URCF Workshop Nov. 2021: Benchmarking & the Command Line

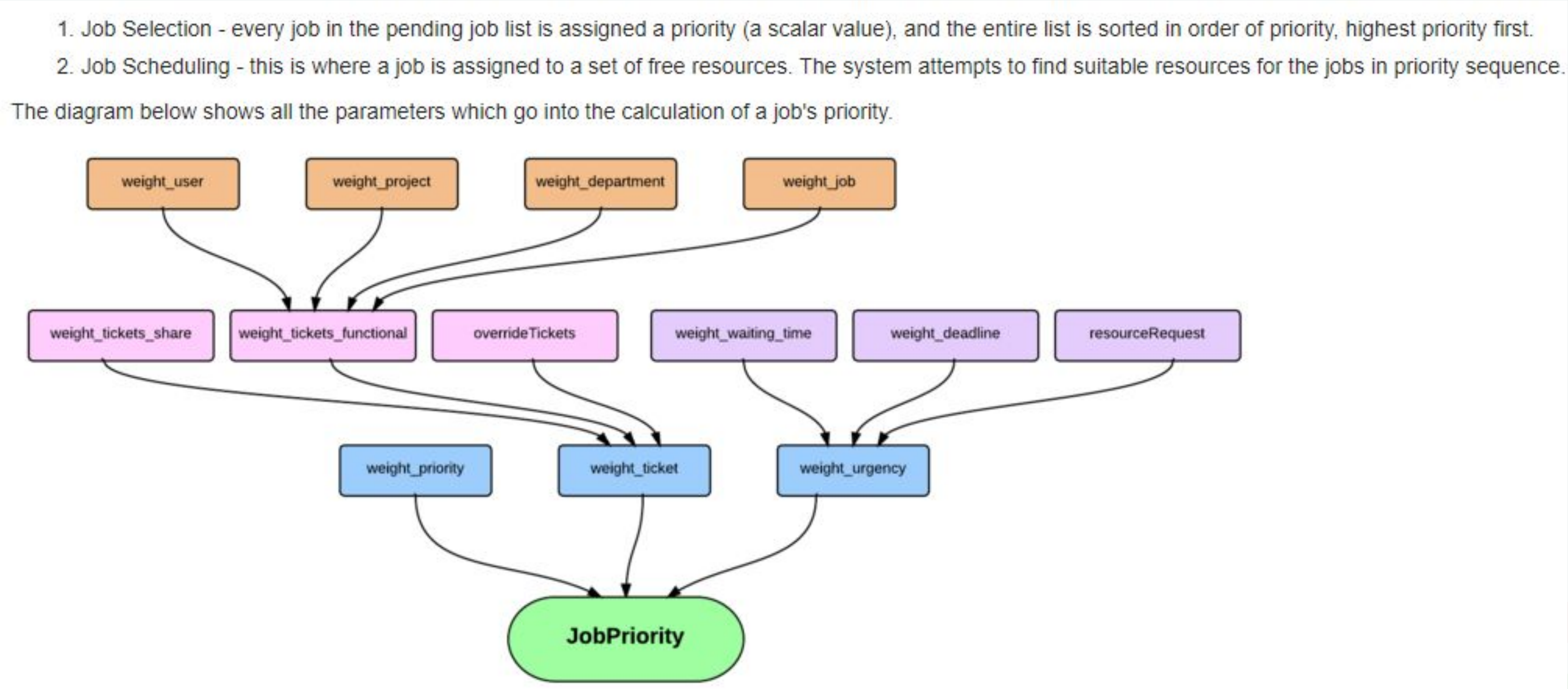
Thomas G. Coard & Bahrad A. Sokhansanj

November 18, 2021

tgc37@drexel.edu

bas44@drexel.edu

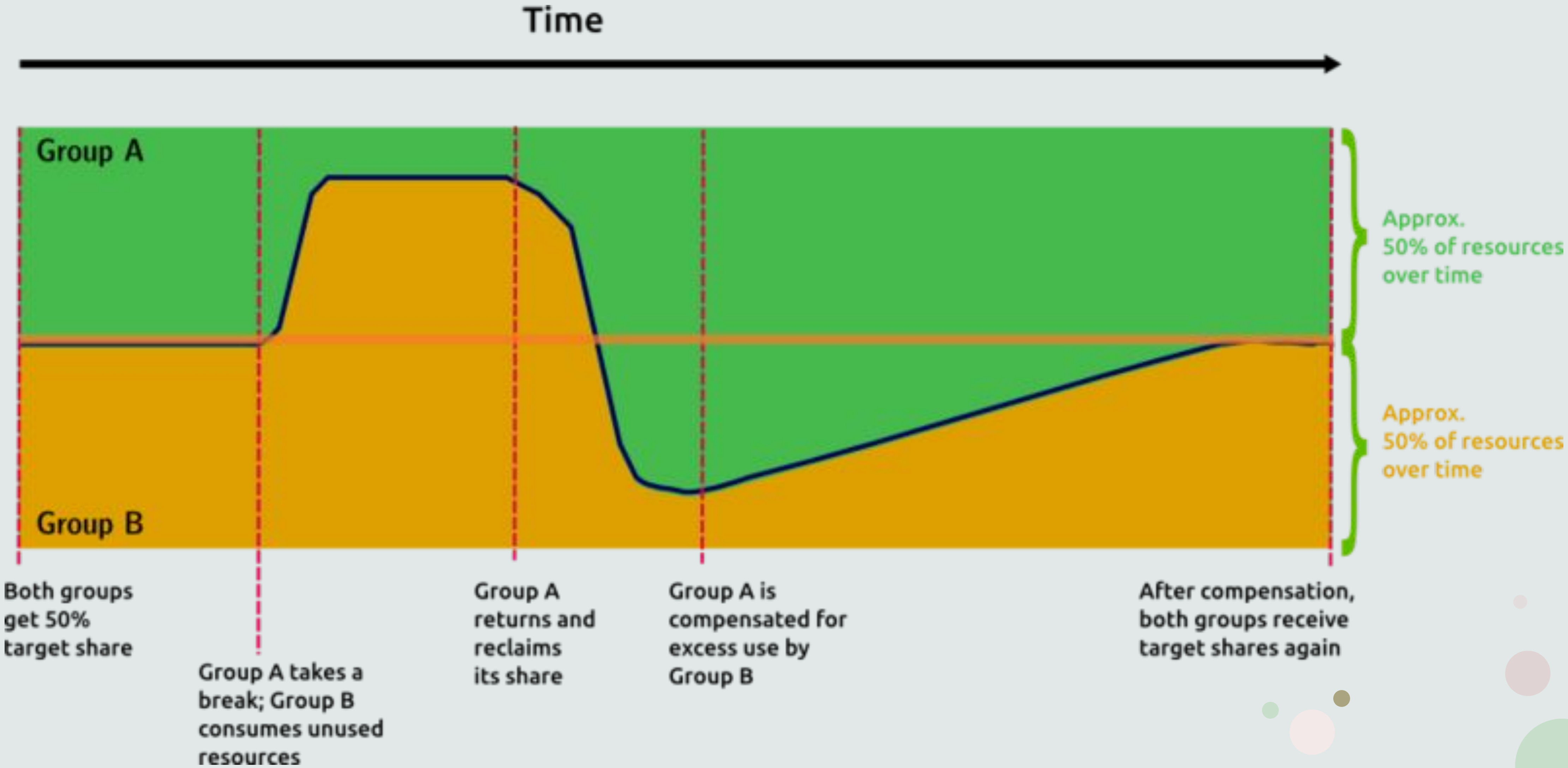
Job Scheduling in Picotte



1. Job Selection - every job in the pending job list is assigned a priority (a scalar value), and the entire list is sorted in order of priority, highest priority first.
2. Job Scheduling - this is where a job is assigned to a set of free resources. The system attempts to find suitable resources for the jobs in priority sequence.

The diagram below shows all the parameters which go into the calculation of a job's priority.

