Using the NAF

A short introduction for beginner

Beyer Christoph
Hamburg, 6-11-2025

HELMHOLTZ

DESY.

An Introduction to Using HTCondor

Christina Koch

HTCondor Workshop Autumn 2020

September 21, 2020

What is HTCondor?

- Software that schedules and runs computing tasks on computers

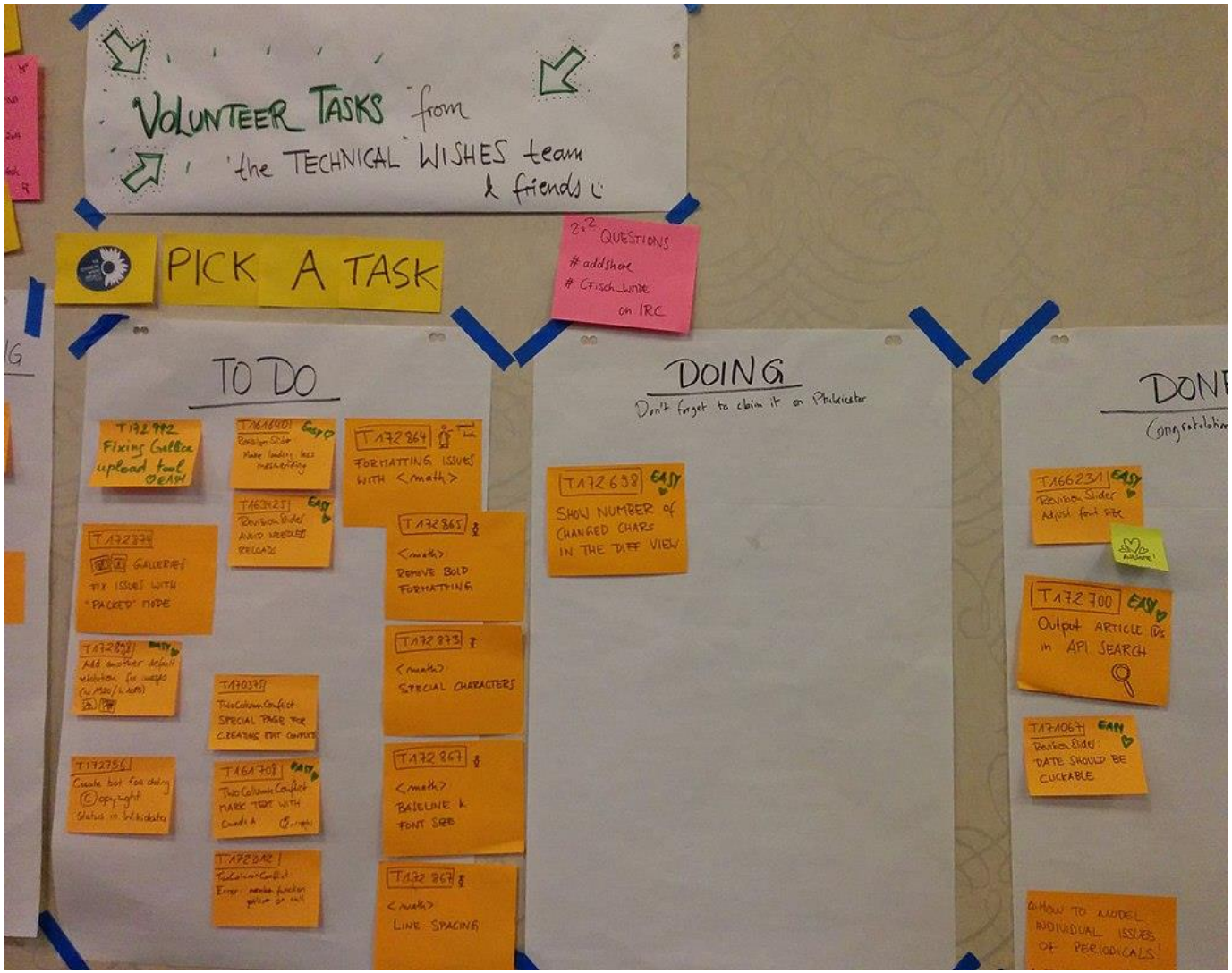
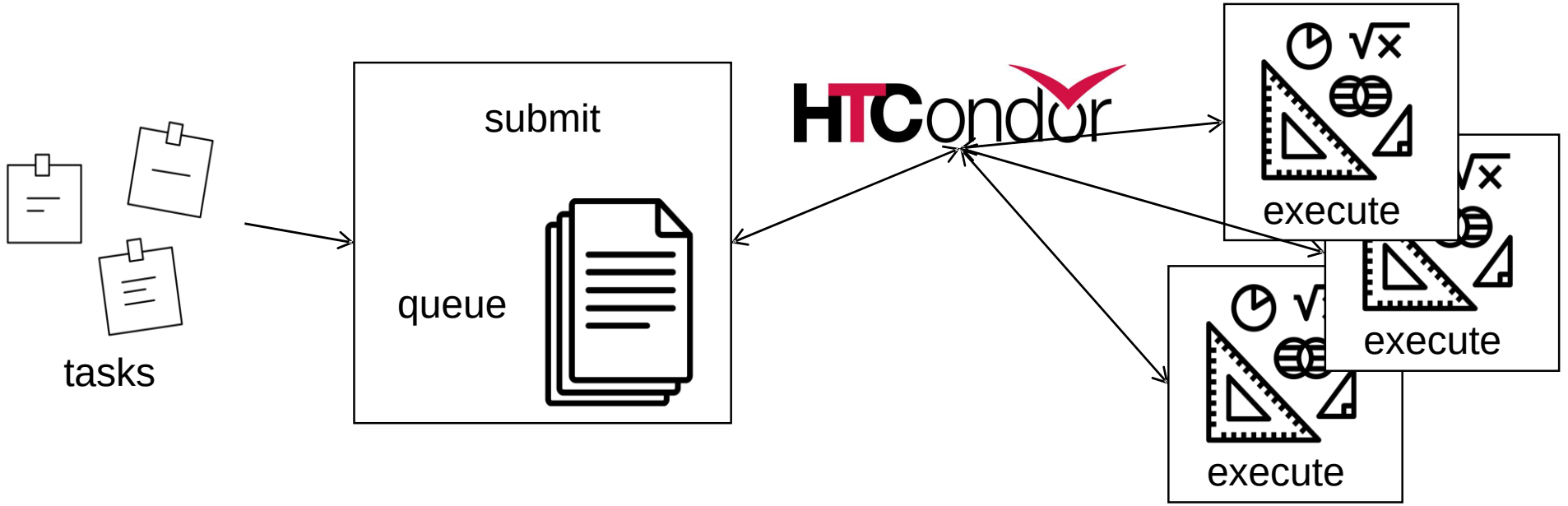


Photo by Johanna Strodt on WikiMedia, CC-BY-SA

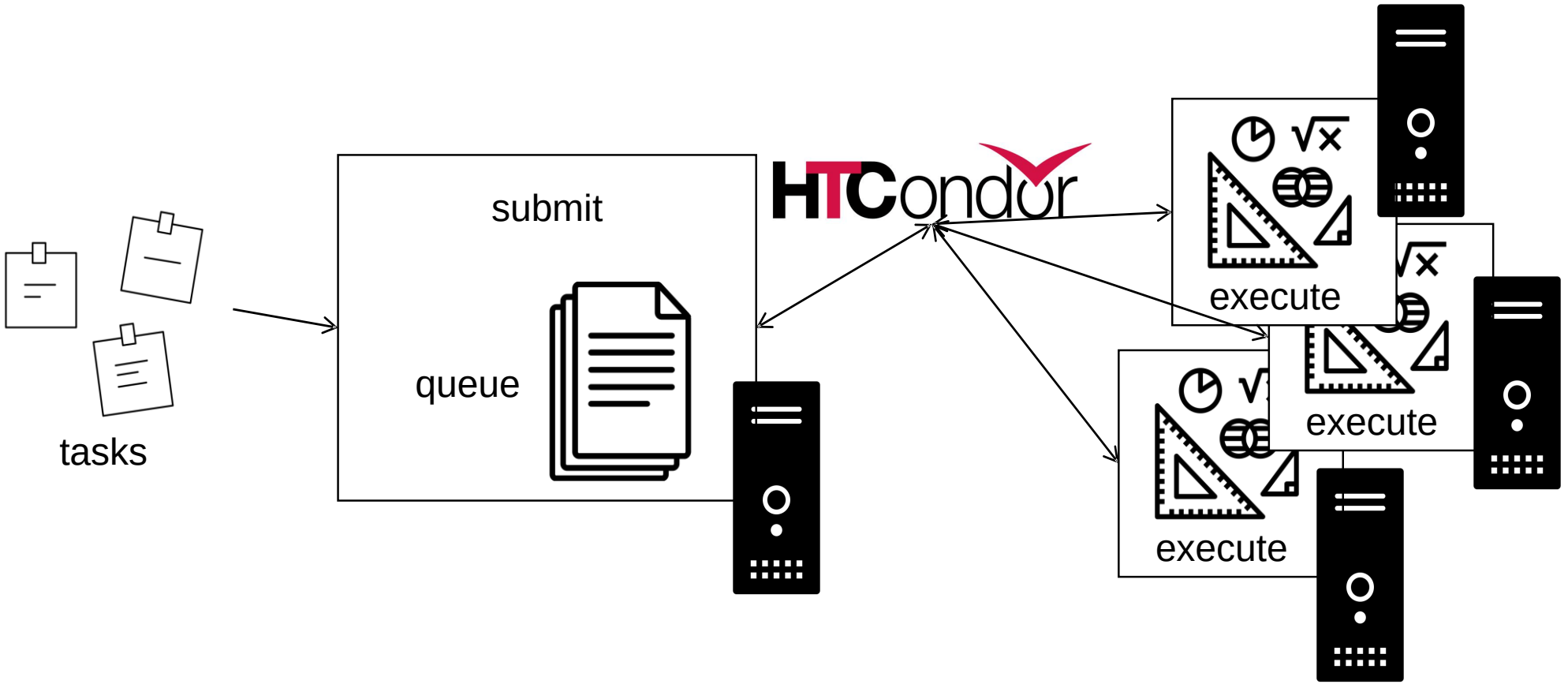
How It Works

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



Post It by Arianna Galimberti; papers by jabbar; Math by Turkkub from the Noun Project

HTCondor on Many Computers



computer tower by Ralf Schmitzer from the Noun Project

Why HTCondor?

- HTCondor manages and runs work on your behalf.
- Manage shared resources among users:
  - Schedule tasks on a single computer to manage computer capacity.
  - Schedule tasks on a group* of computers (which may/may n be directly accessible to the user).
  - Schedule tasks submitted by multiple users on one or more computers.

*in HTCondor-speak, a “pool”

HTCondor Jobs

- HTCondor schedules and executes Jobs
- What is a Job?
  - A ‘Job’ is a single computational task
    1. Input Data
    2. Executable (program)
    3. Output Data
- Executable must be runnable from the command line without any interactive input

Sources of ‘pizza ingredients’ in the NAF

Shared filesystems are part of the design approach

- The login node (called WGS) is the place where you login (usually through SSH) in order to submit and control your jobs
- The WGS is named as your VO e.g. naf-cms.desy.de and the actual, physical server is chosen in a round-robin mechanism
- The WGS is setup equally to the workernode that later will execute your program, including all the fileservice mounts, anything that runs ‘here’ and is accessible will run and be accessible ‘there’ (on the target worker node) without further thinking or action !
- Executables and data live in the shared FS, smaller executable can be transferred with the job for strategical reasons
- No batchsystem will create directories on your behalf, if you rely on a certain dir-structure e.g. for your output you need to create the structure in forehand

Describing an HTCondor Job

```
executable = sleep_runtime.sh
Arguments = 600




output  = $(Cluster).$(Process).out
error   = $(Cluster).$(Process).err
log     = $(Cluster).$(Process).log


request_cpus = 1
request_disk = 50MB
request_memory = 20MB

queue 1
```

CAT-Analysis.sub

Describe an HTCondor job with the Job Description Language (JDL)

Describing an HTCondor Job

```
executable = sleep_runtime.sh
arguments = 600




output  = $(Cluster).$(Process).out
error   = $(Cluster).$(Process).err
log     = $(Cluster).$(Process).log


request_cpus = 1
request_disk = 50MB
request_memory = 20MB

queue 1
```

CAT-Analysis.sub

List the executable and arguments for HTCondor to run at the EP like how you would run the executable by hand

```
$ sleep_runtime.sh 600
```

Describing an HTCondor Job

```
executable = sleep_runtime.sh
arguments = 600

output  = $(Cluster).$(Process).out
error   = $(Cluster).$(Process).err
log     = $(Cluster).$(Process).log

request_cpus = 1
request_disk = 50MB
request_memory = 20MB

queue 1
```

CAT-Analysis.sub

- **log -** Generated by HTCondor to track job progress from key events in the jobs lifetime
- **output -** File that captures the jobs STDOUT
- **error -** File that captures the jobs STDERR

Describing an HTCondor Job

```
executable = sleep_runtime.sh
Arguments = 600

output  = $(Cluster).$(Process).out
error   = $(Cluster).$(Process).err
log     = $(Cluster).$(Process).log

request_cpus = 1
request_disk = 50MB
request_memory = 20MB

queue 1
```

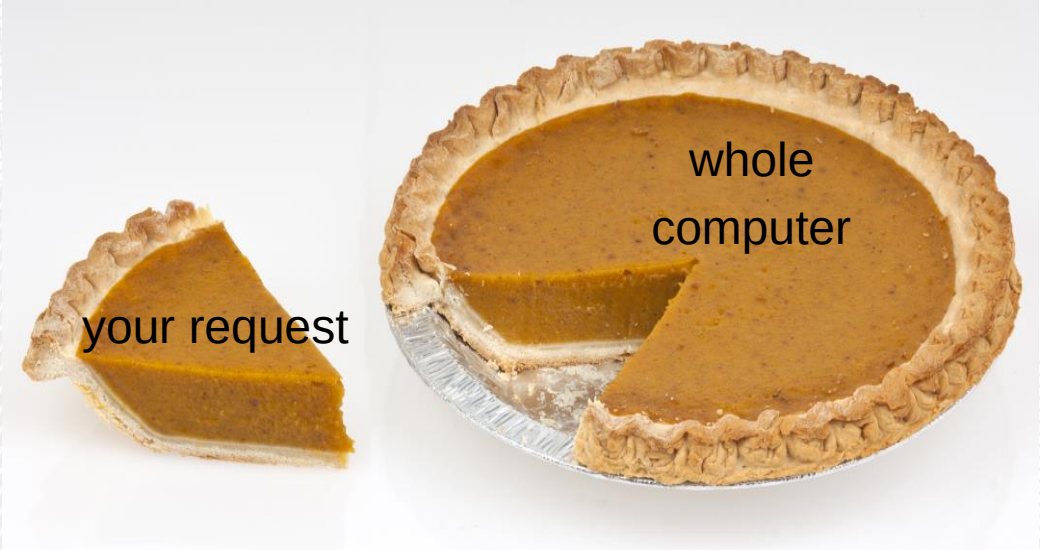
CAT-Analysis.sub

- Request the appropriate resources for your job to run.
- **queue -** keyword indicating “create a job.”

