Open Science Grid

# Introduction to High Throughput Computing and HTCondor

Monday AM, Lecture 1

Ian Ross

Center for High Throughput Computing

University of Wisconsin-Madison

# 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.

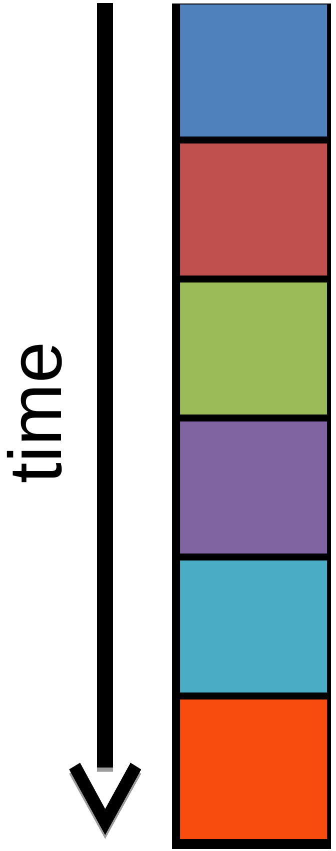
# Goals for this Session

- Understand the basics of High Throughput Computing (HTC)
- Understand a few things about HTCondor, which is one kind of HTC system
- Use basic HTCondor commands
- Run a job locally

# Serial Computing

## What most programs look like:

- Serial execution, running on one processor at a time
- Overall compute time grows significantly as individual tasks get more complicated and number of tasks in a workflow increase
- *How can you speed things up?*

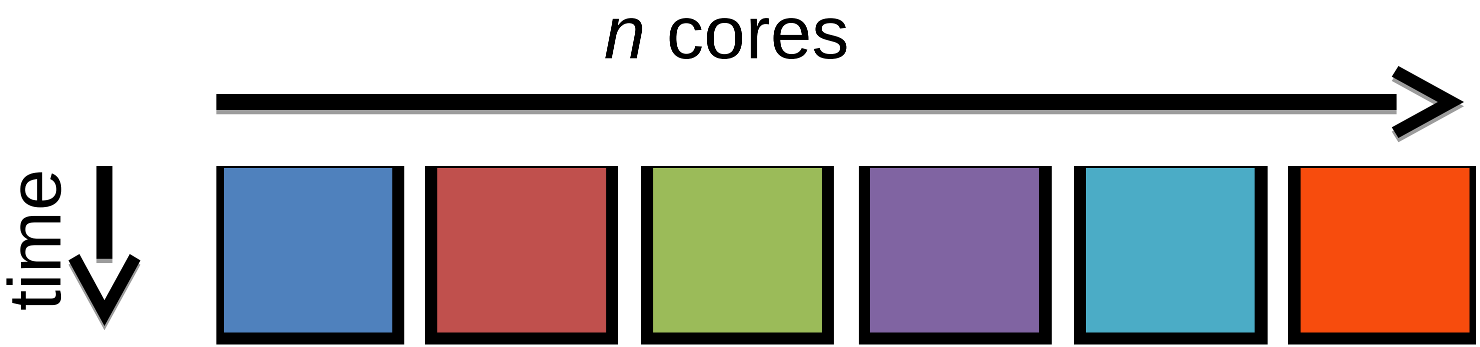


# Parallelized Computing

- Parallelize!
- Use more processors to break up the work!

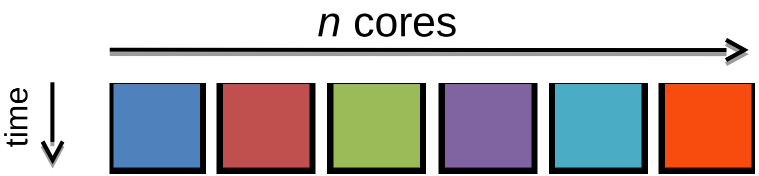
# High Throughput Computing (HTC)

