



Open Science Grid

# Intermediate Condor: DAGMan

Tuesday, Dec 7, 10:45am

Zach Miller <[zmiller@cs.wisc.edu](mailto:zmiller@cs.wisc.edu)>  
University of Wisconsin-Madison

# Before we begin...

---

- Any questions on the lectures or exercises up to this point?





# DAGMan

Directed  
Acyclic  
Graph

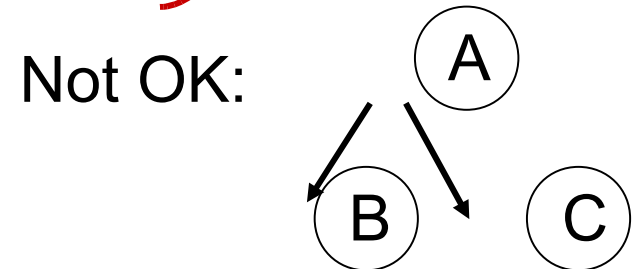
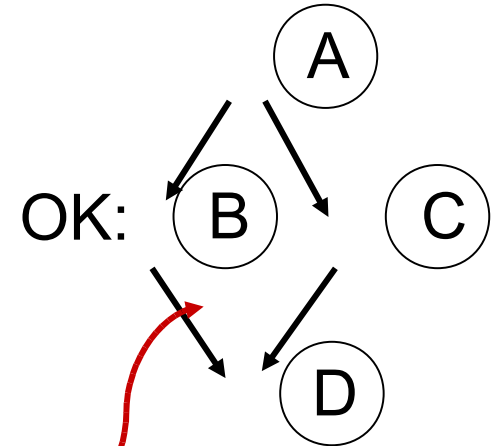
Manager

- DAGMan allows you to specify the dependencies between your Condor jobs, so it can manage them automatically for you.
- Example: “Don’t run job B until job A has completed successfully.”



# What is a DAG?

- A DAG is the data structure used by DAGMan to represent these dependencies.
- Each job is a node in the DAG.
- Each node can have any number of “parent” or “children” nodes – as long as there are no loops!

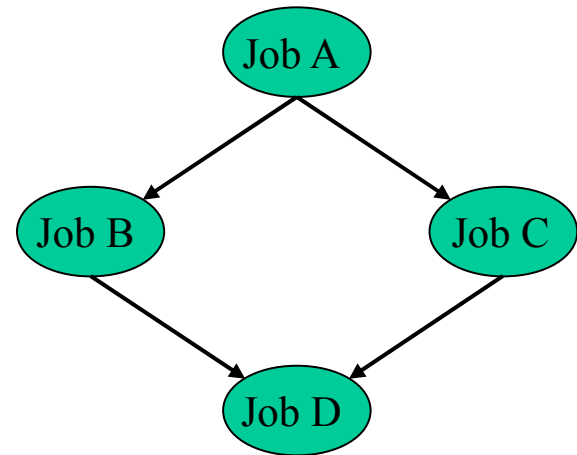


# Defining a DAG

- A DAG is defined by a *.dag file*, listing each of its nodes and their dependencies:

```
Job A a.sub  
Job B b.sub  
Job C c.sub  
Job D d.sub
```

```
Parent A Child B C  
Parent B C Child D
```



# DAG Files....

- The complete DAG has five files

One DAG File:

```
job A a.sub
job B b.sub
job C c.sub
job D d.sub

parent A Child B C
parent B C Child D
```

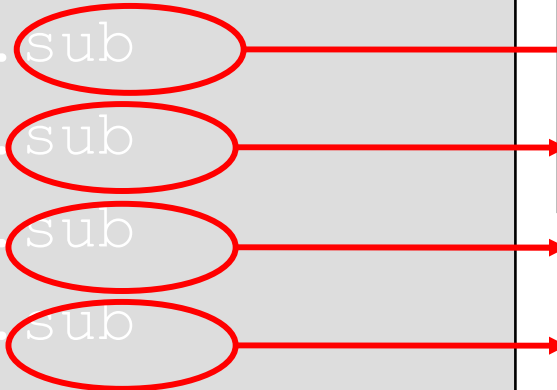
Four Submit Files:

```
Universe = Vanilla
Executable = analysis...
```

```
[Redacted Submit File 2]
```

```
[Redacted Submit File 3]
```

```
[Redacted Submit File 4]
```



