# Workload Management

## Spotlight on Current Advancements and Future Plans

The 10th Dirac User Workshop

June 20th 2024

Federico Stagni & Alexandre Boyer

federico.stagni@cern.ch & alexandre.boyer@cern.ch

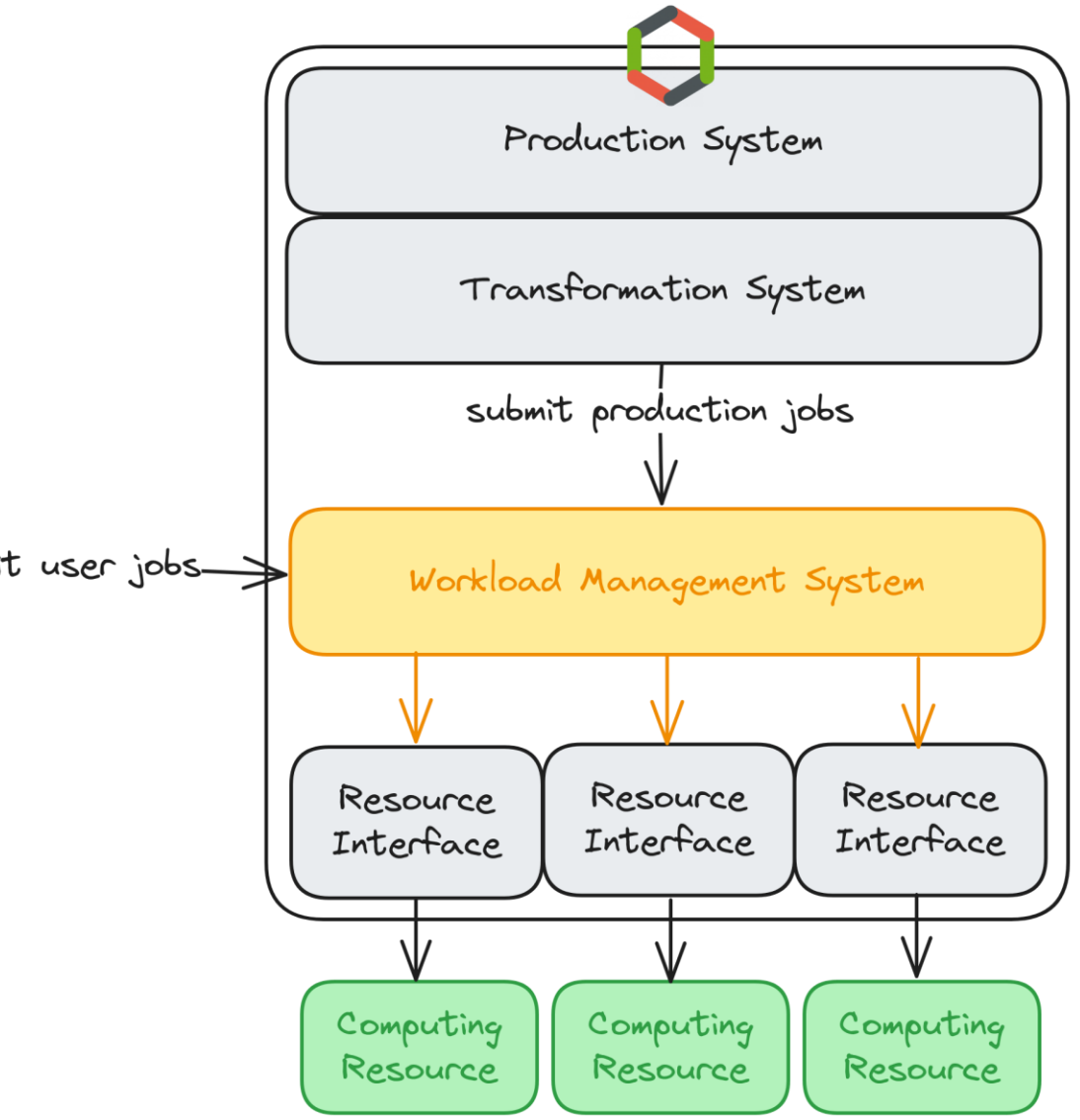
**European Organization for Nuclear Research**

Meyrin, Switzerland

# Introduction

## Workload Management System

- **Central Role in Dirac:** as the backbone of Dirac, it ensures efficient and effective job scheduling and execution.
- **Comprehensive Resource Federation:** aggregates a vast network of heterogeneous computing resources.
- **Empowering Communities:** facilitates the execution of complex workflows at a large scale, essential for scientific and research communities.



# Defining Jobs

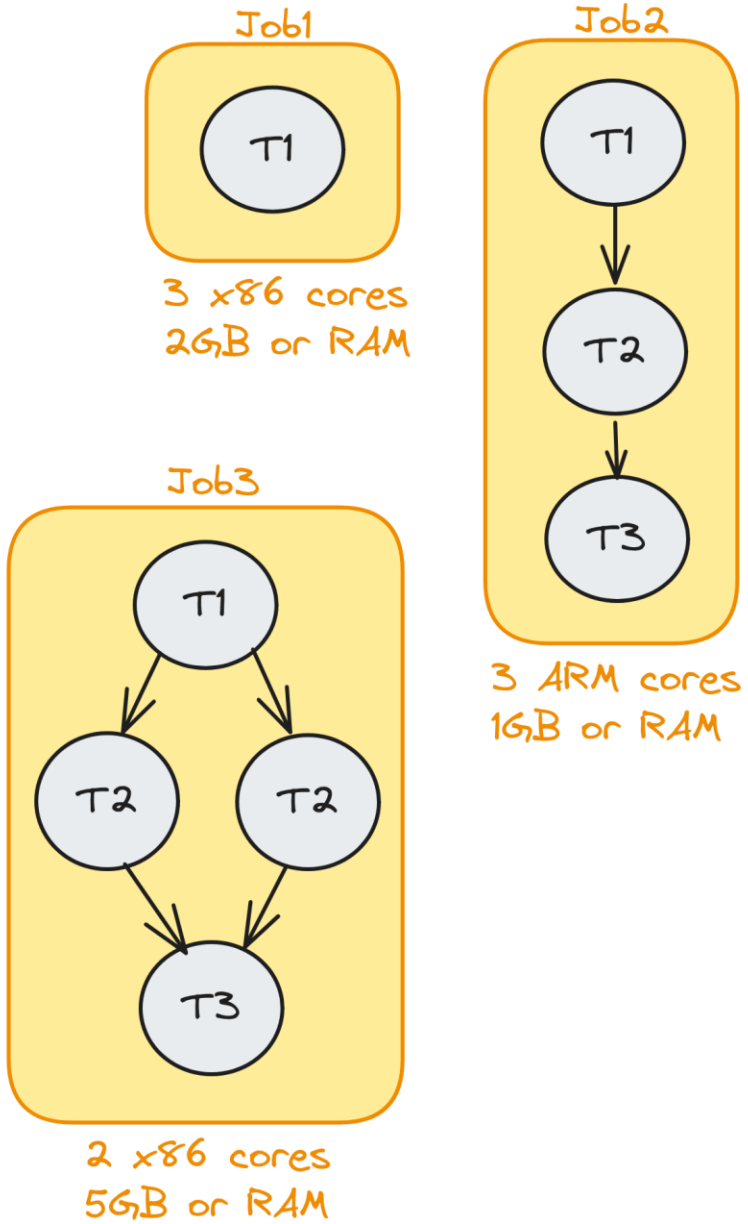
# 1.1.1 Dirac jobs

## Dirac Job

- Type of container to acquire resources on a computing system.
- Combination of a task (or workflow) along with its metadata (hardware, software)

## A word about multi-node tasks

While multi-core jobs running on a single node are supported, multi-node jobs are currently out of scope (no use case).



