HTCSS Vocabulary, Architecture, and User View

European HTCondor Workshop 2023

Todd Tannenbaum
Center for High Throughput Computing
University of Wisconsin-Madison

Center for High Throughput Computing

HTCondor Software Suite

PATh Partnership to Advance Throughput Computing

HTC = High Throughput Computing

HTCSS = HTCondor Software Suite

What is the HTCSS, what does it do?

HTCSS provides a distributed high-throughput batch computing environment

- Manages workflows / sets of jobs for researchers
- Federates and supervises computing capacity
- Matches the capacity to workflows
- Distributed, highly available

What is the HTCSS, what does it do?

HTCSS provides a distributed high-throughput batch computing environment

- Manages workflows / sets of jobs for researchers
- Federates and supervises computing capacity
- Matches the capacity to workflows
- Distributed, highly available

HTCondor Suite Components

- Access Point (**AP**)
- Directed Acyclic Graph Manager (**DAGMan**)
- Execution Point (**EP**)
- Central Manager (**CM**)
- Compute Entrypoint (**CE**)

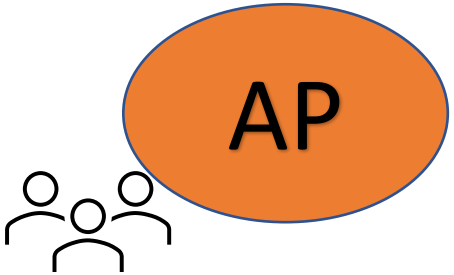
What is the HTCSS, what does it do?

HTCSS provides a distributed high-throughput batch computing environment

- Manages workflows / sets of jobs for researchers
- Federates and supervises computing capacity
- Matches the capacity to workflows
- Distributed, highly available

HTCondor Suite Components

- Access Point (**AP**)
- Directed Acyclic Graph Manager (**DAGMan**)
- Execution Point (**EP**)
- Central Manager (**CM**)
- Compute Entrypoint (**CE**)



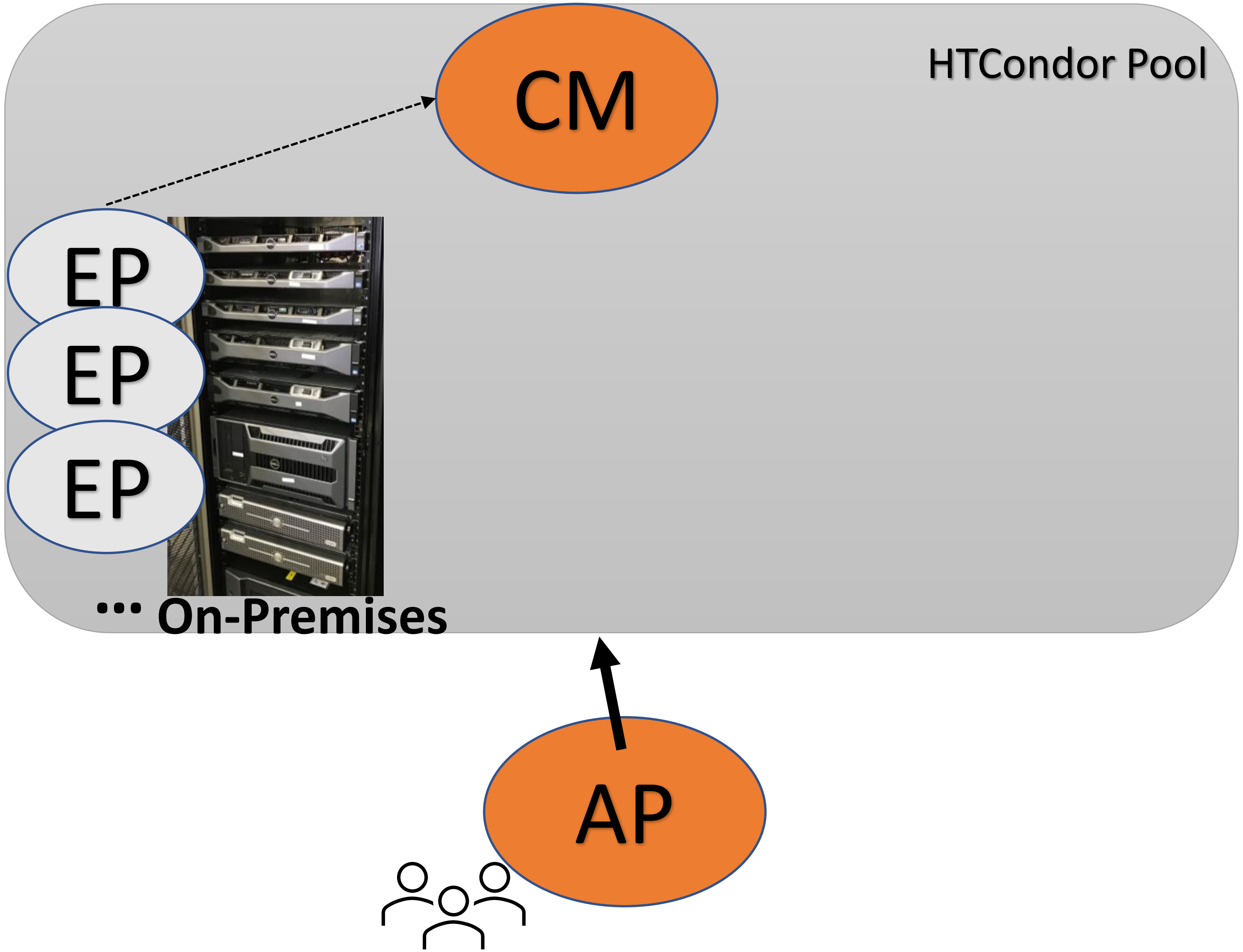
What is the HTCSS, what does it do?

HTCSS provides a distributed high-throughput batch computing environment

- Manages workflows / sets of jobs for researchers
- Federates and supervises computing capacity
- Matches the capacity to workflows
- Distributed, highly available

HTCondor Suite Components

- Access Point (**AP**)
- Directed Acyclic Graph Manager (**DAGMan**)
- Execution Point (**EP**)
- Central Manager (**CM**)
- Compute Entrypoint (**CE**)



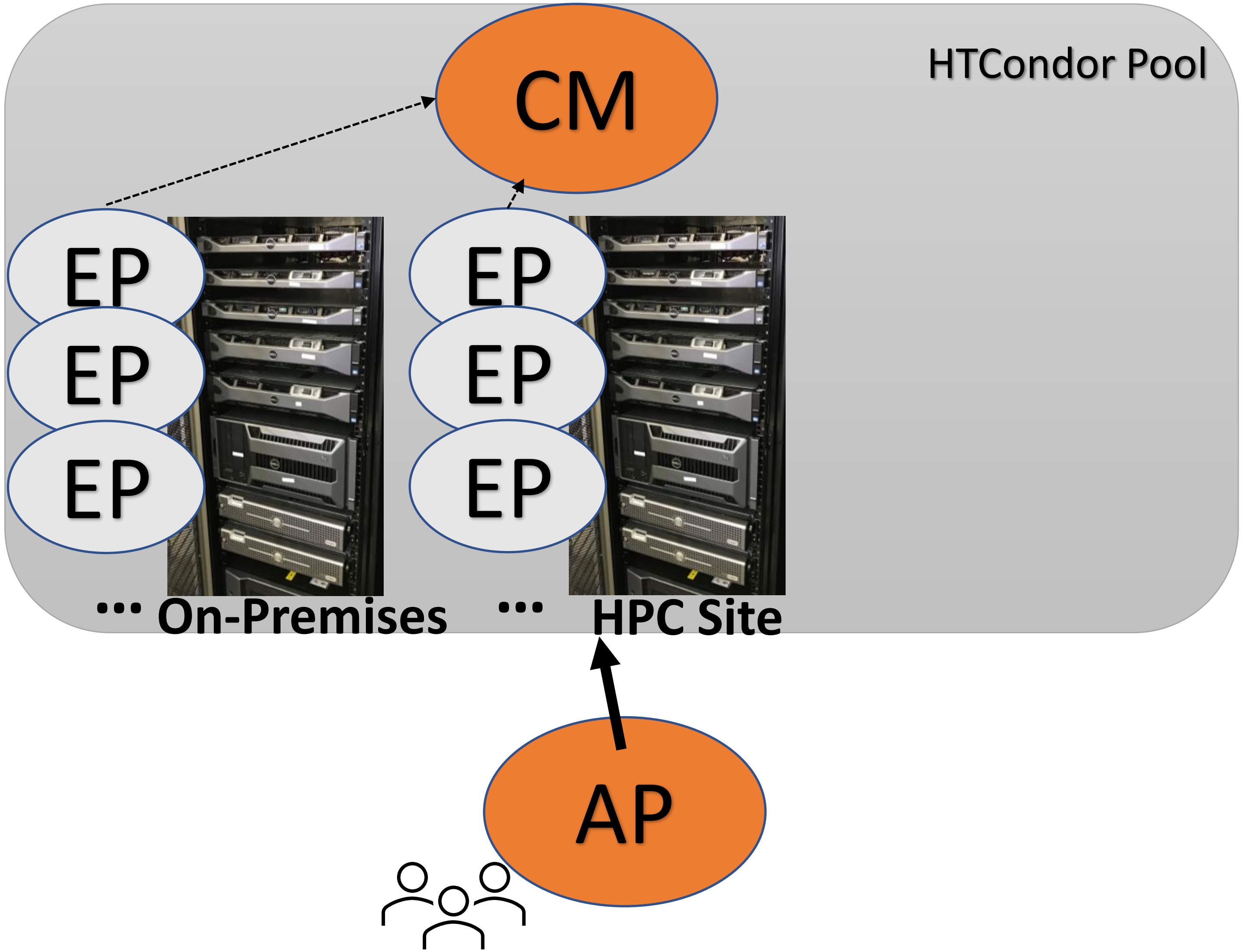
What is the HTCSS, what does it do?

HTCSS provides a distributed high-throughput batch computing environment

- Manages workflows / sets of jobs for researchers
- Federates and supervises computing capacity
- Matches the capacity to workflows
- Distributed, highly available

HTCondor Suite Components

- Access Point (**AP**)
- Directed Acyclic Graph Manager (**DAGMan**)
- Execution Point (**EP**)
- Central Manager (**CM**)
- Compute Entrypoint (**CE**)



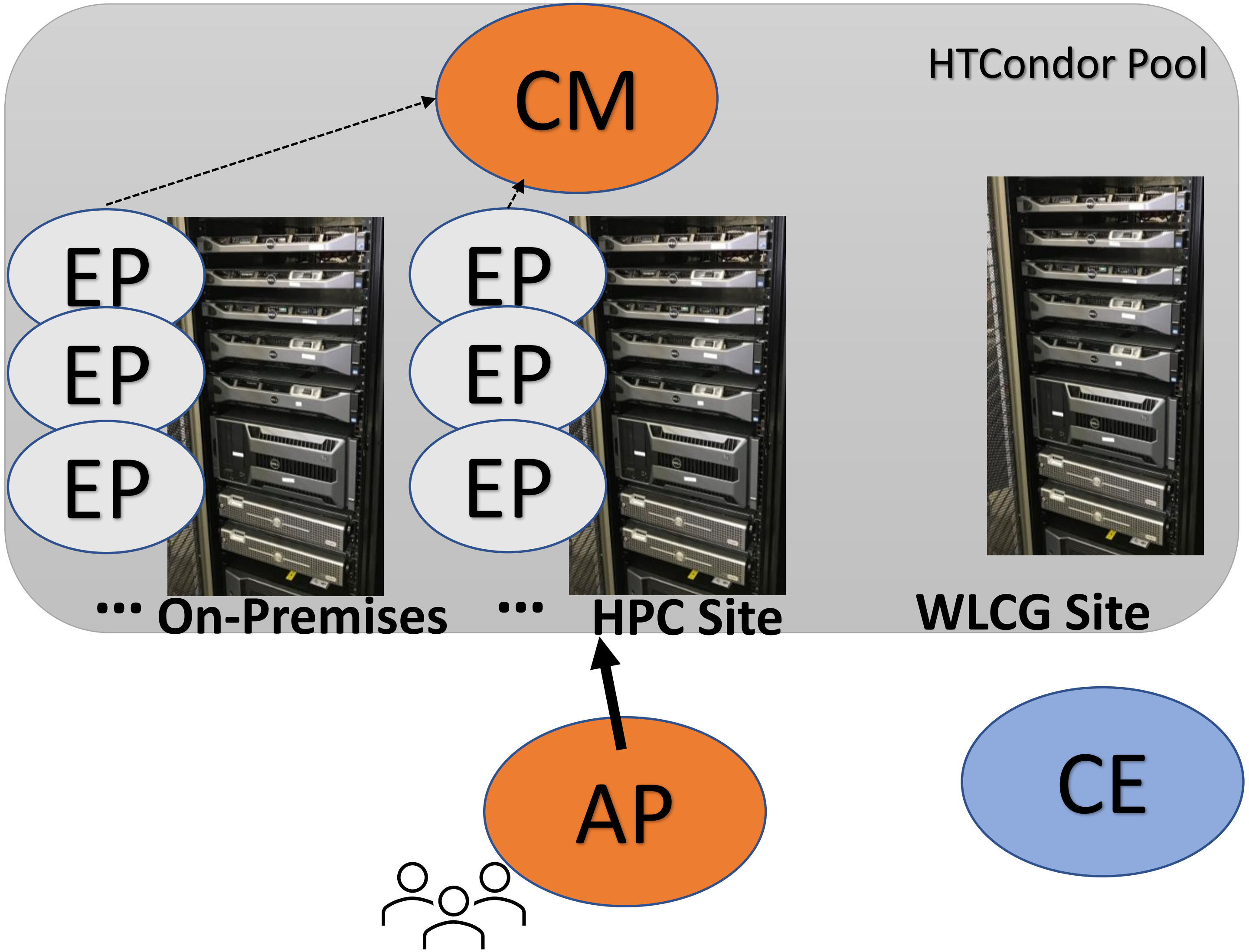
What is the HTCSS, what does it do?

HTCSS provides a distributed high-throughput batch computing environment

- Manages workflows / sets of jobs for researchers
- Federates and supervises computing capacity
- Matches the capacity to workflows
- Distributed, highly available

HTCondor Suite Components

- Access Point (**AP**)
- Directed Acyclic Graph Manager (**DAGMan**)
- Execution Point (**EP**)
- Central Manager (**CM**)
- Compute Entrypoint (**CE**)



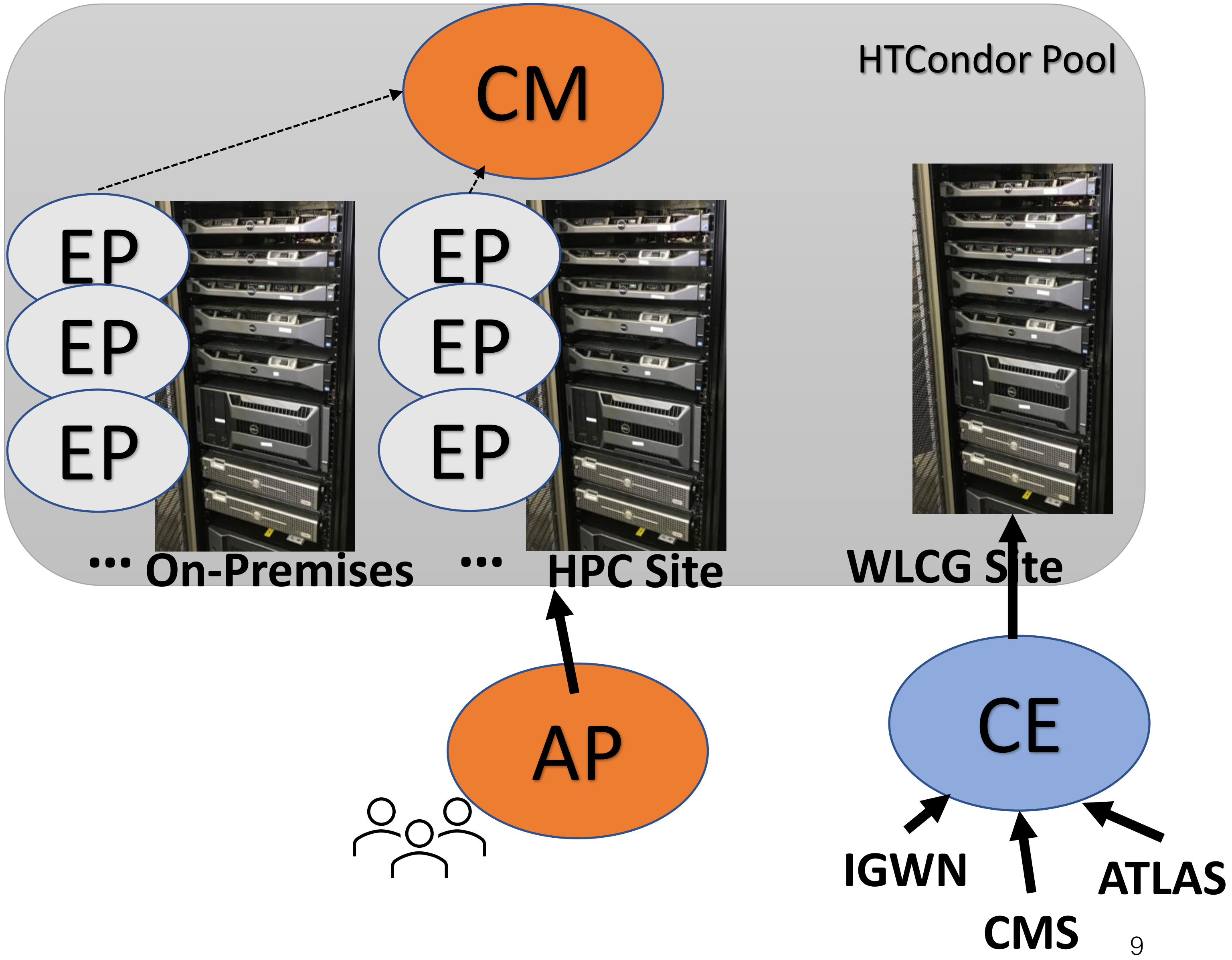
What is the HTCSS, what does it do?

HTCSS provides a distributed high-throughput batch computing environment

- Manages workflows / sets of jobs for researchers
- Federates and supervises computing capacity
- Matches the capacity to workflows
- Distributed, highly available

HTCondor Suite Components

- Access Point (**AP**)
- Directed Acyclic Graph Manager (**DAGMan**)
- Execution Point (**EP**)
- Central Manager (**CM**)
- Compute Entrypoint (**CE**)



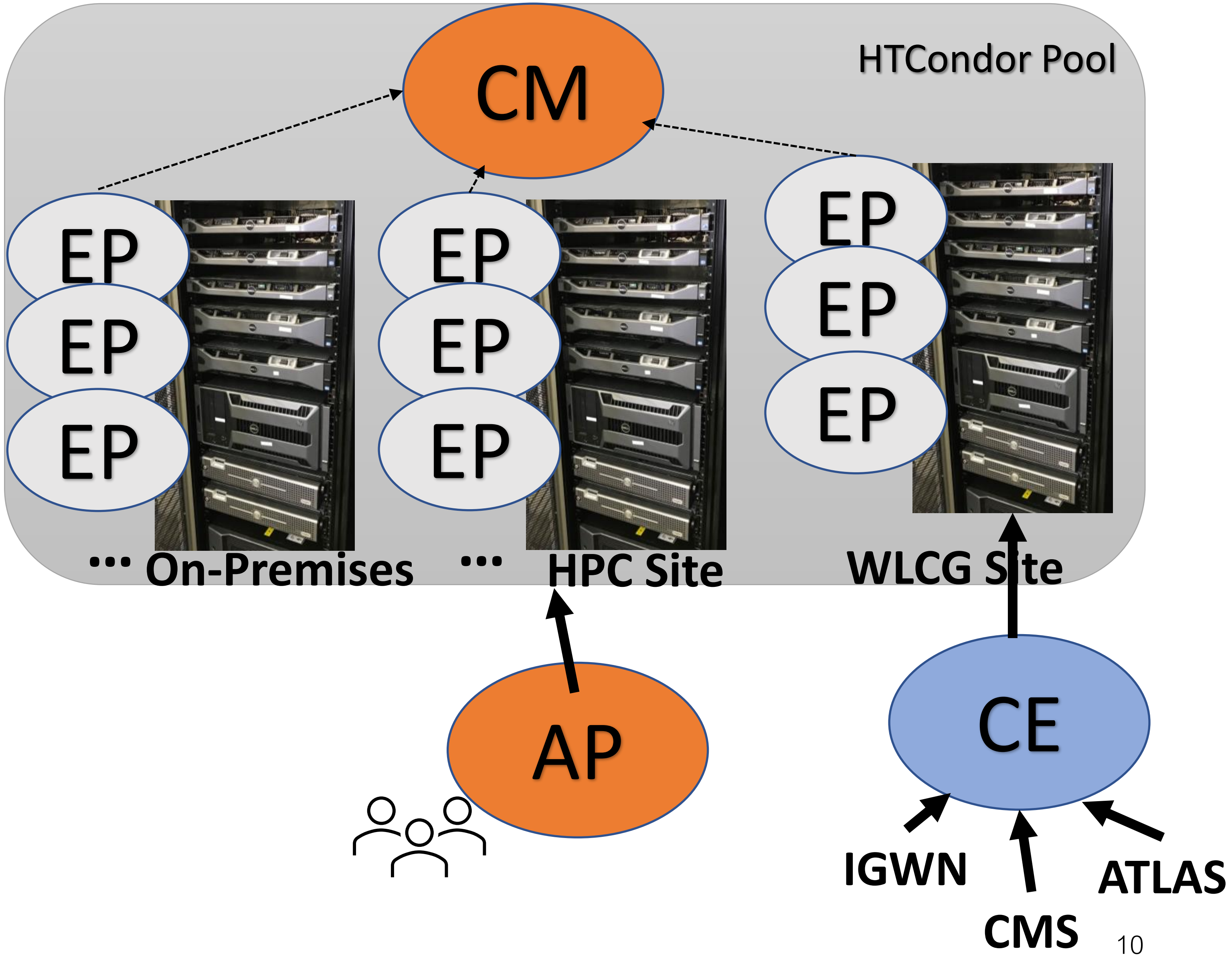
What is the HTCSS, what does it do?

