Multiple GPU programming with MPI

NRIS

Norwegian research infrastructure services

Hicham Agueny, PhD.
Scientific Computing Group
University of Bergen/NRIS

Main goal:

To initiate your interest in combining GPU programming models

with a MPI library

Motivation

Synchroneous OpenACC/OpenMP offloading: **limited to single GPU**.

Asynchroneous OpenACC/OpenMP offloading: **multiple GPUs BUT limited to a single node**.

Accelerating existing MPI-based codes.

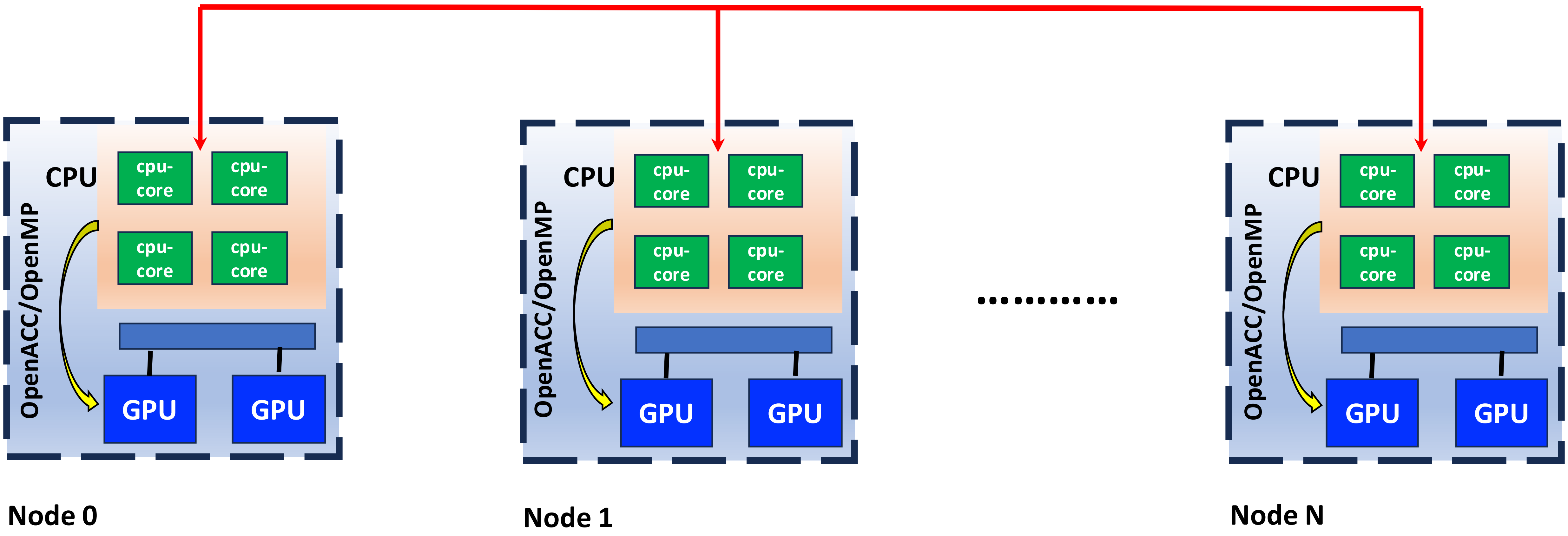
Fully utilise the capacity of exascale supercomputer such as LUMI.

There is a need of multiple GPU programming.

Multiple GPU with MPI communications

Distributed computing

MPI communication library



CPU

cpu-core

cpu-core

cpu-core

cpu-core

OpenACC/OpenMP

GPU

GPU

Node 0

Node 1

Node N

Outline

❑ **Concept of:**

- Traditional MPI without GPU awareness (with involvement of the CPU memory)
- MPI with GPU awareness (without involvement of the CPU memory)

Outline

- ❑**Concept of:**
    - Traditional MPI without GPU awareness (with involvement of the CPU memory)
    - MPI with GPU awareness (without involvement of the CPU memory)
- ❑**Application: MPI-OpenACC & MPI-OpenMP offloading**
    - Point-to-point communication (MPI_Send & MPI_Recv)
- ❑**Conclusion & Benchmark**
- ❑**Hands-on examples and exercises**

Learning Outcomes

❑ **How to assign each MPI rank to a GPU device ?**

❑ How to combine MPI and OpenACC/OpenMP

- **MPI operations** : **MPI_Send and MPI_Recv**
- **OpenACC directives**: **update host(); update device()**
- **OpenMP directives**: **update device() from(); update device() to()**

❑ How to perform MPI operations with GPU-awareness support

- **MPI operations** : **MPI_Send and MPI_Recv**
- **OpenACC directive**: **host_data device_ptr()**
- **OpenMP directive**: **use_device_ptr()**

See our documentation for further details

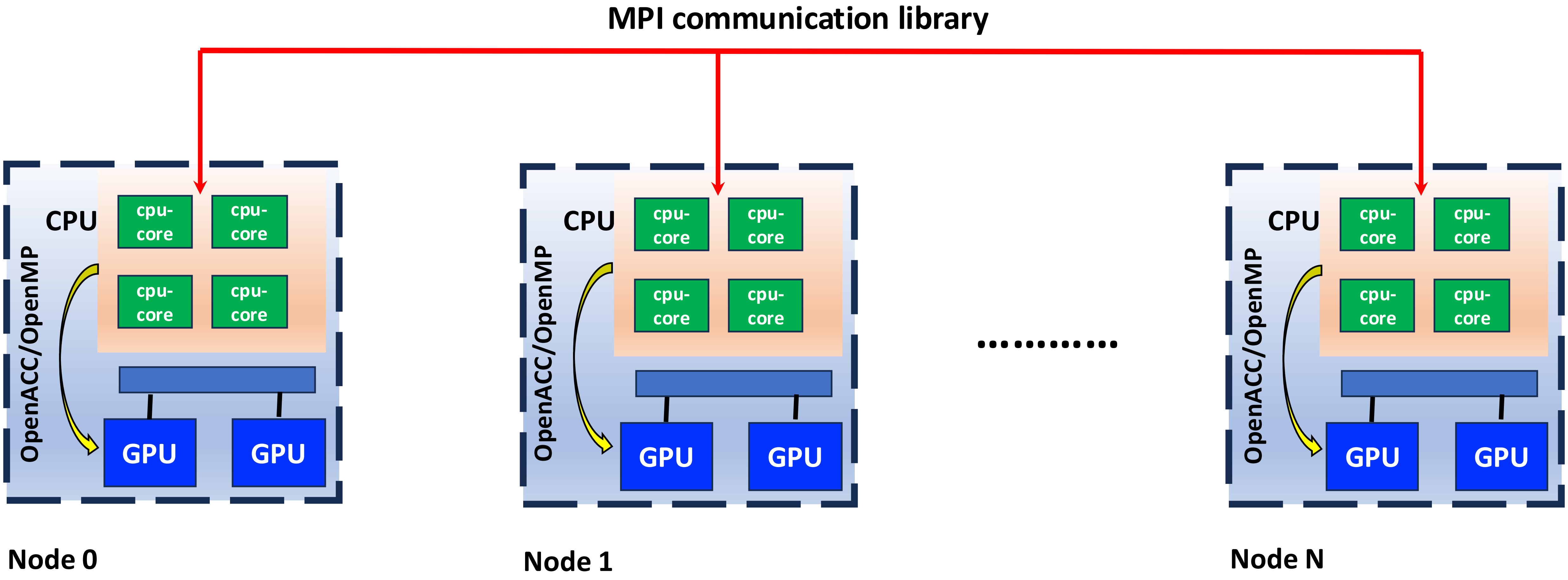
https://enccs.github.io/gpu-programming/8-multiple_gpu/
https://documentation.sigma2.no/code_development/guides/gpuaware_mpi.html

Concept of Multiple GPU with MPI

What is MPI (Message Passing Interface)?

MPI allows different processes to communicate with each other via messages (point-to-point & collective communications) .

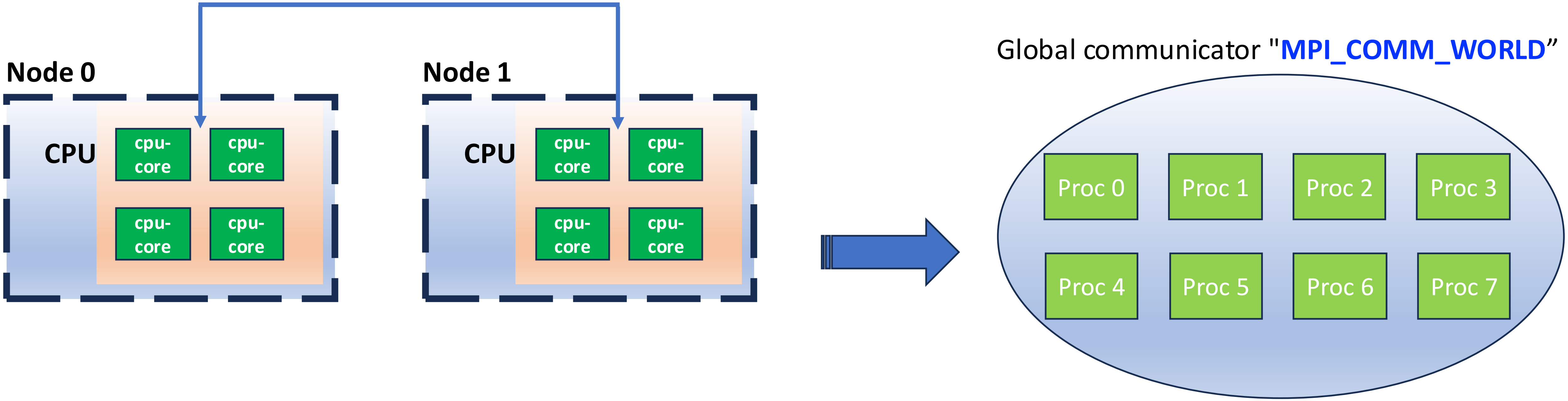
MPI is designed to build HPC applications that can scale across multiple computer-nodes in distributed systems.



How to assign each MPI rank to a GPU device ?

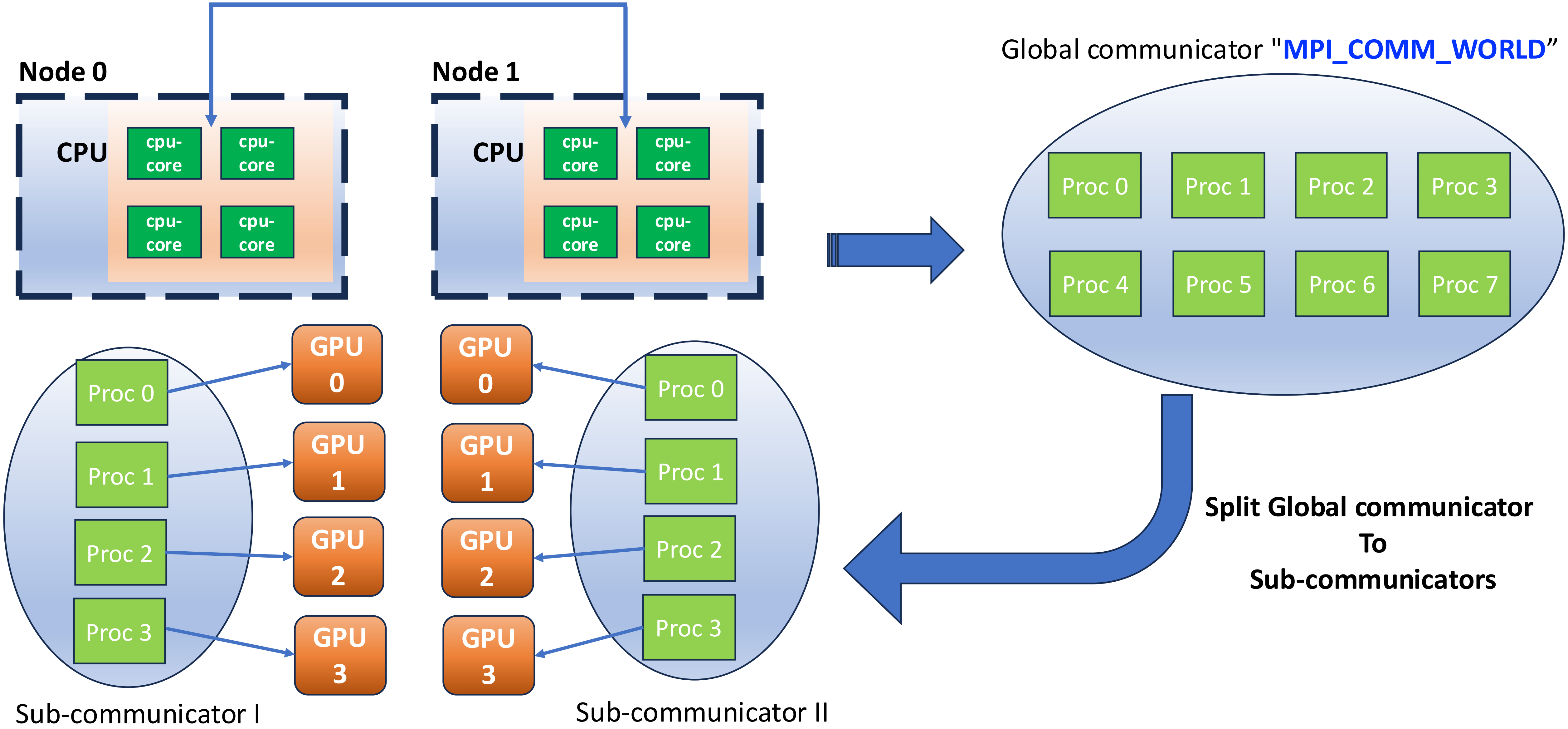
How to assign each MPI rank to a GPU device ?

Split the global communicator "MPI_COMM_WORLD” into sub-communicators



How to assign each MPI rank to a GPU device ?

Split the global communicator "MPI_COMM_WORLD” into sub-communicators



How to assign each MPI rank to a GPU device ?

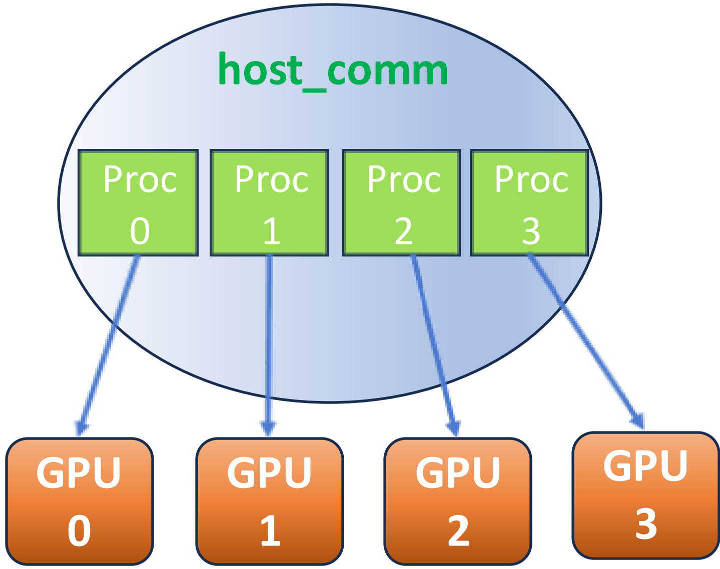
MPI functions

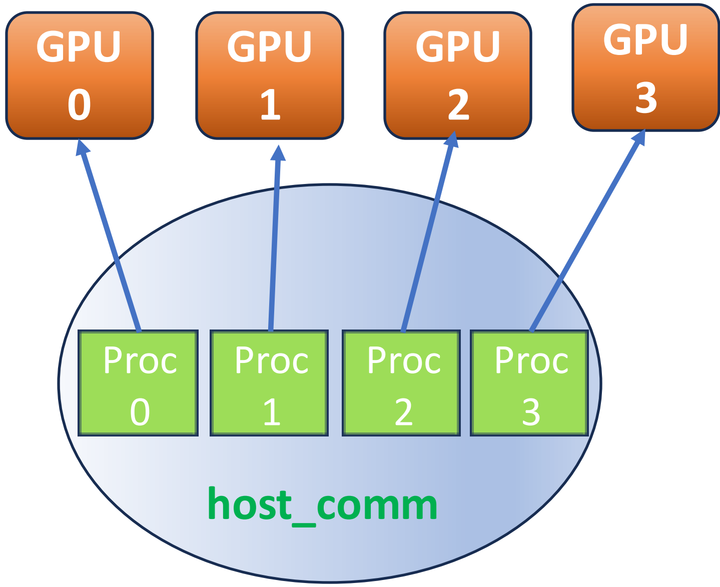
```
call MPI_COMM_SPLIT_TYPE(MPI_COMM_WORLD,
                         MPI_COMM_TYPE_SHARED, 0,
                         MPI_INFO_NULL, host_comm, ierr)

call MPI_COMM_RANK(host_comm, host_rank, ierr)
```

Sub-communicator

New MPI rank

Sub-communicator





Sub-communicator

How to assign each MPI rank to a GPU device ?