# 1.1.2 Content of a job

## Task definition

- **Python and bash script**
- **Dirac workflow:** Linear workflow described as an XML file, specific to Dirac. Tasks can be described in Python.
- New in v8.0 **CWL (common workflow language)**: open standard for describing how to run command line tools and connect them to create workflows. (#7542)

## Metadata

- **JDL file (Job Description Language):** provide metadata for the Dirac WMS.
- Along with task requirements, you can also define Dirac-specific parameters such as the site(s) you want to target.
- JDL allows specifying parametric jobs (same task processing different inputs).

# 1.1.3 A word about CWL

## Powerful description language

- **Benefits of CWL:**
  Interoperable, portable, reusable, scalable, transparent, community support
- **Limitations of CWL:** can be complex

```yaml
cwlVersion: v1.2

class: CommandLineTool
baseCommand: echo

inputs:
  message:
    type: string
    default: "Hello World"
    inputBinding:
      position: 1
outputs: []
```

## Future directions:

- CWL support is (and will stay) very limited in Dirac.
- But we aim to make **CWL the primary method for job description in DiracX.**
  - Should replace Dirac workflows at some point, as well as a large part of the JDL.
  - Ongoing efforts to use at the production/transformation level.
- Ongoing discussion about the user interface (feel free to participate): #175

# 1.2.1 Submitting jobs: Validation

## Dirac workflow and CWL

- `dirac-jobexec`: execute the Dirac workflow locally.
- `cwltool`: equivalent to `dirac-jobexec` for cwl workflows.

## New in v9.0 JDL validation (#6973)

- Performed server side
- Immediate feedback if any obvious error:
  - `inputData contains too many files (must contain at most 500)`
  - `maxNumberOfProcessors must be greater than minNumberOfProcessors`
  - `sites and bannedSites are mutually exclusive`
  - `Invalid platform`

# 1.2.2 Submitting jobs: Interfaces

## Interfaces

- **CLI:** create your JDL manually and submit it with Dirac commands.
- **Python API:** define the metadata of the task, the API is handling the rest.
- **Web app:** similar to the Python API but easier for muggles.

## Note:

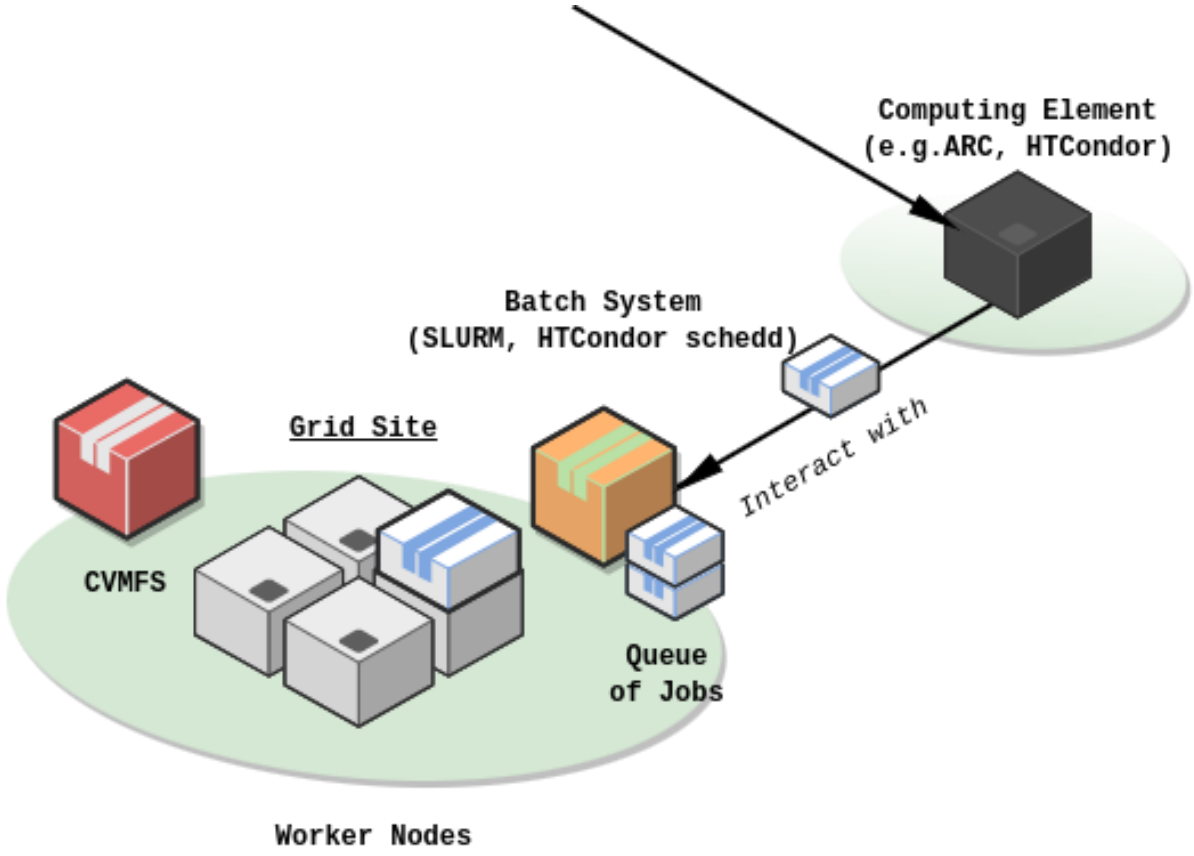
- Submitting a parametric JDLs will result in the generation of multiple jobs.
- CWL can be tested by specifying `cwltool` as the executable and the cwl file as an argument in the JDL.

# Accessing Heterogeneous Computing Resources

# 2.1.1 Traditional Grid Sites

## Description

- Clusters composed of worker nodes and orchestrated by a batch system.
- Mostly composed of x86 CPUs (Intel, AMD).
- External connectivity from the worker nodes.
- CVMFS is installed and mounted on the worker nodes (software dependencies).
- Available through a Computing Element (CE)
  using X509 certificates and/or OIDC tokens.



# 2.1.2 Computing Elements: ARC

## ARC

- **LTS**: v6 (v7 is coming "soon").
- **Job management:** replacing gridftp with AREX to manage jobs. AREX comes with a REST interface.
- **AuthN/Z:** through x509 certificates. Limited support for OIDC tokens (that we have not been able to use so far). They should be better supported from v7.

## Dirac interface

- **ARCCE**: the good old interface, leveraging the python arc client and the gridftp interface to interact with ARC instances. Dropped from v9.0
- **ARC6CE:** intermediate solutions, using the AREX services through the python arc client. Dropped from v9.0
- **AREXCE:** the new interface, leveraging the REST interface to interact with the AREX services.
- **Transition details**: wiki

# 2.1.3 Computing Elements: HTCondor

## HTCondor

- **LTS**: v23
- Change release management, new terminology, better support for containers…
- **Job management:** HTCondor client necessary to interact with HTCondor instances. Python bindings available.
- **AuthN/Z:** dropped support for Globus toolkit in v9.3 and fully embraced OIDC tokens. SSL certificates are still supported though.

