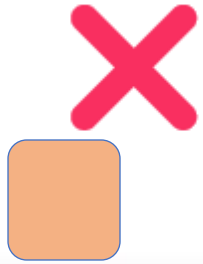
Setup Instructions

- Please ensure that you have:
  - An active HCC account and can connect to Swan
  - Login to swan-ood.unl.edu in your browser
  - Open the workshop website: https://hcc.unl.edu/hcc-ai-and-ml-workshop-nov2025
  - Open the logistics page: https://hcc.unl.edu/docs/Events/2025/mlai_nov_2025/
    - Complete the NRP setup!!

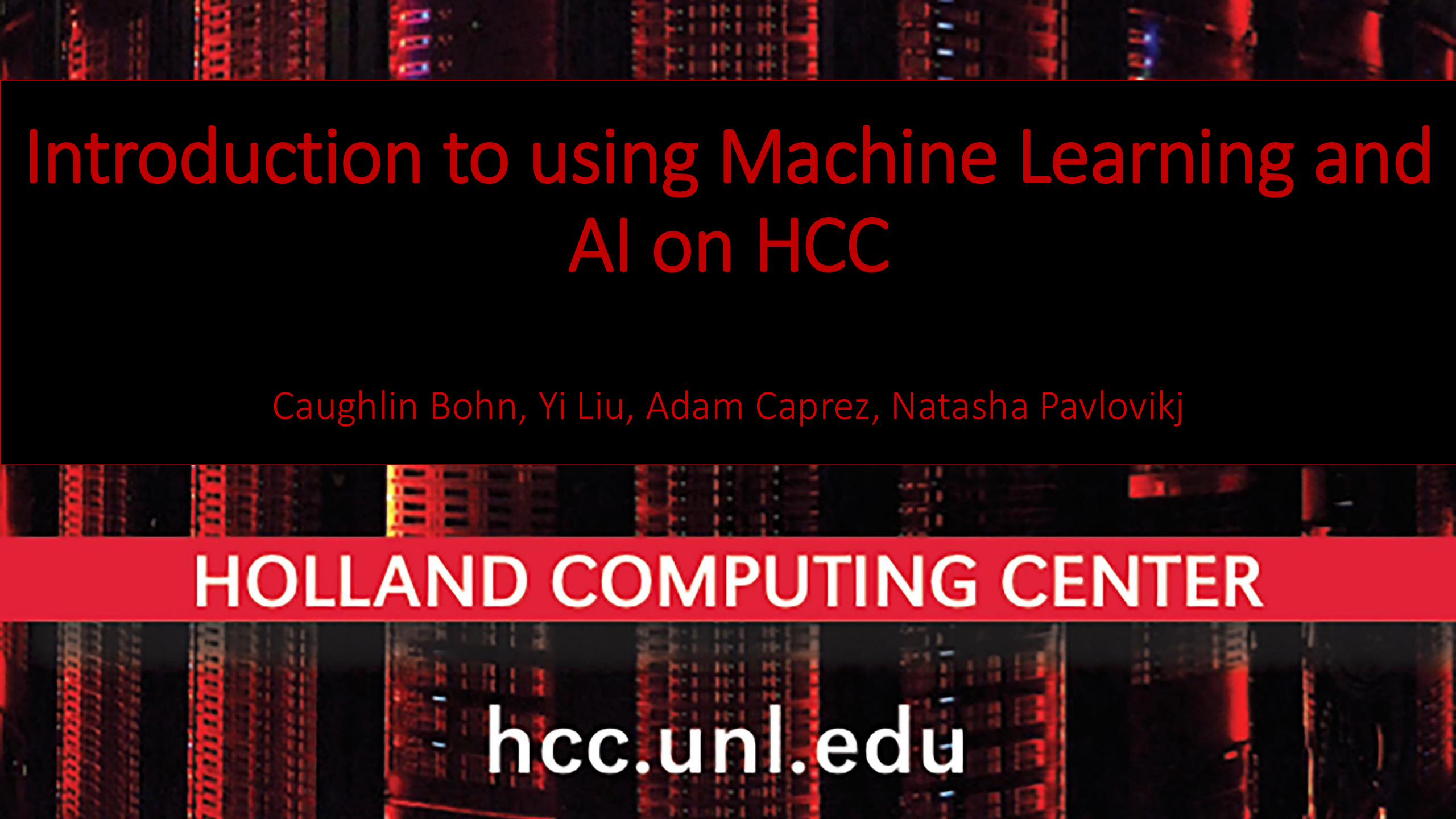
If you **are done**, please put up green check or yellow sticky note.



