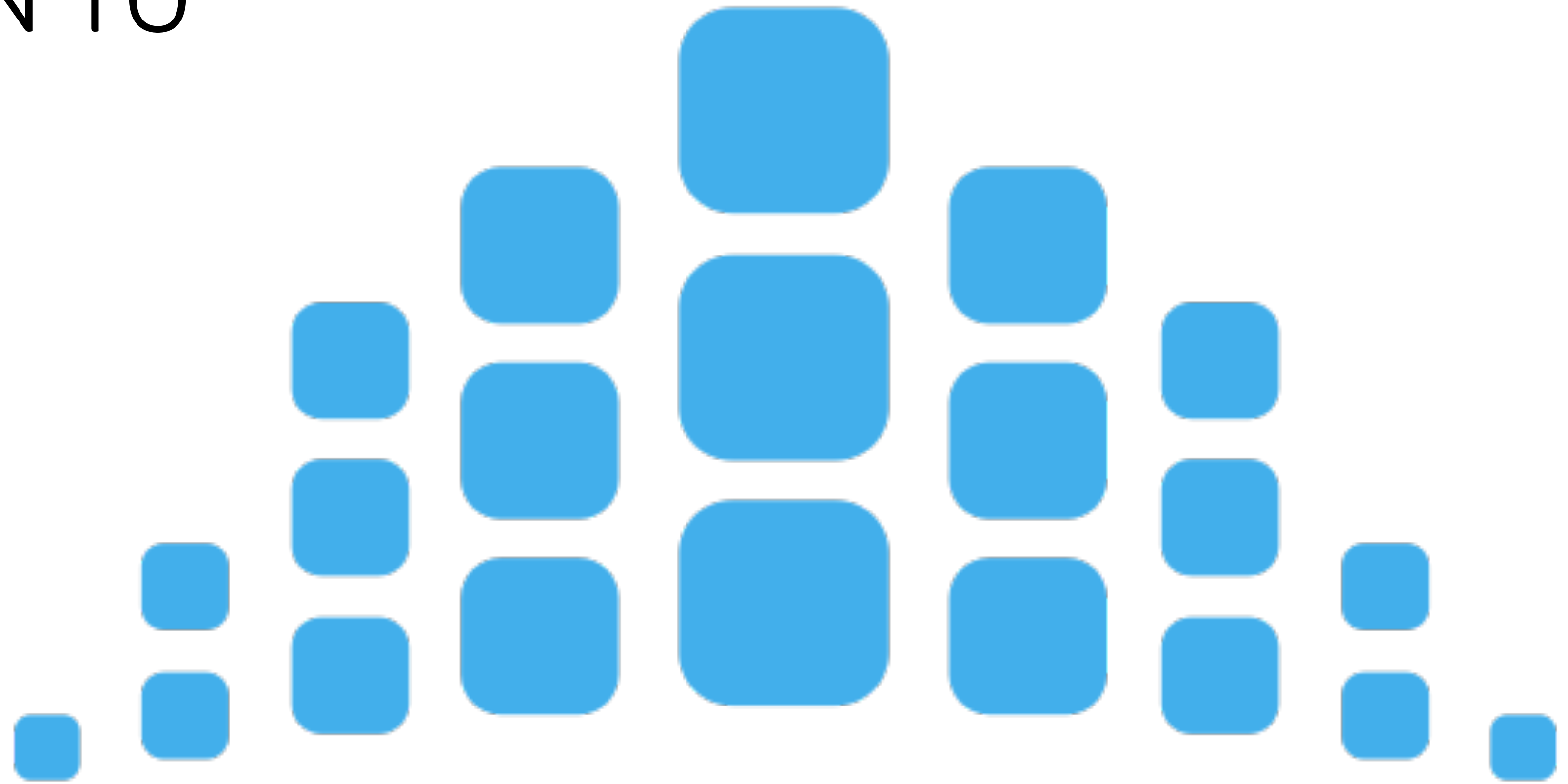


INTRODUCTION TO SLURM



Dany Vohl

Astronomy Data and Computing Services,
Swinburne

TABLE OF CONTENT

- Overview
- Behind the scene: Slurm's architecture
- User commands
 - *Commands summary*
- Gathering information
- Creating a job
- Script examples

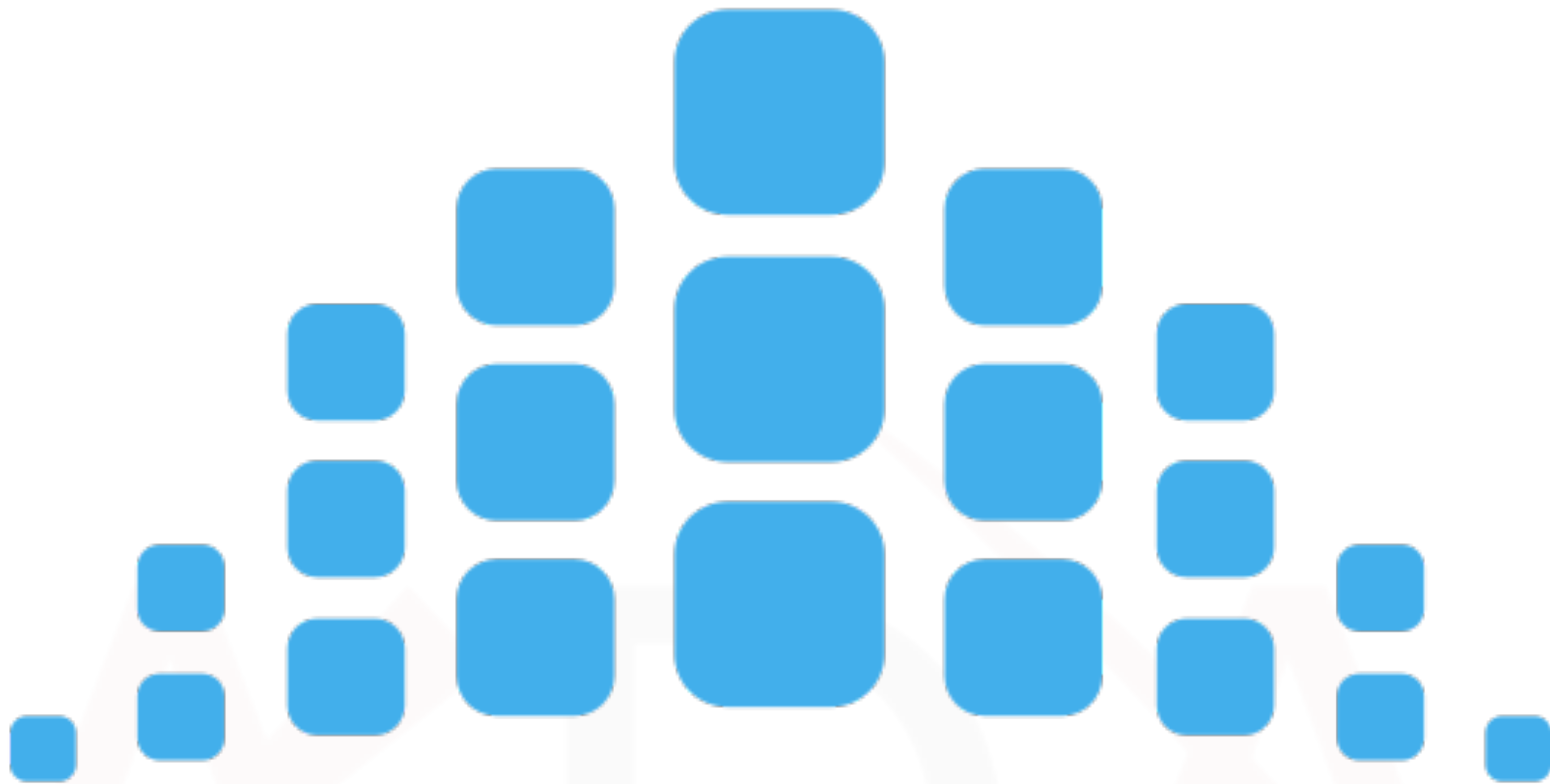
ADS
ASTRONOMY DATA AND COMPUTING SERVICES

REFERENCES

- Slurm Documentation: <https://slurm.schedmd.com>
- C.E.C.I. "Slurm Quick Start Tutorial": <https://bit.ly/2GETVCZ>
- OzSTAR Documentation: supercomputing.swin.edu.au/docs
- https://en.wikipedia.org/wiki/Embarrassingly_parallel

