Introduction to Job Submission with HTCondor

July 13, 2026

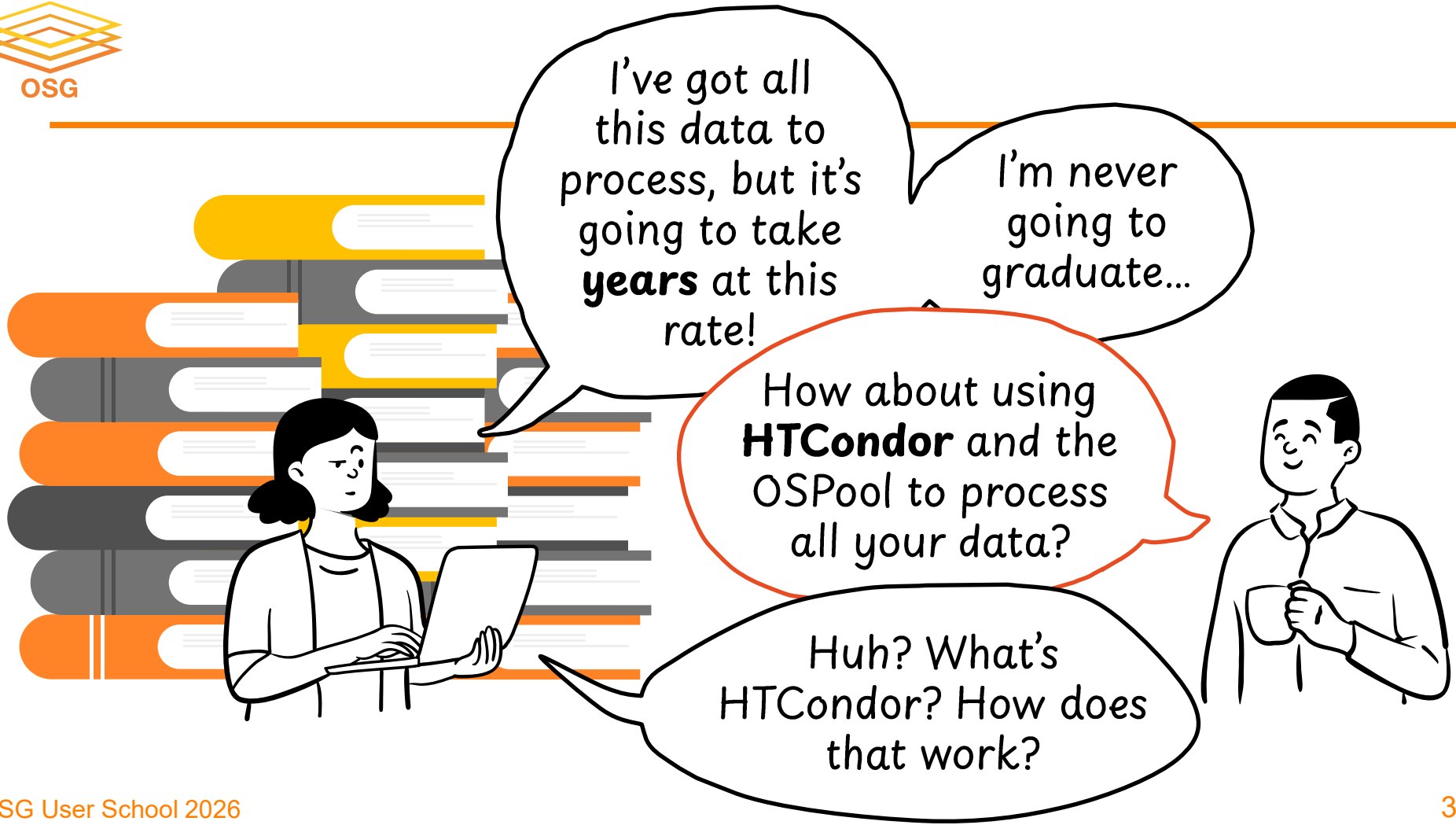
Amber Lim

Slides adapted from Lauren Michaels, Rachel Lombardi, Andrew Owen
This work was supported by NSF grants MPS-1148698, OAC-1836650, and OAC-2030508

Objective

By the end of this session, you should be able to:

- Describe how the HTCondor manages workflows
- Translate your computational tasks to “jobs”
- Run, monitor, and review your jobs


I've got all this data to process, but it's going to take **years** at this rate!
I'm never going to graduate...
How about using **HTCondor** and the OSPool to process all your data?
Huh? What's HTCondor? How does that work?

History of HTCondor

HTCondor History and Status

- Beginnings
  - Started in 1988 as a “cycle scavenger”
- Today
  - Distributed computing framework
  - Developed at CHTC by professional developers
  - Used all over the world, by:
    - Campuses, national labs, Einstein/Folding@Home
    - Dreamworks, Boeing, SpaceX, investment firms, …
    - **The OSPool!!**

Miron Livny

- Professor, UW-Madison Computer Sciences
- CHTC Director, OSG Technical Director

How does HTCondor work?

