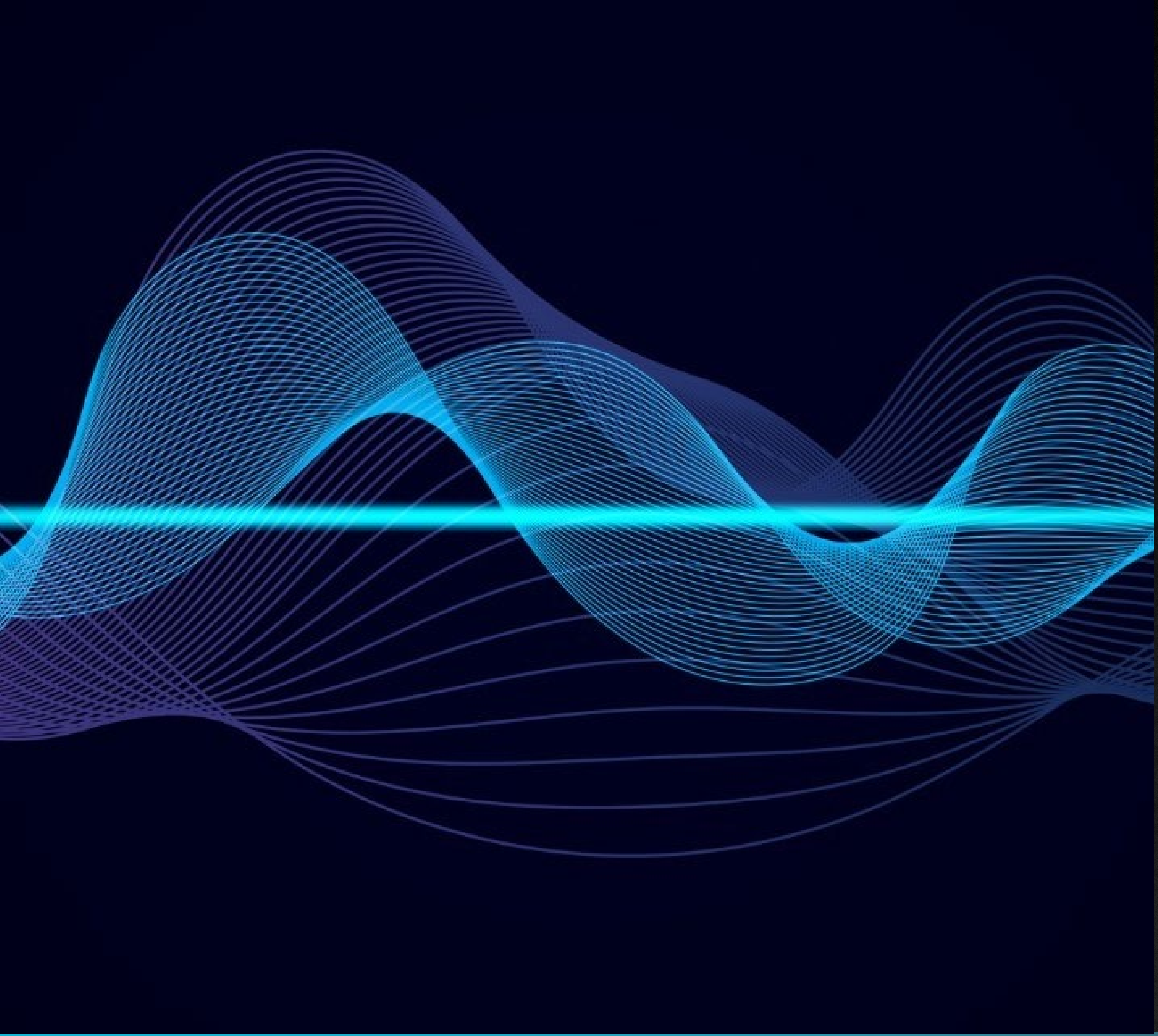


Computational Physics

PHYS 6260

HPC Capabilities

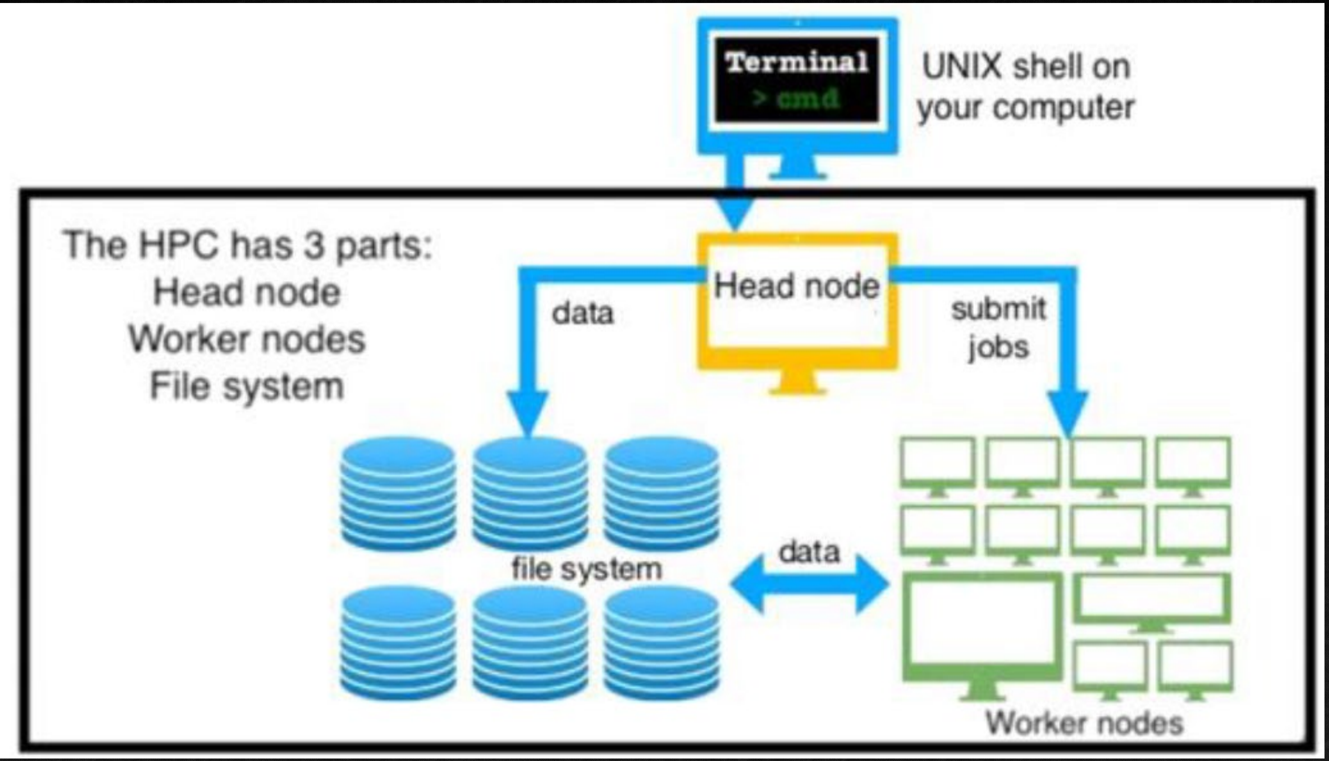
Announcements:

- Project proposal, Due Friday 3/7

Basics of HPC

- HPC technology that uses clusters of powerful machines working in parallel to process large multi-D data and solve complex problems at scale

- Typically increases speed based on the number of cores you use
- The largest HPC systems have upwards of 1 million cores
- Allows us to tack problems that would ordinarily be difficult or impossible to solve on a single computer