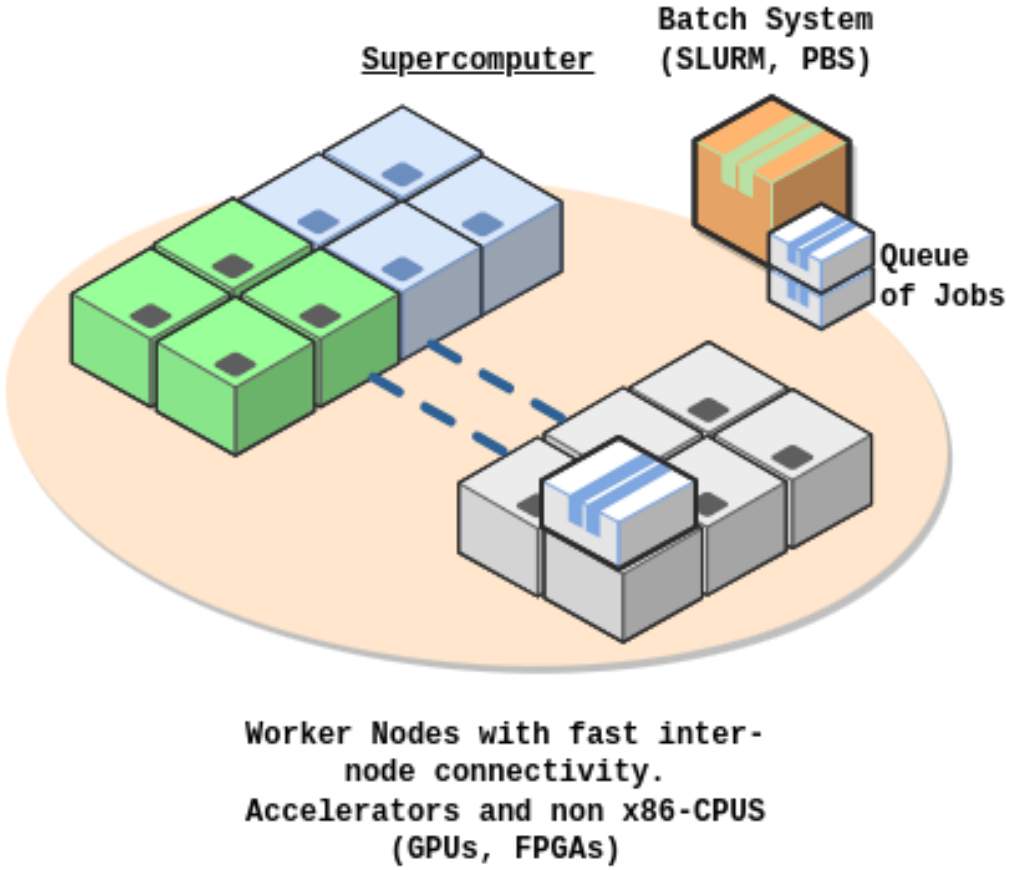
## Dirac interface: HTCondorCE

- Calls the HTCondor CLI to interact with the instances (could be worth trying the Python bindings).
- Support OIDC tokens (#6803) and, temporarily, SSL certificates (#7630). New from v8.0
- A few fixes to get more details about failed/aborted jobs in v8.0 (#7069)

# 2.2.1 Opportunistic resources and HPCs

## Description

- Clusters composed of worker nodes with fast inter-node connectivity.
- Can contain non-x86 CPUs (ARM) and GPUs.
- External connectivity is not guaranteed and access can be protected via a VPN.
- CVMFS is not installed and mounted on the worker nodes.
- Available through SSH.



# 2.2.2 SSH and batch systems:

## Batch Systems

- **Various Batch Systems:** HTCondor, Slurm, PBS/Torque, SGE, LSF.
- Nowadays, 2 of them are predominant: HTCondor in HTC, Slurm in HPC.

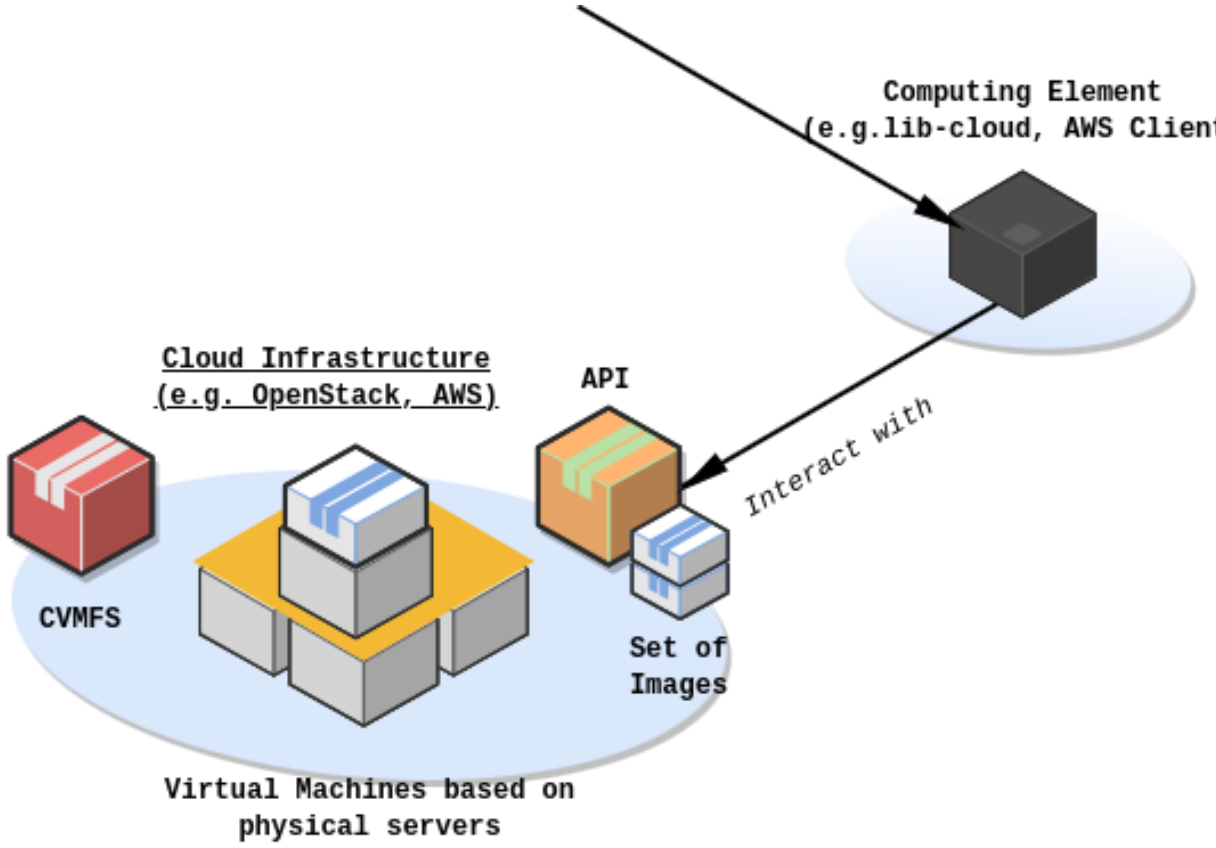
## Dirac interface: SSHCE

- Dirac-specific solution calling SSH commands to interact with the batch system.
- Python library such as Fabric could be investigated.
- Batch Systems have different interfaces: there are Dirac-specific plugins to interact with them properly.
  - Slurm is well supported. Plugins for Condor, PBS, SGE, LSF, OAR too.
  - New in v8.0 A job parameter indicating the batch system used (#7289)

# 2.3.1 Clouds (IaaS)

## Description

- Clusters composed of Virtual Machines spawned by the users according to their needs.
- External connectivity from the VMs.
- CVMFS can be installed and mounted on the VMs (software dependencies).
- Available through the cloud service provider API.



