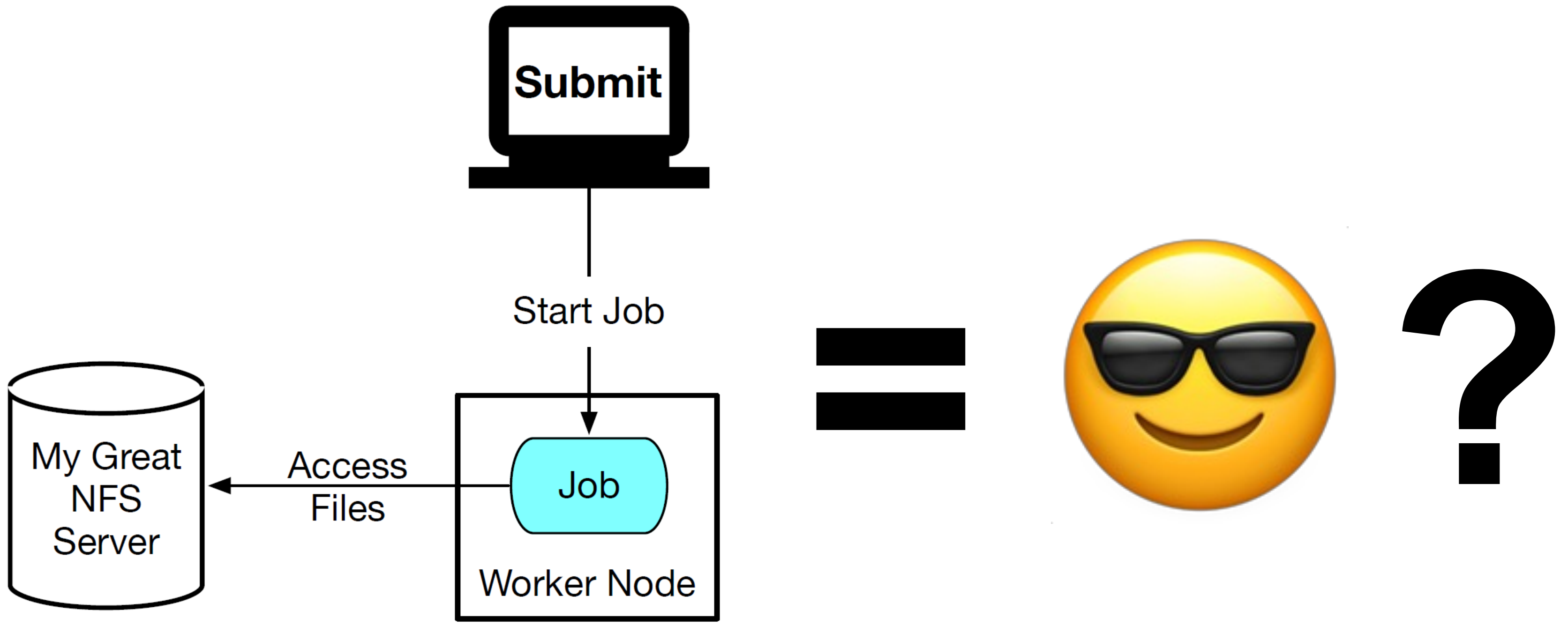
Getting Your Data with HTCondor

Brian Bockelman

