Introduction to GPU Computing on Raad2

February 2026

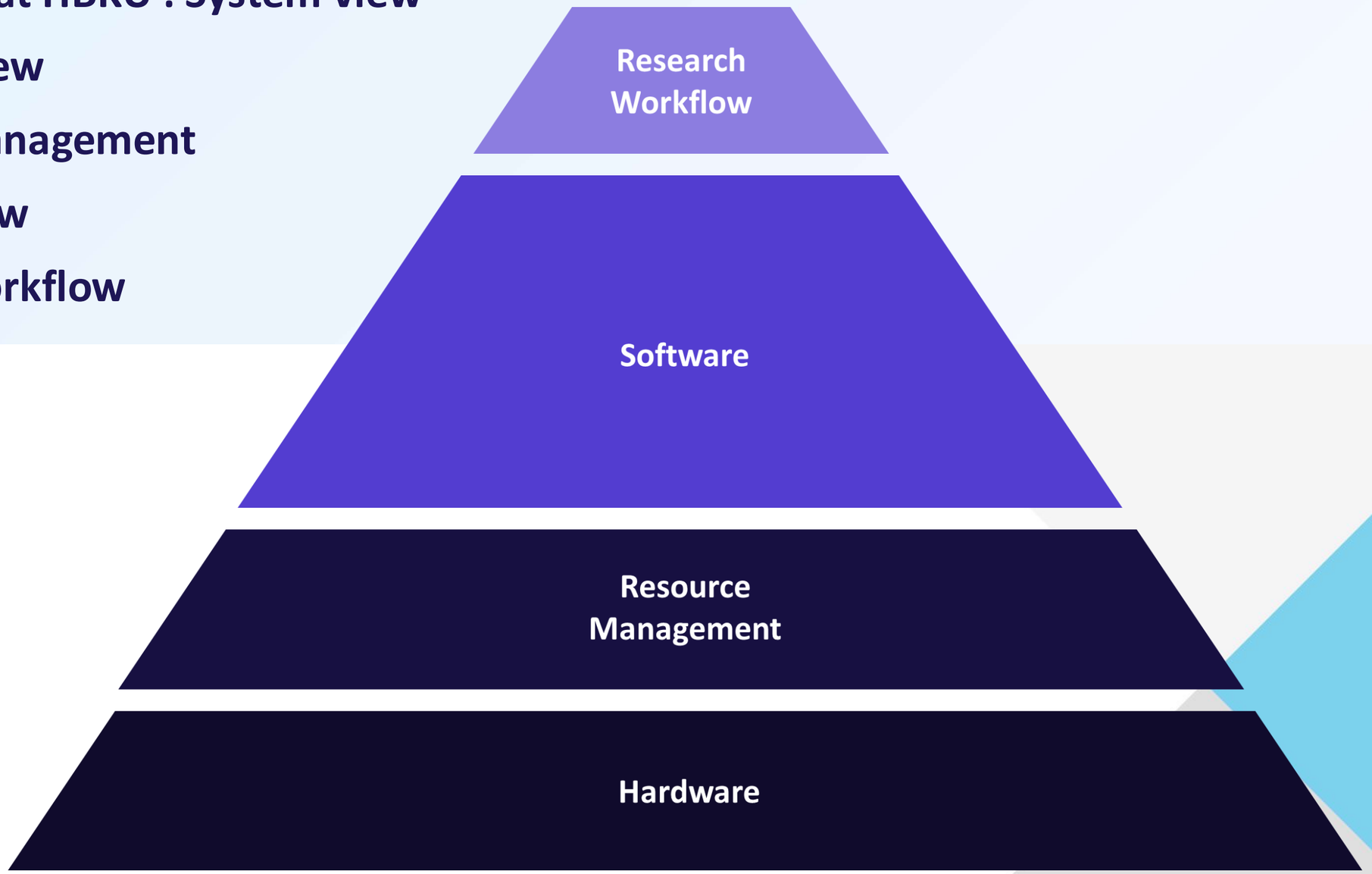
RCCG in-house Training Session

كلية العلوم والهندسة
College of Science & Engineering
جامعة حمد بن خليفة
HAMAD BIN KHALIFA UNIVERSITY

Outline

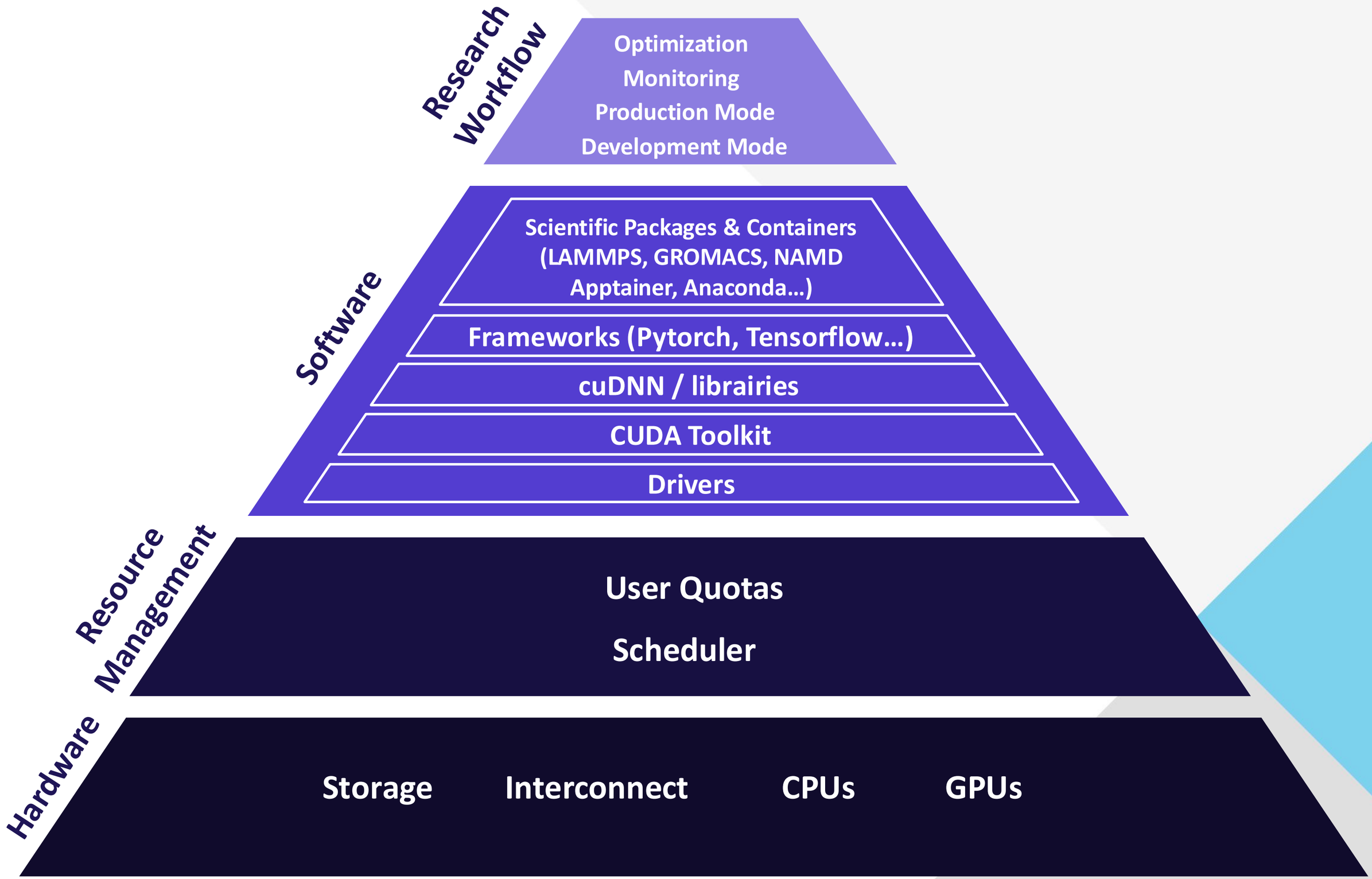
Today's Focus

- **GPU Cluster at HBKU : System view**
- **Hardware view**
- **Resource Management**
- **Software view**
- **Research Workflow**



The “Introduction to Linux” and “Introduction to computing on raad2” short courses offered by Research Computing should be treated as a prerequisite for this course.

From Hardware to Research Output



Introduction

➔ Raad2-gfx : 4-node GPU cluster with **8 NVIDIA Volta V100** GPUs was acquired in 2019

