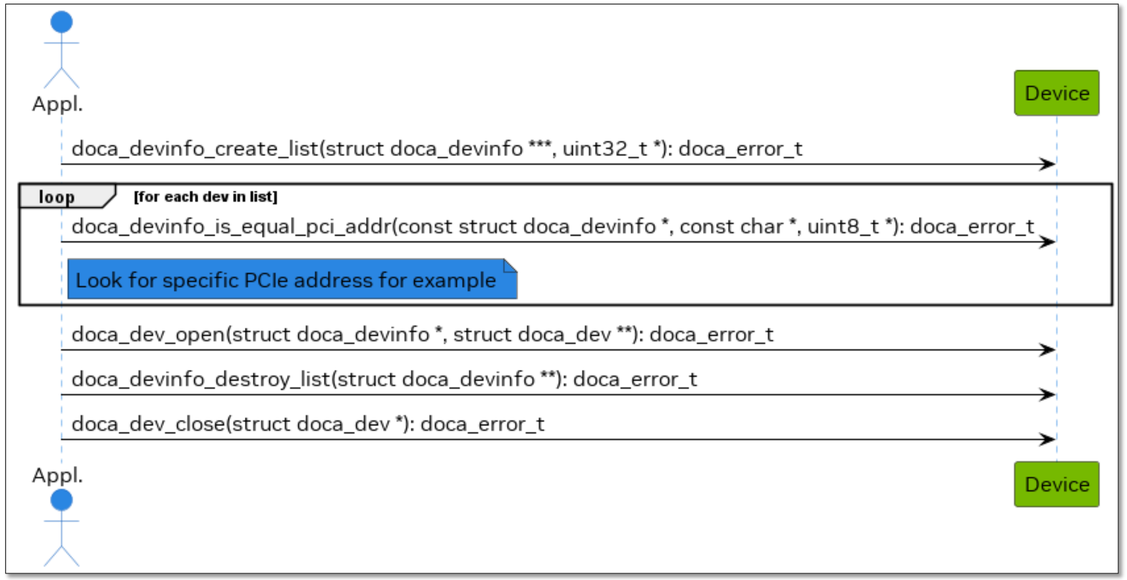
Open Device by PCI address



Device Discovery

To work with DOCA libraries, an application must open a device available on the DPU