European HTCondor Workshop 2023

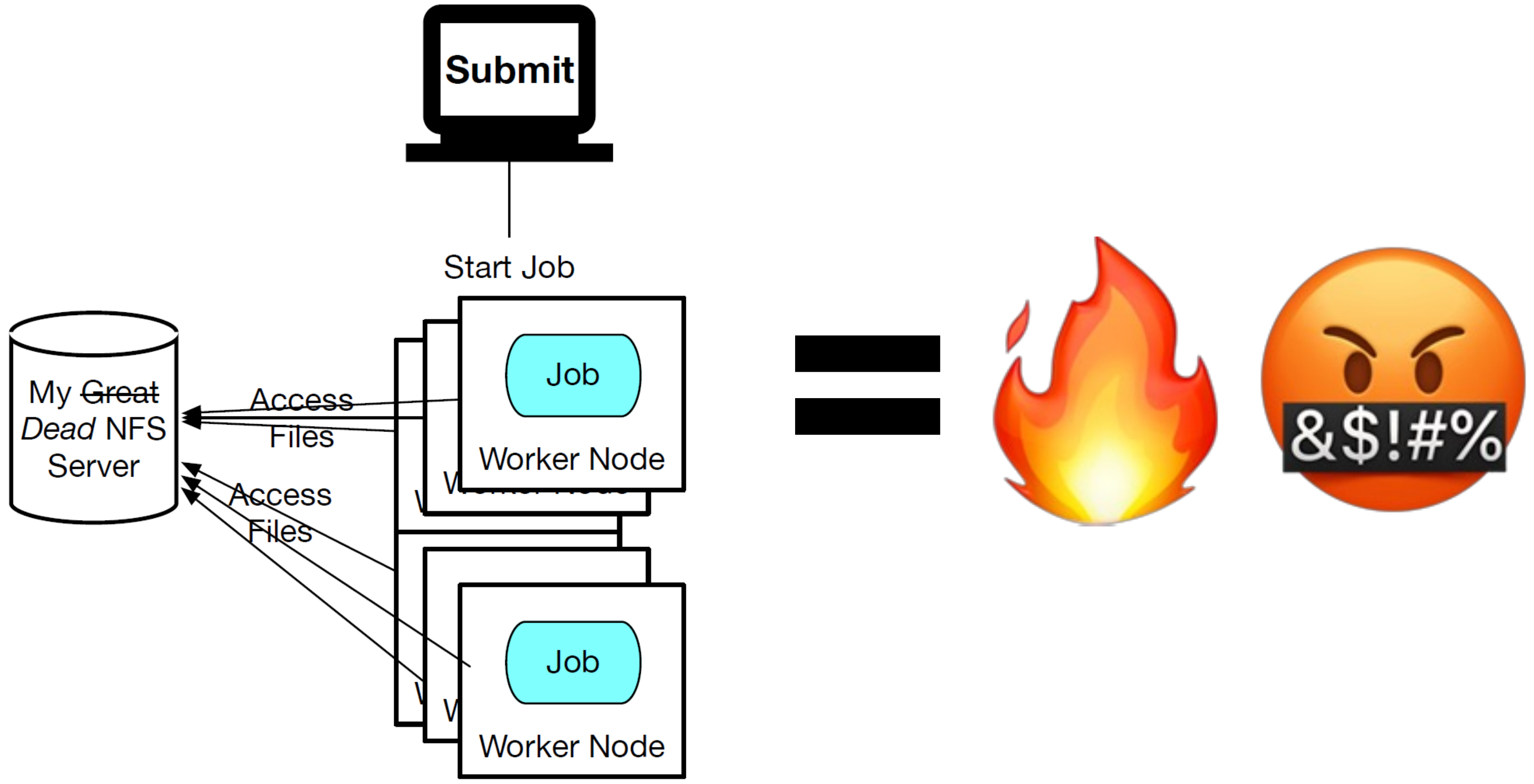
Why “move” my data?

Why is this even a talk?



