

# Job Parallelization with GNU Parallel and Slurm Arrays

Georgia Advanced Computing Resource Center  
University of Georgia  
Isaiah Davis  
[isaiah.davis@uga.edu](mailto:isaiah.davis@uga.edu)

# Job Parallelism

Job parallelism refers to the practice of dividing a computational task into smaller tasks that can be run concurrently, typically across multiple processors or computers.

## Benefits:

- Speed – Complete tasks faster by working on multiple parts at once.
- Scalability – Handle larger datasets or more complex simulations.
- Efficiency – Make better use of available hardware resources.

## Types of parallelism:

- Embarrassingly Parallel
- Complex Parallelization (*e.g. Interdependent Tasks*)

# Embarrassingly Parallel vs. Complex Parallelization

| Type                    | Description                                                                       | Examples                                                                    | Challenges                                                          |
|-------------------------|-----------------------------------------------------------------------------------|-----------------------------------------------------------------------------|---------------------------------------------------------------------|
| Embarrassingly Parallel | Tasks can run completely independently with no need to communicate or share data. | Image rendering, Monte Carlo simulations, batch data processing.            | Minimal—easy to implement and scale.                                |
| Complex Parallelization | Tasks are interdependent and may need to exchange data during execution.          | Simulations with shared state, machine learning training, weather modeling. | Requires coordination, synchronization, and efficient data sharing. |



# Job Parallelism

## Covered Today:

- Embarrassingly parallel problems, using GNU Parallel and Slurm Array jobs.

## Not Covered Today:

- Programming libraries for parallelism.
  - *e.g. doParallel (R), multiprocessing (Python), OpenMP, Pthreads, OpenMPI*

# Job Parallelism

## Today's Focus:

- Running multiple instances of the same command at the same time.

```
command input1  
command input2  
command input3  
command input4  
command input5
```

## Example

- Use the `process_data` command with the files:
  - `input1.csv` `input2.csv` `input3.csv` `input4.csv` `input5.csv`



- Example
  - Use the `process_data` command with the files:
    - `input1.csv` `input2.csv` `input3.csv` `input4.csv` `input5.csv`

```
#!/bin/bash
#SBATCH --partition=batch
#SBATCH --job-name=test
#SBATCH --ntasks=1
#SBATCH --time=01:00:00
#SBATCH --mem=1G
```

```
process_data input1.csv
process_data input2.csv
process_data input3.csv
process_data input4.csv
process_data input5.csv
```

**Manually write out the command for each input.**

- Example
  - Use the `process_data` command with the files:
    - `input1.csv` `input2.csv` `input3.csv` `input4.csv` `input5.csv`

- 1) Create a file containing the name of each input.
- 2) Iterate through the file using a for loop.

`inputs.txt`

```
input1.csv  
input2.csv  
input3.csv  
input4.csv  
input5.csv
```

```
#!/bin/bash  
#SBATCH --partition=batch  
#SBATCH --job-name=test  
#SBATCH --ntasks=1  
#SBATCH --time=01:00:00  
#SBATCH --mem=1G  
  
for FILE in $(cat inputs.txt)  
do  
    process_data ${FILE}  
done
```



- Example
  - Use the `process_data` command with the files:
    - `input1.csv` `input2.csv` `input3.csv` `input4.csv` `input5.csv`

### Sequential Execution

```
#!/bin/bash
#SBATCH --partition=batch
#SBATCH --job-name=test
#SBATCH --ntasks=1
#SBATCH --time=01:00:00
#SBATCH --mem=1G
```

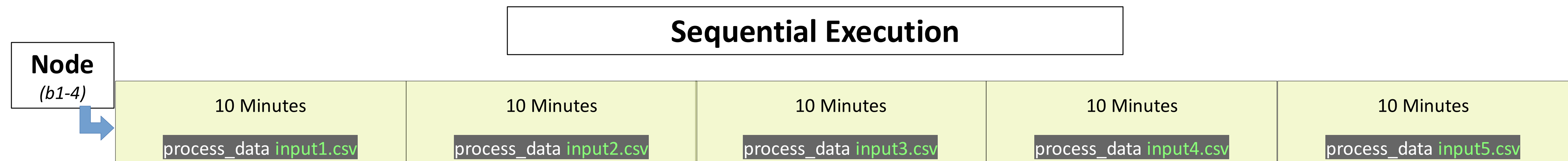
```
process_data input1.csv
process_data input2.csv
process_data input3.csv
process_data input4.csv
process_data input5.csv
```

```
#!/bin/bash
#SBATCH --partition=batch
#SBATCH --job-name=test
#SBATCH --ntasks=1
#SBATCH --time=01:00:00
#SBATCH --mem=1G
```

```
for FILE in $(cat inputs.txt)
do
    process_data ${FILE}
done
```

- Example
  - Use the `process_data` command with the files:

- `input1.csv` `input2.csv` `input3.csv` `input4.csv` `input5.csv`



**Total Time: 50 Minutes**

- Example

- Use the `process_data` command with the files:

- `input1.csv` `input2.csv` `input3.csv` `input4.csv` `input5.csv`

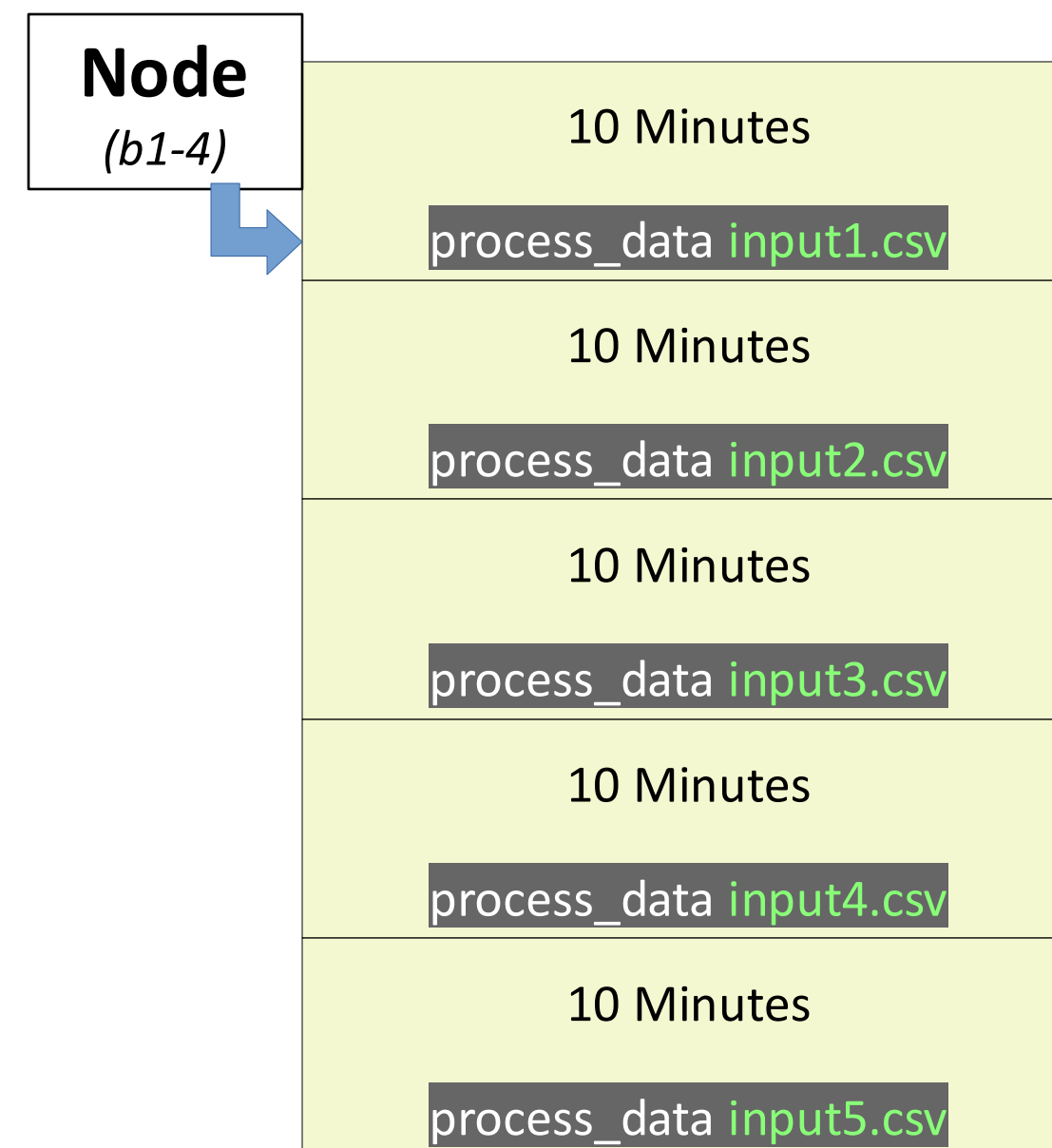
**Parallel Execution**

|                                                    |
|----------------------------------------------------|
| 10 Minutes<br><code>process_data input1.csv</code> |
| 10 Minutes<br><code>process_data input2.csv</code> |
| 10 Minutes<br><code>process_data input3.csv</code> |
| 10 Minutes<br><code>process_data input4.csv</code> |
| 10 Minutes<br><code>process_data input5.csv</code> |

**Total Time: ~10 Minutes**  
*(parallelization incurs some overhead)*



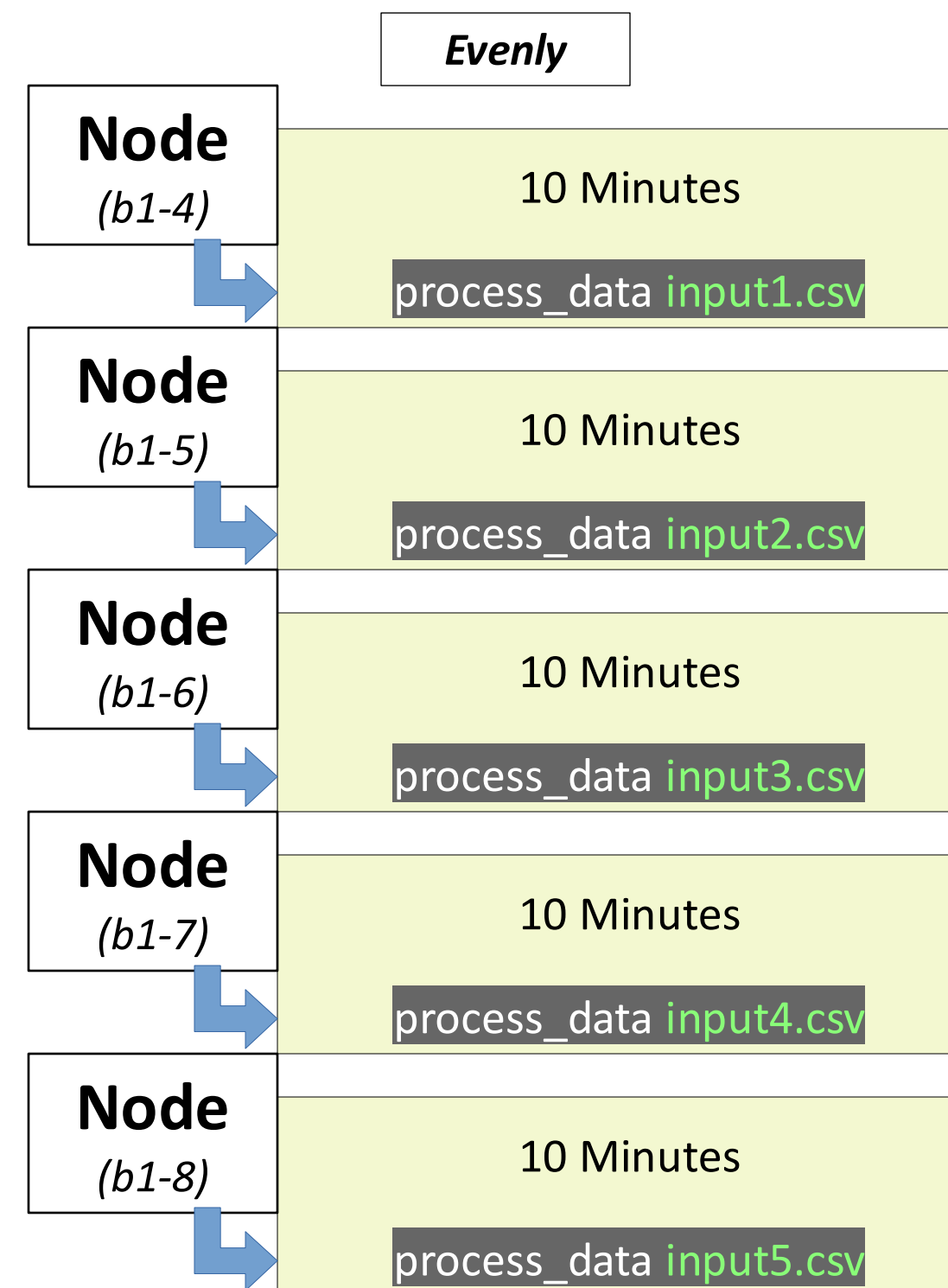
- Example
  - Use the `process_data` command with the files:
    - `input1.csv` `input2.csv` `input3.csv` `input4.csv` `input5.csv`