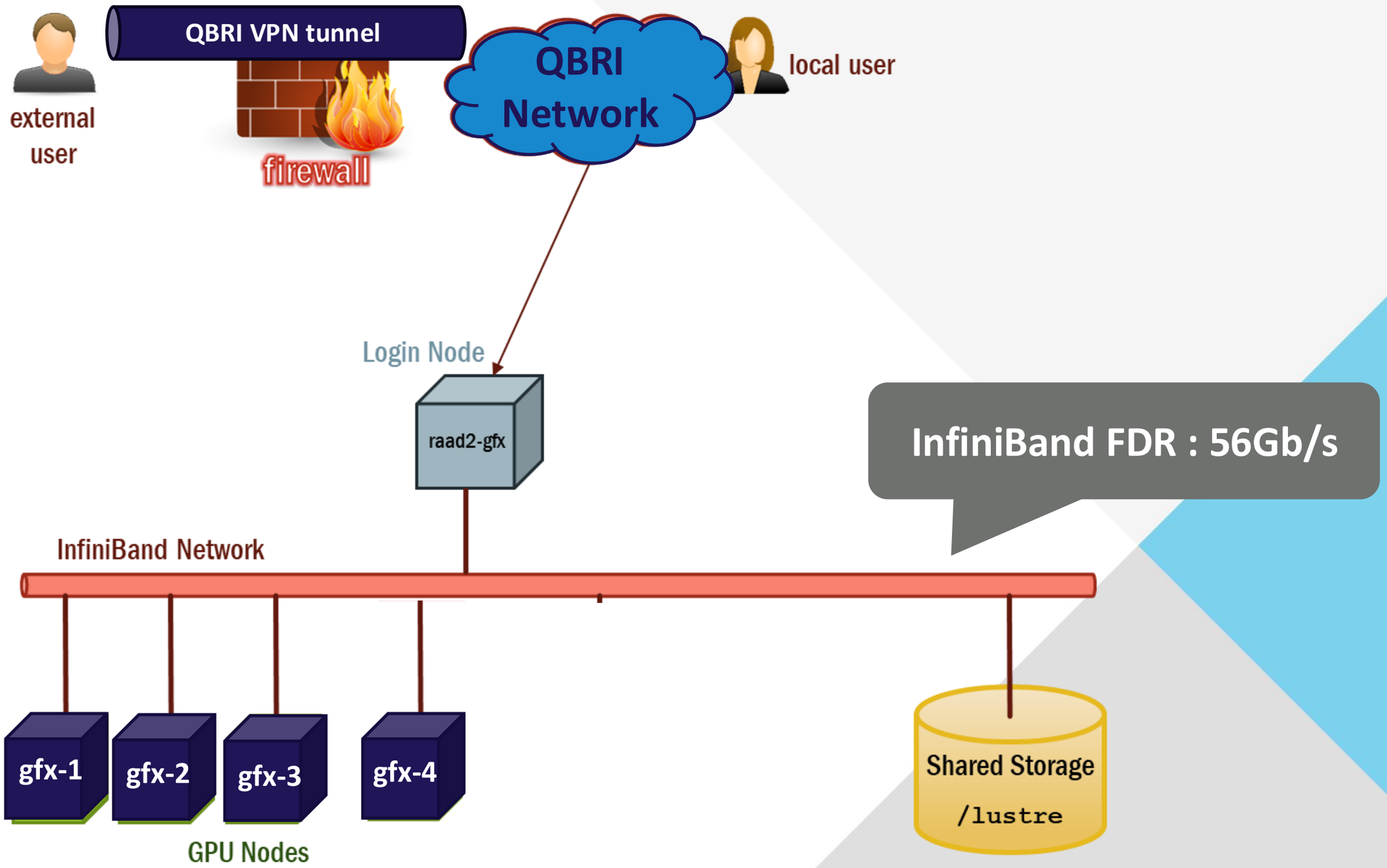
- Raad2-gfx is “part of” Raad2 but it is a different cluster
  - Shares same storage system i.e. Lustre
  - Shares same authentication mechanism & passwords are synced
  - Different node configurations (HW) & high-speed network
  - Different Resource Manager : PBS vs. SLURM
  - Different OS & Software Stack : Rocky Linux 8.10 vs. SUSE SLES 15.3
  - Different interconnect: InfiniBand vs. CRAY ARIES

➔ Supports multiple AI/ML and HPC applications
➔ Support Containers

Hardware

System Architecture

➔ There are 4 GPU Nodes interconnected via high-speed network

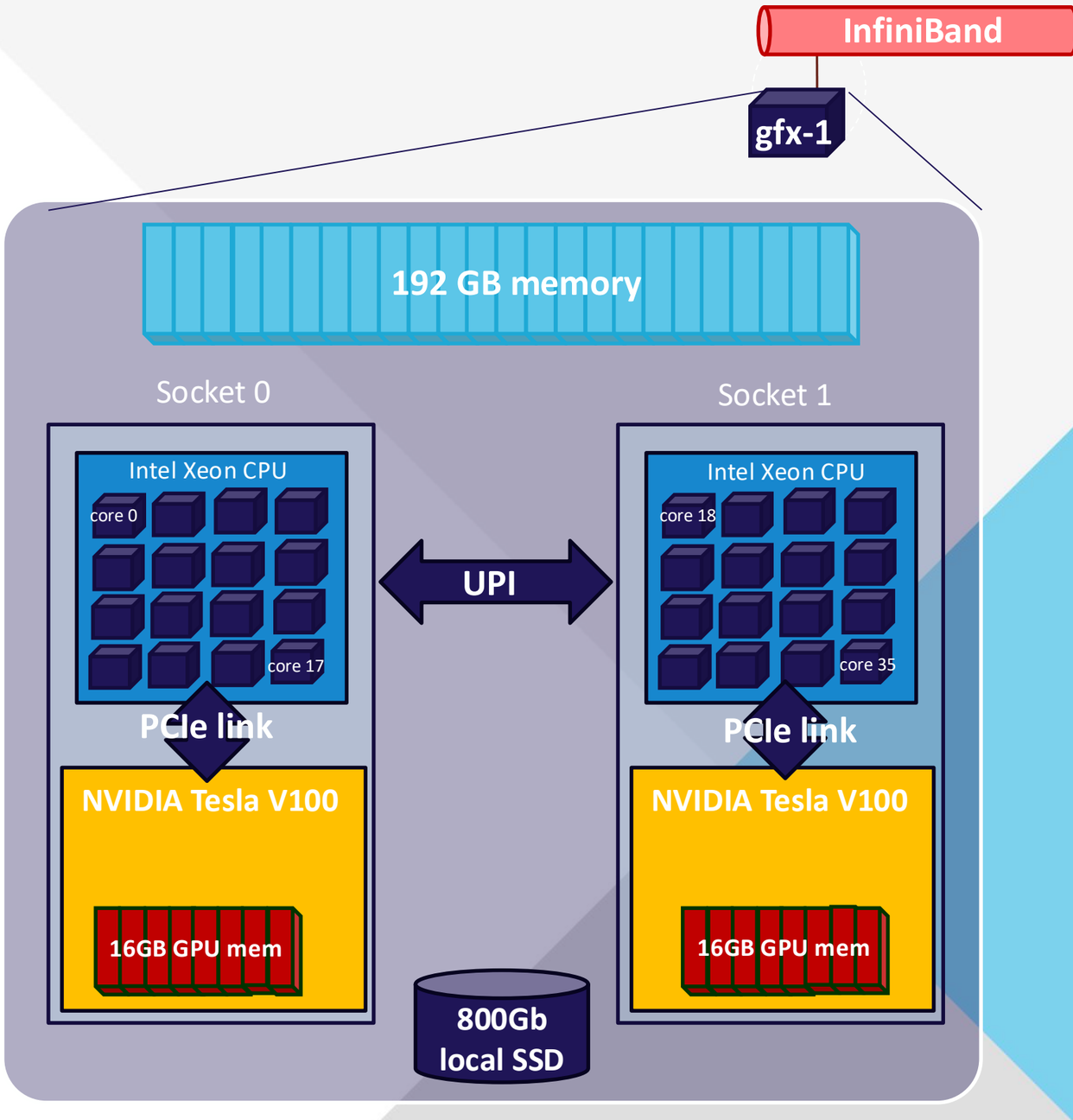


GPU Node Architecture

➔ Each node has:

Login Node	raad2-gfx.hbku.edu.qa
GPU	2* NVIDIA Tesla V100 Per Node
Interconnect	PCIe link
GPU Node names	gfx[1-4]
Memory	192 GB per Node
CPU	2* Intel Xeon 6140, 18 cores, 2.30 GHz Base & 3.70 GHz Turbo Freq.
Sockets	2 Per Server
Node Local Storage	800GB per node (SSD)

what can you notice here between the 2 GPUs ?



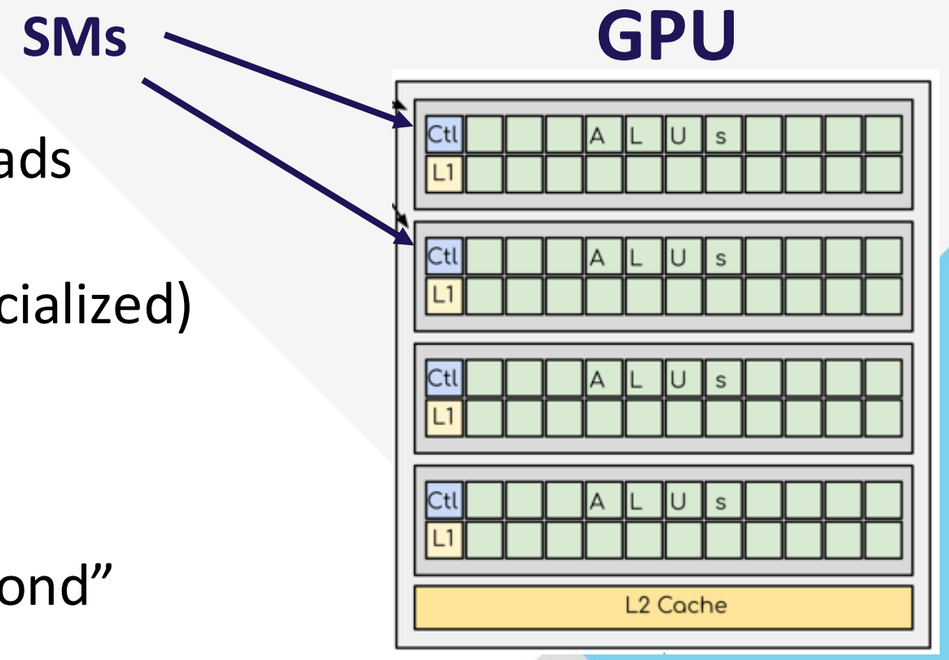
Anatomy of a GPU

Context

- GPU = Graphics Processing Unit
- Designed to render millions of pixels simultaneously
- Originally built for video games, revolutionized scientific computing !
- CUDA (2006) was the turning point
- Specialized parallel HW for floating point ops

Building blocks

- GPU = many-core processor optimized for parallel workloads
- **SM** (Streaming Multiprocessors), 1 SM is made of :
  - CUDA Cores : general math engine (flexible but not specialized)
  - Tensor Cores: specialized matrix ops (AI acceleration)
  - Up to 64 active Warps (+ 4 Warp Schedulers)
    - 1 warp = group of 32 threads executing the same instruction
- Perf is measured **FLOPS**: “floating point operation per second”
- L1 and L2 caches
- VRAM : High-bandwidth memory (HBM2) for fast data access



➔ To use the GPU efficiency, task must be split in many independent sub-tasks

GPUs excel at tasks with many parallel operations !

