Fakultät für
Mathematik und Informatik

CUDA Graphs and Device-initiated Communication with NVSHMEM

Lena Oden, FernUniversität in Hagen

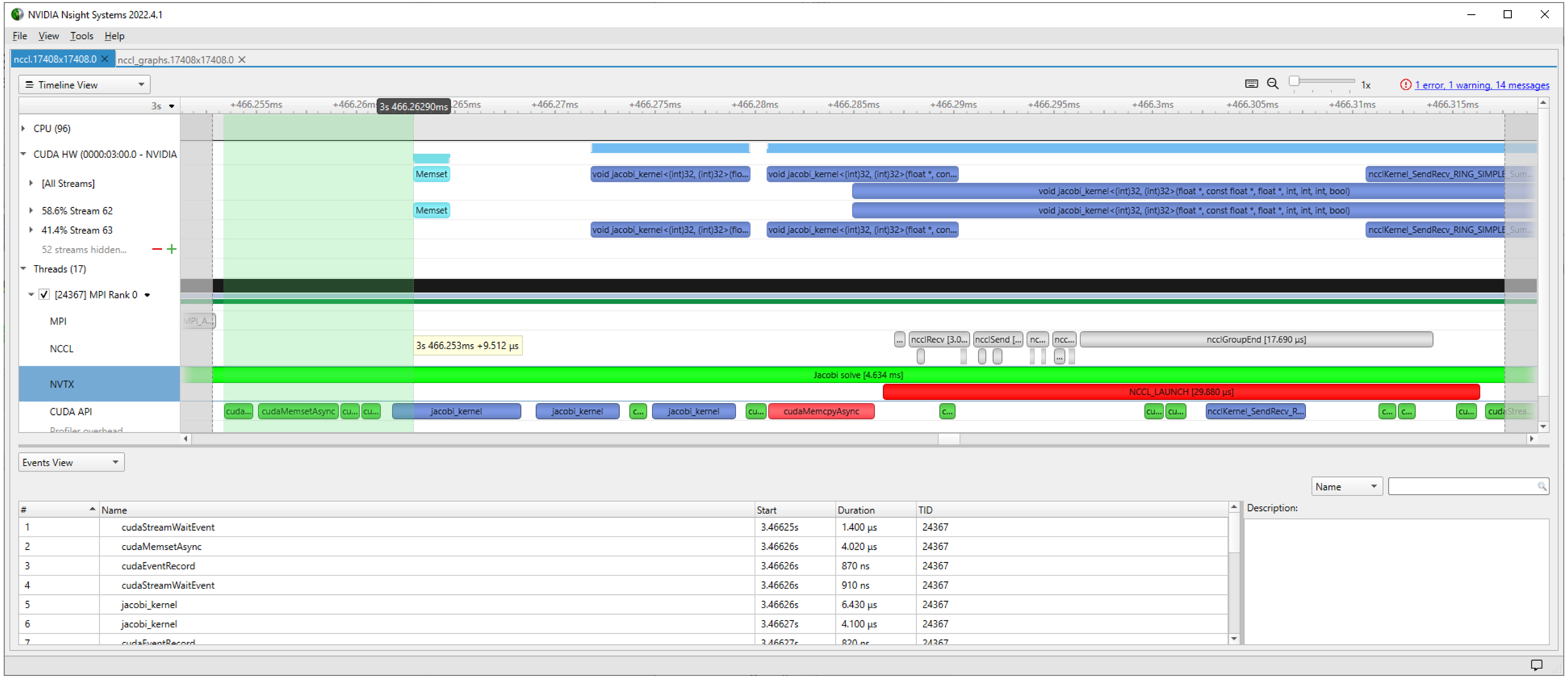
Jiri Kraus, Principal Devtech Compute, Nvidia

FernUniversität in Hagen

What you will learn?

- CUDA-Graphs
  - What are CUDA-Graphs
  - Stream-Capture API
  - How to use them
- GPU-Initiated Communication
  - Thread Leve Communication
  - Group Level Communication
  - On-Device Synchronisation
  - Implementation : Performance

Multi GPU Jacobi Nsight Systems Timeline



Asynchronous Task Graph

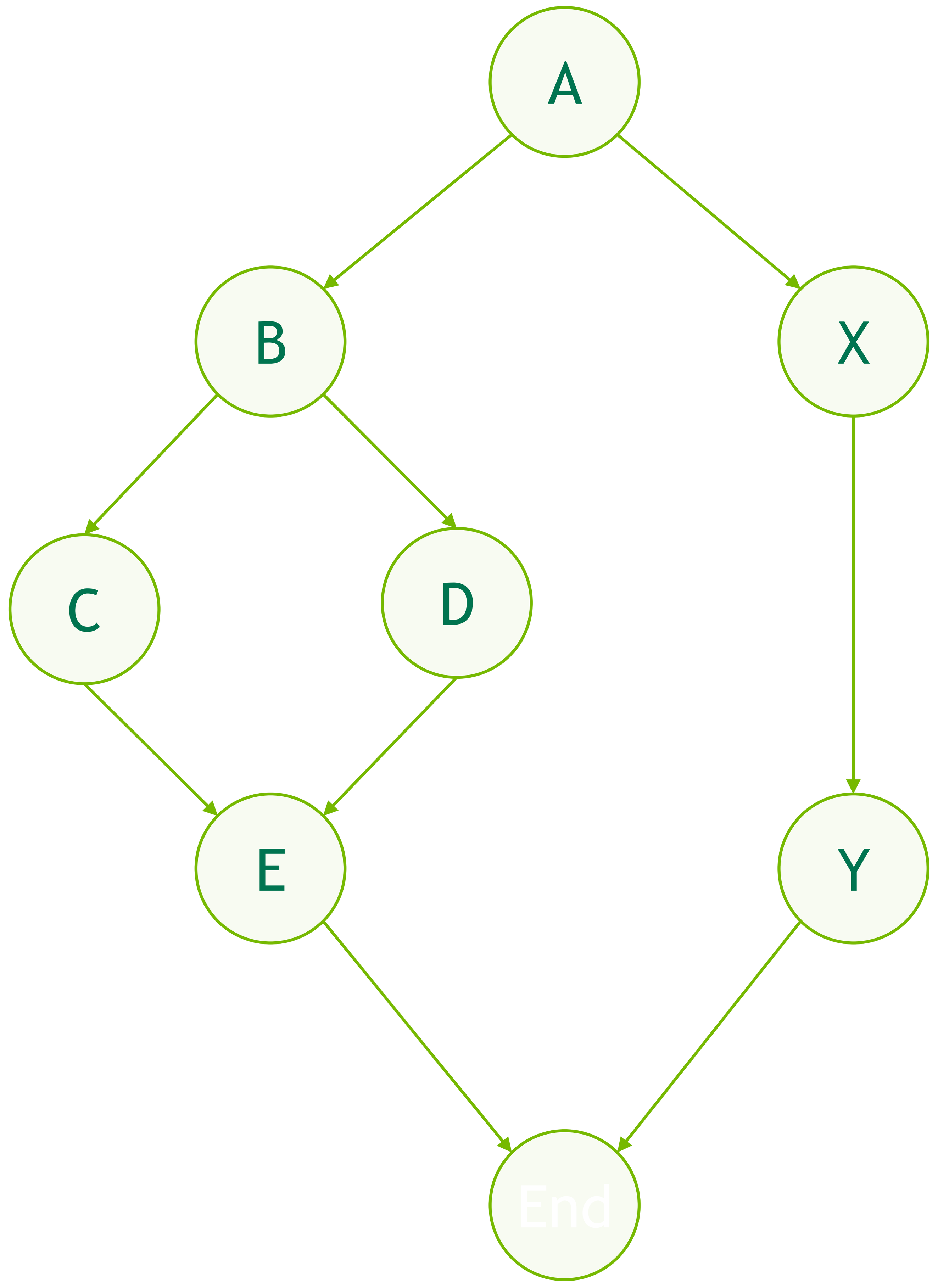
A Graph Node Is A CUDA Operation

Sequence of operations (nodes), connected by dependencies

Operations are one of:

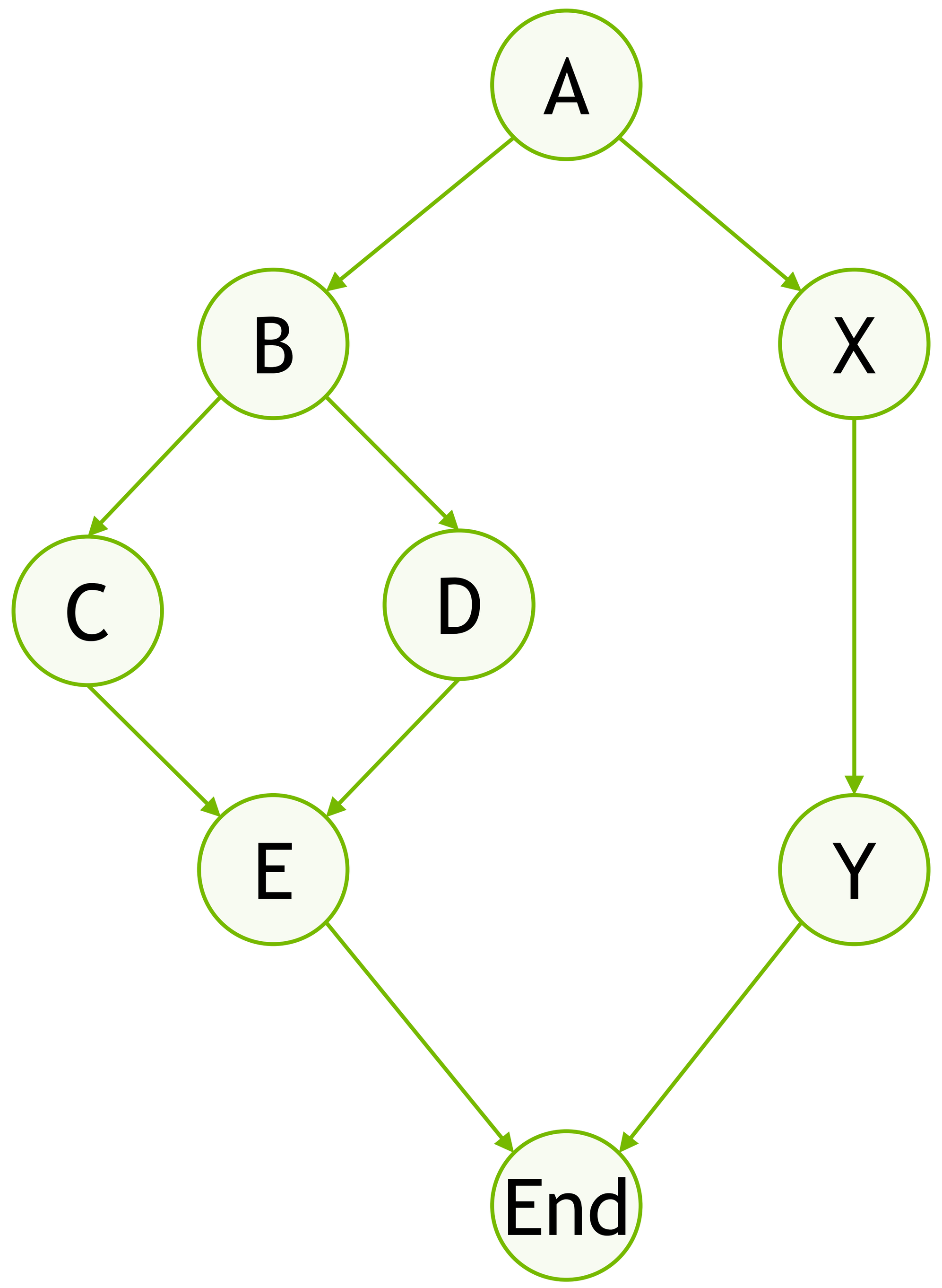
Kernel Launch	CUDA kernel running on GPU
CPU Function Call	Callback function on CPU
Memcopy/Memset	GPU data management
Mem Alloc/Free	Memory management
External Dependency	External semaphores/events
Sub-Graph	Graphs are hierarchical

