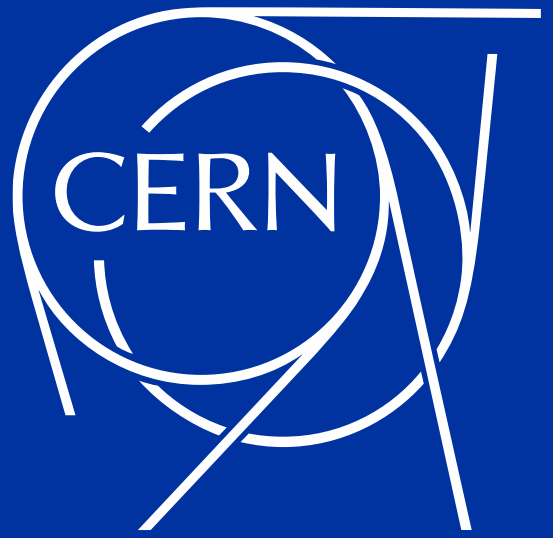

CERN

CERN LxBatch Service: HTCondor

Ben Jones IT-CD-CC

Agenda

- Batch Service
- What is HTCondor ?
- Module 1: Job submission
- Module 2: Multiple Jobs & Requirements
- Module 3: File Transfer
- Feedback / Discussion

Batch Service aka LxBatch

- **IT-CD-CC Mandate:**
  “Provide high-level compute services to the CERN Tier-0 and WLCG”
- **HTCondor: the software running our production batch service.**
- **Service used for both “grid” and “local” job submission**
- **“Local” means open to all CERN users, kerberos, shared filesystems, managed submission nodes**
- **~400k CPU cores in 2 HTCondor “pools”**
- **Over a million jobs a day**
- **Service Element: Batch Service**

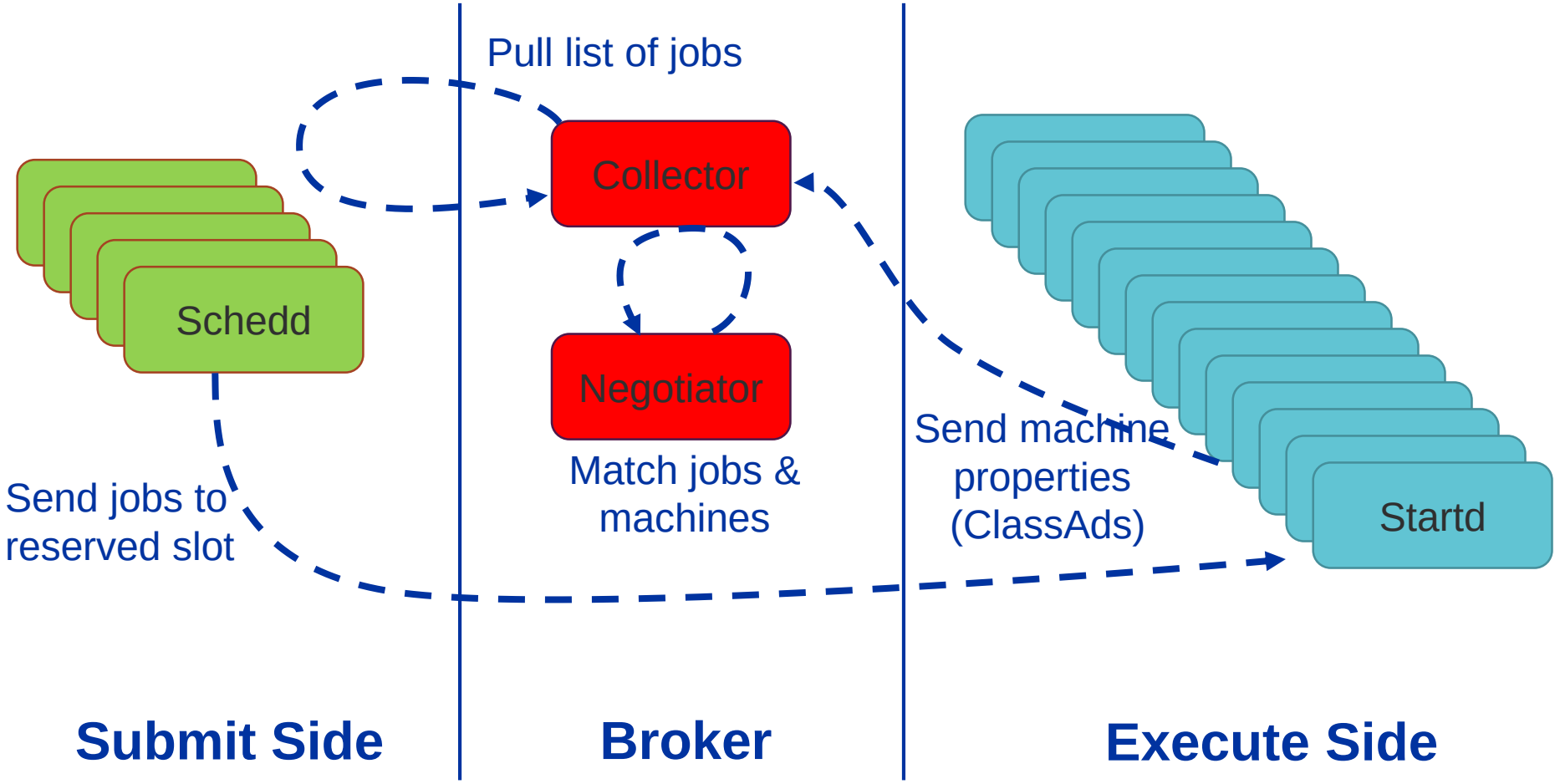
What is  HTCondor Software Suite ?

[Some content adapted from “An introduction to using HTCondor” by Christina Kock CHTC]

What is  HTCondor Software Suite ?

