Open Science Grid

Introduction to High Throughput Computing and HTCondor

Monday AM, Lecture 1

Lauren Michael

Overview – 1.1

- What is *high throughput computing (HTC)* ?
- How does the HTCondor job scheduler work?
- How do you run jobs on an HTCondor compute system?

Open Science Grid

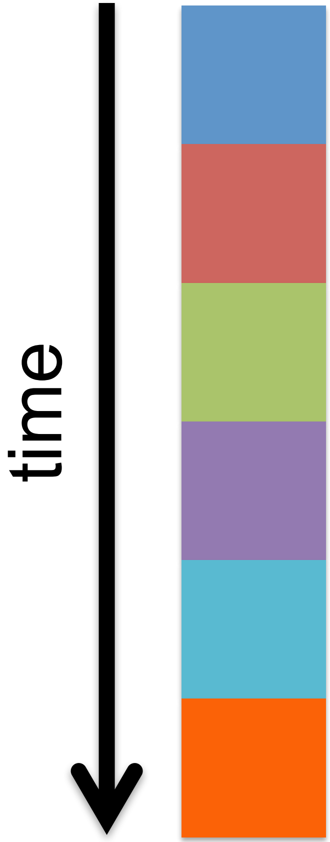
Keys to Success

- Work hard
- Ask questions!
  - …during lectures
  - ...during exercises
  - ...during breaks
  - ...during meals
- If we do not know an answer, we will try to find the person who does.

Serial Computing

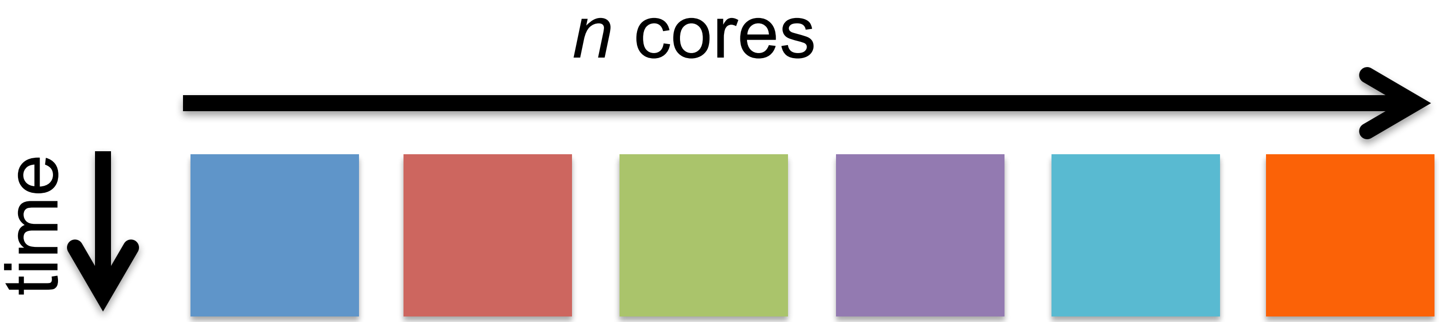
What many programs look like:

- Serial execution, running on one processor (CPU core) at a time
- Overall compute time grows significantly as individual tasks get more complicated (long) or if the number of tasks increases
- ***How can you speed things up?***

