Introduction to HPC at the UNM Center for Advanced Research Computing

Matthew Fricke (mfricke@unm.edu)

Research Assistant Professor

http://unm.edu/~mfricke

Please send corrections to me at mfricke@unm.edu.
Version 1.2.

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 π

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 π
- 15 min break

Outline

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

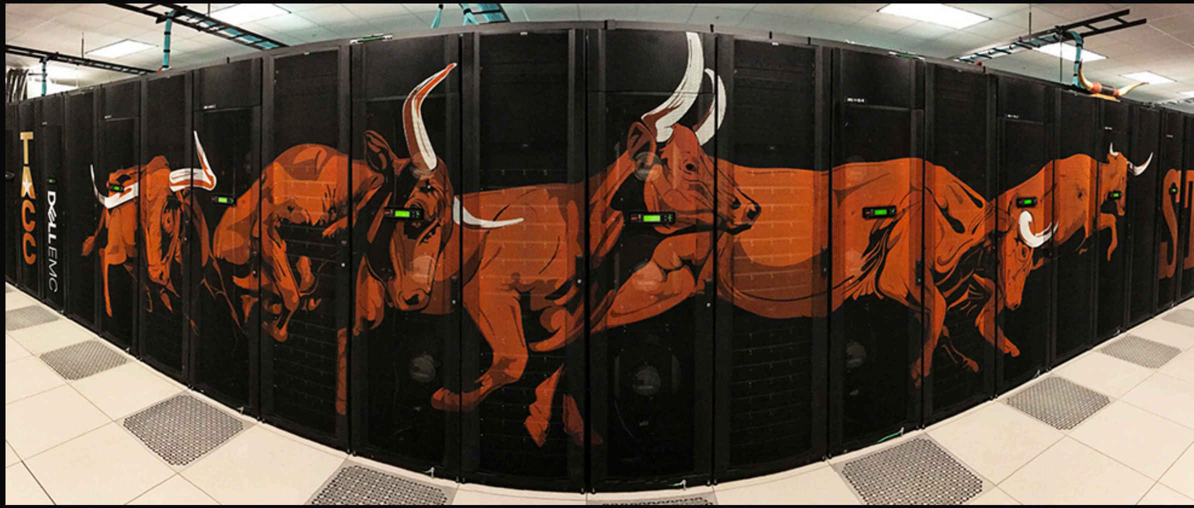
What is High Performance Computing?

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

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.

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

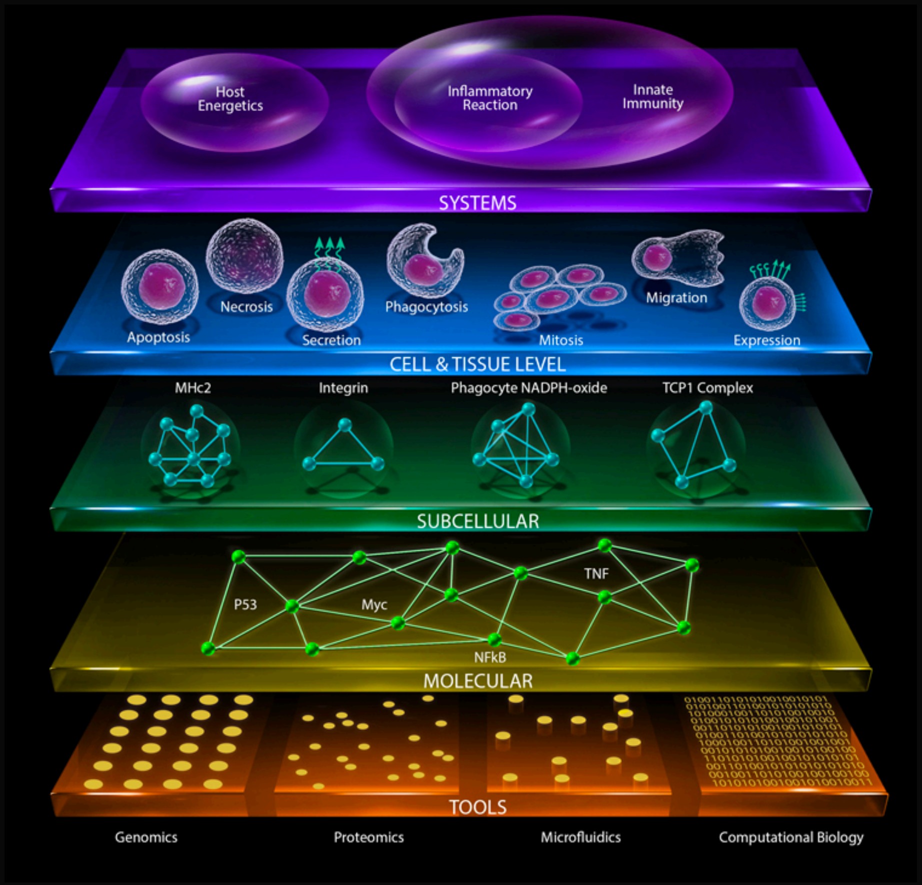
Big Data and Machine Learning

- 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 fast at them 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.

