I Didn't Know DAGMan Could Do That!?

Expanded DAGMan Functionality

By: Cole Bollig

Software Developer for CHTC

European HTCondor Workshop 2024

CHTC HTCondor Software Suite PATh

DAGMan Introductory Material

- Previous Tutorials/Presentations
  - HTCondor Week 2022 DAGMan Introduction Tutorial
  - HTCondor Week 2014 Advance DAGMan Tutorial
  - HTCondor Week 2014 Introductory DAGMan Tutorial
- DAGMan Documentation
  - HTCondor DAGMan Documentation
- Example DAGMan Tutorial
  - https://github.com/OSGConnect/tutorial-dagman-intermediate

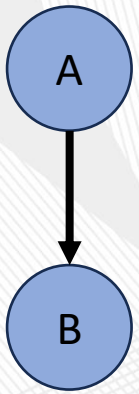
Quick Refresher

- DAGMan is a Directed Acyclic Graph (DAG) Manager that is used to help automate a workflow of jobs.
- A DAG is comprised of Nodes and Dependencies.
- A Job is the core of a DAG Node
- DAGMan makes as much forward progress as possible

simple.dag

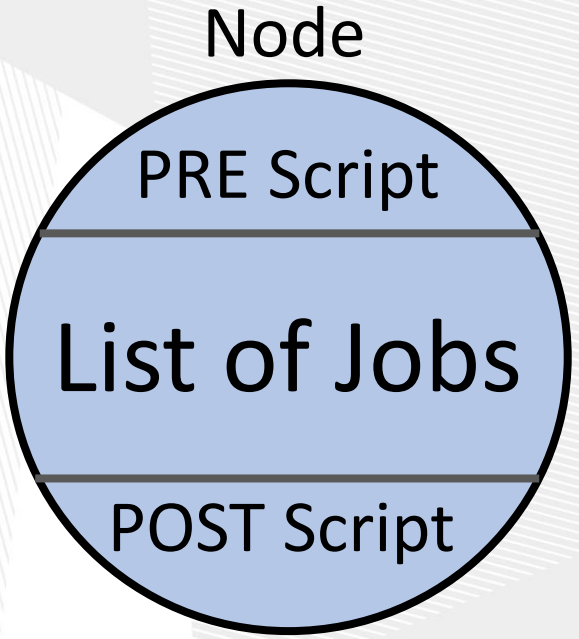
```
JOB A job1.sub
JOB B job2.sub

PARENT A CHILD B
```

Simple DAG visualized



What is a Node?

- A node is comprised of three parts
  1. PRE Script: Runs before placing the job list to an AP
  2. List of Jobs: DAGMan requires all jobs in list to be successful
  3. POST Script: Runs after all associated jobs leave the AP
- All scripts run on the submit host and not the Execution Point (EP).



Important Knowledge

- Submitting a DAG to HTCondor produces an HTCondor scheduler universe job that executes a DAGMan process.

Lots of files produced:

- Informational DAG files
  - *.dagman.out = DAG progress/error output
  - *.nodes.log = Collective job event log (Heart of DAGMan)
  - *.metrics = JSON formatted DAG information
- DAGMan job files
  - *.condor.sub = Submit File
  - *.dagman.log = Job Log
  - *.lib.err = Job Error
  - *.lib.out = Job Output

See a DAG's Status via htcondor dag status

```
colebollig@Coles-MacBook-Pro % htcondor dag status 454
DAG 454 [sample.dag] has been running for 09:13:45
DAG has submitted 8 job(s), of which:
        1 is submitted and waiting for resources.
        1 is running.
        5 have completed.
        1 has failed.
DAG contains 11 node(s) total, of which:
   [#]   4 have completed.
   [=]   4 are running: 1 pre-script, 2 jobs, 1 post-script.
   [!]   2 will never run.
   [!]   1 has failed.
DAG had at least one node fail. Only 72.73% of the DAG can complete.
[##################################==============================!!!!!!!!!!!!!!!!!!!!] DAG is 36.36% complete.
```

Apply Modifiers to All Nodes

- The following DAG commands can be applied to every node in a DAG in one line:
  - ABORT-DAG-ON
  - CATEGORY
  - PRE_SKIP
  - PRIORITY
  - RETRY
  - SCRIPT
  - VARS

Note: Does not apply to Service and Final Nodes.

