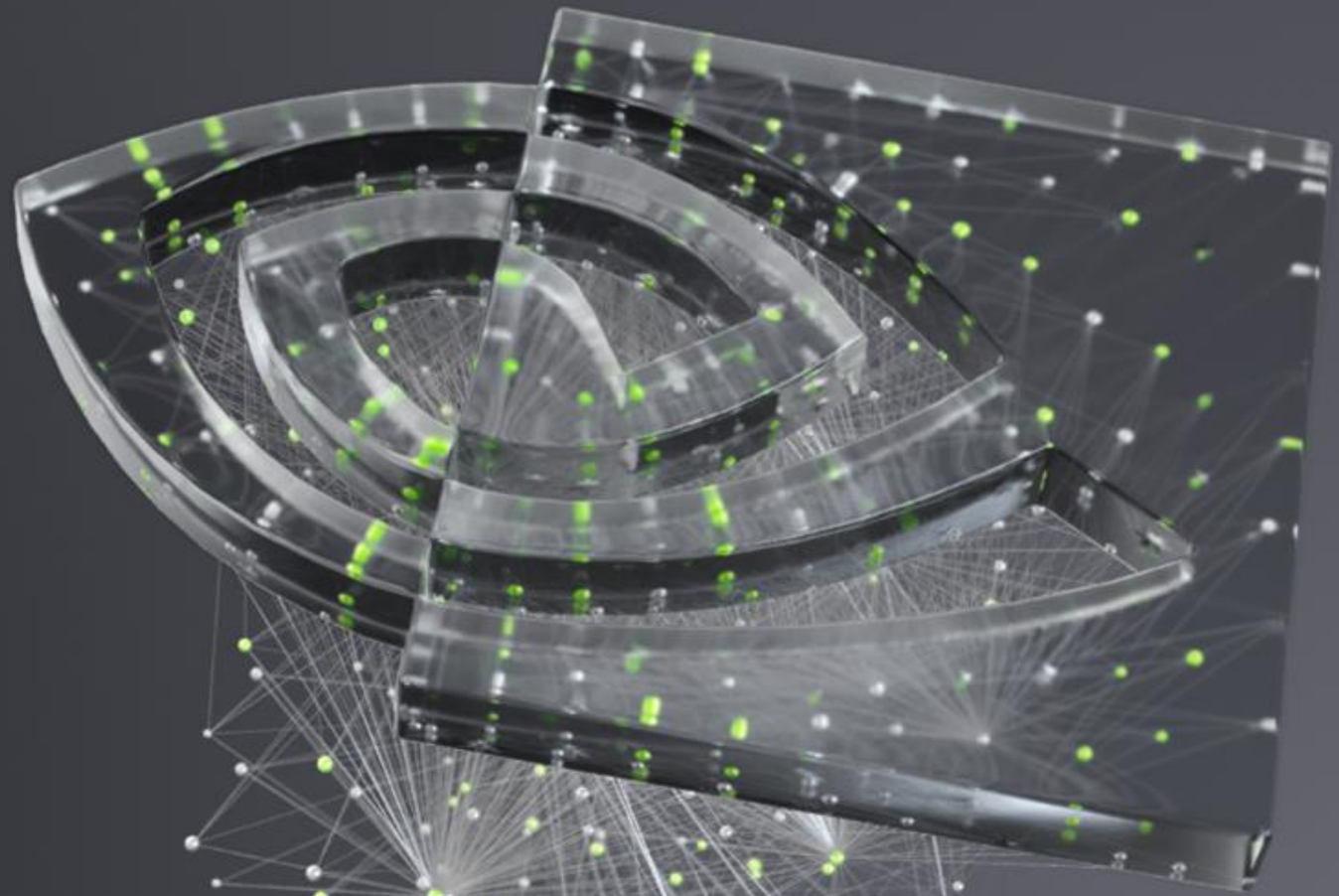
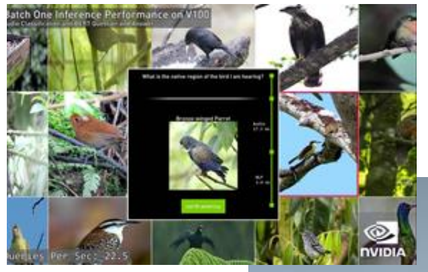


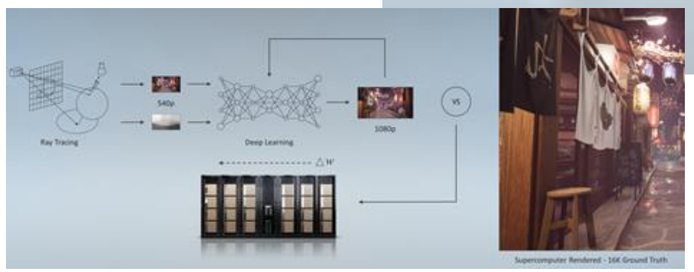
HIGH PERFORMANCE NEXT GENERATION DEEP LEARNING CLUSTERS

Julie Bernauer, 2020/05/21

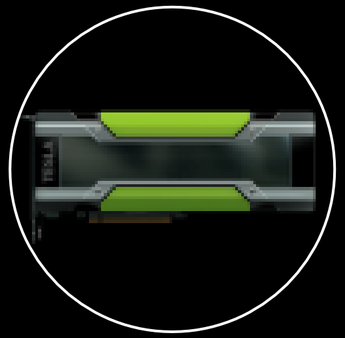
AI EVERYWHERE

NVIDIA





DL TRAINING: FROM SINGLE GPU TO MULTI-NODE



36000 Mins (25 Days)
1xK80 | 2015
CUDA

2015

1200 Mins (20 Hours)
DGX-1P | 2016
NVLink

2016

480 Mins (8 Hours)
DGX-1V | 2017
Tensor Core

2017

70 Minutes on MLPerf
DGX-2H | 2018
NVSwitch

6.3 Minutes on MLPerf
At Scale | 2018
DGX Cluster

2018

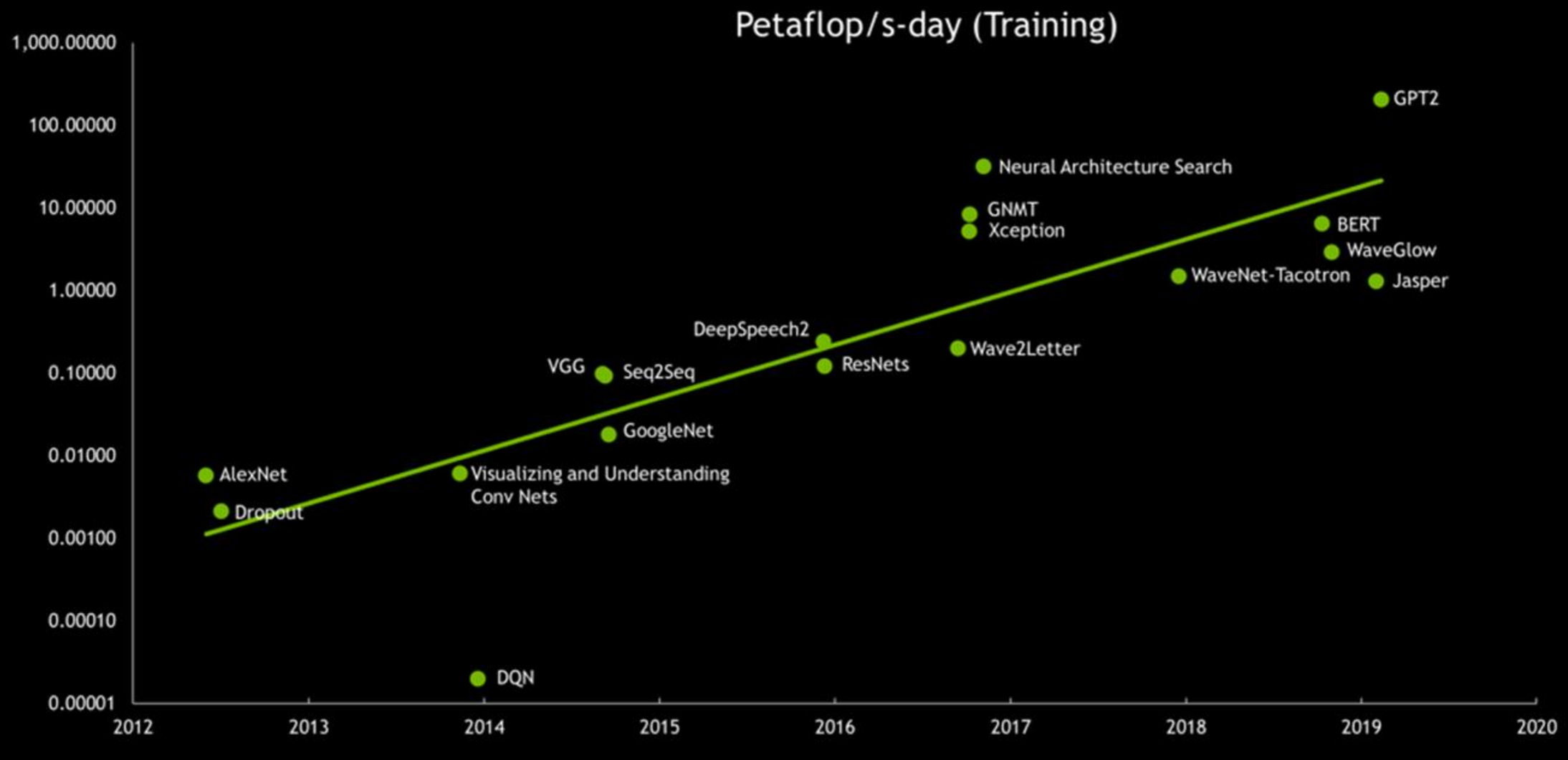
52.7 Minutes on MLPerf
DGX-2H | 2019
NVSwitch

1.33 Minutes on MLPerf
At Scale | 2019
DGX SuperPOD

2019

ResNet50 v1.5 training

MODELS GETTING MORE COMPLEX



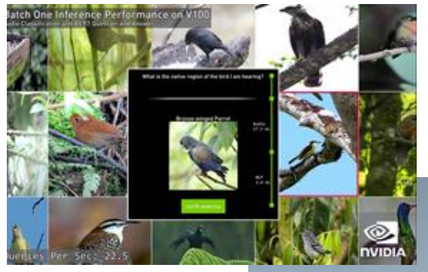
Source: OpenAI and NVIDIA

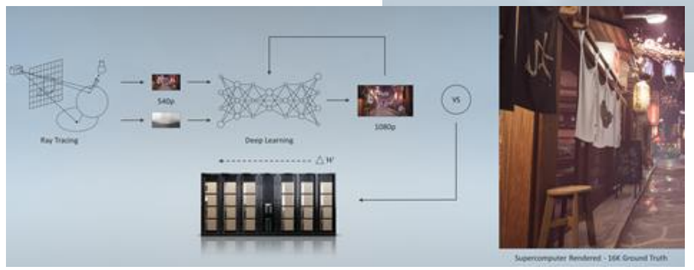
DATASETS GETTING LARGER

- **Unlabeled data:**
  - Language model: BooksCorpus (800M words), English Wikipedia (2.5B words), WebText (8M documents, 40 GB), C4 (Common Crawl, 745 GB)
  - GAN: unlabeled images and videos
  - Reinforcement learning: unsupervised self-play generates unlimited data
- **Labeled data:**
  - ImageNet (2012) - 1.3M images, 1000 categories → Open Images (2019) - 9M images, 6000 categories
  - Semi-autonomous vehicles: 0.5-1.1TB of data for every 8h driving

AI EVERYWHERE

NVIDIA





NVIDIA DGX-2H SUPERPOD

An AI supercomputer

Highest-Performance AI Research Supercomputer

- #20 on Top500 list | Top AI Performance Records
- 96 DGX-2H nodes

Fast multi-node interconnect

- Mellanox EDR 100G InfiniBand Network

AI Infrastructure

- Modular and scalable architecture
- Integrated and optimized compute, networking, storage, and software

NVIDIA-Optimized Software Stacks

- Containers Freely available on NGC

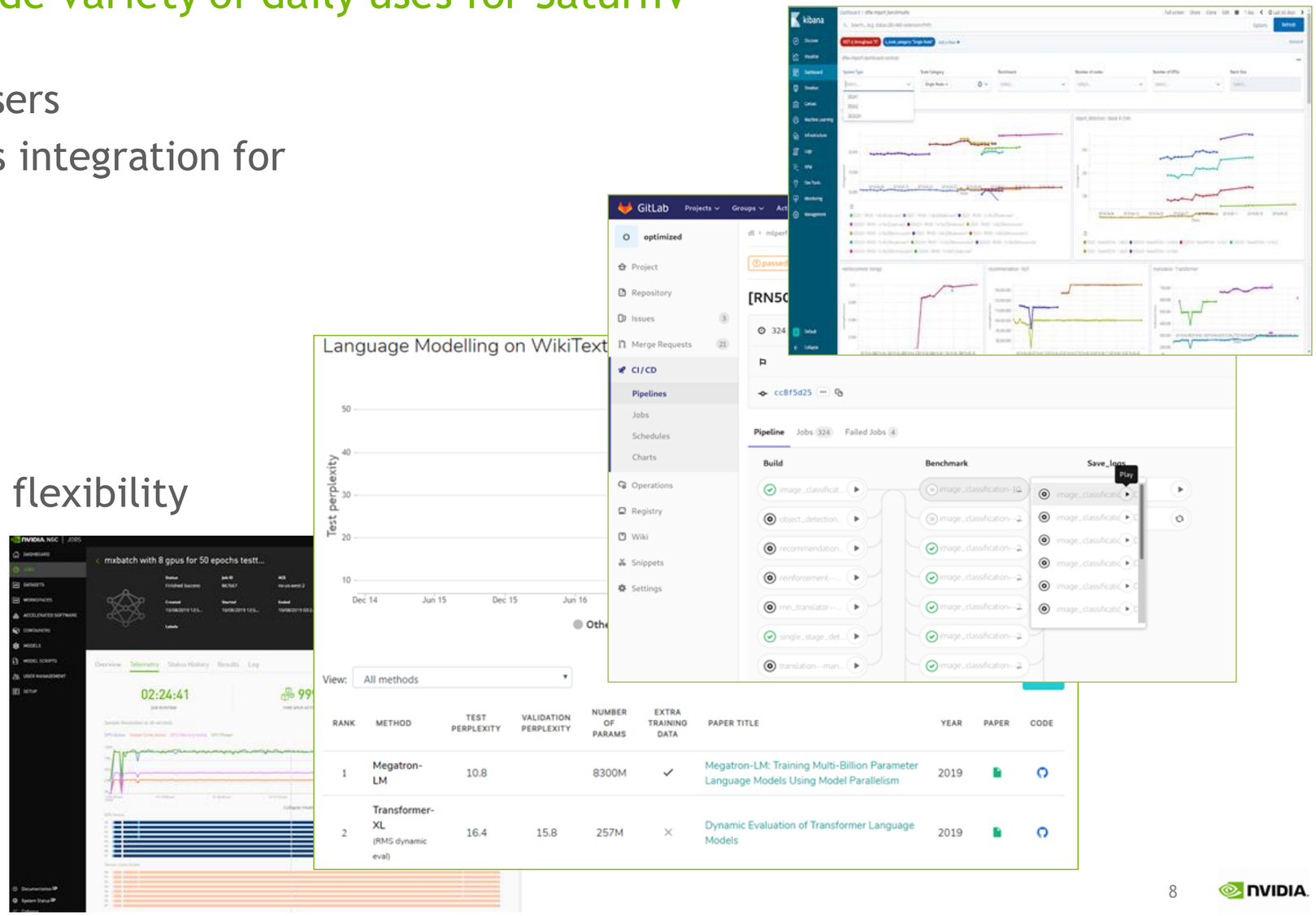
CLUSTERS AT NVIDIA

A wide variety of daily uses for SaturnV

Supporting a wide community of users

- supercomputer-scale continuous integration for software
- research
- automotive
- QA

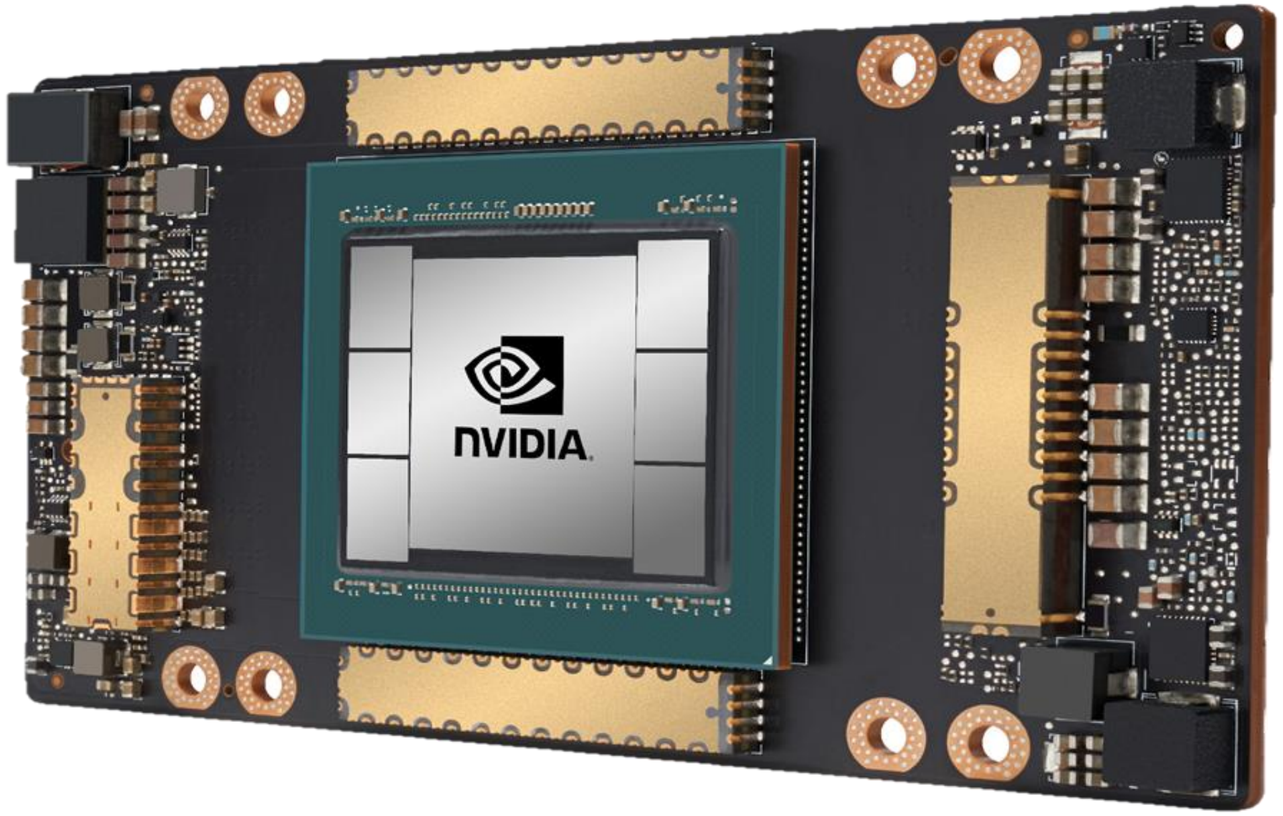
Need for performance at scale and flexibility



A new architecture

A NEW GENERATION OF GPUS

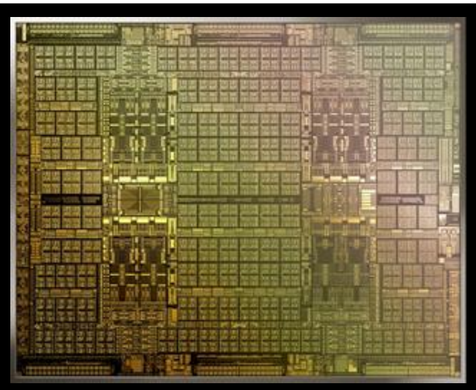
NVIDIA A100



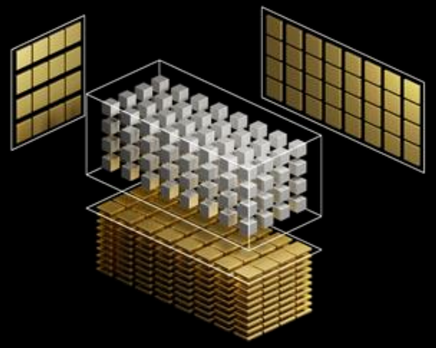
54B XTOR | 826mm2 | TSMC 7N | 40GB Samsung HBM2 | 600 GB/s NVLink