LIVE CODE EXECUTED ON THE OZSTAR SUPERCOMPUTER

- e.g. `ssh [your-username]@ozstar.swin.edu.au`



SLURM

Before we begin

ASTRONOMY DATA AND COMPUTING SERVICES

FOREWORD

INTRODUCTION TO HIGH PERFORMANCE COMPUTING (HPC)

*introduction to working with HPC
computing clusters*

Gregory B. Poole

Astronomy Data and Computing Services, Swinburne

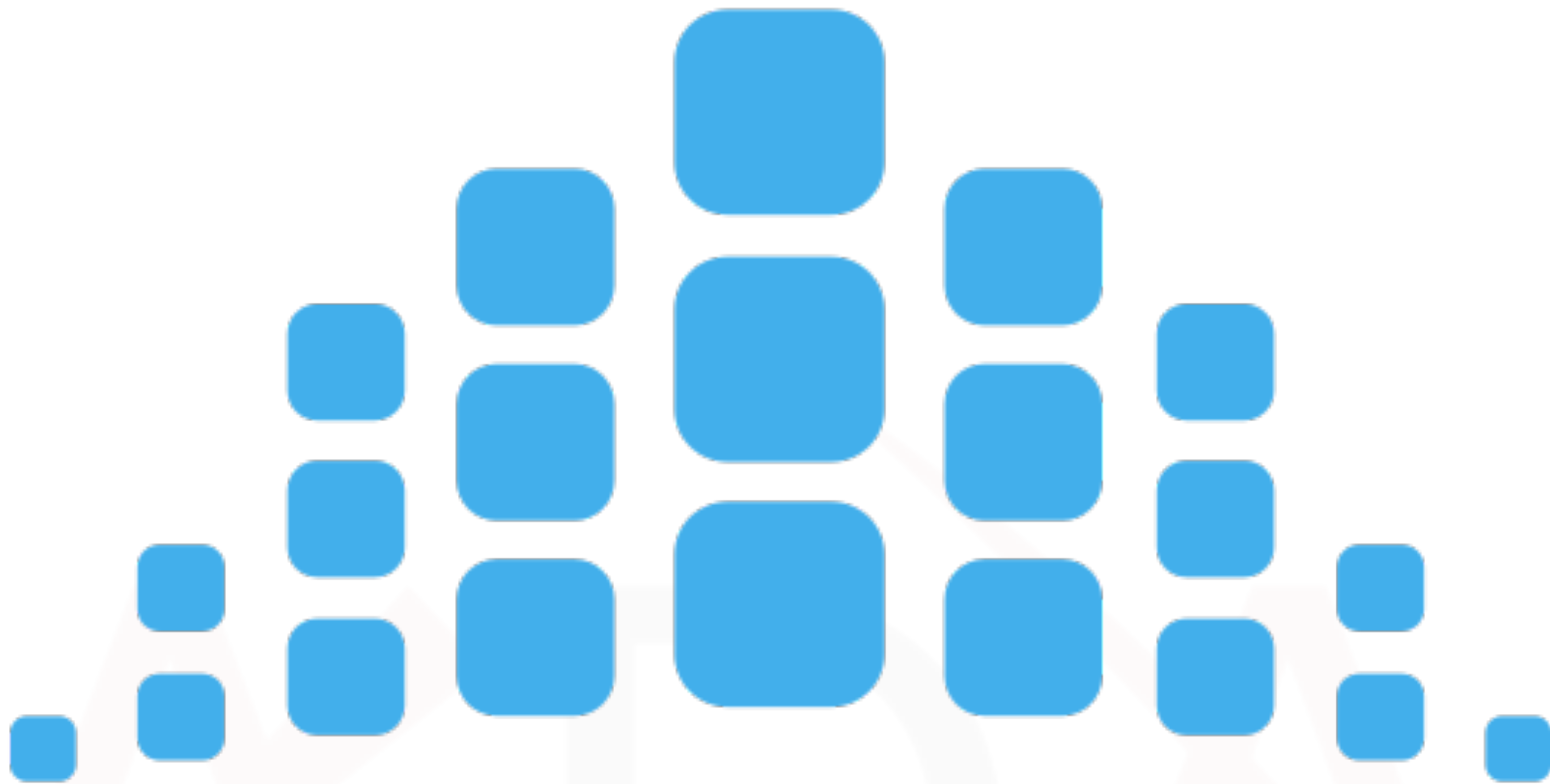


*Based on content developed by
Amr H. Hassan*



TABLE OF CONTENTS

- Introduction to parallel computing
- Introduction to computing clusters - Hardware configuration
 - Node layout
 - Storage
- Introduction to computing clusters - Queues
- Operating a computing cluster
 - Shell scripts
 - Working with queues
 - Logging in with SSH



SLURM

overview

ASTRONOMY DATA AND COMPUTING SERVICES

OVERVIEW

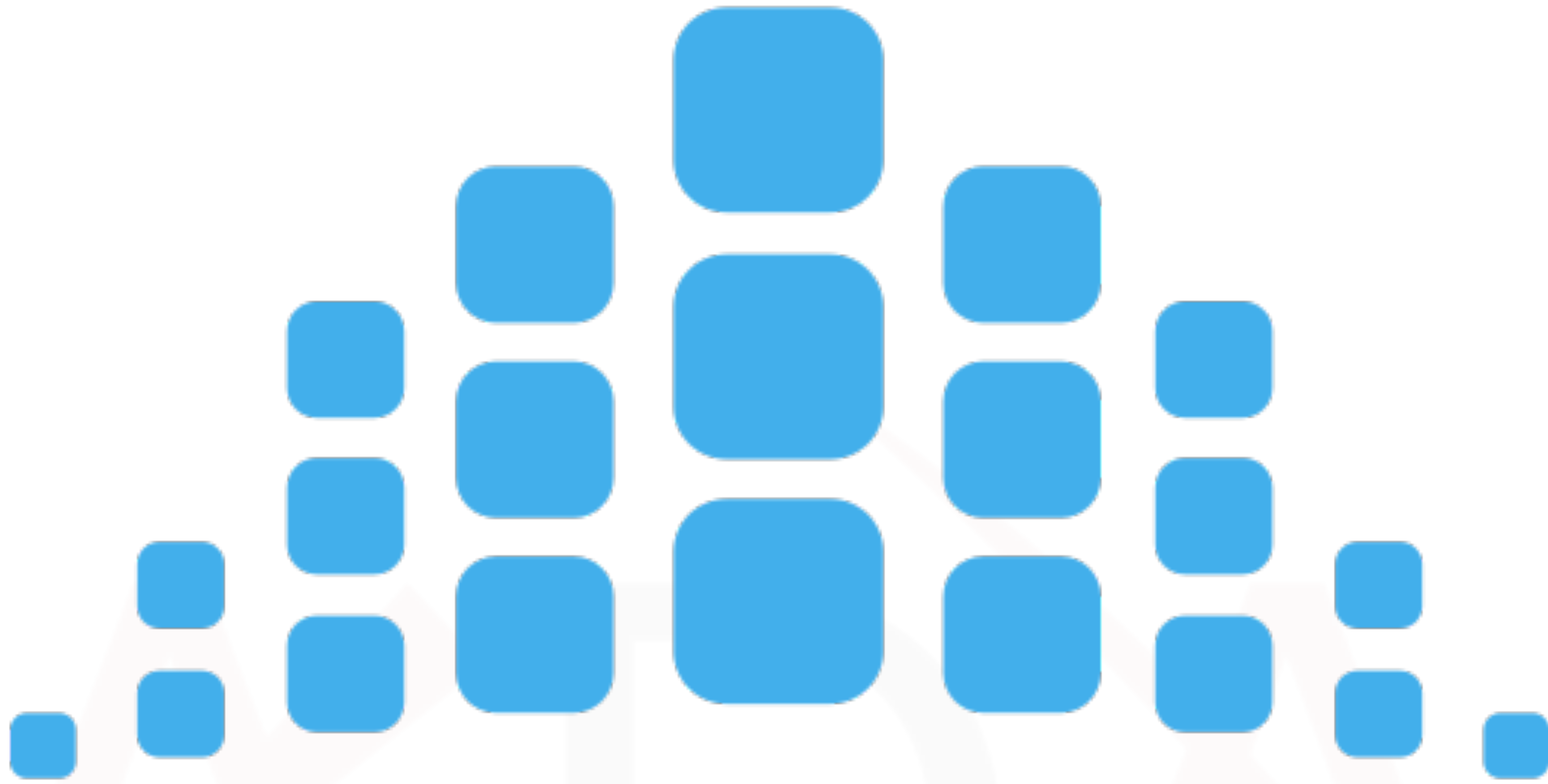
- Generally, the preferred way to use a supercomputer is through submitting batch jobs to a scheduler system which governs accesses to the compute nodes.
- The basic goal of a queue system is to maximize utilisation of the supercomputer and to do this in a way that is fair to all users.
- It should also ease the workload for users who do a lot of computation.