ALL_NODES Keyword

sample.dag

```
JOB TEST-0 job0.sub
JOB TEST-1 job1.sub
…
JOB TEST-998 job998.sub
JOB TEST-999 job999.sub

SCRIPT POST ALL_NODES check.sh
```

DAGMan ALL_NODES Documentation

Pass DAG/Node Information to Scripts

Inform DAGMan of DAG/Node information to pass as arguments to a node script.

verify-success.dag

```
JOB A A.sub
SCRIPT A POST check_exit.sh $NODE $RETRY $RETURN
```

Note: Some Script macros only apply to the POST Script

- DAG Information
  - Counts of nodes per status (Done, Failed, etc.)
  - DAG Status
  - DAGManJobId
- Node Information
  - Node name
  - Retries (current retry # and the max)
  - The job ID
  - Node Success/Failure up till this point
  - Job exit codes
  - Number of associated jobs
  - Return value of the node's PRE Script

DAGMan Script Macro Documentation

Capture Script Output

- Specify a file to capture the STDOUT and/or the STDERR of a node's script
- Multiple scripts can write to the same file because all output is captured by DAGMan and written in a single write
- Debug file includes divider line containing information about the script execution (including the exact command DAGMan executed)

debug.dag

```
JOB A A.sub

SCRIPT DEBUG script.out ALL POST A check.sh $NODE
```

script.out

```
*** Node=A Type=POST Status=0 Completion=1726165734 Cmd='check.sh A'
Args ['check.sh', 'A']
Verifying outputs exist…
A-analysis.txt exists
A-simulation.txt exists
A-aggregate.txt exists
A-quatum.txt exists
All files exists!
```

DAGMan Script Debug Documenation

Skip a Node Based on the PRE Script

- Mark a node as done based on the return code of a PRE Script.
  - Use the PRE_SKIP command
  - Don’t submit any jobs or execute the POST Script
  - Node is successful
- Useful for skipping nodes when re-running a DAG

DAGMan PRE_SKIP Command Documentation

skip-node.dag

```
JOB A simulation.sub
JOB B analysis.sub
JOB C aggregation.sub

SCRIPT A PRE check_simulation_ran.sh
PRE_SKIP A 2
```

check_simulation_ran.sh

```
#!/bin/bash
if [ -e “complex-data.sim” ]; then
   exit 2
else
    exit 0
fi
```

Using DAG VARS in IF Conditionals

- The use the VARS command is very common for sharing a submit description (job template)
  - Use PREPEND keyword to add the macro(s) before description parsing
  - Use APPEND keyword to add the macro(s) after description parsing
  - No PREPEND or APPEND specified will add VARS according to DAGMAN_DEFAULT_APPEND_VARS

DAGMan PREPEND/APPEND VARS Documentation

skip-node.dag

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

VARS A PREPEND src="./work/source"
```

generic.sub

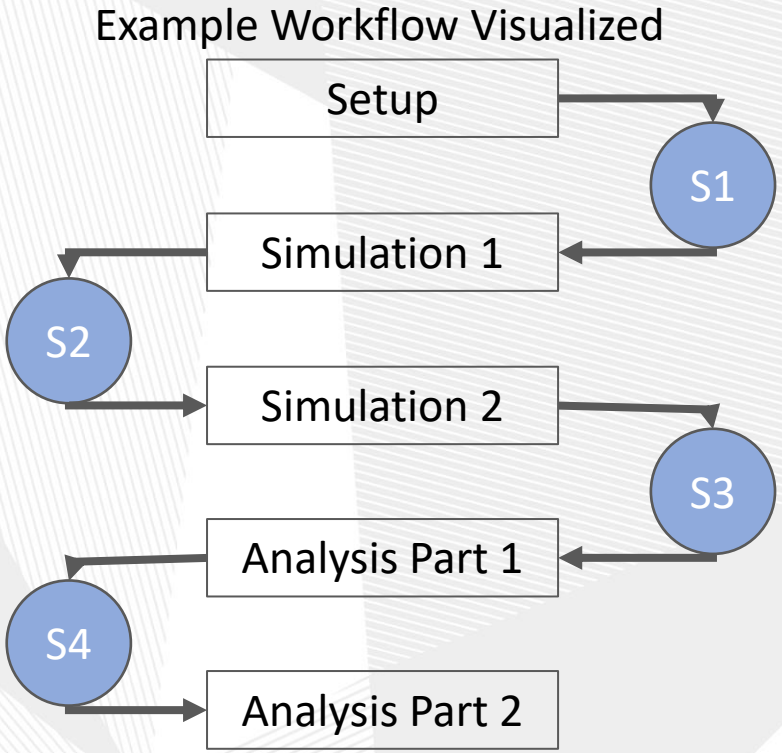
```
executable = ./physics.sh
arguments = -a heavy -l –src $(SOURCE)
…
if defined src
   SOURCE = $(src)
else
   SOURCE = /home/default/source
endif
queue
```

Save a DAGs Progress

- Saves the current progress of the DAG comparable to a video game save file
  - File is similar too a rescue file
  - Written the first time a specified node runs

sample.dag

