Intermediate Level Introduction to Computing at CARC

Second in a three-part series – 1hr version

b

Matthew Fricke

Version 0.1b

Goals

- 1) SLURM scheduler literacy
- 2) Message Passing Interface (MPI)

- We won’t cover file transfer, storage systems, module system, conda, PBS. (These are all covered in depth in the video tutorials)

Logging into Hopper

First login to the Linux **workstation** in front of you.

Use your CARC username and password.

This is an “important step” so don’t let me move on until you have logged in

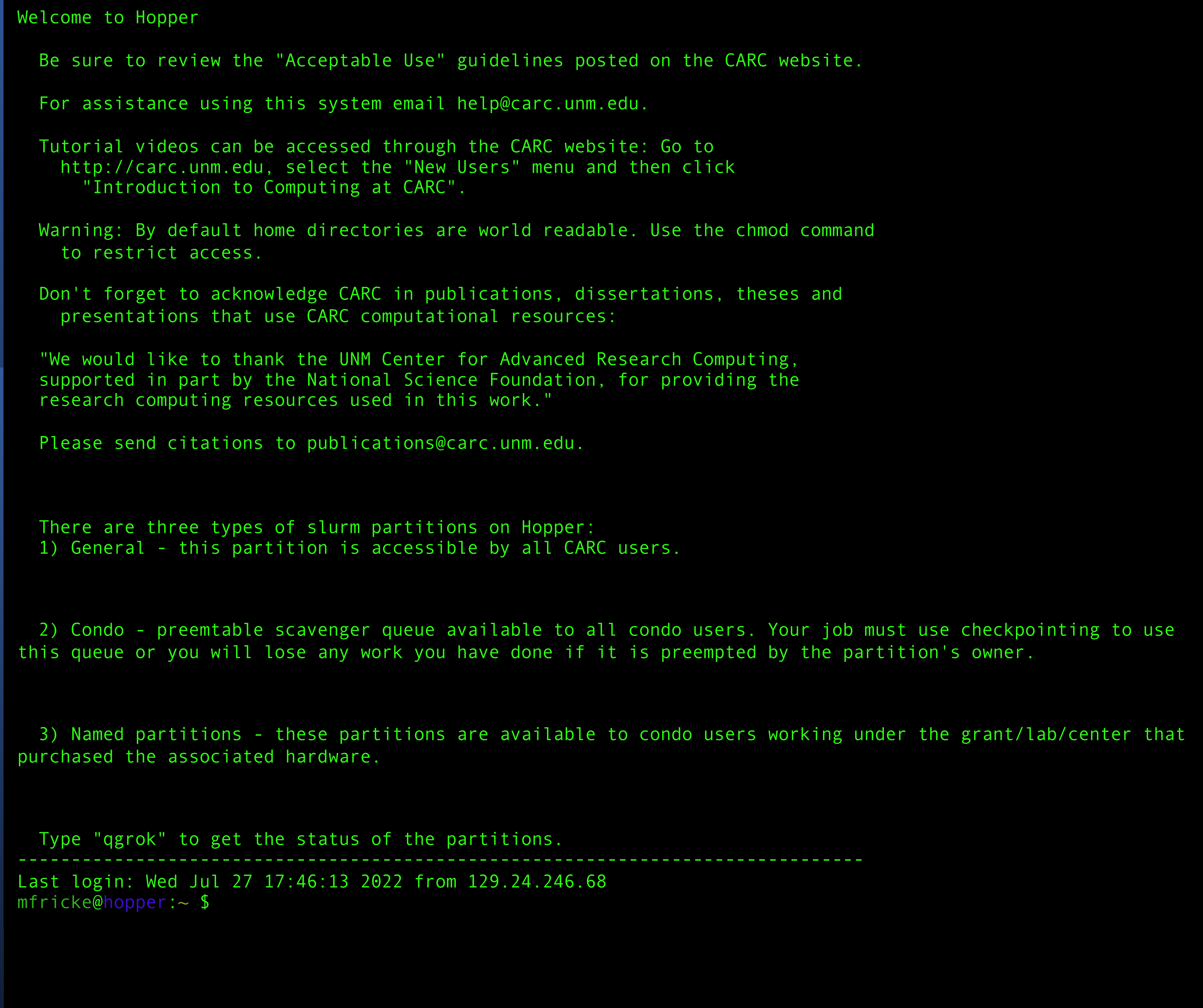
Logging into Hopper

```
ssh vanilla@hopper.alliance.unm.edu
```

Should prompt you for a password…

Don’t let me move on until you are able to login.

Logging into Hopper



```
Welcome to Hopper

  Be sure to review the "Acceptable Use" guidelines posted on the CARC website.

  For assistance using this system email help@carc.unm.edu.

  Tutorial videos can be accessed through the CARC website: Go to
    http://carc.unm.edu, select the "New Users" menu and then click
      "Introduction to Computing at CARC".

  Warning: By default home directories are world readable. Use the chmod command
    to restrict access.

  Don't forget to acknowledge CARC in publications, dissertations, theses and
    presentations that use CARC computational resources:

  "We would like to thank the UNM Center for Advanced Research Computing,
  supported in part by the National Science Foundation, for providing the
  research computing resources used in this work."

  Please send citations to publications@carc.unm.edu.



  There are three types of slurm partitions on Hopper:
  1) General - this partition is accessible by all CARC users.



  2) Condo - preemtable scavenger queue available to all condo users. Your job must use checkpointing to use
this queue or you will lose any work you have done if it is preempted by the partition's owner.



  3) Named partitions - these partitions are available to condo users working under the grant/lab/center that
purchased the associated hardware.



  Type "qgrok" to get the status of the partitions.
-------------------------------------------------------------------------------
Last login: Wed Jul 27 17:46:13 2022 from 129.24.246.68
mfricke@hopper:~ $
```

Technology, IT etc.

SLURM

means

Simple Linux Utility for Resource Management

by acronymsandslang.com

ENJOY
Slurm
SODA
IT'S HIGHLY ADDICTIVE!
VOTED #1 SOFT DRINK OF THE 31ST CENTURY!
Slurm
YOU WIN!
WHIMMY WHAM WHAM WOZZLE!
SLURMS McKENZIE

```
[vanilla@hopper~] $ qgrok
queues     free    busy    offline   jobs    nodes   CPUs   CPUs/node   Memory/node   time_limit    CPU_limit   RAM_limit
-----       -----   -----   -----     -----   -----   -----    -----        -----         -----         -----       -----
general     3       7       0         5       10      320         32          93G        2-00:00:00     128         380000M
debug       1       1       0         1       2       64          32          93G           4:00:00       8          25000M
```

Compute Nodes are divided into partitions with different limits

```
[vanilla@hopper~] $ qgrok
queues    free   busy   offline  jobs   nodes  CPUs  CPUs/node  Memory/node  time_limit   CPU_limit  RAM_limit
-----      -----  -----  -----    -----  -----  -----   -----      -----       -----        -----     -----
general    3      7      0        5      10     320        32       93G       2-00:00:00    128       380000M
debug      1      1      0        1      2      64         32       93G          4:00:00      8        25000M
```

The general partition has high limits so you generally have to wait a little bit to run.

For debugged jobs.

```
[vanilla@hopper~] $ qgrok
queues     free    busy    offline   jobs    nodes   CPUs   CPUs/node   Memory/node   time_limit   CPU_limit  RAM_limit
-----       -----   -----   -----     -----   -----   -----    -----        -----         -----        -----      -----
general     3       7       0         5       10      320         32         93G      2-00:00:00     128        380000M
debug       1       1       0         1       2       64          32         93G         4:00:00       8         25000M
```

The debug partition has low limits so generally there is no wait.

For debugging jobs.
For interactive jobs.

```
[vanilla@hopper ~]$ sinfo --partition debug
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
debug        up    4:00:00      2   idle hopper[011-012]
```

sinfo reports information about partitions

```
[vanilla@hopper ~]$ sinfo --partition debug
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
debug        up    4:00:00      2   idle hopper[011-012]
```

The debug queues are intended for testing your programs.

And for interactive jobs.

```
[vanilla@hopper ~]$ sinfo --partition debug
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
debug        up    4:00:00      2   idle hopper[011-012]
```

Name

```
[vanilla@hopper ~]$ sinfo --partition debug
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
debug        up    4:00:00      2   idle hopper[011-012]
```

You can run a “job” for up to 4 hrs.

```
[vanilla@hopper ~]$ sinfo --partition debug
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
debug        up    4:00:00      2   idle hopper[011-012]
```

There are two nodes in this partition.

```
[vanilla@hopper ~]$ sinfo --partition debug
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
debug        up    4:00:00      2   idle hopper[011-012]
```

The names of the nodes in the partition

```
[vanilla@hopper ~]$ sinfo --partition debug
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
debug        up    4:00:00      2   idle hopper[011-012]
```

The names of the nodes in the partition

```
[vanilla@hopper ~]$ sinfo --partition general
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
general*     up 2-00:00:00      9  alloc hopper[001-009]
general*     up 2-00:00:00      1   idle hopper010
```

Hopper001 through 009 are running jobs.

Hopper010 is waiting to be used.

```
[vanilla@hopper ~]$ hostname
hopper
[vanilla@hopper ~]$
```

Running on the Head Node.

