

# RCBio: easy, quick HPC pipeline builder & runner

HMS Research Computing

Lingsheng Dong: [Lingsheng\\_Dong@hms.harvard.edu](mailto:Lingsheng_Dong@hms.harvard.edu)

Kathleen Keating: [Kathleen\\_Keating@hms.harvard.edu](mailto:Kathleen_Keating@hms.harvard.edu)

RC Help: [rchelp@hms.harvard.edu](mailto:rchelp@hms.harvard.edu)



# What is RCBio?

---

- RCBio is a pipeline runner and workflow tool developed and supported by HMS RC consultants.
- It was initially designed for the [Lee lab](#) for their complex 15+ step image analysis workflow.
- The project was later expanded to all users on our previous cluster, Orchestra, which is now retired.
- RCBio was ported to our [current cluster, O2](#), and we have continued development of the tool. RCBio currently supports job submission to the Slurm scheduler.

# Advantages of using RCBio

---

- *Proven track record:* The RCBio project has already helped many HMS researchers with great successes.
- *Quick to setup and run:* Researchers can easily build and run new HPC pipelines and share with colleagues and collaborators.
- *Assistance available:* HMS RC consultants also [build selected pipelines and share with researchers](#), as well as provide support for RCBio.

# User comments about RCBio

---

"PS-- **this pipeline runner is THE most amazing thing.** Super organized, super easy to rerun failed jobs, and so easy to debug right from my email because the failure emails are so informative. Thank you for creating this!!"

**"I was able to code these steps up in a single bash script with rcbio pipeline decorators; all tracking and job dependencies were taken care of and I received emails for successful and failed jobs.** On AWS, the time to code and run these steps would have been significantly longer. First, .... In contrast, error logs generated by rcbio are intuitively named and located, and the pipeline runner enables us to automatically resubmit failed jobs only (all at once, instead of one-by-one). Second, ... . Using the pipeline runner, subsequent jobs automatically begin running once their dependencies have been satisfied."

-- Shilpa Kobren, a postdoc from HMS DBMI

# How do I access RCBio?

---

- RCBio is available through the module system on our cluster O2:

```
# shows currently available versions of rcbio
$ module spider rcbio
```

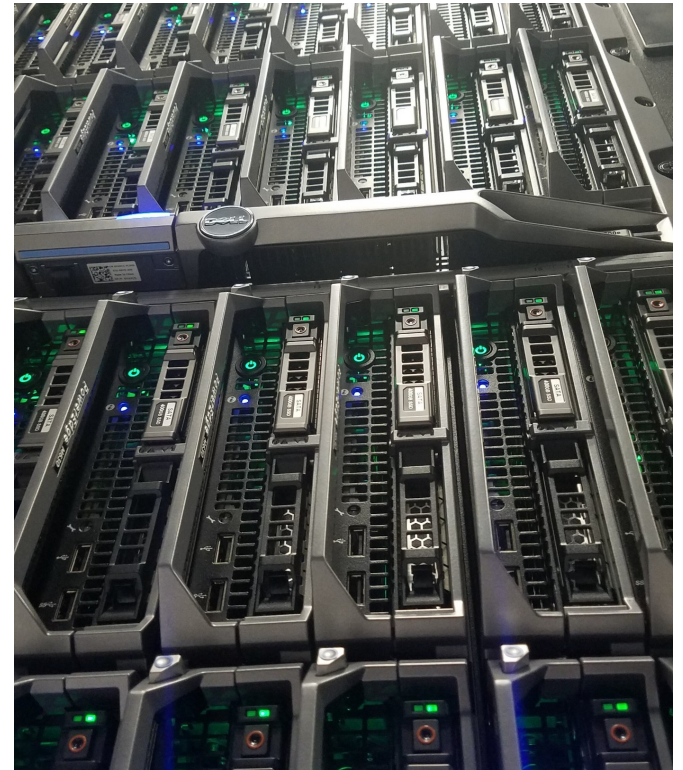
```
# load version 1.3 of rcbio
$ module load rcbio/1.3
```

- This class does require knowledge of O2 and the SLURM scheduler.
- We will have a quick review of using HPC, such as O2, and the SLURM scheduler now, for those who aren't familiar.