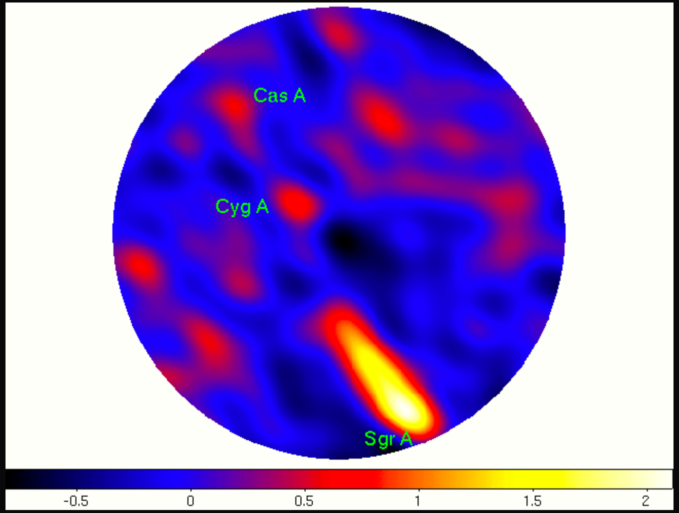


- Pandemic flu modelling
- Tuberculosis antibiotic resistance
- Pacific island bird genetics
- Intra-species viral spread
- NM Tree species mapping from LASER scans

Physics

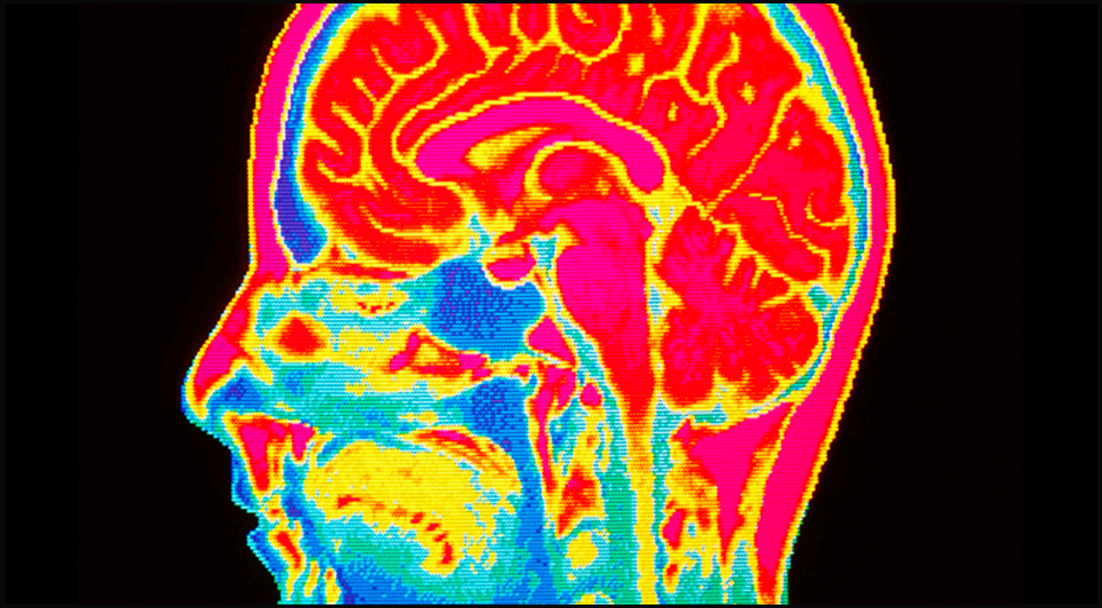
- Wheeler is a general purpose scale-out machine used by biophysicists, cosmologists, and many others.
- Gibbs is primarily used by computational chemists.

