nextflow

nextflow

- Job dependency management
- Caching
- Reporting
- Flexible+Portable software environments
  - Cluster environment
  - Conda
  - Docker / Singularity
- Runs scripts in any language
- Version Control
- Complex pipelines:
  - branching
  - explore large parameter spaces

Who is using Nextflow?

CRG Centre for Genomic Regulation

wellcome sanger institute

CANCER RESEARCH UK | BEATSON INSTITUTE

International Agency for Research on Cancer

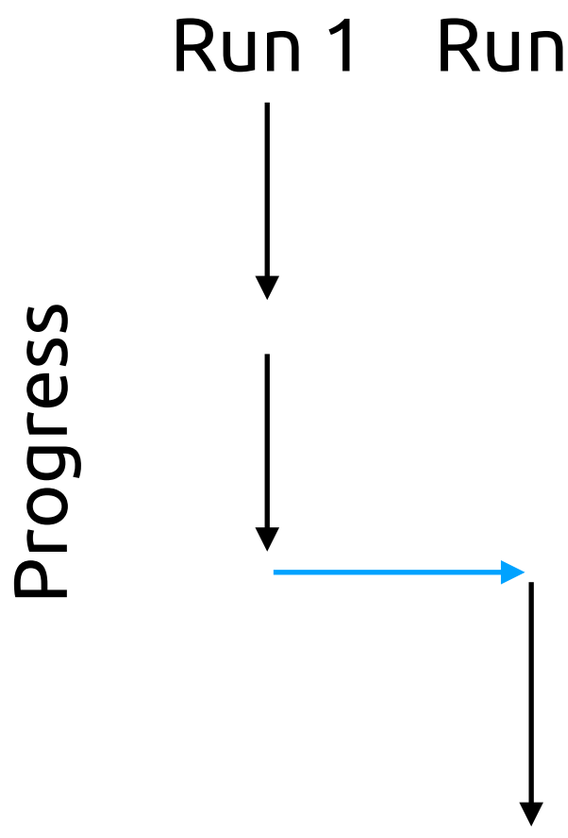
World Health Organization

SciLifeLab

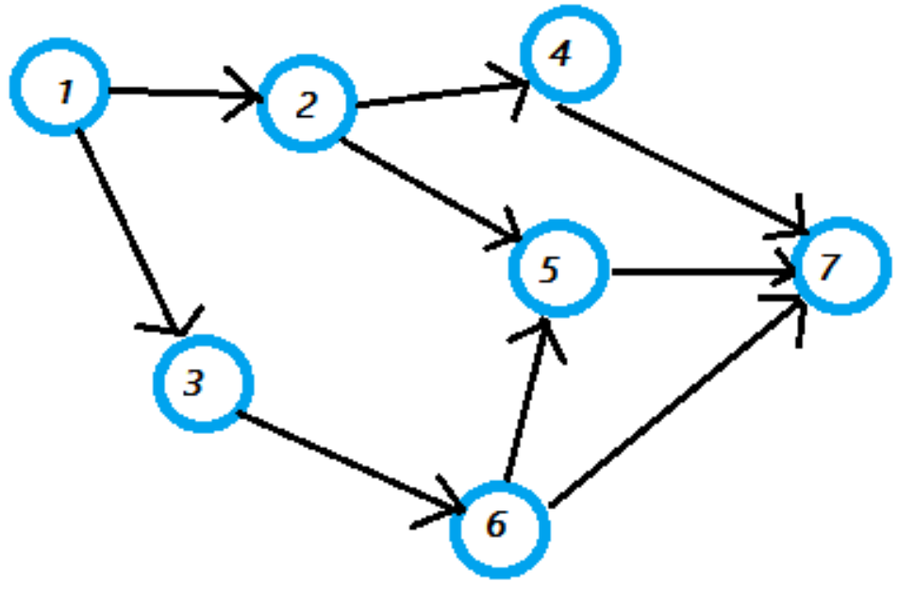
THE FRANCIS CRICK INSTITUTE

Why do we need pipeline workflow software?

Re-entrancy



Job dependency



Major features of Nextflow

- **Portability**
- **Reproducibility**
- **Management of Complexity**
- **Increases productivity**
- **Standardized format**

