HTC: Dealing with Data

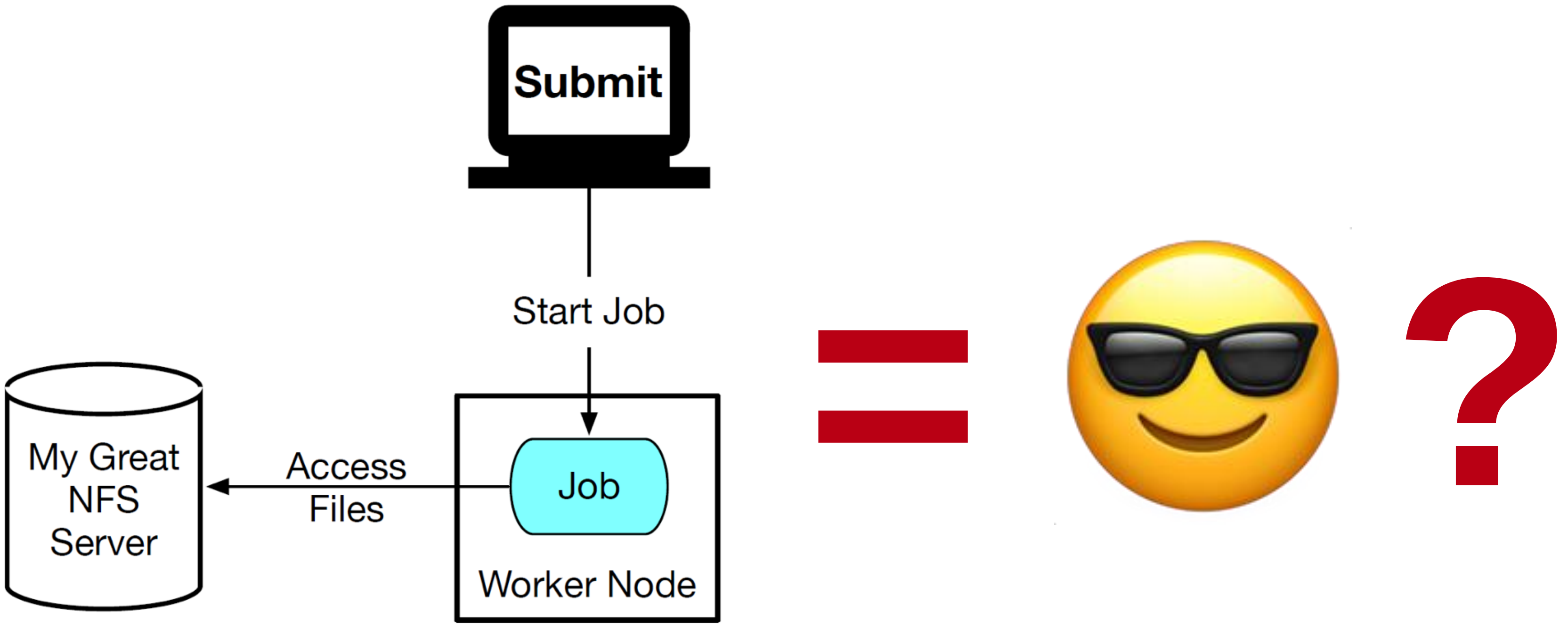
Some friendly advice, some discussion

Brian Bockelman, 25 September 2024

MORGRIDGE
INSTITUTE FOR RESEARCH

Why does HTCSS care about data?

Why is this even a talk?



