REPRODUCIBLE NGS WORKFLOWS WITH NEXTFLOW

Paolo Di Tommaso

NGS'17 - Workshop, 5 April 2017

nextflow

elixir SPAIN

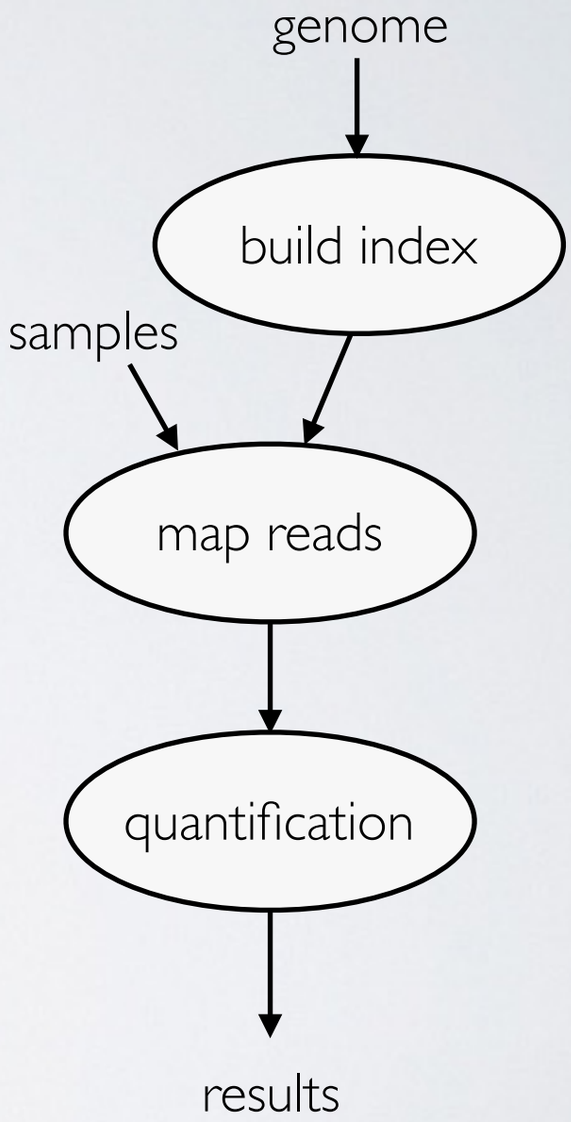
CRG Centre for Genomic Regulation

AGENDA

- Common problems with genomic pipelines
- *Coffee break*
- Quick overview of Nextflow framework
- How write a Nextflow pipeline
- Handle dependencies with Docker containers

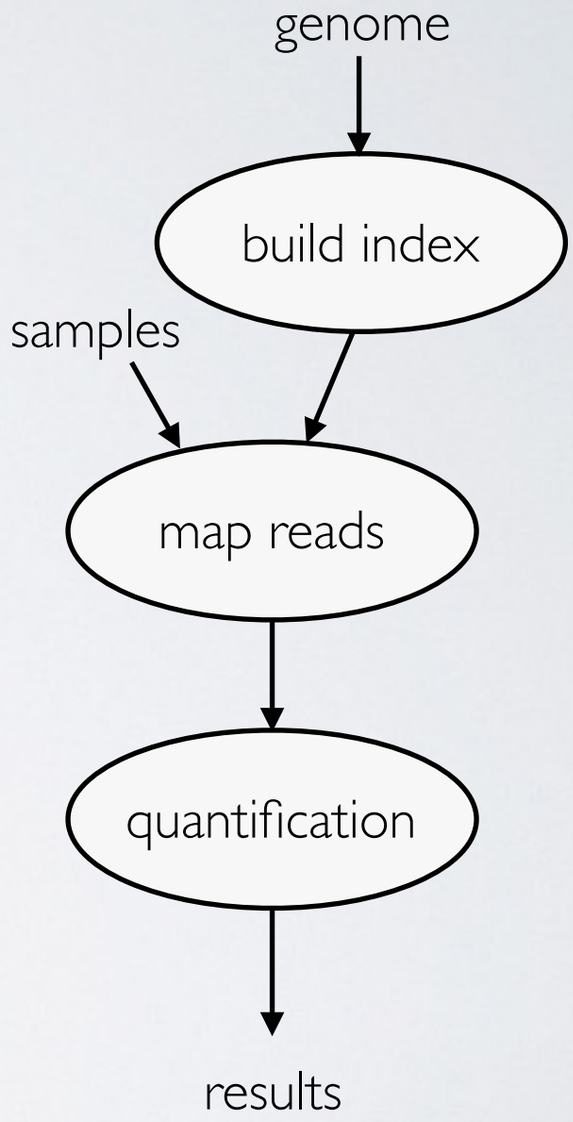
DEFINITION

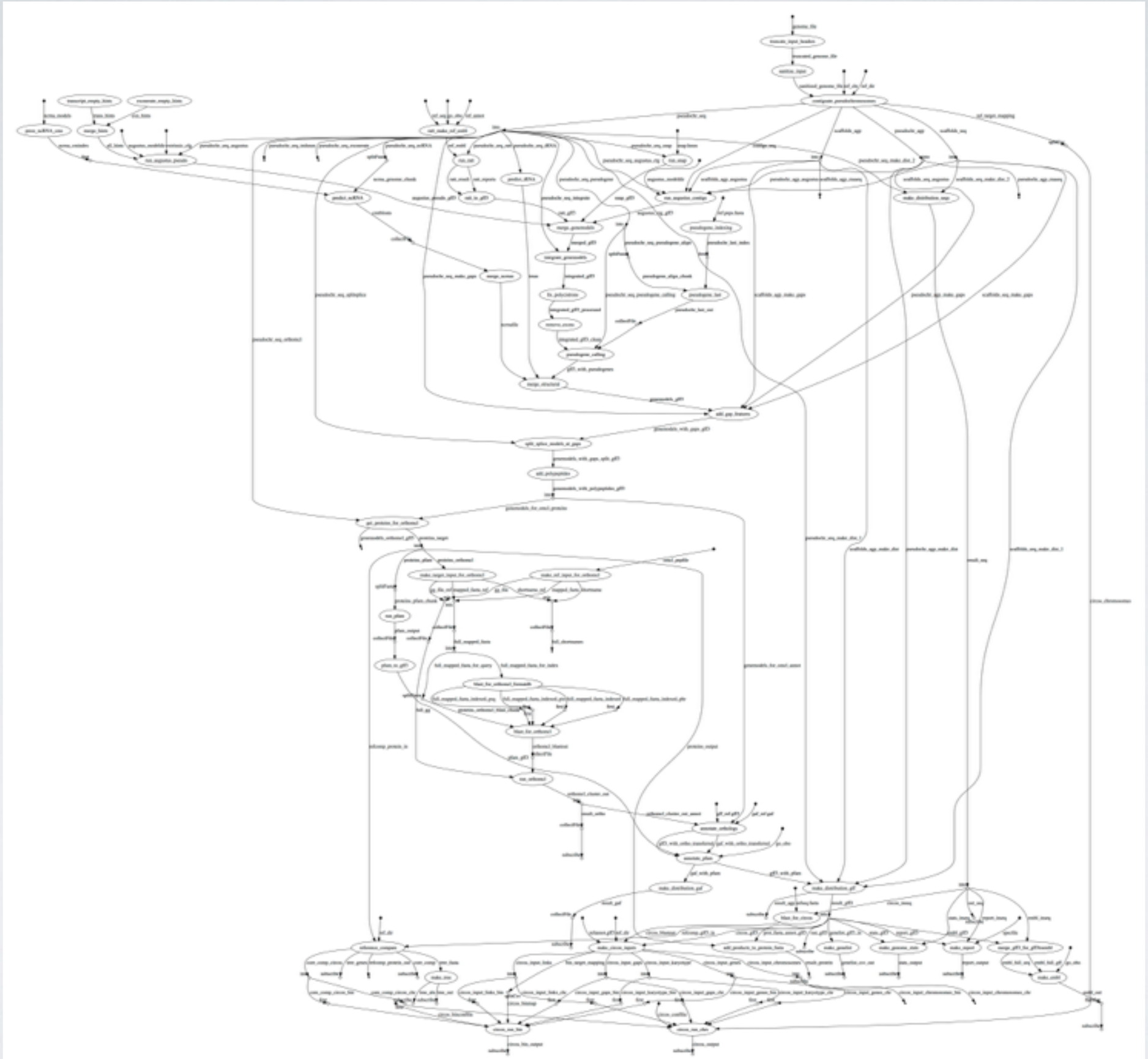
A sequence of steps or processes in which a piece of work passes from initiation to completion



EXAMPLE

```
#!/bin/bash
genome=$1
annot=$2
reads=$3

bowtie2-build ${genome} genome.index

tophat2 --GTF $annot genome.index $reads

cufflinks -q -G $annot accepted_hits.bam
```





* Companion parassite genome annotation pipeline, Steinbiss et al., DOI: 10.1093/nar/gkw292

COMPLEXITY

- Dozens of dependencies (binary tools, compilers, libraries, system tools, etc)

- Experimental nature of academic SW tends to be difficult to install, configure and deploy

