Science & Technology Facilities Council

Introduction to HTCondor

Andrew Lahiff

Distributed Computing Infrastructure Group,

Rutherford Appleton Laboratory

GridKa School 2016, Karlsruhe

Outline

- Introduction
- HTCondor as a batch system
- Expanding a HTCondor pool across the grid & clouds
- LHC usage of HTCondor

HTCondor

- Open source software for distributed high throughput computing
- More specifically, it is:
  - a job scheduler
  - a resource manager
  - a workflow management system

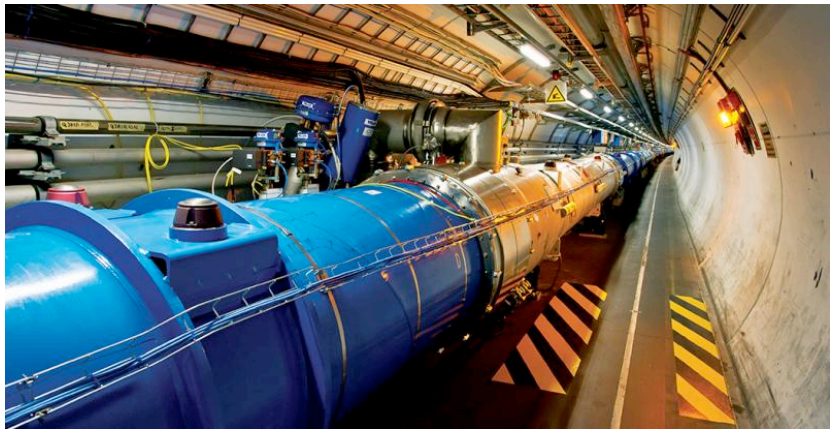
HTC vs HPC

- High Performance Computing (HPC)
  - Run parallel software over many processes simultaneously
  - Generally small numbers of jobs but large amounts of computing power required
  - Short periods of time
- High Throughput Computing (HTC)
  - Running multiple independent instances of software on multiple processors at the same time
  - Long time periods
  - Generally runs on clusters of commodity hardware

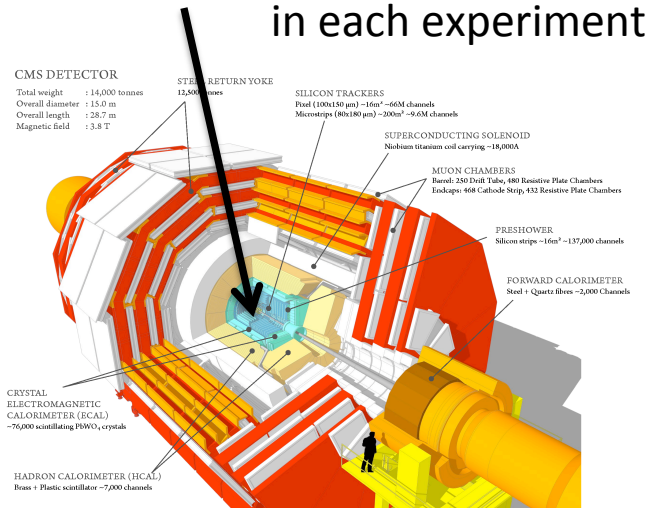
An example of HTC

- Consider the CERN Large Hadron Collider

Protons accelerated to very high energies



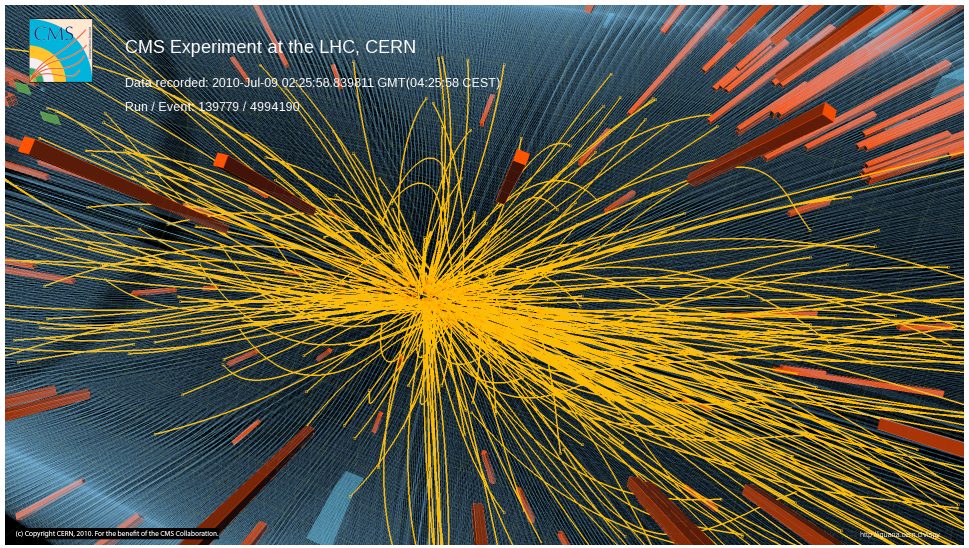
Millions of protons collide every second in each experiment



Interesting collisions are recorded: these are called “events”

An example of HTC

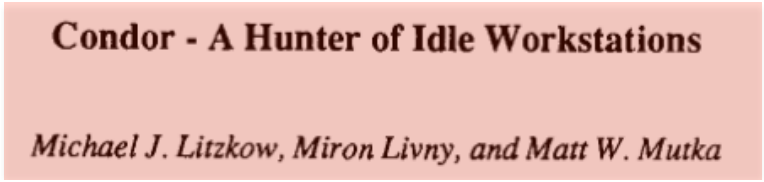
- Many billions of independent events are stored
  - Understanding what happened in each event can take minutes of CPU time



