



# Reliable, flexible and secure logging system for distributed workflows

Leonardo Lai

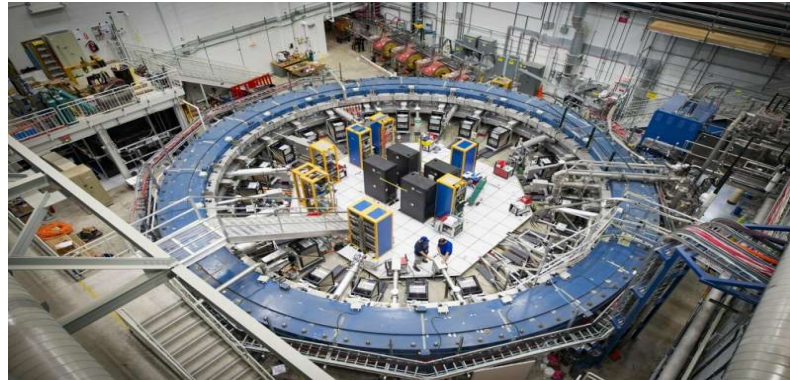
Final presentation

September 24<sup>th</sup>, 2019

Supervisor:

Marco Mambelli

# The need for HPC/HTC



**Muon g-2 experiment:  
18 Gb/s data acquisition rate**

[Gohn, Wesley. "Data acquisition for the new muon g-2 experiment at Fermilab." Journal of Physics: Conference Series. 2015.](#)

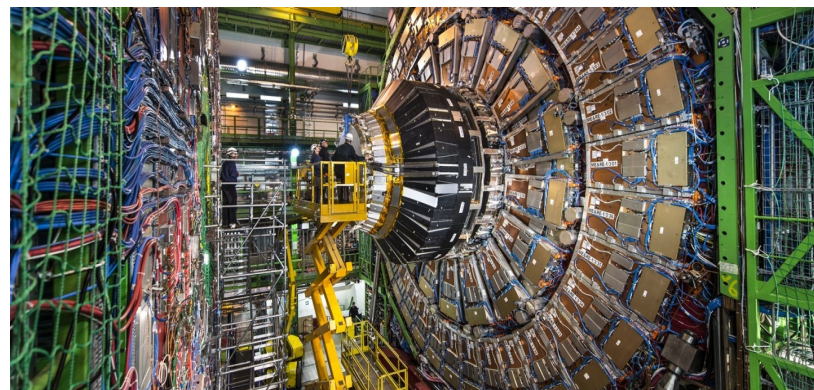
**DES (Dark Energy Survey):  
produces 2.5 TB of images per night**

<https://www.darkenergysurvey.org/the-des-project/survey-and-operations/>



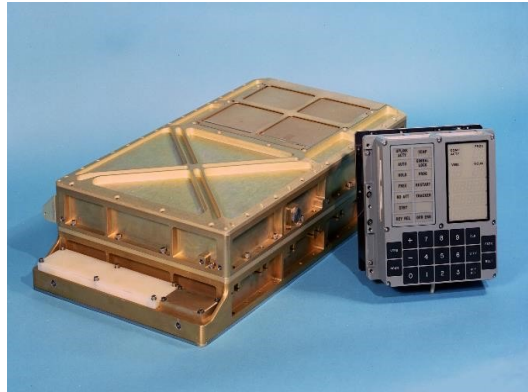
**LHC (Large Hadron Collider):  
generates 1 PB/s of data**

<https://home.cern/news/news/computing/cern-data-centre-passes-200-petabyte-milestone>