# High Performance Computing (HPC)

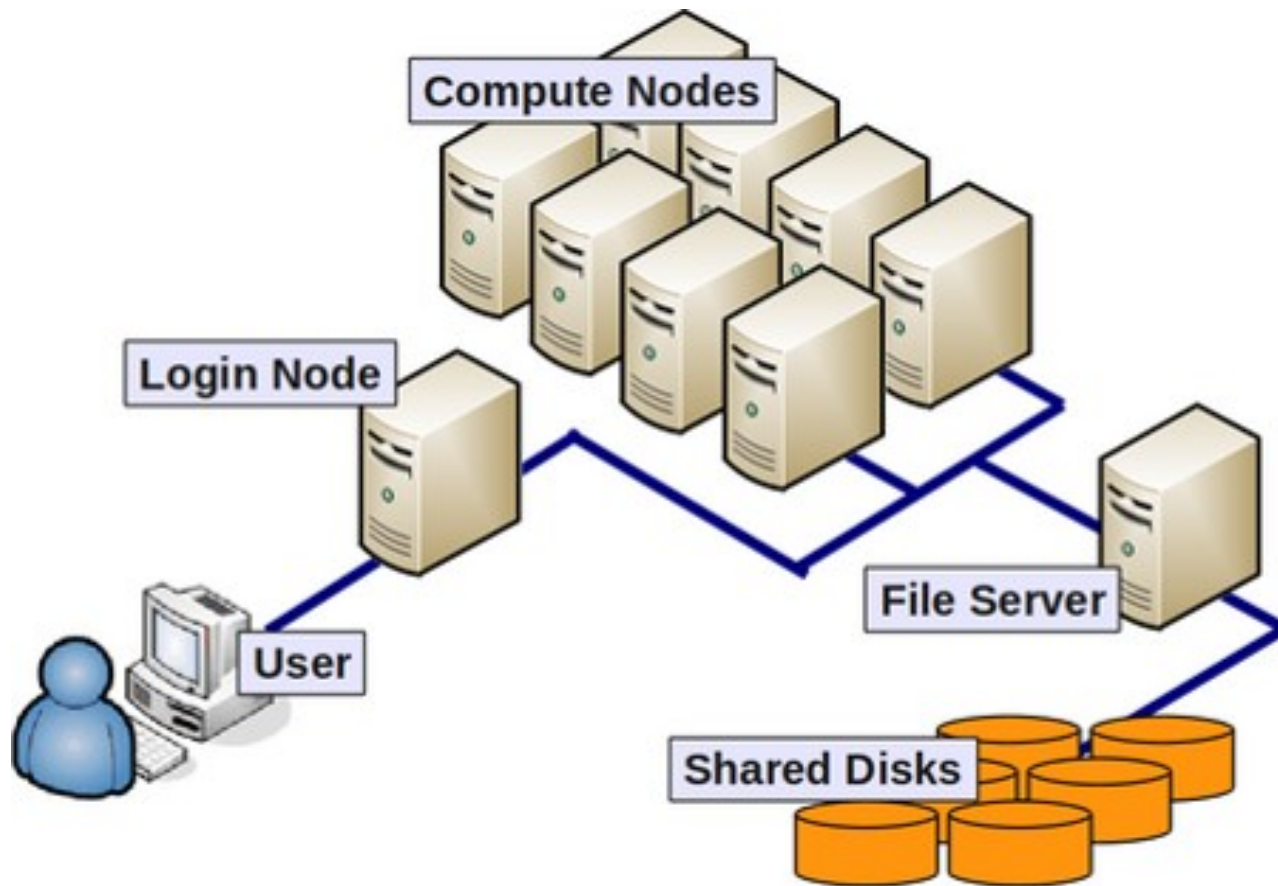
---

- Hundreds/thousands of computers with a lot of CPUs and memory
- Dedicated scheduler for parallel processing
- Shared software environment
- Shared storage
- Great support
  - [rchelp@hms.harvard.edu](mailto:rchelp@hms.harvard.edu)



# Generic Cluster Architecture

---



# Ways to submit jobs on O2 using the SLURM scheduler

---

- The login servers are not designed to handle intensive processes, and CPU usage is throttled. We need to submit jobs that request resources (like # cpus, memory, time, etc.) instead!
- Two types of jobs:
  - Interactive jobs using `srun`
    - Programs can be run directly once on compute node
  - Batch jobs using `sbatch`
    - Requires writing a job script (a text file) to submit



# Interactive Job example

```
mfk8@login0~$ srun --pty -p interactive -t 0-2:00 --mem 2G bash
```

“srun --pty” is how interactive jobs are started

“-p interactive” is the partition

“-t 0-2:00” is the time limit (2 hours)

“--mem 2G” is the memory requested, 2GiB

```
mfk8@compute-a:~$
```

Once on the compute node, you can run any command you want!

```
mfk8@compute-a:~$ bowtie -c 4 hg19 file1_1.fq file1_2.fq
```

# Batch job example script

---

```
#!/bin/bash
#SBATCH -p short #partition
#SBATCH -t 0-01:00 #time days-hr:min
#SBATCH -c X #number of cores
#SBATCH --mem=XG #memory per job (all cores), GiB
#SBATCH -o %j.out #out file
#SBATCH -e %j.err #error file
#SBATCH --mail-type=BEGIN,END,FAIL,ALL
#SBATCH --mail-user=mfk8@med.harvard.edu
# put any module load commands here
# put any analysis commands you want to run here
```