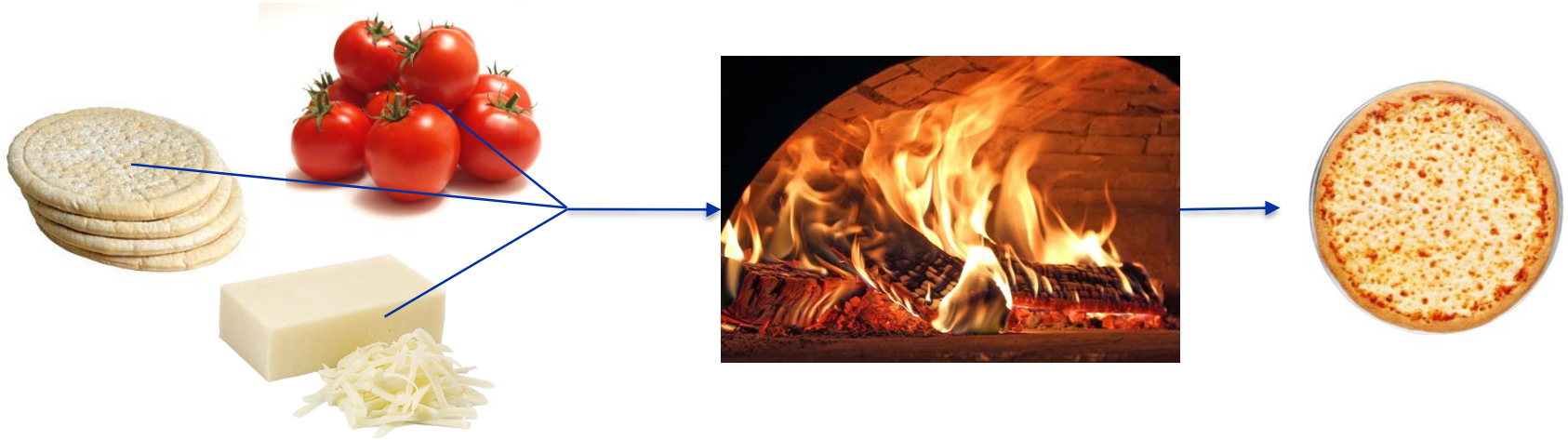
- A batch system – a system to run compute tasks or “jobs”
- Open Source batch system developed at the CHTC at the University of Wisconsin
- “High Throughput Computing” – maximise for overall throughput and utilisation
- Long history in High Energy Physics (and elsewhere)
- Used extensively in the WLCG & OSG, and the CMS global pool
- System of symmetric matching job requests to resources using ‘ClassAds’ of job requirements and machine capabilities

HTCondor Elements

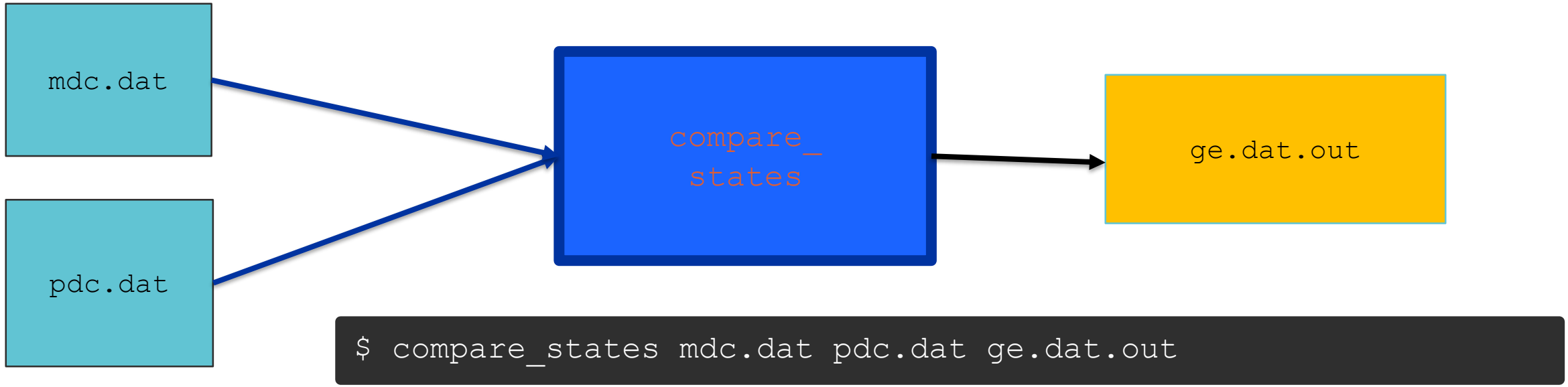


Jobs

- A single computing task is called a “job”
- Three main pieces of a job are the input, executable and output



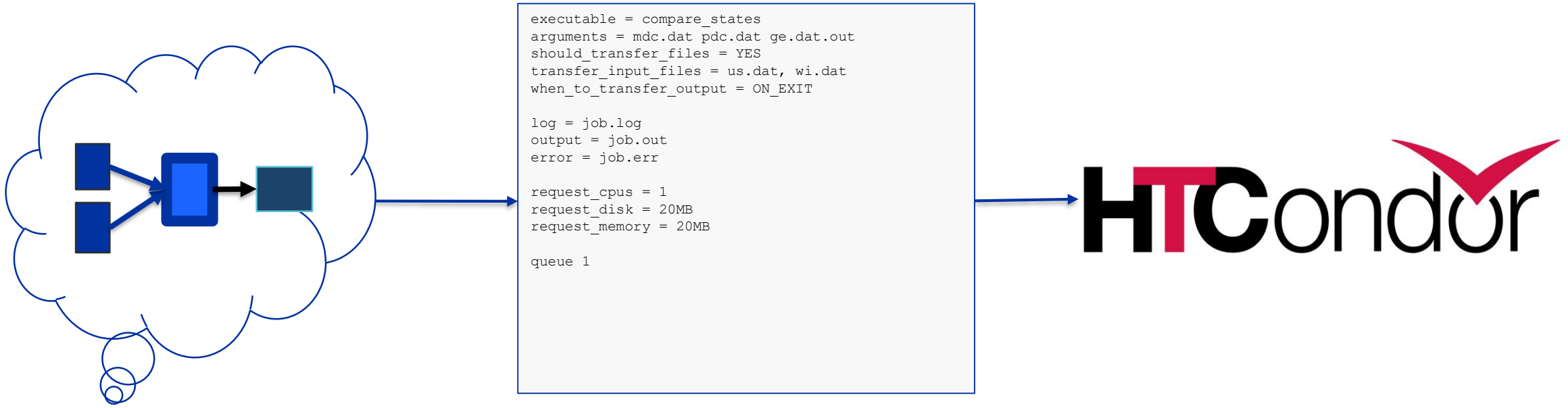
Job Example



The executable must be runnable from the command line without any interactive input