Job Scheduling in Picotte



Approx. 50% of resources over time

Approx. 50% of resources over time

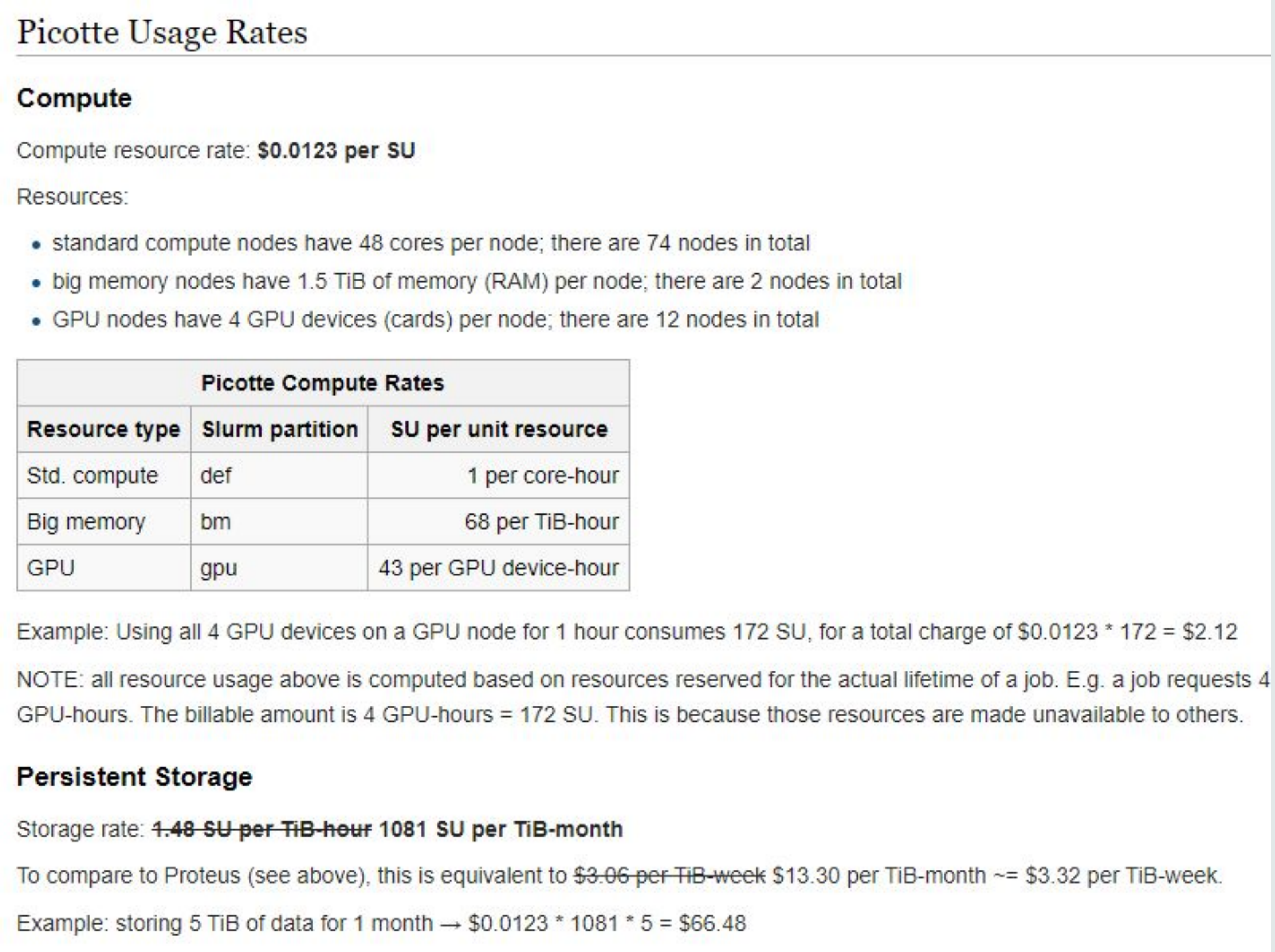
Both groups get 50% target share

Group A takes a break; Group B consumes unused resources

Group A returns and reclaims its share

Group A is compensated for excess use by Group B

After compensation, both groups receive target shares again



Picotte Usage Rates

Compute

Compute resource rate: **$0.0123 per SU**

Resources:

- standard compute nodes have 48 cores per node; there are 74 nodes in total
- big memory nodes have 1.5 TiB of memory (RAM) per node; there are 2 nodes in total
- GPU nodes have 4 GPU devices (cards) per node; there are 12 nodes in total

Picotte Compute Rates		
Resource type	**Slurm partition**	**SU per unit resource**
Std. compute	def	1 per core-hour
Big memory	bm	68 per TiB-hour
GPU	gpu	43 per GPU device-hour

Example: Using all 4 GPU devices on a GPU node for 1 hour consumes 172 SU, for a total charge of $0.0123 * 172 = $2.12

NOTE: all resource usage above is computed based on resources reserved for the actual lifetime of a job. E.g. a job requests 4 GPU-hours. The billable amount is 4 GPU-hours = 172 SU. This is because those resources are made unavailable to others.

