JOBFLOW-REMOTE

GUIDO PETRETTO

AUTOMATED AB INITIO WORKFLOWS WITH JOBFLOW AND ATOMATE2 - CECAM SCHOOL

LAUSANNE, MARCH 17, 2025 – MARCH 20, 2025

PLAN OF THE TALK

- Overview
- Job execution process
- Interacting with jobflow-remote
- Dealing with failures
- Configure jobflow-remote
- Fine tuning job execution

OVERVIEW

JOBFLOW VS JOBFLOW-REMOTE

Jobflow

Workflow definition

- Job and Flow objects
- Maker
- Outputs → JobStore
- Connections
- Composition

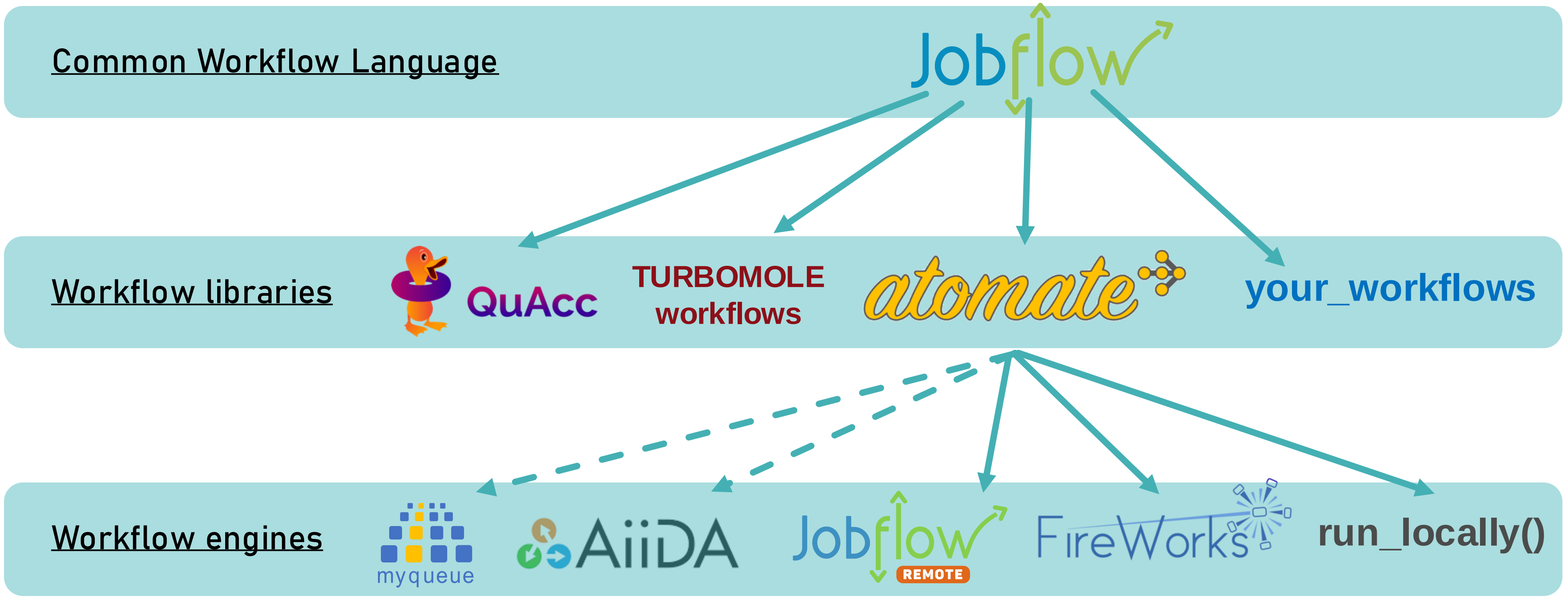
Jobflow REMOTE

Workflow execution

- Jobs and Flows in a DB
- Jobs and Flows state evolution
- Workers
- Submitting jobs

A WORKFLOW ENGINE

- Referred in previous presentations as a workflow engine
- Alternative to the `run_locally()` of the previous tutorials



WHY JOBFLOW-REMOTE?

Why a new manager for jobflow?

- Tailored to jobflow
  - Full features support
  - Better integration
  - JSON serialization
- Overlapping functionalities between Fireworks and Jobflow
  - workflow definition
- Request from a customer: umicore
  - Internal DB cannot be accessed from the HPC centre
  - Only outbound connections

JOBFLOW-REMOTE PACKAGE

- Github repository: https://github.com/Matgenix/jobflow-remote
- Documentation: https://matgenix.github.io/jobflow-remote
- Forum: https://matsci.org/jobflow
- Open source
- License: modified BSD (3-clause BSD)

MAIN FEATURES

- Manage the state of Jobs and Flows
- Job execution does not need access to the DB
- Daemon process orchestrating Jobs execution
  - Handles multiple “workers” (supercomputer, local execution, supercomputer frontend, …)
- Retries, restarts (with fail-safe mechanisms)
- Extensive command line interface (CLI)
- programmatic API
- Optional multiple projects

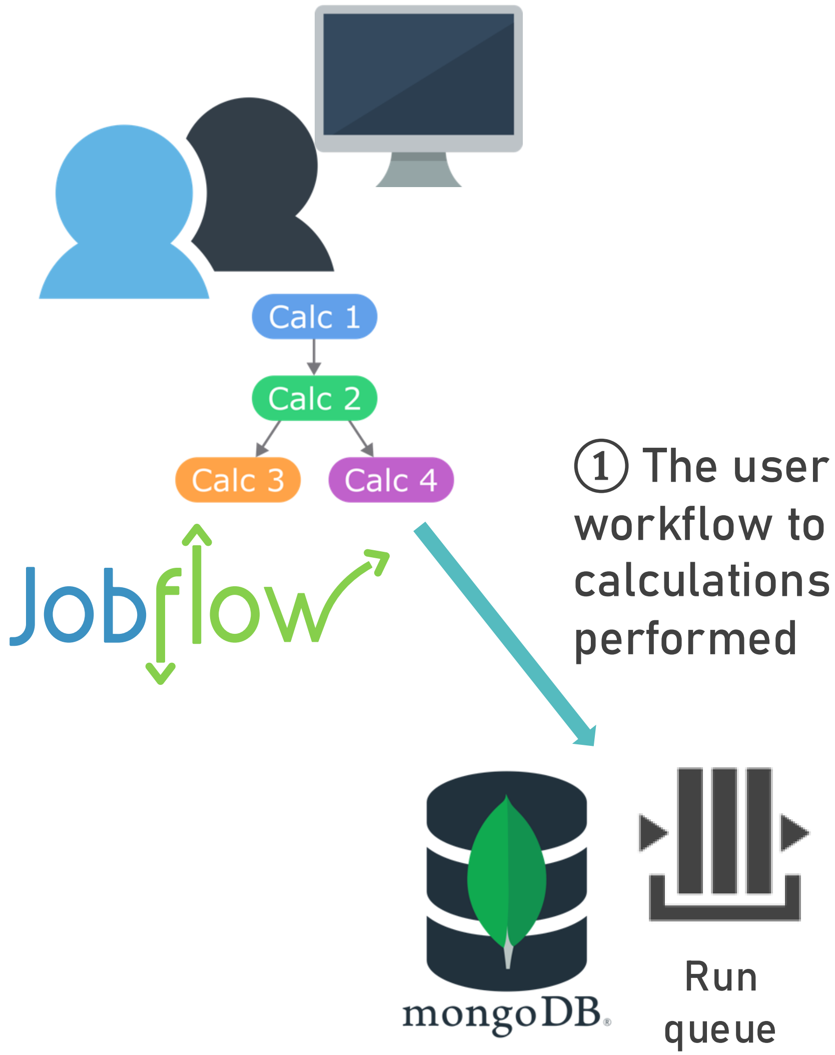
Experimental

- Batch submission
  - Possible parallel Jobs execution
- Connection with OTP
- GUI

Development

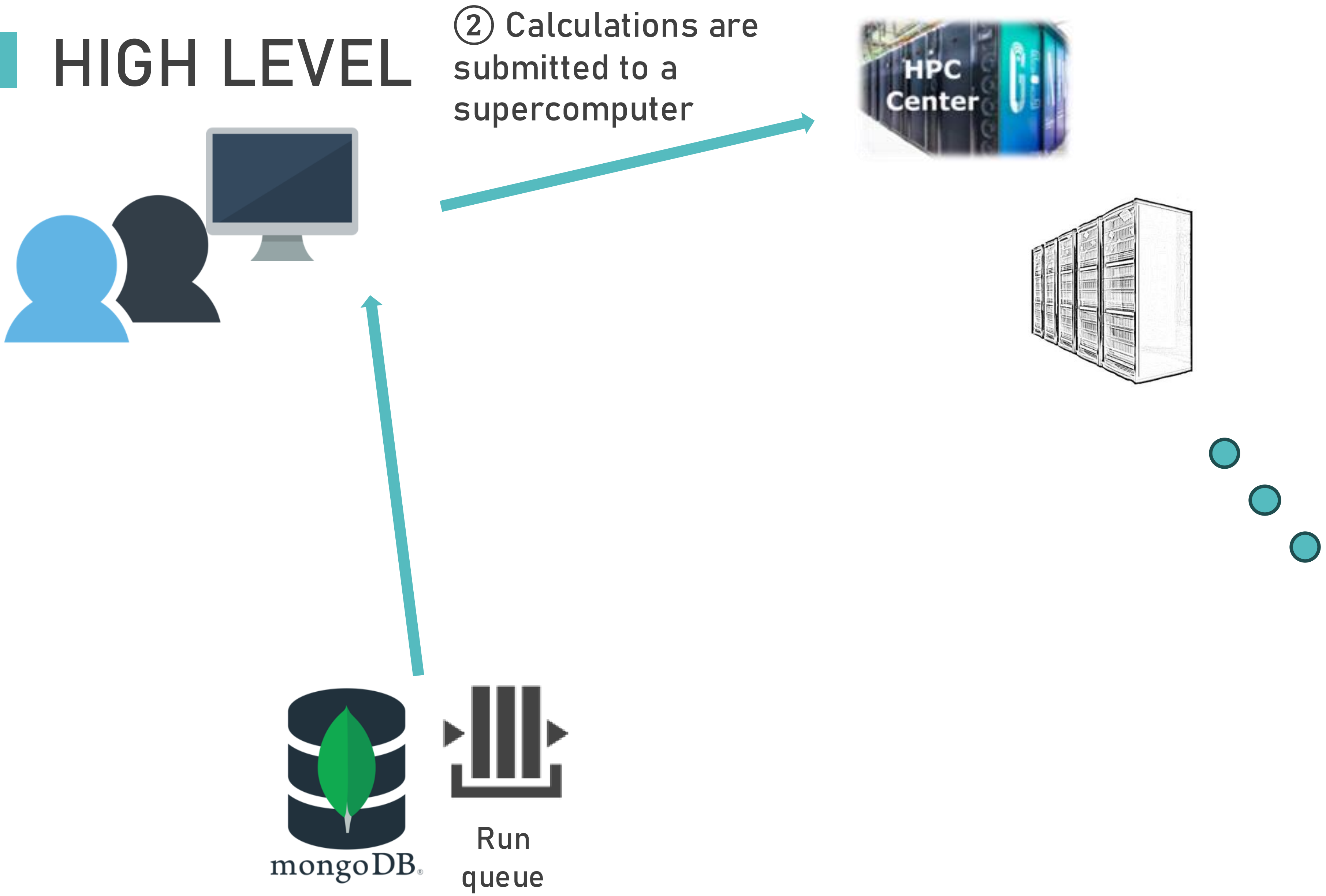
- Integration tests: real MongoDB and queueing systems with docker containers

HIGH LEVEL

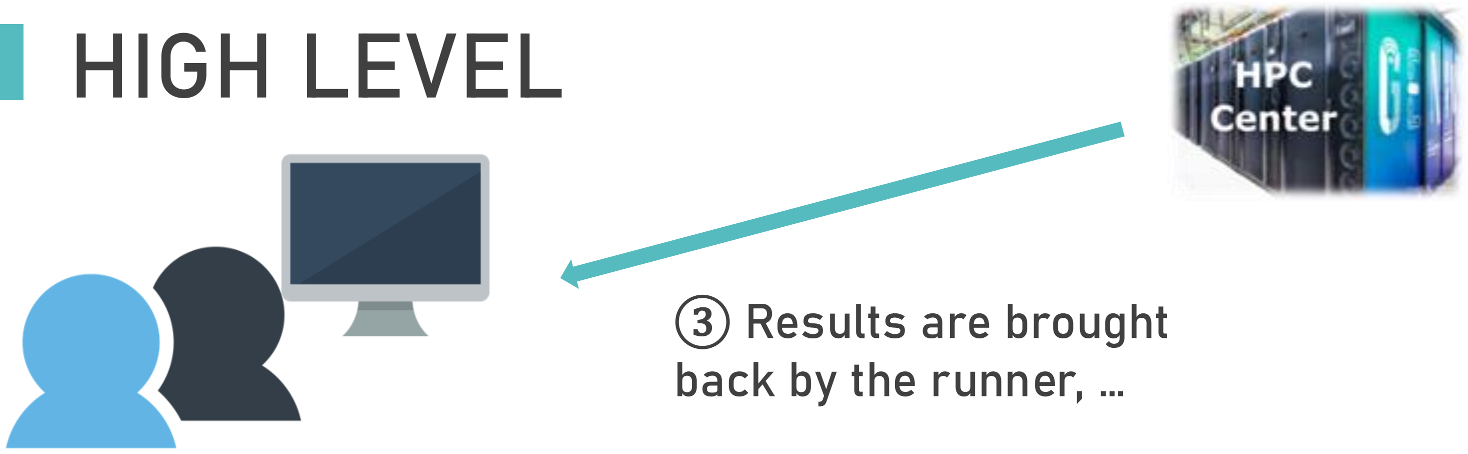


① The user adds a workflow to the list of calculations to be performed

HIGH LEVEL

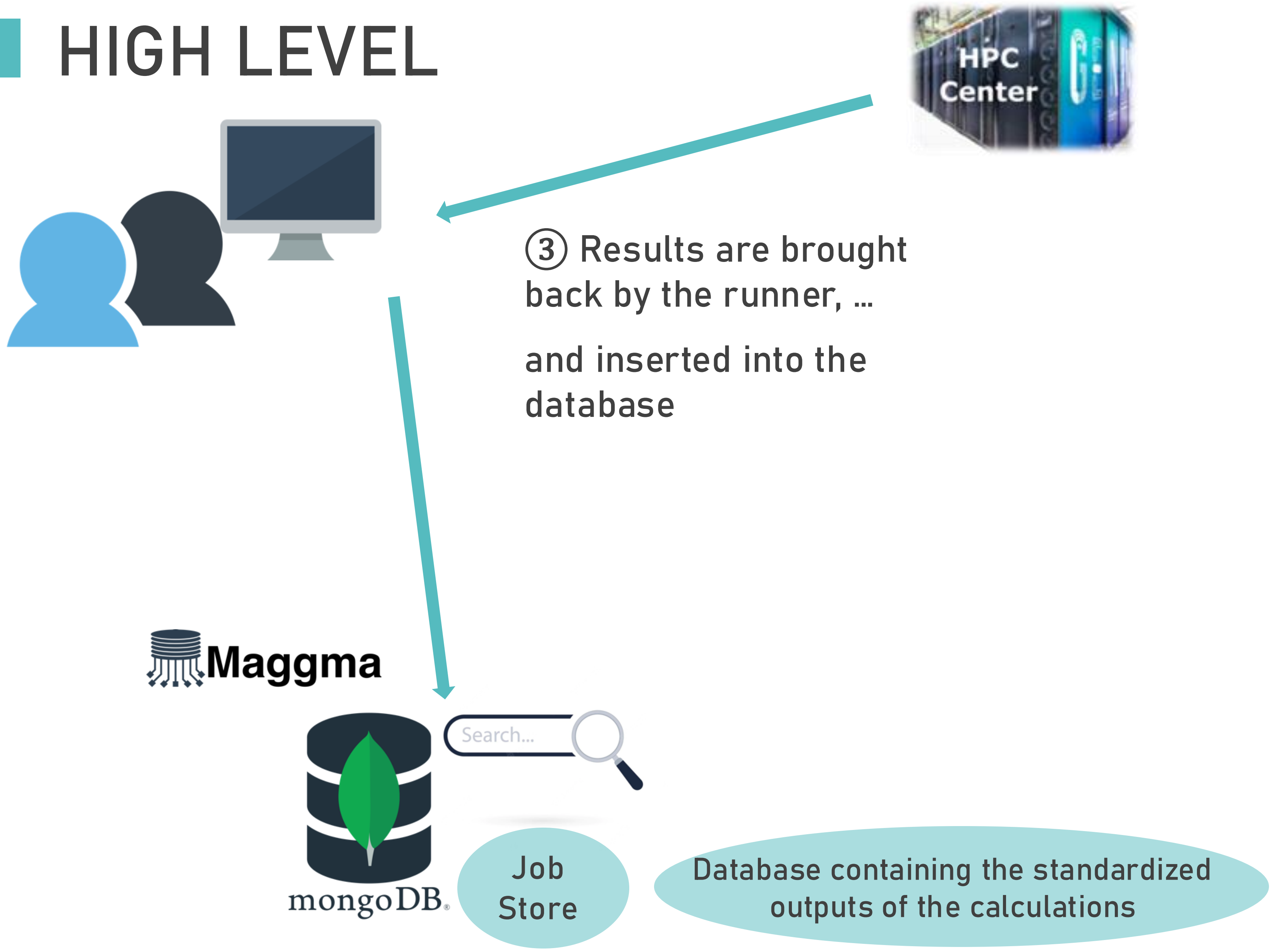

② Calculations are submitted to a supercomputer
HPC Center
mongoDB
Run queue

HIGH LEVEL



③ Results are brought back by the runner, …

HIGH LEVEL



③ Results are brought back by the runner, …

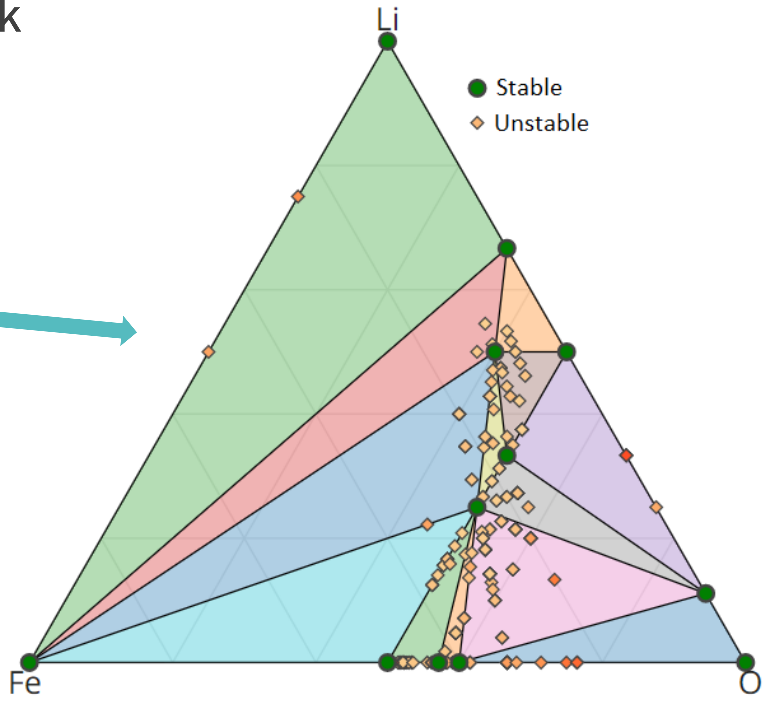
and inserted into the database

Job Store

Database containing the standardized outputs of the calculations

HIGH LEVEL

④ The user can access the results from the work station/virtual machine and perform analysis, visualizations, …



Maggma

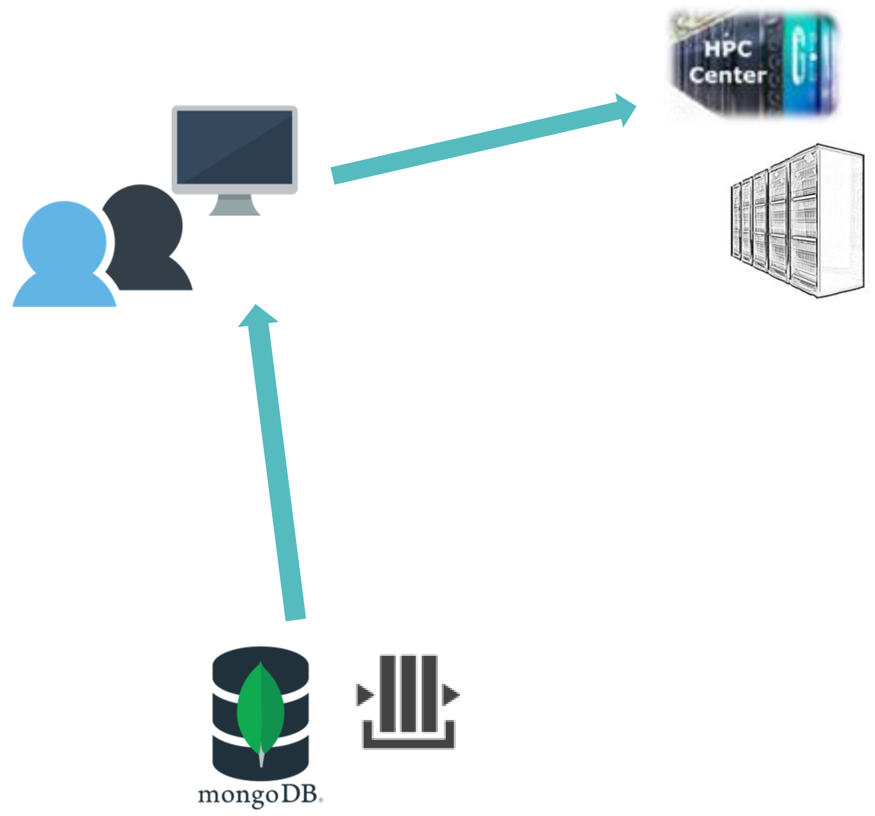
mongoDB

Job Store

Database containing the standardized outputs of the calculations

REMOTE EXECUTION

CONNECTIONS SCHEMA

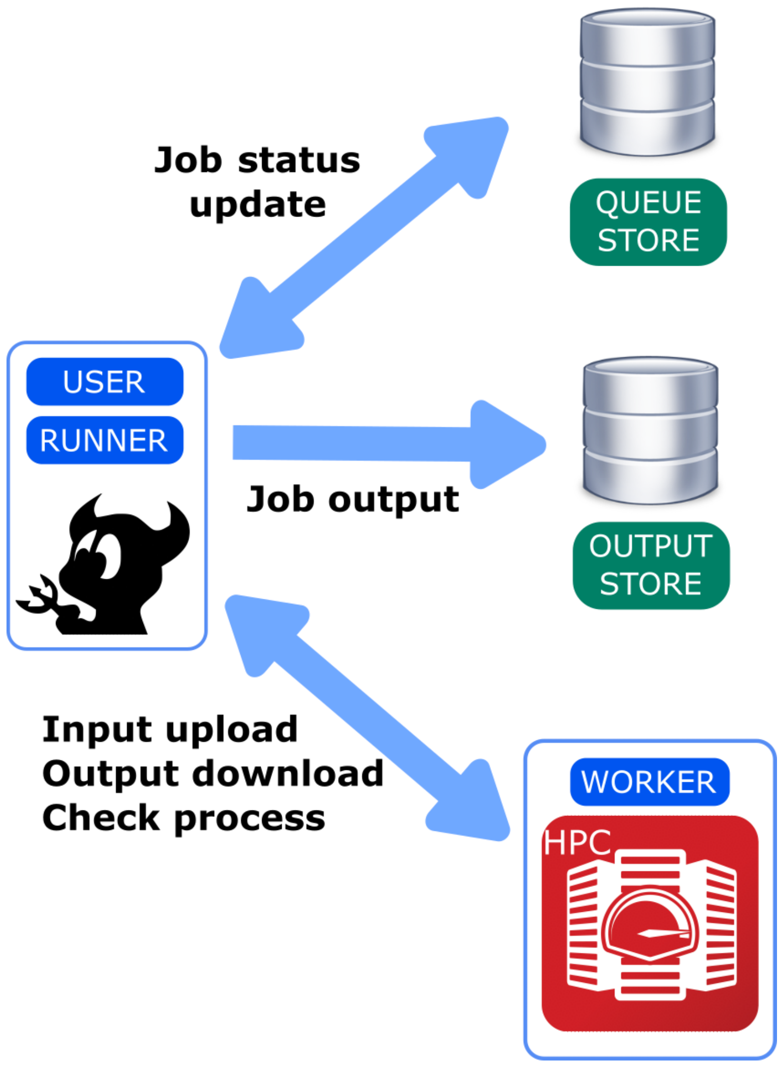


The machine hosting the system that orchestrates the execution connects to

- Storage
- Workers

And should be accessible from the user

CONNECTIONS SCHEMA – MORE ACCURATE

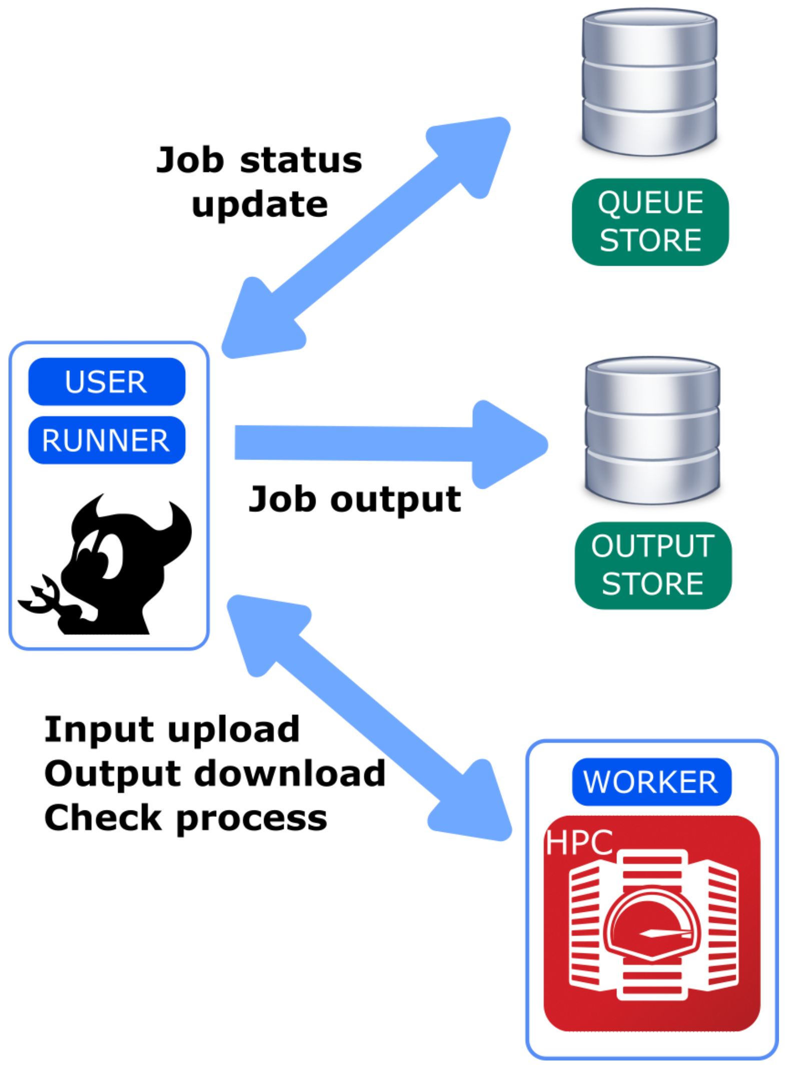


The machine hosting the system that orchestrates the execution connects to

- Database
- JobStore
- Workers

And should be accessible from the user

DATA DISTRIBUTION



2 distinct storing locations:

- Queue: Job and Flow status
  - Defined for jobflow-remote
  - Strictly MongoDB
- Output: Job outputs
  - Jobflow's JobStore
  - A Maggma Store

Queue and Output can be the same MongoDB database but contain different kind of data.

Use different collections

CREATE A FLOW

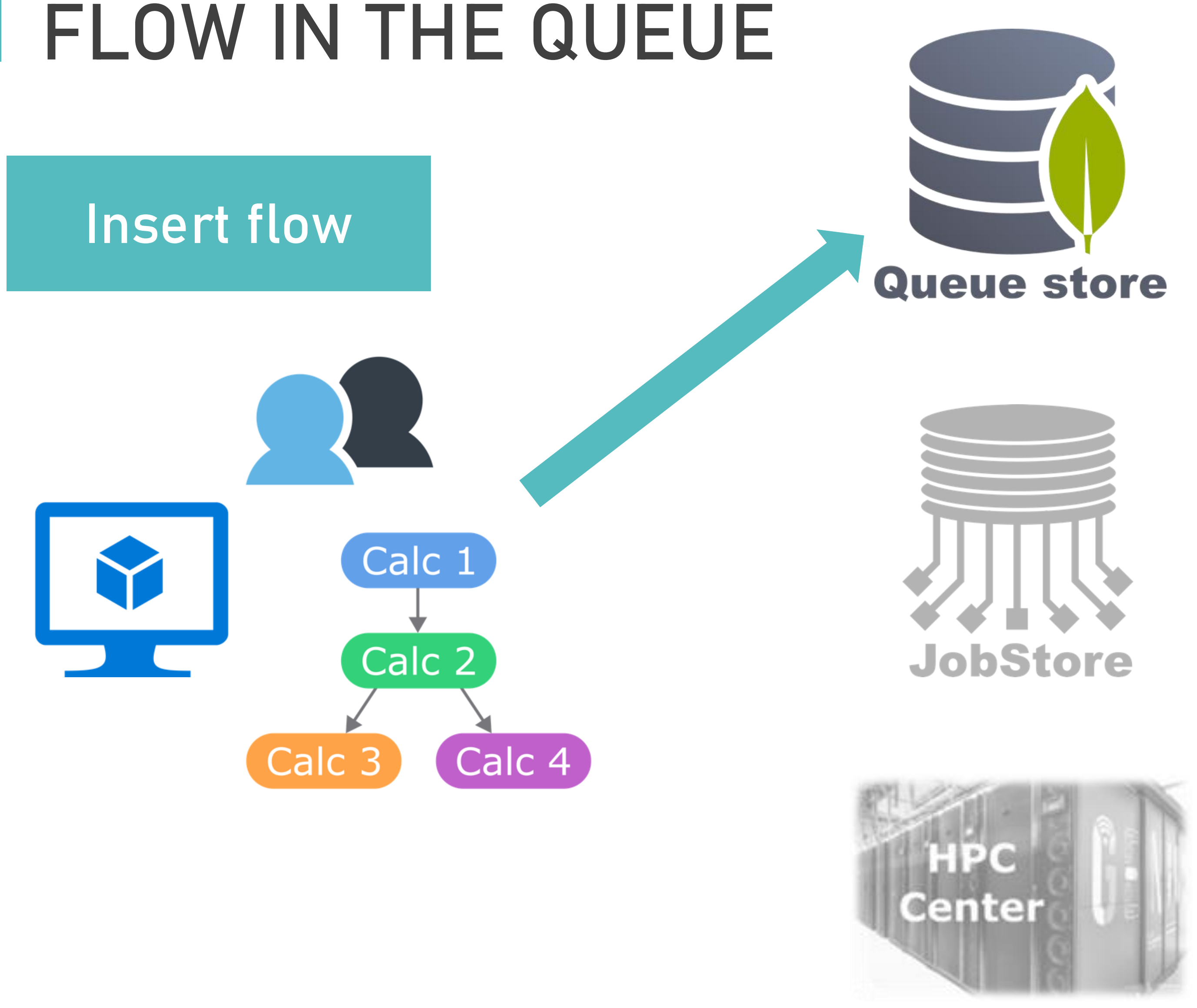
Create a Jobflow Flow object:

- As in standard Jobflow
- Jobs, Flows and Makers can be used

```
from jobflow import job, Flow
from jobflow_remote.testing import add

j1 = add(1, 2)
j2 = add(j1.output, 3)
flow = Flow([j1, j2])
```

Note: the “add” Job is imported from a package.