Molecular simulation of new photovoltaic materials

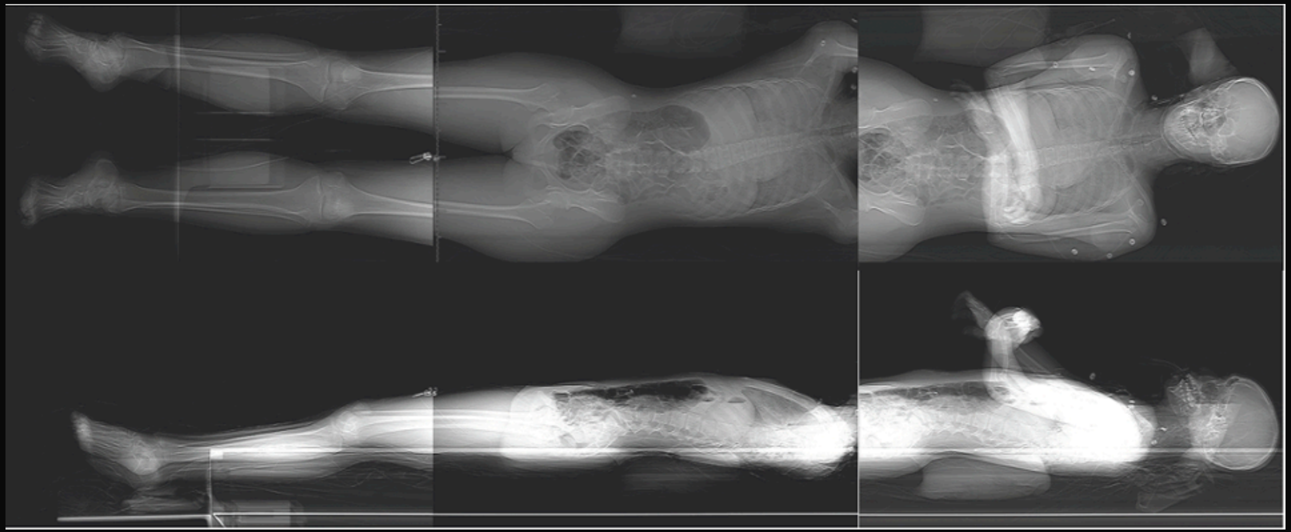


Long Wavelength Array Radio Telescope
Data Processing on Wheeler

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

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

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

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

2016050	Investigating the impact of metal contaminants in environmental microbial populations	Cerrato, Jose M.
2016049	COMPUTATIONAL INVESTIGATION OF PARAMETER SPACE APPLICABLE TO NUCLEAR FACILITY SAFEGUARDS USING THE UNM CENTER FOR ADVANCED RESEARCH COMPUTING	Arthur, Edward D
2016047	Multiscale Mechanistic Model to Study Nanotherapy Delivery in Tumors	Bearer, Elaine L
2016046	Designing of Fuel Performance Experiments to be Performed Using the Annular Research Reactor (ACRR)	Lee, Youho
2016045	Model building the HELP physical unclonable function	Plusquellic, Jim F
2016044	Genomic Comparisons of Multipartite Symbiosis: Understanding the metabolic basis of parasitism	Kamel, Bishoy S
20		
20		
20		
20		
20		
20		
20		
20		
20		
2016031	"Mountain Lions on the Edge: Integrating Conservation into Urban Planning through Predictive Modeling"	Milne, Bruce T
2016029	Discrete Element Modeling of Drilled Shafts in Granular Materials	Ng, Tang-Tat
2016028	Exploration of optical rogue wave phenomena in dielectrics as a function of the intrinsic randomness or disorder	Mafi, Arash
2016026	Differential Splicing by Sex in DNA Repair Genes	Berwick, Marianne
2016021	Atlantic salmon microbiome	Salinas, Irene
2016019	Small Area Population Estimates	Rhatigan, Robert
2016018	Differential Gene Expression in Cancer	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
  - Xmod.alliance.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

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 is 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

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 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 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 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

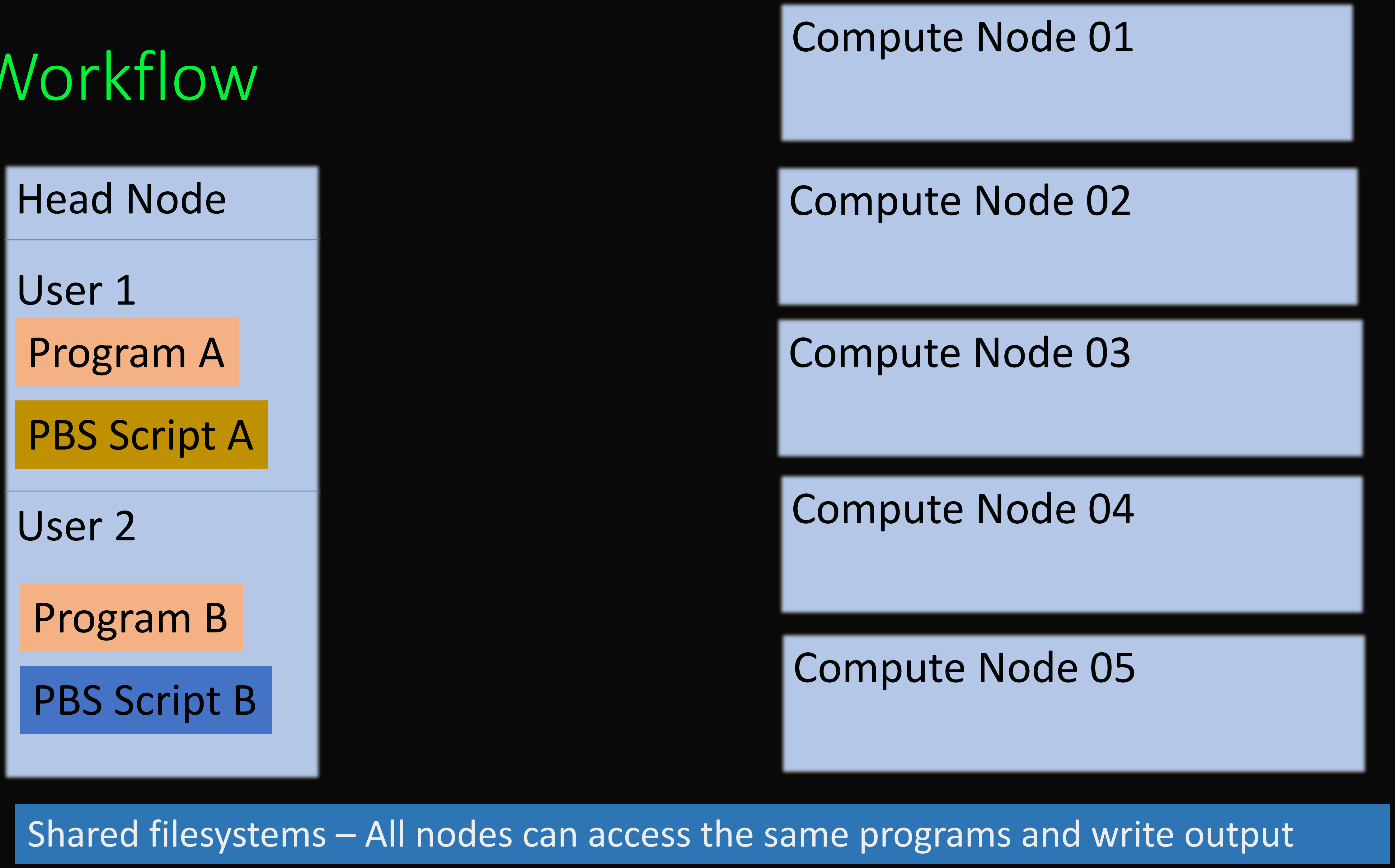
Enter the following:

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

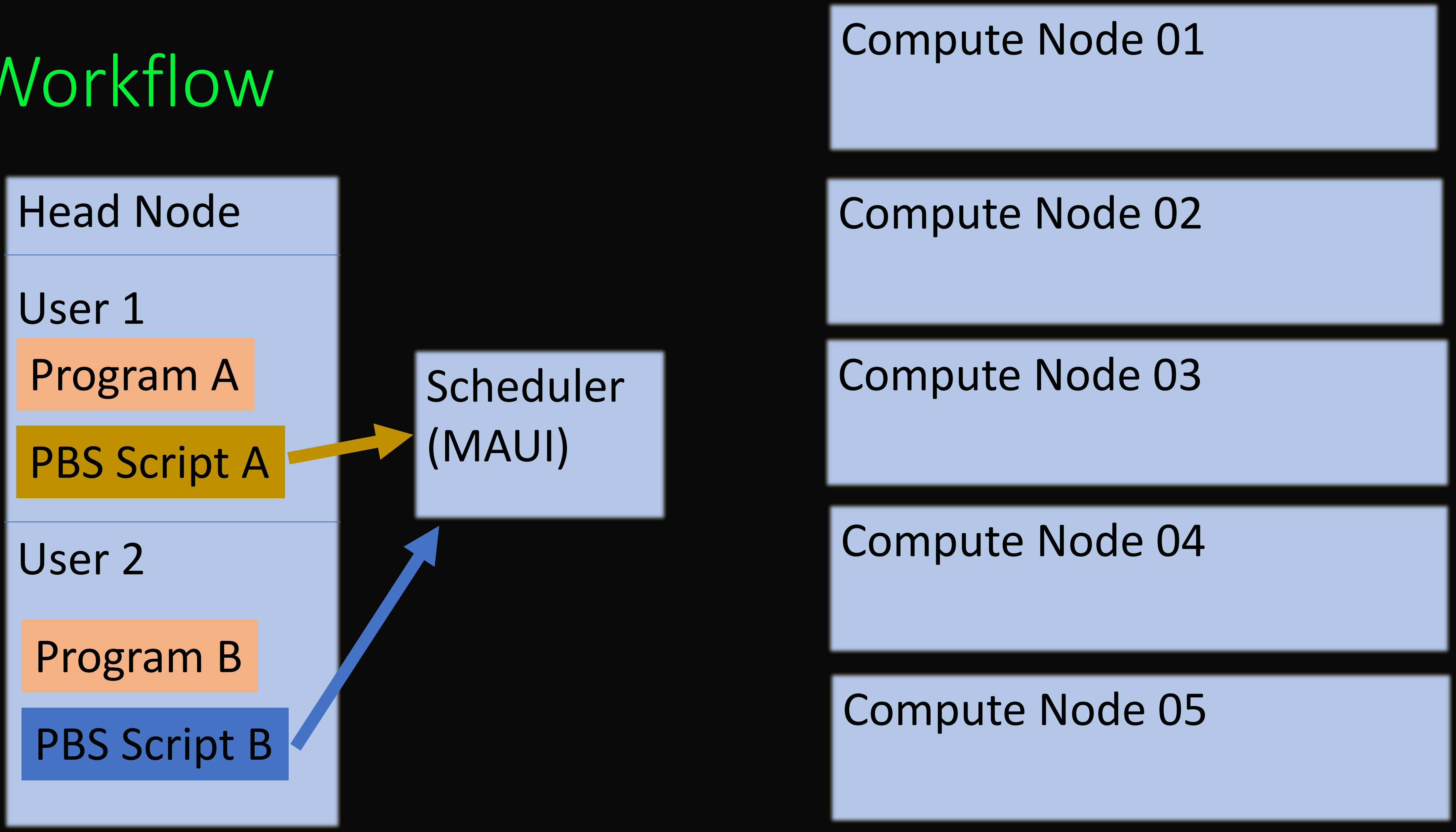
Multiuser Systems and Batch Scheduling

- TORQUE (PBS)
- MAUI

Workflow

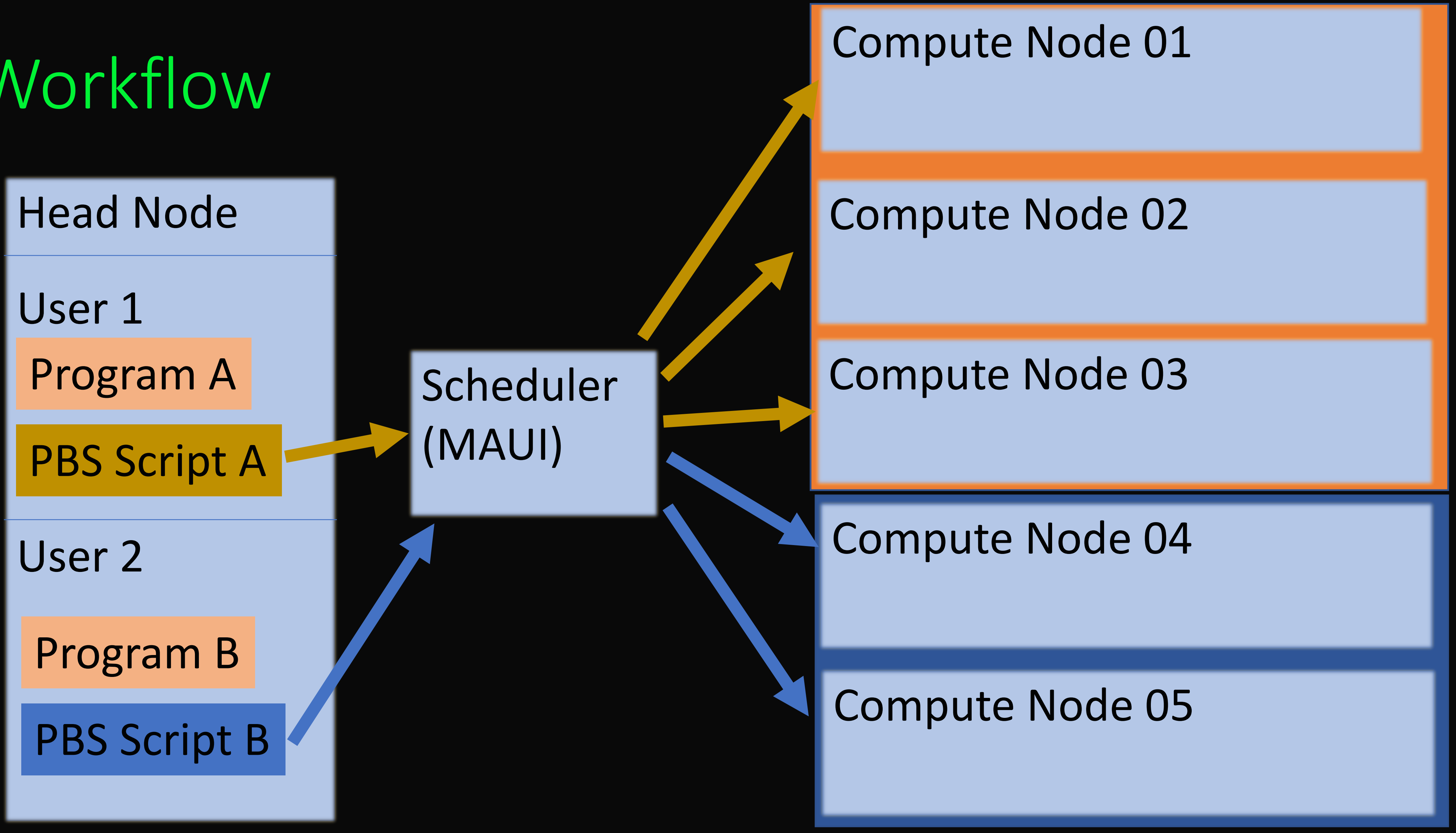

Head Node
User 1
Program A
PBS Script A
User 2
Program B
PBS Script B
