Intermediate Level Introduction to Computing at CARC FORTRAN

Matthew Fricke

Version 0.2

Goals

- 1) SLURM scheduler literacy
- 2) Ability to employ embarrassingly parallel solutions
- 3) Ability to employ coupled parallel solutions

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

Agenda

- HPC and Parallelism
- HPC Schedulers
- SLURM
- Embarrassingly Parallel
  - SLURM Arrays
  - GNU Parallel
- Coupled Parallelism
- MPIwith MPI

We will have one 15 minute break. Opportunity to see the machine room.

Logging into Hopper

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

Use your CARC username and password.

Jacob, Keven, Tannor, and Jose can help you login if you have trouble.

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

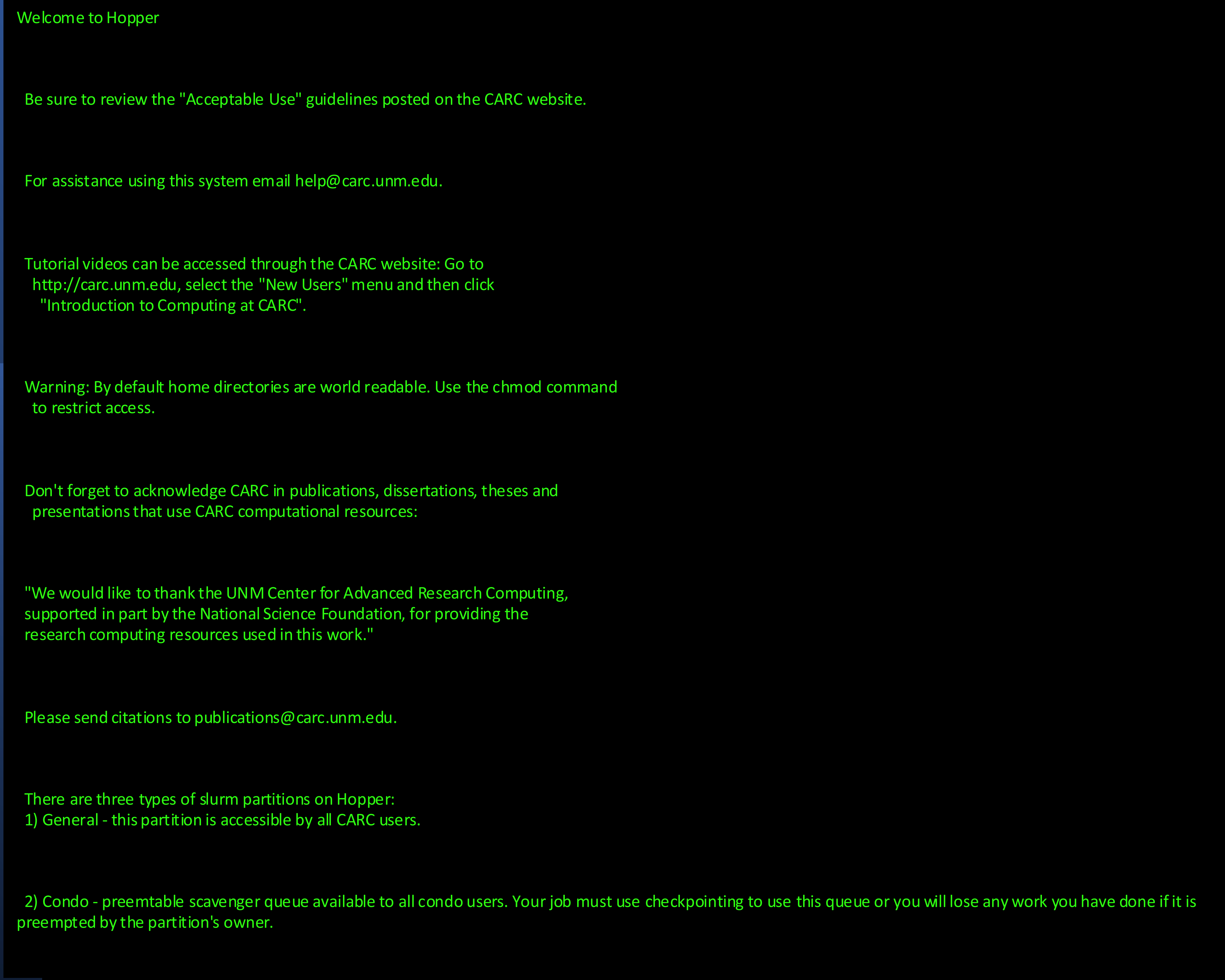
Logging into Hopper

```
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.
```

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   GPUs
-----    -----  -----  -----    -----  -----  -----  -----
general  1      9      0        7      10     320    0
debug    2      0      0        0      2      64     0
totals:  1      9      0        7      10     320    0
```

```
[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 ~]$ man sinfo

sinfo(1)                    Slurm Commands                    sinfo(1)

NAME
      sinfo - View information about Slurm nodes and partitions.

SYNOPSIS
      sinfo [OPTIONS...]

DESCRIPTION
      sinfo is used to view partition and node information for a system running Slurm

OPTIONS
      -a, --all
            Display  information  about  all partitions. This causes information to be displayed about partitions that are
configured as hidden and partitions that are unavailable to the user's group.
```

```
[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)
                                                              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)
                                                               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_account, using latest
project
hopper012
hopper012
You have not been all                                        r
submission script.
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_account, using latest
project
hopper012
hopper012
You have not been all                                          r
submission script.
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 been all                                                  r
submission script.
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 alloca                                                        r
submission script.
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 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.

```
[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]$
```

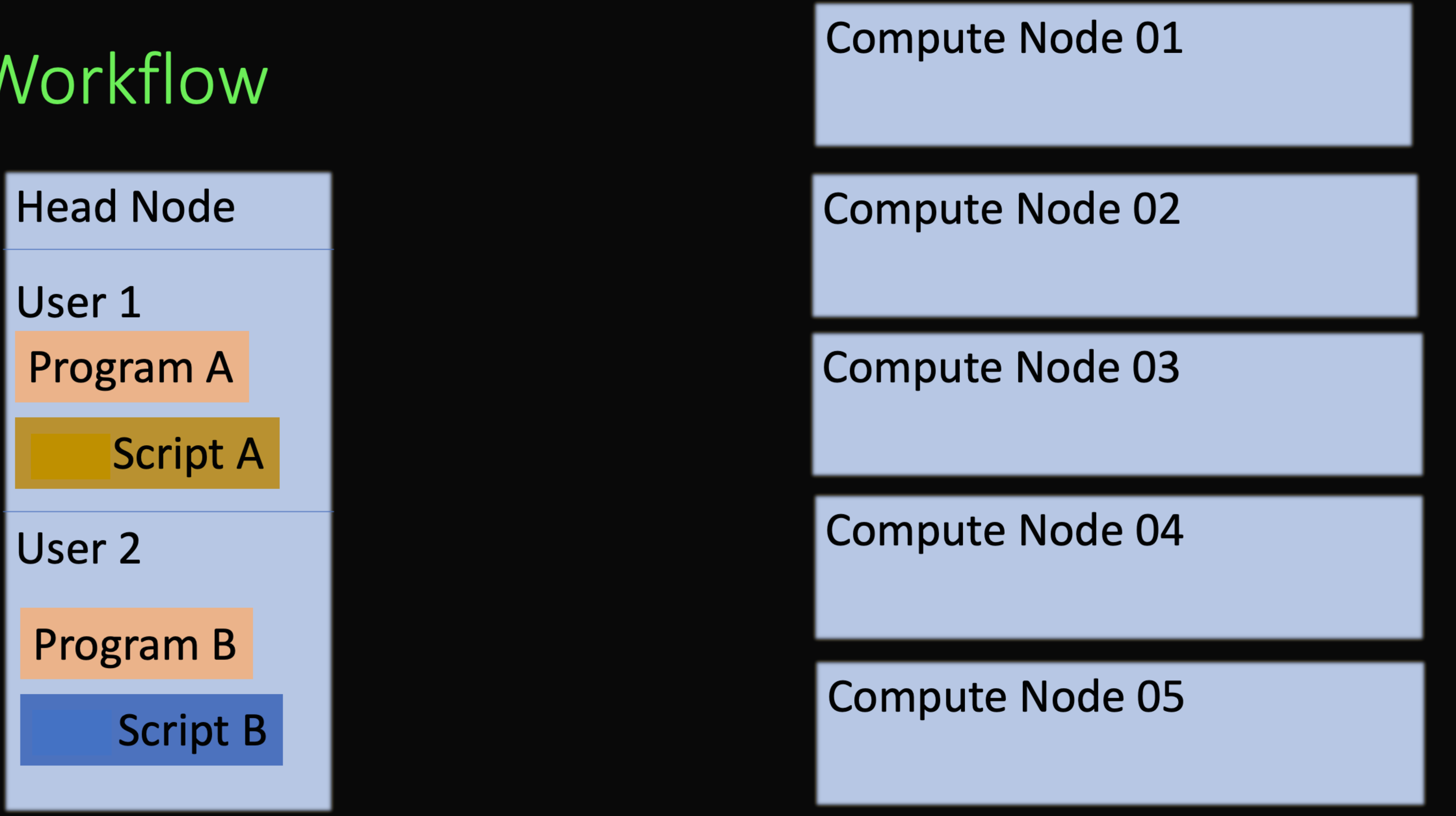
We submit our slurm shell script with the sbatch command.

