www.locuz.com

DGX-1 User Training

Presented By: Mandeep Kumar

NVIDIA

LOCUZ Converge to the Cloud

Agenda

- DGX-1 Overview
- NVIDIA GPU Cloud (NGC)
- Virtual Machine vs. Container
- What’s Docker?
- Why NVIDIA Docker?
- NVIDIA Docker Sub-Commands
- Running Docker Containers
- Docker on HPC Systems
- Singularity: A Container Engine for HPC
- Running with Singularity
- SLURM Overview and Architecture
- SLURM Commands for the User
- Containers (Singularity) with SLURM Sample Script

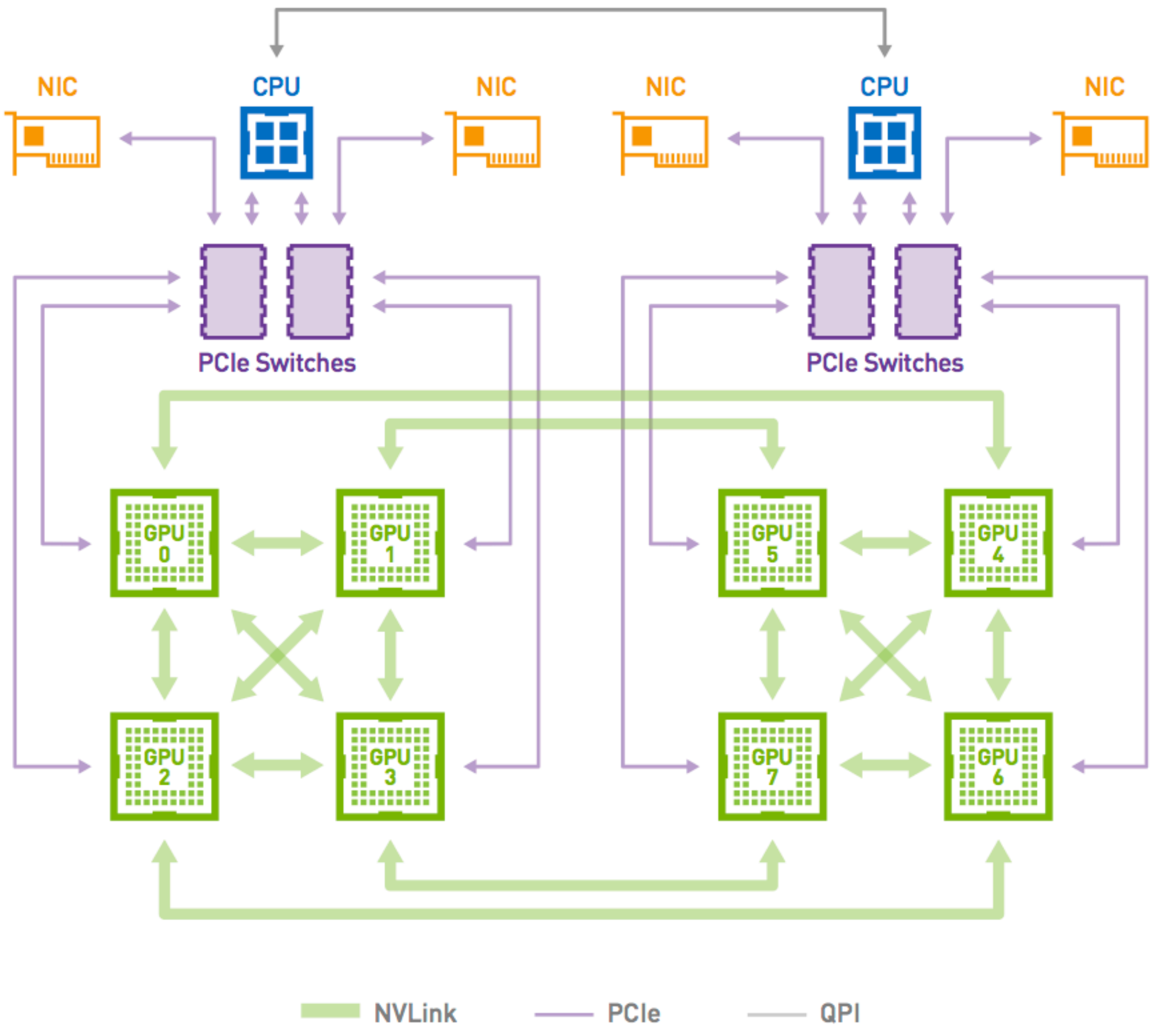
NVIDIA DGX-1

DGX-1 Overview

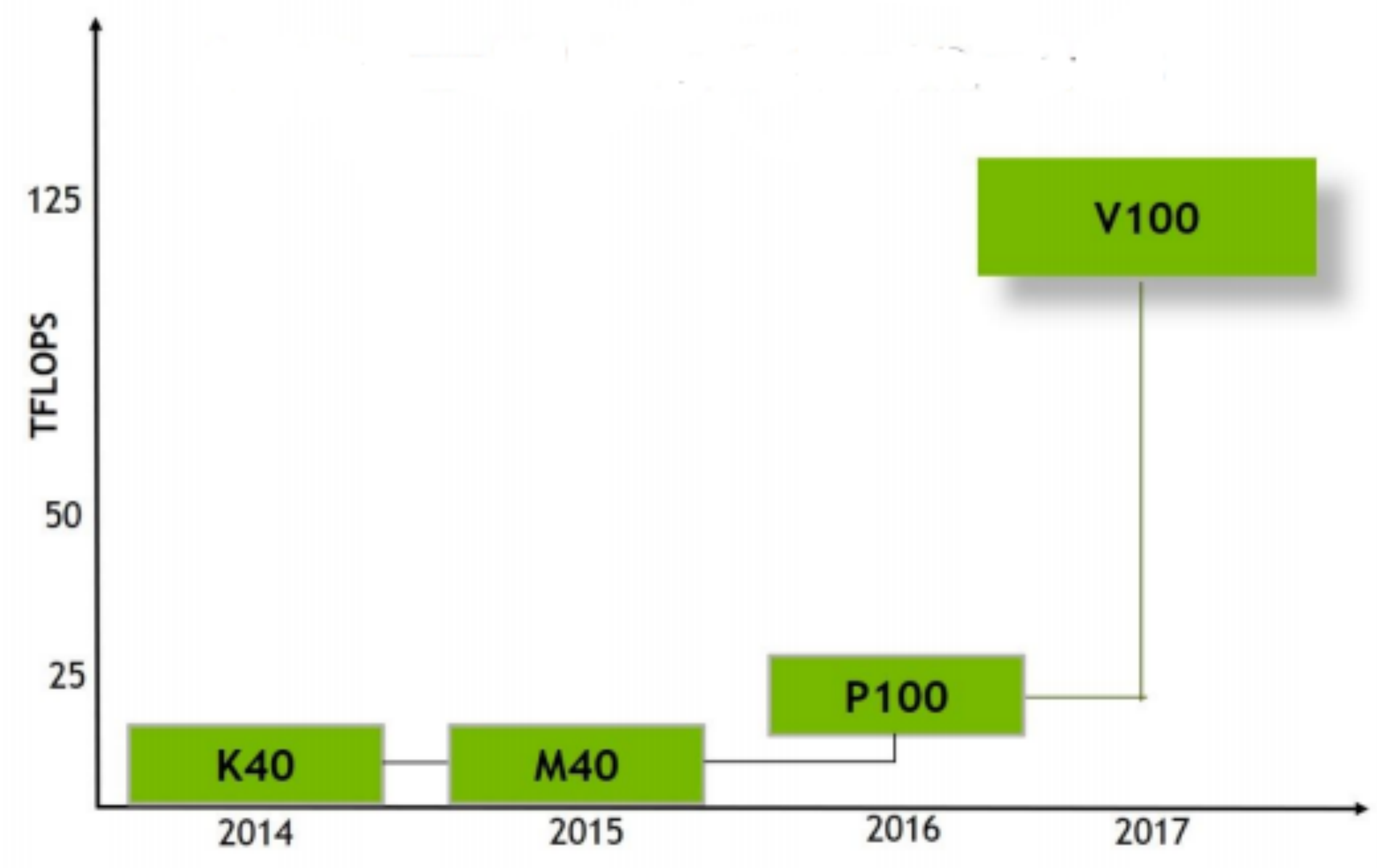
System Specifications

GPUs	8xTesla V100