**Utilize multiple cores on a single Sapelo2 node.**

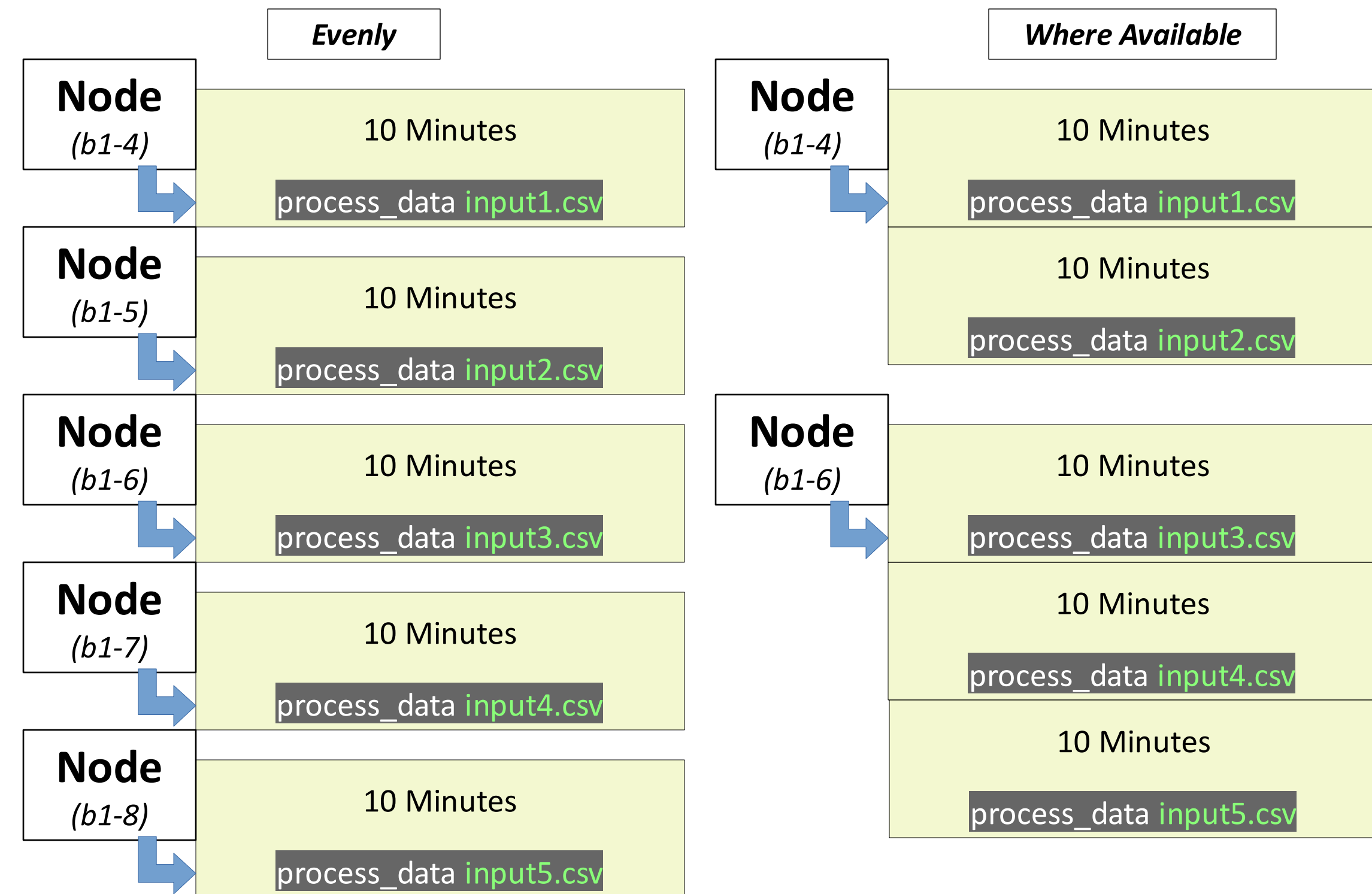
- Example
  - Use the `process_data` command with the files:
    - `input1.csv` `input2.csv` `input3.csv` `input4.csv` `input5.csv`

**Spread commands equally across multiple nodes on Sapelo2.**



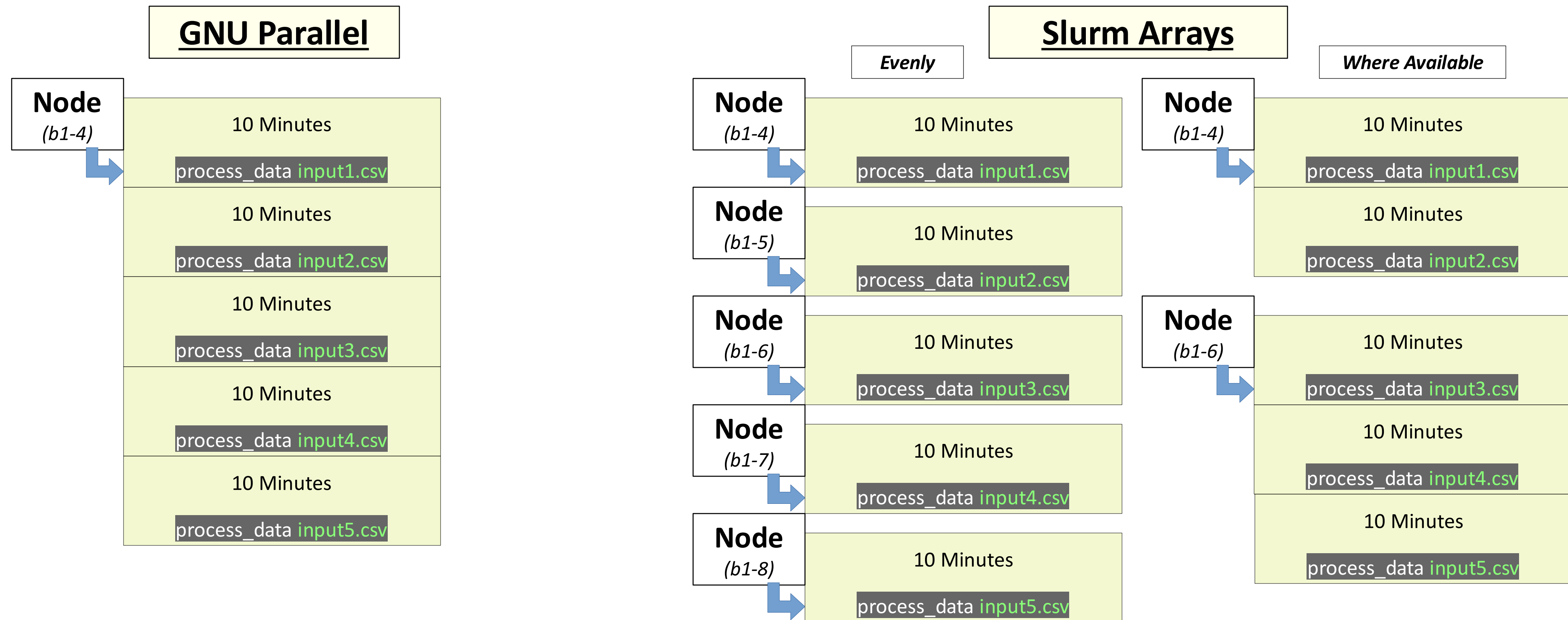
- Example
  - Use the `process_data` command with the files:
    - `input1.csv` `input2.csv` `input3.csv` `input4.csv` `input5.csv`

Spread commands across multiple nodes on Sapelo2 wherever available.

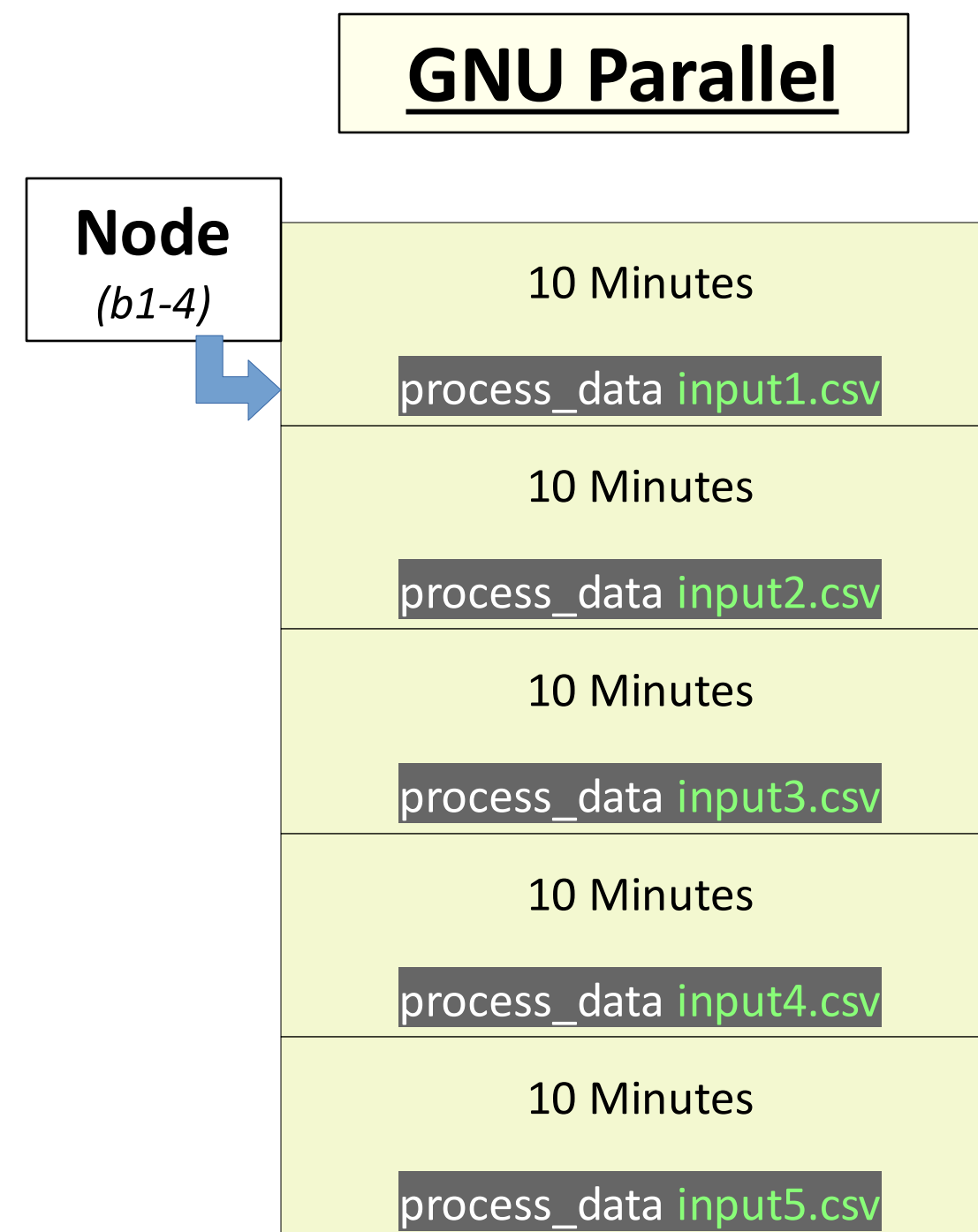




- Example
- Use the `process_data` command with the files:
  - `input1.csv` `input2.csv` `input3.csv` `input4.csv` `input5.csv`



