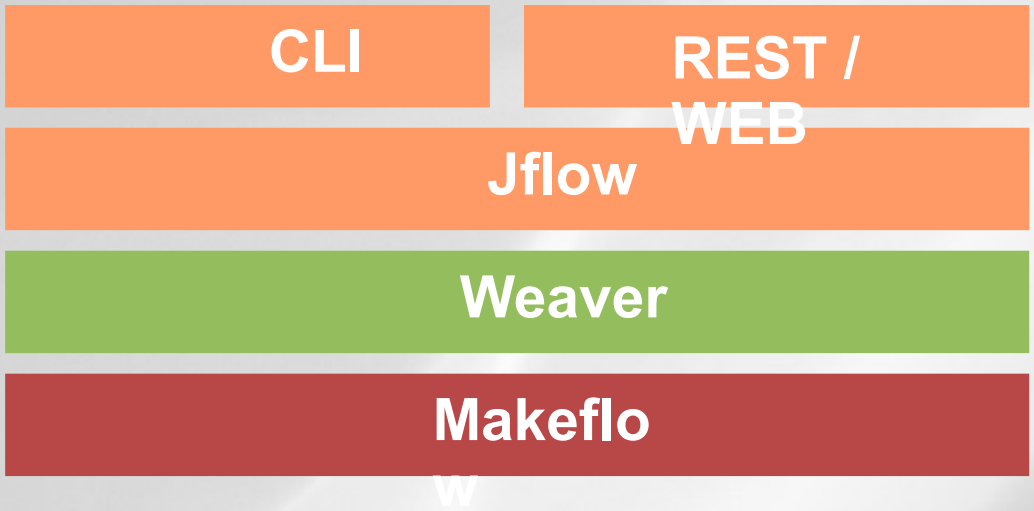
Plateforme Bioinformatique Midi-Pyrénées

Makeflow, weaver & co …

Introduction



User interfaces

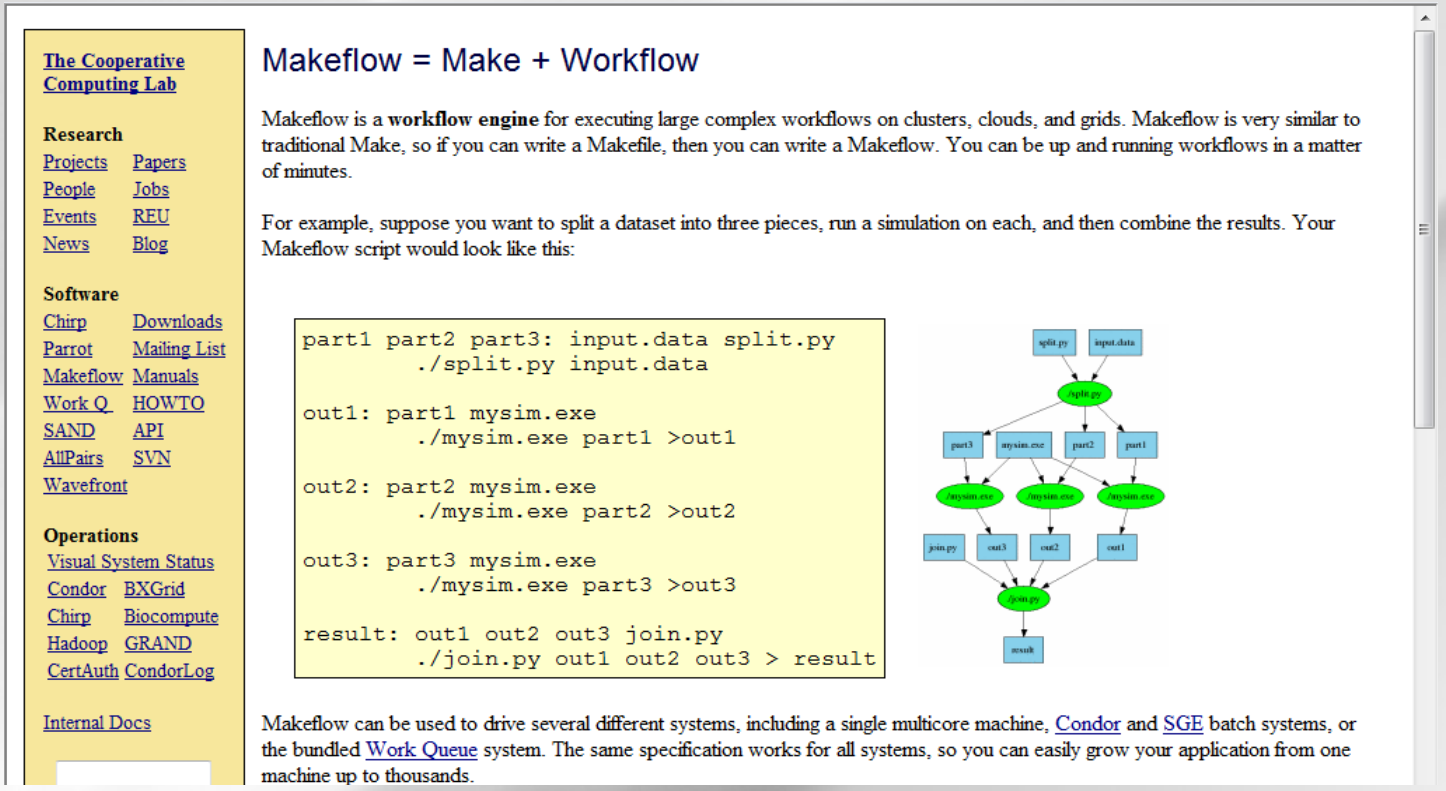
Workflow/Component API

Python API to MAKEFLOW

Workflow manager / Job submitter

Introduction to MAKEFLOW

- A Portable Abstraction for Data Intensive Computing on Clusters, Clouds, and Grids.
- University of Notre Dame.
- Parts of the cctools (gathers computing tools).



The Cooperative Computing Lab

Research
Projects Papers
People Jobs
Events REU
News Blog

Software
Chirp Downloads
Parrot Mailing List
Makeflow Manuals
Work Q HOWTO
SAND API
AllPairs SVN
Wavefront

Operations
Visual System Status
Condor BXGrid
Chirp Biocompute
Hadoop GRAND
CertAuth CondorLog

Internal Docs

Makeflow = Make + Workflow

Makeflow is a **workflow engine** for executing large complex workflows on clusters, clouds, and grids. Makeflow is very similar to traditional Make, so if you can write a Makefile, then you can write a Makeflow. You can be up and running workflows in a matter of minutes.

For example, suppose you want to split a dataset into three pieces, run a simulation on each, and then combine the results. Your Makeflow script would look like this:

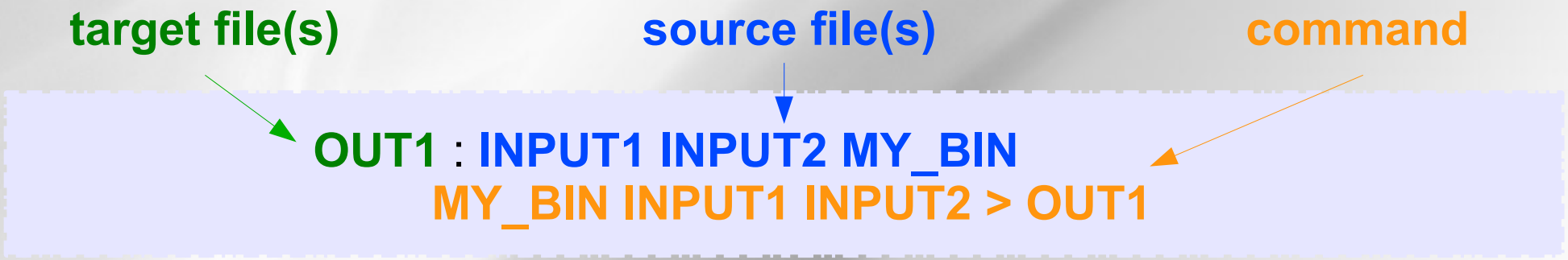
```
part1 part2 part3: input.data split.py
        ./split.py input.data

out1: part1 mysim.exe
        ./mysim.exe part1 >out1

out2: part2 mysim.exe
        ./mysim.exe part2 >out2

out3: part3 mysim.exe
        ./mysim.exe part3 >out3

result: out1 out2 out3 join.py
        ./join.py out1 out2 out3 > result
```


Makeflow can be used to drive several different systems, including a single multicore machine, Condor and SGE batch systems, or the bundled Work Queue system. The same specification works for all systems, so you can easily grow your application from one machine up to thousands.

Introduction to MAKEFLOW

- Support local, condor, sge, moab, cluster, wq, hadoop, mpi-queue.
- Easy to deploy, no dependencies.
- First application = Biocompute, an improved collaborative workspace for data intensive bio-science.
- Has a syntax similar to traditional UNIX Make.

- A Makeflow script consists of a set of rules. Each rule specifies :
  - a set of target files to create,
  - a set of source files needed to create them,
  - a command that generates the target files from the source files.



Introduction to MAKEFLOW

- No phony rules

```
out1 : input1 my_bin
    my_bin input1 > out1
```

correct rule

```
out1 :
    my_bin input1 > out1
```

incorrect rule

```
clean :
    rm -rf *.o
```

an other incorrect rule

MAKEFLOW a real example

```
# This is an example of Makeflow.
# 1. Download the picture.
a.jpg: /usr/bin/curl
	/usr/bin/curl -o a.jpg http://www.cse.nd.edu/~ccl/images/capitol.jpg
# 2. Curve the pictures at 90°, 180°, 270° and 360°
a.90.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 90 a.jpg a.90.jpg
a.180.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 180 a.jpg a.180.jpg
a.270.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 270 a.jpg a.270.jpg
a.360.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 360 a.jpg a.360.jpg
# 3. Combine curved pictures into a movie
a.montage.gif: a.jpg a.90.jpg a.180.jpg a.270.jpg a.360.jpg
	/usr/bin/convert -delay 10 -loop 0 a.jpg a.90.jpg a.180.jpg
a.270.jpg a.360.jpg a.270.jpg a.180.jpg a.90.jpg a.montage.gif
```