OVERVIEW

- **Slurm** is an open source, fault-tolerant, and highly scalable cluster management and job scheduling system for large and small Linux clusters
- Slurm was first developed at the Livermore Computing Center, and has grown into a complete open-source software backed up by a large community, commercially supported by the original developers, and used by many of the Top500 supercomputers.
- It is therefore worth learning its mechanisms.

OVERVIEW

- As a cluster workload manager, Slurm has three key functions
 - *Allocation of exclusive and/or non-exclusive access to resources (compute nodes) to users for some duration of time*
 - *Provide a framework for starting, executing, and monitoring work (normally a parallel job) on the set of allocated nodes*
 - *Arbitrate contention for resources by managing a queue of pending work*



SLURM

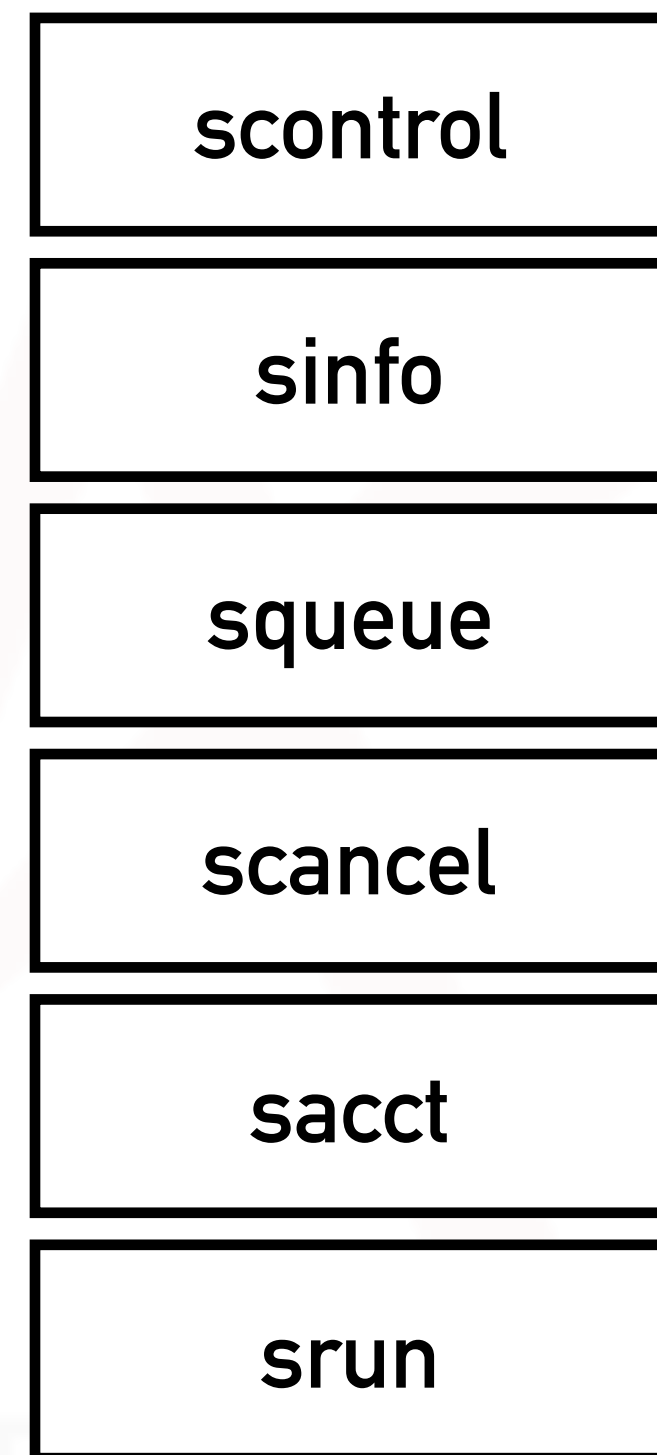
*Behind the scene:
Slurm's architecture*

ARCHITECTURE

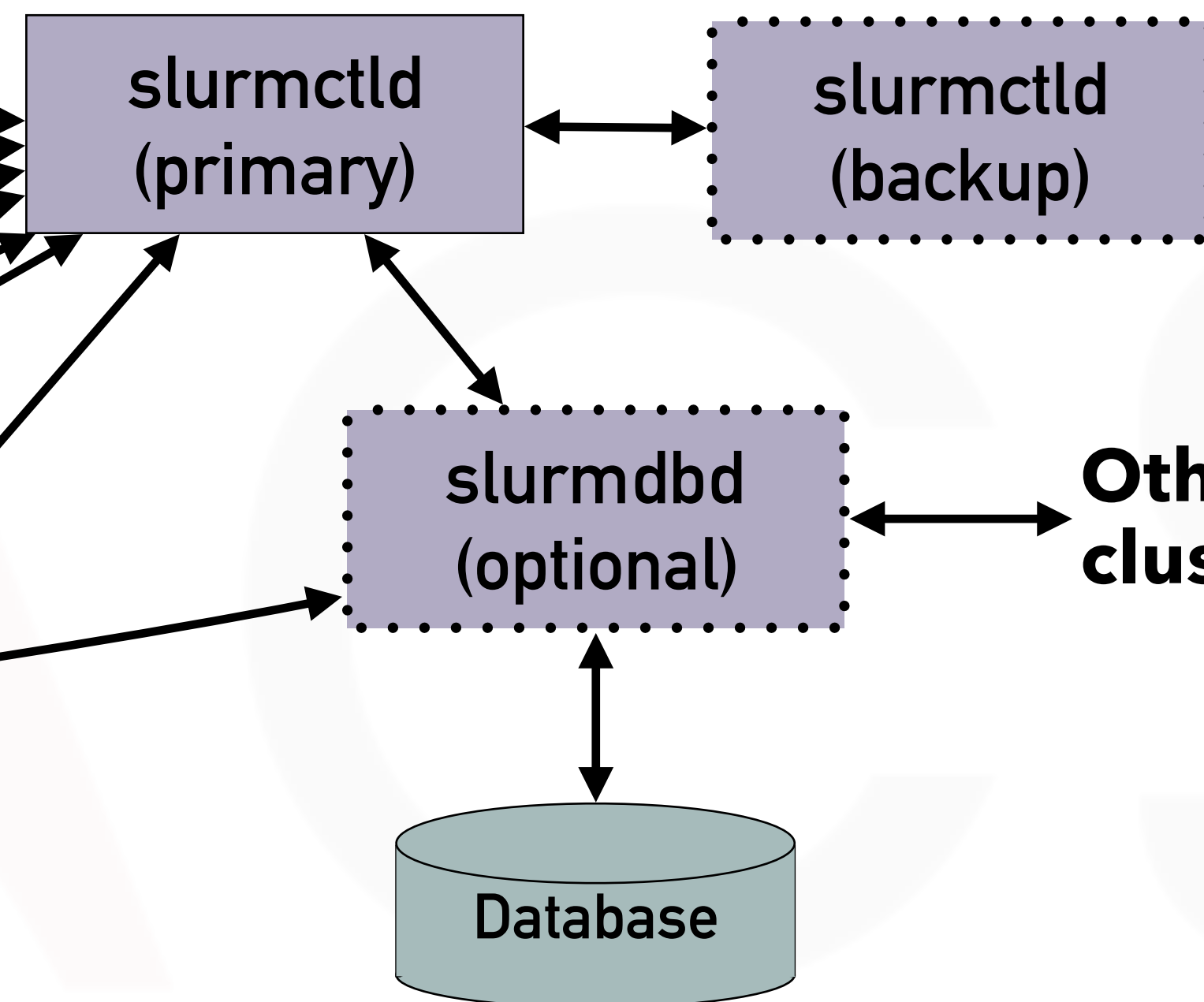
- Slurm consists of a daemon (called **slurmd**) running on each compute node and a central daemon (called **slurmctld**) running on a management node (with optional fail-over twin).
- The **slurmd** daemons provide fault-tolerant hierarchical communications.
- Information can be queried using several user commands:
sacct, salloc, sattach, sbatch, sbcast, scancel, scontrol, sinfo, smap, squeue, srun, strigger and **sview**.
- All of user commands can run anywhere in the cluster.