Submit

Start Job

My Great NFS Server

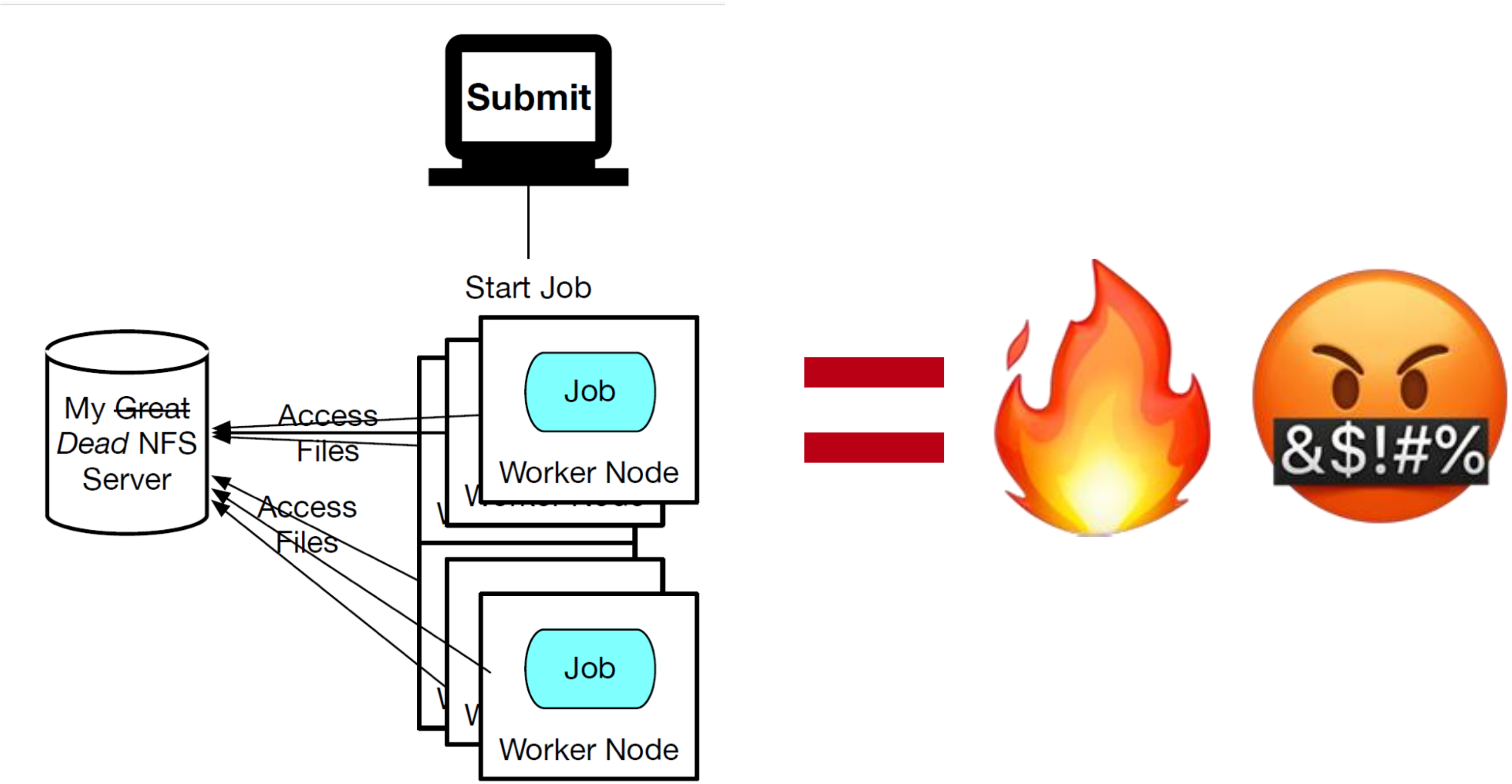
Access Files

Job

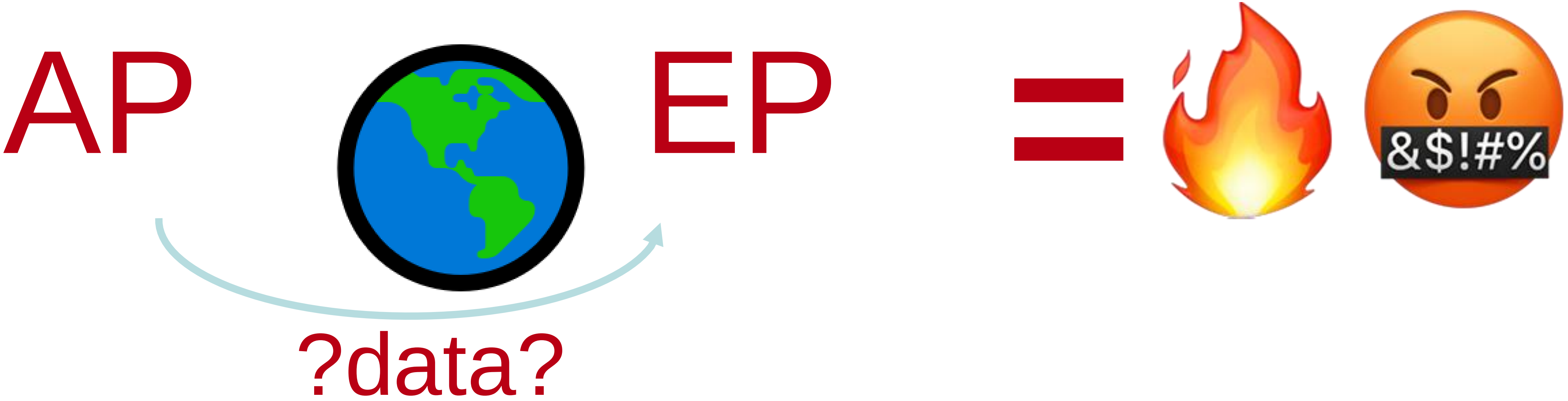
Worker Node

= 😎 ?

