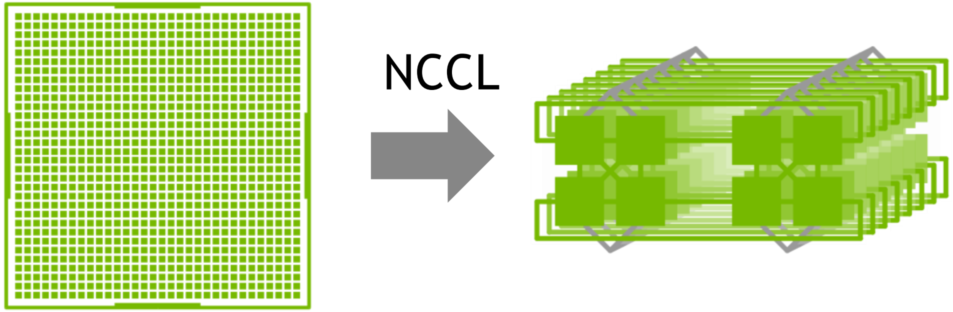
MULTI-GPU TRAINING WITH NCCL

Sylvain Jeaugey

MULTI-GPU COMPUTING

Harvesting the power of multiple GPUs



NCCL

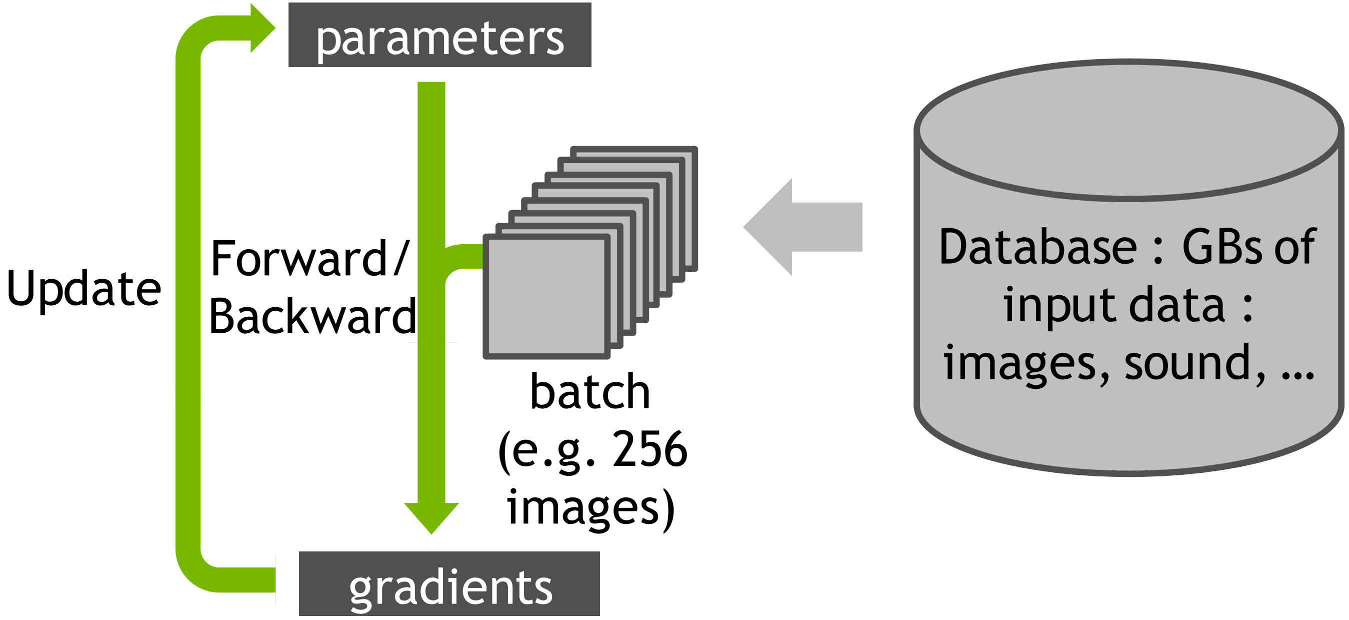
1 GPU

Multiple GPUs per system
Multiple systems connected

NCCL : **N**VIDIA **C**ollective **C**ommunication **L**ibrary