# A growing demand for resources

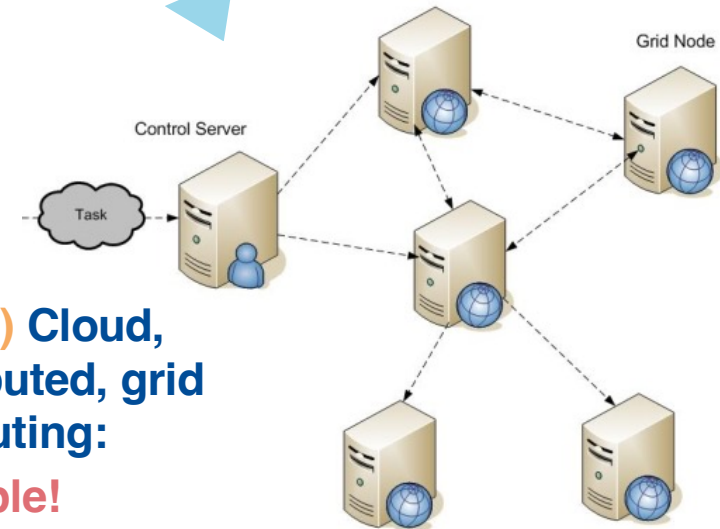


**(1966) Apollo guidance computer:**  
2MHz cpu  
2KB ram

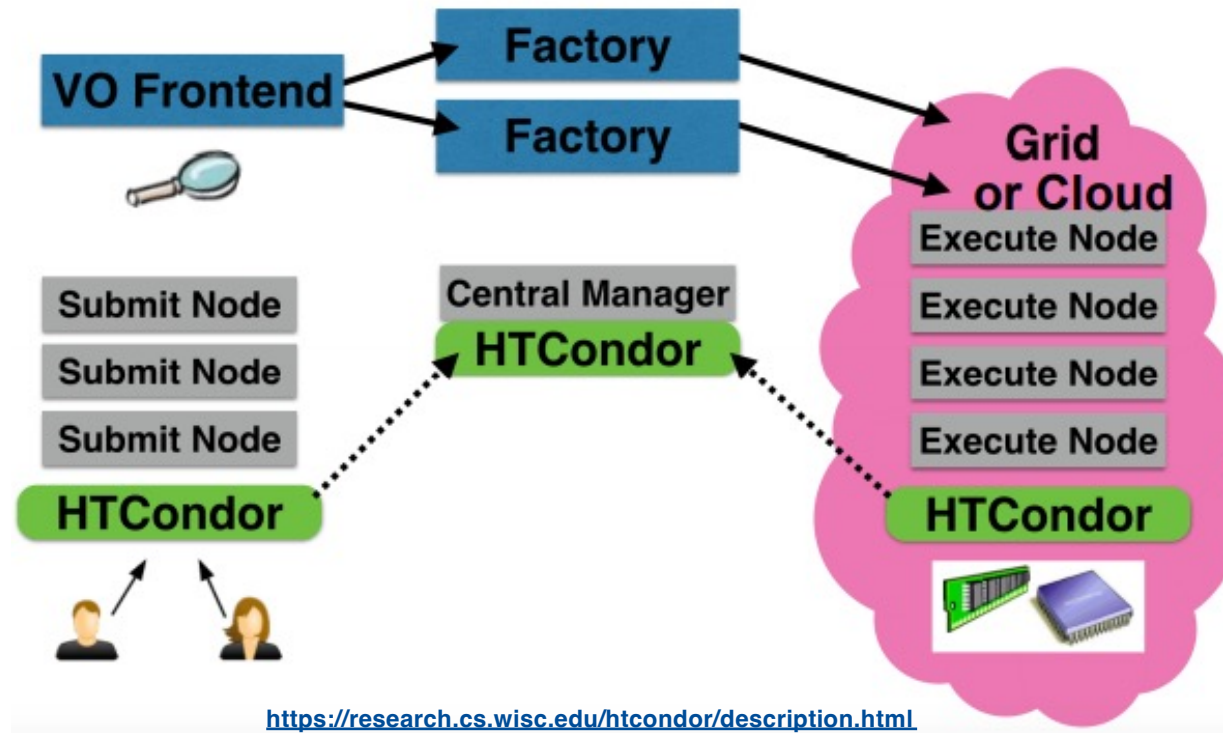


**(2002) Earth Simulator:**  
640 nodes,  
35.8 TFLOPS

**(1985) Cray-2 Supercomputer:**  
250 MHz cpu,  
1.9 GFLOPS



**(today) Cloud, distributed, grid computing:**  
**Scalable!**



HTCondor is a specialized [workload management system](#) for compute-intensive jobs.

Users submit their jobs to HTCondor, which enqueues and runs them somewhere in the grid, and eventually returns the result to the user.

