GET STARTED WITH

Paolo Di Tommaso

Bio in Docker Symposium - London

10 Nov 2015

WHAT NEXTFLOW IS

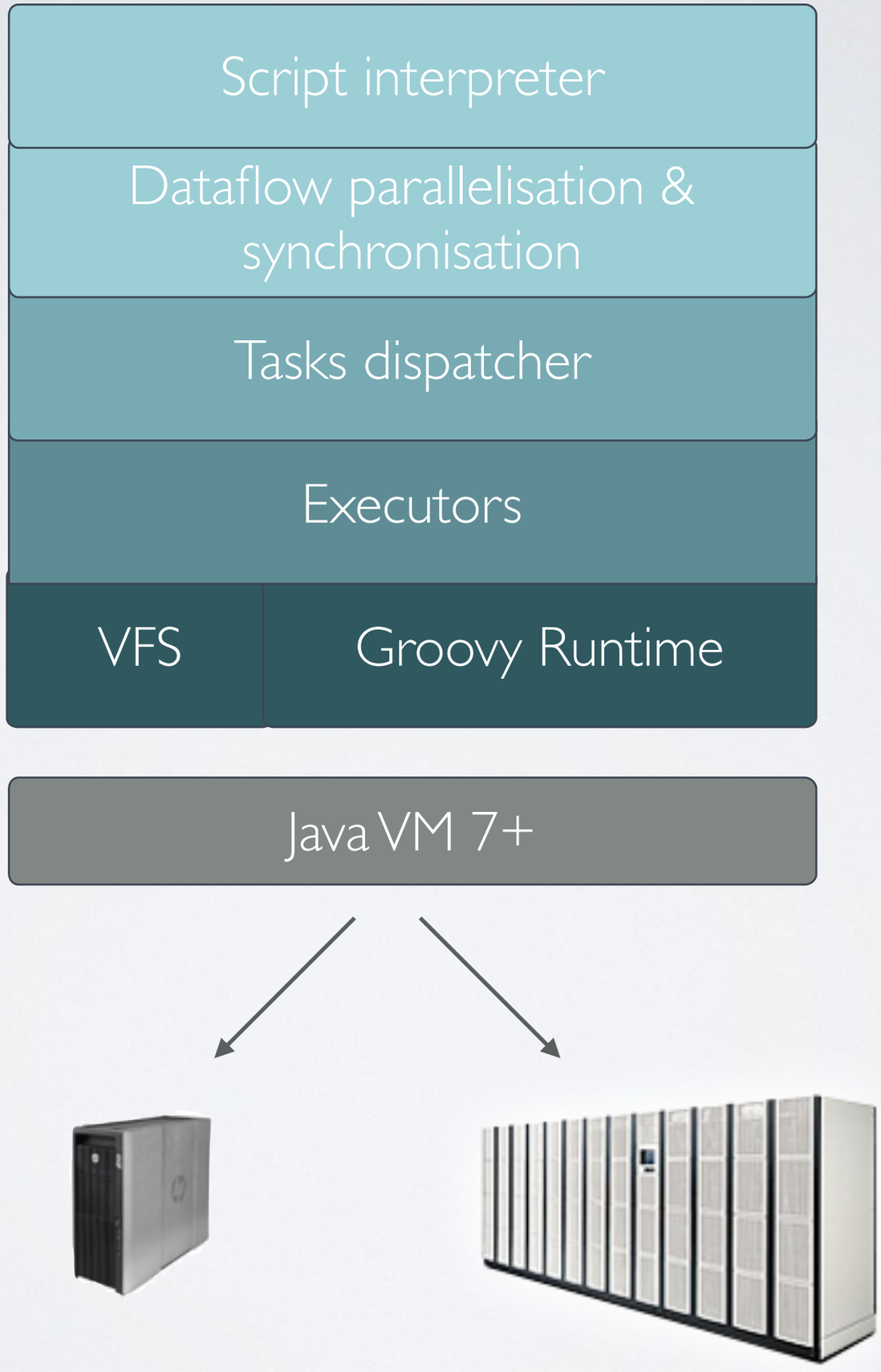
- A computing runtime which executes Nextflow pipeline scripts
- A programming DSL that simplify writing of highly parallel computational pipelines reusing your existing scripts and tools

NEXTFLOW DSL

- It is NOT a new programming language
- It extends the Groovy scripting language
- It provides a multi-paradigm programming environment

MULTI-PARADIGM

Imperative
Object-oriented programming
+
Declarative concurrency
Reactive-functional programming