HTCondor — How It Works

1. You decide **what kind of food** you want.

2. You tell your order to the **waiter**. The waiter relays your order to the **kitchen manager**.

3. The **manager** delegates your orders to **chefs**.

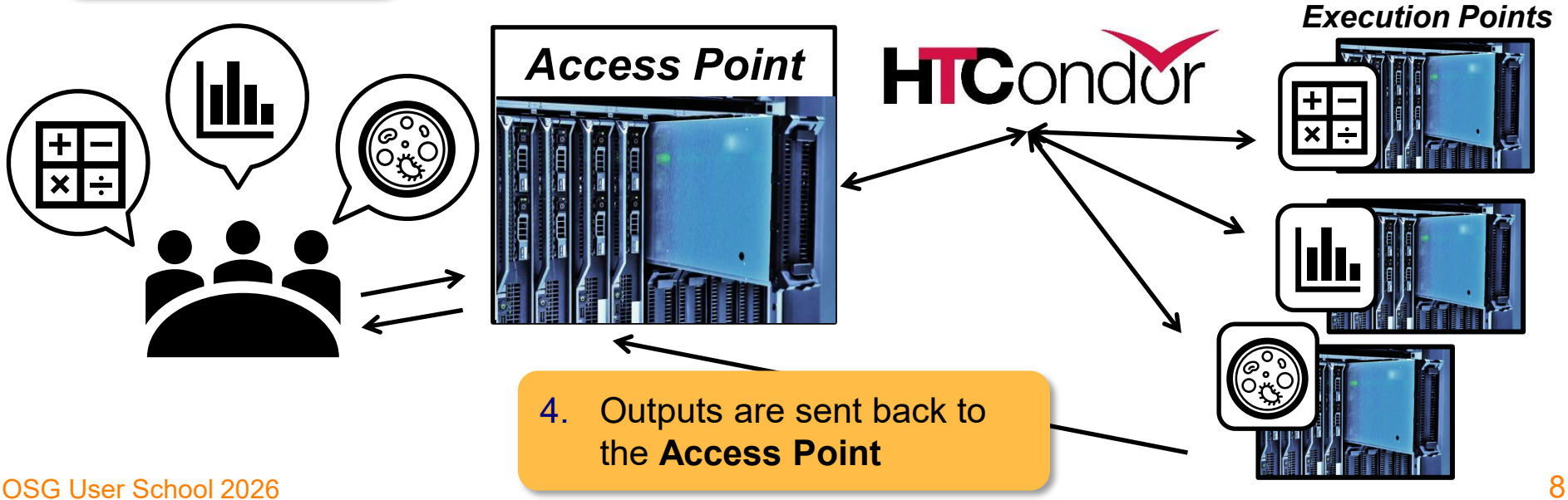
4. Finished orders are given to the **waiter**, who brings you your food.

HTCondor — How It Works

1. You create and describe your tasks on the **Access Point**.

2. From the **Access Point**, you submit tasks to **HTCondor**.

3. HTCondor schedules your tasks to run on **Execution Points**



4. Outputs are sent back to the **Access Point**

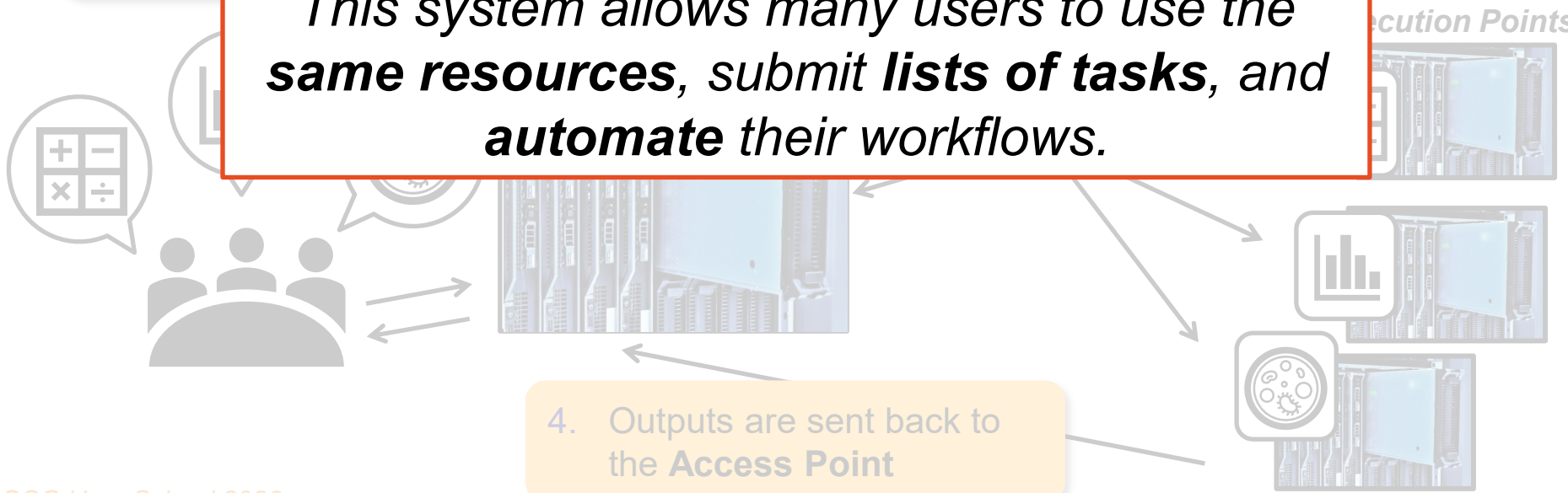
HTCondor — How It Works

1. You create and describe your tasks on the **Access Point**

2. From the **Access Point**, you submit tasks to **HTCondor**.

3. HTCondor schedules your tasks to run on **Execution Points**

*This system allows many users to use the **same resources**, submit **lists of tasks**, and **automate** their workflows.*



