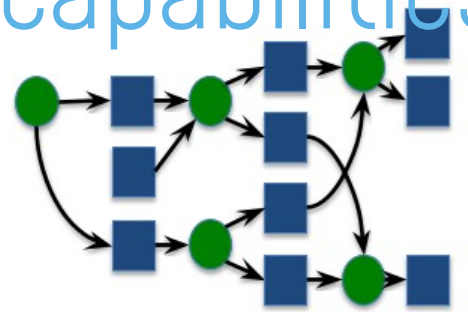
Make(file) + (Work)flow = Makeflow

damien.francois@uclouvain.be UCLouvain Calcul Intensif et Stockage de Masse

Goal of this session:

Get an overview of the capabilities of **Makeflow** and of its usage.



http://ccl.cse.nd.edu/software/makeflow/

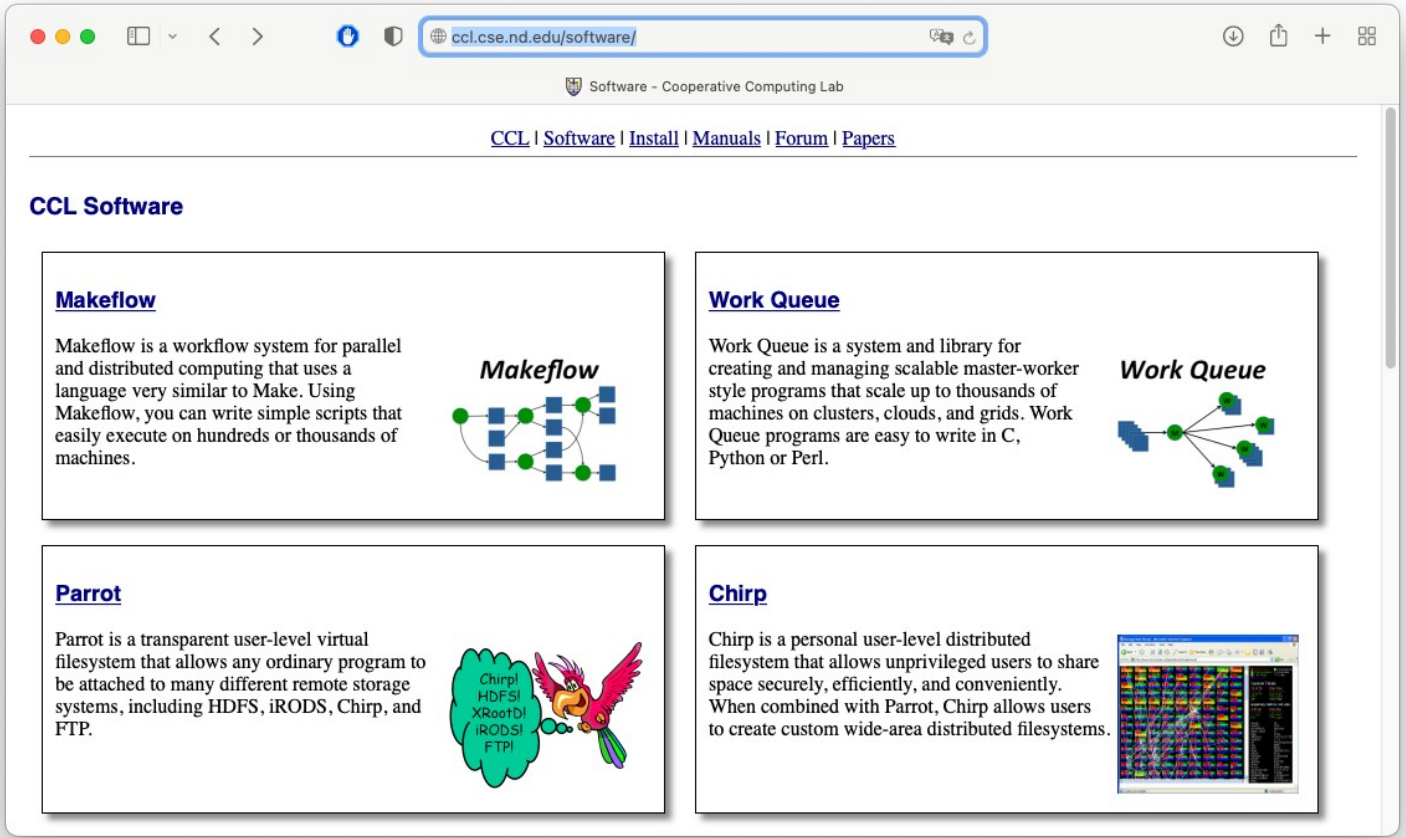
About

"Makeflow is a workflow system for parallel and distributed computing that uses a language very similar to Make. Using Makeflow, you can write simple scripts that easily execute on hundreds or thousands of machines."

Part of a larger ecosystem: cctools

http://ccl.cse.nd.edu/software/

About



http://ccl.cse.nd.edu/software/

Topics covered:

- Installation
- Syntax
- Run, monitoring, etc.
- Interaction with the scheduler

Installation

Installation

Download archive from the website, then the usual

```
$ wget http://ccl.cse.nd.edu/software/files/cctools-6.0.16-source.tar.gz

$ tar zxpvf cctools-6.0.16-source.tar.gz
$ cd cctools-6.0.16-source
$ ./configure --prefix $HOME/cctools
$ make
$ make install
$ cd ~

$ echo “export PATH=$PATH:$HOME/cctools/bin” >> .bashrc
```