Script interpreter
Dataflow parallelisation & synchronisation
Tasks dispatcher
Executors
VFS
Groovy Runtime
Java VM 7+

HOW TO INSTALL*

Use the following command:

```
wget -qO- get.nextflow.io | bash
```



```
nextflow
```

* It requires: Unix-like OS (Linux, OSX, etc) and Java 7/8

GET STARTED

Install Nextflow:

```
$ wget -qO- get.nextflow.io | bash
```

- or -

```
$ curl -fsSL get.nextflow.io | bash
```

DOWNLOAD EXAMPLES

Open a shell terminal and type:

```
$ git clone https://github.com/nextflow-io/examples.git
$ cd examples
$ echo 'docker.enabled=true' >> nextflow.config
$ docker pull nextflow/examples
```

THE BASIC

Variables and assignments

```
x = 1
y = 10.5
str = 'hello world!'
p = x; q = y
```

THE BASIC

Printing values

```
x = 1
y = 10.5
str = 'hello world!'

print x

print str

print str + '\n'

println str
```

THE BASIC

Printing values

```
x = 1
y = 10.5
str = 'hello world!'

print(x)

print(str)

print(str + '\n')

println(str)
```

MORE ON STRINGS

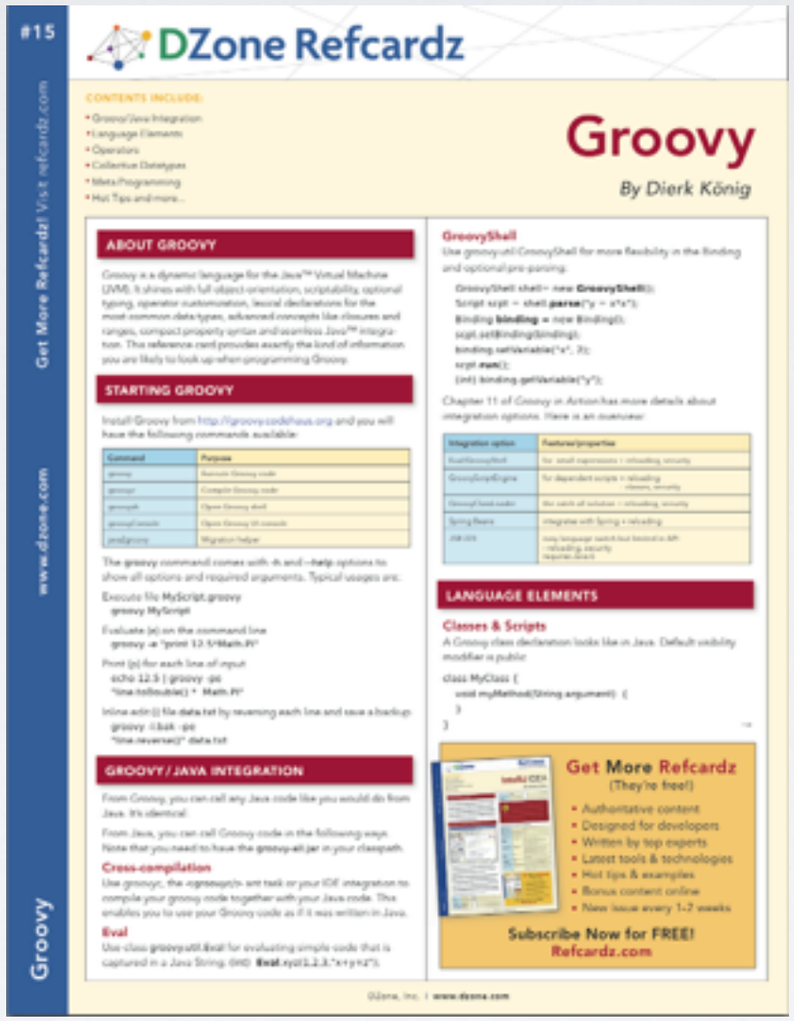
```
str = 'bioinformatics'
print str[0]

print "$str is cool!"
print "Current path: $PWD"
```

```
str = '''
      multi
      line
      string
      '''

str = """
      User: $USER
      Home: $HOME
      """
```

COMMON STRUCTURES & PROGRAMMING IDIOMS

- Data structures: Lists & Maps
- Control statements: if, for, while, etc.
- Functions and classes
- File I/O operations

