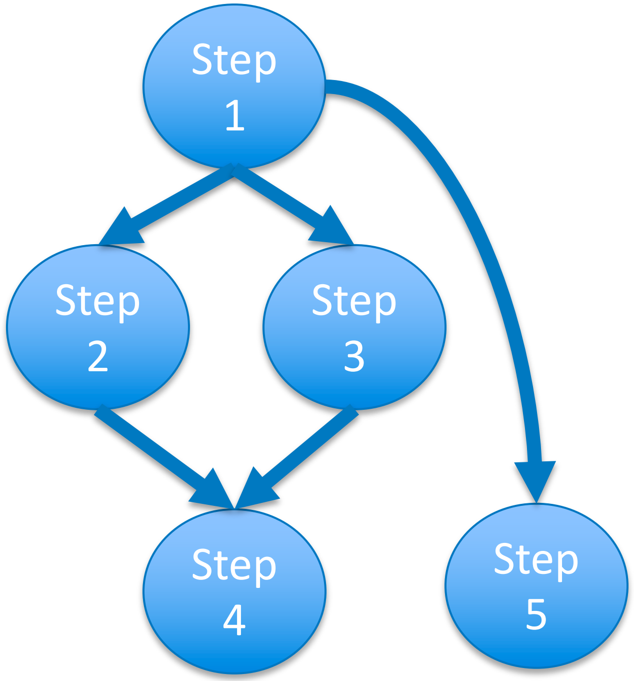
An Intro to Workflow Management Systems

Kevin Sayers
HPC User and Applications Support

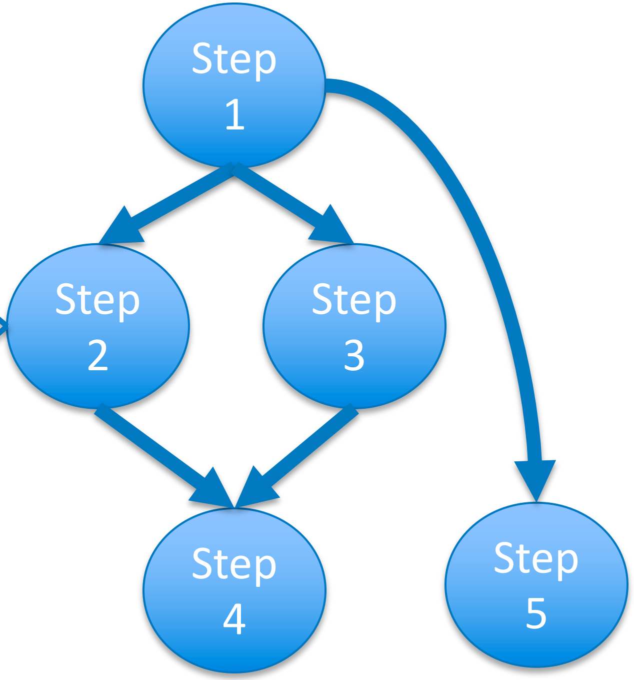
Workflow?

- Running multiple programs or analysis steps
- Directed Acyclic Graph (DAG)
- Specifies steps and their dependencies
- Defined in either general programming languages or workflow specific



Anatomy of a workflow step

```
Step 2:
        input:
                filename
                variables
        output:
                filename
        code:
                executable –run …..
```



WMS?

- **Workflow Management System**
    - Manage the the steps of a workflow and orchestration
    - Abstractions
        - Automatic parallelization
        - Handle the data and staging
        - Submission to HPC, cloud, local
        - Dependency management
    - Reproducibility of analysis
    - Provenance tracking