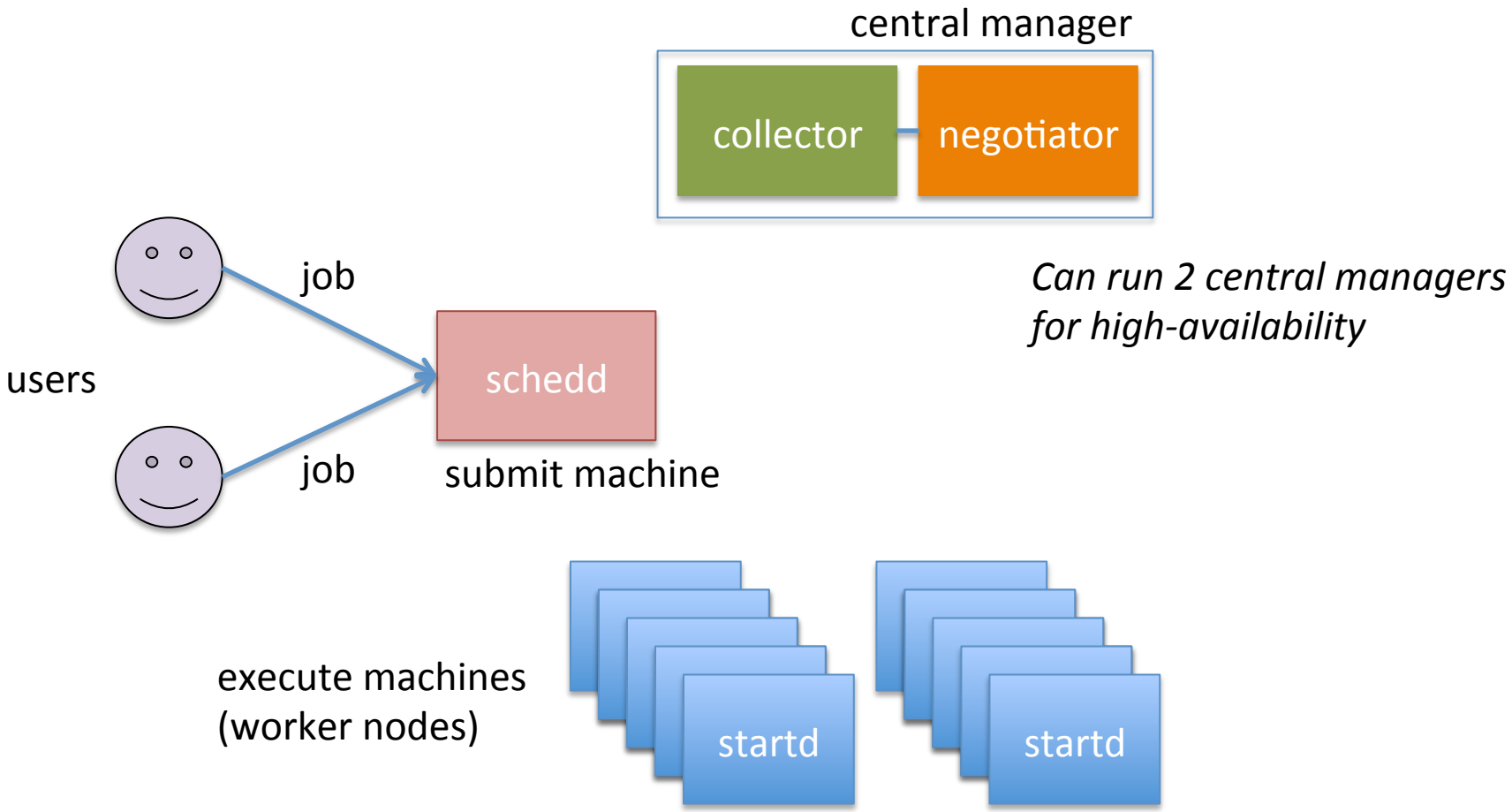
Billions of events where each takes minutes to process = HTC

History

- HTCondor was developed at the University of Wisconsin Madison in the 1980's
- Motivation
  - many workstations have idle cycles wasted
  - some users need more processing power than their workstation can provide
- Designed to maximize the utilization of desktop workstations
  - identify idle workstations
  - run background jobs on them without impacting the owners of the workstations
  - when workstation owner resumes activity, remote jobs are checkpointed & moved to another workstation



Architecture of a HTCondor pool



ClassAds

- Lists of information stored about each job and worker node (& also all daemons)
  - collector stores details about worker nodes
  - schedd stores details about each job
- This information is stored as a "ClassAd" which have the format

  AttributeName = Value

  or

  AttributeName = Expression

ClassAds are an important & powerful feature of HTCondor

ClassAds

- ClassAds can state facts
  - the job's executable is "cmsRun"
  - the machine's current load is 2.3
- ClassAds can state requirements
  - this job requires a machine running SL6
  - this machine will only run jobs needing 8 CPUs
- ClassAds can state preferences
  - this job prefers to run on a machine with low CPU load
  - this machine prefers to run jobs from the physics department

Job ClassAd

- Example (extract):

```
Cmd = "/home/tier1/alahiff/tutorial/basic.sh"
Arguments = ""
Err = "job.92537.0.err"
Out = "job.92537.0.out"
UserLog = "/home/tier1/alahiff/tutorial/job.92537.0.log"
RequestCpus = 1
RequestMemory = 500
TransferInput = "input.dat,input1.dat"
JobUniverse = 5
Owner = "alahiff"
JobDescription = "testjob1"
WhenToTransferOutput = "ON_EXIT"
ShouldTransferFiles = "YES"
...
```

Machine ClassAd

- Example (extract):

```
CondorVersion = "$CondorVersion: 8.4.6 Apr 20 2016 BuildID: 364106 $"
MyType = "Machine"
OpSys = "LINUX"
OpSysAndVer = "SL6"
TotalMemory = 174150
TotalCpus = 32.0
TotalDisk = 3388883196
Name = "slot1@lcg1986.gridpp.rl.ac.uk"
Machine = "lcg1986.gridpp.rl.ac.uk"
Start = true
TotalLoadAvg = 23.85
Activity = "Idle"
EnteredCurrentState = 1470818572
...
```

Daemon ClassAd

- Example from a schedd (extract):