FLOW IN THE QUEUE



Job state **READY**

Use the `submit_flow` function from jobflow-remote

The Job inputs are stored as JSON

```
from jobflow_remote import submit_flow

output = submit_flow(flow)
print(output)
```

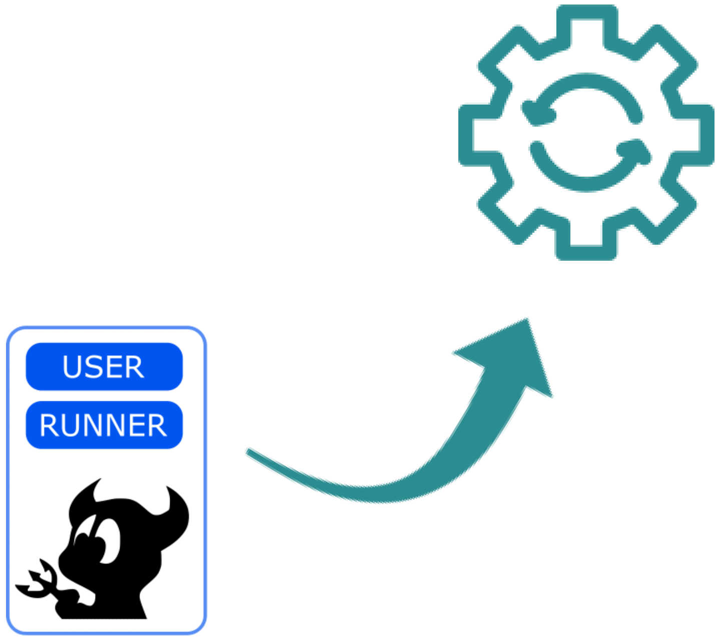
Note: A `db_id` is added as unique identifier in the DB. In `output`

Caution: the flow is not in the HPC queue at this stage

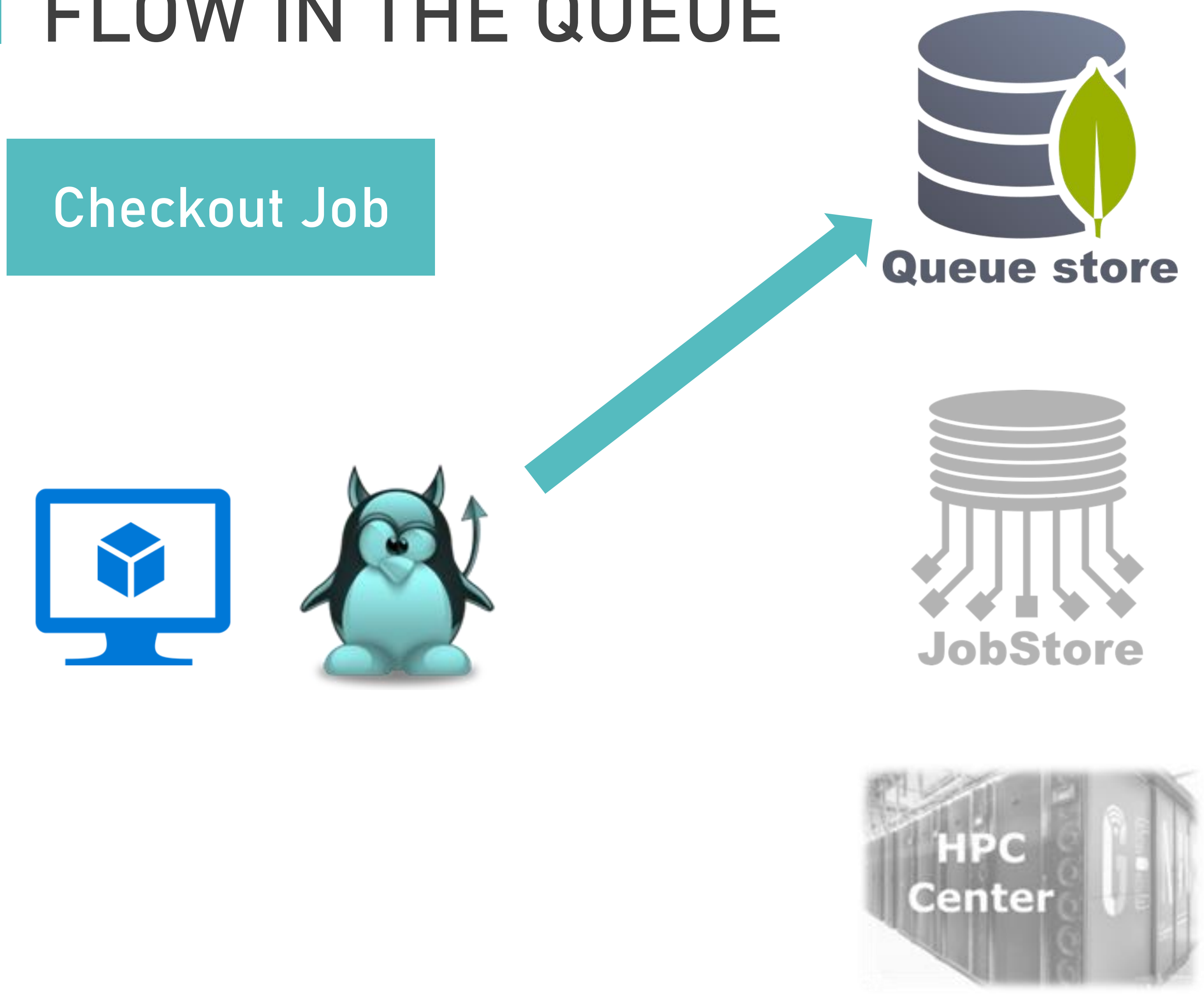
THE RUNNER

The Runner is the key element making the Job state evolve

- Daemon process(es) handling the whole execution of the jobflow workflows
- Runs in the background
- Keeps working in parallel on all the jobs that are not completed
- Possibly attempts the same action again in case of failure
- Started and monitored with the CLI



FLOW IN THE QUEUE

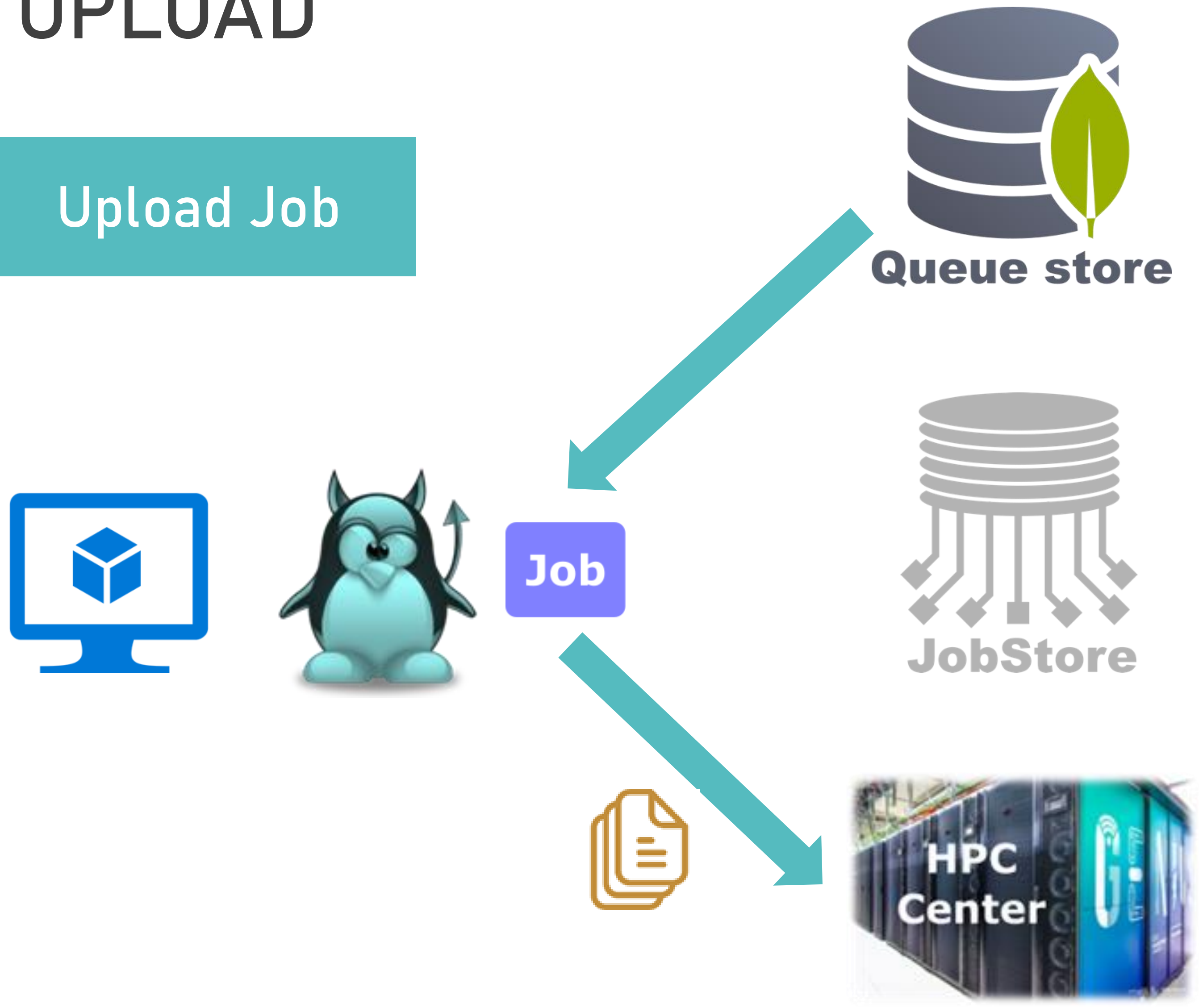


Job state **CHECKED_OUT**

- The runner acknowledges the presence of a READY Job
- Only the state of the Job is updated

UPLOAD

Upload Job



Job state UPLOADED

- Fetch the JSON serialized representation of the Job
- Resolve references
- Upload a JSON file to the selected worker
- Target is a folder determined by the job UUID

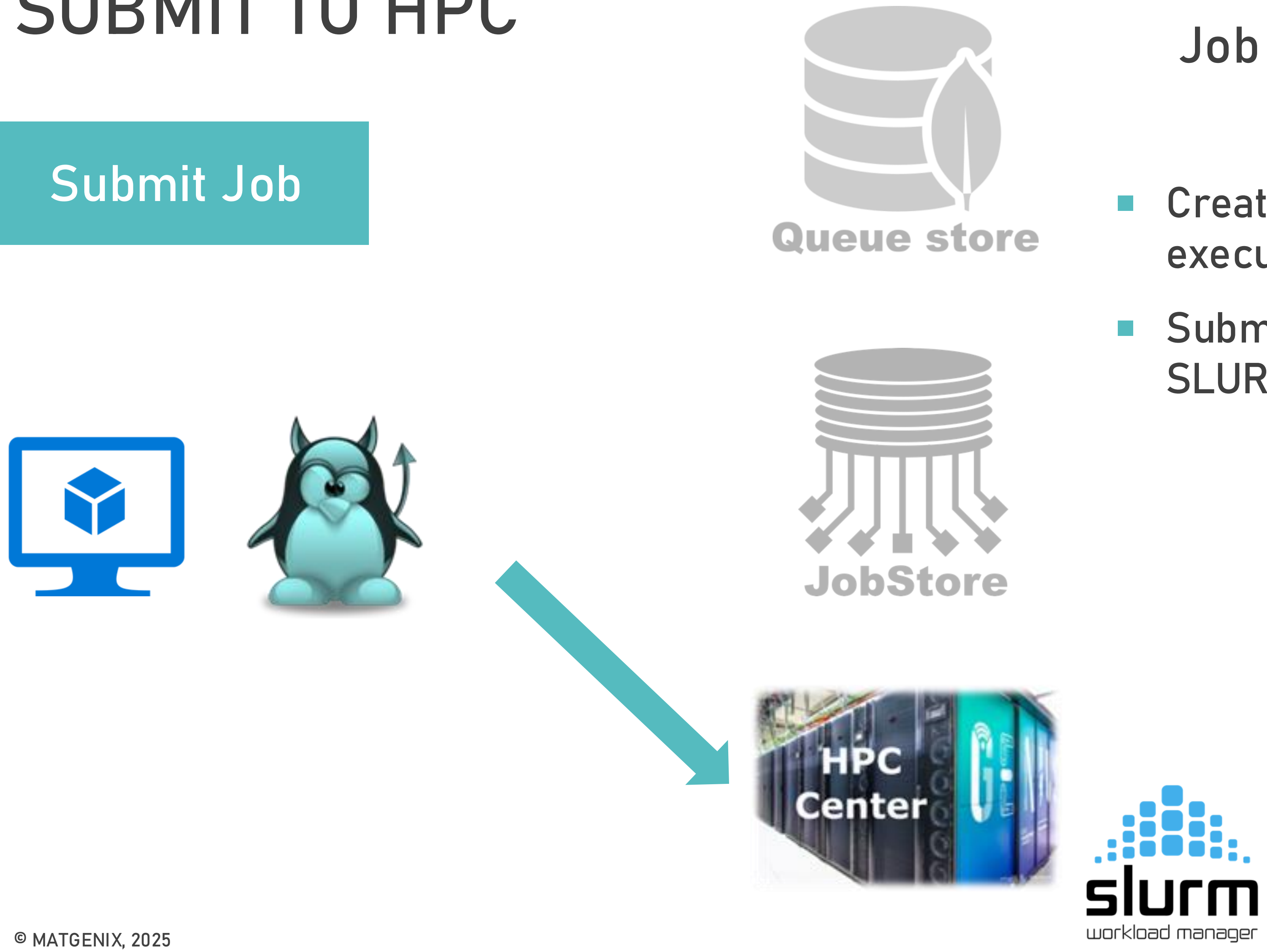
Note: when running an external code (e.g. VASP), it is not the input file of the code that is uploaded

SUBMIT TO HPC

Submit Job

Job state SUBMITTED

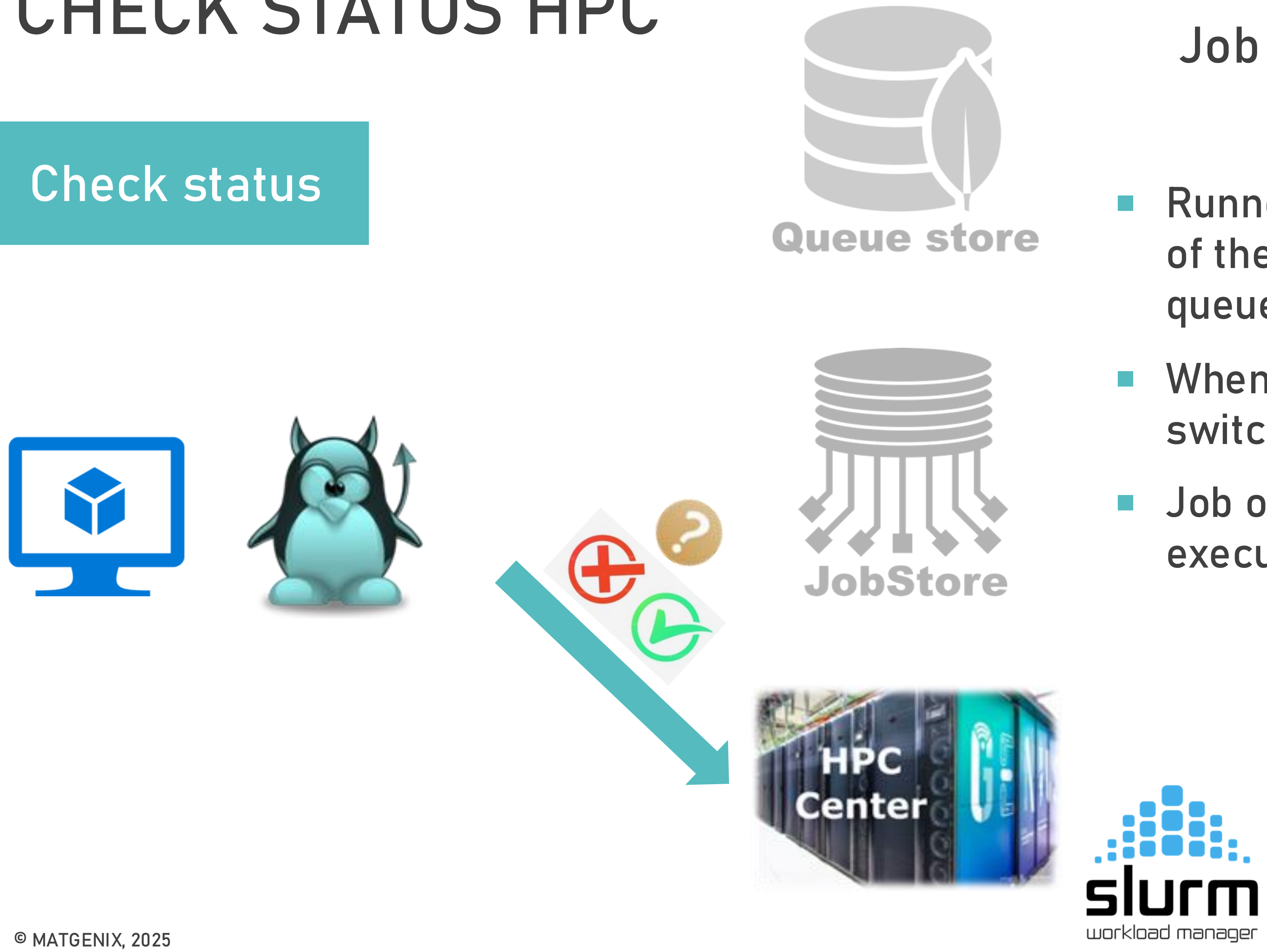
- Create a submission script in the execution folder of the Job
- Submit to the system queue (e.g. SLURM, PBS, …)



CHECK STATUS HPC

Check status

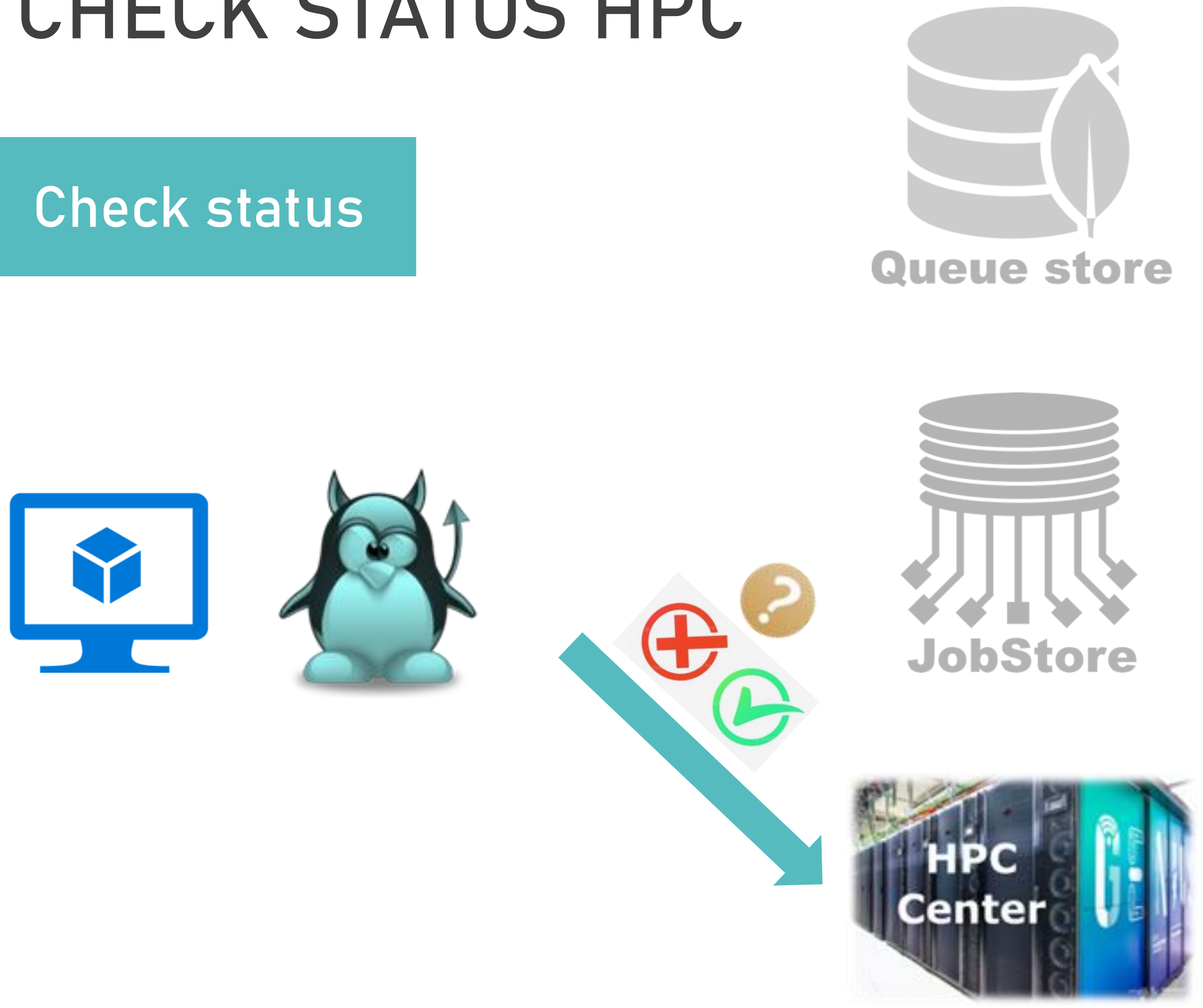
Job state RUNNING



- Runner regularly check the status of the Job submitted to the HPC queue.
- When the Job starts, the status is switched to RUNNING
- Job object is deserialized and executed like a normal Jobflow Job

CHECK STATUS HPC

Check status



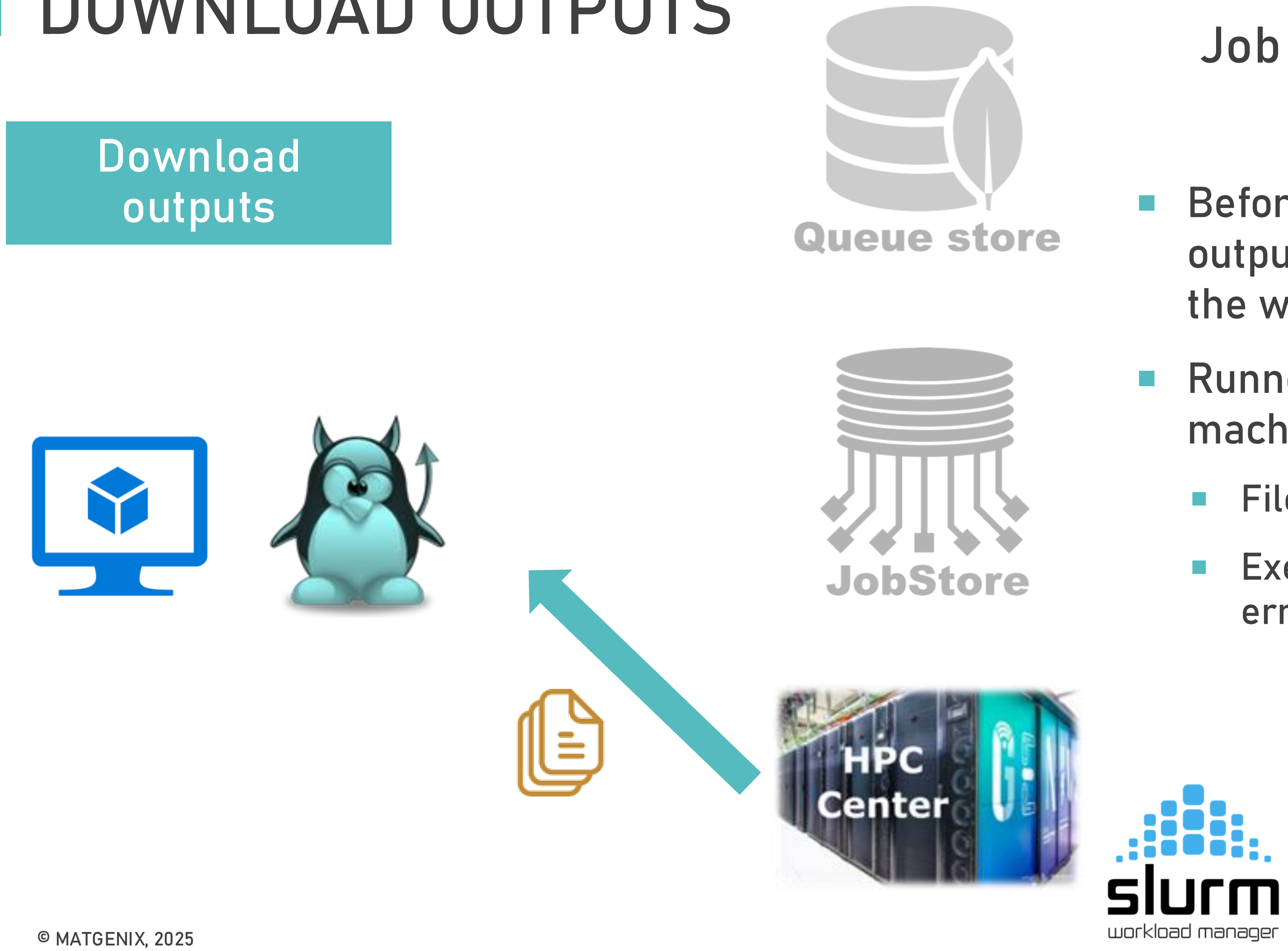
Job state TERMINATED

- Runner regularly check the status of the Job submitted to the HPC queue.
- When the Job is finished, the status is switched to TERMINATED

TERMINATED means just that the job in the queue has stopped running. No implication on errors

Name will likely change to EXECUTED

DOWNLOAD OUTPUTS

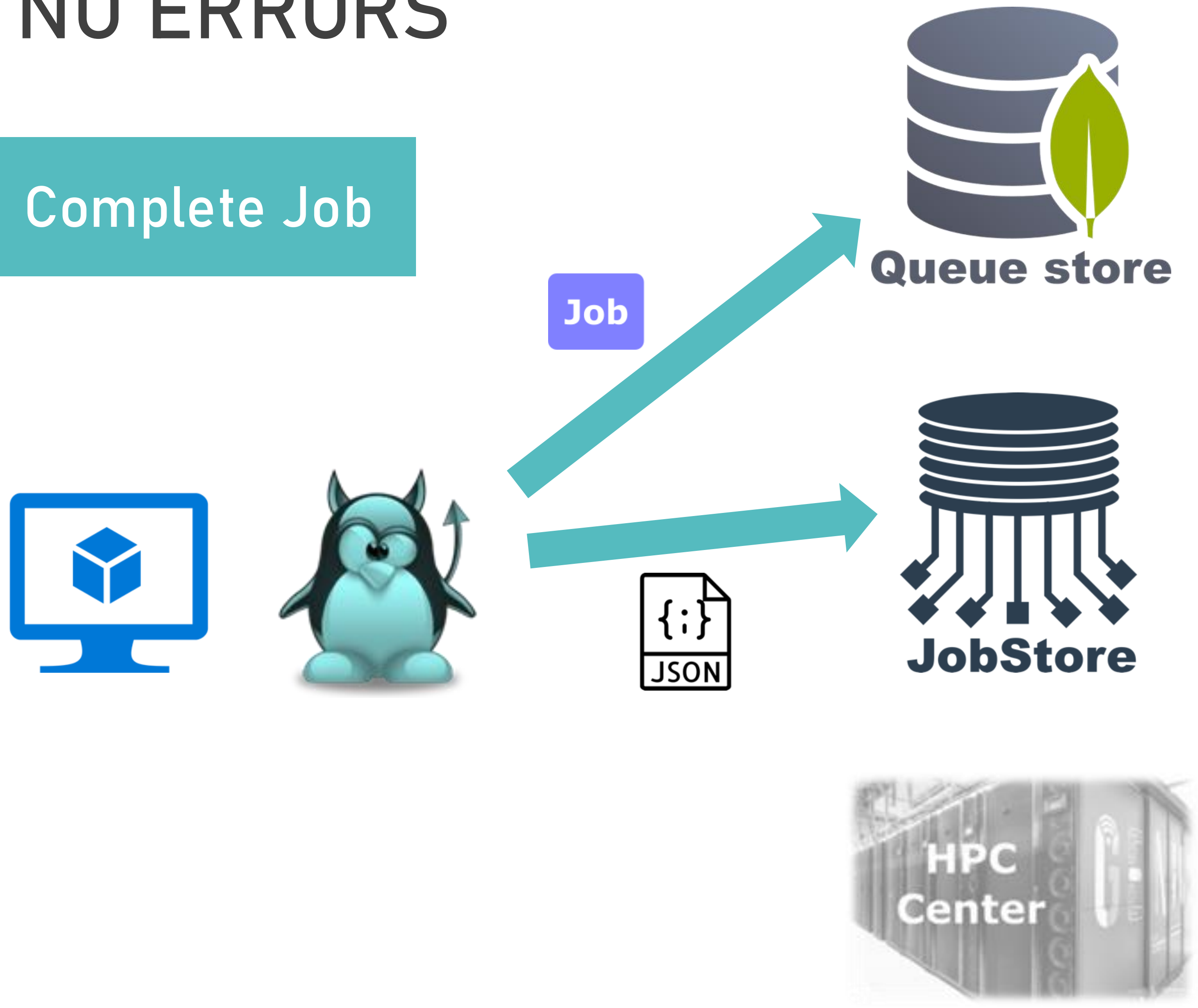


Job state DOWNLOADED

- Before finishing Job writes the output to a file-based JobStore on the worker
- Runner download to the local machine:
  - File-based JobStore
  - Execution information (e.g. timings, errors, …)