Compute Node 01
Compute Node 02
Compute Node 03
Compute Node 04
Compute Node 05
Shared filesystems – All nodes can access the same programs and write output

Workflow



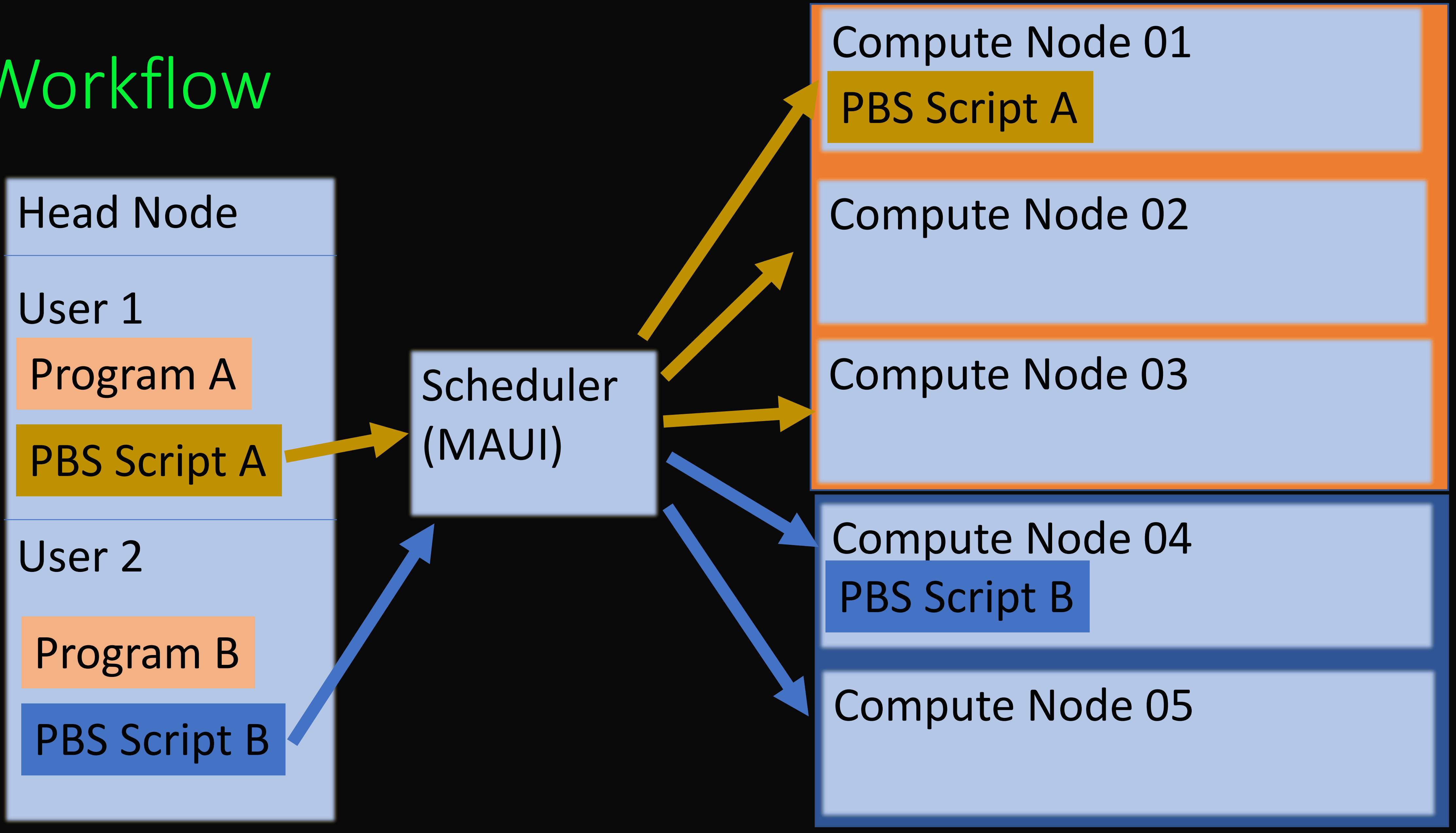
Shared filesystems – All nodes can access the same programs and write output