Short interlude - live-demo

Resource Requirements

- Jobs are nearly always using a part of a computer, not the whole thing. EP divides worker node into execute "Slots".
- Very important to request appropriate resources (memory, cpus, disk) for a job
- Job will likely be enforced to only use what it requests



Resource Assumptions

- Try to run default jobs whenever possible in order to use surplus Quota
  IF default resources are not working for your jobtype request Resources accordingly
- Important to run test jobs and use the log file to request the right amount of resources:
  - 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”

How should your piece of the pie look – resource requests

Only request what you need and if it’s beyond the default limits

- Request_Runtime = <seconds> → defaults to 3h if not requested
- Request_Memory = <Mbyte> → defaults to 3GB if not requested
- Request_Disk = <kByte> → defaults to 512 MB
- Request_Cpus = <num of cpus> → defaults to 1 if not requested
- Request_GPUs = <num of gpus> → you need a special resource to use GPUS
- (Request_OpSysAndVer = "<OS>" → defaults to RHEL9)

- Check your ressource requests using ‘***condor_q <jobid> -af Request<ressource>***’ (no ‘_’ here)
- Use same syntax with ***condor_history*** for finished jobs

Lite & bide jobs/ quotas/ priorities

Once you run a lot of jobs this will be a thing

- All jobs in the NAF gets automatically categorized in ‘lite’ and ‘bide’ jobs
  - Lite jobs are submissions with no special resource requests (diskspace as an exception) they will run 3h with 1 core and 3GB memory
  - Lite jobs are able to run on nearly the whole pool, outside of your groups quota (surplus)
- Bide jobs are jobs with ‘bigger’ specific resource requests and are bound to your groups quota, if the quota is tight it will take time to get these started also bigger pieces of ‘the pie’ may be harder to find even if quota and priority are in your favor
- Groupquotas are fix for ‘bide’ jobs and elastic for ‘lite’ jobs
- Inside the groups there is no quotation (as in ‘per user’) instead there is a priority system based on previous and current usage called fairshare
- Use ‘condor_userprio.desy’ (!) to check your current usage, priority etc.

Submitting and Monitoring Jobs

- To submit a job/jobs:
  `condor_submit submit_file_name`
- To monitor submitted jobs, use:
  `condor_q`

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

```
$ condor_q
-- Schedd: submit-1.chtc.wisc.edu : <128.104.101.92:9618?... @ 05/01/19 10:35:54
OWNER  BATCH_NAME    SUBMITTED   DONE   RUN    IDLE  TOTAL JOB_IDS
alice  ID: 128      5/9  11:09     _      _      1      1 128.0


1 jobs; 0 completed, 0 removed, 1 idle, 0 running, 0 held, 0 suspended
```

HTCondor Manual: condor_submit

More about `condor_q`

- By default **`condor_q`** shows:
  - user's job(s) only (as of 8.6)
  - jobs summarized in "batches" (as of 8.6)
- Constrain with username, **`ClusterId`** or full **`JobId`**, which will be denoted **`[U/C/J]`** in the following slides.

```
$ condor_q
-- Schedd: submit-1.chtc.wisc.edu : <128.104.101.92:9618?... @ 05/09/19 11:35:54
OWNER  BATCH_NAME    SUBMITTED   DONE   RUN    IDLE  TOTAL JOB_IDS
alice  ID: 128       5/9  11:09     _      _      1      1 128.0

1 jobs; 0 completed, 0 removed, 1 idle, 0 running, 0 held, 0 suspended
```

`JobId` = `ClusterId` . `ProcId`

More about `condor_q`

- To see individual job information, use:

  `condor_q -nobatch`

```
$ condor_q -nobatch
-- Schedd: submit-1.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 I  0    0.0 compare_states wi.dat us.dat

1 jobs; 0 completed, 0 removed, 1 idle, 0 running, 0 held, 0 suspended
```

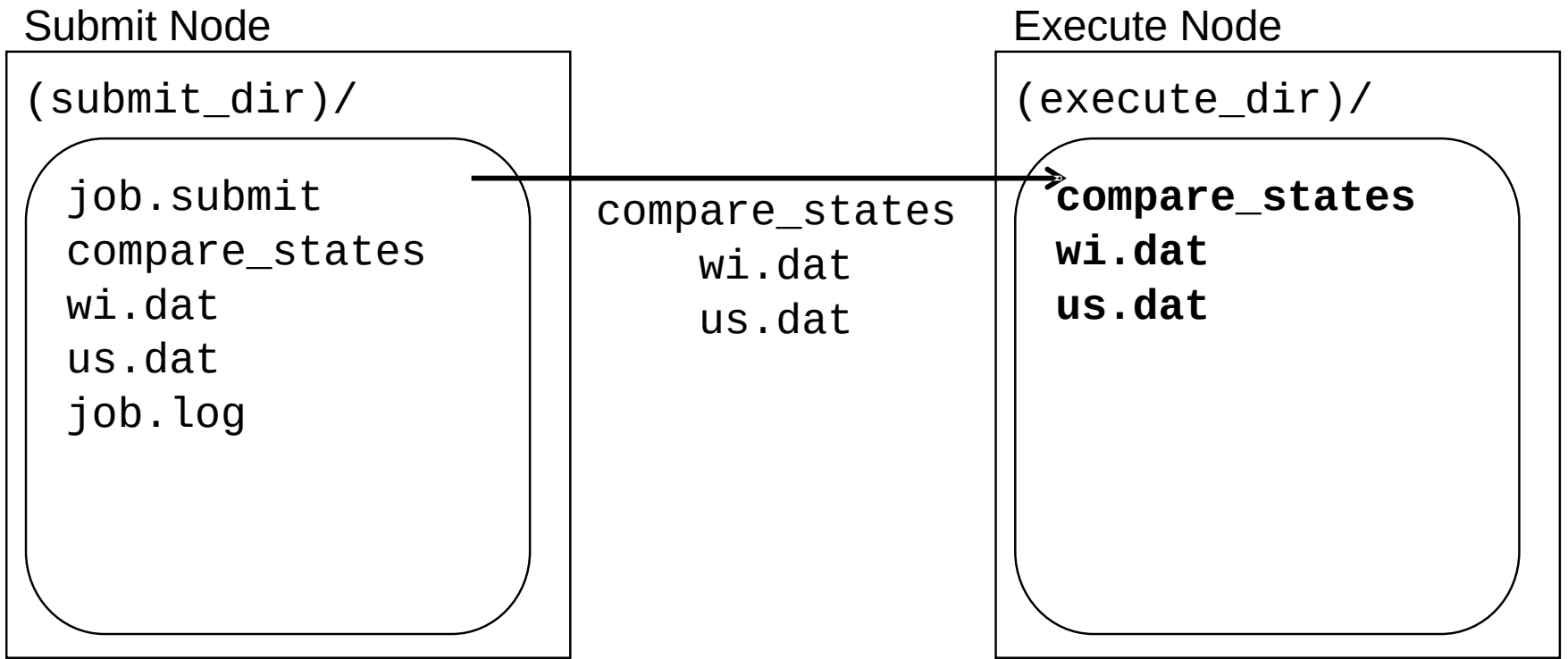
- We will use the `-nobatch` option in the following slides to see extra detail about what is happening with a job

Job Idle

```
$ condor_q -nobatch
-- Schedd: submit-1.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 I  0    0.0 compare_states wi.dat us.dat

1 jobs; 0 completed, 0 removed, 1 idle, 0 running, 0 held, 0 suspended
```

Job Starts

```
$ condor_q -nobatch
-- Schedd: submit-1.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+00:00:0  <  0    0.0 compare_states wi.dat us.dat w

1 jobs; 0 completed, 0 removed, 0 idle, 1 running, 0 held, 0 suspended
```



Job Running

```
$ condor_q -nobatch

-- Schedd: submit-1.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+00:01:03 R  0    0.0 compare_states wi.dat us.dat

1 jobs; 0 completed, 0 removed, 0 idle, 1 running, 0 held, 0 suspended
```

Submit Node

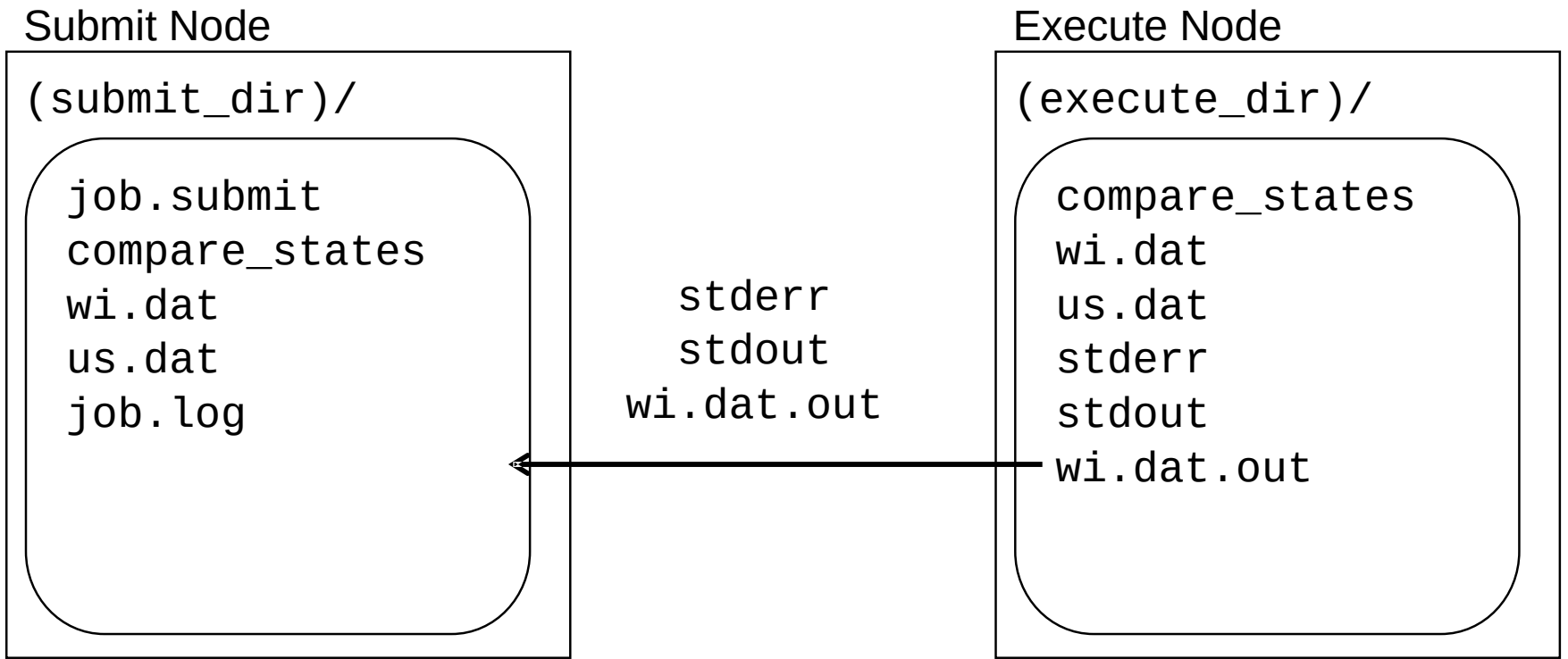
```
(submit_dir)/
    job.submit
    compare_states
    wi.dat
    us.dat
    job.log
```

Execute Node

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

Job Completes

```
$ condor_q -nobatch
-- Schedd: submit-1.chtc.wisc.edu : <128.104.101.92:9618?...
 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
```



Job Completes (cont.)

```
$ condor_q -nobatch

-- Schedd: submit-1.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
```