HTCSS provides a distributed high-throughput batch computing environment

- Manages workflows / sets of jobs for researchers
- Federates and supervises computing capacity
- Matches the capacity to workflows
- Distributed, highly available

HTCondor Suite Components

- Access Point (**AP**)
- Directed Acyclic Graph Manager (**DAGMan**)
- Execution Point (**EP**)
- Central Manager (**CM**)
- Compute Entrypoint (**CE**)



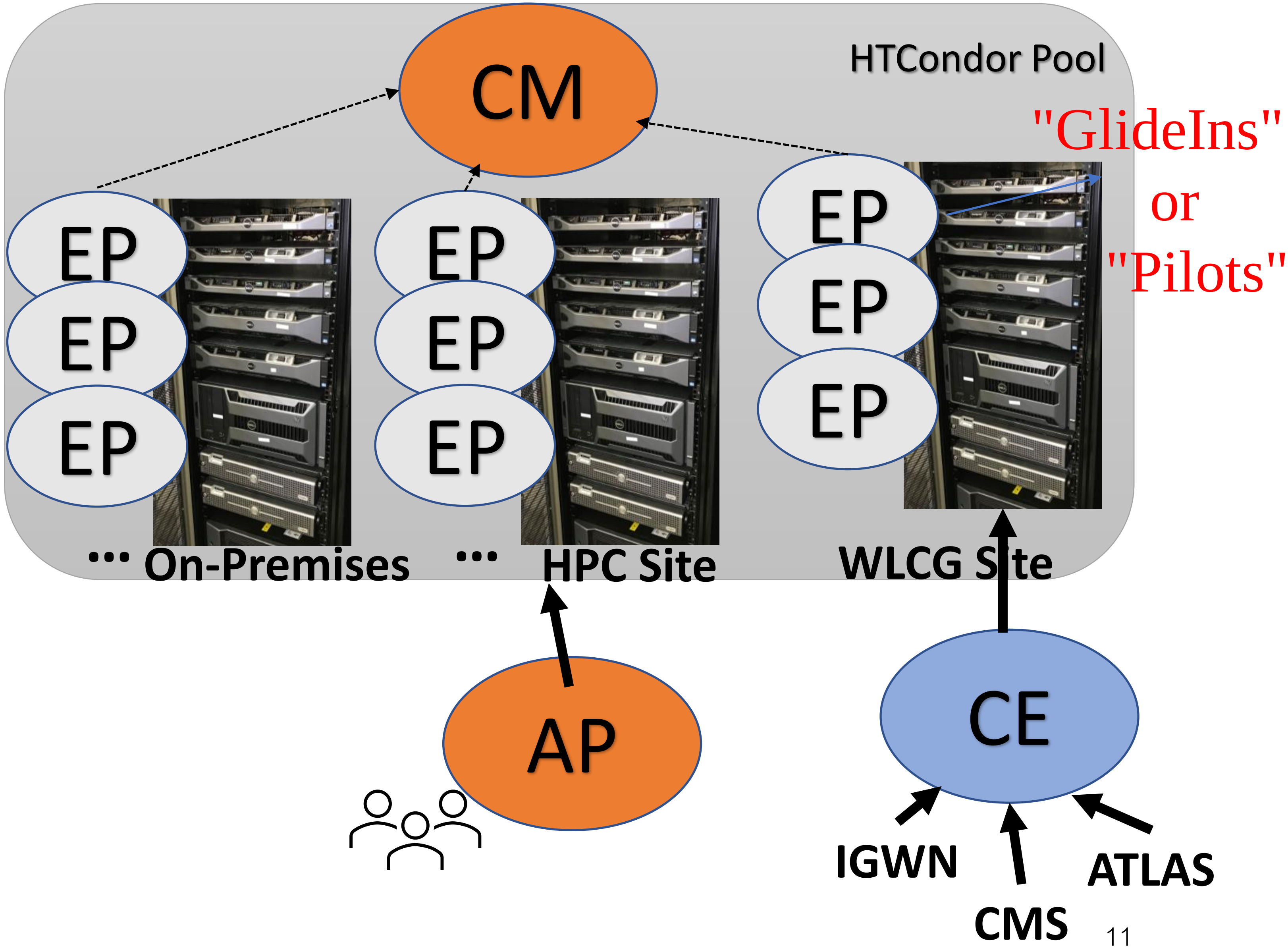
What is the HTCSS, what does it do?

HTCSS provides a distributed high-throughput batch computing environment

- Manages workflows / sets of jobs for researchers
- Federates and supervises computing capacity
- Matches the capacity to workflows
- Distributed, highly available

HTCondor Suite Components

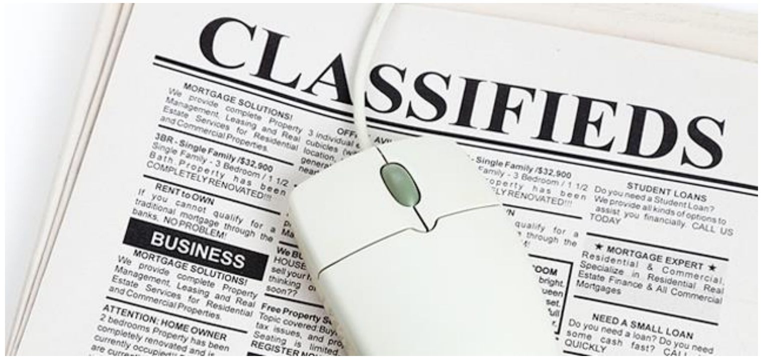
- Access Point (**AP**)
- Directed Acyclic Graph Manager (**DAGMan**)
- Execution Point (**EP**)
- Central Manager (**CM**)
- Compute Entrypoint (**CE**)



Job Matching and Class Ad Attributes

Class Ads

- HTCondor stores a list of information about each job and each computer.
- This information is stored as a “Class Ad”



- Class Ads have the format:

```
AttributeName = value
```

can be a boolean, number, string, or expression

HTCondor Manual: Appendix A: Class Ad Attributes

ClassAd Values

- Literals
  - Strings (“RedHat6”), integers, floats, boolean (true/false), …
- Expressions
  - Similar look to C/C++ or Java : operators, references, functions
  - References: to other attributes in the same ad, or attributes in an ad that is a candidate for a match
  - Operators: +, -, *, /, <, <=,>, >=, ==, !=, &&, and || all work as expected
  - Built-in Functions: if/then/else, string manipulation, regular expression pattern matching, list operations, dates, randomization, math (ceil, floor, quantize,…), time functions, eval, …

ClassAd Examples

AP Job Ad

```
Type   = "Job"
Requirements =
   HasMatlabLicense
    == True &&
   Memory >= 1024
Rank = kflops + 1000000 *
 Memory
Cmd= "/bin/sleep"
Args = "3600"
Owner = "gthain"
NumJobStarts = 8
KindOfJob = "simulation"
Department = "Math"
```

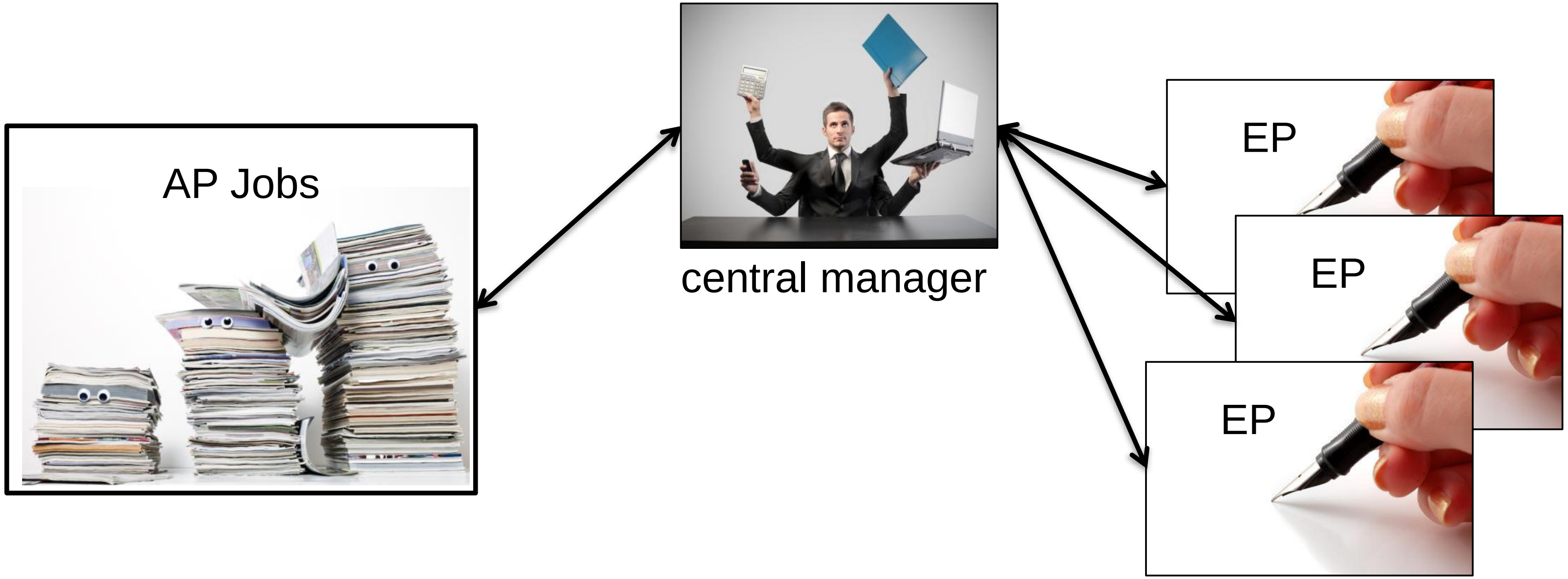
EP Machine Slot Ad

```
Type = "Machine"
Cpus   = 40
Memory = 2048
Requirements =
  (Owner == “gthain”)   ||
  (KindOfJob == “simulation”)
Rank = Department == "Math"
HasMatlabLicense = true
MaxTries = 4
kflops = 41403
```

Job Matching

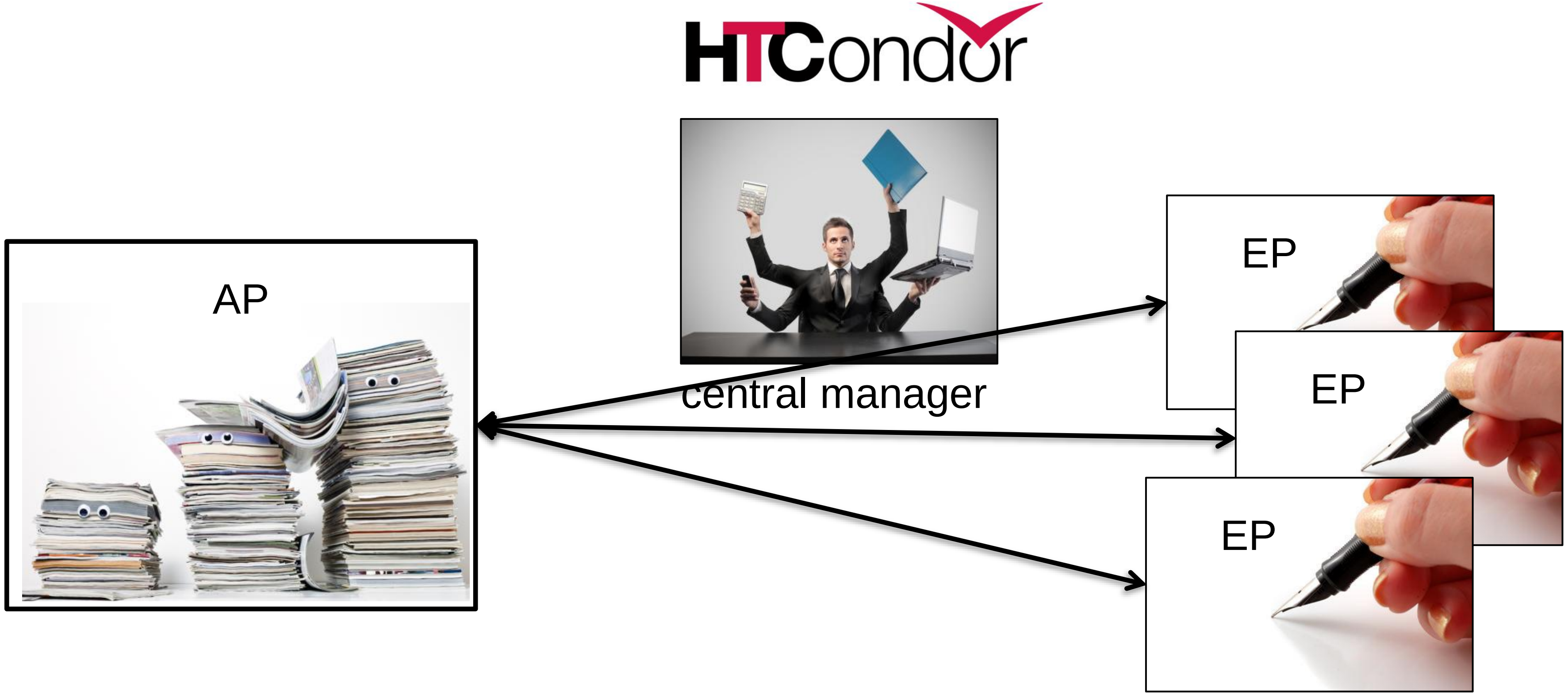
- On a regular basis, the central manager reviews Job resource requests from APs and matches them to EP Slot ads.

HTCondor



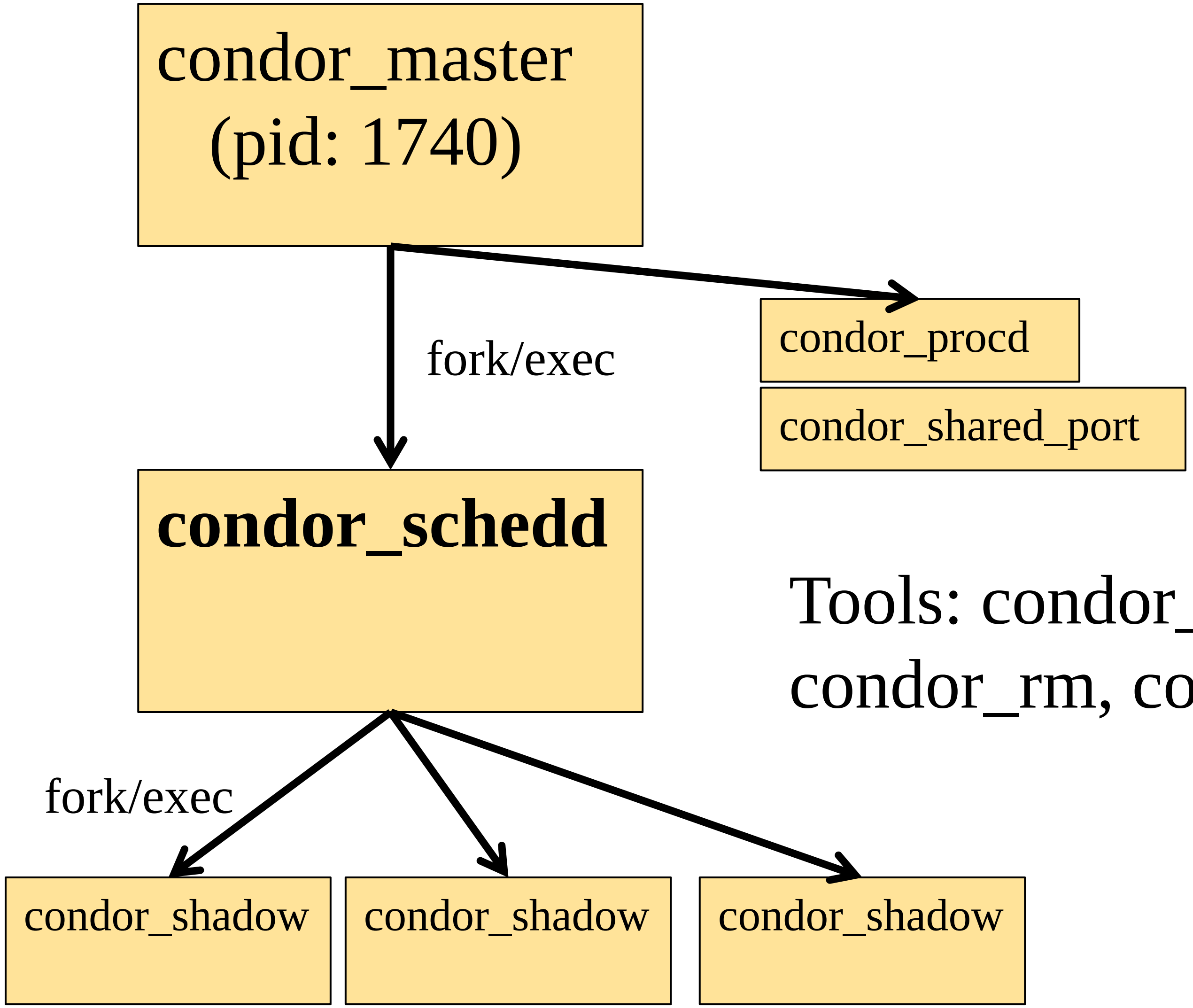
Job Execution

- (Then the AP and EP points communicate directly.)