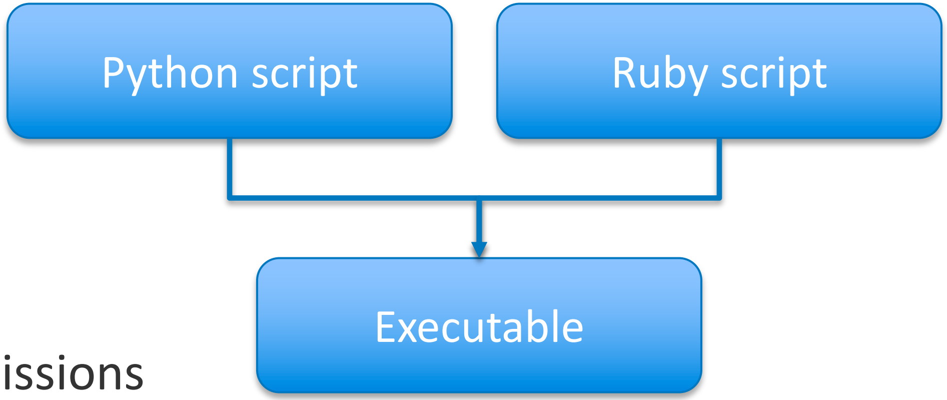
I can just use my language of choice

- Why don’t I just use Python/Bash/Java/My language of choice?
- Abstractions dramatically reduce coding burden
- Focus on domain research

Example

How much impact can a WMS have?

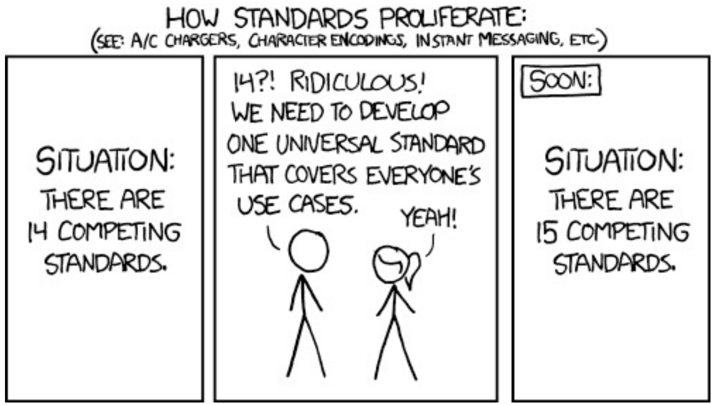
- Workflow composed of ≈ 1000 LOC
- Python scripts to run steps
- Steps were executed in unique Docker/Singularity containers
- WMS 135 LOC:
  - Container mounts
  - Slurm and AWS Batch submissions
  - Data movement
- Maintain just the scientific logic



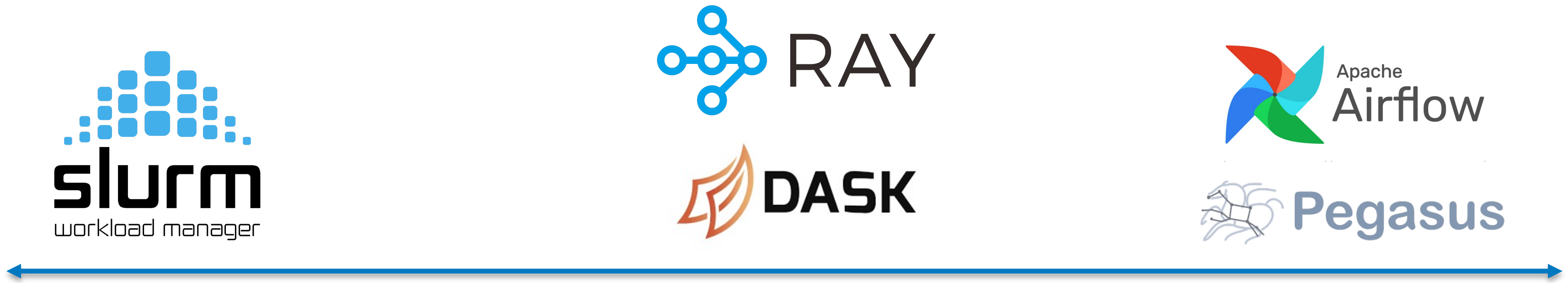
Current offerings

So you need to pick a WMS…

- 110 and growing on community list
  - https://github.com/pditommaso/awesome-pipeline
- Domain specific, data science, and general workflows
- Command line and Graphical
  - Graphical: Galaxy, Knime, etc.
- No one magic tool, but plenty of options