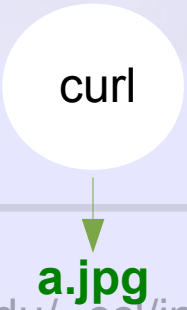
MAKEFLOW a real example

```
# This is an example of Makeflow.
# 1. Download the picture.
a.jpg: /usr/bin/curl
	/usr/bin/curl -o a.jpg http://www.cse.nd.edu/~ccl/images/capitol.jpg
# 2. Curve the pictures at 90°, 180°, 270° and 360°
a.90.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 90 a.jpg a.90.jpg
a.180.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 180 a.jpg a.180.jpg
a.270.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 270 a.jpg a.270.jpg
a.360.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 360 a.jpg a.360.jpg
# 3. Combine curved pictures into a movie
a.montage.gif: a.jpg a.90.jpg a.180.jpg a.270.jpg a.360.jpg
	/usr/bin/convert -delay 10 -loop 0 a.jpg a.90.jpg a.180.jpg
a.270.jpg a.360.jpg a.270.jpg a.180.jpg a.90.jpg a.montage.gif
```

geno toul bioinfo

MAKEFLOW a real example

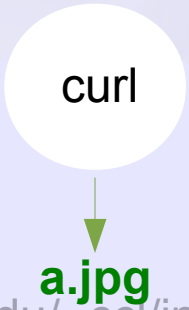
```
# This is an example of Makeflow.
# 1. Download the picture.
a.jpg: /usr/bin/curl
	/usr/bin/curl -o a.jpg http://www.cse.nd.edu/~ccl/images/capitol.jpg
# 2. Curve the pictures at 90°, 180°, 270° and 360°
a.90.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 90 a.jpg a.90.jpg
a.180.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 180 a.jpg a.180.jpg
a.270.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 270 a.jpg a.270.jpg
a.360.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 360 a.jpg a.360.jpg
# 3. Combine curved pictures into a movie
a.montage.gif: a.jpg a.90.jpg a.180.jpg a.270.jpg a.360.jpg
	/usr/bin/convert -delay 10 -loop 0 a.jpg a.90.jpg a.180.jpg
a.270.jpg a.360.jpg a.270.jpg a.180.jpg a.90.jpg a.montage.gif
```



geno toul bioinfo

MAKEFLOW a real example

```
# This is an example of Makeflow.
# 1. Download the picture.
a.jpg: /usr/bin/curl
	/usr/bin/curl -o a.jpg http://www.cse.nd.edu/~ccl/images/capitol.jpg
# 2. Curve the pictures at 90°, 180°, 270° and 360°
a.90.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 90 a.jpg a.90.jpg
a.180.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 180 a.jpg a.180.jpg
a.270.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 270 a.jpg a.270.jpg
a.360.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 360 a.jpg a.360.jpg
# 3. Combine curved pictures into a movie
a.montage.gif: a.jpg a.90.jpg a.180.jpg a.270.jpg a.360.jpg
	/usr/bin/convert -delay 10 -loop 0 a.jpg a.90.jpg a.180.jpg
a.270.jpg a.360.jpg a.270.jpg a.180.jpg a.90.jpg a.montage.gif
```



MAKEFLOW a real example

```
# This is an example of Makeflow.
# 1. Download the picture.
a.jpg: /usr/bin/curl
	/usr/bin/curl -o a.jpg http://www.cse.nd.edu/~ccl/images/capitol.jpg
# 2. Curve the pictures a.jpg 90°, 180°, 270° and 360°
a.90.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 90 a.jpg a.90.jpg
a.180.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 180 a.jpg a.180.jpg
a.270.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 270 a.jpg a.270.jpg
a.360.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 360 a.jpg a.360.jpg
# 3. Combine curved pictures into a movie
a.montage.gif: a.jpg a.90.jpg a.180.jpg a.270.jpg a.360.jpg
	/usr/bin/convert -delay 10 -loop 0 a.jpg a.90.jpg a.180.jpg
a.270.jpg a.360.jpg a.270.jpg a.180.jpg a.90.jpg a.montage.gif
```

geno toul bioinfo

MAKEFLOW a real example

```
# This is an example of Makeflow.
# 1. Download the picture.
a.jpg: /usr/bin/curl
	/usr/bin/curl -o a.jpg http://www.cse.nd.edu/~ccl/images/capitol.jpg
# 2. Curve the pictures a[illegible] 180°, 270° and 360°
a.90.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 90 a.jpg a.90.jpg
a.180.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 180 a.jpg a.180.jpg
a.270.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 270 a.jpg a.270.jpg
a.360.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 360 a.jpg a.360.jpg
# 3. Combine curved pictures into a movie
a.montage.gif: a.jpg a.90.jpg a.180.jpg a.270.jpg a.360.jpg
	/usr/bin/convert -delay 10 -loop 0 a.jpg a.90.jpg a.180.jpg
a.270.jpg a.360.jpg a.270.jpg a.180.jpg a.90.jpg a.montage.gif
```

curl

convert
90

MAKEFLOW a real example

```
# This is an example of Makeflow.
# 1. Download the picture.
a.jpg: /usr/bin/curl
	/usr/bin/curl -o a.jpg http://www.cse.nd.edu/~ccl/images/capitol.jpg
# 2. Curve the pictures a[illegible] 180°, 270° and 360°
a.90.jpg: a.jpg /usr/bin/c[illegible]
	/usr/bin/convert -swirl 90 a.jpg a.90.jpg
a.180.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 180 a.jpg a.180.jpg
a.270.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 270 a.jpg a.270.jpg
a.360.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 360 a.jpg a.360.jpg
# 3. Combine curved pictures into a movie
a.montage.gif: a.jpg a.90.jpg a.180.jpg a.270.jpg a.360.jpg
	/usr/bin/convert -delay 10 -loop 0 a.jpg a.90.jpg a.180.jpg
a.270.jpg a.360.jpg a.270.jpg a.180.jpg a.90.jpg a.montage.gif
```

curl

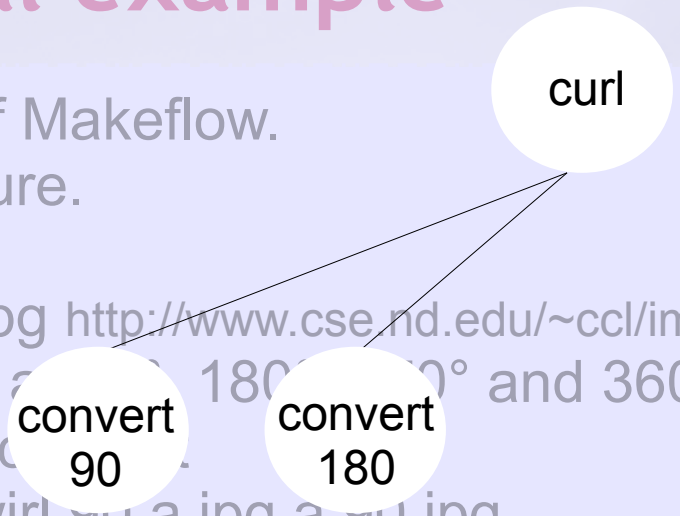
convert
90

MAKEFLOW a real example

```
# This is an example of Makeflow.
# 1. Download the picture.
a.jpg: /usr/bin/curl
	/usr/bin/curl -o a.jpg http://www.cse.nd.edu/~ccl/images/capitol.jpg
# 2. Curve the pictures at 90°, 180°, 270° and 360°
a.90.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 90 a.jpg a.90.jpg
a.180.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 180 a.jpg a.180.jpg
a.270.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 270 a.jpg a.270.jpg
a.360.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 360 a.jpg a.360.jpg
# 3. Combine curved pictures into a movie
a.montage.gif: a.jpg a.90.jpg a.180.jpg a.270.jpg a.360.jpg
	/usr/bin/convert -delay 10 -loop 0 a.jpg a.90.jpg a.180.jpg
a.270.jpg a.360.jpg a.270.jpg a.180.jpg a.90.jpg a.montage.gif
```

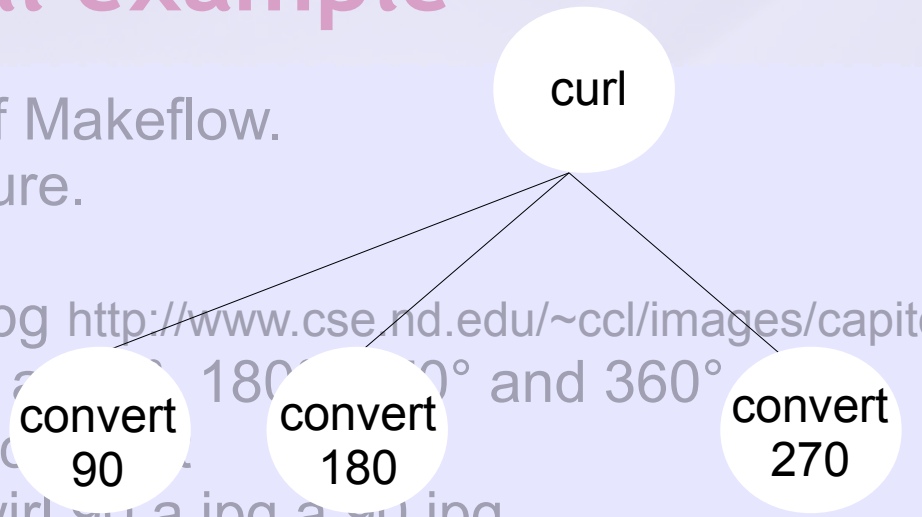
MAKEFLOW a real example

```
# This is an example of Makeflow.
# 1. Download the picture.
a.jpg: /usr/bin/curl
	/usr/bin/curl -o a.jpg http://www.cse.nd.edu/~ccl/images/capitol.jpg
# 2. Curve the pictures a.90.jpg, a.180.jpg, a.270.jpg and a.360.jpg
a.90.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 90 a.jpg a.90.jpg
a.180.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 180 a.jpg a.180.jpg
a.270.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 270 a.jpg a.270.jpg
a.360.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 360 a.jpg a.360.jpg
# 3. Combine curved pictures into a movie
a.montage.gif: a.jpg a.90.jpg a.180.jpg a.270.jpg a.360.jpg
	/usr/bin/convert -delay 10 -loop 0 a.jpg a.90.jpg a.180.jpg
a.270.jpg a.360.jpg a.270.jpg a.180.jpg a.90.jpg a.montage.gif
```



