PIPELINES WITH SLURM

Bio Code club end-of-semester event

BASH AND SLURM

Conventional script:

```
> bash scriptname.bash
```

script is executed

On HPC:

```
> sbatch scriptname.bash
```

1)resources requested, job queued

2)job allocated on compute node

3)script is executed

[Your job here]

Compute cluster

SCALING UP ON HPC

Benefits

- More time
- More memory
- Larger/more datasets

Requirements

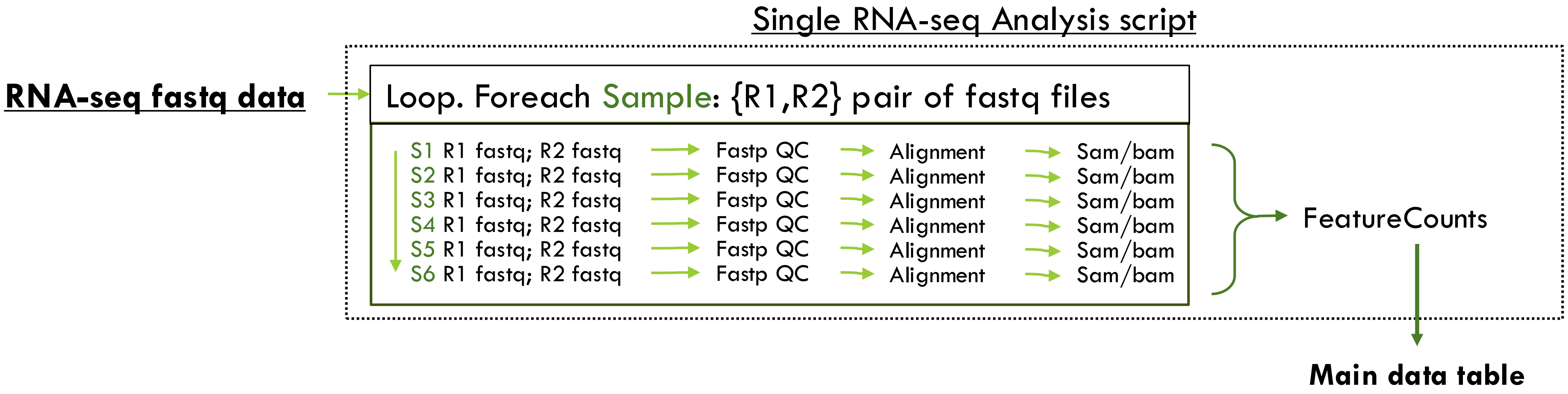
- Say you want to scale from 10 input files to 50 or 100
- But you have a complex pipelines (branching, multiple steps)
- Learning Workflow tools like Nextflow are a whole new system
- Basic → intermediate level scripting might be enough to do the coordination

This talk

- Introduce some features in SLURM to parallelize and coordinate scripts
- Talk about some debugging techniques
- Tools and Templates

Compute cluster

All data processed in single script, single job



Single RNA-seq Analysis script

RNA-seq fastq data

Loop. Foreach Sample: {R1,R2} pair of fastq files

S1 R1 fastq; R2 fastq → Fastp QC → Alignment → Sam/bam
S2 R1 fastq; R2 fastq → Fastp QC → Alignment → Sam/bam
S3 R1 fastq; R2 fastq → Fastp QC → Alignment → Sam/bam
S4 R1 fastq; R2 fastq → Fastp QC → Alignment → Sam/bam
S5 R1 fastq; R2 fastq → Fastp QC → Alignment → Sam/bam
S6 R1 fastq; R2 fastq → Fastp QC → Alignment → Sam/bam

FeatureCounts

Main data table

Pipeline synopsis

- Several steps per input file set S1…S6
- Loop iteration contains bulk of work, but …
- …not parallelized: doesn’t scale well to large/many files
- Final step FeatureCounts run after loop has completed

All data processed in single script, single job

Loop. Sequential execution

		20min		36min		16min	
S1 R1 fastq; R2 fastq	→	Fastp QC	→	Alignment	→	Sam/bam	72min
S2 R1 fastq; R2 fastq	→	Fastp QC	→	Alignment	→	Sam/bam	72min
S3 R1 fastq; R2 fastq	→	Fastp QC	→	Alignment	→	Sam/bam	72min
S4 R1 fastq; R2 fastq	→	Fastp QC	→	Alignment	→	Sam/bam	72min
S5 R1 fastq; R2 fastq	→	Fastp QC	→	Alignment	→	Sam/bam	72min
S6 R1 fastq; R2 fastq	→	Fastp QC	→	Alignment	→	Sam/bam	72min
							7 hours, 12 minutes

Loop. 2x cores

		10min		12min		8min	
S1	→	Fastp QC	→	Alignment	→	Sam/bam	36min
S2	→	Fastp QC	→	Alignment	→	Sam/bam	36min
S3	→	Fastp QC	→	Alignment	→	Sam/bam	36min
S4	→	Fastp QC	→	Alignment	→	Sam/bam	36min
S5	→	Fastp QC	→	Alignment	→	Sam/bam	36min
S6	→	Fastp QC	→	Alignment	→	Sam/bam	36min
							3 hours, 36 minutes

Cores:	1x	2x	4x*	8x*
Per sample time (minutes):	72	36	24	20
Total time (hours:minutes) for 6 iterations:	7:12	3:36	2:24	2:00

**diminishing return*

ALL DATA PROCESSED SEQUENTIALLY

Loop. Sequential execution

		20min		36min		16min	
S1 R1 fastq; R2 fastq	→	Fastp QC	→	Alignment	→	Sam/bam	72min
S2 R1 fastq; R2 fastq	→	Fastp QC	→	Alignment	→	Sam/bam	72min
S3 R1 fastq; R2 fastq	→	Fastp QC	→	Alignment	→	Sam/bam	72min
S4 R1 fastq; R2 fastq	→	Fastp QC	→	Alignment	→	Sam/bam	72min
S5 R1 fastq; R2 fastq	→	Fastp QC	→	Alignment	→	Sam/bam	72min
S6 R1 fastq; R2 fastq	→	Fastp QC	→	Alignment	→	Sam/bam	72min
							7 hours, 12 minutes

Loop. 2x cores

		10min		12min		8min	
S1	→	Fastp QC	→	Alignment	→	Sam/bam	36min
S2	→	Fastp QC	→	Alignment	→	Sam/bam	36min
S3	→	Fastp QC	→	Alignment	→	Sam/bam	36min
S4	→	Fastp QC	→	Alignment	→	Sam/bam	36min
S5	→	Fastp QC	→	Alignment	→	Sam/bam	36min
S6	→	Fastp QC	→	Alignment	→	Sam/bam	36min
							3 hours, 36 minutes

Cores:	1x	2x	4x*	8x*
Per sample time (minutes):	72	36	24	20
Total time (hours:minutes) for 6 iterations:	7:12	3:36	2:24	2:00