The head node’s name is “hopper”.

```
[vanilla@hopper ~]$ hostname
hopper
[vanilla@hopper ~]$ man hostname
```

```
[vanilla@hopper ~]$ hostname
hopper
[vanilla@hopper ~]$ man hostname
(‘q’ to quit)

[vanilla@hopper ~]$ man man
(‘q’ to quit)
```

```
[vanilla@hopper ~]$ sinfo --all
```

```
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
general*     up 2-00:00:00      9  alloc hopper[001-009]
general*     up 2-00:00:00      1   idle hopper010
debug        up    4:00:00      2   idle hopper[011-012]
condo        up 2-00:00:00      1  down* hopper045
condo        up 2-00:00:00      3    mix hopper[018-020]
condo        up 2-00:00:00     16  alloc hopper[013-015,028-036,049-052]
condo        up 2-00:00:00     18   idle hopper[016-017,021-027,037-044,053]
bugs         up 7-00:00:00      2  alloc hopper[013-014]
pcnc         up 7-00:00:00      1  alloc hopper015
pcnc         up 7-00:00:00      1   idle hopper016
pathogen     up 7-00:00:00      1   idle hopper017
tc           up 7-00:00:00      3    mix hopper[018-020]
tc           up 7-00:00:00      2  alloc hopper[029-030]
tc           up 7-00:00:00      5   idle hopper[021-025]
gold         up 7-00:00:00      2   idle hopper[026-027]
fishgen      up 7-00:00:00      1  alloc hopper028
neuro-hsc    up 7-00:00:00      6  alloc hopper[031-036]
neuro-hsc    up 7-00:00:00      8   idle hopper[037-044]
cup-ecs      up 7-00:00:00      2  alloc hopper[049-050]
tid          up 7-00:00:00      1  alloc hopper051
biocomp      up 7-00:00:00      1  alloc hopper052
chakra       up 7-00:00:00      1   idle hopper053
pna          up 7-00:00:00      1  down* hopper045
```

```
[vanilla@hopper ~]$ srun --partition debug hostname
```

Tell slurm to run a program
on a compute node…

```
[vanilla@hopper ~]$ srun --partition debug hostname
```

Run the program on a compute node in the debug partition.

```
[vanilla@hopper ~]$ srun --partition debug hostname
```

The program to run.

```
[vanilla@hopper ~]$ srun --partition debug hostname
srun: Account not specified in script or
~/.default_slurm_account, using latest project
You have not been allocated GPUs. To request GPUs,
use the -G option in your submission script.
hopper011
```

```
[vanilla@hopper ~]$ squeue
```

```
[vanilla@hopper ~]$ squeue
 JOBID PARTITION     NAME     USER ST       TIM
 4314    general     PRE erowland PD       0:00
 4315    general     PRE erowland PD       0:00
 4317    general     PRE erowland PD       0:00
 4318    general     PRE erowland PD       0:00      2 (QOSMaxCpuPerUserLimit)
              4319    general     PRE erowland PD       0:00      2 (QOSMaxCpuPerUserLimit)
                                                                  2 (QOSMaxCpuPerUserLimit)
                                                                  2 (QOSMaxCpuPerUserLimit)
                                                                  2 (QOSMaxCpuPerUserLimit)
                                                                  2 (QOSMaxCpuPerUserLimit)
                                                                  2 (QOSMaxCpuPerUserLimit)
                                                                  2 (QOSMaxCpuPerUserLimit)
                                                                  2 (QOSMaxCpuPerUserLimit)
                                                                  2 (QOSMaxCpuPerUserLimit)
                                                                  2 (QOSMaxCpuPerUserLimit)
                                                                  2 (QOSMaxCpuPerUserLimit)
                                                                  2 (QOSMaxCpuPerUserLimit)
                                                                  2 (QOSMaxCpuPerUserLimit)
                                                                  2 (QOSMaxCpuPerUserLimit)
                                                                  2 (QOSMaxCpuPerUserLimit)
                                                                  2 (QOSMaxCpuPerUserLimit)
                                                                  2 (QOSMaxCpuPerUserLimit)
              4337    general     PRE erowland PD       0:00      2 (QOSMaxCpuPerUserLimit)
```

PD means programs that are waiting their turn.

Shows you what the slurm scheduler is doing right now.

Here we can see that user ’erowland’ has a lot of programs waiting to run.

```
[vanilla@hopper ~]$ squeue
  JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
               4314   general       PRE erowland PD       0:00      2 (QOSMaxCpuPerUserLimit)
               4315   general       PRE erowland PD       0:00      2 (QOSMaxCpuPerUserLimit)
               4317   general       PRE erowland PD       0:00      2 (QOSMaxCpuPerUserLimit)
               4318   general       PRE erowland PD       0:00      2 (QOSMaxCpuPerUserLimit)
               4319   general       PRE erowland PD       0:00      2 (QOSMaxCpuPerUserLimit)
                                                                    2 (QOSMaxCpuPerUserLimit)
                                                                    2 (QOSMaxCpuPerUserLimit)
                                                                    2 (QOSMaxCpuPerUserLimit)
                                                                    2 (QOSMaxCpuPerUserLimit)
                                                                    2 (QOSMaxCpuPerUserLimit)
                                                                    2 (QOSMaxCpuPerUserLimit)
                                                                    2 (QOSMaxCpuPerUserLimit)
                                                                    2 (QOSMaxCpuPerUserLimit)
                                                                    2 (QOSMaxCpuPerUserLimit)
                                                                    2 (QOSMaxCpuPerUserLimit)
                                                                    2 (QOSMaxCpuPerUserLimit)
                                                                    2 (QOSMaxCpuPerUserLimit)
                                                                    2 (QOSMaxCpuPerUserLimit)
                                                                    2 (QOSMaxCpuPerUserLimit)
                                                                    2 (QOSMaxCpuPerUserLimit)
                                                                    2 (QOSMaxCpuPerUserLimit)
               4337   general       PRE erowland PD       0:00      2 (QOSMaxCpuPerUserLimit)
```

The reason these jobs are not running is that ‘erowland’ is already using the maximum number of CPUs they are allowed.

```
[vanilla@hopper ~]$ squeue -t R --all
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
              4405     condo    2ndMA  mfricke  R 1-07:48:30      6 hopper[031-036]
              5208     condo       NN      kgu  R    5:48:49      1 hopper015
              5210     condo       NN      kgu  R    6:30:13      1 hopper014
              5209     condo       NN      kgu  R    6:31:13      1 hopper013
              5206     condo       NN      kgu  R    6:32:13      1 hopper051
              5207     condo       NN      kgu  R    6:32:13      1 hopper052
              5205     condo       NN      kgu  R    6:32:43      1 hopper028
              4595   cup-ecs golConfi aalasand  R 2-06:51:59      1 hopper050
              4594   cup-ecs golConfi aalasand  R 2-06:52:03      1 hopper049
              5120   general jupyterh   jacobm  R   11:45:47      1 hopper007
              4313   general      PRE erowland  R    1:17:29      2 hopper[003-004]
              5111   general    1stMA  mfricke  R   11:15:28      2 hopper[005-006]
              5025   general      c2n    jxzuo  R       1:50      1 hopper001
              5024   general      c2n    jxzuo  R      31:28      1 hopper002
              5203   general       NN      kgu  R    6:37:50      1 hopper009
              5201   general       NN      kgu  R    6:38:14      1 hopper008
              4390        tc UCsTpCyd lepluart  R 2-15:18:18      3 hopper[018-020]
              5198        tc       NN      kgu  R    6:40:19      1 hopper030
              5196        tc       NN      kgu  R    6:40:31      1 hopper029
```

```
[vanilla@hopper ~]$ srun --partition debug --ntasks 2 hostname
srun: Account not specified in script or
~/.default_slurm_account, using latest project
You have not been allocated GPUs. To request GPUs, use the -G
option in your submission script.
hopper011
hopper011
```

```
[vanilla@hopper ~]$ srun --partition debug --ntasks 2 hostname
srun: Account not specified in script or
~/.default_slurm_account, using latest project
You have not been allocated GPUs. To request GPUs, use the -G
option in your submission script.
hopper011
hopper011
```

You ran two copies of your program.

ntasks is the number of copies to run.

```
[vanilla@hopper ~]$ srun --partition debug --ntasks 8 hostname
srun: Account not specified in script or
~/.default_slurm_account, using latest project
hopper011
hopper011
hopper011
You have not been allocated GPUs. To request GPUs, use the -G
option in your submission script.
hopper011
hopper011
hopper011
hopper011
hopper011
```

You ran eight copies of your program.

ntasks is the number of copies to run.

```
[vanilla@hopper ~]$ srun --partition debug --ntasks 8 hostname
srun: Account not specified in script or
~/.default_slurm_account, using latest project
hopper011
hopper011
hopper011
You have not been allocated GPUs. To request GPUs, use the -G
option in your submission script.
hopper011
hopper011
hopper011
hopper011
hopper011
```

By default, each task (copy of your program) is allowed to use one CPU.

Many programs are able to use more than one CPU at a time.

```
[vanilla@hopper ~]$ srun --partition debug --ntasks 2 --cpus-per-task 2 hostname
srun: Account not specified in script or ~/.default_slurm_account, using latest project
You have not been allocated GPUs. To request GPUs, use the -G option in your submission
script.
hopper011
hopper011
```

Here we are telling SLURM to run 2 copies of our program and let each copy of our program use 2 CPUs.

```
[vanilla@hopper ~]$ srun --partition debug --nodes 2 --ntasks-per-node 4 hostname
srun: Account not specified in script or ~/.default_slurm_account, using
latest project
hopper012
You have not been allocated GPUs. To request GPUs, use the -G option in
your submission script.
hopper012
hopper011
hopper011
hopper012
hopper012
hopper011
hopper011
```

Here we are telling SLURM to run 4 copies of our program on 2 different compute nodes.