Open Device by PCI address

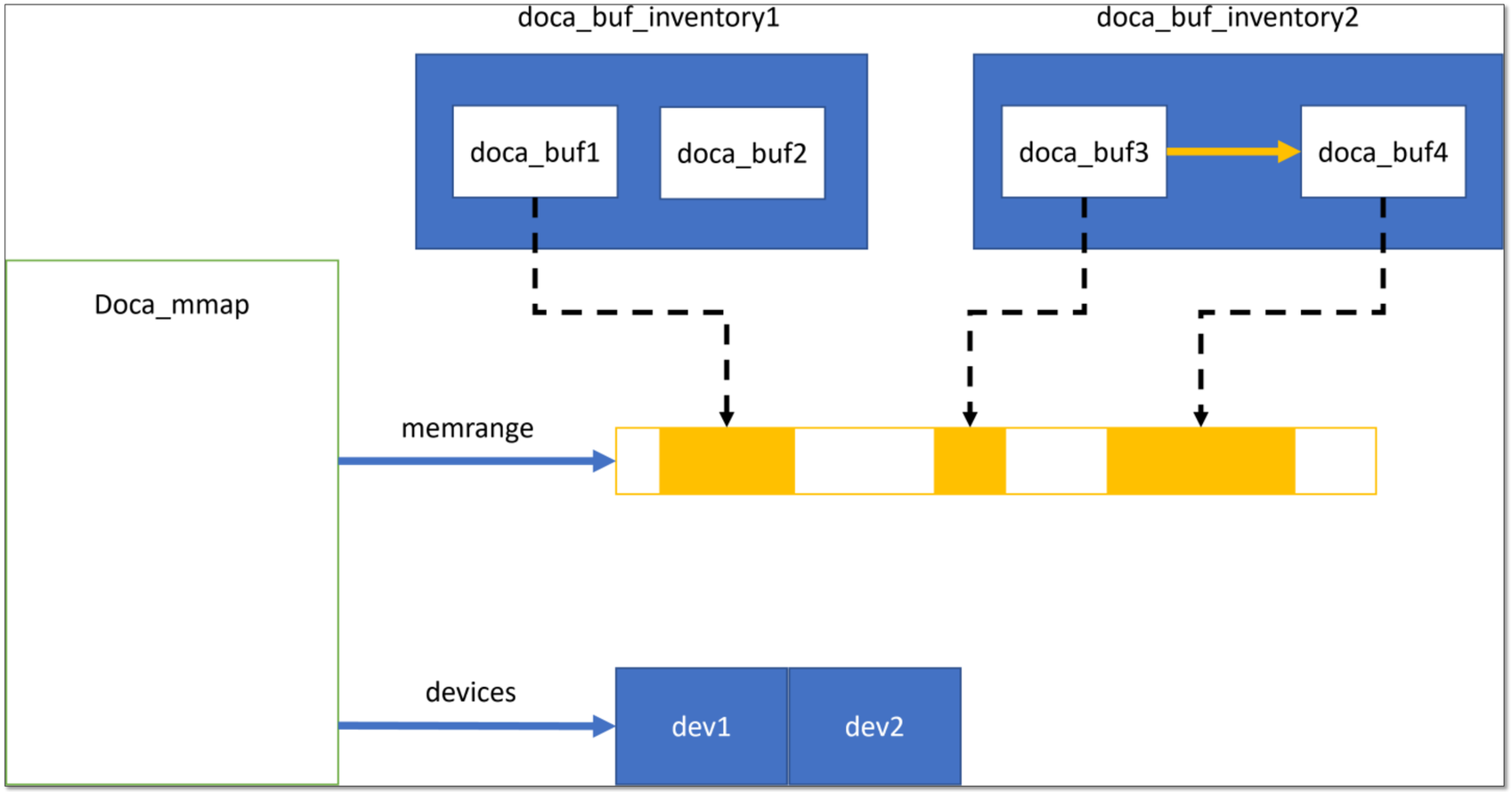


Open Device by capability

```
struct doca_devinfo **dev_list;
uint32_t nb_devs;
int res = doca_devinfo_create_list(&dev_list, &nb_devs);
if (res != DOCA_SUCCESS) {
    DOCA_LOG_ERR("Failed to load doca devices list: %s",
        doca_error_get_descr(res));
    return res;
}
for (int i = 0; i < nb_devs; i++) {
    res = doca_compress_cap_task_decompress_deflate_is_supported(dev_list[i]);
    if (res != DOCA_SUCCESS) {
        continue;
    }
    if(doca_dev_open(dev_list[i], &state->base.device) == DOCA_SUCCESS) {
        doca_devinfo_destroy_list(dev_list);
        break;
    }
}
```

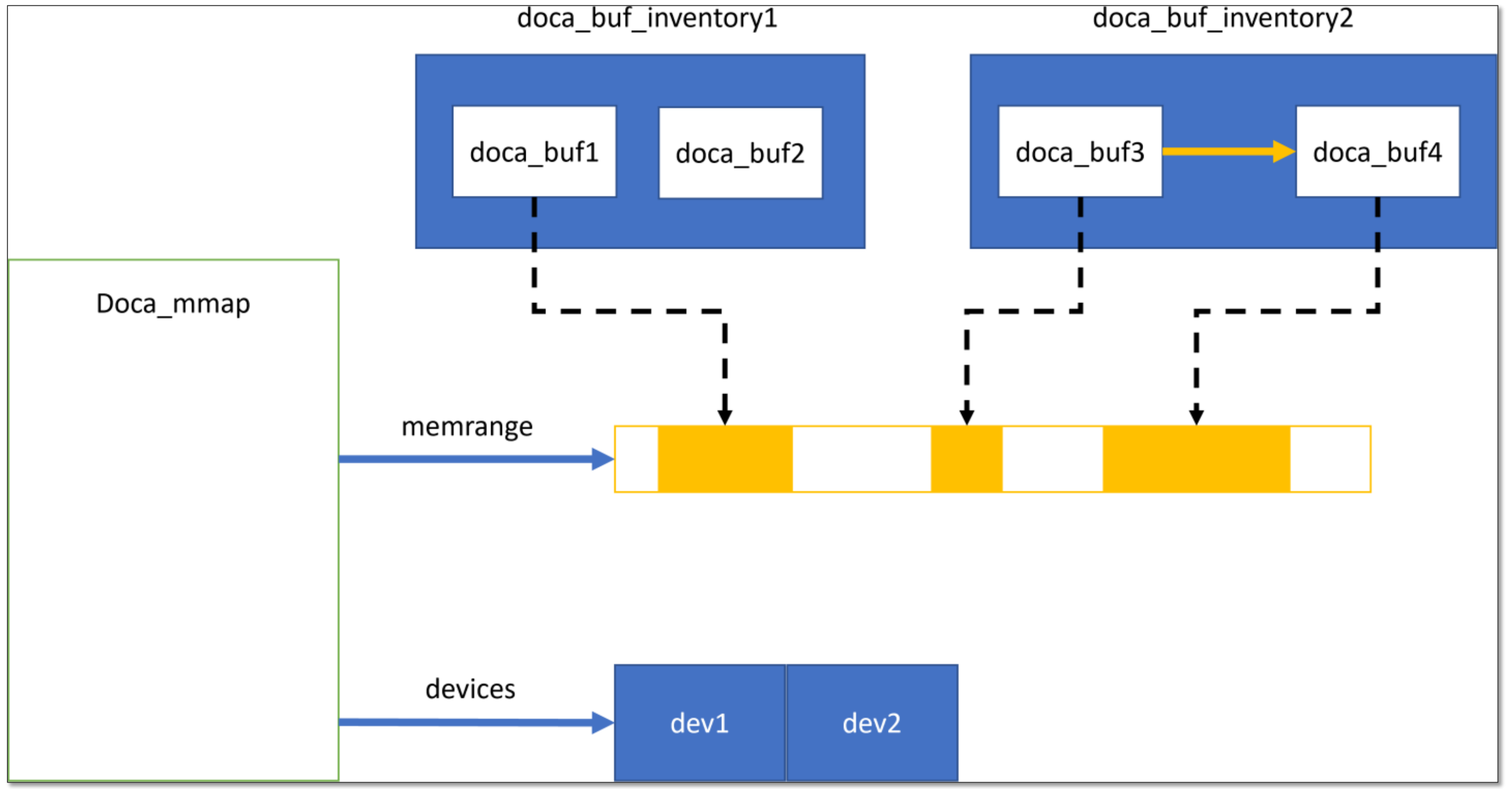
Memory Map (Mmap)

- doca_mmap serves as the memory pool for data buffers
- The application provides a single memory region, as well as permissions for certain devices to access



