S51111: Scaling Deep Learning Training: Fast Inter-GPU Communication with NCCL

Sylvain Jeaugey | GTC 2023

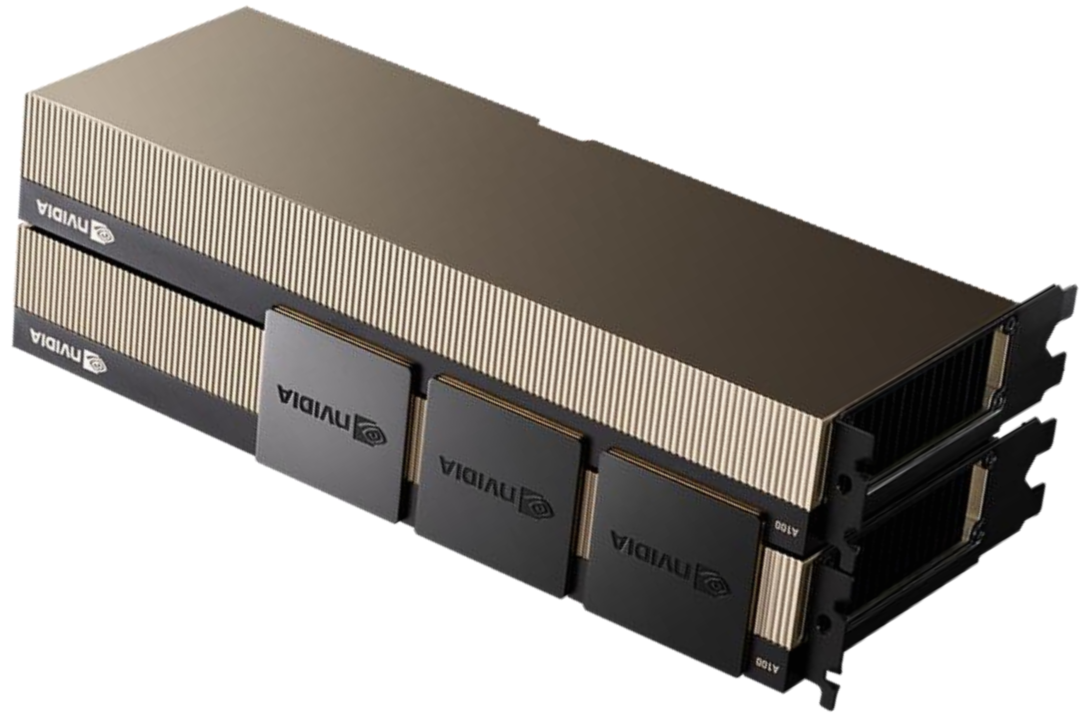
Multi-GPU Computing

NCCL : Key communication library for multi-GPU computing.

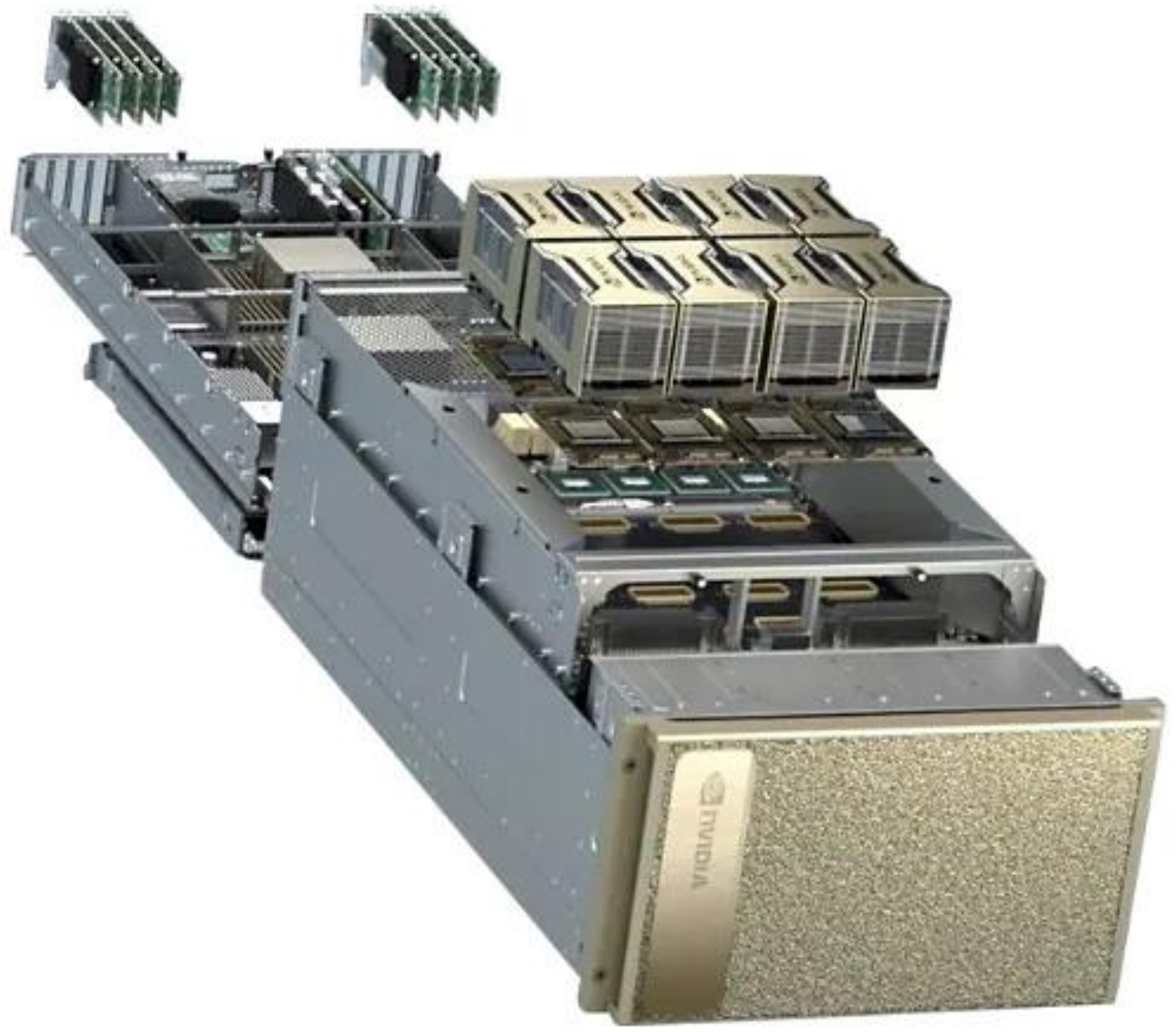
Optimized for all platforms, from desktop to DGX Superpod.

PCI

NVLink



NVSwitch + Network



Download from https://developer.nvidia.com/nccl and in NGC containers.
Source code at https://github.com/nvidia/nccl

Agenda

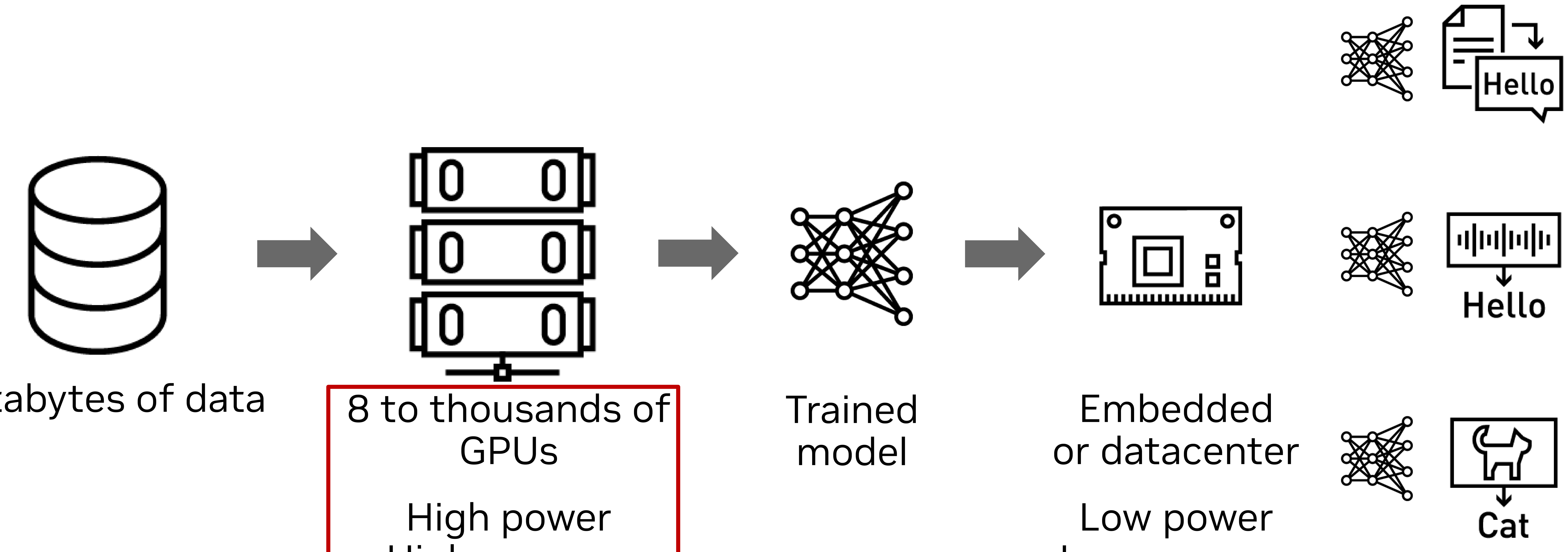
- Deep Learning Training
- NCCL Overview
- Collective Operations
- Topology detection
- Point-to-point Communication
- New and Future

Agenda

- Deep Learning Training
- NCCL Overview
- Collective Operations
- Topology detection
- Point-to-point Communication
- New and Future

Deep Learning

From data to AI



Petabytes of data

8 to thousands of GPUs

High power
High memory

Training

1 hour to 1 week

NCCL : make multi-GPU training as efficient as possible to reduce training time.

