Open Science Grid

HTC Job Execution with HTCondor

Tuesday, July 14

Lauren Michael

Overview

- How does the HTCondor job scheduler work?
- How do you run, monitor, and review jobs?
- Best ways to submit multiple jobs (what we’re here for, *right?*)
- Testing, tuning, and troubleshooting to scale up.

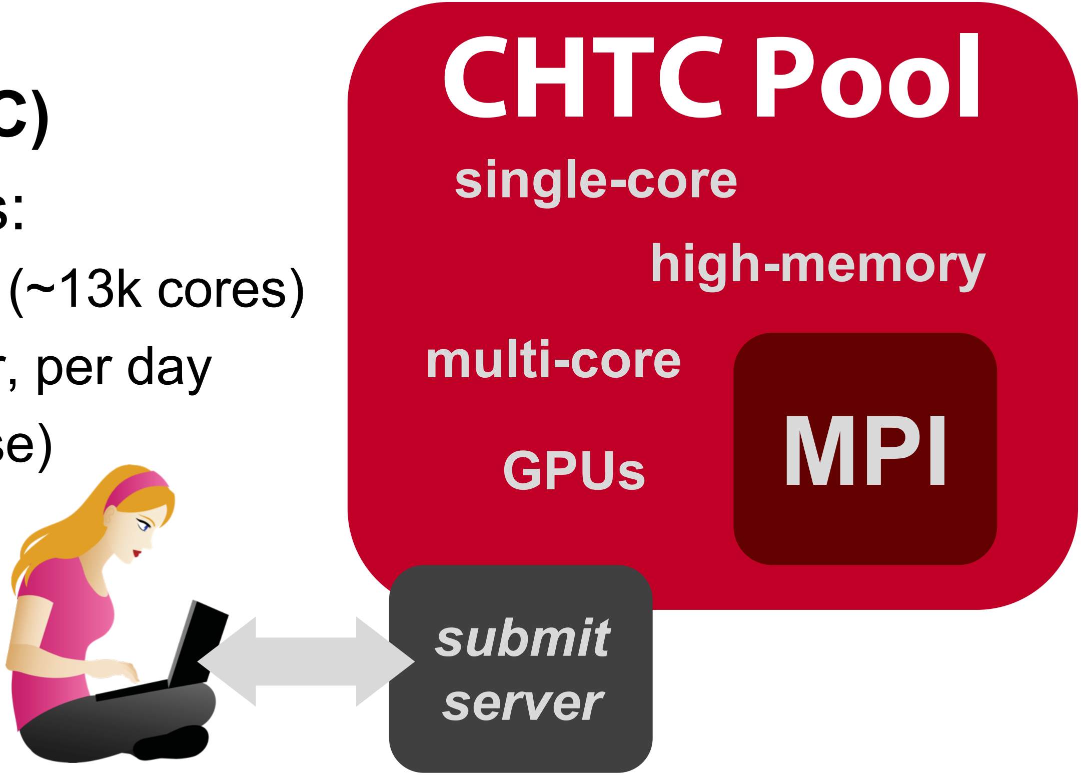
Example Local Cluster

- UW-Madison’s **Center for High Throughput Computing (CHTC)**
- Recent CPU hours:

  ~120 million hrs/year (~13k cores)

  up to 15,000 per user, per day

  (~600 cores in use)

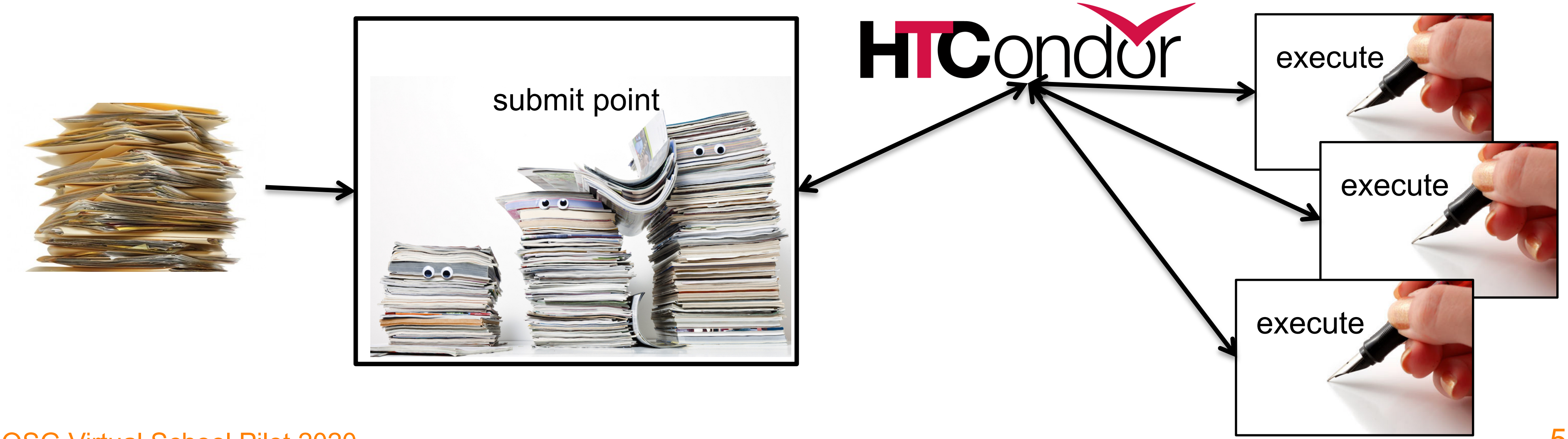


HTCondor History and Status

- History
  - Started in 1988 as a “cycle scavenger”
- Today
  - Developed within the CHTC by professional developers
  - Used all over the world, by:
    - Campuses, national labs, Einstein/Folding@Home
    - Dreamworks, Boeing, SpaceX, investment firms, …
    - **The Open Science Grid!!**
- Miron Livny,
  - Professor, UW-Madison Computer Sciences
  - CHTC Director, HTCondor PI, OSG Technical Director

HTCondor -- How It Works

- Submit tasks to a queue (on a ***submit server***)
- HTCondor schedules them to run on computers (***execute server***)



Terminology: *Job*

- ***Job*: An independently-scheduled unit of computing work**
- Three main pieces:

  Executable: the script or program to run

  Input: any options (arguments) and/or file-based information

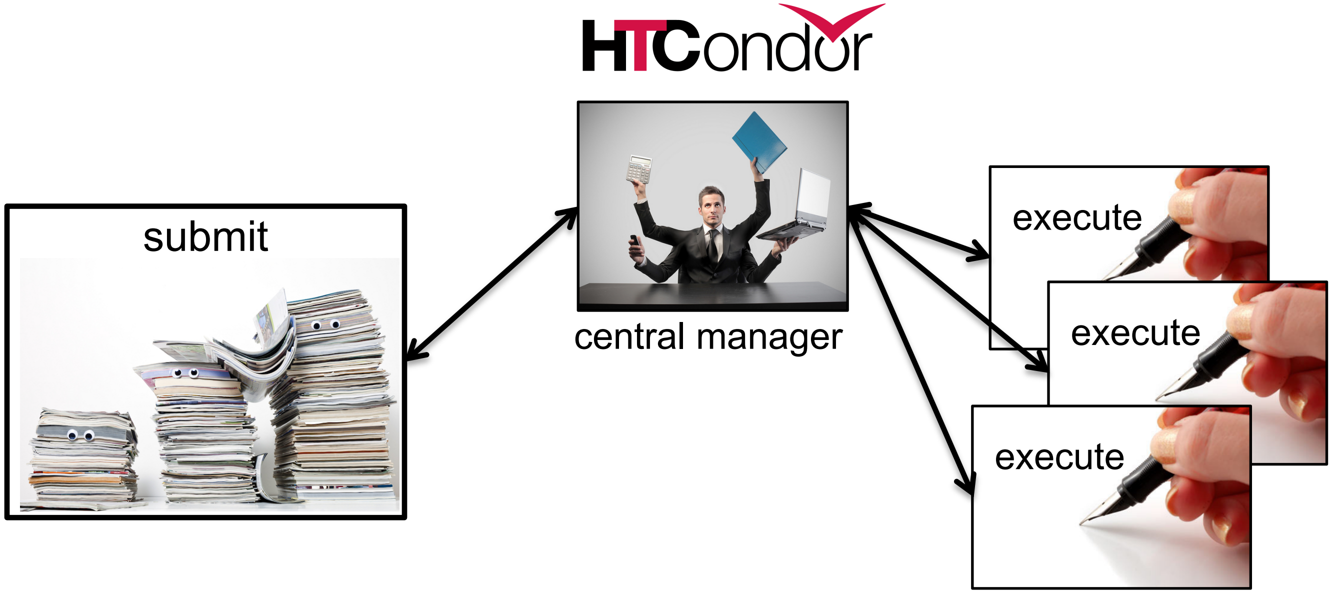
  Output: any files or screen information produced by the executable
- In order to run *many* jobs, executable must run on the command-line without any graphical input from the user

Terminology: *Machine, Slot*

- ***Machine***
  - A whole computer (desktop or server)
  - Has multiple processors (***CPU cores***), some amount of **memory**, and some amount of file space (**disk**)
- ***Slot***
  - **an assignable unit of a machine (i.e. 1 job per slot)**
  - most often, corresponds to one core with some memory and disk
  - a typical machine will have multiple slots
- HTCondor can break up and create new slots, dynamically, as resources become available from completed jobs

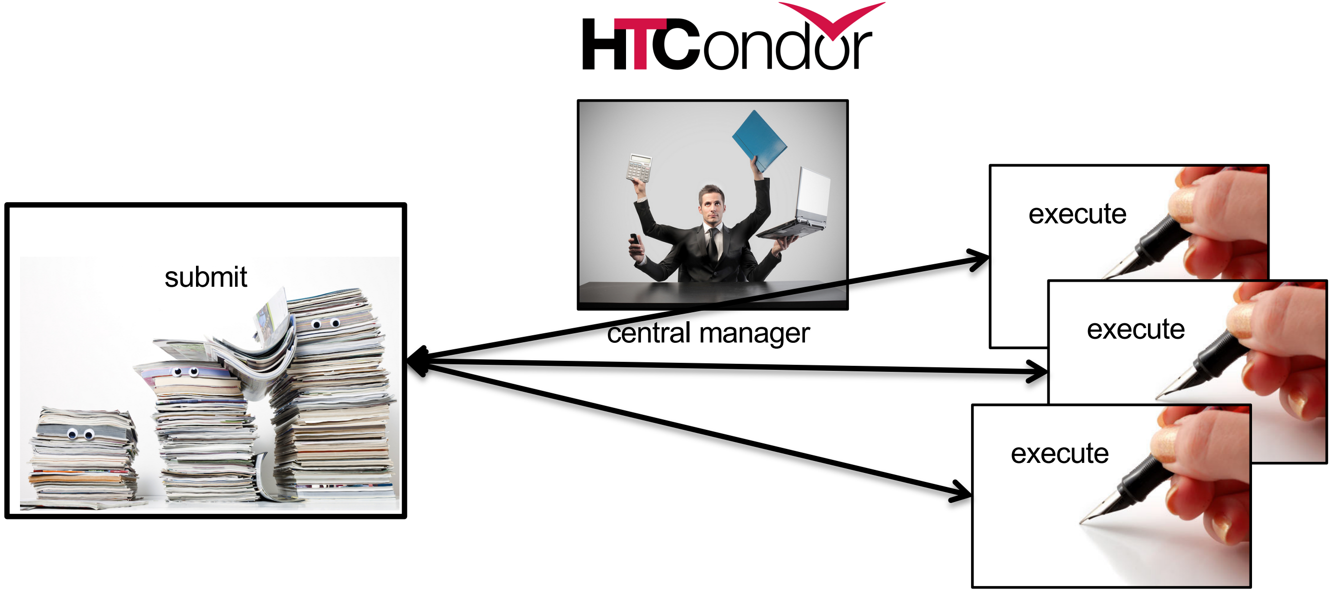
Job Matching

- On a regular basis, the central manager reviews ***Job*** and ***Machine*** attributes and matches jobs to ***Slots***.