# GNU Parallel

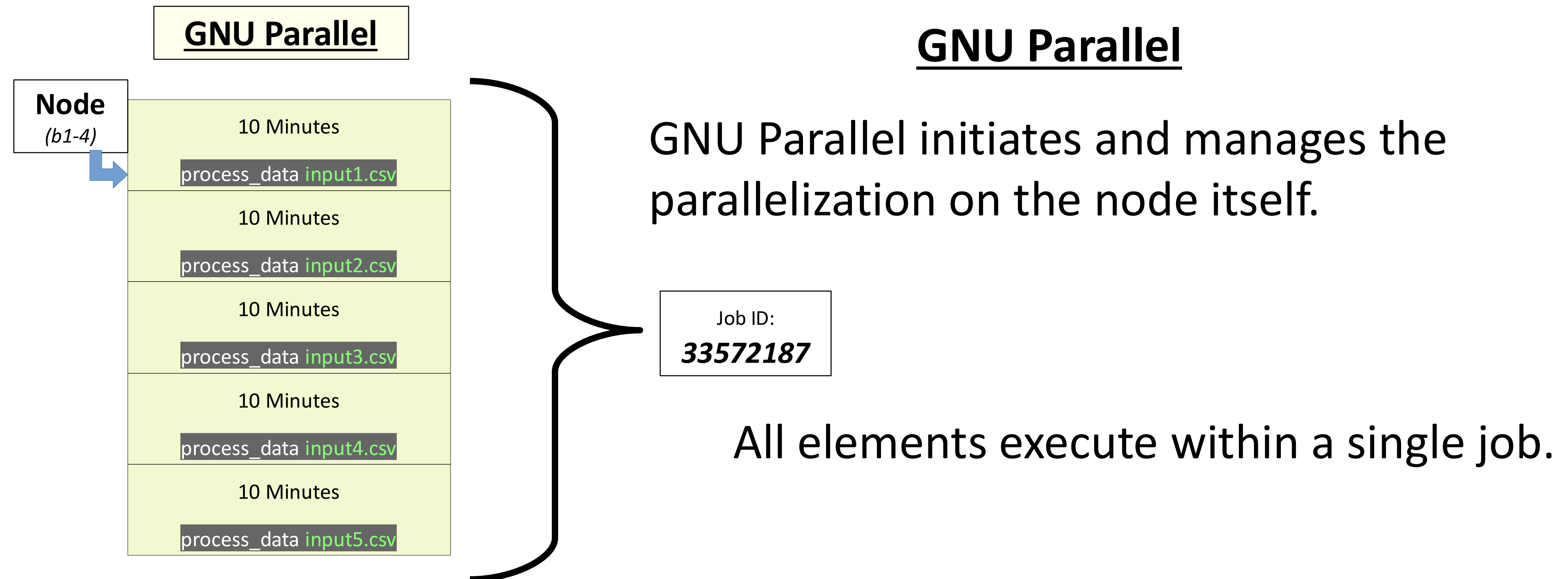


## GNU Parallel

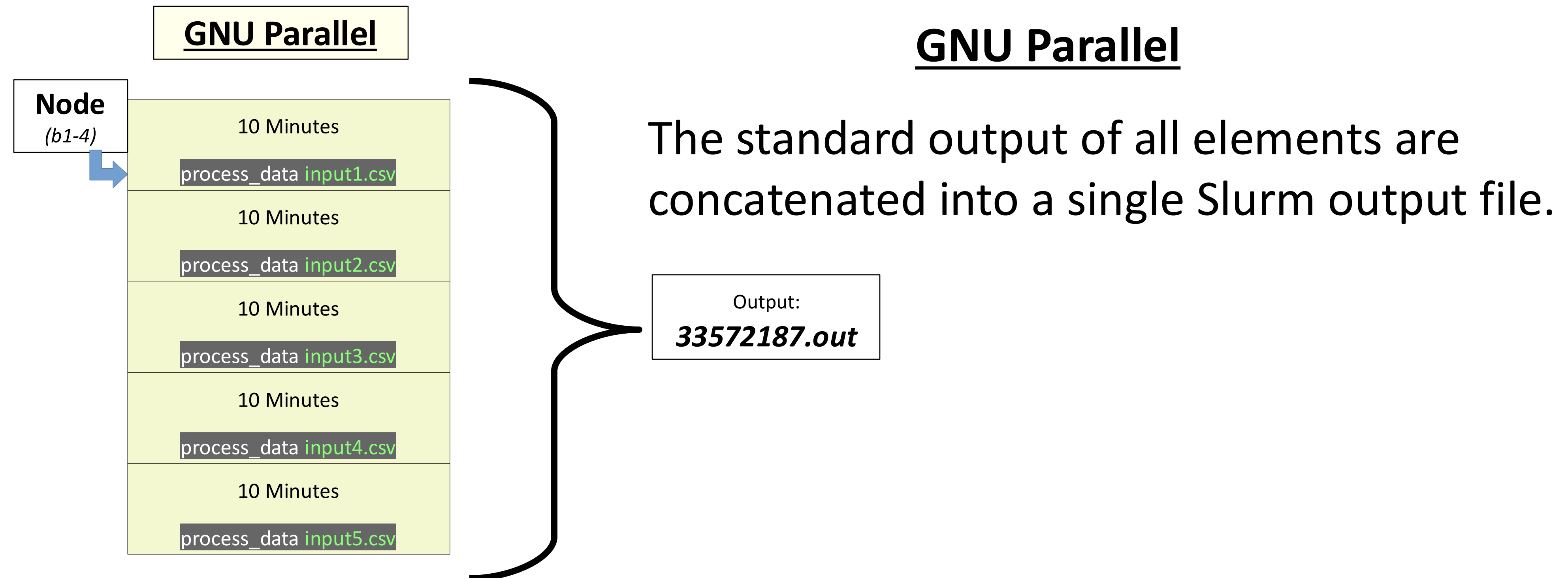
*GNU Parallel is a general parallelizer to run multiple serial command line programs in parallel without changing them.*



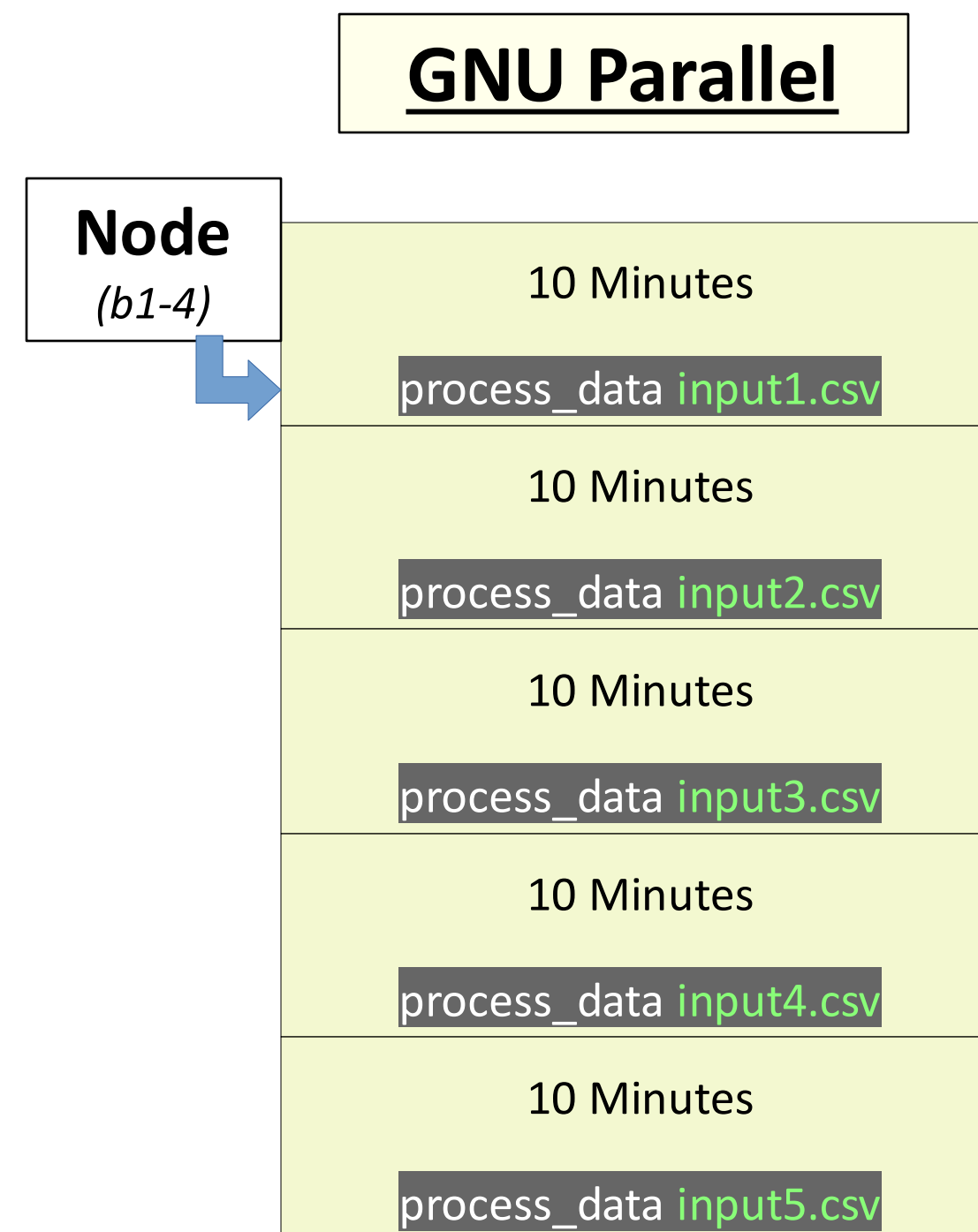
# GNU Parallel



# GNU Parallel



# GNU Parallel



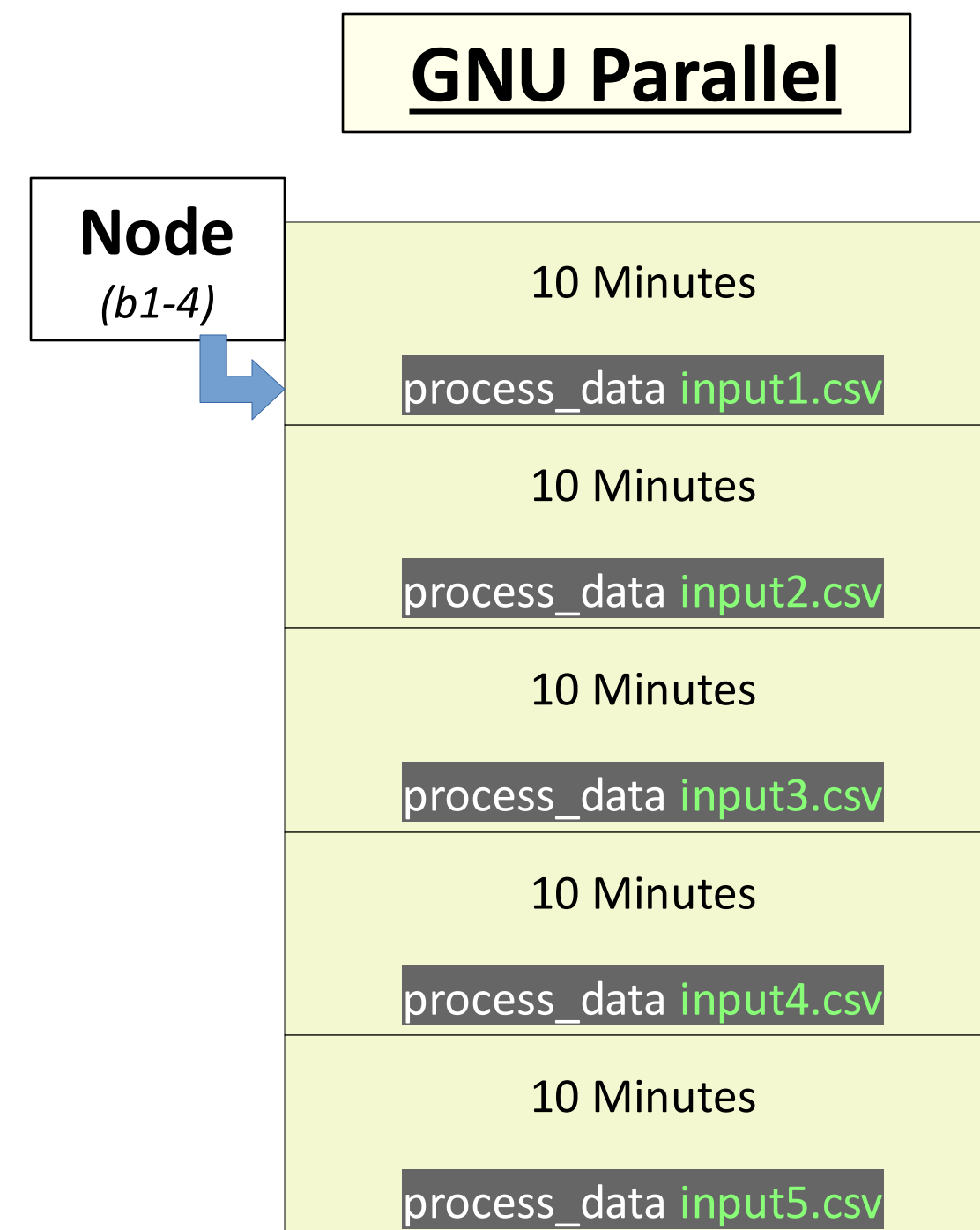
Parallelize a series of commands by prepending the command with *parallel*.

