OSG

Intro to HTC and HTCondor

Monday, July 25

Lauren Michael

This work was supported by NSF grants MPS-1148698, OAC-1836650, and OAC-2030508

Intro to HTC and OSG

Overview

- What is *high throughput computing (HTC)* ?
- What is the OSG?
- How do you get the most out of the above?

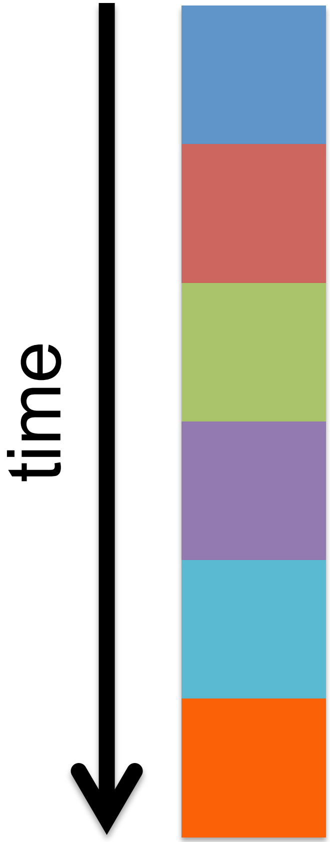
HTC: An Analogy

HTC: An Analogy

Serial Computing

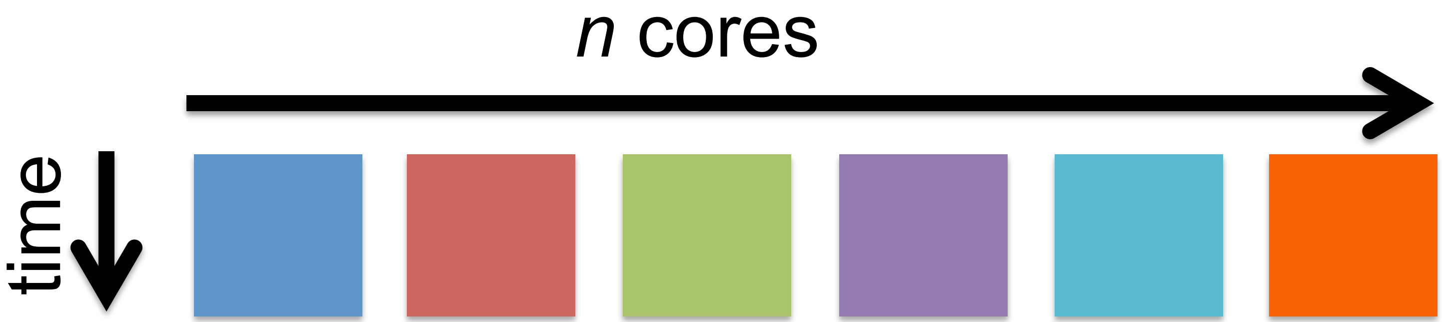
What many programs look like:

- *Serial execution*, running one task 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?***



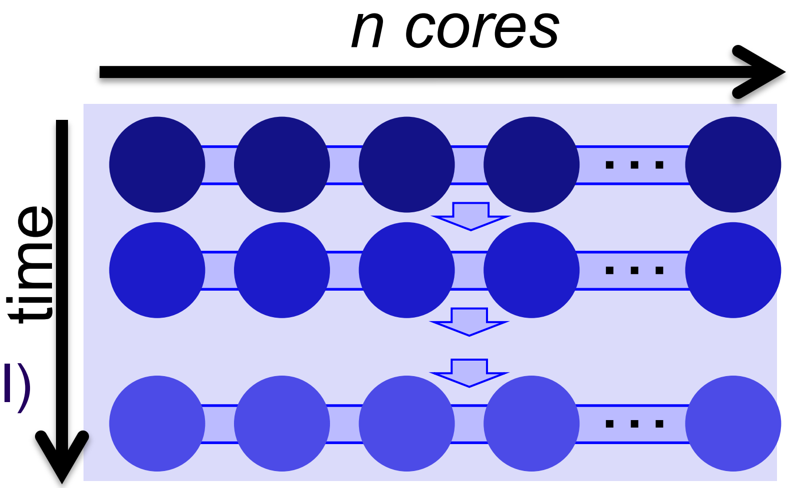
High Throughput Computing (HTC)

- Parallelize!
- Independent tasks run on different cores



High Performance Computing (HPC)

- Benefits greatly from:
  - CPU speed + homogeneity
  - shared filesystems
  - fast, expensive networking (e.g. Infiniband) and co-located servers
- Requires special programming (MP/MPI)
- Scheduling: **Must wait until all processors are available**, *at the same time* and *for the full duration*
- ***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