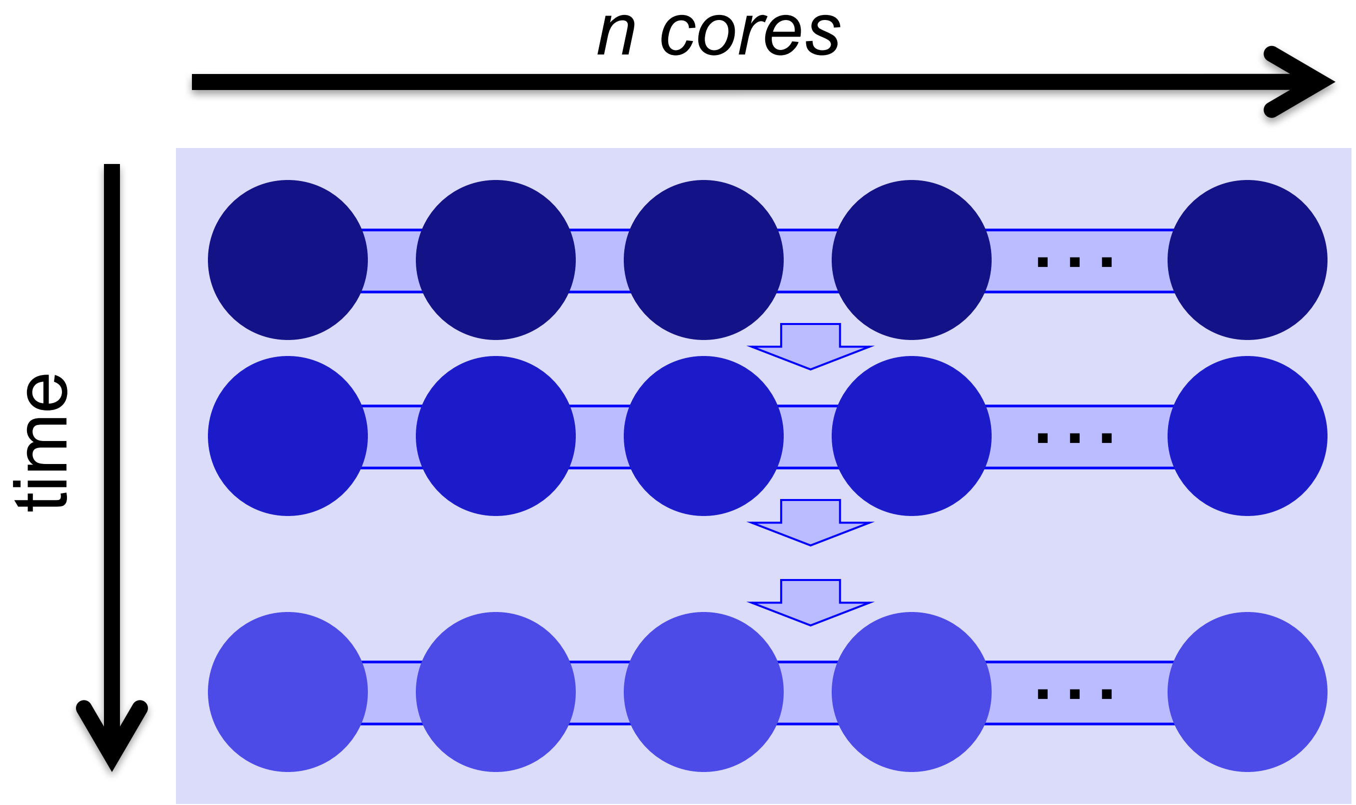


High Throughput Computing (HTC)

- Parallelize!
- Independent tasks run on different cores

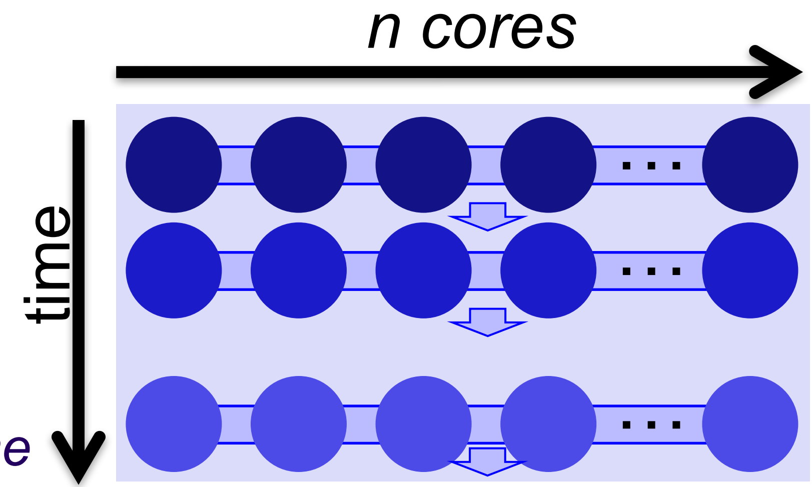


High Performance Computing (HPC)



High Performance Computing (HPC)