Job Translation

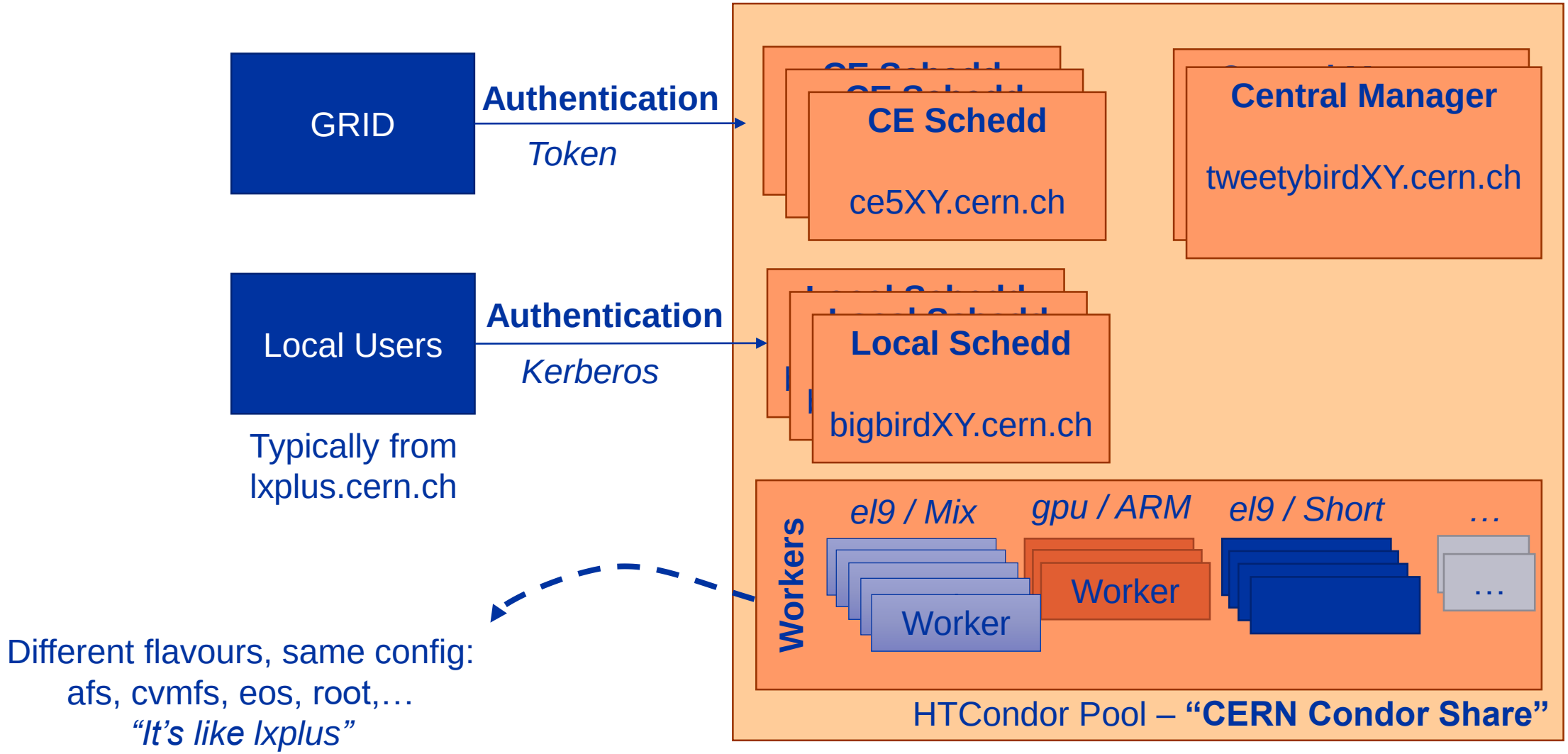
Submit file: communicates everything about your job(s) to HTCondor



```
executable = compare_states
arguments = mdc.dat pdc.dat ge.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
```


A key goal of this training is to show you how to represent your job in a submit file

CERN HTCondor Service



Environment Setup

Submit Environment (I)

- **All the exercises are to be run from lxplus:**

```
% ssh lxplus.cern.ch
[bejones@lxplus943 ~]$
```

- **HTCondor client tools present:**

```
[bejones@lxplus943 ~]$ condor_version
$CondorVersion: 23.0.2 2023-11-20 BuildID: 690948 PackageID: 23.0.2-1 $
$CondorPlatform: x86_64_AlmaLinux9 $
```

- **Condor versions vary between the submit side and the execute side, but are compatible**

Submit Environment (II)

```
[bejones@lxplus9119 ~]$ condor_q  -totals


-- Schedd: bigbird25.cern.ch : <188.185.6.50:9618?... @ 10/28/24 16:19:04
Total for query: 0 jobs; 0 completed, 0 removed, 0 idle, 0 running, 0 held, 0 suspended
Total for bejones: 0 jobs; 0 completed, 0 removed, 0 idle, 0 running, 0 held, 0 suspended
Total for all users: 35458 jobs; 1052 completed, 1455 removed, 20037 idle, 1566 running,
11348 held, 0 suspended
```

- LxPlus is preconfigured for CERN HTCondor:
  - The central manager hostnames
  - A default authentication method: kerberos
  - A predefined schedd for the user

```
[bejones@lxplus9119 ~]$ myschedd out
SCHEDD_HOST = bigbird25.cern.ch
CREDD_HOST = $(SCHEDD_HOST)
```

Submit Environment (III)

- **The training exercises are available in gitlab:**

https://gitlab.cern.ch/batch-team/htcondor-training

- **Use the “ShortTraining” folder:**

```
[bejones@lxplus9124 Dev $ git clone https://gitlab.cern.ch/batch-team/htcondor-training.git
Cloning into 'htcondor-training'...
remote: Enumerating objects: 232, done.
remote: Total 232 (delta 0), reused 0 (delta 0), pack-reused 232 (from 1)
Receiving objects: 100% (232/232), 15.28 MiB | 17.48 MiB/s, done.
Resolving deltas: 100% (92/92), done.
[bejones@lxplus9124 Dev $ cd htcondor-training/ShortTraining
[bejones@lxplus9124 ShortTraining $
```

Module I:
Basic Submission & Monitoring

Ex. 1: Job Submission: submit file

```
[bejones@lxplus943 ~]$ vi ex1.sub
```

```
universe   = vanilla
executable = ex1.sh
arguments  = "training 2024"

output     = output/ex1.out
error      = error/ex1.err
log        = log/ex1.log

queue
```

log: file created by HTCondor to log job progress

queue: keyword indicating “create a job”

universe: an HTCondor execution environment.
Vanilla is the default and should cover 90% of cases

executable: the thing you want to run – beware a script needs to be correctly formatted!

arguments: arguments are any options passed to the executable from the command line.

output/error: captures stdout & stderr

Ex.1: Job Submission: script

```
[bejones@lxplus943 ~]$ vi ex1.sub
```

```
#!/bin/sh
echo 'Date:     ' $(date)
echo 'Host:     ' $(hostname)
echo 'System:   ' $(uname -spo)
echo 'Home:     ' $HOME
echo 'Workdir:  ' $PWD
echo 'Path:     ' $PATH
echo "Program: $0"
echo "Args:    $*"
```

The shebang (`#!`) is mandatory when submitting script files in HTCondor:
"`#!/bin/sh`" "`#!/bin/bash`" "`#!/bin/env python`"
Malformed or invalid shebang silently ignored and no error reported

```
[bejones@lxplus943 ~]$ chmod +x ex1.sh
```