# Submitting a DAG

---

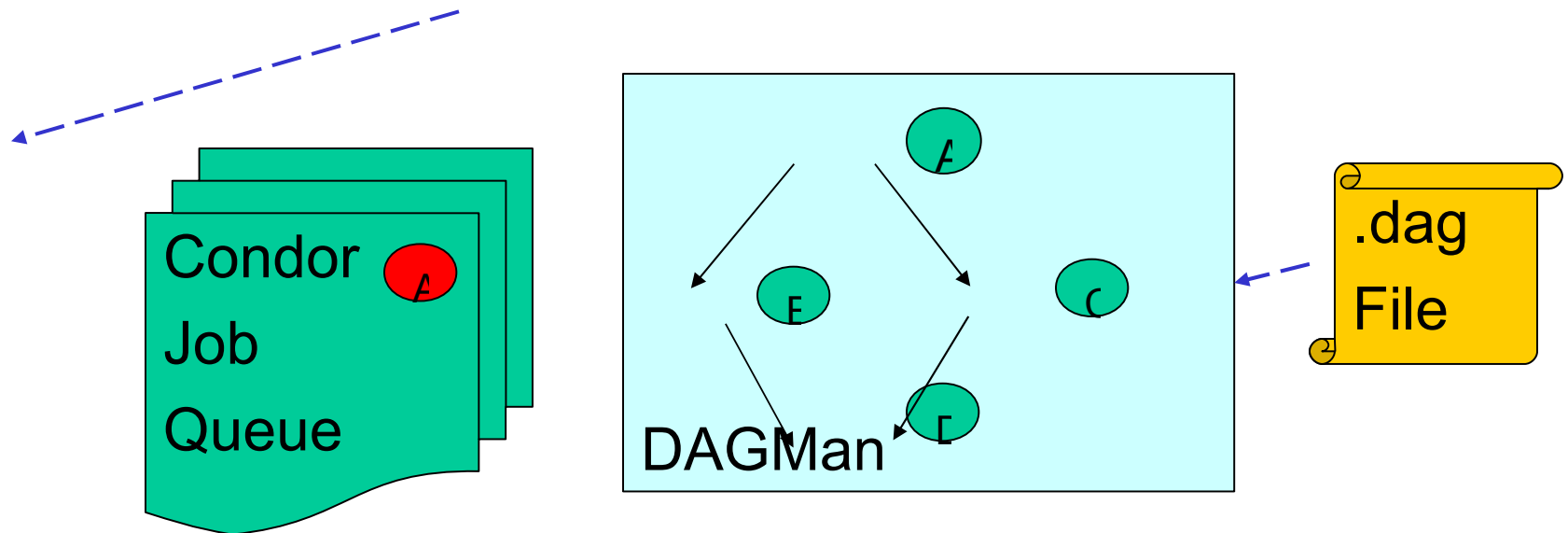
- To start your DAG, just run `condor_submit_dag` with your `.dag` file, and Condor will start a personal DAGMan process which to begin running your jobs:

```
% condor_submit_dag diamond.dag
```

- `condor_submit_dag` submits a Scheduler Universe job with DAGMan as the executable.
- Thus the DAGMan daemon itself runs as a Condor job, so you don't have to baby-sit it.