If you **need help**, please put a red "X" or red sticky note.

Introduction to using Machine Learning and AI on HCC

Caughlin Bohn, Yi Liu, Adam Caprez, Natasha Pavlovikj



Why use HCC for ML and AI?

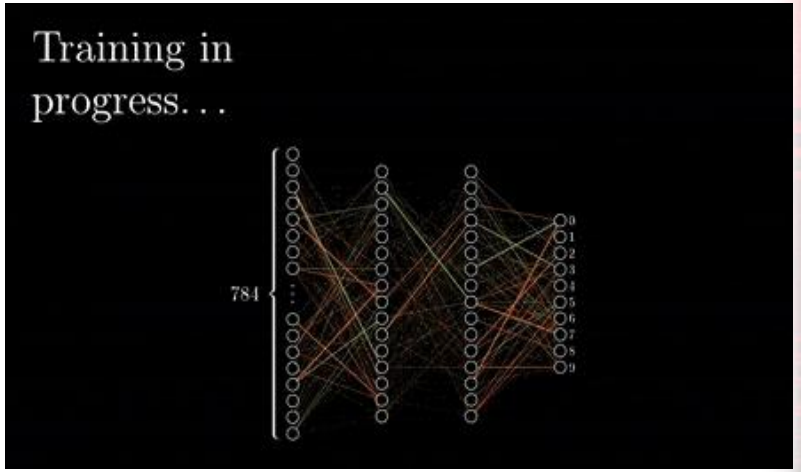
- **AI and ML workflows** are increasingly popular and require significant storage and computational resources.

- **HCC** provides the computational power and storage capacity needed to train large models quickly and efficiently—essential for many AI and ML applications!

Why use HCC for ML and AI?

Example Scenario:

Model: ResNet-50 (image classification)
Dataset: CIFAR-10 (50 k images)
Training Setup: 90 epochs, batch size ~128



Configuration	Hardware	**Approx. Training Time**
Personal Desktop	- Intel i7 CPU (4–8 cores) - 16 GB RAM - No dedicated GPU	**8–10 hours**
HPC Single GPU Node	- 1x NVIDIA V100 (16–32 GB) - Dual Xeon CPUs (56 cores) - High-speed interconnect	**~1 hour (x10 faster!)**

Introduction to HCC resources for ML and AI

Computing Resources

- Overview of HCC clusters and GPUs
- Why GPUs?
  - CPU vs. GPU for AI/ML workloads
- How to request GPUs

Software Resources

- GPU-based software
- Environment management tools
- Good practices for installing and managing dependencies

Data Storage Resources

- HCC's available data storage resources
- Good practices for data storage and file management

Job Scheduling Principles & Good Practices

Swan

- HCC's newest and fastest resource
- 330+ node Linux cluster
- 56 cores per node
- 256+ GB RAM in most nodes
- 2 nodes with 2048 GB RAM
- 5200 TB shared storage
- 4TB scratch per node.