Ex. 1: Job Submission

```
universe   = vanilla
executable = ex1.sh
arguments  = "training 2018"

output     = output/ex1.out
error      = error/ex1.err
log        = log/ex1.log

queue
```

To submit a job or some jobs:

condor_submit <submit file>

To monitor submitted jobs:

condor_q

```
[bejones@lxplus9124 ShortTraining]$ condor_submit ex1.sub
Submitting job(s).
1 job(s) submitted to cluster 4793413.
```

```
[bejones@lxplus9124 ShortTraining]$ condor_q


-- Schedd: bigbird25.cern.ch : <188.185.6.50:9618?... @ 10/29/24 15:03:45
OWNER   BATCH_NAME     SUBMITTED   DONE   RUN    IDLE  TOTAL JOB_IDS
bejones ID: 4793413  10/29 15:03      _      _      1      1 4793413.0

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

More about condor_q

- **By default condor_q shows:**
  - User's job only
  - Jobs summarized in batches: *same cluster*, *same executable*, or *same batch name*

```
[bejones@lxplus9124 ShortTraining]$ condor_q


-- Schedd: bigbird25.cern.ch : <188.185.6.50:9618?... @ 10/29/24 15:03:45
OWNER   BATCH_NAME     SUBMITTED   DONE   RUN    IDLE  TOTAL JOB_IDS
bejones ID: 4793413  10/29 15:03      _      _      1      1 4793413.0

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

Job_ID = ClusterId.ProcID

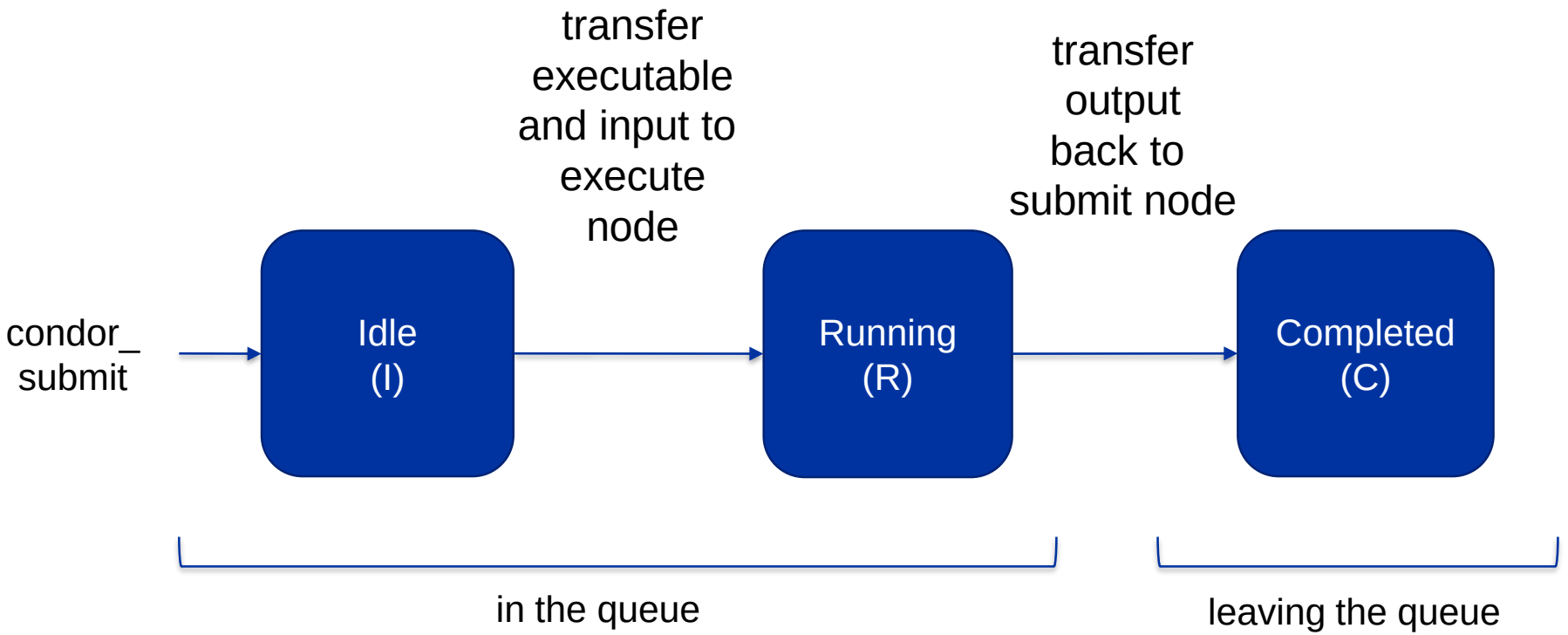
More about condor_q

- **To see individual job information use:**

```
condor_q -nobatch
```

```
[bejones@lxplus9121 condor]$ condor_q -nobatch


-- Schedd: bigbird25.cern.ch : <188.185.6.50:9618?... @ 11/01/24 09:26:59
 ID         OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
4899199.0   bejones         11/1  09:26   0+00:00:00 I  0    0.0 hello.sh
4899199.1   bejones         11/1  09:26   0+00:00:00 I  0    0.0 hello.sh
4899199.2   bejones         11/1  09:26   0+00:00:00 I  0    0.0 hello.sh

Total for query: 3 jobs; 0 completed, 0 removed, 3 idle, 0 running, 0 held, 0 suspended
```

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

Log File

```
000 (168.000.000) 11/20 11:34:25 Job submitted from host:
<137.138.120.138:9618?addrs=137.138.120.138-9618&noUDP&sock=1069_d2d4_3>
...
001 (168.000.000) 11/20 11:37:26 Job executing on host:
<188.185.217.222:9618?addrs=188.185.217.222-9618+[--1]-9618&noUDP&sock=3285_211b_3>
...
006 (168.000.000) 11/20 11:37:30 Image size of job updated: 15
        0  -  MemoryUsage of job (MB)
        0  -  ResidentSetSize of job (KB)
...
005 (168.000.000) 11/20 11:37:30 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
        19  -  Run Bytes Sent By Job
        15768  -  Run Bytes Received By Job
        19  -  Total Bytes Sent By Job
        15768  -  Total Bytes Received By Job
        Partitionable Resources :    Usage  Request Allocated
           Cpus                 :                1         1
           Disk (KB)            :       31       15   1841176
           Memory (MB)          :        0     2000      2000
...
```