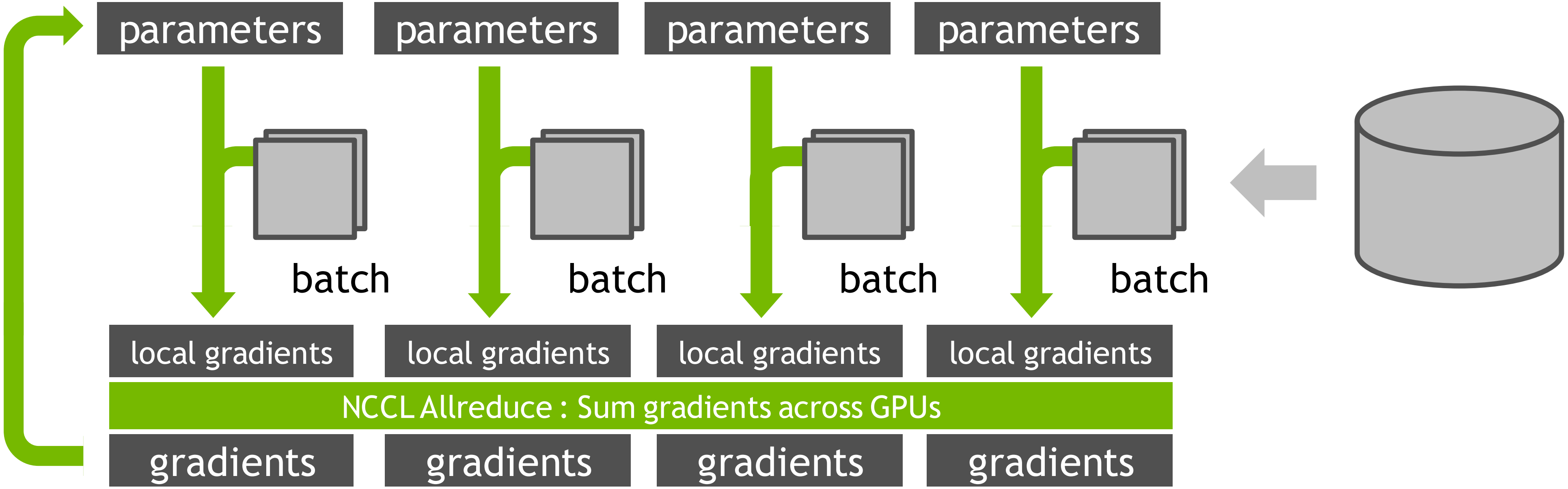
MULTI-GPU DL TRAINING

Single-GPU



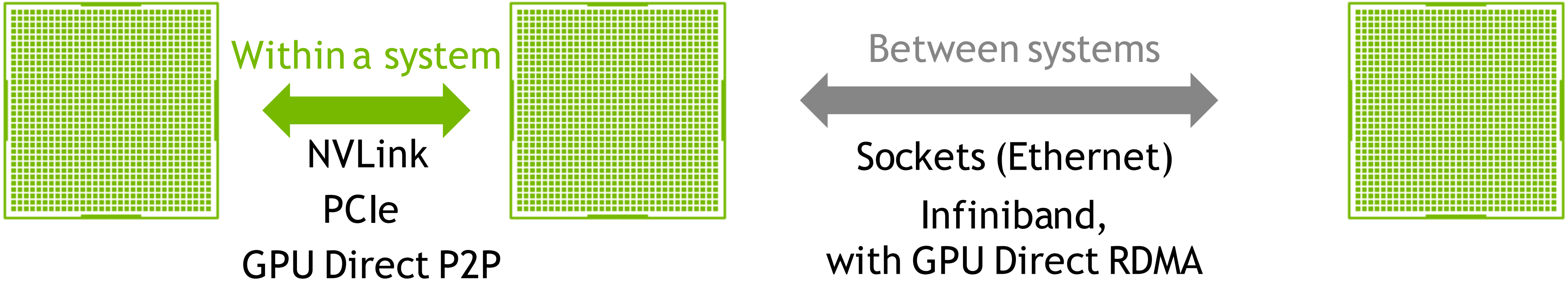
MULTI-GPU DL TRAINING

Data parallel



NCCL

A multi-GPU communication library



Within a system

NVLink

PCIe

GPU Direct P2P

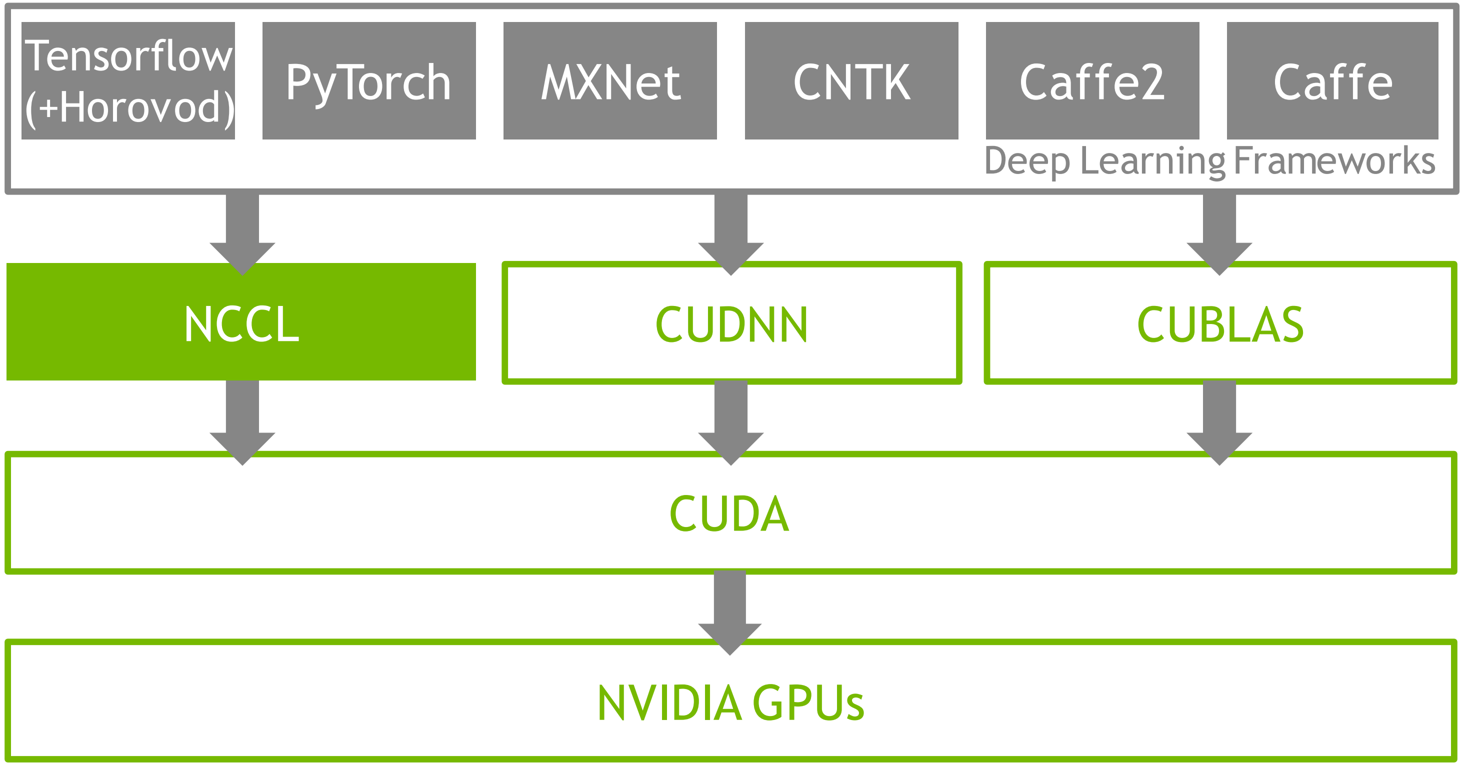
Between systems

Sockets (Ethernet)

