NVSHMEM: CUDA-INTEGRATED COMMUNICATION FOR NVIDIA GPUS

JIM DINAN

CPU-INITIATED COMMUICATION

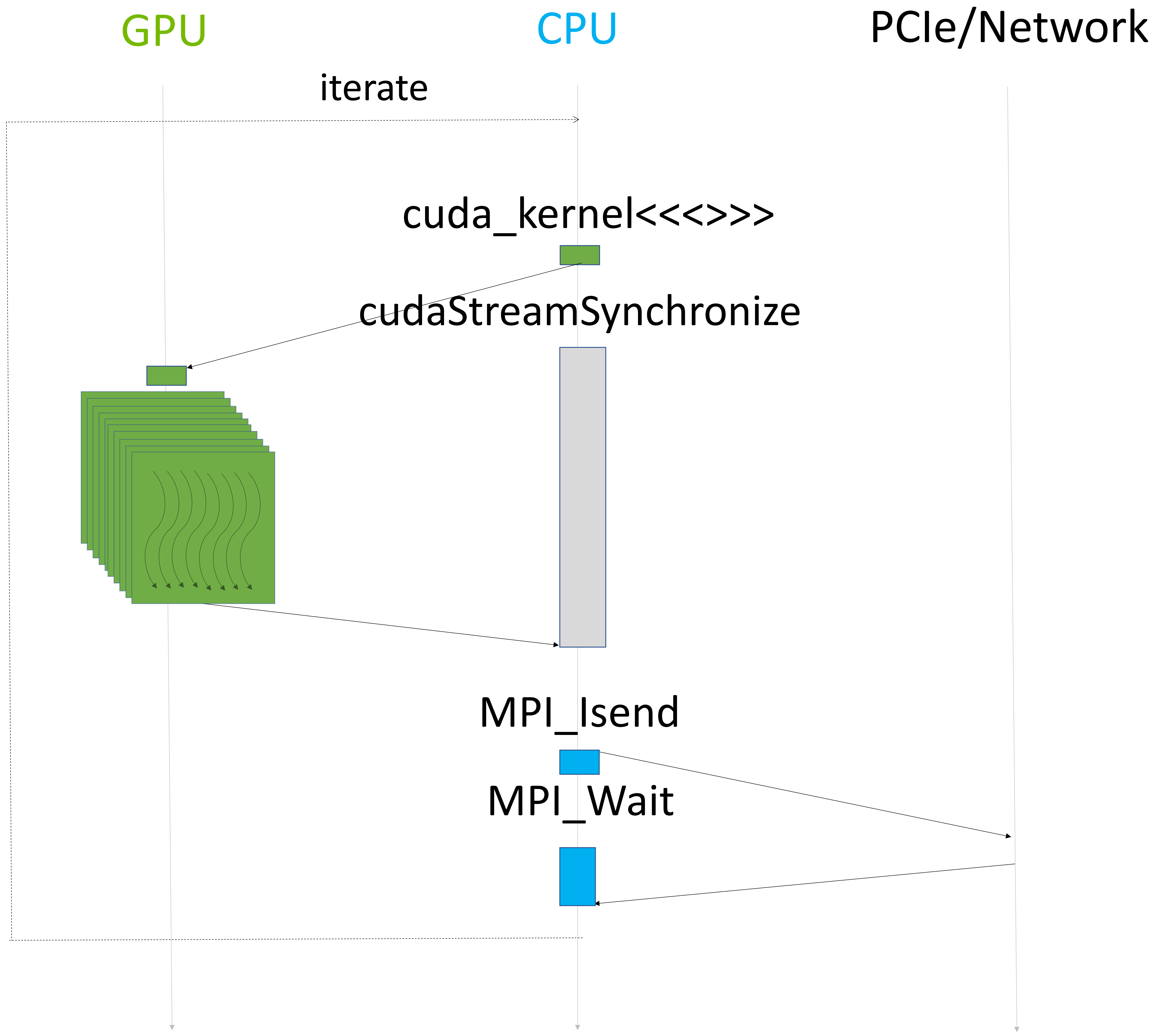
- Compute on GPU
- Communication from CPU

Synchronization at boundaries

Commonly used model, but –

- ❑ Offload latencies in critical path
- ❑ Communication is not overlapped

Hiding increases code complexity,
Not hiding limits strong scaling



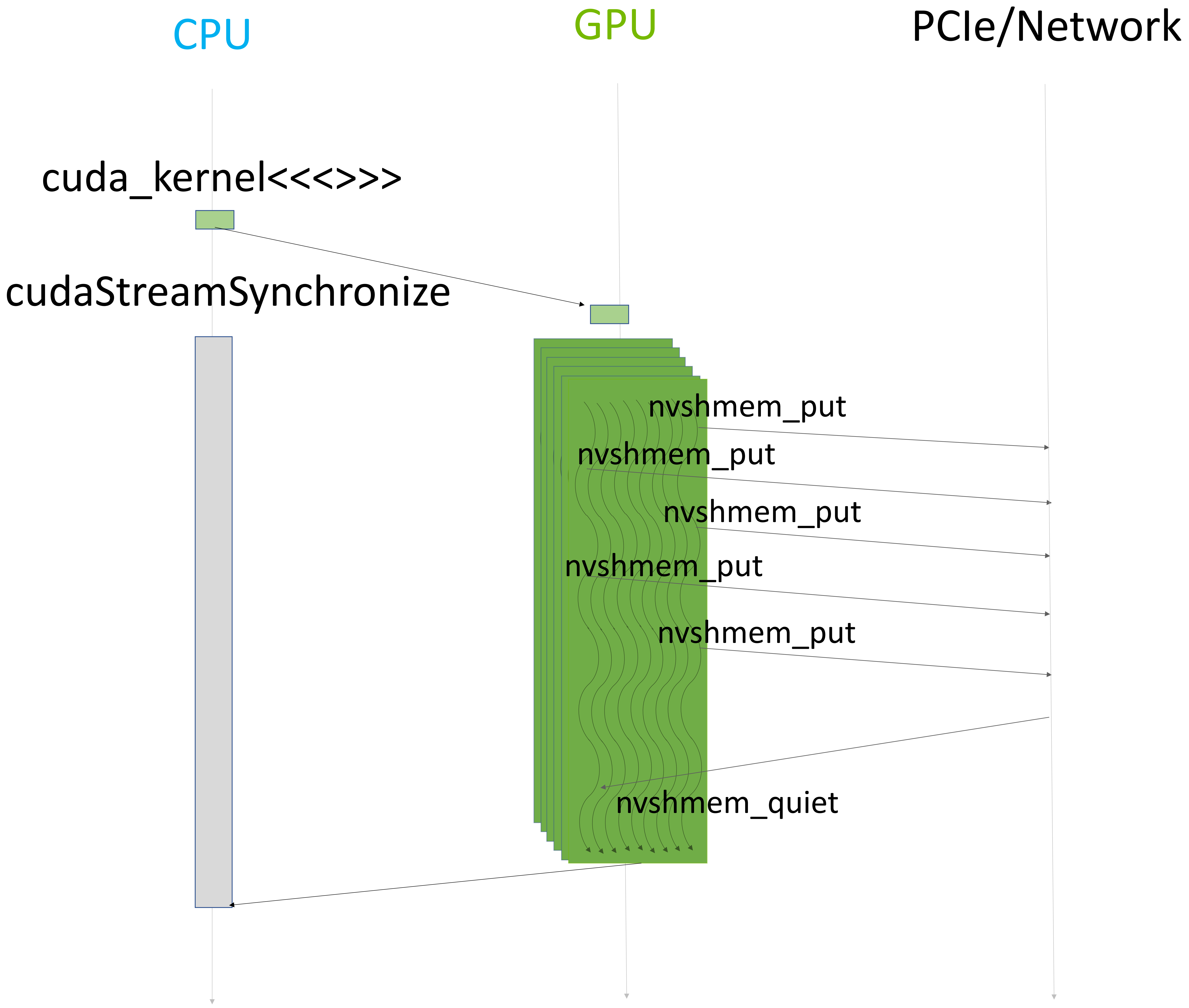
GPU-INITIATED COMMUNICATION

- Compute on GPU
- Communication from GPU

Benefits -

- ❑ Eliminates offload latencies
- ❑ Compute and communication overlap
- ❑ Latencies hidden by threading
- ❑ Easier to express algorithms with inline communication

Improving performance while making it easier to program



Overview of NVSHMEM

NVSHMEM Latest Features

Performance Case Studies

Upcoming Features and Conclusion

NVSHMEM

OpenSHMEM, Adapted for Best Performance on NVIDIA GPU Clusters

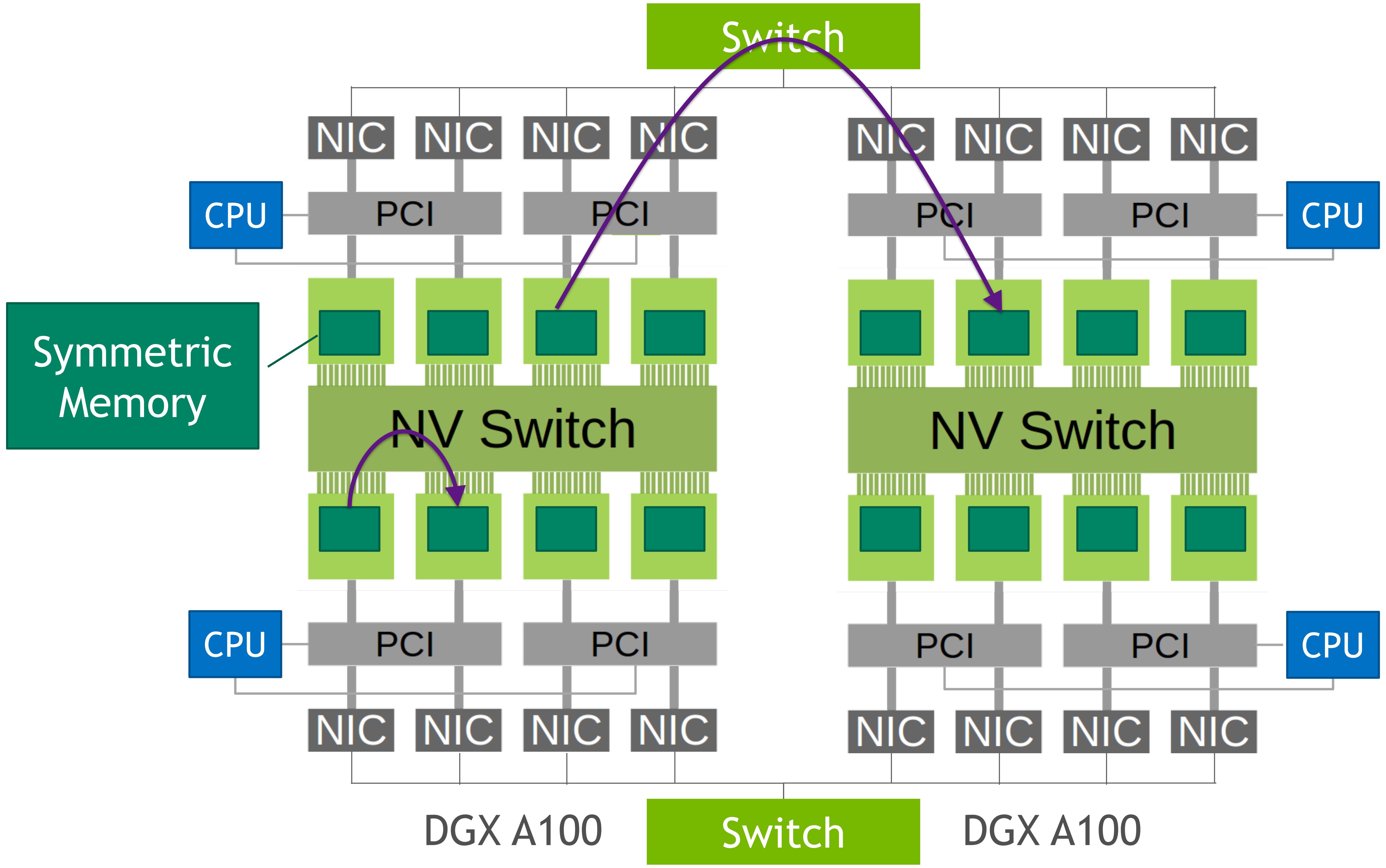
Aggregate the memory of multiple GPUs in a cluster into a distributed global address space

