

# Introduction to HPC at the UNM Center for Advanced Research Computing

Matthew Fricke ([mfricke@unm.edu](mailto:mfricke@unm.edu))

Research Assistant Professor

<http://unm.edu/~mfricke>

Please send corrections to me at [mfricke@unm.edu](mailto:mfricke@unm.edu).

Version 1.5.

# Why we are here!

- You have a computational task of some sort
- Most people who come to CARC fall into these categories
  - ... for a publication you are working on
  - ... an upcoming class assignment
  - ... analysis of data for a government agency
- The common feature is that they require more compute/memory/storage resources than is commonly available.

# Why we are here!

- To use the resources at CARC effectively there are things you have to learn! Some are commonplace but some are very specific to HPC and CARC.

By the end of the day you will know how to:

- Request time compute time on the clusters (PBS scripting)
- Interact with contained environments (Anaconda, Modules, Singularity)
- Get output from your compute jobs
- Use storage appropriately (scratch vs user storage)
- Write scripts to run embarrassingly parallel tasks
- Write and run a simple MPI program for tightly coupled tasks
- Ultimately get a huge increase in the computing power you can apply.

## 1.1 Changelog

- Corrected various typos
- Changed font to make code clearer
- Reordered MAUI/Torque slides
- Corrected “conda list” to “conda env list”
- Added –machinefile \$PBS\_NODEFILE to mpirun example
- Added slide on torque queue status commands
- Modified pbs examples for wheeler instead of galles

## 1.2

- Specify the debug queue in the examples
- Ask for 8 cores not 2 since we are using wheeler/wheelie for workshops

## 1.3

- Typo “-l” should be “-L”

## 1.4

- Restructured examples around python program to calculate  $\pi$

# Outline

- What High Performance Computing (HPC) and the Center for Advanced Research Computing all about
- Accessing your account and transferring files
- Some useful Linux commands
- Software environments
- A short python program to calculate  $\pi$
- 15 min break

# Outline

- Tour
- Submitting compute jobs at CARC
- Parallel Jobs
- GNU Parallel
- 15 min break
- JupyterHub
- Message Passing Interface

# What is High Performance Computing?

## Scaling up

- NVIDIA DGX-2H (\$400,000 each, 81k CUDA cores, 10240 tensor cores)

[https://www.nvidia.com/content/dam/en-zz/es\\_em/Solutions/Data-Center/dgx-2/nvidia-dgx-2h-datasheet.pdf](https://www.nvidia.com/content/dam/en-zz/es_em/Solutions/Data-Center/dgx-2/nvidia-dgx-2h-datasheet.pdf)



## Scaling out

- Stampede 2
- <https://www.tacc.utexas.edu/systems/stampede2>
- \$30,000,000, 285,000 CPUs



# Why it Matters to You

- Grant and publication reviewers know about these systems so there are no excuses for small sample sizes.
- Machine Learning is showing up everywhere from cosmology to firefighting. Machine Learning requires enormous resources to process huge datasets.
- `ssh username@wheeler.alliance.unm.edu`

# The Center for Advanced Research Computing's Mission

The UNM Center for Advanced Research Computing is the hub of computational research at UNM and one of the largest computing centers in the State of New Mexico. It is an interdisciplinary community that uses computational resources to create new research insights. The goal is to lead and grow the computational research community at UNM.

CARC provides not just the computing resources but also the expertise and support to help the university's researchers. This service is available to faculty, staff, and student researchers free of charge through support from the UNM Office of the Vice President for Research.

<http://carc.unm.edu>



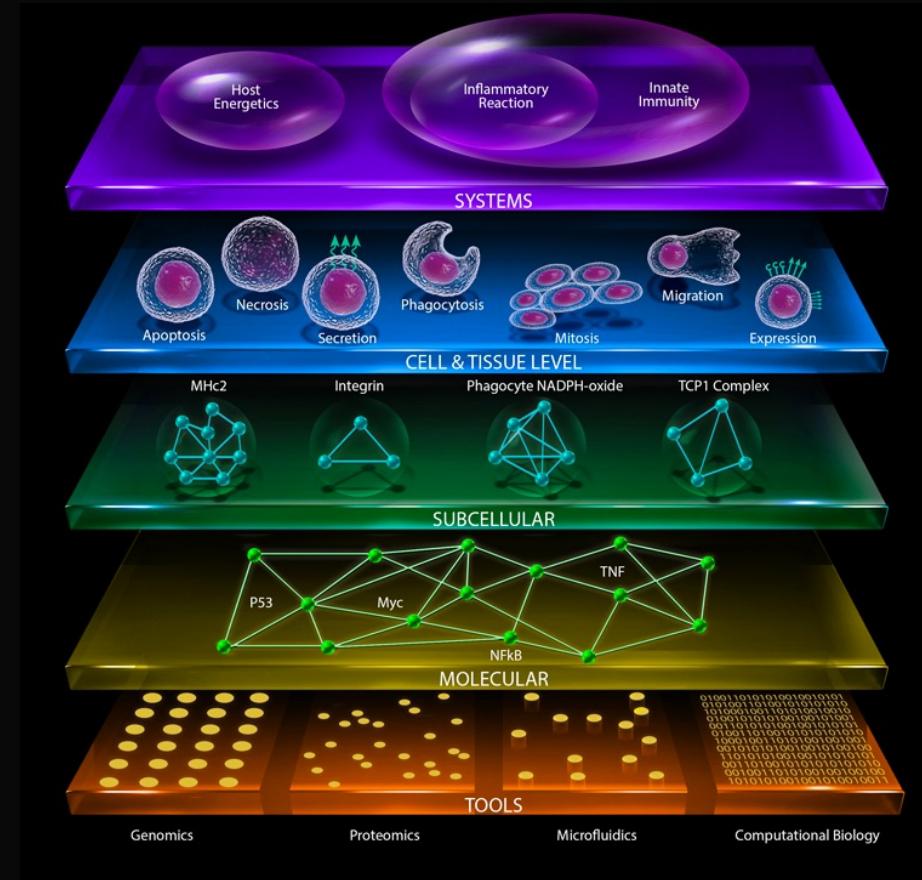
# Big Data and Machine Learning

- Machine learning needs lots of everything.
- The current revolution in convolutional neural networks (Think Google, Self-driving cars, etc) is due to algorithms rooted in the 1960s being given huge training sets.
- Most of machine learning comes down to floating point matrix and vector operations. GPUs excel at those operations and are orders of magnitude faster than CPUs.
- Xena has dual Nvidia Tesla K40M GPUs for this purpose.



# Biology

- Computational biology memory usage increases with input sizes. Rapid genotyping tools generate sequences faster and faster.
- Hundreds of GB of RAM are becoming a normal requirement to complete these calculations.
- The Taos cluster is dedicated to computational biology and has 440 CPUs and 300 GB per node.
- Xena has 3 TB RAM nodes.



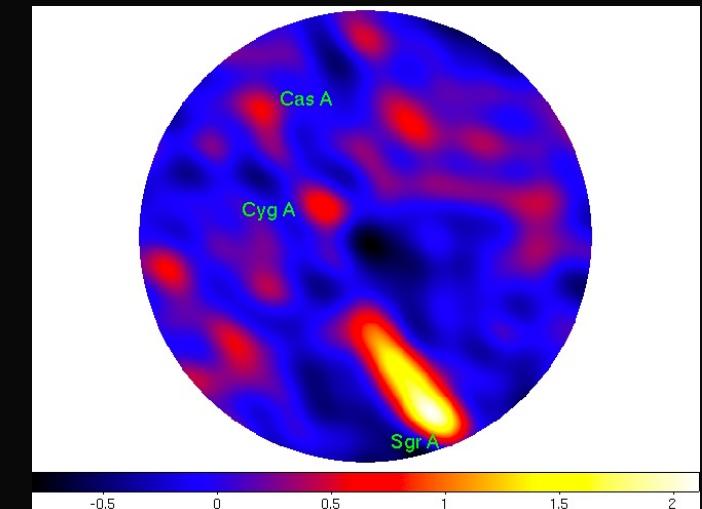
- Pandemic flu modelling
- Tuberculosis antibiotic resistance
- Pacific island bird genetics
- Intra-species viral spread
- NM Tree species mapping from LASER scans

# Physics

- Wheeler is a general purpose scale-out machine used by biophysicists, cosmologists, and many others.
- Gibbs is primarily used by computational chemists.



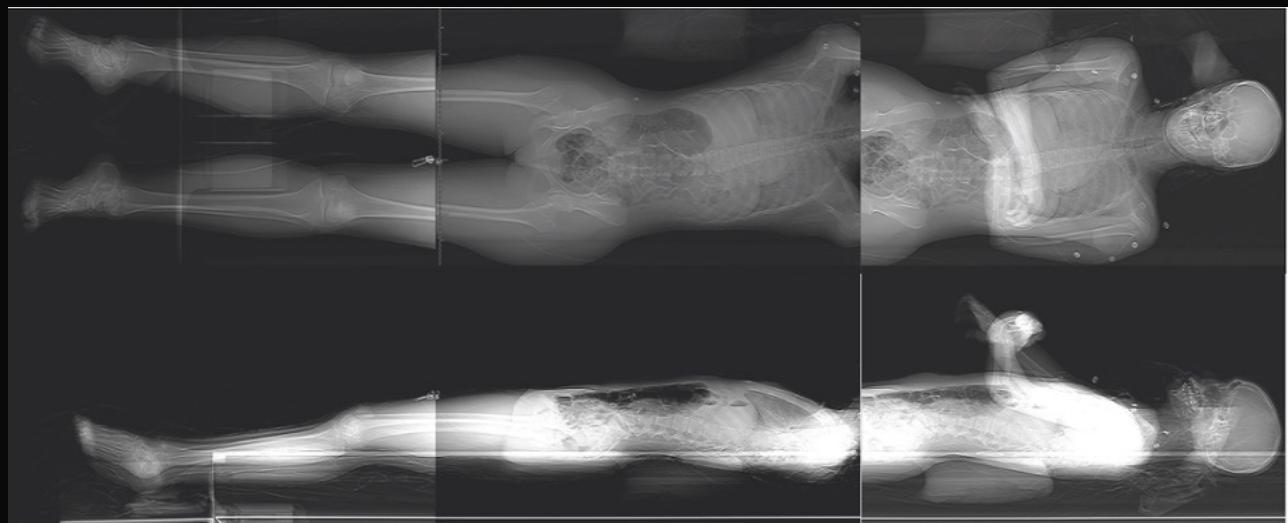
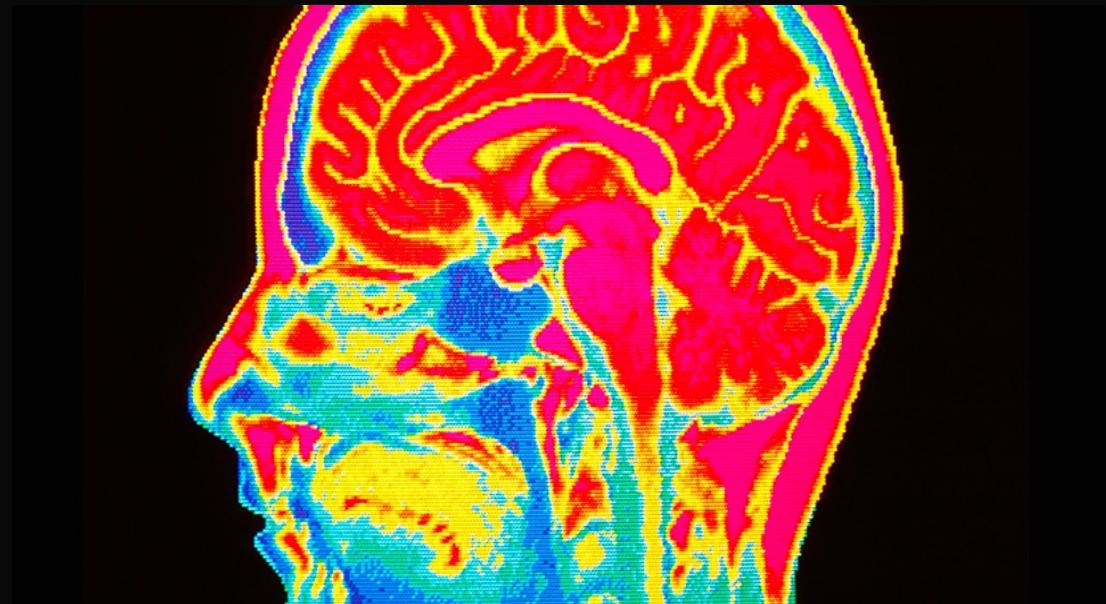
Molecular simulation of new photovoltaic materials



Long Wavelength Array Radio Telescope  
Data Processing on Wheeler

# When users need a lot of network bandwidth or storage

- Lots of data comes in the form of large images
  - Largest online pathology database (15k people, 15k image each)
  - MRI and fMRI image databases
- 
- Specialized Network Access
    - Mapping the Great Firewall





# Albuquerque Integrated Reporting System

<a href="#">2016101</a>	<a href="#">Finding approximate symmetries in graphs</a>	Fricke, Matthew
<a href="#">2016100</a>	<a href="#">Simulation of impact on armor plate using Velodyne</a>	Hogeveen, Jeremy P
<a href="#">2016099</a>	<a href="#">Cyber-infrastructure Performance Modeling</a>	Christodoulou, Christos
<a href="#">2016098</a>	<a href="#">Effective refinement of protein structure</a>	ANDEROGLU, Osman
<a href="#">2016097</a>	<a href="#">LANL HIGRAD</a>	Sorrentino, Francesco
<a href="#">2016096</a>	<a href="#">Coarse-grained modeling of biomolecules</a>	Shen, Yu-Lin
<a href="#">2016095</a>	<a href="#">Resilient Composites</a>	Bridges, Patrick G
<a href="#">2016093</a>	<a href="#">Computational Investigation of Ru Photochromes</a>	Nishima, Wataru
<a href="#">2016090</a>	<a href="#">Reinforcement learning neuropathologies underlying psychiatric sequelae in Traumatic Brain Injury</a>	Poroseva, Svetlana
<a href="#">2016087</a>	<a href="#">Driving forces and dynamics between small molecules and amyloid- A<math>\beta</math> aggregates</a>	He, Yi
<a href="#">2016086</a>	<a href="#">Polyurethane Foam Modeling</a>	Taha, Mahmoud Reda
<a href="#">2016084</a>	<a href="#">An Interval Multi-level Monte Carlo Method for Reliability Analysis of Imprecise Probabilistic Systems</a>	Rack, Jeffrey J
<a href="#">2016083</a>	<a href="#">Modeling Immune System Cell Search Processes</a>	Hogeveen, Jeremy P
<a href="#">2016081</a>	<a href="#">RNA transcriptome sequencing of T-cells exposed to Uranium and Arsenic</a>	Chi, Eva Y
<a href="#">2016080</a>	<a href="#">Predicting Progression to Alzheimer's Disease</a>	Tjiptowidjojo, Kristianto
<a href="#">2016079</a>	<a href="#">Relating plant traits to biomass dynamics in New Mexico aridlands</a>	Motamed, Mohammad
<a href="#">2016078</a>	<a href="#">Statistical methods for investigating large scale gene environment interaction</a>	Fricke, Matthew
<a href="#">2016077</a>	<a href="#">TB Genomic Analysis</a>	Schilz, Jodi R
<a href="#">2016076</a>	<a href="#">LiDAR-based tree identification in Northern New Mexico</a>	Calhoun, Vince
		Whitney, Kenneth
		Luo, Li
		Wearing, Helen