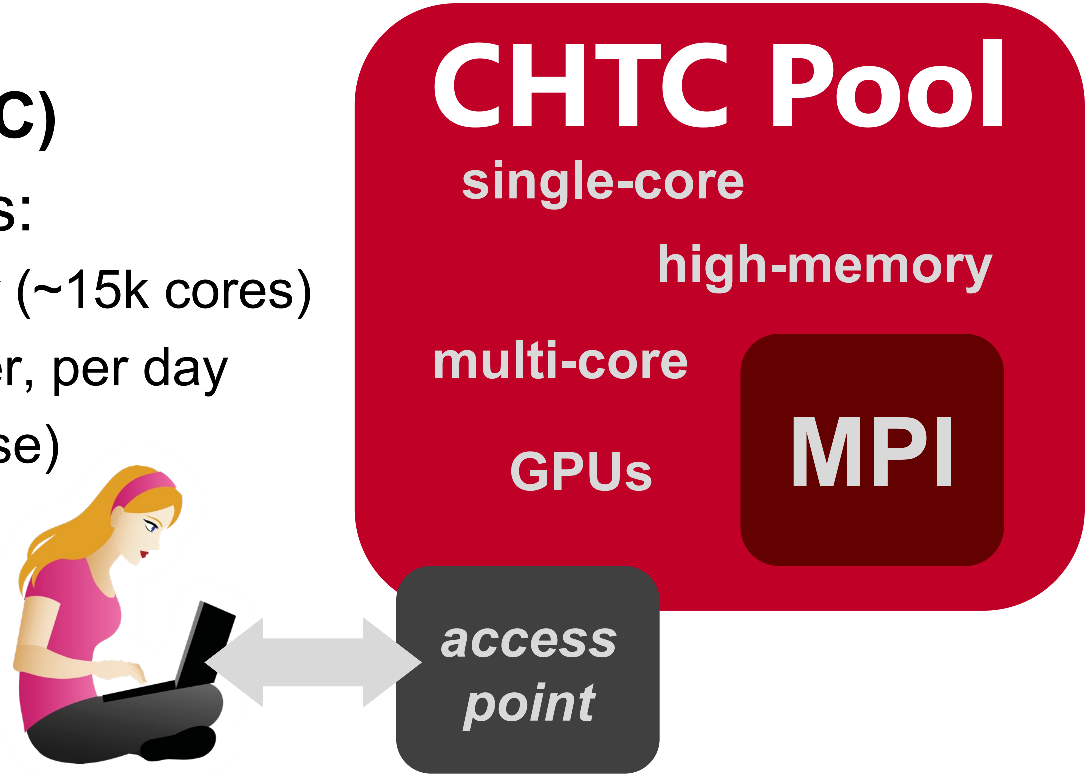
Example Local Cluster

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

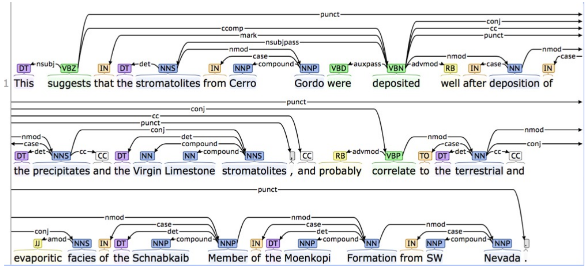
  ~120 million hrs/year (~15k cores)

  Up to 15,000 per user, per day

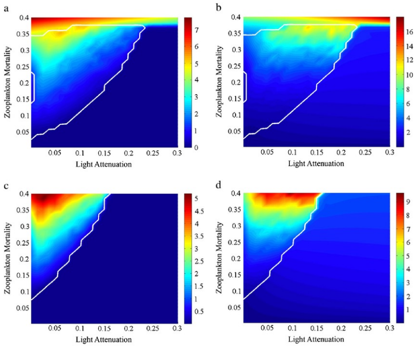
  (~600 cores in use)



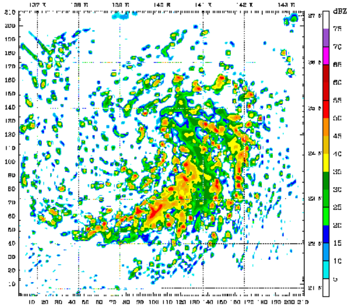
HTC Examples



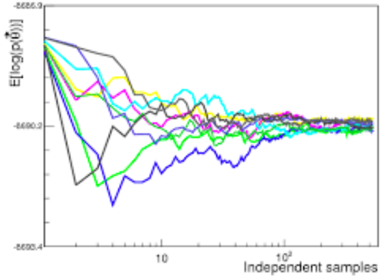
text analysis (most genomics …)



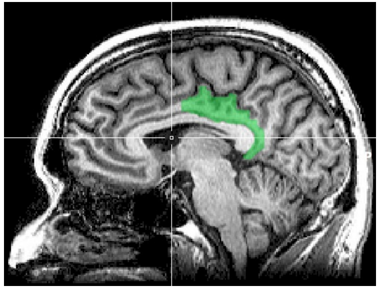
parameter sweeps



multi-start simulations



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



multi-image and multi-sample analysis

Signs of HTC-able work

- Any mention of **numerous** samples, images, models, parameters, etc.
- Nearly anything **written by the primary user** (e.g. c/fortran, Python, R)
  - Break out of loops!
  - Common internal parallelism could really be HTC (e.g. Matlab's 'parfor', 'distributed server', etc.)
- Some community softwares that use **multi-threading or multiprocessing** (e.g. OpenMP)
  - many are simply looping over data portions or independent tasks
  - HTC-able: break up input (or 'parameter' space), turn off multi-threading, combine results
- **Long-running** jobs (especially if non-MPI); see above explanations

Real HTC Use Cases

OSG website 'Spotlight'

https://osg-htc.org/spotlight.html

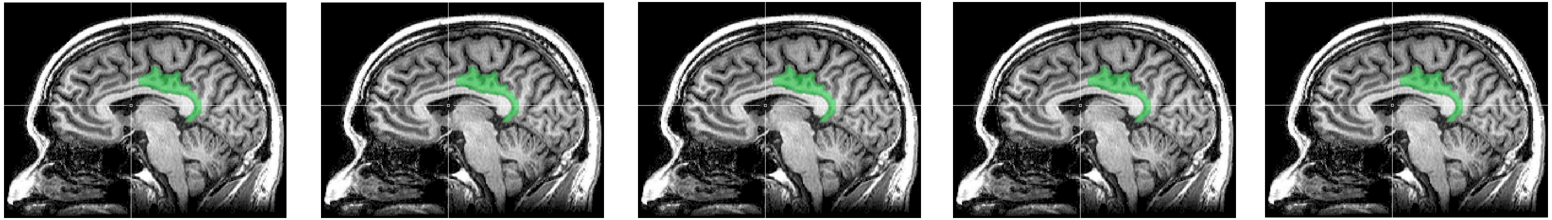
OSG All-Hands Meetings (research talks usually day 1)

https://osg-htc.org/all-hands/

HTCondor Week Presentations (usually first or last day)

https://htcondor.org/past_condor_weeks.html

Example Challenge



You need to process 72 brain images for each of 168 patients. **Each image takes ~1 hour of compute time.**

168 patients x 72 images = ~12000 tasks = ~12000 hrs

Conference is next week.

Distributed Computing

- Use many computers, each running one instance of our program
- Example:
  - **1 laptop (1 core) => 12,000 hrs = ~1.5 years**
  - 1 server (~40 cores) => 750 hrs = ~2 weeks
  - 1 MPI job (400 cores) => 30 hrs = ~1 days
  - **A whole cluster (10,000 cores) = ~1 hour**

What computing resources are available?