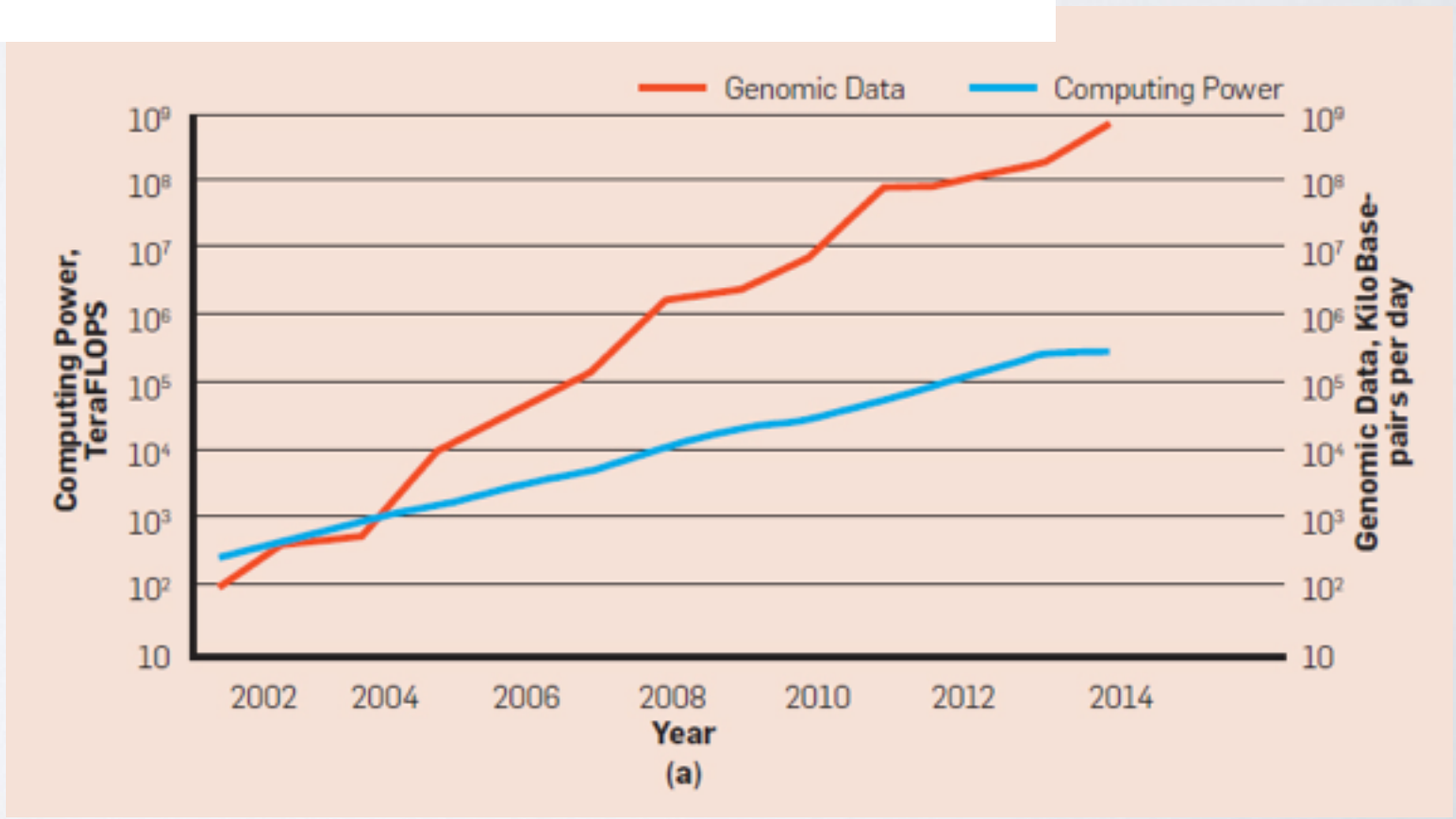
- Heterogeneous executing platforms and system architecture (laptop→supercomputer)

A TSUNAMI OF DATA

Four domains of big data in 2025

Data Phase	Astronomy	Twitter	YouTube	Genomics
Acquisition	25 zetta-bytes/year	0.5–15 billion tweets/year	500–900 million hours/year	1 zetta-bases/year
Storage	1 EB/year	1–17 PB/year	1–2 EB/year	2–40 EB/year
Analysis	In situ data reduction	Topic and sentiment mining	Limited requirements	Heterogeneous data and analysis
	Real-time processing	Metadata analysis		Variant calling, ~2 trillion central processing unit (CPU) hours
	Massive volumes			All-pairs genome alignments, ~10,000 trillion CPU hours
Distribution	Dedicated lines from antennae to server (600 TB/s)	Small units of distribution	Major component of modern user's bandwidth (10 MB/s)	Many small (10 MB/s) and fewer massive (10 TB/s) data movement

doi:10.1371/journal.pbio.1002195.t001



doi:10.1145/2957324

PARALLELISATION

- The spectacular increase of dataset size requires efficient parallel programming techniques

- Writing parallel applications is a complex and error prone task

- Many different parallelisation programming models, architecture and platforms: multi-cores, OpenMP, MPI, GPU, CUDA, Map/Reduce, Hadoop, Spark, etc.

PORTABILITY

- Ideally a pipeline should be able to run across different platforms (pc, cluster, hpc, cloud)

- In practice it's strictly tied to a specific platform or infrastructure

- This makes difficulty to share, re-use data analysis application and enable collaboration

REPRODUCIBILITY

- In-silico experiments are expected to be reproducible

- In practice it's very difficult to replicate the results of bioinformatics pipelines

- Acquire data, reconfigure the system requires advanced skills

OPEN ACCESS Freely available online

PLOS | ONE

Quantifying Reproducibility in Computational Biology: The Case of the Tuberculosis Drugome

Daniel Garijo[1], Sarah Kinnings[2], Li Xie[3], Lei Xie[4], Yinliang Zhang[5], Philip E. Bourne[3]*, Yolanda Gil[6]*

1 Ontology Engineering Group, Facultad de Informática, Universidad Politécnica de Madrid, Madrid, Spain, **2** Department of Chemistry and Biochemistry, University of California San Diego, La Jolla, California, United States of America, **3** Skaggs School of Pharmacy and Pharmaceutical Sciences, University of California San Diego, La Jolla, California, United States of America, **4** Department of Computer Science, Hunter College, The City University of New York, New York, New York, United States of America, **5** School of Life Sciences, University of Science and Technology of China, Hefei, Anhui, China, **6** Information Sciences Institute and Department of Computer Science, University of Southern California, LosAngeles, California, United States of America

To replicate the result of a typical computational biology paper requires 280 hours.

≈ 1.7 months!

WHAT IF AN ANALYSIS IS NOT REPLICABLE AT ALL

SMALL CHANGES IN THE COMPUTATIONAL ENVIRONMENT HAVE UNPREDICTABLE CONSEQUENCES

Comparison of the Companion pipeline annotation of *Leishmania infantum* genome executed across different platforms *

Platform	Amazon Linux	Debian Linux	Mac OSX
Number of chromosomes	36	36	36
Overall length (bp)	32,032,223	32,032,223	32,032,223
Number of genes	7,781	7,783	7,771
Gene density	236.64	236.64	236.32
Number of coding genes	7,580	7,580	7570
Average coding length (bp)	1,764	1,764	1,762
Number of genes with multiple CDS	113	113	111
Number of genes with known function	4,147	4,147	4,142
Number of t-RNAs	88	90	88

* Di Tommaso P, et al., *Nextflow enables computational reproducibility*, Nature Biotech, 2017 (publication pending)

IT'S NOT POSSIBLE TO IDENTICALLY REPLICATE THE SAME EXECUTION ENVIRONMENT IN TWO DIFFERENT SYSTEMS

THE SAME APPLICATION
PRODUCES
DIFFERENT RESULTS
IN
DIFFERENT SYSTEMS!

NEXTFLOW

- A framework for computational workflows
- It provides a DSL to simplify the writing complex parallel workflows
- Enables transparent deployment on multiple platforms
- Built-in integration with containers technology

- Easy installation
- Use existing tools an scripts
- Implicit parallelisation
- Simplified deployment
- Lightweight, self-contained

Easiness

nextflow

HPC

clusters and cloud

UNIVA

GRID ENGINE

slurm
workload manager

amazon
web services

Platform
Computing
an IBM Company

apache
Ignite

kubernetes

BSC

Reproducibility

containers

versioning

Git

GitHub

GitLab

Bitbucket

WHO IS USING NEXTFLOW?

UPPSALA UNIVERSITET

Karolinska Institutet

NATIONAL MARROW DONOR PROGRAM®

BERKELEY LAB

JGI JOINT GENOME INSTITUTE DEPARTMENT OF ENERGY

International Agency Research on Cancer
World Health Organization

SciLifeLab

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

UT Southwestern Medical Center

UCSF

UiO : University of Oslo

Universität Bielefeld

Weill Cornell Medical College

Institut Pasteur

CRG Centre for Genomic Regulation

UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Colorado State University

Curtin University

bina

wellcome trust sanger institute

PAWSEY supercomputing centre

BENAROYA RESEARCH INSTITUTE Virginia Mason

SYNTHETIC GENOMICS

Roche

HOW IT WORKS

The workflow

1. Mapping to the reference

The first major difference relative to the DNAseq Best Practices is the mapping step. For DNA-seq, we recommend BWA. For RNA-seq, we evaluated all the major software packages that are specialized in RNAseq alignment, and we found that we were able to achieve the highest sensitivity to both SNPs and, importantly, indels, using STAR aligner. Specifically, we use the STAR 2-pass method which was described in a recent publication (see page 43 of the Supplemental text of the Pär G Engström et al. paper referenced below for full protocol details -- we used the suggested protocol with the default parameters). In brief, in the STAR 2-pass approach, splice junctions detected in a first alignment run are used to guide the final alignment.

Here is a walkthrough of the STAR 2-pass alignment steps:

1) STAR uses genome index files that must be saved in unique directories. The human genome index was built from the FASTA file hg19.fa as follows:

```
genomeDir=/path/to/hg19
mkdir $genomeDir
STAR --runMode genomeGenerate --genomeDir $genomeDir --genomeFastaFiles hg19.fa\  --runThreadN <n>
```

2) Alignment jobs were executed as follows:

```
runDir=/path/to/1pass
mkdir $runDir
cd $runDir
STAR --genomeDir $genomeDir --readFilesIn mate1.fq mate2.fq --runThreadN <n>
```

3) For the 2-pass STAR, a new index is then created using splice junction information contained in the file SJ.out.tab from the first pass:

```
genomeDir=/path/to/hg19_2pass
mkdir $genomeDir
STAR --runMode genomeGenerate --genomeDir $genomeDir --genomeFastaFiles hg19.fa \
    --sjdbFileChrStartEnd /path/to/1pass/SJ.out.tab --sjdbOverhang 75 --runThreadN <n>
```

4) The resulting index is then used to produce the final alignments as follows:

```
runDir=/path/to/2pass
mkdir $runDir
```

HOW IT WORKS

```
mkdir genome_dir
STAR --runMode genomeGenerate \
     --genomeDir genome_dir \
     --genomeFastaFiles hg19.fa
```

```
STAR --genomeDir genome_dir \
     --readFilesIn mate1.fq mate2.fq

:

java -jar picard.jar MarkDuplicates \
     I=rg_added_sorted.bam \
     O=dedupped.bam \
     CREATE_INDEX=true
:
```

HOW IT WORKS

```
mkdir genome_dir
STAR --runMode genomeGenerate \
     --genomeDir genome_dir \
     --genomeFastaFiles hg19.fa
```

