A Job's Journey

Greg Thain

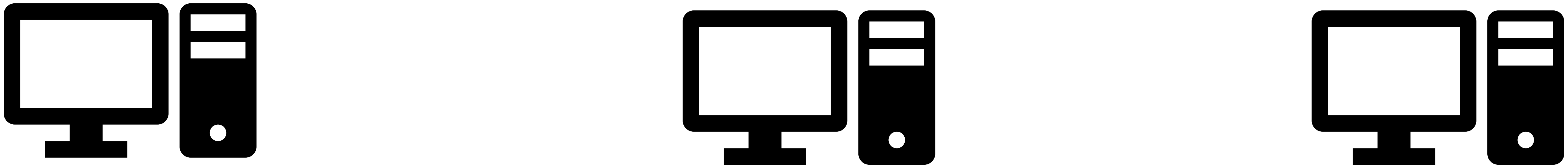
Center for High Throughput Computing

What is a "Job"?

```
$ my_program arg1 arg2 > output_file
```

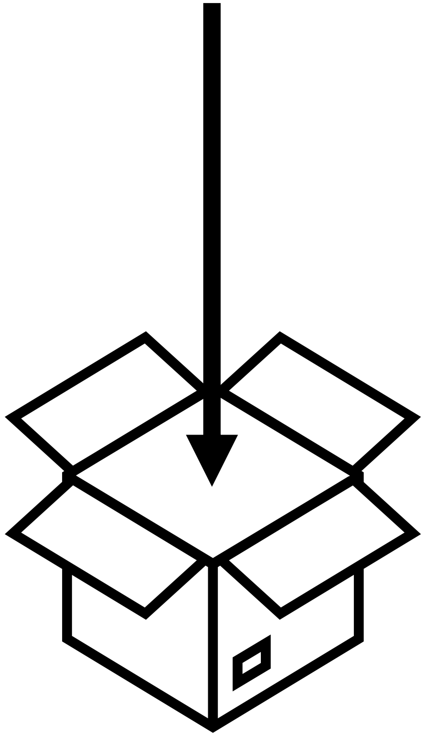

```
$ my_program arg1 arg2 > output_file
```

```
$ my_program arg1 arg2 > output_file
```

We need jobs to be reliable

```
$ my_program arg1 arg2 > output_file
```



Packing Label:

Describes **everything** about the job

What is on the "packing label"?

- Executable
- Arguments
- Environment vars
- Input files
- Output files
- Resources needed
  - (memory, GPU, disk)
- Credentials
- Container Image
- Microarchitecture
- Environment vars
- Retry?
- Policies
- Batch Name
- Cron runtimes
- Etc. etc. etc.

Submit file is packing label

```
Executable = program
Arguments = one two three
…
```

Warning: None of this is verified / checked

Full description of submit commands..

See previous talk…
https://indico.cern.ch/event/1274213/contributions/5571142/

Submit file is job description, not the job!