**Assuming we saved the script above as myjobscript.sh, submit it with sbatch like so: `sbatch myjobscript.sh`**

# Partitions (queues): -p

Partition	Priority	Max Runtime	Max Cores	Limits
short	12	12 hours	20	
medium	6	5 days	20	
long	4	30 days	20	
interactive	14	12 hours	20	2 job limit
priority	14	30 days	20	2 job limit
mpi	12	5 days	640	20 core min
highmem		5 days	20	
gpu, gpu_quad, gpu_requeue		160 GPU hours	34 (total)	420GiB (total)
transfer		5 days	4	

# Wall-Time: -t

---

- -t days-hours:minutes
- -t hours:minutes:seconds
- Need to specify how long you estimate your job will run for
- Aim for 125% over
- Subject to maximum per partition
- Excessive wall-time (like partition max) takes longer to dispatch, and affect fair-share

# CPU: -c

---

- -c X to designate CPU: max 20
- -N X to constrain all cores to X nodes
- CPU time: wall time (-t) \* (-c) CPUs used
- Unable to use CPU not requested (no overefficient jobs): cgroups constraint
- Adding more cores does not mean jobs will scale linearly with time, and causes longer pend times

# Memory: --mem

---

- Only 1GiB is allocated by default
- `--mem XG #total memory over all cores`
- `--mem-per-cpu XG #total memory per CPU requested, use for MPI`
- If you don't include a unit request (like GiB), it defaults to Mebibytes

# Output/Error Files

---

- Can add jobid to filename with %j
- Sample:
  - e %j.err
  - o %j.out
- SLURM by default creates this outfile: slurm-<jobid>.out
- Additional Flags
  - %a job array id
  - %A master array job id
  - %N node name
  - %u user id



# Mail

---

- Mail is not auto-generated upon completion/failure
- `#SBATCH --mail-type= NONE, BEGIN, END, FAIL, REQUEUE, ALL`
- `#SBATCH --mail-user=mfk8@med.harvard.edu`
- Not recommended, not a verbose output
- Use `02sacct` or `sacct` commands instead



# Job Dependencies

---

- `sbatch -d (or --dependency=)`
- `after:jobid #(asynchronous)`
- `afterany:jobid #after exit or done`
- `afterok:jobid #success, exit code 0`
- `afternotok:jobid #failure`
- `singleton #after jobs with same name have terminated`
- `--kill-on-invalid-dep=<yes|no> #kill on unmet dependency (on by default)`

# Job Monitoring: Current jobs

---

- `$ 02squeue`
  - JOBID, PARTITION, STATE, TIME\_LIMIT, TIME, NODELIST(REASON), ELIGIBLE\_TIME, START\_TIME, TRES
  - [02squeue documentation](#)
- *Other options:*
  - `$ squeue -u eCommons -t RUNNING/PENDING`
  - `$ squeue -u eCommons -p Partition`
  - `$ squeue -u eCommons -start`
- *Detailed job info:*
  - `$ scontrol show jobid <jobid>`

# Job Information: Past Jobs

---

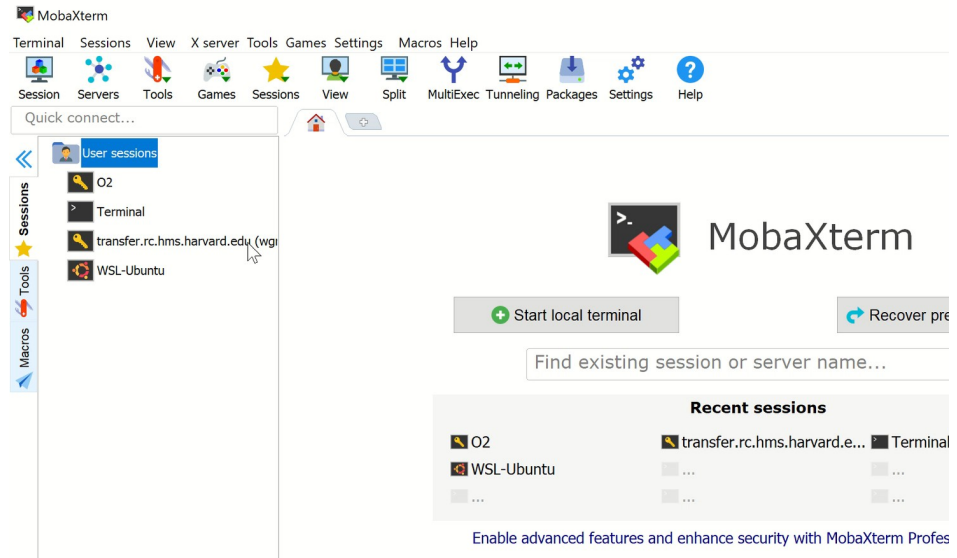
- `$ 02sacct`
  - JobID, Partition, State , NodeList, Start, Timelimit, Elapsed, CPUTime , TotalCPU, AllocTRES, MaxRSS
  - Can specify job ID, job status, and/or timeframe to report accounting info for
  - [02sacct documentation](#)
- *Other options:*
  - `$ sacct -j jobid`
  - `$ sacct -r partition`
  - `$ squeue -s state`
  - `$ sacct --helpformat #get available fields you can specify`