# Running a DAG

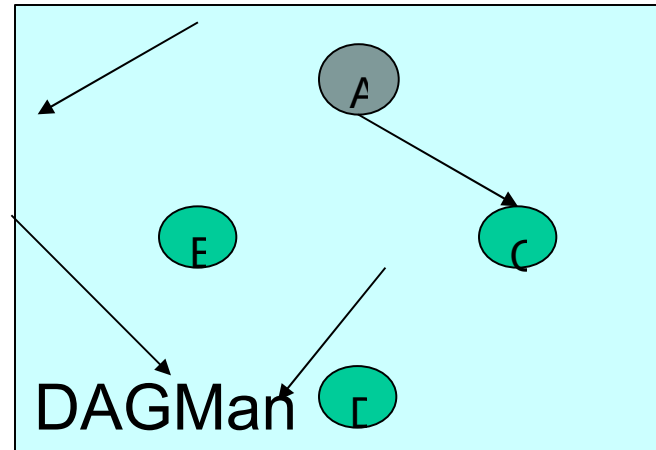
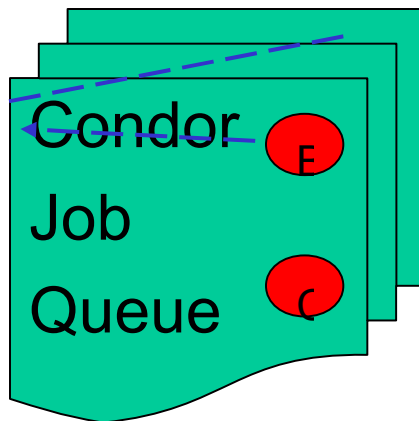
- DAGMan acts as a scheduler, managing the submission of your jobs to Condor based on the DAG dependencies.





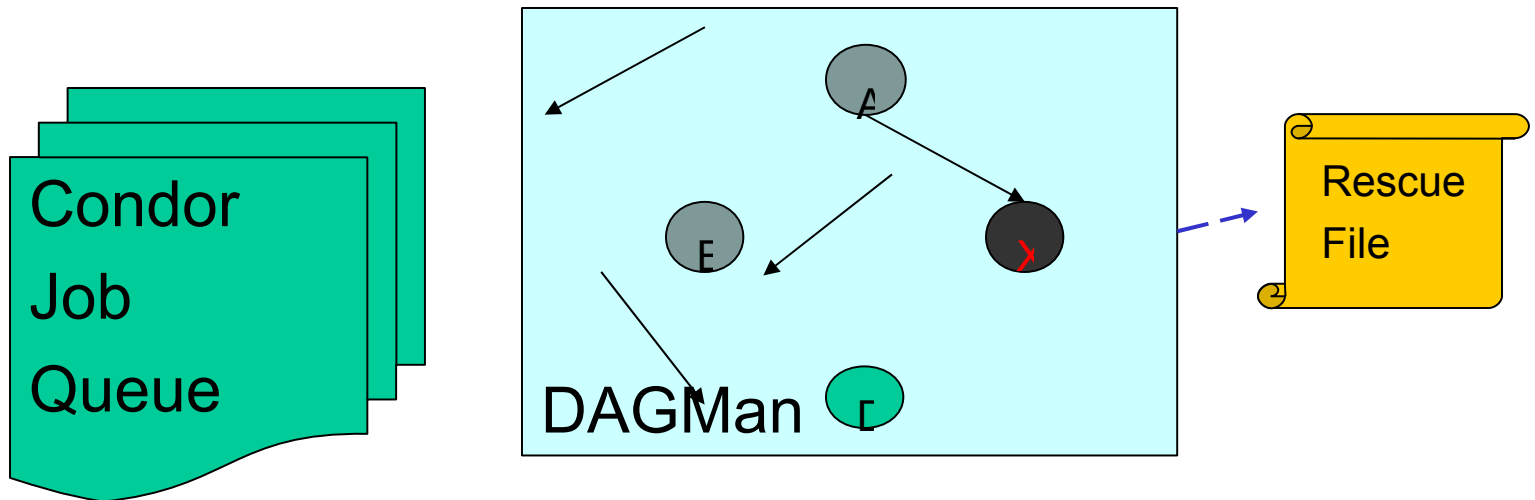
# Running a DAG (cont'd)

- DAGMan holds & submits jobs to the Condor queue at the appropriate times.