Two Versions of Nextflow syntax exist

DSL 1

Original Version

DSL 2

Modular

Less verbose

More powerful

Core elements of Nextflow pipelines

Processes / jobs

Channels

Operators

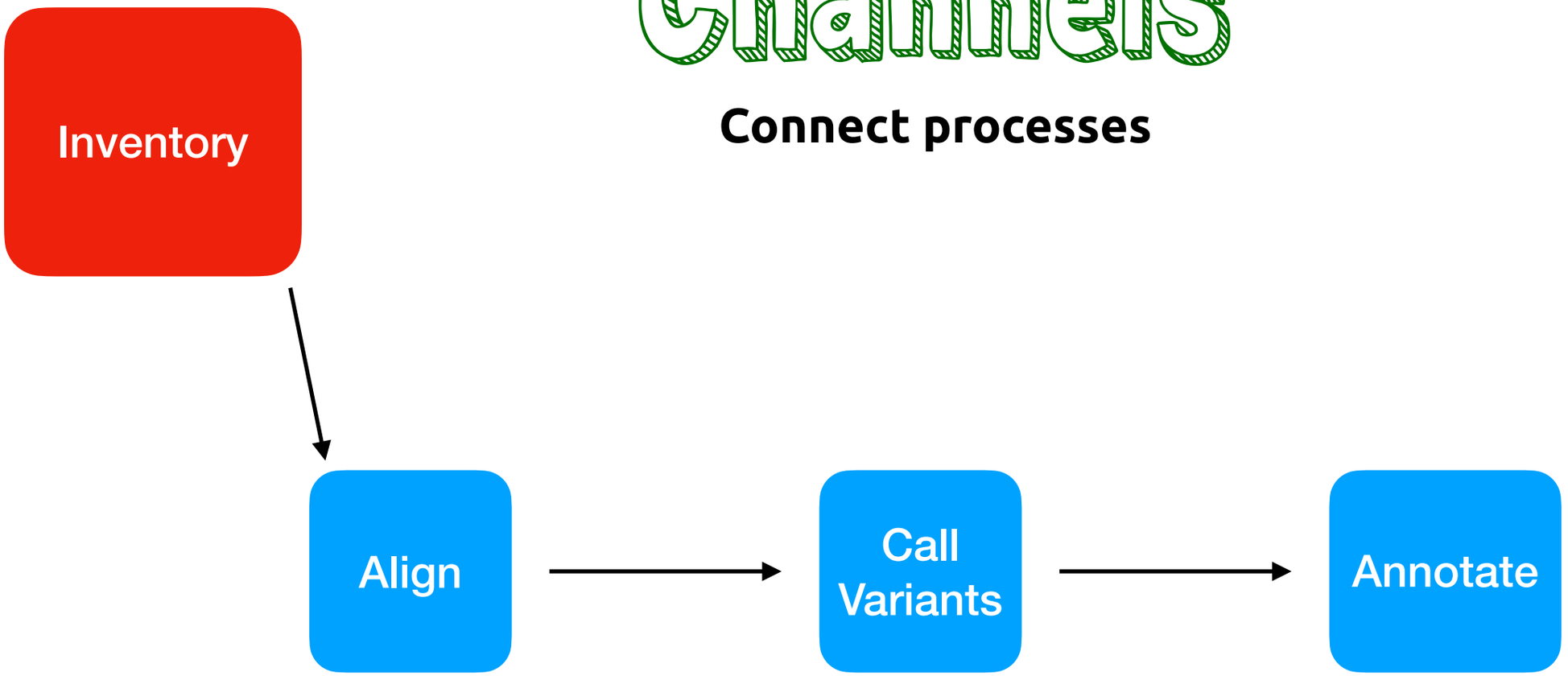
Executors

Processes

Define tasks

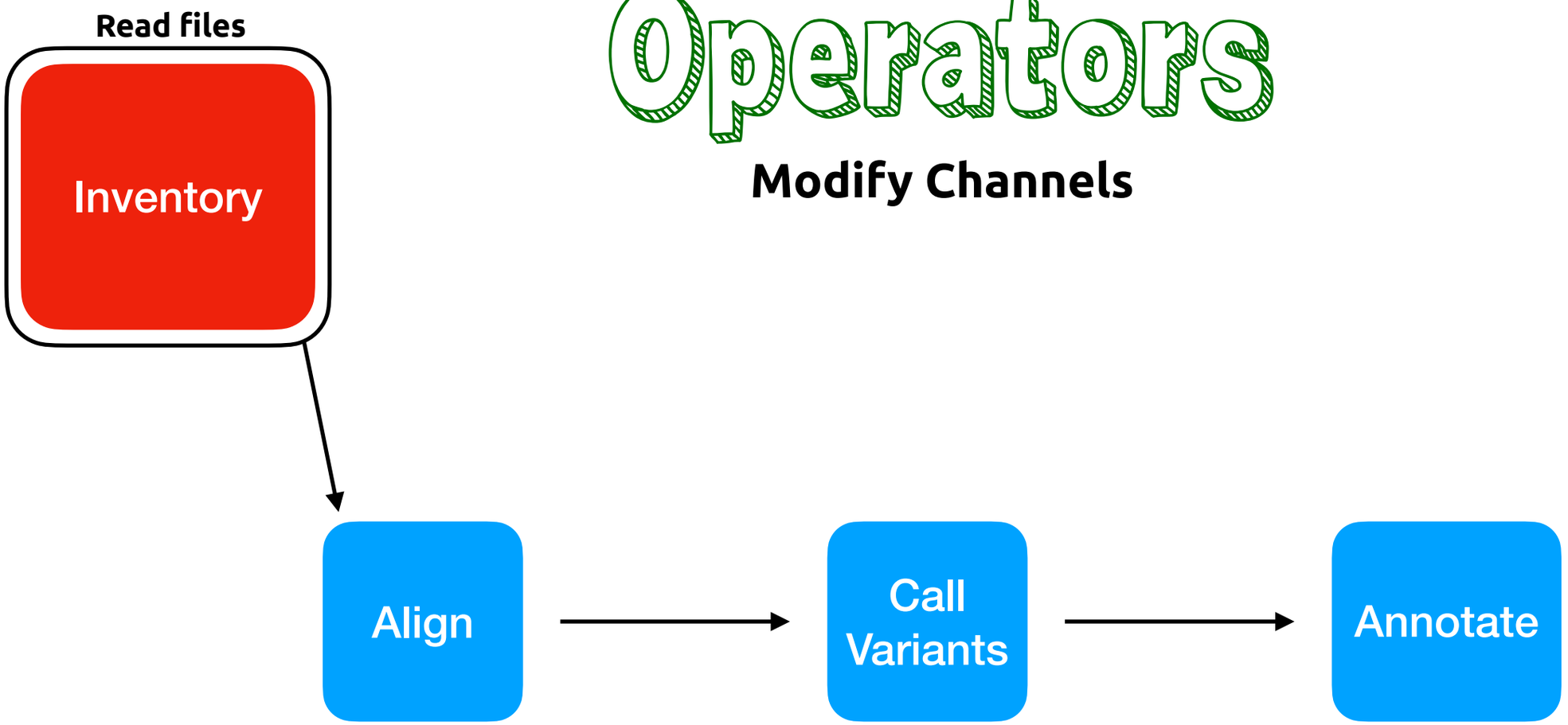
Channels

Connect processes



Operators

Modify Channels

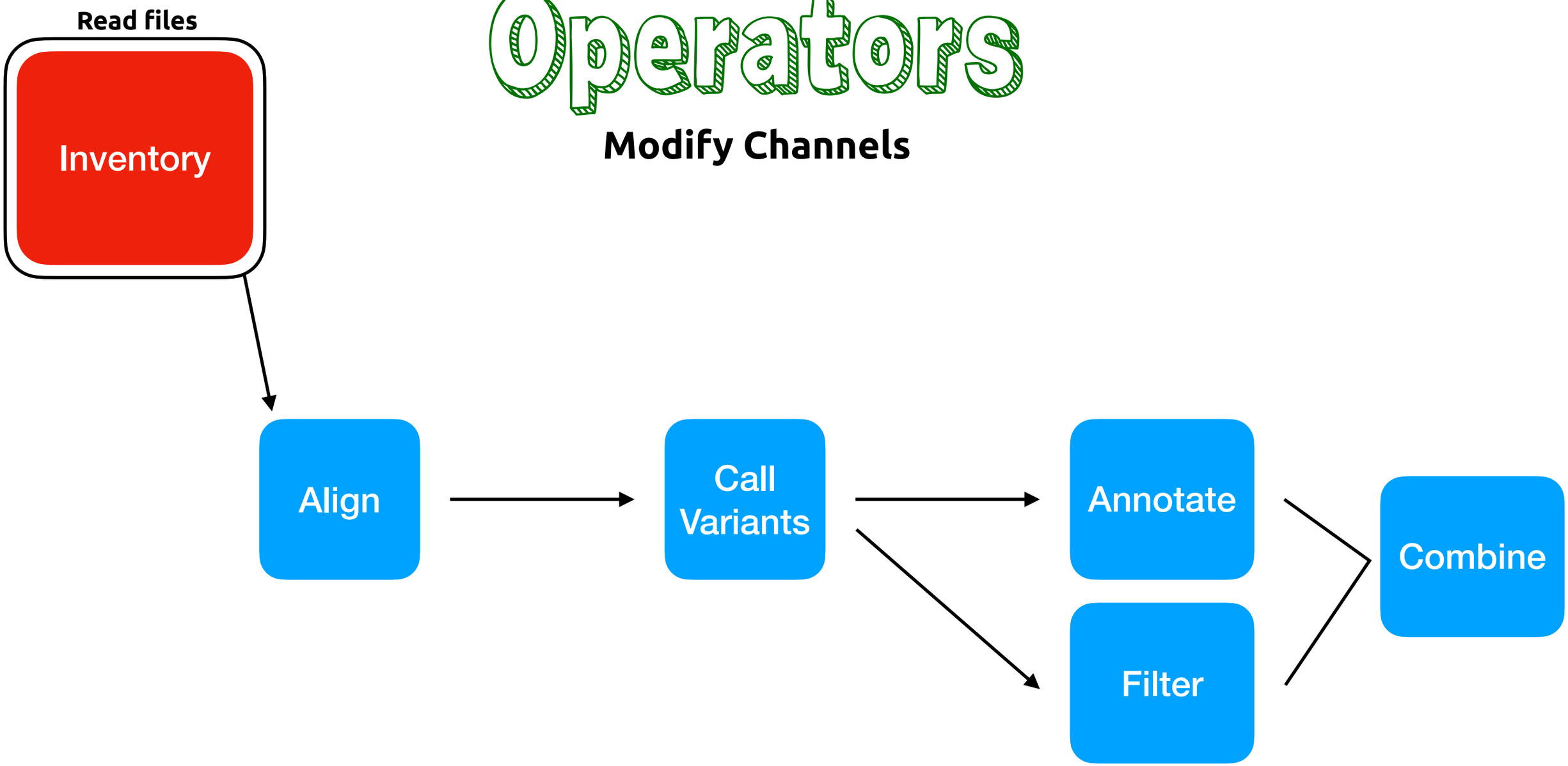


Create a channel

Modify a channel