Foundational parameters

- Three colons `...` to pass text as inputs.
- Four colons `....` to pass files as inputs.



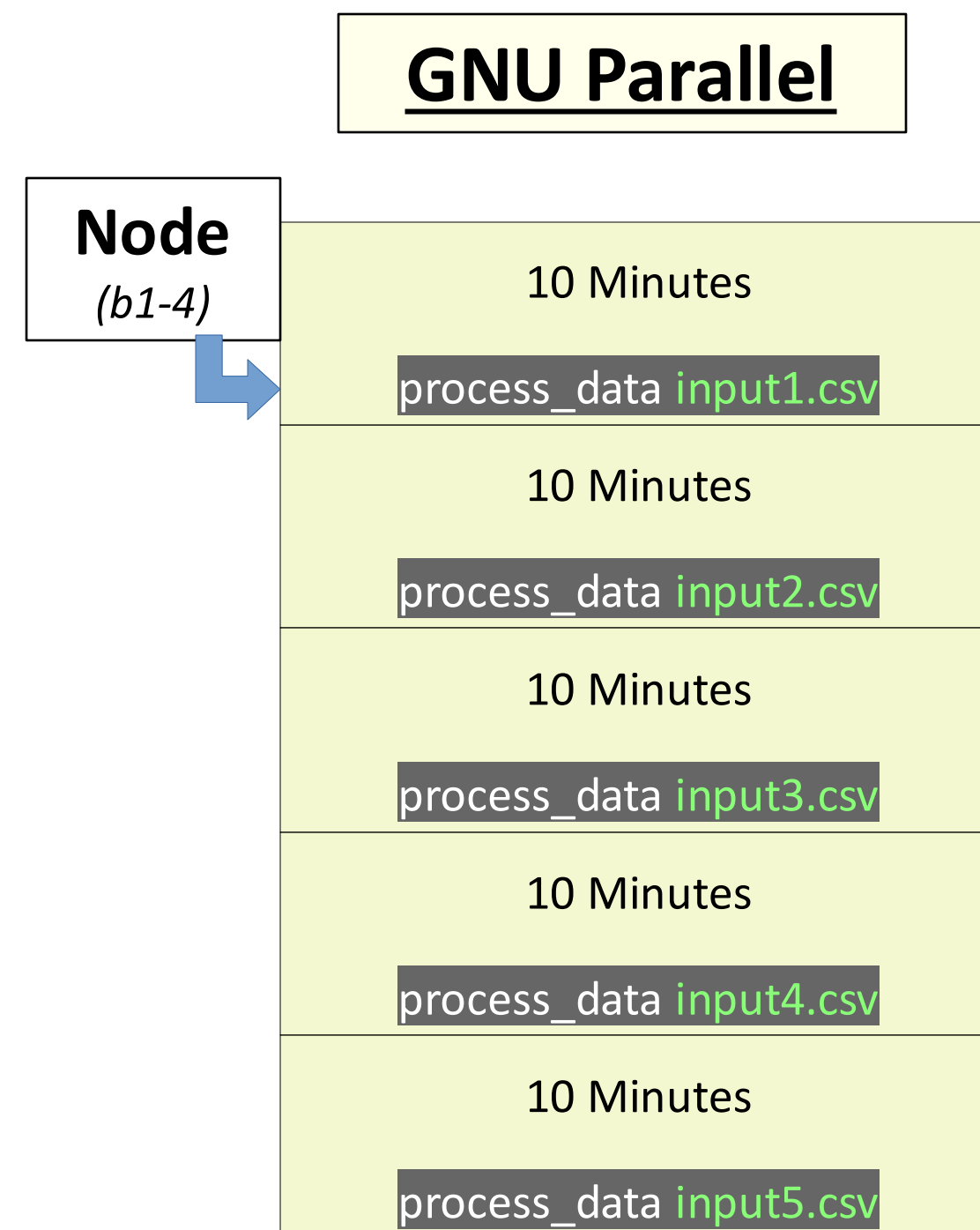
# GNU Parallel



Use three colons `:::` and then specify each input by name.

```
parallel process_data ::: input1.csv input2.csv input3.csv input4.csv input5.csv
```

# GNU Parallel



1) Create a file containing the names of all the inputs called *inputs.txt*.

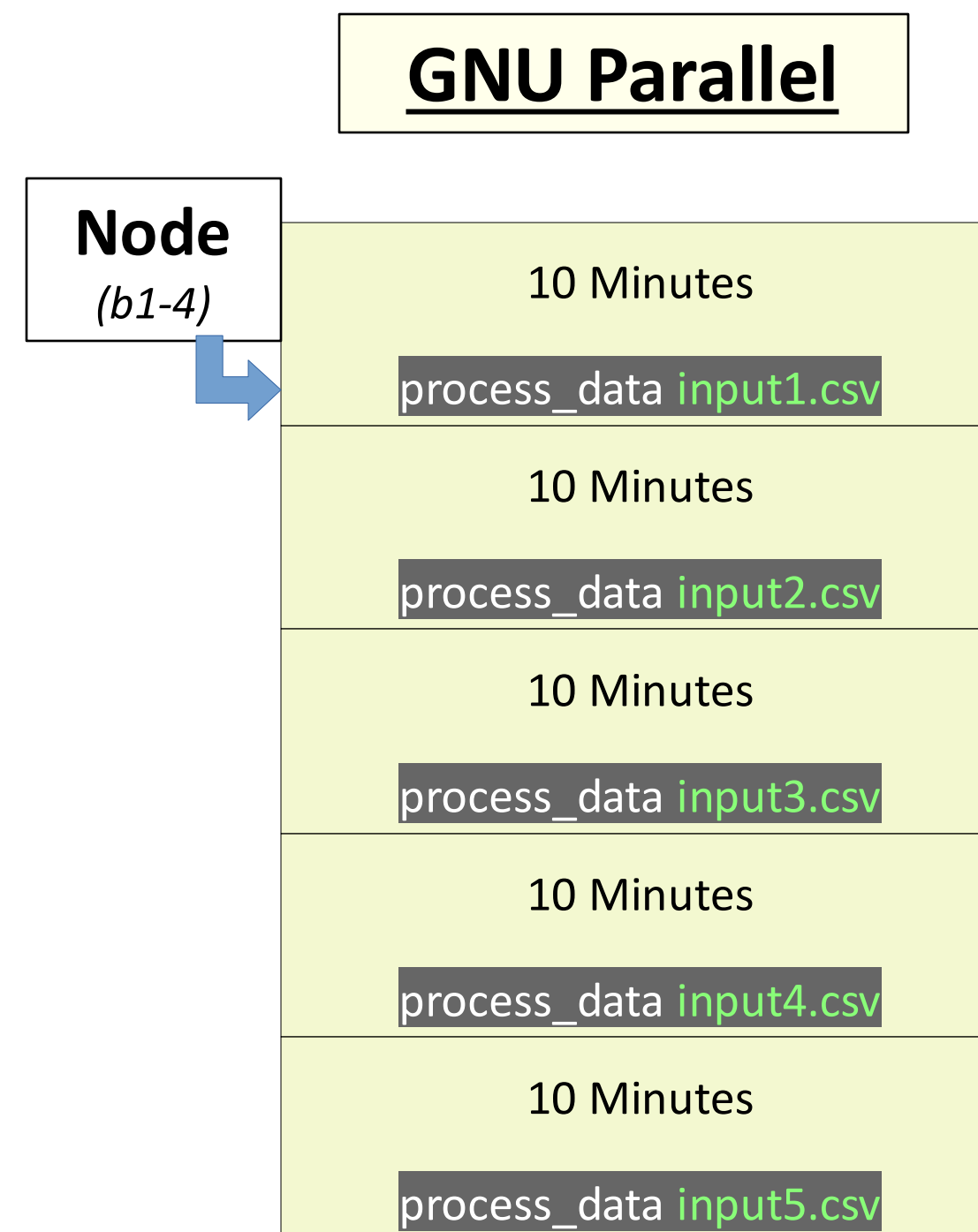
2) Specify the file name following four colons *:::*.

```
parallel process_data ::: inputs.txt
```

**inputs.txt**

```
input1.csv
input2.csv
input3.csv
input4.csv
input5.csv
```

# GNU Parallel



**Load the parallel module.**

```
#!/bin/bash
#SBATCH --partition=batch
#SBATCH --job-name=test
#SBATCH --nodes=1
#SBATCH --time=00:15:00
#SBATCH --mem=5G
#SBATCH --ntasks=5
```