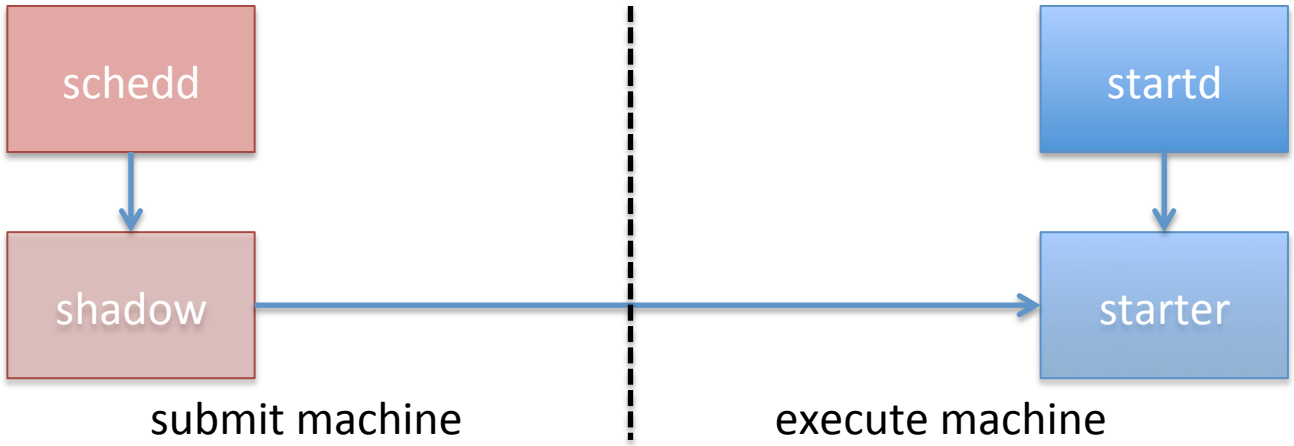
```
JobsExecFailed = 0
RecentJobsExitedNormally = 258
TotalHeldJobs = 0
JobsRestartReconnectsFailed = 5
FileTransferDownloadBytesPerSecond_1d = 519425.6784118257
JobsCompleted = 639744
RecentJobsShouldHold = 0
RecentStatsLifetime = 1200
FileTransferMBWaitingToUpload = 0.0
RecentJobsSubmitted = 195
MyType = "Scheduler"
TotalIdleJobs = 219
MyCurrentTime = 1472380803
TotalFlockedJobs = 0
JobsExitedNormally = 639744
Machine = "arc-ce02.gridpp.rl.ac.uk"
RecentJobsCompleted = 258
JobsStarted = 648016
RecentJobsKilled = 8
MaxJobsRunning = 10000
ShadowsStarted = 499631
...
```

Match making

- The negotiator matches machines and jobs
  - Obtains list of startds from collector
  - Obtains list of jobs from schedd(s)
  - Generates a list of matches
  - Informs the relevant schedd(s)
- Also takes into account priorities: users, groups, jobs
- Match-making involves two sides
  - jobs can specify requirements & preferences
  - machines can specify requirements & preferences

Match making

- When a job has been matched to a machine
  - schedd spawns a shadow
    - Shadow takes ownership of the running job
  - startd spawns a starter
    - Starter takes ownership of the claimed slot



A shadow & starter are created for every running job

Scalability

- A condor_shadow process for every running job – does this scale?
  - Lots of effort in recent years in reducing the memory footprint of the shadow process
  - Tests have demonstrated over 40,000 running jobs on a single schedd
  - Can run multiple schedds
    - Also useful for redundancy
- Collector scaling
  - Can run multiple secondary collectors
    - Startds send ClassAds to the secondary collectors
    - Secondary collectors forward the updates to the top-level collector

Running jobs

- Simple example

```
-bash-4.1$ cat job.sub
universe = vanilla
executable = /bin/sleep
arguments = 300
queue 5
```

Submit description file

```
-bash-4.1$ condor_submit job.sub
Submitting job(s).....
5 job(s) submitted to cluster 92689.


-bash-4.1$ condor_q 92689

-- Schedd: lcgui03.gridpp.rl.ac.uk : <130.246.180.41:9618?...
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
92689.0   alahiff         8/28 13:12   0+00:00:17 R  0   0.0  sleep 300
92689.1   alahiff         8/28 13:12   0+00:00:17 R  0   0.0  sleep 300
92689.2   alahiff         8/28 13:12   0+00:00:16 R  0   0.0  sleep 300
92689.3   alahiff         8/28 13:12   0+00:00:17 R  0   0.0  sleep 300
92689.4   alahiff         8/28 13:12   0+00:00:16 R  0   0.0  sleep 300

5 jobs; 0 completed, 0 removed, 0 idle, 5 running, 0 held, 0 suspended
```

Running jobs

- Simple example

```
-bash-4.1$ cat job.sub
universe = vanilla
executable = /bin/sleep
arguments = 300
queue 5
```

```
-bash-4.1$ condor_submit job.sub
Submitting job(s).....
5 job(s) submitted to cluster 92689.
```

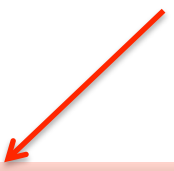
Submit job

```
-bash-4.1$ condor_q 92689

-- Schedd: lcgui03.gridpp.rl.ac.uk : <130.246.180.41:9618?...
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
92689.0   alahiff         8/28 13:12   0+00:00:17 R  0   0.0  sleep 300
92689.1   alahiff         8/28 13:12   0+00:00:17 R  0   0.0  sleep 300
92689.2   alahiff         8/28 13:12   0+00:00:16 R  0   0.0  sleep 300
92689.3   alahiff         8/28 13:12   0+00:00:17 R  0   0.0  sleep 300
92689.4   alahiff         8/28 13:12   0+00:00:16 R  0   0.0  sleep 300

5 jobs; 0 completed, 0 removed, 0 idle, 5 running, 0 held, 0 suspended
```

Running jobs

- Simple example

```
-bash-4.1$ cat job.sub
universe = vanilla
executable = /bin/sleep
arguments = 300
queue 5


-bash-4.1$ condor_submit job.sub
Submitting job(s).....
5 job(s) submitted to cluster 92689.
```




```
-bash-4.1$ condor_q 92689

-- Schedd: lcgui03.gridpp.rl.ac.uk : <130.246.180.41:9618?...
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
92689.0   alahiff         8/28 13:12   0+00:00:17 R  0   0.0  sleep 300
92689.1   alahiff         8/28 13:12   0+00:00:17 R  0   0.0  sleep 300
92689.2   alahiff         8/28 13:12   0+00:00:16 R  0   0.0  sleep 300
92689.3   alahiff         8/28 13:12   0+00:00:17 R  0   0.0  sleep 300
92689.4   alahiff         8/28 13:12   0+00:00:16 R  0   0.0  sleep 300

5 jobs; 0 completed, 0 removed, 0 idle, 5 running, 0 held, 0 suspended
```