Operators

Modify Channels



split. map combine filter

Executors

Define task runners

Read files

Inventory

Align

Call Variants

Annotate

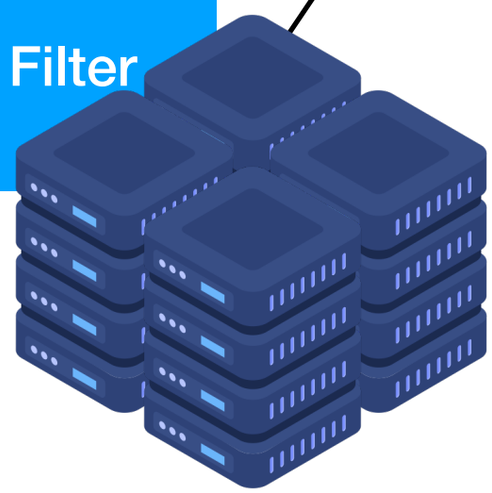
Filter

Combine

Locally

aws

Cloud



HPC

Processes: Directives

Modify process behavior

Read files

Inventory

Set requirements

Use docker

publish outputs

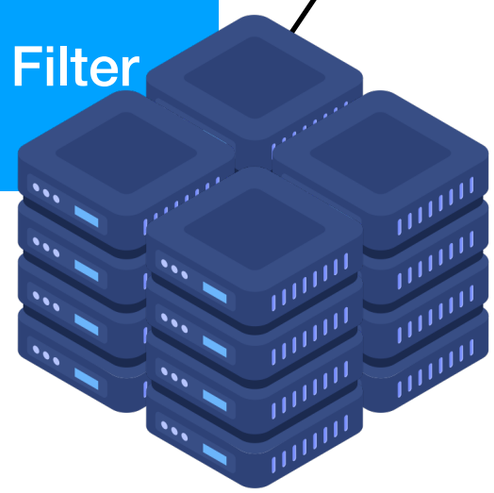
Align → Call Variants → Annotate → Combine