Red

- Used to store and analyze data from the Large Hadron Collider CMS experiment
  ~600 node Linux cluster
- ~10,000 cores
- 18 PB raw storage

Total Resources

almost 28k cores
~35 PetaBytes of storage
64 GB to 2 TB memory per node
GPUs: 200+ GPUs

NRDStor – Now Available

- Used to store experiment data for researchers in an easy-to-access format.
- 42 storage nodes
- 8.3 PB of storage

GPUs on Swan

Swan has a large variety of GPUs available for researchers! Swan contains 210 GPUs across 12 models.

- 4× A100 (80 GB)
- 14× A30 (24 GB)
- 8× L40S (48 GB)
- 3× H100 (96 GB)
- 4× V100 (16 GB)
- 62× V100S (32 GB)
- 24× T4 (16 GB)
- 7× RTX 5000 (16 GB)
- 2× RTX 8000 (48 GB)
- 24× A6000 (48 GB)
- **6× NVIDIA H200 NVL (141 GB)**
- **52× NVIDIA L40S (48 GB)**

To add GPU constraint, use SLURM command:

```
--constraint=gpu_v100
```

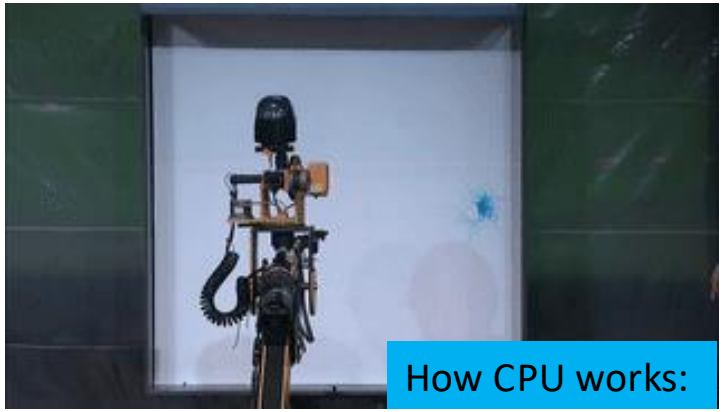
https://hcc.unl.edu/docs/submitting_jobs/submitting_gpu_jobs/

Why GPUs?

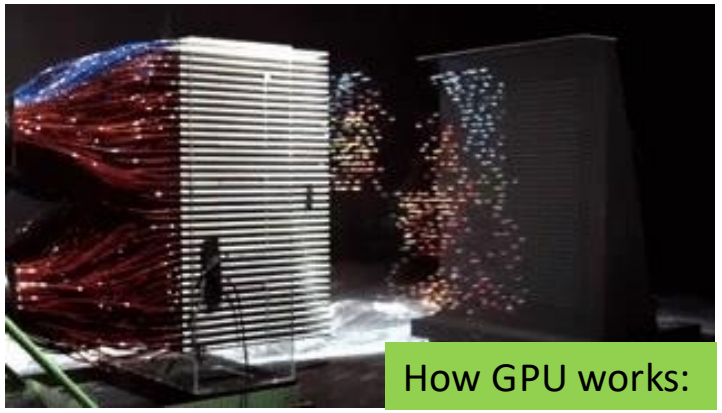
Parallelism - GPUs are designed for massive parallel workloads, ideal for tasks broken into smaller, independent operations.

Efficiency - GPUs perform the same operation on many data points simultaneously, making them highly efficient for computations like matrix multiplication, FFTs, etc.

High Throughput - With a large number of cores and high memory bandwidth, GPUs can process substantial data volumes at once, accelerating simulations and model training.



How CPU works:



How GPU works:

https://www.youtube.com/watch?v=-P28LKWTzrI

How to Request GPUs

- **SLURM options:**

  `--gres=gpu:<count>`

  `--gpus=<count>`

- **Partitions dedicated for GPU use**

  Standard GPU partition: `--partition=gpu`

- **Software & Modules**

  CUDA toolkit, TensorFlow, PyTorch...

  available via `module load`

https://hcc.unl.edu/docs/submitting_jobs/submitting_gpu_jobs/

How to Submit a GPU Job

```
#!/bin/bash
#SBATCH --time=03:15:00
#SBATCH --mem-per-cpu=1024
#SBATCH --partition=gpu
#SBATCH --gres=gpu
#SBATCH --job-name=example
#SBATCH --error=/work/[groupname]/[username]/job.%J.err
#SBATCH --output=/work/[groupname]/[username]/job.%J.out

module load cuda

./cuda-app.exe
```

To submit a GPU-enabled job, two lines need added to your submit file.

