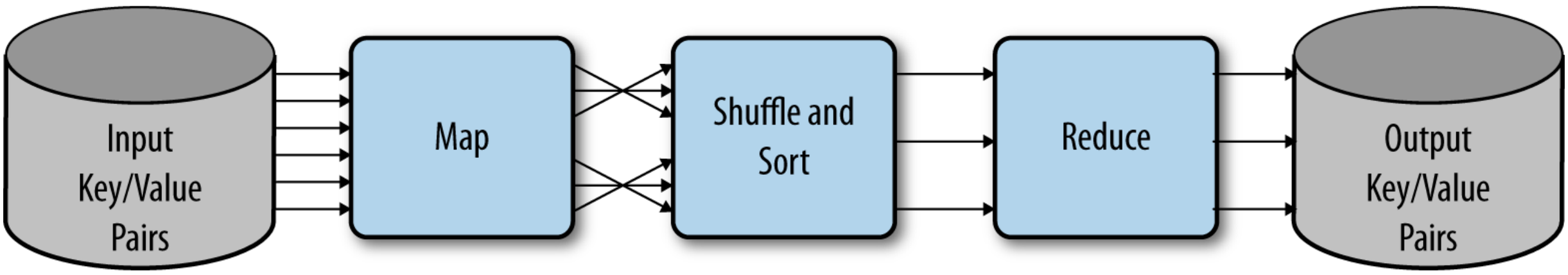
Ejemplos Hadoop y Nextflow

Alex Di Genova

25/10/2023

Hadoop

MapReduce: Un modelo de programación funcional



1. Los datos locales se cargan en un proceso de mapeo como pares clave/valor de HDFS.

2. Los mapeadores generan cero o más pares clave/valor, asignando valores calculados a una clave en particular.

3. Luego, estos pares se ordenan/barajan en función de la clave y luego se pasan a un reductor de modo que todos los valores de una clave estén disponibles.

4. Los reductores deben generar cero o más pares clave/valor finales, que son la salida.

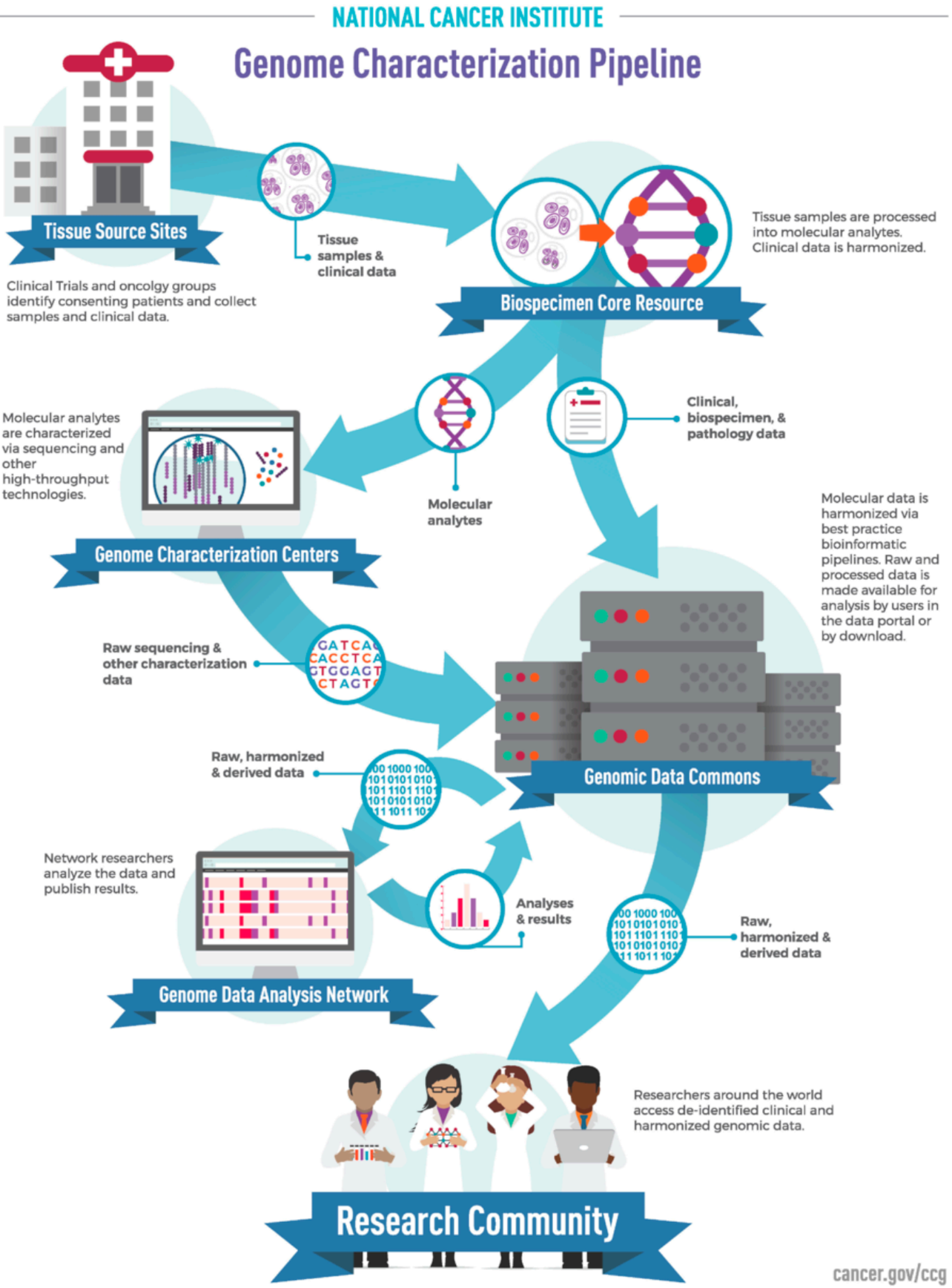
Hadoop
Map/Reduce

```
def map(dockey, line):
    for word in value.split():
        emit(word, 1)
```

```
def reduce(word, values):
    count = sum(value for value in values)
    emit(word, count)
```