Nodes within a graph can also span multiple devices



Three-Stage Execution Model

Define

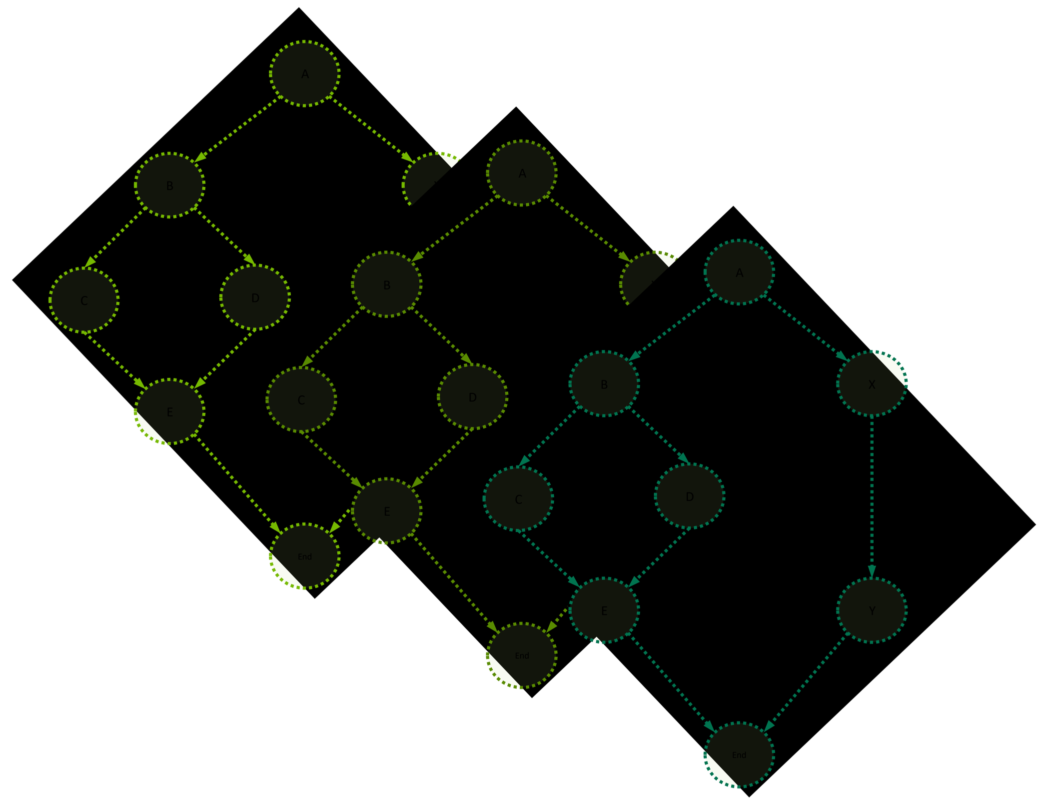


Single Graph "Template"

Created in host code

or built up from libraries

Instantiate



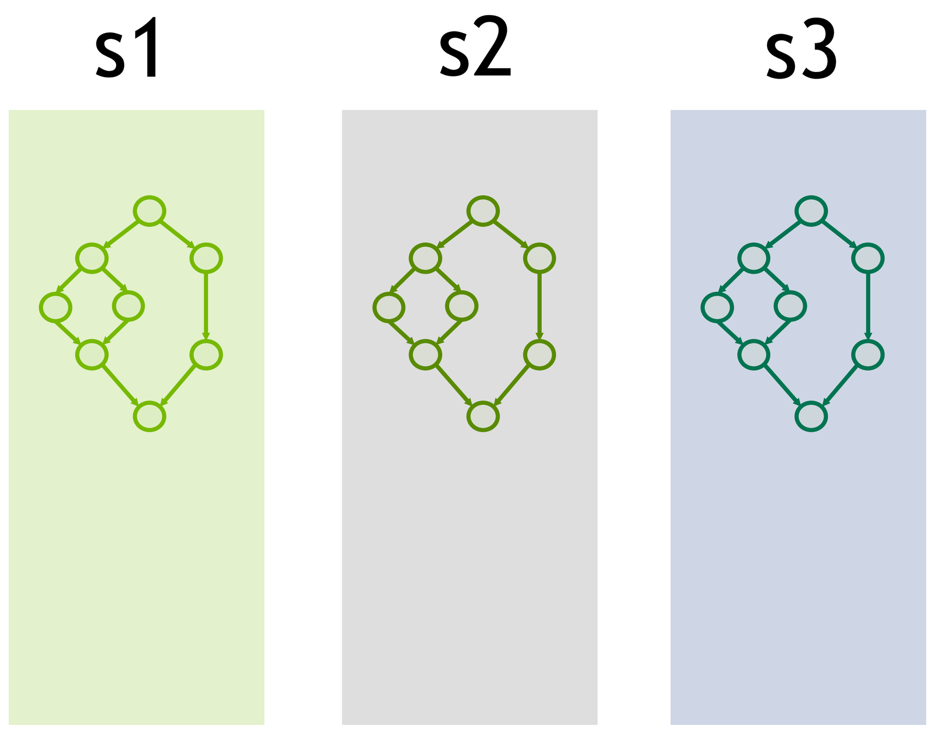
Multiple "Executable Graphs"

Snapshot of templates

Sets up & initializes GPU execution structures

(create once, run many times)

Execute



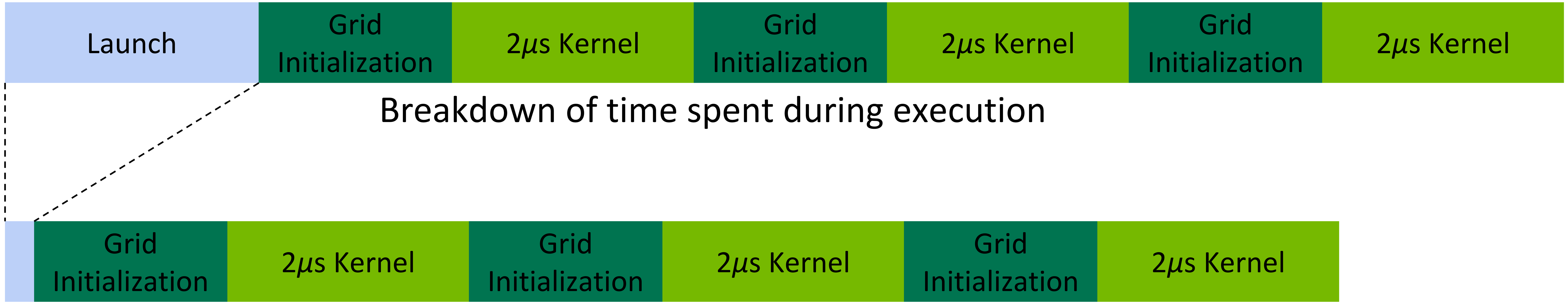
Executable Graphs Running in CUDA Streams

Concurrency in graph **is not** limited by stream

Where is performance coming from?

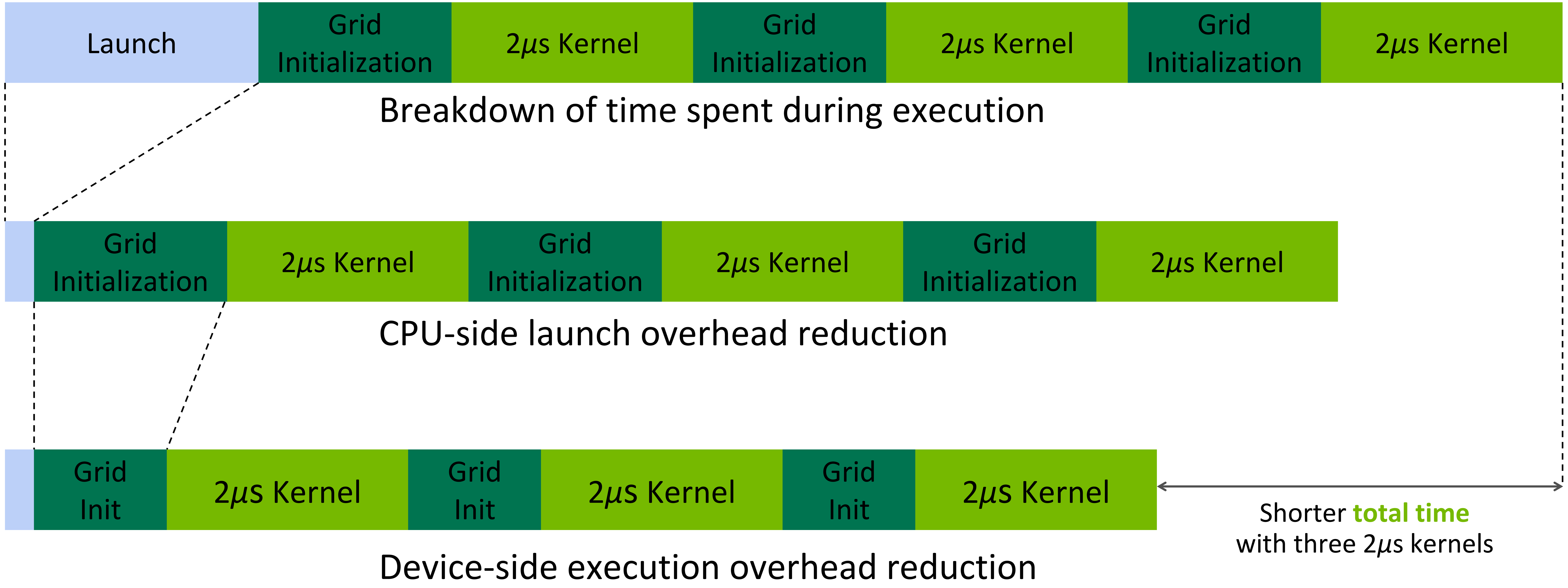
Breakdown of time spent during execution

Where is performance coming from?