**diminishing return*

PARALLEL TIMING

				1x	2x	4x	8x
S1 R1 fastq; R2 fastq	→ Fastp QC	→ Alignment	→ Sam/bam	72min	36min	24min	20min
S2 R1 fastq; R2 fastq	→ Fastp QC	→ Alignment	→ Sam/bam	72min	36min	24min	20min
S3 R1 fastq; R2 fastq	→ Fastp QC	→ Alignment	→ Sam/bam	72min	36min	24min	20min
S4 R1 fastq; R2 fastq	→ Fastp QC	→ Alignment	→ Sam/bam	72min	36min	24min	20min
S5 R1 fastq; R2 fastq	→ Fastp QC	→ Alignment	→ Sam/bam	72min	36min	24min	20min
S6 R1 fastq; R2 fastq	→ Fastp QC	→ Alignment	→ Sam/bam	72min	36min	24min	20min
				72min	**36min**	**24min**	**20min**

- Put independent work in independent jobs:
  - Riviera: 40 Nodes, 7912 cores
  - Alpine: 696 Nodes, 40,200 cores
  - HPC!!!

- Parallelization
  - In a bash script: distribute jobs across cores
  - Scaling up:
    - Total run time is largest individual job, not sum of jobs
    - Dependent on queue- resources/priority

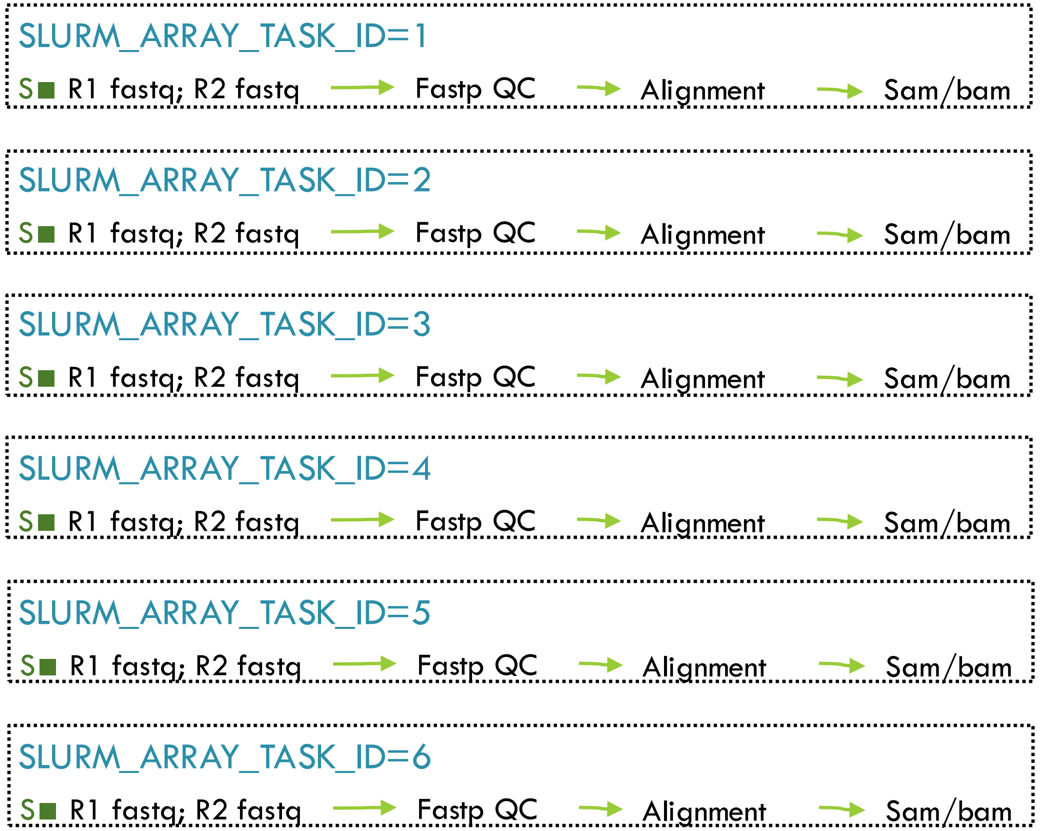
Using sbatch with the array argument

```
$ sbatch --array=1-6 main.sh
```

<- hit enter

Submitted job with ID 9223918

Launches Main RNA-seq Analysis script *6 times;*
sets SLURM_ARRAY_TASK_ID to different values: (1,2,3,4,5,6)



