GPU Communication Libraries for Accelerating HPC and AI Applications

Benjamin Glick, Arnav Goel, Pouya Kousha, GPU Communications & Networking, NVIDIA

IEEE HOT Interconnects, 2025

- Motivation
- Introduction to NCCL
- NCCL API Walkthrough and Examples
- Introduction to NVSHMEM
- NVSHMEM API Walkthrough and Examples
- New Features For Communication Libraries & Roadmap
- Questions & Feedback

Motivation and Goals

Why GPU Communication Matters ?

- Modern AI and HPC workloads require multiple GPUs to work together efficiently
  - TOP500 graphs showing considerable share of multi-GPU
  - Fast, scalable GPU-to-GPU communication
- Technologies like NVLink, PCIe, and RDMA (InfiniBand, RoCE, etc) enable high-bandwidth, low-latency data transfer between GPUs

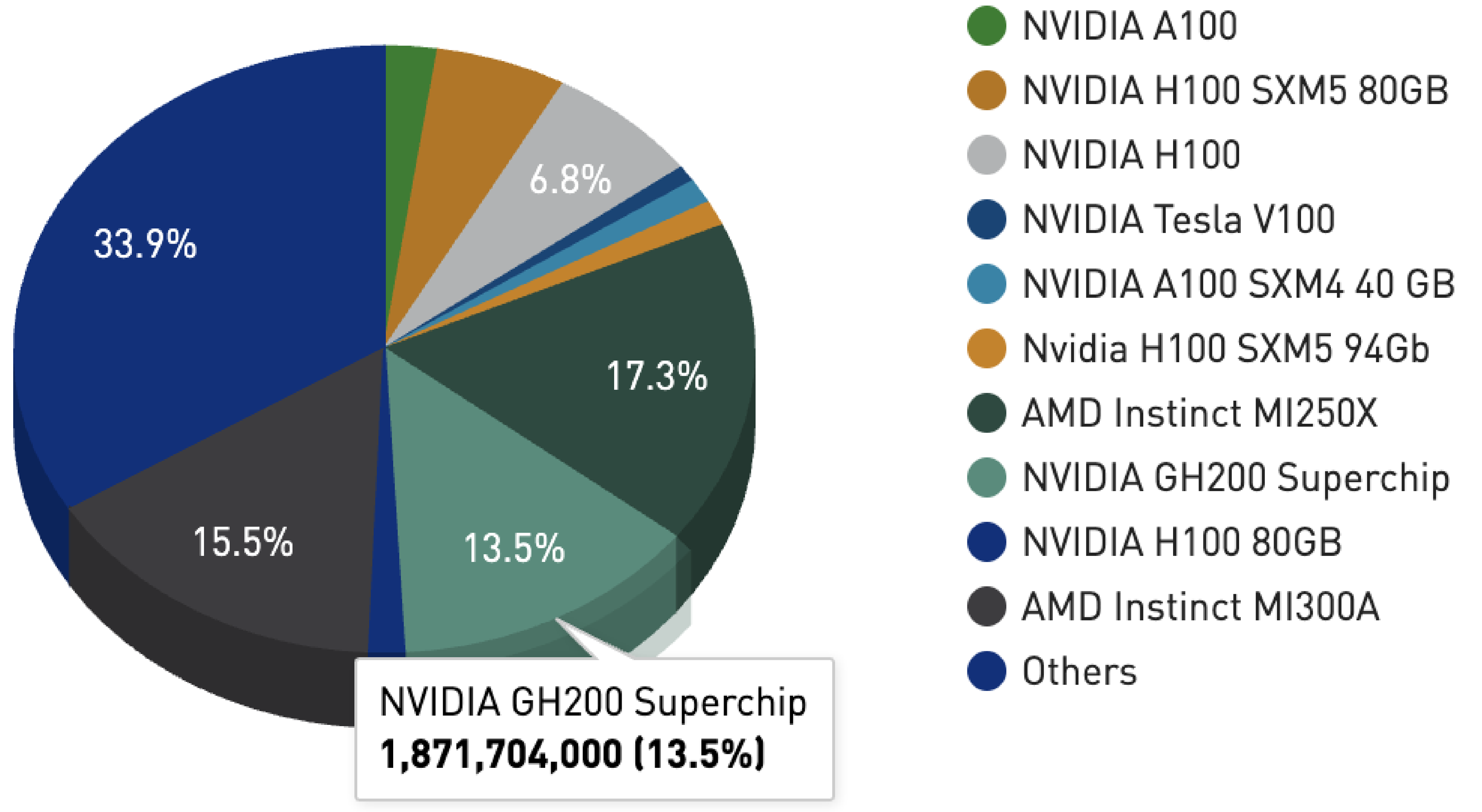
Common Use Cases

- Distributed deep learning (e.g., PyTorch, vLLM, DeepEP, TRTLLM, etc).
- Large-scale simulations and scientific computing.
- Real-time data analytics and inference pipeline

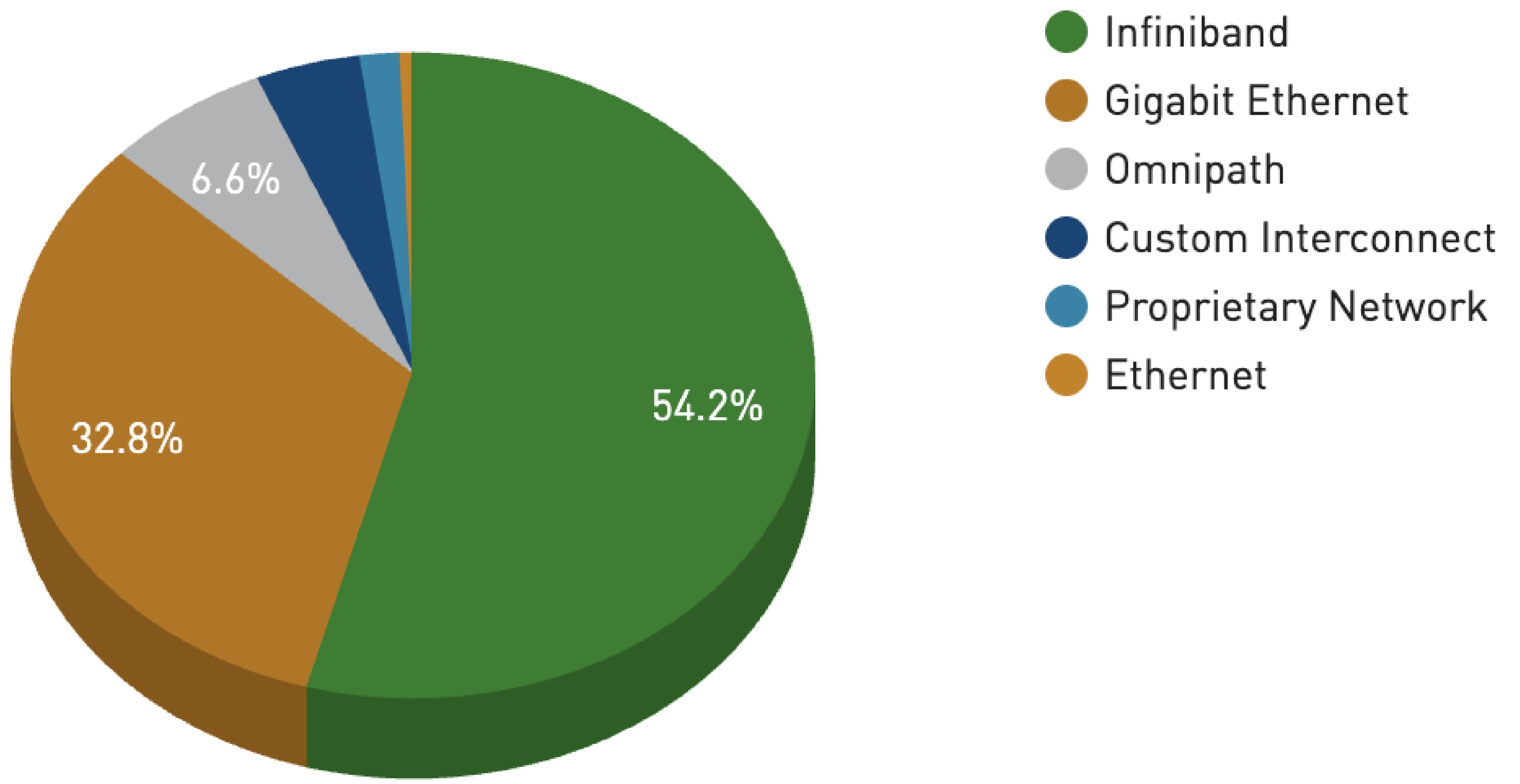
What will you learn ?

