Producing Madgraph5_aMC@NLO gridpacks and using TensorFlow GPU resources in the CMS HTCondor Global Pool

Brian Bockelman[6], Edgar Fajardo Hernandez[4], Diego Davila Foyo[1], **Kenyi Hurtado Anampa**[7], Farrukh Aftab Khan[3], Krista Larson[3], James Letts[4], Marco Mascheroni[3], David Mason[3], Antonio Perez-Calero Yzquierdo[2], Todor Trendafilov Ivanov[8]

[1] Autonomous University of Puebla (MX)
[2] CIEMAT & PIC (ES)
[3] Fermilab (US)

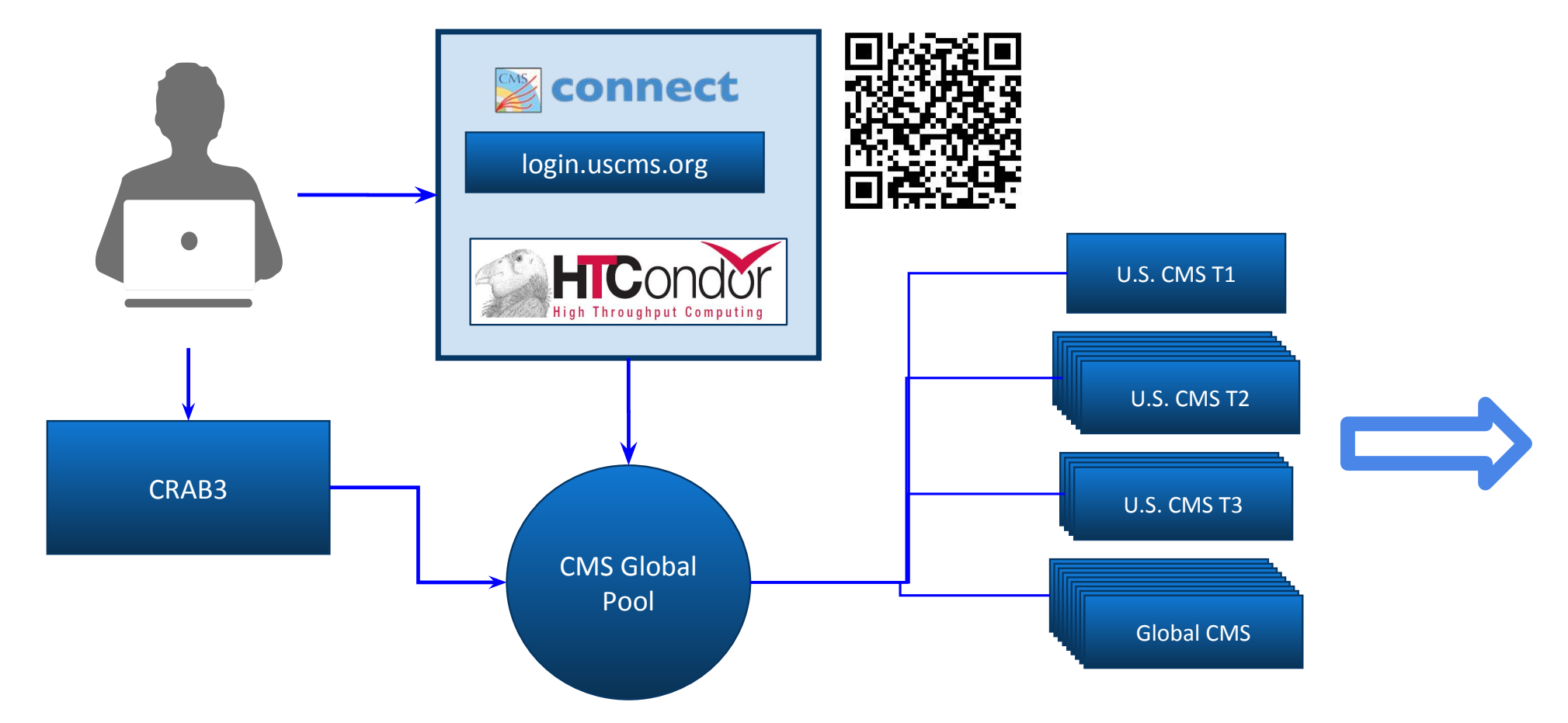
[4] University of California San Diego (US)
[5] University of Malaya (MY)
[6] University of Nebraska - Lincoln (US)