MAKEFLOW a real example

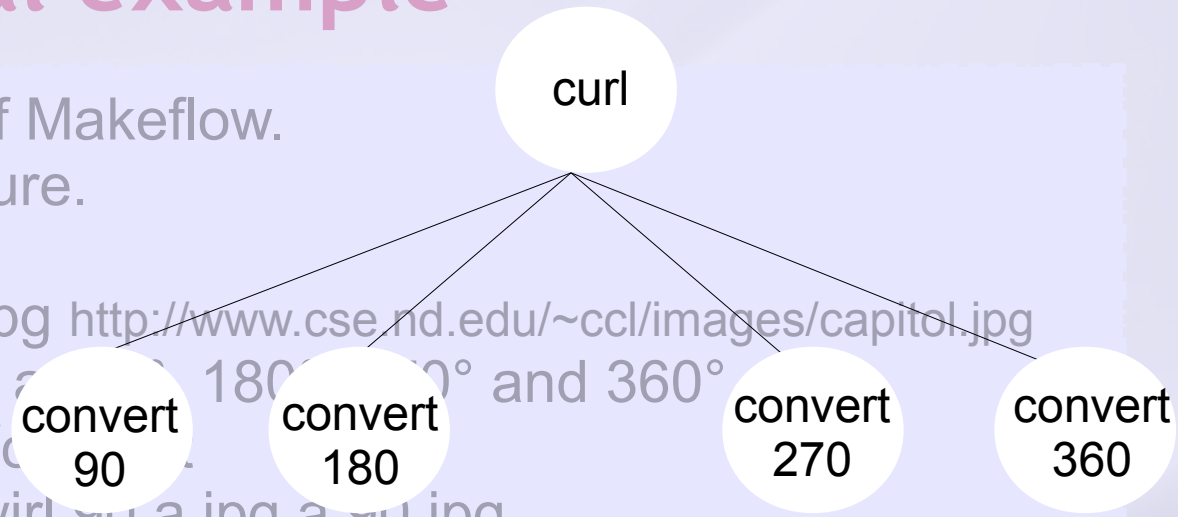
```
# This is an example of Makeflow.
# 1. Download the picture.
a.jpg: /usr/bin/curl
	/usr/bin/curl -o a.jpg http://www.cse.nd.edu/~ccl/images/capitol.jpg
# 2. Curve the pictures a 180° and 360°
a.90.jpg: a.jpg /usr/bin/c
	/usr/bin/convert -swirl 90 a.jpg a.90.jpg
a.180.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 180 a.jpg a.180.jpg
a.270.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 270 a.jpg a.270.jpg
a.360.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 360 a.jpg a.360.jpg
# 3. Combine curved pictures into a movie
a.montage.gif: a.jpg a.90.jpg a.180.jpg a.270.jpg a.360.jpg
	/usr/bin/convert -delay 10 -loop 0 a.jpg a.90.jpg a.180.jpg
a.270.jpg a.360.jpg a.270.jpg a.180.jpg a.90.jpg a.montage.gif
```



geno toul bioinfo

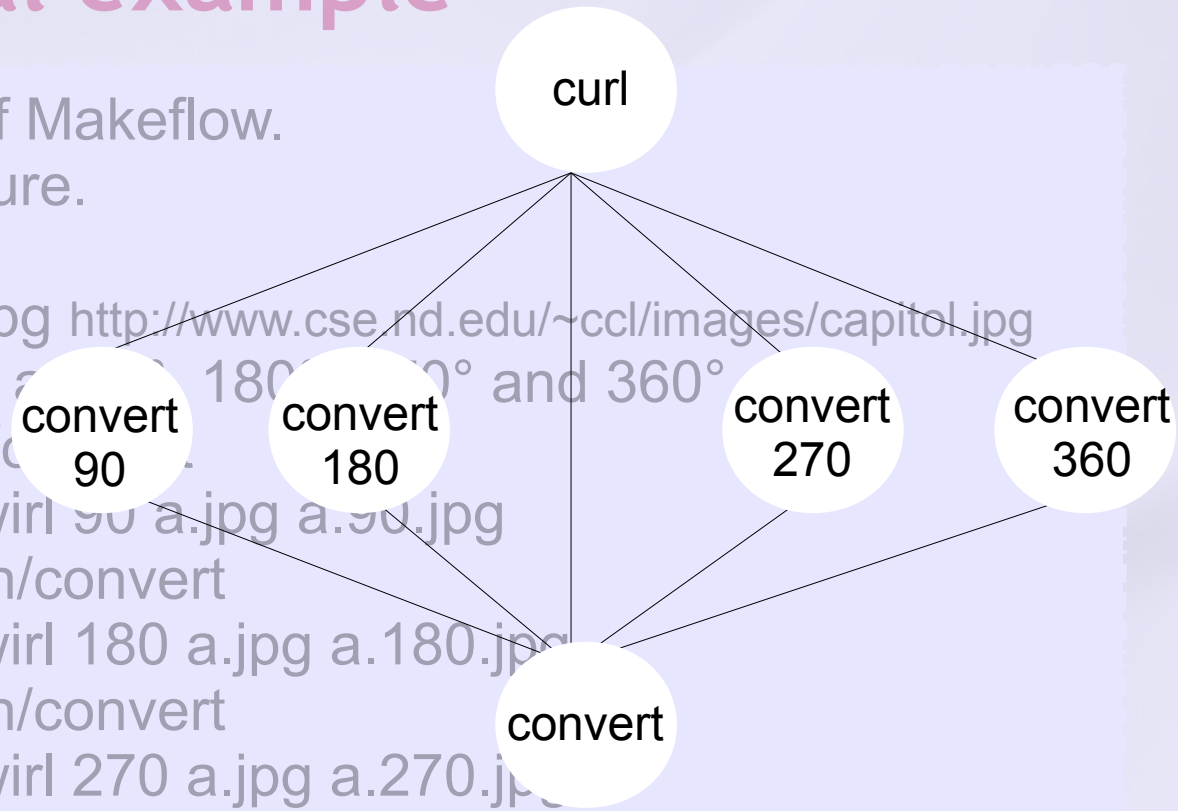
MAKEFLOW a real example

```
# This is an example of Makeflow.
# 1. Download the picture.
a.jpg: /usr/bin/curl
	/usr/bin/curl -o a.jpg http://www.cse.nd.edu/~ccl/images/capitol.jpg
# 2. Curve the pictures a[illegible] 180[illegible]° and 360°
a.90.jpg: a.jpg /usr/bin/c[illegible]
	/usr/bin/convert -swirl 90 a.jpg a.90.jpg
a.180.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 180 a.jpg a.180.jpg
a.270.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 270 a.jpg a.270.jpg
a.360.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 360 a.jpg a.360.jpg
# 3. Combine curved pictures into a movie
a.montage.gif: a.jpg a.90.jpg a.180.jpg a.270.jpg a.360.jpg
	/usr/bin/convert -delay 10 -loop 0 a.jpg a.90.jpg a.180.jpg
a.270.jpg a.360.jpg a.270.jpg a.180.jpg a.90.jpg a.montage.gif
```



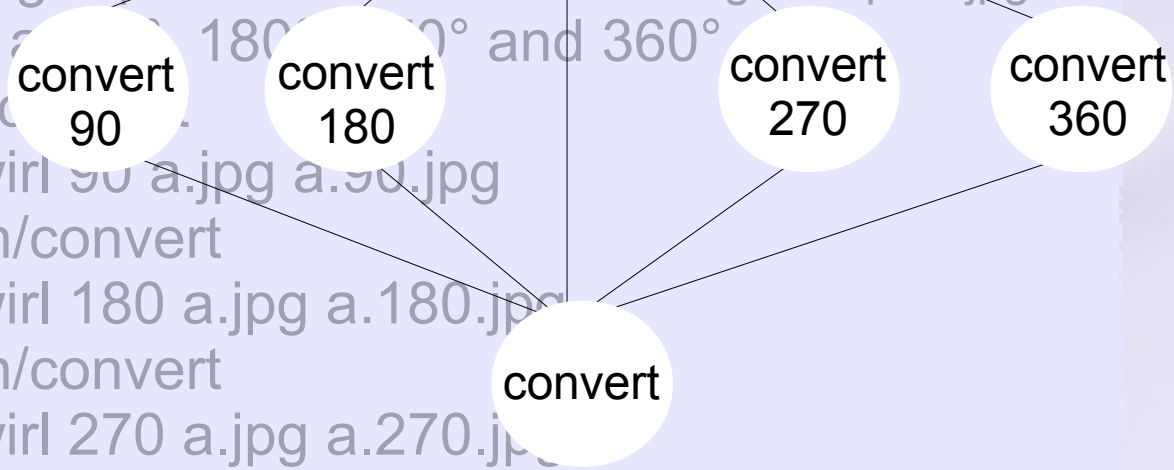
MAKEFLOW a real example

```
# This is an example of Makeflow.
# 1. Download the picture.
a.jpg: /usr/bin/curl
	/usr/bin/curl -o a.jpg http://www.cse.nd.edu/~ccl/images/capitol.jpg
# 2. Curve the pictures a 180° and 360°
a.90.jpg: a.jpg /usr/bin/c
	/usr/bin/convert -swirl 90 a.jpg a.90.jpg
a.180.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 180 a.jpg a.180.jpg
a.270.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 270 a.jpg a.270.jpg
a.360.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 360 a.jpg a.360.jpg
# 3. Combine curved pictures into a movie
a.montage.gif: a.jpg a.90.jpg a.180.jpg a.270.jpg a.360.jpg
	/usr/bin/convert -delay 10 -loop 0 a.jpg a.90.jpg a.180.jpg
a.270.jpg a.360.jpg a.270.jpg a.180.jpg a.90.jpg a.montage.gif
```



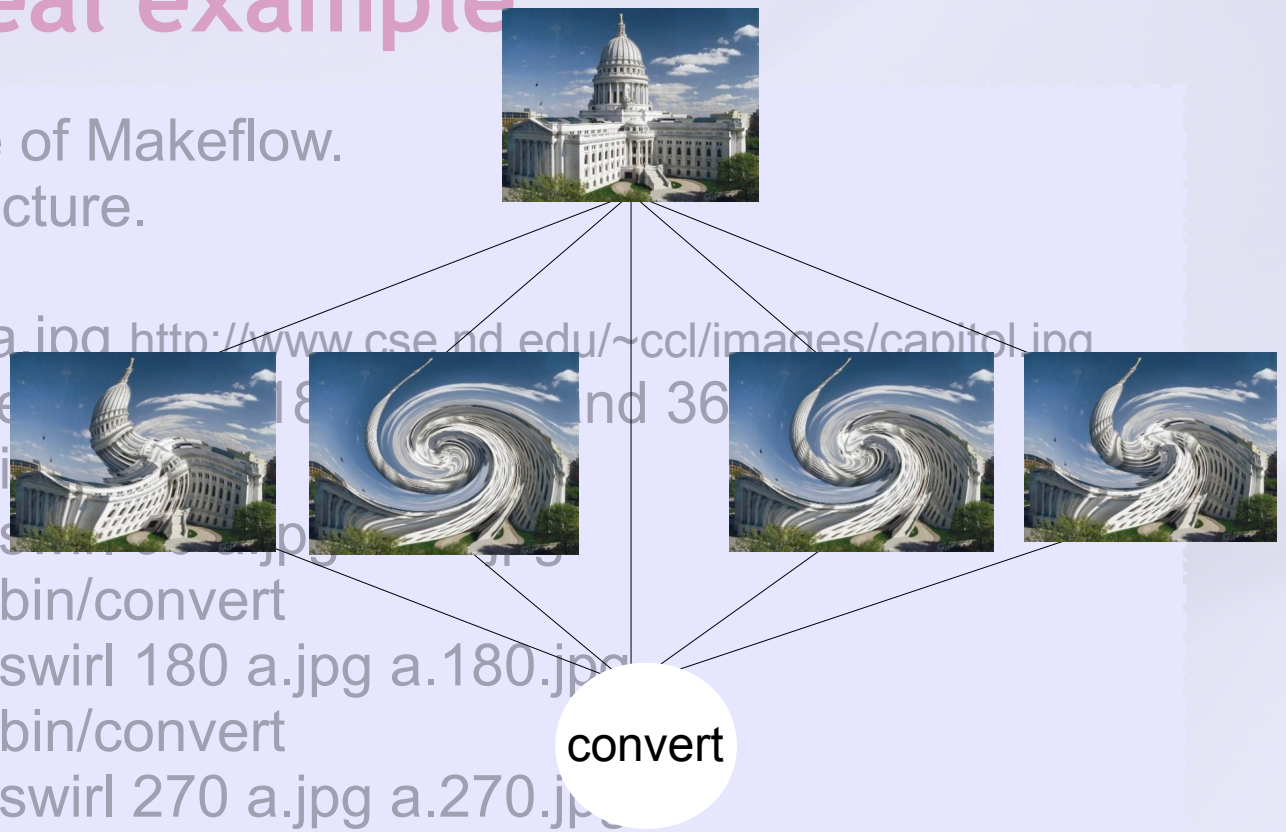
MAKEFLOW a real example

```
# This is an example of Makeflow.
# 1. Download the picture.
a.jpg: /usr/bin/curl
	/usr/bin/curl -o a.jpg http://www.cse.nd.edu/~ccl/images/capitol.jpg
# 2. Curve the pictures a.90.jpg, a.180.jpg, a.270.jpg and a.360.jpg
a.90.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 90 a.jpg a.90.jpg
a.180.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 180 a.jpg a.180.jpg
a.270.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 270 a.jpg a.270.jpg
a.360.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 360 a.jpg a.360.jpg
# 3. Combine curved pictures into a movie
a.montage.gif: a.jpg a.90.jpg a.180.jpg a.270.jpg a.360.jpg
	/usr/bin/convert -delay 10 -loop 0 a.jpg a.90.jpg a.180.jpg
a.270.jpg a.360.jpg a.270.jpg a.180.jpg a.90.jpg a.montage.gif
```



