COMPUTATIONAL PIPELINES WITH NEXTFLOW & DOCKER CONTAINERS

Paolo Di Tommaso, CRG

22 November 2016

nextflow

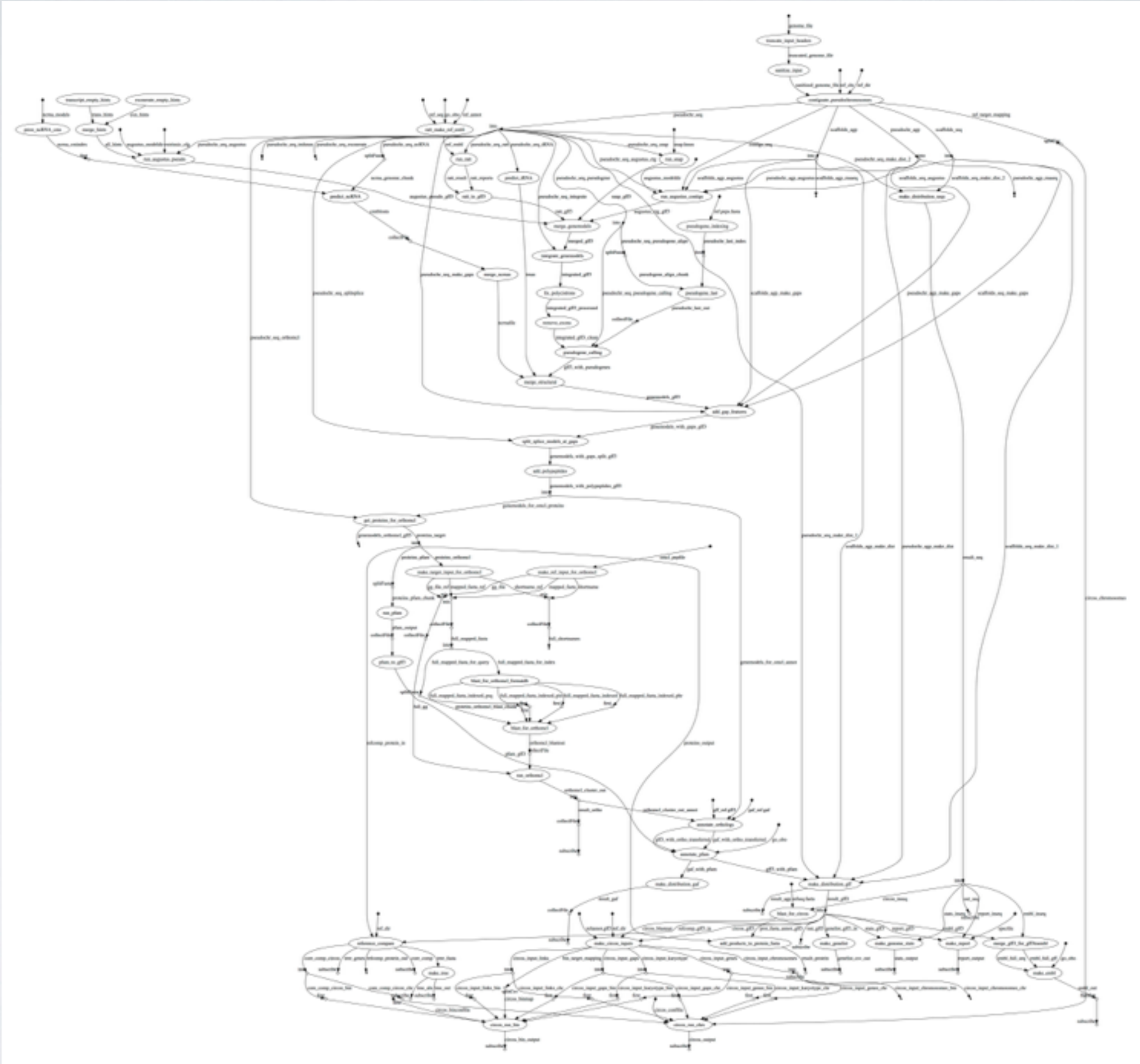
CRG Centre for Genomic Regulation

AGENDA

- Introduction to Nextflow
- Hands-on Nextflow tutorial
- Break (~ 11.15 - 11.30)
- Introdution to Docker containers
- Hands-on Docker tutorial
- Deployment in the cluster and the cloud
- Exercise, write a small pipeline

BASH PIPELINE

```
#!/bin/bash
blastp -query sampla.fasta -outfmt 6 \
    | head -n 10 \
    | cut -f 2 \
    | blastdbcmd -entry - > sequences.txt
```

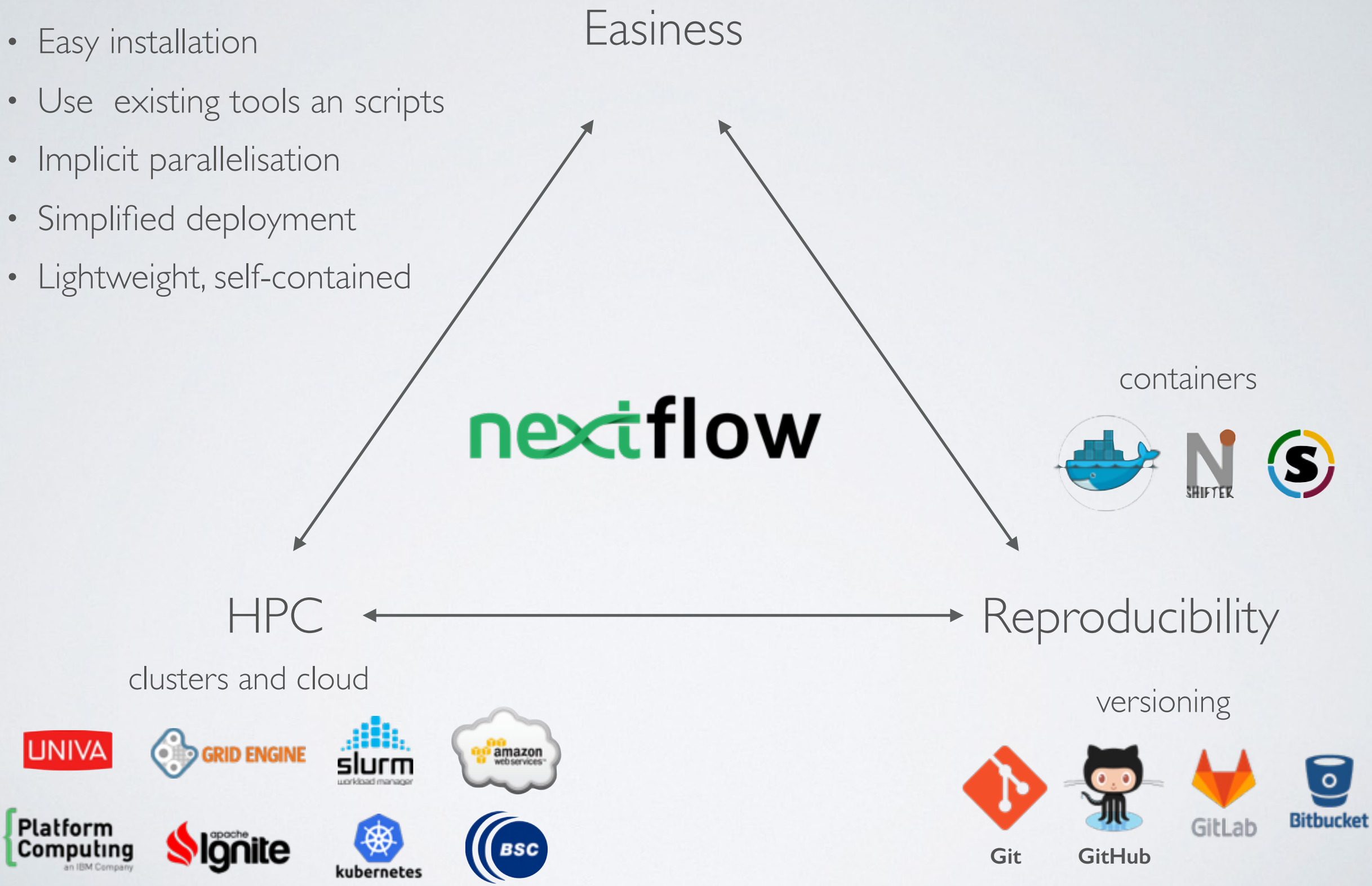


* Companion parassite genome annotation pipeline, Steinbiss et al., DOI: 10.1093/nar/gkw292