NO ERRORS

Complete Job



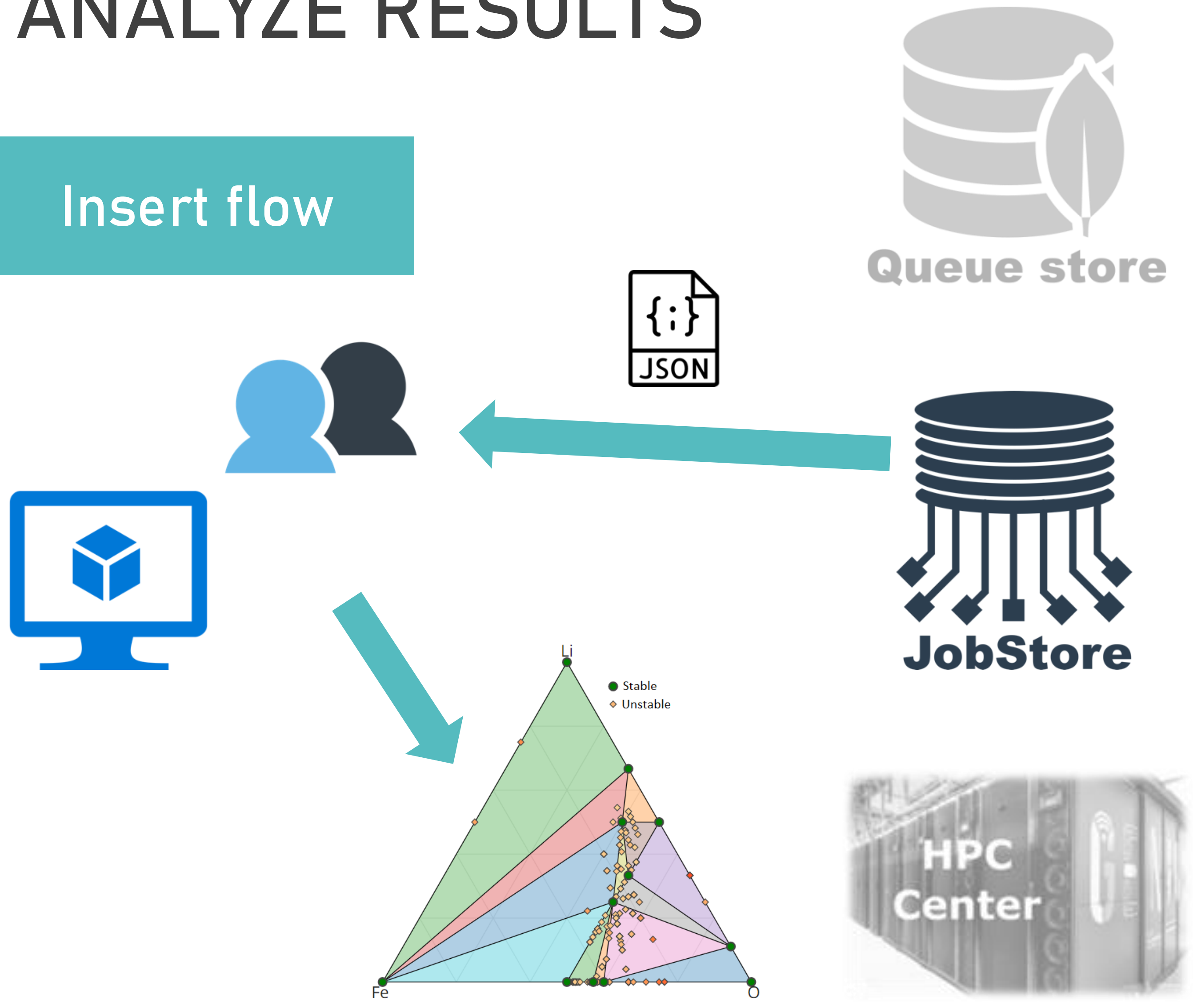
Job state COMPLETED

If no errors during Job execution

- Jobs execution information in the Queue store
- Job outputs inserted in the actual JobStore

ANALYZE RESULTS

Insert flow



Job state COMPLETED

Retrieve Outputs from JobStore to analyze, plot data, …

The outputs in JobStore are the same as in standard Jobflow execution

JOB ERRORS

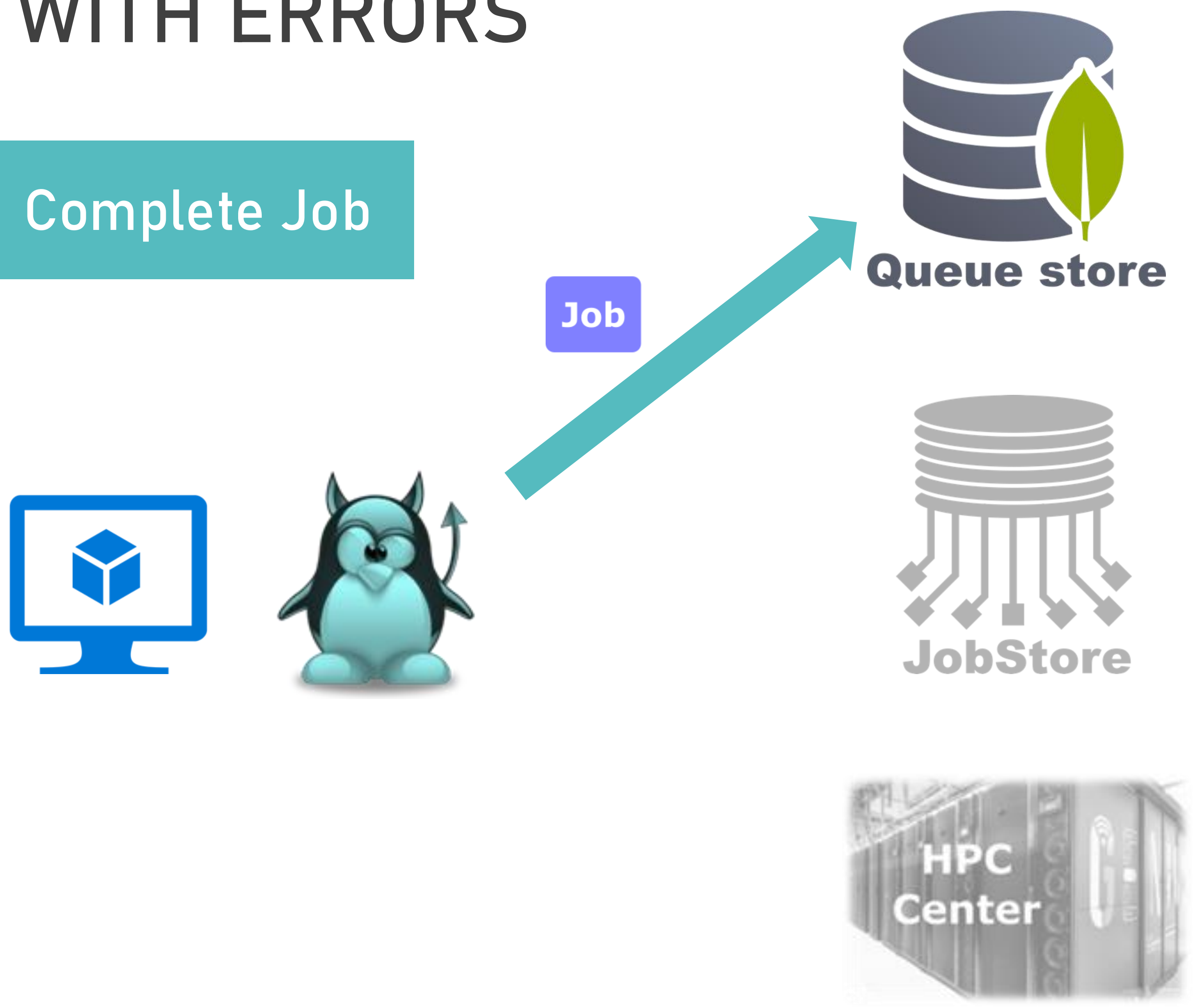
First category of errors: Job failure

- Job raises an exception during execution
- Several potential causes:
  - Bad inputs
  - External code does not complete successfully
  - External code fails
  - Bug in the Job code
- Python code running Jobflow on the worker is not killed

➡ Job state FAILED

WITH ERRORS

Complete Job



Job state FAILED

If errors during Job execution

- Job execution information has been downloaded
- Jobs execution information in the Queue store
  - Including errors messages
- No data in JobStore

Note: FAILED means error during Job execution (the Runner procedure was executed correctly)

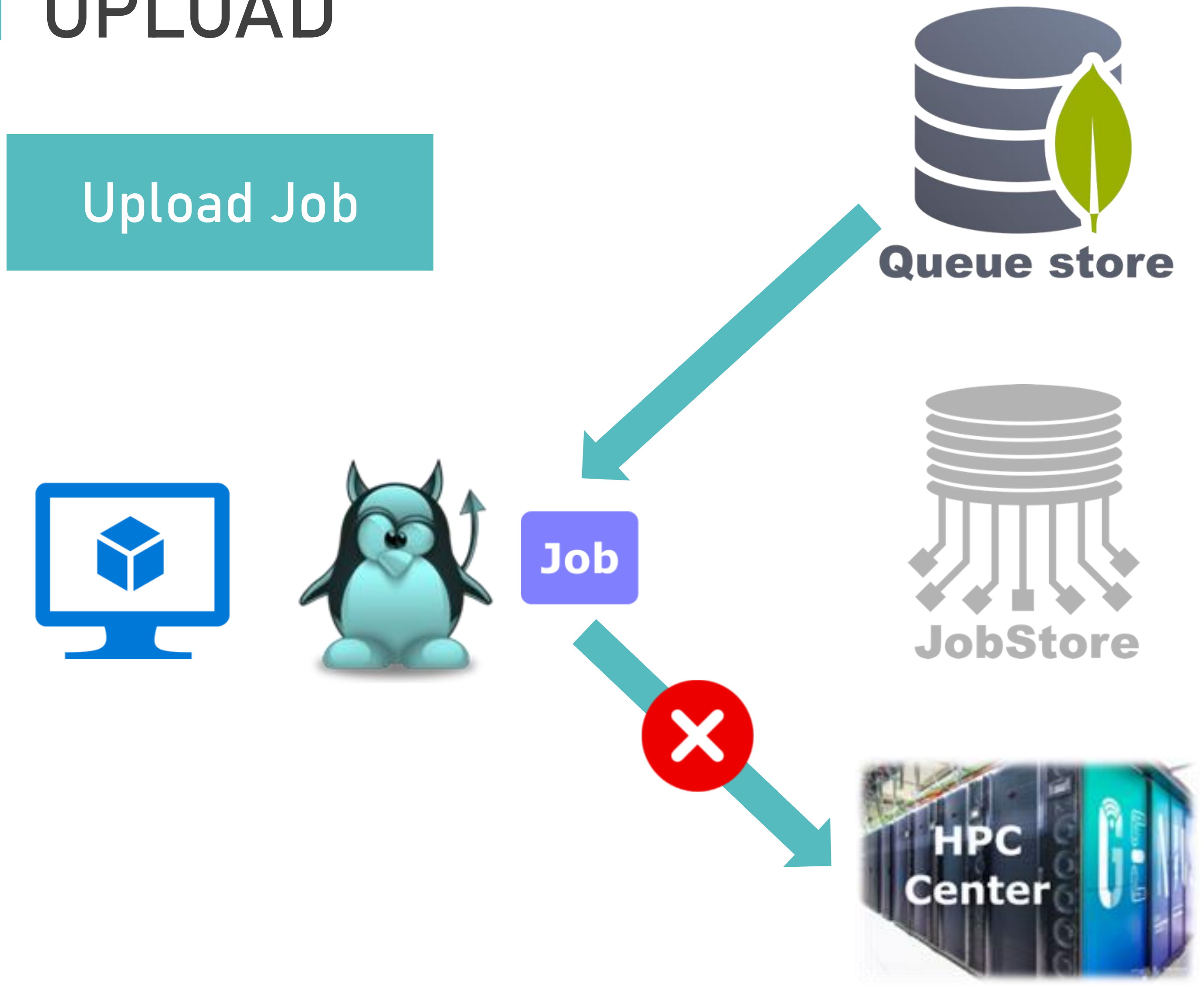
RUNNER ERRORS

Second category of errors: Runner execution error

- The runner fails while performing one of the actions
- Several potential causes:
  - Connection issues (worker, JobStore)
  - HPC queueing system errors
  - Queued job unexpectedly killed
  - Queued job reached walltime
  - …
- The Runner attempts the action multiple times (exponential backoff)

➡ Job state REMOTE_ERROR

UPLOAD



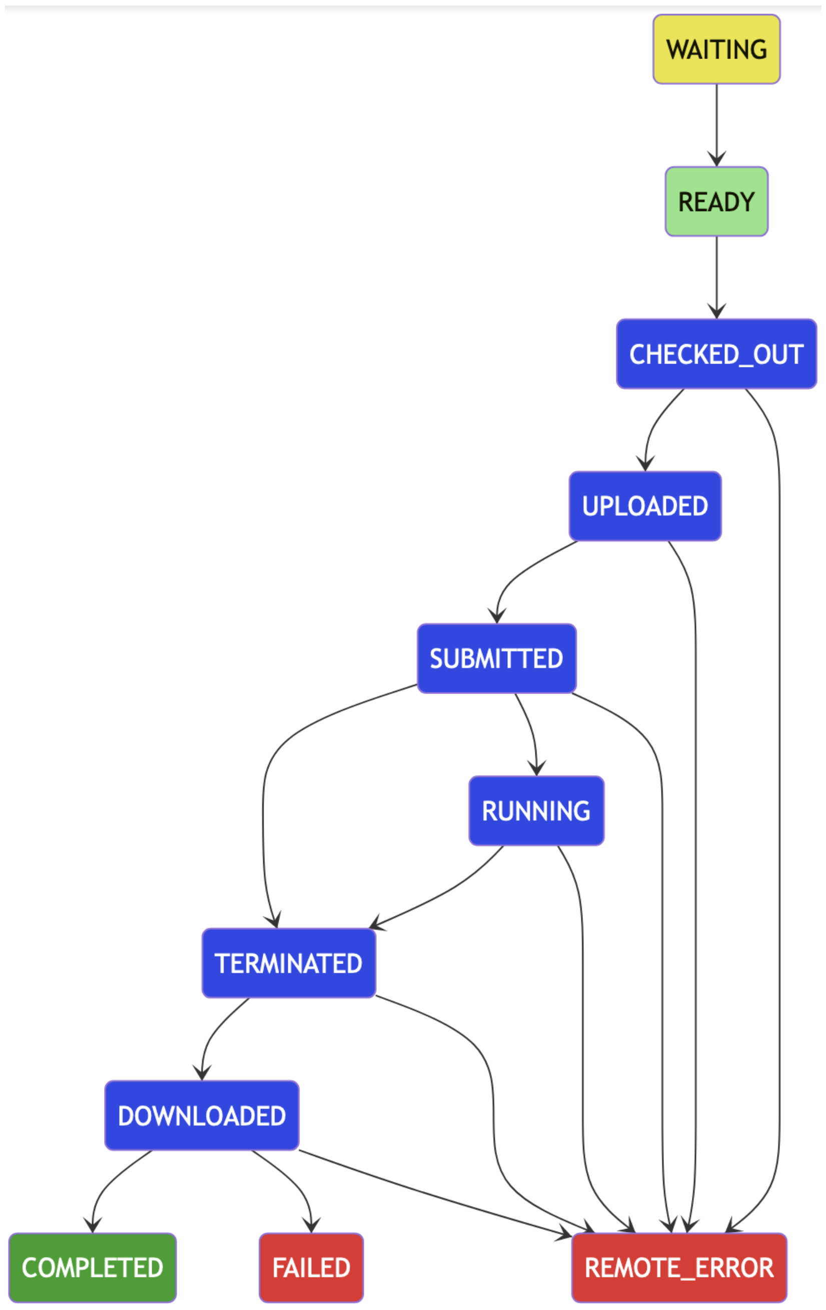
Job state REMOTE_ERROR

- Fetch the JSON serialized representation of the Job
- Upload of the JSON file to the worker fails due to connection issue
- After failing multiple times, the Job is set to the REMOTE_ERROR state

Note: REMOTE_ERROR is independent from Job successful execution

STATES EVOLUTION RECAP

- All possible states evolutions during Runner execution
- WAITING state: a Job wating for outputs from a previous Job not yet completed
  - Will switch to READY when all previous Jobs are completed.

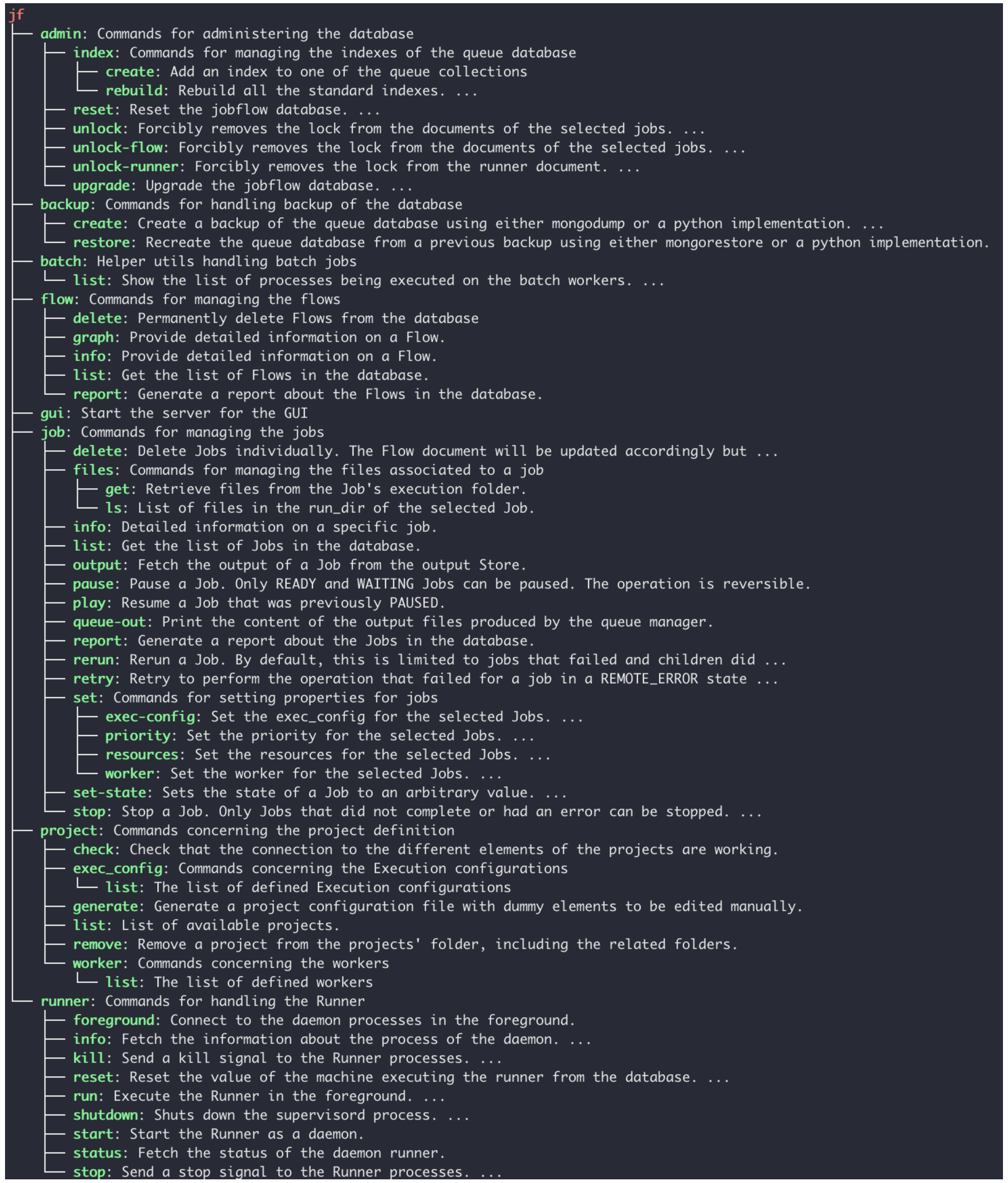


INTERACTING WITH JOBFLOW-REMOTE

COMMAND LINE INTERFACE

CLI is the main entry point for interacting with jobflow-remote

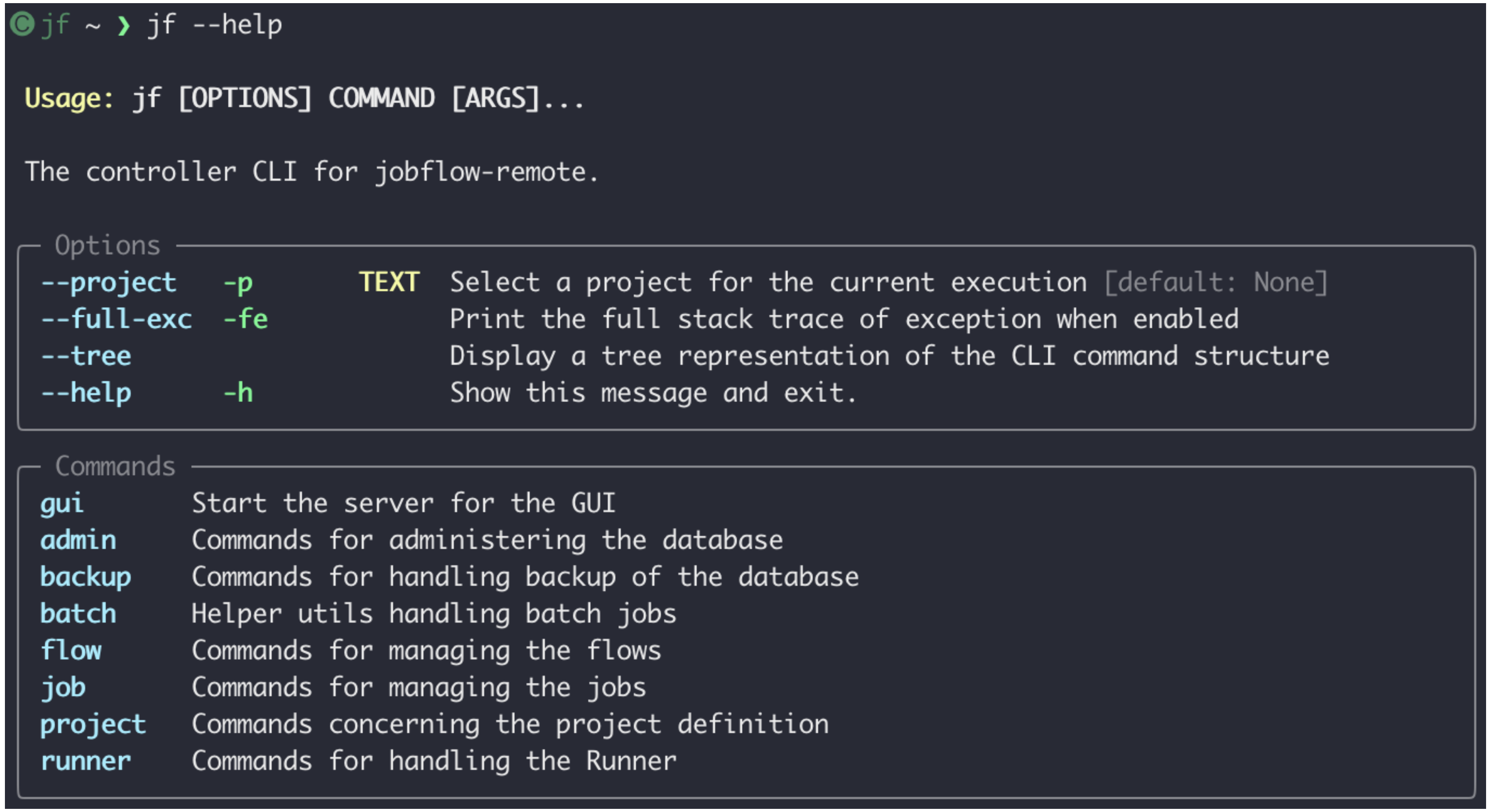
- `jf` command
- Several commands and subcommands
- Tree representation
- Interfaces with the different projects



```
jf
├── admin: Commands for administering the database
│   ├── index: Commands for managing the indexes of the queue database
│   │   ├── create: Add an index to one of the queue collections
│   │   └── rebuild: Rebuild all the standard indexes. ...
│   ├── reset: Reset the jobflow database. ...
│   ├── unlock: Forcibly removes the lock from the documents of the selected jobs. ...
│   ├── unlock-flow: Forcibly removes the lock from the documents of the selected jobs. ...
│   ├── unlock-runner: Forcibly removes the lock from the runner document. ...
│   └── upgrade: Upgrade the jobflow database. ...
├── backup: Commands for handling backup of the database
│   ├── create: Create a backup of the queue database using either mongodump or a python implementation. ...
│   └── restore: Recreate the queue database from a previous backup using either mongorestore or a python implementation.
├── batch: Helper utils handling batch jobs
│   └── list: Show the list of processes being executed on the batch workers. ...
├── flow: Commands for managing the flows
│   ├── delete: Permanently delete Flows from the database
│   ├── graph: Provide detailed information on a Flow.
│   ├── info: Provide detailed information on a Flow.
│   ├── list: Get the list of Flows in the database.
│   └── report: Generate a report about the Flows in the database.
├── gui: Start the server for the GUI
├── job: Commands for managing the jobs
│   ├── delete: Delete Jobs individually. The Flow document will be updated accordingly but ...
│   ├── files: Commands for managing the files associated to a job
│   │   ├── get: Retrieve files from the Job's execution folder.
│   │   └── ls: List of files in the run_dir of the selected Job.
│   ├── info: Detailed information on a specific job.
│   ├── list: Get the list of Jobs in the database.
│   ├── output: Fetch the output of a Job from the output Store.
│   ├── pause: Pause a Job. Only READY and WAITING Jobs can be paused. The operation is reversible.
│   ├── play: Resume a Job that was previously PAUSED.
│   ├── queue-out: Print the content of the output files produced by the queue manager.
│   ├── report: Generate a report about the Jobs in the database.
│   ├── rerun: Rerun a Job. By default, this is limited to jobs that failed and children did ...
│   ├── retry: Retry to perform the operation that failed for a job in a REMOTE_ERROR state ...
│   ├── set: Commands for setting properties for jobs
│   │   ├── exec-config: Set the exec_config for the selected Jobs. ...
│   │   ├── priority: Set the priority for the selected Jobs. ...
│   │   ├── resources: Set the resources for the selected Jobs. ...
│   │   └── worker: Set the worker for the selected Jobs. ...
│   ├── set-state: Sets the state of a Job to an arbitrary value. ...
│   └── stop: Stop a Job. Only Jobs that did not complete or had an error can be stopped. ...
├── project: Commands concerning the project definition
│   ├── check: Check that the connection to the different elements of the projects are working.
│   ├── exec_config: Commands concerning the Execution configurations
│   │   └── list: The list of defined Execution configurations
│   ├── generate: Generate a project configuration file with dummy elements to be edited manually.
│   ├── list: List of available projects.
│   ├── remove: Remove a project from the projects' folder, including the related folders.
│   └── worker: Commands concerning the workers
│       └── list: The list of defined workers
└── runner: Commands for handling the Runner
    ├── foreground: Connect to the daemon processes in the foreground.
    ├── info: Fetch the information about the process of the daemon. ...
    ├── kill: Send a kill signal to the Runner processes. ...
    ├── reset: Reset the value of the machine executing the runner from the database. ...
    ├── run: Execute the Runner in the foreground. ...
    ├── shutdown: Shuts down the supervisord process. ...
    ├── start: Start the Runner as a daemon.
    ├── status: Fetch the status of the daemon runner.
    └── stop: Send a stop signal to the Runner processes. ...
```

CLI --HELP

Every command has a `--help`/`-h` option for details and list of options

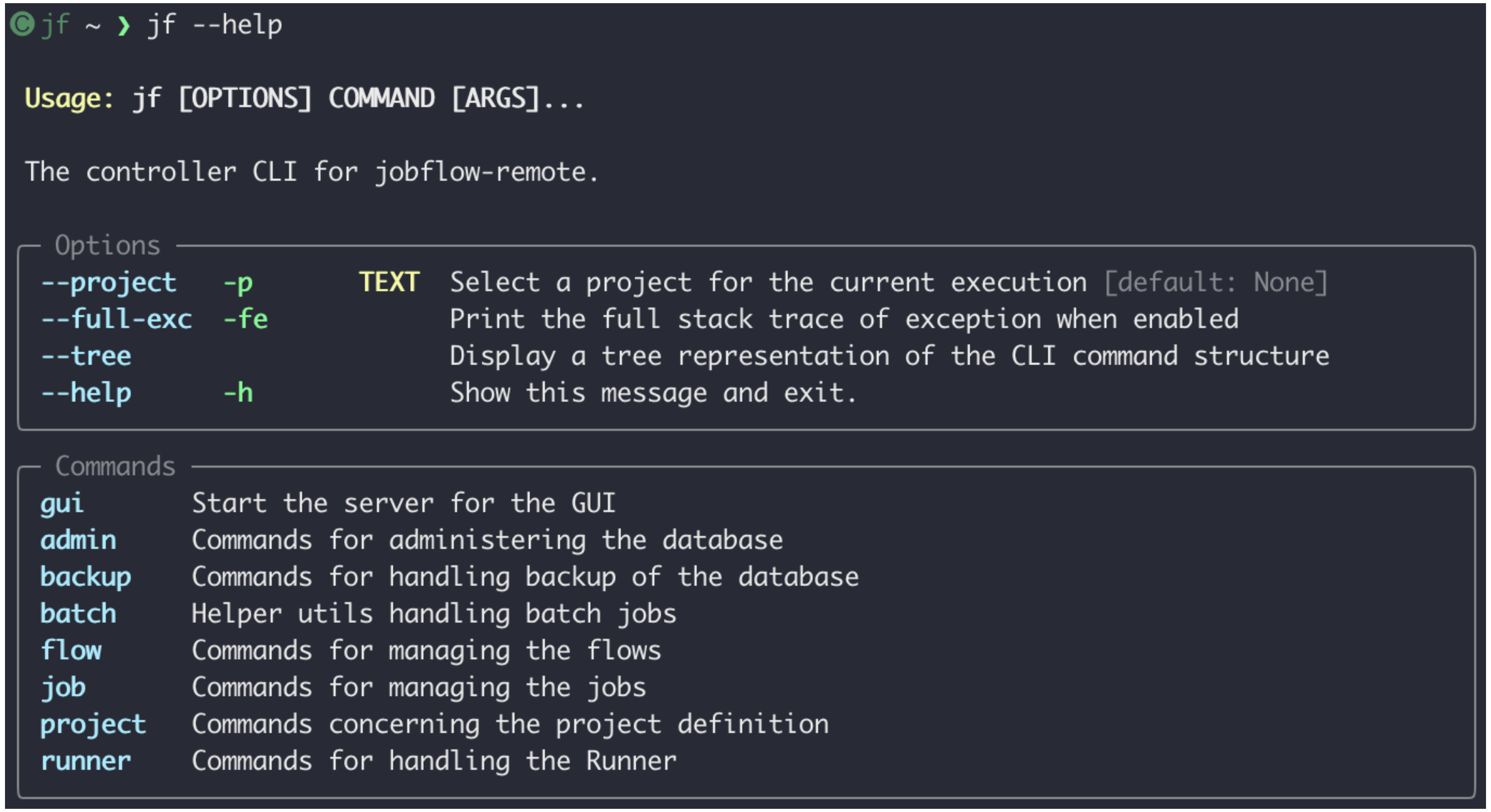


```
jf ~ ❯ jf --help

Usage: jf [OPTIONS] COMMAND [ARGS]...

The controller CLI for jobflow-remote.

╭─ Options ──────────────────────────────────────────────────────────────────╮
│ --project    -p      TEXT  Select a project for the current execution [default: None] │
│ --full-exc   -fe           Print the full stack trace of exception when enabled        │
│ --tree                     Display a tree representation of the CLI command structure  │
│ --help       -h            Show this message and exit.                                 │
╰────────────────────────────────────────────────────────────────────────────╯
╭─ Commands ─────────────────────────────────────────────────────────────────╮
│ gui        Start the server for the GUI                                    │
│ admin      Commands for administering the database                         │
│ backup     Commands for handling backup of the database                    │
│ batch      Helper utils handling batch jobs                                │
│ flow       Commands for managing the flows                                 │
│ job        Commands for managing the jobs                                  │
│ project    Commands concerning the project definition                      │
│ runner     Commands for handling the Runner                                │
╰────────────────────────────────────────────────────────────────────────────╯
```