MPI functions

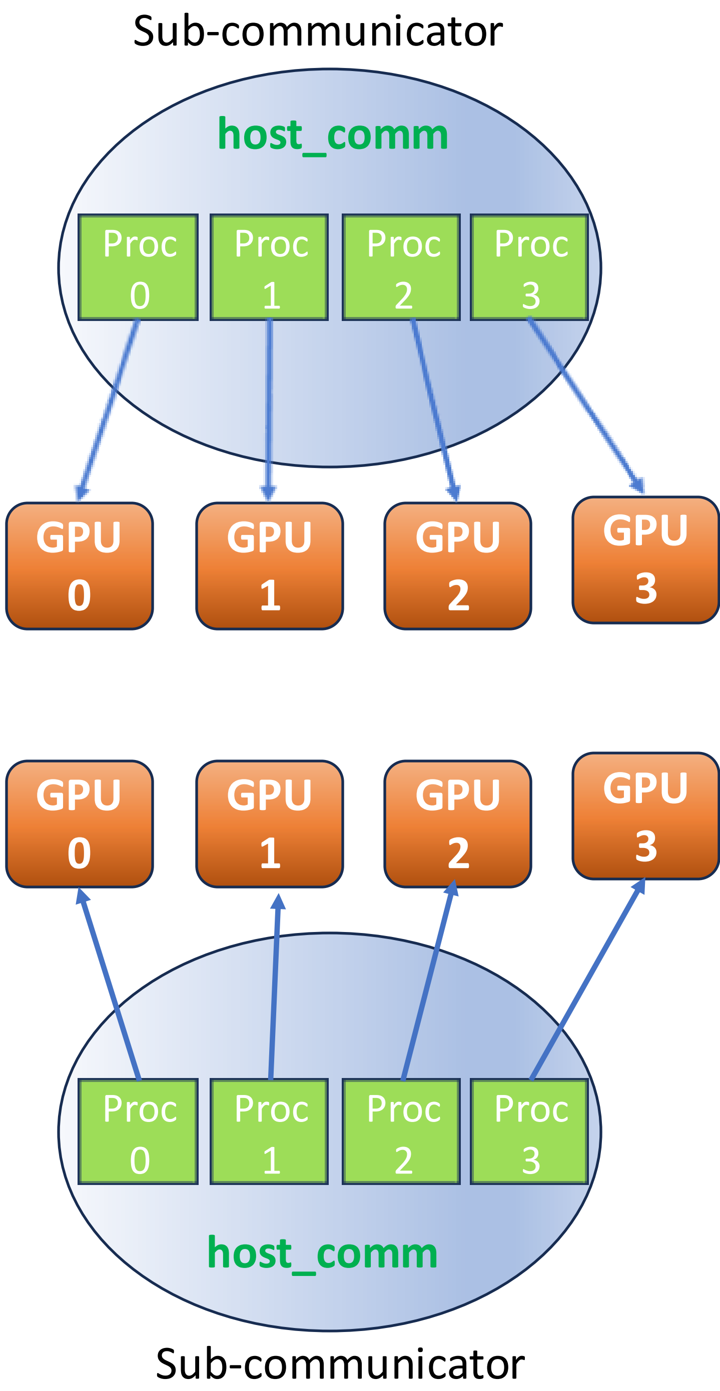
```
call MPI_COMM_SPLIT_TYPE(MPI_COMM_WORLD,
                         MPI_COMM_TYPE_SHARED, 0,
                         MPI_INFO_NULL, host_comm, ierr)

call MPI_COMM_RANK(host_comm, host_rank, ierr)
```

Sub-communicator

New MPI rank

API functions: OpenACC

```
! Sets the device INDEX and the device type to be used
call acc_set_device_num(host_rank, acc_get_device_type())

! Returns the number of devices available on the host (here on each node)
 numDevice = acc_get_num_devices(acc_get_device_type())
```



How to assign each MPI rank to a GPU device ?

MPI functions

```
call MPI_COMM_SPLIT_TYPE(MPI_COMM_WORLD,
                         MPI_COMM_TYPE_SHARED, 0,
                         MPI_INFO_NULL, host_comm, ierr)

call MPI_COMM_RANK(host_comm, host_rank, ierr)
```

API functions: OpenACC

```
! Sets the device INDEX and the device type to be used
call acc_set_device_num(host_rank, acc_get_device_type())

! Returns the number of devices available on the host (here on each node)
 numDevice = acc_get_num_devices(acc_get_device_type())
```

API functions: OpenMP

```
! Sets the device INDEX and the device type to be used
call omp_set_default_device(host_rank)

! Returns the number of devices available on the host (here on each node)
 numDevice = omp_get_num_devices()
```

Hands-on Example 1:

How to assign each MPI rank to a GPU device ?

- **Download the repo**

```
$ git clone https://github.com/HichamAgueny/multiGPU_MPI_examples
$ cd multiGPU_MPI_examples
```

- **For MPI-OpenACC**

```
$ cd example_1/setDevice_acc
```

- **For MPI-OpenMP**

```
$ cd example_1/setDevice_omp
```

- **To compile and execute the code**

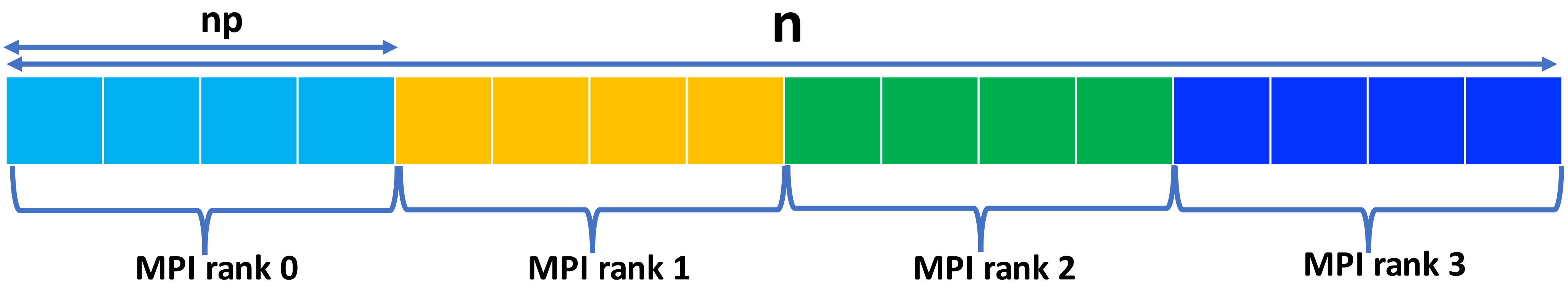
Load the LUMI software stack

```
$ module load LUMI/24.03 partition/G
$ module load cpeCray
```

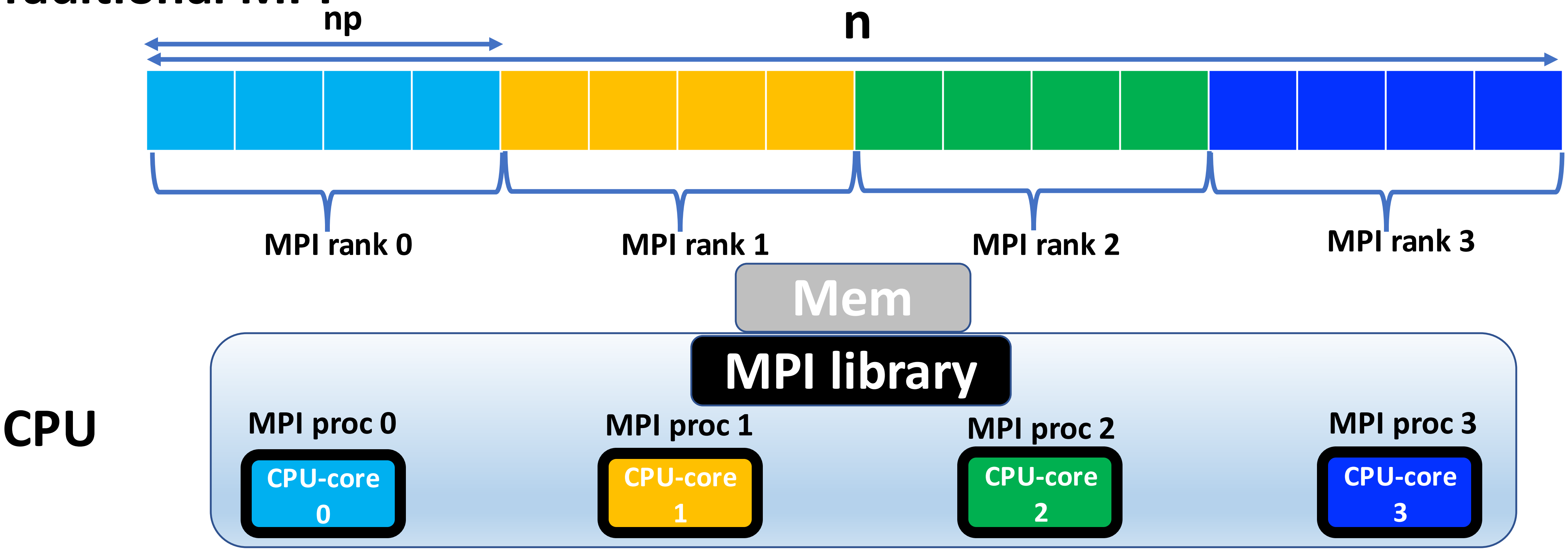
Compile: `$ ./compile.sh`
Submit a job: `$ sbatch script.slurm`
View the output file: `$ vi setDevice_accxxxxxx.out`

Concept:
Traditional MPI without GPU awareness

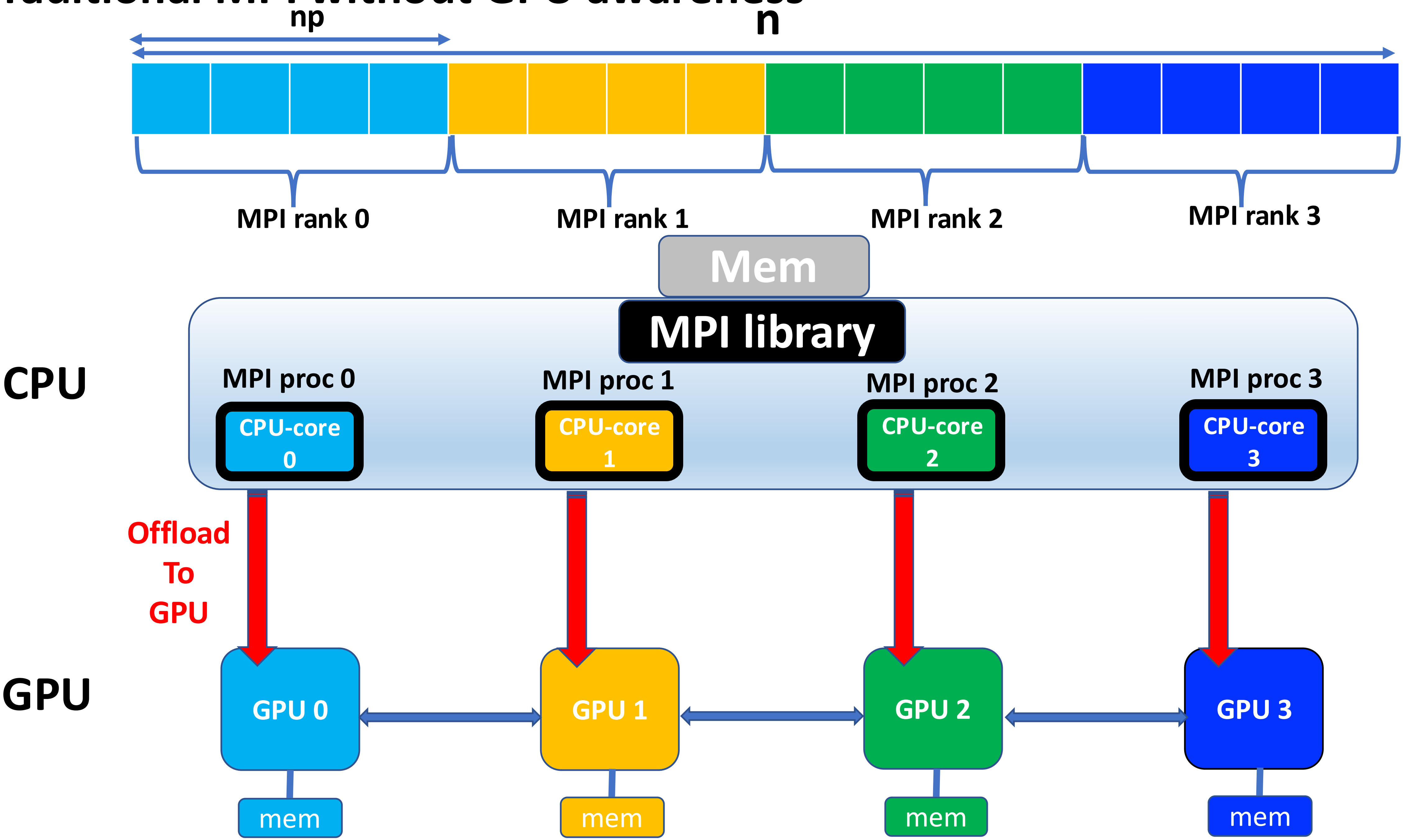
Traditional MPI



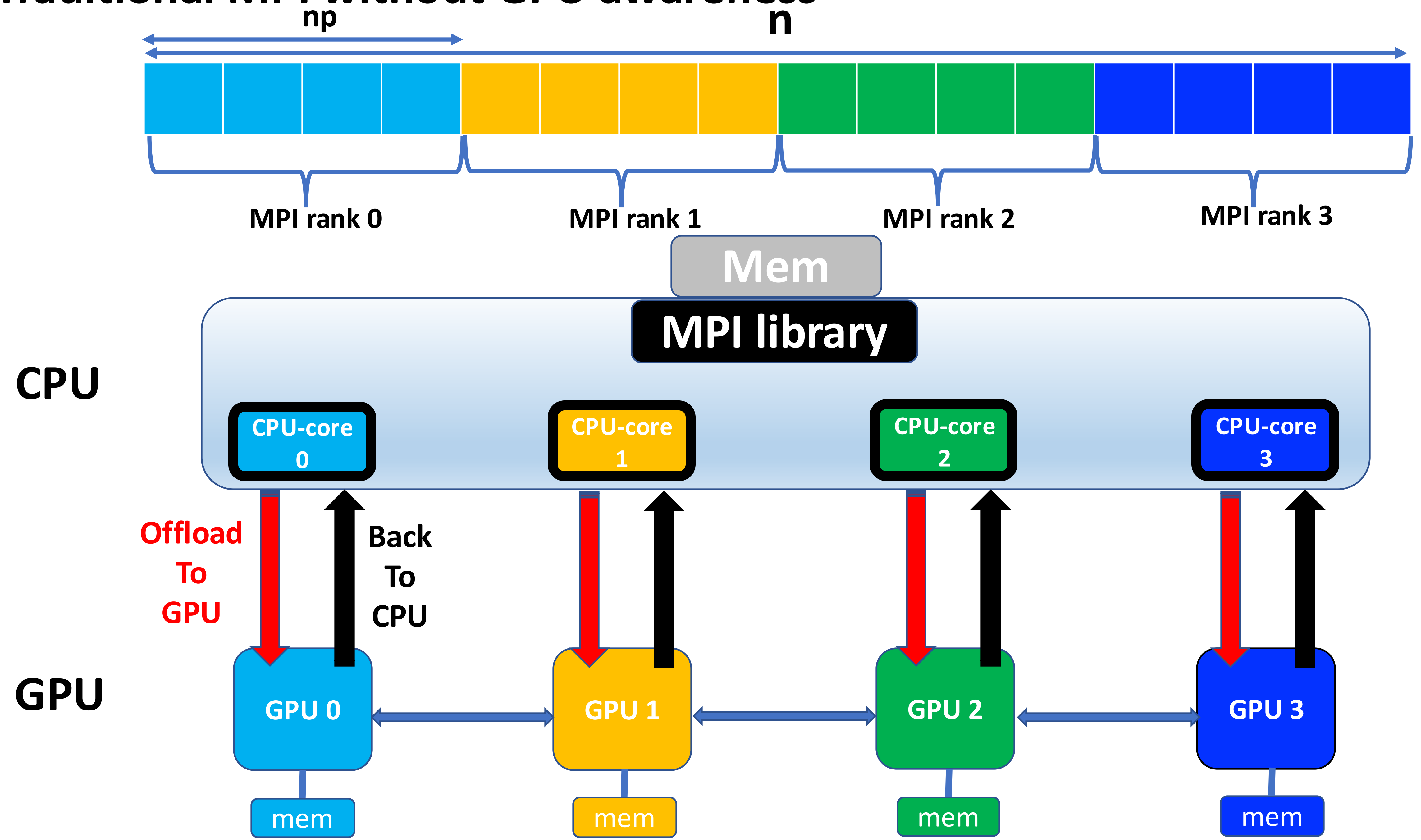
Traditional MPI



Traditional MPI without GPU awareness



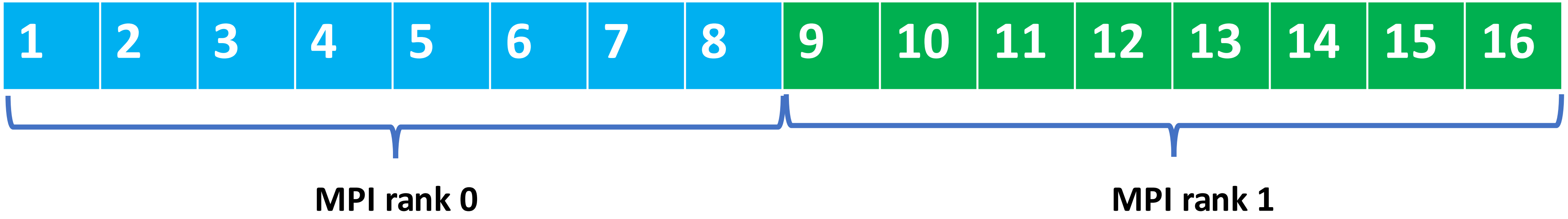
Traditional MPI without GPU awareness



Example: Combining MPI with OpenACC/OpenMP APIs

- **MPI operations** : **MPI_Send & MPI_Recv**
- **OpenACC directives**: **update host(); update device()**
- **OpenMP directives**: **update device() from(); update device() to()**

Scenario with only 2 MPI-processes: MPI_Send & MPI_Recv



```
if(MPIrank.eq.0) then
   call MPI_Send(f_send(:), N,MPI_DOUBLE_PRECISION, MPIrank=1, tag1, ..)
   call MPI_Recv(f_recv(:), N, MPI_DOUBLE_PRECISION, MPIrank=1, tag2, ..)
endif
if(MPIrank.eq.1) then
   call MPI_Recv(f_recv(:), N, MPI_DOUBLE_PRECISION, MPIrank=0, tag1, ..)
   call MPI_Send(f_send(:), N, MPI_DOUBLE_PRECISION, MPIrank=0, tag2, ..)
endif
```