- Lean about Nvidia solutions for efficient data movement between GPUs
- **NCCL** for low-latency and high-throughput GPU-GPU communication
- **NVSHMEM** for fine-grained GPU-centric communication

Accelerator/Co-Processor Performance Share



Interconnect Family System Share



Based on slides from Lena Oden @SC21, ISC22, SC22, ISC23: https://github.com/FZJ-JSC/tutorial-multi-gpu/blob/v4.0-isc23/07-L_Overview_of_NCCL_and_NVSHMEM_in_MPI_Programs/slides.pdf

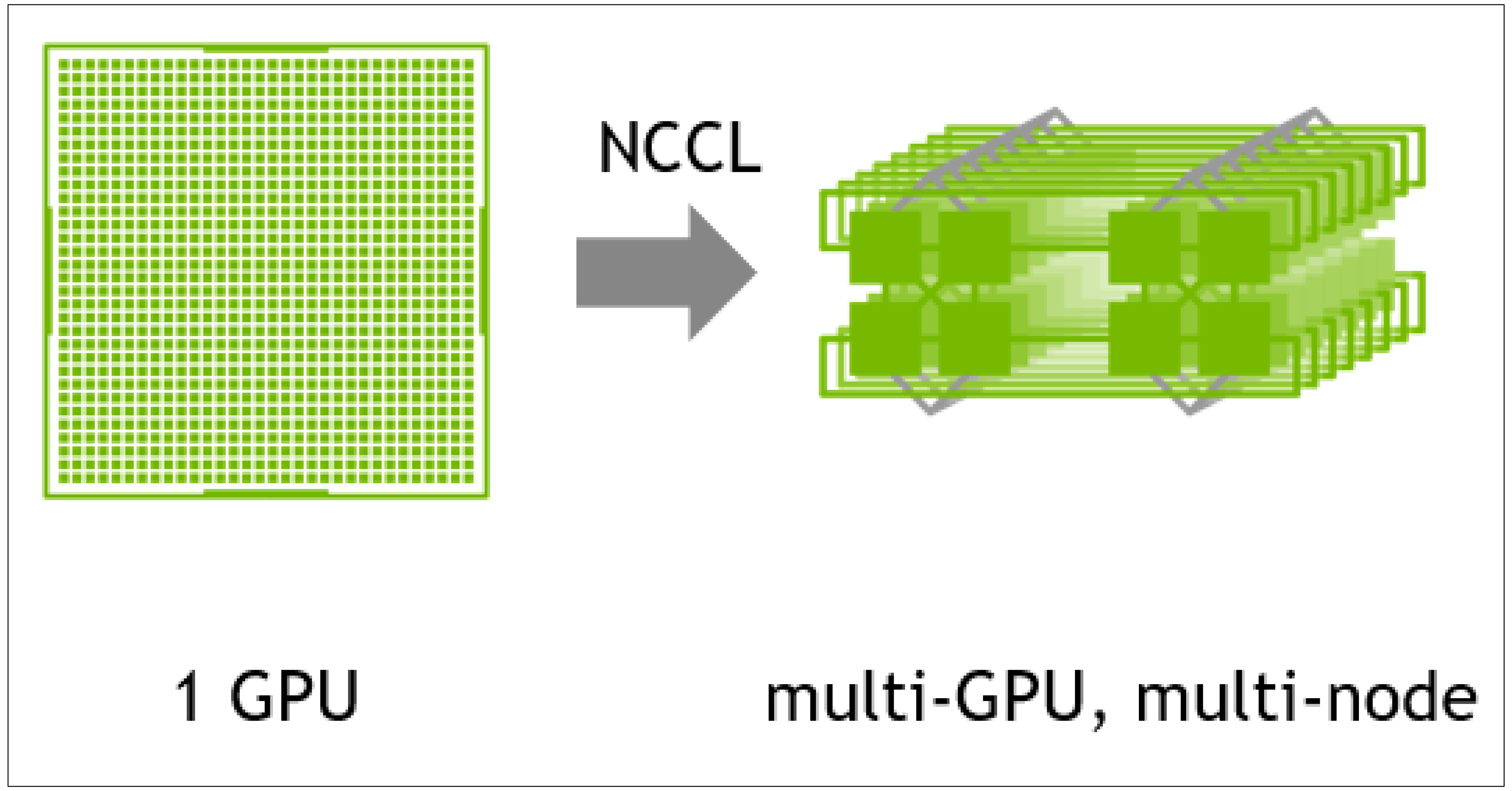
NCCL Introduction

Optimized inter-GPU communication

NCCL : NVIDIA Collective Communication Library

Communication library running on GPUs, for GPU buffers.

- NCCL (pronounced “Nickel”) is a library developed by NVIDIA for efficient communication between multiple GPUs
  - Supports single node and across multiple nodes
- P2P and Collective Operations (e.g. Allreduce, Broadcast)
- **Library running on GPU**: Communication calls are translated to a GPU kernel (running on a CUDA stream)
- Since 2.27: Low-latency symmetric kernels
  - Will be covered in advanced section



Binaries : https://developer.nvidia.com/nccl and in NGC containers
Source code : https://github.com/nvidia/nccl
Perf tests : https://github.com/nvidia/nccl-tests