http://ccl.cse.nd.edu/software/downloadfiles.php

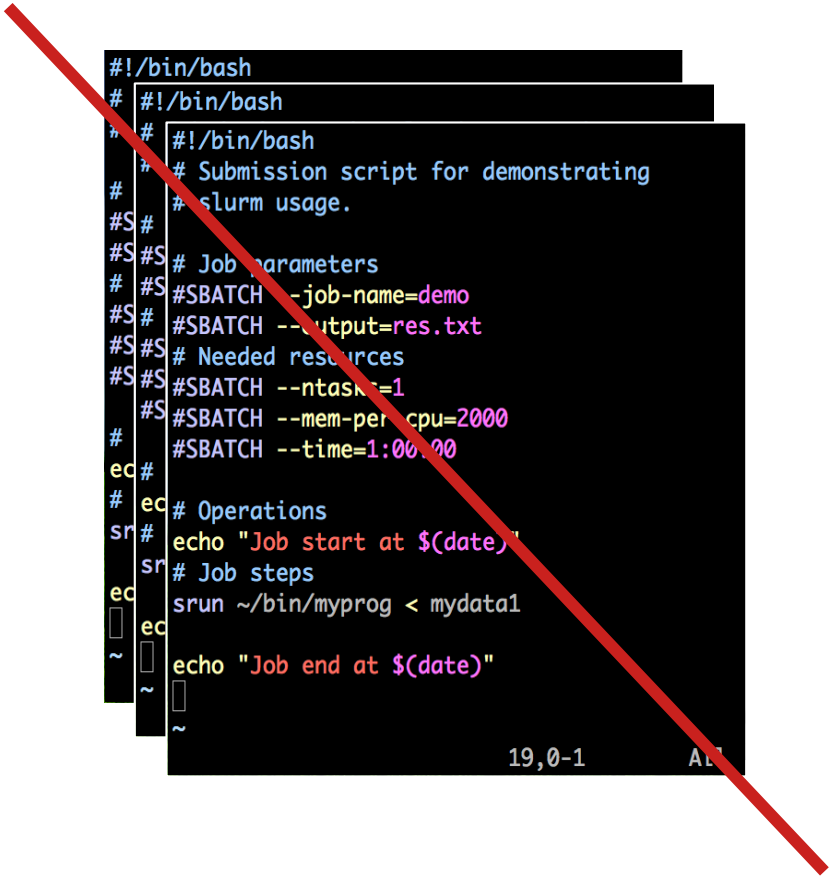
Installation

Or use Easybuild

```
$ eb cctools-7.0.22-GCCcore-8.3.0.eb
$ ml use.own cctools
```

Syntax

Syntax


#!/bin/bash
#!/bin/bash
#!/bin/bash
Submission script for demonstrating
slurm usage.
Job parameters
#SBATCH --job-name=demo
#SBATCH --output=res.txt
Needed resources
#SBATCH --ntasks=1
#SBATCH --mem-per-cpu=2000
#SBATCH --time=1:00:00
Operations
echo "Job start at $(date)"
Job steps
srun ~/bin/myprog < mydata1
echo "Job end at $(date)"
19,0-1

Syntax

Makeflow file : files dependencies

```
# Simple makeflow file to build an archive

GZ=module load gzip ; gzip -k -f

archive.tar.b.gz: archive.tar
    $(GZ) --best -S.b.gz archive.tar

archive.tar.f.gz: archive.tar
    $(GZ) --fast -S.f.gz archive.tar

archive.tar:  directory/file1.txt directory
    tar cvf archive.tar directory

directory directory/file1.txt:
    mkdir -p directory ; touch directory/file1.txt
```

Comment

Variable definition

Rule 1

Rule 2

Rule 3

Rule 4

Syntax

Anatomy of a rule:

```
output file(s): input file(s) or directory
    [LOCAL] command(s) to generate output(s) from input(s)
```

- Commands must be on a single line (`;`-separated)
- Commands prefixed with `LOCAL` are not submitted to the scheduler
- Just plain rules (less powerful than regular Makefiles)

Variables

Variables are scoped:

```
SOME_VARIABLE=original_value

target_1: source_1
    command_1

target_2: source_2
SOME_VARIABLE=local_value_for_2
    command_2
```

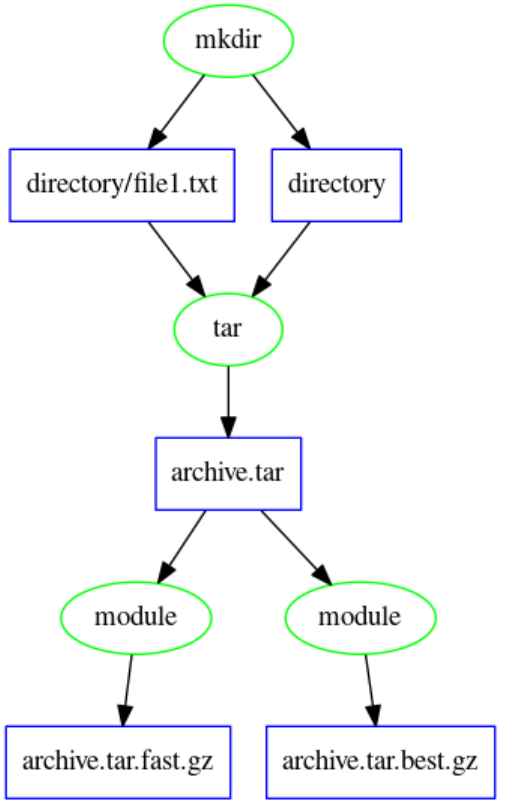
Environment variables can be set:

```
export PATH=/opt/bin/:${PATH}

export USER
```

Graph

Dependency graph:

```
$ makeflow_viz -D dot build_archive.makeflow \|
      dot -Tpng > build_archive.png
```



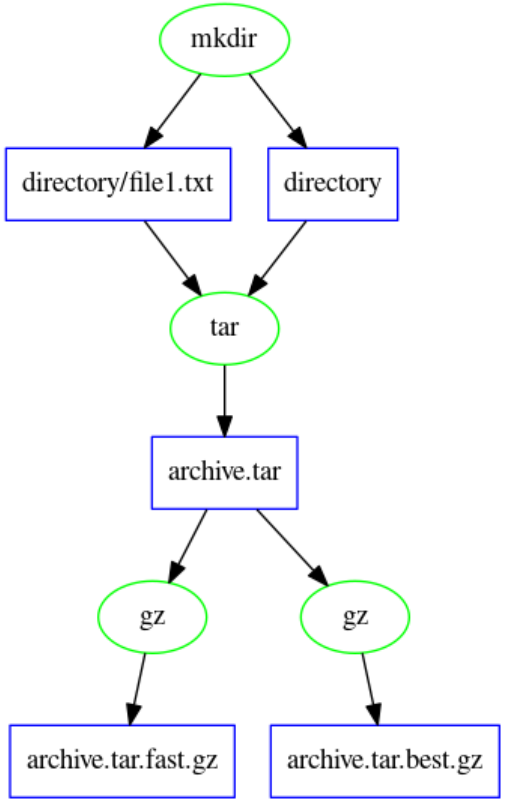
- easy way to check the correctness
- easy way to communicate about it