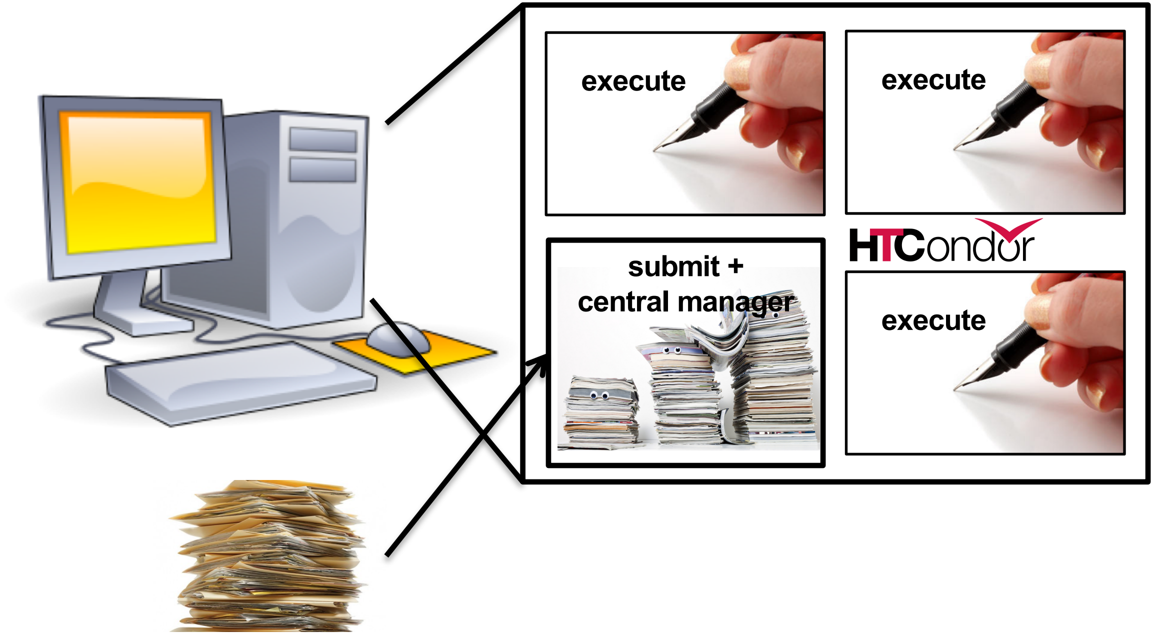


Job Execution

- (Then the submit and execute points communicate directly.)



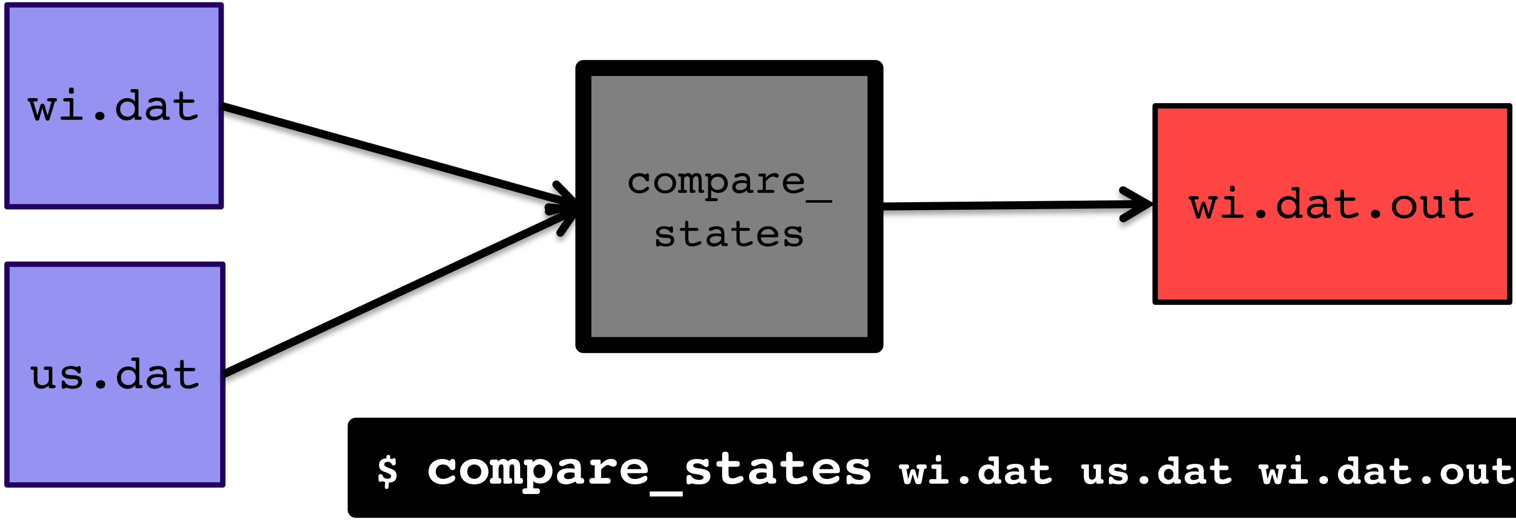
Single Computer



BASIC JOB SUBMISSION

Job Example

- program called “compare_states” (executable), which compares two data files (input) and produces a single output file.



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

Open Science Grid

Basic Submit File

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

transfer_input_files = us.dat, wi.dat

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

request_cpus = 1
request_disk = 20MB
request_memory = 20MB

queue 1
```

Basic Submit File

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

transfer_input_files = us.dat, wi.dat

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

request_cpus = 1
request_disk = 20MB
request_memory = 20MB

queue 1
```

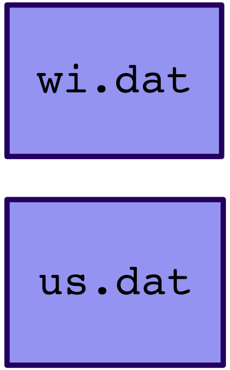
- List your **executable** and any **arguments** it takes
- Arguments are any options passed to the executable from the command line

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

Basic Submit File

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

transfer_input_files = us.dat, wi.dat

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

request_cpus = 1
request_disk = 20MB
request_memory = 20MB

queue 1
```

- comma-separated list of **input files to transfer** to the slot



Basic Submit File

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

transfer_input_files = us.dat, wi.dat

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 (output) from the job, automatically.

wi.dat.out

Basic Submit File

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

transfer_input_files = us.dat, wi.dat

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
  - *Explored in exercises!*
- **`output`/`error`**: captures stdout and stderr from your program (what would otherwise be printed to the terminal)

Basic Submit File

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

transfer_input_files = us.dat, wi.dat

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

request_cpus = 1
request_disk = 20MB
request_memory = 20MB

queue 1
```

- **`request`** the resources your job needs.
  - *More on this later!*
- **`queue`**: keyword indicating “create 1 job”

SUBMITTING AND MONITORING

Submitting and Monitoring

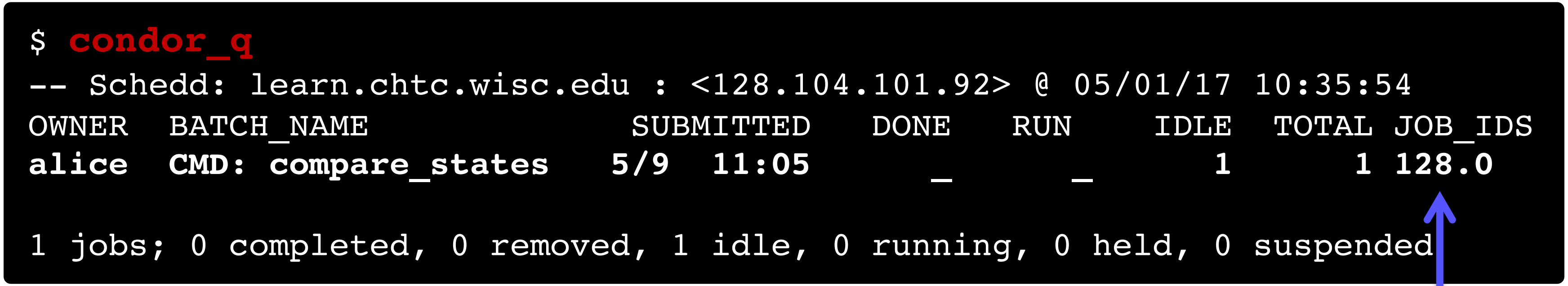
- To submit a job/jobs: `condor_submit` ***`submit_file`***
- To monitor submitted jobs: `condor_q`