# 2.3.2 Cloud service provider APIs

## APIs & libcloud

- **Various APIs**: AWS, Google cloud, Azure, Openstack, OpenNebula…
- **libcloud:** python library for interacting with many of the popular cloud service providers using a unified API.

## Dirac interface

- **VMDirac**: Dirac-specific and complex solution to interact with a few cloud service providers (#6380). Dropped from v9.0
- **CloudCE:** Replace VMDirac. A much simpler solution based on libcloud.

# 2.4.1 A word about volunteering computing

## BOINC

- There was some efforts to support integration of BOINC within Dirac.
- Trustless environments based on small clusters and desktop computers.
- Interesting for preemptible HTC workloads.

## Dirac interface: BOINCCE

- Allows to interact with BOINC resources via SOAP. No progress since 2013.
- Could potentially be resurrected from the work done with HPC resources, but no use case.
- If not used anymore, it will likely be dropped during the transition to v9.0.

# Supplying Computing Resources with Jobs

# 3.1.1 Pre-processing Dirac jobs

## Default asynchronous operations (Optimizers)

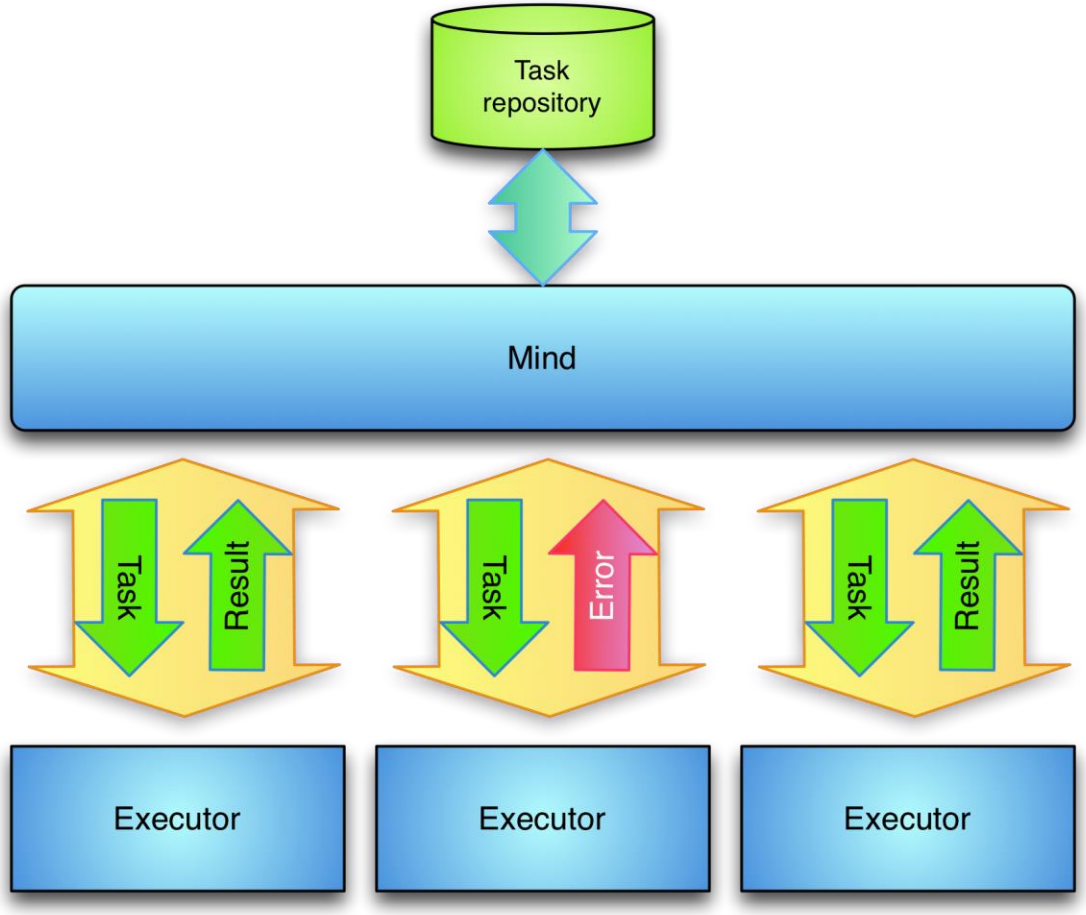
- **JobSanity**: New from v8.0 Simply assign a sandbox to a job (JDL validation is performed synchronously now).
- **InputData:** Query the file catalog for specified input data and adds information for the next operation.
- **JobScheduling:** Make a scheduling decision and gathers similar jobs in task queues, waiting for being matched with a computing resource.

A few old and unused features were deleted from v8.0: VO plugins (#6161), filtering by platforms (#6178).