MPI-OpenACC: MPI_Send & MPI_Recv

Data region

!$acc enter data copyin(f_send,f_recv)

Perform computation on GPU

!$acc update host(f_send,f_recv)

Do MPI operations on CPU

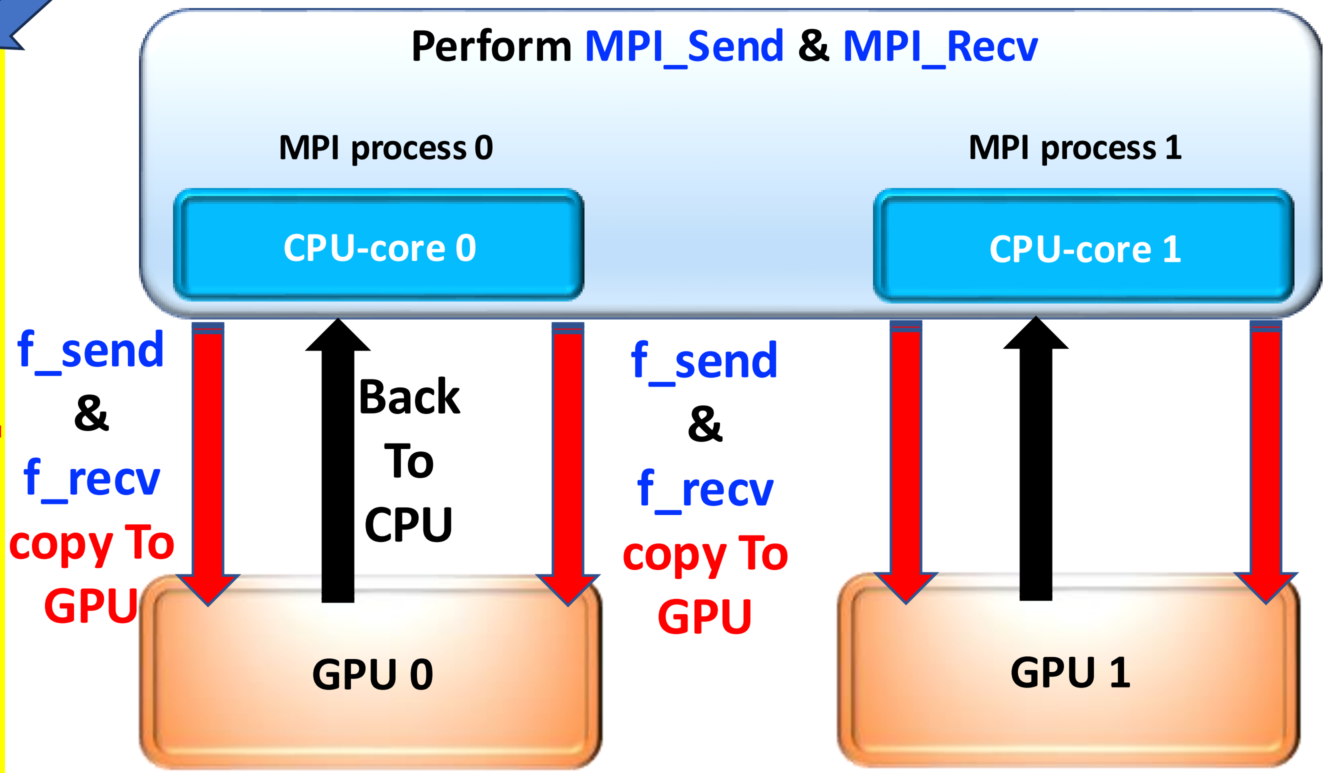
call MPI_Send(f_send, N , ……)

call MPI_Recv(f_recv, N, ……..)

!$acc update device(f_send,f_recv)

Perform computation on GPU

!$acc exit data copyout(f_recv)



Perform MPI_Send & MPI_Recv
MPI process 0
MPI process 1
CPU-core 0
CPU-core 1
f_send & f_recv copy To GPU
Back To CPU
f_send & f_recv copy To GPU
GPU 0
GPU 1

MPI-OpenACC

```
!$acc enter data copyin(f_send,f_recv)
```

Perform computation on GPU

```
!$acc update host(f_send,f_recv)
```

Do MPI operations on CPU

```
call MPI_Send(f_send, N , ……)
call MPI_Recv(f_recv, N , ……..)
```

```
!$acc update device(f_send,f_recv)
```

Perform computation on GPU

```
!$acc exit data copyout(f_recv)
```

MPI-OpenMP

```
!$omp target enter data map(to: f_send,f_recv)
```

Perform computation on GPU

```
!$omp target update from(f_send,f_recv)
```

Do MPI operations on CPU

```
call MPI_Send(f_send, N , ……)
call MPI_Recv(f_recv, N , ……..)
```

```
!$omp target update to(f_send,f_recv)
```

Perform computation on GPU

```
!$omp target exit data map(from: f_recv)
```

Hands-on Example 2: Traditional MPI with OpenACC/OpenMP ?

Purpose: To measure **the time it takes to transfer data** during **MPI communication**

- **Download the repo**

$ git clone https://github.com/HichamAgueny/multiGPU_MPI_examples
$ cd multiGPU_MPI_examples

- **For MPI-OpenACC**

$ cd example_2/mpiacc

- **For MPI-OpenMP**

$ cd example_2/mpiomp

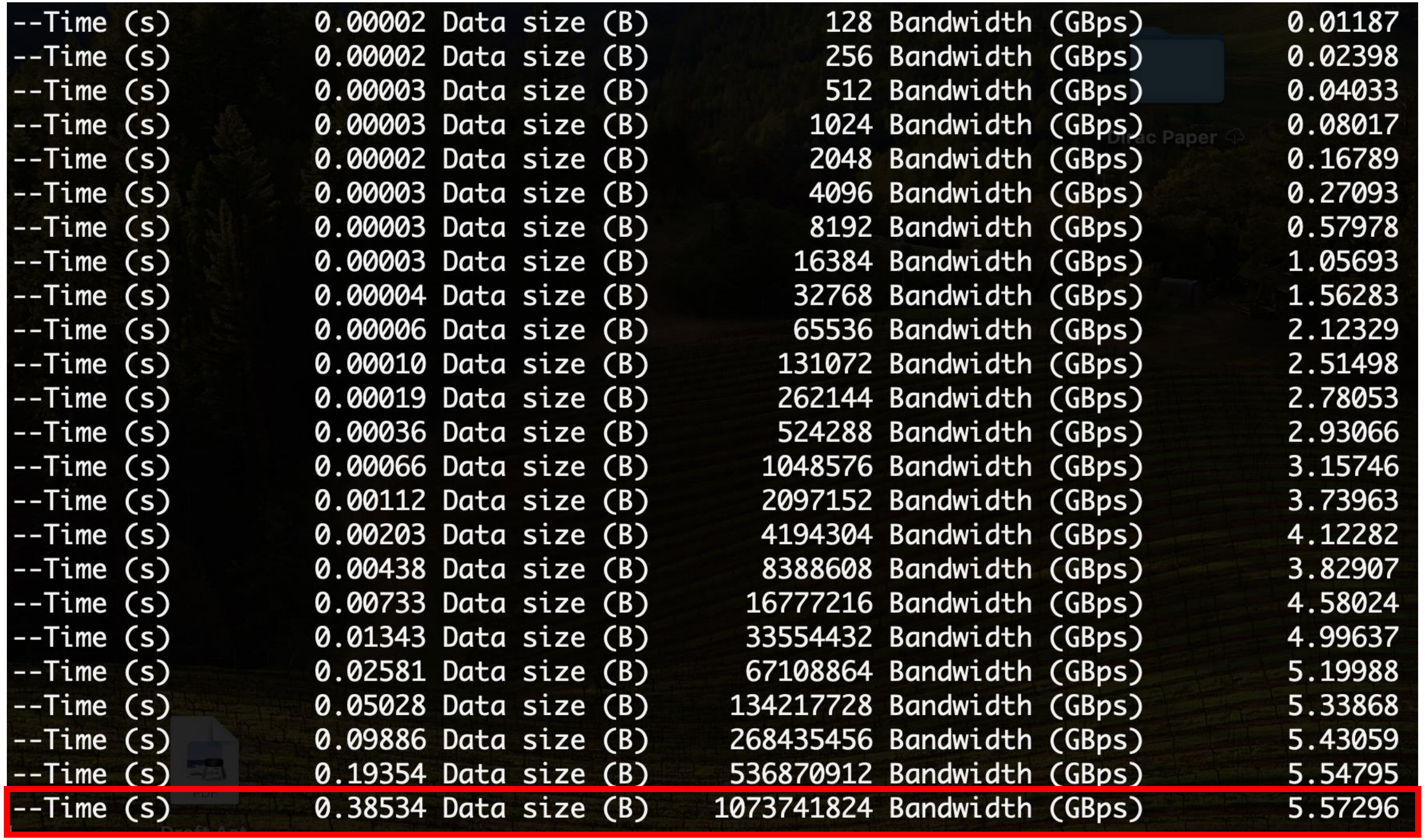
- **To compile and execute the code**

Load the LUMI software stack
$ module load LUMI/24.03 partition/G

Compile: $ ./compile.sh
Submit a job: $ sbatch script.slurm
View the output file: $ vi staging_mpiacc-xxxxxx.out

Output from running the code **example_2/mpiacc/mpiacc_staging.f90**

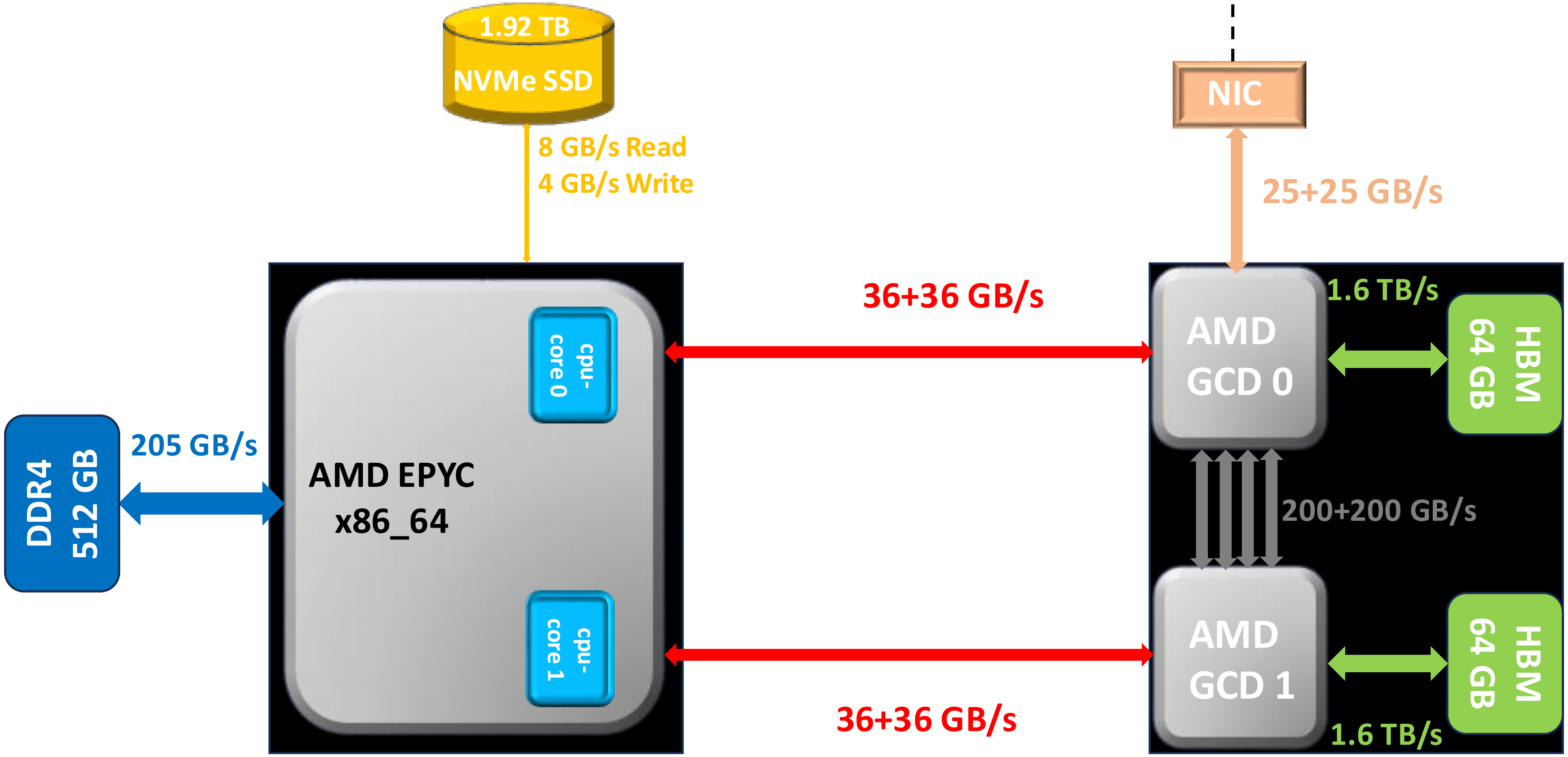
Measruing bandwidth: 2 MPI processes and 2 GPUs, single Node



```
--Time (s)          0.00002 Data size (B)           128 Bandwidth (GBps)         0.01187
--Time (s)          0.00002 Data size (B)           256 Bandwidth (GBps)         0.02398
--Time (s)          0.00003 Data size (B)           512 Bandwidth (GBps)         0.04033
--Time (s)          0.00003 Data size (B)          1024 Bandwidth (GBps)         0.08017
--Time (s)          0.00002 Data size (B)          2048 Bandwidth (GBps)         0.16789
--Time (s)          0.00003 Data size (B)          4096 Bandwidth (GBps)         0.27093
--Time (s)          0.00003 Data size (B)          8192 Bandwidth (GBps)         0.57978
--Time (s)          0.00003 Data size (B)         16384 Bandwidth (GBps)         1.05693
--Time (s)          0.00004 Data size (B)         32768 Bandwidth (GBps)         1.56283
--Time (s)          0.00006 Data size (B)         65536 Bandwidth (GBps)         2.12329
--Time (s)          0.00010 Data size (B)        131072 Bandwidth (GBps)         2.51498
--Time (s)          0.00019 Data size (B)        262144 Bandwidth (GBps)         2.78053
--Time (s)          0.00036 Data size (B)        524288 Bandwidth (GBps)         2.93066
--Time (s)          0.00066 Data size (B)       1048576 Bandwidth (GBps)         3.15746
--Time (s)          0.00112 Data size (B)       2097152 Bandwidth (GBps)         3.73963
--Time (s)          0.00203 Data size (B)       4194304 Bandwidth (GBps)         4.12282
--Time (s)          0.00438 Data size (B)       8388608 Bandwidth (GBps)         3.82907
--Time (s)          0.00733 Data size (B)      16777216 Bandwidth (GBps)         4.58024
--Time (s)          0.01343 Data size (B)      33554432 Bandwidth (GBps)         4.99637
--Time (s)          0.02581 Data size (B)      67108864 Bandwidth (GBps)         5.19988
--Time (s)          0.05028 Data size (B)     134217728 Bandwidth (GBps)         5.33868
--Time (s)          0.09886 Data size (B)     268435456 Bandwidth (GBps)         5.43059
--Time (s)          0.19354 Data size (B)     536870912 Bandwidth (GBps)         5.54795
--Time (s)          0.38534 Data size (B)    1073741824 Bandwidth (GBps)         5.57296
```

The speed of transferring 1 GB of data between 2 MPI-processes is about 6 GB/s

Too slow!!