Why? This is why…



It only gets worse…



AP 🌎 EP = 🔥🤬

?data?

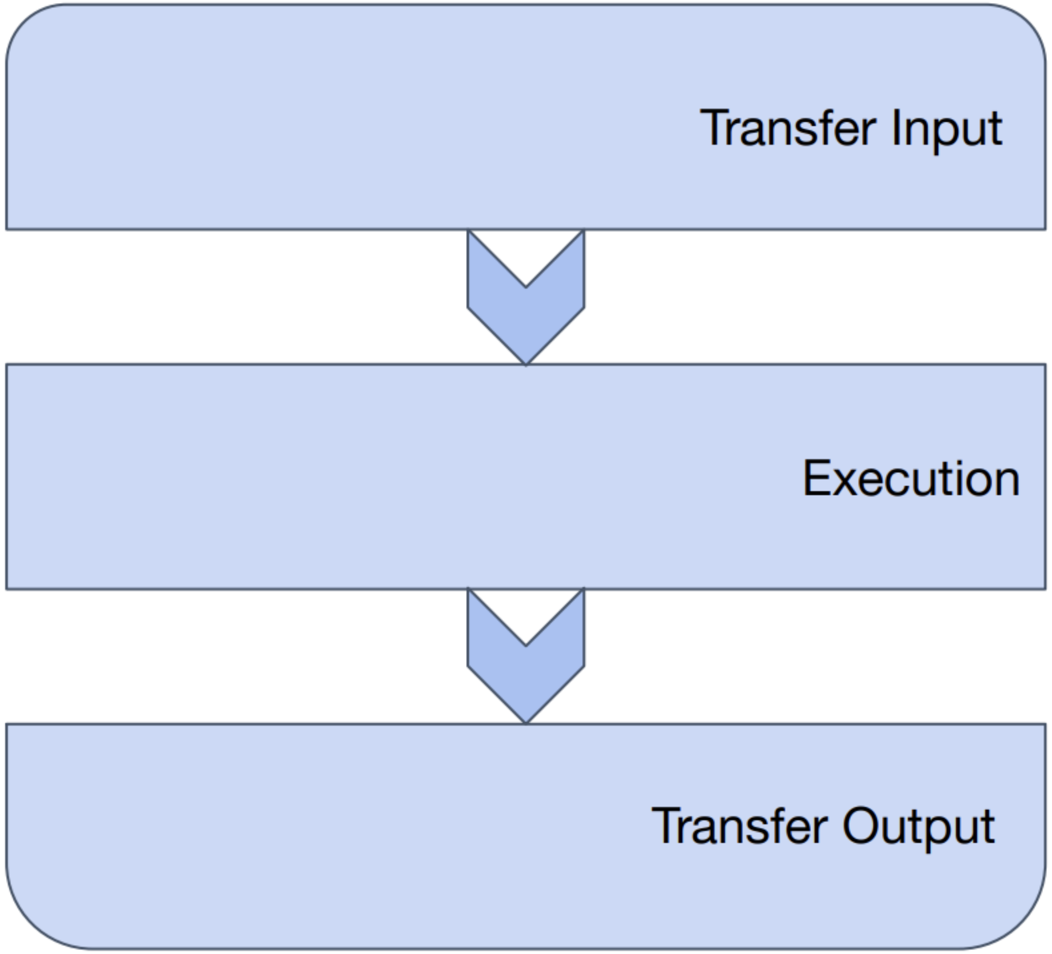
Dealing with Data

- What are the important parts of “dealing with data”?
  - Understanding data dependencies between tasks.
  - Detecting failures & handling retries.
  - Managing finite resources (storage, I/O).
  - And yes, moving data quickly.
- What’s perhaps the least of these? **Data transfer rates**!
- What is a user or site to do?
  - **Power through with hardware**: Pay enough money so you can assume the hardware never fails and is never the bottleneck.
  - **Do it yourself**: Provide all the functionality yourself (or use a workload manager layered on top of HTCSS).
  - Let the HTCondor Software Suite help!