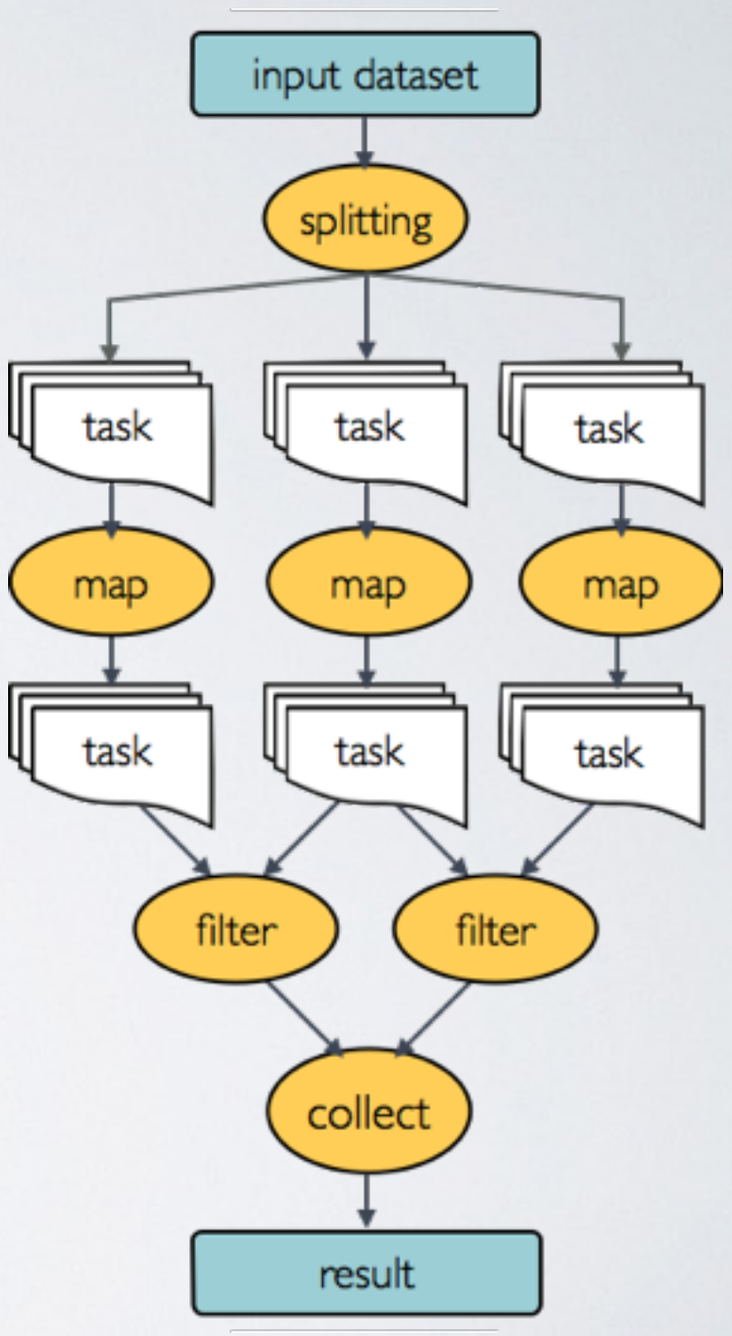
```
STAR --genomeDir genome_dir \
     --readFilesIn mate1.fq mate2.fq

:

java -jar picard.jar MarkDuplicates \
     I=rg_added_sorted.bam \
     O=dedupped.bam \
     CREATE_INDEX=true
:
```

```
mkdir genome_dir
STAR --runMode genomeGenerate \
     --genomeDir genome_dir \
     --genomeFastaFiles hg19.fa
```

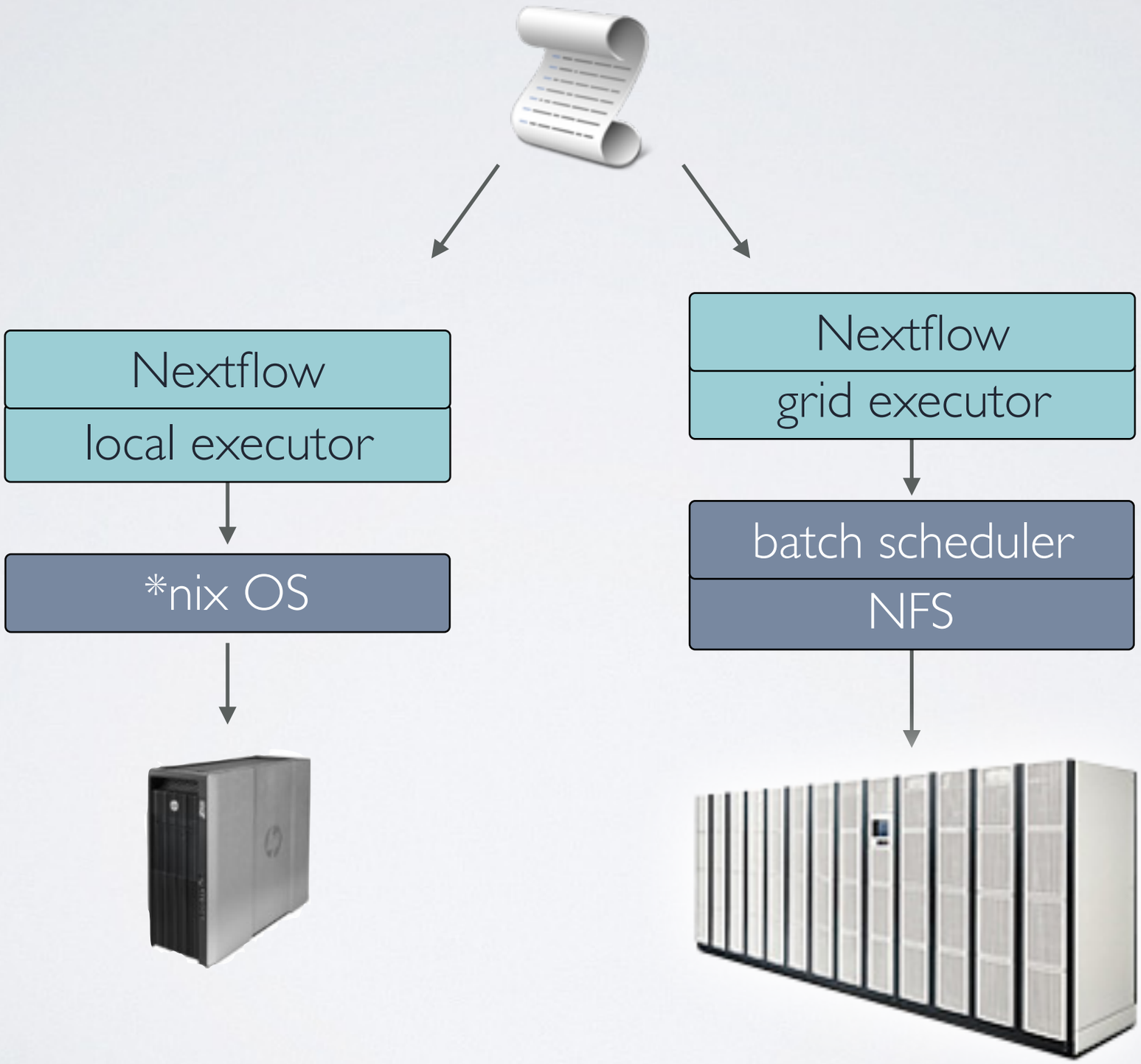
```
process makeIndex {

  input:
    file "hg19.fa" from genome_ch

  output:
    file "genome_dir" into index_ch

  script:
  """
      mkdir genome_dir
      STAR --runMode genomeGenerate \
           --genomeDir genome_dir \
           --genomeFastaFiles hg19.fa


  """
}
```

DATAFLOW

- Declarative computational model for concurrent processes
- Processes wait for data, when an input set is ready the process is executed
- They communicate by using dataflow variables i.e. async FIFO queues called channels
- Parallelisation and tasks dependencies are implicitly defined by process in/out declarations



PORTABILITY



SUPPORTED PLATFORMS

UNIVA

GRID ENGINE

Platform Computing
an IBM Company

slurm
workload manager

PBS Works™

kubernetes

HTCondor
High Throughput Computing

apache Ignite

amazon web services™

CONFIGURATION FILE

```
process {
  executor = 'sge'
  queue = 'cn-el6'
  memory = '10GB'
  cpus = 8
  time = '2h'
}
```

NEXTFLOW DSL

- Scripting language for computational pipelines
- Multi-paradigm: imperative + declarative
- Extension of Java/Groovy programming lang

MAIN ABSTRACTIONS

- Processes: run any piece of script
- Channels: unidirectional async queues that allows the processes to comunicate
- Operators: transform channels content

PROCESS DEFINITION

```
process makeIndex {
  cpus 4
  memory 8.GB
  input:
    file "hg19.fa" from genome_ch
  output:
    file "genome_dir" into index_ch
  script:
  """
    STAR --this --that hg19.fa
    samtools index genome.bam ...
  """
}
```

INPUTS

```
process step_x {

  input:
    file "genome.fa" from ..

  script:
  """
    STAR --genomeFastaFiles genome.fa --other ..
  """
}
```

INPUTS

```
process step_x {

  input:
    file genome from ..

  script:
  """
    STAR --genomeFastaFiles $genome --other ..
  """
}
```

INPUTS

```
process step_x {

  input:
    file genome_dir from ..
    file reads      from ..

  script:
  """
    STAR --genomeDir $genome_dir --readFilesIn $reads
  """
}
```

INPUTS

```
process step_x {

  input:
    file genome_dir from ..
    file reads      from ..
    val  sample_id  from ..

  script:
  """
    STAR --genomeDir $genome_dir \
         --readFilesIn $reads \
         --other $sample_id
  """

}
```

INPUTS

```
process step_x {

  input:
    file genome_dir from ..
    set  sample_id, file(reads) from ..

  script:
  """
    STAR --genomeDir $genome_dir \
         --readFilesIn $reads \
         --other $pair_id
  """

}
```

OUTPUTS

```
process step_x {

  input:
    file "genome.fa" from ..
  output:
    file "genome.dict" into ..


  script:
  """
    PICARD CreateSequenceDictionary \
           R= genome.fa O= genome.dict
  """

}
```

OUTPUTS

```
process step_x {

  input:
    file genome from ..
  output:
    file "${genome}.dict" into ..

  script:
  """
    PICARD CreateSequenceDictionary \
           R= $genome O= ${genome}.dict
  """

}
```