Architecture & Job Startup

AP Core Process View



condor_master (pid: 1740)

fork/exec

condor_procd

condor_shared_port

condor_schedd

fork/exec

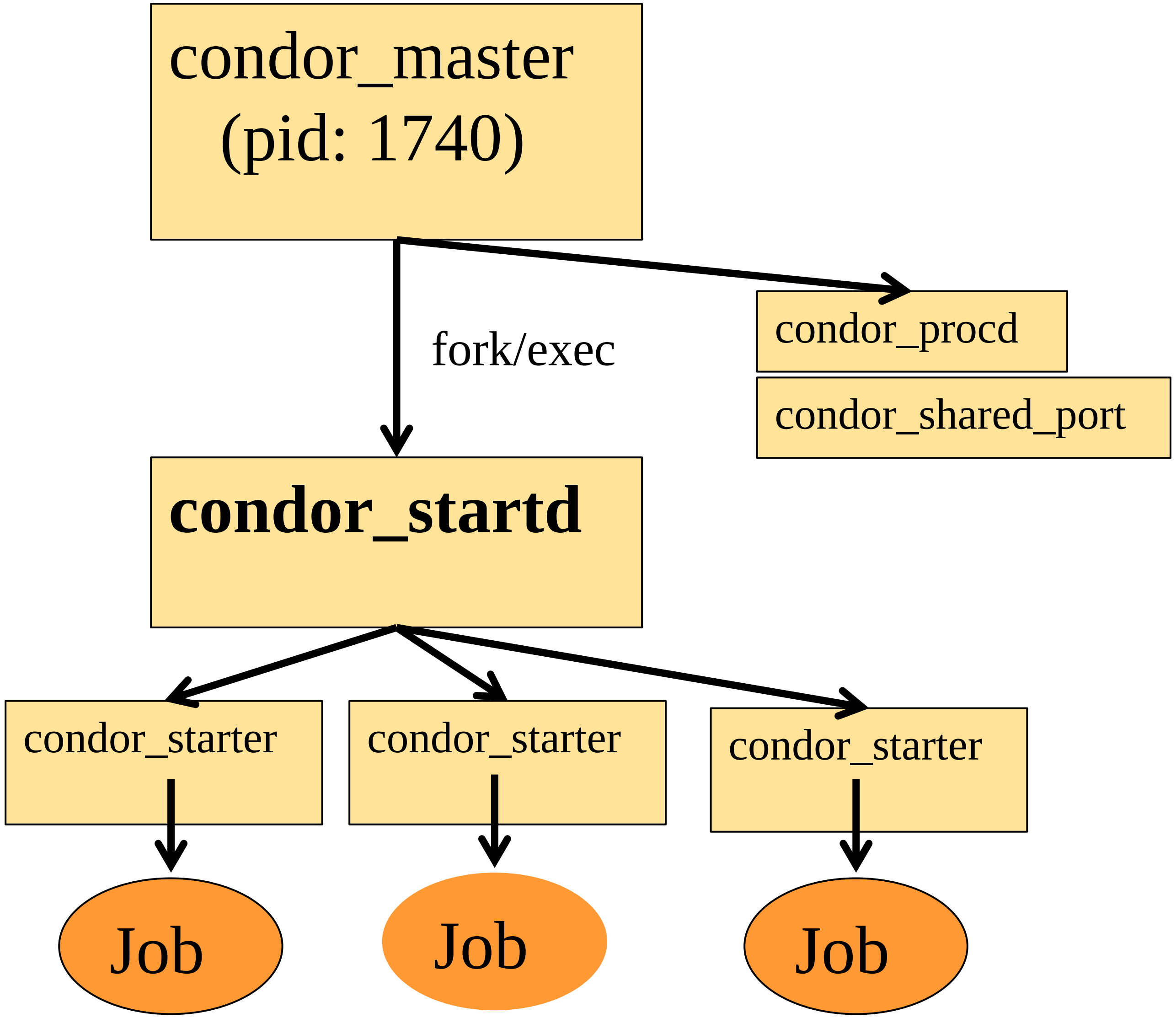
condor_shadow

condor_shadow

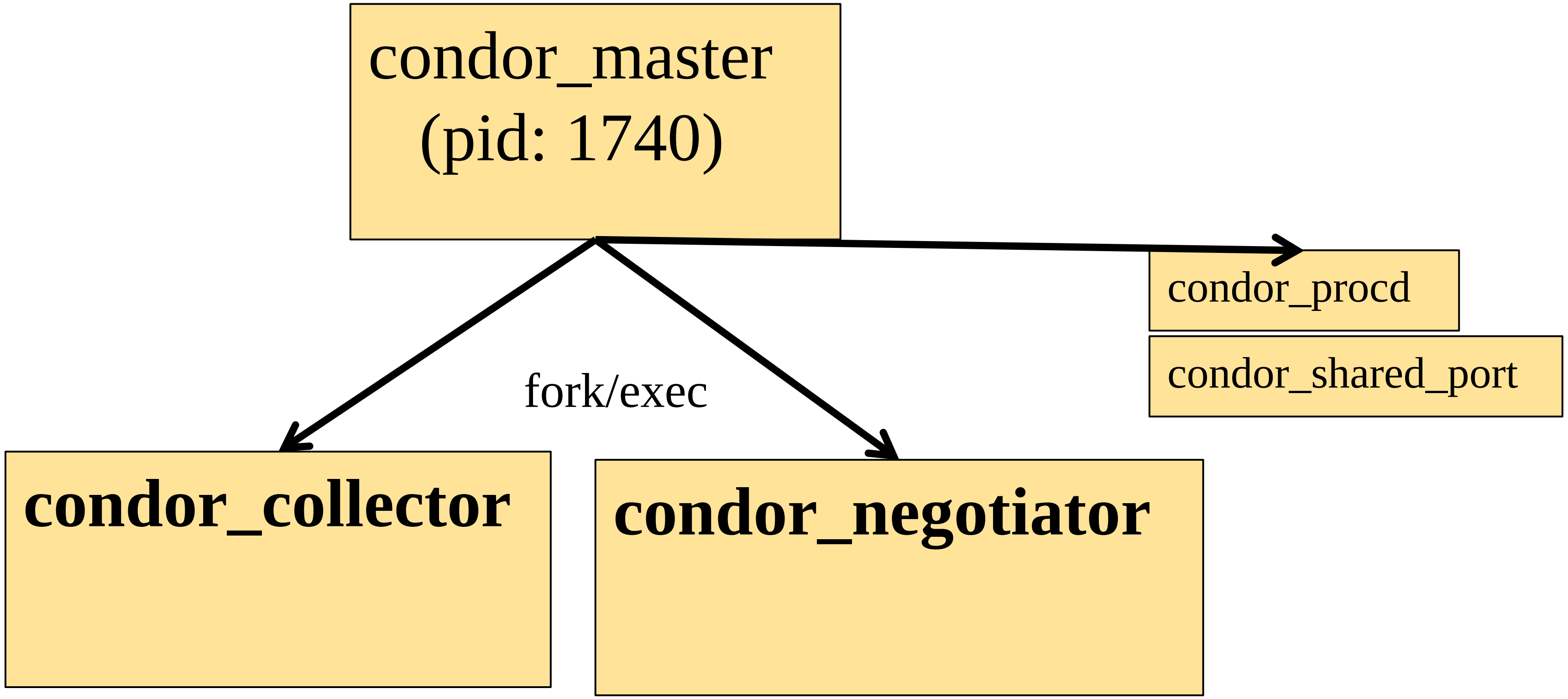
condor_shadow

Tools: condor_submit, condor_q, condor_rm, condor_hold, …

EP Core Process View



Central Manager Process View



condor_master
(pid: 1740)