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

$ condor_q
-- Schedd: learn.chtc.wisc.edu : <128.104.101.92> @ 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 your jobs only and batches jobs that were submitted together:



```
$ condor_q
-- Schedd: learn.chtc.wisc.edu : <128.104.101.92> @ 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
```

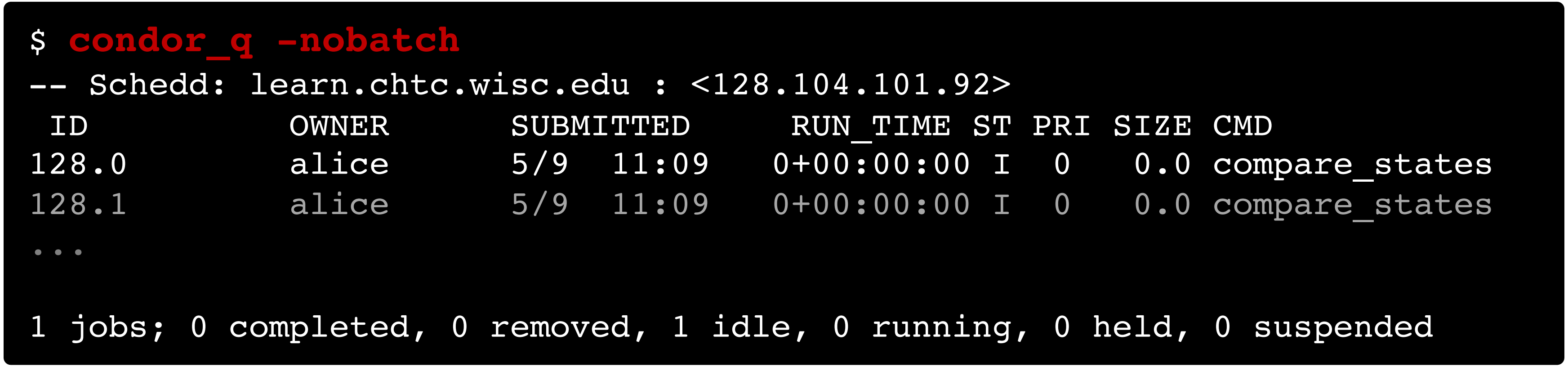
`JobId` = **`ClusterID.ProcID`**

- Limit **`condor_q`** by username, *`ClusterId`* or full *`JobId`*, (denoted `[U/C/J]` in following slides).

More about condor_q

- To see individual job details, use:

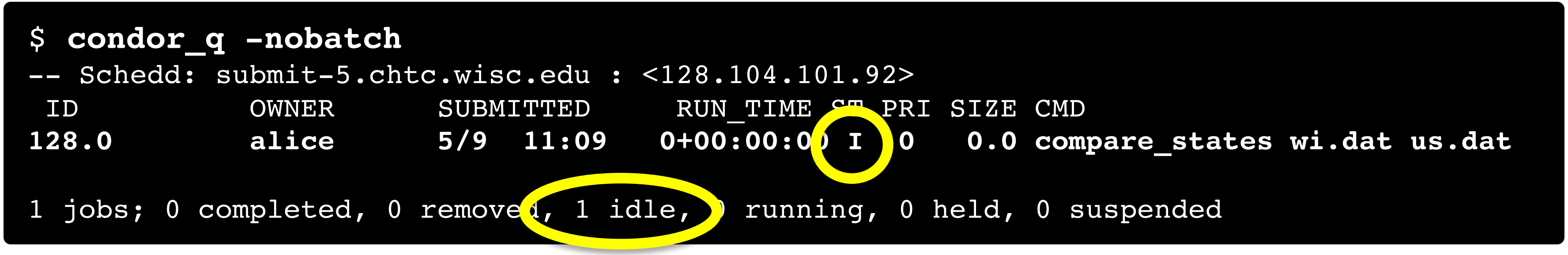
  condor_q –nobatch



```
$ condor_q -nobatch
-- Schedd: learn.chtc.wisc.edu : <128.104.101.92>
 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
128.1        alice      5/9  11:09   0+00:00:00 I  0    0.0 compare_states
...

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



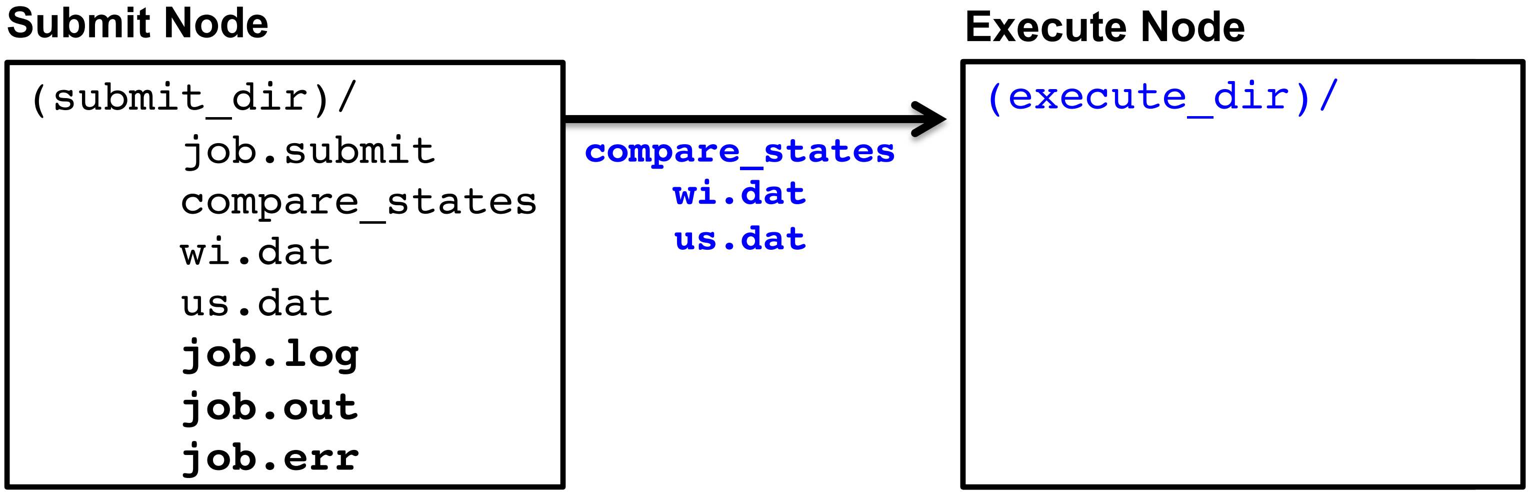
```
$ condor_q -nobatch
-- Schedd: submit-5.chtc.wisc.edu : <128.104.101.92>
 ID          OWNER        SUBMITTED       RUN_TIME  ST PRI SIZE CMD
128.0        alice        5/9  11:09    0+00:00:00  I  0    0.0 compare_states wi.dat us.dat