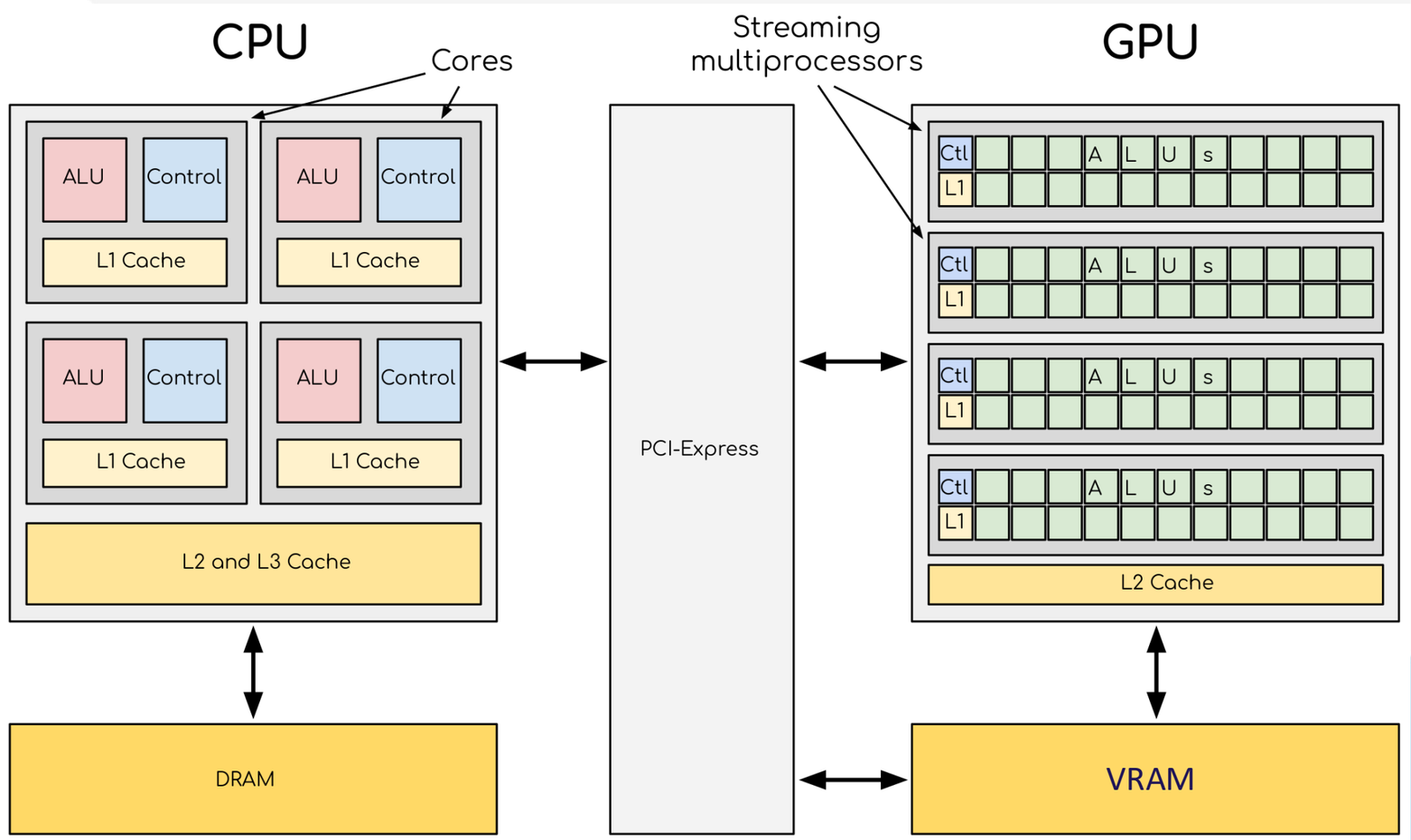
CPU vs GPU: When and Why It Matters

CPU

- Few cores (e.g., 36)
- Complex cores
- Optimized for **sequential** logic
- Large cache
- Workload: Serial, complex

GPU

- Thousands of simpler cores
- Designed for **parallel** throughput
- Massive arithmetic capability
- Smaller control logic
- Workload: Parallel, repetitive



FFTs	Branching logic
Matrix operations	Large vector ops
Serial code	Small datasets
Particle simulations	Deep learning
I/O-bound workflows	

CPU workload 1

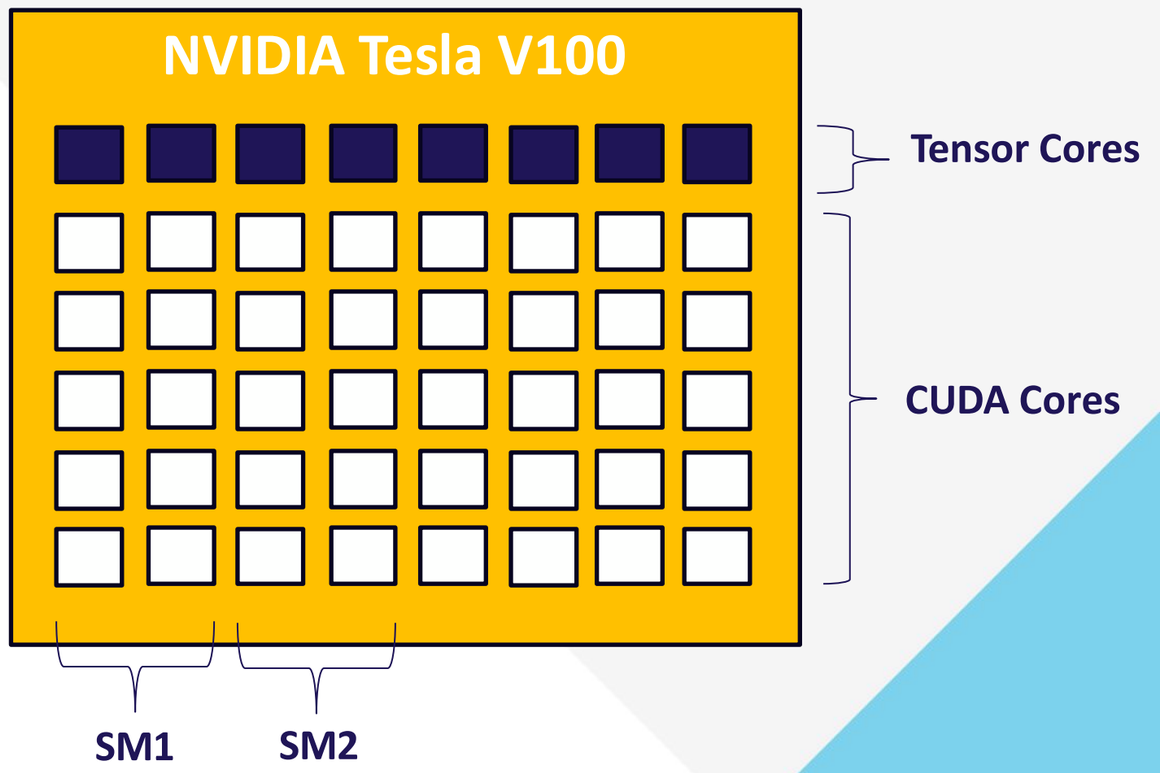
GPU workload 2

→ CPU still controls the workflow, delegating highly-parallel tasks to the GPU(s)

Zoom on NVIDIA Tesla V100

- Nvidia Tesla V100 is a bit old,
- But remains a powerful datacenter GPU
- Designed for AI, HPC and graphics

GPU Architecture	NVIDIA V100
SM count	80
CUDA Cores	5,120
Tensor Cores (introduced in V100)	640
Warps	Up to 64 active per SM
Double precision performance (FP64)	~7 TFLOPS
Single precision performance (FP32)	~14 TFLOPS
GPU Memory (VRAM)	16GB
Memory bandwidth	900 GB/sec
Compute APIs	CUDA, DirectCompute, OpenCL, OpenACC



→ GPUs executes threads in groups (warps)

→ If threads diverge, performance drops

Some additional Precisions…

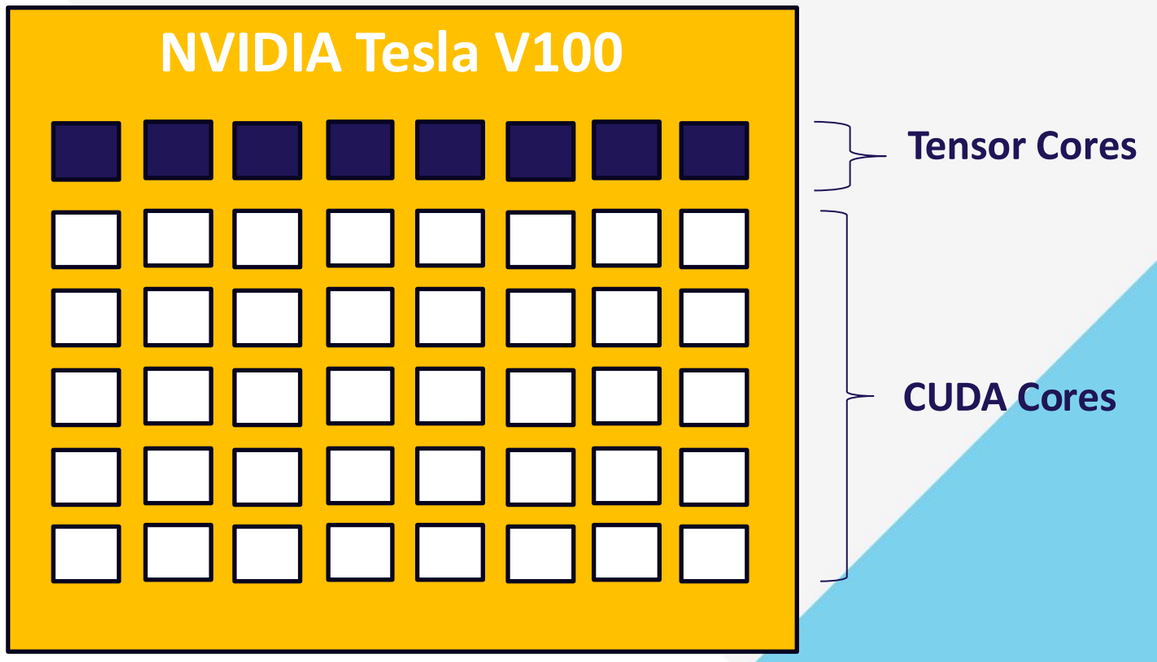
High Precision use cases (FP64)
- Computational Fluid Dynamic (CFD)
- Quantum chemistry
- Molecular Dynamics
- Climate simulation
- Finite-element modeling

Simple Precision use cases (FP32)
- Traditional Deep Learning
- Computer Vision
- NLP

Half Precision use cases (FP16)
- Modern Deep Learning training
- Real-time/cloud/mobile inference
- Generative AI transformers

Mixed Precision use cases (FP16+FP32)
- Modern Deep Learning
- Multi-GPU training

- 32 and FP64 run on CUDA Cores
- Tensor Cores accelerates FP16 / mixed precisions
- FP16 on Tensor Cores: ~125 TFLOPS !!



