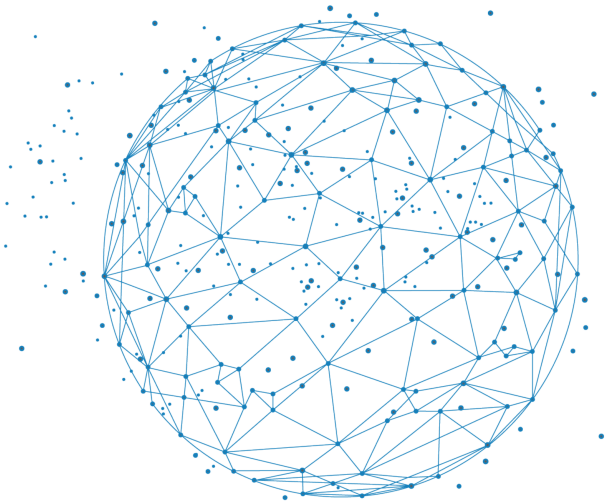


nextflow

Frédéric Jarlier
Julien Romejon
Philippe Hupé

Institut Curie,
CUBIC Plateforme,
PSL Research University,
CNRS UMR 144,
INSERM U900,
Mines ParisTech

IFB
INSTITUT FRANÇAIS DE BIOINFORMATIQUE

elixir
FRANCE

institut Curie

cnrs INRAE Inserm cea

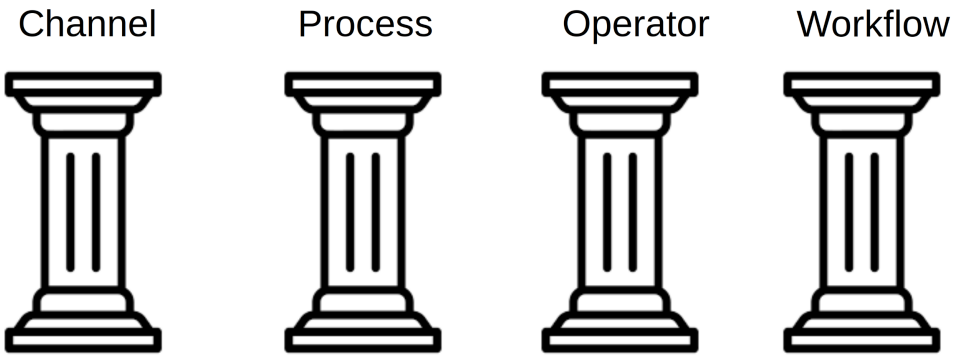
INVESTIR L'AVENIR

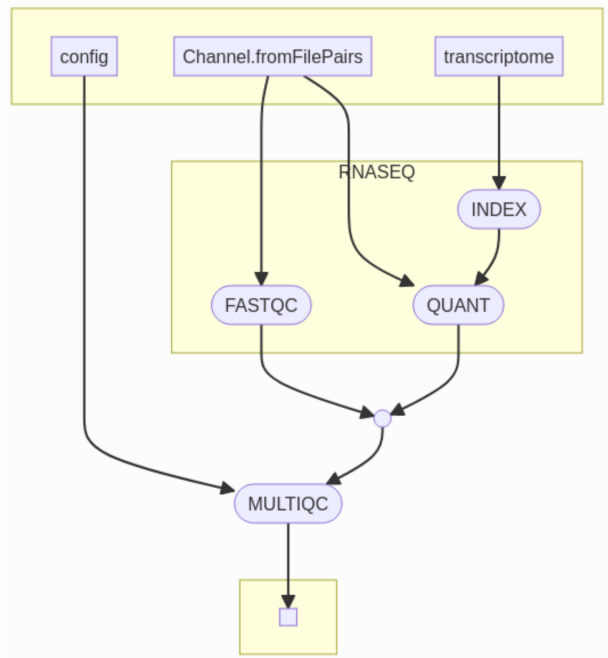
- Introduction
- Channels
- Operators
- Processes
- Configuration
- Profile
- Error Management
- Debugging
- Metrics

Introduction

- Nextflow is Data Flow Model programming language that helps for building complex worflows
- The idea is to chain multiple simple tasks like in *nix system with piping command
- Nextflow runs upon Java and extensively uses Groovy
- The elementary bricks of Nextflow are processes, channels, operators and workflows (since DSL2)