fork/exec

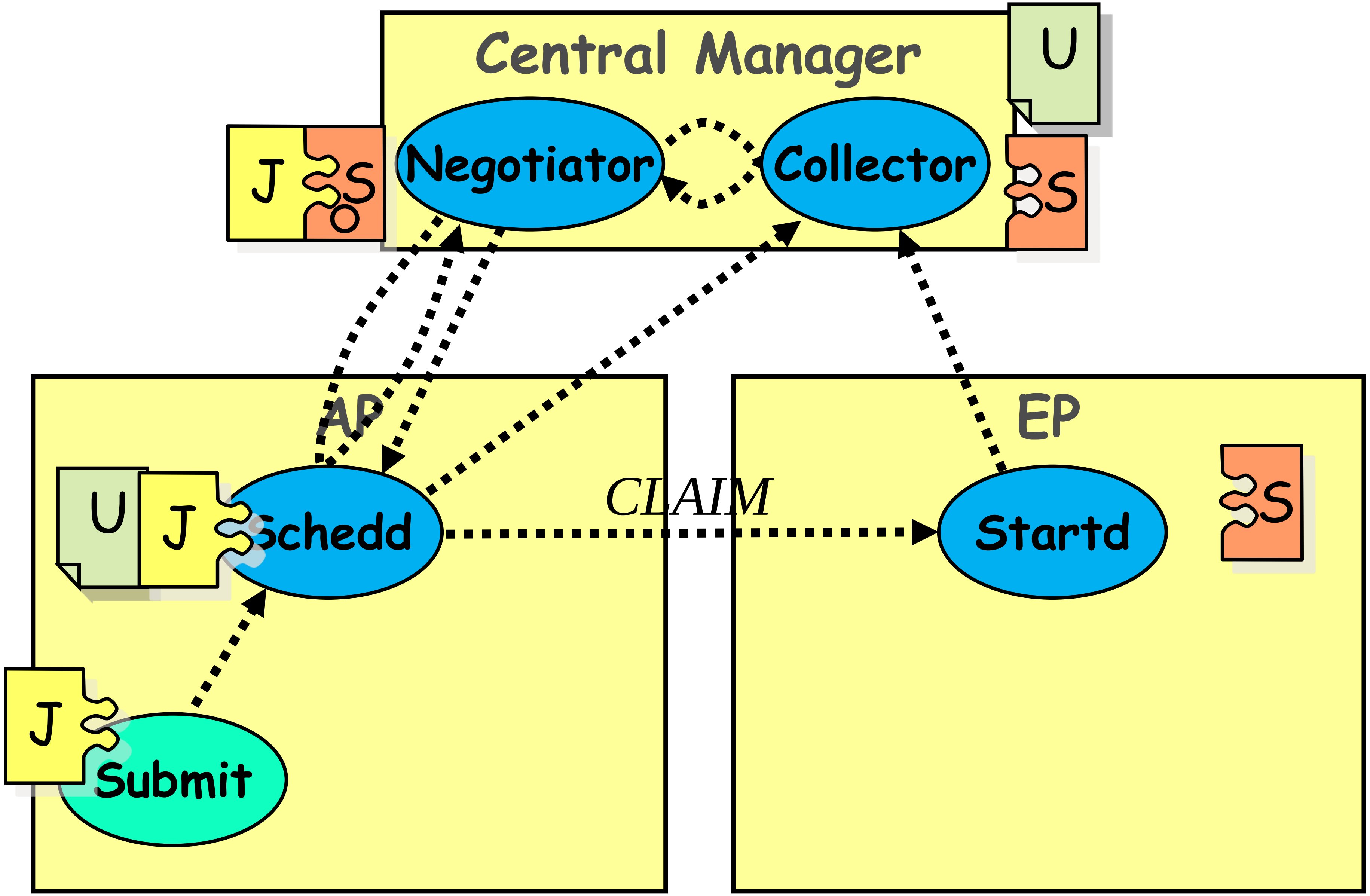
condor_procd

condor_shared_port

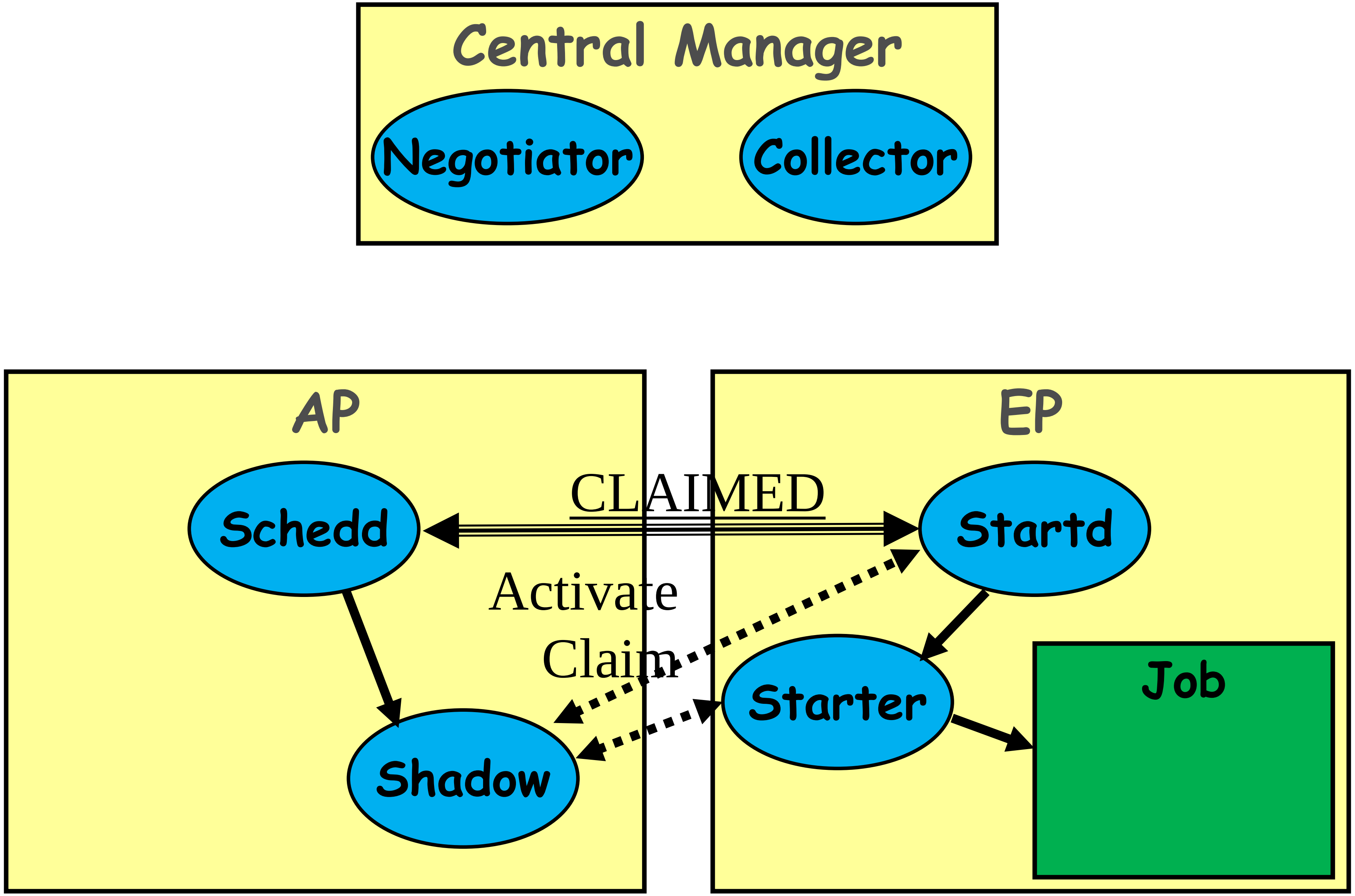
condor_collector

condor_negotiator

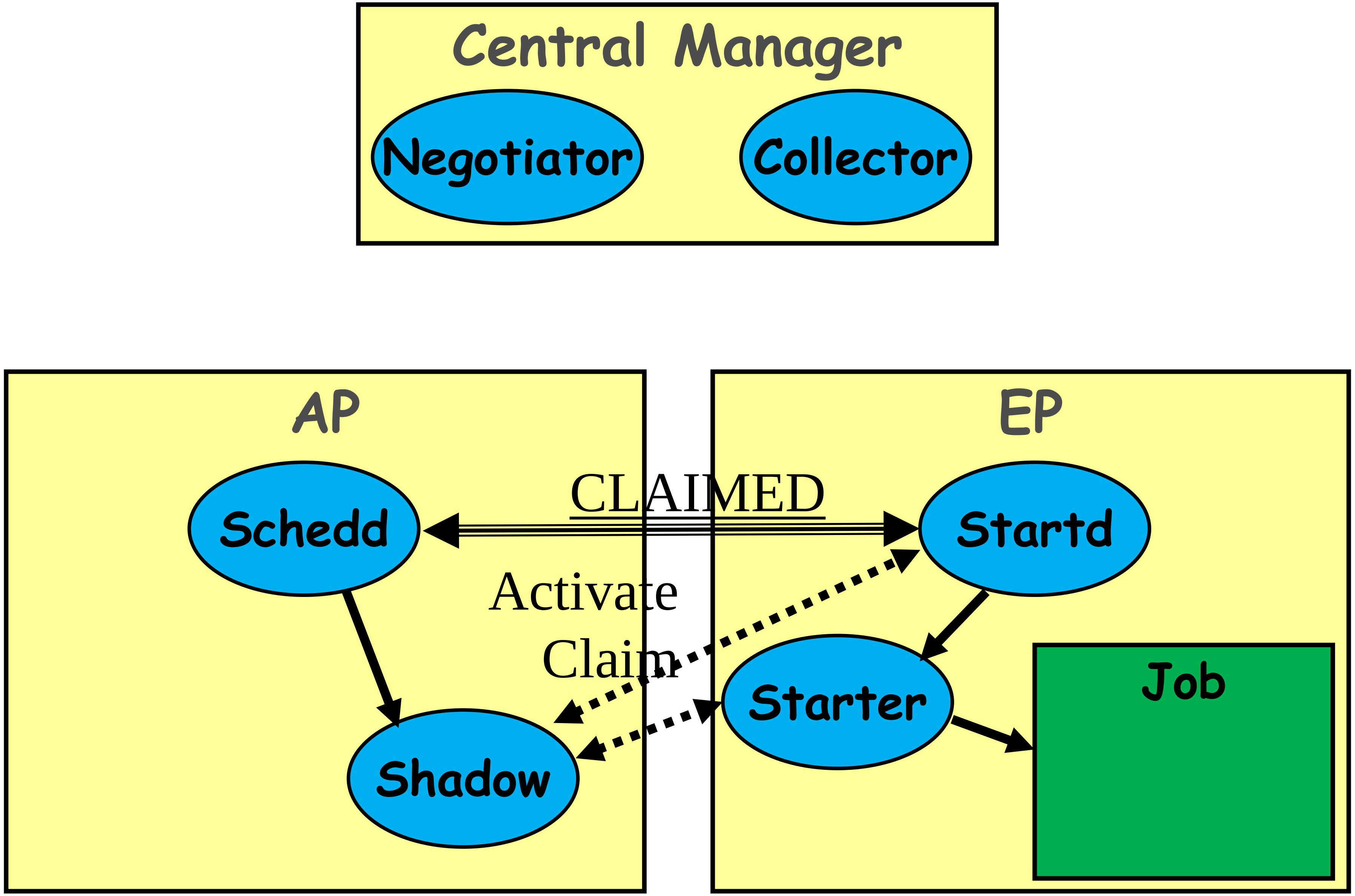
Claiming Protocol



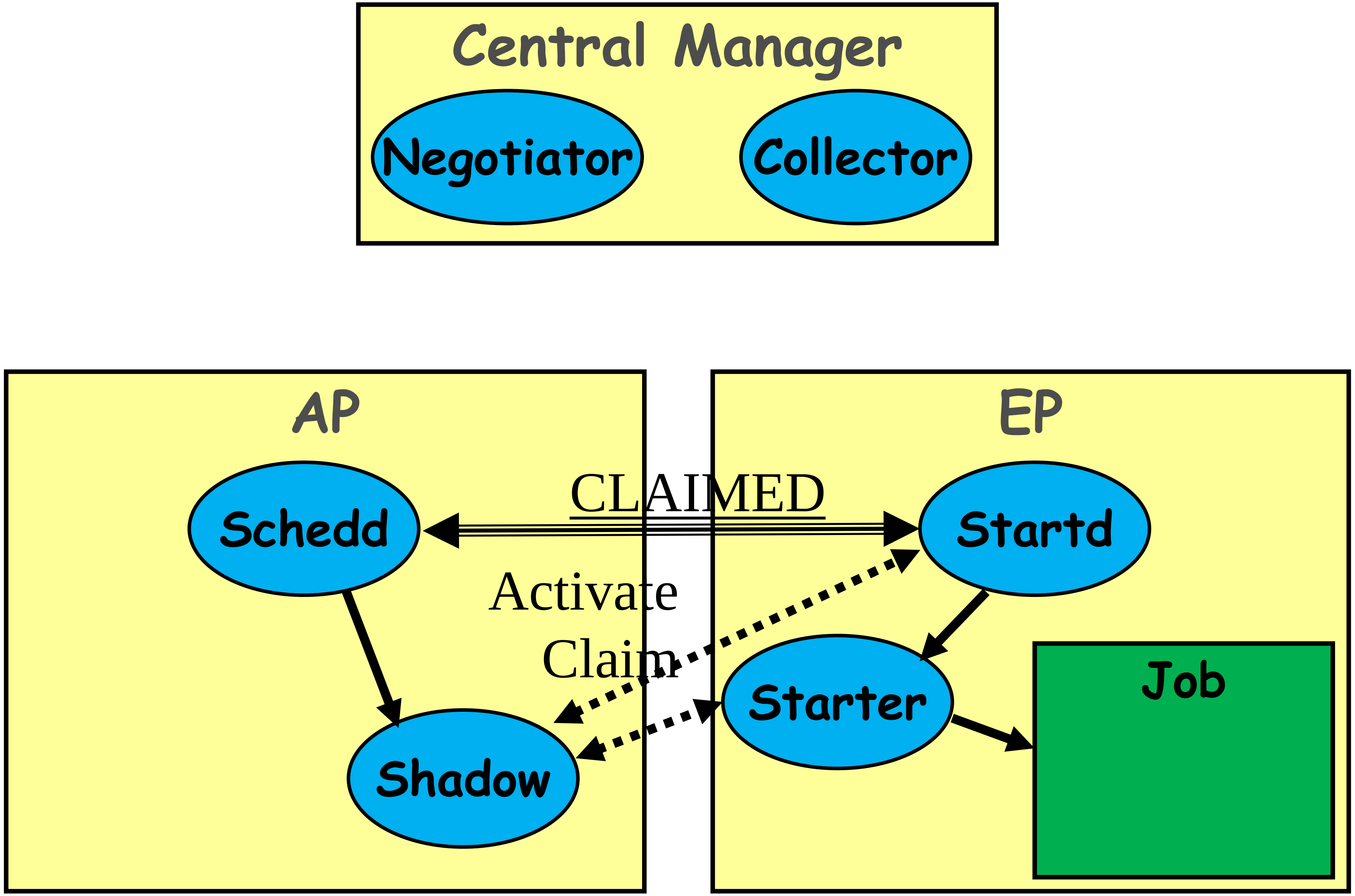
Claim Activation



Repeat until Claim released



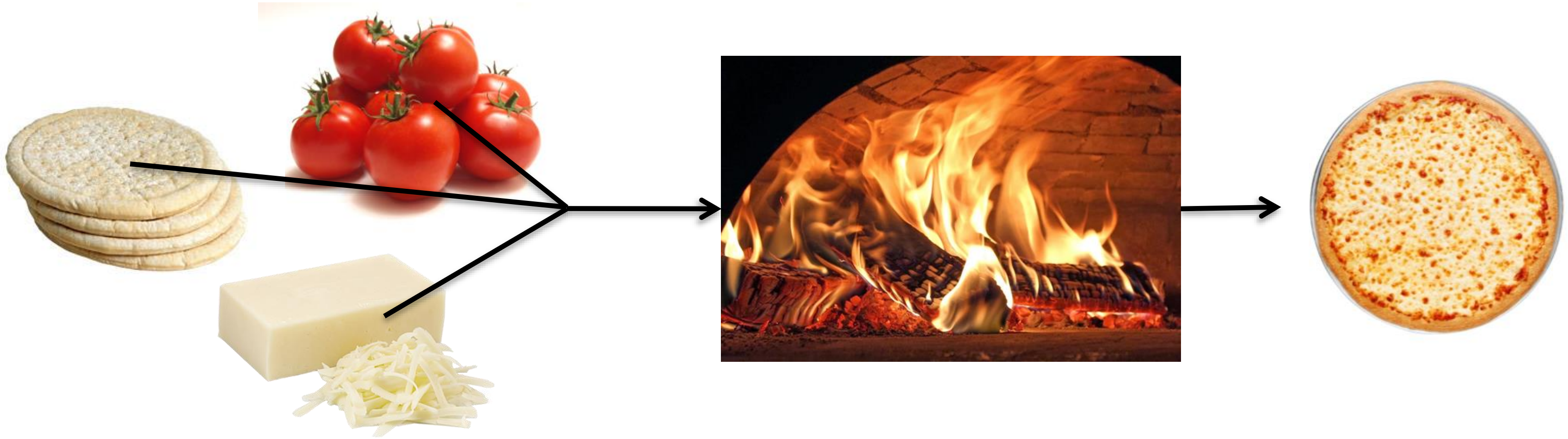
Repeat until Claim released



Running a Job with HTCondor

Jobs

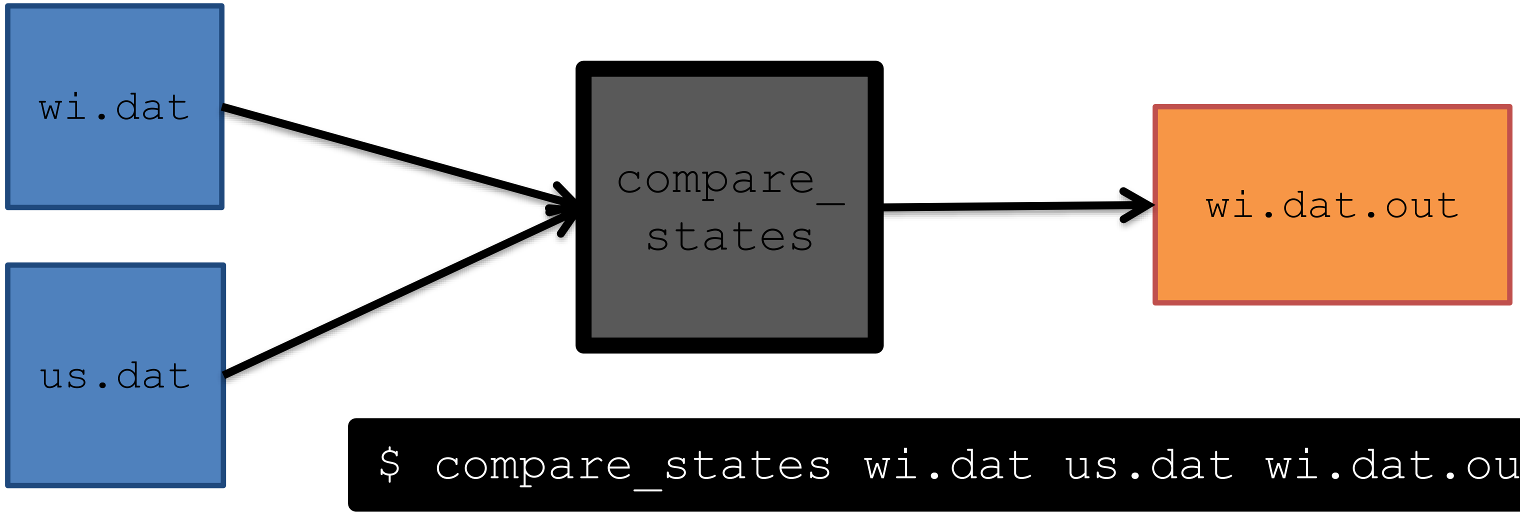
- A single computing task is called a “job”
- Three main pieces of a job are the input, executable (program) and output



- Executable must be runnable from the command line without any interactive input

Job Example

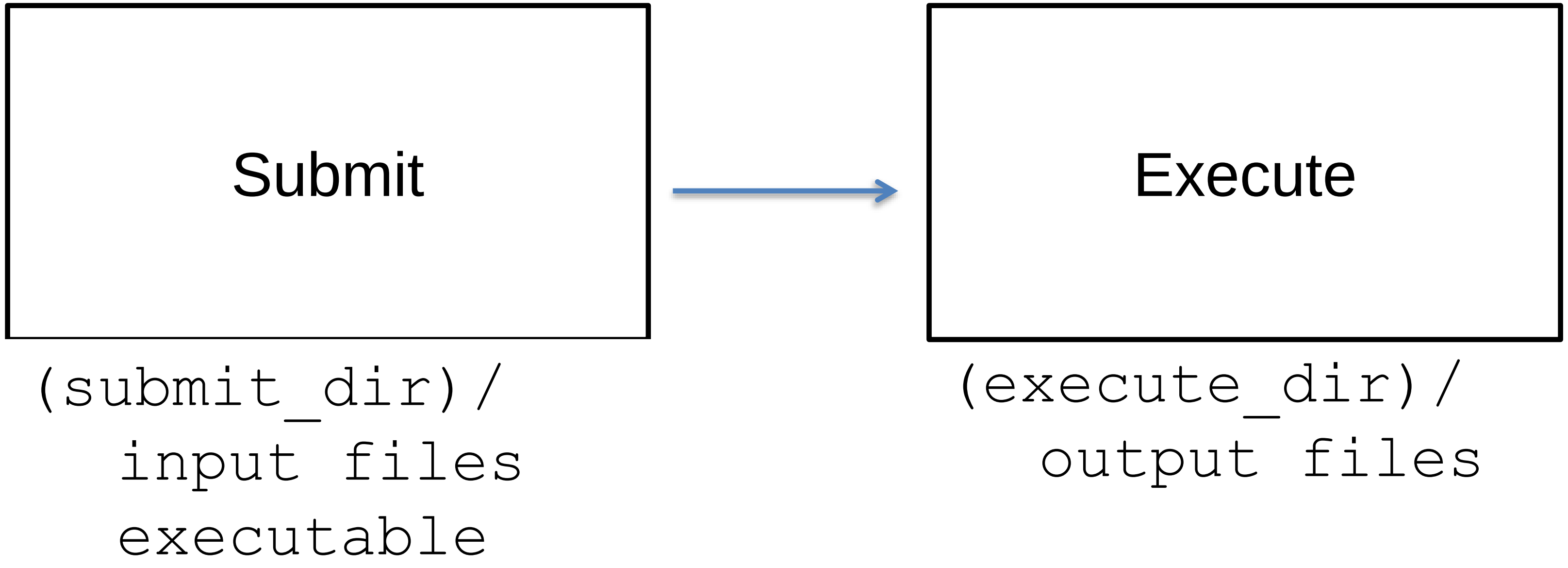
- For our example, we will be using an imaginary program called “compare_states”, which compares two data files and produces a single output file.



```
$ compare_states wi.dat us.dat wi.dat.out
```

File Transfer

- What about files? Can use a shared file system, chirp, or file transfer mechanism.
- Our example will use HTCondor’s file transfer :

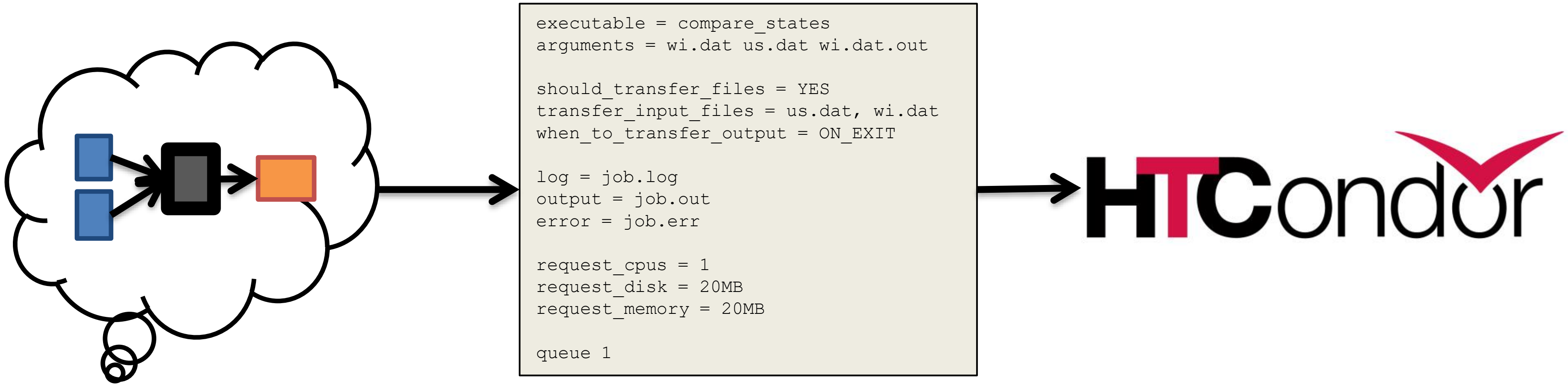


```
(submit_dir)/
  input files
  executable
```

```
(execute_dir)/
  output files
```

Job Translation

- Submit file: communicates everything about your job(s) to the HTCondor Access Point



```
executable = compare_states
arguments = wi.dat us.dat wi.dat.out

should_transfer_files = YES
transfer_input_files = us.dat, wi.dat
when_to_transfer_output = ON_EXIT

log = job.log
output = job.out
error = job.err

request_cpus = 1
request_disk = 20MB
request_memory = 20MB

queue 1
```

Submit File

job.submit

```
executable = compare_states
arguments = wi.dat us.dat wi.dat.out

should_transfer_files = YES
transfer_input_files = us.dat, wi.dat
when_to_transfer_output = ON_EXIT

log = job.log
output = job.out
error = job.err

request_cpus = 1
request_disk = 20MB
request_memory = 20MB

queue 1
```

Submit File

job.submit

```
executable = compare_states
arguments = wi.dat us.dat wi.dat.out

should_transfer_files = YES
transfer_input_files = us.dat, wi.dat
when_to_transfer_output = ON_EXIT

log = job.log
output = job.out
error = job.err

request_cpus = 1
request_disk = 20MB
request_memory = 20MB

queue 1
```

- List your executable and any arguments it takes.

compare_
states

- Arguments are any options passed to the executable from the command line.

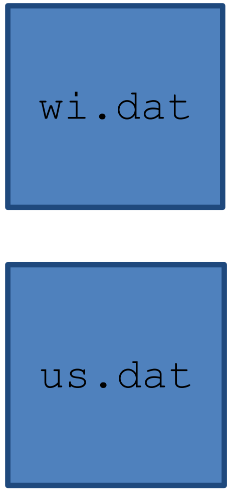
```
$ compare_states wi.dat us.dat wi.dat.out
```

Submit File

job.submit

```
executable = compare_states
arguments = wi.dat us.dat wi.dat.out

should_transfer_files = YES
transfer_input_files = us.dat, wi.dat
when_to_transfer_output = ON_EXIT

log = job.log
output = job.out
error = job.err

request_cpus = 1
request_disk = 20MB
request_memory = 20MB

queue 1
```

- Indicate your input files.



Submit File

job.submit

```
executable = compare_states
arguments = wi.dat us.dat wi.dat.out

should_transfer_files = YES
transfer_input_files = us.dat, wi.dat
when_to_transfer_output = ON_EXIT

log = job.log
output = job.out
error = job.err

request_cpus = 1
request_disk = 20MB
request_memory = 20MB

queue 1
```

- HTCondor will transfer back all new and changed files (usually output) from the job.

wi.dat.out

Submit File

job.submit

```
executable = compare_states
arguments = wi.dat us.dat wi.dat.out

should_transfer_files = YES
transfer_input_files = us.dat, wi.dat
when_to_transfer_output = ON_EXIT

log = job.log
output = job.out
error = job.err

request_cpus = 1
request_disk = 20MB
request_memory = 20MB

queue 1
```

- `log`: file created by HTCondor to track job progress
- `output`/`error`: captures stdout and stderr

Submit File

job.submit

```
executable = compare_states
arguments = wi.dat us.dat wi.dat.out

should_transfer_files = YES
transfer_input_files = us.dat, wi.dat
when_to_transfer_output = ON_EXIT

log = job.log
output = job.out
error = job.err

request_cpus = 1
request_disk = 20MB
request_memory = 20MB

queue 1
```

- Request the appropriate resources for your job to run.
- `queue`: keyword indicating “create a job.”

Resource Request

- Jobs are nearly always using a part of a computer, not the whole thing. EP divides worker node into execute "Slots".
- Very important to request appropriate resources (memory, cpus, disk) for a job

Submitting and Monitoring

- To submit a job/jobs:

  `condor_submit` *`submit_file_name`*

- To monitor submitted jobs, use:

  `condor_q`

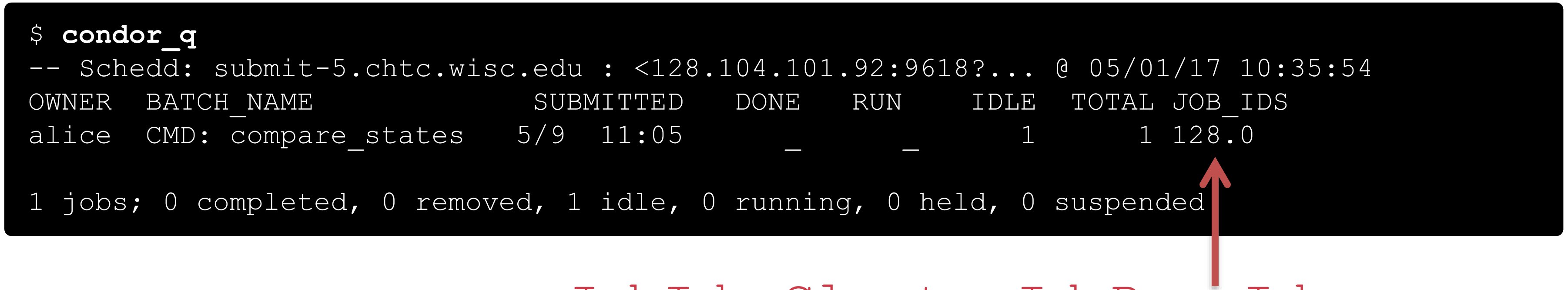
```
$ condor_submit job.submit
Submitting job(s).
1 job(s) submitted to cluster 128.
```

```
$ condor_q
-- Schedd: submit-5.chtc.wisc.edu : <128.104.101.92:9618?... @ 05/01/17 10:35:54
OWNER  BATCH_NAME             SUBMITTED   DONE   RUN    IDLE  TOTAL JOB_IDS
alice  CMD: compare_states   5/9  11:05      _      _      1      1 128.0

1 jobs; 0 completed, 0 removed, 1 idle, 0 running, 0 held, 0 suspended
```

[HTCondor Manual: condor_submit]
[HTCondor Manual: condor_q]

More about condor_q

- By default **condor_q** shows:
  - user’s job only
    - See everyone with "condor_q –allusers"
  - jobs summarized in “batches”
- Constrain with username, ClusterId or full JobId, which will be denoted [U/C/J] in the following slides



```
$ condor_q
-- Schedd: submit-5.chtc.wisc.edu : <128.104.101.92:9618?... @ 05/01/17 10:35:54
OWNER  BATCH_NAME             SUBMITTED   DONE   RUN    IDLE  TOTAL JOB_IDS
alice  CMD: compare_states   5/9  11:05      _      _      1      1 128.0

1 jobs; 0 completed, 0 removed, 1 idle, 0 running, 0 held, 0 suspended
```

JobId = ClusterId.ProcId

More about condor_q

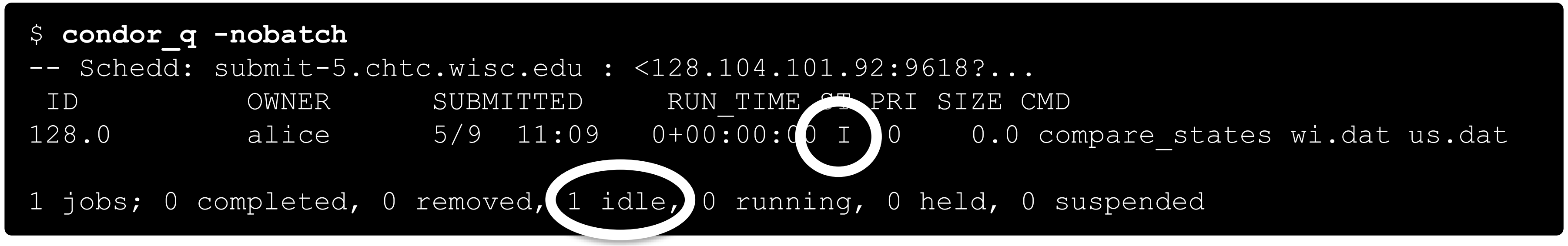
- To see individual job information, use:

  condor_q -nobatch

```
$ condor_q -nobatch
-- Schedd: submit-5.chtc.wisc.edu : <128.104.101.92:9618?...
 ID            OWNER       SUBMITTED      RUN_TIME ST PRI SIZE CMD
128.0          alice       5/9   11:09   0+00:00:00 I  0    0.0 compare_states wi.dat us.dat

1 jobs; 0 completed, 0 removed, 1 idle, 0 running, 0 held, 0 suspended
```

- We will use the `-nobatch` option in the following slides to see extra detail about what is happening with a job