- Benefits greatly from:
  - CPU speed + homogeneity
  - Shared filesystems
  - Fast, expensive networking (e.g. Infiniband) and servers co-located
- Scheduling: **Must wait until all processors are available**, *at the same time* and *for the full duration*
- Requires special programming (MP/MPI)
- ***What happens if one core or server fails or runs slower than the others?***



High Throughput Computing (HTC)

- Scheduling: only need **1 CPU core for each** (shorter wait)
- Easier recovery from failure
- No special programming required
- Number of concurrently running jobs is *more* important
- CPU speed and homogeneity are *less* important

HPC vs HTC: An Analogy

HPC vs HTC: An Analogy

High *Throughput* vs High *Performance*

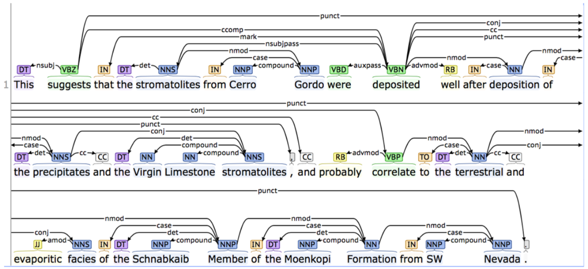
HTC

- Focus: Large workflows of ***numerous***, ***relatively small***, and ***independent*** compute tasks
- More important: maximized number of running tasks
- Less important: CPU speed, homogeneity

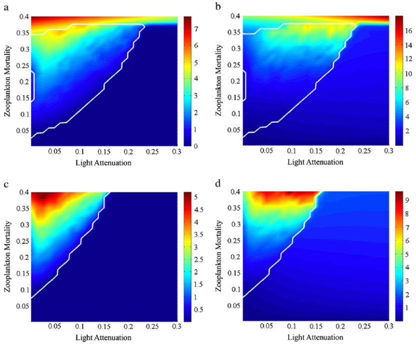
HPC

- Focus: Large workflows of ***highly-interdependent*** sub-tasks
- More important: persistent access to the *fastest* cores, CPU homogeneity, special coding, shared filesystems, fast networks

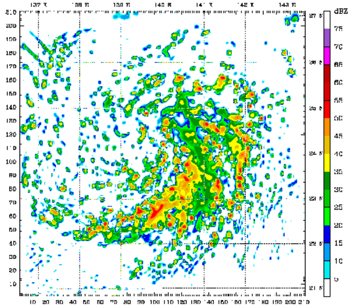
HTC Examples



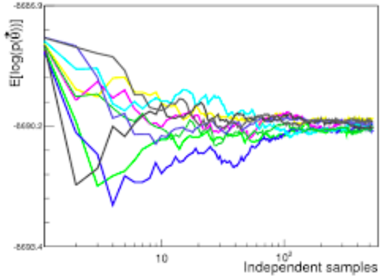
text analysis (most genomics …)



parameter sweeps



multi-start simulations



statistical model optimization (MCMC, numerical methods, etc.)

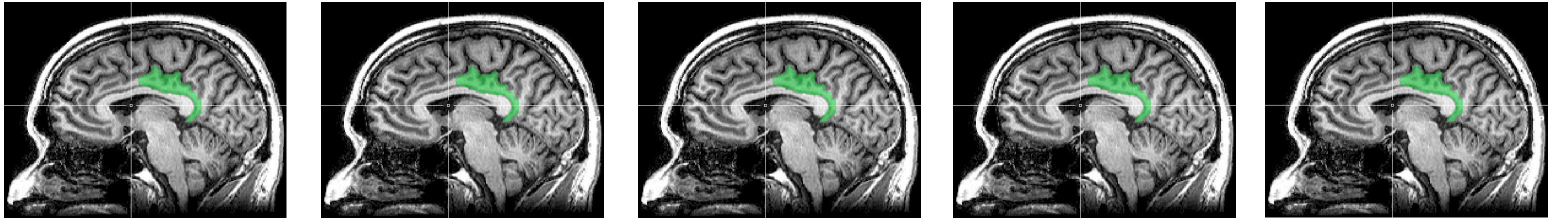
multi-image and mulit-sample analysis

Is your research HTC-able?