4. Outputs are sent back to the **Access Point**

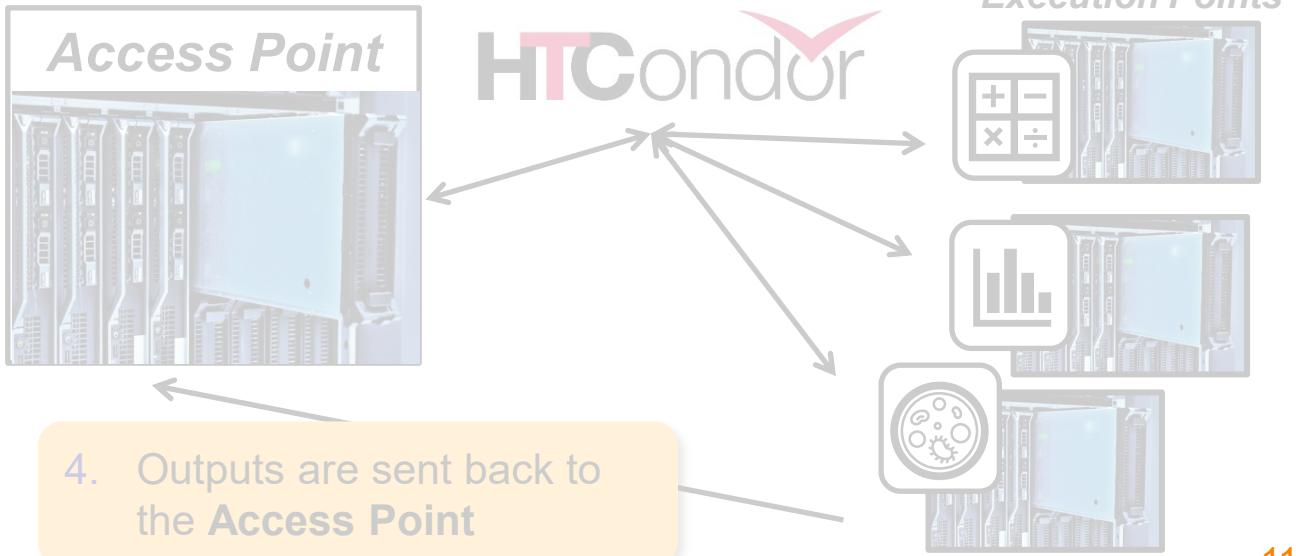
Basics of submitting jobs

HTCondor — How It Works

1. You create and describe your tasks on the **Access Point**.

2. From the **Access Point**, you submit tasks to **HTCondor**.

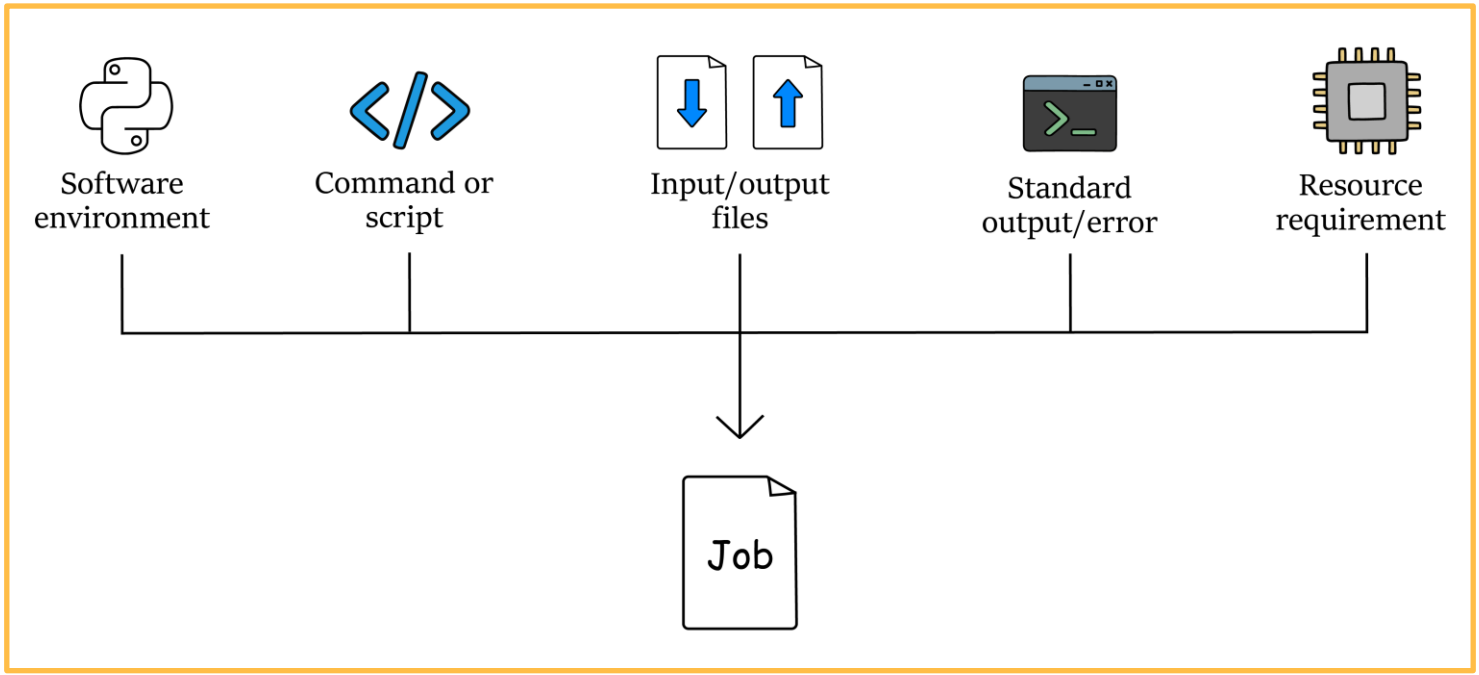
3. HTCondor schedules your tasks to run on **Execution Points**



4. Outputs are sent back to the **Access Point**

Terminology: *Job*

Job: An independently-scheduled unit of computing work



Components of a job

Software environment

What software, packages, and libraries do you need?