ARCHITECTURE

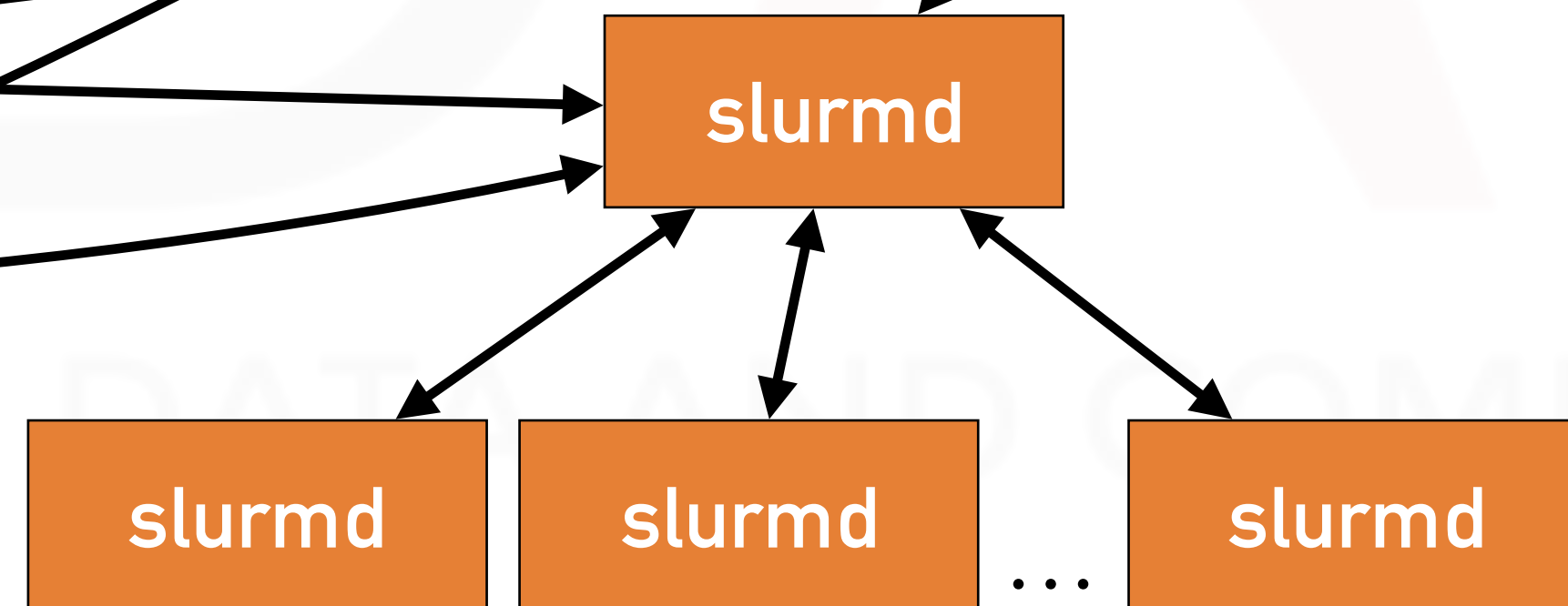
User commands (parial list)



Controller daemons



**Other
clusters**



Compute node daemons

ARCHITECTURE

- The entities managed by these Slurm daemons include
 - **nodes**: *the compute resource in Slurm*
 - **partitions**: *group nodes into logical (possibly overlapping) sets, jobs, or allocations of resources assigned to a user for a specified amount of time*
 - **job steps**: *sets of (possibly parallel) tasks within a job*

ARCHITECTURE

- **Partitions** can be considered **job queues**
- Each of which has an assortment of constraints such as
 - job size limit
 - job time limit
 - users permitted to use it
 - etc.

ADACS

ASTRONOMY DATA AND COMPUTING SERVICES

ARCHITECTURE

- **Priority-ordered jobs** are **allocated nodes** within a **partition** until the resources (nodes, processors, memory, etc.) within that partition are exhausted.
- Once a job is assigned a set of nodes, the user is able to initiate parallel work in the form of job steps in any configuration within the allocation
- E.g.
 - *A single job step may be started that utilizes all nodes allocated to the job*
 - *Several job steps may independently use a portion of the allocation*

ASTRONOMY DATA AND COMPUTING SERVICES

ARCHITECTURE

➤ E.g.

- *A single job step may be started that utilizes all nodes allocated to the job*
- *Several job steps may independently use a portion of the allocation*

```
#!/bin/bash
#
#SBATCH --job-name=test_omp
#SBATCH --output=res_omp.txt
#
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4
#SBATCH --time=10:00
#SBATCH --mem-per-cpu=100

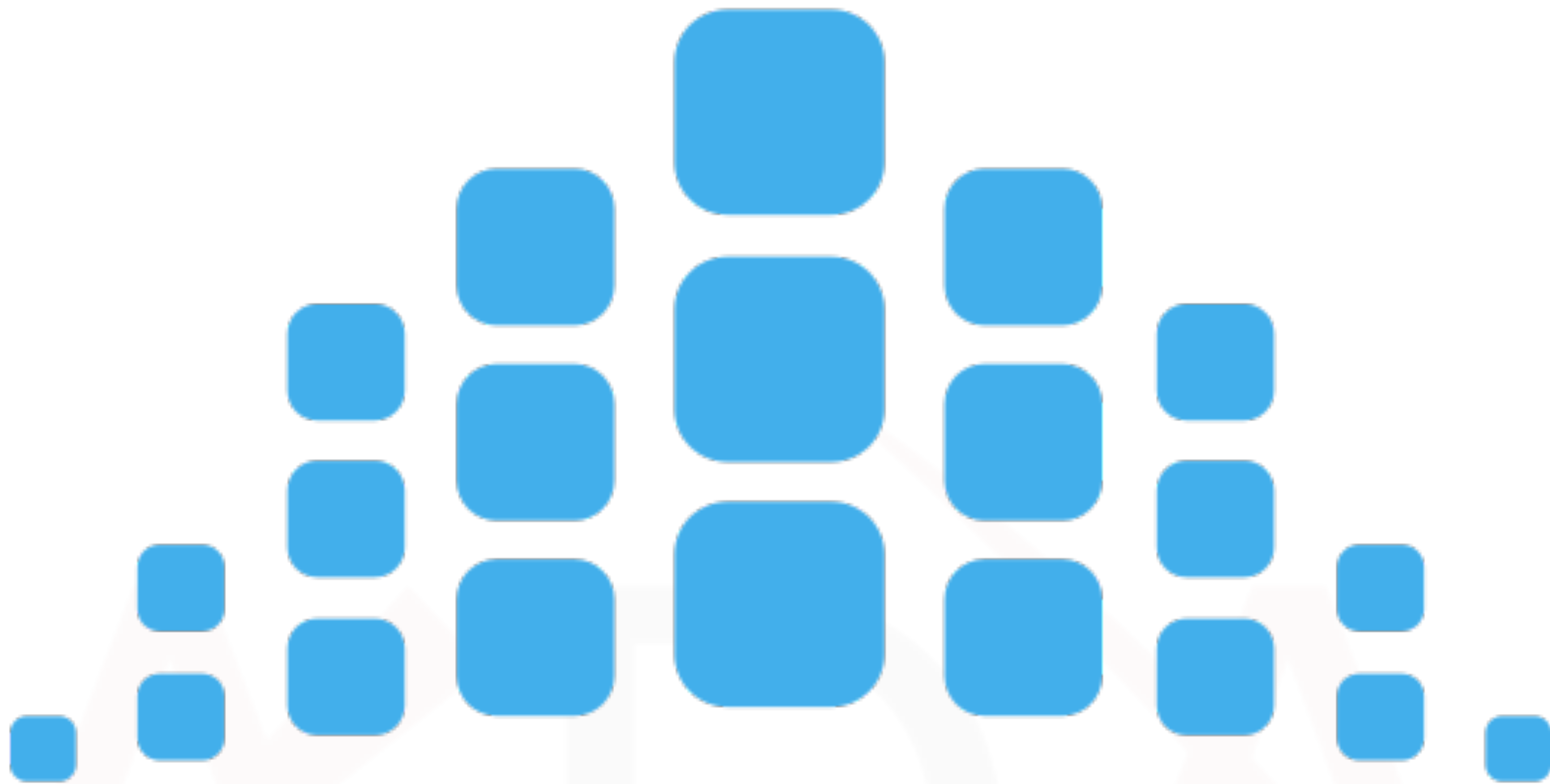
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
./hello.omp
```

ARCHITECTURE

➤ E.g.