Queues

- Traditional batch systems have the concept of "queues"
  - when you submit a job you need to specify a queue
  - queues for different job lengths, different memory requirements, serial or parallel jobs, ...
- With HTCondor you don't do this
  - jobs should specify what resources they require
    - CPUs, memory, disk, GPUs, ...
  - can configure jobs to be killed after specified CPU & wall time limits

Running jobs

- A more advanced submit description file

```
universe = vanilla
executable = run.sh

request_cpus = 4
request_memory = 8000

transfer_input_files = input.dat

output = job.$(cluster).$(process).out
error = job.$(cluster).$(process).err

should_transfer_files = YES
when_to_transfer_output = ON_EXIT

requirements = Machine =?= "lcg1111.gridpp.rl.ac.uk"

periodic_remove = JobStatus == 2 && (CurrentTime - EnteredCurrentStatus > 24*60*60)

queue 1
```

Running jobs

- A more advanced submit description file

```
universe = vanilla
executable = run.sh

request_cpus = 4
request_memory = 8000

transfer_input_files = input.dat

output = job.$(cluster).$(process).out
error = job.$(cluster).$(process).err

should_transfer_files = YES
when_to_transfer_output = ON_EXIT

requirements = Machine =?= "lcg1111.gridpp.rl.ac.uk"

periodic_remove = JobStatus == 2 && (CurrentTime - EnteredCurrentStatus > 24*60*60)

queue 1
```

Job requires 4 CPUs & 8 GB memory

Running jobs

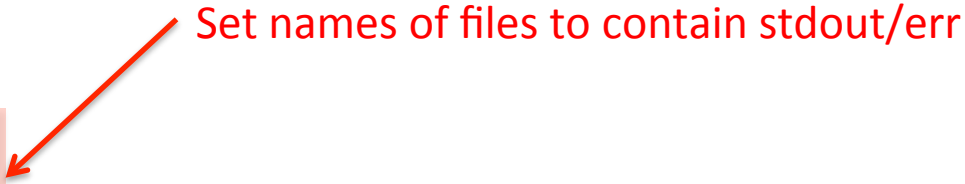
- A more advanced submit description file

```
universe = vanilla
executable = run.sh

request_cpus = 4
request_memory = 8000

transfer_input_files = input.dat

output = job.$(cluster).$(process).out
error = job.$(cluster).$(process).err

should_transfer_files = YES
when_to_transfer_output = ON_EXIT

requirements = Machine =?= "lcg1111.gridpp.rl.ac.uk"

periodic_remove = JobStatus == 2 && (CurrentTime - EnteredCurrentStatus > 24*60*60)

queue 1
```

Running jobs

- A more advanced submit description file

```
universe = vanilla
executable = run.sh

request_cpus = 4
request_memory = 8000

transfer_input_files = input.dat

output = job.$(cluster).$(process).out
error = job.$(cluster).$(process).err

should_transfer_files = YES
when_to_transfer_output = ON_EXIT

requirements = Machine =?= "lcg1111.gridpp.rl.ac.uk"

periodic_remove = JobStatus == 2 && (CurrentTime - EnteredCurrentStatus > 24*60*60)

queue 1
```



Running jobs

- A more advanced submit description file

```
universe = vanilla
executable = run.sh

request_cpus = 4
request_memory = 8000

transfer_input_files = input.dat

output = job.$(cluster).$(process).out
error = job.$(cluster).$(process).err

should_transfer_files = YES
when_to_transfer_output = ON_EXIT

requirements = Machine =?= "lcg1111.gridpp.rl.ac.uk"

periodic_remove = JobStatus == 2 && (CurrentTime - EnteredCurrentStatus > 24*60*60)

queue 1
```

Transfer output files from worker node back to submit host

Running jobs

- A more advanced submit description file

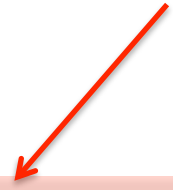
```
universe = vanilla
executable = run.sh

request_cpus = 4
request_memory = 8000

transfer_input_files = input.dat

output = job.$(cluster).$(process).out
error = job.$(cluster).$(process).err

should_transfer_files = YES
when_to_transfer_output = ON_EXIT

requirements = Machine =?= "lcg1111.gridpp.rl.ac.uk"

periodic_remove = JobStatus == 2 && (CurrentTime - EnteredCurrentStatus > 24*60*60)

queue 1
```

Running jobs

- A more advanced submit description file

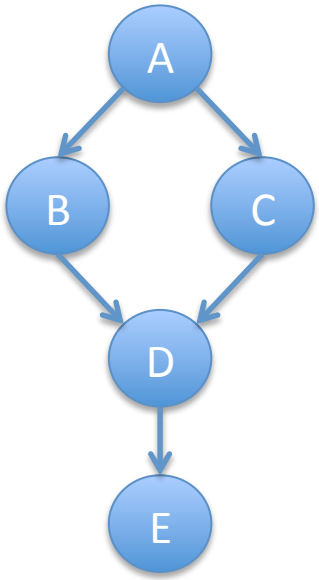
```
universe = vanilla
executable = run.sh

request_cpus = 4
request_memory = 8000

transfer_input_files = input.dat

output = job.$(cluster).$(process).out
error = job.$(cluster).$(process).err

should_transfer_files = YES
when_to_transfer_output = ON_EXIT

requirements = Machine =?= "lcg1111.gridpp.rl.ac.uk"

periodic_remove = JobStatus == 2 && (CurrentTime - EnteredCurrentStatus > 24*60*60)

queue 1
```

Remove job if it has been running for more than 24 hours



Workflow management

- Sometimes you need to run more than just lots of independent jobs
- DAGMan: Directed Acyclic Graph Manager
  - A DAG can represent a set of computations
  - Input, output or execution of one or more of the computations depends on other computations
- Submit a DAGMan job to HTCondor
  - HTCondor then submits the individual jobs & enforces the dependencies