- Independent tasks
  - “Embarrassingly” parallel



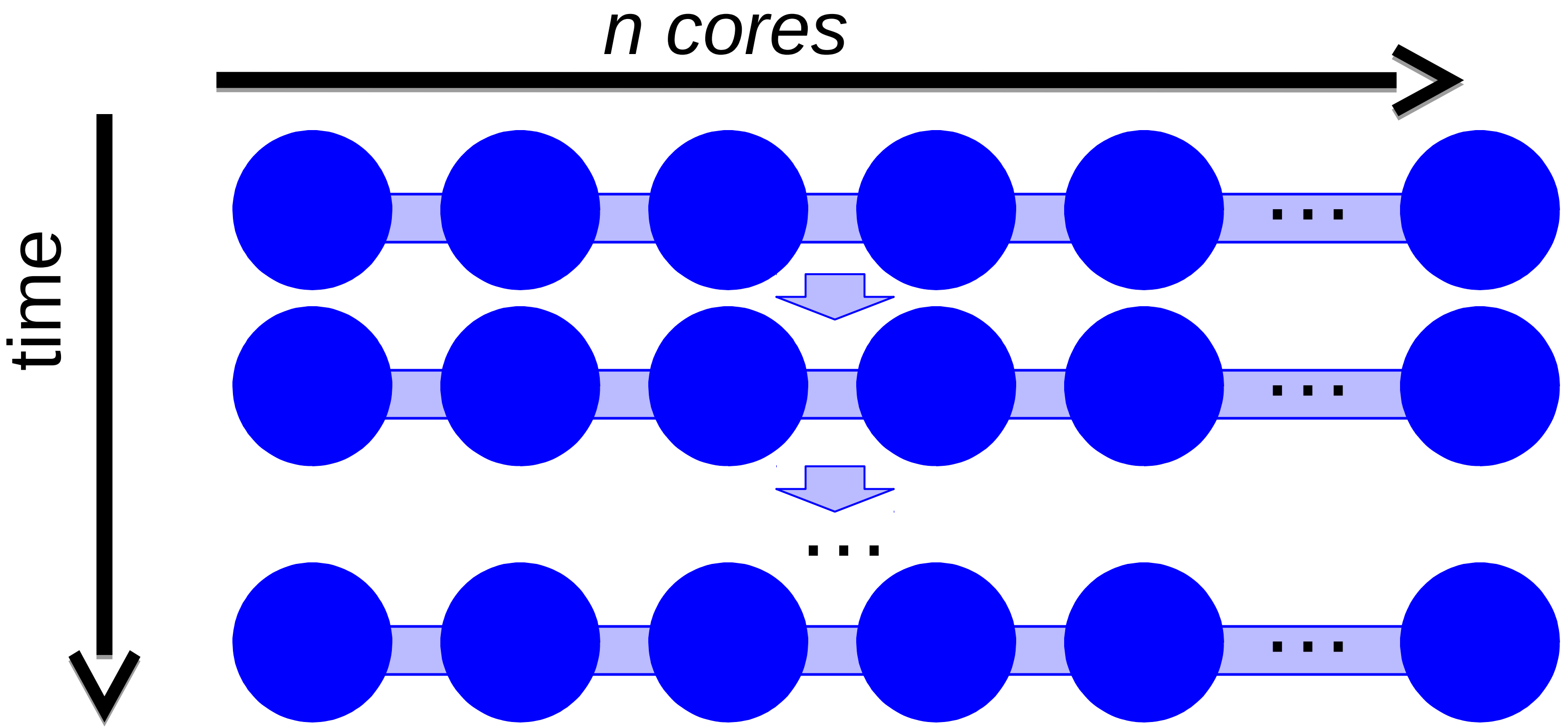
# High Throughput Computing (HTC)

- Scheduling: Only need to find 1 CPU at a time (shorter wait)
- Easy recovery from failure
- No special programming required
- Number of concurrently running jobs is *more* important
- CPU speed and homogeneity are *less* important



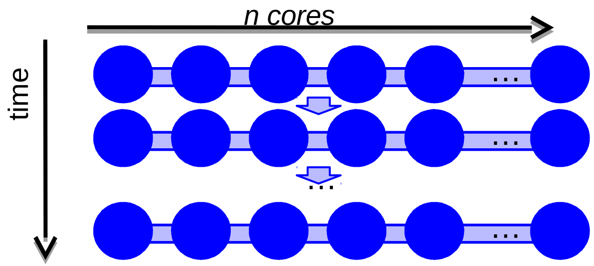
# High Performance Computing (HPC)