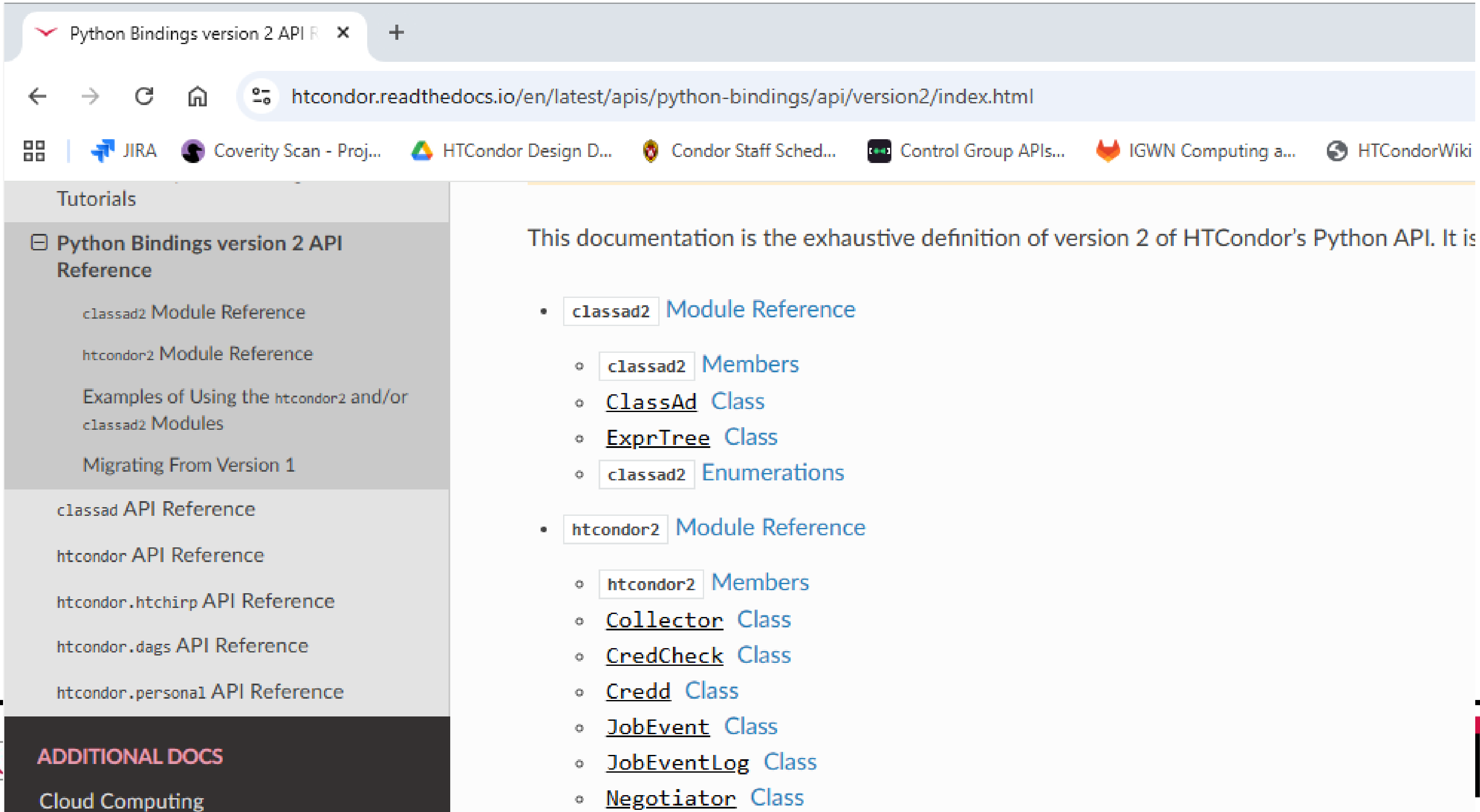
- Two ways to place a job: condor_submit, python
  - Or indirectly via dagman/Pegasus/portal, which call above

condor_submit command line tool

- $ condor_submit
  - Good for interactive use (esp with condor_submit –i)

Consider python bindings for other uses

- Python bindings around job placement better for portals, APIs



Submit file is job description, not the job!

- One the job is placed in schedd, it truly exists
- It exists because the AP gives it a name
  - Really a number cluster.proc
  - You can give it a BATCH_NAME, but the id is schedd-assigned
- When the job exists, it has (is?) a job classad

- Except in one case…

Late materialization

```
Executable = program
Arguments = one two three
max_idle = 100
…
```

- AP is **responsible** for creating the job as needed
- Not on by default today, but soon (?)
- Vital for placing large clusters (say > 10,000)

How do you know a job exists?

