Day 11: Workflows with DAGMan

Suggested reading: Condor 7.7 Manual:

http://www.cs.wisc.edu/condor/manual/v7.7/

Section 2.10: DAGMan Applications
Chapter 9: condor_submit_dag

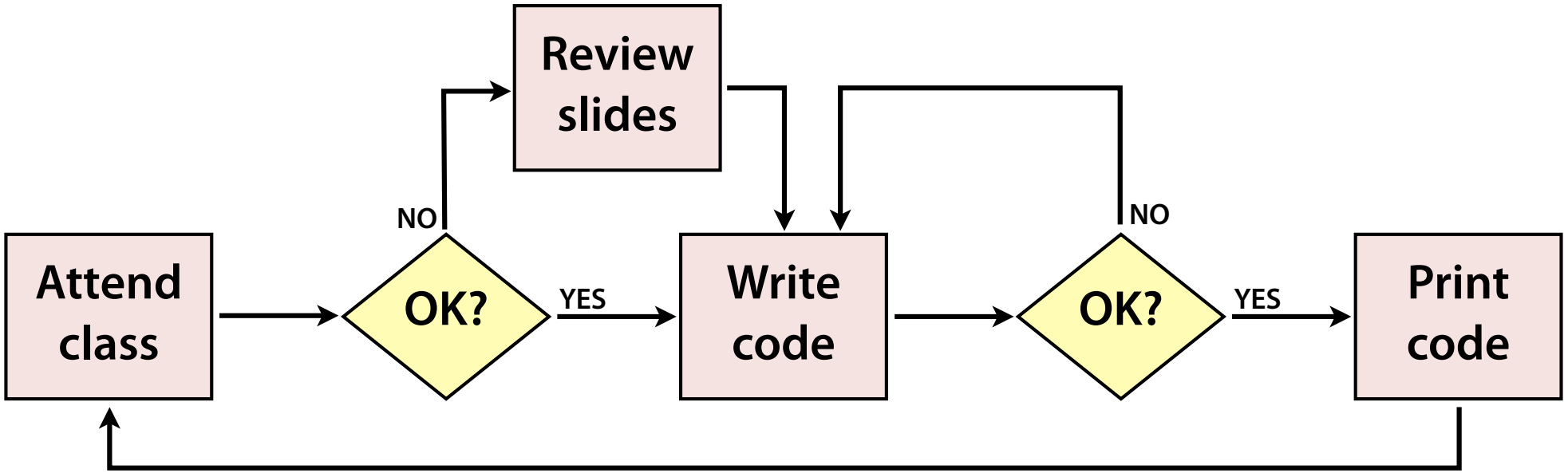
Turn In Homework

Homework Review

Workflows

Introduction to Workflow

- Series of related steps to complete a complex task

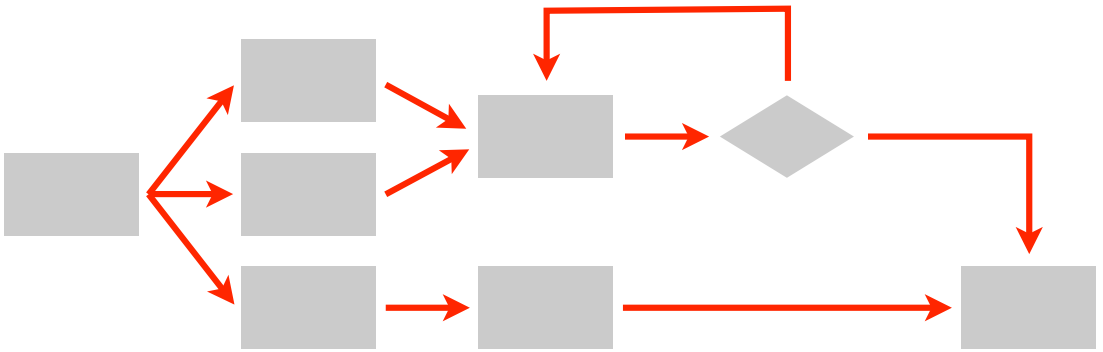


- Organize, manage, and make a process reliable
- Important in science, where repeatability is key