Notice that the only output we get is a job id.

This indicates that the script was successfully sent to the scheduler.

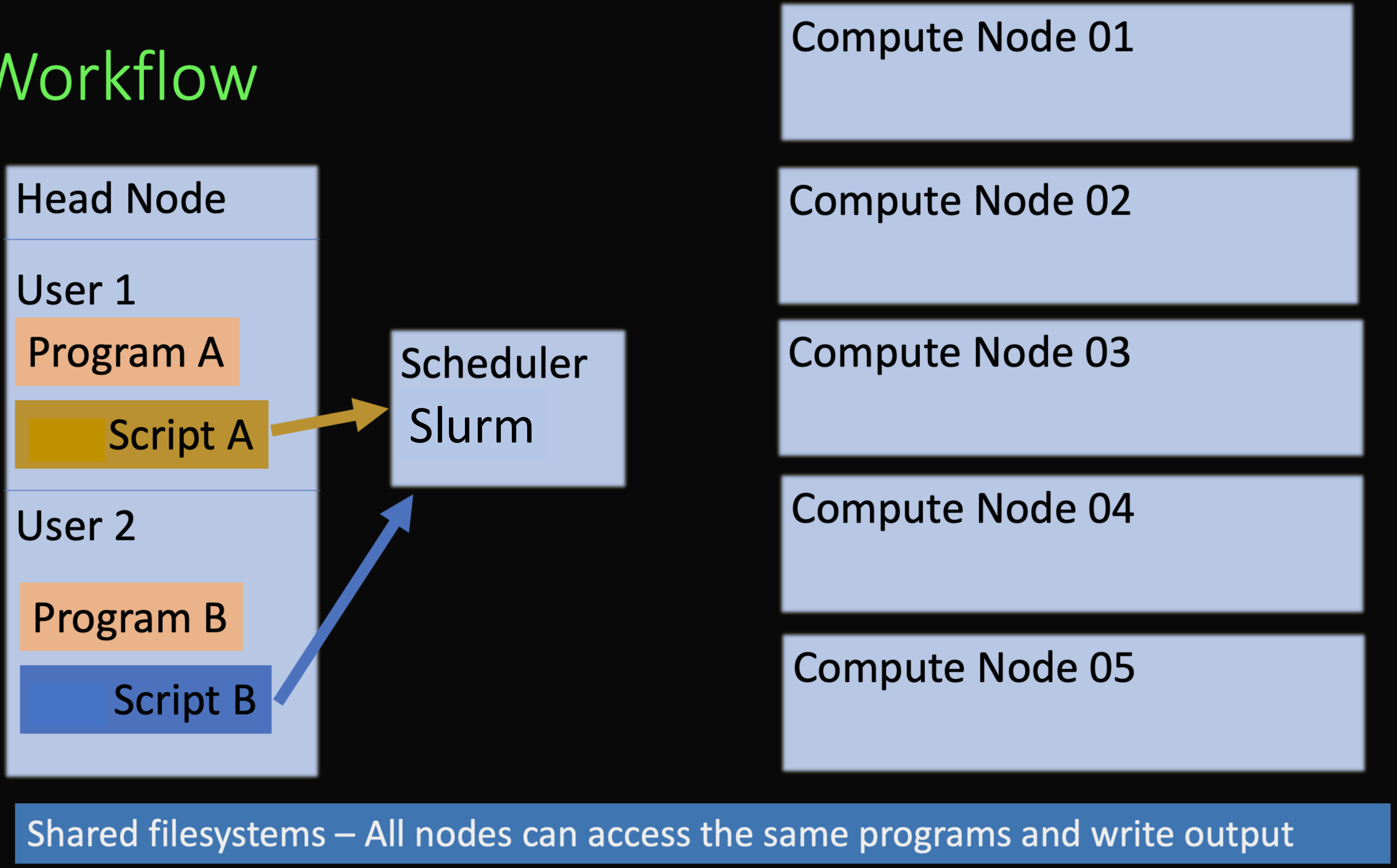
The commands in the script will run as soon as the hardware requested is available.