This is useful when our programs need a bigger share of the compute node.

```
[vanilla@hopper ~]$ srun --partition debug --nodes 2
--ntasks-per-node 2 --cpus-per-task 2 hostname
srun: Account not specified in script or
~/.default_slurm_account, using latest project
hopper011
You have not been allocated GPUs. To request GPUs, use
the -G option in your submission script.
hopper011
hopper012
hopper012
```

And we can combine all three.

```
[vanilla@hopper ~]$ srun --partition debug --mem 4G
--nodes 2 --ntasks-per-node 2 --cpus-per-task 2
hostname
srun: Account not specified in script or
~/.default_slurm
hopper012
hopper012
You have not be                                          se
the -G option i
hopper011
Hopper011
```

And we can specify how much memory we want.

--mem 4G means give me 4 gigabytes of memory per node.

```
[vanilla@hopper ~]$ srun --partition debug --mem 4G
--nodes 2 --ntasks-per-node 2 --cpus-per-task 2
hostname
srun: Account not specified in script or
~/.default_slurm
hopper012
hopper012
You have not be                                        se
the -G option i
hopper011
Hopper011
```

Why does all this matter?

The purpose of SLURM is to provide you the hardware your programs need.

So you have to understand what those requirements are really well.

```
[vanilla@hopper ~]$ srun --partition debug --mem 4G
--nodes 2 --ntasks-per-node 2 --cpus-per-task 2 hostname
srun: Account not specified in script or
~/.default_slurm_account, using latest project
hopper012
hopper012
You have not be                                    se
the -G option i
hopper011
Hopper011
```

1) **Can my program use multiple CPUs?**
2) **How much memory does my program need?**
3) **Can my program use multiple compute nodes (MPI*, GNU Parallel*)?**
4) **Can my program use GPUs?**

```
[vanilla@hopper ~]$ srun --partition debug --mem 4G
--nodes 2 --ntasks-per-node 2 --cpus-per-task 2 hostname
srun: Account not specified in script or
~/.default_slurm_account, using latest project
hopper012
hopper012
You have not been                                         se
the -G option in
hopper011
Hopper011
```

This command is getting pretty long.

We can use salloc to avoid asking for the same resources every time we use srun.

```
[vanilla@hopper ~]$ salloc --partition debug --nodes 2
--ntasks-per-node 2
salloc: Account not specified in script or
~/.default_slurm_account, using latest project
salloc: Granted job allocation 5251
salloc: Waiting for resource configuration
salloc: Nodes hopper[011-012] are ready for job
[vanilla@hopper ~]$
```

This command is getting pretty long.

We can use salloc to avoid asking for the same resources every time we use srun.

```
[vanilla@hopper ~]$ srun hostname
hopper012
hopper012
hopper011
You have not been allocated GPUs. To request GPUs, use
the -G option in your submission script.
hopper011
[vanilla@hopper ~]$ srun hostname
hopper012
hopper011
hopper012
hopper011
[vanilla@hopper ~]$
```

Now we can use srun over and over without having to ask for a new hardware allocation each time.

```
[vanilla@hopper ~]$ exit
exit
salloc: Relinquishing job allocation 5251
```

Always type exit when you are done with the hardware.
Running salloc inside an allocation gets very confusing.

Interactive vs Batch Mode

Interactive Mode

- Everything so far has been interactive. You request hardware, run your program, and get the output on your screen right away.

Batch Mode

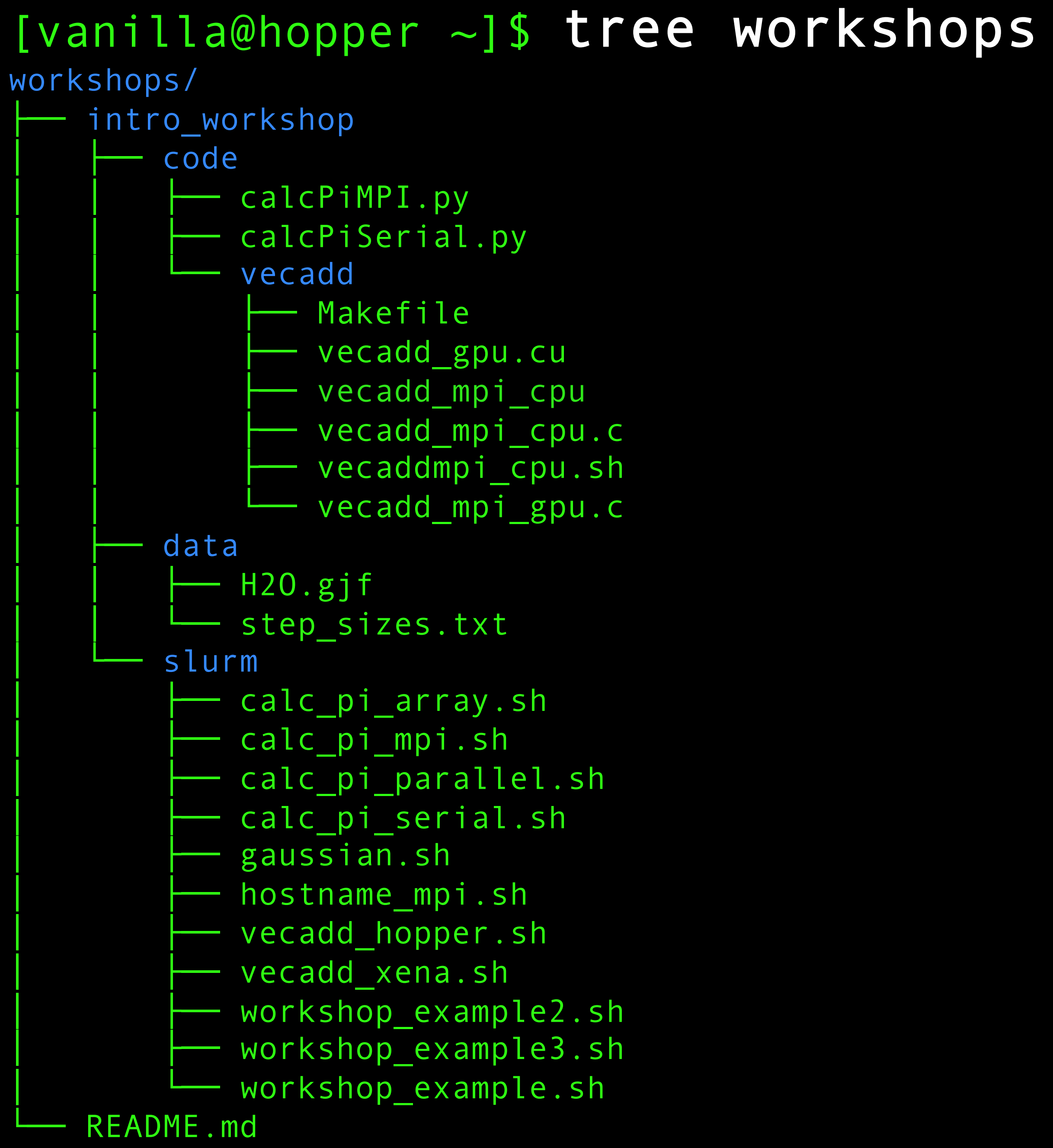
- Most programs at an HPC center are run in “batch” mode.
- Batch mode means we write a shell script that the SLURM scheduler runs for us. The script requests hardware just like we did with salloc and then runs the commands in the script.
- Whatever would have been written to the screen is saved to a file instead.

```
[vanilla@hopper ~]$ git clone https://lobogit.unm.edu/CARC/workshops.git
Cloning into 'workshops'...
remote: Enumerating objects: 132, done.
remote: Counting objects: 100% (75/75), done.
remote: Compressing objects: 100% (43/43), done.
remote: Total 132 (delta 33), reused 74 (delta 32), pack-reused 57
Receiving objects: 100% (132/132), 57.58 KiB | 3.60 MiB/s, done.
Resolving deltas: 100% (51/51), done.
```

Rather than make you write shell scripts lets just download some we wrote for this workshop…

```
[vanilla@hopper ~]$ tree workshops
workshops/
├── intro_workshop
│   ├── code
│   │   ├── calcPiMPI.py
│   │   ├── calcPiSerial.py
│   │   └── vecadd
│   │       ├── Makefile
│   │       ├── vecadd_gpu.cu
│   │       ├── vecadd_mpi_cpu
│   │       ├── vecadd_mpi_cpu.c
│   │       ├── vecaddmpi_cpu.sh
│   │       └── vecadd_mpi_gpu.c
│   ├── data
│   │   ├── H2O.gjf
│   │   └── step_sizes.txt
│   └── slurm
│       ├── calc_pi_array.sh
│       ├── calc_pi_mpi.sh
│       ├── calc_pi_parallel.sh
│       ├── calc_pi_serial.sh
│       ├── gaussian.sh
│       ├── hostname_mpi.sh
│       ├── vecadd_hopper.sh
│       ├── vecadd_xena.sh
│       ├── workshop_example2.sh
│       ├── workshop_example3.sh
│       └── workshop_example.sh
└── README.md
```

Run tree to see how the workshops directories are organized…



```
[vanilla@hopper ~]$ tree workshops
workshops/
├── intro_workshop
│   ├── code
│   │   ├── calcPiMPI.py
│   │   ├── calcPiSerial.py
│   │   └── vecadd
│   │       ├── Makefile
│   │       ├── vecadd_gpu.cu
│   │       ├── vecadd_mpi_cpu
│   │       ├── vecadd_mpi_cpu.c
│   │       ├── vecaddmpi_cpu.sh
│   │       └── vecadd_mpi_gpu.c
│   ├── data
│   │   ├── H2O.gjf
│   │   └── step_sizes.txt
│   └── slurm
│       ├── calc_pi_array.sh
│       ├── calc_pi_mpi.sh
│       ├── calc_pi_parallel.sh
│       ├── calc_pi_serial.sh
│       ├── gaussian.sh
│       ├── hostname_mpi.sh
│       ├── vecadd_hopper.sh
│       ├── vecadd_xena.sh
│       ├── workshop_example2.sh
│       ├── workshop_example3.sh
│       └── workshop_example.sh
└── README.md
```

Run **tree** to see how the workshops directories are organized…

The workshop files are divided into “code”, “slurm”, and “data” directories.