Graph

Dependency graph:

```
$ makeflow_viz -D dot build_archive.makeflow \|
      sed ‘s/module/gz/’ \|
      dot -Tpng > build_archive.png
```

- some manual corrections could be needed



Syntax check

You can check the file before running it

```
$ makeflow_analyze -k build_archive.makeflow
build_archive.makeflow: Syntax OK.

$ makeflow_analyze -O build_archive.makeflow
archive.tar.best.gz
directory/file1.txt
directory
archive.tar.fast.gz
archive.tar
```

Syntax check

List outputs

Running, monitoring

Run workflow

```
$ makeflow build_archive.makeflow

parsing build_archive.makeflow…
local resources: 24.000 cores, 95346 MB memory
max running local jobs: 24

checking build_archive.makeflow for consistency…
build_archive.makeflow has 4 rules.

starting workflow....

submitting job: mkdir -p directory ; touch
directory/file1.txt
submitted job 153166
job 153166 completed

[…]
```

Prolog

Rule 4

Run workflow

```
[…]

submitting job: tar cvzf archive.tar directory
submitted job 153169
directory/file1.txt
job 153169 completed

submitting job: module load gzip ;
        gzip -k --fast -S.fast.gz archive.tar
submitted job 153171
submitting job: module load gzip ;
        gzip -k --best -S.best.gz archive.tar
submitted job 153172
job 153171 completed
job 153172 completed

nothing left to do.
```

Rule 3

Rules 1 and 2 running in parallel

Run workflow

- Makeflow is blocking: you must keep connection to submission host or use terminal multiplexer.

- Second run does nothing (like GNU Make)

```
$ makeflow build_archive.makeflow
parsing build_archive.makeflow...

[…]

starting workflow.…
nothing left to do.
```

Run workflow

Cleaning:

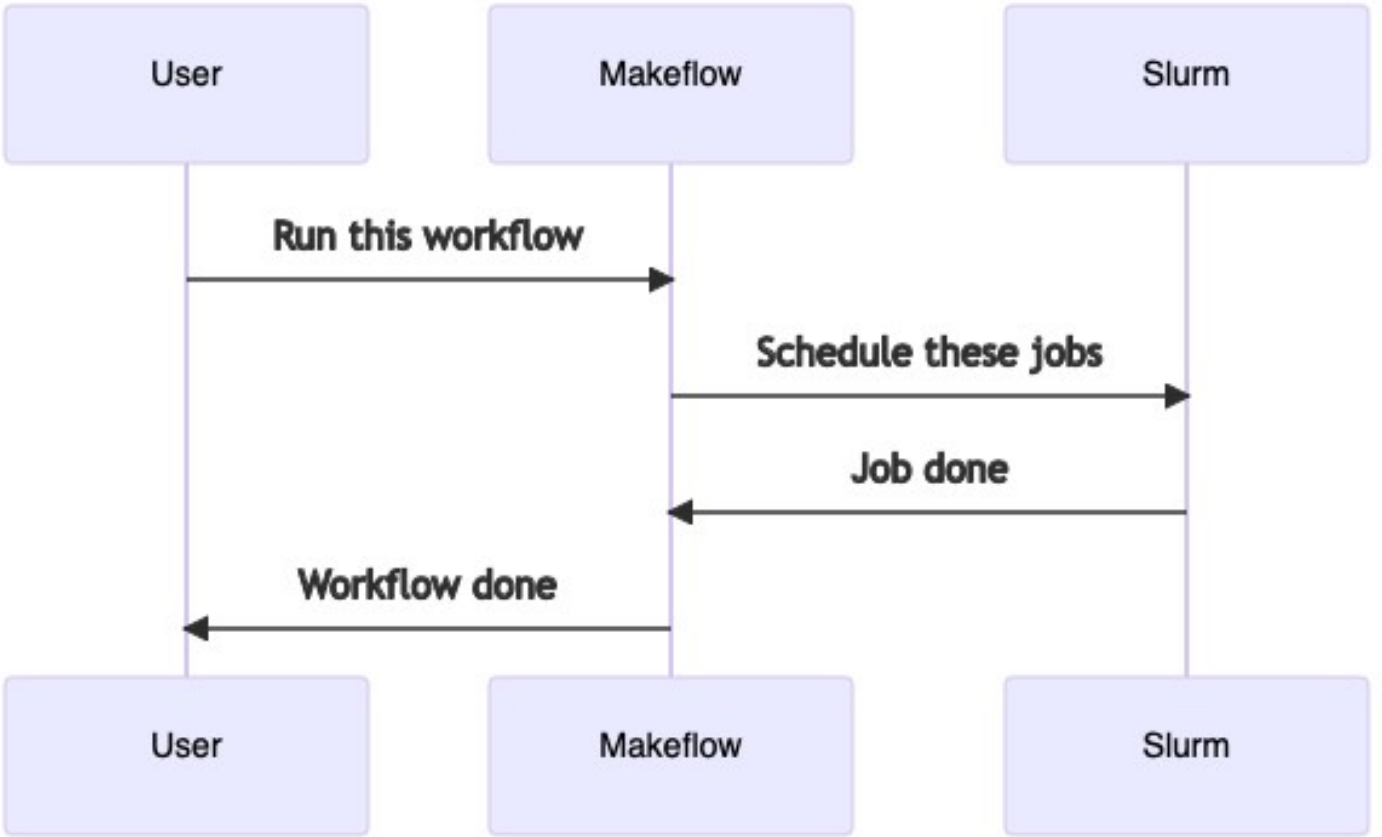
```
$ makeflow build_archive.makeflow --clean

parsing build_archive.makeflow...

[…]

cleaning filesystem…
deleted archive.tar.best.gz
deleted directory/file1.txt
deleted directory
deleted archive.tar.fast.gz
deleted archive.tar

nothing left to do.
```

Scheduler interaction

3 modes of operation

1. Makeflow submits jobs
2. Makeflow runs inside an allocation
3. Work queues

Makeflow runs on the login node



Run workflow

Many compatible schedulers

```
$ makeflow --help |& grep -- -T,

 -T,--batch-type=<type> Select batch system: local, wq,
condor, sge, pbs, lsf, torque, moab, mpi, slurm, chirp,
amazon, amazon-batch, lambda, mesos, k8s, dryrun
```

and scheduler types (HPC, Cloud, etc.).