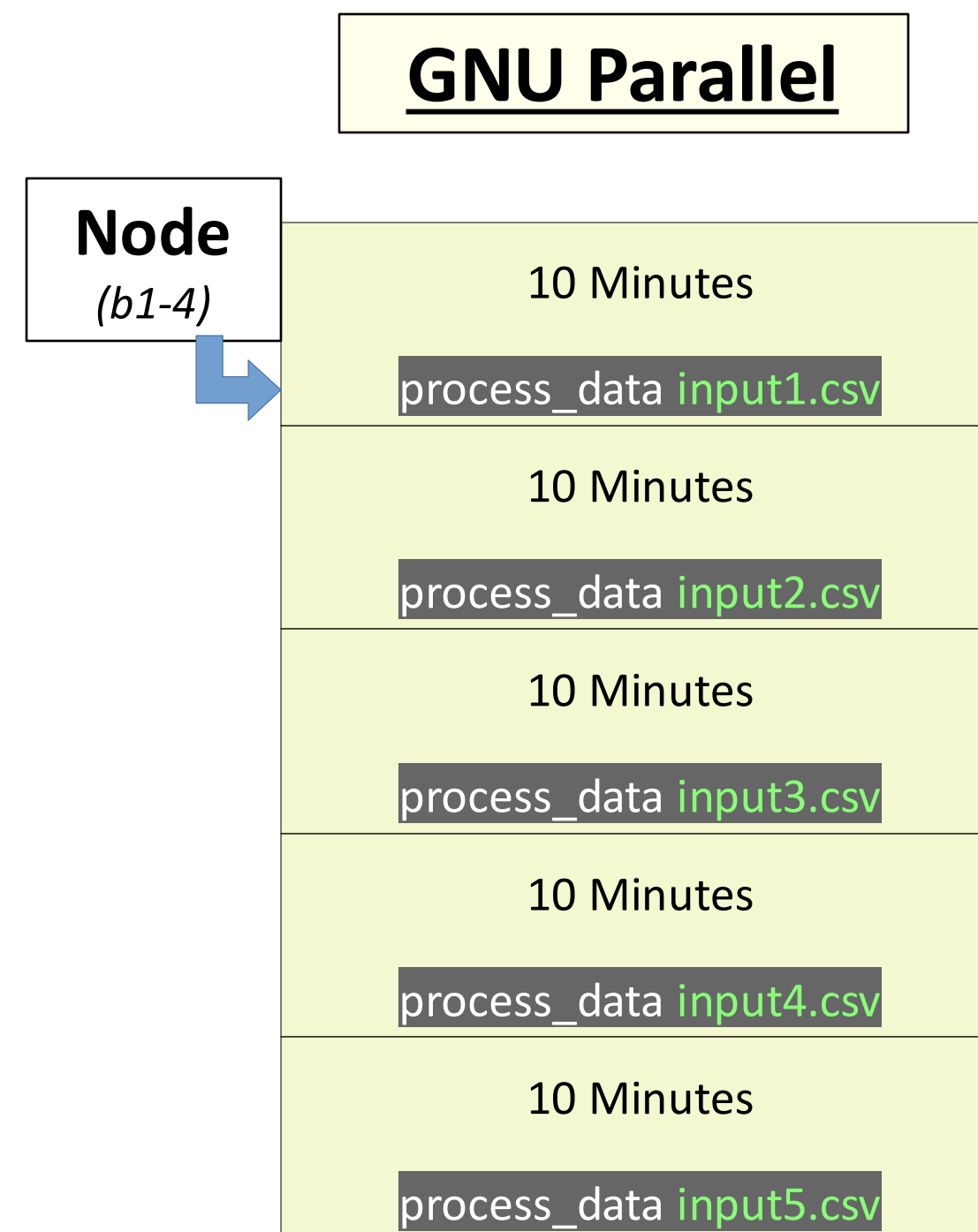
**Request adequate memory and  
tasks in the Slurm header.**

```
module load parallel/20230722-GCCcore-12.3.0
```

```
parallel process_data ::: inputs.txt
```



# GNU Parallel



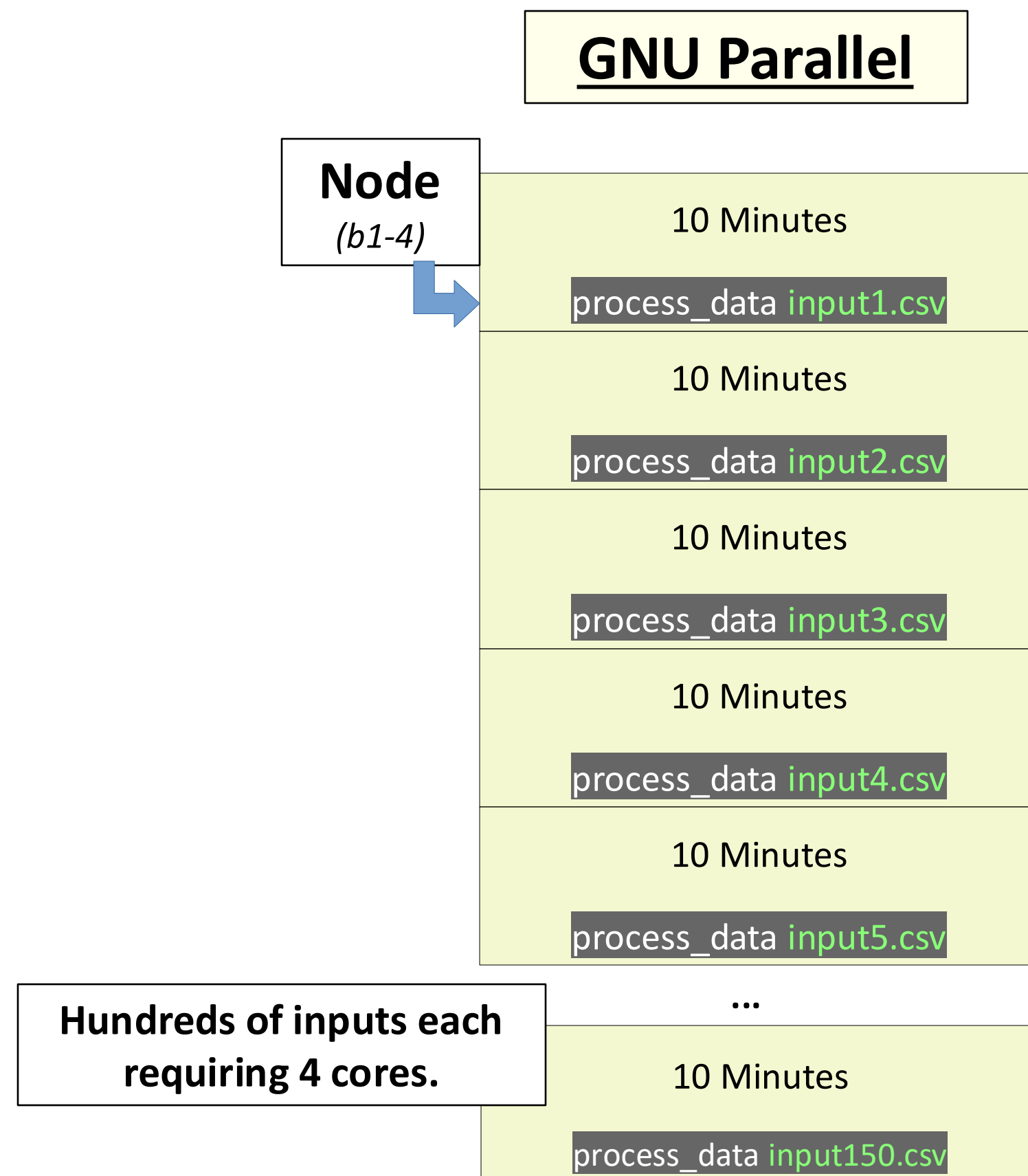
```
#!/bin/bash
#SBATCH --partition=batch
#SBATCH --job-name=test
#SBATCH --nodes=1
#SBATCH --time=00:15:00
#SBATCH --mem=5G
#SBATCH --ntasks=5
#SBATCH --cpus-per-task=4
```

```
module load parallel/20230722-GCCcore-12.3.0
```

```
parallel process_data ::: inputs.txt
```

**Multiple CPU cores can be  
requested for each job element.**

# GNU Parallel



```
#!/bin/bash
#SBATCH --partition=batch
#SBATCH --job-name=test
#SBATCH --nodes=1
#SBATCH --time=00:15:00
#SBATCH --mem=150G
#SBATCH --ntasks=150
#SBATCH --cpus-per-task=4
```

```
module load parallel/20230722-GCCcore-12.3.0
```

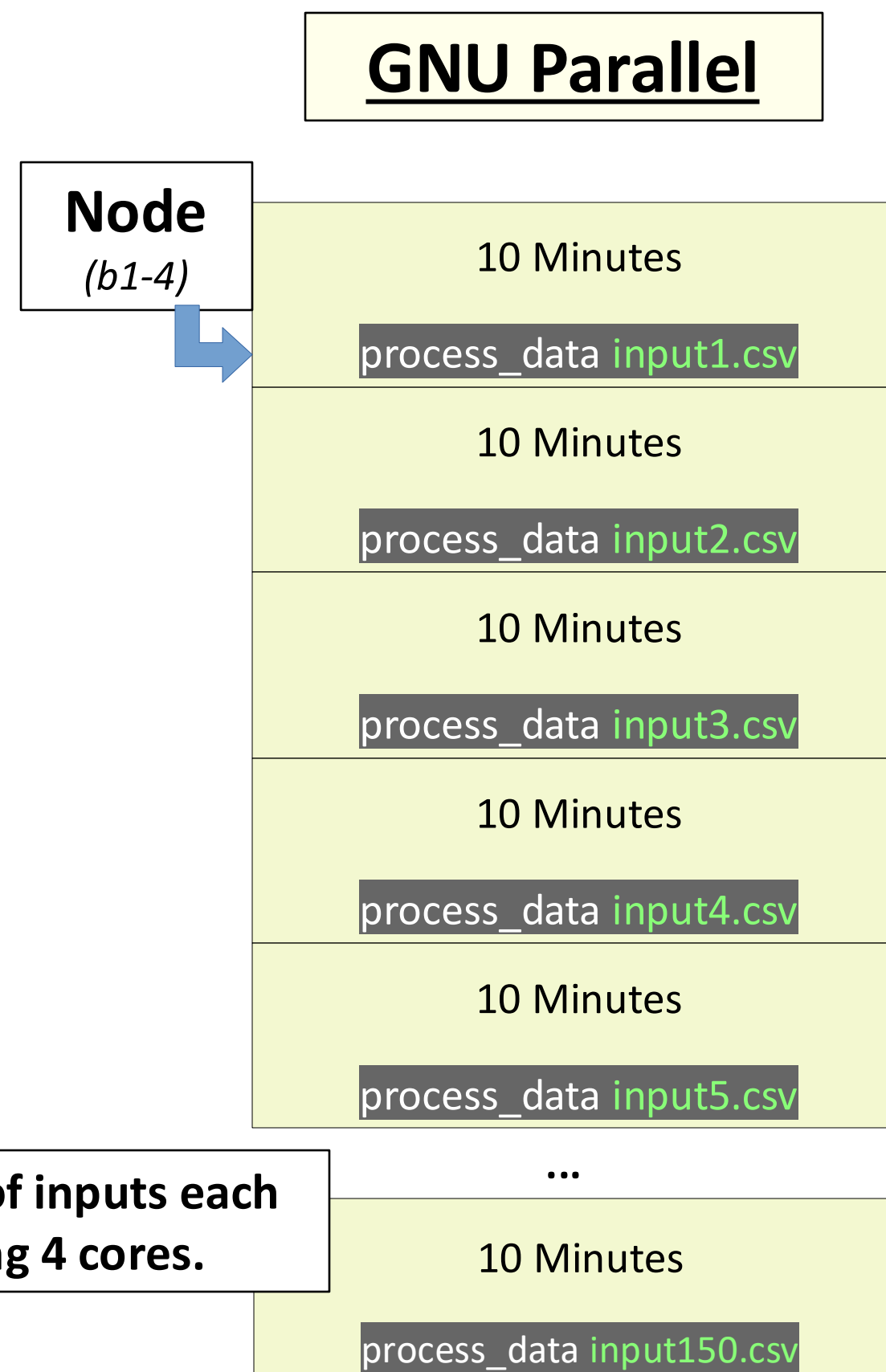
```
parallel process_data ::: inputs.txt
```

A single node cannot provide  
this many cores.

$150 * 4 = 600$  cores



# GNU Parallel



Node specifications can be found on the GACRC wiki:

[https://wiki.gacrc.uga.edu/wiki/Job\\_Submission\\_partitions\\_on\\_Sapelo2](https://wiki.gacrc.uga.edu/wiki/Job_Submission_partitions_on_Sapelo2)

```
#!/bin/bash
#SBATCH --partition=batch
#SBATCH --job-name=test
#SBATCH --nodes=1
#SBATCH --time=00:15:00
#SBATCH --mem=16G
#SBATCH --ntasks=16
#SBATCH --cpus-per-task=4
```

All resources must be provided by a single node; ensure requested resources can fit on only one node.

