UNIVERSITY OF NOTRE DAME

Introduction to Makeflow and Work Queue

CCTools

The Cooperative Computing Lab

- We **collaborate with people** who have large scale computing problems in science, engineering, and other fields.
- We **operate computer systems** on the O(10,000) cores: clusters, clouds, grids.
- We **conduct computer science research** in the context of real people and problems.
- We **develop open source software** for large scale distributed computing.

CCTools

ccl.cse.nd.edu

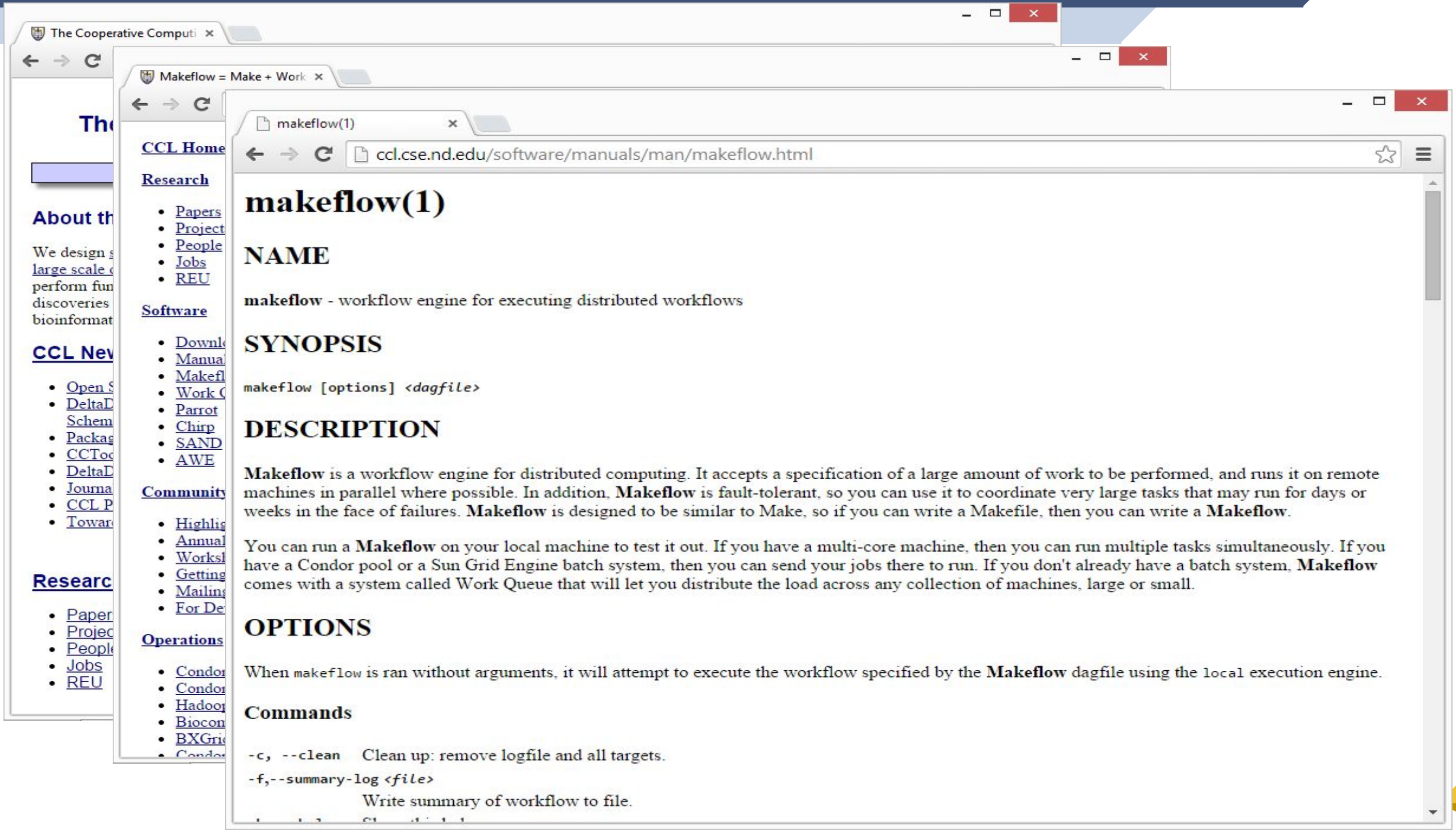
Our philosophy

- Harness all available resources: desktops, clusters, clouds, and grids.
- Make it easy to scale up from one desktop to national scale infrastructure.
- Provide familiar interfaces that make it easy to connect existing apps together.
- Allow portability across operating systems, storage systems, middleware…
- Make simple things easy, and complex things possible.
- **No special privileges required.**

A quick tour of CCTools

- Open source, GNU General Public License.
- Compiles in 1-2 minutes, installs in $HOME.
- Runs on Linux, Solaris, MacOS, FreeBSD, …
- Interoperates with many distributed computing systems.
    - Condor, SGE, Torque, Globus, iRODS, Hadoop…
- Components:
    - Makeflow – A portable workflow manager.
    - Work Queue – A lightweight distributed execution system.
    - Parrot – A personal user-level virtual file system.
    - Chirp – A user-level distributed filesystem.