Persistent Storage

Storage rate: ~~**1.48 SU per TiB-hour**~~ **1081 SU per TiB-month**

To compare to Proteus (see above), this is equivalent to ~~$3.06 per TiB-week~~ $13.30 per TiB-month ~= $3.32 per TiB-week.

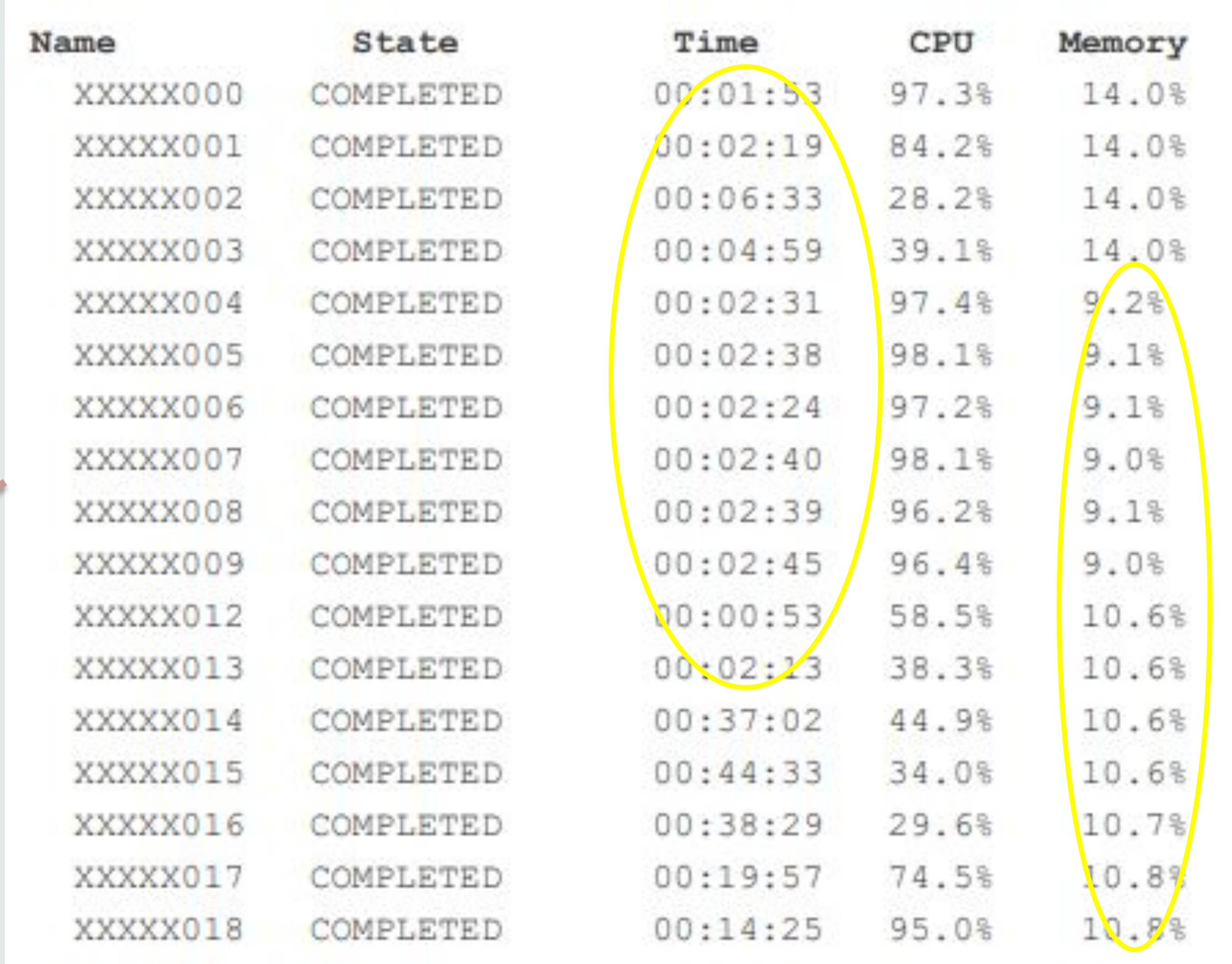
Example: storing 5 TiB of data for 1 month → $0.0123 * 1081 * 5 = $66.48