- Data access via put, get, atomic APIs
- Collective communication APIs

Communication integrated with CUDA execution model

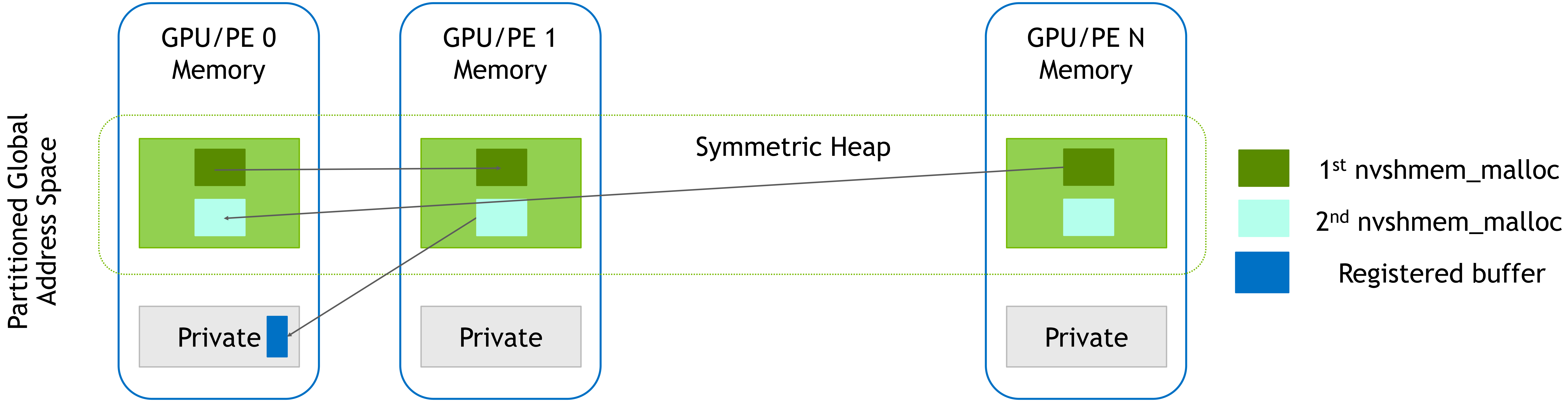
1. GPU kernel-initiated operations
2. Operations on CUDA streams/graphs
3. CPU initiated operations

Can be used together with a CPU OpenSHMEM or MPI library for host memory communication



NVSHMEM SYMMETRIC MEMORY MODEL

Individual Partitions are Aggregated into a Global Address Space



Symmetric objects are allocated collectively with the same size on every PE
Symmetric memory: nvshmem_malloc(…); **Private memory**: cudaMalloc(…)

Read: nvshmem_get(…); **Write**: nvshmem_put(…); **Atomic**: nvshmem_atomic_add(…)
Flush writes: nvshmem_quiet(); Order writes: nvshmem_fence()

Synchronize: nvshmem_barrier(); **Poll**: nvshmem_wait_until(…)

OPTIMIZED FOR NVLINK COMMUNICATION

Seamlessly Scale-up to 256 GPUs Using 4th Generation NVLink

NVSHMEM seamlessly:

- Scales-up using NVLink and PCIe
- Scales-out using InfiniBand, RoCE, …

Internally uses CUDA IPC and cuMem APIs to map symmetric memory of peer PEs into virtual address space:

❑ nvshmem_put/get on device

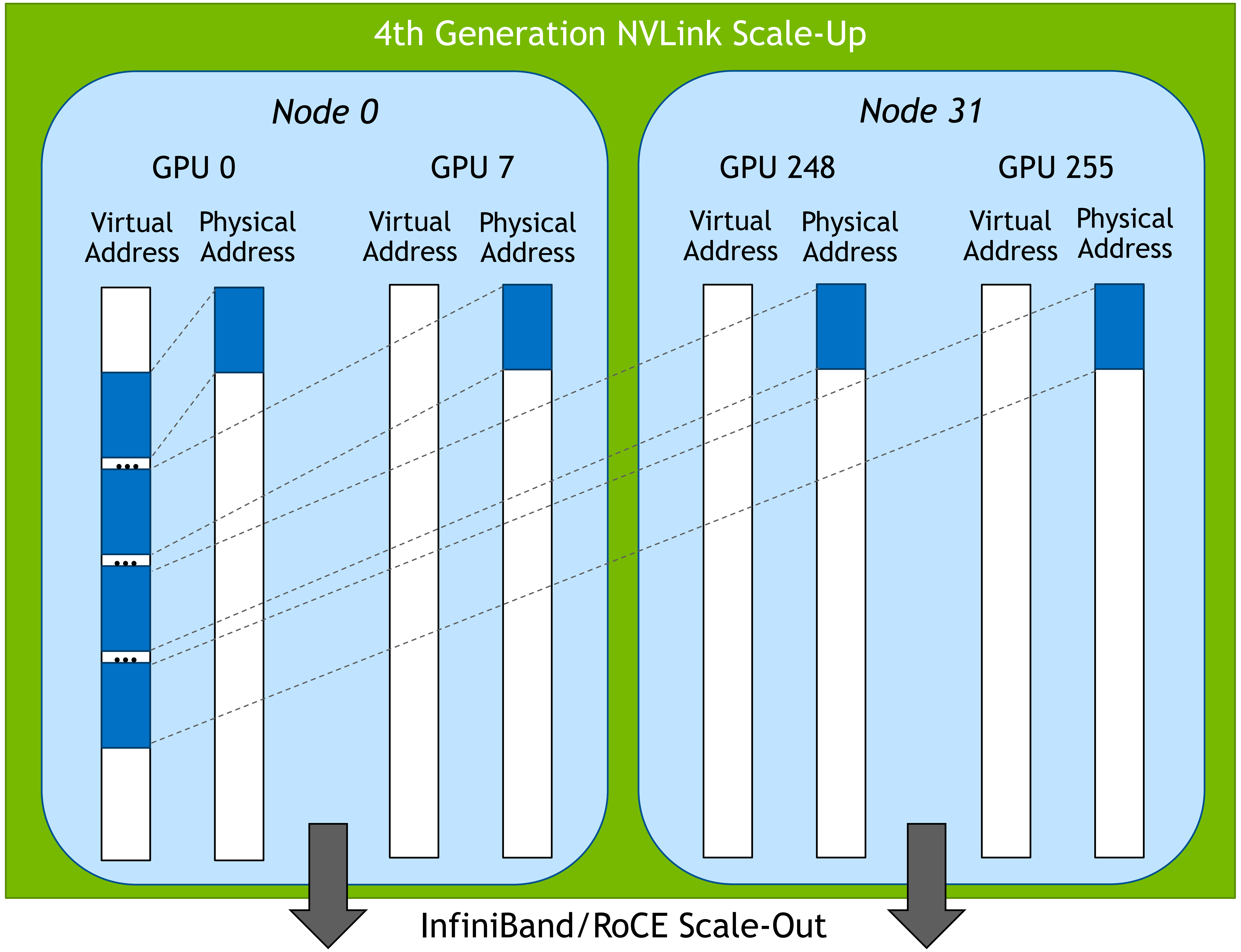
→ load/store

❑ nvshmem_put/get_on_stream

→ cudaMemcpyAsync

❑ nvshmem_ptr direct pointer bypass

→ kernel direct load/store



SCALE-UP AND SCALE-OUT BANDWIDTH

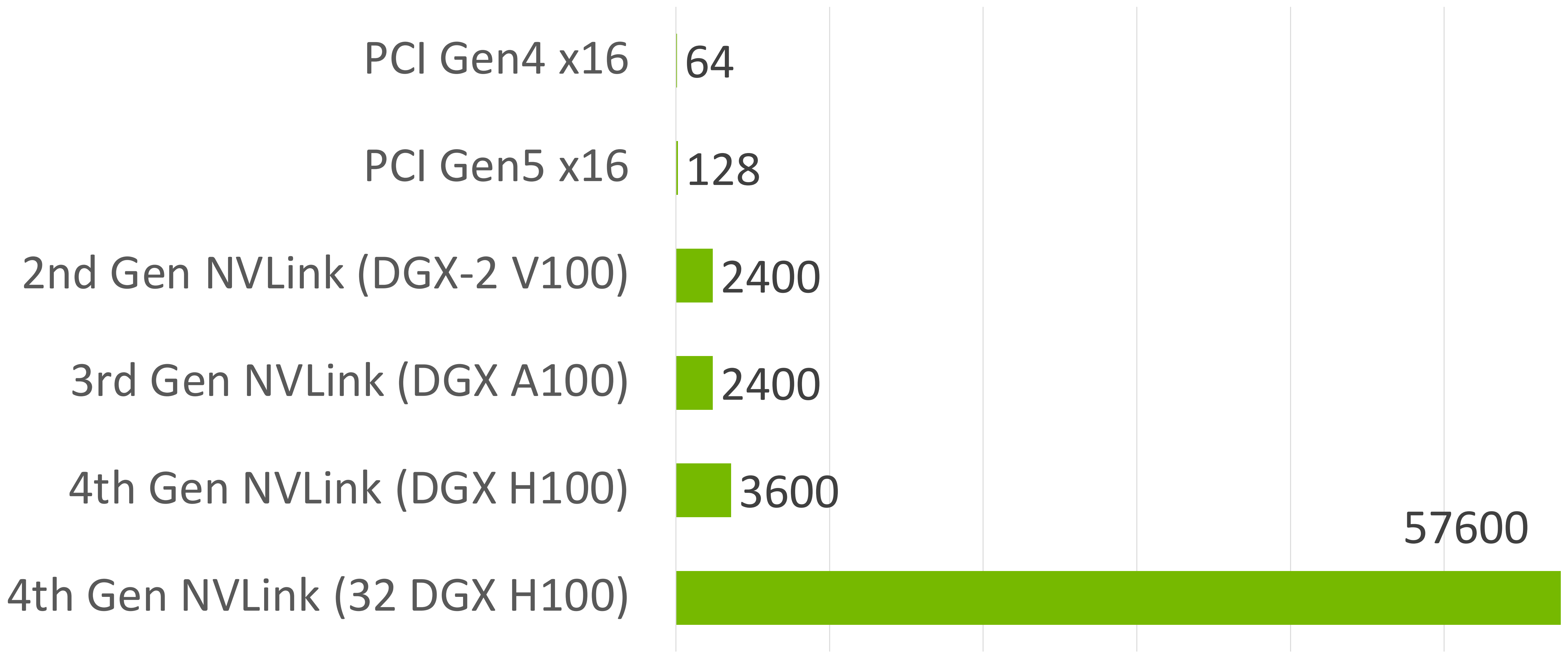
Multi-GPU

PCIe
NVLink

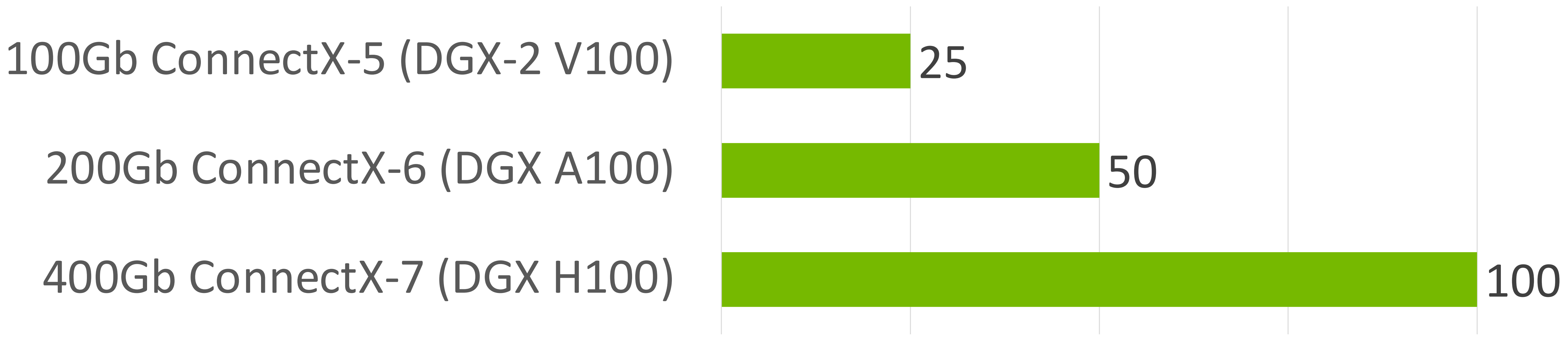
Multi-node

InfiniBand
RoCE

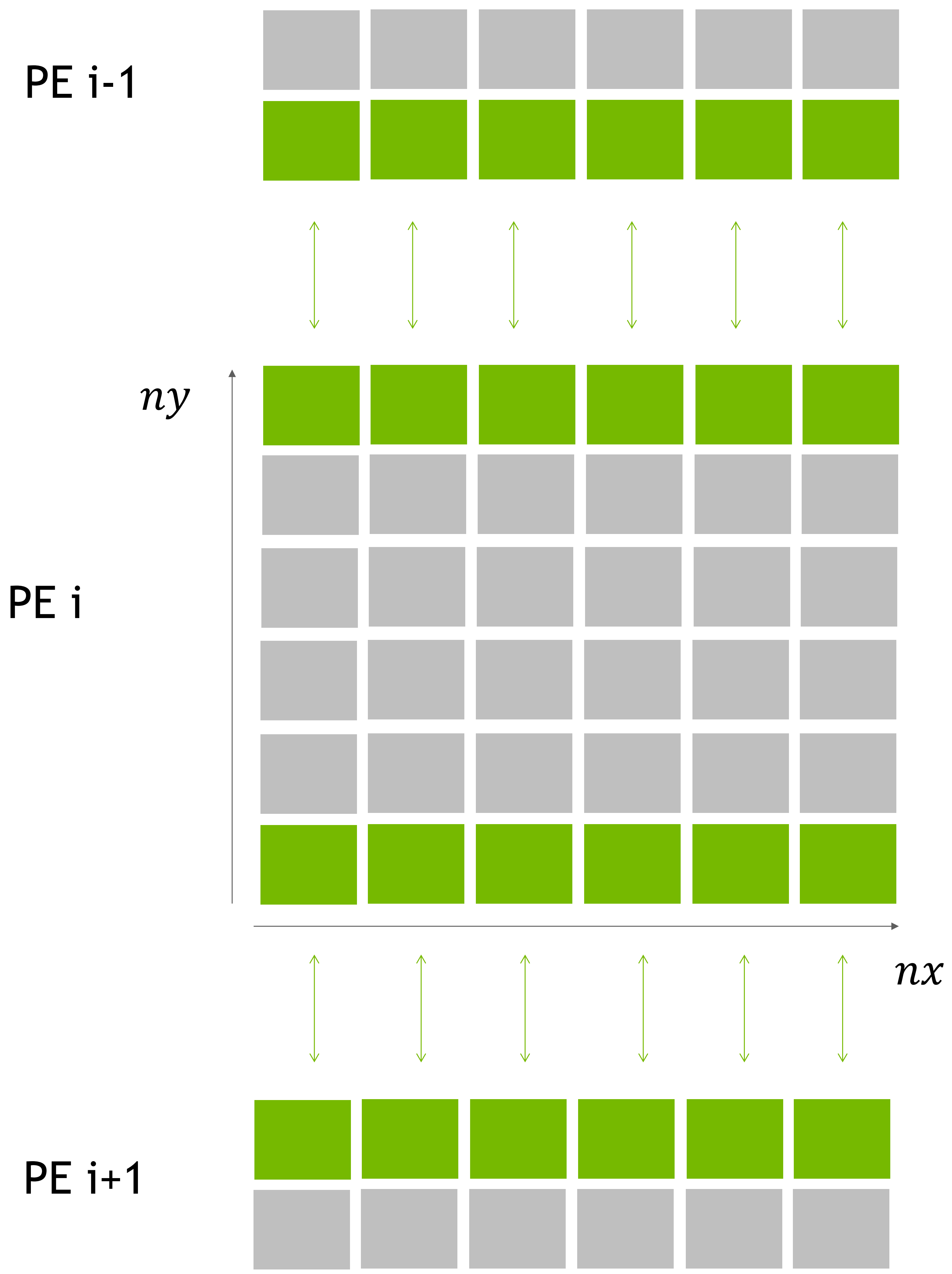
Theoretical Peer-to-Peer
Bisection Bandwidth (GB/s)



Theoretical Network Injection +
Ejection Bandwidth Per GPU (GB/s)



THREAD-LEVEL COMMUNICATION

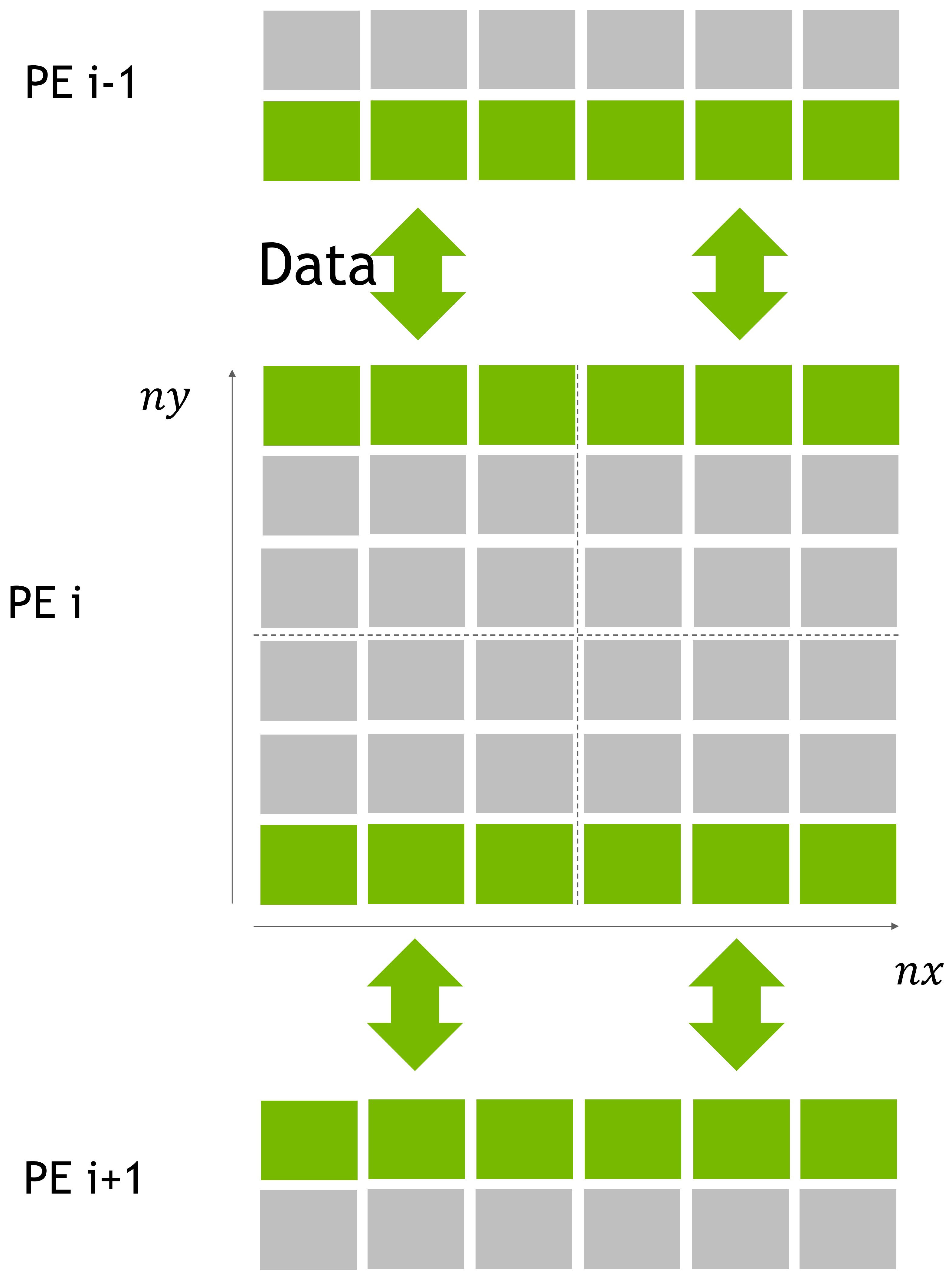


- ❑ Allows fine grained communication and overlap
- ❑ Efficient mapping to NVLink network on DGX systems

```
__global__ void stencil_single_step(float *u, float *v, …) {
   int ix = get_ix(blockIdx, blockDim, threadIdx);
   int iy = get_iy(blockIdx, blockDim, threadIdx);

   compute(u, v, ix, iy);
   // Thread-level data communication API
   if (iy == 1)
      nvshmem_float_p(u+(ny+1)*nx+ix, u[nx+ix], top_pe);
   if (iy == ny)
      nvshmem_float_p(u+ix, u[ny*nx+ix], bottom_pe);
}
```

```
for (int iter = 0; iter < N; iter++) {
   swap(u, v);
   stencil_single_step<<<..., stream>>>(u, v, …);
   nvshmem_barrier_all_on_stream(stream);
}
```

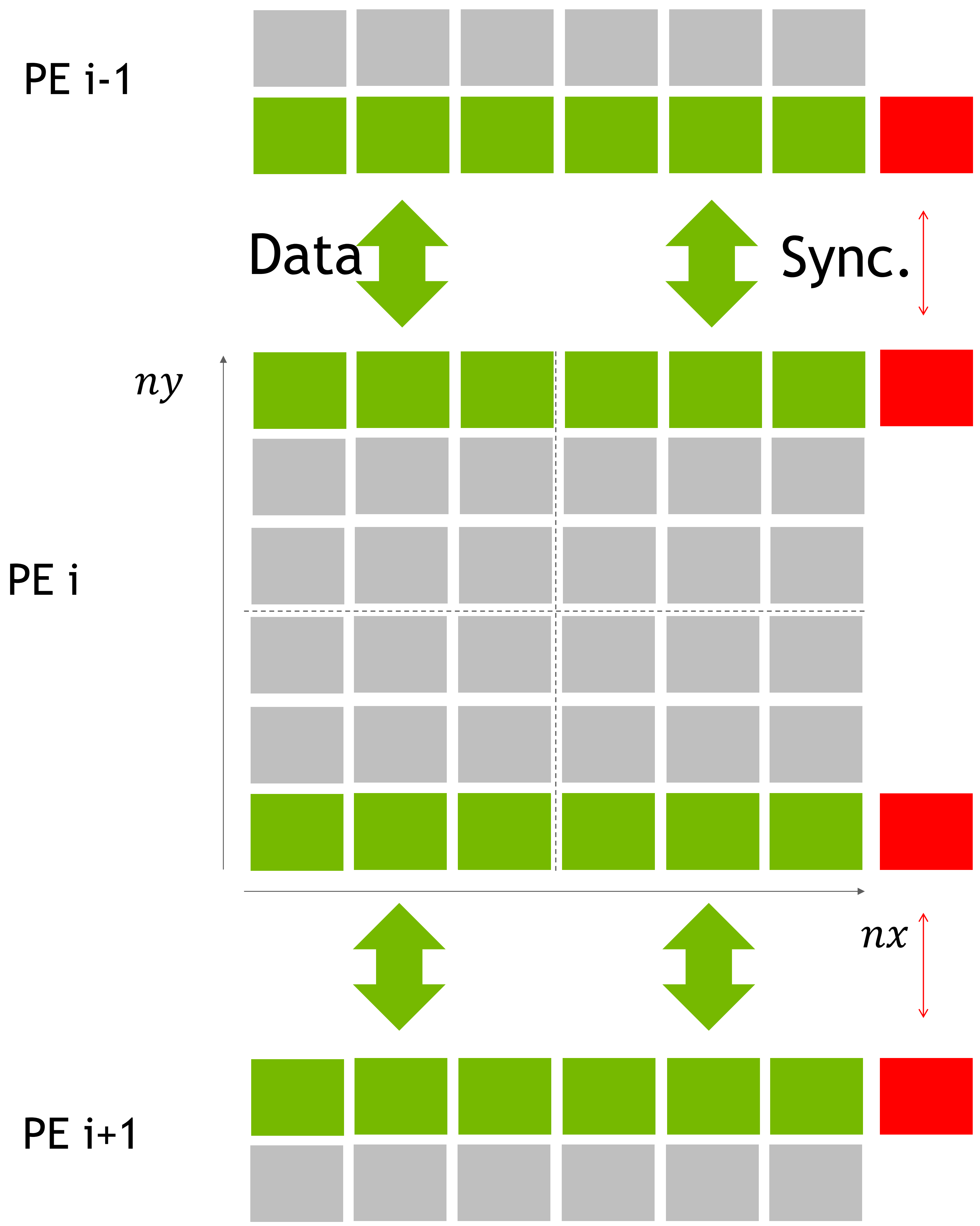
THREAD-GROUP COMMUNICATION



- NVSHMEM operations can be issued by all threads in a block/warp
- More efficient data transfers over networks like IB
- Still allows inter-warp/inter-block overlap

```
__global__ void stencil_single_step(float *u, float *v, …) {
  int ix = get_ix(blockIdx, blockDim, threadIdx);
  int iy = get_iy(blockIdx, blockDim, threadIdx);

  compute(u, v, ix, iy);

  // Thread block-level communication API
  int boffset = get_block_offet(blockIdx,blockDim);
  if (blockIdx.y == 0)
    nvshmemx_float_put_nbi_block(u+(ny+1)*nx+boffset, u+nx+boffset, blockDim.x, top_pe);
  if (blockIdx.y == (blockDim.y-1))
    nvshmemx_float_put_nbi_block(u+boffset, u+ny*nx+boffset, blockDim.x, bottom_pe);
}

for (int iter = 0; iter < N; iter++) {
   swap(u, v);
   stencil_single_step<<<..., stream>>>(u, v, …);
   nvshmem_barrier_all_on_stream(stream);
}
```

IN-KERNEL SYNCHRONIZATION



Collective or Point-to-point synchronization across PEs within a kernel

Offload larger portion of application to the same CUDA kernel

```
__global__ void stencil_multi_step(float *u, float *v, int N, int *sync, …) {
  int ix = get_ix(blockIdx, blockDim, threadIdx);
  int iy = get_iy(blockIdx, blockDim, threadIdx);

  for (int iter = 0; iter < N; iter++) {
    swap(u, v);
    compute(u, v, ix, iy);
    // Thread block-level data exchange (assume even/odd iter buffering)
    int boffset = get_block_offet(blockIdx,blockDim);
    if (blockIdx.y == 0)
      nvshmemx_float_put_nbi_block(u+(ny+1)*nx+boffset, u+n                            _pe);
    if (blockIdx.y == (blockDim.y-1))
      nvshmemx_float_put_nbi_block(u+boffset, u+ny*nx+boffs                           );

    this_grid.sync();
    if (!tid) nvshmem_barrier();
    this_grid.sync();
  }
}
```

Be aware of synchronization costs. Best synchronization approach is application dependent!