- *A single job step may be started that utilizes all nodes allocated to the job*
- *Several job steps may independently use a portion of the allocation*

```
#!/bin/bash
#
#SBATCH --ntasks=8
for i in {1..1000}
do
    srun -n1 --exclusive ./myprog $i &
done
wait
```



SLURM

User commands

USER COMMANDS

- Manual pages exist **for all** Slurm daemons, commands, and API functions.
 - e.g. type ``man sinfo`` to see the manual for the `sinfo` command.
- The command option **--help** also provides a brief summary of options.
 - ``sinfo --help``
- Command options are **case sensitive**
 - e.g. ``sinfo -o`` is not the same as ``sinfo -O``

USER COMMANDS

- Now that we know the basics, let's glance at the different commands
 - *Note that if you want to know more about the specifics of each command, you can type the **<command> --help**, or **man <command>** in a shell.*

ADACS

ASTRONOMY DATA AND COMPUTING SERVICES

USER COMMANDS

➤ **sacct**

- *Report job or job step accounting information about active or completed jobs.*

➤ **salloc**

- *Allocate resources for a job in real time. Typically this is used to allocate resources and spawn a shell. The shell is then used to execute srun commands to launch parallel tasks.*

➤ **sattach**

- *Attach standard input, output, and error plus signal capabilities to a currently running job or job step. One can attach to and detach from jobs multiple times.*

USER COMMANDS

➤ **sbatch**

- *Submit a job script for later execution. The script will typically contain one or more srun commands to launch parallel tasks.*

➤ **sbcast**

- *Transfer a file from local disk to local disk on the nodes allocated to a job. This can be used to effectively use diskless compute nodes or provide improved performance relative to a shared file system.*

➤ **scancel**

- *Cancel a pending or running job or job step. It can also be used to send an arbitrary signal to all processes associated with a running job or job step.*

USER COMMANDS

➤ **scontrol**

- *The administrative tool used to view and/or modify Slurm state.
Note that many scontrol commands can only be executed as user root.*

➤ **sinfo**

- *Reports the state of partitions and nodes managed by Slurm.
It has a wide variety of filtering, sorting, and formatting options.*

➤ **smap**

- *Reports state information for jobs, partitions, and nodes managed by Slurm,
but graphically displays the information to reflect network topology.*

USER COMMANDS

➤ **squeue**

- *Reports the state of jobs or job steps. It has a wide variety of filtering, sorting, and formatting options. By default, it reports the running jobs in priority order and then the pending jobs in priority order.*

➤ **strigger**

- *Set, get or view event triggers. Event triggers include things such as nodes going down or jobs approaching their time limit.*

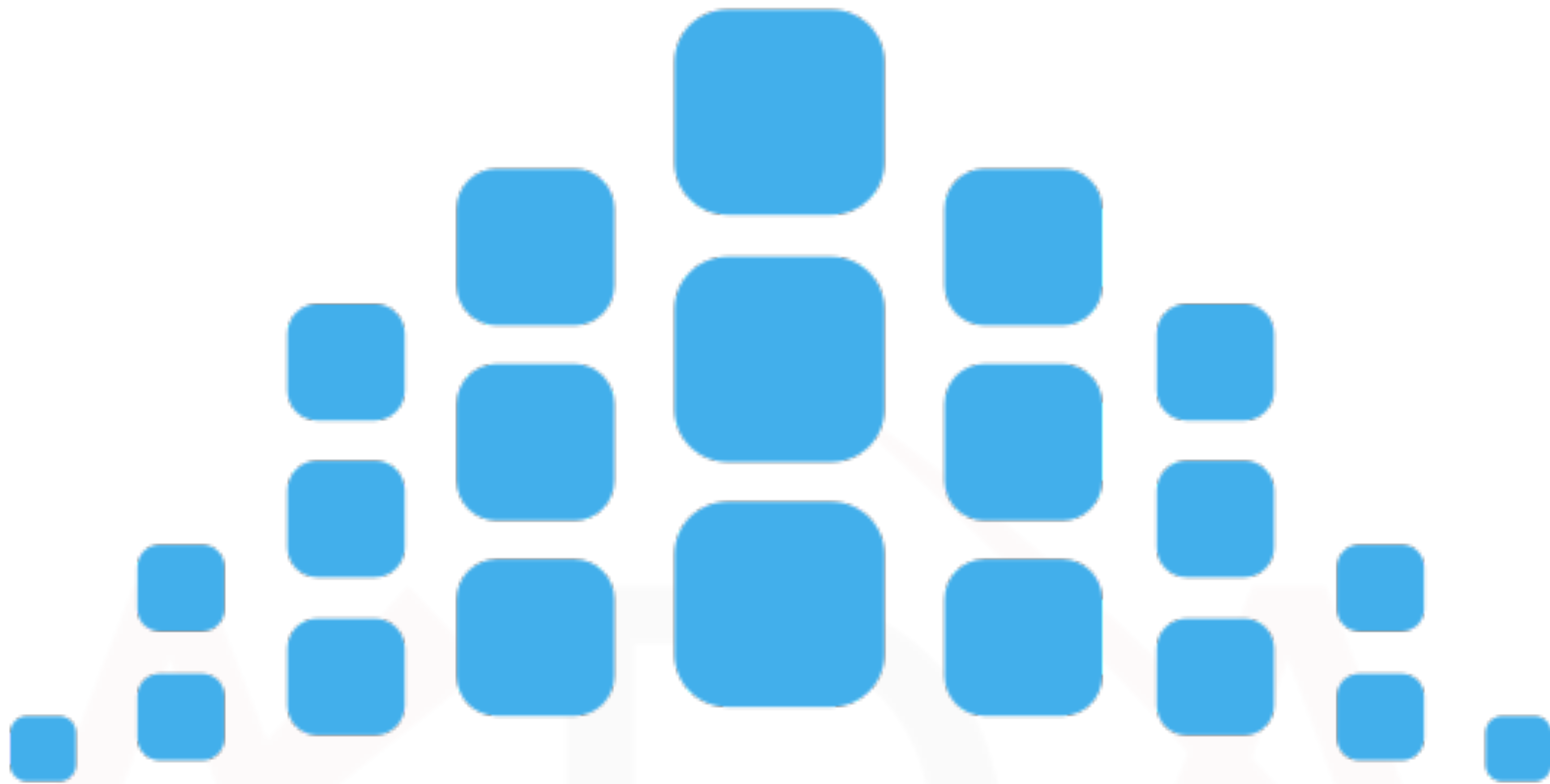
➤ **sview**

- *Graphical user interface to get and update state information for jobs, partitions, and nodes managed by Slurm.*

USER COMMANDS

➤ **srun**

- *Used to submit a job for execution or initiate job steps in real time.*
- *srun has a wide variety of options to specify resource requirements, including:*
 - minimum and maximum node count,
 - processor count,
 - specific nodes to use or not use,
 - and specific node characteristics (so much memory, disk space, certain required features, etc.).
- A job can contain multiple job steps executing sequentially or in parallel on independent or shared resources within the job's node allocation.