[7] University of Notre Dame (US)
[8] University of Sofia (BG)

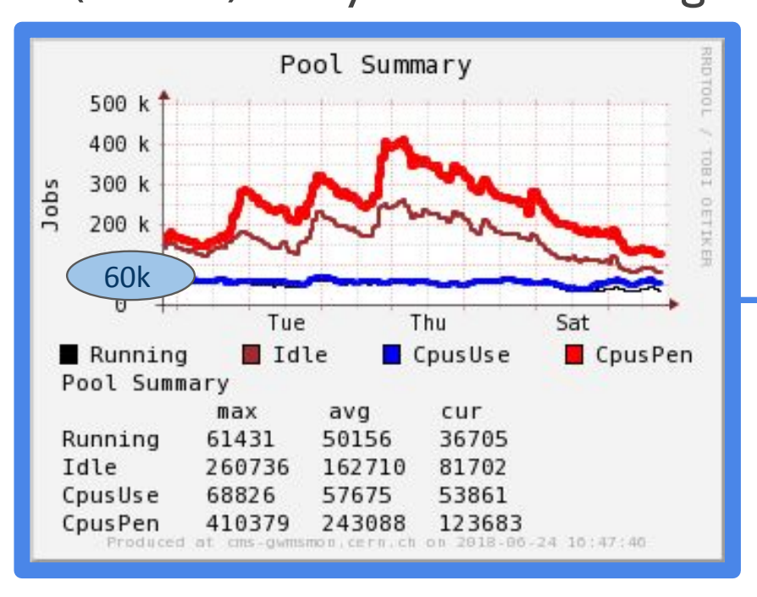
While submission of CMS user jobs to the Global Pool is mostly managed by CRAB3, the standard analysis workflow management tool, the generation of matrix elements for high energy physics processes via Madgraph5_aMC@NLO and the usage of machine learning tools with GPU resources are independent use-cases that require special adaptation in order to take advantage of the Global Pool resources. This work describes the challenges and efforts performed towards adapting such workflows for it.

The submission point - CMS Connect

CMS Connect provides a service with a Tier 3-Like interface where users can submit condor jobs to the CMS Global Pool, a global HTCondor pool provisioned by GlideinWMS. It complements CRAB3, dealing with a different set of analysis workflows, such as Madgraph gridpacks and the use of GPU resources with TensorFlow jobs.



(Offline) Analysis resource usage



```
Pool Summary
            max      avg      cur
Running     61431    50156    36705
Idle        260736   162710   81702
CpusUse     68826    57675    53861
CpusPen     410379   243088   123683
  Produced at cms-gwmsmon.cern.ch on 2018-06-24 16:47:40
```