Breakdown of time spent during execution

Where is performance coming from?



Capture Stream work into a Graph

Create A Graph With Two Lines of Code

```
{

    cudaStreamBeginCapture(compute_stream, cudaStreamCaptureModeGlobal);

    cudaMemsetAsync(l2_norm_d, 0, sizeof(real), compute_stream));
    cudaEventRecord(reset_l2norm_done, compute_stream);

    ...



    cudaStreamEndCapture(compute_stream, graphs[calculate_norm]+is_even);

    std::swap(a_new, a);
    iter++;
}
```

Capture Stream work into a Graph

Create A Graph With Two Lines of Code

```
{

    cudaStreamBeginCapture(compute_stream, cudaStreamCaptureModeGlobal);

    cudaMemsetAsync(l2_norm_d, 0, sizeof(real), compute_stream));
    cudaEventRecord(reset_l2norm_done, compute_stream);

    ...
    cudaEventRecord(push_done, push_stream);
-   ...

    cudaStreamWaitEvent(compute_stream, push_done, 0);

    cudaStreamEndCapture(compute_stream, graphs[calculate_norm]+is_even);

    std::swap(a_new, a);
    iter++;
}
```

CUDA Graph Management API

Instantiate CUDA Graphs

```
__host__cudaError_t cudaGraphInstantiateWithFlags ( cudaGraphExec_t* pGraphExec, cudaGraph_t graph,
                                                    unsigned long long flags )
```

- pGraphExec [OUT]   Returns instantiated graph
- graph [IN]: Graph to instantiate
- flags [IN]: Flags to control instantiation (cudaGraphInstantiateFlagAutoFreeOnLaunch |

  cudaGraphInstantiateFlagUseNodePriority |

  cudaGraphInstantiateFlagDeviceLaunch)

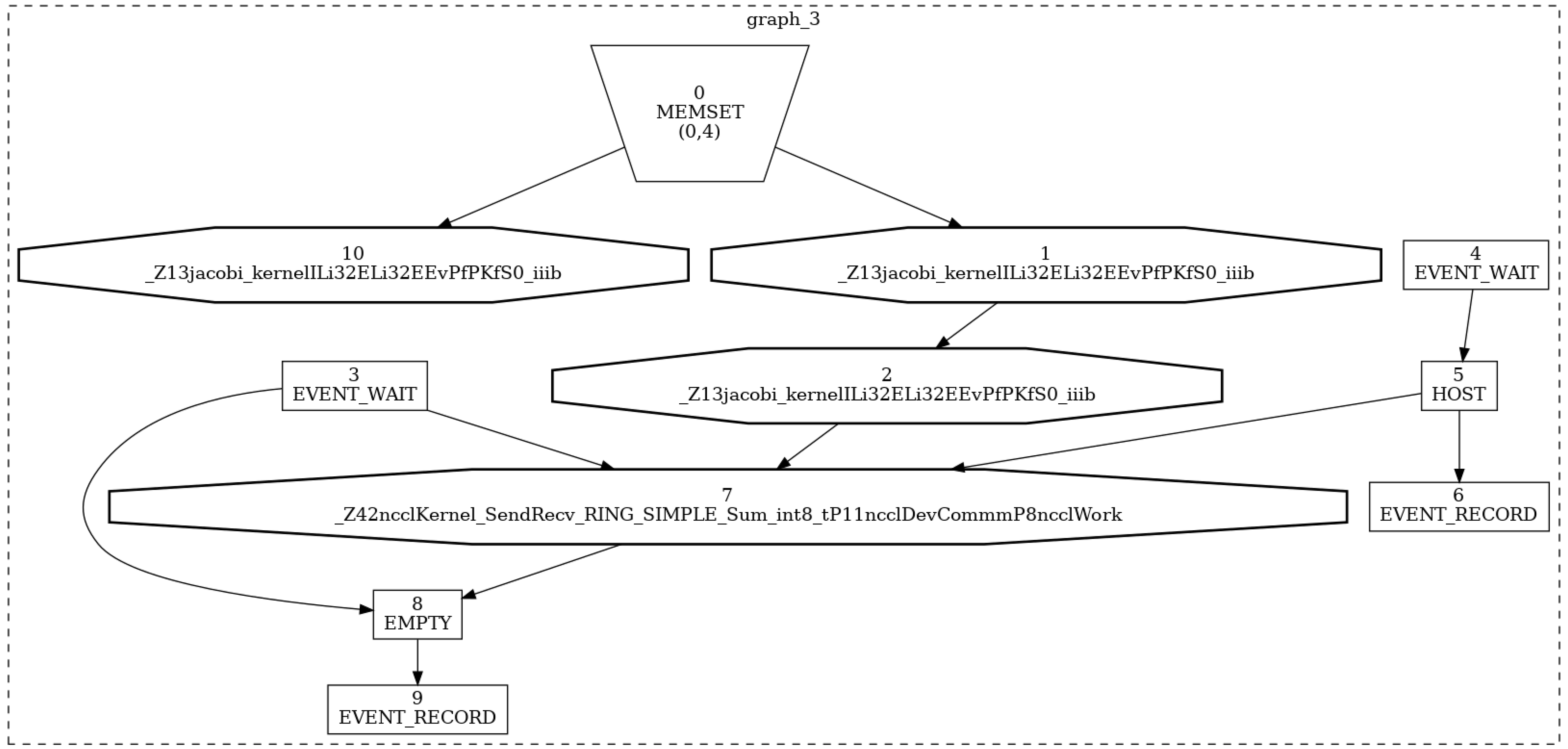
).

- Returns: cudaSuccess, cudaErrorInvalidValue

New Execution Mechanism

Graphs Can Be Generated Once Then Launched Repeatedly

```
while (l2_norm > tol && iter < iter_max) {

    cudaGraphLaunch(graph_calc_norm_exec[iter%2],
                    compute_stream);
    cudaStreamSynchronize(compute_stream);
    MPI_Allreduce(l2_norm_h, &l2_norm, 1,
                  MPI_REAL_TYPE, MPI_SUM,
                  MPI_COMM_WORLD);
    l2_norm = std::sqrt(l2_norm);

    if (!csv && 0 == rank && (iter % 100) == 0) {
        printf("%5d, %0.6f\n", iter, l2_norm);
    }
}
```




```
cudaGraphDebugDotPrint(graphs[calculate_norm][0],
                       "jacobi_graph.dot",0)
```

```
dot -Tpng jacobi_graph.dot -o jacobi_grap.png
```

CUDA Graph Management API

```
__host__ cudaError_t cudaGraphDestroy( cudaGraph_t graph)
```

- graph [IN]: Graph to destroy

Returns: cudaSuccess, cudaErrorInvalidValue

Destroys the graph specified by graph, as well as all of its nodes.

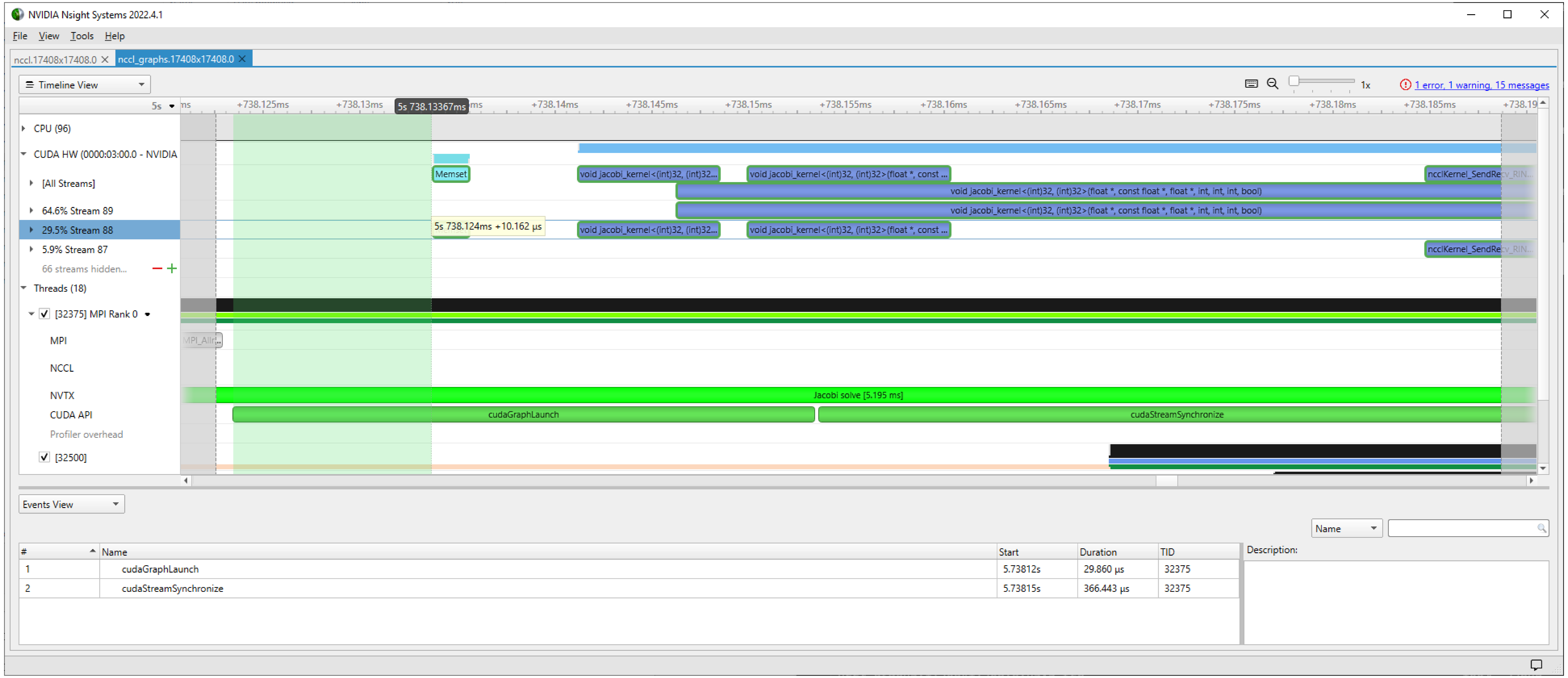
```
__host__cudaError_t cudaGraphExecDestroy ( cudaGraphExec_t graphExec )
```

- graphExec [IN]: Executable graph to destroy

Returns: cudaSuccess, cudaErrorInvalidValue

Destroys the executable graph specified by graphExec.