Lots of documentation



The Cooperative Computi

The

About th

We design
large scale
perform fun
discoveries
bioinformat

CCL Nev

- Open S
- DeltaD Schem
- Packag
- CCToo
- DeltaD
- Journa
- CCL P
- Towar

Researc

- Paper
- Projec
- Peopl
- Jobs
- REU

Makeflow = Make + Work

CCL Home

Research

- Papers
- Project
- People
- Jobs
- REU

Software

- Downl
- Manua
- Makefl
- Work Q
- Parrot
- Chirp
- SAND
- AWE

Community

- Highlig
- Annual
- Worksh
- Getting
- Mailing
- For De

Operations

- Condor
- Condor
- Hadoop
- Biocom
- BXGri

makeflow(1)

ccl.cse.nd.edu/software/manuals/man/makeflow.html

makeflow(1)

NAME

makeflow - workflow engine for executing distributed workflows

SYNOPSIS

```
makeflow [options] <dagfile>
```

DESCRIPTION

Makeflow is a workflow engine for distributed computing. It accepts a specification of a large amount of work to be performed, and runs it on remote machines in parallel where possible. In addition, **Makeflow** is fault-tolerant, so you can use it to coordinate very large tasks that may run for days or weeks in the face of failures. **Makeflow** is designed to be similar to Make, so if you can write a Makefile, then you can write a **Makeflow**.

You can run a **Makeflow** on your local machine to test it out. If you have a multi-core machine, then you can run multiple tasks simultaneously. If you have a Condor pool or a Sun Grid Engine batch system, then you can send your jobs there to run. If you don't already have a batch system, **Makeflow** comes with a system called Work Queue that will let you distribute the load across any collection of machines, large or small.

OPTIONS

When `makeflow` is ran without arguments, it will attempt to execute the workflow specified by the **Makeflow** dagfile using the `local` execution engine.

Commands

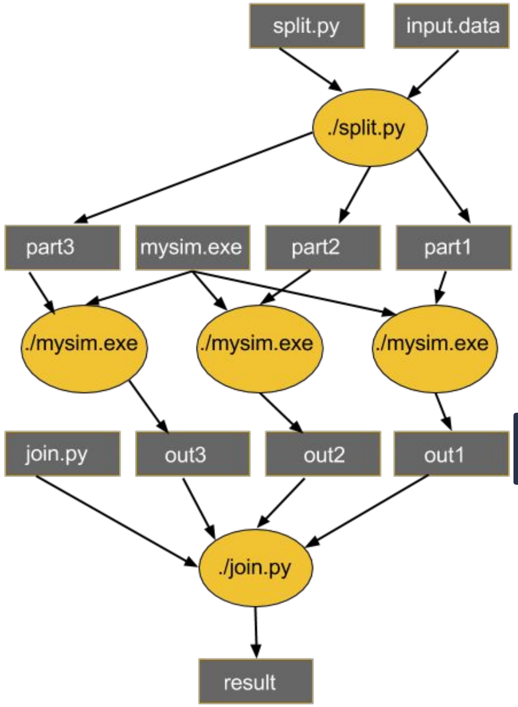
`-c, --clean` Clean up: remove logfile and all targets.

`-f,--summary-log <file>`
Write summary of workflow to file.