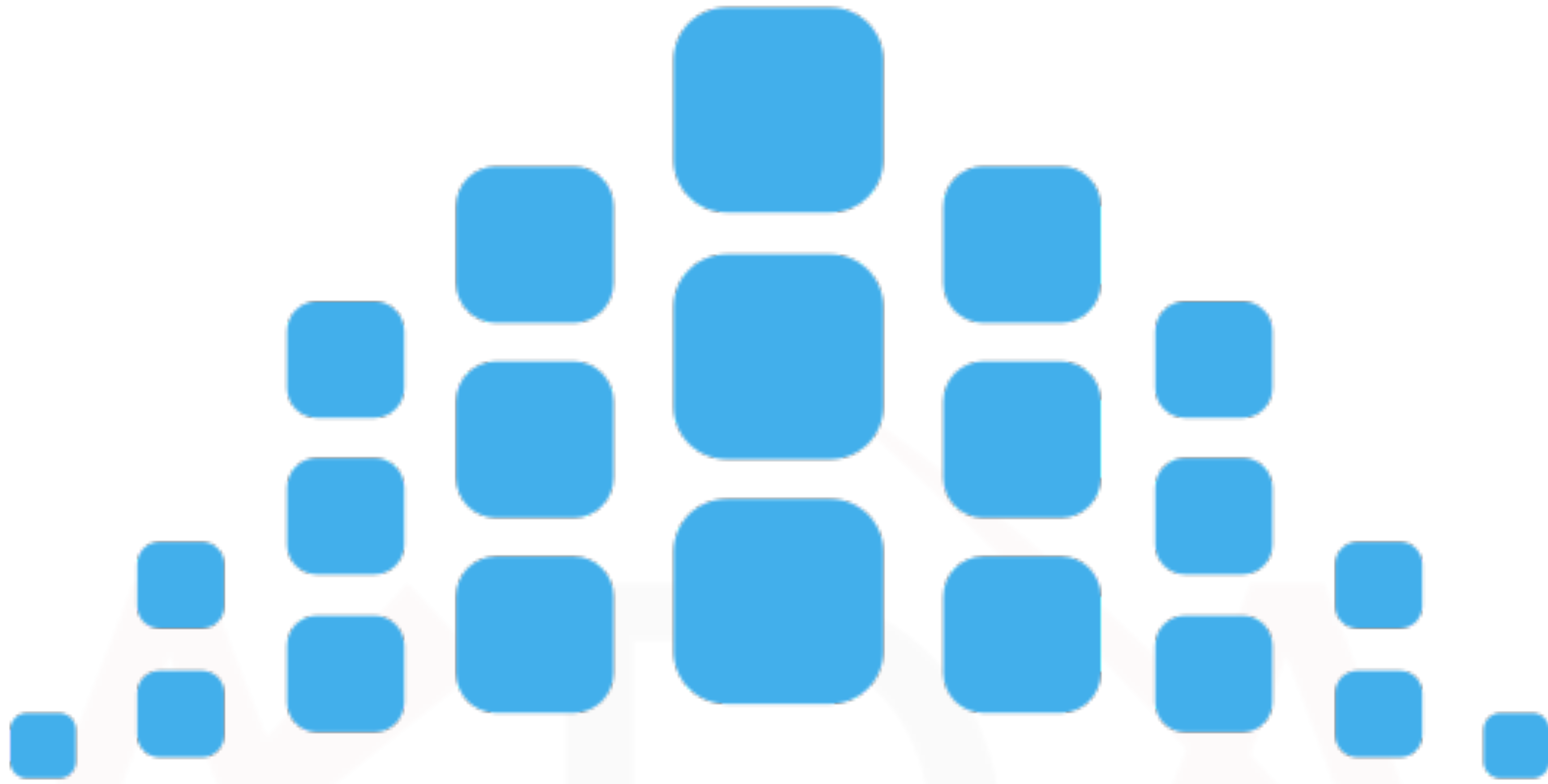
SLURM

Commands summary

ASTRONOMY DATA AND COMPUTING SERVICES

BRIEF COMMAND SUMMARY (AS AVAILABLE ON THE OZSTAR SUPERCOMPUTER)

- Job control and monitoring: `scontrol` and `squeue` and `showq` .
- Submit batch jobs: `sbatch yourJobScript` .
- Request interactive job sessions: through `sinteractive` .
- Launch/running a job: `srun` .
- Nodes info and cluster status: `sinfo` .
- Currently running job detailed status: `sstat` .
- Job and job steps accounting data: `sacct` .
- **Cancel a job:** `scancel jobid` .
 - Cancel all your jobs: `scancel -u username` .
 - Cancel all your pending jobs: `scancel -t PD` .



SLURM

Gathering information

ASTRONOMY DATA AND COMPUTING SERVICES

GATHERING INFORMATION

- Slurm includes many commands to interact with the system.
- For example, the **sinfo** command provides an overview of the resources offered by the cluster, and the **squeue** command shows to which jobs those resources are currently allocated.

GATHERING INFORMATION

- By default, **sinfo** lists the partitions that are available. A partition is a set of logically-grouped compute nodes.
- Example:

```
% sinfo
PARTITION    AVAIL  TIMELIMIT  NODES  STATE  NODELIST
skylake*      up 7-00:00:00      3  alloc  john[31,98,103]
skylake*      up 7-00:00:00     33  idle   bryan[1-3,5-7],
                                     john[1-4,30,44,70,72-73,87-97,100-102,104-107]
skylake-gpu   up 7-00:00:00     33  idle ↵
                                     bryan[1-3,5-7],john[1-4,30,44,70,72-73,87-97,100-102,104-107]
kn1           up 1-00:00:00      3  idle   gina[1-2,4]
```

GATHERING INFORMATION

- **sinfo** can output the information in a node-oriented fashion, with the argument **-N**
- Also, the **-l** argument allow to print more information about the nodes:
 - *E.g. Number of CPUs, Memory, temporary disk (also called scratch space; TMP_DISK), node weight, features of the nodes (e.g. AVAIL_FE), reason why a node is down, if applicable (REASON)...*

```
% sinfo -N -l
Thu Jan 25 15:43:52 2018
NODELIST      NODES PARTITION      STATE CPUS      S:C:T MEMORY TMP_DISK WEIGHT AVAIL_FE REASON
bryan1         1  skylake*      idle  36      2:18:1 384000      0    1000  (null) none
bryan2         1  skylake*      idle  36      2:18:1 384000      0    1000  (null) none
(...)
john106         1  skylake*      idle  36      2:18:1 191000      0      1  (null) none
john107         1  skylake*      idle  36      2:18:1 191000      0      1  (null) none
```

GATHERING INFORMATION

- The **squeue** command shows the list of jobs which are currently running (they are in the RUNNING state, noted as 'R') or waiting for resources (noted as 'PD', short for PENDING).

```
% squeue
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
12345   skylake job1 dave  R  0:21    4 gina[1-4]
12346   skylake job2 dave PD  0:00    8 (Resources)
12348   skylake job3 ed   PD  0:00    4 (Priority)
```

- Note that many options are available to filter the output
 - e.g. **--user**, **--partition**, **--state**, etc.

GATHERING INFORMATION

```
% queue
```

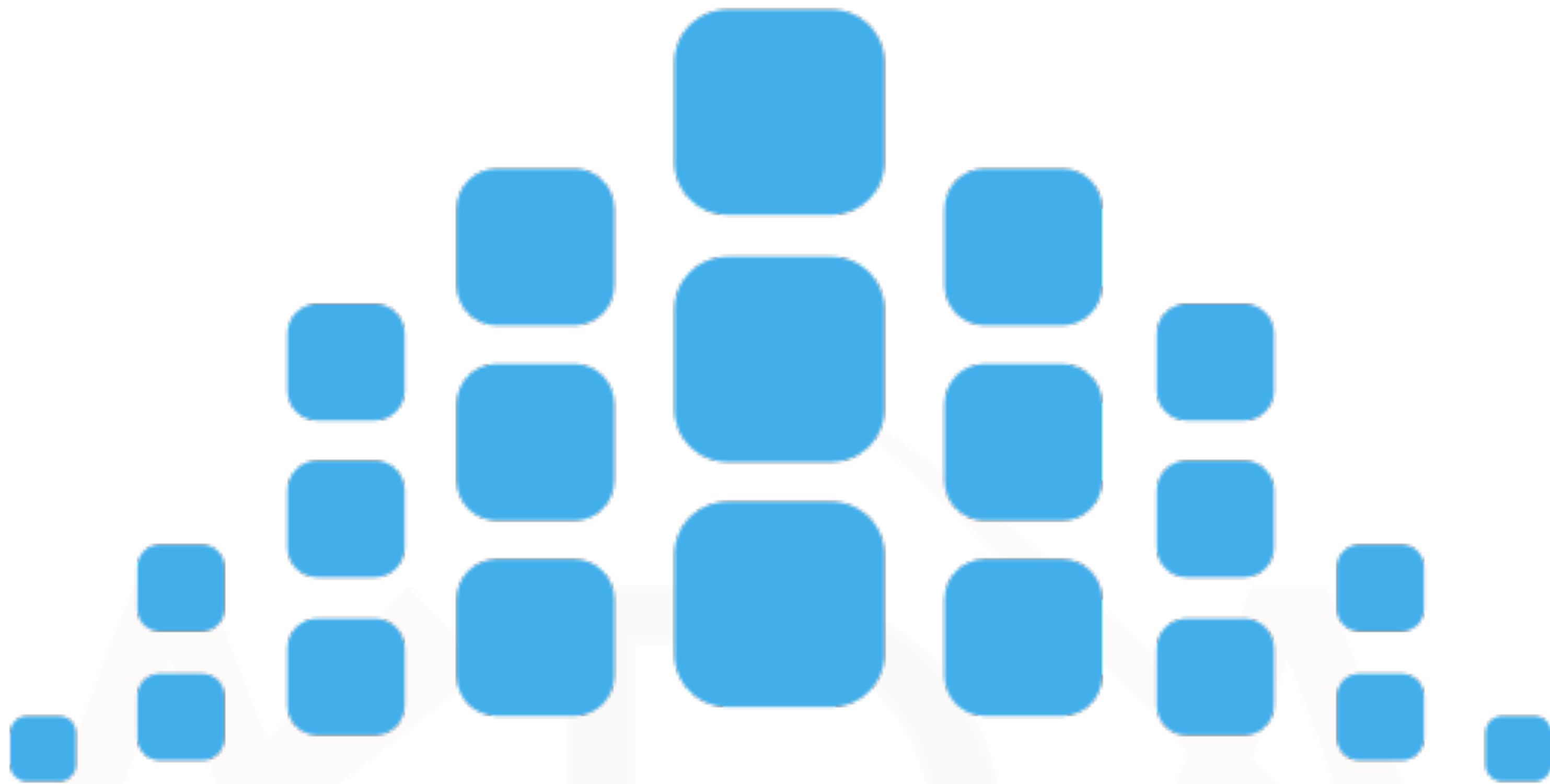
JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
12345	skylake	job1	dave	R	0:21	4	gina[1-4]
12346	skylake	job2	dave	PD	0:00	8	(Resources)
12348	skylake	job3	ed	PD	0:00	4	(Priority)

```
% queue --user dave
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
12345	skylake	job1	dave	R	0:21	4	gina[1-4]
12346	skylake	job2	dave	PD	0:00	8	(Resources)