Universes

- A universe specifies a HTCondor runtime environment

Universe	
Vanilla	run any executable
Standard	run executables linked to HTCondor libraries, supports checkpointing
Parallel	jobs that span multiple machines (e.g. MPI)
Docker	executable run inside a Docker container
VM	instantiate a virtual machine on a worker node
Grid	run jobs on another HTCondor pool or the grid, or instantiate a virtual machine on a cloud
Local	a job which runs immediately on the submit machine, useful for meta-schedulers

Grid universe

- Also known as HTCondor-G
- Example

```
-bash-4.1$ cat grid.sub
universe = grid
grid_resource = nordugrid t2arc01.physics.ox.ac.uk
executable = test.sh
Log = log.$(Cluster).$(Process)
Output = out.$(Cluster).$(Process)
Error = err.$(Cluster).$(Process)
should_transfer_files = YES
when_to_transfer_output = ON_EXIT
queue 1


-bash-4.1$ condor_submit grid.sub
Submitting job(s).
1 job(s) submitted to cluster 16240.
```

Grid universe

- Checking status of job:

```
-bash-4.1$ condor_q

-- Submitter: lcgui03.gridpp.rl.ac.uk : <130.246.180.41:45033> : lcgui03.gridpp.rl.ac.uk
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
16240.0   alahiff         5/30 15:42   0+00:00:00 I  0   0.0  test.sh

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


-bash-4.1$ condor_q -grid

-- Submitter: lcgui03.gridpp.rl.ac.uk : <130.246.180.41:45033> : lcgui03.gridpp.rl.ac.uk
 ID       OWNER          STATUS     GRID->MANAGER    HOST       GRID_JOB_ID
16240.0   alahiff        INLRMS:R   nordugrid->[?] t2arc01.phys N84MDm8BdAknD0VB
```

Running a job on a remote site is just as easy as running a job locally

Grid universe

- Can instantiate cloud VMs using EC2
- Submit description file:

```
universe = grid
grid_resource = ec2 http://cloud.mydomain:4567
executable = $(ec2_ami_id)
log = $(executable).$(cluster).$(Process).log
ec2_access_key_id = /home/alahiff/AccessKeyID
ec2_secret_access_key = /home/alahiff/SecretAccessKey
ec2_ami_id = ami-00000067
ec2_instance_type = m1.large
ec2_keypair_file = $(executable).$(cluster).$(Process).pem
ec2_user_data_file = /home/alahiff/my_user_data
queue 5
```

Grid universe

- Submit job as normal

```
-bash-3.2$ condor_submit vms.sub
Submitting job(s).....
5 job(s) submitted to cluster 86.
```

- Query status

```
-bash-3.2$ condor_q -af EC2InstanceName EC2RemoteVirtualMachineName
i-00003557 130.246.223.217
i-00003559 130.246.223.243
i-00003555 130.246.223.208
i-00003558 130.246.223.219
i-00003556 130.246.223.211
```

In HTCondor a “job” can be a virtual machine on a remote cloud

Job router

- Job router is a HTCondor daemon which transforms jobs according to a configurable policy
  - Edits job ClassAds
  - Can specify multiple policies with job requirements for each
  - Can specify how to detect failed jobs
  - Black hole detection
- Simple example
  - If user X's jobs have been idle for more than 2 hours, convert to Grid universe

Dynamic environments

- Unlike many other batch systems, the central manager doesn't require a list of worker nodes
  - It is the responsibility of worker nodes to advertise themselves to the collector
  - The collector maintains a list of worker nodes
  - If a worker node goes away, it will eventually disappear from the collector & no longer be matched to jobs
- This makes HTCondor an ideal choice for situations where there are dynamic resources
  - Dynamically create worker nodes on demand