Multi GPU Jacobi Nsight Systems Timeline



CPU-Initiated Communication

Compute on GPU

Communication from CPU

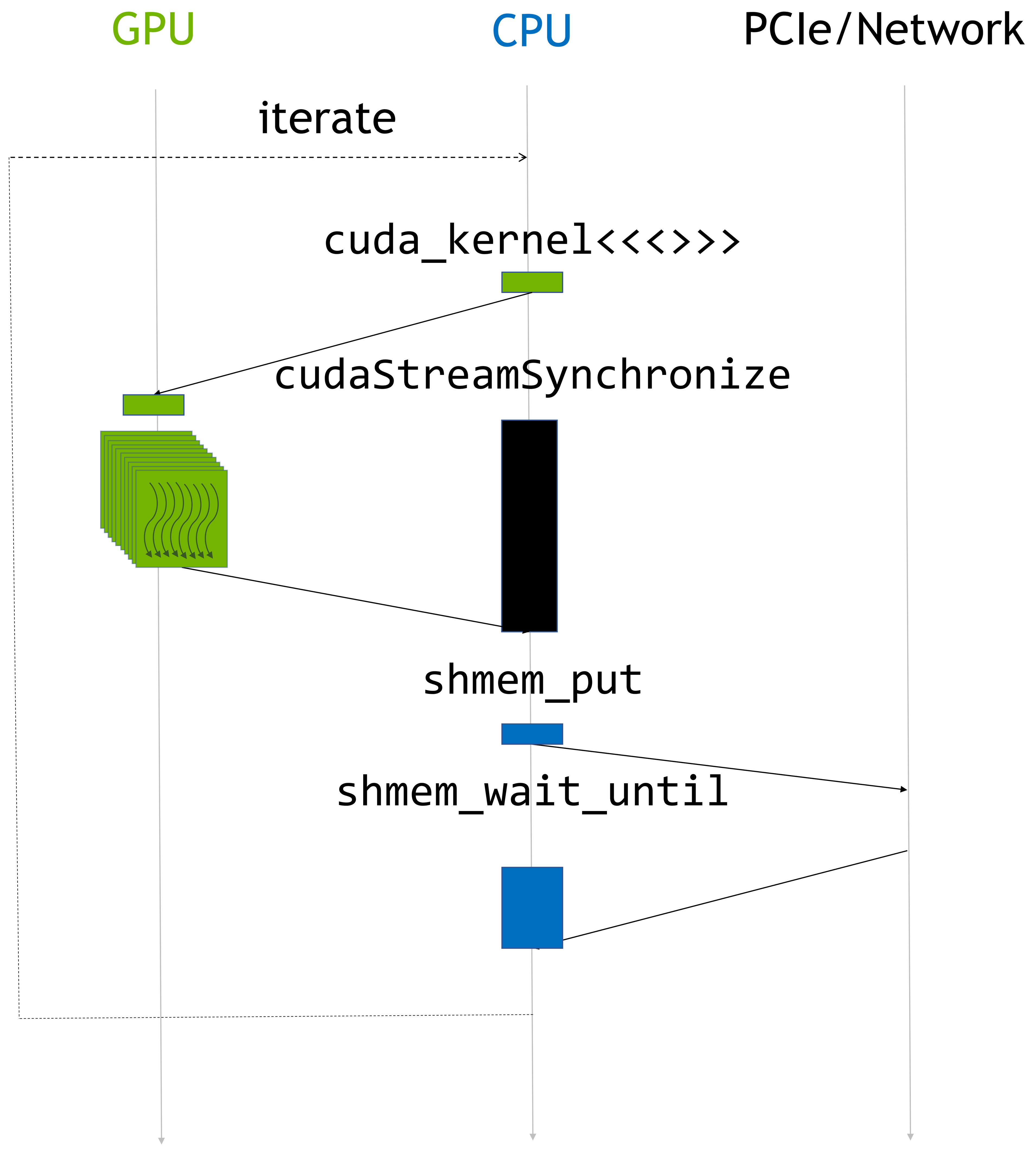
Synchronization at boundaries

Commonly used model, but –

Offload latencies in critical path

Communication is not overlapped

Hiding increased code complexity, not hiding limits strong scaling



GPU-Initiated Communication

Compute on GPU

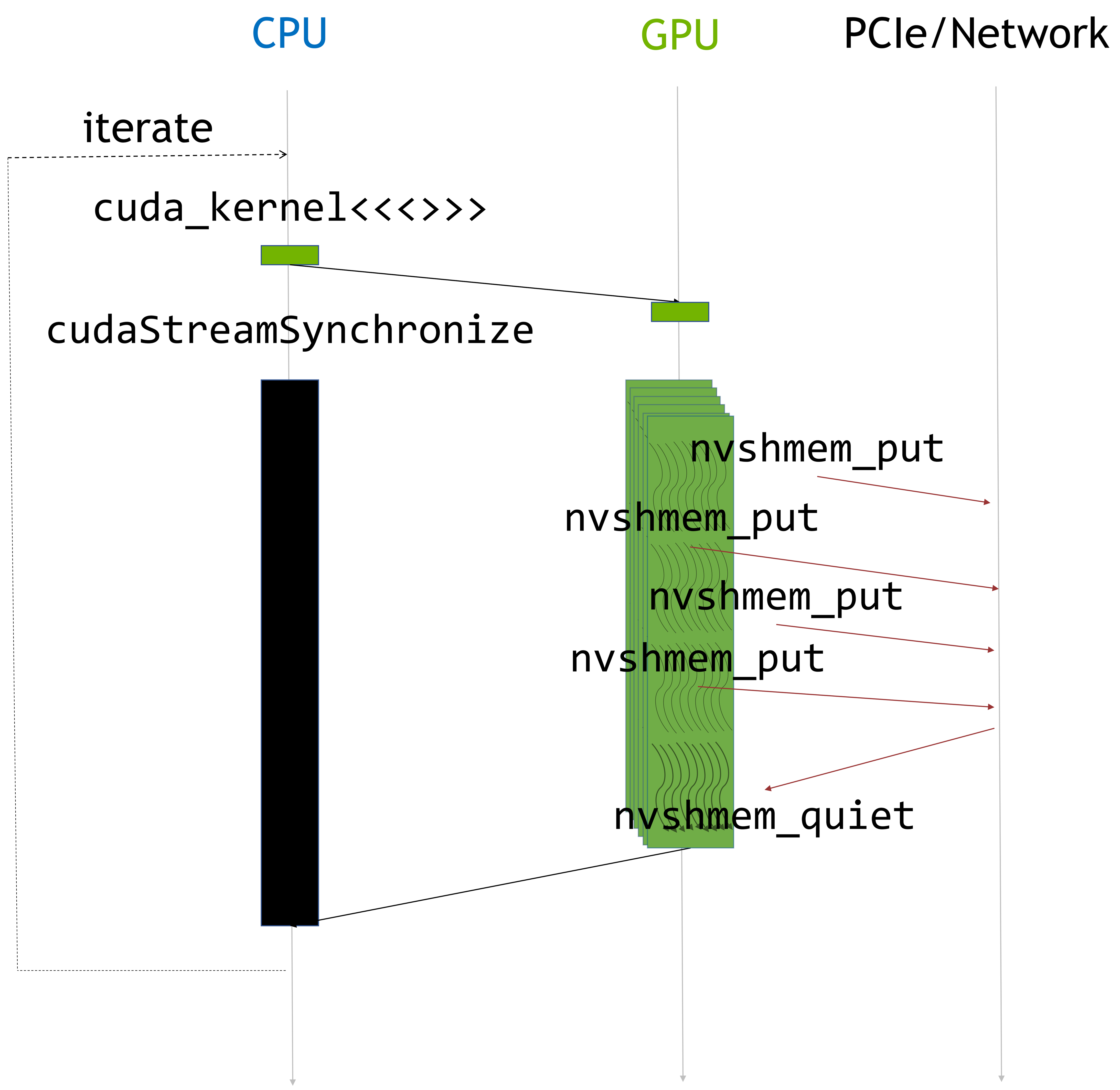
Communication from GPU

Benefits

Eliminates offloads latencies

Compute and communication overlap by threading

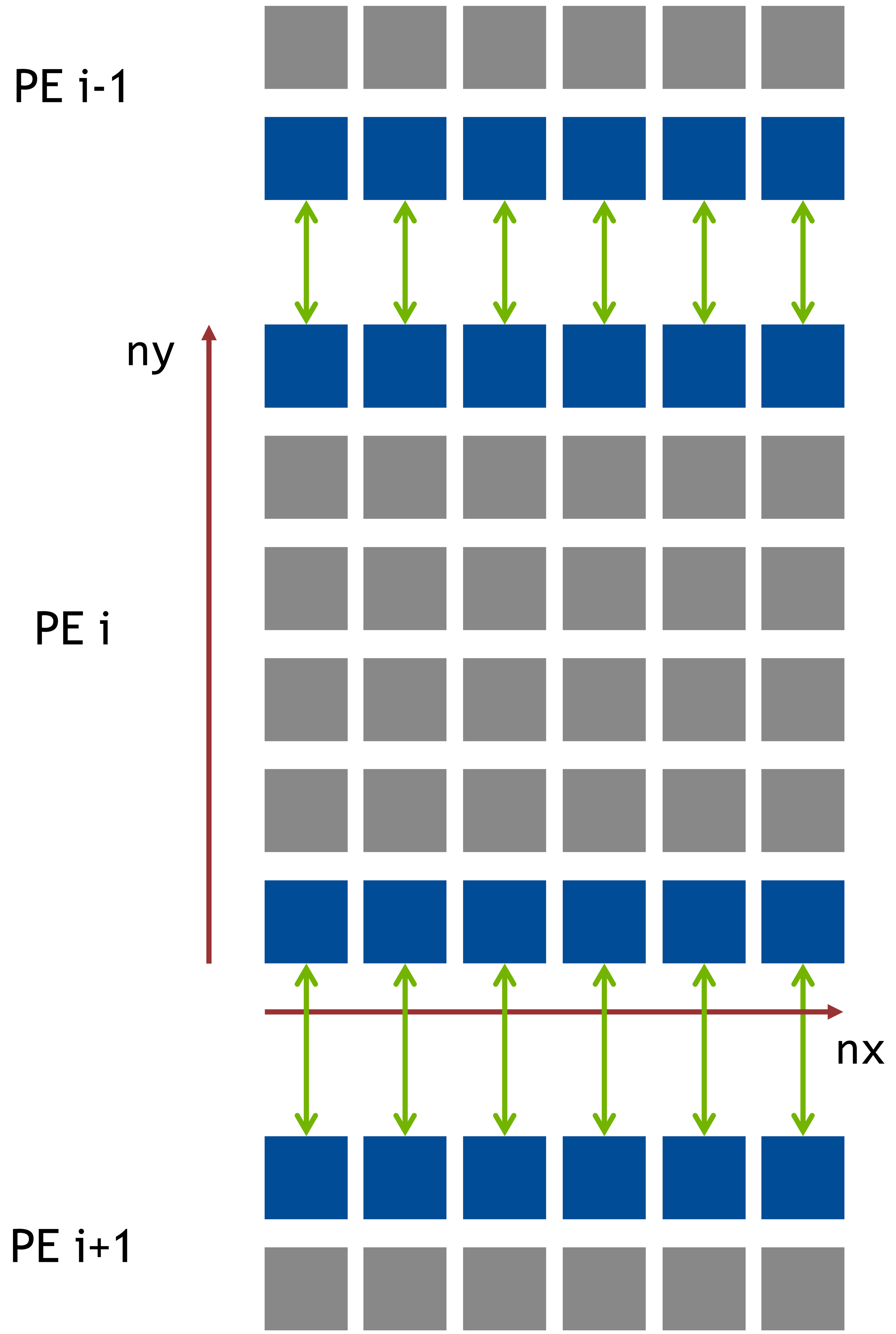
Easier to express algorithms with inline communication