Trained model

Embedded or datacenter

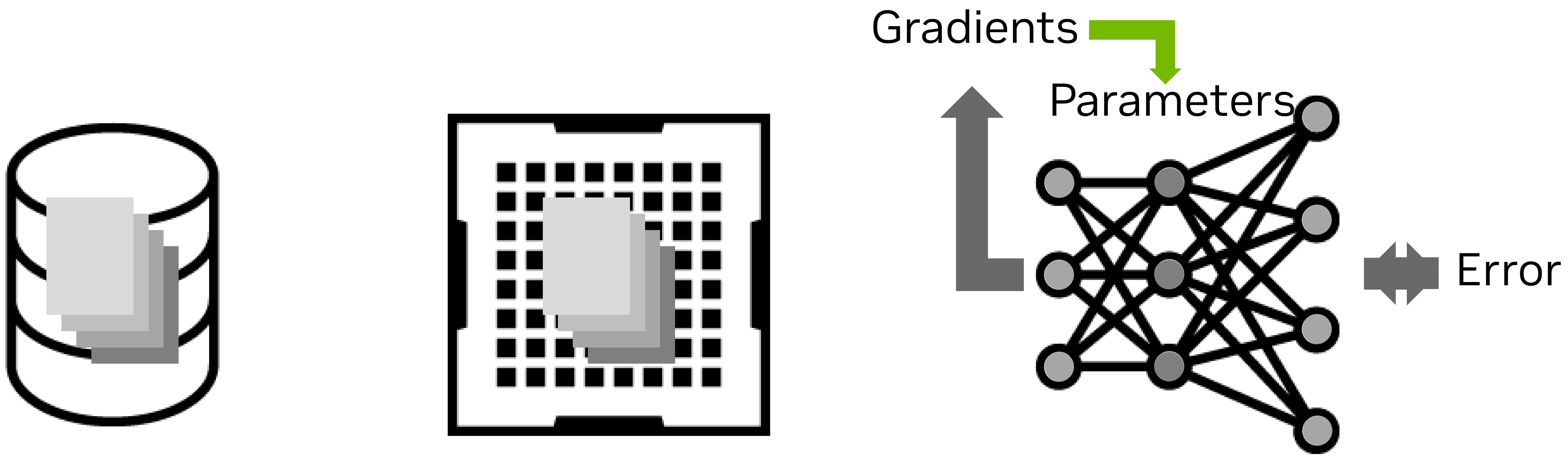
Low power
Low memory

Inference

1 ms to 1 s

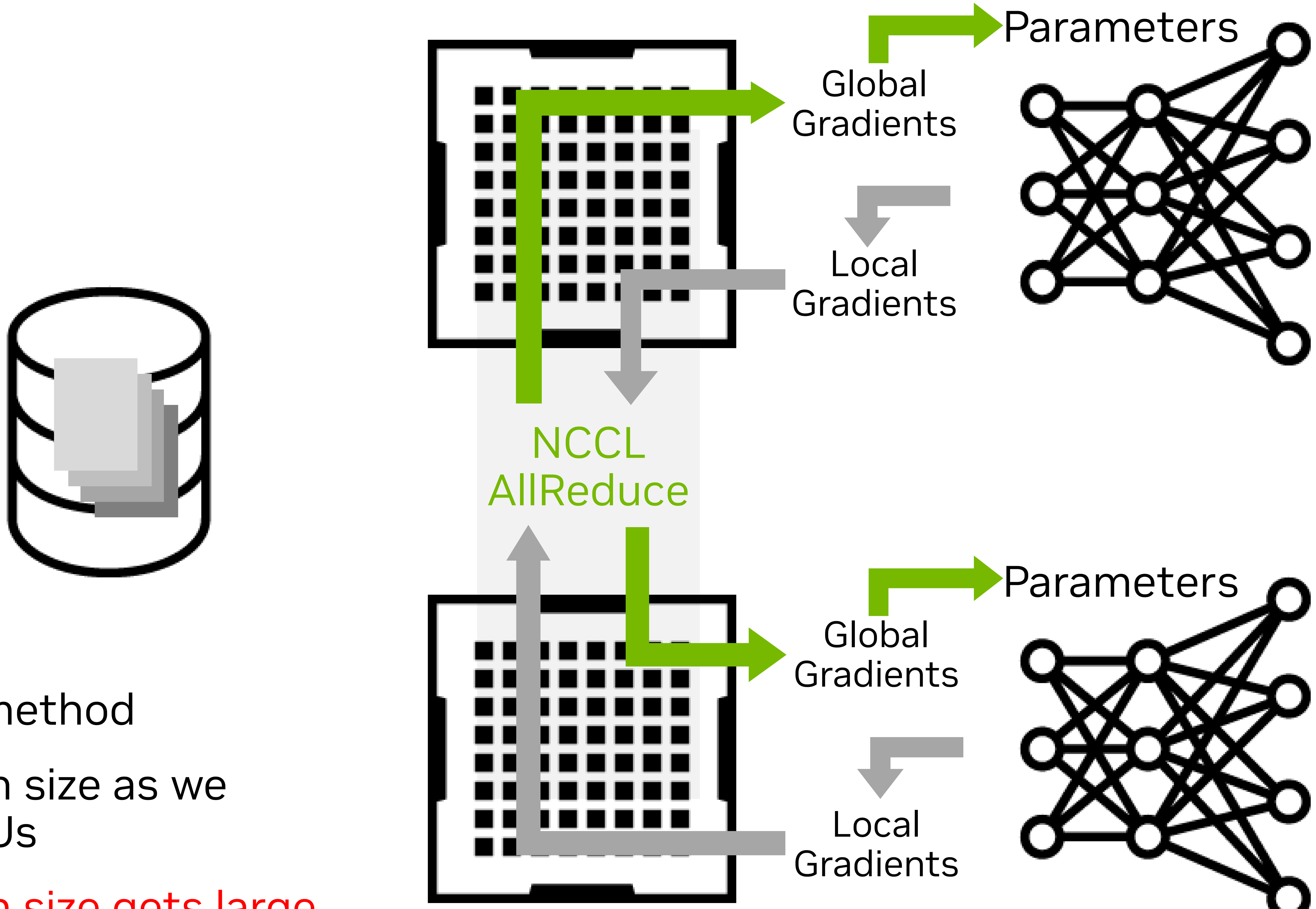
Deep Learning Training

On a single GPU



Deep Learning Training

Data parallelism



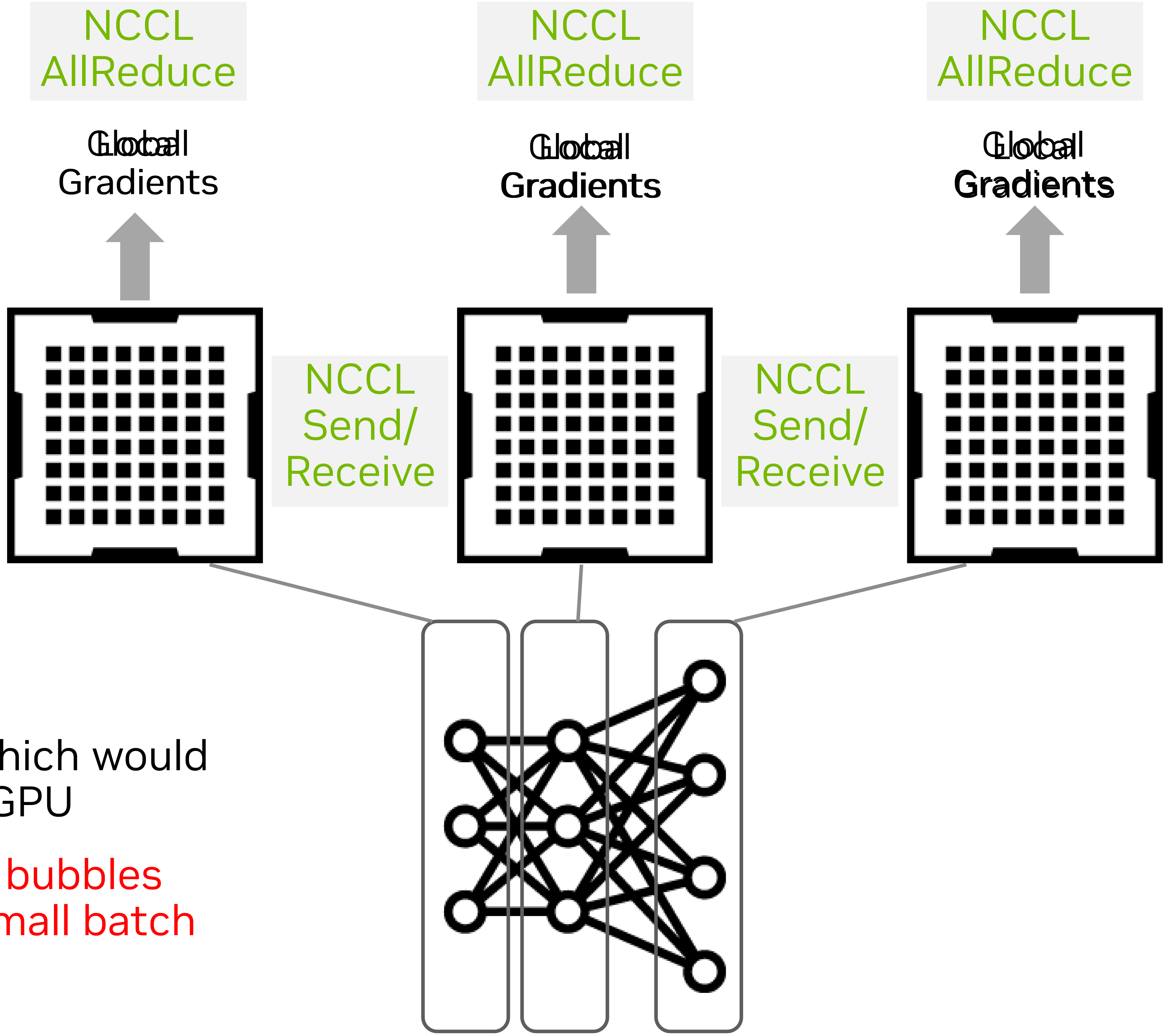
Most common parallelism method

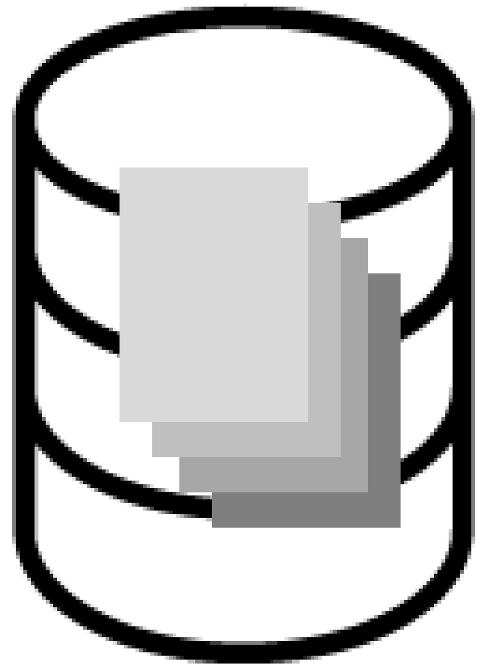
Needs to increase the batch size as we increase the number of GPUs

Accuracy may drop as batch size gets large

Deep Learning Training

Pipeline parallelism



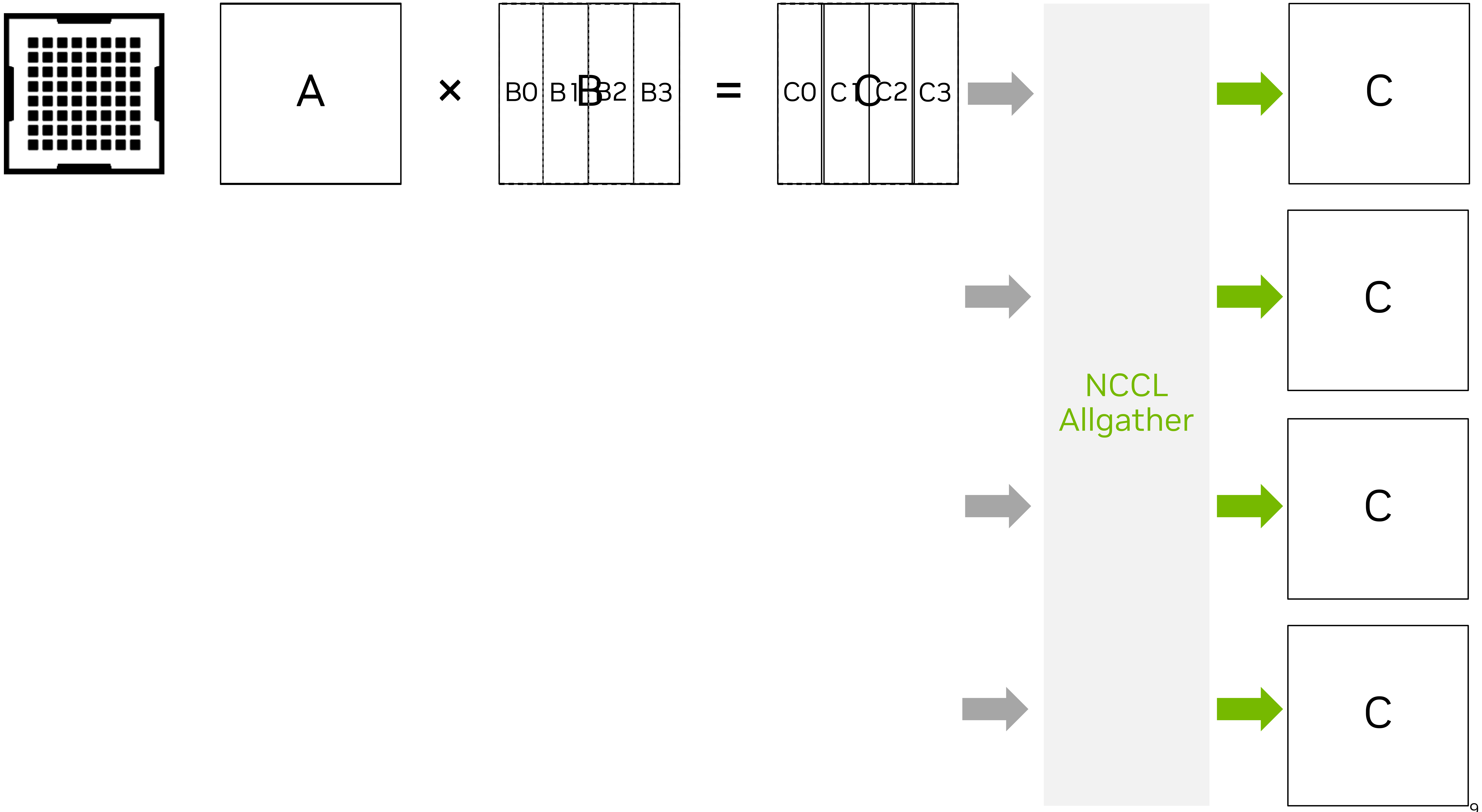


Allows to run larger models which would not fit in the memory of one GPU

The send/receive pipeline has bubbles which can be inefficient for small batch sizes.

Deep Learning Training

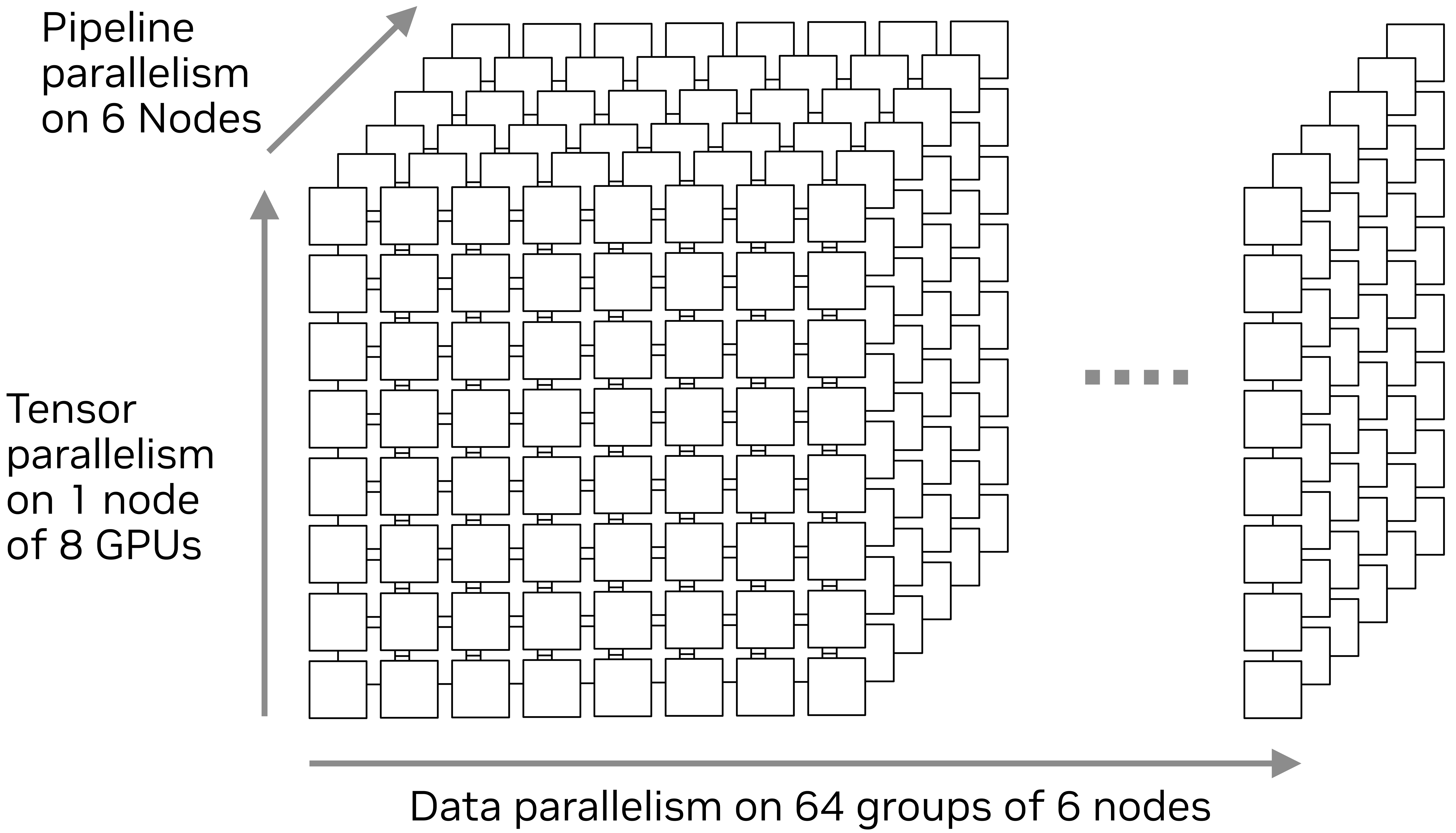
Tensor parallelism



Deep Learning Training

Megatron / GPT

Goal: train GPT on 1 trillion parameters



502 Petaflops

3072 GPUs

[Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM, Deepak Narayanan et al., 2021]

Agenda

- Deep Learning Training
- NCCL Overview
- Collective Operations
- Topology detection
- Point-to-point Communication
- New and Future

NCCL Overview

NCCL provides primitives for GPU-to-GPU communication, optimized for all platforms.

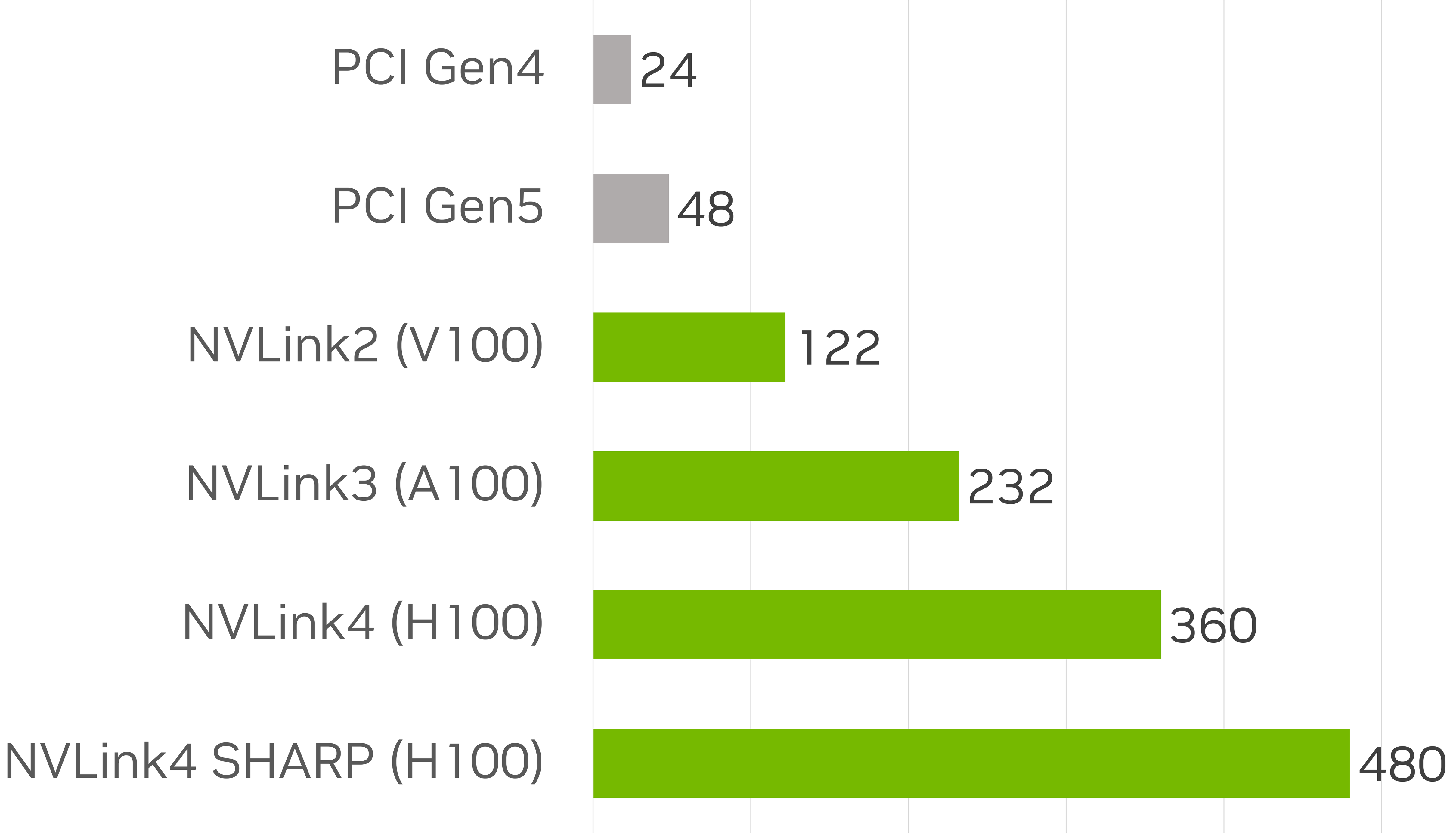
It is implemented as a library launching CUDA kernels on the GPU, interleaved with compute kernels via CUDA stream semantics.

