SOSC 2022 Fourth International School on Open Science Cloud

Parallelize and distribute workloads: Workflow management

Daniele Spiga, INFN-Perugia
spiga@pg.infn.it

28 November 2022 to 2 December 2022
INFN Perugia

Why this session

- An overview about: exploiting computing resources when they are installed on a distributed environment
  - Local Cluster of resources
  - Geographically distributed/dispersed
- To discuss main technical challenges (high level)
  - Trying to see them from a user perspective (reducing underlying technical details)
- To introduce the next session and thus the final part of the school

Sometime I'll try to show the relationship with national activities and opportunities

spiga@pg.infn.it

Agenda

General introduction and few definitions

Going distributed: What are the challenges
- Few more definitions
- Few examples, best practices

What are Batch systems
- And then going beyond

And here we enter the next session. The core business of the day

spiga@pg.infn.it

Introduction

Assuming that we are able to manage our working environment to deal with the software that we need locally, dependencies, input data and, of course, produced results

Our next objective is to being able **to scale up to a full workstation and transparently scale out to a distributed computing environment**

The ambition today to address few basics aspects such us

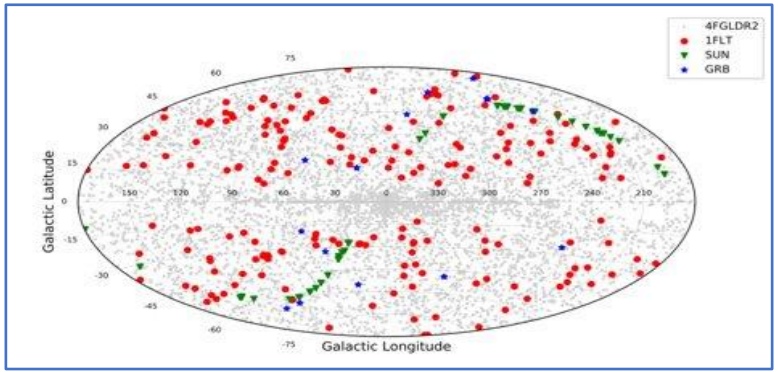
- What does it mean to parallelize and to distributed our payload
- What are the main challenges (or better the first mandatory things to be aware of)
- What are the opportunities at national level (INFN)
- Accessing and exploiting distributed resources
- A few best practices

spiga@pg.infn.it

What is the problem we'd like to address

- The LAT instrument onboard of Fermi gamma-ray science telescope (Atwood et al 2009) observes the sky in the gamma rays range between 30MeV - 300 GeV since August 2008.
  - Extract catalog of transient sources (monthly basis) from Fermi-LAT data (1FLT; Fermi-LAT collaboration in preparation)
    - 10 years of data in monthly time scale—> 120 independent skies + 120 (15-day shifted month)
  - Detected ~1000 seeds/monthly skies
    - ~260 binned maximum likelihood analysis (ML) for each month.

- **Submitted roughly 60k ML analysis jobs ~ 960 h of computing time without interruptions.**



Aitoff projection of 1FLT in red, sun detections from transient catalog in green, and GRB detection in blue. Standard catalog (4FGLDR2) in gray

507 new detections —>extraction of standard products: monthly light curves **(120 ML jobs per source)** and Spectral energy distributions **(4 ML jobs per source)**

Scaling out to a cluster allowed us to gain a factor of 10 (roughly) in time. This implies a faster turnaround in the data analysis steps **(VERY PROMISING)** **Dr. Sara Cutini**

FERMI GAMMA-RAY SCIENCE TELESCOPE

spiga@pg.infn.it

Distributed computing

Computing represents a **limiting factor for the quantity** (as well as **the type**) of the physics you can do in an experiment.

- I.e. computing can be a limit to the events collected by a HEP experiment such as CMS.

Distributed computing is about to use many computers, each running one instance of our program

Example:

- **1 laptop** (1 core) => 12,000 hrs = **~1.5 years**
- **1 server** (~40 cores) => 750 hrs = **~2 weeks**
- **1 MPI job** (400 cores) => 30 hrs = **~1 days**
- **A whole cluster** (10,000 cores) = **~1 hour**

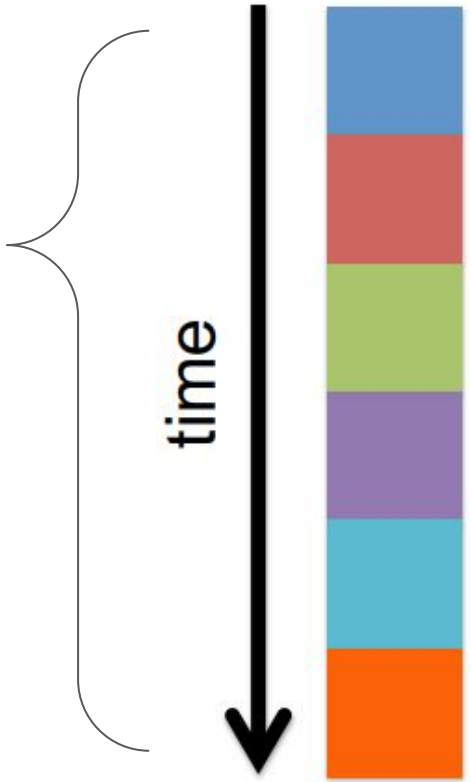
Requires thinking of work in a different way (splitting jobs; optimizing for throughput rather than single-job performance).

spiga@pg.infn.it

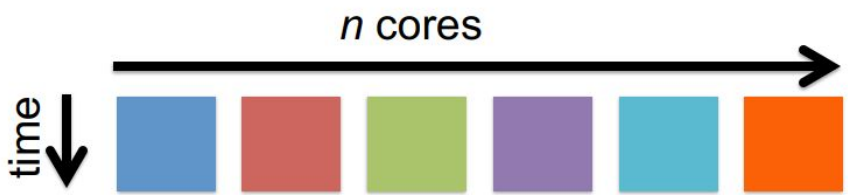
Parallelize the execution

Serial execution means running **one task at a time**. The overall compute time grows significantly:

1. as **individual tasks get more complicated** (long)
2. if the **number of tasks increases**



If you **parallelize** your tasks you can create **many Independent unit** and run them on different cores



So far…

We introduced two concepts and, implicitly, a lot of challenges.

Distributed computing capacity:

- Need to prepare the computing environment.
- Need someone that organize it for us
- Need a unique interface to many execution hosts (server)
- …

Resource management

Parallelization and payload organization

- Preparing the task to execute
- break/split it in to sub-task (parallelize)
- Distribute payloads over distributed computing capacity
- …

Workload Management

spiga@pg.infn.it

A high level schema

Resource gathering, and management of resource (abstraction layer)

Queue of payloads, dispatcher, bookkeeping..

Distributed computing resources

[HTC, HPC, Coud…]

Might depend on specific implementation

Resources provisioning

Harvesting resources for the end users is quite a complex system admin job.

- Geographically distributed systems, heterogeneous system (in term of hardware, policies etc)
  - It might be cloud, HPC, HTC…

The objective of the session is on **how to use resources** rather than **how to provision resources**

- Trying to highlight the major challenges we face to use distributed systems

spiga@pg.infn.it

HTC vs HPC (in a nutshell)

High-Throughput Computing (HTC)

Many problems require years of computation to solve

- **computational power over a long period of time**.

If problem is made of **individual tasks** that do **not need to interact** while running

- Embarrassing parallel computation

High Performance Computing (HPC)

In contrast: if the computation needs

- **Frequent** and **fast** exchanges of intermediate results
- tremendous amount of compute power
- **over a short period of time.**

HPC environments are often measured in terms of Floating point Operations Per Second (FLOPS). Benefits greatly from:

- CPU speed, homogeneity, shared filesystems, Fast networking (e.g. Infiniband)

spiga@pg.infn.it

HTC vs HPC In a nutshell

high-throughput | high-performance

high-throughput

FLOPY

$$\left(\frac{\sum Completed\ Job\ Runtime}{Wall\ Time}\right)^{*}$$

- Several indepenent "single node" programs
- Independent Computing nodes

high-performance

FLOPS

- Huge multithread programs
- Span through several nodes
- The whole system has to be thought of as a single "computer" with an internal fabric joining it all up.

spiga@pg.infn.it

And where is the Cloud?

