



Workflows with HTCondor's DAGMan

OSG School 2026

Andrew Owen

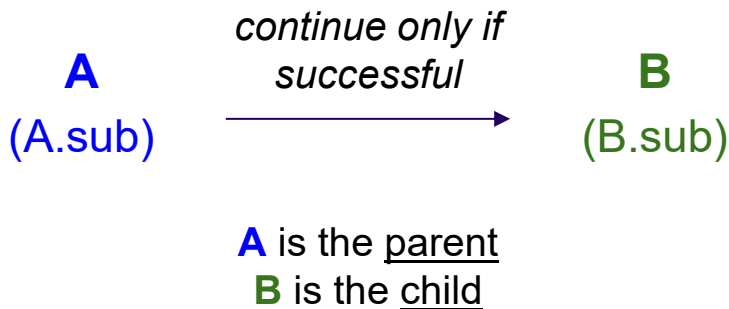


Scenario

You have two jobs to run: job **A** and job **B**.

You have two corresponding template submit files: **A.sub** and **B.sub**

You want job **B to run only after job **A** has completed successfully**





How?

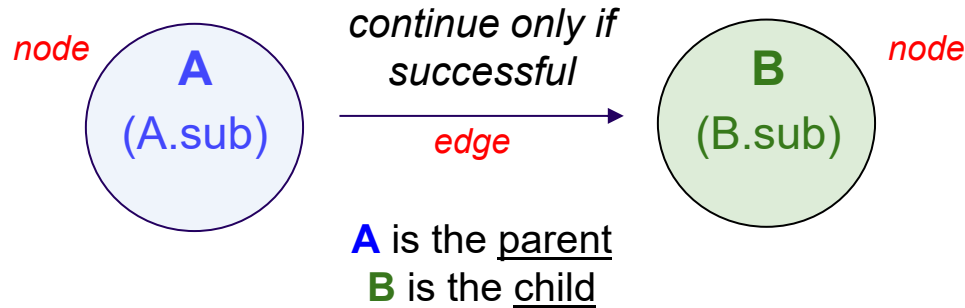
HTCondor offers you the services of the

Directed Acyclic Graph Manager → **DAGMan**

to automate the submission of jobs (with dependencies)

How?

The Directed Acyclic Graph Manager (**DAGMan**) manages the placement of lists of jobs represented by "nodes" that are connected by "edges"





Create the DAG input file

In a file, you need to

(1) declare the job submissions and (2) declare the dependencies.



my-first.dag



Create the DAG input file

In a file, you need to

(1) declare the job submissions and (2) declare the dependencies.

```
JOB A A.sub  
JOB B B.sub
```

Syntax

JOB <node_name> <submit_file_name>

my-first.dag



Create the DAG input file

In a file, you need to

(1) declare the job submissions and (2) declare the dependencies.

```
JOB A A.sub  
JOB B B.sub
```

my-first.dag

Syntax

JOB <node_name> <submit_file_name>

If any job fails in this submit file,
the whole job (node) fails!



Create the DAG input file

In a file, you need to

(1) declare the job submissions and (2) declare the dependencies.

A is the parent

B is the child

```
JOB A A.sub
```

```
JOB B B.sub
```

```
PARENT A CHILD B
```

Syntax

PARENT <node_name> CHILD <node_name>

depends on

my-first.dag



Create the DAG input file

In a file, you need to

(1) declare the job submissions and (2) declare the dependencies.

```
JOB A A.sub  
JOB B B.sub  
  
PARENT A CHILD B
```

my-first.dag



Create the DAG input file

How can we tell if job **A** completed successfully?

- Default behavior: if the job exits with code 0 → successful

```
JOB A A.sub  
JOB B B.sub  
  
PARENT A CHILD B
```

my-first.dag



Create the DAG input file

How can we tell if job **A** completed successfully?

- For more complex checks, you can use a script

```
JOB A A.sub  
JOB B B.sub  
  
PARENT A CHILD B
```

my-first.dag



Create the DAG input file

How can we tell if job **A** completed successfully?