- ***Can it be broken into relatively numerous, independent pieces?***

- *Think about your research! Can you think of a good high throughput candidate task? Talk to your neighbor!*

Example Challenge



You need to process 48 brain images for each of 168 patients. **Each image takes ~1 hour of compute time.**

168 patients x 48 images = ~8000 tasks = ~8000 hrs

Conference is next week.

Distributed Computing

- Use many computers, each running one instance of our program
- Example:
  - **1 laptop (1 core) => 4,000 hours = ~½ year**
  - 1 server (~20 cores) => 500 hours = ~3 weeks
  - 1 large job (400 cores) => 20 hours = ~1 day
  - **A whole cluster (8,000 cores) = ~8 hours**

Break Up to Scale Up

- Computing tasks that are ***easy to break up*** are ***easy to scale up***.

- To truly grow your computing capabilities, you also need a system appropriate for your computing task!

What computing resources are available?

- A single computer?
- A local cluster?
  - Consider: What *kind* of cluster is it? Typical clusters tuned for HPC (large MPI) jobs typically may not be best for HTC workflows! Do you need even more than that?
- Open Science Grid (OSG)
- Other
  - European Grid Infrastructure
  - Other national and regional grids
  - Commercial cloud systems (e.g. HTCondor on Amazon)

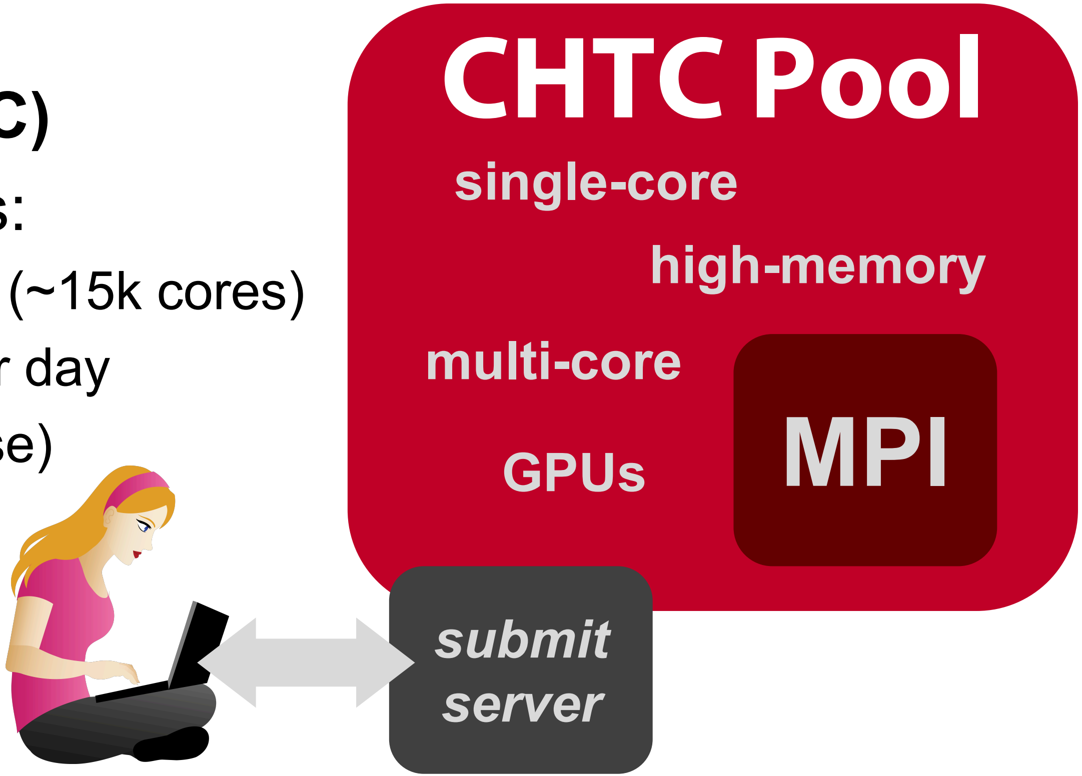
Example Local Cluster

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

  ~130 million hrs/year (~15k cores)

  ~10,000 per user, per day

  (~400 cores in use)