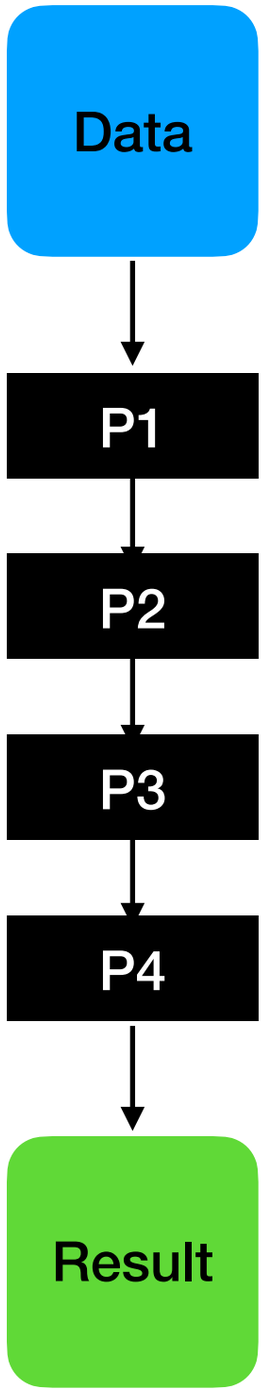
```
# Input to WordCount mappers

(27183, "The fast cat wears no hat.")
(31416, "The cat in the hat ran fast.")

# Mapper 1 output

("The", 1), ("fast", 1), ("cat", 1), ("wears", 1),
("no", 1), ("hat", 1),(".", 1)

# Mapper 2 output

("The", 1), ("cat", 1), ("in", 1), ("the", 1),
("hat", 1), ("ran", 1),("fast", 1),(".", 1)


# Input to WordCount reducers
# This data was computed by shuffle and sort

(., [1, 1])
("cat", [1, 1])
("fast", [1, 1])
("hat", [1, 1])
("in", [1])
("no", [1])
("ran", [1])
("the", [1])
("wears", [1])
("The", [1, 1])

# Output by all WordCount reducers

(., 2)
("cat", 2)
("fast", 2)
("hat", 2)
("in", 1)
("no", 1)
```