More details: https://github.com/NVIDIA/multi-gpu-programming-models

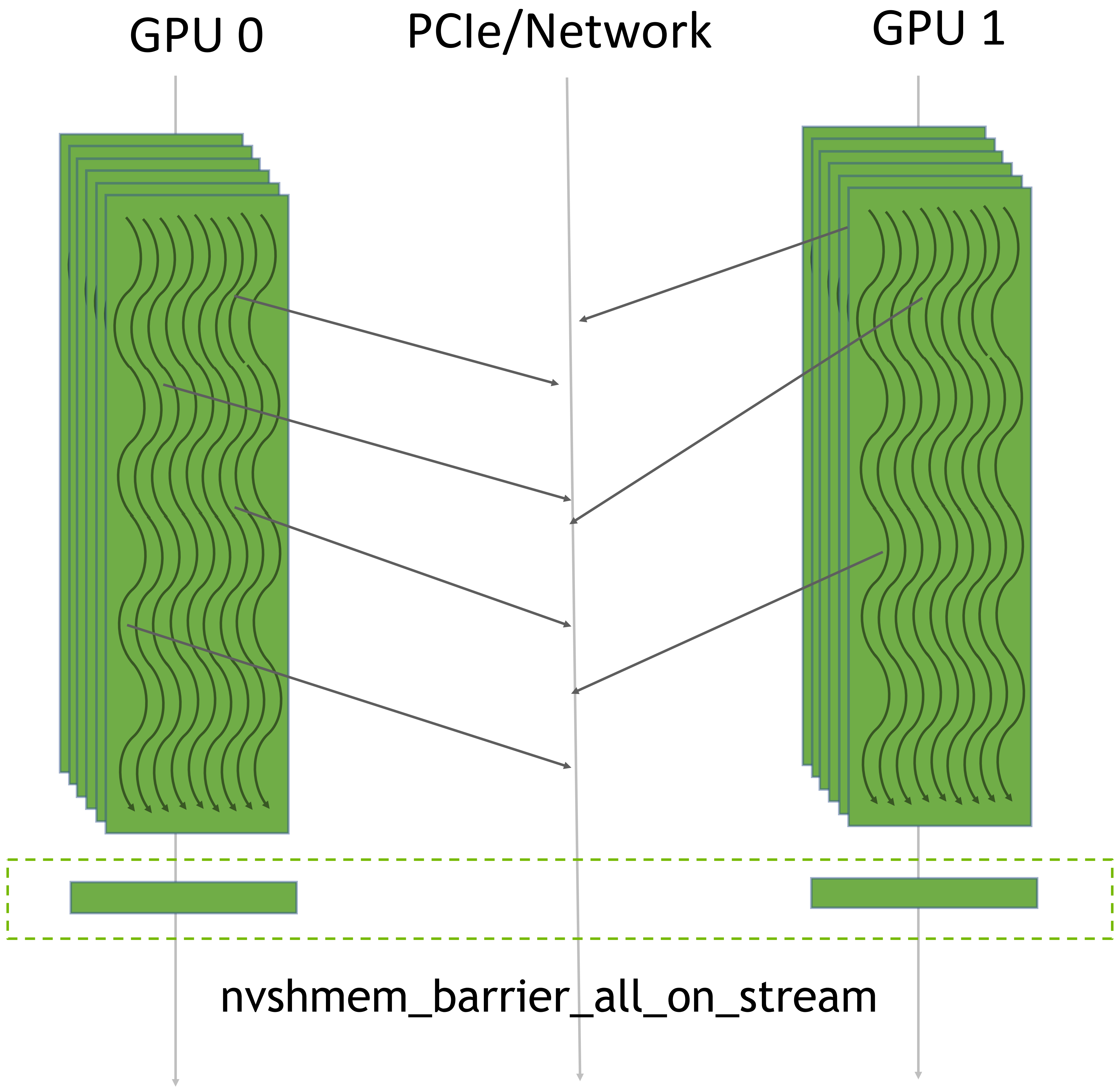
STREAM-ORDERED OPERATIONS

NVSHMEM CPU-initiated Operations Enqueued on CUDA Streams and Graphs

Not always optimal to move all communication or synchronization into CUDA kernels

Inter-CTA synchronization (e.g. *grid.sync()*) latencies can be longer than kernel launch latencies

Allows mixing fine-grained communication + coarse-grained synchronization



COLLECTIVE KERNEL LAUNCH

Ensures progress when using device-side inter-kernel synchronization

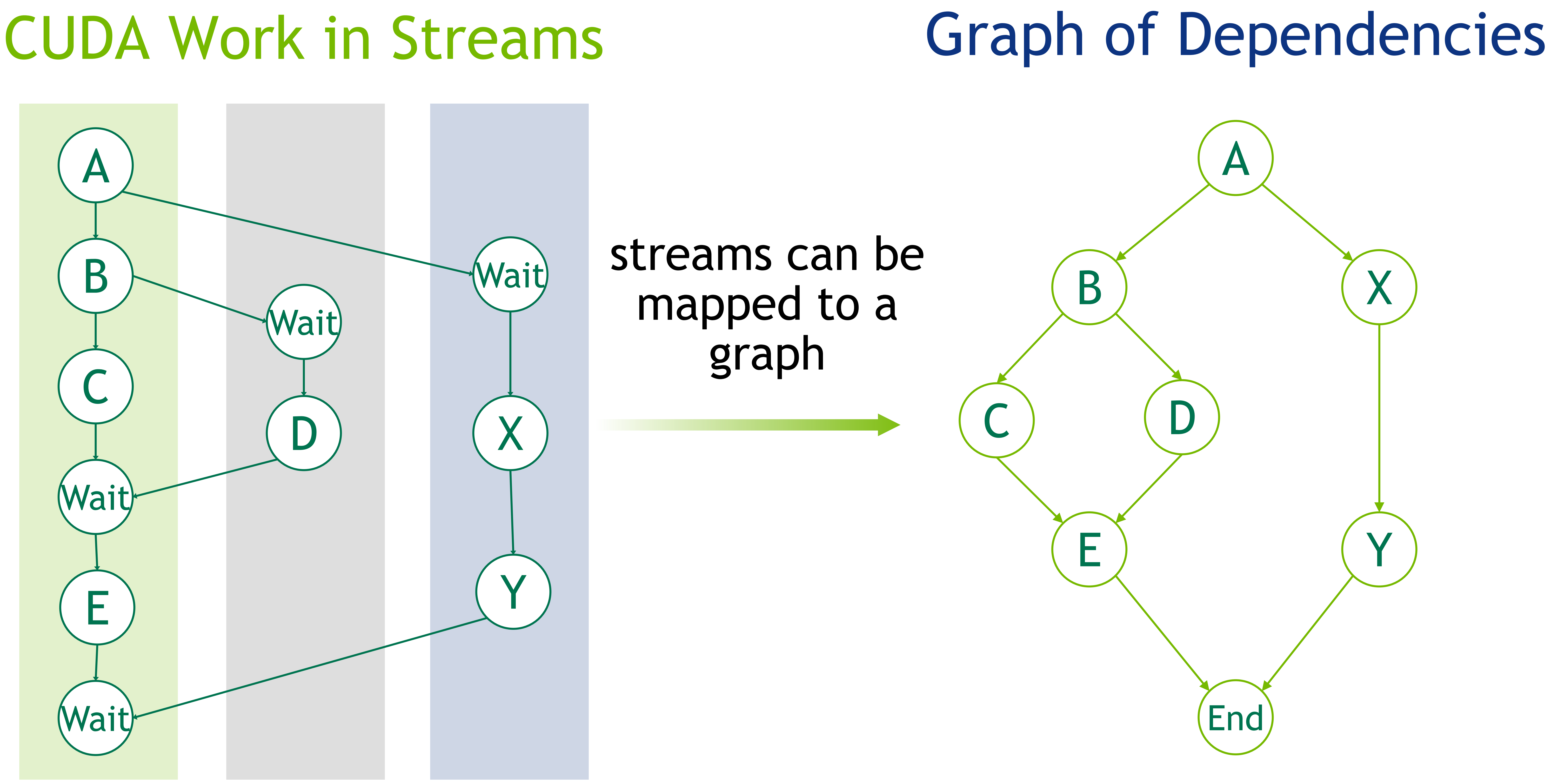
NVSHMEM Usage	CUDA kernel launch
Device-Initiated Communication	Execution config syntax <<<...>>> or launch APIs
Device-Initiated Synchronization	*nvshmemx_collective_launch*

CUDA's throughput computing model allows (encourages) grids much larger than a GPU can fit

Inter-kernel synchronization requires producer and consumer threads to execute concurrently

Collective launch guarantees co-residency using CUDA cooperative launch and the requirement of 1PE/GPU

INTEROPERABILITY WITH CUDA GRAPHS



Introduced in CUDA 10.0

Reduces launch overheads

Workflow optimizations

NVSHMEM is composable with Graphs

- On-stream operations as nodes
- CUDA kernels can use NVSHMEM

INTEROPERABILITY WITH MPI/OPENSHMEM

Enabled via Attribute-Based Initialization Routine

```
MPI_Init(&argc, &argv);

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);

mype_node = nvshmem_team_my_pe(NVSHMEMX_TEAM_NODE);
CUDA_CHECK(cudaSetDevice(mype_node));
```

MPI

Initialize NVSHMEM using MPI_COMM_WORLD

```
shmem_init();

nvshmemx_init_attr_t attr;
nvshmemx_init_attr(NVSHMEMX_INIT_WITH_SHMEM, &attr);

mype_node = nvshmem_team_my_pe(NVSHMEMX_TEAM_NODE);
CUDA_CHECK(cudaSetDevice(mype_node));
```