Workflow

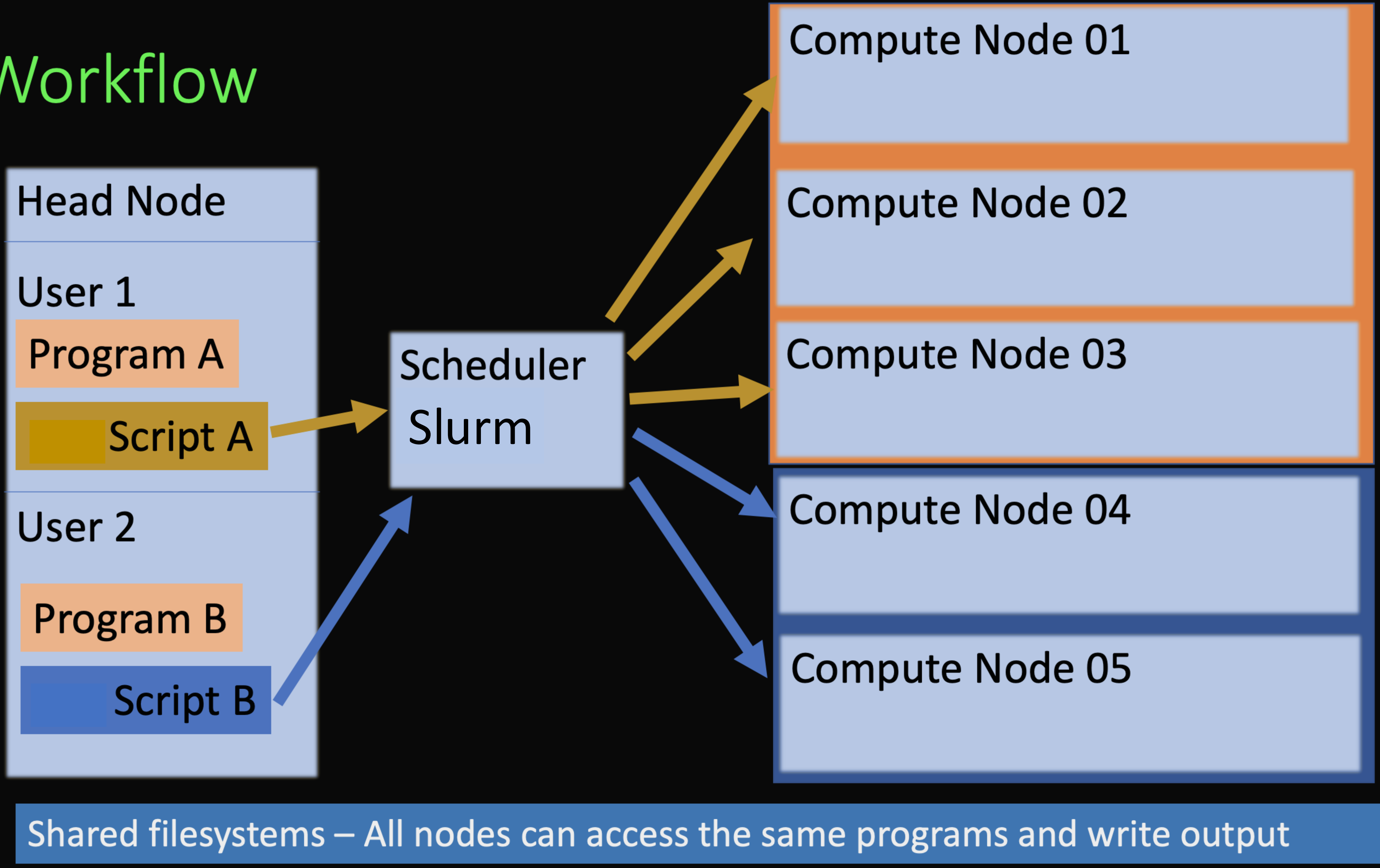


Shared filesystems – All nodes can access the same programs and write output

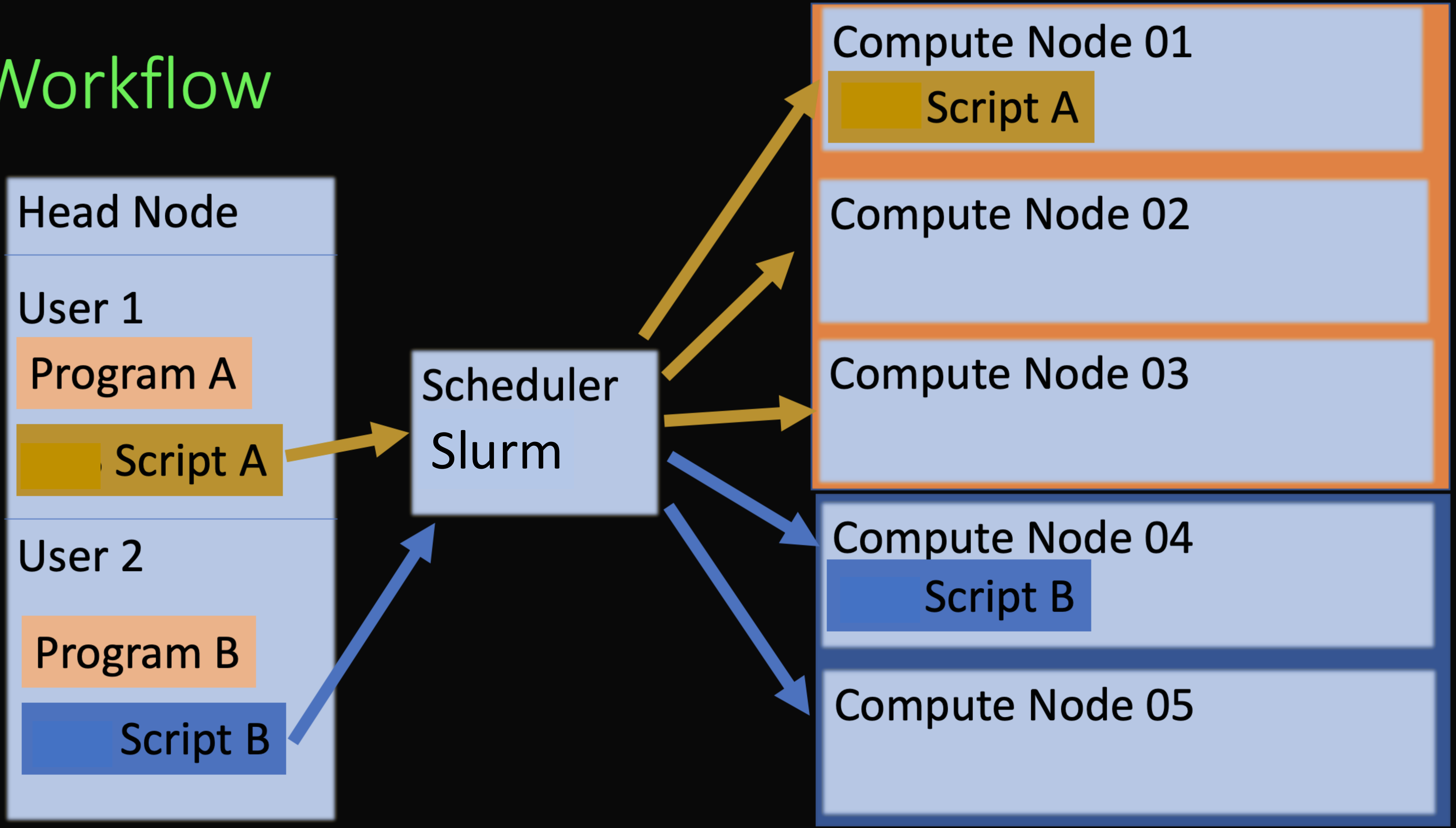
Workflow


Head Node
User 1
Program A
Script A
User 2
Program B
Script B
Scheduler
Slurm
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


Head Node
User 1
Program A
Script A
User 2
Program B
Script B
Scheduler
Slurm
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