OUTPUTS

```
process step_x {

  input:
    file "genome.bam" from ..
  output:
    file "split.bam" into ..
     file "split.bai" into ..


   script:
   """
     GATK SplitNCigarReads -i genome.bam -o split.bam ..
   """


}
```

OUTPUTS

```
process step_x {

  input:
    file "genome.bam" from ..
  output:
    file "split.*" into ..

  script:
  """
    GATK SplitNCigarReads -i genome.bam -o split.bam ..
  """

}
```

OUTPUTS

```
process step_x {

  input:
    file "genome.bam" from ..
  output:
    file "split.{bam,bai}" into ..


  script:
  """
    GATK SplitNCigarReads -i genome.bam -o split.bam ..
  """

}
```

OUTPUTS

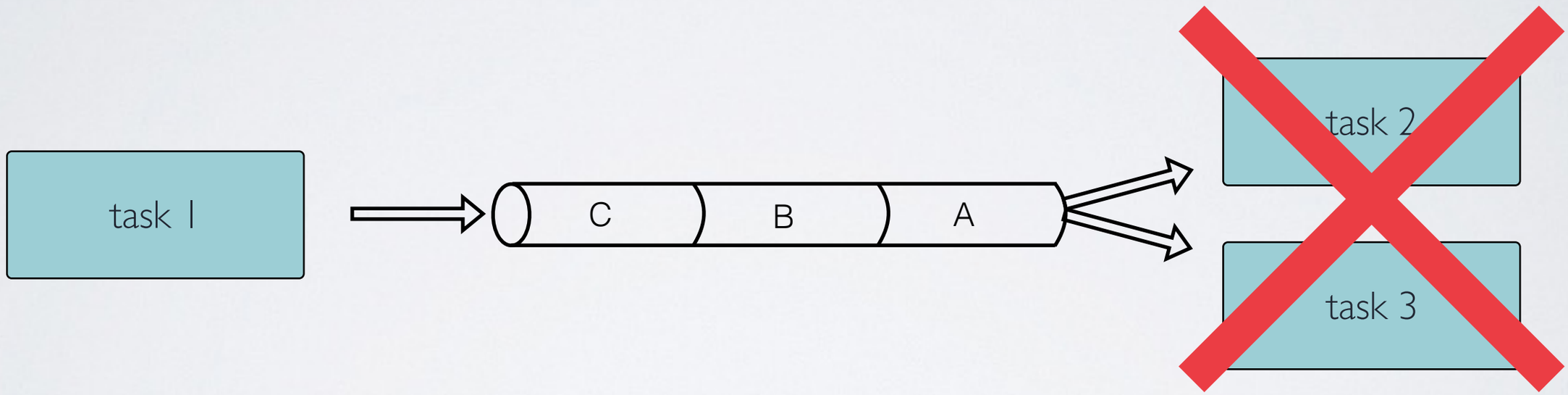
```
process step_x {

  input:
    file "genome.bam" from ..
  output:
    set file("split.bam"), file("split.bai") into ..

  script:
  """
    GATK SplitNCigarReads -i genome.bam -o split.bam ..
  """

}
```

CHANNELS

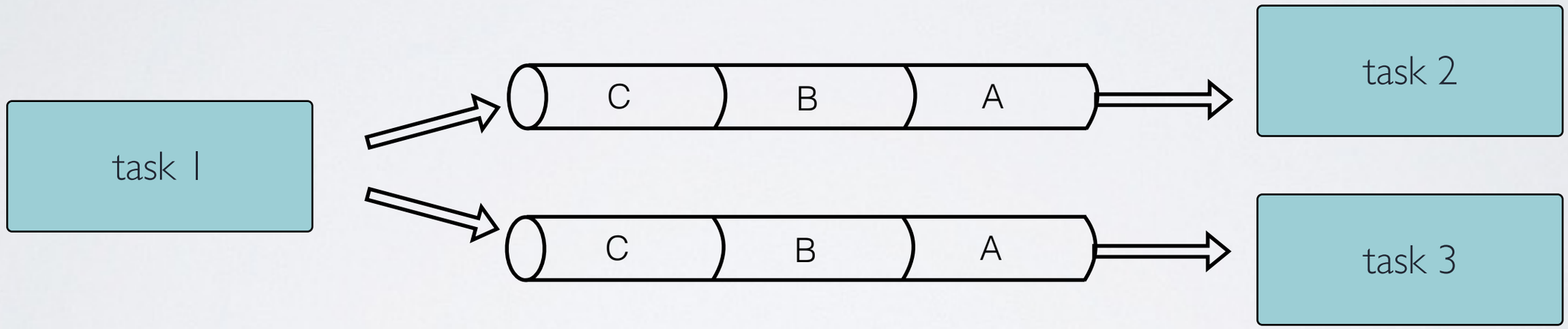
- A channel is a asynchronous unidirectional FIFO queue
- It connects two processes/operators
- Write operations is NOT blocking
- Read operation is blocking
- Once an item is read is removed from the queue

CHANNELS

CHANNELS



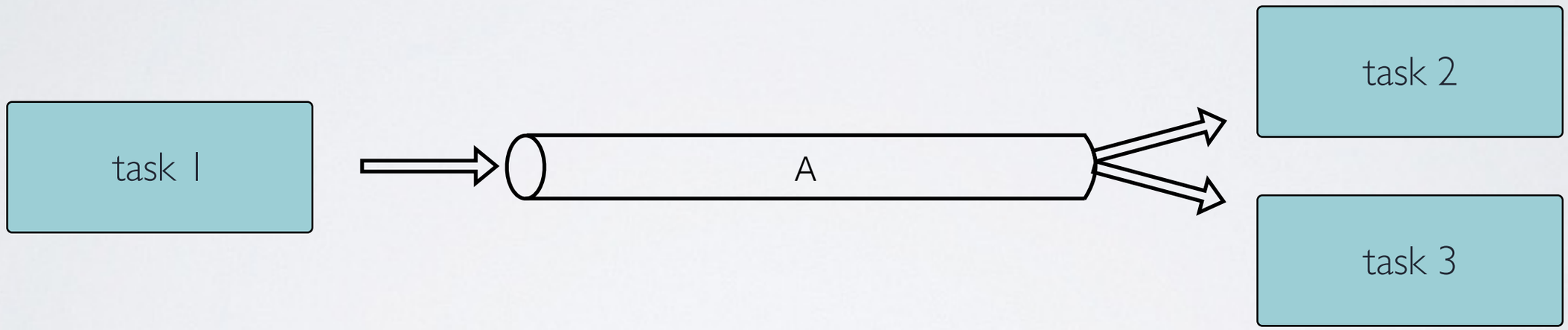
CHANNELS



VALUE CHANNELS

- Allow only one value to be written (bound)
- Read operation does not consume it
- Allow multiple consumers

VALUE CHANNELS



HOW CREATE CHANNELS

```
process step_x {

  output:
    file "name.txt" into foo_ch


  script:
  """
    touch name.txt
  """
}
```

HOW CREATE CHANNELS

```
process step_x {

  output:
    file "name.txt" into (foo_ch, bar_ch)

  script:
  """
    touch name.txt
  """
}
```

CHANNEL FACTORIES

```
my_channel = Channel.create()

some_items = Channel.from(10, 20, 30, ..)

single_file = Channel.fromPath('some/file/name')

more_files = Channel.fromPath('some/data/path/*')

pair_files = Channel.fromFilePairs('some/data/path/*_{1,2}.fq')
```

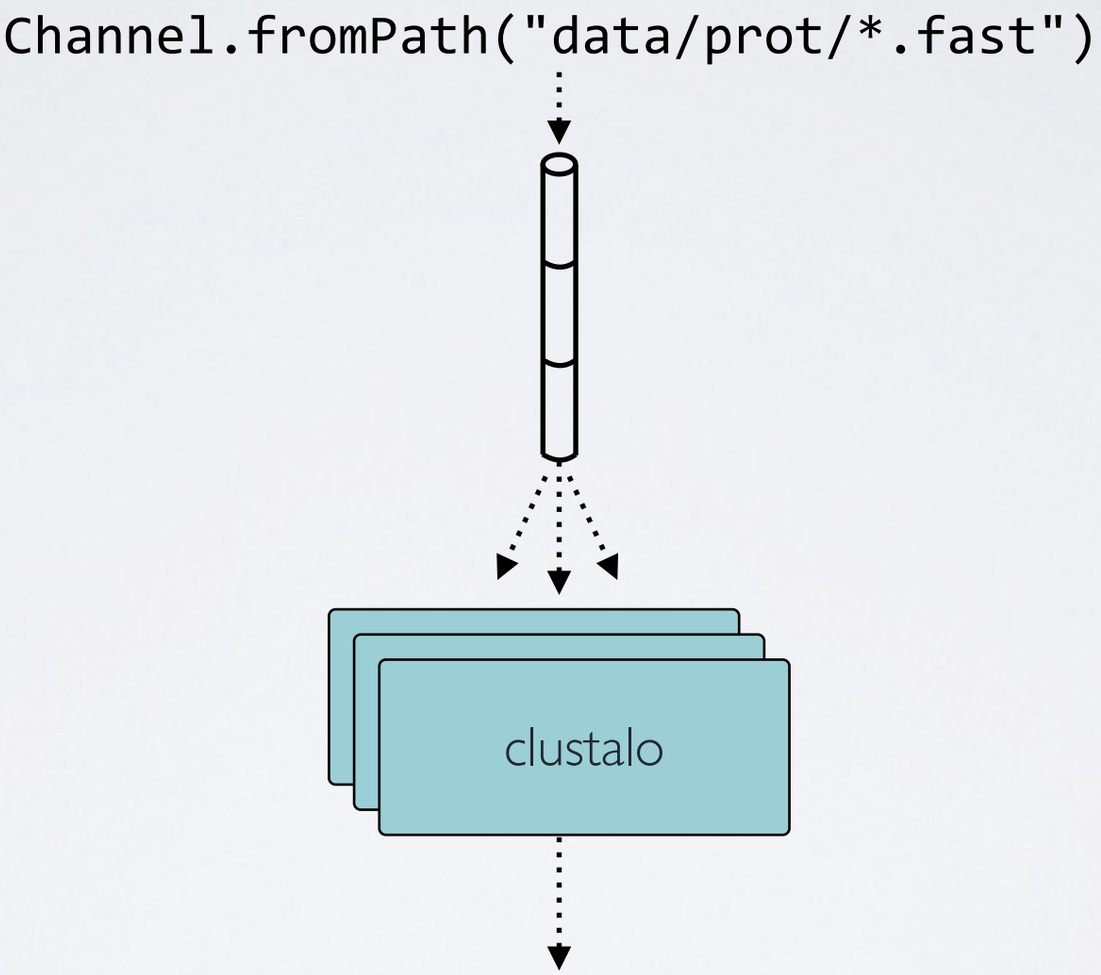
HANDLING PARALLELISM WITH DATAFLOW

```
sequences = file('data/prot/seq.fasta')


process align {

    input:
      file fasta from sequences
    output:
      file 'result.aln' into alignments

    """
     clustalo -i $fasta -o result.aln
    """
}


alignments.println { it.fileName }
```

```
sequences = Channel.fromPath('data/prot/seq.fasta')


process align {

    input:
      file fasta from sequences
    output:
      file 'result.aln' into alignments

    """
     clustalo -i $fasta -o result.aln
    """
}


alignments.println { it.fileName }
```

```
sequences = Channel.fromPath('data/prot/*.fasta')


process align {

    input:
      file fasta from sequences
    output:
      file 'result.aln' into alignments

    """
     clustalo -i $fasta -o result.aln
    """
}


alignments.println { it.fileName }
```

