SCALING GPU WORKLOADS ON DATACENTER CLUSTERS USING GPU AND MPI AWARE CONTAINERS

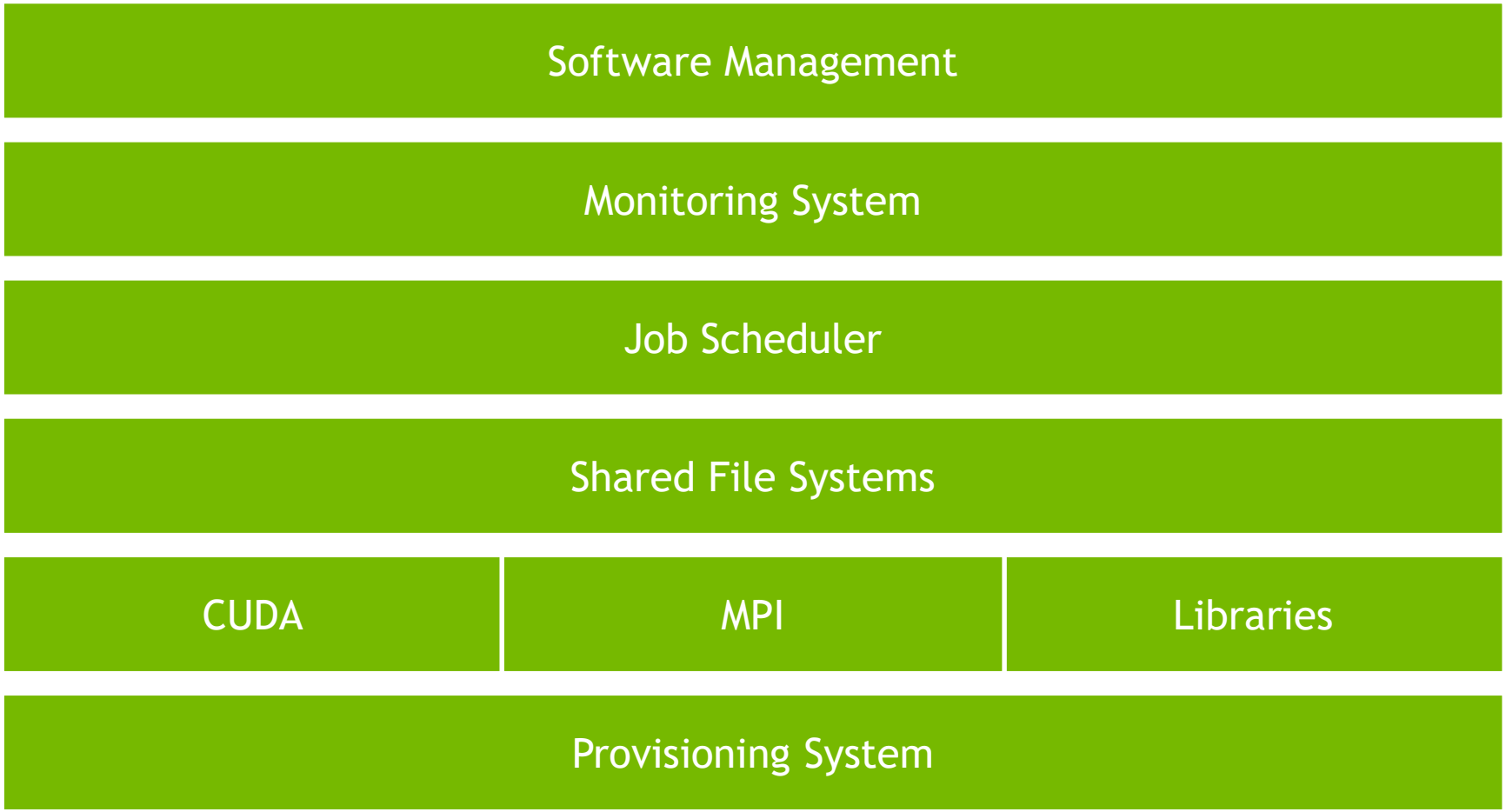
NVIDIA

ABSTRACT:

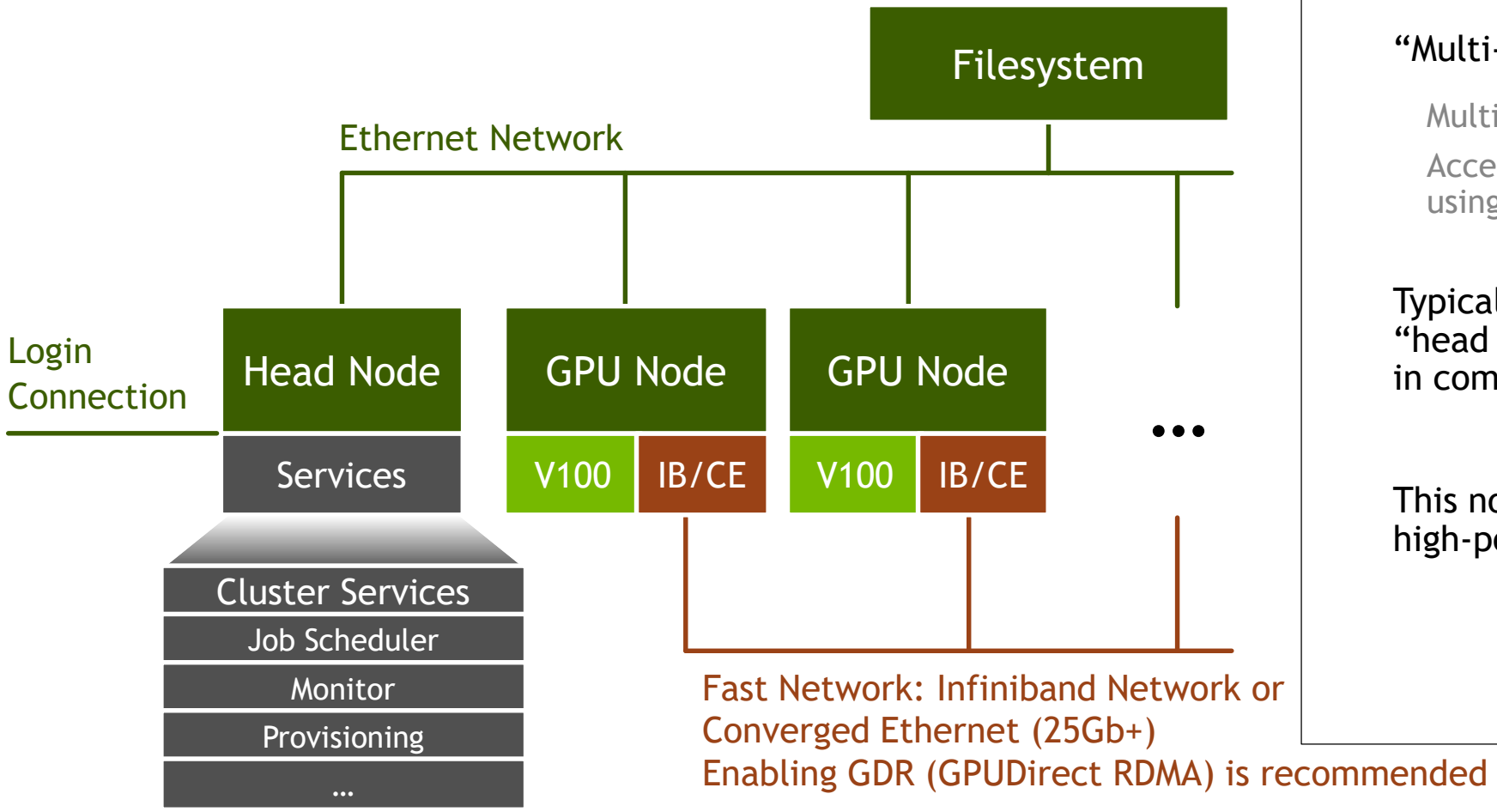
Graphical Processing Units (GPUs) are critical to modern HPC (high performance compute) and ML/DL (machine learning/deep learning) computing workloads. Requirements of engineers and scientists can easily scale to petaflops whereas the current state of the art GPU performance is in teraflops range. Continuing in the tradition of cluster computing GPUs are scaled to petaflops performance by using traditional technologies such as MPI (message passing interface), high performance interconnects (such as Infiniband and RoCE), and RDMA (remote direct memory access). The presentation will explore the challenges involved with multi-node scaling and how containerization is helping manage the software complexities of running workloads on clusters. An overview will be presented of how to orchestrate multinode workflows using GPU hardware and MPI using containers. The containers technology focus in the presentation will be on docker, singularity, HPC resource schedulers such as SLURM/PBS/etc., and container orchestration platforms such as Kubernetes.

GPU Cluster Management Stack

CLUSTER SERVICES



A MULTI-USER GPU CLUSTER



"Multi-user mode":

- Multiple users share cluster
- Access to GPU nodes is allocated using job scheduler

Typically requires at least one "head node" which doesn't take part in compute, runs cluster services

This node doesn't have to be high-powered; can often be a VM

DATA CENTER GPU MANAGER

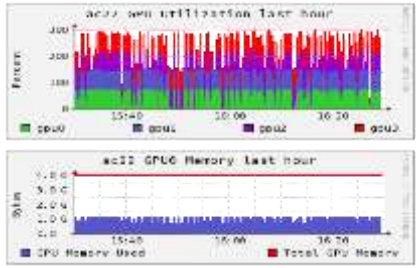
Integrated into Leading Industry Tools for HPC

Supports Only Tesla Accelerators

DATA CENTER GPU MANAGEMENT

Enterprise-Grade Management Tool for Operating the Data Center

Device Management



Per GPU Configuration & Monitoring

- Device Identification
- Board Monitoring
- Clock Management

All GPUs Supported

Data Center GPU Manager (Tesla GPUs Only)

Active Health Monitoring

Runtime Health Checks
Prologue Checks
Epilogue Checks

Diagnostics & System Validation

Deep HW Diagnostics
System Validation Tests

Policy & Group Config Management

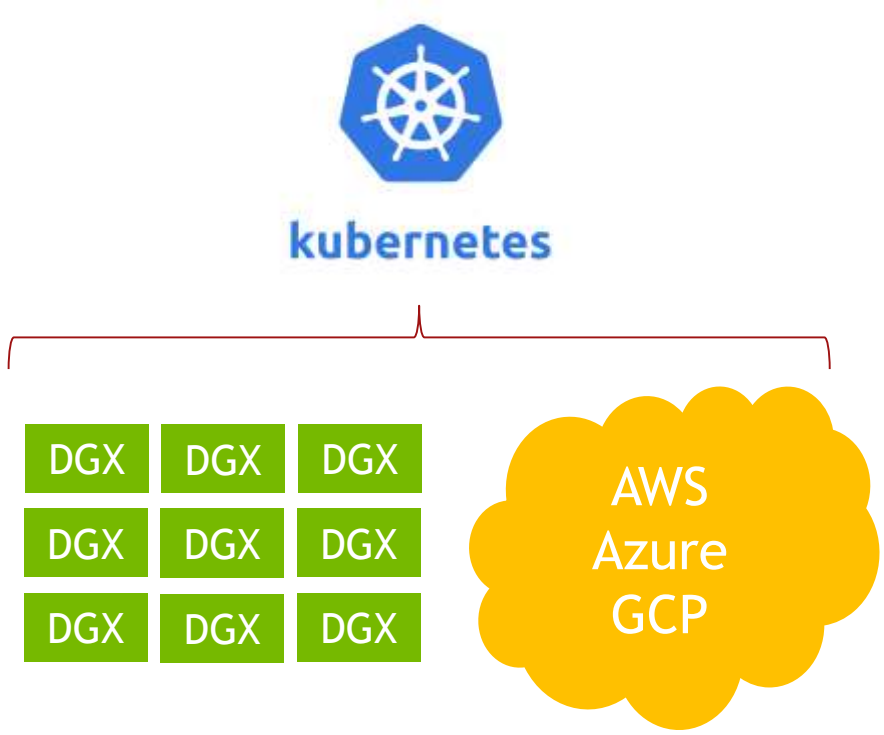
Pre-configured policies
Job level accounting
Stateful configuration

Power & Clock Mgmt.

Dynamic Power Capping
Synchronous Clock Boost

KUBERNETES

What is it?



Think of Kubernetes as a cloud and container oriented resource manager and scheduler.

The cluster's servers can be on-prem, in the cloud, or a mix (hybrid)

Use Kubernetes to manage nodes in the cluster, administer user access, launch jobs as containers, expose running services externally, and more

EX: KUBERNETES VS SLURM

Cloud vs HPC

Service oriented

Containerized

Meant to scale

Distributes load

Keeps applications running

Batch job oriented

Bare metal

Efficiently allocates resources

Fine-grained user control

Maximizes system utilization

NVIDIA GPU COMMUNICATION TECHNOLOGIES

NVIDIA GPUDIRECT™

Overview

Using GPUDirect, multiple GPUs, third party network adapters, solid-state drives (SSDs) and other devices can directly read and write CUDA host and device memory

Eliminates unnecessary memory copies

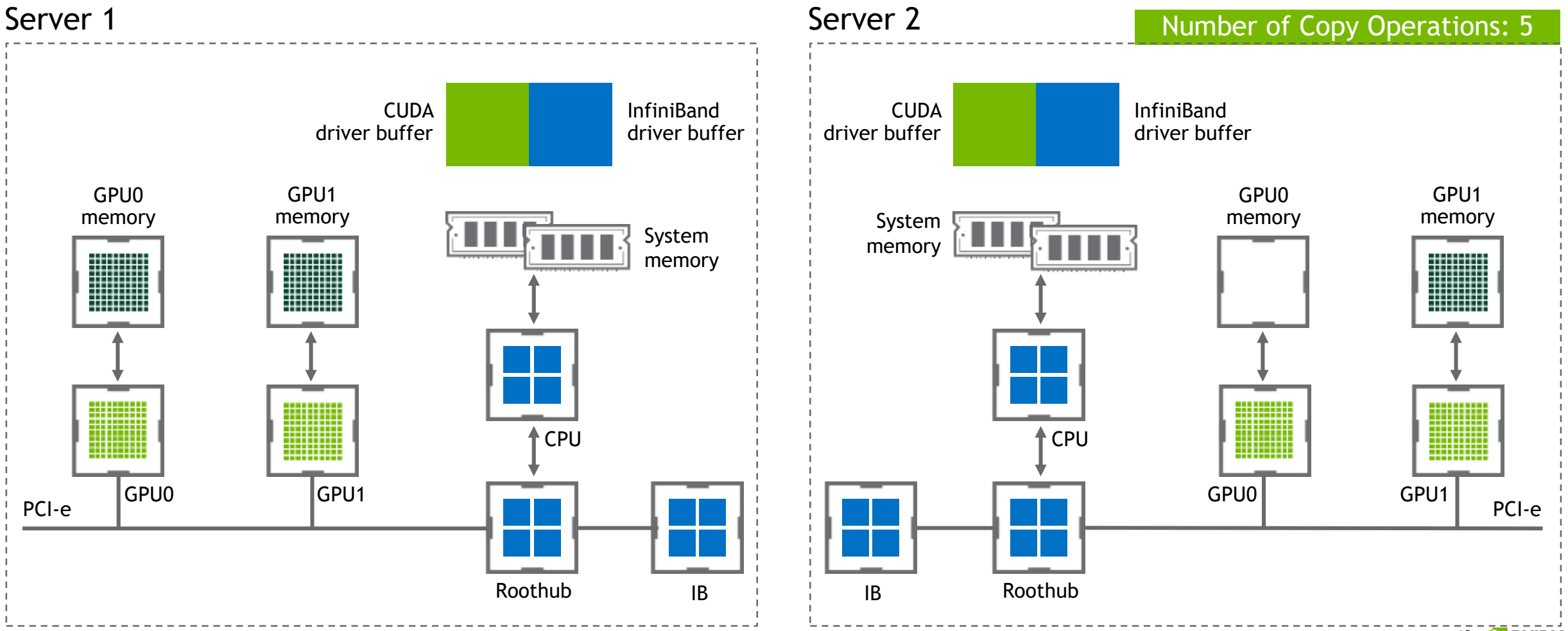
Dramatically lowers CPU overhead

Reduces latency

The result is significant performance improvements in data transfer times for applications running on NVIDIA Tesla™ products