```
…
SAVE_POINT_FILE S1
SAVE_POINT_FILE S2 post_simulation1.save
SAVE_POINT_FILE S3 ./post_simulation2.save
SAVE_POINT_FILE S4 ../../foo/mid_analysis.save
…
```



DAGMan Save Point File Documentation

Save a DAGs Progress cont.

- Where are the save files written?
  - Nodes S1 & S2 write their save files to a new subdirectory called **save_files** in the DAG's working directory.
  - Nodes S3 & S4 write their save files to the specified path relative to the DAG's working directory.
- S1 save will be written to a file named S1-sample.dag.save

saved.dag

```
…
SAVE_POINT_FILE S1
SAVE_POINT_FILE S2 post_simulation1.save
SAVE_POINT_FILE S3 ./post_simulation2.save
SAVE_POINT_FILE S4 ../../foo/mid_analysis.save
…
```

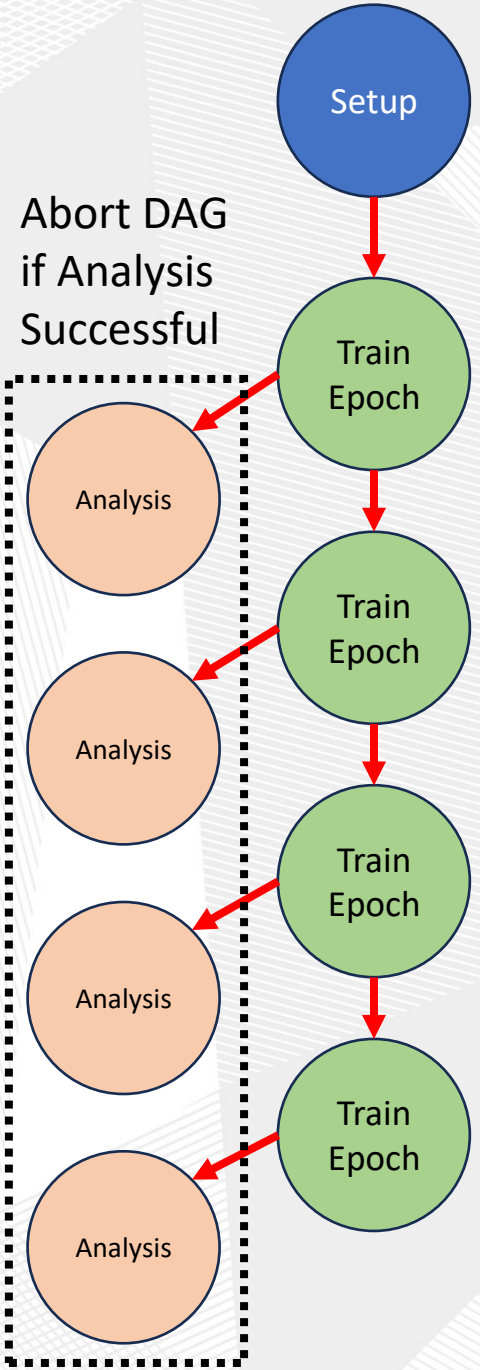
```
condor_submit_dag –load_save [save_file] saved.dag
```

If given a path, then **condor_submit_dag** will use that path to look for the save file. Otherwise DAGMan looks in the **save_files** sub-directory for the save files.

Stop a DAG Early

- ABORT-DAG-ON Command
  - Notifies DAG to write a rescue file and abort the workflow early
  - Specify an exit code that triggers the DAG abort
    - Checked with each part of the node (PRE/JOB/POST)
  - Specify DAG exit code (Success/Failure)
  - FINAL node is still run

DAGMan ABORT-DAG-ON Command



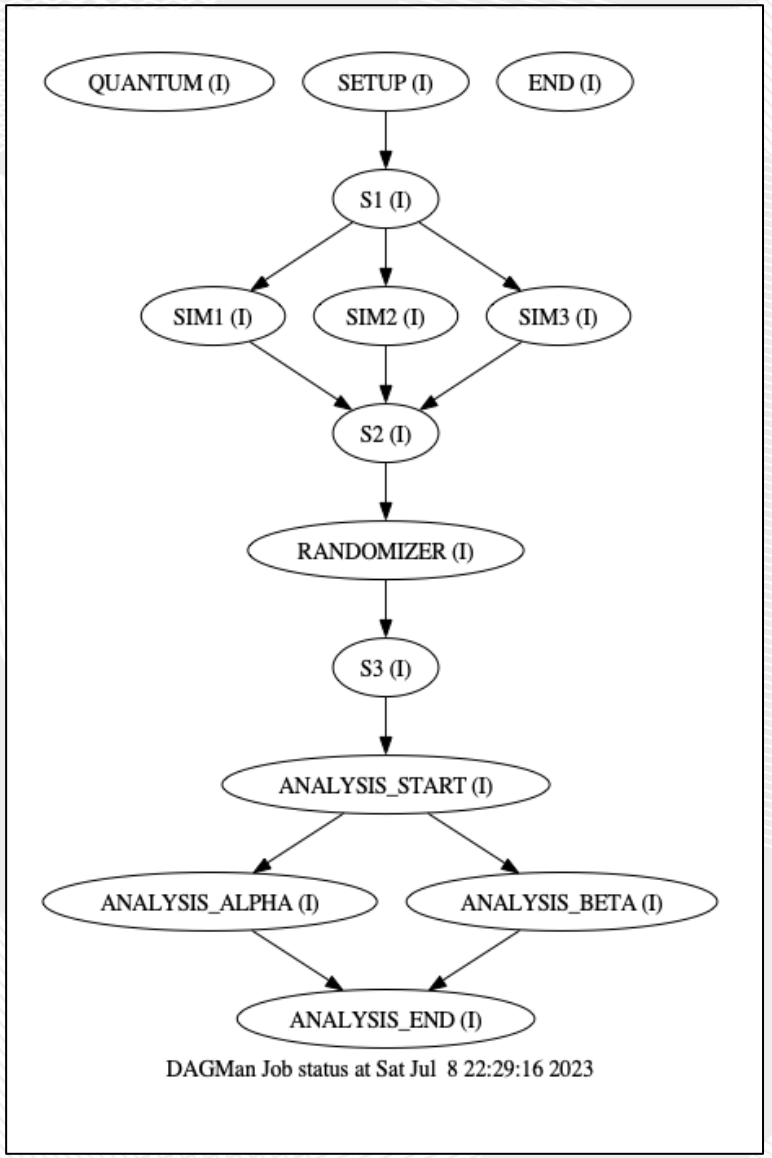
Visualize a DAG

- DAGMan can produce a DOT file to easily help visualize a DAG utilizing the AT&T Research Labs *graphviz* package

sample.dag