Ex1: Job Submission: result

```
[bejones@lxplus] $ ./ex1.sh  training 2024
```

```
Date:     Fri Nov 1 09:45:36 AM CET 2024
Host:     lxplus9121.cern.ch
System:   Linux x86_64 GNU/Linux
Home:     /afs/cern.ch/user/b/bejones
Workdir:  /afs/cern.ch/user/b/bejones/Dev/htcondor-training/ShortTraining
Path:
/opt/conda/bin:/usr/sue/bin:/usr/share/Modules/bin:/usr/condabin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/opt/puppetlabs/bin:/afs/cern.ch/user/b/bejones/bin
Program: ./ex1.sh
Args:    training 2024
```

```
[bejones@lxplus] $ cat output/ex1.out
```

```
Date:     Tue Oct 29 03:05:22 PM CET 2024
Host:     b9g37p4324.cern.ch
System:   Linux x86_64 GNU/Linux
Home:     /afs/cern.ch/user/b/bejones
Workdir:  /pool/condor/dir_1770839
Path:
/usr/sue/bin:/usr/share/Modules/bin:/usr/condabin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/opt/puppetlabs/bin
Program: condor_exec.exe
Args:    training 2024
```

Jobs start with a basic environment configuration:
Env files such as ~/.bashrc or ~/.bash_profile are not loaded
You should explicitly define the environment of your job

Ex1: Environment

```
[bejones@lxplus] $ vi ex2.sub ex2.sh
```

```
universe    = vanilla
executable  = ex2.sh
arguments   = "training 2024"

Environment = PATH=$ENV(PATH)

output      = output/ex2.out
error       = error/ex2.err
log         = log/ex2.log

+TrainingJob=True

queue
```

```
#!/bin/sh

export PATH=$PATH:$HOME/magic

echo 'Date:    ' $(date)
echo 'Host:    ' $(hostname)
echo 'System:  ' $(uname -spo)

echo 'Home:    ' $HOME
echo 'Workdir: ' $PWD
echo 'Path:    ' $PATH

echo "Program: $0"
echo "Args:    $*"
```

Submit files accept the keyword “`getenv=True`” that copies the entire environment.
We don’t recommend its usage, and we encourage users to identify correctly the job dependencies regarding environment variables.

The Central Manager

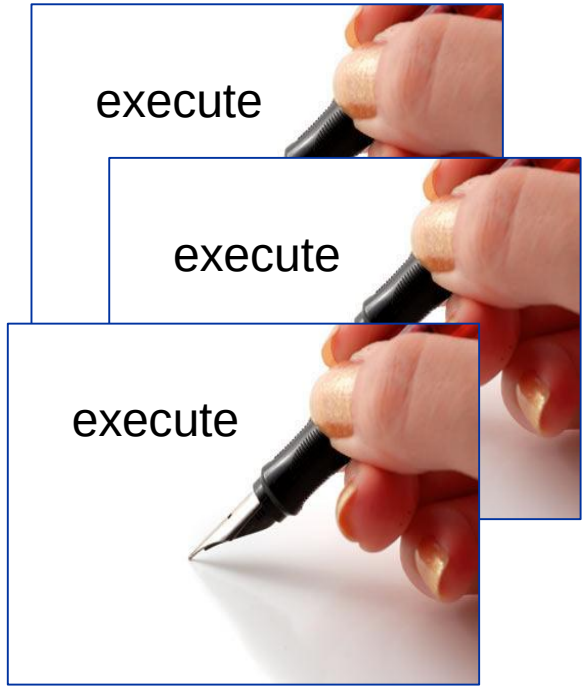
Class Ads & Matchmaking

The Central Manager

HTCondor matches jobs with computers via a “central manager”

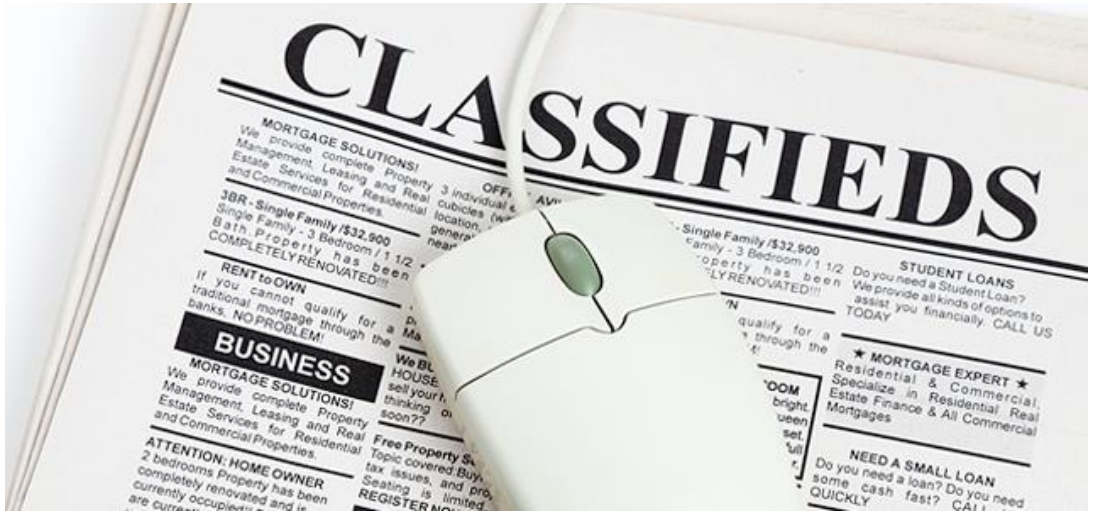
HTCondor

central manager



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`
- `value` **can be Boolean, number or string**

Job Class Ads

```
executable = exe
Arguments = “x y z”

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


queue 1
```

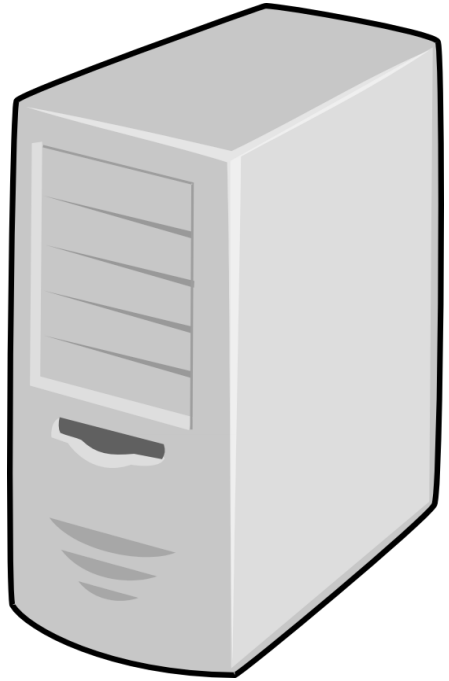
+

HTCondor configuration*

=

```
RequestCpus = 1
Err = “job.err"
WhenToTransferOutput = "ON_EXIT"
TargetType = "Machine"
Cmd =
“/afs/cern.ch/user/f/fernandl/condor/exe“
Arguments = “x y z”
JobUniverse = 5
Iwd = “/afs/cern.ch/user/f/fernandl/condor"
RequestDisk = 20480
NumJobStarts = 0
WantRemoteIO = true
OnExitRemove = true
MyType = "Job"
Out = "job.out"
UserLog =
“/afs/cern.ch/user/f/fernandl/condor/job.log"
RequestMemory = 20
...
...
```