<a href="#"><u>2016074</u></a>	<a href="#"><u>Implicit Monte Carlo Simulation of Thermal Radiation Transport</u></a>	Prinja, Anil K.
<a href="#"><u>2016073</u></a>	<a href="#"><u>Implicit Monte Carlo Simulation of Transport Phenomena</u></a>	Prinja, Anil K.
<a href="#"><u>2016072</u></a>	<a href="#"><u>simulations for LEGEND neutrino-less double beta decay experiment</u></a>	Gold, Michael S
<a href="#"><u>2016070</u></a>	<a href="#"><u>Transcriptomic Analysis of Drosophila Neuroblasts</u></a>	Johnston, Christopher
<a href="#"><u>2016068</u></a>	<a href="#"><u>Simulation of quantum many-body systems</u></a>	Miyake, Akimasa
<a href="#"><u>2016066</u></a>	<a href="#"><u>microRNA and epigenetic control on gene expression</u></a>	Liang, Fu-Sen
<a href="#"><u>2016063</u></a>	<a href="#"><u>Genomic Data Analysis</u></a>	Guo, Yan
<a href="#"><u>2016062</u></a>	<a href="#"><u>Simulation of terawatt x-ray free electron lasers</u></a>	Freund, Henry
<a href="#"><u>2016061</u></a>	<a href="#"><u>Evolutionary convergence in mainland and island lizards</u></a>	Poe, Steve
<a href="#"><u>2016059</u></a>	<a href="#"><u>Curucmin modulation of amyloid-beta peptide interaction with lipid membranes</u></a>	Chi, Eva Y
<a href="#"><u>2016058</u></a>	<a href="#"><u>Atlantic salmon gill microbiome</u></a>	Salinas, Irene
<a href="#"><u>2016057</u></a>	<a href="#"><u>High-fidelity Model for Wind Farms</u></a>	Lee, Sang
<a href="#"><u>2016056</u></a>	<a href="#"><u>Monte Carlo Simulation of Stochastic Multiplying Systems</u></a>	O'Rourke, Patrick F
<a href="#"><u>2016053</u></a>	<a href="#"><u>Nanophotonic metasurfaces</u></a>	Acosta, Victor
<a href="#"><u>2016052</u></a>	<a href="#"><u>Genomic analyses of cellular quiescence</u></a>	Osley, Mary Ann
<a href="#"><u>2016051</u></a>	<a href="#"><u>Performance Optimization of LANL Multi-Physics Applications</u></a>	Bridges, Patrick G

<a href="#">2016050</a>	<a href="#">Investigating the impact of metal contaminants in environmental microbial populations</a>	Cerrato, Jose M.
<a href="#">2016049</a>	<a href="#">COMPUTATIONAL INVESTIGATION OF PARAMETER SPACE APPLICABLE TO NUCLEAR FACILITY SAFEGUARDS USING THE UNM CENTER FOR ADVANCED RESEARCH COMPUTING</a>	Arthur, Edward D
<a href="#">2016047</a>	<a href="#">Multiscale Mechanistic Model to Study Nanotherapy Delivery in Tumors</a>	Bearer, Elaine L
<a href="#">2016046</a>	<a href="#">Designing of Fuel Performance Experiments to be Performed Using the Annular Research Reactor (ACRR)</a>	Lee, Youho
<a href="#">2016045</a>	<a href="#">Model building the HELP physical unclonable function</a>	Plusquellic, Jim F
<a href="#">2016044</a>	<a href="#">Genomic Comparisons of Multipartite Symbiosis: Understanding the metabolic basis of parasitism</a>	Kamel, Bishoy S
<a href="#">2016042</a>	<a href="#">QIIME analyses of microbial sequences acquired from saliva samples</a>	Carroll-Portillo, Amanda
<a href="#">2016041</a>	<a href="#">Deterministic and Bayesian Seismic source inversion involves</a>	Appelo, Daniel EA
<a href="#">2016040</a>	<a href="#">Consequence-based Impact Rating using ANSYS</a>	Moreu, Fernando
<a href="#">2016039</a>	<a href="#">Machine-Learning Design of Novel Photovoltaic Materials Based on Conceptual Understanding of Electronic Structure Calculations</a>	Talipov, Marat R
<a href="#">2016036</a>	<a href="#">Melt migration in continental interiors</a>	Roy, Mousumi
<a href="#">2016035</a>	<a href="#">Monte Carlo Simulation for Weak Neutron Sources</a>	Goss, Vanessa
<a href="#">2016034</a>	<a href="#">Programmable Nanowalkers: Models and Simulations</a>	Stefanovic, Darko
<a href="#">2016033</a>	<a href="#">Deep Learning and Differential Geometry</a>	Huang, Hongnian
<a href="#">2016032</a>	<a href="#">Analyzing Neuronal Coordination during a task of Behavioral Flexibility in a Model of Fetal Alcohol Spectrum Disorder</a>	Brigman, Jonathan L
<a href="#">2016031</a>	<a href="#">"Mountain Lions on the Edge: Integrating Conservation into Urban Planning through Predictive Modeling"</a>	Milne, Bruce T
<a href="#">2016029</a>	<a href="#">Discrete Element Modeling of Drilled Shafts in Granular Materials</a>	Ng, Tang-Tat
<a href="#">2016028</a>	<a href="#">Exploration of optical rogue wave phenomena in dielectrics as a function of the intrinsic randomness or disorder</a>	Mafi, Arash
<a href="#">2016026</a>	<a href="#">Differential Splicing by Sex in DNA Repair Genes</a>	Berwick, Marianne
<a href="#">2016021</a>	<a href="#">Atlantic salmon microbiome</a>	Salinas, Irene
<a href="#">2016019</a>	<a href="#">Small Area Population Estimates</a>	Rhatigan, Robert
<a href="#">2016018</a>	<a href="#">Differential Gene Expression in Cancer</a>	Trujillo, Kristina

CARC supports all sorts of computation.  
We can support yours too.  
(TLDR)

# Basics of HPC Systems

- Parallelism within CPUs (Central Processing Units)
- Parallelism within GPUs (Graphics Processing Units)
- Parallelism of CPUs and GPUs
- Parallelism of whole computers

# CARC Systems

- Monitoring
  - [Ganglia.alliance.unm.edu](http://Ganglia.alliance.unm.edu)
  - [Xdmod.hpc.unm.edu](http://Xdmod.hpc.unm.edu)
- QuickBytes: <http://carc.unm.edu/user-support-2/User%20support%20at%20a%20glance.html>
- Help
  - <http://help.carc.unm.edu>
  - [help@carc.unm.edu](mailto:help@carc.unm.edu)

# Some useful Linux commands

- ssh `username@wheeler.alliance.unm.edu`
- scp `myfile.txt username@wheeler.unm.edu:~`
- scp `username@wheeler.unm.edu:~/myfile.txt .`
- CyberDuck, or WinScp
- CARC supports secure file transfer (SFTP), so choose that protocol if you use a graphical transfer program.
- Rsync
- ls –lah
- find . –name “\*.txt”

# Compartmentalization

- A challenge running large multi-user systems is supporting all the different software required for hundreds of projects.
- Compartmentalization keeps user software isolated.
- We use 3 general methods at CARC: Environment Modules, Anaconda, and Singularity (these are the current standards so what you learn here will translate to other HPC centers)

# Environment Modules

- Open a secure shell on wheeler if you haven't already

```
$ ssh username@wheeler.alliance.unm.edu
```

- Display your environment variables

```
$ env
```

- All the modules do is set the environment variables for different software

# Environment Modules

- Let's load the R module so we can use it.

```
$ module list
```

```
$ module load r  Choose the appropriate R module (use tab complete !!)
```

```
$ R
```

# Environment Modules

- Let's load the R module so we can use it.

\$ module list (Again)

Will take a while!

\$ module avail (to show all available modules)

\$ module spider <software name> (to find a software package)

[https://lmod.readthedocs.io/en/latest/010\\_user.html](https://lmod.readthedocs.io/en/latest/010_user.html)

# Conda

- CARC staff have to install the software and create the environment modules you just saw.
- Anaconda provides an environment manager called `conda` that allows you to install the software you need into your home directory.
- Conda works with python, perl, R, and theoretically any language

# Conda – Hands On

- Let's setup a local install of numpy

```
$ module load anaconda
```

```
$ conda -V
```

```
$ conda create -n numpy numpy
```

Wait a while – introduce yourselves to your neighbor... believe there is a reason we are doing this...

# Conda – Hands On

- Let's setup a local install of numpy

```
$ module load anaconda
```

```
$ conda -V
```

```
$ conda create -n numpy numpy
```

Now we have defined a conda environment called numpy and installed numpy in our home directories.

We can now use numpy in the next program.

# Conda – Hands On

- Let's setup a local install of numpy

```
$ module load anaconda
```

```
$ conda -V
```

```
$ conda create -n numpy numpy
```

- Now we can load the environment

```
$ source activate numpy
```

# Conda – Hands On

Conda just installs the software under `~/.conda`

```
$ conda env list
```

```
$ source deactivate numpy
```

FYI: <https://pythonclock.org/>

# Docker and Singularity

- Singularity allows you to load converted Docker images on HPC systems.
- Docker is not secure so singularity locks down access to the host machine.
- Docker containers allow you to configure a whole virtual operating system environment (e.g. your software needs Ubuntu but Wheeler runs CentOS).
- Convert your docker image to singularity and you can run the container. There is a “QuickByte” (short tutorial on the CARC website) on how to do this:

<http://carc.unm.edu/usersupport2/User%20support%20at%20a%20glance.html>

15 mins Break

# Submitting Jobs

# Hands On

Download some example code that we can use to practice

Enter the following:

```
cd ~
```

```
git clone https://lobogit.unm.edu/CARC/workshops.git
```