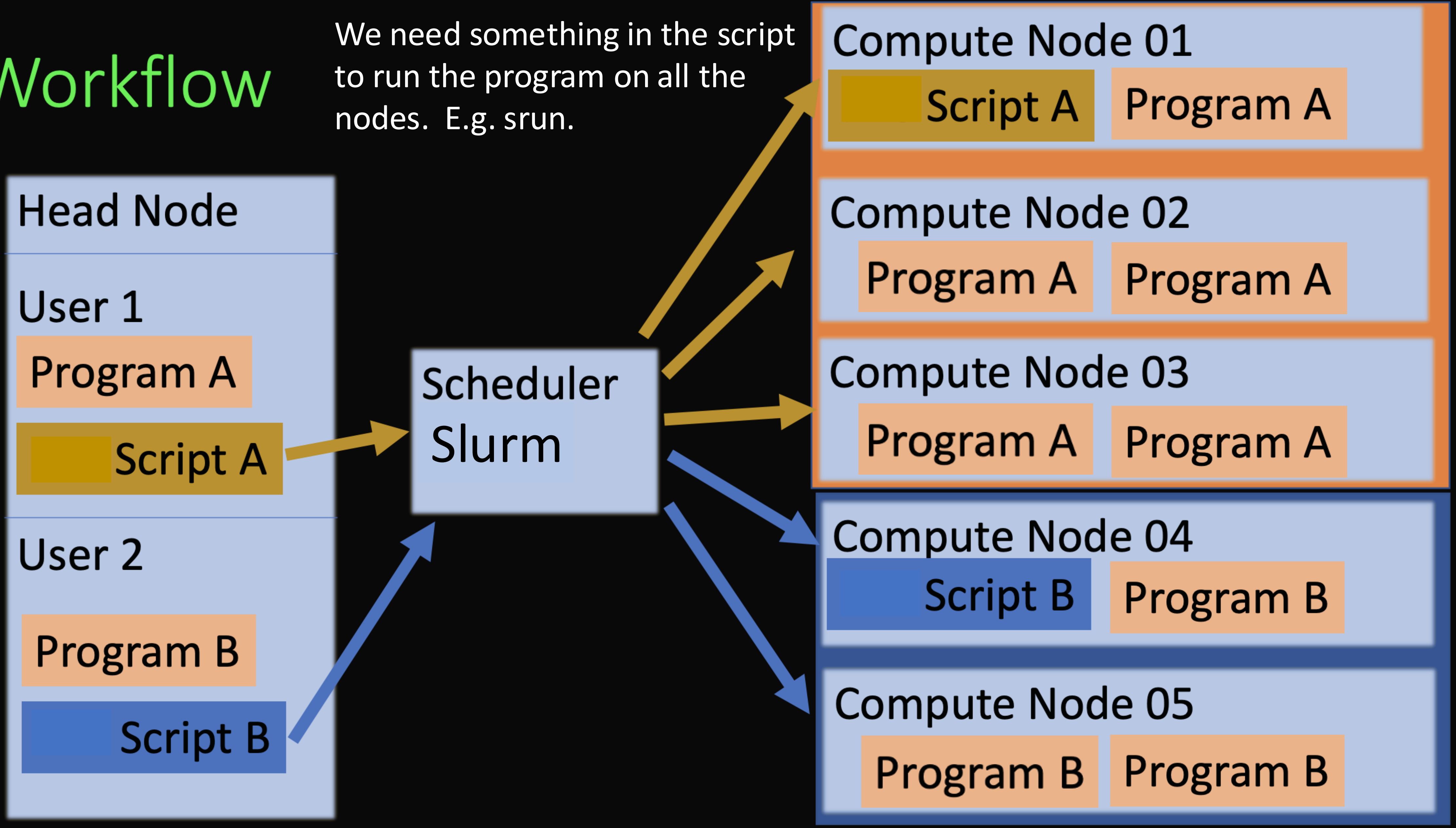
Workflow



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

Workflow

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
```

```
[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.

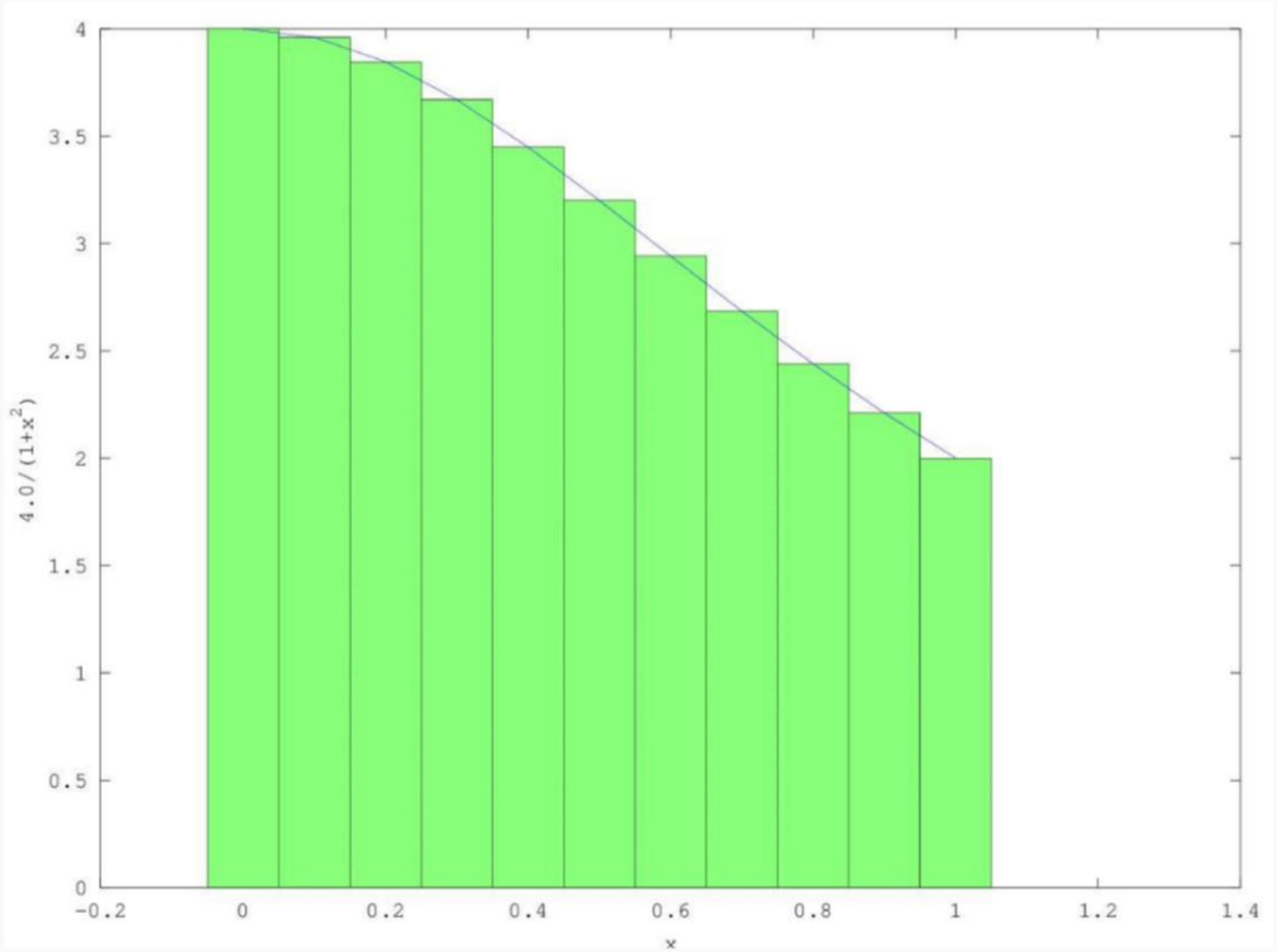
```
[vanilla@hopper intro_workshop]$ cat slurm-5252.out
```

What do you see?

```
[vanilla@hopper intro_workshop]$ cd code
[vanilla@hopper code]$ pwd
~/workshops/intro_workshop/code
```

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

Serial Program to Calculate π



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

$$\Sigma_{i=1}^{N} \frac{4}{1+\left(\frac{i+\frac{1}{2}}{N}\right)^2} \approx \pi$$

*Because arctan…

Program to estimate π $- Main\ program$

```
[vanilla@hopper code]$ cat calc_pi_serial.f90
program cal_pi_serial
  implicit none
  integer steps
  character(len=100) :: steps_arg
  real :: start, finish
  double precision :: p,pi_ref = 3.14159265358979323846264338

  call getarg(1, steps_arg) ! Read argument
  read(steps_arg,*) steps ! Convert string to integer

  call cpu_time(start) ! Time how long it takes to est pi
  p = pi(steps)
  call cpu_time(finish)

  print '("Pi=", g0, " (Diff=", g0,")")', p, abs(p-pi_ref)
  print '("(calculated in ", g0,"s with ", g0, " steps)")',        finish-start, steps

  contains
```