6 PAGES PRIMER



http://refcardz.dzone.com/refcardz/groovy

MAIN ABSTRACTIONS

- Processes: run any piece of script
- Channels: data streams that allows processes to comunicate
- Operators: transform channels content

CHANNELS

- It connects two processes/operators
- Asynchronous unidirectional FIFO queue
- Write operation is NOT blocking
- Read operation is blocking
- Once an item is read is removed from the queue

CHANNELS

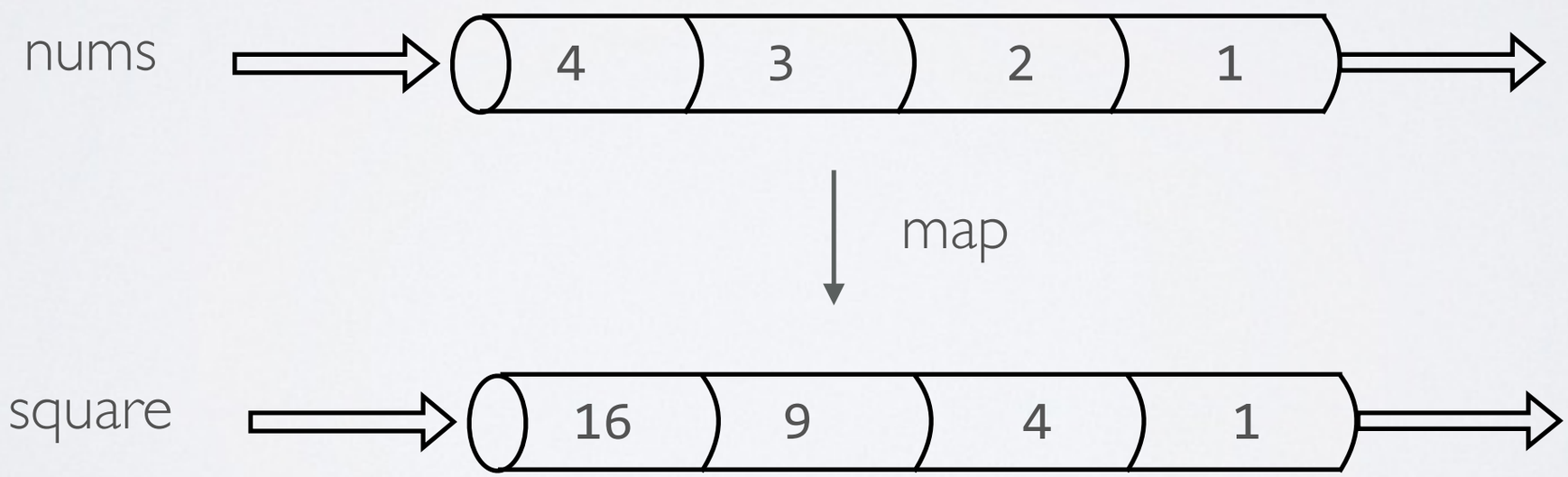
```
my_channel = Channel.create()

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

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

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

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 * it }
```



OPERATORS CHAINING

```
Channel.from(1,2,3,4)
        .map { it -> [it, it*it] }
        .subscribe { num, sqr -> println "Square of: $num is $sqr" }
```

```
// it prints
Square of: 1 is 1
Square of: 2 is 4
Square of: 3 is 9
Square of: 4 is 16
```

SPLIT FASTA FILE(S)

```
Channel.fromPath('/some/path/fasta.fa')
       .splitFasta()
       .println()
```

```
Channel.fromPath('/some/path/fasta.fa')
       .splitFasta(by: 3)
       .println()

Channel.fromPath('/some/path/*.fa')
       .splitFasta(by: 3)
       .println()
```

SPLITTING OPERATORS

You can split text object or files using the splitting methods:

- splitText - line by line
- splitCsv - comma separated values format
- splitFasta - by FASTA sequences
- splitFastq - by FASTQ sequences

EXAMPLE 1

- Split a FASTA file in sequence
- Parse a FASTA file and count number of sequences matching specified ID

EXAMPLE 1

```
$ nextflow run channel_split.nf
```

```
$ nextflow run channel_filter.nf
```

PROCESS

```
str = Channel.from('hello', 'hola', 'bonjour', 'ciao')

process sayHello {

    input:
    val str

    output:
    stdout into result

    script:
    """
    echo $str world!
    """
}

result.subscribe { print it }
```

PROCESS INPUTS

```
process procName {

  input:
    <input type> <name> [from <source channel>] [attributes]

    """
    <your script>
    """

}
```