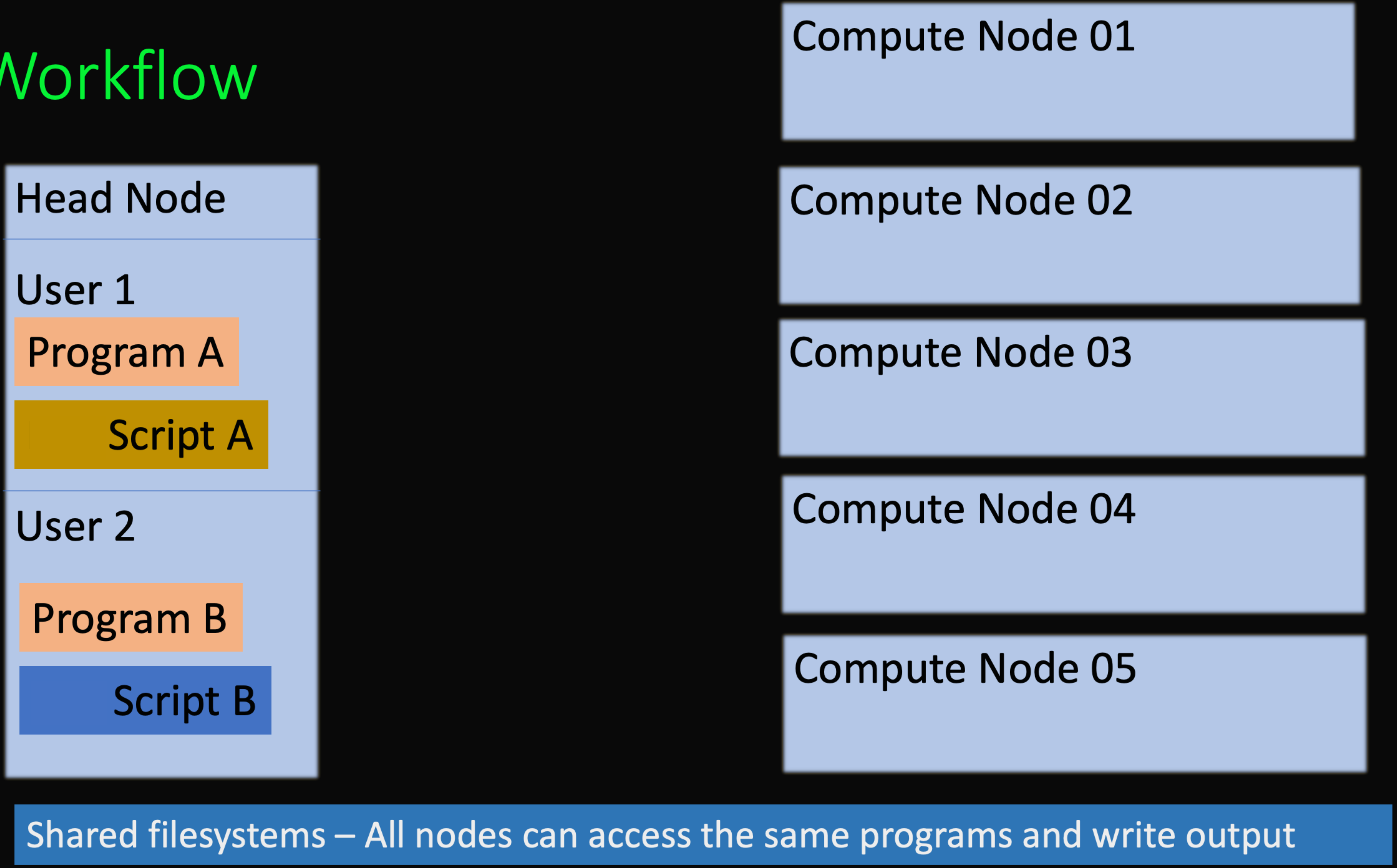
```
[vanilla@hopper ~]$ cd workshops/intro_workshop
[vanilla@hopper intro_workshop]$ pwd
/users/vanilla/workshops/intro_workshop
[vanilla@hopper intro_workshop]$ cat slurm/workshop_example1.sh
#!/bin/bash
#SBATCH --partition debug
#SBATCH --ntasks 4
#SBATCH --time 00:05:00
#SBATCH --job-name ws_example
#SBATCH --mail-user your_username@unm.edu
#SBATCH --mail-type ALL

srun hostname
```

Let's take a look at the workshop_example.sh script in the slurm directory…

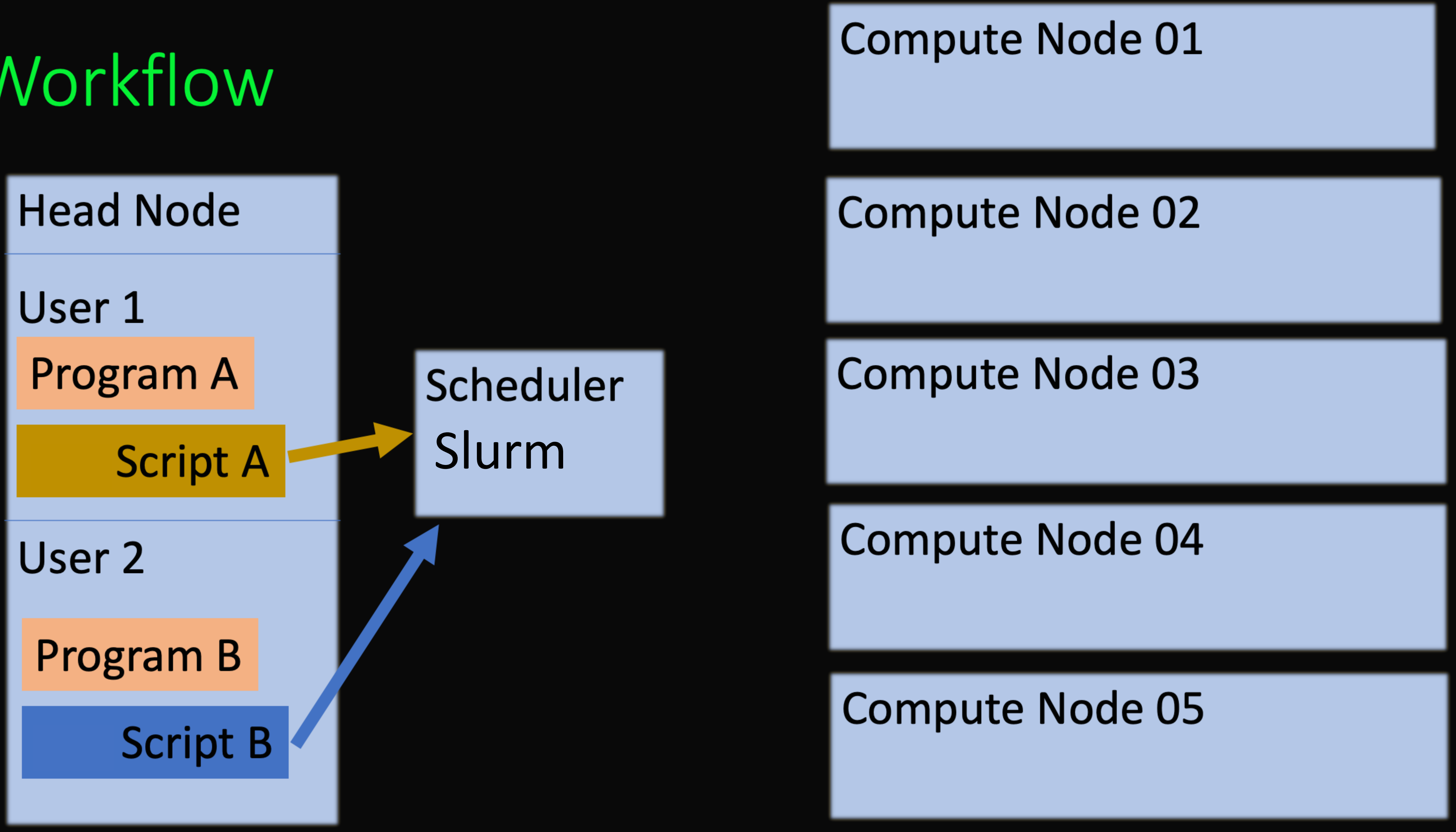
```
[vanilla@hopper intro_workshop]$ sbatch slurm/workshop_example1.sh
sbatch: Account not specified in script or ~/.default_slurm_account,
using latest project
Submitted batch job 5252
[vanilla@hopper intro_workshop]$
```

We submit our slurm shell script with the sbatch command.