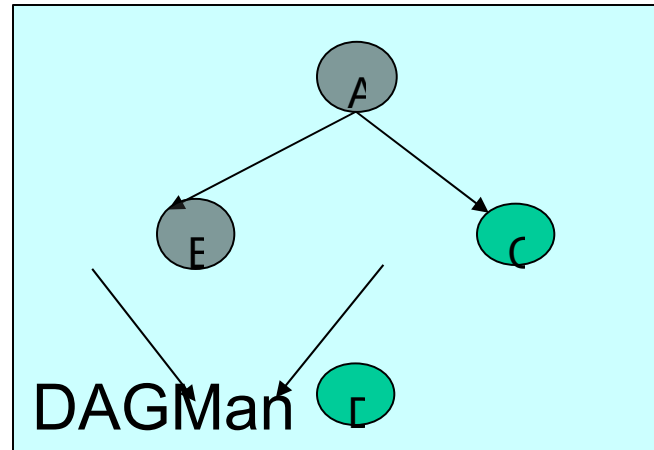
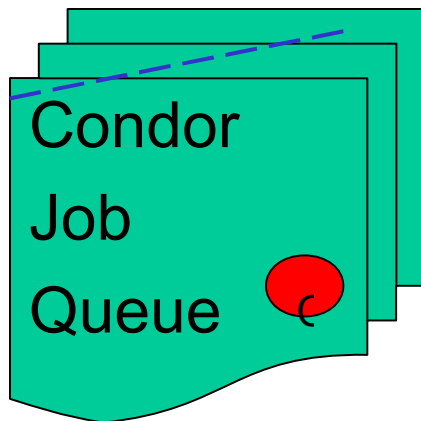
# Running a DAG (cont'd)

- In case of a job failure, DAGMan continues until it can no longer make progress, and then creates a “*rescue*” file with the current state of the DAG.



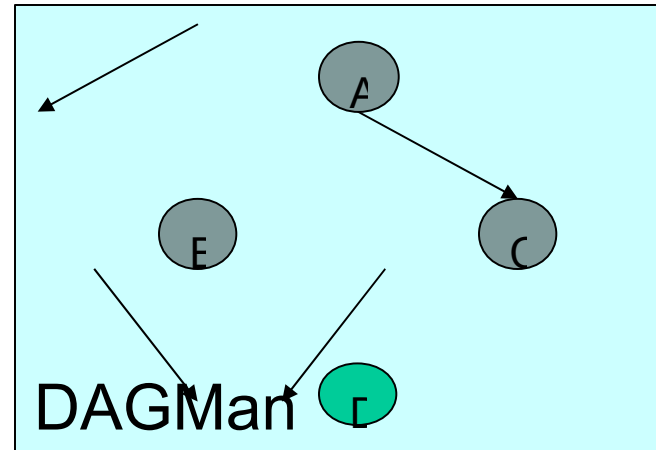
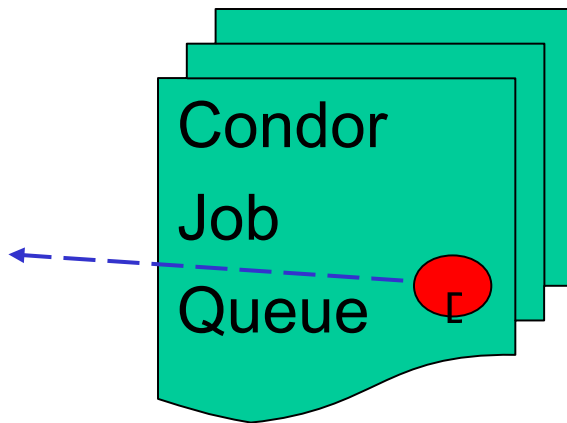
# Recovering a DAG

- Once the failed job is ready to be re-run, the rescue file can be used to restore the prior state of the DAG.



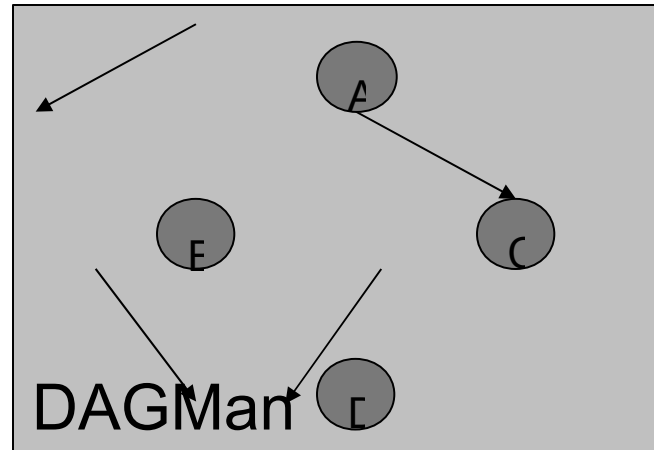
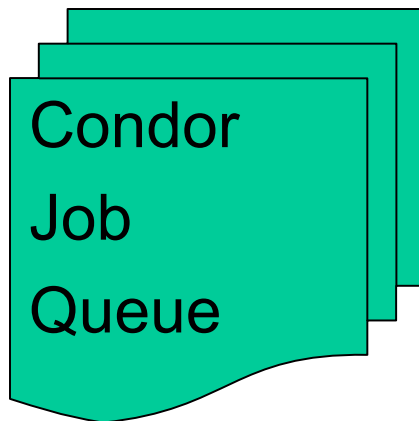
# Recovering a DAG (cont'd)