IMPLICIT PARALLELISM



DEMO

HANDLING FILE PAIRS

```
Channel.fromFilePairs("*_{1,2}.fq")
```

gut_1.fq
gut_2.fq
lung_1.fq
lung_2.fq
liver_1.fq
liver_2.fq
annotation.gtf
genome.fasta

(gut, [gut_1.fq, gut_2.fq])

(lung, [lung_1.fq, lung_2.fq])

(liver, [liver_1.fq, liver_2.fq])

(gut, [gut_1.fq, gut_2.fq])

(lung, [lung_1.fq, lung_2.fq])

(liver, [liver_1.fq, liver_2.fq])

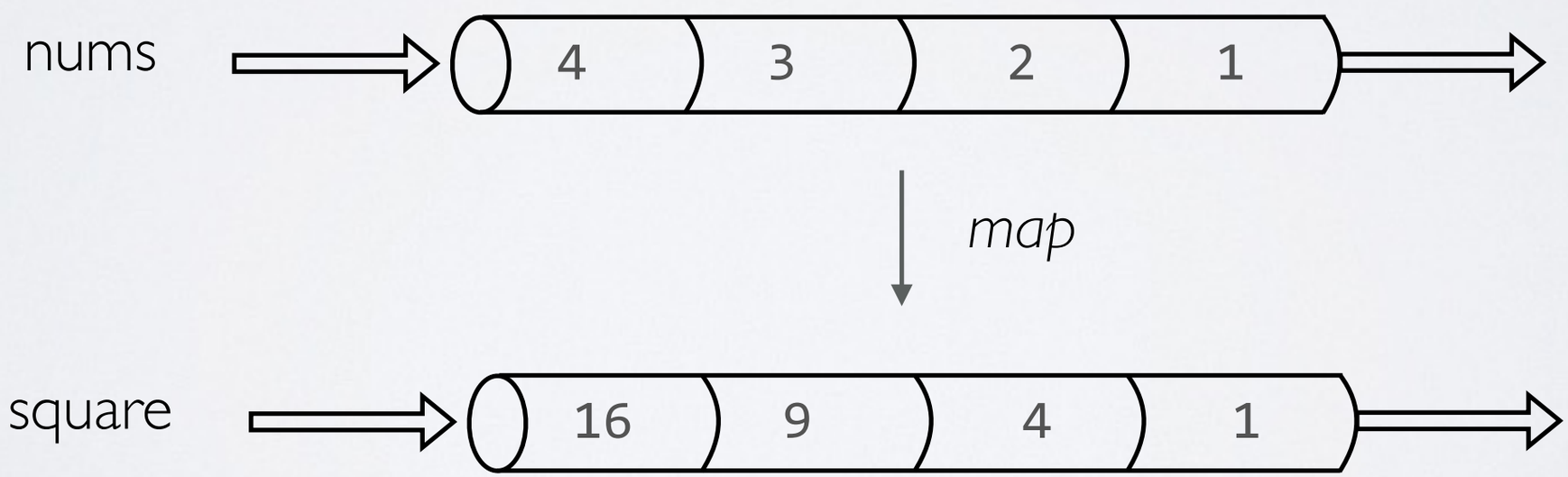
```
process mapping {
    input:
     file index from genome_index
     set pair_id, file(reads) from read_pairs

    output:
     set pair_id, "accepted_hits.bam" into bam

    """
     tophat2 $index $reads
     mv tophat_out/accepted_hits.bam .
    """
}
```

OPERATORS

- Functions applied to channels
- Transform channels content
- Can be used also to filter, fork and combine channels
- Operators can be chained to implement custom behaviours

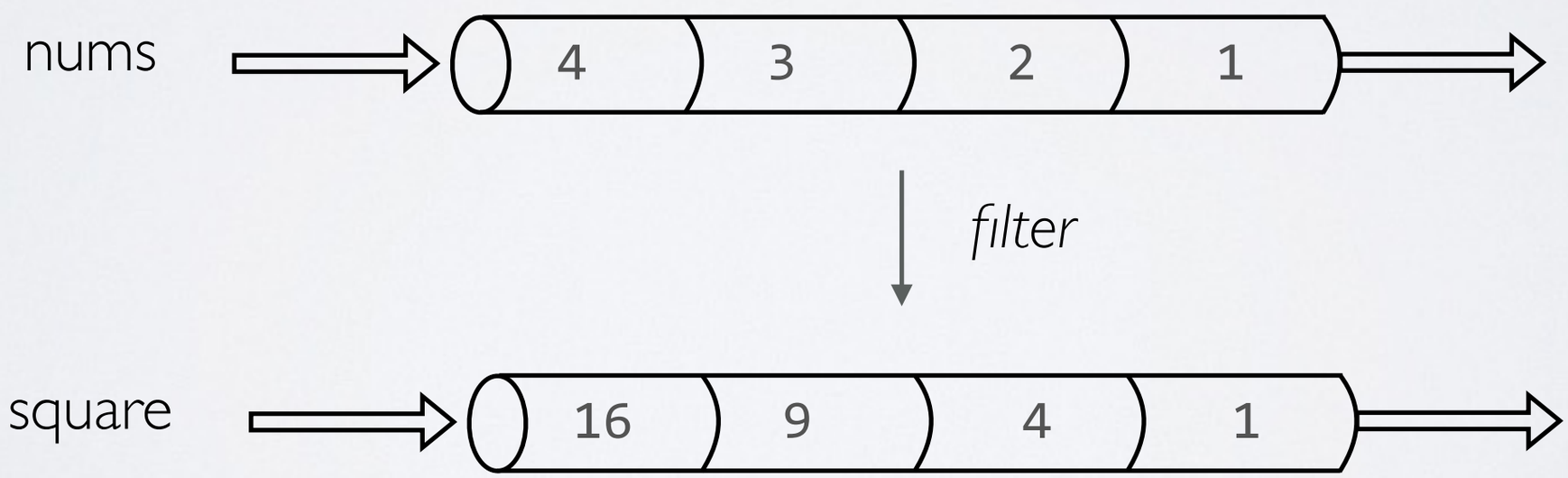
OPERATORS

```
nums = Channel.from(1,2,3,4)
square = nums.map { it * it }
```



OPERATORS

```
nums = Channel.from(1,2,3,4)
square = nums.filter { it > 2  }
```



OPERATORS CHAINING

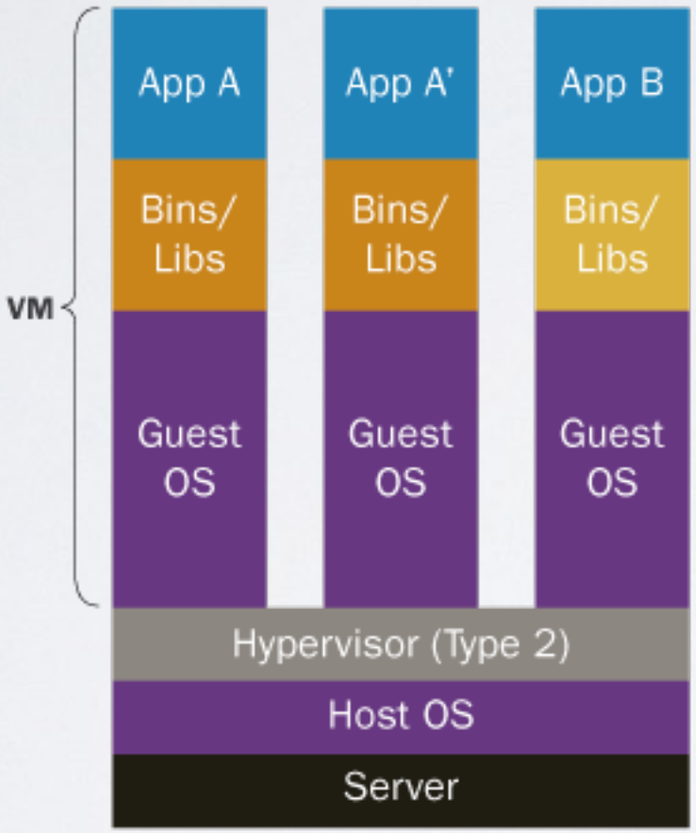
```
Channel.fromPath('misc/*.fa')
       .splitFasta( record: [id: true, seqString: true ])
       .filter { record -> record.id =~ /^ENST0.*/ }
       .println { record -> "$record.id - $record.seqString" }
```

```
// it prints
ENST0001 - NLFVALYDFVASGDNTLSITKGEKLRVLGYNHNGEWCEAQTKNGQGWVPS
ENST0005 - KGVIYALWDYEPQNDDELPMKEGDCMTIIHREDEDEIEWWWARLNDKEGY
ENST0007 - GYQYRALYDYKKEREEDIDLHLGDILTVNKGSLVALGFSDGQEARPEEIG
..
```

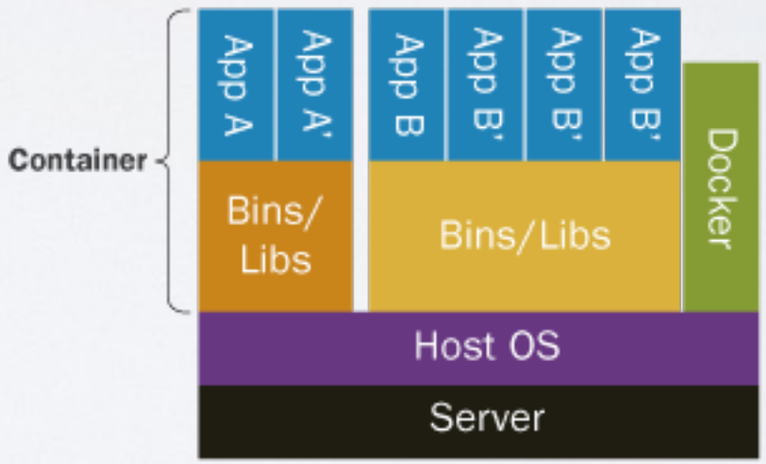
MANAGING DEPENDENCIES WITH CONTAINERS

CONTAINERS ARE THE THIRD BIG WAVE IN VIRTUALISATION TECHNOLOGY

VM vs CONTAINER



Containers are isolated, but share OS and, where appropriate, bins/libraries



CONTAINER vs VM

- Lighter: MB vs GB
- Faster startup: ms/secs vs minutes
- Virtualise a process/application instead of a OS/Hardware
- Immutable: don't change over time, thus guarantee replicability over executions.
- Composable: the output of one container is directly consumable as input by another container.
- Transparent: they are created with a well defined automated procedure.