Program to estimate π $- Function\ pi()$

```
! The area under the curve 4/(1+x^2) is pi.
! Estimate with the Riemann sum.

  function pi(num_steps) result(area)
  double precision x, area, step, sum
  integer num_steps
  sum = 0
  step_size = 1.0/num_steps ! Determine the rectangle size

! Loop over the rectangles under the curve
  do i = 0, num_steps
    x = (i + 0.5)*step_size ! Calc the ith rectangle area
    sum = sum + 4.0/(1.0+x*x) ! Sum them up
  end do

  area = step_size*sum ! Normalise to be between 0 and 1

end function


end program
```

Program to estimate π $- Function\ pi()$

```
! The area under the curve 4/(1+x^2) is pi.
! Estimate with the Riemann sum.

  function pi(num_steps) result(area)
  double precision x, area, step, sum
  integer num_steps
  sum = 0
  step_size = 1.0/num_steps ! Determine the rectangle size

! Loop over the rectangles under the curve
  do i = 0, num_steps
    x = (i + 0.5)*step_size ! Calc the ith rectangle area
    sum = sum + 4.0/(1.0+x*x) ! Sum them up
  end do

  area = step_size*sum ! Normalise to be between 0 and 1

end function


end program
```

$$\Sigma_{i=1}^{N} \frac{4}{1+\left(\frac{i+\frac{1}{2}}{N}\right)^{2}} \approx \pi$$

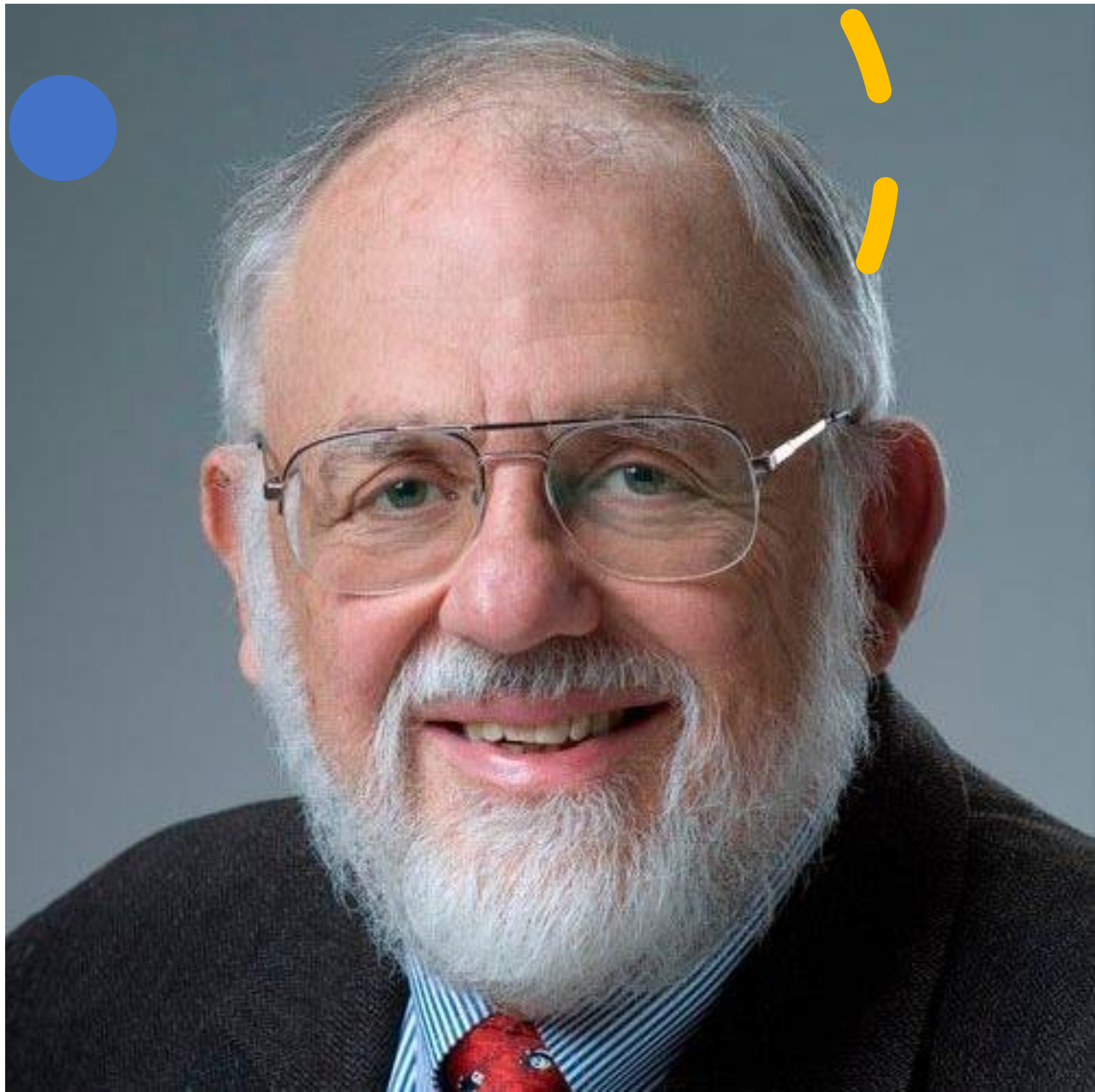
```
[vanilla@hopper code]$ module load intel/20.0.4
[vanilla@hopper code]$ ifort calc_pi_serial.f90 -o calc_pi_serial
[vanilla@hopper code]$ srun --partition debug ./calc_pi_serial 1000
srun: Using account 2016199 from ~/.default_slurm_account
You have not been allocated GPUs. To request GPUs, use the -G option in your submission
script.
Pi=3.143591832167984 (Diff=.1999091155410415E-02)
(calculated in .2999790E-05s with 1000 steps)
```

Load intel compilers then compile our FORTRAN program.

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.