GPU Memory	256 GB (32 GB/GPU)
CPU	Dual 20-core Intel Xeon E5-2698 v4 2.2 GHz
NVIDIA CUDA Cores	40,960
NVIDIA Tensor Cores (on V100 based systems)	5,120
System Memory	512 GB 2,133 MHz DDR4 LRDIMM
Storage	Data: 4x1.92 TB SSD RAID-0
Network	Dual 10 GbE, 4 IB EDR

Hybrid Cube Mesh Architecture



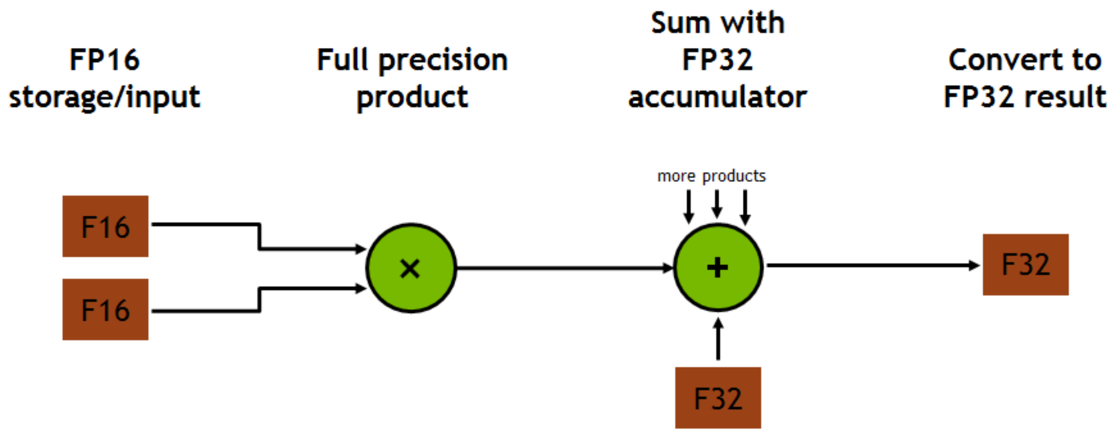
PERFORMANCE FOR AI AND HPC



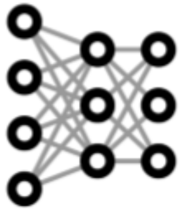
Tensor Cores

$$D = \begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix} \begin{pmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{pmatrix} + \begin{pmatrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{pmatrix}$$