NCCL Basic Example

NCCL APIs

Communication

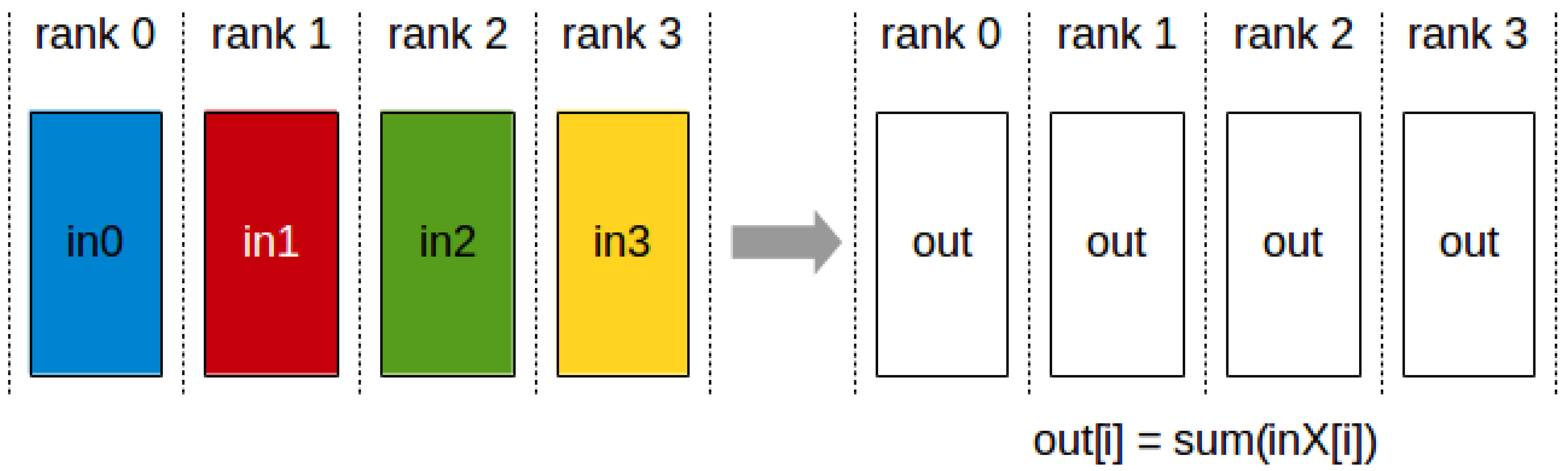
- Send/Recv

```
ncclSend(void* sbuff, size_t count, ncclDataType_t type, int peer, ncclComm_t comm, cudaStream_t stream);
ncclRecv(void* rbuff, size_t count, ncclDataType_t type, int peer, ncclComm_t comm, cudaStream_t stream);
```

- Collective Operations

```
ncclAllReduce(void* sbuff, void* rbuff, size_t count, ncclDataType_t type, ncclRedOp_t op,
          ncclComm_t comm, cudaStream_t stream);
ncclBroadcast(void* sbuff, void* rbuff, size_t count, ncclDataType_t type, int root,
          ncclComm_t comm, cudaStream_t stream);
ncclReduce(void* sbuff, void* rbuff, size_t count, ncclDataType_t type, ncclRedOp_t op, int root,
          ncclComm_t comm, cudaStream_t stream);
```

- Allreduce example



NCCL API

Initialization and Teardown

```
…
// assuming app is assigned a rank, comm_size
ncclUniqueId nccl_uid;
if (rank == 0) ncclGetUniqueId(&nccl_uid);
// nccl_uid should be distributed to all ranks (out-of-band) before
creating communicator

ncclComm_t nccl_comm;
ncclCommInitRank(&nccl_comm, comm_size, nccl_uid, rank);
…
…
ncclCommDestroy(nccl_comm);
```

Should be called once when creating a communicator

Creating a communicator group of comm_size for each rank

Create `ncclUniqueId`

Broadcast `ncclUniqueId` to all ranks

Initialize communicator per rank by `ncclCommInitRank`

Based on slides from Lena Oden @SC21, ISC22, SC22, ISC23: https://github.com/FZJ-JSC/tutorial-multi-gpu/blob/v4.0-isc23/07-L_Overview_of_NCCL_and_NVSHMEM_in_MPI_Programs/slides.pdf

NCCL API

Send/Receive with NCCL

```
…
// Initialized NCCL communicator

int N=16;
cudaStream_t stream;
cudaStreamCreate(&stream);
if (rank == 0) {
  ncclSend(send_buf, N, ncclInt, 1, nccl_comm, stream);
} else if (rank == 1) {
  ncclRecv(recv_buf, N, ncclInt, 0, nccl_comm, stream);
}

cudaStreamSynchronize(stream);

// Destroy NCCL communicator
…
```

Creating GPU stream for NCCL stream

Sending 16 * ncclInt to rank 1 as part of nccl_comm

NCCL API

Fused Communication Calls

- Multiple calls to `ncclSend()` and `ncclRecv()` should be fused with `ncclGroupStart()` and `ncclGroupEnd()` to
  - Avoid deadlocks, e.g. if calls need to progress concurrently
  - For more performance: fused operations can be more efficient by better utilizing the available IO

Send/Recv

```
ncclGroupStart();
 ncclSend(sendbuff, sendcount, sendtype, peer, comm, stream);
 ncclRecv(recvbuff, recvcount, recvtype, peer, comm, stream);
ncclGroupEnd();
```

Bcast:

```
ncclGroupStart();
if (rank == root) {
  for (int r=0; r<nranks; r++)
    ncclSend(sendbuff[r], size, type, r, comm, stream);
}
ncclRecv(recvbuff, size, type, root, comm, stream);
ncclGroupEnd();
```

NCCL Hello World – Lab 1

Compiling MPI+NCCL Applications

Include the NCCL header file and link against NCCL

```
#include <nccl.h>
```

Open nccl/lab1 -> nccl_basic.cpp

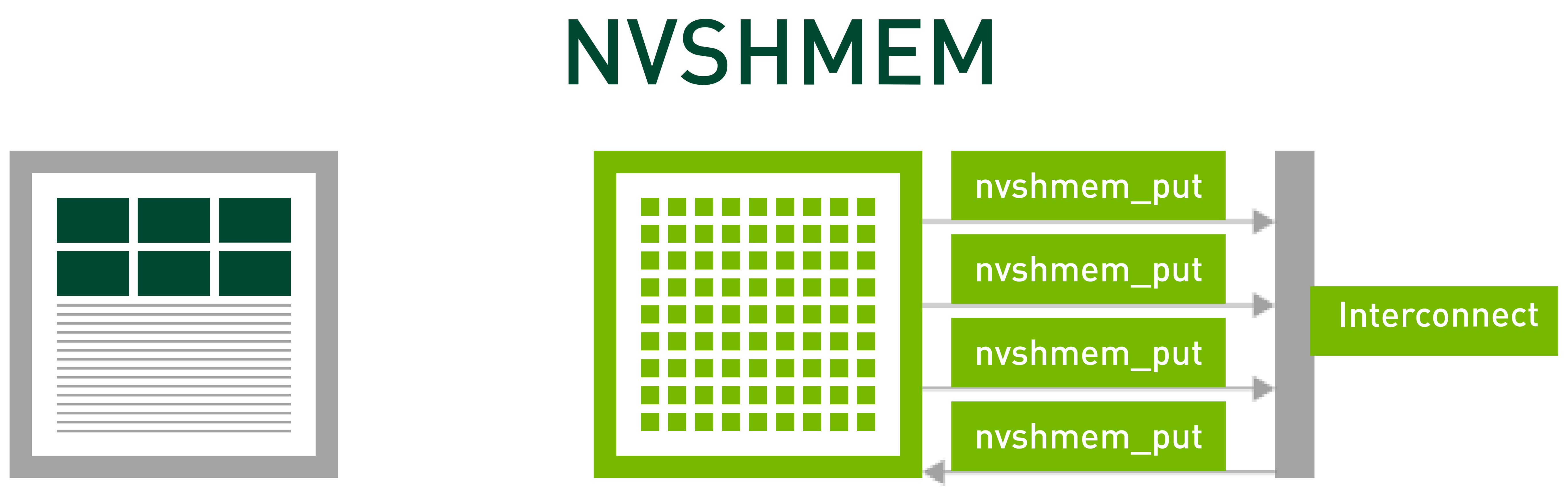
```
# Source the environment (if not previously done)
source $PROJECT_training2537/env.sh
jsc-material-sync

# Compile and link app using NCCL & MPI
make

# Run the application
make run
```