It depends on what we mean actually mean with “Cloud”

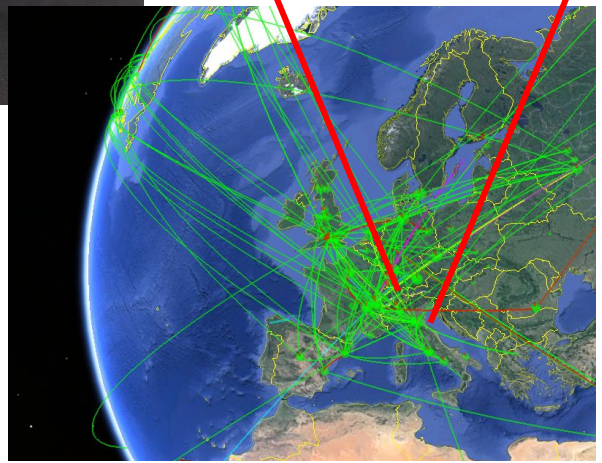
If we consider the cloud as an abstraction to the underlying fabric layer

- The cloud is a **technological mean to provide**, among other services, either HTC or HPC.
  - A convenient interface to provide HTC/HPC resources
- In the real life It is a bit more complicated than that
  - i.e. depends on the **underlying Hardware**
  - There are a lot of **cloud-native solutions** such those you’ve seen on Tuesday
    - (Argo, MinIO) etc and that you will see today

HPC, HTC, Cloud.. It is a matter of ~~taste~~ use case (type of computing needs)

spiga@pg.infn.it

The National (IT) Landscape today



spiga@pg.infn.it

One of the Challenge: the continuum

We'd like to integrate all them to hide technicalities and facilitate the exploitation

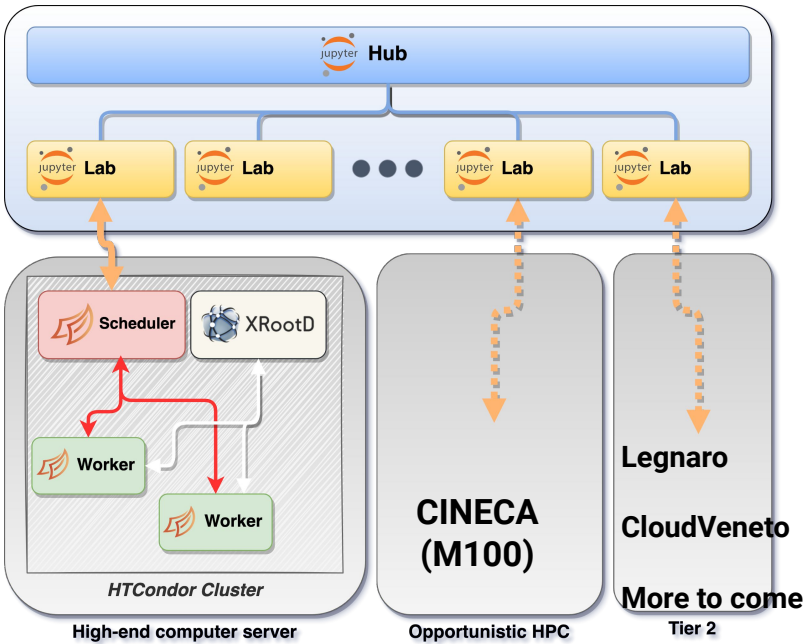
The ICSC aim and objectives

Create the **national digital infrastructure** for research and innovation, starting from the existing HPC, HTC and Big Data infrastructures ...

... evolving towards a **cloud datalake** model accessible by the scientific and industrial communities through flexible and uniform cloud web interfaces, relying on a high-level support team ...

... form a globally attractive **ecosystem based on strategic public-private partnerships** to fully exploit top level digital infrastructure for scientific and technical computing and promote the development of new computing technologies

From Davide (Monday)



Ok we've got our resources let's use them

Challenges

Software distribution

- **Objectives:**
  - Learn about and use software portability techniques
  - Understand the basics of distribution and accessing software and libraries in a distributed
    - the implications for Distributed environment

Data access

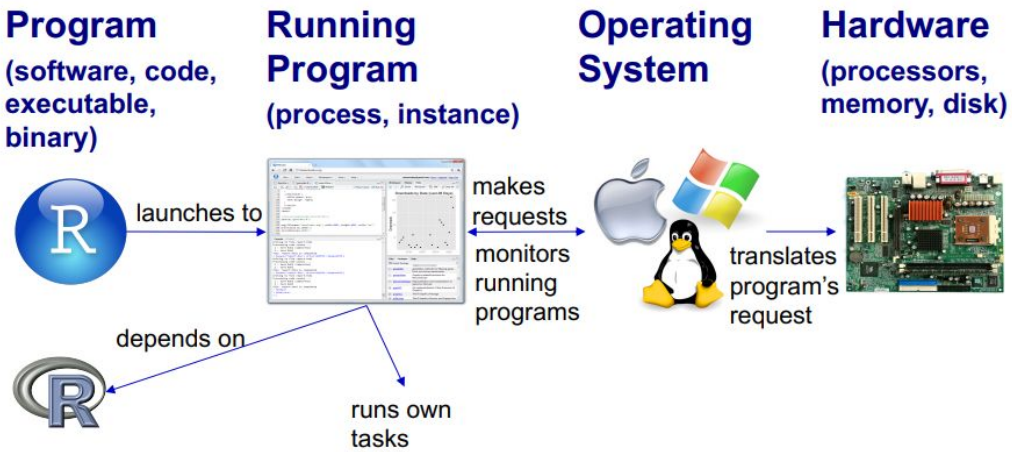
- **Objectives:**
  - Define what data access is about (input and output)
  - Possible patterns to access data in a distributed environment
    - And a few best practices

spiga@pg.infn.it

The problem

The principle

- Software is a **set of files**.
- These files have **instructions for the computer to execute**
- Moreover: Software **depends on the operating system, and other** installed **programs** (dependecies)



distributed environment implication

- Software must be able to run on **target operating system** (usually Linux). -
- Know **what else your software depends on**

Isolate the specific software files needed for a programme and bring them along

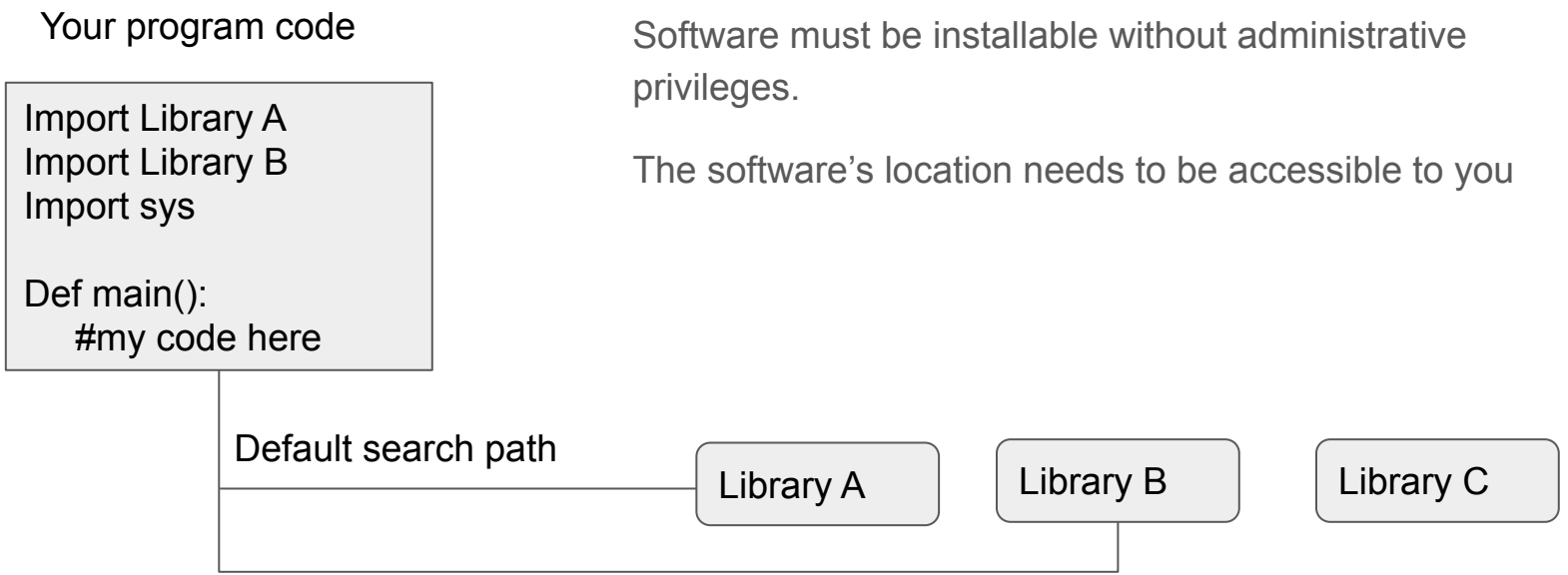
spiga@pg.infn.it

Software dependencies

Software files have to be installed somewhere in the file

Software must be installable without administrative privileges.