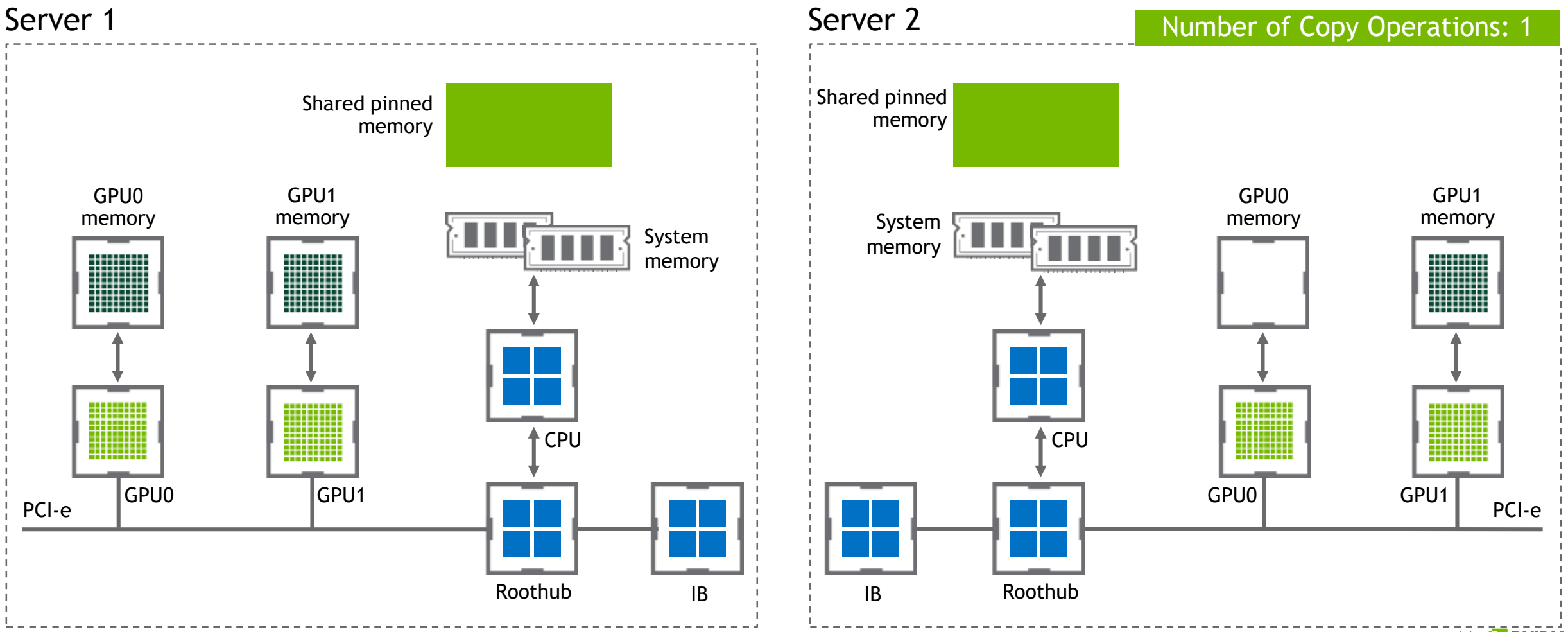
NVIDIA GPUDIRECT™

Worst Case Without GPUDirect RDMA



NVIDIA GPUDIRECT™

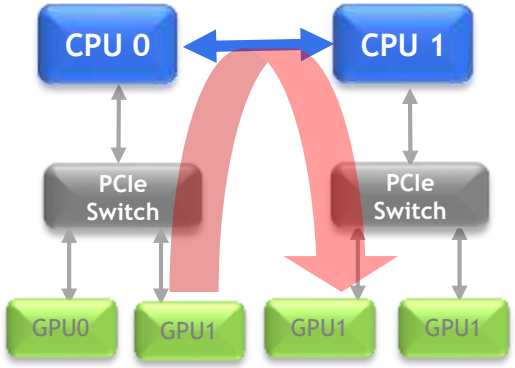
Best Case With GPUDirect RDMA



WHY NVLINK?

PCIE/QPI Bottleneck

PCIe Gen 3: 16GB
NVLINK : 80-300GB/s



Higher Throughput,
More Jobs/day

Seismic Analysis,
Deep Learning

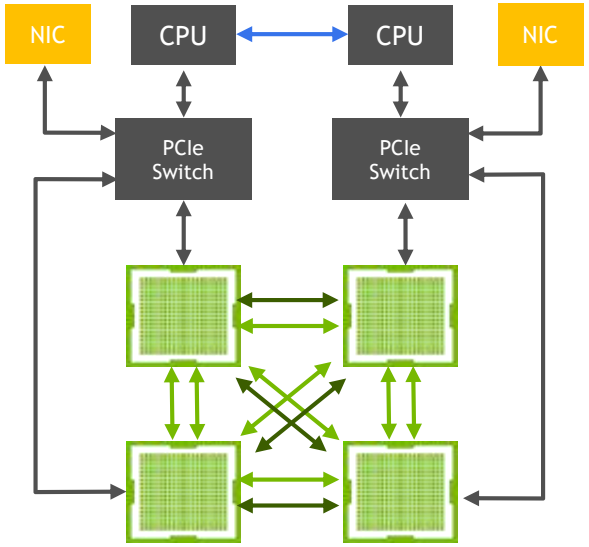
Higher Performance for
Larger Problems

Life Sc*, Particle Physics,
Material Sc, Climate

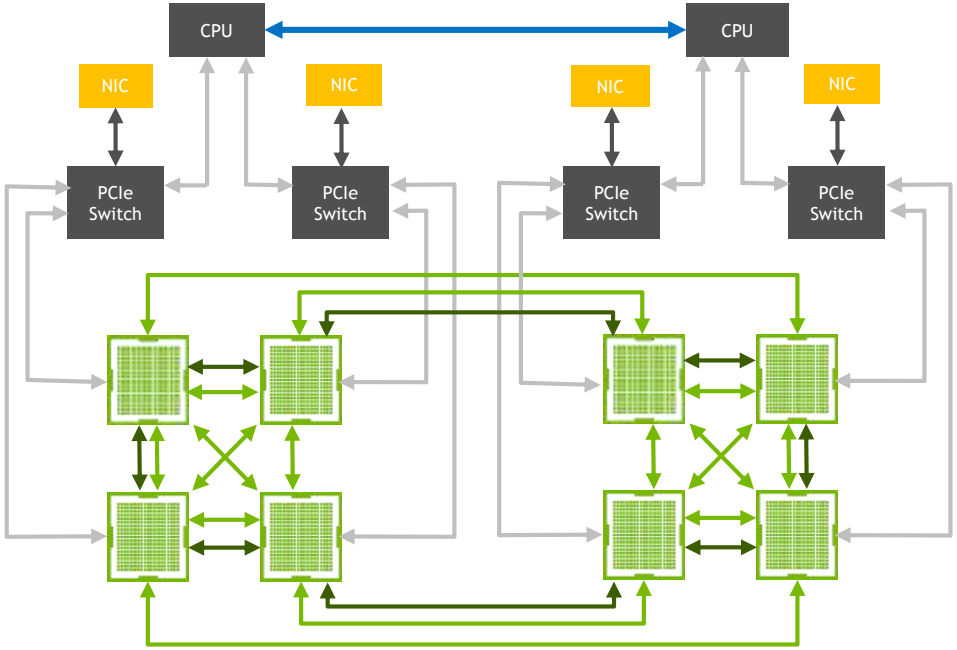
More cost effective

Fewer Nodes,
Networking HW

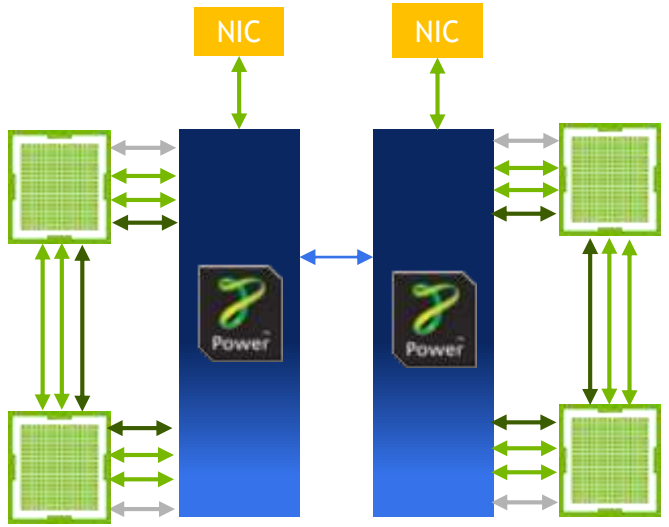
SERVER DESIGNS WITH V100 NVLINK



4x V100-SXM2
Dual x86

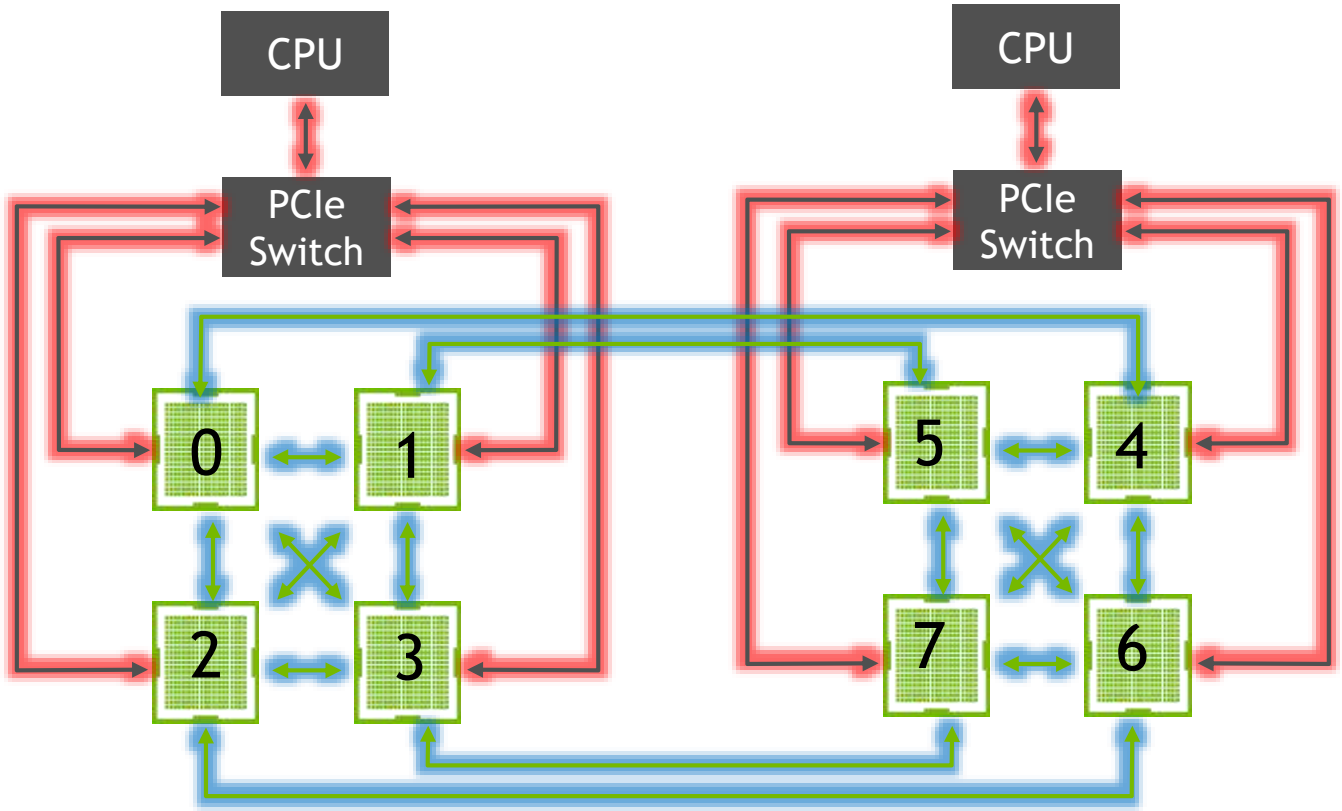


8x V100-SXM2
Dual x86



Dual Power9+,
4x V100-SXM2

DL DATA PARALLELISM - NVLINK

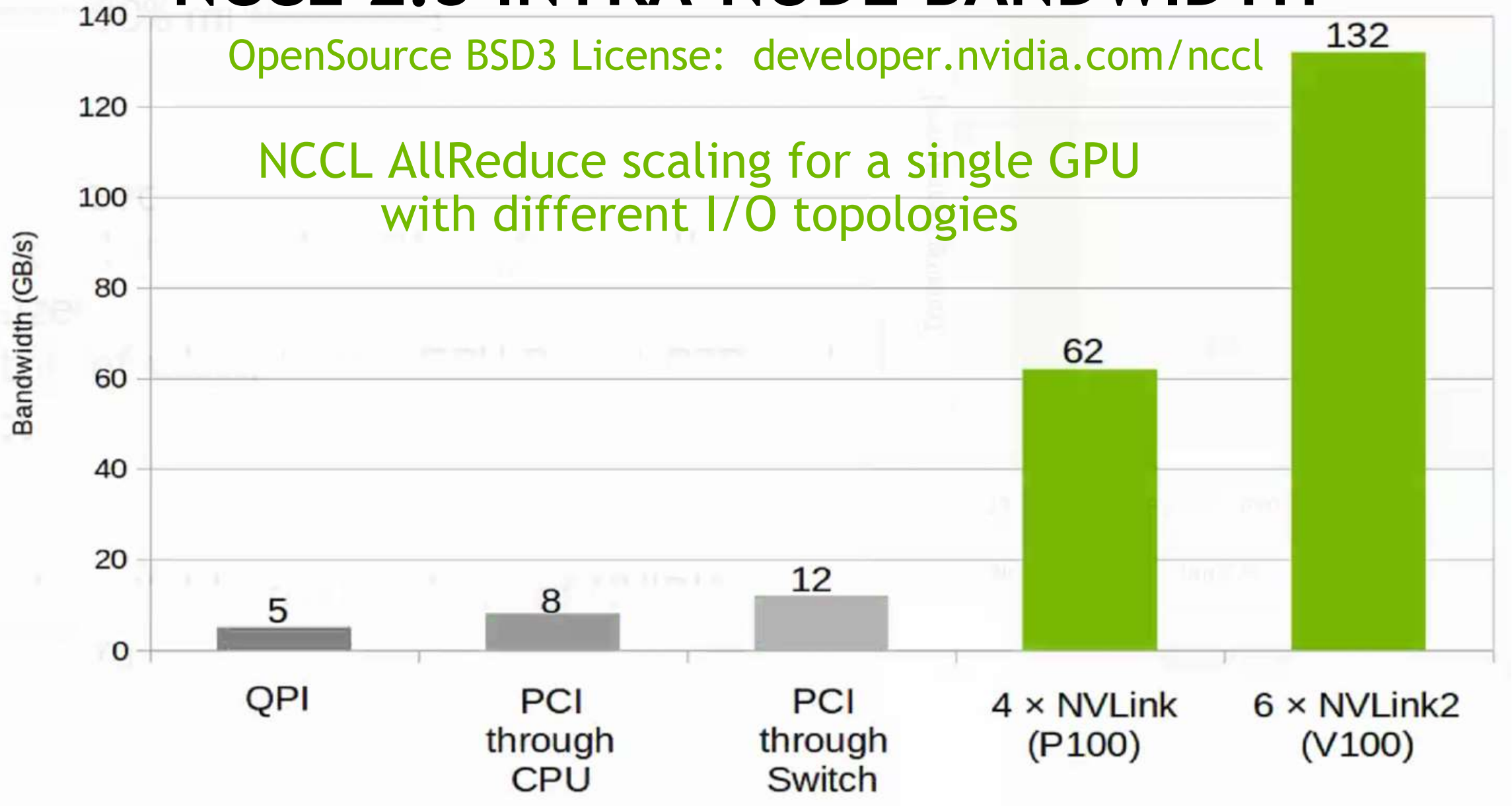


No sharing of communication resources: No congestion

NCCL 2.3 INTRA-NODE BANDWIDTH

OpenSource BSD3 License: developer.nvidia.com/nccl

NCCL AllReduce scaling for a single GPU
with different I/O topologies



NVSWITCH

NVLink Performs physical, datalink & transaction layer functions

Forwarding Determines packet routing

Crossbar (non-blocking) Schedules traffic flows to outputs

Management Configuration, errors, monitors

Features:

- 18 NVLink ports
  - @ 50 GB/s per port
  - 900 GBs total
- Fully connected crossbar
- x4 PCIe Gen2 Management Port
- GPIO, I2C

Transistor count:

- 2 billion

Package:

- 47.5 x 47.5mm
- 1937 Ball @ 1mm pitch

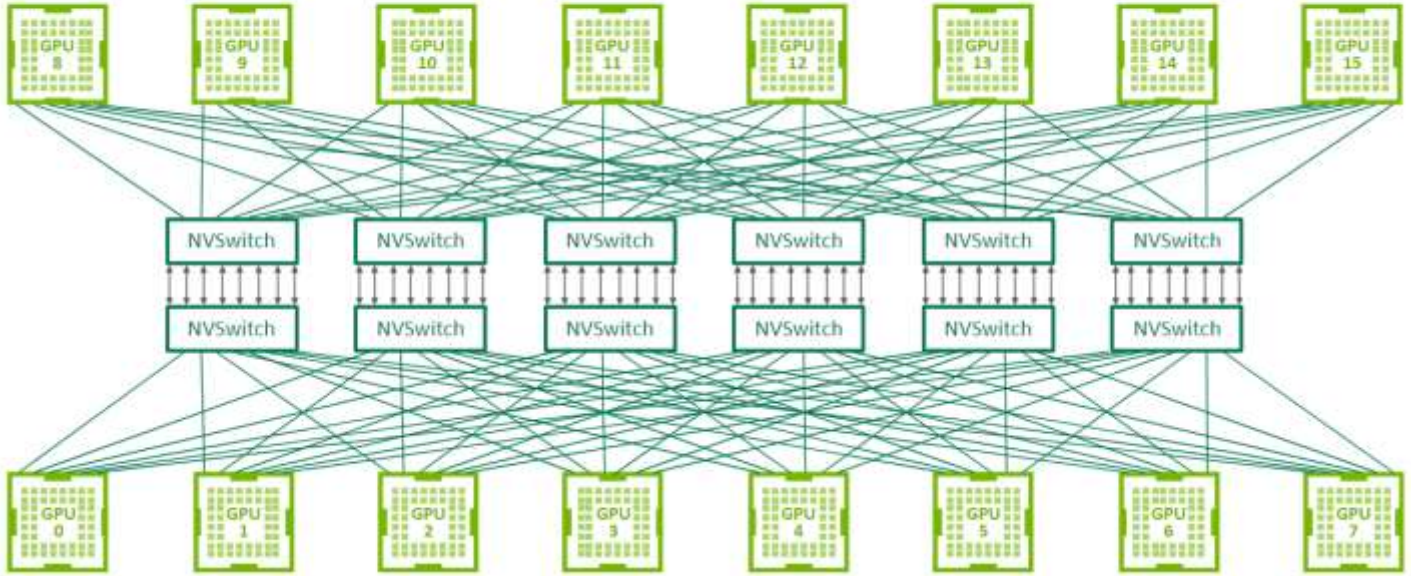
BROADCAST ON DGX-1

Ring Scatter without NVSwitch

BROADCAST ON DGX-2

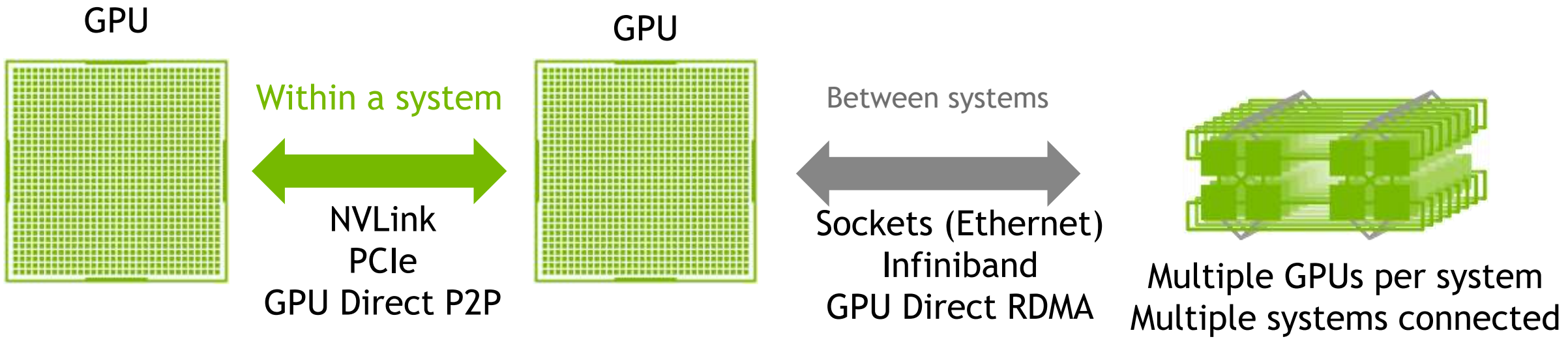
Direct Broadcast NVSWITCH

FULL NON-BLOCKING BANDWIDTH



NCCL: NVIDIA COLLECTIVE COMMUNICATION LIBRARY

A multi-GPU communication library



NCCL legacy algorithm : **ring**
- **Linear** latency
- **Full bandwidth**

NCCL 2.3
Scales to 100s
of GPUs

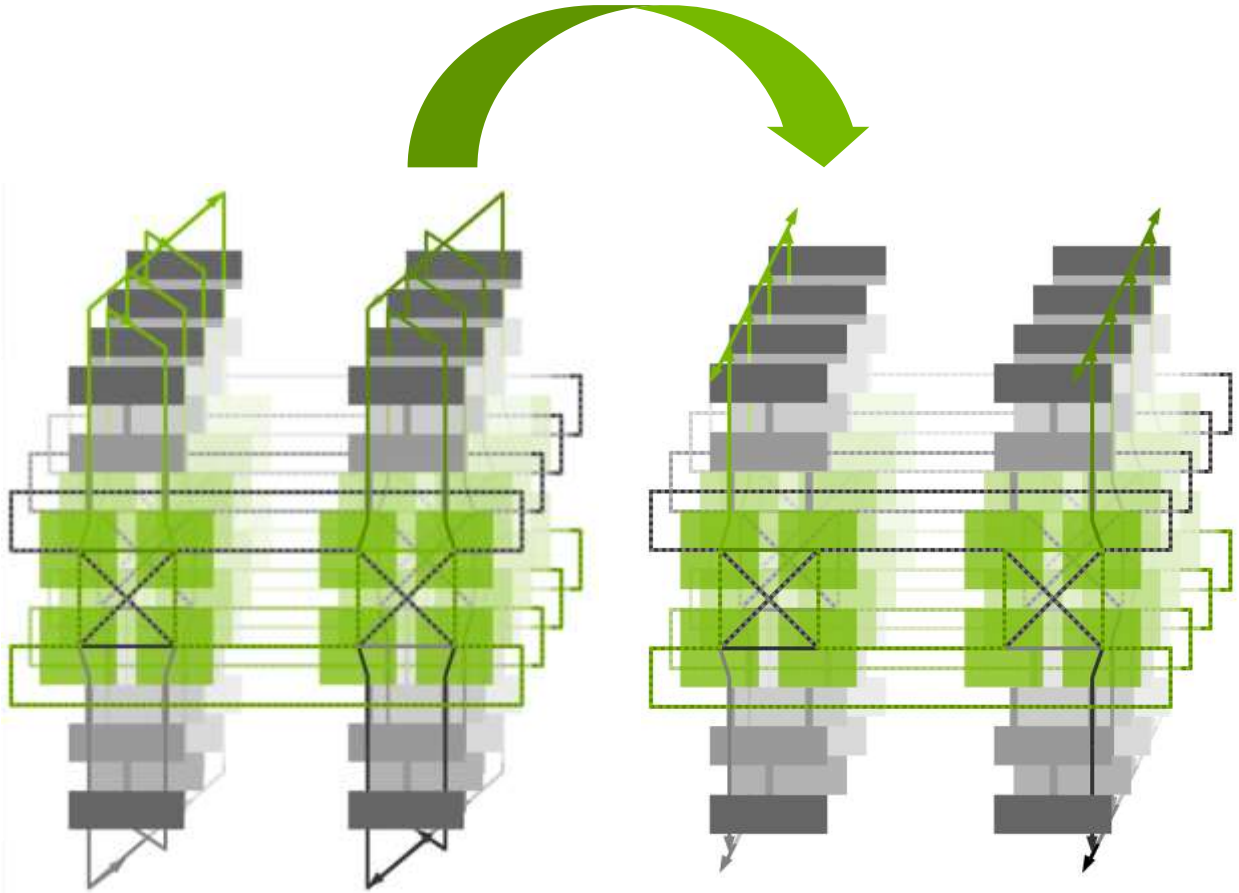
New base algorithm (allreduce only, for now) : **double binary tree**
- **Logarithmic** latency
- **Full bandwidth**

NCCL 2.4
Scales to 10,000s
of GPUs

Blog: https://devblogs.nvidia.com/massively-scale-deep-learning-training-nccl-2-4/

NCCL 2.4

From rings to hierarchical trees



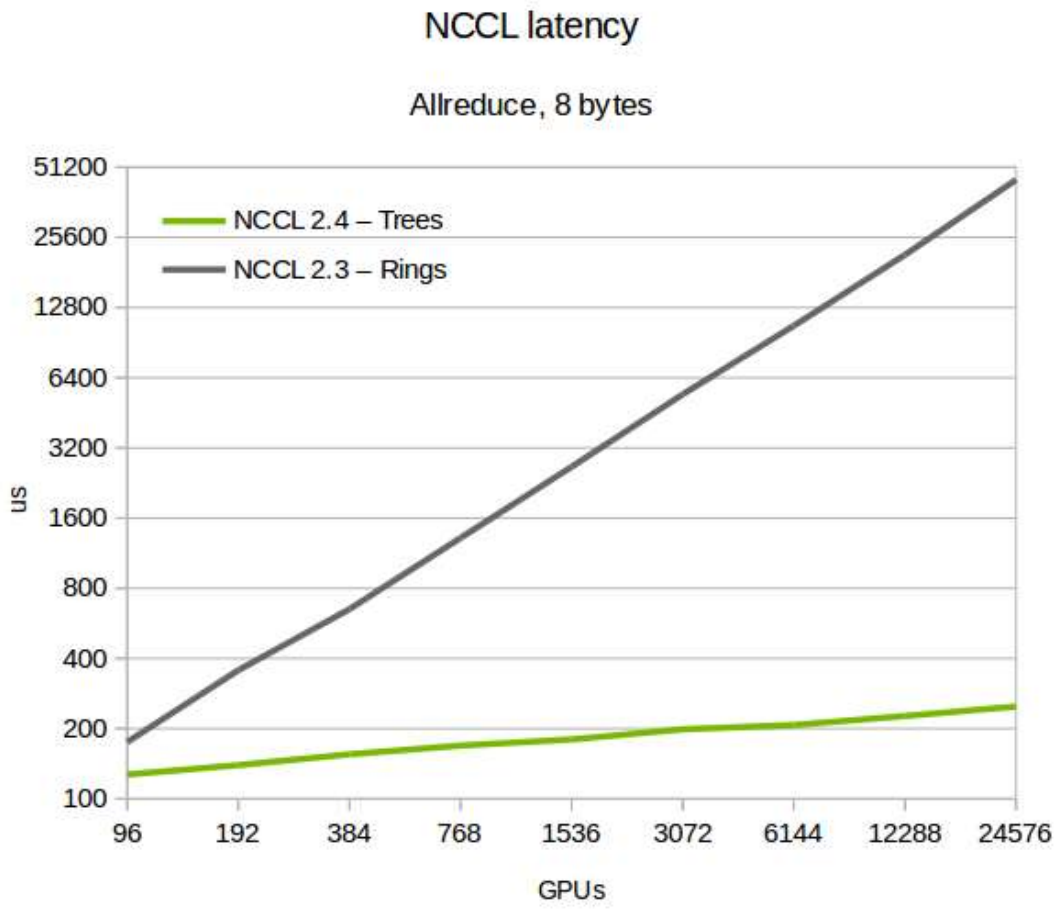
Multiple rings become multiple trees.

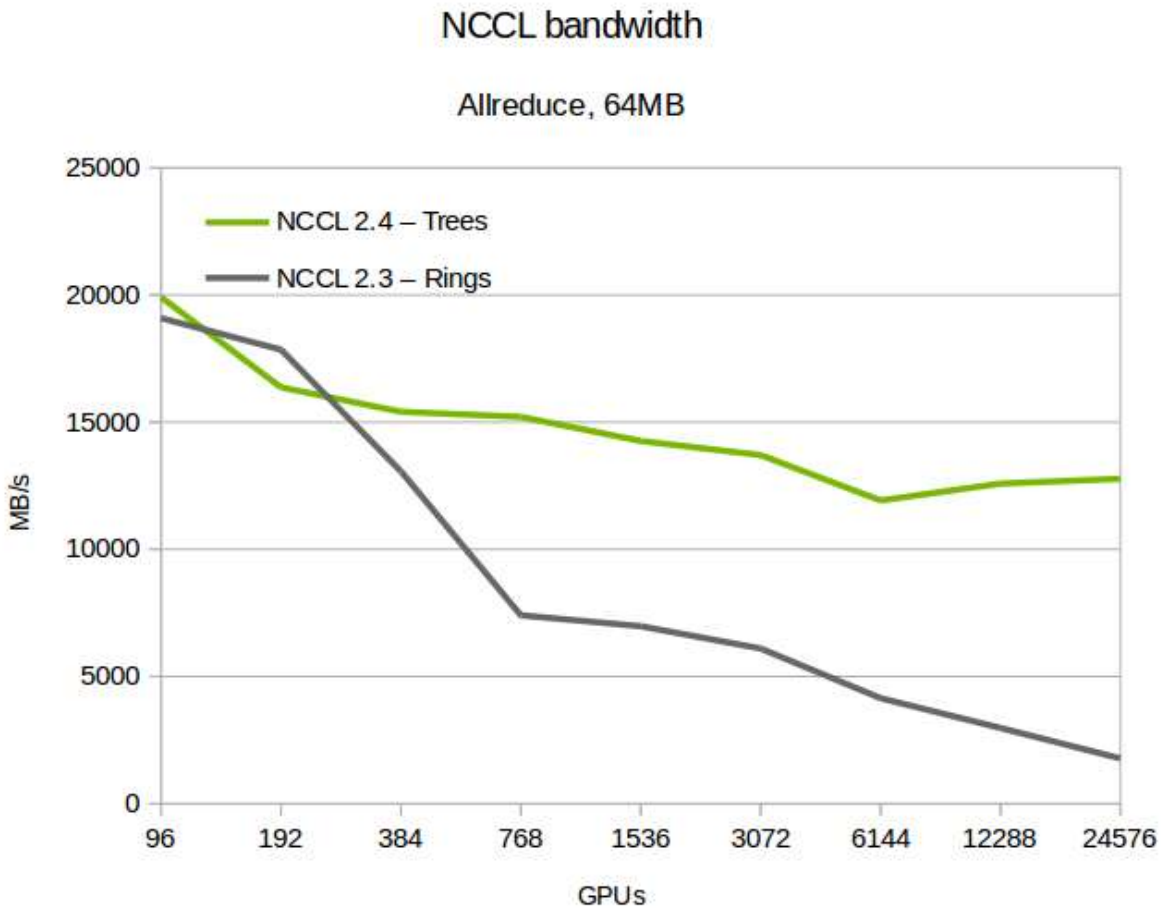
Intra-node still aggregating multiple NVLinks bandwidth.

NICs still communicating along planes.

TREES VS. RINGS

Summit, 4096 x 6 V100 (IBM P9)