Command or script

How do you run your computation?

Input files/output files

What input files are needed? What output files are created?

Standard output/error

Where do you save messages printed to the screen?

Requirements

What resources (CPU, GPU, memory, disk) do you need?

Components of a job

Software environment

What software, packages, and libraries do you need?

We will cover Software on Tuesday!

Command or script

How do you run your computation?

Input files/output files

What input files are needed? What output files are created?

Standard output/error

Where do you save messages printed to the screen?

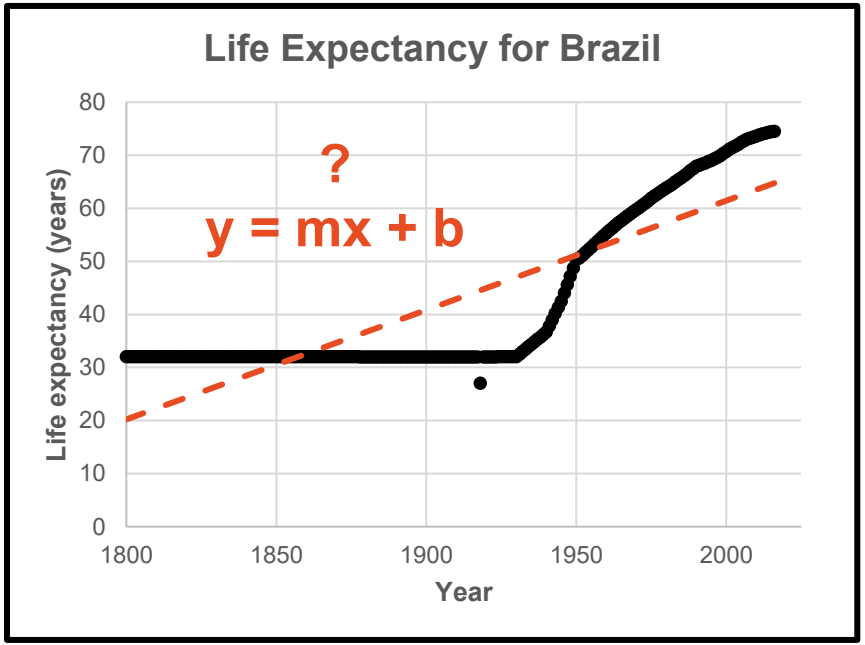
Requirements

What resources (CPU, GPU, memory, disk) do you need?

Job Example

I'm a researcher trying to model life expectancies of people in different countries in different years.

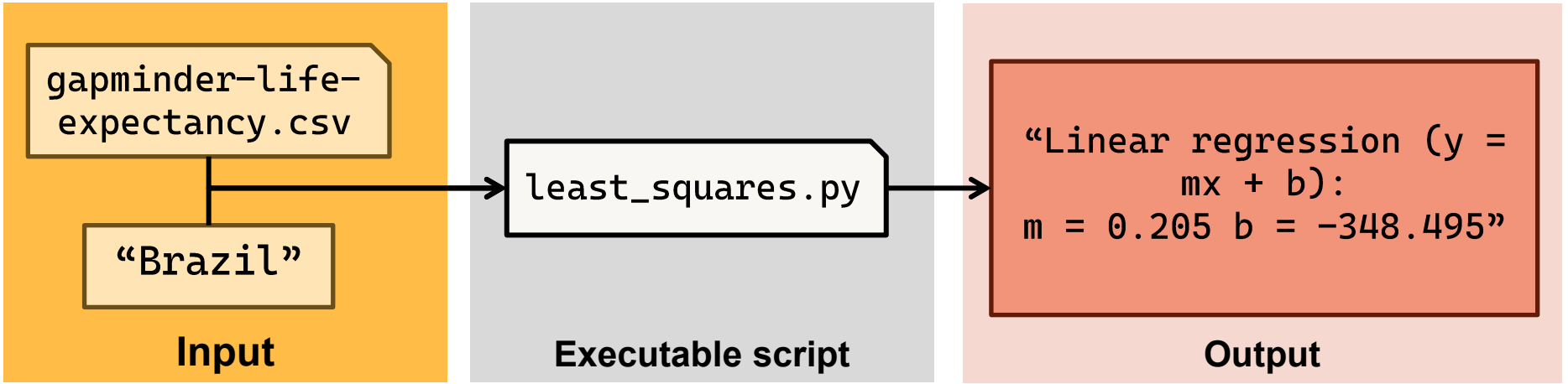
I want to calculate the **linear regression** for specified countries.



Job Example

On my laptop:

```
$  ./least_squares.py gapminder-life-expectancy.csv Brazil
Linear regression (y = mx + b):
m = 0.205 b = -348.495
```



Components of a job

Software environment

Python (os, sys, csv)

Command or script

./least_squares.py gapminder-life-expectancy.csv Brazil

Input files/output files

gapminder-life-expectancy.csv

Standard output/error

I want to save it in a text file, maybe with “Brazil” in its name.