NVSHMEM Introduction

NVSHMEM

Overview

- Implements & Extends the OpenSHMEM API for clusters of NVIDIA GPUs
- Partitioned Global Address Space (PGAS) programming model
  - One sided Communication with put/get
  - Shared memory Heap
- GPU Centric communication APIs
  - GPU Initiated: thread, warp, block (narrow datatypes and tensor-operands)
  - CPU Initiated: Stream/Graph-Based (communication kernel or cudaMemcpyAsync)
- Since 3.3: First-party language bindings for Python Ecosystem
  - Will be covered in the new features section



Project Home Page: https://docs.nvidia.com/nvshmem/
Developer Forum Page: https://forums.developer.nvidia.com/tag/nvshmem

NVSHMEM

Memory Model

Symmetric objects are allocated collectively with the same size on every PE

- Symmetric memory: `nvshmem_malloc(size);`
- Private memory: `cudaMalloc(...)`

Must be the same on all PEs

NVSHMEM Basic Example

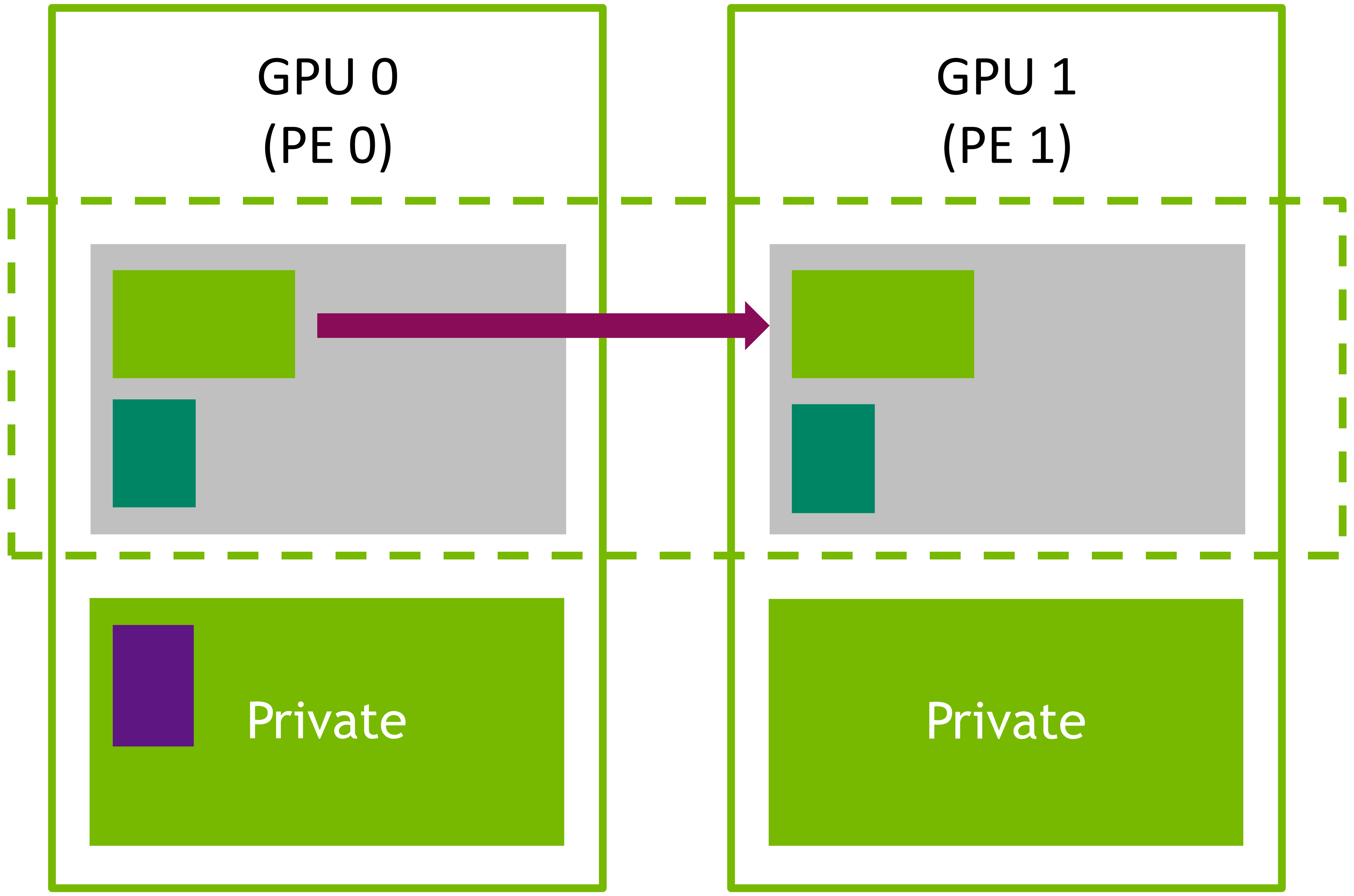
NVSHMEM API

Interoperability with MPI

```
MPI_Init(&argc, &argv);
// Assuming size, rank are populated by MPI_Comm_rank/size
MPI_Comm mpi_comm = MPI_COMM_WORLD;
nvshmemx_init_attr_t attr;
attr.mpi_comm = &mpi_comm;
nvshmemx_init_attr(NVSHMEMX_INIT_WITH_MPI_COMM, &attr);
assert( size == nvshmem_n_pes() );
assert( rank == nvshmem_my_pe() );
…
nvshmem_finalize();
MPI_Finalize();
```