Call Variants → Filter → Combine

Locally

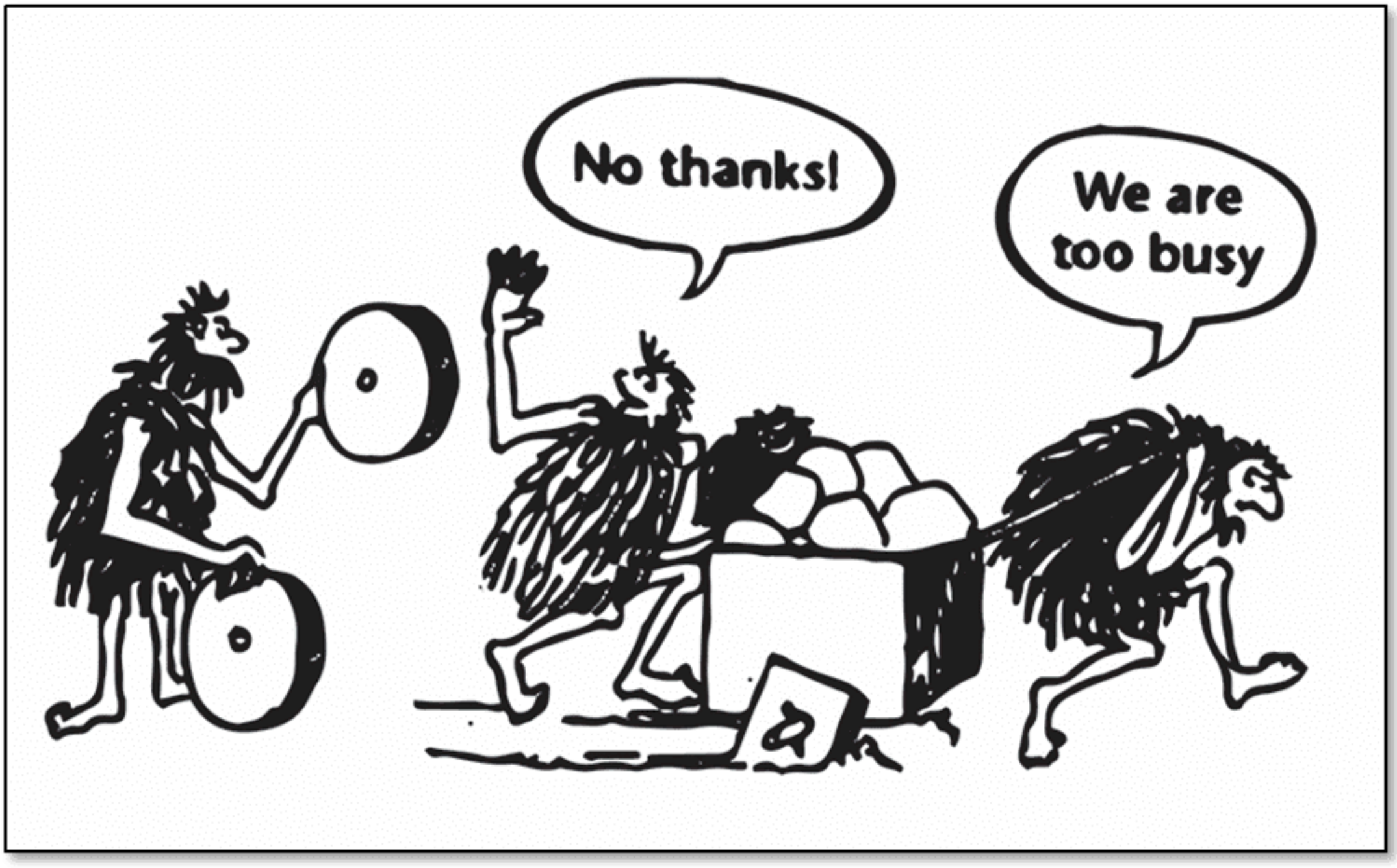
aws

Cloud



HPC

Nextflow has a learning curve



Sacrifice short-term loss for long-term gain

Nextflow...

is stable

has 75 contributors

is used by thousands

Groovy Basics

Variables

```
println "Hello, World!"

x = 1
println x

x = new java.util.Date()
println x

x = -3.1499392
println x

x = false
println x

x = "Hi"
println x
```

Lists

```
myList = [1776, -1, 33, 99, 0, 928734928763]

println myList[0] // 1776

println myList.size() // 6
```

Maps (dictionaries)

```
scores = [ "Brett":100, "Pete":"Did not finish", "Andrew":86.87934 ]

println scores["Pete"]
println scores.Pete
```

Conditional Execution

```
x = Math.random()
if( x < 0.5 ) {
    println "You lost."
}
else {
    println "You won!"
}
```

Multiple Assignment

```
(a, b, c) = [10, 20, 'foo']
assert a == 10 && b == 20 && c == 'foo'
```