```
...
DOT dag.dot
...
```

```
dot -Tps dag.dot -o dag.ps
```

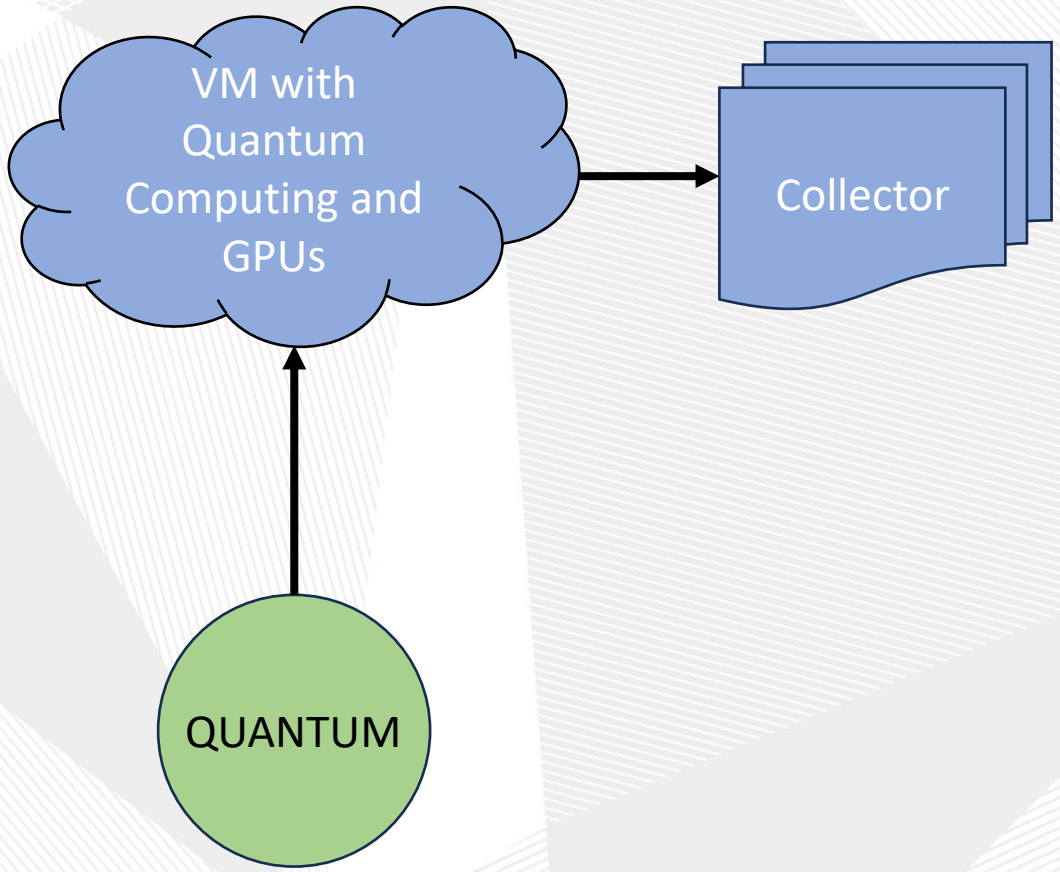


[DAGMan Dot Files Documentation]

DAGMan has special Node Types

Provisioner Node

- Good for setting up unique resources to be used by nodes in a DAG
- Always starts prior to other nodes
- Runs for a set amount of time defined in the job itself
- Can only have one provisioner node



simple.dag