Choose the right precision for your workload + HW: FP64 for scientific accuracy, FP32 for good balance, FP16 for maximum DL throughput

Hands-On: Access & Exploration

Access the raad2-gfx login nodes the same way you access the raad2 login nodes:

1) QBRI VPN connect (FortiClient)
2) ssh username@raad2-gfx (if name is not resolved by DNS, try: username@192.168.41.15)
3) (use Mobaxterm to improve your user experience)

Task 1

Explore Node HW info 1/2

- lscpu
- free –h
- lspci | grep V100

What do you notice ?

Task 2

Explore Node HW info 2/2

- sinteractive
- Task 2 commands
- exit

Hostname of the GPU node ?
GPU Memory ?

Task 3

Resource Management

User Resource Limits

➔ raad2-gfx is a shared resource among all HBKU members: it should be fairly allocated

Queue	workq
Local SSD Storage	/local (400GB)
Per User GPU Limit	2 GPUs
Per User CPU Limit	36 CPUs
Max simultaneous jobs per user	4 Jobs
Max Wall time	48 hours (default = 1 hour)

Purpose: fairness and resource efficiency

Resource Management : PBS Pro

A different Scheduler : PBS Pro

- Uses PBS (2025.2.1) as resource manager and job scheduler
- If you know SLURM, you know already know PBS (well, almost)

	SLURM	PBS
Interactive job	salloc / srun	qsub -I -l
Submit batch job	sbatch script.sh	qsub script.sh
Show queue	squeue	qstat
Show job details	scontrol show job	qstat -f
Cancel job	scancel jobid	qdel jobid
Node info	sinfo	pbsnodes -a

PBS Pro is like SLURM, but with more features

Resource Management : PBS

Resource Request

- SLURM and PBS express resources differently.

SLURM

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks=4
#SBATCH --gres=gpu:1
#SBATCH --time=01:00:00

#SBATCH   –-partition=workq
#SBATCH   –-mem=8G
```

PBS

```
#!/bin/bash
#PBS -l select=1:ncpus=4:ngpus=1:mem=8gb
#PBS -l walltime=01:00:00

#PBS   –q  workq
```

Task 1

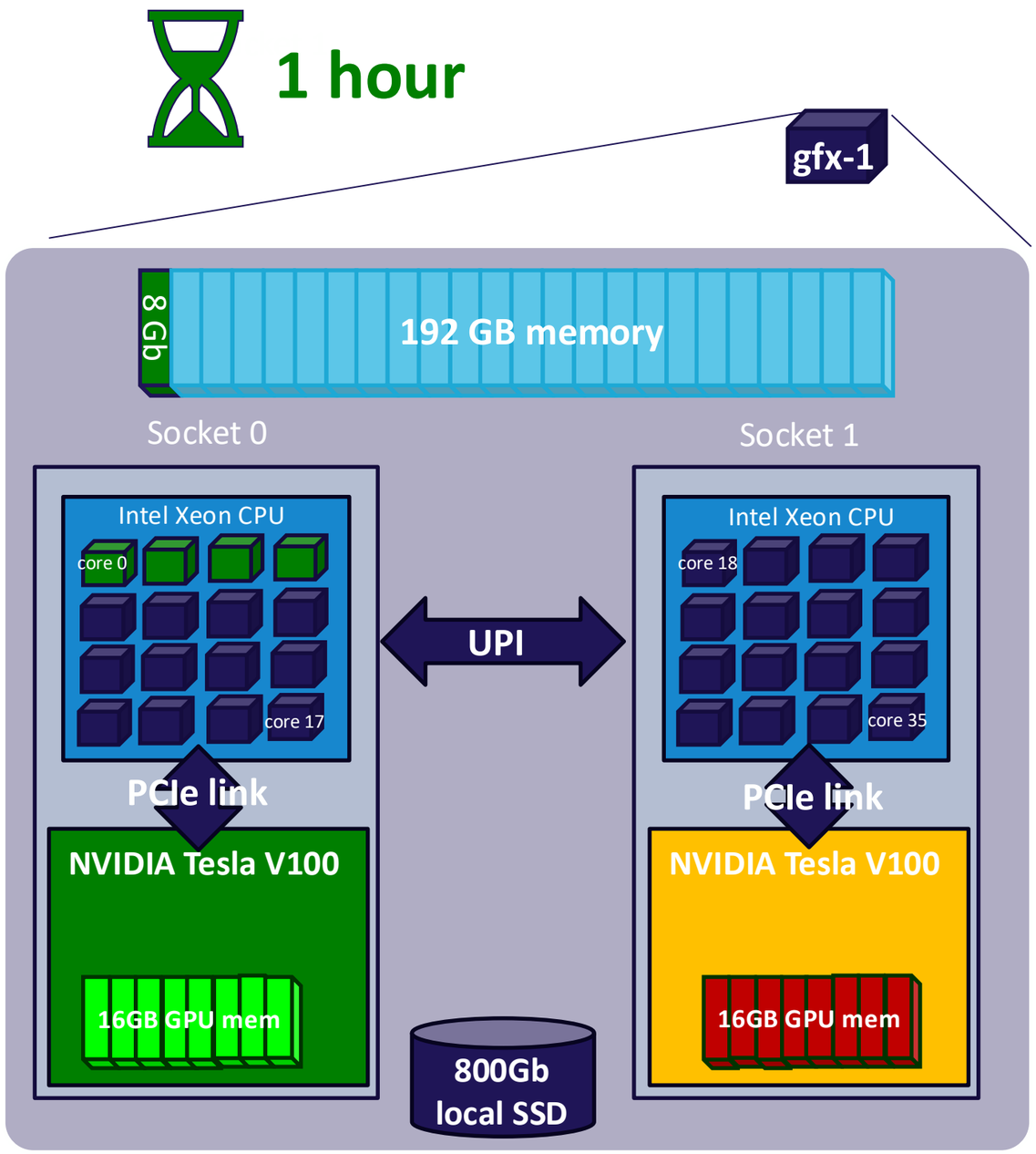
Check resource and queue

- pbnodes –a
- qstat

Resource Management : PBS

Resource Request

- SLURM and PBS express resources differently.



PBS

```
#!/bin/bash
#PBS -l select=1:ncpus=4:ngpus=1:mem=8gb
#PBS -l walltime=01:00:00
#PBS   –q  workq
```

For now, ignore this

Task 1

Check resource and queue

- pbnodes –a
- qstat

Resource Management : PBS

Resource Request

- Good ratio example:
  - 1 GPU -> 4-8 CPUs
  - 2 GPUs -> 8-16 CPUs
  - extended range: up to 36 CPUs if justified (e.g., CPU-heavy preprocessing, hybrid workloads)

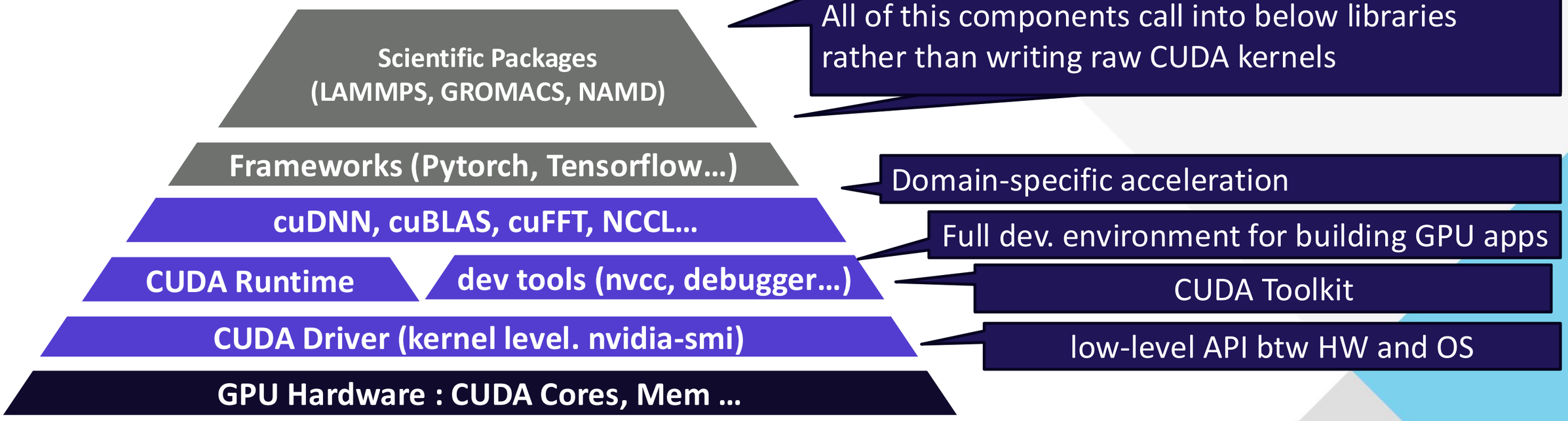
```
#!/bin/bash
#PBS -l select=1:ncpus=4:ngpus=1:mem=8gb
#PBS -l walltime=01:00:00
#PBS  -q  workq
```