String interpolation

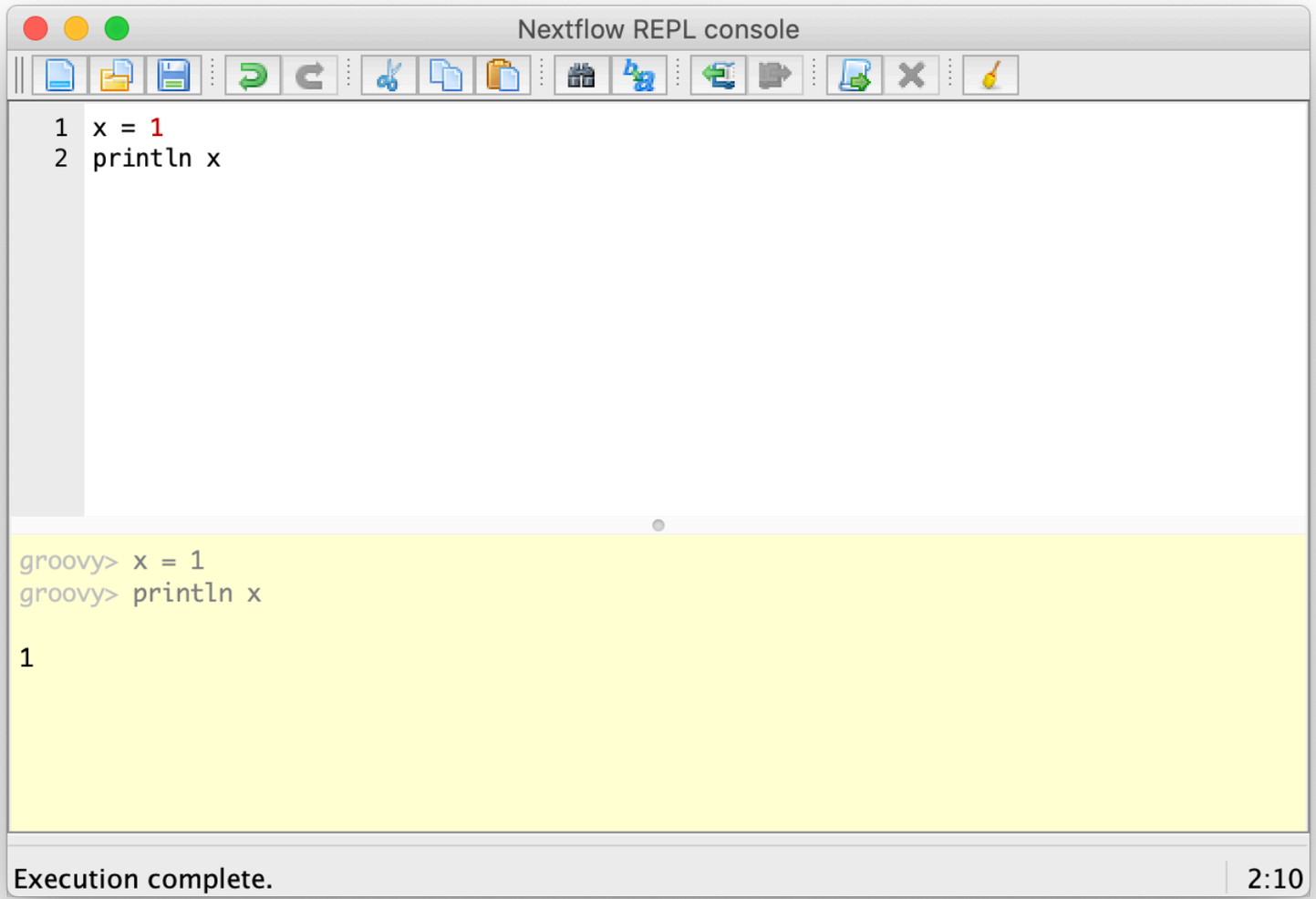
```
foxtype = 'quick'
foxcolor = ['b', 'r', 'o', 'w', 'n']
println "The $foxtype ${foxcolor.join()} fox"

x = 'Hello'
println '$x + $y'
```

Nextflow console

Nextflow has a built in console you can use to experiment

Type `nextflow console`



Start a workflow off by bringing in data

From Values

From File Paths

From an Inventory

Starting a workflow with values

Creating values

```
>> NXF_VER=20.07.1 nextflow run main.nf
```

```
1   nextflow.enable.dsl=2
2
3   Channel
4       .of(1..23, 'X', 'Y')
5       .view()
```

- 1..23 expands to a list
- Operators are chained
- `.view()` prints the channel content

```
[dec::~/nf-01-basic] (master) → NXF_VER=20.07.1 nextflow run main.nf
N E X T F L O W  ~  version 20.07.1
Launching `main.nf` [extravagant_joliot] - revision: 4e35a8a2e1
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
X
Y
```

Combine generates the cartesian product of two channels

```
>> NXF_VER=20.07.1 nextflow run main.nf
```

```
ch_sample = Channel.fromList(["A","B"])
ch_assay = Channel.of(1..5)
ch_sample.combine(ch_assay).view()
```

- Use to explore a parameter space

```
N E X T F L O W  ~  version 20.07.1
Launching `main.nf` [maniac_crick] - revision: a591d7605e
[A, 1]
[A, 2]
[B, 1]
[B, 2]
[A, 3]
[B, 3]
[A, 4]
[B, 4]
[A, 5]
[B, 5]
```

Some operators accept a closure argument

>> NXF_VER=20.07.1 nextflow run main.nf

```
Channel
    .of(1..10)
    .map { it ^ 2 }
    .view()
```

- map can be used to modify channels

```
N E X T F L O W  ~  version 20.07.1
Launching `main.nf` [deadly_murdock] - revision: 1302ad7ff0
1
4
9
16
25
36
49
64
81
100
```

Groovy Closures

Anonymous functions

Defined using `{}`

`it` is the implicit argument

last statement is the return value

```
{ "$it.bai" }

                { it * 10 }

                        { println it; it * 10 }
```