```
% queue --state R
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
12345	skylake	job1	dave	R	0:21	4	gina[1-4]



SLURM

Creating a job

ASTRONOMY DATA AND COMPUTING SERVICES

CREATING A JOB

- A job is formed of two sections: **resource requests** and **job steps**.
- **Resource requests** involves specifying:
 - *the required a number of CPUs/GPUs*
 - *expected job duration*
 - *amounts of RAM*
 - *disk space*
 - *etc.*

CREATING A JOB

- A job is formed of two sections: **resource requests** and **job steps**.
- **Job steps** involve describing what needs to be done (i.e. computing steps, which software to run, parameter space, etc.).

ADACS

ASTRONOMY DATA AND COMPUTING SERVICES

CREATING A JOB

- Typically, a job is created via a submission script (e.g. a shell script).
- The very first line of the submission file has to be the hashbang (e.g. `#!/bin/bash`). Then, the next lines must be the SBATCH directives. Finally, you can input any other line.
- For example, **comments** (a line starting with `#`) **prefixed with SBATCH** at the beginning of a Bash script **are understood** by Slurm **as parameters describing resource requests and other submissions options**.
- A complete list of parameters can be obtained from the sbatch manual page (**man sbatch**).
- The script itself is a job step. Other job steps are created with the `srun` command.

CREATING A JOB

- Here is an example bash script named e.g. submit.sh:

```
#!/bin/bash
#
#SBATCH --job-name=test
#SBATCH --output=res.txt
#
#SBATCH --ntasks=1
#SBATCH --time=60:00
#SBATCH --mem-per-cpu=200

srun hostname
srun sleep 60
```

- To submit it to the queue:

```
% sbatch submit.sh
sbatch: Submitted batch job 99999999
```

JOB STEPS ON THE QUEUE

- Once a job has been submitted to a queue with **sbatch**, execution will follow these steps/states:
 - **PENDING**: *The job then enters the queue in the PENDING state.*
 - **RUNNING**: *Once resources become available and the job has highest priority, an allocation is created for it and it goes to the RUNNING state.*
 - *If the job completes correctly, it goes to the **COMPLETED** state, otherwise, it is set to the **FAILED** state.*

QUERYING JOB STATE

- It is possible to query information about a job in near-realtime (memory consumption, etc.) with the **sstat** command, by introducing **sstat -j jobid**.
- It is possible to select specific information to output with **sstat** via the **--format** parameter.
- For more details, refer to the manual page (e.g. **man sstat**).

PARALLEL JOBS

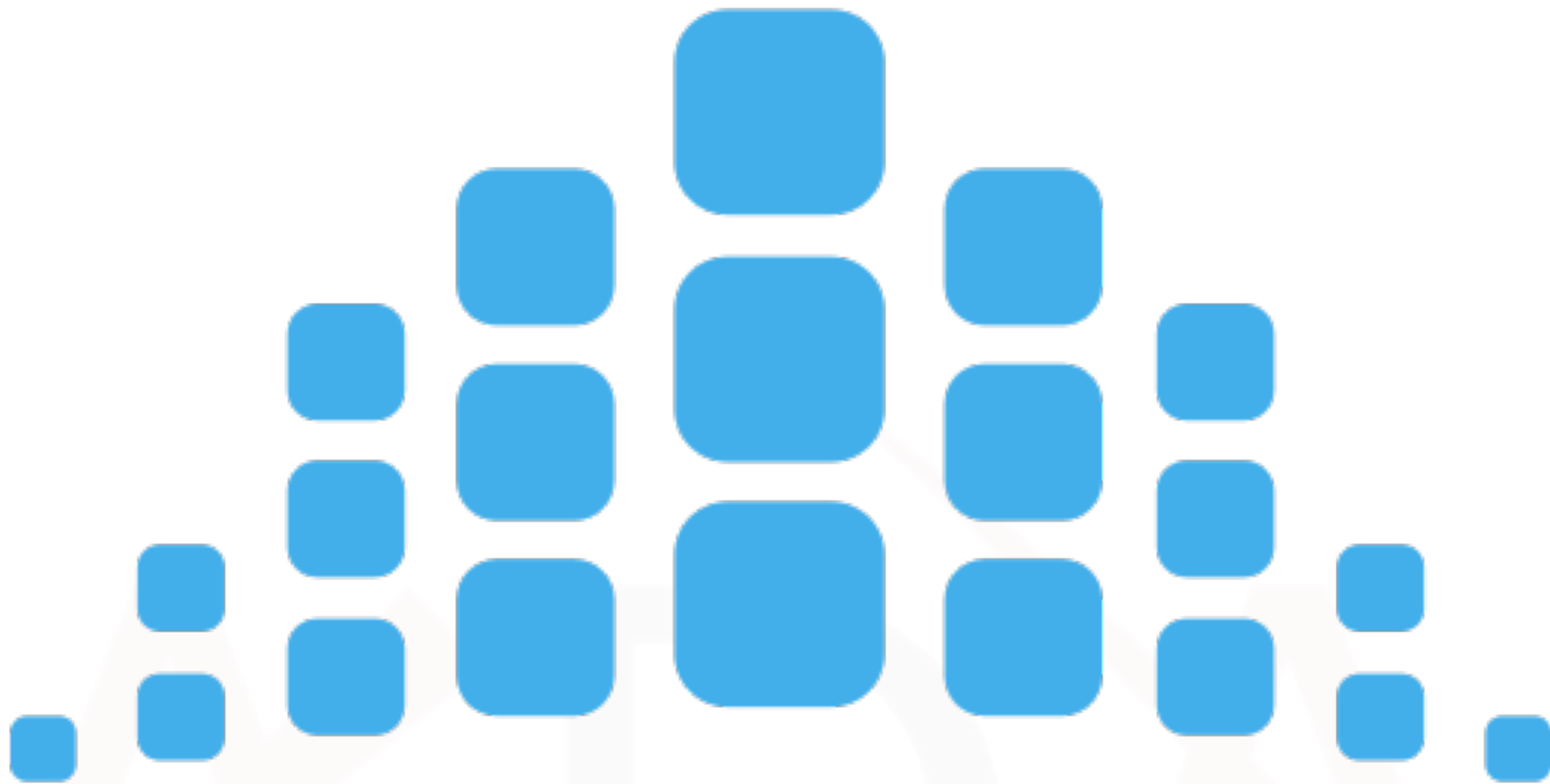
- Parallel job (e.g. tasks ran simultaneously) can be created via different methods by running:
 - **a multi-process program**
(Single Process, Multiple Data (SPMD) paradigm, e.g. with MPI)
 - **a multi-threaded program**
(shared memory paradigm, e.g. with OpenMP or pthreads)
 - **several instances of a single-threaded program**
(Embarrassingly parallel paradigm or a job array)
 - **one master program controlling several slave programs**
(master/slave paradigm)

PARALLEL JOBS

- In the context of Slurm,
 - *a task represents a process;*
 - *a multi-process program is made of several tasks.*
 - By contrast, a multi-threaded program is composed of only one task, which uses several CPUs.