Open Science Grid

- **HTC for Everyone**
  - ~100 contributors
  - **Past year:**
    - >420 million jobs
    - >1.5 billion CPU hours
    - >200 petabytes transferred

- Can submit jobs locally, they backfill across the country - interrupted at any time (but not too frequent)
- http://www.opensciencegrid.org/

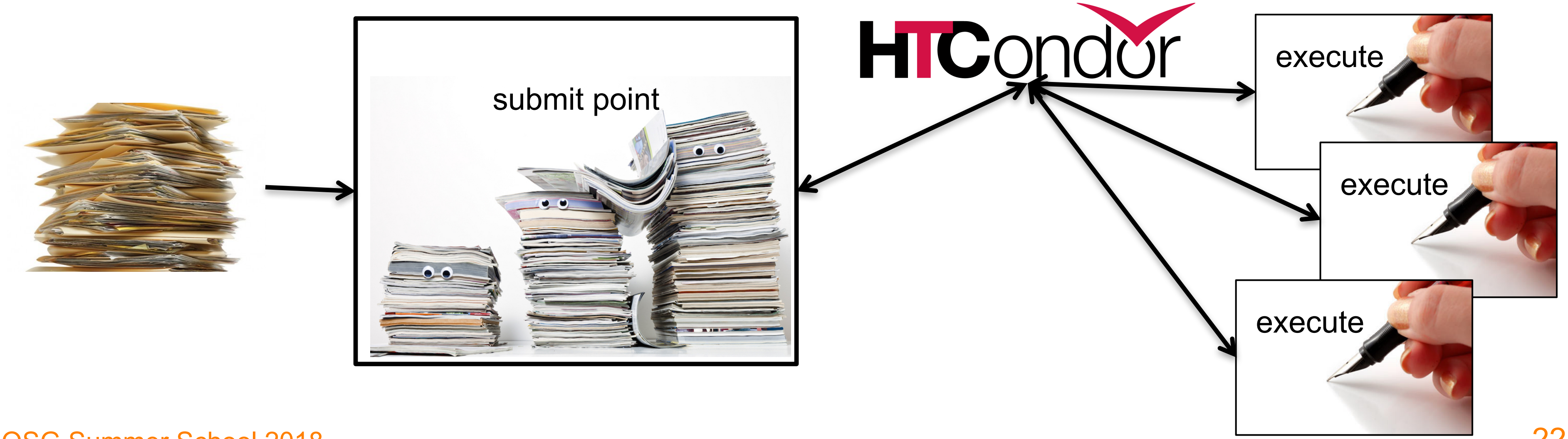
HTCONDOR

HTCondor History and Status

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