Submit Node

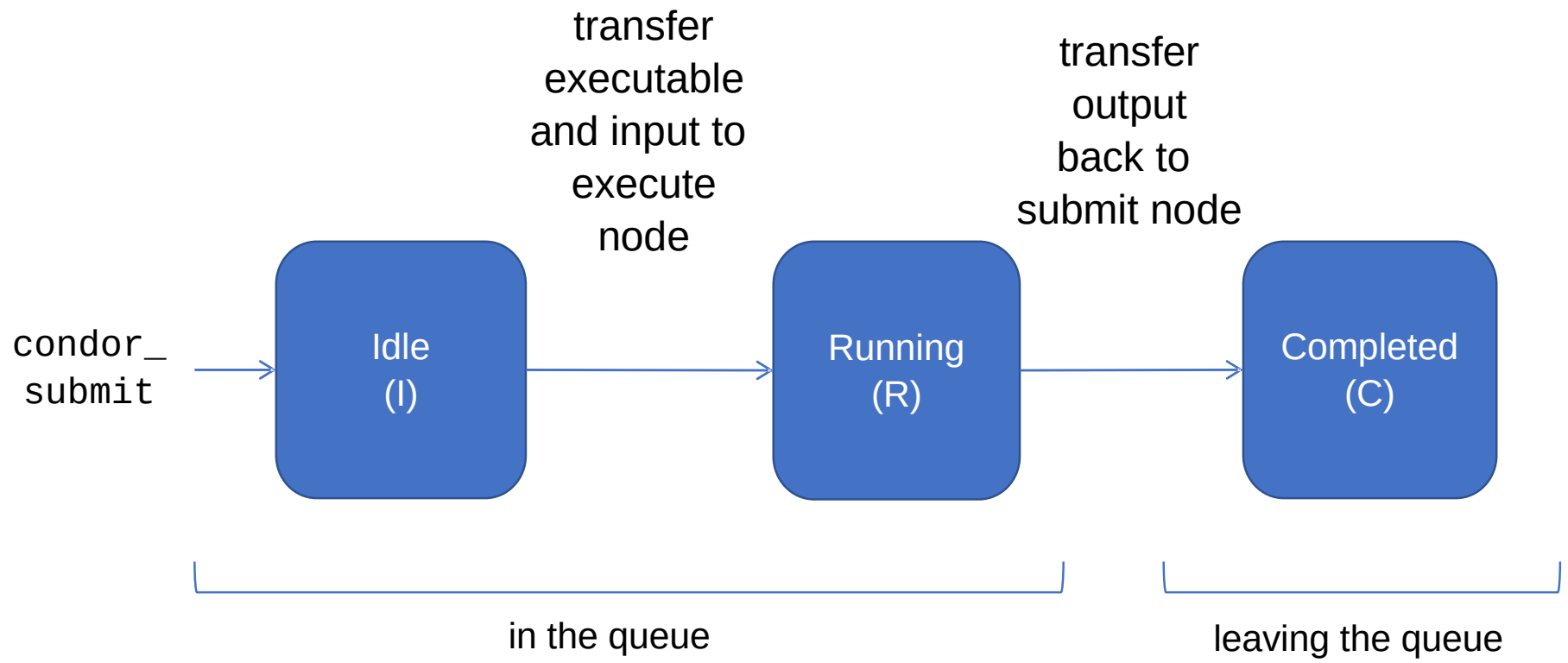
(submit_dir)/

job.submit
compare_states
wi.dat
us.dat
job.log
job.err
job.out
wi.dat.out

Log File

```
000 (7195807.000.000) 05/19 14:30:18 Job submitted from host:
<128.105.244.191:9618 ...>
......
040 (7195807.000.000) 05/19 14:31:55 Started transferring input files
        Transferring to host: <128.105.245.85:9618 ...>
...
040 (7195807.000.000) 05/19 14:31:55 Finished transferring input files
...
001 (7195807.000.000) 05/19 14:31:56 Job executing on host:
<128.105.245.85:9618? ...>
...
005 (7195807.000.000) 05/19 14:35:56 Job terminated.
        (1) Normal termination (return value 0)

        ...

        Partitionable Resources :    Usage  Request Allocated
           Cpus                 :        0        1         1
           Disk (KB)            :       26     1024    995252
                  Memory (MB)           :        1     1024      1024
```

Job States


transfer executable and input to execute node
transfer output back to submit node
condor_ submit
Idle (I)
Running (R)
Completed (C)
in the queue
leaving the queue

Assumptions

- Aspects of your submit file may be dictated by infrastructure and configuration.
- For example: file transfer
  - previous example assumed files would need to be transferred between submit/execute

```
should_transfer_files = YES
```

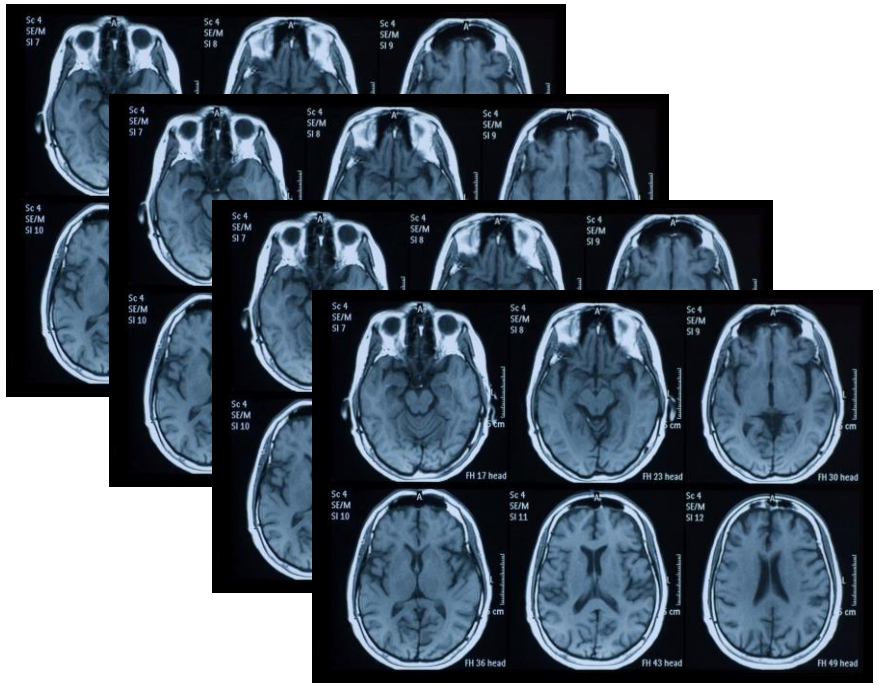
  - not the case with a shared filesystem = true for NAF

```
should_transfer_files = NO
```

Submitting Multiple Jobs with HTCondor

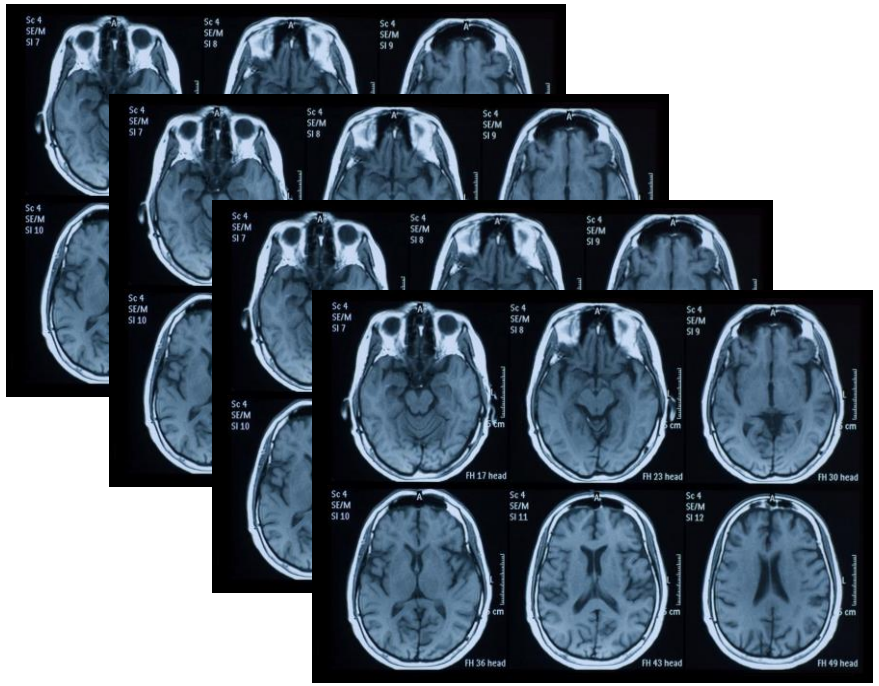
Why do we care?

- Run many independent jobs…
  - analyze multiple data files
  - test parameter or input combinations
  - and more!



Why do we care?

- Run many independent jobs…
  - analyze multiple data files
  - test parameter or input combinations
  - and more!
- …without having to:
  - start each job individually
  - create separate submit files for each job



Many Jobs, One Submit File

- HTCondor has built-in ways to submit multiple independent jobs with one submit file.

Photo by Joanna Kosinska on Unsplash

Numbered Input Files

- Goal: create 3 jobs that each analyze a different input file.

job.submit

```
executable = analyze.exe
arguments = file0.in file0.out
transfer_input_files = file0.in

log = job.log
output = job.out
error = job.err

queue
```

(submit_dir)/

```
analyze.exe
file0.in
file1.in
file2.in

job.submit
```

Multiple Jobs, No Variation

- This file generates 3 jobs, but doesn't use multiple inputs and will overwrite outputs

job.submit

```
executable = analyze.exe
arguments = file0.in file0.out
transfer_input_files = file0.in

log = job.log
output = job.out
error = job.err

queue 3
```

(submit_dir)/

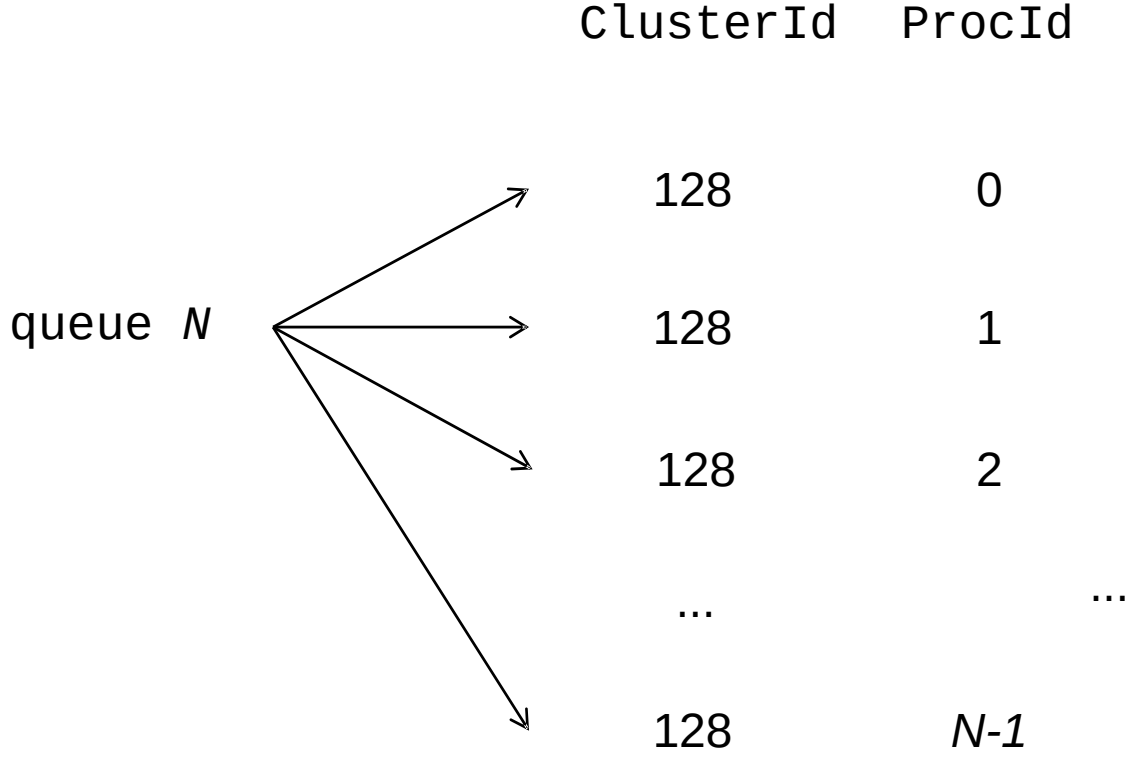
```
analyze.exe
file0.in
file1.in
file2.in

job.submit
```

Automatic Variables

- Each job's `ClusterId` and `ProcId` can be accessed inside the submit file using:

  `$(ClusterId)`

  `$(ProcId)`



Job Variation

- How to uniquely identify each job (filenames, log/out/err names)?

job.submit

```
executable = analyze.exe
arguments = file0.in file0.out
transfer_input_files = file0.in

log = job.log
output = job.out
error = job.err

queue 3
```

(submit_dir)/

```
analyze.exe
file0.in
file1.in
file2.in

job.submit
```

Using $(ProcId)

- Use the `$(ClusterId)`, `$(ProcId)` variables to provide unique values to jobs.*

job.submit

```
executable = analyze.exe
arguments = file$(ProcId).in file$(ProcId).out
transfer_input_files = file$(ProcId).in

log = job-$(ClusterId)-$(ProcId).log
output = job-$(ClusterId)-$(ProcId).out
error = job-$(ClusterId)-$(ProcId).err

queue 3
```

(submit_dir)/

```
analyze.exe
file0.in
file1.in
file2.in

job.submit
```

Submit and Monitor (review)

`condor_submit submit_file_name`

`condor_q`

- Jobs in the queue will be grouped in batches (in this case by cluster number)

```
$ condor_submit job.submit
Submitting job(s).
3 job(s) submitted to cluster 128.
```

```
$ condor_q
-- Schedd: submit-1.chtc.wisc.edu : <128.104.101.92:9618?... @ 05/09/19 10:35:54
OWNER  BATCH_NAME    SUBMITTED    DONE       RUN    IDLE  TOTAL JOB_IDS
alice  ID: 128      5/9  11:03      _      _      3      3      128.0-2

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

HTCondor Manual: condor_submit

Using Batches

- Alternatively, batches can be grouped manually using the `JobBatchName` attribute in a submit file:

```
+JobBatchName = "CoolJobs"
```

```
$ condor_q
OWNER  BATCH_NAME     SUBMITTED   DONE   RUN    IDLE  TOTAL JOB_IDS
alice  CoolJobs      5/9  11:03     _      _      3      3 128.0-2
```

- To see individual jobs, use:

  `condor_q -nobatch`

Estimated end :)

Please consider the rest of the talk – it has a lot of valuable tips and useful information – find it in INDICO !

NAF Introduction - hands-on

Hello world job

- Login to ‘your’ workgroupserver aka submit host should be something like ‘naf-<VO>.desy.de’
- Clone the repo for the hands-on: ***git clone https://gitlab.desy.de/christoph.beyer/naf-hands-on***
- Change the directory to ***naf-hands-on***
- Have a look at the directory structure
- Change directory to ***submit_files***
- Check the hello-world submit file (***hello_world.sub***)
- Run the executable on the comand line (./***hello_world.sh***)
- Use ***condor_submit hello_world.sub***
- Use condor_q to check on your job and condor_history once it has left the queue
- Check the output in ***log/<jobid.procid.out> (e.g. cat ../log/4485756.0.out)***
- Check the log file & the error file
- Let’s do something crazy – alter the queue command to ***queue 1000*** and repeat the other steps

NAF Introduction - hands-on

Containerized job

- Have a look at containerized hello-world submit file (***hello_world_container.sub***)
- Change the initialdir to your actual working dir
- Use ***condor_submit hello_world_container.sub***
- Use condor_q to check on your job and condor_history once it has left the queue
- Check the output in ***log/<jobid.procid.out>***
- Nothing spectacular right ? Let’s check if the container really gets started :)

NAF Introduction - hands-on

Container sleep job

- Have a look at the sleep_container submit file (***sleep_container.sub)***
- Use ***condor_submit sleep_container.sub***
- This job is similar to the previous one but it does not produce output, it goes to ‘sleep mode’
- Use ***condor_q*** to determine if the job is running, when it is
  - Use ***condor_ssh_to_job <JobID>*** to ‘jump’ into the sleeping job on the workernode
- Ps -ef | grep <your UID> should reveal if apptainer is actual running your image
- Maybe there is a better way to proove we are inside a container ?
- Use <CTRL> D to leave the container/job slot

NAF Introduction - hands-on

More complex job

- We want to transform a collection of pics from ‘.jpg’ to ‘.tif’ format using the UNIX tool convert in a one-line script called ***./convert_pic.sh***
- A collection of pics is in ***data/image_collection***
- Check the script and execute it on one pic :
  - ***./convert_pic.sh ../data/opo0004_large.jpg***
  - You should now find ***../data/opo0004_large.tif***
  - Delete ***../data/opo0004_large.tif*** or expect one job to fail :)
- Check the submit file ***convert_pics.sub*** – there is a nice queue feature in it that creates an array job in a ‘foreach’ style
- You need to change the initial working directory (IWD) – if in doubt what to put type <PWD> and use the output
- Run ***condor_submit convert_pics.sub***
- Use condor_q to check the progress and the job structure (array job)
- Use ***ls ../data/*** to check the created converted images

NAF Introduction - hands-on

Impossible job

- Use condor_submit impossible.sub
- Use ***condor_q -analyze <jobid>*** - are there possible machine candidates for this job ?
- Use ***condor_q -better-analyze <jobid>*** - what does the job request and why is it failing ?

NAF Introduction - hands-on

Impossible job

- Use condor_submit impossible.sub
- Use ***condor_q -analyze <jobid>*** - are there possible machine candidates for this job ?
- Use ***condor_q -better-analyze <jobid>*** - what does the job request and why is it failing ?

- Solution: The job requests 300 CPUS which is above the number of CPUS provided per machine

Backup slides – rest of the introduction talk

Very worthwhile !

Organizing Jobs

```
12181445_0.err  16058473_0.err  17381628_0.err  18159900_0.err  5175744_0.err  7266263_0.err
12181445_0.log  16058473_0.log  17381628_0.log  18159900_0.log  5175744_0.log  7266263_0.log
12181445_0.out  16058473_0.out  17381628_0.out  18159900_0.out  5175744_0.out  7266263_0.out
13609567_0.err  16060330_0.err  17381640_0.err  3446080_0.err   5176204_0.err  7266267_0.err
13609567_0.log  16060330_0.log  17381640_0.log  3446080_0.log   5176204_0.log  7266267_0.log
13609567_0.out  16060330_0.out  17381640_0.out  3446080_0.out   5176204_0.out  7266267_0.out
13612268_0.err  16254074_0.err  17381665_0.err  3446306_0.err   5295132_0.err  7937420_0.err
13612268_0.log  16254074_0.log  17381665_0.log  3446306_0.log   5295132_0.log  7937420_0.log
13612268_0.out  16254074_0.out  17381665_0.out  3446306_0.out   5295132_0.out  7937420_0.out
13630381_0.err  17134215_0.err  17381676_0.err  4347054_0.err   5318339_0.err  8779997_0.err
13630381_0.log  17134215_0.log  17381676_0.log  4347054_0.log   5318339_0.log  8779997_0.log
13630381_0.out  17134215_0.out  17381676_0.out  4347054_0.out   5318339_0.out  8779997_0.out
```

Shared Files

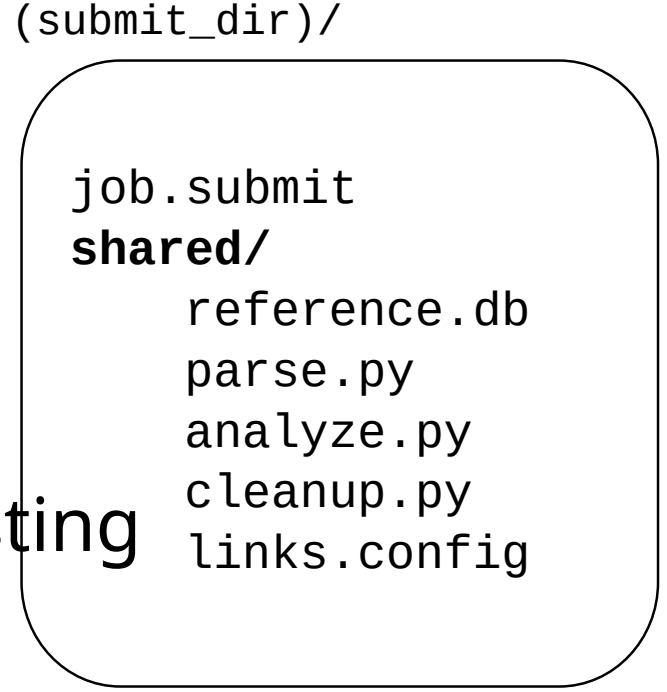
- HTCondor can transfer an entire directory or all the contents of a directory
  - transfer whole directory

```
transfer_input_files = shared
```

  - transfer contents only

```
transfer_input_files = shared/
```

- Useful for jobs with many shared files; transfer a directory of files instead of listing files individually

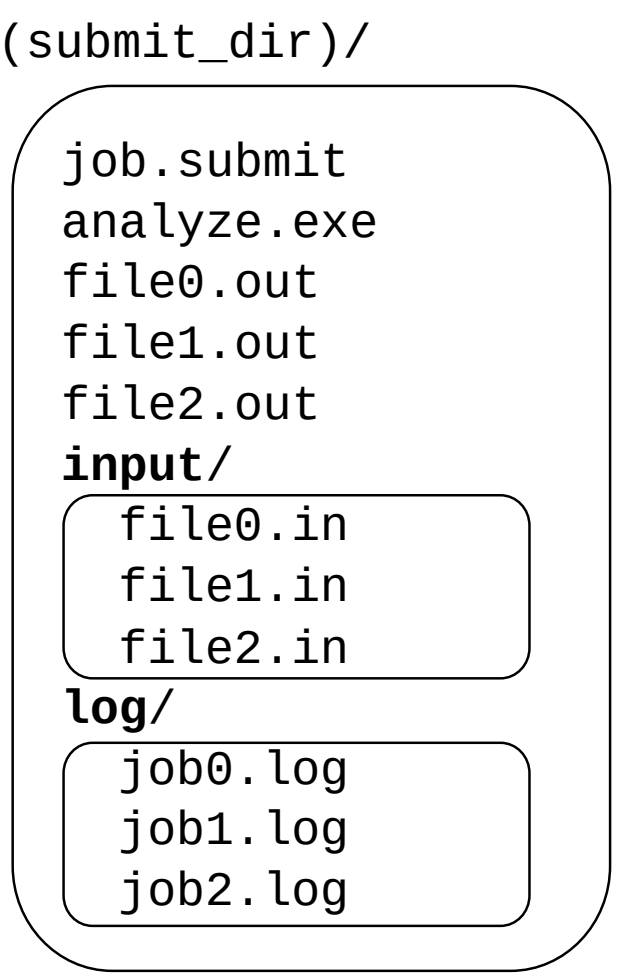


Use Sub-Directories for File Type

- Create sub-directories* and use paths in the submit file to separate input, error, log, and output files.

job.submit

```
executable = analyze.exe
arguments = file$(Process).in file$(ProcId).out
transfer_input_files = input/file$(ProcId).in

log = log/job$(ProcId).log

queue 3
```

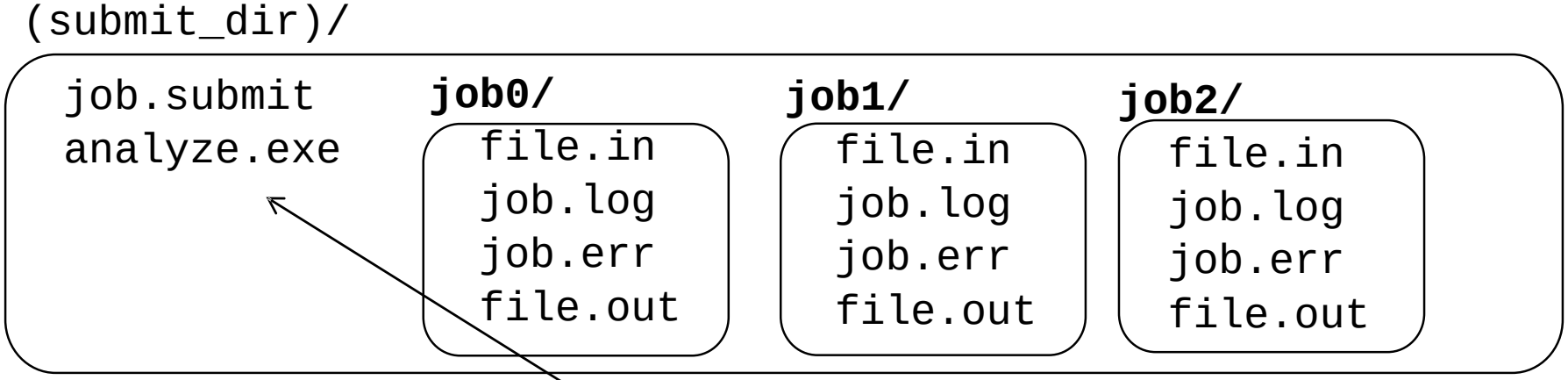


InitialDir

- Change the submission directory for each job using initialdir
- Allows the user to organize job files into separate directories.
- Use the same name for all input/output files
- Useful for jobs with lots of output files

Box by Alice Design from the Noun Project

Separate Jobs with InitialDir



job.submit

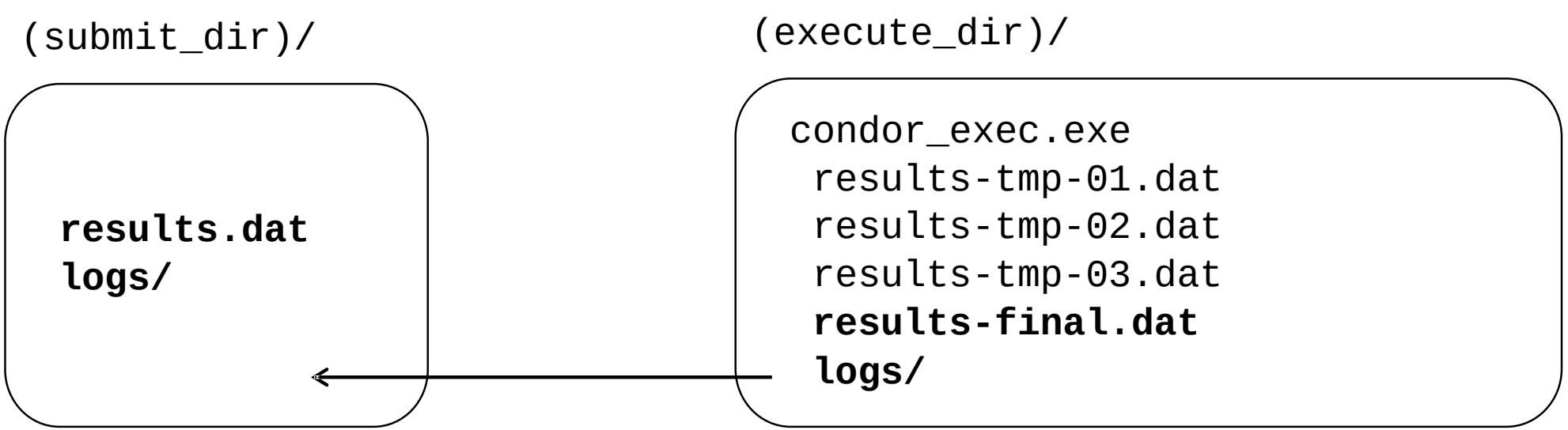
```
executable = analyze.exe
initialdir = job$(ProcId)
arguments = file.in file.out
transfer_input_files = file.in

log = job.log
error = job.err

queue 3
```

Executable should be in the directory with the submit file, *not* in the individual job directories

Output Handling

- Only transfer back specific files or directories from the job's execut using `transfer_ouput_files`
- rename with `transfer_output_remaps`

```
transfer_output_files = results-final.dat, logs
transfer_output_remaps = "results-final.dat=results.dat"
```



Other Submission Methods

- What if your input files/directories aren't numbered from `0` to `(N-1)`?
- There are other ways to submit many jobs!

Photo by Andrew Toskin on Flickr, CC-BY-SA

Submitting Multiple Jobs

- Replacing single job inpu