- A server?
- A local cluster?
  - Consider: Queue wait time? Can you program MP/MPI? Typical clusters tuned for HPC (large MPI) jobs may not be best for HTC workflows! Could you use even more than that?
- **OSG?**
- Other
  - EGI (European Grid Infrastructure)
  - Other national and regional grids
  - Commercial cloud systems (e.g. HTCondor on AWS)

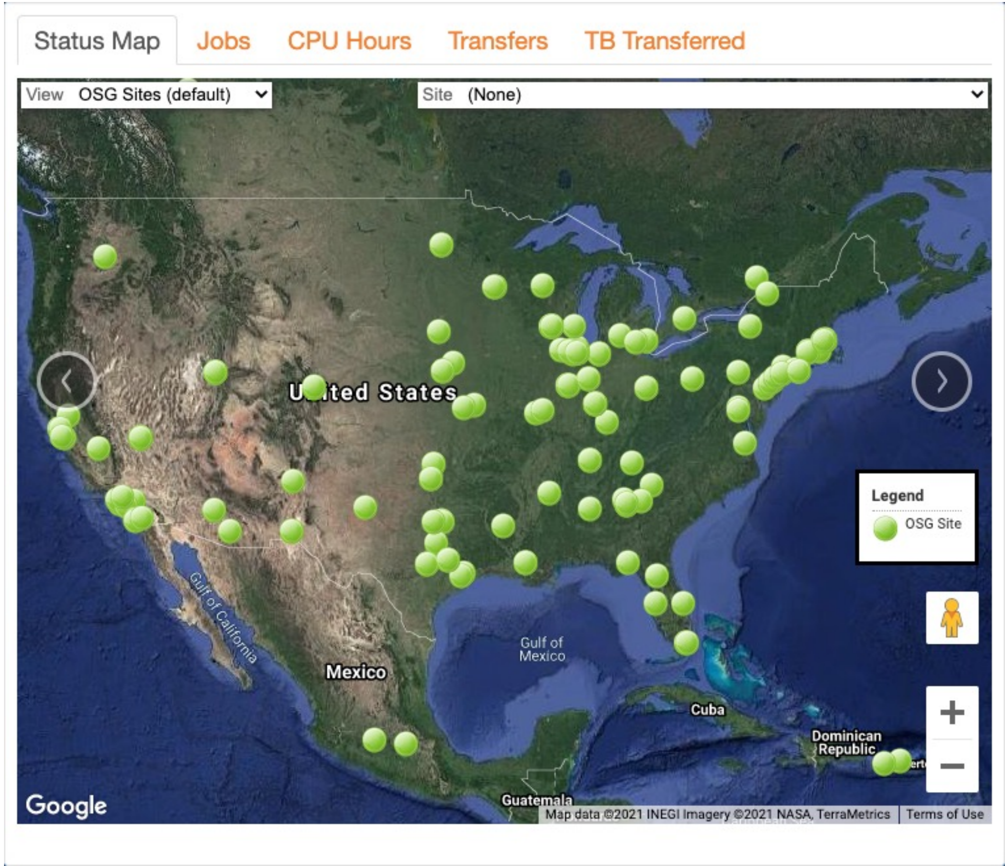
What is the OSG?

a consortium of researchers and institutions who *share* compute and data resources for ***distributed*** **high-throughput computing (dHTC)** in support of open science

Who Participates?

- Researchers
- Science Gateways
- Multi-Institution Collaborations
  - Atlas/CMS (Higg Boson), IceCube, South Pole Telescope, and others
- Academic Institutions and National Laboratories that support the above

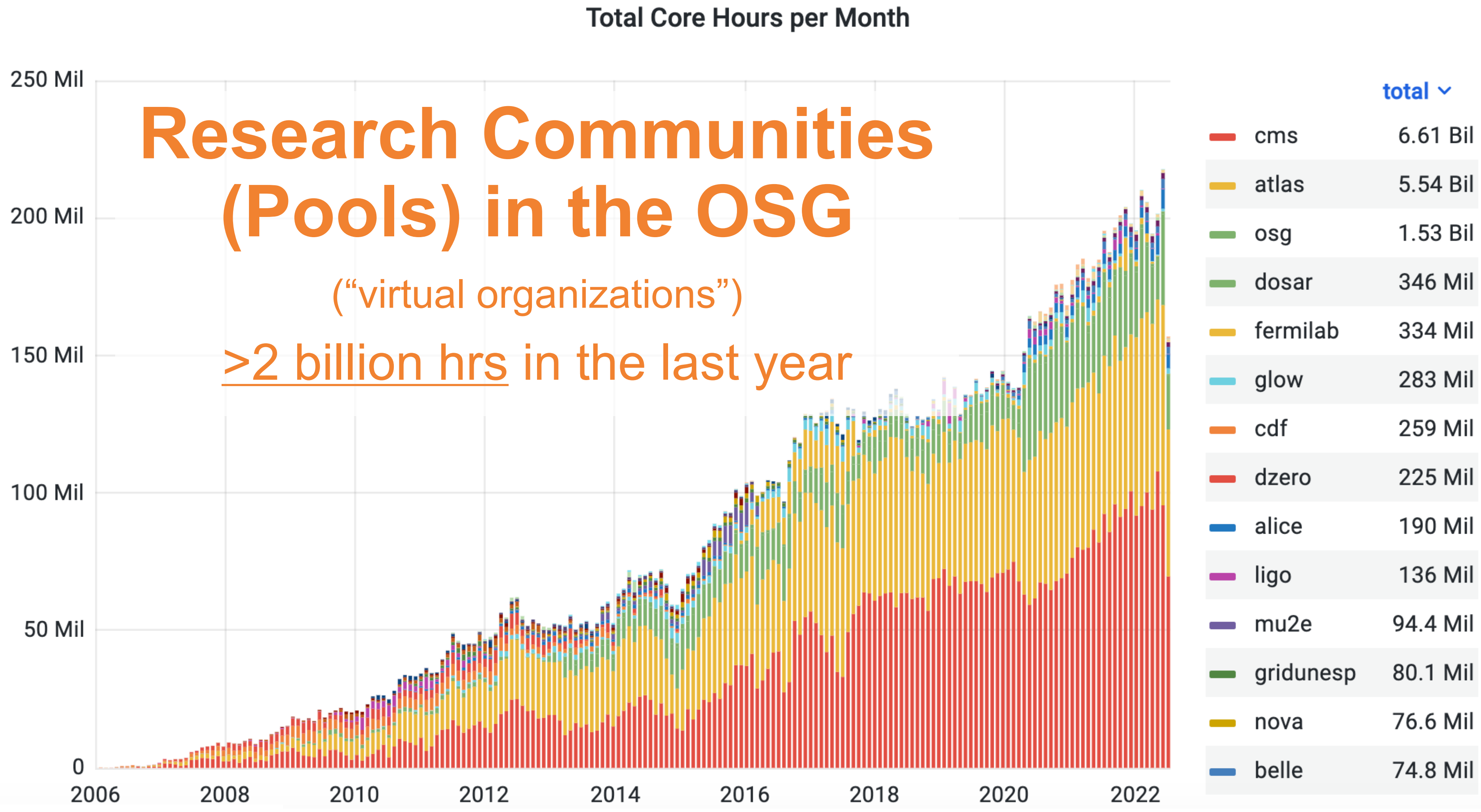
Campuses are critical to OSG's ability to advance research.