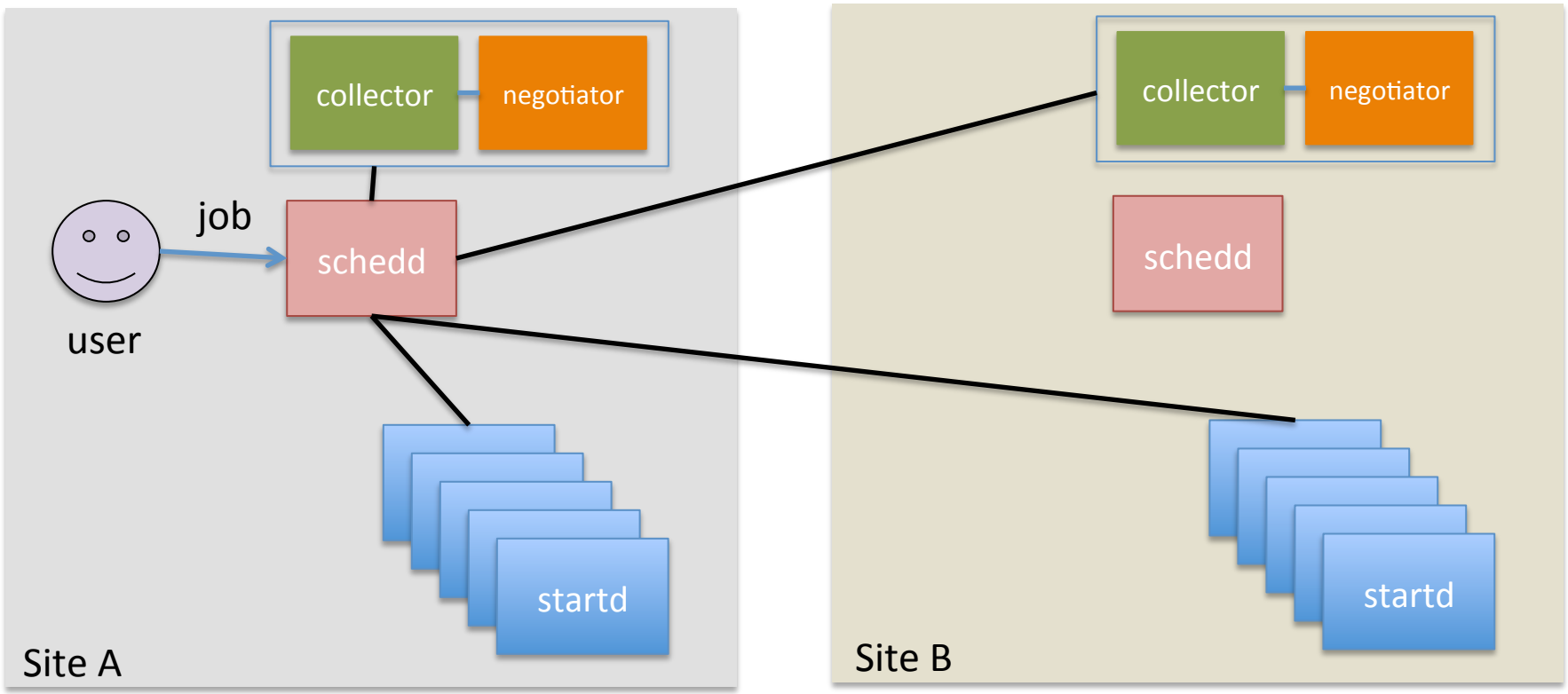
Dynamic environments

- Allows for possibilities including:
  - Elastic HTCondor pool on a cloud
    - all worker nodes are on cloud VMs
    - (third-party) mechanism to create worker node VMs as needed
    - size of HTCondor pool depends on number of jobs
  - Bursting a HTCondor pool into a cloud
    - dedicated HTCondor pool on-premises
    - when local pool runs out of resources, worker nodes are created on cloud VMs (private and/or public clouds)
    - example: at the RAL Tier-1 we do this to make use of idle resources in our private cloud

Can use HTCondor to aggregate multiple clouds (& local resources)

Flocking

- Federation of HTCondor pools
  - idle jobs from one HTCondor pool can “flock” to another pool which has free resources



GlideinWMS

- Aggregation of many unrelated HTC systems into a single overlay HTCondor pool
  - Submit jobs, which themselves are HTCondor worker nodes, to external resources
- Main function of GlideinWMS is resource provisioning
  - decides when more resources are needed
  - decides where to get the resources from
  - validates & configures them

Users see just a standard HTCondor pool, but the worker nodes could be all over the world

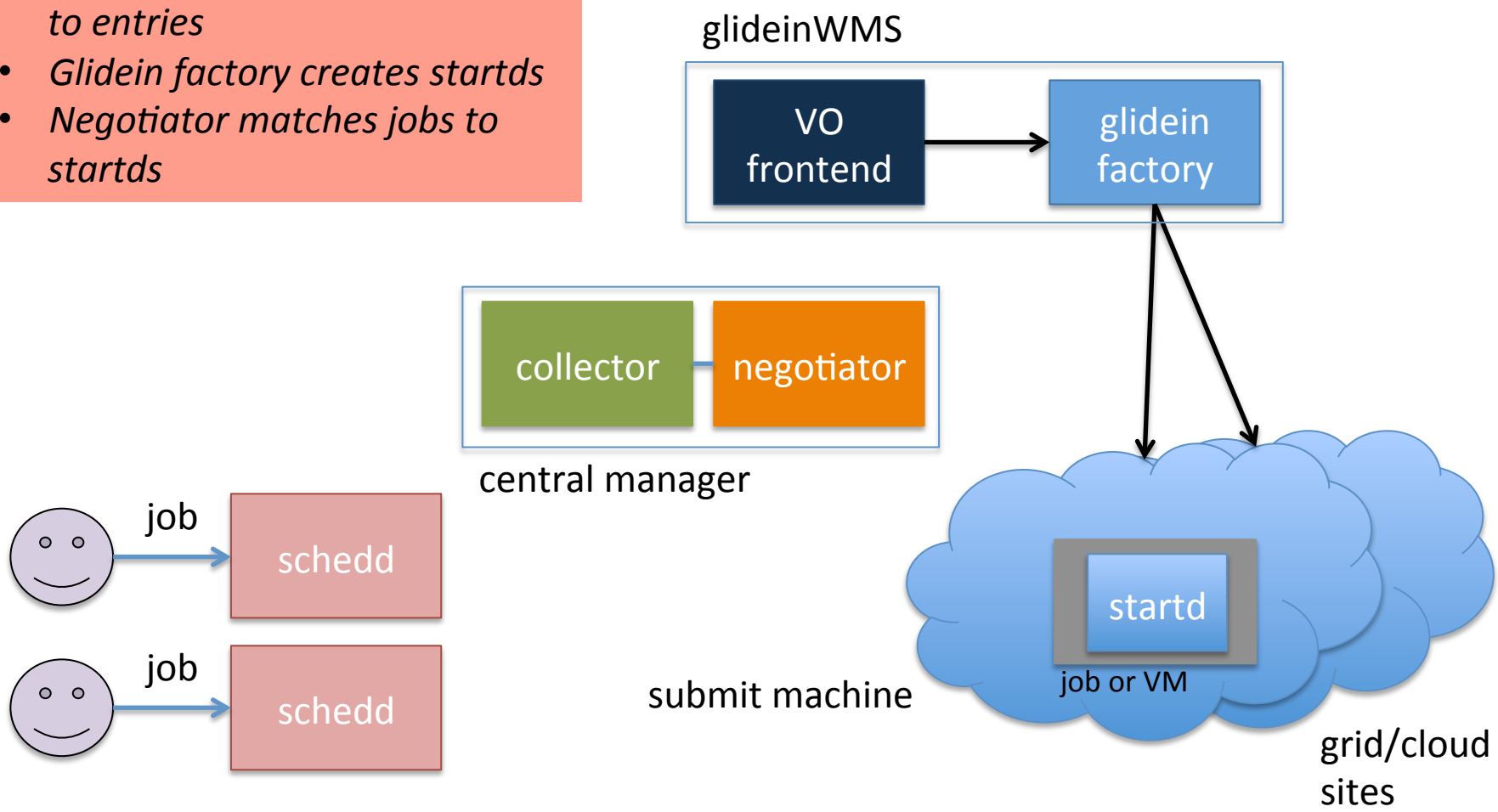
GlideinWMS

- GlideinWMS composed of
  - User pool
    - A HTCondor pool which users submit jobs to
  - VO frontend
    - matches idle jobs to entries (queues at sites)
    - instructs the glidein factory to increase or decrease the number of glideins in each entry
  - Glidein factory
    - submits glideins to grid sites using HTCondor-G
    - instantiates VMs on clouds using HTCondor-G