# Connecting to O2...

## MAC/Linux: Terminal

```
wrodriguez@MIT-RCG-L1801:~$
```

## Windows: MobaXterm



`ssh yourhmsid@o2.hms.harvard.edu`

2-Factor (when necessary): Choose 1/2/3 (push/phone/sms)

# Welcome to O2!

```
02

You've landed on login04 which is a
8 core system with 15.51 GiB memory
running kernel 3.10.0 born on 2018-12-04

==== O2 =====

News (Mar 15 2019)

+-----+
| * With a recent expansion, O2 now contains over 11,000 shared cpu cores! |
| * O2 requires 2-factor authentication for all logins originating from outside |
|   of the HMS network. Please see: |
|   https://wiki.rc.hms.harvard.edu/display/O2/Two+Factor+Authentication+on+O2 |
| |
| * Resource intensive jobs should not be run on O2's login servers. |
|   Any programs (Python, R, samtools, gzip, etc.) that run for more |
|   than ten minutes or use too much CPU on O2's login servers will be |
|   automatically killed. |
| * Learn more about O2 at: http://hmsrc.me/O2docs |
| * Status updates and upcoming service outages are posted at: |
|   https://wiki.rc.hms.harvard.edu/display/O2/O2+Cluster+Status |
+-----+

-----
Contact HMS Research Computing:

E-mail      rchelp@hms.harvard.edu
Web         https://rc.hms.harvard.edu
Twitter     @hms_rc

=====

[wgr4@login04 ~]$
```



# Welcome to O2!

```
02

You've landed on login04 which is a
8 core system with 15.51 GiB memory
running kernel 3.10.0 born on 2018-12-04

=====

[=====]

[wgr4@login04 ~]$

=====

contains over 11,000 shared cpu cores!

on for all logins originating from outside

display/O2/Two+Factor+Authentication+on+O2

t be run on O2's login servers.
, gzip, etc.) that run for more
CPU on O2's login servers will be
automatically killed.

* Learn more about O2 at: http://hmsrc.me/O2docs

* Status updates and upcoming service outages are posted at:
https://wiki.rc.hms.harvard.edu/display/O2/O2+Cluster+Status

-----
Contact HMS Research Computing:
E-mail      rchelp@hms.harvard.edu
Web         https://rc.hms.harvard.edu
Twitter     @hms_rc

=====

[wgr4@login04 ~]$
```



# Welcome to O2!

Your HMS ID



# Welcome to O2!

```
=====
```

```
[wgr4@login04 ~]$
```

You are logged into a “shell login server”  
Do **not** meant for heavy lifting!



# Welcome to O2!

```
=====
[wgr4@login04 ~]$
```

You're in your HOME directory.  
/home/<HMSID>

# Welcome to O2!

```
=====

[wgr4@login04 ~]$
```

Ready to receive commands!

# Bash commands

---

- `cd`: go to a directory
- `ls`: list directory and file
- `cp`: copy directory and file
- `mkdir`: create new directory
- `less`: view file content
- `cat`: show file content



# Automate commands with a “for” loop

---

- Repeat commands against an designated list
  - this syntax is for **bash**, but other shells (tcsh) are different

- Examples

```
$ for i in 1 2 3 ; do mkdir $i ; done
```

```
$ for i in `cat list.txt` ; do cp $i ~/work ; done
```

# Bash variable

---

Variable name must begin with alphabet character or underscore character (`_`), followed by one or more alphanumeric or underscore characters.

