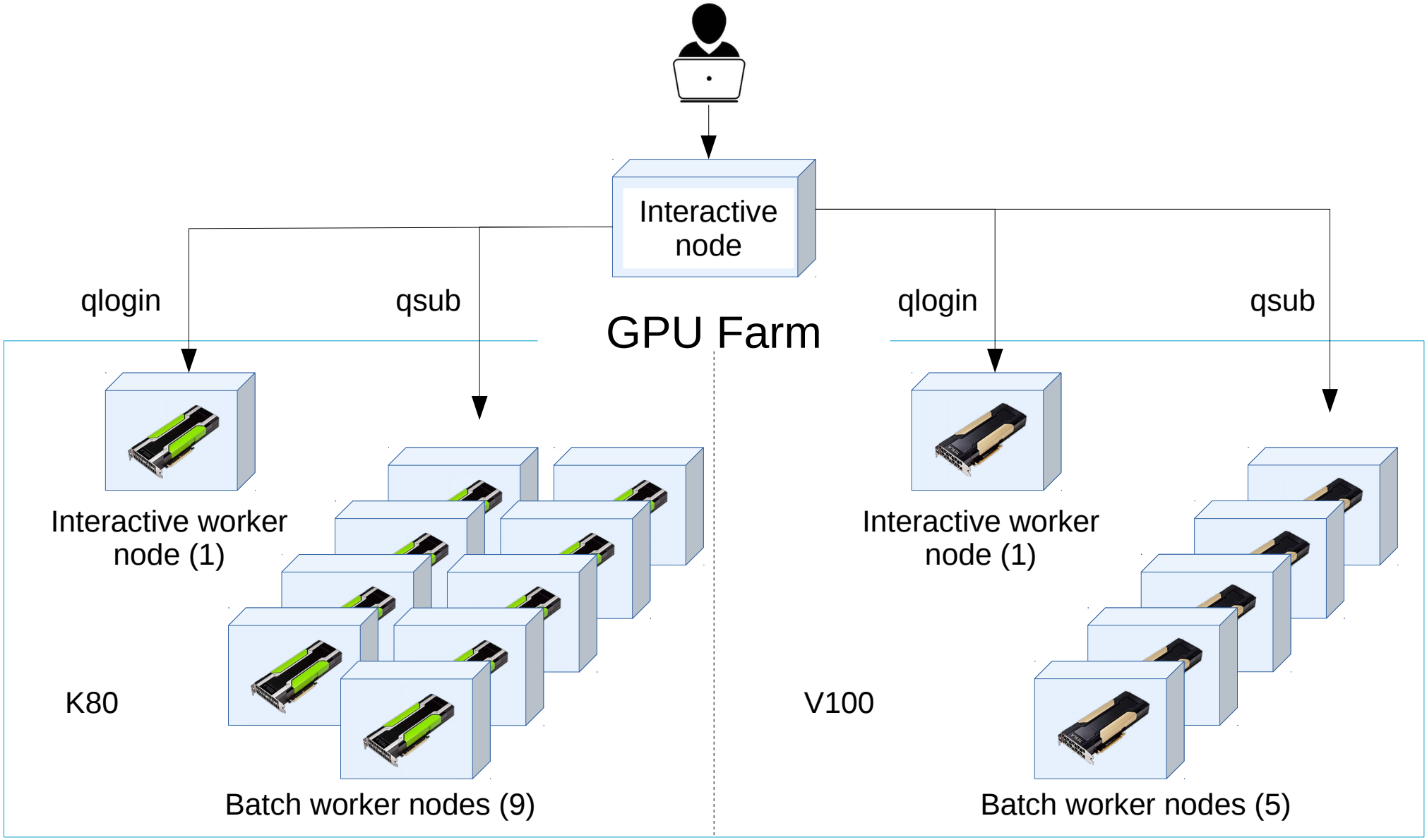
Job Submission on CC-IN2P3 GPU Farm

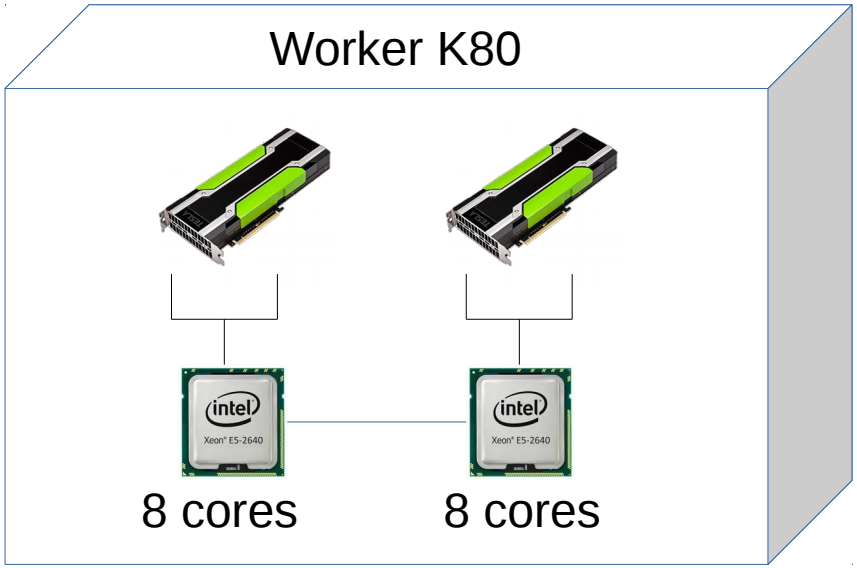
April 2019

Current Farm Architecture