```
JOB A job1.sub
JOB B job2.sub

PROVISIONER QUANTUM cloud.sub
...
```

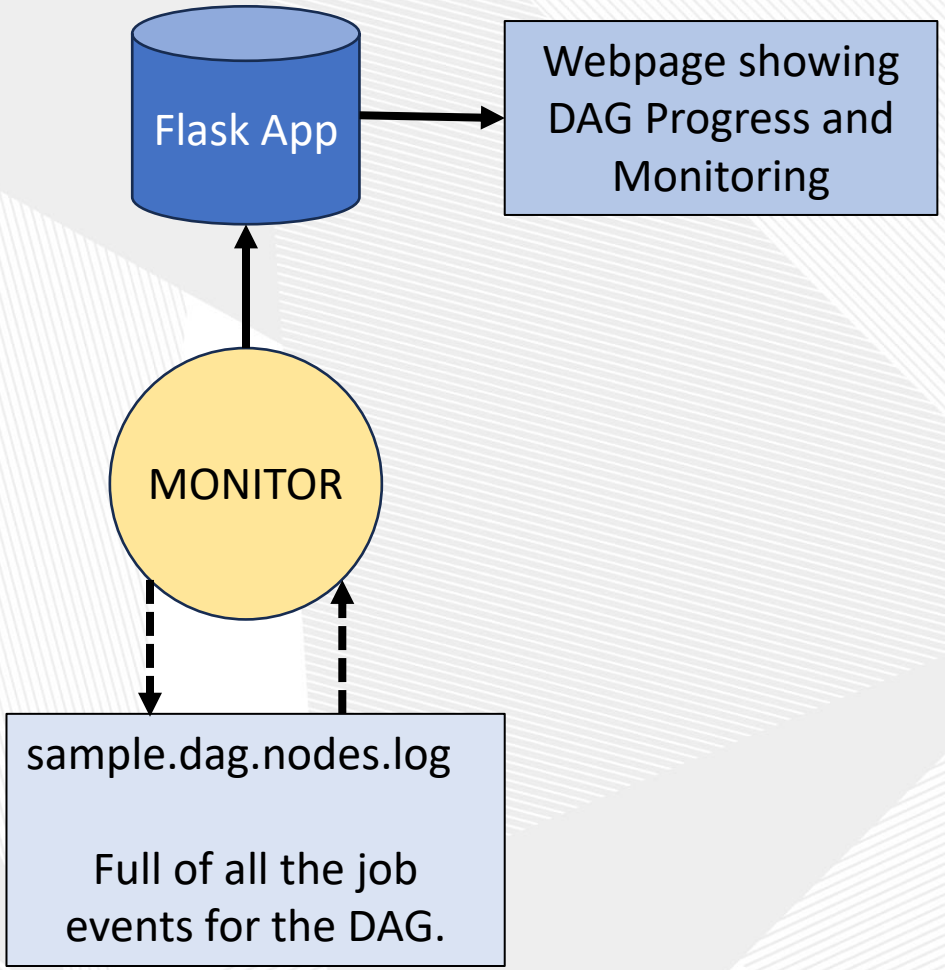
DAGMan Provisioner Node

Service Node

- The ‘sidecar node’ that runs along side the DAG and perform tasks
- Begin running at the beginning of the DAG but isn’t guaranteed to run before other nodes.
- Best effort. If the submit fails, the DAG will carry on.
- Is managed by DAGMan such that DAGMan will remove all service nodes before exiting

sample.dag