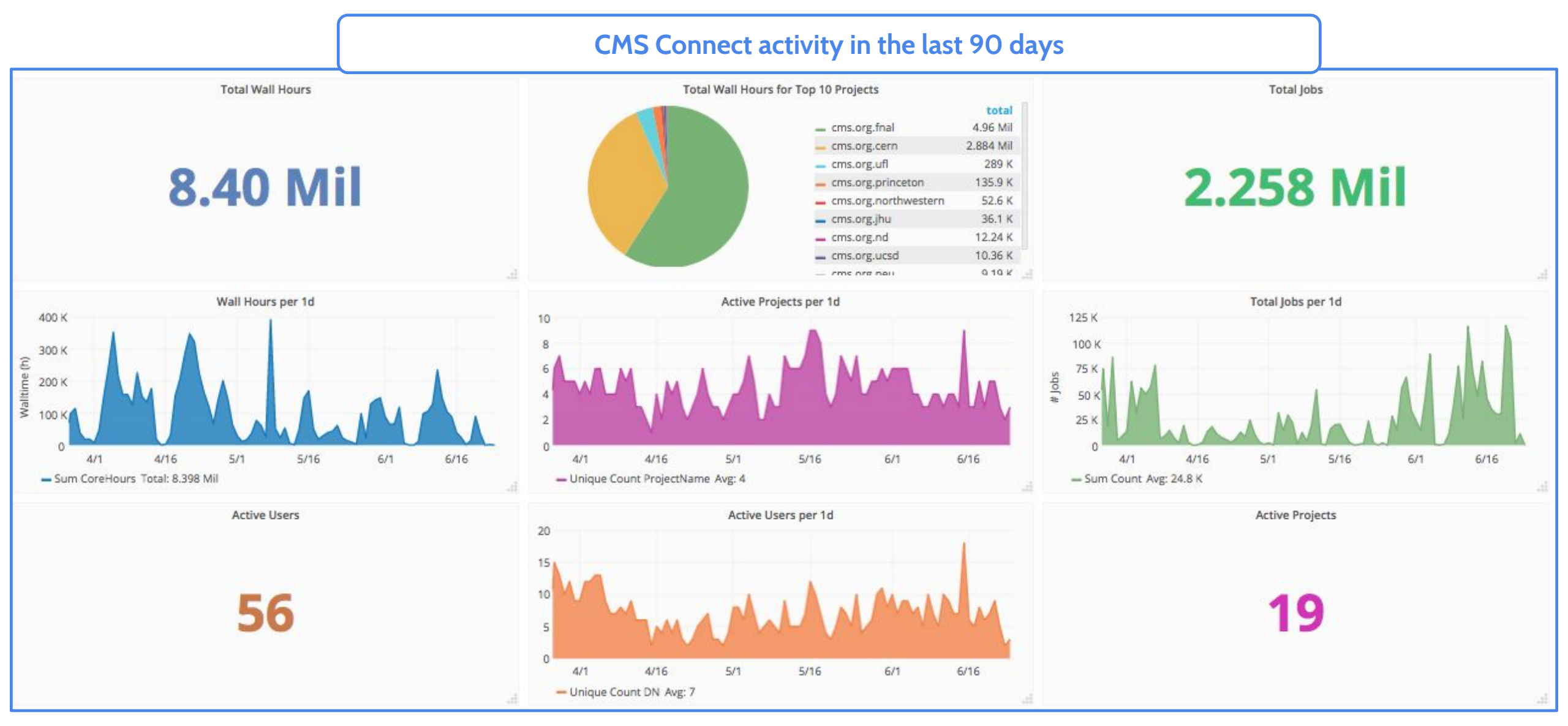
CRAB 3

- Better suited for:
  - cmsRun jobs
    - These include a large range of analysis workflows within the CMSSW framework for simulation and data analysis.
  - Random scripts using distributed data sets, which CRAB can handle easily.