MAKEFLOW a real example

```
# This is an example of Makeflow.
# 1. Download the picture.
a.jpg: /usr/bin/curl
	/usr/bin/curl -o a.jpg http://www.cse.nd.edu/~ccl/images/capitol.jpg
# 2. Curve the picture 18 and 36
a.90.jpg: a.jpg /usr/bi
	/usr/bin/convert -
a.180.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 180 a.jpg a.180.jpg
a.270.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 270 a.jpg a.270.jpg
a.360.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 360 a.jpg a.360.jpg
# 3. Combine curved pictures into a movie
a.montage.gif: a.jpg a.90.jpg a.180.jpg a.270.jpg a.360.jpg
	/usr/bin/convert -delay 10 -loop 0 a.jpg a.90.jpg a.180.jpg
a.270.jpg a.360.jpg a.270.jpg a.180.jpg a.90.jpg a.montage.gif
```



geno toul bioinfo

MAKEFLOW a real example

```
# This is an example of Makeflow.
# 1. Download the picture.
a.jpg: /usr/bin/curl
	/usr/bin/curl -o a.jpg http://www.cse.nd.edu/~ccl/images/capitol.jpg
# 2. Curve the picture 180 and 360
a.90.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 90 a.jpg a.90.jpg
a.180.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 180 a.jpg a.180.jpg
a.270.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 270 a.jpg a.270.jpg
a.360.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 360 a.jpg a.360.jpg
# 3. Combine curved pictures into a movie
a.montage.gif: a.jpg a.90.jpg a.180.jpg a.270.jpg a.360.jpg
	/usr/bin/convert -delay 10 -loop 0 a.jpg a.90.jpg a.180.jpg
a.270.jpg a.360.jpg a.270.jpg a.180.jpg a.90.jpg a.montage.gif
```

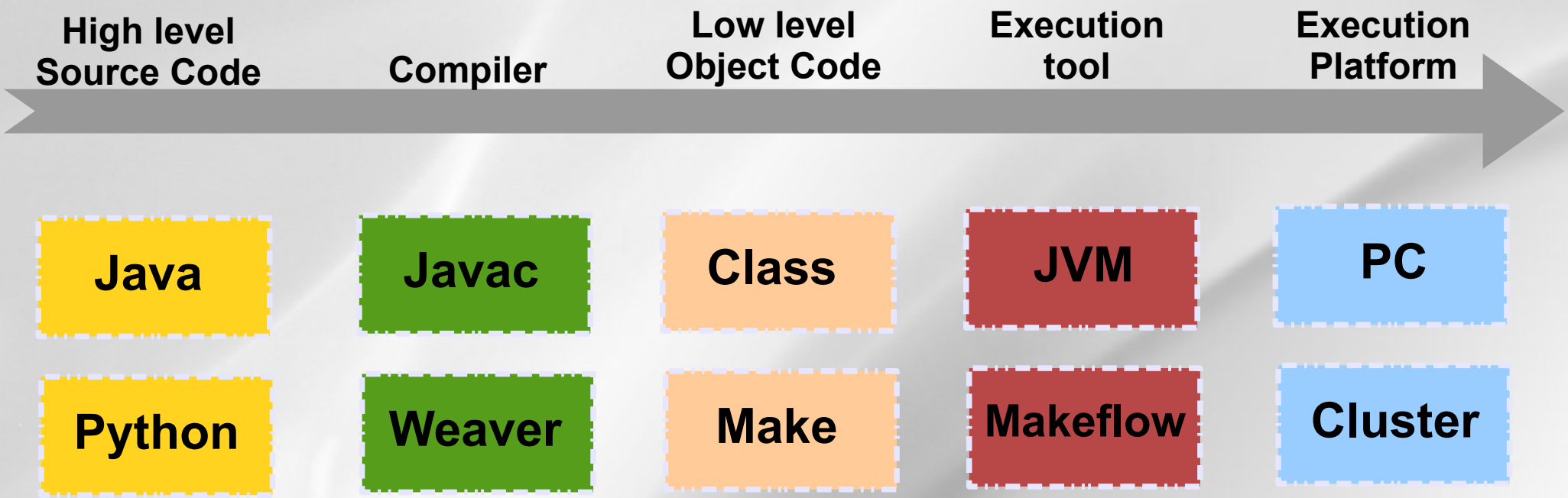
MAKEFLOW let's give it a try !

Exercice #1 :

```
# This is an example of Makeflow.
# 1. Download the picture.
a.jpg: /usr/bin/curl
	/usr/bin/curl -o a.jpg http://www.cse.nd.edu/~ccl/images/capitol.jpg
# 2. Curve the pictures at 90°, 180°, 270° and 360°
a.90.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 90 a.jpg a.90.jpg
a.180.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 180 a.jpg a.180.jpg
a.270.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 270 a.jpg a.270.jpg
a.360.jpg: a.jpg /usr/bin/convert
	/usr/bin/convert -swirl 360 a.jpg a.360.jpg
# 3. Combine curved pictures into a movie
a.montage.gif: a.jpg a.90.jpg a.180.jpg a.270.jpg a.360.jpg
	/usr/bin/convert -delay 10 -loop 0 a.jpg a.90.jpg a.180.jpg
a.270.jpg a.360.jpg a.270.jpg a.180.jpg a.90.jpg a.montage.gif
```

Introduction to WEAVER

- A workflow compiler to build distributed workflows.
- API written in Python



Introduction to WEAVER

- **Dataset :** Collections of data objects that represent physical files
- **Function :** Specification of executables used to process data.
- **Abstraction :** Patterns of execution that define how functions are applied to datasets

Function(**Dataset**)

```
out1 : input1 my_bin
   my_bin input1 > out1
```

Abstraction(**Function**, **Dataset**)

```
out1 : input1 my_bin
   my_bin input1 > out1
out2 : input2 my_bin
   my_bin input2 > out2
...
```

WEAVER a simple example

- **Dataset :** Collections of data objects that represent physical files
- **Function :** Specification of executables used to process data.
- **Abstraction :** Patterns of execution that define how functions are applied to datasets

```
input = '/path/to/file1.fastq'
output = '/path/to/file1.sai'
func = ShellFunction('bwa aln /path/to/ref.fa $1 > $2', cmd_format='{EXE} {IN} {OUT}')
func(inputs=input, outputs=output)
```

WEAVER a simple example

- **Dataset :** Collections of data objects that represent physical files
- **Function :** Specification of executables used to process data.
- **Abstraction :** Patterns of execution that define how functions are applied to datasets

```
input = '/path/to/file1.fastq'
output = '/path/to/file1.sai'
func = ShellFunction('bwa aln /path/to/ref.fa $1 > $2', cmd_format='{EXE} {IN} {OUT}')
func(inputs=input, outputs=output)
```

```
/path/to/file1.sai: /path/to/file1.fastq
   bwa aln /path/to/ref.fa /path/to/file1.fastq > /path/to/file1.sai
```

WEAVER an other example

- **Dataset :** Collections of data objects that represent physical files
- **Function :** Specification of executables used to process data.
- **Abstraction :** Patterns of execution that define how functions are applied to datasets

```
input = ArrayList('/path/to/file1.fastq', '/path/to/file2.fastq', '/path/to/file2.fastq')
func = ShellFunction('bwa aln /path/to/ref.fa $1 > $2', cmd_format='{EXE} {IN} {OUT}')
Map(func, inputs=input, outputs='{basename}.sai')
```

WEAVER an other example

- **Dataset :** Collections of data objects that represent physical files
- **Function :** Specification of executables used to process data.
- **Abstraction :** Patterns of execution that define how functions are applied to datasets

```
input = ArrayList('/path/to/file1.fastq', '/path/to/file2.fastq', '/path/to/file2.fastq')
func = ShellFunction('bwa aln /path/to/ref.fa $1 > $2', cmd_format='{EXE} {IN} {OUT}')
Map(func, inputs=input, outputs='{basename}.sai')
```

```
/path/to/file1.sai : /path/to/file1.fastq
   bwa aln /path/to/ref.fa /path/to/file1.fastq > /path/to/file1.sai
```

WEAVER an other example

- **Dataset :** Collections of data objects that represent physical files
- **Function :** Specification of executables used to process data.
- **Abstraction :** Patterns of execution that define how functions are applied to datasets

```
input = ArrayList('/path/to/file1.fastq', '/path/to/file2.fastq', '/path/to/file2.fastq')
func = ShellFunction('bwa aln /path/to/ref.fa $1 > $2', cmd_format='{EXE} {IN} {OUT}')
Map(func, inputs=input, outputs='{basename}.sai')
```

```
/path/to/file1.sai : /path/to/file1.fastq
   bwa aln /path/to/ref.fa /path/to/file1.fastq > /path/to/file1.sai
/path/to/file2.sai : /path/to/file2.fastq
   bwa aln /path/to/ref.fa /path/to/file2.fastq > /path/to/file2.sai
```