Workflow Components

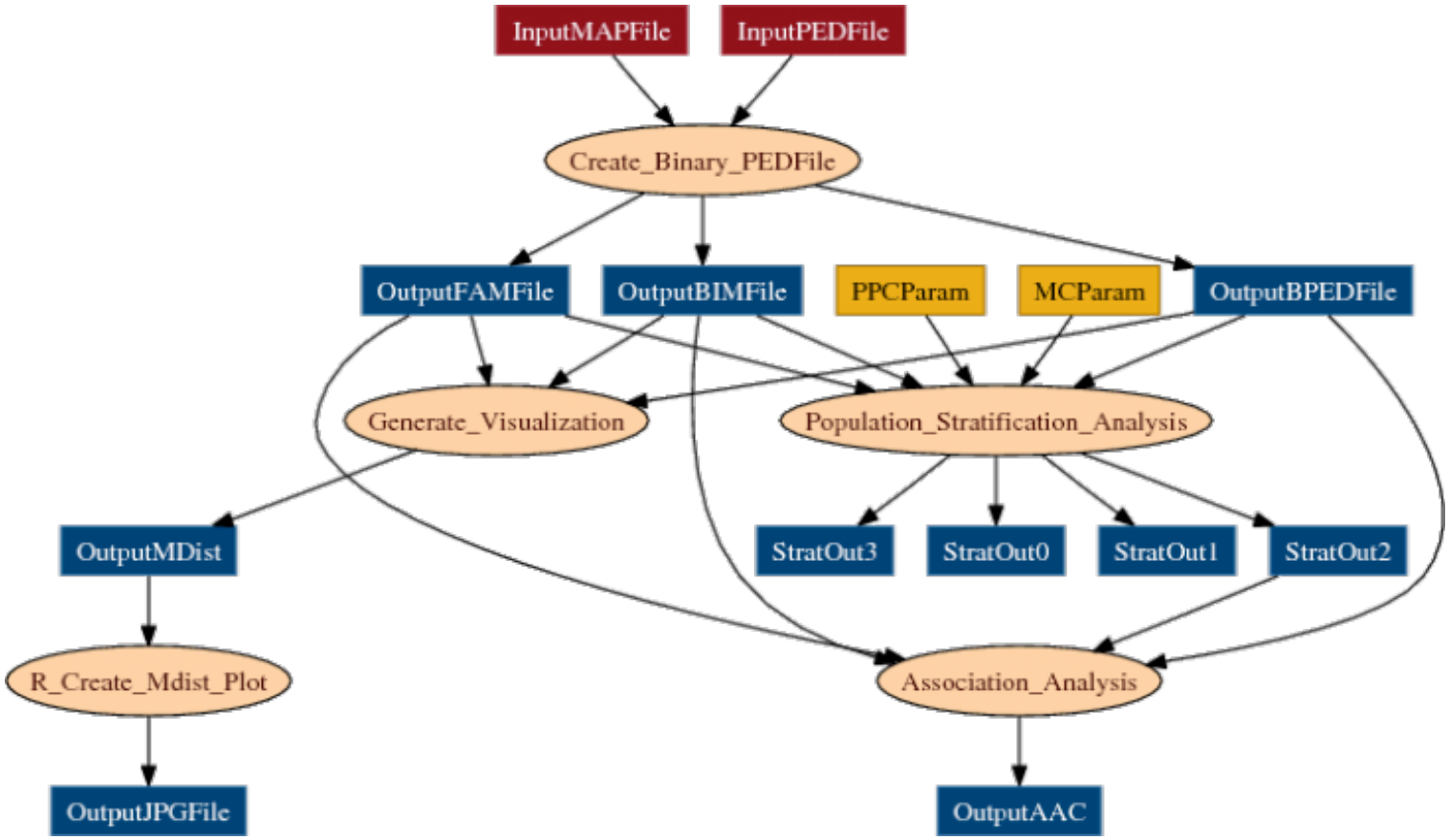
- Workflows are essentially algorithmic!
- **Steps**
  - Prerequisites and inputs
  - Process (black box / white box)
  - Outputs