CLI OVERVIEW

Several main level functionalities:

- admin: handle the queue DB
- project: manage projects configurations
- runner: control the Runner
- job: query and control the Jobs in the queue DB
- flow: query and control the Flows in the queue DB
- backup: import/export backup
- batch: monitor batch jobs
- gui: start the GUI

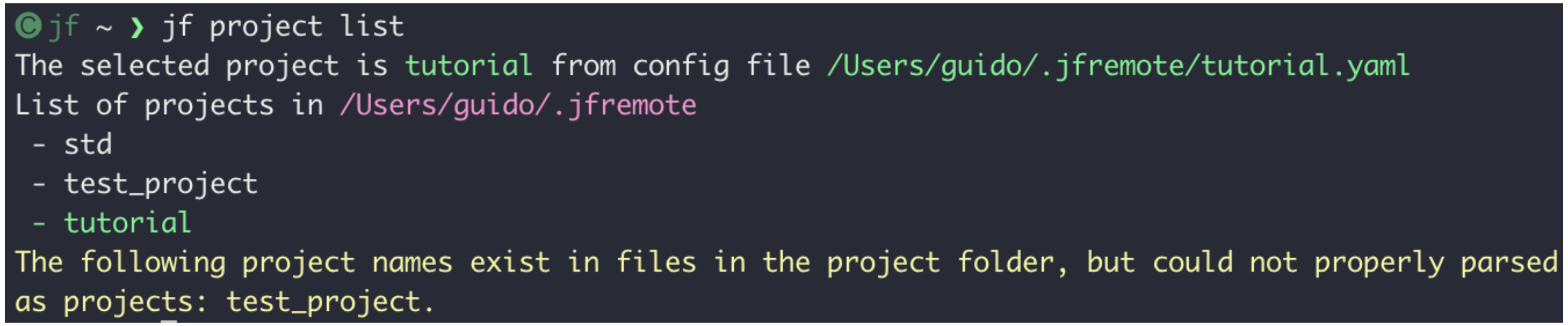


```
jf ~ ❯ jf --help

Usage: jf [OPTIONS] COMMAND [ARGS]...

The controller CLI for jobflow-remote.

╭─ Options ──────────────────────────────────────────────────────────────────────────────────╮
│ --project     -p      TEXT  Select a project for the current execution [default: None]     │
│ --full-exc    -fe           Print the full stack trace of exception when enabled           │
│ --tree                      Display a tree representation of the CLI command structure     │
│ --help        -h            Show this message and exit.                                    │
╰────────────────────────────────────────────────────────────────────────────────────────────╯
╭─ Commands ─────────────────────────────────────────────────────────────────────────────────╮
│ gui       Start the server for the GUI                                                     │
│ admin     Commands for administering the database                                          │
│ backup    Commands for handling backup of the database                                     │
│ batch     Helper utils handling batch jobs                                                 │
│ flow      Commands for managing the flows                                                  │
│ job       Commands for managing the jobs                                                   │
│ project   Commands concerning the project definition                                       │
│ runner    Commands for handling the Runner                                                 │
╰────────────────────────────────────────────────────────────────────────────────────────────╯
```

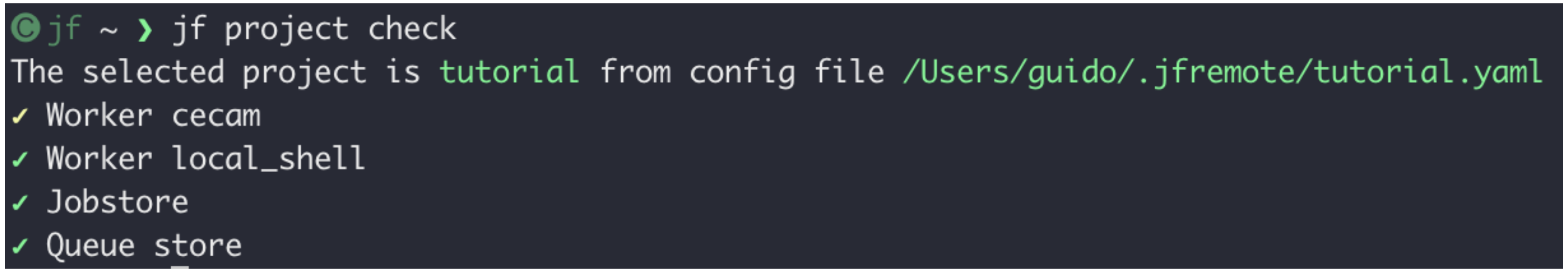
CLI - PROJECT

project: manage projects configurations

- List of current projects
- Check the connections to workers and databases



```
jf ~ ❯ jf project list
The selected project is tutorial from config file /Users/guido/.jfremote/tutorial.yaml
List of projects in /Users/guido/.jfremote
 - std
 - test_project
 - tutorial
The following project names exist in files in the project folder, but could not properly parsed
as projects: test_project.
```

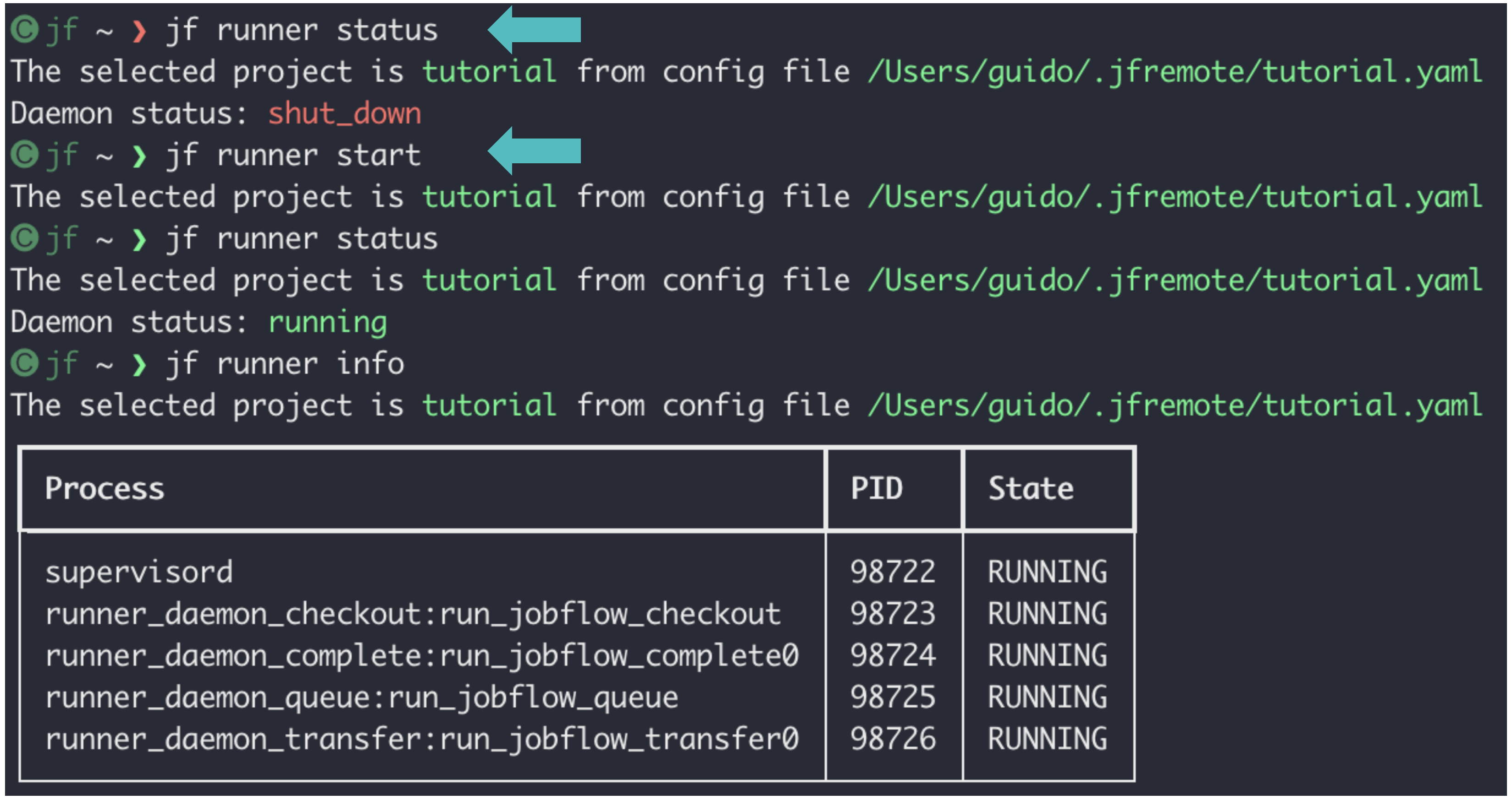


```
jf ~ ❯ jf project check
The selected project is tutorial from config file /Users/guido/.jfremote/tutorial.yaml
✓ Worker cecam
✓ Worker local_shell
✓ Jobstore
✓ Queue store
```

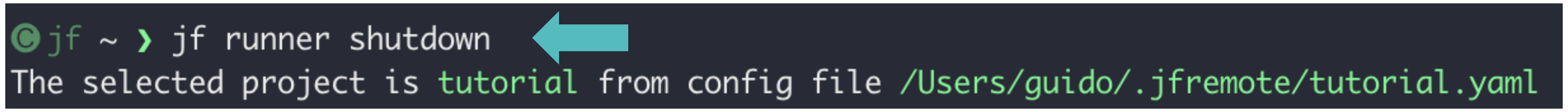
CLI - RUNNER

runner: control the Runner

- Start
- Stop
- Status
- Subprocesses information
- Kill



```
Ⓒjf ~ ❯ jf runner status
The selected project is tutorial from config file /Users/guido/.jfremote/tutorial.yaml
Daemon status: shut_down
Ⓒjf ~ ❯ jf runner start
The selected project is tutorial from config file /Users/guido/.jfremote/tutorial.yaml
Ⓒjf ~ ❯ jf runner status
The selected project is tutorial from config file /Users/guido/.jfremote/tutorial.yaml
Daemon status: running
Ⓒjf ~ ❯ jf runner info
The selected project is tutorial from config file /Users/guido/.jfremote/tutorial.yaml

┌─────────────────────────────────────────────┬───────┬─────────┐
│ Process                                     │ PID   │ State   │
├─────────────────────────────────────────────┼───────┼─────────┤
│ supervisord                                 │ 98722 │ RUNNING │
│ runner_daemon_checkout:run_jobflow_checkout │ 98723 │ RUNNING │
│ runner_daemon_complete:run_jobflow_complete0│ 98724 │ RUNNING │
│ runner_daemon_queue:run_jobflow_queue       │ 98725 │ RUNNING │
│ runner_daemon_transfer:run_jobflow_transfer0│ 98726 │ RUNNING │
└─────────────────────────────────────────────┴───────┴─────────┘
```

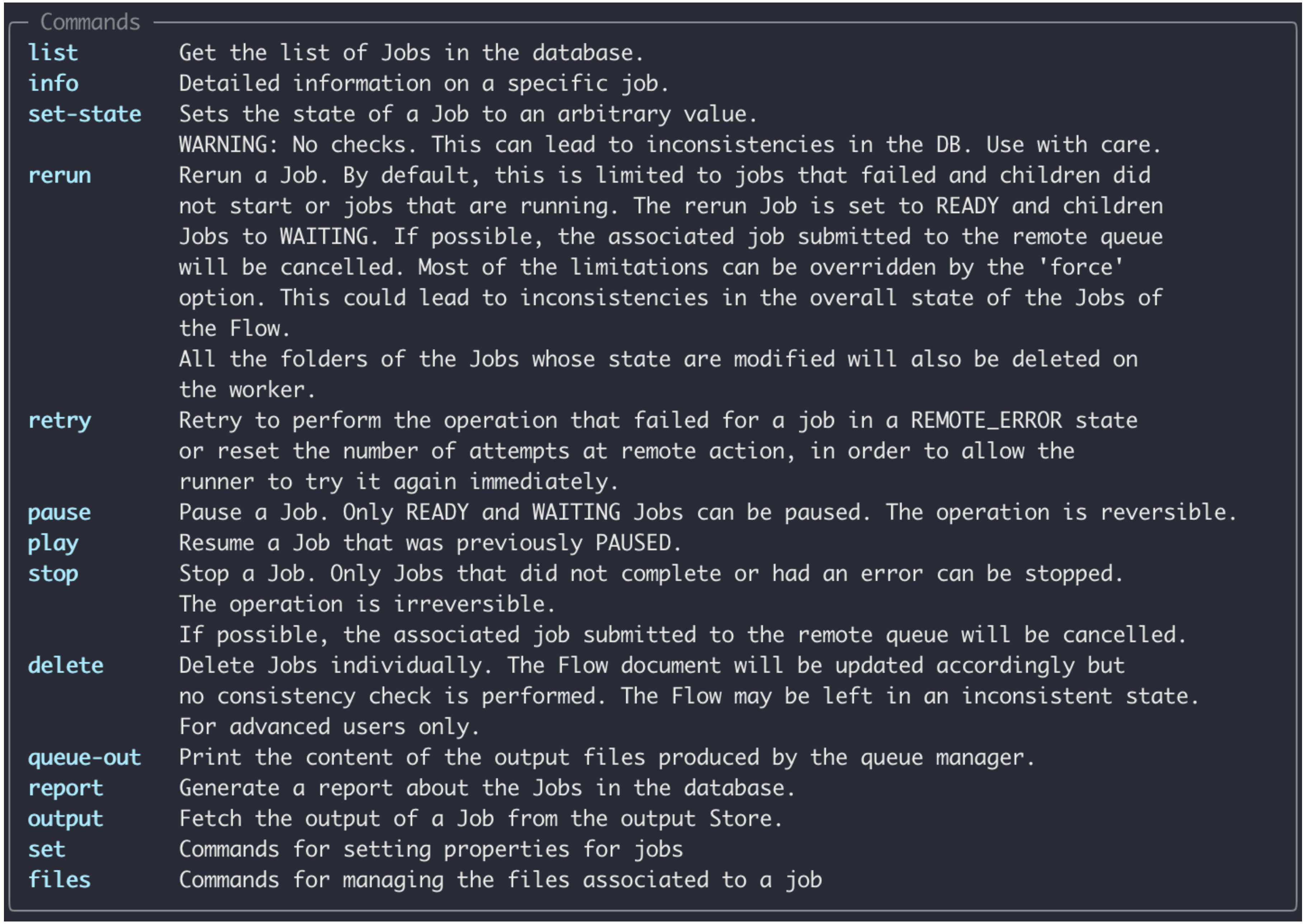


```
Ⓒjf ~ ❯ jf runner shutdown
The selected project is tutorial from config file /Users/guido/.jfremote/tutorial.yaml
```

CLI -JOB

job: query and control the Jobs in the queue DB

- List Jobs
  - Several filtering options
    - State, ids, names,…
  - -v verbosity option
- Detailed information
- Act on jobs
  - Rerun/retry
  - Set properties
- Report

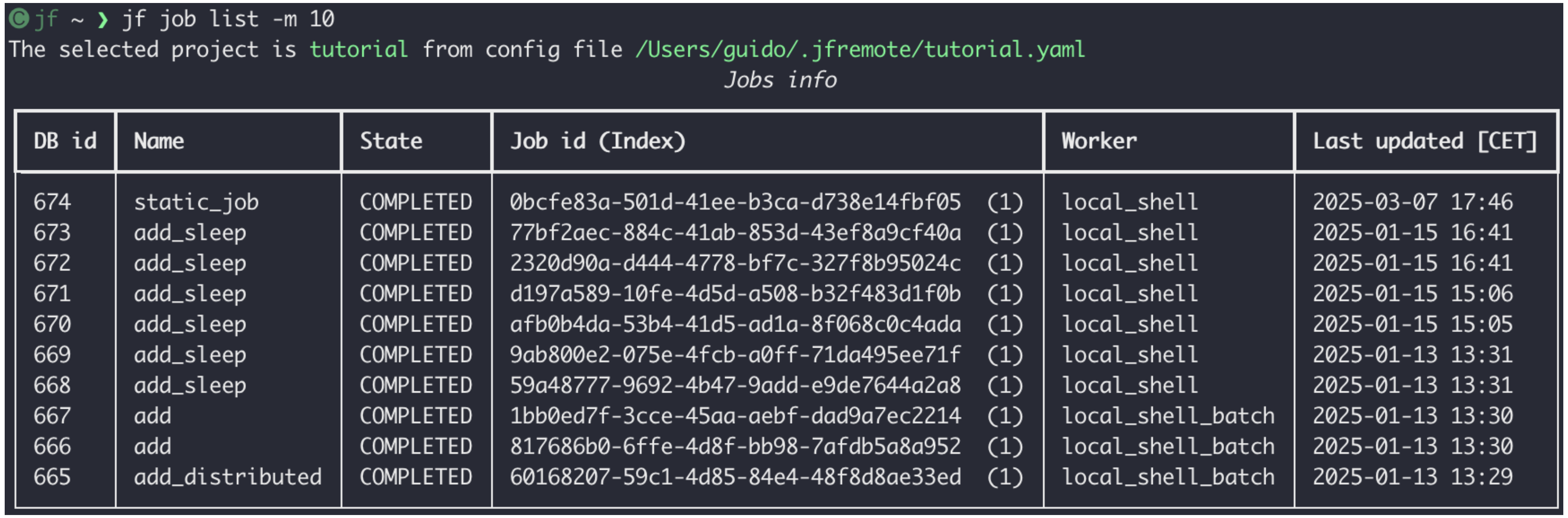


```
─ Commands ─────────────────────────────────────────────────────────────────
list        Get the list of Jobs in the database.
info        Detailed information on a specific job.
set-state   Sets the state of a Job to an arbitrary value.
            WARNING: No checks. This can lead to inconsistencies in the DB. Use with care.
rerun       Rerun a Job. By default, this is limited to jobs that failed and children did
            not start or jobs that are running. The rerun Job is set to READY and children
            Jobs to WAITING. If possible, the associated job submitted to the remote queue
            will be cancelled. Most of the limitations can be overridden by the 'force'
            option. This could lead to inconsistencies in the overall state of the Jobs of
            the Flow.
            All the folders of the Jobs whose state are modified will also be deleted on
            the worker.
retry       Retry to perform the operation that failed for a job in a REMOTE_ERROR state
            or reset the number of attempts at remote action, in order to allow the
            runner to try it again immediately.
pause       Pause a Job. Only READY and WAITING Jobs can be paused. The operation is reversible.
play        Resume a Job that was previously PAUSED.
stop        Stop a Job. Only Jobs that did not complete or had an error can be stopped.
            The operation is irreversible.
            If possible, the associated job submitted to the remote queue will be cancelled.
delete      Delete Jobs individually. The Flow document will be updated accordingly but
            no consistency check is performed. The Flow may be left in an inconsistent state.
            For advanced users only.
queue-out   Print the content of the output files produced by the queue manager.
report      Generate a report about the Jobs in the database.
output      Fetch the output of a Job from the output Store.
set         Commands for setting properties for jobs
files       Commands for managing the files associated to a job
```

CLI - JOB

job: query and control the Jobs in the queue DB

- List Jobs
  - Several filtering options
    - State, ids, names,…
  - -v verbosity option
- Detailed information
- Act on jobs
  - Rerun/retry
  - Set properties
- Report

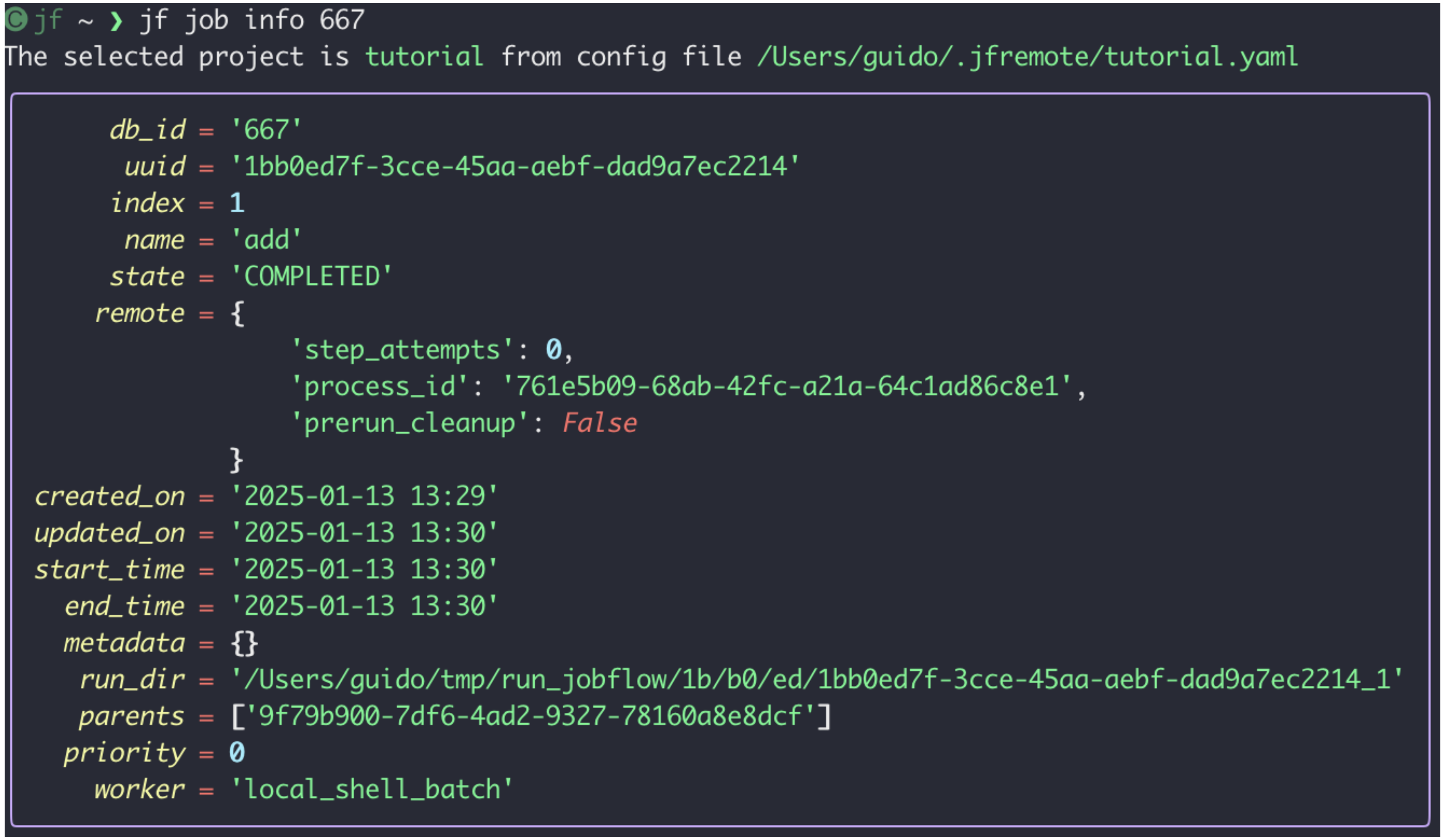


```
jf ~ ❯ jf job list -m 10
The selected project is tutorial from config file /Users/guido/.jfremote/tutorial.yaml
                                              Jobs info
┌────────┬─────────────────┬───────────┬──────────────────────────────────────────────┬──────────────────┬────────────────────┐
│ DB id  │ Name            │ State     │ Job id (Index)                               │ Worker           │ Last updated [CET] │
├────────┼─────────────────┼───────────┼──────────────────────────────────────────────┼──────────────────┼────────────────────┤
│ 674    │ static_job      │ COMPLETED │ 0bcfe83a-501d-41ee-b3ca-d738e14fbf05  (1)    │ local_shell      │ 2025-03-07 17:46   │
│ 673    │ add_sleep       │ COMPLETED │ 77bf2aec-884c-41ab-853d-43ef8a9cf40a  (1)    │ local_shell      │ 2025-01-15 16:41   │
│ 672    │ add_sleep       │ COMPLETED │ 2320d90a-d444-4778-bf7c-327f8b95024c  (1)    │ local_shell      │ 2025-01-15 16:41   │
│ 671    │ add_sleep       │ COMPLETED │ d197a589-10fe-4d5d-a508-b32f483d1f0b  (1)    │ local_shell      │ 2025-01-15 15:06   │
│ 670    │ add_sleep       │ COMPLETED │ afb0b4da-53b4-41d5-ad1a-8f068c0c4ada  (1)    │ local_shell      │ 2025-01-15 15:05   │
│ 669    │ add_sleep       │ COMPLETED │ 9ab800e2-075e-4fcb-a0ff-71da495ee71f  (1)    │ local_shell      │ 2025-01-13 13:31   │
│ 668    │ add_sleep       │ COMPLETED │ 59a48777-9692-4b47-9add-e9de7644a2a8  (1)    │ local_shell      │ 2025-01-13 13:31   │
│ 667    │ add             │ COMPLETED │ 1bb0ed7f-3cce-45aa-aebf-dad9a7ec2214  (1)    │ local_shell_batch│ 2025-01-13 13:30   │
│ 666    │ add             │ COMPLETED │ 817686b0-6ffe-4d8f-bb98-7afdb5a8a952  (1)    │ local_shell_batch│ 2025-01-13 13:30   │
│ 665    │ add_distributed │ COMPLETED │ 60168207-59c1-4d85-84e4-48f8d8ae33ed  (1)    │ local_shell_batch│ 2025-01-13 13:29   │
└────────┴─────────────────┴───────────┴──────────────────────────────────────────────┴──────────────────┴────────────────────┘
```

CLI - JOB

job: query and control the Jobs in the queue DB

- List Jobs
  - Several filtering options
    - State, ids, names,…
  - -v verbosity option
- Detailed information
- Act on jobs
  - Rerun/retry
  - Set properties
- Report

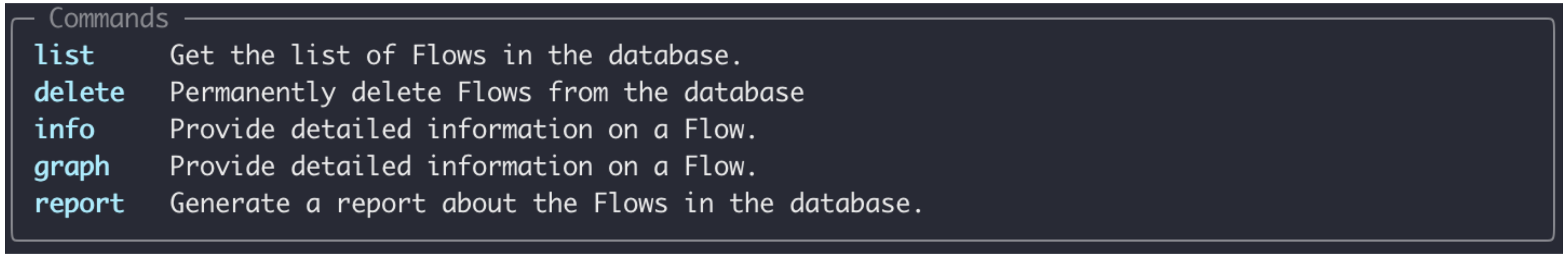


```
jf ~ ❯ jf job info 667
The selected project is tutorial from config file /Users/guido/.jfremote/tutorial.yaml

        db_id = '667'
         uuid = '1bb0ed7f-3cce-45aa-aebf-dad9a7ec2214'
        index = 1
         name = 'add'
        state = 'COMPLETED'
       remote = {
                    'step_attempts': 0,
                    'process_id': '761e5b09-68ab-42fc-a21a-64c1ad86c8e1',
                    'prerun_cleanup': False
                }
   created_on = '2025-01-13 13:29'
   updated_on = '2025-01-13 13:30'
   start_time = '2025-01-13 13:30'
     end_time = '2025-01-13 13:30'
     metadata = {}
      run_dir = '/Users/guido/tmp/run_jobflow/1b/b0/ed/1bb0ed7f-3cce-45aa-aebf-dad9a7ec2214_1'
      parents = ['9f79b900-7df6-4ad2-9327-78160a8e8dcf']
     priority = 0
       worker = 'local_shell_batch'
```