- Assign value:

`varName="value"`

`varName=value`

- Refer variable:

`${varName}`, `$varName`, `"$varName"`, `"$varName"`

# Bash path/file name manipulation

---

```
fileWithPath=/home/id/data/tumor.fq
```

```
dirname $fileWithPath          # /home/id/data
```

```
basename $fileWithPath         # tumor.fq
```

```
basename $fileWithPath .fq     #tumor
```

# Bash string manipulation

---

```
someString=abcdefghijk
```

```
echo ${someString#abc}      #defghijk
```

```
echo ${someString%ijk}      #abcdefgh
```

```
prefix=abc; suffix=ijk
```

```
echo ${someString#$prefix}   #defghijk
```

```
echo ${someString%$suffix}   #abcdefgh
```

# Bash string concatenate

---

```
sample=tumor; rep=2
```

```
echo $sample.$ref          # rumor2
```

```
echo $sample_$rep         # does not work
```

```
echo /data/group/$sample/$tumor/$rep
```



# Catch the output of command

---

```
output=`cmd -in $input -out $out`
```

```
output=$(cmd -in $input -out $out)
```

```
output=$(echo today is $(date))
```

```
output=`echo today is `date``
```

# Combine multiple commands

---

```
cmd1; cmd2; cmd3;
```

```
cmd1 && cmd2 && cmd3
```

```
mkdir outputDir && cmdToCreateOutput
```

```
ls $input && cmd -in $input || exit
```

# Creating a Simple Text File

---

“Nano,” “vi”, “emacs” are simple command-line editors available.

To create a new file, type the editor you want, then the name of the new file. To edit an existing file, do the same.

```
mfk8@compute-a:~$ nano myfile.txt
```

```
This is my new file text.
```

*(Control-X to save (yes) and exit.)*

```
mfk8@compute-a:~$
```

```
mfk8@compute-a:~$ ls  
myfile.txt
```