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

```
$ 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

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
```

compare_states
wi.dat
us.dat

Execute Node

```
(execute_dir)/
```

Job Running

```
$ condor_q -nobatch
-- Schedd: submit-5.chtc.wisc.edu : <128.104.101.92>
 ID          OWNER       SUBMITTED     RUN_TIME ST PRI SIZE CMD
128.0        alice       5/9  11:09   0+00:01:08 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

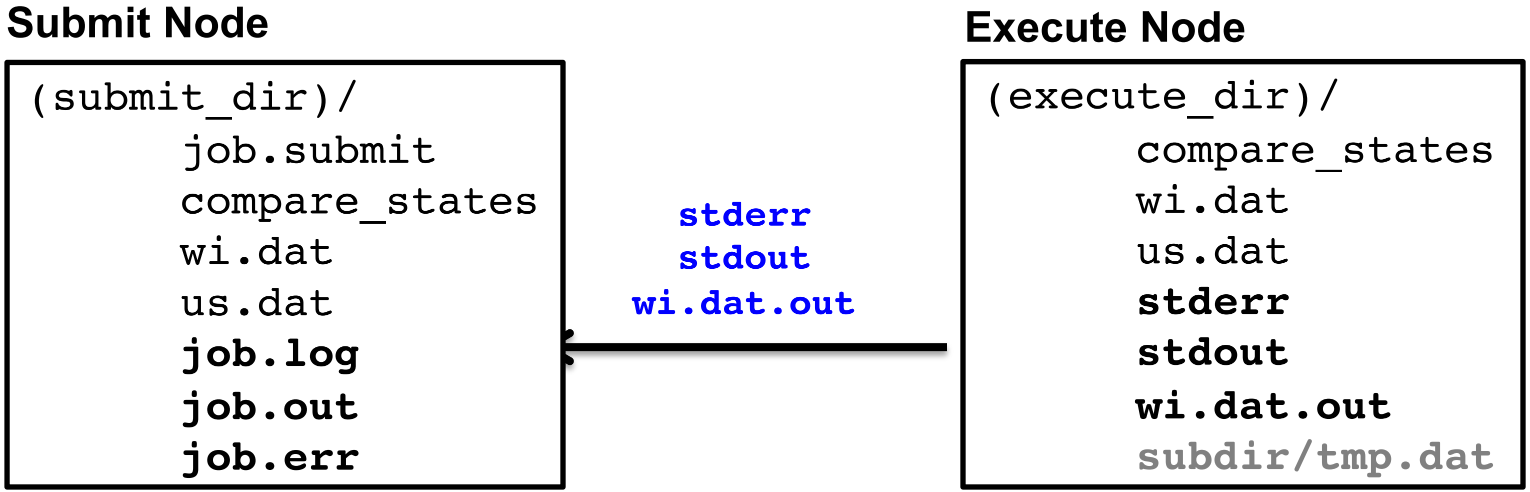
```
(submit_dir)/
        job.submit
        compare_states
        wi.dat
        us.dat
        job.log
        job.out
        job.err
```

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>
 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
```



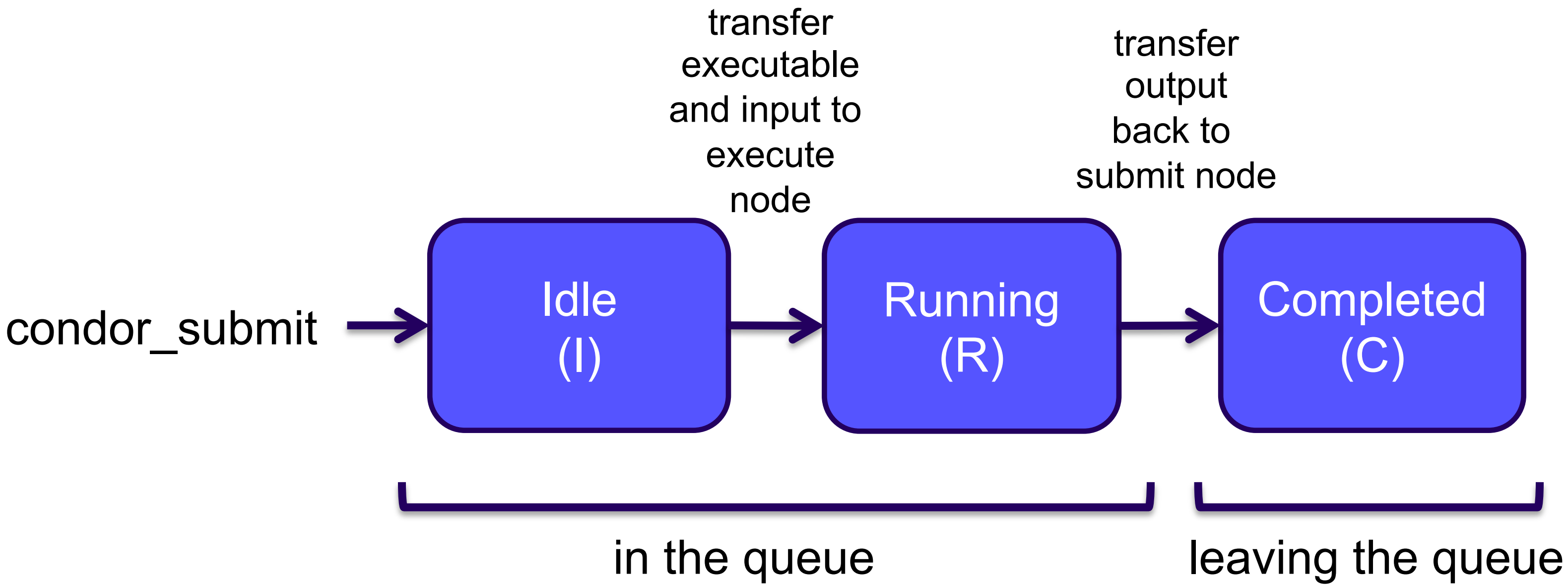
Job Completes (cont.)

```
$ 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 States



condor_submit

transfer executable and input to execute node

transfer output back to submit node

Idle (I)

Running (R)

Completed (C)

in the queue

leaving the queue

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
           Disk (KB)            :        14   20480  17203728
           Memory (MB)          :         1      20        20
```

Resource Request

- Jobs are nearly always using a ***part of*** a machine (a single slot), and not the whole thing
- Very important to request appropriate resources (***memory***, ***cpus***, ***disk***)
  - **requesting too little**: causes problems for your and other jobs; jobs might by 'held' by HTCondor
  - **requesting too much:** jobs will match to fewer "slots" than they could, and you'll block other jobs

Is it *OSG-able*?