- Interdependent sub-tasks



# High Performance Computing (HPC)

- Benefits greatly from:
  - Shared filesystems
  - Fast networking (e.g. Infiniband)
  - CPU speed + homegeneity
- Scheduling: Must wait for (and reserve) multiple processors for full duration
- Often requires special code
- Focus on biggest, fastest systems (supercomputers)



# HPC vs HTC: An Analogy

# HPC vs HTC: An Analogy


# High *Throughput* vs High *Performance*

## HTC

- Focus: Workflows with many *small*, *largely independent* compute tasks
- CPU speed and homogeneity are less important

## HPC

- Focus: Workflows with *large, highly coupled* tasks
- Supercomputers, highly specialized code, benefit from shared filesystems and fast networks

# Research Computing

- When considering a strategy for your research computing, there are two key considerations:
  - Is your problem HTC-able?
  - Which available resources are best?

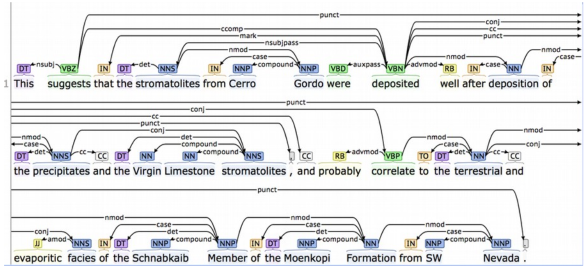
Open Science Grid

# Is your research HTC-able?

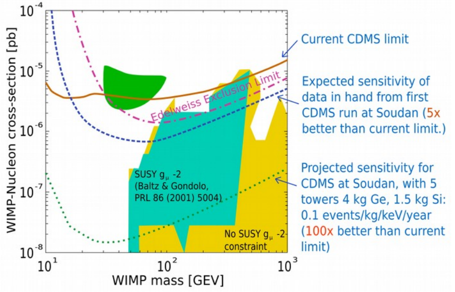
- Can it be broken into fairly small, independent pieces?
  - Easy to ask, harder to answer!