- Entry in job event log
- Entry in global event log

```
035 (7.-01.000) 2025-01-01 15:44:31 Cluster submitted from host: <1.2.3.4:38087>
...
000 (7.000.000) 2025-01-01 15:44:31 Job submitted from host: <1.2.3.4:38087>
```

```
000 (6.000.000) 2025-01-01 13:45:12 Job submitted from host: <1.2.3.4:38087>
...
```

- $ htcondor job status <job id>
- $ condor_q job_id

Once job exists, AP finds matching slot

- (Except for Grid/Local/Scheduler universes)

A word about states…

Idle

- Job is created in the "idle" state
  (unless hold = true in submit file)
- We don't really have enough states
  We'd like to add more, but that is … difficult
- So, the states are overloaded more than we would like
- Don't read too much into the state diagrams we will show later

Once job exists, AP finds matching slot

- AP tries to (re) use an unused match it already has
- Or may ask negotiator for a new match
- Or may be able to get a match from a directly connected EP

- If it doesn't match, condor_q –better

Even when matched, may not start right away

- Job start delay queue
- Transfer queue

JOB_START_DELAY and surprises

Idle??

- JOB_START_DELAY = 2
  JOB_START_COUNT = 10

- During the delay, condor_q will say job is "R"unning
- Condor_status will be claimed/idle

- Be careful when looking at "states"

Surprises with transfer queueing

Run??

- Jobs with file transfer go through a queue
- MAX_CONCURRENT_UPLOADS
- MAX_CONCURRENT_DOWNLOADS
- FILE_TRANSFER_DISK_LOAD_THROTTLE

- "Small" transfers skip the queue
- By default, plugin transfer also skip queue

Surprises with transfer queueing

Run??

- When in the queue, waiting for go-ahead
- $ condor_q

```
1.0 gthain 1/1 09:31 0+00:00:02 < 0      run.sh
```

- Where '<' means input, and has been in the queue for two seconds

Finally! Actually running the job

Run!

- After file xfer, file xfer queueing, job finally runs

- UNLESS docker universe, and docker needs to pull the image

Unless, the activation fails…

- If a typo in list of input files
- If permission error in input file

- If EP has changed before the AP starts job…

- Then activation fails…

How do you know a job is running?

Run!

- Entry in job event log
- Entry in global event log

```
001 (2.000) 2025-09-10 15:44:33 Job executing on host:
<1.2.3.4>
        SlotName: slot1@machine
        CondorScratchDir = "/execute"
        Cpus = 1
        Disk = 1500
        GPUs = 0
        Memory = 3981
```

- $ htcondor job status <job id>
- $ condor_q job_id (may be lying)

Another way to learn a job is runni

Run!

- condor_ssh_to_job (-retry)

Things that can happen when runni

Run!

HOLD state

Run!

Explicitly via condor_hold

Via policy via periodic_hold

or via system_periodic_hold

Or by condor if **bad things** happen

e.g. input file doesn't exit

memory exceeded

SUSPEND state

Run!

Explicitly via condor_suspend

via WANT_SUSPEND in startd

Released explicit via condor_continue

User checkpointing and the run state

```
checkpoint_exit_code = 85
```

When job exits with this code:

- HTCondor sends the output/checkpoint files back to AP
- When transfer is complete, restarts the job from scratch
- Job stays in the "R"unning state the whole time