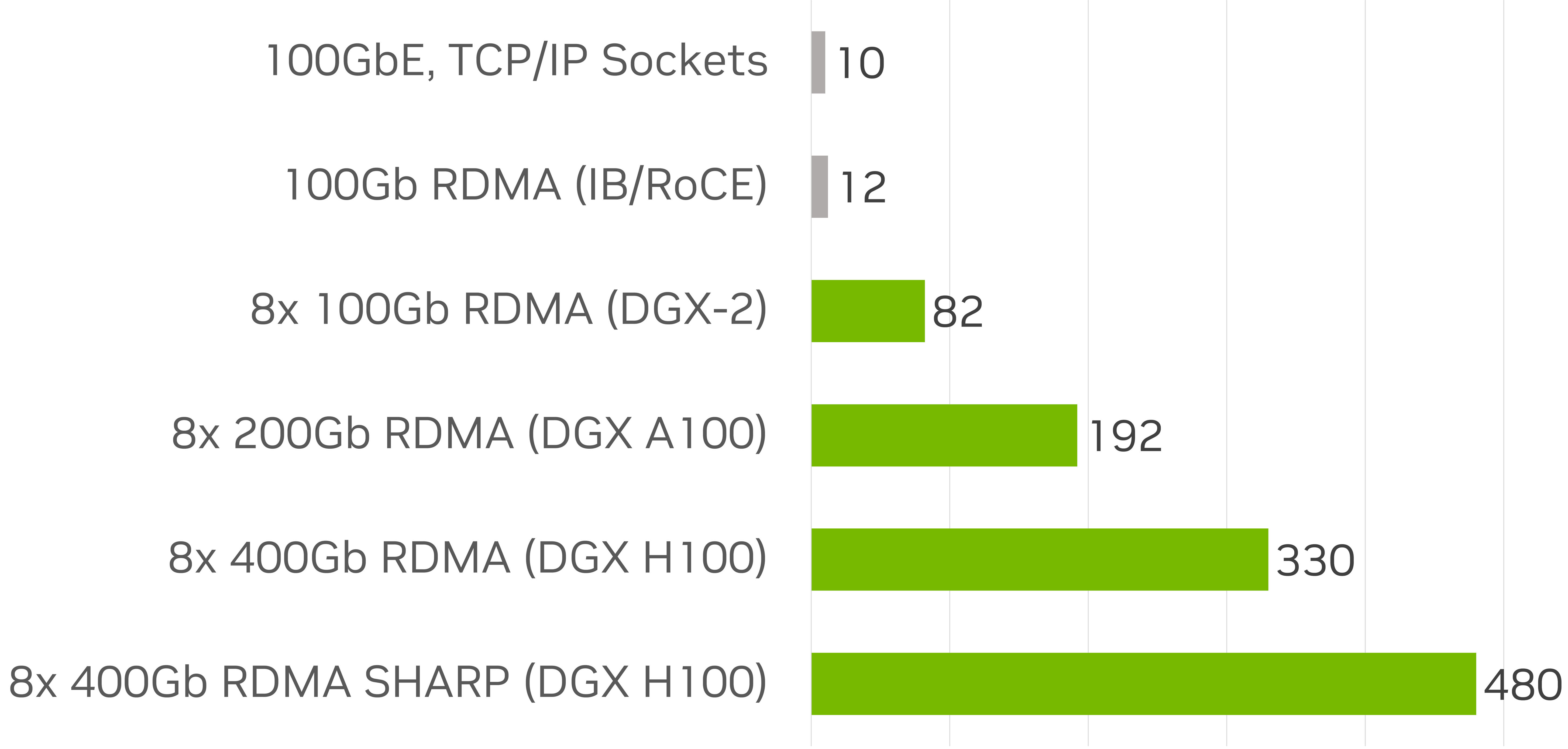
Initialization	Hardware topology detection	Collective communication algorithms	Point-to-point communication
CPU-side communication Socket-based out-of-band bootstrap Fault tolerance	Graph with GPUs, CPUs, NICs, PCI switches, NVLinks and NVSwitches. Graph search for collective algorithms.	Ring, Tree, Collnet (Chain/Direct), NVLink SHARP.	Send/Receive semantics Communication schedule Aggregation
GTC '21	GTC '20	GTC '22	GTC '22

NCCL API

```
// Initialization, finalize, abort and fault tolerance
ncclResult_t ncclGetUniqueId(ncclUniqueId* uniqueId);                                                          // Since 2.0
ncclResult_t ncclCommInitRank(ncclComm_t* comm, int nranks, ncclUniqueId commId, int rank);                    // Since 2.0
ncclResult_t ncclCommInitRankConfig(ncclComm_t* comm, int nranks, ncclUniqueId commId, int rank, ncclConfig_t* config); // Since 2.14
ncclResult_t ncclCommFinalize(ncclComm_t comm);                                                                // Since 2.14
ncclResult_t ncclCommDestroy(ncclComm_t comm);                                                                 // Since 2.0
ncclResult_t ncclCommAbort(ncclComm_t comm);                                                                   // Since 2.4
ncclResult_t ncclCommGetAsyncError(ncclComm_t comm, ncclResult_t *asyncError);                                 // Since 2.4
ncclResult_t ncclCommSplit(ncclComm_t comm, int color, int key, ncclComm_t *newcomm, ncclConfig_t* config);    // Planned for 2.18

// Collective communication - since 2.0
ncclResult_t ncclReduce       (const void* sbuff, void* rbuff, size_t cnt,  ncclDataType_t dtype, ncclRedOp_t op, int root, ncclComm_t comm, cudaStream_t s);
ncclResult_t ncclBroadcast    (const void* sbuff, void* rbuff, size_t cnt,  ncclDataType_t dtype,                 int root, ncclComm_t comm, cudaStream_t s);
ncclResult_t ncclAllReduce    (const void* sbuff, void* rbuff, size_t cnt,  ncclDataType_t dtype, ncclRedOp_t op,           ncclComm_t comm, cudaStream_t s);
ncclResult_t ncclReduceScatter(const void* sbuff, void* rbuff, size_t rcnt, ncclDataType_t dtype, ncclRedOp_t op,           ncclComm_t comm, cudaStream_t s);
ncclResult_t ncclAllGather    (const void* sbuff, void* rbuff, size_t scnt, ncclDataType_t dtype,                           ncclComm_t comm, cudaStream_t s);

// Point to point communication - since 2.7
ncclResult_t ncclSend         (const void* sbuff,              size_t cnt,  ncclDataType_t dtype,                 int peer, ncclComm_t comm, cudaStream_t s);
ncclResult_t ncclRecv         (                   void* rbuff, size_t cnt,  ncclDataType_t dtype,                 int peer, ncclComm_t comm, cudaStream_t s);

// Fuse operations together - since 2.0
ncclResult_t ncclGroupStart();
ncclResult_t ncclGroupEnd();

// Custom scaling factor - since 2.11
ncclResult_t ncclRedOpCreatePreMulSum(ncclRedOp_t *op, void *scalar, ncclDataType_t datatype, ncclScalarResidence_t residence, ncclComm_t comm);
ncclResult_t ncclRedOpDestroy(ncclRedOp_t op, ncclComm_t comm);

// Misc
ncclResult_t ncclGetVersion(int *version);                                                                     // Since 2.0
const char*  ncclGetLastError(ncclComm_t comm);                                                                // Since 2.13
ncclResult_t ncclCommCount(const ncclComm_t comm, int* count);                                                 // Since 2.0
ncclResult_t ncclCommCuDevice(const ncclComm_t comm, int* device);                                             // Since 2.0
ncclResult_t ncclCommUserRank(const ncclComm_t comm, int* rank);                                               // Since 2.0
```