The software's location needs to be accessible to you



spiga@pg.infn.it

Why it is a problem

When you are on your pc (in principle) You have full control.

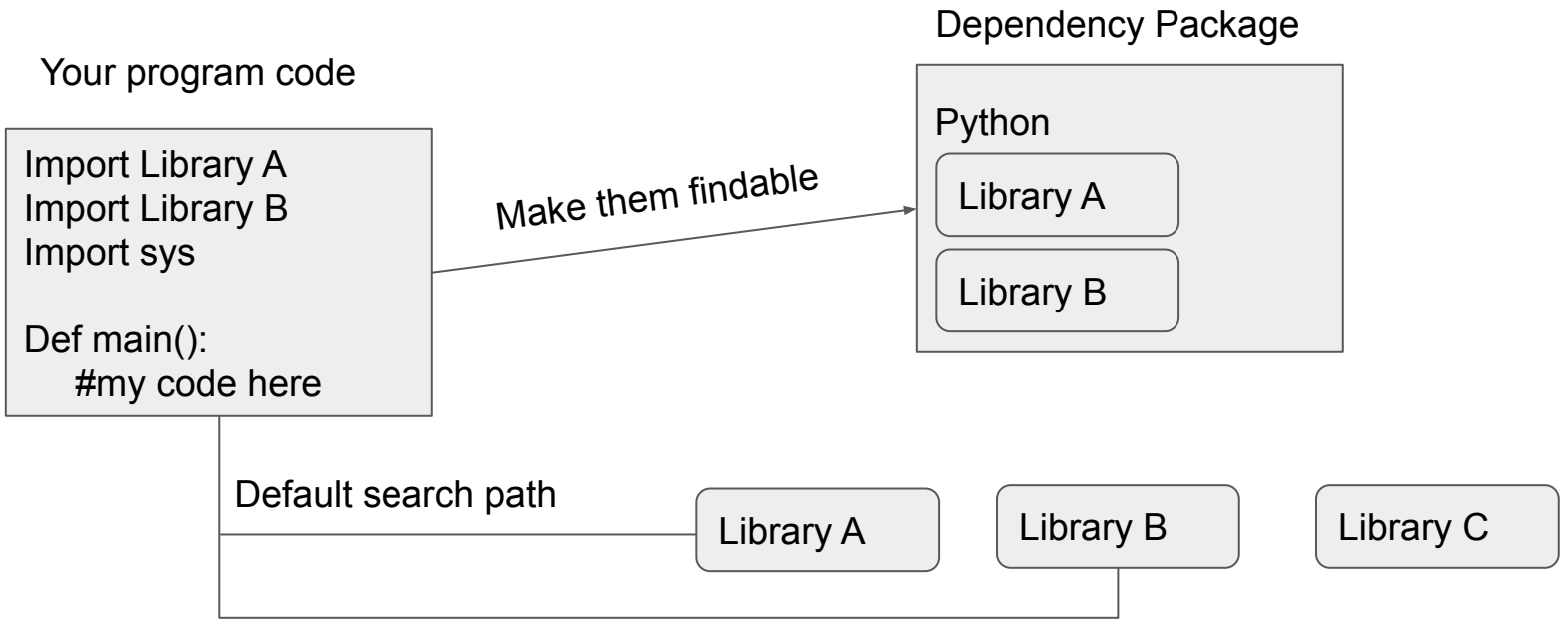
- You know what you already have
- All the software you need is already installed.
- You know where everything is (mostly).
- You can add new programs when and where you want.

If you start using someone else pc (in principle) you've not full control.

- What's already there?
- Is R installed? Or Python
- What about the packages you need?
- Do you know where anything is?
- Are you allowed to change whatever you want?

spiga@pg.infn.it

Understand and collect dependencies



spiga@pg.infn.it

The solution: Software Portability

Take your software with you Install it anywhere and Run anywhere

Run “anywhere” by:

- bringing along the (Linux-compatible) software files you need…
- to a location you can access/control…
- telling the command line where that location is…
- and using it to run your code.

Easiest case:

- Job written in a single compiled language (Link Statically)

Harder case:

- jobs with a combination of languages, libraries, scripts

Davide on Monday already told you a technical solution to this problem: containers

spiga@pg.infn.it

Containers (recap)

Containers are a tool for **capturing an entire job "environment"** (software, libraries, operating system) into an "image" that can be used again.

Why use containers instead of compiling/installing code

- **Permissions**: Software that can't be moved, do files or libraries have to be at a specific path?
- **Complex installations**: software that has a lot of dependencies or components.
- **Sharing with others**: one container can be used by a whole group that's doing the same thing.
- **Running on different systems**: The same container can run on Linux, Mac and Windows
- **Reproducibility**: save a copy of your environment.

spiga@pg.infn.it

Container (cont)

To use a container as your software portability tool, need to either:

- **Find a pre-existing container with what you need.**
  - Dockerhub is your friend
- **Build your own container**
  - Docker file is the keyword (Davide's lesson)

Two common container systems:

widely used container system for HPC

spiga@pg.infn.it

Another path… pre-existing software

Like if you could execute your programme in a distributed environment, **working in the assumption all the needed software is pre-installed everywhere** (in the same location/PATH)

Ideally:

- you distribute software as simple as you install it in your laptop
- Find it, everywhere, same path, same library no dependency issue
- Keep everything always up to date and in synch

Access the same software as if it is in the local working station

spiga@pg.infn.it

This is possible: CernVM-FS

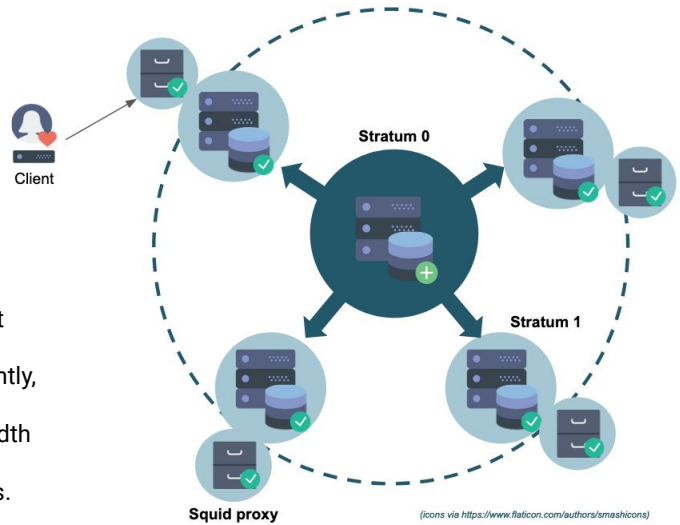
The CernVM File System (CernVM-FS) **provides a scalable and reliable software distribution service**, which is implemented as a **read-only POSIX filesystem in user space** (a FUSE module).