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

Spectrum

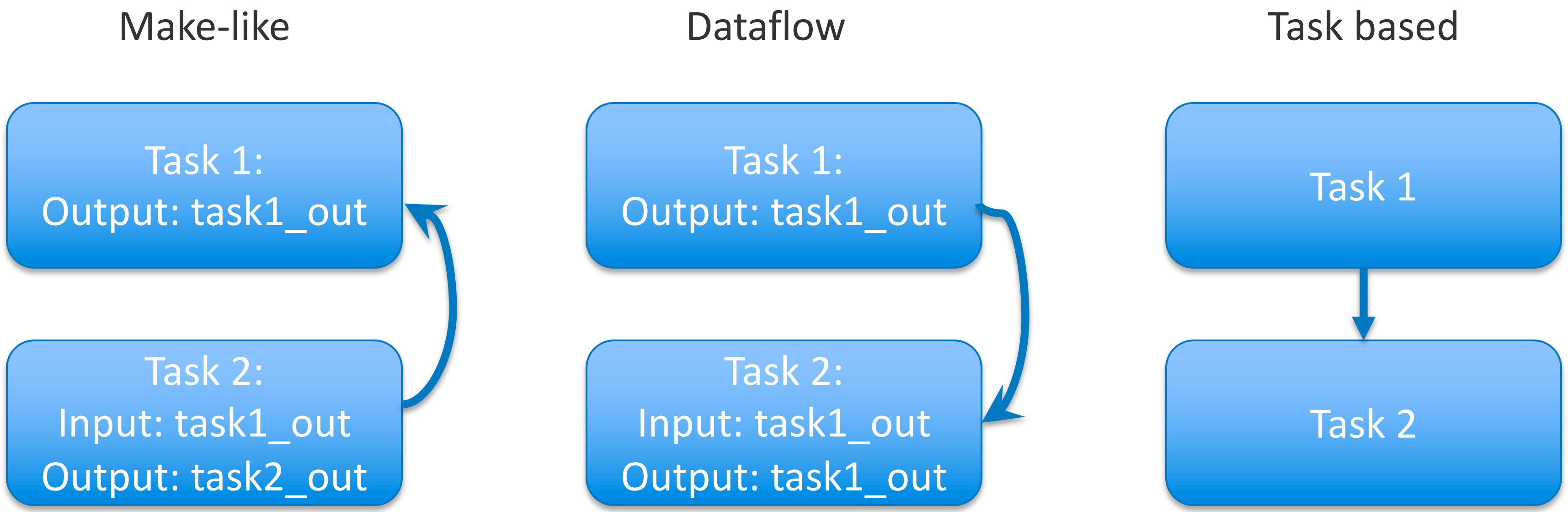


Task dependencies

Extensive Abstraction

Specific considerations for picking a WMS

Styles of WMS



WMS definitions

```
rule getLineCounts:
      input: "input.csv"
      output: "lines"
      shell: "wc -l {input} > {output}"
```

```
getLineCounts = Job("wc")
getLineCounts.addArguments("-l", infile)
getLineCounts.setStdout(outfile)

getLineCounts.uses(infile, link=Link.INPUT,
transfer=True, register=True)
getLineCounts.uses(outfile,
link=Link.OUTPUT, transfer=True,
register=True)
dax.addJob(getLineCounts)
```

```
class pythonScript(luigi.Task):
      task_complete = False
      myinput = luigi.Parameter()

      def requires(self):
           yield getLineCounts(self.myinput)

      def run(self):
           infile = self.input()[0].path
           with open(infile) as f:
           print (f.read())
           self.task_complete = True
      def complete(self):
           return self.task_complete
```

WMS definitions

```
rule getLineCounts:
     input: "input.csv"
     output: "lines"
     shell: "wc -l {input} > {output}"
```

```
getLineCounts = Job("wc")
getLineCounts.addArguments("-l", infile)
getLineCounts.setStdout(outfile)

getLineCounts.uses(infile, link=Link.INPUT,
transfer=True, register=True)
getLineCounts.uses(outfile,
link=Link.OUTPUT, transfer=True,
register=True)
dax.addJob(getLineCounts)
```

```
class pythonScript(luigi.Task):
     task_complete = False
     myinput = luigi.Parameter()

     def requires(self):
          yield getLineCounts(self.myinput)

     def run(self):
          infile = self.input()[0].path
          with open(infile) as f:
          print (f.read())
          self.task_complete = True
     def complete(self):
          return self.task_complete
```