https://github.com/adigenova/uohpmd/blob/main/code/Haddop_nextflow.ipynb

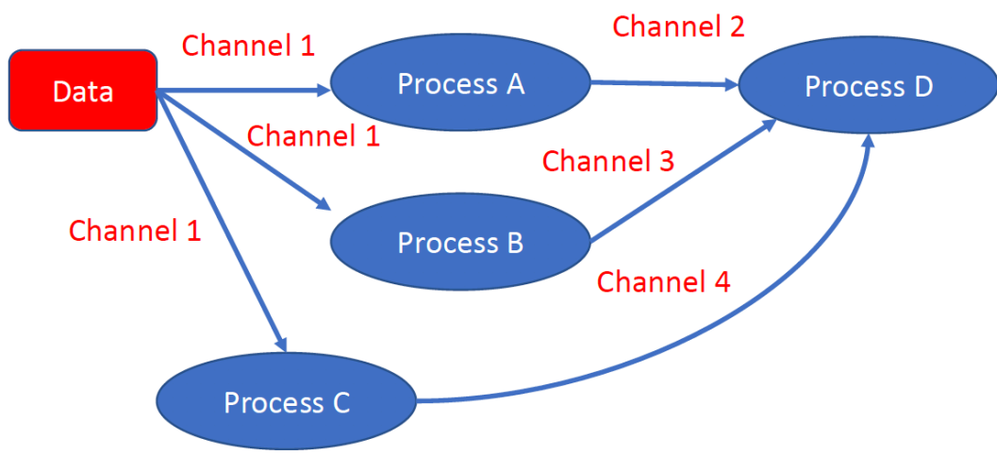


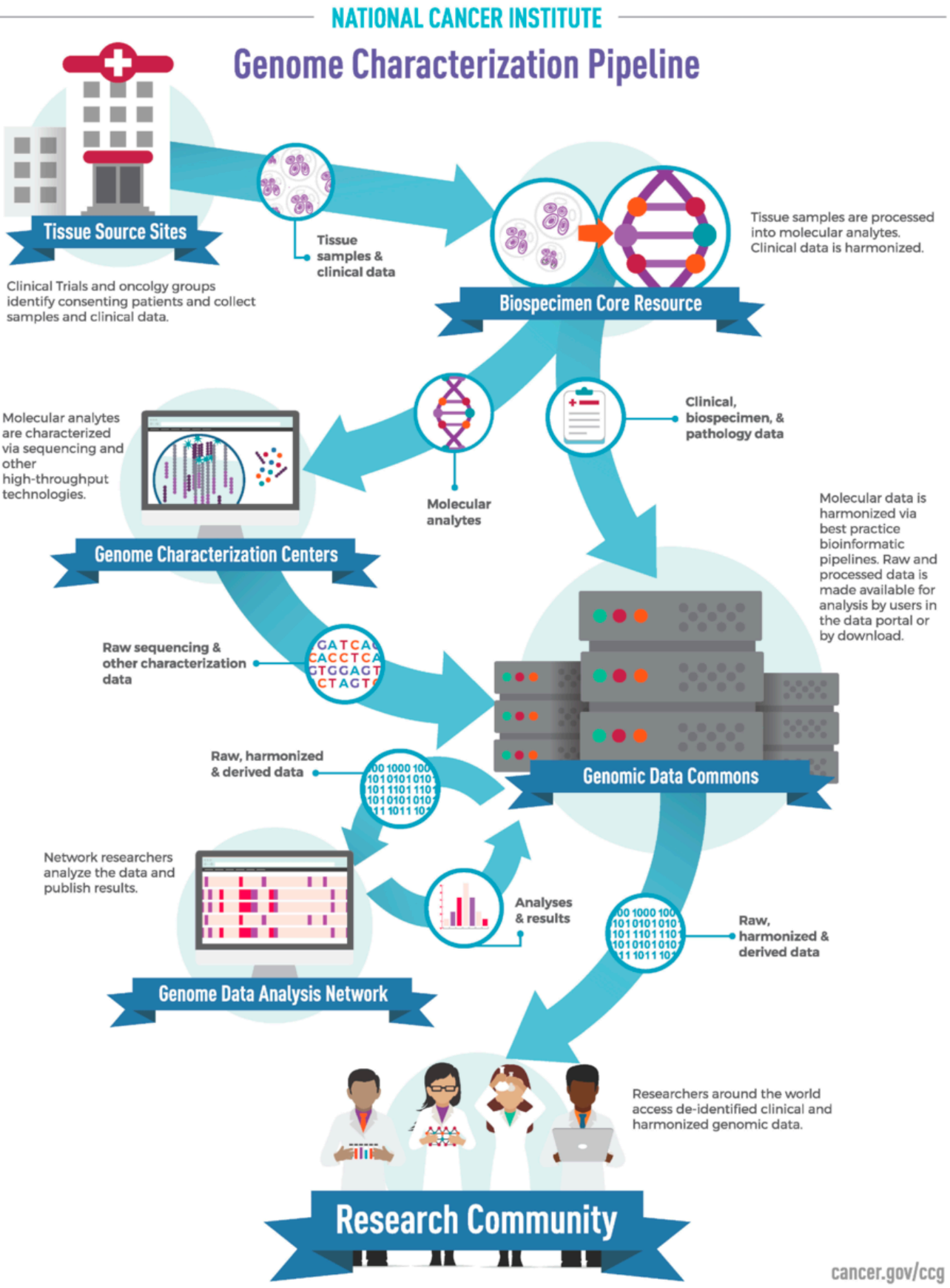
Nextflow

Genómica



Workflow





Nextflow

Genómica

Requisitos de pipelines para proyectos genómicos a gran escala

- **Reproducible**: los resultados genómicos deben ser totalmente reproducibles
- **Escalable**: Facil ejecucion en cluster HPC o cloud.
- **Portátil**: puede ejecutarse en varias infraestructuras (diferente sistema operativo, cloud)
- **Manejar la heterogeneidad**: funciona con dependencias de software en conflicto y varios requisitos de recursos.

Nextflow

Primer pipeline

Procesos

```
//pipeline_01.nf
nextflow.enable.dsl=2

process INDEX {
    input:
      path transcriptome
    output:
      path 'index'
    script:
      """
      salmon index -t $transcriptome -i index
      """
}

 process QUANT {
    input:
      each  path(index)
      tuple(val(pair_id), path(reads))
    output:
      path pair_id
    script:
      """
      salmon quant --threads $task.cpus --libType=U \
          -I $index \
          -1 ${reads[0]} -2 ${reads[1]} -o $pair_id
      """
}

//Definición del pipeline
workflow {
     //canales de entrada
    transcriptome_ch =
     channel.fromPath('transcriptome/*.fa.gz',checkIfExists: true)
    read_pairs_ch =
     channel.fromFilePairs('reads/*_{1,2}.fq.gz',checkIfExists: true)
    //el proceso INDEX toma el calar transcriptome_ch
    index_ch = INDEX(transcriptome_ch)
    //el proceso QUANT toma dos canales como argumentos: Indice y las lecturas
    QUANT( index_ch, read_pairs_ch ).view()
}
```

Nextflow patterns

```
process foo {
  debug true
  input:
  path x

  script:
  """
  echo your_command --input $x