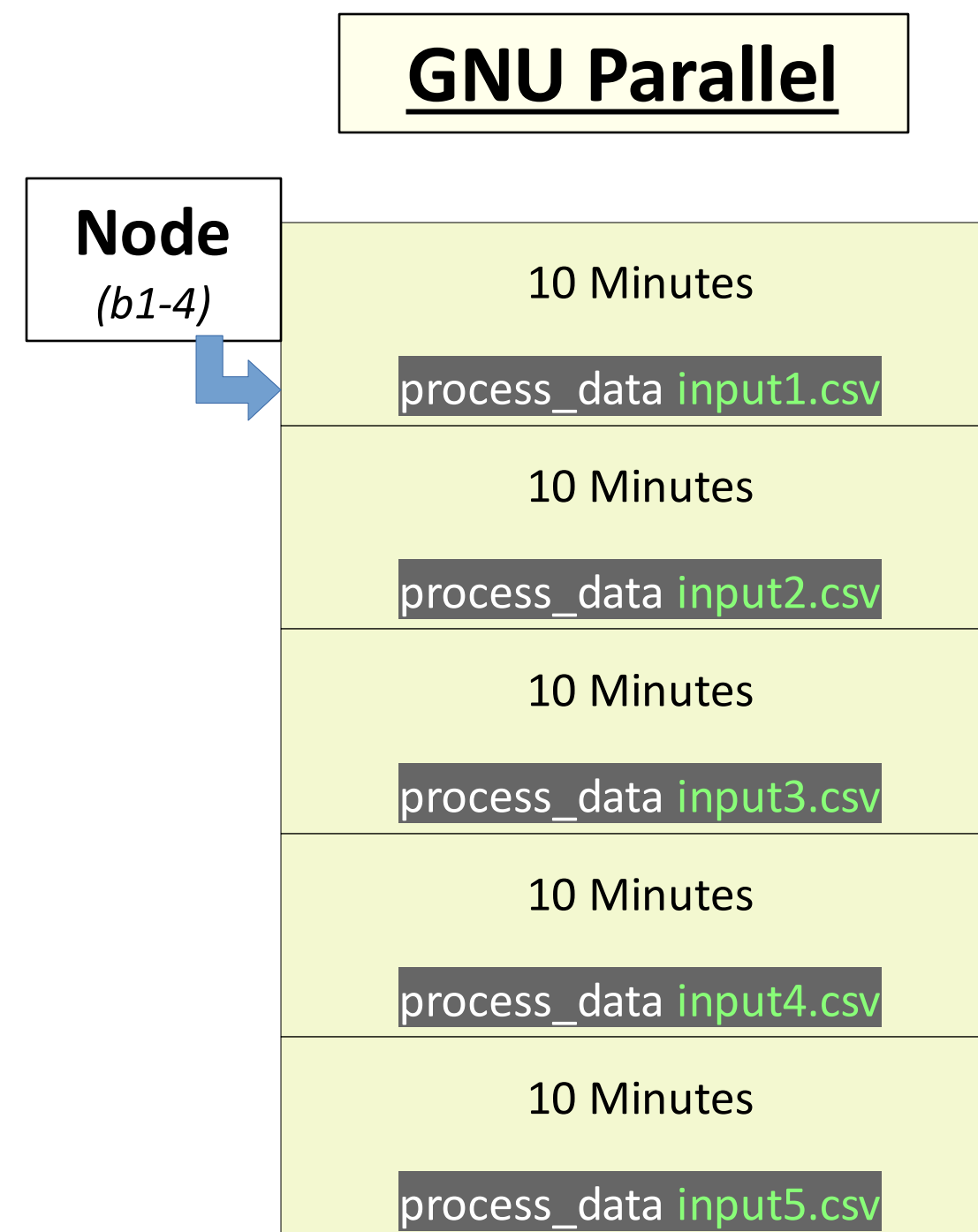
$16 * 4 = 64$  cores

module load parallel/20230722-GCCcore-12.3.0

parallel -j 16 process\_data ::: inputs.txt

Limit the number of concurrent jobs.

# GNU Parallel



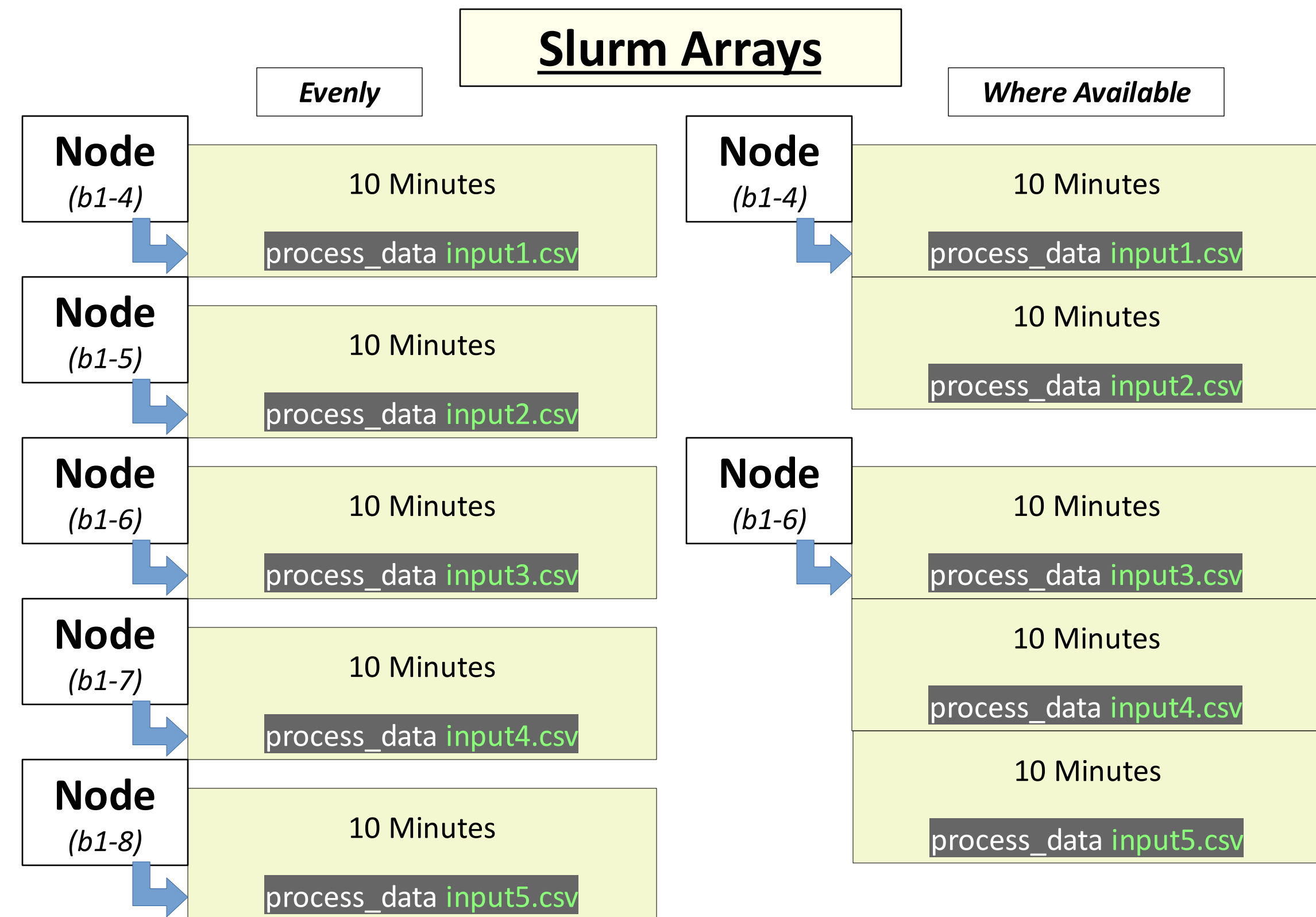
If using GNU Parallel to develop and publish a program, use the `--citation` flag to generate a citation for the project.

```
parallel --citation
```

# Slurm Job Arrays

## Slurm Arrays

*Built-in Slurm functionality that provides a mechanism for submitting and managing collections of similar jobs quickly and easily.*