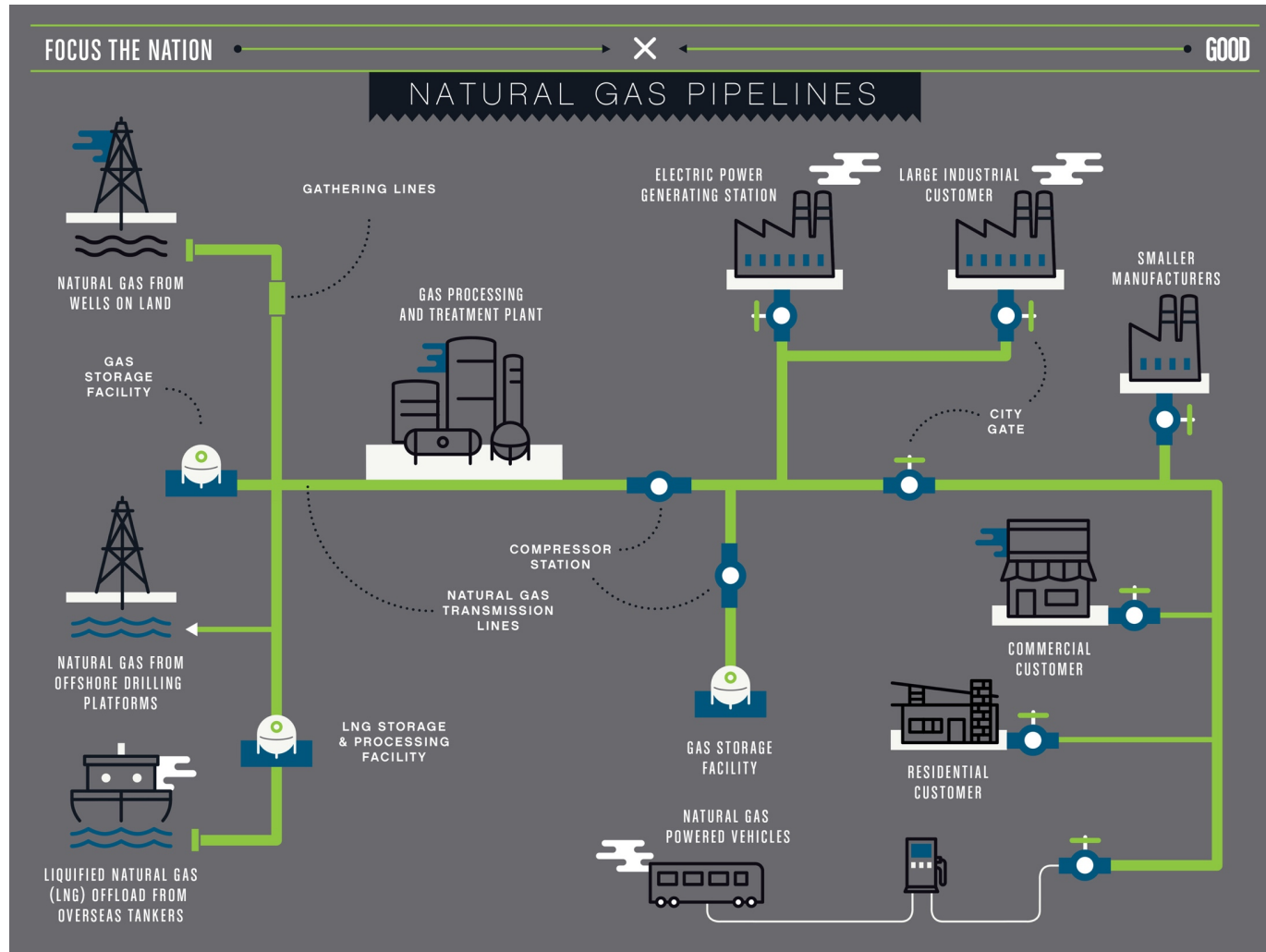


# Part of Natural gas pipeline



-- <https://en.mercoPress.com/data/cache/noticias/27951/0x0/gas.jpg>

# The whole picture



-- <https://assets.rbl.ms/19920757/origin.jpg>

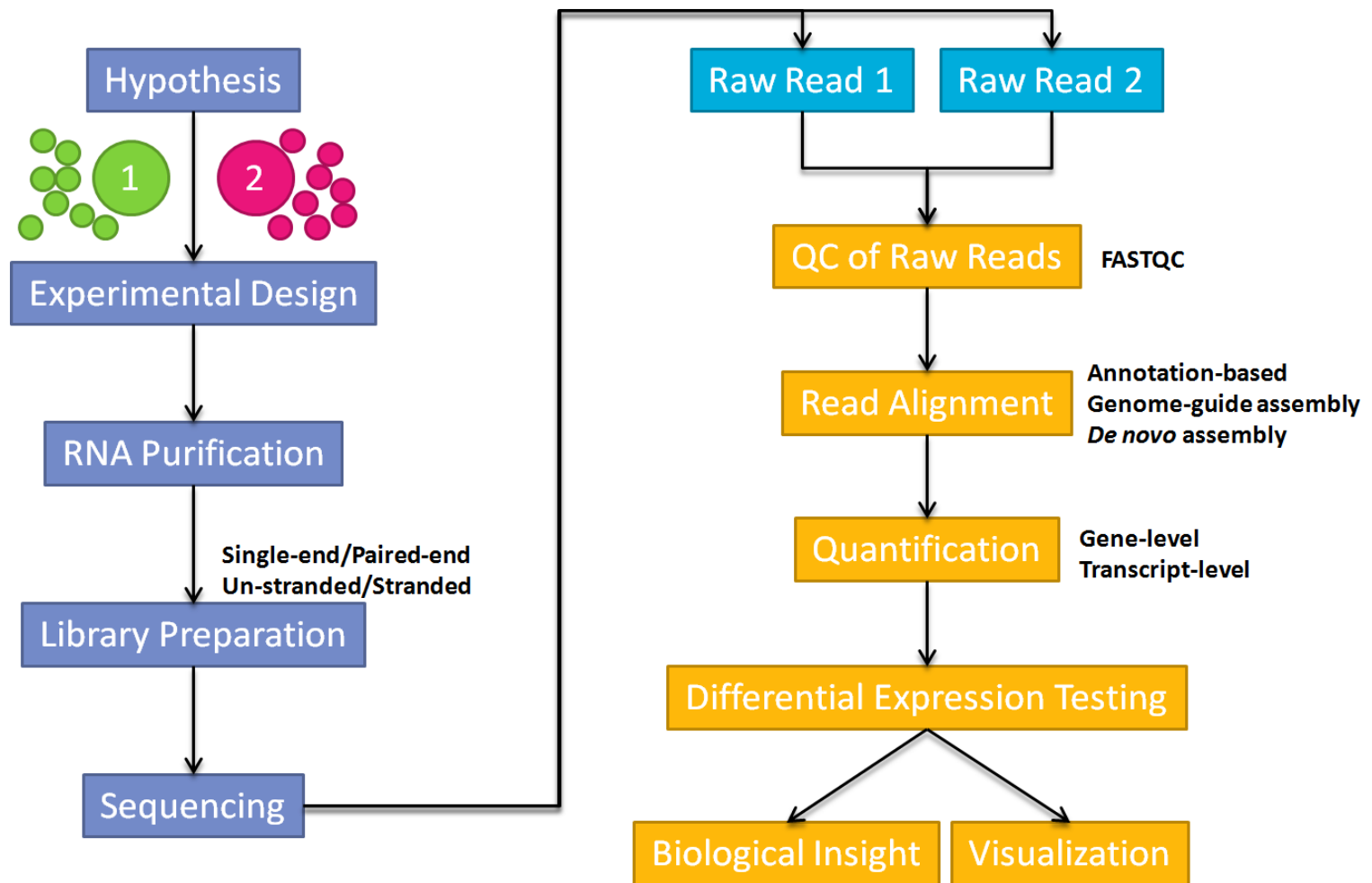
# Data Pipeline

---

“In computing, a **pipeline**, also known as a **data pipeline**, is a set of data processing elements connected in series, where the output of one element is the input of the next one”.