The filter operator can remove values from a channel

```
>> NXF_VER=20.07.1 nextflow run main.nf
```

```
1   nextflow.enable.dsl=2
2
3   Channel
4       .of(1..23, 'X', 'Y')
5       // Regular expression filtering
6       .filter { it -> it =~ /[0-9]+/ }
7       .view()
```

- 1..23 expands to a list
- Operators are chained
- `.view()` prints the channel content
- `{ }` indicates a closure; Filter is an operator that takes a closure argument.

```
[dec::~/nf-01-basic] (master) → NXF_VER=20.07.1 nextflow run main.nf
N E X T F L O W  ~  version 20.07.1
Launching `main.nf` [extravagant_joliot] - revision: 4e35a8a2e1
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
```

Start things off from a set of files

Define a channel with a single file

```
myFileChannel = Channel.fromPath( '/data/some/bigfile.txt' )
```

[bigfile.txt]

Match multiple files

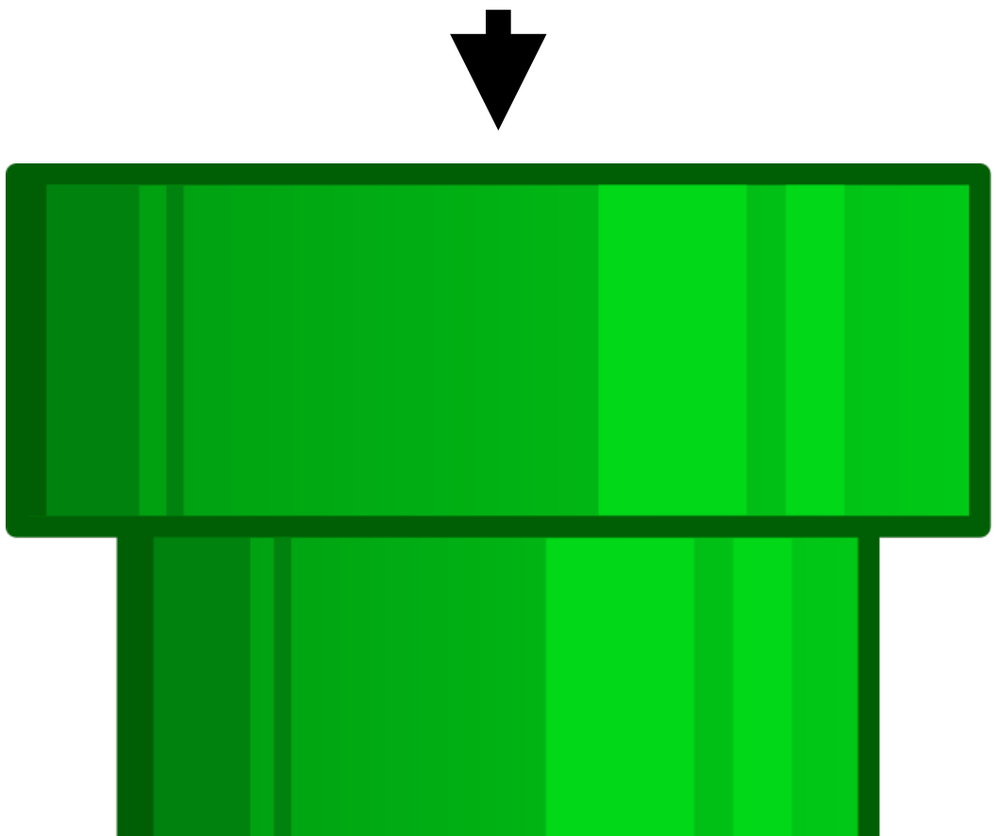
```
myFileChannel = Channel.fromPath( '/data/big/*.txt' )
```

[A.txt B.txt C.txt]

Match pairs of files

```
pairFiles = Channel.fromPath( 'data/file_{1,2}.fq' )
```

[[file_1.fq, file_2.fq]]



Start things off from a set of files

Match multiple pairs of files

```
pairFiles = Channel.fromPath( 'data/S*_{1,2}.fq' )
```

[[SA_1.fq, SA_2.fq] [SB_1.fq, SB_2.fq]]

Channels can pass lists of objects

...Or lists of lists of objects

Start things off from a set of files

Match multiple pairs of files

```
pairFiles = Channel.fromPath( 'data/S*_{1,2}.fq' )
```

[SA_1.fq, SA_2.fq]
[SB_1.fq, SB_2.fq]

↓ **Channel passed to process**

Alignment Process

Run as two separate jobs!

Start things off from an inventory

Input

`sample_list.tsv`

sample_name	filename
A	xyz.txt
B	TATAATCTAT.txt
C	cacgttgatg.csv

Script

```
Channel.fromPath("sample_list.csv")
    .splitCsv(header: true)
    .view()
```

Output

```
N E X T F L O W  ~  version 20.07.1
Launching `main.nf` [modest_stonebraker] - revision: b16d8121c8
[sampe_name:A, filename:xyz.txt]
[sampe_name:B, filename:TATAATCTAT.txt]
[sampe_name:C, filename:cacgttgatg.csv]
```

Stores each row in a hash map / dictionary

Start things off from an inventory

Input

`sample_list.tsv`

sample_name	filename
A	xyz.txt
B	TATAATCTAT.txt
C	cacgttgatg.csv

Script

```
Channel.fromPath("sample_list.csv")
    .splitCsv(header: true)
    .view()
```

Output

```
N E X T F L O W  ~  version 20.07.1
Launching `main.nf` [modest_stonebraker] - revision: b16d8121c8
[sampe_name:A, filename:xyz.txt]
[sampe_name:B, filename:TATAATCTAT.txt]
[sampe_name:C, filename:cacgttgatg.csv]
```

Advantages:

Map sample names to files

Not reliant on the file system

Apply filters / remove poor quality

Pass additional parameters

Define a custom parameter space

Stores each row in a hash map / dictionary

Start things off from an inventory

Input

`sample_list.tsv`

sample_name	filename
A	xyz.txt
B	TATAATCTAT.txt
C	cacgttgatg.csv

Script

```
Channel.fromPath("sample_list.csv")
    .splitCsv(header: true)
    .view()
```

Output

```
N E X T F L O W  ~  version 20.07.1
Launching `main.nf` [modest_stonebraker] - revision: b16d8121c8
[sampe_name:A, filename:xyz.txt]
[sampe_name:B, filename:TATAATCTAT.txt]
[sampe_name:C, filename:cacgttgatg.csv]
```

Advantages:

Map sample names to files

Not reliant on the file system

Apply filters / remove poor quality

Pass additional parameters

Define a custom parameter space

Stores each row in a hash map / dictionary

My recommendation: This is the best way to start your pipeline

Processes

Define a simple process

sample_list.csv

id	filename	sample
1	d1.txt	sample_A
2	d2.txt	sample_B
3	d3.txt	sample_C

NXF_VER=20.07.1 nextflow run main.nf

```
nextflow.enable.dsl=2

// By convention, paths leave off trailing slash
data_folder = "data"

// Define Channels
samples = Channel.fromPath("sample_list.csv")
                 .splitCsv(header: true, sep: ",")
                 .view()
                 .map { [
                          it.sample,
                          file("$data_folder/${it.filename}")
                        ]
                 }
                 .view()


process count_chars {

    input:
        tuple val(sample), path(dna)

    """
        chars=\$(cat $dna | wc -c)
        echo \$chars > char_count.txt
    """

}


workflow {
    count_chars(samples)
}
```

Define a simple process

sample_list.csv

id	filename	sample
1	d1.txt	sample_A
2	d2.txt	sample_B
3	d3.txt	sample_C

`NXF_VER=20.07.1 nextflow run main.nf`

```
nextflow.enable.dsl=2

// By convention, paths leave off trailing slash
data_folder = "data"

// Define Channels
samples = Channel.fromPath("sample_list.csv")
                 .splitCsv(header: true, sep: ",")
                 .view()
                 .map { [
                             it.sample,
                             file("$data_folder/${it.filename}")
                         ]
                 }
                 .view()


process count_chars {

    input:
```

```
N E X T F L O W  ~  version 20.07.1
Launching `main.nf` [small_galileo] - revision: df93f8f8cf
executor >  local (3)
[11/c3f933] process > count_chars (1) [100%] 3 of 3
[id:1, filename:d1.txt, sample:sample_A]
[id:2, filename:d2.txt, sample:sample_B]
[id:3, filename:d3.txt, sample:sample_C]
[sample_A, /Users/dec/Documents/coding/nextflow-example/nf-01-basic/data/d1.txt]
[sample_B, /Users/dec/Documents/coding/nextflow-example/nf-01-basic/data/d2.txt]
[sample_C, /Users/dec/Documents/coding/nextflow-example/nf-01-basic/data/d3.txt]
```

Describes the executors

Lists progress for every process

```
    count_chars(samples)
}
```

Define a simple process

sample_list.csv

id	filename	sample
1	d1.txt	sample_A
2	d2.txt	sample_B
3	d3.txt	sample_C

NXF_VER=20.07.1 nextflow run main.nf

```
nextflow.enable.dsl=2

// By convention, paths leave off trailing slash
data_folder = "data"

// Define Channels
samples = Channel.fromPath("sample_list.csv")
                 .splitCsv(header: true, sep: ",")
                 .view()
                 .map { [
                         it.sample,
                         file("$data_folder/${it.filename}")
                     ]
                 }
                 .view()


process count_chars {

    input:
        tuple val(sample), path(dna)

    count_chars(samples)
}
```

11/c3f933 is the work directory

```
[11/c3f933] process > count_chars (1) [100%] 3 of 3 ✔
[id:1, filename:d1.txt, sample:sample_A]
[id:2, filename:d2.txt, sample:sample_B]
[id:3, filename:d3.txt, sample:sample_C]
[sample_A, /Users/dec/Documents/coding/nextflow-example/nf-01-basic/data/d1.txt]
[sample_B, /Users/dec/Documents/coding/nextflow-example/nf-01-basic/data/d2.txt]
[sample_C, /Users/dec/Documents/coding/nextflow-example/nf-01-basic/data/d3.txt]
```

Use view statements to debug

sample_list.csv

id	filename	sample
1	d1.txt	sample_A
2	d2.txt	sample_B
3	d3.txt	sample_C

Print and pass on

NXF_VER=20.07.1 nextflow run main.nf

```
nextflow.enable.dsl=2

// By convention, paths leave off trailing slash
data_folder = "data"

// Define Channels
samples = Channel.fromPath("sample_list.csv")
                 .splitCsv(header: true, sep: ",")
                 .view()
                 .map { [
                         it.sample,
                         file("$data_folder/${it.filename}")
                        ]
                 }
                 .view()


process count_chars {

    input:
        tuple val(sample), path(dna)

    count_chars(samples)
}
```

```
executor >  local (3)
[11/c3f933] process > count_chars (1) [100%] 3 of 3 ✔
[id:1, filename:d1.txt, sample:sample_A]
[id:2, filename:d2.txt, sample:sample_B]
[id:3, filename:d3.txt, sample:sample_C]
[sample_A, /Users/dec/Documents/coding/nextflow-example/nf-01-basic/data/d1.txt]
[sample_B, /Users/dec/Documents/coding/nextflow-example/nf-01-basic/data/d2.txt]
[sample_C, /Users/dec/Documents/coding/nextflow-example/nf-01-basic/data/d3.txt]
```