Requirements

Very minimal! 1 cpu, a little memorry, a little disk

HTCondor Submit File

```
container_image = /cvmfs/singularity.opensciencegrid.org/htc/ubuntu:24.04

shell = ./least_squares.py gapminder-life-expectancy.csv Brazil

transfer_input_files = least_squares.py, gapminder-life-expectancy.csv

log = log/Brazil.log
error = outputs/Brazil.err
output = outputs/Brazil.out

request_cpus = 1
request_memory = 1GB
request_disk = 1GB

queue
```

Let's break this down, piece-by-piece.

container_image

```
container_image =
    /cvmfs/singularity.opensciencegrid.org/htc/ubuntu:24.04

shell = ./least_squares.py
    gapminder-life-expectancy.csv
    Brazil

transfer_input_files =
    least_squares.py, gapminder-
    life-expectancy.csv

log = log/Brazil.log
error = outputs/Brazil.err
output = outputs/Brazil.out
```

`container_image` points to your container, which contains your software environment.

Stay tuned for the software lecture on Tuesday!

I'm using this container because I know it has Python.

shell

```
container_image =
    /cvmfs/singularity.openscienc
    egrid.org/htc/ubuntu:24.04

shell = ./least_squares.py
    gapminder-life-expectancy.csv
    Brazil

transfer_input_files =
    least_squares.py, gapminder-
    life-expectancy.csv

log = log/Brazil.log
error = outputs/Brazil.err
output = outputs/Brazil.out
```

List your `shell` command (`executable` and any `arguments`)

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

```
$  ./least_squares.py gapminder-life-expectancy.csv Brazil
```

transfer_input_files

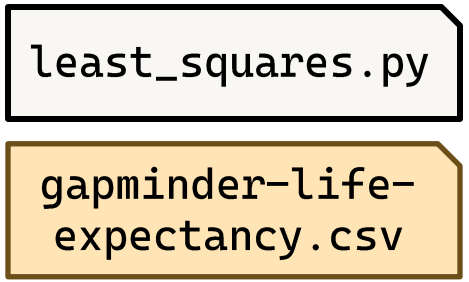
```
container_image =
    /cvmfs/singularity.openscienc
    egrid.org/htc/ubuntu:24.04

shell = ./least_squares.py
    gapminder-life-expectancy.csv
    Brazil

transfer_input_files =
    least_squares.py, gapminder-
    life-expectancy.csv

log = log/Brazil.log
error = outputs/Brazil.err
output = outputs/Brazil.out
```

Provide a comma-separated list of **input files to transfer** to the Execution Point.



The Access Point and the Execution Point are **separate** machines, so we must specify which files to transfer.

What about output files?

```
container_image =
    /cvmfs/singularity.openscienc
    egrid.org/htc/ubuntu:24.04

shell = ./least_squares.py
    gapminder-life-expectancy.csv
    Brazil

transfer_input_files =
    least_squares.py, gapminder-
    life-expectancy.csv

log = log/Brazil.log
error = outputs/Brazil.err
output = outputs/Brazil.out
```

HTCondor will transfer back all new and changed files (output) from the **top-level directory** of the job, automatically.

We will talk more about this later!

log, output, error

```
container_image =
    /cvmfs/singularity.opensciencegrid.org/htc/ubuntu:24.04

shell = ./least_squares.py
    gapminder-life-expectancy.csv
    Brazil

transfer_input_files =
    least_squares.py, gapminder-
    life-expectancy.csv

log = log/Brazil.log
error = outputs/Brazil.err
output = outputs/Brazil.out
```

`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)

Resource requests

```
log = log/Brazil.log
error = outputs/Brazil.err
output = outputs/Brazil.out

request_cpus = 1
request_memory = 1GB
request_disk = 1GB

queue
```

`request_cpus`, `request_disk`, `request_memory`:

the resources your job needs.

Resource requests

```
log = log/Brazil.log
error = outputs/Brazil.err
output = outputs/Brazil.out

request_cpus = 1
request_memory = 1GB
request_disk = 1GB

queue
```

Very important to request appropriate resources (***memory***, ***cpus***, ***disk***)

- **requesting too little**: causes problems for your 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

HTCondor Submit File

```
log = log/Brazil.log
error = outputs/Brazil.err
output = outputs/Brazil.out

request_cpus = 1
request_memory = 1GB
request_disk = 1GB

queue
```

`queue`: keyword indicating the number of jobs to queue

- must be the last line of the submit file

- has different syntax options we will learn later!

Let’s pause!