Research Communities (Pools) in the OSG

("virtual organizations")

>2 billion hrs in the last year



gracc.opensciencegrid.org

HOW IS CMS SEARCHING FOR THE HIGGS BOSON?

Previous

OSG Supports Multi-Messenger Astronomy.

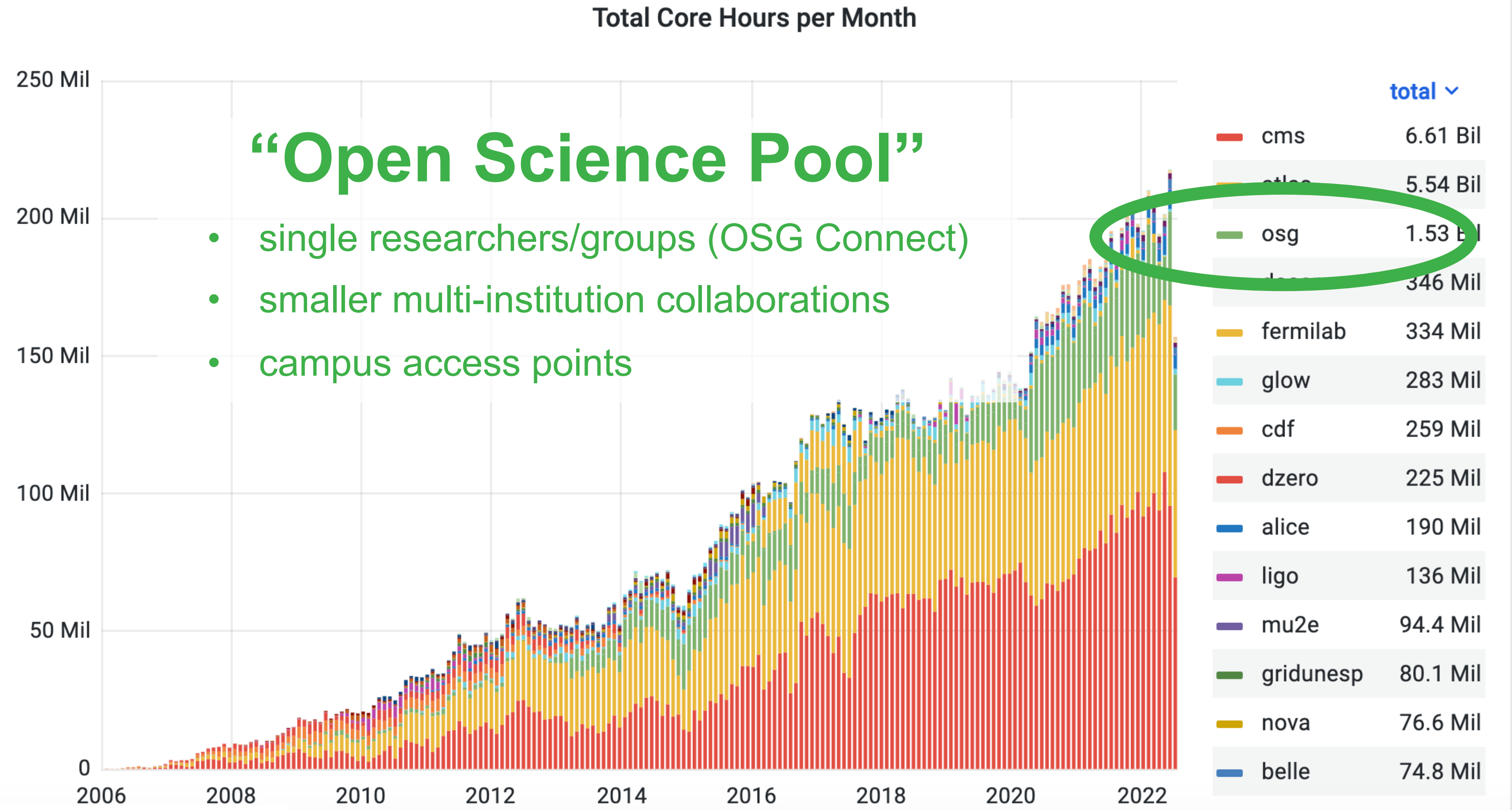
OSG integrates global computing to support detection of colliding neutron stars by LIGO, VIRGO, and DECam.

Read more

Next

“Open Science Pool”

- single researchers/groups (OSG Connect)
- smaller multi-institution collaborations
- campus access points



gracc.opensciencegrid.org

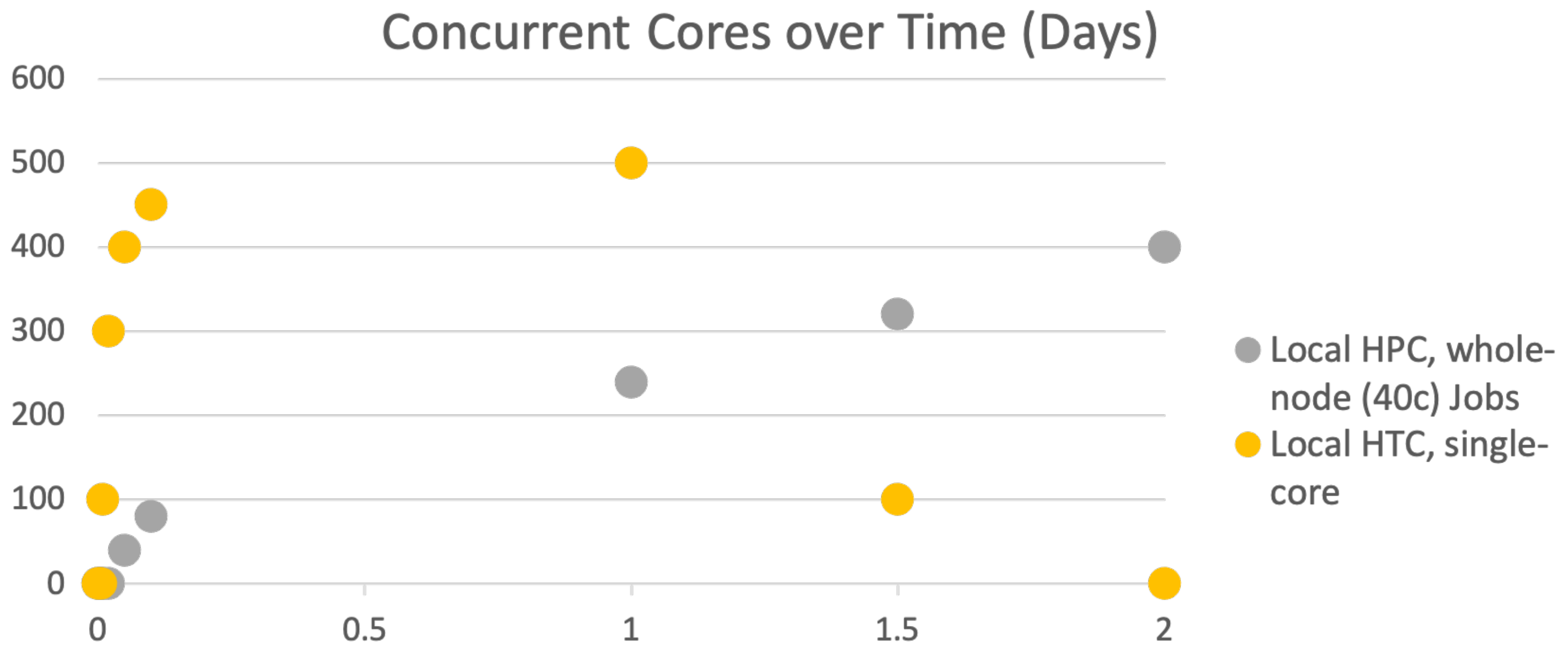
Can the OSPool Help?