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

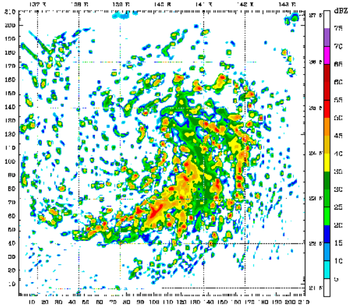
# HTC Examples



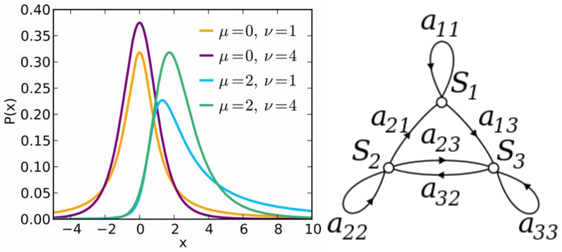

text analysis




parameter sweeps



multi-start simulations




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

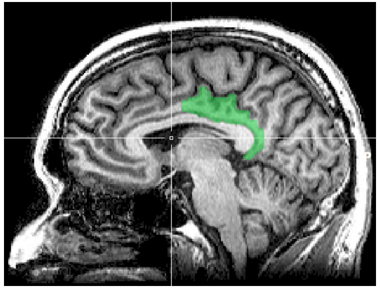


image or data analysis

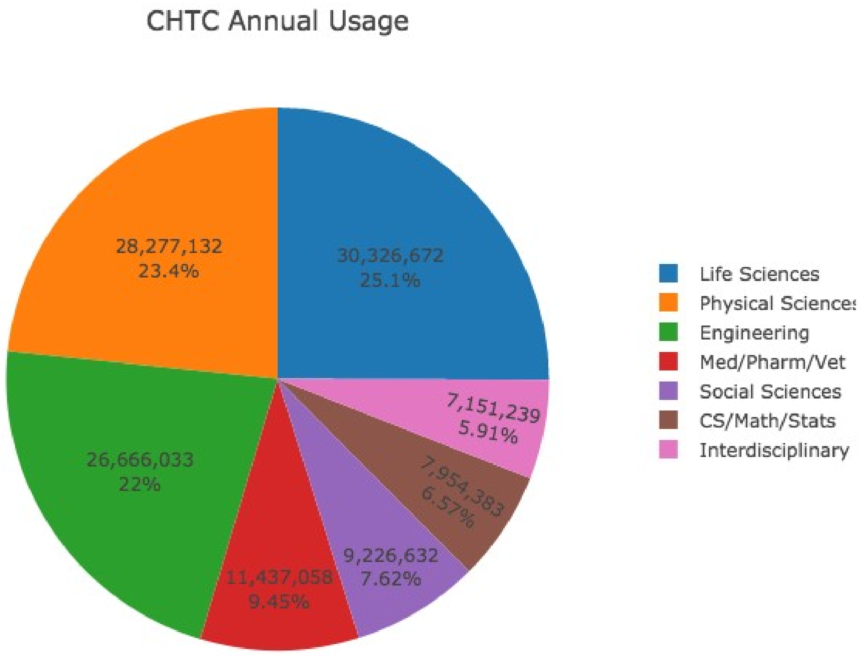
Open Science Grid

# What computing resources are available to you?

- A single computer?
- A local cluster?
  - Consider: What *kind* of cluster is it? Clusters tuned for HPC jobs typically aren’t appropriate for HTC workflows!
- Open Science Grid (OSG)
- Other
  - European Grid Infrastructure
  - Other national and regional grids
  - Commercial cloud systems used to augment grids

# Example Local Cluster

- Wisconsin's Center for High Throughput Computing (CHTC)
- Local pool recent CPU hours:
  - ~360,000/day
  - ~10 million/month
  - ~120 million/year



# Open Science Grid

- HTC Scaled Way up
  - Over 120 sites
  - Past year:
    - ~180 million jobs
    - ~1.2 billion CPU hours
    - ~243 petabytes transferred


- Can submit jobs locally, move to OSG
- http://www.opensciencegrid.org/

- Computing tasks that are easy to *break up* are easy to *scale up*.
- To truly grow your computing capabilities, you need a system appropriate for your computing task!

# Example Challenge

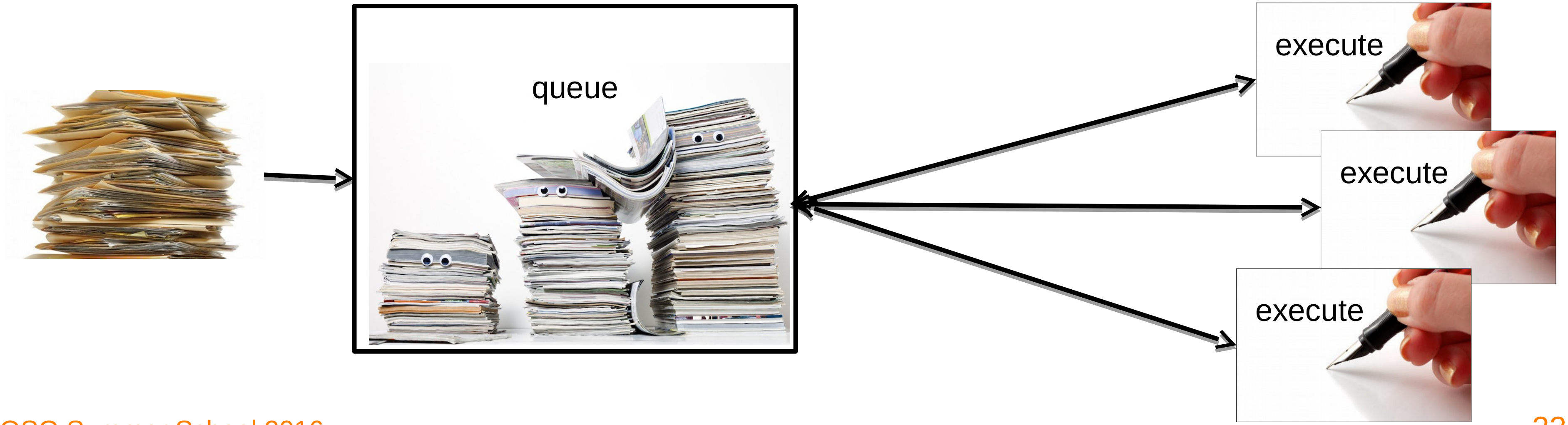
- You have a program that optimizes traffic signals in intersections.
  – Your input is the number of cars passing through a stoplight in an hour.
- You have 100 days of data, from 4 different stoplights
- Each run of your program takes about 1 hour
- 4 x 24 x 100 x 1 hour = 9600 hours ~= 1.1 years running nonstop!
- Conference is next week