WMS definitions

```
rule getLineCounts:
     input: "input.csv"
     output: "lines"
     shell: "wc -l {input} > {output}"
```

```
getLineCounts = Job("wc")
getLineCounts.addArguments("-l", infile)
getLineCounts.setStdout(outfile)

getLineCounts.uses(infile, link=Link.INPUT,
transfer=True, register=True)
getLineCounts.uses(outfile,
link=Link.OUTPUT, transfer=True,
register=True)
dax.addJob(getLineCounts)
```

```
class pythonScript(luigi.Task):
     task_complete = False
     myinput = luigi.Parameter()

     def requires(self):
          yield getLineCounts(self.myinput)

     def run(self):
          infile = self.input()[0].path
          with open(infile) as f:
          print (f.read())
          self.task_complete = True
     def complete(self):
          return self.task_complete
```

WMS definitions

```
rule getLineCounts:
     input: "input.csv"
     output: "lines"
     shell: "wc -l {input} > {output}"
```

```
getLineCounts = Job("wc")
getLineCounts.addArguments("-l", infile)
getLineCounts.setStdout(outfile)

getLineCounts.uses(infile, link=Link.INPUT,
transfer=True, register=True)
getLineCounts.uses(outfile,
link=Link.OUTPUT, transfer=True,
register=True)
dax.addJob(getLineCounts)
```

```
class pythonScript(luigi.Task):
     task_complete = False
     myinput = luigi.Parameter()

     def requires(self):
          yield getLineCounts(self.myinput)

     def run(self):
          infile = self.input()[0].path
          with open(infile) as f:
          print (f.read())
          self.task_complete = True
     def complete(self):
          return self.task_complete
```

Implicit vs explicit flow

```
## Implicit
rule all:
    input: "/tmp/smout"

rule getLineCounts:
    input: "input.csv"
    output: "lines"
    shell: "wc -l {input} > {output}”

rule pythonScript:
    input: "lines"
    output: "temp"
    run:
        with open(input[0]) as f:
            print (f.read())
            ofile = open("temp", "w")
            ofile.close()
```

```
## Explicit
t2 = PythonOperator(
    task_id='pythonScript',
    python_callable=printFile,
    dag=dag
)

t3 = DockerOperator(
    task_id='dc',
    image='ubuntu',
    command="echo hello",
    dag=dag
)

t2 >> t3
```