A NEW GENERATION OF GPUS

NVIDIA A100

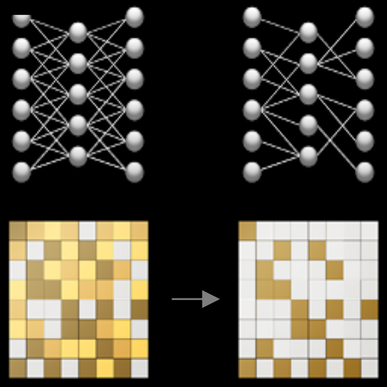


Ampere architecture
World's Largest 7nm chip
54B XTORS, HBM2

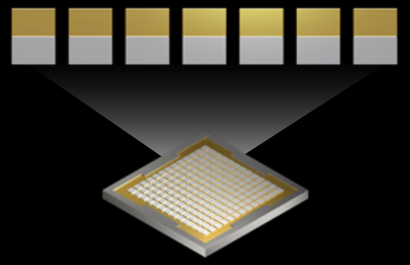


3rd Gen Tensor Cores
Faster, Flexible, Easier to use
20x AI Perf with TF32

3rd Gen NVLINK and NVSWITCH
Efficient Scaling
2X More Bandwidth

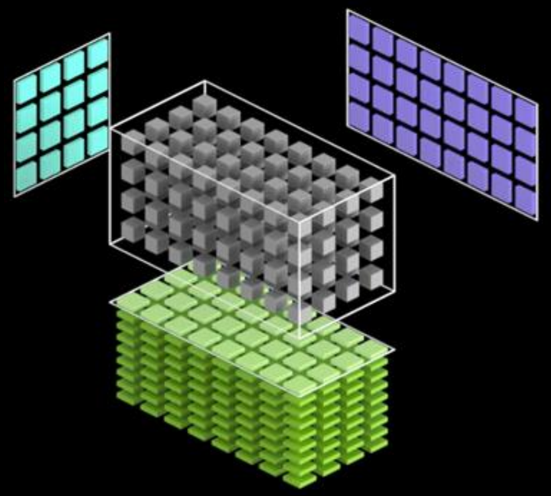


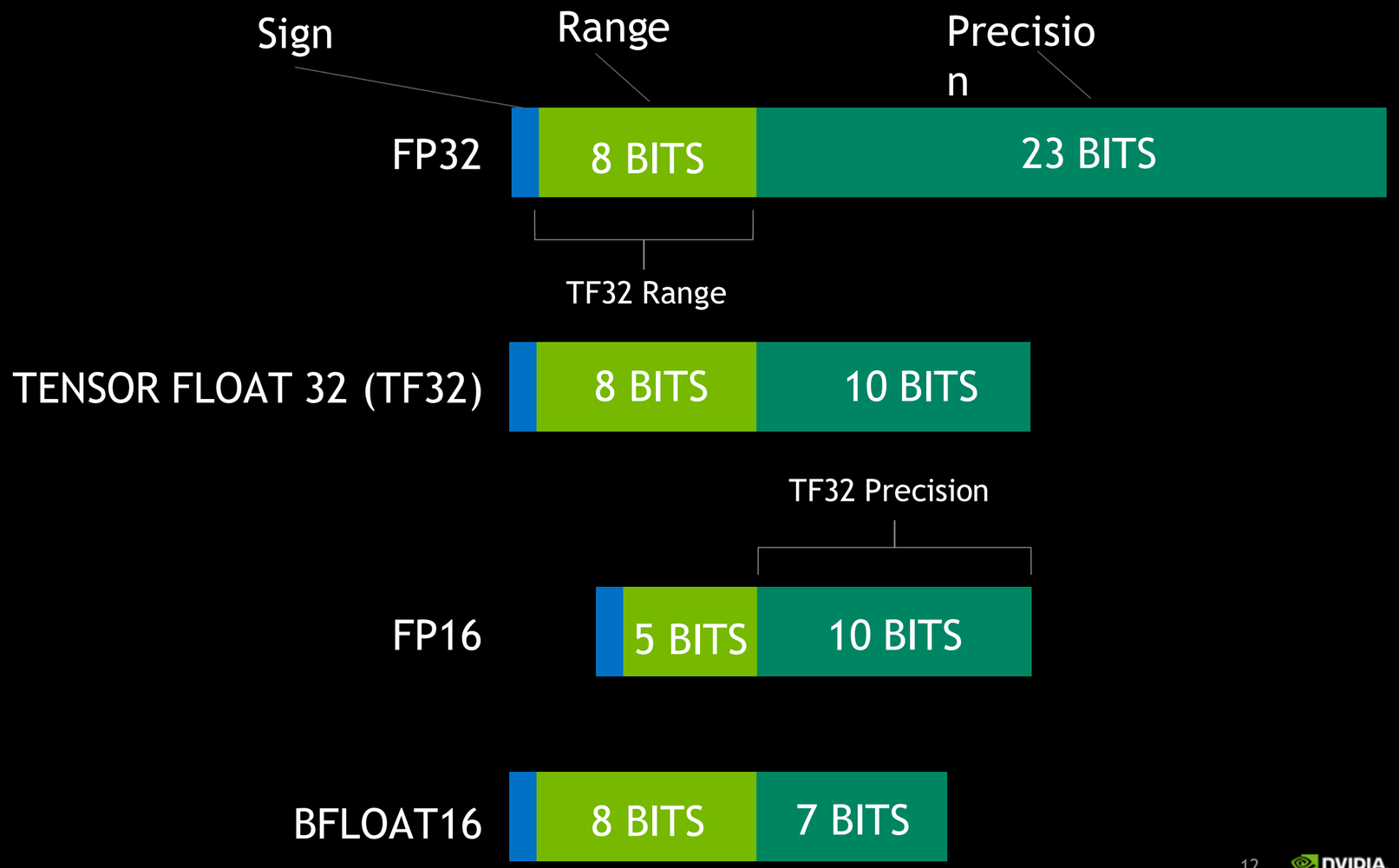
New Sparsity Acceleration
Sparsity in AI Models
2x AI Performance



New Multi-Instance GPU
7x Simultaneous Instances per GPU

NEW TF32 TENSOR CORES



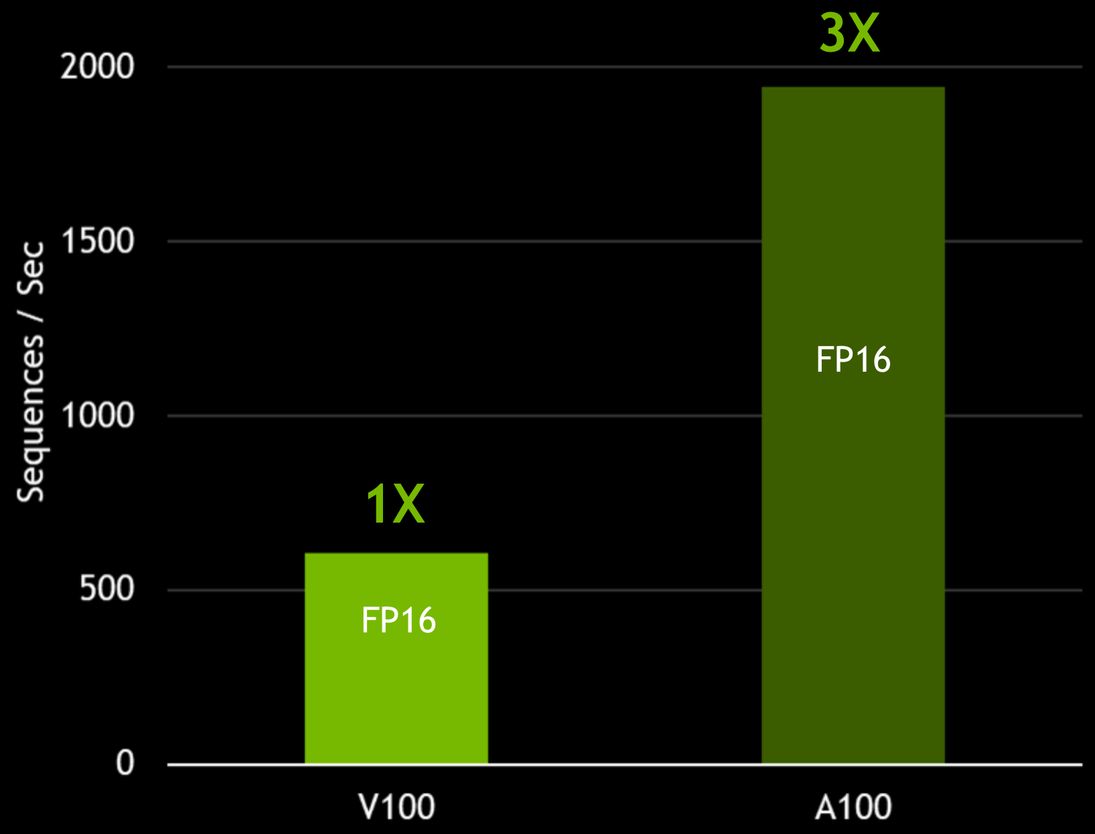


Range of FP32 and Precision of FP16

Input in FP32 and Accumulation in FP32

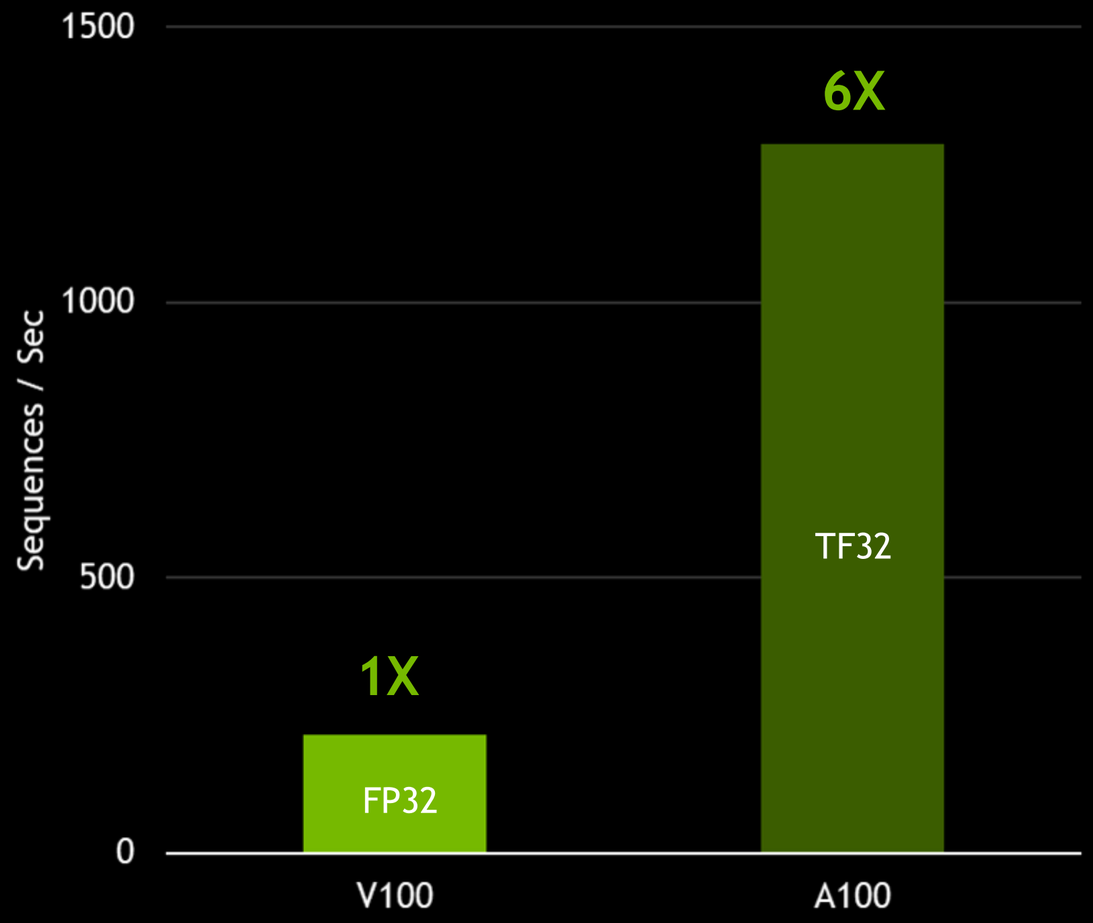
No Code Change Speed-up for Training

FP16 & AMP FOR AI TRAINING - BERT



BERT Pre-Training Throughput using Pytorch including (2/3)Phase 1 and (1/3)Phase 2 | Phase 1 Seq Len = 128, Phase 2 Seq Len = 512 V100: DGX-1 Server with 8xV100 using FP32 and FP16 precision A100: DGX A100 Server with 8xA100 using TF32 precision and FP16 |

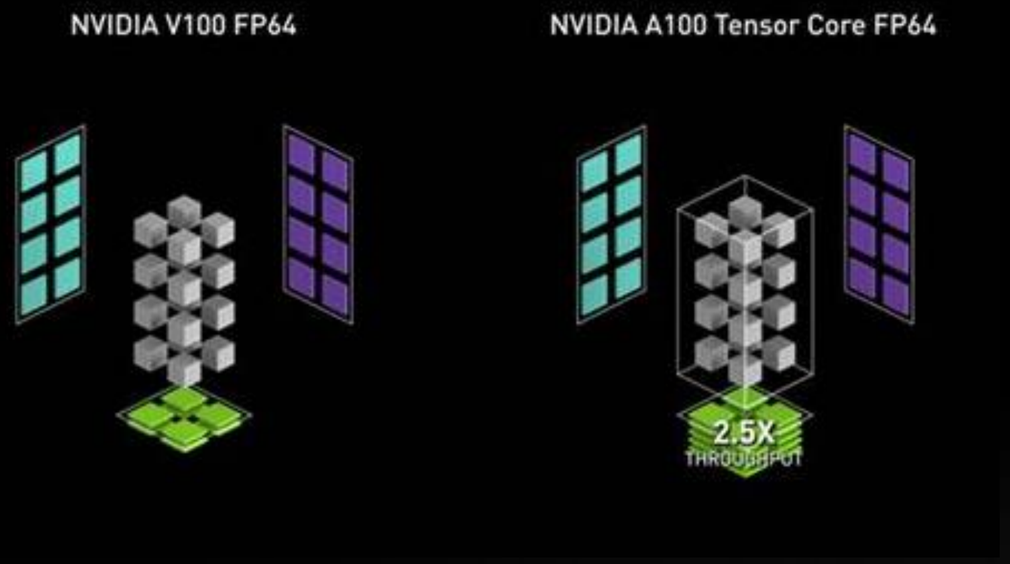
TF32 FOR AI TRAINING - BERT

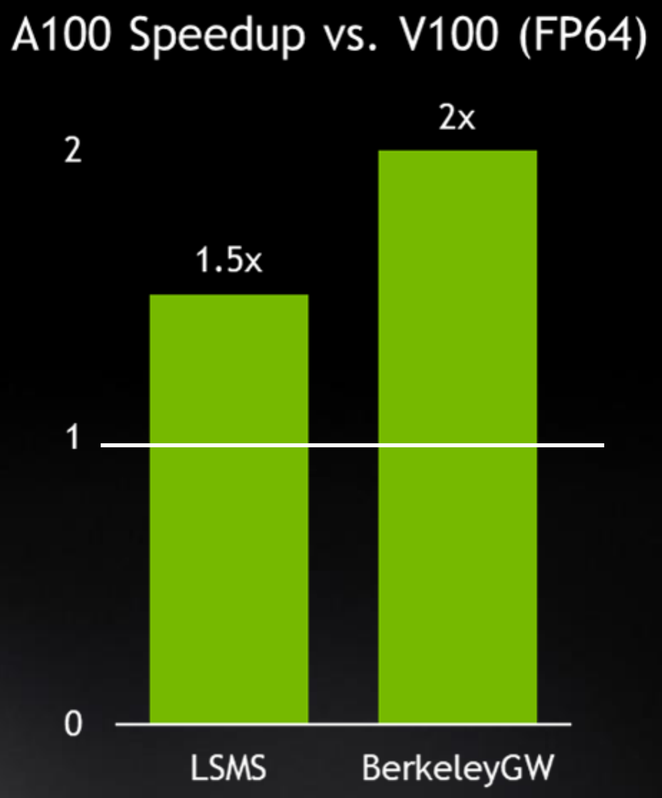


BERT Pre-Training Throughput using Pytorch including (2/3)Phase 1 and (1/3)Phase 2 | Phase 1 Seq Len = 128, Phase 2 Seq Len = 512 V100: DGX-1 Server with 8xV100 using FP32 precision
A100: DGX A100 Server with 8xA100 using TF32 precision |

A100 FP64 Tensor Cores

For even more Performance



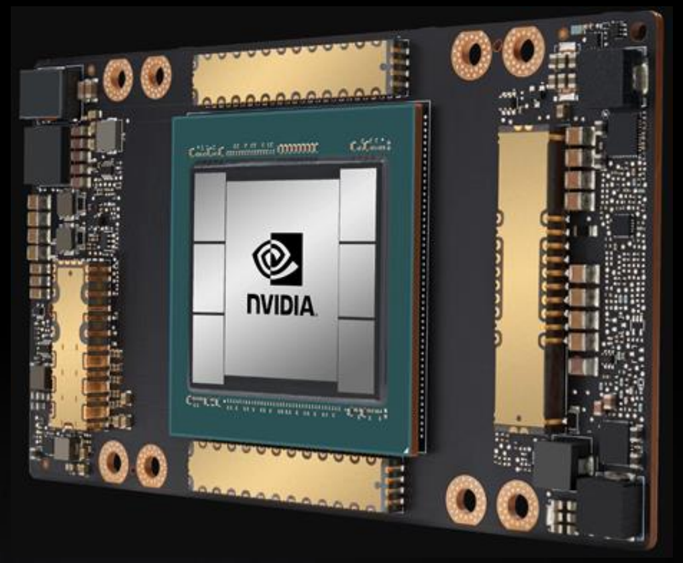


- IEEE FP64
- Drop-in acceleration via cuBLAS, cuTensor, and cuSolver

Application [Benchmarks]: BerkeleyGW [Chi Sum + MTXEL] using DGX-1V (8xV100) and DGX-A100 (8xA100) | LSMS [Fe128] single V100 SXM2 vs. A100 SXM4

NVIDIA AMPERE GPU ARCHITECTURE

	V100	A100
SMs	80	108
Tensor Core Precision	FP16, I8, I4, B1	FP64, TF32, BF16, FP16, I8, I4, B1
Shared Memory per Block	96 kB	160 kB
L2 Cache Size	6144 kB	40960 kB
Memory Bandwidth	900 GB/sec	1600 GB/sec
NVLink Interconnect	300 GB/sec	600 GB/sec




A new machine

A NEW GENERATION OF MACHINES

NVIDIA DGX A100

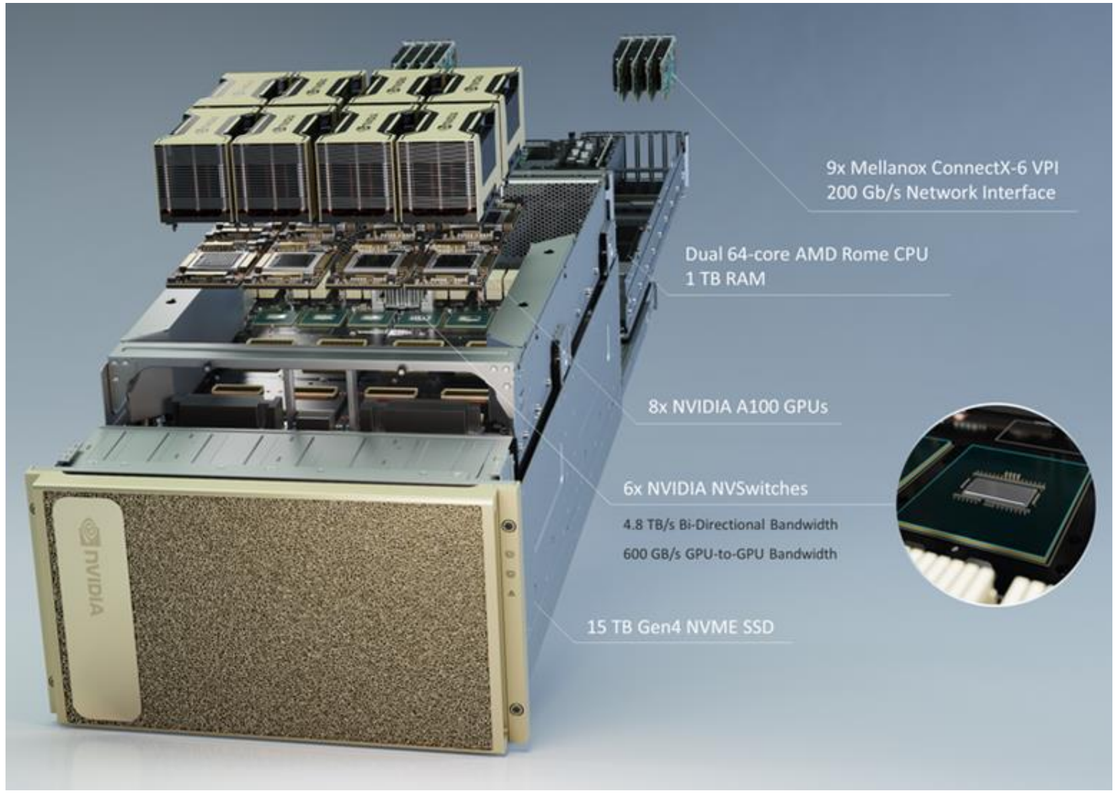
https://www.youtube.com/watch?v=TJcKYUTaBtg

A NEW GENERATION OF MACHINES

NVIDIA DGX A100

GPUs	**8x NVIDIA A100**
GPU Memory	320 GB total
Peak performance	5 petaFLOPS AI 10 petaOPS INT8
NVSwitches	6
System Power Usage	**6.5kW max**
CPU	**Dual AMD Rome 7742** 128 cores total, 2.25 GHz(base), 3.4GHz (max boost)
System Memory	**1TB**
Networking	**8x Single-Port Mellanox ConnectX-6** 200Gb/s HDR Infiniband (**Compute Network**) 1x (or **2x***) **Dual-Port Mellanox ConnectX-6** 200GB/s HDR Infiniband (**Storage Network** also used for Eth*)
Storage	OS: 2x 1.92TB M.2 NVME drives Internal Storage: 15TB (4x 3.86TB) U.2 NVME drives
Software	Ubuntu Linux OS (**5.3+ kernel**)
System Weight	271 lbs (123 kgs)
Packaged System Weight	315 lbs (143 kgs)
Height	**6U**
Operating temperature range	**5C to 30C** (41F to 86F)

* Optional upgrades



DGX A100

Specs Comparison

- 8x A100-SXM4-40GB GPUs
  - Power Limit 400 W
  - Max <GPU, mem> clks= <1410,1215> MHz
  - GPU Memory BW (SOL) = 1555 GB/s
- NVLink 3.0 (300 GB/s unidirectional BW SOL)
- PCIe Gen 4
- 10x Mellanox CX-6 IB cards
- Ubuntu 18.04.3 LTS
- Ubuntu HWE edge kernel (5.3)

System Info	DGX-2H	DGX A100
GPU	16x V100-SXM3-32GB-H	8x A100-SXM4-40GB
SMs / GPU	80	108
Mem / GPU	32 GB	40 GB
Max <GPU, Mem> Clocks	1702,1107 MHz	1410,1215 MHz
GPU Memory BW	1134 GB/s	1555 GB/s
GPU TDP	450 W	400 W
CPU Info	24 core Intel Xeon Platinum 8174	64 core AMD EPYC 7742
System RAM	1.5 TB	2 TB
NVLink (BW per GPU)	2.0 (150 GB/s)	3.0 (300 GB/s)
IB Network	8x 100 Gb/s	8x 200 Gb/s