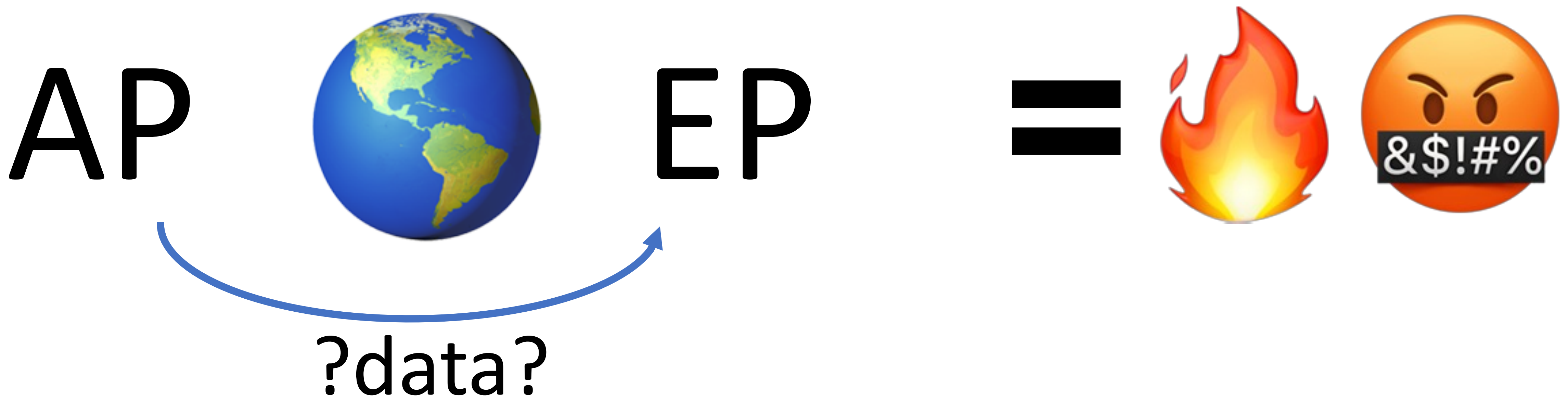
Why "move" my data?

This is why …



Why “move” my data?

And this is why …



HTCSS Does Not Simply “Move” your files…

For HTCondor to deliver throughput, it must **manage capacity**.

- While we tend to think “CPU, GPU, and memory”, not managing “I/O and storage” can lead to equally large messes!
- I/O resources are perhaps the most important ones to manage!
- HTCondor can limit I/O activity based on concurrency level (and concurrency limits can be adjusted up/down based on I/O activity).

But first, you must tell us what you need…

Tell us what you’re doing

It’s good for you, we promise!

HTCondor Submit Files

By declaring your jobs' inputs and outputs to HTCondor, you:

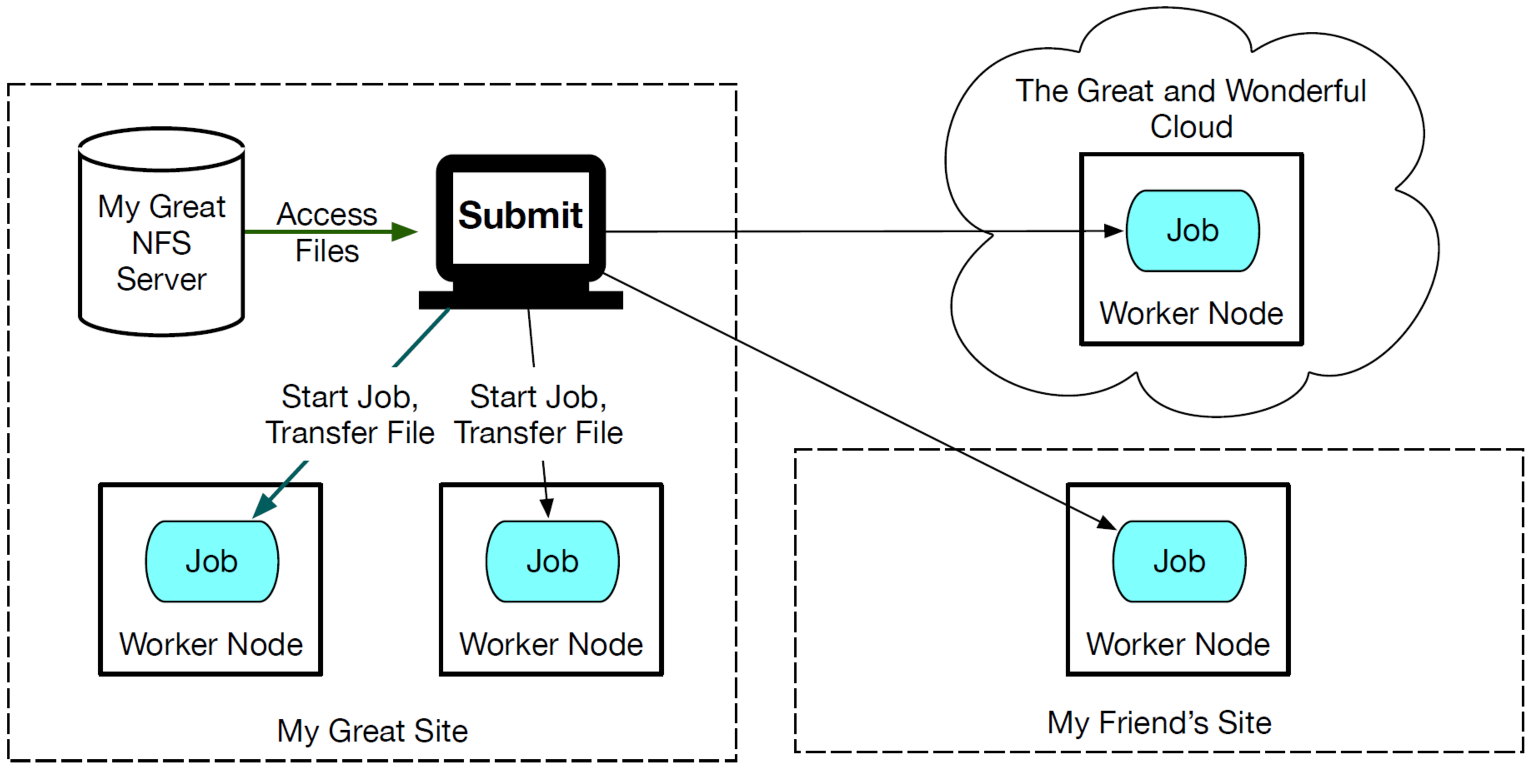
- Allow HTCondor to manage the movement of files.
- Allow HTCondor to prepare your job environment.
- HTCondor knows to not even start your job if the input is unavailable.
- Can make your job portable to other infrastructures.

In the simplest - and most common - case, HTCondor will also perform the file transfer.

```
universe = vanilla
executable = science.exe
arguments = $(Process)
transfer_input_file = \
              input.txt
output = science.out
error = science.err
log = science.log
queue
```