	Ideal Jobs!	Still very advantageous	Maybe not, but get in touch!
Expected Throughput, per user	1000s concurrent cores	100s concurrent cores	Let's discuss!
CPU	1 per job	< 8 per job	> 8 per job
Walltime	< 10 hrs*	< 20 hrs*	> 20 hrs
RAM	< few GB	< 40 GB	> 40 GB
Input	< 500 MB	< 10 GB	> 10 GB**
Output	< 1 GB	< 10 GB	> 10 GB**
Software	pre-compiled binaries, containers	Most other than →	Licensed Software, non-Linux

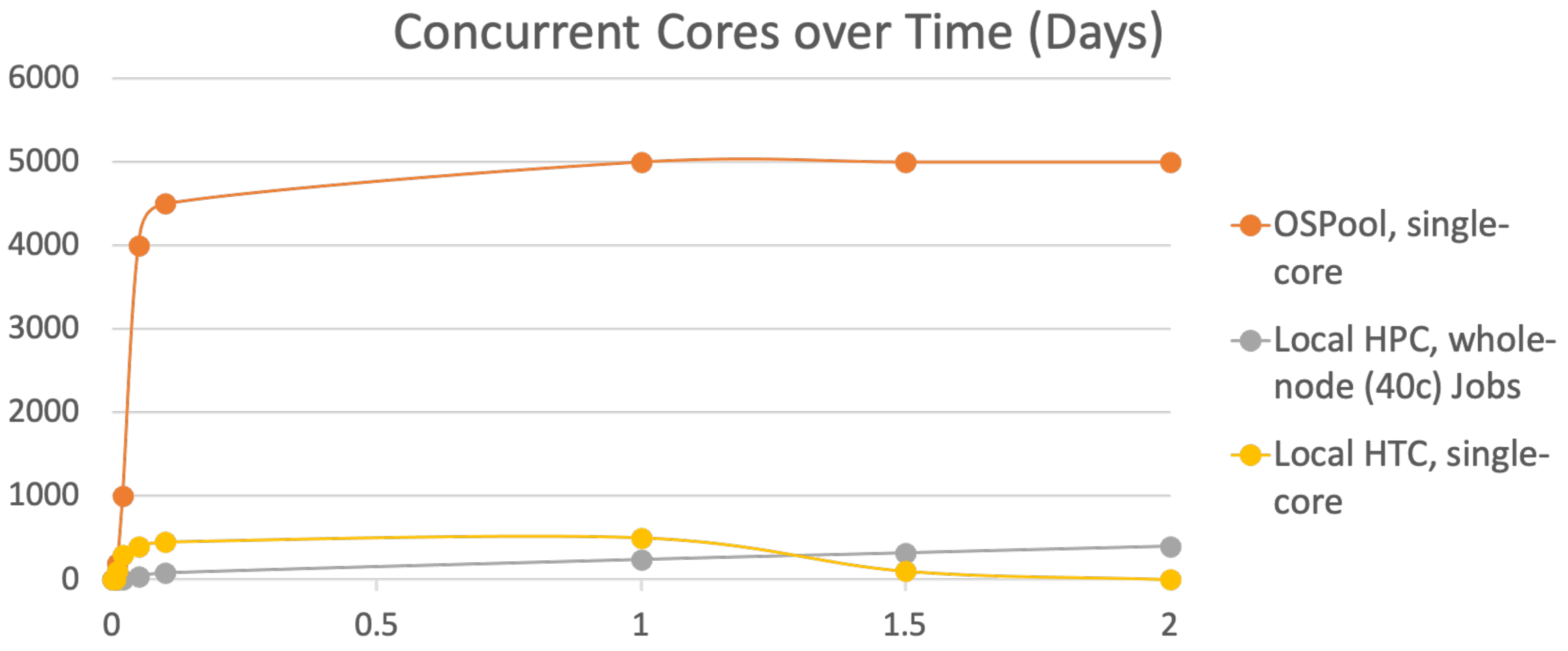
*or checkpointable

** per job; you can work with a large dataset on OSG if it can be split into pieces

Hypothetical Throughput, 12k core hours



Hypothetical Throughput, ~~12k~~ core hours



For Researchers and Campuses

Proactive, personalized facilitation and support for:

- Individual researchers via **OSG Connect**
- Institutions and large collaborations
  - **Share local resources** via the OSG
  - Locally-supported **access points**
    - data and identity federation
    - integration of cloud capacity
  - Local HTC Capacity
    - Learn from OSG's **Research Computing Facilitators**

- **Presentations/Training** in OSG compute execution, HTC Facilitation, and local HTC systems administration