WEAVER an other example

- **Dataset :** Collections of data objects that represent physical files
- **Function :** Specification of executables used to process data.
- **Abstraction :** Patterns of execution that define how functions are applied to datasets

```
input = ArrayList('/path/to/file1.fastq', '/path/to/file2.fastq', '/path/to/file2.fastq')
func = ShellFunction('bwa aln /path/to/ref.fa $1 > $2', cmd_format='{EXE} {IN} {OUT}')
Map(func, inputs=input, outputs='{basename}.sai')
```

```
/path/to/file1.sai : /path/to/file1.fastq
   bwa aln /path/to/ref.fa /path/to/file1.fastq > /path/to/file1.sai
/path/to/file2.sai : /path/to/file2.fastq
   bwa aln /path/to/ref.fa /path/to/file2.fastq > /path/to/file2.sai
/path/to/file3.sai : /path/to/file3.fastq
   bwa aln /path/to/ref.fa /path/to/file3.fastq > /path/to/file3.sai
```

WEAVER an other example

- **Dataset :** Collections of data objects that represent physical files
- **Function :** Specification of executables used to process data.
- **Abstraction :** Patterns of execution that define how functions are applied to datasets

```
input = ArrayList('/path/to/file1.fastq', '/path/to/file2.fastq', '/path/to/file2.fastq')
func = ShellFunction('bwa aln /path/to/ref.fa $1 > $2', cmd_format='{EXE} {IN} {OUT}')
Map(func, inputs=input, outputs='{basename}.sai')
```

```
/path/to/file1.sai : /path/to/file1.fastq
   bwa aln /path/to/ref.fa /path/to/file1.fastq > /path/to/file1.sai
/path/to/file2.sai : /path/to/file2.fastq
   bwa aln /path/to/ref.fa /path/to/file2.fastq > /path/to/file2.sai
/path/to/file3.sai : /path/to/file3.fastq
   bwa aln /path/to/ref.fa /path/to/file3.fastq > /path/to/file3.sai
```

WEAVER concepts

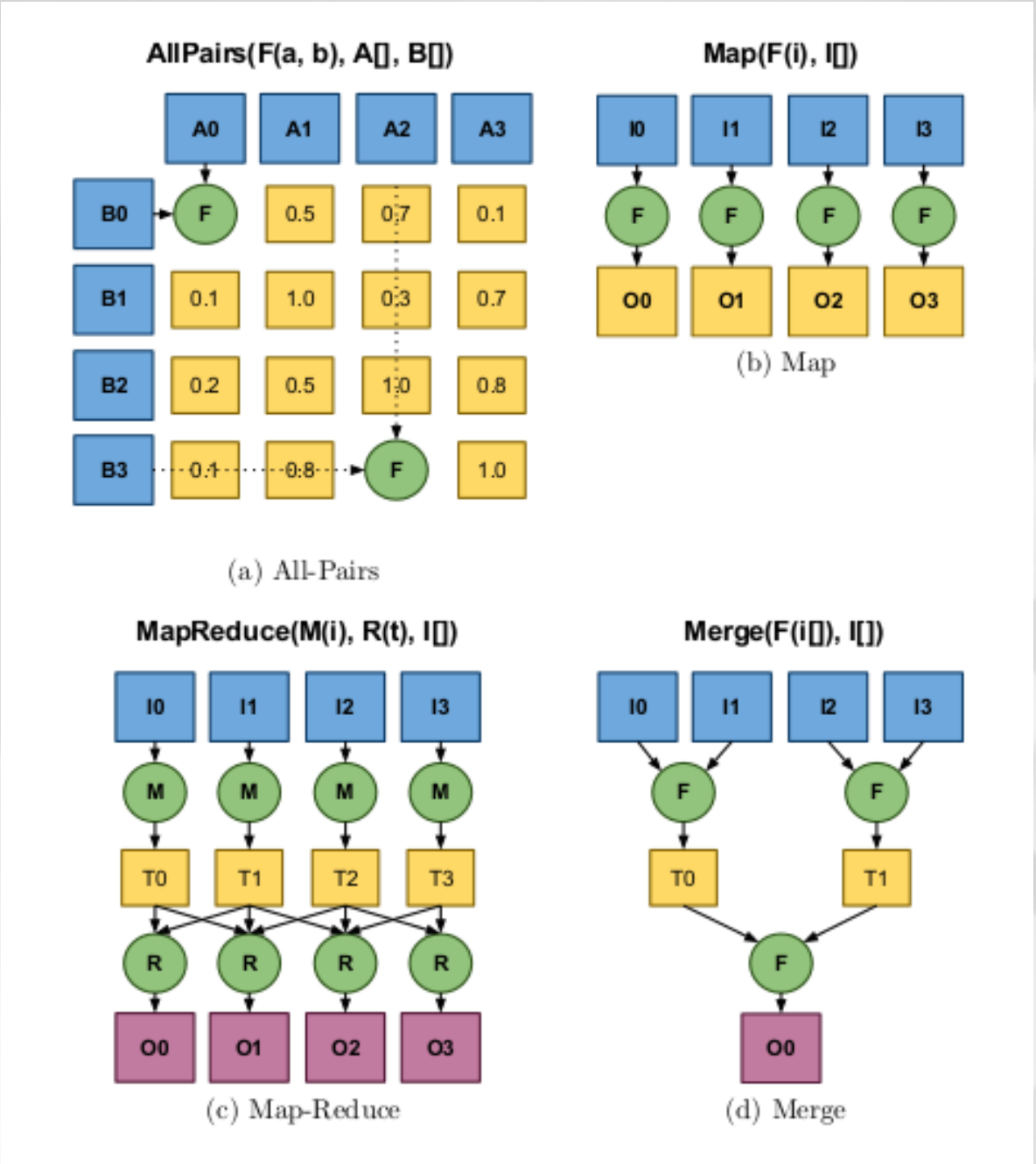
- **Datasets : Collections of data objects that represent physical files.**
  - Glob : Collection of files based on path expression.
  - FileList : Collection of files stored in a text file.
  - SQLDataset : Collection of data from a SQL database.
  - Query : ORM selection and filtering function for Weaver Datasets.
  - *ArrayList : Collection of files stored in a python array object.*

- **Functions : Specifications of executables used to process data.**
  - Function : Base Function object constructor.
  - ParseFunction : Convenience wrapper that constructs Function and sets command format.
  - ShellFunction : Constructs Function out of shell script specified as string.
  - PythonFunction : Constructs Function from inline Python code.
  - Pipeline : Combines multiple Functions into a single meta-Function.

WEAVER concepts

- **Abstractions : Patterns of execution that define how functions are applied to datasets.**
  - AllPairs : Apply function to all pair-wise combinations of inputs a and inputs b.
  - Map : For each input in inputs, apply function.
  - MapReduce : For each input in inputs apply mapper, sort intermediate outputs, and then apply reducer.
  - Merge : Use function combine inputs by using a parallel reduction.
  - *MultiMap : For each input(s) in inputs matrice and for each output(s) in outputs matrice, apply function.*
- **Output patterns :**
  - {fullpath}, {FULL} : full file path
  - {basename}, {BASE} : base file name
  - {fullpath_woext}, {FULLWE} : full file path without extension
  - basename_woext, BASEWE : base file name without extension

WEAVER concepts



(a) All-Pairs

(b) Map

(c) Map-Reduce

(d) Merge

WEAVER files

Output /

Makeflow — the make to execute by makeflow

Makeflow.makeflowlog — the makeflow log (written by makeflow)

WEAVER files

Output /

- Makeflow — the make to execute by makeflow
- Makeflow.makeflowlog — the makeflow log (written by makeflow)
- _Stash / — directories to store executables and files list
  - 0 /
    - 0 /
      - 0 /
        - 0000000
        - 0000001
        - 0000002
        - 00000... — executables and files list

WEAVER files

```
input = ArrayList('/path/to/file1.fastq', '/path/to/file2.fastq', '/path/to/file3.fastq')
func = ShellFunction('bwa aln /path/to/ref.fa $1 > $2', cmd_format='{EXE} {IN} {OUT}')
Map(func, inputs=input, outputs='{basename}.sai')
```


Stash /

```
/path/to/file1.sai : /path/to/file1.fastq
  bwa aln /path/to/ref.fa /path/to/file1.fastq > /path/to/file1.sai
/path/to/file2.sai : /path/to/file2.fastq
  bwa aln /path/to/ref.fa /path/to/file2.fastq > /path/to/file2.sai
/path/to/file3.sai : /path/to/file3.fastq
  bwa aln /path/to/ref.fa /path/to/file3.fastq > /path/to/file3.sai
```

00000...

executables and files list

WEAVER files

```
input = ArrayList('/path/to/file1.fastq', '/path/to/file2.fastq', '/path/to/file3.fastq')
func = ShellFunction('bwa aln /path/to/ref.fa $1 > $2', cmd_format='{EXE} {IN} {OUT}')
Map(func, inputs=input, outputs='{basename}.sai')
```

Stash /

```
/path/to/file1.sai : /path/to/file1.fastq
  /path/to/_Stash/0/0/0/000000 /path/to/file1.fastq /path/to/file1.sai
/path/to/file2.sai : /path/to/file2.fastq
  /path/to/_Stash/0/0/0/000000 /path/to/file2.fastq /path/to/file2.sai
/path/to/file3.sai : /path/to/file3.fastq
  /path/to/_Stash/0/0/0/000000 /path/to/file3.fastq /path/to/file3.sai
```

00000...

executables and files list

WEAVER files

```
input = ArrayList('/path/to/file1.fastq', '/path/to/file2.fastq', '/path/to/file3.fastq')
func = ShellFunction('bwa aln /path/to/ref.fa $1 > $2', cmd_format='{EXE} {IN} {OUT}')
Map(func, inputs=input, outputs='{basename}.sai')
```

```
/path/to/file1.sai : /path/to/file1.fastq
  /path/to/_Stash/0/0/0/000000 /path/to/file1.fastq /path/to/file1.sai
/path/to/file2.sai : /path/to/file2.fastq
  /path/to/_Stash/0/0/0/000000 /path/to/file2.fastq /path/to/file2.sai
/path/to/file3.sai : /path/to/file3.fastq
  /path/to/_Stash/0/0/0/000000 /path/to/file3.fastq /path/to/file3.sai
```

executables and files list

```
#!/bin/sh
bwa aln /path/to/ref.fa $1 > $2
```

WEAVER files

Output /

- Makeflow — the make to execute by makeflow
- Makeflow.makeflowlog — the makeflow log (written by makeflow)
- _Stash / — directories to store executables and files list
  - 0 /
    - 0 /
      - 0 /
        - 0000000
        - 0000001
        - 0000002
        - 00000... — executables and files list

WEAVER files

```
Output /
        Makeflow                          } the make to execute by makeflow
        Makeflow.makeflowlog              } the makeflow log (written by makeflow)

        _Stash /                          }
                0 /                       } directories to store executables and
                    0 /                   } files list
                        0 /               }
                            0000000       }
                            0000001       } executables and files list
                            0000002       }
                            00000...      }

        outputs                           } all outputs if no path provided
        ...                               }
```

WEAVER files

```
input = ArrayList('/path/to/file1.fastq', '/path/to/file2.fastq', '/path/to/file2.fastq')
func = ShellFunction('bwa aln /path/to/ref.fa $1 > $2', cmd_format='{EXE} {IN} {OUT}')
Map(func, inputs=input, outputs='{basename}.sai')
```

```
func = ShellFunction('bwa aln /path/to/ref.fa $1 > $2', cmd_format='{EXE} {IN} {OUT}')
func(inputs='/path/to/file1.fastq', outputs='file1.sai')
```

0 /

0 /

directories to store executables and

If no fullpath provided, the default path is the weaver output directory

0000002

00000...

outputs

...

all outputs if no path provided

WEAVER limits

- No component and workflow point of view : writing a script does not allow to reuse what's inside

- Command lines outputs are stored in the same directory, or has to be managed by the developer

- Dynamic command lines generating X files from 1 file requires a separated execution

- If a command line outputs a file but does not generate it (*ex: formatdb*), it requires a separated execution

WEAVER let's give it a try !