WMS domains

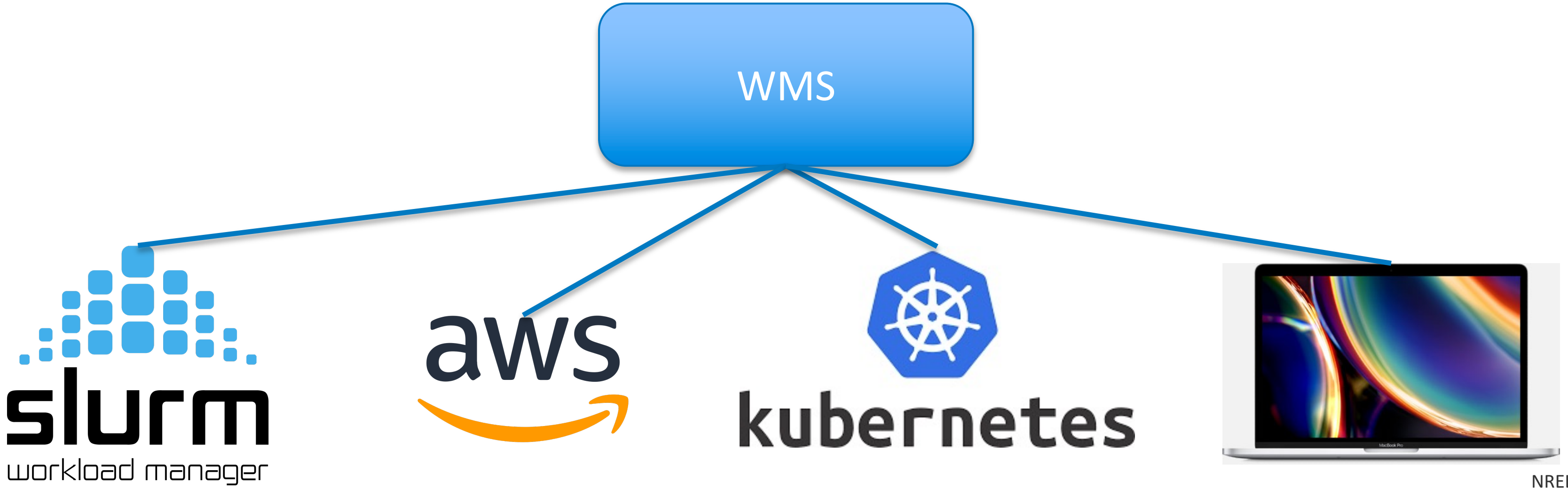
Domain and general HPC WMS

- Batch processing
  - HPC schedulers and cloud
  - Examples: Nextflow, Pegasus, Fireworks
- Data science:
  - Emphasis on retrieving data (SQL and data APIs)
  - Scheduling: Run these steps weekly
  - Cloud or Kubernetes
  - Examples: Airflow, Prefect, Luigi
- Misc
  - Kubeflow: Kubernetes focused WMS
  - Dask: Python quasi WMS

Portability

Run where you need to

- Enabling execution on HPC, cloud, and locally
- Ideally executor agnostic



Backend support

WMS	Local	HPC	Cloud	Kubernetes
Airflow	Yes		AWS Batch operator	Yes
FireWorks	Yes	Slurm, PBS/Torque, SGE, IBM Loadlever	No	No
Luigi	Yes	LSF,SGE	AWS Batch	Yes
Nextflow	Yes	Slurm, LSF, SGE, PBS/Torque, HTCondor	AWS Batch, Google Pipelines	Yes
Pegasus	Yes	Slurm, SGE, PBS, others support by glite	AWS Batch	Yes (OLCF)
Snakemake	Yes	Slurm, PBS/Torque, SGE, HTCondor	Google Pipelines	Yes

Portability

WMS	Local	HPC	Cloud	Kubernetes
Airflow	Yes		AWS Batch operator	Yes
FireWorks	Yes	Slurm, PBS/Torque, SGE, IBM Loadlever	No	No
Luigi	Yes	LSF,SGE	AWS Batch	Yes
Nextflow	Yes	Slurm, LSF, SGE, PBS/Torque, HTCondor	AWS Batch, Azure, Google Pipelines	Yes
Pegasus	Yes	Slurm, SGE, PBS, others support by glite	AWS Batch	Yes (OLCF)
Snakemake	Yes	Slurm, PBS/Torque, SGE, HTCondor	Google Pipelines	Yes