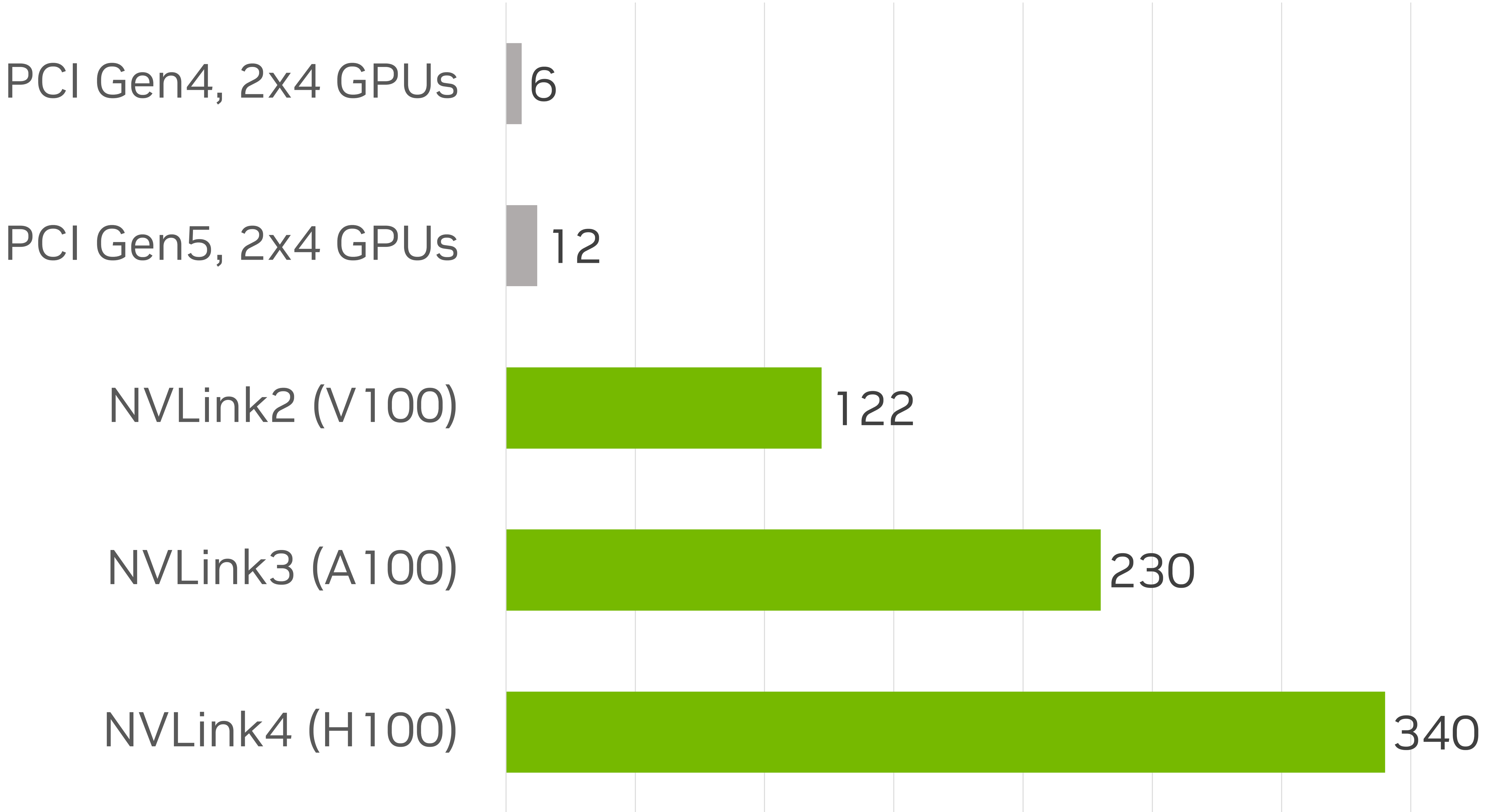
Collective Communication Bandwidth

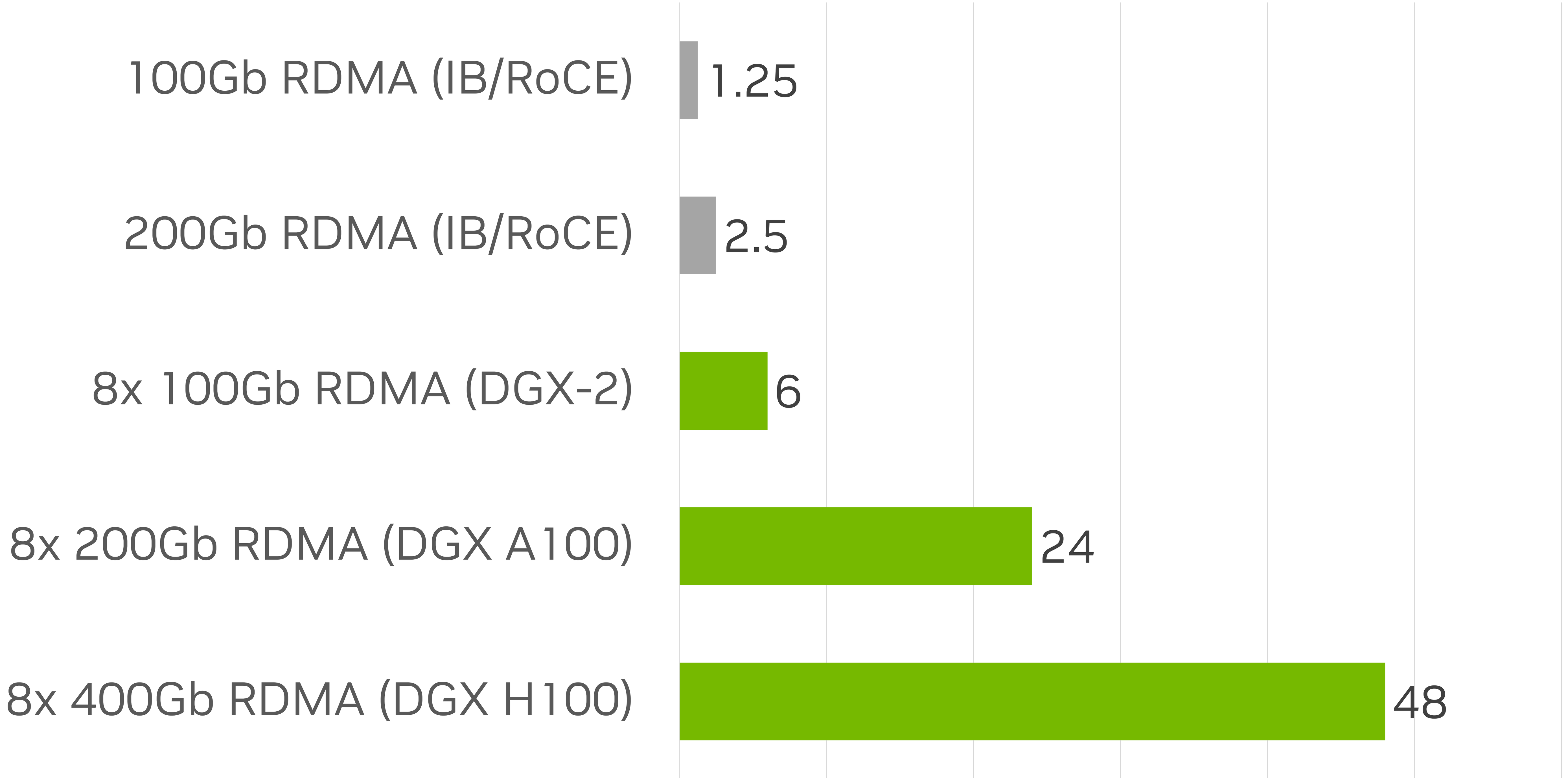




NCCL Tests Allreduce Bus Bandwidth in GB/s

Point-to-point Communication Bandwidth





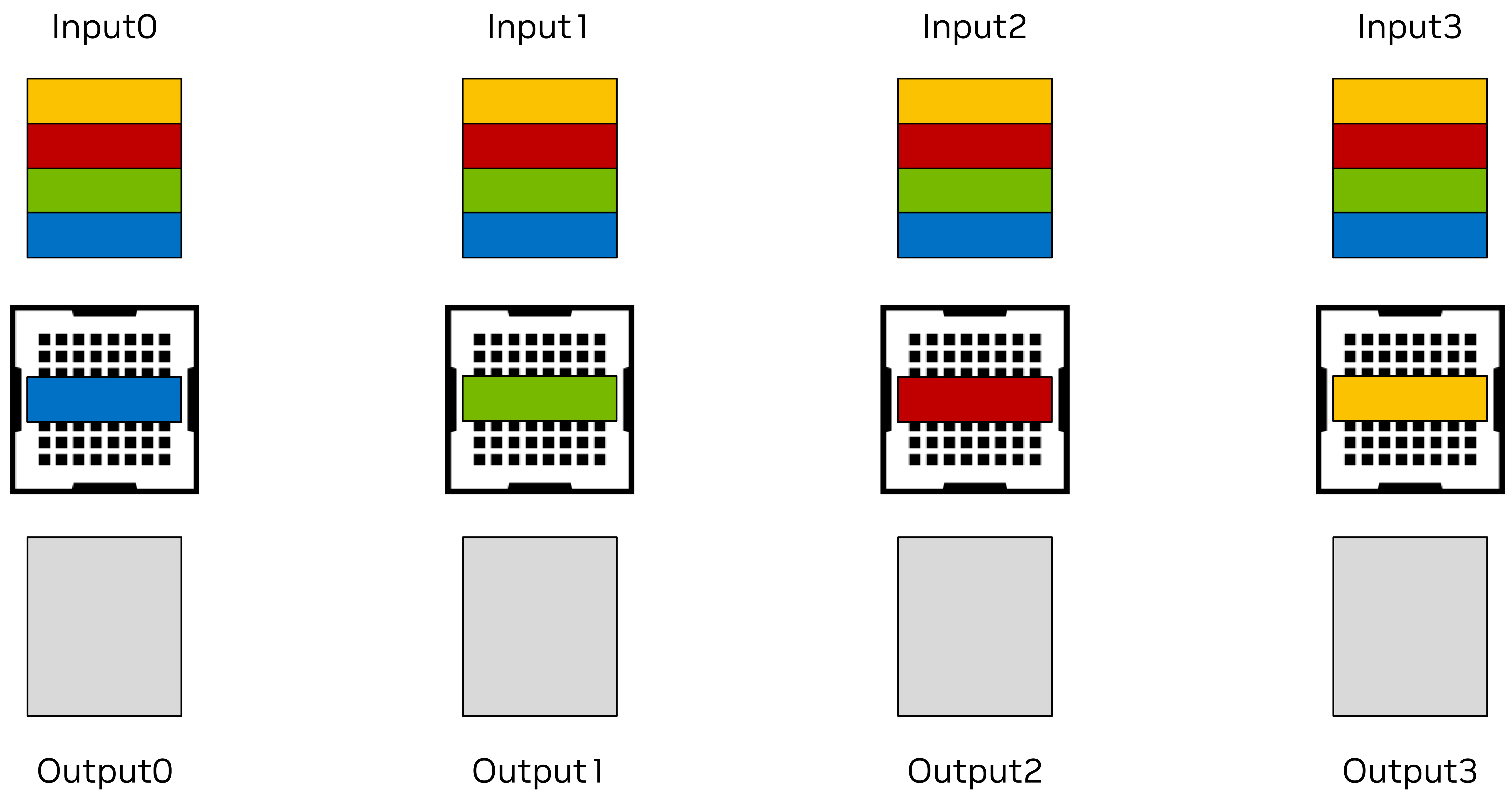
NCCL Tests Alltoall Bus Bandwidth in GB/s

Agenda

- Deep Learning Training
- NCCL Overview
- Collective Operations
- Topology detection
- Point-to-point Communication
- New and Future

Ring Algorithm

For allreduce



Simple algorithm, works on all topologies, balanced computation.

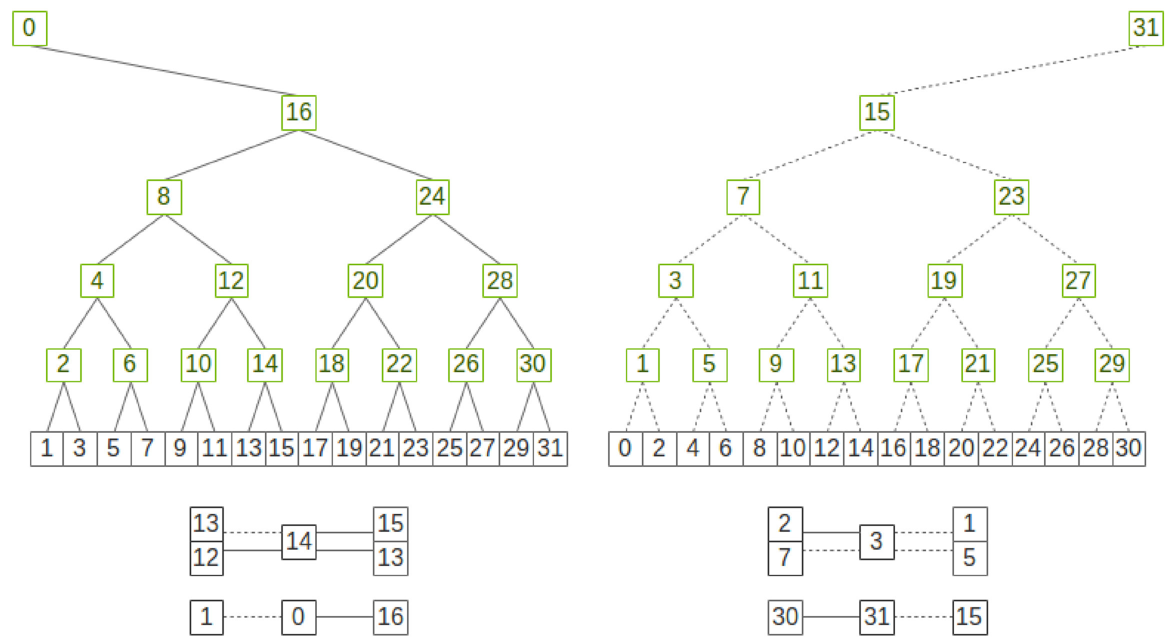
Latency increases linearly with the number of GPUs; quickly degrades at scale.

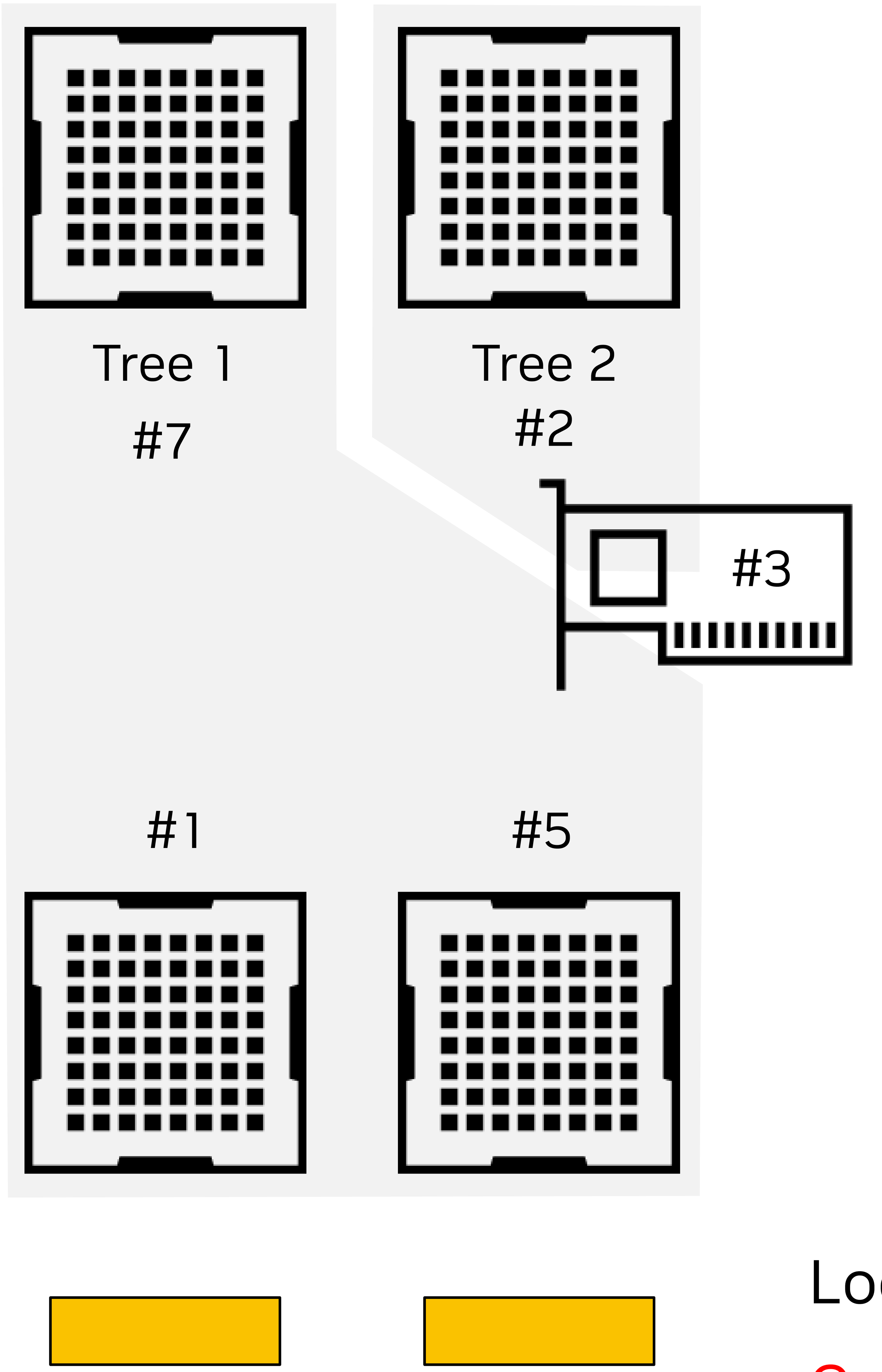
Ring algorithm

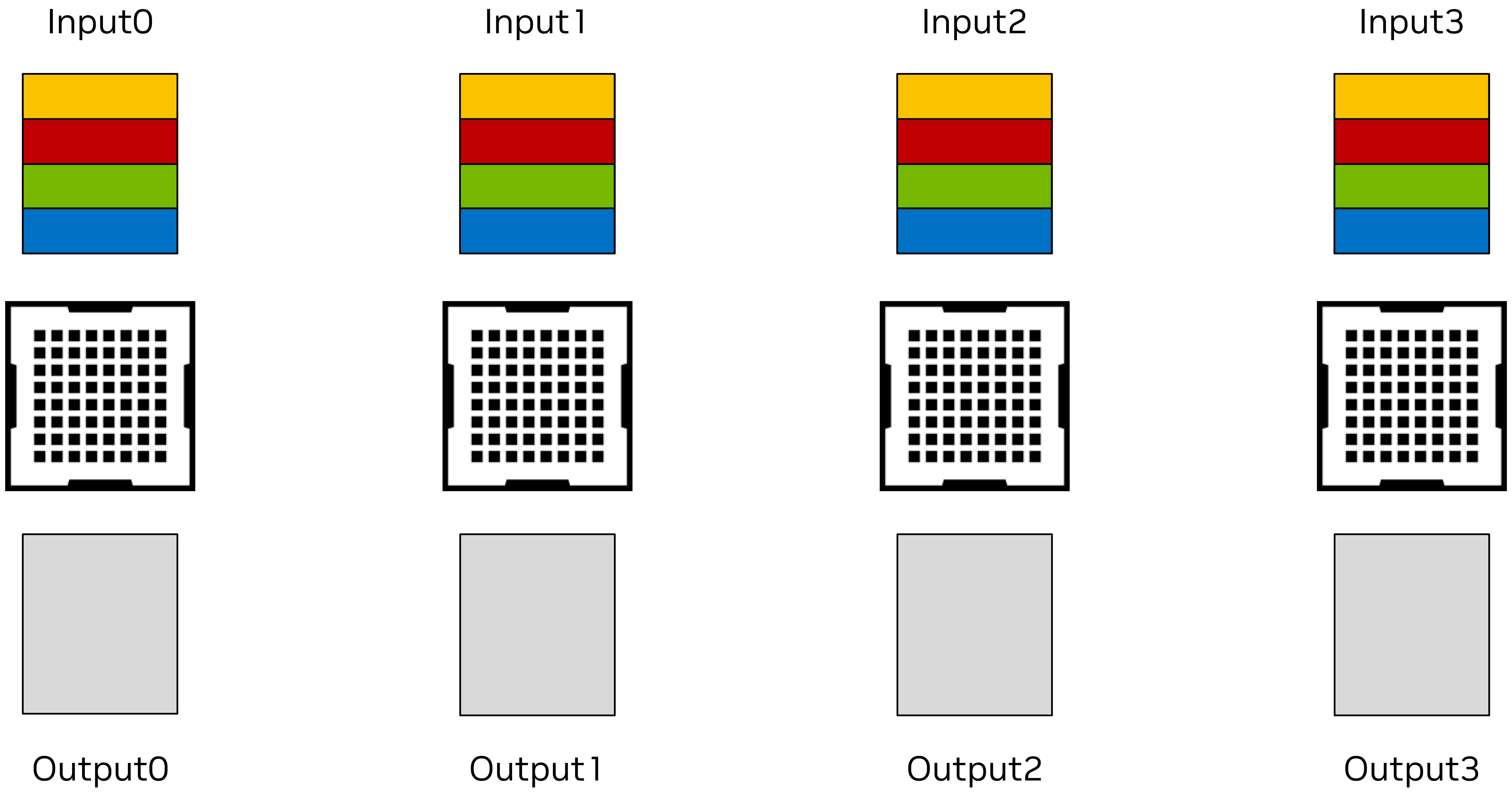
Multi-rings on DGX

Tree Algorithm

For allreduce





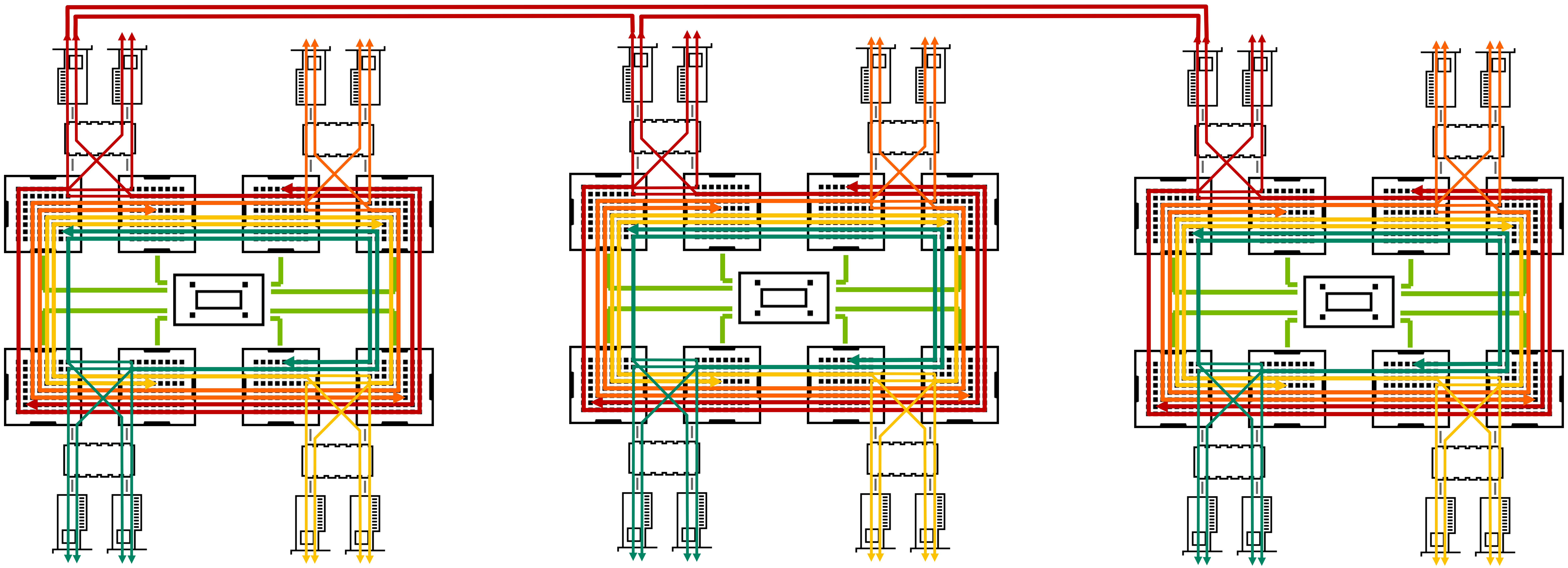


Log latency scales better than ring.