CHALLENGES

- Tame complexity
- Simplify parallelisation
- Portability & scalability
- Enable reproduciblity
- Fast prototyping

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



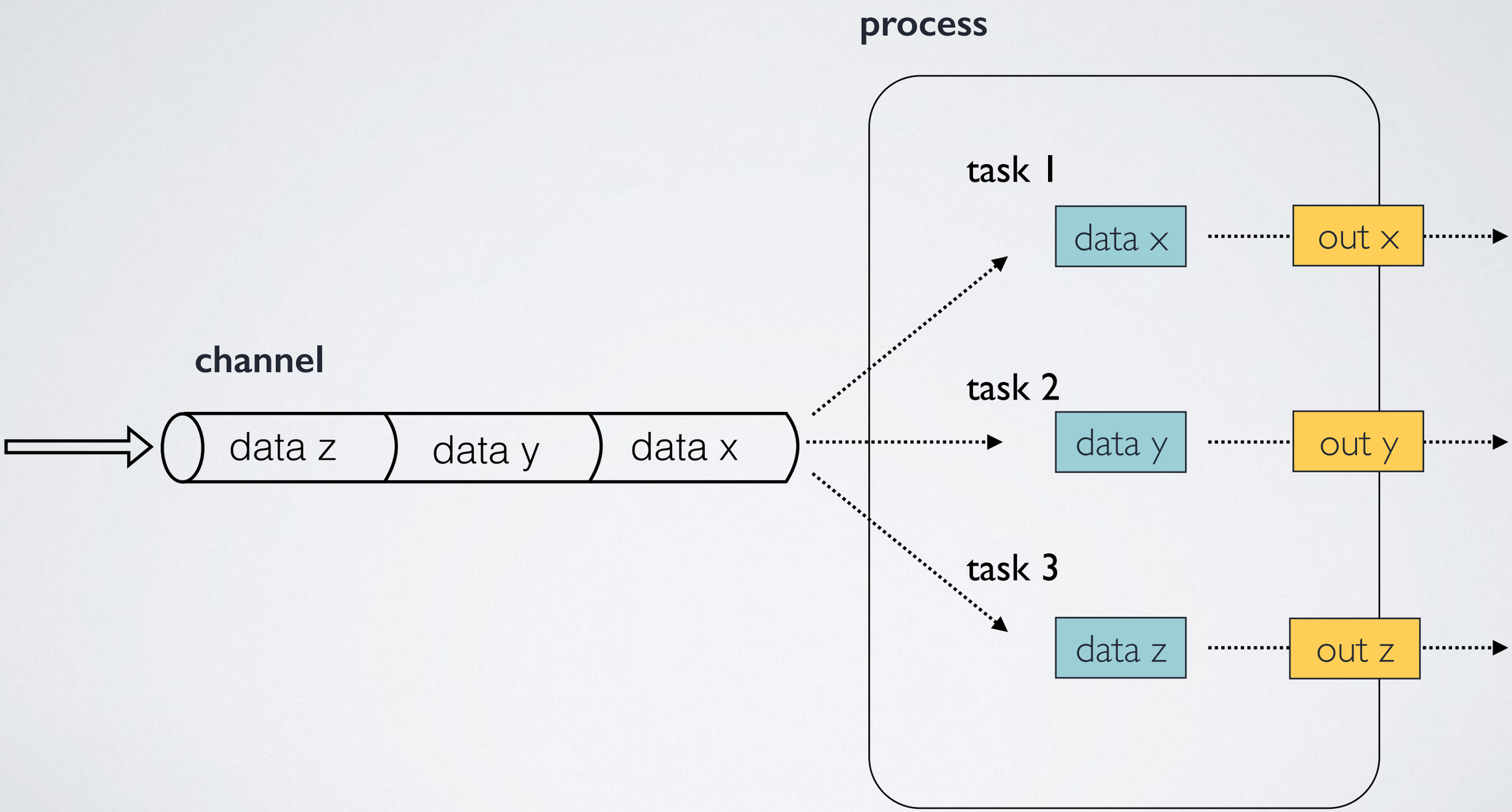
HOW IT WORKS

- Nextflow defines a Domain Specific Language (DSL) for computational pipelines
- It's a superset of Groovy programming language
- Three main abstractions:
  - *process*: executes user code
  - *channels*: connects processes
  - *operators*: transforms channel contents

PROCESS DEFINITION

```
process foo {

    input:
    file 'sample.fasta' from fasta_files

    output:
    file 'sequences.txt' into result_files

    script:
    """
    #!/bin/bash

    blastp -query sampla.fasta -outfmt 6 \
        | head -n 10 \
        | cut -f 2 \
        | blastdbcmd -entry - > sequences.txt
    """
}
```

HOW PARALLELISATION WORKS



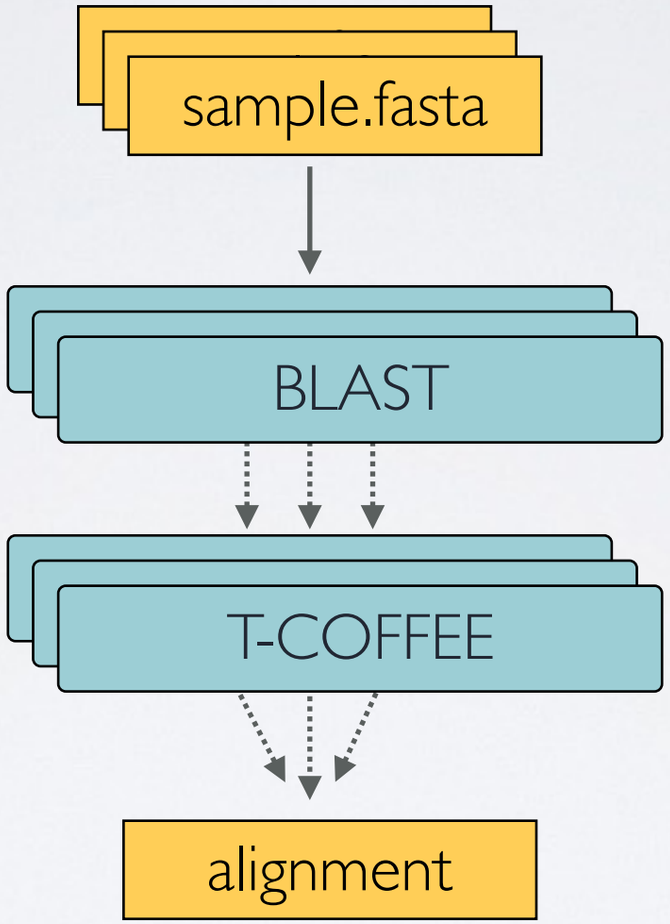
WHAT A SCRIPT LOOKS LIKE

```
sequences = Channel.fromPath("/data/sample.fasta")

process blast {
    input:
    file 'in.fasta' from sequences
    output:
    file 'out.txt' into blast_result

    """
    blastp -query in.fasta -outfmt 6 | cut -f 2 | \
    blastdbcmd -entry_batch - > out.txt
    """
}

process align {
  input:
  file all_seqs from blast_result
  output:
  file align_result

  """
  t_coffee $all_seqs 2>&- | tee align_result
  """
}

blast_result.collectFile(name: 'final_alignment')
```