# Distributed Computing

- Use many computers, each running one instance of our program
- Example:
  - 2 computers => 4,800 hours ~= ½ year
  - 8 computers => 1,200 hours ~= 2 months
  - 100 computers => 96 hours ~= 4 days
  - 9,600 computers => 1 hour! (but…)
- In HTC, these machines are no faster than your laptop!

# How It Works

- Submit tasks to a queue (on a submit point)
- Tasks are scheduled to run on computers (execute points)



# Distributed Computing Systems

- HTCondor (much, much more to come!)
- Other systems to manage a local cluster:
  - PBS/Torque
  - LSF
  - Sun Grid Engine/Oracle Grid Engine
  - SLURM

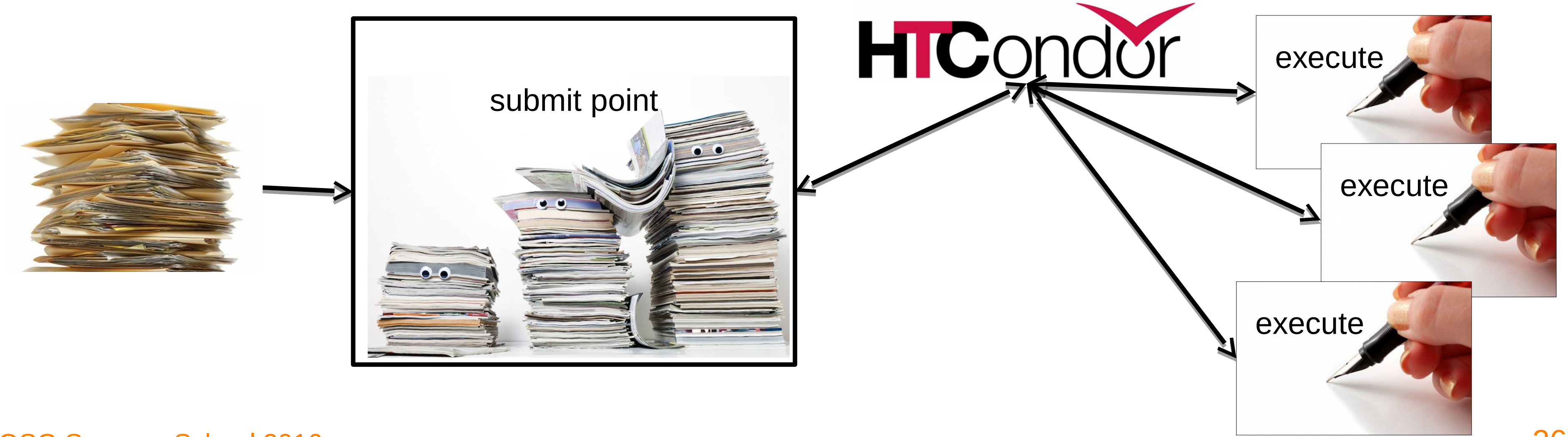
# HTCONDOR

# HTCondor History and Status

- History
  - Started in 1988 as a “cycle scavenger”
  - Protected interests of users and machine owners
- Today
  - Expanded to become CHTC team: 20+ full-time staff
  - Current production release: HTCondor 8.4.8
  - HTCondor software: ~700,000 lines of C/C++ code
- Miron Livny
  - Professor, UW-Madison CompSci
  - Director, CHTC
  - Dir. Of Core Comp. Tech., WID/MIR
  - Tech. Director & PI, OSG