Thread-level communication

- Allows fine grained communication and computation overlap
- Efficient mapping to NVLink fabric on DGX systems

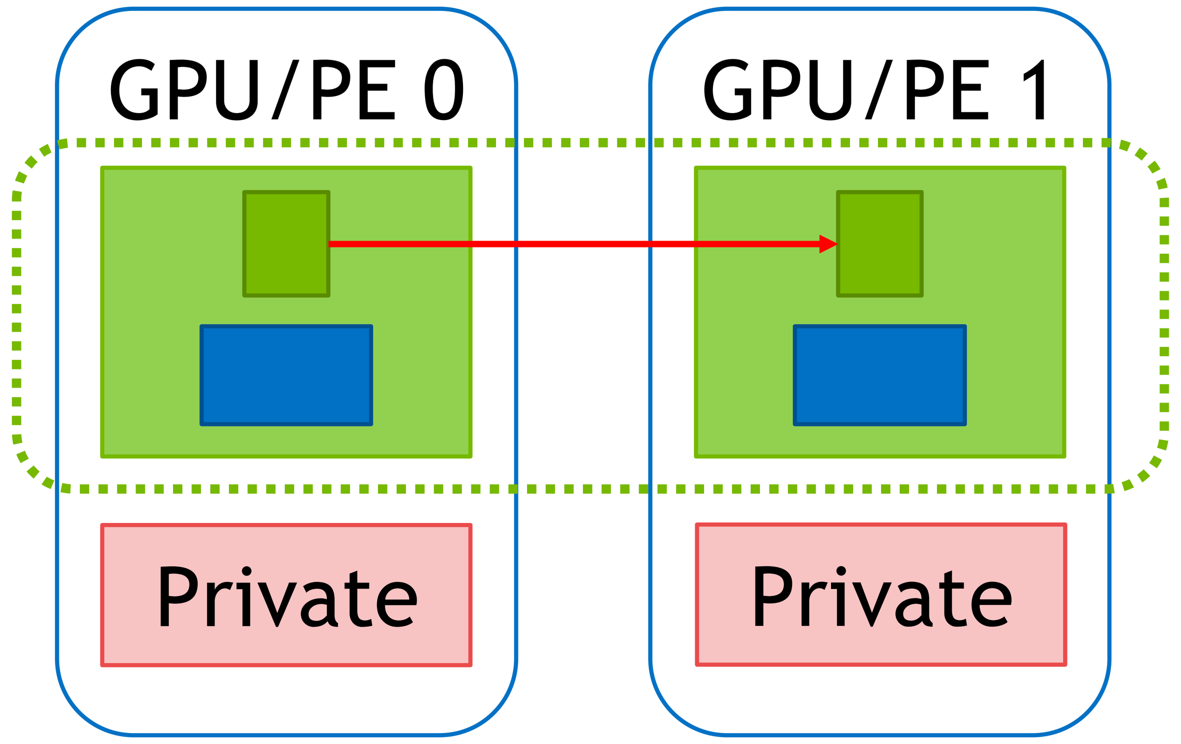
```
__global__ void stencil_single_step(float *u, float *v, …) {
  int ix = get_ix(blockIdx, blockDim, threadIdx);
  int iy = get_iy(blockIdx, blockDim, threadIdx);
  compute(u, v, ix, iy);
  // Thread-level data communication API

  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);
}
```



NVSHMEM API

```
__device__ void nvshmem_TYPENAME_p(TYPE *dest, TYPE value, int pe)
```

- dest [OUT]: Symmetric address of the destination data object.
- value [IN]: The value to be transferred to dest.
- pe [IN]: The number of the remote PE.



https://docs.nvidia.com/nvshmem/api/gen/api/rma.html#c.nvshmem_TYPENAME_p

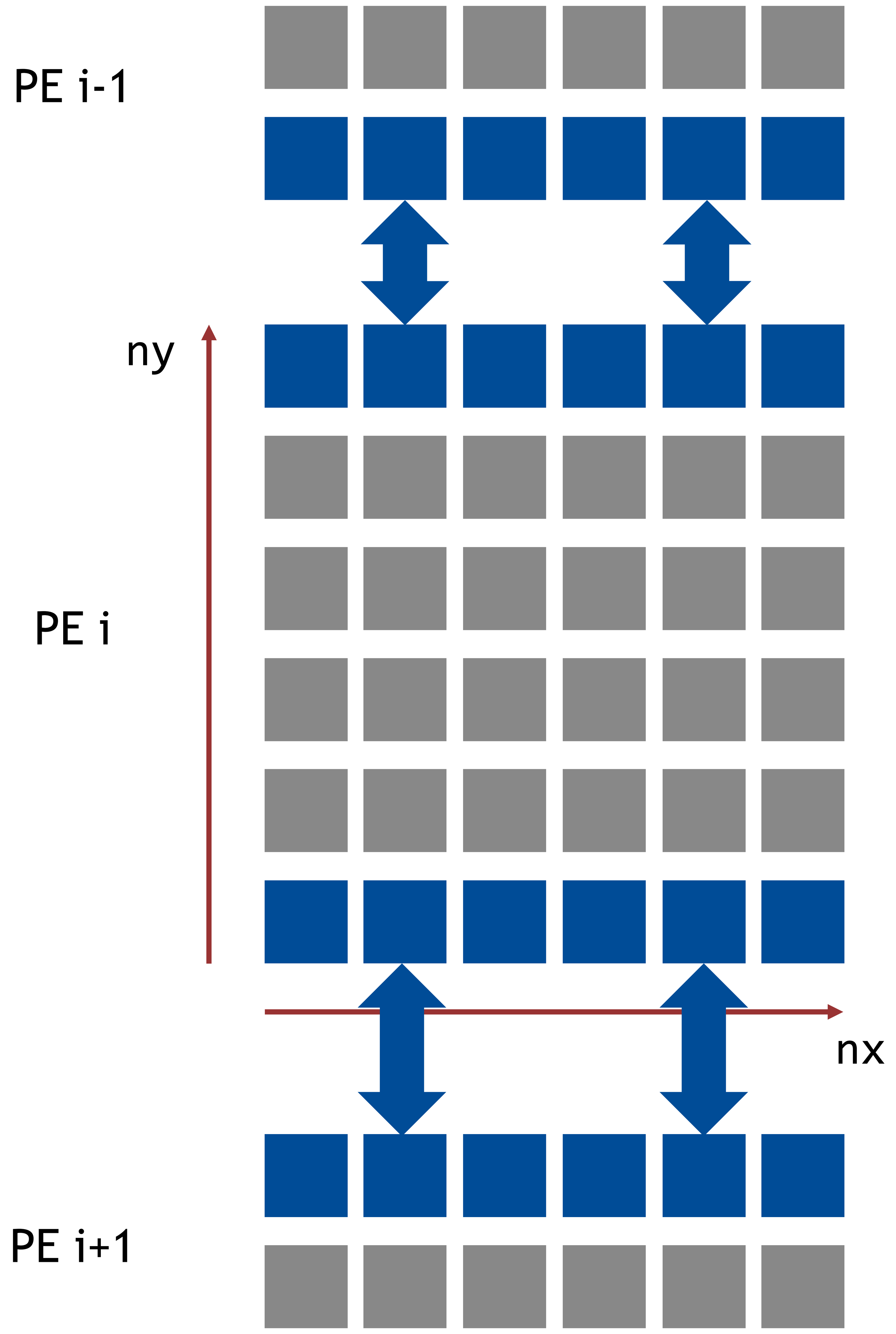
TYPENAME can be: float, double, char, schar, short, int, long, longlong, uchar, ushort, uint, ulong, ulonglong, …, ptrdiff