# Multiuser Systems and Batch Scheduling

- TORQUE (PBS)
- MAUI

# Workflow

Head Node

User 1

Program A

PBS Script A

User 2

Program B

PBS Script B

Compute Node 01

Compute Node 02

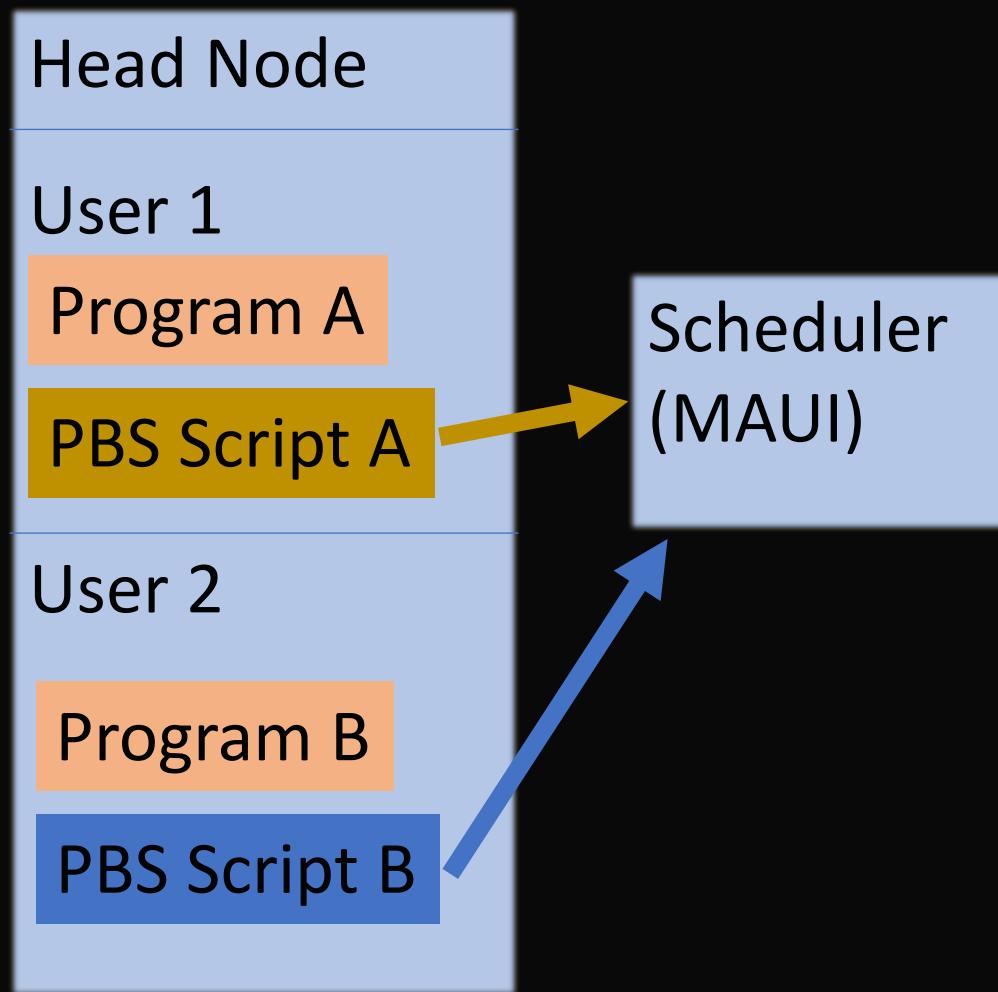
Compute Node 03

Compute Node 04

Compute Node 05

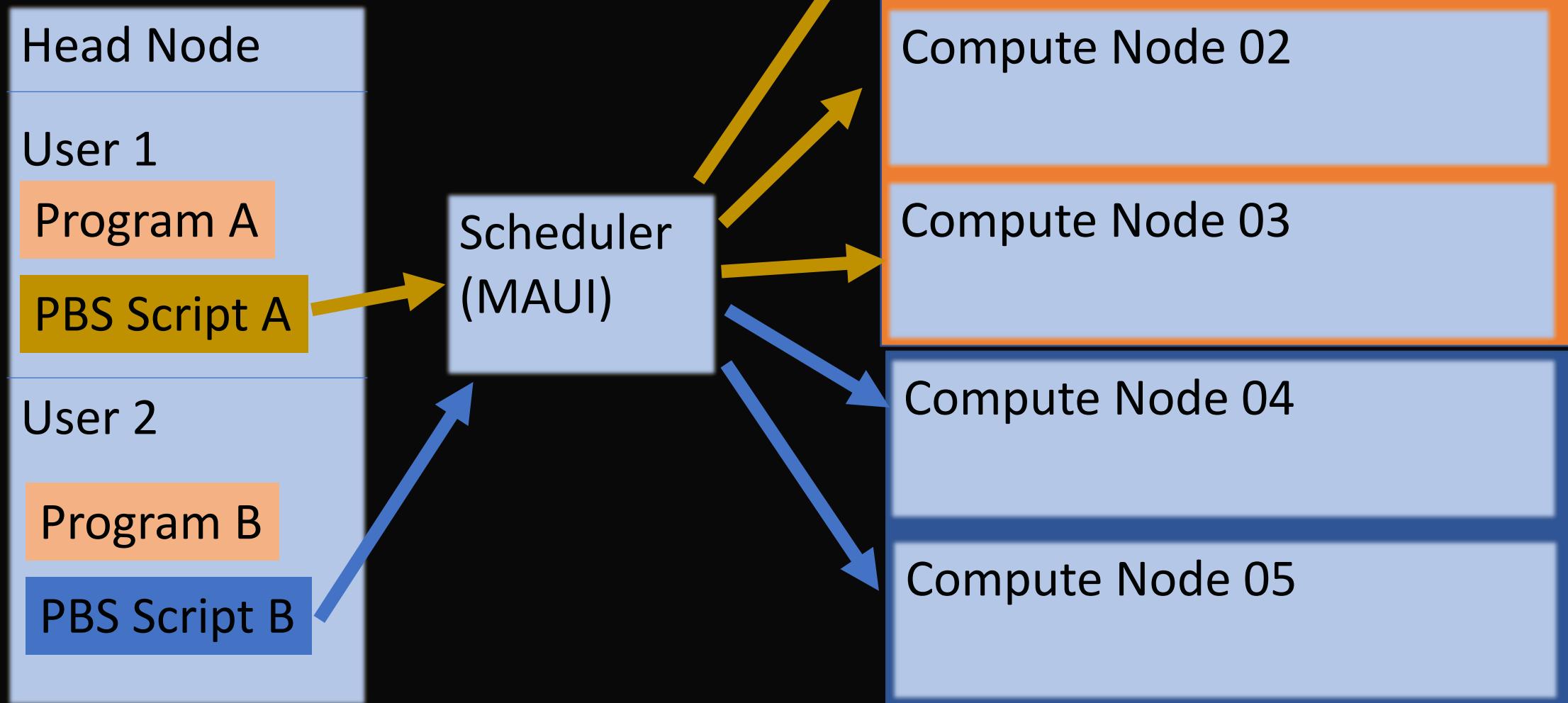
Shared filesystems – All nodes can access the same programs and write output

# Workflow



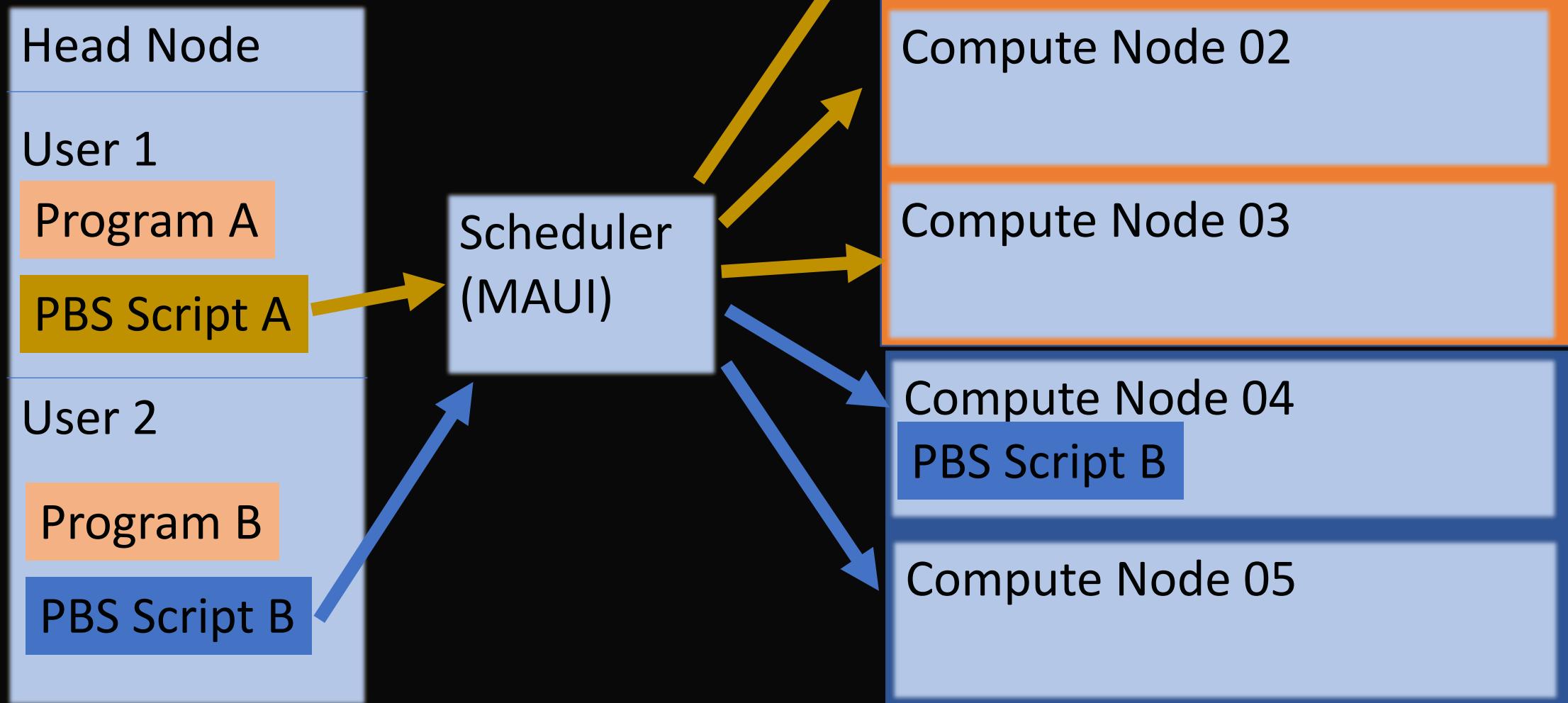
Shared filesystems – All nodes can access the same programs and write output

# Workflow



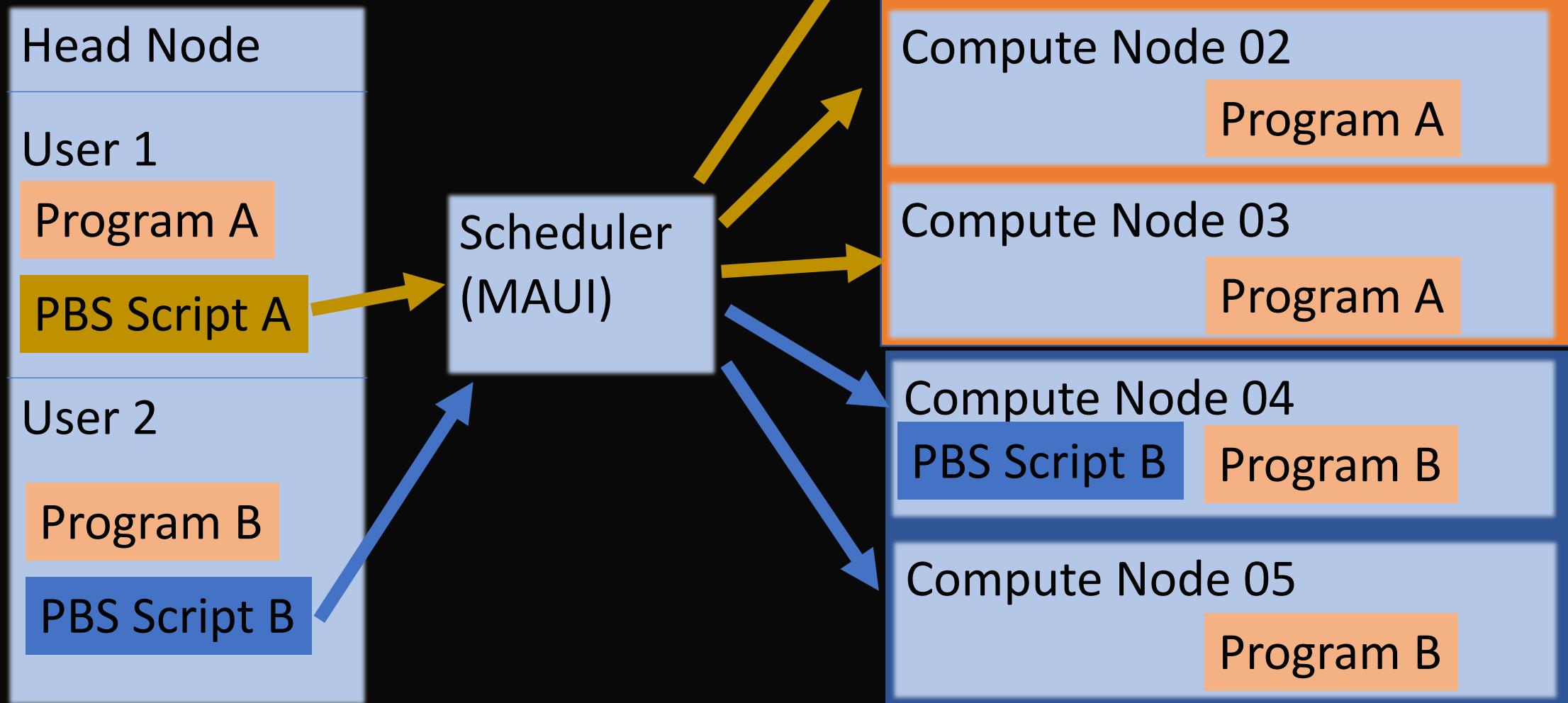
Shared filesystems – All nodes can access the same programs and write output

# Workflow



Shared filesystems – All nodes can access the same programs and write output

# Workflow



Shared filesystems – All nodes can access the same programs and write output

# Interactive Mode

```
$ qsub -I -l nodes=1:ppn=1 -q debug
```

# PBS Variables Provide Information

- In interactive mode try:

```
$ echo $PBS_O_WORKDIR
```

```
$ echo $PBS_NODEFILE
```

```
$ cat $PBS_NODEFILE
```

# MAUI Scheduler

The scheduler looks at all the currently queued and running jobs and runs a backfill algorithm.

Smaller jobs in terms of number of CPUs and requested time are easier to schedule since there is more likely to be space for them.

However the longer a job is in the queue the more priority it gets. This way every job runs eventually.

# PBS Variables

- There are lots:

\$PBS_ENVIRONMENT	\$PBS_JOBID	\$PBS_MOMPORT	\$PBS_NP	\$PBS_O_HOME
\$PBS_O_LOGNAME	\$PBS_O_QUEUE	\$PBS_O_WORKDIR	\$PBS_VERSION	\$PBS_GPUFILE
\$PBS_JOBNAME	\$PBS_NODEFILE	\$PBS_NUM_NODES	\$PBS_O_HOST	\$PBS_O_MAIL
\$PBS_O_SERVER	\$PBS_QUEUE	\$PBS_VNODENUM	\$PBS_JOBCOOKIE	\$PBS_MICFILE
\$PBS_NODENUM	\$PBS_NUM_PPN	\$PBS_O_LANG	\$PBS_O_PATH.	\$PBS_O_SHELL
\$PBS_TASKNUM	\$PBS_WALLTIME			

# Writing a Torque Batch Script

- Make a directory called `~/workshop`
- Edit a new file in the `workshop` directory, name it `workshop_example.pbs`



- This REQUESTS time on the debug queue. We are asking for 1 nodes, and 8 cores on that node. We promise our job won't take more than 5 minutes.

Email me when the begins, aborts, and ends (bae). Combine standard out and standard error into one file.

```
#!/bin/bash

#PBS -q debug
#PBS -l nodes=1:ppn=8
#PBS -l walltime=00:5:00
#PBS -N ws_example
#PBS -j oe
#PBS -m bae
#PBS -M my_email@unm.edu
```

# Writing a Torque Batch Script

```
#!/bin/bash

#PBS -q debug
#PBS -l nodes=1:ppn=8
#PBS -l walltime=00:5:00
#PBS -N ws_example
#PBS -j oe
#PBS -m bae
#PBS -M my_email@unm.edu

echo $HOSTNAME
```

Everything that comes after the PBS preamble is executed on the first node you were allocated.

# Writing a Torque Batch Script

```
#!/bin/bash

#PBS -q debug
#PBS -l nodes=1:ppn=8
#PBS -l walltime=00:5:00
#PBS -N ws_example
#PBS -j oe
#PBS -m bae
#PBS -M my_email@unm.edu

echo $HOSTNAME
```

qsub pbs/workshop\_example.pbs

# Writing a Torque Batch Script

```
#!/bin/bash

#PBS -q default
#PBS -l nodes=2:ppn=8
#PBS -l walltime=00:05:00
#PBS -N G09_H2O
#PBS -j oe
#PBS -m bae
#PBS -M my_email@unm.edu

INPUT_MOLECULE=$PBS_O_WORKDIR/data/H2O.gjf
OUTPUT_FILE=$PBS_O_WORKDIR/H2O.log

module load gaussian/g09
g09 $INPUT_MOLECULE $OUTPUT_FILE
```

qsub pbs/gaussian.pbs

# Managing your jobs

To submit your batch job:

```
$ cd workshops/intro_workshop  
$ ls  
pbs code data  
$ qsub pbs/workshop_example.pbs
```

# Writing a Torque Batch Script

Whatever is written to standard out and standard error is saved to <Job Name>.o<Job ID>  
When the job ends.