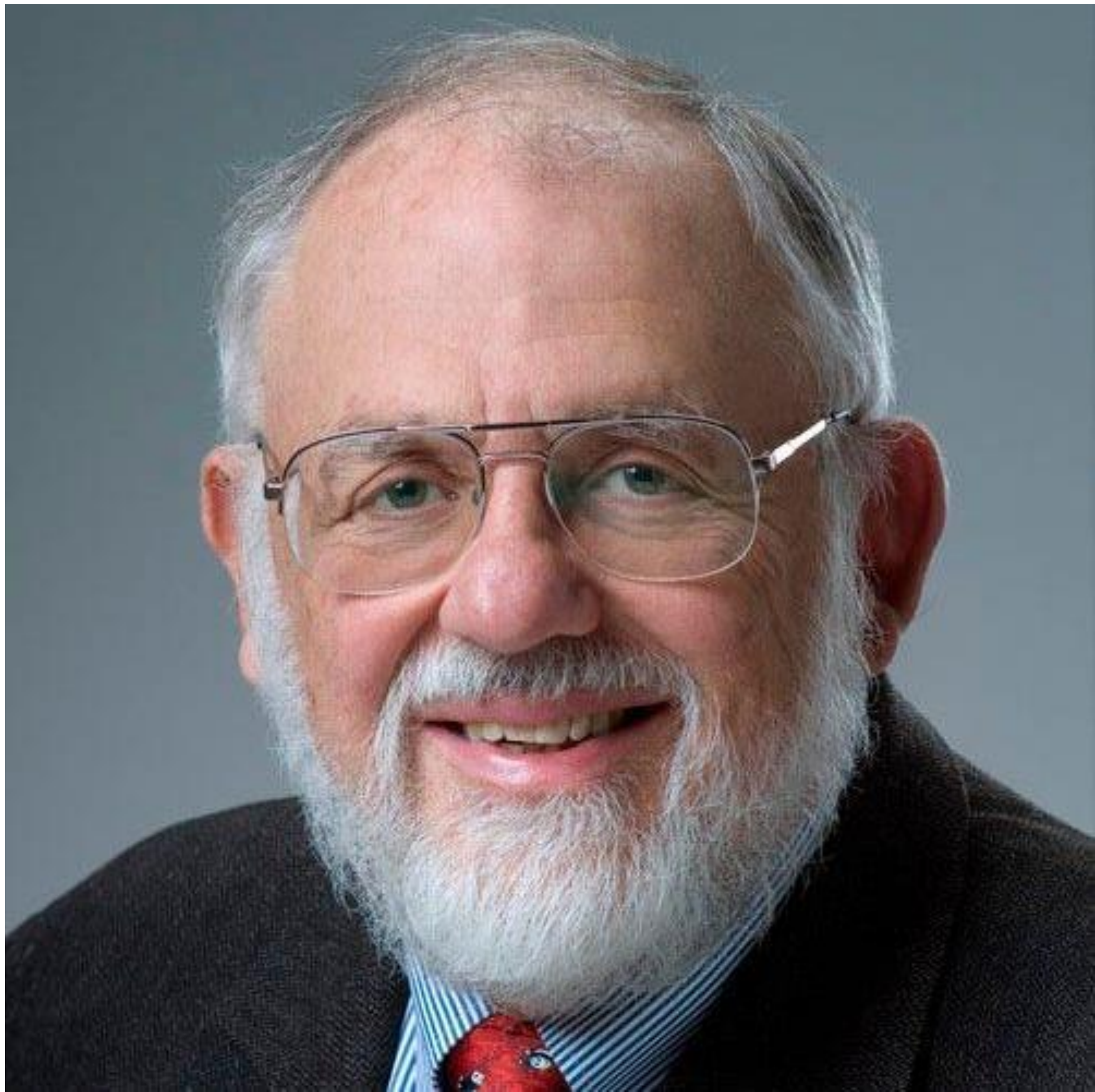
Parallelism – Embarrassingly Parallel

- 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.



SLURM ARRAYS

- One way to run the 1,000 copies of your program on 1,000 different inputs would be to write 1,000 slurm scripts each specifying a different input to your program and then sbatch submit them all. (this would work but there are better ways).

- SLURM arrays are used to schedule a lot of jobs with one slurm script.

```
[vanilla@hopper intro_workshop]$ nano slurm/calc_pi_array_f90.sh
#!/bin/bash
#SBATCH --partition debug
#SBATCH --ntasks 1
#SBATCH --time 00:05:00
#SBATCH --job-name calc_pi_array
#SBATCH --mail-user your_username@unm.edu
#SBATCH --mail-type ALL
#SBATCH --array=1-12%12


echo "$HOSTNAME - $SLURM_ARRAY_TASK_ID"

module load intel/20.0.4 intel-mpi/2019.10.317-ruxn


NUM_STEPS="${SLURM_ARRAY_TASK_ID}0000"
echo "Calculating pi with $NUM_STEPS..."
code/calc_pi_serial $NUM_STEPS
```

Requires some annoying bash scripting.

$something means get the value of the variable "something"

--array says

1) **run 12 separate jobs**
2) **Store the count of the job in the variable SLURM_ARRAY_TASK_ID**

```
[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?)

JOB arrays are OK or very simple inputs like programs that take a single file as input. But even passing in a value takes some annoying variable manipulation.

GNU Parallel is much more sophisticated it can take inputs in all sorts of ways. We will look at just 3 ways.

To access GNU parallel enter **module load parallel**

Let’s experiment with parallel interactively…

***NOTE: we don’t use srun to run parallel.**

GNU Parallel

Has been around for a very long time and has lots and lots of great features.

But basically it creates a job for every input it receives. The inputs can be specified in the command, read from a file, or be the output of another program.

It also remembers which jobs have finished and which still need to be run. So when you run out of time and resubmit it will automatically pick up where it left off.

```
[vanilla@hopper intro_workshop]$ seq 10 10 100
10
20
30
40
50
60
70
80
90
100
```

GNU Parallel can read the output of other programs and use them as inputs to your program.
Here a copy of calc pi is run for each row in the output of seq

```
[vanilla@hopper intro_workshop]$ module load parallel
[vanilla@hopper intro_workshop]$ seq 10 10 100 | parallel code/calc_pi_serial
Pi = 3.14180098689309428295, (Diff=0.00020833330330116695) (calculated in 0.000007 secs with 20 steps)
Pi = 3.14242598500109870940, (Diff=0.00083333141130559341) (calculated in 0.000006 secs with 10 steps)
Pi = 3.14168524617974842528, (Diff=0.00009259258995530928) (calculated in 0.000007 secs with 30 steps)
etc
```

```
[vanilla@hopper intro_workshop]$ find -name *.sh
./slurm/calc_pi_array.sh
./slurm/calc_pi_mpi.sh
./slurm/calc_pi_parallel.sh
./slurm/calc_pi_serial.sh
./slurm/gaussian.sh
./slurm/hostname_mpi.sh
etc

$ find -name *.sh | parallel wc -l
7 ./code/vecadd/vecaddmpi_cpu.sh
19 ./slurm/calc_pi_array.sh
15 ./slurm/calc_pi_mpi.sh
20 ./slurm/calc_pi_parallel.sh
14 ./slurm/calc_pi_serial.sh
16 ./slurm/gaussian.sh
15 ./slurm/hostname_mpi.sh
etc
```

A common application is to use find to produce a list of paths with some extension.

Then parallel runs some program on each path.

In this case wc -l counts the lines in a file. In some real CARC examples the input files are phylogenetic trees, graphs, neuroimages, or CT scans.

```
[vanilla@hopper intro_workshop]$ exit
exit
salloc: Relinquishing job allocation 5275
```

Don’t forget to exit your salloc allocation.

```
$sbatch slurm/calc_pi_parallel_f90.sh
sbatch: Using account 2016199 from ~/.default_slurm_account
Submitted batch job 5276
```

Submit the job, check it’s progress with squeue --me, and take a look at the output.