FP16 or FP32 | FP16 | FP16 | FP16 or FP32



FASTEST PATH TO DEEP LEARNING



Fully-integrated and pre-optimized Insights in hours instead of weeks

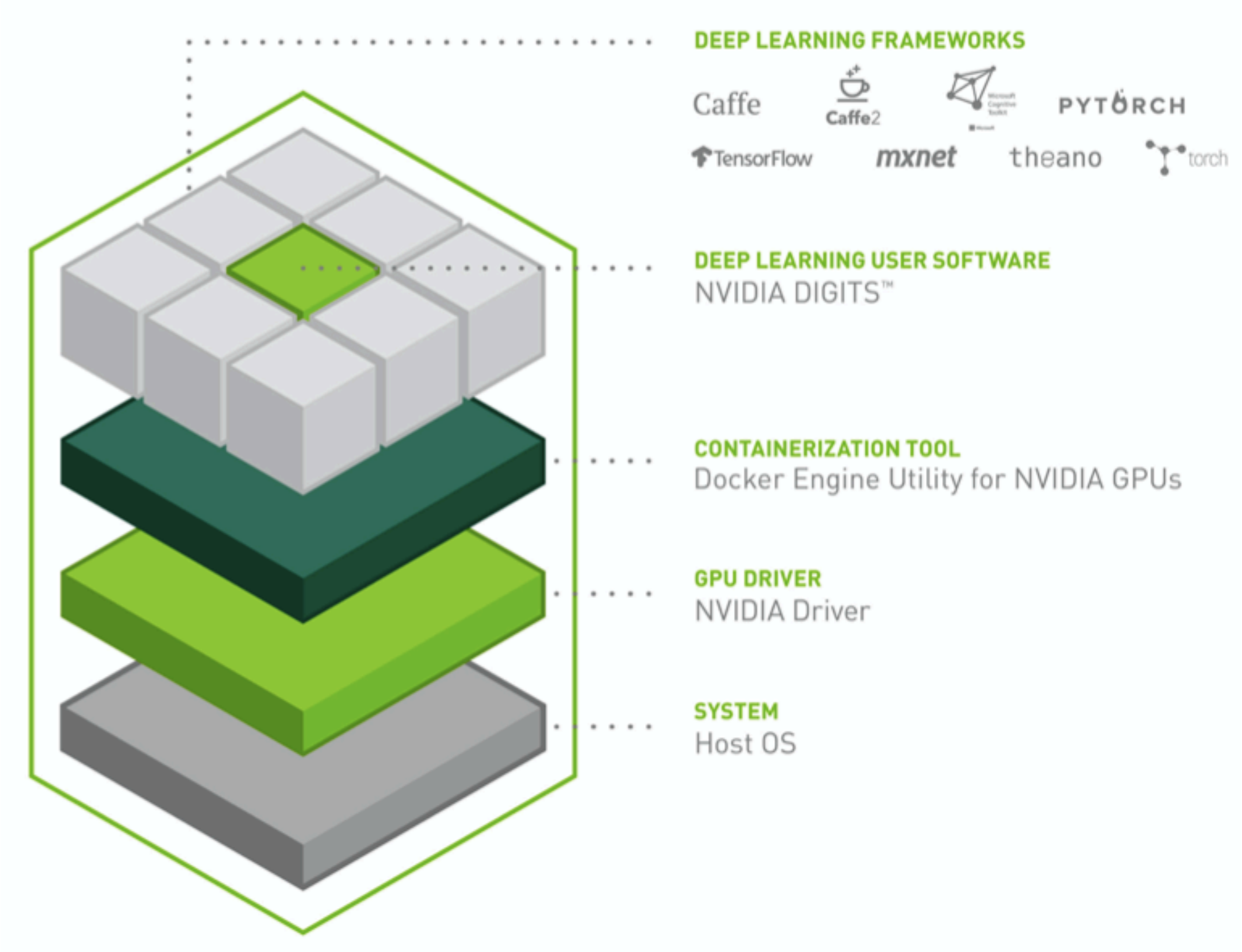
EFFORTLESS PRODUCTIVITY