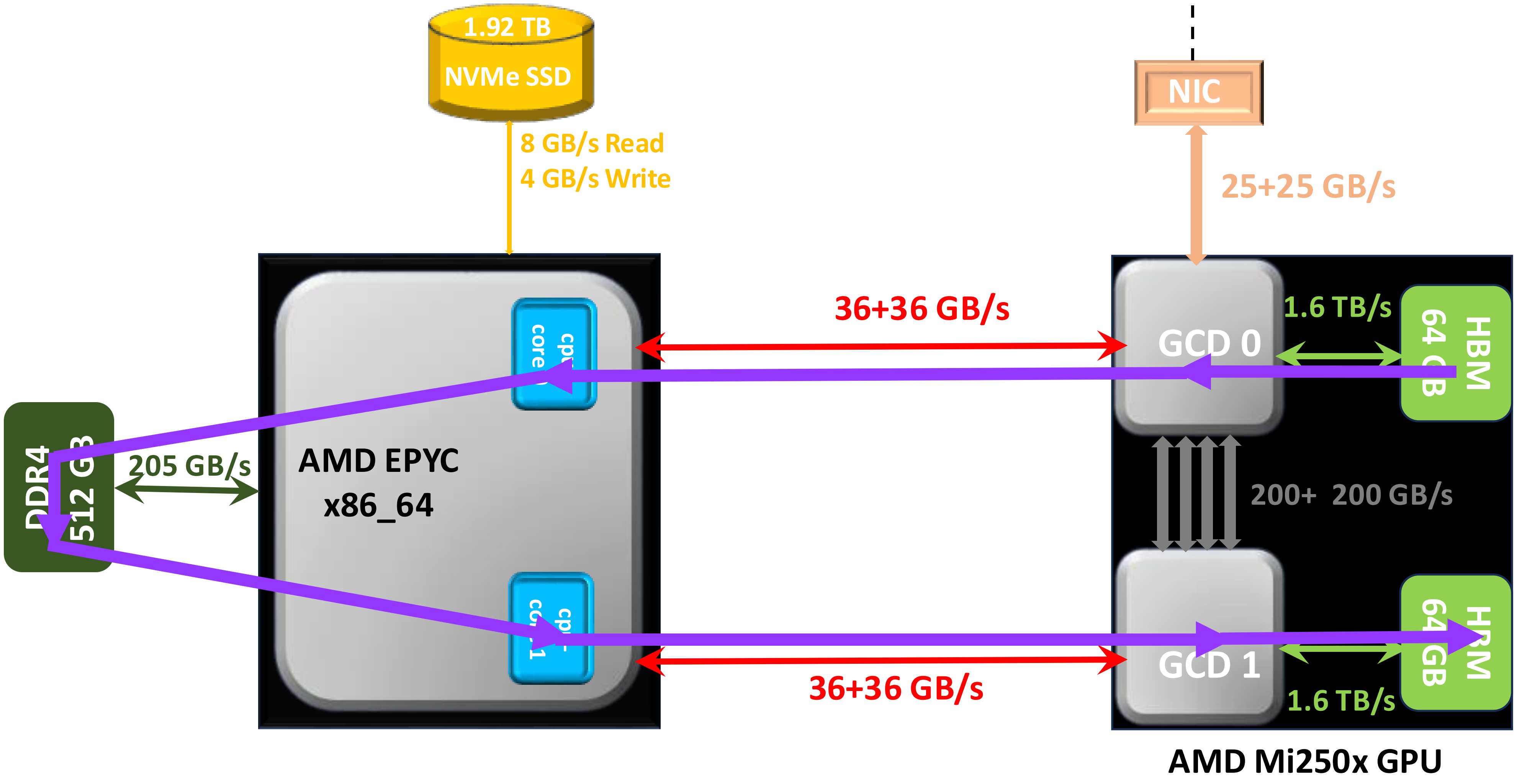
AMD Mi250x GPU

GCD === ❯GPU

Bidirectional bandwidths

⟷	Infinity Fabric	**36 + 36 GB/s**
⟷	Infinity Fabric	**200 + 200 GB/s**
⟷	PCIe Gen4 ESM	**50 + 50 GB/s**
⟷	PCIe Gen4	**8 + 8 GB/s**
- - - - - - -	Ethernet	**25 + 25 GB/s**

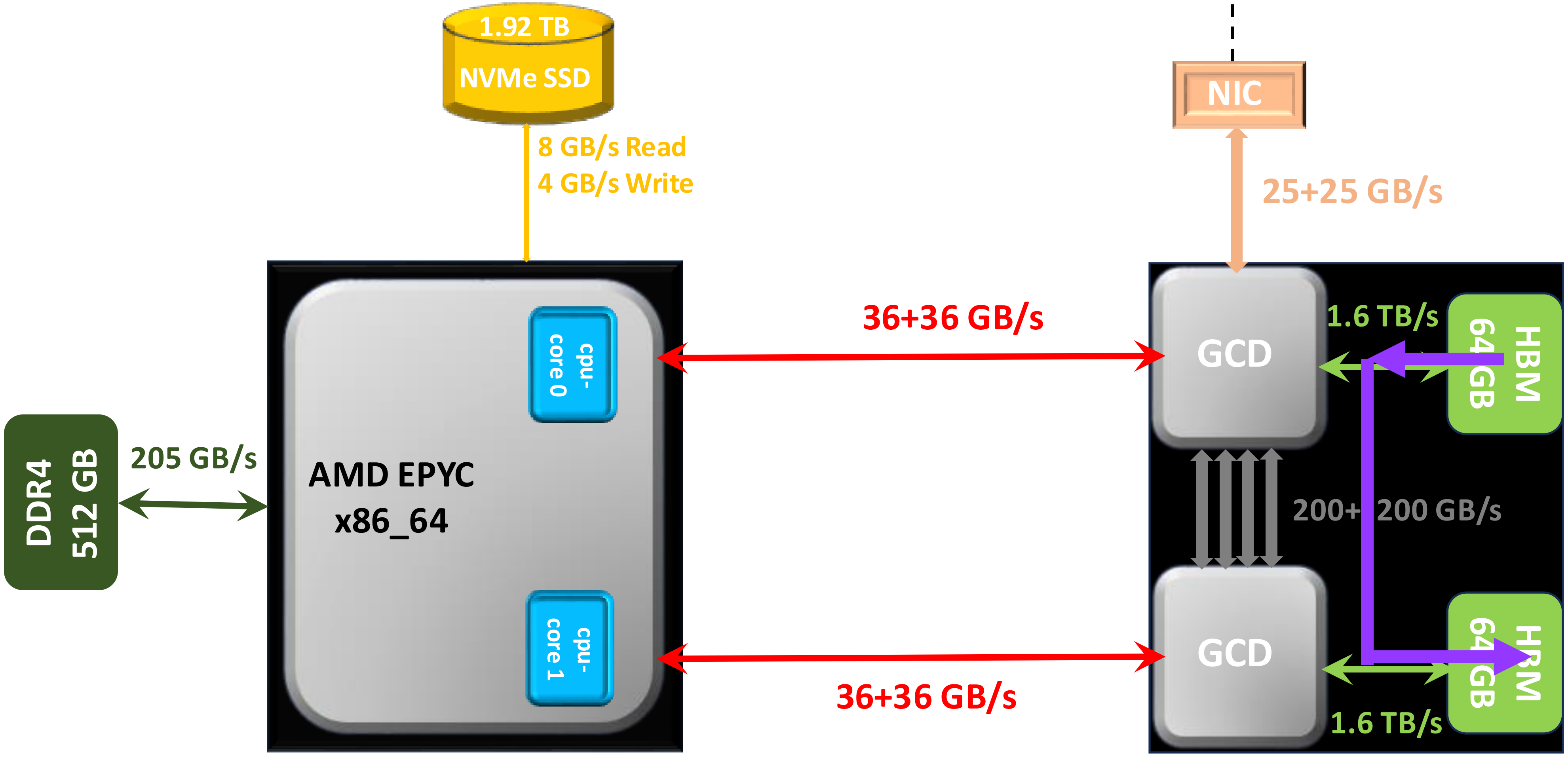
Bandwidth values are taken from LUMI doc



GCD === XGPU

Bidirectional bandwidths

🟥	Infinity Fabric	**36 + 36 GB/s**
▨	Infinity Fabric	**200 + 200 GB/s**
🟧	PCIe Gen4 ESM	**50 + 50 GB/s**
🟨	PCIe Gen4	**8 + 8 GB/s**
- - - - - - -	Ethernet	**25 + 25 GB/s**

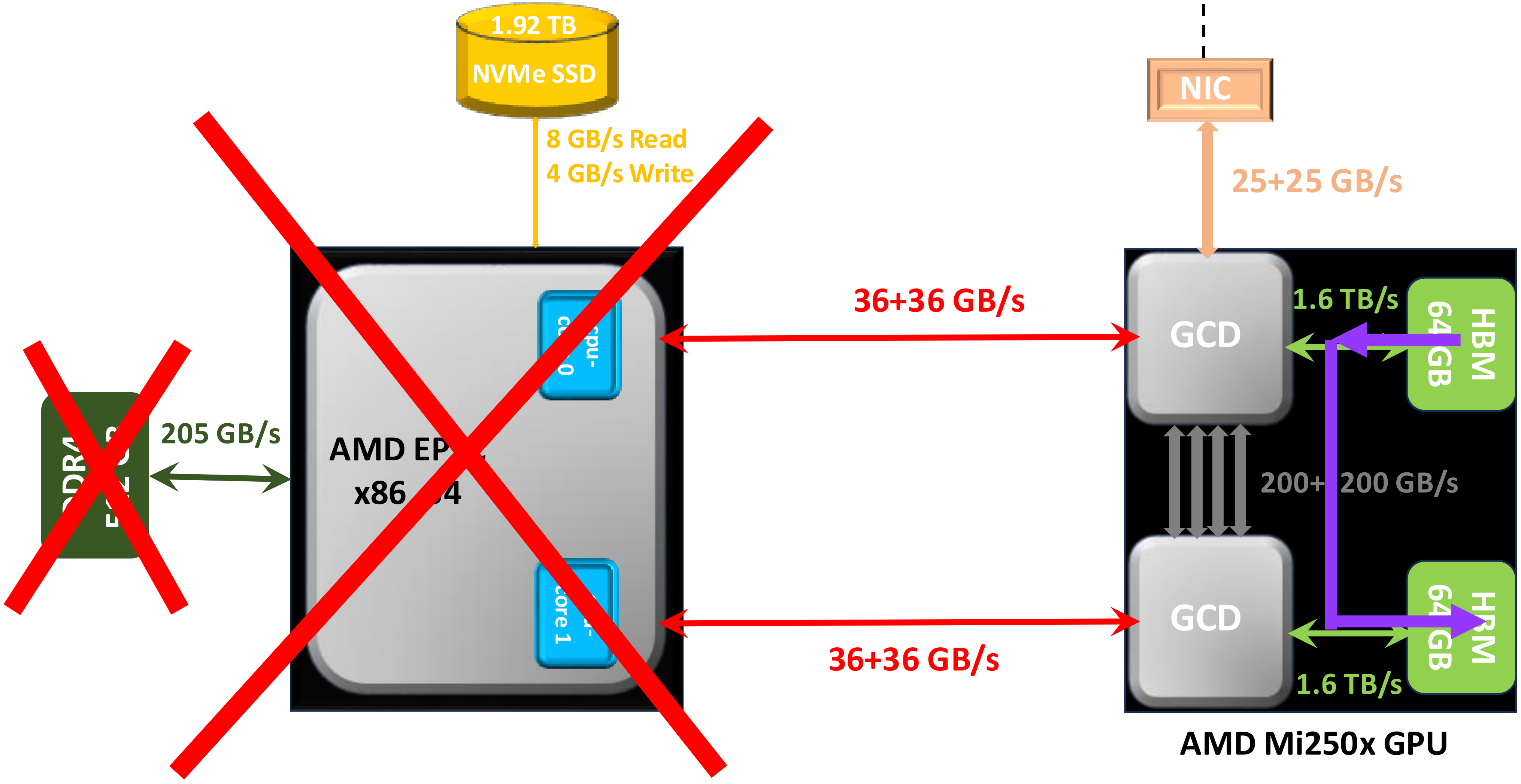


AMD Mi250x GPU

GCD === XGPU

Bidirectional bandwidths

🟥	Infinity Fabric	36 + 36	GB/s
⬜	Infinity Fabric	200 + 200	GB/s
🟧	PCIe Gen4 ESM	50 + 50	GB/s
🟨	PCIe Gen4	8 + 8	GB/s
- - - - - - -	Ethernet	25 + 25	GB/s



AMD Mi250x GPU

GCD === GPU

Bidirectional bandwidths

⟷	Infinity Fabric	**36 + 36**	**GB/s**
⟷	Infinity Fabric	**200 + 200**	**GB/s**
⟷	PCIe Gen4 ESM	**50 + 50**	**GB/s**
⟷	PCIe Gen4	**8 + 8**	**GB/s**
- - - - - - -	Ethernet	**25 + 25**	**GB/s**

<u>Concept</u>:
MPI with GPU awareness support

What is GPU-aware MPI ?

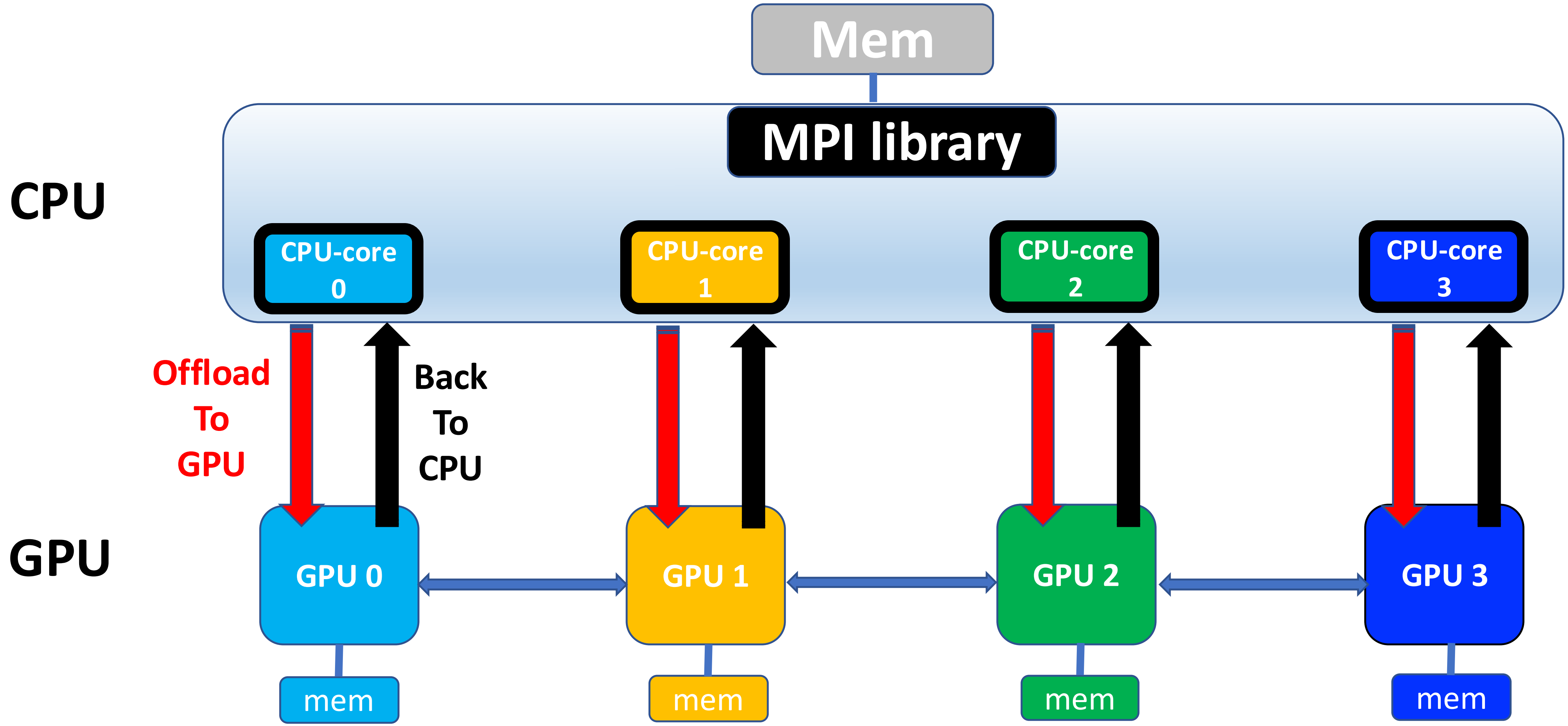
In GPU-aware MPI concept:

MPI library can directly access the GPU memory without necessarily passing by the CPU memory.

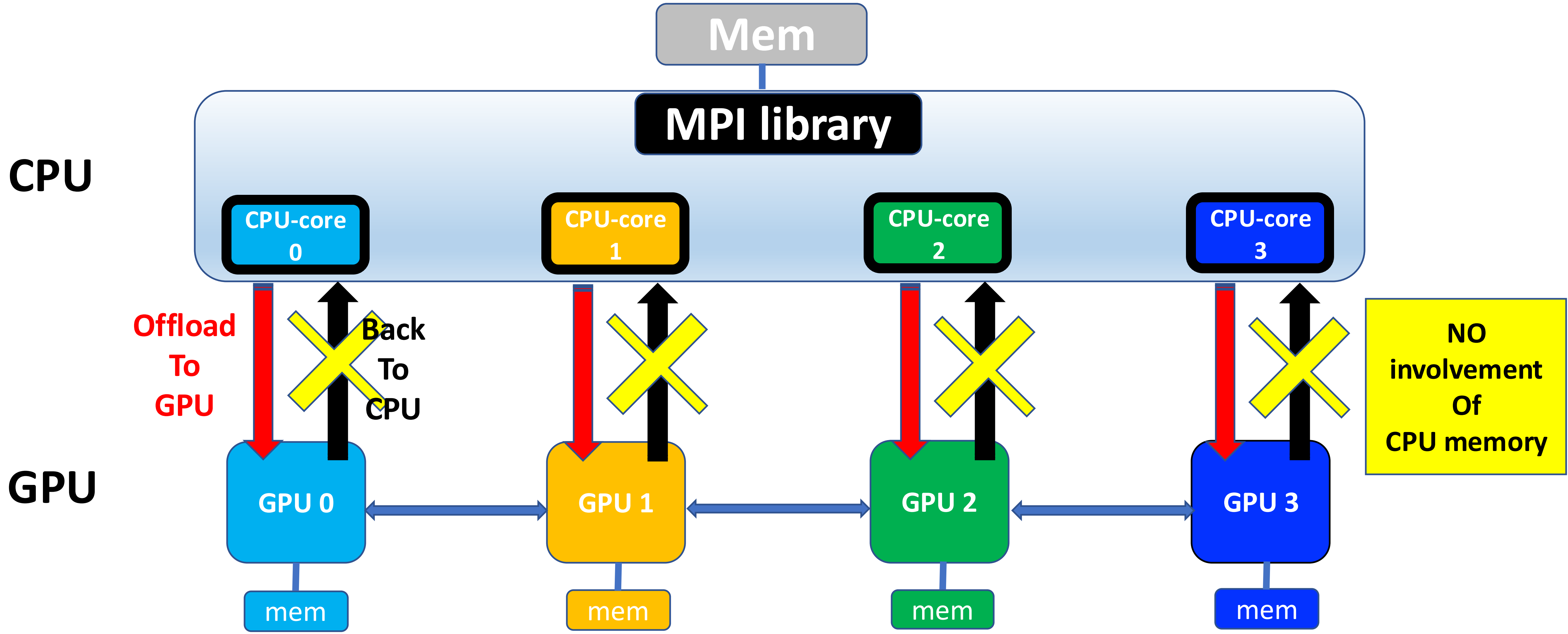
Takes advantage of GPUDirect Technology (RDMA and Peer-To-Peer)