Compute imbalance often causes peak bandwidth to be sub-par.

Tree algorithm

Multi-tree on DGX



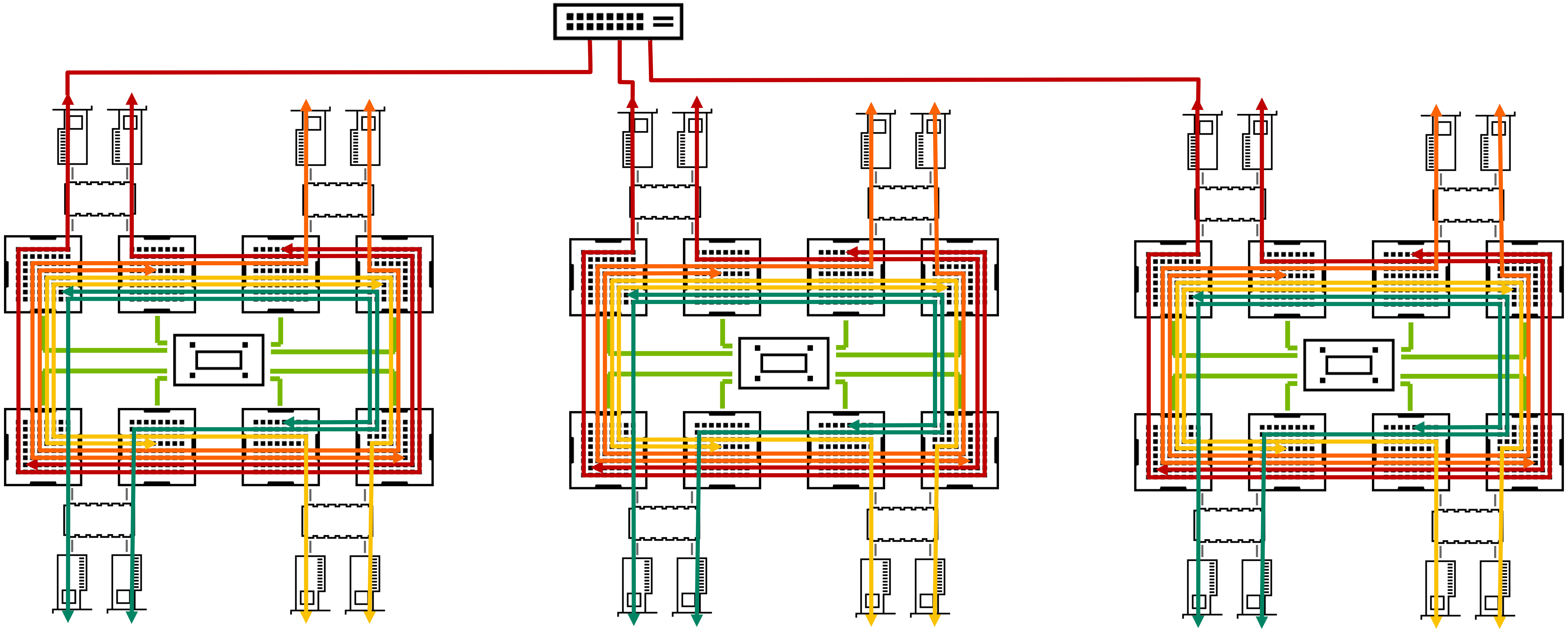
Collnet Algorithm

Chain version

Traffic on the network is largely reduced, improving peak bandwidth if the network was the bottleneck.
At-scale latency is nearly constant so performance is stable at scale.

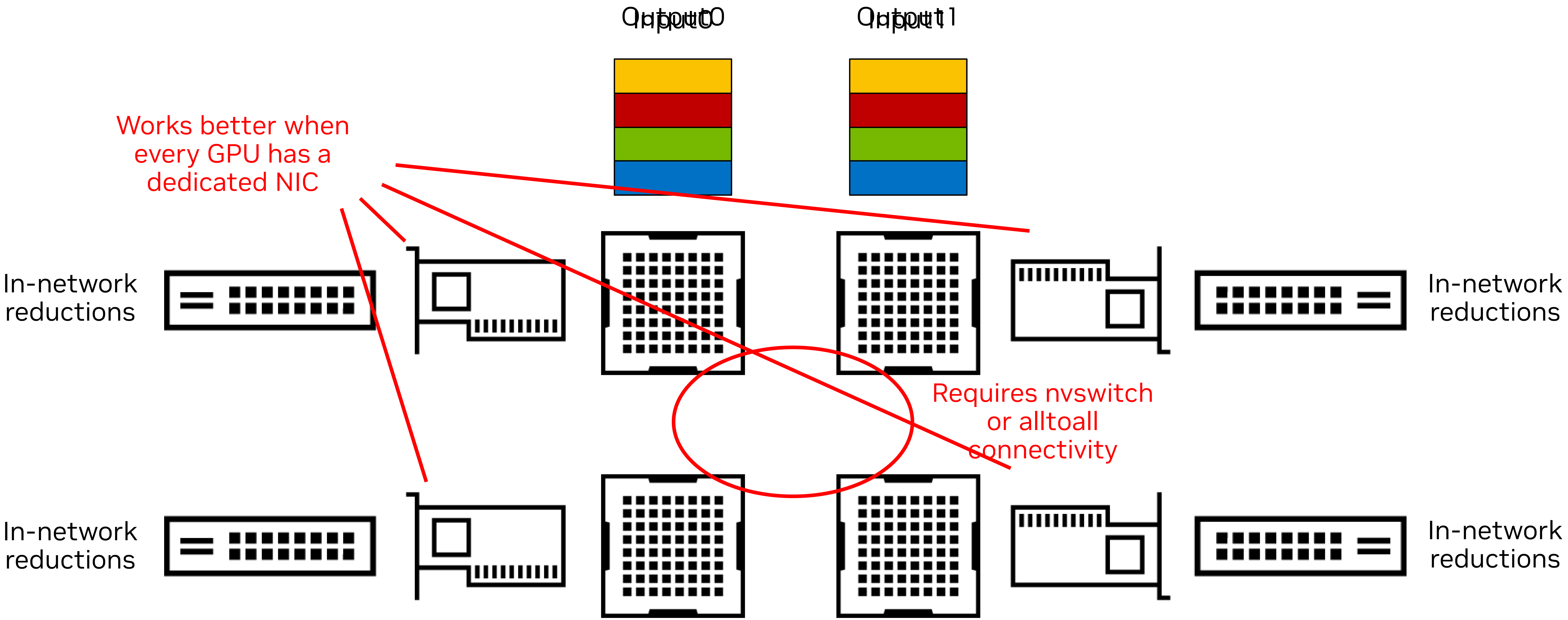
Collnet/chain algorithm

Multiple inter-node chains on DGX



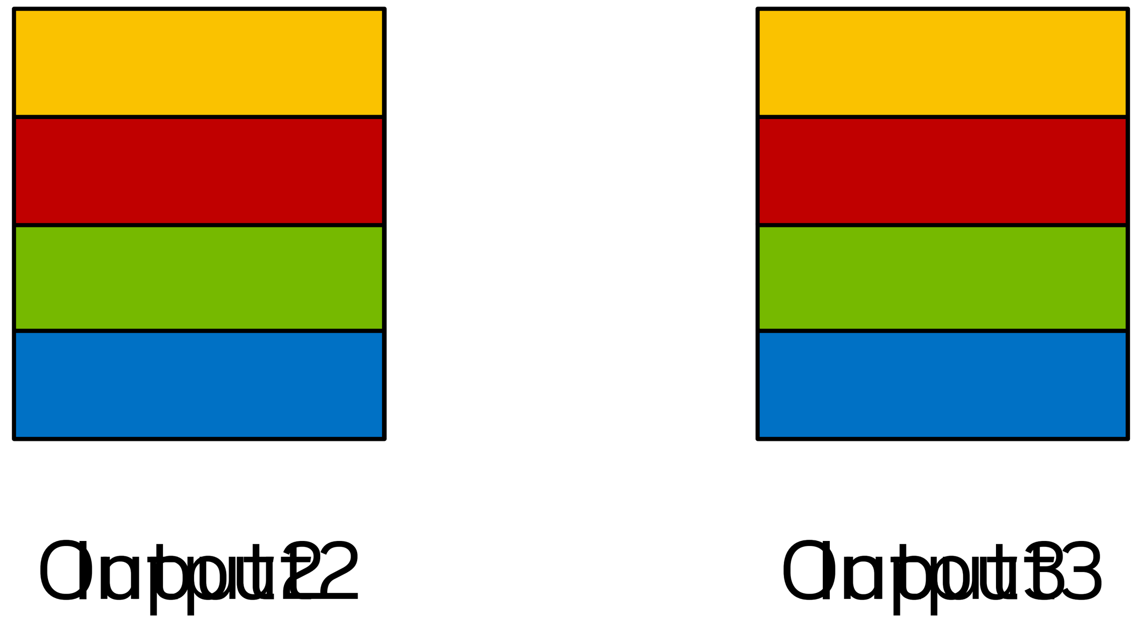
COLLNET Algorithm

Direct version



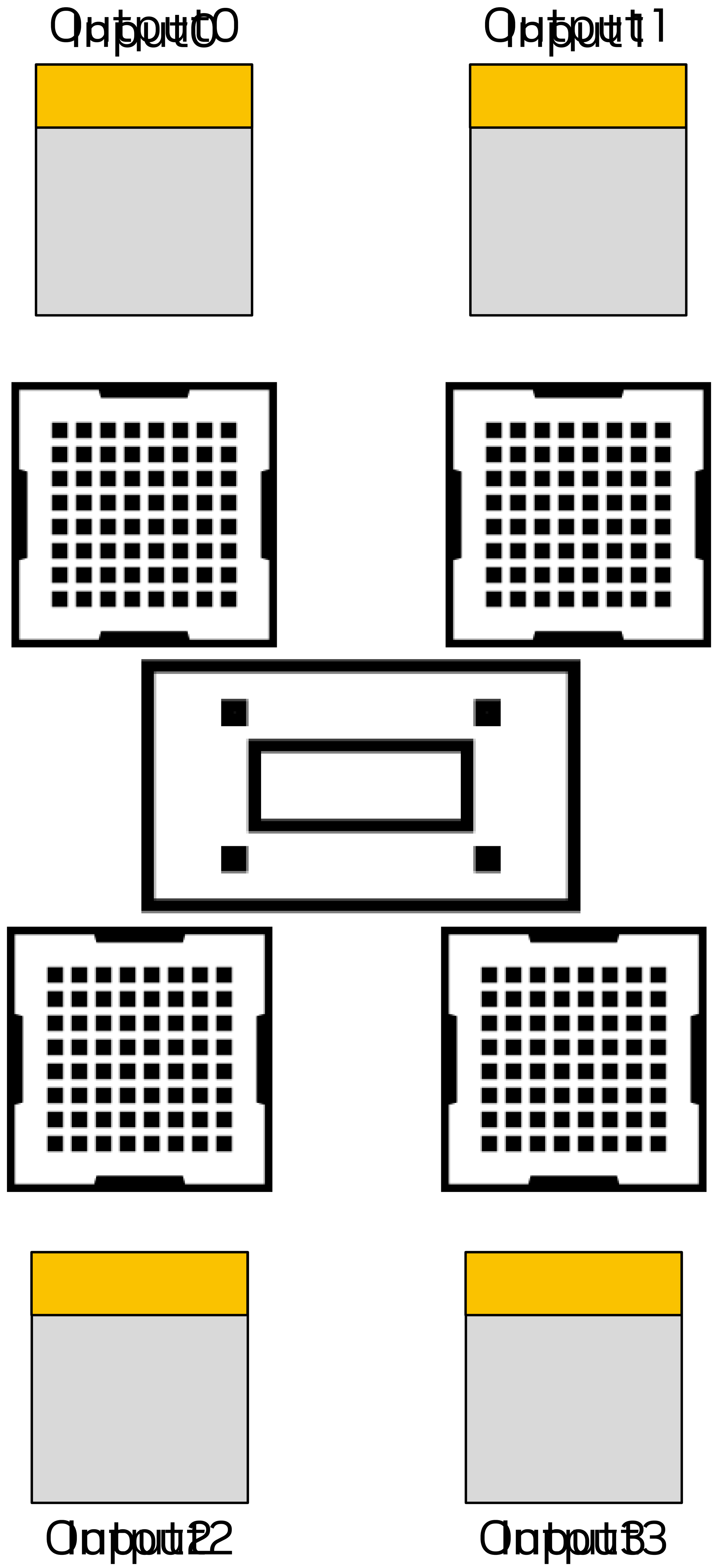
Better intra-node latency than the Chain version, especially when registering buffers with CUDA graphs.

Alltoall phase may not reach the optimal bandwidth.