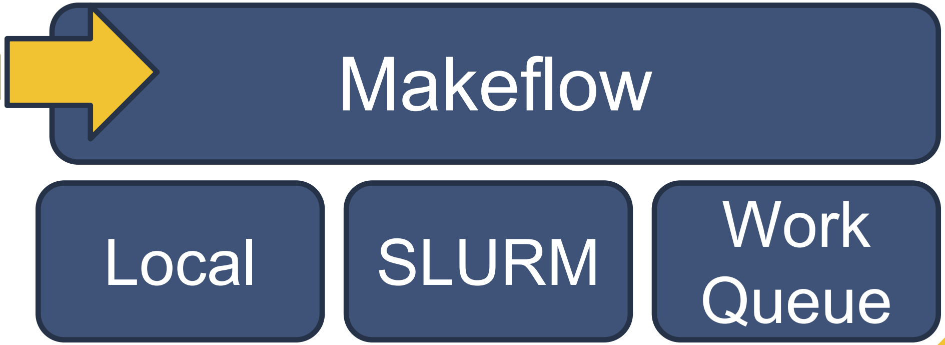
Makeflow

A portable workflow system

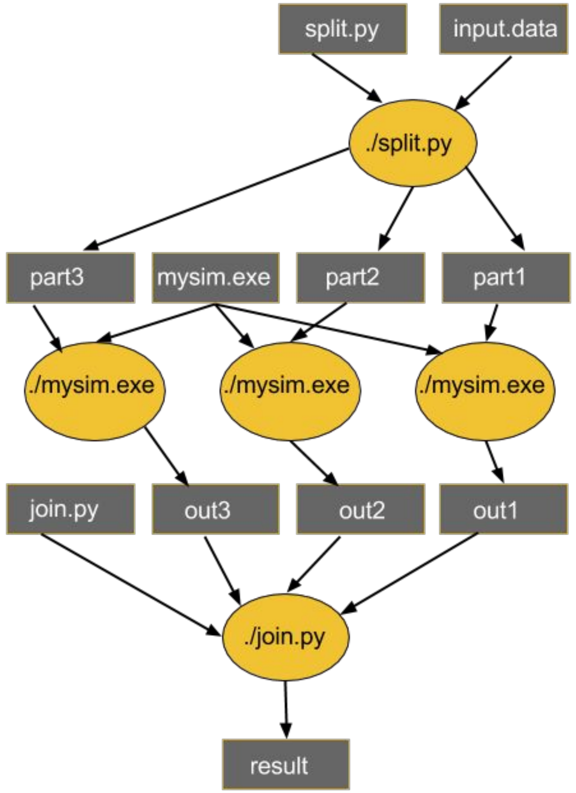
Makeflow (make + workflow)



- Provides portability across batch systems.
- Enable parallelism (but not too much!)
- Trickle out work to batch system
- Fault tolerance at multiple scales.
- Data and resource management.



Based off an old idea: Makefiles



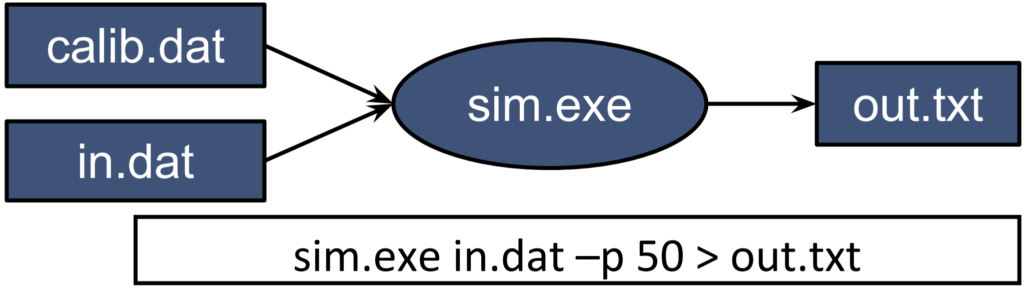
```
part1 part2 part3: input.data split.py
    ./split.py input.data

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 syntax

[output files] : [input files]

[command to run]

One Rule



```
out.txt : in.dat calib.dat sim.exe
    sim.exe in.data –p 50 > out.txt
```

Makeflow syntax

```
out.10 : in.dat calib.dat sim.exe
    sim.exe –p 10 in.data > out.10

out.20 : in.dat calib.dat sim.exe
    sim.exe –p 20 in.data > out.20

out.30 : in.dat calib.dat sim.exe
    sim.exe –p 30 in.data > out.30
```

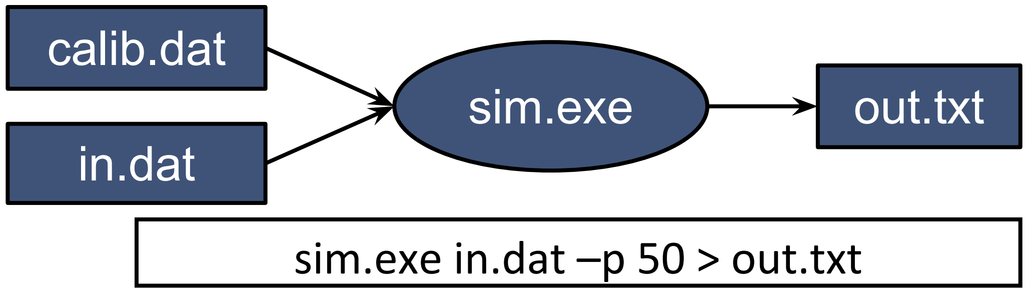
A Makefile is a really compact specification.

How about we try JSON to more verbosely define our tasks!

Makeflow JSON syntax

- Verbose and flexible
- Familiar structure
- Consists of four items:
  - "categories": Object<Category>
  - "default_category": String
  - "environment": Object<String>
  - "rules": Array<Rule>

Makeflow JSON syntax



sim.exe in.dat –p 50 > out.txt