# HTCondor -- How It Works

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



# Roles in an HTCondor System

- Users
  - Define jobs, their requirements, and preferences
  - Submit and cancel jobs
  - Check on the state of a job
- Administrators
  - Configure and control the HTCondor system
  - Implement policies
  - Check on the state of the machines
- HTCondor Software
  - Match jobs to machines (enforcing all policies)
  - Track and manage machines
  - Track and run jobs

# Terminology: *job*

- *Job*: A computer program or one run of it
- Not interactive, no graphic interface (e.g. not Word or email)
  – How would you interact with 1,000 programs running at once?
- Three main pieces
  1. Input: Command-line arguments and/or files
  2. Executable: the program to run
  3. Output: standard output & error and/or files
- Scheduling
  – User decides when to *submit* job to be run
  – System decides when to run job, based on policy

# Terminology: *Machine, Slot*

- *Machine*
  - A machine is a physical computer (typically)
  - May have multiple *processors* (computer chips)
  - One processor may have multiple *cores* (CPUs)
- HTCondor: *Slot*
  - One assignable unit of a machine (i.e. 1 job per slot)
  - Most often, corresponds to one core
  - Thus, typical machines today have 4-40 slots
- Advanced HTCondor features: Can get 1 slot with many cores on one machine for multicore jobs

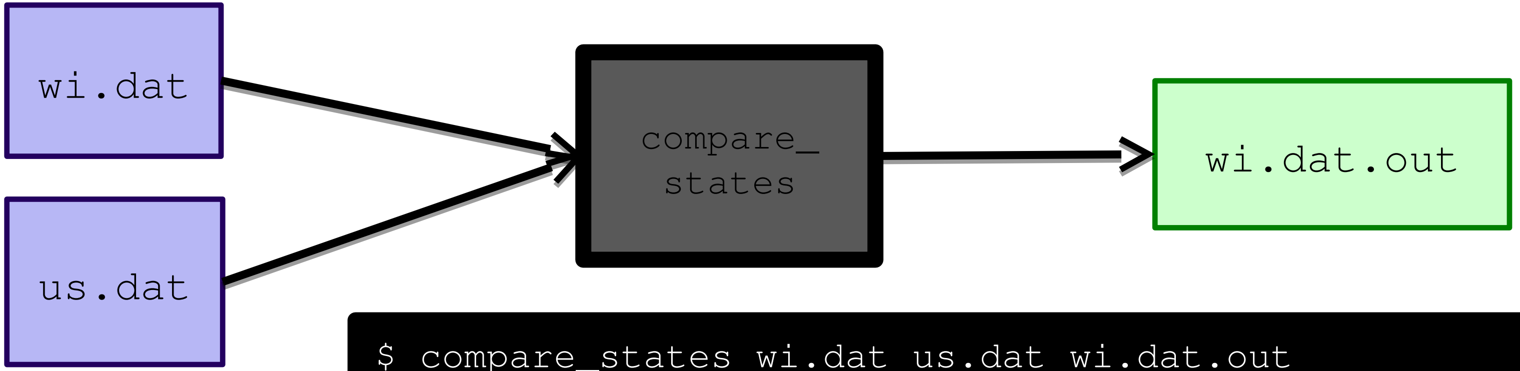
# Terminology: Matchmaking

- Two-way process of finding a slot for a job
- Jobs have requirements and preferences
  - E.g.: I need Red Hat Linux 6 and 100 GB of disk space, and prefer to get as much memory as possible
- Machines have requirements and preferences
  - E.g.: I run jobs only from users in the Comp. Sci. dept., and prefer to run ones that ask for a lot of memory
- *Important jobs may replace less important ones*
  - Thus: Not as simple as waiting in a line!