The default is `local`. `dryrun` explains what would be done but does not actually do it.

Run workflow

```
$ makeflow -T slurm build_archive.makeflow

local resources: 24.000 cores, 95346 MB memory
running remote jobs: 100
max running local jobs: 24

checking build_archive.makeflow for consistency…
build_archive.makeflow has 4 rules.

starting workflow....

submitting job: mkdir -p directory ; touch
directory/file1.txt
submitted job 70383663
job 70383663 completed

[…]
```

Prolog

Rule 4

Run workflow

```
[…]

submitting job: tar cvzf archive.tar directory
submitted job 70383664
directory/file1.txt
job 70383664 completed

submitting job: module load gzip ;
        gzip -k --fast -S.fast.gz archive.tar
submitted job 70383665
submitting job: module load gzip ;
        gzip -k --best -S.best.gz archive.tar
submitted job 70383666
job 70383665 completed
job 70383666 completed

nothing left to do.
```

Rule 3

Rules 1 and 2 *submitted* in parallel

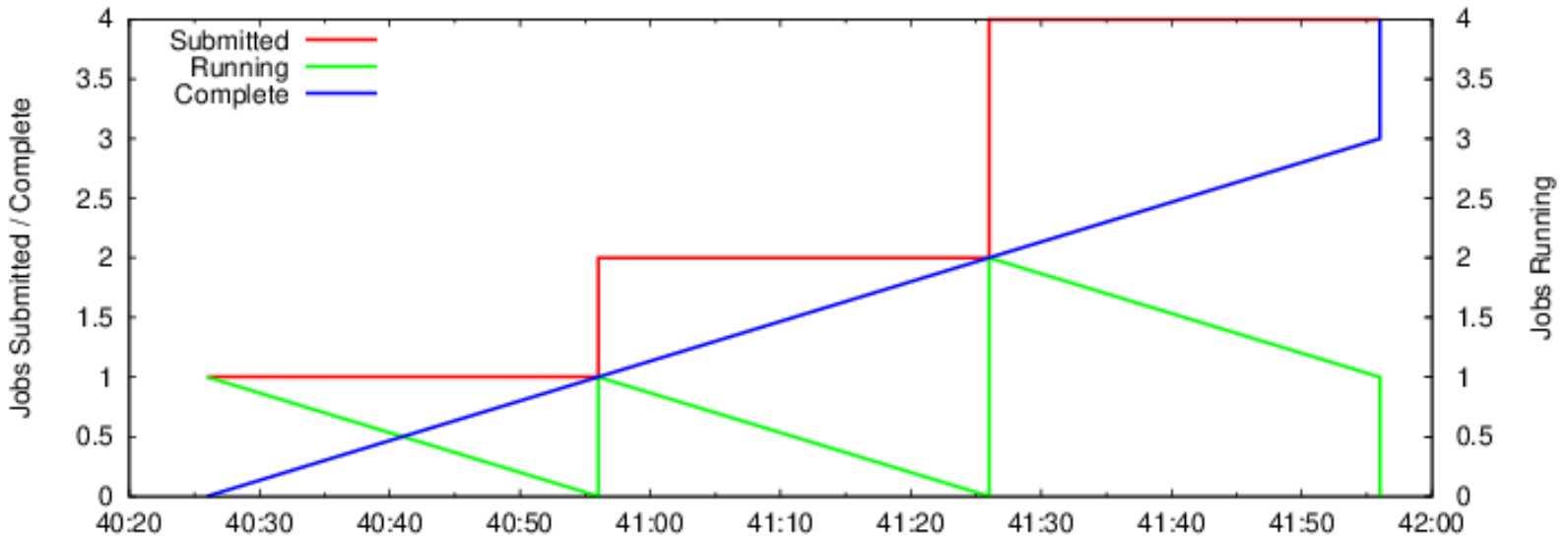
Run workflow

```
$ sacct -X -o jobid,jobname,state,start,elapsed
       JobID    JobName      State               Start    Elapsed
------------ ---------- ---------- ------------------- ----------
70383663      makeflow0  COMPLETED 2022-01-28T14:40:43   00:00:01
70383664      makeflow1  COMPLETED 2022-01-28T14:41:13   00:00:02
70383665      makeflow2  COMPLETED 2022-01-28T14:41:44   00:00:01
70383666      makeflow3  COMPLETED 2022-01-28T14:41:44   00:00:01
```

Run workflow

Graphical timeline:

```
$ makeflow_graph_log build_archive.makeflow.makeflowlog |\
> timeline.png
```



Scheduler options

Generic options:

```
# Simple makeflow file to build an archive

CORES=4
MEMORY=1024 # MB
WALLTIME=3600 # s

GZ=module load gzip ; gzip -k

archive.tar.b.gz: archive.tar
    $(GZ) --best -S.b.gz archive.best.tar

archive.tar.f.gz: archive.tar
CORES=8
    $(GZ) --fast -S.f.gz archive.fast.tar
```

Comment

Variable definition

Rule 1

Rule 2

Scheduler options

Scheduler-specific options:

```
# Simple makeflow file to build an archive

BATCH_OPTIONS=-c 4 --mem 1G --partition debug


GZ=module load gzip ; gzip -k

archive.tar.b.gz: archive.tar
    $(GZ) --best -S.b.gz archive.best.tar

archive.tar.f.gz: archive.tar
    $(GZ) --fast -S.f.gz archive.fast.tar

[…]
```

Comment

Variable definition

Rule 1

Rule 2

Scheduler options

Can also be passed in the command line

```
$ makeflow -B “-c 4 --mem 1G –partition debug” \|
    build_archive.makeflow
```