```
$sbatch slurm/calc_pi_parallel_f90.sh
sbatch: Using account 2016199 from ~/.default_slurm_account
Submitted batch job 5276
```

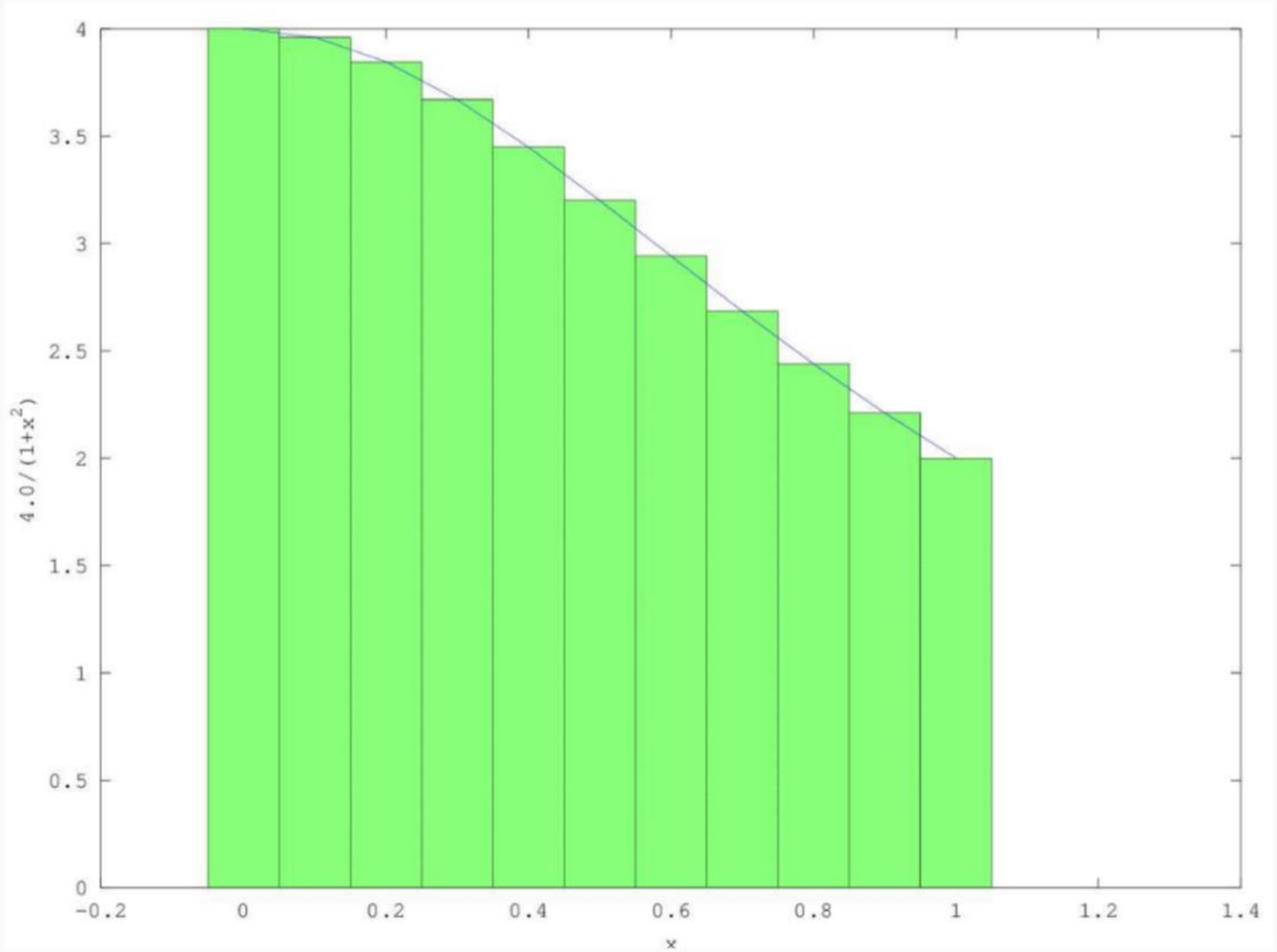
Submit the job, check it’s progress with squeue --me, and take a look at the output.

Parallelism – Coupled Parallelism

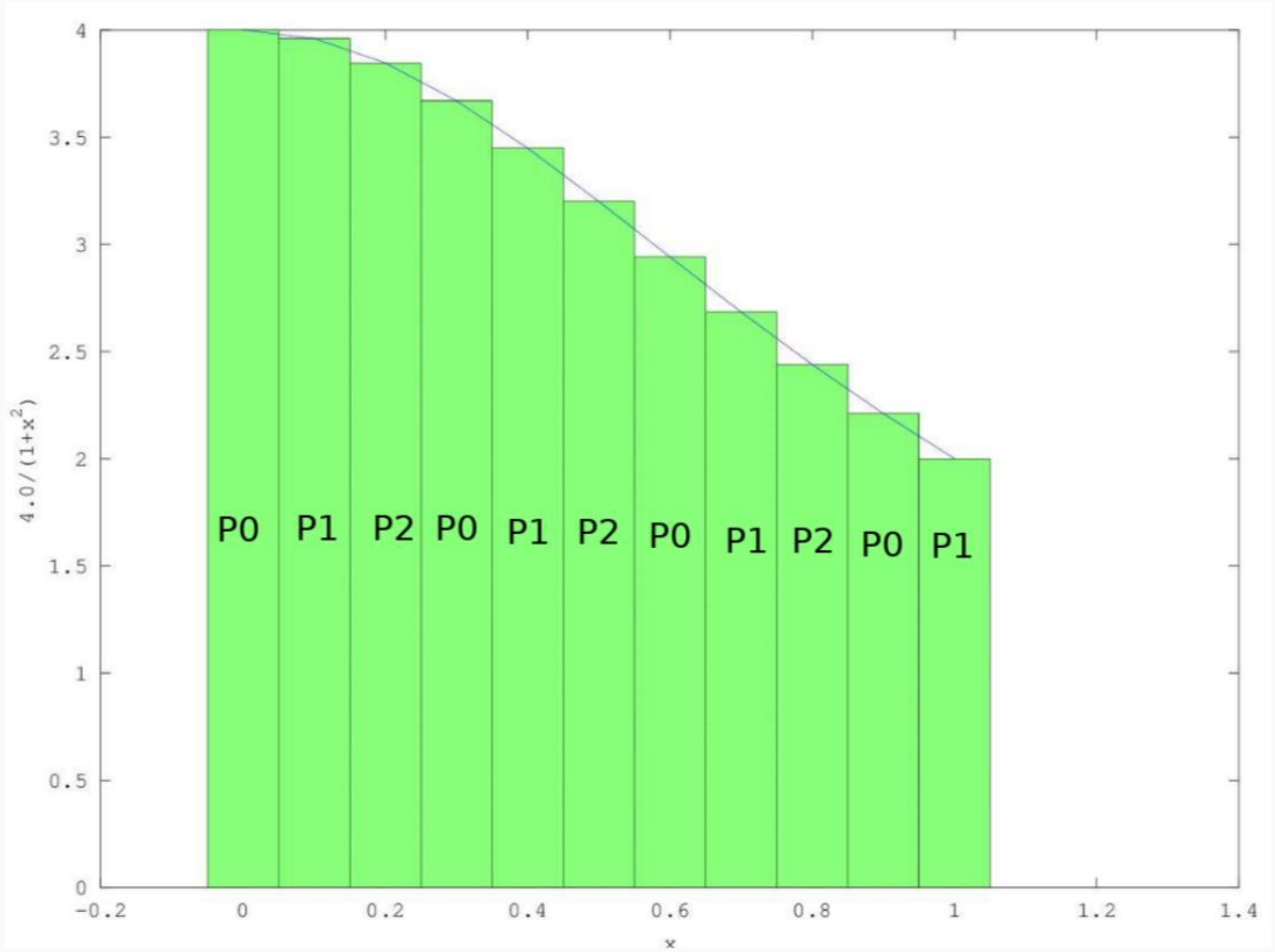
- Coupled problems are those where the CPUs need to work together to solve a problem by communicating with each other.
- Many commercial and research programs designed to run on HPC systems like CARC use a library called the message passing interface (MPI) to do this.

- We have written an MPI version of our python pi calculator to demonstrate.

Serial Program to Calculate π



Parallel Program to Calculate π



MPI Program to estimate π

```
program cal_pi_mpi
  use mpi ! Import Message Passing Interface
  implicit none ! Don't allow FORTRAN to infer types
  integer steps, ierr, num_procs, root, rank
  character(len=100) steps_arg
  real :: start, finish
  double precision :: p,pi_ref = 3.141592653589793238462643383279502884197