Understanding your jobs

- Memory?
  - MaxRSS (Resident Set Size) is what matters (maximum memory usage by process)
  - Virtual Memory (VM) doesn’t contribute to job limit
- Storage Space?
- Run Time?
  - **Don’t** run large volumes of short jobs (overhead on scheduler)
  - Do run a few iterations of an interative job (e.g., machine learning, run a few epochs) to estimate run-time
    - Use verbose outputs, e.g., verbose = True for sklearn, verbose = 1 or 2 for Tensorflow
- Sub-processes?

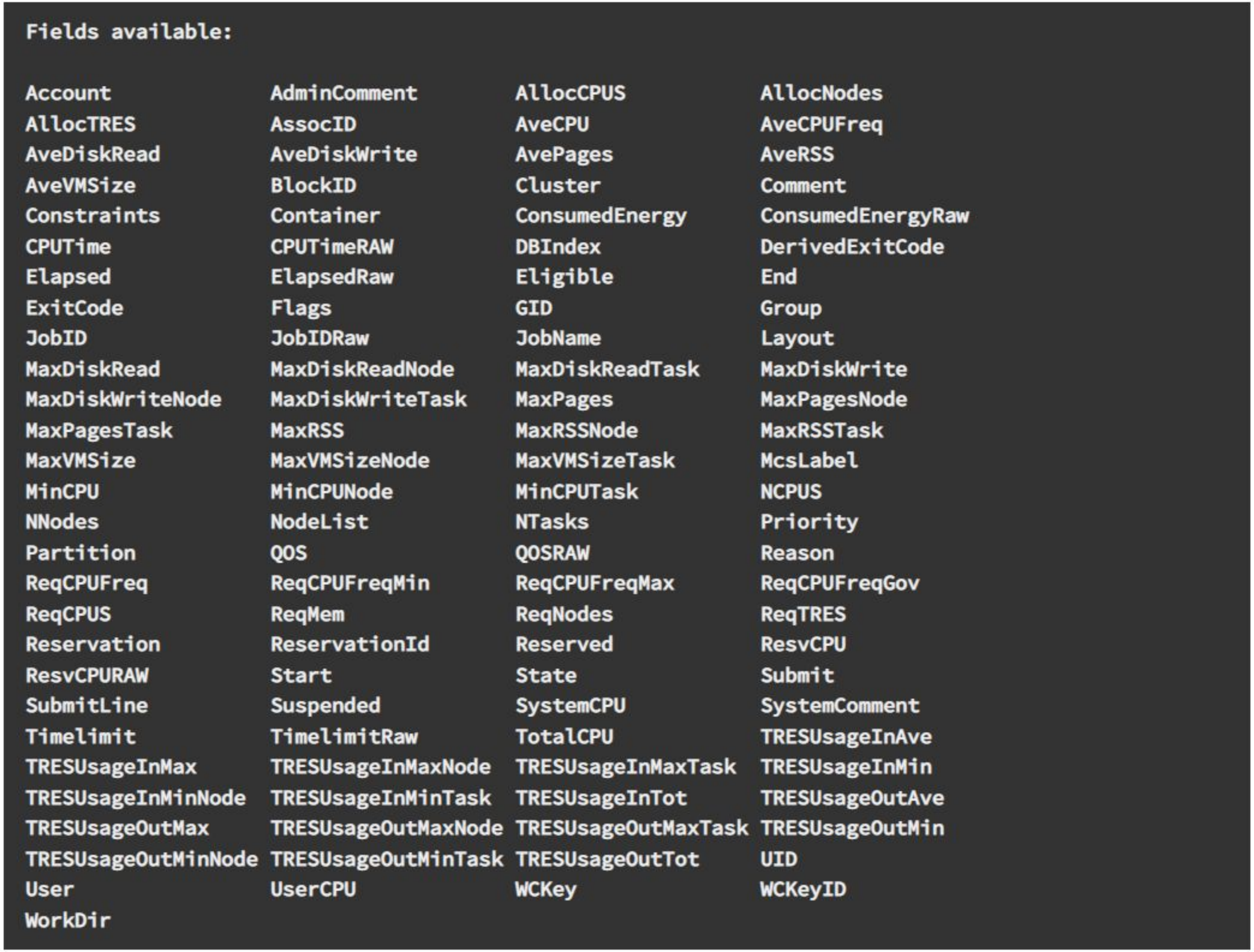
- **sacct** - displays accounting data for all jobs and job steps in the Slurm job accounting log or Slurm database
  - https://slurm.schedmd.com/sacct.html
  - Also, **sstat** (running jobs), **seff**, **sreport** (all information at the man pages above)