Managing workloads = managing data

- HTCSS needs to manage your data to manage your workloads:
  - **I/O capacity**: Limit the I/O load (MB/s, IOPS) experienced by the remote service. Managed via limiting the transfer concurrency.
    - Can consider read and write activity separately.
  - **Retry policy**: What to do when a transfer fails?
    - Should we consider it permanent or transient?
    - Run the job at a different site?
    - Have the AP start avoiding the EP in general?
  - **Portability**: Run the job at a wider range of resources, not just local to the AP/EP's site.
- Can you skip this? Sure! But then you're in charge…

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!

HTCondor Submit Files

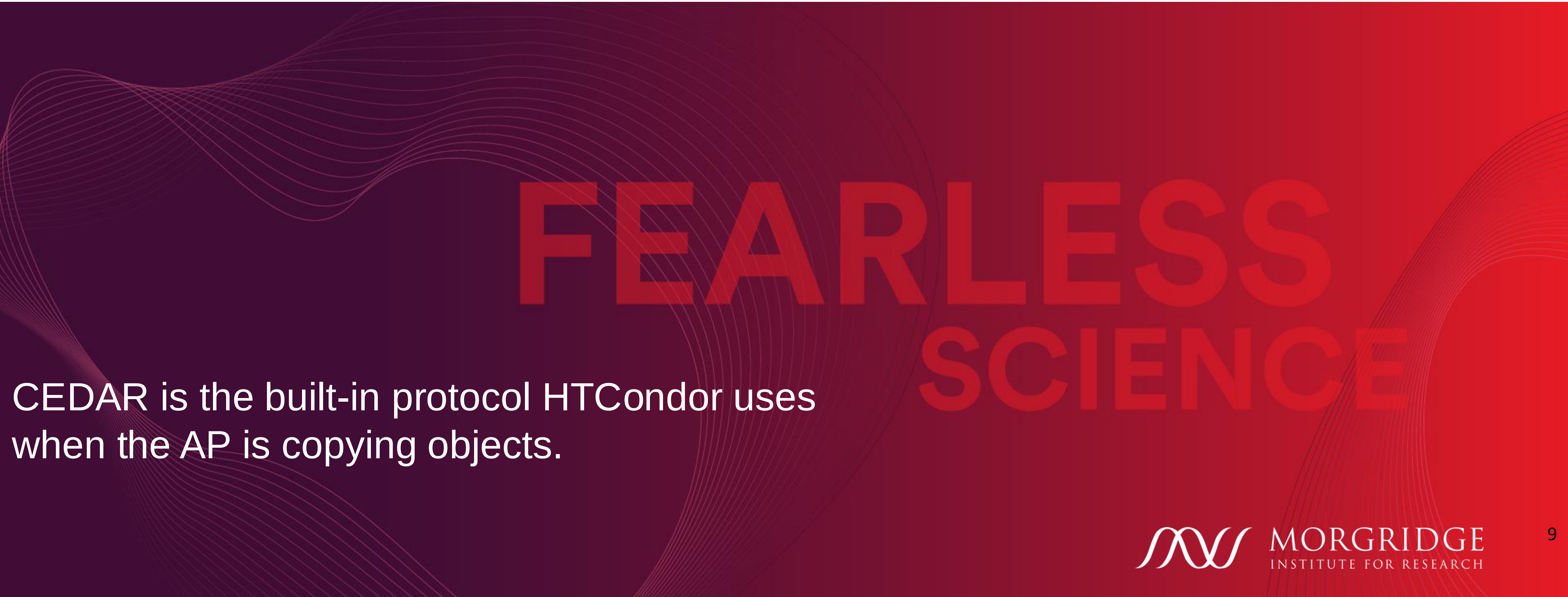
By declaring your jobs' inputs and outputs to HTCondor, you:

- 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
```

All about CEDAR



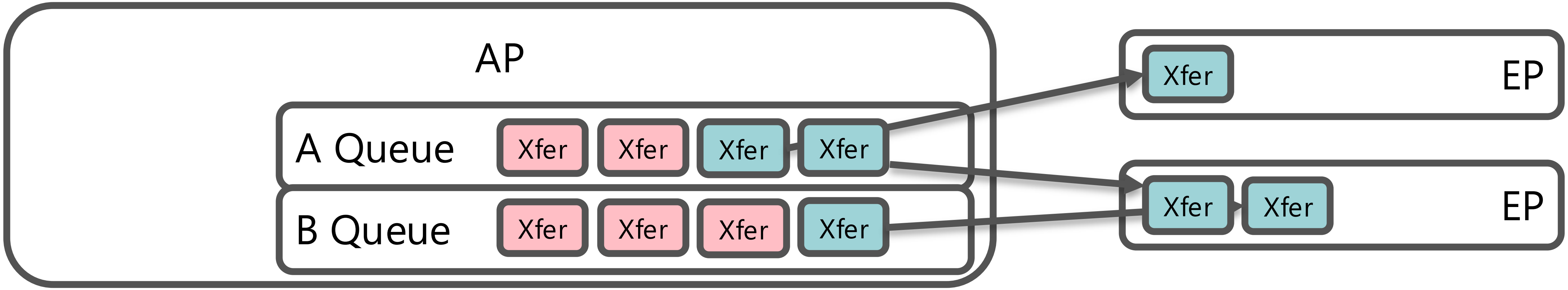
CEDAR is the built-in protocol HTCondor uses when the AP is copying objects.

CEDAR Transfers

- CEDAR is HTCondor’s internal binary protocol for transferring files.
- Uses the TCP connection established between client and server:
  - E.g., between AP and EP.
  - Can use HTCondor’s connection broker to reverse connections if server is behind a firewall.
  - Can only read/write local files to the AP/EP.
  - Effective: minimal use of round-trip blocking during transfers: can move a directory of small files, even with large network latencies.
  - No optimizations around object reuse: no caching if the same file is moved repeatedly.
  - “Plays well” with firewalls.
- Effective, simple, no setup required: the baseline for users.

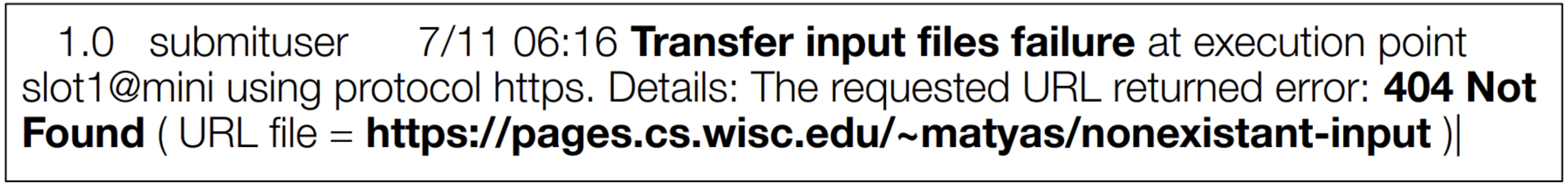
Resource management: CEDAR

- 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.
  - Transfer queue is a ClassAd expression: can be based on the EP's sit ename.
- 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.
  - The concurrency is decreased until the low-water limit is reached.

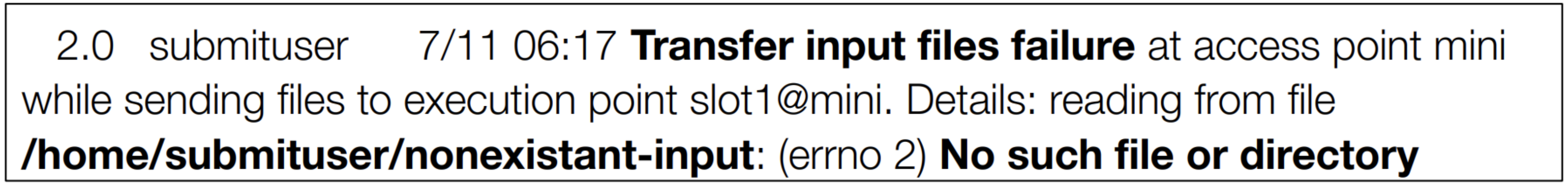


Working on error messages

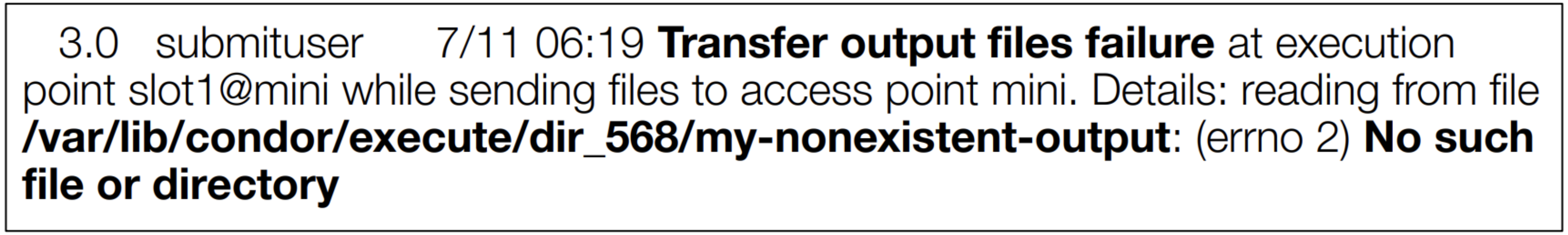
- During the course of HTCondor 23, we’ve tried to cleanup CEDAR errors messages:
  - Fewer repetitions.
  - User-centric, not admin-centric.



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.

Delegated Transfers

AKA, “URL-Based Transfers”

FEARLESS SCIENCE

Bringing your friends along for the ride…

MORGRIDGE
INSTITUTE FOR RESEARCH

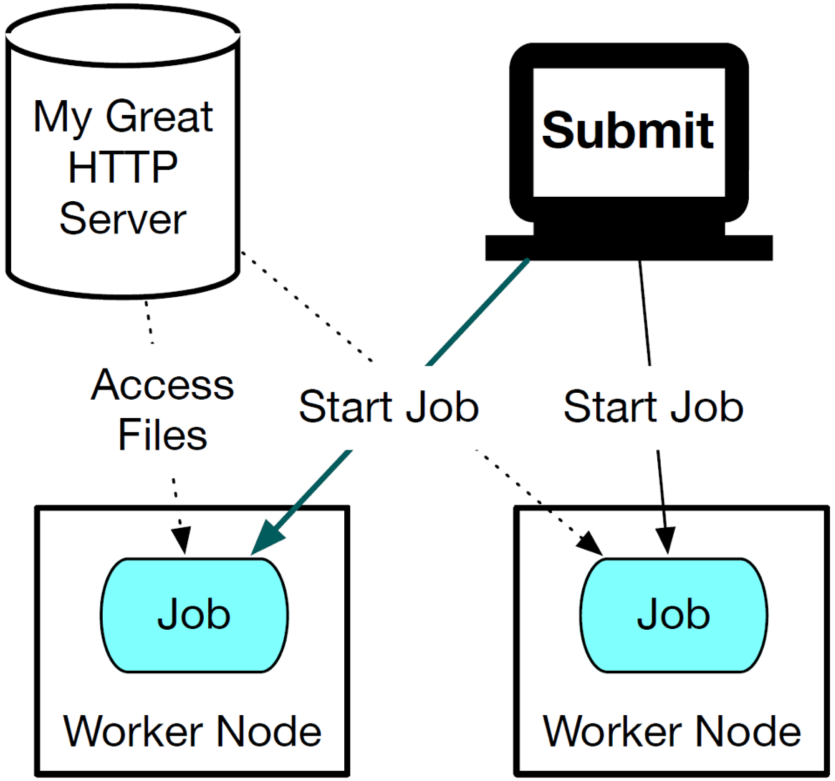
What are Delegated Transfers?

- 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…

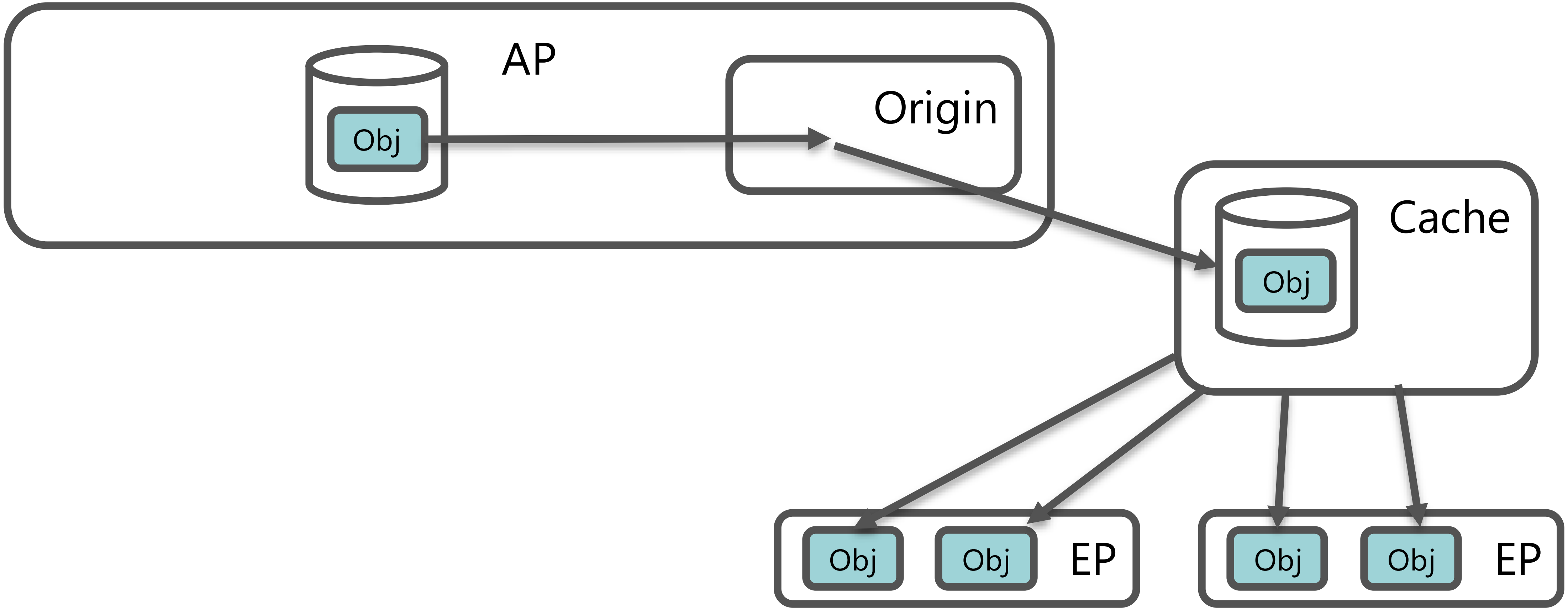
- HTCSS now keeps statistics on these transfers. You can see how many bytes were moved, how many files, number of successes.
  - Also the file transfer stage is present in the job's event log.
- CEDAR transfers are done first for the job input sandbox and last for the job output sandbox:
  - Delegated transfers can rely on configuration sent via CEDAR; they can drop files that are returned via CEDAR.
- 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.

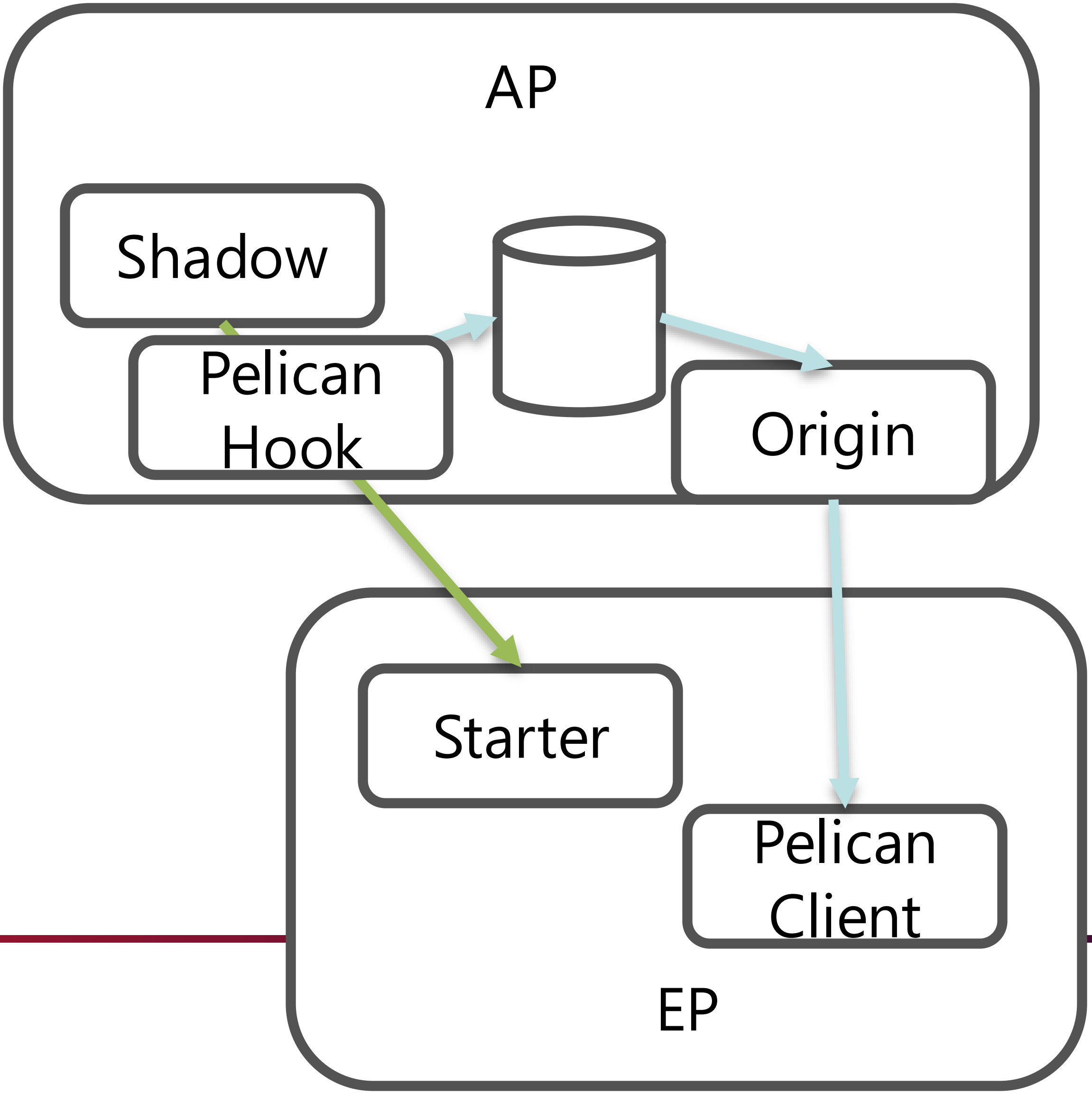
Example - Pelican

The Pelican plugin allows for reuse at the content distribution network. **More later today!**



Arcane Knowledge…

- “Well-hidden” in HTCSS is the shadow job hook.
  - Arbitrary code invoked by the condor_shadow process before sending a job’s ClassAd to the condor_starter.
    - Receives the shadow’s copy of the job ClassAd and any credentials for the job.
    - Output is updates to the ClassAd.