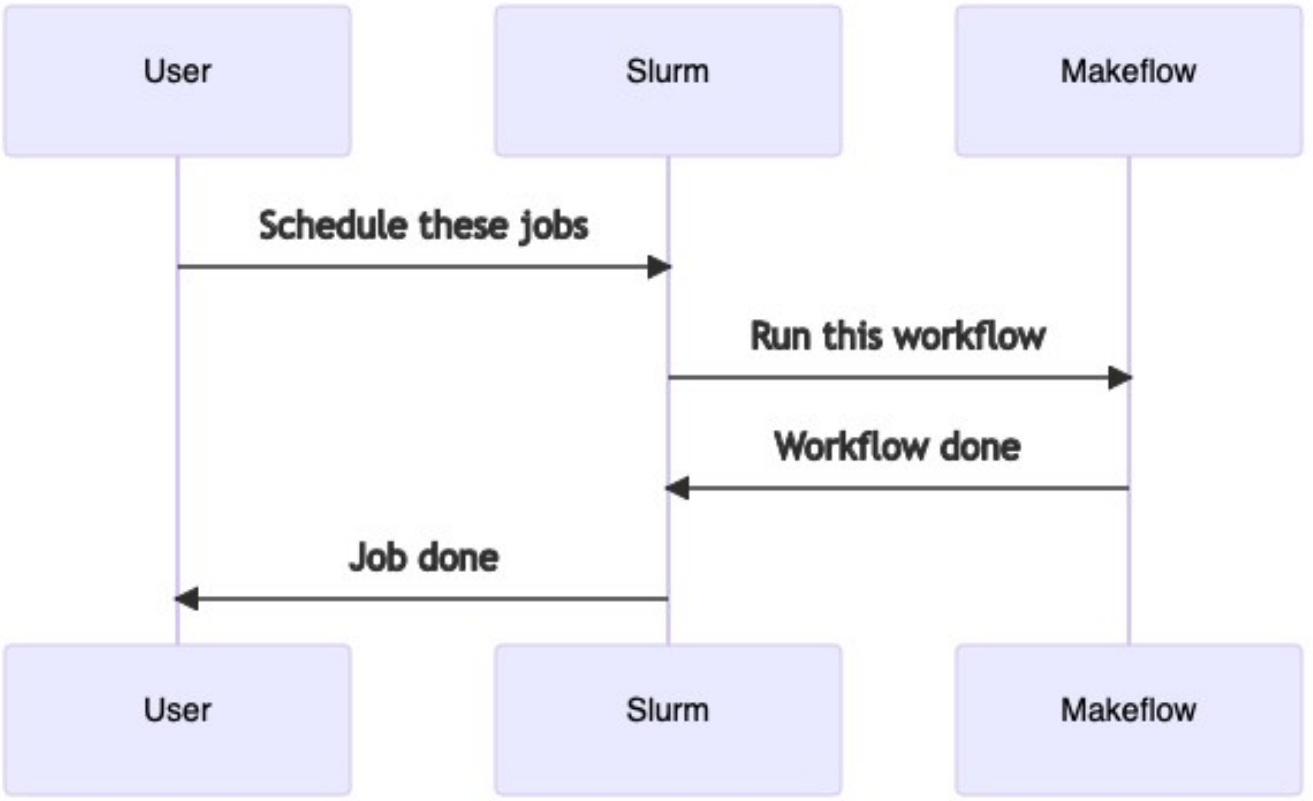
Makeflow copies files between nodes by default, unless

```
$ makeflow --shared-fs /home --shared-fs /scratch \
    build_archive.makeflow
```

3 modes of operation

1. Makeflow submits jobs
2. Makeflow runs inside an allocation
3. Work queues

Makeflow runs on the compute node



Scheduler options

Simply submit a job starting Makeflow

```
#! /bin/bash

#SBATCH -c 4
#SBATCH --mem 1G
#SBATCH --partition debug


makeflow -j $SLURM_NPROCS build_archive.makeflow
```

3 modes of operation

1. Makeflow submits jobs
2. Makeflow runs inside an allocation
3. Work queues

Workqueue (pilot jobs)

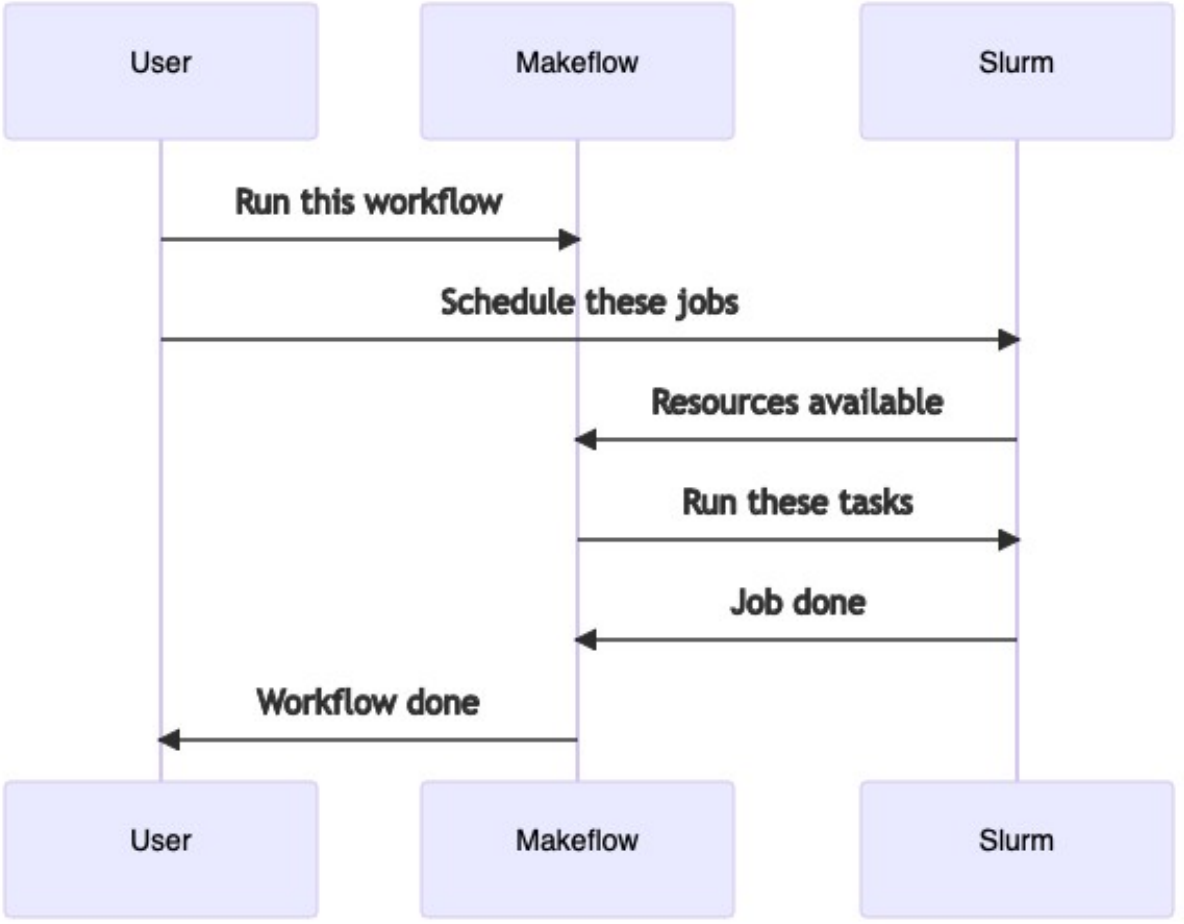
Makeflow does not submit jobs.