```
executable = compare_states
arguments = wi.dat us.dat wi.dat.out

transfer_input_files = us.dat, wi.dat

queue 1
```

- with a variable of choice

```
executable = compare_states
arguments = $(infile) us.dat $(infile).out

transfer_input_files = us.dat, $(infile)

queue ...
```

Possible Queue Statements

matching ... pattern	`queue infile matching *.dat`
in ... list	`queue infile in (wi.dat ca.dat ia.dat)`

from ... file	`queue infile from state_list.txt`
	`wi.dat`
	`ca.dat`
	`ia.dat`
	`state_list.txt`

multiple “queue” statements	`infile = wi.dat`
	`queue 1`
	`infile = ca.dat`
	`queue 1`
	`infile = ia.dat`
	`queue 1`

Possible Queue Statements

matching ... pattern	`queue infile matching *.dat`
in ... list	`queue infile in (wi.dat ca.dat ia.dat)`

from ... file	`queue infile from state_list.txt`
	`wi.dat`
	`ca.dat`
	`ia.dat`
	`state_list.txt`

multiple “queue” statements	`infile = wi.dat`
	`queue 1`
	`infile = ca.dat`
	`queue 1`
	`infile = ia.dat`
	`queue 1`
	Not Recommended

Queue Statement Comparison

matching .. pattern	Natural nested looping, minimal programming, use optional “files” and “dirs” keywords to only match files or directories Requires good naming conventions,
in .. list	Supports multiple variables, all information contained in a single file, reproducible Harder to automate submit file creation
from .. file	Supports multiple variables, highly modular (easy to use one submit file for many job batches), reproducible Additional file needed
multiple queue statements	Not recommended. Can be useful when submitting job batches where a single (non-file/argument) characteristic is changing

Using Multiple Variables

- The "from" syntax supports using multiple variables from a list.

job.submit

```
executable = compare_states
arguments = -y $(option) -i $(file)

should_transfer_files = YES
when_to_transfer_output = ON_EXIT
transfer_input_files = $(file)

queue file,option from job_list.txt
```

job_list.txt

```
wi.dat, 2010
wi.dat, 2015
ca.dat, 2010
ca.dat, 2015
ia.dat, 2010
ia.dat, 2015
```

Other Features

- Match existing files or directories:

```
queue input matching files *.dat
```

```
queue directory matching dirs job*
```

- Submit multiple jobs with same input data

```
queue 10 input matching files *.dat
```

  - Use other automatic variables: `$(Step)`

```
arguments = -i $(input) -rep $(Step)
queue 10 input matching files *.dat
```

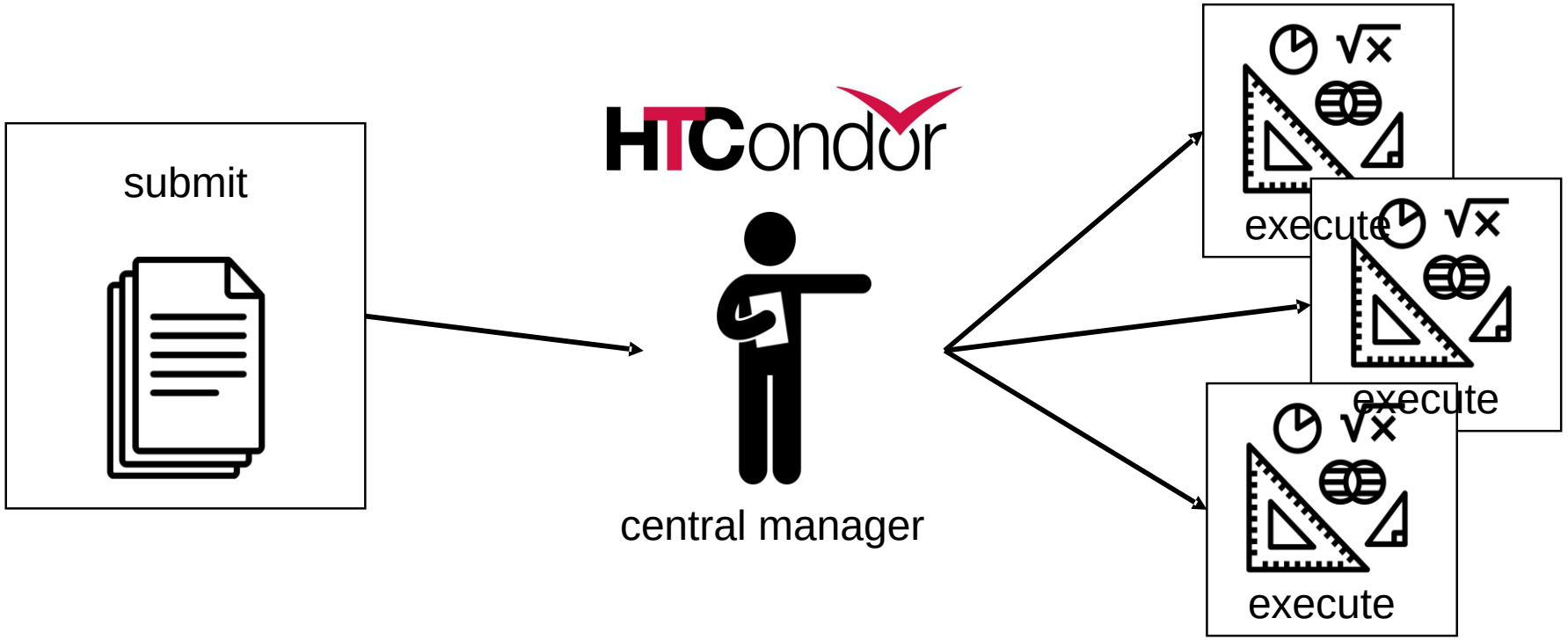
(60 second) Pause

Questions so far?

Job Matching and Class Ad Attributes

The Central Manager

- HTCondor matches jobs with computers via a “central manager”.



manager by Gan Khoon Lay from the Noun Project

Class Ads

- HTCondor stores a list of information about each job and each computer.
- This information is stored as a “Class Ad”
- Class Ads have the format:
  - `AttributeName = value`

can be a boolean, number, or string

Photo by Wherda Arsianto on Unsplash

Job Class Ad

```
executable = compare_states
arguments = wi.dat us.dat wi.dat.out

should_transfer_files = YES
transfer_input_files = us.dat, wi.dat
when_to_transfer_output = ON_EXIT

log = job.log
output = job.out
error = job.err

request_cpus = 1
request_disk = 20MB
request_memory = 20MB

queue 1
```

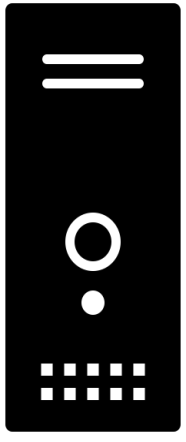
+

HTCondor configuration*

=>

```
RequestCpus = 1
Err = "job.err"
WhenToTransferOutput = "ON_EXIT"
TargetType = "Machine"
Cmd =
"/home/alice/tests/htcondor_week/compare_states"
JobUniverse = 5
Iwd = "/home/alice/tests/htcondor_week"
RequestDisk = 20480
NumJobStarts = 0
WantRemoteIO = true
TransferInput = "us.dat,wi.dat"
MyType = "Job"
Out = "job.out"
UserLog =
"/home/alice/tests/htcondor_week/job.log"
RequestMemory = 20
...
```

Computer “Machine” Class Ad



+

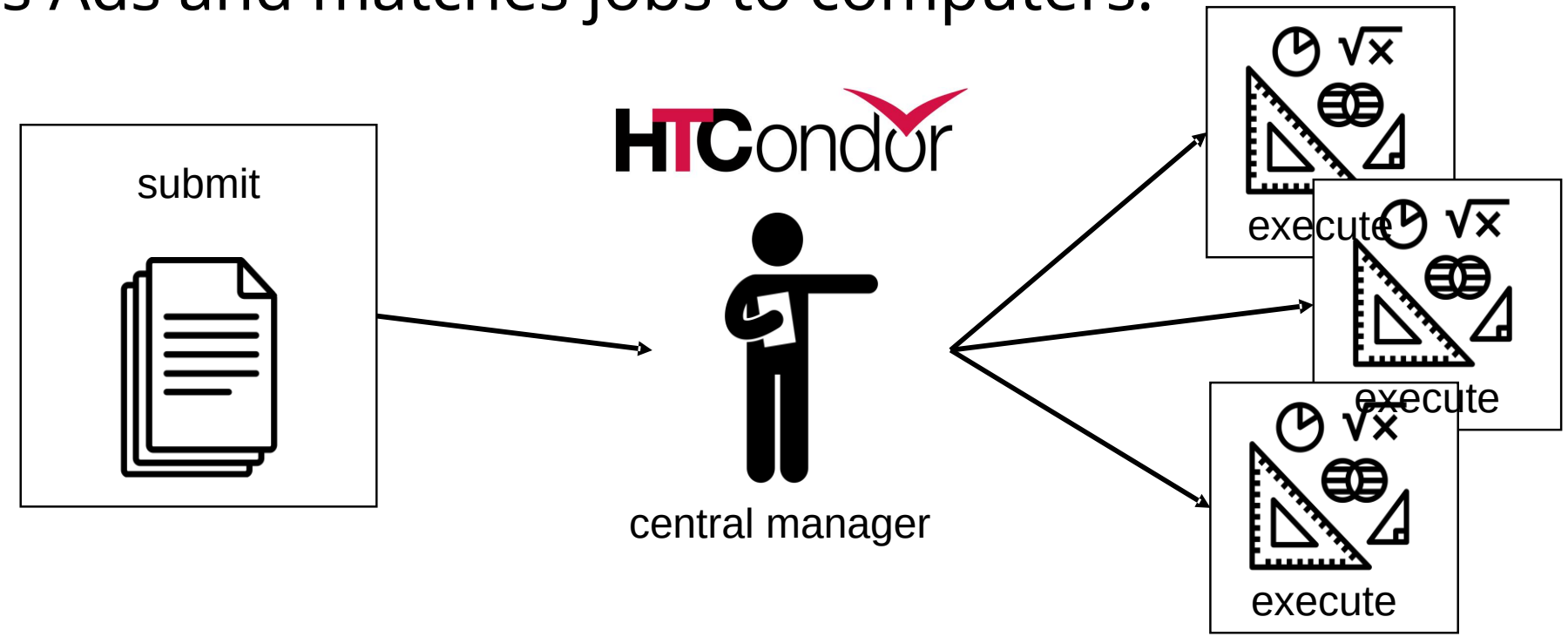
HTCondor configuration

=>

```
HasFileTransfer = true
DynamicSlot = true
TotalSlotDisk = 4300218.0
TargetType = "Job"
TotalSlotMemory = 2048
Mips = 17902
Memory = 2048
UtsnameSysname = "Linux"
MAX_PREEMPT = ( 3600 * 72 )
Requirements = ( START ) && (
IsValidCheckpointPlatform ) && (
WithinResourceLimits )
OpSysMajorVer = 6
TotalMemory = 9889
HasGluster = true
OpSysName = "SL"
HasDocker = true
...
```

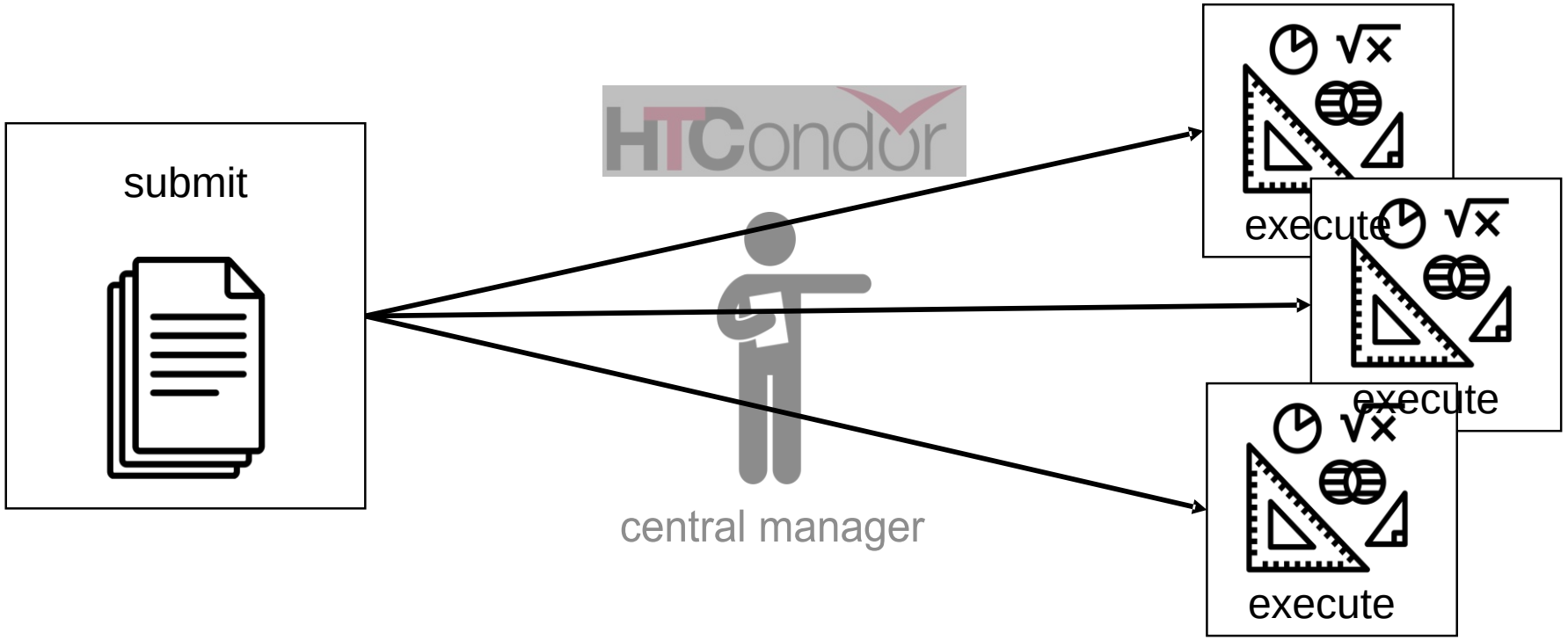
Job Matching

- On a regular basis, the central manager reviews Job and Machine Class Ads and matches jobs to computers.



Job Execution

- (Then the submit and execute points communicate directly.)



Class Ads for People

- Class Ads also provide lots of useful information about jobs and computers to HTCondor users and administrators

Photo by Roman Kraft on Unsplash

Finding Job Attributes

- Use the “long” option for **condor_q**

  condor_q -l *JobId*