Optimized frameworks and cloud managed for faster insights

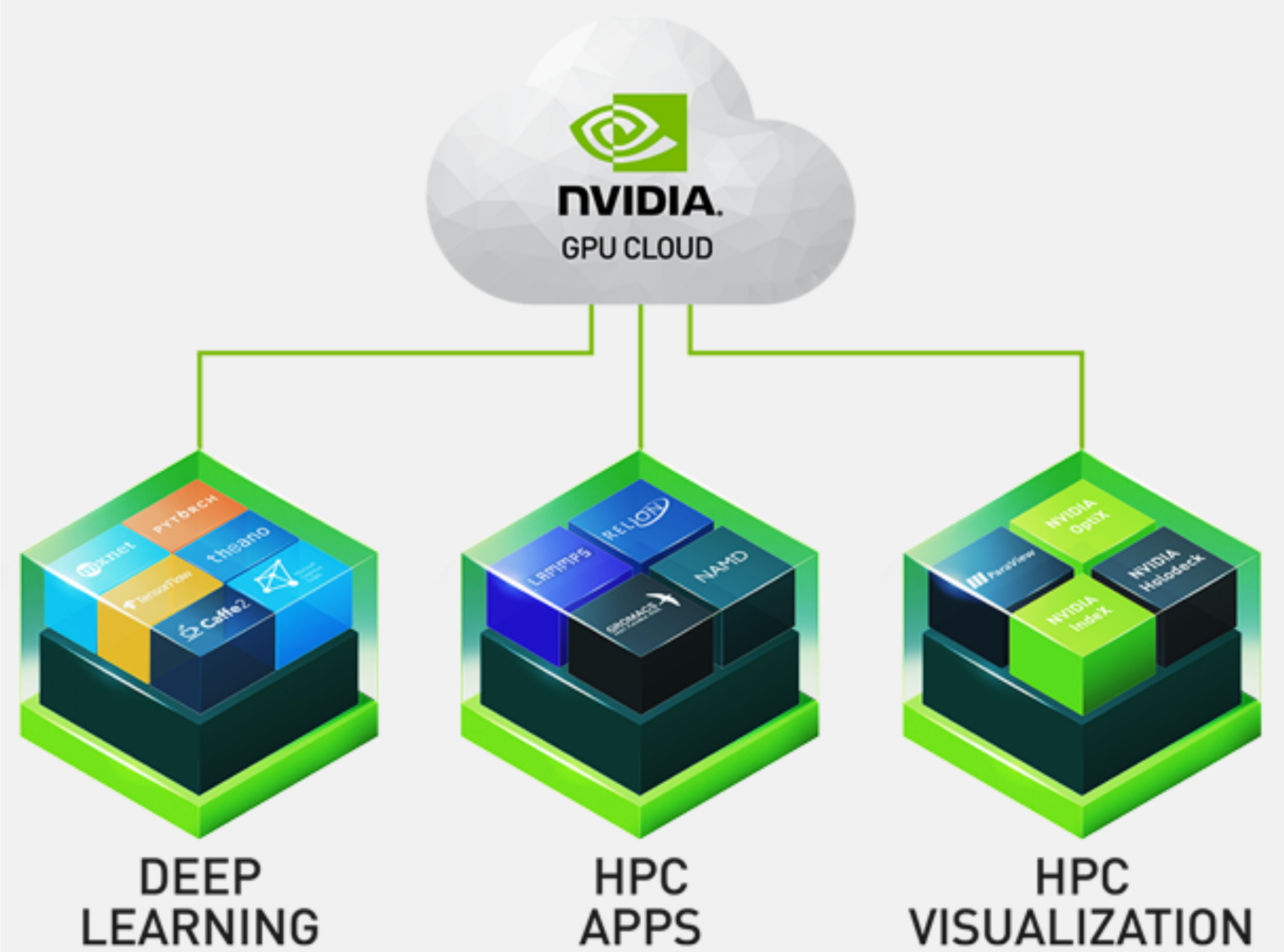
REVOLUTIONARY AI PERFORMANCE

DGX software stack for fastest GPU performance in the industry

The DGX-1 Deep Learning Software Stack