Rather it waits and listens for worker registration

Workers are started manually

https://cctools.readthedocs.io/en/latest/makeflow/#using-work-queue

Makeflow runs on a (cloud) server



Run workflow, blocked

```
$ makeflow -T wq build_archive.makeflow

parsing build_archive.makeflow…
local resources: 24.000 cores, 95346 MB memory,
max running remote jobs: 1000
max running local jobs: 24
checking build_archive.makeflow for consistency…
build_archive.makeflow has 4 rules.

starting workflow....

listening for workers on port 9123.

submitting job: mkdir -p directory ; touch
directory/file1.txt

submitted job 1
```

Prolog

Wq mode

Rule 4

Waiting

Start “worker” jobs

Start 2 jobs and point them to Makeflow instance

```
$ slurm_submit_workers frontend.cluster.hpc 9123 2

Creating worker submit scripts in dfr-workers…
Submitted batch job 70384127 on cluster hpc1
Submitted batch job 70384128 on cluster hpc1



$ squeue --me
CLUSTER: hpc1
             JOBID      NAME ST       TIME  NODELIST(REASON)
          70384127  wqWorker PD       0:00  (Resources)
          70384128  wqWorker PD       0:00  (Resources)
```

Unblocked, workflow running

```
$ makeflow -T wq build_archive.makeflow

[…]

submitted job 1
job 1 completed

submitting job: tar cvf archive.tar directory
submitted job 2
directory/
directory/file1.txt
job 2 completed

submitting job: module load gzip ; gzip -k -f --fast
-S.fast.gz archive.tar

[…]

nothing left to do.
```

Appeared as soon as one submitted job started

Resources for “worker” jobs

Specify scheduler options with -p

```
$ slurm_submit_workers -p “-p debug” \
    frontend.cluster.hpc 9123 2

Creating worker submit scripts in dfr-workers…
Submitted batch job 70384131 on cluster hpc1
Submitted batch job 70384132 on cluster hpc1
```

Or through env variables

```
$ SBATCH_PARTITION=debug slurm_submit_workers  \
    frontend.cluster.hpc 9123 2

Creating worker submit scripts in dfr-workers…
Submitted batch job 70384133 on cluster hpc1
Submitted batch job 70384134 on cluster hpc1
```

Starting workers manually

Worker advertise resources

Connects and is disconnected when workflow is finished. Eventually times out.

```
$ work_queue_worker frontend.cluster.hpc 9123

work_queue_worker: creating workspace /tmp/worker-3000003-153350

work_queue_worker: using 24 cores, 95346 MB memory, 41139 MB disk, 0 gpus

connected to manager frontend.cluster.hpc via local address 12.34.56.78:39604

disconnected from manager frontend.cluster.hpc:9123
```

Workqueue (pilot jobs)

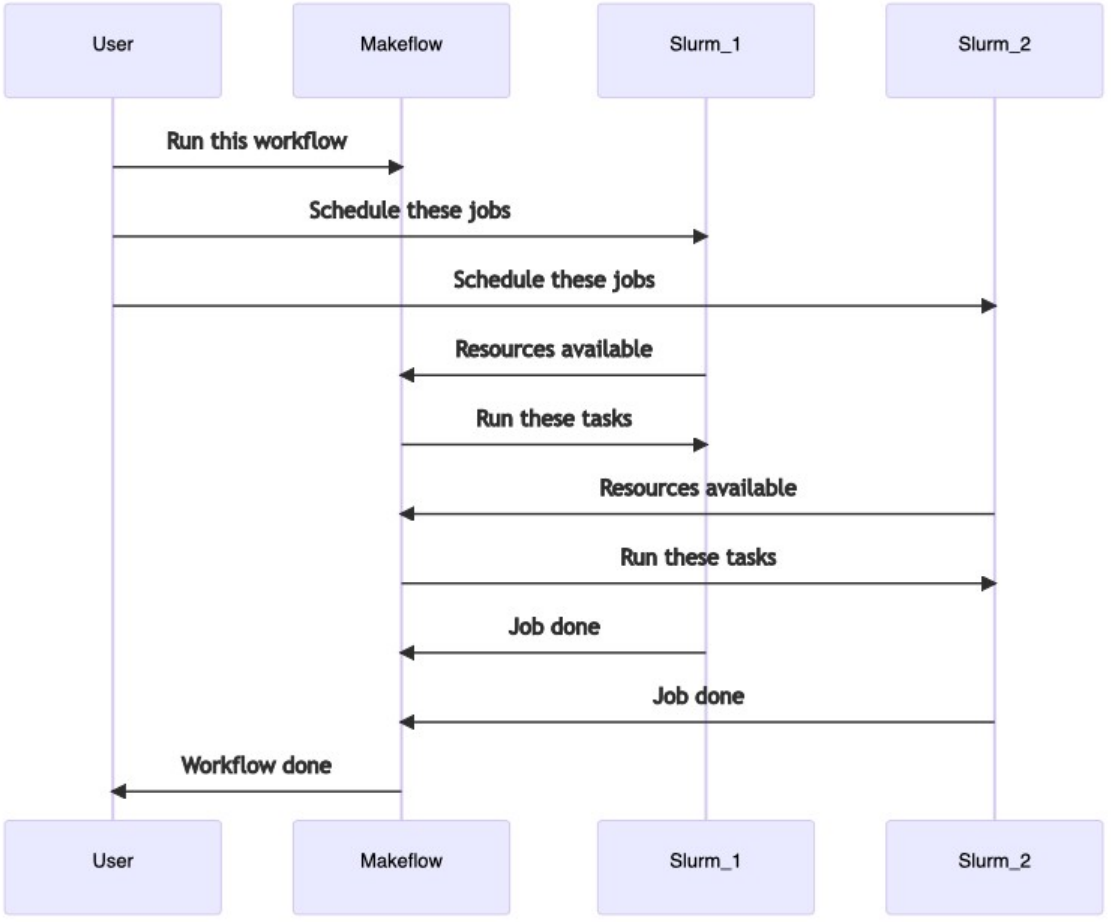
Makeflow does not submit jobs.