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

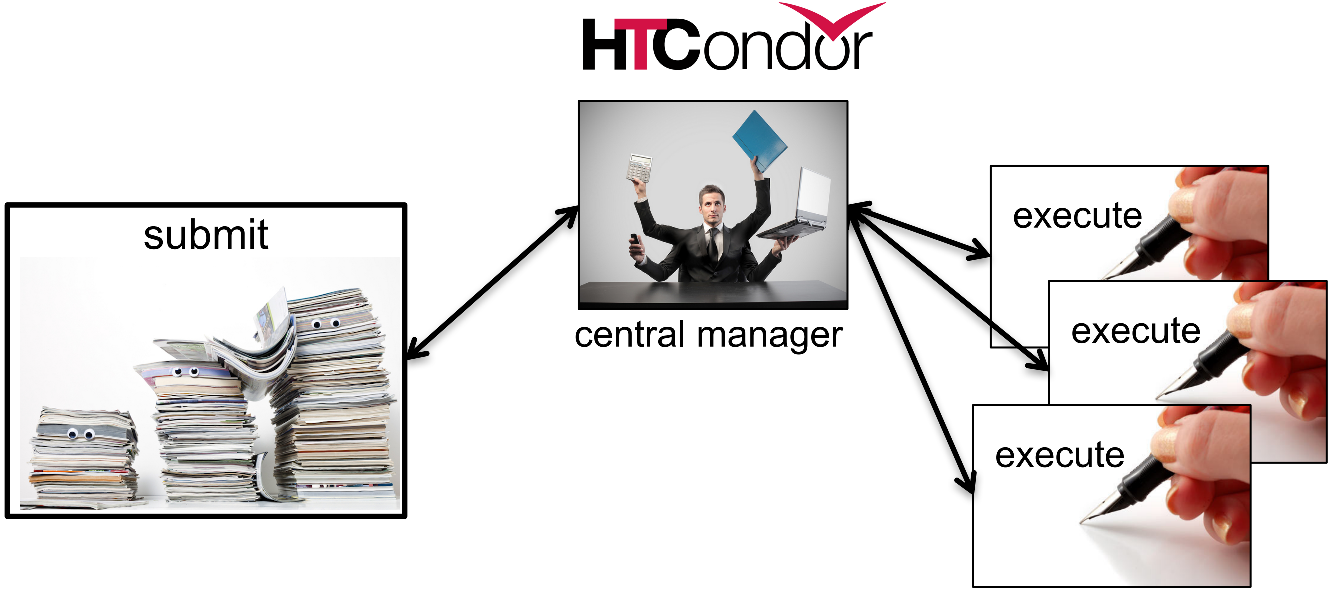
Terminology: *Machine, Slot*

- ***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 may have 4-40 slots

- HTCondor can break up and create new slots, dynamically, as resources become available from completed jobs

Job Matching

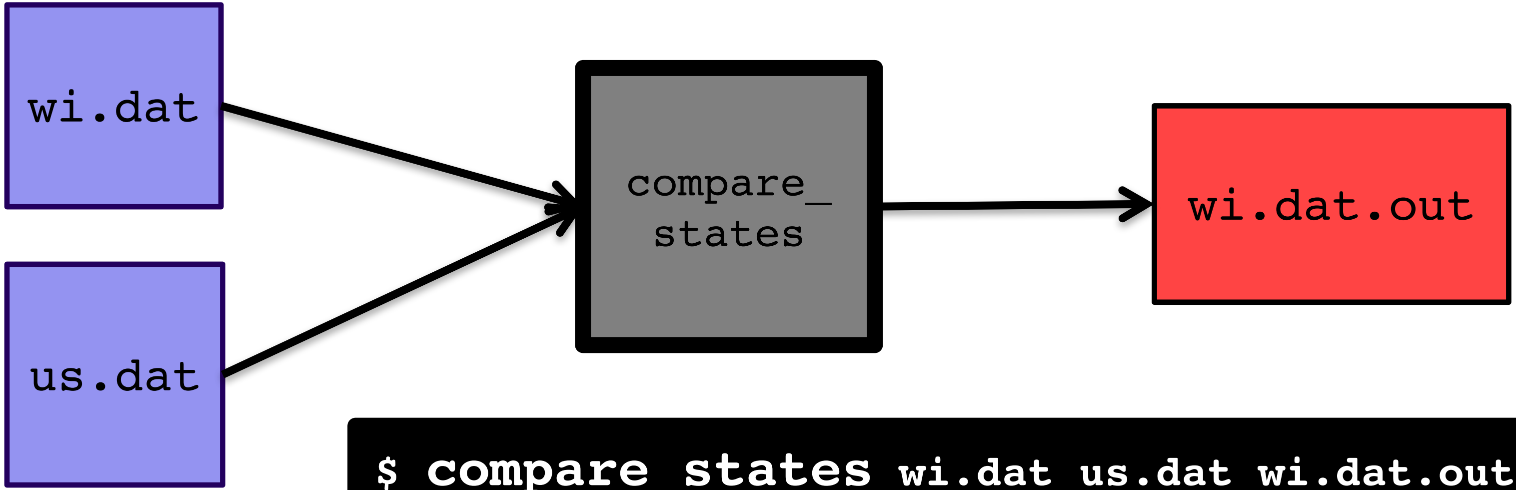
- On a regular basis, the central manager reviews ***Job*** and ***Machine*** attributes and matches jobs to ***Slots***.