Eviction / Preemption

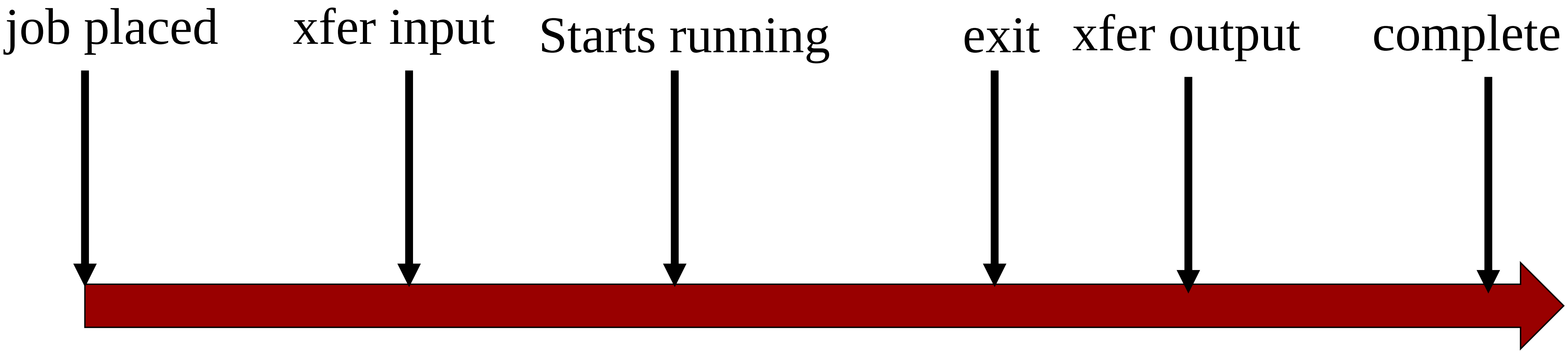
A job can go back to "I"dle

- condor_vacate / condor_vacate_job
- periodic_vacate is true (in newer htcondor versions)
- PREEMPT is true in the startd (and time > MaxJobRetirementTime)
- Startd machine crashes
- Network is broken

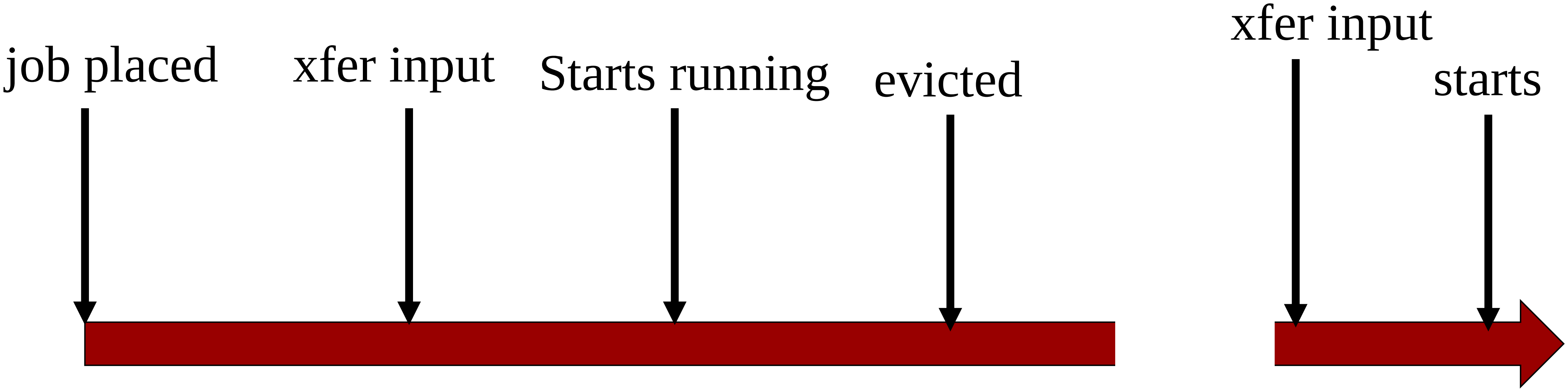
A job can never get out of "I"dle

- When activation fails early
- Nothing in the event log
- NumShadowStarts is incremented

"Epochs"


job placed
xfer input
Starts running
exit
xfer output
complete

"Epochs"


job placed
xfer input
Starts running
evicted
xfer input
starts

"Epochs"

epoch epoch epoch epoch epoch epoch

Viewing epochs

- condor_history –epoch
- Condor_adstash -epoch

End of life of a job

- When exits of it's own accord (by default …)
  - "There's a knob for that…"
  - At the end of the last epoch..
- When condor_rm's by a human
- When Periodic_Remove is true (or SYSTEM…)

- After than, the job is HISTORY
  - - copy of job ad goes to AP's history
  - - also a copy goes to the startd's history
- Both can be read by condor_adstash -> opensearch

Thank you!

This work is supported by NSF under Cooperative Agreement OAC-2030508 as part of the PATh Project. 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 NSF

PATh PARTNERSHIP to ADVANCE THROUGHPUT COMPUTING