Device pointers are passed as arguments to an MPI routine

Traditional MPI without GPU awareness

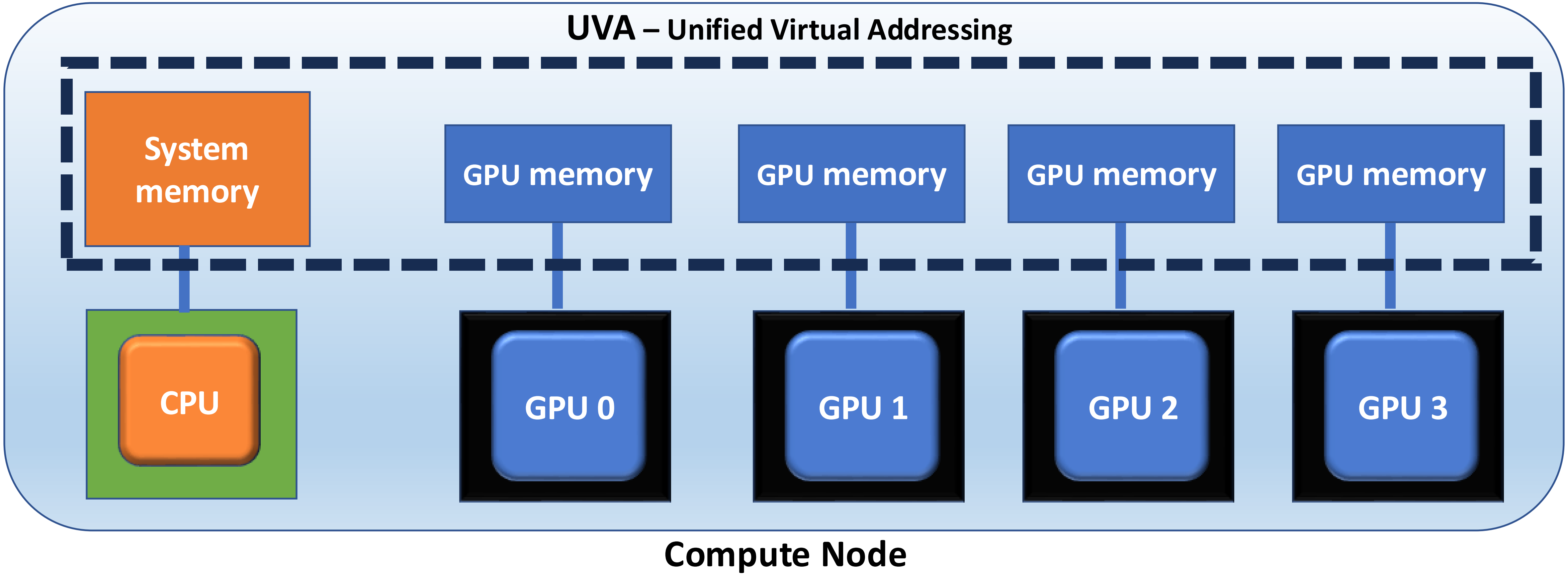


MPI with GPU awareness



MPI library directly access GPU memory without passing by the CPU memory

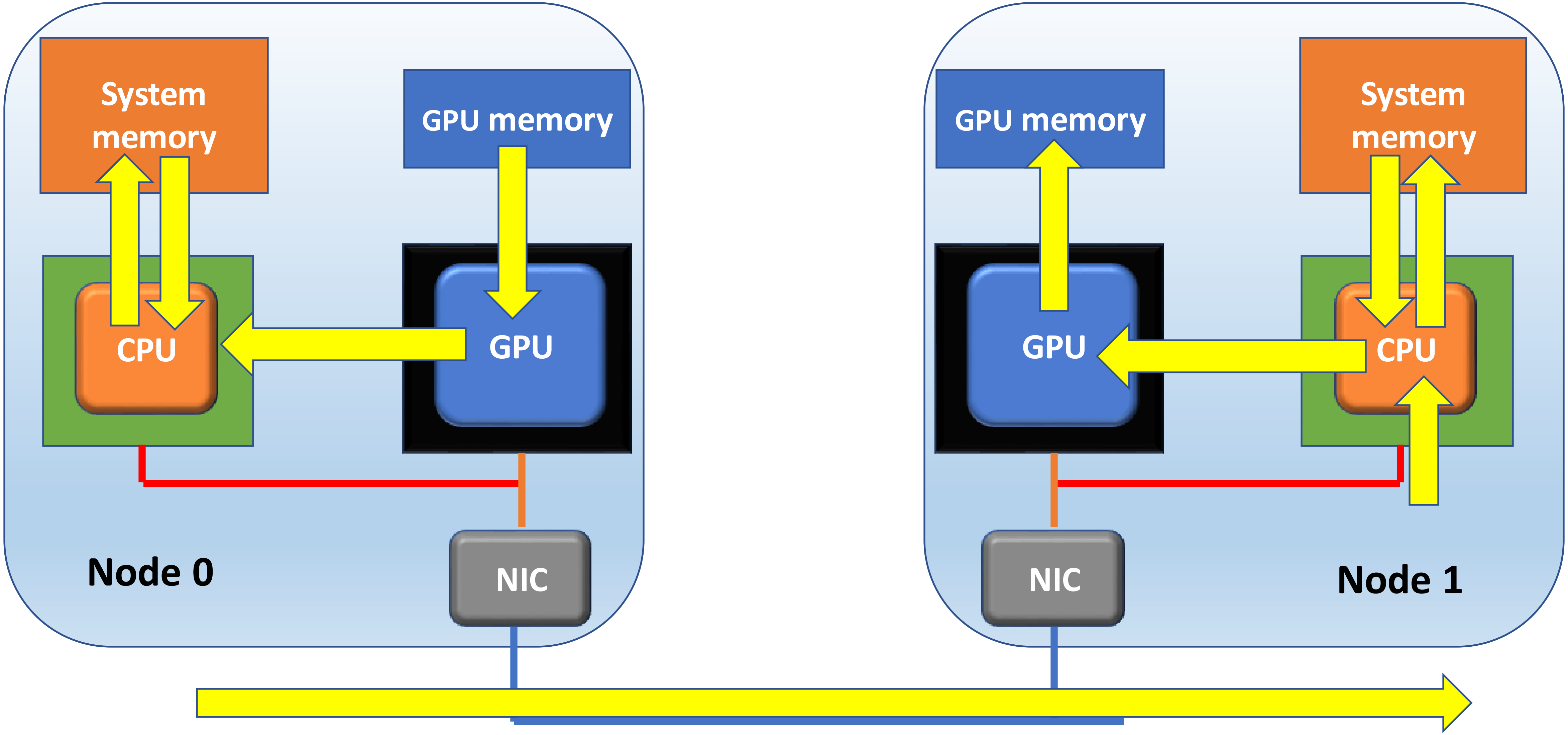
Single node: MPI with GPU awareness



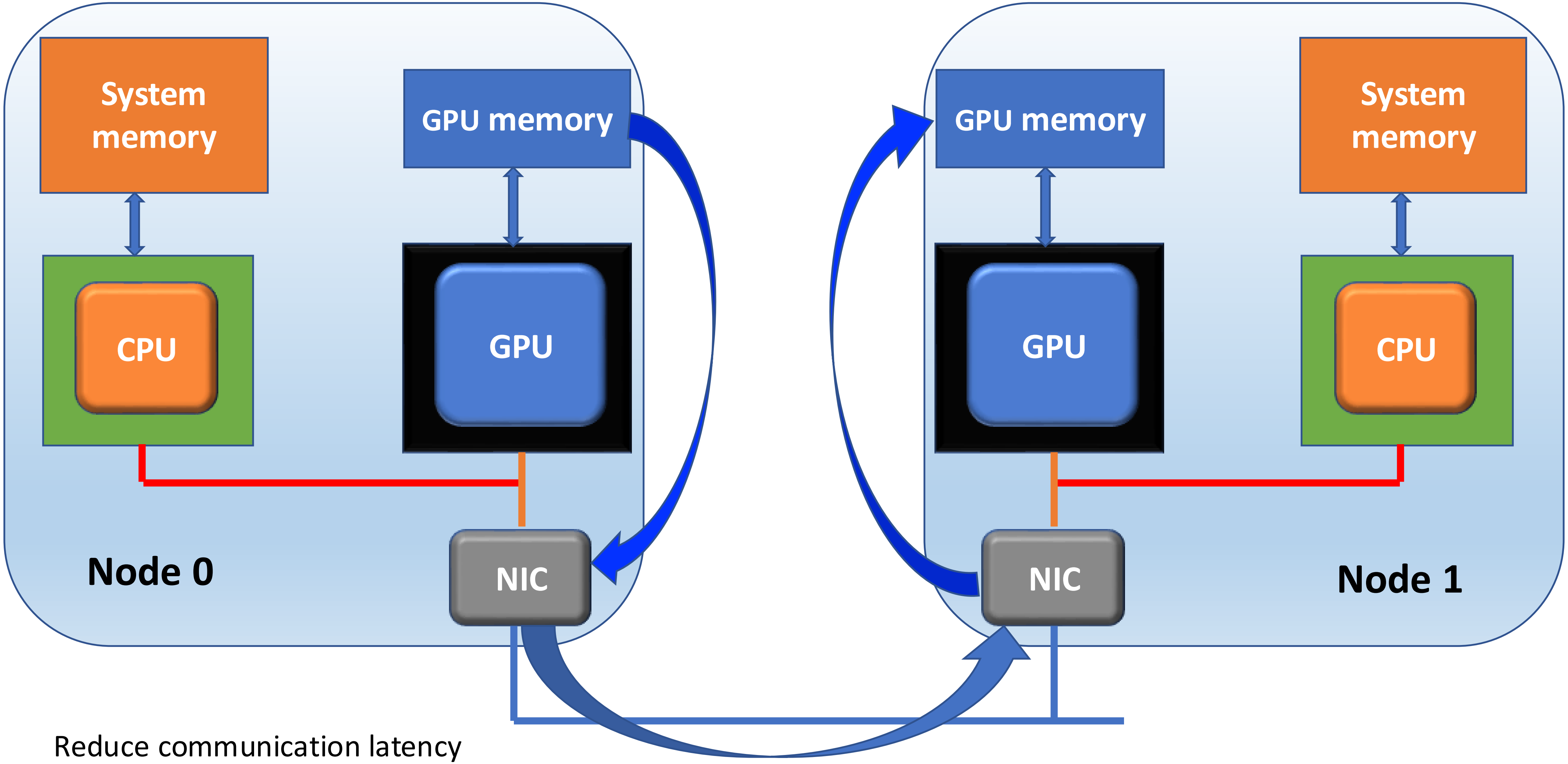
- Device pointers are passed as arguments to an MPI routine
- MPI library will detect that the pointer is a device pointer
- Unified Virtual Addressing (UVA): single virtual memory system for all memory (GPUs, CPU)
- Direct GPU-to-GPU communication

https://developer.nvidia.com/blog/introduction-cuda-aware-mpi/

Multiple nodes: Traditional way of transferring data



Multiple nodes: GPUDirect RDMA (GDR)

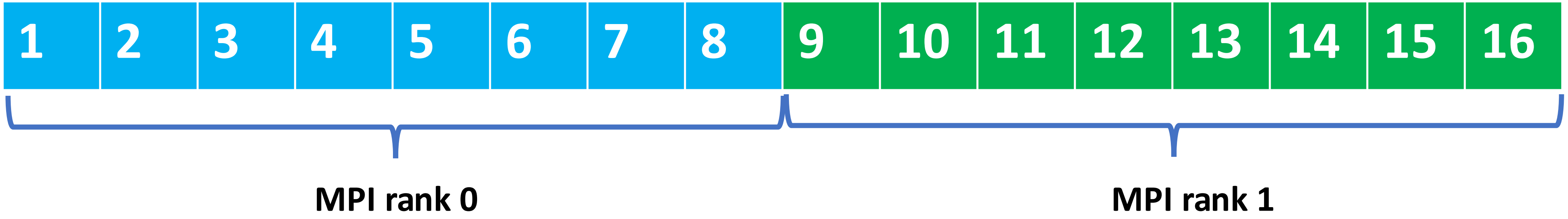


Reduce communication latency
Improve performance

Example: GPU-aware MPI with OpenACC/OpenMP APIs

- **MPI operations** : **MPI_Send & MPI_Recv**
- **OpenACC directive**: **host_data device_ptr()**
- **OpenMP directive**: **use_device_ptr()**

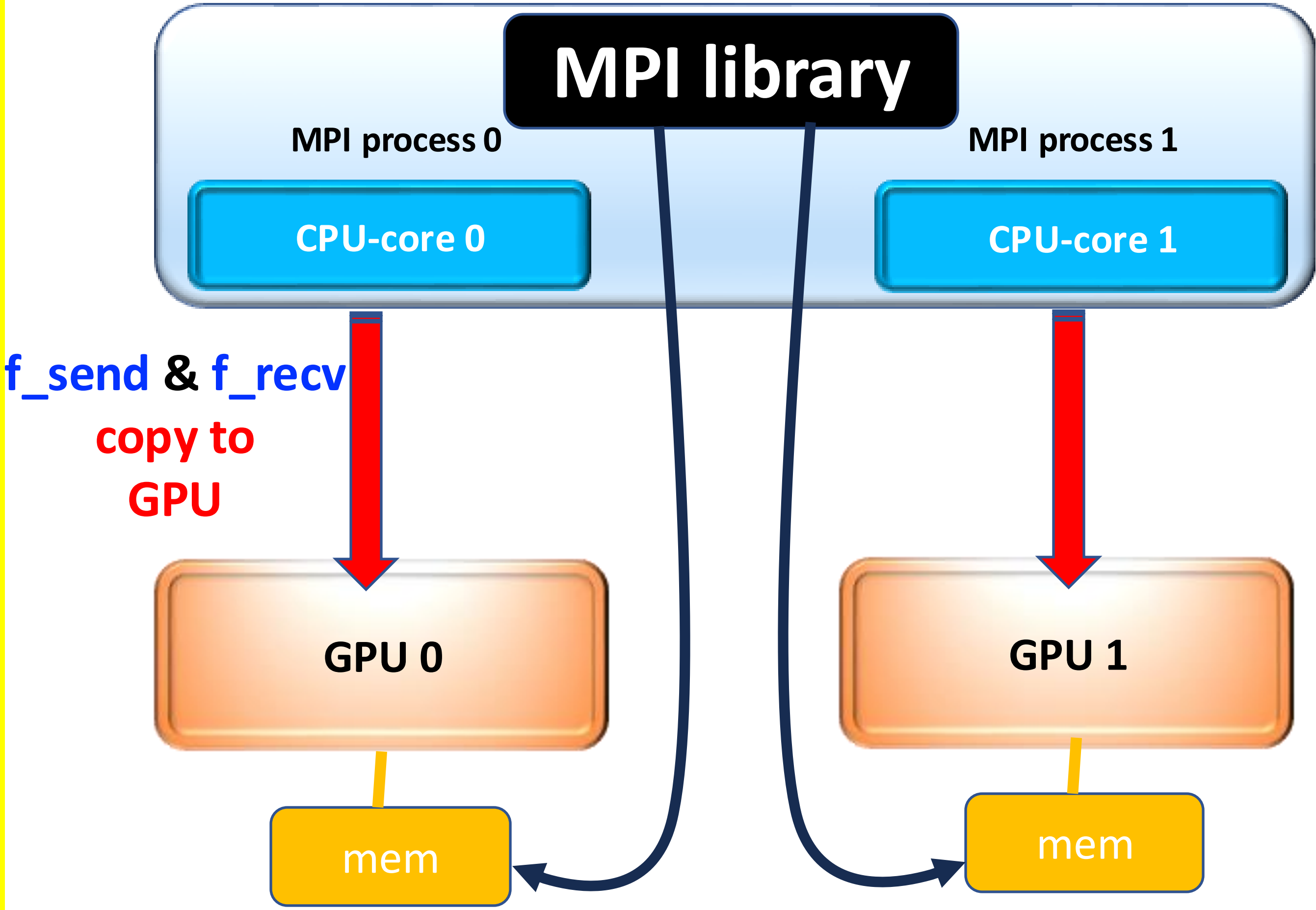
Scenario with only 2 MPI-processes: MPI_Send & MPI_Recv



```
if(MPIrank.eq.0) then
   call MPI_Send(f_send(:), N,MPI_DOUBLE_PRECISION, MPIrank=1, tag1, ..)
   call MPI_Recv(f_recv(:), N, MPI_DOUBLE_PRECISION, MPIrank=1, tag2, ..)
endif
if(MPIrank.eq.1) then
   call MPI_Recv(f_recv(:), N, MPI_DOUBLE_PRECISION, MPIrank=0, tag1, ..)
   call MPI_Send(f_send(:), N, MPI_DOUBLE_PRECISION, MPIrank=0, tag2, ..)
endif
```

GPU-aware MPI with OpenACC: MPI_Send & MPI_Recv

```
!$acc enter data copyin(f_send,f_recv)
Perform computation on GPU

!Device pointers f_send & f_recv are passed to MPI_send & MPI_recv
!$acc host_data use_device(f_send,f_recv)


  call MPI_Send(f_send, N , ……)

  call MPI_Recv(f_recv, N , ……..)


!$acc end host_data
Perform computation on GPU

!$acc exit data copyout(f_recv)
```