DGX A100

Two AMD Rome CPUs

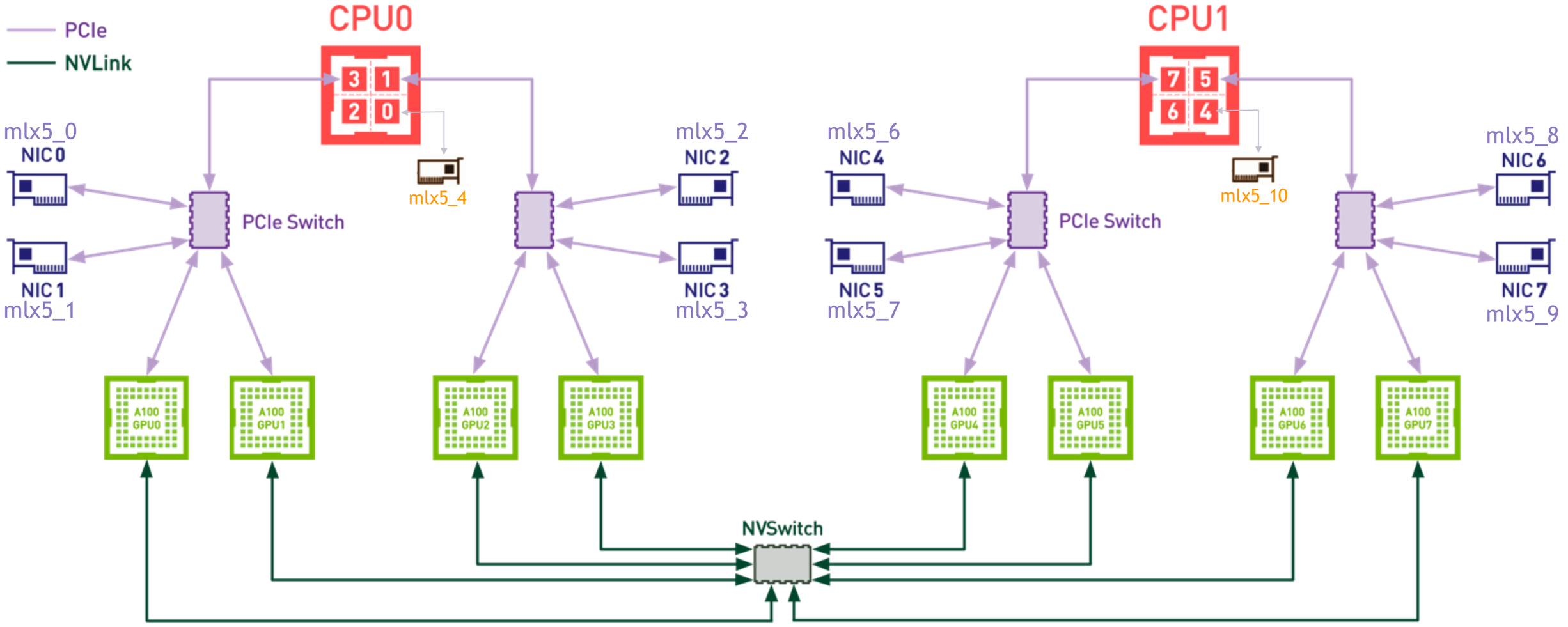
AMD EPYC 7742 (aka AMD Rome)

- 64 cores
- 9-die MCM (multichip module)
- 4 xGMI-2 (~18Gbps/channel) cross-socket interconnect
- CPU Base <min, max> MHz: <1500, 2250>
- Max Boost Clocks 3400 MHz
- cTDP = 225 W

System Info	DGX-2H	DGX A100
CPU Info	Intel Xeon Platinum 8174	AMD EPYC 7742
Cores / Socket	24	64
Total CPU threads	96	256
CPU NUMA Nodes per Socket	1	4
L2 Cache	1024 KB	512 KB
L3 Cache	33792K (33 MB)	16384 K /CCX 256 MB total
PCIe	Gen 3	Gen 4

DGX A100

High-level Topology Overview (with options)



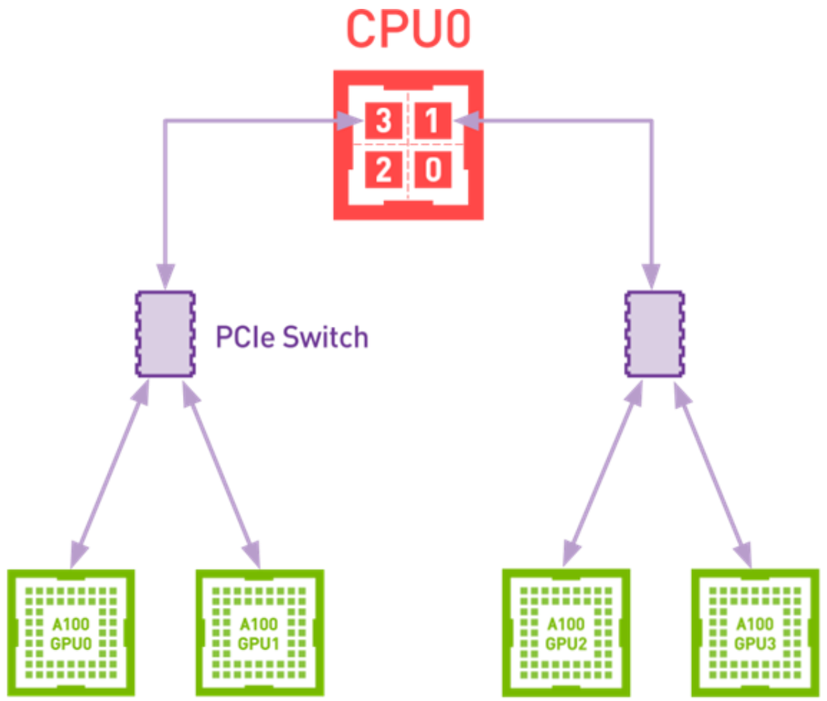
DGX A100

AMD Rome and NUMA

NPS = NUMA nodes per socket

- On DGX A100, **NPS = 4** (8 total NUMA nodes across two sockets).
- Each NUMA node has 16 physical cores and 2 memory channels providing total 50 GB/s DRAM BW (SOL).

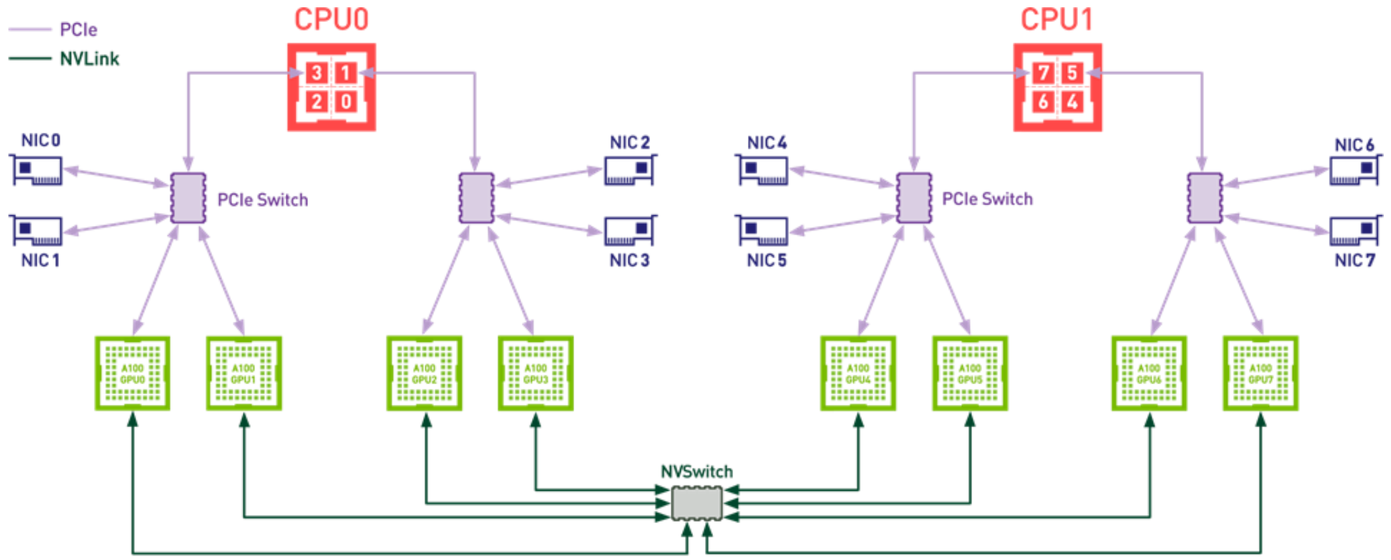
```
$ lscpu
...
NUMA node0 CPU(s):   0-15,128-143
NUMA node1 CPU(s):   16-31,144-159
NUMA node2 CPU(s):   32-47,160-175
NUMA node3 CPU(s):   48-63,176-191
NUMA node4 CPU(s):   64-79,192-207
NUMA node5 CPU(s):   80-95,208-223
NUMA node6 CPU(s):   96-111,224-239
NUMA node7 CPU(s):   112-127,240-255
```



DGX A100

GPUs and NUMA Affinity

Each pair of GPUs is affined to 1 NUMA node. Each pair of GPUs is connected to the CPU through a Gen4 PCIe switch. 4 NUMA nodes have affined GPU pairs; 4 NUMA nodes do not.



GPUs [0,1] <-> NUMA node 3

GPUs [2,3] <-> NUMA node 1

GPUs [4,5] <-> NUMA node 7

GPUs [6,7] <-> NUMA node 5

```
$ nvidia-smi topo -m

        GPU0    GPU1    GPU2    GPU3    GPU4    GPU5    GPU6    GPU7    ...  CPU Affinity        [NUMA Node]
GPU0     X      NV12    NV12    NV12    NV12    NV12    NV12    NV12    ...  48-63,176-191       3
GPU1    NV12     X      NV12    NV12    NV12    NV12    NV12    NV12    ...  48-63,176-191       3
GPU2    NV12    NV12     X      NV12    NV12    NV12    NV12    NV12    ...  16-31,144-159       1
GPU3    NV12    NV12    NV12     X      NV12    NV12    NV12    NV12    ...  16-31,144-159       1
GPU4    NV12    NV12    NV12    NV12     X      NV12    NV12    NV12    ...  112-127,240-255     7
GPU5    NV12    NV12    NV12    NV12    NV12     X      NV12    NV12    ...  112-127,240-255     7
GPU6    NV12    NV12    NV12    NV12    NV12    NV12     X      NV12    ...  80-95,208-223       5
GPU7    NV12    NV12    NV12    NV12    NV12    NV12    NV12     X      ...  80-95,208-223       5

Legend:
  X    = Self
  ...
  NV#  = Connection traversing a bonded set of # NVLinks
```

A new datacenter design

DGX A100 SuperPOD

Fast deployment ready
Cold aisle containment design

DGX A100 SuperPOD

A modular model

1K GPU SuperPOD Cluster

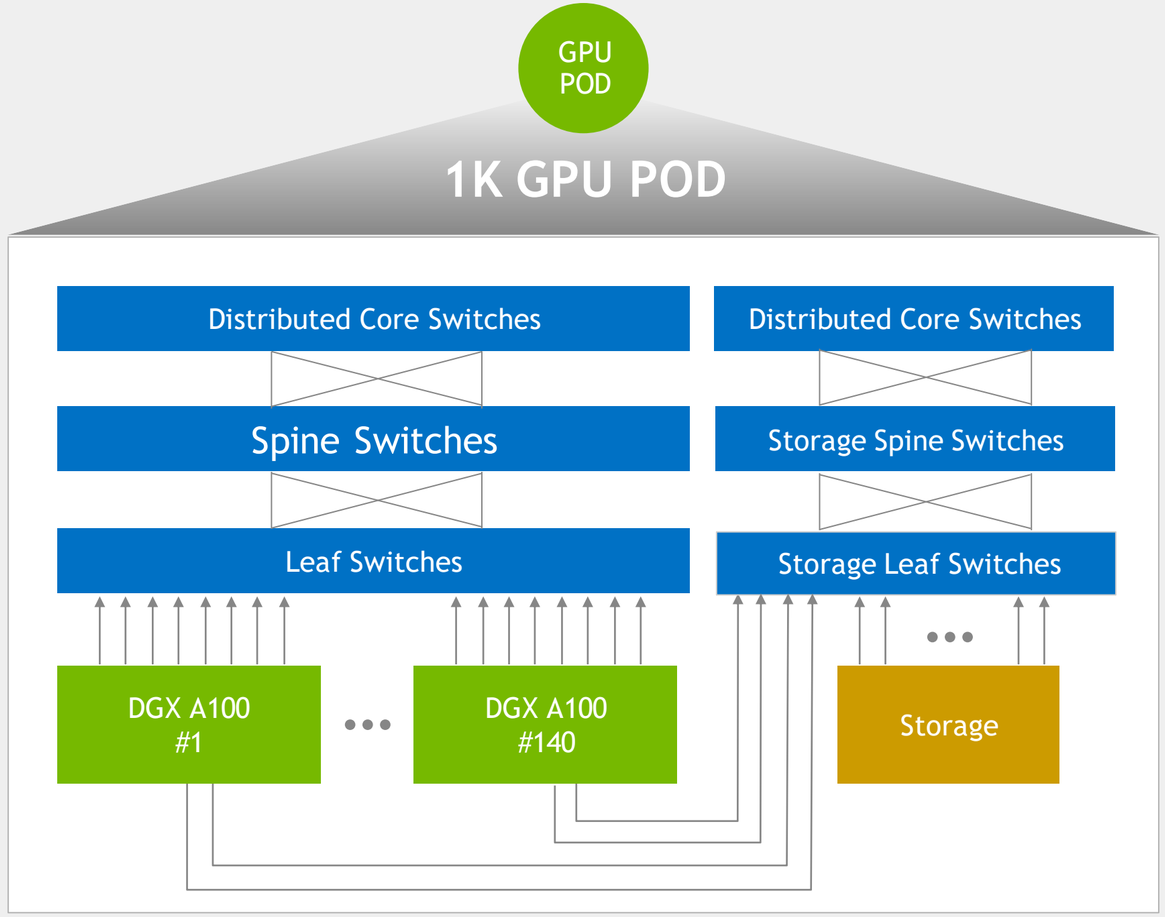
- 140 DGX A100 nodes (1120 GPUs) in a GPU POD
- 1st tier fast storage - DDN AI400x with Lustre
- Mellanox HDR 200Gb/s InfiniBand - Full Fat-tree
- Network optimized for AI and HPC

DGX A100 Nodes

- 2x AMD 7742 EPYC CPUs + 8x A100 GPUs
- NVLINK 3.0 Fully Connected Switch
- 8 Compute + 2 Storage HDR IB Ports

A fast interconnect

- Modular IB Fat-tree
- Separate network for Compute vs Storage
- Adaptive routing and SharpV2 support for offload

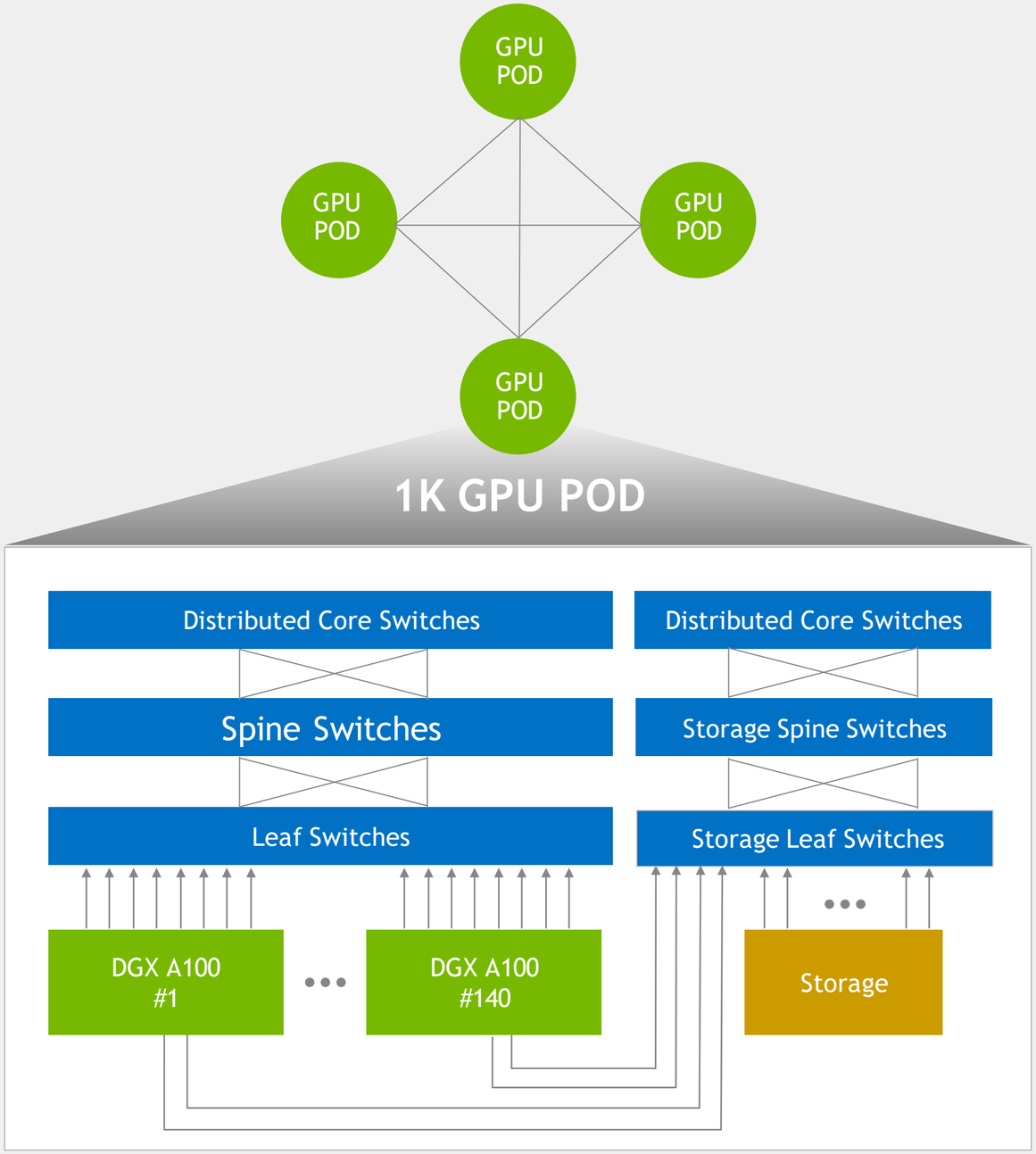


The DGXA100 Superpod

An extensible model

POD to POD

- Modular IB Fat-tree
  - Core IB Switches Distributed Between PODs
  - Direct connect POD to POD
- Separate network for Compute vs Storage
- Adaptive routing and SharpV2 support for offload