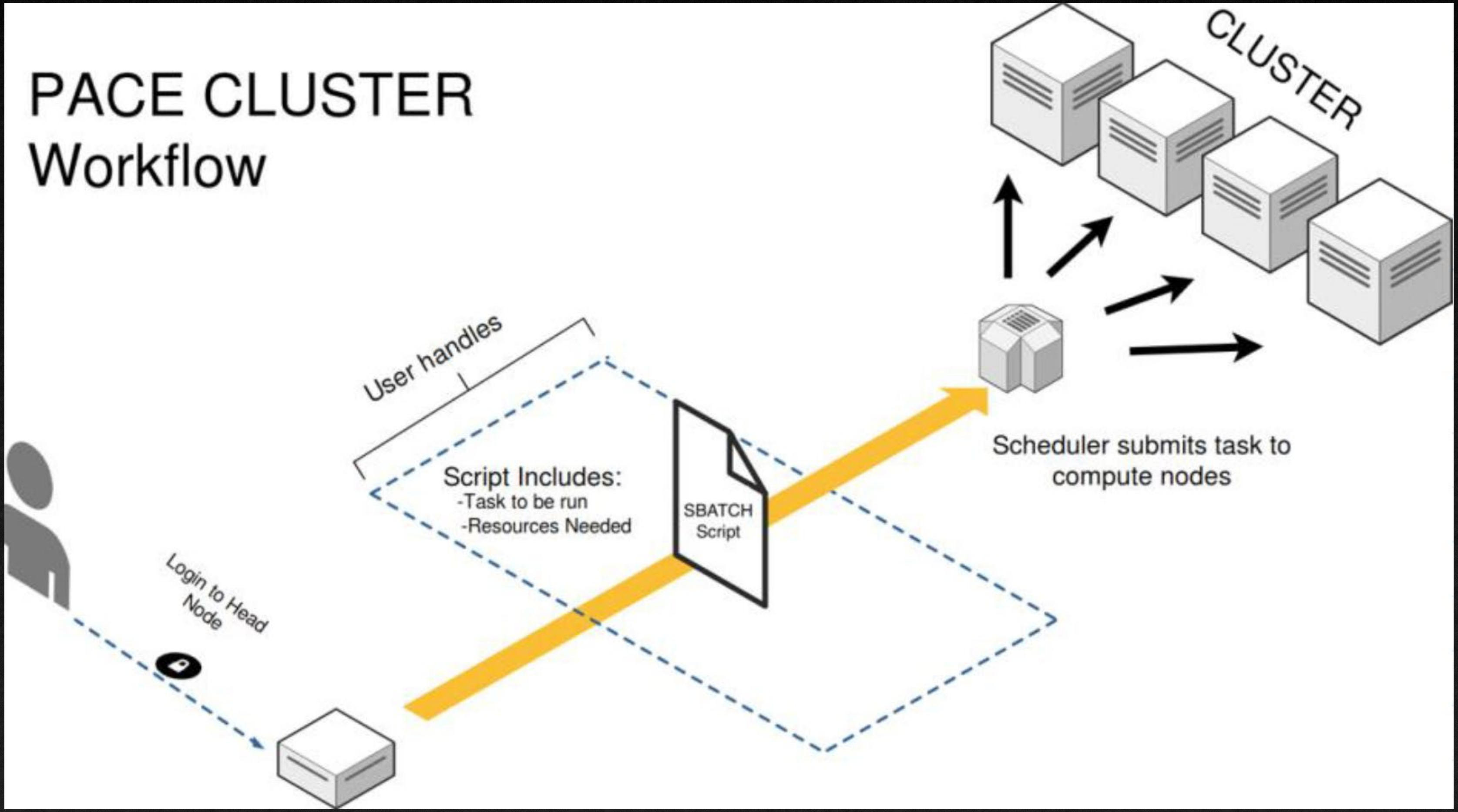


Basics of HPC

- Head versus compute nodes
- Head node: the machine that you log into
  - Shared resource
  - Use for editing code and data management
  - Not good or prohibited for computations
- Compute node: machines that run calculations
  - No direct access by users outside of a compute job
  - Allocated per-job by scheduler