NVSHMEM API

Host/Device Put



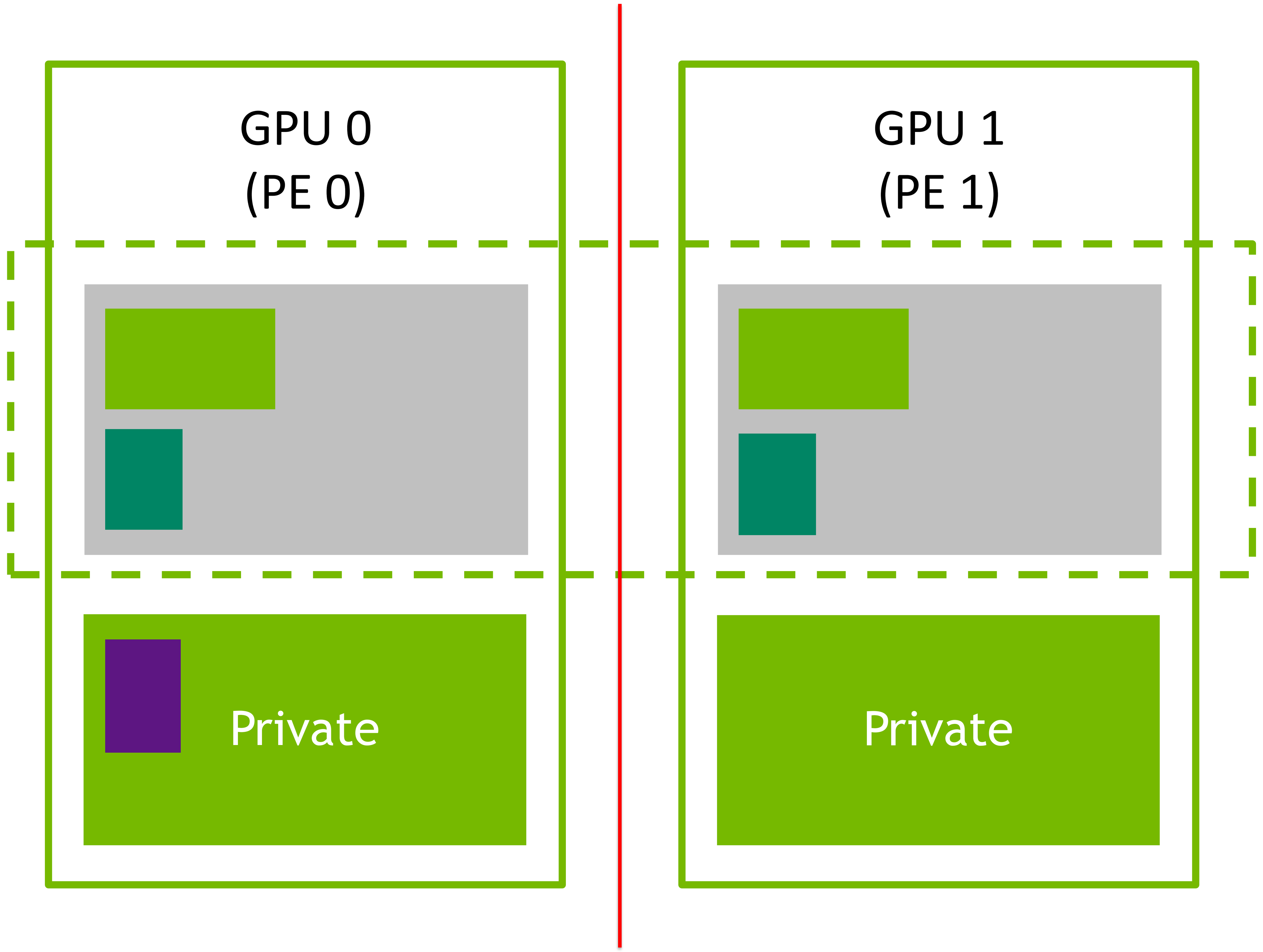
Copies `nelems` data elements of type `T` from symmetric object `src` to `dest` on PE `pe`

```
void nvshmemx_<T>_put_on_stream(T* dest, const T* src, size_t nelems, int pe, cudaStream_t stream);
// SCOPE can be thread, warp, block
__device__ void nvshmemx_<T>_put_<SCOPE>(T* dest, const T* src, size_t nelems, int pe);
```

The x marks extensions to the OpenSHMEM API

NVSHMEM API

Host/Device Barrier



Synchronizes all PEs and ensures communication performed prior to the barrier has completed

```
void nvshmemx_barrier_all_on_stream(cudaStream_t stream)
// SCOPE can be thread, warp, block
__device__ void nvshmemx_barrier_all_<SCOPE>(void);
```

NVSHMEM – Lab 2

Compiling MPI+NVSHMEM Applications

Include the NVSHMEM header files

```
#include <nvshmem.h>
#include <nvshmemx.h>
```

Compile and link against the NVSHMEM library `-lnvshmem`. In nvshmem/lab2, open nvshmem_basic.cu

```
# Source the environment (if not previously done)
source $PROJECT_training2537/env.sh
jsc-material-sync

# Compile & link application using NVSHMEM
make

# Run the application
make run
```