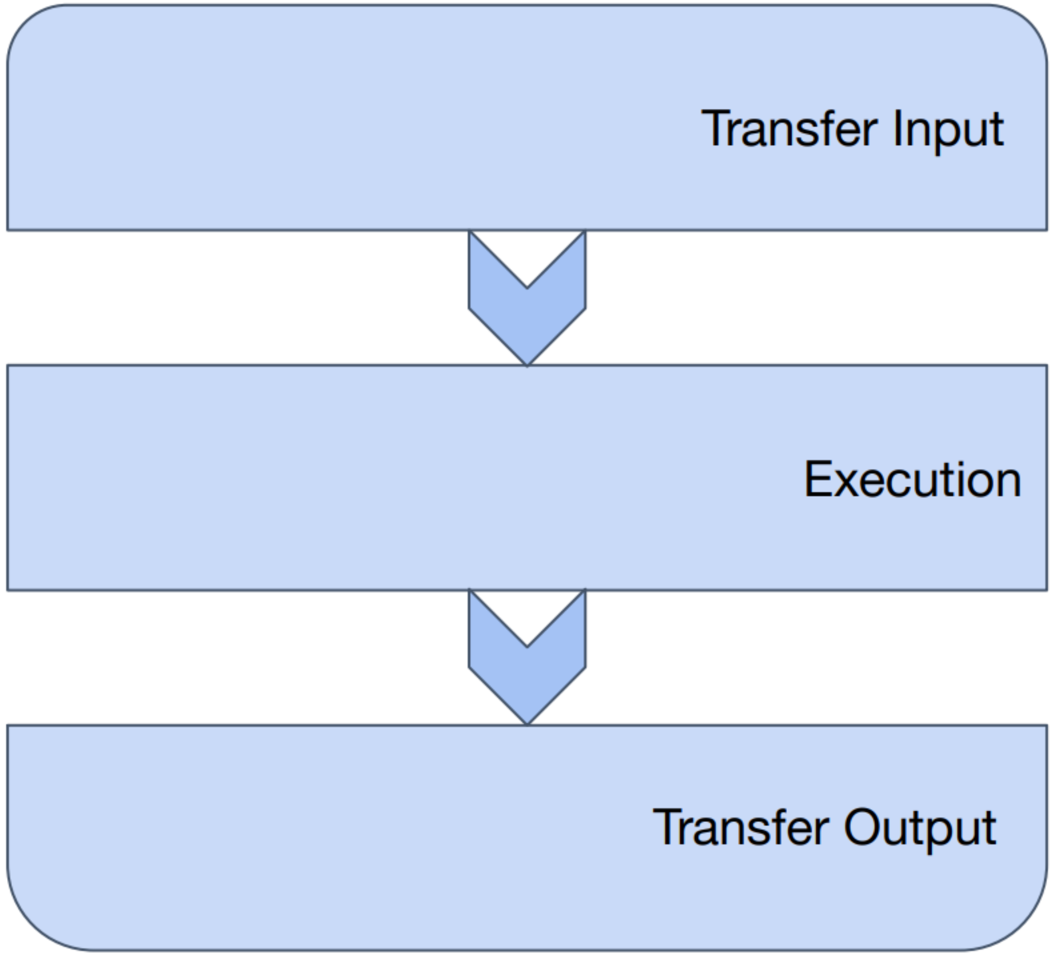
If HTCondor knows the job's I/O requirements…

… it can take you places!



Ever tried exporting NFS offsite? 🤬

HTCondor breaks the job into stages



transfer_input_files = input1.dat, input2.dat

executable = my_processor.py

transfer_output_files = complete.dat

If HTCondor manages the I/O, it can delineate these stages

Execute policy on failures!

- If everything is being done in the middle of the job, all failures look like job failures
  - Couldn't read input data? Job terminates early, go dig through the logfiles to figure out why…
  - Program crash? Job terminates early, go dig through the logfiles to figure out why…
  - Couldn't write output data? Job terminates without output data, go dig through the logfiles to figure out why…
- If HTCondor manages the job, it can <u>execute policy</u>.
  - “If I mistyped the input file name, hold the job”
  - “If the HTTP server returns ‘503 Service Unavailable’, try again later”

All about CEDAR

CEDAR is the built-in protocol HTCondor uses for

CEDAR Transfers

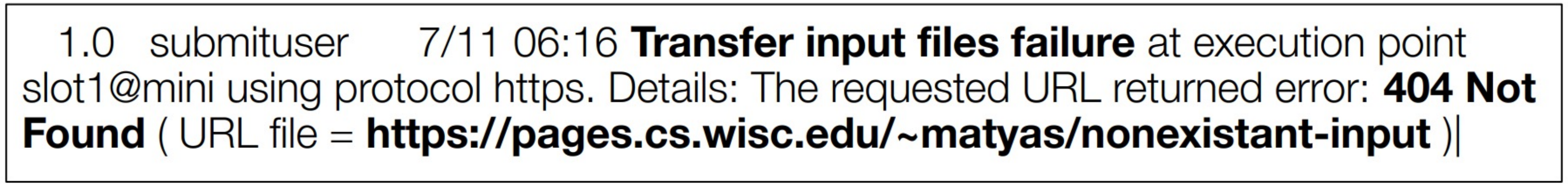
- CEDAR is HTCondor’s internal binary protocol for transferring files.
- Uses the TCP connection established between client and server:
  - Can use HTCondor’s connection broker to reverse connections if server is behind a firewall.
  - Can only read/write local files.
  - Minimal use of round-trip blocking during transfers: can effectively move a directory of small files.
  - No caching (usually).
  - “Plays well” with firewalls.
- Effective, simple, no setup required: the baseline for users.