Introduction to PACE

- Multiple clusters:
  - **ICE***
  - Phoenix
  - Hive
  - Firebird

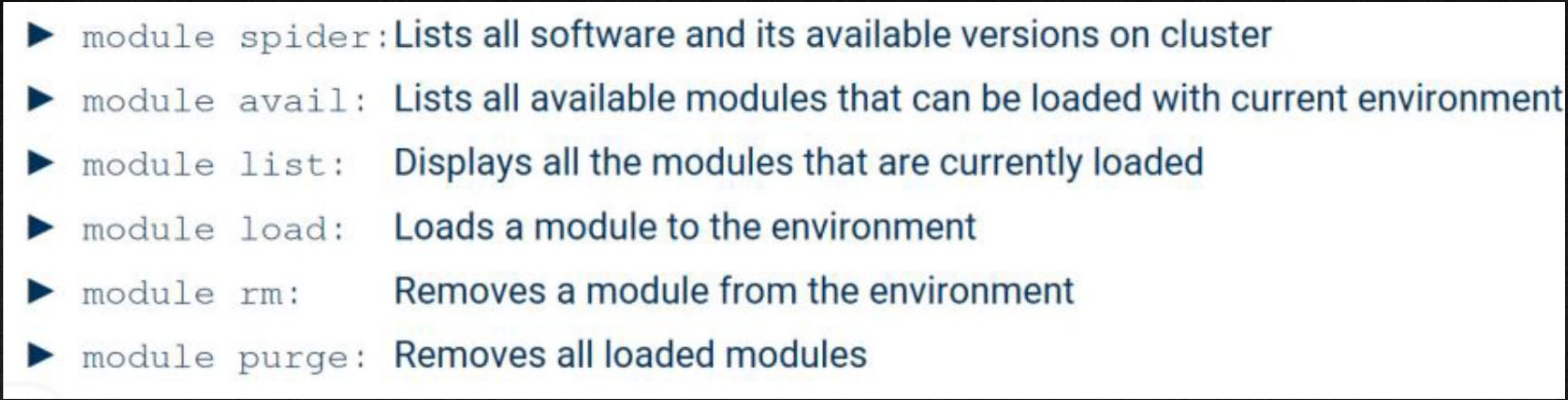


Introduction to PACE: Storage

- Data is accessible from all head and compute nodes
- Three storage directories
  - Home: 10GB on Phoenix & Hive; 15GB on ICE
  - Project storage: limit depends on the amount purchased by your PI. Not available on ICE
  - Scratch: 15 TB (Phoenix), 1TB (Hive), 100GB (ICE); subject to deletion after 60 days
- Use Globus to transfer data between computers, e.g. local machine and PACE and/or other HPC systems

Introduction to PACE: Modules

- PACE and HPC systems in general use a module system in which you can load various software packages
- There are defaults and some (especially MPI) have been optimized for the particular system



▶ `module spider:`	Lists all software and its available versions on cluster
▶ `module avail:`	Lists all available modules that can be loaded with current environment
▶ `module list:`	Displays all the modules that are currently loaded
▶ `module load:`	Loads a module to the environment
▶ `module rm:`	Removes a module from the environment
▶ `module purge:`	Removes all loaded modules

Log into PACE on ICE Cluster & Run Code

1. Connect to GT VPN
2. Open VSCode and install the Remote SSH extension if you haven’t done so already
3. Add a new SSH connection

   e.g. `ssh <GT_user_ID>@login-ice.pace.gatech.edu`

4. Load the python module using `module load anaconda3`
5. Untar folder w/ code from my folder to your local folder

   e.g. `tgz xf ~jw254/phys6260/16_InClass.tgz`

6. Open code: `code heat_code.py`
7. Run code on compute node

Introduction to PACE: Scheduler

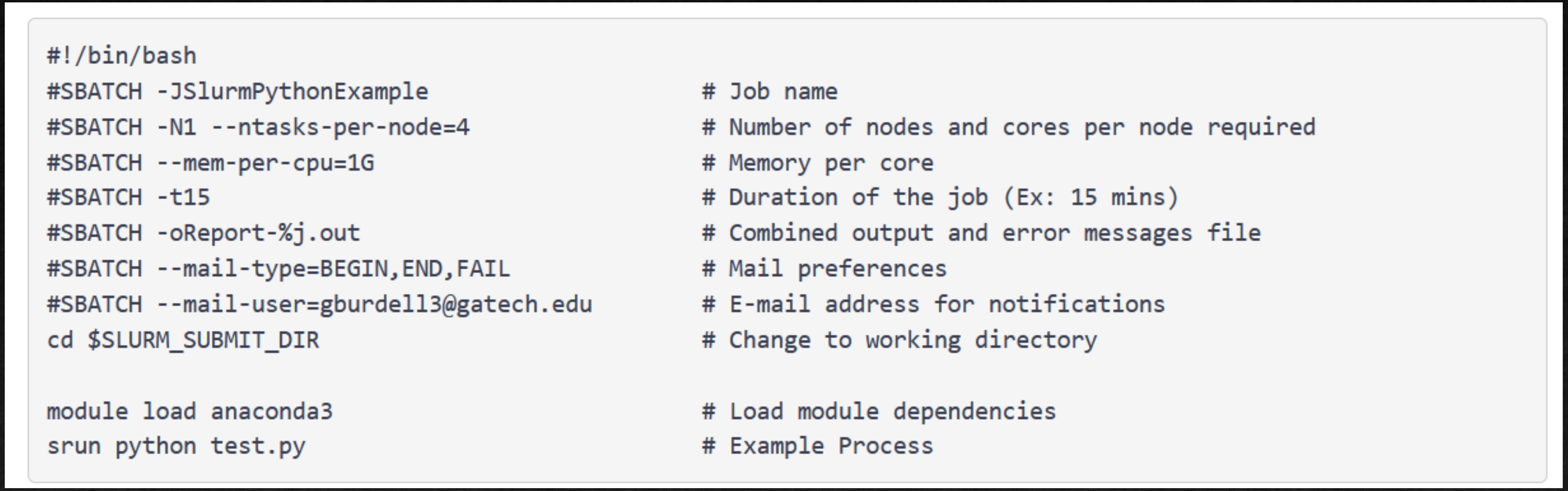
- Users submit batch jobs (non-interactively) to the scheduler
- Scheduler stores the batch job, evaluates the resource requirements and priorities, and then distributes it to the compute nodes
- A few different popular schedulers: SLURM*, PBS, Torque
- Information needed to write a batch script for SLURM
  - Number of nodes (--nodes or -N)
  - Cores per node (--ntasks-per-node or -n). Can omit if unsure
  - Memory per core (--mem-per-cpu or –m). Depends on program
  - GPU (--gres). Only use if job specifically uses GPUs