```
JOB A job1.sub
JOB B job2.sub
…
JOB Y job3.sub
JOB Z job4.sub

SERVICE  MONITOR flask.sub
…
```



DAGMan Service Node

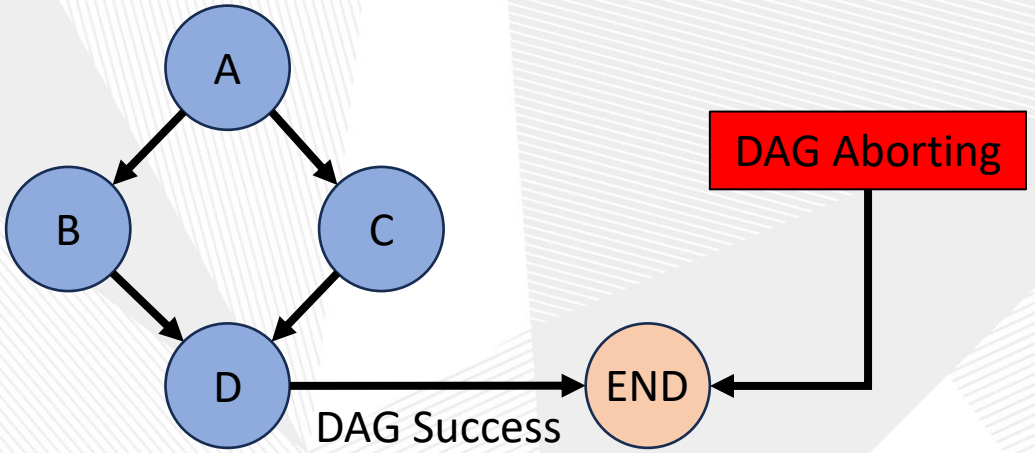
Final Node

- Always the last node to run whether the DAG has aborted or completed successfully
- Good for cleanup and verifying output of previous node
- Can only be one final node in a DAG

diamond.dag