Job Scheduling & Memory Requests



```
Name              State              Time       CPU      Memory
   XXXXX000     COMPLETED          00:01:53    97.3%     14.0%
   XXXXX001     COMPLETED          00:02:19    84.2%     14.0%
   XXXXX002     COMPLETED          00:06:33    28.2%     14.0%
   XXXXX003     COMPLETED          00:04:59    39.1%     14.0%
   XXXXX004     COMPLETED          00:02:31    97.4%      9.2%
   XXXXX005     COMPLETED          00:02:38    98.1%      9.1%
   XXXXX006     COMPLETED          00:02:24    97.2%      9.1%
   XXXXX007     COMPLETED          00:02:40    98.1%      9.0%
   XXXXX008     COMPLETED          00:02:39    96.2%      9.1%
   XXXXX009     COMPLETED          00:02:45    96.4%      9.0%
   XXXXX012     COMPLETED          00:00:53    58.5%     10.6%
   XXXXX013     COMPLETED          00:02:13    38.3%     10.6%
   XXXXX014     COMPLETED          00:37:02    44.9%     10.6%
   XXXXX015     COMPLETED          00:44:33    34.0%     10.6%
   XXXXX016     COMPLETED          00:38:29    29.6%     10.7%
   XXXXX017     COMPLETED          00:19:57    74.5%     10.8%
   XXXXX018     COMPLETED          00:14:25    95.0%     10.8%
```

Not using memory allocation = inefficient, will use up space on nodes and leave less resources available

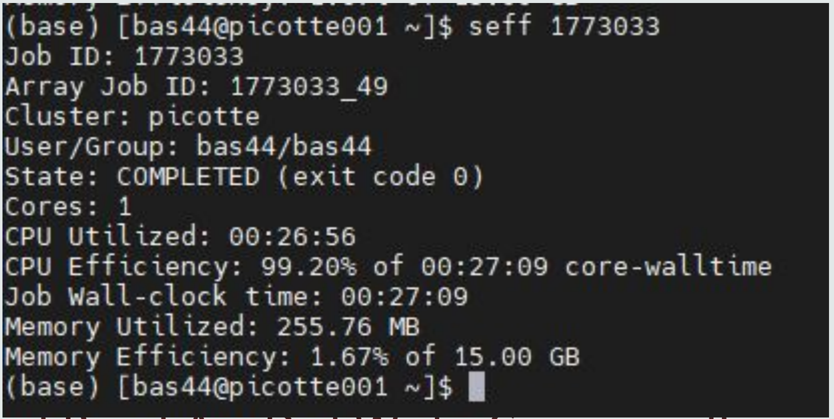


```
Fields available:

Account             AdminComment        AllocCPUS           AllocNodes
AllocTRES           AssocID             AveCPU              AveCPUFreq
AveDiskRead         AveDiskWrite        AvePages            AveRSS
AveVMSize           BlockID             Cluster             Comment
Constraints         Container           ConsumedEnergy      ConsumedEnergyRaw
CPUTime             CPUTimeRAW          DBIndex             DerivedExitCode
Elapsed             ElapsedRaw          Eligible            End
ExitCode            Flags               GID                 Group
JobID               JobIDRaw            JobName             Layout
MaxDiskRead         MaxDiskReadNode     MaxDiskReadTask     MaxDiskWrite
MaxDiskWriteNode    MaxDiskWriteTask    MaxPages            MaxPagesNode
MaxPagesTask        MaxRSS              MaxRSSNode          MaxRSSTask
MaxVMSize           MaxVMSizeNode       MaxVMSizeTask       McsLabel
MinCPU              MinCPUNode          MinCPUTask          NCPUS
NNodes              NodeList            NTasks              Priority
Partition           QOS                 QOSRAW              Reason
ReqCPUFreq          ReqCPUFreqMin       ReqCPUFreqMax       ReqCPUFreqGov
ReqCPUS             ReqMem              ReqNodes            ReqTRES
Reservation         ReservationId       Reserved            ResvCPU
ResvCPURAW          Start               State               Submit
SubmitLine          Suspended           SystemCPU           SystemComment
Timelimit           TimelimitRaw        TotalCPU            TRESUsageInAve
TRESUsageInMax      TRESUsageInMaxNode  TRESUsageInMaxTask  TRESUsageInMin
TRESUsageInMinNode  TRESUsageInMinTask  TRESUsageInTot      TRESUsageOutAve
TRESUsageOutMax     TRESUsageOutMaxNode TRESUsageOutMaxTask TRESUsageOutMin
TRESUsageOutMinNode TRESUsageOutMinTask TRESUsageOutTot     UID
User                UserCPU             WCKey               WCKeyID
WorkDir
```