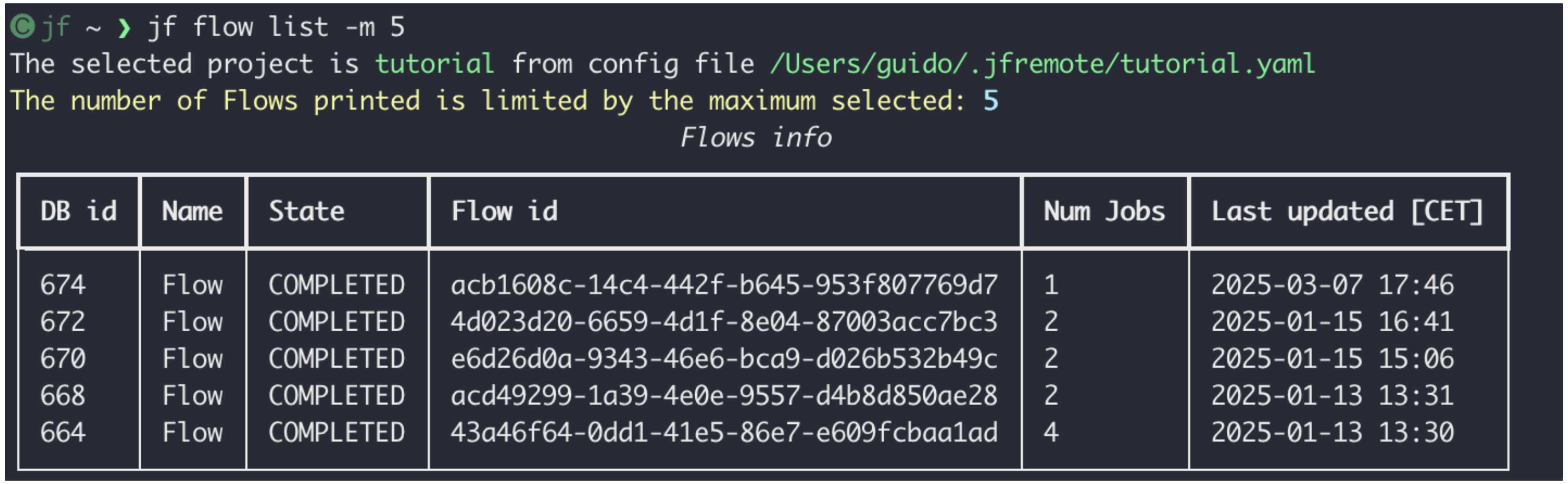
CLI - FLOW

flow: query and control the Flows in the queue DB

- List Flows
  - Several filtering options
    - State, ids, names,…
  - -v verbosity option
- Detailed information
- Delete Flows
- Report
- Graph



```
─ Commands ──────────────────────────────────────────────
list      Get the list of Flows in the database.
delete    Permanently delete Flows from the database
info      Provide detailed information on a Flow.
graph     Provide detailed information on a Flow.
report    Generate a report about the Flows in the database.
```

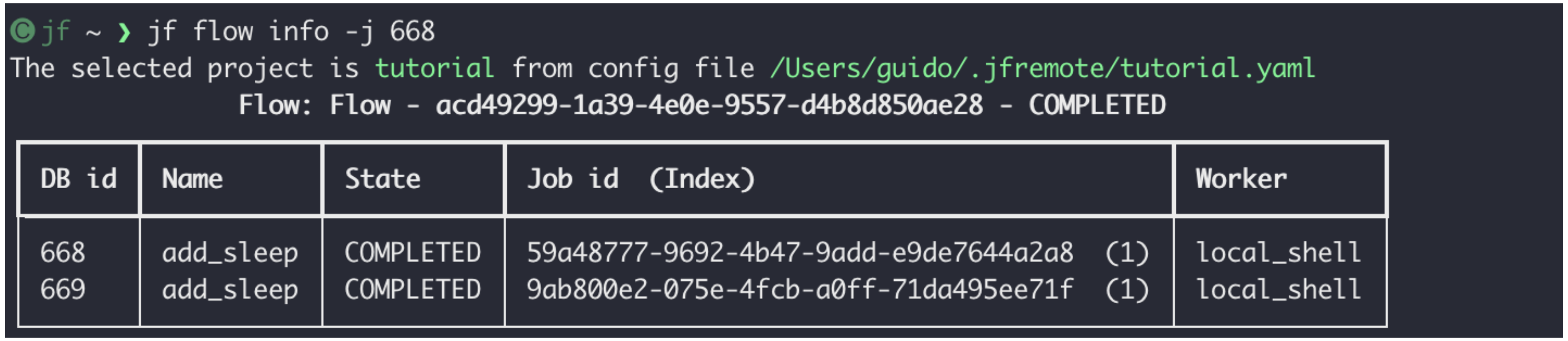


```
jf ~ ❯ jf flow list -m 5
The selected project is tutorial from config file /Users/guido/.jfremote/tutorial.yaml
The number of Flows printed is limited by the maximum selected: 5
                                   Flows info
┌───────┬──────┬───────────┬──────────────────────────────────────┬──────────┬────────────────────┐
│ DB id │ Name │ State     │ Flow id                              │ Num Jobs │ Last updated [CET] │
├───────┼──────┼───────────┼──────────────────────────────────────┼──────────┼────────────────────┤
│ 674   │ Flow │ COMPLETED │ acb1608c-14c4-442f-b645-953f807769d7 │ 1        │ 2025-03-07 17:46   │
│ 672   │ Flow │ COMPLETED │ 4d023d20-6659-4d1f-8e04-87003acc7bc3 │ 2        │ 2025-01-15 16:41   │
│ 670   │ Flow │ COMPLETED │ e6d26d0a-9343-46e6-bca9-d026b532b49c │ 2        │ 2025-01-15 15:06   │
│ 668   │ Flow │ COMPLETED │ acd49299-1a39-4e0e-9557-d4b8d850ae28 │ 2        │ 2025-01-13 13:31   │
│ 664   │ Flow │ COMPLETED │ 43a46f64-0dd1-41e5-86e7-e609fcbaa1ad │ 4        │ 2025-01-13 13:30   │
└───────┴──────┴───────────┴──────────────────────────────────────┴──────────┴────────────────────┘
```

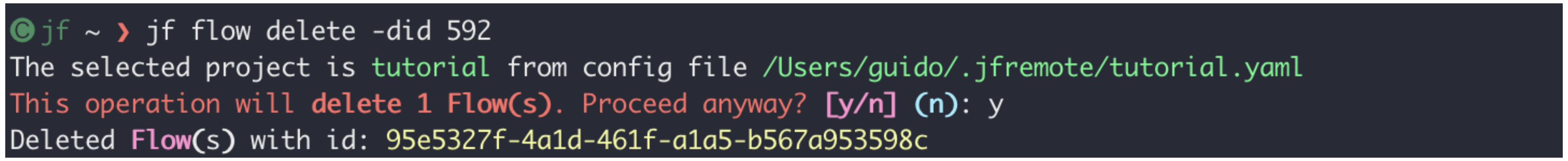
CLI - FLOW

flow: query and control the Flows in the queue DB

- List Flows
  - Several filtering options
    - State, ids, names,…
  - -v verbosity option
- Detailed information
- Delete Flows
- Report
- Graph



```
jf ~ ❯ jf flow info -j 668
The selected project is tutorial from config file /Users/guido/.jfremote/tutorial.yaml
               Flow: Flow - acd49299-1a39-4e0e-9557-d4b8d850ae28 - COMPLETED
┌───────┬──────────┬───────────┬──────────────────────────────────────────────┬─────────────┐
│ DB id │ Name     │ State     │ Job id  (Index)                              │ Worker      │
├───────┼──────────┼───────────┼──────────────────────────────────────────────┼─────────────┤
│ 668   │ add_sleep│ COMPLETED │ 59a48777-9692-4b47-9add-e9de7644a2a8  (1)    │ local_shell │
│ 669   │ add_sleep│ COMPLETED │ 9ab800e2-075e-4fcb-a0ff-71da495ee71f  (1)    │ local_shell │
└───────┴──────────┴───────────┴──────────────────────────────────────────────┴─────────────┘
```

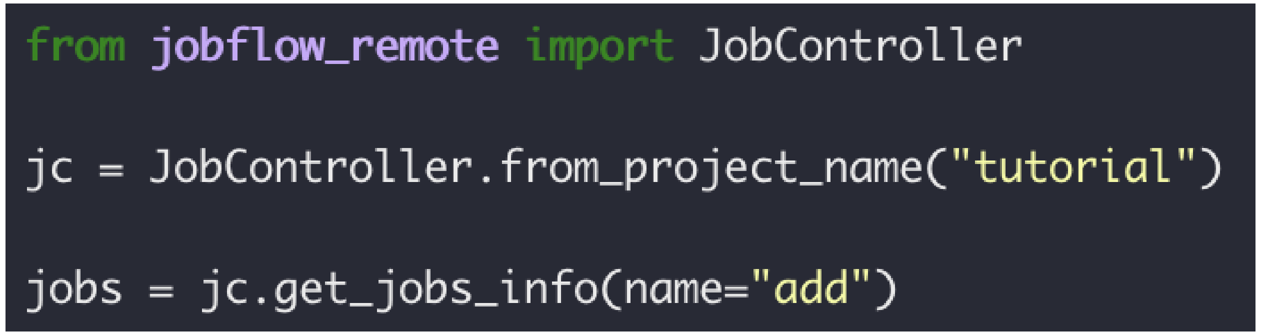


```
jf ~ ❯ jf flow delete -did 592
The selected project is tutorial from config file /Users/guido/.jfremote/tutorial.yaml
This operation will delete 1 Flow(s). Proceed anyway? [y/n] (n): y
Deleted Flow(s) with id: 95e5327f-4a1d-461f-a1a5-b567a953598c
```

PYTHON API

Most of the functionalities exposed in the CLI are matched by objects and functions to perform the same actions from python.

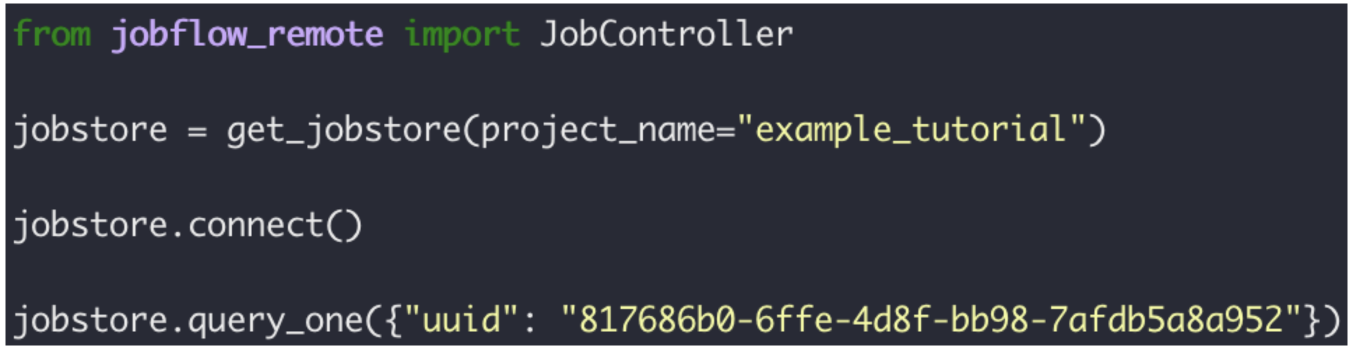
- JobController: interactions with the queue DB
- DaemonManager: manage the Runner daemonized process
- ConfigManager: manage projects and their content



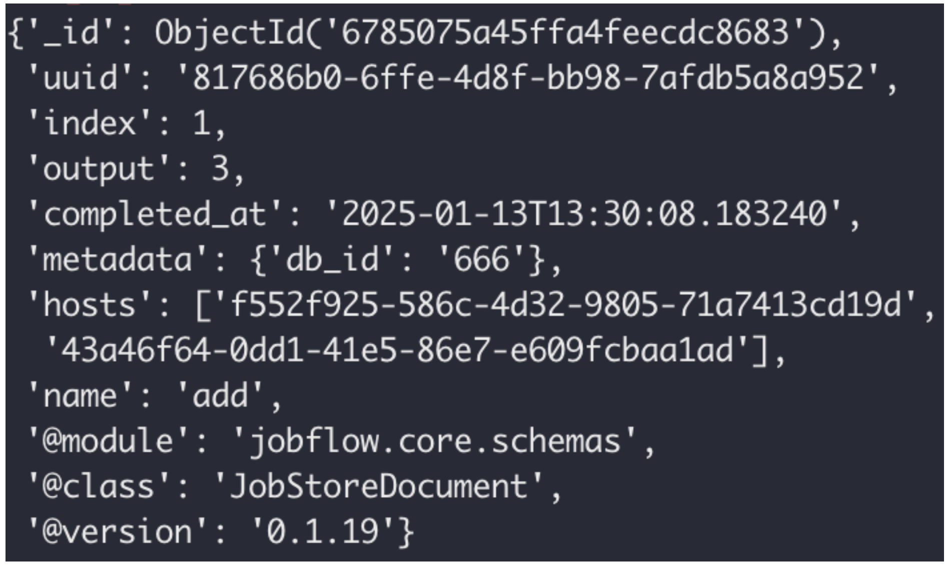
```python
from jobflow_remote import JobController

jc = JobController.from_project_name("tutorial")

jobs = jc.get_jobs_info(name="add")
```

ACCESS TO OUTPUT RESULTS

- Based on the standard Jobflow's JobStore
  - Same content and approach
- Access the correct JobStore based on the project
- get_jobstore from jobflow-remote



```python
from jobflow_remote import JobController

jobstore = get_jobstore(project_name="example_tutorial")

jobstore.connect()

jobstore.query_one({"uuid": "817686b0-6ffe-4d8f-bb98-7afdb5a8a952"})
```

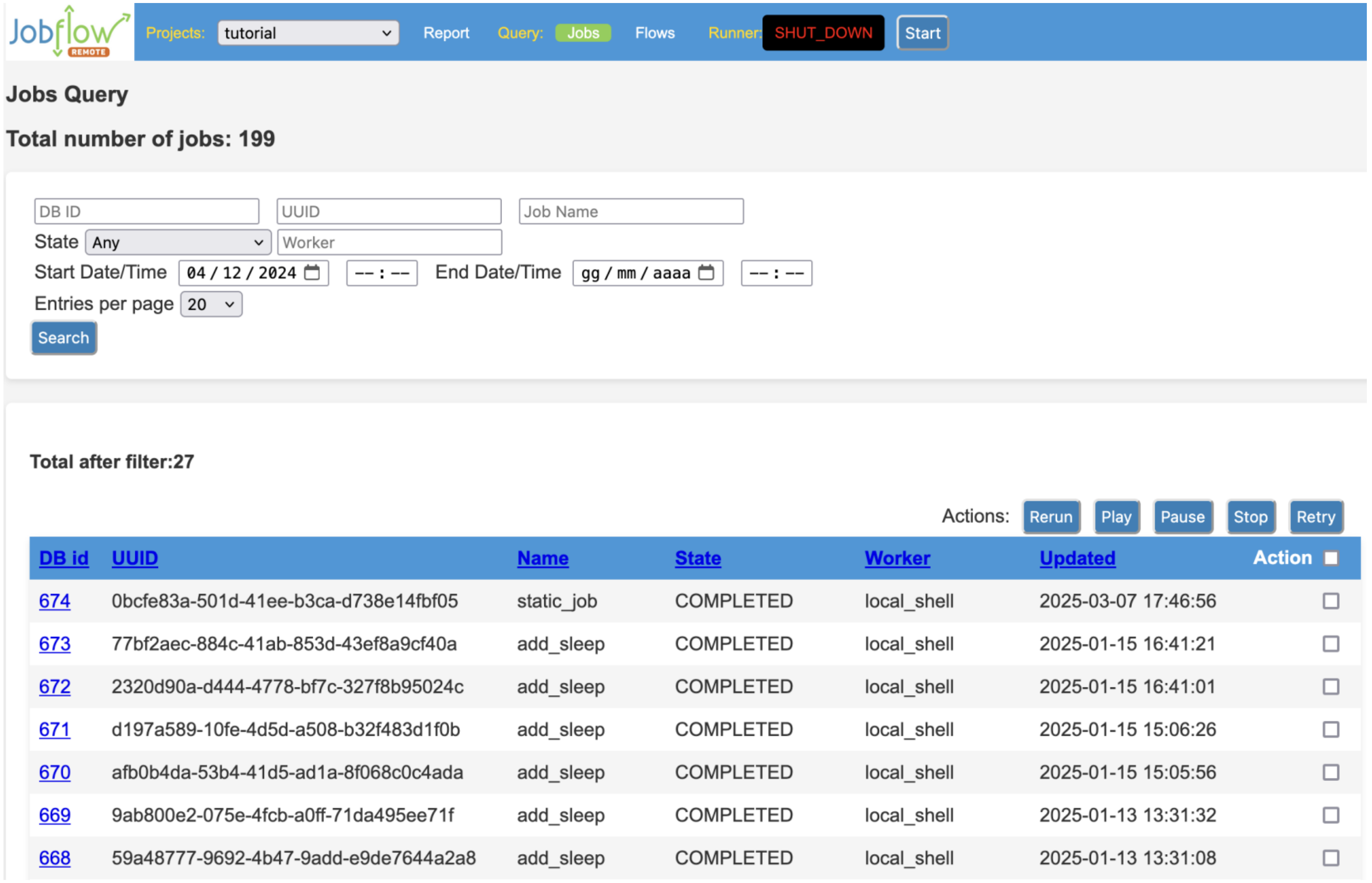


```
{'_id': ObjectId('6785075a45ffa4feecdc8683'),
 'uuid': '817686b0-6ffe-4d8f-bb98-7afdb5a8a952',
 'index': 1,
 'output': 3,
 'completed_at': '2025-01-13T13:30:08.183240',
 'metadata': {'db_id': '666'},
 'hosts': ['f552f925-586c-4d32-9805-71a7413cd19d',
  '43a46f64-0dd1-41e5-86e7-e609fcbaa1ad'],
 'name': 'add',
 '@module': 'jobflow.core.schemas',
 '@class': 'JobStoreDocument',
 '@version': '0.1.19'}
```

GUI

Experimental GUI based on FastHTML

- Runner
- Jobs/Flows
    - List
    - Info
    - Control
    - Delete
- Report



Jobs Query

Total number of jobs: 199

DB ID	UUID	Job Name
State: Any	Worker	
Start Date/Time: 04 / 12 / 2024 -- : --	End Date/Time: gg / mm / aaaa -- : --	
Entries per page: 20		

Search

Total after filter:27

Actions: Rerun Play Pause Stop Retry

DB id	UUID	Name	State	Worker	Updated	Action ☐
674	0bcfe83a-501d-41ee-b3ca-d738e14fbf05	static_job	COMPLETED	local_shell	2025-03-07 17:46:56	☐
673	77bf2aec-884c-41ab-853d-43ef8a9cf40a	add_sleep	COMPLETED	local_shell	2025-01-15 16:41:21	☐
672	2320d90a-d444-4778-bf7c-327f8b95024c	add_sleep	COMPLETED	local_shell	2025-01-15 16:41:01	☐
671	d197a589-10fe-4d5d-a508-b32f483d1f0b	add_sleep	COMPLETED	local_shell	2025-01-15 15:06:26	☐
670	afb0b4da-53b4-41d5-ad1a-8f068c0c4ada	add_sleep	COMPLETED	local_shell	2025-01-15 15:05:56	☐
669	9ab800e2-075e-4fcb-a0ff-71da495ee71f	add_sleep	COMPLETED	local_shell	2025-01-13 13:31:32	☐
668	59a48777-9692-4b47-9add-e9de7644a2a8	add_sleep	COMPLETED	local_shell	2025-01-13 13:31:08	☐

DEALING WITH ERRORS

ERRORS

Two categories of errors:

- Job raises an exception during execution

  FAILED

  - Bad inputs
  - External code does not complete successfully
  - …

- The runner fails while performing one of the actions

  REMOTE_ERROR

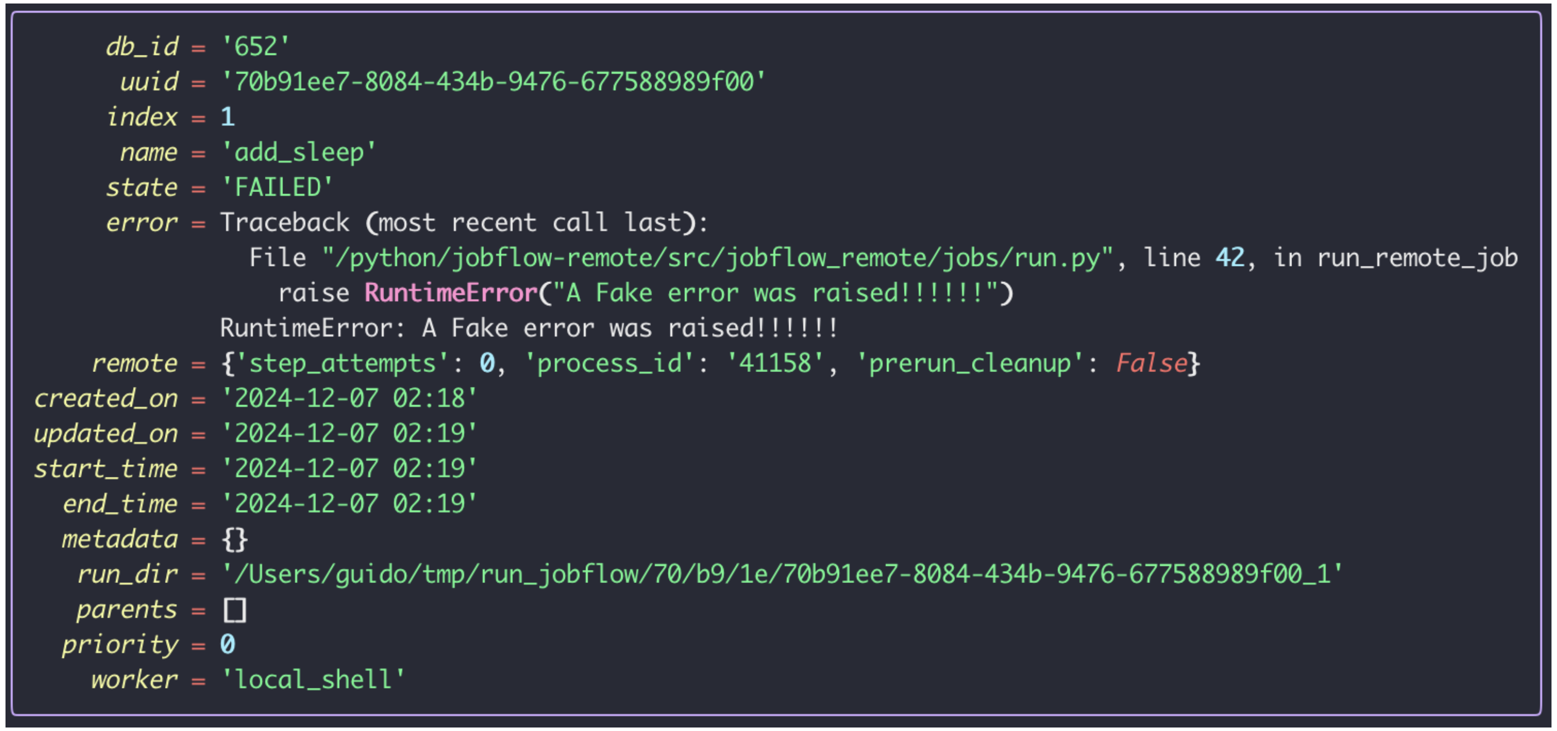
  - Connection issues (worker, JobStore)
  - HPC queueing system errors
  - …

FAILED - ERROR INFORMATION

FAILED

Where to look for information about errors?

- `jf job info <JOB_ID>`: “error” keyword
- Files on in the worker:
  - `run_dir`
  - Queueing system files
    - queue.out, queue.err
  - External code outputs



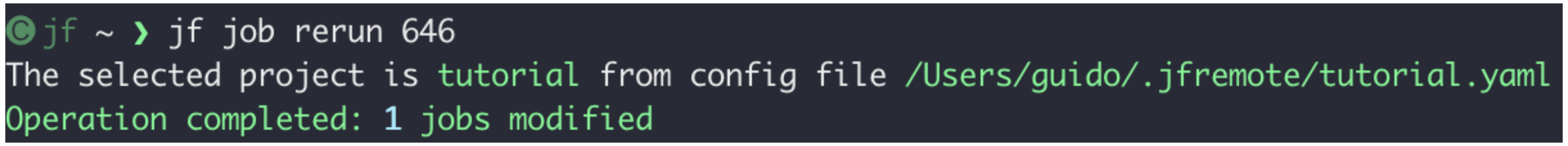
```
      db_id = '652'
       uuid = '70b91ee7-8084-434b-9476-677588989f00'
      index = 1
       name = 'add_sleep'
      state = 'FAILED'
      error = Traceback (most recent call last):
                File "/python/jobflow-remote/src/jobflow_remote/jobs/run.py", line 42, in run_remote_job
                  raise RuntimeError("A Fake error was raised!!!!!!")
              RuntimeError: A Fake error was raised!!!!!!
     remote = {'step_attempts': 0, 'process_id': '41158', 'prerun_cleanup': False}
 created_on = '2024-12-07 02:18'
 updated_on = '2024-12-07 02:19'
 start_time = '2024-12-07 02:19'
   end_time = '2024-12-07 02:19'
   metadata = {}
    run_dir = '/Users/guido/tmp/run_jobflow/70/b9/1e/70b91ee7-8084-434b-9476-677588989f00_1'
    parents = []
   priority = 0
     worker = 'local_shell'
```

FAILED - FIX

FAILED

No *general recipe* for fixing failures

- Temporary issue: rerun = Job back to READY state
- Wrong inputs:
  - Change inputs and rerun
  - Resubmit a new flow (delete the previous one)
- Bug in the code:
  - Fix and resubmit flow



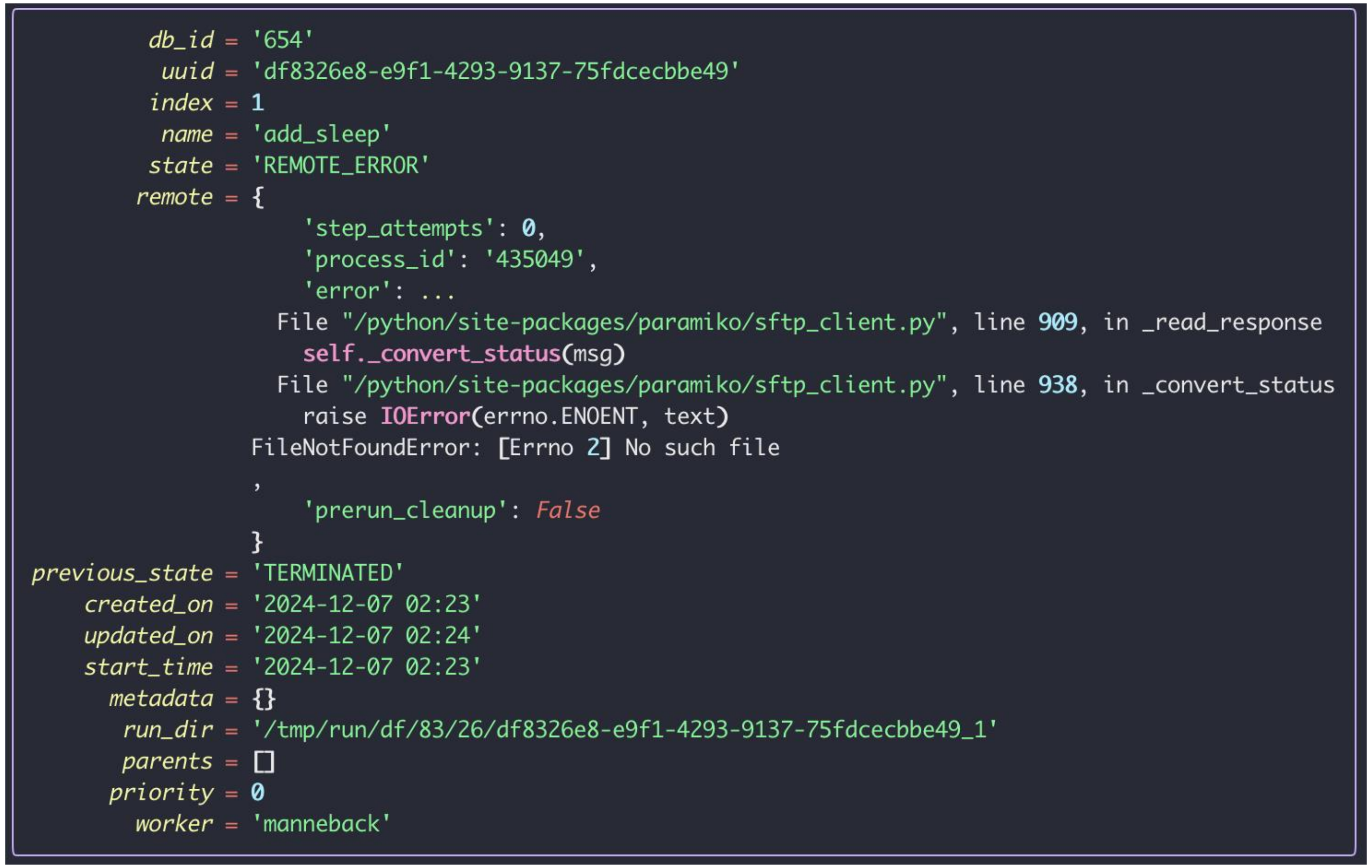
```
jf ~ ❯ jf job rerun 646
The selected project is tutorial from config file /Users/guido/.jfremote/tutorial.yaml
Operation completed: 1 jobs modified
```

REMOTE_ERROR - ERROR INFORMATION

REMOTE_ERROR

Where to look for information about errors?