If you want to see the output live, start your job with:

```
$ qsub -k oe <script_name.pbs>
```

# Writing a Torque Batch Script

```
#!/bin/bash

#PBS -q debug
#PBS -l nodes=2:ppn=8
#PBS -l walltime=00:05:00
#PBS -N ws_example
#PBS -j oe
#PBS -m bae
#PBS -M my_email@unm.edu

cat $PBS_NODEFILE
```

qsub pbs/workshop\_example\_2.pbs

Everything that comes after the PBS preamble is executed on the first node you were allocated.

## Example Problem: Calculate $\pi$

---

## Serial Calculation of $\pi$

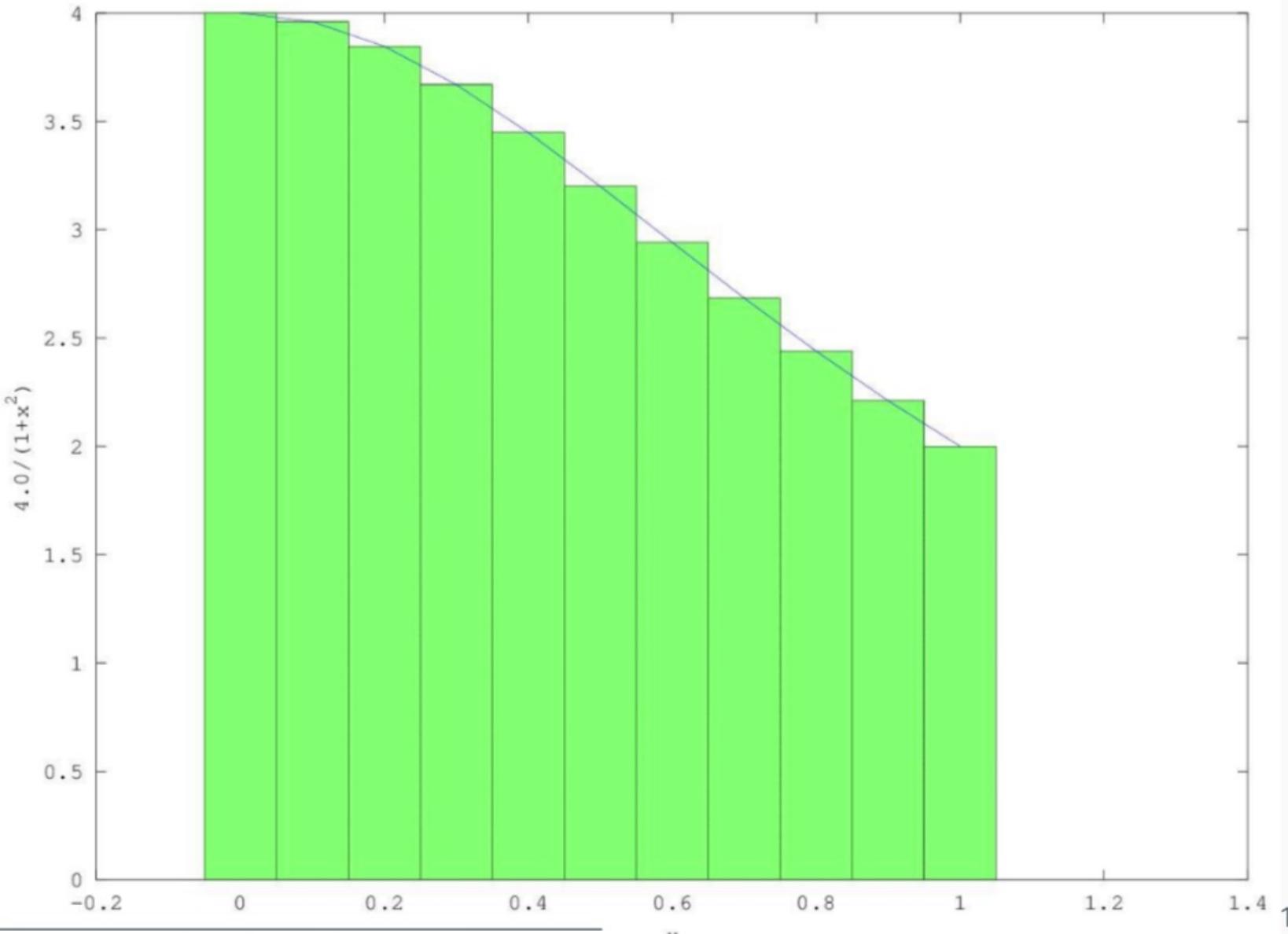
$$\int_0^1 \frac{4}{1+x^2} dx = \pi$$

And can be numerically approximated with:

$$\sum_{i=0}^N \frac{4}{1+x_i^2} \Delta x \approx \pi$$

As  $\Delta x$  gets smaller and  $N$  gets larger the approximation converges on  $\pi$

# Serial Program to Calculate $\pi$



```
# A program that calculates pi using the area under a  
curve  
# The program checks the value of pi calculated against  
the  
# value provided by numpy
```

```
import time  
import sys  
import numpy as np # Value of PI to compare to  
  
def Pi(num_steps): #Function to calculate pi  
    step = 1.0 / num_steps  
    sum = 0  
    for i in range(num_steps):  
        x = (i + 0.5) * step  
        sum = sum + 4.0 / (1.0 + x * x)  
    pi = step * sum  
    return pi
```

```
# Check that the caller gave us the number of steps to  
use  
if len(sys.argv) != 2:  
    print("Usage: ", sys.argv[0], " <number of steps>")  
    sys.exit(1)  
  
num_steps = int(sys.argv[1],10);  
  
# Call function to calculate pi  
start = time.time() #Start timing  
pi = Pi(num_steps)  
end = time.time() # End timing  
  
# Print our estimation of pi, the difference from numpy's  
value, and how long it took  
print("Pi = %.20f, (Diff=%.20f) (calculated in %f secs with  
%d steps)" %(pi, pi-np.pi, end - start, num_steps))  
sys.exit(0)
```

```
#!/bin/bash
#PBS -q debug
#PBS -l nodes=1:ppn=2
#PBS -l walltime=00:05:00
#PBS -N calc_pi_serial
#PBS -j oe
#PBS -M youremailaddress@unm.edu
```

```
module load anaconda
source activate numpy
```

```
cd $PBS_O_WORKDIR
python code/calcPiSerial.py
```

```
qsub pbs/calc_pi_serial.pbs
```

# Monitoring Jobs

Shows an overview of the queue status

```
$ qgrok
```

Lists all the jobs in the queue

```
$ qstat
```

Shows the resources requested by the jobs

```
$ qstat -a
```

Just the jobs submitted by a particular user

```
$ qstat -u <username>
```

-n shows the nodes being used by a running job

```
$ qstat -n -u <username>
```

-f gives detailed information about a particular job

```
$ qstat -f <job id>
```

Watch is a useful command that automatically updates the command that follows

```
$ watch qstat -n -u <username>
```

To see an estimate of how long it will be before a job starts and completes, enter:

```
$ /usr/local/maui/bin/showstart <job ID>
```

# Embarrassingly Parallel Problems

- Perfect case!
- All computation is independent so speedup is equal to the number of additional computers you throw at the problem.
- Especially good for generating lots of samples when you have a stochastic algorithm, simulation, or want to benchmark performance.

# Job Arrays

```
#!/bin/bash

#PBS -q debug
#PBS -l nodes=1:ppn=8
#PBS -l walltime=00:05:00
#PBS -N ws_example
#PBS -j oe
#PBS -m bae
#PBS -M my_email@unm.edu
#PBS -t 1-12%3

echo "$HOSTNAME - $PBS_ARRAYID"
```

qsub pbs/workshop\_example\_3.pbs

-t allows you to schedule many jobs at once. -t 1-12%3 means run 12 jobs but only schedule 3 at a time.

# Writing a Torque Batch Script

```
#!/bin/bash

#PBS -q default
#PBS -l nodes=1:ppn=8
#PBS -l walltime=00:05:00
#PBS -N calc_pi_array
#PBS -j oe
#PBS -m bae
#PBS -M my_email@unm.edu
#PBS -t 1-12%3

module load anaconda
source activate numpy

NUM_STEPS="${PBS_ARRAYID}000"
echo "Calculating pi with $NUM_STEPS..."
cd $PBS_O_WORKDIR
python code/calcPiSerial.py $NUM_STEPS
```

**qsub pbs/calc\_pi\_array.pbs**

Everything that comes after the PBS preamble is executed on the first node you were allocated.

# GNU Parallels – Input Driven

```
$ module load parallel  
#module load parallel #Load the appropriate version f  
  
$ find . -name "*.txt"  
  
$ parallel echo :::: A B C :::: D E F  
  
$ find . -name "*.txt" | parallel echo {}  
  
$ find . -name "*.txt" | parallel echo {.}  
  
$ find . -name "*.txt" | parallel echo {/.}
```

# GNU Parallel – Monitoring Progress

- A logfile of the jobs completed so far can be generated with --joblog:

```
$ parallel --joblog $PBS_O_WORKDIR/job.log exit :::: 1 2 3 0  
$ cat $PBS_O_WORKDIR/job.log
```

- The log contains the job sequence, which host the job was run on, the start time and run time, how much data was transferred, the exit value, the signal that killed the job, and finally the command being run.

# GNU Parallel – Resuming Jobs

- With a joblog GNU **parallel** can be stopped and later pickup where it left off. It is important that the input of the completed jobs is unchanged.
- Why would you want to do this...???

```
$ parallel --joblog $PBS_O_WORKDIR/job.log exit :::: 1 2 3 0
$ cat $PBS_O_WORKDIR/job.log
$ parallel --resume --joblog $PBS_O_WORKDIR/job.log exit :::: 1
2 3 0 0 0
$ cat $PBS_O_WORKDIR/job.log
```

# GNU Parallel – Resuming Jobs

- The previous command just ran the jobs that didn't finish
- This command reruns jobs that has a failing exit code

```
$ parallel --joblog $PBS_O_WORKDIR/job.log exit :::: 1 2 3 0
$ cat $PBS_O_WORKDIR/job.log
$ parallel -resume-failed --joblog $PBS_O_WORKDIR/job.log exit :::: 1 2
3 0 0 0
$ cat $PBS_O_WORKDIR/job.log
```

We are running “exit” so we can fake jobs that succeed and fail.

# GNU Parallel

- GNU Parallel is what you should be using to run many experiments, solve many independent instances of a problem.
- If you have 1000 input files and 100 CPUs allocated parallel will do all the scheduling for you to process those files.

# GNU Parallel

- If you have 1000 input files and 100 CPUs allocated parallel will do all the scheduling for you to process those files.
- Remember though: Torque assigns you resources, parallel makes use of them. You have to use  
`--sshloginfile $PBS_NODEFILE`
- To be sure parallel is using resources you were actually allocated

# GNU Parallel – Environments

Recall that software may require particular environments. GNU Parallel by itself loses the environment in which it was called.

Use `env_parallel` to keep the environment. Need to tell it what shell you are using.

```
source `which env_parallel.bash`
```

Then you can use `env_parallel` exactly like `parallel`.

```
#!/bin/bash

#PBS -l nodes=2:ppn=8
#PBS -l walltime=00:05:00
#PBS -N calc_pi_parallel
#PBS -j oe
#PBS -M youremailaddress@unm.edu

module load parallel-20170322-gcc-4.8.5-2ycpx7e
module load anaconda
source activate numpy

source $(which env_parallel.bash)

cd $PBS_O_WORKDIR
env_parallel --sshloginfile $PBS_NODEFILE --joblog $PBS_JOBNAME.joblog "python
$PBS_O_WORKDIR/code/calcPiSerial.py {}" :::: data/step_sizes.txt
```

qsub pbs/calc\_pi\_parallel.pbs

# GNU Parallels

```
# Create a temporary unique directory in which to store the summary output for each job
TEMP_DIR=$(mktemp -d -p $PBS_O_WORKDIR)

# Setup Gurobi solver environment
module load parallel #Load the appropriate version for the cluster on which you are running
module load gurobi
module load anaconda
source activate gurobi

source `which env_parallel.bash`

# Use find to make a list of all the .graph files to pass to the integer program solver.
# Divide the work up among compute nodes using the GNU parallel tool. Use a local /tmp work directory.
# ":::: -" reads from stdin (find ... *.graph) to {1}, ":::: $EPSILON_VALUES_PATH" reads from the
# epsilon parameter file to {2}, {1/.} fetches the input base filename only
find $GRAPH_INPUT_DIR -name '*.graph' | env_parallel --jobs 8 --joblog
$PBS_O_WORKDIR/ip_progress.joblog --resume-failed --sshloginfile $PBS_NODEFILE --workdir $(mktemp -d)
"python $PBS_O_WORKDIR/ip/solve_ip.py {1} $IP_METHOD {2} $TEMP_DIR/{1/.}.txt $TIME_LIMIT
$SOLUTION_OUTPUT_DIR" :::: - $EPSILON_VALUES_PATH
```

# GNU Parallel

```
find $GRAPH_INPUT_DIR -name '*.graph' | env_parallel --jobs 8  
$PBS_O_WORKDIR/ip_progress.joblog --resume-failed --sshloginfile $PBS_NODEFILE --  
workdir $(mktemp -d) "python $PBS_O_WORKDIR/ip/solve_ip.py {1} $IP_METHOD {2}  
$TEMP_DIR/{1/.}.txt $TIME_LIMIT $SOLUTION_OUTPUT_DIR" :::: - $EPSILON_VALUES_PATH
```

Let's try to parse this command together...