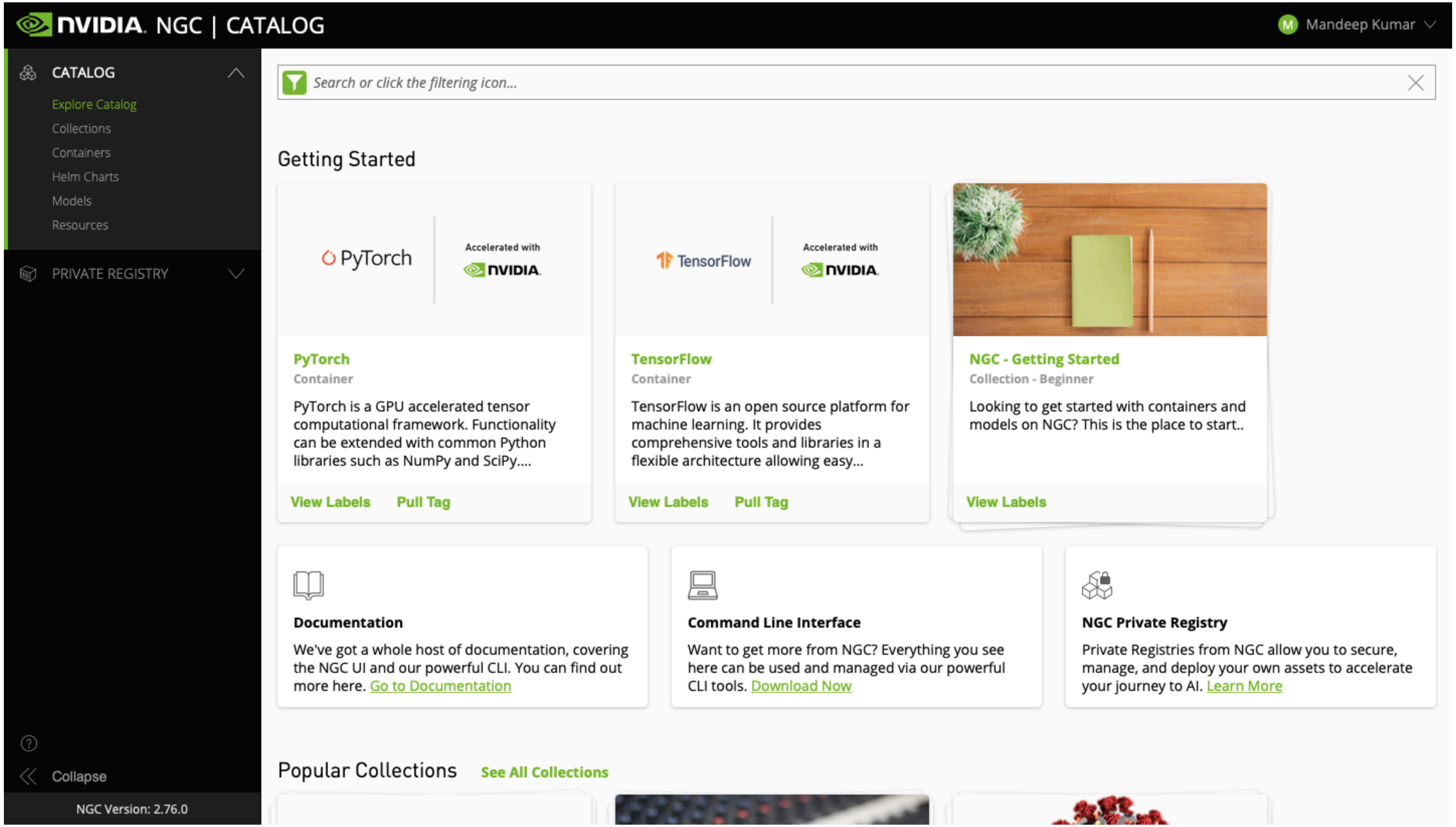


NVIDIA GPU Cloud (NGC)



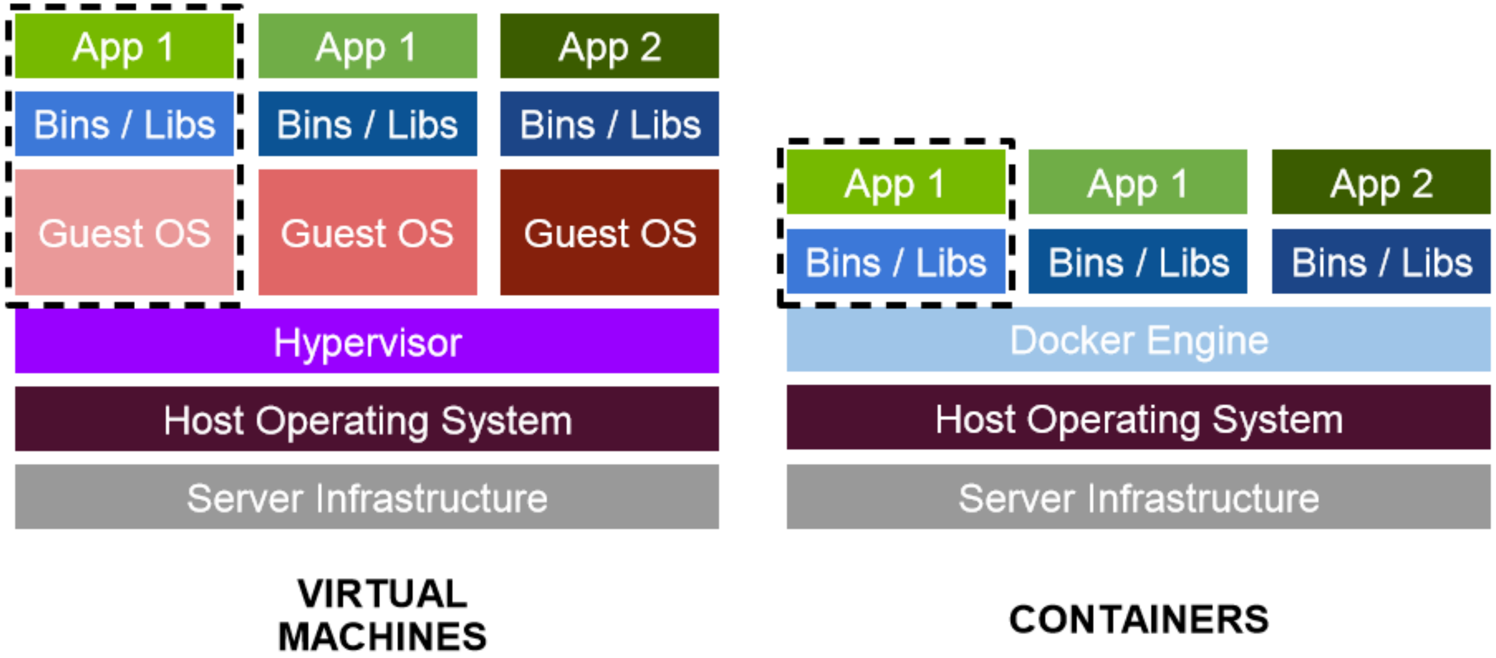
NVIDIA GPU Cloud (NGC)

https://ngc.nvidia.com/catalog



VIRTUAL MACHINE VS. CONTAINER

Not so similar



What’s Docker?