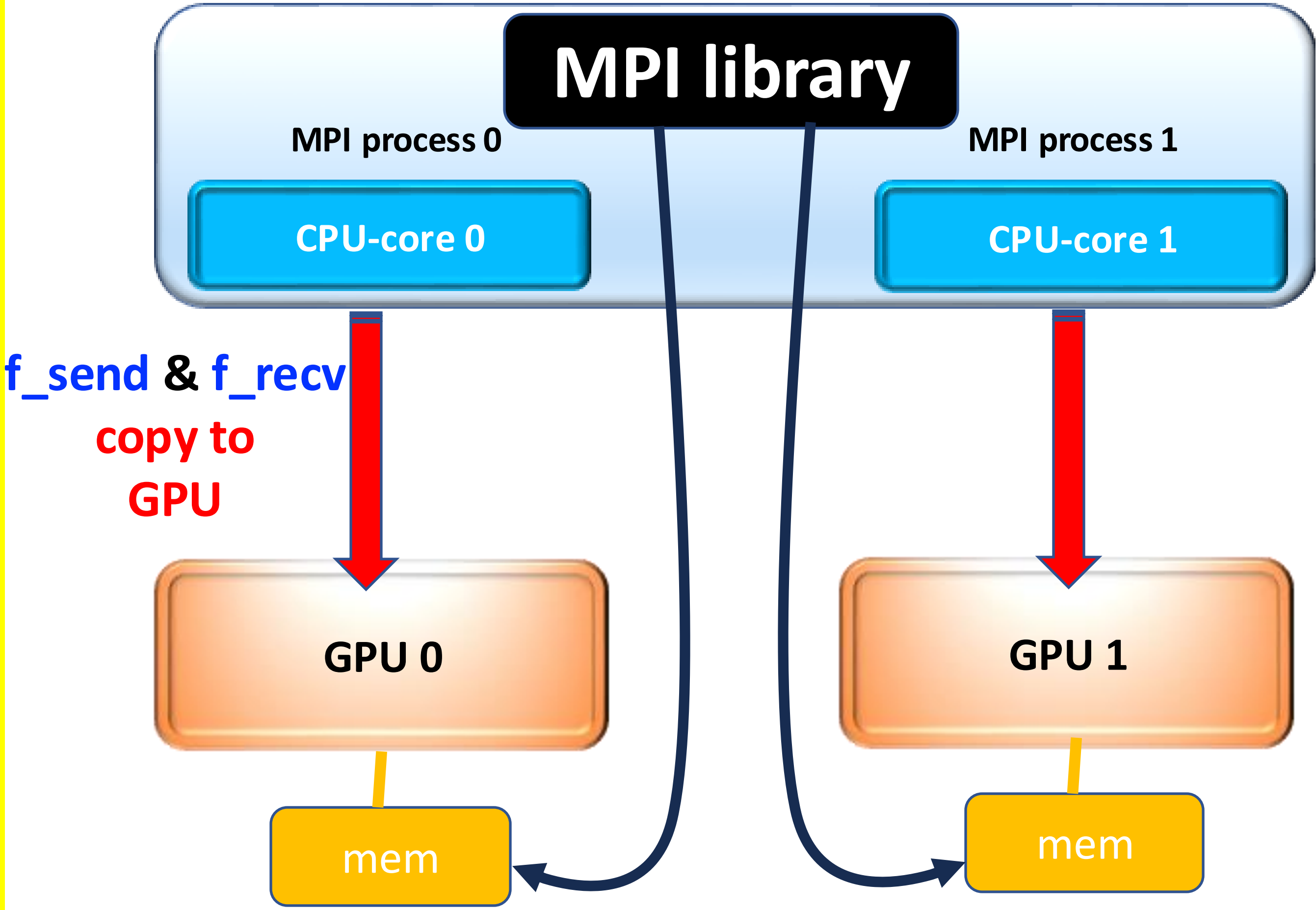
Communication can be seen as it is between a pair of GPUs

GPU-aware MPI with OpenACC: MPI_Send & MPI_Recv

```
!$acc enter data copyin(f_send,f_recv)
Do something on GPU

!Device pointers f_send & f_recv are passed to MPI_send & MPI_recv
!$acc host_data use_device(f_send,f_recv)


  call MPI_Send(f_send, N , ……)

  call MPI_Recv(f_recv, N , ……..)


!$acc end host_data

Do something on GPU

!$acc exit data copyout(f_recv)
```

To enable GPU-aware support in MPICH lib

```
$ export MPICH_GPU_SUPPORT_ENABLED=1
```



Communication can be seen as it is between a pair of GPUs

GPU-aware MPI with OpenACC

```
!$acc enter data copyin(f_send,f_recv)
Perform computation on GPU

!Device pointers f_send & f_recv are passed to MPI_send & MPI_recv
!$acc host_data use_device(f_send,f_recv)


  call MPI_Send(f_send, N , ......)

  call MPI_Recv(f_recv, N, ........)


!$acc end host_data

Perform computation on GPU

!$acc exit data copyout(f_recv)
```

GPU-aware MPI with OpenMP

```
!$omp target enter data map(to: f_send,f_recv)
Perform computation on GPU

!Device pointers f_send & f_recv are passed to MPI_send & MPI_recv
!$omp target data use_device_ptr(f_send,f_recv)


  call MPI_Send(f_send, N , ......)

  call MPI_Recv(f_recv, N, ........)


!$omp end target data

Perform computation on GPU

!$omp target exit data map(from: f_recv)
```

Hands-on Example 3: GPU-aware MPI with OpenACC/OpenMP ?

Purpose: To measure **the time it takes to transfer data** during **MPI communication**

- **Download the repo**

$ git clone **https://github.com/HichamAgueny/multiGPU_MPI_examples**
$ cd multiGPU_MPI_examples

- **For MPI-OpenACC**

$ cd example_3/gpuaware_mpiacc

- **For MPI-OpenMP**

$ cd example_3/gpuaware_mpiomp

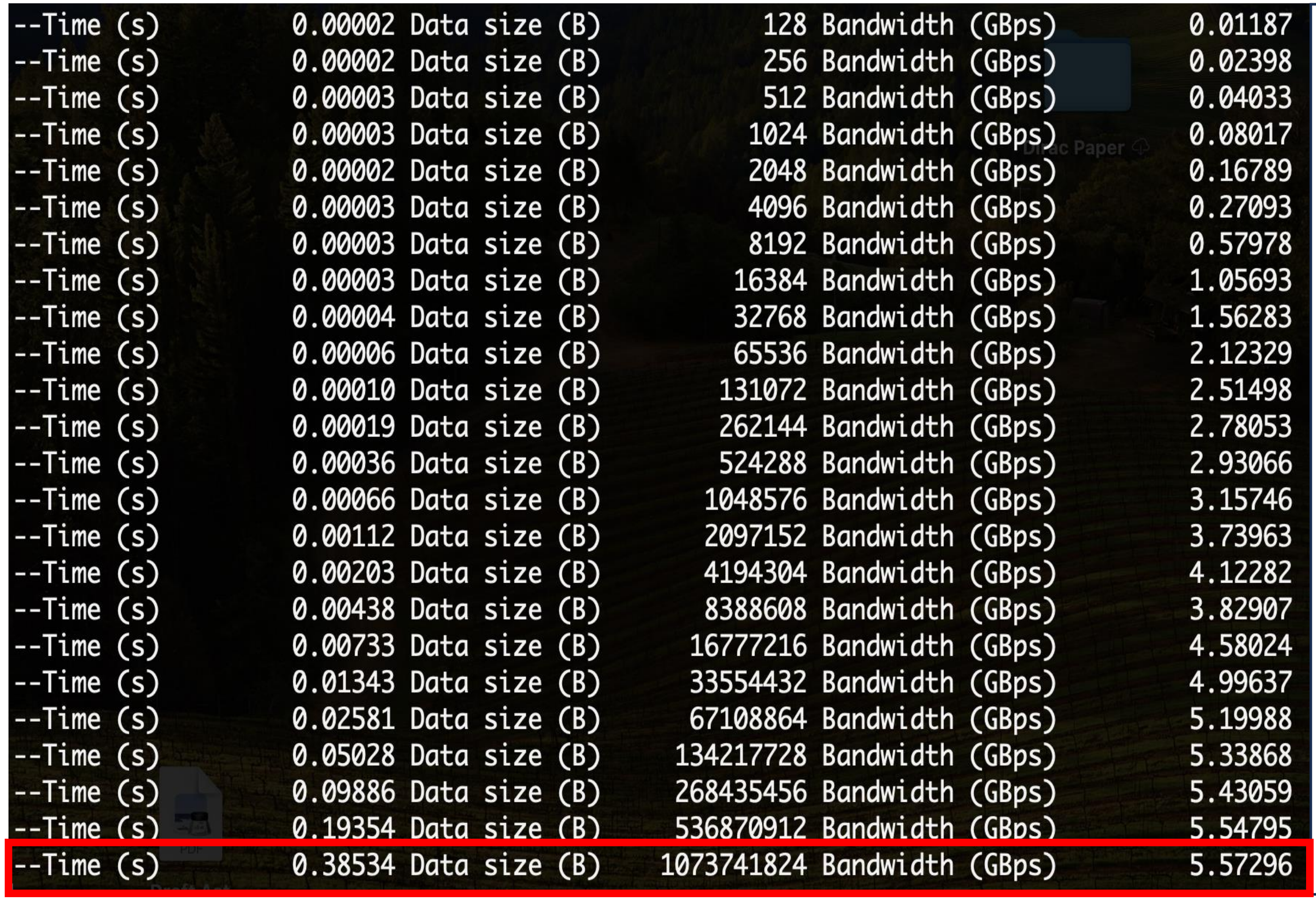
- **To compile and execute the code**

Load the LUMI software stack
$ module load LUMI/24.03 partition/G

Compile: $ ./compile.sh
Submit a job: $ sbatch script.slurm
View the output file: $ vi gpuaware_mpiacc-xxxxxx.out

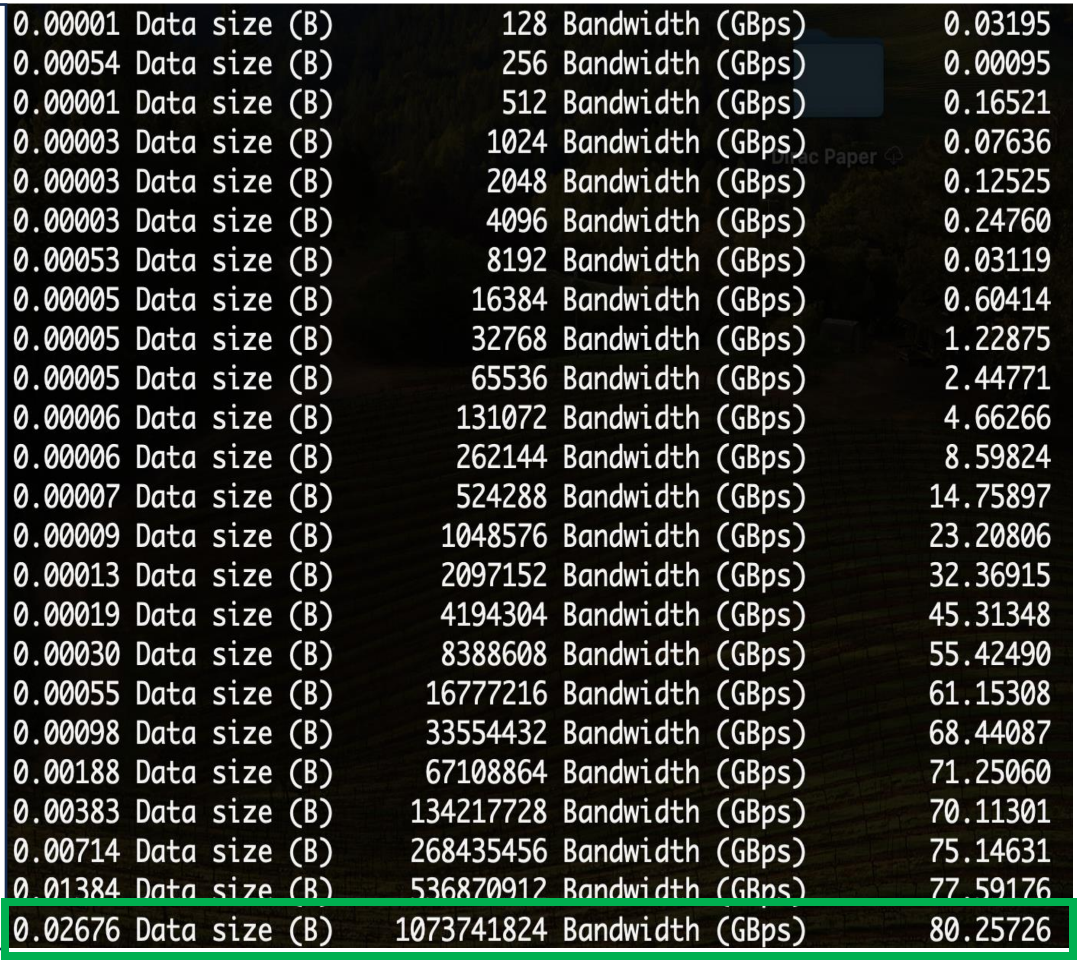
Measruing bandwidth: 2 MPI processes and 2 GPUs, single Node

Traditional MPI combined with OpenACC



```
--Time (s)          0.00002 Data size (B)          128 Bandwidth (GBps)         0.01187
--Time (s)          0.00002 Data size (B)          256 Bandwidth (GBps)         0.02398
--Time (s)          0.00003 Data size (B)          512 Bandwidth (GBps)         0.04033
--Time (s)          0.00003 Data size (B)         1024 Bandwidth (GBps)         0.08017
--Time (s)          0.00002 Data size (B)         2048 Bandwidth (GBps)         0.16789
--Time (s)          0.00003 Data size (B)         4096 Bandwidth (GBps)         0.27093
--Time (s)          0.00003 Data size (B)         8192 Bandwidth (GBps)         0.57978
--Time (s)          0.00003 Data size (B)        16384 Bandwidth (GBps)         1.05693
--Time (s)          0.00004 Data size (B)        32768 Bandwidth (GBps)         1.56283
--Time (s)          0.00006 Data size (B)        65536 Bandwidth (GBps)         2.12329
--Time (s)          0.00010 Data size (B)       131072 Bandwidth (GBps)         2.51498
--Time (s)          0.00019 Data size (B)       262144 Bandwidth (GBps)         2.78053
--Time (s)          0.00036 Data size (B)       524288 Bandwidth (GBps)         2.93066
--Time (s)          0.00066 Data size (B)      1048576 Bandwidth (GBps)         3.15746
--Time (s)          0.00112 Data size (B)      2097152 Bandwidth (GBps)         3.73963
--Time (s)          0.00203 Data size (B)      4194304 Bandwidth (GBps)         4.12282
--Time (s)          0.00438 Data size (B)      8388608 Bandwidth (GBps)         3.82907
--Time (s)          0.00733 Data size (B)     16777216 Bandwidth (GBps)         4.58024
--Time (s)          0.01343 Data size (B)     33554432 Bandwidth (GBps)         4.99637
--Time (s)          0.02581 Data size (B)     67108864 Bandwidth (GBps)         5.19988
--Time (s)          0.05028 Data size (B)    134217728 Bandwidth (GBps)         5.33868
--Time (s)          0.09886 Data size (B)    268435456 Bandwidth (GBps)         5.43059
--Time (s)          0.19354 Data size (B)    536870912 Bandwidth (GBps)         5.54795
--Time (s)          0.38534 Data size (B)   1073741824 Bandwidth (GBps)         5.57296
```

GPU-aware MPI with OpenACC

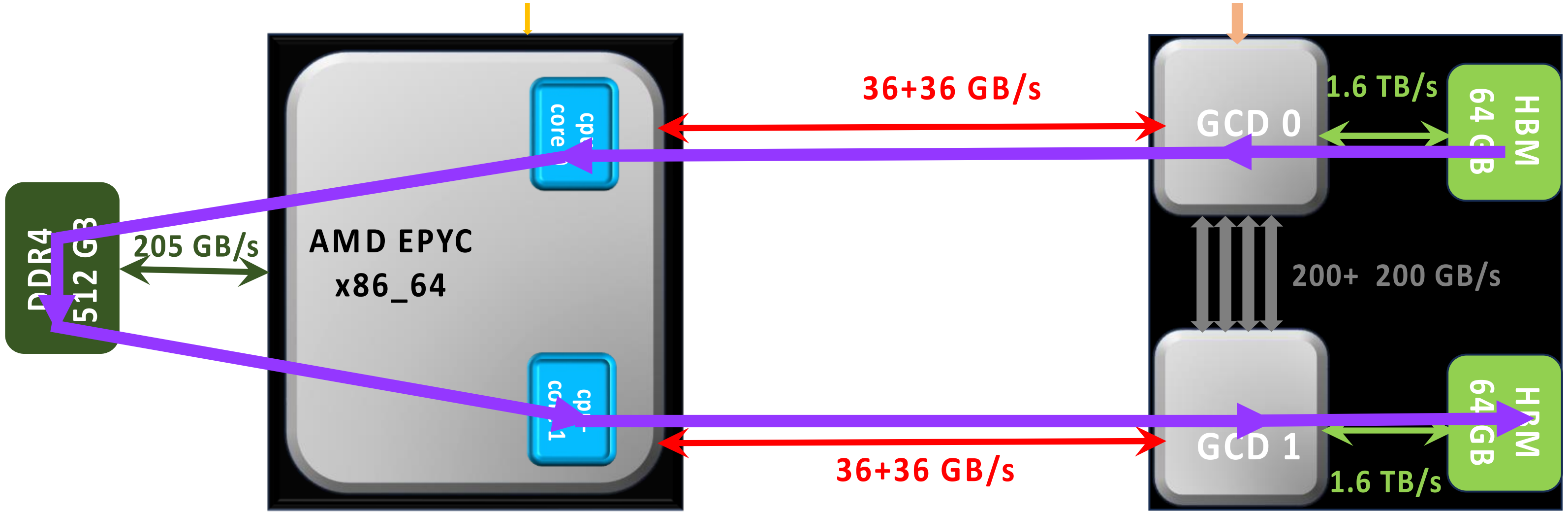


```
0.00001 Data size (B)          128 Bandwidth (GBps)         0.03195
0.00054 Data size (B)          256 Bandwidth (GBps)         0.00095
0.00001 Data size (B)          512 Bandwidth (GBps)         0.16521
0.00003 Data size (B)         1024 Bandwidth (GBps)         0.07636
0.00003 Data size (B)         2048 Bandwidth (GBps)         0.12525
0.00003 Data size (B)         4096 Bandwidth (GBps)         0.24760
0.00053 Data size (B)         8192 Bandwidth (GBps)         0.03119
0.00005 Data size (B)        16384 Bandwidth (GBps)         0.60414
0.00005 Data size (B)        32768 Bandwidth (GBps)         1.22875
0.00005 Data size (B)        65536 Bandwidth (GBps)         2.44771
0.00006 Data size (B)       131072 Bandwidth (GBps)         4.66266
0.00006 Data size (B)       262144 Bandwidth (GBps)         8.59824
0.00007 Data size (B)       524288 Bandwidth (GBps)        14.75897
0.00009 Data size (B)      1048576 Bandwidth (GBps)        23.20806
0.00013 Data size (B)      2097152 Bandwidth (GBps)        32.36915
0.00019 Data size (B)      4194304 Bandwidth (GBps)        45.31348
0.00030 Data size (B)      8388608 Bandwidth (GBps)        55.42490
0.00055 Data size (B)     16777216 Bandwidth (GBps)        61.15308
0.00098 Data size (B)     33554432 Bandwidth (GBps)        68.44087
0.00188 Data size (B)     67108864 Bandwidth (GBps)        71.25060
0.00383 Data size (B)    134217728 Bandwidth (GBps)        70.11301
0.00714 Data size (B)    268435456 Bandwidth (GBps)        75.14631
0.01384 Data size (B)    536870912 Bandwidth (GBps)        77.59176
0.02676 Data size (B)   1073741824 Bandwidth (GBps)        80.25726
```