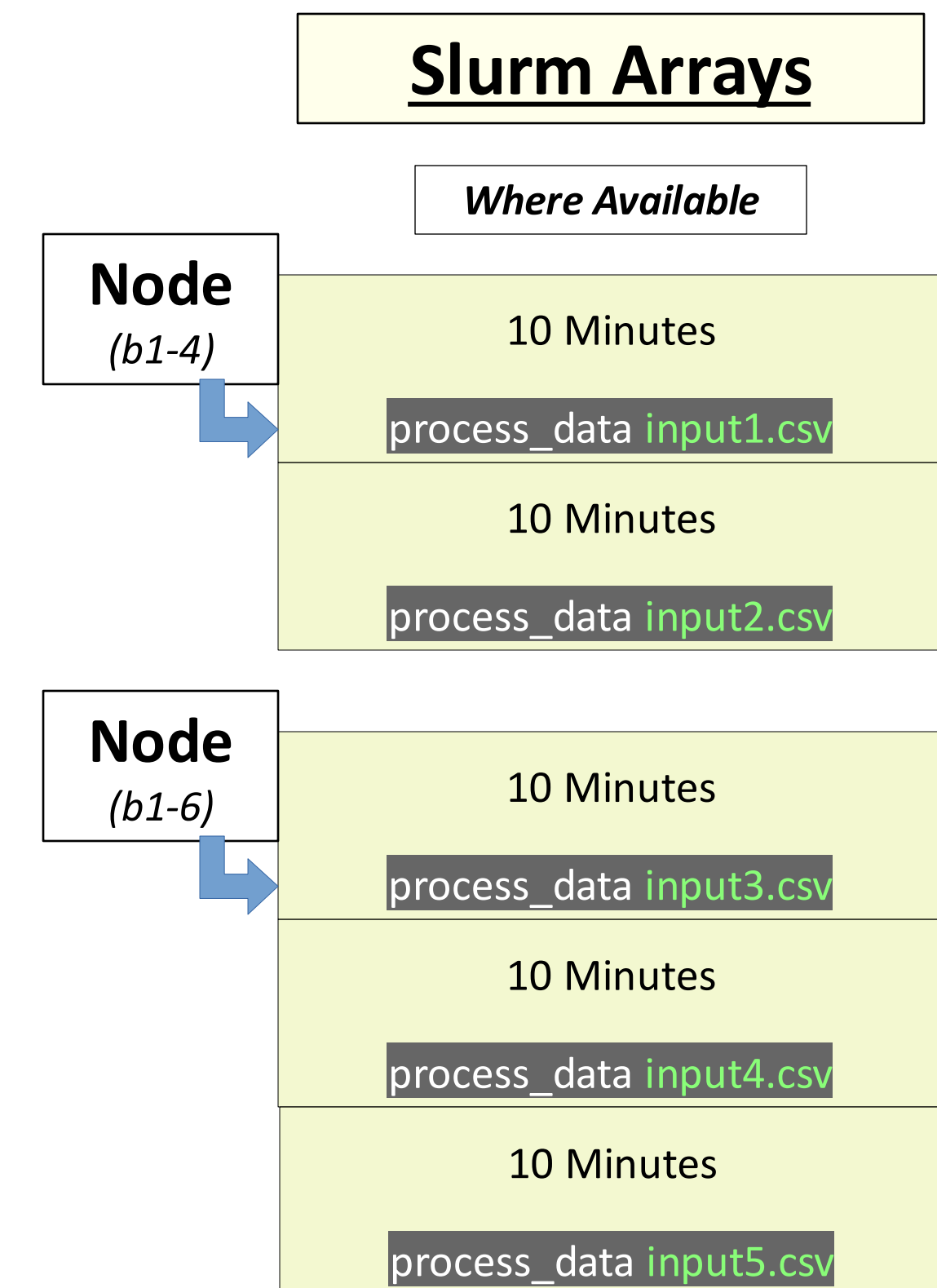




# Slurm Job Arrays

## Slurm Arrays

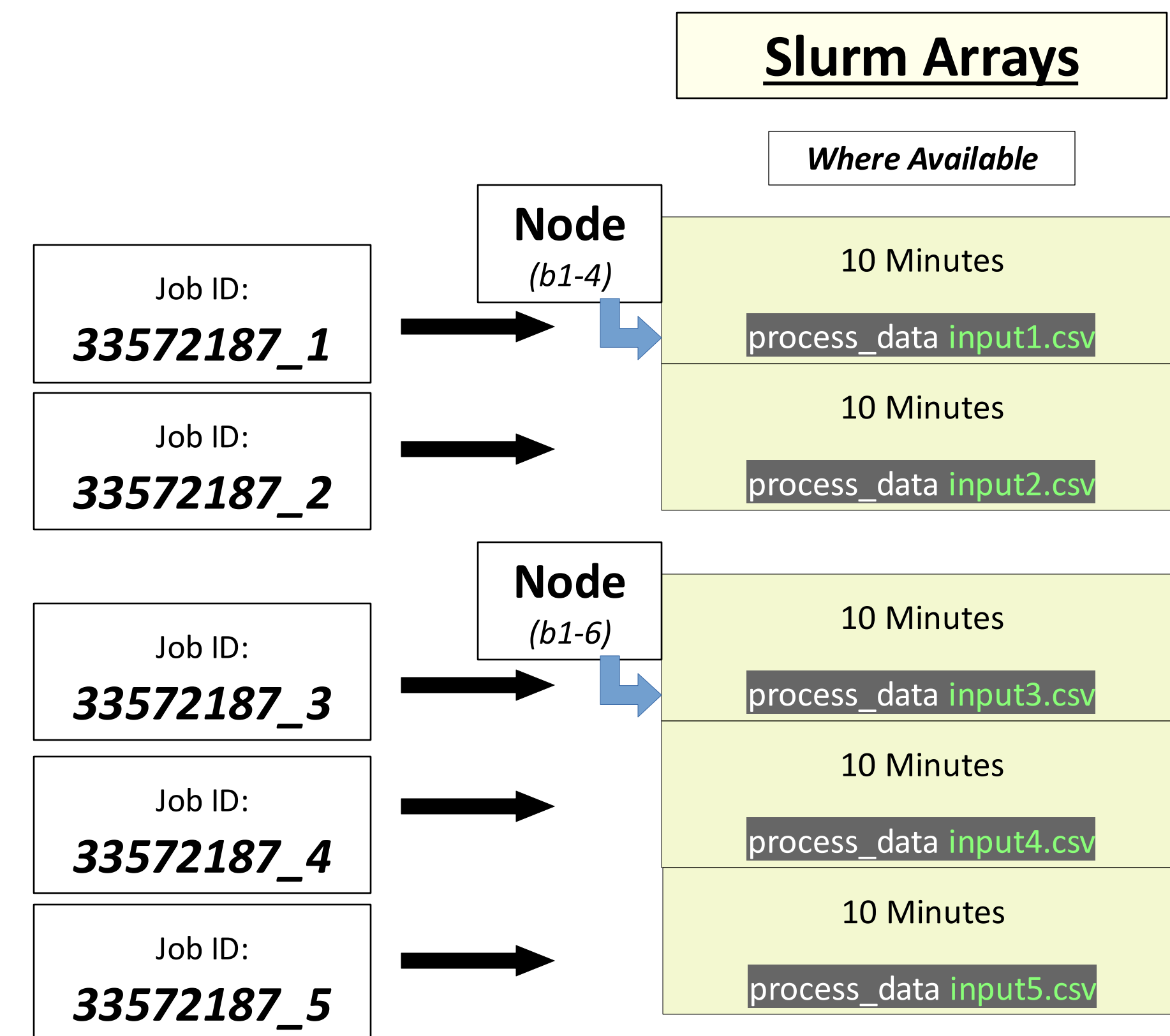
By default, Slurm will schedule array job elements wherever available.



# Slurm Job Arrays

## Slurm Arrays

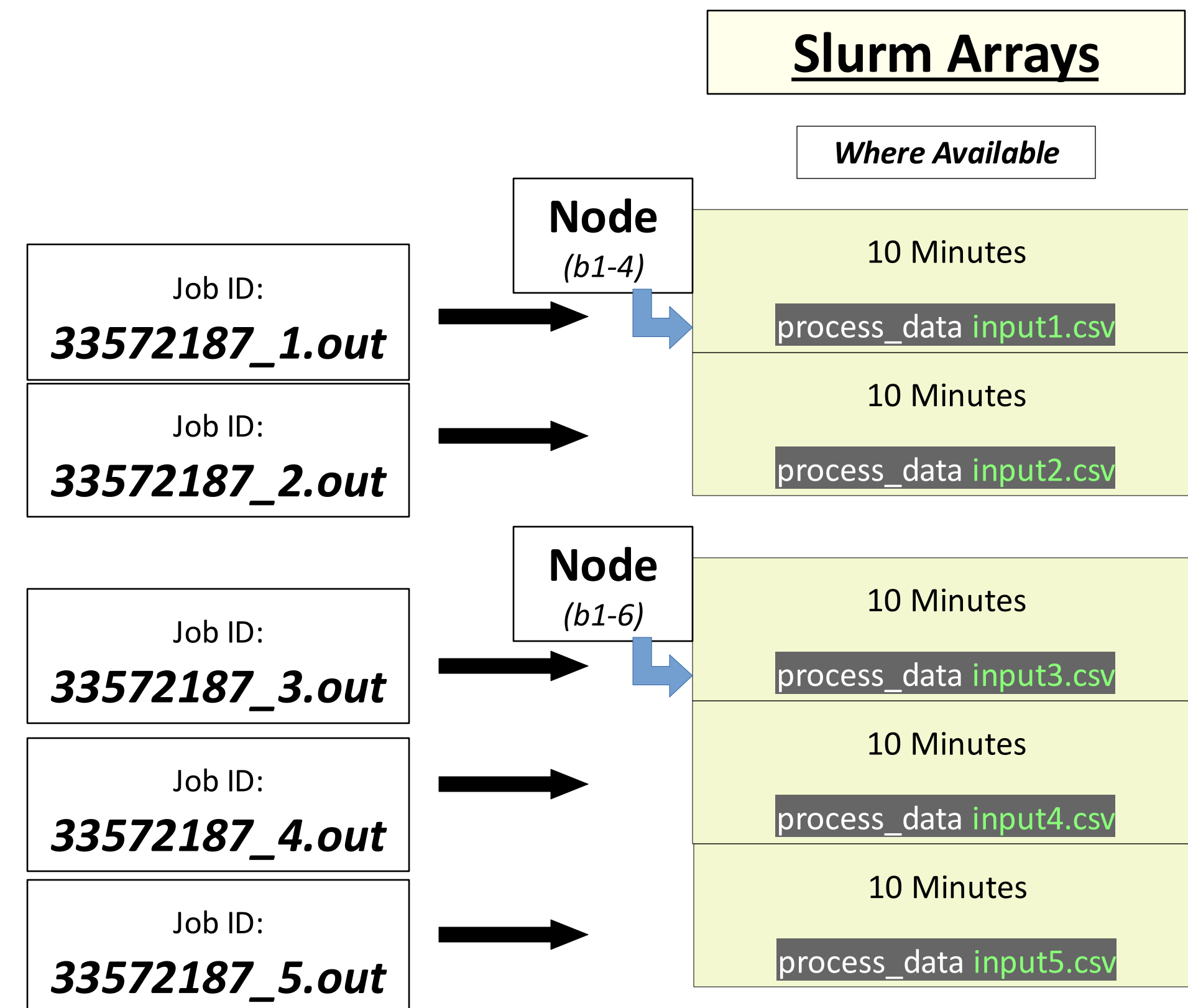
- The Slurm head node initiates and manages the parallelization.
- Each element in a Slurm array has its own Job ID.



# Slurm Job Arrays

## Slurm Arrays

Each job element will produce its own Slurm output file.





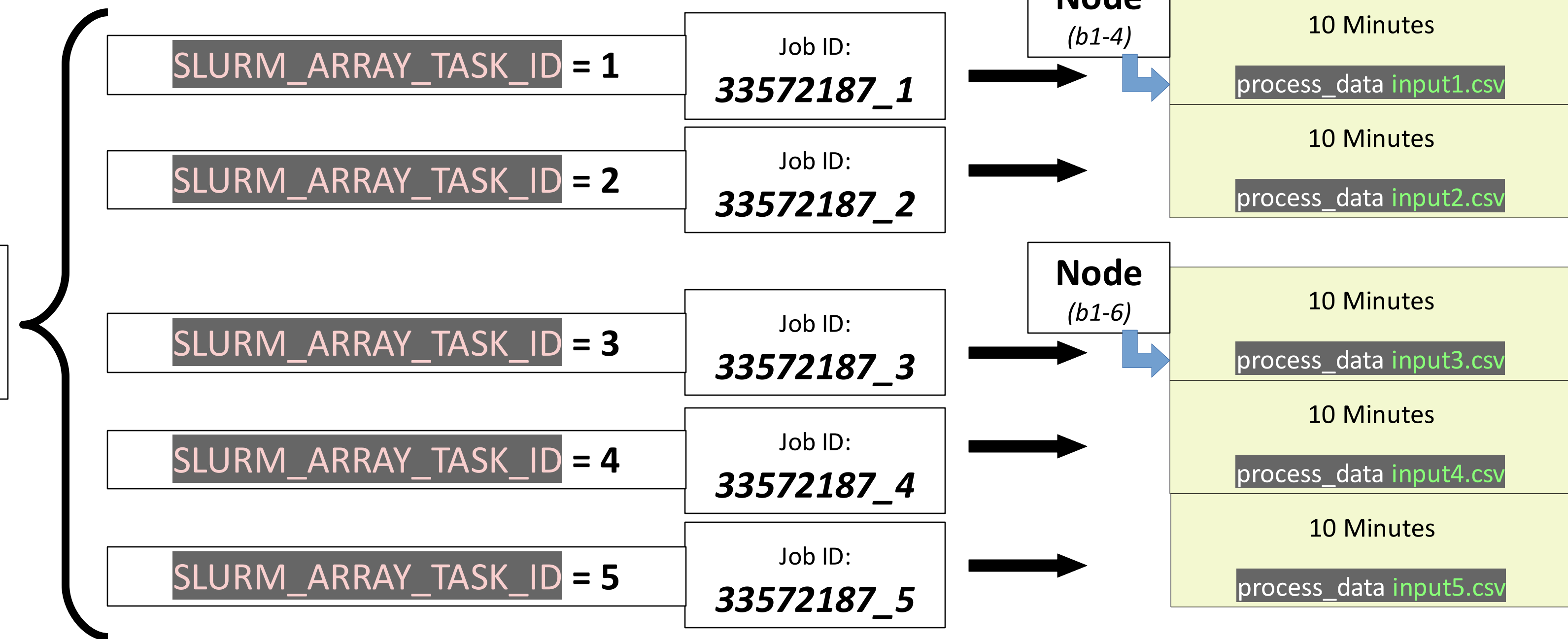
# Slurm Job Arrays

Initiated with the Slurm header:

`#SBATCH --array=START-END`

Each job element in the array will  
have its own

`SLURM_ARRAY_TASK_ID`





# Slurm Job Arrays

```
#!/bin/bash
#SBATCH --partition=batch
#SBATCH --job-name=test
#SBATCH --ntasks=1
#SBATCH --time=00:15:00
#SBATCH --mem=1G
#SBATCH --array=1-5
```