- For more complex checks, you can use a script

```
JOB A A.sub  
SCRIPT POST A A-check.sh  
JOB B B.sub  
  
PARENT A CHILD B
```

my-first.dag

Syntax

SCRIPT POST <node_name> <script_name>

**order of lines does not actually matter*

Submitting and Monitoring the DAG



Submit the DAG

By default, DAGMan expects the submit files `A.sub` and `B.sub` are in the same directory as `my-first.dag`, along with `A-check.sh`, on an HTCondor Access Point

Basic Working Directory

```
DAG_simple/  
|-- my-first.dag  
|-- A.sub  
|-- A-check.sh  
|-- B.sub
```



Submit the DAG

By default, DAGMan expects the submit files `A.sub` and `B.sub` are in the same directory as `my-first.dag`, along with `A-check.sh`, on an HTCondor Access Point

Basic Working Directory

```
DAG_simple/  
|-- my-first.dag  
|-- A.sub  
|-- A-check.sh  
|-- B.sub
```

It is possible to create other directory structures, but for now we will use this simple, flat organization.



Submit the DAG

Command to submit, or place, the DAGMan job:

```
condor_submit_dag <dag_description_file>  
condor_submit_dag my-first.dag
```

This then starts the DAG **node scheduler** job, which we can see in the queue:

```
[user@ap40 DAG_simple]$ condor_q
```

```
-- Schedd: ap40.uw.osg-htc.org : <128.105.68.92:9618?... @ 09/01/24 11:26:51  
OWNER    BATCH_NAME          SUBMITTED     DONE   RUN    IDLE   TOTAL  JOB_IDS  
user     my-first.dag+562265      09/01 11:26   _      _      1       2   562279.0
```



Monitor the DAG

This then starts the DAG **node scheduler** job, which we can see in the queue:

```
[user@ap40 DAG_simple]$ condor_q
```

```
-- Schedd: ap40.uw.osg-htc.org : <128.105.68.92:9618?... @ 09/01/24 11:26:51
OWNER   BATCH_NAME          SUBMITTED   DONE   RUN    IDLE   TOTAL  JOB_IDS
user    my-first.dag+562265   09/01 11:26   _     _      1       2   562279.0
```



BATCH_NAME for the DAGMan job is the name of the input description file, **my-first.dag**, plus the Job ID of the scheduler job (562265)



Monitor the DAG

This then starts the DAG **node scheduler** job, which we can see in the queue:

```
[user@ap40 DAG_simple]$ condor_q
```

```
-- Schedd: ap40.uw.osg-htc.org : <128.105.68.92:9618?... @ 09/01/24 11:26:51
OWNER   BATCH_NAME          SUBMITTED   DONE   RUN    IDLE   TOTAL   JOB_IDS
user    my-first.dag+562265    09/01 11:26   _     _      1       2    562279.0
```



The total number of jobs for **my-first.dag+562265** corresponds to the total number of nodes in the DAG (**2**)



Monitor the DAG

This then starts the DAG **node scheduler** job, which we can see in the queue:

```
[user@ap40 DAG_simple]$ condor_q
```

```
-- Schedd: ap40.uw.osg-htc.org : <128.105.68.92:9618?... @ 09/01/24 11:26:51
OWNER   BATCH_NAME          SUBMITTED   DONE   RUN    IDLE   TOTAL  JOB_IDS
user    my-first.dag+562265   09/01 11:26   _     _     1      2   562279.0
```



Only 1 node is listed as "Idle", so DAGMan has only submitted 1 job so far. This is consistent with the fact that node **A** has to complete before DAGMan can submit the job for node **B**.



Monitor the DAG

For more detailed monitoring:

```
[user@ap40 DAG_simple]$ condor_q -dag -nob
```

```
-- Schedd: ap40.uw.osg-htc.org : <128.105.68.92:9618?... @ 12/14/23 11:27:03
```

ID	OWNER/NODENAME	SUBMITTED	RUN_TIME	ST	PRI	SIZE	CMD
562265.0	user	09/01 11:26	0+00:00:37	R	0	0.5	condor_dagman
562279.0	-A	09/01 11:26	0+00:00:00	I	0	0.0	A.sh

First entry: **dag node scheduler job** created upon submission



Monitor the DAG

For more detailed monitoring:

```
[user@ap40 DAG_simple]$ condor_q -dag -nob
```

```
-- Schedd: ap40.uw.osg-htc.org : <128.105.68.92:9618?... @ 12/14/23 11:27:03
```

ID	OWNER/NODENAME	SUBMITTED	RUN_TIME	ST	PRI	SIZE	CMD
562265.0	user	09/01 11:26	0+00:00:37	R	0	0.5	condor_dagman
562279.0	-A	09/01 11:26	0+00:00:00	I	0	0.0	A.sh

Additional entries: correspond to **nodes** whose jobs are **currently** in the queue.