The first is to select a GPU-enabled partition such as ‘gpu’ or ‘guest_gpu’.

The second is to request the general resource of ‘gpu’.

Multiple GPUs can be requested using a value of ‘gpu:#’. For example, 2 GPUs would be ‘gpu:2’

https://hcc.unl.edu/docs/submitting_jobs/submitting_gpu_jobs/

Additional HCC GPU Resources

Using the Guest GPU Partition (`guest_gpu`)

- The **guest_gpu** partition is pre-emptible.
- Use **SLURM options**:

```
#SBATCH --partition=guest_gpu
#SBATCH --gres=gpu:<count>
```

- This is useful for exploratory work, testing, or workloads with flexible GPU needs.

https://hcc.unl.edu/docs/submitting_jobs/submitting_gpu_jobs/

Available Nodes in Each Partition

Partition	Owner	Node total	SLURM Specification
batch	Shared	104	#SBATCH --partition=batch
gpu	Shared	39	#SBATCH --partition=gpu
guest_gpu	Shared	26	#SBATCH --partition=guest_gpu
guest	Shared*	*187*	#SBATCH --partition=guest
labname	*Lab Group*	...	*#SBATCH --partition=labname*

For more detailed information see:
https://hcc.unl.edu/docs/submitting_jobs/partitions/swan_available_partitions/

GPU-Based Software

System-Wide Modules:

Load popular frameworks (e.g., TensorFlow, PyTorch) with `module load …`

Custom Conda GPU Environments:

Lets you install specialized packages/frameworks in an isolated environment tailored to GPU usage.

Docker/Apptainer Images:

Use GPU-enabled Docker images via Apptainer (either HCC-provided or custom-built).
Ensures consistency and reproducibility across different systems.

Monitoring GPU Jobs:

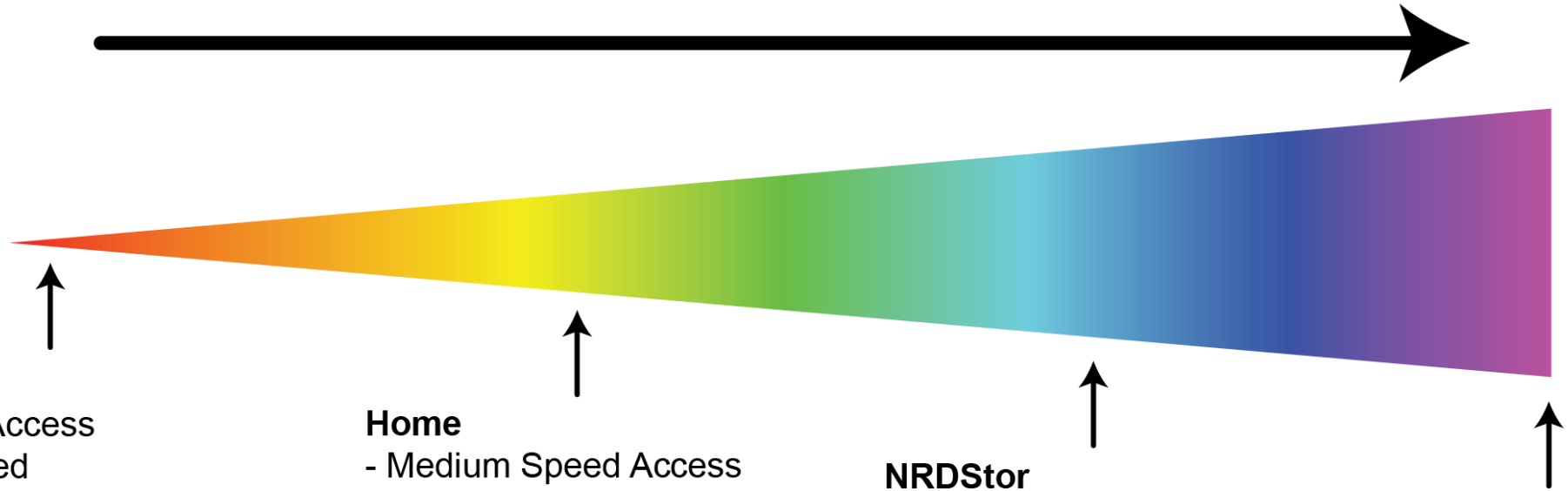
```
srun --jobid=<JOB_ID> --pty bash
```