Per-Job **Resources**	**Ideal Jobs!** (up to 10,000 cores, per user!)	**Still Very Advantageous!**	**Probably not…**
cores (GPUs)	**1** (1; non-specific)	**<8** (1; specific GPU type)	**>8 (or MPI)** (multiple)
Walltime *(per job)*	**<10 hrs*** *or checkpointable	**<20 hrs*** *or checkpointable	**>20 hrs**
RAM *(per job)*	**<few GB**	**<10 GB**	**>10 GB**
Input *(per job)*	**<500 MB**	**<10 GB**	**>10 GB**
Output *(per job)*	**<1 GB**	**<10 GB**	**>10 GB**
Software	*'portable' (pre-compiled binaries, transferable, containerizable, etc.)*	*most other than* →→→	*licensed software; non-Linux*

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
           Disk (KB)            :       14    20480  17203728
           Memory (MB)          :        1       20        20
```

SUBMITTING MULTIPLE JOBS

From one job ...

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.

One submit file per job (not recommended!)

job0.submit

```
executable = analyze.exe

arguments = file0.in file0.out
transfer_input_files = file0.in
output = job0.out
error = job0.err
queue
```

job1.submit

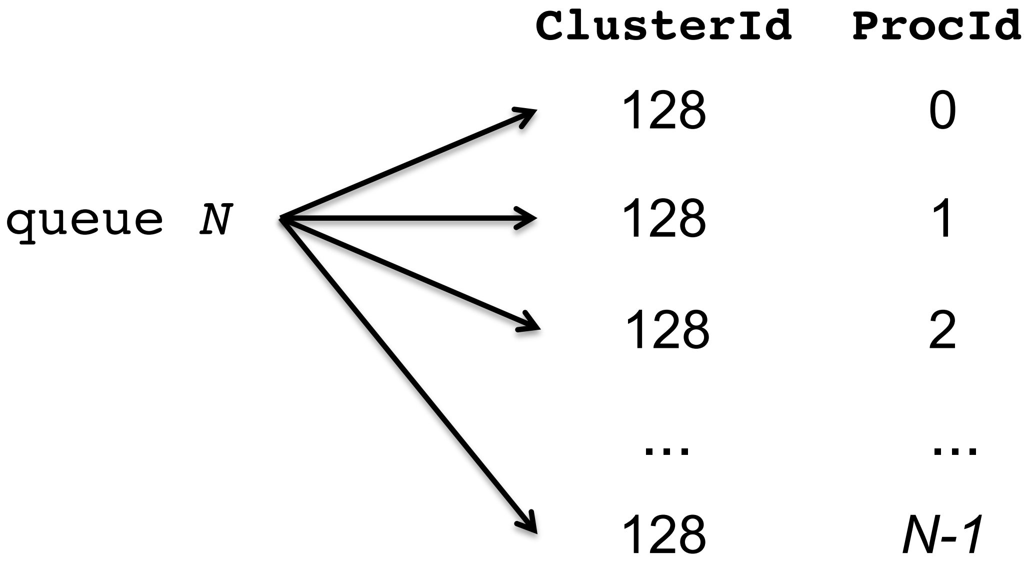
```
executable = analyze.exe

arguments = file1.in file1.out
transfer_input_files = file1.in
output = job1.out
error = job1.err
queue
```

(etc…)

(submit_dir)/

```
analyze.exe
file0.in
file1.in
file2.in
(etc.)

job0.submit
job1.submit
job2.submit
(etc.)
```

Automatic Variables



`queue N`

ClusterId	ProcId
128	0
128	1
128	2
…	…
128	*N-1*

Each job's **`ClusterId`** and **`ProcId`** numbers are autogenerated and saved as job attributes.

You can reference them inside the submit file using:*

- **`$(Cluster)`**
- **`$(Process)`**

* $(ClusterId) and $(ProcId) are also okay

Using $(Process) for Numbered Files

job.submit

```
executable = analyze.exe
arguments = file$(Process).in file$(Process).out
transfer_input_files = file$(Process).in

log = job_$(Cluster).log
output = job_$(Process).out
error = job_$(Process).err

queue 3
```

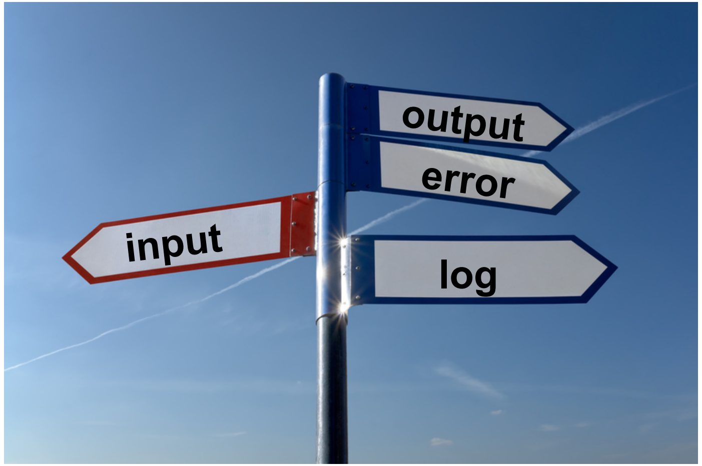
(submit_dir)/

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

job.submit
```

- $(Process) and $(Cluster) allow us to provide unique values to each job and submission!

Organizing Files in Sub-Directories

- Create sub-directories and use paths in the submit file to separate various input, error, log, and output files.



Use a Directory* per File Type

(submit_dir)/

```
job.submit      file0.out      input/          log/           err/
analyze.exe     file1.out        file0.in        job0.log       job0.err
                file2.out        file1.in        job1.log       job1.err
                                 file2.in        job2.log       job2.err
```