PROCESS INPUTS

```
process procName {

  input:
    val  x from ch_1
    file y from ch_2
    file 'data.fa' from ch_3
    stdin from from ch_4
    set (x, 'file.txt') from ch_5


  """
  <your script>
  """

}
```

PROCESS INPUTS

```
proteins = Channel.fromPath( '/some/path/data.fa' )


process blastThemAll {
  input:
  file 'query.fa' from proteins

  """
  blastp -query query.fa -db nr
  """

}
```

PROCESS OUTPUTS

```
process randomNum {

    output:
    file 'result.txt' into numbers

    '''
    echo $RANDOM > result.txt
    '''

}


numbers.println { "Received: " + it.text }
```

EXAMPLE 2

- Execute a process running a BLAST job given an input file

- Execute a BLAST job printing the produced output

EXAMPLE 2

```
$ nextflow run process_input.nf

$ nextflow run process_output.nf
```

USE YOUR FAVOURITE LANG

```
process pyStuff {
    """
    #!/usr/bin/env python

    x = 'Hello'
    y = 'world!'
    print "%s - %s" % (x,y)
    """
}
```

PIPELINES PARAMETERS

Simply declares some variables prefixed by params

```
params.p1 = 'alpha'
params.p2 = 'beta'
:
```

When launching your script you can override the default values

```
$ nextflow run <script file> --p1 'delta' --p2 'gamma'
```

EXAMPLE 3

Split a FASTA file and execute a BLAST query for each chunk

EXAMPLE 3

```
$ nextflow run split_fasta.nf
```

```
$ nextflow run split_fasta.nf --chunkSize 2
```

```
$ nextflow run split_fasta.nf --chunkSize 2 --query data/p\*.fa
```

COLLECT FILE

The operator collectFile allows to gather items produced by upstream processes

Collect all items to a single file

```
my_results.collectFile(name:'result.txt')
```

COLLECT FILE

The operator collectFile allows to gather items produced by upstream processes

Collect the items and group them into files having a names defined by a grouping criteria

```
my_items.collectFile(storeDir:'path/name') {

    def key = get_a_key_from_the_item(it)
    def content = get_the_item_value(it)
    [ key, content ]

}
```

EXAMPLE 4

Execute many BLAST queries and gather the results to a single file

```
$ nextflow run split_and_gather.nf
```

EXAMPLE 5

Gather results using a process a user process

```
$ nextflow run split_and_gather.nf
```

EXAMPLE 6

Groups read-pairs and process them using a process

```
$ nextflow run group_read_pairs.nf
```

CONFIG FILE

- Pipeline configuration can be externalised to a file named `nextflow.config`
  - parameters
  - environment variables
  - required resources (mem, cpus, queue, etc)
  - modules/containers
  - execution profiles

CONFIG FILE

```
params.p1 = 'alpha'
params.p2 = 'beta'

env.VAR_1 = 'some_value'
env.CACHE_4_TCOFFEE = '/some/path/cache'
env.LOCKDIR_4_TCOFFEE = '/some/path/lock'

process.executor = 'sge'
```

CONFIG FILE

Alternate syntax (almost) equivalent

```
params {
  p1 = 'alpha'
  p2 = 'beta'
}

env {
  VAR_1 = 'some_value'
  CACHE_4_TCOFFEE = '/some/path/cache'
  LOCKDIR_4_TCOFFEE = '/some/path/lock'
}

process {
  executor = 'sge'
}
```

HOW USE DOCKER

Specify in the config file the Docker image to use

```
process {
    container = 'your/image:tag'
}
```

Add the with-docker option when launching it

```
$ nextflow run <script name> -with-docker
```

MULTIPLE CONTAINERS

Specify in the config file the Docker image to use

```
process {
    $foo {
        container = 'your/image:tag'
    }

    $bar {
        container = 'another/image:latest'
    }

}
```

HOW USE A CLUSTER

Define the CRG executor in `nextflow.config`

```
// default properties for any process
process {
   executor = 'sge'  // other: lsf, pbs, slurm
   queue = 'short'
   cpus = 2
   memory = '4GB'
   scratch = true
}
```

PROCESS RESOURCES

```
// default properties for any process
process {
    executor = 'sge'
    queue = 'short'
    scratch = true
}

// cpus for process 'foo'
process.$foo.cpus = 2

// resources for 'bar'
process.$bar.queue = 'long'
process.$bar.cpus = 4
process.$bar.memory = '4GB'
```