```
$ condor_q -l 128.0
WhenToTransferOutput = "ON_EXIT"
TargetType = "Machine"
Cmd = "/home/alice/tests/htcondor_week/compare_states"
JobUniverse = 5
Iwd = "/home/alice/tests/htcondor_week"
RequestDisk = 20480
NumJobStarts = 0
WantRemoteIO = true
OnExitRemove = true
TransferInput = "us.dat,wi.dat"
MyType = "Job”
UserLog = "/home/alice/tests/htcondor_week/job.log"
RequestMemory = 20
...
```

Useful Job Attributes

- **`UserLog`**: location of job log
- **`Iwd`**: Initial Working Directory (i.e. submission directory) on submit node
- **`MemoryUsage`**: maximum memory the job has used
- **`RemoteHost`**: where the job is running
- **`ClusterId`**, **`ProcID`**, **`JobBatchName`**
- …and more (see the manual)

Displaying Job Attributes

- Use the “auto-format” option:
  condor_q **[U/C/J] -af *Attribute1 Attribute2 ...***

```
$ condor_q -af ClusterId ProcId RemoteHost MemoryUsage

1725 116 slot1_1@e092.chtc.wisc.edu 1709
1725 118 slot1_2@e093.chtc.wisc.edu 1709
1725 137 slot1_8@e125.chtc.wisc.edu 1709
1725 139 slot1_7@e121.chtc.wisc.edu 1709
1861 0 slot1_5@c025.chtc.wisc.edu 196
1863 0 slot1_3@atlas10.chtc.wisc.edu 269
1864 0 slot1_25@e348.chtc.wisc.edu 245
1865 0 slot1_23@e305.chtc.wisc.edu 196
1871 0 slot1_6@e176.chtc.wisc.edu 220
```

Selecting Job Attributes

- Use the "constraint" option, along with an expression for what jobs you want to look at:

  condor_q **[U/C/J] -constraint '*Attribute >/</== value*'**

```
$ condor_q -constraint 'JobBatchName == "CoolJobs"'

OWNER  BATCH_NAME     SUBMITTED   DONE   RUN    IDLE  TOTAL JOB_IDS
alice  CoolJobs      5/9  11:03     _      _      3      3 128.0-2
```

Other Displays

- See the whole queue (all users, all jobs)
  `condor_q -all`

```
$ condor_q -all

-- Schedd: submit-1.chtc.wisc.edu : <128.104.101.92:9618?...
OWNER    BATCH_NAME    SUBMITTED   DONE   RUN    IDLE   HOLD  TOTAL JOB_IDS
alice    DAG: 128      5/9  02:52    982      2       _      _   1000 18888976.0 ...
bob      DAG: 139      5/9  09:21      _      1      89      _    180 18910071.0 ...
alice    DAG: 219      5/9  10:31      1    997       2      _   1000 18911030.0 ...
bob      DAG: 226      5/9  10:51     10      _       1      _     44 18913051.0
bob      CMD: ce.sh    5/9  10:55      _      _       _      2      _ 18913029.0 ...
alice    CMD: sb       5/9  10:57      _      2     998      _      _ 18913030.0-999
```

Class Ads for Computers

- as **`condor_q`** is to jobs, **`condor_status`** is to computers (or “machines”)

```
$ condor_status
Name                                                    OpSys     Arch      State
                  Activity  LoadAv    Mem Actvty
slot1@c001.chtc.wisc.edu                  LINUX      X86_64 Unclaimed Idle      0.000     673
25+01
slot1_1@c001.chtc.wisc.edu           LINUX      X86_64 Claimed    Busy      1.000    2048  0+01
slot1_2@c001.chtc.wisc.edu           LINUX      X86_64 Claimed    Busy      1.000    2048  0+01
slot1_3@c001.chtc.wisc.edu           LINUX      X86_64 Claimed    Busy      1.000    2048  0+00
slot1_4@c001.chtc.wisc.edu           LINUX      X86_64 Claimed    Busy      1.000    2048  0+14
slot1@c002.chtc.wisc.edu             LINUX      X86_64 Unclaimed Idle      1.000    2693 19+19
slot1_1@c002.chtc.wisc.edu           LINUX      X86_64 Claimed    Busy      1.000    2048  0+04
slot1_2@c002.chtc.wisc.edu           LINUX      X86_64 Claimed    Busy      1.000    2048  0+01
slot1@c004.chtc.wisc.edu             LINUX      X86_64 Unclaimed Idle      0.010     645 25+05
slot1_1@c004.chtc.wisc.edu           LINUX      X86_64 Claimed    Busy      1.000    2048  0+01

                                 Total Owner Claimed Unclaimed Matched Preempting
Backfill  Drain

        X86_64/LINUX 10962     0   10340       613       0         0        0      9
      X86_64/WINDOWS     2     2       0         0       0         0        0      0

               Total 10964     2   10340       613       0         0        0      9
```

Machine Attributes

- To summarize, use the "-compact" option

  `condor_status -compact`

```
$ condor_status -compact
Machine                      Platform     Slots Cpus Gpus  TotalGb FreCpu  FreeGb  CpuLoad ST
e007.chtc.wisc.edu           x64/SL6          8    8          23.46      0     0.00    1.24 Cb
e008.chtc.wisc.edu           x64/SL6          8    8          23.46      0     0.46    0.97 Cb
e009.chtc.wisc.edu           x64/SL6         11   16          23.46      5     0.00    0.81 **
e010.chtc.wisc.edu           x64/SL6          8    8          23.46      0     4.46    0.76 Cb
matlab-build-1.chtc.wisc.edu x64/SL6          1   12          23.45     11    13.45    0.00 **
matlab-build-5.chtc.wisc.edu x64/SL6          0   24          23.45     24    23.45    0.04 Ui
mem1.chtc.wisc.edu           x64/SL6         24   80        1009.67      8     0.17    0.60 **

                     Total Owner Claimed Unclaimed Matched Preempting Backfill  Drain

          x64/SL6 10416     0    9984       427       0          0        0      5
    x64/WinVista     2     2       0         0       0          0        0      0

            Total 10418     2    9984       427       0          0        0      5
```

Machine Attributes

- Use same options as **`condor_q`**:

  `condor_status` **`-l`** ***`Slot/Machine`***

  `condor_status` **`[Machine] -af`** ***`Attribute1 Attribute2 ...`***

```
$ condor_status -l slot1_1@c001.chtc.wisc.edu
HasFileTransfer = true
COLLECTOR_HOST_STRING = "cm.chtc.wisc.edu”
TargetType = "Job”
TotalTimeClaimedBusy = 43334c001.chtc.wisc.edu
Mips = 17902
MAX_PREEMPT = ( 3600 * ( 72 - 68 * ( WantGlidein =?= true ) ) )
Requirements = ( START ) && ( IsValidCheckpointPlatform ) && ( WithinResourceLimits )
State = "Claimed"
OpSysMajorVer = 6
OpSysName = "SL”
...
```

Testing and Troubleshooting

What Can Go Wrong?

- Jobs can go wrong “internally”:
  - something happens after the executable begins to run
- Jobs can go wrong from HTCondor’s perspective:
  - A job can’t be started at all,
  - Uses too much memory,
  - Has a badly formatted executable,
  - And more…

Reviewing Failed Jobs

- A job’s log, output and error files can provide valuable information troubleshooting

Log	**Output**	**Error**
• When jobs were submitted, started, and stopped • Resources used • Exit status • Where job ran • Interruption reasons	Any “print” or “display” information from your program	Captured by the operating system

Reviewing Recent Jobs

- To review a large group of jobs at once, use **`condor_history [U/C/J]`**
- 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
```

“Live” Troubleshooting

- To log in to a job where it is running, use:
  `condor_ssh_to_job JobId`

```
$ condor_ssh_to_job 128.0
Welcome to slot1_31@e395.chtc.wisc.edu!
Your condor job is running with pid(s) 3954839.
```

Held Jobs

- HTCondor will put your job on hold there's something YOU need to fix.
- A job that goes on hold is interrupte (all progress is lost) and kept from running again, but remains in the queue in the "H" state.

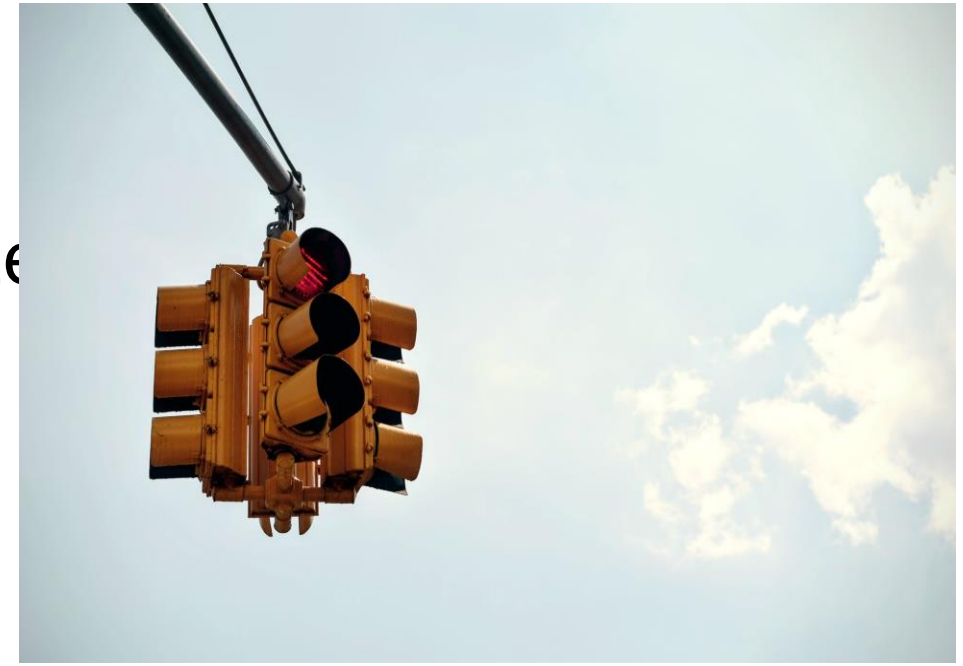


Photo by Tim Gouw on Unsplash

```
$ condor_q -nobatch
ID     OWNER  SUBMITTED   RUN_TIME   ST PRI SIZE CMD
128.0  alice  5/9  11:09  0+00:00:00 H  0    0.0 analyze.exe

1 jobs; 0 completed, 0 removed, 0 idle, 0 running, 1 held, 0 suspended
```

Diagnosing Holds

- If HTCondor puts jobs on hold, it provides a hold reason, which can viewed with:

  `condor_q -hold`

```
$ condor_q -hold
ID    OWNER    HELD_SINCE  HOLD_REASON
125.0 bob      5/09 17:12  Error from slot1_1@wid-003.chtc.wisc.edu: Job has
  gone over memory limit of 2048 megabytes.
128.0 alice    5/11 12:06  Error from slot1_11@e138.chtc.wisc.edu: STARTER
  at 128.104.101.138 failed to send file(s) to <128.104.101.92:9618>; SHADOW at
  128.104.101.92 failed to write to file /home/alice/Test_18925319_16.err:
  (errno 122) Disk quota exceeded
131.0 bob      5/12 09:02  Error from slot1_38@e270.chtc.wisc.edu: Failed
  to execute '/var/lib/condor/execute/slot1/dir_2471876/condor_exec.exe' with
  arguments 2: (errno=2: 'No such file or directory')
```

Common Hold Reasons

- Job has used more memory than requested
- Incorrect path to files that need to be transferred
- Badly formatted bash scripts (have Windows instead of Unix line endings, are missing a header)
- Submit directory is over quota
- The administrator has put your job on hold

Fixing Holds

- Job attributes can be edited while jobs are in the queue using:
  `condor_qedit [U/C/J] Attribute Value`

```
$ condor_qedit 128.0 RequestMemory 3072
Set attribute ”RequestMemory".
```

- If a job has been fixed and can run again, release it with:
  `condor_release [U/C/J]`

```
$ condor_release 128.0
Job 18933774.0 released
```

HTCondor Manual: condor_qedit

Holding or Removing Jobs

- If you know your job has a problem and it hasn't yet completed, yo can:
  - Place it on hold yourself, with **`condor_hold [U/C/J]`**

```
$ condor_hold bob
All jobs of user ”bob" have been held
```