https://docs.nvidia.com/nvshmem/api/gen/api/rma.html#stdrmatypes

Thread-GROUP communication

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

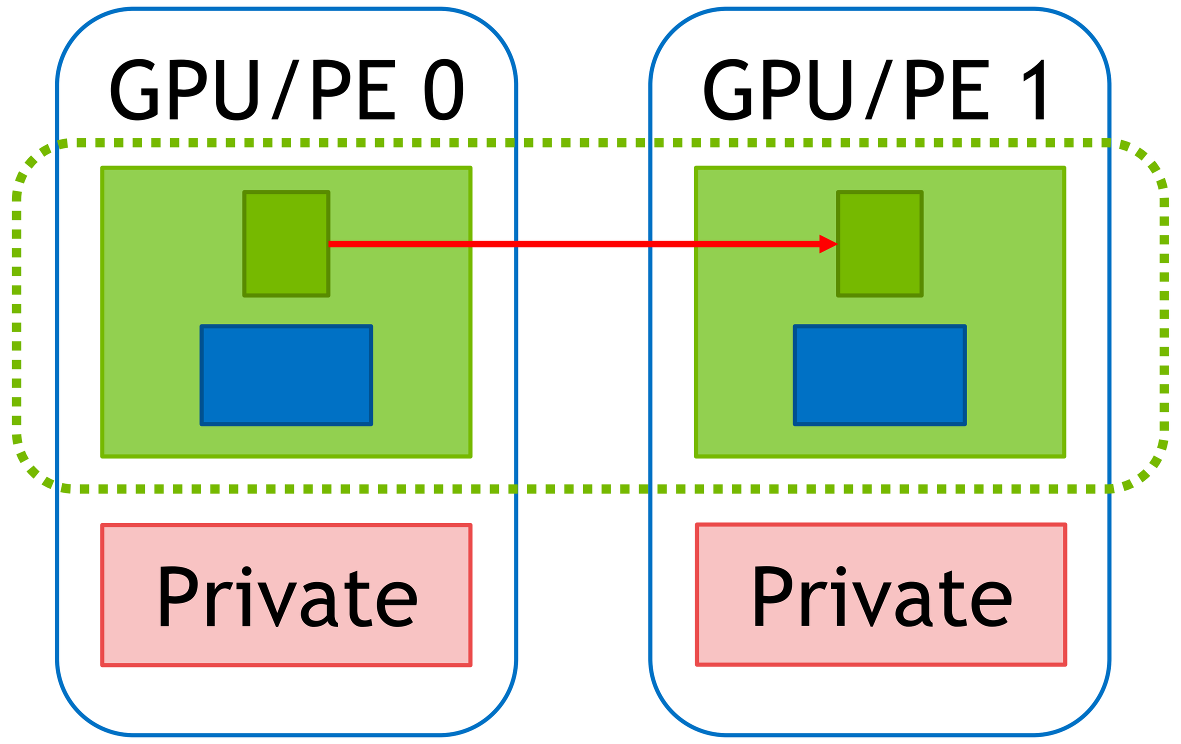
```
__global__ void stencil_single_step(float *   float *v, …) {
  int ix = get_ix(blockIdx  blockDim  threadIdx);
  int iy = get_iy(blockIdx  blockDim  threadIdx);
  compute(u, v, ix, iy);
  // Thread block-level communication API
  int boffset = get_block_offet(blockIdx blockDim);
  if (blockIdx.y == 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);
}
```



NVSHMEM API

```
__device__ void nvshmemx_TYPENAME_put_nbi_block(TYPE *dest, const TYPE *source, size_t nelems, int pe)
```

- dest [OUT]: Symmetric address of the destination data object.
- source [IN]: Symmetric address of the object containing the data to be copied.
- nelems [IN]: Number of elements in the dest and source arrays.
- pe [IN]: The number of the remote PE.



Cooperative call: Needs to be called by all threads in a block. thread and warp are also available.

x in nvshmem**x** marks API as extension of the OpenSHMEM APIs.

https://docs.nvidia.com/nvshmem/api/gen/api/rma.html#c.nvshmemx_TYPENAME_put_nbi_block

`TYPENAME` can be: `float`, `double`, `char`, `schar`, `short`, `int`, `long`, `longlong`, `uchar`, `ushort`, `uint`, `ulong`, `ulonglong`, …, `ptrdiff`

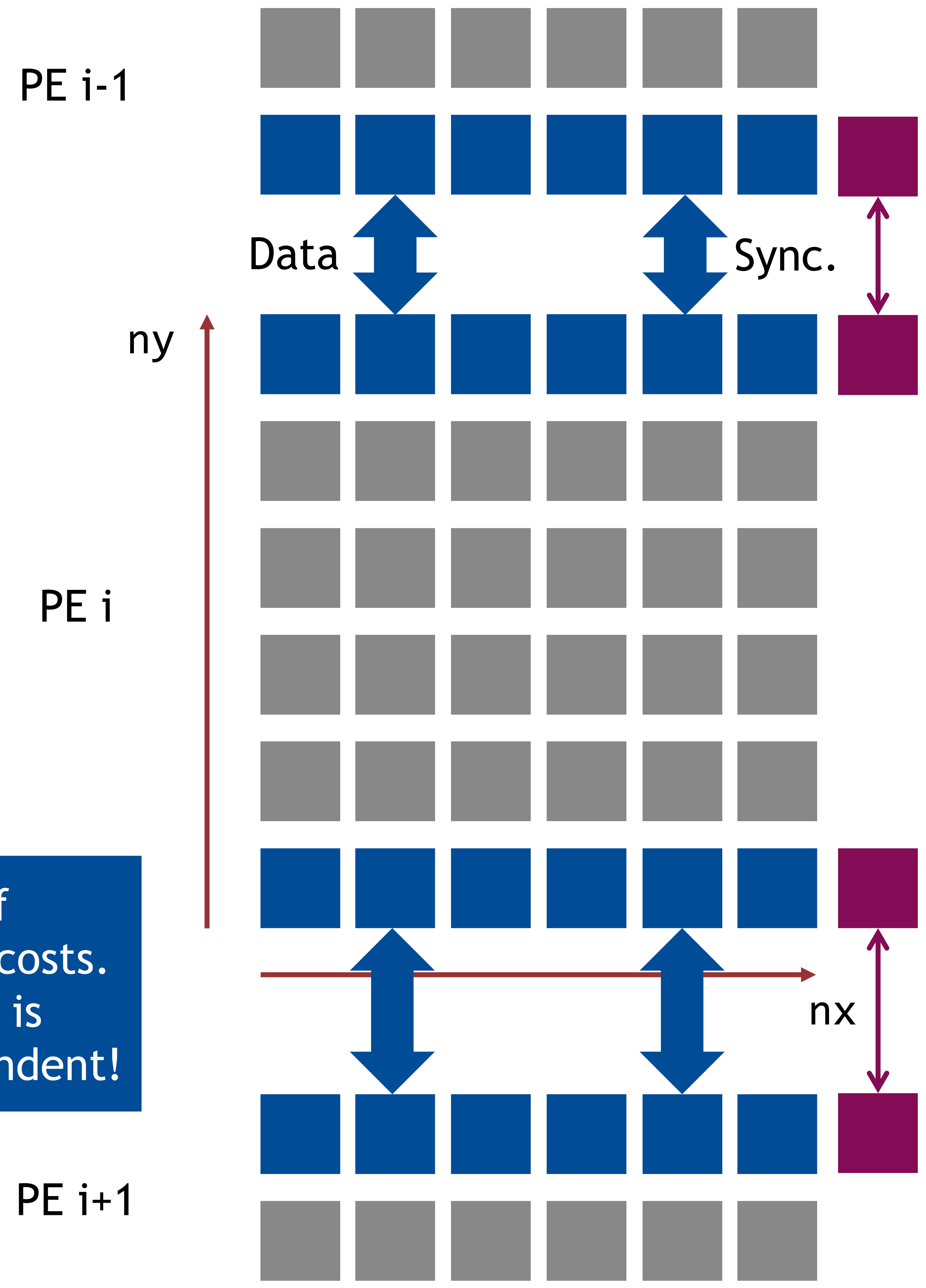
https://docs.nvidia.com/nvshmem/api/gen/api/rma.html#stdrmatypes

In-Kernel Synchronization

- Point-to-point synchronization across PEs within a kernel
- Enables kernel fusion

```
__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+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);
    if (blockIdx.y == 0 || blockIdx.y == (blockDim.y-1)) {
      __syncthreads();
      nvshmem_quiet();
      if (threadIdx.x == 0 && threadIdx.y == 0) {
        nvshmem_atomic_inc(sync, top_pe);
        nvshmem_atomic_inc(sync, bottom_pe);
    }}
    nvshmem_wait_until(sync, NVSHMEM_CMP_GT, 2*iter*gridDim.x);
}}
```

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



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 requirement of 1PE/GPU

NVSHMEM API

```
__device__ void nvshmem_quiet(void)
```

Ensures completion of all operations on symmetric data objects issued by the calling PE.

https://docs.nvidia.com/nvshmem/api/gen/api/ordering.html?highlight=quiet#c.nvshmem_quiet

NVSHMEM API

```
__device__ void nvshmem_TYPENAME_atomic_inc(TYPE *dest, int pe)
```

- dest [OUT]: Symmetric address of the signal word to be updated.
- pe [IN]: The number of the remote PE.

These routines perform an atomic increment operation on the dest data object on PE.

https://docs.nvidia.com/nvshmem/api/gen/api/amo.html?highlight=atomic_inc#c.nvshmem_TYPENAME_atomic_inc

TYPENAME can be: float, double, char, schar, short, int, long, longlong, uchar, ushort, uint, ulong, ulonglong, …, ptrdiff

https://docs.nvidia.com/nvshmem/api/gen/api/rma.html#stdrmatypes

NVSHMEM API

```
__device__ void nvshmem_TYPENAME_wait_until_all(TYPE *ivars, size_t nelems, const int *status,
                                                 int cmp, TYPE cmp_value)

__device__ void nvshmem_TYPENAME_wait_until(TYPE *ivar  int cmp, TYPE cmp_value)
```

- ivars[N]: Symmetric address of an array of remotely accessible data objects..
- nelems [IN]: The number of elements in the ivars array.
- status [IN]: Local address of an optional mask array of length nelems that indicates which elements in ivars are excluded from the wait set. Set to NULL when not used.
- cmp [IN]: A comparison operator (NVSHMEM_CMP_EQ, NVSHMEM_CMP_NE, NVSHMEM_CMP_GT, NVSHMEM_CMP_GE, NVSHMEM_CMP_LT, NVSHMEM_CMP_LE) that compares elements of ivars ivar with cmp_value.
- cmp_value [IN]: The value to be compared with the objects pointed to by ivars.

https://docs.nvidia.com/nvshmem/api/gen/api/sync.html?highlight=wait_until#c.nvshmem_TYPENAME_wait_until

TYPENAME can be: float, double, char, schar, short, int, long, longlong, uchar, ushort, uint, ulong, ulonglong, …, ptrdiff

https://docs.nvidia.com/nvshmem/api/gen/api/rma.html#stdrmatypes

NVSHMEM API

```
__device__ inline void nvshmemx_signal_op(uint64_t *sig_addr, uint64_t signal, int sig_op, int pe)
```

- sig_addr [OUT]: Symmetric address of the signal word to be updated.
- signal [IN]: The value used to update sig_addr.
- sig_op [IN]: Operation used to update sig_addr with signal. (NVSHMEM_SIGNAL_SET or NVSHMEM_SIGNAL_ADD)
- pe [IN]: The number of the remote PE.

x in nvshmem**x** marks API as extension of the OpenSHMEM APIs.

https://docs.nvidia.com/nvshmem/api/gen/api/signal.html?highlight=signal_op#c.nvshmemx_signal_op

NVSHMEM perftests

shmem_p_bw|shmem_put_bw from NVSHMEM perftest
JUWELS Booster: https://apps.fz-juelich.de/jsc/hps/juwels/booster-overview.html

NVSHMEM perftests

shmem_p_bw|shmem_put_bw from NVSHMEM perftest
JUWELS Booster: https://apps.fz-juelich.de/jsc/hps/juwels/booster-overview.html