- *Reminder: Nodes that have not yet been submitted by DAGMan or that have completed and thus left the queue will not show up in condor_q output.*



Additional Tools to Monitor your Workflow

DAGMan will produce helpful files to learn about and troubleshoot your workflow.

```
[user@ap40 DAG_simple]$ condor_submit_dag my-first.dag
```

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

```
Submitting job(s).  
1 job(s) submitted to cluster 562265.  
-----
```



Overview of process

JOB **A** **A.sub**

JOB **B** **B.sub**

PARENT **A** CHILD **B**

my-first.dag



condor_submit_dag

1. DAG node scheduler job starts
2. **A.sub** executes → completes
3. **A** returned exit code 0 → continue
4. **B.sub** executes → completes
5. DAG node scheduler job completes



Overview of process

```
JOB A A.sub  
SCRIPT POST A A-check.sh  
JOB B B.sub  
  
PARENT A CHILD B
```

my-first.dag

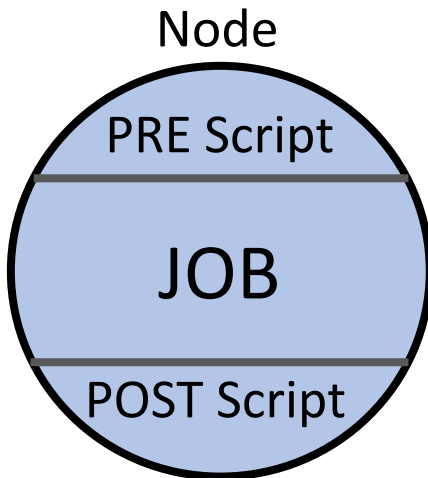
condor_submit_dag

1. DAG node scheduler job starts
2. **A.sub** executes → completes
3. **A-check.sh** **succeeds** → **continue**
4. **B.sub** executes → completes
5. DAG node scheduler job completes



PRE/POST scripts

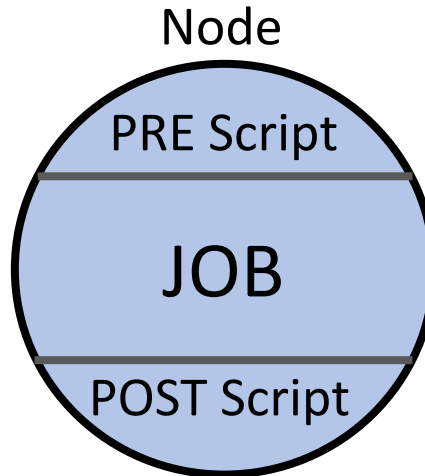
- All DAGMan PRE/POST scripts run on the Access Point and not on an Execution Point Slot.
- Scripts provide a way to perform tasks at key points in a node's lifetime.
 - *E.g., checking if files exist, creating directories, consolidating files*
- Should be lightweight (low computational) programs/tasks





What is Considered a Failure

- A **non-zero exit code** in the PRE script, JOB, or POST script is considered a failure
- DAGMan will continue running work until can no longer progress



Overall

DAGMan will do *as much work as it can* until completion ("success") or failure



A Failed DAG

- Once a node has failed and no more progress in the DAG can be made, DAGMan will produce a rescue file and exit.
 - Rescue file is named **<dag_description_file>.rescue001**
 - "001" increments for each new rescue file
 - Records which NODEs have completed successfully
 - does not contain the actual DAG structure

DAG_simple/

A.sub	B.sub	check-A.sh
my-first.dag	my.-first.dag.condor.sub	my.dag.dagman.log
my-first.dag.dagman.out	my-first.dag.lib.err	my-first.dag.lib.out
my-first.dag.metrics	my-first.dag.nodes.log	my-first.dag.rescue001
<i>(other job files)</i>		



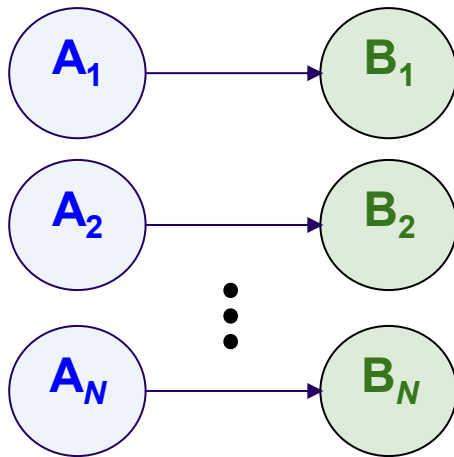
Dealing with a Failed DAG

- Search for issue in `<dag filename>.dagman.out` and job standard error/output files
- Once issue is fixed, resubmit with `condor_submit_dag`
 - Rescue file will be automatically detected and progress will resume from the point it left off

Many DAGs

Many DAGs

Scenario: Now you have to run the $A \rightarrow B$ workflow many times in parallel



How to accomplish?