```
JOB A job1.sub
JOB B job2.sub
JOB C job3.sub
JOB D job4.sub

FINAL END cleanup.sub
…
```

Diamond DAG visualized



[DAGMan Final Node]

A DAG can be comprised of DAGs

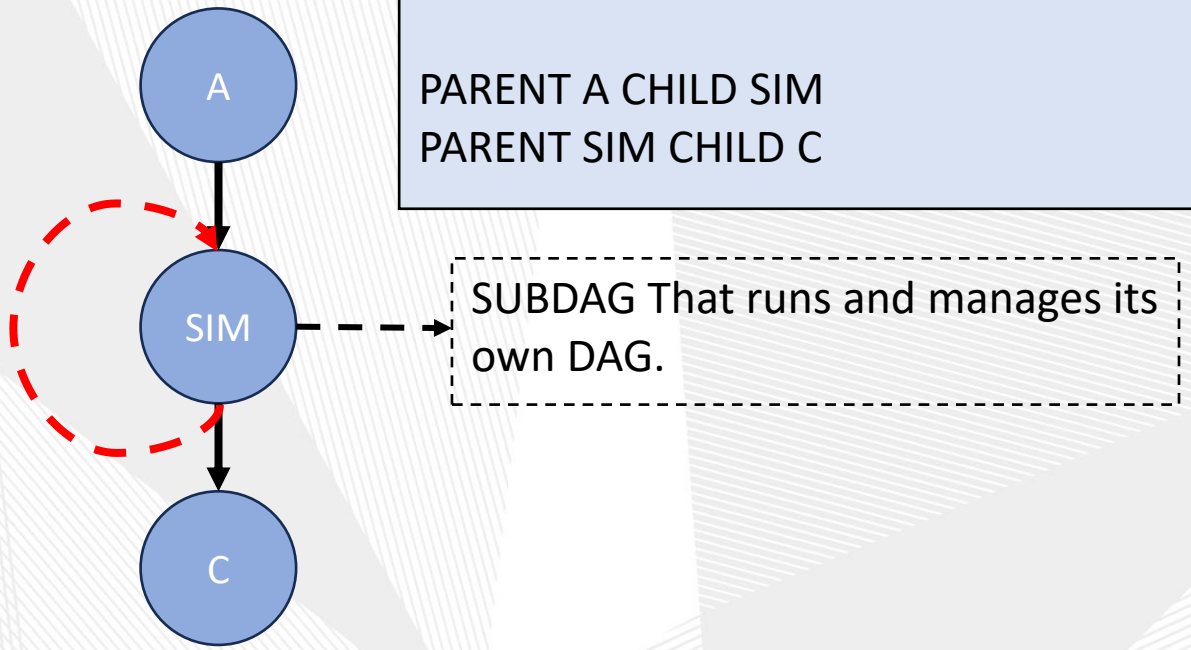
DAGMan Composing DAG of DAG's Documentation

SUBDAGs

- To the parent DAG it is just a single node
  - Can use RETRY
  - Can add PRE & POST Scripts
- Submits as another DAG to the AP such that has its own DAGMan process and output files.
- DAG file and nodes don't need to exist at submission time of parent DAG
- Good for running sub-workflows where the number of jobs is not predefined

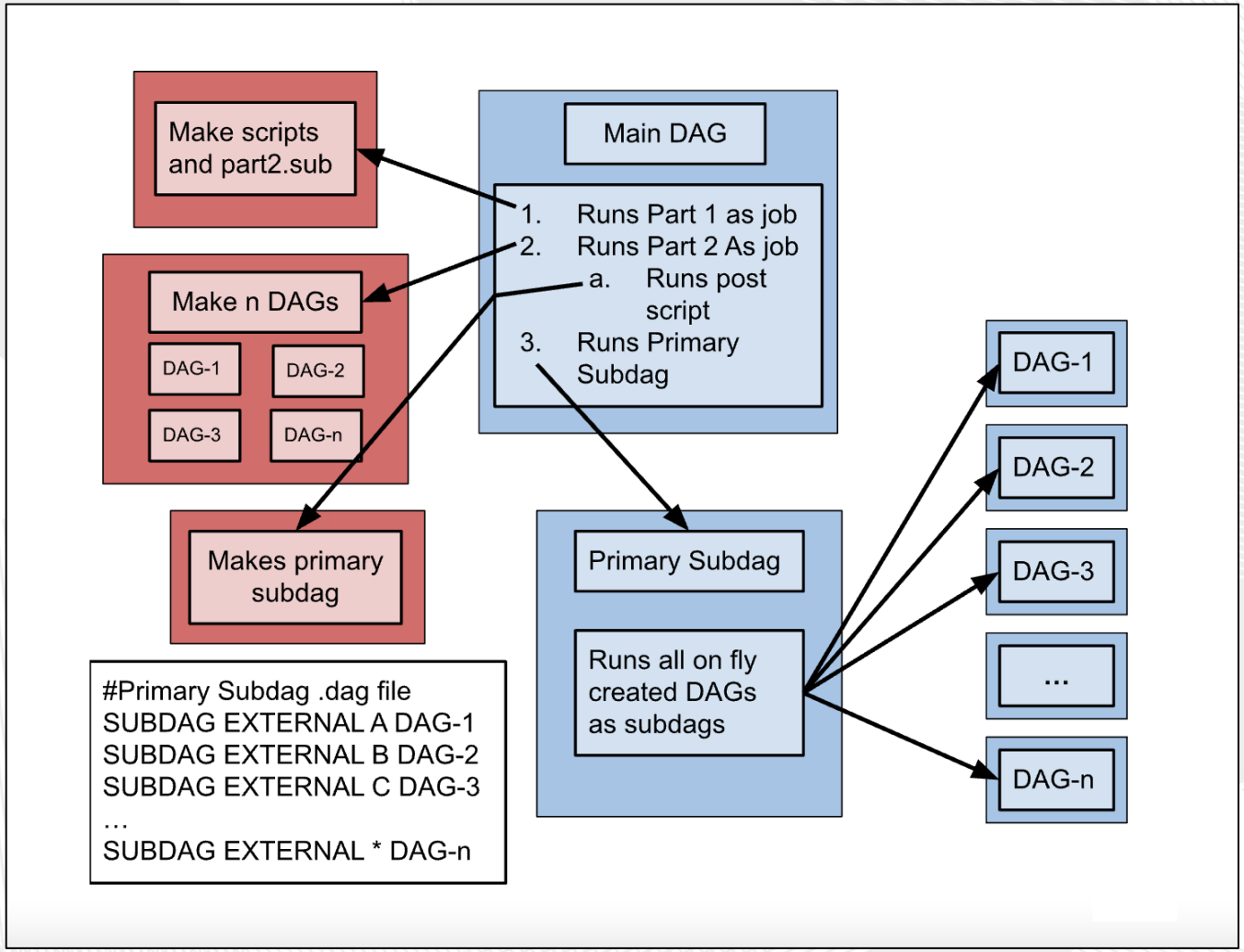
sample.dag