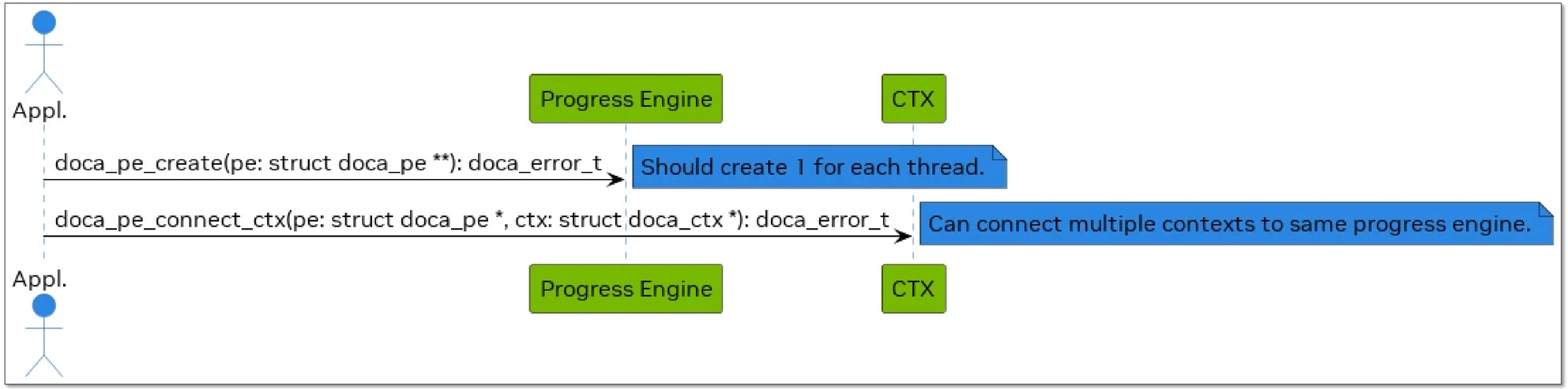
Buffer Inventory

- doca_buf is the data buffer descriptor
- This is not the actual data buffer, rather, it is a descriptor that holds metadata on the "pointed" data buffer
- doca_buf_inventory serves as a pool of doca_buf



Progress Engine (PE)

- The progress engine PE is responsible for scheduling workloads (i.e., picking the next workload to execute).
- PE enables asynchronous processing and handling of multiple tasks and events



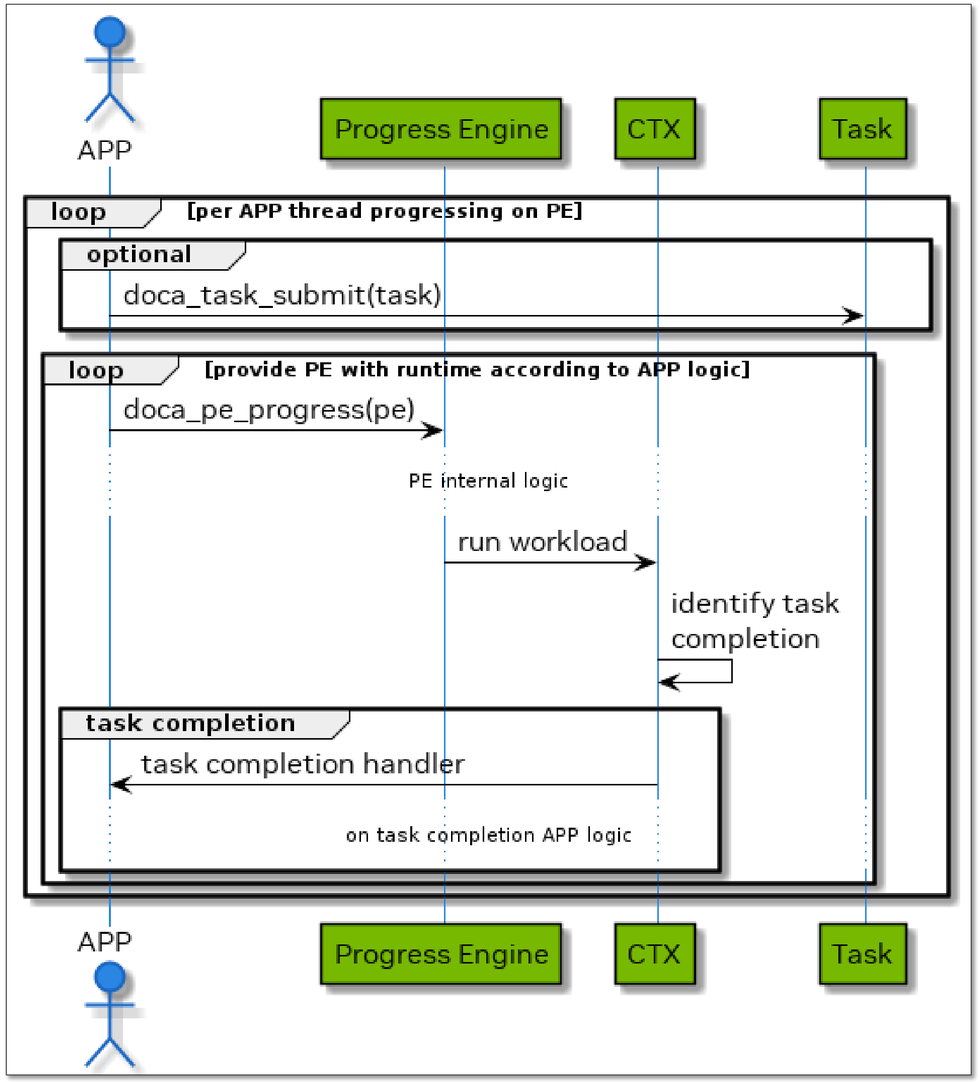
Accelerator Instance

- Create an instance of the accelerator
  - DOCA compress
  - DOCA SHA
  - DOCA RDMA
  - DOCA DMA
  - DOCA Ethernet
  - DOCA Comch
  - DOCA App Shield
  - …

```
DOCA_LOG_INFO("Creating Compress");
EXIT_ON_FAILURE(doca_compress_create(state->base.device, &state->compress));
```