- **Install weaver**
  Redo the previous example using weaver

- **Test the install**

WEAVER let's give it a try !

- **Install and test the install**
  Follow the online documentation https://bitbucket.org/pbui/weaver/src

- **Exercice #1 :**
  Redo the previous example using weaver

- **Exercice #2 :**
  Produce a script able to :
  - index a fasta using bwa index
  - align paired reads using bwa aln
  - create a sam file using bwa sampe
  - convert the sam file in bam

  Where is the trap ?

  You will need :

  - CurrentScript().arguments: catch args from a weaver scripts
  - from weaver.data import parse_output_list : to build an output list
  - from weaver.data import parse_input_list : to build an input list

WEAVER let's give it a try !

- **Correction #1 :**

```
curl = ShellFunction("/usr/bin/curl -o $1 'http://www.cse.nd.edu/~ccl/images/capitol.jpg'",
cmd_format='{EXE} {OUT}')
curl(outputs="a.jpg")
degrees = ["90", "180", "270", "360"]
output_files = []
for degree in degrees:
      convert = ShellFunction("/usr/bin/convert -swirl " + degree + " $1 $2",
cmd_format='{EXE} {IN} {OUT}')
      convert(inputs="a.jpg", outputs="a."+degree+".jpg")
montage = ShellFunction("/usr/bin/convert -delay 10 -loop 0 $1 $2 $3 $4 $5 $6 $7 $8
$9", cmd_format='{EXE} {IN} {OUT}')
montage(inputs=["a.jpg", "a.90.jpg", "a.180.jpg", "a.270.jpg", "a.360.jpg", "a.270.jpg",
"a.180.jpg", "a.90.jpg"], outputs="a.montage.gif")
```

```
[jmariett@genotoul weaver]$ weaver -x exercice1.py
```

WEAVER let's give it a try !

- **Correction #2a :**

```
bwaindex = ShellFunction("bwa index -a bwtsw -p $1 $2", cmd_format='{EXE} {OUT} {IN}')
bwaindex(inputs=CurrentScript().arguments[0], outputs=CurrentScript().arguments[0])
```

```
[jmariett@genotoul weaver]$ weaver -x exercice2a.py /path/to/file.fasta
```

WEAVER let's give it a try !

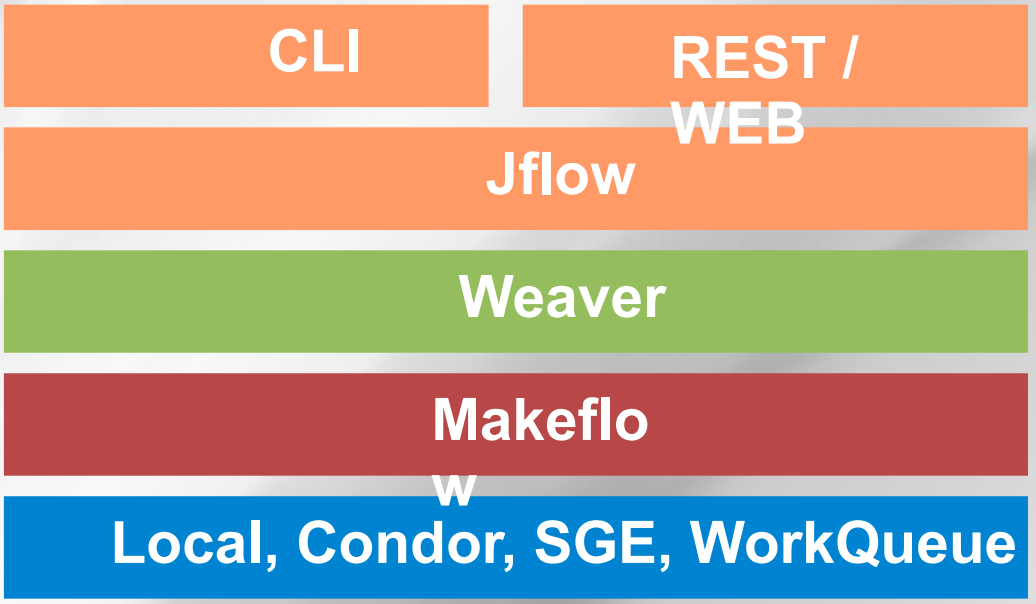
- **Correction #2b :**

```
from weaver.data import parse_output_list, parse_input_list
# Define inputs and outputs
bwainputs = parse_input_list([CurrentScript().arguments[1],
CurrentScript().arguments[2]])
bwaoutputs = parse_output_list("{basename_woext}.sai", bwainputs)
# Align
bwa = ShellFunction("bwa aln " + CurrentScript().arguments[0] + " $1 > $2",
cmd_format='{EXE} {IN} {OUT}')
Map(bwa, inputs=bwainputs, outputs=bwaoutputs)
bwasampe = ShellFunction("bwa sampe " + CurrentScript().arguments[0] + " $1 $2 $3
$4 > $5", cmd_format='{EXE} {IN} {OUT}')
bwasampe(inputs=bwaoutputs+bwainputs, outputs="myalignment.sam")
samtoolsview = ShellFunction("samtools view -bS $1 > $2", cmd_format='{EXE} {IN}
{OUT}')
samtoolsview(inputs="myalignment.sam", outputs="myalignment.bam")
```

```
[jmariett@genotoul weaver]$ weaver -x exercice2b.py /path/to/file.fasta
/path/to/read1.fq /path/to/read2.fq
```

Introduction to ?JFLOW?

- Objectives :
  - having a set of workflows available from CLI and WEB interfaces (mashup)
  - having a component and workflow point of view
  - managing outputs per component and per workflow
  - hiding some makeflow/weaver complexity for the developer
  - having low dependencies and having a easy to deploy system



Installing JFLOW

- Download sources :

```
svn checkout svn://scm.mulcyber.toulouse.inra.fr/svnroot/jflow/trunk
```

- Edit the configuration file:

```
[grid]
makeflow = /usr/bin/makeflow
# batch system type: local, condor, sge, moab, cluster, wq, hadoop, mpi-queue
batch_system_type = local
# add these options to all batch submit files
batch_options =

[storage]
# where should be written the log file
log_file = /home/jmariett/scratch/work/jflow.log
# Where should the pipelines write results, should be accessible
# by all cluster nodes
work_directory = /home/jmariett/scratch/work
# Where should the pipelines write temporary files, should be
# accessible by all cluster nodes
tmp_directory = /home/jmariett/scratch/tmp

[softwares]
blastall = /usr/bin/blastall
formatdb = /usr/bin/formatdb
sfffile = /usr/bin/sfffile
fastqc = /usr/bin/fastqc
runAssembly = /usr/bin/runAssembly
bwa = /usr/bin/bwa
samtools = /usr/bin/samtools
```

JFLOW sources structure

bin /	
_preamble.py	
jflow_cli.py	the jflow CLI user interface
jflow_server.py	the jflow REST service script
src /	
cctools /	makeflow python library
jflow /	the jflow source code (Workflow, Component classes)
js /	jquery plugins for the WEB interface
weaver /	the weaver package
workflows /	
__init__.py	
<myworkflow> /	the workflows package gathering all workflow to provide to the final user
__init__.py	
workflows.properties	
application.properties	the jflow config file
index.html	an HTML template example to use the jquery plugins
README	

JFLOW components

- Where can be added a component?
  - In the workflows package
  - In the created workflow package

- How is structured a component?
  A component is composed by:
  - a module file: a class inheriting from the Component class and implementing the define_parameter() and the process() function

JFLOW components

- Where can be added a componer
  - In the workflows pacl
  - In the created workfl
- How is structured a component?
  A component is composed by:
  - a module file: a class t class
    and implementing the
    process() function

```
bin /
        _preamble.py
        jflow_cli.py
        jflow_server.py

src /
        cctools /
        jflow /
        js /
        weaver /

workflows /
        __init__.py
       components /
       <myworkflow> /
             components /
              __init__.py
              workflows.properties

application.properties
index.html
README
```

JFLOW components

```
import os
from jflow.component import Component
from jflow.iotypes import OutputFile, InputFile, Formats
from weaver.function import ShellFunction

class BWAIndex (Component):

    def define_parameters(self, input_fasta, algorithm="bwtsw"):
        self.input_fasta = InputFile(input_fasta, Formats.FASTA)
        self.algorithm = algorithm
        self.databank = OutputFile(os.path.join(self.output_directory, os.path.basename(input_fasta)))
        self.stdout = OutputFile(os.path.join(self.output_directory, "bwaindex.stdout"))
        self.stderr = OutputFile(os.path.join(self.output_directory, "bwaindex.stderr"))

    def process(self):
        # first make the symbolic link
        os.symlink(self.input_fasta, self.databank)
        bwaindex = ShellFunction(self.get_exec_path("bwa") + " index -a " + self.algorithm + " -p $1 $2 > " + \
                                 self.stdout + " 2> " + self.stderr, cmd_format='{EXE} {OUT} {IN}')
        bwaindex(inputs=self.input_fasta, outputs=self.databank)
```

geno toul bioinfo

JFLOW components

```
import os
from jflow.component import Component
from jflow.iotypes import OutputFile, InputFile, Formats
from weaver.function import ShellFunction

class BWAIndex (Component):

    def define_parameters(self, input_fasta, algorithm="bwtsw"):
        self.input_fasta = InputFile(input_fasta, Formats.FASTA)
        self.algorithm = algorithm
        self.databank = OutputFile(os.path.join(self.output_directory, os.path.basename(input_fasta)))
        self.stdout = OutputFile(os.path.join(self.output_directory, "bwaindex.stdout"))
        self.stderr = OutputFile(os.path.join(self.output_directory, "bwaindex.stderr"))

    def process(self):
        # first make the symbolic link
        os.symlink(self.input_fasta, self.databank)
        bwaindex = ShellFunction(self.get_exec_path("bwa") + " index -a " + self.algorithm + " -p $1 $2 > " + \
                                 self.stdout + " 2> " + self.stderr, cmd_format='{EXE} {OUT} {IN}')
        bwaindex(inputs=self.input_fasta, outputs=self.databank)
```