Many DAGs ... or One Big DAG

Write a script that generates your DAG description file* for you
(and the needed files)

```
JOB A A.sub  
JOB B B.sub  
  
PARENT A CHILD B
```

my-first.dag

*for now. We are working to develop better of ways of handling this scenario.



Many DAGs ... or One Big DAG

Write a script that generates your DAG description file* for you
(and the needed files)

```
JOB A A.sub  
JOB B B.sub  
  
PARENT A CHILD B
```

my-first.dag

python
bash
...

```
JOB A1 A1.sub  
JOB B1 B1.sub  
PARENT A1 CHILD B1  
  
JOB A2 A2.sub  
JOB B2 B2.sub  
PARENT A2 CHILD B2  
⋮
```

my-big.dag

*for now. We are working to develop better of ways of handling this scenario.



One Big DAG

Once ready, do a single `condor_submit_dag` command

The DAG node scheduler job will manage all of the submissions while keeping track of the dependencies

```
JOB A1 A1.sub
JOB B1 B1.sub
PARENT A1 CHILD B1

JOB A2 A2.sub
JOB B2 B2.sub
PARENT A2 CHILD B2
      ⋮
```

my-big.dag



One Big DAG – Reuse Files

In the big DAG, there were a lot of similar files: **A{x}.sub**, **B{x}.sub**

Instead of **A1.sub**, **A2.sub**, ... **AN.sub**, can use **A.sub**

```
JOB A1 A.sub
JOB B1 B1.sub
PARENT A1 CHILD B1
```

```
JOB A2 A.sub
JOB B2 B2.sub
PARENT A2 CHILD B2
```

⋮

my-big.dag



One Big DAG – Reuse Files

In the big DAG, there were a lot of similar files: **A{x}.sub**, **B{x}.sub**

Instead of **A1.sub**, **A2.sub**, ... **AN.sub**, can use **A.sub**

Then pass the number to the submit file with the **VARs** command

DAG Description File Syntax

VARs <node_name> <variable>=<value>

Submit File Syntax

arguments = \$(<variable>)

```
JOB A1 A.sub
VARs A1 number=1
JOB B1 B1.sub
PARENT A1 CHILD B1
```

```
JOB A2 A.sub
VARs A2 number=2
JOB B2 B2.sub
PARENT A2 CHILD B2
```

:

my-big.dag



One Big DAG – Reuse Files

In the big DAG, there were a lot of similar files: **A{x}.sub**, **B{x}.sub**

Instead of **A1.sub**, **A2.sub**, ... **AN.sub**, can use **A.sub**

Then pass the number to the submit file with the **VARs** command

Can repeat for **B.sub**

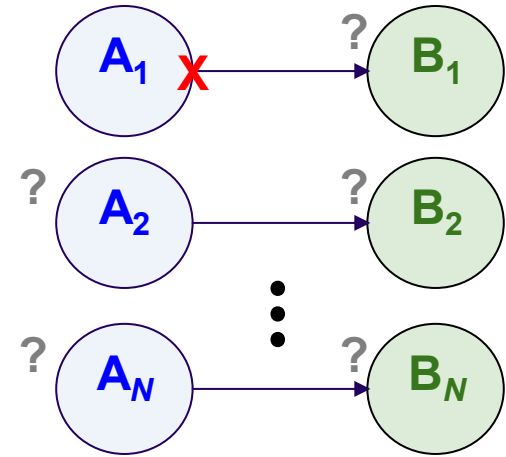
```
JOB A1 A.sub
VARs A1 number=1
JOB B1 B.sub
VARs B1 number=1
PARENT A1 CHILD B1
```

```
JOB A2 A.sub
VARs A2 number=2
JOB B2 B.sub
VARs B1 number=2
PARENT A2 CHILD B2
```

:

One Big DAG - What If It Fails?

Let's say that A_1 job finishes and `A-check.sh` finds that the output of A_1 is incorrect, and that A_1 has failed. What happens?