What questions do you have about—

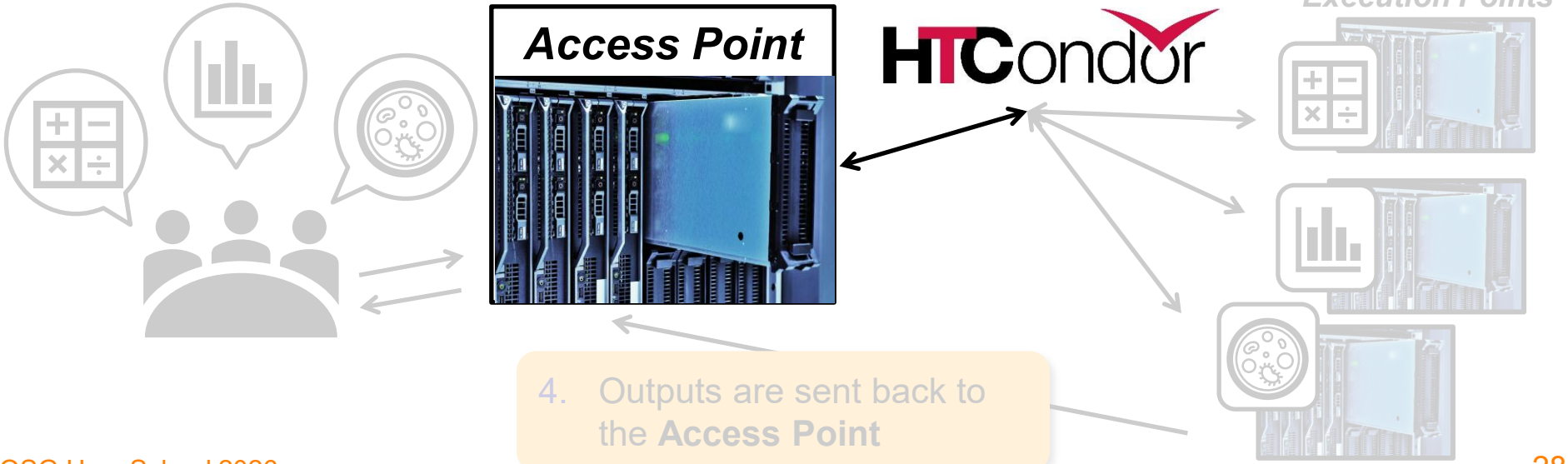
- How HTCondor works?
- Components of a job?
- The HTCondor submit file?

HTCondor — How It Works

1. You create and describe your tasks on the **Access Point**.

2. From the **Access Point**, you submit tasks to **HTCondor**.

3. HTCondor schedules your tasks to run on **Execution Points**



4. Outputs are sent back to the **Access Point**

Submitting and monitoring HTCondor jobs

Submitting and Monitoring

- Submit jobs on the Access Point
- To submit jobs: `condor_submit submit_file`
- To monitor submitted jobs: `condor_q`

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

$ condor_q
-- Schedd: ap40.uw.osg-htc.org : <128.105.68.62:9618> @ 08/01/24 10:35:54
OWNER  BATCH_NAME              SUBMITTED   DONE   RUN    IDLE  TOTAL JOB_IDS
alice  ID:128                 8/1  10:05      _      _      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` …
  - Only shows your jobs and not anyone else's
  - Groups jobs that were submitted together ("batch" or "cluster")
  - Only shows active batches
  - Can limit `condor_q` by `username`, `ClusterId` or full `JobId`.

```
$ condor_q
-- Schedd: ap40.uw.osg-htc.org : <128.105.68.62:9618> @ 08/01/24 10:35:54
OWNER  BATCH_NAME              SUBMITTED   DONE   RUN    IDLE  TOTAL JOB_IDS
alice  least_squares_list     8/1  10:09      3     4       3     10 129.0-9

10 jobs; 3 completed, 0 removed, 3 idle, 4 running, 0 held, 0 suspended
```

`JobId = ClusterID.ProcID`

More about `condor_q`

- To see individual job details, use:
  `condor_q -nobatch`

```
$ condor_q -nobatch
-- Schedd: ap40.uw.osg-htc.org : <128.105.68.62:9618>
 ID          OWNER      SUBMITTED     RUN_TIME ST PRI SIZE CMD
129.0        alice      8/1  10:09   0+00:00:00 I  0   0.0 least_squares.py
129.1        alice      8/1  10:09   0+00:00:00 R  0   0.0 least_squares.py
 ...

7 jobs; 0 completed, 0 removed, 3 idle, 4 running, 0 held, 0 suspended
```

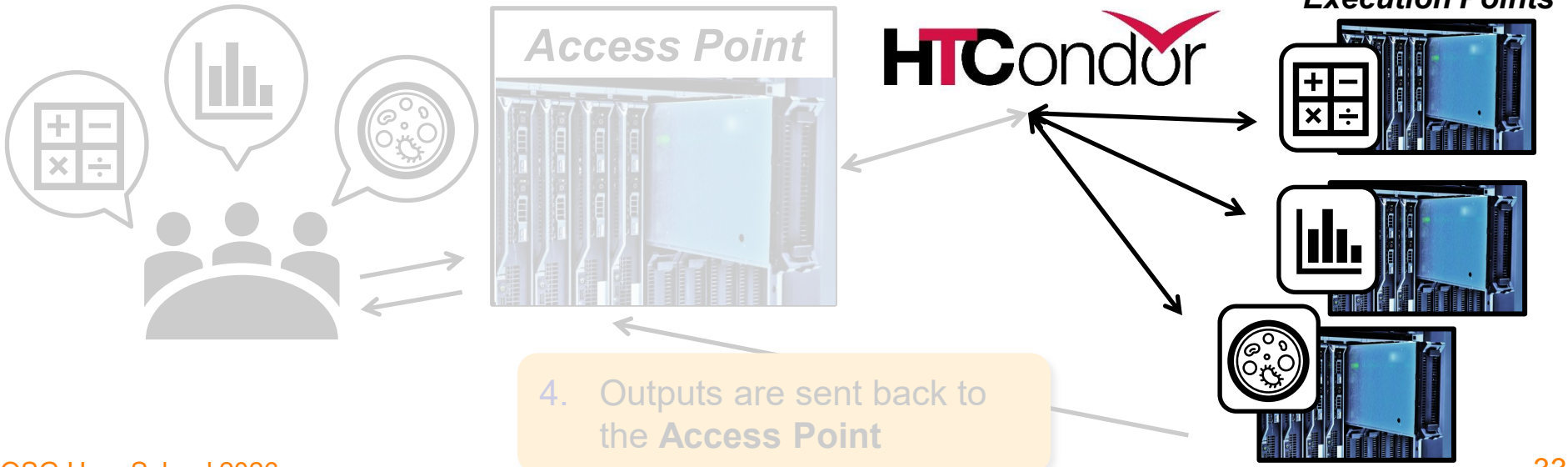
- We will use the `-nobatch` option in the following slides to see extra detail about what is happening with a job