- Inherit from the Component class

JFLOW components

```
import os
from jflow.component import Component
from jflow.iotypes import OutputFile, InputFile, Formats
from weaver.function import ShellFunction

class BWAIndex (Component):

    def define_parameters(self, input_fasta, algorithm="bwtsw"):
        self.input_fasta = InputFile(input_fasta, Formats.FASTA)
        self.algorithm = algorithm
        self.databank = OutputFile(os.path.join(self.output_directory, os.path.basename(input_fasta)))
        self.stdout = OutputFile(os.path.join(self.output_directory, "bwaindex.stdout"))
        self.stderr = OutputFile(os.path.join(self.output_directory, "bwaindex.stderr"))

    def process(self):
        # first make the symbolic link
        os.symlink(self.input_fasta, self.databank)
        bwaindex = ShellFunction(self.get_exec_path("bwa") + " index -a " + self.algorithm + " -p $1 $2 > " + \
                                 self.stdout + " 2> " + self.stderr, cmd_format='{EXE} {OUT} {IN}')
        bwaindex(inputs=self.input_fasta, outputs=self.databank)
```

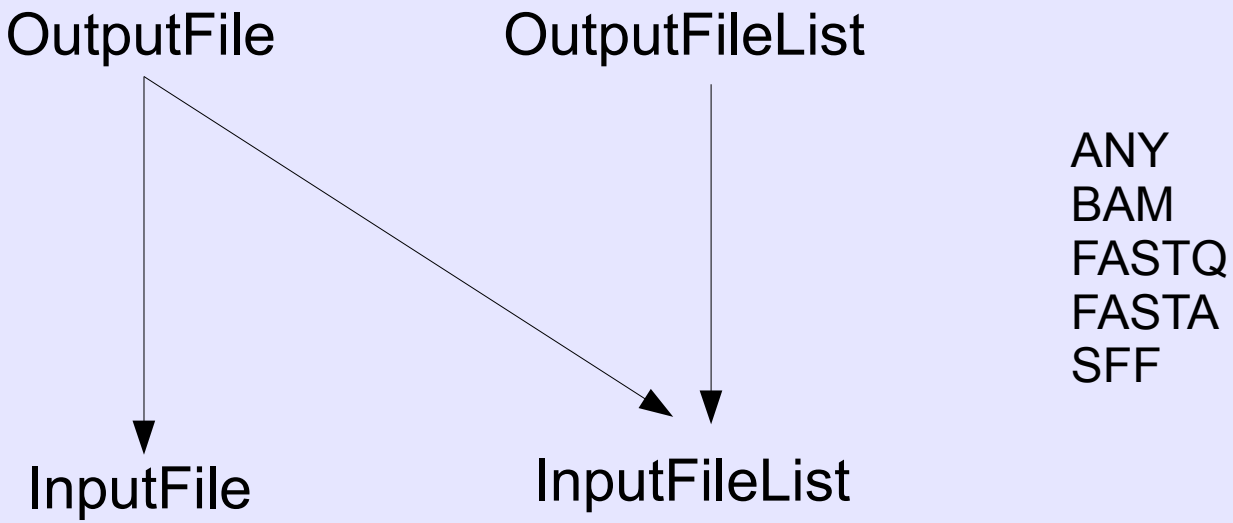
- Inherit from the Component class

JFLOW components

```
import os
from jflow.component import Component
from jflow.iotypes import OutputFile, InputFile, Formats
from weaver.function import ShellFunction

class BWAIndex (Component):

    def define_parameters(self, input_fasta, algorithm="bwtsw"):
        self.input_fasta = InputFile(input_fasta, Formats.FASTA)
        self.algorithm = algorithm
        self.databank = OutputFile(os.path.join(self.output_directory, os.path.basename(input_fasta)))
        self.stdout = OutputFile(os.path.join(self.output_directory, "bwaindex.stdout"))
        self.stderr = OutputFile(os.path.join(self.output_directory, "bwaindex.stderr"))

    def process(self):
        # first make the symbolic link
        os.symlink(self.input_fasta, self.databank)
        bwaindex = ShellFunction(self.get_exec_path("bwa") + " index -a " + self.algorithm + " -p $1 $2 > " + \
                                 self.stdout + " 2> " + self.stderr, cmd_format='{EXE} {OUT} {IN}')
        bwaindex(inputs=self.input_fasta, outputs=self.databank)
```

- Inherit from the Component class
- Overload the define_parametes function to define inputs and outputs

JFLOW components



ANY
BAM
FASTQ
FASTA
SFF

JFLOW components

```
import os
from jflow.component import Component
from jflow.iotypes import OutputFile, InputFile, Formats
from weaver.function import ShellFunction

class BWAIndex (Component):

    def define_parameters(self, input_fasta, algorithm="bwtsw"):
        self.input_fasta = InputFile(input_fasta, Formats.FASTA)
        self.algorithm = algorithm
        self.databank = OutputFile(os.path.join(self.output_directory, os.path.basename(input_fasta)))
        self.stdout = OutputFile(os.path.join(self.output_directory, "bwaindex.stdout"))
        self.stderr = OutputFile(os.path.join(self.output_directory, "bwaindex.stderr"))

    def process(self):
        # first make the symbolic link
        os.symlink(self.input_fasta, self.databank)
        bwaindex = ShellFunction(self.get_exec_path("bwa") + " index -a " + self.algorithm + " -p $1 $2 > " + \
                                 self.stdout + " 2> " + self.stderr, cmd_format='{EXE} {OUT} {IN}')
        bwaindex(inputs=self.input_fasta, outputs=self.databank)
```

- Inherit from the Component class
- Overload the define_parametes function to define inputs and outputs
- Overload the process function to define the component execution using weaver concepts

JFLOW workflows

- Where can be added a workflow ?
  So far only on a workflow python package, which has to be on the python path. The default one is the workflow directory at the root path of sources, any other idea ?

- How is structured a workflow ?
  A workflow is composed by:
  - a workflow.properties file: gather all information on the workflow (name, description, parameters)
  - a __init__.py file: a class inheriting from the Workflow class and implementing the process() function

JFLOW workflows

- Where can be added a workflow ?
  So far only on a workflow python p… e
python path. The default one is the …ath of
sources, any other idea ?

- How is structured a workflow ?
  A workflow is composed by:
  - a workflow.properties … he
    workflow (name, des…
  - a __init__.py file: a c… w class
    and implementing the…

```
bin /
        _preamble.py
        jflow_cli.py
        jflow_server.py

src /
        cctools /
        jflow /
        js /
        weaver /

workflows /
        __init__.py
       components /
       <myworkflow> /
              components /
               __init__.py
               workflows.properties

application.properties
index.html
README
```

JFLOW workflows / workflow.properties

- Define workflow name & description
- Define workflow parameters
  - parameter name: how should this parameter be displayed to the user and how can I retrieve this parameter from my script
  - parameter type: which type should be tested from both cli and WEB interfaces
  - parameter choices: is this parameter limited to some values
  - ...

JFLOW workflows / workflow.properties

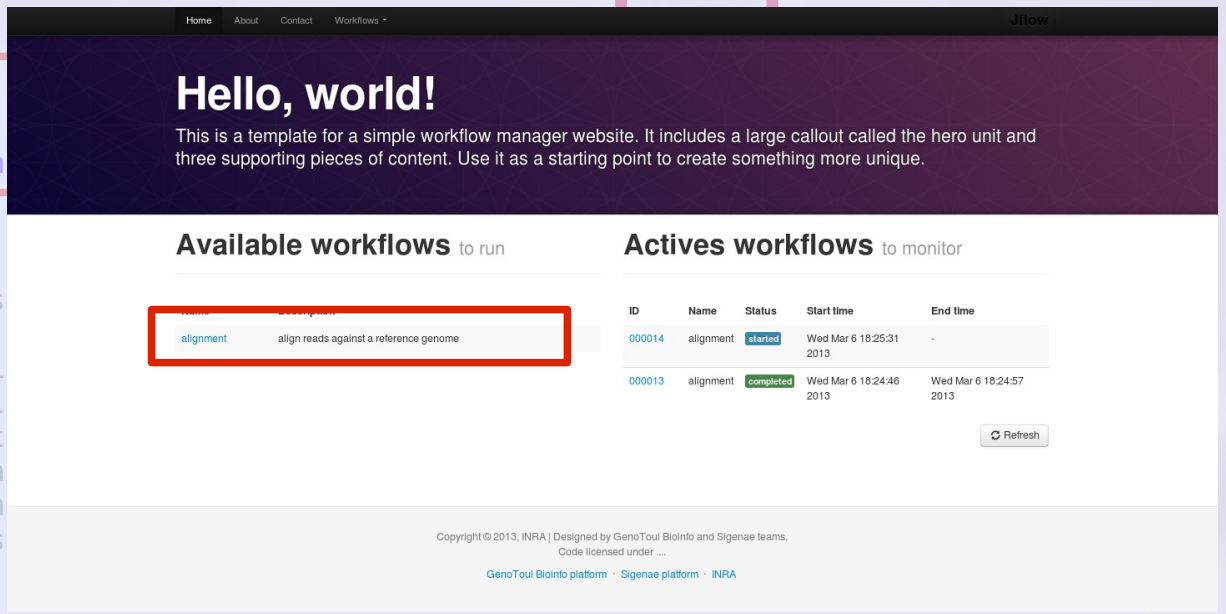
```
[global]
name = alignment
description = align reads against a reference genome

#
# Parameter section
# param.name: the parameter display name
#       .flag: the command line flag to use the argument
#       .help: a brief description of what the parameter does
#       .default [None]: the value produced if the parameter is not provided
#       .type [str]: the parameter type that should be tested (str|int|date|file|bool)
#       .choices [None]: a container of the allowable values for the parameter
#       .required [False]: whether or not the command-line option may be omitted
#       .action [store]: the basic type of action to be taken (store|append)
#
[parameters]
read_1.name = read_1
read_1.flag = --read-1
read_1.help = Which read1 files should be used
read_1.required = True
read_1.action = append

read_2.name = read_2
read_2.flag = --read-2
read_2.help = Which read2 files should be used (if single end, leave empty)
read_2.action = append

reference_genome.name = reference_genome
reference_genome.flag = --reference-genome
reference_genome.help = Which genome should the read being align on
reference_genome.required = True
```