CEDAR Transfer Queueing

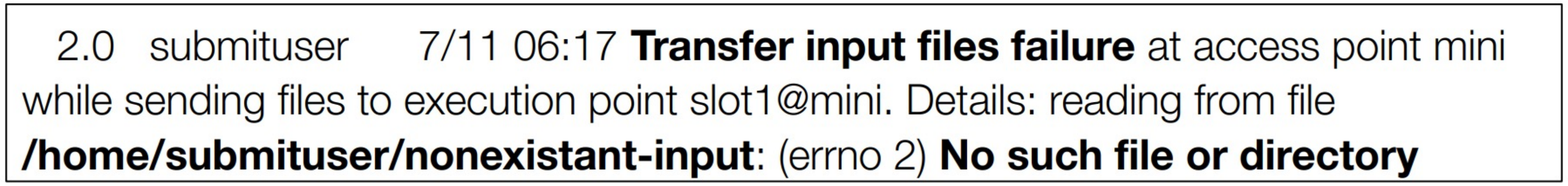
- Before any transfer starts, the source side enters a transfer queue at the AP.
  - This allows the AP to understand the concurrency of currently-running transfers.
- The queue entered defaults to the owner’s HTCondor identity.
  - When it is ready to start a new transfer, the AP will round-robin between queues.
- The AP records the time spent in I/O and adjusts the concurrency based on a high-/low-watermark algorithm: number of transfers is slowly increased until a high-water load limit is reached. Then, the concurrency is decreased until the low-water limit is reached.

Working on error messages

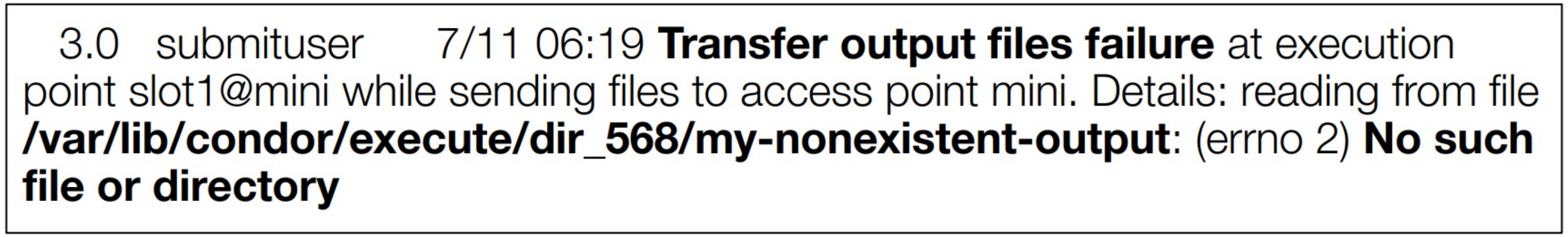
During the course of HTCondor 10.x, we've tried to cleanup CEDAR errors messages:



```
  1.0   submituser      7/11 06:16 Transfer input files failure at execution point
slot1@mini using protocol https. Details: The requested URL returned error: 404 Not
Found ( URL file = https://pages.cs.wisc.edu/~matyas/nonexistant-input )
```



```
  2.0   submituser      7/11 06:17 Transfer input files failure at access point mini
while sending files to execution point slot1@mini. Details: reading from file
/home/submituser/nonexistant-input: (errno 2) No such file or directory
```



```
  3.0   submituser      7/11 06:19 Transfer output files failure at execution
point slot1@mini while sending files to access point mini. Details: reading from file
/var/lib/condor/execute/dir_568/my-nonexistent-output: (errno 2) No such
file or directory
```