Lists are passed in as tuples

sample_list.csv

id	filename	sample
1	d1.txt	sample_A
2	d2.txt	sample_B
3	d3.txt	sample_C

NXF_VER=20.07.1 nextflow run main.nf

```
nextflow.enable.dsl=2

// By convention, paths leave off trailing slash
data_folder = "data"

// Define Channels
samples = Channel.fromPath("sample_list.csv")
                 .splitCsv(header: true, sep: ",")
                 .view()
                 .map { [
                          it.sample,
                          file("$data_folder/${it.filename}")
                        ]
                 }
                 .view()


process count_chars {

    input:
        tuple val(sample), path(dna)

    """
        chars=\$(cat $dna | wc -c)
        echo \$chars > char_count.txt
    """

}


workflow {
    count_chars(samples)
}
```

Use path to indicate a file is being passed in

Rename passed in items

Lists are passed in as tuples

sample_list.csv

id	filename	sample
1	d1.txt	sample_A
2	d2.txt	sample_B
3	d3.txt	sample_C

NXF_VER=20.07.1 nextflow run main.nf

```
nextflow.enable.dsl=2

// By convention, paths leave off trailing slash
data_folder = "data"

// Define Channels
samples = Channel.fromPath("sample_list.csv")
                 .splitCsv(header: true, sep: ",")
                 .view()
                 .map { [
                             it.sample,
                             file("$data_folder/${it.filename}")
                         ]
                 }
                 .view()


process count_chars {

    input:
        tuple val(sample), path(dna)

    """
        chars=\$(cat $dna | wc -c)
        echo \$chars > char_count.txt
    """

}


workflow {
    count_chars(samples)
}
```

Bash!

Must escape certain bash characters

Use single quotes to avoid escaping

Nextflow variables

!{dna}

Bash Variables

$chars

NXF_VER=20.07.1 nextflow run main.nf

```
nextflow.enable.dsl=2

// By convention, paths leave off trailing slash
data_folder = "data"

// Define Channels
samples = Channel.fromPath("sample_list.csv")
                 .splitCsv(header: true, sep: ",")
                 .view()
                 .map { [
                             it.sample,
                             file("$data_folder/${it.filename}")
                         ]
                 }
                 .view()


process count_chars {

    input:
        tuple val(sample), path(dna)

    shell:
    '''
        chars=$(cat !{dna} | wc -c)
        echo $chars > char_count.txt
    '''

}


workflow {
    count_chars(samples)
}
```

*This only works for SHELL/BASH scripts

Add more directives to extend the functionality of processes!

memory

time

cpus

```
process count_chars {

    memory '4G'
    time '1:00:00'

    input:
        tuple val(sample), path(dna)

    '''
        chars=$(cat !dna | wc -c)
        echo $chars > char_count.txt
    '''

}
```

Aggregate results using collectFile

```
// Outputs
params.output = "results"
```

```
process count_chars {

    memory '4G'
    time '1:00:00'

    input:
        tuple val(sample), path(dna)

    output:
        path("char_count.txt")

    shell:
    '''
        chars=$(cat !{dna} | wc -c)
        echo $chars > char_count.txt
    '''

}


workflow {
    count_chars(samples)

    // Aggregate results
    count_chars.out.collectFile(name: "char_count.tsv", storeDir: params.output)
}
```

output is passed to collectFile

Aggregating files with headers

```
skip: 1
keepHeader: true
```

Running custom scripts within processes

```r
#!/usr/bin/env Rscript
library(readr)
df <- data.frame(random_numbers = runif(100))
readr::write_tsv(df, "random.tsv")
```

```
process generate_random {

    echo true

    output:
        path "random.tsv"

    """
        generate_random.R
    """
}

workflow {
    generate_random()
}
```

Store in workflow directory at:

`bin/generate_random.R`

The #! (hashbang) is very important!

`#!/usr/bin/env`

You must also:

`chmod +x  generate_random.R`

Specifying the environment using conda

```
process generate_random {

    conda 'r-tidyverse'

    output:
        path "random.tsv"

    """
        generate_random.R
    """
}

workflow {
    generate_random()
}
```

Specifying the environment using conda

```
process generate_random {

    conda 'r-tidyverse'

    output:
        path "random.tsv"

    """
        generate_random.R
    """
}

workflow {
    generate_random()
}
```

```
process generate_random {

    conda 'r-tidyverse=1.2.1 r-data.table=1.12.2'

    output:
        path "random.tsv"

    """
        generate_random.R
    """
}
```

Pro tip: Use version numbers

Specifying the environment using conda

```
process generate_random {

    conda 'r-tidyverse'

    output:
        path "random.tsv"

    """
        generate_random.R
    """
}

workflow {
    generate_random()
}
```

```
process generate_random {

    conda 'r-tidyverse=1.2.1 r-data.table=1.12.2'

    output:
        path "random.tsv"

    """
        generate_random.R
    """
}
```

Pro tip: Use version numbers

```
conda 'env/r-env.yml'
```

You can also specify the environment using a conda yaml file

Specifying the environment using conda

```
process generate_random {

    conda 'r-tidyverse'

    output:
        path "random.tsv"

    """
        generate_random.R
    """
}

workflow {
    generate_random()
}
```

Specifying the environment using a container (docker / singularity)

```
process generate_random {

    container 'danielecook/renv'

    output:
        path "random.tsv"

    """
        generate_random.R
    """
}
```

Nextflow will pull the environment for docker / singularity

Processes can also make use of a unix-style bar syntax

```
nextflow.enable.dsl=2


process sayHello {
  input:
    val cheers
  output:
    stdout

  """
  echo $cheers
  """
}

workflow {
  channel.of('Ciao','Hello','Hola') | sayHello | view
}
```