Machine Class Ads



+

HTCondor configuration*

=

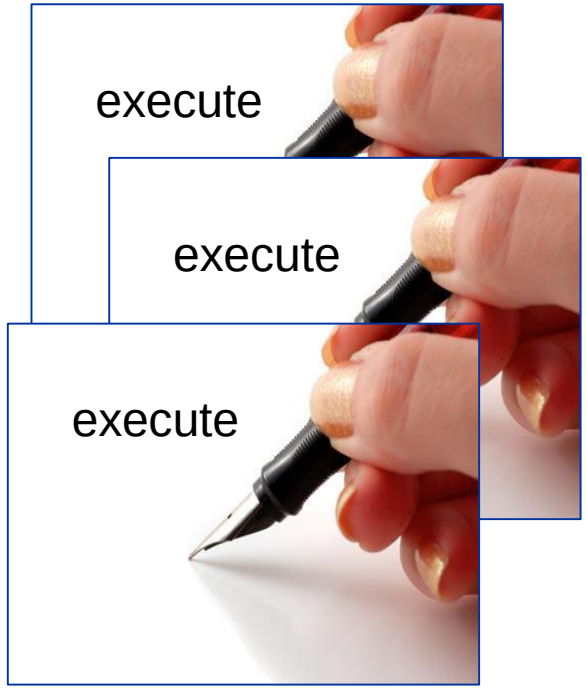
```
HasFileTransfer = true
DynamicSlot = true
TotalSlotDisk = 4300218.0
TargetType = "Job"
TotalSlotMemory = 2048
Mips = 17902
Memory = 2048
UtsnameSysname = "Linux"
MAX_PREEMPT = ( 3600 * 72 )
Requirements = ( START ) && (
IsValidCheckpointPlatform ) && (
WithinResourceLimits )
OpSysMajorVer = 6
TotalMemory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

...
```

Job Matching

On a regular basis – the “negotiation cycle” – the central manager reviews Job and Machine Class Ads and matches jobs to computers

HTCondor

central manager



Class Ads for People

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

Finding Job Attributes

Use the “long” option for condor_q: `condor_q -l <jobid>`

```
$ condor_q -l 4910855.0
AccountingGroup = "group_u_DTEAM.grid_DTEAM.bejones"
Arguments = "training 2024"
AutoClusterId = 45671
CernFlagCentOS7 = false
CernFlagComplexOpSysAndVer = false
CernFlagUnsupportedOpSysRequirement = false
CernUnparsedRequirements = "(TARGET.Arch == \"X86_64\") && (TARGET.OpSys == \"LINUX\") && (TARGET.Disk >=
RequestDisk) && (TARGET.Memory >= RequestMemory) && ((TARGET.FileSystemDomain == MY.FileSystemDomain) ||
(TARGET.HasFileTransfer))"
ChargeGroup = "dteam"
ChargeGroupType = "experiment"
ChargeRole = "DTEAM Grid"
ClusterId = 4910855
Cmd = "/afs/cern.ch/user/b/bejones/public/Dev/htcondor-training/ShortTraining/ex2.sh”
. . .
```

Use the “af” option for condor_q for one key: `condor_q <jobid> -af <key>`

```
$ condor_q  4910855.0 -af Cmd
/afs/cern.ch/user/b/bejones/public/Dev/htcondor-training/ShortTraining/ex2.sh
```

Ex. 3: Input Files

```
[bejones@lxplus943 ~]$ vi ex3.sub ex3.py
```

```
universe    = vanilla
executable  = ex3.sh
arguments   = "words.txt"

should_transfer_files = YES
transfer_input_files = words.txt
when_to_transfer_output = ON_EXIT

output      = output/ex3.out
error       = error/ex3.err
log         = log/ex3.log

queue
```

```
#!/usr/bin/env python3

import os
import sys

if len(sys.argv) != 2:
    print(f'Usage: %s DATA
{os.path.basename(sys.argv[0])}')
    sys.exit(1)
input_filename = sys.argv[1]

words = {}

my_file = open(input_filename, 'r',
encoding='latin-1’)

…
```

Ex. 3: Input Files

```
universe    = vanilla
executable  = ex3.py
arguments   = "words.txt"

should_transfer_files = YES
transfer_input_files  = words.txt
when_to_transfer_output = ON_EXIT

output      = output/ex3.out
error       = error/ex3.err
log         = log/ex3.log

queue
```

`should_transfer_files`
`transfer_input_files`

indicate to HTCondor what is the required input

`when_to_transfer_output`

HTCondor will transfer back all new and changed files (usually output) from the job

```
$ condor_q 4911831.0 -af TransferInput Iwd Arguments
words.txt /afs/cern.ch/user/b/bejones/public/Dev/htcondor-training/ShortTraining words.txt
```

Resource Request

- **Jobs use a part of the computer, not the whole thing**
- **Important to size job requirements appropriately: memory, cpus and disk.**
- **CERN HTCondor defaults:**
  - 1 CPU
  - 2 Gb ram
  - 20 GiB disk
- **Size for what you need!**
  - Too little: your job might try to use more, and may be killed by the system
  - Use too much: your job will be inefficient and will waste resources

Time to start running

- **Job don’t start immediately after submission**
- **Many factors involved:**
  - **Negotiation Cycle**: the central managers loop through a matchmaking cycle about every 2-5 minutes.
  - **User priority**: users priority is dynamic, and recalculate according to usage and the quota of the groups to which they belong
  - **Availabilty of resources**: Machines matching your job requirements might be busy.

Module II:
Multiple Jobs & Job Lifecycle

Ex. 4: Multiple Jobs (queue)

```
[bejones@lxplus943 ~]$ vi ex4.sub
```

```
universe   = vanilla
executable = ex4.sh
arguments  = $(ClusterId) $(JobId)
output     = output/$(ClusterId).$(ProcId).out
error      = error/$(ClusterId).$(ProcId).err
log        = log/$(ClusterId).log

queue  5
```

Pre-defined-macros: we can use the `$ClusterId` and `$ProcId` vars in order to provide unique values to the job files

queue: this keyword controls how many instances of the job are submitted (default: 1). It also supports more sophisticated patterns.

Ex. 5: Multiple Jobs (queue)

```
[bejones@lxplus943 ~]$ vi ex5.sub
```

```
universe   = vanilla
executable = $(filename)
output     = output/$(ClusterId).$(ProcId).out
error      = error/$(ClusterId).$(ProcId).err
log        = log/$(ClusterId).log

queue filename matching files ex5/*.sh
```

The usage of regular expressions in queue allows us to submit more than one different job

The resulting jobs point to different executables, but they will belong to the same `ClusterId` with different `ProcIds`

Queue Statement Comparison

Multiple queue statements	Not recommended, and deprecated in future versions.
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.