Arcane Knowledge…

For Users:

- If you use the **-spool** option, HTCondor will make a copy of your input files to a private directory. This allows you to make changes locally while your jobs are running.
- The **stream_output** submit file command will cause HTCondor to stream output back to the submit host while the job is running. Useful - but use sparingly (consider condor_tail or condor_ssh_to_job as well).
- **max_transfer_output_mb** allows you to put a maximum cap on the data you transfer back; a useful sanity check if your job produced 100GB when you expected 100KB.
- **encrypt_input_files** allows you to force some files to be encrypted in flight - even if HTCondor would not otherwise do this.
- The **transfer_output_remaps** command allows you to provide arbitrary mappings from files in the job execute directory

For Admins:

- **MAX_CONCURRENT_UPLOADS** / **MAX_CONCURRENT_DOWNLOADS** provide an absolute limit on the number of files being transferred at a time
- **FILE_TRANSFER_DISK_LOAD_THROTTLE** will further lower the number of concurrent file transfers based on the I/O load measured on the submit host's storage.
- **MAX_TRANSFER_OUTPUT_MB** sets the schedd-wide default for maximum data transfers per jobs (users can override).
- **MAX_TRANSFER_QUEUE_AGE** is the maximum time, in seconds, that a transfer is allowed to proceed before it is killed.

Coming up soon…

Integration with the LotMan library (part of the Pelican Platform):

- LotMan performs accounting for storage.
    - Finally (!) can ask HTCSS questions like “how much spool is Brian using?”
- Can ask jobs to provide estimates of input/output sandbox needs.
    - Don’t schedule a job with 1TB output if the user doesn’t have 1TB of space allocated!
- First step in the ability to set policies for storage.
    - As with I/O, users should not get a blank check for storage.

Delegated Transfers

Bringing your friends along for the ride…

AKA, “URL-Based Transfers”

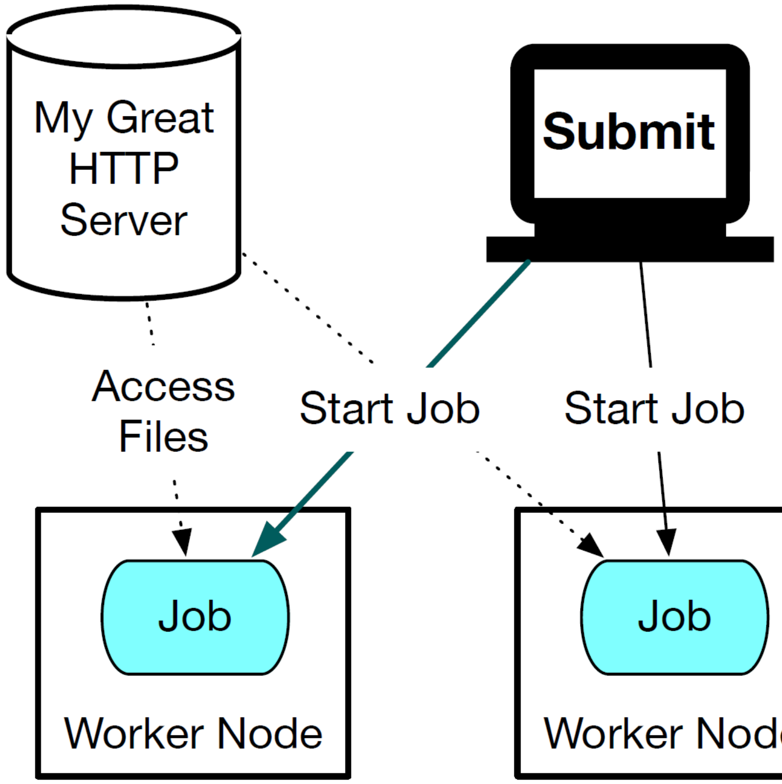
What are Delegated Transfers?