Infiniband,
with GPU Direct RDMA

NCCL

Architecture



Tensorflow (+Horovod)

PyTorch

MXNet

CNTK

Caffe2

Caffe

Deep Learning Frameworks

NCCL

CUDNN

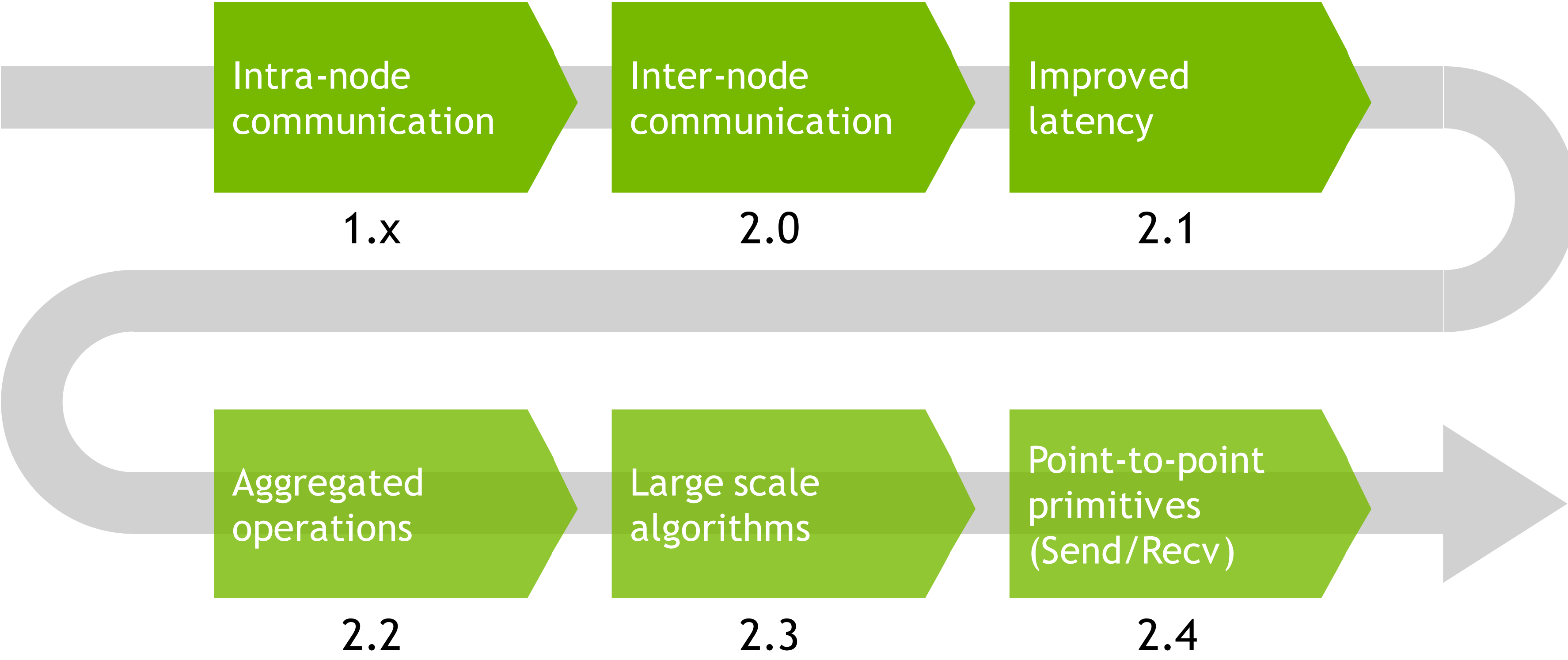
CUBLAS

CUDA