- Files and directories are hosted on standard web servers and mounted in the universal namespace **/cvmfs.**

How is made (from 10km far)

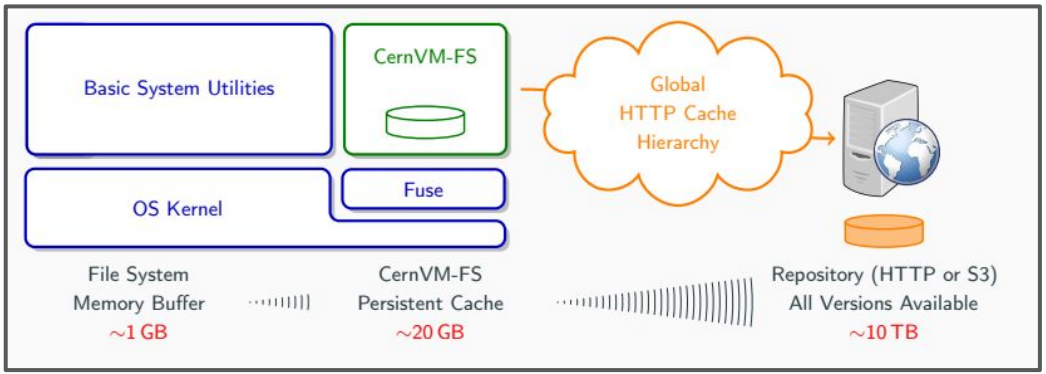
- the central *Stratum 0* server which hosts the filesystem;
- the *Stratum 1* replica servers, and the associated *proxies*;
- the *client* accessing the filesystem provided via CernVM-FS.

A Squid proxy caches content that has been accessed recently, and helps to reduce bandwidth and improve response times.



How it works from a user perspective

Populate and propagate new and updated content



A few “software librarians” can publish into **/cvmfs**

All content in /cvmfs is cryptographically signed

Transactional writes as in git commit/push

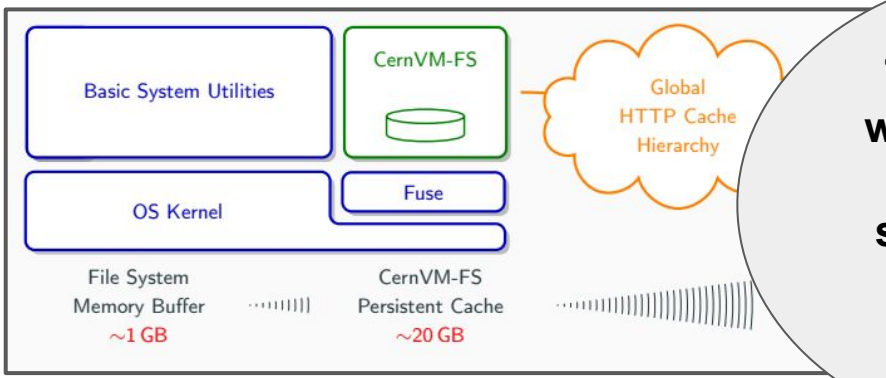
CernVM-FS provides uniform, consistent, and versioned **POSIX file system access to /cvmfs**

```
$ ls /cvmfs/cms.cern.ch
slc7_amd64_gcc700  slc7_ppc64le_gcc530  slc7_aarch64_gcc700  slc6_mic_gcc481
...
```


spiga@pg.infn.it

How it works from a user perspective

Populate and propagate new and updated content



A few “software librarians” can
ublish into **/cvmfs**

ntent in /cvmfs is
graphically signed

sactional writes as in git
mmit/push

This is true and working on grids, clouds, supercomputers and end user laptops

```
$ ls /cvmfs/cms.cern.ch
slc7_amd64_gcc700  slc7_ppc64le_gcc530  slc7_aarch64_gcc700  slc6_mic_gcc481
...
```

CernVM-FS provides uniform, consistent, and versioned **POSIX file system access to /cvmfs**

Recap

CernVM-FS is a ***read-only*** filesystem for those who access it : This is you

Who administer the **stratum 0** is able to add or change the contents.

- Someone should allow you (or your friend) to write on a stratum 0

What is the most important message to remeber so far:

- You know it exist and what it does
- You ask for support if you think you need it for your research
  - To who? **A possibility is INFN-Cloud**

Few references: https://cernvm.cern.ch/fs/ ; https://github.com/cvmfs/cvmfs

Container and CVMFS

Can works together, ideally use containers for isolation and orchestration, and CVMFS for distribution

- Container are fine for distribution but no necessarily ideal

If **/cvmfs** is available on the host

- Bind mount from host to container as an external volume (Davide told you about this)

Container Images can be on /cvmfs, particularly suitable for **large scale distribution** but not only... I think that this become a ideal configuration

- Example1: **unpacked at CERN** is a service that unpacks Docker images and makes them available via a dedicated CVMFS area. Images will be automatically synchronised from the image registry to the CVMFS area within a few minutes whenever you create a new version of the image.
- Example2: **singularity.opensciencegrid.org**
  - /cvmfs/singularity.opensciencegrid.org/centos/pyt hon-34-centos7:latest

spiga@pg.infn.it

We (at INFN) are working for you :)

INFN-Cloud portfolio of solution will be extended with two main services

1. Automate the CVMFS stratum 0 setup and configuration:

- Translation: user doesn't need to know how to setup such a system. Push a button, the cloud system provides a personal server. In turn you become librarian of yourself/your group
  - Use the client everyware

2. Enable everybody to be librarians hiding completely the interaction with CMVFS stratum 0

- Dropbox like approach

WORK IN PROGRES

spiga@pg.infn.it

Wrap-up

Create/find software package. Account for all dependencies, files, and requirements and then options:

- download pre-compiled code
- compile your own on the node
- create/find a container
- use CVMFS

In addition, in the real life script to set up the environment on a remote host often is needed,.

- I.e. you might need to play with software paths etc.

spiga@pg.infn.it

Data handling

"**Input**" includes any files needed for the job to run

- executable
- **data to process**
- (and software)

"**Output**" includes **any files produced** that you need to come back

- Output results
- Log, error

What matter today is the **data to process**

- I.e. the data that was obtained from the Kepler mission (photographies)
- Experiment data (remember what Tommaso told us on Monday)

spiga@pg.infn.it

Input data size: Analogy

What method would you use to send data to a collaborator?

amount	method of delivery
words	email body
tiny – 100MB	email attachment (managed transfer)
100MB – GBs	download from Google Drive, Drop/Box, other web-accessible repository
TBs	ship an external drive (local copy needed)

Moving data to a distributed computing environment can be less friendly than that…

spiga@pg.infn.it

Input data handling: possible choices

Data can be **shipped to the execution host** (a node of the distributed environment) together with the executable

- **Via input sandbox**
- This is ok for **testing purposes and more in general when you deal with (very) small input data**, otherwise you will hit:
  - Hardware transfer limits
  - Hardware storage limits (spool)
  - Network bottleneck

Data can be **pre-placed** on the execution host

- This can be done manually by end users (scp, rsync). Ineffective and time consuming, error prone
- So what? The answer is: **Data Management** Not part of the lecture nor of the school [see later]

spiga@pg.infn.it

A path forward: Download or stream data

You are aware of cloud storage (you saw S3 and played with MinIO)

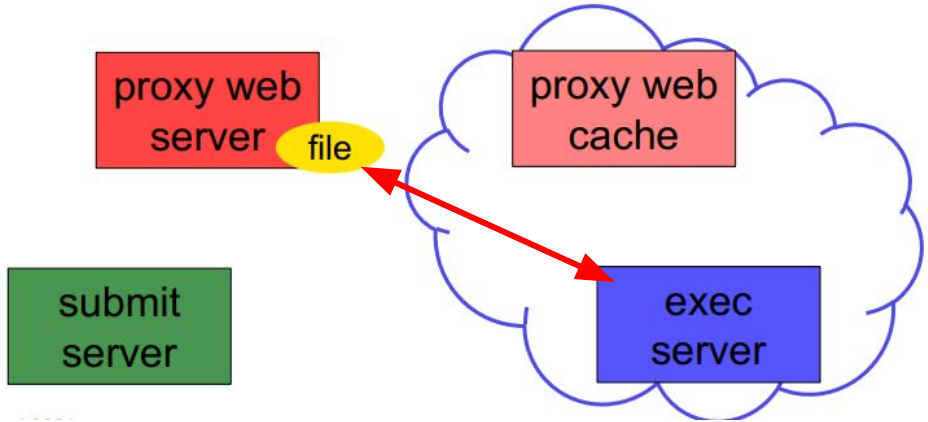
- Other protocols are also possible as well as other technologies

Data (files) **can be placed onto a Cloud Storage** and programme (execution servers) get data from there