Job Idle



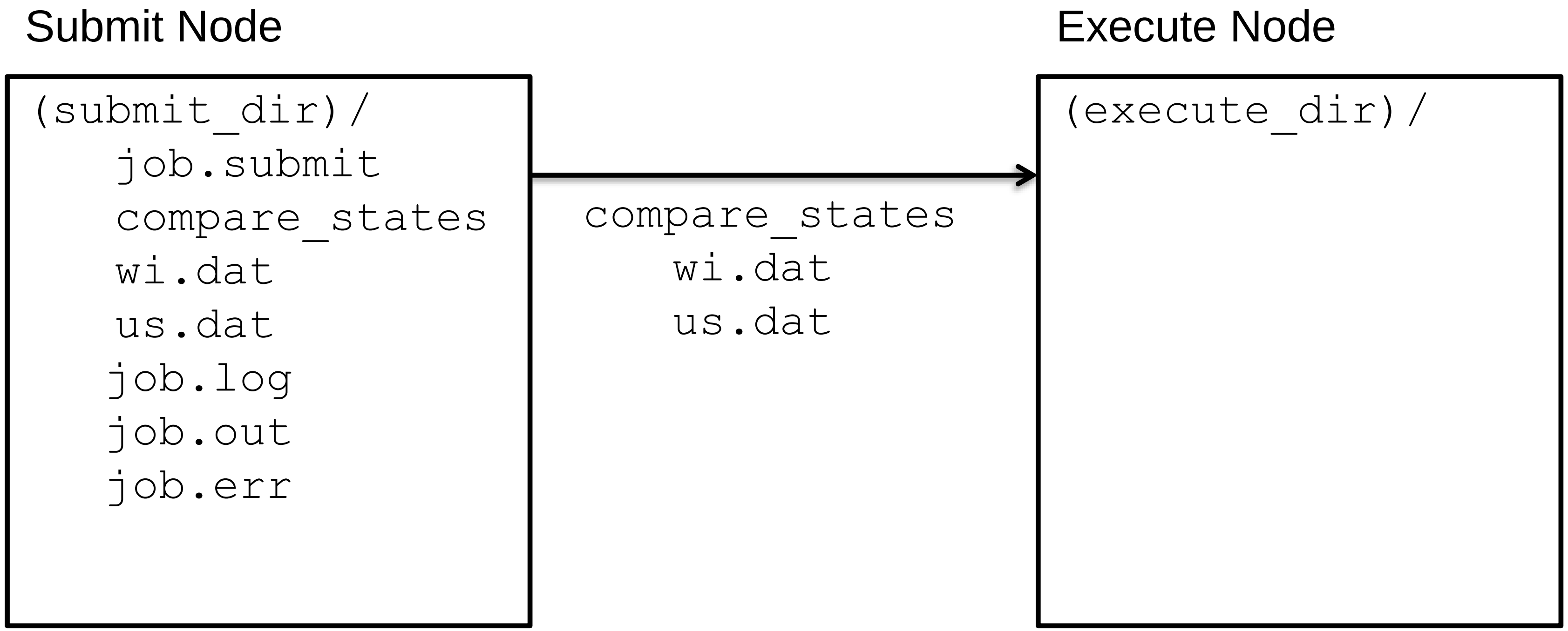
```
$ condor_q -nobatch
-- Schedd: submit-5.chtc.wisc.edu : <128.104.101.92:9618?...
 ID          OWNER      SUBMITTED     RUN_TIME ST PRI SIZE CMD
128.0        alice      5/9  11:09   0+00:00:00 I  0    0.0 compare_states wi.dat us.dat

1 jobs; 0 completed, 0 removed, 1 idle, 0 running, 0 held, 0 suspended
```

Submit Node

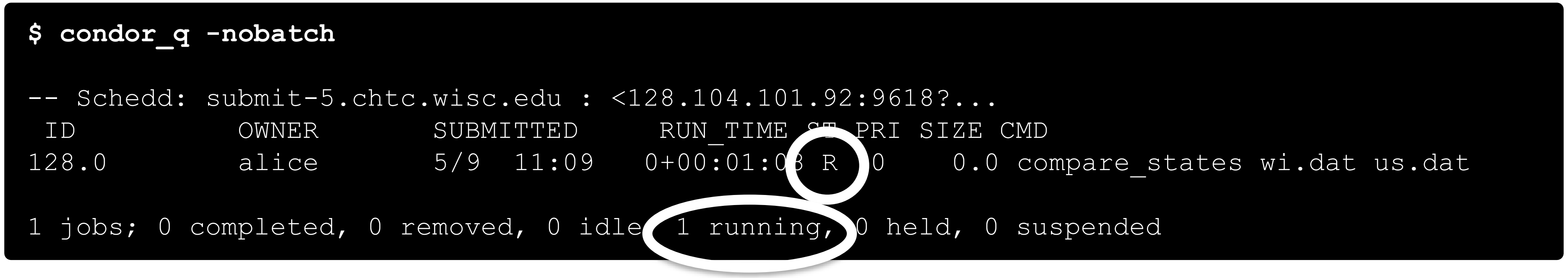
```
(submit_dir)/
    job.submit
    compare_states
    wi.dat
    us.dat
    job.log
    job.out
    job.err
```

Job Starts by doing File Transfer

```
$ condor_q -nobatch
-- Schedd: submit-5.chtc.wisc.edu : <128.104.101.92:9618?...
 ID          OWNER      SUBMITTED     RUN_TIME ST PRI SIZE CMD
128.0        alice      5/9  11:09   0+00:00:00  <  0    0.0 compare_states wi.dat us.dat w

1 jobs; 0 completed, 0 removed, 0 idle, 1 running, 0 held, 0 suspended
```



Job Running



```
$ condor_q -nobatch

-- Schedd: submit-5.chtc.wisc.edu : <128.104.101.92:9618?...
 ID          OWNER        SUBMITTED     RUN_TIME ST PRI SIZE CMD
128.0        alice        5/9  11:09   0+00:01:03 R  0    0.0 compare_states wi.dat us.dat

1 jobs; 0 completed, 0 removed, 0 idle, 1 running, 0 held, 0 suspended
```

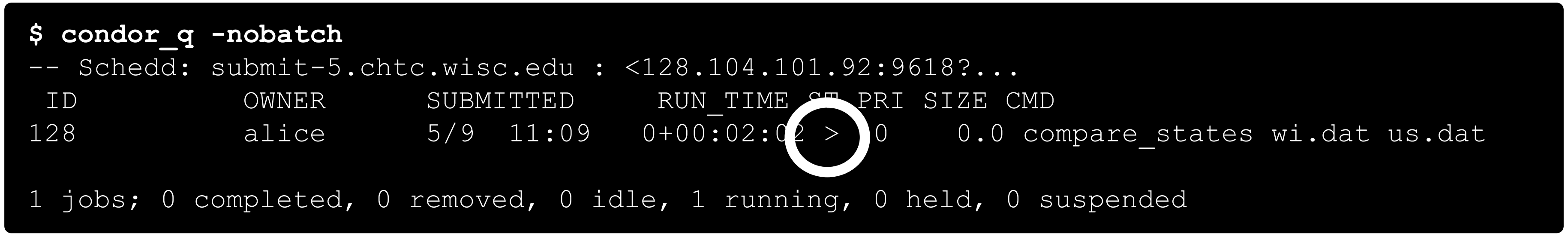
Submit Node

```
(submit_dir)/
    job.submit
    compare_states
    wi.dat
    us.dat
    job.log
    job.out
    job.err
```

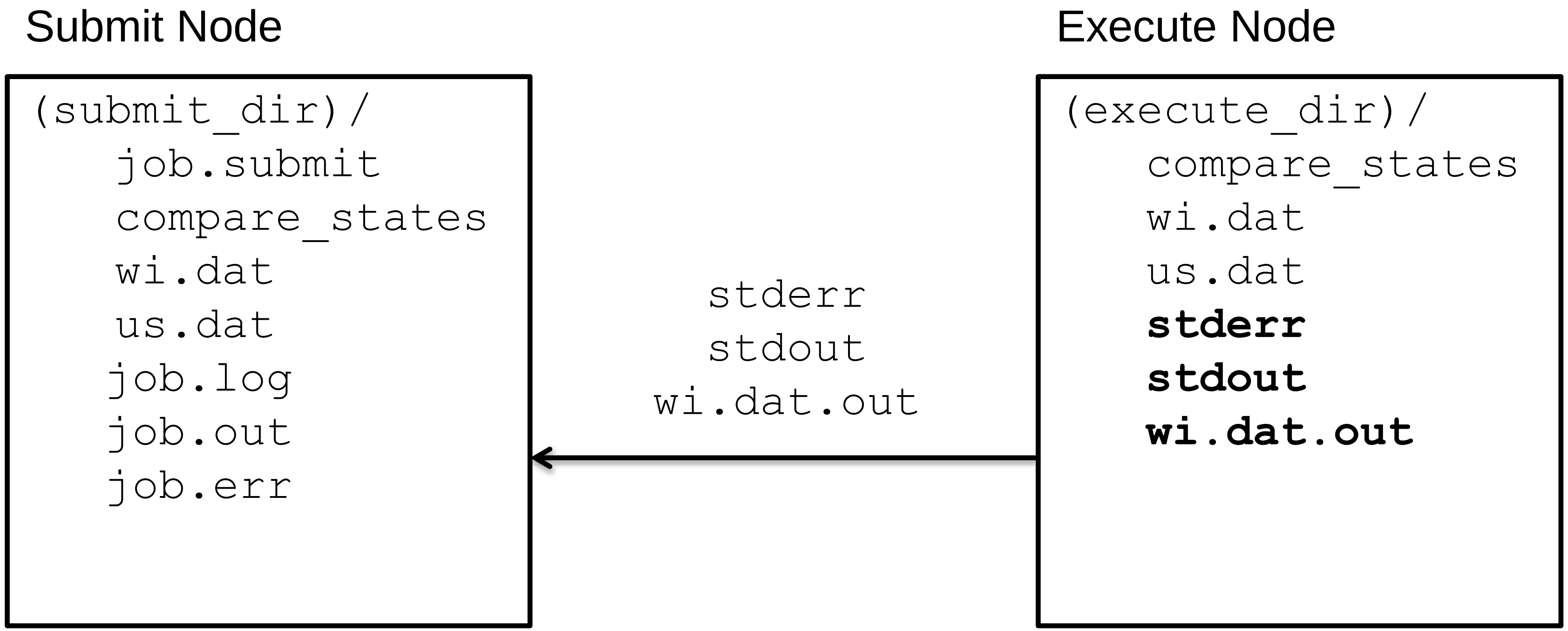
Execute Node

(execute_dir)/
compare_states
wi.dat
us.dat
stderr
stdout
wi.dat.out

Job Completes



```
$ condor_q -nobatch
-- Schedd: submit-5.chtc.wisc.edu : <128.104.101.92:9618?...
 ID          OWNER      SUBMITTED     RUN_TIME ST PRI SIZE CMD
128          alice      5/9  11:09   0+00:02:02 >  0    0.0 compare_states wi.dat us.dat

1 jobs; 0 completed, 0 removed, 0 idle, 1 running, 0 held, 0 suspended
```



Submit Node

```
(submit_dir)/
    job.submit
    compare_states
    wi.dat
    us.dat
   job.log
   job.out
   job.err
```

Execute Node

```
(execute_dir)/
    compare_states
    wi.dat
    us.dat
    stderr
    stdout
    wi.dat.out
```

```
stderr
stdout
wi.dat.out
```

Job Completes (cont.)

```
$ condor_q -nobatch


-- Schedd: submit-5.chtc.wisc.edu : <128.104.101.92:9618?...
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD

0 jobs; 0 completed, 0 removed, 0 idle, 0 running, 0 held, 0 suspended
```

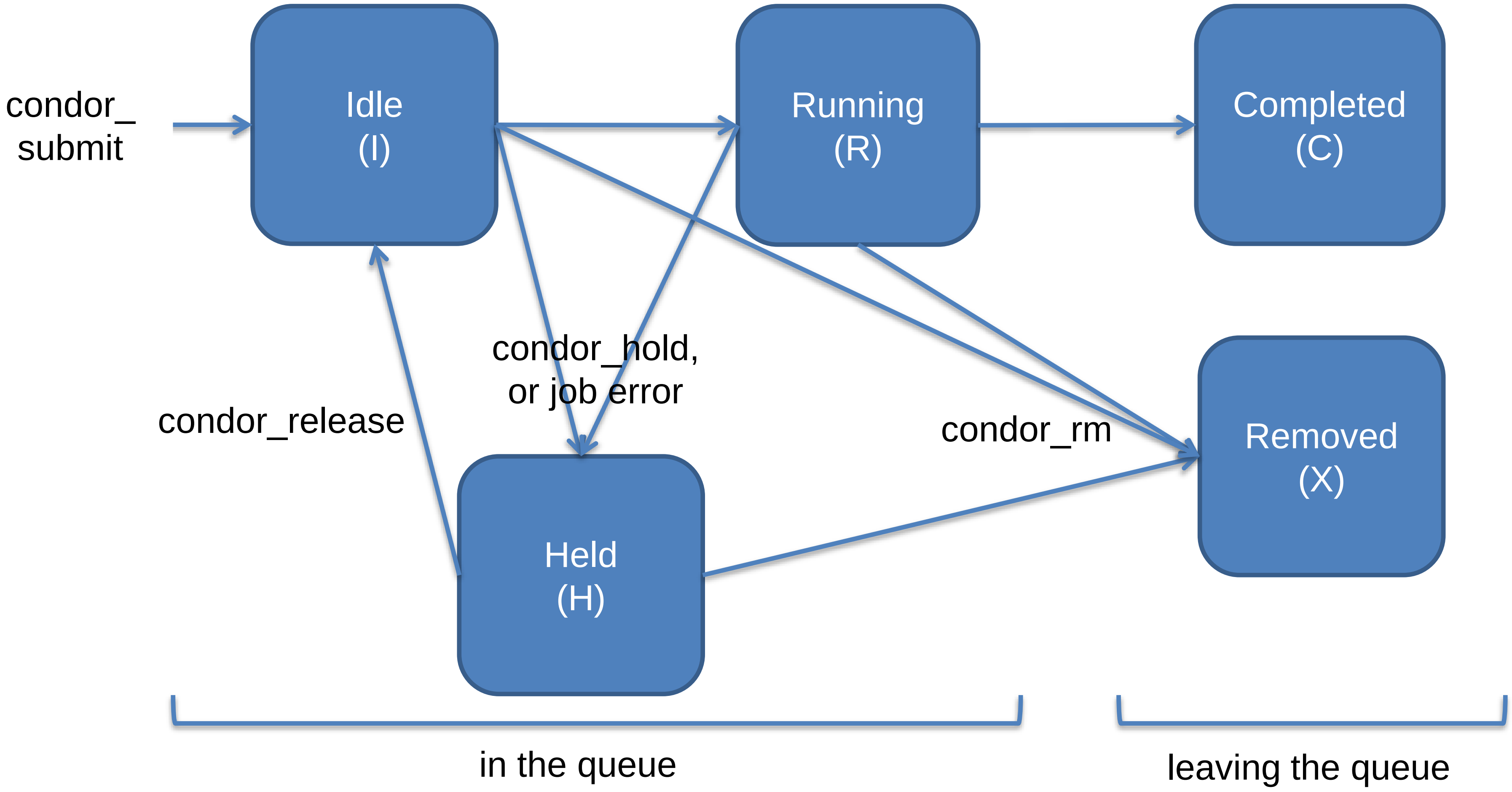
Submit Node

```
(submit_dir)/
    job.submit
    compare_states
    wi.dat
    us.dat
    job.log
    job.out
    job.err
    wi.dat.out
```

Job Event Log File

```
000 (128.000.000) 05/09 11:09:08 Job submitted from host:
<128.104.101.92&sock=6423_b881_3>
...
001 (128.000.000) 05/09 11:10:46 Job executing on host:
<128.104.101.128:9618&sock=5053_3126_3>
...
006 (128.000.000) 05/09 11:10:54 Image size of job updated: 220
    1  -  MemoryUsage of job (MB)
    220  -  ResidentSetSize of job (KB)
...
005 (128.000.000) 05/09 11:12:48 Job terminated.
    (1) Normal termination (return value 0)
        Usr 0 00:00:00, Sys 0 00:00:00  -  Run Remote Usage
        Usr 0 00:00:00, Sys 0 00:00:00  -  Run Local Usage
        Usr 0 00:00:00, Sys 0 00:00:00  -  Total Remote Usage
        Usr 0 00:00:00, Sys 0 00:00:00  -  Total Local Usage
    0  -  Run Bytes Sent By Job
    33  -  Run Bytes Received By Job
    0  -  Total Bytes Sent By Job
    33  -  Total Bytes Received By Job
    Partitionable Resources :    Usage  Request Allocated
       Cpus                 :        1        1         1
       Disk (KB)            :       14    20480  17203728
       Memory (MB)          :        1       20        20
```

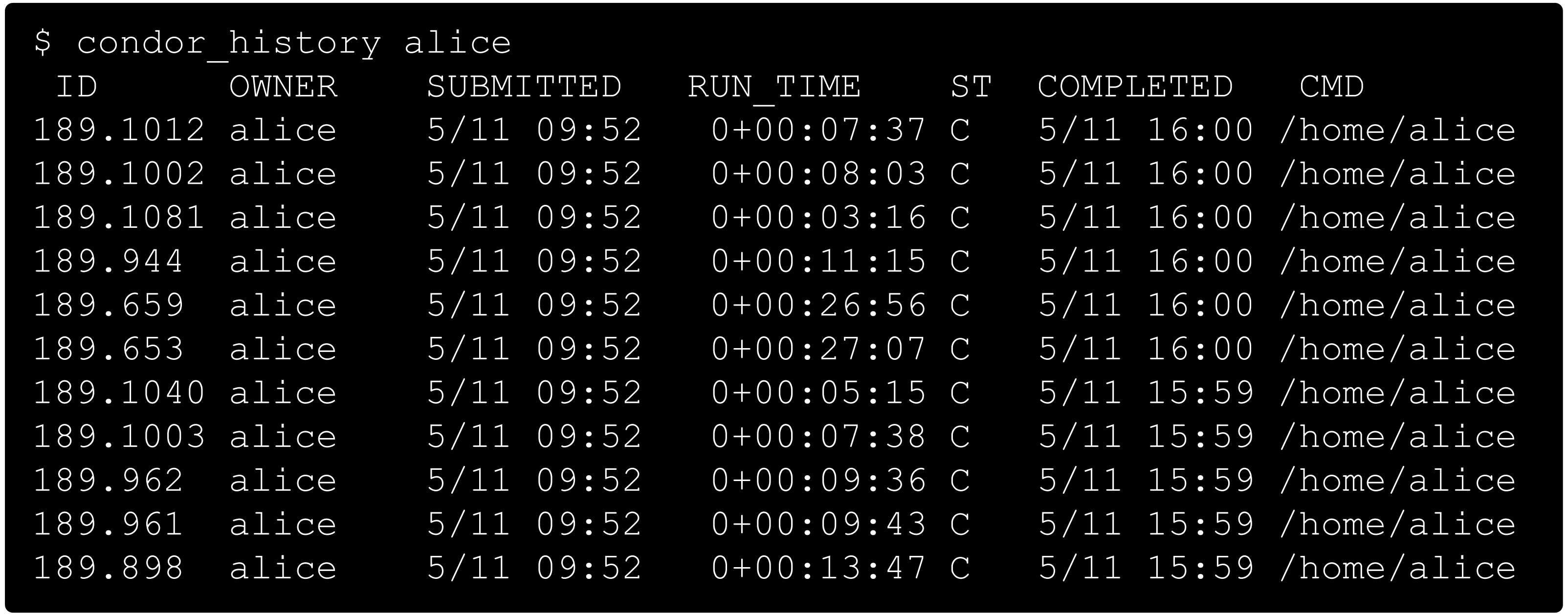
Job States


condor_
submit
Idle
(I)
Running
(R)
Completed
(C)
condor_hold,
or job error
condor_release
condor_rm
Removed
(X)
Held
(H)
in the queue
leaving the queue

Reviewing Completed Jobs

- To review completed jobs, use **condor_history**

As **condor_q** is to the present, **condor_history** is to the past



```
$ condor_history alice
 ID      OWNER    SUBMITTED   RUN_TIME    ST  COMPLETED   CMD
189.1012 alice    5/11 09:52   0+00:07:37 C   5/11 16:00 /home/alice
189.1002 alice    5/11 09:52   0+00:08:03 C   5/11 16:00 /home/alice
189.1081 alice    5/11 09:52   0+00:03:16 C   5/11 16:00 /home/alice
189.944  alice    5/11 09:52   0+00:11:15 C   5/11 16:00 /home/alice
189.659  alice    5/11 09:52   0+00:26:56 C   5/11 16:00 /home/alice
189.653  alice    5/11 09:52   0+00:27:07 C   5/11 16:00 /home/alice
189.1040 alice    5/11 09:52   0+00:05:15 C   5/11 15:59 /home/alice
189.1003 alice    5/11 09:52   0+00:07:38 C   5/11 15:59 /home/alice
189.962  alice    5/11 09:52   0+00:09:36 C   5/11 15:59 /home/alice
189.961  alice    5/11 09:52   0+00:09:43 C   5/11 15:59 /home/alice
189.898  alice    5/11 09:52   0+00:13:47 C   5/11 15:59 /home/alice
```

HTCondor Manual: condor_history

Submitting Multiple Jobs with HTCondor

Many Jobs, One Submit File

- HTCondor has built-in ways to submit multiple independent jobs with one submit file

Advantages

- Run many independent jobs...
  - analyze multiple data files
  - test parameter or input combinations
  - and more!
- ...without having to:
  - start each job individually
  - create separate submit files for each job

Multiple, Numbered, Input Files

job.submit

```
executable = analyze.exe
arguments = file.in file.out
transfer_input_files = file.in

log = job.log
output = job.out
error = job.err

queue
```

(submit_dir)/

```
analyze.exe
file0.in
file1.in
file2.in

job.submit
```

- Goal: create 3 jobs that each analyze a different input file.