Rather it waits and listens for worker registration

Workers are started manually

→ enable resource pooling from multiple clusters

https://cctools.readthedocs.io/en/latest/makeflow/#using-work-queue


User
Makeflow
Slurm_1
Slurm_2
Run this workflow
Schedule these jobs
Schedule these jobs
Resources available
Run these tasks
Resources available
Run these tasks
Job done
Job done
Workflow done
User
Makeflow
Slurm_1
Slurm_2

Workqueue (pilot jobs)

Makeflow does not submit jobs.

Rather it waits and listens for worker registration

Workers are started manually

→ enable resource pooling from multiple clusters

But then need to manage infrastructure, beware of data transfers, be cautious of security, etc.

Topics covered:

- Installation
- Syntax
- Run, monitoring, etc.
- Interaction with the scheduler

Topics covered:

- Installation
- Syntax(es)
- Run, monitoring, etc.
- Interaction with the scheduler
- Nested workflows
- Singularity integration

Syntax again

Syntax 2: jx

Alternative syntax based on JSON:

```
# Simple makeflow file to build an archive
{
 "define": {
  "GZ": "module load gzip ; gzip -k -f "
 },
 "rules": [
  {
   "command": GZ+"--best -S.best.gz archive.tar",
   "inputs": ["archive.tar"],
   "outputs": ["archive.tar.best.gz"]
  },{
   "command": GZ+"--fast -S.fast.gz archive.tar",
[…]
```

Comment

Variable definition

Rule 1

Rule 2

https://cctools.readthedocs.io/en/latest/makeflow/#jx-language

Syntax 2: jx

Alternative syntax based on JSON:

- use the `--jx` option

```
$ makeflow --jx build_archive.jx
```

- machine-friendly; JSON libraries available in virtually all languages (Python, R, C, Julia, Fortran, etc.)

- can be generated from Makeflow file

```
$ makeflow_viz -D json build_archive.makeflow
```

Syntax 2: jx

Alternative syntax based on JSON with extensions:

```
{
 "define": {
  "GZ": "module load gzip ; gzip -k ",
  "RANGE": [1,2,3]
 },

 "rules": [
  {
    "command": GZ + template("--{P} -S.{P}.gz archive.tar"),
    "inputs": ["archive.tar"],
    "outputs": ["archive.tar." + P + ".gz"]
  } for P in ["best", "fast"],

[…]
```

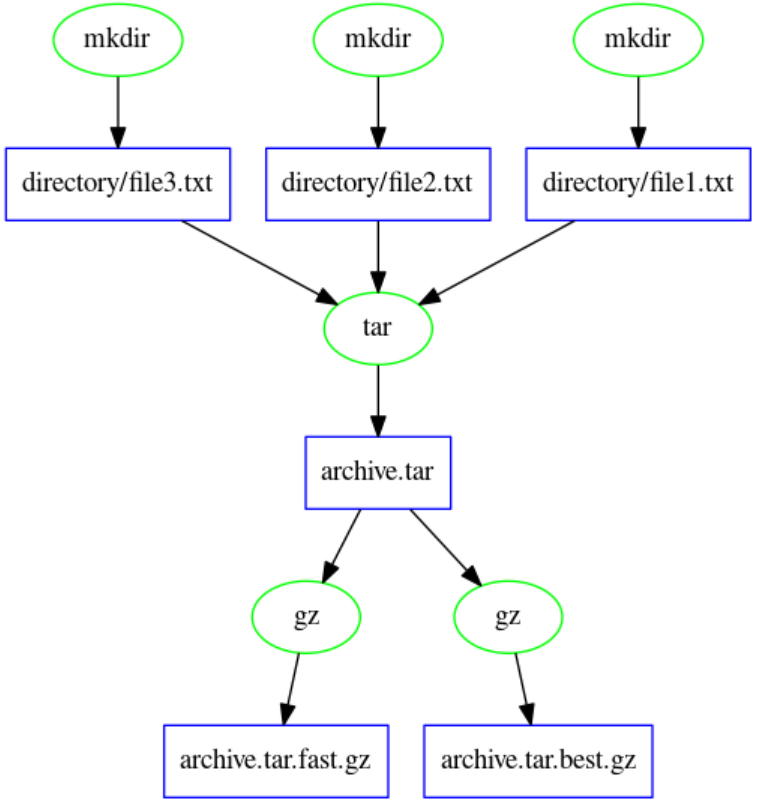
Syntax 2: jx

Alternative syntax based on JSON with extensions

```
{
 "define": {
  "GZ": "module load gzip ; gzip -k ",
  "RANGE": [1,2,3]
 },

 "rules": [
[…]
  {
   "command": template("mkdir -p directory; touch
                    directory/file{N}.txt"),
   "inputs": [],
   "outputs": ["directory/file" + N + ".txt"]
  } for N in RANGE,
```

Syntax 2: jx

Alternative syntax based on JSON with extensions



Nested workflows

Makeflow inside makeflow

The MAKEFLOW keyword designates another makeflow:

```
# Example of nested workflow

archive.tar.best.gz archive.tar.fast.gz:  build_archive.makeflow
    MAKEFLOW build_archive.makeflow

comparison.txt: archive.tar.best.gz archive.tar.fast.gz
    ls -ls *gz > comparison.txt
```

Makeflow inside makeflow

Nested makeflows do not submit additional jobs:

```
$ makeflow nested_workflow.makeflow
[…]
starting workflow
submitting job: makeflow -T local build_archive.makeflow -l
       build_archive.makeflow.0.makeflowlog
submitted job 267777
parsing build_archive.makeflow…
[…]
nothing left to do.
job 267777 completed
submitting job: ls -ls *gz > comparison.txt
submitted job 267808
job 267808 completed
nothing left to do.
```