# GNU Parallels

```
find $GRAPH_INPUT_DIR -name '*.graph' | env_parallel --jobs 8  
$PBS_O_WORKDIR/ip_progress.joblog --resume-failed --sshloginfile $PBS_NODEFILE --  
workdir $(mktemp -d) "python $PBS_O_WORKDIR/ip/solve_ip.py {1} $IP_METHOD {2}  
$TEMP_DIR/{1/.}.txt $TIME_LIMIT $SOLUTION_OUTPUT_DIR" :::: - $EPSILON_VALUES_PATH
```

Pass the paths to all the files with a graph extension in the directory specified in the user shell variable \$GRAPH\_INPUT\_DIR to env\_parallel.

# GNU Parallels

```
find $GRAPH_INPUT_DIR -name '*.graph' | env_parallel --jobs 8 --joblog  
$PBS_O_WORKDIR/ip_progress.joblog --resume-failed --sshloginfile $PBS_NODEFILE --  
workdir $(mktemp -d) "python $PBS_O_WORKDIR/ip/solve_ip.py {1} $IP_METHOD {2}  
$TEMP_DIR/{1/.}.txt $TIME_LIMIT $SOLUTION_OUTPUT_DIR" :::: - $EPSILON_VALUES_PATH
```

Pipes the paths to all the files with a graph extension in the directory specified in the user shell variable \$GRAPH\_INPUT\_DIR to env\_parallel.

The piped input is mapped to the first parameter to parallel referred to with “-”.

# GNU Parallels

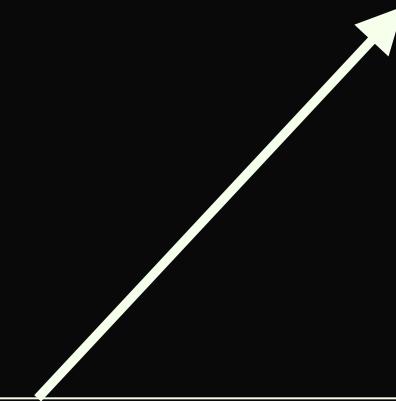
... and referred to with {1}.

```
find $GRAPH_INPUT_DIR -name '*_graph' | env_parallel --jobs 8 --joblog  
$PBS_O_WORKDIR/ip_progress.joblog --resume-failed --sshloginfile $PBS_NODEFILE --  
workdir $(mktemp -d) "python $PBS_O_WORKDIR/ip/solve_ip.py {1} $IP_METHOD {2}  
$TEMP_DIR/{1/.}.txt $TIME_LIMIT $SOLUTION_OUTPUT_DIR" :::: - $EPSILON_VALUES_PATH
```

The piped input is mapped to the first parameter to parallel referred to with “-”.

# GNU Parallel

```
find $GRAPH_INPUT_DIR -name '*.graph' | env_parallel --jobs 8 --joblog  
$PBS_O_WORKDIR/ip_progress.joblog --resume-failed --sshloginfile $PBS_NODEFILE --  
workdir $(mktemp -d) "python $PBS_O_WORKDIR/ip/solve_ip.py {1} $IP_METHOD {2}  
$TEMP_DIR/{1/.}.txt $TIME_LIMIT $SOLUTION_OUTPUT_DIR" :::: - $EPSILON_VALUES_PATH
```



The second set of parameters is read from a file. In this example the path to the values is specified by the user variable \$EPSILON\_VALUES\_PATH.

# GNU Parallels

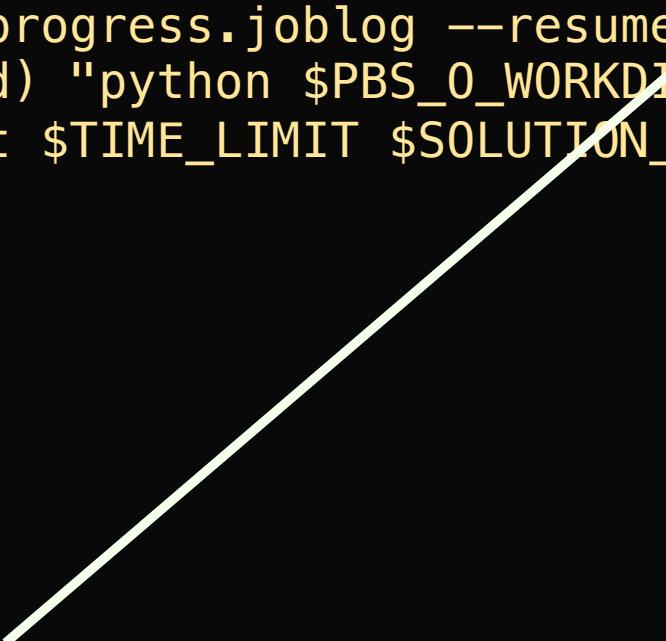
```
find $GRAPH_INPUT_DIR -name '*.graph' | env_parallel --jobs 8 --jobtag  
$PBS_O_WORKDIR/ip_progress.joblog --resume-failed --sshloginfile $PBS_NODEFILE --  
workdir $(mktemp -d) "python $PBS_O_WORKDIR/ip/solve_ip.py {1} $IP_METHOD {2}  
$TEMP_DIR/{1/.}.txt $TIME_LIMIT $SOLUTION_OUTPUT_DIR" :::: - $EPSILON_VALUES_PATH
```

... and referred to with {2}.

The second set of parameters is read from a file. In this example the path to the values is specified by the user variable \$EPSILON\_VALUES\_PATH.

# GNU Parallels

```
find $GRAPH_INPUT_DIR -name '*.graph' | env_parallel --jobs 8 --joblog  
$PBS_O_WORKDIR/ip_progress.joblog --resume-failed --sshloginfile $PBS_NODEFILE --  
workdir $(mktemp -d) "python $PBS_O_WORKDIR/ip/solve_ip.py {1} $IP_METHOD {2}  
$TEMP_DIR/{1/.}.txt $TIME_LIMIT $SOLUTION_OUTPUT_DIR" :::: - $EPSILON_VALUES_PATH
```



We are using `env_parallel` which passes the current shell environment to the jobs. In this example the user code uses shell variables.

# GNU Parallel

```
find $GRAPH_INPUT_DIR -name '*.graph' | env_parallel --jobs 8 --joblog  
$PBS_O_WORKDIR/ip_progress.joblog --resume-failed --shloginfile $PBS_NODEFILE --  
workdir $(mktemp -d) "python $PBS_O_WORKDIR/ip/solve_ip.py {1} $IP_METHOD {2}  
$TEMP_DIR/{1/.}.txt $TIME_LIMIT $SOLUTION_OUTPUT_DIR" :::: - $EPSILON_VALUES_PATH
```

Specifies the number of jobs to run on each node.

# GNU Parallel

```
find $GRAPH_INPUT_DIR -name '*.graph' | env_parallel --jobs 8 --joblog  
$PBS_O_WORKDIR/ip_progress.joblog --resume-failed --sshloginfile $PBS_NODEFILE --  
workdir $(mktemp -d) "python $PBS_O_WORKDIR/ip/solve_ip.py {1} $IP_METHOD {2}  
$TEMP_DIR/{1/.}.txt $TIME_LIMIT $SOLUTION_OUTPUT_DIR" :::: - $EPSILON_VALUES_PATH
```

Record the progress to a file.

# GNU Parallels

```
find $GRAPH_INPUT_DIR -name '*.graph' | env_parallel --jobs 8 --joblog  
$PBS_O_WORKDIR/ip_progress.joblog --resume-failed --sshloginfile $PBS_NODEFILE --  
workdir $(mktemp -d) "python $PBS_O_WORKDIR/ip_solve_ip.py {1} $IP_METHOD {2}  
$TEMP_DIR/{1/.}.txt $TIME_LIMIT $SOLUTION_OUTPUT_DIR" :::: - $EPSILON_VALUES_PATH
```

Record the progress to a file.

...and tell parallels to rerun any failed jobs listed in the joblog (those where the exit value not equal to 0).

# GNU Parallels

```
find $GRAPH_INPUT_DIR -name '*.graph' | env_parallel --jobs 8 --joblog  
$PBS_O_WORKDIR/ip_progress.joblog --resume-failed --sshloginfile $PBS_NODEFILE --  
workdir $(mktemp -d) "python $PBS_O_WORKDIR/ip/solve_ip.py {1} $IP_METHOD {2}  
$TEMP_DIR/{1/.}.txt $TIME_LIMIT $SOLUTION_OUTPUT_DIR" :::: - $EPSILON_VALUES_PATH
```

Tell parallels which nodes we were allocated to run our jobs.

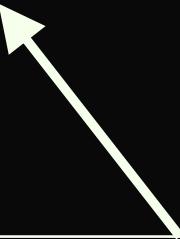
# GNU Parallels

```
find $GRAPH_INPUT_DIR -name '*.graph' | env_parallel --jobs 8 --joblog  
$PBS_O_WORKDIR/ip_progress.joblog --resume-failed --sshloginfile $PBS_NODEFILE --  
workdir $(mktemp -d) "python $PBS_O_WORKDIR/ip/solve_ip.py {1} $IP_METHOD {2}  
$TEMP_DIR/{1/.}.txt $TIME_LIMIT $SOLUTION_OUTPUT_DIR" :::: - $EPSILON_VALUES_PATH
```

--workdir is set to a temporary directory.

# GNU Parallel

```
find $GRAPH_INPUT_DIR -name '*.graph' | env_parallel --jobs 8 --joblog  
$PBS_O_WORKDIR/ip_progress.joblog --resume-failed --sshloginfile $PBS_NODEFILE --  
workdir $(mktemp -d) "python $PBS_O_WORKDIR/ip/solve_ip.py {1} $IP_METHOD {2}  
$TEMP_DIR/{1/.}.txt $TIME_LIMIT $SOLUTION_OUTPUT_DIR" :::: - $EPSILON_VALUES_PATH
```



The command that will be run in each parallel job. This program takes 6 arguments. Parallel will generate a job for all combinations of input parameter {1} and {2}. Argument 4 specifies an output path based on the input file name {1}. {1/.} gets just the input file basename.

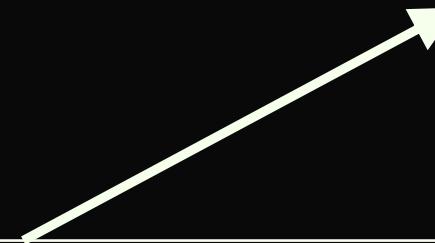
# GNU Parallel

```
find $GRAPH_INPUT_DIR -name '*.graph' | env_parallel --jobs 8 --joblog  
$PBS_O_WORKDIR/ip_progress.joblog --resume-failed --sshloginfile $PBS_NODEFILE --  
workdir $(mktemp -d) "python $PBS_O_WORKDIR/ip/solve_ip.py {1} $IP_METHOD {2}  
$TEMP_DIR/{1/.}.txt $TIME_LIMIT $SOLUTION_OUTPUT_DIR" :::: - $EPSILON_VALUES_PATH
```

Notice “::::” rather than “::”. Four colons tells parallel that a list of parameters comes next.

# GNU Parallels

```
find $GRAPH_INPUT_DIR -name '*.graph' | env_parallel --jobs 8 --joblog  
$PBS_O_WORKDIR/ip_progress.joblog --resume-failed --sshloginfile $PBS_NODEFILE --  
workdir $(mktemp -d) "python $PBS_O_WORKDIR/ip/solve_ip.py {1} $IP_METHOD {2}  
$TEMP_DIR/{1/.}.txt $TIME_LIMIT $SOLUTION_OUTPUT_DIR" :::: - $EPSILON_VALUES_PATH
```



So in this example, parallels will spawn a job for every combination of input file and value in the \$EPSILON\_VALUES\_PATH file.

Parallels records its progress in a joblog file so it can pick up where it left off if the torque job runs out of time before all the jobs are complete, and the torque job needs to be resubmitted.

15 Min Break

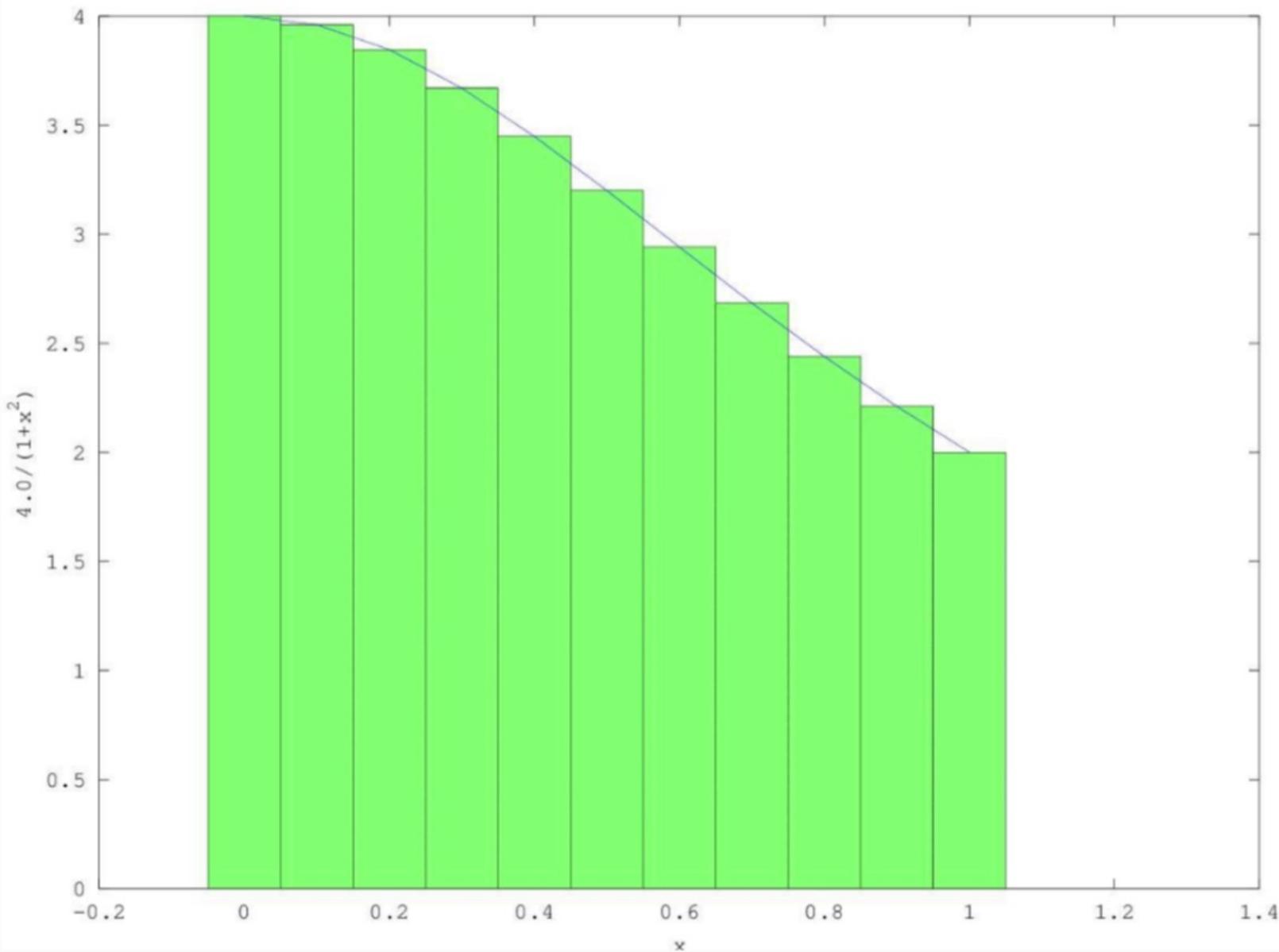
# File Systems

Home Directory (~): Limited space (200 GB), backed up, slowest access times.