Task Submission and Completion

- Submit task and wait for completion

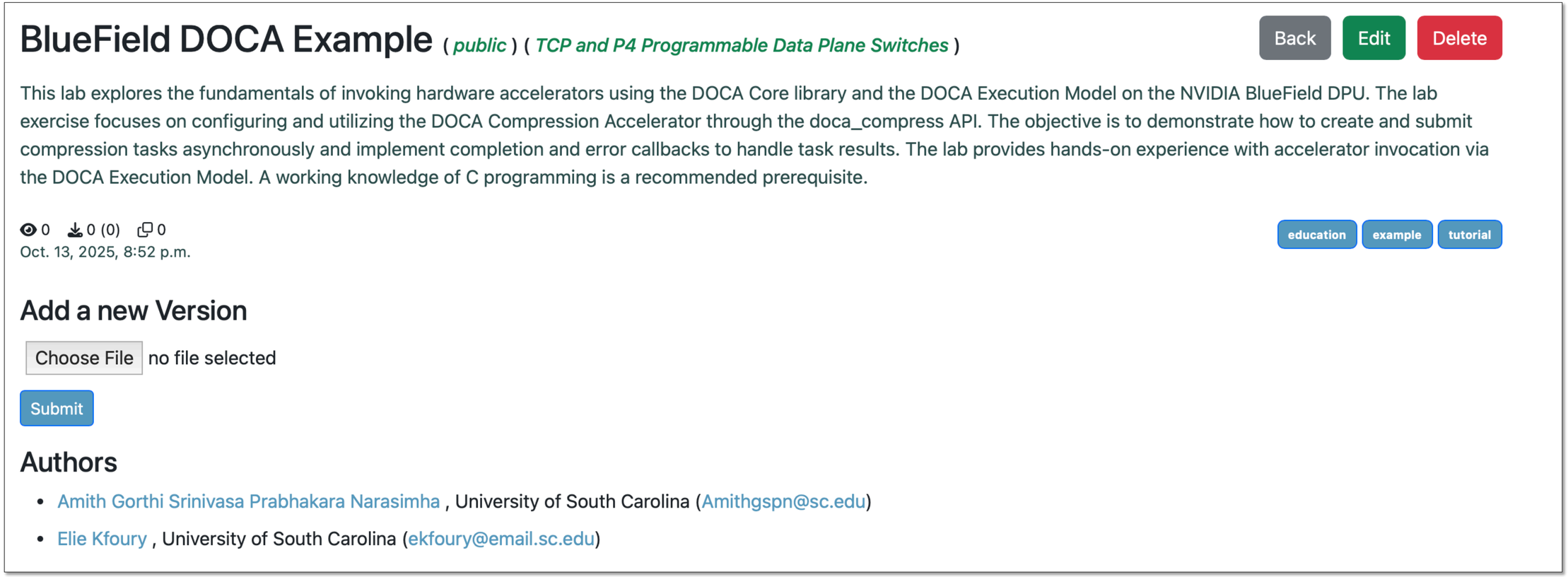


Cleanup

- Gracefully release everything to avoid leaks and dangling mappings
- Make sure all tasks are complete before stopping/destroying resources

DOCA Example on FABRIC Artifact Manager

- Compression acceleration example is available on the Artifact Manager



BlueField DOCA Example (*public*) (*TCP and P4 Programmable Data Plane Switches*)

Back Edit Delete

This lab explores the fundamentals of invoking hardware accelerators using the DOCA Core library and the DOCA Execution Model on the NVIDIA BlueField DPU. The lab exercise focuses on configuring and utilizing the DOCA Compression Accelerator through the doca_compress API. The objective is to demonstrate how to create and submit compression tasks asynchronously and implement completion and error callbacks to handle task results. The lab provides hands-on experience with accelerator invocation via the DOCA Execution Model. A working knowledge of C programming is a recommended prerequisite.

0 0 (0) 0
Oct. 13, 2025, 8:52 p.m.

education example tutorial

Add a new Version

Choose File no file selected

Submit

Authors

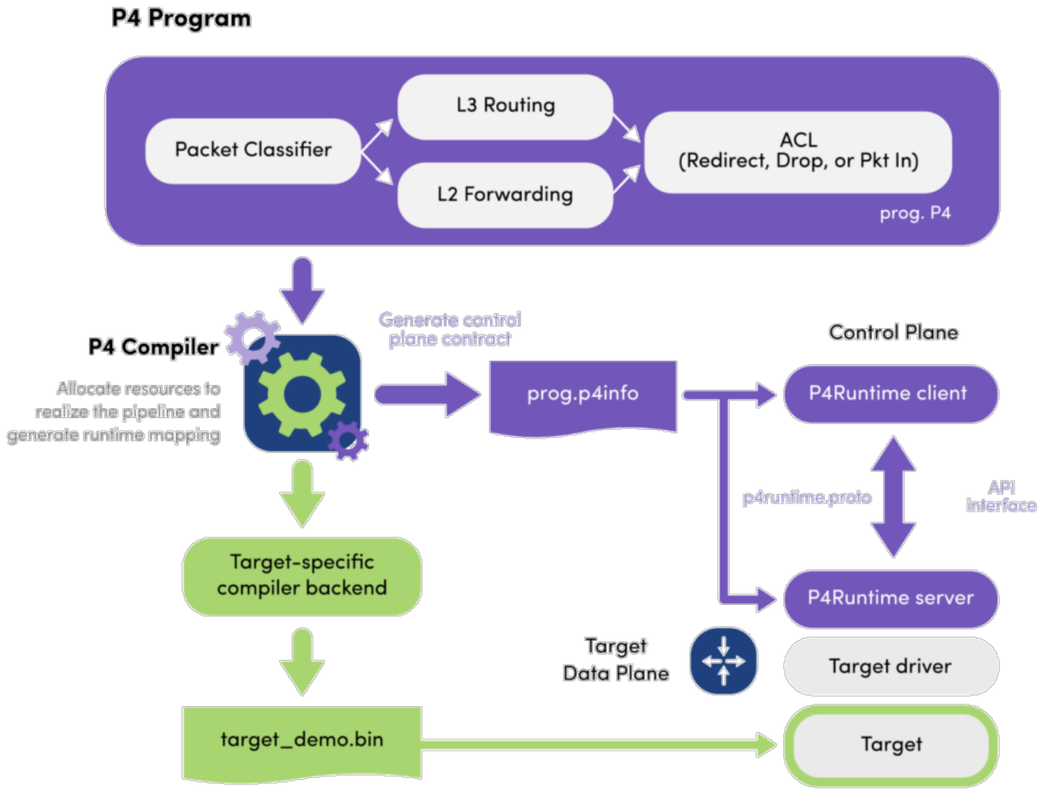
- Amith Gorthi Srinivasa Prabhakara Narasimha , University of South Carolina (Amithgspn@sc.edu)
- Elie Kfoury , University of South Carolina (ekfoury@email.sc.edu)

https://artifacts.fabric-testbed.net/artifacts/4813a7e0-b9a0-4553-98f8-a5a6f08c7a61