```
{
        "outputs": ["out.txt"],
        "inputs": [ "in.dat", "calib.dat", "sim.exe"]
        "command": "sim.exe –p 50 in.data > out.txt",
}
```

One Rule

Makeflow JSON syntax

```
{
        "outputs": [{"out_10.txt"}],
        "inputs": [ {"in.dat"},  {"calib.dat"},
                                {"sim.exe"}]
        "command": "sim.exe –p 10 in.data > out_10.txt",
},
{
        "outputs": [{"path": "out_20.txt"}],
        "inputs": [ {"in.dat"},  {"calib.dat"},
                                {"sim.exe"}]
        "command": "sim.exe –p 20 in.data > out_20.txt",
},…
```

Makeflow JSON rule

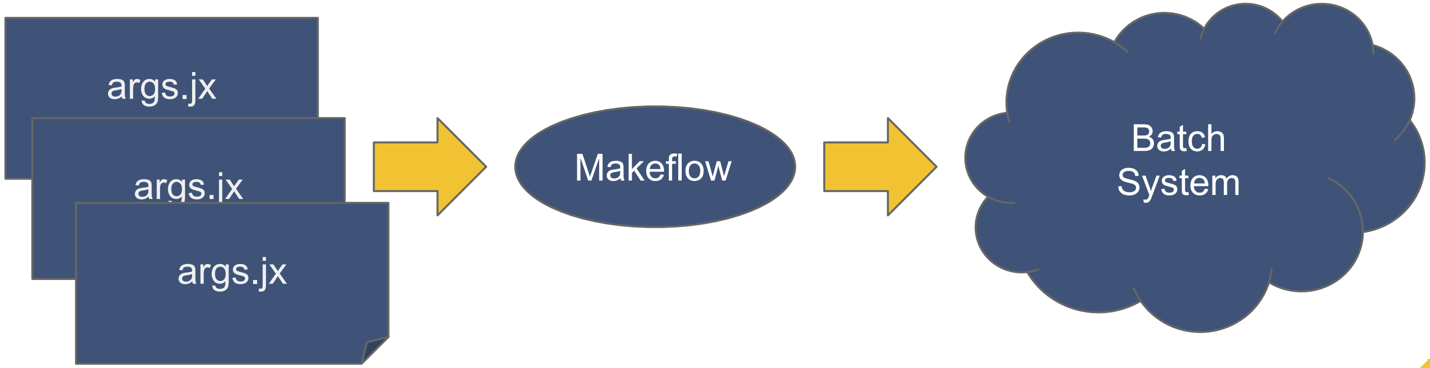
- "inputs": Array<File>
- "outputs": Array<File>
- "command": String
- "local_job": Boolean
- "category": String
- "resources": Resources
- "allocation": String
- "environment": Object<String>

JSON can be a bit too verbose sometimes.

How about we shorten it with JX!

Makeflow JX syntax

- Allows for more compact makeflows.
  - Provides functions for expanding tasks: range, variables, etc...
- Can be used as templates in conjunction with an arguments file.
- Useful for consistently structure data and different data.



Makeflow JX syntax

```
{
        "outputs": [{"out_10.txt"}],
        "inputs": [ {"in.dat"},  {"calib.dat"},
                                {"sim.exe"}]
        "command": "sim.exe –p 10 in.data > out_10.txt",
},...
```

We can represent this JSON with JX:

```
{
        "outputs": [{format("out_%d.txt", i)}],
        "inputs": [ {"in.dat"},  {"calib.dat"},
                                {"sim.exe"}]
        "command": format("sim.exe –p %d in.data > out_%d.txt", i),
} for i in range(10, 30, 10),
```

Tutorial time!

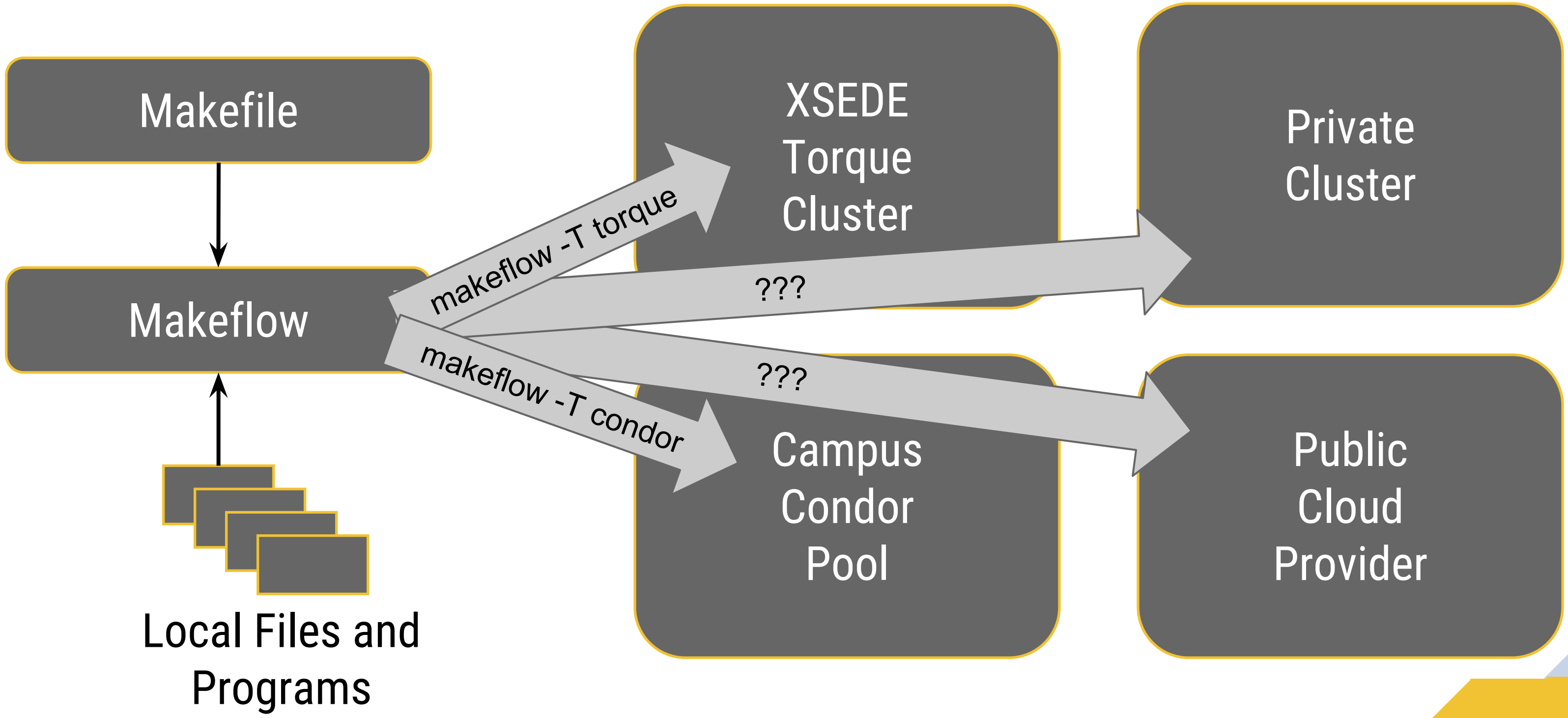
ccl.cse.nd.edu/software/tutorials/makeflow/makeflow-tutorial.php

Work to the end of part 1

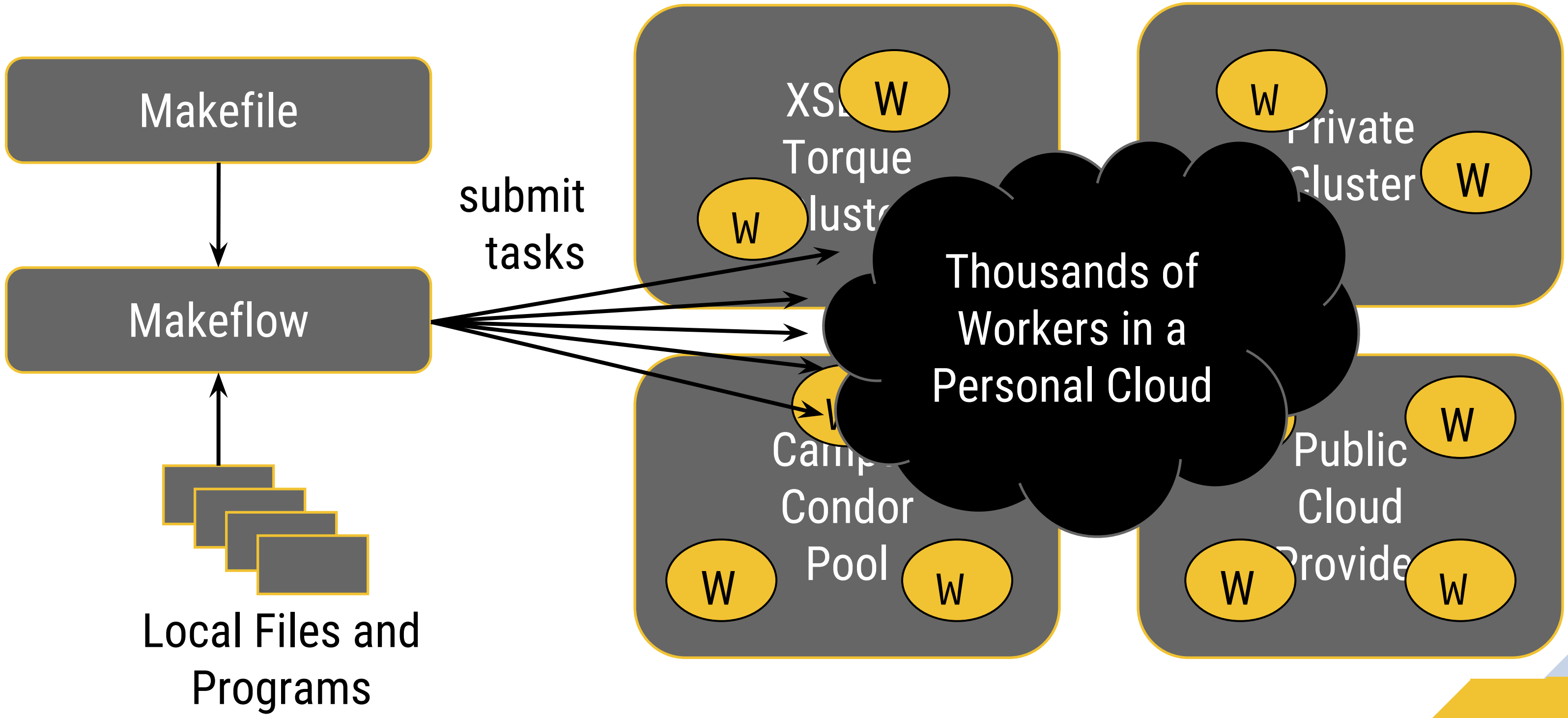
Makeflow + Work Queue

Harnessing concurrency with an execution engine

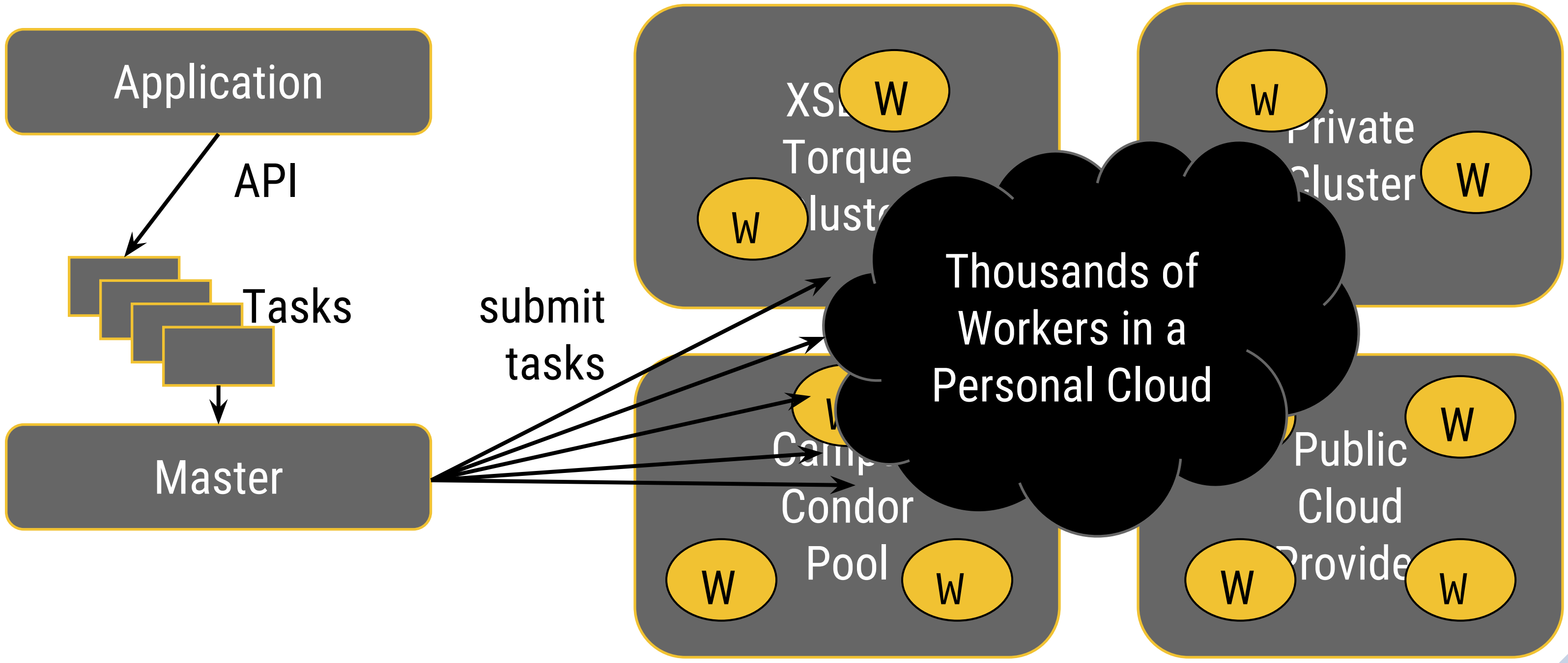
MAKEFLOW


Makefile
Makeflow
Local Files and Programs
makeflow -T torque
???
???
makeflow -T condor
XSEDE Torque Cluster
Private Cluster
Campus Condor Pool
Public Cloud Provider