SLURM_ARRAY_TASK_ID=1

S■ R1 fastq; R2 fastq → Fastp QC → Alignment → Sam/bam

SLURM_ARRAY_TASK_ID=2

S■ R1 fastq; R2 fastq → Fastp QC → Alignment → Sam/bam

SLURM_ARRAY_TASK_ID=3

S■ R1 fastq; R2 fastq → Fastp QC → Alignment → Sam/bam

SLURM_ARRAY_TASK_ID=4

S■ R1 fastq; R2 fastq → Fastp QC → Alignment → Sam/bam

SLURM_ARRAY_TASK_ID=5

S■ R1 fastq; R2 fastq → Fastp QC → Alignment → Sam/bam

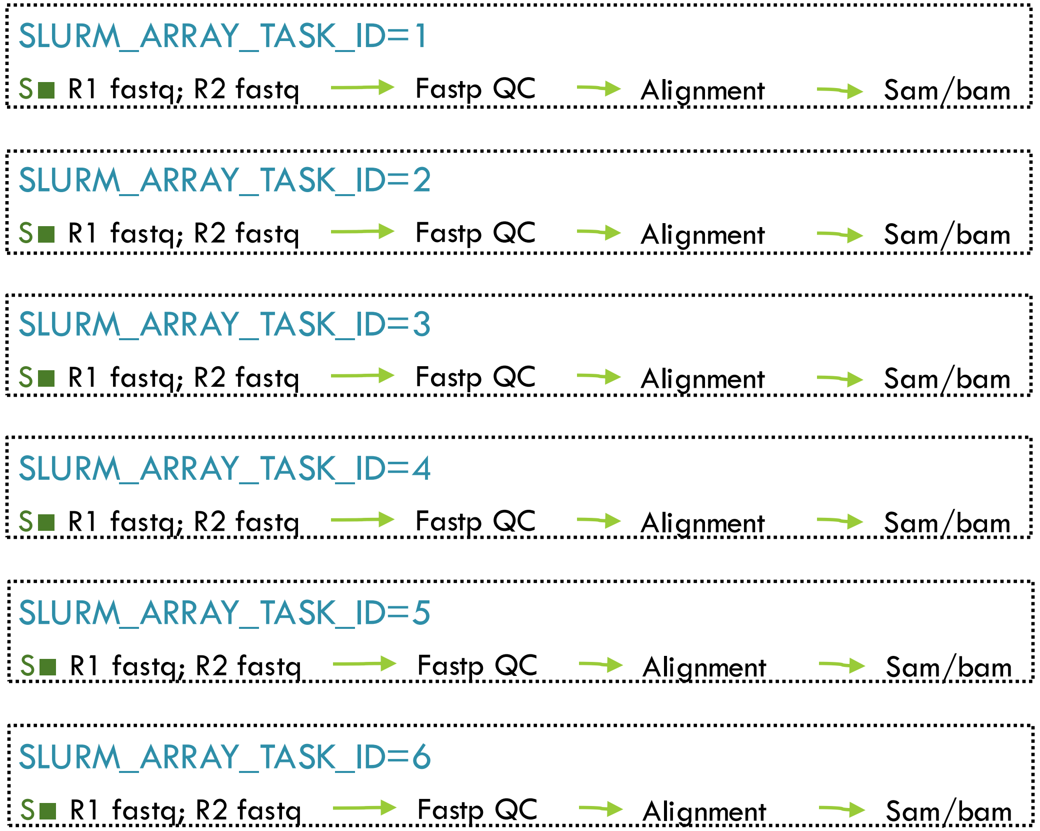
SLURM_ARRAY_TASK_ID=6

S■ R1 fastq; R2 fastq → Fastp QC → Alignment → Sam/bam

```
JOBID   PARTITION    NAME    USER ST     TIME  NODES NODELIST(REASON)
25736_1 short-cpu rna-seq   dking  R     0:04     1 node001
25736_2 short-cpu rna-seq   dking  R     0:04     1 node001
25736_3 short-cpu rna-seq   dking  R     0:04     1 node001
25736_4 short-cpu rna-seq   dking  R     0:01     1 node001
25736_5 short-cpu rna-seq   dking  R     0:01     1 node001
25736_6 short-cpu rna-seq   dking  R     0:01     1 node001
```

Using sbatch with the array argument

```
$ sbatch --array=1-6 main.sh                <- hit enter

 Submitted job with ID 9223918
```



SLURM_ARRAY_TASK_ID=1
S■ R1 fastq; R2 fastq → Fastp QC → Alignment → Sam/bam

SLURM_ARRAY_TASK_ID=2
S■ R1 fastq; R2 fastq → Fastp QC → Alignment → Sam/bam

SLURM_ARRAY_TASK_ID=3
S■ R1 fastq; R2 fastq → Fastp QC → Alignment → Sam/bam

SLURM_ARRAY_TASK_ID=4
S■ R1 fastq; R2 fastq → Fastp QC → Alignment → Sam/bam

SLURM_ARRAY_TASK_ID=5
S■ R1 fastq; R2 fastq → Fastp QC → Alignment → Sam/bam

SLURM_ARRAY_TASK_ID=6
S■ R1 fastq; R2 fastq → Fastp QC → Alignment → Sam/bam

Launches Main RNA-seq Analysis script *6 times;*
sets SLURM_ARRAY_TASK_ID to different values: (1,2,3,4,5,6)

What's this? → S■ R1 fastq; R2 fastq

Base input filenames on SLURM_ARRAY_TASK_ID

e.g.:

```
R1=S${SLURM_ARRAY_TASK_ID}_R1.fastq
R2=S${SLURM_ARRAY_TASK_ID}_R2.fastq
fastp -i $R1 -I $R2
```

↓ SLURM_ARRAY_TASK_ID=3

```
R1=S3_R1.fastq
R2=S3_R2.fastq
fastp -i S3_R1.fastq -I S3_R2.fastq
```

Features of array jobs:

monitoring with squeue

```
$ squeue -u $USER
       JOBID   PARTITION     NAME     USER ST      TIME  NODES NODELIST(REASON)
       25736_1 short-cpu rna-seq    dking  R      0:04      1 node001
       25736_2 short-cpu rna-seq    dking  R      0:04      1 node001
       25736_3 short-cpu rna-seq    dking  R      0:04      1 node001
       25736_4 short-cpu rna-seq    dking  R      0:01      1 node001
       25736_5 short-cpu rna-seq    dking  R      0:01      1 node001
       25736_6 short-cpu rna-seq    dking  R      0:01      1 node001
```

Run as a group,
Share JOB_ID
Number after '_' is array job ID.

Might be staggered
due to resource/priority

Features of array jobs: Specifying log file names

Single job script

single.sh

```
#!/usr/bin/env bash
#SBATCH …
#SBATCH --job-name=rna-seq
#SBATCH --output=%x.%j.log
#SBATCH …
…
```

Log file:

```
rna-seq.25736.log
  %x      %j
```

Array job script

array.sh

```
#!/usr/bin/env bash
#SBATCH …
#SBATCH --job-name=rna-seq
#SBATCH --output=%x.%A_%a.log
#SBATCH …
…
```

```
rna-seq.25736_1.log
rna-seq.25736_2.log
rna-seq.25736_3.log
rna-seq.25736_4.log
rna-seq.25736_5.log
rna-seq.25736_6.log
   %x      %A  %a
```

Features of array jobs: filename patterns for logs

FILENAME PATTERN

sbatch allows for a filename pattern to contain one or more replacement symbols, which are a percent sign "%" followed by a letter (e.g. %j).

%A Job array's master job allocation number.
%a Job array ID (index) number.
%j jobid of the running job.
%x Job name.

--From “sbatch” help page

```
rna-seq.25736.log
   %x      %j

rna-seq.25736_1.log
rna-seq.25736_2.log
rna-seq.25736_3.log
rna-seq.25736_4.log
rna-seq.25736_5.log
rna-seq.25736_6.log
   %x     %A  %a
```

SUBMISSION SCRIPT VERSUS ARRAY

submit.sh

```
#!/usr/bin/env bash
# run as `bash submit.sh`

for inputfile in *.R1.fastq
do
   sbatch single_rnaseq_analyzer.sh $inputfile
done
```

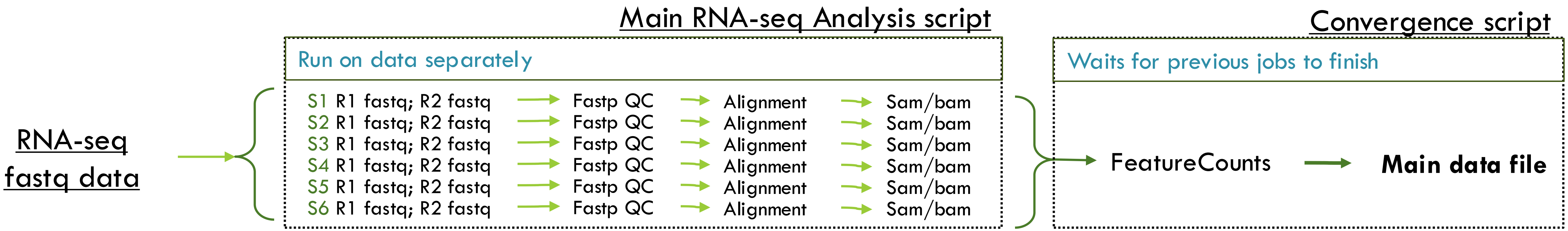
Simpler script, but no longer tied to a shared job id.