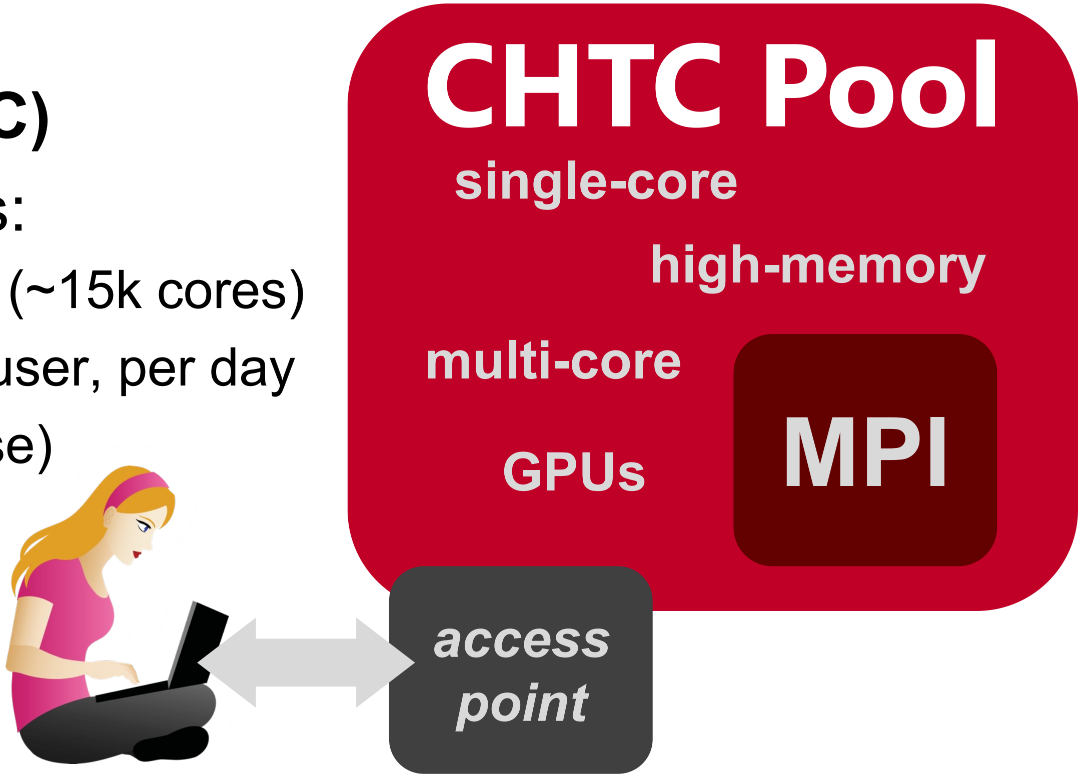
Example Local Cluster

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

  ~120 million hrs/year (~15k cores)

  Up to 15,000hrs per user, per day

  (~600 cores in use)



Intro to Job Submission with HTCondor

Overview

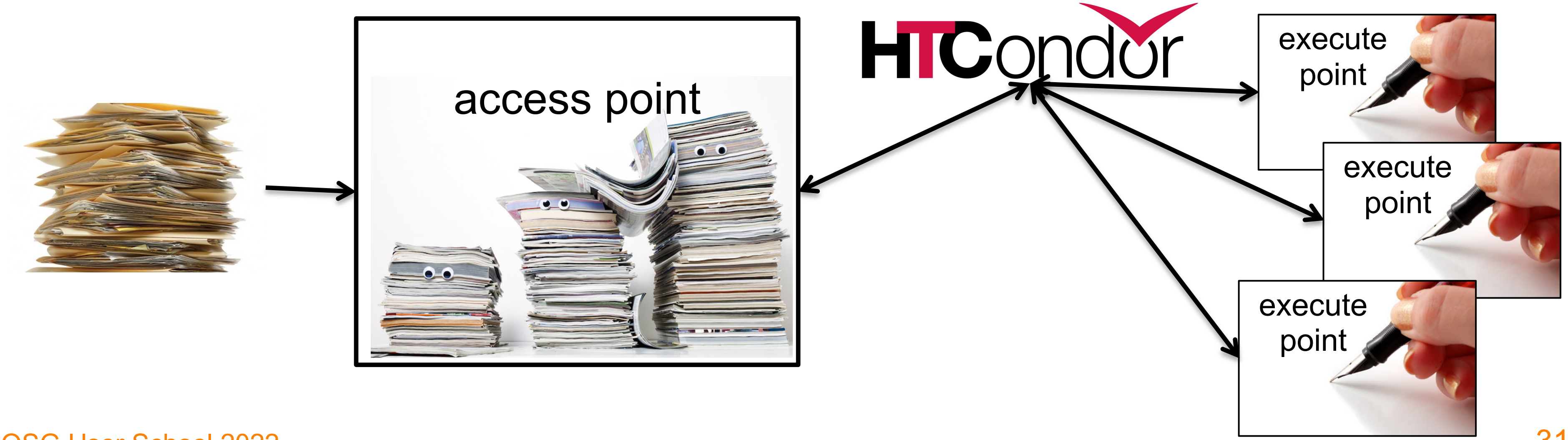
- How does the HTCondor job scheduler work?
- How do you run, monitor, and review jobs?
- Best ways to submit multiple jobs (what we're here for, *right?*)
- Testing, tuning, and troubleshooting to scale up.

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 OSG!!**
- Miron Livny
  - Professor, UW-Madison Computer Sciences
  - CHTC Director, OSG Technical Director