```
#!/bin/bash
#PBS -l select=1:ncpus=16:ngpus=2: mem=16gb
#PBS -l walltime=01:00:00
#PBS  -q  workq
```

Software

The NVIDIA Software Ecosystem

➔ CUDA (Compute Unified Device Architecture) is the core
➔ It enables: C/C++, Fortran, Python, AI frameworks

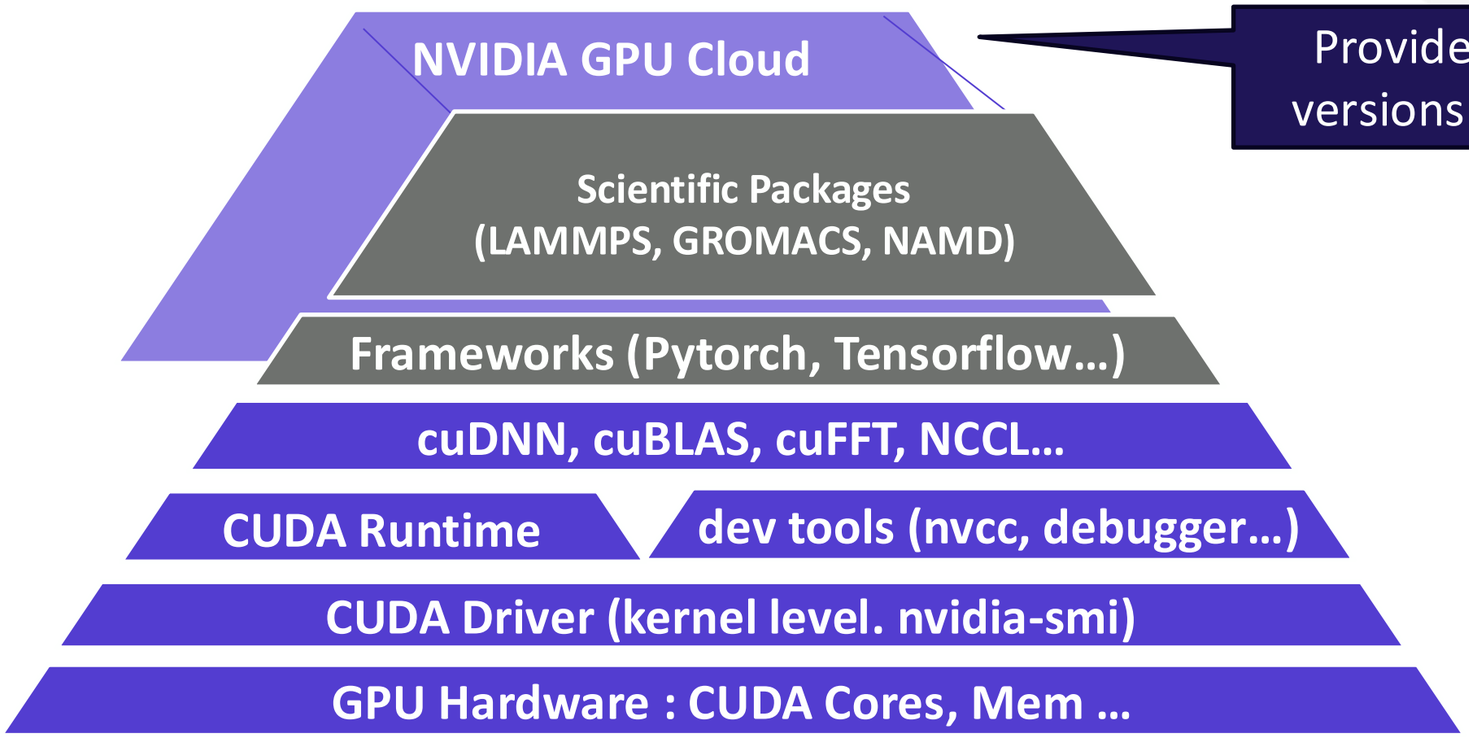


➔ Nvidia dominance: early CUDA ecosystem (2006), developer adoption, deep learning de facto standard, and HW innovation

This layered architecture allows you to focus on higher-level algorithms instead of low-level GPU optimization

The NVIDIA Software Ecosystem

➔ CUDA (Compute Unified Device Architecture) is the core
➔ It enables: C/C++, Fortran, Python, AI frameworks



Provides containers with pre-built, GPU-optimized versions of frameworks, libraries and scientific apps.

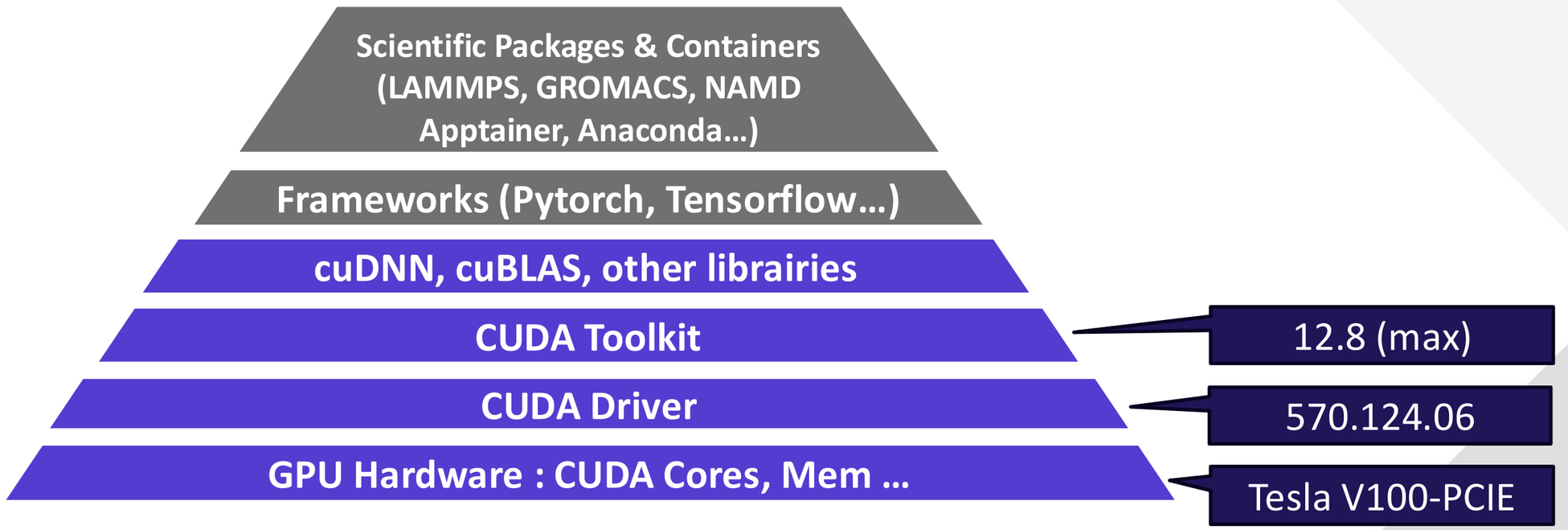
NGC boosts the reproducibility of SW environment and speed up the research workflow

The NVIDIA Software Ecosystem

- nvidia-smi

```
Mon Mar  9 12:17:47 2026
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 570.124.06             Driver Version: 570.124.06     CUDA Version: 12.8     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |                  MIG M. |
|=========================================+========================+======================|
|   0  Tesla V100-PCIE-16GB           On  |   00000000:18:00.0 Off |                    0 |
| N/A   24C    P0             23W /  250W |       1MiB /  16384MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+
|   1  Tesla V100-PCIE-16GB           On  |   00000000:AF:00.0 Off |                    0 |
| N/A   23C    P0             23W /  250W |       1MiB /  16384MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI              PID   Type   Process name                      GPU Memory |
|        ID   ID                                                             Usage      |
|=========================================================================================|
|  No running processes found                                                             |
+-----------------------------------------------------------------------------------------+
```



Software Stack on raad2-gfx

- sinteractive
- module avail
- nvidia-smi
- module list
- module load cuda12.2
- module list

Task 1

Which CUDA versions are available? What is the latest CUDA version available?

```
[abelfer21@raad2-gfx ~]$ module avail
------------------------------------------------------------ /cm/local/modulefiles ------------------------------------------------------------
boost/1.81.0            cm-bios-tools  cuda-dcgm/4.1.1.1  gcc/13.1.0      luajit        module-info  python3     sedutil/1.16.0
cluster-tools-dell/10.0 cmd            dot                ipmitool/1.8.19 mariadb-libs  null         python39    shared
cluster-tools/10.0      cmjob          freeipmi/1.6.14    lua/5.4.6       module-git    openldap     python3113  singularitypro/4.3.3

------------------------------------------------------------ /cm/shared/modulefiles ------------------------------------------------------------
anaconda/2024.10                 cuda12.2/nsight/12.2.2      gdb/13.1                          julia/1.11.8              openblas/dynamic/0.3.18
blacs/openmpi/gcc/64/1.1patch03  cuda12.2/profiler/12.2.2    globalarrays/openmpi/gcc/64/5.8   lammps/2023.06            openmpi/gcc/64/4.1.5
blas/gcc/64/3.11.0               cuda12.2/toolkit/12.2.2     hdf5/1.14.0                       lapack/gcc/64/3.11.0      openmpi4/gcc/4.1.5
bonnie++/2.00a                   cuda12.8/blas/12.8.0        hdf5_18/1.8.21                    mpich/ge/gcc/64/4.1.1     R/4.5.1
cm-pmix3/3.1.7                   cuda12.8/fft/12.8.0         hwloc/1.11.13                     mvapich2/gcc/64/2.3.7     ucx/1.14.1
cm-pmix4/4.1.3                   cuda12.8/toolkit/12.8.0     hwloc2/2.8.0                      namd/3.0.1
cmake/3.29                       cudnn9.1-cuda12.2/9.1.1.17  ior/4.0.0                         netcdf/gcc/64/gcc/64/4.9.2
cuda12.2/blas/12.2.2             default-environment         iozone/3.494                      netperf/2.7.0
cuda12.2/fft/12.2.2              fftw3/openmpi/gcc/64/3.3.10 iperf/3.17.1                      openblas/dynamic/(default)
[abelfer21@raad2-gfx ~]$
```

What do you notice with CUDA ?

Both are now loaded, conflict !!

- nvidia-smi
- exit
- module load cuda12.8
- module list
- module unload cuda12.2
- module list

Task 2

Software Stack on raad2-gfx

```
[abelfer21@raad2-gfx ~]$ module avail
------------------------------------------------------------------ /cm/local/modulefiles ------------------------------------------------------------------
boost/1.81.0             cm-bios-tools  cuda-dcgm/4.1.1.1  gcc/13.1.0      luajit        module-info  python3     sedutil/1.16.0
cluster-tools-dell/10.0  cmd            dot                ipmitool/1.8.19  mariadb-libs  null         python39    shared
cluster-tools/10.0       cmjob          freeipmi/1.6.14    lua/5.4.6        module-git    openldap     python3113  singularitypro/4.3.3