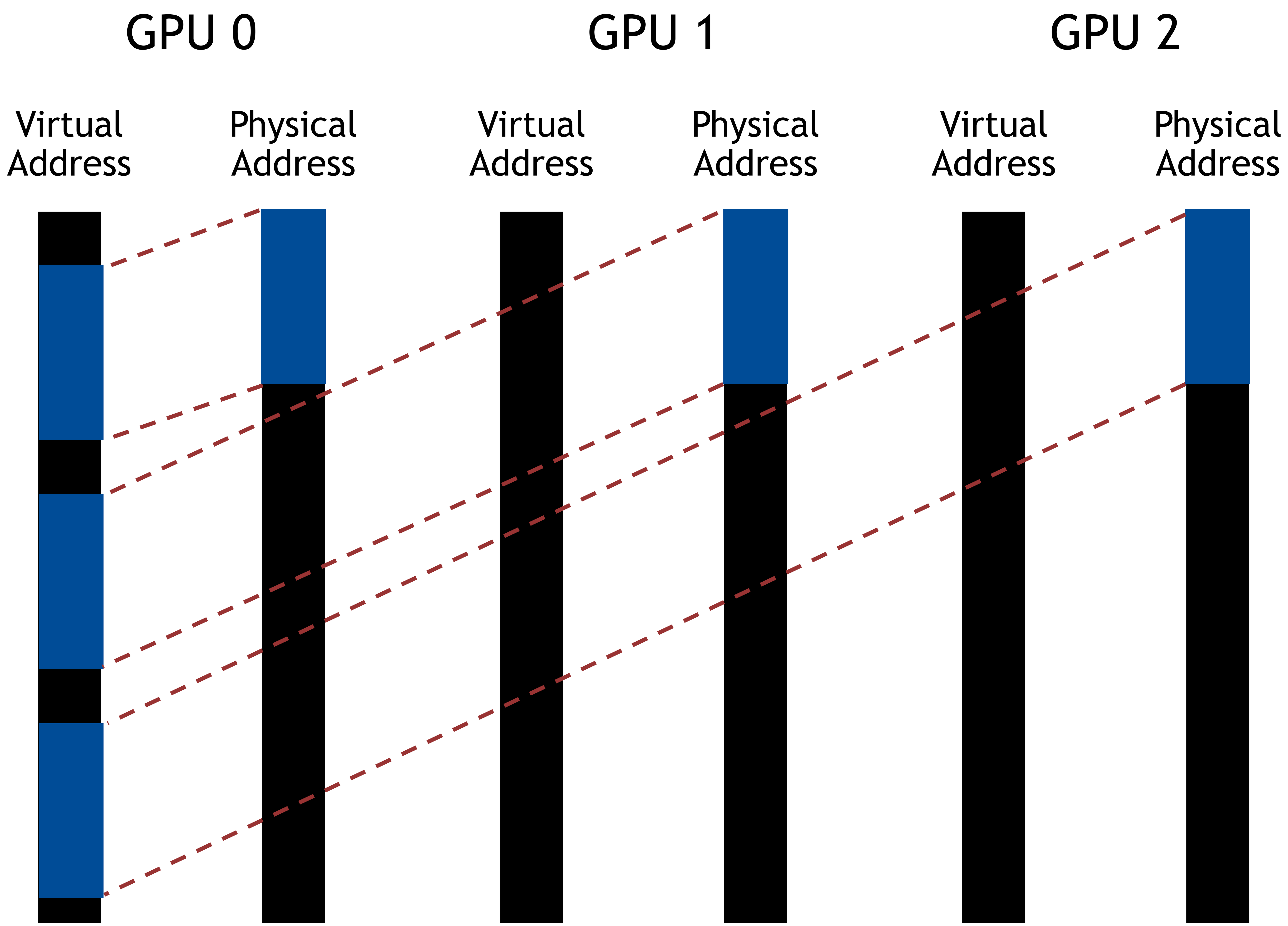
Optimized Intra-Node Communication

Supported on NVLink and PCI-E

Use CUDA IPC or `cuMem*` API to map symmetric memory of intra-node PEs into virtual address space

`nvshmem_[put|get]` on device -> load/store

`nvshmem_[put|get]_on_stream` -> `cudaMemcpyAsync`



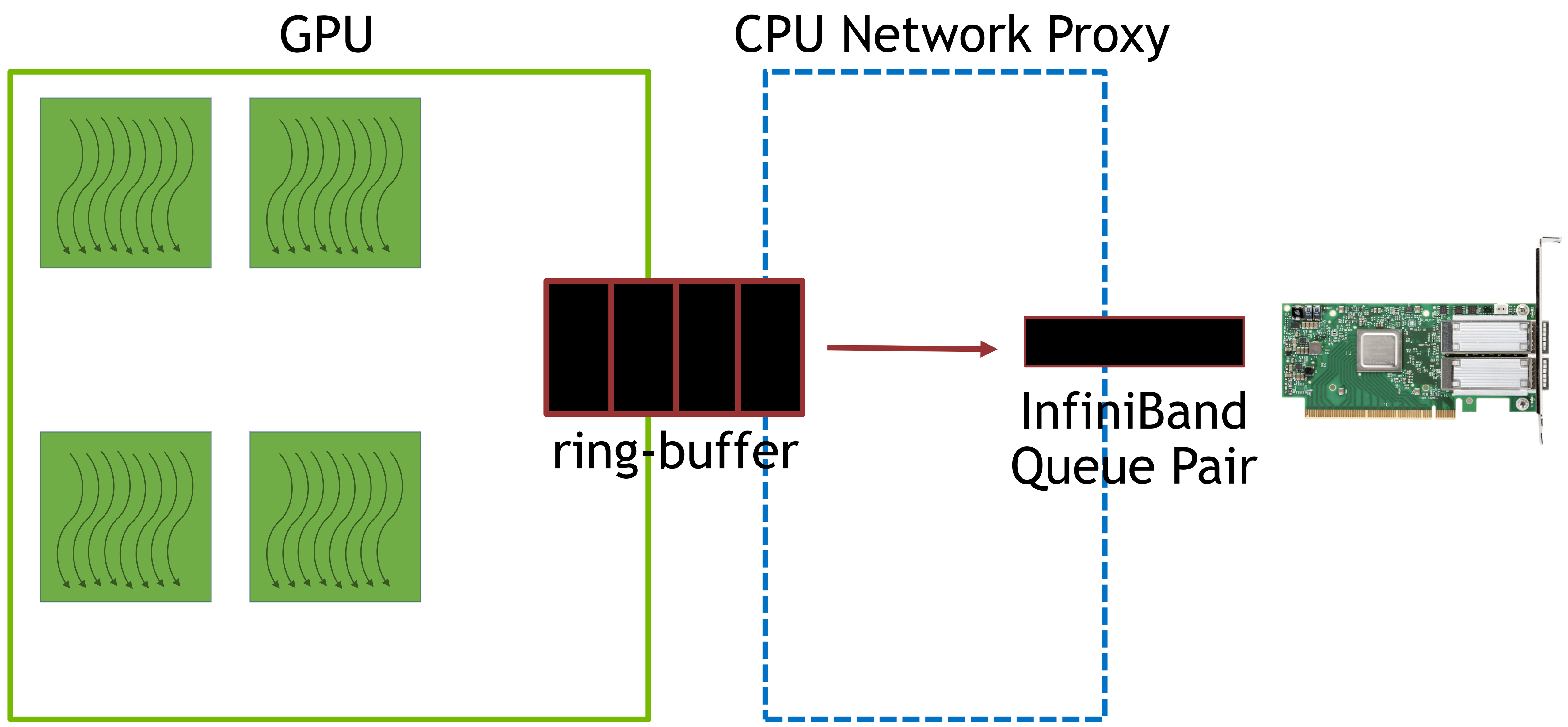
Optimized Inter-Node Communication

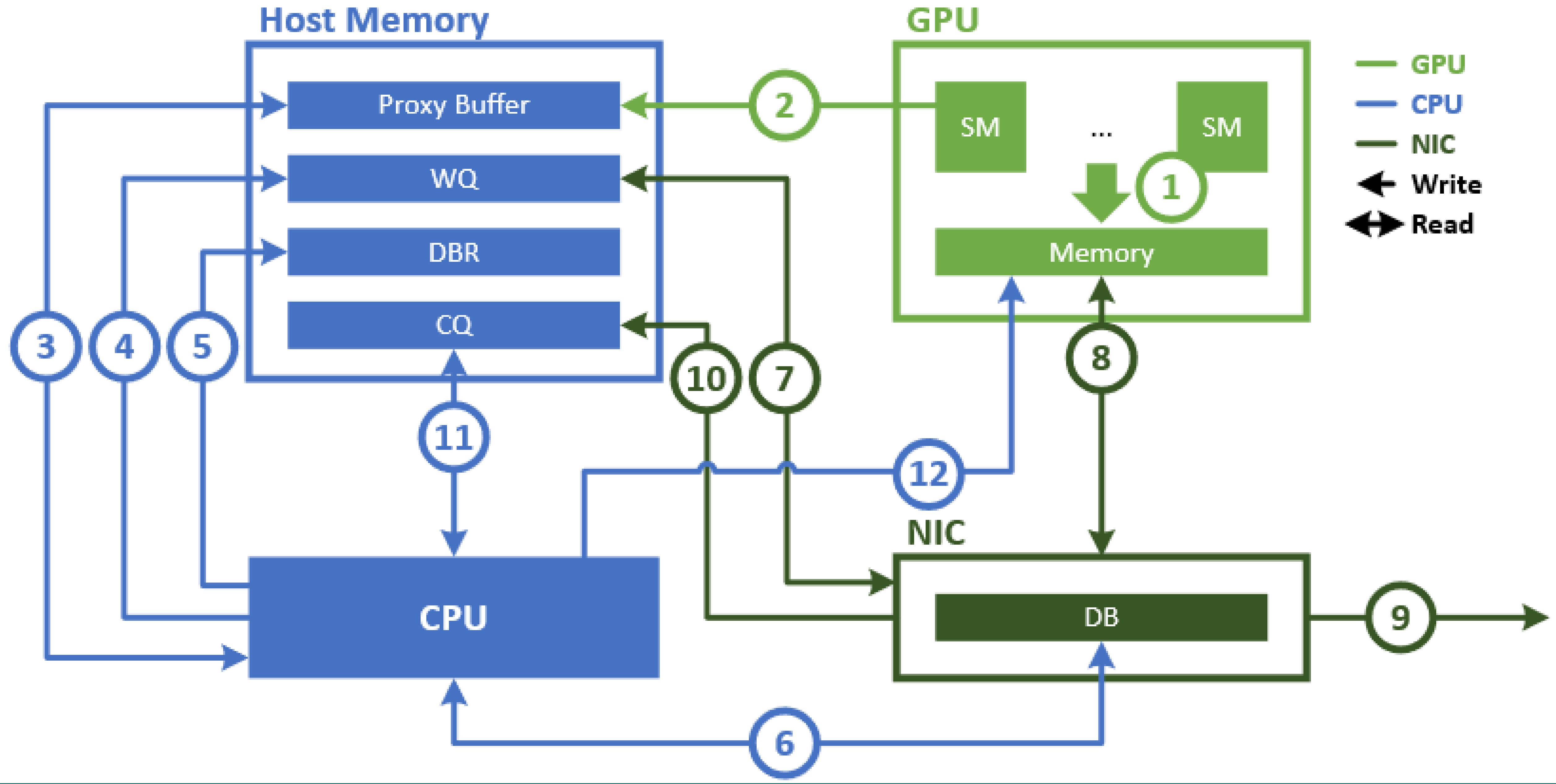
NVSHMEM supports inter-node communication over InfiniBand, RoCE, and UCX (experimental)