OPENSHMEM

Initialize NVSHMEM with SHMEM default context

Overview of NVSHMEM

NVSHMEM Latest Features

Performance Case Studies

Upcoming Features and Conclusion

NVSHMEM LATEST FEATURES

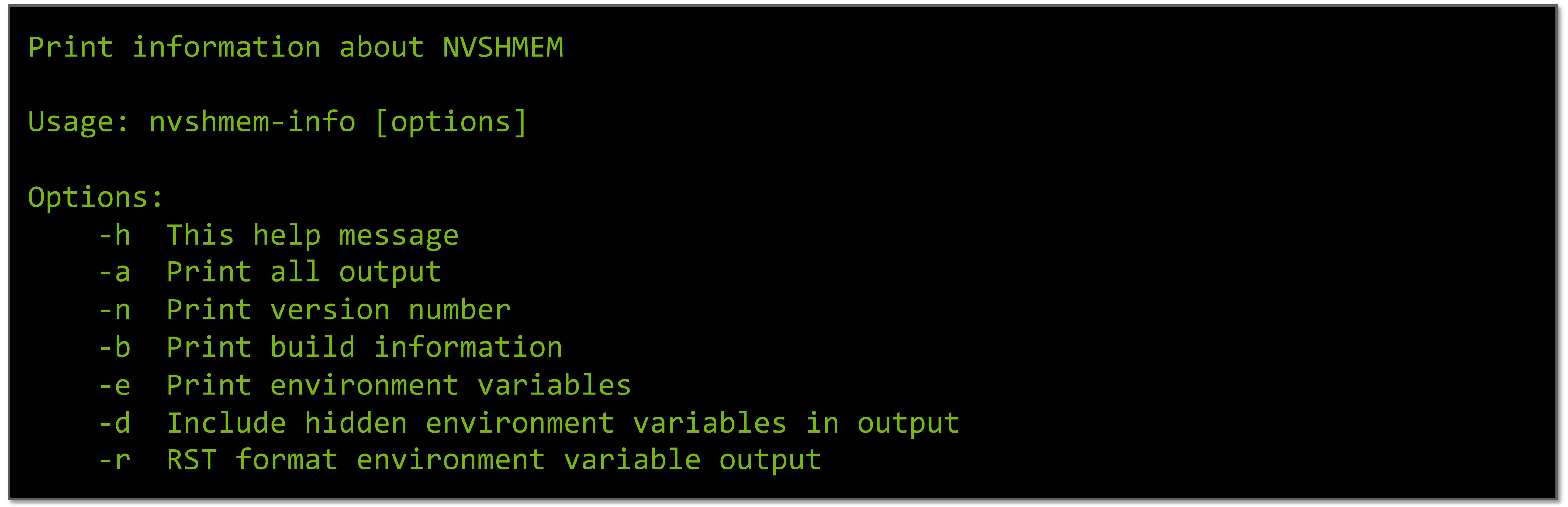
Announcing NVSHMEM 2.5.0

NVSHMEM 2.5.0 – March, 2022

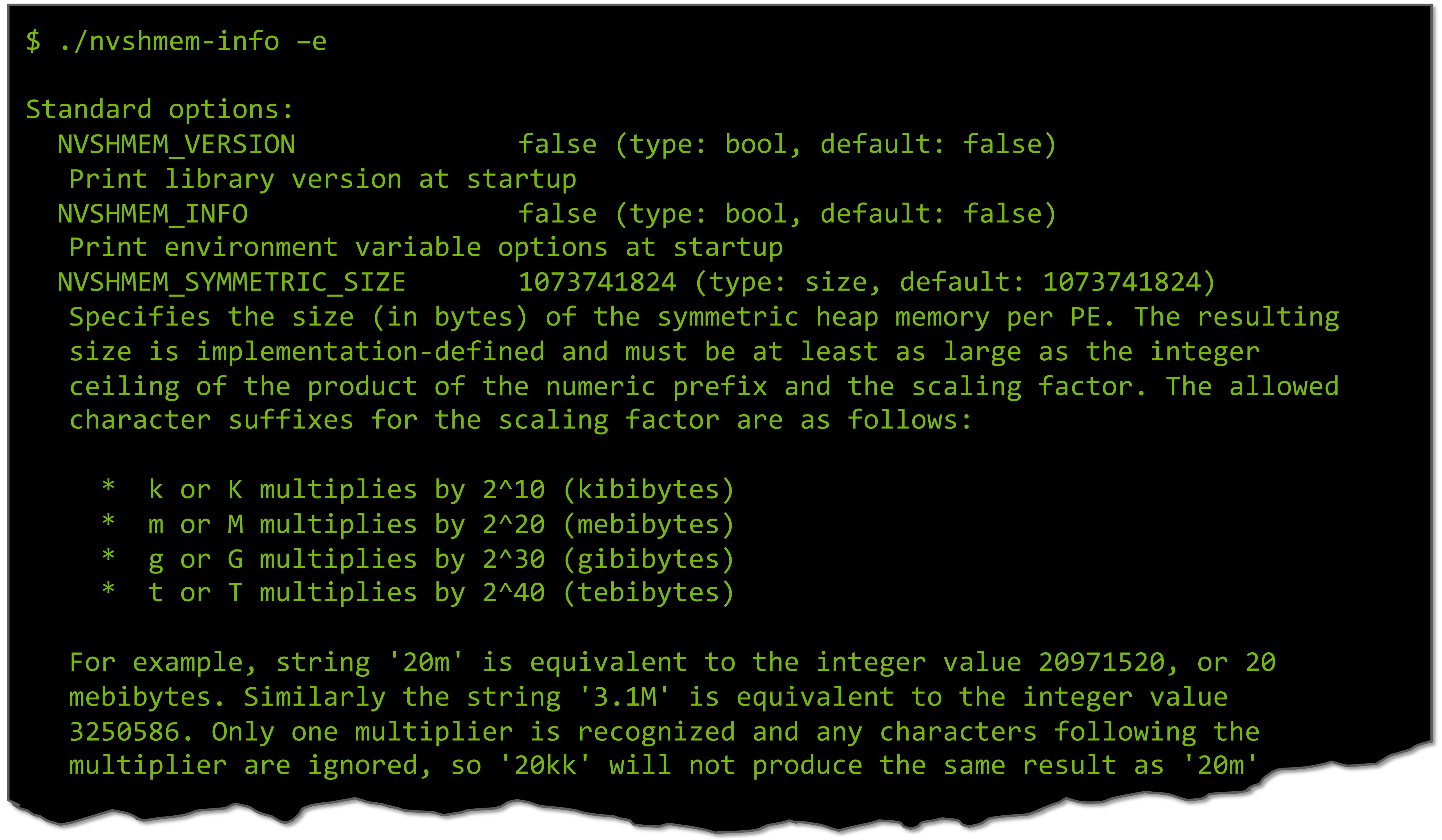
- Multiple libraries support
- Experimental libfabric and mlx5 transports
- Bootstrap plugins: PMI, PMI-2, and SHMEM
- nvshmem-info utility

NVSHMEM 2.4.1 – November, 2021

- Multi-process GPU sharing (MPG) support
- Local buffer registration API
- Dynamic symmetric heap allocation



```
Print information about NVSHMEM

Usage: nvshmem-info [options]

Options:
    -h  This help message
    -a  Print all output
    -n  Print version number
    -b  Print build information
    -e  Print environment variables
    -d  Include hidden environment variables in output
    -r  RST format environment variable output
```

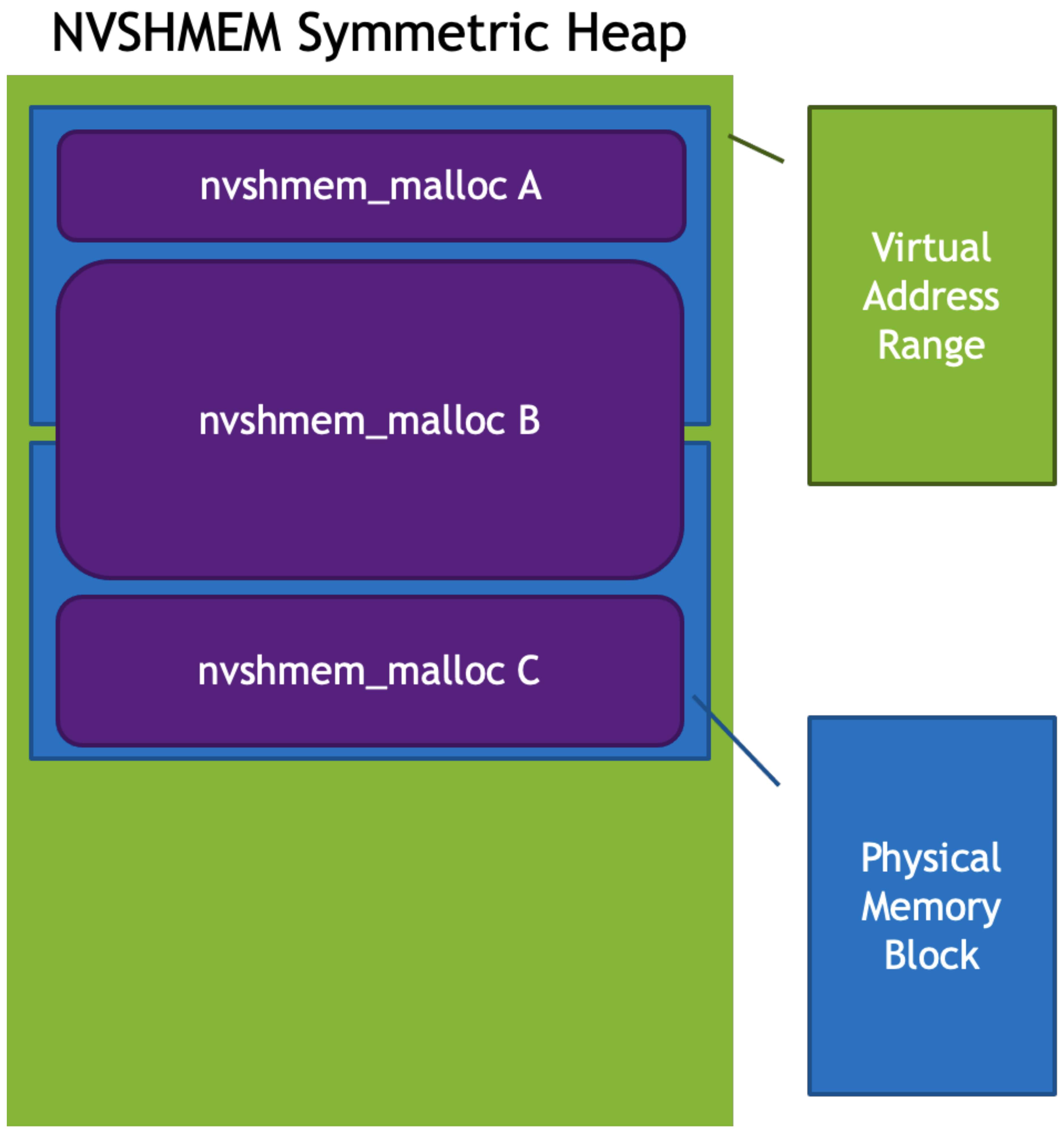


```
$ ./nvshmem-info –e

Standard options:
  NVSHMEM_VERSION              false (type: bool, default: false)
   Print library version at startup
  NVSHMEM_INFO                 false (type: bool, default: false)
   Print environment variable options at startup
  NVSHMEM_SYMMETRIC_SIZE       1073741824 (type: size, default: 1073741824)
   Specifies the size (in bytes) of the symmetric heap memory per PE. The resulting
   size is implementation-defined and must be at least as large as the integer
   ceiling of the product of the numeric prefix and the scaling factor. The allowed
   character suffixes for the scaling factor are as follows:

     *  k or K multiplies by 2^10 (kibibytes)
     *  m or M multiplies by 2^20 (mebibytes)
     *  g or G multiplies by 2^30 (gibibytes)
     *  t or T multiplies by 2^40 (tebibytes)

   For example, string '20m' is equivalent to the integer value 20971520, or 20
   mebibytes. Similarly the string '3.1M' is equivalent to the integer value
   3250586. Only one multiplier is recognized and any characters following the
   multiplier are ignored, so '20kk' will not produce the same result as '20m'
```

DYNAMIC SYMMETRIC HEAP ALLOCATION IN NVSHMEM

- NVSHMEM allocates a slab of memory as symmetric heap
  - Suballocate from slab when user calls nvshmem_malloc
- Drawbacks to this approach:
  - Users must know how much memory ahead of time
  - Reserves memory in NVSHMEM, not available to other code modules

- Dynamic symmetric heap allocation:
  - Allocate a contiguous slab of virtual address space using CUDA VMM API
  - Dynamically back virtual memory pages with physical memory pages
- NVSHMEM symmetric objects may span physical allocations
  - Bookkeeping managed by the NVSHMEM runtime



"Dynamic Symmetric Heap Allocation in NVSHMEM." Akhil Langer, Seth Howell, Sreeram Potluri, Jim Dinan, and Jiri Kraus. Sixth workshop on OpenSHMEM and Related Technologies (OpenSHMEM ’21). September 12-16, 2021.

MULTIPROCESS PER GPU (MPG) SUPPORT

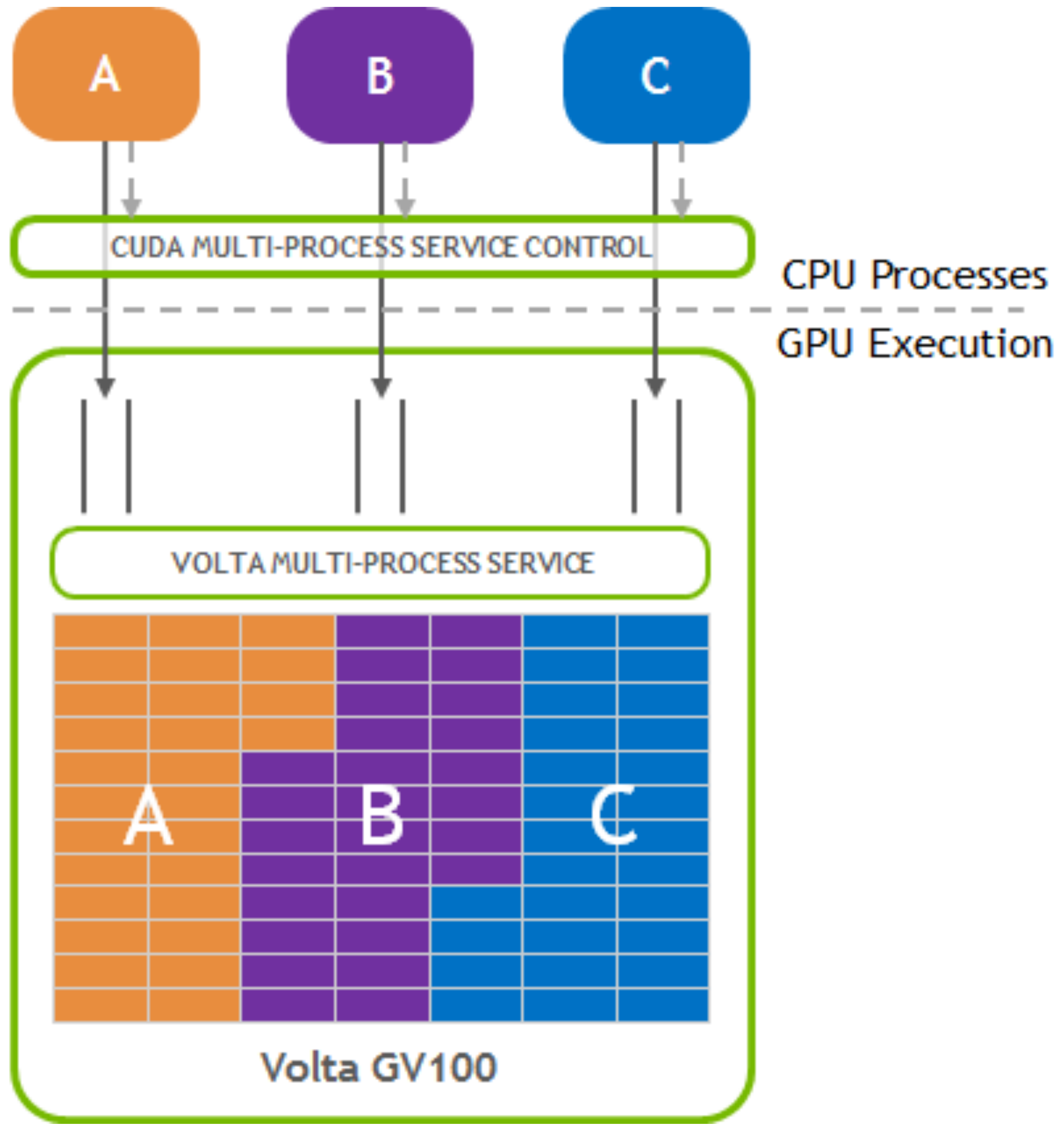
Enable Multiple Processes to Share a GPU