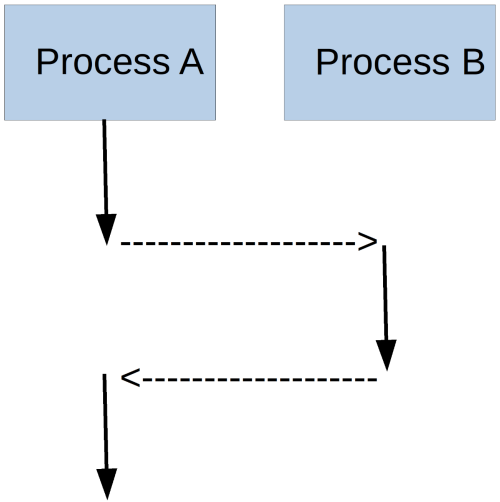
Introduction



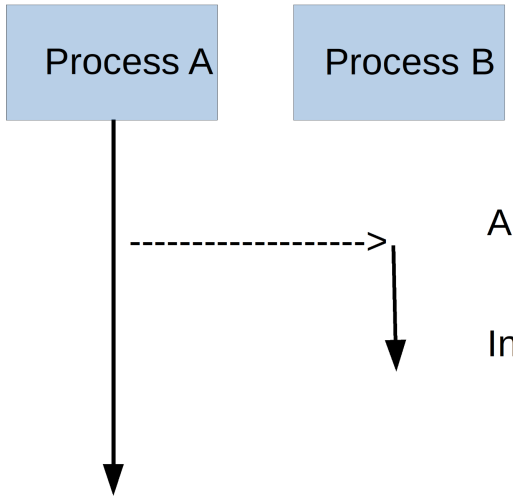


Channels

synchronous => blocking



asynchronous => non blocking



Advantage:

masking computing and communication

Inconvenient:

difficult to debug

difficult to develop

Channels

- Channels are 2 types:

  Queue channels:
  - queue channels are FIFO queues and block other input queue channels

  Value channels:
  - only one value and consumed repeatedly and don't block other input queue channels

- Channels Factory:

```
Channel.value("toto")                    Channel.of(1,2,3)

Channel.value([1,2,3])                   Channel.of([1,2,3])

Channel.fromList([1,2,3])                Channel.fromPath("/input/*.gz")

Channel.fromFilePairs('input/**.R{1,2}.fastq.gz')
```

Value Channel

```
process foo {

    input:
    val x

    output:
    path 'x.txt'

    """
    echo $x > x.txt
    """
}

workflow {
    result = foo(1)
    result.view()
}
```

Channel Factory

```
ch1 = Channel.of( 1, 3, 5, 7, 9 )
ch2 = Channel.of( [1, 3, 5, 7, 9] )
ch3 = Channel.of( [1, 2], [5,6], [7,9] )

Channel.fromList( ['a', 'b', 'c', 'd'] )
       .view()
```

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

** for directory recursion

example1.nf
example2.nf

Operators

- Operators manipulate channels.

- operators return queue channels or value channels

- With channels you can:
  - Filter (filter, first, unique)
  - combine (map, groupTuple, collect, flatten)
  - process text (splitCSV, splitFasta)
  - fork (multiMap, branch)
  - Math (count, min, max)

Operators

Branch

```
result = Channel.of(1, 2, 3, 40, 50)
    .branch {
        small: it < 10
        large: it > 10
}

result.small.view()
result.large.view()
```

Text

```
Channel.of( '10,20,30\n70,80,90' )
       .splitCsv()
       .view()
```

```
Channel.fromPath('misc/sample.fa')
       .splitFasta( by: 10 )
       .view()
```

example3.nf

Reduce

```
Channel.of( [1, 'A'], [1, 'B'], [2, 'C'], [1, 'C'], [2, 'A'] )
       .groupTuple()
       .view()
```

Math

```
Channel.of( 8, 6, 2, 5 )
       .min()
       .view()
```

example4.nf

Operators - closures

- Closures a block of code that can be passed as argument of an operator
- Closures are written between brackets: { put the code here }
- By default only 1 argument : it
- Multiple arguments can be passed

```
Channel.of(1,2,3,4,5)
       .map{it * it}
```

```
Channel.of(1,2,3,4,5)
       .map{ a -> a * a}
```

example4.nf

Skeleton

```
process name  {

  [directives]

  input:
    < process inputs >

  output:
    < process outputs >

  when:
    < condition >

  [script|shell|exec]:
  """
  < user script to be executed>
  """
}
```

Only one script
Script executed as a Bash
Script runs in the host environment
Prefer script
Prefer """ over '''

```
process doOtherThings  {

  script:
  """
    blast -db \${DB} -query.fa -outfmt 6 > blast_result
    cat blast_result | head -n ${MAX} | cut -f2 > top_hits
    blastdbcmd -db \${DB} -entry_batch top_hits > sequences
  """
}
```

\${DB} is a bash variable
${MAX} is a nextflow variable

exec is for groovy commands

Processes

- Processes contain the commands to execute.

Input can be:
- val
- file
- path
- env
- stdin
- tuple

Output can be:
- val
- path
- env
- stdout
- tuple

Files use separator (:), path don't

Tuple gather multiple values in a single channel

If inputs of different sizes the first empty stops the process

Chose your environment

```
process perlTask {
    script:
    """
    #!/usr/bin/perl
    print 'Hi there!' . '\n';
    """
}
```

```
process pythonTask {

    script:
    """
    #!/usr/bin/python
    X = 'Hello'
    Y = 'world'
    print("%s - %s", X,Y)
    """
}
```

example5.nf

Processes – Dry run

Prototype your process without real data

nextflow --stub-run

```
process Index {

  input:
    path transcriptome

  output:
    path 'index'

  script:
    """
    salmon index --thread $task.cpus -t $transcriptome -i index
    """

  stub:
    """
    mkdir index
    touch index/seq.bin
    touch index/info.json
    touch index/refseq.bin
    """
}
```