Using GPUDirect RDMA (data plane)

Reverse offloads network transfers from GPU to the CPU (control plane)

Ring buffer implementation avoids memory fences when interacting with CPU network proxy



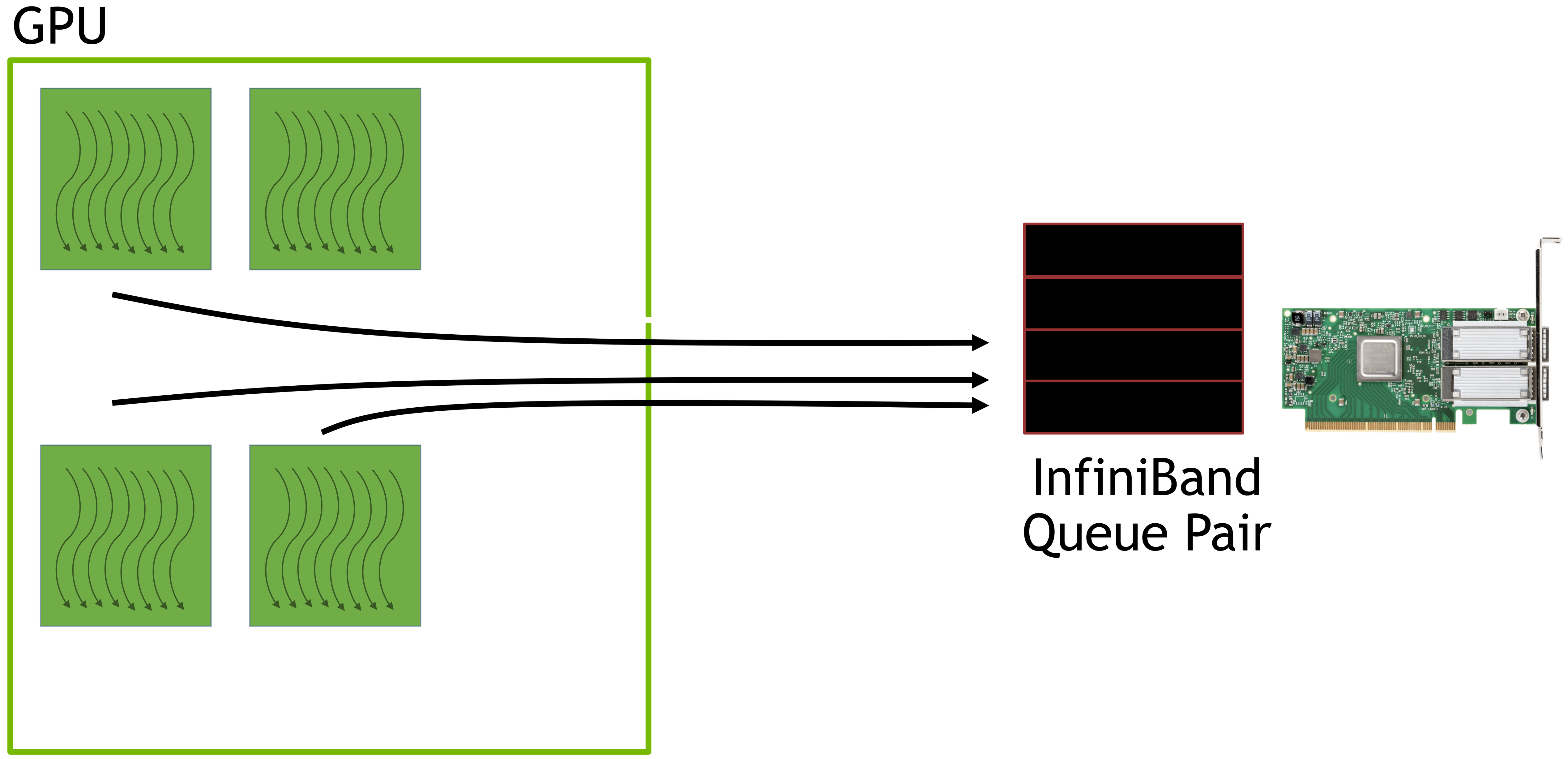


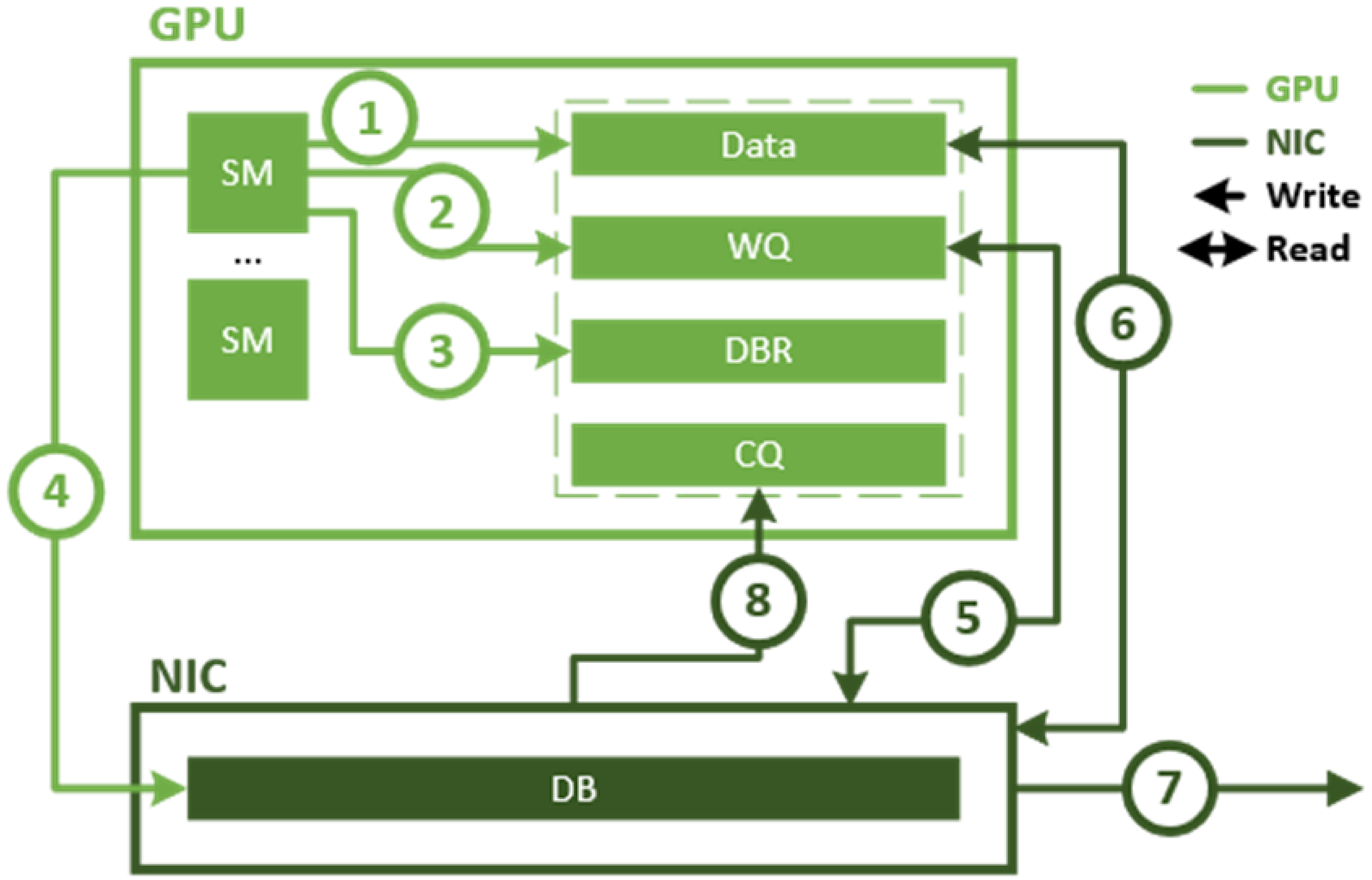
Optimized Inter-Node Communication Improved

IB GPUDirect **A**sync (IBGDA) over InfiniBand

Using GPUDirect RDMA (data plane)

GPU directly initiates **network transfers** involving the CPU only for the setup of control data structures





NVSHMEM perftests with IBGDA

shmem_p_bw|shmem_put_bw from NVSHMEM perftest
JUWELS Booster: https://apps.fz-juelich.de/jsc/hps/juwels/booster-overview.html

NVSHMEM Version with NVL, CPU Proxy and IBGDA

Source: https://github.com/NVIDIA/multi-gpu-programming-models
JUWELS Booster: https://apps.fz-juelich.de/jsc/hps/juwels/booster-overview.html

QUDA Strong Scaling on Selene

https://www.nvidia.com/en-us/on-demand/session/gtcsj20-s21673/

up to 1.6x Speedup (512 GPUs half precision)

Summary and More Information

- CUDA Graphs help minimize CPU-side launch overhead and Device-side execution overhead
- Device-initiate communication enables:
  - fine grained communication and computation overlap with sometimes less coding effort
  - kernel fusion not possible with host initiate communication models like MPI and NCCL
- With **IB G**PU**D**irect **A**sync (IBGDA) NVSHMEM can achieve peak Network message rates
- Without IBGDA for good intranode device-initiated communication performance it is necessary to aggregate larger messages (nvshmemx_TYPENAME_put_nbi_block)

- CUDA Graphs documentation https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#cuda-graphs
- NVSHMEM: CUDA-Integrated Communication for NVIDIA GPUs (a Magnum IO session): https://www.nvidia.com/en-us/on-demand/session/gtcspring22-s41044/

Overcoming Latency Barriers: Strong Scaling HPC Applications with NVSHMEM:

- https://www.nvidia.com/en-us/on-demand/session/gtcsj20-s21673/
- https://developer.nvidia.com/blog/scaling-scientific-computing-with-nvshmem/
- https://developer.nvidia.com/blog/accelerating-nvshmem-2-0-team-based-collectives-using-nccl/