HTCondor -- How It Works

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



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

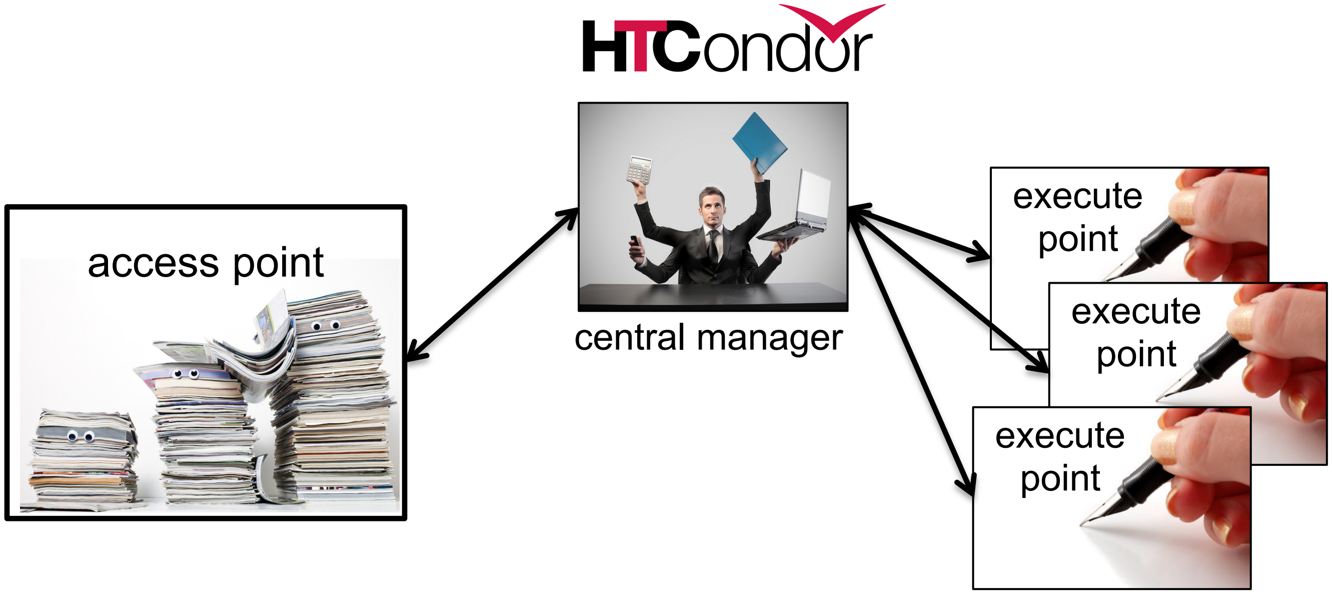
Terminology: *Machine, Slot*

- ***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)**
  - may correspond 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

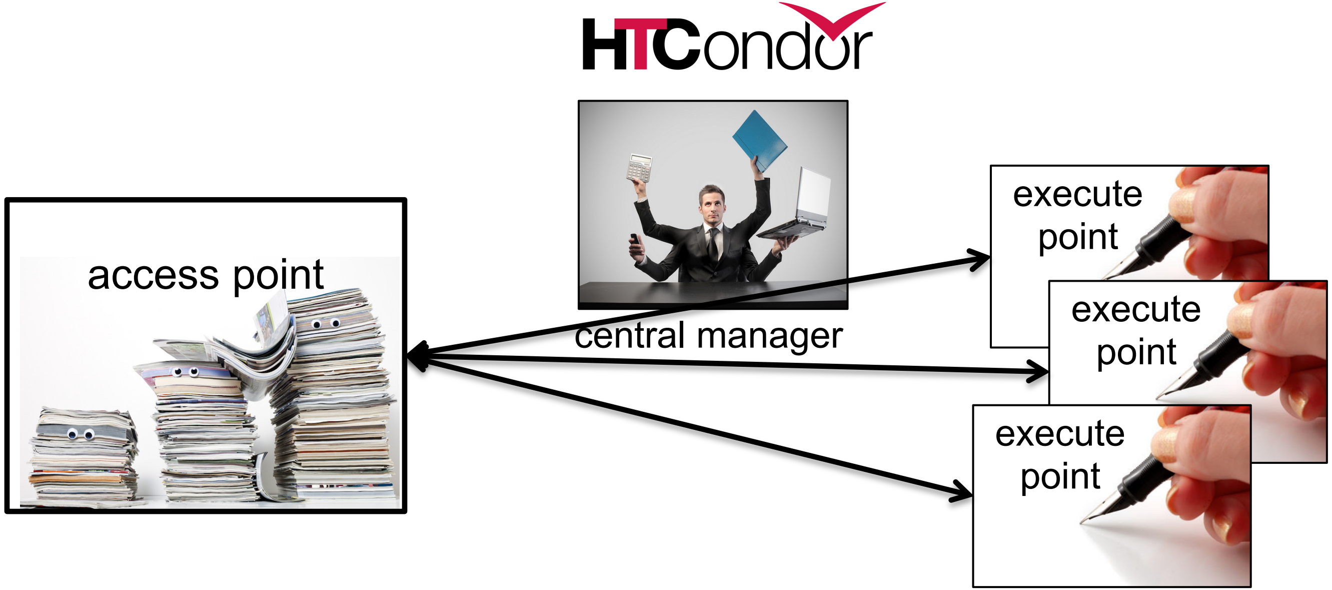
Job Matching

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

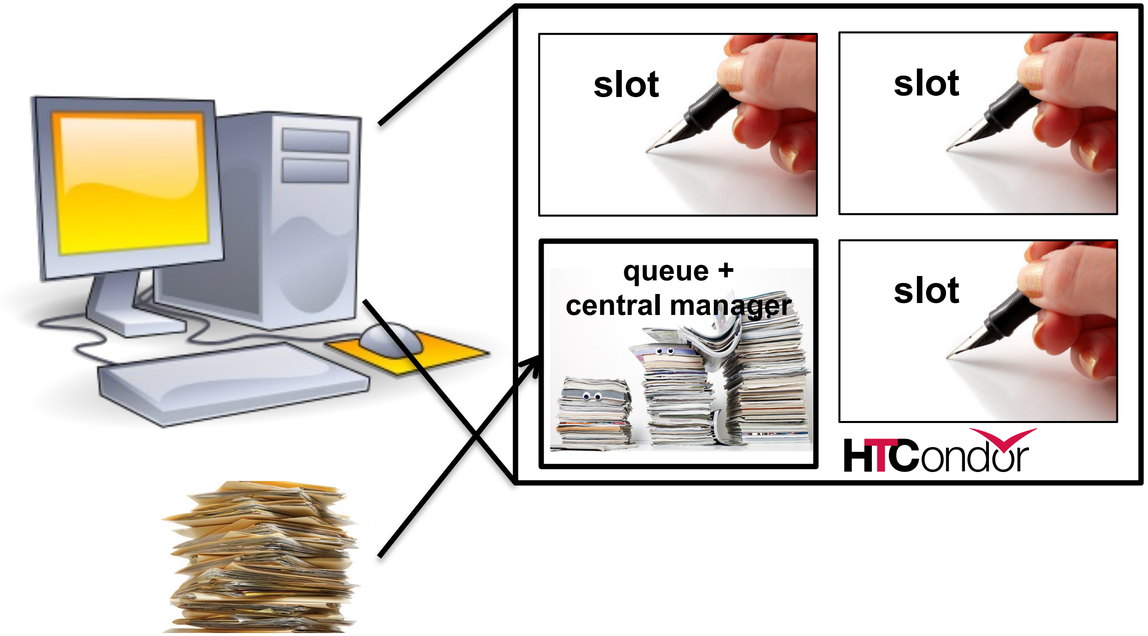


Job Execution

- Then the access and execute points communicate directly.