NVIDIA GPUs

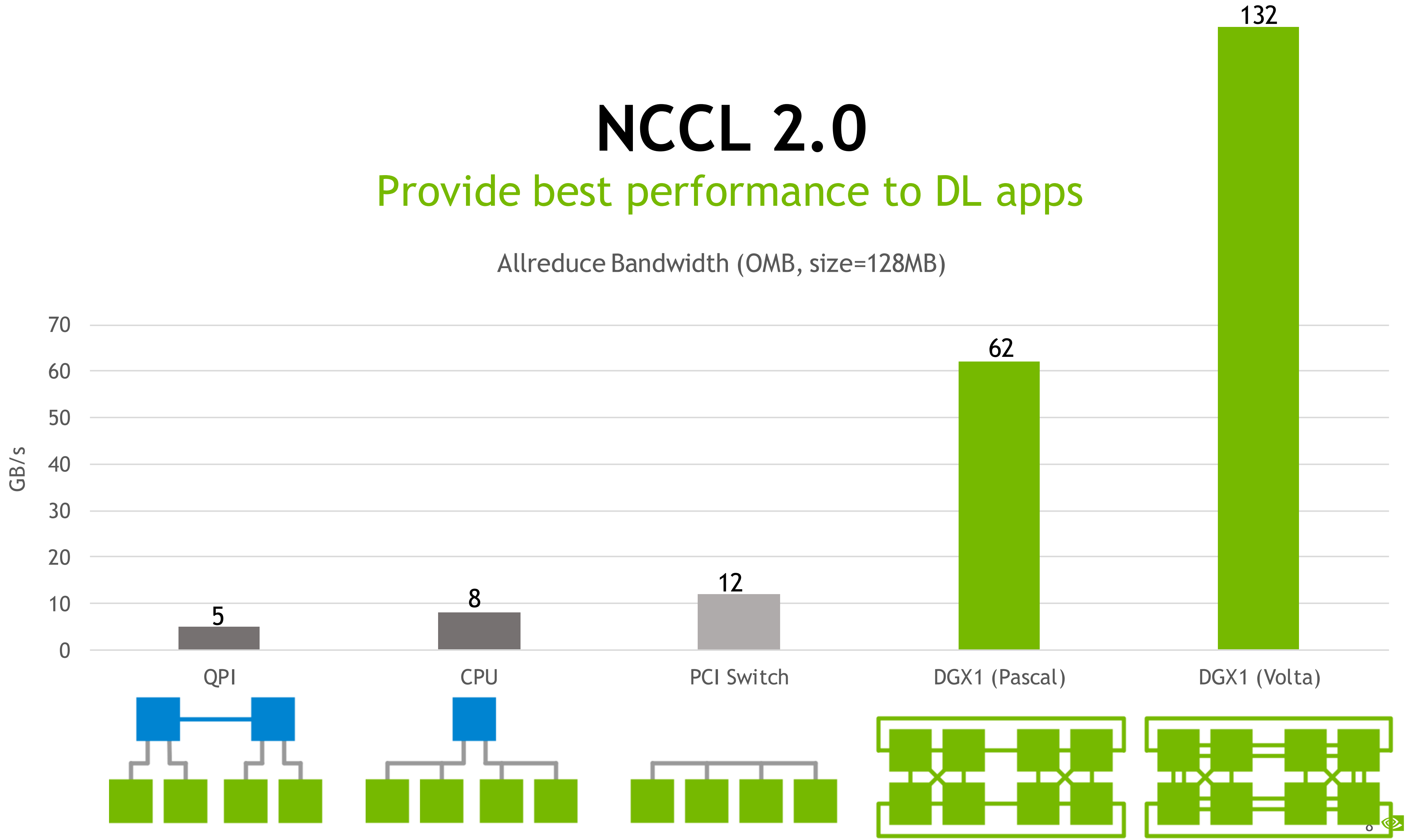
TIMELINE

NCCL history & roadmap



NCCL 2.0

Provide best performance to DL apps

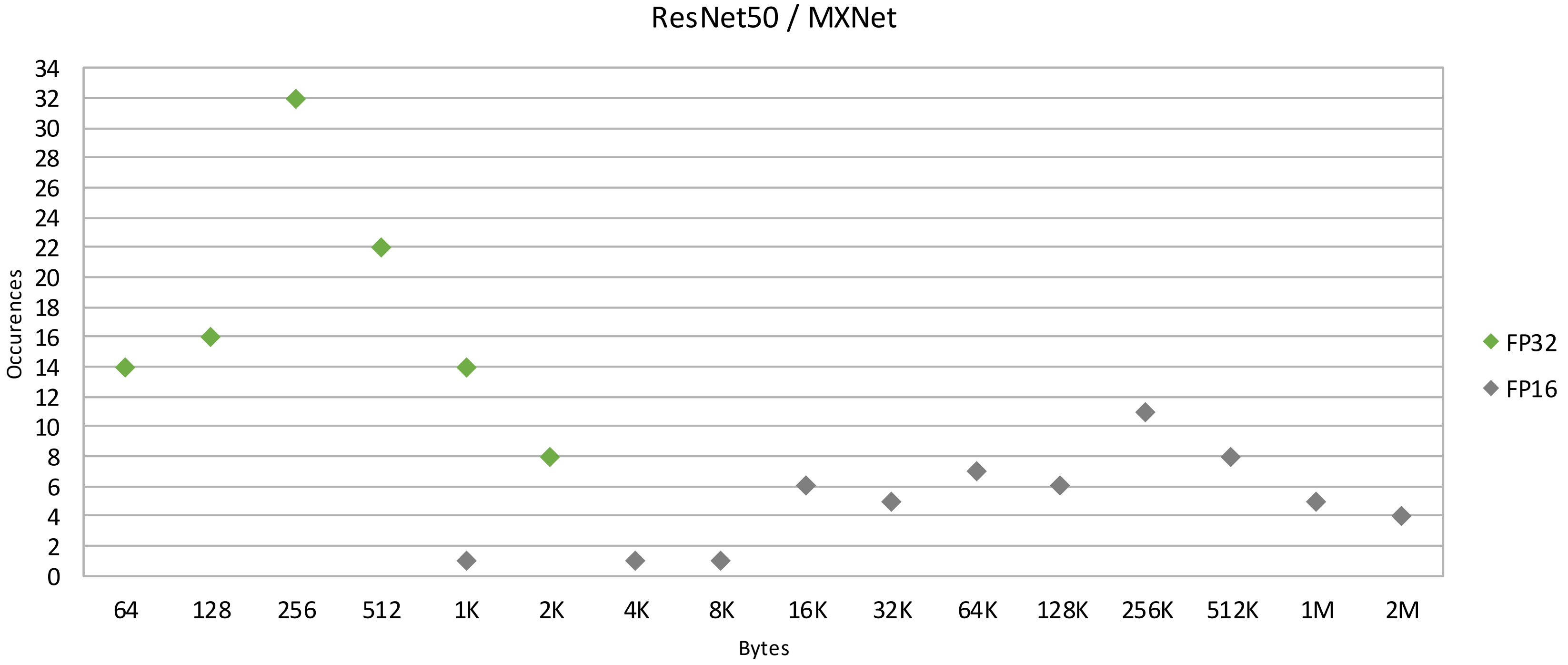


Allreduce Bandwidth (OMB, size=128MB)