Harder to track, coordinate with other jobs

JOB COORDINATION

Converge after array job

Parallel workflow ∃Convergence



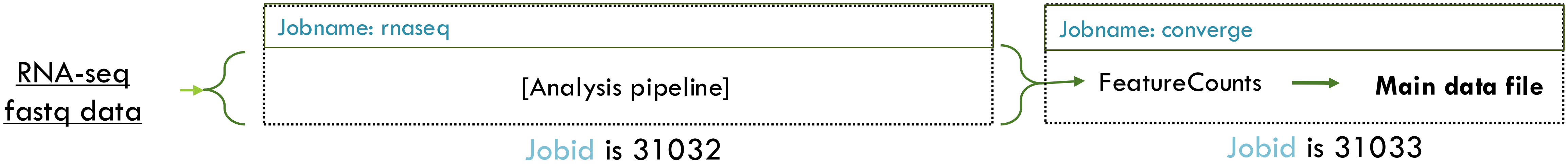
How?

```
$ sbatch --array=1-6 main.sh
Submitted job with ID 9223918
$ sbatch --dependency=afterok:9223918 converge.sh
Submitted job with ID 9223925
```

Capture job id

```
$ jobid=$(sbatch –p --array=1-6 main.sh)
$ sbatch --dependency=afterok:$(jobid) converge.sh
Submitted job with ID 9223925
```

SLURM DEPENDENCIES



RNA-seq fastq data

Jobname: rnaseq

[Analysis pipeline]

Jobid is 31032

Jobname: converge

FeatureCounts → Main data file

Jobid is 31033

```
$ jobid=$(sbatch –p --array=1-6 main.sh)
$ sbatch --dependency=afterok:$(jobid) converge.sh
Submitted job with ID 31033
```

```
$ squeue –u $USER
JOBID PARTITION     NAME     USER ST TIME  NODES NODELIST(REASON)
31033   short-cpu converge    dking PD 0:00   1 (Dependency)
31032_1 short-cpu rnaseq      dking  R 0:01   1 node001
31032_2 short-cpu rnaseq      dking  R 0:01   1 node001
31032_3 short-cpu rnaseq      dking  R 0:01   1 node001
…
```

SLURM DEPENDENCIES

-d, --dependency=<dependency_list>

- Defer the start of this job until the specified dependencies have been satisfied.
- <dependency_list> is of the form

afterok:job_id -Job with job_id must complete successfully

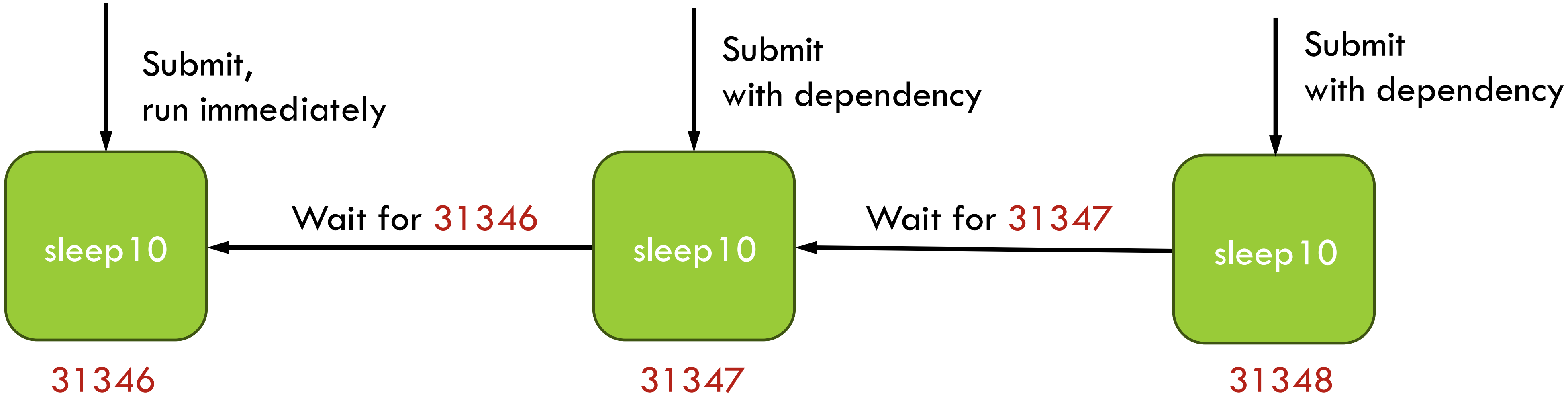
afternotok:job_id -Job with job_id failed: (non-zero exit code, node failure, timed out, etc). *Cleanup/error script?*

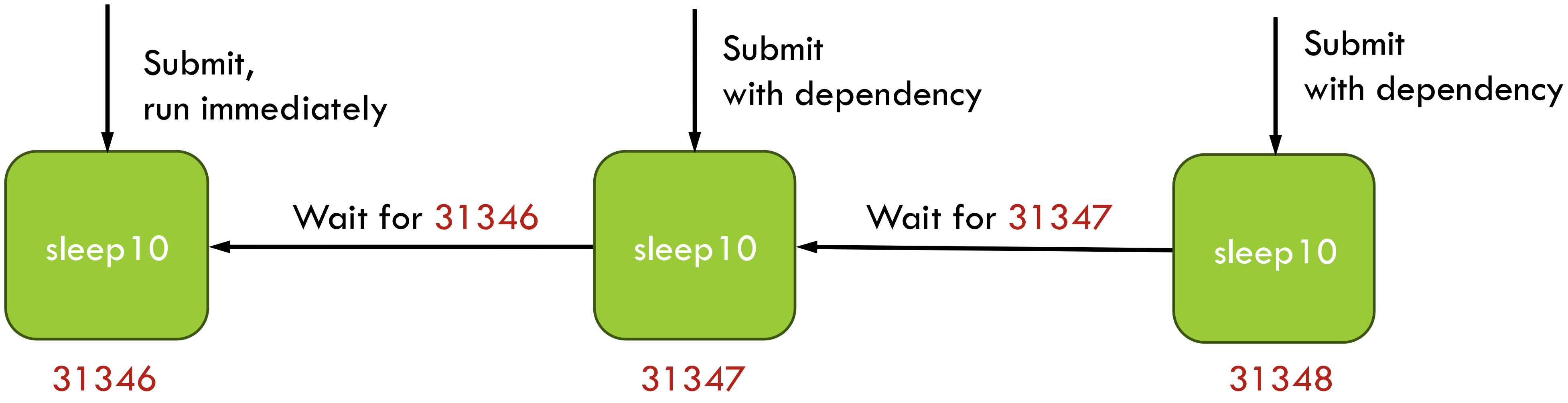
afterany:job_id – Job with job_id completes for any reason