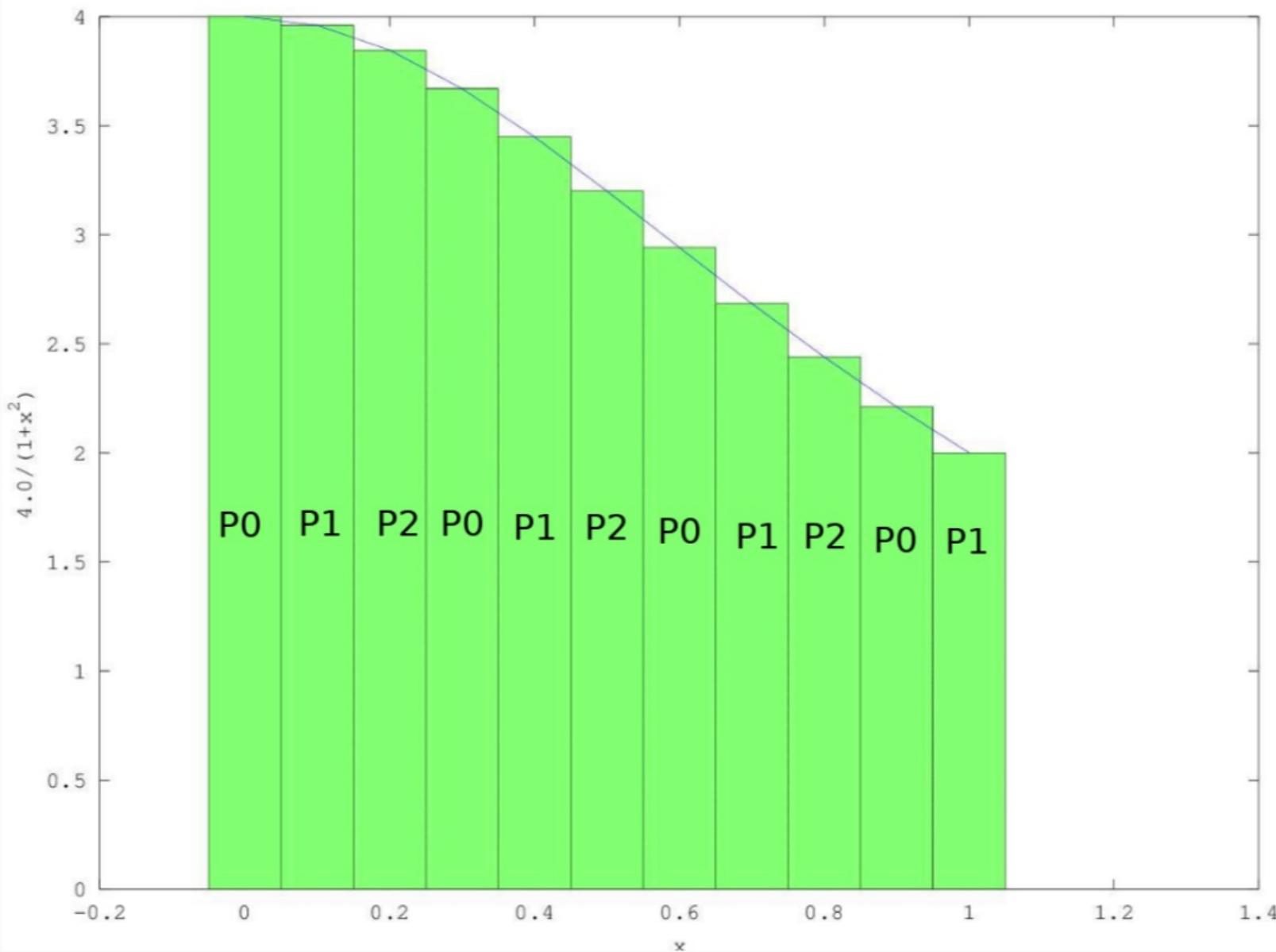
Scratch (~/wheeler-scratch): Fast. Lots of space (up to 1 TB usage), store data that you can regenerate here.

Temp (/tmp): on Wheeler these are RAM drives. Very fast but usage decreases available memory.

# Serial Program to Calculate $\pi$



# Parallel Program to Calculate $\pi$



# MPI: Message Passing Interface

When programs need to run on many processors but also communicate with one another.

Here the parallel version of calcPi needs to communicate the partial sums computed by each process so they can all be added up.

To communicate we will use the MPI library:

```
$ module load anaconda
```

```
$ conda create -n mpi_numpy mpi mpi4py numpy
```

```

import time
import sys
import numpy as np # Value of PI to compare to

##### SETUP MPI - START #####
from mpi4py import MPI      #Import the MPI library
comm = MPI.COMM_WORLD       #Communication framework
root = 0                    #Root process
rank = comm.Get_rank()       #Rank of this process
num_procs = comm.Get_size()  #Total number of processes
##### END #####
#Distributed function to calculate pi
def Pi(num_steps):
    step = 1.0 / num_steps
    sum = 0
    for i in range(rank, num_steps, num_procs): # Divide sum among
processes
        x = (i + 0.5) * step
        sum = sum + 4.0 / (1.0 + x * x)
    mypi = step * sum

    # Get that partial sums from all the processes, add them up, and give
to the root process
    pi = comm.reduce(mypi, MPI.SUM, root)
    return pi

```

```

#Main function
# Check that the caller gave us the number of steps to use
if len(sys.argv) != 2:
    print("Usage: ", sys.argv[0], " <number of steps>")
    sys.exit(1)

num_steps = int(sys.argv[1],10);

#Broadcast number of steps to use to the other processes
comm.bcast(num_steps, root)

# Call function to calculate pi
start = time.time() #Start timing
pi = Pi(num_steps) # Call the function that calculates pi
end = time.time() # End timing

# If we are the root process then print our estimation of pi,
# the difference from numpy's value, and how long it took
print("Pi = %.20f, (Diff=%.20f) (calculated in %f secs with %d
steps)" %(pi, pi-np.pi, end - start, num_steps))

```

```
#!/bin/bash

#PBS -l nodes=2:ppn=8
#PBS -l walltime=00:05:00
#PBS -N calc_pi_mpi
#PBS -j oe
#PBS -M youremailaddress@unm.edu

module load openmpi-3.1.3-gcc-4.8.5-5fyhoph
module load anaconda
source activate mpi_numpy

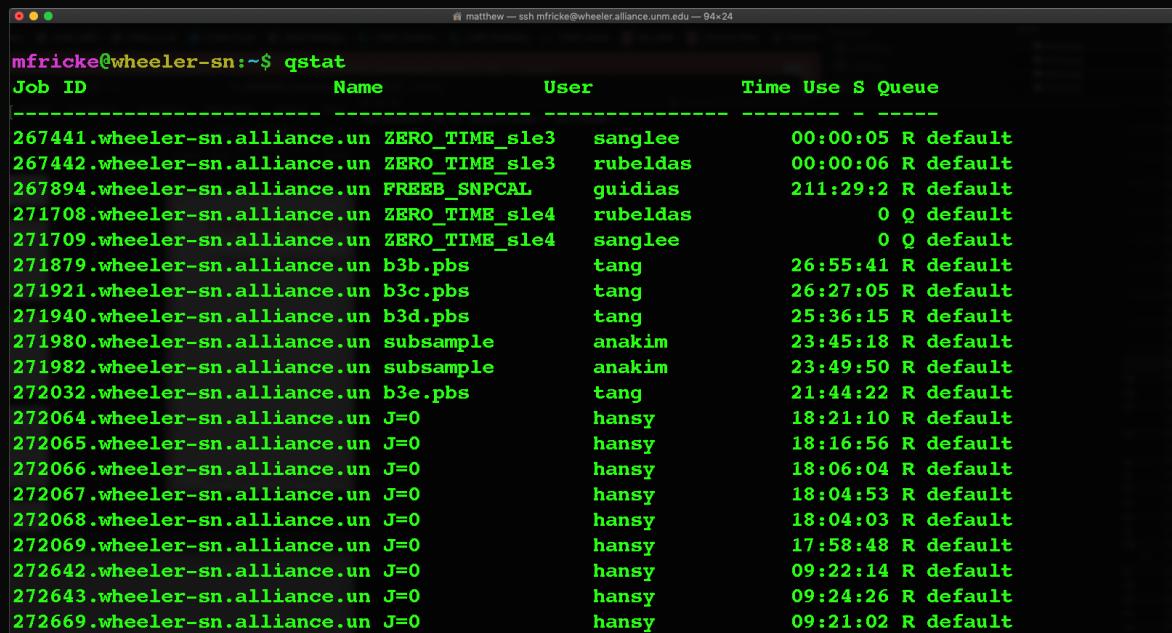
cd $PBS_O_WORKDIR
mpirun -machinefile $PBS_NODEFILE -n $PBS_NP python code/calcPiMPI.py
```

qsub pbs/calc\_pi\_mpi.pbs

# Flexible Access to CARC

- There are several ways to access CARC:

Terminal access is the traditional way to interact with our systems.



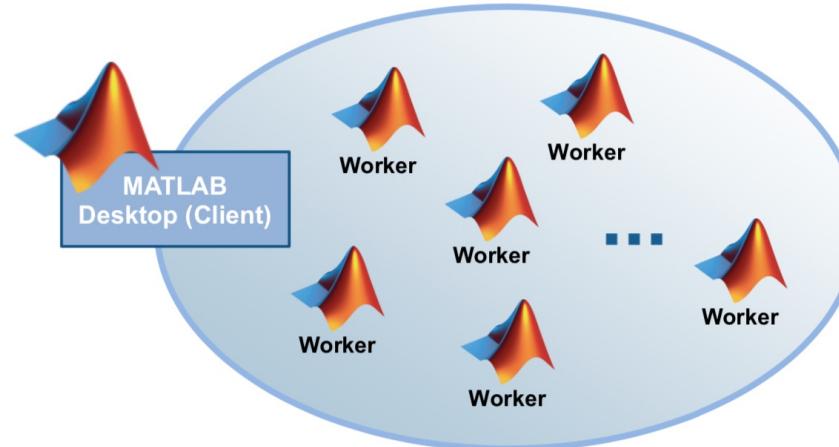
A screenshot of a terminal window titled "matthew — ssh mfricke@wheeler.alliance.unm.edu — 94x24". The window displays the output of the "qstat" command, which lists various jobs running on the system. The columns in the table are Job ID, Name, User, Time, Use, S, and Queue. The "Time" column shows the start time of the job, and the "Use" column shows the current usage. Most jobs are in the "R default" queue.

Job ID	Name	User	Time	Use	S	Queue
267441.wheeler-sn.alliance.un	ZERO_TIME_sle3	sanglee	00:00:05			R default
267442.wheeler-sn.alliance.un	ZERO_TIME_sle3	rubeldas	00:00:06			R default
267894.wheeler-sn.alliance.un	FREEB_SNPCAL	guidias	211:29:2			R default
271708.wheeler-sn.alliance.un	ZERO_TIME_sle4	rubeldas		0	Q	default
271709.wheeler-sn.alliance.un	ZERO_TIME_sle4	sanglee		0	Q	default
271879.wheeler-sn.alliance.un	b3b.pbs	tang	26:55:41			R default
271921.wheeler-sn.alliance.un	b3c.pbs	tang	26:27:05			R default
271940.wheeler-sn.alliance.un	b3d.pbs	tang	25:36:15			R default
271980.wheeler-sn.alliance.un	subsample	anakim	23:45:18			R default
271982.wheeler-sn.alliance.un	subsample	anakim	23:49:50			R default
272032.wheeler-sn.alliance.un	b3e.pbs	tang	21:44:22			R default
272064.wheeler-sn.alliance.un	J=0	hansy	18:21:10			R default
272065.wheeler-sn.alliance.un	J=0	hansy	18:16:56			R default
272066.wheeler-sn.alliance.un	J=0	hansy	18:06:04			R default
272067.wheeler-sn.alliance.un	J=0	hansy	18:04:53			R default
272068.wheeler-sn.alliance.un	J=0	hansy	18:04:03			R default
272069.wheeler-sn.alliance.un	J=0	hansy	17:58:48			R default
272642.wheeler-sn.alliance.un	J=0	hansy	09:22:14			R default
272643.wheeler-sn.alliance.un	J=0	hansy	09:24:26			R default
272669.wheeler-sn.alliance.un	J=0	hansy	09:21:02			R default