A glidein is a properly-configured HTCondor execution node submitted as a grid job

GlideinWMS architecture

- *VO frontend matches idles jobs to entries*
- *Glidein factory creates startds*
- *Negotiator matches jobs to startds*

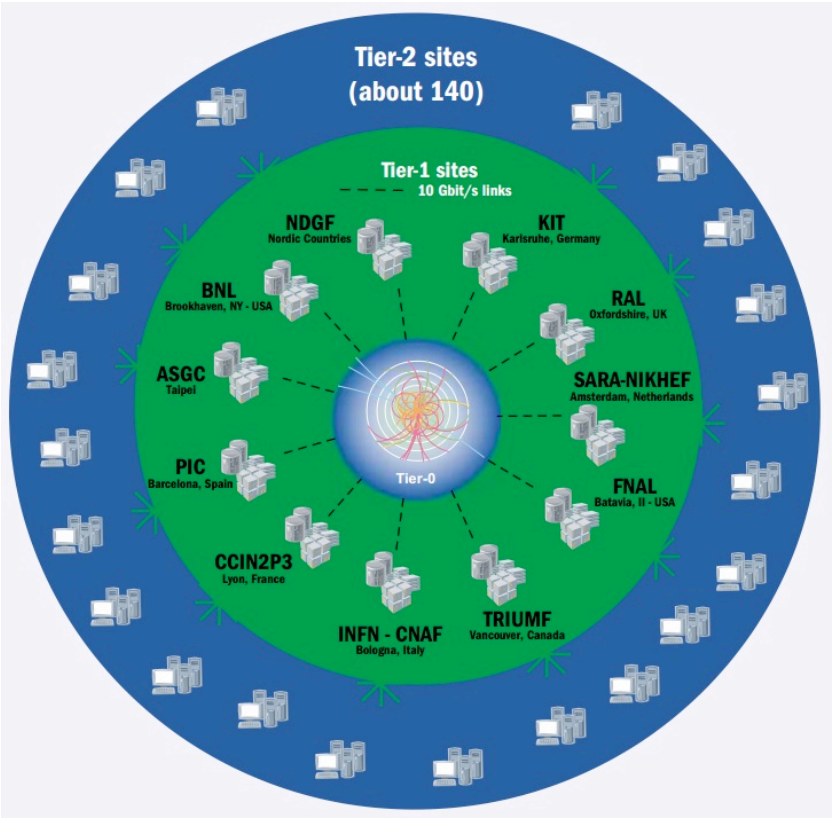


HTCondor & the LHC

- HTCondor plays many important roles in computing for the Large Hadron Collider
- It's used
  - as a batch system
  - as a grid computing element
  - as an overlay batch system
  - to provision resources
  - as a cloud batch system
  - to manage workflows

LHC computing

- Not just a single site, but many sites
  - need to be able to efficiently make use of all these distributed resources



Tier-0 (CERN)
- custodial storage
- first-pass reconstruction

Tier-1s
- custodial storage
- reprocessing

Tier-2s
- event simulation
- end-user analysis

Tier-3s
- smaller facilities

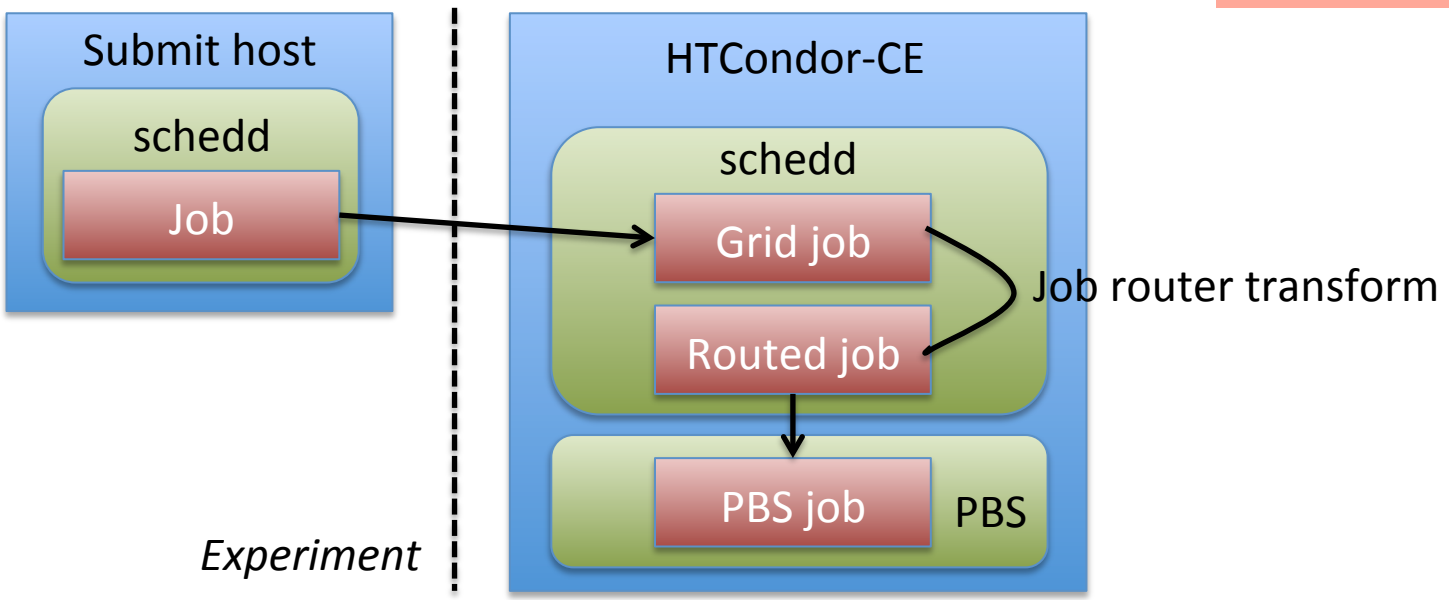
HTCondor as a batch system

- Used as a batch system at many US sites for a long time
- In recent years also being adopted by European sites
  - Started with the RAL Tier-1 (UK) in 2013
    - pool currently contains ~20,000 cores
  - Many European Tier-2s have since migrated
  - CERN currently migrating to HTCondor
- Reasons for moving to HTCondor
  - Scalability
  - Stability
  - Ability to handle dynamic resources