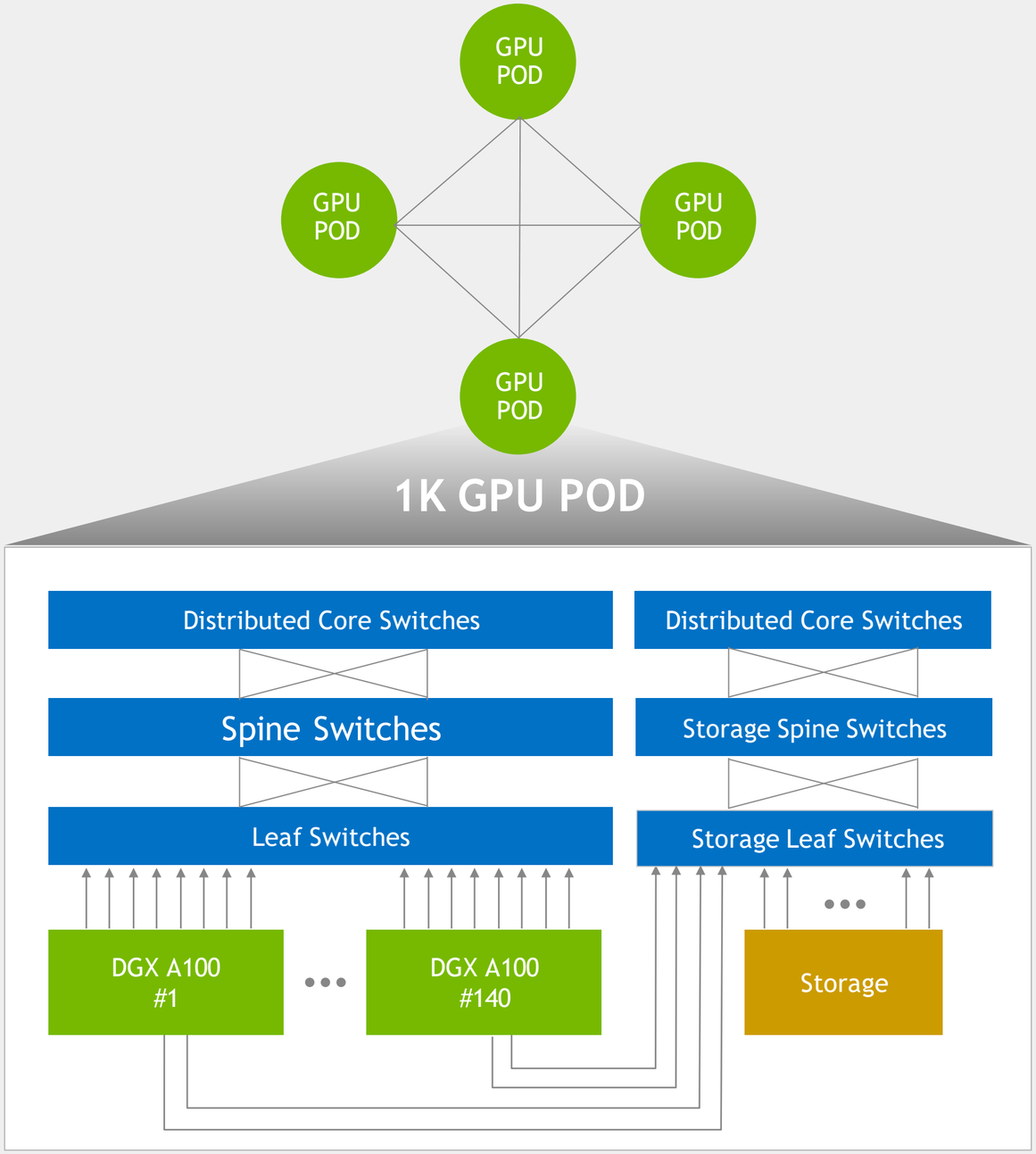


The DGXA100 Superpod

An extensible model

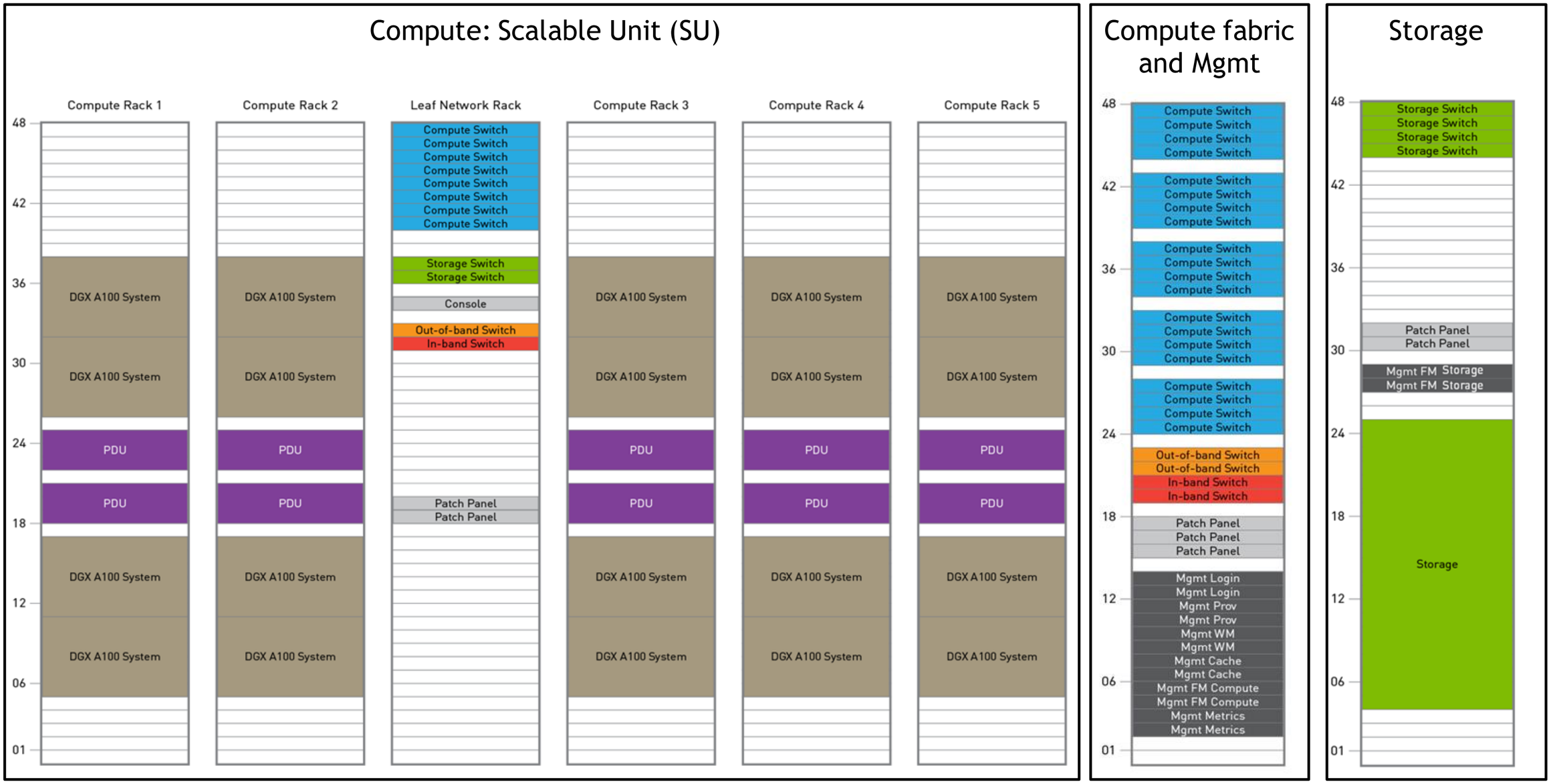
POD to POD

- Modular IB Fat-tree
  - Core IB Switches Distributed Between PODs
  - Direct connect POD to POD
- Separate network for Compute vs Storage
- Adaptive routing and SharpV2 support for offload



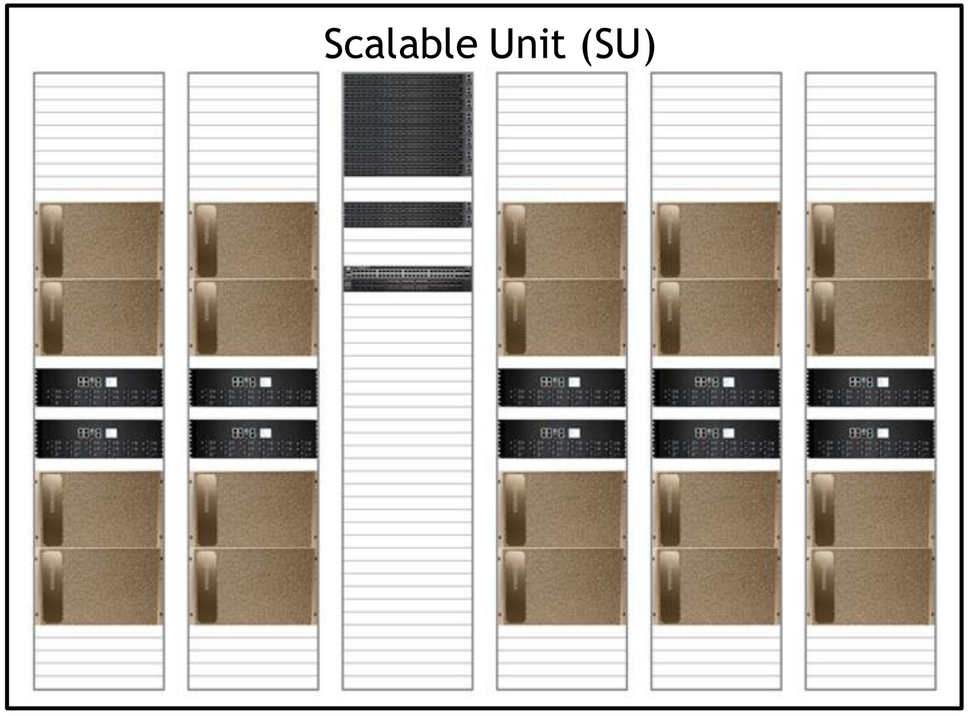
DESIGNING FOR PERFORMANCE

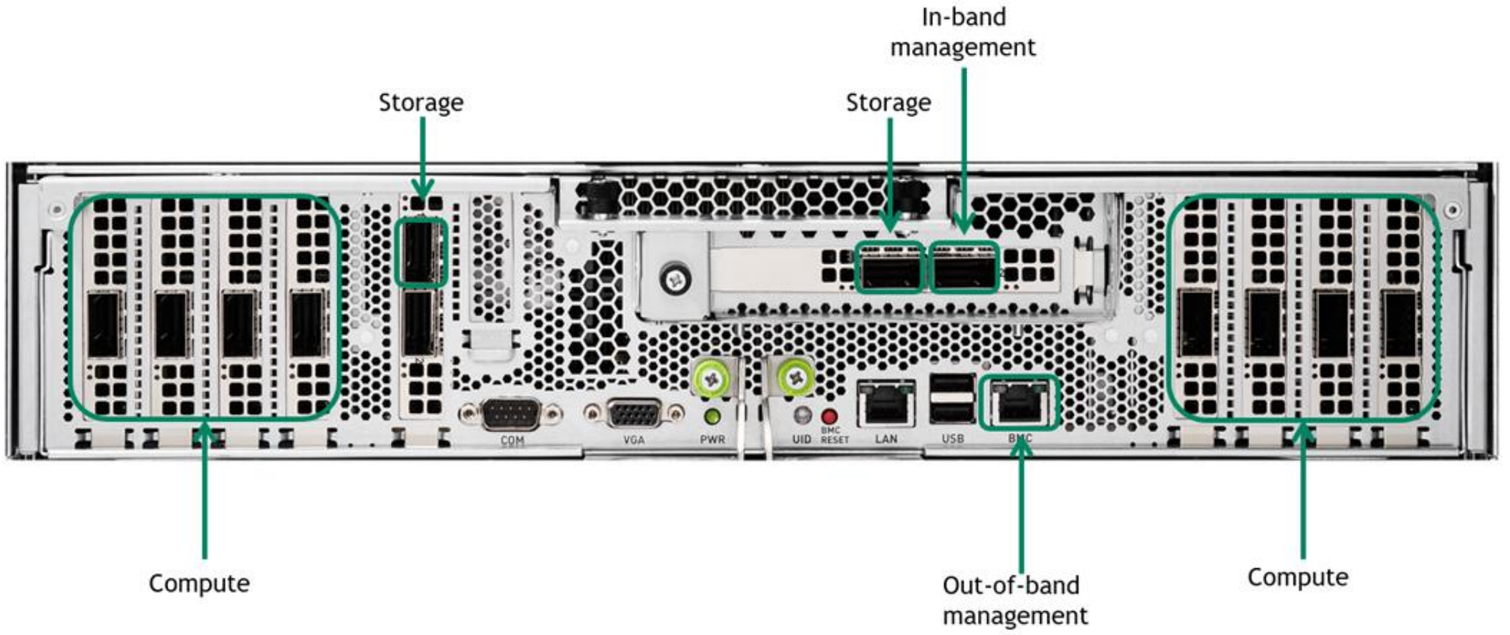
In the datacenter



DESIGNING FOR PERFORMANCE

In the datacenter





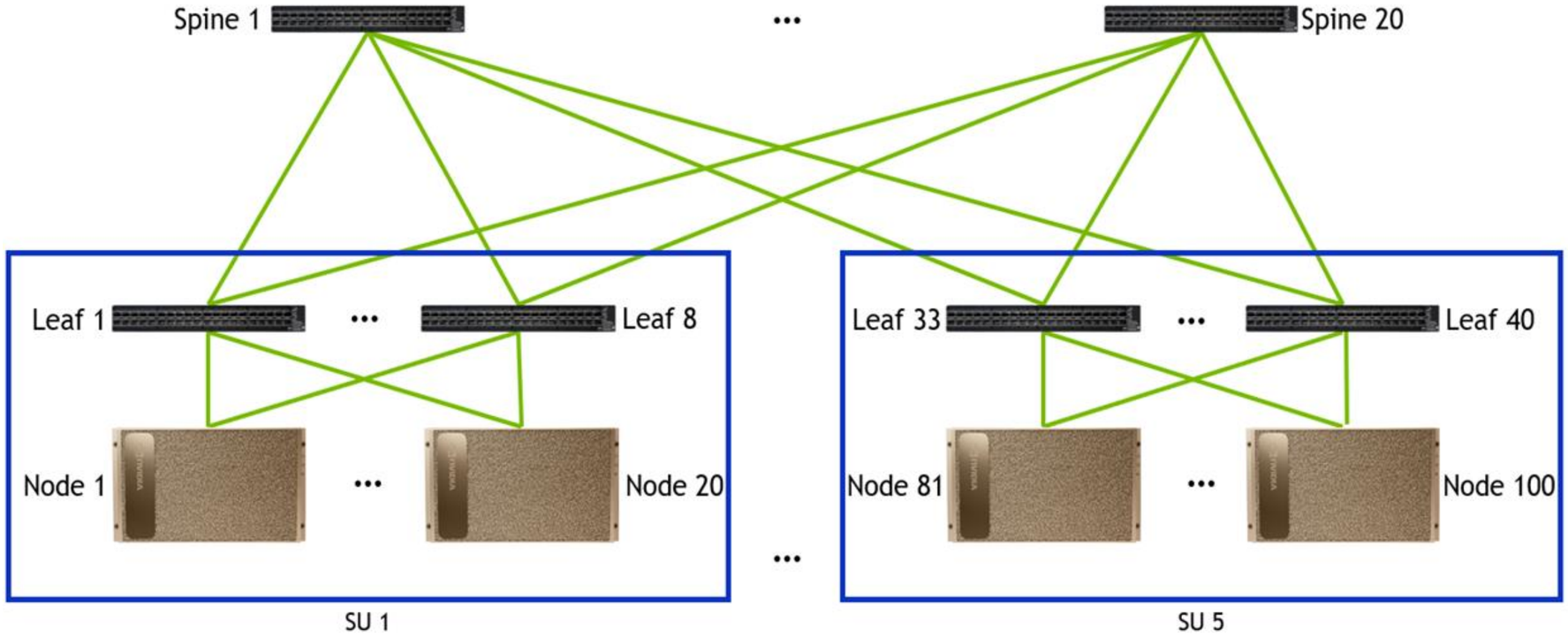
A POD at any scale

Growing with Scalable Units (SU)

Nodes	SUs	QM8790 Switches			Cables		
		Leaf	Spine	Core	Leaf	Spine	Core
10	½	8	2		80	80	
20 (Single SU)	1	8	4		160	160	
40	2	16	10		320	320	
80	4	32	20		640	640	
100	5	40	20		800	800	
140 (DGX A100 SuperPOD)	7	56	80	28	1120	1120	560

Full fat tree compute fabric

100 node example



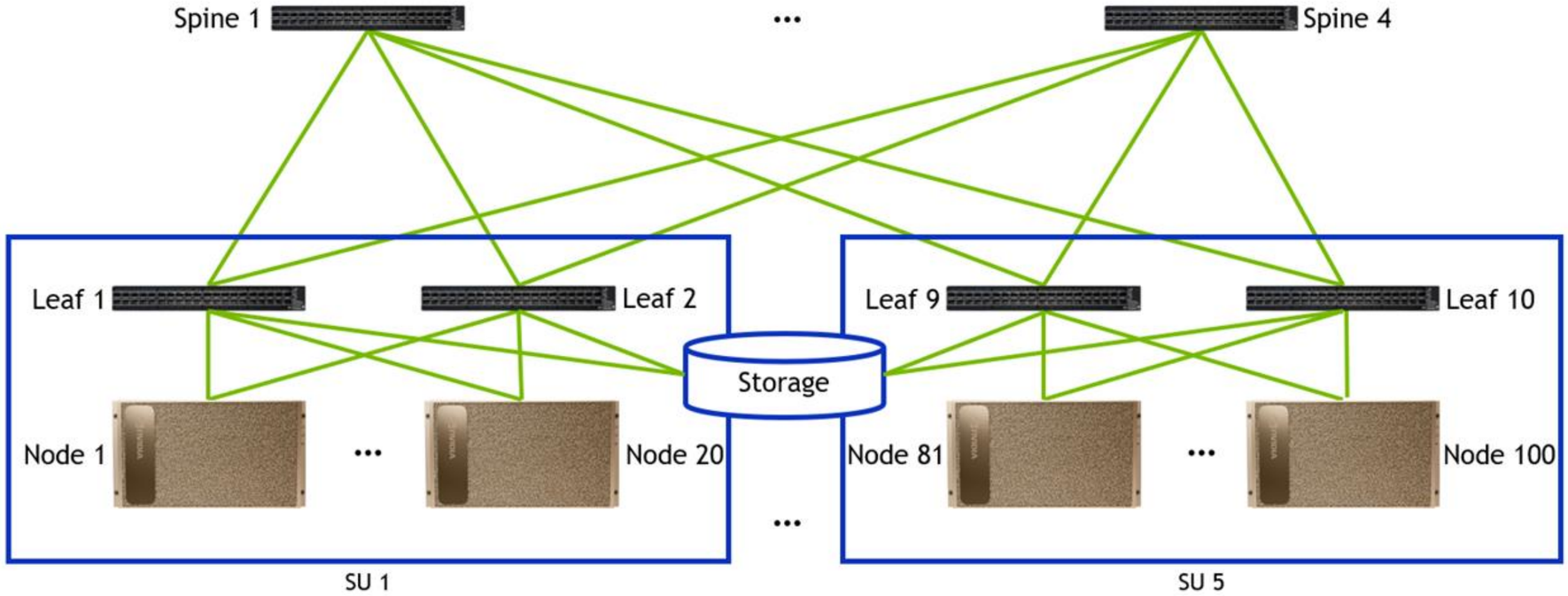
A POD at any scale

Growing with Scalable Units (SU)

Nodes	SUs	Storage Ports	QM8790 Switches		Cables			Subscription Ratio
			Leaf	Spine	Leaf	Spine	Storage	
10	1/2	4	2	1	20	20	4	1:1
20	1	8	2	1	40	32	8	3:2
40	2	16	4	2	80	64	16	3:2
80	4	32	8	4	160	128	32	3:2
100	5	40	10	4	200	160	40	3:2
140	7	56	14	8	280	224	56	5:4

Storage fabric with different ratios

100 node example



Multi node IB compute

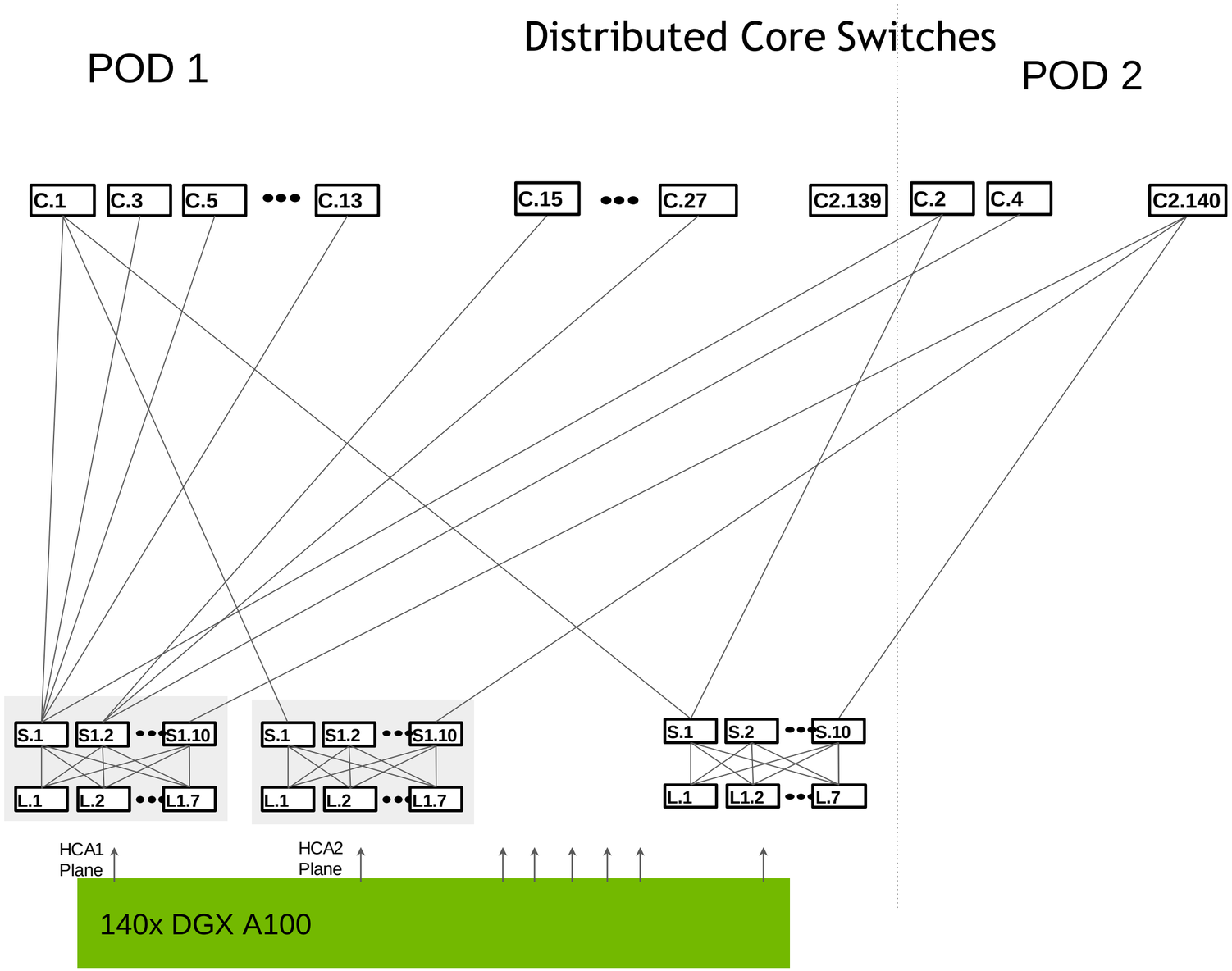
The details

Designed with Mellanox 200Gb HDR IB network

Separate compute and storage fabric
- 8 Links for compute
- 2 Links for storage (Lustre)
- Both networks share a similar fat-tree design

Modular POD design
- 140 DGX A100 nodes are fully connected in a POD
- POD contains compute nodes and storage
- All nodes and storage are usable between PODs