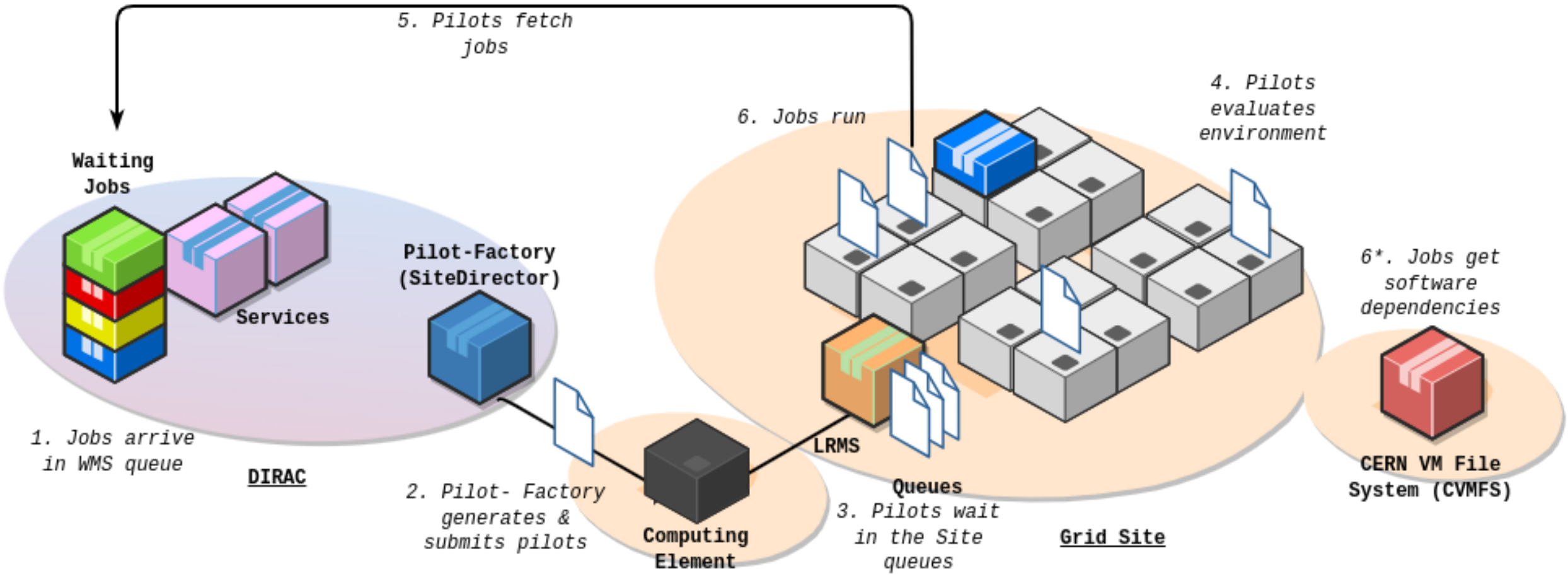
# 3.1.2 Moving away from executors

## Executors

- Dirac-specific task queue solution: composed of a Mind that distribute tasks to Executors.
- There has been an attempt to replace the framework with Celery and message queues from v8.0 (#7022), but it was not straightforward.
- Solutions will be reassess within DiracX.



# 3.2.1 Getting allocations using the pull model

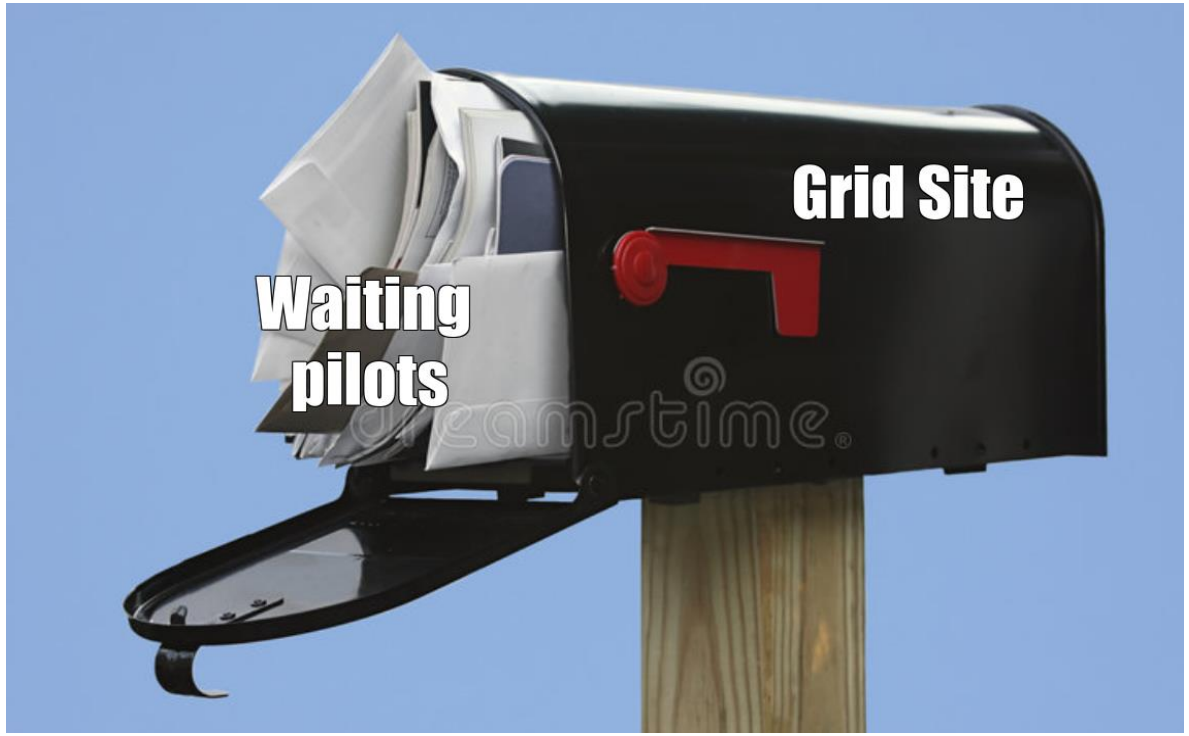


# 3.2.2 Generating Pilots

## Site Director

- **Submit pilots:** check available jobs for a given resource and slots available and submit pilots accordingly (using Dirac Interfaces).
  - New from v9.0, SiteDirector does not take Task queues into account anymore, submissions are parallelized per CE (#7110).
- **Monitor pilots:** check the status of the submitted pilots (parallelized per CE).

A few old and unused options were deleted from v9.0 (#7110) and 1 Site Director acts for 1 specific VO (#7263).



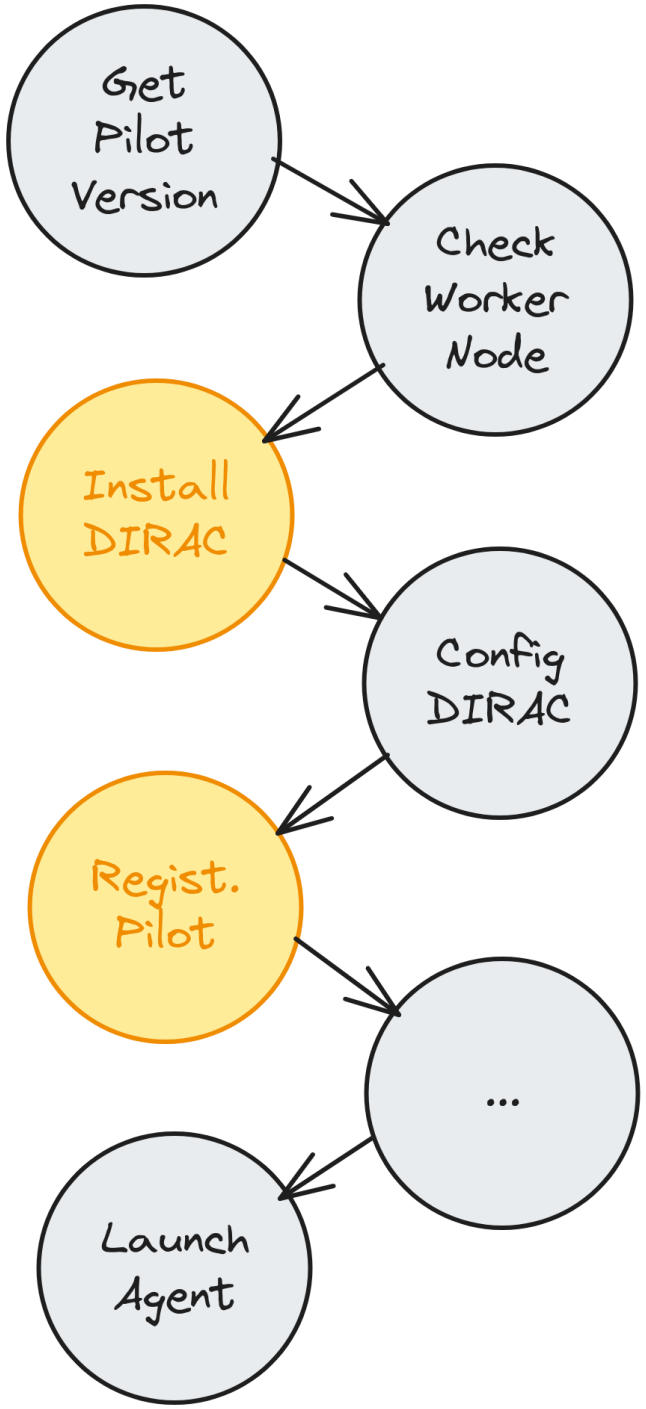
# 3.2.3 Pilot Structure

## Pilot architecture

- **Various commands**: At a minimum install DIRAC, configures it, and run a JobAgent.