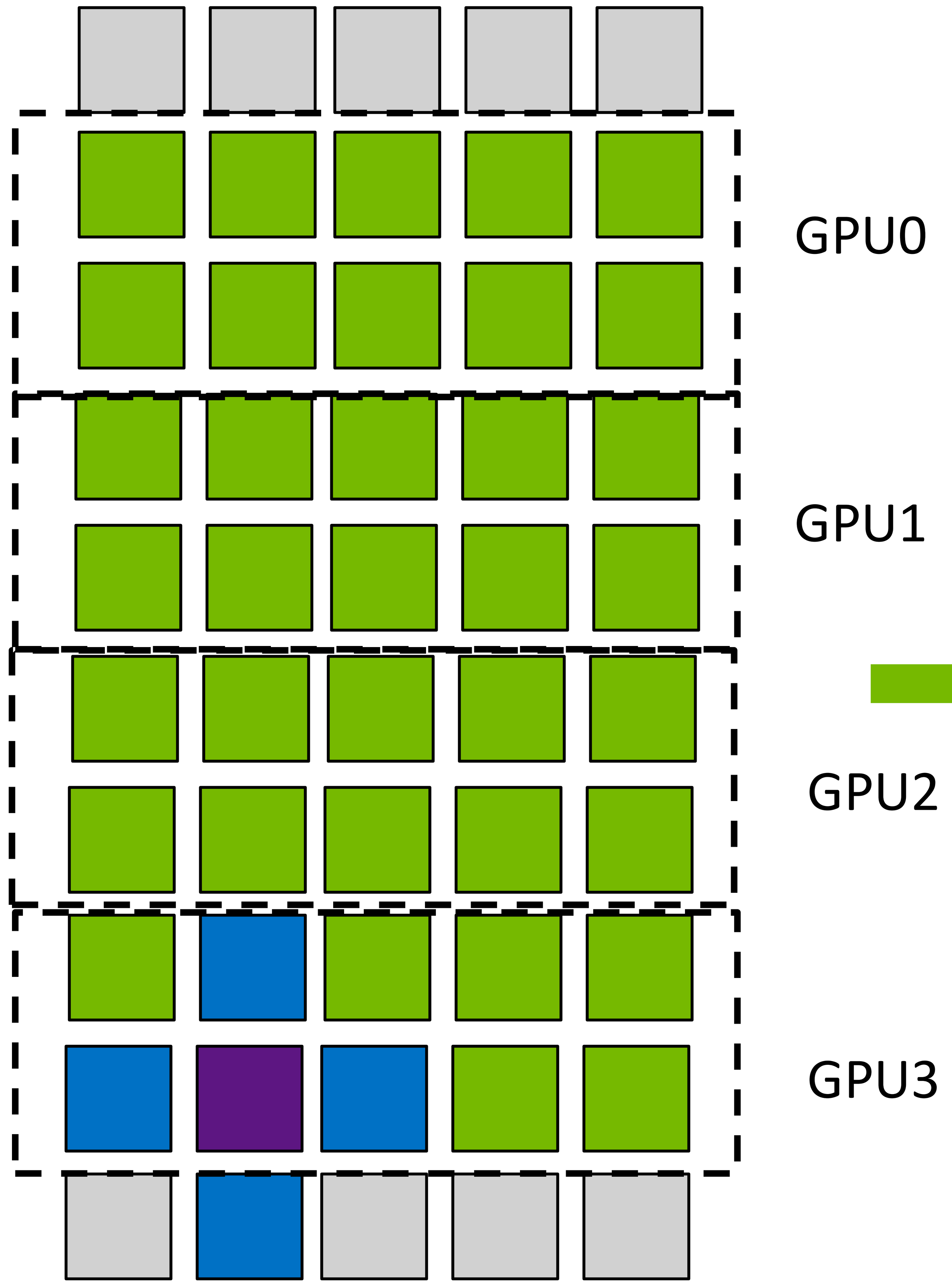
Jacobi Solver - First Look

Jacobi Solver

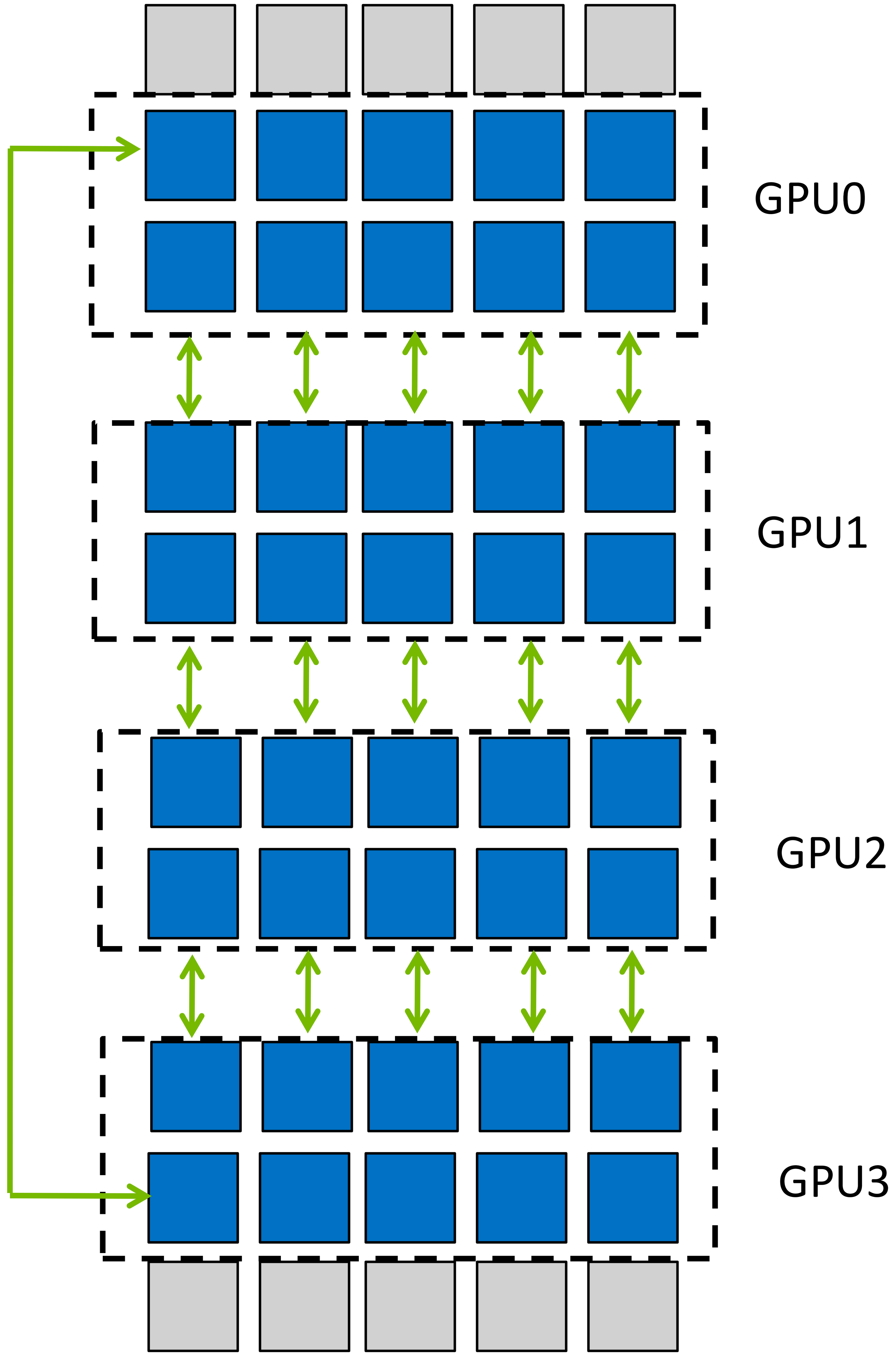
What is happening under the covers ?

- jacobi<<<grid,block, 0,stream>>>
  - // Stencil Update
  - const real new_val =
    - 0.25 * (a[iy * nx + ix + 1] +
    - a[iy * nx + ix - 1] +
    - a[(iy + 1) * nx + ix] +
    - a[(iy - 1) * nx + ix]);
  - a_new[iy * nx + ix] = new_val;

  - // Halo Exchange
    - ncclSend/ncclRecv
    - nvshmemx_float_put/p



2D 4-pt stencil based parallel decomposition



Halo Exchange between top/bottom neighbor GPUs

NCCL Advanced Example

NCCL

Overlapping Communication and Computation

- GPUs support multiple CUDA streams to run concurrently
- So far, no overlap of communication and computation
- Make sure that communication streams are scheduled
  - CUDA high priority streams!

```
int leastPriority = 0;
int greatestPriority = leastPriority;
cudaDeviceGetStreamPriorityRange(&leastPriority, &greatestPriority);

cudaStream_t compute_stream;
cudaStream_t push_stream;

cudaStreamCreateWithPriority(&compute_stream, cudaStreamDefault, leastPriority);
cudaStreamCreateWithPriority(&push_stream, cudaStreamDefault, greatestPriority);
```

Getting the range of priorities

Assigning priority

Jacobi Solver Exercise – Lab 3

What needs to be done with NCCL ?

- Use the APIs introduced on the previous slide to achieve stream-initiated communication (Jacobi)
  - **Look for `//TODO:` to get started in jacobi_unsolved.cpp**

- NCCL API

```
ncclSend(void* sbuff, size_t count, ncclDataType_t type, int peer, ncclComm_t comm, cudaStream_t stream);
ncclRecv(void* rbuff, size_t count, ncclDataType_t type, int peer, ncclComm_t comm, cudaStream_t stream);
ncclGroupStart(void);
ncclGroupEnd(void);
```

- CUDA API

```
cudaDeviceGetStreamPriorityRange(int *min, int *max);
cudaStreamCreateWithPriority(cudaStream_t *stream, int flags, int priority);
```

To compile & run: make && make run

Jacobi with NCCL

Solution: Overlapping Communication and Computation

```
launch_jacobi_kernel(a_new, a, l2_norm_d, iy_start,       (iy_start + 1), nx, push_stream);
launch_jacobi_kernel(a_new, a, l2_norm_d, (iy_end - 1),   iy_end,         nx, push_stream);
launch_jacobi_kernel(a_new, a, l2_norm_d, (iy_start + 1), (iy_end - 1),   nx, compute_stream);
```

```
ncclGroupStart();
ncclRecv(a_new,                     nx, NCCL_REAL_TYPE, top, nccl_comm, push_stream)
ncclSend(a_new + (iy_end - 1) * nx, nx, NCCL_REAL_TYPE, btm, nccl_comm, push_stream);
ncclRecv(a_new + (iy_end * nx),     nx, NCCL_REAL_TYPE, btm, nccl_comm, push_stream);
ncclSend(a_new + iy_start * nx,     nx, NCCL_REAL_TYPE, top, nccl_comm, push_stream);
ncclGroupEnd();
```

NVSHMEM Advanced Example

Jacobi Solver Exercise – Lab 4

What needs to be done with NVSHMEM ?