# GlideinWMS

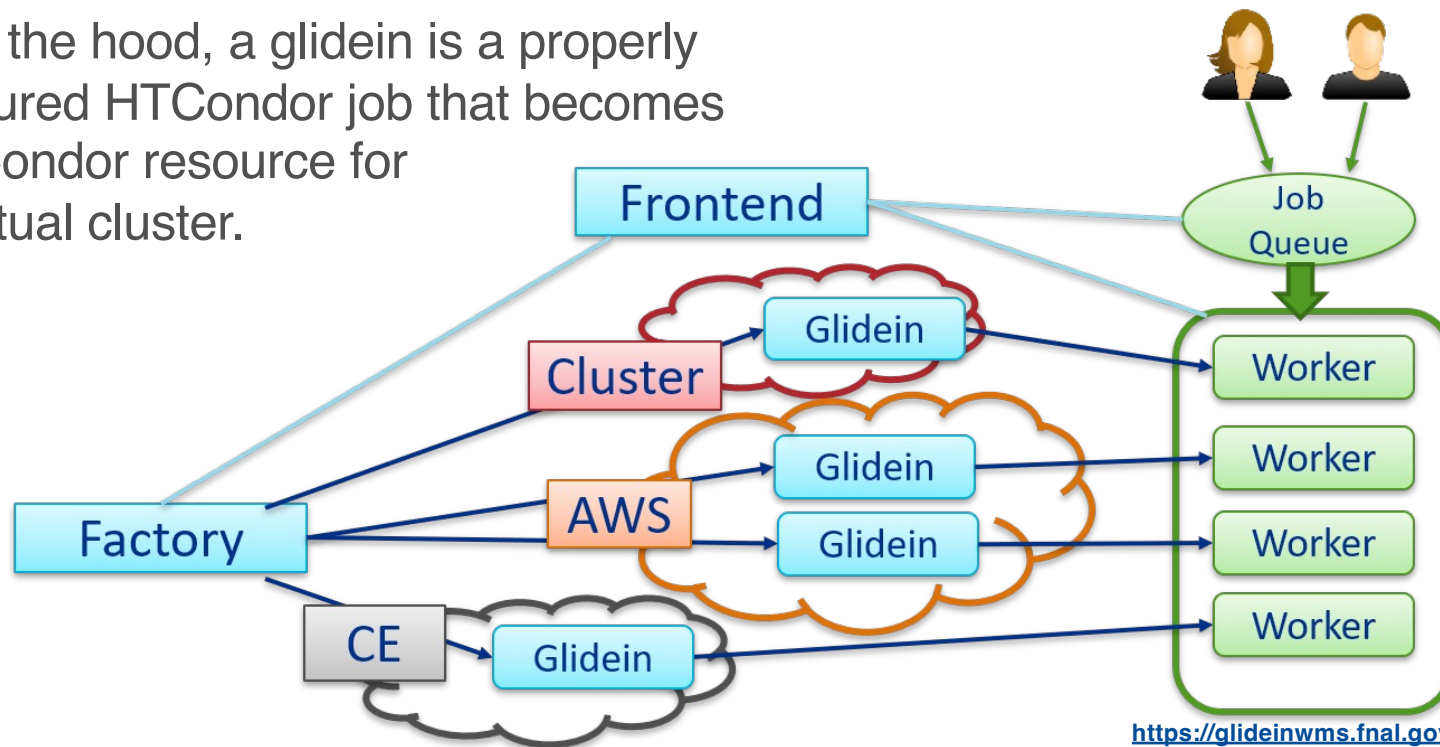
A grid is actually made of many independent sites, each with different characteristics. Abstraction needed: [GlideinWMS](#)

GlideinWMS works on top of HTCondor exploiting the concept of [glidein](#), a way to represent a resource in a HTCondor pool.

Under the hood, a glidein is a properly configured HTCondor job that becomes a HTCondor resource for the virtual cluster.

## Advantages:

- Transparent to the user
- Supports any kind of site
- The same Glidein can be reused for multiple jobs
- Can handle sites failures



<https://glideinwms.fnal.gov/doc.prd/index.html>

# Monitoring

Everything eventually breaks, even software.



If so, you may want to know where and why it is failing, to hopefully implement a fix for it (~~or blame someone else~~) → [debugging](#)

On the other hand, getting info from the system while it's running is always helpful, be it a critical message or just a minor update → [monitoring](#)

A flexible and reliable communication infrastructure is of key importance!

# Current issues

In GlideinWMS, there is currently a number of issues that justify a revamp of the logging system:

- Glideins stdout/stderr format is heterogeneous, not ideal for parsing
- If a glidein gets killed, its stdout/stderr disappears with him
- Having multiple jobs or threads in the same glidein results in a messy interleaved stdout/stderr
- Information becomes available only at the end of the glidein life
- A mechanism to broadcast this info to multiple listeners is lacking

# Issue: format

## stdout/stderr

```
Executing /tmp/glide_bSJAw6/main/check_proxy.sh
Executing /tmp/glide_bSJAw6/main/create_mapfile.sh
Executing /tmp/glide_bSJAw6/main/validate_node.sh
...
Unsetting X509_USER_PROXY
...
Signature OK for client_group:untar.j85dMS.cfg.
No signature for /tmp/glide_UI3SdR/client_group_main/nodes.blacklist.
cat_consts.j9hanE.sh: OK
...
SiteWMS_WN_Preempt = False
Executing /tmp/glide_UI3SdR/main/mjf_setparams.sh
MACHINE_TOTAL_CPU = "Unknown"
```

- ← missing subject
- ← no timestamp
- ← is it an error?
- ← mixing info of different nature

## New log

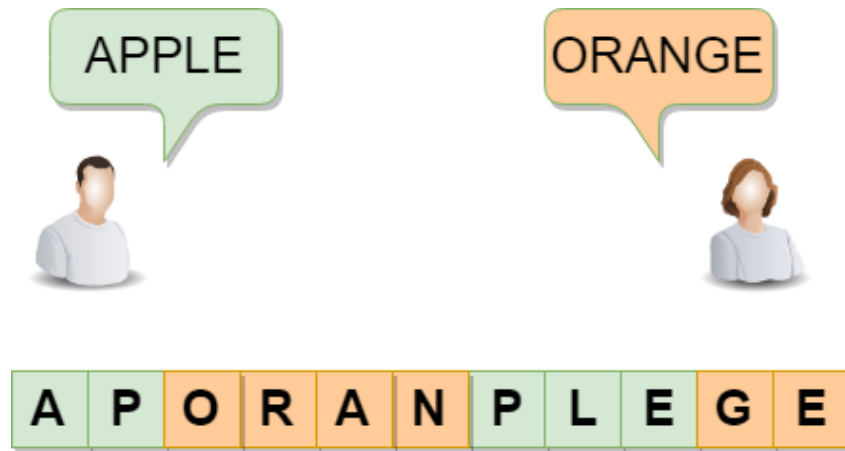
- homogeneous (JSON)
- complete
- parsable
- readable