Workflow



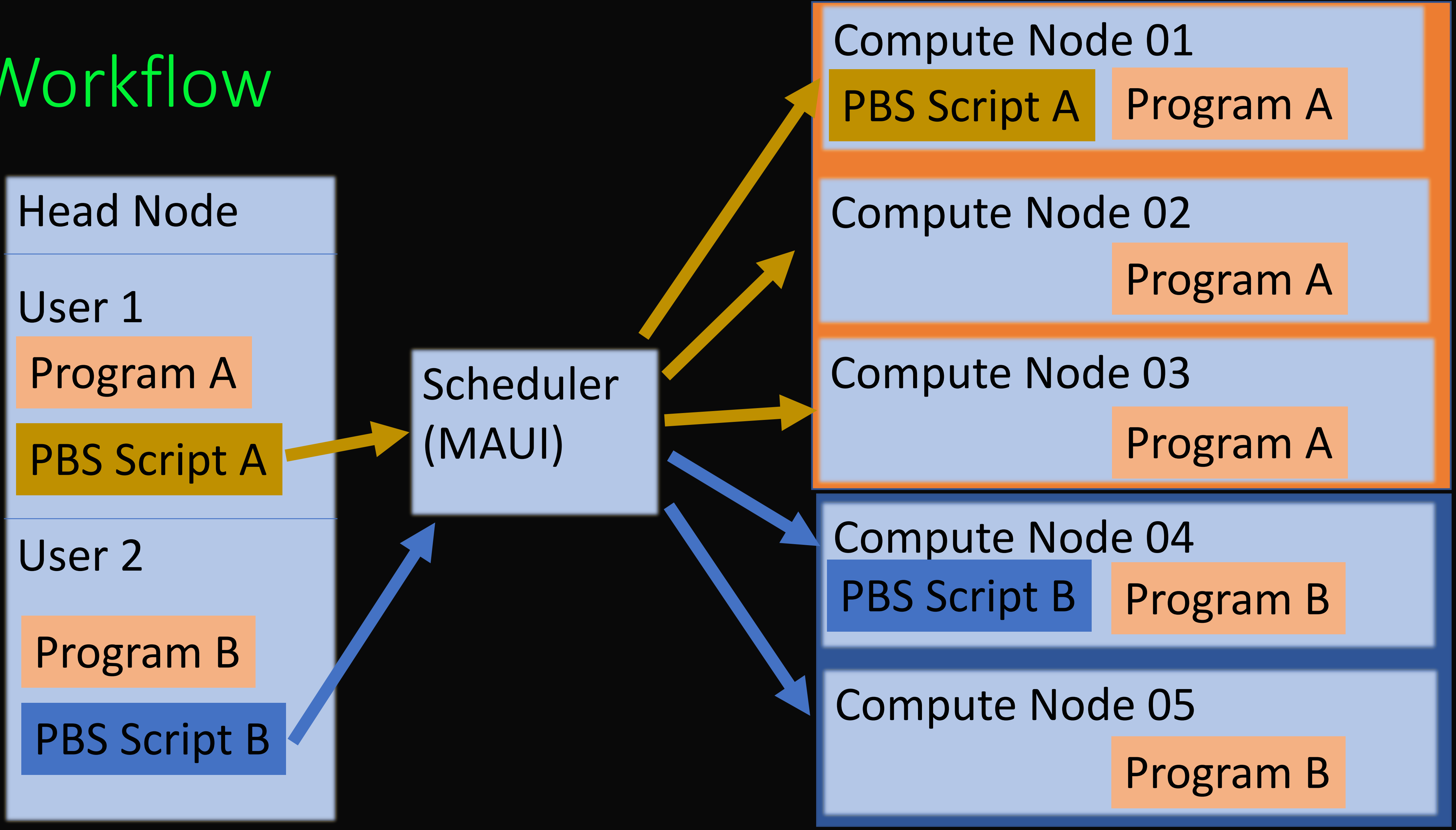
Shared filesystems – All nodes can access the same programs and write output