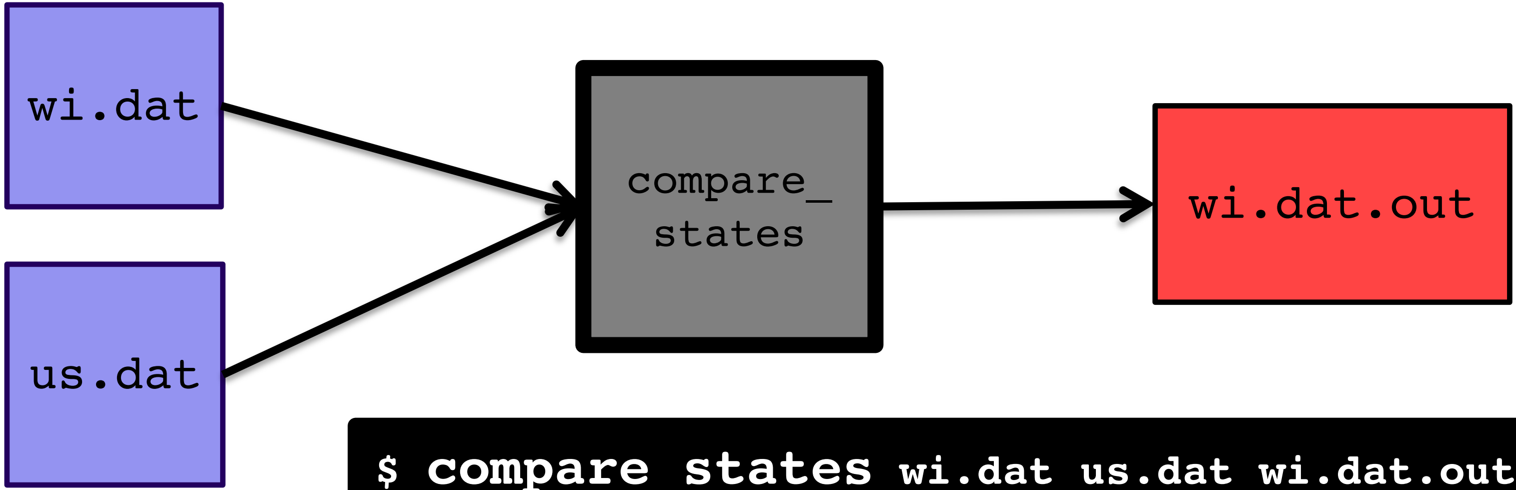
Single Computer



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
```

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

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`**: *final* keyword indicating "create 1 job" according to the above

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> @ 05/01/22 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: learn.chtc.wisc.edu : <128.104.101.92> @ 05/01/22 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
```

Access Point

```
(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
```

Access Point

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

compare_states
wi.dat
us.dat

Execute Point

```
(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
```

Access Point

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

Execute Point

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

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
```

Access Point

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

stderr
stdout
wi.dat.out

Execute Point

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

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
```

Access Point

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

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]

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 Requests

- Jobs are nearly always using a ***portion of*** a machine, 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

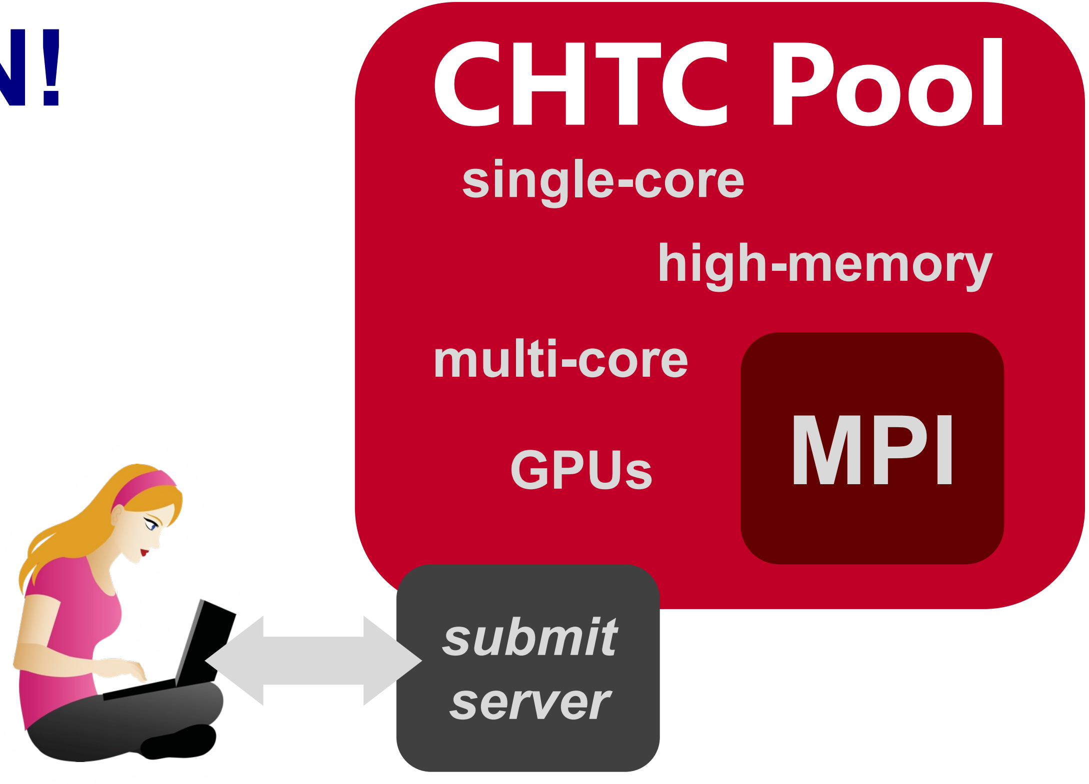
Ideal OSPool Job Sizes

	Ideal Jobs!	Still very advantageous	Maybe not, but get in touch!
Expected Throughput, per user	1000s concurrent cores	100s concurrent cores	Let's discuss!
CPU	1 per job	< 8 per job	> 8 per job
Walltime	< 10 hrs*	< 20 hrs*	> 20 hrs
RAM	< few GB	< 40 GB	> 40 GB
Input	< 500 MB	< 10 GB	> 10 GB**
Output	< 1 GB	< 10 GB	> 10 GB**
Software	pre-compiled binaries, containers	Most other than →	Licensed Software, non-Linux

*or checkpointable

** per job; you can work with a large dataset on OSG if it can be split into pieces

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!**
- **Exercises in THIS unit** are important to complete *in order*, before moving on! (You can save “bonus” exercises for later.)

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