CMS Connect

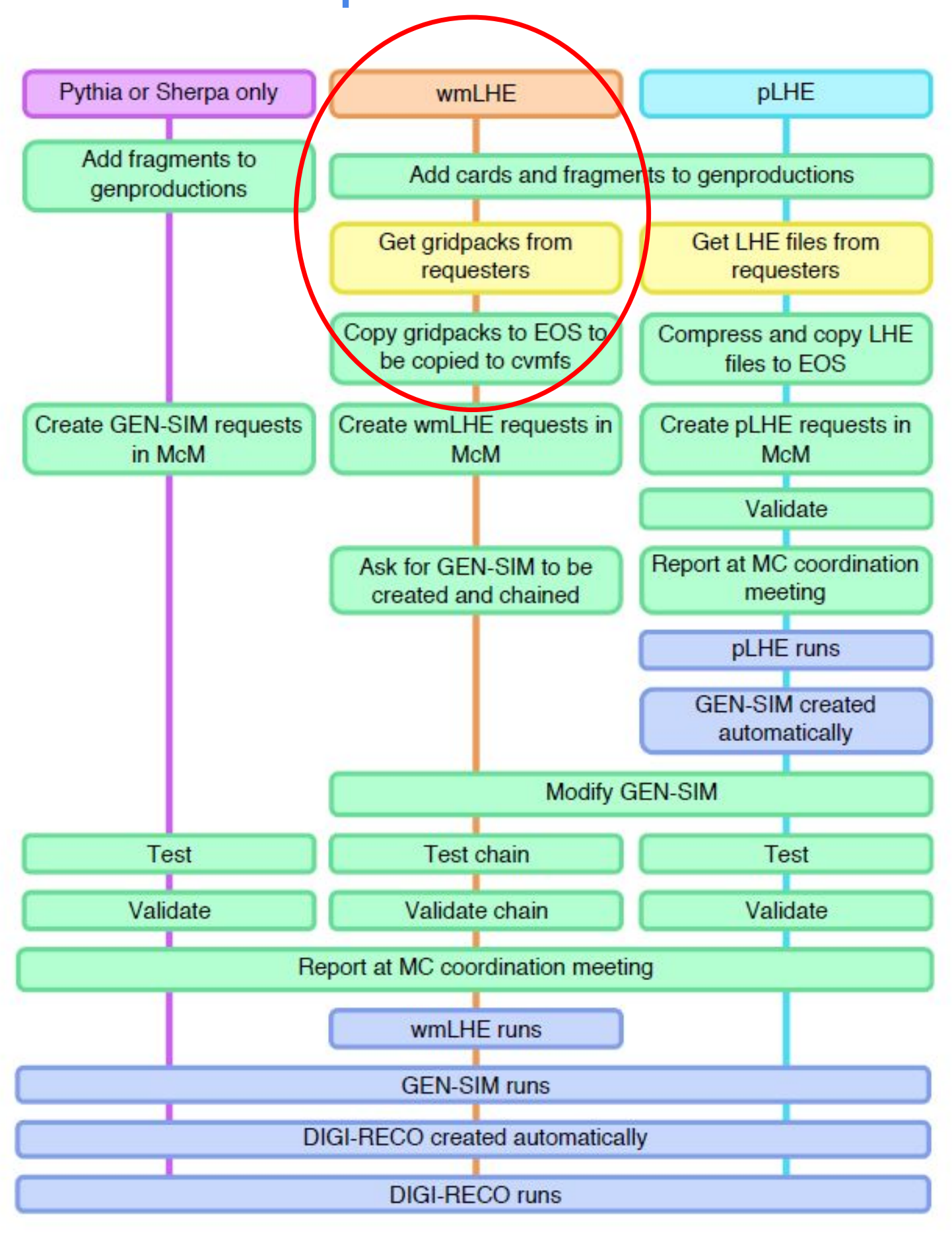
- Good for more general purpose workflows. For example:
  - Analysis scripts that users might prefer to handle directly with condor/condor_dagman.
  - Workflows that already have a batch submission manager (e.g., Madgraph5).
  - Machine learning tools + GPU resources (TensorFlow)

CMS Connect activity in the last 90 days



Generating Madgraph5_amC@NLO gridpacks

MC request submission


Pythia or Sherpa only
wmLHE
pLHE
Add fragments to genproductions
Add cards and fragments to genproductions
Get gridpacks from requesters
Get LHE files from requesters
Copy gridpacks to EOS to be copied to cvmfs
Compress and copy LHE files to EOS
Create GEN-SIM requests in McM
Create wmLHE requests in McM
Create pLHE requests in McM
Validate
Ask for GEN-SIM to be created and chained
Report at MC coordination meeting
pLHE runs
GEN-SIM created automatically
Modify GEN-SIM
Test
Test chain
Test
Validate
Validate chain
Validate
Report at MC coordination meeting
wmLHE runs
GEN-SIM runs
DIGI-RECO created automatically
DIGI-RECO runs

Producing "gridpacks" is one of the very first steps in the Monte Carlo request submission chain.
This kind of workflow however, is not integrated with the CMS standard executable (cmsRun) or CRAB3. A generator's package is used instead to produce these gridpacks in a standalone way, using its own submission managers.
The Monte Carlo contact persons using different local resources for producing these gridpacks didn't record the usage in CMS dashboards and had to take into account some submission differences for each resource. For example:

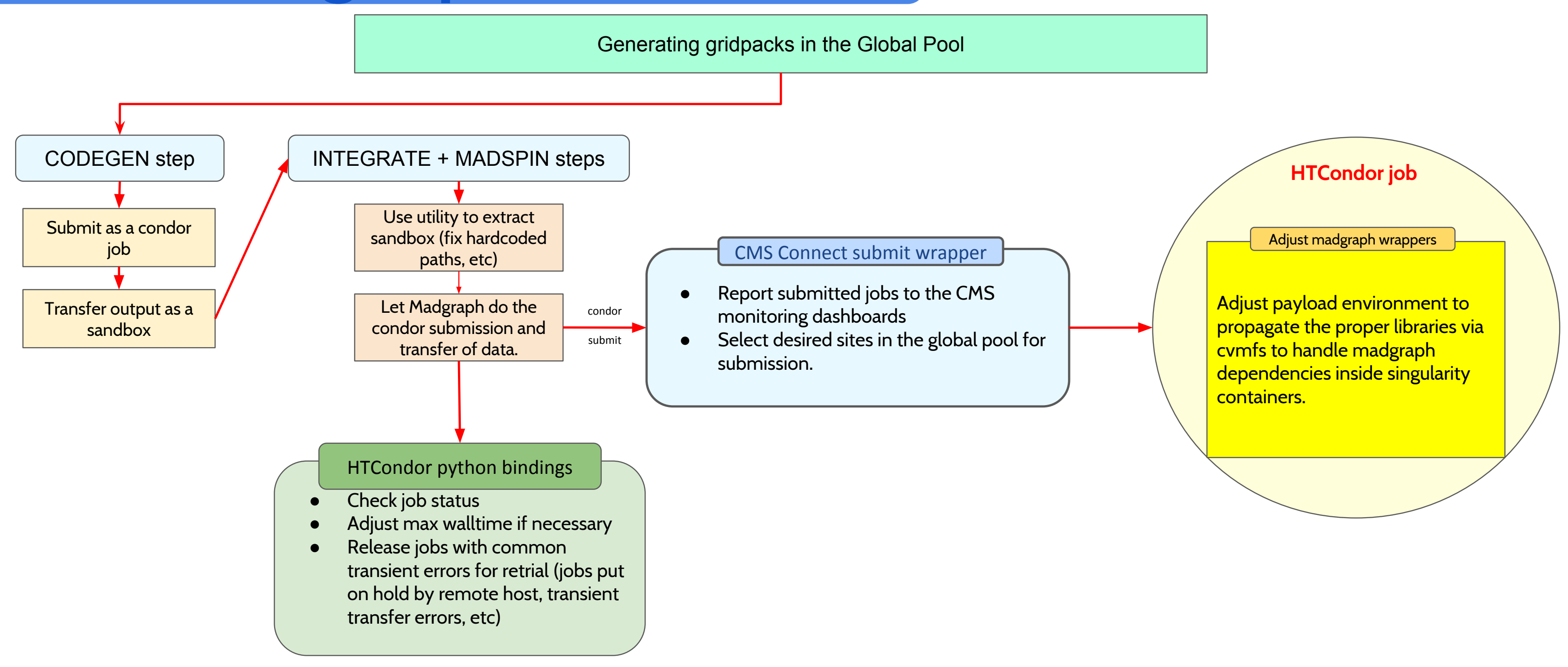
- CERN CAF (resources shared between all CERN-based experiments: ATLAS, CMS, ALICE, LHCb, etc):
  - Using the LSF batch system, requires working with different queue types (1nd, 2nw, etc) and submitting the whole master process as a job.
  - Using HTCondor needs to deal with AFS tokens and k5reauth.
- FNAL LPC CAF uses HTCondor, used by US-based Institutions.

Submitting these gridpacks to the Global Pool offers:

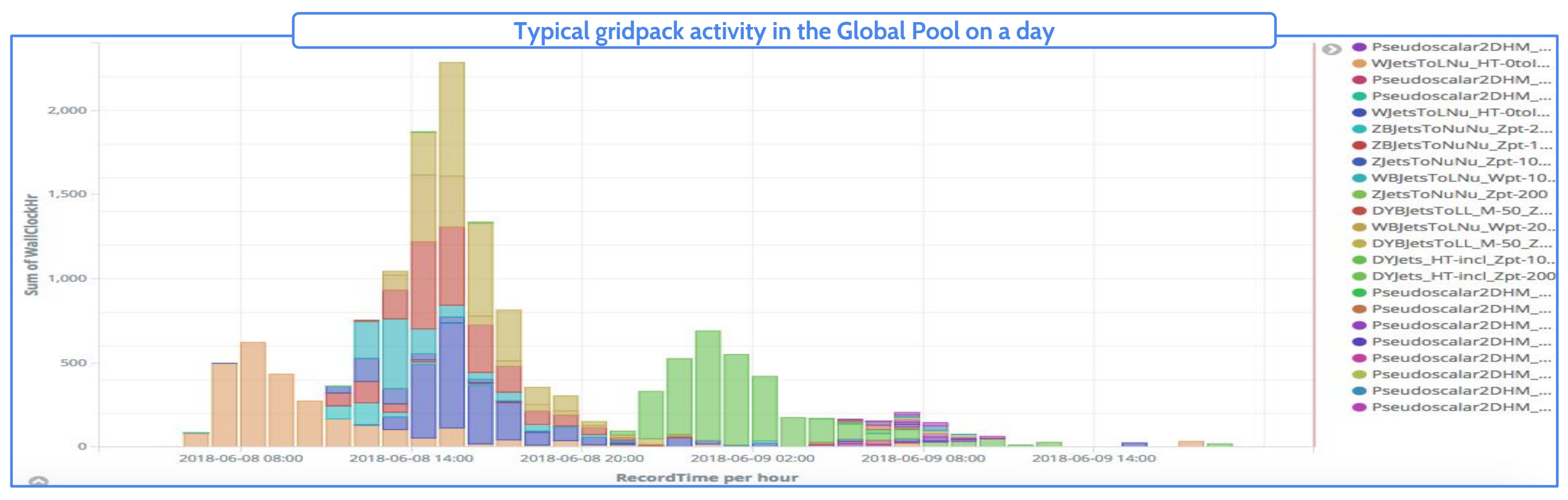
- A uniform layer through HTCondor for these workflows, accessing all resources CMS has access to, like any other workflows handled e.g., via CRAB3.
- The activity is monitored and recorded in CMS monitoring dashboards. (Better accounting).

Some adaptations were made to make madgraph compatible with the way the Global Pool works, including:

- Dynamic adjustment of requested walltime per job when needed.
- Specifying remote CMS Sites for submission.
- Handling typical transient errors translating into jobs getting held.
- Setting specific HTCondor classads for dashboard reporting.


Generating gridpacks in the Global Pool
CODEGEN step
Submit as a condor job
Transfer output as a sandbox
INTEGRATE + MADSPIN steps
Use utility to extract sandbox (fix hardcoded paths, etc)
Let Madgraph do the condor submission and transfer of data.
condor submit
CMS Connect submit wrapper
Report submitted jobs to the CMS monitoring dashboards
Select desired sites in the global pool for submission.
HTCondor python bindings
Check job status
Adjust max walltime if necessary
Release jobs with common transient errors for retrial (jobs put on hold by remote host, transient transfer errors, etc)
HTCondor job
Adjust madgraph wrappers
Adjust payload environment to propagate the proper libraries via cvmfs to handle madgraph dependencies inside singularity containers.