## Pilot upgrades

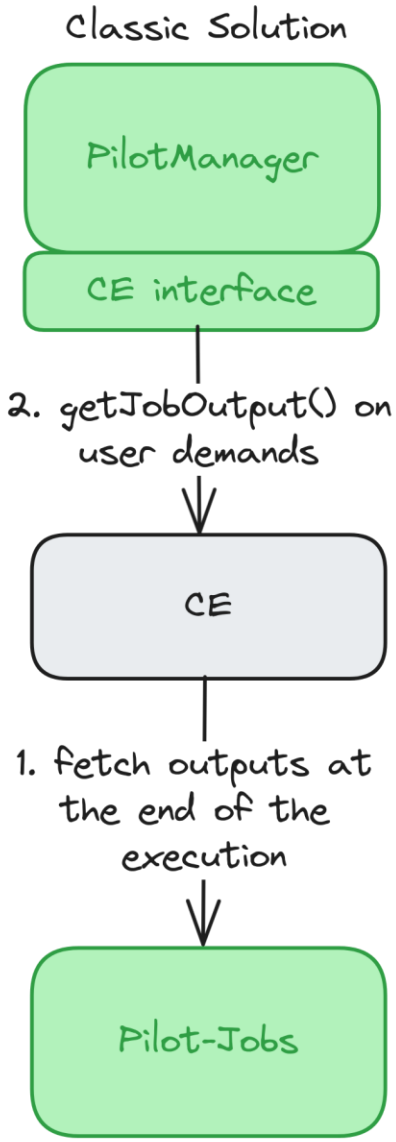
- **devel branch:** fixes and new features should target the development branch (tested in certification).
- **master branch:** admins are taking care of merging devel into master when required.

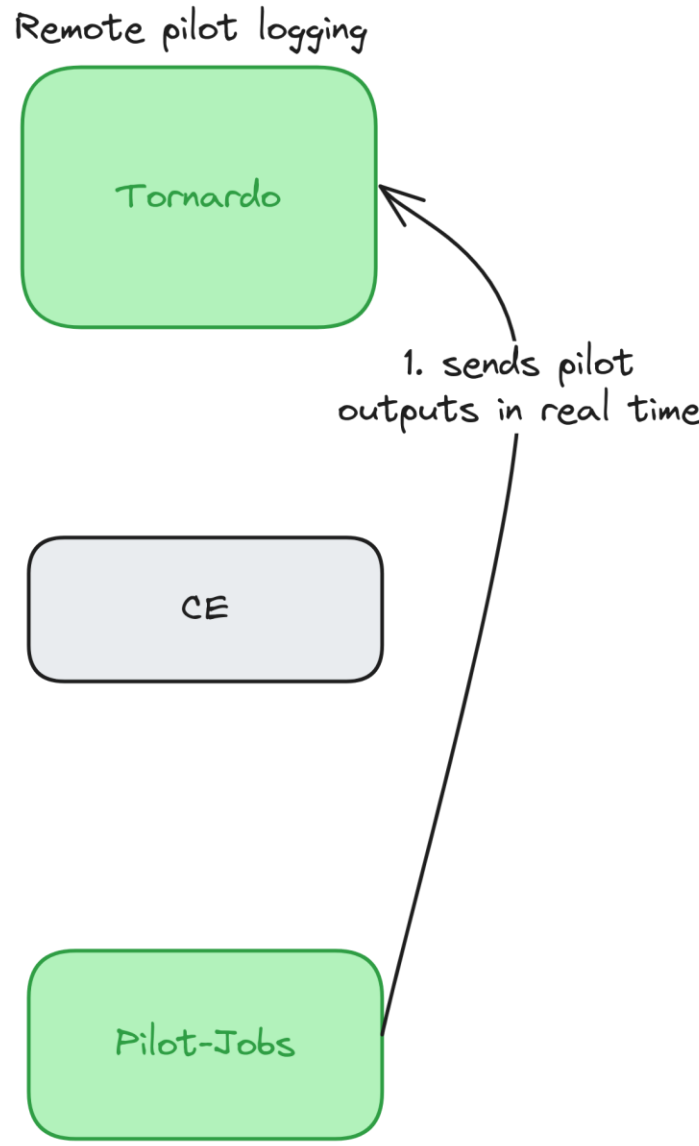


# 3.2.4 Pilot: various new features

## New features

- **"Register Pilot" command**: Pilots started from the vacuum can register themselves in the DB and be declared as "Running" (#6914).
- **Remote Pilot logging:**
  Pilots sends their logs to Tornado by themselves instead of being fetched by the CE interfaces (#158). Configuration includes (#227):
  - Log buffer size: avoid too many interactions with Tornado
  - CE white list: enable remote pilot logging only for a specific set of CEs.



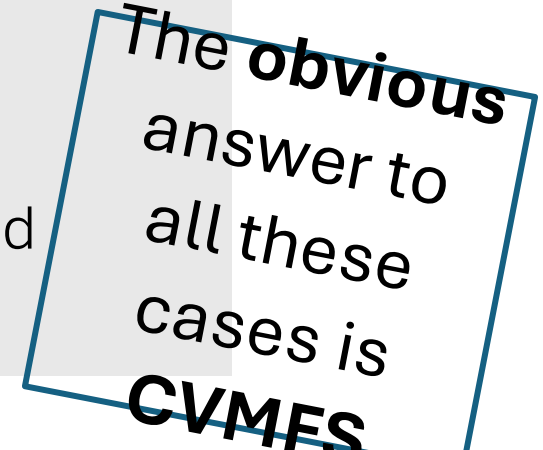
# 3.2.5 Pilot: install Dirac from CVMFS (#205)

## Dirac Client (Pilot) requirements

- Certificates for each users (for creating proxies)
- CAs and CRLs, because…certificates: Bundle Delivery service, or better found locally.
- Certificate proxies, so VOMS: Dirac needs VOMS config files (pointers to VOMS servers).
- Python 2.7+
- Internet access
- Container images: at a minimum for isolation through Apptainer.
- A pointer to a CS.
- DIRACOS and the DIRAC code: can be downloaded on-the-fly, or better found locally.