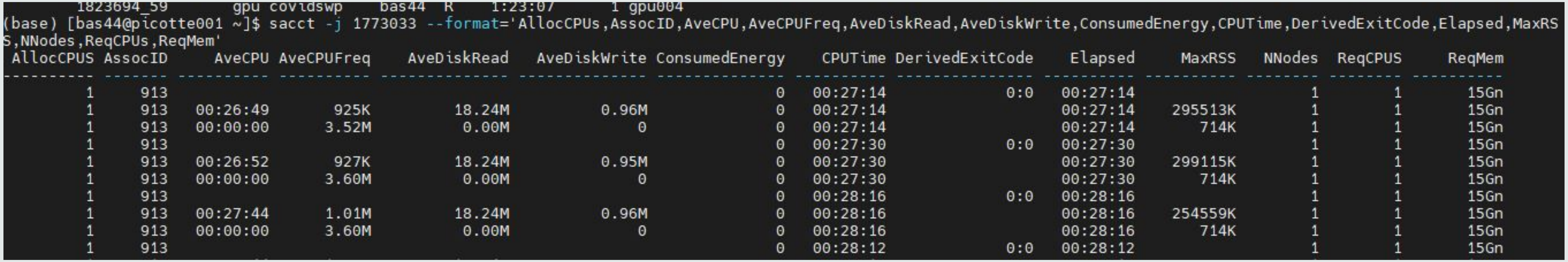
Example

- **seff** (note: may be inaccurate) for $JOBID = 1773033



```
(base) [bas44@picotte001 ~]$ seff 1773033
Job ID: 1773033
Array Job ID: 1773033_49
Cluster: picotte
User/Group: bas44/bas44
State: COMPLETED (exit code 0)
Cores: 1
CPU Utilized: 00:26:56
CPU Efficiency: 99.20% of 00:27:09 core-walltime
Job Wall-clock time: 00:27:09
Memory Utilized: 255.76 MB
Memory Efficiency: 1.67% of 15.00 GB
(base) [bas44@picotte001 ~]$
```