NCCL 2.1

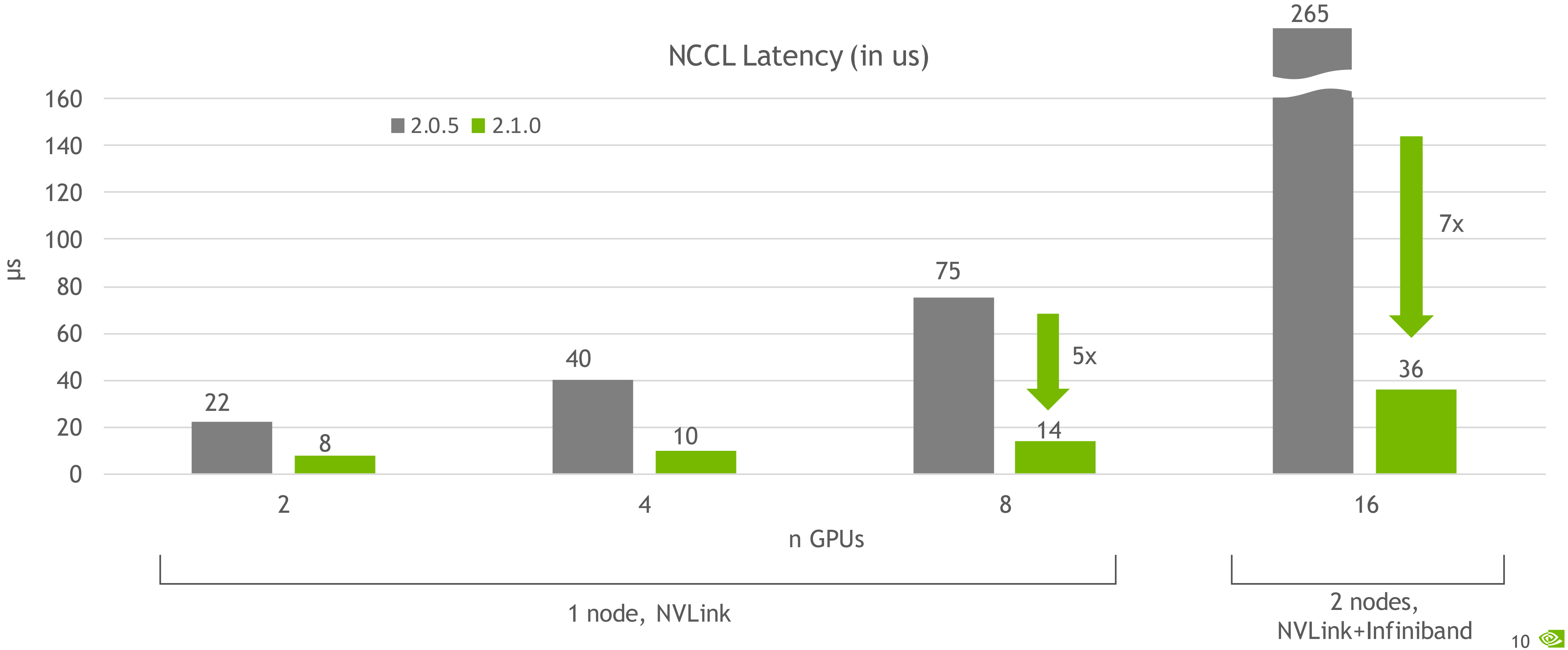
ResNet50 buffer size

Latency is important in some workloads, e.g. ResNet 50, in particular when reductions are done for each layer.