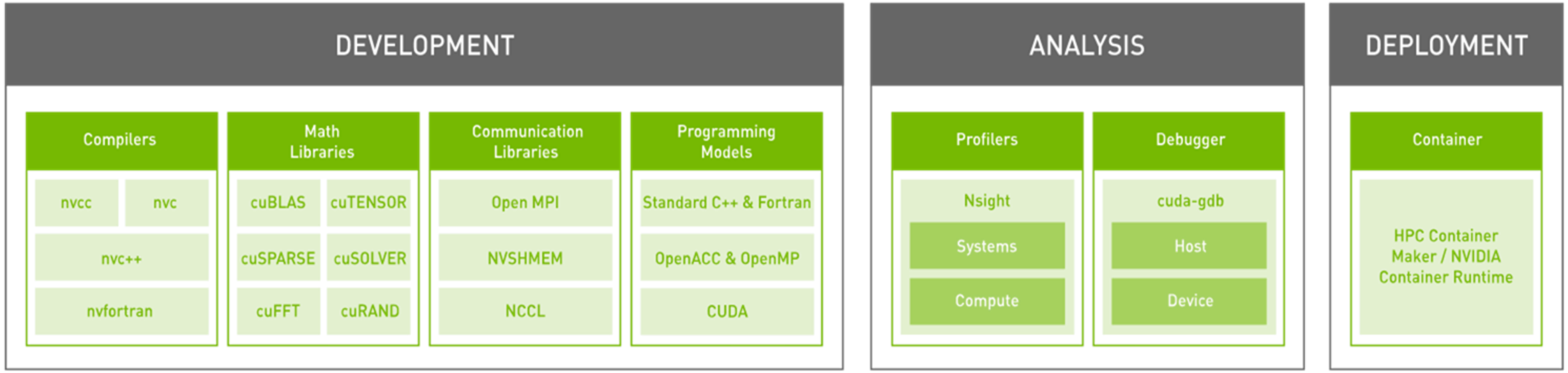
Sharp optimized design
- Leaf and Spines organized in HCA planes
- For a POD, all HCA1 from 140 DGX-2 connect to a HCA1 Plane fat-tree network
- Traffic from HCA1 to HCA1 between any two nodes in a POD stay either at the Leaf or Spine level
- Only use core switches when
  - 1. Moving data between HCA planes (e.g. mlx5_0 to mlx5_1 in another system)
  - 2. Moving any data between PODs



Software

NVIDIA HPC SDK

developer.nvidia.com/hpc-sdk

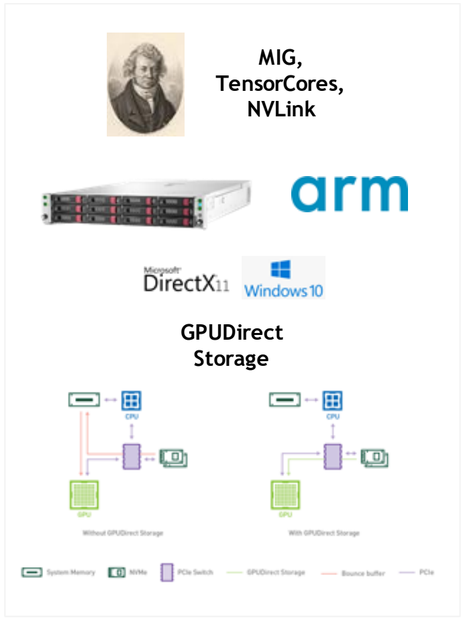


Develop for the NVIDIA HPC Platform: GPU, CPU and Interconnect

HPC Libraries | GPU Accelerated C++ and Fortran | Directives | CUDA

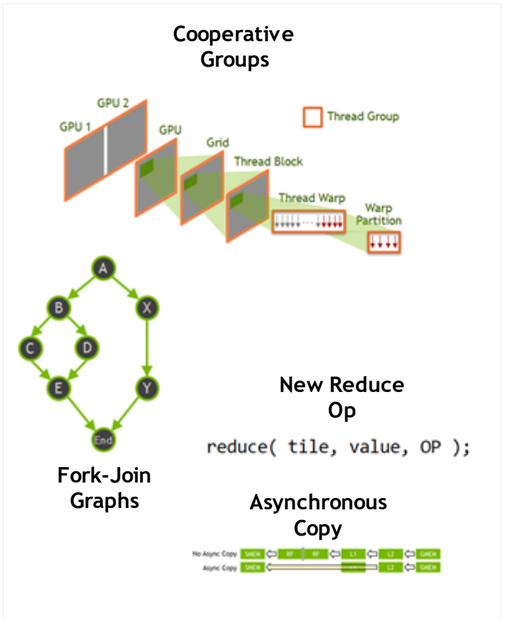
CUDA PLATFORM

Major Feature Areas for CUDA 11.0



New Platform Capabilities

- *Ampere Features*
- *CUDA on Arm Platforms*

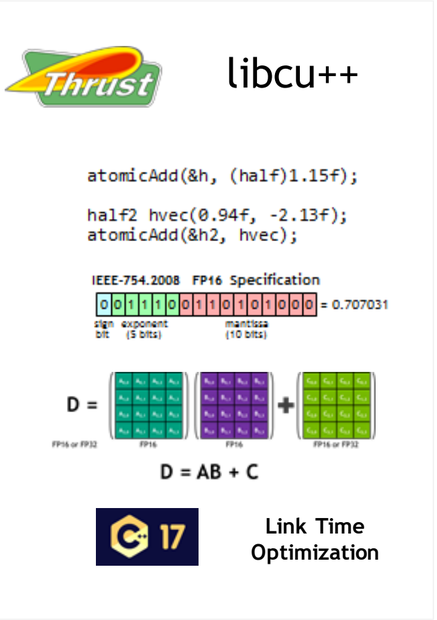


Programming Model Updates

- *Ampere Programming Model*
- *New APIs for CUDA Graphs*
- *Flexible Thread Programming*
- *Memory Management APIs*

Developer Tools

- *Support for Ampere*
- *Roofline plots with Nsight*
- *Next generation correctness tools*

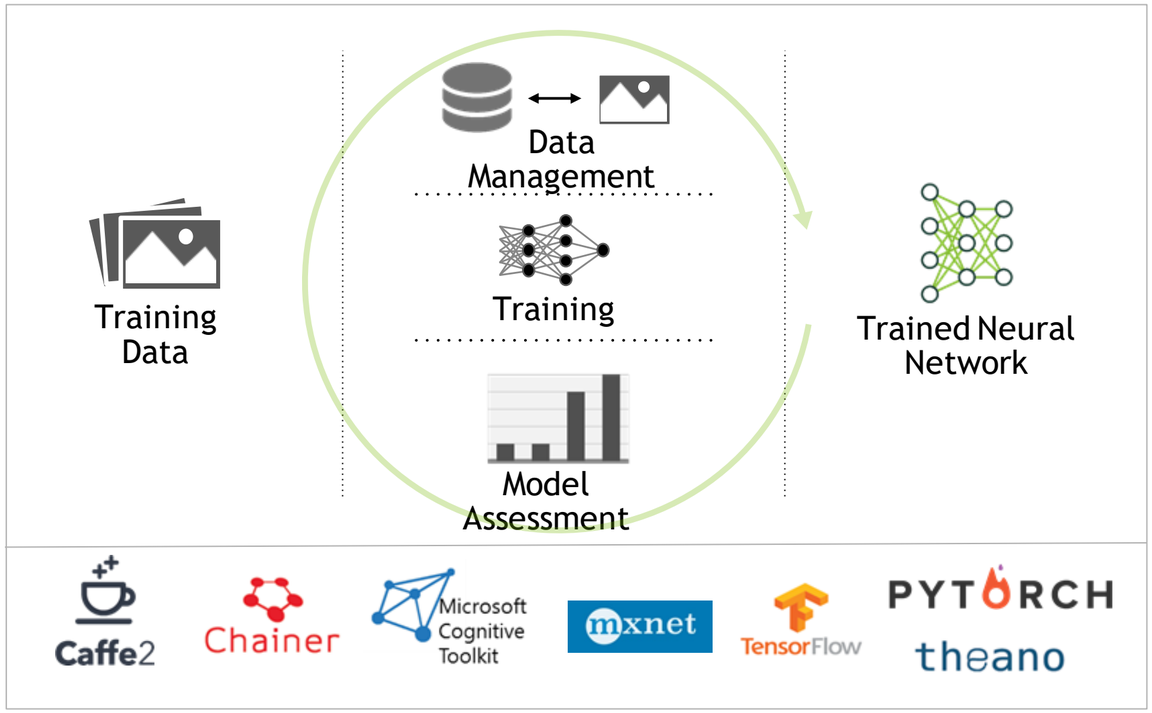


CUDA C++

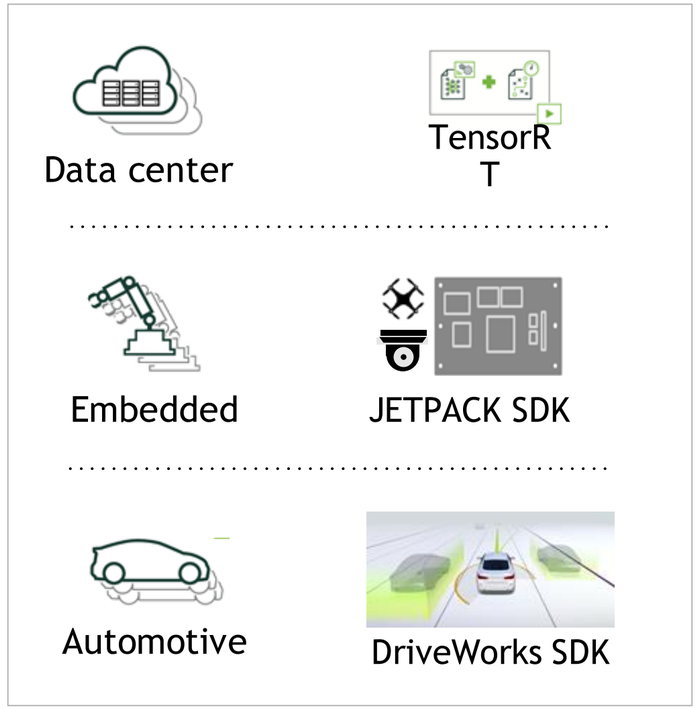
- *C++ Modernization*
- *Parallel standard C++ library*
- *Low precision datatypes and WMMA*

NVIDIA DEEP LEARNING SOFTWARE STACK

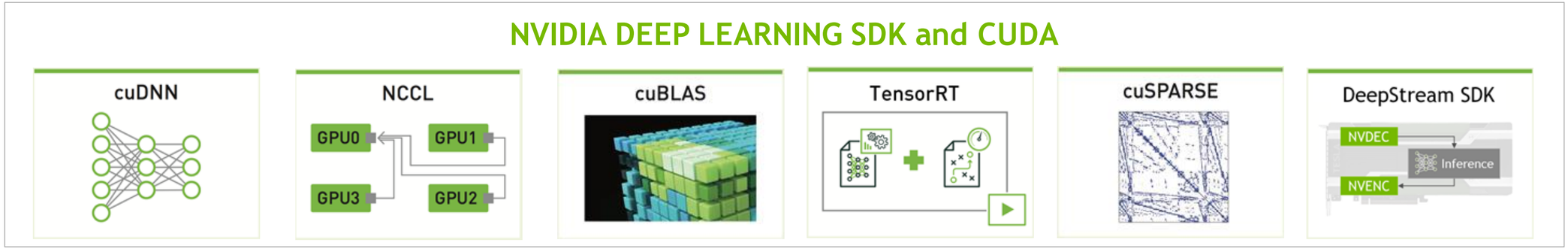
TRAINING



INFERENCE



NVIDIA DEEP LEARNING SDK and CUDA



developer.nvidia.com/deep-learning-software

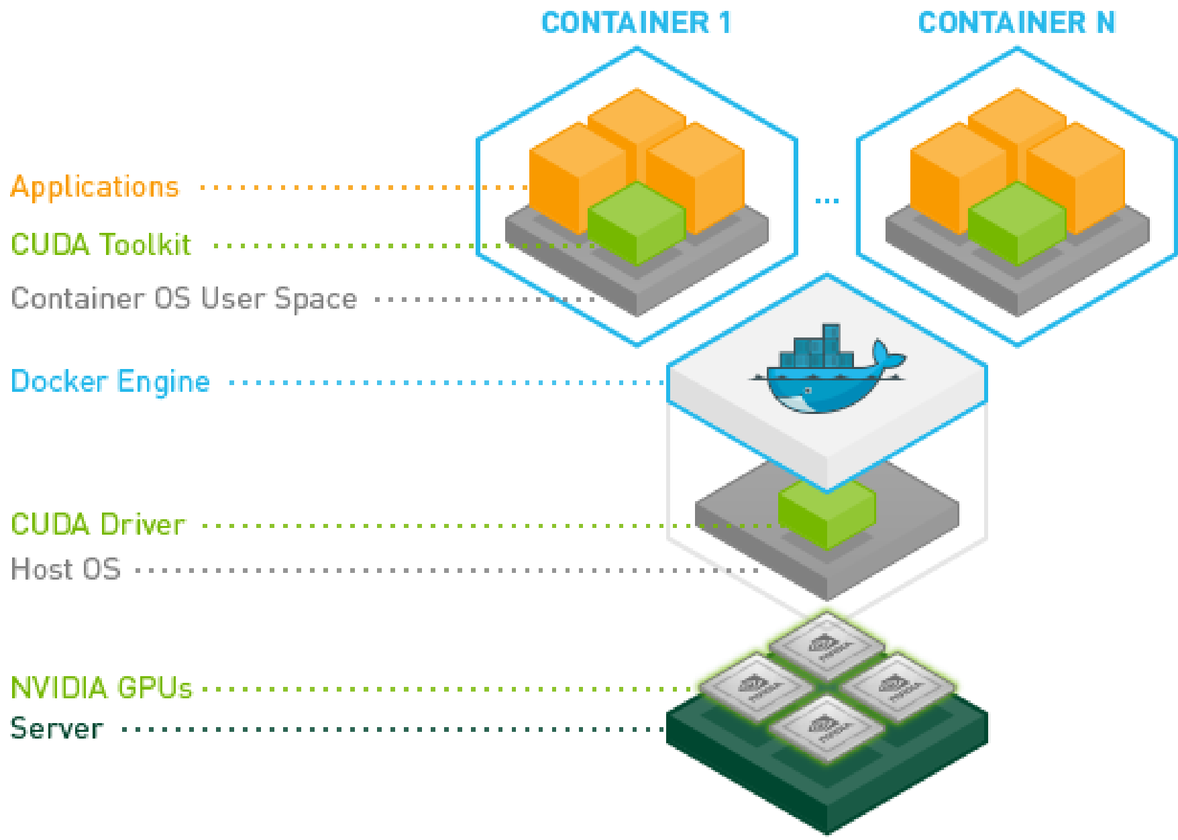
NGC CONTAINERS

We built libnvidia-container to make it easy to run CUDA applications inside containers.

We release optimized container images for each of the major DL frameworks every month, and provide them for anyone to use.

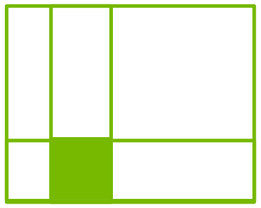
We use containers for everything on our HPC clusters - R&D, official benchmarks, etc.

Containers give us portable software stacks without sacrificing performance.



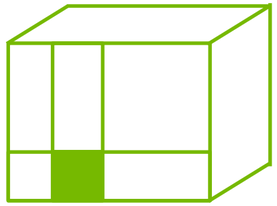
NVIDIA MATH LIBRARIES

Linear Algebra, FFT, RNG and Basic Math



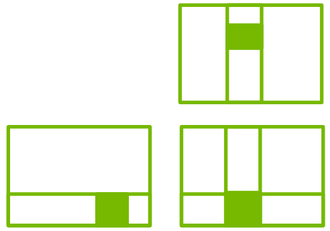
cuBLAS

cuSPARSE

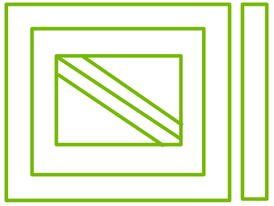


cuTENSOR

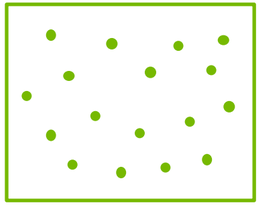
cuSOLVER



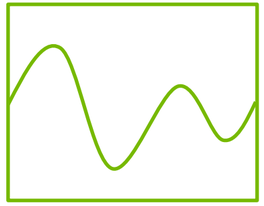
CUTLASS



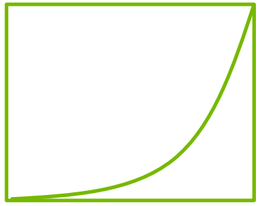
AMGX



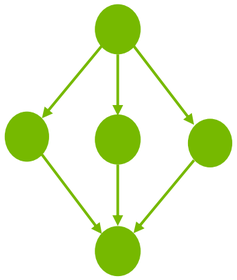
cuRAND



cuFFT



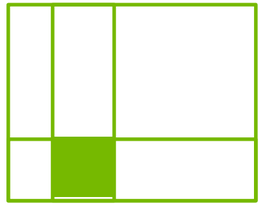
CUDA Math API



Legate

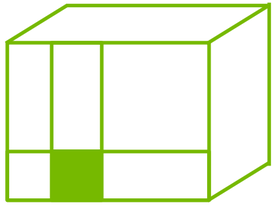
A100 TENSOR CORES IN LIBRARIES

Automatic Acceleration of Critical Routines in HPC and AI



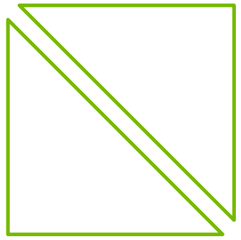
cuBLAS

- BLAS 3
  - IMMA
  - HMMA
  - Bfloat16
  - TF32
  - DMMA



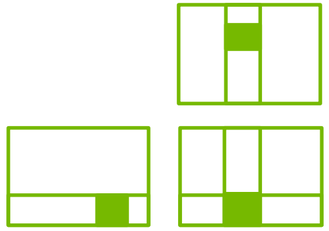
cuTENSOR

- Tensor Contraction
  - IMMA
  - HMMA
  - Bfloat16
  - TF32
  - DMMA



cuSOLVER

- Factorizations
  - DMMA
- Linear Solvers
  - HMMA
  - Bfloat16
  - TF32
  - DMMA