IMPLICIT PARALLELISM

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

process blast {
    input:
    file 'in.fasta' from sequences
    output:
    file 'out.txt' into blast_result

    """
    blastp -query in.fasta -outfmt 6 | cut -f 2 | \
    blastdbcmd -entry_batch - > out.txt
    """
}

process align {
  input:
  file all_seqs from blast_result
  output:
  file align_result

  """
  t_coffee $all_seqs 2>&- | tee align_result
  """
}

blast_result.collectFile(name: 'final_alignment')
```

IMPLICIT PARALLELISM



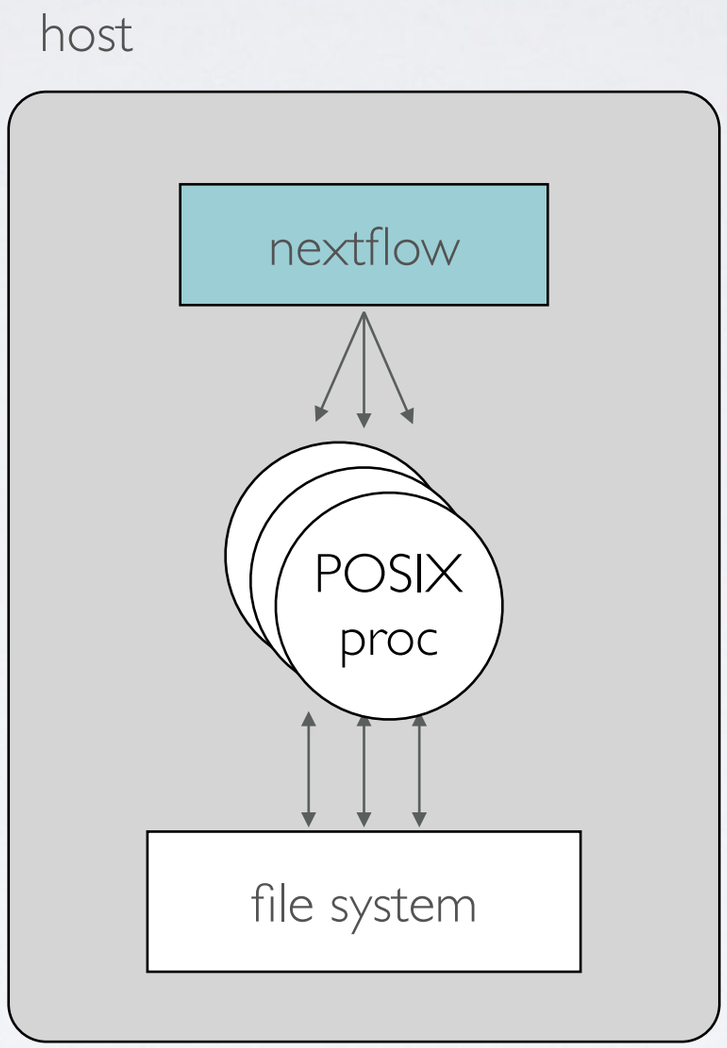
BENEFITS

- High-level declarative parallelisation abstraction
- Portable across different platforms
- Isolates task dependencies with Docker containers

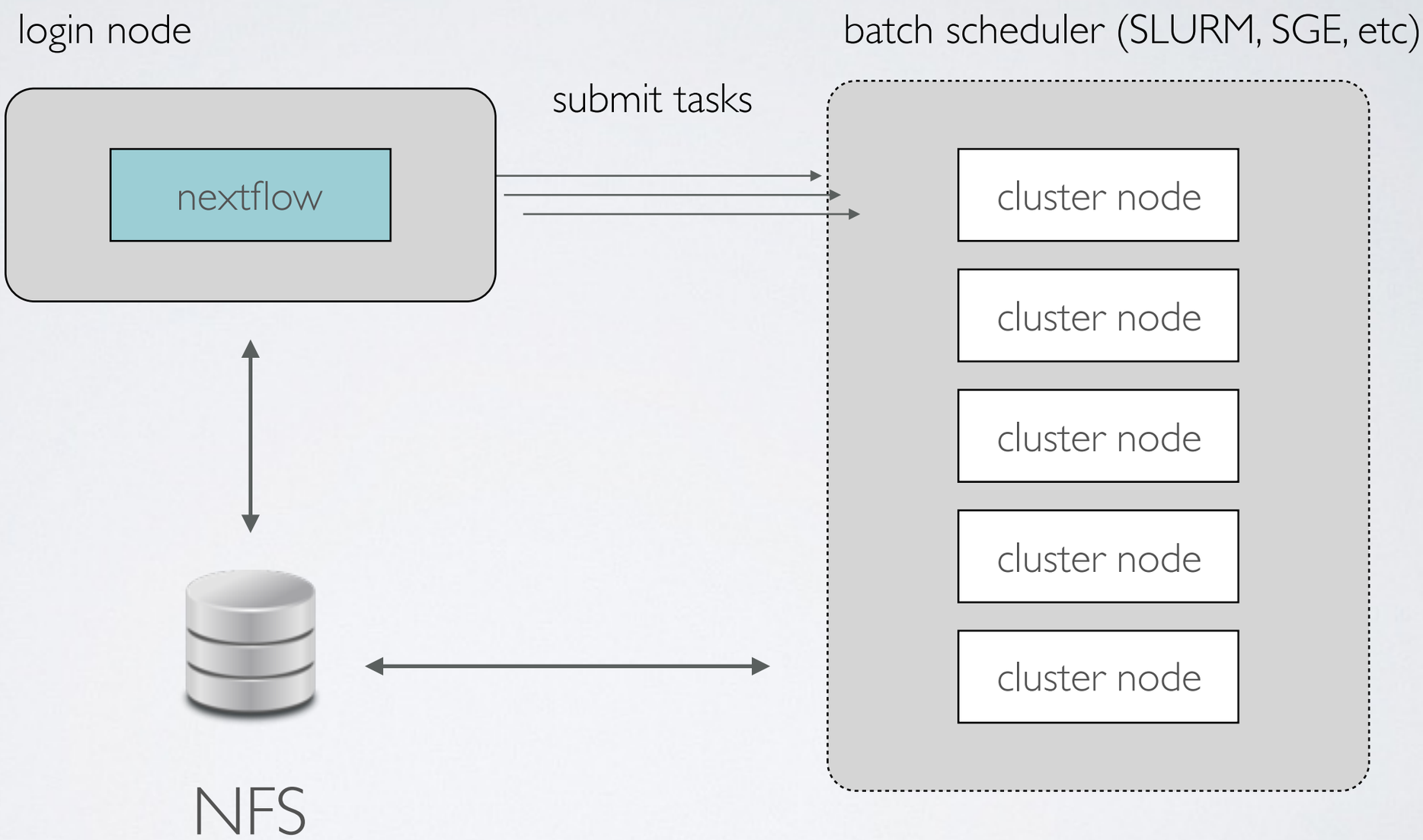
CONFIGURATION FILE

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

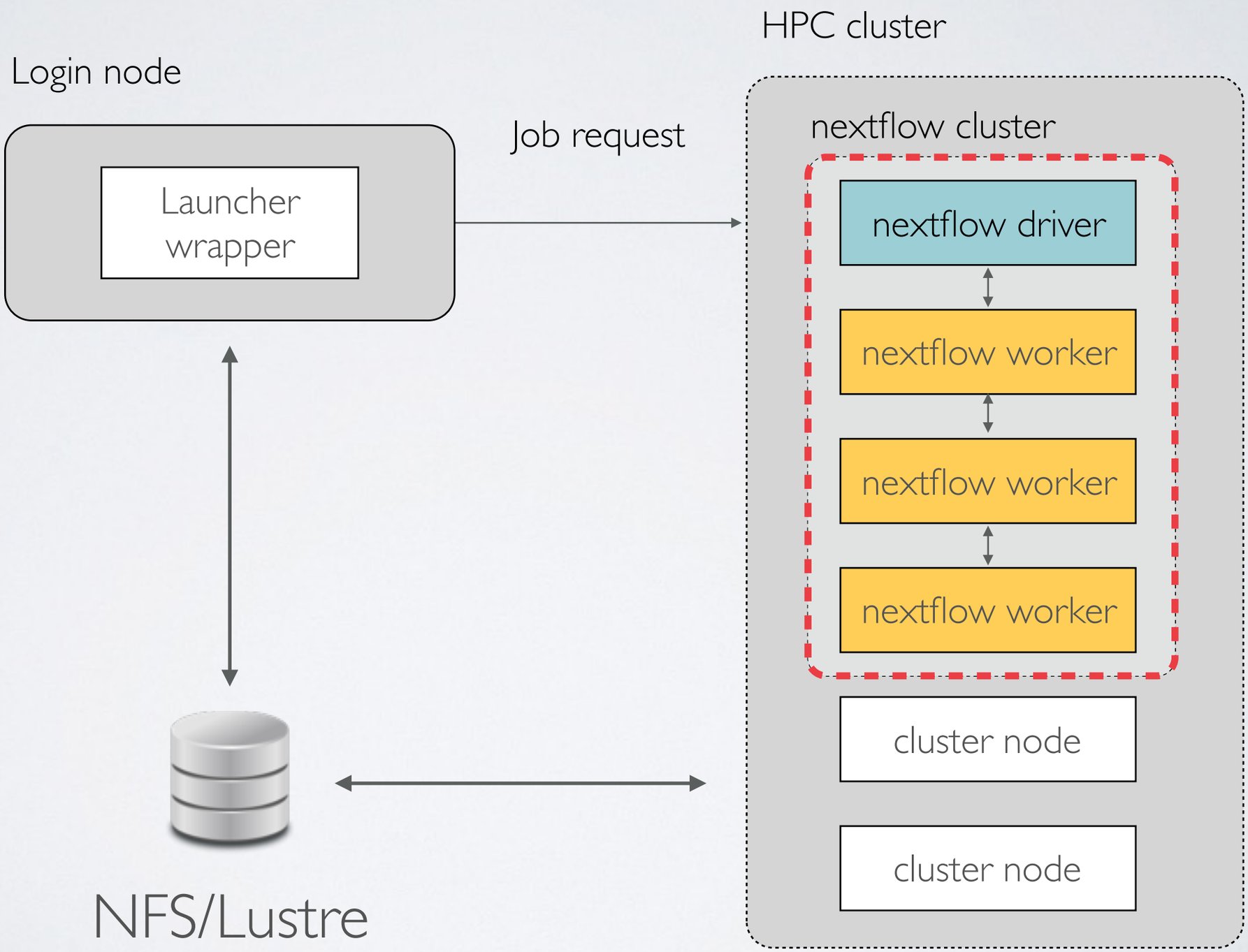
LOCAL EXECUTOR



CLUSTER EXECUTION



DISTRIBUTED EXECUTION



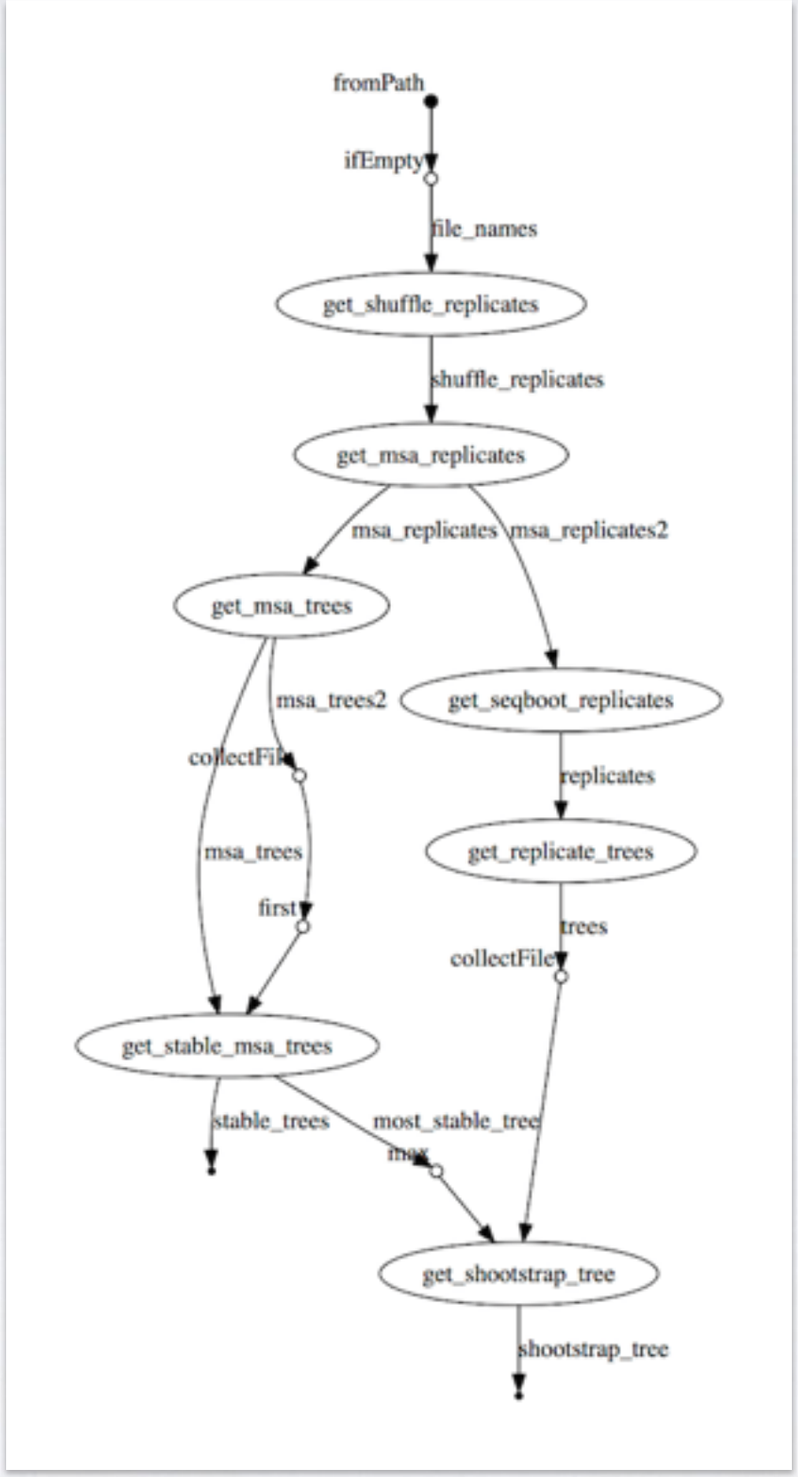