- **Connections**
  - Sequence
  - Branching
  - Parallelism



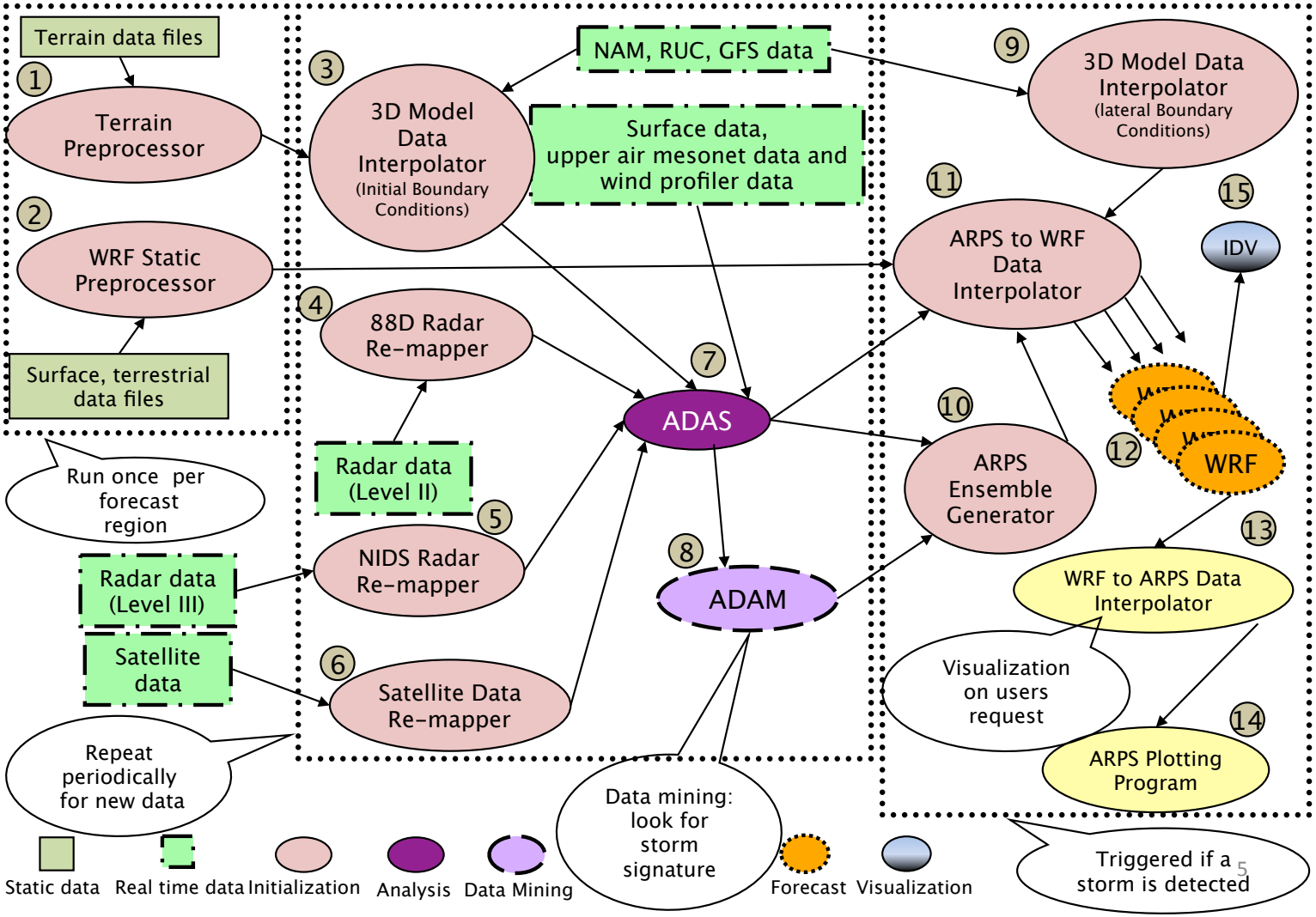
- ***Metadata:*** Resources, owners, timing, etc.