-- [https://en.wikipedia.org/wiki/Pipeline\\_\(computing\)](https://en.wikipedia.org/wiki/Pipeline_(computing))

# RNA-Seq Pipeline



-- [https://databeauty.com/figures/2016-09-13-RNA-seq-analysis/rna\\_seq\\_workflow.png](https://databeauty.com/figures/2016-09-13-RNA-seq-analysis/rna_seq_workflow.png)

# Continue RNA-Seq pipeline

---

Assume we have RNA-Seq data files:

```
ld32@login02 /home/ld32/myRNASeq$ ls  
sample1_1.fq sample1_2.fq sample2_1.fq  
sample2_2.fq sample3_1.fq sample3_2.fq  
sample4_1.fq sample4_2.fq
```



# RNA-Seq pipeline (single computer)

---

```
#!/bin/bash
for i in 1 2 3 4; do
    sp=sample${1}; r1=${sp}_1.fq; r2=${sp}_2.fq;

    qcCmd -input1 ${r1} -input2 ${r2} -output ${sp}.qc.txt

    alignCmd -cpu 4 -input1 ${r1} -input2 ${r2} -output ${sp}.bam

    countCmd -input ${sp}.bam -output ${sp}.cnt
done

mergeCmd -input *.cnt -output allCount.txt

statCmd -input allCount.txt -output finalResult.txt
```

# A simpler pipeline

---

```
for u in universityA.txt ; do
```

```
    grep -H John $u >> John.txt
```

```
    grep -H Nick $u >> Nick.txt
```

```
done
```

```
cat John.txt Nike.txt > all.txt
```

# Converting to Slurm pipeline

---

```
for u in universityA.txt; do
```

```
    job1ID=$(sbatch -p short -t 5 --wrap "grep -H John $u >> John.txt")
```

```
    job2ID=$(sbatch -p short -t 5 --wrap "grep -H Nick $u >> Nick.txt")
```

```
done
```

```
sbatch -p short -t 5 -d afterok:$job1ID:$job2ID --wrap "cat \
    John.txt Nike.txt > all.txt"
```

# Also keep log and send emails

---

```
job1ID=$(checkIfItsDone.sh ||  
sbatch -p short -t 5 --mem 2G --  
wrap "someCmd && touch  
job1.success; sendBetterEmail.sh;")
```

What are `checkIfItsDone.sh`? And `sendBetterEmail.sh`?

# A little more complex

---

```
for i in A B C E F; do
```

```
    u=university${i}.txt
```

```
    grep -H John $u >> John.txt
```

```
    grep -H Nick $u >> Nick.txt
```

```
done
```

```
cat John.txt Nike.txt > all.txt
```



# Can we automate it?

---

```
for i in A B C E F ; do
```

```
  u=university${i}.txt
```

```
  #@ Please submit this command as slurm job with 1G memory,
```

```
  #@ 5 minute run time and 1 CPU. And this job can start right away.
```

```
  grep -H John $u >> John.txt
```

```
  #@ Please submit this command as slurm job with 1G memory,
```

```
  #@ 5 minute run time and 1 CPU. And this job can start right away
```

```
  grep -H Nick $u >> Nick.txt
```

```
Done
```

```
#@ Please submit this command as slurm job with 2G memory,
```

```
#@ 5 minute run time and 1 CPU. And this job needs to wait for the other 10 jobs
```

```
cat John.txt Nike.txt > all.txt
```

# RCBio decorator

---

```
#@3,1.2,merge,,sbatch -p short -t 5 -c 1 --mem 1G  
cat John.txt Nick.txt >> all.txt
```

#@	→	This row is decorator for RCBio to parse
3	→	Step ID.
1.2	→	The step IDs this step depends on.
merge	→	Step name.
sbatch...	→	sbatch command to use

# Can we automate it?

---

```
for i in A B C E F ; do
```

```
u=university${i}.txt
```

```
#@1,0,findJohn,u,sbatch -p short -t 5 -c 1 --mem 1G
```

```
grep -H John $u >> John.txt
```

```
#@2,0,findNick,u,sbatch -p short -t 5 -c 1 --mem 1G
```

```
grep -H Nick $u >> Nick.txt
```

```
Done
```

```
#@3,1.2,merge,,sbatch -p short -t 5 -c 1 --mem 1G
```

```
cat John.txt Nike.txt > all.txt
```