Introduction to PACE: Command cheat sheet

- pace-quota: check available storage / charge account
- pace-check-queue ice-cpu –c: PACE compute node availability
- pace-job-summary <job #>: Overview of job
- salloc –N1 –ntasks-per-node=#: interactive job
- srun: run script
- sbatch: submits batch job
- squeue –A <GT-username>: check on queue status

Write SLURM Script to run on Compute Node

1. Create a sbatch file (w/ extension of .sh) to run on SLURM compute nodes
2. Follow the format of this to write your SLURM script:



```
#!/bin/bash
#SBATCH -JSlurmPythonExample                    # Job name
#SBATCH -N1 --ntasks-per-node=4                 # Number of nodes and cores per node required
#SBATCH --mem-per-cpu=1G                        # Memory per core
#SBATCH -t15                                    # Duration of the job (Ex: 15 mins)
#SBATCH -oReport-%j.out                         # Combined output and error messages file
#SBATCH --mail-type=BEGIN,END,FAIL              # Mail preferences
#SBATCH --mail-user=gburdell3@gatech.edu        # E-mail address for notifications
cd $SLURM_SUBMIT_DIR                            # Change to working directory

module load anaconda3                           # Load module dependencies
srun python test.py                             # Example Process
```

★ Write log file to the output_logs folder

3. Submit your job using sbatch <SLURM-script-name>.sh
4. Open the output files: less output_logs/<log-file-name>.log
   code –r output_plots/config_number_1.png

SLURM script to execute code

```
#!/bin/bash
#SBATCH --job-name=2DHeatExample               # Job name
#SBATCH --mail-user=<GT_user_ID>@gatech.edu # E-mail address for notifications
#SBATCH --mail-type=BEGIN,END,FAIL             # Mail preferences
#SBATCH --nodes=1                              # Use one node
#SBATCH --ntasks-per-node=1                    # Number of tasks per node
#SBATCH --mem-per-cpu=1gb                      # Memory per processor
#SBATCH --time=00:30:00                        # Time limit hrs:min:sec
#SBATCH --output=output_logs/test_one.out    # Standard output and error log

# Load module dependencies
module load python

# run two configurations
python heat_code.py 1

date
```

GPU Example on PACE

[Builds on relaxation example from last class]

Run relaxation script <u>with</u> GPU:

1. Open gpu_example from 16_InClass folder
2. Load nvhpc module (automatically purges all default modules)
3. Compile the c code using ./15_compile.sh
4. Launch a compute node w/ GPU: salloc –N1 --gres=gpu
5. Run accelerated code using ./15_relax_acc.c

Run relaxation script <u>without</u> GPU:

1. Open a new terminal (where the default modules have not been purged)
2. Compile code w/ openMPI: gcc –O3 -fopenmp 15_relax.c -lm –g –o 15_relax_omp
3. Launch a compute node w/ 6 tasks: salloc –N1 --ntasks-per-node=6
4. Export the number of thread environment variable: export OMP_NUM_THREADS=6
5. Run the compiled code w/o GPU using ./15_relax_omp.c

What is the time difference?