Multiple Jobs, No Variation

job.submit

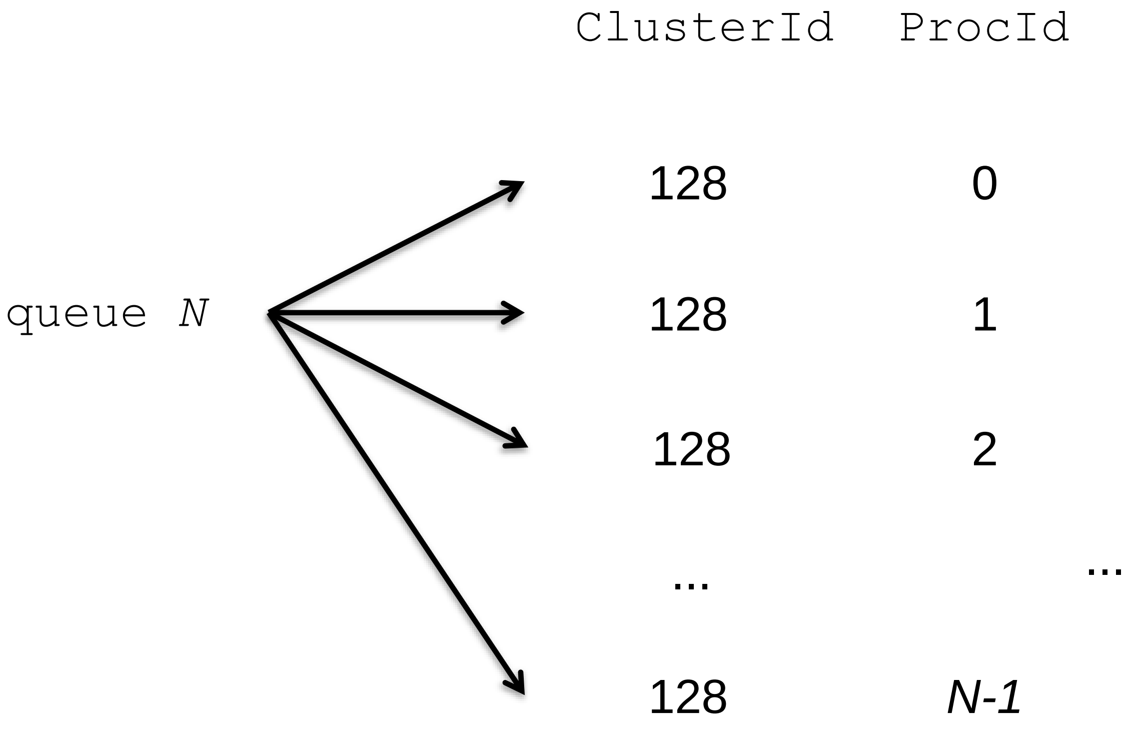
```
executable = analyze.exe
arguments = file0.in file0.out
transfer_input_files = file.in

log = job.log
output = job.out
error = job.err

queue 3
```

(submit_dir)/

```
analyze.exe
file0.in
file1.in
file2.in

job.submit
```

- This file generates 3 jobs, but doesn’t use multiple inputs and will overwrite outputs

Automatic Variables



`queue` *`N`*

`ClusterId`	`ProcId`
128	0
128	1
128	2
...	...
128	*N-1*

- Each job’s `ClusterId` and `ProcId` numbers are saved as job attributes
- They can be accessed inside the submit file using:
  - `$(ClusterId)`
  - `$(ProcId)`

Separate Jobs with InitialDir

(submit_dir)/

```
job.submit      job0/          job1/          job2/
analyze.exe       file.in        file.in        file.in
                  job.log        job.log        job.log
                  job.err        job.err        job.err
                  file.out       file.out       file.out
```

job.submit

```
executable = analyze.exe
initialdir = job$(ProcId)
arguments = file.in file.out
transfer_input_files = file.in

log = job.log
error = job.err

queue 3
```

Other Submission Methods

- What if your input files/directories aren’t numbered from 0 - (N-1)?
- There are other ways to submit many jobs!

Possible Queue Statements

matching ... pattern	`queue infile matching *.dat`
in ... list	`queue infile in (wi.dat ca.dat ia.dat)`

from ... file	`queue infile from state_list.txt`
	`wi.dat`
	`ca.dat`
	`ia.dat`
	`state_list.txt`

... or use the HTCSS Python API!

Using Multiple Variables

- Both the “`from`” and “`in`” syntax support using multiple variables from a list.

job.submit

```
executable = compare_states
arguments = -year $(option) -input $(file)

should_transfer_files = YES
when_to_transfer_output = ON_EXIT
transfer_input_files = $(file)

queue file,option from job_list.txt
```

job_list.txt

```
wi.dat, 2010
wi.dat, 2015
ca.dat, 2010
ca.dat, 2015
ia.dat, 2010
ia.dat, 2015
```

HTCondor Manual: submit file options

Class Ads for Users

- Class Ads also provide lots of useful information about jobs, slots, and daemons to HTCondor users and administrators

Finding Job Attributes

- Use the “long” option for `condor_q`

  `condor_q -l` ***`JobId`***

```
$ condor_q -l 128.0
WhenToTransferOutput = "ON_EXIT"
TargetType = "Machine"
Cmd = "/home/alice/tests/htcondor_week/compare_states"
JobUniverse = 5
Iwd = "/home/alice/tests/htcondor_week"
RequestDisk = 20480
NumJobStarts = 0
WantRemoteIO = true
OnExitRemove = true
TransferInput = "us.dat,wi.dat"
MyType = "Job”
UserLog = "/home/alice/tests/htcondor_week/job.log"
RequestMemory = 20
...
```

Some Useful Job Attributes

- `UserLog`: location of job log
- `Iwd`: Initial Working Directory (i.e. submission directory) on submit node
- `MemoryUsage`: maximum memory the job has used
- `RemoteHost`: where the job is running
- `BatchName`: attribute to label job batches
- ...and more

Selectively display specific attributes

- Use the “auto-format” option:

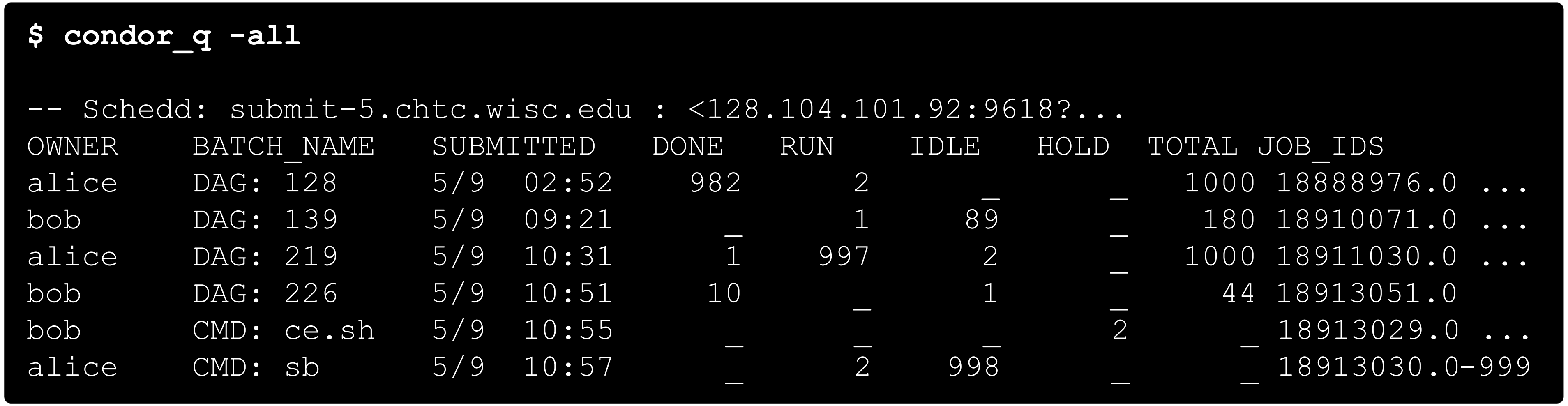
```
condor_q [U/C/J] -af Attribute1 Attribute2 ...
```

```
$ condor_q -af ClusterId ProcId RemoteHost MemoryUsage

17315225 116 slot1_1@e092.chtc.wisc.edu 1709
17315225 118 slot1_2@e093.chtc.wisc.edu 1709
17315225 137 slot1_8@e125.chtc.wisc.edu 1709
17315225 139 slot1_7@e121.chtc.wisc.edu 1709
18050961 0 slot1_5@c025.chtc.wisc.edu 196
18050963 0 slot1_3@atlas10.chtc.wisc.edu 269
18050964 0 slot1_25@e348.chtc.wisc.edu 245
18050965 0 slot1_23@e305.chtc.wisc.edu 196
18050971 0 slot1_6@e176.chtc.wisc.edu 220
```

Other Displays

- See the whole queue (all users, all jobs)

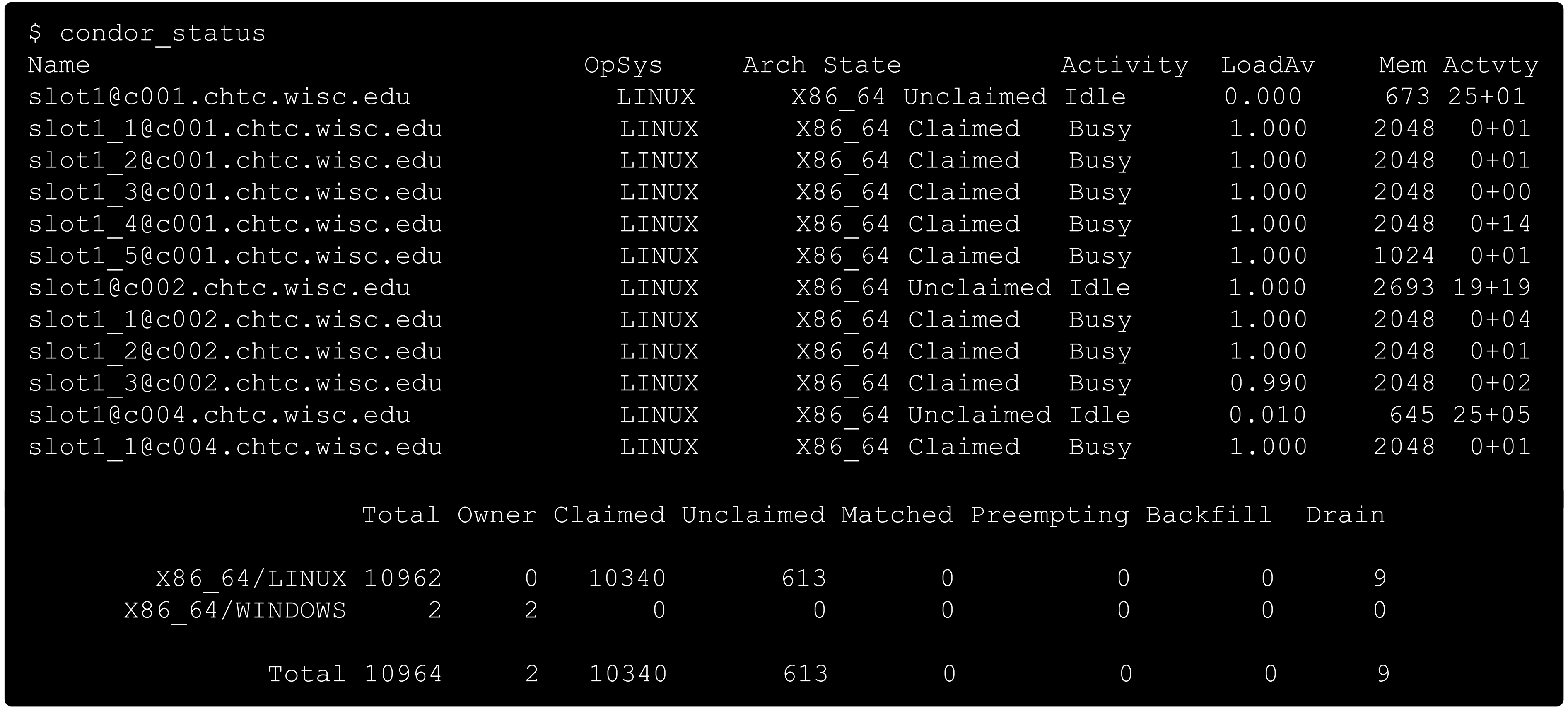
  `condor_q -all`



```
$ condor_q -all

-- Schedd: submit-5.chtc.wisc.edu : <128.104.101.92:9618?...
OWNER    BATCH_NAME    SUBMITTED   DONE   RUN    IDLE   HOLD  TOTAL JOB_IDS
alice    DAG: 128     5/9  02:52    982      2      _      _   1000 18888976.0 ...
bob      DAG: 139     5/9  09:21      _      1     89      _    180 18910071.0 ...
alice    DAG: 219     5/9  10:31      1    997      2      _   1000 18911030.0 ...
bob      DAG: 226     5/9  10:51     10      _      1      _     44 18913051.0
bob      CMD: ce.sh   5/9  10:55      _      _      _      2      _ 18913029.0 ...
alice    CMD: sb      5/9  10:57      _      2    998      _      _ 18913030.0-999
```

Query the Collector: Class Ads from EPs

as **condor_q** is to jobs, **condor_status** is to EP Slots (or “machines”)



```
$ condor_status
Name                               OpSys     Arch State         Activity  LoadAv    Mem Actvty
slot1@c001.chtc.wisc.edu             LINUX      X86_64 Unclaimed Idle      0.000     673 25+01
slot1_1@c001.chtc.wisc.edu           LINUX      X86_64 Claimed   Busy      1.000    2048  0+01
slot1_2@c001.chtc.wisc.edu           LINUX      X86_64 Claimed   Busy      1.000    2048  0+01
slot1_3@c001.chtc.wisc.edu           LINUX      X86_64 Claimed   Busy      1.000    2048  0+00
slot1_4@c001.chtc.wisc.edu           LINUX      X86_64 Claimed   Busy      1.000    2048  0+14
slot1_5@c001.chtc.wisc.edu           LINUX      X86_64 Claimed   Busy      1.000    1024  0+01
slot1@c002.chtc.wisc.edu             LINUX      X86_64 Unclaimed Idle      1.000    2693 19+19
slot1_1@c002.chtc.wisc.edu           LINUX      X86_64 Claimed   Busy      1.000    2048  0+04
slot1_2@c002.chtc.wisc.edu           LINUX      X86_64 Claimed   Busy      1.000    2048  0+01
slot1_3@c002.chtc.wisc.edu           LINUX      X86_64 Claimed   Busy      0.990    2048  0+02
slot1@c004.chtc.wisc.edu             LINUX      X86_64 Unclaimed Idle      0.010     645 25+05
slot1_1@c004.chtc.wisc.edu           LINUX      X86_64 Claimed   Busy      1.000    2048  0+01

                     Total Owner Claimed Unclaimed Matched Preempting Backfill  Drain

        X86_64/LINUX 10962     0   10340       613       0          0        0      9
      X86_64/WINDOWS     2     2       0         0       0          0        0      0

               Total 10964     2   10340       613       0          0        0      9
```

HTCondor Manual: condor_status

Machine Attributes

- Use same options as **`condor_q`**:

`condor_status` ***`-l Slot/Machine`***

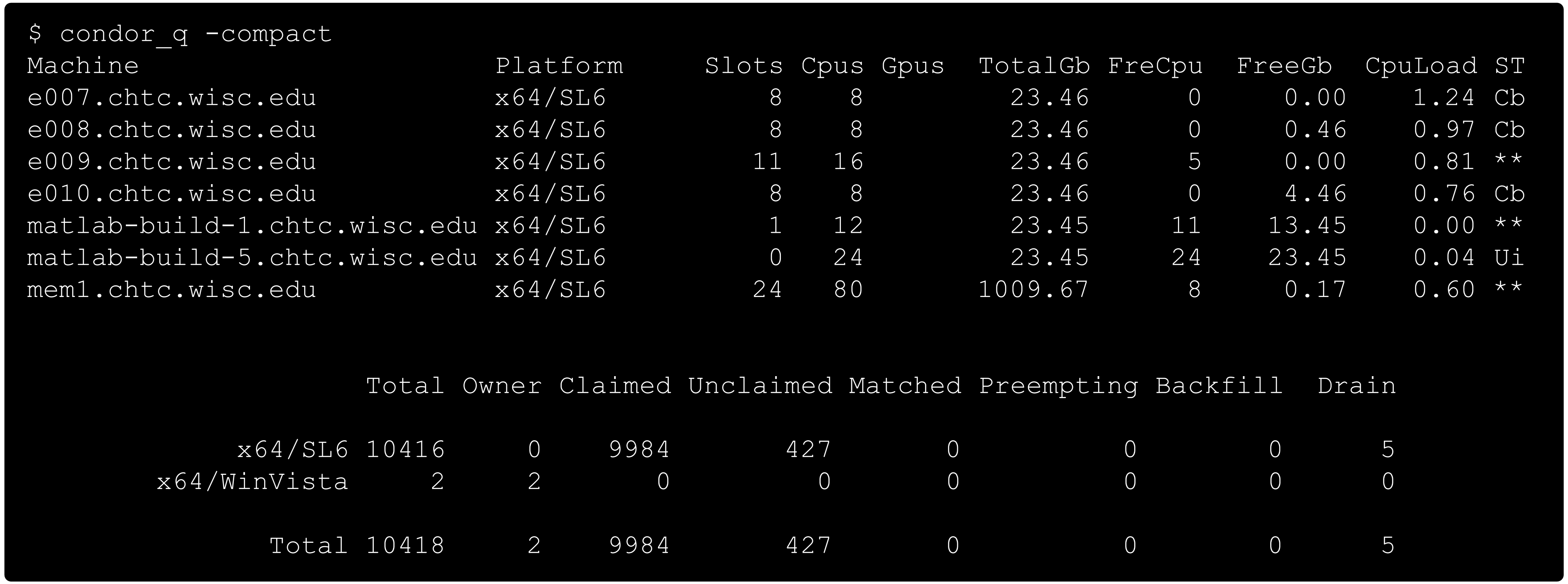
`condor_status` ***`[Machine] -af Attribute1 Attribute2 ...`***

```
$ condor_status -l slot1_1@c001.chtc.wisc.edu
HasFileTransfer = true
COLLECTOR_HOST_STRING = "cm.chtc.wisc.edu”
TargetType = "Job”
TotalTimeClaimedBusy = 43334c001.chtc.wisc.edu
UtsnameNodename = ""
Mips = 17902
MAX_PREEMPT = ( 3600 * ( 72 - 68 * ( WantGlidein =?= true ) ) )
Requirements = ( START ) && ( IsValidCheckpointPlatform ) && (
WithinResourceLimits )
State = "Claimed"
OpSysMajorVer = 6
OpSysName = "SL”
...
```

Machine Attributes

- To summarize, use the “-compact” option

  condor_status -compact



```
$ condor_q -compact
Machine                      Platform     Slots Cpus Gpus  TotalGb FreCpu  FreeGb  CpuLoad ST
e007.chtc.wisc.edu           x64/SL6          8    8         23.46      0     0.00    1.24 Cb
e008.chtc.wisc.edu           x64/SL6          8    8         23.46      0     0.46    0.97 Cb
e009.chtc.wisc.edu           x64/SL6         11   16         23.46      5     0.00    0.81 **
e010.chtc.wisc.edu           x64/SL6          8    8         23.46      0     4.46    0.76 Cb
matlab-build-1.chtc.wisc.edu x64/SL6          1   12         23.45     11    13.45    0.00 **
matlab-build-5.chtc.wisc.edu x64/SL6          0   24         23.45     24    23.45    0.04 Ui
mem1.chtc.wisc.edu           x64/SL6         24   80       1009.67      8     0.17    0.60 **


                     Total Owner Claimed Unclaimed Matched Preempting Backfill  Drain

             x64/SL6 10416     0    9984       427       0          0        0      5
        x64/WinVista     2     2       0         0       0          0        0      0

               Total 10418     2    9984       427       0          0        0      5
```

Job Universes

- HTCondor has different “universes” for running specialized job types

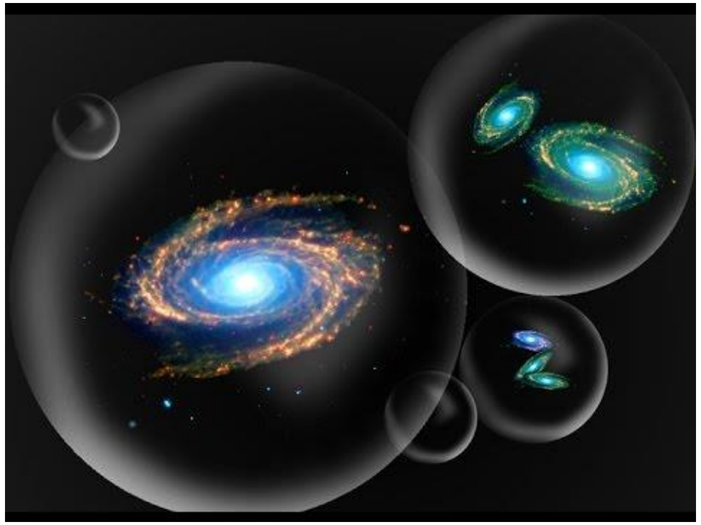
  HTCondor Manual: Choosing an HTCondor Universe