- `jf job info <JOB_ID>`: “error.queue” keyword
- Files on in the worker:
  - `run_dir`
  - Queueing system files
    - queue.out, queue.err
  - Missing jfremote outputs
  - Runner logs
    - ~/.jfremote/PROJ_NAME/log



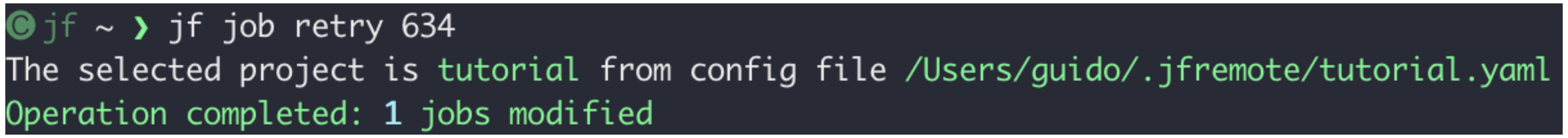
```
          db_id = '654'
           uuid = 'df8326e8-e9f1-4293-9137-75fdcecbbe49'
          index = 1
           name = 'add_sleep'
          state = 'REMOTE_ERROR'
         remote = {
                      'step_attempts': 0,
                      'process_id': '435049',
                      'error': ...
                    File "/python/site-packages/paramiko/sftp_client.py", line 909, in _read_response
                      self._convert_status(msg)
                    File "/python/site-packages/paramiko/sftp_client.py", line 938, in _convert_status
                      raise IOError(errno.ENOENT, text)
                  FileNotFoundError: [Errno 2] No such file
                  ,
                      'prerun_cleanup': False
                  }
 previous_state = 'TERMINATED'
     created_on = '2024-12-07 02:23'
     updated_on = '2024-12-07 02:24'
     start_time = '2024-12-07 02:23'
       metadata = {}
        run_dir = '/tmp/run/df/83/26/df8326e8-e9f1-4293-9137-75fdcecbbe49_1'
        parents = []
       priority = 0
         worker = 'manneback'
```

REMOTE_ERROR - FIX

REMOTE_ERROR

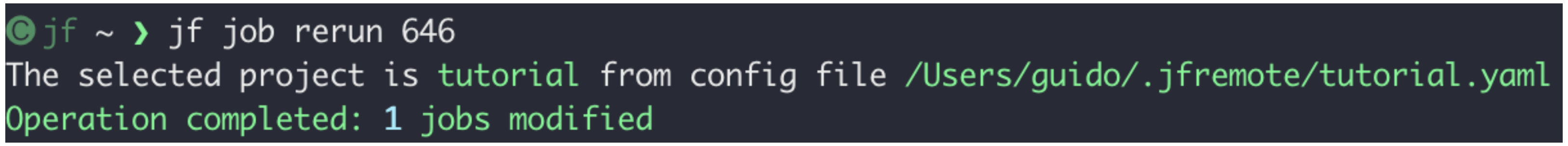
No *general recipe* for fixing failures

- Temporary issue: retry = try again the same remote action (e.g. job back to UPLOADED)



```
jf ~ ❯ jf job retry 634
The selected project is tutorial from config file /Users/guido/.jfremote/tutorial.yaml
Operation completed: 1 jobs modified
```

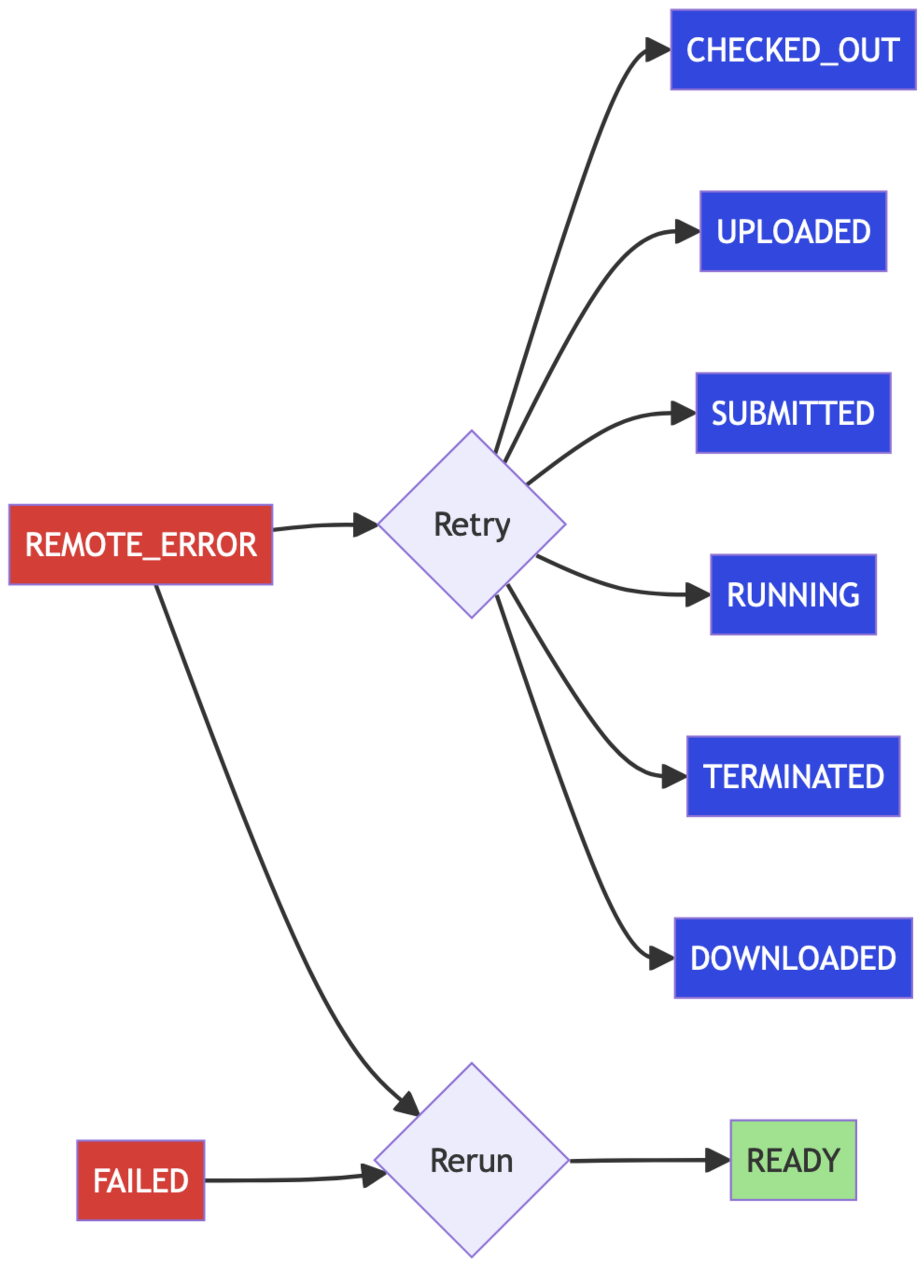
- Wrong resources:
  - Updates resources (CLI or python API) and retry
- Wrong connection configuration:
  - Fix config and retry
- …
- If problems from previous steps are involved: full rerun



```
jf ~ ❯ jf job rerun 646
The selected project is tutorial from config file /Users/guido/.jfremote/tutorial.yaml
Operation completed: 1 jobs modified
```

RERUN/RETRY SCHEMA

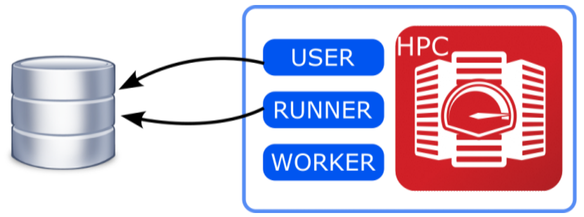
- rerun = Job back to READY state
- retry = try again the same remote action (e.g. job back to UPLOADED, TERMINATED, …)



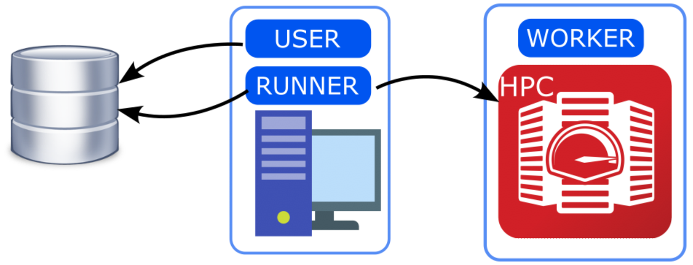
CONFIGURATION

POSSIBLE CONFIGURATIONS

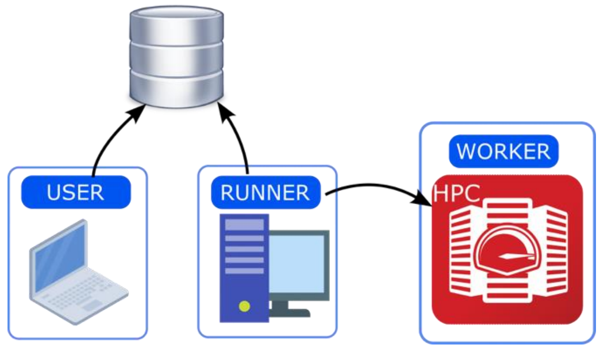
- All-in-one
  - Running completely on the cluster



- User-Workstation
  - A workstation hosting the daemon and used for user interactions



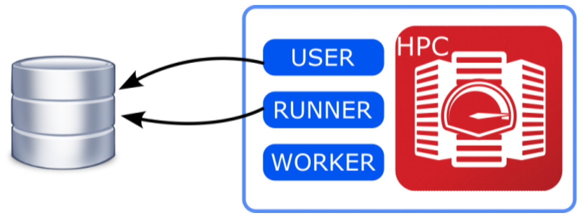
- Full split
  - Workstation for the daemon and separate system for user interaction



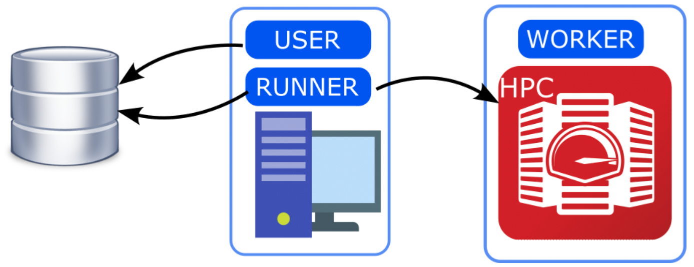
WARNING: The same python environment should be present on all the machines

POSSIBLE CONFIGURATIONS

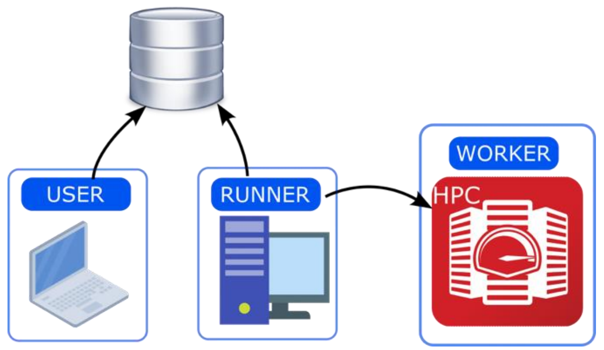
- All-in-one
  - Running completely on the cluster



- User-Workstation
  - A workstation hosting the daemon and used for user interactions



- Full split
  - Workstation for the daemon and separate system for user interaction



WARNING: The same python environment should be present on all the machines

PROJECTS

A project:

- The set of configurations defining DBs and workers
- Defined in a file (yaml, json, toml)
- Associated with a single JobStore and Queue
- Preferably bound to a single python environment (avoid incompatibilities)
- Has its own runner

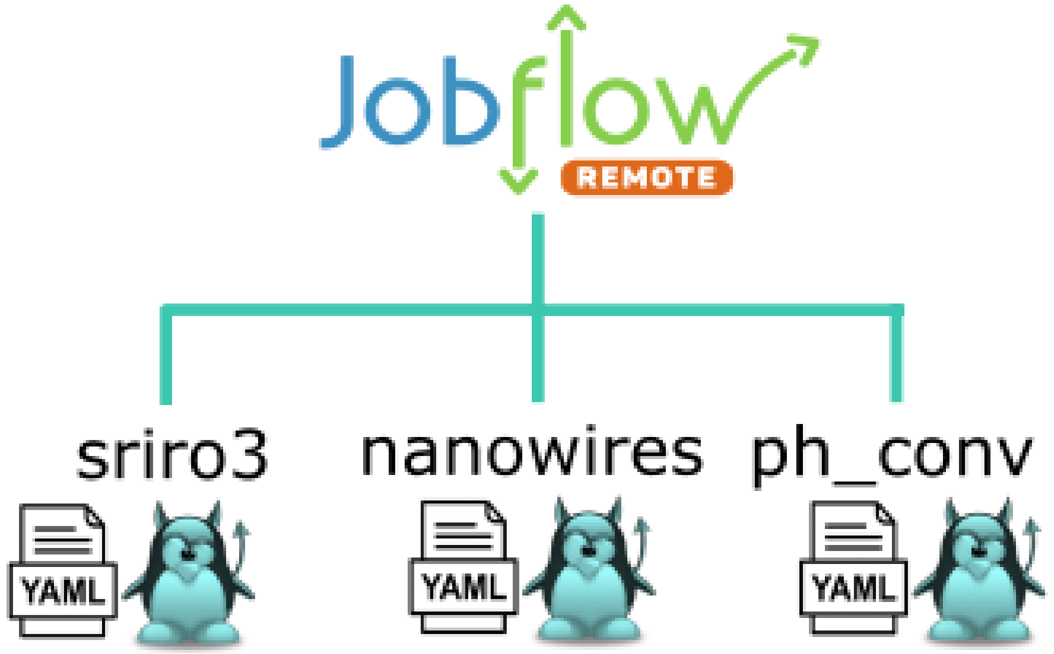
YAML

JSON

Queue store

JobStore

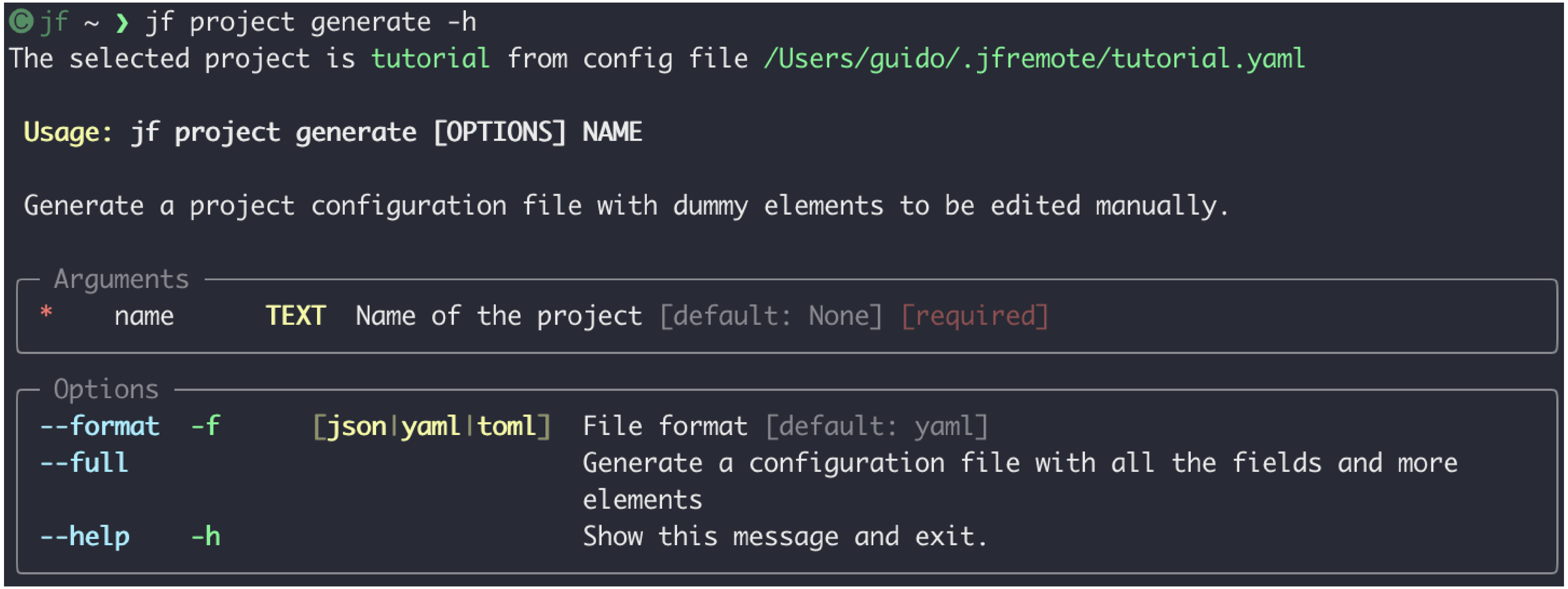
MULTIPLE PROJECTS



Why multiple project?

- Separate research project
- Separate results
- Run independently from other projects
- Different python packages

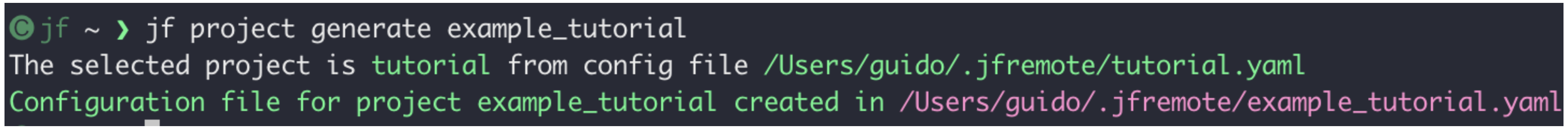
CREATE A PROJECT



```
 jf ~ ❯ jf project generate -h
The selected project is tutorial from config file /Users/guido/.jfremote/tutorial.yaml

 Usage: jf project generate [OPTIONS] NAME

 Generate a project configuration file with dummy elements to be edited manually.

╭─ Arguments ──────────────────────────────────────────────────────────────────────────────────────╮
│ *    name      TEXT  Name of the project [default: None] [required]                              │
╰──────────────────────────────────────────────────────────────────────────────────────────────────╯
╭─ Options ────────────────────────────────────────────────────────────────────────────────────────╮
│ --format  -f      [json|yaml|toml]  File format [default: yaml]                                  │
│ --full                              Generate a configuration file with all the fields and more   │
│                                     elements                                                     │
│ --help    -h                        Show this message and exit.                                  │
╰──────────────────────────────────────────────────────────────────────────────────────────────────╯
```

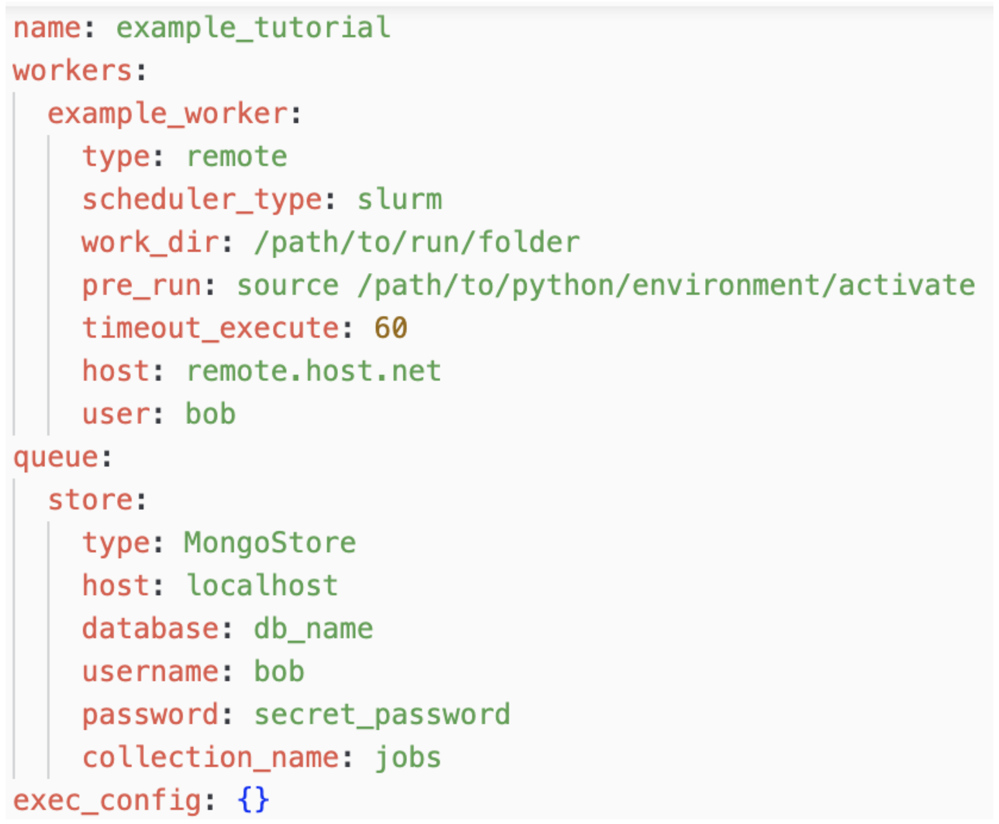
CREATE A PROJECT

Generate a minimal configuration file to fill in

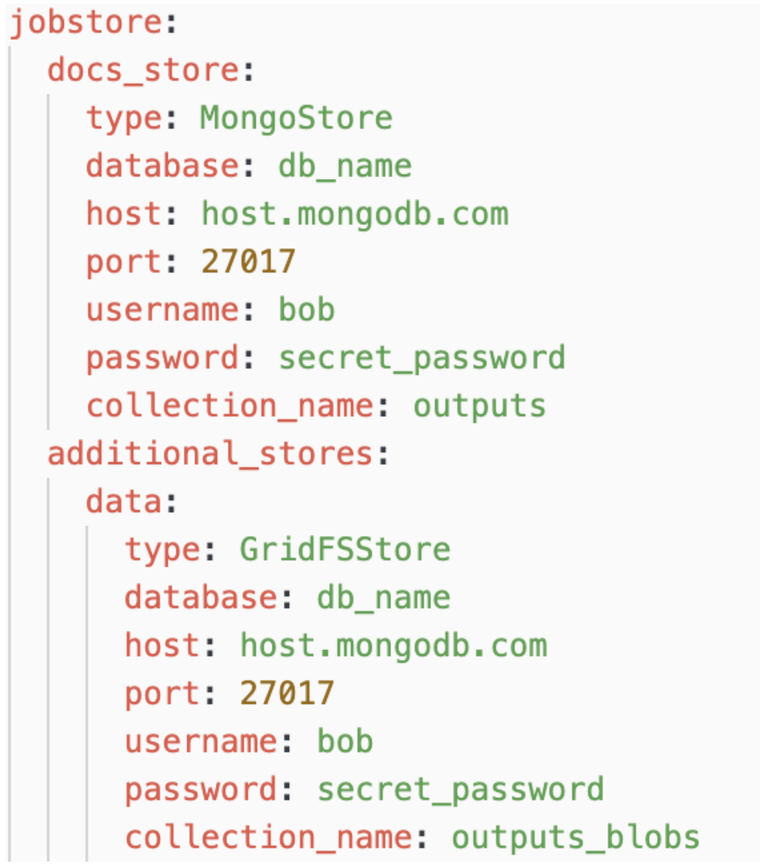


```
jf ~ ❯ jf project generate example_tutorial
The selected project is tutorial from config file /Users/guido/.jfremote/tutorial.yaml
Configuration file for project example_tutorial created in /Users/guido/.jfremote/example_tutorial.yaml
```

example_tutorial.yaml:



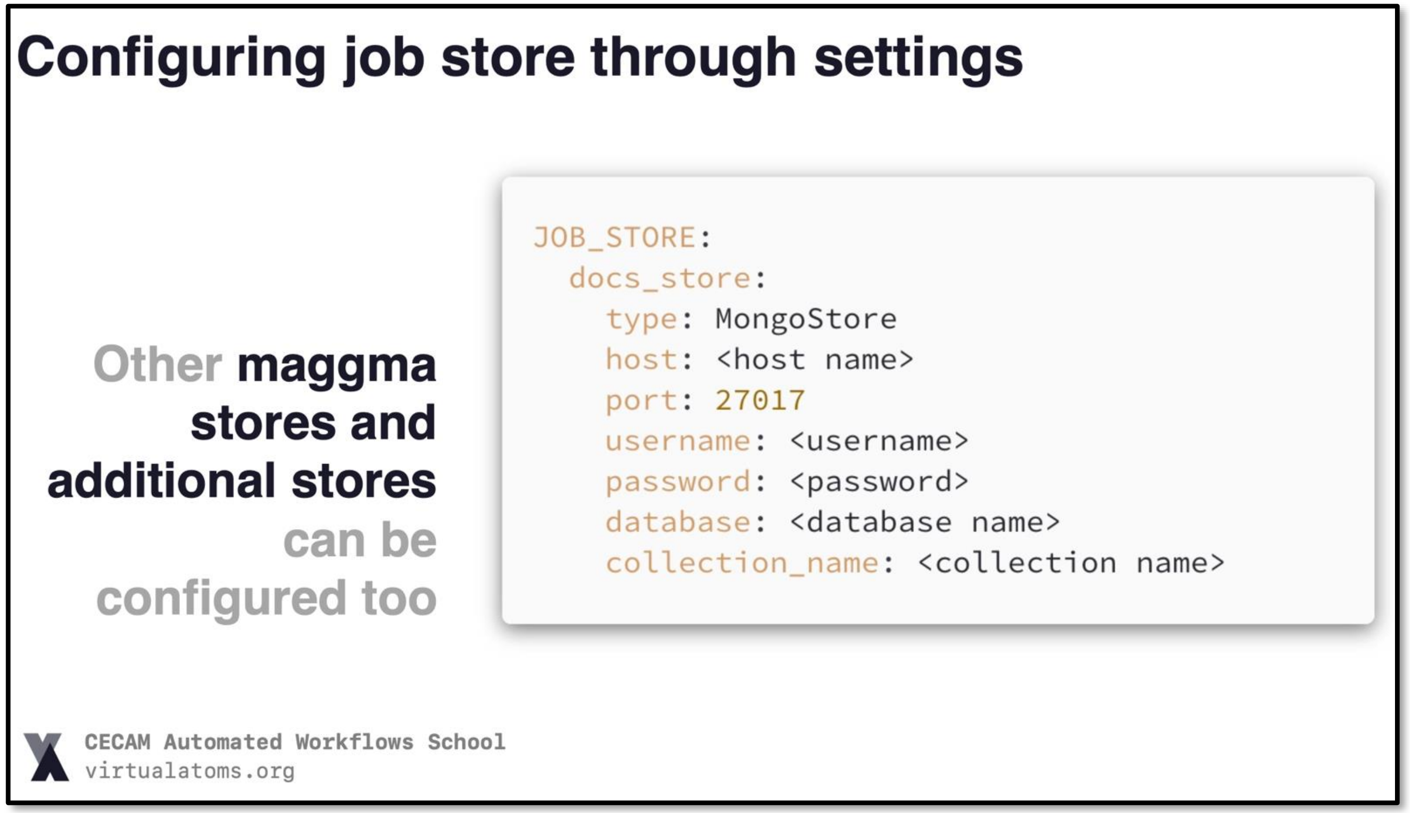
```yaml
name: example_tutorial
workers:
  example_worker:
    type: remote
    scheduler_type: slurm
    work_dir: /path/to/run/folder
    pre_run: source /path/to/python/environment/activate
    timeout_execute: 60
    host: remote.host.net
    user: bob
queue:
  store:
    type: MongoStore
    host: localhost
    database: db_name
    username: bob
    password: secret_password
    collection_name: jobs
exec_config: {}
```



```yaml
jobstore:
  docs_store:
    type: MongoStore
    database: db_name
    host: host.mongodb.com
    port: 27017
    username: bob
    password: secret_password
    collection_name: outputs
  additional_stores:
    data:
      type: GridFSStore
      database: db_name
      host: host.mongodb.com
      port: 27017
      username: bob
      password: secret_password
      collection_name: outputs_blobs
```

JOBSTORE

Same format as standard Jobflow, but not from jobflow.yaml



Configuring job store through settings

Other **maggma stores and additional stores** can be configured too

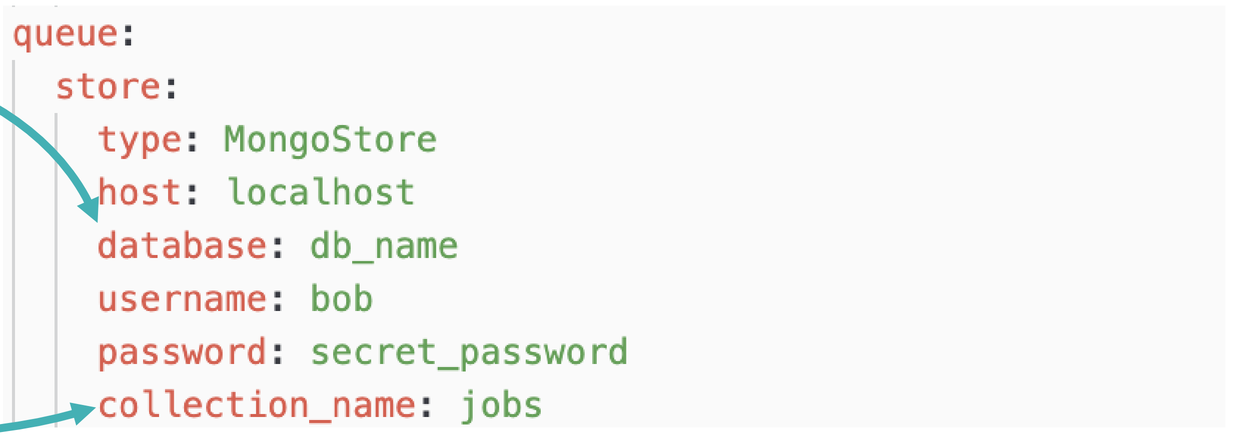
```
JOB_STORE:
  docs_store:
    type: MongoStore
    host: <host name>
    port: 27017
    username: <username>
    password: <password>
    database: <database name>
    collection_name: <collection name>
```

CECAM Automated Workflows School
virtualatoms.org

```yaml
jobstore:
  docs_store:
    type: MongoStore
    database: db_name
    host: host.mongodb.com
    port: 27017
    username: bob
    password: secret_password
    collection_name: outputs
  additional_stores:
    data:
      type: GridFSStore
      database: db_name
      host: host.mongodb.com
      port: 27017
      username: bob
      password: secret_password
      collection_name: outputs_blobs
```

QUEUE STORE

- Same format as standard jobflow for maggma store
- Must be a “mongo-like” Store with an underlying real MongoDB
- Can be the same database as JobStore, but different collection

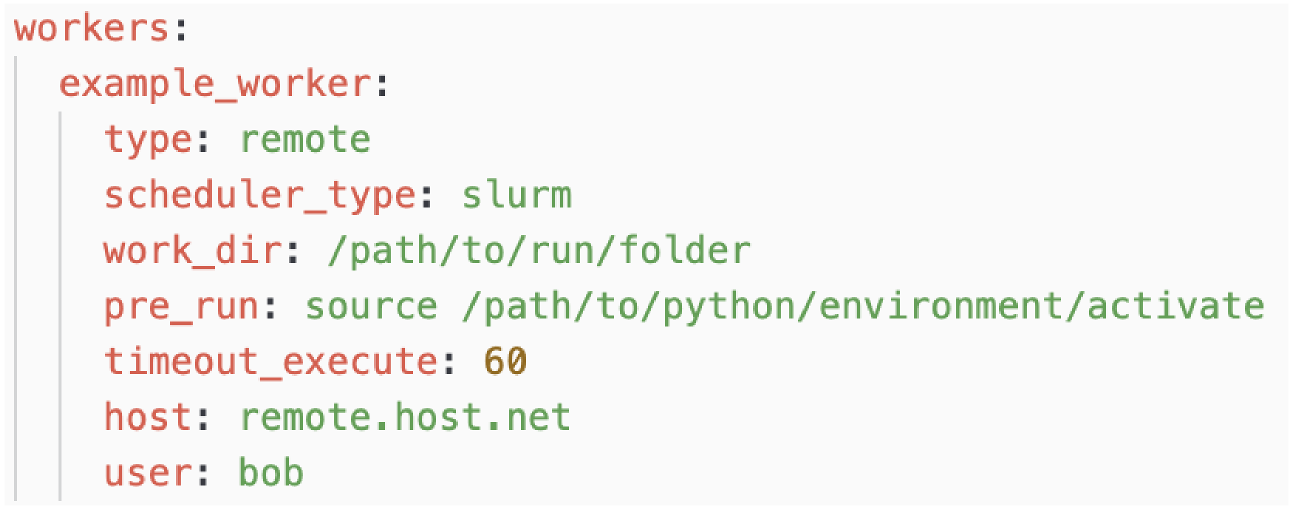


```yaml
queue:
  store:
    type: MongoStore
    host: localhost
    database: db_name
    username: bob
    password: secret_password
    collection_name: jobs
```