NCCL 2.1

Latency improvement

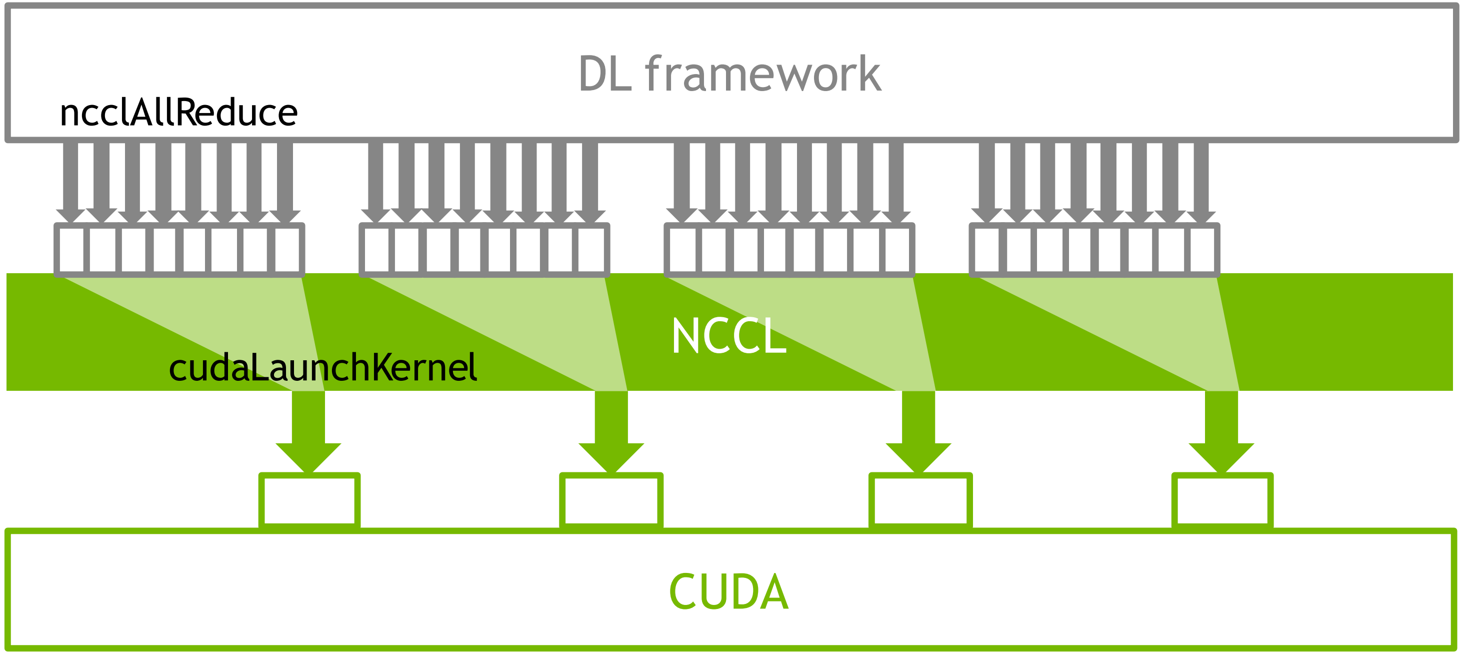


NCCL 2.2

Aggregated operations : principle

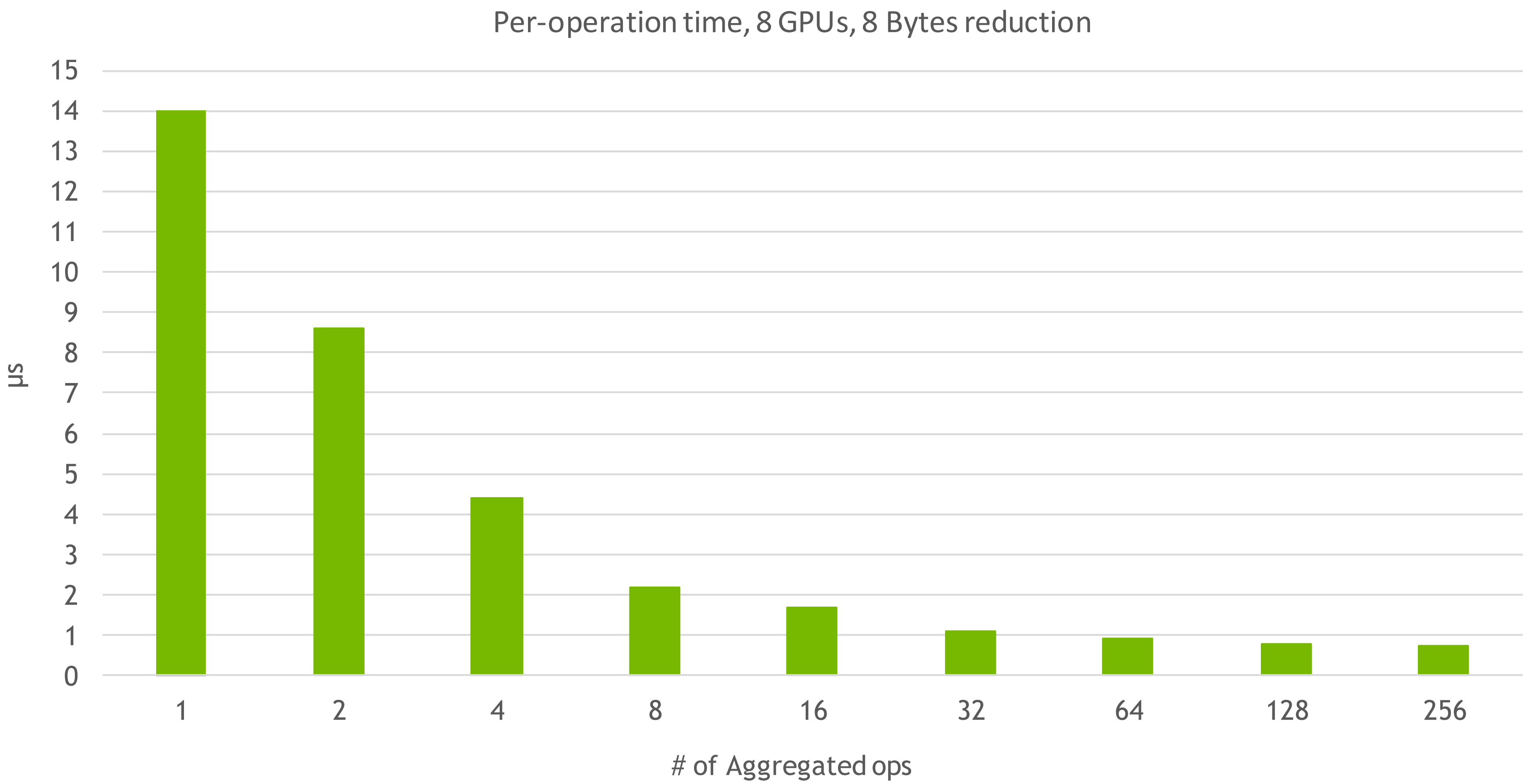
Principle : Merge multiple operations on the same CUDA device

- Pay the launch overhead only once (more operations per second)
- Use multiple NVLinks simultaneously (more bandwidth)



NCCL 2.2

Aggregated operations : overhead



NCCL 2.2

Aggregated operations : usage

Use ncclGroupStart() / ncclGroupEnd() around the NCCL operations we want to aggregate :

```
ncclGroupStart();
for (int op=0; op<nops; op++) {
  ncclAllReduce(
    layers[op].localGradients,
    layers[op].globalGradients,
    layers[op].gradientSize,
    ncclFloat, ncclSum, ncclComm, ncclStream);
}
ncclGroupEnd();
// All operations are only guaranteed to be posted on the stream after ncclGroupEnd
cudaStreamSynchronize(ncclStream);
```