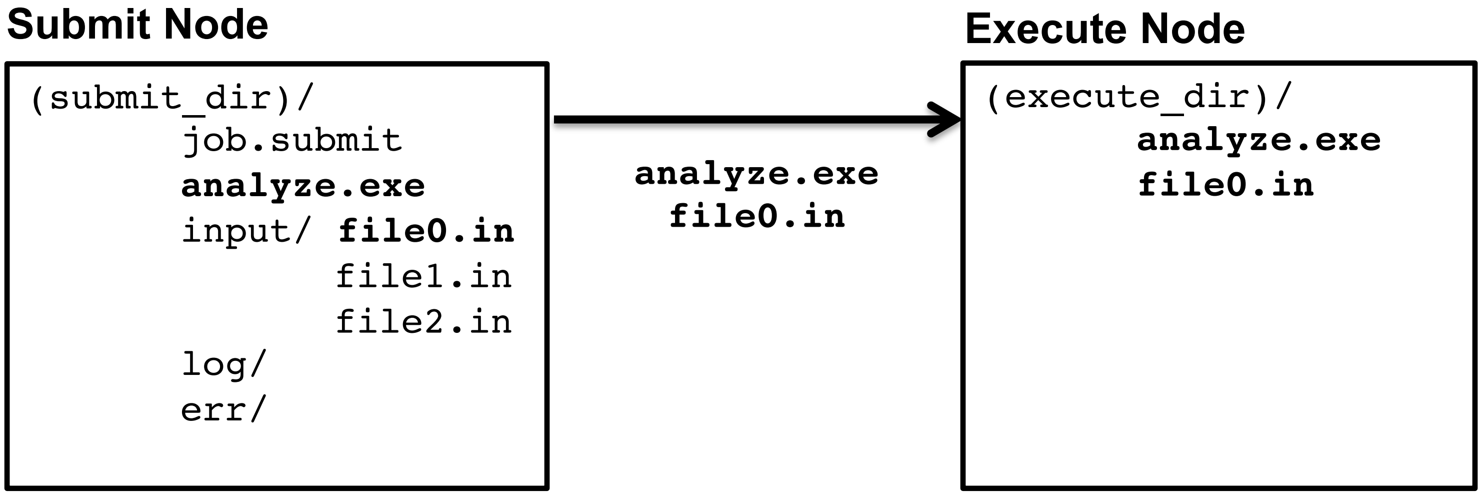
job.submit

```
executable = analyze.exe
arguments = file$(Process).in file$(Process).out
transfer_input_files = input/file$(Process).in

log = log/job$(Process).log
error = err/job$(Process).err

queue 3
```

*directories must be created before jobs are submitted

Open Science Grid

Job Running



Submit Node

```
(submit_dir)/
        job.submit
        analyze.exe
        input/  file0.in
                file1.in
                file2.in
        log/
        err/
```

```
analyze.exe
 file0.in
```

Execute Node

```
(execute_dir)/
        analyze.exe
        file0.in
```

File always get transferred into the ***top level*** of the execute directory, **regardless of how they are organized on the submit server**.

Separating 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

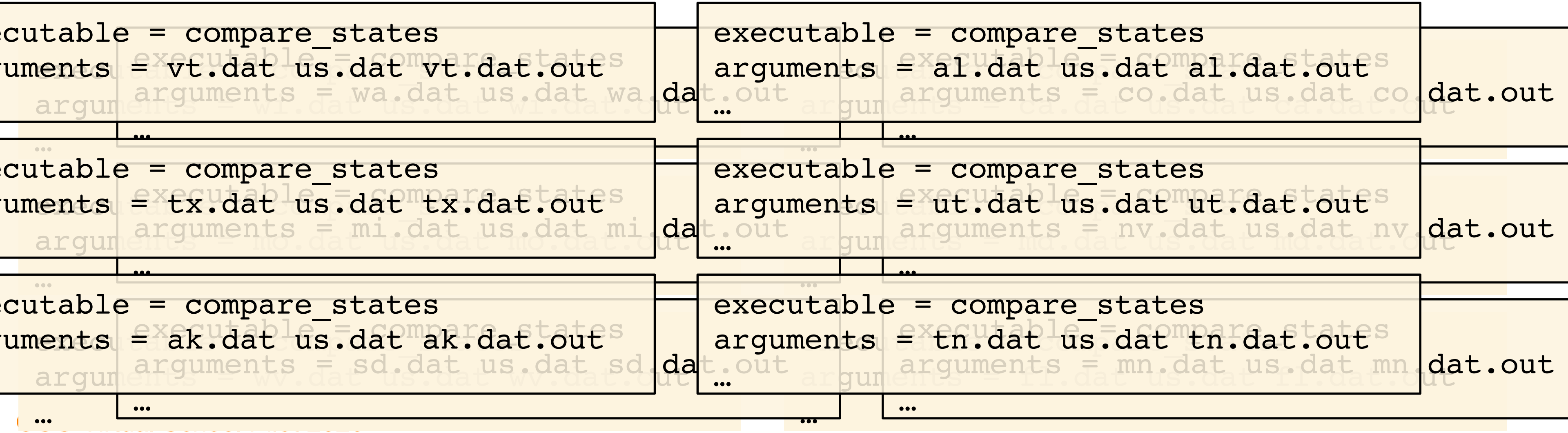
```
executable = analyze.exe
initialdir = job$(Process)
arguments = file.in file.out
transfer_input_files = file.in

log = job.log
error = job.err

queue 3
```

executable must be relative to the submission directory, and *not* in the InitialDir.

*directories must be created before jobs are submitted

What about non-numbered jobs?

- Back to our compare_states example…
- What if we had data for each state? We could do 50 submit files (or 50 "`queue 1`" statements) ...



```
cutable = compare_states
uments = vt.dat us.dat vt.dat.out
```

```
executable = compare_states
arguments = wa.dat us.dat wa.dat.out
…
```

```
arguments = wi.dat us.dat wi.dat.out
…
```

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

```
executable = compare_states
arguments = co.dat us.dat co.dat.out
…
```

```
arguments = ca.dat us.dat ca.dat.out
…
```

```
cutable = compare_states
uments = tx.dat us.dat tx.dat.out
```

```
executable = compare_states
arguments = mi.dat us.dat mi.dat.out
…
```

```
arguments = mo.dat us.dat mo.dat.out
…
```

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

```
executable = compare_states
arguments = nv.dat us.dat nv.dat.out
…
```

```
arguments = md.dat us.dat md.dat.out
…
```

```
cutable = compare_states
uments = ak.dat us.dat ak.dat.out
```

```
executable = compare_states
arguments = sd.dat us.dat sd.dat.out
…
```

```
arguments = wv.dat us.dat wv.dat.out
…
```

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

```
executable = compare_states
arguments = mn.dat us.dat mn.dat.out
…
```

```
arguments = fl.dat us.dat fl.dat.out
…
```

Submitting Multiple Jobs – Queue Statements

multiple submit files (multiple queue statements)	Not Recommended

var **matching** *pattern*	`queue state matching *.dat`
	`queue directory matching job*`

var **in** (i ii iii …)	`queue state in (wi.dat ca.dat co.dat)`

var1,var2 **from** *csv_file*	`queue state from state_list.txt`
	`state_list.txt:`
	`wi.dat`
	`ca.dat`
	`mo.dat`
	`...`

Multiple Job Use Cases – Queue Statements

multiple submit files	**Not recommended.** Though, may be useful for separating job *batches*, conceptually, for yourself.
var **matching** *pattern*	Minimal preparation, can use “files” or “dirs” keywords to narrow possible matches. Requires good naming conventions, less reproducible.
var **in** (i,ii,iii,…)	**All information contained in the submit file**: reproducible. Harder to automate submit file creation.
var1,var2 **from** *csv_file*	**Supports multiple variables**, highly modular (easy to use one submit file for many job batches that have different *var* lists), reproducible. Additional file needed, but can be automated.