Workflow Example I



Bioinformatics @ Yale: C. Mason, S. Sanders, M. State

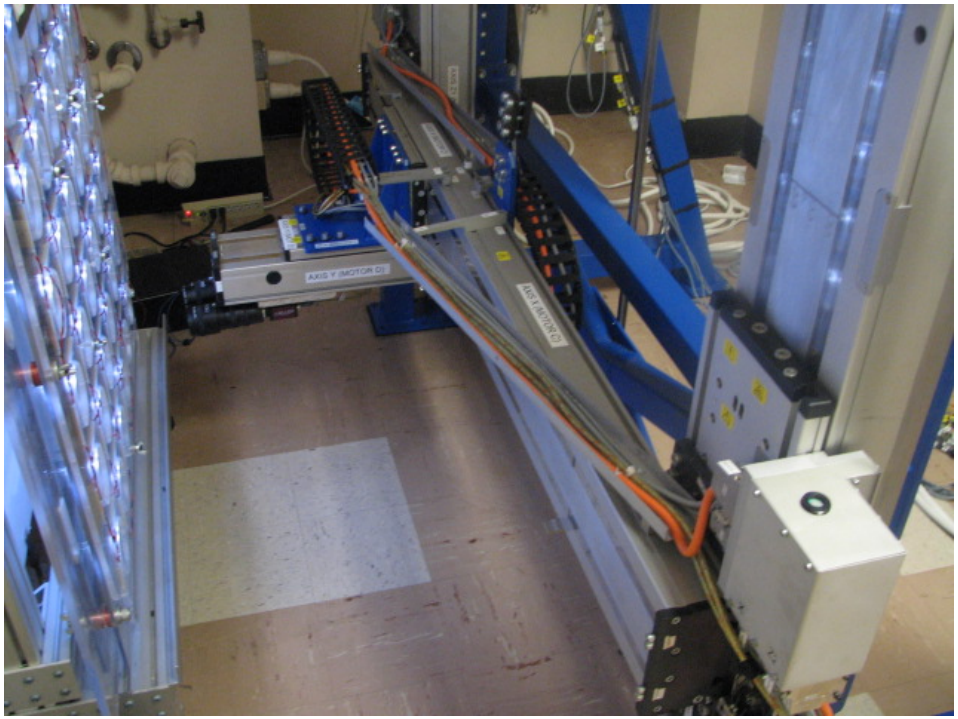
Workflow Example II



LEAD Weather Forecasting

Automated Workflows

- Ideally, we want to automate workflows
  - Minimize wait times and (certain kinds of) errors
  - Allow humans to concentrate on design and results
- Broad objectives:
  - Capture whole workflow
  - Define steps clearly
  - Identify easy automation
    - Copying files
    - Changing data formats
    - Running jobs!
  - Balance costs vs. savings!



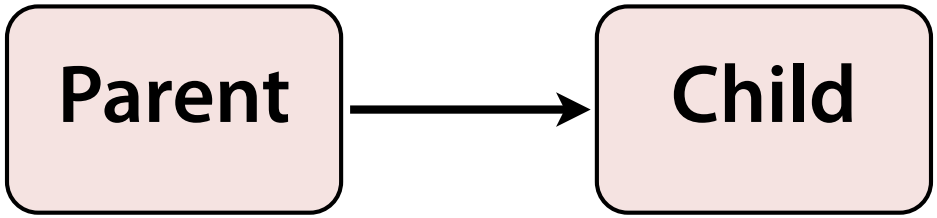
Workflows in CHTC

Directed Acyclic Graphs (DAGs)

- Abstract, formal definition of allowable workflows
- Terminology
  - Step (typically, a job) = ***Node***
  - Connection is *directed:* **Parent → Child**

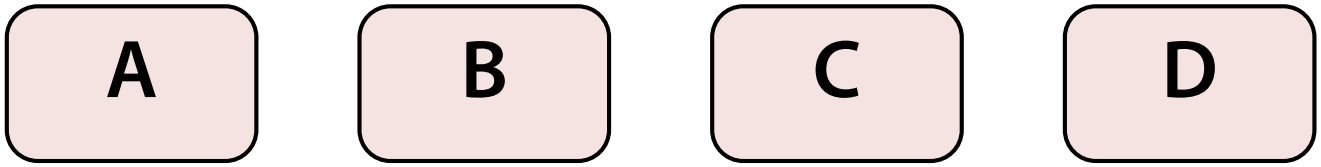
"… must succeed before running …"

  - No loops (or cycles, hence *acyclic*)
  - Each node may have 0–*n* children
  - Each node may have 0–*n* parents