ERROR HANDLING

- Continuous task check-points
- Resume from the last successfully executed step
- Multiple error handling strategies: stop, ignore, retry
- Update resources requirement on failure

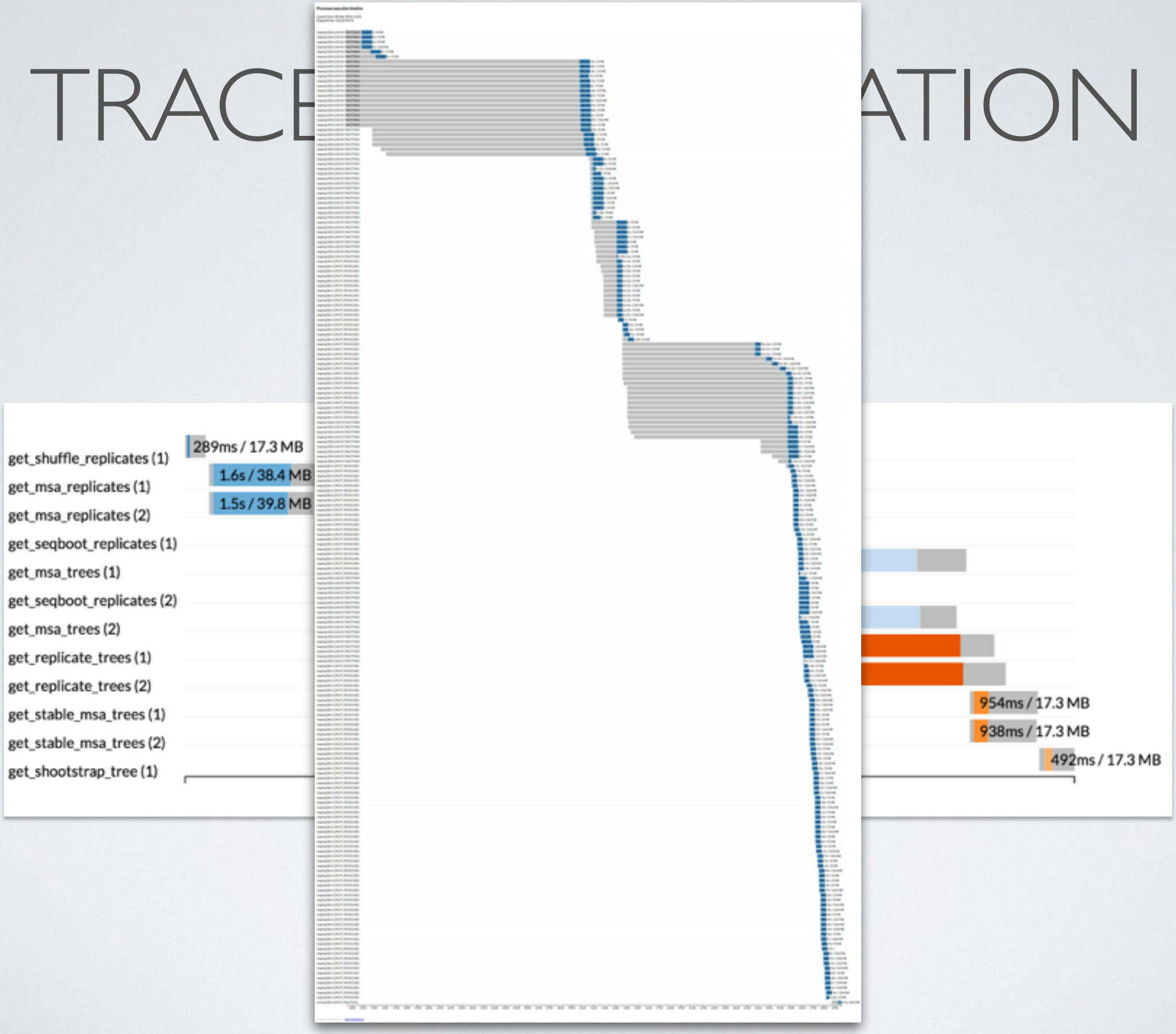
TRACE & VISUALISATION

- Trace tasks runtime metrics such as cpus, memory used, real-time, wall-time, etc.
- Create provenance report including tasks information such as name, container, script executed, work directory, etc.

TRACE & VISUALISATION



TRACE ATION


get_shuffle_replicates (1)
get_msa_replicates (1)
get_msa_replicates (2)
get_seqboot_replicates (1)
get_msa_trees (1)
get_seqboot_replicates (2)
get_msa_trees (2)
get_replicate_trees (1)
get_replicate_trees (2)
get_stable_msa_trees (1)
get_stable_msa_trees (2)
get_shootstrap_tree (1)
289ms / 17.3 MB
1.6s / 38.4 MB
1.5s / 39.8 MB
954ms / 17.3 MB
938ms / 17.3 MB
492ms / 17.3 MB

WHO IS USING NEXTFLOW?

HANDS-ON

https://goo.gl/9ffynC

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)
```