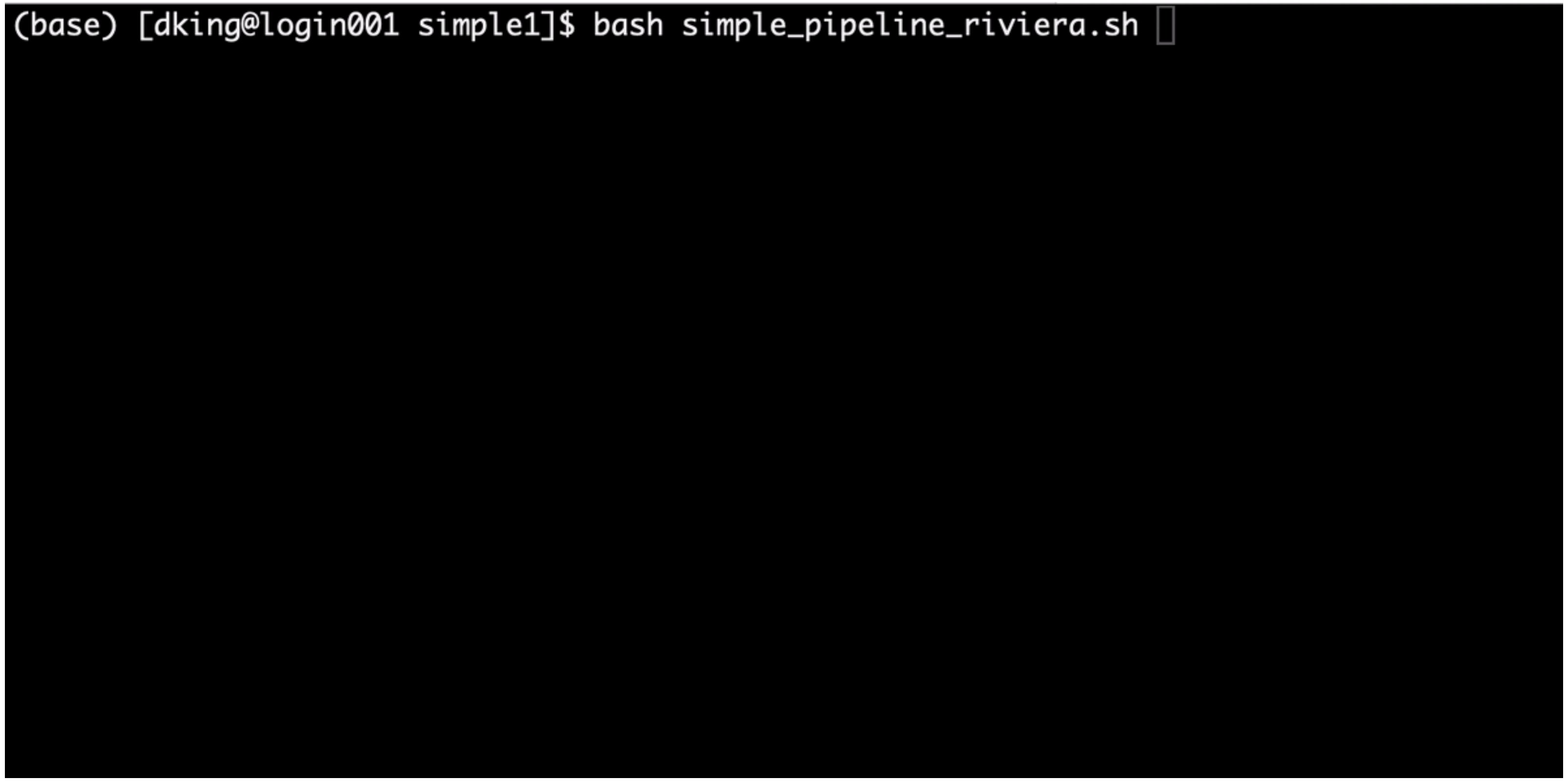
See sbatch documentation for more complex forms

SIMPLE PIPELINE WITH DEPENDENCIES

```
$ sbatch sleep10.sh
Submitted job id 31346
$ sbatch --dependency=afterok:31346 sleep10.sh
Submitted job id 31347
$ sbatch --dependency=afterok:31347 sleep10.sh
Submitted job id 31348
```



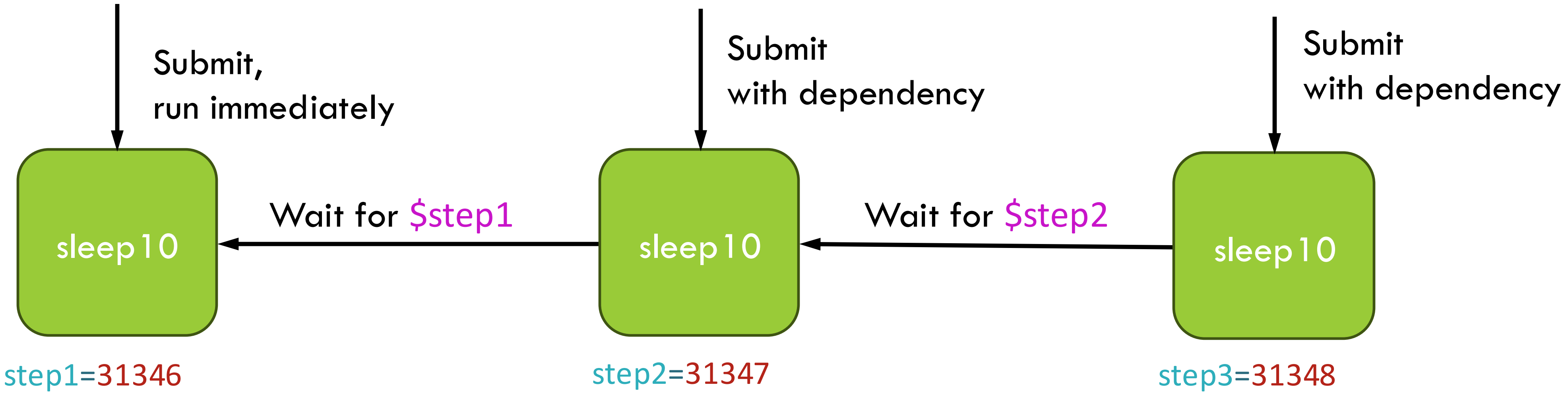




```
(base) [dking@login001 simple1]$ bash simple_pipeline_riviera.sh
```