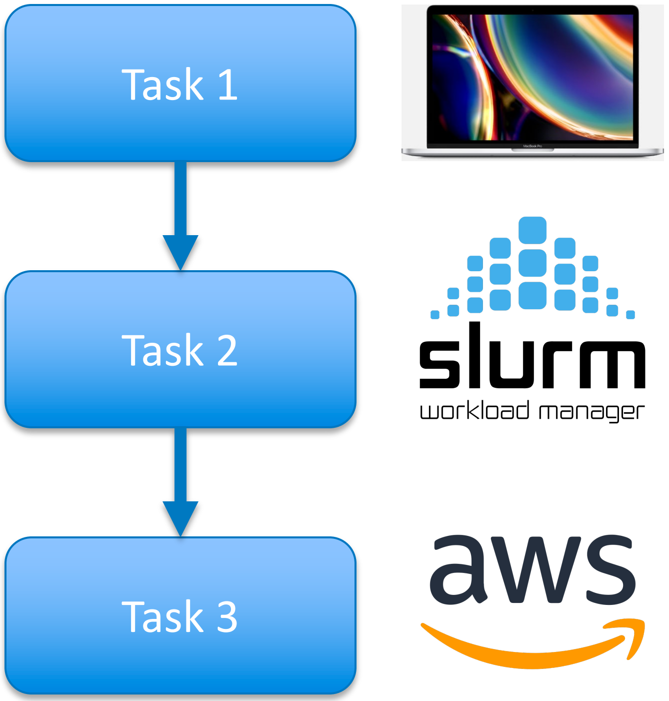
Portability

WMS	Local	HPC	Cloud	Kubernetes
Airflow	Yes		AWS Batch operator	Yes
FireWorks	Yes	Slurm, PBS/Torque, SGE, IBM Loadlever	No	No
Luigi	Yes	LSF,SGE	AWS Batch	Yes
Nextflow	Yes	Slurm, LSF, SGE, PBS/Torque, HTCondor	AWS Batch, Google Pipelines	Yes
Pegasus	Yes	Slurm, SGE, PBS, others support by glite	AWS Batch	Yes (OLCF)
Snakemake	Yes	Slurm, PBS/Torque, SGE, HTCondor	Google Pipelines	Yes

Hybrid workflow

Run steps where needed

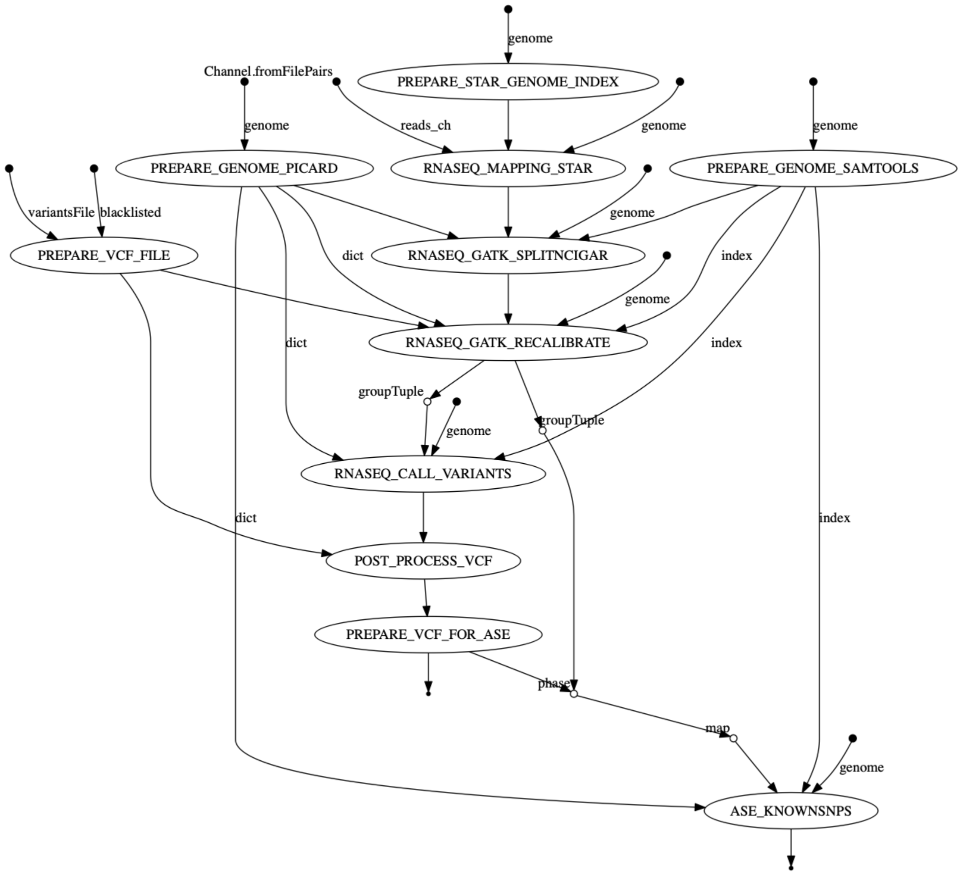
- Nextflow
  - Each process can have a unique executor
- Luigi
  - Tasks for (LSF, SGE, AWS Batch)
- Airflow
  - Operators for supported services (e.g. AWS Batch, Kubernetes)
  - Plugins for services (AWS Athena)



Dependency management

Portability of tools

- Ways to manage the environments for each task
- Conflicting versions
- Containers are widely supported
  - Docker for cloud
  - Singularity for HPC