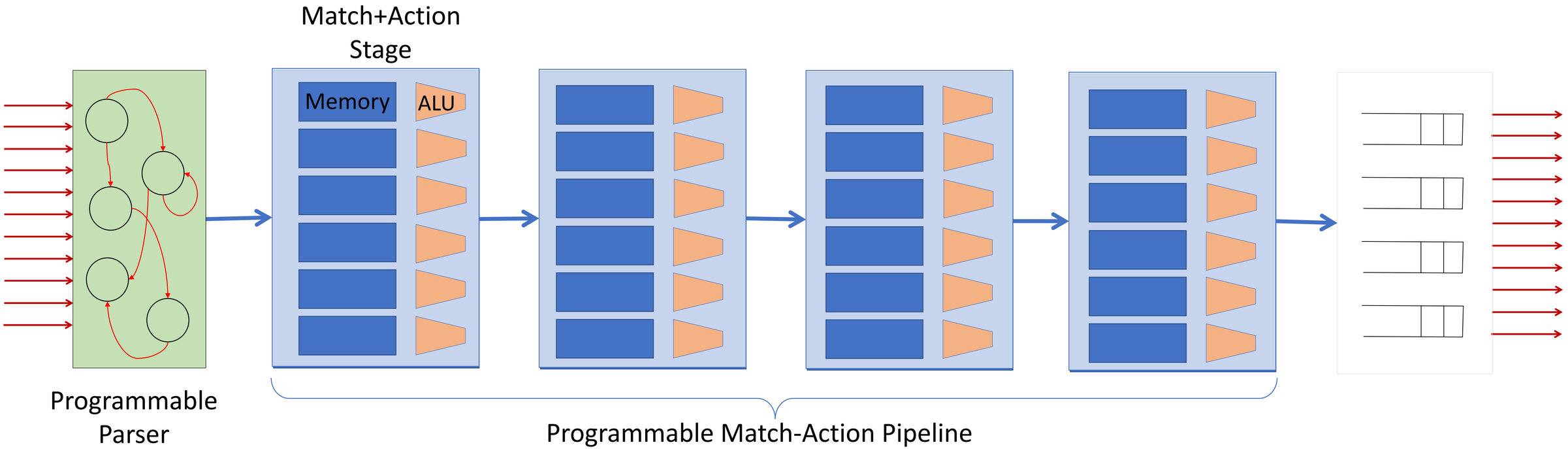
Programming the Accelerated Pipeline with P4

P4 Language

- P4 emerged as domain-specific language to define packet processing:
  - Define and parse new protocols
  - Customize packet processing functions

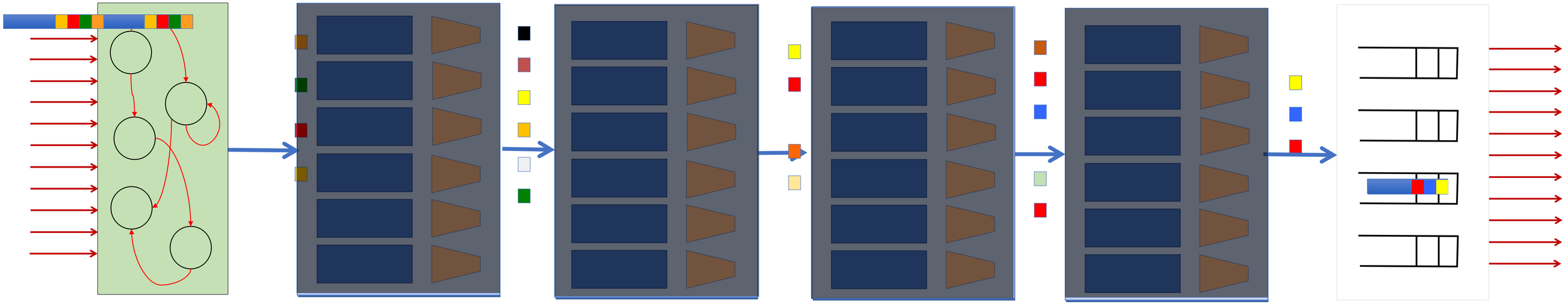


PISA: Protocol Independent Switch Architecture



N. McKeown. Creating an End-to-End Programming Model for Packet Forwarding. Available: https://www.youtube.com/watch?v=fiBuao6YZl0&t=4216s

PISA: Protocol Independent Switch Architecture



N. McKeown. Creating an End-to-End Programming Model for Packet Forwarding. Available: https://www.youtube.com/watch?v=fiBuao6YZl0&t=4216s