Workflow

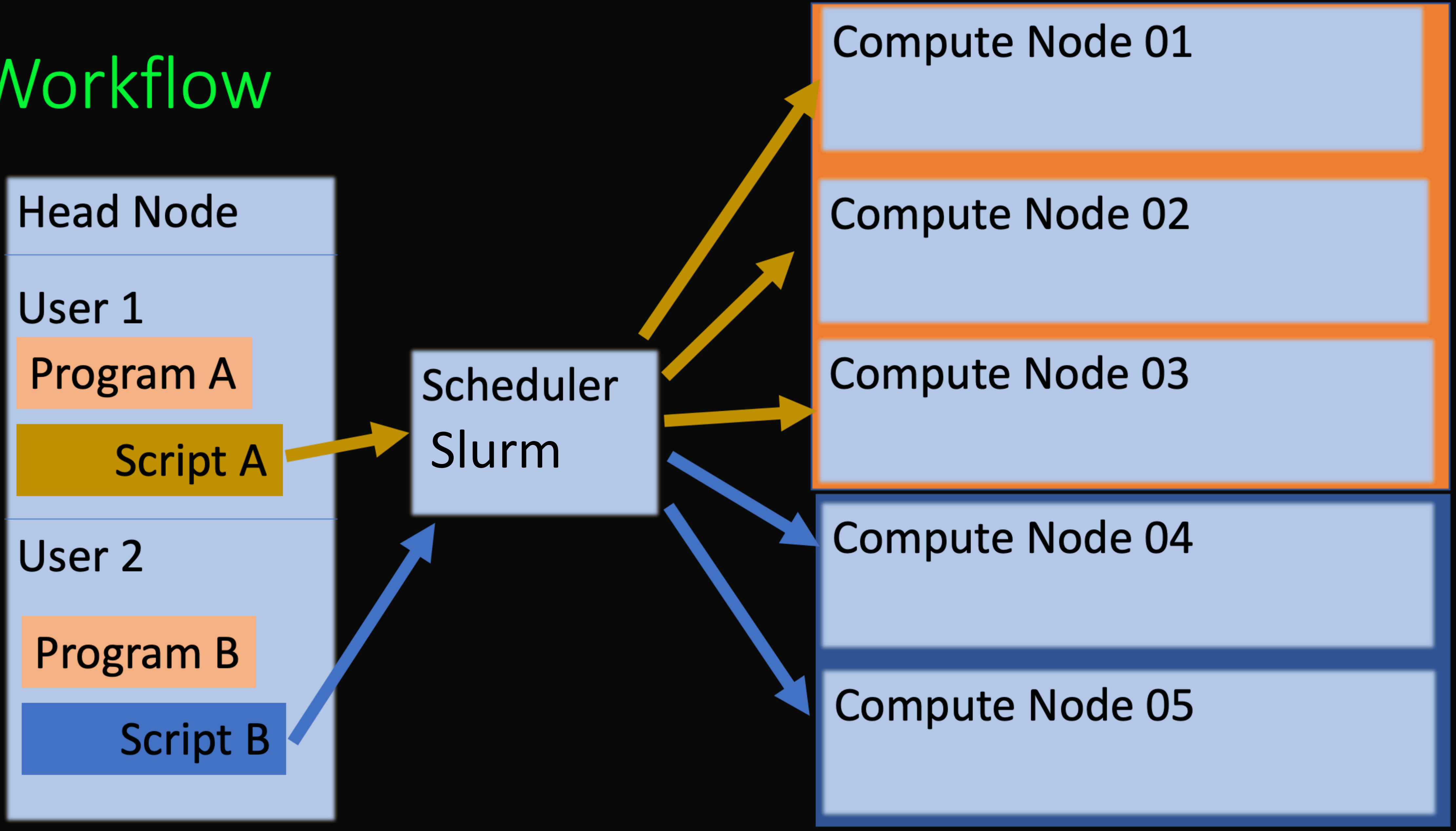

Head Node
User 1
Program A
Script A
User 2
Program B
Script B
Compute Node 01
Compute Node 02
Compute Node 03
Compute Node 04
Compute Node 05
Shared filesystems – All nodes can access the same programs and write output

Workflow



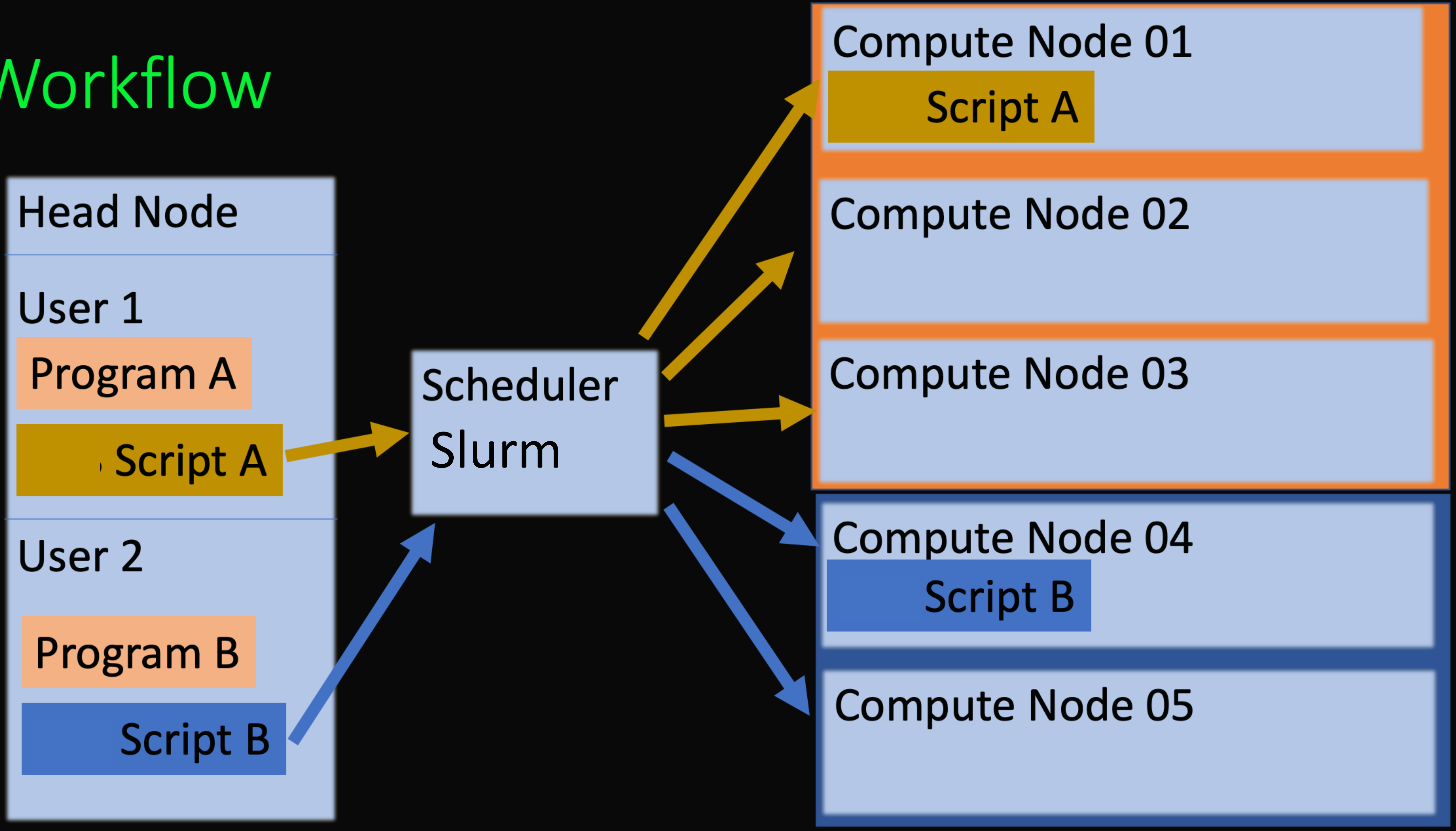
Shared filesystems – All nodes can access the same programs and write output