CONFIG PROFILES

```
profiles {

    standard {
        process.executor = 'local'
    }

    crg {
        process.executor = 'sge'
        process.queue = 'long'
        process.memory = '10GB'
        process.container = 'image/name'
    }

    ebi {
        process.executor = 'lsf'
        process.queue = 'bio16'
        process.module = 'ncbi-blast/2.2.27'
    }

}
```

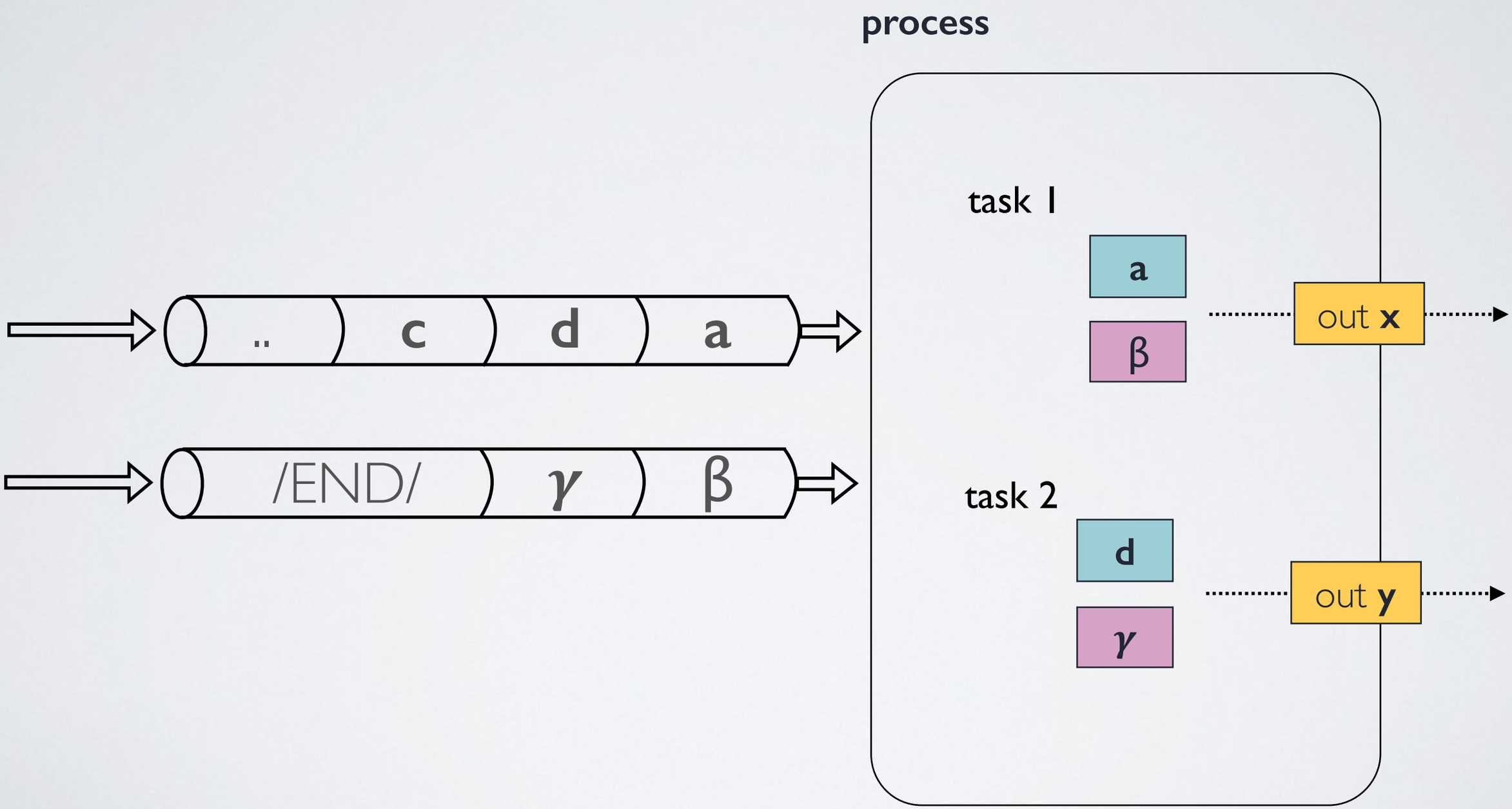
CONFIG PROFILES

```
$ nextflow run <project> -profile crg|ebi
```