- Once that job completes, DAGMan will continue the DAG as if the failure never happened.

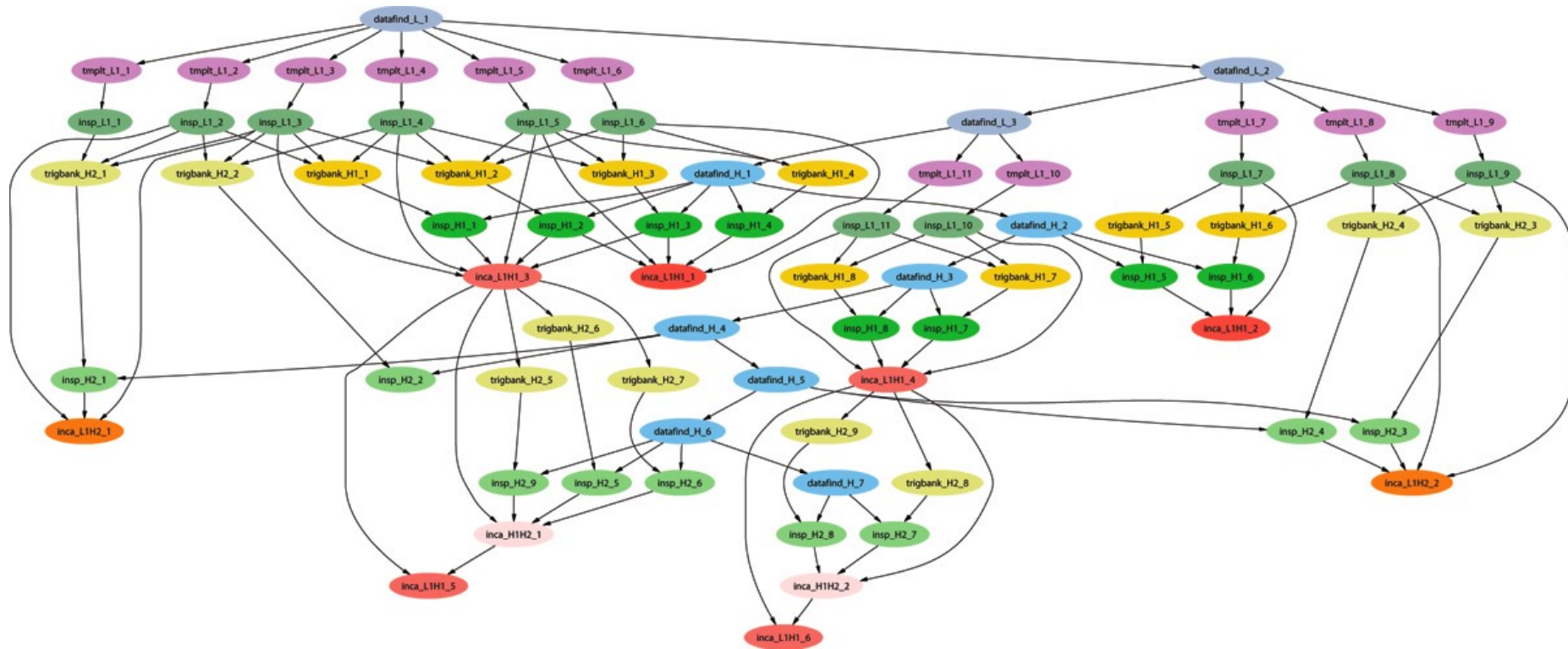


# Finishing a DAG

- Once the DAG is complete, the DAGMan job itself is finished, and exits.