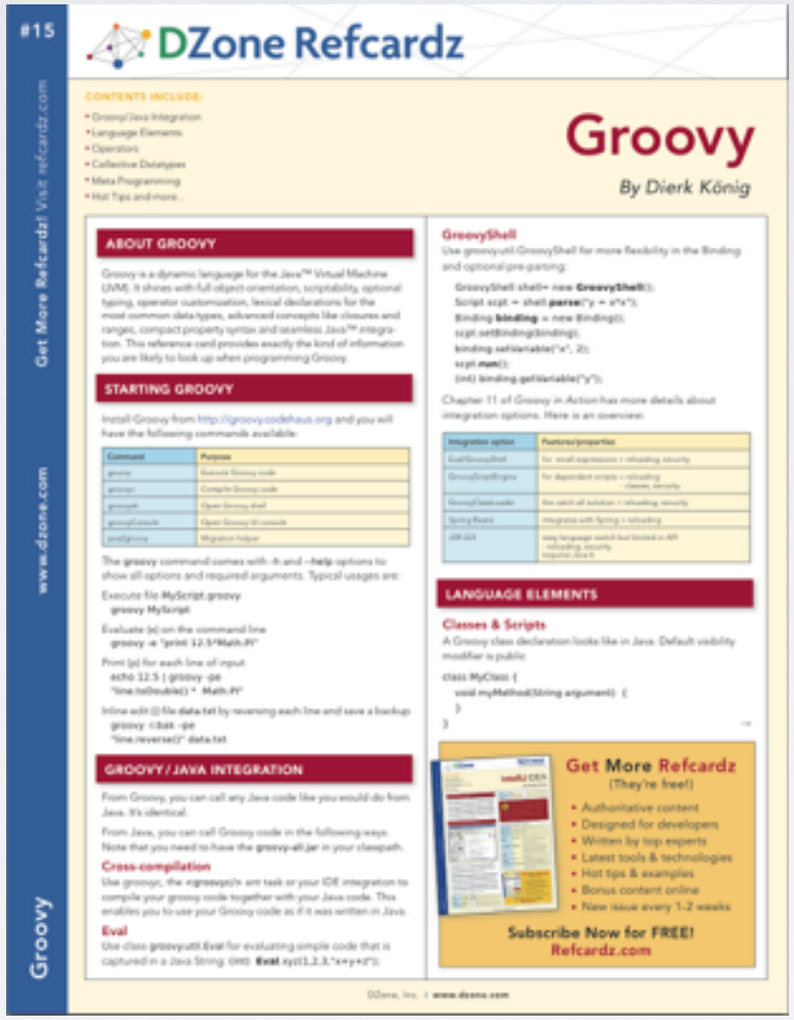
MORE ON STRINGS

```
str = 'groovy'
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



#15

DZone Refcardz

CONTENTS INCLUDE:

- Groovy/Java Integration
- Language Elements
- Operators
- Collective Datatypes
- Meta Programming
- Hot Tips and more...

Groovy

By Dierk König

ABOUT GROOVY

Groovy is a dynamic language for the Java™ Virtual Machine (JVM). It shines with full object orientation, scriptability, optional typing, operator customization, lexical declarations for the most common data types, advanced concepts like closures and ranges, compact property syntax and seamless Java™ integration. This reference card provides exactly the kind of information you are likely to look up when programming Groovy.

STARTING GROOVY

Install Groovy from http://groovy.codehaus.org and you will have the following commands available:

Command	Purpose
groovy	Execute Groovy code
groovyc	Compile Groovy code
groovysh	Open Groovy shell
groovyConsole	Open Groovy UI console
java2groovy	Migration helper

The **groovy** command comes with **-h** and **--help** options to show all options and required arguments. Typical usages are:

Execute file **MyScript.groovy**