Typical gridpack activity in the Global Pool on a day

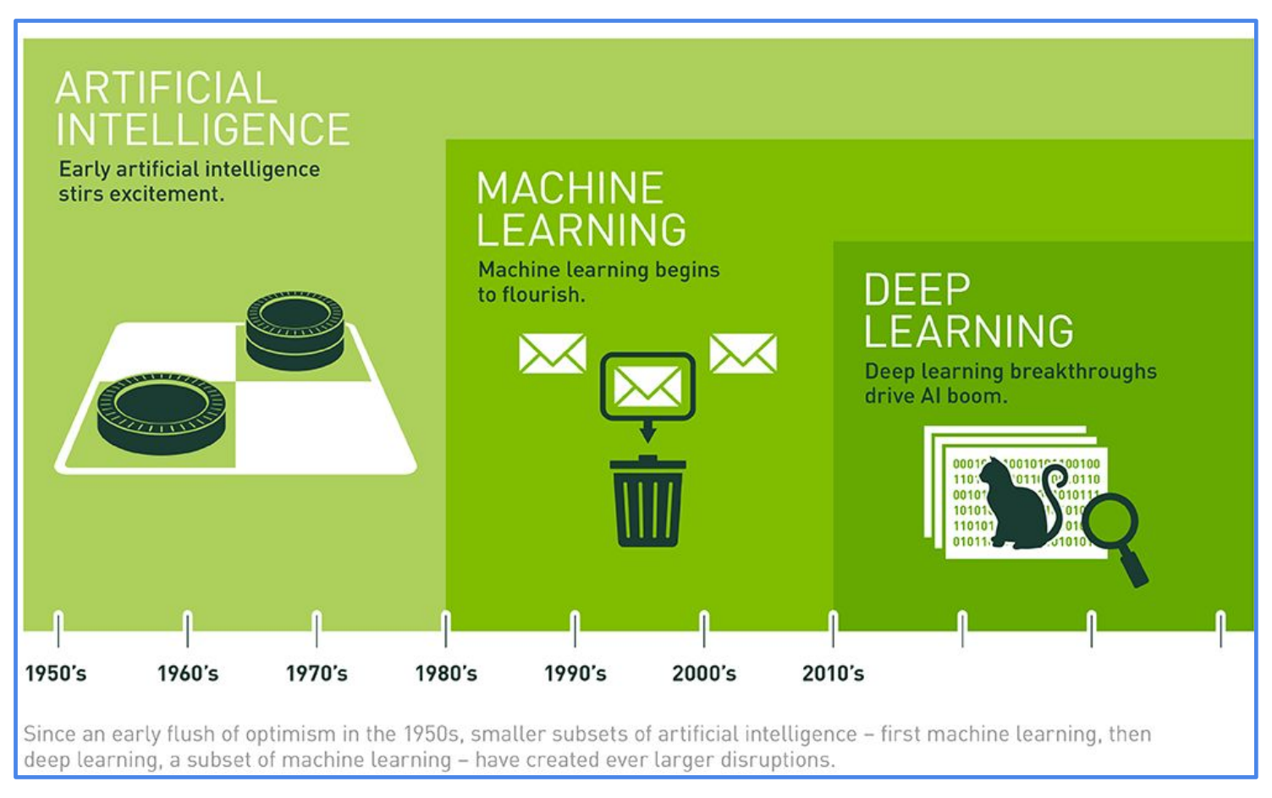


Deep learning and GPU resources

Deep learning

Even though Machine Learning (ML) has been a topic for decades and different ML algorithms have been used in high energy physics analysis since the nineties (e.g., Boosted decision trees, random forest, artificial neural networks, etc), the boom in terms of GPU resources demand started with the training of deep neural networks (a subset of ML inspired in artificial neural networks) just few years ago.

Deep learning algorithms involve a fair amount of matrix multiplications and other operations that can be massively parallelized and thus sped up on GPUs, due to the fact that GPUs can have thousands of cores and faster bandwidth to memory.
The usage of deep learning algorithms in industry has lead to the development of powerful machine learning frameworks, such as TensorFlow[1] (developed by Google), providing APIs for programming languages such as Python and C++, two popular languages in the HEP community. As a result, more progress than even has been made driving machine learning forward.


ARTIFICIAL INTELLIGENCE
Early artificial intelligence stirs excitement.
MACHINE LEARNING
Machine learning begins to flourish.
DEEP LEARNING
Deep learning breakthroughs drive AI boom.
1950's
1960's
1970's
1980's
1990's
2000's
2010's
Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

[1] https://www.tensorflow.org/

Using TensorFlow and GPU resources in the Global Pool

In order to use machine learning tools with Global Pool resources, the framework dependencies (i.e: TensorFlow, Keras, etc) need to be resolved first. CMS works mostly with Red Hat based Operating Systems (6 and 7), but TensorFlow officially supports Ubuntu only, so installing it by hand is not necessarily an easy task for a user.

To help with this, the CMS framework provides such dependencies via CVMFS, but its support is at the CPU-level only, since the integration with GPU resources can get tricky due to potential conflicts with GPU library dependencies. For instance, different TensorFlow versions can require specific versions of cuDNN (the Nvidia Deep Learning SDK) or the CUDA toolkit to work.

To overcome this issue on a wider scale, the OSG builds and maintains Singularity[2] containers based on Ubuntu for TensorFlow with GPU support. But new dependency trees arise: CUDA, CuDNN, GCC, etc. (See OSG oral presentation #49 at this conference for more details[3]).