Requirements

- In HTCondor we can specify requirements in the job submit file, which will end up in the job's ClassAd. This can include things like the *name of the execute machine*, the *desired operating system*, etc.

- We can modify the requirements based on our preferences by just adding the following line in the submit file:

```
requirements = <ClassAd expression>
```

- We can use comparision operators like <, >, <=, =>, ==, or use the special operators =?= and =!= which better handle cases where variables are undefined.

- It's a good idea to test that your requirement will match machines! The same requirement can be used as a "--constraint" to the `condor_status` command.

```
condor_status -const 'OpSysAndVer =?= "AlmaLinux9"'
```

Ex. 6: Requirements (resources)

```
[bejones@lxplus943 ~]$ vi ex6.sub
```

```
universe       = vanilla
executable     = ex6.sh
output         = output/$(ClusterId).$(ProcId).out
error          = error/$(ClusterId).$(ProcId).err
log            = log/$(ClusterId).log
request_cpus = 2
queue
```

request_cpus: this example shows how to specify the number of CPUs to request for the job.

We scale CPU/memory to the WLCG standard of 2Gb / CPU core. This request will automatically ask for 4Gb of memory.

Ex. 7: Requirements (operating system)

```
[bejones@lxplus943 ~]$ vi ex7.sub
```

```
universe      = vanilla
executable    = ex6.sh
output        = output/$(ClusterId).$(ProcId).out
error         = error/$(ClusterId).$(ProcId).err
log           = log/$(ClusterId).log
requirements = (OpSysAndVer =?= "AlmaLinux9")
queue
```

requirements: this example shows how to use requirements to select an attribute of the machines, in this case Operating System.

Note at the time of writing, we have 100% the same operating system, but this will not always be the case. Try `condor_status -compact -af OpSysAndVer | sort | uniq -c`

Ex. 8: Requirements (MaxRuntime)

```
[bejones@lxplus943 ~]$ vi ex8.sub
```

```
universe     = vanilla
executable   = ex8.sh
output       = output/$(ClusterId).$(ProcId).out
error        = error/$(ClusterId).$(ProcId).err
log          = log/$(ClusterId).log
+MaxRuntime = 120

queue
```

MaxRuntime: maximum number of seconds that your job will be allowed to run (wall time)

MaxRuntime should be indicative of how long jobs should take to run. It doesn't need to be very accurate, but helps the service be more efficient, and prioritise shorter jobs.

CERNism: JobFlavour

JobFlavour is a macro to select a pre-defined MaxRuntime. The default, if neither MaxRuntime nor JobFlavour are defined, is espresso

```
universe     = vanilla
executable   = training.sh
output       = output/$(ClusterId).$(ProcId).out
error        = error/$(ClusterId).$(ProcId).err
Log          = log/$(ClusterId).log
+JobFlavour = "microcentury"

queue
```

```
espresso     = 20 min
microcentury = 1  hour
longlunch    = 2  hours
workday      = 8  hours
tomorrow     = 1  day
testmatch    = 3  days
nextweek     = 1  week
```

Note if the job exceeds MaxRuntime, it will be removed by the system

Debug tools: condor_status

- **condor_status can be used to query machines and other infrastructure**

```
$ condor_status -avail          # show available machines
$ condor_status -schedd         # show schedds
$ condor_status <hostname>      # show worker nodes summary
$ condor_status -l <hostname>   # show worker node ClassAd
```

- **It also supports filtering by ClassAd:**

```
$ condor_status -const 'Arch =?= "aarch64"' -compact -limit 2
Machine            Platform           Slots Cpus Gpus  TotalGb FreCpu  FreeGb  CpuLoad ST Jobs/Min MaxSlotGb

b9g40p6295.cern.ch aarch64/AlmaLinux9    1    80        375.00     72   359.38    0.00 **     0.10     15.62
b9g40p7942.cern.ch aarch64/AlmaLinux9    0    80        376.46     80   376.46    0.00 Ui     0.00 *

                   Total Owner Claimed Unclaimed Matched Preempting  Drain Backfill BkIdle

aarch64/AlmaLinux9     3     0       1         2       0          0      0        0      0

               Total     3     0       1         2       0          0      0        0      0
```

Debug tools: `condor_ssh_to_job`

- **Creates an ssh session to a running job.**

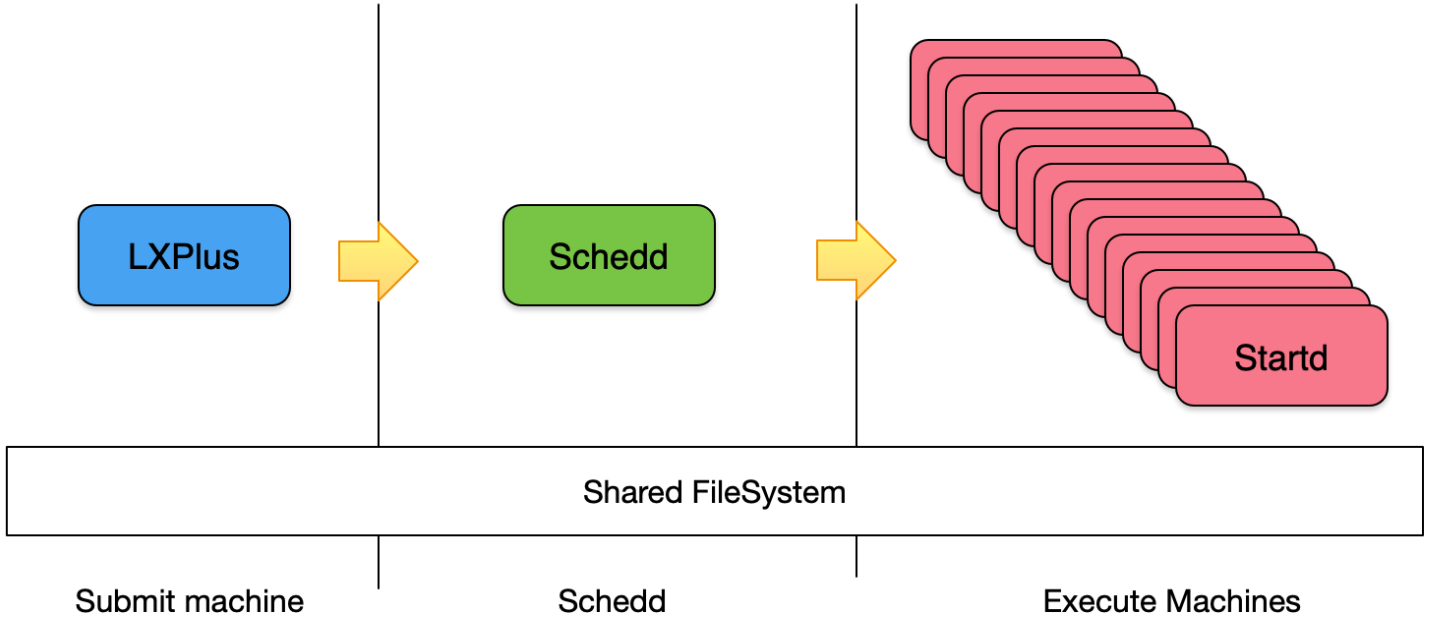
```
$ condor_ssh_to_job <jobid>
$ condor_ssh_to_job -auto-retry <jobid>
```