Dependency management

WMS	Container support	Conda support	Environment modules
Airflow	Docker, Singularity	No	No
FireWorks	No	No	No
Luigi	Docker	No	No
Nextflow	Docker, Singularity, udocker, Shifter, Podman	Yes	Yes
Pegasus	Docker, Singularity, Shifter	No	No
Snakemake	Singularity	Yes	Yes

Portable

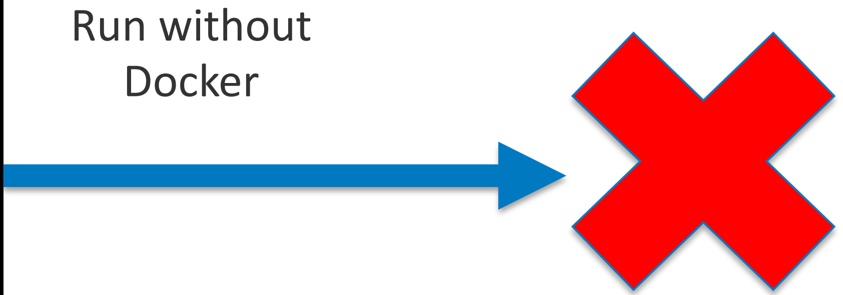
```
process containerHello{
     """
     #!/usr/bin/env python
     print ("hello")
     """
}
```

```
nextflow run wf –profile local
nextflow run wf –profile docker
nextflow run wf –profile conda
```

```
process.container = 'python:3.6.12'
profiles{
     local {
          process.executor = 'local'
     }
     docker {
          docker.enabled = true
     }
     singularity {
          singularity.enabled = true
     }
     shifter {
          shifter.enabled = true
     }
     conda {
          process.conda = 'python=3.6'
     }
}
```

Not portable

```
t3 = DockerOperator(
    task_id='dc',
    image='ubuntu',
    command="echo hello",
    dag=dag
)
```



Example

```
process run_sampling{
      publishDir
"${params.project}/characteristics",
            saveAs: { filename -> 'builds.csv' },
            mode: 'copy'

     input:
     file resources from
     Channel.fromPath("${params.open}/resources/")
     file projectdir from
     Channel.fromPath("${params.project}")

     output:
     file 'resources/out.csv' into sampled
     """
     python ${resources}/run_sa.py \
     -p ${projectdir} \
     -n ${params.datapoints} \
     -o out.csv
      """
}
```

```
profiles {
  awsbatch{
    aws.batch.cliPath = '/home/ec2-user/miniconda/bin/aws'
    workDir = 's3://nfbuildtest/work/'
    params {
      project = 's3://nfbuildtest/data'
      tool = 's3://nfbuildtest/'
    }
    process {
      executor = 'awsbatch'
      queue = 'nfbuildq'
      container = 'sayerskt/run:latest'
    }
  }
  eagle{
    workDir = '/scratch/ksayers/work'
    params {
      project = '/scratch/ksayers/project_ks'
      data = '/scratch/ksayers/data'
    }
    singularity{
      enabled = true
      autoMounts = true
    }
    process{
      executor = 'slurm'
      clusterOptions = '--account=hpcapps'
      time = '3h'
      container = 'sayerskt/run:latest'
    }
  }
}
```

Example

```
process run_sampling{
      publishDir
"${params.project}/characteristics",
            saveAs: { filename -> 'builds.csv' },
            mode: 'copy’

     input:
     file resources from
     Channel.fromPath("${params.open}/resources/")
     file projectdir from
     Channel.fromPath("${params.project}")

     output:
     file 'resources/out.csv' into sampled
     """
     python ${resources}/run_sa.py \
     -p ${projectdir} \
     -n ${params.datapoints} \
     -o out.csv
      """
}
```

```
profiles {
   awsbatch{
      aws.batch.cliPath = '/home/ec2-user/miniconda/bin/aws'
      workDir = 's3://nfbuildtest/work/'
      params {
         project = 's3://nfbuildtest/data'
         tool = 's3://nfbuildtest/'
      }
      process {
         executor = 'awsbatch'
         queue = 'nfbuildq'
         container = 'sayerskt/run:latest'
      }
   }
   eagle{
      workDir = '/scratch/ksayers/work'
      params {
         project = '/scratch/ksayers/project_ks'
         data = '/scratch/ksayers/data'
      }
      singularity{
         enabled = true
         autoMounts = true
      }
      process{
         executor = 'slurm'
         clusterOptions = '--account=hpcapps'
         time = '3h’
         container = 'sayerskt/run:latest'
      }
   }
}
```