Workflow



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

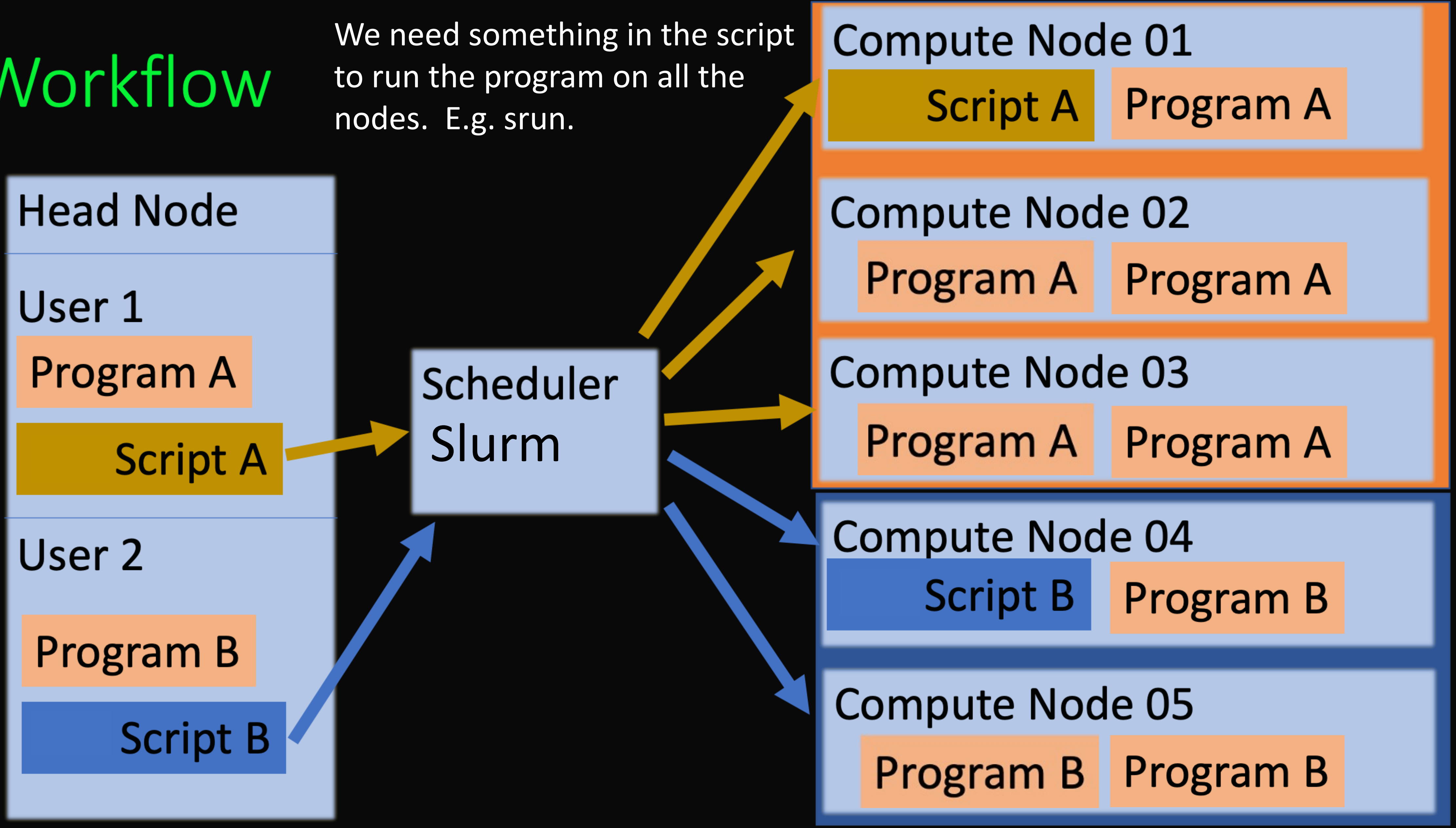
Workflow



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

Workflow

We need something in the script to run the program on all the nodes. E.g. srun.



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

```
[vanilla@hopper intro_workshop]$ ls
code  data  pbs  slurm  slurm-5252.out
```

The **hostname** command is very fast so everyone's job should finish in a few seconds.

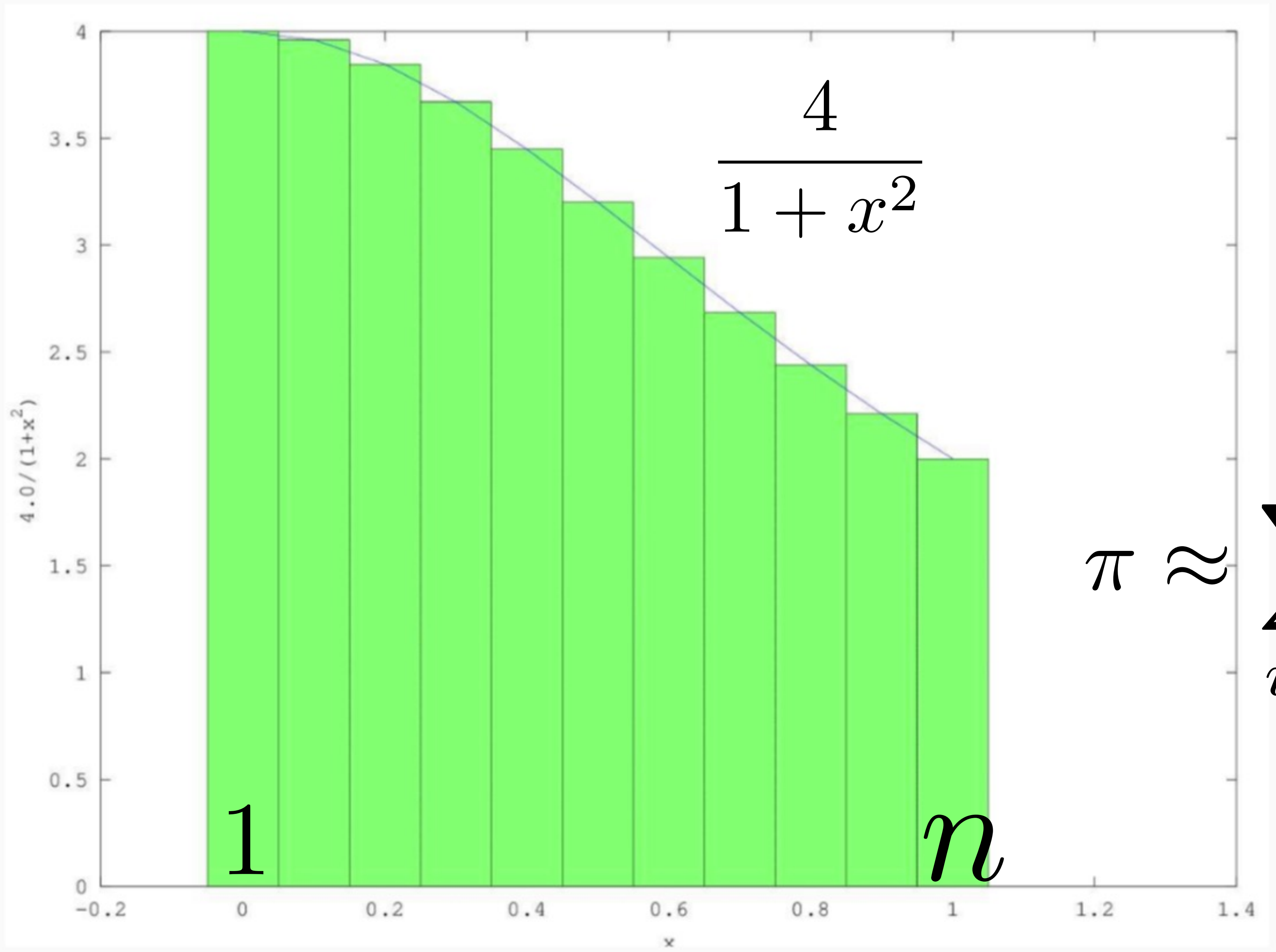
When it is finished you will have a new file named slurm-{your job id}.out.

```
[vanilla@hopper intro_workshop]$ ls
code  data  pbs  slurm  slurm-5252.out
```

When it is finished you will have a new file named slurm-{your job id}.out.

```
[vanilla@hopper intro_workshop]$ cat slurm-5252.out
hopper011
hopper011
hopper011
hopper011
```

Serial Program to Calculate π



$$\frac{4}{1+x^2}$$

1 n

$$w = \frac{1}{n}$$

$$\pi \approx \sum_{i=0}^{n} \frac{4}{1+(i+\frac{w}{2})^2}$$

```
# 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)
```

```
# 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)
```

```
[vanilla@hopper intro_workshop]$ module load miniconda3
[vanilla@hopper intro_workshop]$ conda create -n numpy numpy
```

Wait a while – introduce yourselves to your neighbor…

Conda allows you to install software into your home directory. In this case we need the numerical python libraries for calcPiSerial.py

Let’s experiment with a program that does slightly more than print the hostname.

```
[vanilla@hopper intro_workshop]$source activate numpy
[vanilla@hopper intro_workshop]$srun --partition debug python
code/calcPiSerial.py 10
srun: Using account 2016199 from ~/.default_slurm_account
You have not been allocated GPUs. To request GPUs, use the -G
option in your submission script.
Pi = 3.14242598500109870940, (Diff=0.00083333141130559341)
(calculated in 0.000005 secs with 10 steps)
```

Activate the numpy environment and
Run calcPiSerial.py on a compute node.

For our example program the more steps it takes the more accurate it is, but the longer it takes.

```
[vanilla@hopper intro_workshop]$ sbatch slurm/calc_pi_serial.sh
 sbatch: Using account 2016199 from ~/.default_slurm_account
Submitted batch job 5263
vanilla@hopper:~/workshops/intro_workshop$ squeue -me
JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
5263      debug calc_pi_  vanilla  R       0:44      1 hopper011
```