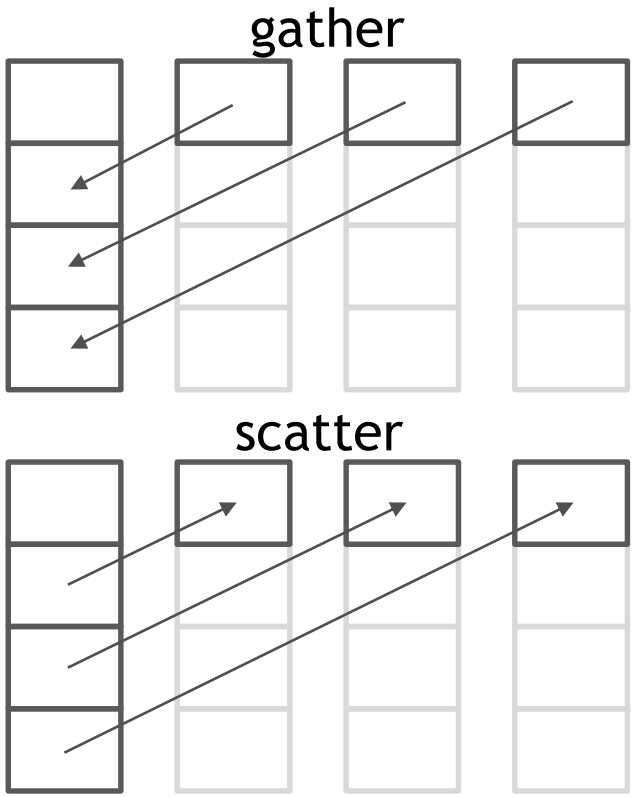


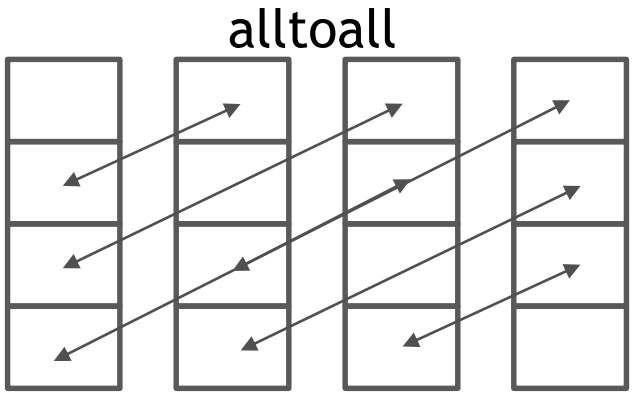


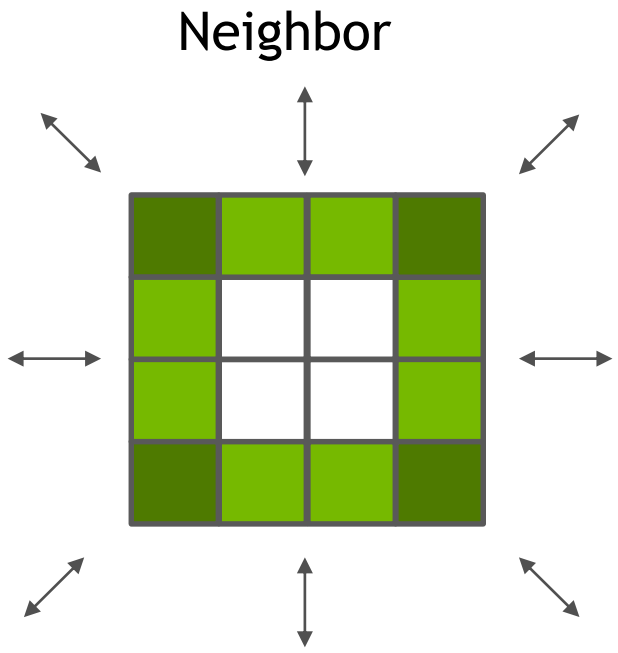
FUTURE

Point-to-point communication

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





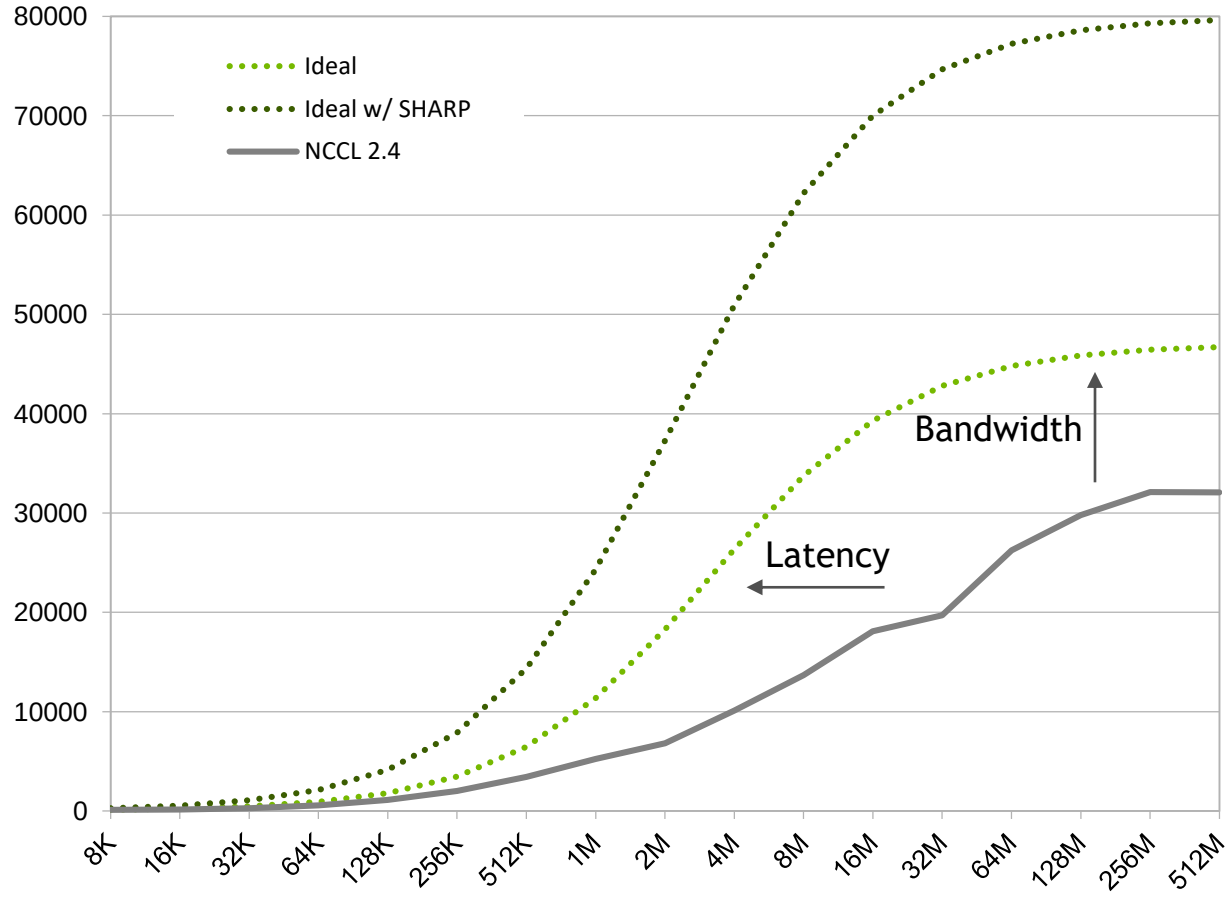


FUTURE

Further optimization

Continue to improve performance of trees
- Latency optimization for inter-node
- Peak bandwidth
- Bandwidth at scale

Add support for accelerated network collectives e.g. SHARP (1,2)



(1) http://www.mellanox.com/page/press_release_item?id=2151
(2) S9268 - Pushing the Limits of AI with NVIDIA GPUs and Mellanox Interconnect, Wednesday, Mar 20, 3:00 PM - 03:50 PM

NCCL
Summary

Optimized inter-GPU communication for DL and HPC
- Optimized for all NVIDIA platforms, most OEMs and Cloud
- Scales to ~~100s of GPUs, soon to~~ 10,000s of GPUs.

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

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

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

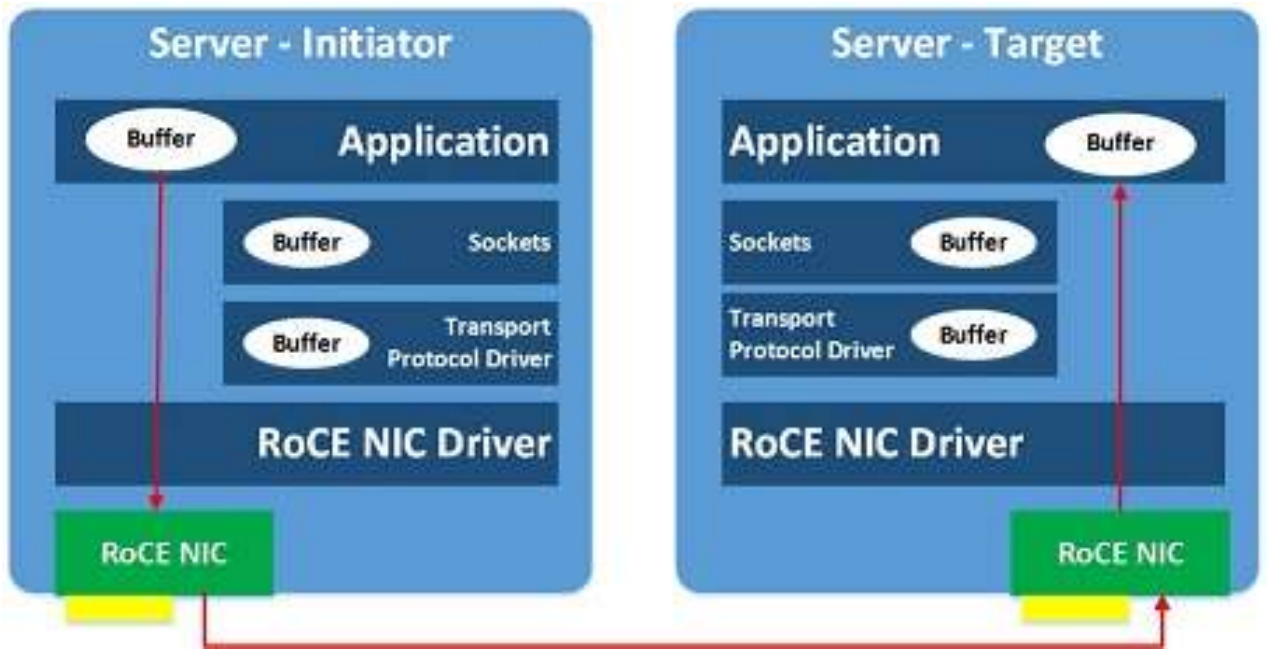
HIGH PERFORMANCE NETWORK COMMUNICATION TECHNOLOGIES

RDMA ENABLED NETWORKS BASICS

Popular networks: Infiniband, RoCE, iWARP, Omni-Path, Cray/uGNI

Access from the memory of one host (storage or compute) to the memory of another host without involving the remote Operating System and CPU, boosting network and host performance with lower latency, lower CPU load and higher bandwidth.

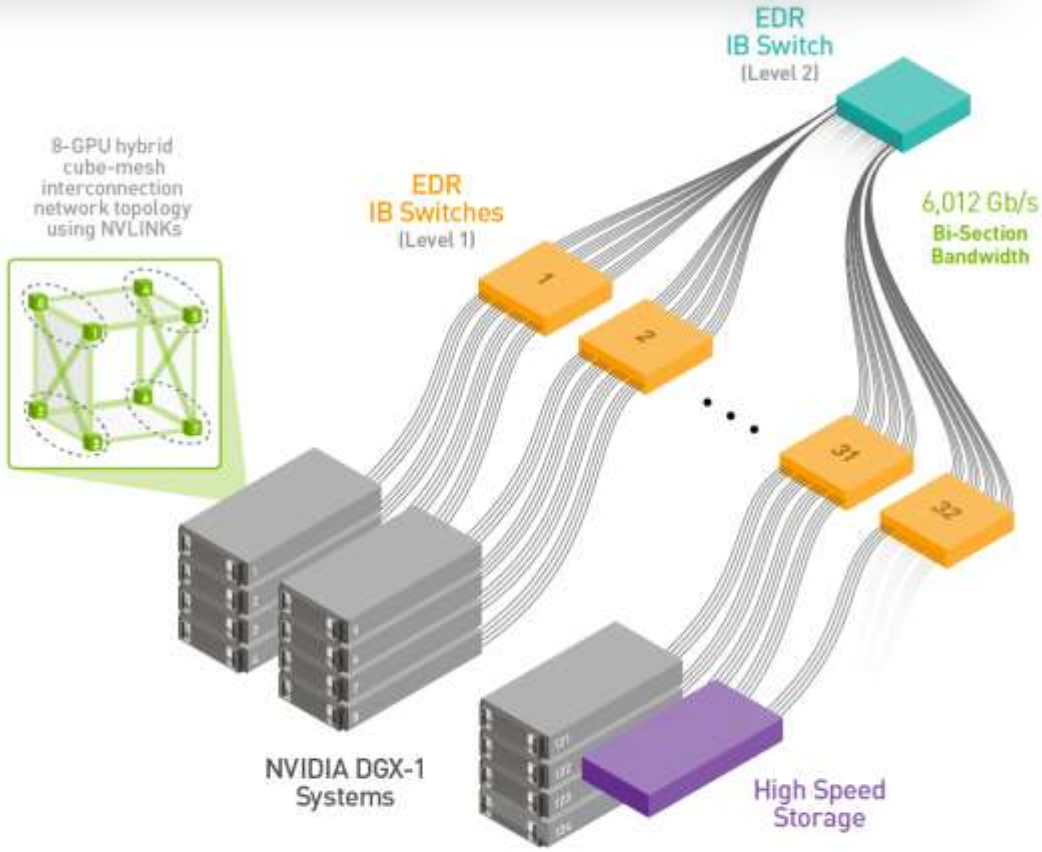
- Zero-copy – Data sent/recvd between buffers without network layers involvement.
- Kernel bypass – data transfers directly from user-space without kernel involvement.
- No CPU involvement – Applications access remote memory without consuming CPU in remote server.



* Diagram borrowed from Mellanox site

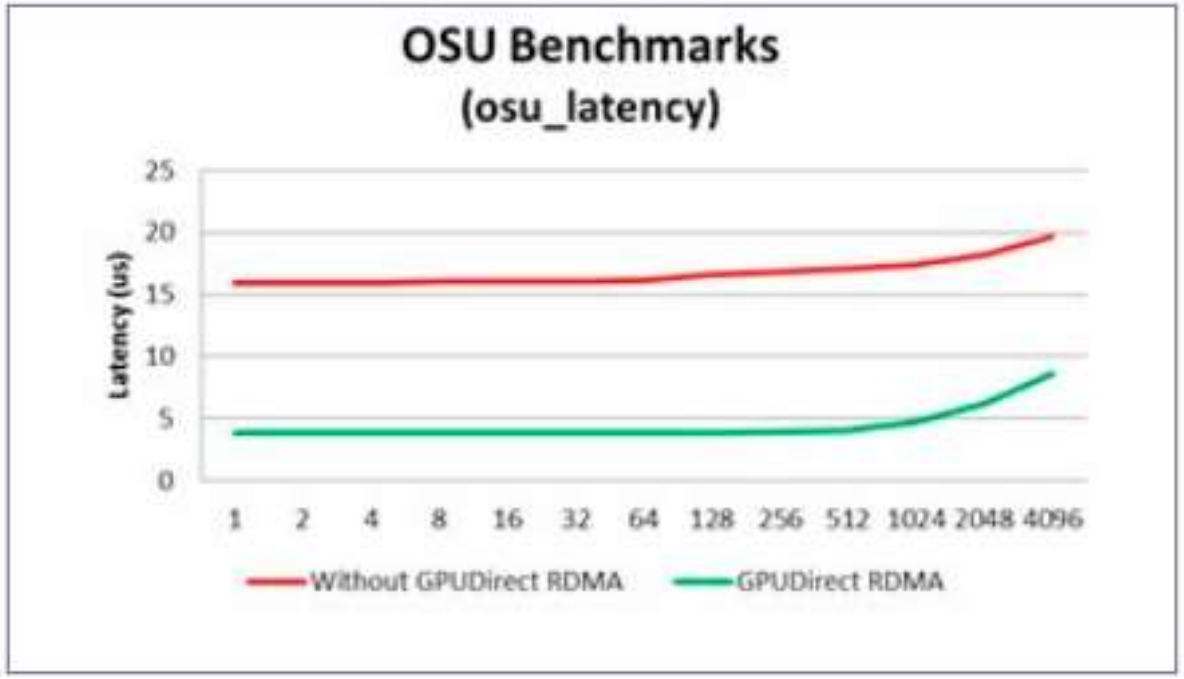
NETWORKING TOPOLOGY

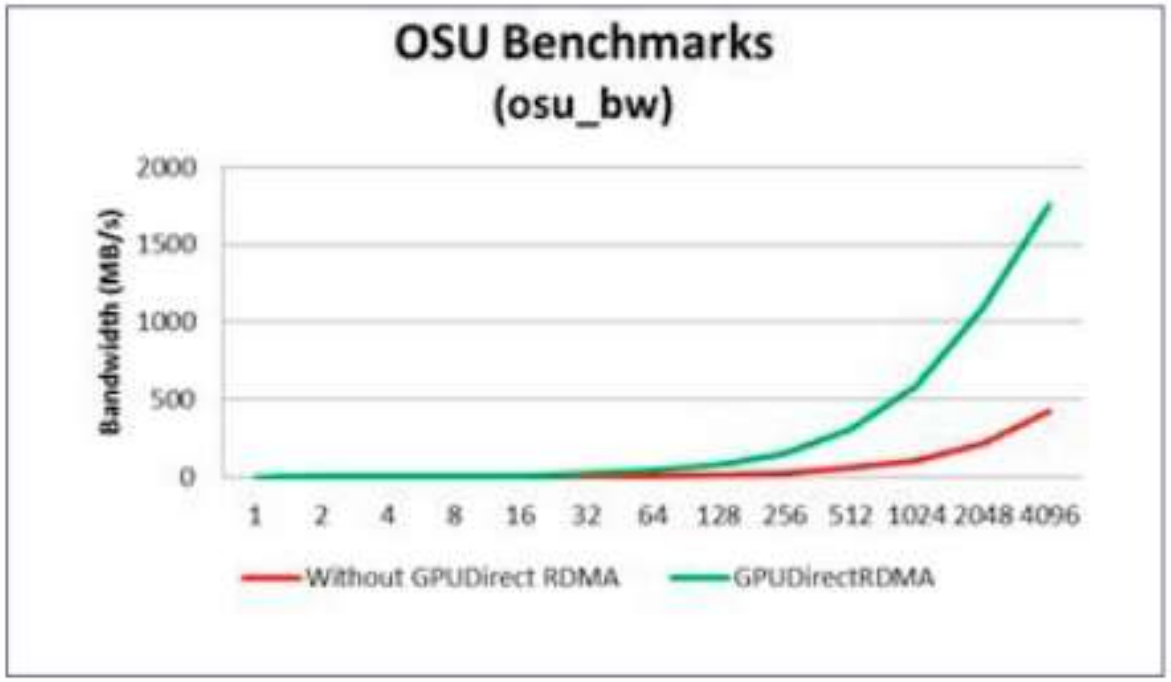
- Ingest data as fast as possible
- Pass data rapidly between nodes across cluster
- Similar to HPC networking architecture
- InfiniBand = ultra high bandwidth, low latency
- Two-tier network with root and leaf switches
- any to any connectivity with full bi-section bandwidth & minimum contention between nodes