- Compare to sacct (optionally dump to text file)
- **sacct** -j 1773033 --format='AllocCPUs,AssocID,AveCPU,AveCPUFreq,AveDiskRead,AveDiskWrite,ConsumedEnergy,CPUTime,DerivedExitCode,Elapsed,MaxRSS,NNodes,ReqCPUs,ReqMem' > sacct.txt



```
          1823694_59       gpu covidswp    bas44  R    1:23:07       1 gpu004
(base) [bas44@picotte001 ~]$ sacct -j 1773033 --format='AllocCPUs,AssocID,AveCPU,AveCPUFreq,AveDiskRead,AveDiskWrite,ConsumedEnergy,CPUTime,DerivedExitCode,Elapsed,MaxRS
S,NNodes,ReqCPUs,ReqMem'
 AllocCPUS AssocID     AveCPU AveCPUFreq    AveDiskRead   AveDiskWrite ConsumedEnergy    CPUTime DerivedExitCode    Elapsed     MaxRSS   NNodes  ReqCPUS     ReqMem
---------- ------- ---------- ---------- -------------- -------------- -------------- ---------- --------------- ---------- ---------- -------- -------- ----------
         1     913                                                                 0   00:27:14             0:0   00:27:14                   1        1       15Gn
         1     913   00:26:49       925K         18.24M          0.96M              0   00:27:14                   00:27:14    295513K        1        1       15Gn
         1     913   00:00:00      3.52M          0.00M              0              0   00:27:14                   00:27:14       714K        1        1       15Gn
         1     913                                                                 0   00:27:30             0:0   00:27:30                   1        1       15Gn
         1     913   00:26:52       927K         18.24M          0.95M              0   00:27:30                   00:27:30    299115K        1        1       15Gn
         1     913   00:00:00      3.60M          0.00M              0              0   00:27:30                   00:27:30       714K        1        1       15Gn
         1     913                                                                 0   00:28:16             0:0   00:28:16                   1        1       15Gn
         1     913   00:27:44      1.01M         18.24M          0.96M              0   00:28:16                   00:28:16    254559K        1        1       15Gn
         1     913   00:00:00      3.60M          0.00M              0              0   00:28:16                   00:28:16       714K        1        1       15Gn
         1     913                                                                 0   00:28:12             0:0   00:28:12                   1        1       15Gn
```

What is the Command Line?

It is an interactive text environment for running commands within the shell.

The shell is a language that provides an interface between the user and the programs of the operating system.

There are many different shell languages, but they often use the same/similar syntax for common tasks.

Shell examples: bash, sh, dash, zsh, fish, PowerShell, etc.

Useful Bash Syntax: History Expansion

Examples:

- !! (to run previous command)
- !:1 (to get all but the first parameter of the last command)
- !3 (to run the third item in history)
- !cd (to re-run the last cd command, or the last command that started with anything after “!”)

And there is much more that can be done with history expansion.

Helpful Commands for Finding Data

- grep
- find
- cut
- sort
- uniq
- wc

To get more information about these commands run “man” followed by the command name.

You can pass data between commands with “|”

Examples

```
(base) [tgc37@picotte001 workshop]$ sort abc.txt
aaa aaa
aaa aaa
aaa aaa
aaa zzz
aba abc
bbb abc
bbb bbb
bbb zzz
cbc cbc
cbc cbc
ccc ddd
(base) [tgc37@picotte001 workshop]$ head -n 2 !:1
head -n 2 abc.txt
bbb bbb
aaa aaa
(base) [tgc37@picotte001 workshop]$ grep zzz abc.txt
aaa zzz
bbb zz
```

```
(base) [tgc37@picotte001 workshop]$ sort -k 2 abc.txt # sort
by the second column
aaa aaa
aaa aaa
aaa aaa
aba abc
bbb abc
bbb bbb
cbc cbc
cbc cbc
ccc ddd
aaa zzz
bbb zzz
(base) [tgc37@picotte001 workshop]$ cut -d" " -f 1 abc.txt |
sort | tail -n 2 # get first column, then sort it, then get the last
two rows
cbc
ccc
```

Other Topics to Explore

xargs, sed, awk, process and file substitution, profile files.

Support & Office Hours

- David Chin, Ph.D., System Administrator: urcf-support@drexel.edu

- Zoom link for Thomas Coard's Office Hours (Mon. 12-1 pm / Wed. 1-2 pm EST):
  https://drexel.zoom.us/j/87266595816?pwd=bW11eXJGVUlPZm96azRaL0U2RHNKQT09
  Meeting ID: 872 6659 5816
  Passcode: 662662

- Zoom link for Bahrad Sokhansanj's Office Hours (Tues. 1-2 pm / Thurs. 4-5 pm EST):
  https://drexel.zoom.us/j/86773001944?pwd=SWw4ZFp3MXFTbWtOVmZucXVBZWFUdz09
  Meeting ID: 867 7300 1944
  Passcode: 600246