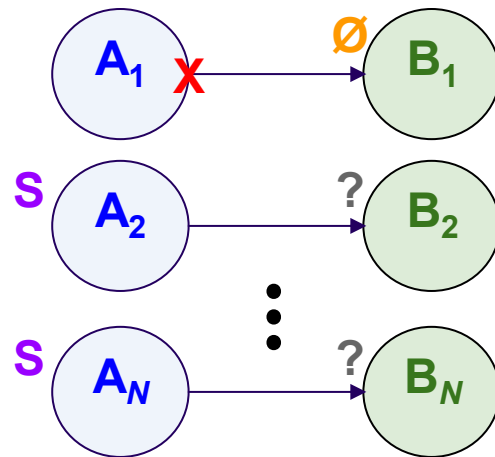
X = Failed

? = Not known yet

One Big DAG - What If It Fails?

Let's say that A_1 job finishes and `A-check.sh` finds that the output of A_1 is incorrect, and that A_1 has failed. What happens?

- **DAGMan does as much work as it can, then creates a Rescue DAG.**
- While B_1 won't be started, the DAG node scheduler will keep submitting and managing the other A_N & B_N jobs until there is no more work.



S = Submitted

X = Failed

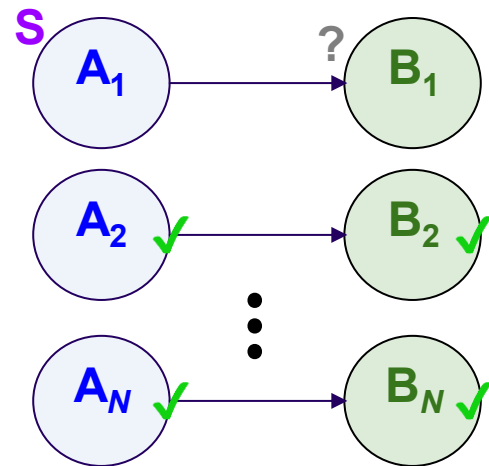
$\emptyset$ = Will not be submitted

? = Not known yet

One Big DAG - What If It Fails

Let's say that A_1 job finishes and `A-check.sh` finds that the output of A_1 is incorrect, and that A_1 has failed. What happens?

- The Rescue DAG is used automatically the next time you run `condor_submit_dag`, and the DAG node scheduler job will only submit the unsuccessful nodes.
 - If all but $A_1 \rightarrow B_1$ completed successfully, then when the Rescue DAG is submitted, only the $A_1 \rightarrow B_1$ will be attempted.



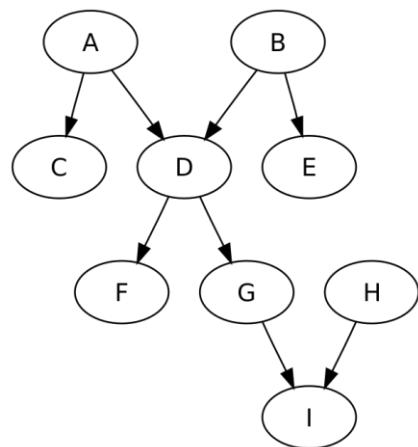
S = Submitted

? = Not known yet

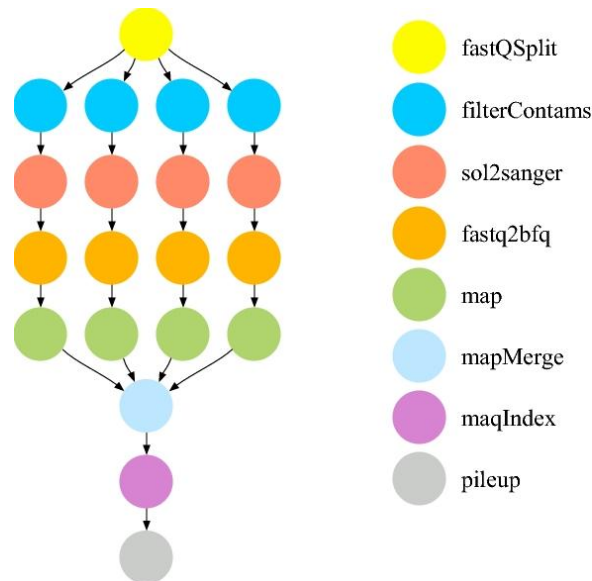
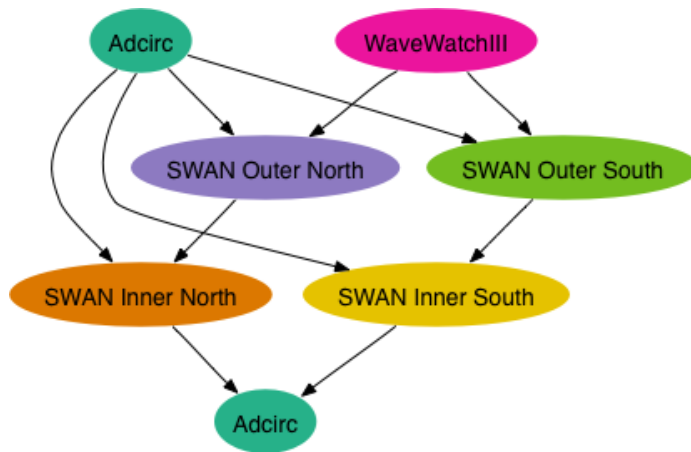
✓ = Successful completion