Who  
Why  
What

```
{
  "invoker": "glidein_startup.sh",
  "pid": "16070",
  "timestamp": "2019-08-19T17:02:17-05:00",
  "severity": "warn",
  "type": "text",
  "filename": "",
  "content": "curl not installed on this machine"
},
{
  "invoker": "glidein_startup.sh",
  "pid": "16070",
  "timestamp": "2019-08-19T17:02:06-05:00",
  "severity": "info",
  "type": "text",
  "filename": "",
  "content": "Remote logging has been setup."
}
```



# Issue: concurrency



Multi-glideins behave like this when writing to log.

In general, bad things happen if multiple processes **write** the **same location** at the **same time!**

## Possible solution: locking

Before accessing the resource (log file):

- if busy (others writing) -> WAIT
- if free (nobody writing) -> ENTER

When releasing the resource:

- NOTIFY those who are waiting, if any



## Problem

Time spent waiting is basically time wasted.

**Performance decrease** if many glideins write frequently.

# Issue: concurrency

## Another solution...

Write always to different locations (files), a.k.a. *shards*.

All the shards will be eventually concatenated (*coalesced*) to a single log.



## ...another problem

What if one glidein starts merging the shards while others are writing them?

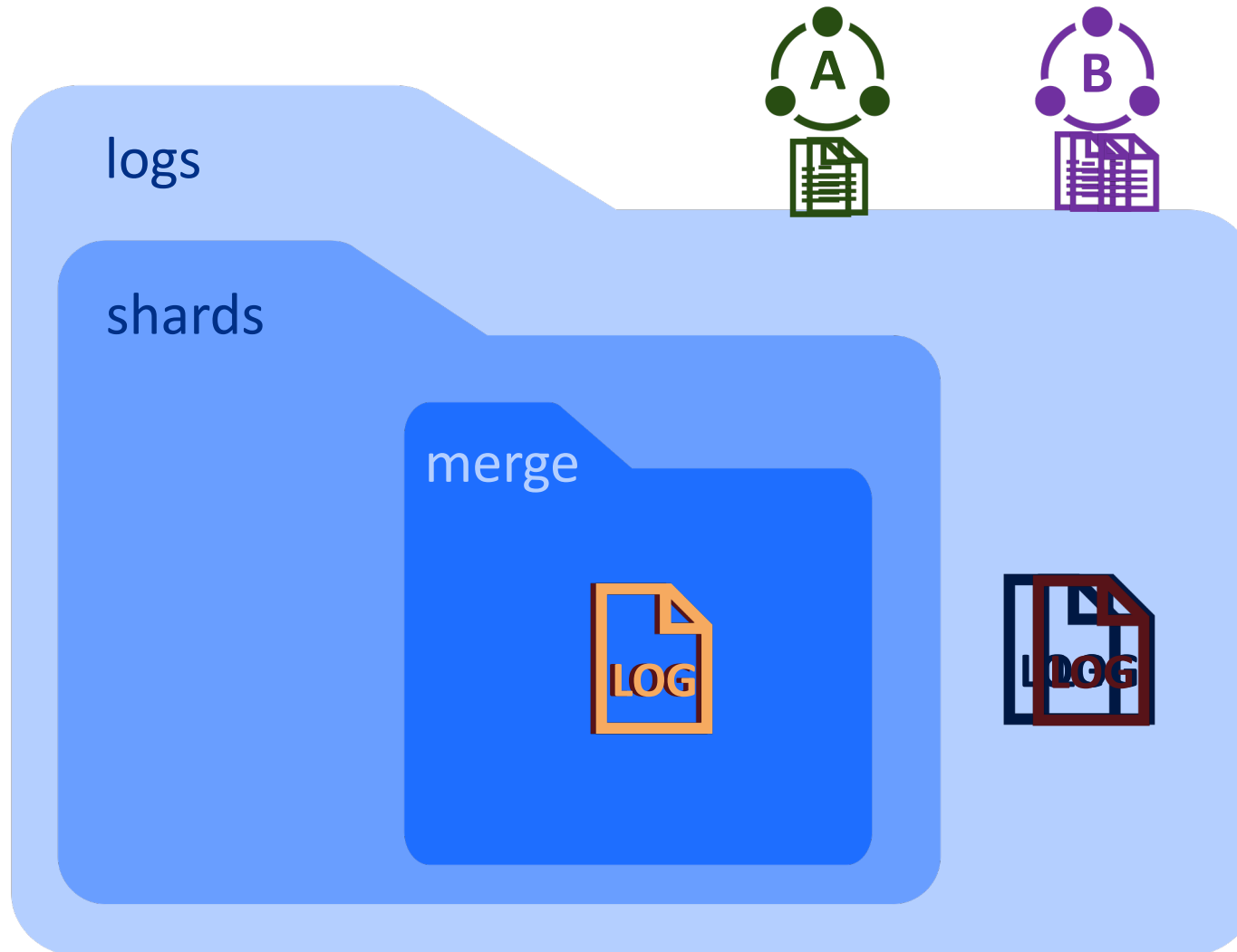


## The solution (working!)

Based on 3 ideas:

- Sharding
- Minimal use of locks (hidden)
- Isolation (different kinds of operations are performed in distinct places)

# Issue: concurrency



- 1) A write
- 2) B write
- 3) A & B write
- 4) A merge begin
- 5) B write
- 6) B merge (skip)
- 7) A merge end
- ...
- 8) B merge

**Note:** this works because the Unix commands **mv** and **mkdir** are atomic!

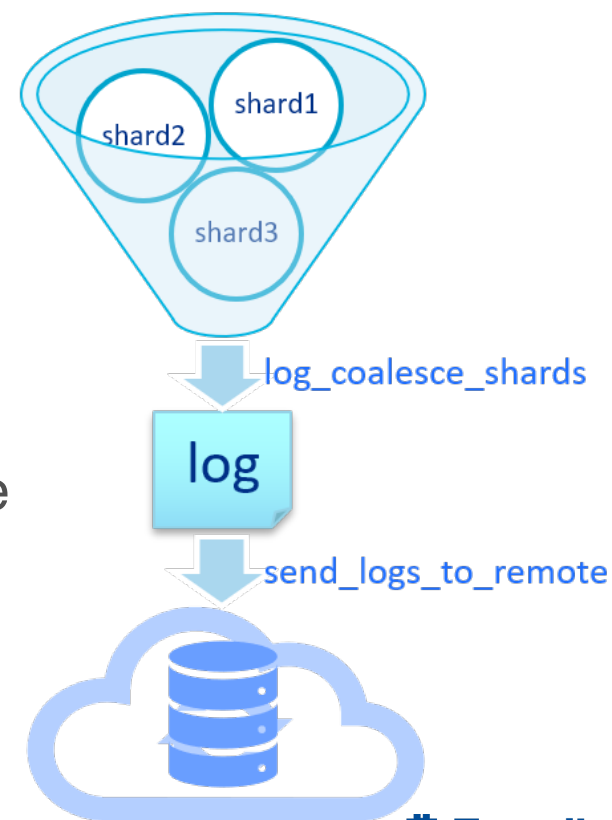
# Feature: remote logging

We have the log finally, but it's not where we need it!

The grid nodes are scattered across the globe, and we usually don't have access to those machines, especially if they belong to different parties.

The new logging system enables alternatives:

- send to the **Factory**: this is easy (just append the log to stderr), but you need access rights to the Factory machine
- send to a **remote HTTP server**: you specify the server(s) URL and the log is forwarded there; definitely handier!





# Issue: security



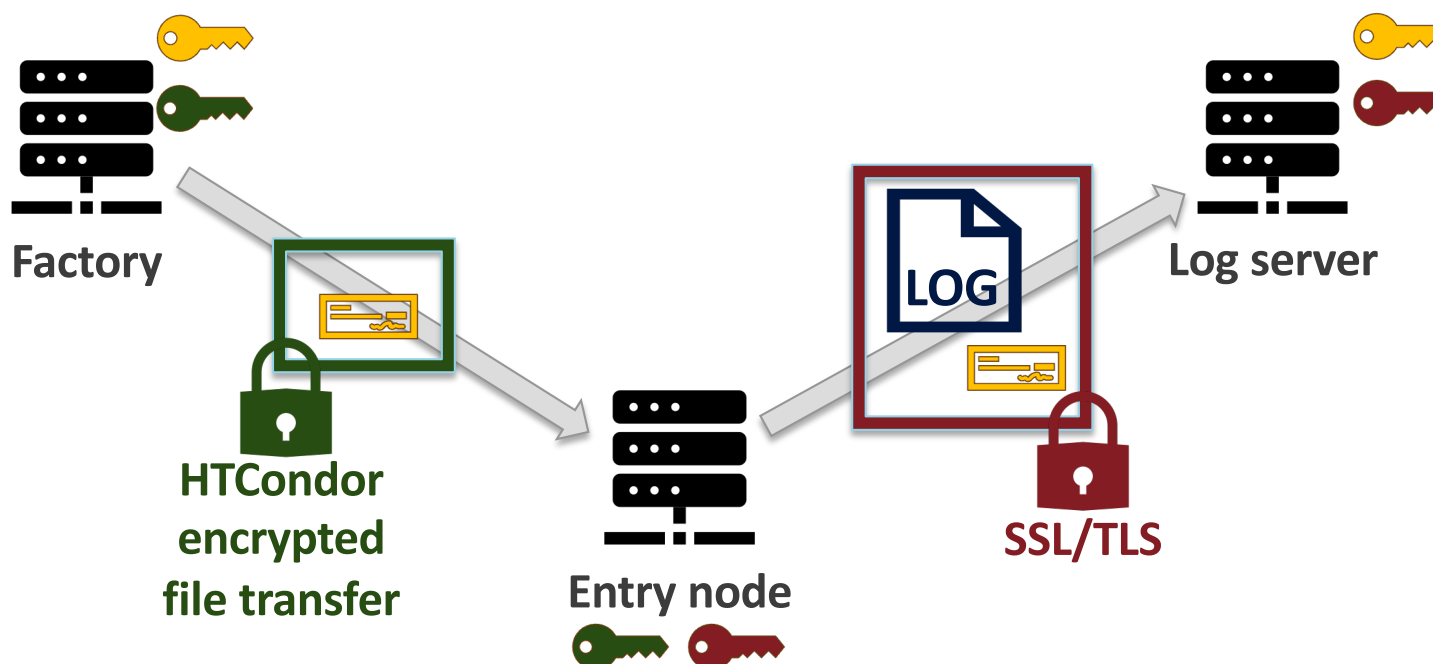
Logs are sent from the entry nodes to the log server over the Internet. Without adopting security measures, an attacker could:

- read the content of the logs
- tamper the content of the logs
- craft arbitrary logs, claiming to be you

You need at least the following:

- **Encryption**: a secret key is needed to decrypt and read the message
- **Authentication**: the recipient can verify the identity of the sender

# Issue: security



## Overview:

- A *JSON Web Token (JWT)* certifies the identity of the entry node
  - The Factory either issues and signs it, or receives it from a Frontend
- The token is sent to the entry node whenever a glidein spawns there, exploiting **HTCondor** built-in encrypted file transfer
- When a glidein posts a log to the server, it must also include its token to authenticate. The whole message (log + token) is encrypted with **SSL**

# Summary

The new glidein logging channel is:

- **Structured:** the new format (JSON) is readable and easy to be processed
- **Versatile:** a line contains either a custom string or the content of a file
- **Available:** any script can use the logging functionalities
- **Independent:** decoupled from stdout/err, autonomous from the process
- **Efficient:** JSON is compact (space efficient). Also, the log are built incrementally, thanks to a mechanism of *sharding* and *coalescing*
- **Thread-safe:** atomicity of the operations is guaranteed even in the presence of multiple glideins or multiple processes
- **Reliable:** the logs can be sent to a remote server upon request, allowing replication and centralized log management
- **Secure:** all data are transferred with cryptographic protocols

# Side activity: self-extracting scripts

The bash scripts need to share some sections of code (functions).

Two typical approaches:

- put the shared code in a **standalone file**; other scripts source it
  - Pro: clean
  - Cons: increased overhead when transferring files to grid nodes
- **Heredoc**: shared code in a string; the main script writes it to a file
  - Pro: constant number of files
  - Cons: messy, syntax highlighting spoiled, hard to test

New approach: **self-extracting scripts**

the code is developed as standalone files, but the utility scripts are archived, compressed and concatenated to the main one before the transfer phase. On the receiving side, when the main script executes, it is able to automatically strip, unzip and share these files.



# Side activity: scripts static analysis

Two paradigms for software testing:

- Static: check the code for potential defects (e.g. undeclared var)
- Dynamic: run the program and check if it works as expected

## ShellCheck

ShellCheck is a [linter](#) for bash scripts, that is a tool that analyzes the code to flag programming errors, bugs, stylistic errors, and suspicious constructs.

ShellCheck integration to the GlideinWMS CI platform is a work in progress.

```
In glidein_startup.sh line 55:  
  for ii in `ls`; do  
    ^-- SC2045: Iterating over ls output is fragile. Use globs.  
    ^-- SC2006: Use $(..) instead of legacy `..`.
```

```
-bash-4.1$ ssh root@fermicloud152.fnal.gov
Last login: Wed Aug 21 12:27:51 2019 from 131.225.67.176
```