HTCondor — How It Works

1. You create and describe your tasks on the **Access Point**.

2. From the **Access Point**, you submit tasks to **HTCondor**.

3. HTCondor schedules your tasks to run on **Execution Points**



4. Outputs are sent back to the **Access Point**

Job Idle

```
$ condor_q -nobatch
-- Schedd: ap40.uw.osg-htc.org : <128.105.68.62:9618>
 ID          OWNER      SUBMITTED     RUN_TIME ST PRI SIZE CMD
128.0        alice      8/1  10:05   0+00:00:00 I  0   0.0 compare_states wi.dat us.dat

Total for query: 1 jobs; 0 completed, 0 removed, 1 idle, 0 running, 0 held, 0 suspended
```

Access Point

```
(submit_dir)/
├── gapminder-life-expectancy.csv
├── least_squares.py
├── least_squares.sub
└── log/
    └── Brazil.log
```

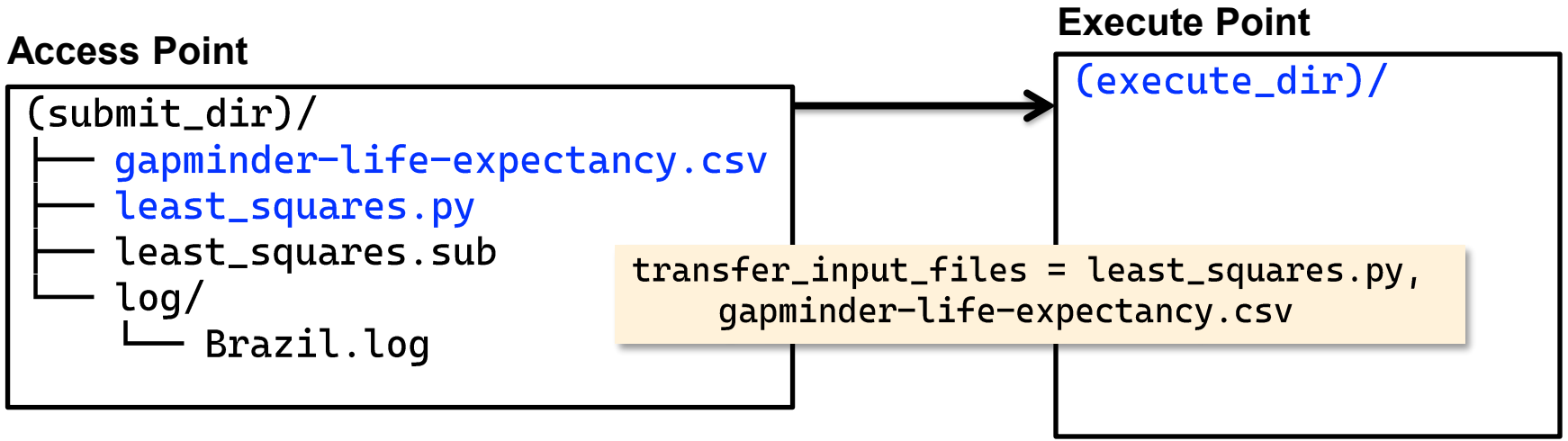
As soon as we submit the job, HTCondor starts tracking it in the **log** file, as specified by the submit file:

```
log = log/Brazil.log
```

Job Starts

```
$ condor_q -nobatch
-- Schedd: ap40.uw.osg-htc.org : <128.105.68.62:9618>
 ID          OWNER      SUBMITTED     RUN_TIME ST PRI SIZE CMD
128.0        alice      8/1  10:05   0+00:00:00 <  0   0.0 compare_states wi.dat us.dat

Total for query: 1 jobs; 0 completed, 0 removed, 1 idle, 0 running, 0 held, 0 suspended
```



Job Running

```
$ condor_q -nobatch
-- Schedd: ap40.uw.osg-htc.org : <128.105.68.62:9618>
 ID          OWNER      SUBMITTED     RUN_TIME ST PRI SIZE CMD
128.0        alice      8/1  10:05   0+00:00:00 R  0   0.0 compare_states wi.dat us.dat

Total for query: 1 jobs; 0 completed, 0 removed, 0 idle, 1 running, 0 held, 0 suspended
```

Access Point

```
(submit_dir)/
├── gapminder-life-expectancy.csv
├── least_squares.py
├── least_squares.sub
└── log/
    └── Brazil.log
```

Execution Point

```
(execute_dir)/
├── gapminder-life-expectancy.csv
├── least_squares.py
├── _condor_stdout
├── _condor_stderr
├── extra_output.txt
└── subdir/
    └── more_output.txt
```