## Can Dirac work without CVMFS?

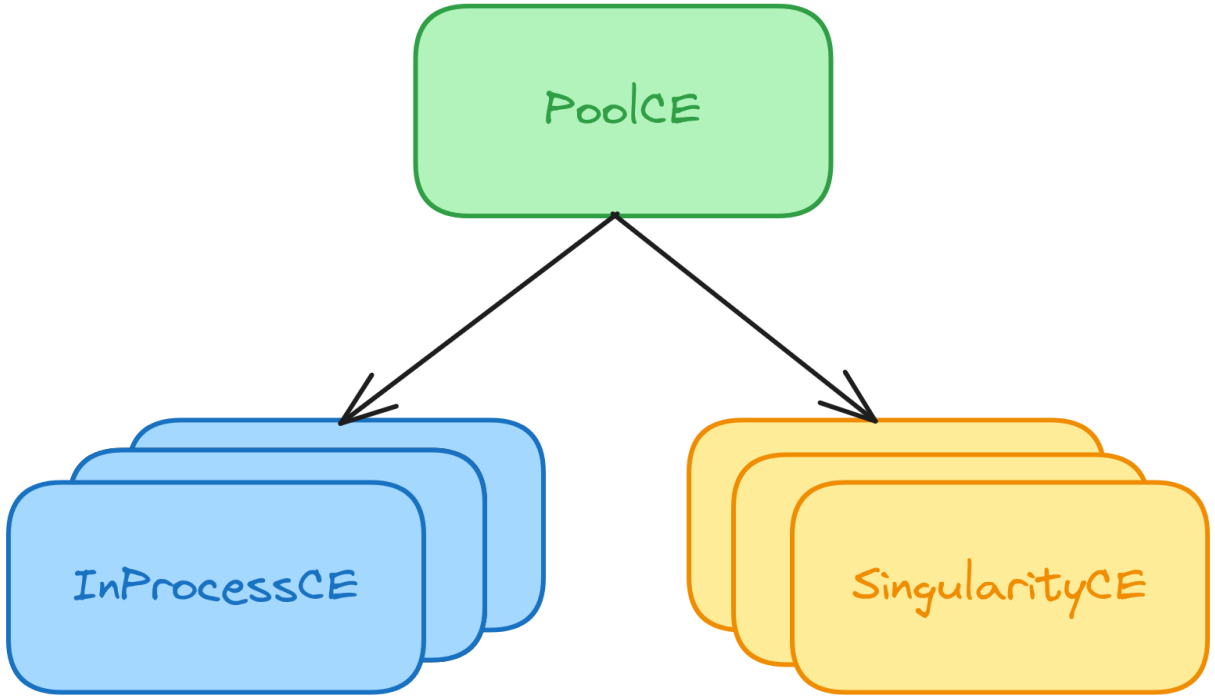
- Yes, and you may well have this case in HPCs
- CVMFS_locations may point to non-CVMFS locations…



# 3.2.6 JobAgent and inner CEs

## JobAgent

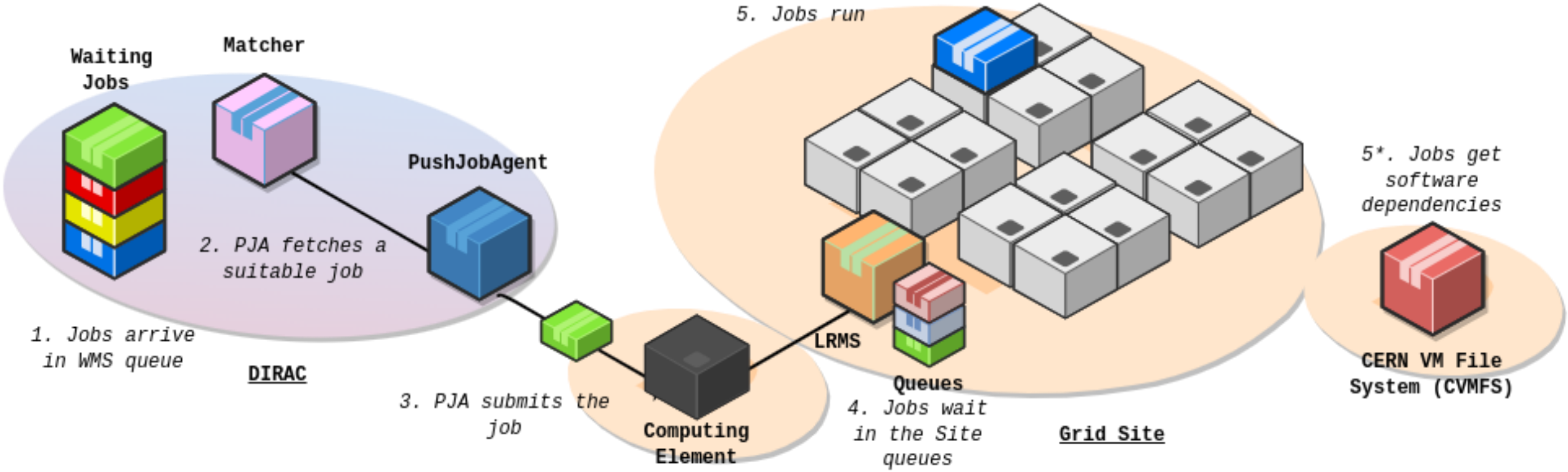
- **Match jobs and submit them to inner CEs:**
  - **InProcessCE:** manages 1 job at a time on bare-metal
  - **SingularityCE:** manages 1 job at a time in a container (isolated)
  - **PoolCE:** rely on multi-process and previous CEs to manage multiple jobs in parallel.




## A word about multi-core allocations

**SingularityCE** should be preferred over **InProcessCE** to isolate jobs from each other.

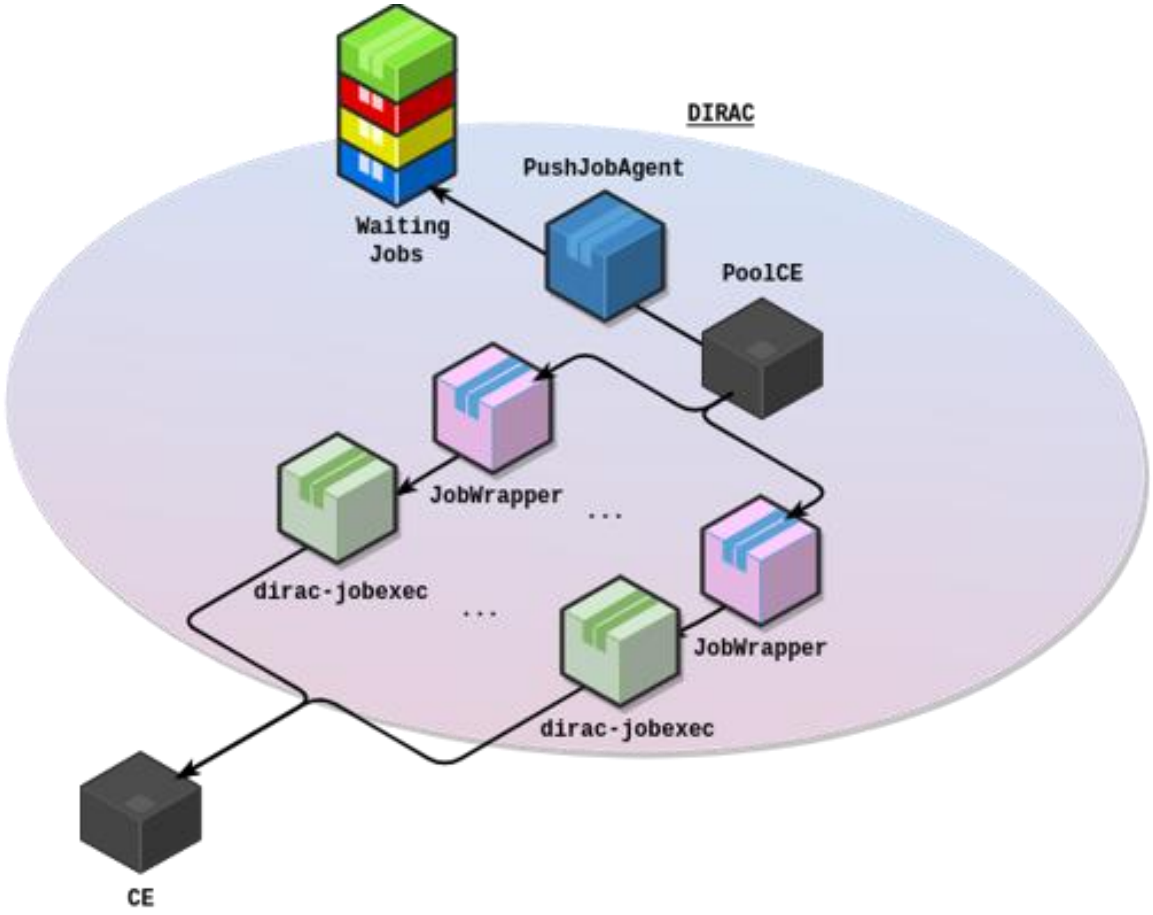
# 3.3.1 Getting allocations using the push model



