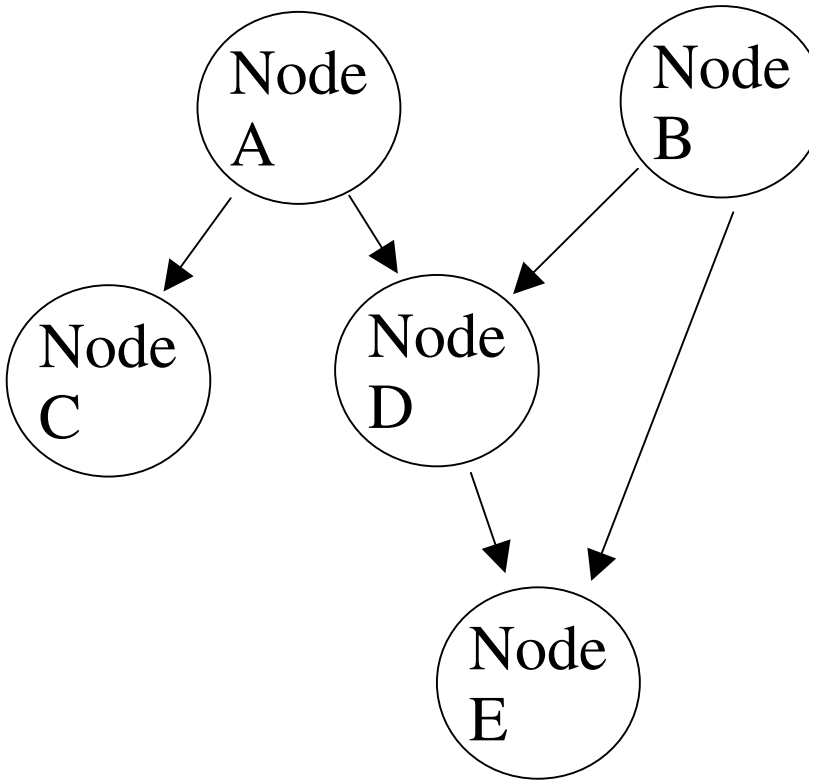
# Job DAGs in WP1

Francesco Giacomini – INFN CNAF

# What is a DAG?

- Directed Acyclic Graph
- Each node represents a job
- Each edge represents a (temporal) dependency between two nodes
  - e.g. NodeC starts only after NodeA has finished



# What is a DAG Node?

- A node represents a job
  - job description plus some extra information
- ClassAd representation
  - Predefined attributes: `node_type, node_retry_count, description_file, description_ad, pre, pre_args, post, post_args`
    - pre and post scripts are not currently available to users because of security reasons
  - Any other attribute of any type

```
NodeA = [
  node_type = "edg-jdl";
  node_retry_count = 3;
  description_ad = [];
  any_other_attribute = { "one", 2 }
]
```

# DAG Node Attributes

- `node_type`
  - string, mandatory (but it can be inherited)
  - Type of the node. No check is done on the contents. If specified, it overrides the default one
- `node_retry_count`
  - integer, optional
  - Number of retries for the node. Its value must be greater than or equal to zero. If specified, it overrides the default one. If neither the default one nor the node specific one are specified the default is zero, i.e. in case of node failure no retry is performed

# DAG Node Attributes (cont.)

- `description_file`
  - string, optional
  - Name of the file containing the node description. No check is performed on the value. It cannot be inherited. This attribute is optional, but, if missing, `description_ad` must be present. `description_file` and `description_ad` cannot coexist
- `description_ad`
  - ClassAd, optional
  - ClassAd describing the node. No check is performed on the value. It cannot be inherited. This attributed is optional, but, if missing, `description_file` must be present. `description_file` and `description_ad` cannot coexist

# DAG Node Attributes (cont.)

- `pre`
  - string, optional
  - name of the executable to be run before the node's job. No check is done on the value
- `pre_args`
  - string, optional
  - Arguments to be passed to the pre executable. All the arguments are wrapped in a single string value. No check is done on the value. `pre_args` can be present only if `pre` is also present