“an open-source project that automates the deployment of software applications inside **containers** by providing an additional layer of abstraction and automation of **OS-level virtualization** on Linux”

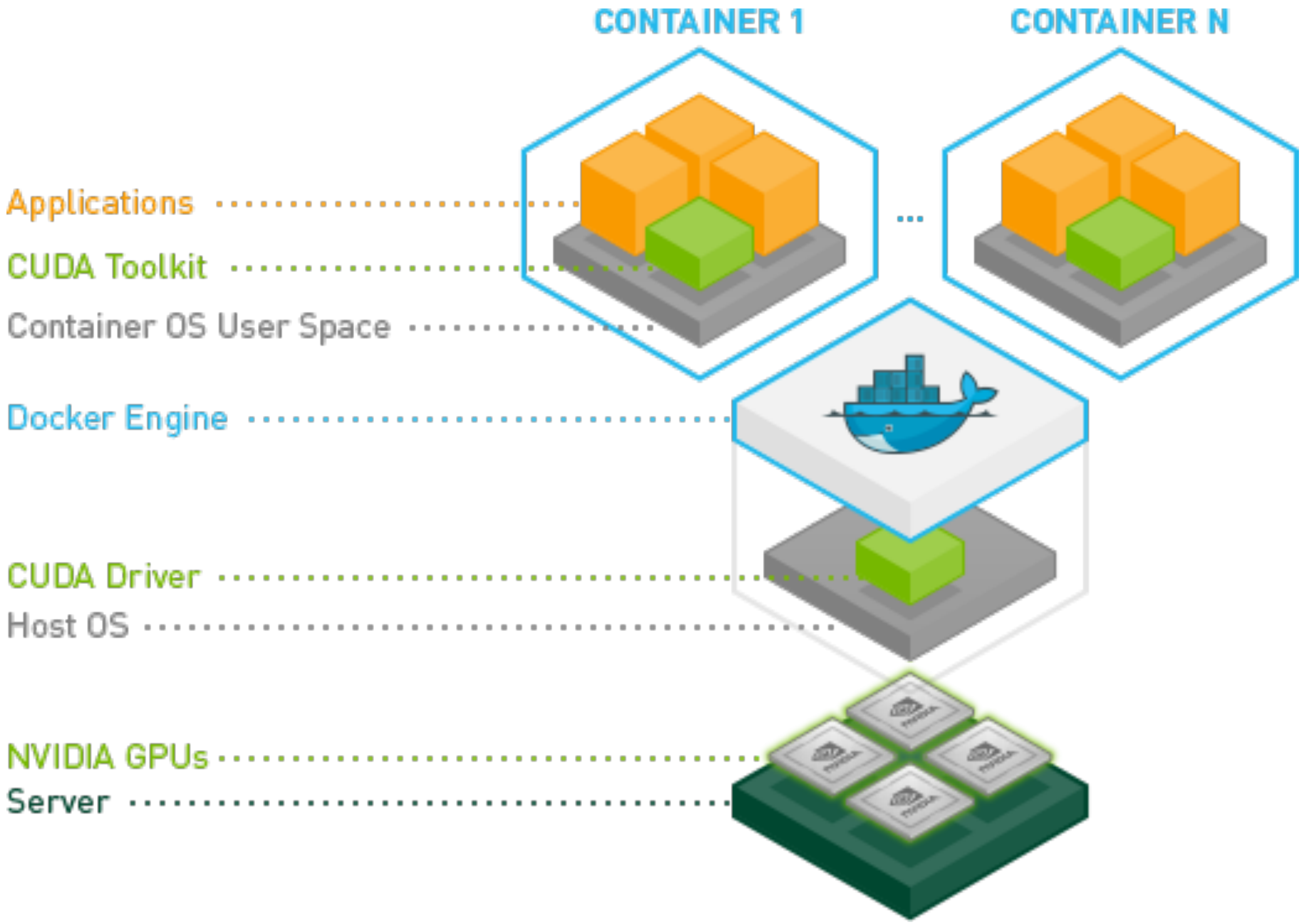
The key benefit of Docker:

- It allows users to **package an application with all of its dependencies into a standardized unit** for software development

Why NVIDIA Docker?