# 3.3.2 Limitations of the PushJobAgent

## Limitations

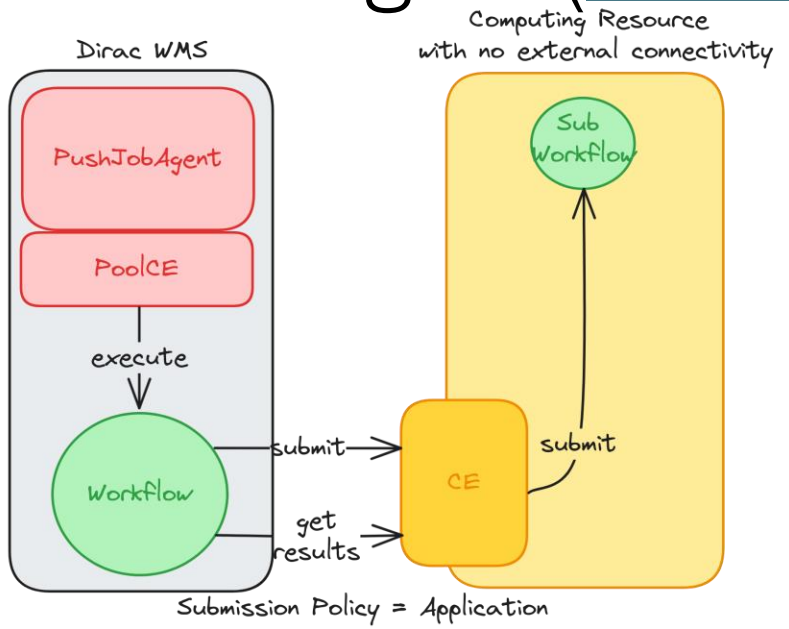
- **Works only with dirac workflows.**
- **Not robust:** any temporary issue with the agent or CE results in the loss of the running jobs.
- **Not scalable: consumes too much RAM:** 200-250 jobs in parallel consumes around 50GB of RAM.
- **Current workflow:**
  - PJA fetches jobs, creates JobWrapper processes for each of them and executes them in parallel (77MB/proc).
  - Each JobWrapper generates a dirac-jobexec process (90MB/proc), which executes the "DIRAC workflow".
  - These processes are remaining in memory as long as jobs are running, even if they just wait for results from remote computing resources.

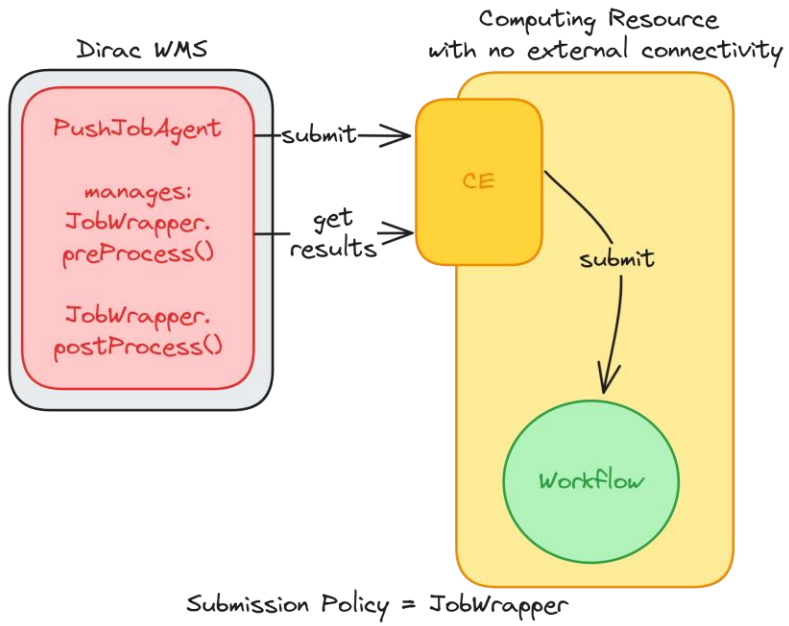


# 3.3.3 Towards a more robust PushJobAgent(#7459)

## Changes

- **Pre/Post process methods in JobWrapper:** Operations around tasks/workflows execution should be transferred from the workflows to the preProcess() and postProcess() method (will be customisable).
- **JobWrapperOfflineTemplate:** JobWrapperTemplate solely executing the task/workflow (no pre/post processing).
- **PushJobAgent submission policies:**
  - **Application (temporary):** the original and limited solution: only a small part of the workflow is submitted to a remote computing resources
  - **JobWrapper:** pre/postProcess() operations executed by the agent, the whole task/workflow is sent to the remote computing resource.



# Adapting for Tomorrow: Future Plans and Strategies

# 4.1 CWL in the Production/Transformation Systems

## Unified language

- **Production:** A CWL with multiple steps. Each step could be a workflow or a command line tool.
- **Transformation:** correspond to a step of the main CWL.
- **Job:** similar to a transformation with additional details.

A few examples of workflows are available in dirac-cwl-proto.

## Goal

- Local Workflow Testing: Initially, the user would test the CWL workflow locally using cwltool, validating the workflow's structure and ensuring that it executes correctly with the provided inputs.
- Submission as a Dirac Job or a Dirac Production: Just by providing the CWL and a few additional parameters (outside the CWL). It should work transparently.

# 4.2 A word about Dirac Benchmark

## Challenges

- **DB12 code depends on Python updates:** the language becomes more performant, which impacts the benchmark scores.
- **HS06:** DB12 was originally mapped to HS06, which is not used anymore by WLCG.
- **ARM and GPUs are more prevalent:** we are not sure whether DB12 performs correctly in these contexts.

## Considered solution (#12)

- **Freeze DB12 code and dependencies:** create an executable that would last until Python 3.12 is deprecated.
- **HEPScore:** Align DB12 on HEPScore, which should be representative of WLCG workloads.
- **Assess differences between the freezed DB12 package and HEPScore:** each time the package needs to be updated (factors to correct the scores accordingly).

# Thank you for your attention

## Questions? Comments?