```
groovy MyScript
```

Evaluate (**e**) on the command line

```
groovy -e "print 12.5*Math.PI"
```

Print (**p**) for each line of input

```
echo 12.5 | groovy -pe
"line.toDouble() * Math.PI"
```

Inline edit (**i**) file **data.txt** by reversing each line and save a backup

```
groovy -i.bak -pe
"line.reverse()" data.txt
```

GROOVY / JAVA INTEGRATION

From Groovy, you can call any Java code like you would do from Java. It's identical.

From Java, you can call Groovy code in the following ways. Note that you need to have the **groovy-all.jar** in your classpath.

Cross-compilation

Use groovyc, the **<groovyc/>** ant task or your IDE integration to compile your groovy code together with your Java code. This enables you to use your Groovy code as if it was written in Java.

Eval

Use class **groovy.util.Eval** for evaluating simple code that is captured in a Java String. EXD: **Eval.xy(1,2,3,"x+y+z");**

GroovyShell

Use groovy.util.GroovyShell for more flexibility in the Binding and optional pre-parsing:

```
GroovyShell shell= new GroovyShell();
Script scpt = shell.parse("y = x*x");
Binding binding = new Binding();
scpt.setBinding(binding);
binding.setVariable("x", 2);
scpt.run();
(int) binding.getVariable("y");
```

Chapter 11 of *Groovy in Action* has more details about integration options. Here is an overview:

Integration option	Features/properties
GroovyShell	for small expressions + reloading, security
GroovyScriptEngine	for dependent scripts + reloading + classes, security
GroovyClassLoader	the catch all solution + reloading, security
Spring Beans	integrate with Spring + reloading
JSR-223	new language switch but limited to API + reloading, security requires Java 6

LANGUAGE ELEMENTS

Classes & Scripts

A Groovy class declaration looks like in Java. Default visibility modifier is public:

```
class MyClass {
  void myMethod(String argument) {
  }
}
```

Get More Refcardz
(They're free!)
- Authoritative content
- Designed for developers
- Written by top experts
- Latest tools & technologies
- Hot tips & examples
- Bonus content online
- New issue every 1-2 weeks

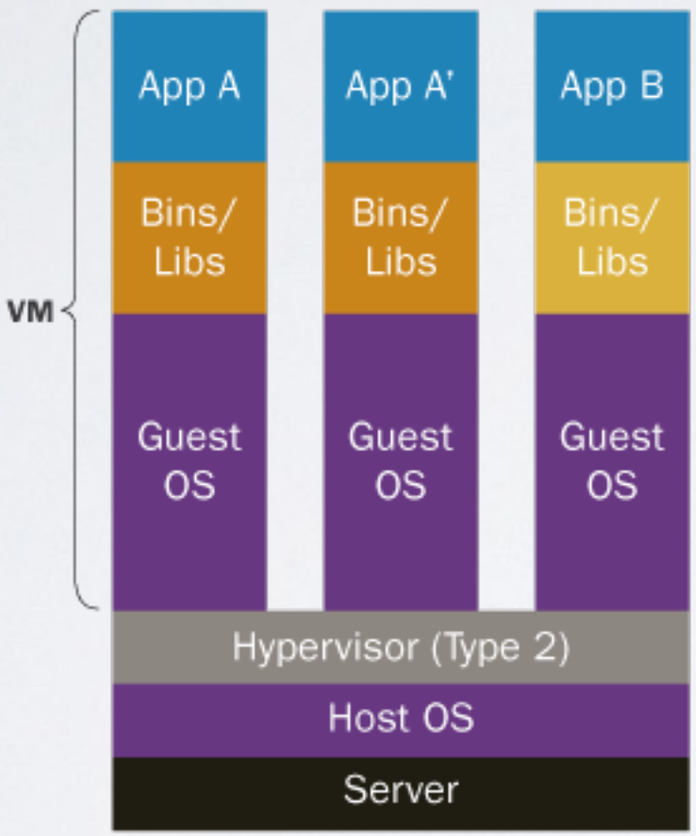
Subscribe Now for FREE!
Refcardz.com

Get More Refcardz! Visit refcardz.com
www.dzone.com

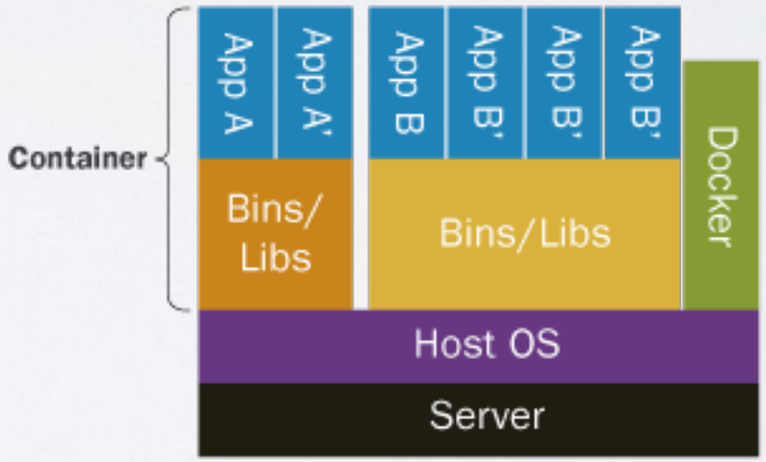
https://dzone.com/refcardz/groovy

CONTAINERS ARE THE THIRD BIG WAVE IN VIRTUALISATION TECHNOLOGY

VM vs CONTAINER



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



BENEFITS

- Smaller images (~100MB)
- Fast instantiation time (~1sec)
- Almost native performance
- Easy to build, publish, share and deploy
- Transparent build process

WHY CONTAINERS MATTER?

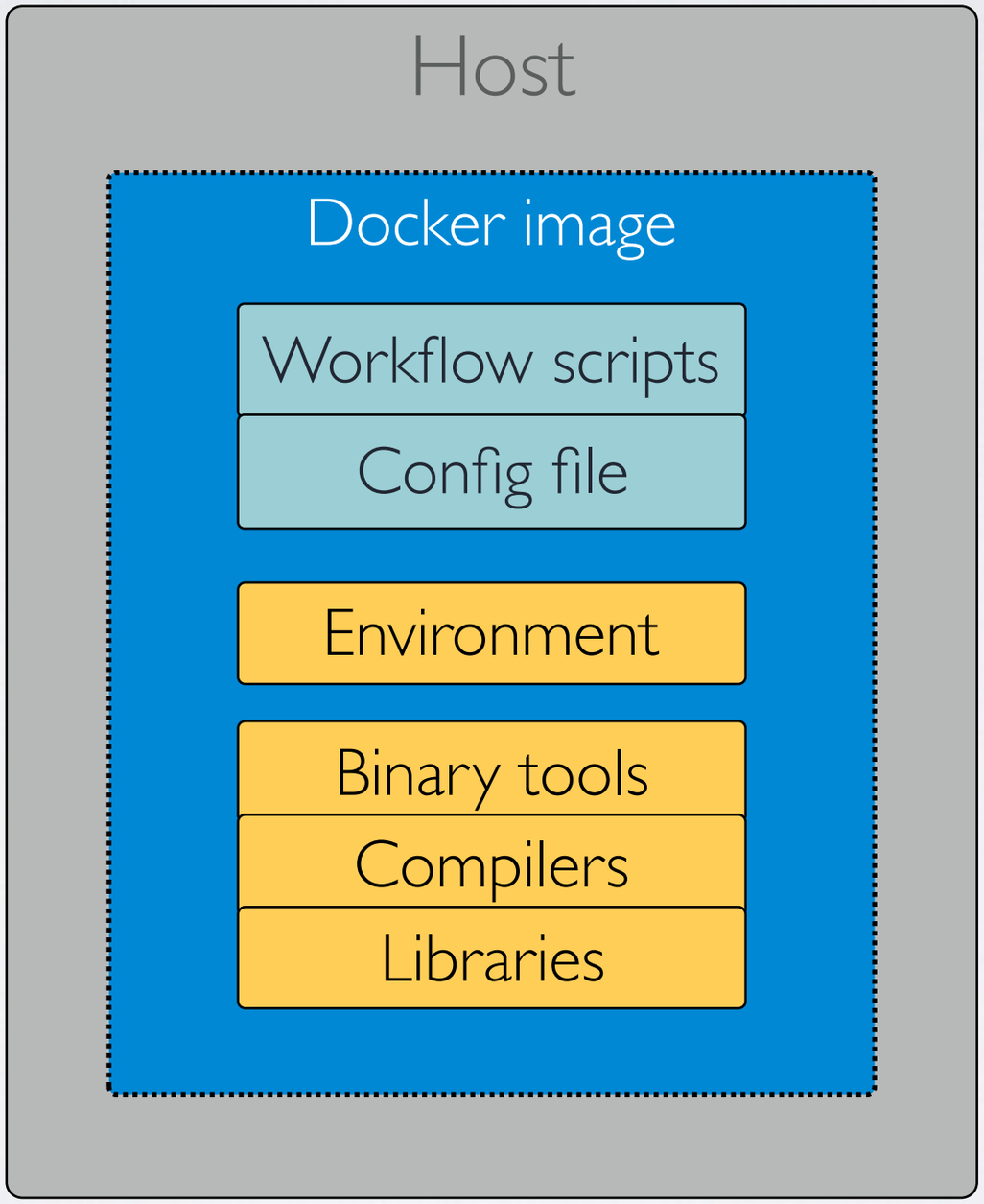
- Allows dependencies to be isolated in a easy and portable runtime
- Enables reproducible cross-platforms deployment
- Makes it possible to create a ready-to-run reference implementation of your application
- Preserves against software decommission/obsolescence

DOCKER IN A NUTSHELL

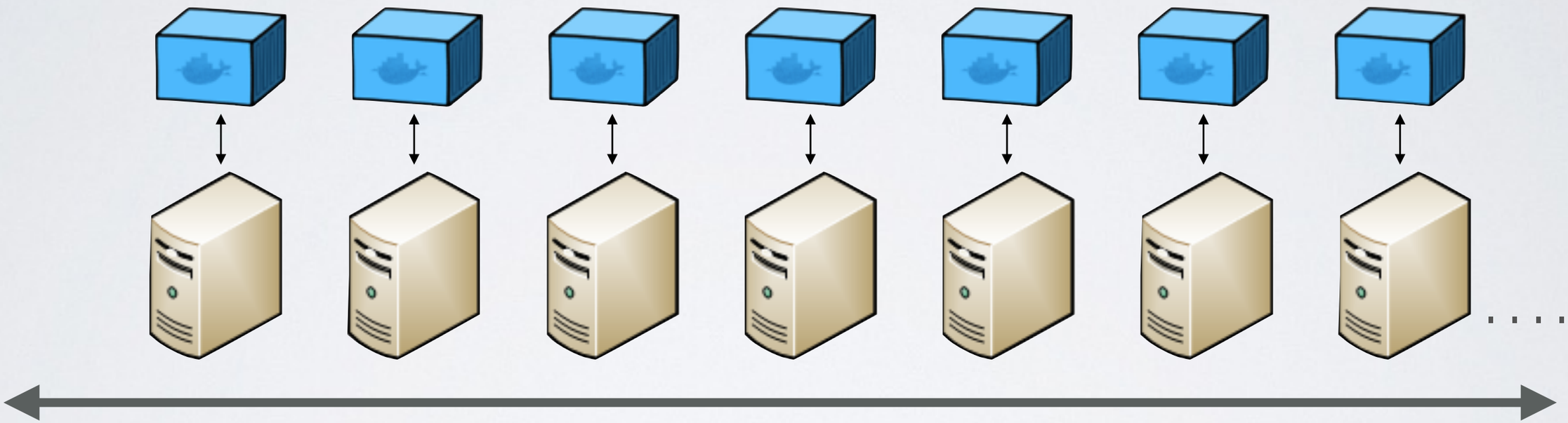
```
$ my_favourite_tool --foo --bar
```