TESTING AND TROUBLESHOOTING

What Can Go Wrong?

- Jobs can go wrong “internally”:
  - the executable experiences an error
- Jobs can go wrong *logistically*, from HTCondor’s perspective:
  - a job can’t be matched
  - files not found for transfer
  - job used too much memory
  - badly-formatted executable
  - and more...

Open Science Grid

Reviewing Failed Jobs

- Job log, output and error files can provide valuable troubleshooting details:

Log	Output	Error
• when jobs were submitted, started, held, or stopped • where job ran • resources used • interruption reasons • **exit status**	• stdout (or other output files) • any “print” or “display” information from your program (may contain errors from the executable)	• stderr captures errors from the operating system, or reported by the executable, itself.

Job Holds

- HTCondor will ***hold*** your job if there's logistical issue that YOU (or maybe an admin) need to fix.
  - files not found for transfer, over memory, etc.
- A job that goes on hold is interrupted (all progress is lost), but remains in the queue in the "**H**" state until removed, or (fixed and) released.

Open Science Grid

Diagnosing Holds

- If HTCondor puts a job on hold, it provides a hold reason, which can be viewed in the log file, with **`condor_q -hold <Job.ID>`**, or with:

`condor_q -hold -af HoldReason`

```
$ condor_q -hold -af HoldReason
Error from slot1_1@wid-003.chtc.wisc.edu: Job has gone over
  memory limit of 2048 megabytes.
Error from slot1_20@e098.chtc.wisc.edu: SHADOW at
  128.104.101.92 failed to send file(s) to <128.104.101.98:35110>: error
  reading from /home/alice/script.py: (errno 2) No such file or directory;
  STARTER failed to receive file(s) from <128.104.101.92:9618>
Error from slot1_11@e138.chtc.wisc.edu: STARTER
  at 128.104.101.138 failed to send file(s) to <128.104.101.92:9618>;
SHADOW at
  128.104.101.92 failed to write to file /home/alice/Test_18925319_16.err:
  (errno 122) Disk quota exceeded
```

Common Hold Reasons

- Job has used **more memory or disk** than requested.
- **Incorrect path to files** that need to be transferred
- **Badly formatted executables**
  (e.g. Windows line endings on Linux)
- Submit directory is **over quota**.
- **Job has run for too long**.
  (72-hour default in CHTC Pool)
- The **admin has put your job on hold**.

Holding and Removing Jobs

- If you know your job has a problem and it hasn't yet completed, you can fix it!
- **If the problem requires resubmission:**
  - Remove it from the queue:

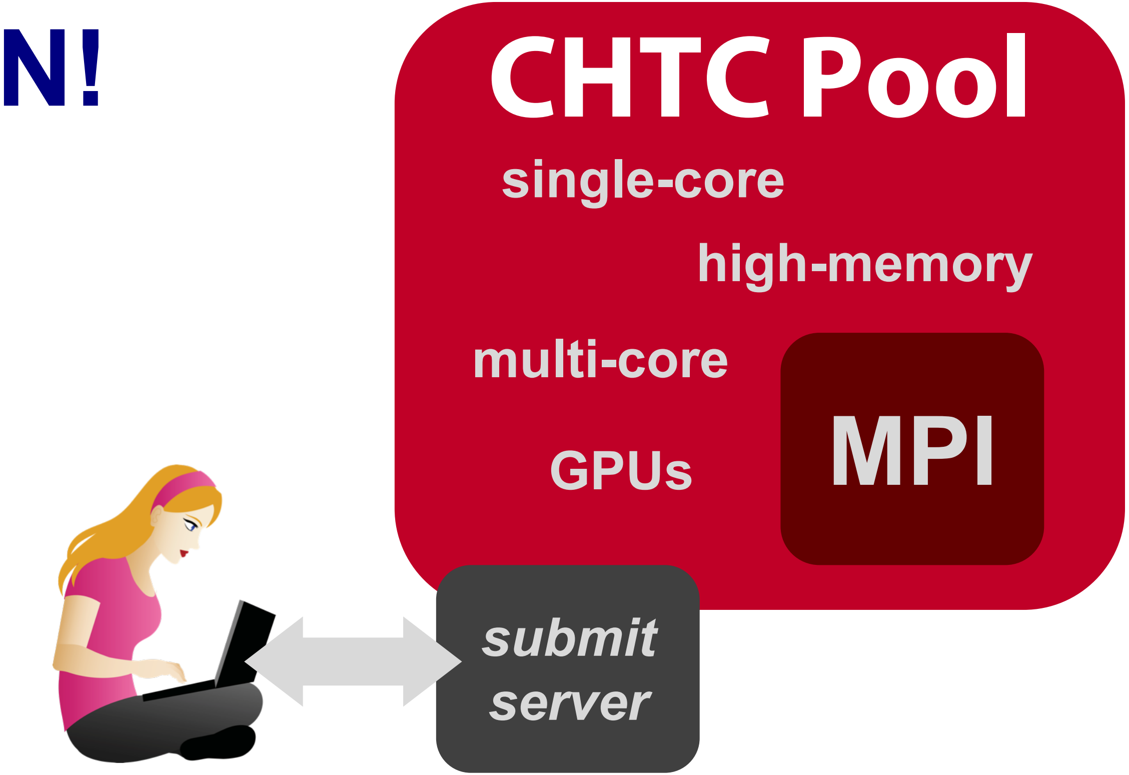
```
condor_rm [U/C/J]
```

- **If problem is within the executable or input file(s):**
  - Hold the job, fix it, and release it:

```
condor_hold [U/C/J]
condor_release [U/C/J]
```

HTCondor Manual: condor_hold
HTCondor Manual: condor_rm

YOUR TURN!



CHTC Pool

single-core

high-memory

multi-core

GPUs

MPI

submit server

Thoughts on Exercises

- Copy-and-paste is quick, but you ***WILL*** learn more by typing out commands and submit file contents
- **Ask Questions during Work Time! (Slack)**
- **Exercises in THIS unit** are important to finish before moving on! (You can save “bonus” exercises for later.)

- **(See 1.6 if you need to remove jobs!)**

Reviewing Jobs

- To review a large group of jobs at once, 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

Using Multiple Variables

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

`job.submit`

```
executable = compare_states
arguments = -y $(year) -i $(infile)

transfer_input_files = $(infile)

queue infile,year from job_list.txt
```

`job_list.txt`

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

Shared Files

- HTCondor can transfer an entire directory or all the contents of a directory
  - transfer whole directory

```
transfer_input_files = shared
```

  - transfer contents only

```
transfer_input_files = shared/
```

- Useful for jobs with many shared files; transfer a directory of files instead of listing files individually

```
(submit_dir)/

job.submit
shared/
    reference.db
    parse.py
    analyze.py
    cleanup.py
    links.config
```