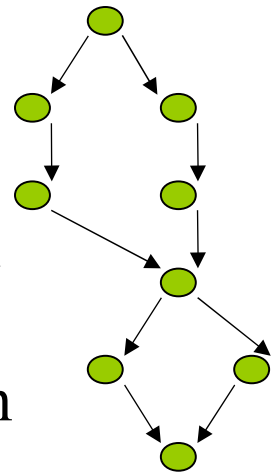


# Example of a LIGO Inspiral DAG



# Use of Condor by the LIGO Scientific Collaboration

- Condor handles 10's of millions of jobs per year running on the LDG, and up to 500k jobs per DAG.
- Condor standard universe check pointing widely used, saving us from having to manage this.
- At Caltech, 30 million jobs processed using 22.8 million CPU hrs. on 1324 CPUs in last 30 months.
- For example, to search 1 yr. of data for GWs from the inspiral of binary neutron star and black hole systems takes ~2 million jobs, and months to run on several thousand ~2.6 GHz nodes.



# DAGMan & Fancy Features

---

- DAGMan doesn't have a lot of “fancy features”
  - No loops
  - Not much assistance in writing very large DAGs (script it yourself)
- Focus is on solid core
  - Add the features people need in order to run large DAGs well.
  - People build systems on top of DAGMan





# DAGMan & Log Files

---

- For each job, Condor generates a log file
- DAGMan reads this log to see what has happened
- If DAGMan dies (crash, power failure, etc...)
  - Condor will restart DAGMan
  - DAGMan re-reads log file
  - DAGMan knows everything it needs to know

# Advanced DAGMan Tricks

---

- Throttles and degenerative DAGs
- Sub-DAGs
- Pre and Post scripts: editing your DAG



# Throttles

---

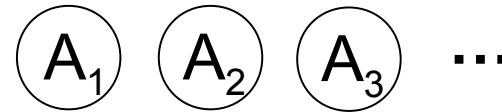
- Failed nodes can be automatically retried a configurable number of times
  - Can retry N times
  - Can retry N times, unless a node returns specific exit code
- Throttles to control job submissions
  - Max jobs submitted
  - Max scripts running



# Degenerative DAG

- Submit DAG with:

- 200,000 nodes



- No dependencies

- Use DAGMan to throttle the jobs

- Condor is scalable, but it will have problems if you submit 200,000 jobs simultaneously