NVSHMEM synchronization API requires simultaneous execution of CUDA kernels for deadlock free execution

NVSHMEM 2.4 adds support for multiple processes sharing a GPU

- **CUDA Multi-Process Service (MPS)** allows multiple processes to run simultaneously on the GPU by doing resource sharing
- Users can specify percentage GPU resources used by a process using CUDA_MPS_ACTIVE_THREAD_PERCENTAGE environment variable

	API	Configurations
Full MPG Support	Entire NVSHMEM API	MPS with Sum of active thread percentages <= 100
Limited MPG Support	All point-to-point API, host nvshmem_sync_all*() host nvshmem_barrier_all*()	• Without MPS • MPS but sum of active thread percentanges > 100



A
B
C
CUDA MULTI-PROCESS SERVICE CONTROL
CPU Processes
GPU Execution
VOLTA MULTI-PROCESS SERVICE
A
B
C
Volta GV100

NVSHMEM FOR LIBRARIES

Until NVSHMEM 2.5 release, NVSHMEM only supported static linking

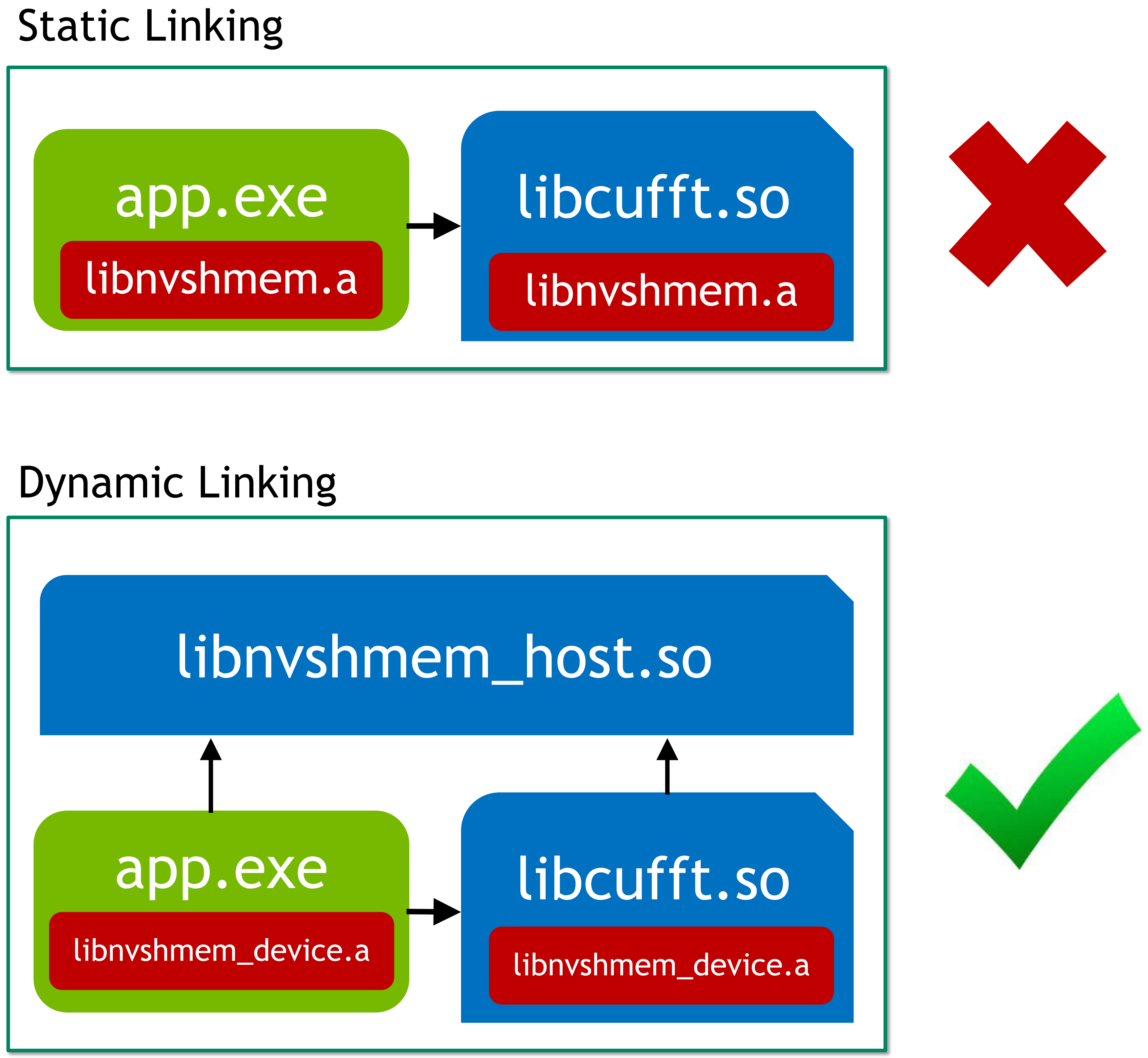
- Challenge for libraries because of separate instances of the NVSHMEM library

Challenge: NVSHMEM device API must be statically linked

- Device APIs are not accessible across shared library boundaries

Solution:

- Two libraries: libnvshmem_host.so and libnvshmem_device.a
- NVSHMEM host APIs can be dynamically linked, and connect with
- NVSHMEM device APIs that are statically linked



BOOTSTRAP

Flexible NVSHMEM Launching

During initialization, NVSHMEM's bootstrap:

- Gathers information from other PEs, e.g. hostname, GPU sharing, etc.
- Exchanges network addresses and memory registration keys

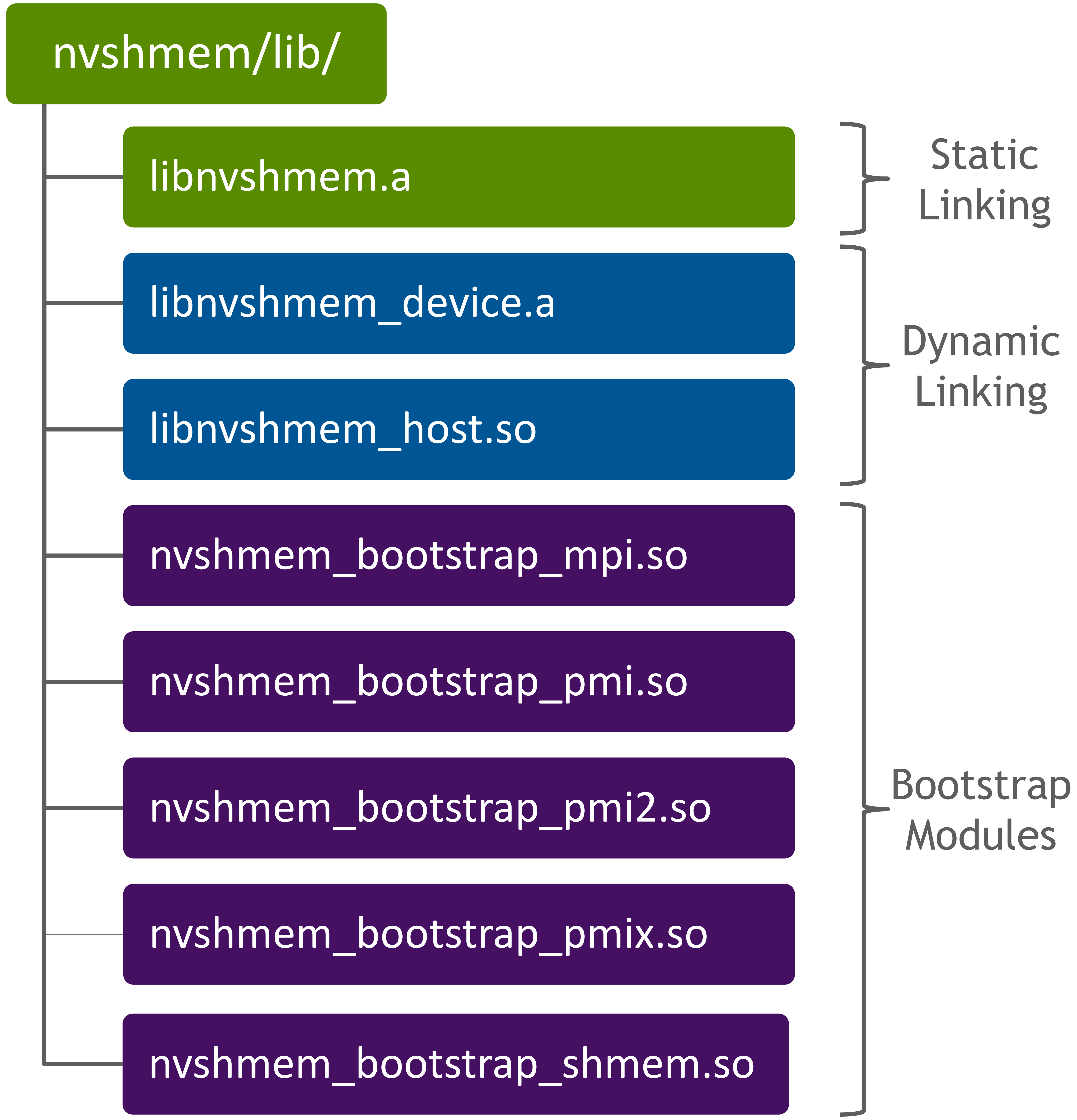
Bootstrap plugins were introduced in NVSHMEM v2.2.1

- Extended to all bootstraps in 2.5
- Provides flexibility in how NVSHMEM links with bootstrap libraries

Configured using NVSHMEM_BOOTSTRAP* environment variables or selected by calling `nvshmemx_init_attr` (See API Documentation)

Source code for bootstrap modules is installed in share/nvshmem

- Allow building bootstraps separately from NVSHMEM library
- Improves portability for binary distribution with NVSHMEM