- **Opportunity**: Can transform the input sandbox.
  - Example: The Pelican shadow hook will examine large files, upload them to a local origin, and then instruct the starter to use the Pelican copy of the file.



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.

Common Pitfalls / Challenges

Why do folks manage their own data?

- **Data volume**: The volume of data the job will read is larger than the local scratch space.
- **Streaming / subsets**: The application reads a small subset of the data and CEDAR moves the entire file.
- **Unknown application dependencies**: The user is utilizing a community-developed application and has little insight into what data is needed.
- **Workflow engine assumptions**: The user is running a workflow engine that assumes a shared filesystem.

Common Pitfalls: Approaches

Why do folks manage their own data?

- **Data volume**: The volume of data the job will read is larger than the local scratch space.
  - **Approach**: Consider splitting into smaller jobs, each with less data volume.
  - **Approach**: Still declare dependencies, let HTCSS know the filesystem requirements. Can still use the policy/retry engine!
- **Streaming / subsets**: The application reads a small subset of the data while CEDAR moves the entire file.
  - **Approach**: Move subsetting step into job creation time or an earlier node in DAG.
  - **Approach**: Delegate subsetting into a custom transfer plugin.
- **Unknown application dependencies**: The user is utilizing a community-developed application and has little insight into what data is needed.
  - **Approach**: Provide interactive host, run inside a container.
- **Workflow engine assumptions**: The user is running a workflow engine that assumes a shared filesystem.
  - **Approach**: Search for a “AWS mode”; any HTCondor integration is more like AWS than SLURM.

Parting Thoughts

- Rely on HTCSS to manage your data and you get:
  - **Policy engine**: Clear phases for transfer and ever-improving policies for failure.
  - **Capacity management**: Concurrency limits in the AP and queue management implements management of the I/O capacity of the AP.
  - **Portability**: Jobs are not tied to the local shared filesystem and can moved to anywhere!
- Using HTCSS relies on you building a knowledge of the data dependencies of your workload:
  - This does not come “for free”: new users often struggle with understanding their workload.
  - <u>Worthwhile</u>: good for the hygiene of the workload, opens doors with HTCondor!

Questions?

FEARLESS SCIENCE

This project is supported by the National Science Foundation under Cooperative Agreements OAC-2030508. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

MORGRIDGE
INSTITUTE FOR RESEARCH