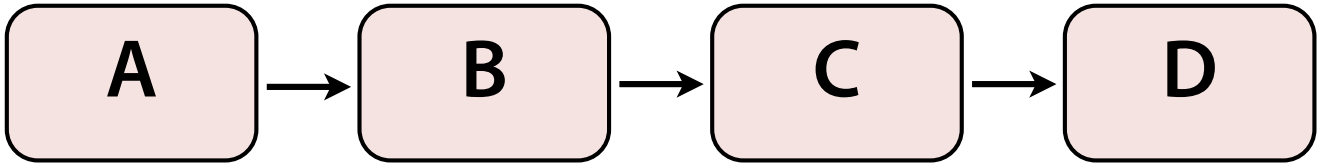


Example DAG Shapes

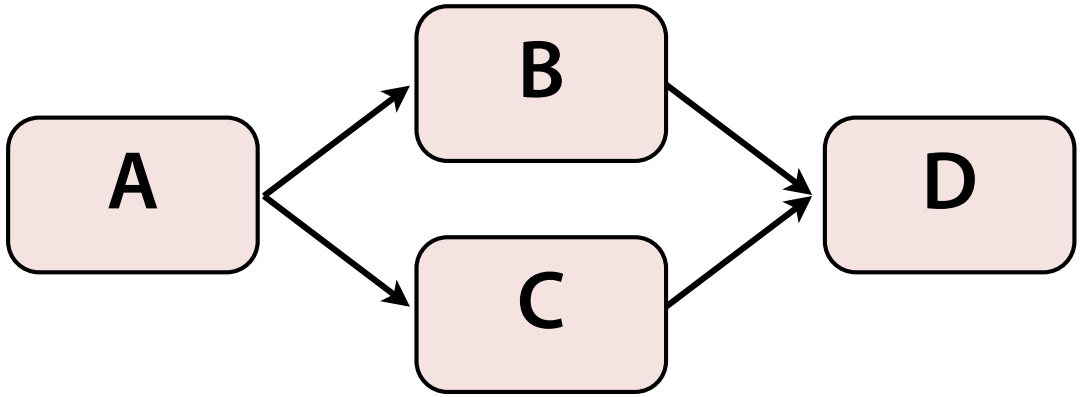
Disconnected



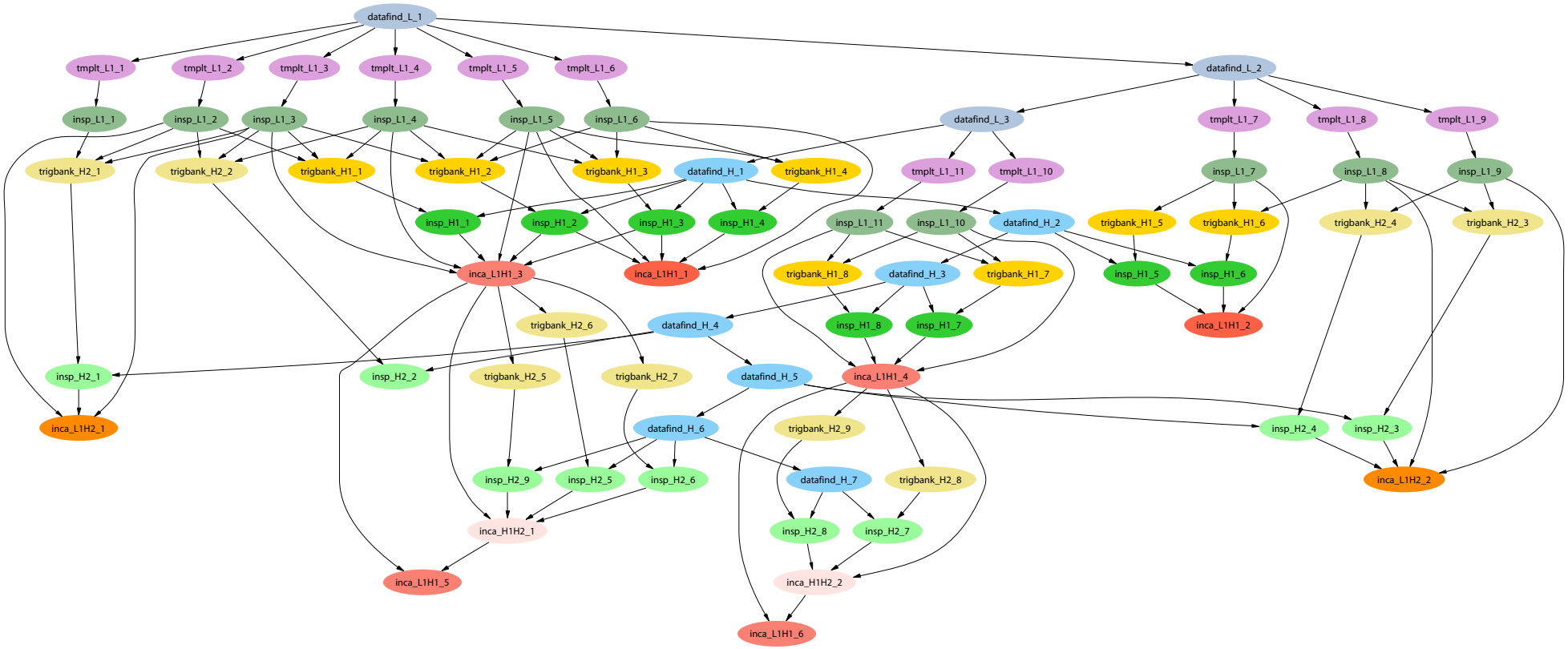
Linear / Serial



Diamond



A Real Scientific DAG

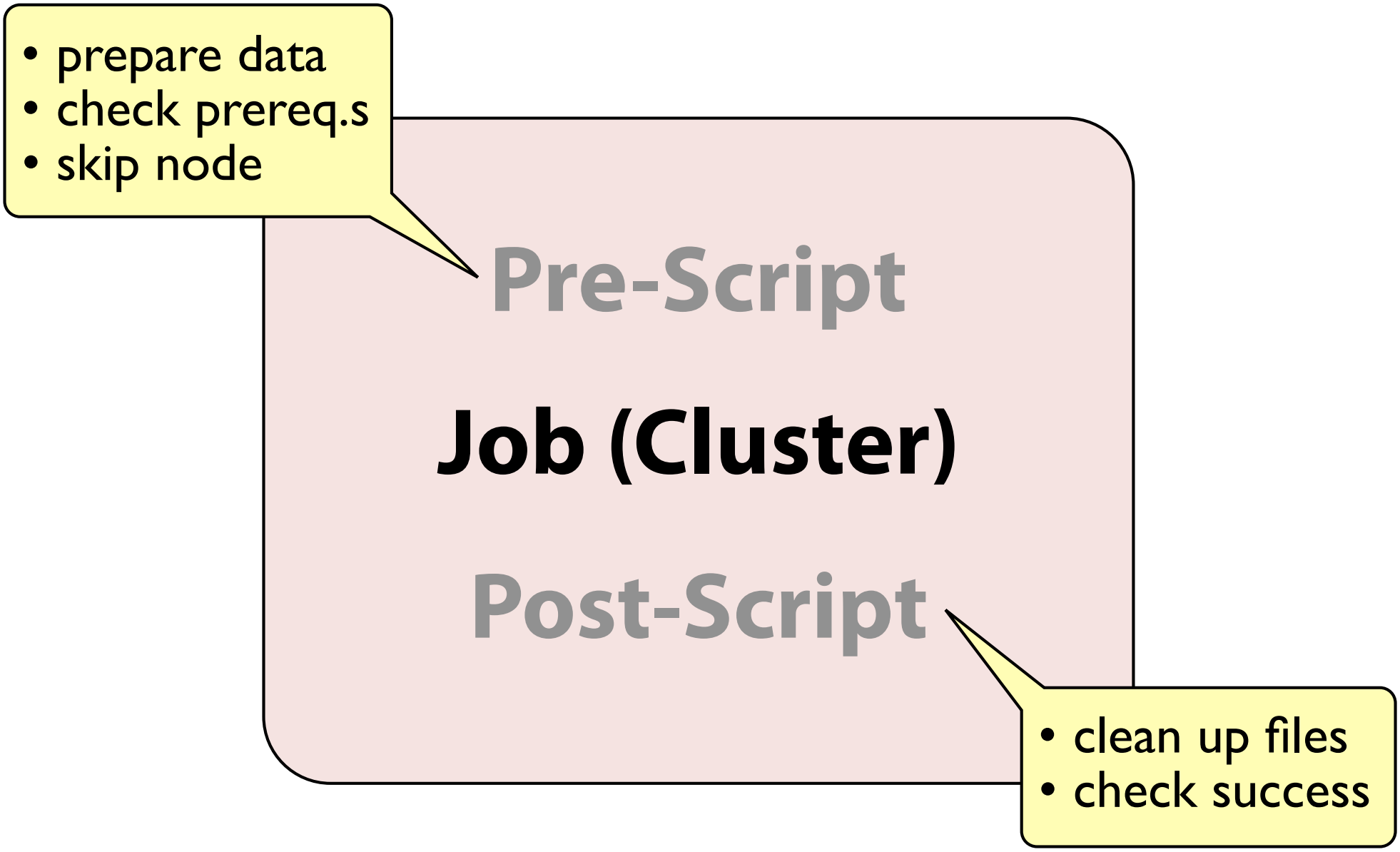


Laser Interferometer Gravitational-wave Observatory (LIGO)

Condor DAGMan

- **DAGMan: D**irected **A**cyclic **G**raph **Man**ager
- Organize Condor jobs into a DAG
- Condor handles ***all*** details of running workflow
  - Submits individual jobs when appropriate
  - Tracks overall workflow
  - Can retry failed nodes and resume failed workflow
  - Can limit amount of work done at once
- DAGs up to 1,000,000 nodes have been run!

DAGMan Nodes I



DAGMan Nodes II

- **Order of execution**
  1. Pre-script *on submit machine*
  2. Job(s) *on pool*
  3. Post-script *on submit machine*
- **Failure handling**
  - Pre-script exit **≠ 0**: Skip job, run post-script (if any)
  - Any job exit **≠ 0**: Run post-script (if any)
  - ***Last exit status*** determines success/failure of node
- Make sure scripts exit 0 upon success!
- Can skip job & post on given pre-script exit status