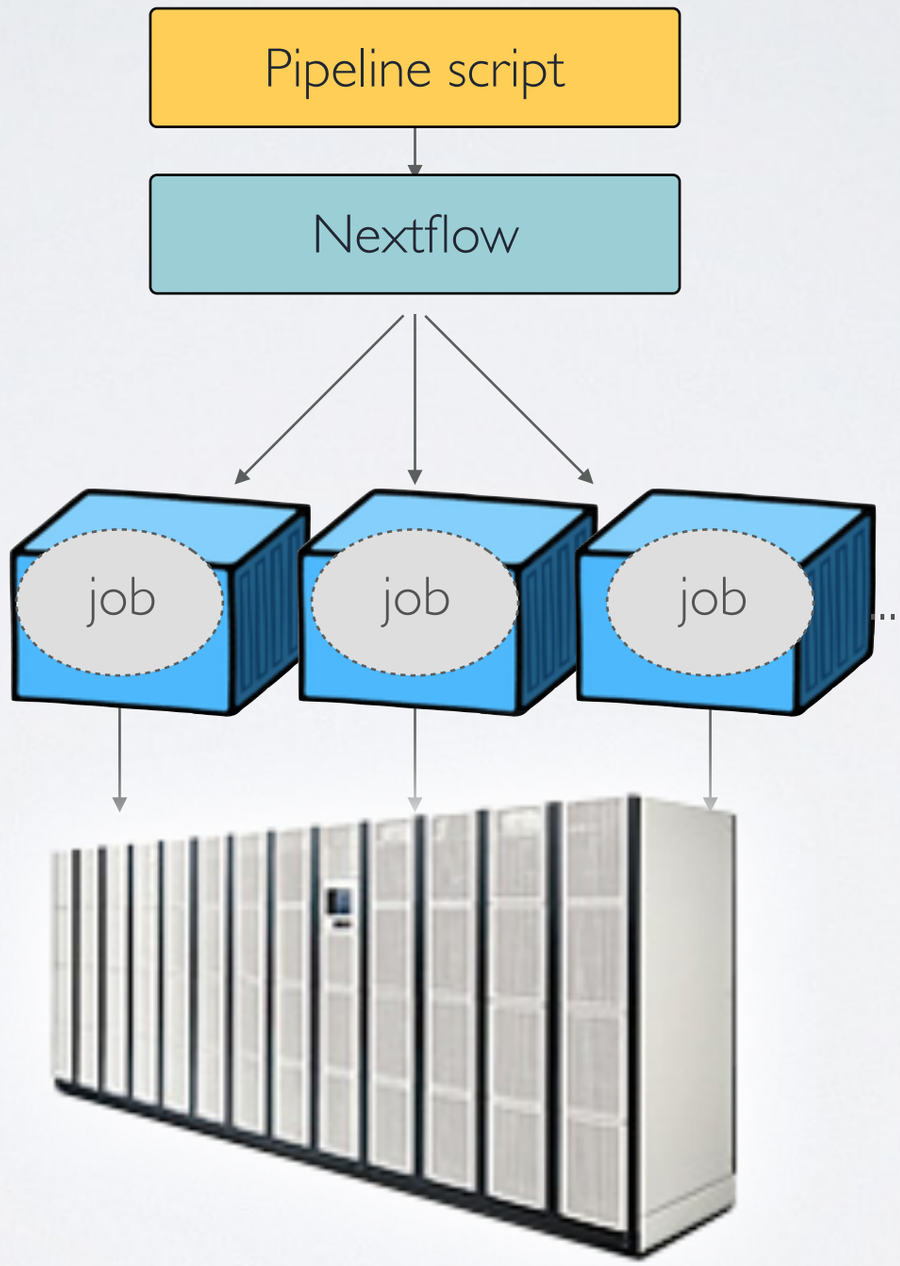
WHY THEY ARE USEFUL

- Allows you to create a ready-to-run package with all software dependencies in your pipelines
- Just one dependencies instead of dozens
- Docker is the most popular implementation