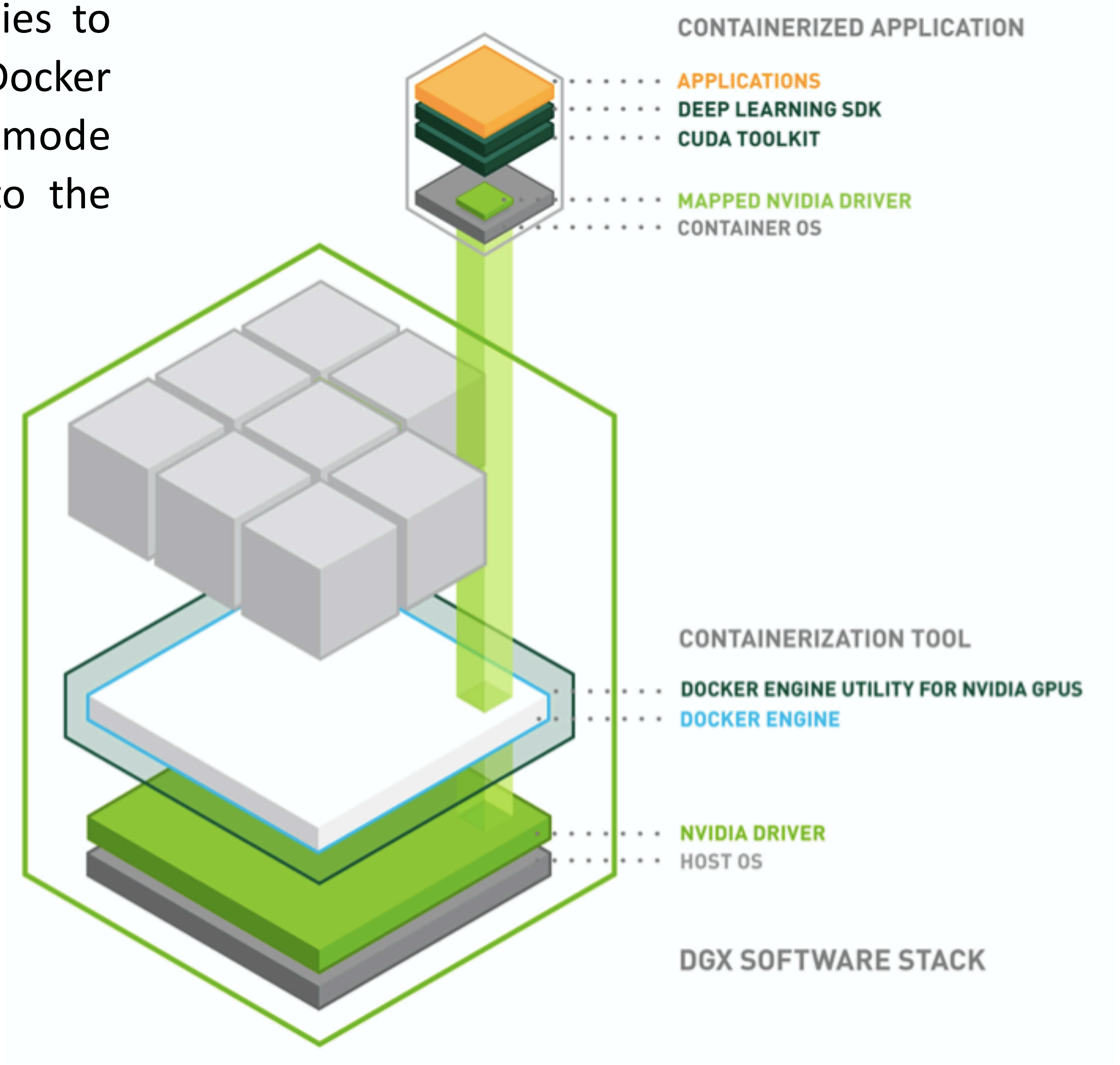
- Docker containers are hardware-agnostic and platform-agnostic
- NVIDIA GPUs are specialized hardware that require the NVIDIA driver
- Docker does not natively supported NVIDIA GPUs with containers
- nvidia-docker makes the images agnostic of the NVIDIA driver

NVIDIA Docker



NVIDIA Docker

Docker containers encapsulate application dependencies to provide reproducible and reliable execution. The Docker Engine Utility for NVIDIA GPUs maps the user-mode components of the NVIDIA driver and the GPUs into the Docker container at launch



NVIDIA Docker Sub-Commands

nvidia-docker pull

nvidia-docker images

nvidia-docker run

nvidia-docker ps

nvidia-docker exec

nvidia-docker commit

nvidia-docker logs

Running Containers

nvidia-docker run -it --rm --name <container_name> -u $(id -u):$(id -g) -p 8080:8888 --net=host -v local_dir:container_dir nvcr.io/nvidia/<framwork_name>:<xx.xx>

Docker run Options:

- **--rm** remove the container after it exits
- **-i -t** or **-it** interactive, and connect a “tty”
- **--name** give the container a name
- **-u $(id -u):$(id -g)** set the ID of the user in the container
- **-p 8080:8888** port map from host to container
- **--net=host** networking stack in the container
- **-v ~/data:/data** map storage volume from host to container (bind mount) i.e. bind the ~data directory in your home directory to /data in the container

Docker on HPC Systems

- HPC systems are shared resources
- Docker’s security model is designed to support trusted users running trusted containers; e.g., users can escalate to root
- Docker not designed to support batch-based workflows
- Docker not designed to support tightly-coupled, highly distributed parallel applications (MPI)
- No native support with open source workload managers like SLURM

Overcome these Issues with Singularity

Singularity: A Container Engine for HPC

- Reproducible, portable, sharable, and distributable containers
- No trust security model: untrusted users running untrusted containers
- No user contextual changes or root escalation allowed; user inside container is always the same user who started the container
- It automatically derived user’s home directory; user can also bind other directories at runtime

Running with Singularity

Save the NGC container as a local Singularity image file:

singularity build <framwork_name>.sif docker://nvcr.io/nvidia/<framwork_name>:<xx.xx>

e.g,

singularity build tensorflow_21.07-tf2-py3.sif docker://nvcr.io/nvidia/tensorflow:21.07-tf2-py3