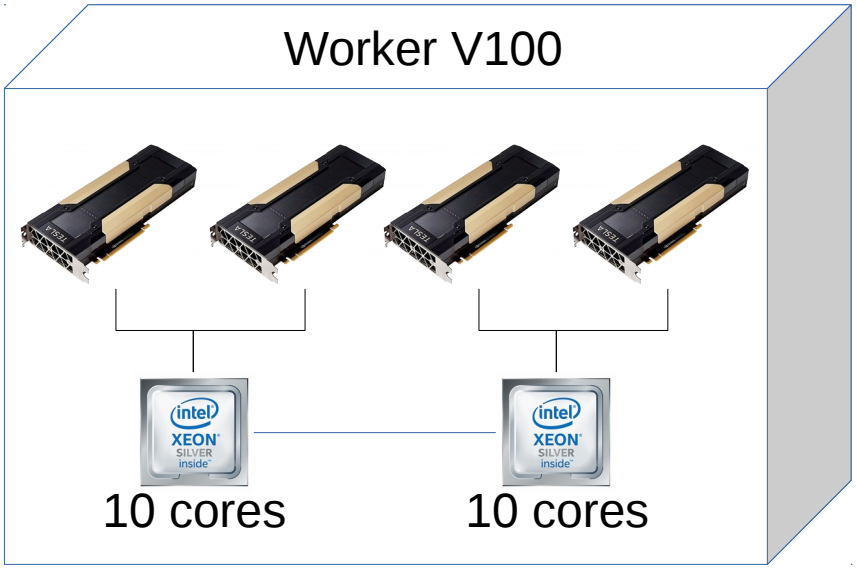
- Access from an interactive host
- Univa Grid Engine as batch system (https://doc.cc.in2p3.fr/jobs_gpu)