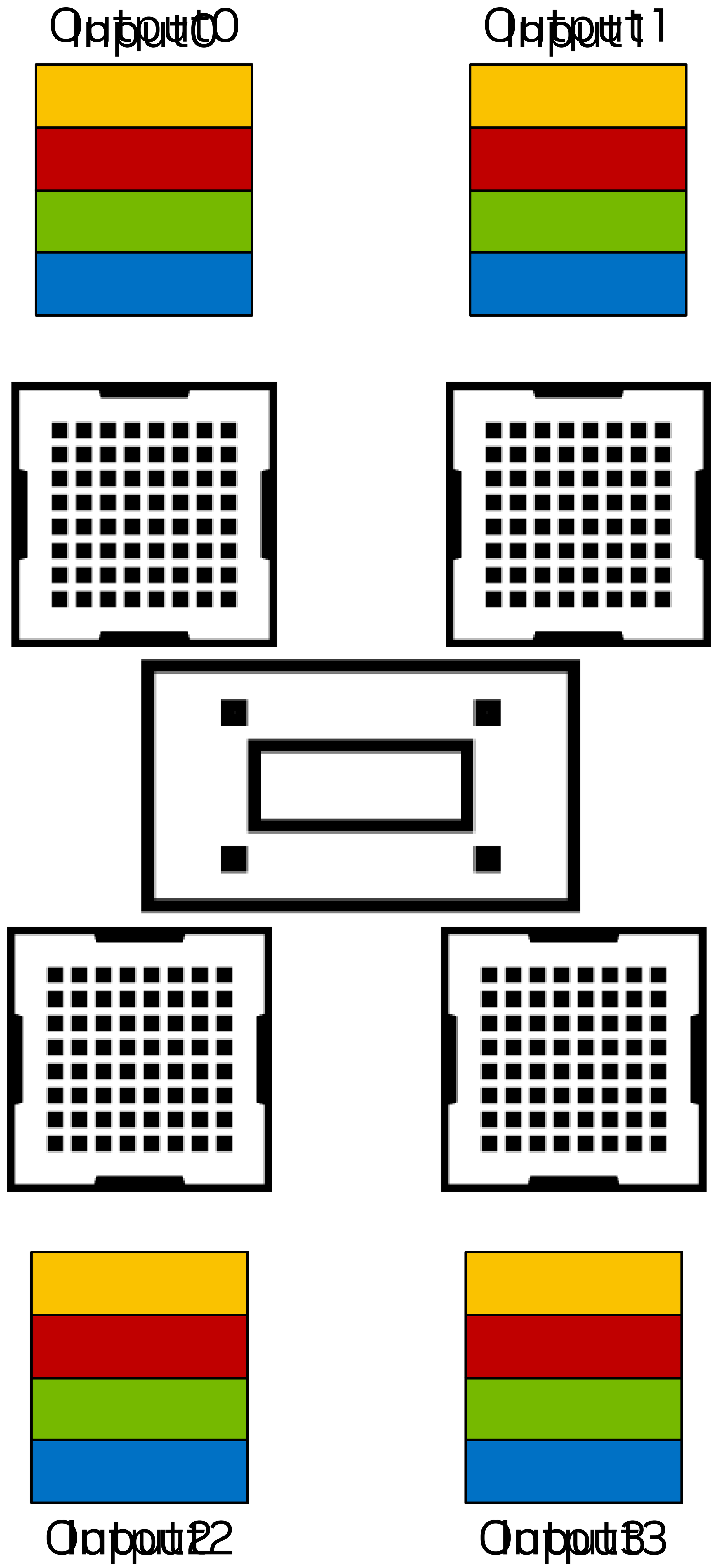
H100 SHARP

NVLink SHARP



H100 SHARP

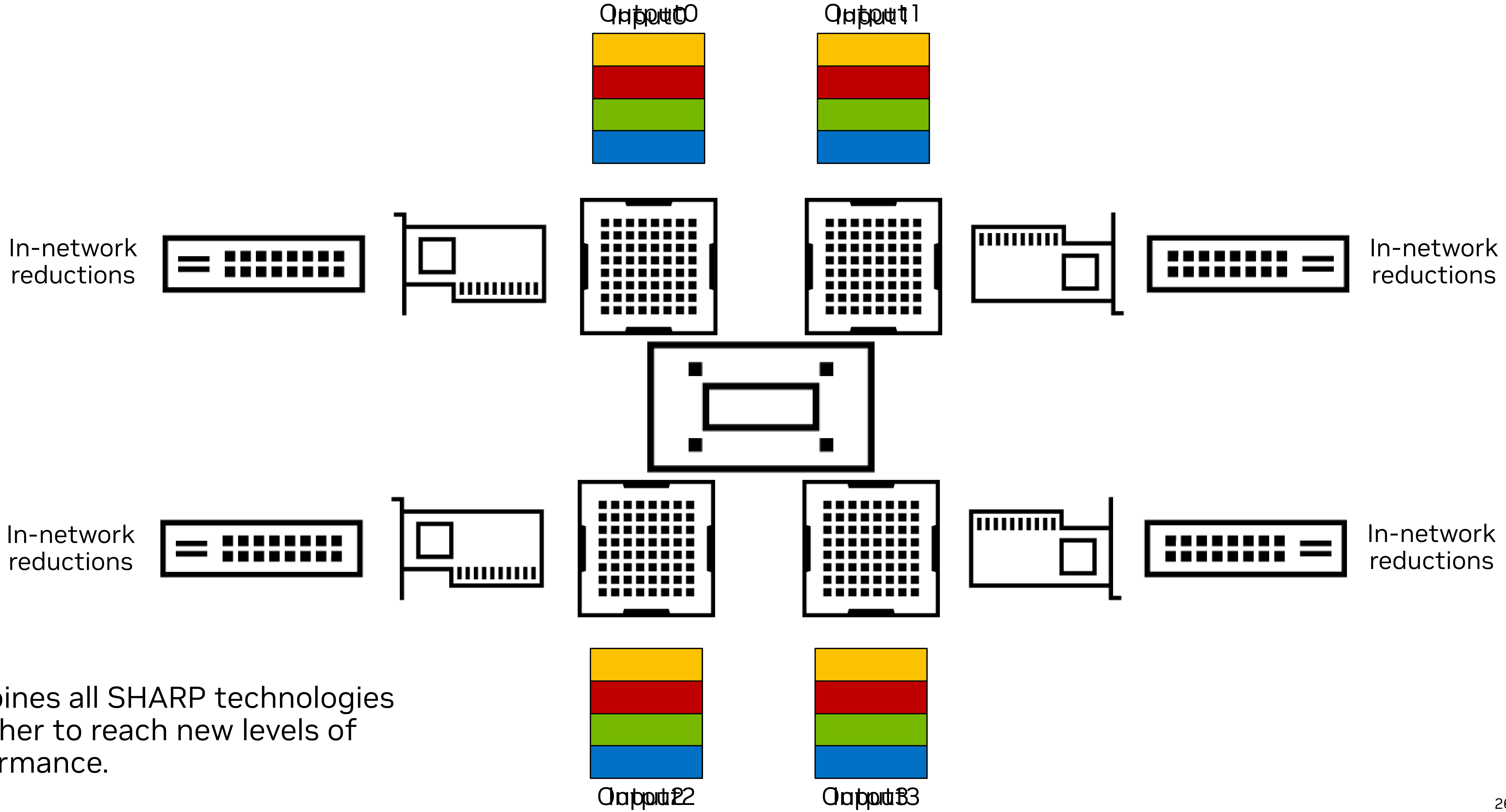
NVLink SHARP



As for SHARP, communication through NVLink is reduced, reaching higher bandwidth.

H100 SHARP

NVLink and IB



Combines all SHARP technologies together to reach new levels of performance.

Agenda

- Deep Learning Training
- NCCL Overview
- Collective Operations
- **Topology detection**
- Point-to-point Communication
- New and Future

Topology Detection

Mapping algorithms to the hardware

Topology detection

Build graph with all GPUs, NICs, CPUs, PCI switches, NVLink, NVSwitch.

Topology injection for VMs.

Graph search

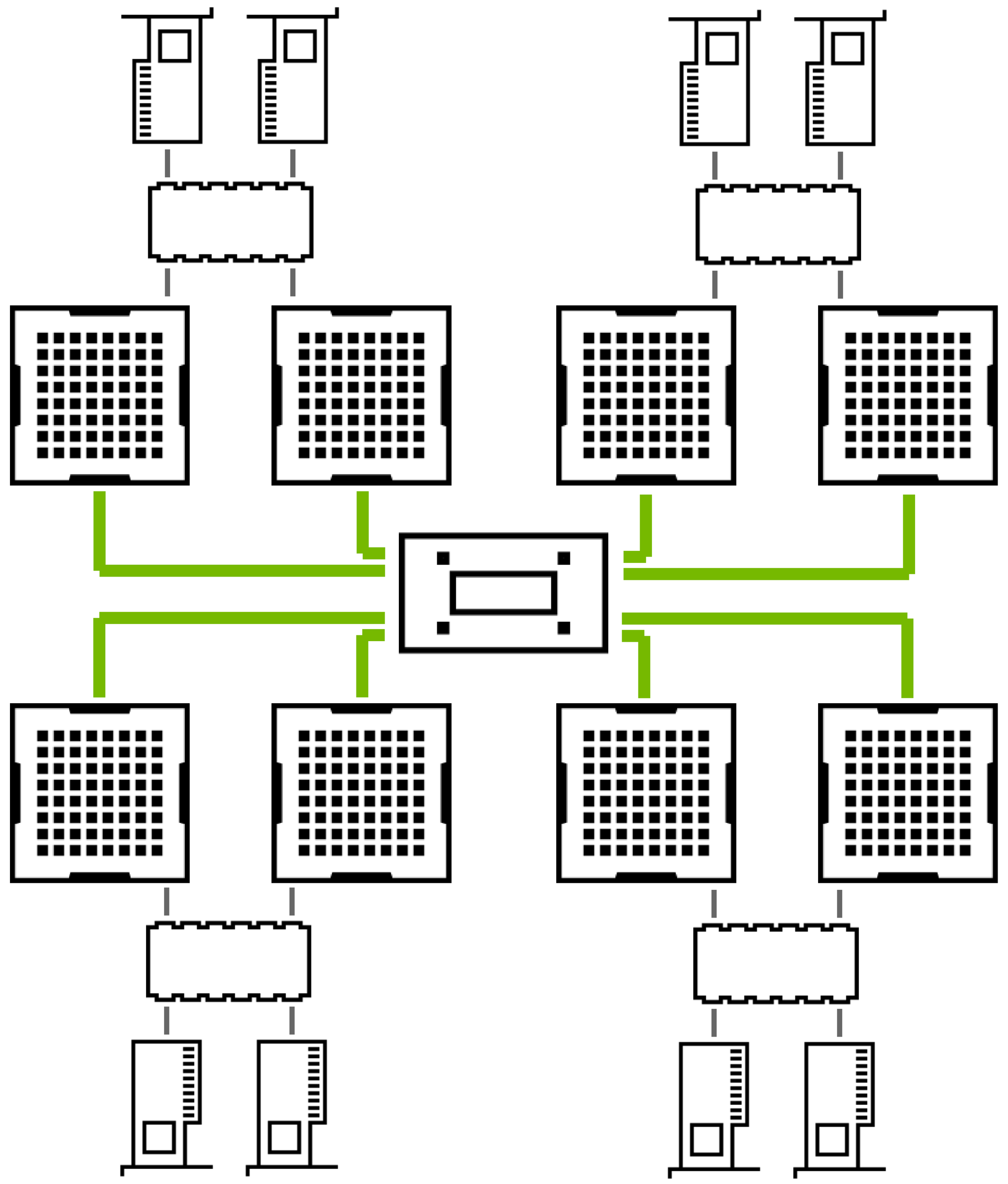
Find optimal set of paths within the node for rings, trees and chains.

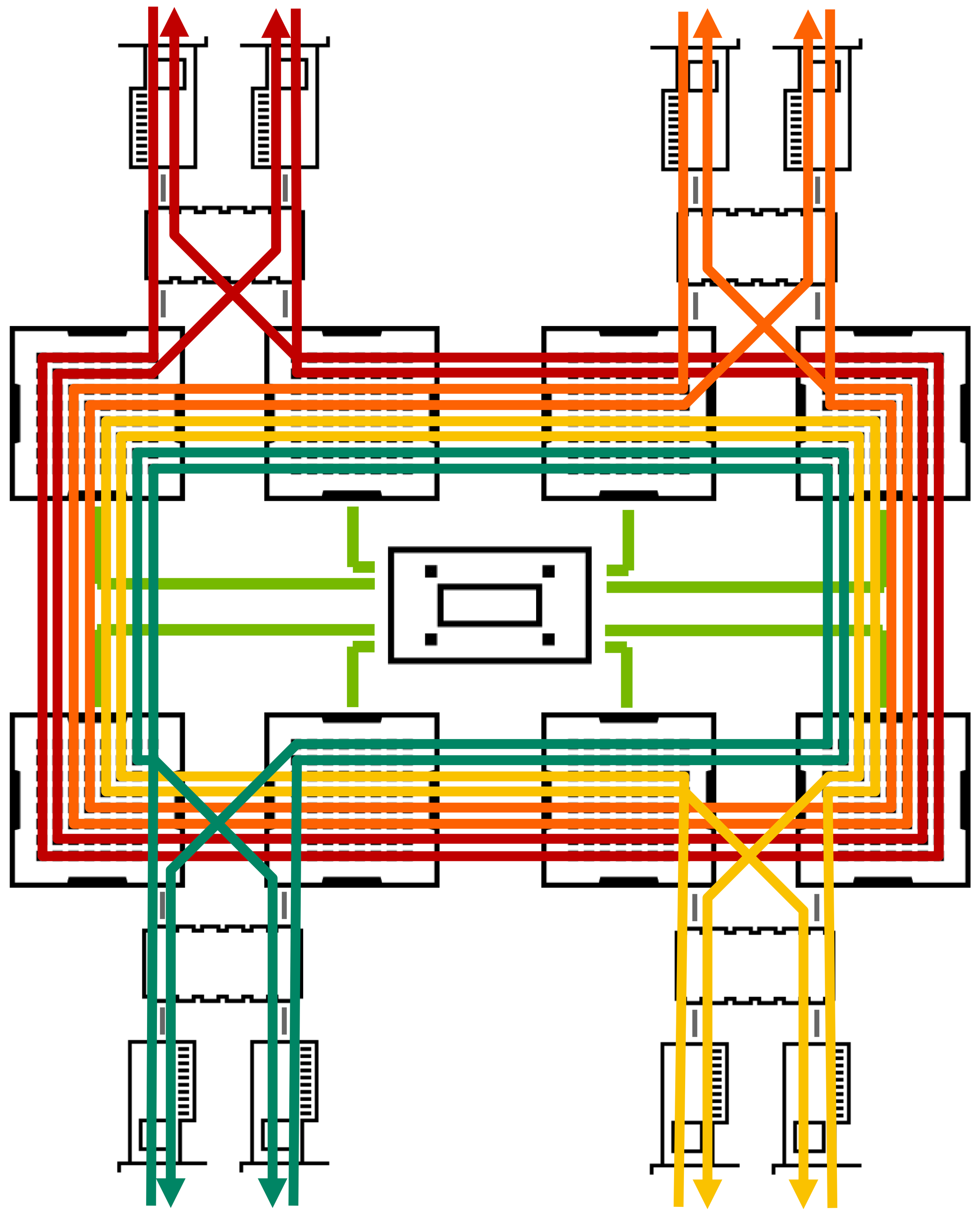
Performance model for each algorithm, tuning.

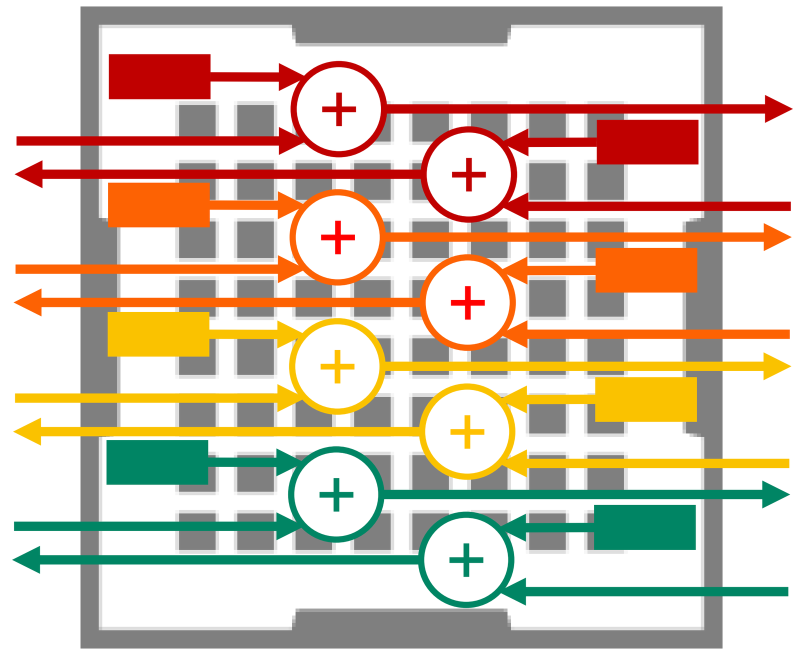
CUDA Kernels

Optimized reductions and copies for a minimal SM usage.

CPU threads for network communication.