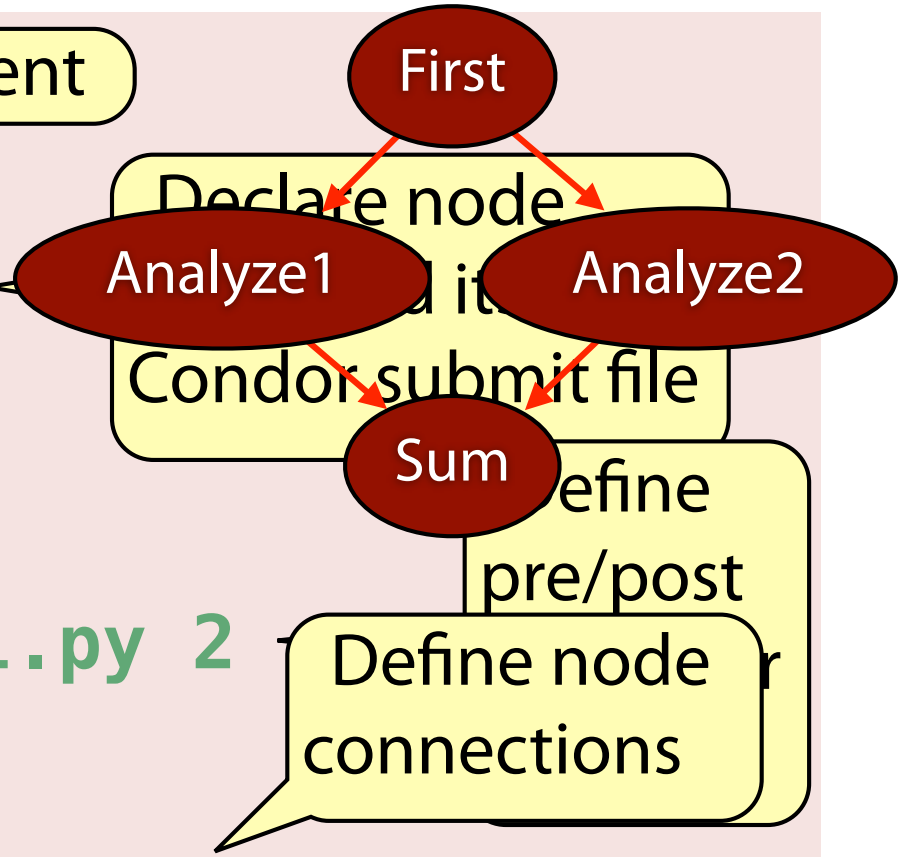
DAGMan Files

Basic DAGMan Submit File

```
# Define nodes
JOB First     first.sub
JOB Analyze1  stats-1.sub
JOB Analyze2  stats-2.sub
JOB Sum       collate.sub

SCRIPT PRE Sum verify-all.py 2

# Define connections
PARENT First CHILD Analyze1 Analyze2
PARENT Analyze1 Analyze2 CHILD Sum
```



Define a Job

```
JOB name submit-file
```

- One per node
- Defines node's ***name***, unique within this DAG
- Associated with a Condor ***submit-file***
- Job must yield 1 cluster; *may* have many processes

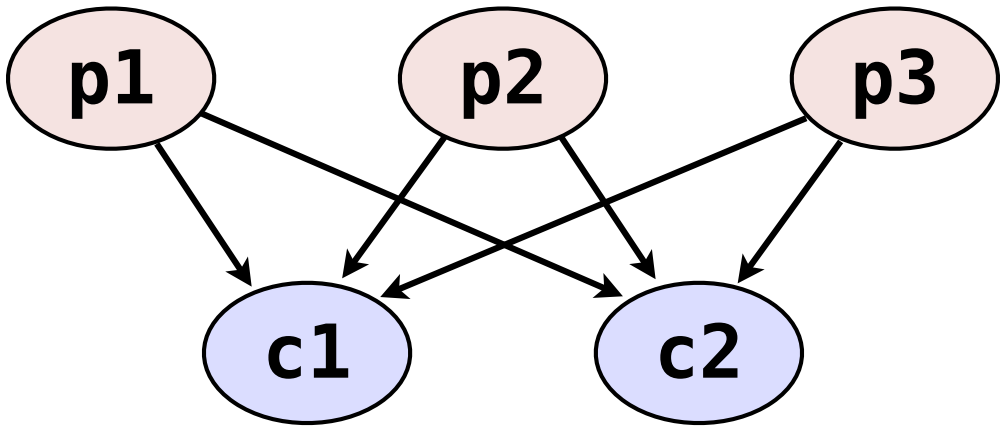
```
JOB Collate collate.sub
JOB Rjob3 run-r-3.sub
```

Define Dependencies

```
PARENT parent1 p2 … CHILD child1 c2 …
```

- Defines the "lines" (dependencies) between nodes
- Parent and child names are node names (cf. **JOB**)
- *EACH* child depends on *ALL* parents

```
PARENT p1 p2 p3 CHILD c1 c2
```



Define Pre- and Post-Scripts

```
SCRIPT PRE  name executable arguments
SCRIPT POST name executable arguments
```