Reference: https://hcc.unl.edu/docs/submitting_jobs/monitoring_gpu_usage/

Keep an eye on resource utilization (e.g., GPU memory, GPU load) to optimize performance and troubleshoot issues.

Home vs Work vs NRDStor vs Attic

- Higher Storage Performance to Jobs
- Greater Ease of Access
- Diminishing Resiliency of Files



Attic
- Globus Access
- Fee based
- Not mounted on computational resources
- Off-site Backups

Home
- Medium Speed Access
- 20 GB per User
- Backed Up
- Not for direct job use

NRDStor
- Fast Access
- Accessible on laptop
- 50 TB per Group
- 5 Million files per Group
- Additional storage upon request
- Not backed up

Work
- Fastest Access
- No paid access
- 100 TB per Group
- 5 Million files per Group
- Not backed up
- Inactive files will be purged

Good Practices for Data Storage

- **Back up your data regularly!**
  - e.g. copy important files from $WORK to $NRDSTOR, your local machine, or external drives (use Globus or *scp* for efficient transfer)
    - Utilize cloud storage such as OneDrive provided by the University of Nebraska
    - Leverage HCC's Attic near-line archive service

- **Keep critical files in $HOME**
  - store vital small files (scripts, source code, etc.) in your $HOME directory, which is automatically backed up daily
  - *(but avoid cluttering home with large data)*

- **Use version control**
  - manage code and documents with Git (preserve revision history and off-site copies; UNL GitLab or GitHub can host your repositories)

Job Submission Principles and Good Practices

1. Use the worker nodes (SLURM)

- All heavy workloads must be submitted to the worker nodes using the SLURM job scheduler.
- **Do not run workflows on login nodes**—those are for editing/submitting only!

2. Resource Considerations

- Your code must explicitly support GPUs—requesting a GPU does not automatically provide performance gain.
- If you want to use multiple worker nodes for a single job, your code needs to explicitly support
- CPU-based code will **not** run faster on GPUs; in fact, using a GPU for CPU-only code may slow things down.

3. Batch Job Submission

- Prepare a job script (specify CPU cores, memory, runtime, GPU needs, etc.) and submit with `sbatch`.
- SLURM will queue your job and dispatch it to a suitable worker node (or GPU node, if requested).

4. Interactive Jobs

- For debugging or exploratory work, launch an interactive SLURM session (e.g., `srun`) or use HCC's OnDemand web portal.
- Let's you run commands in real time within a temporary compute allocation.

Job Submission Principles and Good Practices

5. Efficient & Fair Scheduling

- SLURM dispatches jobs to worker nodes when resources are available, ensuring high utilization and fair sharing (so no single user dominates resources).

6. Job Monitoring & Management

- Track jobs with `squeue` for queue status and `sacct` for job history. Cancel jobs using `scancel`.
- **HCC OnDemand** provides a GUI to view and manage active or completed jobs.

https://hcc.unl.edu/docs/submitting_jobs/monitoring_gpu_usage/