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=2:ppn=8
```

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

- 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 π

Serial Calculation of π

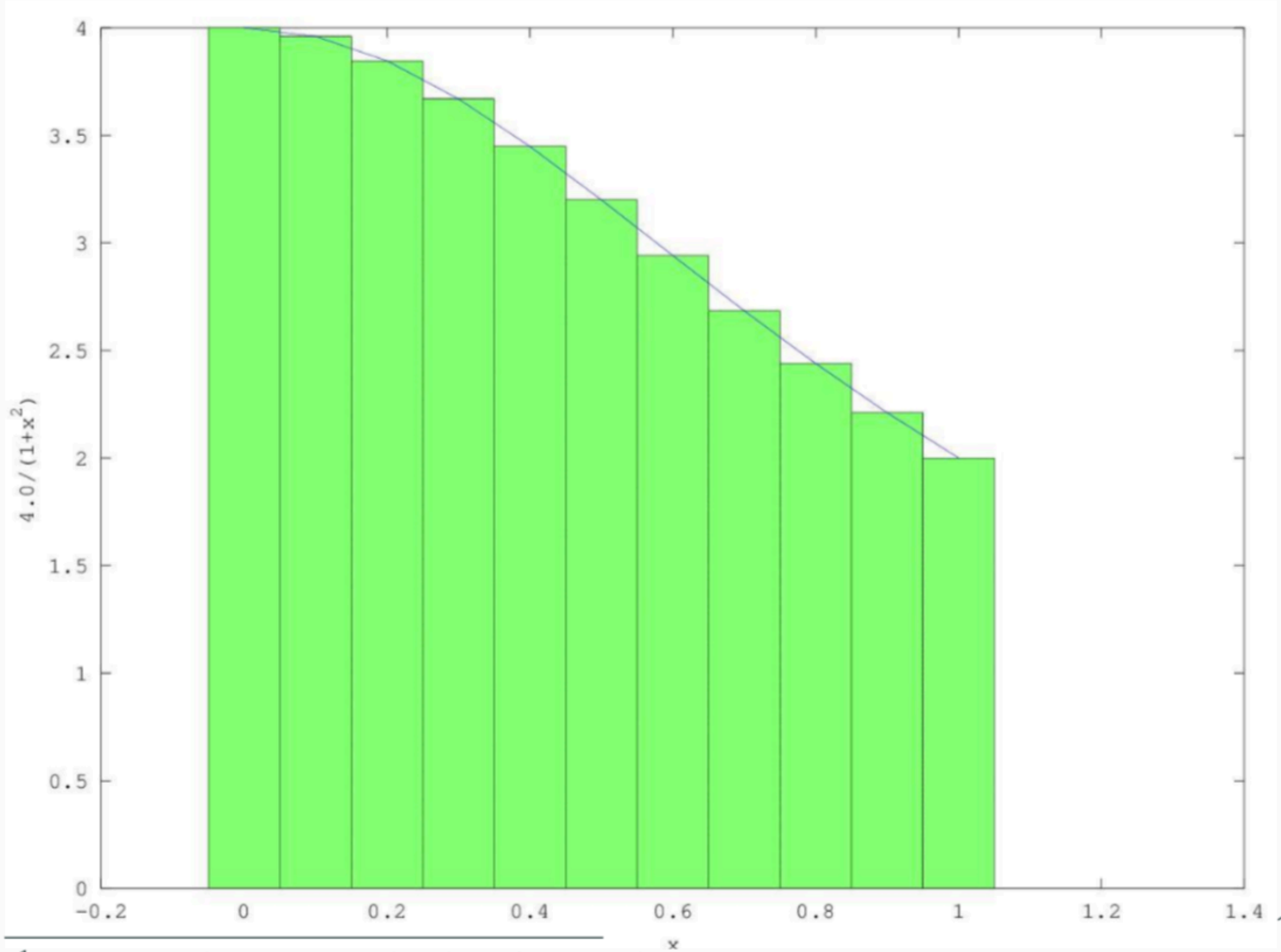
$$\int_0^1 \frac{4}{1+x^2} dx = \pi$$

And can be numerically approximated with:

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

As Δx gets smaller and N gets larger the approximation converges on π

Serial Program to Calculate π



[1]

[1]Sung Bae, Ph.D, New Zealand eScience Infrastructure

```
# 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 -l nodes=2:ppn=8
#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 10000000
```

```
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 Parallels – Monitoring Progress

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

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

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

GNU Parallels – Resuming Jobs

- With a joblog GNU **parallel** can be stopped and later pickup where it left off. It it 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 Parallels – 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 Parallels

- 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 Parallels

- 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, parallels makes use of them. You have to use

```
--sshloginfile $PBS_NODEFILE
```

- To be sure parallels is using resources you were actually allocated

GNU Parallels – 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 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
```

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
```

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

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.

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 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
```

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

... and referred to with {2}.

```
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 --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 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
```

Specifies the number of jobs to run on each node.

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.

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 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
```