Run Singularity image file on NVIDIA GPU:

singularity run --nv --bind local_dir:container_dir <framwork_name>.sif <Container-Name>

e.g,

singularity run --nv /opt/apps/sif/tensorflow_21.07-tf2-py3.sif yourcode.py

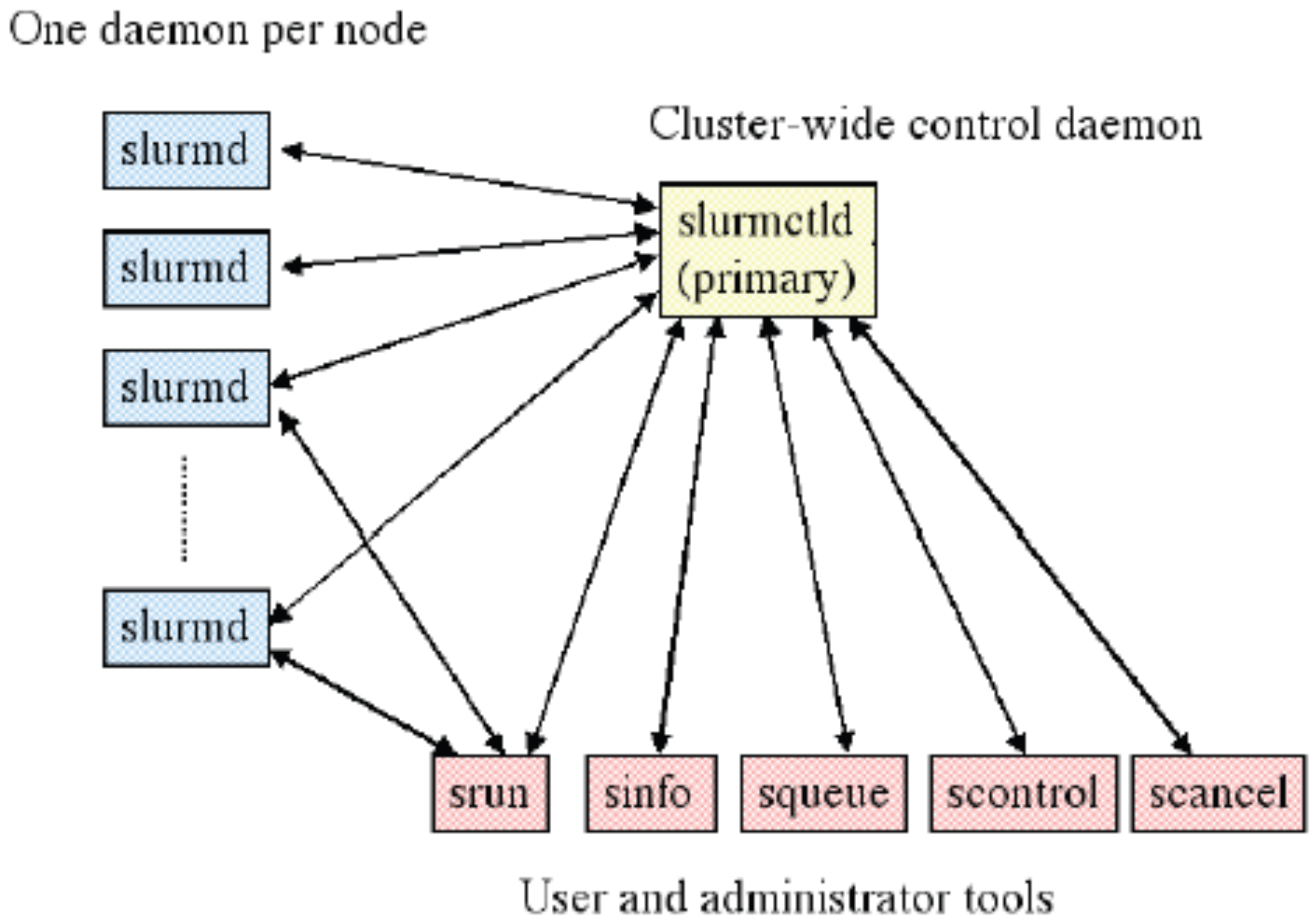
SLURM Overview

SLURM (Simple Linux Utility for Resource Management) is a highly configurable open source workload and resource manager

It provides three key functions:

- It allocates exclusive and/or non-exclusive access to resources to users for some duration of time so they can perform work
- It provides a framework for starting, executing, and monitoring work on a set of allocated resources
- It arbitrates contention for resources by managing a queue of pending work

SLURM Architecture



SLURM Commands for the User

- **sbatch:** Submit a batch script to SLURM

  sbatch <SCRIPT NAME>

- **squeue:** View information about jobs located in the SLURM scheduling queue

  squeue <JOB ID>

- **sinfo:** View information about SLURM nodes and partitions

  sinfo

- **scancel:** Used to signal or cancel a jobs or job steps that are under the control of SLURM

  scancel <JOB ID>

Containers (Singularity) with SLURM Sample Script

TensorFlow Container with SLURM Sample Script:

```
#!/bin/bash
#SBATCH --job-name=tf_test
#SBATCH --ntasks=8
#SBATCH --output=test_tf_%j.out
#SBATCH --gres=gpu:1
#SBATCH --partition=debug
singularity run --nv /opt/apps/sif/tensorflow_21.07-tf2-py3.sif python -c 'import tensorflow as tf; print(tf.__version__)'
```

PyTorch Container with SLURM Sample Script:

```
#!/bin/bash
#SBATCH --job-name=pytorch_test
#SBATCH --ntasks=8
#SBATCH --output=test_pytorch_%j.out
#SBATCH --gres=gpu:1
#SBATCH --partition=debug
singularity run --nv /opt/apps/sif/pytorch_21.07-py3.sif python -c 'import torch; print(torch.__version__)'
```

Thanks!