HTCondor as a grid compute element

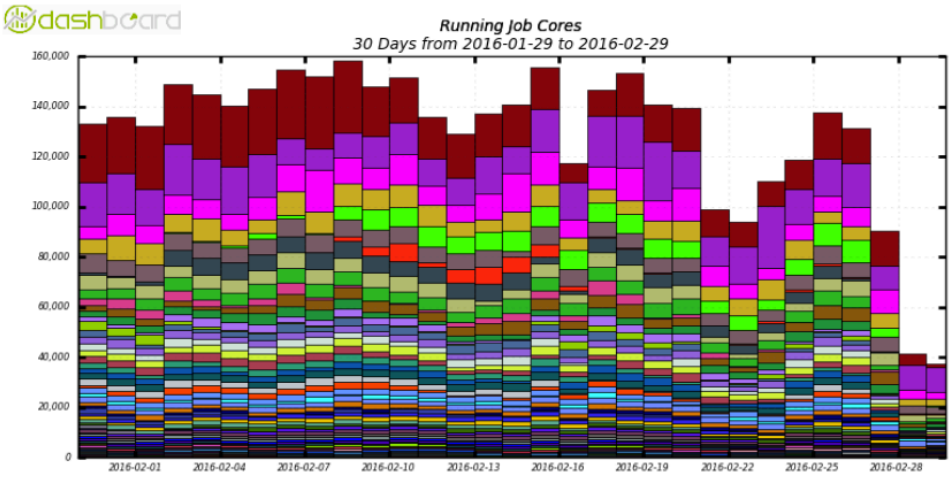
- Compute Element: grid job gateway to a site
  - Provide a single interface to different batch system technologies (HTCondor, LSF, GE, SLURM, PBS)
- Several types of CE
  - CREAM, ARC, recently HTCondor-CE

HTCondor-CE uses only existing HTCondor functionality



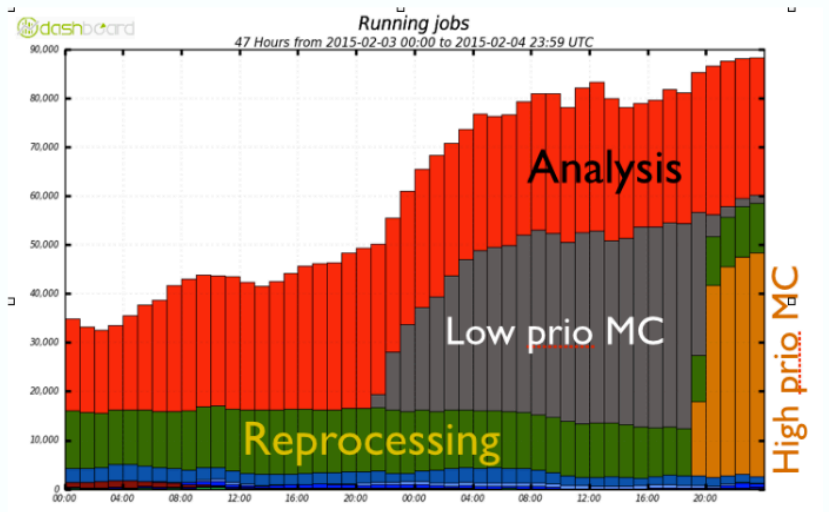
HTCondor as an overlay pool

- CMS (Compact Muon Solenoid) is one of the main 4 LHC particle physics experiments at CERN
- Uses a single HTCondor pool with glideinWMS



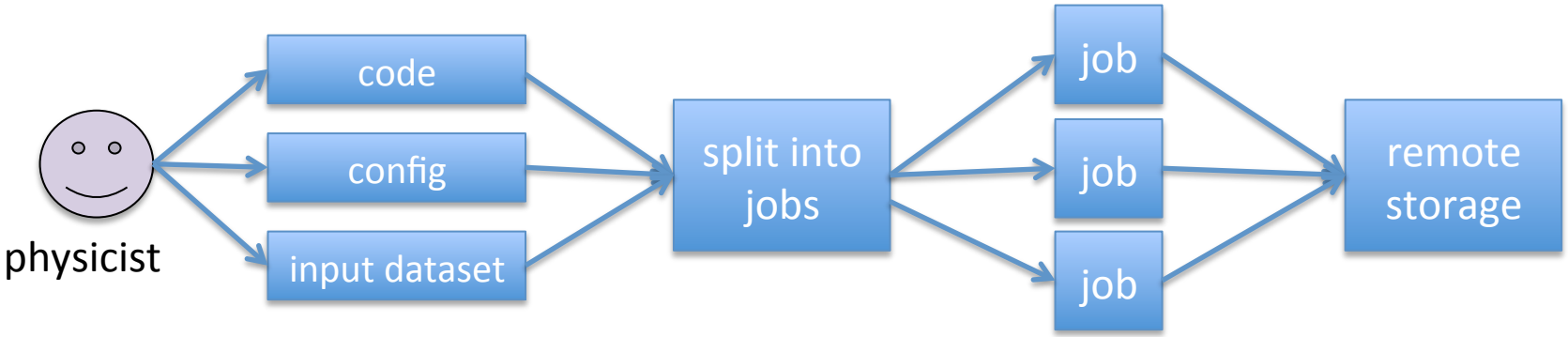
Over 150,000 simultaneous running jobs
(each colour is a different site)

Prioritization of global resources



HTCondor managing workflows

- CRAB3 – CMS Remote Analysis Builder
- Tool used by CMS physicists to carry out analysis



- HTCondor DAGMan used to manage tasks
  - Handles failures
  - Manages transfer of output to remote site

Summary

- HTCondor as a batch system
  - Powerful
  - Very scalable
  - Ideal for dynamic environments
- HTCondor beyond a “standard” batch system
  - Can provision resources
  - Can aggregate compute resources around the world
- HTCondor plays a crucial role in computing for the Large Hadron Collider