# Flexible Access to CARC

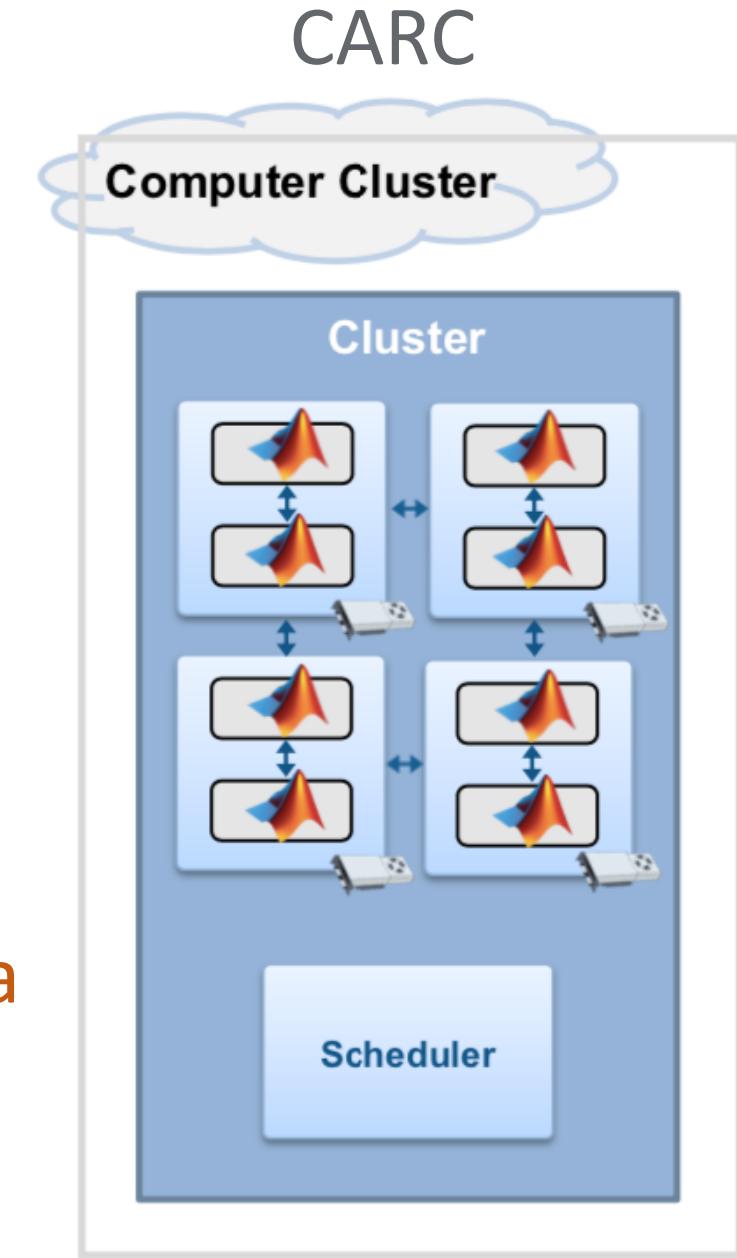
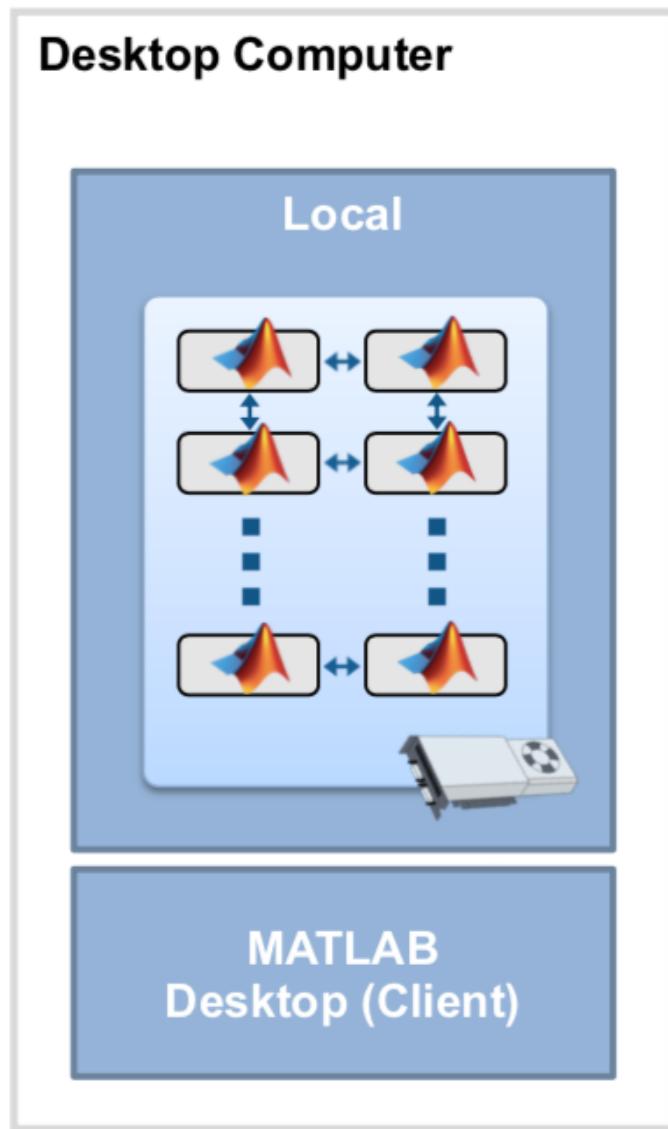
But users can also access CARC's systems through MATLAB and JupyterHub...

Look for the CARC QuickBytes on how to use these:



<https://github.com/UNM-CARC/QuickBytes/blob/master/README.md>

# Parallel MATLAB

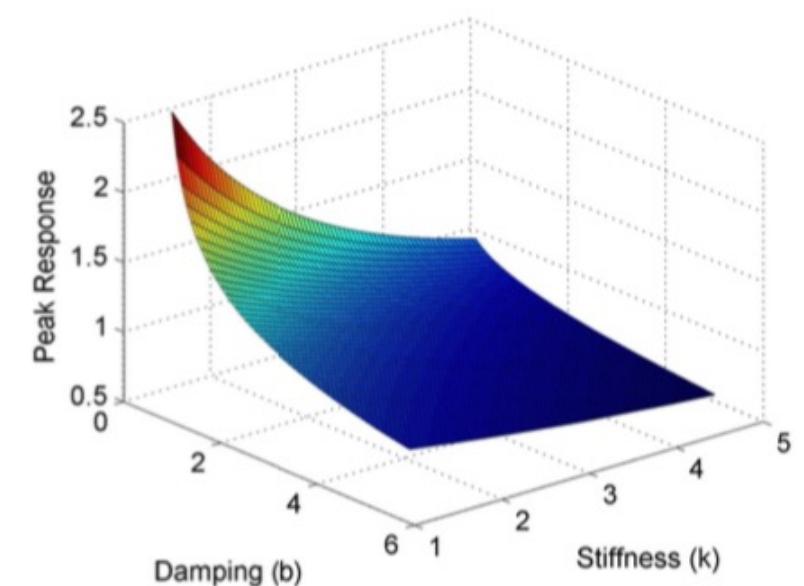
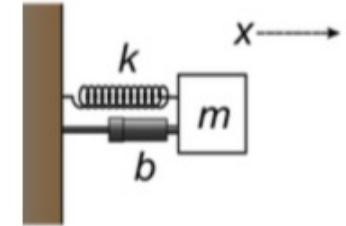
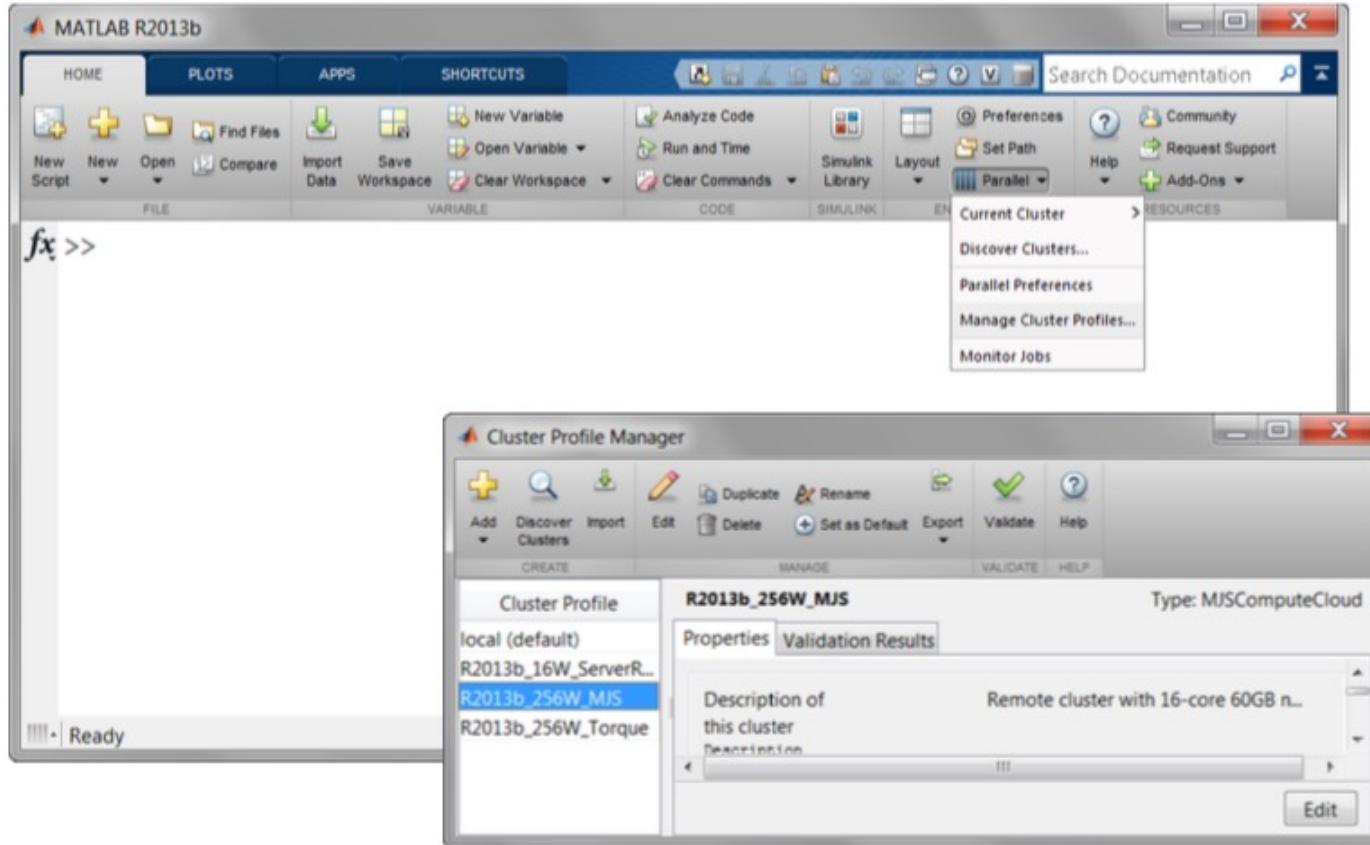


Code and Data  
→  
←  
Results

# Parallel MATLAB

- Change hardware without changing code
- Example: ODE Parameter Sweep

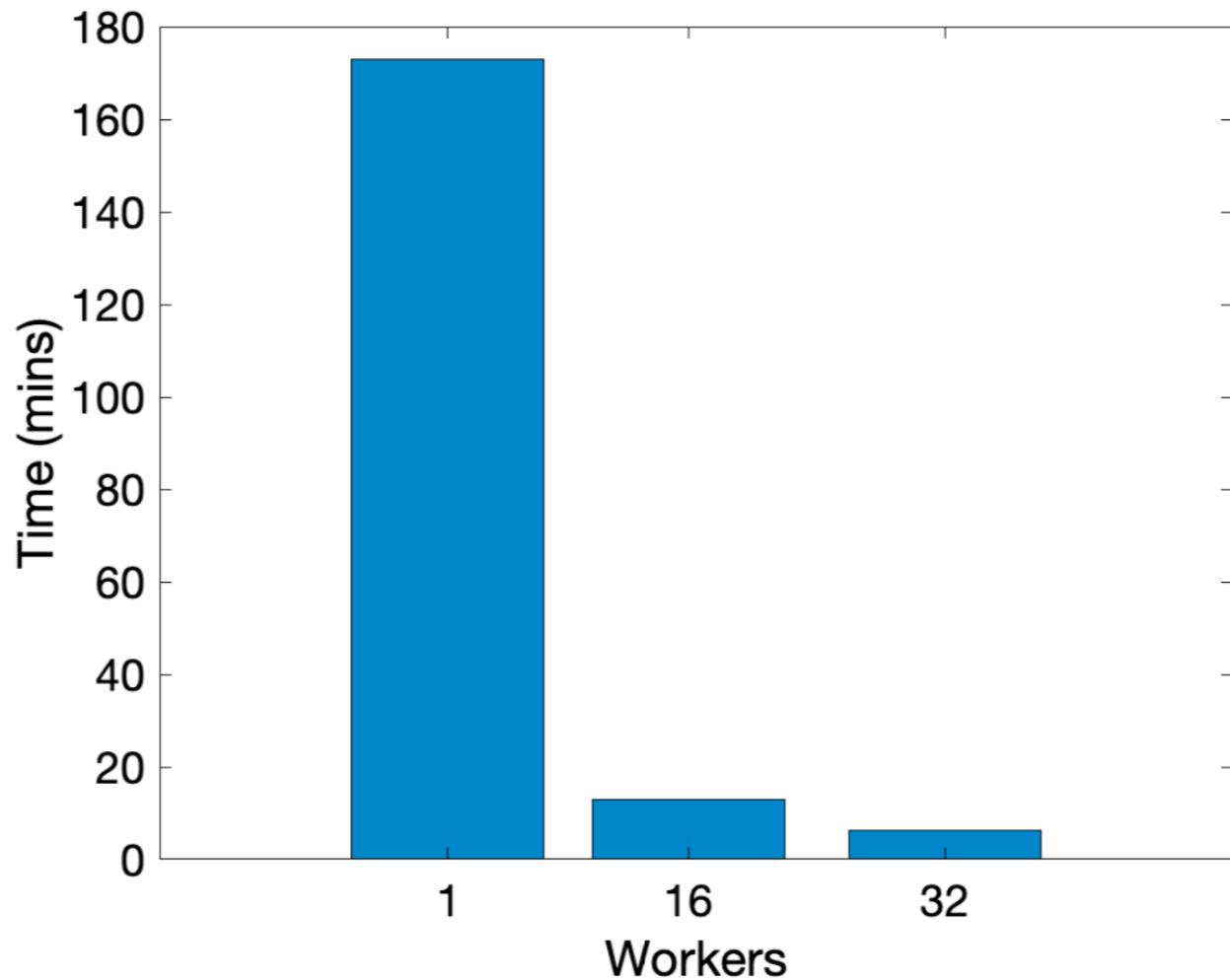
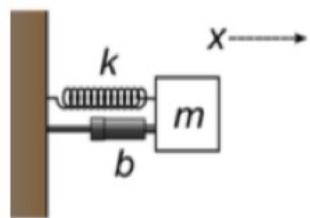
$$m\ddot{x} + \underbrace{b}_{1,2,\dots} \dot{x} + \underbrace{k}_{1,2,\dots} x = 0$$



# Parallel MATLAB

$$m\ddot{x} + b\dot{x} + kx = 0$$

1,2,...      1,2,...