Fault tolerance

When things don’t go well

- An ability to resume workflows
- Quick prototyping, skip computationally intensive steps
- Snakemake:
  - If output is present step is automatically skipped
- Nextflow:
  - Process block and inputs are hashed can skip if hashes match
- Luigi:
  - If outputs exist skips, recommend using a working dir
- Pegasus:
  - Rescue workflow for only steps still needing to be processed

Version control

Integration with GitHub, GitLab

- Pull and run workflow directly from GitHub
  - `nextflow run http://github.com/nextflow-io/hello`
  - `nextflow run nextflow-io/hello`
- Run a specific branch
  - `nextflow run nextflow-io/hello -r mybranch`
- Run a specific version
  - `nextflow run nextflow-io/hello -r v1.1`
- Very useful for reproducibility!

Modularity

Composing workflows

- Each task can be standalone sub-workflow of one or more steps
- Enables reproducibility/robustness through reuse of tasks
- Testing for each step
- Snakemake
  - **Include**: can include one or more steps
- Nextflow
  - DSL2: each process can be a standalone file, workflow defines inputs/outputs
- Python imports

Dask

- Commonly known for Dask Dataframes (Pandas) and Dask Arrays (numpy)
- Quasi WMS, task dependencies
- Entire ecosystem of distributed tooling
- Dask Distributed implements a scheduler and workers
- Dask-jobqueue (HPC schedulers), Dask Cloud Provider (AWS, GCP, Azure)

Dask

```
cluster = SLURMCluster( cores=18,
          memory='24GB',
          queue='short',
          project='hpcapps',
          walltime='00:30:00',
          interface='ib0',
          processes=18, )
```

```
cluster = FargateCluster(image="image",
        task_role_arn="arn:aws:iam::
          execution_role_arn="arn:aws:iam",
          n_workers=1,
          scheduler_cpu=256,
          scheduler_mem=512,
           worker_cpu=256,
          worker_mem=512,
          scheduler_timeout="15 minutes", )
```

```
cluster = LocalCluster(
          n_workers=2)
```

```
client = Client(cluster)
```

Converging

In general, HPC simulations and big data analytics are on a converging path. As the scientific community is preparing for the next-generation extreme-scale computing, big data analytics is becoming an essential part of the scientific process for insights and discoveries [25]. Effectively integrating big data analytics into HPC simulations still remains a daunting challenge, and it requires new workflow technologies to reduce user burden and improve efficiency.

- *da Silva, Rafael Ferreira, et al. "A characterization of workflow management systems for extreme-scale applications." Future Generation Computer Systems 75 (2017): 228-238.*

Engagement

- Short survey
  - https://forms.gle/9KMu4ccZwHaRQdSH8
- HPC User and Applications Support
  - Reach out we are interested in hearing your workflow needs
- Evolving documentation
  - https://nrel.github.io/HPC/

Conclusions

- A wide variety of WMS systems and implementations
- Consider what your needs are
- Selecting a WMS can amplify your research
  - Reproducibility
  - Shareable
- Continuously evolving field, cloud native workflows currently growing rapidly (Argo/Pachyderm/Kubeflow)
- Commonly emerging trends
  - Containerization
  - HPC + cloud
- Take a look at existing WMS before making your own
- Workshops for various WMS in the near future

Resources

Further reading

- https://github.com/pditommaso/awesome-pipeline
- https://docs.nersc.gov/jobs/workflow-tools/
- da Silva, Rafael Ferreira, et al. "A characterization of workflow management systems for extreme-scale applications." *Future Generation Computer Systems* 75 (2017): 228-238.
- Atkinson, Malcolm, et al. "Scientific workflows: Past, present and future." (2017): 216-227.
- Deelman, Ewa, et al. "The future of scientific workflows." *The International Journal of High Performance Computing Applications* 32.1 (2018): 159-175.
- Mitchell, Ryan, et al. "Exploration of Workflow Management Systems Emerging Features from Users Perspectives." *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 2019