Endless Workflow Possibilities

Shared ancestor workflows



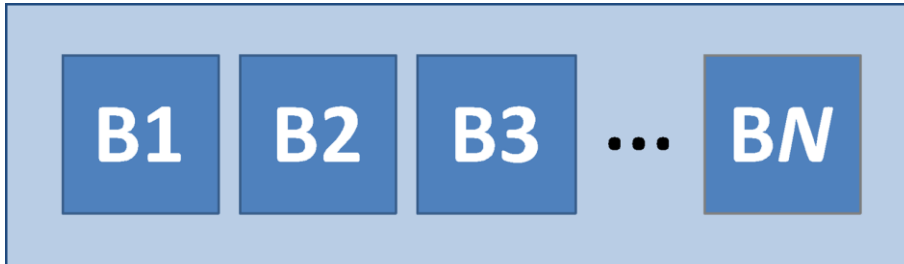
Wikimedia Commons



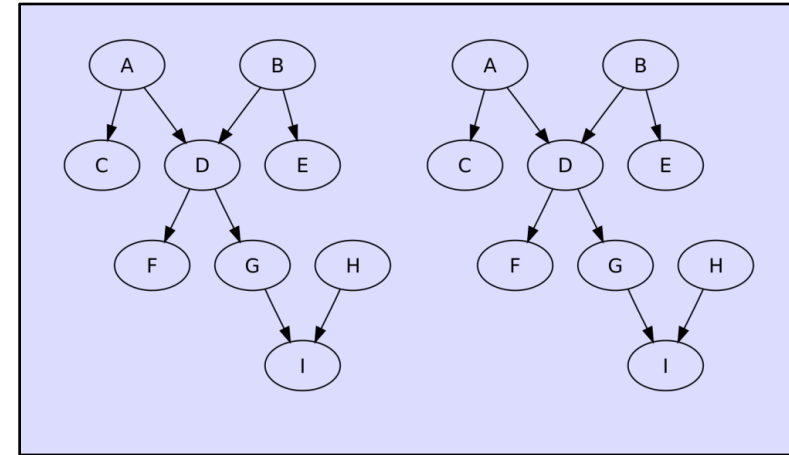
Endless Workflow Possibilities

Different ancestor workflows

A "Bag" of Jobs



Disjoined Workflows





Learn More

- Beginner DAGMan Resources:
 - <https://www.youtube.com/watch?v=OulBf6x24r0&pp=ygUGZGFnbWFu>
 - https://portal.osg-htc.org/documentation/htc_workloads/automated_workflows/dagman-workflows/
 - https://portal.osg-htc.org/documentation/htc_workloads/automated_workflows/dagman-simple-example/
- Intermediate DAGMan Resources:
 - https://portal.osg-htc.org/documentation/support_and_training/training/osgusertraining/
 - <https://github.com/OSGConnect/tutorial-dagman-intermediate>
- DAGMan Core Documentation
 - <https://htcondor.readthedocs.io/en/latest/automated-workflows/index.html>





Questions?



Submit File Templates via *VAR*S

- **VAR**S line defines node-specific values that are passed into submit file variables

*VAR*S *node_name* *var1*="value" [*var2*="value"]

- Allows a single submit file shared by all B jobs, rather than one submit file for each JOB.

my.dag

```
JOB B1 B.sub
VARS B1 data="B1" opt="10"
JOB B2 B.sub
VARS B2 data="B2" opt="12"
JOB B3 B.sub
VARS B3 data="B3" opt="14"
```