NCCL 2.2

Aggregated operations : usage

Can be combined/nested with multi-GPU grouping :

```
ncclGroupStart();
for (int op=0; op<nops; op++) {
  for (int gpu=0; gpu<ngpus; gpu++) {
    ncclGroupStart();
    ncclAllReduce(
      layers[op].localGradients[gpu],
      layers[op].globalGradients[gpu],
      layers[op].gradientSize,
      ncclFloat, ncclSum, ncclComms[gpu], ncclStreams[gpu]);
    ncclGroupEnd();
  }
}
ncclGroupEnd();
// All operations are only guaranteed to be posted on the stream after the last ncclGroupEnd
for (int gpu=0; gpu<ngpus; gpu++)
  cudaStreamSynchronize(ncclStreams[gpu]);
```

NCCL 2.2

Aggregated operations : other uses

ReduceScatterV = Aggregation of multiple reductions operations

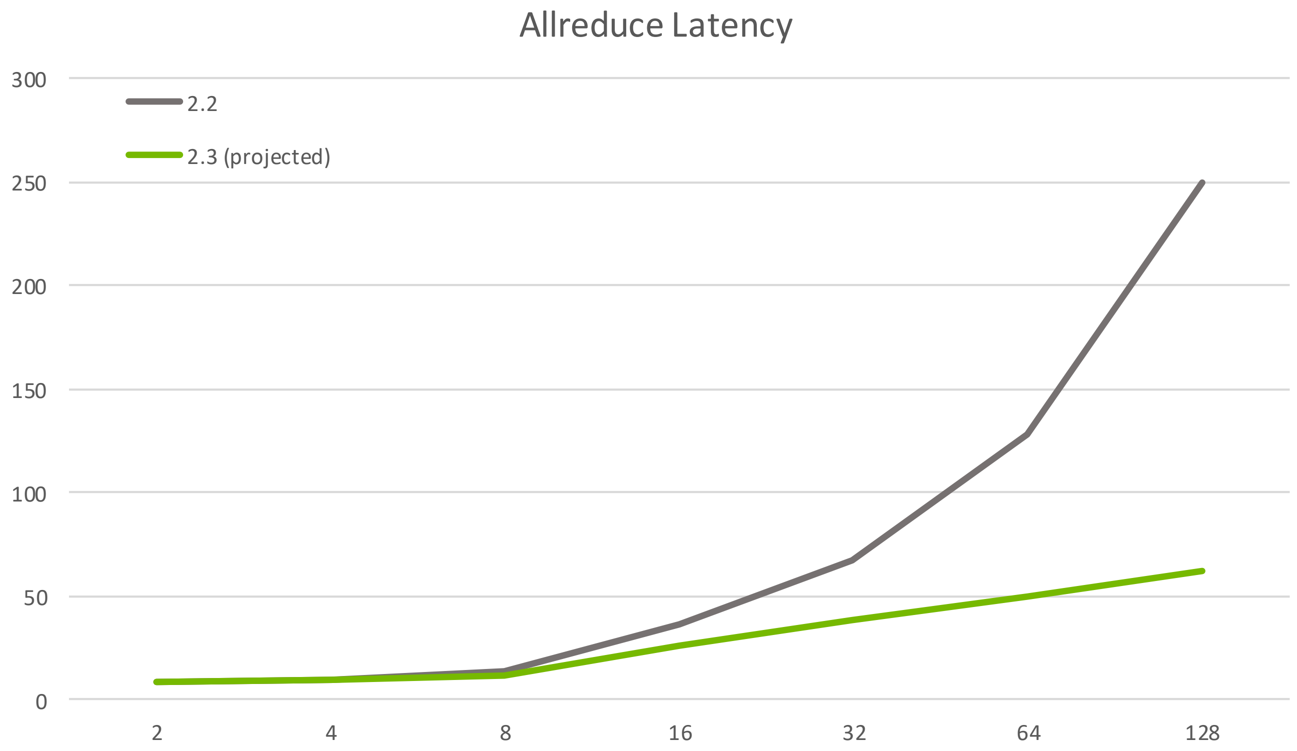
```
ncclGroupStart();
for (int rank=0; rank<nranks; rank++) {
  ncclReduce(sendbuff+offsets[rank], recvbuff+offsets[rank],
      recvcounts[rank], datatype, redOp, rank, comm, stream);
}
ncclGroupEnd();
```

AllGatherV = Aggregation of multiple broadcasts operations

```
ncclGroupStart();
for (int rank=0; rank<nranks; rank++) {
  ncclBroadcast(sendbuff+offsets[rank], recvbuff+offsets[rank],
     recvcounts[rank], datatype, rank, comm, stream);
}
ncclGroupEnd();
```