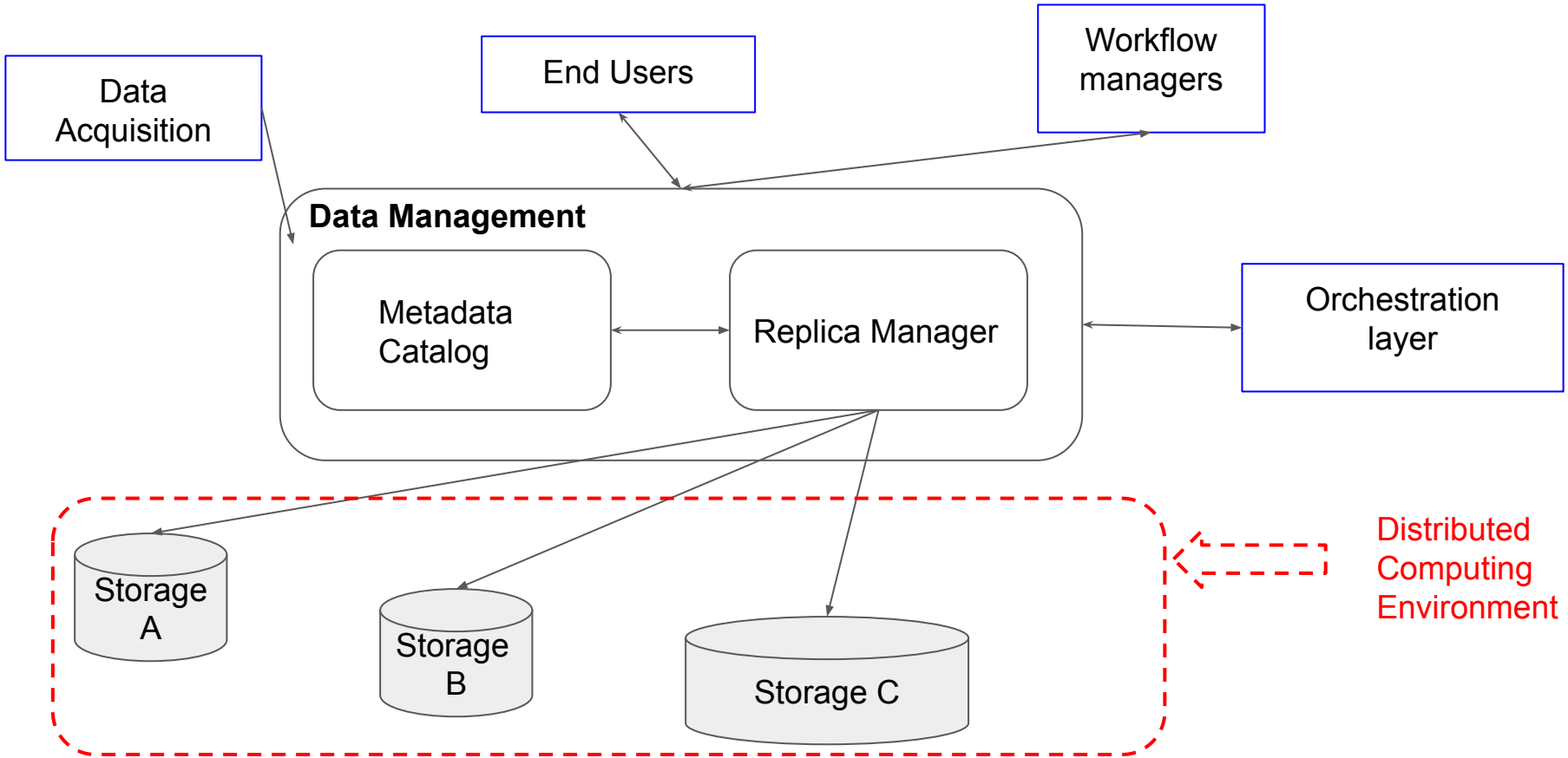
- Simple example the presigned url

Once data are in a storage:

- **Lazy download** to the exec host
- **Remote data read** (stream)



Data Management: a Sketch



Ok, we've resources, software and data

What's next? **How can one exploit these distributed resources**…

I.e Using a **batch system:** This is the most common (in HEP but may scientific domain) system to distribute work although not the unique [see later]

Why a batch system, **advantages**:

- Users **don't have to worry about exactly which machines are available** for use
- Grants that **jobs will not interfere with each other** by running on the same machine
- Provides a **central point of control for** enforcing scheduling an sharing **policies**

spiga@pg.infn.it

A de facto standard: HTCondor

HTCondor: it is a de facto standard that we can find everywhere

- INFN Cloud is offering the possibility to generate a HTCondor batch system on demand [see later]
- Most of the INFN facilities uses HTCondor
- Most of the LHC Grid is made of HTCondor

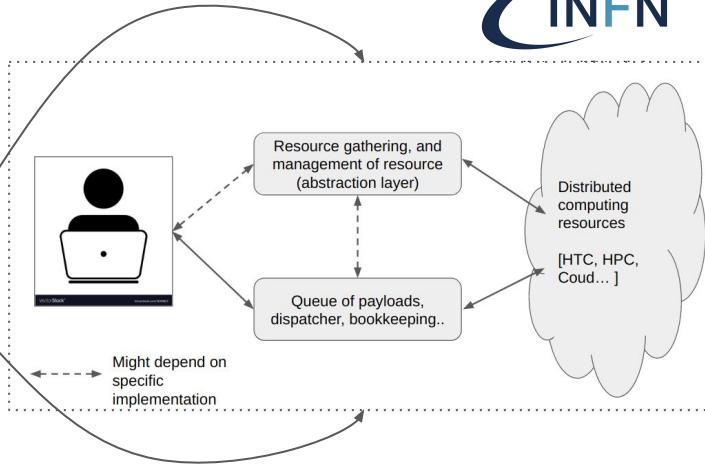
However it is not the unique solution. Most of the HPC centers uses Slurm

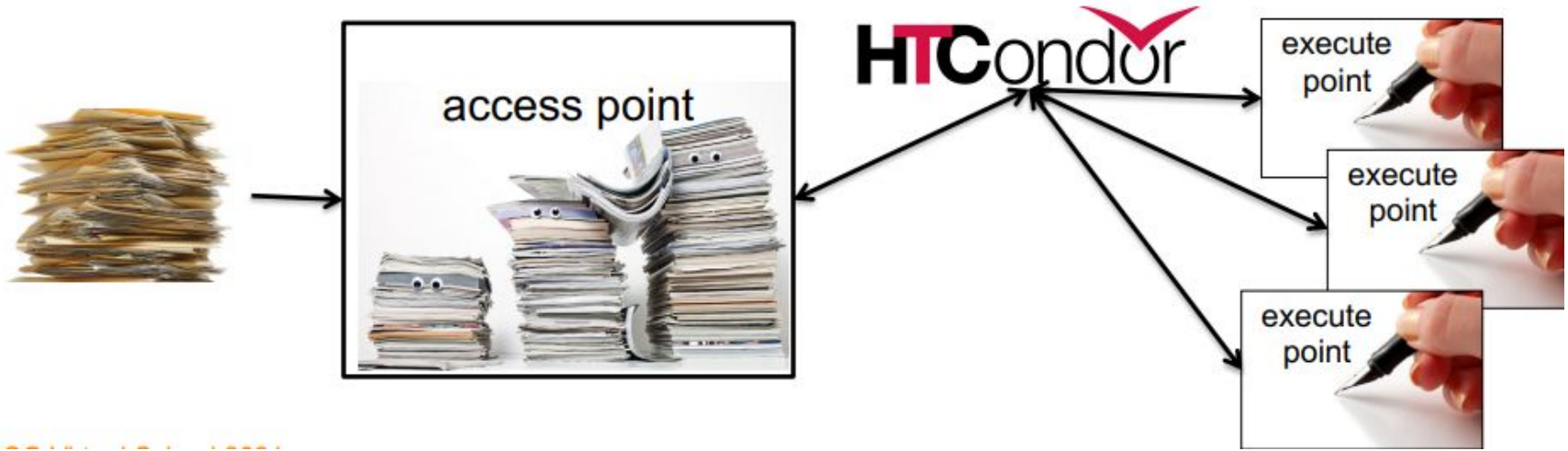
- If you login to the CINECA (example) you need to interact with Slurm.

spiga@pg.infn.it

HTCondor from 10km far

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





Terminology

Job: An independently-scheduled unit of computing work

Three main pieces:

- **Executable**: the script or program to run
- **Input**: any options (arguments) and/or file-based information
- **Output**: files printed by the executable