- DAGMan can help you get scalability even if you don't have dependencies



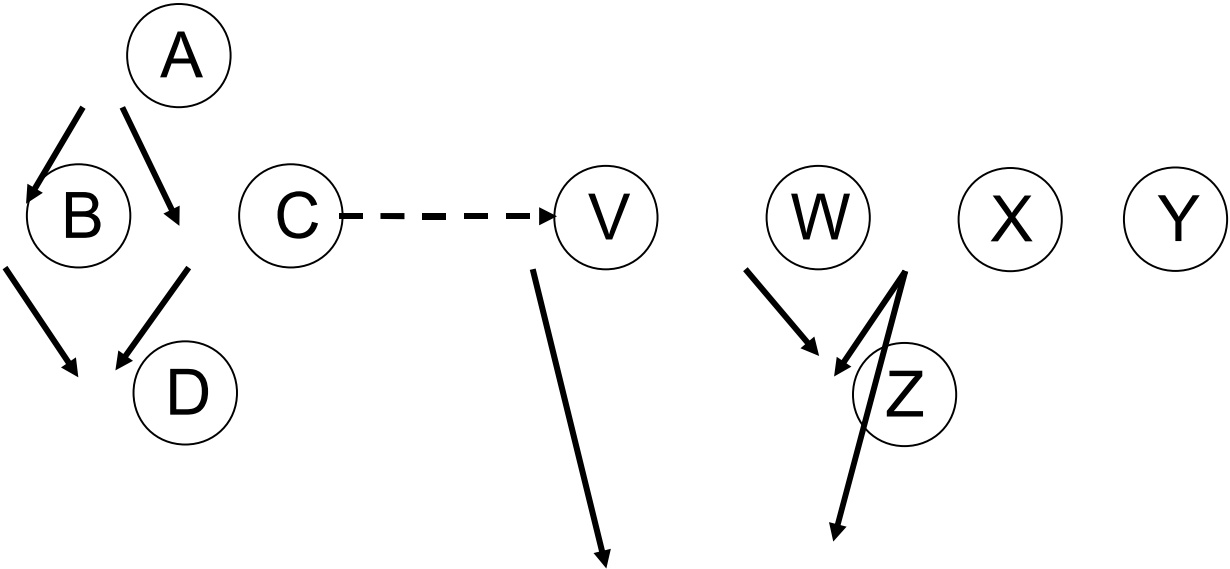
# Sub-DAG

---

- Idea: any given DAG node can be another DAG
  - SUBDAG External Name DAG-file
- DAG node will not complete until recursive DAG finishes,
- Interesting idea: A previous node could *generate* this DAG node
- Why?
  - Simpler DAG structure
  - Implement a fixed-length loop
  - Modify behavior on the fly



# Sub-DAG



# DAGMan scripts

---

- DAGMan allows pre & post scripts
  - Don't have to be scripts: any executable
  - Run before (pre) or after (post) job
  - Run on the same computer you submitted from

- **Syntax:**

```
JOB A a.sub
```

```
SCRIPT PRE A before-script $JOB
```

```
SCRIPT POST A after-script $JOB $RETURN
```



# So What?

---

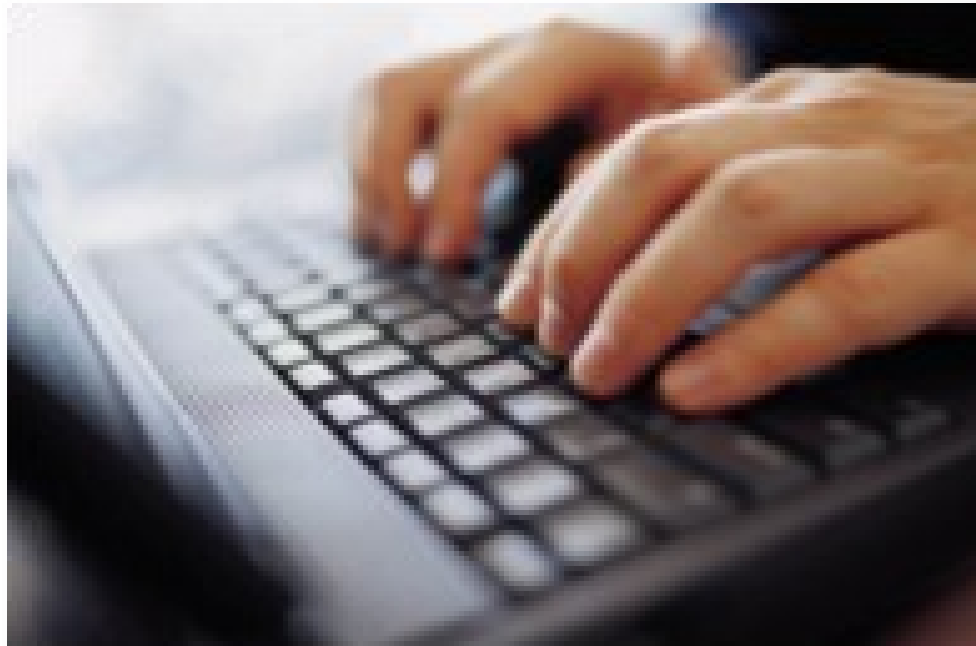
- Pre script can make decisions
  - Where should my job run? (Particularly useful to make job run in same place as last job.)
  - Generate Sub-DAG
  - Should I pass different arguments to the job?
  - Lazy decision making
    - Noop jobs
- Post script can change return value
  - DAGMan decides job failed in non-zero return value
  - Post-script can look at {error code, output files, etc} and return zero or non-zero based on deeper knowledge.



# Let's try it out!

---

- Exercises with DAGMan.



# Questions?

---

- Questions? Comments?
- Feel free to ask me questions later:  
Zach Miller <[zmiller@cs.wisc.edu](mailto:zmiller@cs.wisc.edu)>
- Upcoming sessions
  - Now – 12:30pm
    - Hands-on exercises
  - 2:15pm – 3:30pm
    - Turning science problems into HTC jobs
  - 3:45pm – 5:30pm
    - Final Condor session