```
$ docker run image/name my_favourite_tool --foo --bar
```

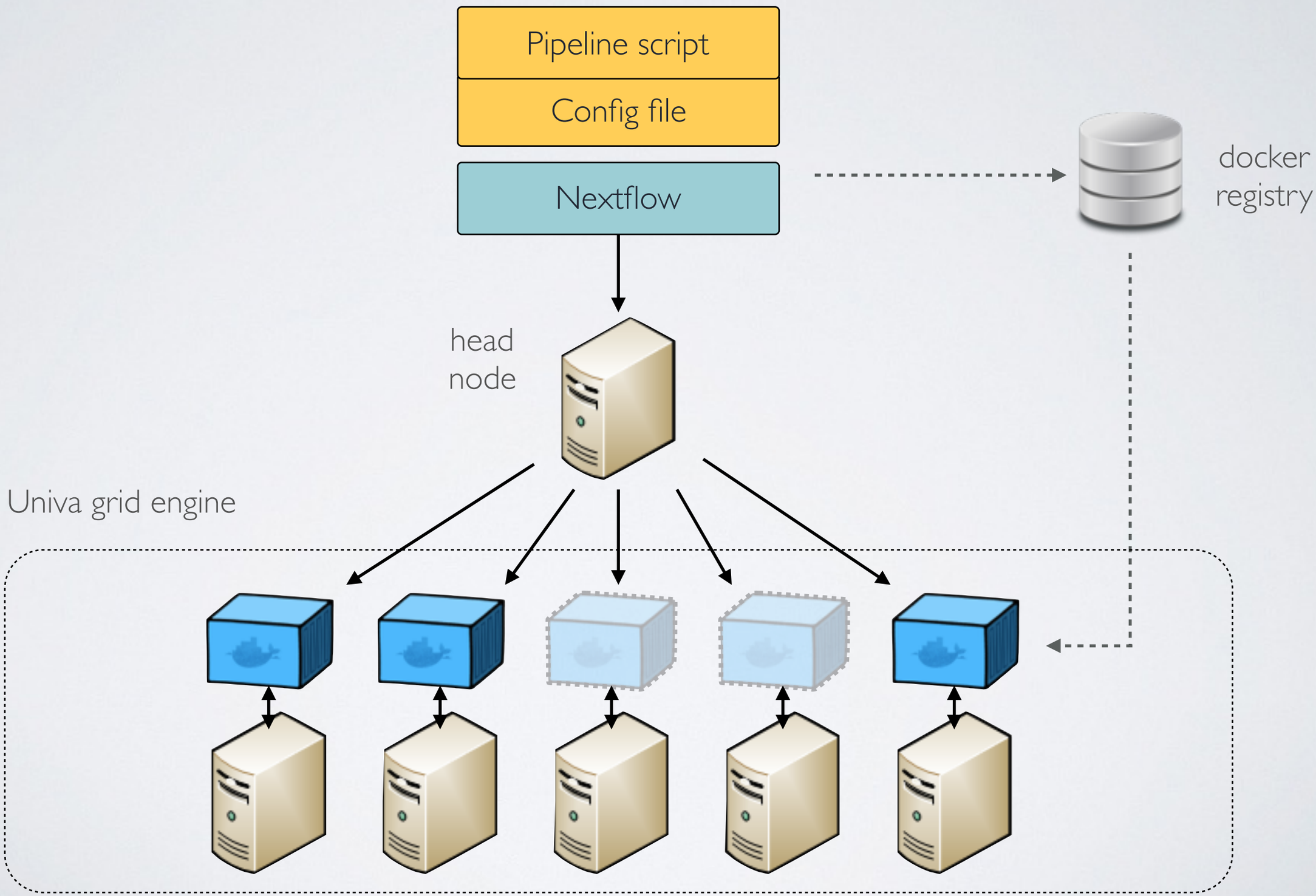
NAIVE DOCKERISATION



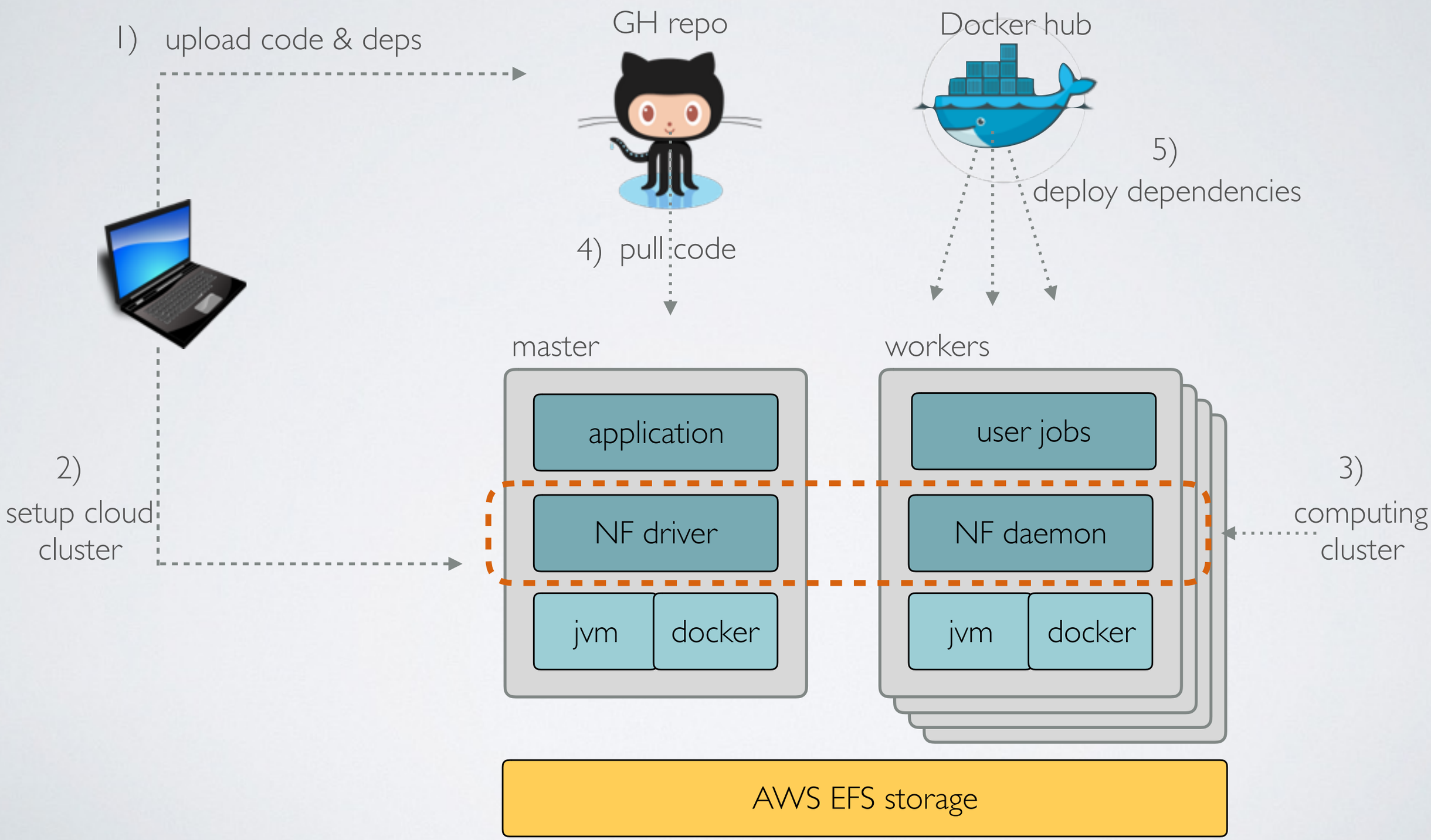
SCALING OUT



DOCKER AT CRG



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

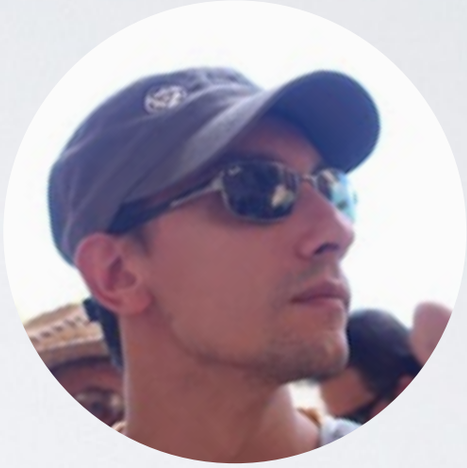
DOCKER HANDS-ON

HIGHLIGHTS

- Lightweight, command line tool
- Functional-reactive programming model
- Built-in support for multiple execution platforms (clusters, HPC, cloud)
- Integrates community standards such as Docker and Git/GitHub to enable reproducible deployments

CREDITS

Evan Floden, CRG



Emilio Palumbo, CRG

Maria Chatzou, CRG

Cedric Notredame, CRG

THANK YOU

LINKS

project home

http://nextflow.io

gitter

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

tutorial

https://goo.gl/9ffynC