- **This will get us to the same machine, in the working directory of the job, and we can see the stdout/err etc**
  - (though if you just want to see those files, you can use `condor_tail`)

```
$ condor_ssh_to_job 5024157.0
Welcome to slot1_1@b9g37p4425.cern.ch!
Your condor job is running with pid(s) 653021.
[bejones@b9g37p4425 dir_652931]$ ls
bejones.cc  _condor_stderr  _condor_stdout  long.sh  srv  tmp  var
```

Module III: File Transfer

File Transfer



condor_submit files are shared on filesystem between submit and schedd machines. Then file transferred from schedd to startd (and back)

condor_submit -spool files are transferred from submit to schedd, then from schedd to startd (and back)

- A job will need input and output data.
- There are several ways to get data in or out of the batch system, so we need to know a little about the **trade offs.**
- Do you want to use a shared filesystem? Do you want to have condor transfer data for you? Should you input or output in the job payload / executable itself?

Adding Input Files

- **In order to add input files, we just need to use the `transfer_input_files` directive in our submit file**
- **It's a list of files to take from the working directory to send to the job sandbox**
- **This example will produce one output file "merge.out"**

```
executable = merge.sh
arguments = a.txt b.txt merge.out

transfer_input_files = a.txt, b.txt

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

+JobFlavour = "longlunch"
queue 1
```

Transferring output back

- **By default condor with transfer everything in your "sandbox" back to your submit directory**
- **To only transfer the file(s) back you need, use** `transfer_output_files`
- **Adding to** `transfer_output_files` **means those listed files will be available to** `condor_tail`
- **If you don't need anything back – perhaps because the job transfers anything it needs – use an empty string ie ""**

```
executable = merge.sh
arguments = a.txt b.txt merge.out

transfer_input_files = a.txt, b.txt
transfer_output_files = merge.out

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

+JobFlavour = "longlunch"
queue 1
```

Transfer to EOS using URL

- **The HTCondor schedds do not have /eos/ available on the filesystem, but you can transfer to/from EOS using a root:// url**
- **The example here uses output_destination to send the whole output sandbox to this URL**
  - This means everything except the ”log” which is not sent to the worker node
- **This is better than having the job script transfer because there is error checking**

```
executable = merge.sh
arguments = a.txt b.txt merge.out

transfer_input_files = a.txt, b.txt
output_destination =
root://eosuser.cern.ch//eos/user/b/bejones/condor/$(ClusterId)/

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

+JobFlavour = “longlunch”
queue 1
```

Input files via URL also possible

```
executable = merge.sh
arguments = a.txt b.txt merge.out

transfer_input_files =
root://eosuser.cern.ch//eos/user/b/bejones/condor/input/a.txt
output_destination =
root://eosuser.cern.ch//eos/user/b/bejones/condor/$(ClusterId)/

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

+JobFlavour = “longlunch”
queue 1
```

Important Considerations

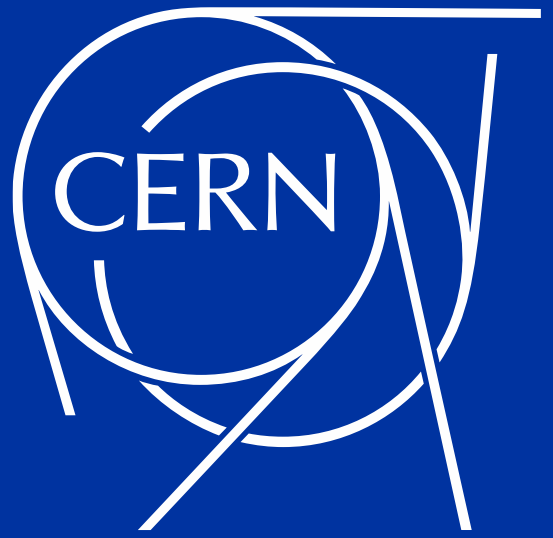
- **Even when using a share filesystem, input files and exectuables are transferred to a scratch space on the workers; the “sandbox”**
- **When writing jobs, remember the impact on the filesystem! The most efficient use of network filesystems is typically to write once at the end of a job.**
- **We provision ~30GiB of sandbox disk per CPU**
- **When using the standard htcondor file transfer mechanism, we limit to 1GB per job**
  - There’s no such limit using the URL based method
- **The job itself has access to filesystems and file transfer protocols, so can do some i/o itself**

condor_submit -spool

- **You may not want condor to create files in your home directory**
  - Particularly if you are submitting 10s of 1000s of jobs
- `condor_submit -spool` **transfers all files to the schedd**
- **Important notes:**
  - This makes the system asynchronous – to get any files back directly from htcondor you need to run condor_transfer_data
  - The spool on the Schedd is limited! Make sure transfer_output_files limits the output to what you absolutely require
- **Best practice:**
  - Use the URL file transfer plugin method
  - Alternatively set transfer_output_files to “” and only look at stdout/err in case of errors

Note on AFS & EOS

- **Share filesystem is used a lot for batch jobs**
- **Current best practices:**
  - AFS, EOS FUSE, EOS via xrootd/xrdcp are all available on the worker node
  - Between the submit node (ie lxplus) and the schedd, only the AFS home directory is available
  - No exe, log, stdout, err in /eos in the submit file
  - With all network filesytesm, it's best to write at the end of the job, not constant I/O whilst the job is running
- **The schedd is sensitive to filesystem problems – it has to write on behalf of the user. This means that breakage to EOS FUSE takes down the whole schedd**
- **Using shared filesystems of any type in the job can become problematic as the jobs scale to the 1000s of cores available in the batch system**


CERN

Links…

- LxBatch User Guide: https://batchdocs.web.cern.ch
- How to benchmark jobs: https://batchdocs.web.cern.ch/local/benchmark.html
- Submit support tickets: https://cern.service-now.com/service-portal?id=functional_element&name=LXBATCH
- Mattermost channel: https://mattermost.web.cern.ch/it-dep/channels/batchers