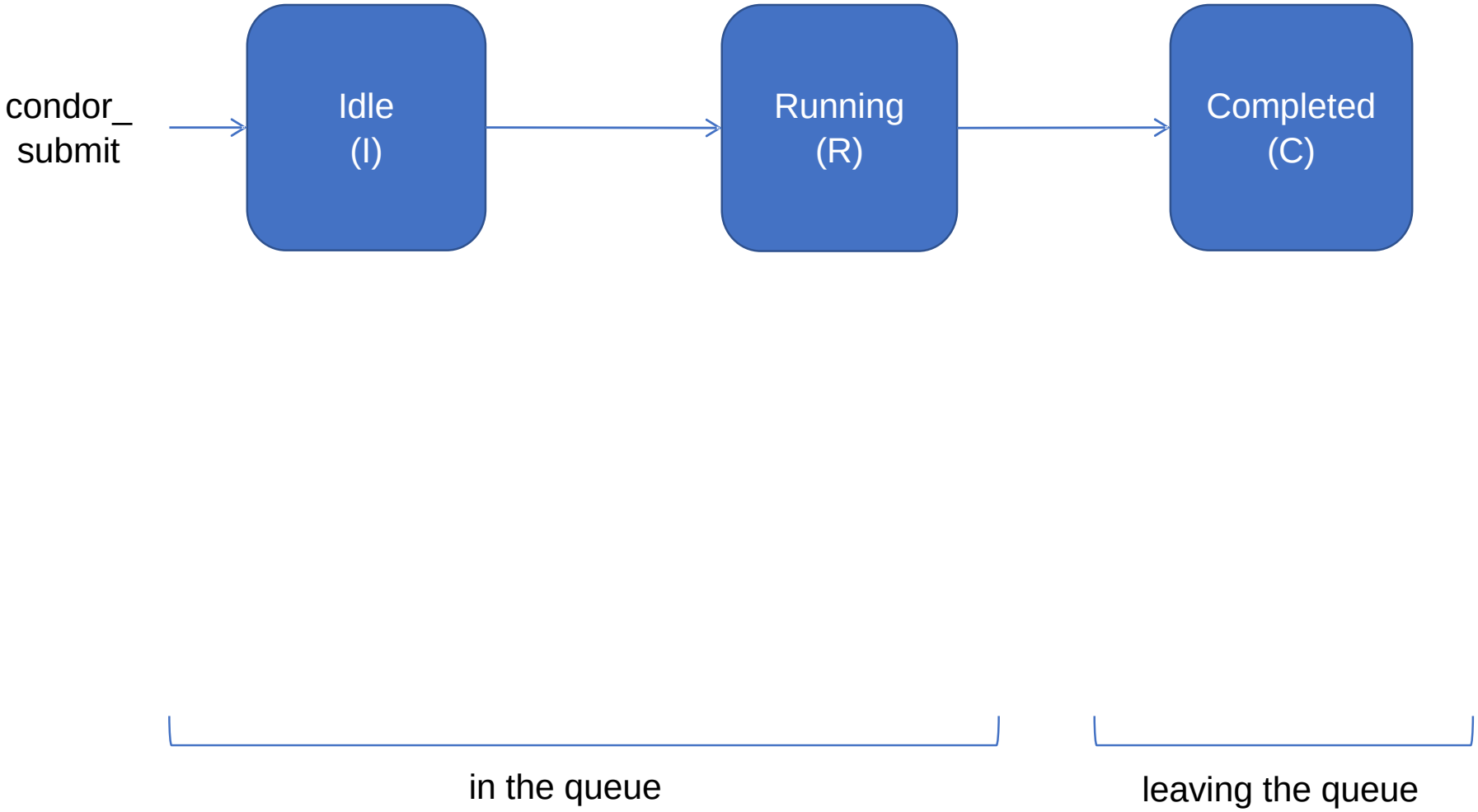
```
$ condor_hold 128
All jobs in cluster 128 have been held
```

```
$ condor_hold 128.0
Job 128.0 held
```

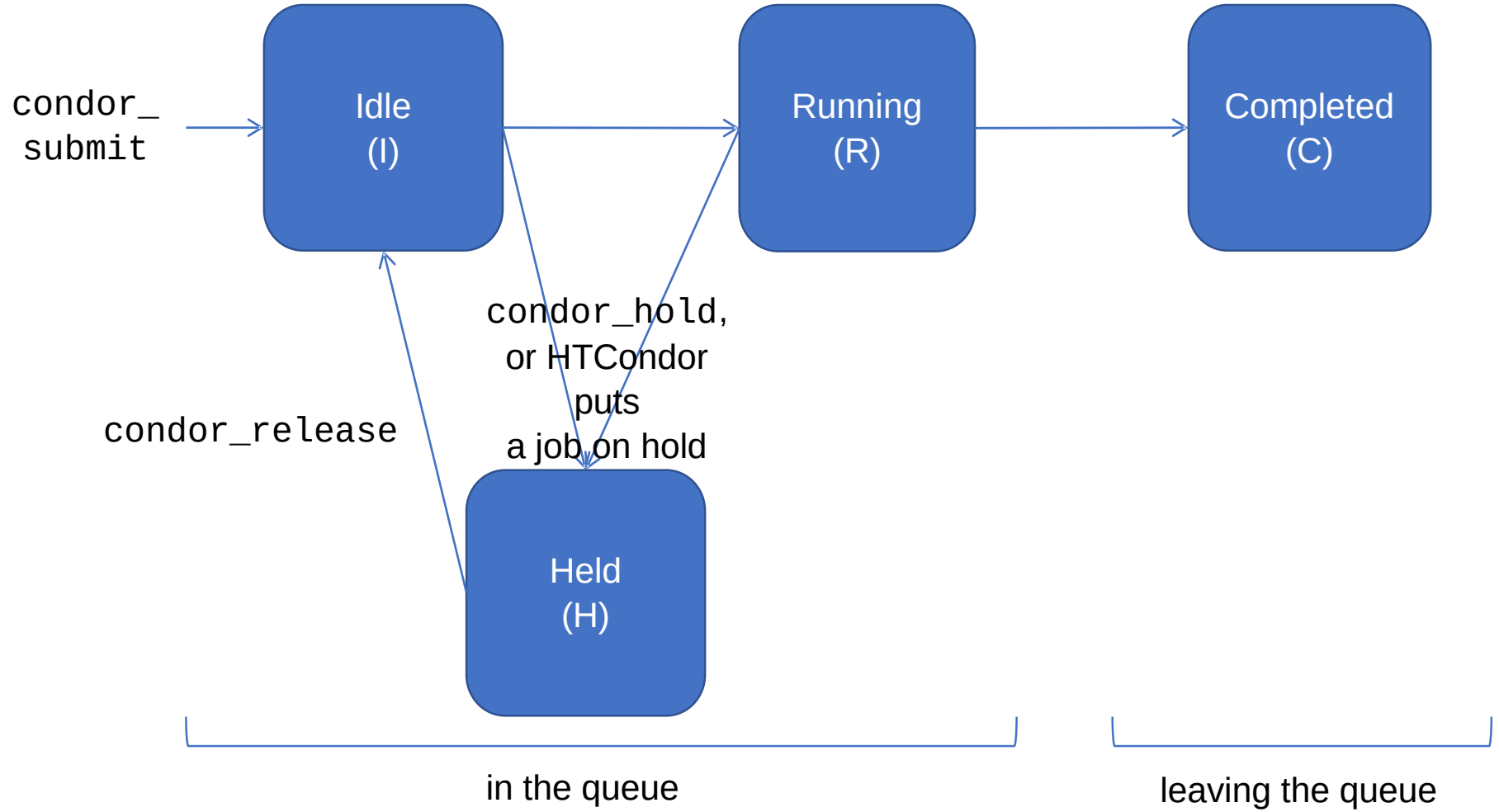
  - Remove it from the queue, using **`condor_rm [U/C/J]`**

HTCondor Manual: condor_hold

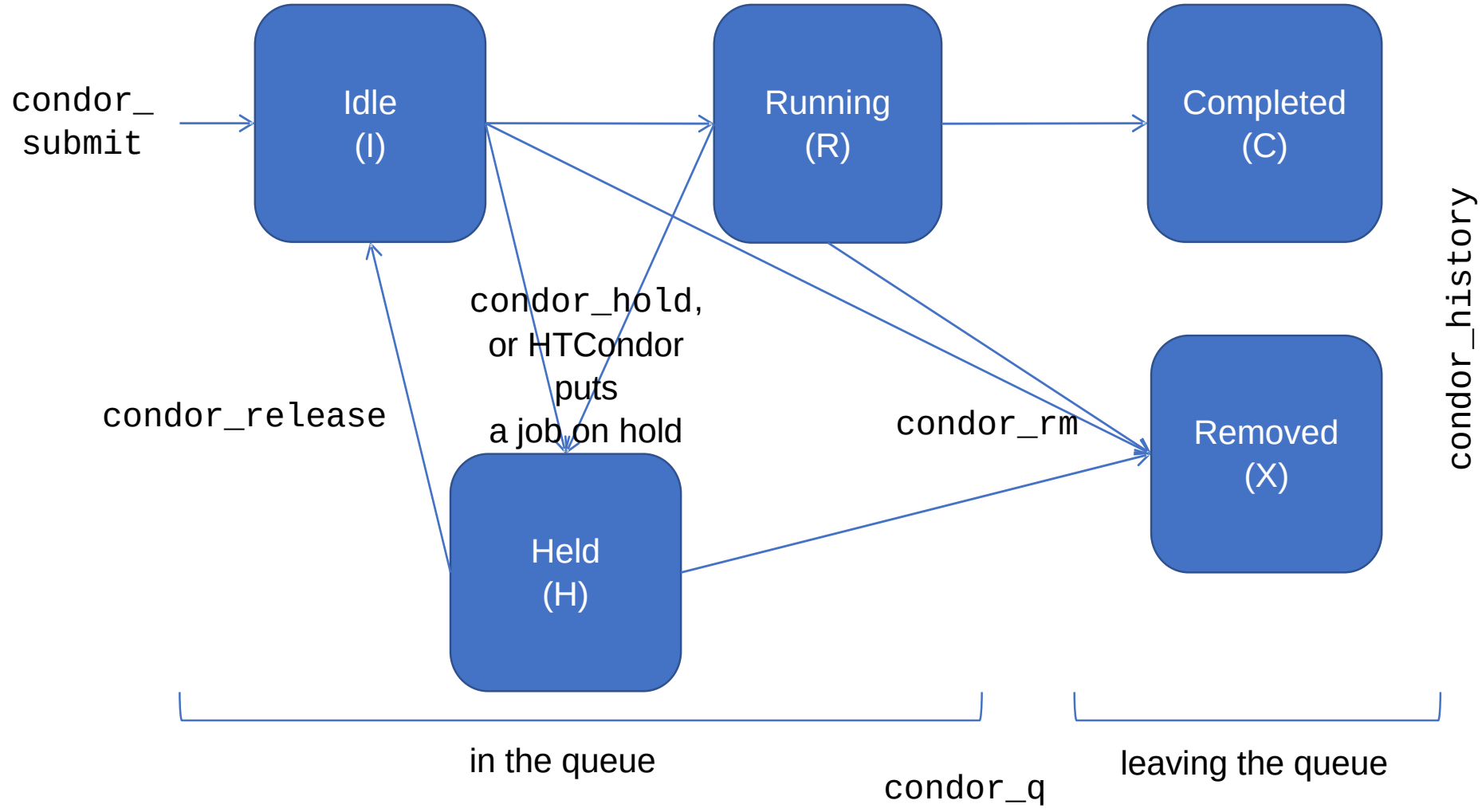
Job States, Revisited



Job States, Revisited



Job States, Revisited*



Use Cases and HTCondor Features

Interactive Jobs

- An interactive job proceeds like a normal batch job, but opens a ba session into the job's execution directory instead of running an executable.

```
condor_submit -i submit_file
```

```
$ condor_submit -i interactive.submit
Submitting job(s).
1 job(s) submitted to cluster 18980881.
Waiting for job to start...
Welcome to slot1_9@e184.chtc.wisc.edu!
```

- Useful for testing and troubleshooting

Self-Checkpointing

- By default, a job that is interrupted will start from the beginning if it is restarted.
- It is possible to implement self-checkpointing, which will allow a job to restart from a saved state if interrupted.
- Self-checkpointing is useful for:
  - very long jobs
  - running on opportunistic resources.

Self-Checkpointing How-To

- Edit executable:
  - Save intermediate states to a checkpoint file
  - Always check for a checkpoint file when starting
- Add HTCondor option that a) saves all intermediate/output files fro
  the interrupted job and b) transfers them to the job when HTCondo
  runs it again

```
when_to_transfer_output = ON_EXIT_OR_EVICT
```

Job Universes

- HTCondor has different “universes” for runni
  specialized job types
  - HTCondor Manual: Choosing an HTCondor Unive
- Vanilla (default)
  - good for most software
  - HTCondor Manual: Vanilla Universe
  - Set in the submit file using:

```
universe = vanilla
```

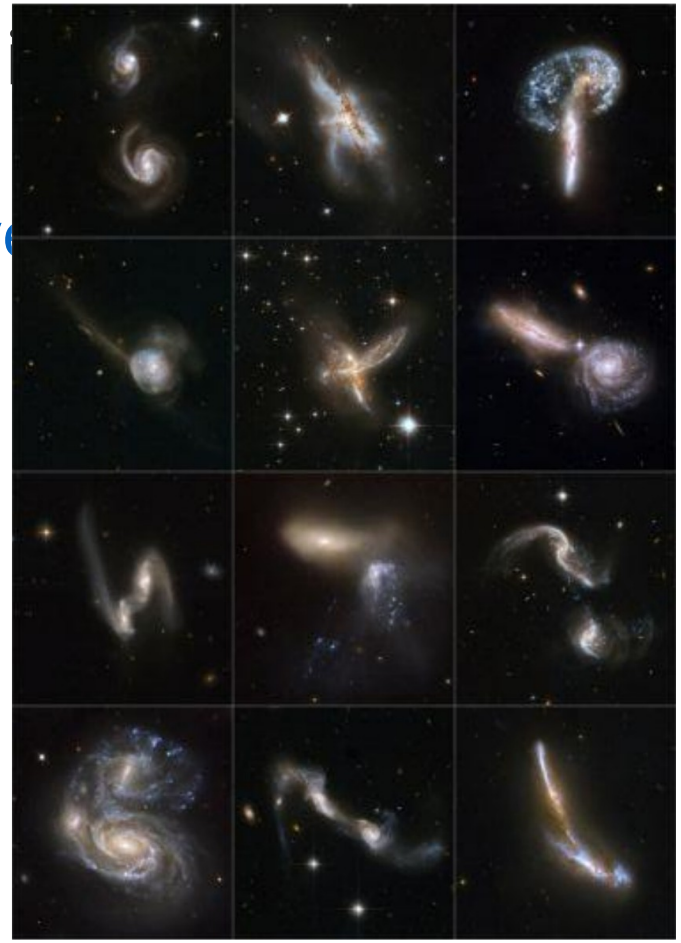
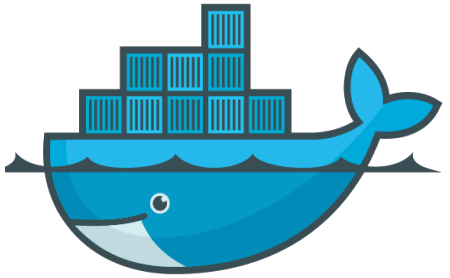