  ! Setup Message Passing Interface
  call MPI_Init( ierr )
  call MPI_Comm_rank( MPI_COMM_WORLD, rank, ierr )
  call MPI_Comm_size( MPI_COMM_WORLD, num_procs, ierr )
  root = 0 ! Call the 0th process "root"

  call getarg(1, steps_arg) ! Read argument
  read(steps_arg,*) steps ! Convert string to integer

  call cpu_time(start) ! Time how long it takes to est pi
  ! Call the pi function and print the result
  p = pi(steps, num_procs, rank)
  call cpu_time(finish)

  if (rank==root) then
    print '("Pi=", g0, " (Diff=", g0,")")', p, abs(p-pi_ref)
    print '("(calculated in ", g0,"s with ", g0, " steps and ", g0, " processes)")', finish-start, steps, num_procs
  endif
```

```
contains
  ! The area under the curve 4/(1+x^2) is pi
  ! Estimate with the Riemann sum
  function pi(num_steps, num_procs, rank) result(area)
    double precision x, area, my_area, step, sum, step_size
    integer num_steps, num_procs, rank, i
  sum = 0
  step_size = 1.0/num_steps ! Normalise the area between 0 and 1
  do i = rank, num_steps, num_procs ! Only loop over our part of the sum
     x = (i + 0.5)*step_size
     sum = sum + 4.0/(1.0+x*x)
  end do
  my_area = step_size*sum


! Get that partial sums from all the processes, add them up, and give to the root  syntax: input variable, output
variable, data size, MPI Type, Operation, output rank, MPI World, error int
  call mpi_reduce(my_area,area,1,MPI_DOUBLE,MPI_SUM,root,MPI_COMM_WORLD,ierr)

end function
end program
```

```
[vanilla@hopper code]$ module load intel-mpi/2019.10.317-ruxn
[vanilla@hopper code]$ mpiifort calc_pi_mpi.f90 -o calc_pi_mpi
```

Compile the code with Intel MPI FORTRAN compiler

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
```

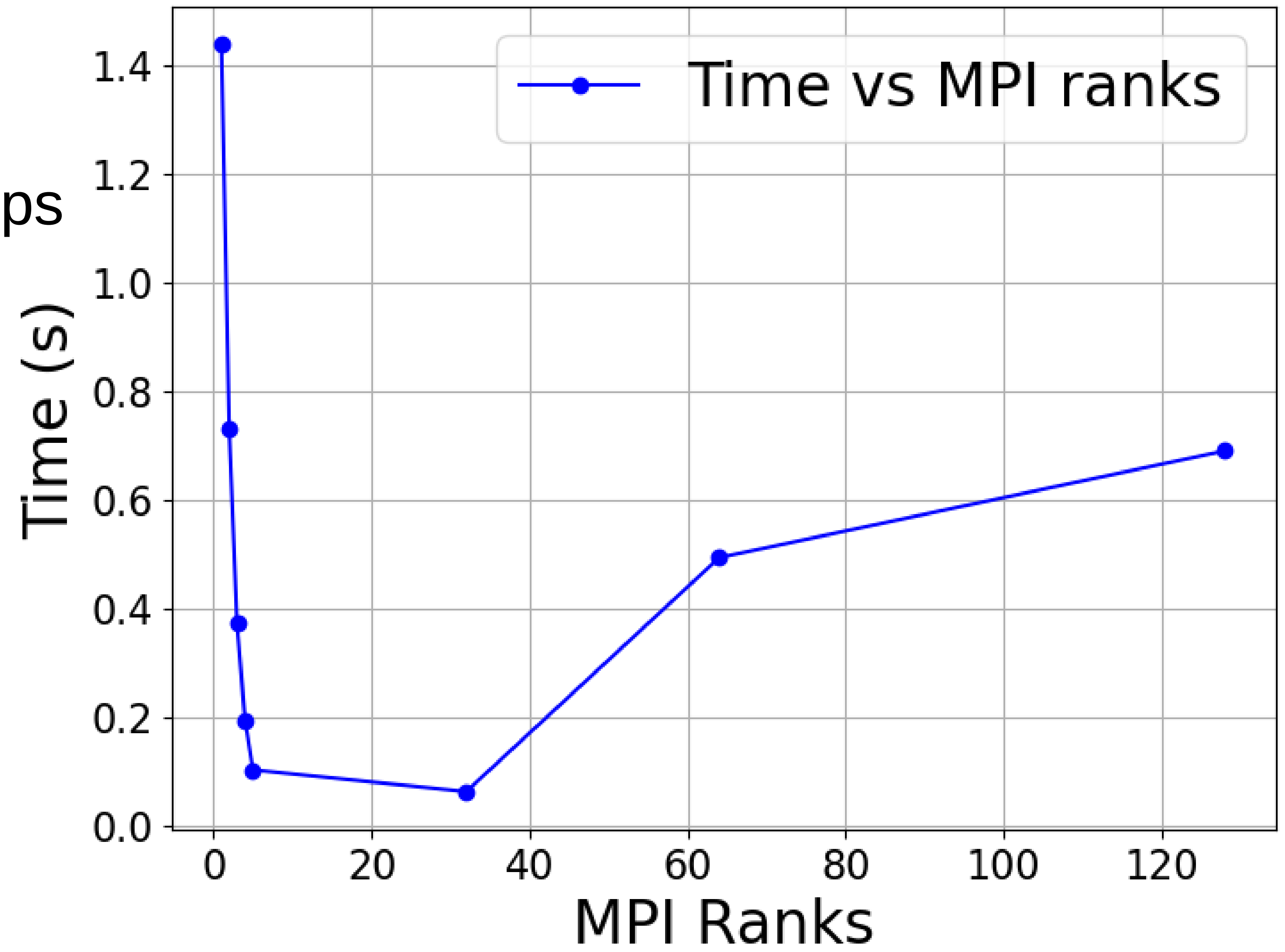
```
#!/bin/bash

#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 intel/20.0.4 intel-mpi/2019.10.317-ruxn

srun --mpi=pmi2  code/calc_pi_mpi 1000000000
```

```
sbatch slurm/calc_pi_mpi_f90.sh
```

```
[vanilla@hopper intro_workshop]$ module load parallel
```

```
[vanilla@hopper code]$ seq 1 1 6 | parallel srun --partition debug --ntasks {} calc_pi_mpi 100000
Job 3056979 running on hopper011
Pi=3.141612602973879 (Diff=.19861961305878 14E-04)
(calculated in .2641998E-02s with 100000 steps and 2 processes)
Job 3056977 running on hopper011
Pi=3.141612602973873 (Diff=.1986196130010498E-04)
(calculated in .2910048E-03s with 100000 steps and 1 processes)
Job 3056976 running on hopper011
Pi=3.141612602973876 (Diff=.1986196130232543E-04)
(calculated in .4189983E-03s with 100000 steps and 3 processes)
Job 3056978 running on hopper011
Pi=3.141612602973876 (Diff=.1986196130321360E-04)
(calculated in .2626002E-02s with 100000 steps and 6 processes)
Job 3056980 running on hopper011
Pi=3.141612602973874 (Diff=.1986196130054907E-04)
(calculated in .3805004E-02s with 100000 steps and 4 processes)
Job 3056981 running on hopper011
Pi=3.141612602973876 (Diff=.1986196130232543E-04)
(calculated in .2314001E-02s with 100000 steps and 5 processes)
```

Speedup with MPI Rank for 2×10^{10} steps



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