- Vanilla (default)
  - good for most software

  HTCondor Manual: Vanilla Universe

- Set in the submit file using:

```
universe =
vanilla
```



Other Universes

- Local
  - Run jobs on the submit node
- Container
  - Runs jobs inside a container
  - Container image can be specified by user or by admin
- Grid
  - Delegate jobs to another scheduler (*e.g.* SLURM, PBS, …)
  - The basis for HTCondor-CE

Other (Less Popular) Universes

- VM
  - Run jobs inside a virtual machine
- Parallel
  - Used for coordinating jobs across multiple servers (e.g. MPI code)
  - Not necessary for single server multi-core jobs

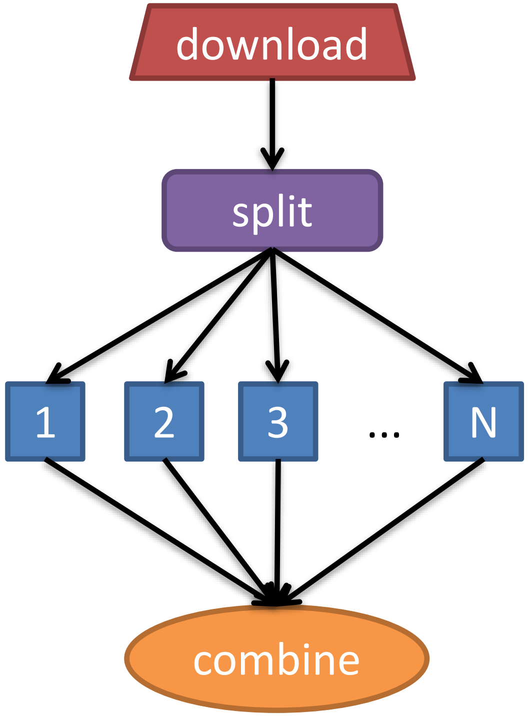
Typical User Command-Line Tools

• condor_submit	Submit new Jobs
• condor_status	View Ads in the Collector (e.g. EP Slots)
• condor_q	View Jobs at an AP
• condor_q -analyze	Why job/machines fail to match?
• condor_ssh_to_job	Create ssh session to active job
• condor_submit -i	Submit interactive job
• condor_hold / release	Hold a job, or release a held job
• condor_run	Submit and block
• condor_rm	Remove Jobs
• condor_prio	Intra-User Job Prios
• condor_history	Completed Job Info
• condor_submit_dag	Submit new DAG workflow
• condor_chirp	Access files/ad from active job

Describing Workflows with DAGMan

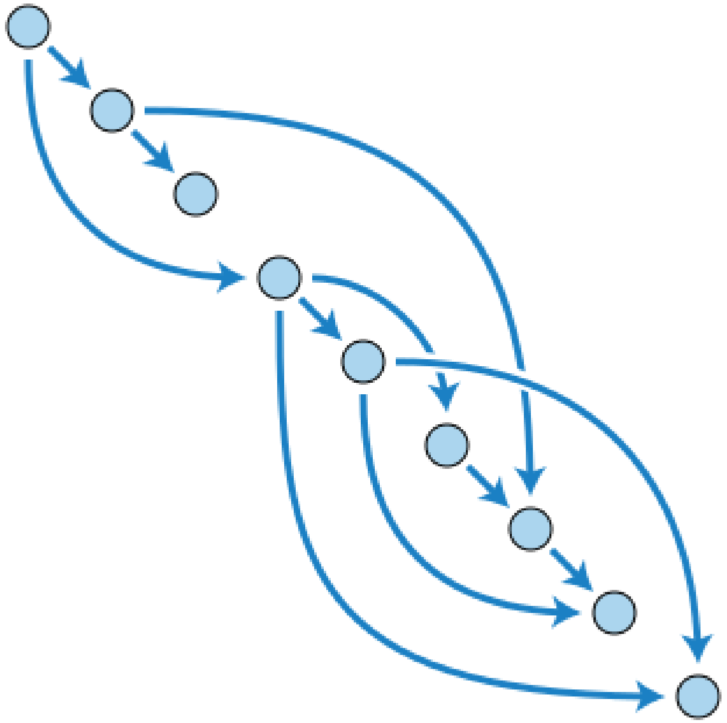
Workflows

- Problem: Want to submit jobs in a particular order, with dependencies between groups of jobs
- Solution: Write a DAG



DAG = ”directed acyclic graph”

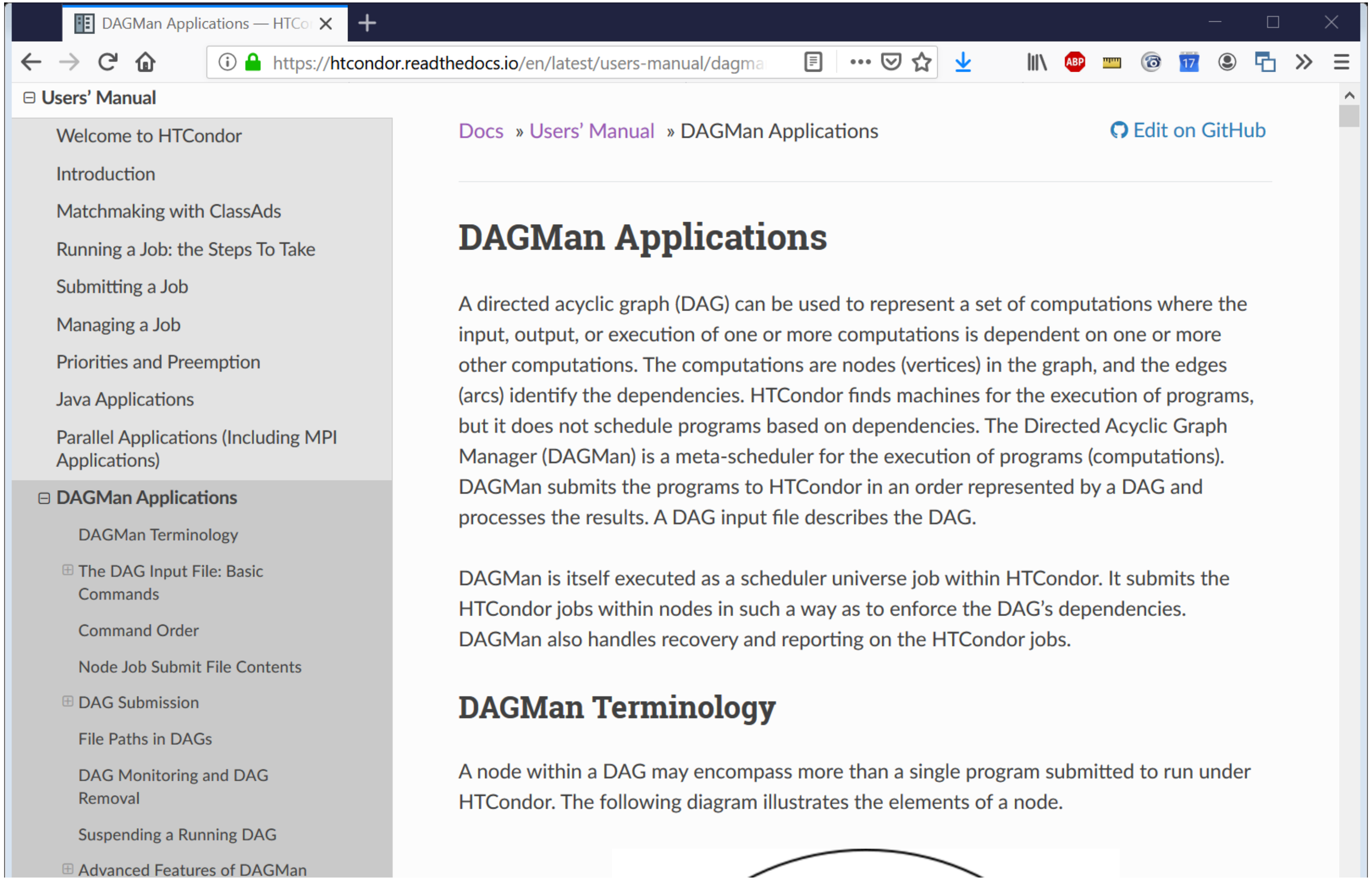
- topological ordering of vertices (“**nodes**”) is established by directional connections (“**edges**”)
- “acyclic” aspect requires a start and end, with no looped repetition
  - can contain cyclic subcomponents, covered in later slides for workflows



Wikimedia Commons

wikipedia.org/wiki/Directed_acyclic_graph

DAGMan in the HTCondor Manual


DAGMan Applications — HTCo
https://htcondor.readthedocs.io/en/latest/users-manual/dagma
⊟ Users' Manual
Welcome to HTCondor
Introduction
Matchmaking with ClassAds
Running a Job: the Steps To Take
Submitting a Job
Managing a Job
Priorities and Preemption
Java Applications
Parallel Applications (Including MPI Applications)
⊟ DAGMan Applications
DAGMan Terminology
⊞ The DAG Input File: Basic Commands
Command Order
Node Job Submit File Contents
⊞ DAG Submission
File Paths in DAGs
DAG Monitoring and DAG Removal
Suspending a Running DAG
⊞ Advanced Features of DAGMan
Docs » Users' Manual » DAGMan Applications
Edit on GitHub

DAGMan Applications

A directed acyclic graph (DAG) can be used to represent a set of computations where the input, output, or execution of one or more computations is dependent on one or more other computations. The computations are nodes (vertices) in the graph, and the edges (arcs) identify the dependencies. HTCondor finds machines for the execution of programs, but it does not schedule programs based on dependencies. The Directed Acyclic Graph Manager (DAGMan) is a meta-scheduler for the execution of programs (computations). DAGMan submits the programs to HTCondor in an order represented by a DAG and processes the results. A DAG input file describes the DAG.

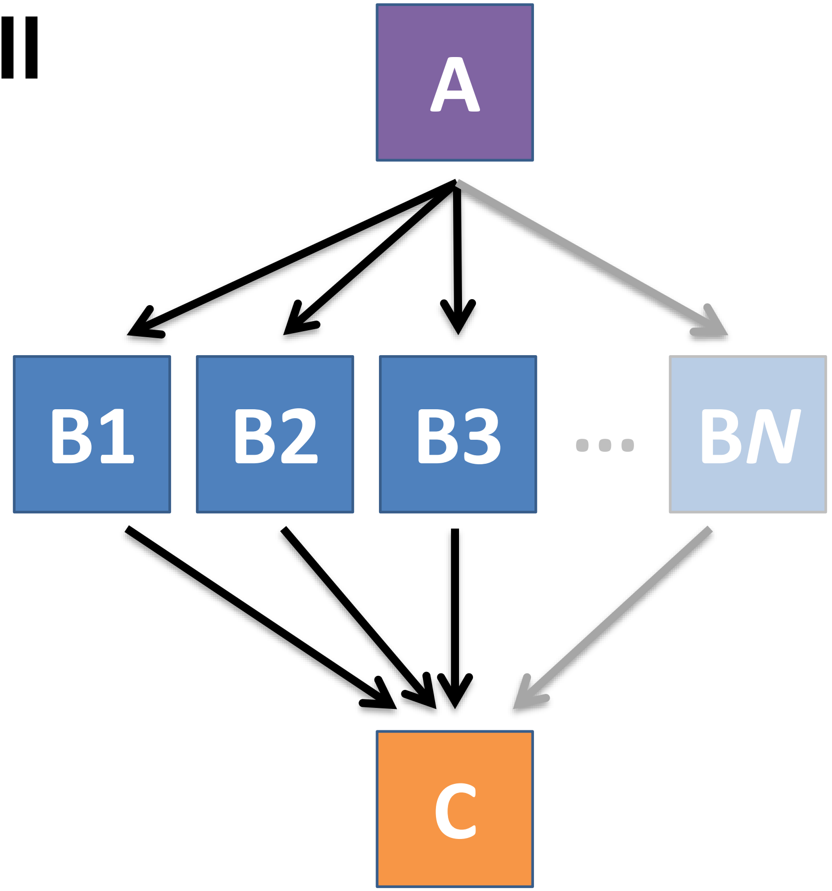
DAGMan is itself executed as a scheduler universe job within HTCondor. It submits the HTCondor jobs within nodes in such a way as to enforce the DAG's dependencies. DAGMan also handles recovery and reporting on the HTCondor jobs.

DAGMan Terminology

A node within a DAG may encompass more than a single program submitted to run under HTCondor. The following diagram illustrates the elements of a node.

Simple Example for this Tutorial

- **The DAG input file will** communicate the “nodes” and directional “edges” of the DAG

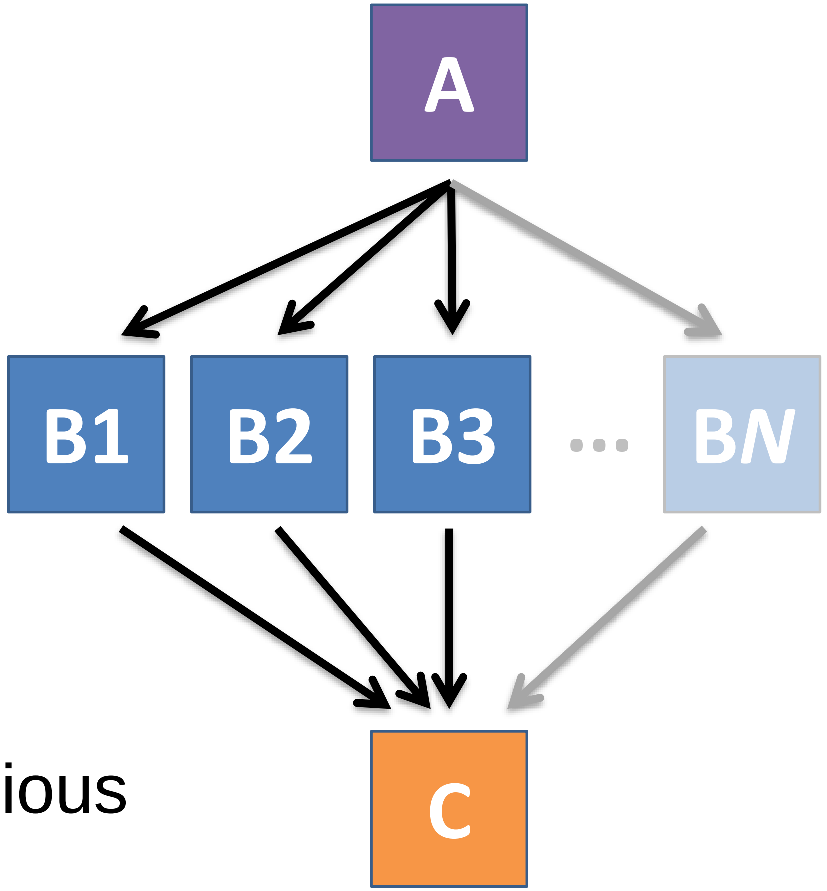


HTCondor Manual: DAGMan Applications > DAG Input File

Basic DAG input file: JOB nodes, PARENT-CHILD edges

my.dag

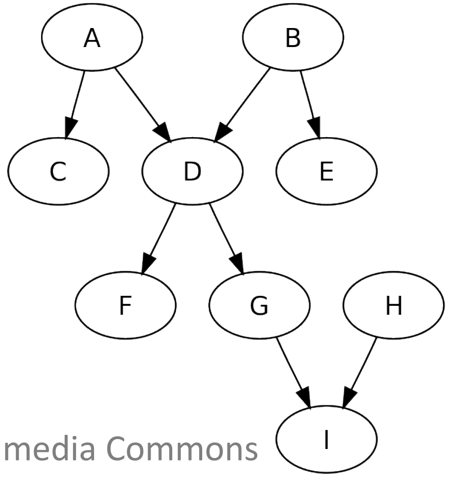
```
JOB A A.sub
JOB B1 B1.sub
JOB B2 B2.sub
JOB B3 B3.sub
JOB C C.sub
PARENT A CHILD B1 B2 B3
PARENT B1 B2 B3 CHILD C
```



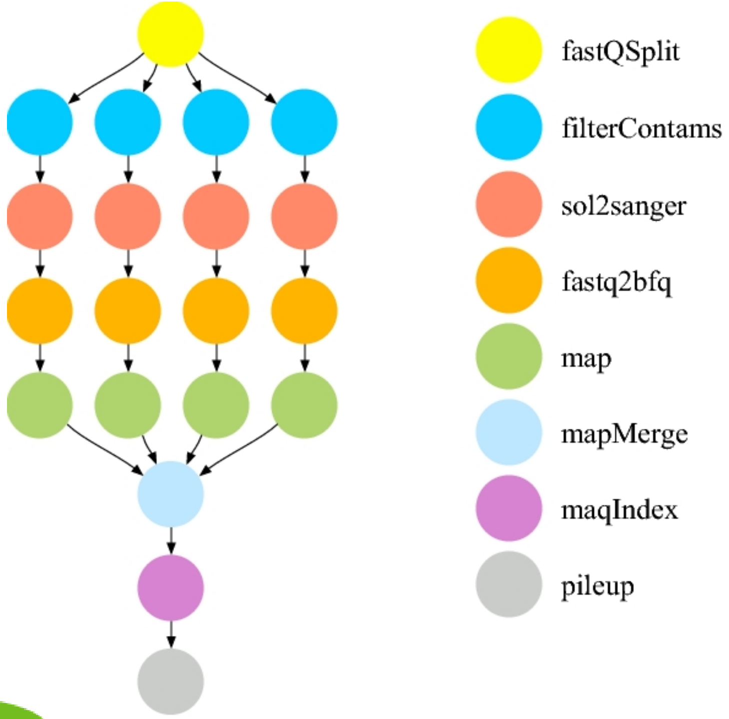
- Node names are used by various DAG features to modify their execution by DAG Manager.