Example P4 Program

Parser Program

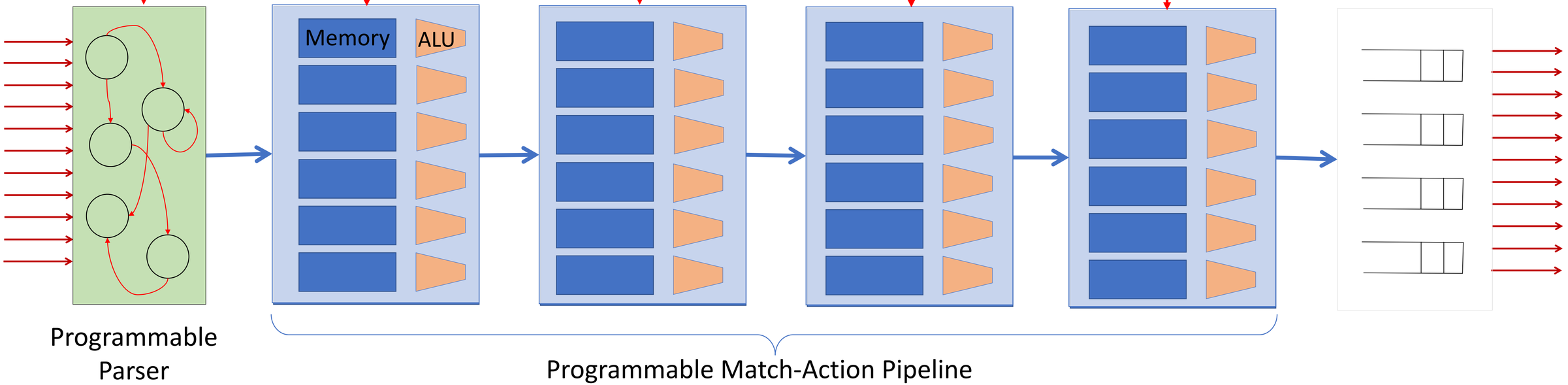
```
parser parse_ethernet {
    extract(ethernet);
    return switch(ethernet.ethertype) {
        0x8100 : parse_vlan_tag;
        0x0800 : parse_ipv4;
        0x8847 : parse_mpls;
        default: ingress;
}
```

Header and Data Declarations

```
header_type  ethernet_t     { … }
header_type  l2_metadata_t  { … }

header     ethernet_t     ethernet;
header     vlan_tag_t
vlan_tag[2];
metadata   l2_metadata_t  l2_meta;
```

Tables and Control Flow

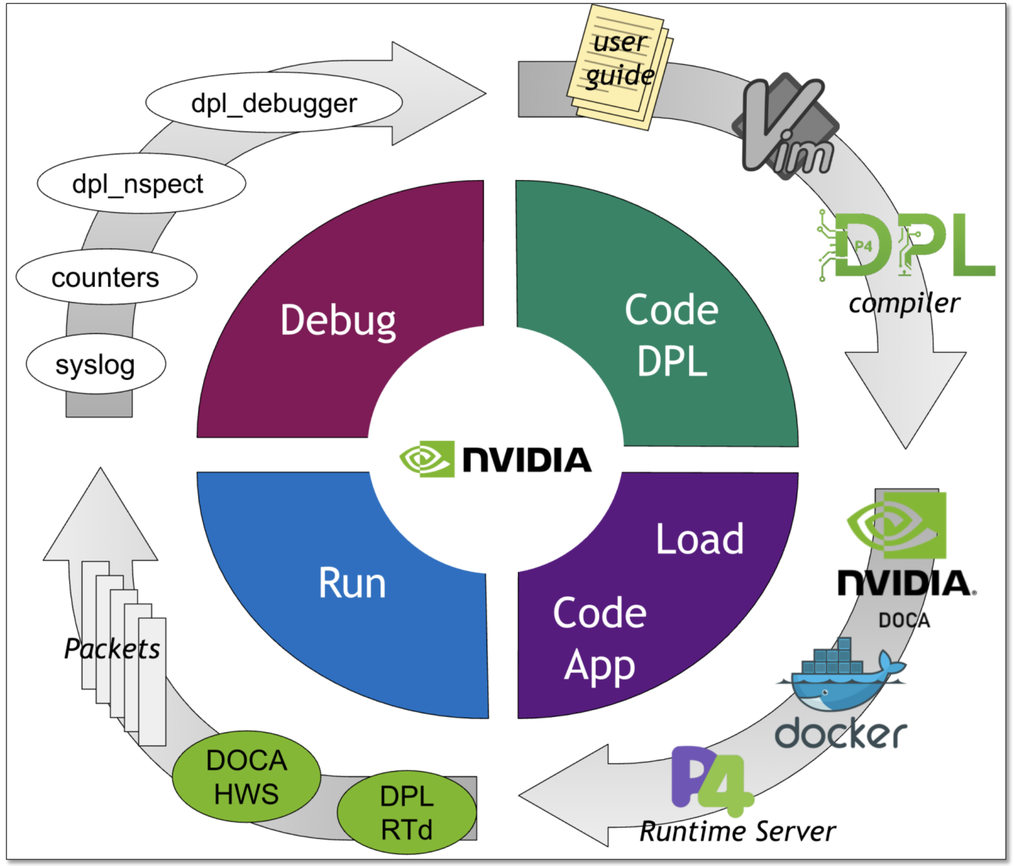
```
table port_table { … }

control ingress {
    apply(port_table);
    if (l2_meta.vlan_tags == 0) {
        process_assign_vlan();
    }
}
```



N. McKeown. Creating an End-to-End Programming Model for Packet Forwarding. Available: https://www.youtube.com/watch?v=fiBuao6YZl0&t=4216s