- Delegated transfers* are transfers that are initiated by HTCSS but performed by some other component.
  - We typically call these “URL-based transfers” but I feel the fact they’re specified by URL secondary.
- There’s an enormous world of transfer tools and protocols out there. Delegated transfers are how HTCSS taps into that for the input/output sandbox.
- Shipped with HTCSS:
  - HTTP, FTP, HTTPS, DAVS, file (basically, anything that libcurl supports!)
  - data:// - base64-encode the data in the URL itself.
  - osdf:// (and soon, pelican:// !) – transfer with a data federation.

* Still searching for a good name here – suggestions welcome!

Delegated Transfers

```
universe = vanilla
executable = science.exe
arguments = $(Process)
transfer_input_file = \
 https://example.co/input
output = science.out
error = science.err
log = science.log
queue
```



I can do that!

- Wait, why not call `curl` inside my job? I can do that!...
- As we say at CHTC, **Miron has a lot of questions**:
  - Are you sure you call curl correctly?
  - Did you pass the right headers to make caching work?
  - Did you discover the right proxy?
  - Did you set timeouts appropriately?
  - Did you fine-tune your retry policy?
  - When the transfer fails, is this reflected correctly in the job status?
- If HTCondor doesn’t know about it, HTCondor can’t schedule it!
- Same as with normal file transfers, HTCondor can do the hard work and (difficult) management if it is told what URLs are needed.

Your own delegated transfers!

- The world is a lot larger than the supported mechanisms that ship with HTCondor.
- *Don't see your preferred schema? You can write your own plugin…*
  - "You" applies to both users and EP admins.
- The plugin must:
  - Specify the schemes it supports (gs://, box://, gdrive://, etc).
  - Take an input file describing a list of transfers to perform.
  - (Actually perform the transfers, of course!)
  - Produce an output file describing the results of the transfer.
- HTCSS will group the transfers so the plugin is invoked *once* per URL schema.
  - Optimize to your heart's content

Arcane Knowledge…

- HTCSS now keeps statistics on these transfers. You can see how many bytes were moved, how many files, number of successes.
  - Also the file transfer stage is present in the job's event log.
- CEDAR transfers are done first for the job input sandbox and last for the job output sandbox:
  - You can keep
- The s3:// URLs are special: instead of transferring the S3 credentials to the EP, it will automatically create a signed URL on the AP. This https:// URL is then sent to the EP for transfer.
  - The EP only receives a single URL, not your credentials! Minimizes risk of a malicious EP.

Off into the future – Managing Delegated Transfers

- Originally *all* transfers for a job were done with an active token in the transfer queue.
  - This made no sense: we are not managing the AP's I/O resources while transferring with a 3rd party!
- In 9.x, we changed this so no transfer tokens were held.
  - Which also makes no sense! That means delegated transfers are completely unmanaged.
- Soon-to-appear: the AP manages a separate queue for delegated transfers.
  - Targeting services that AP has a close relationship with.
  - Others (think AWS…) may not need to be managed by the AP.

Plans for Pelican

We want close interactions between the data federation and HTCSS. Examples:

- Informing the AP about the properties of the transfer before it launches the job. How many GB (TB?) will be moved?
  - Avoid executing a transfer where don’t have enough sandbox space.
- Stage large inputs to use Pelican instead of CEDAR transfers.

Parting Thoughts

Parting thoughts

If you could only learn three things from this presentation, what should they be?

1. Always leverage HTCondor's "sandbox" model where you can: this leads to more robust, managed transfers.
2. CEDAR-based transfers are simple and managed – as long as the AP can scale to the job!
3. Delegated transfers enable HTCondor to tap into a wide range of transfer possibilities.

And finally – be on the lookout for more Pelican-led functionality in the near future!

Questions?

This work is supported by NSF under Grant Nos. 2030508. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.