Nested workflow

Makeflow inside makeflow

Cleaning works recursively:

```
$ makeflow  nested_workflow.makeflow --clean
[…]
parsing nested_workflow.makeflow…
cleaning sub-workflow build_archive.makeflow
makeflow -T local build_archive.makeflow -l
     build_archive.makeflow.0.makeflowlog –clean
[…]
cleaning filesystem…
[…]
nothing left to do.
done cleaning sub-workflow build_archive.makeflow
deleted comparison.txt
nothing left to do.
```

Nested workflow

Makeflow inside makeflow

Also for JX syntax:

```
[…]
"rules":
 [{
  "workflow":"build_archive.jx",
  "inputs": [ "build_archive.jx" ],
  "outputs": [ "archive.tar.best.gz", "archive.tar.fast.gz" ]
  },{
  "command":"ls -ls *gz > comparison.txt",
  "inputs": [ "archive.tar.best.gz", "archive.tar.fast.gz" ],
  "outputs": [ "comparison.txt" ]
  }]
[…]
```

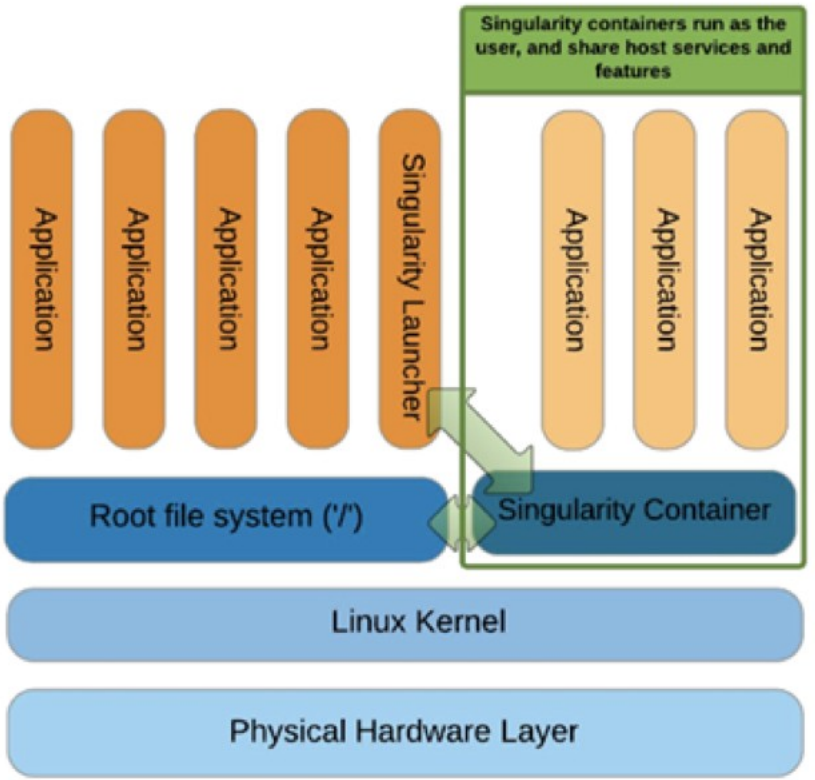
Singularity integration

Singularity in a slide

- Containers for HPC ("cluster-friendly docker")
- Easy way to pack and deploy software with all dependencies

Renamed/forked as "Apptainer"

Singularity in a slide



https://www.admin-magazine.com/HPC/Articles/Singularity-A-Container-for-HPC

Singularity; an OS inside a file

On a CentOS cluster:

```
$ head -1 /etc/*release
CentOS Linux release 7.9.2009 (Core)
```

With an Ubuntu image (lolcow.sif):

```
$ singularity exec lolcow.sif head -2 /etc/*release
NAME="Ubuntu"
VERSION="16.04.5 LTS (Xenial Xerus)"
```

The `head` command is run inside the image file

Makeflow + Singularity

Example makeflow:

```
comparison.txt: archive.tar.best.gz archive.tar.fast.gz
    head -3 /etc/*release ; ls -ls *gz > comparison.txt
```

By default, the `/etc/` directories are taken from the image, and the working directory (`/home`) from the cluster filesystem.

Makeflow + Singularity

Example run:

```
$ makeflow --singularity ~/lolcow.sif singularity_example.makeflow

parsing singularity_example.makeflow…
[…]
submitting job: ./singularity.wrapper.sh_A49uad
submitted job 186446
NAME="Ubuntu"
VERSION="16.04.5 LTS (Xenial Xerus)"
ID=ubuntu
job 186446 completed
deleted ./singularity.wrapper.sh_A49uad
nothing left to do.
```

Makeflow + Singularity

Example run:

```
$ ls *tar*
archive.tar.fast.gz archive.tar.best.gz archive.tar
```

The files were created out of the image from files outside the image using software in the image.

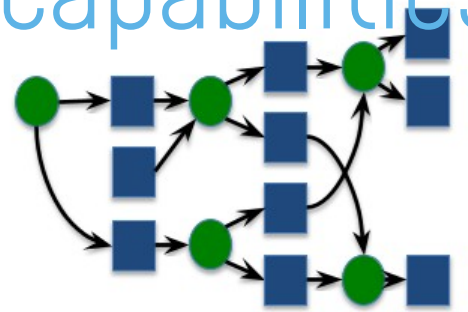
Makeflow + Singularity

Idea:

- install all needed software in the Singularity image
- submit all jobs “inside” the Singularity image “on” the compute node (possible on multiple clusters)
- Makeflow carries the image and the data along

Goal of this session:

Get an overview of the capabilities of **Makeflow** and of its usage.



http://ccl.cse.nd.edu/software/makeflow/

Topics covered:

- Installation
- Syntax(es)
- Run, monitoring, etc.
- Interaction with the scheduler

Wget, configure, make, make install
Or Easybuild

Topics covered:

- Installation
- Syntax(es)
- Run, monitoring, etc.
- Interaction with the scheduler

Makefile (simplified) or
JSON with extension
Variables

Topics covered:

- Installation
- Syntax(es)
- Run, monitoring, etc.
- Interaction with the scheduler

Make-like behavior (do not rebuild results that are already computed)
Blocking run
Automatic graphical representation
Singularity integration

Three modes of operation

-T slurm | -T local | -T wq

Multi-cluster with work queues

Nested workflows

Topics covered:

- Installation
- Syntax(es)
- Run, monitoring, etc.
- Interaction with the scheduler

Make(file) + (Work)flow = Makeflow