BASIC JOB SUBMISSION

Job Example

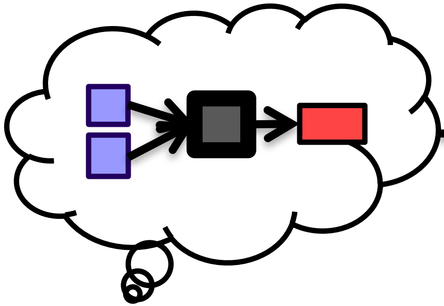
- program called “compare_states” (executable), which compares two data files (input) and produces a single output file.




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

Job Translation

- ***Submit file***: communicates everything about your job(s) to HTCondor



```
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
```

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

compare_states

- 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

wi.dat

us.dat

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: 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 <u>your jobs only</u> and <u>batches</u> jobs that were submitted together:

```
$ 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`**

- 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: 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

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

```
$ 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)/
```

Open Science Grid

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+00:01:08 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 : <128.104.101.92:9618?...
 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
```

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

←

Execute Node

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

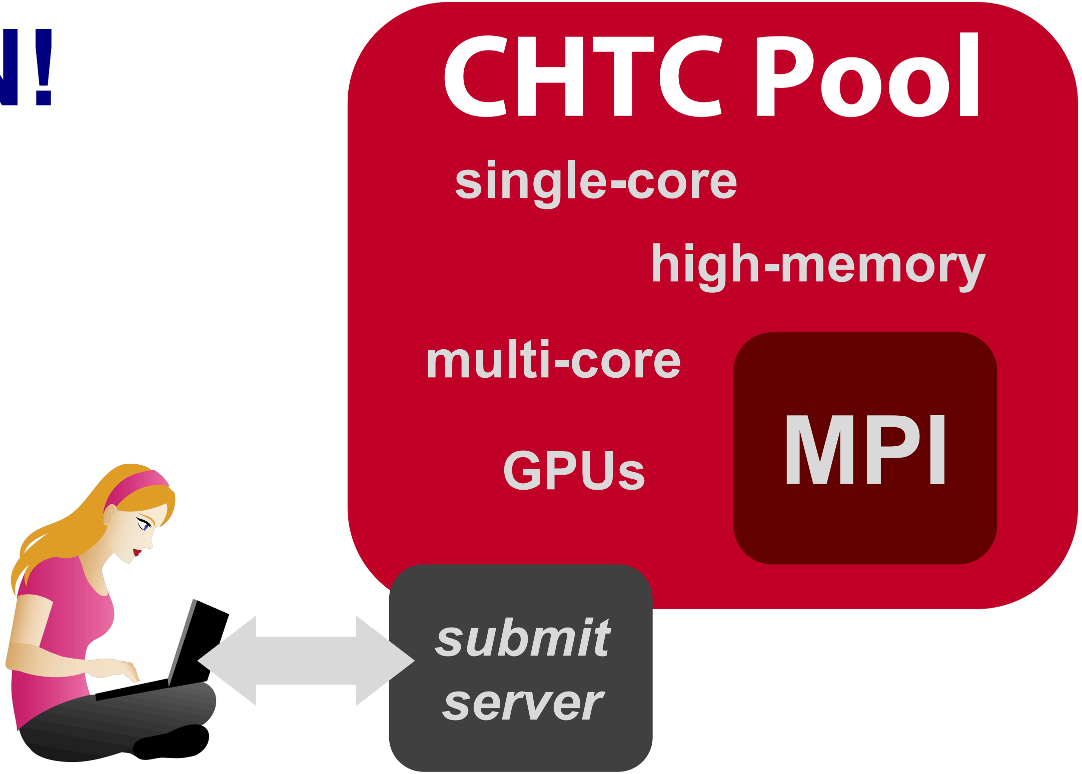
Job Completes (cont.)

```
$ 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
```

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 (first) submit file contents
- **Exercises 1.1-1.3** are most important to finish THIS time **(see 1.6 if you need to remove jobs)!**
- If you do not finish, that's OK – You can make up work later or during evenings, if you like. (There are even "bonus" challenges, if you finish early.)

Exercises!

- Ask questions!
- Lots of instructors around
- Coming next:
  - Now – 10:30 Hands-on Exercises
  - 10:45 – 11:00 Break
  - 11:00 – 11:30 Submitting Many Jobs
  - 11:30 – 12:15 Hands-on Exercises