- Use the APIs below to implement communication in device-initiated communication (Jacobi)
  - **Look for `//TODO:` to get started in nvshmem/lab4/jacobi_UNSOLVED.cu**
- NVSHMEM APIs:
  - **One-sided communication:**
    - `__device__ void nvshmemx_float_put_nbi_block(float *dest, const float *source, size_t nelems, int pe)`
  - **Synchronization:**
    - `void nvshmemx_barrier_all_on_stream(void, cudaStream_t stream)` from host
- CUDA APIs:
  - `void cudaStreamSynchronize(cudaStream_t stream)`

Jacobi with NVSHMEM

Solution: Overlap Compute and Communication Device API

```
// Block-scoped vector put
// All threads in the block arrive at these calls together
if ((block_iy <= iy_start) && (iy_start < block_iy + blockDim.y)) {
    nvshmemx_float_put_nbi_block(a_new + top_iy * nx + block_ix, a_new + iy_start * nx + block_ix,
                                 min(blockDim.x, nx - 1 - block_ix), top_pe);
}
if ((block_iy < iy_end) && (iy_end <= block_iy + blockDim.y)) {
    nvshmemx_float_put_nbi_block(a_new + bottom_iy * nx + block_ix, a_new + (iy_end - 1) * nx +
block_ix,
                                 min(blockDim.x, nx - 1 - block_ix), bottom_pe);
}
// Synchronize the data movement + compute kernel with a barrier across all PEs on the same stream.
nvshmemx_barrier_all_on_stream(compute_stream);
```

Non-blocking vector operations running on <BLOCK> scope

New & Upcoming Features

New Features for Comms Libraries

NCCL 2.27.6 (May 2025)

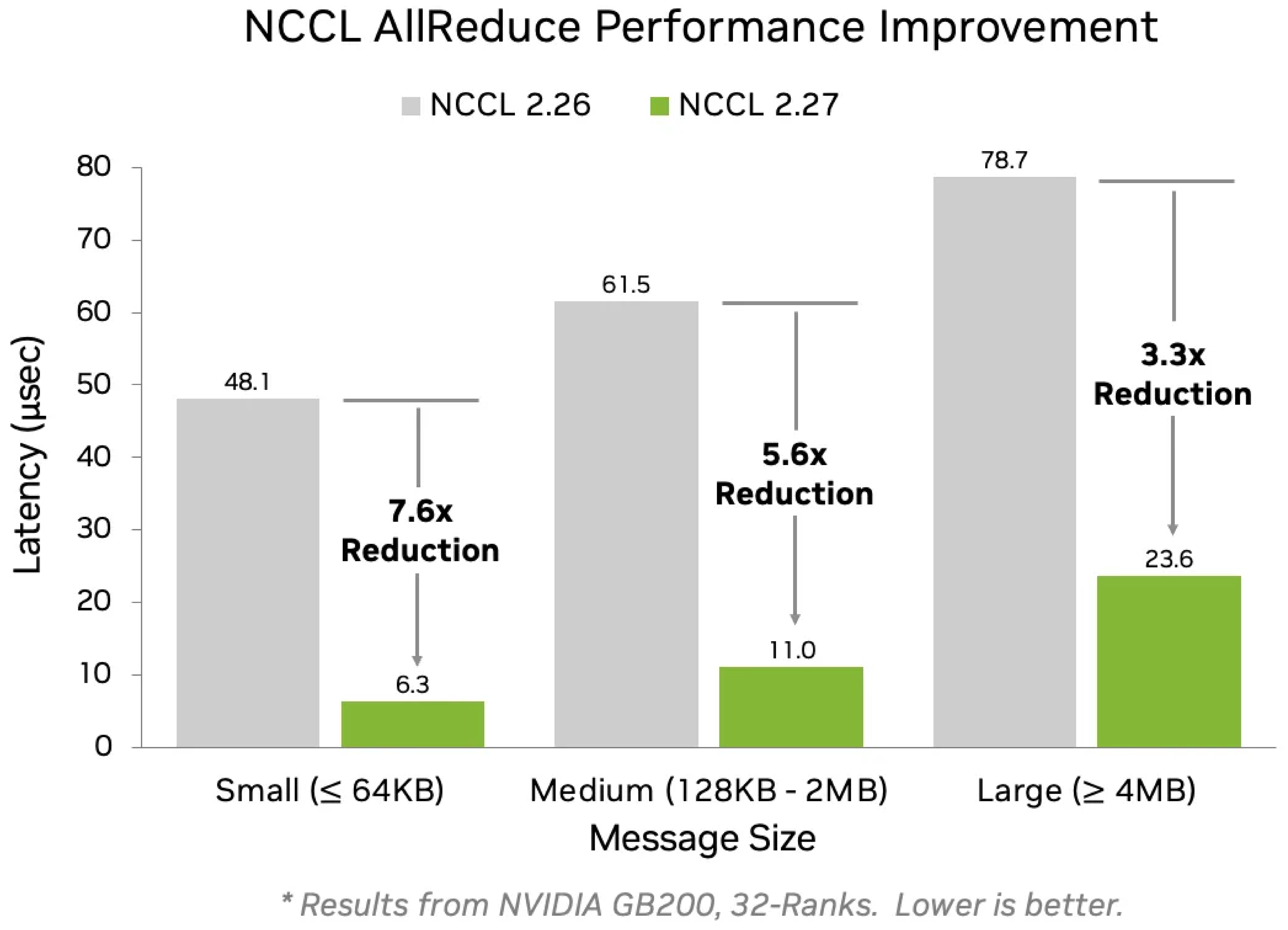
Symmetric Memory is a foundational capability in NCCL 2.27 that enables high-performance, low-latency collective operations. When memory buffers are allocated at identical virtual addresses across all ranks, NCCL can execute optimized kernels that reduce synchronization overhead and improve bandwidth efficiency. For more details, refer to blog.

```
// Assuming comm (ncclComm_t) and push_stream (cudaStream_t) is created a priori
...
// Allocate a user buffer in GPU device memory using CUDA VMM APIs
void *buf = ncclMemAlloc(size);

// Register the user buffer to a memory window
ncclCommWindowRegister(comm, buf, size, &win, NCCL_WIN_COLL_SYMMETRIC);
```

```
ncclAllReduce(buf, buf, count, datatype, op, comm, push_stream);
...
// Deregister the memory window
ncclCommWindowDeregister(comm, win);
...
```

NCCL Symmetric Memory Benefits

Low-latency kernels with symmetric memory



AllReduce latency improvements using low-latency kernels in NCCL 2.27

Based on slides from Lena Oden @SC21, ISC22, SC22, ISC23: https://github.com/FZJ-JSC/tutorial-multi-gpu/blob/v4.0-isc23/07-L_Overview_of_NCCL_and_NVSHMEM_in_MPI_Programs/slides.pdf

NCCL Symmetric Memory Exercise – Lab 5

What needs to be done with NCCL ?