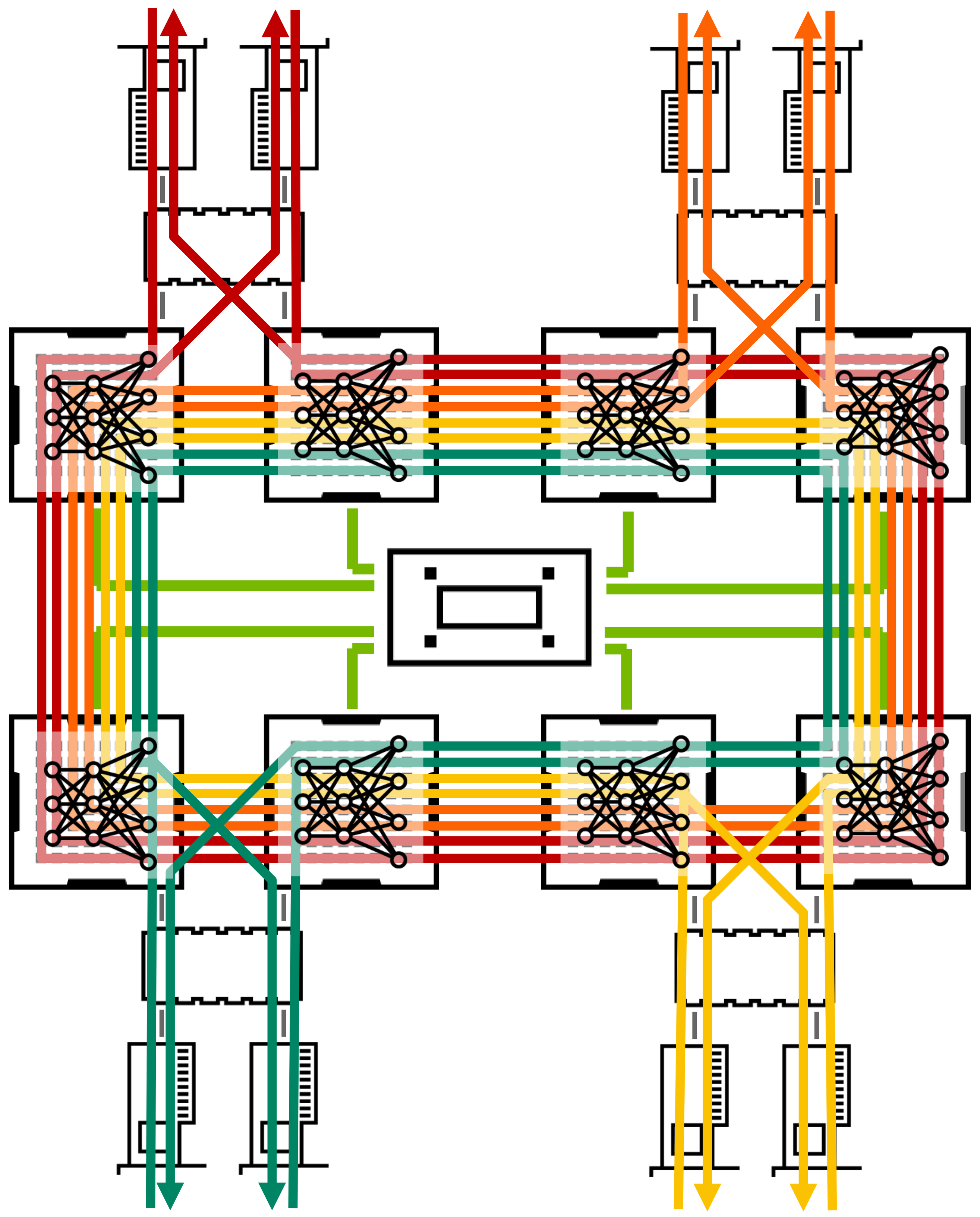






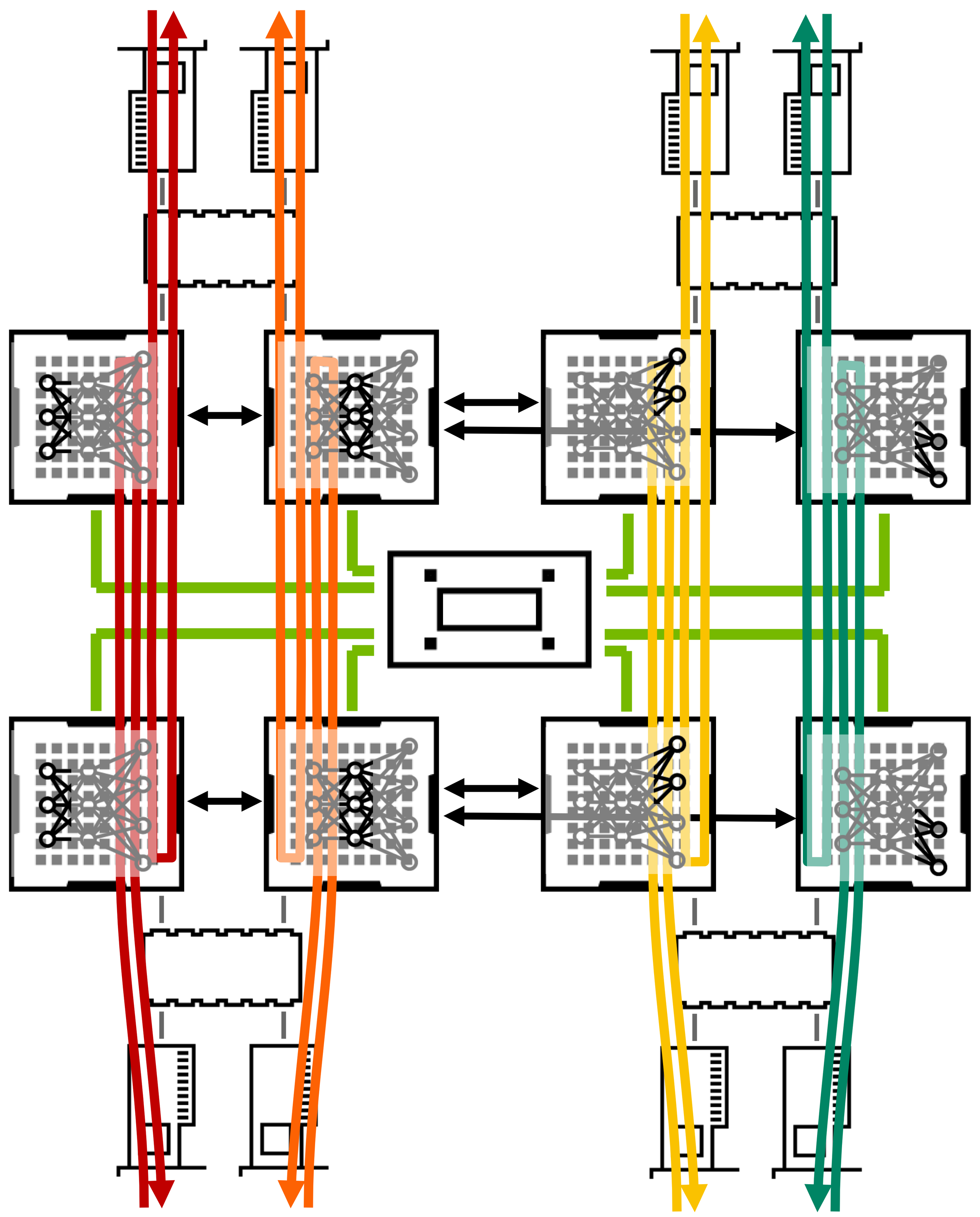
Topology Detection

Mapping algorithms to the hardware



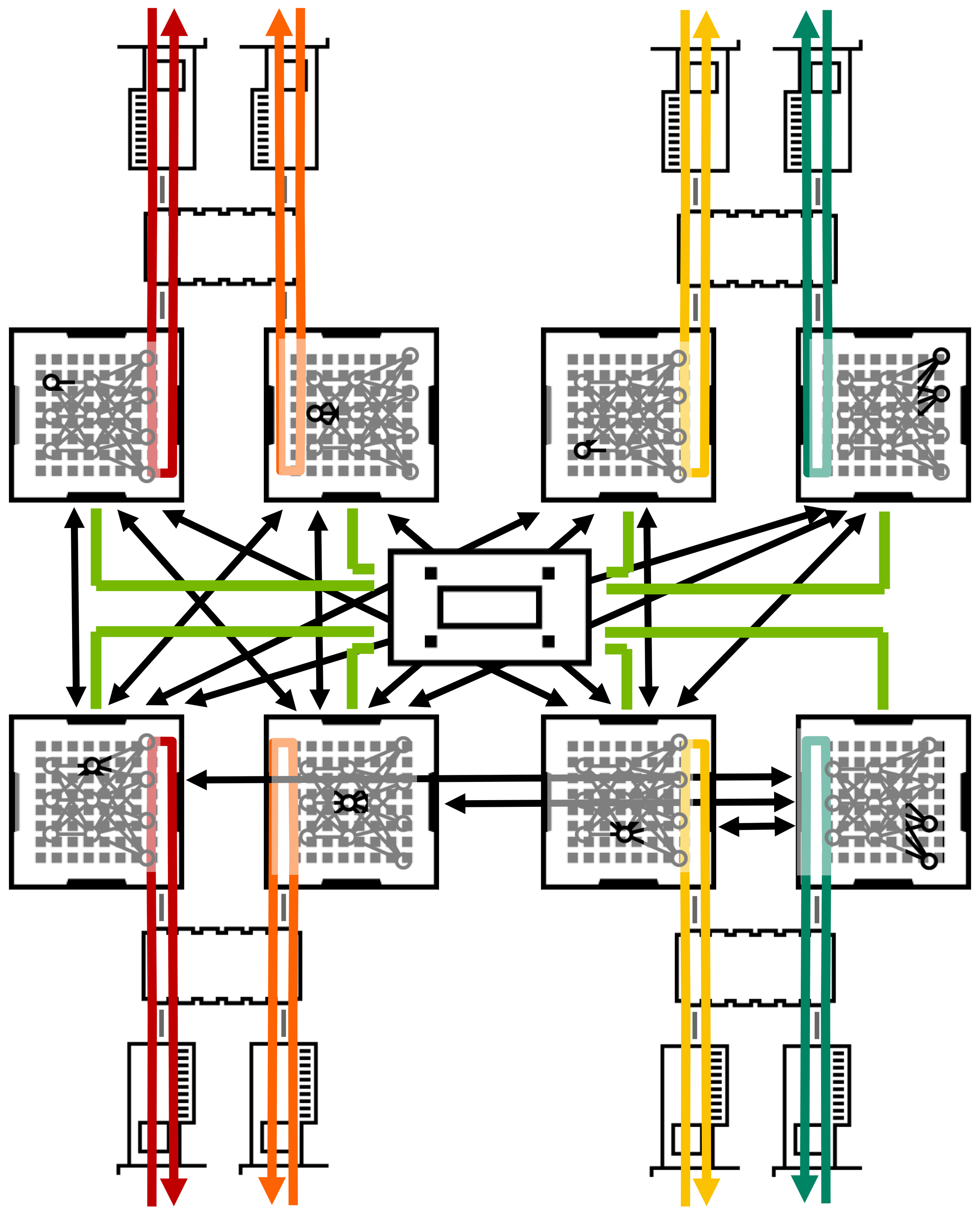
Pure data parallelism

8 GPUs per node in the global communicator.



Pipeline parallelism on 4 GPUs.

2 GPUs per node in each of the 4 NCCL communicators.



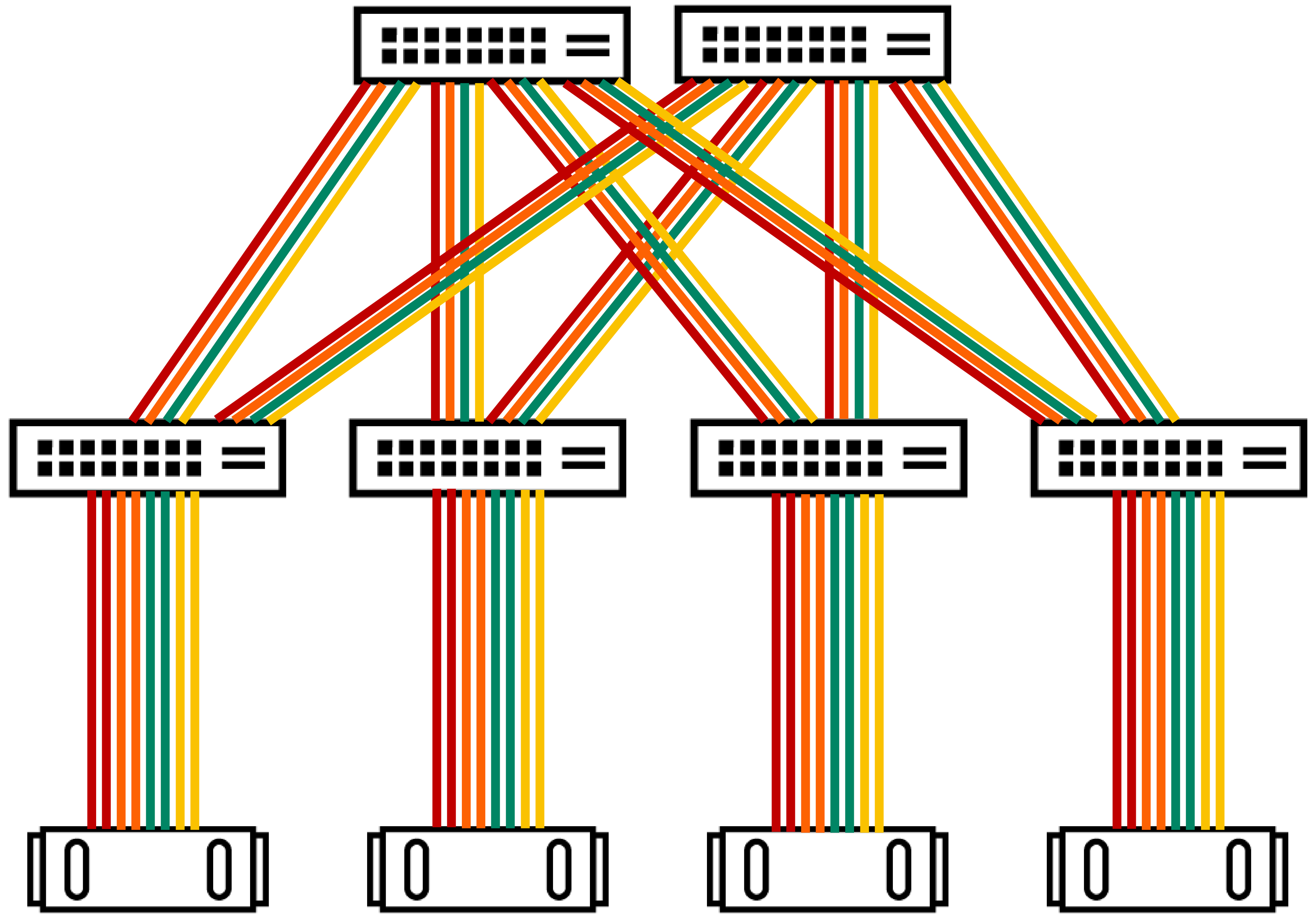
Pipeline parallelism on 8 GPUs.