  """
}

workflow {
  foo("$baseDir/data/reads/*_1.fq.gz")
}
```

Process per file path

```
process foo {
  debug true

  input:
  tuple val(sampleId), file(reads)

  script:
  """
  echo your_command --sample $sampleId --reads $reads
  """
}

workflow {
  Channel.fromFilePairs("$baseDir/data/reads/*_{1,2}.fq.gz", checkIfExists:true) \
    | foo
}
```

Process per file pairs

Nextflow patterns

Table 1

sampleId	read1	read2
C816RLABX	1_I315_FC816RLABXX_L1_HUMrutRGXDIA	1_I315_FC816RLABXX_L1_HUMrutRGXDIA
812MWAB	5_I186_FC812MWABXX_L8_HUMrutRGVDIA	5_I186_FC812MWABXX_L8_HUMrutRGVDIA
C81DE8AB	1_I288_FC81DE8ABXX_L3_HUMrutRGXDIA	1_I288_FC81DE8ABXX_L3_HUMrutRGXDIA
C81DB5AB	2_I329_FC81DB5ABXX_L6_HUMrutRGVDIA	2_I329_FC81DB5ABXX_L6_HUMrutRGVDIA
C819P0AB	8_I481_FC819P0ABXX_L5_HUMrutRGWDIA	8_I481_FC819P0ABXX_L5_HUMrutRGWDIA

```
params.index = "$baseDir/data/index.csv"

process foo {
    debug true
    input:
    tuple val(sampleId), file(read1), file(read2)

    script:
    """
    echo your_command --sample $sampleId --reads $read1 $read2
    """
}

workflow {
    Channel.fromPath(params.index) \
        | splitCsv(header:true) \
        | map { row-> tuple(row.sampleId, file(row.read1), file(row.read2)) } \
        | foo
}
```

Nextflow patterns

```
process foo {
  input:
  path x
  output:
  path 'file.fq'
  script:
  """
  < $x zcat > file.fq
  """
}

process bar {
  debug true
  input:
  path '*.fq'
  script:
  """
  cat *.fq | head -n 50
  """
}

workflow {
  Channel.fromPath("$baseDir/data/reads/*_1.fq.gz", checkIfExists: true) \
    | foo \
    | collect \
    | bar
}
```

Process all outputs altogether

https://github.com/adigenova/uohpmd/blob/main/code/Haddop_nextflow.ipynb

Preguntas?