In order to run many jobs, executable must run on the command-line without any graphical input from the user

Machine

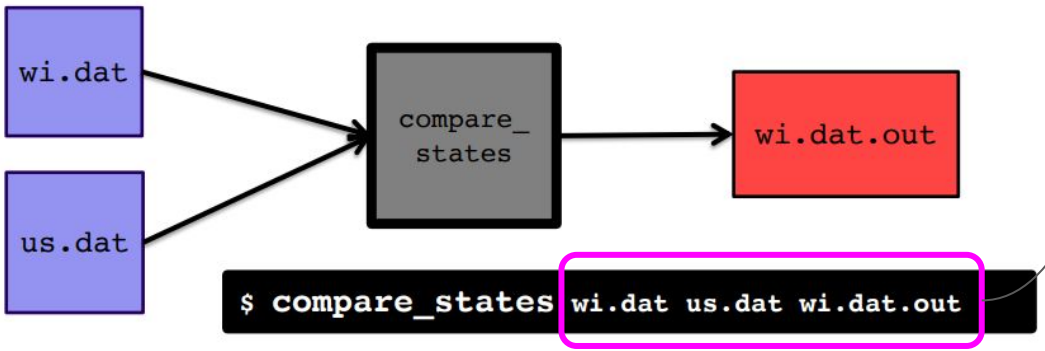
A whole computer (desktop or server)
Has multiple processors (CPU cores), some amount of memory, and some amount of file space (disk).

- **Slot**
  - an assignable unit of a machine (i.e. 1 job per slot)
  - may correspond to one core with some memory and disk
  - a typical machine will have multiple slots

spiga@pg.infn.it

HTCondor: Job Example

We have a program called **“compare_states”** (executable), which compares **two data files** (input) and produces a **single output file**



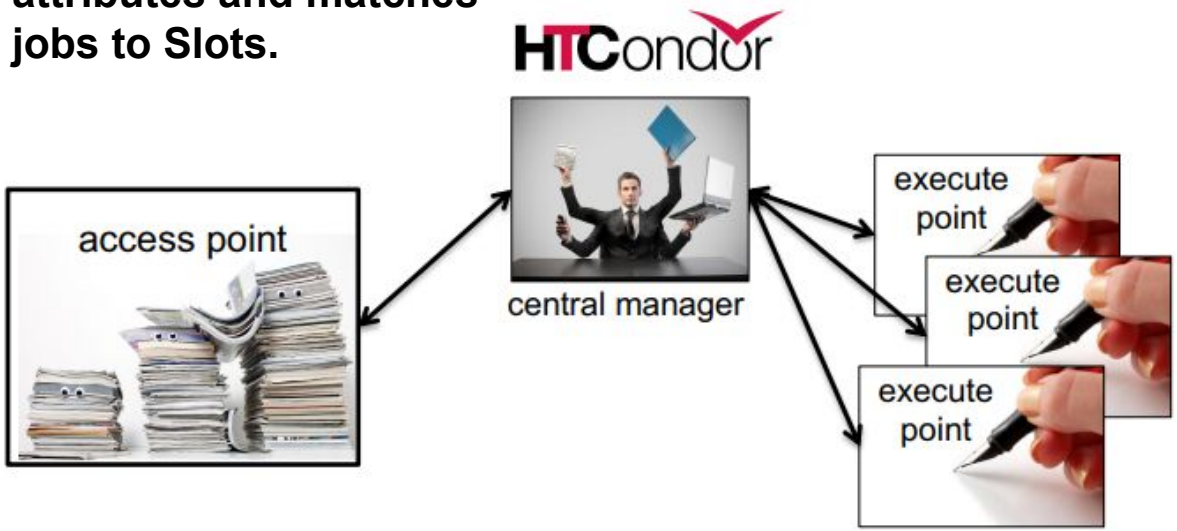

```
executable = compare_states
arguments = wi.dat us.dat wi.dat.out

transfer_input_files = us.dat, wi.dat

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

request_cpus = 1
request_disk = 20MB
request_memory = 20MB

queue 1
```

Input sandbox

Resources requests and Match making

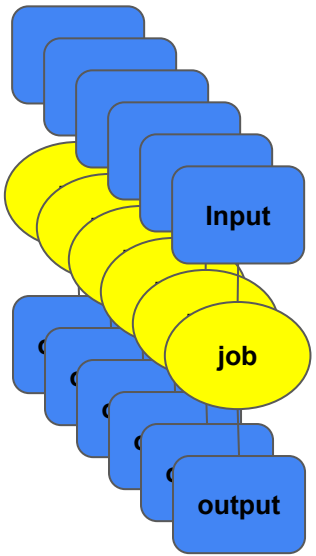
On a regular basis, **the central manager reviews Job and Machine attributes and matches jobs to Slots.**

Very important to request appropriate resources (memory, cpus, disk)
- 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” than they could, and you’ll block other jobs