DOCKER DEMO

https://goo.gl/d2g1vl

NEXTFLOW + DOCKER



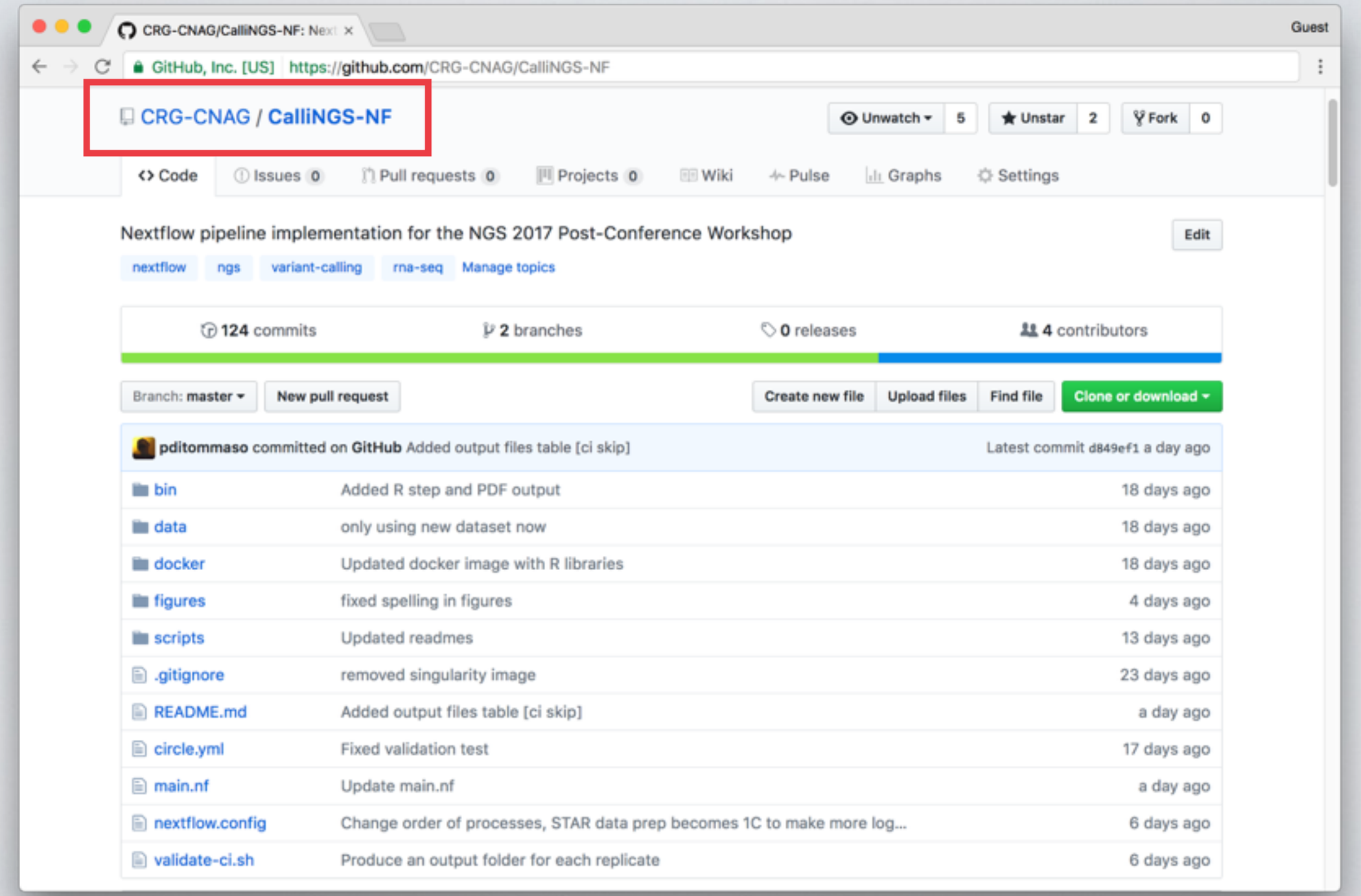
CONFIGURATION FILE

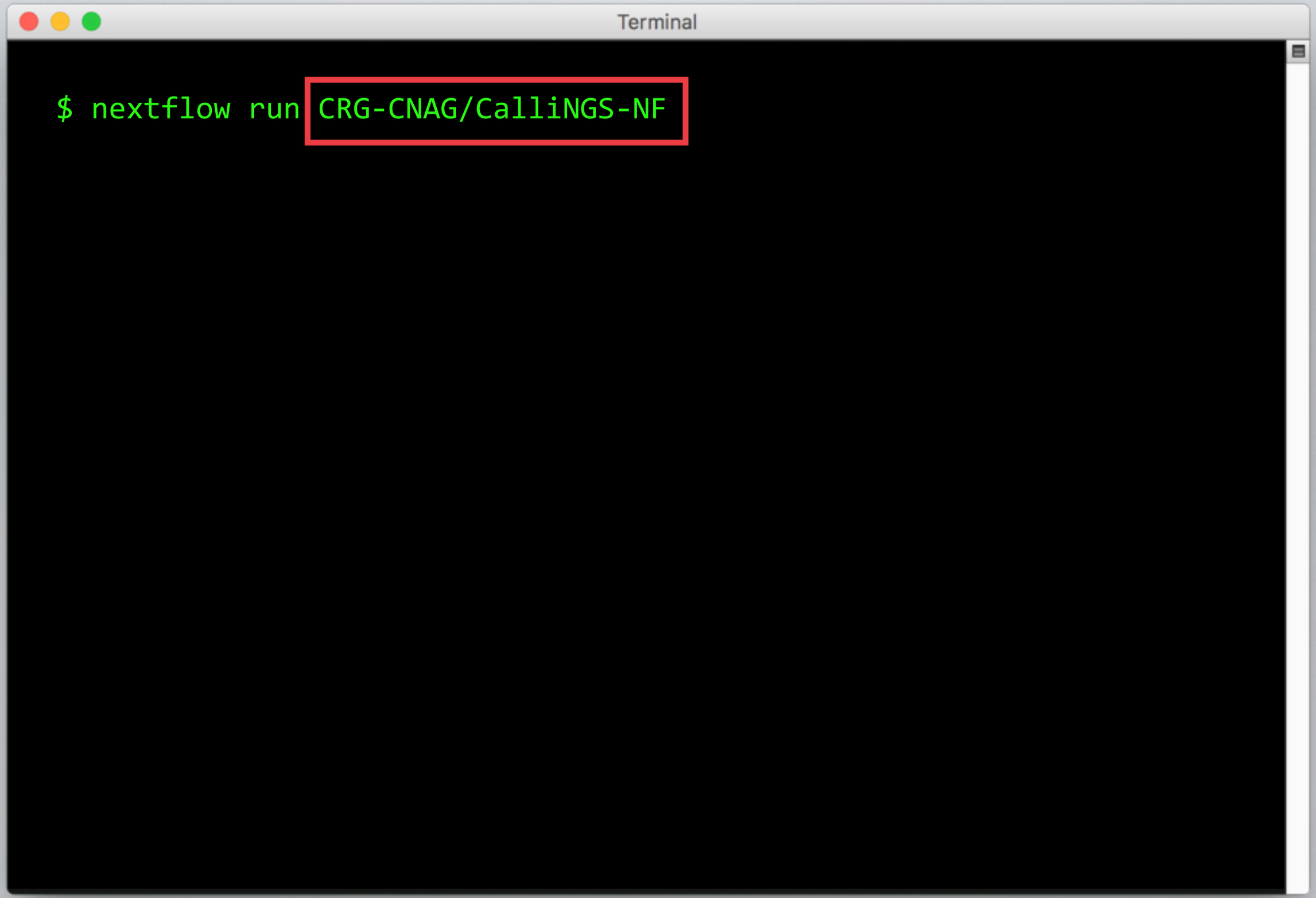
```
process {
  container = 'my-image'
  executor = 'sge'
  queue = 'cn-el6'
  memory = '10GB'
  cpus = 8
  time = '2h'
}
```

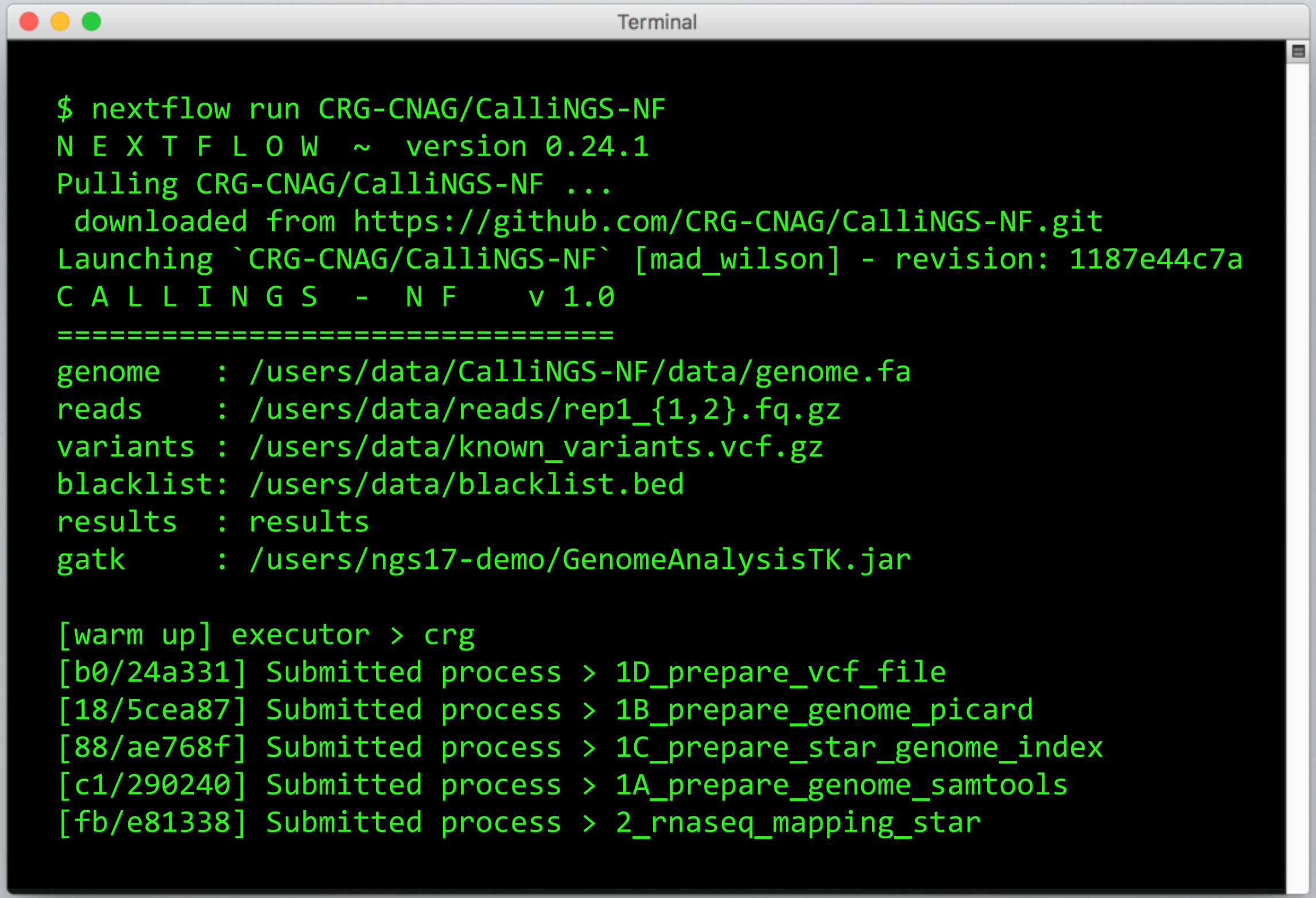
Q&A

PIPELINE SHARING & DEPLOYMENT

GITHUB SUPPORT



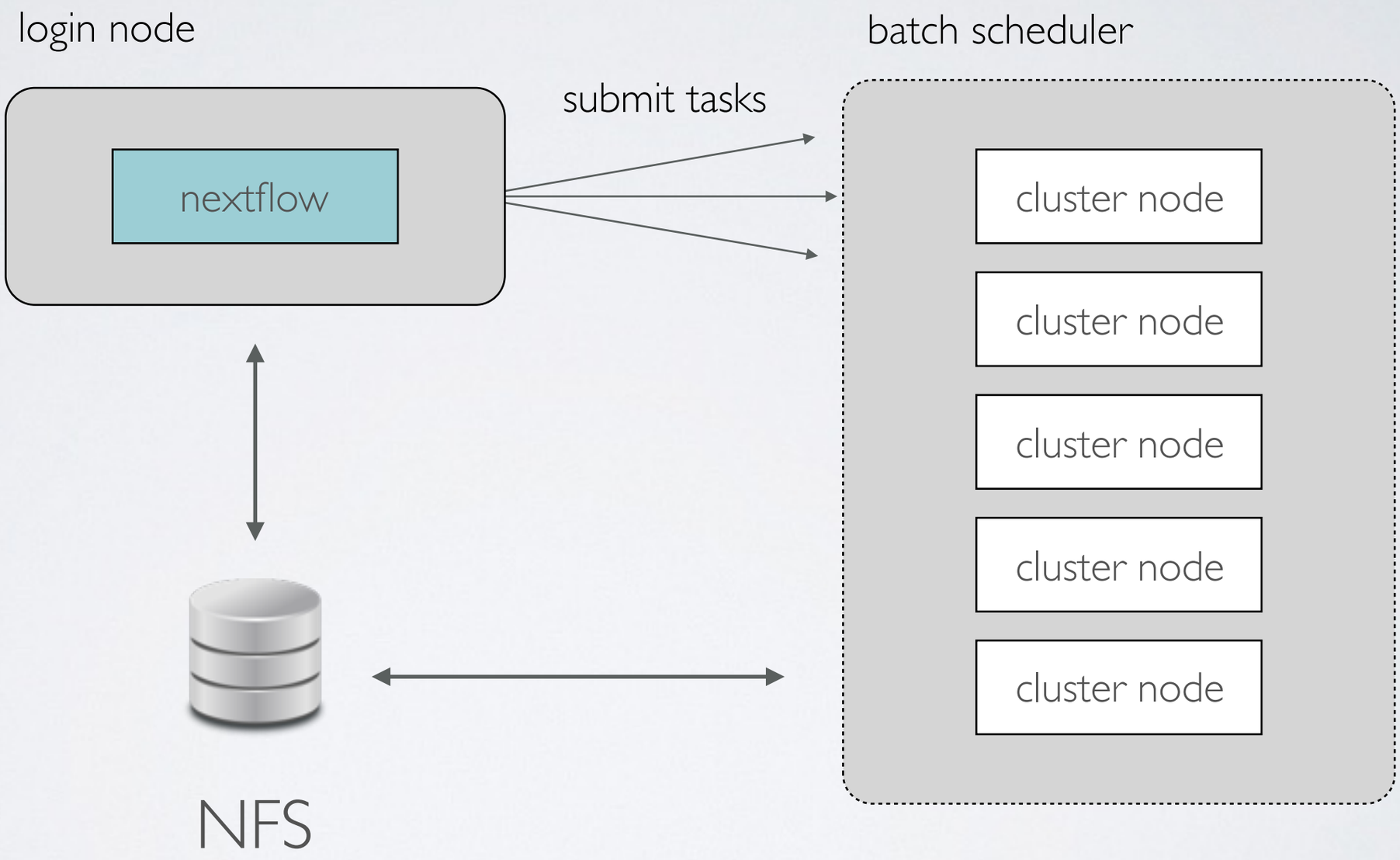

Terminal
$ nextflow run CRG-CNAG/CalliNGS-NF


Terminal
$ nextflow run CRG-CNAG/CalliNGS-NF
N E X T F L O W ~ version 0.24.1
Pulling CRG-CNAG/CalliNGS-NF ...
downloaded from https://github.com/CRG-CNAG/CalliNGS-NF.git
Launching `CRG-CNAG/CalliNGS-NF` [mad_wilson] - revision: 1187e44c7a
C A L L I N G S - N F v 1.0
genome : /users/data/CalliNGS-NF/data/genome.fa
reads : /users/data/reads/rep1_{1,2}.fq.gz
variants : /users/data/known_variants.vcf.gz
blacklist: /users/data/blacklist.bed
results : results
gatk : /users/ngs17-demo/GenomeAnalysisTK.jar
[warm up] executor > crg
[b0/24a331] Submitted process > 1D_prepare_vcf_file
[18/5cea87] Submitted process > 1B_prepare_genome_picard
[88/ae768f] Submitted process > 1C_prepare_star_genome_index
[c1/290240] Submitted process > 1A_prepare_genome_samtools
[fb/e81338] Submitted process > 2_rnaseq_mapping_star