WELCOME



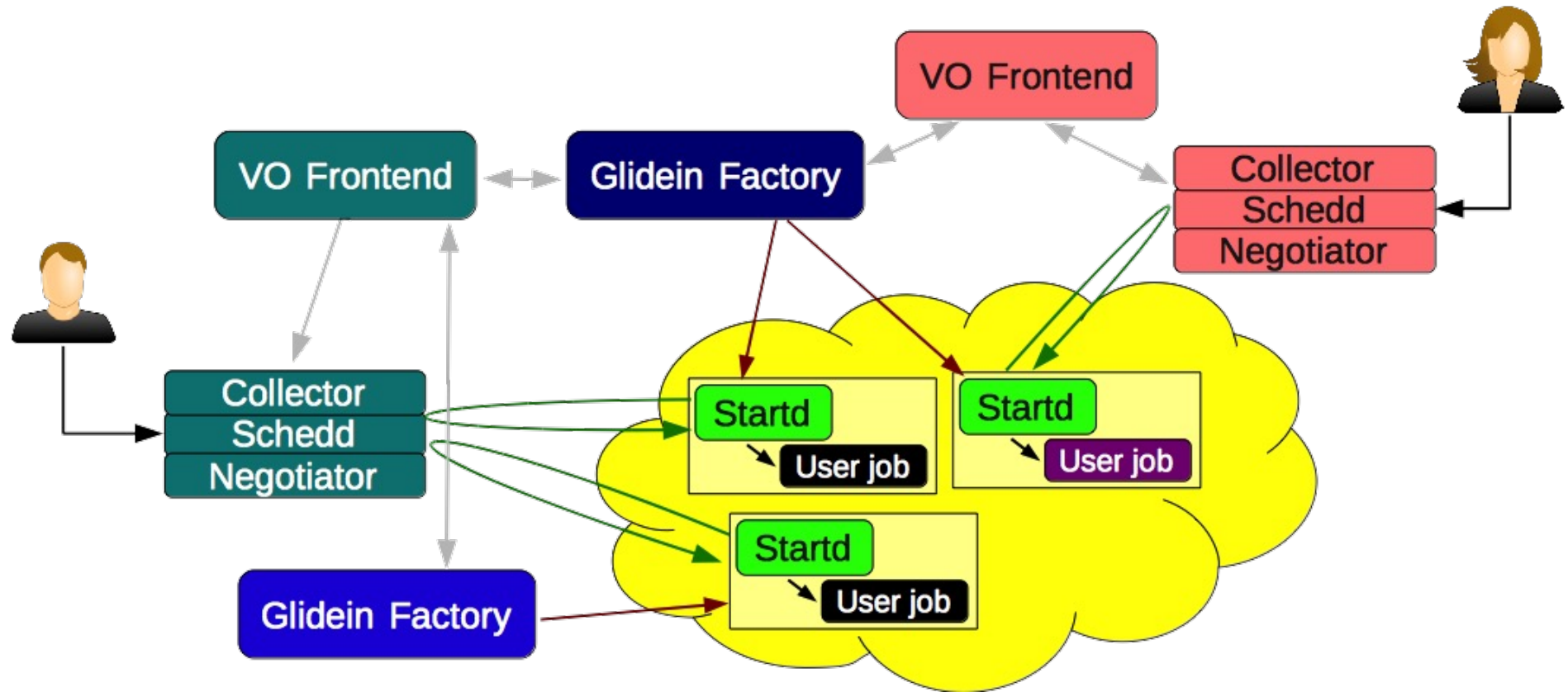
UNIVERSITÀ DI PISA



# Example of log

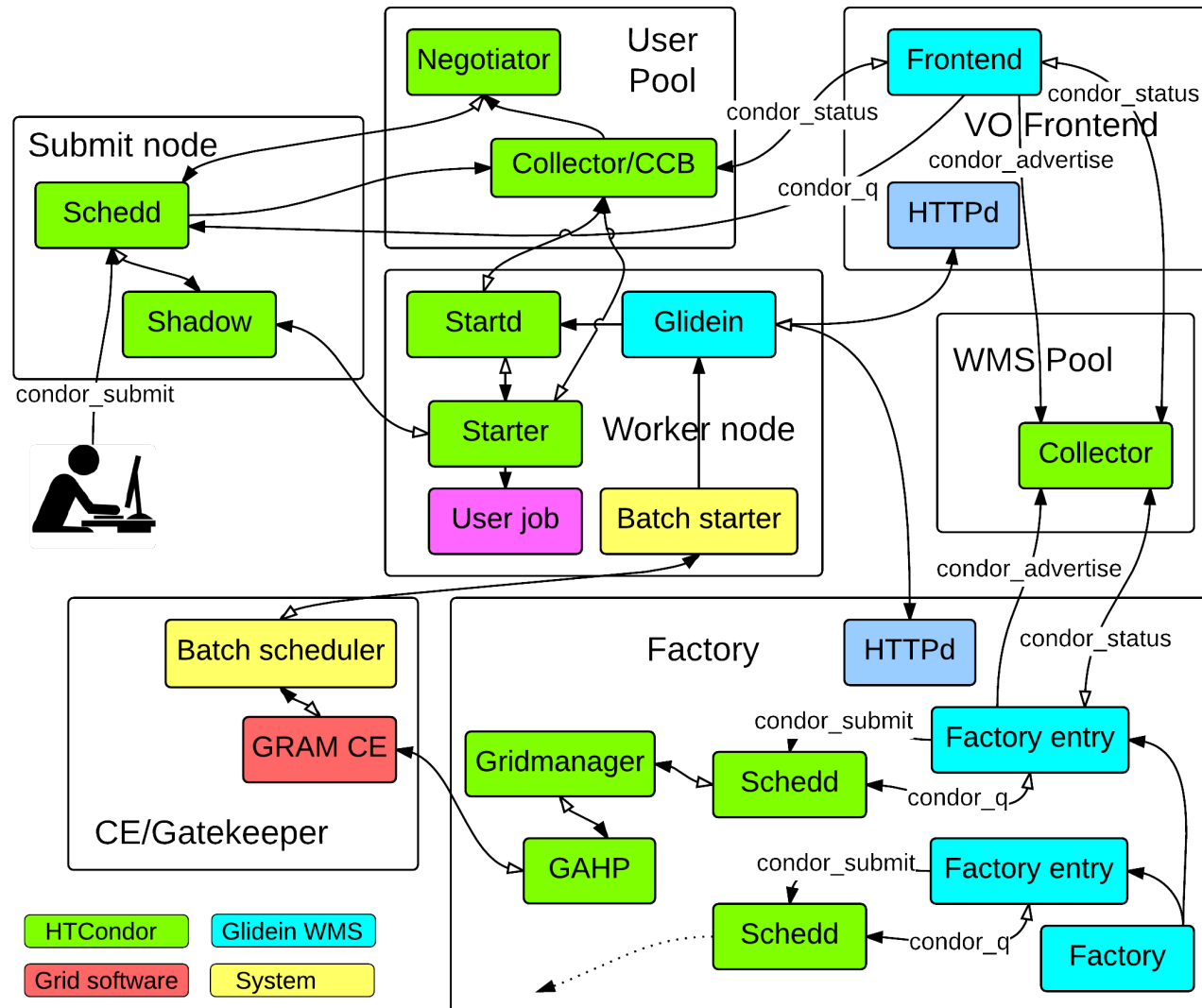
```
[
  {
    "uuid": "a8fd0e49-7a46-4ae6-b188-fc1c5256707e",
    "name": "gfactory_instance",
    "factory": "gfactory_service",
    "client": "fermicloud093-fnal-gov_OSG_gWMSFrontend.main",
    "client_group": "main",
    "schedd": "schedd_glideins4@fermicloud070.fnal.gov"
  },
  {
    "invoker": "glidein_startup.sh",
    "pid": "16070",
    "timestamp": "2019-08-19T17:02:06-05:00",
    "severity": "info",
    "type": "text",
    "filename": "",
    "content": "Remote logging has been setup. Server address: http://fermicloud152.fnal.gov:80"
  },
  {
    "invoker": "glidein_startup.sh",
    "pid": "16070",
    "timestamp": "2019-08-19T17:02:17-05:00",
    "severity": "warn",
    "type": "text",
    "filename": "",
    "content": "curl not installed on this machine"
  },
  {
    "invoker": "glidein_memory_setup.sh",
    "pid": "29801",
    "timestamp": "2019-08-19T17:02:18-05:00",
    "severity": "debug",
    "type": "file",
    "filename": "logging_utils.source",
    "content": "begin-base64 644 -H4sIAGocW10AA+MCAJMG1zIBAAAA===="
  }
]
```