------------------------------------------------------------------ /cm/shared/modulefiles ------------------------------------------------------------------
anaconda/2024.10                 cuda12.2/nsight/12.2.2       gdb/13.1                          julia/1.11.8              openblas/dynamic/0.3.18
blacs/openmpi/gcc/64/1.1patch03  cuda12.2/profiler/12.2.2     globalarrays/openmpi/gcc/64/5.8   lammps/2023.06            openmpi/gcc/64/4.1.5
blas/gcc/64/3.11.0               cuda12.2/toolkit/12.2.2      hdf5/1.14.0                       lapack/gcc/64/3.11.0      openmpi4/gcc/4.1.5
bonnie++/2.00a                   cuda12.8/blas/12.8.0         hdf5_18/1.8.21                    mpich/ge/gcc/64/4.1.1     R/4.5.1
cm-pmix3/3.1.7                   cuda12.8/fft/12.8.0          hwloc/1.11.13                     mvapich2/gcc/64/2.3.7     ucx/1.14.1
cm-pmix4/4.1.3                   cuda12.8/toolkit/12.8.0      hwloc2/2.8.0                      namd/3.0.1
cmake/3.29                       cudnn9.1-cuda12.2/9.1.1.17   ior/4.0.0                         netcdf/gcc/64/gcc/64/4.9.2
cuda12.2/blas/12.2.2             default-environment          iozone/3.494                      netperf/2.7.0
cuda12.2/fft/12.2.2              fftw3/openmpi/gcc/64/3.3.10  iperf/3.17.1                      openblas/dynamic/(default)
[abelfer21@raad2-gfx ~]$
```

- CUDA toolkit 12.2 & 12.8
- cuDNN 9.1
- cuBLAS 12.2 & 12.8
- SingularityPro 4.3.3
- Anaconda 2024.10
- Python 3.9 & 3.11.3
- R 4.5.1
- Julia 1.11.8
- lammps 2023
- namd 3.0.1

User applications and code

➔ You can bring your code/applications to raad2-gfx the same way as in raad2 by using:

Self compile/install

- You have root privileges in your home directory
- You can download source code from github, gitlab, vendor repos…
- And compile/run it using gcc, CMake/Make, Python…

Anaconda

- popular open-source ecosystem bundling Python with over 1,500 scientific and data science packages
- includes conda, a powerful package manager

SingularityPro

- To import containerized applications
- Same as Docker but for HPC
- More details at the end

User Container

Pre-built Container

Nvidia Cloud GPU

- Pre-built / optimized containers provided by Nvidia
- Can be imported by using SingularityPro

Zoom on Anaconda

➔ Easiest ecosystem to perform Python/R data science and machine learning on raad2-gfx

Example

```
source /cm/shared/apps/anaconda/2024.10/etc/profile.d/conda.sh
conda create -n dl_env python=3.11 -y
conda activate dl_env
(dl_env ) $ conda search pytorch-cuda -c pytorch -c nvidia
(dl_env ) $ conda install pytorch pytorch-cuda=12.1 numpy -c pytorch -c nvidia -c conda-forge -y
(dl_env ) $ conda list (option)
(dl_env ) $ conda deactivate
(dl_env ) $ conda env list (option)
```

Pytorch is now ready to use !

- Quickly download 1,500+ Python/R data science packages
- Conda: Manage libraries, dependencies, and environments
- Develop and train machine learning and deep learning models with Pytorch, scikit-learn, TensorFlow, and Theano
- Analyze data with scalability & performance with Dask, NumPy, pandas, Numba
- Visualize results with Matplotlib, Bokeh, Datashader, and Holoviews
- Sometimes, a package or a specific version is not available in Anaconda channels
  - no worries, you can use pip from within your conda environment !

ANACONDA®

GPU Workflow

Typical GPU Workflow on an HPC System

➔ Keep in mind that it is a shared system (university-wide)

Login → Development Mode → Production Mode

- Connect remotely the cluster
- Land on login node
- Not meant for heavy computation !
- Used to:
  - edit code
  - prepare env.
  - submit jobs

Typical GPU Workflow on an HPC System

➔ Keep in mind that it is a shared system (university-wide)

Login

1

- Request a GPU interactive session via
  - sinteractive
  - or qsub -I -l select=1:ncpus=8:ngpus=1
- PBS will allocate a free GPU node + open a shell on that node

Development Mode

2

- Prepare Software Env.
  - module load lammps
  - conda activate torch-gpu
- GPU sanity check:
  - nvidia-smi
  - run a light version of your script
  - nvidia-smi in other terminal to monitor gpu usage

Production Mode

3

- Iterative development
  - edit code -> run -> adjust -> run
  - e.g. change model size, algo…
  - fast experimentation and debug
- Validate final script
  - Training run, no crashes
  - Output files generated

Typical GPU Workflow on an HPC System

➔ Keep in mind that it is a shared system (university-wide)

Login → Development Mode → Production Mode

- Also called “batch mode”
- For long running jobs (> 1 hour)
- Done by submitting PBS script e.g. `qsub batch_job.pbs`
- User can have multiple jobs active at the same time
- User can “fire and forget”
- User can monitor the state of his/her jobs : qstat

Typical GPU Workflow on an HPC System

➔We will explore this workflow via a simple but representative example:

small_dl_seqential.py

- Pure Python script
- Creates a simple neural network that learns to classify random data into 10 categories using GPU acceleration
- Uses PyTorch to perform deep learning training on a GPU


```
Your Python code
        ↓
PyTorch
        ↓
CUDA runtime
        ↓
CUDA libraries (cuBLAS, cuDNN)
        ↓
NVIDIA driver
        ↓
V100 GPU hardware
```

Simple Deep Learning example

Everything that should run on GPU must be placed on the device

➔ In PyTorch, a device is simply an object that tells tensors where they should live:

- "cpu" : run on CPU
- "cuda“ : run on the default GPU

PyTorch doesn’t automatically move data between CPU & GPU
You must manually place tensors & models on a device

Moves all model weights in VRAM

Training loop

views (slices) of the original tensors

makes prediction

how wrong ?

how each weight contributed to the error?

adjust params to reduce loss

A GPU accelerates computation only when both the data & the model are placed on the GPU device

```python
import torch
import time

# Check if GPU is available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")
if torch.cuda.is_available():
    print(f"GPU Name: {torch.cuda.get_device_name(0)}")
    print(f"GPU Memory: {torch.cuda.get_device_properties(0).total_memory / 1e9:.2f} GB\n")

# Create large random dataset
print("Creating dataset...")
X = torch.randn(100000, 512, device=device)  # 100k samples, 512 features
y = torch.randint(0, 10, (100000,), device=device)  # 10 classes

# Simple neural network
model = torch.nn.Sequential(
    torch.nn.Linear(512, 1024),
    torch.nn.ReLU(),
    torch.nn.Linear(1024, 512),
    torch.nn.ReLU(),
    torch.nn.Linear(512, 10)
).to(device)

criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

# Training
print("Starting training...\n")
batch_size = 512
epochs = 10

start_time = time.time()

for epoch in range(epochs):
    epoch_loss = 0
    for i in range(0, X.size(0), batch_size):
        # Get batch
        X_batch = X[i:i+batch_size]
        y_batch = y[i:i+batch_size]

        # Forward pass
        outputs = model(X_batch)
        loss = criterion(outputs, y_batch)

        # Backward pass
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        epoch_loss += loss.item()

    avg_loss = epoch_loss / (X.size(0) // batch_size)
    print(f"Epoch {epoch+1}/{epochs} - Loss: {avg_loss:.4f}")

elapsed_time = time.time() - start_time
print(f"\nTraining completed in {elapsed_time:.2f} seconds")
print(f"Average time per epoch: {elapsed_time/epochs:.2f} seconds")
```

Pytorch framework already compiled against CUDA and cuDNN

This creates tensors directly on the GPU memory

Feed Forward MPL
~1M parameters

Memory footprint:
X has 100,000 × 512 ≈ 51.2M floats. With float32 (4 bytes), that’s ~205 MB just for X. y is ~100k integers (int64), ~0.8 MB

Development Mode: Interactive GPU Sessions

Login → Development Mode → Production Mode

➔ Develop and test the code before production

Option 1: Command line

- Run training scripts directly from the terminal
- python train_model.py
- Fast and lightweight workflow
- Easy to integrate with PBS batch jobs later
- Output stored in log files
- Preferred for reproducible and automated workflows

Option 2: Jupyter Lab / Notebooks

- Run training scripts directly from the terminal
- Interactive Python env. accessible from a web browser
- Code executed cell by cell
- Immediate visualization of results (plots, metrics)
- Supports data exploration and model prototyping
- Good for : testing ideas, visualizing training curves, inspecting datasets, teaching and demos

Both run inside the interactive GPU session

Development Mode: Command line

➔ Develop and test the code before production

- To submit an interactive job to the system, issue: `sinteractive`
  - Allocates 1 GPU with 4 CPU cores for 1 hour with ssh access.

- Submit an interactive job with customized parameters:

```
qsub -I -N param -l select=1:ncpus=4:ngpus=1:mem=16 -l walltime=02:00:00
```

- Load required programming environment.

```
module load cuda12.8
```

- Run your app (e.g. for CUDA C++ code)

```
nvcc mycode.cu
```

- Terminate the interactive session.