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 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
```

Notice "::::" rather than ":::". Four colons tells parallels 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.

JupyterHub

- https://wheeler.alliance.unm.edu

15 Min Break

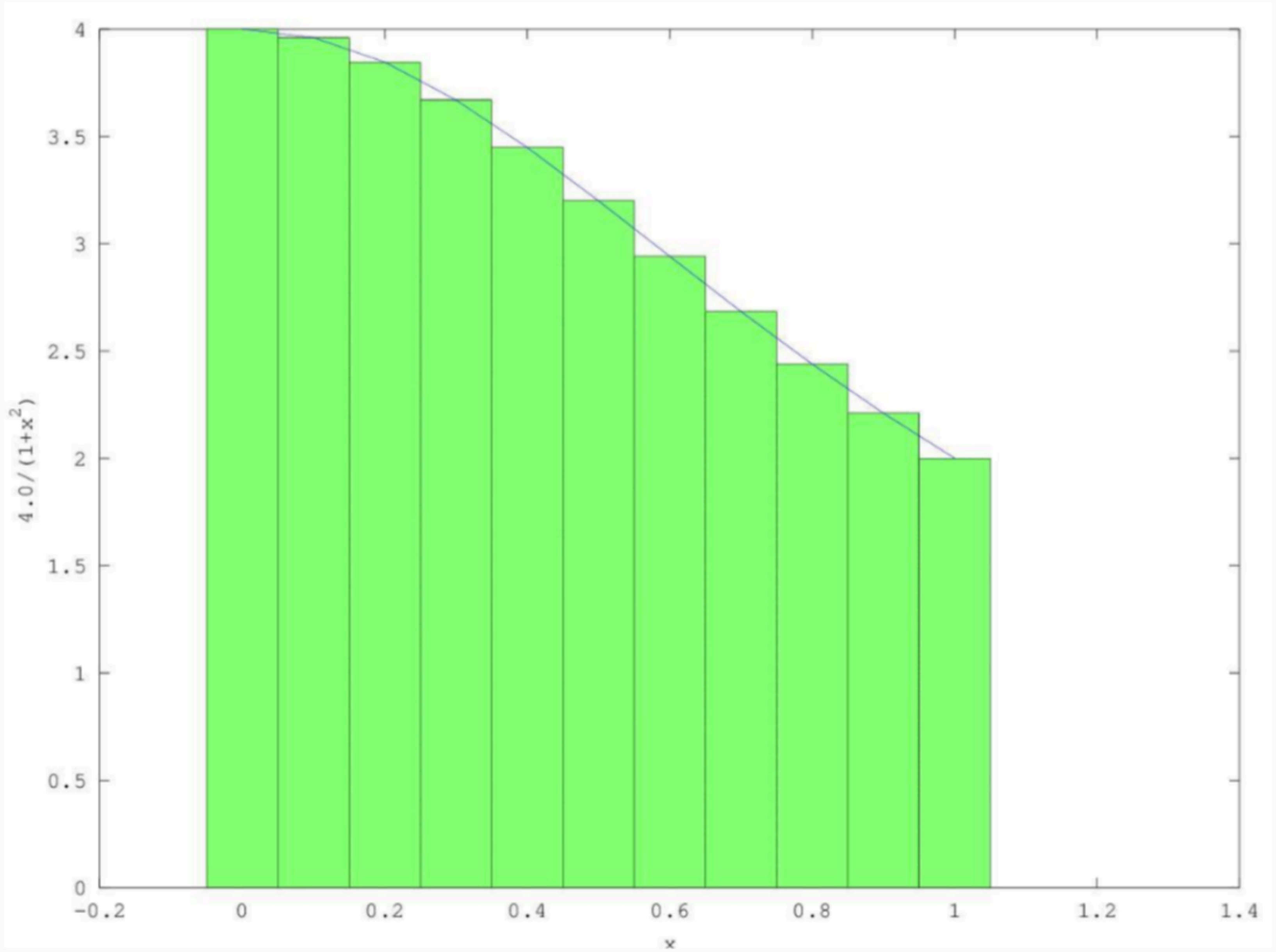
File Systems

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

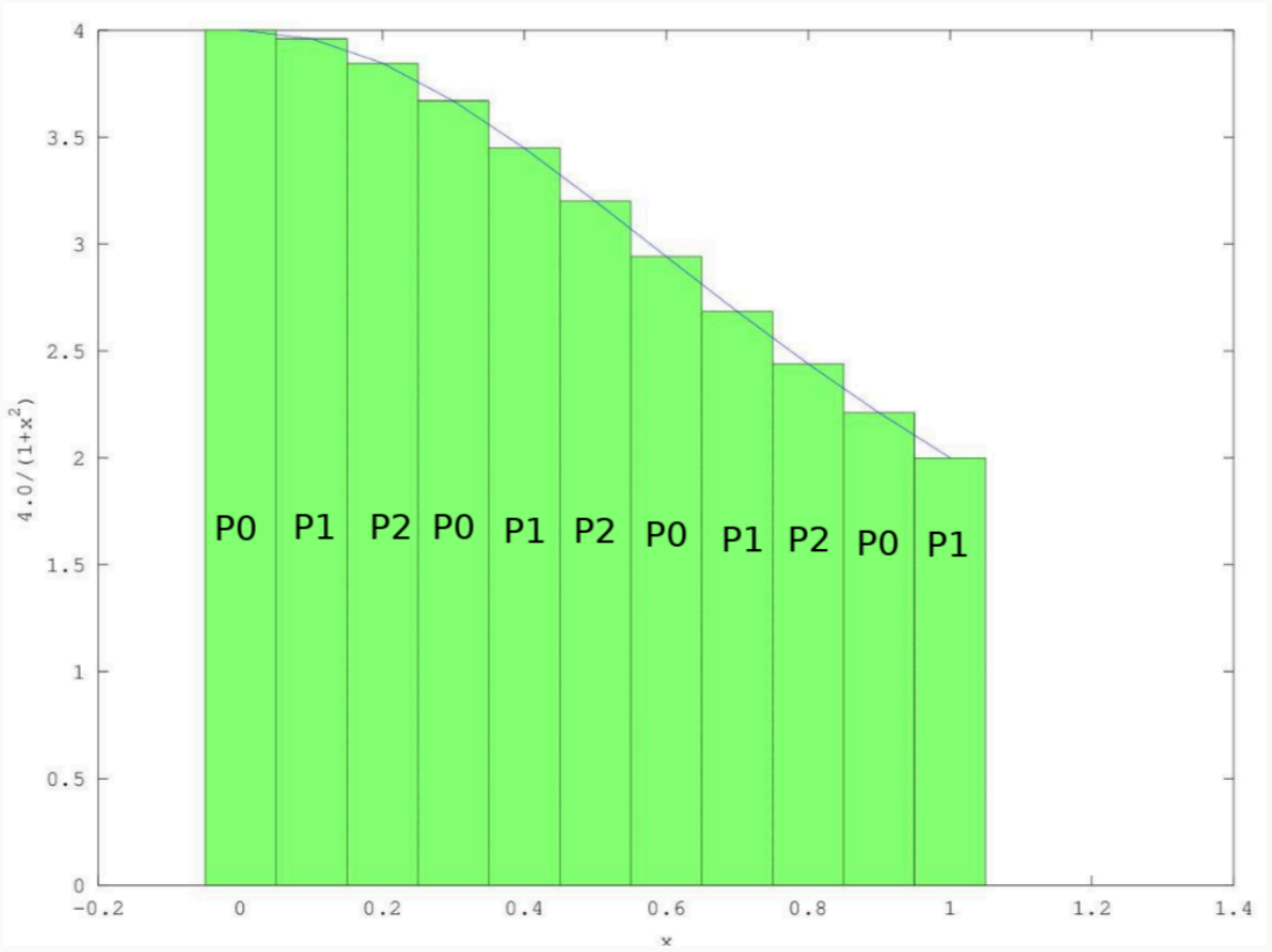
Scratch (~/wheeler-scratch): Fast. Lots of space (up to 1 TB usage), store data that you can
regenerate here.

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

Serial Program to Calculate π



Parallel Program to Calculate π



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 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
If root==rank:
    print("Pi = %.20f, (Diff=%.20f) (calculated in %f secs wit
%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 10000000
```

```
qsub pbs/calc_pi_mpi.pbs
```

We are here to help you!

help@carc.unm.edu