# Practice session

Go through the wiki page and run a simple pipeline

<https://harvardmed.atlassian.net/wiki/spaces/O2/pages/1600651373/RC+workflows>



# Other solution 1: Cromwell using 'CWL' file

```
workflow myWorkflowName {
```

```
File my_ref  
File my_input  
String name
```

```
call task_A {
```

```
input: ref= my_ref, in= my_input, id= name
```

```
}
```

```
call task_B {
```

```
input: ref= my_ref, in= task_A.out
```

```
}
```

```
}
```

```
task task_A {
```

```
...
```

```
}
```

```
task task_B {
```

```
...
```

```
}
```

```
task task_A {
```

```
File ref  
File in  
String id
```

```
command {
```

```
do_stuff -R ${ref} -I ${in} -O ${id}.ext
```

```
}
```

```
runtime {
```

```
docker: "my_project/do_stuff:1.2.0"
```

```
}
```

```
output {
```

```
File out= "${id}.ext"
```

```
}
```

```
}
```

<https://bluewaters.ncsa.illinois.edu/liferay-content/document-library/content/Ruchi%20Munshi.%20Blue%20Waters%20webinar.%20Cromwell%20and%20WDL.pdf>

## Other solution 2: BCBio using 'yaml' and 'CWL' file

```
upload:
  dir: ../final
details:
  - files: [../input/NA12878-NGv3-LAB1360-A_1.fastq.gz, ..
    description: NA12878
    metadata:
      sex: female
    analysis: variant2
    genome_build: hg38
    algorithm:
      aligner: bwa
      variantcaller: gatk-haplotype
      validate: giab-NA12878/truth_small_variants.vcf.gz
      validate_regions: giab-NA12878/truth_regions.bed
      variant_regions: capture_regions/Exome-NGv3
```

`bcbio_nextgen.py config/NA12878-exome-methodcmp.yaml -n 8`

<https://bcbio-nextgen.readthedocs.io/en/latest/contents/intro.html#workflow>

# Other solution 3: NextFlow script

```
#!/usr/bin/env nextflow

params.in = "$baseDir/data/sample.fa"

/*
 * split a fasta file in multiple files
 */
process splitSequences {

    input:
    path 'input.fa' from params.in

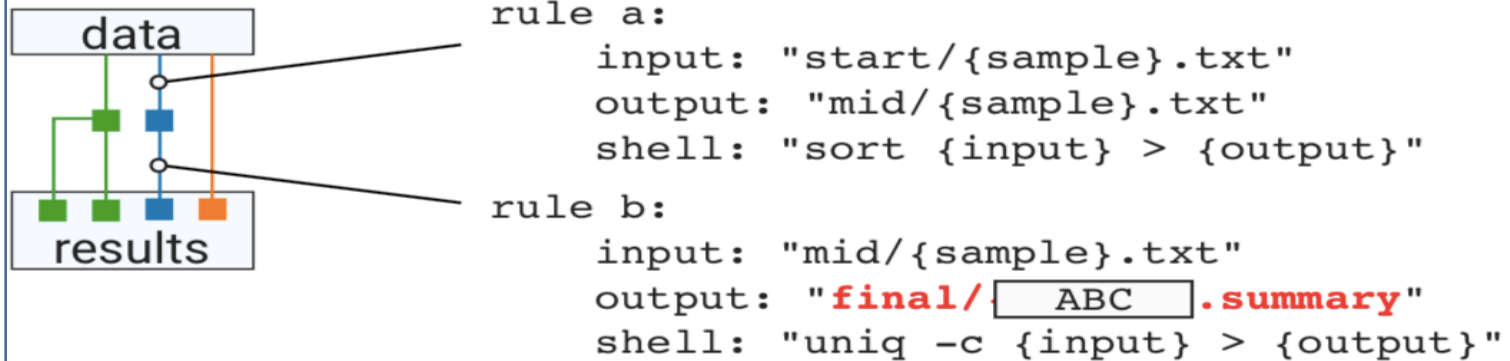
    output:
    path 'seq_*' into records

    """
    awk '/^>/{f="seq_"++d} {print > f}' < input.fa
    """
}
```

<https://www.nextflow.io/example1.html>

# Other solution 4: Snakemake using makefile

## Dependencies are implicit and 'backwards'



```
$ snakemake final/ABC.summary
```

[https://hpc.nih.gov/training/handouts/180221\\_snakemake\\_class\\_web.pdf](https://hpc.nih.gov/training/handouts/180221_snakemake_class_web.pdf)

# For more direction

---

- **Email:** [rchelp@hms.harvard.edu](mailto:rchelp@hms.harvard.edu)
- **O2 wiki:**  
<https://harvardmed.atlassian.net/wiki/spaces/O2/overview>
- **Website:** <http://rc.hms.harvard.edu>
- **RC Office Hours:** Wed 1-3p Gordon Hall 500
  - <https://rc.hms.harvard.edu/office-hours/> for Zoom web conferencing during remote work



# Please fill out the survey

---

- Accessible through the Harvard Training Portal
  - <https://trainingportal.harvard.edu/>
- Click on “Me” then “RCBio: Easy and Quick HPC Pipeline Builder and Runner”
- Scroll to “Evaluations” and click on the survey
- We appreciate any feedback or comments!