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

transfer_input_files = us.dat, wi.dat

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

request_cpus = 1
request_disk = 20MB
request_memory = 20MB

queue 1
```

Best Practices: Try to minimize your data



Eliminate unnecessary data

Split large input for better throughput

File compression and consolidation

- I.e. tar/zip are your friend
- Do it prior to moving data between your laptop and the submit server

spiga@pg.infn.it

Best Practices: throughput

Suppose you want to analize **1M documents**, and each analysis takes **5s**.

- **Bad idea:** Analyze one doc per batch job
- Why? Rule of thumb: 30s to start one job

- **Better idea:** analysize 100 docs per batch job.
- Each job takes 500s (about 8 min)

- Throughput is 12 docs/sec* number of machines

spiga@pg.infn.it

Be careful: scale up by degrees

A batch system is merely an amplifier:

- You are going to multiply 1000000 x Success or **1000000 x Failure**!

Get your jobs and data organized first

- Then, run one job and verify the results
- Then, run 10…
- Then, run 100 jobs..

Each order of magnitude is likely to have a new problem:

- Performances of queueing system
- Load placed on network
- Limit on number of file in a directory
- Correctness of the code when run with new parameters
- Make happy the sys admins :)

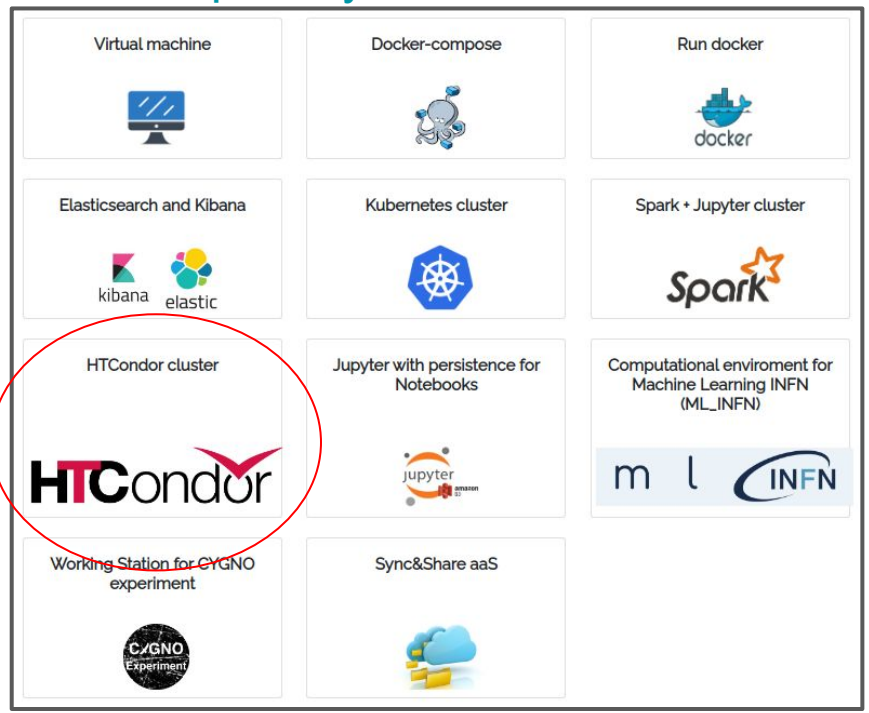
spiga@pg.infn.it

Try it out yourself

https://my.cloud.infn.it/

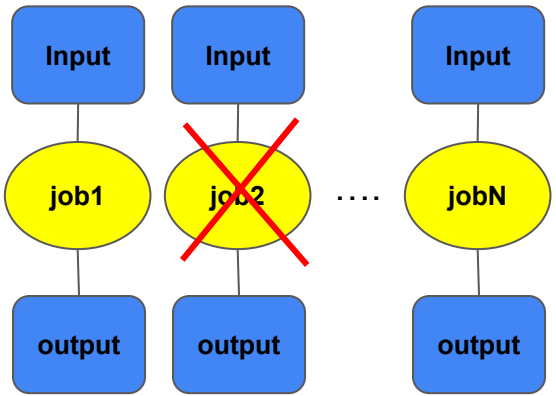
You can apply almost all what has been discussed via INFN-Cloud

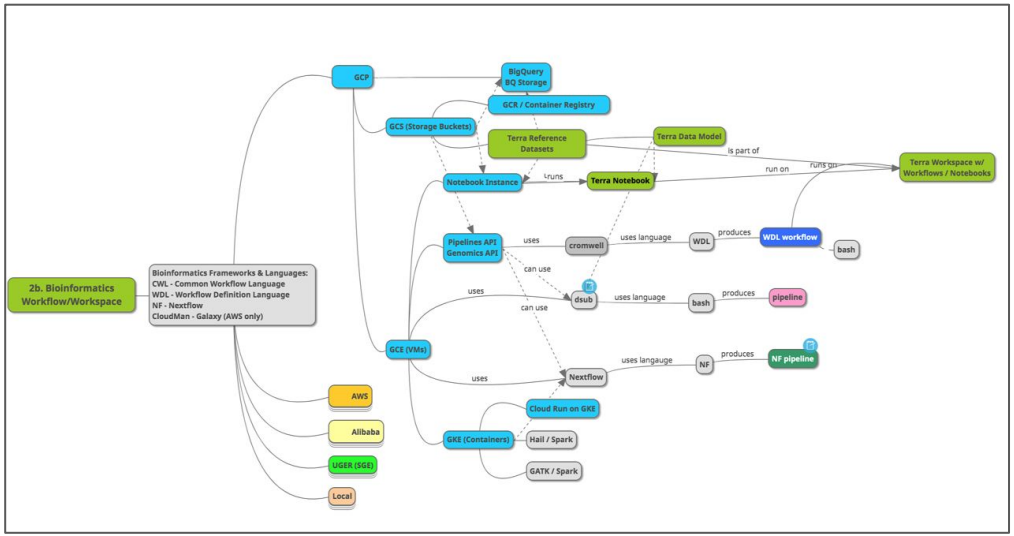
- istanciate yourself a HTCondor batch system on Cloud
- Do your first splitting, parallelisation



https://guides.cloud.infn.it/docs/users-guides/en/guide_htcondor/users_guides/howto17_htcondor.html

Ok but what about…





If some of the jobs start failing?

Need to care about them,

- To deal with error handling and perhaps resubmission (which in turn means bookkeeping) etc

If a workflow is complex enough i.e. describes a large number of batch jobs each with their own complex details and inter dependencies

Workflow Management System

A workflow is a description of a large number of batch jobs and files with internal dependencies

A Workflow Management System (WMS) is a software that executes a workflow by dispatching it to a batch system

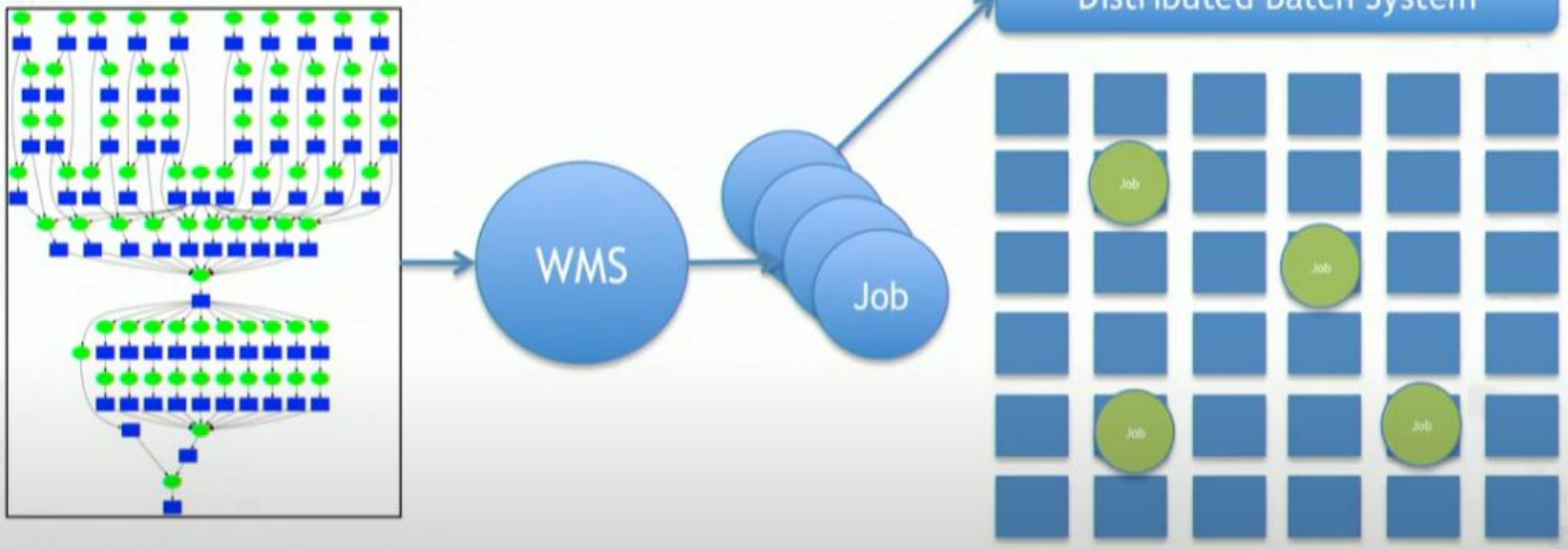
A scientific workflow management system is a system that supports

- specification, modification, execution,
- failure handling
- and monitoring

of scientific workflows using workflow logic to control the order of executing workflow tasks.

spiga@pg.infn.it

EXECUTING A WORKFLOW



No one solution fits all..