WORKERS

Define the workers executing the jobs

- type
  - remote: SSH connection
    - Provide connection details
  - local: same machine as the Runner
- scheduler_type
  - shell: executed as a script in the shell
  - slurm/pbs/…: queueing system
- work_dir: folder of execution of jobs
- pre_run: commands added to the submission script



```yaml
workers:
  example_worker:
    type: remote
    scheduler_type: slurm
    work_dir: /path/to/run/folder
    pre_run: source /path/to/python/environment/activate
    timeout_execute: 60
    host: remote.host.net
    user: bob
```

EXECUTION CONFIGURATION

A list of configuration options to be added to the submission script on the worker

Can set:

- Modules to be loaded
- Environmental variables
- Pre_run/post_run: commands before/after the job execution

Needs to be passed to the Job when submitting.

```
exec_config:
  vasp_6.4.3_cecam:
    modules:
    - gcc/11.3.0
    - openmpi/4.1.3
    - openblas/0.3.20
    - fftw/3.3.10
    export:
      PATH: /scratch/cecam.school/Atomate/vasp/vasp.6.4.3_gnu/bin:$PATH
      atomate2_VASP_CMD: '"srun vasp_std"'
      atomate2_VASP_GAMMA_CMD: '"srun vasp_gam"'
      atomate2_VASP_NCL_CMD: '"srun vasp_ncl"'
      atomate2_VASP_STORE_ADDITIONAL_JSON: 'False'
      VASP_PSP_DIR: /scratch/cecam.school/Atomate/vasp/potcar_pmg
      LD_LIBRARY_PATH: /scratch/cecam.school/Atomate/libs/scalapack-2.2.2:$LD_LIBRARY_PATH
    pre_run:
    post_run:
```

SELECTING A PROJECT

- If only one project no need to specify it
- Python API: `project` argument

```
submit_flow(flow, project="example_tutorial")
```

- CLI
  - **-p** argument to `jf`. Applied to the single command

```
jf -p example_tutorial job list
```

❗ Not `jf job list -p example_tutorial`

  - Export `jfremote_project` environment variable. Applied to all commands.

```
export jfremote_project=example_tutorial
jf job list
```

TUNING JOB EXECUTION

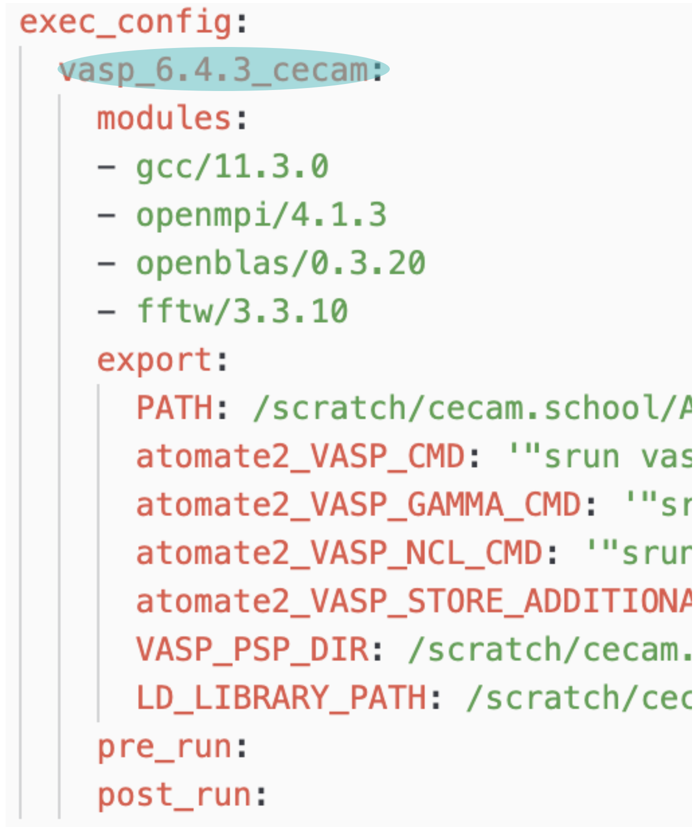
HOW TO TUNE THE EXECUTION OF THE JOB

- Execution configuration
  - See previous slides
  - Can be set:
    - at submission level
    - using a powerup

- Resources (e.g. slurm-related)
  - Worker name
  - Number of cores, memory, partition…
  - Can be set:
    - at worker level
    - at submission level
    - using a powerup

```yaml
exec_config:
  vasp_6.4.3_cecam:
    modules:
    - gcc/11.3.0
    - openmpi/4.1.3
    - openblas/0.3.20
    - fftw/3.3.10
    export:
      PATH: /scratch/cecam.school/Atomate/vasp/vasp.6.4.3_gnu/bin:$PATH
      atomate2_VASP_CMD: '"srun vasp_std"'
      atomate2_VASP_GAMMA_CMD: '"srun vasp_gam"'
      atomate2_VASP_NCL_CMD: '"srun vasp_ncl"'
      atomate2_VASP_STORE_ADDITIONAL_JSON: 'False'
      VASP_PSP_DIR: /scratch/cecam.school/Atomate/vasp/potcar_pmg
      LD_LIBRARY_PATH: /scratch/cecam.school/Atomate/libs/scalapack-2.2.2:$LD_LIBRARY_PATH
    pre_run:
    post_run:
```

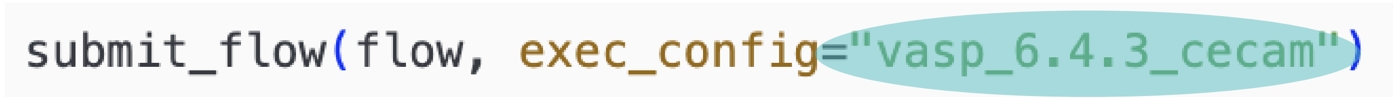
EXECUTION CONFIGS



```
exec_config:
  vasp_6.4.3_cecam:
    modules:
    - gcc/11.3.0
    - openmpi/4.1.3
    - openblas/0.3.20
    - fftw/3.3.10
    export:
      PATH: /scratch/cecam.school/A
      atomate2_VASP_CMD: '"srun vas
      atomate2_VASP_GAMMA_CMD: '"sr
      atomate2_VASP_NCL_CMD: '"srur
      atomate2_VASP_STORE_ADDITIONA
      VASP_PSP_DIR: /scratch/cecam.
      LD_LIBRARY_PATH: /scratch/cec
    pre_run:
    post_run:
```

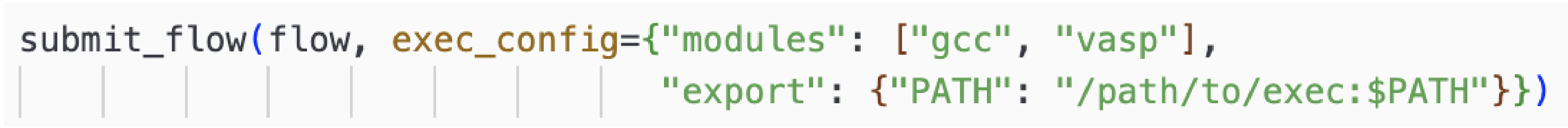
At submission

Use the name of one defined in the configuration



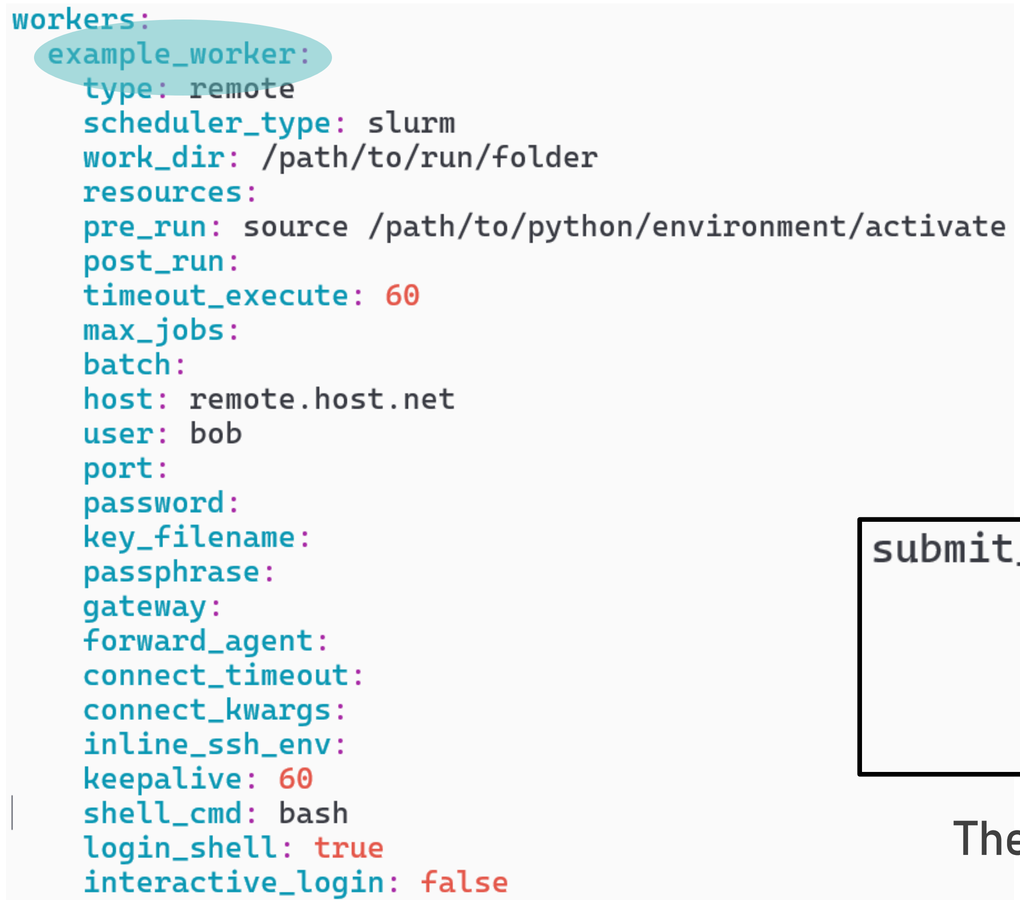
```
submit_flow(flow, exec_config="vasp_6.4.3_cecam")
```

Or you can directly pass an exec_config dictionary:

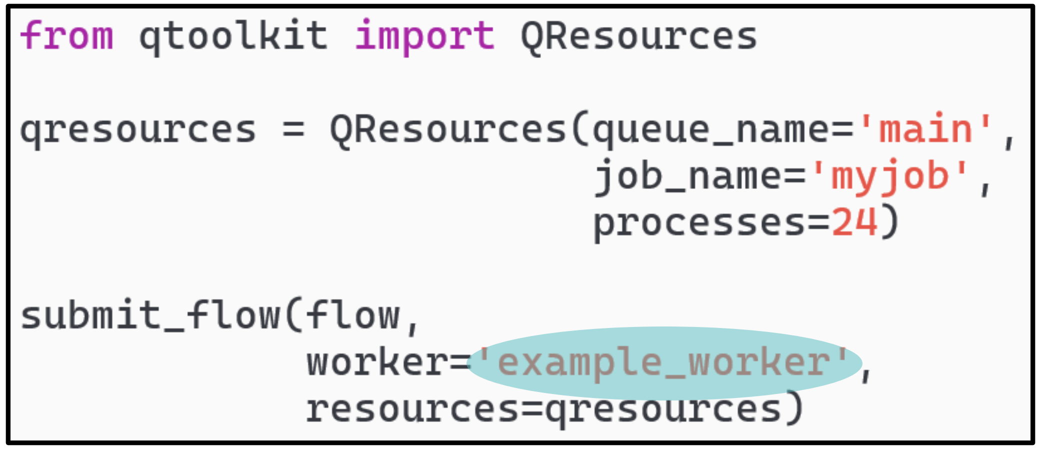


```
submit_flow(flow, exec_config={"modules": ["gcc", "vasp"],
                               "export": {"PATH": "/path/to/exec:$PATH"}})
```

SETTING RESOURCES AT SUBMISSION LEVEL

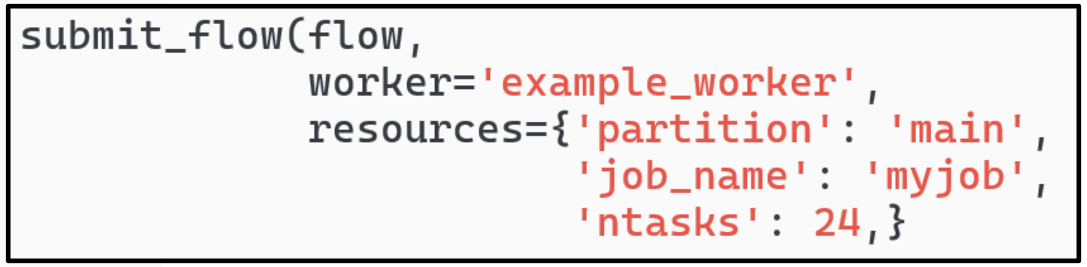


```yaml
workers:
  example_worker:
    type: remote
    scheduler_type: slurm
    work_dir: /path/to/run/folder
    resources:
    pre_run: source /path/to/python/environment/activate
    post_run:
    timeout_execute: 60
    max_jobs:
    batch:
    host: remote.host.net
    user: bob
    port:
    password:
    key_filename:
    passphrase:
    gateway:
    forward_agent:
    connect_timeout:
    connect_kwargs:
    inline_ssh_env:
    keepalive: 60
    shell_cmd: bash
    login_shell: true
    interactive_login: false
```



```python
from qtoolkit import QResources

qresources = QResources(queue_name='main',
                        job_name='myjob',
                        processes=24)

submit_flow(flow,
            worker='example_worker',
            resources=qresources)
```

Or you can directly pass a specific dictionary:

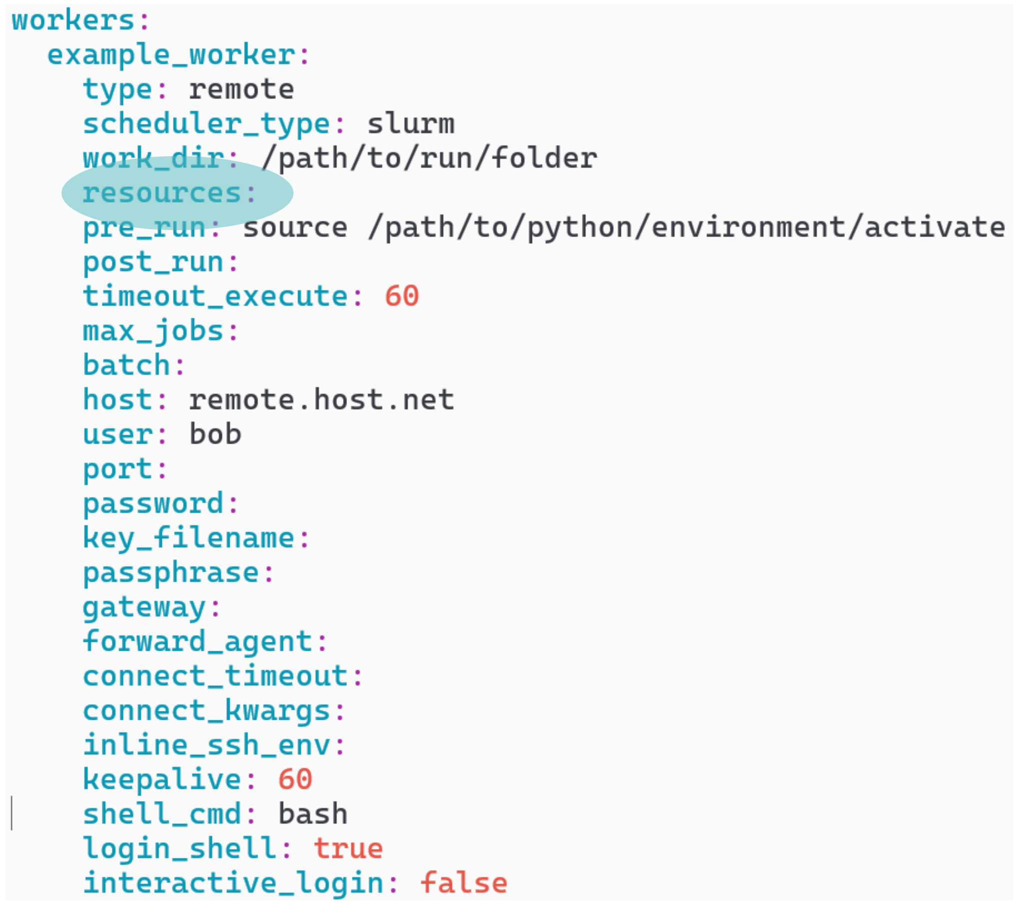


```python
submit_flow(flow,
            worker='example_worker',
            resources={'partition': 'main',
                       'job_name': 'myjob',
                       'ntasks': 24,}
```

Then it is slurm/pbs/…-specific

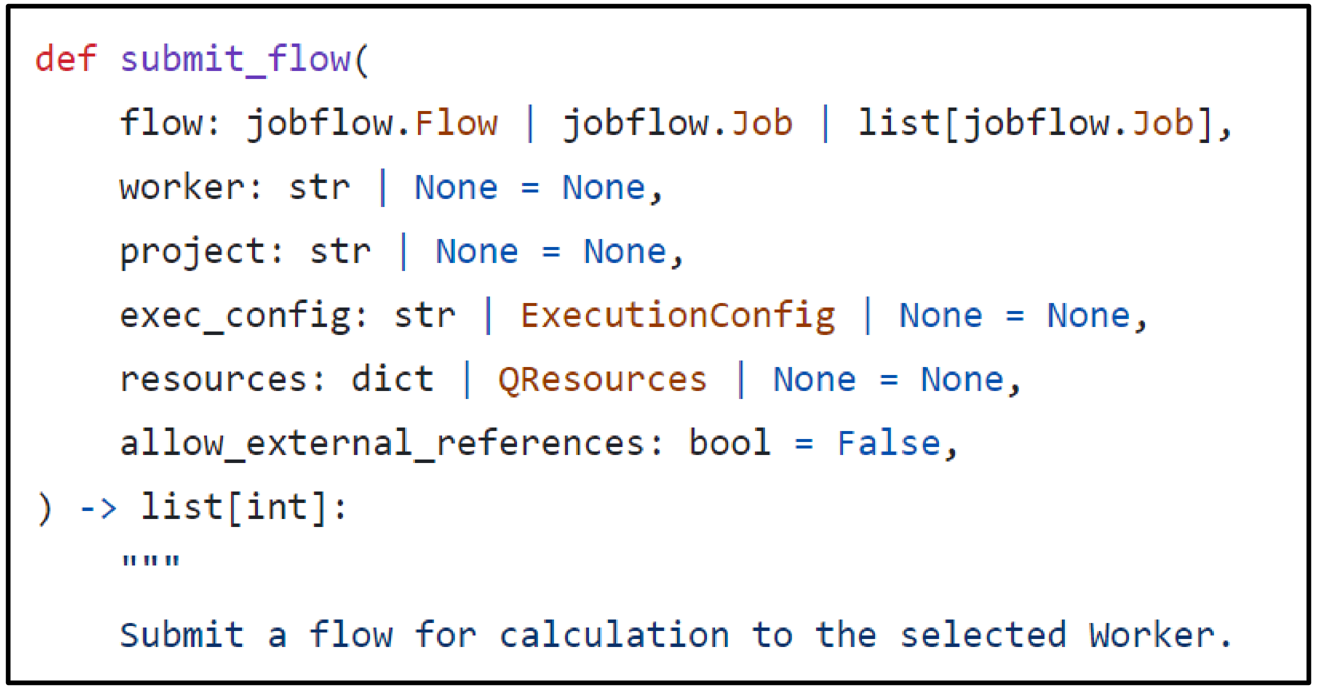
SETTING RESOURCES AT WORKER LEVEL

Resources



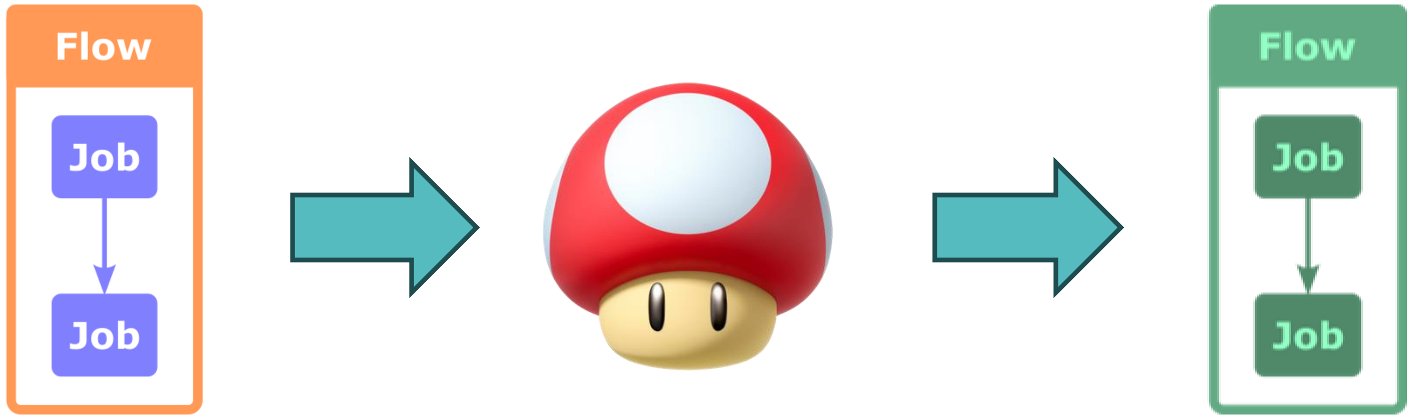
```yaml
workers:
  example_worker:
    type: remote
    scheduler_type: slurm
    work_dir: /path/to/run/folder
    resources:
    pre_run: source /path/to/python/environment/activate
    post_run:
    timeout_execute: 60
    max_jobs:
    batch:
    host: remote.host.net
    user: bob
    port:
    password:
    key_filename:
    passphrase:
    gateway:
    forward_agent:
    connect_timeout:
    connect_kwargs:
    inline_ssh_env:
    keepalive: 60
    shell_cmd: bash
    login_shell: true
    interactive_login: false
```

THE SUBMIT_FLOW FUNCTION



```
def submit_flow(
    flow: jobflow.Flow | jobflow.Job | list[jobflow.Job],
    worker: str | None = None,
    project: str | None = None,
    exec_config: str | ExecutionConfig | None = None,
    resources: dict | QResources | None = None,
    allow_external_references: bool = False,
) -> list[int]:
    """

    Submit a flow for calculation to the selected Worker.
```

USING A POWERUP



```
def set_run_config(
    flow_or_job: Flow | Job,
    name_filter: str = None,
    function_filter: Callable = None,
    exec_config: str | ExecutionConfig | None = None,
    resources: dict | QResources | None = None,
    worker: str | None = None,
    dynamic: bool = True,
) -> Flow | Job:
    """
    Modify in place a Flow or a Job by setting the properties in the
    "manager_config" entry in the JobConfig associated to each Job
    matching the filter.
```

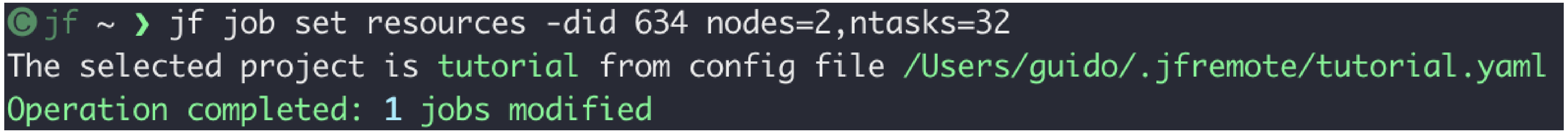
```
flow = set_run_config(flow,
                          name_filter='relax1',
                          resources={'partition': 'main',
                                     'job_name': 'myjob',
                                     'ntasks': 24,})

submit_flow(flow,
            worker='example_worker')
```

SETTING RESOURCES WITH CLI

Modify resources after job has been submitted with submit_flow

Only for READY Jobs to ensure not yet submitted to the HPC queue

CLI: `jf job set resources`

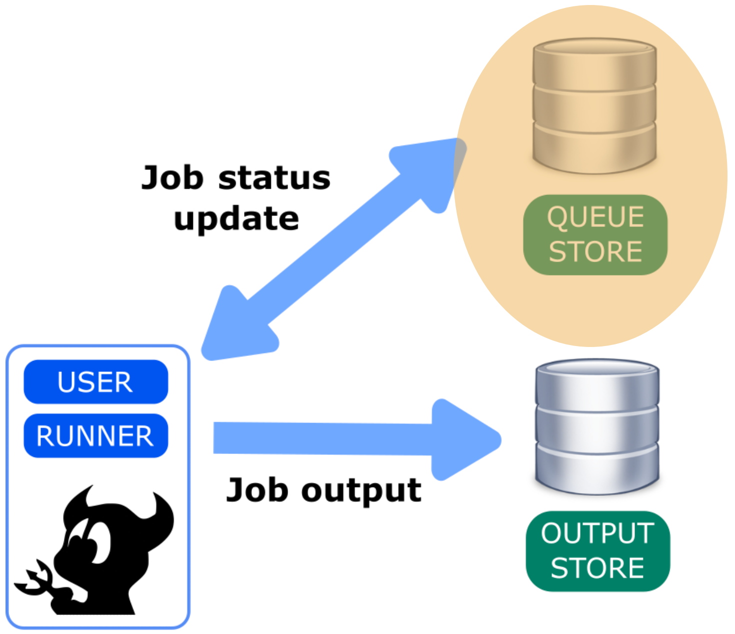


```
jf ~ ❯ jf job set resources -did 634 nodes=2,ntasks=32
The selected project is tutorial from config file /Users/guido/.jfremote/tutorial.yaml
Operation completed: 1 jobs modified
```

BACKEND DETAILS

QUEUE DB STRUCTURE

- Job documents collection
- Flow documents collection
- Auxiliary collection (unique index, …)



JobDoc

- Job as_dict
- Uuid
- Index
- Db_id: unique id
- State
- Parents (uuid)
- Errors
- Run info (remote, dates, resources,…)

FlowDoc

- Flow uuid
- Jobs ids (uuid, db_id, index)
- Job connections
- State

REMOTE EXECUTION: JOB SUBMISSION

- Jobs cannot be executed directly
  - HPC infrastructure is shared between users
- A Distributed Resource Management (DRM) system (e.g. SLURM) is used to schedule jobs submitted by the users using special scripts:

```
sbatch script.sh

#script content:
…
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=10
#SBATCH --mem=64gb
#SBATCH --time=2-00:00:00
…
```

- Need for a python interface to automatically generate these scripts
- Existing solutions “buried” inside large codes or not directly usable

=> Implementation of a Python interface for jobs submission

JOB SUBMISSION: QTOOLKIT

- Github repository: https://github.com/Matgenix/qtoolkit
- Documentation: https://matgenix.github.io/qtoolkit
- Open source
- License: modified BSD (3-clause BSD)

QTOOLKIT: FEATURES

- Programmatic API
- Well-defined objects to represent a job in a queue, its state, additional information, …
- Submit jobs to PBS, Slurm, Shell, …
- Get info about a job in a queue
- Get list of jobs in a queue
- No dependency on any external package (only optional dependency on monty)
- Used by jobflow-remote

SUMMARY

SUMMARY

- Execution process
  - Runner
  - States evolution
- How to interact with jobflow-remote
  - CLI, python API, GUI
- Configurations
  - Setting up a project
- Fine tuning job execution
- Some backend details
- Dealing with failures

THANK YOU

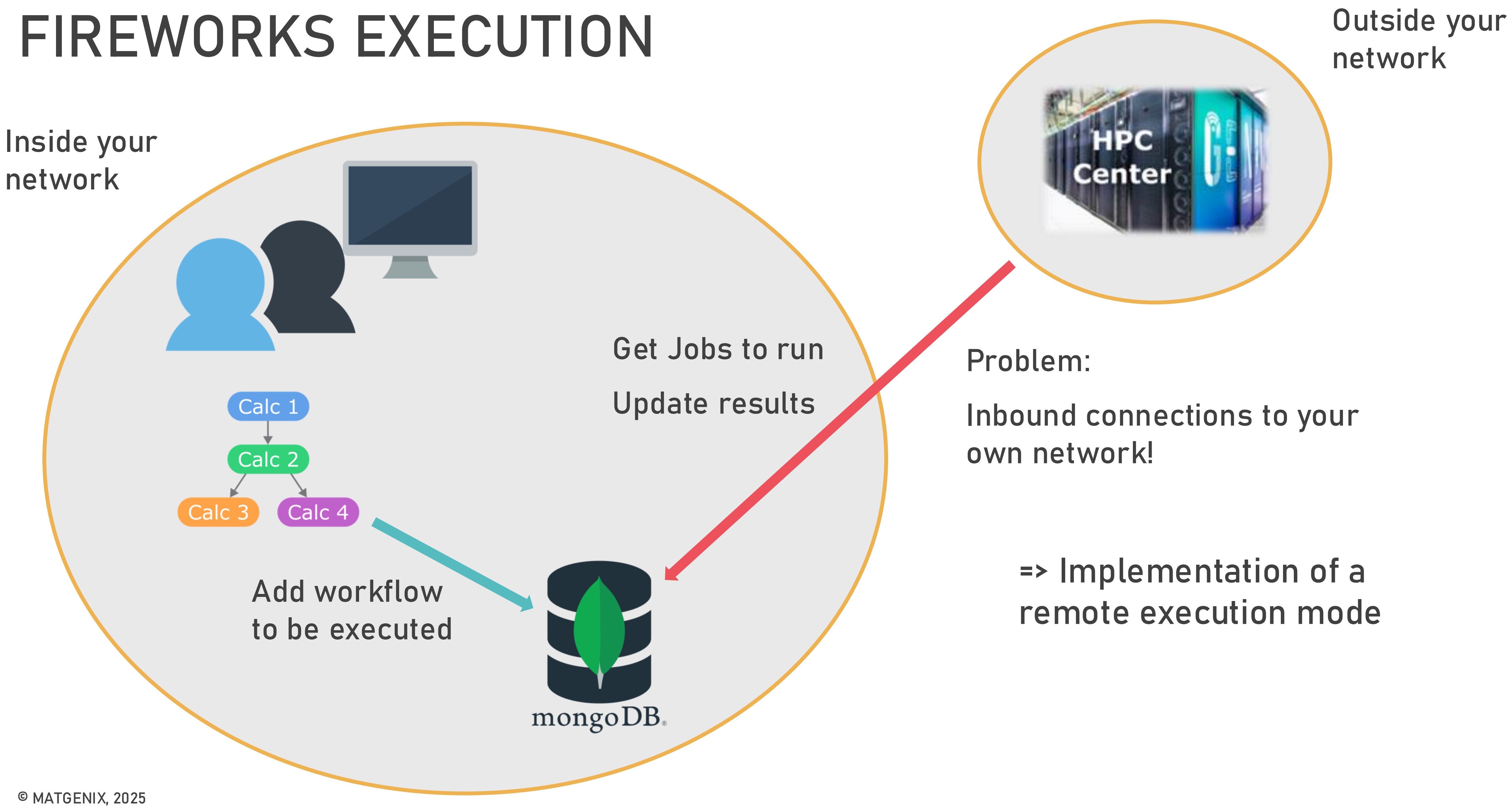
HANDS-ON

Jupyter notebooks (~/work/notebooks/jobflow_remote):

1. Introduction
   - Submit flows
   - Runner
   - CLI
2. Handling errors
   - Failures
   - Remote errors
   - Rerun/retry
3. Configuration
   - Set up a new project

Matgenix

FIREWORKS EXECUTION

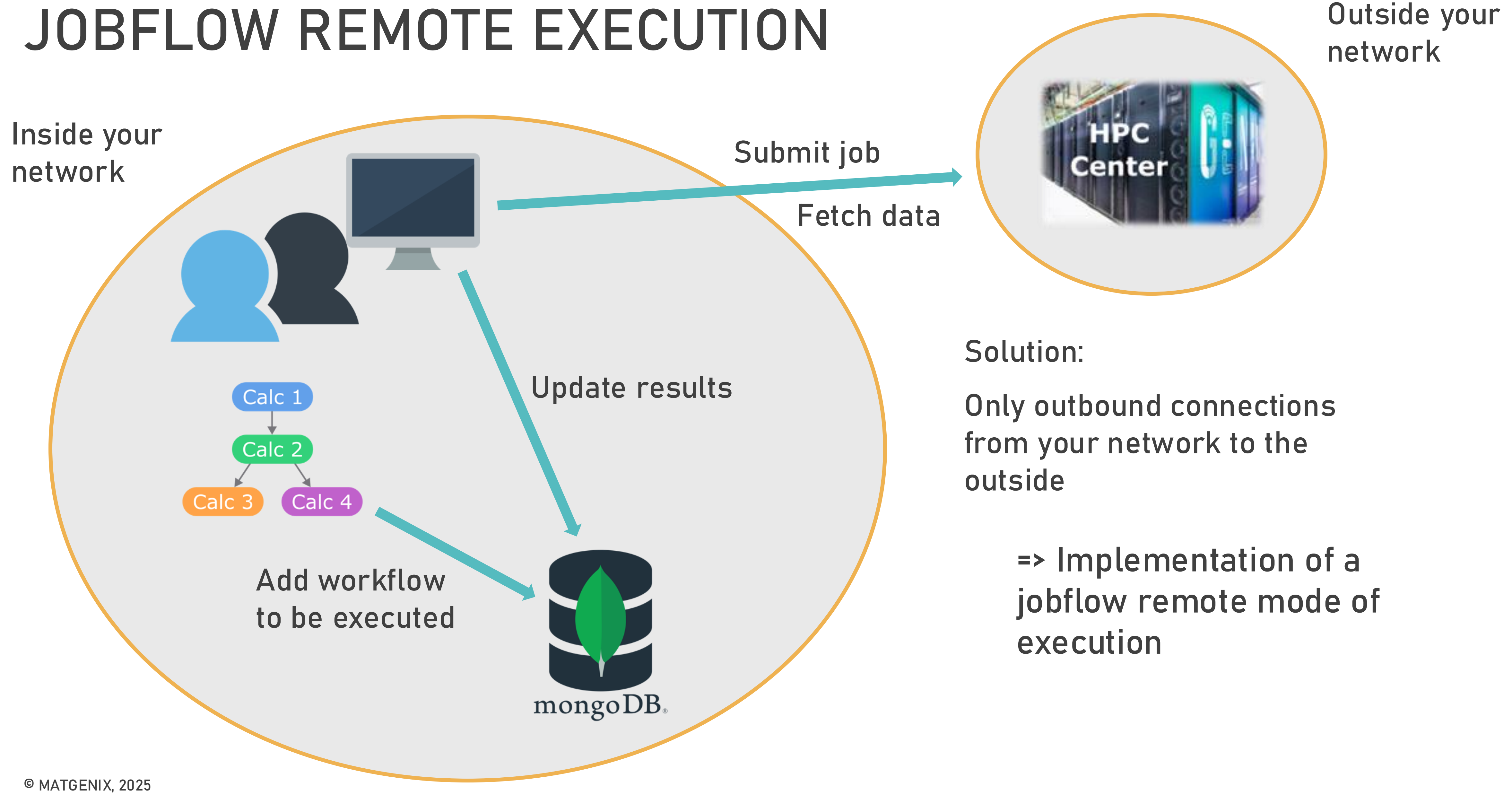


Problem:

Inbound connections to your own network!