B.sub

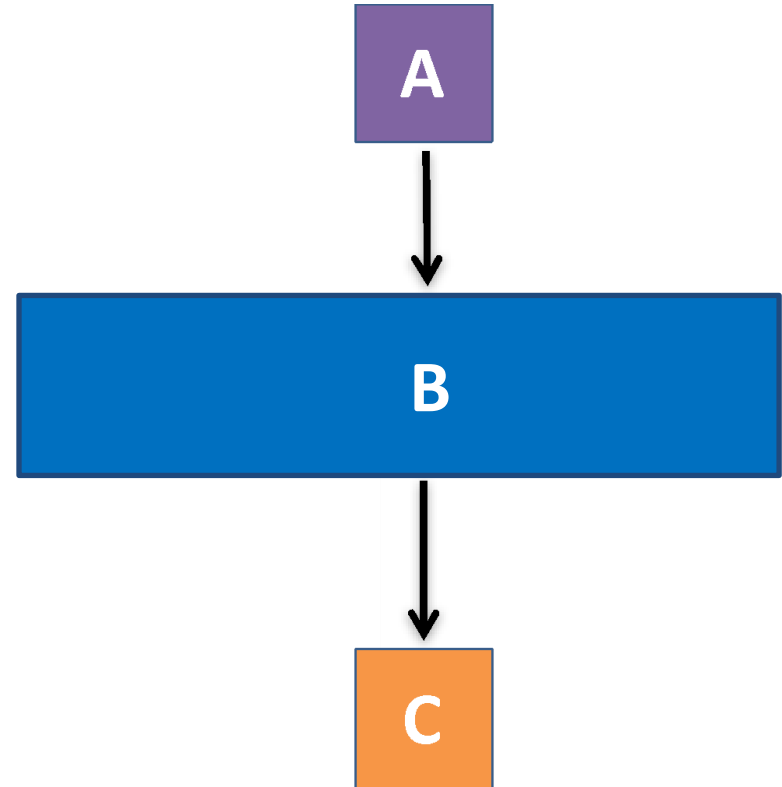
```
...
InitialDir = $(data)
arguments = $(data).csv $(opt)
...
queue
```



A DAG within a DAG = "SUBDAG"

my.dag

```
JOB A A.sub  
SCRIPT POST A generate_B.sh  
SUBDAG EXTERNAL B B.dag  
JOB C C.sub  
PARENT A CHILD B  
PARENT B CHILD C
```



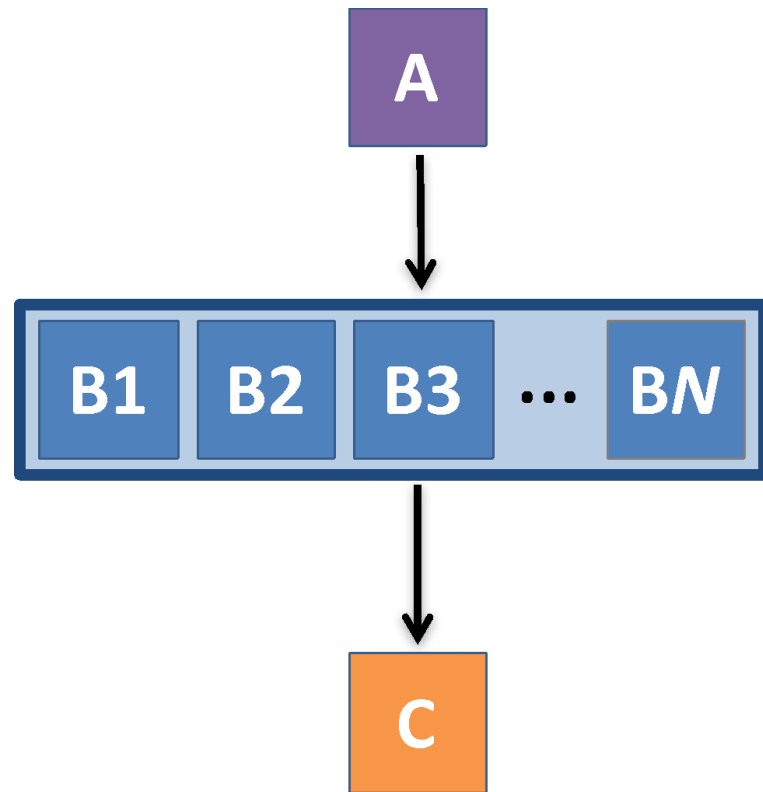
A DAG within a DAG = "SUBDAG"

my.dag

```
JOB A A.sub
SCRIPT POST A generate_B.sh
SUBDAG EXTERNAL B B.dag
JOB C C.sub
PARENT A CHILD B
PARENT B CHILD C
```