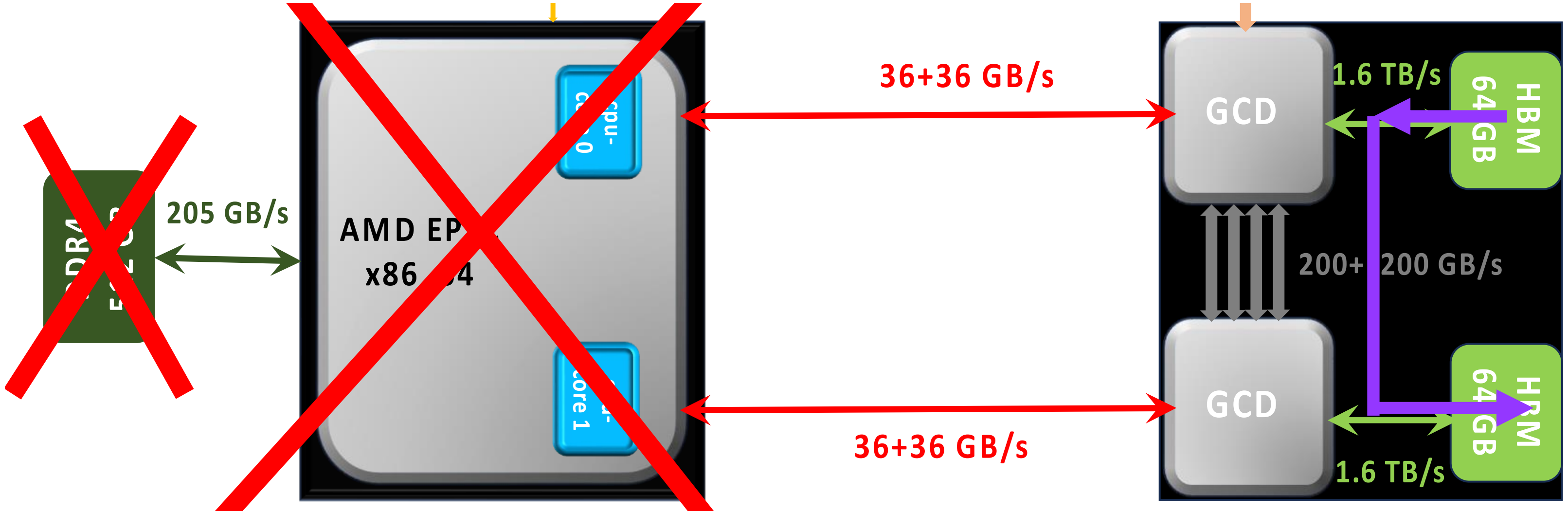
The speed of transferring 1 GB of data between 2 MPI-processes is about:

Traditional MPI 6 GB/s vs 80 GB/s GPU-aware MPI

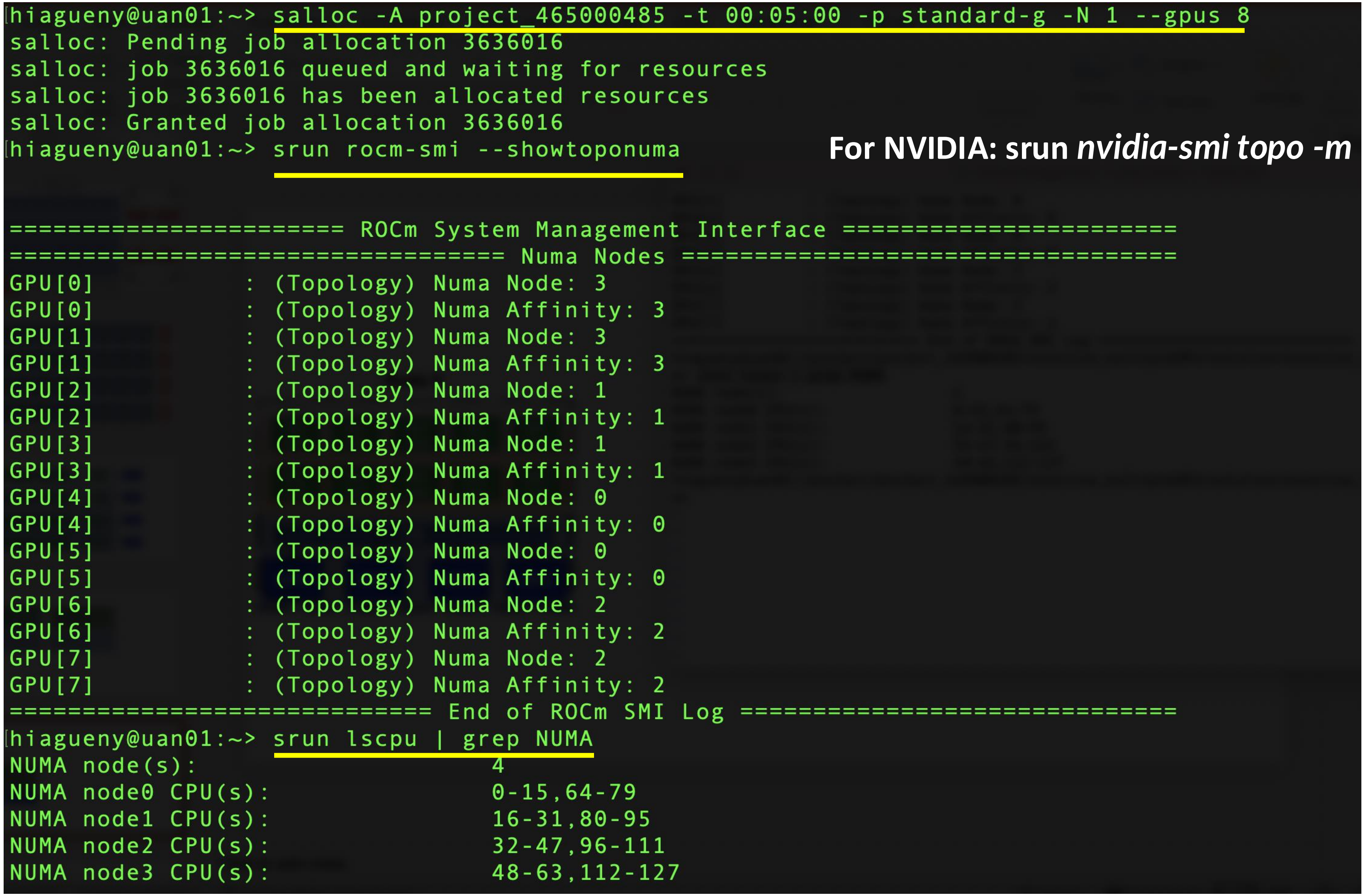
Traditional MPI



GPU-aware MPI



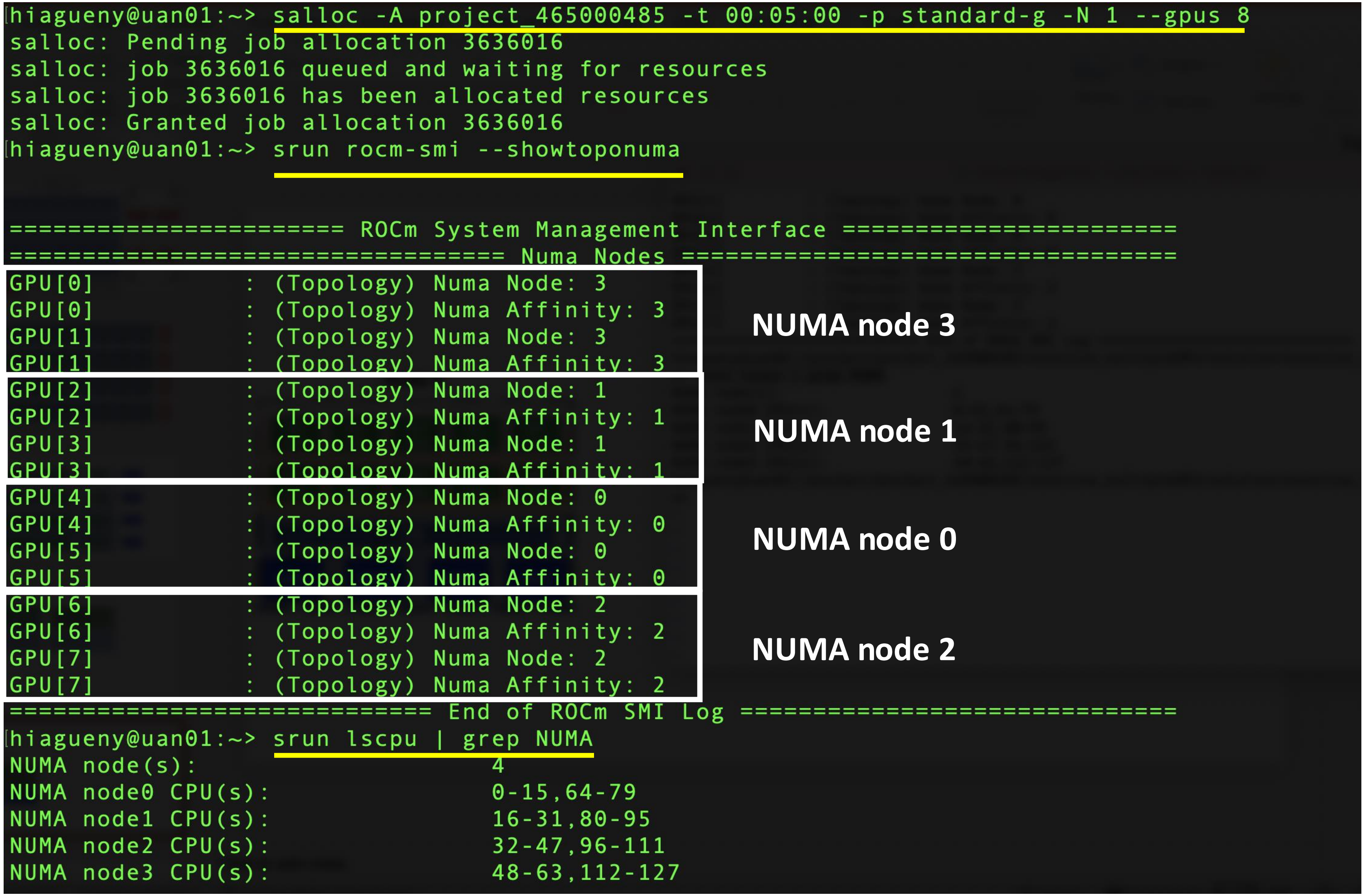
GPU-Binding (Efficient data transfer)



```
[hiagueny@uan01:~> salloc -A project_465000485 -t 00:05:00 -p standard-g -N 1 --gpus 8
salloc: Pending job allocation 3636016
salloc: job 3636016 queued and waiting for resources
salloc: job 3636016 has been allocated resources
salloc: Granted job allocation 3636016
[hiagueny@uan01:~> srun rocm-smi --showtoponuma


======================= ROCm System Management Interface =======================
================================== Numa Nodes ==================================
GPU[0]          : (Topology) Numa Node: 3
GPU[0]          : (Topology) Numa Affinity: 3
GPU[1]          : (Topology) Numa Node: 3
GPU[1]          : (Topology) Numa Affinity: 3
GPU[2]          : (Topology) Numa Node: 1
GPU[2]          : (Topology) Numa Affinity: 1
GPU[3]          : (Topology) Numa Node: 1
GPU[3]          : (Topology) Numa Affinity: 1
GPU[4]          : (Topology) Numa Node: 0
GPU[4]          : (Topology) Numa Affinity: 0
GPU[5]          : (Topology) Numa Node: 0
GPU[5]          : (Topology) Numa Affinity: 0
GPU[6]          : (Topology) Numa Node: 2
GPU[6]          : (Topology) Numa Affinity: 2
GPU[7]          : (Topology) Numa Node: 2
GPU[7]          : (Topology) Numa Affinity: 2
============================= End of ROCm SMI Log ==============================
[hiagueny@uan01:~> srun lscpu | grep NUMA
NUMA node(s):                    4
NUMA node0 CPU(s):               0-15,64-79
NUMA node1 CPU(s):               16-31,80-95
NUMA node2 CPU(s):               32-47,96-111
NUMA node3 CPU(s):               48-63,112-127
```

For NVIDIA: srun *nvidia-smi topo -m*

Binding option: CPU-GPU affinity



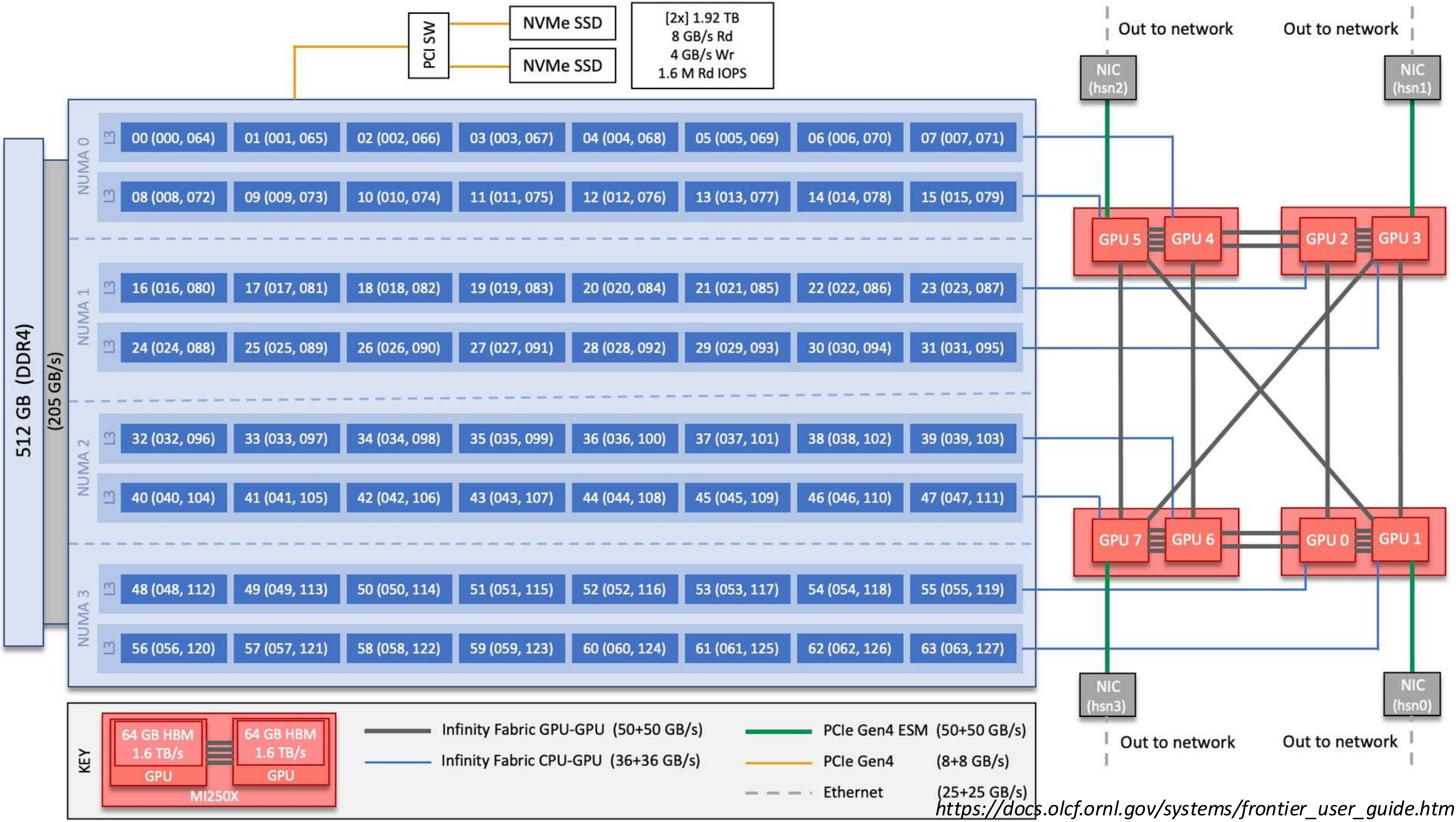
```
hiagueny@uan01:~> salloc -A project_465000485 -t 00:05:00 -p standard-g -N 1 --gpus 8
salloc: Pending job allocation 3636016
salloc: job 3636016 queued and waiting for resources
salloc: job 3636016 has been allocated resources
salloc: Granted job allocation 3636016
hiagueny@uan01:~> srun rocm-smi --showtoponuma


======================= ROCm System Management Interface =======================
================================== Numa Nodes ==================================
GPU[0]          : (Topology) Numa Node: 3
GPU[0]          : (Topology) Numa Affinity: 3
GPU[1]          : (Topology) Numa Node: 3
GPU[1]          : (Topology) Numa Affinity: 3
GPU[2]          : (Topology) Numa Node: 1
GPU[2]          : (Topology) Numa Affinity: 1
GPU[3]          : (Topology) Numa Node: 1
GPU[3]          : (Topology) Numa Affinity: 1
GPU[4]          : (Topology) Numa Node: 0
GPU[4]          : (Topology) Numa Affinity: 0
GPU[5]          : (Topology) Numa Node: 0
GPU[5]          : (Topology) Numa Affinity: 0
GPU[6]          : (Topology) Numa Node: 2
GPU[6]          : (Topology) Numa Affinity: 2
GPU[7]          : (Topology) Numa Node: 2
GPU[7]          : (Topology) Numa Affinity: 2
============================= End of ROCm SMI Log ==============================
hiagueny@uan01:~> srun lscpu | grep NUMA
NUMA node(s):                    4
NUMA node0 CPU(s):               0-15,64-79
NUMA node1 CPU(s):               16-31,80-95
NUMA node2 CPU(s):               32-47,96-111
NUMA node3 CPU(s):               48-63,112-127
```