SCRIPTABLE PIPELINE

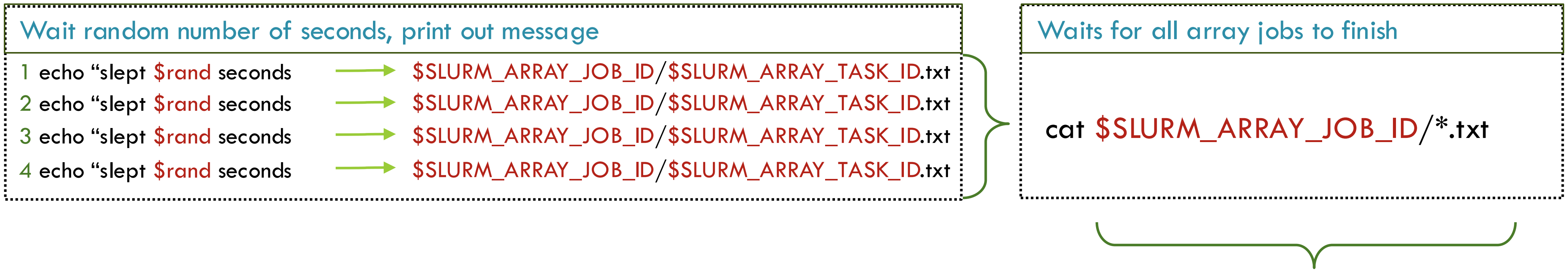
```
step1=$(sbatch --parsable sleep10.sh)
step2=$(sbatch --parsable --dependency=afterok:$step1 sleep10.sh)
step3=$(sbatch --parsable --dependency=afterok:$step2 sleep10.sh)
```



ARRAY WITH CONVERGENCE

Array Script

Convergence script



Wait random number of seconds, print out message

1 echo “slept $rand seconds → $SLURM_ARRAY_JOB_ID/$SLURM_ARRAY_TASK_ID.txt

2 echo “slept $rand seconds → $SLURM_ARRAY_JOB_ID/$SLURM_ARRAY_TASK_ID.txt

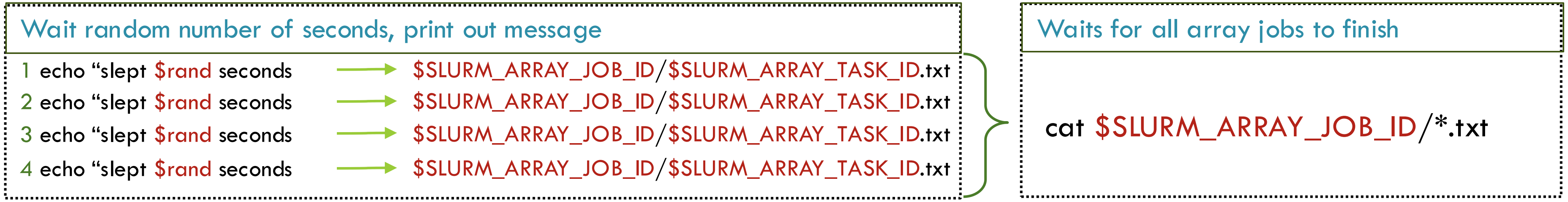
3 echo “slept $rand seconds → $SLURM_ARRAY_JOB_ID/$SLURM_ARRAY_TASK_ID.txt

4 echo “slept $rand seconds → $SLURM_ARRAY_JOB_ID/$SLURM_ARRAY_TASK_ID.txt

Waits for all array jobs to finish

```
cat $SLURM_ARRAY_JOB_ID/*.txt
```

```
slept 15 seconds
slept 11 seconds
slept 16 seconds
slept 14 seconds
```



Wait random number of seconds, print out message

1 echo "slept $rand seconds → $SLURM_ARRAY_JOB_ID/$SLURM_ARRAY_TASK_ID.txt

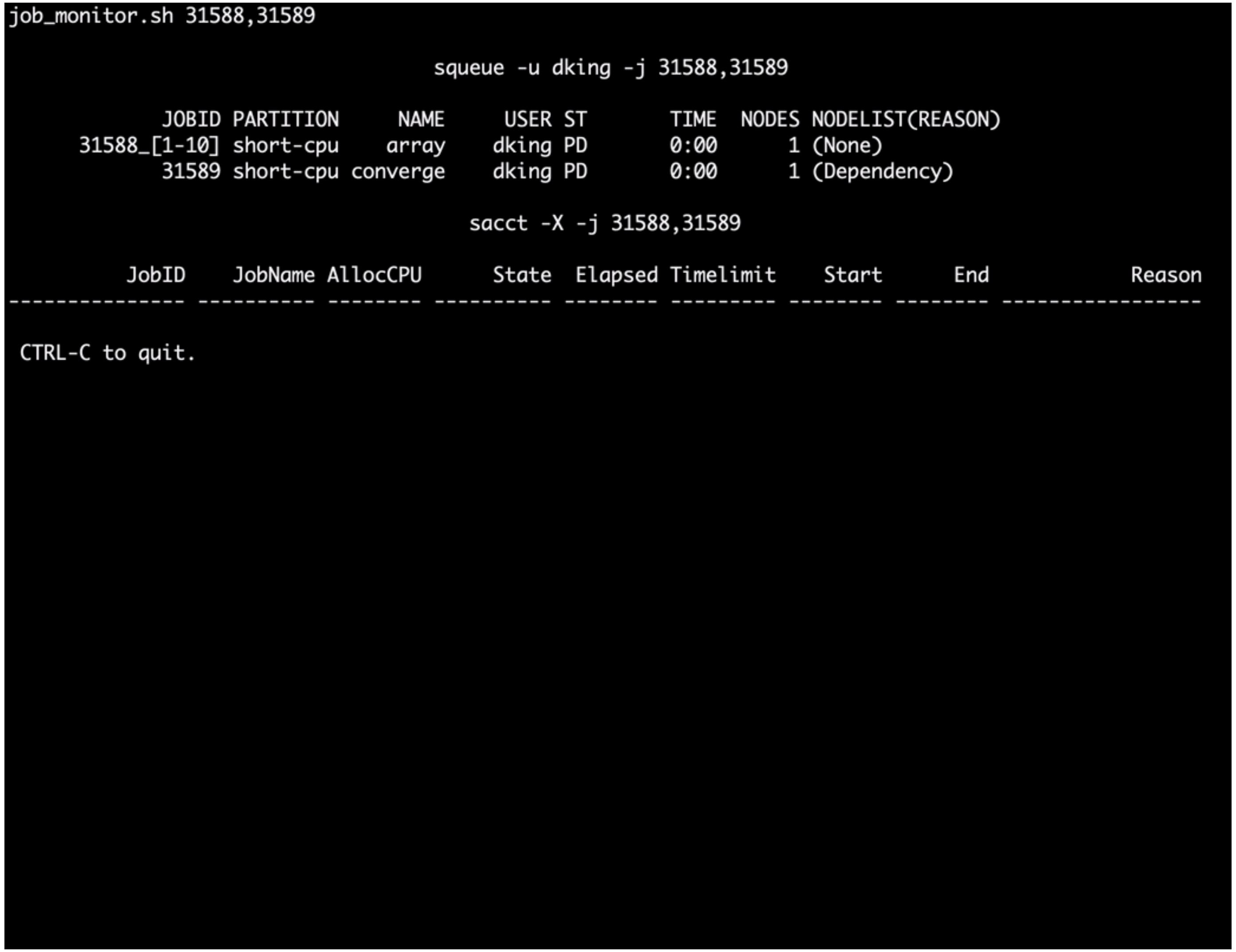
2 echo "slept $rand seconds → $SLURM_ARRAY_JOB_ID/$SLURM_ARRAY_TASK_ID.txt