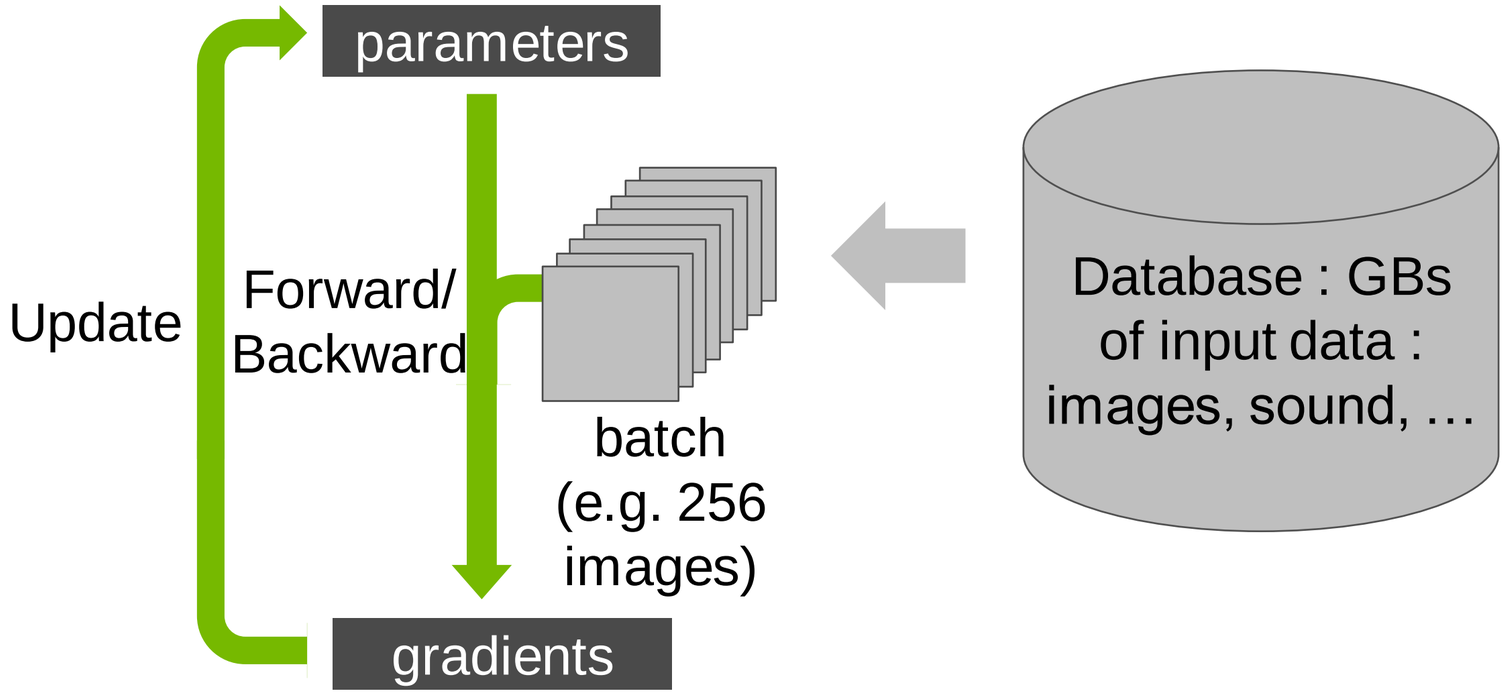
CUTLASS

- GEMM
  - IMMA
  - HMMA
  - Bfloat16
  - TF32
  - DMMA

+ Full BFloat16 support in the CUDA Math API

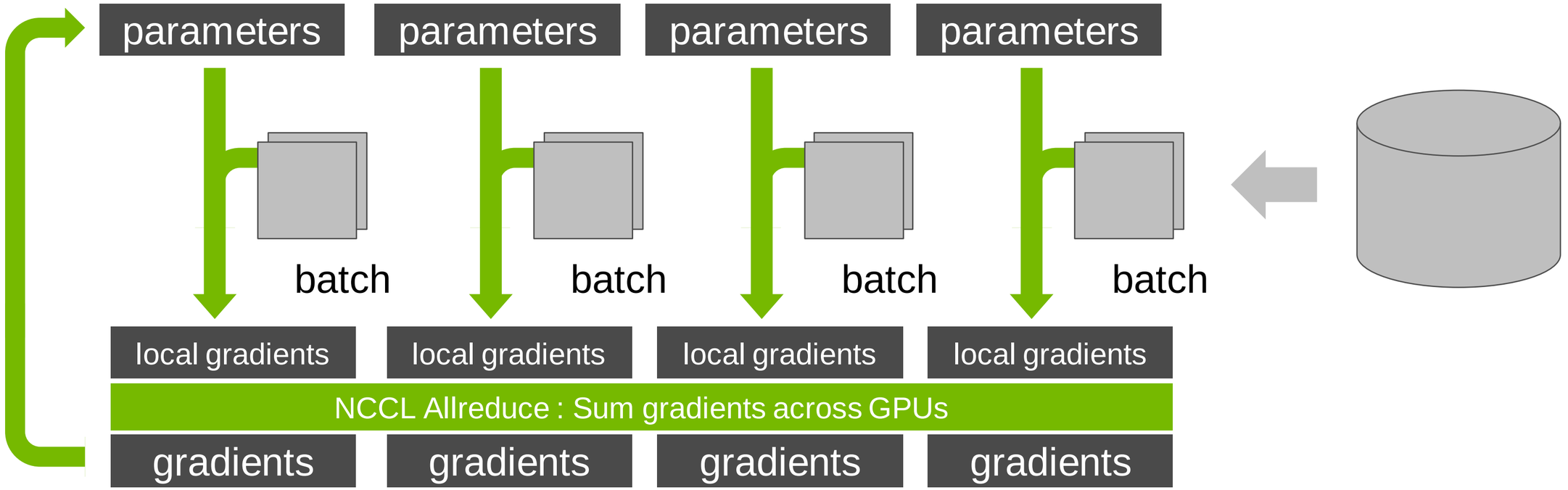
DL TRAINING

Single-GPU



DL MULTI-GPU TRAINING

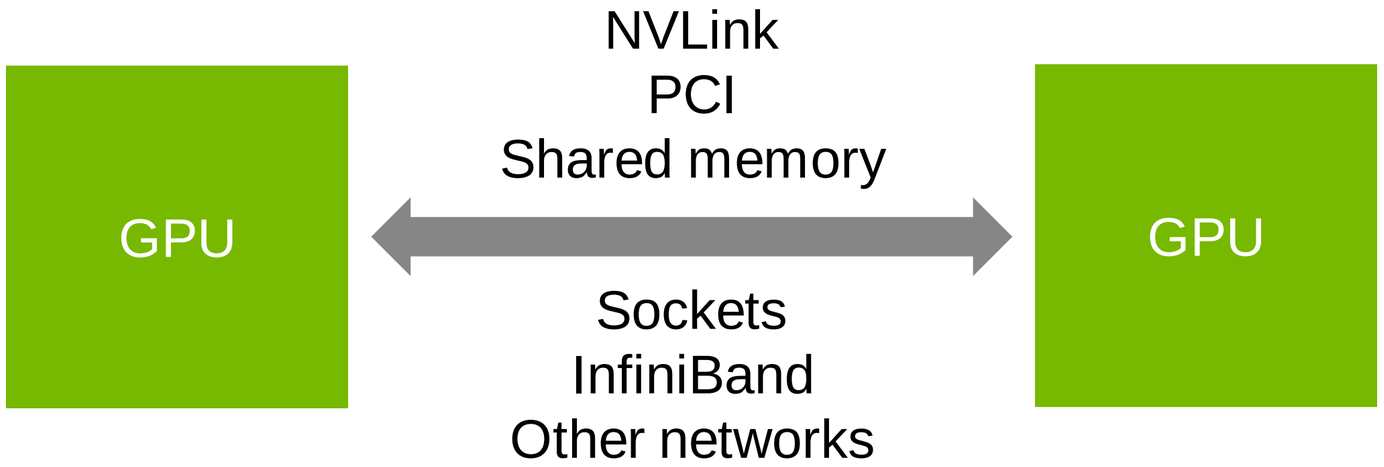
Data parallel



OPTIMIZED INTER-GPU COMMUNICATION

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

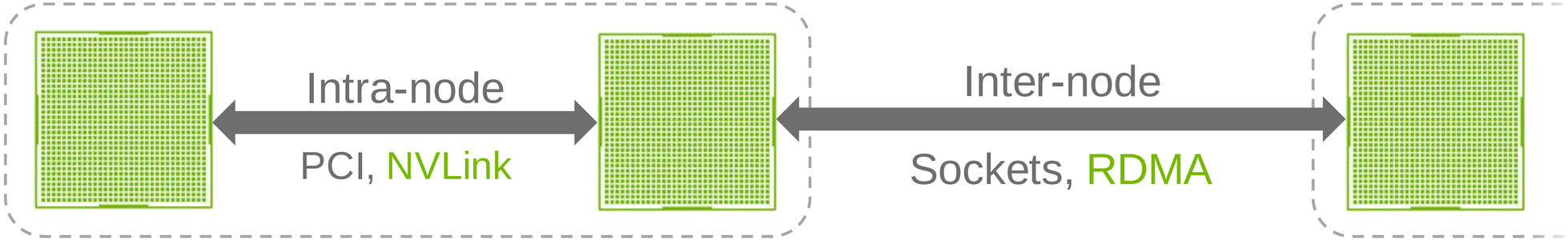
Communication library running on GPUs, for GPU buffers.

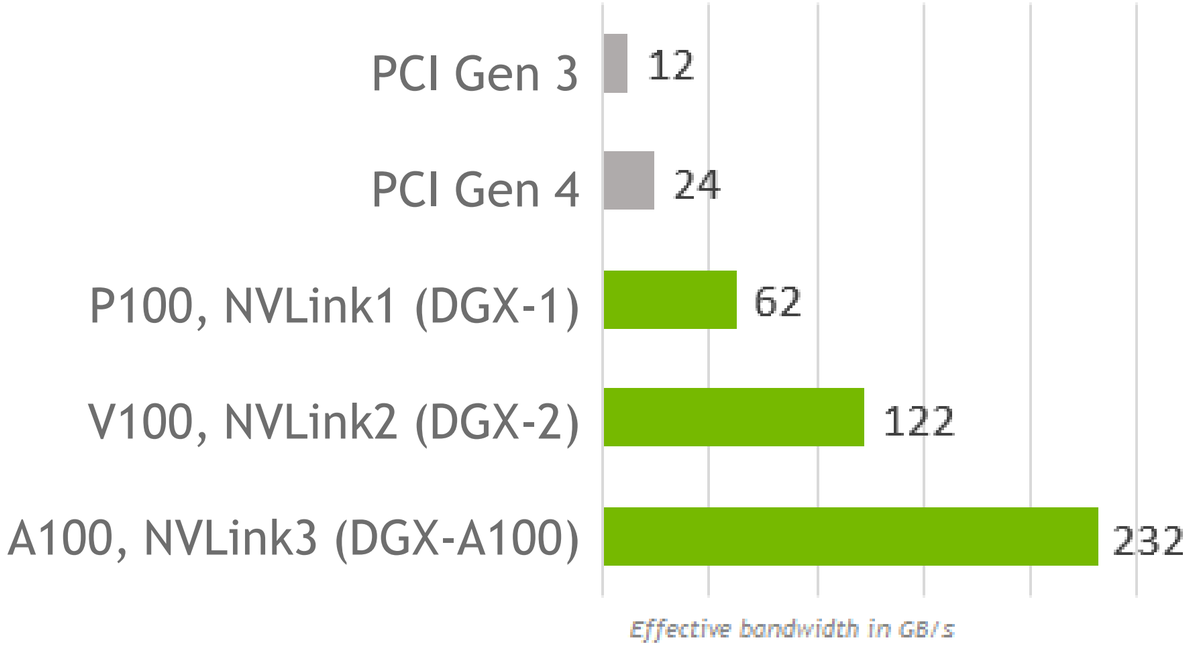


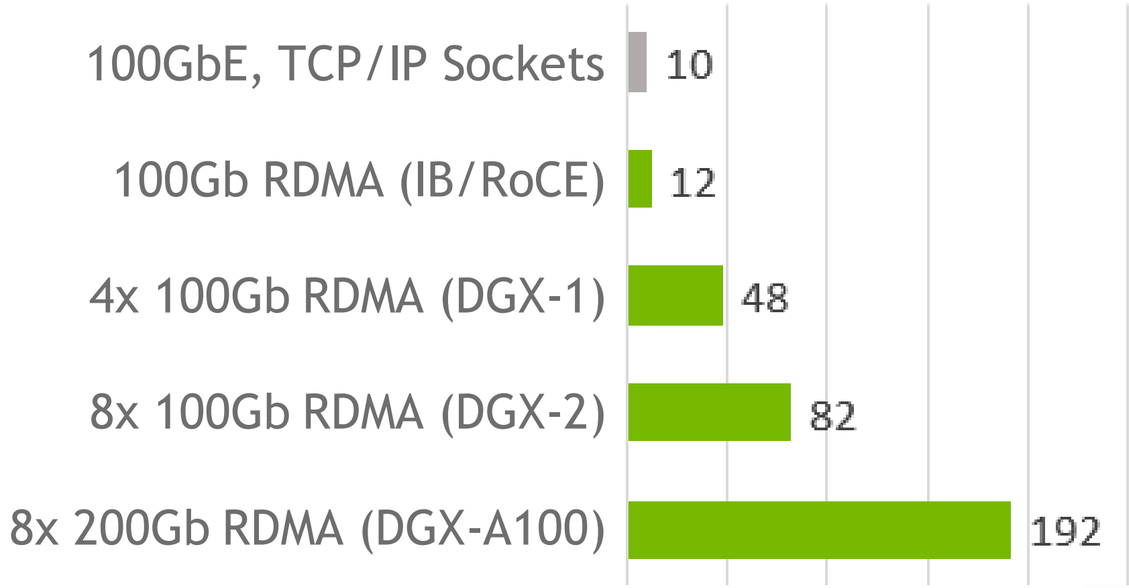
Binaries : https://developer.nvidia.com/nccl and in NGC containers
Source code : https://github.com/nvidia/nccl
Perf tests : https://github.com/nvidia/nccl-tests

INTER-GPU COMMUNICATION

Intra-node and Inter-node



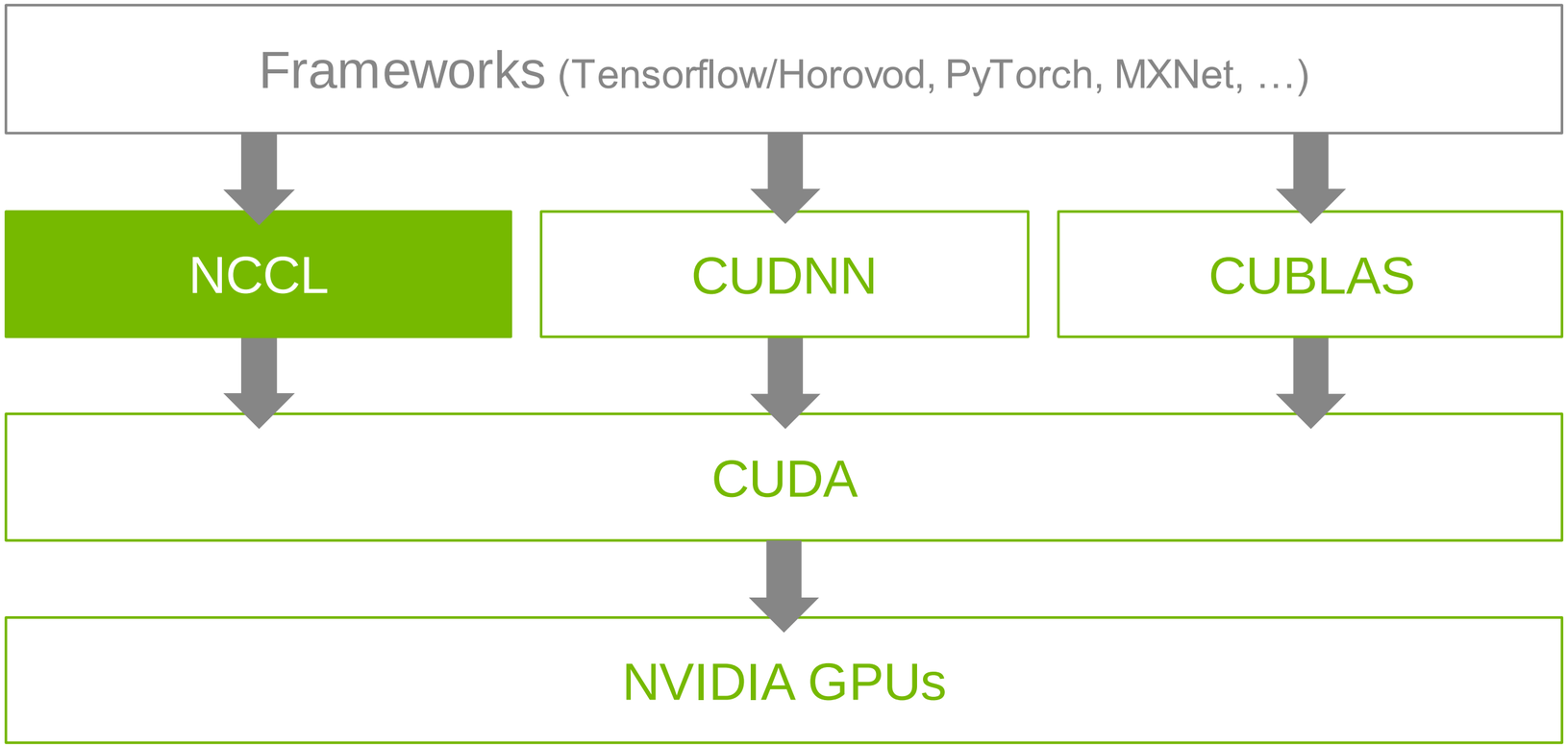




NVIDIA

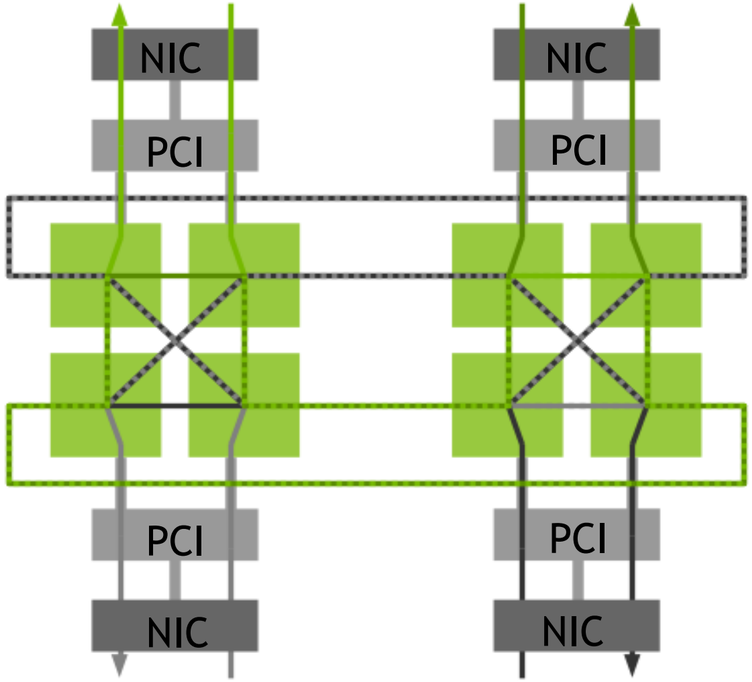
NCCL

DL stack

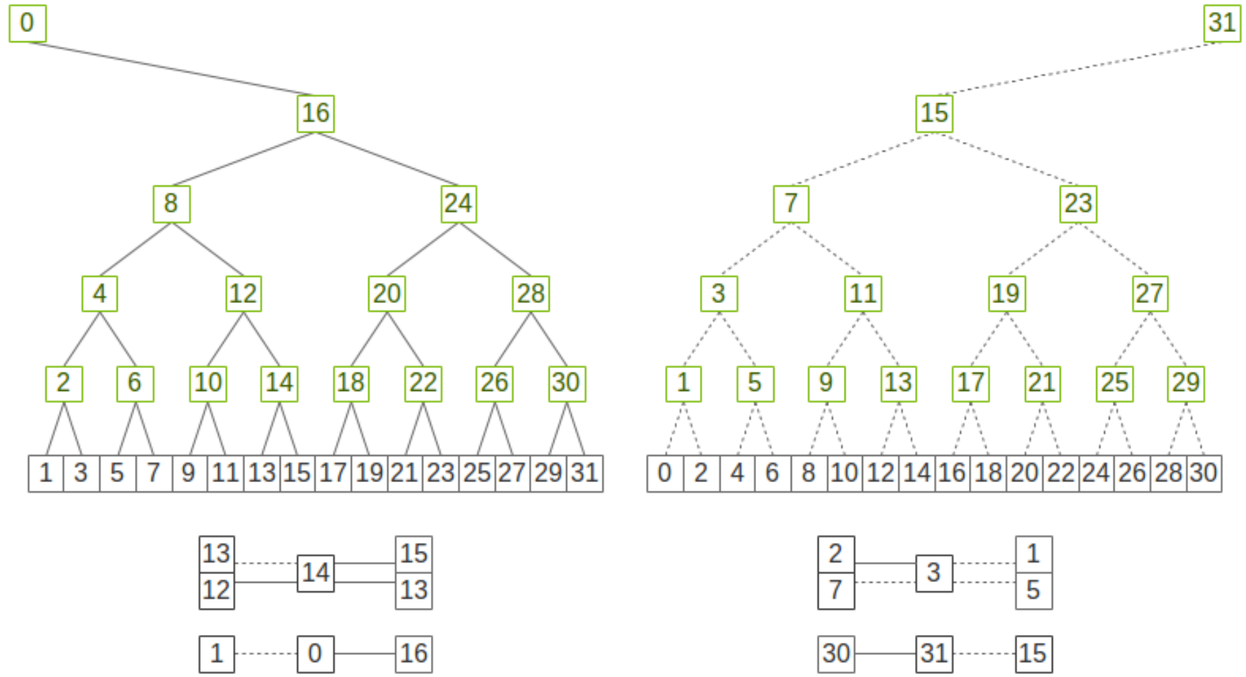


NCCL Communication Patterns

Map to Rings and Trees



Adapt to most topologies
Full bandwidth
Linear latency



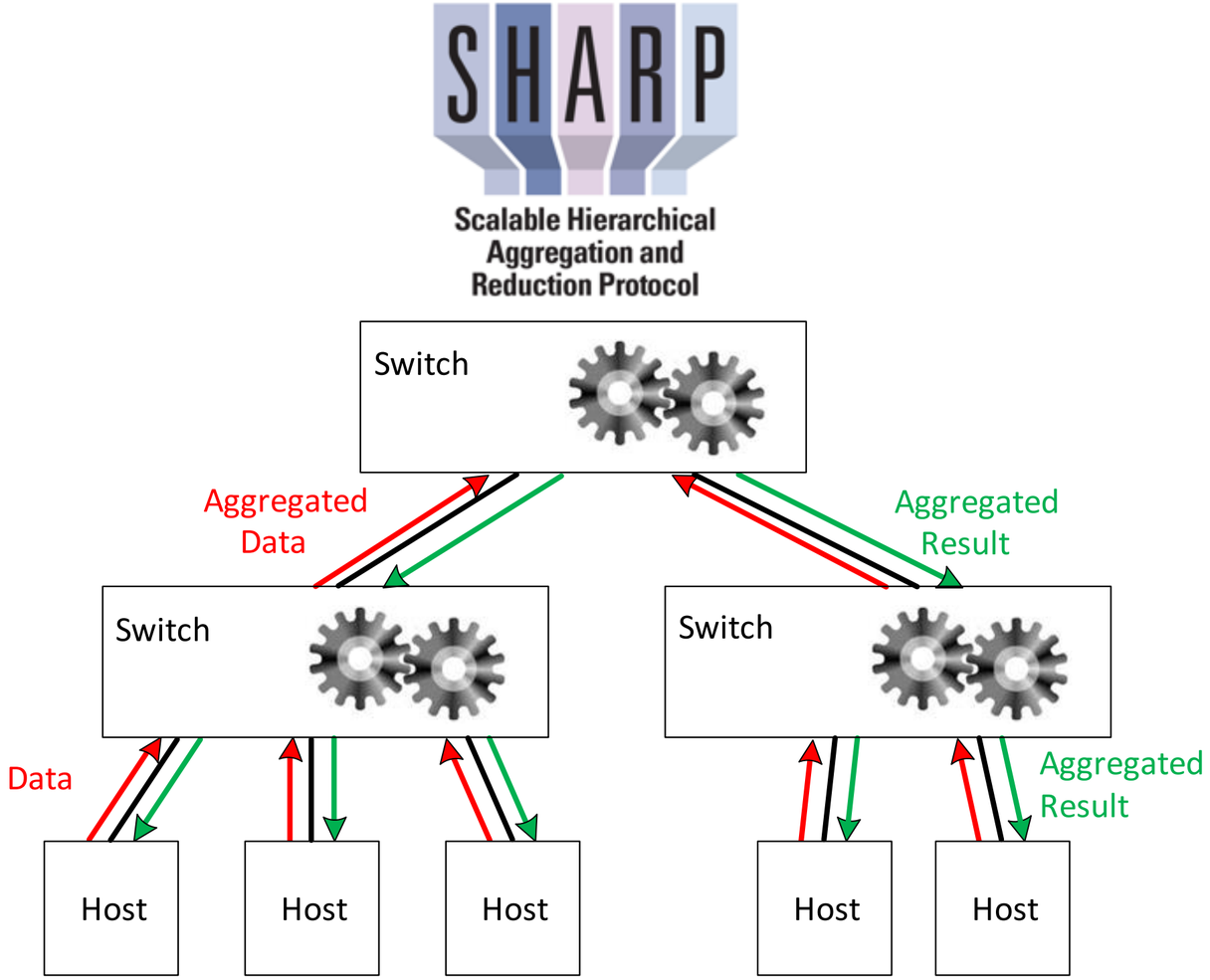
Use double binary tree for inter-node

SCALABLE HIERARCHICAL AGGREGATION AND REDUCTION PROTOCOL (SHARP)