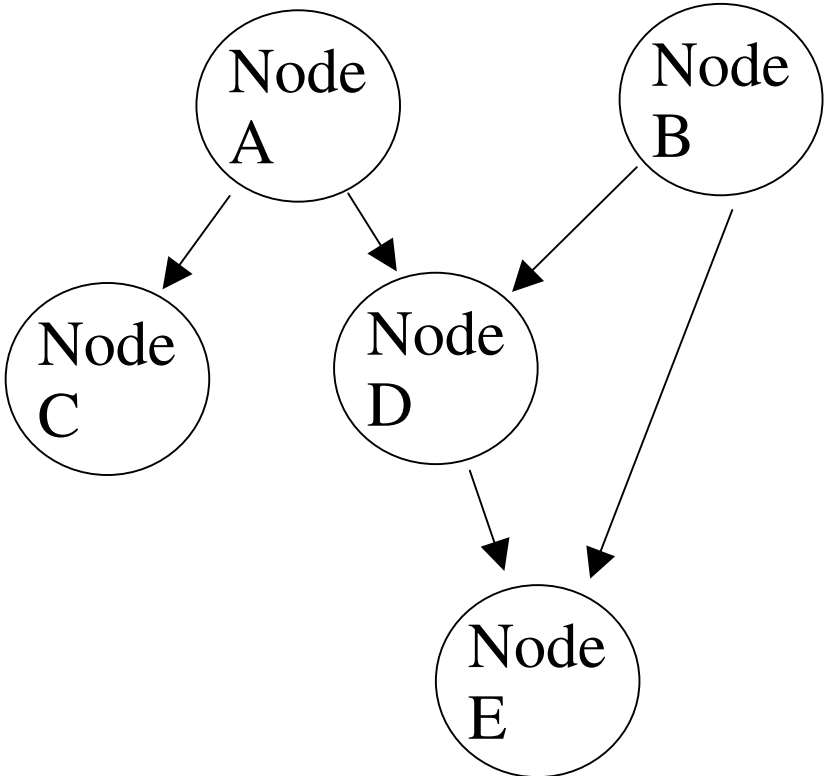
# DAG Node Attributes (cont.)

- `post`
  - String, optional
  - name of the executable to be run after the node's job. No check is done on the value
- `post_args`
  - string, optional
  - Arguments to be passed to the post executable. All the arguments are wrapped in a single string value. No check is done on the value. `post_args` can be present only if `post` is also present

# What is a Dependency?

- A dependency represents a constraint on the time a node can be executed
  - Limited scope, it may be extended in the future
- Dependencies are represented as “expression lists” in the ClassAd language

```
dependencies = {
   {NodeA, {NodeC, NodeD}},
   {NodeB, NodeD},
   {{NodeB, NodeD}, NodeE}
}
```



# How Dependencies Are Specified

- ***Each element in the list must comply with one of the following formats:***
  - ***{nodeA, nodeB}***

    ***nodeB** depends on **nodeA***
  - ***{node, nodes}***

    *Each node in **nodes** depends on **node***
  - ***{nodes, node}***

    ***node** depends on each node in **nodes***

***node** must refer to an existing node (AttributeReference)*

***nodes** can be empty*

# DAG Ad Attributes

- `type`
  - string, mandatory
  - type of job. Must be “dag”
- `max_running_nodes`
  - integer, optional
  - hint about the maximum number of nodes executing concurrently. Its value must be greater than 0
- `node_retry_count`
  - Integer, optional
  - default number of retries for a node. A node can override it.

# DAG Ad Attributes (cont.)

- `node_type`
  - string, optional
  - default type of a node. A node can override it
- `nodes`
  - ClassAd, mandatory
  - Set of nodes in the dag. Additional constraints:
    - Each nested classad is considered a node specification
    - It must contain an ExprList, named `dependencies`, representing the dependencies
- Any other attribute of any type

# Putting It All Together

```
[
  type = "dag";
  max_running_nodes = 10;
  node_type = "edg-jdl";
  node_retry_count = 3;
  nodes = [
    NodeA = [ /* ... */ ];
    NodeB = [ /* ... */ ];
    NodeC = [ /* ... */ ];
    NodeD = [ /* ... */ ];
    NodeE = [ /* ... */ ];
    dependencies = {
      {NodeA, {NodeC, NodeD}},
      {NodeB, NodeD},
      {{NodeB, NodeD}, NodeE}
    }
    something_else = "???"; /* not a ClassAd! */
  ];
  anything_else = {1, "two", []}
]
```

# How To Write a DAG Ad

- Using your favourite editor
- Using our graphical editor
- Using the DAGAd API (C++)
  - For developers
  - Not EDG-specific; e.g.:
    - No checks on the job JDL
    - No knowledge of job id
  - Utility library for EDG issues built on top of the generic API

# How To Manage a DAG

- Current UI commands extended to manage a DAG, which is a job on its own
- DAG submission:
  - Check the DAG, its nodes and the corresponding jobs
  - Resubmission (“rescue dag”) not supported
- DAG cancellation:
  - Cancel the DAG and its subjobs
  - A subjob can be cancelled independently, which indirectly makes the whole DAG fail

# How To Manage a DAG (cont.)

- DAG logging info:
  - Get the events for the DAG job only
- DAG status
  - DAG status and a summary of subjob statuses
  - subjobs are “queriable” independently
- DAG OSB retrieval:
  - Retrieve all the subjob OSB and purge them
  - Subjob OSBs are retrievable independently (w/o purging)

# Sandbox Management

- A dependent job may need to access a file produced by one of its parents
- The idea is to add in the ISB of a job a reference to a file in the OSB of a parent job
- The proposal is to use an AttributeReference in the ISB specification:
  - { “file1.dat”, NodeA.file.out,}
  - Valid syntax, but not valid semantics (i.e. not a reference to an existing attribute)
- Should there be an ISB/OSB for the whole DAG? What does it mean?

# Implementation

- Uses DAGMan, from the Condor project
- A DAG ad is converted to the DAGMan format
- For every DAG passed to CondorG a separate DAGMan process is run