# GlideinWMS





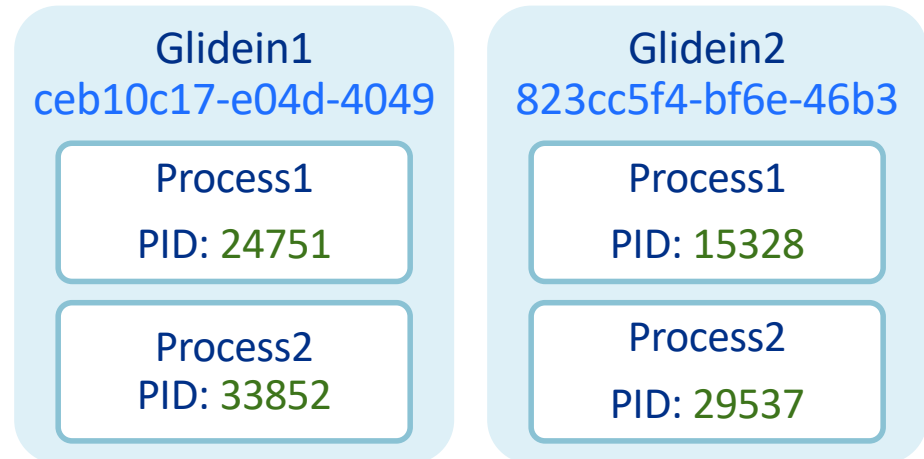
# GlideinWMS



# Sharding + identification

A number of mechanisms make sure that different glideins/processes do not interfere with each other.

A **UUID** (Universally Unique Identifier) is assigned to every glidein at the time of its creation, to distinguish it from the other glideins. Within the glidein, the **PID** (Process ID) discriminates between concurrent processes.




Every shard is basically a separate file, so that location is written once by a single process.

Shards are carefully moved between folders during the various stages (writing, merging, removal, ...), in order to avoid undesirable scenarios like removing a shard while it's being written.

# API: log\_write

A new utility function called `log_write`, available to the glidein bash scripts, adds a new log entry that contains either a string or a file.


```
log_write <invoker> <type> <content/file> <severity>
```




*name of the  
script writing  
to the log*



*text or  
file*



*string content  
or file name,  
depending on  
the type*



*severity level:  
debug, info,  
warn, error*

When invoked, `log_write` creates a new file (*shard*) for the entry, containing all the above attributes plus some other metadata (timestamp, PID, ...).

Files are compressed then UUencoded before being inserted in the shard.