- Scripts are always optional!
- Associated with given node ***name***
- Optional ***arguments*** are passed to ***executable***
- Place scripts in same directory as node's submit file
- Scripts run ***on the submit machine***

```
JOB First prepare.sub
SCRIPT PRE First fetch-data.py

SCRIPT POST Collate sum-stats.py 100
```

Logs in DAGMan

- DAGMan tracks progress via your **`log`** files
- All nodes (i.e., submit files) can use same log file
  - Can be tricky for a person to decode
  - Best DAGMan performance
- Each node may have own log file
  - More like what you are used to
  - Easier to read for a person
  - Cannot use **`$(CLUSTER)`** or **`$(PROCESS)`**, though!
- Can omit log statement entirely!
  - DAGMan defaults to ***`dagfile`*`.nodes.log`**

DAGMan Commands

Submit a DAG

```
condor_submit_dag dag-file
```

- DAGMan itself runs as a Condor job
- On the submit machine
- This command creates submit file and submits it

```
File for submitting this DAG to Condor    : dagman.dag.condor.sub
Log of DAGMan debugging messages          : dagman.dag.dagman.out
Log of Condor library output              : dagman.dag.lib.out
Log of Condor library error messages      : dagman.dag.lib.err
Log of the life of condor_dagman itself   : dagman.dag.dagman.log

Submitting job(s).
1 job(s) submitted to cluster 65.
```

```
condor_submit_dag -no_submit dag-file
```

- Just creates DAGMan submit file, if you are curious

Submit Options

```
condor_submit_dag -maxjobs N dag-file
```

- Maximum number of jobs to submit at once
- Can help avoid overload on submit machine
- Can be limited further by administrator

```
condor_submit_dag -maxpre N dag-file
condor_submit_dag -maxpost N dag-file
```

- Limits pre- and post-scripts
- Again, helps avoid overload on submit machine

- All options are optional and can be combined

Monitor a DAG

```
condor_q -dag
```

- Same command as always; same options available
- But: Organizes DAG jobs visually
- Not required to use **-dag** option!

```
65.0 cat          11/22 15:43 0+00:11:23 R 0 2.2 condor_dagman -f -
67.0  |-Random1   11/22 15:54 0+00:00:00 I 0 0.0 dag_2.py
68.0  |-Random2   11/22 15:54 0+00:00:00 I 0 0.0 dag_2.py
```

- Other options:
  - Watch log file(s)
  - Email notifications on ***each*** job (maybe just on last?)
  - Node status file (later slide)

Remove a DAG

```
condor_rm jobID
```

- But, which job ID?
- Essentially: remove the **condor_dagman** job itself
- Same cluster printed by **condor_submit_dag**
- Removes all jobs (idle & running) within DAG

```
65.0 cat          11/22 15:43 0+00:11:23 R 0 2.2 condor_dagman -f -
67.0  |-Random1   11/22 15:54 0+00:00:00 I 0 0.0 dag_2.py
68.0  |-Random2   11/22 15:54 0+00:00:00 I 0 0.0 dag_2.py
```

Job Recovery

- ***Rescue DAG*** created when DAG does not succeed
  - Due to being removed, or
  - After node fails, when all possible progress completes

```
-rw-rw-r-- 1 cat cat     512 Nov 23 10:26 sleep.dag
-rw-r--r-- 1 cat cat     988 Nov 23 10:38 sleep.dag.condor.sub
-rw-rw-r-- 1 cat cat     517 Nov 23 10:40 sleep.dag.dagman.log
-rw-r--r-- 1 cat cat   13179 Nov 23 10:40 sleep.dag.dagman.out
-rw-r--r-- 1 cat cat     261 Nov 23 10:40 sleep.dag.rescue001
```

- Resubmit the DAG to resume, using Rescue DAG
  - Completed nodes are not rerun

```
condor_submit_dag dagfile.rescueNNN   < 7.7.2
condor_submit_dag dagfile             ≥ 7.7.2
```

Status of DAG Nodes

```
NODE_STATUS_FILE filename seconds
```

- Writes DAG status info to the given ***filename***
- Overwrites file no more often than ***seconds*** apart

```
JOB A STATUS_DONE       ()
JOB B STATUS_DONE       ()
JOB C STATUS_DONE       ()
JOB D STATUS_DONE       ()
JOB E STATUS_DONE       ()
JOB F STATUS_SUBMITTED (not_idle)
JOB G STATUS_SUBMITTED (idle)
JOB H STATUS_UNREADY    ()
```

Homework

Homework

- Run a workflow!
- The queue simulator is back, but does its own loops
- If you have an alternate workflow that you would like to work on instead, talk to me