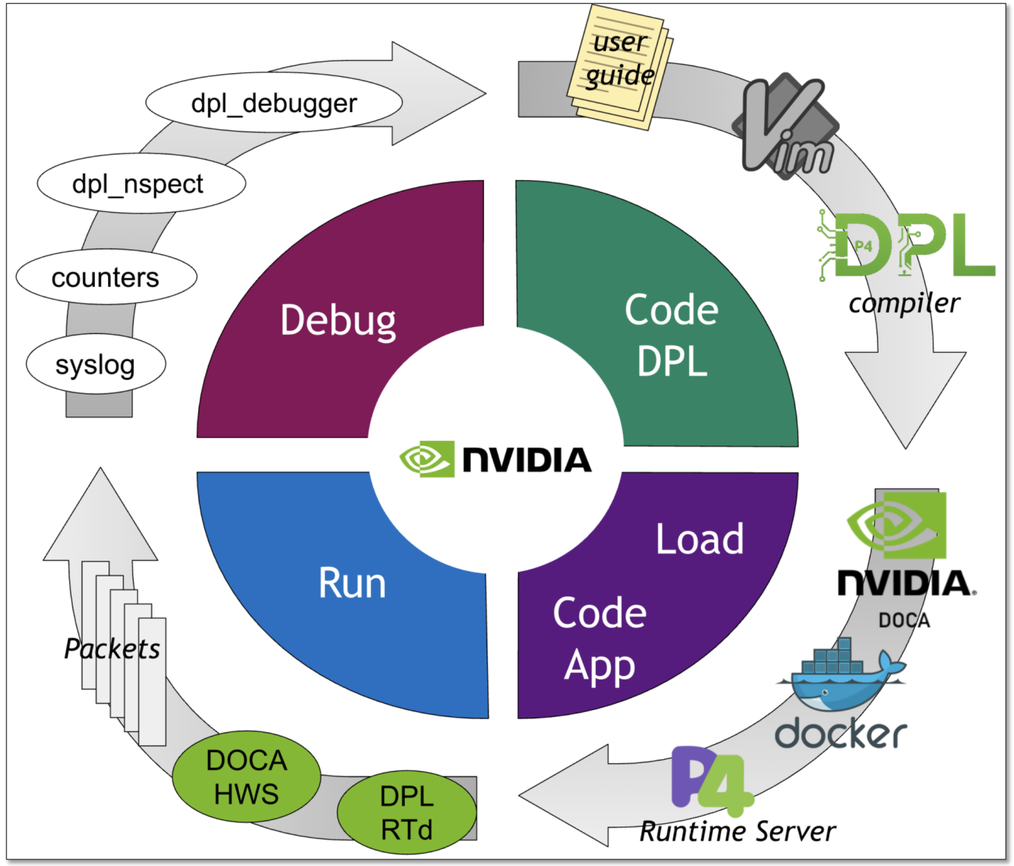
DPL Overview

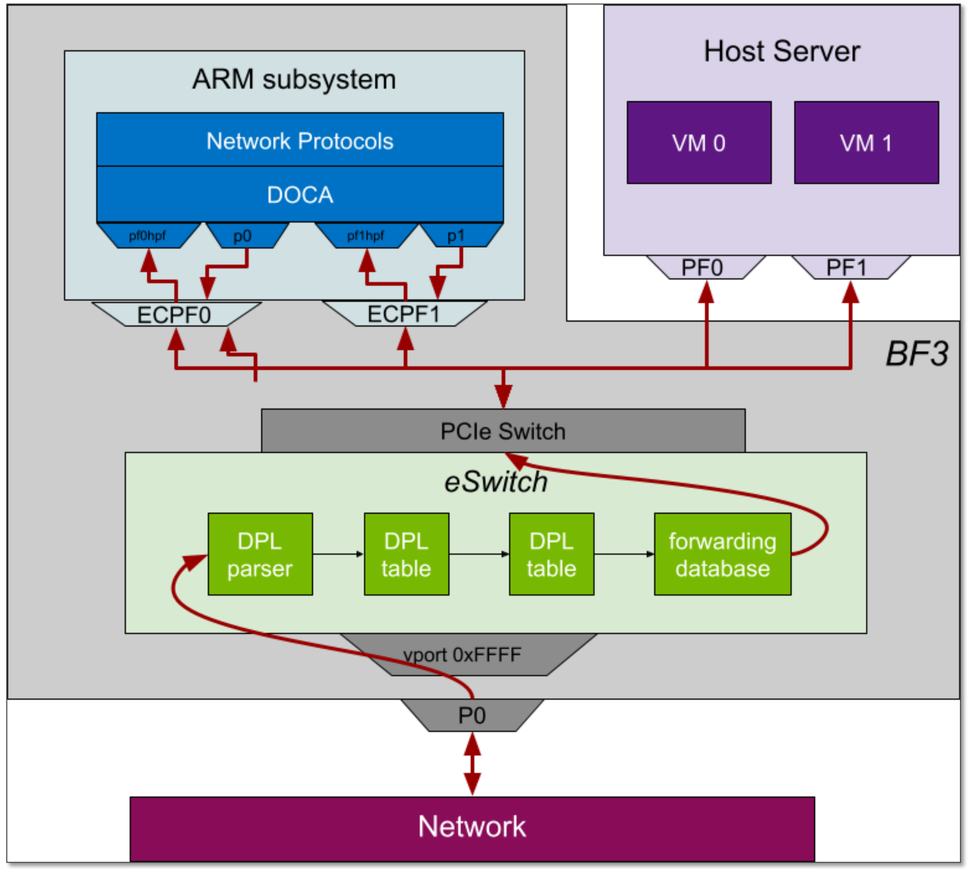
- DOCA Pipeline Language (DPL) introduces a software development solution based on the P4 language
- DPL is derived from the P4-16 language specification



DPL Overview

- DOCA Pipeline Language (DPL) introduces a software development solution based on the P4 language
- DPL is derived from the P4-16 language specification





Hands On Section – Application Experiences

Hands-On Section

Users can test the following codes:

- ODOS-MPI, DOCA, and P4 test cases
- We will also look at brief demo of DPA programming

Please check the instructions for accessing the testbed from our public GitHub repo at:

https://github.com/gt-crnch-rg/smartnic-tutorial-sc25/

Wrap-Up

Summary

In this tutorial we have covered the following:

- Described how SmartNICs can be used as HPC/AI accelerators
- Described how users can benefit from acceleration engines in their current and future applications
- Described how one can use one family of SmartNICs to accelerate applications and communication libraries
- Shared demo experiences using SmartNICs as well as hands-on samples.

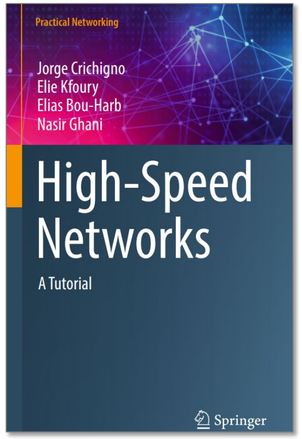
SmartNICs and DPUs for HPC/AI is a wide open area for future research and engineering!