# BASIC JOB SUBMISSION

# Job Example

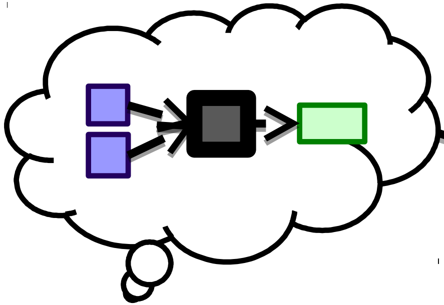
- For our example, we will be using an imaginary program called “compare_states”, which compares two data files and produces a single output file.




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

# Job Translation

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



```
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

# Basic Submit File

```
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
```

# Basic Submit File

```
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
```

- Comma separated list of input files to transfer to the machine

# Basic Submit File

```
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`

# Basic Submit File

```
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
  - *Explored in exercises!*
- `output/error`: Captures stdout and stderr from your program

# Basic Submit File

```
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.
  - *More on this later!*
- `queue`: keyword indicating "create a job"

# BASIC HTCONDOR COMMANDS

# Submit a Job

```
condor_submit submit-file
```

- Submits job to local submit machine
- Use `condor_q` to track

```
Submitting job(s).
1 job(s) submitted to cluster NNN.
```

- Each condor_submit creates one Cluster
- A *job ID* is written as **cluster**.**process** (e.g. **8**.**0**)
- We will see how to make multiple processes later

# Viewing Jobs

## condor_q

- With no arguments, lists all of your* jobs waiting or running here
- For more info: exercises, **-h**, manual, next lecture

```
-- Schedd: learn.chtc.wisc.edu : <...> : ...
 ID      OWNER               SUBMITTED     RUN_TIME ST PRI SIZE CMD
   8.0   cat              11/12 09:30   0+00:00:00 I  0   0.0  compare_states

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

* As of version 8.5. Earlier versions show all jobs submitted from that machine.

# Remove a job

```
condor_rm cluster [...]
condor_rm cluster.process [...]
condor_rm userid
```

- Removes one or more jobs from the queue
- Identify jobs by username, cluster ID, or single job ID
- Only you (or an admin) can remove your jobs

```
Cluster NNN has been marked for removal.
```

# Viewing Slots

```
condor_status
```

- With no arguments, lists *all* slots currently in pool
- Summary info is printed at the end of the list
- For more info: exercises, **-h**, manual, next lecture

```
slot6@opt-a001.cht  LINUX      X86_64 Claimed   Busy     1.000  1024  0+19:09:32
slot7@opt-a001.cht  LINUX      X86_64 Claimed   Busy     1.000  1024  0+19:09:31
slot8@opt-a001.cht  LINUX      X86_64 Unclaimed Idle     1.000  1024  0+17:37:54
slot9@opt-a001.cht  LINUX      X86_64 Claimed   Busy     1.000  1024  0+19:09:32
slot10@opt-a002.ch  LINUX      X86_64 Unclaimed Idle     0.000  1024  0+17:55:15
slot11@opt-a002.ch  LINUX      X86_64 Unclaimed Idle     0.000  1024  0+17:55:16
                     Total Owner Claimed Unclaimed Matched Preempting Backfill

        NTEL/WINNT51     2     0       0         2       0         0        0
       INTEL/WINNT61     52     2       0        50       0          0        0
        X86_64/LINUX     2086  544    1258        284     0         0        0

               Total     2140    546     1258        336
```

# YOUR TURN!

Open Science Grid

# Thoughts on Exercises

- Copy-and-paste is quick, but you may learn more by typing out commands yourself
- Experiment!
  - Try your own variations on the exercises
  - If you have time, try to apply your own work
- If you do not finish, that’s OK – You can make up work later or during evenings, if you like
- If you finish early, try any extra challenges or optional sections, or move ahead to the next section if you are brave

# Sometime today

- Sometime today, sign up for OSG Connect
  – https://twiki.opensciencegrid.org/bin/view/Education/UserSchool16Connect
- It **is not** required today
- It **will be** required tomorrow afternoon
- It is best to start the process early

Open Science Grid

# Exercises!

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

# BACKUP SLIDES

# 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
```