RDMA IN COMBINATION WITH GPUS

GPU-to-GPU internode MPI latency and bandwidth improvement

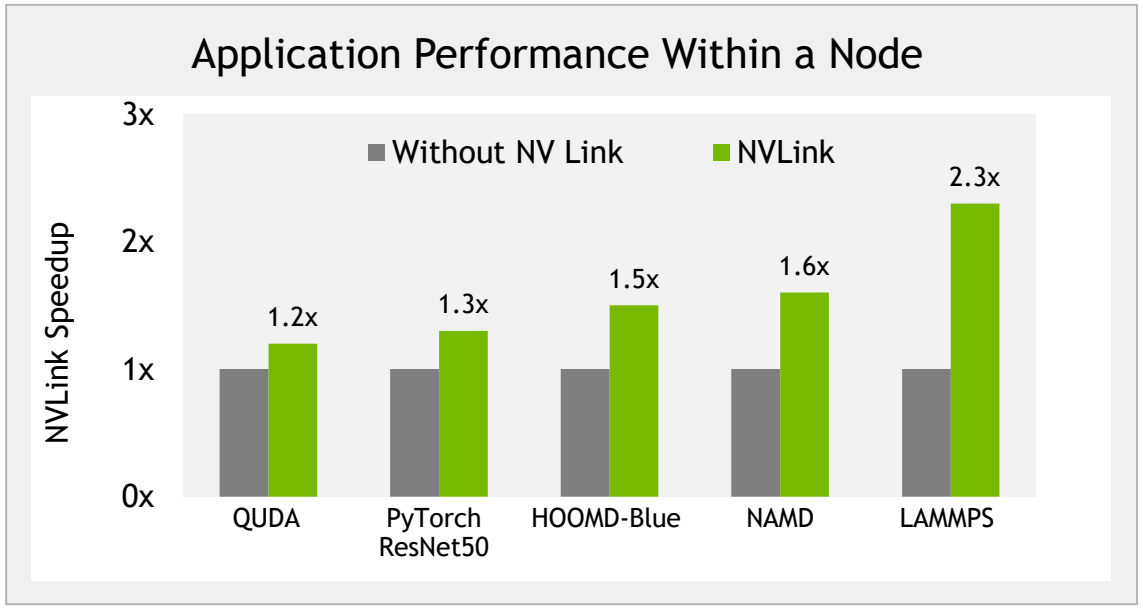




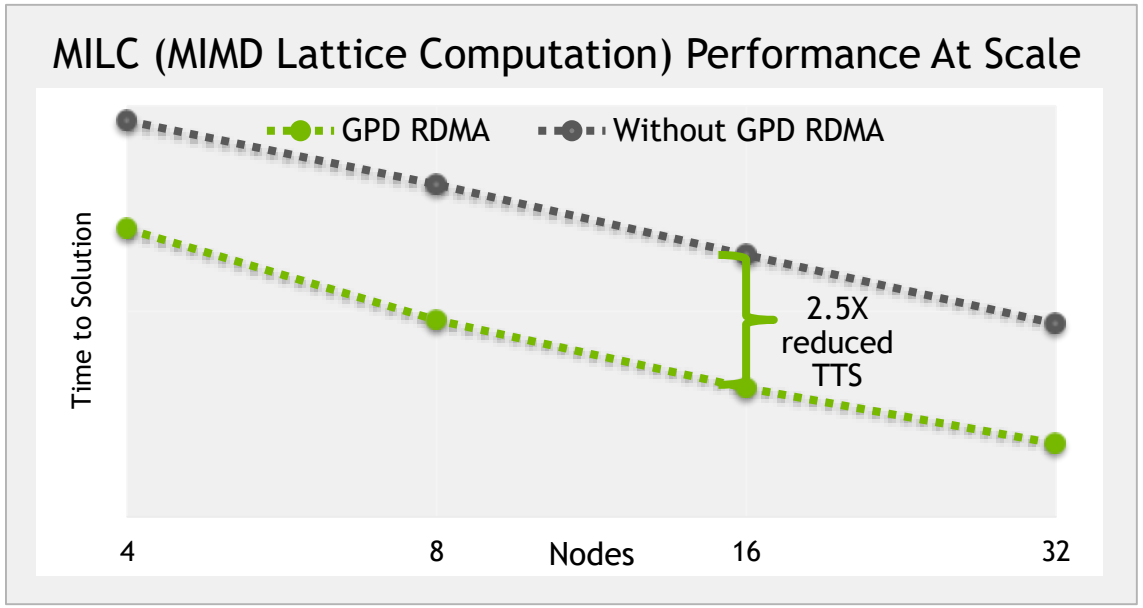
* Borrowed from Mellanox site

SHORTEN TIME TO INSIGHT

Up to 2.5X faster with NVLink and GPUDirect RDMA



NVLink 8xV100



GPUDirect RDMA 8xV100 nodes

NVLink: GPU Servers: Dual Xeon Gold 6140@2.30GHz or E5-2698 v4@3.6GHz for PyTorch with V100 PCIe vs V100 NVLink

GPUDirect RDMA: HW: DGX-1 with 8xV100 SXM2, SW: MILC (APEX large benchmark). This is measured on DGX-1 systems

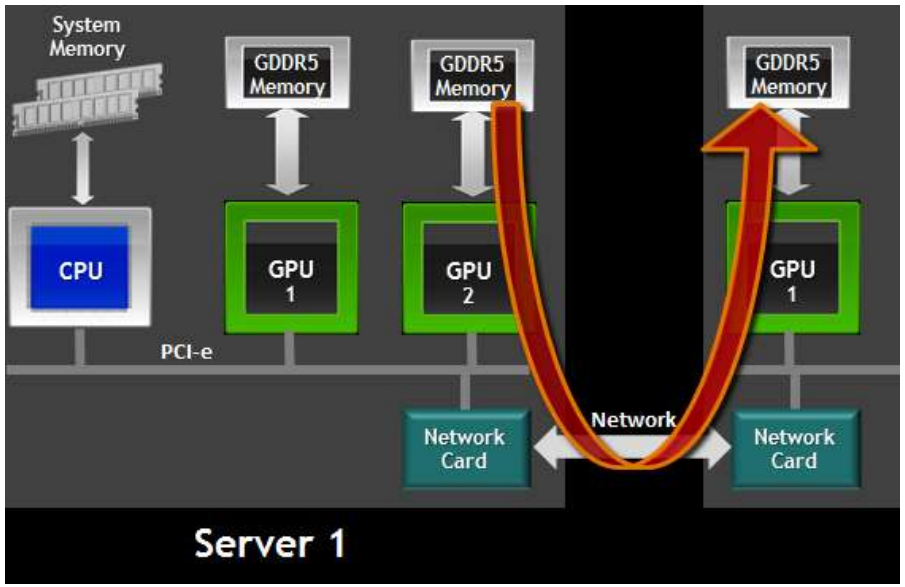
UCX: UNIFIED COMMUNICATION (X) FRAMEWORK

Popular unified communication library used for MPI/PGAS implementations such as OpenMPI, MPICH, OSHMEM, etc

Exposes API for:

- Client-server based connection establishment
- Point-to-point, RMA, atomics capabilities
- Tag matching
- Callbacks on communication events
- Blocking/Polling progress
- Cuda-Aware Point-to-point communication

C library!



CONTAINERIZE

CHALLENGE: SIMPLIFYING WORKFLOWS

WHY CONTAINERS

Simplifies Deployments

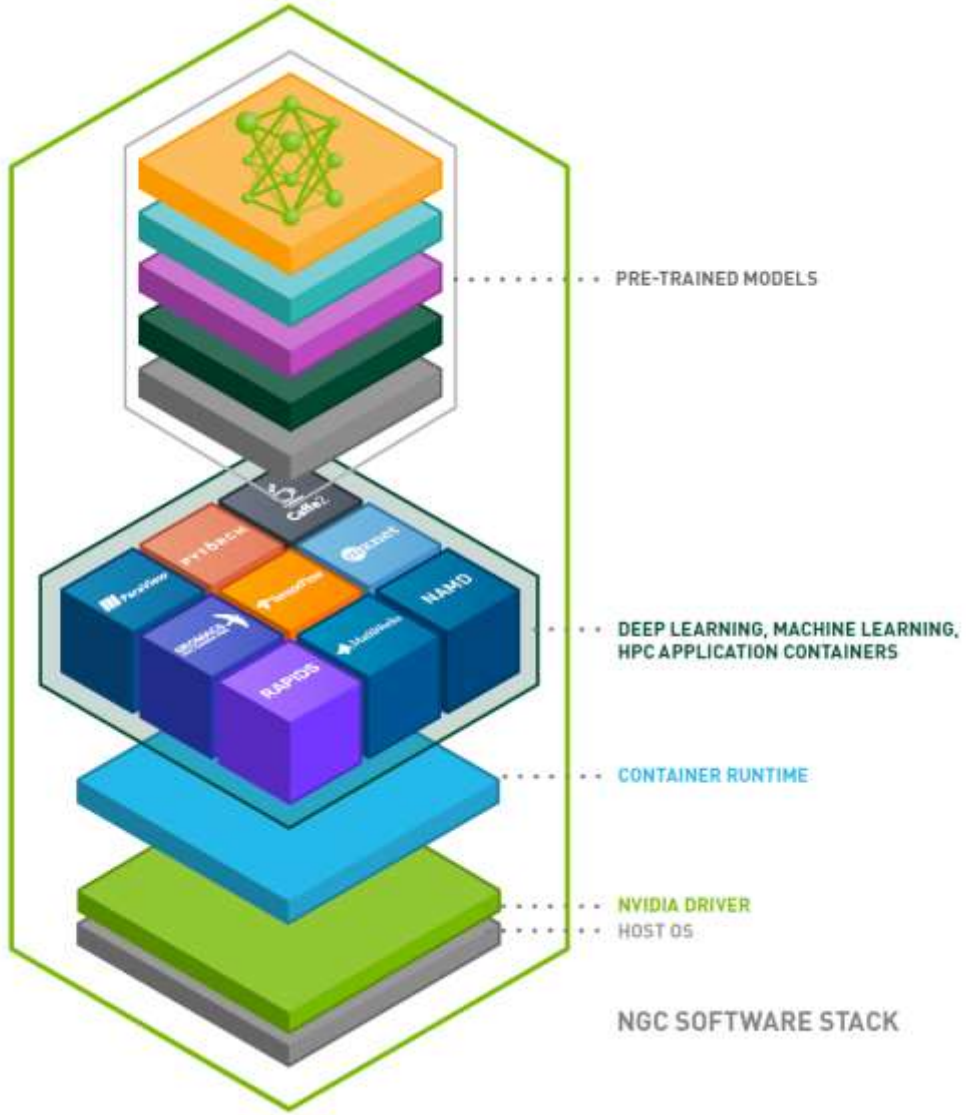
- Eliminates complex, time-consuming builds and installs

Get started in minutes

- Simply Pull & Run the app

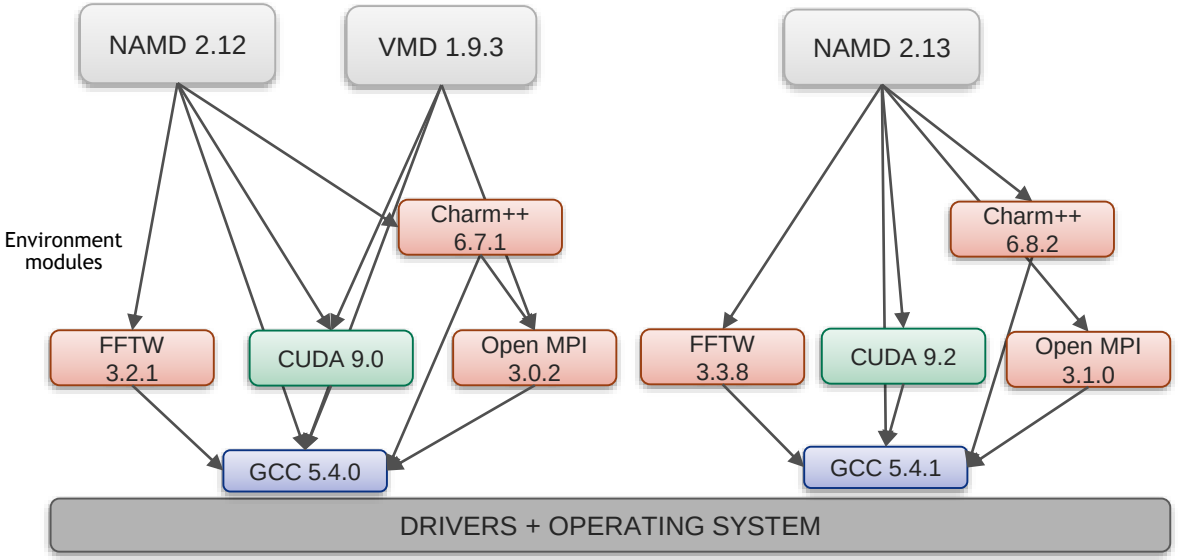
Portable

- Deploy across various environments, from test to production with minimal changes



BARE METAL: COMPLEX, HIGH MAINTENANCE, UNPRODUCTIVE

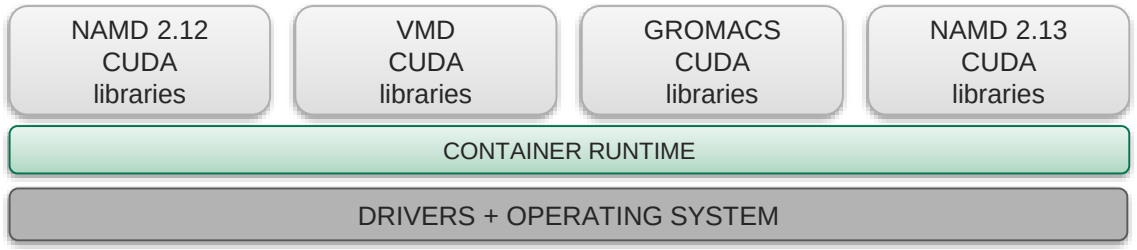
HPC NEEDS SIMPLIFED APPLICATION DEPLOYMENTS



SHARED CLUSTER

- Installations may take days
  - Build complicated env module networks
  - Install issues difficult to diagnose
- Maintain 100s of env modules in perpetuity
- Library hoarding necessary – Lots of interdependencies
- Upgrades might break existing apps – not proactively updated

CONTAINERS SIMPLIFY APPLICATION DEPLOYMENTS

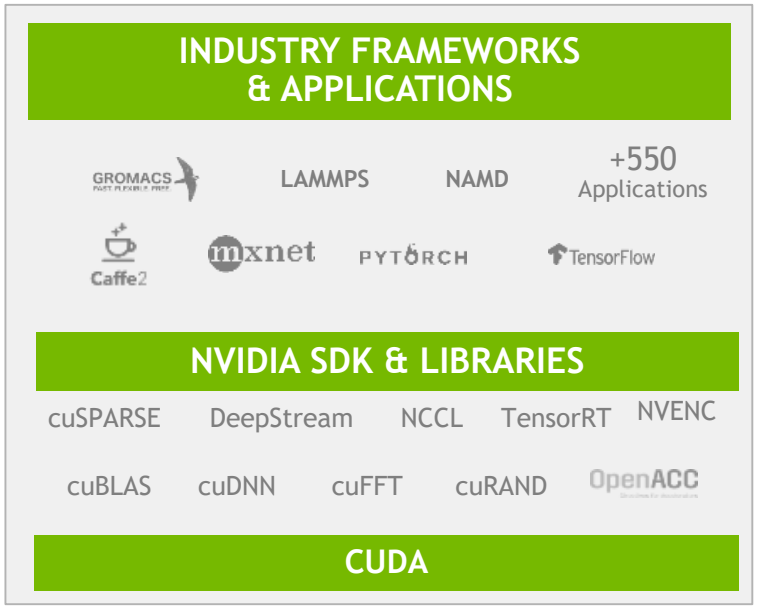


SHARED CLUSTER

- Environment modules simplified/eliminated
- Performance equivalent to bare metal
- Deploy applications in minutes
- Higher productivity for sys admins & users
- Portable on various systems
- Reproducible results

COMPLEX STACKS JUST WORK

Fully optimized



Developer support for Every Application

Works with ISV and ecosystem partners to optimize the stack over time

NVIDIA GPU Cloud

Innovation for every industry

Say goodbye to DIY

Stay up to date

Test and Benchmark Top Applications

Dedicated infrastructure (Prometheus)

BARE METAL V. CONTAINER

USER WORKFLOW

BARE METAL

1. Reserve nodes & schedule a job

```
#!/bin/bash
#PBS -A <project id>
#PBS -l walltime=1:00:00,nodes=1500
```

2. User loads environment modules

```
module swap PrgEnv-pgi PrgEnv-gnu
module load gcc/5.3.0
module load fftw/3.1.2
module load lammps/patch23Oct2017
```

Order of loading modules is critical and is error prone – a time/resource drain

3. Start LAMMPS and run the simulation from the `input` file

```
mpirun lammps -in input
```

4. Simulation output saved on the cluster file system

Assume LAMMPS installed on the cluster

CONTAINER

1. User pulls and saves the container on the cluster file system

```
singularity build lammps.simg nvcr.io/hpc/lammps:patch23Oct2017
```