Cybertraining Material (USC)

Information about lab libraries is available at: https://research.cec.sc.edu/cyberinfra/cybertraining

Programmable data plane devices

1. DPU Programming using P4
2. DPU Programming using DOCA
3. P4-DPDK Security
4. Introduction to P4-DPDK
5. Cybersecurity Apps with P4 Programmable Data Planes
6. P4 Programmable Data Planes: Applications, Stateful Elements, Custom Packet Processing
7. Intro to P4 Programmable Switches
8. Intro to P4 Programmable Switches with Intel's Tofino
9. P4 Monitoring Applications



Traditional Protocols

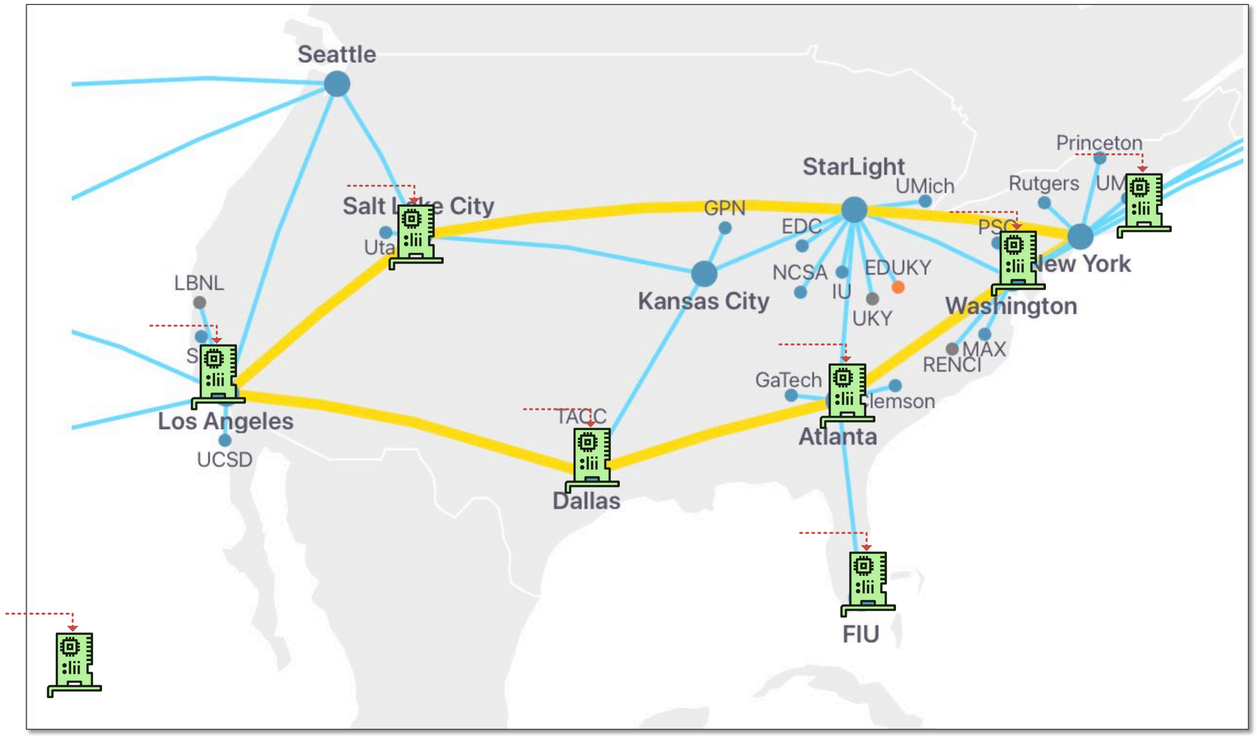
10. Software-defined Networking (SDN)
11. Open vSwitch
12. Network Tools and Protocols
13. Introduction to IPv6
14. Open Shortest Path First (OSPF)
15. Introduction to BGP
16. MPLS and Advanced BGP

Cybersecurity, Management, and Science DMZs

17. Zeek IDS/IPS
18. Cybersecurity Fundamentals
19. perfSONAR 5
20. P4-perfSONAR
21. Network Management Tools
22. High-speed Networks: A Tutorial

DPUs on FABRIC

- BlueField-3 DPUs are being installed on various FABRIC sites
- Currently the following sites have DPUs installed (more will be available soon):
  - MASS, DALL, SALT, FIU, LOSA, NEWY, ATLA, HAWI
- These sites are available to the users (and projects that have the permissions)



https://portal.fabric-testbed.net/community/testbeds-and-facilities

SmartNIC and DPU Testbed Resources

You can access SmartNICs via a variety of general access testbeds:

USC-led workshops and training resources

- https://research.cec.sc.edu/cyberinfra/workshops

FABRIC

- 100 GE connections and testbeds, BlueField-3 DPUs
- https://portal.fabric-testbed.net/

CRNCH Rogues Gallery testbed

- BlueField 2-3, Alveo FPGAs, MangoBoost SmartNICs
- https://crnch-rg.cc.gatech.edu/

ORNL Wombat cluster

- BlueField 2-3
- https://olcf.ornl.gov/olcf-resources/compute-systems/wombat

Xilinx HACC centers

- AMD Alveos for OpenNIC work
- https://www.xilinx.com/support/university/xup-hacc.html

Acknowledgements

- This work was supported by the U.S. National Science Foundation (NSF), under awards 2417823 and 2346726. Work at Georgia Tech was supported by NSF award 2316177.
- Elie Kfoury and Jorge Crichigno would also like to acknowledge the FABRIC team.
- Antonio J. Peña was partially supported by the Ramón y Cajal fellowship RYC2020-030054-I funded by MCIN/AEI/10.13039/501100011033 and, by “ESF Investing in your future”

Questions and Attendee Feedback

Please fill out the Mentimeter Survey and any SC25 surveys!