LOCAL BUFFER REGISTRATION API

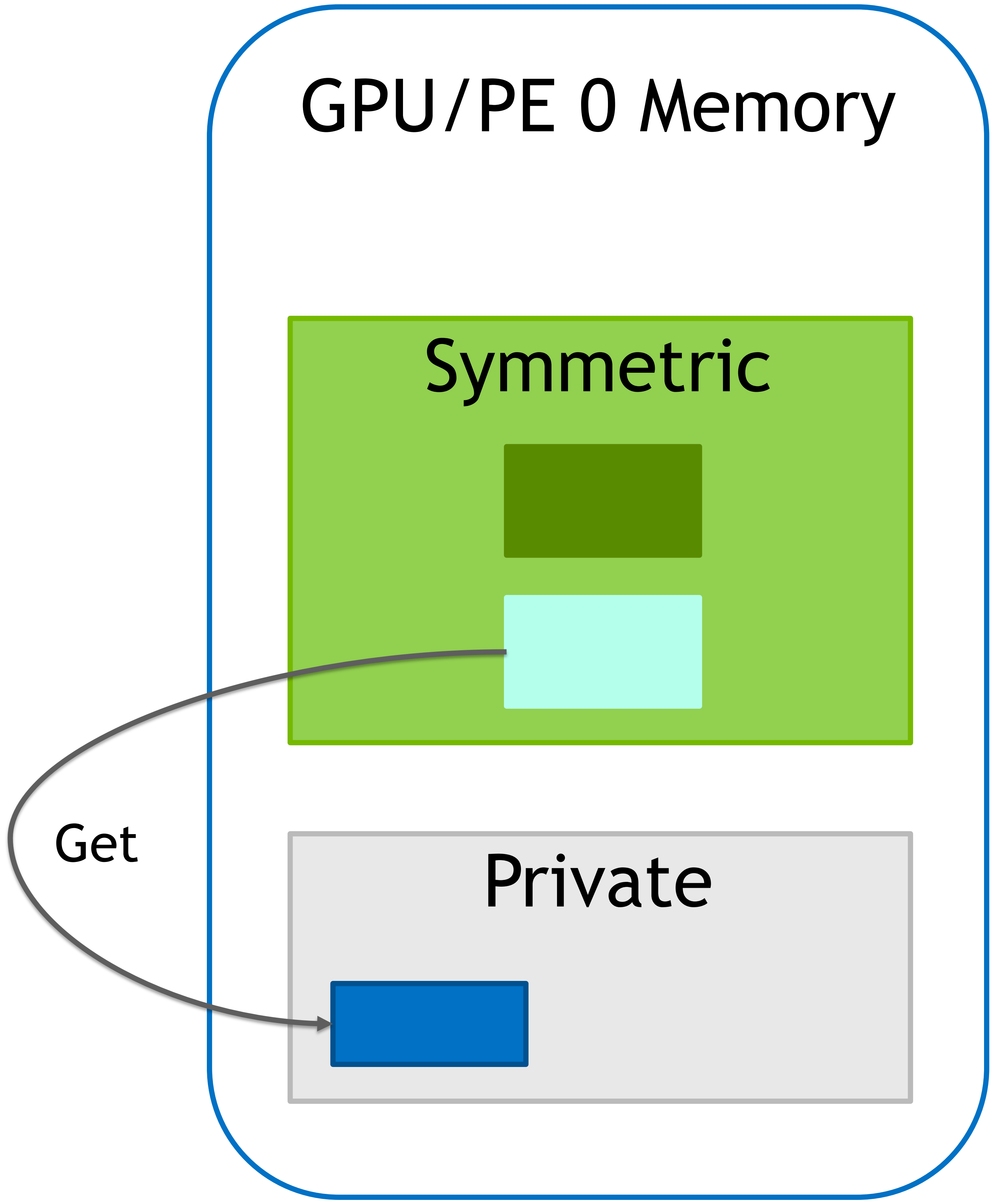
More Flexible Communication

Registered buffers can be used as the local argument in any NVSHMEM API call

- E.g. source for a put, destination for a get, etc.
- Usable for communication between GPUs on the same node and remote GPUs
- Works with host and device Memory

API Signatures:

- `int nvshmemx_buffer_register(void *addr, size_t length);`
- `int nvshmemx_buffer_unregister(void *addr);`
- `void nvshmemx_buffer_unregister_all(void);`



Overview of NVSHMEM

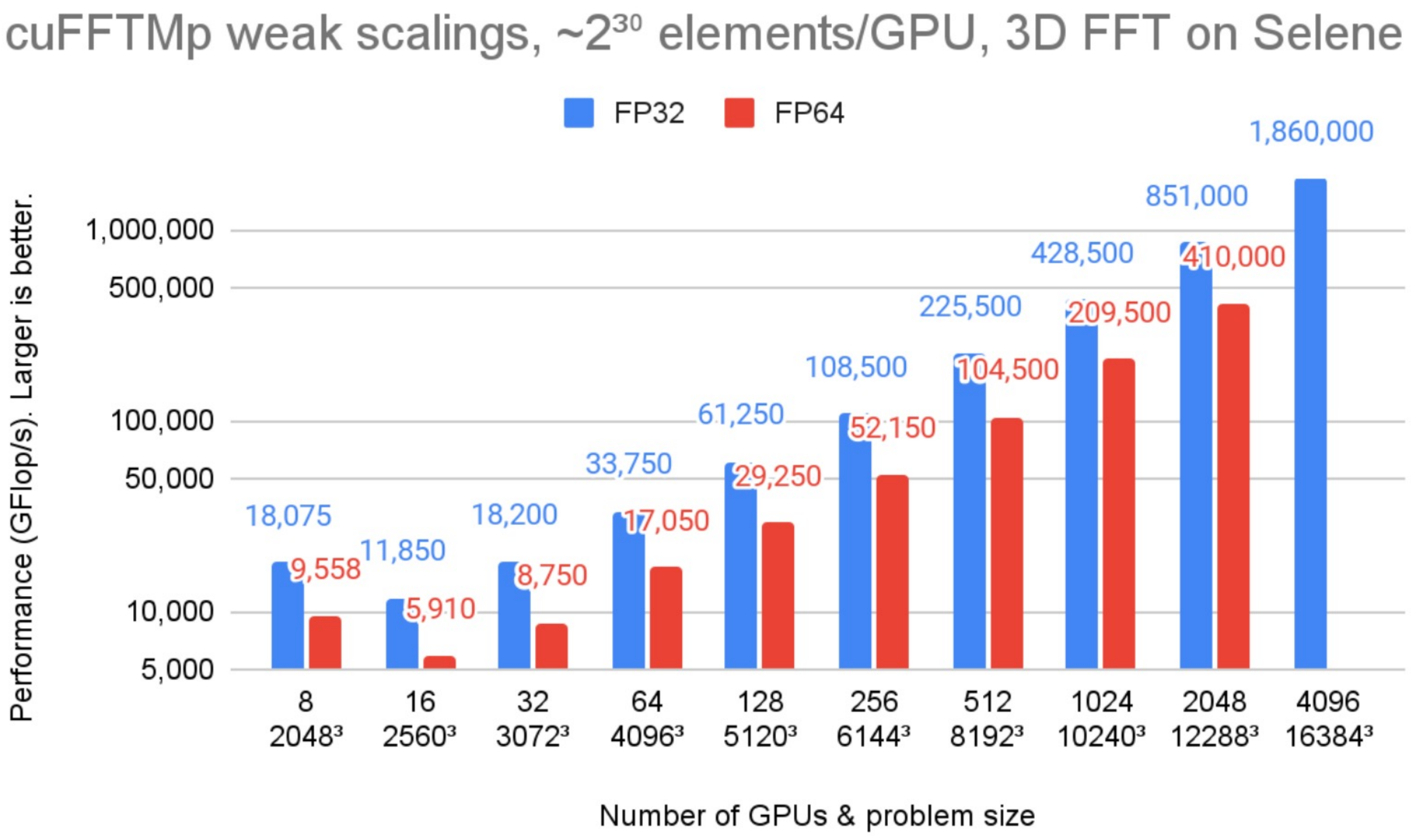
NVSHMEM Latest Features

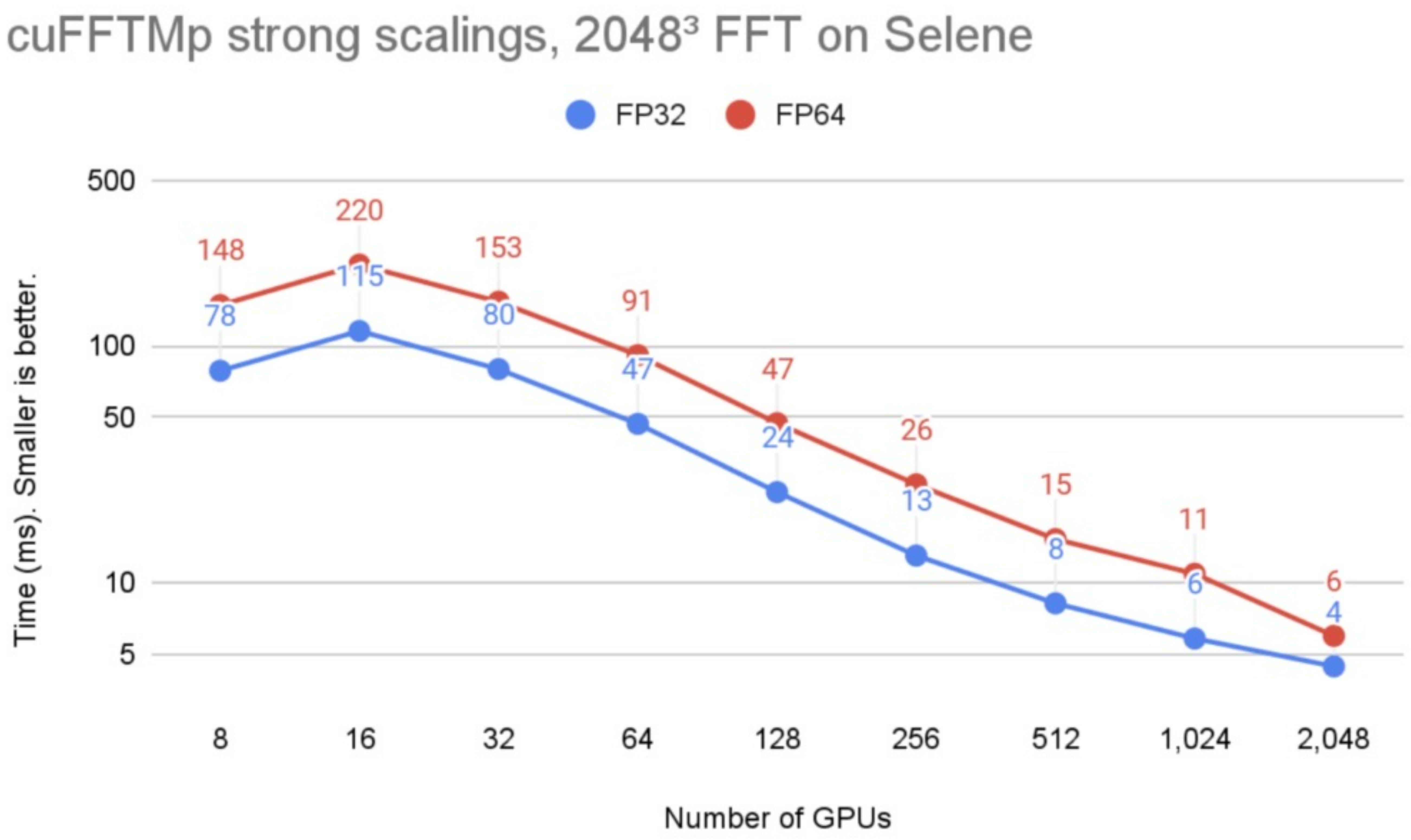
Performance Case Studies

Upcoming Features and Conclusion

CUFFTMP – MULTINODE, MULTIPROCESS CUFFT

Kernel-Initiated Communication using NVSHMEM





NVSHMEM enables efficient weak and strong scaling on Selene, DGX-A100 system

GTC Talk: S41491: Recent Developments in NVIDIA Math Libraries

Developer Blog: https://developer.nvidia.com/blog/multinode-multi-gpu-using-nvidia-cufftmp-ffts-at-scale/

SV-SIM SCALING WITH NVSHMEM

Scalable Simulation of Quantum Circuits

SV-Sim scaling-out on IBM PowerPC **CPU cores** based on OpenSHMEM on Summit

Relative Latency
2.4
2.0
1.6
1.2
0.8
0.4
0.0
cpu=32
cpu=64
cpu=128
cpu=256
cpu=512
cpu=1024
dnn_n16
bigadder_n18
cc_n18
square_root_n18
bv_n19
qft_n20
cat_state_n22
ghz_state_n23

SV-Sim scaling-out on NVIDIA V100 **GPUs** based on NVSHMEM on Summit

Relative Latency
1.2
1.0
0.8
0.6
0.4
0.2
0.0
gpu=4
gpu=8
gpu=16
gpu=32
gpu=64
gpu=128
gpu=256
gpu=512
gpu=1024
dnn_n16
bigadder_n18
cc_n18
square_root_n18
bv_n19
qft_n20
cat_state_n22
ghz_state_n23

More details in Ang Li's GTC Talk: *S41855: Scalable Simulation of Quantum Circuit with Noise on GPU-based HPC Systems*

Ang Li, et al., "QASMBench: A low-level QASM benchmark suite for NISQ evaluation and simulation."
arXiv preprint arXiv:2005.13018 (2020). https://github.com/pnnl/QASMBench

Overview of NVSHMEM

NVSHMEM Latest Features

Performance Case Studies

Upcoming Features and Conclusion

LIBFABRIC TRANSPORT

Supporting Phase 2 of NERSC-9 Perlmutter System

- Experimental libfabric transport added in NVSHMEM v2.5.0
- The libfabric transport will support HPE Cray Slingshot-11 networks
- Will be used to support Phase 2 of NERSC-9 Perlmutter system
- Perlmutter Phase 1 system uses Slingshot-10 configuration
  - NVIDIA Mellanox ConnectX-5 HCAs
  - Supported via existing NVSHMEM ibverbs transport