2. Reserve nodes & schedule a job

```
#!/bin/bash
#PBS -A <project id>
#PBS -l walltime=1:00:00,nodes=1500
```

Eliminates the need to load env modules

3. Start LAMMPS and run the simulation from the `input` file

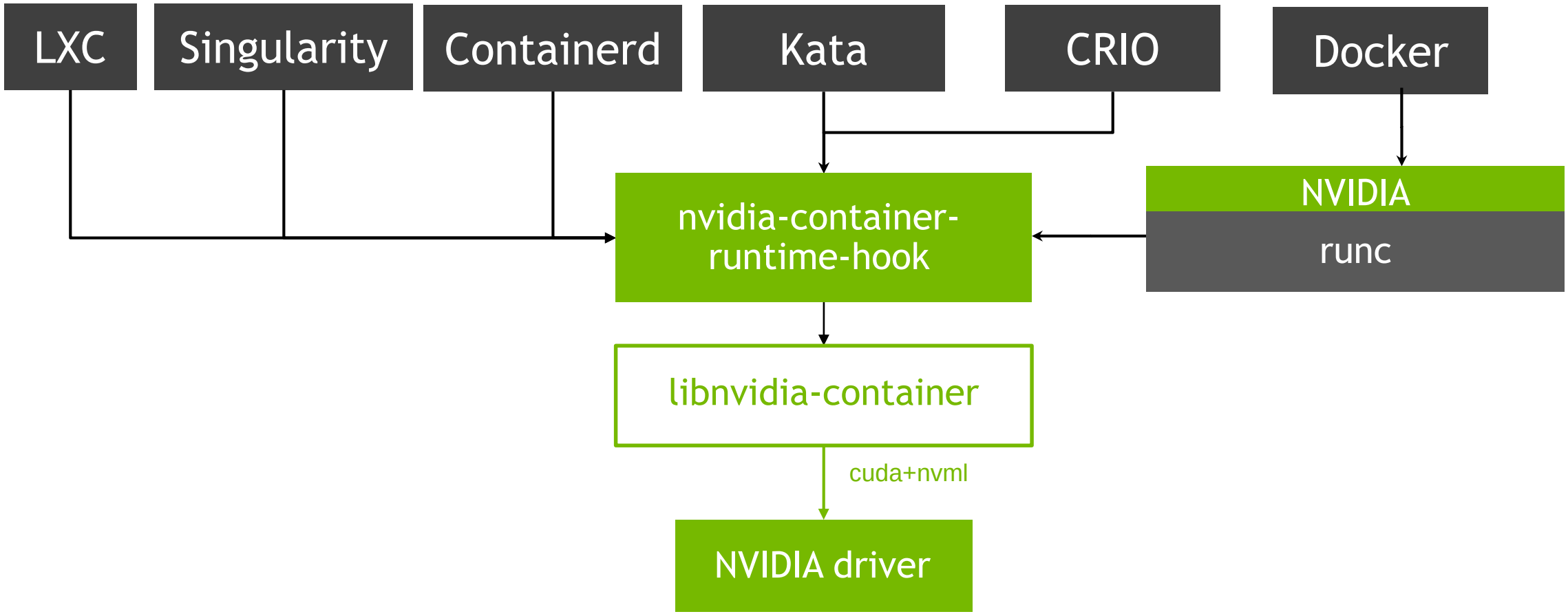
```
mpirun singularity exec --nv lammps.simg lammps -in input
```

4. Simulation output saved on the cluster file system

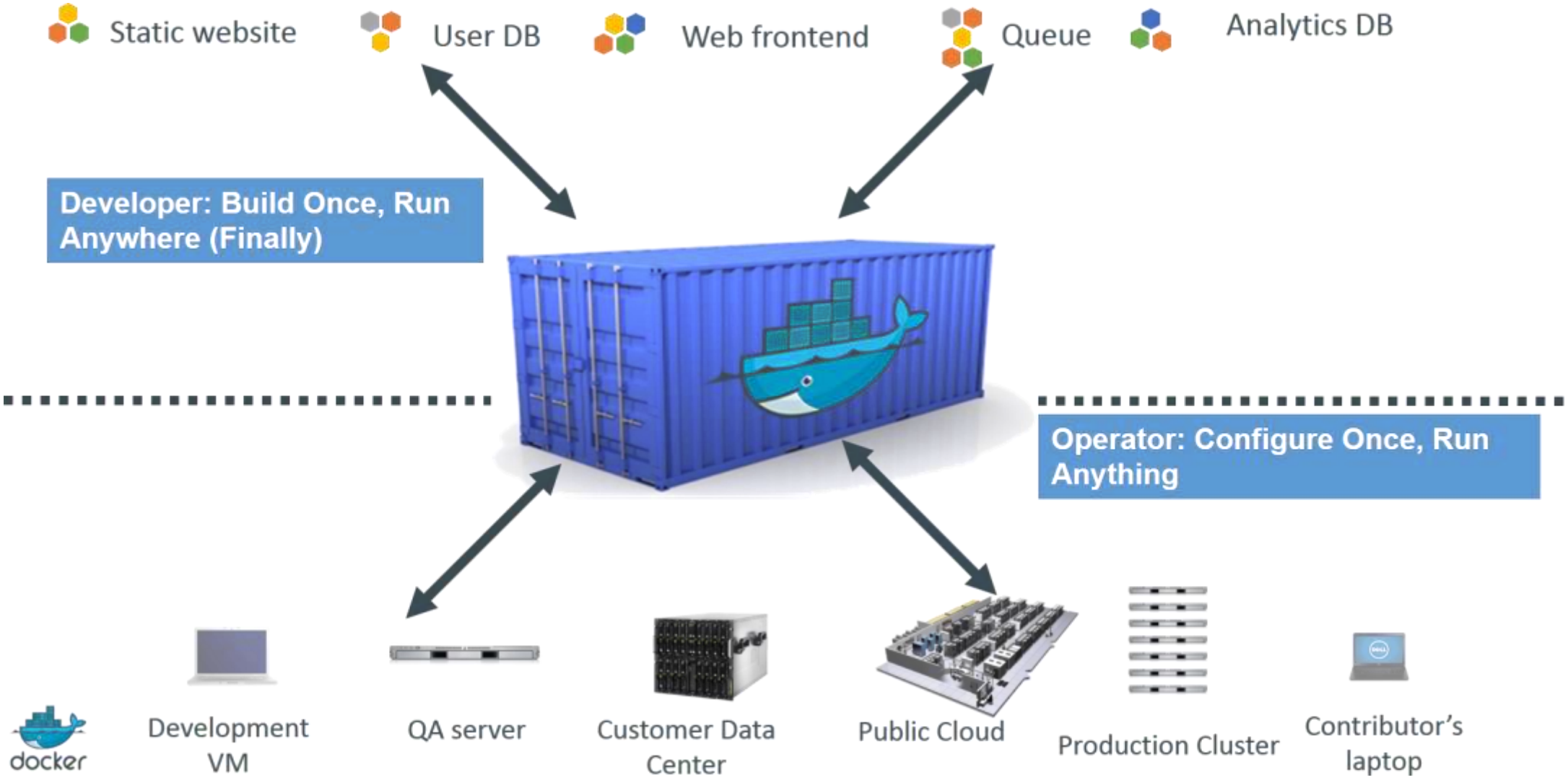
Assume Singularity runtime installed on the cluster. Each users pulls own LAMMPS container

NVIDIA CONTAINER TECHNOLOGY

Libnvidia-container: Usage across multiple runtimes. Docker, Singularity, CRI-O, etc.



DOCKER'S (CONTAINERIZATION) PITCH (2013)



SINGULARITY BENEFITS

Containers run with user privileges and inherit user system config

Out of the box advantages compared to Docker

- Any user can run containers without special privileges (root)
- Integrate seamlessly into existing infrastructure (resource managers com
- Users created and provided containers (less administrative oversight)
- Single image file contains everything necessary (registry not necessary)

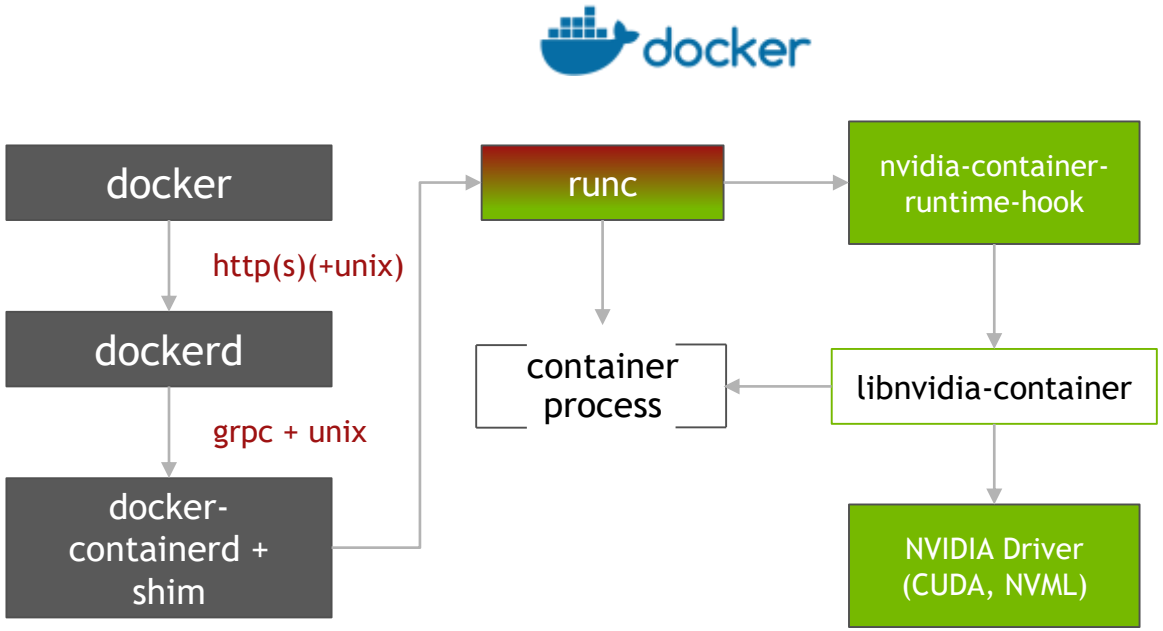
Disadvantages are mainly around its support of ***microservices*** oriented software architecture and orchestration. Setting up daemon processes with customized network and endpoints via singularity is not as mature compared to docker. Further references:

- services - https://www.sylabs.io/guides/3.0/user-guide/running_services.html
- networking - https://www.sylabs.io/guides/3.0/user-guide/networking.html

NVIDIA CONTAINER RUNTIME

Integration into the Docker stack

- Run GPU containers using “--runtime” option in the Docker CLI
  - Registration of custom runtime with Docker daemon
- Platform support
  - Pre-built packages for different OS distributions (Amazon, CentOS, Debian, Ubuntu)
  - Updated with Docker releases (most recent 18.09.3)

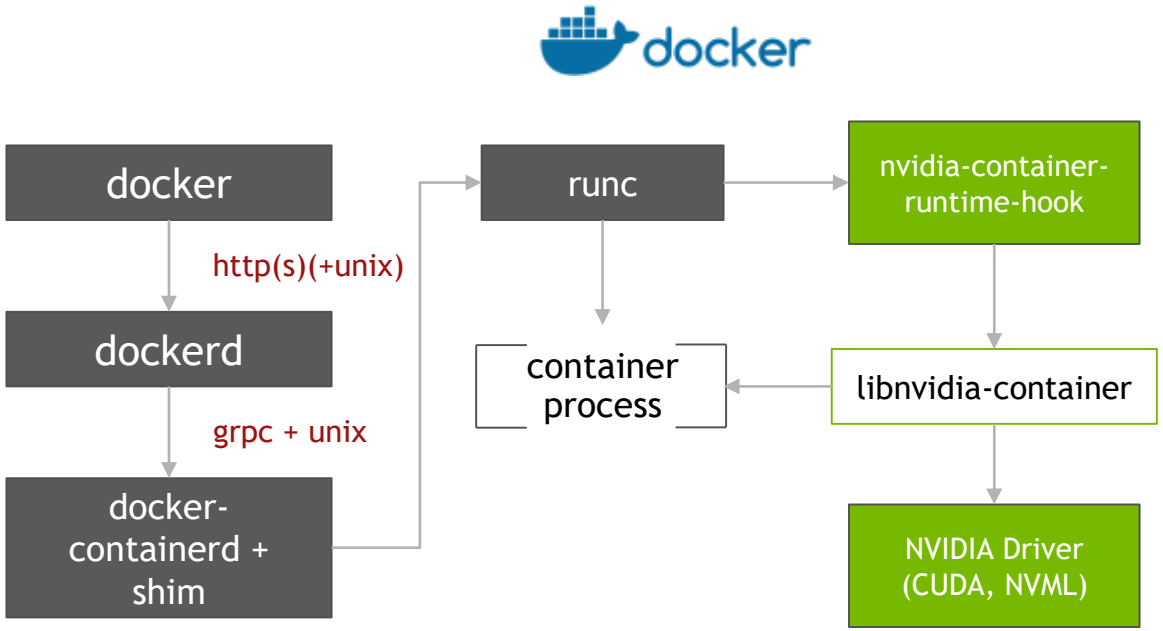



```
$ docker run -it --runtime=nvidia nvidia/cuda
```

NVIDIA CONTAINER RUNTIME

Upcoming native Docker integration

- Native GPU integration into upcoming Docker 19.03 CE
- Run GPU containers using “--gpu” option in the Docker CLI
- Simplified installation to enable GPU support
  - Fewer NVIDIA packages to install
  - No custom runtimes or registration required
- Merged PR: https://github.com/moby/moby/pull/38828

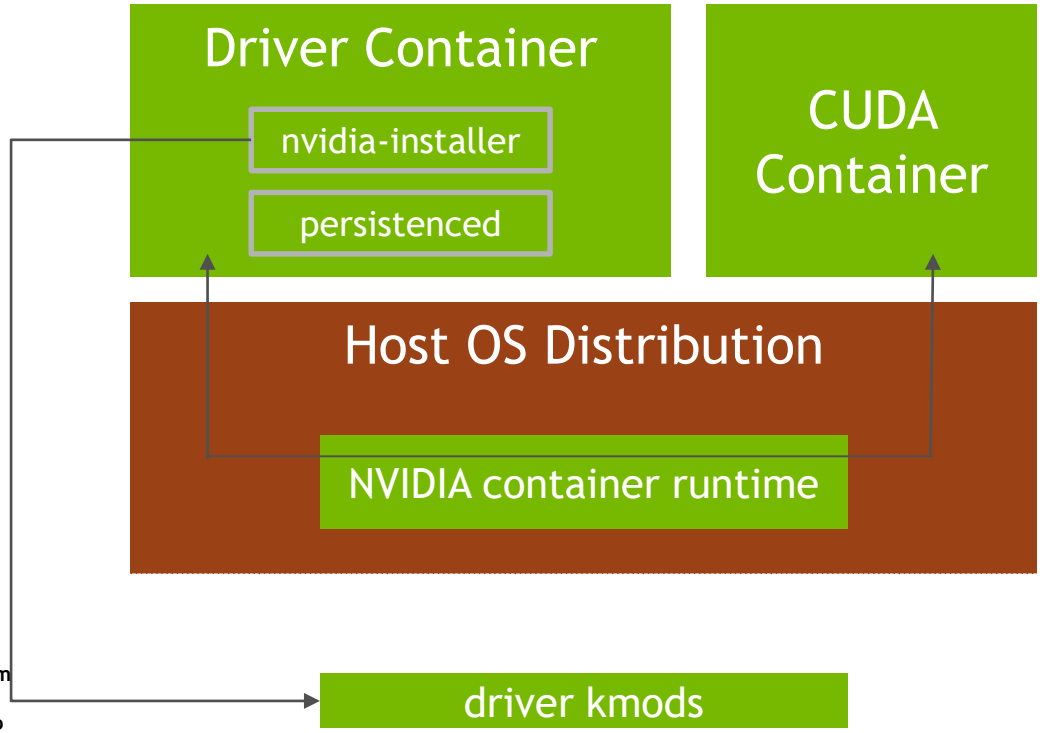



```
$ docker run -it -gpus all nvidia/cuda
```

IN DEVELOPMENT: CONTAINERIZED DRIVERS

Simplifying deployment of NVIDIA software

- Goal is to simplify provisioning NVIDIA drivers (easy as start/stop container)
- Other benefits
  - Speed
  - Use with container operating systems in the cloud (e.g. Container Linux)
  - Portable
- Beta available now on GitHub; productization in 2H 2019
- Demo: https://asciinema.org/a/TXHMQ5och6IjReiHaSjQLpQ0M



LATEST NVIDIA CONTAINER FEATURES

Container Runtime Ecosystem

- CUDA 10 Compatibility
- Container Runtime Ecosystem
  - Docker
  - CRI-O
  - RHEL Docker
  - LXC (since 3.0.0)
  - Singularity (--nv option)

Platform Support

- Driver Containers
- CoreOS
- Volta Multi-Process Service
- Docker GPU support (--gpus CLI option)
- GPUs in Kata Containers