```
exit
```

Development Mode: Command line

➔ In our case

- submit an interactive job
  `sinteractive`
- No need to load any system package, we will use full Anaconda environment
  - Enable anaconda:

```
source /cm/shared/apps/anaconda/2024.10/etc/profile.d/conda.sh
```

  - Activate a conda env prepared for this demo:

```
conda activate /ddn/sw/cm/shared/apps/conda/envs/pytorch_demo
```

- Run your app:

```
(pytorch_demo) $ python3 small_dl_seqential.py
```

- Terminate the interactive session
  `exit`

You can also try to activate the env. we created earlier (dl_env) via: conda activate dl_env

Power consumption

VRAM

GPU % (instant)

```
[abelfer21@gfx1 ~]$ nvidia-smi
Tue Mar 10 11:05:25 2026
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 570.124.06             Driver Versio  570.124.06     CUDA Versi  12.8      |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Pers stence-M | Bus-I          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr  Usage/Cap |                Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  Tesla V100-PCIE-16GB           On  |   00000000:18:00.0 Off |                    0 |
| N/A   30C    P0            112W /  250W |     628MiB /  16384MiB |     73%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+
|   1  Tesla V100-PCIE-16GB           On  |   00000000:AF:00.0 Off |                    0 |
| N/A   23C    P0             23W /  250W |       4MiB /  16384MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI              PID   Type   Process name                      GPU Memory |
|        ID   ID                                                             Usage      |
|=========================================================================================|
|    0   N/A  N/A         3272875      C   python3                                624MiB |
+-----------------------------------------------------------------------------------------+
```

➔ In our case we only use FP32 precision (standard for deep learning), so Tensor Cores were not used

Development Mode: Jupyter Lab / Notebooks

➔ JupyterLab is a web-based interactive development environment for Jupyter notebooks, code, and data

- JupyterLab is flexible: configure and arrange the user interface to support a wide range of workflows in data science, scientific computing, and machine learning
- Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text.
- Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.
- https://jupyter.org/

Development Mode: Jupyter Lab / Notebooks

Clone the Training Repository

- Establish QBRI VPN session
- Open a raad2-gfx session (via MobaXterm or any terminal)
- git clone https://github.com/ResearchComputingTeam/GFX_Training
- cd GFX_training
- Then copy jupyter-gfxlauncher-pbs.sh to a local folder on your Windows machine using the MobaXterm SFTP sidebar (left panel of MobaXterm)

Development Mode: Jupyter Lab / Notebooks

Start Your JupyterLab Session

- Open a MobaXterm local terminal, navigate to the folder where you saved the script, make it executable and run it:
  - chmod +x jupyter-gfxlauncher-pbs.sh
  - ./jupyter-gfxlauncher-pbs.sh <your_raad2_username>
- The script will request a GPU node via PBS and start JupyterLab automatically.
- Wait until you see an SSH tunnel command and a browser URL printed in the terminal : this can take 1–2 minutes.

```
2026-03-12  12:47.33  /home/mobaxterm  ./jupyter-gfxlauncher-pbs.sh abelfer21
[INFO] Using port 59729

>>> Logging into HPC as abelfer21

>>> Will request PBS interactive session, Jupyter on port 59729
>>> When inside compute node, Jupyter will auto-start.
qsub: waiting for job 34196.raad2-cims to start
qsub: job 34196.raad2-cims ready


>>> Running on compute node: gfx1
>>> Waiting for Jupyter to start...
================================================================

>>> Open another terminal and run following command:

 ssh -L 59729:gfx1:59729 abelfer21@raad2-gfx.biolab.local

>>> Open this URL in your browser:

  http://localhost:59729/lab?token=a32927040a89934afc46200e222beb731339490241709ca2

================================================================
```

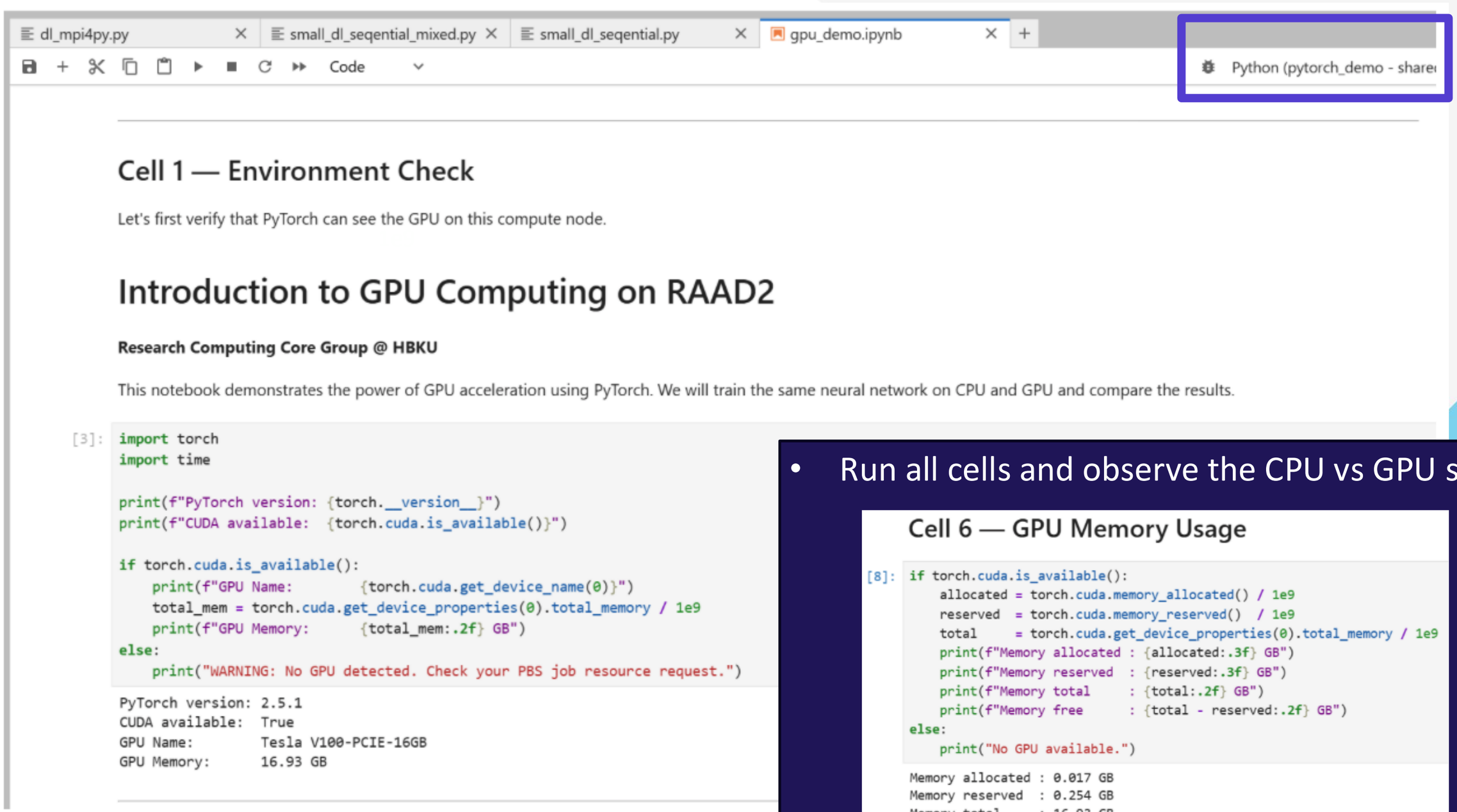
Open JupyterLab

- Open a second MobaXterm local terminal & copy-paste the SSH tunnel command printed by the script
- Copy the URL printed by the script and open it in your browser

Development Mode: Jupyter Lab / Notebooks

Run the Demo

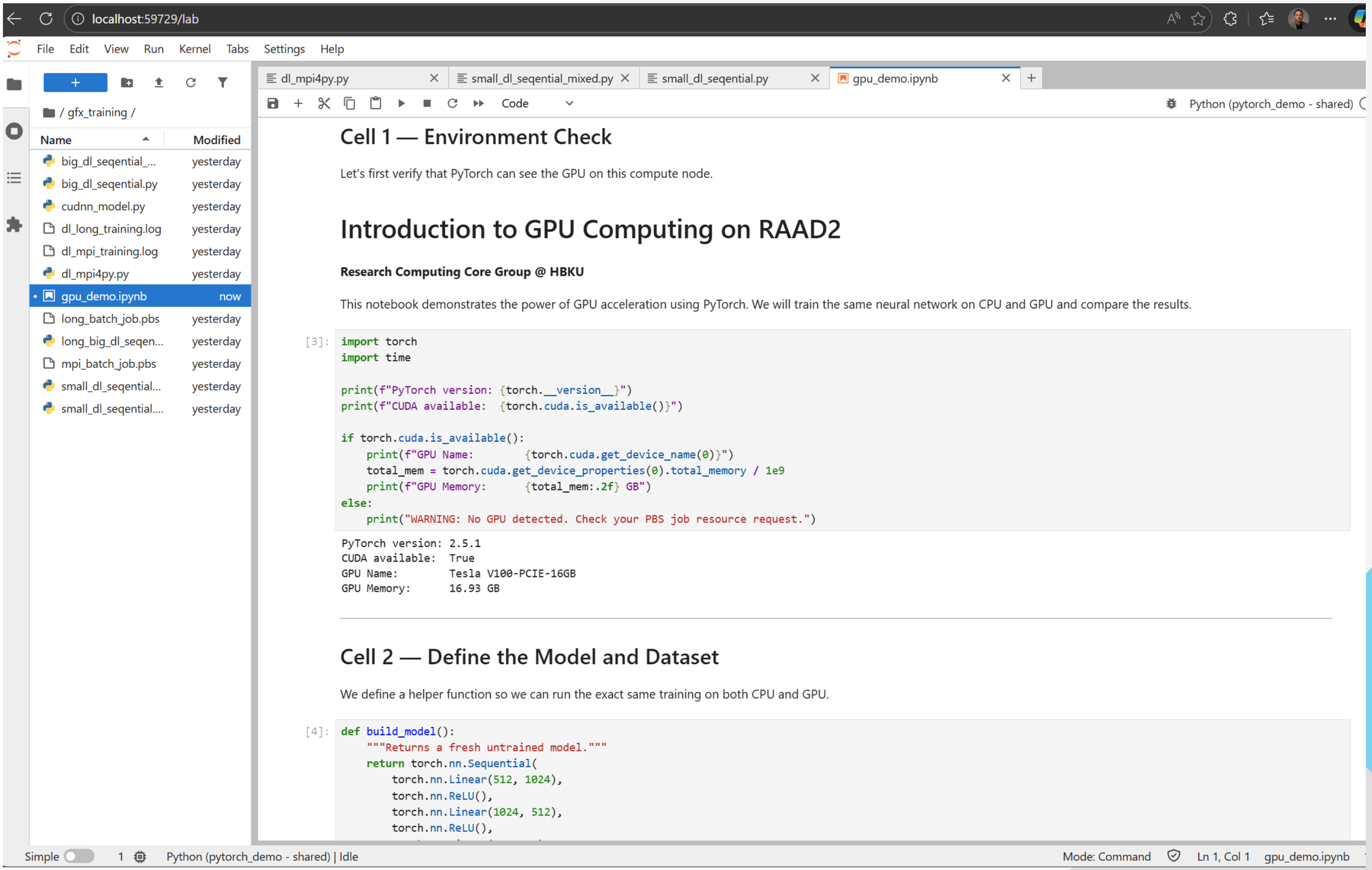
- In JupyterLab, open gpu_demo.ipynb
- Check the kernel in the top-right corner shows "Python (pytorch_demo - shared)". if not, go to Kernel → Change Kernel and select it



- Run all cells and observe the CPU vs GPU speedup

Development Mode: Jupyter Lab / Notebooks

Login → Development Mode → Production Mode



Production Mode: PBS Batch Job

Login → Development Mode → Production Mode

➔ Once you have developed and tested your code in development mode, you can submit batch jobs for longer runs

- create a PBS script (same as SLURM script but with different keywords)