Workers Architecture



2 cpus (2x8 cores) + 4 (2x2) gpus



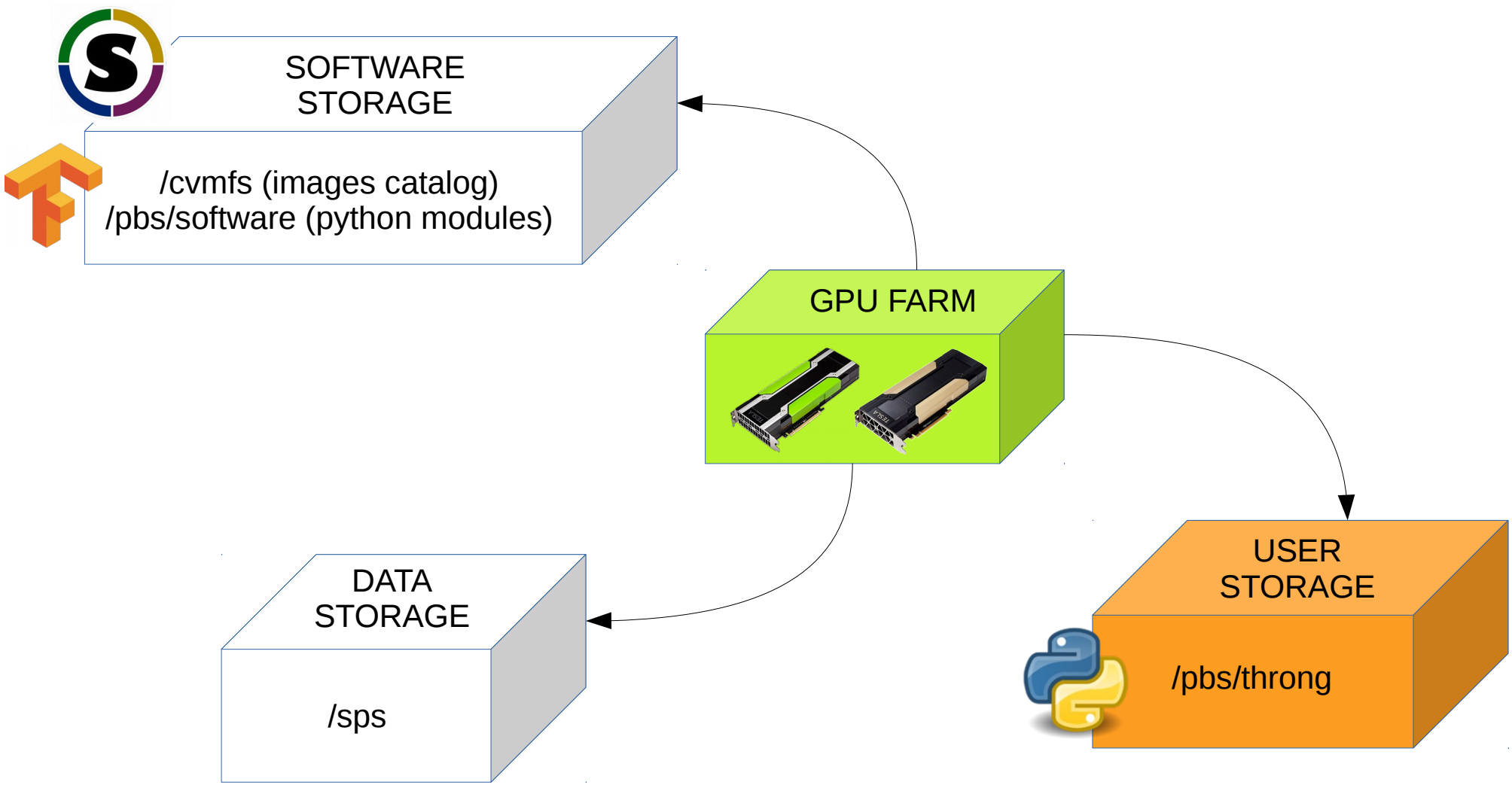
2 cpus (2x10 cores) + 4 gpus

 = 2 gpus

 = 1 gpu

Farm neighbourhood

- Worker nodes can access different types of storages



Interactive mode

- Interactive Worker and Batch Worker nodes are the same in terms of architectures (same cpus, gpus, memory)
- Batch scheduler provides access to Interactive Worker nodes in shell mode

Interactive Worker access (qlogin)

Specifying the use of sps resources

Nb of gpus <1 - 4>

Specifying K80 or V100 farm

```
qlogin -l GPU=1,sps=1,GPUtype=K80|V100 -q mc_gpu_interactive -pe multicores_gpu 4
```