PARALLEL JOBS

- Tasks are requested/created with the **--ntasks** option, while CPUs, for the multithreaded programs, are requested with the **--cpus-per-task** option.
- *Tasks cannot be split across several compute nodes, so requesting several CPUs with the **--cpus-per-task** option will ensure all CPUs are allocated on the same compute node.*
- *By contrast, requesting the same amount of CPUs with the **--ntasks** option may lead to several CPUs being allocated on several, distinct compute nodes.*



SLURM

Script examples

ASTRONOMY DATA AND COMPUTING SERVICES

EXAMPLE #1: MPI

- Let's begin with a simple MPI example: Hello world.
 - An example C program can be found at http://en.wikipedia.org/wiki/Message_Passing_Interface#Example_program
 - Save this code as **wiki_mpi_example.c**
- Next we need to make an sbatch script
- We can either compile the program in advance (possibly better) or compile it before running the code in the sbatch script
 - For simplicity, let's do it all in the queue (but generally, we would compile it first as some debugging issues may need to be handled by us)
- We can send it to the queue typing **sbatch example_mpi.sbatch**

EXAMPLE #1: MPI

- Our **sbatch** script will look like this:

```
#!/bin/bash
#
#SBATCH --job-name=test_mpi
#SBATCH --output=res_mpi.txt
#
#SBATCH --ntasks=4
#SBATCH --time=10:00
#SBATCH --mem-per-cpu=100

module load gcc/6.4.0
module load openmpi/3.0.0

# Compile the C program
mpicc wiki_mpi_example.c -o hello.mpi

# Launch the mpi program
srun hello.mpi
```

EXAMPLE #1: MPI

- Once the job has been executed, the output can be found in the log file
 - e.g. **#SBATCH --output=res_mpi.txt**
 - *Note that we had set --tasks=4*

```
We have 4 processes.  
Process 1 reporting for duty.  
Process 2 reporting for duty.  
Process 3 reporting for duty.
```

EXAMPLE #2: OPENMP

- Next, let's try OpenMP.
 - *The job will be run in an allocation where four cores have been reserved on the same compute node.*
 - *An example C program can be found at https://en.wikipedia.org/wiki/OpenMP#Thread_creation*
 - *Save this code as **wiki_omp_example.c***
- The next steps are similar to the previous example

EXAMPLE #2: OPENMP

- Our **sbatch** script will look like this:

```
#!/bin/bash
#
#SBATCH --job-name=test_omp
#SBATCH --output=res_omp.txt
#
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=16
#SBATCH --time=10:00
#SBATCH --mem-per-cpu=100

# Compile example C code (http://en.wikipedia.org/wiki/Openmp#C)
gcc -fopenmp wiki_omp_example.c -o hello.omp

# Run the code
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
./hello.omp
```

EXAMPLE #2: OPENMP

- Once the job has been executed, the output can be found in the log file
 - e.g. **#SBATCH --output=res_mpi.txt**
 - Note that we had set `--cpus-per-task=16`
 - Therefore, it prints out hello world 16 times; once with each cpu core.
- ```
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
```

[illegible]

# EXAMPLE #2: OPENMP

```
[dvohl@farnarkle1 example_openmp]$ sbatch example_openmp.sbatch
Submitted batch job 304868
```

```
[dvohl@farnarkle1 example_openmp]$ showq
```

| ACTIVE JOBS----- |         |          |       |      |           |           |
|------------------|---------|----------|-------|------|-----------|-----------|
| JOBID            | JOBNAME | USERNAME | STATE | CORE | REMAINING | STARTTIME |
| =====            |         |          |       |      |           |           |
| 269110           | ...     |          |       |      |           |           |

93 active jobs

| WAITING JOBS----- |          |          |          |      |         |                     |
|-------------------|----------|----------|----------|------|---------|---------------------|
| JOBID             | JOBNAME  | USERNAME | STATE    | CORE | WCLIMIT | QUEUETIME           |
| =====             |          |          |          |      |         |                     |
| 304743            | ...      |          |          |      |         |                     |
| 304868            | test_omp | dvohl    | Priority | 16   | 0:10:00 | Wed Jun 13 10:32:42 |
| 303290            | ...      |          |          |      |         |                     |

| BLOCKED JOBS-- |         |          |       |      |         |           |
|----------------|---------|----------|-------|------|---------|-----------|
| JOBID          | JOBNAME | USERNAME | STATE | CORE | WCLIMIT | QUEUETIME |
| =====          |         |          |       |      |         |           |

Total Jobs: 102    Active Jobs: 93    Idle Jobs: 6    Blocked Jobs: 3

## EXAMPLE #3: EMBARRASSINGLY PARALLEL PROBLEM

---

- We call embarrassingly parallel a problem where little or no effort is needed to separate the problem into a number of parallel tasks.
- For example, this is the case for problems based on random draws (e.g. **Monte-Carlo simulations**). In such cases, you can have four programs drawing 1000 random samples and combining their output afterwards (say, with another program), that result in the equivalent of drawing 4000 samples.
- Another typical example is a **parameter sweep**. In this case, the same task is computed several times, with the only difference in the initial value of some high-level parameter at each run.

## EXAMPLE #3: EMBARRASSINGLY PARALLEL PROBLEM

---

- An example of parameter sweep could be the optimisation of an integer-valued parameter by exploring a range:

```
#!/bin/bash
#
#SBATCH --job-name=test_emb_arr
#SBATCH --output=res_emb_arr.txt
#
#SBATCH --ntasks=1
#SBATCH --time=10:00
#SBATCH --mem-per-cpu=100
#
#SBATCH --array=1-8

srun ./my_program $SLURM_ARRAY_TASK_ID
```

## EXAMPLE #3: EMBARRASSINGLY PARALLEL PROBLEM

---

```
#!/bin/bash
#
#SBATCH --job-name=test_emb_arr
#SBATCH --output=res_emb_arr.txt
#
#SBATCH --ntasks=1
#SBATCH --time=10:00
#SBATCH --mem-per-cpu=100
#
#SBATCH --array=1-8

srun ./my_program $SLURM_ARRAY_TASK_ID
```

- In this configuration, the command **my\_program** will be run eight times, creating eight distinct jobs, each time with a different argument passed with the environment variable defined by slurm **SLURM\_ARRAY\_TASK\_ID** ranging from 1 to 8 (--array=).

## EXAMPLE #4: PACKED JOBS

---

- The **srun** command has the **--exclusive** argument that allows scheduling independent processes inside a Slurm job allocation. As stated in the documentation:

This option can also be used when initiating more than one job step within an existing resource allocation, where you want separate processors to be dedicated to each job step. If sufficient processors are not available to initiate the job step, it will be deferred. This can be thought of as providing a mechanism for resource management to the job within its allocation.

## EXAMPLE #4: PACKED JOBS

---

- As an example, the following job submission script will ask Slurm for 8 CPUs, then it will run the myprog program 1000 times with arguments passed from 1 to 1000.
- Using the **-n1 --exclusive** option, the job will ensure that only 8 instances are running at any point in time, each being allocated one CPU.

```
#!/bin/bash
#
#SBATCH --ntasks=8
for i in {1..1000}
do
 srun -n1 --exclusive ./myprog $i &
done
wait
```

## EXAMPLE #4: PACKED JOBS

---

- Similarly, many files can be processed with one job submission script. The following script will run **myprog** for every file in **/path/to/data**, but maximum 8 at a time, and using one CPU per task.

```
#!/bin/bash
#
#SBATCH --ntasks=8
for file in /path/to/data/*
do
 srun -n1 --exclusive ./myprog $file &
done
wait
```

- Note the **&** symbol at the end of the execution line.  
It is used to detach from this process for future executions.

# FINAL THOUGHTS

---

- Now that we have covered most user commands, the different types of parallel jobs, along with practical examples, you should now be able to start using Slurm on your own.
- Don't forget the manual pages (`man <command>`) and `--help` option to learn more about what is available to you!