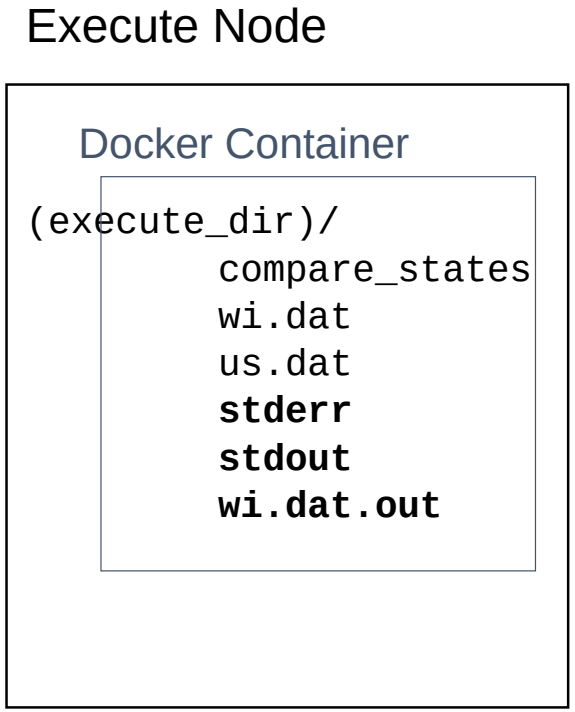
Photo on ~~needpix.com~~

Other Universes

- Docker
  - Run jobs inside a Docker container
  - HTCondor Manual: Docker Universe Applications



```
universe = docker
docker_image = ubuntu:trusty
# by default the docker image
# is pulled from DockerHub
```



Other Universes

- Java
  - Built-in Java support
- Local
  - Run jobs on the submit node
- Standard
  - (No longer supported)
  - For C code compiled against HTCondor libraries
- VM
  - Run jobs inside a virtual machine
- Parallel
  - Used for coordinating jobs across multiple servers (e.g. MPI code)
  - Not necessary for single server multi-core jobs

Multi-CPU and GPU Computing

- Jobs that use multiple cores on a single computer can be r
  in the vanilla universe (parallel universe not needed):

```
request_cpus = 16
```

- If there are computers with GPUs, request them with:

```
request_gpus = 1
```

Automation

Automation

- After job submission, HTCondor manages jobs based on its configuration
- You can use options that will customize job management even further
- These options can automate when jobs are started, stopped, and removed.

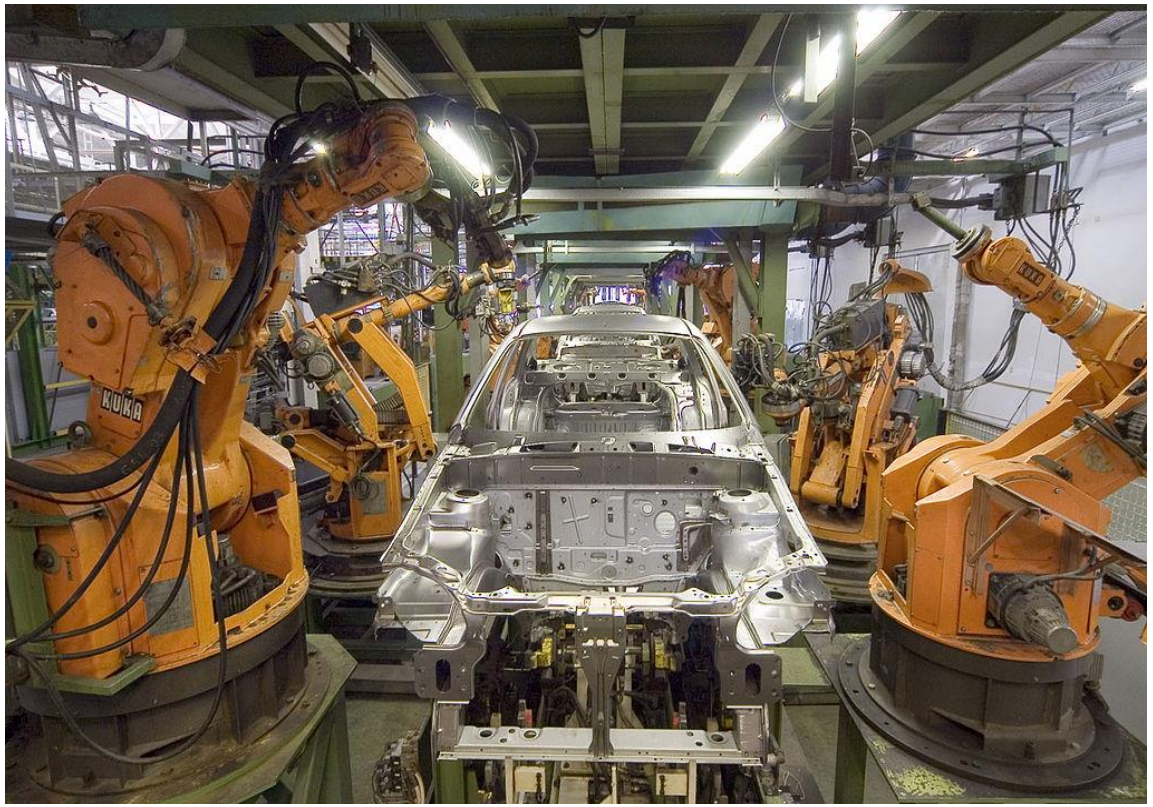


Photo by Mixabest on WikiMedia, CC-BY-SA

Retries

- **Problem**: a small number of jobs fail; if they run again, they comple successfully.
- **Solution**: If the job exits with an error, leave it in the queue to run again. This is done via the automatic option `max_retries`.

```
max_retries = 5
```

Limiting Jobs

- **Problem**: Submitting more than a few thousand jobs to the queue once
- **Solution**: Use the `max_idle` option. This limits the number of jobs submitted at one time, but allows there to always be idle jobs ready to run.

```
max_idle = 1000
```

Useful Job Attributes for Automation

- `CurrentTime`: current time
- `EnteredCurrentStatus`: time of last status change
- `ExitCode`: the exit code from the job
- `HoldReasonCode`: number corresponding to a hold reason
- `NumJobStarts`: how many times the job has gone from idle to running
- `JobStatus`: number indicating idle, running, held, etc.

Automatically Hold Jobs

- **Problem**: Your job should run in 2 hours or less, but a few jobs “ha randomly and run for days
- **Solution**: Put jobs on hold if they run for over 2 hours, using a `periodic_hold` statement

```
periodic_hold = (JobStatus == 2) &&
((CurrentTime - EnteredCurrentStatus) > (60 * 60 * 2))
```

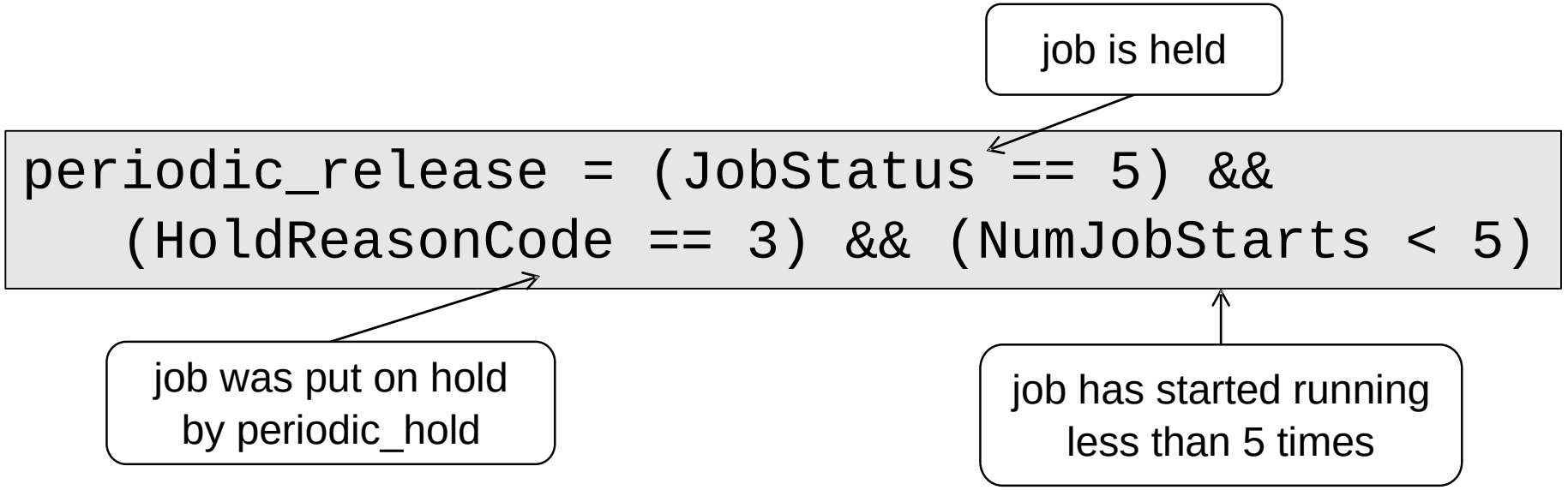
job is running

How long the job has been running, in seconds

2 hours

Automatically Release Jobs

- **Problem** (related to previous): A few jobs are being held for runnin long; they will complete if they run again.
- **Solution**: automatically release those held jobs with a `periodic_release` option, up to 5 times



Automatically Remove Jobs

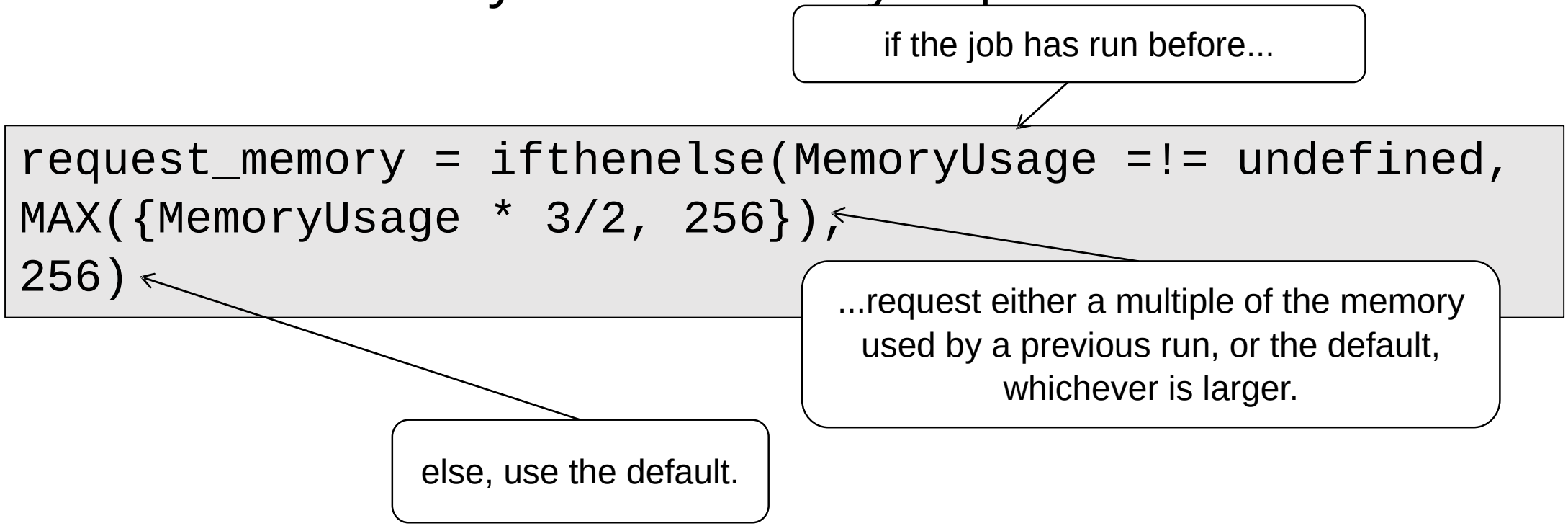
- **Problem**: Jobs are repetitively failing
- **Solution**: Remove jobs from the queue using a `periodic_remove` statement

```
periodic_remove = (NumJobsStarts > 5)
```

job has started running more than 5 times

Dynamically Request Memory

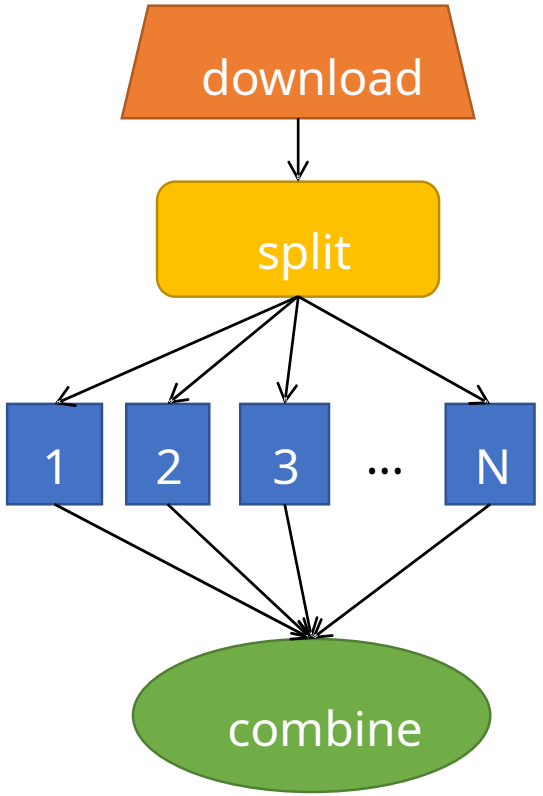
- **Problem**: a batch of jobs uses a wide variety of memory; many jobs only need 256MB, but some need up to 2 GB.
- **Solution**: Use a dynamic memory request.



Workflows

- **Problem**: Want to submit jobs in a particular order, with dependencies between groups of jobs
- **Solution**: Write a DAG

- To learn about this, stay for the next talk, ~~DAGMan: HTCondor and Workflows~~ by Lauren Michael.



Final Questions?