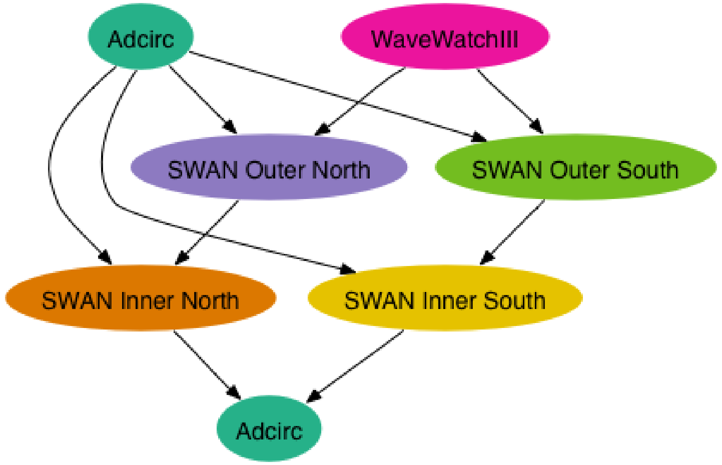
HTCondor Manual: DAGMan Applications > DAG Input File

Endless Workflow Possibilities



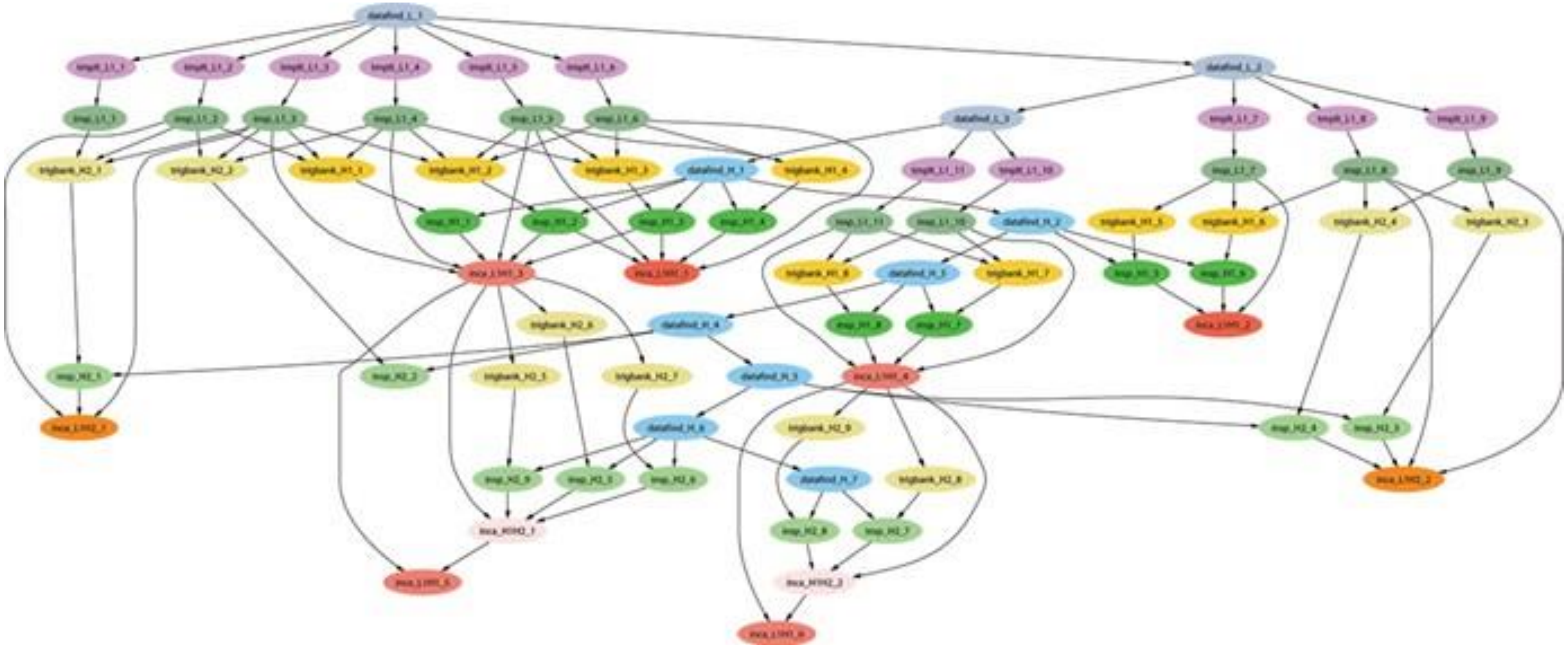
Wikimedia Commons





https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator

Endless Workflow Possibilities



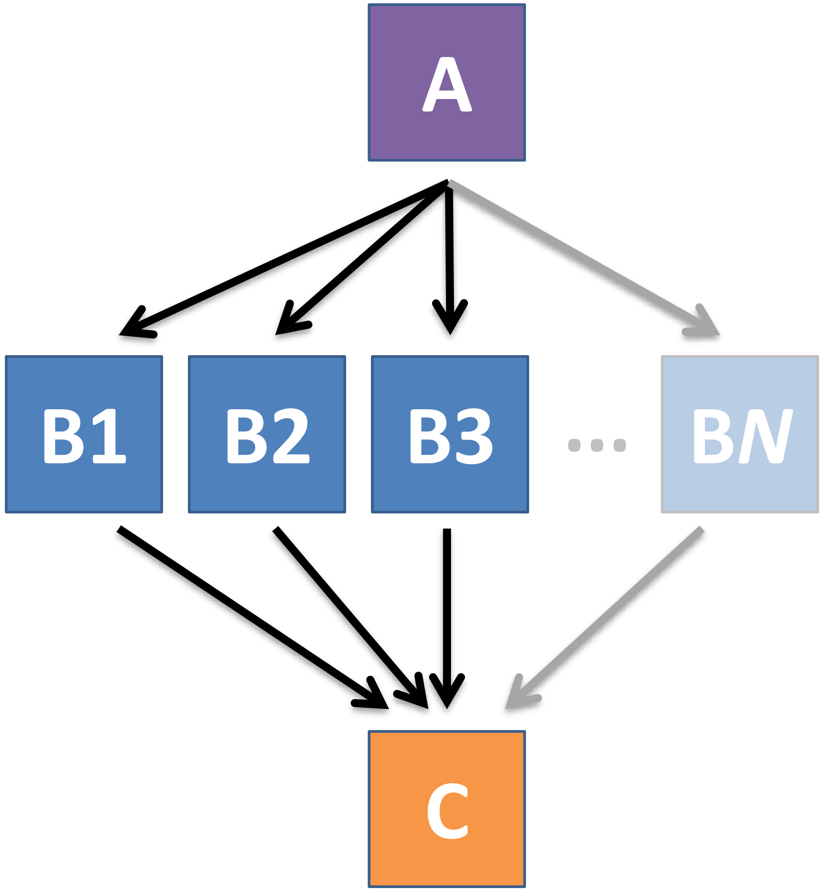
https://confluence.pegasus.isi.edu

Submitting and Monitoring a DAGMan Workflow

Basic DAG input file: JOB nodes, PARENT-CHILD edges

my.dag

```
JOB A A.sub
JOB B1 B1.sub
JOB B2 B2.sub
JOB B3 B3.sub
JOB C C.sub
PARENT A CHILD B1 B2 B3
PARENT B1 B2 B3 CHILD C
```



HTCondor Manual: DAGMan Applications > DAG Input File

Submitting a DAG to the queue

- Submission command:

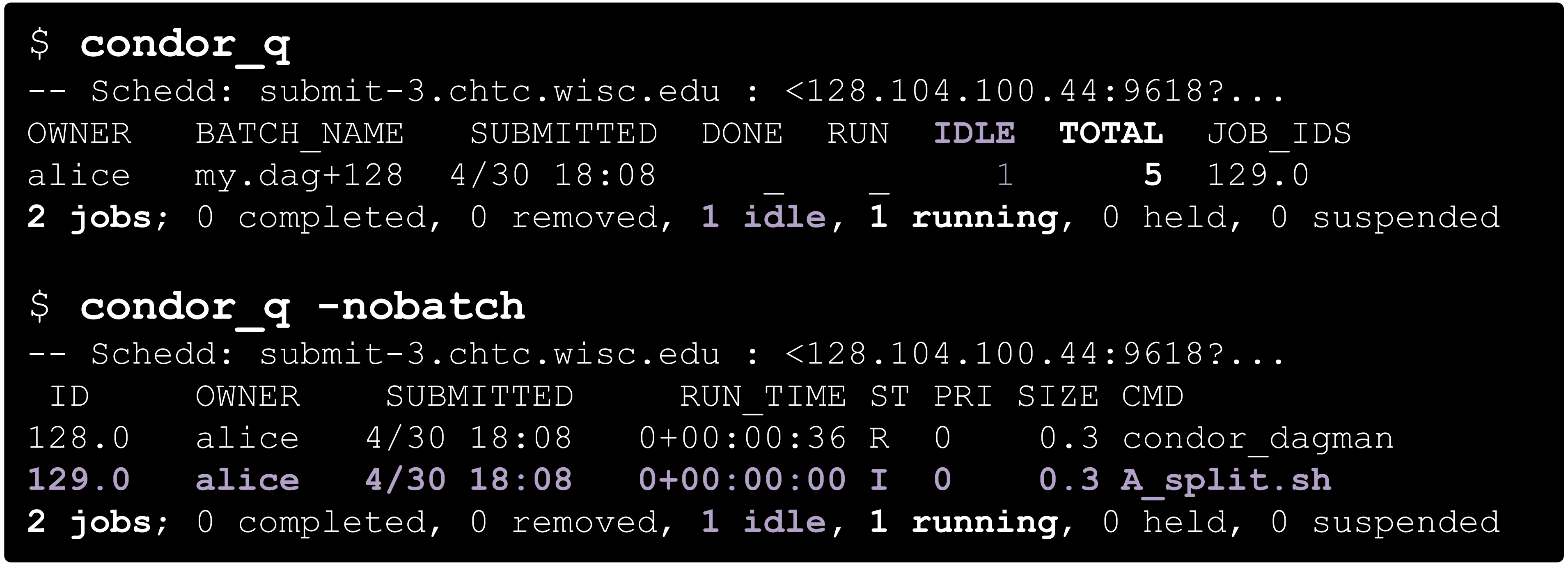
condor_submit_dag *dag_file*

```
$ condor_submit_dag my.dag

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

Submitting job(s).
1 job(s) submitted to cluster 87274940.
------------------------------------------------------------------
```

HTCondor Manual: DAGMan > DAG Submission

Jobs are automatically submitted by the DAGMan job

- Seconds later, node **A** is submitted:

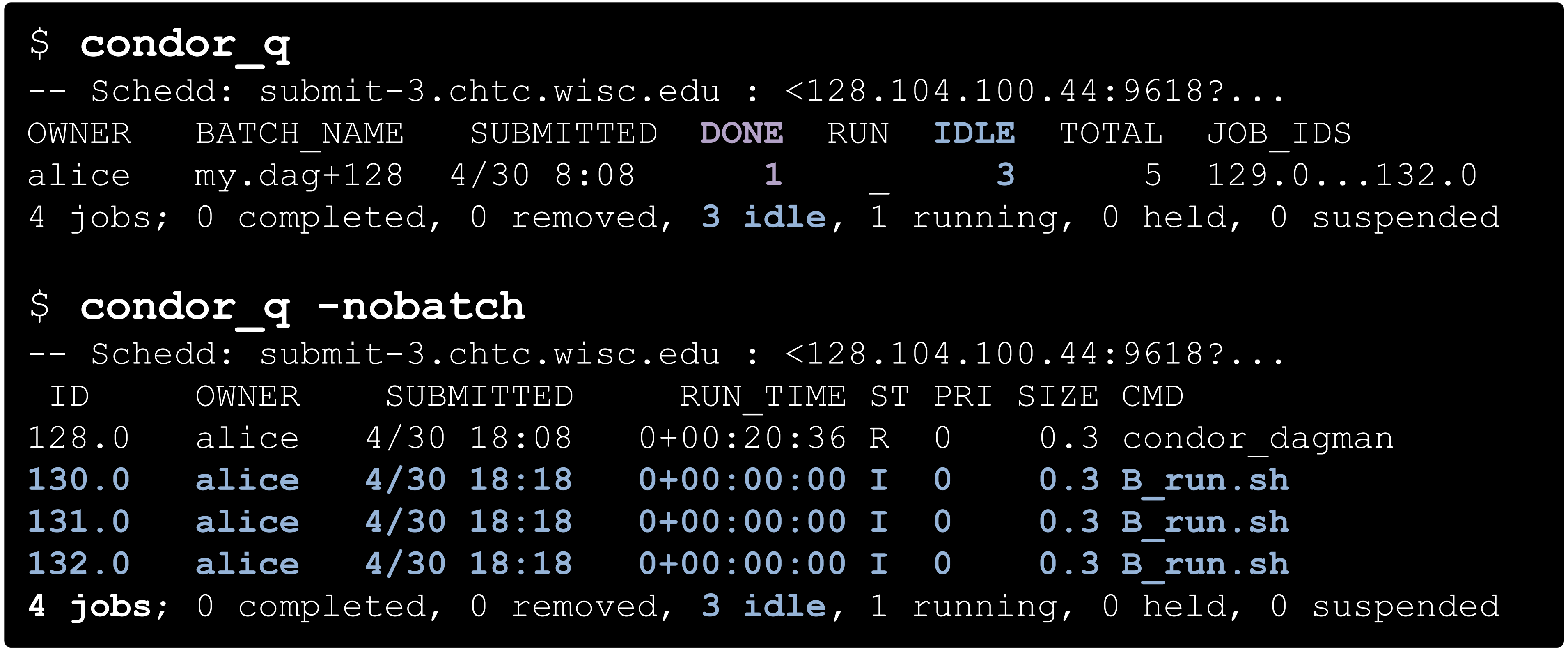


```
$ condor_q
-- Schedd: submit-3.chtc.wisc.edu : <128.104.100.44:9618?...
OWNER   BATCH_NAME   SUBMITTED  DONE  RUN  IDLE  TOTAL  JOB_IDS
alice   my.dag+128  4/30 18:08     _    _     1      5  129.0
2 jobs; 0 completed, 0 removed, 1 idle, 1 running, 0 held, 0 suspended

$ condor_q -nobatch
-- Schedd: submit-3.chtc.wisc.edu : <128.104.100.44:9618?...
 ID      OWNER    SUBMITTED     RUN_TIME ST PRI SIZE CMD
128.0   alice   4/30 18:08   0+00:00:36 R  0    0.3 condor_dagman
129.0   alice   4/30 18:08   0+00:00:00 I  0    0.3 A_split.sh
2 jobs; 0 completed, 0 removed, 1 idle, 1 running, 0 held, 0 suspended
```

HTCondor Manual: DAGMan > DAG Submission

Jobs are automatically submitted by the DAGMan job

- After **A** completes, **B1-3** are submitted



```
$ condor_q
-- Schedd: submit-3.chtc.wisc.edu : <128.104.100.44:9618?...
OWNER   BATCH_NAME   SUBMITTED  DONE  RUN  IDLE  TOTAL  JOB_IDS
alice   my.dag+128  4/30 8:08      1    _     3      5  129.0...132.0
4 jobs; 0 completed, 0 removed, 3 idle, 1 running, 0 held, 0 suspended

$ condor_q -nobatch
-- Schedd: submit-3.chtc.wisc.edu : <128.104.100.44:9618?...
 ID     OWNER    SUBMITTED     RUN_TIME ST PRI SIZE CMD
128.0   alice   4/30 18:08   0+00:20:36 R  0    0.3 condor_dagman
130.0   alice   4/30 18:18   0+00:00:00 I  0    0.3 B_run.sh
131.0   alice   4/30 18:18   0+00:00:00 I  0    0.3 B_run.sh
132.0   alice   4/30 18:18   0+00:00:00 I  0    0.3 B_run.sh
4 jobs; 0 completed, 0 removed, 3 idle, 1 running, 0 held, 0 suspended
```

HTCondor Manual: DAGMan > DAG Submission

Jobs are automatically submitted by the DAGMan job

- After **B1-3** complete, node **C** is submitted

```
$ condor_q
-- Schedd: submit-3.chtc.wisc.edu : <128.104.100.44:9618?...
OWNER   BATCH_NAME   SUBMITTED  DONE  RUN  IDLE  TOTAL  JOB_IDS
alice   my.dag+128  4/30 8:08      4    _     1      5  129.0...133.0
2 jobs; 0 completed, 0 removed, 1 idle, 1 running, 0 held, 0 suspended

$ condor_q -nobatch
-- Schedd: submit-3.chtc.wisc.edu : <128.104.100.44:9618?...
 ID     OWNER    SUBMITTED     RUN_TIME ST PRI SIZE CMD
128.0   alice   4/30 18:08   0+00:46:36 R  0    0.3 condor_dagman
133.0   alice   4/30 18:54   0+00:00:00 I  0    0.3 C_combine.sh
2 jobs; 0 completed, 0 removed, 1 idle, 1 running, 0 held, 0 suspended
```

HTCondor Manual: DAGMan > DAG Submission

Removing a DAG from the queue

- Remove the DAGMan job in order to stop and remove the entire DAG:

  `condor_rm dagman_jobID`

- Creates a **rescue file** so that only incomplete or unsuccessful NODES are repeated upon resubmission

```
$ condor_q
-- Schedd: submit-3.chtc.wisc.edu : <128.104.100.44:9618?...
OWNER    BATCH_NAME   SUBMITTED  DONE   RUN   IDLE   TOTAL  JOB_IDS
alice    my.dag+128  4/30 8:08      4     _      1       6  129.0...133.0
2 jobs; 0 completed, 0 removed, 1 idle, 1 running, 0 held, 0 suspended
$ condor_rm 128
All jobs in cluster 128 have been marked for removal
```

[DAGMan > DAG Monitoring and DAG Removal]

[DAGMan > The Rescue DAG]

Rescue Files For Resuming a Failed DAG

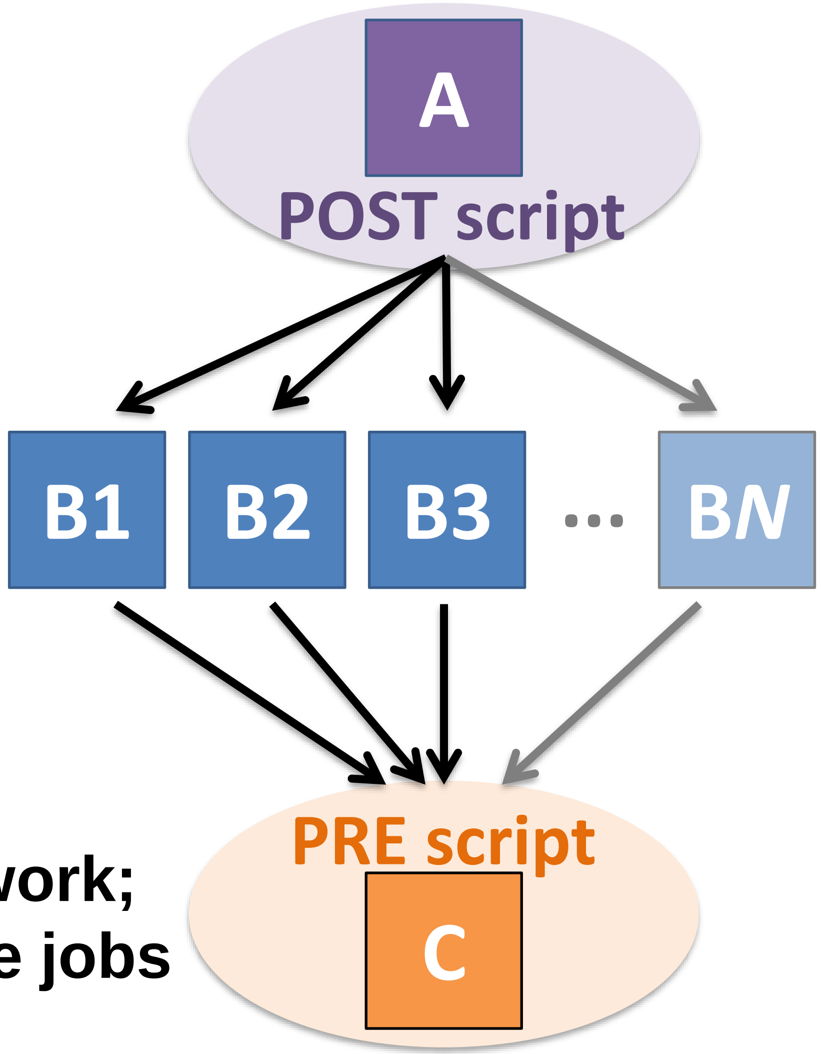
- A rescue file is created when:
  - a node fails, and after DAGMan advances through any other possible nodes
  - the DAG is removed from the queue (or **aborted**; covered later)
  - the DAG is **halted** and not unhalted
- Resubmission uses the rescue file (if it exists) when the original DAG file is resubmitted
  - override: **`condor_submit_dag`** ***`dag_file -f`***

DAGMan > The Rescue DAG

PRE and POST scripts run on the submit server, as part of the node

my.dag

```
JOB A A.sub
SCRIPT POST A sort.sh
JOB B1 B1.sub
JOB B2 B2.sub
JOB B3 B3.sub
JOB C C.sub
SCRIPT PRE C tar_it.sh
PARENT A CHILD B1 B2 B3
PARENT B1 B2 B3 CHILD C
```



- **Use sparingly for lightweight work; otherwise include work in node jobs**

[HTCondor Manual: DAGMan Applications > DAG Input File]

RETRY failed nodes to overcome transient errors

- Retry a node up to *N* times if the exit code is non-zero:

RETRY *node_name N*

Example:

```
JOB A A.sub
RETRY A 5
JOB B B.sub
PARENT A CHILD B
```

- See also: retry except for a particular exit code (`UNLESS-EXIT`), or retry scripts (`DEFER`)
- **Note:** Unnecessary for nodes (jobs) that can use `max_retries` in the submit file

DAGMan Applications > Advanced Features > Retrying

DAGMan Applications > DAG Input File > SCRIPT

RETRY applies to whole node, including PRE/POST scripts

- PRE and POST scripts are included in retries
- RETRY of a node with a POST script uses the exit code from the POST script (not from the job)
  - – POST script can do more to determine node success, perhaps by examining JOB output

Example:

```
SCRIPT PRE A download.sh
JOB A A.sub
SCRIPT POST A checkA.sh
RETRY A 5
```

DAGMan Applications > Advanced Features > Retrying

DAGMan Applications > DAG Input File > SCRIPT

Modular Organization and Control of DAG Components

- Splices and SubDags
- Node Throttling
- Node Priorities
- Lots more in the Manual…

Thank you!

Questions?

Join us on the htcondor-users email list!

https://htcondor.org/mail-lists/#user

This work is supported by NSF under Cooperative Agreement OAC-2030508 as part of the PATh Project. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF

PATh PARTNERSHIP to ADVANCE
THROUGHPUT
COMPUTING