- Collective Knowledge (CK), an open-source framework and repository to enable sustainable, collaborative and reproducible research and development to share artifacts as reusable and customizable components with a unified Python JSON API; assemble portable and customizable experimental workflows
- Toil, scalable, efficient, cross-platform and easy-to-use workflow engine in pure Python
- Dagman, meta-scheduler for HTCondor
- Makeflow, Makefile like workflow system, HTCondor supported
- Airflow, Web-based workflow system
- Nextflow, Data-driven computational pipelines, enables scalable and reproducible scientific workflows using software containers. It allows the adaptation of pipelines written in the most common scripting languages. HTCondor supported.
- Snakemake
- …

spiga@pg.infn.it

In house solutions: what happened @HEP

20 years ago all the experiments at LHC started designing the computing models and preparing the Computing Technical Design Reports. Since then a lot of development has been carried on in order to fit specific need.

As a results several services for managing workflows have been also developed:

- Ganga
- CRAB
- WMagent
- Panda
- Dirac

HEP: the example of CMS

A world wide global pool of computing resources

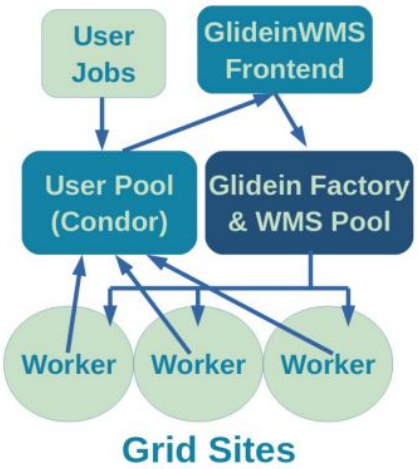
- Dynamic that grows and shrinks based on user's request.

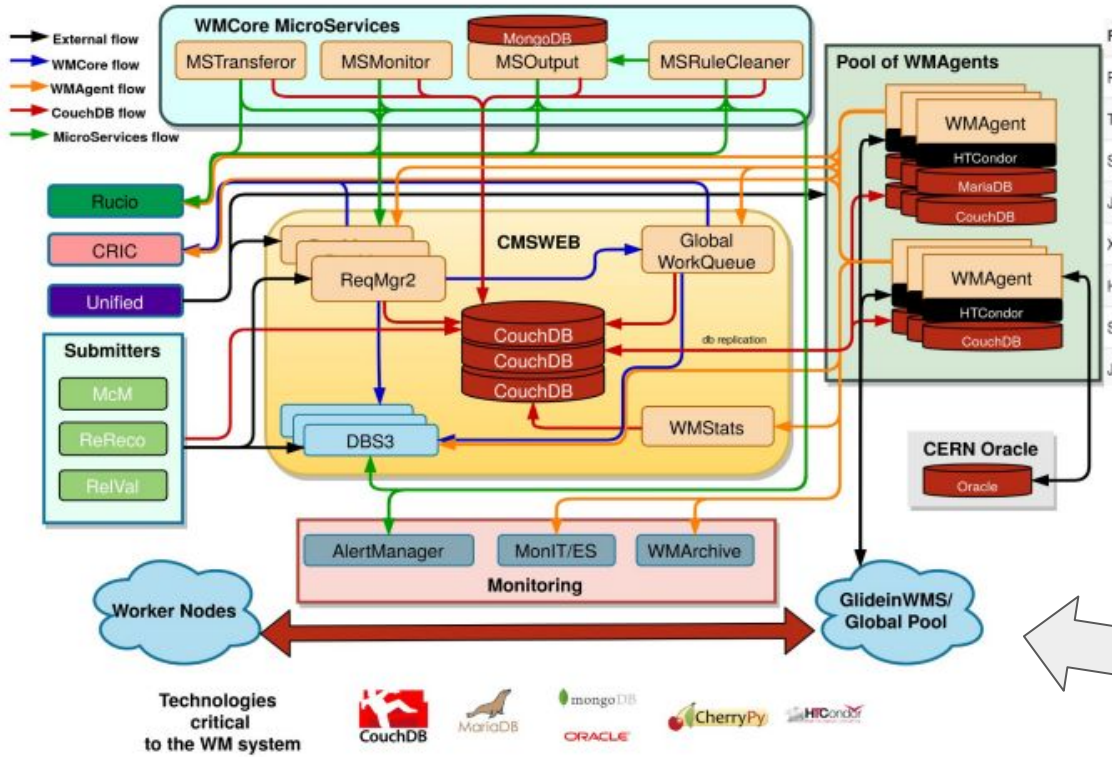
The glideinWMS frontend is **the brain**

- It looks at the user jobs in the condor schedulers
- Matches them over computing elements, aka entries
- Determines number of pilot jobs to submit

The glideinWMS factory is **the arm**

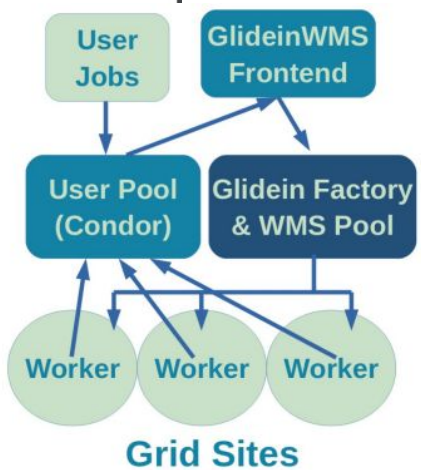
- Submit the glidein_startup.sh script to entries (CEs) based on frontend requests





File Type	Files	Lines of Code	Total lines
Python	1944	221302	283393
Text	15	0	15825
Shell	23	1380	1900
Json	95	766570	766648
Xml	25	13637	13889
Html	56	3894	
Stylesheets	34	3409	
JavaScript	410	35929	

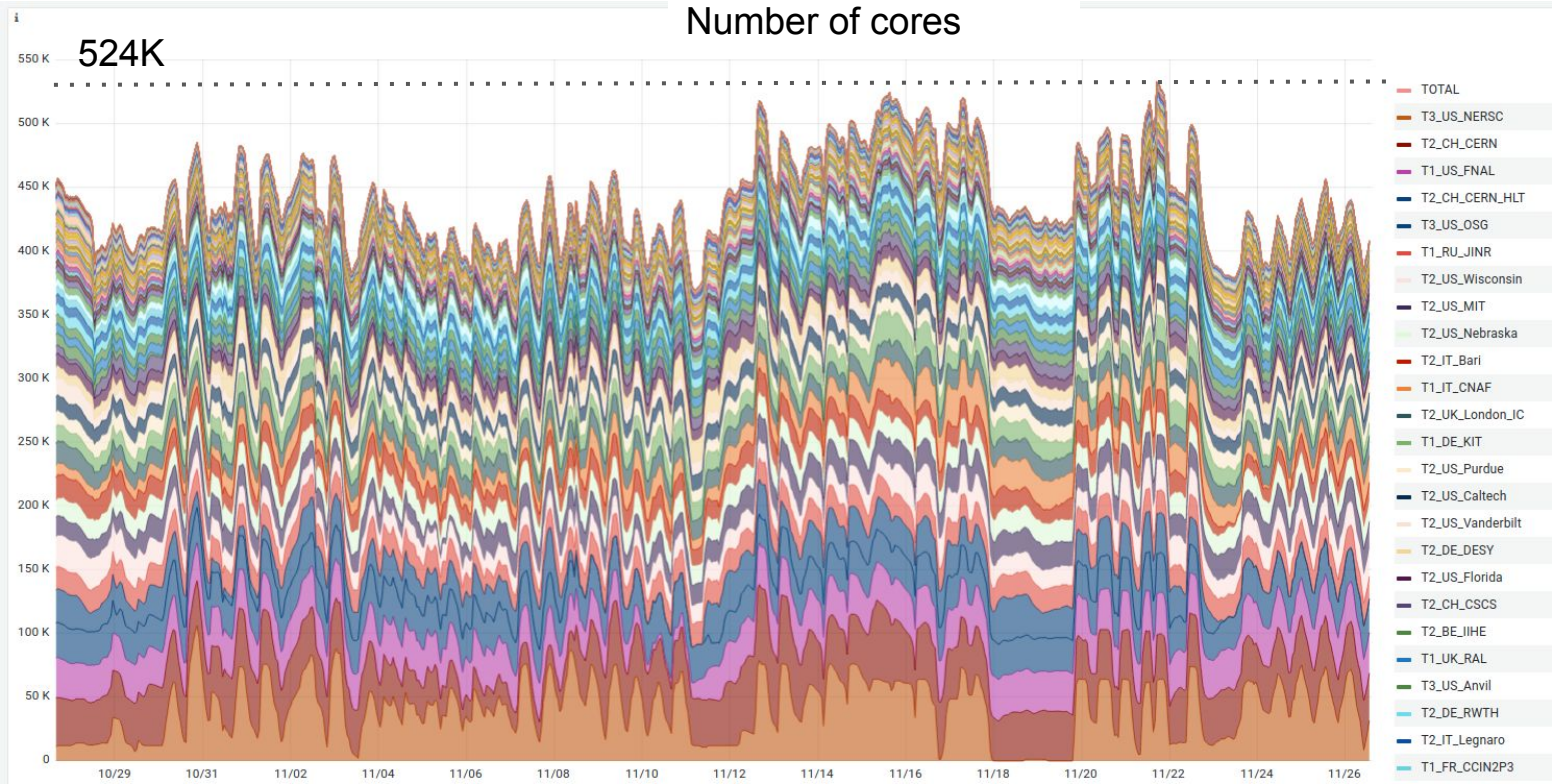
Resources Management



Job Management

How behave today… a record few days ago

Record peak CPU utilization: 524K CPU cores



Recap

Working with batch system requires thinking of work in a different way

- splitting jobs; optimizing for throughput rather than single-job performance
- Managing input data and output data
- Managing software
- Bookkeep and track job status

Question is: Does distributed resources means necessarily using batch systems?

- The short answer is no
- You can **transparently access a distributed computing system** and analyse a huge dataset interactively (quasi-interactively)

spiga@pg.infn.it

Why transparently

Simply because many of the technicalities that we discussed so far are hidden

- Splitting
- Parallelization
- Throughput
- Matchmaking features
- (partially even the data access)

From user perspectives (examples):

- Just run a jupyter cell
- Or use related python API
- …

This will be discussed in the next session

spiga@pg.infn.it