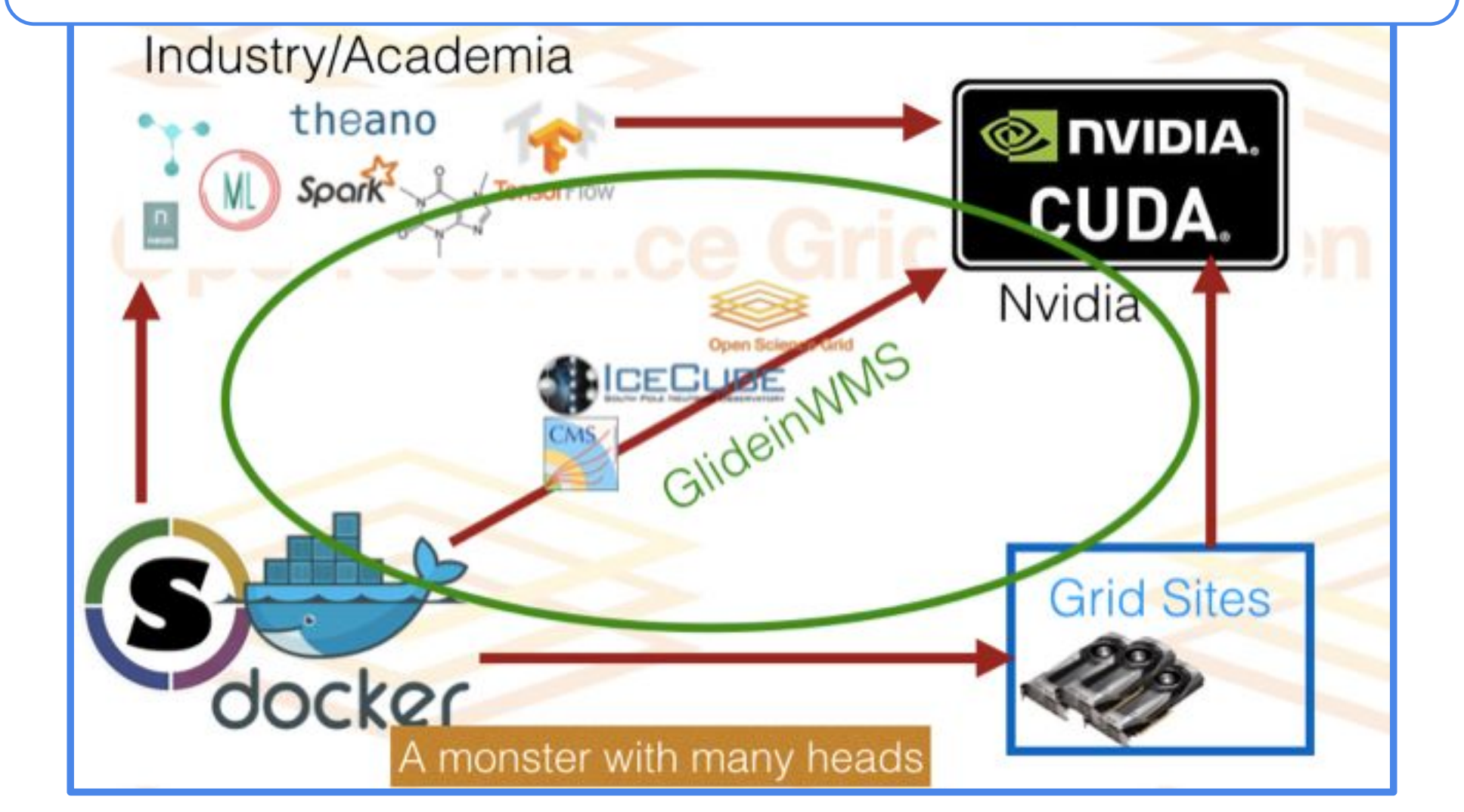
The CMS Global Pool powered by HTCondor and GlideinWMS has full support for Singularity, so OSG images can be used with the infrastructure through HTCondor in a transparent way.

Condor submit example for requesting GPU resources with OSG TensorFlow images

```
Universe = vanilla

#... (request resources, define logfiles, etc)
Executable = tf_matmul_wrapper_cvmfs.sh
transfer_input_files = tf_matmul.py
request_gpus = 1
Requirements = HAS_SINGULARITY == True
+SingularityImage =
"/cvmfs/singularity.opensciencegrid.org/openscience
grid/tensorflow-gpu:latest"
Queue 1
```

Using TensorFlow and GPU resources in the Global Pool



[2] http://singularity.lbl.gov/
[3] E. Fajardo Hernandez et. al., "OSG and GPUs: A tale of two use cases", CHEP 18 Oral Presentation on Track 3, Contribution ID #49