```
JOB A job.sub
SUBDAG EXTERNAL SIM simulation.dag
JOB C job.sub

SCRIPT POST SIM …
RETRY 10 SIM

PARENT A CHILD SIM
PARENT SIM CHILD C
```



Example: DAG that runs N SUBDAGs

This is an example diagram to show a user how to set up a DAG that creates and unknown number of DAGs and subsequently runs them.

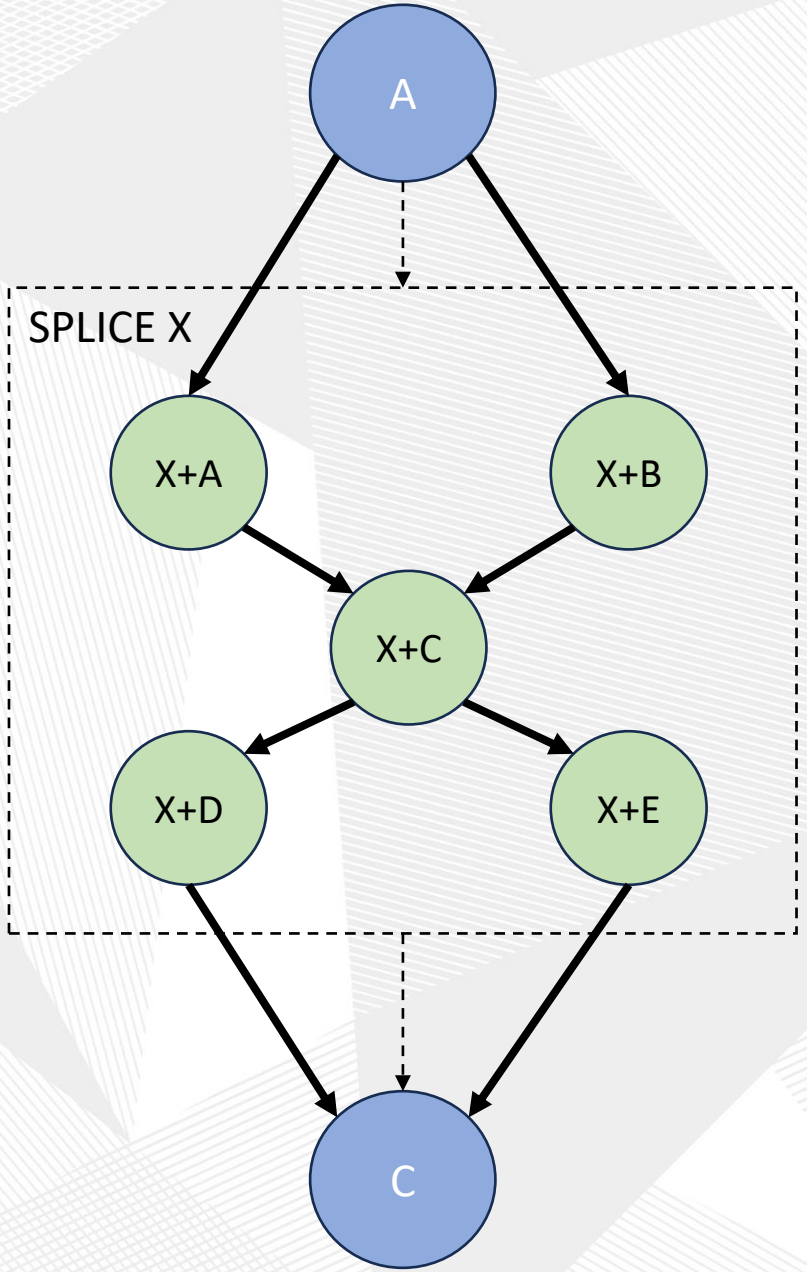


DAG SPLICE

- All spliced DAGs have their nodes merged into the parent DAG
- Allows easy reusability
- Low strain on the Access Point (AP)
- All spliced DAG files must exist at submit time
- Pre and Post scripts cannot run on splices as a whole
- Splices can not use the RETRY capability

sample.dag

```
JOB A job.sub
SPLICE X cross.dag
JOB C job.sub

PARENT A CHILD X
PARENT X CHILD C
```



Questions?

PRE Script Example

diamond.dag

```
JOB A job1.sub
JOB B job2.sub
JOB C job3.sub
JOB D job4.sub

SCRIPT PRE A verify.sh

PARENT A CHILD B C
PARENT B C CHILD D
```

verify.sh

Super cool script that verifies all input files for job are at least 10mb.

Input files < 10mb → Node A Pre Script fails making the node as a whole fail.

Input files >= 10mb → Node A Pre Script succeeds, and the Node A job gets submitted.

Another possibility would be to have the script manipulate Input Files (Rename, Move, Condense)

POST Script Example

diamond.dag

```
JOB A job1.sub
JOB B job2.sub
JOB C job3.sub
JOB D job4.sub

SCRIPT POST C loop.sh $RETURN $RETRY
RETRY C 5 UNLESS-EXIT 2

PARENT A CHILD B C
PARENT B C CHILD D
```

loop.sh

```
#Takes job exit code &
#node retry attempt

if (job exit == 0)
    if (retry >= 4) { exit 0 }
    else { exit 1 }
else
     exit 2
```

- Causes Node C loop and run 5 times.
- Looping behavior can be added to SUBDAG workflows too.

Other possibilities for Post Scripts:

- Verify output
- Fake a node success even though node job failed
- Produce a file that is to be used later by the DAG (job submit file, script, a subdag)