Job Completes

```
$ condor_q -nobatch
-- Schedd: ap40.uw.osg-htc.org : <128.105.68.62:9618>
 ID          OWNER      SUBMITTED     RUN_TIME ST PRI SIZE CMD
128.0        alice      8/1  10:05   0+00:00:00 > 0   0.0 compare_states wi.dat us.dat

Total for query: 1 jobs; 0 completed, 0 removed, 0 idle, 1 running, 0 held, 0 suspended
```

Access Point

```
(submit_dir)/
├── gapminder-life-expectancy.csv
├── least_squares.py
├── least_squares.sub
└── log/
    └── Brazil.log
```

Execution Point

```
(execute_dir)/
├── gapminder-life-expectancy.csv
├── least_squares.py
├── _condor_stdout
├── _condor_stderr
├── extra_output.txt
└── subdir/
    └── more_output.txt
```

```
_condor_stdout
_condor_stderr
extra_output.txt
```

Job Completes (cont.)

```
$ condor_q -nobatch
-- Schedd: ap40.uw.osg-htc.org : <128.105.68.62:9618>
 ID          OWNER      SUBMITTED     RUN_TIME ST PRI SIZE CMD

Total for query: 0 jobs; 0 completed, 0 removed, 0 idle, 0 running, 0 held, 0 suspended
```

Access Point

```
(submit_dir)/
├── extra_output.txt
├── gapminder-life-expectancy.csv
├── least_squares.py
├── least_squares.sub
├── log/
│   └── Brazil.log
└── outputs/
    └── Brazil.err
    └── Brazil.out
```

Job completed →
Disappears from `condor_q` *output!*

Standard **output** and **error** are renamed, as specified by the submit file:

```
error = outputs/Brazil.err
output = outputs/Brazil.out
```

Watching Job Progress with `condor_watch_q`

Watching Progress of Jobs

To get a live update of the progress of your jobs, use `condor_watch_q`.

This command does an initial `condor_q` and then tracks the entries of the corresponding **log** file(s)

```
$ condor_watch_q
BATCH         IDLE  RUN  DONE  TOTAL  JOB_IDS
ID: 129        10    -     -     10   129.0 ... 129.9 [──────────────────]

[──────────────────────────────────────────────────────────────────────────]

Total: 10 jobs; 10 idle

Updated at 2026-07-10 16:20:26
Press any key to exit
```

Watching Progress of Jobs

As the work progresses, output updates with changes to the progress bar.

(updates every second)

```
$ condor_watch_q
BATCH         IDLE  RUN  DONE  TOTAL  JOB_IDS
ID: 129          9    1     -     10  129.0 ... 129.9 [==──────────────────]

[═══──────────────────────────────────────────────────────────────────────]

Total: 10 jobs; 9 idle, 1 running

Updated at 2026-07-10 16:22:01
Press any key to exit
```

Watching Progress of Jobs

- Yellow hyphens (-) = “idle”
- Blue greater than signs (>) = “transferring files”
- Blue equal signs (=) = “running”
- Green number signs (#) = “completed”
- Red exclamation marks (!) = “hold”

```
$ condor_watch_q
BATCH         IDLE   RUN   DONE   TOTAL   JOB_IDS
ID: 129         3     4      3      10    129.0 ... 129.9 [######=========-------]

[####################==========================================-------------------]

Total: 10 jobs; 3 completed, 4 idle, 3 running

Updated at 2026-07-10 16:23:43
Press any key to exit
```

Watching Progress of Jobs

To exit out of the `condor_watch_q` view, press any key.

```
$ condor_watch_q
BATCH         IDLE  RUN  DONE  TOTAL  JOB_IDS
ID: 129         -    -    10     10   129.0 ... 129.9 [####################]

[##########################################################################]

Total: 10 jobs; 3 completed, 4 idle, 3 running

Updated at 2026-07-10 16:25:06
Press any key to exit
```

Reviewing Completed Jobs

Log File

```
000 (128.000.000) 2024-08-01 10:05:08 Job submitted from host: <128.104.101.92>
...
001 (128.000.000) 2024-08-01 10:05:46 Job executing on host: <128.104.101.128:9618>
...
006 (128.000.000) 2024-08-01 10:07:54 Image size of job updated: 220
        1  -  MemoryUsage of job (MB)
        220  -  ResidentSetSize of job (KB)
...
005 (128.000.000) 2024-08-01 10:12:48 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
```

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
```

Questions?

SUPPLEMENTARY SLIDES

shell vs executable+arguments

```
shell = ./least_squares.py
    gapminder-life-expectancy.csv
    Brazil

transfer_input_files =
    least_squares.py, gapminder-
    life-expectancy.csv
```

- Allows you to write command **exactly** as you run it.
- Must transfer in executable script.

```
executable = least_squares.py

arguments = gapminder-life-
    expectancy.csv Brazil

transfer_input_files = gapminder-
    life-expectancy.csv
```

- Breaks command into two pieces.
- Executable script is **automatically transferred.**

Both are valid! Use what you prefer!

My job went on “hold”!

- The *hold (H)* state is HTCondor’s way of telling you that it thinks something’s wrong with the job and that ***you should review it!***
- Lots of things cause holds:
  - Unable to find an input file because of **typos**
  - Job going over **resource allocations** (cpu, memory, disk)
  - Occasional **network issues** affecting file transfer
  - Jobs restarting too many times

My job went on “hold”!

When this happens:

- Run `condor_q -hold`
- Read the message carefully and try to figure out the issue!
- Fix the issue, then `condor_release` or `condor_rm` then resubmit the job!

You can always ask for help in understanding the hold message!