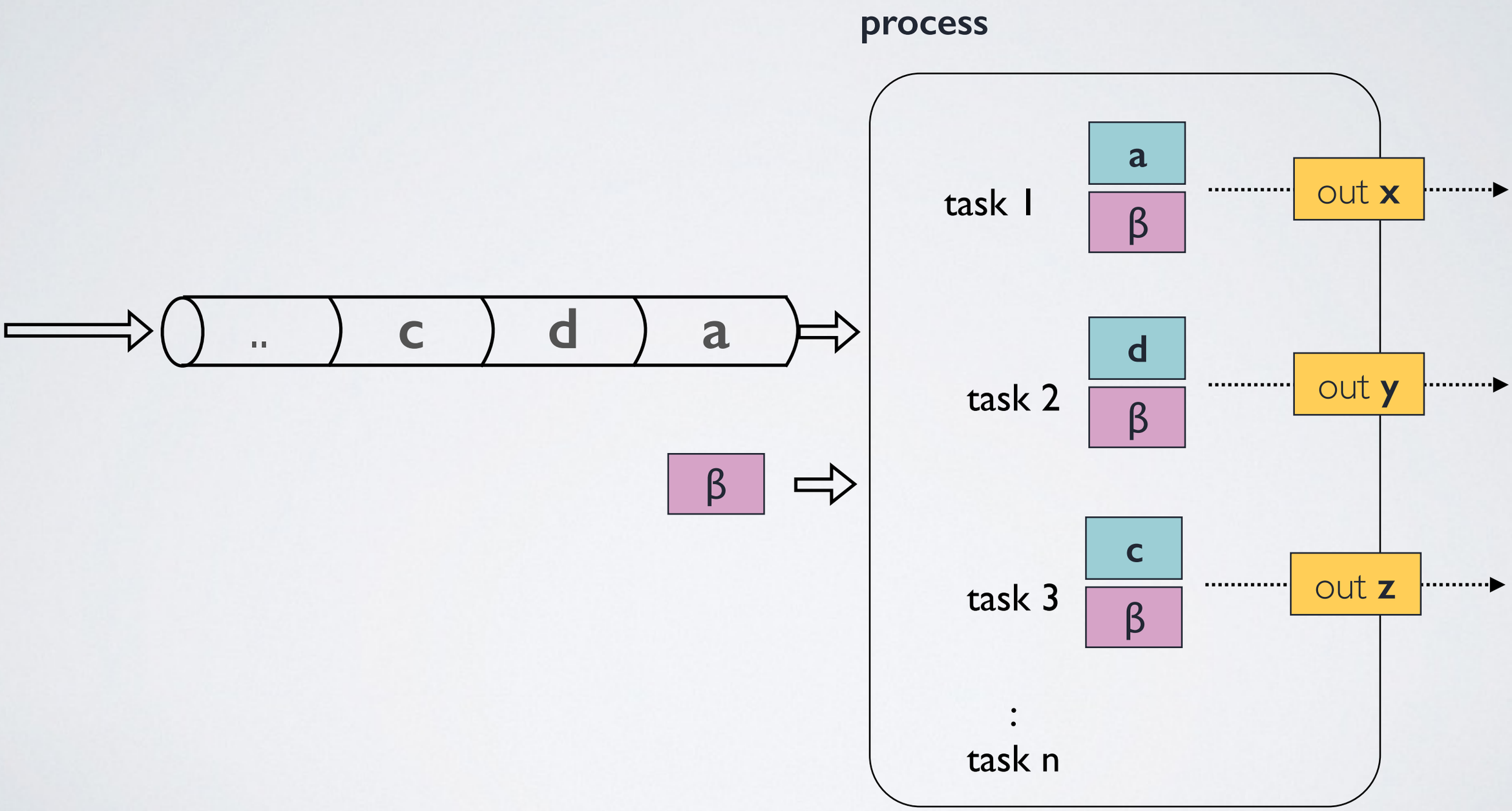
PUBLISH YOUR PIPELINE

- Create a Github repo (or Gitlab/BitBucket)
- Name the pipeline script `main.nf`
- Create a `nextflow.config` file
- Put other scrips into a folder named `bin`
- Create a Docker image for binary dependencies

UNDERSTANDING MULTIPLE INPUTS



UNDERSTANDING MULTIPLE INPUTS



LINKS

project home

http://nextflow.io

tutorials

https://github.com/nextflow-io/examples

chat

https://gitter.im/nextflow-io/nextflow