Custom parameters

Queue

Nb of cpu cores

- Batch scheduler provides access on Batch Worker nodes to execute a program

Specifying the use of sps resources

Nb of gpus <1 - 4>

Specifying K80 or V100 farm

qsub -l GPU=1,sps=1,GPUtype= K80 V100 [options] <file_to_execute>

Custom parameters

Batch mode - Options

- All information can be found here:

  General Batch System: https://doc.cc.in2p3.fr/en:utiliser_le_systeme_batch_ge_depuis_le_centre_de_calcul

  GPU Jobs: https://doc.cc.in2p3.fr/en:jobs_gpu

  Available Queues: http://cctools.in2p3.fr/mrtguser/info_sge_queue.php

- !!! access to GPU queues requires resources request from user groups !!!

Queue (-q)

- Multicores (1 node)

  mc_gpu_medium (~5h)
  mc_gpu_long (~48h)
  mc_gpu_longlasting (~202h)

- Parallel (multinodes) K80 only!!!

  pa_gpu_long (~48h)

Environment (-pe)

- Multicores (1 node)

  multicores_gpu 4

- Parallel (multinodes) K80 only!!!

  openmpigpu_2 x *(with x = 2 * nb of nodes)*

  openmpigpu_4 x *(with x = 4 * nb of nodes)*

Misc.

- Output file path: -o
- Error file path: -e
- Passing environment vars: -V

Installed libraries

- Updates (n, n-1)

  GPU Jobs: https://doc.cc.in2p3.fr/en:jobs_gpu

Custom environment

- Execute your job on a custom environment via **Singularity**

Why Singularity?

- Farm default environment is updated about 2 times a year, so the most recent installed version can already be obsolete (in this mad world of AI ^^)
- We do not keep more than 2 CUDA versions installed in the farm: a current one and the one before that one
- How to keep reproducibility (as long as possible)

Example

- Let's say you need to execute a code that requires Tensorflow 1.13.0 and the latest installed version of CUDA on the farm is 9.2, you need CUDA 10.0

Version	Python version	Compiler	Build tools	cuDNN	CUDA
tensorflow_gpu-1.13.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.19.2	7.4	10.0
tensorflow_gpu-1.12.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.15.0	7	9
tensorflow_gpu-1.11.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.15.0	7	9

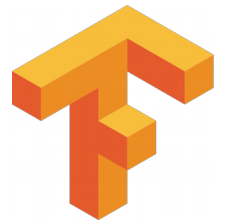
- Singularity will give you the opportunity to execute an image with the right pieces of software installed (i.e CUDA 10.0 library in this case)
- This software flexibility is of course possible as soon as it is still compatible with workers hardware
- One can also create and use its own images which brings maximum flexibility to the farm (see you @ CC Singularity Training Course)

- CC-IN2P3 provides an Image Catalog and Compiled Modules

Where to find what

Images Catalog & corresponding Compiled Modules:
https://gitlab.in2p3.fr/ccin2p3-support/c3/hpc/gpu

Singularity Images:
/cvmfs/singularity.in2p3.fr/images/HPC/GPU



Compiled From Source Modules: (compute speed gain ~20%)
(Python 2.7 – 3.6, for K80 and V100)
/pbs/software/centos-7-x86_64/python/modules/tensorflow

More to come… (pytorch...)

command (from cca)

```
qsub -l sps=1,GPU=<nb_gpus>,GPUtype=<K80-V100> -q <queue> -pe multicores_gpu 4
-o <output_path> -e <error_path> -V <path_to>/batch_launcher.sh
```