Reliable Scalable General Purpose Primitive
- In-network Tree based aggregation mechanism
- Large number of groups
- Multiple simultaneous outstanding operations
- Accelerating the world's fastest supercomputers (Summit, Sierra)
- Support for small and large vectors (HDR/Quantum)

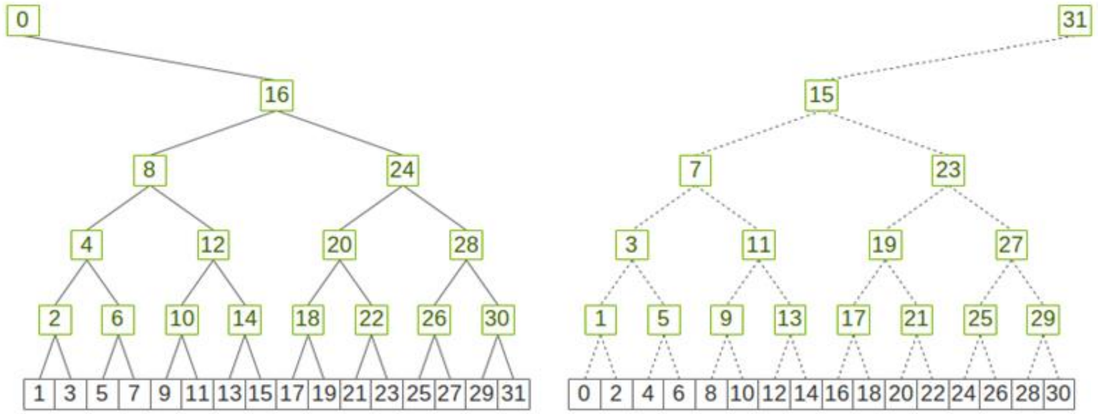
Applicable to Multiple Use-cases
- HPC Applications using MPI / SHMEM
- Distributed Machine Learning applications

Scalable High Performance Collective Offload
- Barrier, Reduce, All-Reduce, Broadcast and more
- Sum, Min, Max, Min-loc, max-loc, OR, XOR, AND
- Integer and Floating-Point, 16/32/64 bits

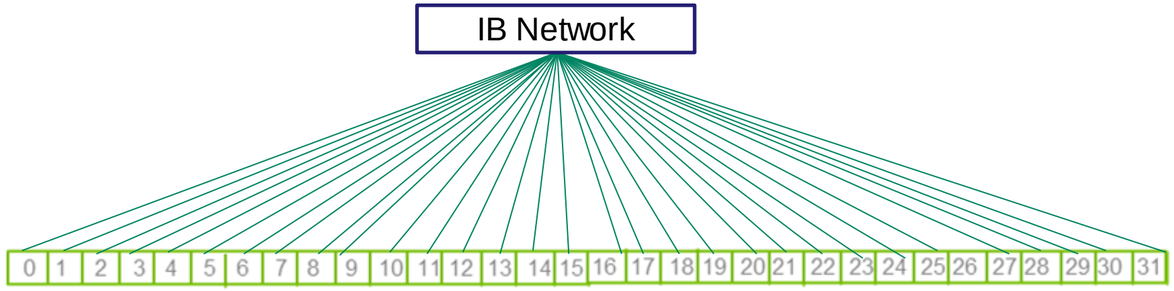


DOUBLE BW + LOWER LATENCY

Switch-based Reductions with SHARP

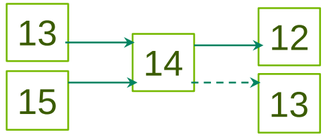


Traditional Tree Reduction

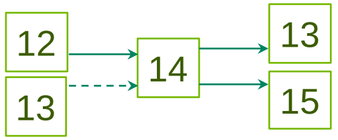


SHARP Reduction

reduce_scatter



broadcast



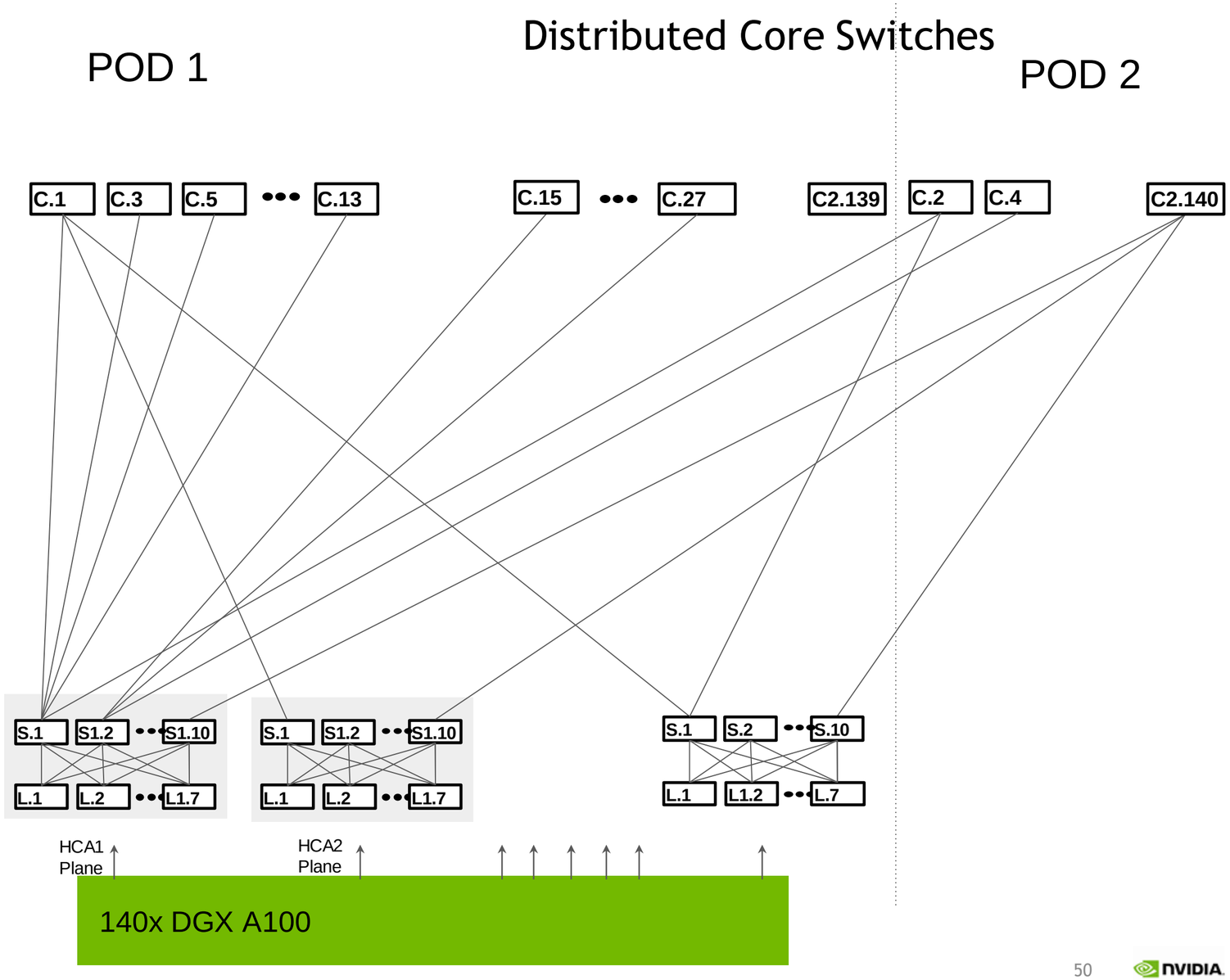
allreduce

Sends data once, receives final result

- No intermediate results: effectively doubles bandwidth
- Fewer hops (1 per switch level): lower latency
- Offload GPU computation
- Less interference with other tenants

A fabric designed for AI and HPC

Multi-GPU Training



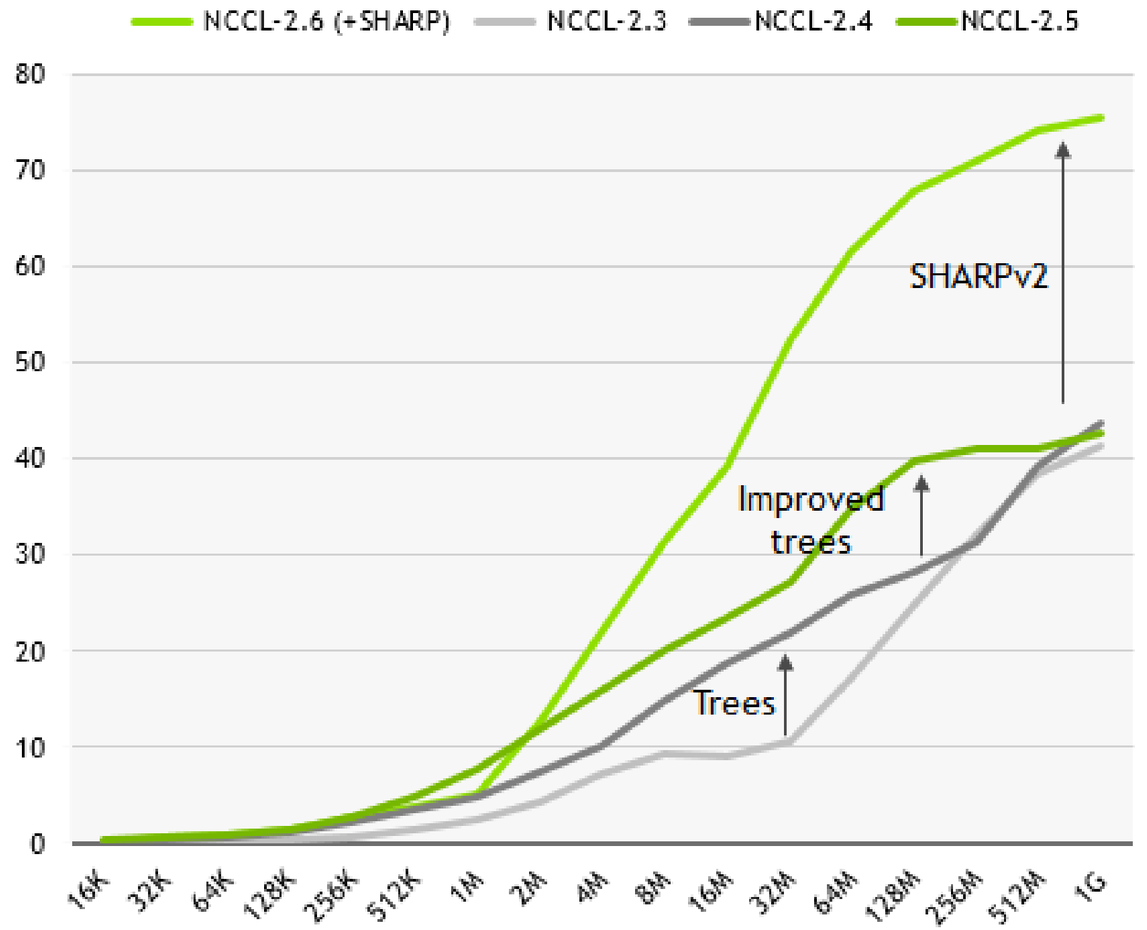
PERFORMANCE

Getting peak BW, at scale

Constantly improving performance

Trees (2.4)
Split trees (2.5)
Add support for adaptive routing (2.6)
In-network all-reduce operations (2.6)

Allow for large scale training, minimize communication time.

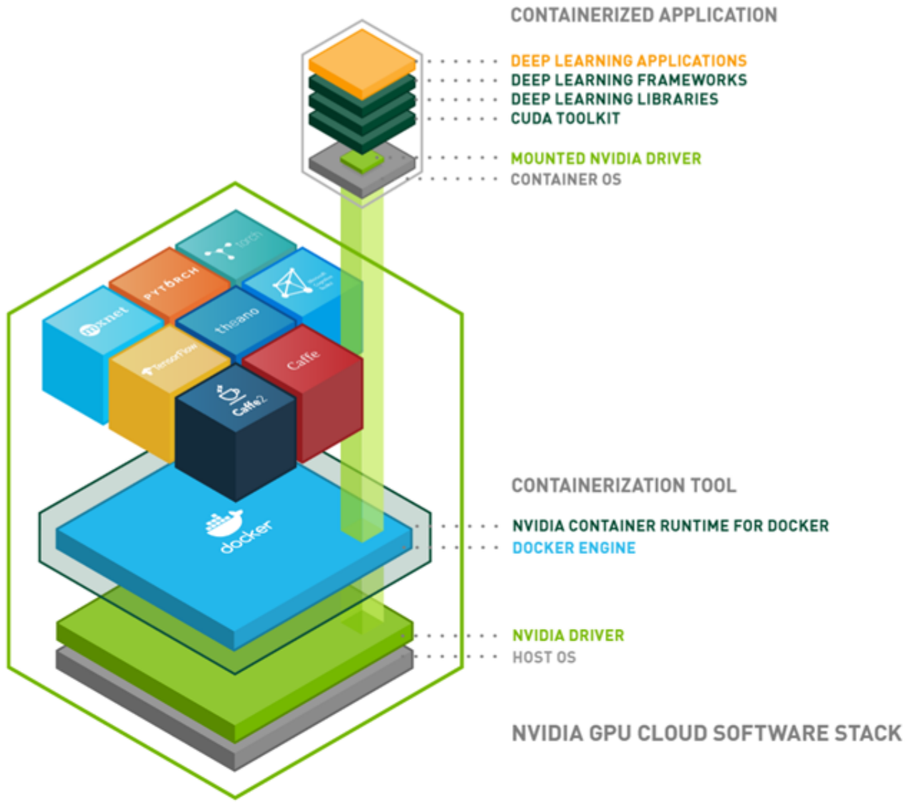


32xDGX1V + 4xMellanox CX-6, NCCL AllReduce BusBW in GB/s

SCALE TO MULTIPLE NODES

Software stack - Application

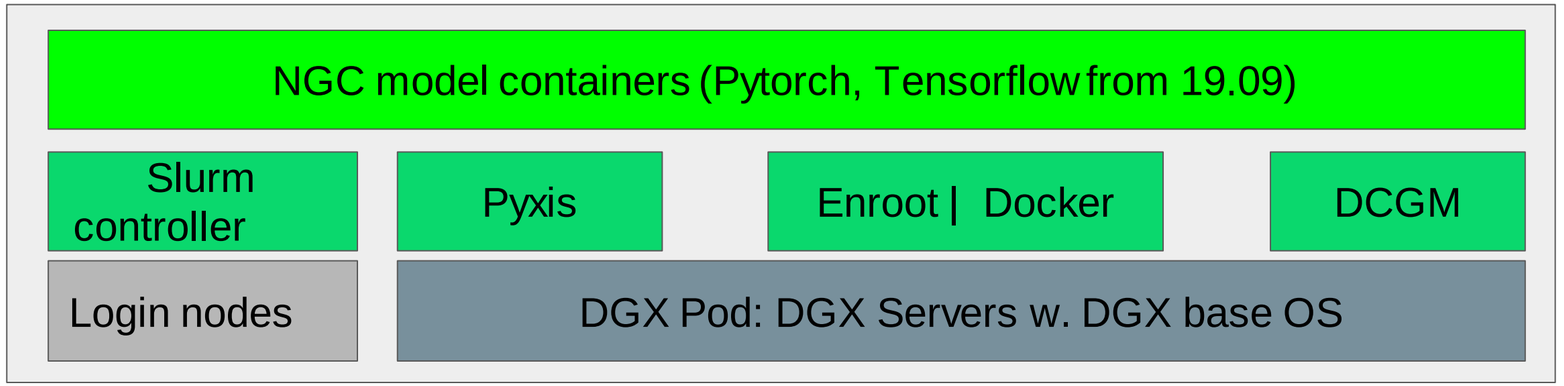
- Deep Learning Model:
  - Hyperparameters tuned for multi-node scaling
  - Multi-node launcher scripts
- Deep Learning Container:
  - Optimized TensorFlow, GPU libraries, and multi-node software
- Host:
  - Host OS, GPU driver, IB driver, container runtime engine (docker, enroot)



SCALE TO MULTIPLE NODES

Software stack - System

- Slurm: User job scheduling & management
- Enroot: NVIDIA open-source tool to convert traditional container/OS images into unprivileged sandboxes
- Pyxis: NVIDIA open-source plugin integrating Enroot with Slurm



- DeepOps: NVIDIA open-source toolbox for GPU cluster management w/Ansible playbooks

EXAMPLE

SLURM+Docker+MPI is hard

Excerpts from an actual script used to launch jobs for the MLPerf v0.5 benchmark (208 LOC total)

1. Setup docker flags
2. Setup mpirun flags
3. Setup SSH
4. Start sleep containers
5. Launch mpirun in rank0 container

```
#!/bin/bash

## Docker params
export VOLS="-v $DATADIR:/data -v $LOGDIR:/results"
export CONTNAME="mpi_${SLURM_JOB_ID}"
export DOCKEREXEC="nvidia-docker run --rm --net=host --uts=host --ipc=host --ulimit stack=67108864 --ulimit
memlock=-1 --security-opt seccomp=unconfined  $IBDEVICES"

MPICMD="mpirun --allow-run-as-root --tag-output --bind-to none -x SLURM_NTASKS_PER_NODE=$SLURM_NTASKS_PER_NODE -x
GPUS=$GPUS -x BATCHSIZE=$BATCHSIZE -x KVSTORE=$KVSTORE -x LR=$LR -x WARMUP_EPOCHS=$WARMUP_EPOCHS -x
EVAL_OFFSET=$EVAL_OFFSET -x DGXSYSTEM=$DGXSYSTEM ./run_and_time.sh"

MASTER_IP=`getent hosts \`hostname\` | cut -d ' ' -f1`
export hosts=( `scontrol show hostname |tr "\n" " "` )
unique_hosts=( $(echo "${hosts[@]}" | tr ' ' '\n' | sort -u | tr '\n' ' ' ) )
export MASTER_HOST=${hosts[0]}

VARS="-e OMPI_MCA_mca_base_param_files=/dev/shm/mpi/${SLURM_JOB_ID}/mca_params.conf -e GPUS -e BATCHSIZE -e KVSTORE
-e LR -e WARMUP_EPOCHS -e EVAL_OFFSET -e CONT -e DGXSYSTEM=$DGXSYSTEM -e MASTER_HOST -e MASTER_IP -e
SLURM_JOB_NUM_NODES -e SLURM_NNODES -e SLURM_NTASKS_PER_NODE "

docker pull $CONT

mkdir -p ${HOME}/.ssh/sbatch/${SLURM_JOB_ID}
ssh-keygen -t rsa -b 2048 -n "" -f "${HOME}/.ssh/sbatch/${SLURM_JOB_ID}/sshkey.rsa" -C "mxnet_${SLURM_JOB_ID}_"
&>/dev/null
echo command=\"/dev/shm/mpi/${SLURM_JOB_ID}/sshentry.sh\",no-port-forwarding,no-agent-forwarding,no-X11-forwarding
$(cat ${HOME}/.ssh/sbatch/${SLURM_JOB_ID}/sshkey.rsa.pub) >> ${HOME}/.ssh/authorized_keys
chmod 600 ~/.ssh/authorized_keys
srun  -n $SLURM_JOB_NUM_NODES --ntasks-per-node=1 bash -c "mkdir -p /dev/shm/mpi/${SLURM_JOB_ID}; cp -R ${HOME}/.ssh
/sbatch/${SLURM_JOB_ID} /dev/shm/mpi; chmod 700 /dev/shm/mpi/${SLURM_JOB_ID}"
sleep 2

# Create mpi config file
srun  -n $SLURM_JOB_NUM_NODES --ntasks-per-node=1 tee /dev/shm/mpi/${SLURM_JOB_ID}/mca_params.conf <<-EOF
    plm_rsh_agent = /usr/bin/ssh
    plm_rsh_args = -i /dev/shm/mpi/${SLURM_JOB_ID}/sshkey.rsa -oStrictHostKeyChecking=no
-oUserKnownHostsFile=/dev/null -oLogLevel=ERROR -l ${USER}
    orte_default_hostfile = /dev/shm/mpi/${SLURM_JOB_ID}/mpi_hosts
    btl_openib_warn_default_gid_prefix = 0
    mpi_warn_on_fork = 0
    allow_run_as_root = 1
EOF

# Create ssh helper script that transfers an ssh into a compute node into the running container on that node
srun -n $SLURM_JOB_NUM_NODES --ntasks-per-node=1 tee /dev/shm/mpi/${SLURM_JOB_ID}/sshentry.sh <<-EOF
    #!/bin/bash
    echo "::sshentry: entered \$(hostname)"
    [[ -f $CONTNAME ]] && "::worker container not found error" && exit 1
    echo "::sshentry: running \$SSH_ORIGINAL_COMMAND"
    exec docker exec $CONTNAME /bin/bash -c "\$SSH_ORIGINAL_COMMAND"
EOF

# Create mpi hostlist
for h in ${hosts[@]}; do
   echo "$h slots=${SLURM_NTASKS_PER_NODE}" >> /dev/shm/mpi/${SLURM_JOB_ID}/mpi_hosts
done
srun -n $SLURM_JOB_NUM_NODES --ntasks-per-node=1 bash -c "cp $(which ssh) /dev/shm/mpi/${SLURM_JOB_ID}/.;  chmod 755
/dev/shm/mpi/${SLURM_JOB_ID}/mca_params.conf;  chmod 755 /dev/shm/mpi/${SLURM_JOB_ID}/sshentry.sh"

# Launch containers behind srun
srun  -n $SLURM_JOB_NUM_NODES --ntasks-per-node=1 $DOCKEREXEC --name $CONTNAME $VOLS $VARS $CONT bash -c 'sleep
infinity' & rv=$?
sleep 30

# Launching app
$(eval echo $SSH) docker exec $VARS $CONTNAME $MPICMD

# Clean up
docker rm -f $CONTNAME
```

CONTAINERS AND SUPERPODS

What do we need?