```
#!/bin/bash
#PBS -N dl_long_training
#PBS -l select=1:ncpus=4:ngpus=1
#PBS -l walltime=02:00:00
#PBS -j oe
#PBS -o dl_long_training.log
#PBS -q workq

# ── Environment setup ──────────────────────────
module purge

source /cm/shared/apps/anaconda/2024.10/etc/profile.d/conda.sh
conda activate /ddn/sw/cm/shared/apps/conda/envs/pytorch_demo

# ── Move to submission directory ───────────────
cd $PBS_O_WORKDIR

# ── Run ────────────────────────────────────────
python3 long_big_dl_seqential.py
```

- Submit the script to PBS: **`qsub batch_job.pbs`**

Production Mode: PBS Batch Job

➔ In our MPI example, the 4 ranks were sharing the same single GPU : no actual HW parallelism

➔ By pinning 1 MPI rank per socket with --map-by socket, each rank owns its local GPU & CPU cores, no cross-socket memory transfers : true HW parallelism across the 2 GPUs

```
#!/bin/bash
#PBS -N dl_mpi_training
#PBS -l select=1:ncpus=36:ngpus=2
#PBS -l walltime=02:00:00
#PBS -j oe
#PBS -o dl_mpi_training.log
#PBS -q workq

# ── Environment setup ──────────────────────────
module purge
module load openmpi/gcc/64/4.1.5

source /cm/shared/apps/anaconda/2024.10/etc/profile.d/conda.sh
conda activate /ddn/sw/cm/shared/apps/conda/envs/pytorch_demo

# ── Move to submission directory ───────────────
cd $PBS_O_WORKDIR

# ── Run ────────────────────────────────────────
# 2 ranks, 1 per socket, pinned to local GPU : topology-aware placement
mpirun -np 2 --map-by socket --bind-to socket python3 dl_mpi4py.py
```

Advanced topics

Development Mode: HPC Packages

➔ A conda environment is not always self-sufficient: some Python packages require system-level libraries provided by the HPC module system

Scenario example:

- When a model or dataset does not fit on one GPU, MPI provides the comm. layer to synch gradients & model weights across multiple nodes, in our case 2 GPUs of the same node
- Neither PyTorch nor Conda can provide this

Software environment update

- We need to load the MPI library of the system: **openmpi**
- It needs to be bound to our Python script via: **mpi4py**

```
# Step 1 — system MPI (the whole point of the demo)
module load openmpi/gcc/64/4.1.5

# Step 2 — conda env
conda activate two_sdl
pip install mpi4py   # links against the system OpenMPI loaded above

# Run with 4 MPI ranks
mpirun -np 4 python3 mpi_model.py
```

Code update

- Split the dataset across 4 MPI ranks, each training independently on its slice. This introduces data parallelism at the forward/backward pass level
- Losses are aggregated at the end of each epoch via comm.reduce

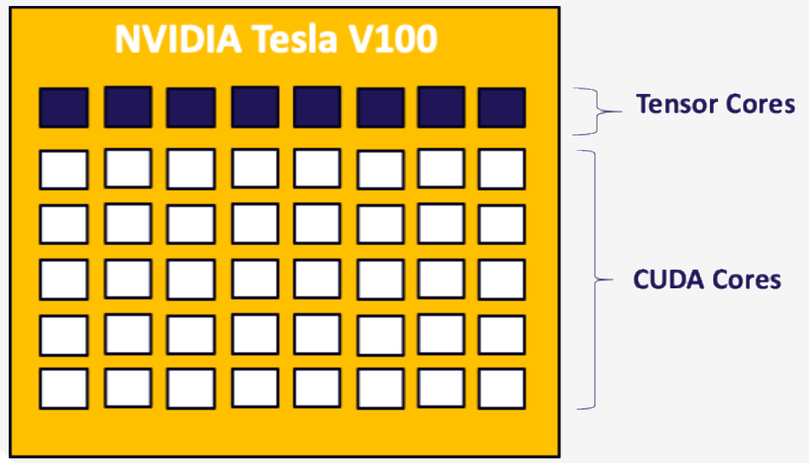
➔ launched via mpirun, not python3 (which is strictly single-machine)

Development Mode: Optimization

➔ Quick Optimization

Remember

- 32 and FP64 run on CUDA Cores
- Tensor Cores accelerates FP16 / mixed precisions



- We only use FP32 precision (standard for traditional deep learning), so Tensor Cores were not used

➔ Let's try to switch from FP32-only training to mixed precision by :

- Importing AMP (Automatic Mixed Precision) tools
  - autocast(): run parts of the model in faster FP16
  - GradScaler()
- Wrapping the forward pass in autocast(),
- Using GradScaler() to keep gradients stable,
- Letting Tensor Cores accelerate FP16 ops automatically

Result

- ~2x time speed up (but potentially less stable)

100 epochs, FP32

```
Training completed in 5.59 seconds
Average time per epoch: 0.06 seconds
```

100 epochs, FP32+FP16

```
Training completed in 2.67 seconds
Average time per epoch: 0.03 seconds
```

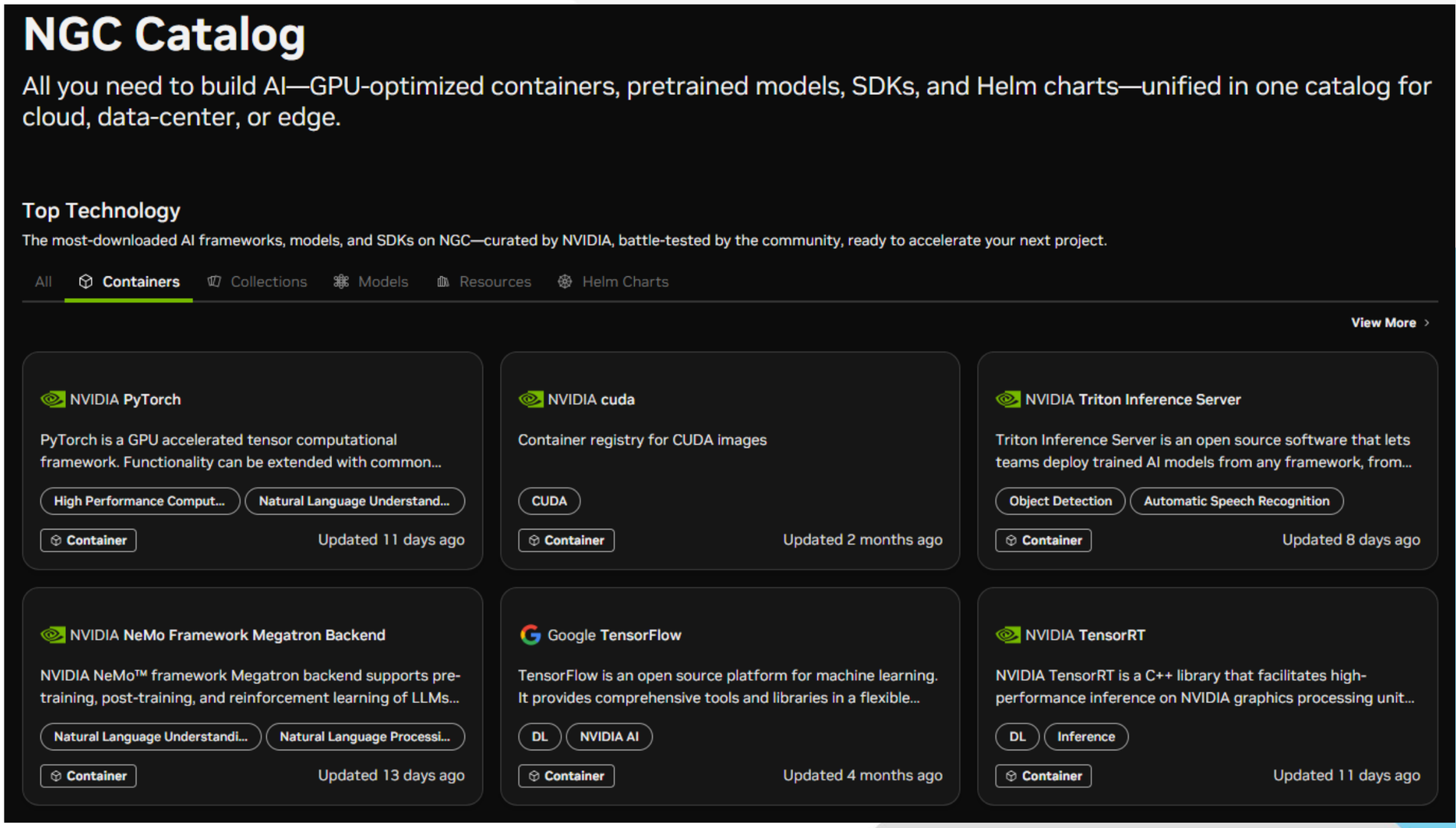
NVIDIA GPU Cloud

➔ NGC containers give you optimized, pre-built deep learning environments

https://catalog.ngc.nvidia.com/

Why NGC containers ?

- Tailored, ready-to-use, tunable, software environment
- Remove the need to manage conda environment and library compatibility



Running NVIDIA NGC containers with Singularity

➔ Singularity makes NGC containers usable on HPC where Docker is not allowed

Why Singularity (Apptainer) and not Docker ?

- Same same, but different
- Docker requires root privileges: forbidden on shared HPC systems
- Singularity main features:
  - Runs containers as a normal user
  - Integrates natively with the module system and PBS scheduler
  - Can directly mount the cluster's parallel filesystem

To go further

Research Computing docs

https://rccg.hbku.edu.qa/

Pytorch docs

https://docs.pytorch.org/docs/stable/index.html

Nvidia GPU Cloud

https://ngc.nvidia.com/

Direct Support

For any issue, question or feedback, contact the RCCG team at: **rccg@hbku.edu.qa**

Typical topics:

- GPU underutilization
- Multi-GPU training
- MPI + GPU jobs
- Performance tuning
- Software conflicts

We’re here to help you succeed