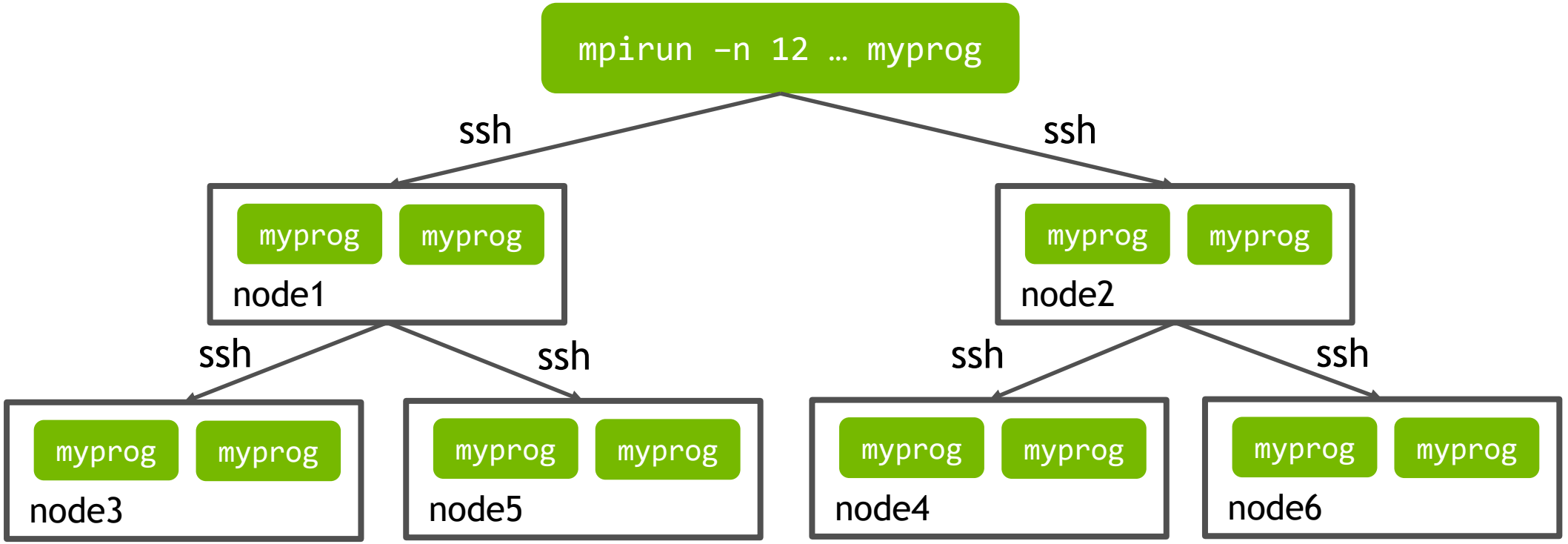
2018 2019

SCALING GPU APPLICATIONS WITH CONTAINERS AND MPI

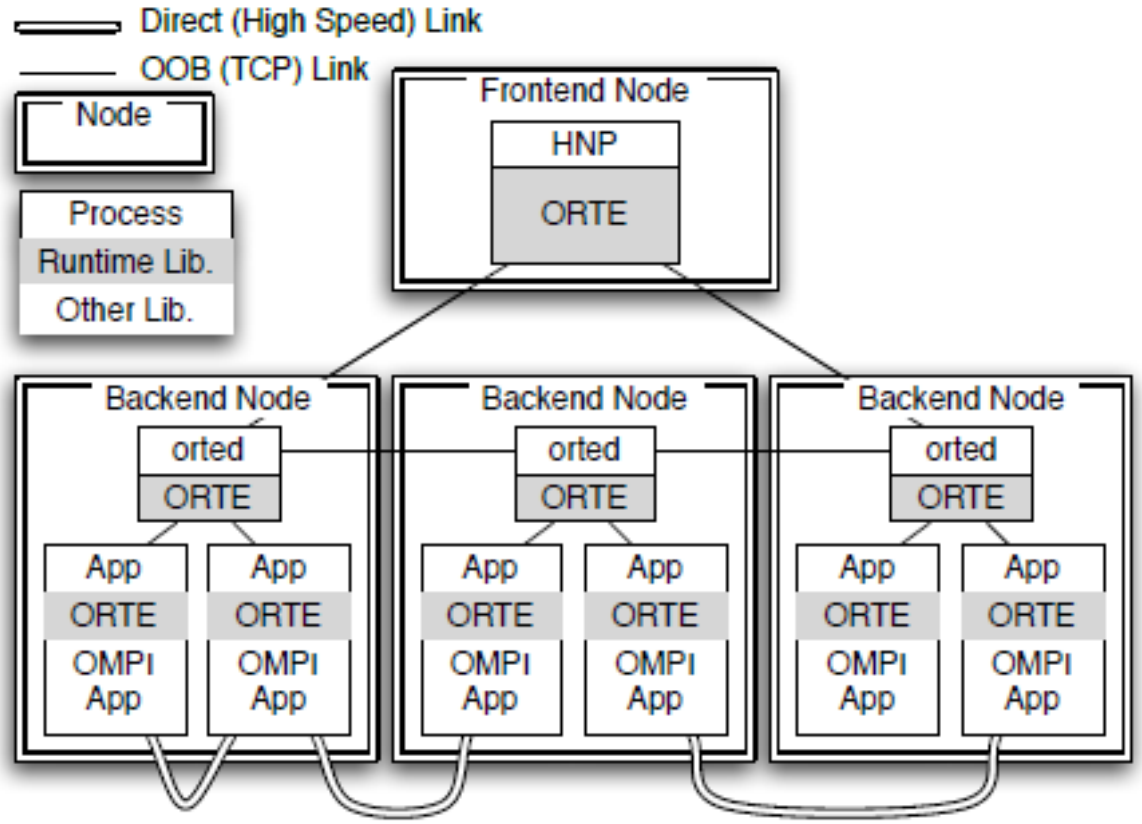
MPI PROCESS SPAWN

General idea of how MPI launcher spawns processes

- MPI implementations provide a job launcher, mpirun or mpiexec, that initializes and wires up distributed MPI ranks (i.e., processes) on a multi-node cluster



OPENMPI ORTE DIAGRAM



* George Bosilca, Thomas H´erault, Ala Rezmerita, and Jack Dongarra. On Scalability for MPI Runtime Systems. In CLUSTER, pages 187–195, 2011.

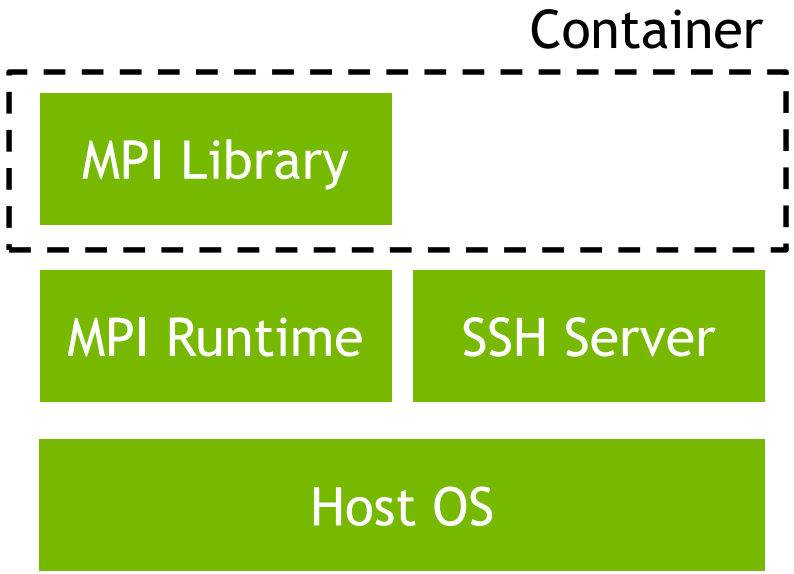
ORTE – Open Run-Time Environment

MPIRUN + CONTAINERS

- “Outside-in”
  - Fits in more “naturally” into the traditional HPC workflow (SSH keys, etc.)
  - mpirun -hostfile hostfile –n 64 app
    becomes
    mpirun –hostfile hostfile –n 64 singularity run app.simg app
  - Requires a compatible MPI runtime on the host
- “Inside-out”
  - Must insert SSH keys into the container image by some other mechanism
  - Must orchestrate the launch of containers on other hosts
  - Completely self-contained, no host MPI dependencies

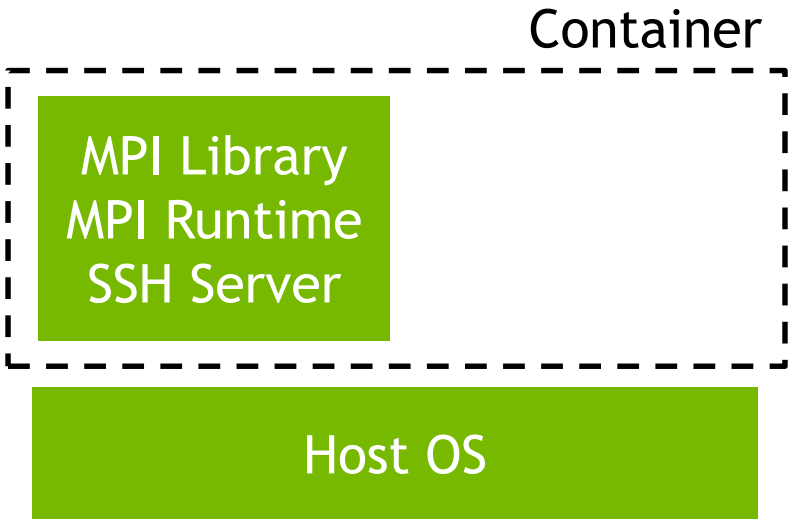
MPIRUN + CONTAINERS

“Outside-in”



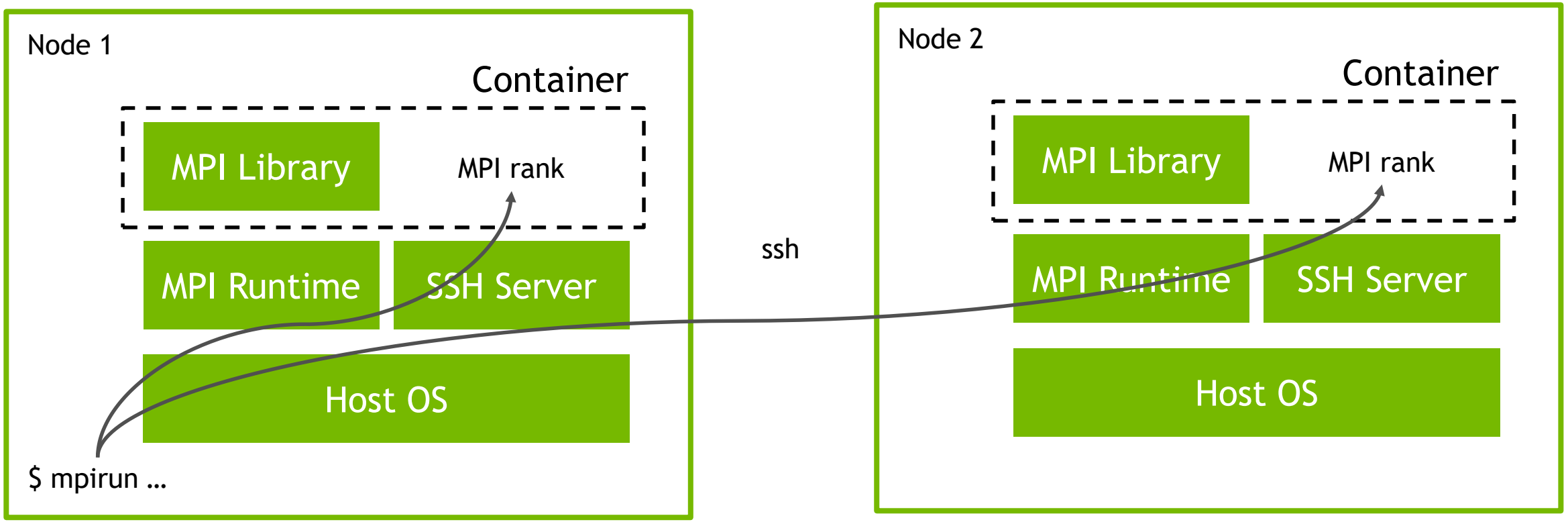
mpirun is invoked outside the container

“Inside-out”

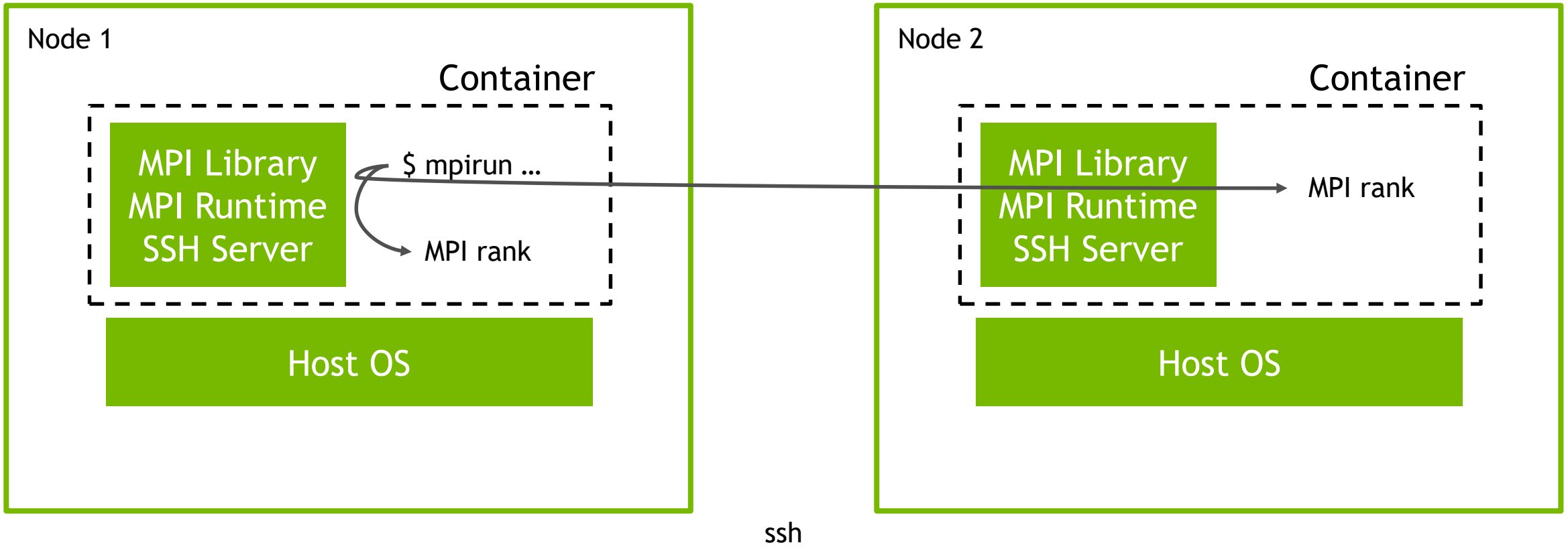


mpirun is invoked inside the container

“OUTSIDE-IN” SCHEMATIC

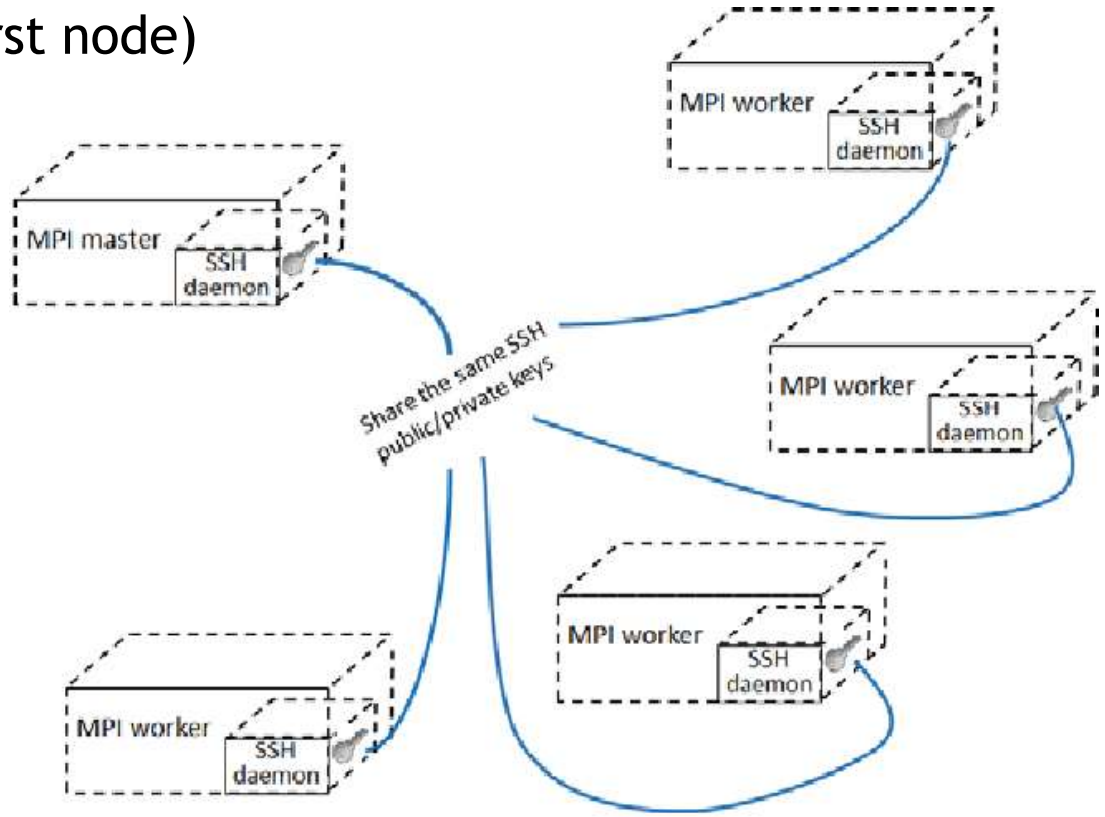


“INSIDE-OUT” SCHEMATIC



INSIDE-OUT: MPI AND CONTAINERS

- Allocate nodes
  - One node acts as master (typically the first node)
  - N-1 worker nodes
- Worker nodes start the container
  - Start sshd
  - Sleep infinity
- Master node starts the container
  - Waits for all workers to be ready
  - mpirun's the job
- Cleanup
  - Kill all the worker nodes and exit