NUMA node 3

NUMA node 1

NUMA node 0

NUMA node 2



512 GB (DDR4)

(205 GB/s)

NUMA 0	L3	00 (000, 064)	01 (001, 065)	02 (002, 066)	03 (003, 067)	04 (004, 068)	05 (005, 069)	06 (006, 070)	07 (007, 071)
	L3	08 (008, 072)	09 (009, 073)	10 (010, 074)	11 (011, 075)	12 (012, 076)	13 (013, 077)	14 (014, 078)	15 (015, 079)
NUMA 1	L3	16 (016, 080)	17 (017, 081)	18 (018, 082)	19 (019, 083)	20 (020, 084)	21 (021, 085)	22 (022, 086)	23 (023, 087)
	L3	24 (024, 088)	25 (025, 089)	26 (026, 090)	27 (027, 091)	28 (028, 092)	29 (029, 093)	30 (030, 094)	31 (031, 095)
NUMA 2	L3	32 (032, 096)	33 (033, 097)	34 (034, 098)	35 (035, 099)	36 (036, 100)	37 (037, 101)	38 (038, 102)	39 (039, 103)
	L3	40 (040, 104)	41 (041, 105)	42 (042, 106)	43 (043, 107)	44 (044, 108)	45 (045, 109)	46 (046, 110)	47 (047, 111)
NUMA 3	L3	48 (048, 112)	49 (049, 113)	50 (050, 114)	51 (051, 115)	52 (052, 116)	53 (053, 117)	54 (054, 118)	55 (055, 119)
	L3	56 (056, 120)	57 (057, 121)	58 (058, 122)	59 (059, 123)	60 (060, 124)	61 (061, 125)	62 (062, 126)	63 (063, 127)

PCI SW

NVMe SSD

NVMe SSD

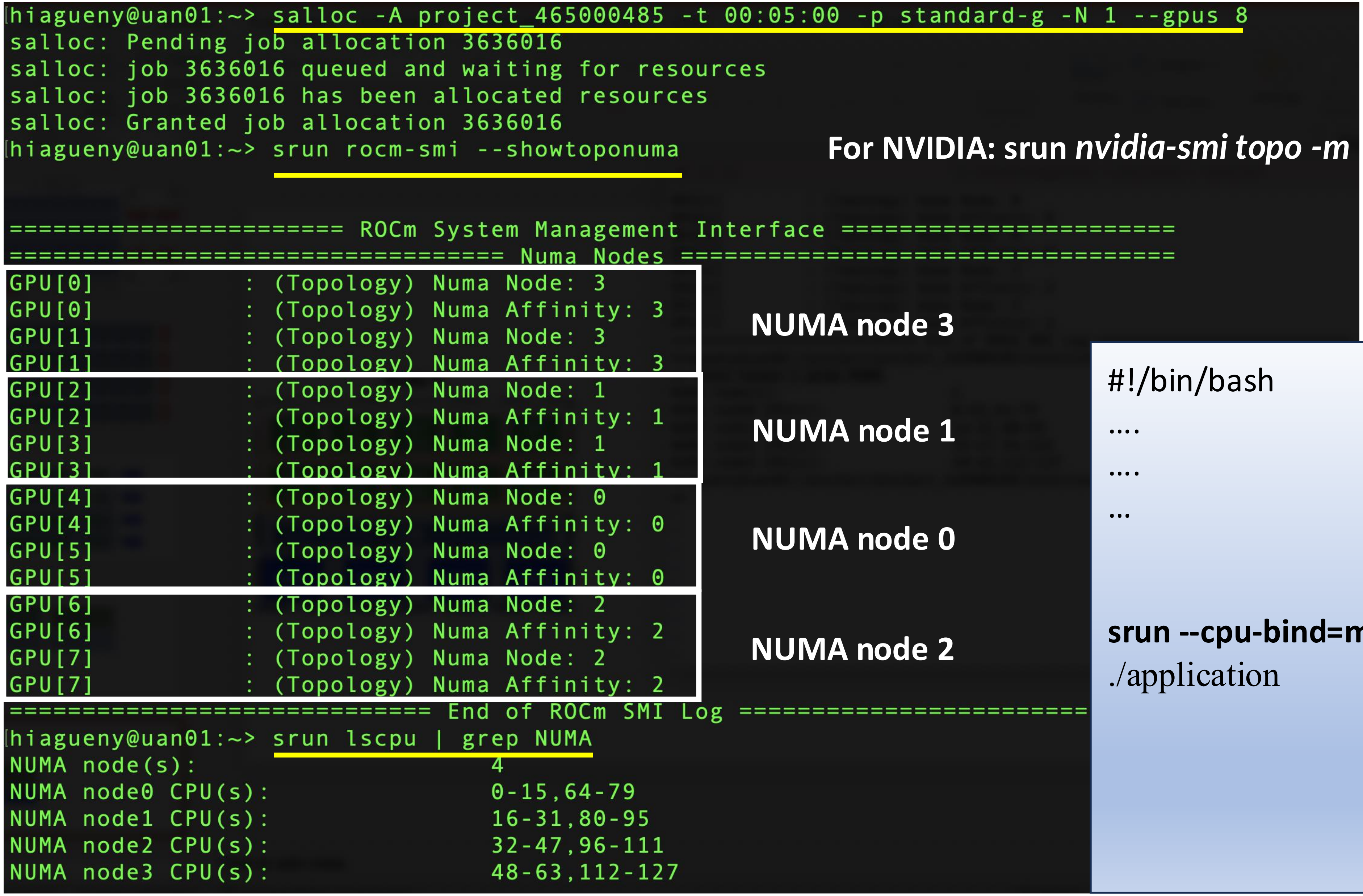
[2x] 1.92 TB
8 GB/s Rd
4 GB/s Wr
1.6 M Rd IOPS

KEY

▬	Infinity Fabric GPU-GPU	(50+50 GB/s)
▬	Infinity Fabric CPU-GPU	(36+36 GB/s)
▬	PCIe Gen4 ESM	(50+50 GB/s)
▬	PCIe Gen4	(8+8 GB/s)
- - -	Ethernet	(25+25 GB/s)

https://docs.olcf.ornl.gov/systems/frontier_user_guide.html

Binding option: CPU-GPU affinity



```
hiagueny@uan01:~> salloc -A project_465000485 -t 00:05:00 -p standard-g -N 1 --gpus 8
salloc: Pending job allocation 3636016
salloc: job 3636016 queued and waiting for resources
salloc: job 3636016 has been allocated resources
salloc: Granted job allocation 3636016
hiagueny@uan01:~> srun rocm-smi --showtoponuma


======================= ROCm System Management Interface =======================
================================== Numa Nodes ==================================
GPU[0]          : (Topology) Numa Node: 3
GPU[0]          : (Topology) Numa Affinity: 3
GPU[1]          : (Topology) Numa Node: 3
GPU[1]          : (Topology) Numa Affinity: 3
GPU[2]          : (Topology) Numa Node: 1
GPU[2]          : (Topology) Numa Affinity: 1
GPU[3]          : (Topology) Numa Node: 1
GPU[3]          : (Topology) Numa Affinity: 1
GPU[4]          : (Topology) Numa Node: 0
GPU[4]          : (Topology) Numa Affinity: 0
GPU[5]          : (Topology) Numa Node: 0
GPU[5]          : (Topology) Numa Affinity: 0
GPU[6]          : (Topology) Numa Node: 2
GPU[6]          : (Topology) Numa Affinity: 2
GPU[7]          : (Topology) Numa Node: 2
GPU[7]          : (Topology) Numa Affinity: 2
============================== End of ROCm SMI Log =========================
hiagueny@uan01:~> srun lscpu | grep NUMA
NUMA node(s):                    4
NUMA node0 CPU(s):               0-15,64-79
NUMA node1 CPU(s):               16-31,80-95
NUMA node2 CPU(s):               32-47,96-111
NUMA node3 CPU(s):               48-63,112-127
```

For NVIDIA: srun *nvidia-smi topo -m*

NUMA node 3

NUMA node 1

NUMA node 0

NUMA node 2

```
#!/bin/bash
….
….
…


srun --cpu-bind=map_cpu: 49,57, 17,25, 1,9, 33,41
./application
```

https://docs.lumi-supercomputer.eu/runjobs/scheduled-jobs/lumig-job/

Conclusion

Conclusion

Traditional MPI without GPU awareness:

- Explicit data transfer between CPU and GPU (with the involvement of CPU memory).
- GPU memory is not directly accessible from/by MPI library.

➡ **Additional overhead caused by data movement.**

MPI with GPU awareness:

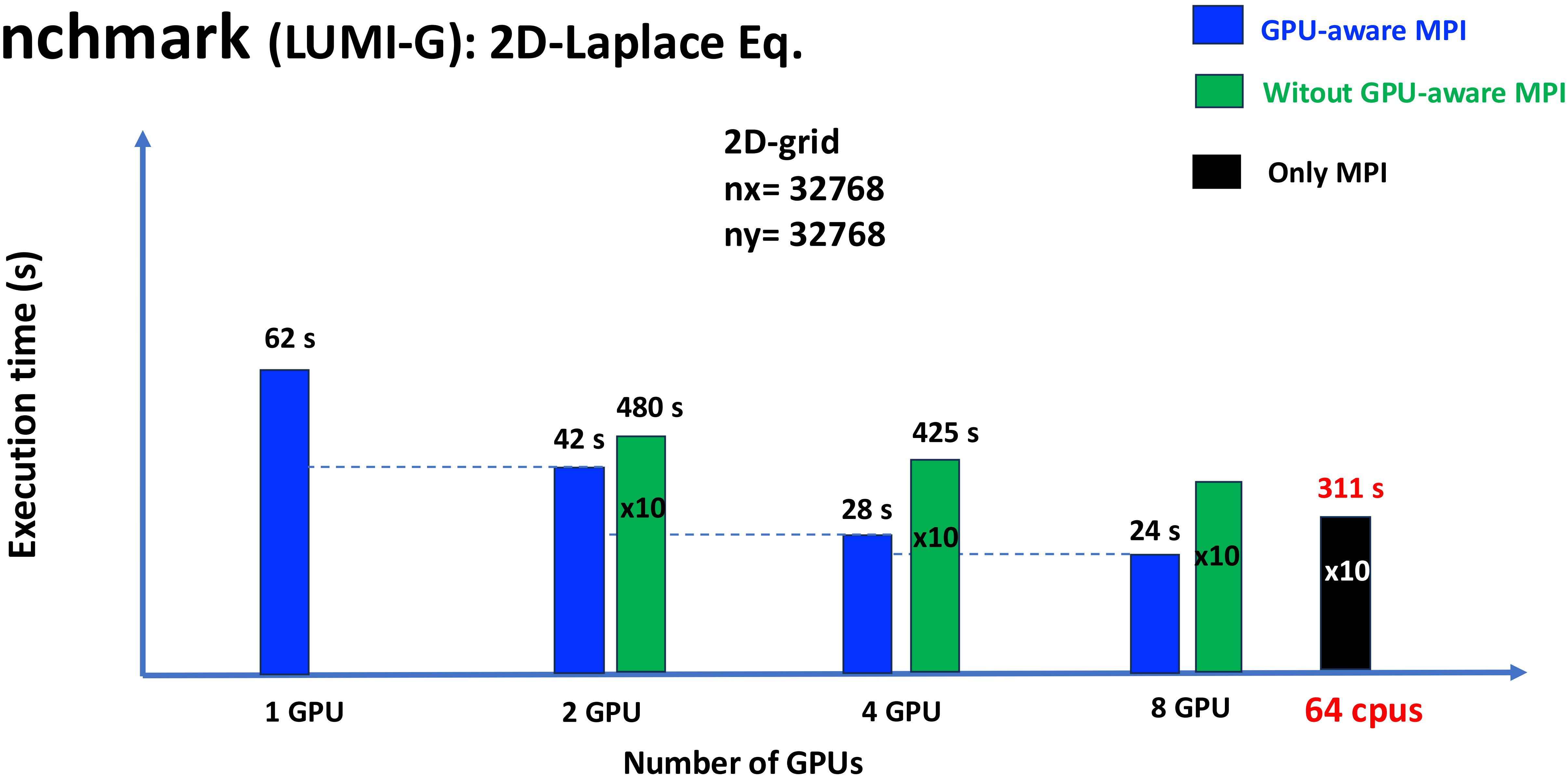
- MPI processes can directly access GPU memory (No involvement of CPU memory).
- Eliminiation of CPU-GPU data transfer.

Performance Benefits:

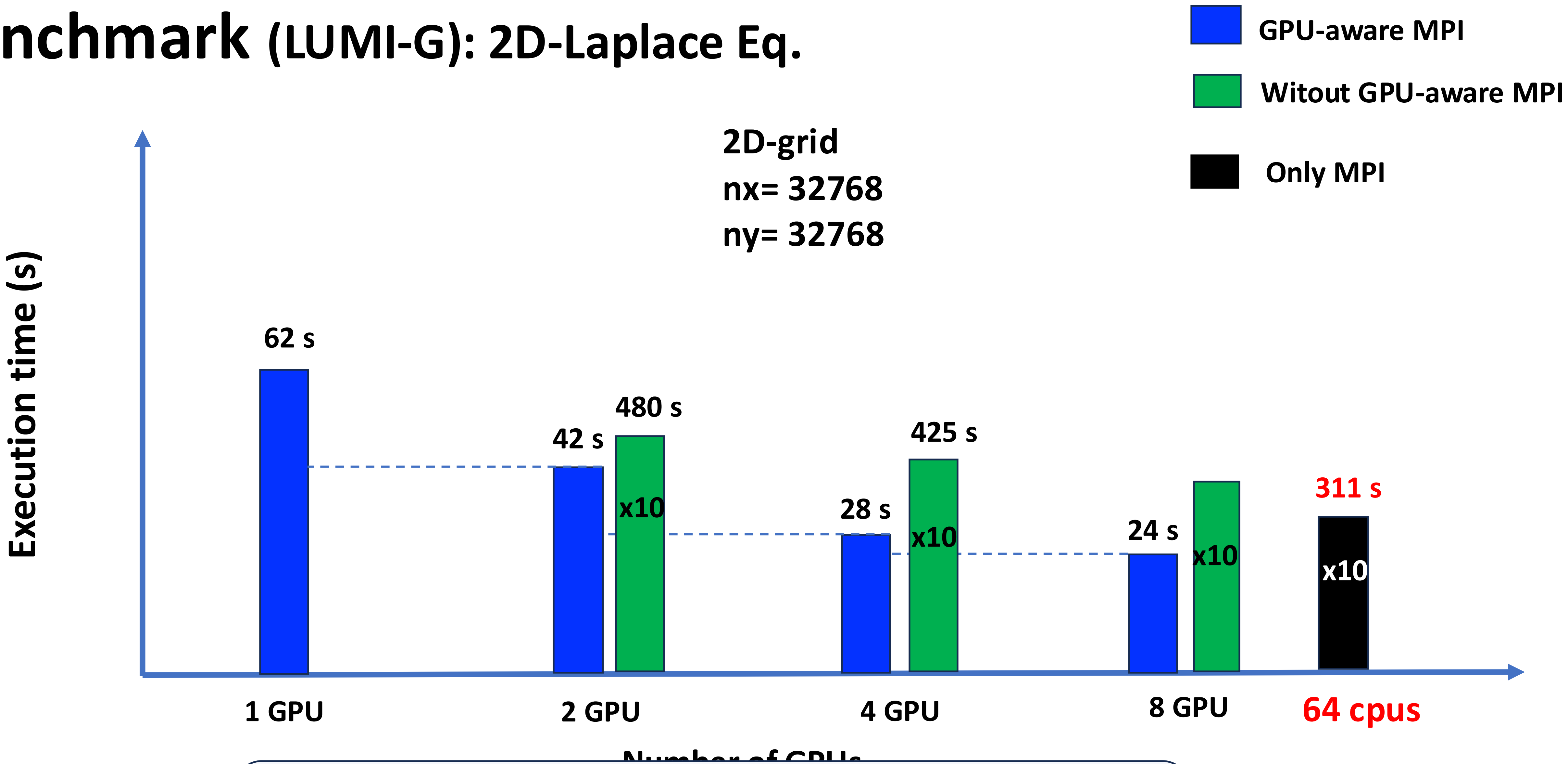
- Reduce data movement overhead.
- Easier to integrate directives (GPU-aware support) into existing MPI codes.

➡ **Significant improvement of performance**

Benchmark (LUMI-G): 2D-Laplace Eq.



Benchmark (LUMI-G): 2D-Laplace Eq.



GPU-aware MPI: speed-up by a factor of 10 (enhanced scalibility)

Hands-on Exercise

Apply GPU-aware MPI in a 2D-Laplace equation

Hands-on Exercise: Apply GPU-aware MPI in a 2D-Laplace equation

- **Download the repo**

$ git clone https://github.com/HichamAgueny/multiGPU_MPI_examples
$ cd multiGPU_MPI_examples

- **For MPI-OpenACC**

$ cd exercise/mpiacc_gpuaware

- **For MPI-OpenMP**

$ cd exercise/mpiomp_gpuaware

- **Follow the instructions described in** laplace_gpuaware_mpiacc.f90 for MPI-OpenACC or in laplace_gpuaware_mpiomp.f90 for MPI-OpenMP

- **To compile and execute the code**

Load the LUMI software stack
$ module load LUMI/24.03 partition/G

Compile: $ ./compile.sh
Submit a job: $ sbatch script.slurm