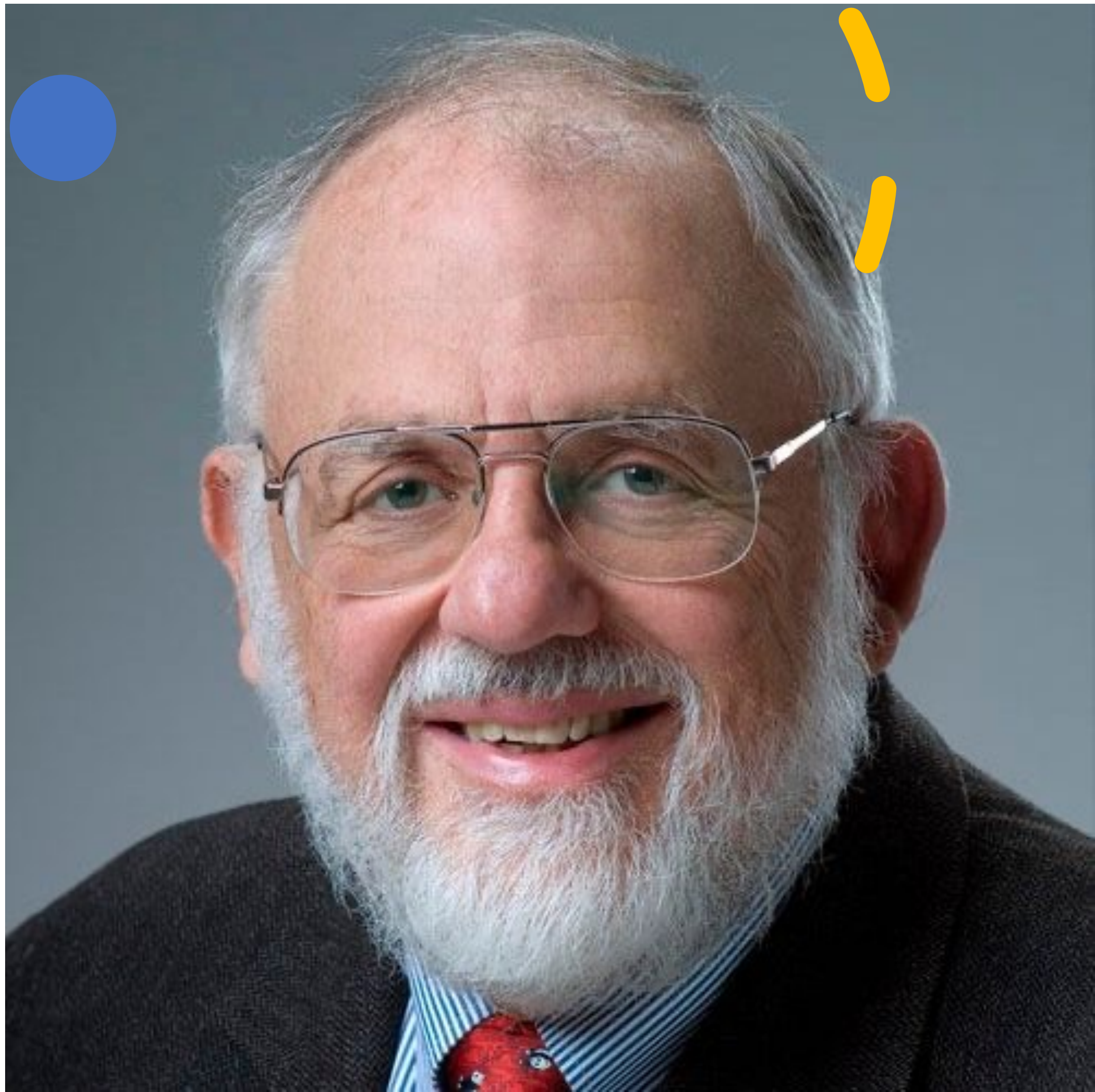
Edit slurm/calc_pi_serial.sh.
Change the email address to your address and submit the script.

Then enter squeue --me to see the job status.

Take a look at the job output.

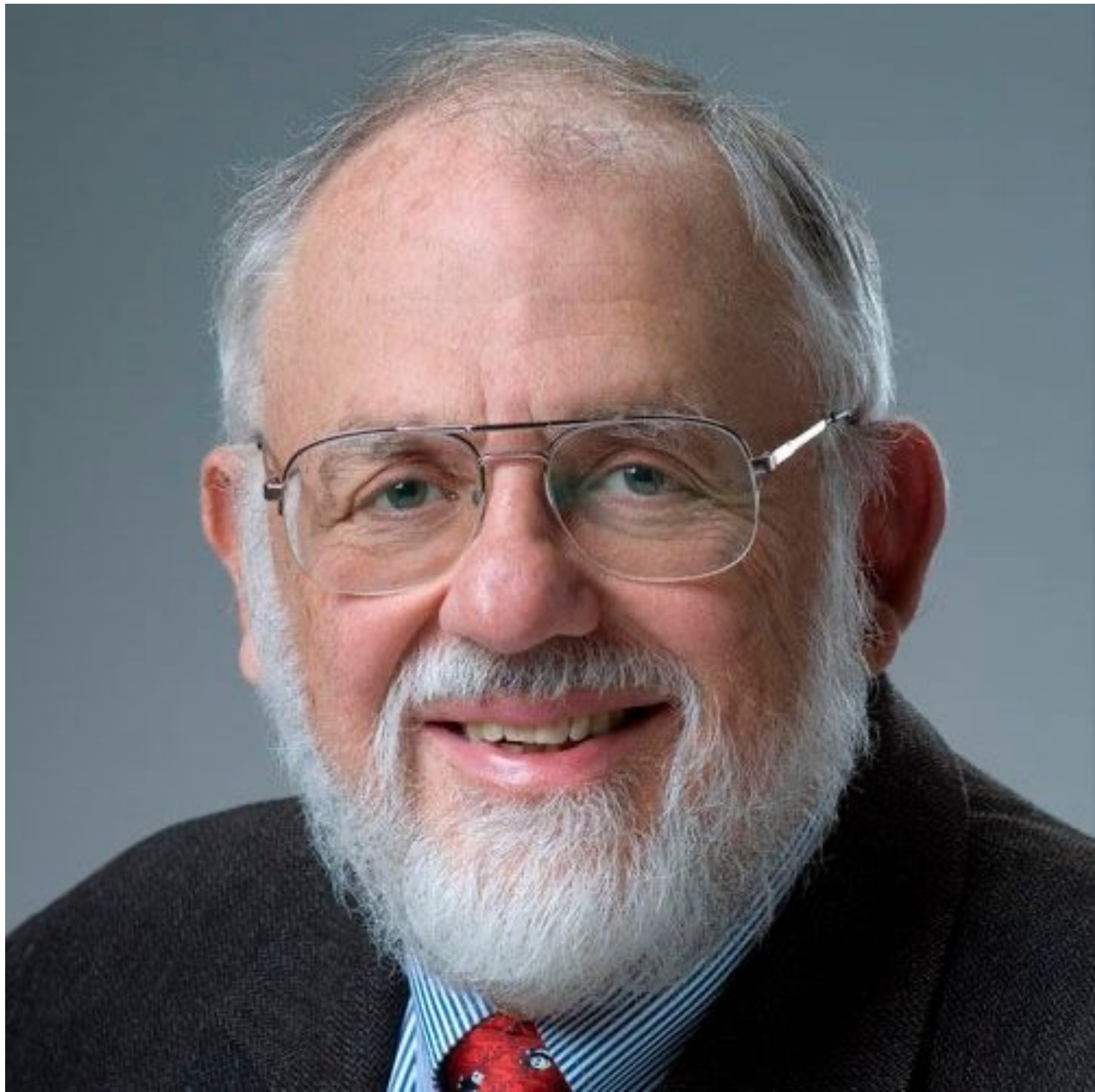
Parallelism – Embarrassingly Parallel

- Embarrassingly parallel (Cleve Moler) are problems that are really really easy to speed up with mode CPUs.
- The most common example is that you have a program that runs in serial and takes some input file, processes it, and produces some output.
- The problem is that you have 1,000 of the input files and want to run your program on each one.



Parallelism – Embarrassingly Parallel

This is ”embarrassing” because all you have to do is run 1,000 copies of your program on 1,000 CPUs each with a different input file and you are done.



```
[vanilla@hopper intro_workshop]$ sbatch slurm/calc_pi_serial.sh
 sbatch: Using account 2016199 from ~/.default_slurm_account
Submitted batch job 5263
vanilla@hopper:~/workshops/intro_workshop$ squeue -me
JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
5263      debug calc_pi_  vanilla  R       0:44      1 hopper011
```

Submit the array script.

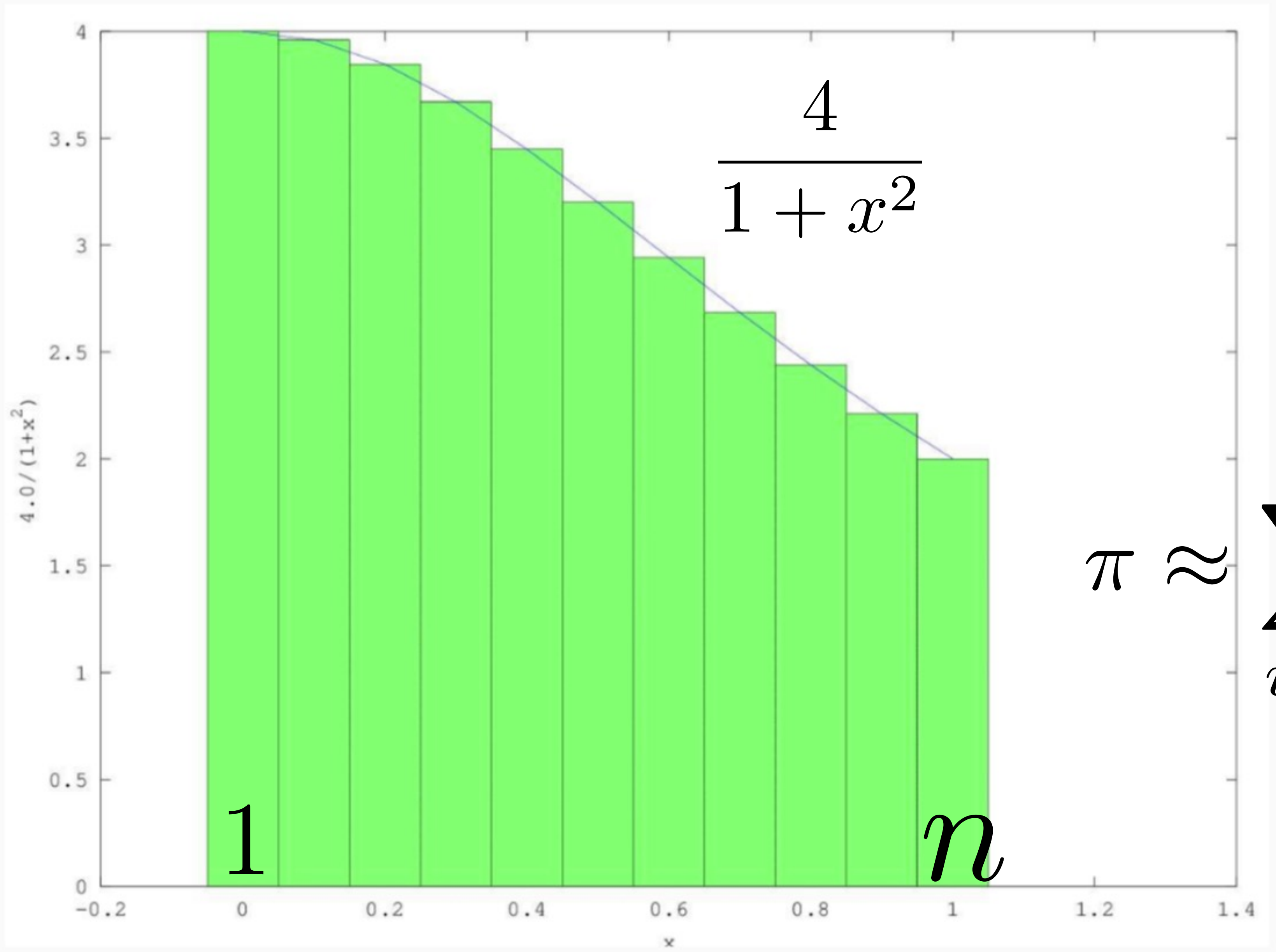
Then enter squeue --me to see the job status.