1 GPU per node in each of the 8 NCCL communicators.

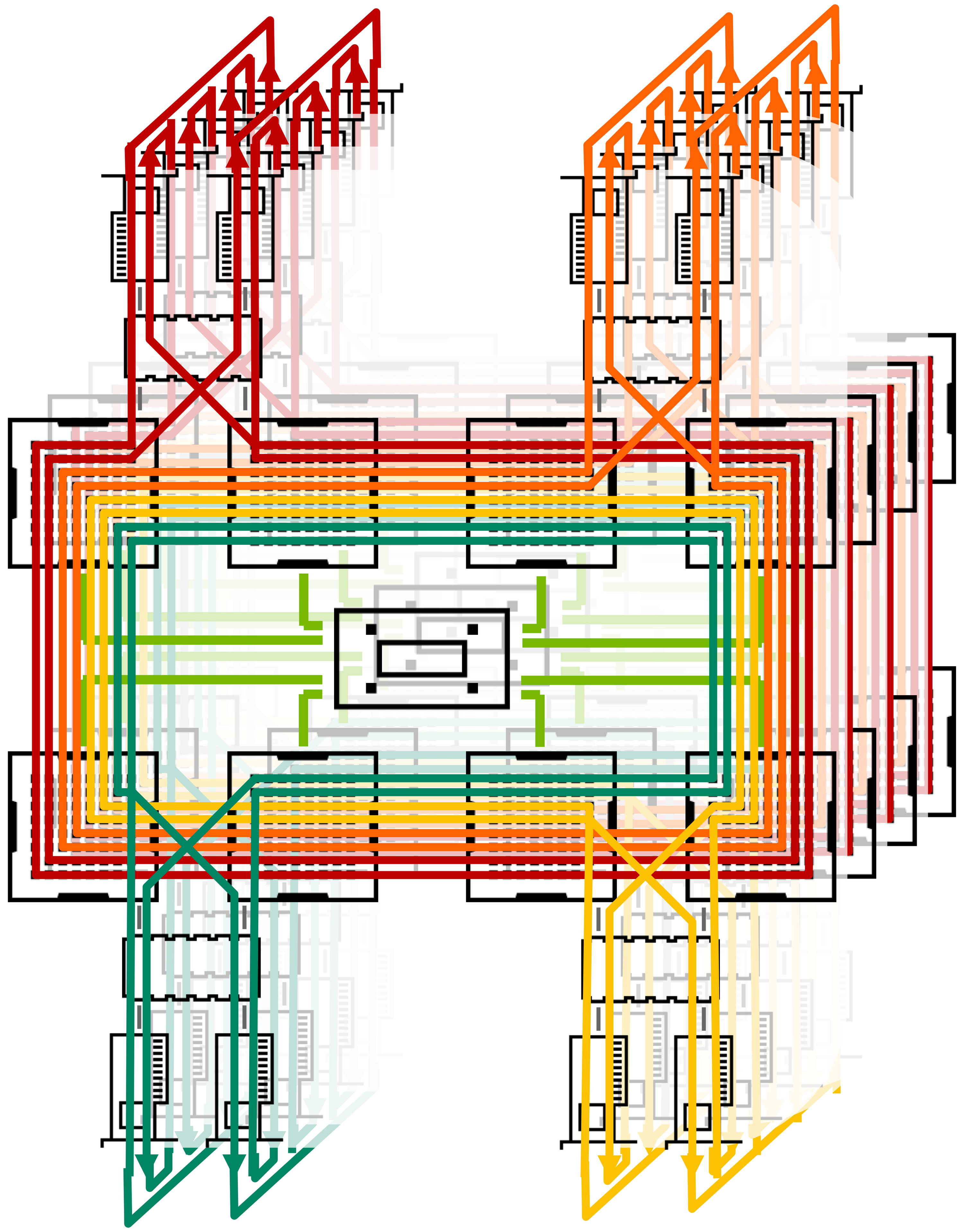
Inter-node Communication

Rail-optimized design

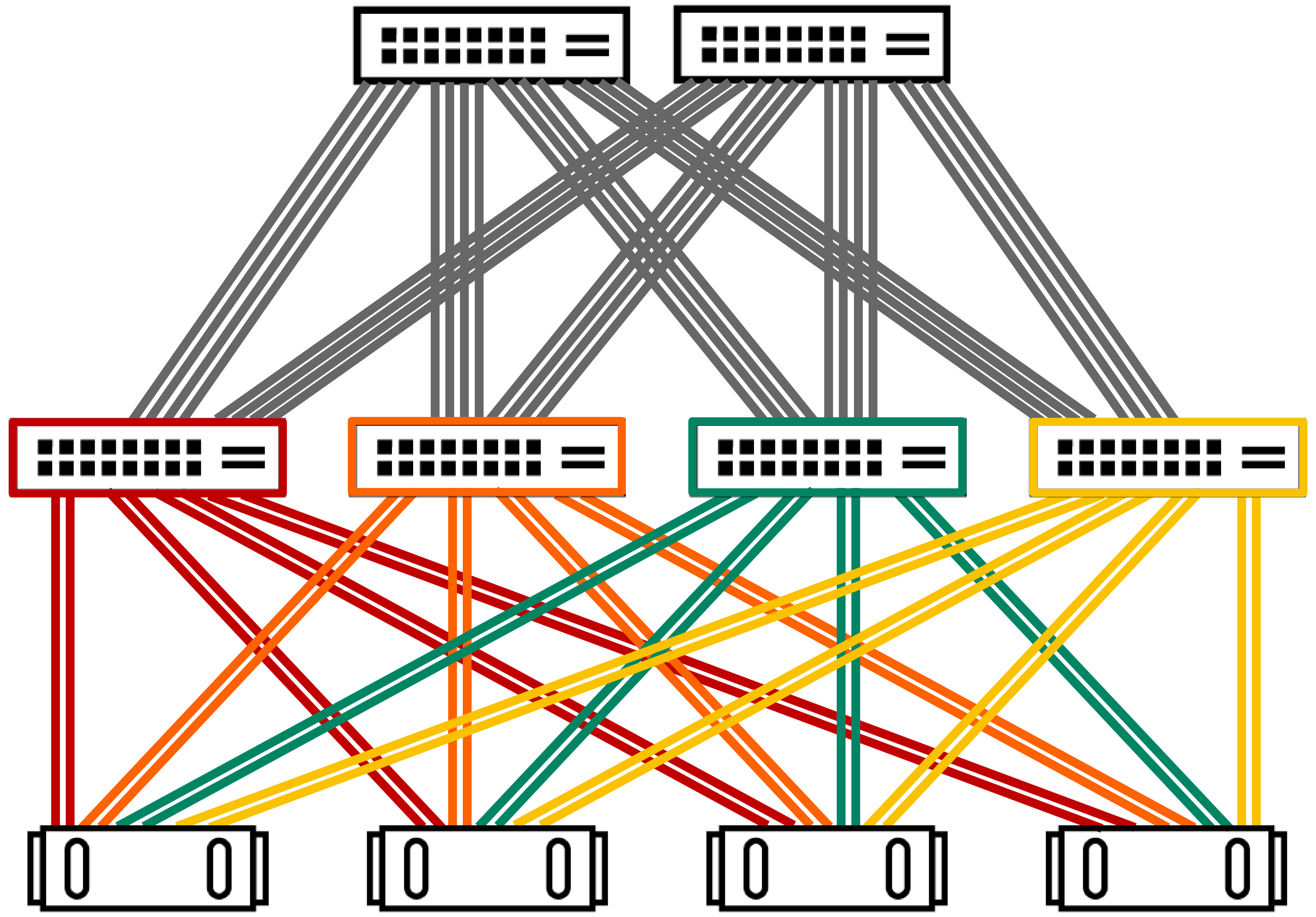
Routing must be perfect to ensure all flows use different links.



Classic fabric design



All traffic is local to leaf switches. Routing collisions are impossible.



Rail-optimized design

Agenda

- Deep Learning Training
- NCCL Overview
- Collective Operations
- Topology detection
- **Point-to-point Communication**
- New and Future

Point-to-point Semantics

Grouping and alltoall example

NCCL defines only two functions for point-to-point communication: `ncclSend` and `ncclRecv` which, grouped using `ncclGroupStart`/`ncclGroupEnd`, can be used to easily write any communication pattern, including alltoall[v,w], scatter[v], gather[v], neighbor collectives, etc …

Alltoall example:

```
ncclGroupStart();
for (int i=0; i<nranks; i++) {
  ncclSend(sendbuffs[i], count, type, i, comm);
  ncclRecv(recvbuffs[i], count, type, i, comm);
}
ncclGroupEnd();
```

The order within the group call does not matter, and all operations will be progressed together in a deadlock-free manner.

Without grouping, `ncclSend` and `ncclRecv` operations may block until the matching operation is executed on the remote GPU.

Point-to-point Communication

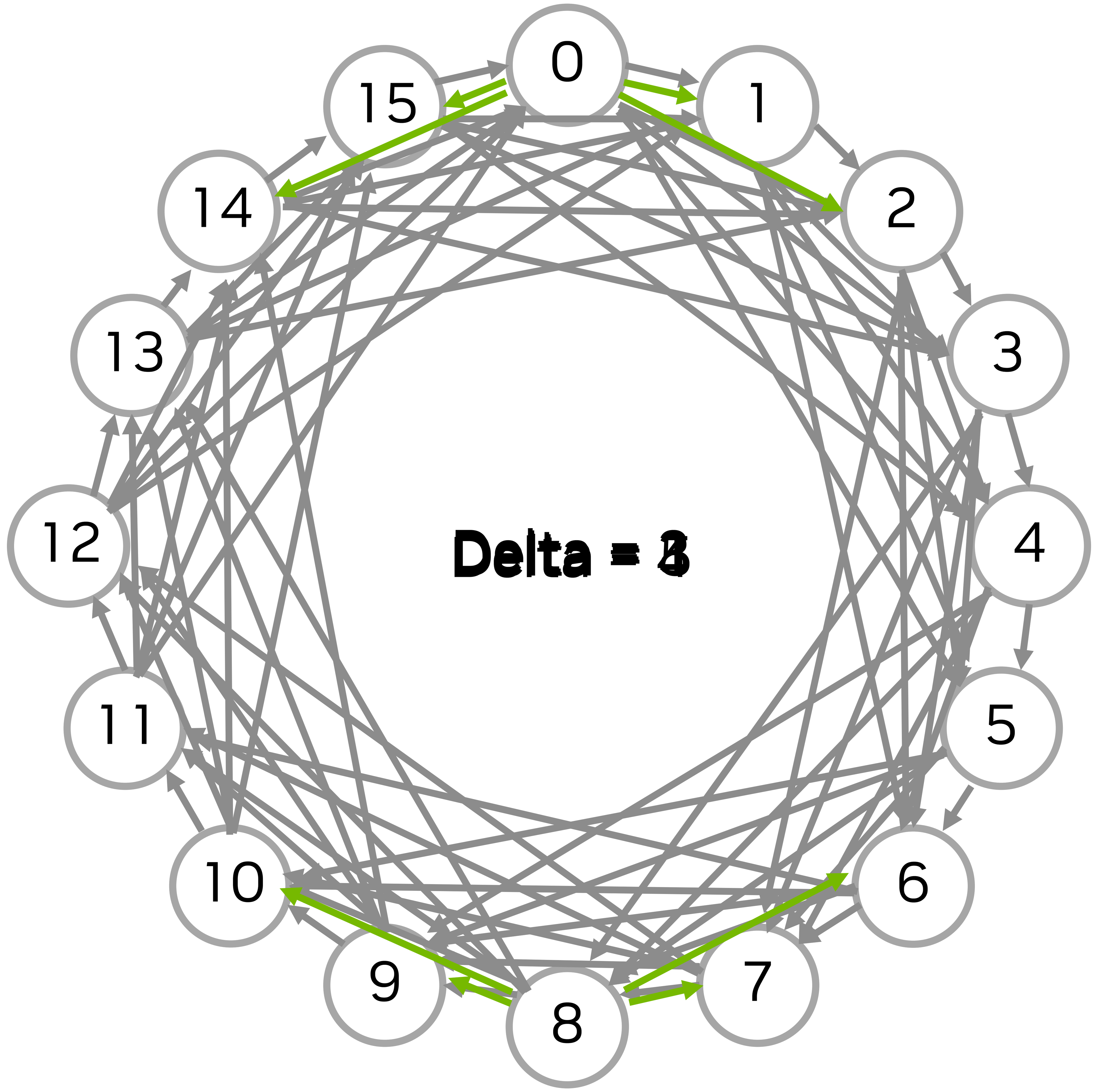
Rotating rings

The point-to-point communication schedule is using the ring principle: simultaneously receive from rank – delta and send to rank + delta. This guarantees deadlock-free operation.

To make sure we use NICs in both directions, we alternate between +delta and – delta.

To ensure we overlap NIC and NVLink usage, we alternate between both ends of the circle (delta = 0 and and delta = N/2).

Finally, for better latency and overlap, we execute X steps in parallel, with X = up to 256.



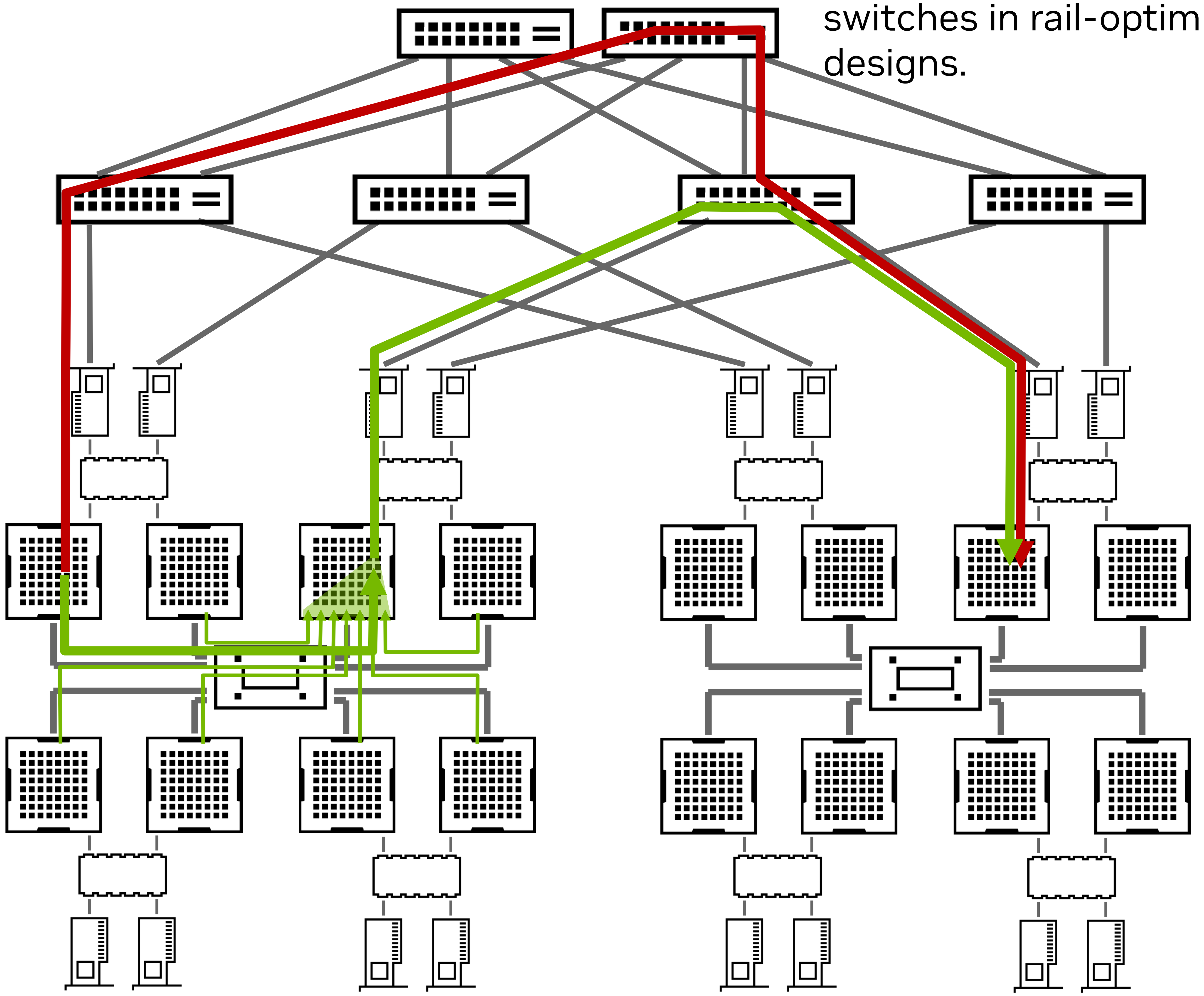
Alltoall Communication

PXN optimization

Alltoall communication has most transfers going through top-level switches in rail-optimized designs.

PXN also allows for the aggregation of all messages to the same destination.

PXN (PCI x NVLink) allows GPUs to send data using a distant NIC, using an intermediate GPU.



Agenda

- Deep Learning Training
- NCCL Overview
- Collective Operations
- Topology detection
- Point-to-point Communication
- New and Future

Fault tolerance

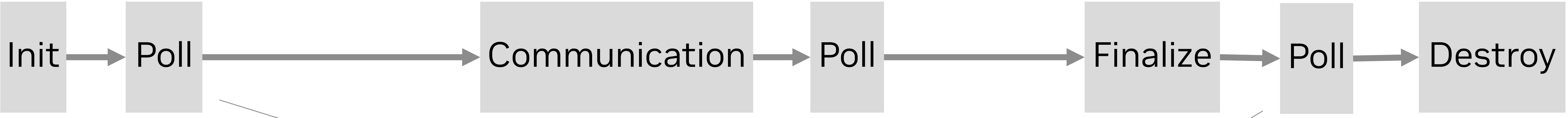
Before NCCL 2.14, NCCL featured the ncclCommAbort function, to stop running kernels on the GPU.

NCCL 2.14 expands the reach of ncclCommAbort to CPU calls which may also block.

Could block if a rank is missing

Could block due to dynamic connection establishment

May need to synchronize with other ranks



config.blocking = 0;
ncclCommInitRankConfig(.., config);

ncclCommFinalize(comm);

Only frees resources

Call ncclCommGetAsyncError until the return code is not **ncclInProgress**.
Call ncclCommAbort if needed.

Recent features

H100 support

NVLink SHARP support, “NVLS” algorithm.

Improved fault tolerance

Ability to abort init/destroy operations, and restart NCCL without restarting the application.

Communicator configuration API

Allow to specify communicator settings through a config structure instead of environment variables: blocking mode, maximum number of CTAs to launch, network type to use, …

Future

More H100 support

NVLink SHARP + IB SHARP
NVLink SHARP + Ring (for non-IB networks)

Communicator split

Create sub-communicator, potentially sharing resources.

CUDA networking

Add device code to optimize network communication (CUDA-initiated communication).

Summary

NCCL : Key communication library for multi-GPU computing.

Optimized for all platforms, from desktop to DGX Superpod.

Download from https://developer.nvidia.com/nccl and in NGC containers.
Source code at https://github.com/nvidia/nccl

NVIDIA