/pbs/throng

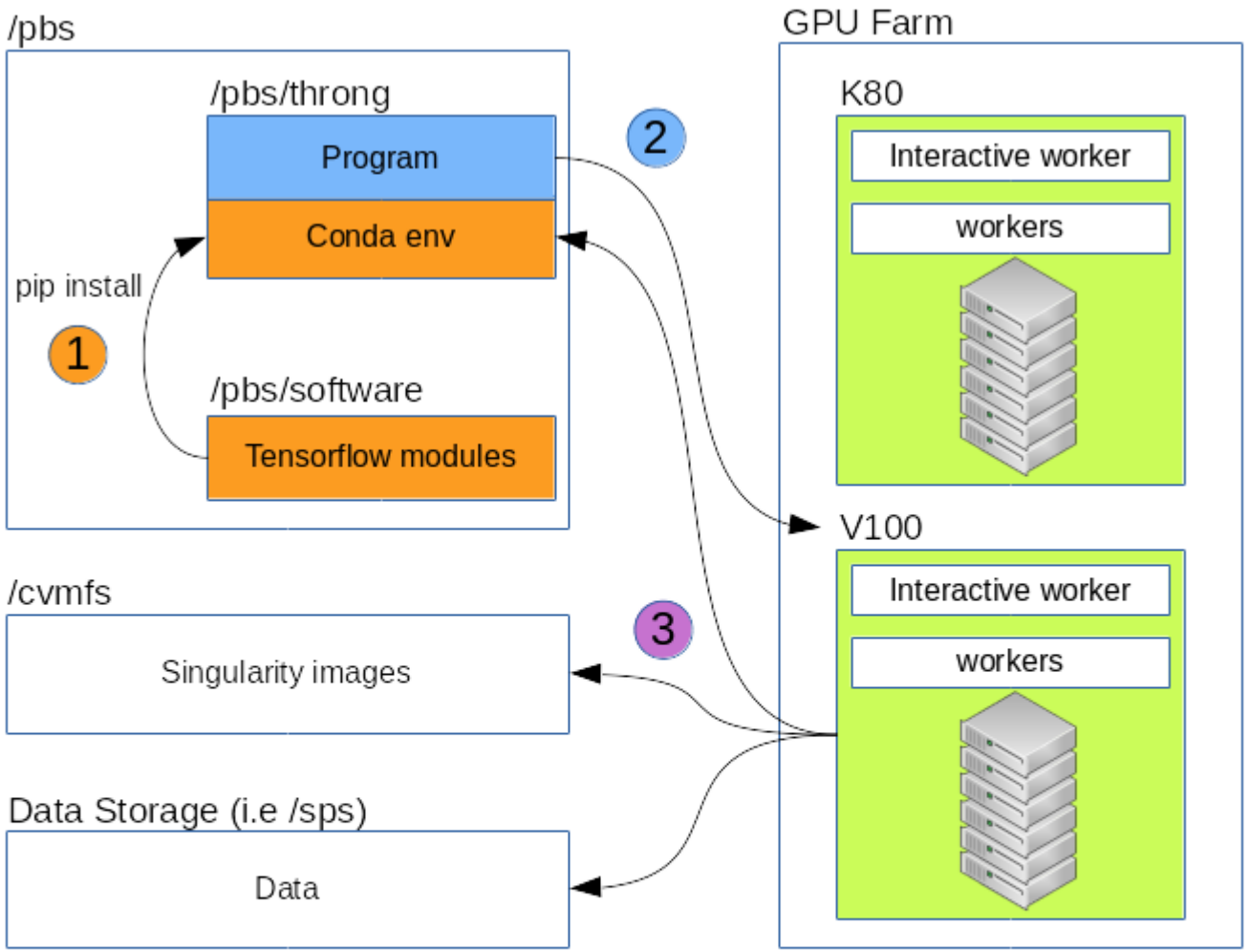
batch_launcher.sh

```
#!/bin/bash
# executed on the worker
/bin/singularity exec --nv --bind /sps:/sps --bind /pbs:/pbs <image_path> <path_to>/start.sh
```

start.sh

```
#!/bin/bash
# executed on the worker, inside the singularity image
source <path_to_python_env> activate <env>
python <path_to>/program.py
```

Workflow



1. Install the tensorflow module you need depending on your python environment (python 2.7 or python 3.6) and the GPU type you want (K80 or V100)
2. Submit your code to the GPU farm, specifying which image you want to run it, and your python environment
3. The GPU farm computes your code through the specified environment

Questions ?

bertrand.rigaud@cc.in2p3.fr

Thanks for your attention.