Take a look at the job output. (How many output files do you have?)

Parallelism – Coupled Parallelism

- Coupled problems are those where the CPUs need to work together to solve a problem by communicating with each other.
- Many commercial and research programs designed to run on HPC systems like CARC use a library called the message passing interface (MPI) to do this.
- We have written an MPI version of our python pi calculator to demonstrate.

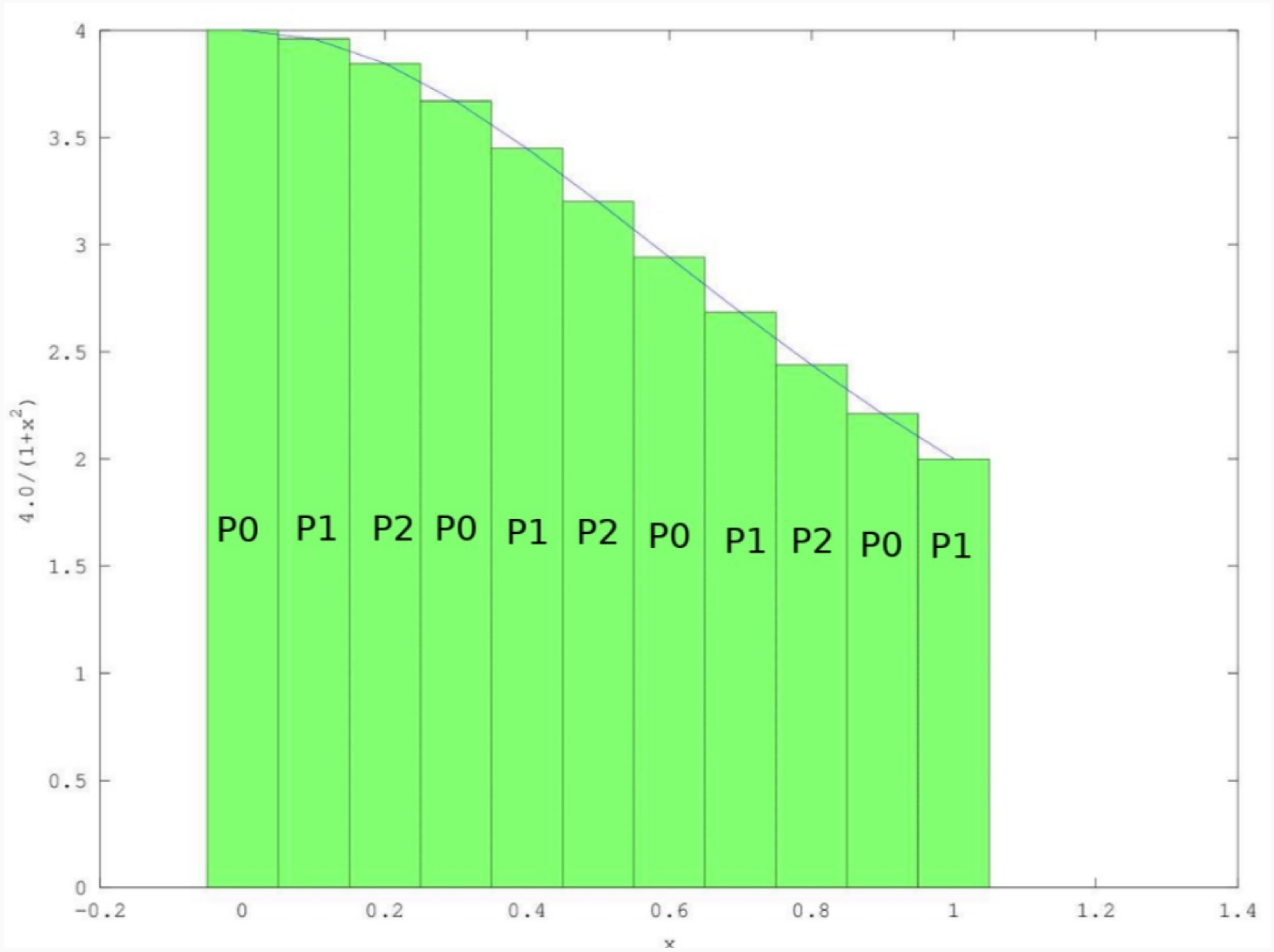
Serial Program to Calculate π



$$w = \frac{1}{n}$$

$$\pi \approx \sum_{i=0}^{n} \frac{4}{1 + (i + \frac{w}{2})^2}$$

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 minconda3
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
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))
```

```
####################### 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
        # Divide sum among processes
        for i in range(rank, num_steps, num_procs):
                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)
```

```
#Main function
<snip>
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
#SBATCH --partition debug
#SBATCH --nodes 2
#SBATCH --ntasks-per-node 4
#SBATCH --time 00:05:00
#SBATCH --job-name calc_pi_mpi
#SBATCH --mail-user your_username@unm.edu
#SBATCH --mail-type ALL

module load miniconda3
source activate mpi_numpy

cd $SLURM_SUBMIT_DIR
srun --mpi=pmi2 python code/calcPiMPI.py 1000000000
```

```
sbatch slurm/calc_pi_mpi.sh
```

```
#!/bin/bash
#SBATCH --partition debug
#SBATCH --nodes 2
#SBATCH --ntasks-per-node 4
#SBATCH --time 00:05:00
#SBATCH --job-name calc_pi_mpi
#SBATCH --mail-user your_username@unm.edu
#SBATCH --mail-type ALL

module load miniconda3
source activate mpi_numpy

cd $SLURM_SUBMIT_DIR
srun --mpi=pmi2 python code/calcPiMPI.py 1000000000
```

srun inside a script
Automatically uses
Whatever you
allocated with
#SBATCH

sbatch slurm/calc_pi_mpi.sh

```
srun --mpi=pmi2 python code/calcPiMPI.py 1000000000
```

srun understands MPI programs!

If you ever used mpirun or mpiexec you had to provide a lot of parameters to describe how many compute nodes you had and what their names are, etc.

But srun is part of SLURM so it already knows all that.

The only thing you have to specify is the communication library to use. In our case “pmi2”.

Experiment

Run calc_pi_mpi.sh.

Vary the number of tasks it uses.

Use squeue to monitor the state of your job,

Look at your output files.

What is the relationship between the number of tasks and how cast it calculates pi?

```
[vanilla@hopper intro_workshop]$ srun --pty bash
 srun: Account not specified in script or
~/.default_slurm_account, using latest project
To request GPUs, add --gpus-per-node X or --gpus X, where X is
the desired number of GPUs.
Job 2061894 running on hopper006
[vanilla@hopper034 ~]$ ls
[vanilla@hopper034 ~]$ exit
[vanilla@hopper~]$ module load matlab
[vanilla@hopper~]$ srun --pty matlab
```

You now have an interactive matlab session through a pseudoterminal!

The "--pty" Option

"--pty" creates a pseudoterminal that lets you interact with a program through the keyboard

```
[vanilla@hopper~]$ module load matlab
[vanilla@hopper~]$ srun --pty matlab
srun: job 879337 queued and waiting for resources
srun: job 879337 has been allocated resources
MATLAB is selecting SOFTWARE OPENGL rendering.


                              < M A T L A B (R) >
                    Copyright 1984-2023 The MathWorks, Inc.
               R2023a Update 5 (9.14.0.2337262) 64-bit (glnxa64)
                                 July 24, 2023



To get started, type doc.
For product information, visit www.mathworks.com.

>>
```

You now have an interactive matlab session through a pseudoterminal!

The “--pty” Option

“--pty” creates a pseudoterminal that lets you interact with a program through the keyboard

Useful Slurm Commands

squeue --me --long	shows information about jobs you submitted
squeue --me --start	shows when slurm expects your job to start
scancel jobid	cancels a job
scancel --u $USER	cancels all your jobs
sacct	shows your job history
seff jobid	shows how efficiently the hardware was used