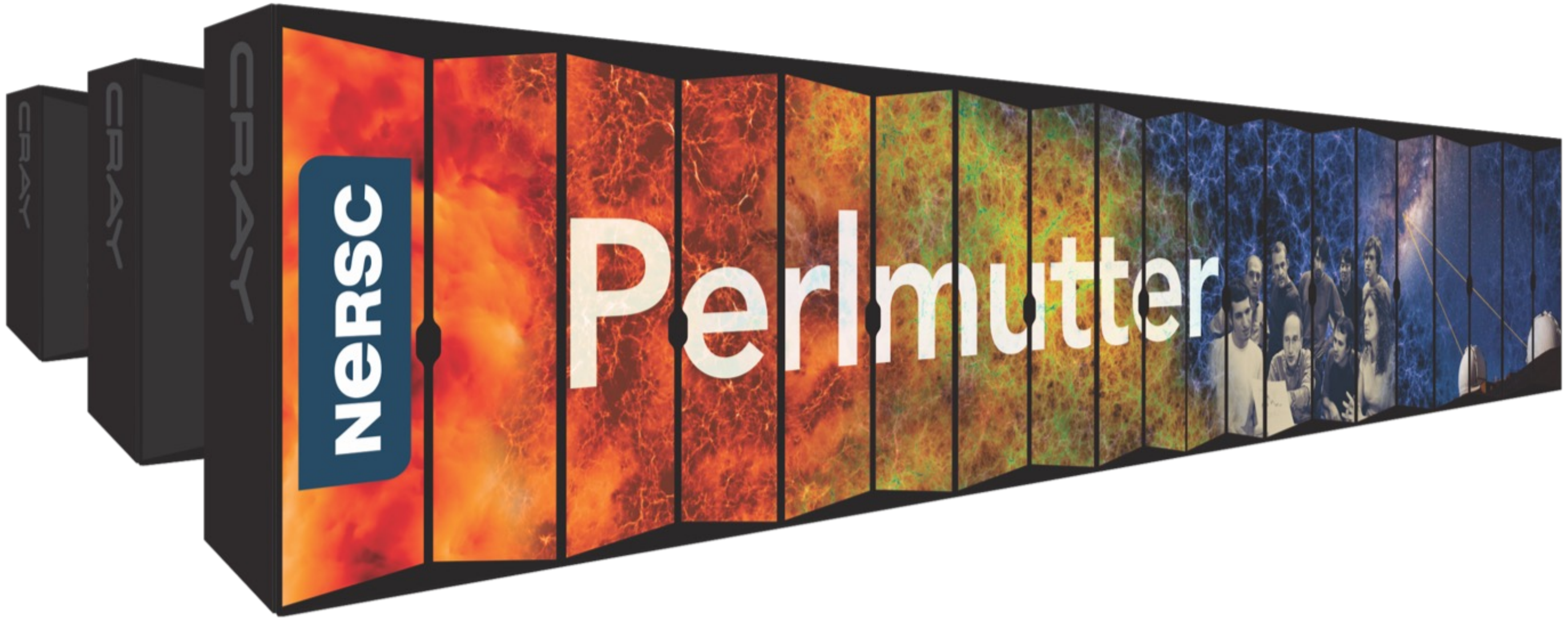


Image credit: National Energy Research Scientific Computing Center

GPU INITIATED COMMUNICATION TRANSPORT

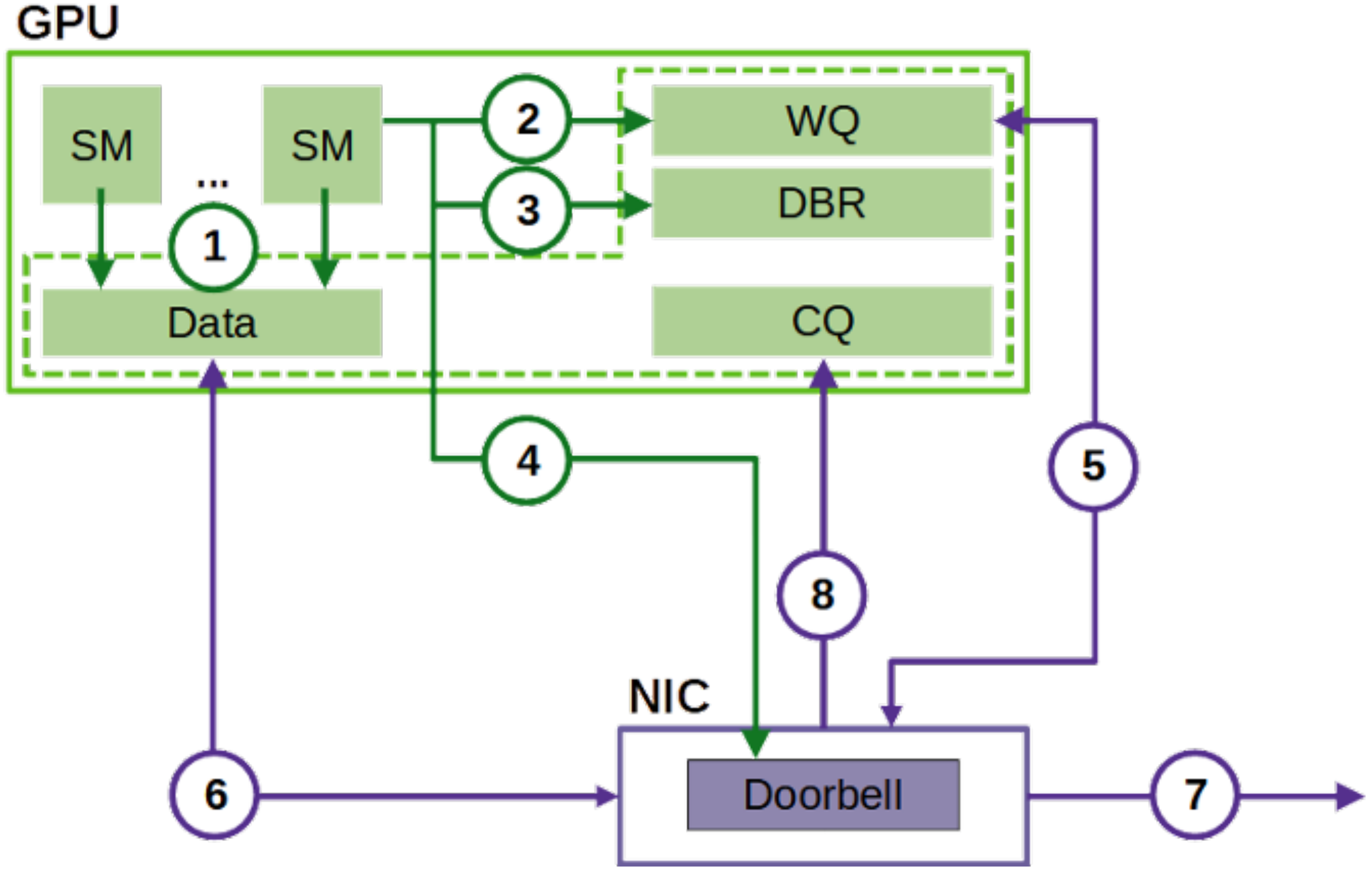
Tighter Integration With Mellanox

Experimental DevX transport added in 2.5.0

- Supports NVIDIA Mellanox networks
- Runs directly over Mellanox software stack and bypasses Verbs API for performance critical operations
- Avoids need for GDRCopy by leveraging native atomics

Precursor to GPU Initiated Communication Transport

- CPU proxy thread will no longer be needed for kernel-initiated communication
- Increased parallelism will lead to improved throughput of small message sizes



For more information, see: S41825: Latest on NVIDIA Magnum IO GPUDirect Technologies

LESSONS FOR MPI RMA

One-sided communication is a good fit for GPU initiated communication

- Simple protocols allow GPU to fully perform communication
- Enables a model where all operations map to GPUDirect RDMA
- Asynchronous data movement is a good fit for data parallel computation (massively multithreaded)
- Application chooses synchronization that's compatible with data parallel execution model

Some MPI RMA features are contra-indicated

- Window synchronization models can force synchronization that is inefficient or deadlock prone in a data parallel execution model
- Window synchronization (e.g. lock/unlock) can violate the "simple protocols" observation
- Window creation model chosen by the user may not be conducive to IPC (peer-to-peer communication)
- Accumulate ordering hurts performance on weakly consistent SM

RMA is an incomplete PGAS model

- Point-to-point synchronization (put-and-notify, wait)
- Atomic limitations prevent building important things, like scalable MCS locks (mix CAS and ADD)
- Not possible to request only ordering (nvshmem_fence)
- Bulk synchronous RMA completion can require applications to mix active and passive target synchronization models

SUMMARY

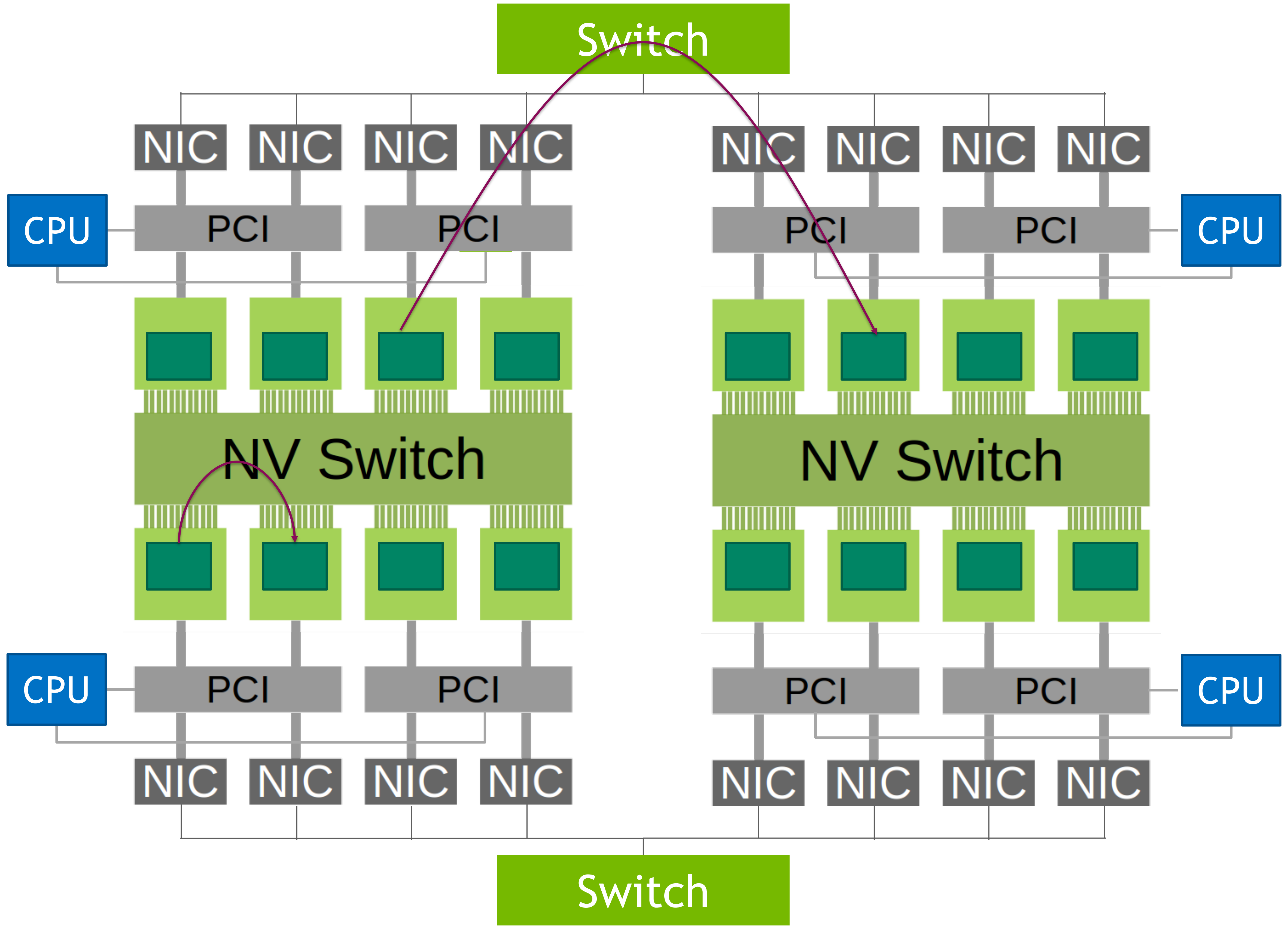
NVSHMEM is a PGAS Library for Clusters of NVIDIA GPUs

NVSHMEM seamlessly scales from,

- Node-level GPU programming with NVLink connected GPUs
- Multi-node GPU clusters connected with InfiniBand or RoCE

NVSHMEM provides Stream/Graph, GPU kernel-initiated, and CPU initiated APIs

- Integration with CUDA programming model can improve performance and improve ease of scaling to GPU clusters



NVSHMEM STATUS

Latest release: NVSHMEM 2.5.0, March 2022

❑ Available to download at https://developer.nvidia.com/nvshmem

❑ Available in upcoming NVIDIA HPC SDK release, containers available through NGC

New Features

- Dynamic linking support
- Multiprocess per GPU support
- Flexible bootstrapping
- User-allocated buffer registration

Implementation Features

- NVLink and PCIe P2P support
- InfiniBand and RoCE support
- X86 and Power9 support
- Interoperable with MPI and OpenSHMEM

NVIDIA®