MAKEFLOW + WORK QUEUE



WORK QUEUE

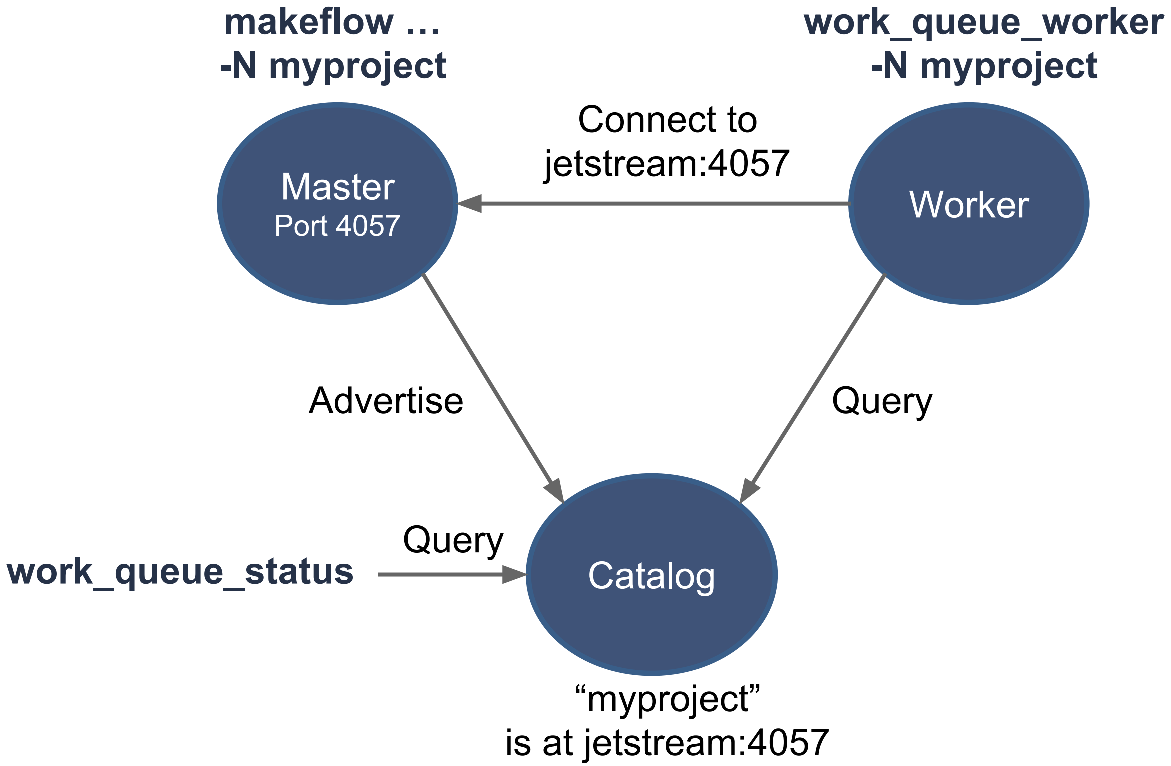

Application
API
Tasks
Master
submit tasks
XSEDE Torque Cluster
Private Cluster
Campus Condor Pool
Public Cloud Provider
W
Thousands of Workers in a Personal Cloud

Advantages of Work Queue

- Harness multiple resources simultaneously.
- Hold on to cluster nodes to execute multiple tasks rapidly.
  - (ms/task instead of min/task)
- Scale resources up and down as needed.
- Better management of data, with local caching for data intensive tasks.
- Matching of tasks to nodes with data.

Keeping track of port numbers is tedious.

Project names



Advantages of Work Queue

- MF +WQ is fault tolerant in many different ways:
  - If Makeflow crashes (or is killed) at any point, it will recover by reading the transaction log and continue where it left off.
  - Makeflow keeps statistics on both network and task performance, so that excessively bad workers are avoided.
  - If a worker crashes, the master detects failure and restarts the task elsewhere.
  - Workers can be added and removed at any time during workflow execution.
  - Multiple masters with the same project name can be added and removed while the workers remain.
  - If the worker sits idle for too long (default 15m) it will exit, so as not to hold resources idle.

Let’s try it out!

ccl.cse.nd.edu/software/tutorials/makeflow/makeflow-tutorial.php

Continue where you left off, and work to the end of the tutorial

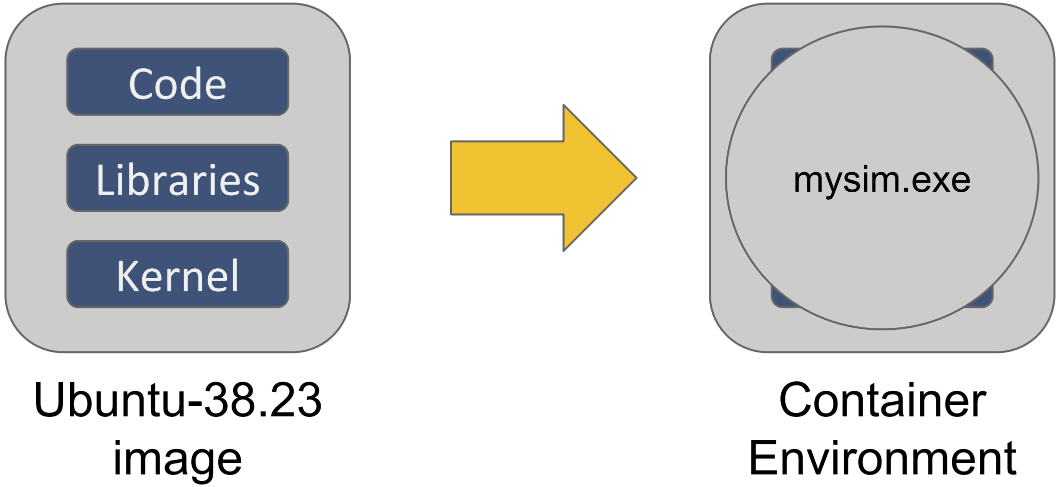
Container Integration

Providing consistent environments

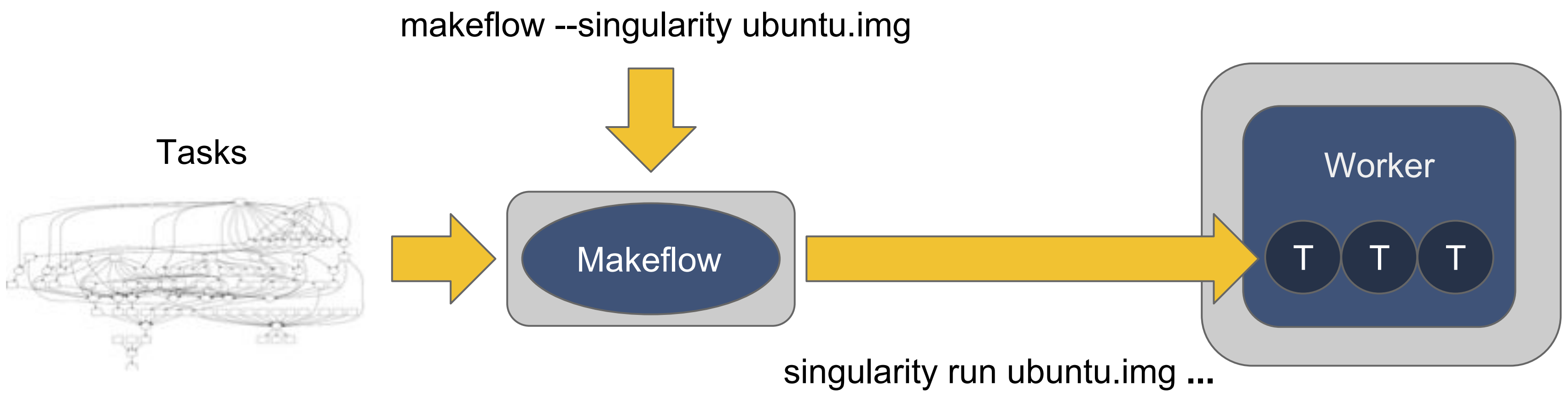
If you are not interested in utilizing containers for your workflows, the following slides will be supplementary material you may skip.

Containers create precise execution environments

```
singularity run ubuntu-38.23.img mysim.exe
```



Running a container for Makeflow tasks


makeflow --singularity ubuntu.img
Tasks
Makeflow
Worker
T
T
T
singularity run ubuntu.img ...

Last hands-on section!

ccl.cse.nd.edu/software/tutorials/makeflow/container-tutorial.php

Work through the container tutorial from start to finish.

Questions?

Please contact us!

Nate Kremer-Herman
nkremerh@nd.edu

ccl.cse.nd.edu