Terminal

```
$ nextflow run CRG-CNAG/CalliNGS-NF -revision v1.2
```

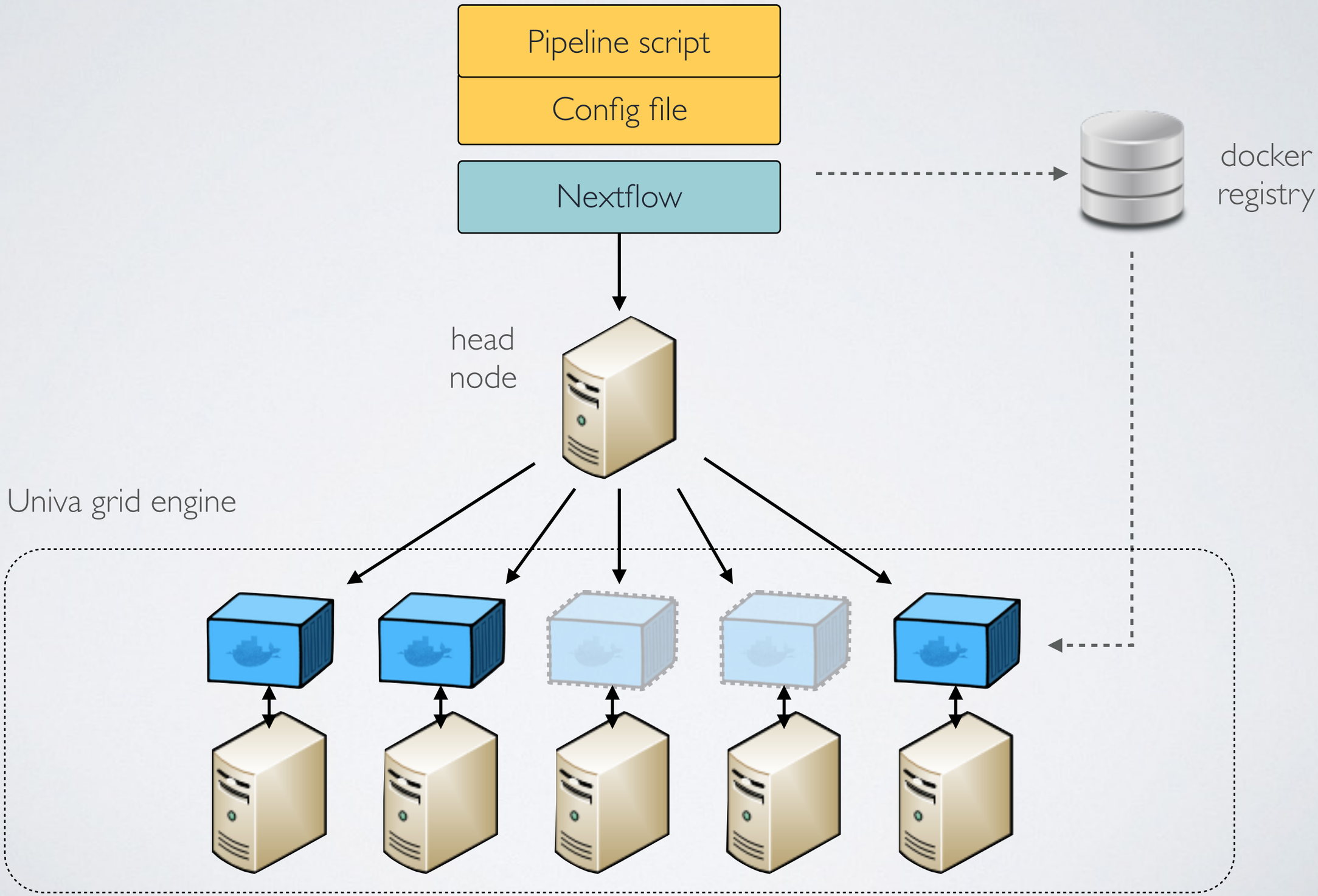
HPC DEPLOYMENT



CONFIGURATION FILE

```
process {
  executor = 'sge'
  queue = 'cn-el6'
  memory = '10GB'
  cpus = 8
  time = '2h'
}
```

MULTI-SCALE CONTAINERS



MULTI-SCALE CONTAINERS

- Docker is not HPC friendly :(
- It requires a modern Linux kernel + daemon running as root on each computing node
- Singularity is a container engine designed for HPC
- Docker images can be converted to Singularity images

CONFIGURATION FILE

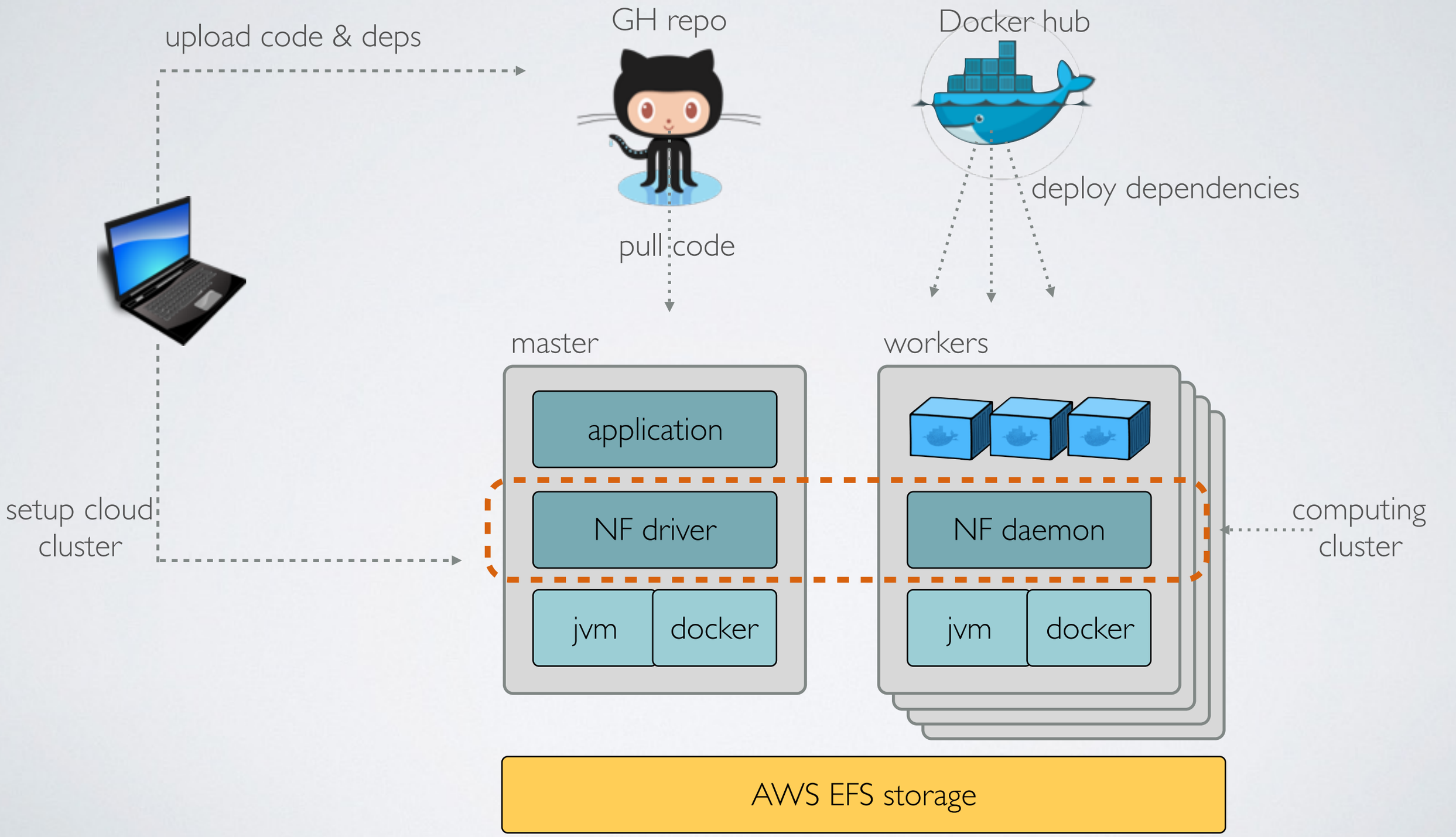
```
process {
  executor = 'sge'
  queue = 'cn-el6'
  container = 'image/name:tag'
 }

docker.enabled = true
```

CONFIGURATION FILE

```
process {
  executor = 'sge'
  queue = 'cn-el6'
  container = '/path/to/file.img'
 }

singularity.enabled = true
```

DEMO

CLOUD DEPLOYMENT



ELASTIC CLUSTER

- Native cloud scheduler supporting auto-scaling
- Instances are added on workload pressure
- Instances are terminated when idle
- Support for EC2 spot instances

CLOUD CONFIG

```
cloud {
  imageId = 'ami-43f49030'
  instanceType = 'm4.xlarge'
  subnetId = 'subnet-05222a43'
  sharedStorageId = 'fs-1803efd1'
  spotPrice = 0.06
}
```

JUST THREE COMMANDS

```
$ nextflow cloud create

$ ssh head ec2-master-node

$ nextflow run <your-pipeline>
```

DEMO

https://goo.gl/0ZoERN

Q&A

THANK YOU