- Use the APIs introduced on the previous slide to enable symmetric memory registration in example.
  - **Look for `//TODO:` to get started**

- NCCL Registration APIs
  - ncclResult_t **ncclCommWindowRegister**(ncclComm_t comm, void* buff, size_t size, ncclWindow_t* win, int winFlags)
  - ncclResult_t **ncclCommWindowDeregister**(ncclComm_t comm, ncclWindow_t win);
- NCCL Collective APIs
  - ncclResult_t **ncclAllGather**(const void* sendbuff, void* recvbuff, size_t sendcount, ncclDataType_t datatype, ncclComm_t comm, cudaStream_t stream)

New Features for Comms Libraries

NVSHMEM 3.3.9 (July 2025)

NVSHMEM4Py is the official Python language binding for NVSHMEM, providing a Pythonic interface to the NVSHMEM library. It enables Python applications to leverage the high-performance, PGAS (Partitioned Global Address Space) programming model offered by NVSHMEM for GPU-centric communication. For more details, refer to API documentation.

```
import nvshmem.core as nvshmem
# Initialize NVSHMEM runtime (assuming device, stream are created and assigned per PE)
nvshmem.init(device=dev, mpi_comm=MPI.COMM_WORLD, initializer_method="mpi")
```

```
# Allocate & initialize symmetric Pytorch Tensor of size FP32 x elems bytes
tensor = nvshmem.tensor((elems,), dtype=torch.float32)
```

```
# Run the allreduce SUM operation
nvshmem.reduce(Teams.TEAM_WORLD, tensor, tensor, "sum", stream=stream);
...
# Finalize the NVSHMEM runtime
nvshmem.finalize()
...
```

Based on slides from Lena Oden @SC21, ISC22, SC22, ISC23: https://github.com/FZJ-JSC/tutorial-multi-gpu/blob/v4.0-isc23/07-L_Overview_of_NCCL_and_NVSHMEM_in_MPI_Programs/slides.pdf

NVIDIA

NVSHMEM Python Library Bindings Exercise – Lab 6

What needs to be done with NVSHMEM4Py ?

- Use the APIs introduced on the previous slide to optimize P2P communication example.
  - **Look for `//TODO:` to get started**

- NVSHMEM one-sided communication, signal and wait API
  - **nvshmem.core.put**(dst: object, src: object, remote_pe: int = -1, stream: cuda.core.Stream = None)
  - **nvshmem.core.barrier**(team: Teams, stream: cuda.core.Stream = None) -> None:
  - **nvshmem.core.put_signal**(dst: object, src: object, signal_var: cuda.core.Buffer, signal_val: int, signal_op: nvshmem.bindings.nvshmem.Signal_op, remote_pe: int = -1, stream=None) → None
  - **nvshmem.core.signal_wait**(signal_var: cuda.core.Buffer, signal_val: int, signal_op: nvshmem.bindings.nvshmem.Signal_op, stream: cuda.core.Stream = None) → None

NCCL and NVSHMEM enable GPU-centric communication

Takeways

NCCL	NVSHMEM
Operates on two-sided semantics	Operates on one-sided semantics
Flexible communicators and ranks	Single global runtime with teams and symmetric memory allocation and registration
On-stream collectives	On-stream and device-initiated collectives
Point to point communication via on-stream send/receive API	Point to point communication via on-stream and device put/get API

Both

- Low-latency symmetric kernels
- Interoperate with MPI and other well-supported communication libraries
- Make use of high-performance communication technologies like GDRDMA, NVLink, SHARP
- Enable CUDA stream-aware, asynchronous GPU-to-GPU bulk and fine-grained communication

Based on slides from Lena Oden @SC21, ISC22, SC22, ISC23: https://github.com/FZJ-JSC/tutorial-multi-gpu/blob/v4.0-isc23/07-L_Overview_of_NCCL_and_NVSHMEM_in_MPI_Programs/slides.pdf

NCCL Roadmap

NCCL v2.27 May '25	*Github Preview – Live* NCCL v2.28 Sept '25	NCCL v2.29 Q4'25
Low latency kernel and algos	**CE Collectives**	MNNVL CE Collectives
Symmetric Memory	**Device API Support**	**Python Host API support (NCCL4Py)**
NCCL Communicator Shrink (for Fault Tolerance)	MNNVL Symmetric memory support	NCCL Put/Get Host API
NVL SHARP with IB SHARP and UB registration	Extend PAT Support	**NCCL Communicator Grow (for Fault Tolerance)**
Profiler Enhancements	New APIs for A2A, Gather, Scatter	New API for A2Av
Improved Cost Model & Tuning	Performance tuning improvements	**More latency optimizations**
User-buffer Optimization	NCCL inspector support	MIG support
Direct NIC GB300 / CX-8 Enablement	CMake support	
DGX Spark Enablement	Multiple ranks per GPU	
Cross-DC Communication Support		

Subject to Change

Prior Release Notes Available on docs.nvidia.com

NVSHMEM Roadmap

NVSHMEM 3.3
NVSHMEM4Py 0.1
(July 2025)

NVSHMEM 3.4 (Sept 2025)

- **GB200/300 + CX8 Direct NIC support**
- Upstream EEP support
- CPU-assisted IBGDA v2

NVSHMEM 3.5 (Oct 2025)

- Tile-granular put/get device APIs
- **IBGDA based QP-selection device API**
- **CE based stream-initiated collectives**

OSS Contributions (July-Sept 2025)

- **Simplify DeepEP IBGDA PR#295**
- Port FlashInfer custom bindings to NVSHMEM4Py PR#1263
- Port PPLX custom bindings to NVSHMEM4py PR#33

NVSHMEM4py 0.2 (Oct 2025)

- **Numba-CUDA DSL based device APIs**
- User Buffer Registration
- Flexible Team Management
- Torch CUDA Interoperability
- NVLS Array Management

Coming to GitHub Soon!

*Subject to Change

References

Blogs, Documentation, Guides

- NVSHMEM Docs
  - API Documentation
  - Quickstart
  - Enhancing Application Portability and Compatibility across New Platforms Using NVIDIA Magnum IO NVSHMEM 3.0
  - Improving Network Performance of HPC Systems Using NVIDIA Magnum IO NVSHMEM and GPUDirect Async
- NCCL Docs
  - API Documentation
  - New Scaling Algorithm and Initialization with NVIDIA Collective Communications Library 2.23
  - Understanding NCCL Tuning to Accelerate GPU-to-GPU Communication
  - Enabling Fast Inference and Resilient Training with NCCL 2.27
  - NCCL Deep Dive: Cross Data Center Communication and Network Topology Awareness
- GPU Mode Talk on Youtube: https://www.youtube.com/live/2xMzQ1Z2Qe0?feature=shared

NCCL Github

NVSHMEM Dev forum

Based on slides from Lena Oden @SC21, ISC22, SC22, ISC23: https://github.com/FZJ-JSC/tutorial-multi-gpu/blob/v4.0-isc23/07-L_Overview_of_NCCL_and_NVSHMEM_in_MPI_Programs/slides.pdf