What we **need**

- High performance
- Unprivileged runtime
- Uses docker image format

What we **want**

- Preserve SLURM cgroups
- NVIDIA+Mellanox devices are available by default
- MPI and UCX between containers is easy
- Can install packages inside containers

ENROOT

Summary

Fully unprivileged “chroot” (with optional root-remapping)

Standalone (no daemon, no extra process)

Simple and easy to use (UNIX philosophy, KISS principle)

Little isolation, no overhead

Docker image support (5x pull speedup, shared cache)

Simple image format (single file + UNIX configs)

Composable and extensible (system/user configs, lifecycle hooks)

Advanced features (runfiles, scriptable configs, in-memory containers)

http://github.com/nvidia/enroot

ENROOT

Basic usage

```
$ enroot import docker://nvcr.io#nvidia/tensorflow:19.08-py3
$ ls nvidia+tensorflow+19.08-py3.sqsh

$ enroot create --name tensorflow nvidia+tensorflow+19.08-py3.sqsh
$ ls -d ${XDG_DATA_PATH}/enroot/tensorflow

$ enroot start tensorflow nvidia-smi -L

$ enroot start --root --rw tensorflow apt update && apt install …

$ enroot bundle --output tensorflow.run nvidia+tensorflow+19.05-py3.sqsh
$ ./tensorflow.run python -c 'import tensorflow as tf; print(tf.__version__)'
```

ENROOT

Improved Linux utils

enroot-unshare : like unshare(1), creates new namespaces

enroot-mount : like mount(8), mounts filesystems

enroot-switchroot : like switch_root(8), changes rootfs

enroot-aufs2ovlfs : converts AUFS whiteouts to OverlayFS

enroot-mksquashovlfs : like mksquashfs(1) on top of OverlayFS

```
$ curl https://cdimage.ubuntu.com/[...]/ubuntu-base-16.04-core-amd64.tar.gz | tar -C ubuntu -xz

$ enroot-unshare bash

$ cat << EOF | enroot-mount --root ubuntu -
  ubuntu        /          none bind,rprivate
  /proc         /proc      none rbind
  /dev          /dev       none rbind
  /sys          /sys       none rbind
EOF

$ exec enroot-switchroot ubuntu bash
```

Pyxis

Slurm spank plugin for Enroot

```
# run a command on a worker node
$ srun grep PRETTY /etc/os-release
PRETTY_NAME="Ubuntu 18.04.2 LTS"

# run the same command, but now inside of a container
$ srun --container-image=centos grep PRETTY /etc/os-release
PRETTY_NAME="CentOS Linux 7 (Core)"

# run inside the container, but mount the file from the host
$ srun --container-image=centos \
    --container-mounts=/etc/os-release:/etc/os-release \
    grep PRETTY /etc/os-release
PRETTY_NAME="Ubuntu 18.04.2 LTS"
```

http://github.com/nvidia/pyxis

Pyxis

Internals

1. slurm_spank_init()
   a. Add flags to srun

2. slurm_spank_user_init() - runs for each JOBSTEP
   a. Download a container image from a registry *(enroot import)*
   b. Unpack the image to a new container rootfs *(enroot create)*
   c. Start up a new “container” process *(enroot start)*
   d. Copy environment variables
   e. Save namespaces for later

3. slurm_spank_task_init() - runs for each TASK
   a. `setns(CLONE_NEWUSER) # join user namespace`
   b. `setns(CLONE_NEWNS)   # join mounts namespace`
   c. `chdir()`
   d. Setup PMIx, if active

Examples

Pyxis, MPI workload

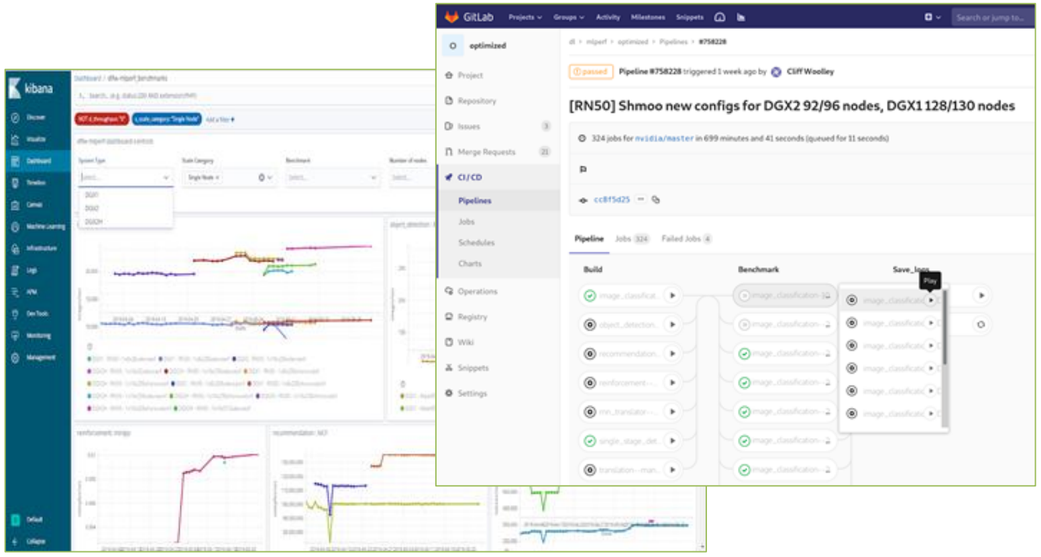
```
srun -N4 --ntasks-per-node=1 --mpi=pmix \
   --container-image "${docker_image}" \
   --container-mounts "/raid/datasets/imagenet:/data,/scratch:/scratch" \
   caffe train --solver "/scratch/snikolaev/rn50/solver_idl_4k_mpi.prototxt" --gpu=all
```

1. No need to pass through environment variables (Pyxis inherits them all)
2. No need for any of these docker args: `--rm --net=host --uts=host --ipc=host --pid=host`
3. No need to configure mpirun (SLURM handles it)
4. No need to setup SSH (PMIx doesn't use it)

Integrating clusters in the development workflow

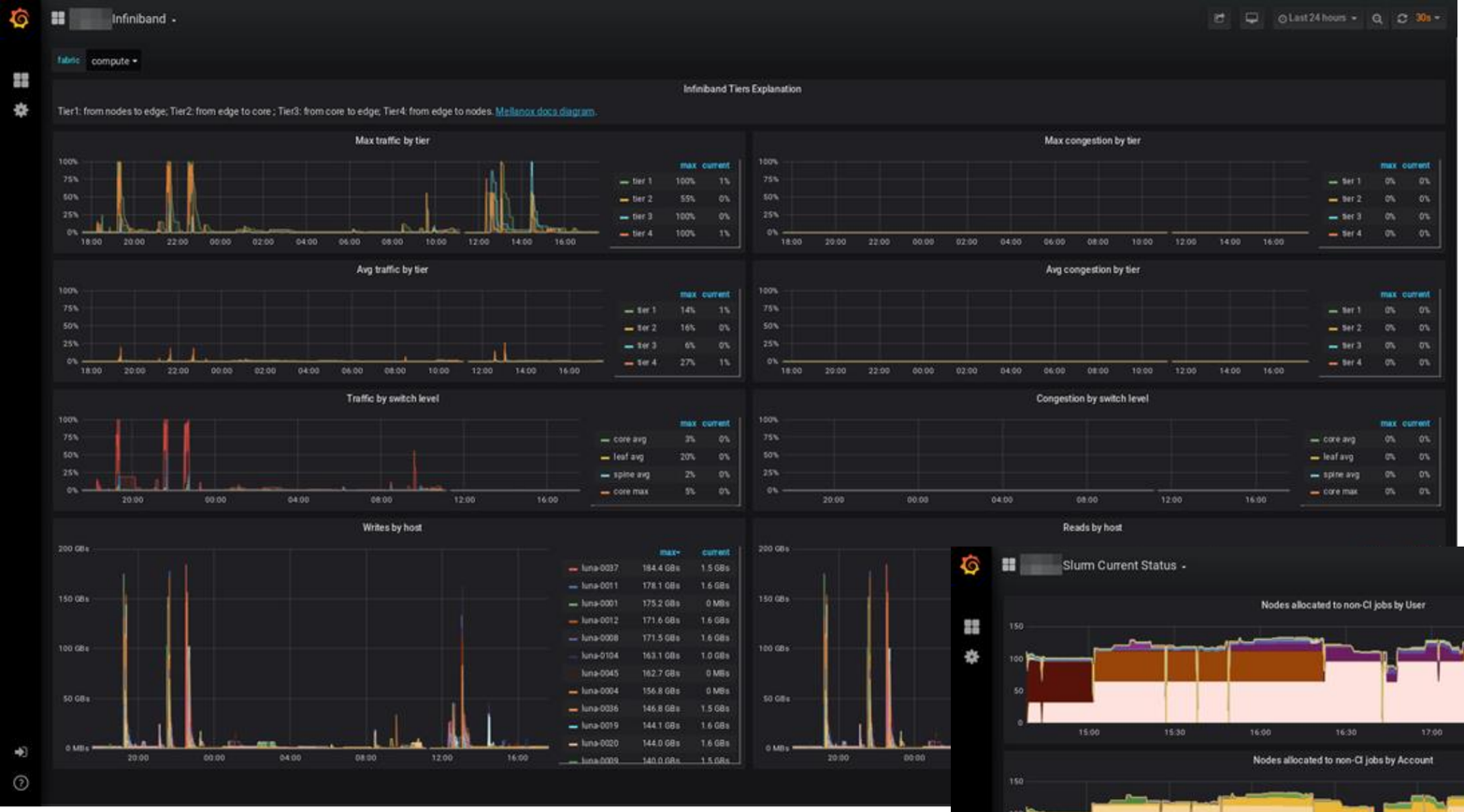
Supercomputer-scale CI (Continuous integration internal at NVIDIA)

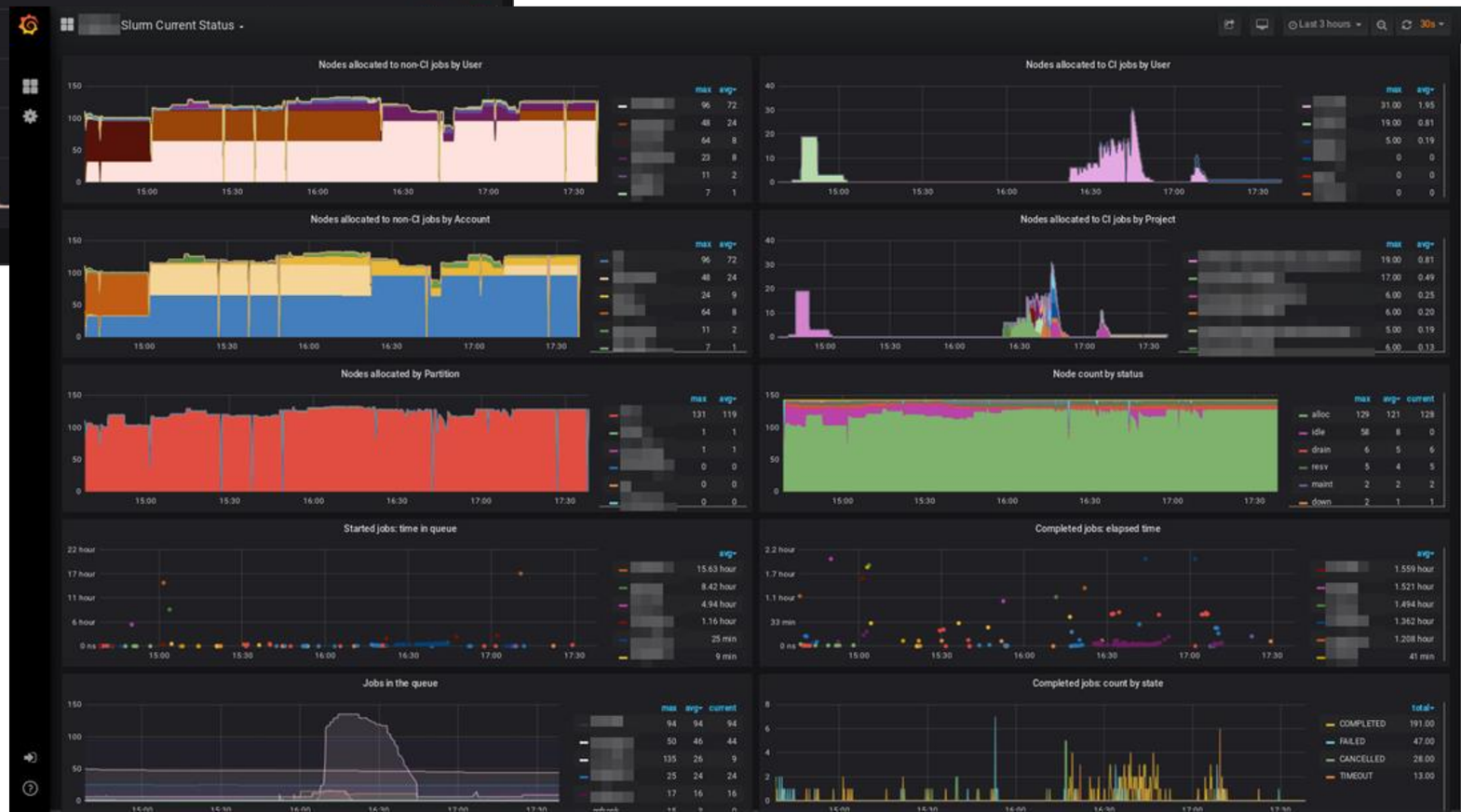
- Integrating DL-friendly tools like GitLab, Docker w/ HPC systems
  - Kick off 10000's of GPU hours of tests with a single button click in GitLab
    - … build and package with Docker
    - … schedule and prioritize with SLURM
    - … on demand or on a schedule
    - … reporting via GitLab, ELK stack, Slack, email
  - Emphasis on keeping things simple for users while hiding integration complexity
  - Ensure reproducibility and rapid triage

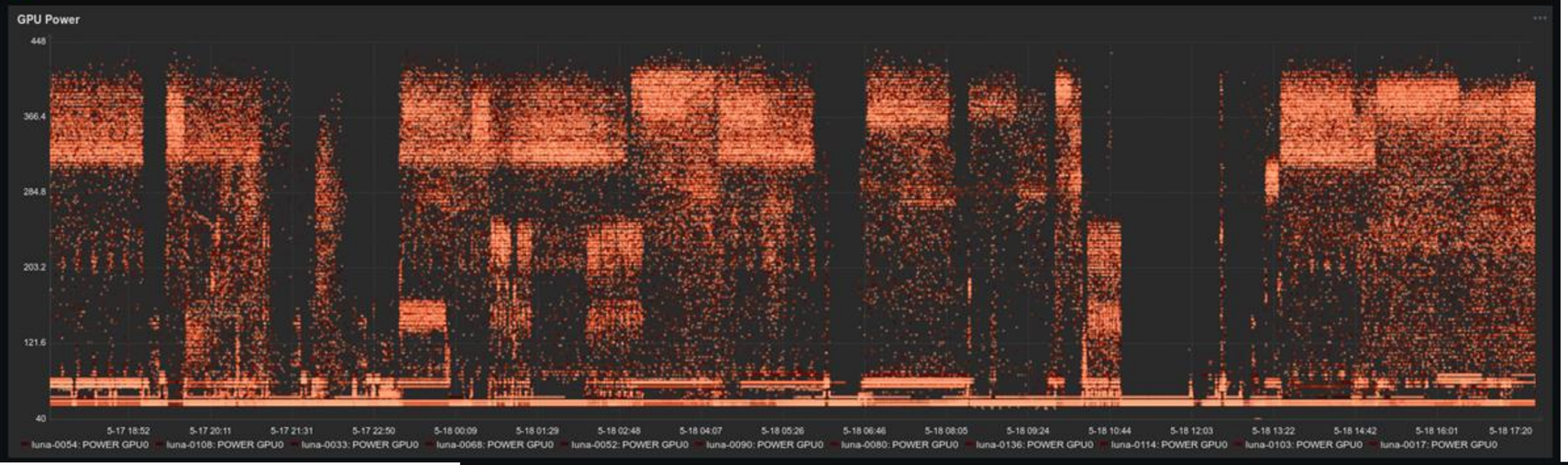


MONITORING

Infiniband, Power, Nodes, etc

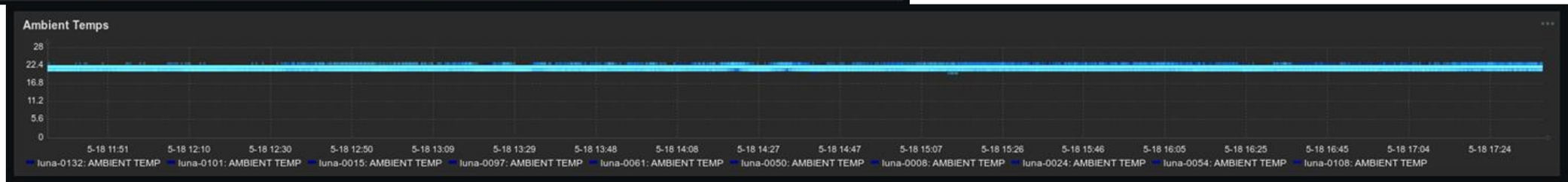






MONITORING

Power, Thermals



Links

Resources

GTC Webinars

JHH Keynote

Under the Hood of the new DGX A100 System Architecture [S21884]

Inside the NVIDIA Ampere Architecture [S21730]

CUDA New Features And Beyond [S21760]

Inside the NVIDIA HPC SDK: the Compilers, Libraries and Tools for Accelerated Computing [S21766]

Introducing NVIDIA DGX A100: the Universal AI System for Enterprise [S21702]

Mixed-Precision Training of Neural Networks [S22082]

Tensor Core Performance on NVIDIA GPUs: The Ultimate Guide [S21929]

Developing CUDA kernels to push Tensor Cores to the Abolute Limit on NVIDIA A100 [S21745]

Catalog available at https://www.nvidia.com/en-us/gtc/session-catalog/

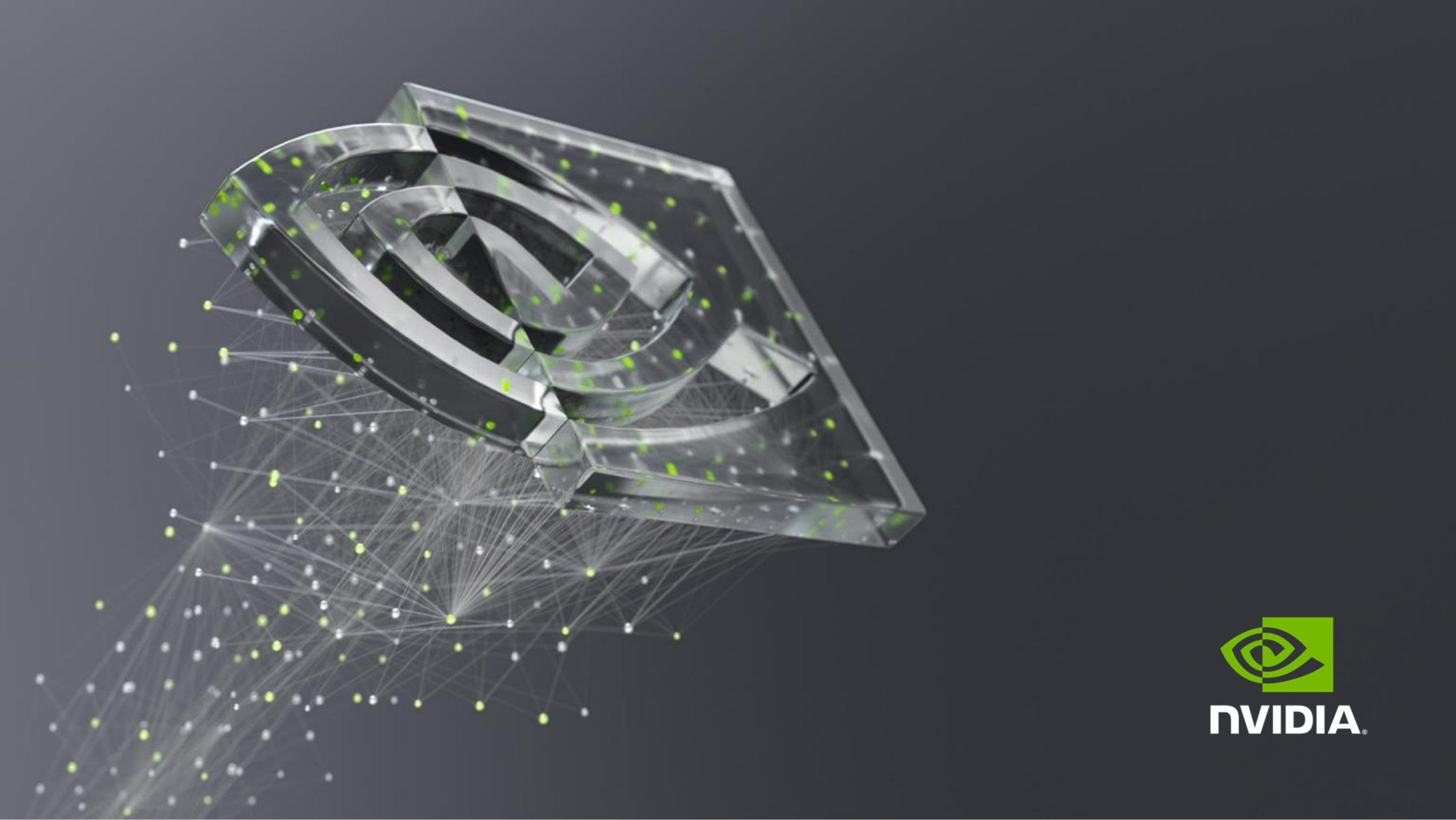
Resources

Links and other doc

DGX A100 Page https://www.nvidia.com/en-us/data-center/dgx-a100/

Blogs

- DGX A100 SuperPOD https://blogs.nvidia.com/blog/2020/05/14/dgx-superpod-a100/
- DDN Blog for DGX A100 Storage https://www.ddn.com/press-releases/ddn-a3i-nvidia-dgx-a100/
- Kitchen Keynote summary https://blogs.nvidia.com/blog/2020/05/14/gtc-2020-keynote/
- Double Precision Tensor Cores https://blogs.nvidia.com/blog/2020/05/14/double-precision-tensor-cores/


NVIDIA