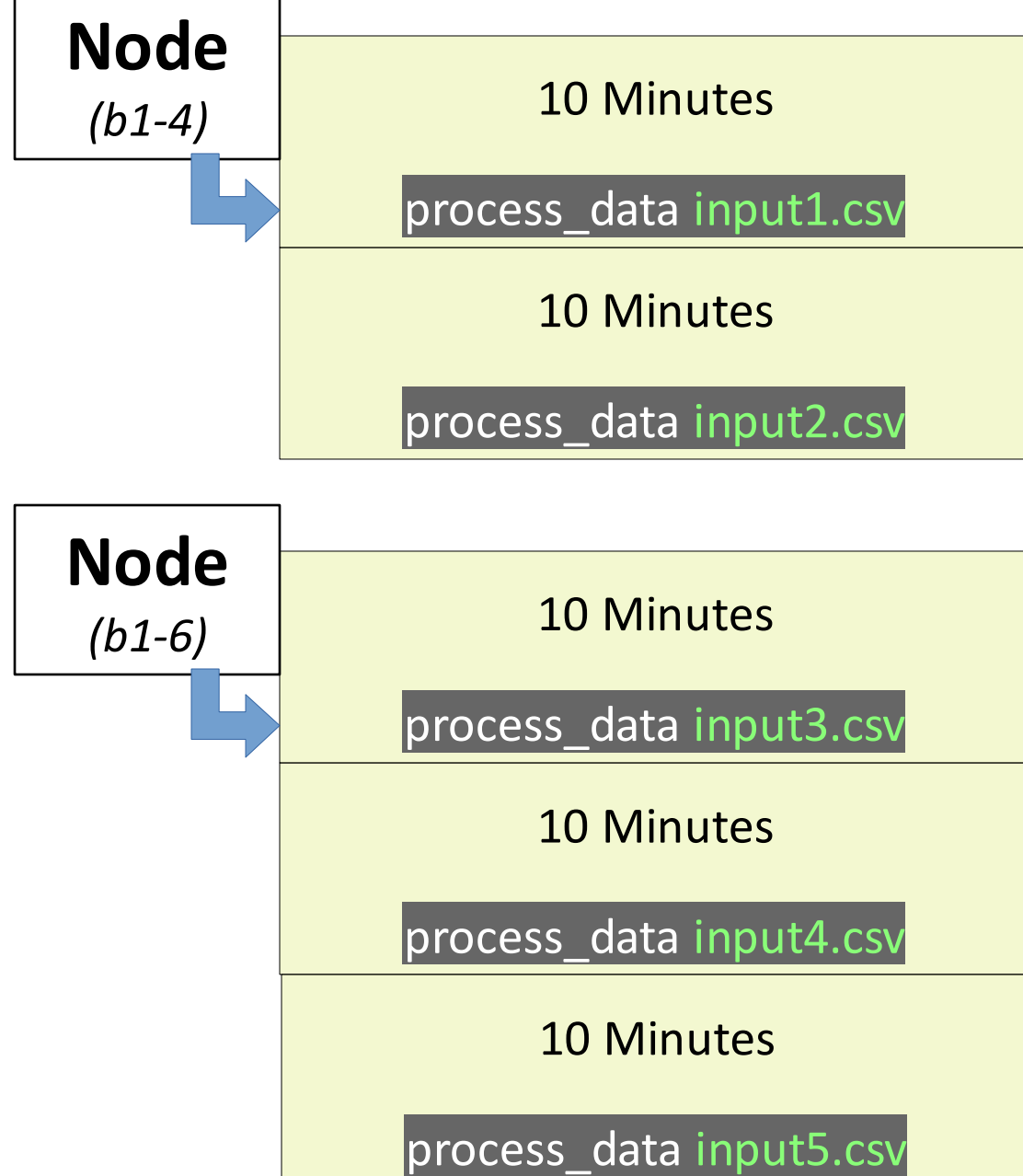
Specify a job array in the Slurm header.

```
process_data input${SLURM_ARRAY_TASK_ID}.csv
```

Incorporate the `SLURM_ARRAY_TASK_ID` variable.

## Slurm Arrays

Where Available





# Slurm Job Arrays

```
#!/bin/bash
#SBATCH --partition=batch
#SBATCH --job-name=test
#SBATCH --ntasks=1
#SBATCH --time=00:15:00
#SBATCH --mem=1G
#SBATCH --cpus-per-task=4
#SBATCH --array=1-5
```

```
process_data input${SLURM_ARRAY_TASK_ID}.csv
```

Multiple CPU cores can be requested  
for each job element.

## Slurm Arrays

Where Available

Node  
(b1-4)

10 Minutes

process\_data input1.csv

10 Minutes

process\_data input2.csv

Node  
(b1-6)

10 Minutes

process\_data input3.csv

10 Minutes

process\_data input4.csv

10 Minutes

process\_data input5.csv

# Slurm Job Arrays

```
#!/bin/bash
#SBATCH --partition=batch
#SBATCH --job-name=test
#SBATCH --ntasks=1
#SBATCH --time=00:15:00
#SBATCH --mem=1G
#SBATCH --cpus-per-task=4
#SBATCH --array=1-5

process_data $(awk "NR==${SLURM_ARRAY_TASK_ID}" inputs.txt)
```

An input file containing the names of all the inputs  
can be used via `awk`.

## Slurm Arrays

Where Available

Node  
(b1-4)

10 Minutes

process\_data input1.csv

10 Minutes

process\_data input2.csv

Node  
(b1-6)

10 Minutes

process\_data input3.csv

10 Minutes

process\_data input4.csv

10 Minutes

process\_data input5.csv

# Slurm Arrays and GNU Parallel

- Slurm Arrays

- Can be configured for tasks to execute across multiple nodes.
- Parallelization handled through Slurm.
- May only be used in non-interactive jobs.

- GNU Parallel

- All tasks execute on a single node.
- Parallelization handled by the GNU Parallel software on the node.
- Can be executed interactively.



# Practice

./scramble.py

```
#!/usr/bin/env python
```

```
import sys
import time
from pathlib import Path
from random import shuffle
```

```
def shuffle_word(word: str) -> str:
    word = list(word)
    shuffle(word)
    return ''.join(word)
```

Function to shuffle characters

```
def main() -> None:
    shuffled = list()
    with open(sys.argv[1]) as file:
        for line in file.readlines():
            shuffled.append(shuffle_word(line.strip('\n')))
```

Read input file and shuffle

```
output_dir = sys.argv[2].split('/')[-1]
```

Create output directory

```
Path(output_dir).mkdir(parents=True, exist_ok=True)
```

```
filename = sys.argv[1].split('/')[-1]
filename = filename.split('.')[0]
```

Write shuffled output

```
with open(f'{output_dir}/{filename}.out', 'w') as file:
    for word in shuffled:
        file.write(word + '\n')
```

```
time.sleep(5)
print(f'"{filename}" shuffled and saved to "{output_dir}/{filename}.out".')
```

```
if __name__ == "__main__":
    main()
```

Sorted/input1.txt

Orange  
Garden  
Planet



output\_dir/input1.out

nraOeg  
dnearG  
eaPltn

# Practice

./scramble.py

```
#!/usr/bin/env python

import sys
import time
from pathlib import Path
from random import shuffle

def shuffle_word(word: str) -> str:
    word = list(word)
    shuffle(word)
    return ''.join(word)

def main() -> None:
    shuffled = list()
    with open(sys.argv[1]) as file:
        for line in file.readlines():
            shuffled.append(shuffle_word(line.strip('\n')))

    output_dir = sys.argv[2].split('/')[1]

    Path(output_dir).mkdir(parents=True, exist_ok=True)

    filename = sys.argv[1].split('/')[1]
    filename = filename.split('.')[0]

    with open(f'{output_dir}/{filename}.out', 'w') as file:
        for word in shuffled:
            file.write(word + '\n')

    time.sleep(5)
    print(f'"{filename}" shuffled and saved to "{output_dir}/{filename}.out".')

if __name__ == "__main__":
    main()
```

Function to shuffle characters

Read input file and shuffle

Create output directory

Write shuffled output

Run script with 10 different input files

|                                             |
|---------------------------------------------|
| ./scramble.py Sorted/input1.txt output_dir  |
| ./scramble.py Sorted/input2.txt output_dir  |
| ./scramble.py Sorted/input3.txt output_dir  |
| ./scramble.py Sorted/input4.txt output_dir  |
| ./scramble.py Sorted/input5.txt output_dir  |
| ./scramble.py Sorted/input6.txt output_dir  |
| ./scramble.py Sorted/input7.txt output_dir  |
| ./scramble.py Sorted/input8.txt output_dir  |
| ./scramble.py Sorted/input9.txt output_dir  |
| ./scramble.py Sorted/input10.txt output_dir |

# Practice

```
[myid@ss-sub2 ~]$ cd /scratch/$USER
```

```
[myid@ss-sub2 myid]$ pwd
```

```
/scratch/myid
```

```
[myid@ss-sub2 myid]$ cp -r /usr/local/training/parallel/ ./
```

**cd to your user's scratch directory**

**Copy practice files**



# Practice

```
[myid@ss-sub2 myid]$ tree parallel
```

```
parallel
```

```
├── Mixed
```

```
│   ├── 2BJn5Oak.txt  
│   ├── 46XCF1H7.txt  
│   ├── TyXa4F8v.txt  
│   ├── Us6yfHmH.txt  
│   ├── fm5la6TZ.txt  
│   ├── luKJRDzy.txt  
│   ├── q6U7KWE9.txt  
│   ├── sTIYZOsG.txt  
│   └── sc8UMm0F.txt
```

```
└── yUvcLpIn.txt
```

```
├── Sorted
```

```
│   ├── input1.txt  
│   ├── input10.txt  
│   ├── input2.txt  
│   ├── input3.txt  
│   ├── input4.txt  
│   ├── input5.txt  
│   ├── input6.txt  
│   ├── input7.txt  
│   └── input8.txt
```

```
└── input9.txt
```

```
├── full_input.txt
```

```
├── scramble.py
```

```
└── sub.sh
```

```
2 directories, 22 files
```

Input files with numbered names

scramble.py script

Sequential job script

# Practice

```
[myid@ss-sub2 myid]$ cd parallel  
[myid@ss-sub2 parallel]$ ls  
Mixed Sorted full_input.txt scramble.py sub.sh
```

**cd to parallel directory**



# Practice

```
[myid@ss-sub2 parallel]$ cat sub.sh
```

```
#!/usr/bin/env bash
```

```
#SBATCH --job-name=Sequential
```

```
#SBATCH --ntasks=1
```

```
#SBATCH --partition=batch
```

```
#SBATCH --mem=1G
```

```
#SBATCH --time=00:10:00
```

```
#SBATCH --output=standard_outputs/%x_%j.out
```

```
cd $SLURM_SUBMIT_DIR
```

```
module load Python/3.11.3-GCCcore-12.3.0
```

```
./scramble.py Sorted/input1.txt ./sequential_outputs
```

```
./scramble.py Sorted/input2.txt ./sequential_outputs
```

```
./scramble.py Sorted/input3.txt ./sequential_outputs
```

```
./scramble.py Sorted/input4.txt ./sequential_outputs
```

```
./scramble.py Sorted/input5.txt ./sequential_outputs
```

```
./scramble.py Sorted/input6.txt ./sequential_outputs
```

```
./scramble.py Sorted/input7.txt ./sequential_outputs
```

```
./scramble.py Sorted/input8.txt ./sequential_outputs
```

```
./scramble.py Sorted/input9.txt ./sequential_outputs
```

```
./scramble.py Sorted/input10.txt ./sequential_outputs
```

**Review provided job submission script**

**Current approach executes sequentially**

# Practice

```
[myid@ss-sub2 parallel]$ ls Sorted/* > inputs.txt  
[myid@ss-sub2 parallel]$ ls  
Mixed Sorted full_input.txt inputs.txt scramble.py sub.sh
```

**Create list of input files**

# GNU Parallel Job

inputs.txt

```
Sorted/input1.txt  
Sorted/input2.txt  
Sorted/input3.txt  
Sorted/input4.txt  
Sorted/input5.txt  
Sorted/input6.txt  
Sorted/input7.txt  
Sorted/input8.txt  
Sorted/input9.txt  
Sorted/input10.txt
```

sub\_parallel.sh

```
#!/bin/bash  
#SBATCH --job-name=Parallel  
#SBATCH --nodes=1  
#SBATCH --ntasks=10  
#SBATCH --partition=batch  
#SBATCH --mem=10G  
#SBATCH --time=00:10:00  
#SBATCH --output=standard_outputs/%x_%j.out  
  
cd $SLURM_SUBMIT_DIR  
  
module load parallel/20230722-GCCcore-12.3.0  
module load Python/3.11.3-GCCcore-12.3.0  
  
parallel ./scramble.py {} ./parallel_outputs ::: inputs.txt
```



# Slurm Array Job

inputs.txt

```
Sorted/input1.txt  
Sorted/input2.txt  
Sorted/input3.txt  
Sorted/input4.txt  
Sorted/input5.txt  
Sorted/input6.txt  
Sorted/input7.txt  
Sorted/input8.txt  
Sorted/input9.txt  
Sorted/input10.txt
```

sub\_array.sh

```
#!/bin/bash  
#SBATCH --partition=batch  
#SBATCH --job-name=Array  
#SBATCH --ntasks=1  
#SBATCH --partition=batch  
#SBATCH --mem=1G  
#SBATCH --time=00:10:00  
#SBATCH --output=standard_outputs/%x_%j.out  
#SBATCH --array=1-10  
  
cd $SLURM_SUBMIT_DIR  
  
module load Python/3.11.3-GCCcore-12.3.0  
  
./scramble.py $(awk "NR==${SLURM_ARRAY_TASK_ID}" inputs.txt) ./array_outputs
```