JFLOW workflows / workflow.properties

```
[global]
name = alignment
description = align reads against a reference genome

#
# Parameter section
# param.name: the parameter display name
#       .flag: the command line flag to use the argument
#       .help: a brief description of what the parameter does
#       .default [None]: the value produced if the parameter is not provided
#       .type [str]: the parameter type that should be tested (str|int|date|file|bool)
#       .choices [None]: a container of the allowable values for the parameter
#       .required [False]: whether or not the command-line option may be omitted
#       .action [store]: the basic type of action to be taken (store|append)
#
[parameters]
read_1.name = read_1
read_1.flag = --read-1
read_1.help = Which read1 files should be used
read_1.required = True
read_1.action = append

read_2.name = read_2
read_2.flag = --read-2
read_2.help = Which read2 files should be used (if single end, leave empty)
read_2.action = append

reference_genome.name = reference_genome
reference_genome.flag = --reference-genome
reference_genome.help = Which genome should the read being align on
reference_genome.required = True
```

JFLOW workflows / workflow.properties

```
[global]
name = alignment
description = align reads again

#
# Parameter section
# param.name: the parameter dis
#       .flag: the command line
#       .help: a brief descripti
#       .default [None]: the val
#       .type [str]: the paramet
#       .choices [None]: a conta
#       .required [False]: wheth
#       .action [store]: the bas
#
[parameters]
read_1.name = read_1
read_1.flag = --read-1
read_1.help = Which read1 files should be used
read_1.required = True
read_1.action = append

read_2.name = read_2
read_2.flag = --read-2
read_2.help = Which read2 file
read_2.action = append

reference_genome.name = refere
reference_genome.flag = --refe
reference_genome.help = Which
reference_genome.required = True
```




```
[heid:~/data] python /home/jmariett/workspace/jflow/trunk/bin/jflow_cli.py --help
usage: jflow_cli.py [-h] {rerun,status,alignment} ...

optional arguments:
  -h, --help            show this help message and exit

Available sub commands:
  {rerun,status,alignment}
    rerun               rerun a sepcific workflow
    status              monitor a specific workflow
    alignment           align reads against a reference genome
```

JFLOW workflows / workflow.properties

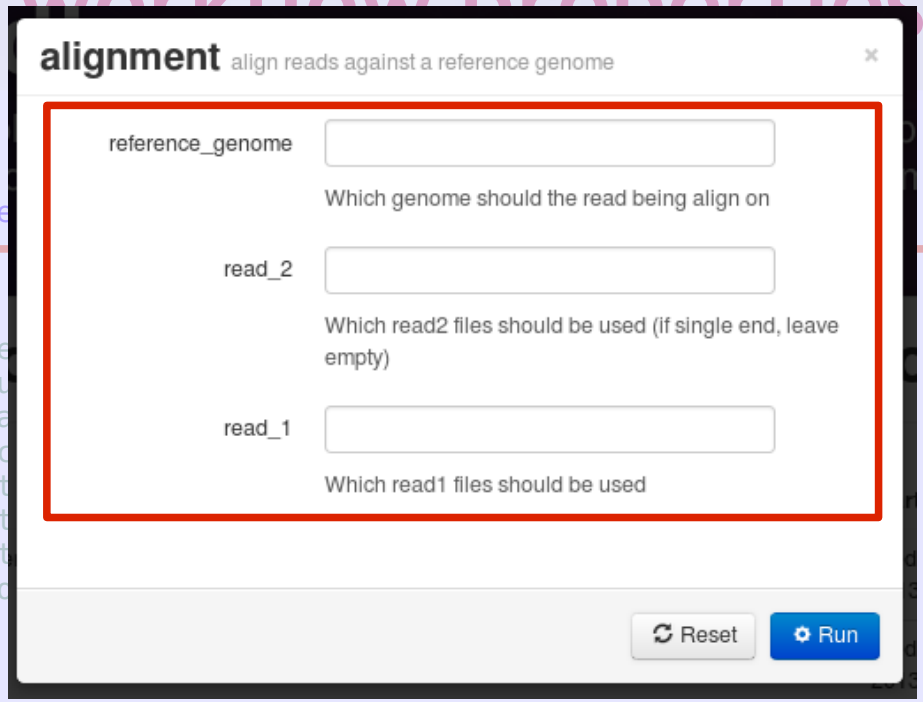
```
[global]
name = alignment
description = align reads against a reference genome
```

```
#
# Parameter section
# param.name: the parameter display name
#       .flag: the command line flag to use the argument
#       .help: a brief description of what the parameter does
#       .default [None]: the value produced if the parameter is not provided
#       .type [str]: the parameter type that should be tested (str|int|date|file|bool)
#       .choices [None]: a container of the allowable values for the parameter
#       .required [False]: whether or not the command-line option may be omitted
#       .action [store]: the basic type of action to be taken (store|append)
#
[parameters]
read_1.name = read_1
read_1.flag = --read-1
read_1.help = Which read1 files should be used
read_1.required = True
read_1.action = append

read_2.name = read_2
read_2.flag = --read-2
read_2.help = Which read2 files should be used (if single end, leave empty)
read_2.action = append

reference_genome.name = reference_genome
reference_genome.flag = --reference-genome
reference_genome.help = Which genome should the read being align on
reference_genome.required = True
```

geno toul bioinfo

JFLOW workflows / workflow properties

```
[global]
name = alignment
description = align reads against a refe

#
# Parameter section
# param.name: the parameter display name
#       .flag: the command line flag to u
#       .help: a brief description of wha
#       .default [None]: the value produc
#       .type [str]: the parameter type t
#       .choices [None]: a container of t
#       .required [False]: whether or not
#       .action [store]: the basic type o
#
[parameters]
read_1.name = read_1
read_1.flag = --read-1
read_1.help = Which read1 fil
read_1.required = True
read_1.action = append

read_2.name = read_2
read_2.flag = --read-2
read_2.help = Which read2 fil
read_2.action = append

reference_genome.name = refer
reference_genome.flag = --ref
reference_genome.help = Which genome should the read being align on
reference_genome.required = True
```




```
[heid:~/data] python /home/jmariett/workspace/jflow/trunk/bin/jflow_cli.py alignment --help
usage: jflow_cli.py alignment [-h] --reference-genome STR [--read-2 STR]
                              --read-1 STR

optional arguments:
  -h, --help            show this help message and exit
  --reference-genome STR
                        Which genome should the read being align on
  --read-2 STR          Which read2 files should be used (if single end, leave
                        empty)
  --read-1 STR          Which read1 files should be used
```

JFLOW workflows / workflow.properties

```
[global]
name = alignment
description = align reads against a reference genome

#
# Parameter section
# param.name: the parameter display name
#       .flag: the command line flag to use the argument
#       .help: a brief description of what the parameter does
#       .default [None]: the value produced if the parameter is not provided
#       .type [str]: the parameter type that should be tested (str|int|date|file|bool)
#       .choices [None]: a container of the allowable values for the parameter
#       .required [False]: whether or not the command-line option may be omitted
#       .action [store]: the basic type of action to be taken (store|append)
#
[parameters]
read_1.name = read_1
read_1.flag = --read-1
read_1.help = Which read1 files should be used
read_1.required = True
read_1.action = append

read_2.name = read_2
read_2.flag = --read-2
read_2.help = Which read2 files should be used (if single end, leave empty)
read_2.action = append

reference_genome.name = reference_genome
reference_genome.flag = --reference-genome
reference_genome.help = Which genome should the read being align on
reference_genome.required = True
```

geno toul bioinfo

JFLOW workflows / __init__.py

```
from jflow.workflow import Workflow


class Alignment (Workflow):

    def process(self):
        """
        Run the workflow
        """
        # index the reference genome
        bwaindex = self.add_component("BWAIndex", [self.args["reference_genome"]])
        # align reads against indexed genome
        bwa = self.add_component("BWA", [bwaindex.databank, self.args["read_1"], self.args["read_2"]])
```

JFLOW workflows / __init__.py

```
from jflow.workflow import Workflow

class Alignment (Workflow):

    def process(self):
        """
        Run the workflow
        """
        # index the reference genome
        bwaindex = self.add_component("BWAIndex", [self.args["reference_genome"]])
        # align reads against indexed genome
        bwa = self.add_component("BWA", [bwaindex.databank, self.args["read_1"], self.args["read_2"]])
```

- Inheriting from the Workflow class

JFLOW workflows / __init__.py

```
from jflow.workflow import Workflow


class Alignment (Workflow):

    def process(self):
        """
        Run the workflow
        """
        # index the reference genome
        bwaindex = self.add_component("BWAIndex", [self.args["reference_genome"]])
        # align reads against indexed genome
        bwa = self.add_component("BWA", [bwaindex.databank, self.args["read_1"], self.args["read_2"]])
```

- Inheriting from the Workflow class
- Implementing the process() function to define the workflow execution

JFLOW workflows / __init__.py

```
from jflow.workflow import Workflow

class Alignment (Workflow):

    def process(self):
        """
        Run the workflow
        """
        # index the reference genome
        bwaindex = self.add_component("BWAIndex", [self.args["reference_genome"]])
        # align reads against indexed genome
        bwa = self.add_component("BWA", [bwaindex.databank, self.args["read_1"], self.args["read_2"]])
```

- Inheriting from the Workflow class
- Implementing the process() function to define the workflow execution
- Add components (`"BWAIndex"`) and provide the component arguments (`[self.args["reference_genome"]]`)

JFLOW workflows / __init__.py

```
from jflow.workflow import Workflow


class Alignment (Workflow):

    def process(self):
        """
        Run the workflow
        """
        # index the reference genome
        bwaindex = self.add_component("BWAIndex", [self.args["reference_genome"]])
        # align reads against indexed genome
        bwa = self.add_component("BWA", [bwaindex.databank, self.args["read_1"], self.args["read_2"]])
```

- Inheriting from the Workflow class
- Implementing the process() function to define the workflow execution
- Add components (`"BWAIndex"`) and provide the component arguments (`[self.args["reference_genome"]]`)
- Link components inputs and outputs (typing test is done)

JFLOW workflows / __init__.py

```
[parameters]
read_1.name = read_1
read_1.flag = --read-1
read_1.help = Which read1 files should be used
read_1.required = True
read_1.action = append

read_2.name = read_2
read_2.flag = --read-2
read_2.help = Which read2 files should be used (if single end, leave empty)
read_2.action = append

reference_genome.name = reference_genome
reference_genome.flag = --reference-genome
reference_genome.help = Which genome should the read being align on
reference_genome.required = True
```

```
from jflow.workflow import Workflow


class Alignment (Workflow):

    def process(self):
        """
        Run the workflow
        """
        # index the reference genome
        bwaindex = self.add_component("BWAIndex", [self.args["reference_genome"]])
        # align reads against indexed genome
        bwa = self.add_component("BWA", [bwaindex.databank, self.args["read_1"], self.args["read_2"] ])
```

JFLOW outputs

```
outputs /
    <workflow_name> /
        wf<workflow_id> /
            workflow.dump
            <component_name>_<prefix>
                <component_outputs>
            […]
            .working/
                <working_dir> /
                    Makeflow
                    Makeflow.makeflowlog
                    _Stash /
        […]
    […]
```

JFLOW give it a try …

… and make a workflow !!!