B.dag

```
JOB B1 B1.sub
JOB B2 B2.sub
...
JOB BN BN.sub
```

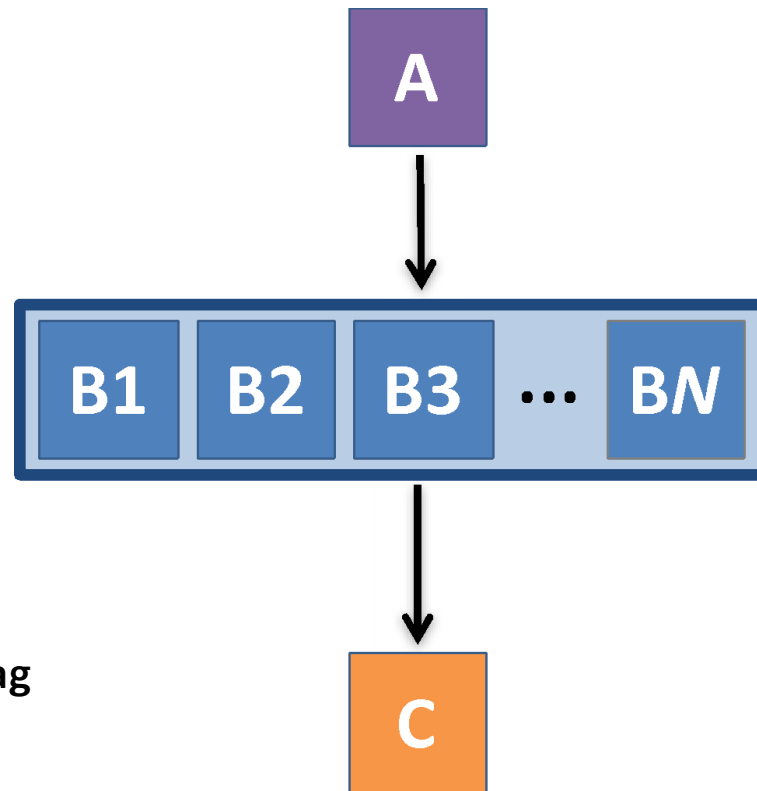


A DAG within a DAG = "SUBDAG"

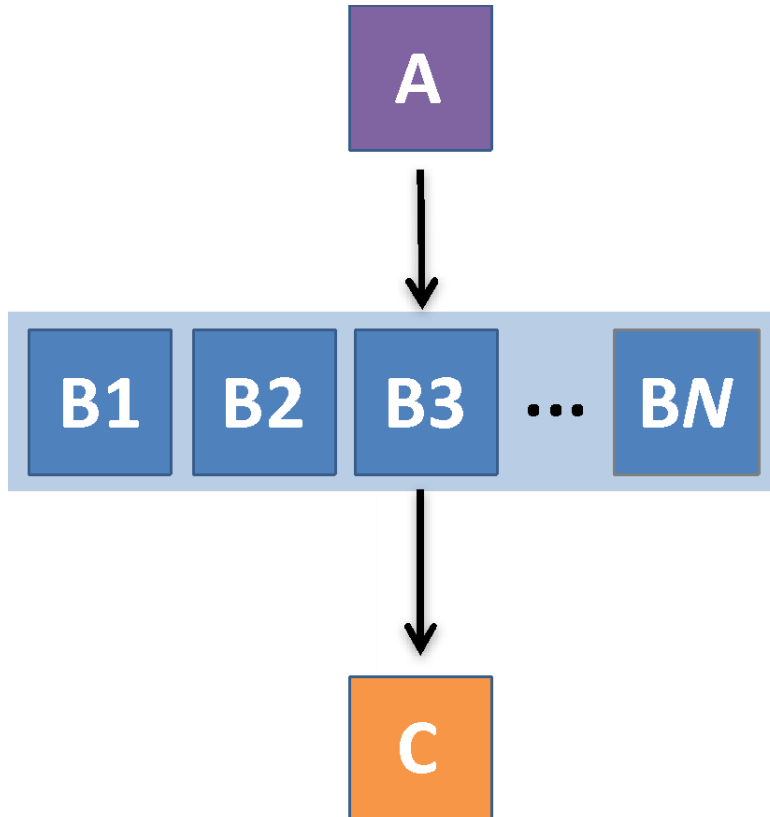
my.dag

```
JOB A A.sub
SCRIPT POST A generate_B.sh
SUBDAG EXTERNAL B B.dag
JOB C C.sub
PARENT A CHILD B
PARENT B CHILD C
```

JOB → condor_submit
SUBDAG EXTERNAL → condor_submit_dag



A DAG within a DAG = "SUBDAG"



A can write the
input files for
subdag **B**!