NCCL 2.3

Large scale algorithms

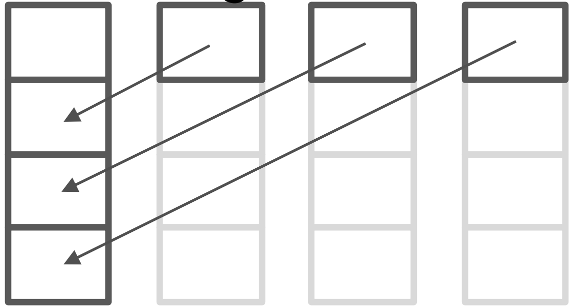


NCCL 2.4

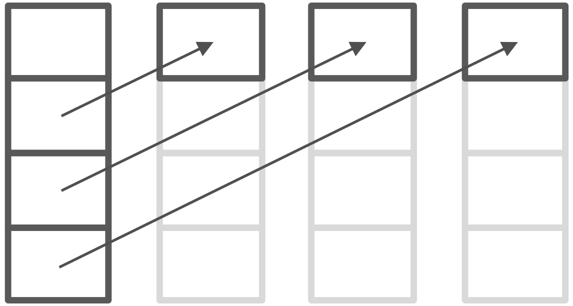
Point-to-point primitives

Send / Receive , Scatter[v],Gather[v],Alltoall[v,w], neighbor collectives, …

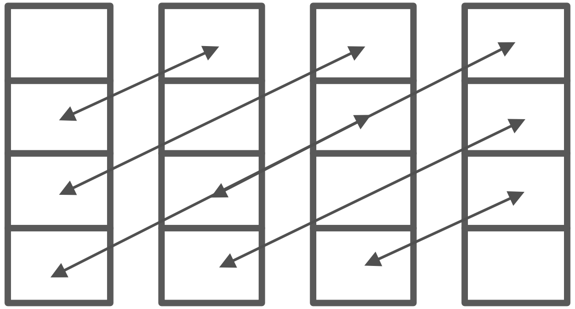
gather



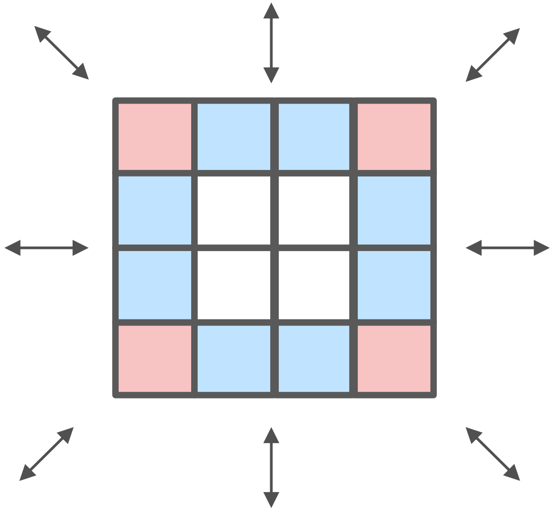
scatter



alltoall



Neighbor



NCCL

Summary

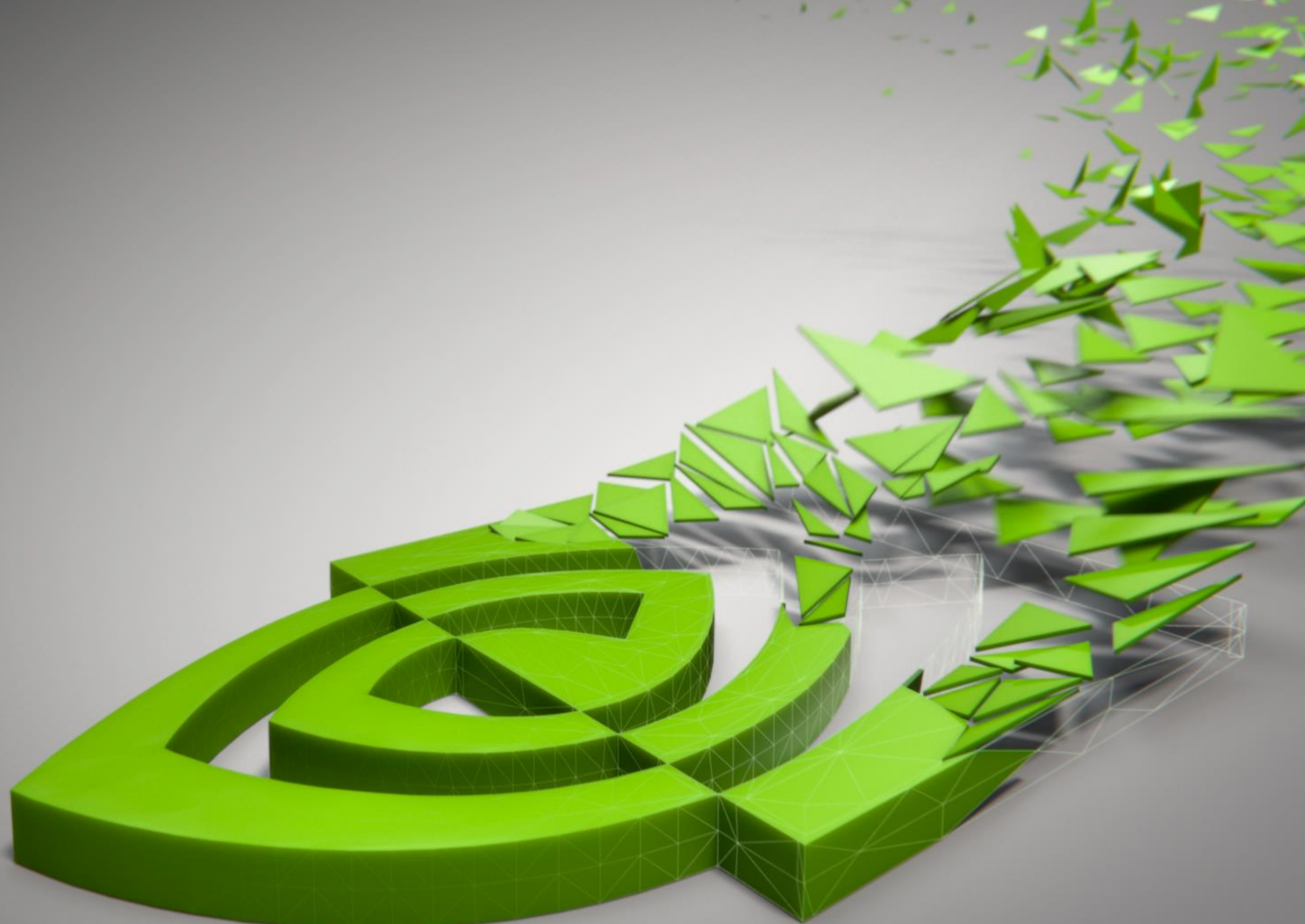
Optimized inter-GPU communication for DL and HPC

- Optimized for all NVIDIA platforms, most OEMs and Cloud
- Scales to 100s of GPUs, targeting 10,000s in the near future.

Aims at covering all communication needs for multi-GPU computing.

Only relies on CUDA. No dependency on MPI or any parallel environment.

More questions ? Connect with the Experts : NCCL Wed 28, 3pm



NVIDIA