Workers	Computation (minutes)	Speed-up
1	173	1
16	13	13
32	6.4	27
64	3.2	55
96	2.1	83
128	1.6	109
160	1.3	134
192	1.1	158

# Parallel MATLAB - GPUs

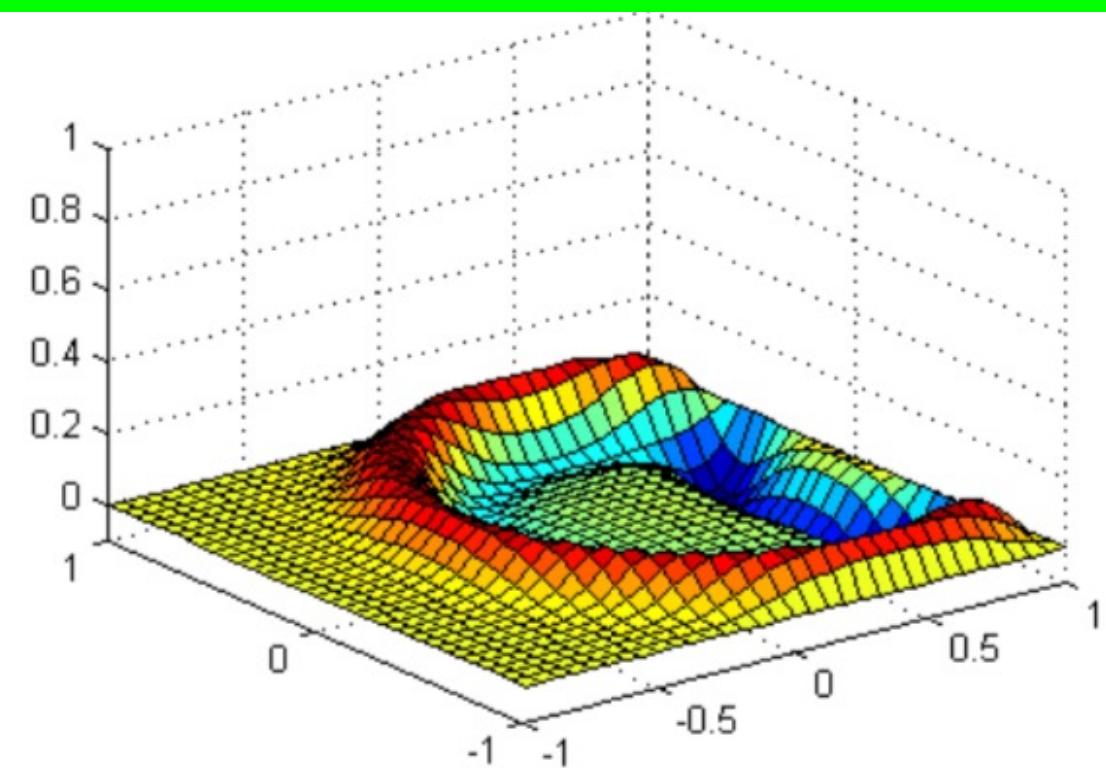
- Solve 2<sup>nd</sup> order wave equation using spectral methods:

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

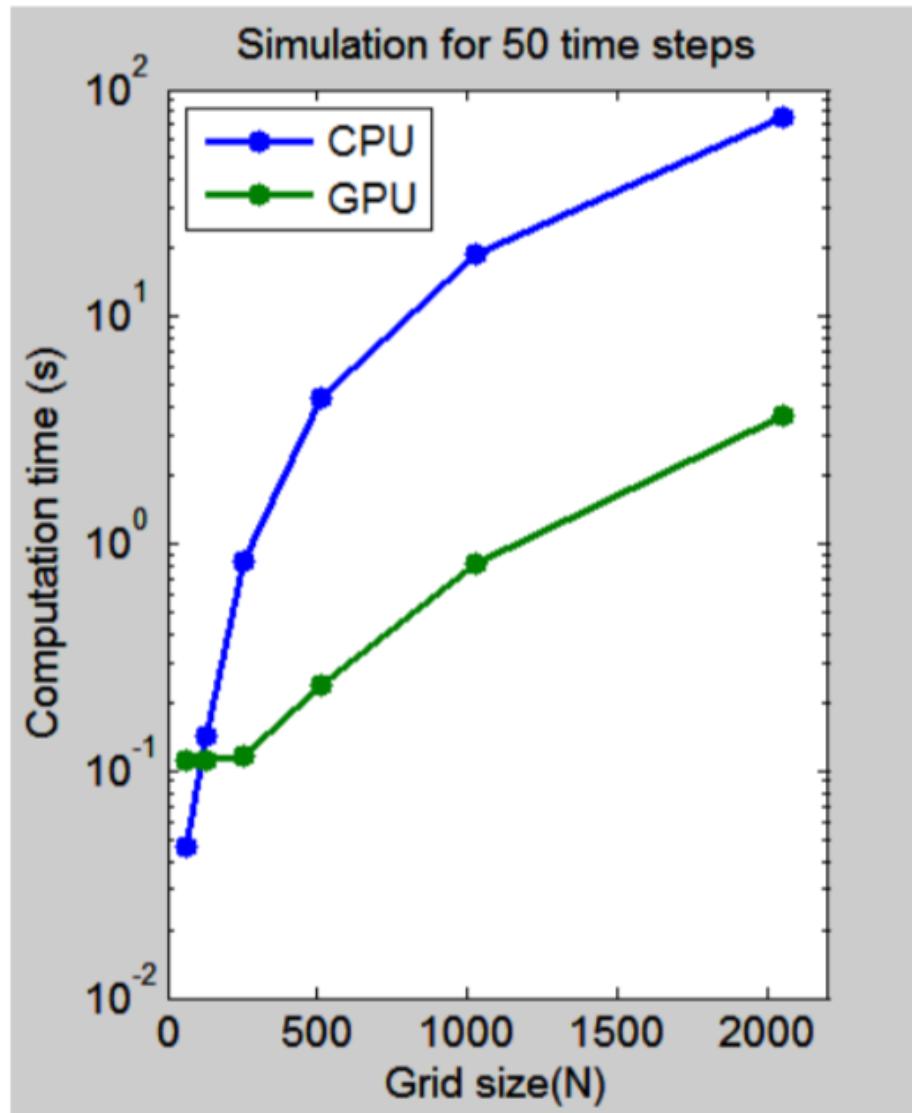
- Run both on CPU and GPU



Plot solutions to the Wave PDE



# Parallel MATLAB - GPUs



Grid Size	CPU (s)	GPU (s)	Speedup
64 x 64	0.05	0.11	0.4
128 x 128	0.14	0.11	1.3
256 x 256	0.83	0.12	7.2
512 x 512	4.40	0.24	18
1024 x 1024	18.79	0.82	23
2048 x 2048	75.03	3.67	20

Intel Xeon Processor W3550 (3.07GHz),  
NVIDIA Tesla K20c GPU



- <https://wheeler.alliance.unm.edu:8000>
- <https://taos.alliance.unm.edu:8000>
- <https://xena.alliance.unm.edu:8000>

Files

Running

IPython Clusters

Select items to perform actions on them.

 0  /

- 8.3.0
- admin\_scripts
- astrobiology
- avogadro-1.2.0
- batchspawner
- bin\_bak
- certs
- conda-bld
- Coursera
- Desktop
- Documents
- dot\_julia\_backup
- Downloads
- freebayes
- freebayes-1.1.0-l5nnhjrcrn4gatqajrcpxbst4y3ejsw
- g16
- Ganglia\_Settings
- gaussian
- IHaskell
- intro\_workshop
- IRC2020
- jupyterhub

 Upload  New 

Notebook:

- Python 3
- Python [conda env:.conda-R]
- Python [conda env:.conda-jupyterhub]
- Python [conda env:.conda-qiime2-2019.10]
- Python [conda env:.conda-qiime2-2019.4]
- Python [conda env:root]
- R
- R [conda env:.conda-R]

Other:

- Text File
- Folder
- Terminal

2 months ago

2 months ago

5 months ago

2 months ago

a month ago

a month ago

a month ago

3 months ago

3 months ago

a month ago

2 months ago

14 days ago

3 months ago



## Lorenz Equations

```
In [ ]: \begin{align}
\dot{x} &= \sigma(y-x) \\
\dot{y} &= \rho x - y - xz \\
\dot{z} &= -\beta z + xy
\end{align}
```





## Lorenz Equations

$$\dot{x} = \sigma(y - x)$$

$$\dot{y} = \rho x - y - xz$$

$$\dot{z} = -\beta z + xy$$

In [ ]:

|



## Lorenz Equations

$$\dot{x} = \sigma(y - x)$$

$$\dot{y} = \rho x - y - xz$$

$$\dot{z} = -\beta z + xy$$

Define a python function to plot the equations above

Import libraries

```
In [ ]: %matplotlib inline
from ipywidgets import interact, interactive
from IPython.display import clear_output, display, HTML
import numpy as np
from scipy import integrate
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import cnames
from matplotlib import animation
```

File Edit View Insert Cell Kernel Widgets Help

Trusted

Python 3



```
    return [sigma * (y - x), x * (rho - z) - y, x * y - beta * z]

# Choose random starting points, uniformly distributed from -15 to 15
np.random.seed(1)
x0 = -15 + 30 * np.random.random((N, 3))

# Solve for the trajectories
t = np.linspace(0, max_time, int(250*max_time))
x_t = np.asarray([integrate.odeint(lorenz_deriv, x0i, t)
                 for x0i in x0])

# choose a different color for each trajectory
colors = plt.cm.viridis(np.linspace(0, 1, N))

for i in range(N):
    x, y, z = x_t[i,:,:].T
    lines = ax.plot(x, y, z, 'r', c=colors[i])
    plt.setp(lines, linewidth=2)

ax.view_init(30, angle)
plt.show()

return t, x_t
```

Call the Lorenz solver with N trajectories and initial viewing angle 0

In [ ]: t, x\_t = solve\_lorenz(angle=0, N=10)|

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3



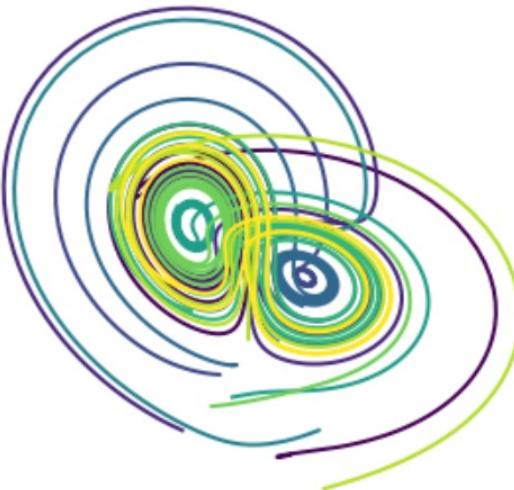
```
for i in range(N):
    x, y, z = x_t[i,:,:].T
    lines = ax.plot(x, y, z, ' ', c=colors[i])
    plt.setp(lines, linewidth=2)

    ax.view_init(30, angle)
    plt.show()

return t, x_t
```

Call the Lorenz solver with N trajectories and initial viewing angle 0

In [4]: `t, x_t = solve_lorenz(angle=0, N=10)`



In [ ]:

File Edit View Insert Cell Kernel Widgets Help

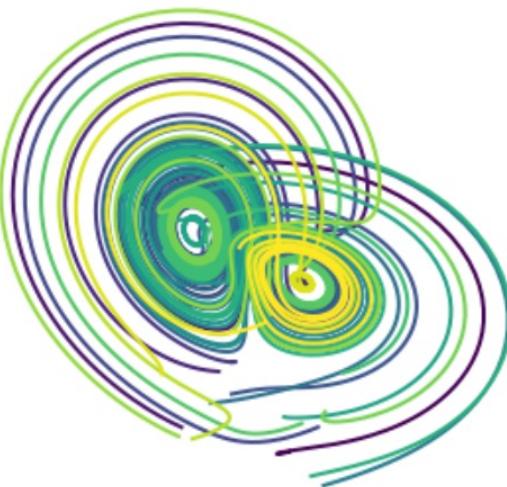
Trusted

Python 3



N=(0,50), sigma=(0.0,50.0), rho=(0.0,50.0))

display(w)

N  20angle  0.00max\_time  4.00sigma  10.00beta  2.67rho  26.40

In [ ]:

File Edit View Insert Cell Kernel Widgets Help

Trusted

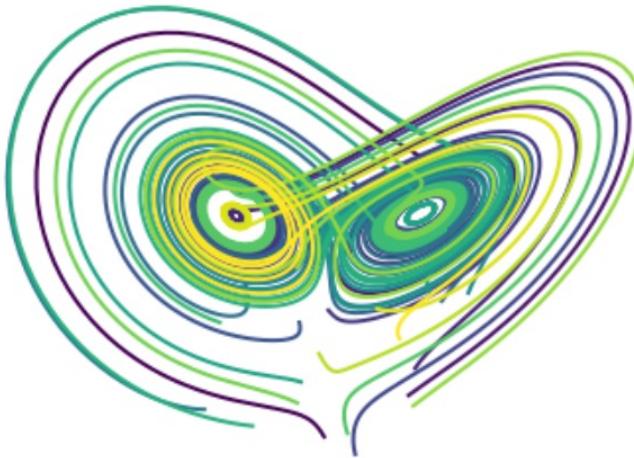
Python 3



N=(0,50), sigma=(0.0,50.0), rho=(0.0,50.0)

display(w)

N	<input type="range"/>	20
angle	<input type="range"/>	123.30
max_time	<input type="range"/>	4.00
sigma	<input type="range"/>	10.00
beta	<input type="range"/>	2.67
rho	<input type="range"/>	26.40



In [ ]:

We are here to help you!  
[help@carc.unm.edu](mailto:help@carc.unm.edu)