DEMOS: MULTINODE MPI CONTAINERS

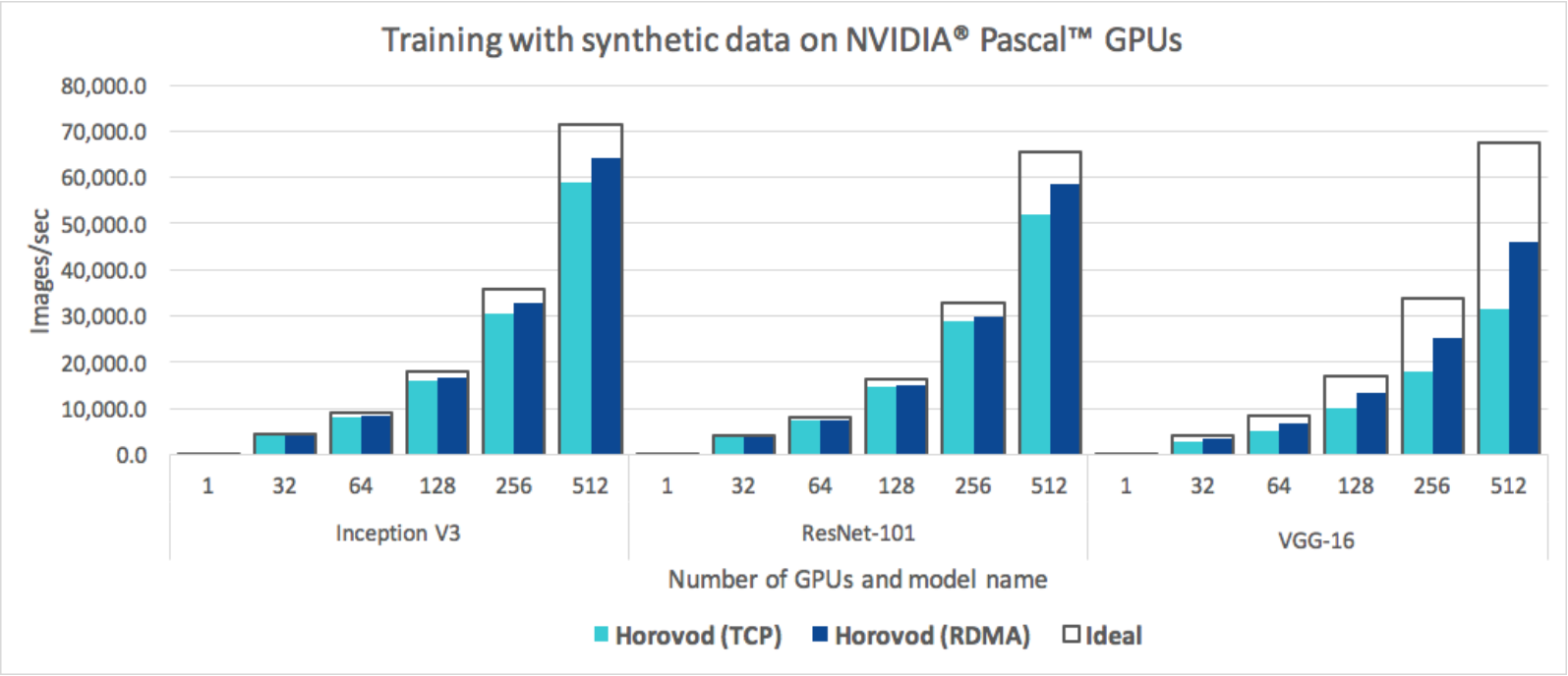
HOROVOD: DISTRIBUTED TRAINING FRAMEWORK

Leverage MPI for distributed training with Deep Learning Frameworks

https://github.com/uber/horovod, https://eng.uber.com/horovod/

Leverage Tensorflow/PyTorch/MxNet + MPI + NCCL2 for a simplified and performant API to enable synchronous multinode + multigpu training.

Support features such as RDMA, GPUDirect RDMA (GDR), via leveraging MPI and NCCL2.



“OUTSIDE-IN”: SINGULARITY

Tensorflow MNIST example: https://github.com/avolkov1/multinode_containers

Example on a SLURM cluster (same idea for PBS/Torque/MOAB/UGE):

```
salloc -N 2 -p some-partition # interactive shell with requested resources
module load PrgEnv/GCC+OpenMPI/2018-05-24
module load openmpi/3.1.0 # load MPI (e.g. openmpi) that is integrated with scheduler
module load singularity/3.1.0
srun --ntasks-per-node=8 --pty bash # interactive compute node 8 tasks per node
# -np 8 for one DGX-1. Use 16 for two DGX-1s. 8 GPUs per DGX-1
# Command to run mnist code “tensorflow_mnist.py”. Exclude non-routed interfaces.
NCCL_SOCKET_IFNAME=^docker0,lo,virbr0 \
  mpirun -mca btl_tcp_if_exclude docker0,lo,virbr0 -x NCCL_SOCKET_IFNAME \
    -np 16 singularity exec --nv \
    /cm/shared/singularity/tf1.8.0py3.simg bash -c '
python ./tensorflow_mnode/tensorflow_mnist.py'
```

“INSIDE-OUT”: SINGULARITY

Tensorflow MNIST example: https://github.com/avolkov1/multinode_containers

Example on a SLURM cluster (same idea for PBS/Torque/MOAB/UGE):

```
salloc -N 2 -p some-partition # interactive shell with requested resources
module load singularity/3.1.0
# using helper/wrapper “srun_singularity.sh” to spawn sshd “inside-out” sessions
srun srun_singularity.sh \
    --container=/cm/shared/singularity/tf1.8.0py3.simg \
    --script=./tensorflow_mnode/hvd_mnist_example.sh
```

Within “hvd_mnist_example.sh” script invoke mpirun/mpiexec command:

```
mpirun -x LD_LIBRARY_PATH -x SHELL ${evars} $hostlistopts -np $np \
    -mca btl_tcp_if_exclude docker0,lo,virbr0 \
    -x NCCL_SOCKET_IFNAME \
    -x NCCL_IB_DISABLE \
    --report-bindings --bind-to none --map-by slot \
    python ./tensorflow_mnode/tensorflow_mnist.py "$@"
```

“OUTSIDE-IN”: DOCKER MULTI-NODE RUN

Outside in approach with docker is more complicated

```
$ module load openmpi
$ mpirun -n 2 -npernode 1 nvidia-docker run --rm --cap-add=IPC_LOCK --device=/dev/infiniband/uverbs0
--env-file $HOME/ompi.env --network=host --pid=host --ulimit memlock=268435456 --user $(id -u):$(id
-g) --volume /tmp:/tmp mpi-bandwidth mpi_bandwidth
******************** MPI Bandwidth Test ********************
Message start size= 100000 bytes
Message finish size= 1000000 bytes
Incremented by 100000 bytes per iteration
Roundtrips per iteration= 100
MPI_Wtick resolution = 1.000000e-09
************************************************************
task    0 is on ivb125.internalnet partner=   1
task    1 is on ivb126.internalnet partner=   0
************************************************************
***Message size:   100000 *** best  /  avg  / worst (MB/sec)
   task pair:    0 -    1:    3732.46 / 3401.84 / 67.57
   OVERALL AVERAGES:          3732.46 / 3401.84 / 67.57
…
***Message size:  1000000 *** best  /  avg  / worst (MB/sec)
   task pair:    0 -    1:    6087.35 / 6044.27 / 3086.63
   OVERALL AVERAGES:          6087.35 / 6044.27 / 3086.63
```

“INSIDE-OUT”: DOCKER

Tensorflow MNIST example: https://github.com/avolkov1/multinode_containers

Example on a SLURM cluster (same idea for PBS/Torque/MOAB/UGE):

```
salloc -N 2 -p some-partition # interactive shell with requested resources
# using helper/wrapper “srun_docker.sh” to spawn sshd “inside-out” sessions
srun srun_docker.sh \
    --container=nvcr.io/nvidian/sae/avolkov:tf1.12.0py3_cuda10.0_nccl2.3.7_ompi3_ibverbs \
    --privileged \
    --script=./tensorflow_mnode/hvd_mnist_example.sh
```

Within “hvd_mnist_example.sh” script invoke mpirun/mpiexec command:

```
mpirun -x LD_LIBRARY_PATH -x SHELL ${evars} $hostlistopts -np $np \
    -mca btl_tcp_if_exclude docker0,lo,virbr0 \
    -x NCCL_SOCKET_IFNAME \
    -x NCCL_IB_DISABLE \
    --report-bindings --bind-to none --map-by slot \
    python ./tensorflow_mnode/tensorflow_mnist.py "$@"
```

RESOURCES AND HELPER UTILITIES

MULTINODE CONTAINERS HELPER SCRIPTS

https://github.com/avolkov1/multinode_containers

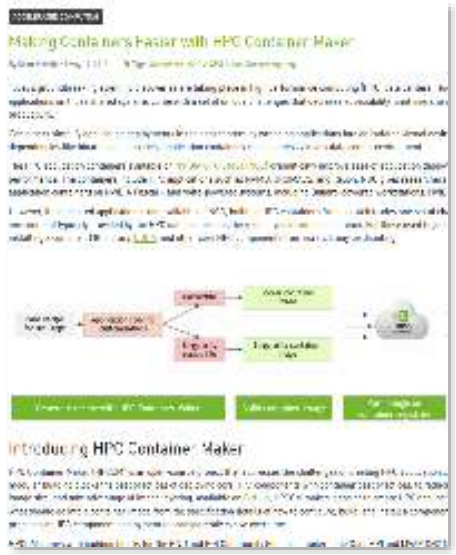
Example scripts to launch on slurm and via pdsh (pdsh when no resource manager)

```
salloc -N 2 -p <some_partition> # using two nodes

# docker where --privileged option is for RDMA support
srun srun_docker.sh --privileged \
    --container=<your_container> \
    --script=./<your_job_script>.sh

PDSH_RCMD_TYPE=ssh PDSH_SSH_ARGS_APPEND="-p 22" pdsh -w $NODES NODES=$NODES \
        pdsh_docker.sh --noderank=%n --privileged \
            --container =<your_container> \
            --script =./<your_job_script>.sh \
            --workingdir=${PWD}
```

HPC CONTAINER MAKER

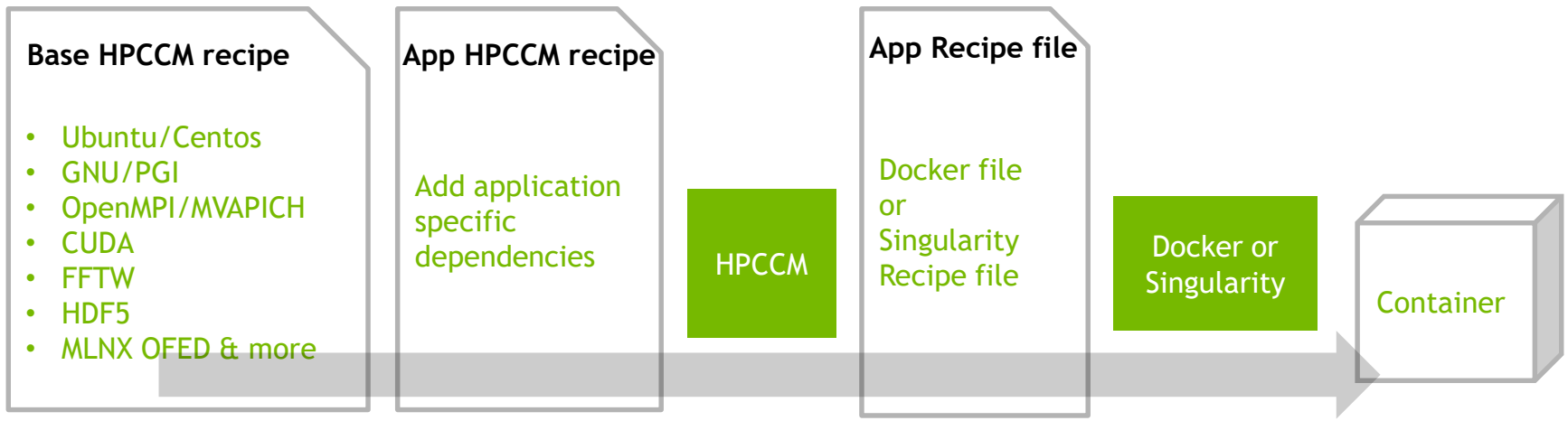
SIMPLEST WAY TO BUILD CONTAINERS

- Tool for creating HPC application Dockerfiles and Singularity recipe files
  - You will still need to build the container images
- Makes it easier to create HPC application containers by encapsulating best practices into building blocks
- Open source (Apache 2.0)
  https://github.com/NVIDIA/hpc-container-maker
- `pip install hpccm`



HPCCM Blog

HPCCM BUILD FLOW



1. Follows best practices
   - Layered builds
   - Optimized container size
   - Configuration, deployment

2. Ready-to-use building blocks
   - Simplify efforts
   - Faster builds
   - Higher reliability
   - Optimized performance

https://github.com/NVIDIA/hpc-container-maker
https://devblogs.nvidia.com/making-containers-easier-with-hpc-container-maker/

BASIC HPCC SCRIPT/RECIPE

- MPI Bandwidth is a microbenchmark from Lawrence Livermore National Lab
  - https://computing.llnl.gov/tutorials/mpi/samples/C/mpi_bandwidth.c
- Below is a HPC Container Maker recipe for MPI Bandwidth

```
# MPI Bandwidth Container
Stage0 += baseimage(image='nvidia/cuda:9.0-devel-centos7')
Stage0 += mlnx_ofed()
Stage0 += gnu()
Stage0 += openmpi()
Stage0 += shell(commands=[
    'wget -q -nc --no-check-certificate -P /var/tmp https://computing.llnl.gov/tutorials/mpi/samples/C/mpi_bandwidth.c',
    'mpicc -o /usr/local/bin/mpi_bandwidth /var/tmp/mpi_bandwidth.c'])
```

HCCM GENERATE DOCKERFILE

Generate the container image

```
$ hpccm --recipe mpi-bandwidth.py --format docker > Dockerfile
$ docker build –t mpi-bandwidth –f Dockerfile .
…
Successfully built 32995bb03fe2
Successfully tagged mpi-bandwidth:latest
```

If using an existing container image, you would skip this step.
Potential gotcha: the container image must exist on all nodes

HCCM GENERATE SINGULARITY DEF

Generate the container image

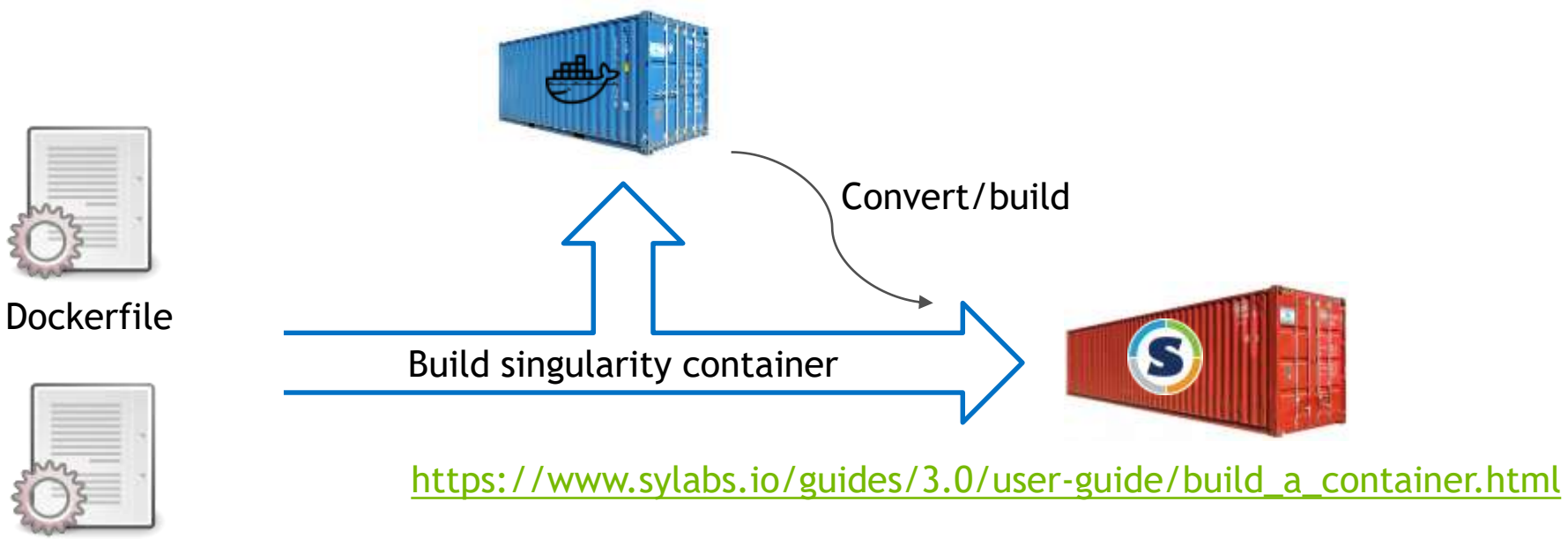
```
$ hpccm --recipe mpi-bandwidth.py --format singularity >
Singularity.def
$ sudo singularity build mpi-bandwidth.simg Singularity.def
…
Building Singularity image...
Singularity container built: mpi-bandwidth.simg
Cleaning up...
```

If using an existing container image, you would skip this step.

BUILDING SINGULARITY CONTAINERS

Typically easier to maintain docker containers and convert

Private docker registry and NGC Support: https://docs.nvidia.com/ngc/ngc-user-guide/singularity.html#singularity



https://www.sylabs.io/guides/3.0/user-guide/build_a_container.html

Similar to Dockerfile one can use singularity definition files:
https://www.sylabs.io/guides/3.0/user-guide/definition_files.html


THE END
NVIDIA