3 echo "slept $rand seconds → $SLURM_ARRAY_JOB_ID/$SLURM_ARRAY_TASK_ID.txt

4 echo "slept $rand seconds → $SLURM_ARRAY_JOB_ID/$SLURM_ARRAY_TASK_ID.txt

Waits for all array jobs to finish

cat $SLURM_ARRAY_JOB_ID/*.txt



```
job_monitor.sh 31588,31589

                                 squeue -u dking -j 31588,31589

            JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
      31588_[1-10] short-cpu    array    dking PD       0:00      1 (None)
            31589 short-cpu converge    dking PD       0:00      1 (Dependency)

                                    sacct -X -j 31588,31589

         JobID    JobName AllocCPU      State  Elapsed Timelimit    Start      End            Reason
--------------- ---------- -------- ---------- -------- --------- -------- -------- -----------------

 CTRL-C to quit.
```

ARRAY/CONVERGENCE LAUNCHER SCRIPT

```
#!/usr/bin/env bash
array_job_id=$(sbatch --parsable --array=1-10 array.sbatch)
converge_job_id=$(sbatch --parsable --dependency=afterok:${array_job_id} converge.sbatch $array_job_id)
```

*This script is run with `bash` not `sbatch`

EXAMPLES: LOOPS TO ARRAYS

Some examples of how to modify your scripts to go from sequential to array

CHANGING YOUR SCRIPT TO AN ARRAY SCRIPT

Three ways to modify/remove loops

1. Arguments on the command line
2. Already looping over an array
3. Reading a file, use sed to select specific line

CHANGING YOUR SCRIPT TO AN ARRAY SCRIPT

1. Arguments on the command line

Original: script loops over ‘$@’

```
for arg in "$@"; do
   progname $arg
   …
done
```

New version:

- convert command line to array
- Set loop variable from array and $SLURM_ARRAY_TASK_ID
- Remove loop

```
args=($@)
arg=${args[$SLURM_ARRAY_TASK_ID]}
#for arg in "$@"; do
   progname $arg
   …
#done
```

```
args=($@)
arg=${args[$SLURM_ARRAY_TASK_ID]}
progname $arg
…
```

CHANGING YOUR SCRIPT TO AN ARRAY SCRIPT

2. Already looping over an array in main script

Original: ‘@’ expands to full array

```
# Loop through each subdirectory
for folder_name in "${subdirectories[@]}"; do
  # Verify if the folder contains exactly two files: .dv and its reference file
  dv_files=($(find "$folder_name" -maxdepth 1 -type f -name "*.dv" | sort))
  …
  …
  …
done
```

New version: Keep the loop, replace with $SLURM_ARRAY_TASK_ID

```
# Loop through each subdirectory
for folder_name in "${subdirectories[$SLURM_ARRAY_TASK_ID]}"; do
  # Verify if the folder contains exactly two files: .dv and its reference file
  dv_files=($(find "$folder_name" -maxdepth 1 -type f -name "*.dv" | sort))
  …
  …
  …
done
```

CHANGING YOUR SCRIPT TO AN ARRAY SCRIPT

3. Select a line of a file for input with **sed**

Original: while loop through file

```
while read -a line; do
   arg1=${line[0]}
   arg2=${line[1]}
   arg3=${line[2]}
   cmd $arg1 $arg2 $arg3
done < metadata.txt
```

New version: Pass $SLURM_ARRAY_TASK_ID to sed

```
line=( $(sed -n "${SLURM_ARRAY_TASK_ID}p" metadata.txt) )
arg1=${line[0]}
arg2=${line[1]}
arg3=${line[2]}
cmd $arg1 $arg2 $arg3
```

DEBUGGING WITH EXIT TRAP

Cool stuff

DEBUGGING/COOL STUFF

script

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

echo "<job started on $(date)>"

cmd1

cmd2

cmd3

cmd4

echo "<job finished on $(date)>"
```

Output/log if successful

```
<job started on Fri Dec  6 13:25>

[ cmd1 output ]

[ cmd2 output ]

[ cmd3 output ]

[ cmd4 output ]

<job finished on Fri Dec  6 15:25>
```

An error early on

```
<job started on Fri Dec  6 13:25>

[ cmd1 output ]

cmd2 error output cmd2 error cmd2 err
or output cmd2 error output
cmd3 error output cmd3 error cmd3 err
or output cmd3 error output
cmd4 error output cmd4 error cmd4 err
or output cmd4 error output
```

DEBUGGING/COOL STUFF – USING BASH SETTINGS

script

```
#!/usr/bin/env bash
set -e # quit on error
set -o pipefail # detect error in pipe
echo "<job started on $(date)>"

cmd1

cmd2
cmd3
cmd4
echo "<job finished on $(date)>"
```

An error early on

```
<job started on Fri Dec  6 13:25>

[ cmd1 output ]

cmd2 error output cmd2 error cmd2 err
or output cmd2 error output
```

DEBUGGING/COOL STUFF – THE EXIT SIGNAL

script

```
#!/usr/bin/env bash
set -e # quit on error
set -o pipefail # detect error in pipe
echo "<job started on $(date)>"

cmd1

cmd2
cmd3
cmd4
echo "<job finished on $(date)>"
```

An error early on

```
<job started on Fri Dec  6 13:25>

[ cmd1 output ]

cmd2 error output cmd2 error cmd2 error output cmd2 error output

<job FAILED on Fri Dec  6 13:26>
```

DEBUGGING/COOL STUFF – TRAPPING SIGNALS

Signals can be sent to any process on linux to interrupt, cancel jobs, among other things

Some signals can be handled “trapped” and given code to run

Syntax:

```
trap code_to_run SIGNAL
```

Example:

Script says “have a nice day” when user CTRL-C’s their script:

```
trap 'echo "have a nice day"; exit 0’ SIGINT
```

DEBUGGING/COOL STUFF – THE EXIT TRAP

script

```
#!/usr/bin/env bash
set -e # quit on error
set -o pipefail # detect error in pipe

echo "<job started on $(date)>"
trap 'echo ”Job ended at $(date) "' EXIT

cmd1

cmd2
cmd3
cmd4
echo "<job finished on $(date)>"
```

An error early on

```
<job started on Fri Dec  6 13:25>

[ cmd1 output ]

cmd2 error output cmd2 error cmd2 err
or output cmd2 error output

<job ended at Fri Dec  6 13:26>
```

DEBUGGING/COOL STUFF – THE EXIT TRAP

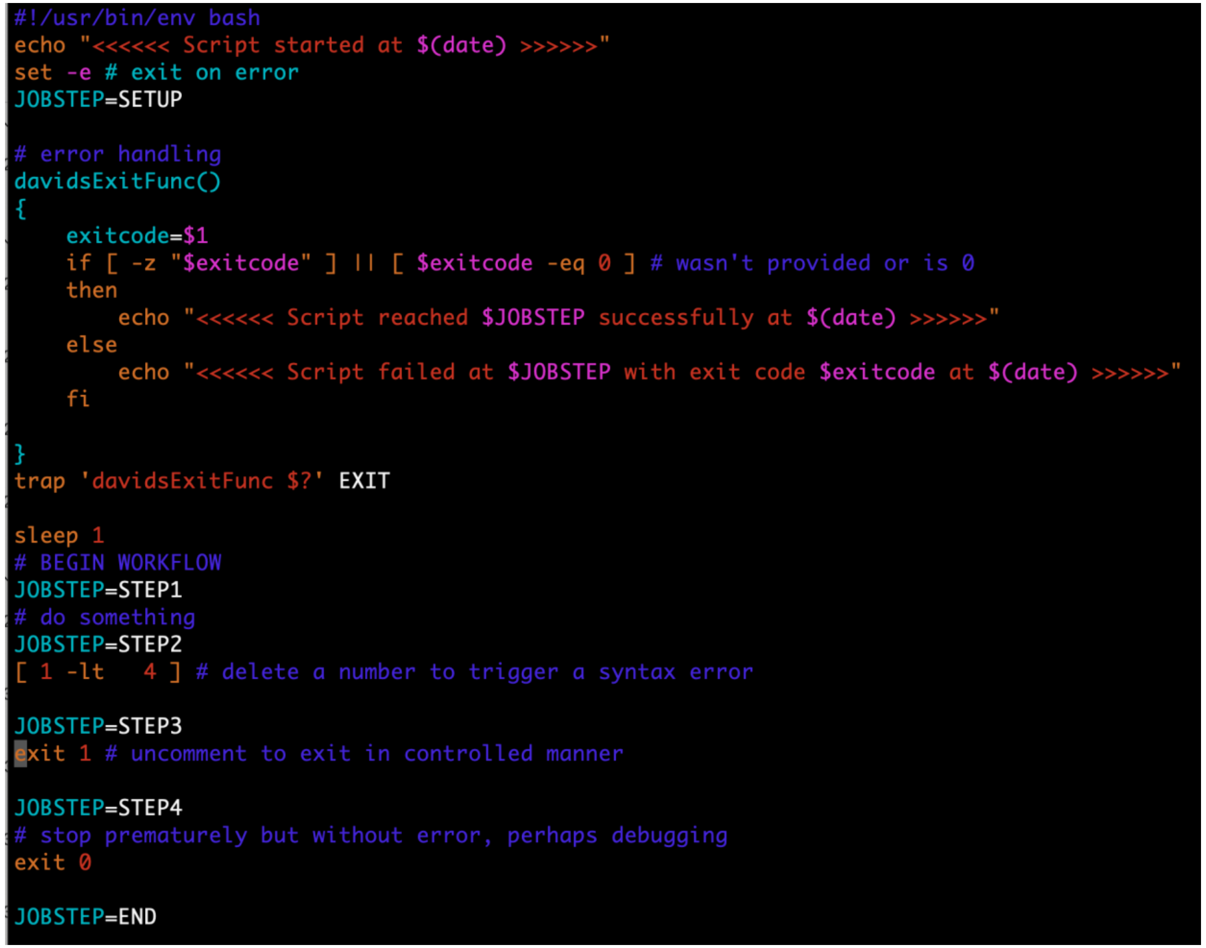
script

```
#!/usr/bin/env bash
set -e # quit on error
set -o pipefail # detect error in pipe

echo "<job started on $(date)>"
trap 'echo ”Job ended at $(date) "' EXIT

cmd1

cmd2
cmd3
cmd4
echo "<job finished on $(date)>"
```

Output/log if successful

```
<job started on Fri Dec  6 13:25>

[ cmd1 output ]

[ cmd2 output ]

[ cmd3 output ]

[ cmd4 output ]

<job finished on Fri Dec  6 15:25>
<job ended at Fri Dec  6 15:25>
```

EXIT TRAP WITH HANDLING FUNCTION



```bash
#!/usr/bin/env bash
echo "<<<<<< Script started at $(date) >>>>>>"
set -e # exit on error
JOBSTEP=SETUP

# error handling
davidsExitFunc()
{
    exitcode=$1
    if [ -z "$exitcode" ] || [ $exitcode -eq 0 ] # wasn't provided or is 0
    then
        echo "<<<<<< Script reached $JOBSTEP successfully at $(date) >>>>>>"
    else
        echo "<<<<<< Script failed at $JOBSTEP with exit code $exitcode at $(date) >>>>>>"
    fi

}
trap 'davidsExitFunc $?' EXIT

sleep 1
# BEGIN WORKFLOW
JOBSTEP=STEP1
# do something
JOBSTEP=STEP2
[ 1 -lt   4 ] # delete a number to trigger a syntax error

JOBSTEP=STEP3
exit 1 # uncomment to exit in controlled manner

JOBSTEP=STEP4
# stop prematurely but without error, perhaps debugging
exit 0

JOBSTEP=END
```

# quits at exit 1 under JOBSTEP=STEP3

```
$ bash cleanup_function.sh
<<<<<< Script started at Mon Dec  9 11:51 >>>>>>
<<<<<< Script failed at STEP3 with exit code 1 at Mon Dec  9 11:51:49 >>>>>>
```

# delete the '4' under JOBSTEP=STEP2 (syntax error)

```
$ bash cleanup_function.sh
<<<<<< Script started at Mon Dec  9 11:53 >>>>>>
cleanup_function.sh: line 25: [: 1: unary operator expected
<<<<<< Script failed at STEP2 with exit code 2 at Mon Dec  9 11:53:42 >>>>>>
```

# replace the 4 and comment out exit statements

```
$ bash cleanup_function.sh
<<<<<< Script started at Mon Dec  9 11:58:11 >>>>>>
<<<<<< Script reached END successfully at Mon Dec  9 11:58:12 >>>>>>
```

EXIT TRAP WITH HANDLING FUNCTION

Other uses:

- Close connections, free resources
- Ex: Close Amazon machine instance (charges MONEY $$$)

```
#!/bin/bash

# define the base AMI ID somehow

ami=$1

function finish {
  ec2-terminate-instances "$ami"
}

trap finish EXIT

ec2-run-instances "$ami"
```

Simplified from: http://redsymbol.net/articles/bash-exit-traps/

REPO FOR THIS TALK AND SLURM TOOLS

https://github.com/Colorado-State-University-CMB/slurm-scripting-pipelines

- Use: git clone https://github.com/Colorado-State-University-CMB/slurm-scripting-pipelines on alpine or riviera
- Do bash INSTALL.sh to install job_monitor.sh, which is used in the pipeline examples
- The subdirectories contain the examples used in this presentation