=> Implementation of a remote execution mode

JOBFLOW REMOTE EXECUTION

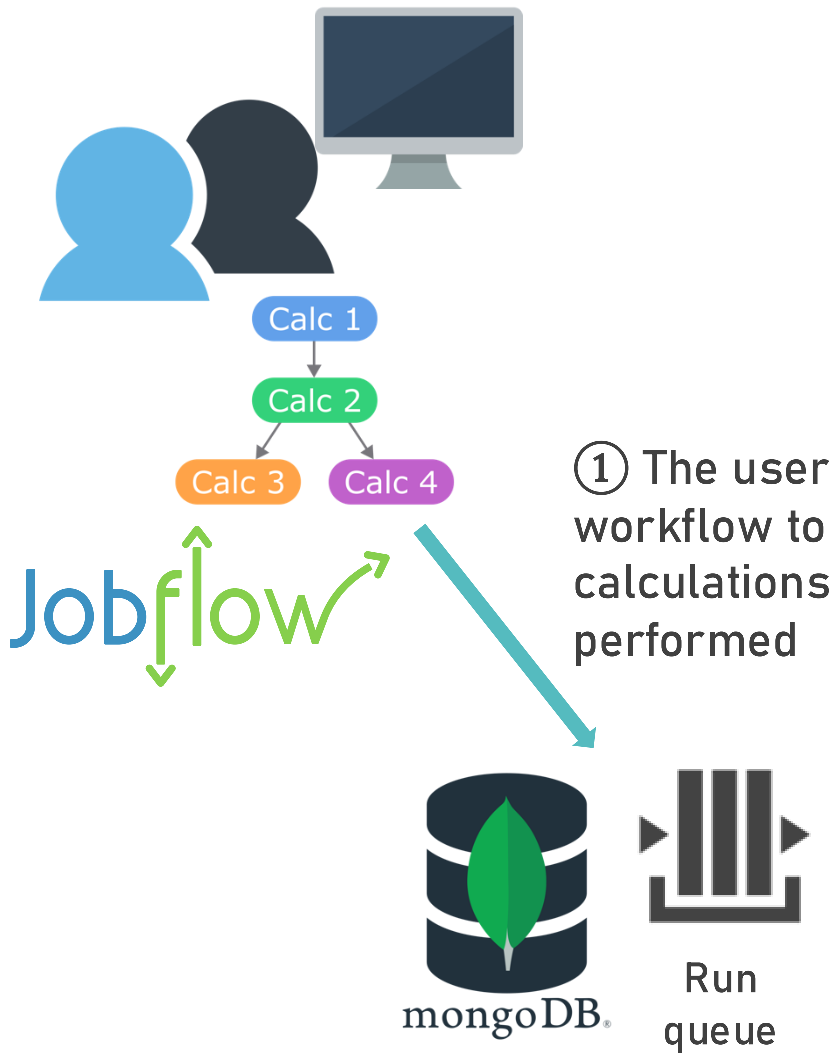


Solution:

Only outbound connections from your network to the outside

=> Implementation of a jobflow remote mode of execution

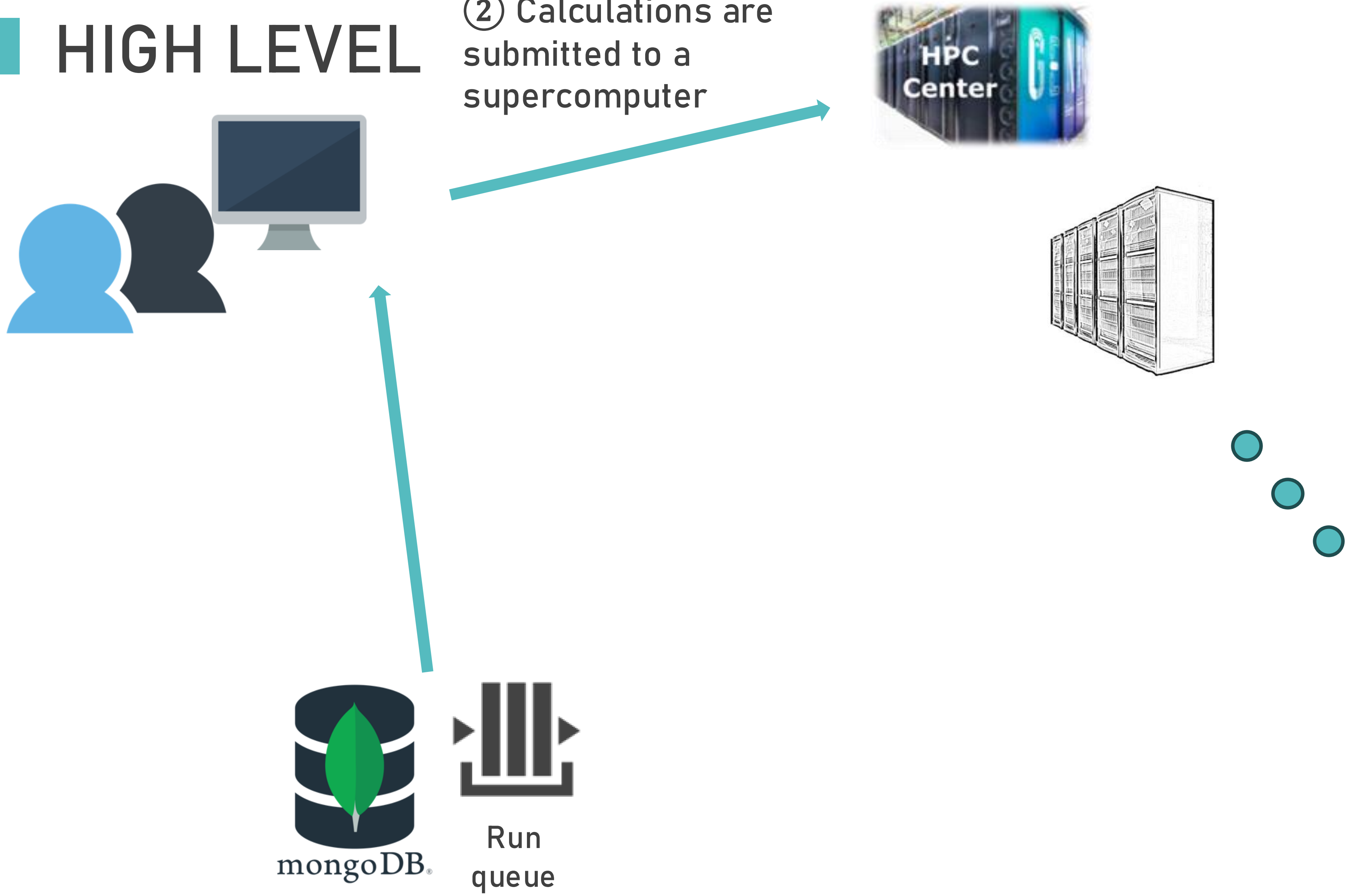
HIGH LEVEL



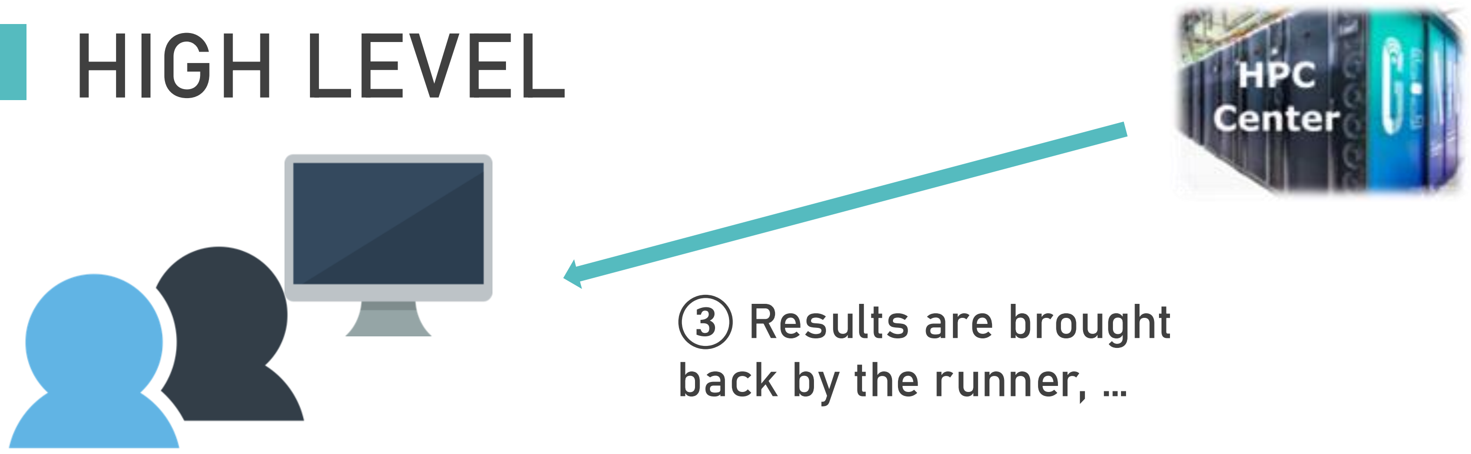
① The user adds a workflow to the list of calculations to be performed

HIGH LEVEL

② Calculations are submitted to a supercomputer

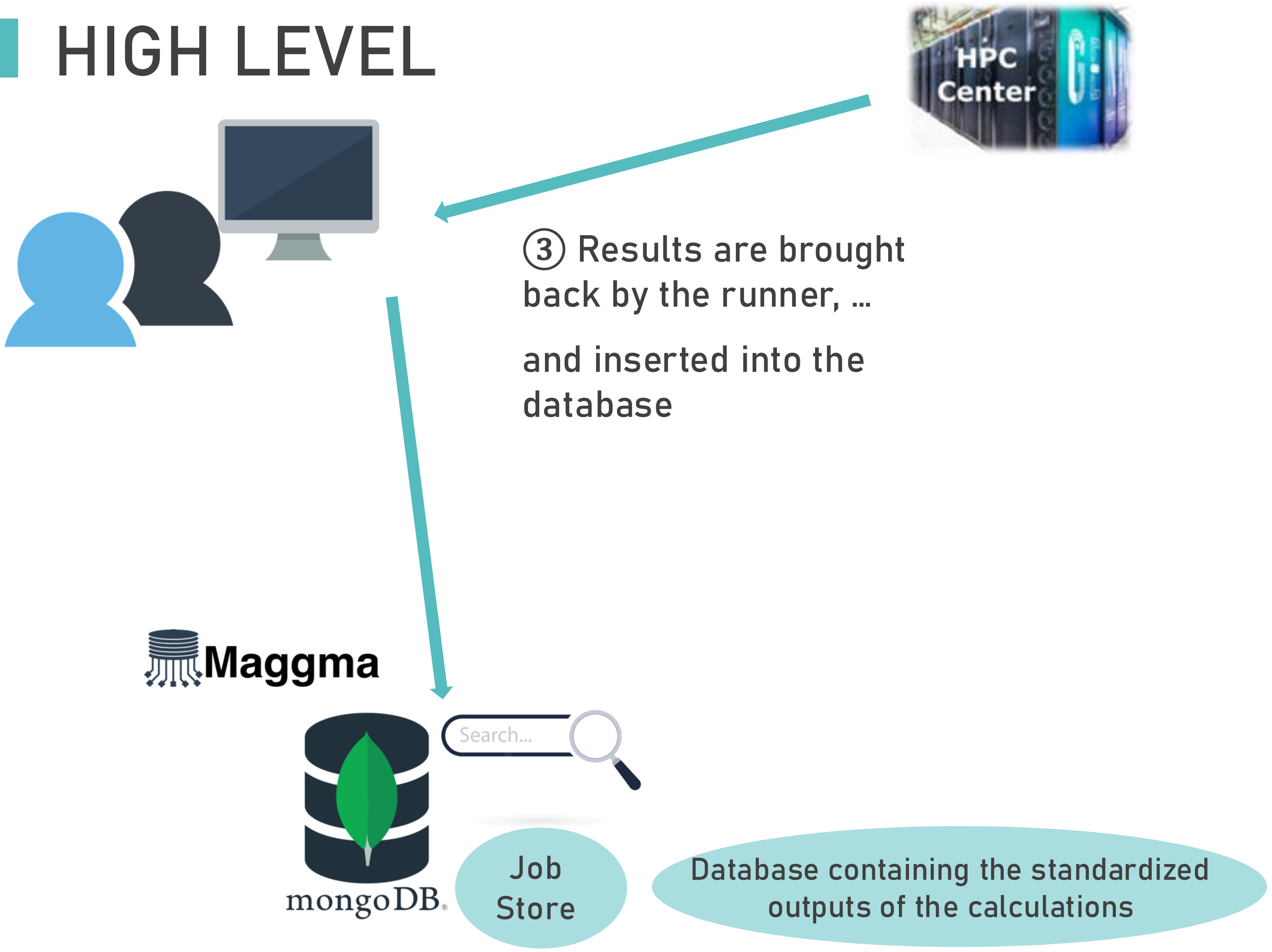


HIGH LEVEL



③ Results are brought back by the runner, …

HIGH LEVEL

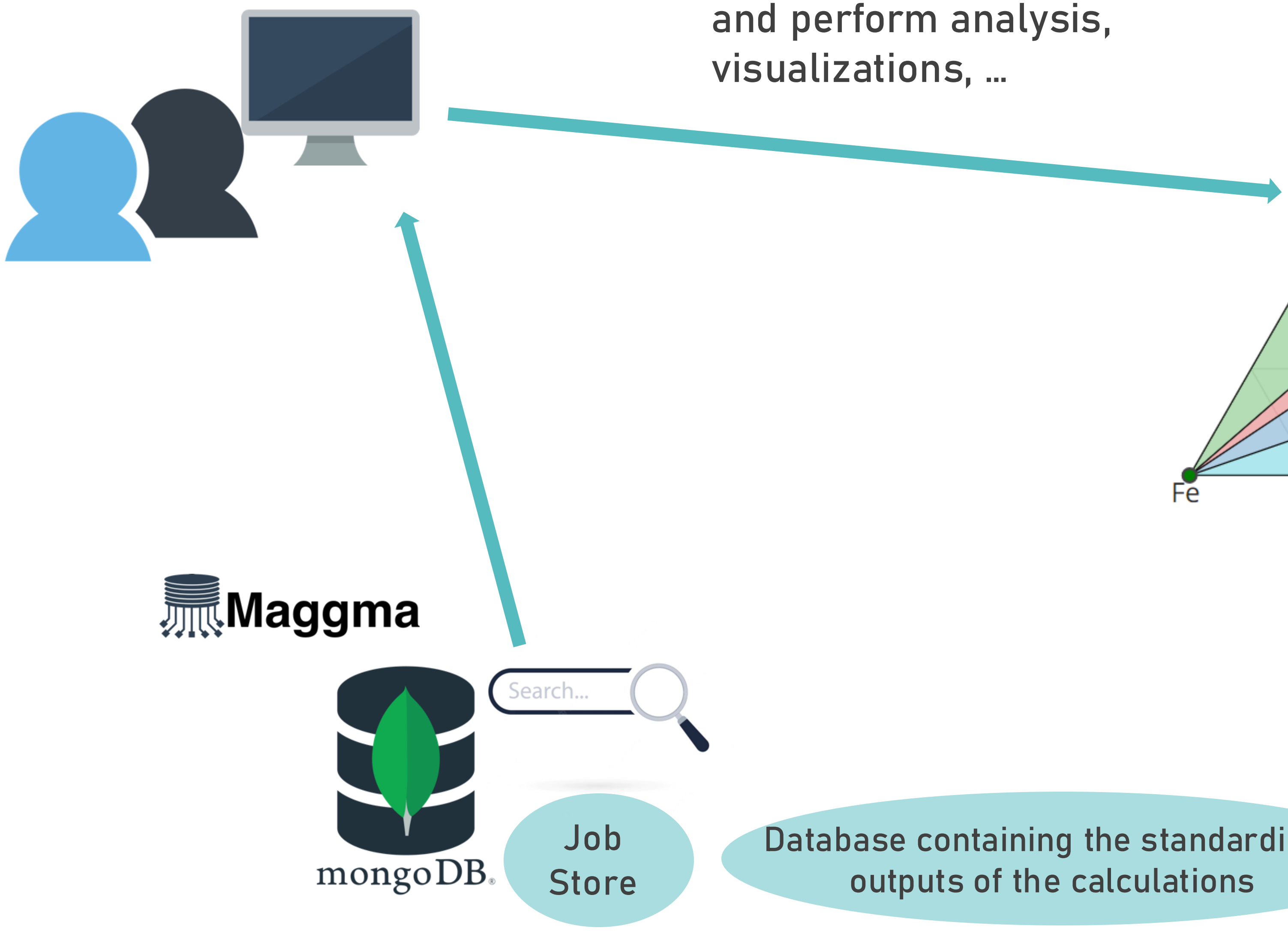

HPC Center
Maggma
Search...
mongoDB
Job Store
Database containing the standardized outputs of the calculations

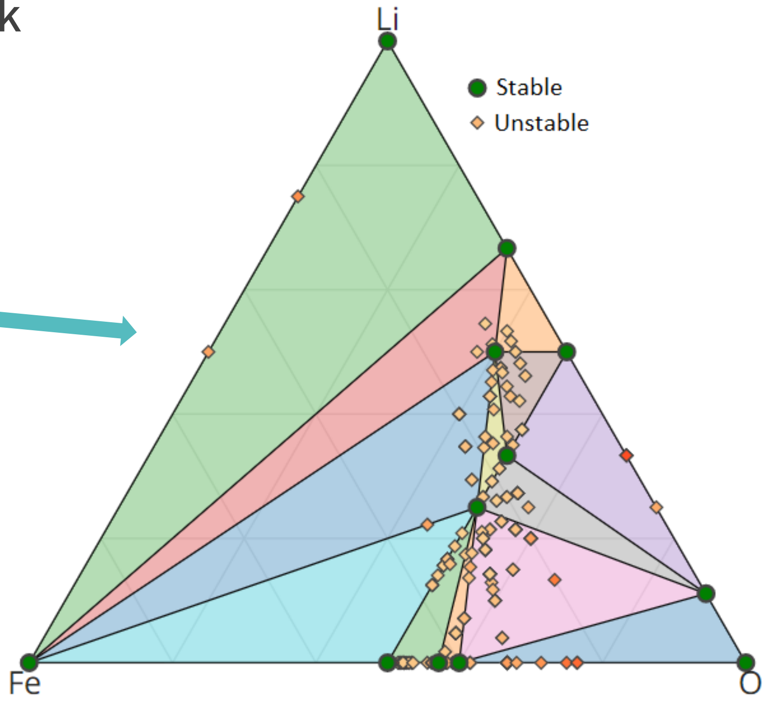
③ Results are brought back by the runner, …

and inserted into the database

HIGH LEVEL

④ The user can access the results from the work station/virtual machine and perform analysis, visualizations, …

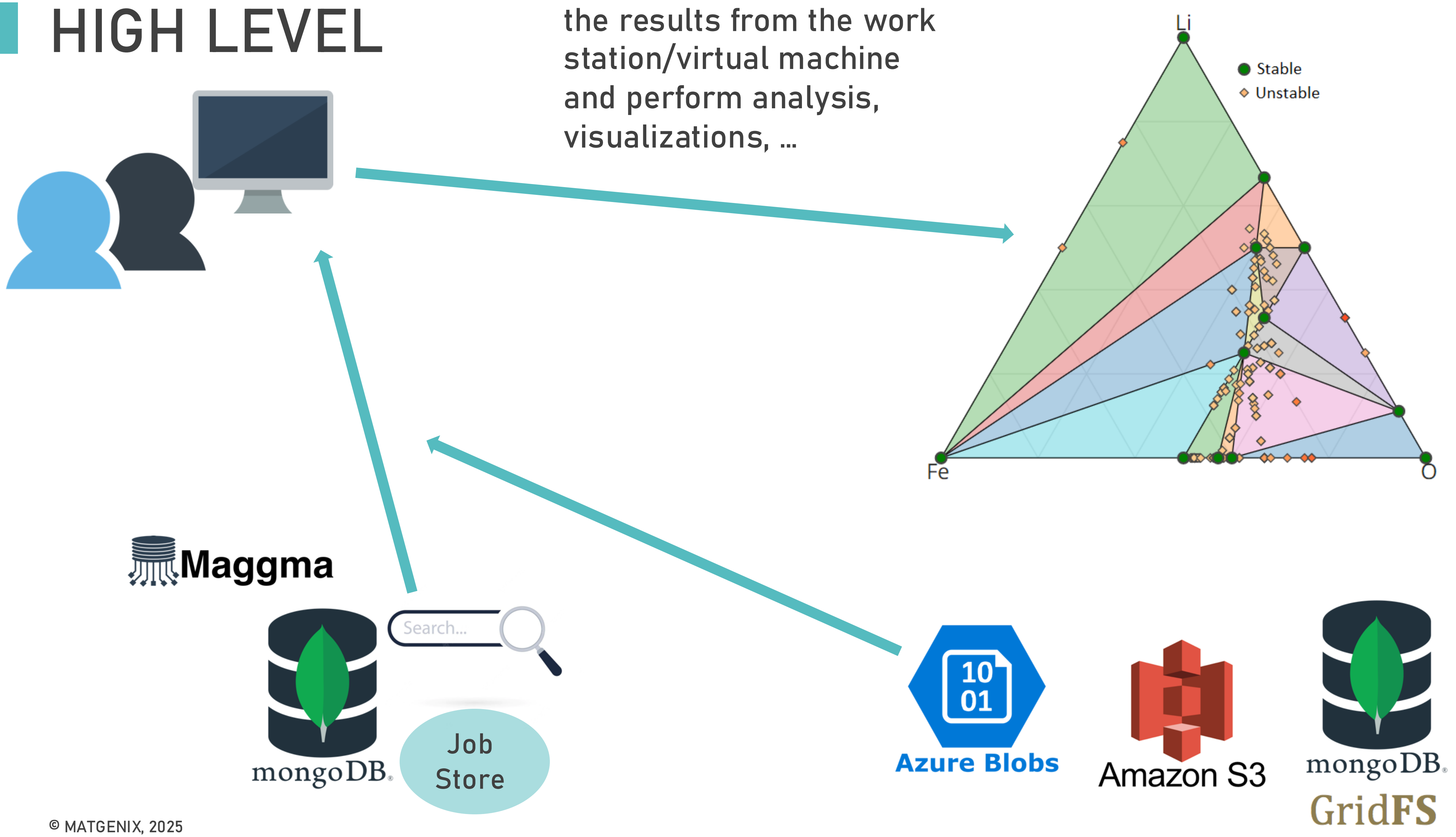




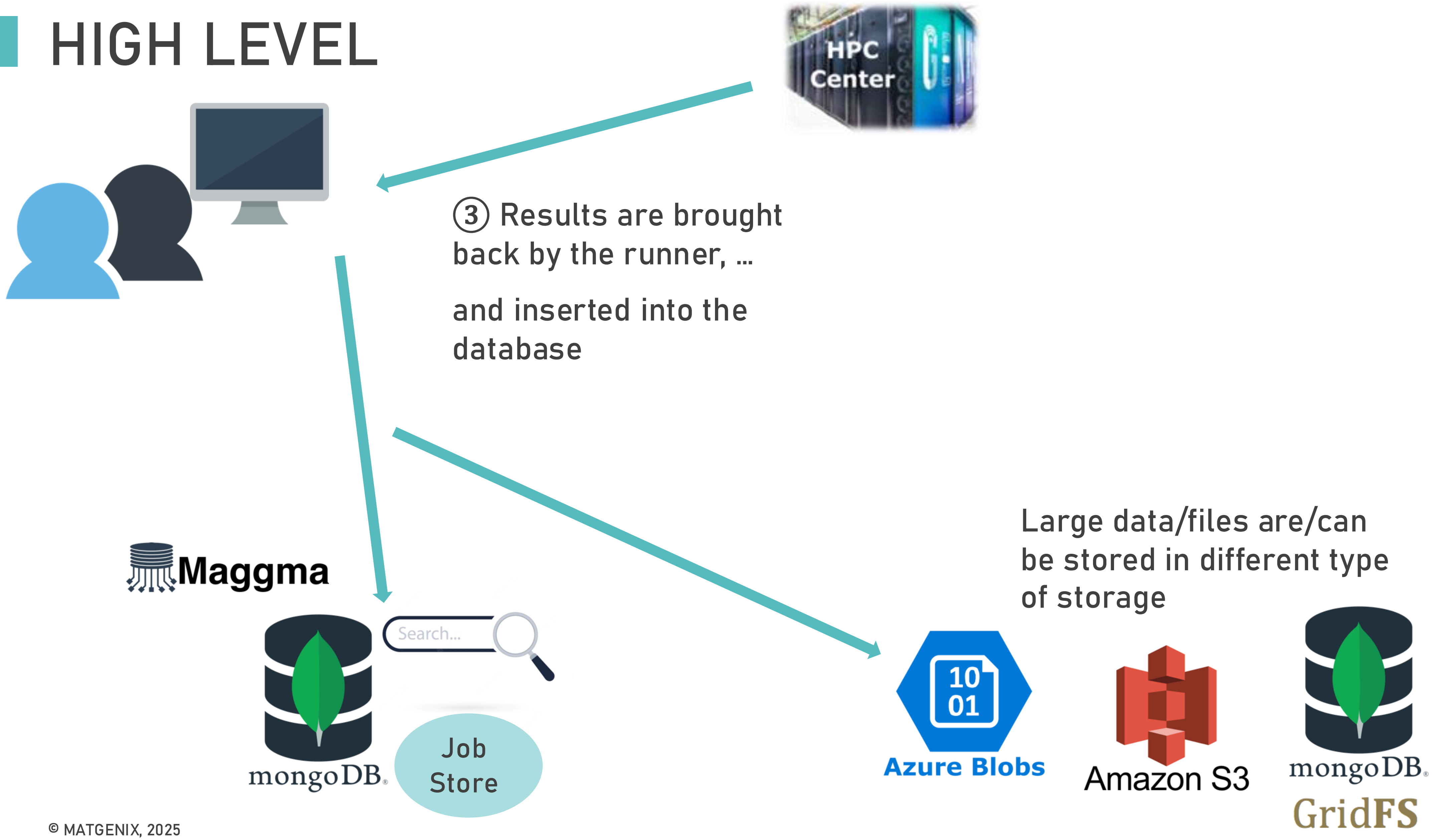
Database containing the standardized outputs of the calculations

HIGH LEVEL

④ The user can access the results from the work station/virtual machine and perform analysis, visualizations, …



HIGH LEVEL



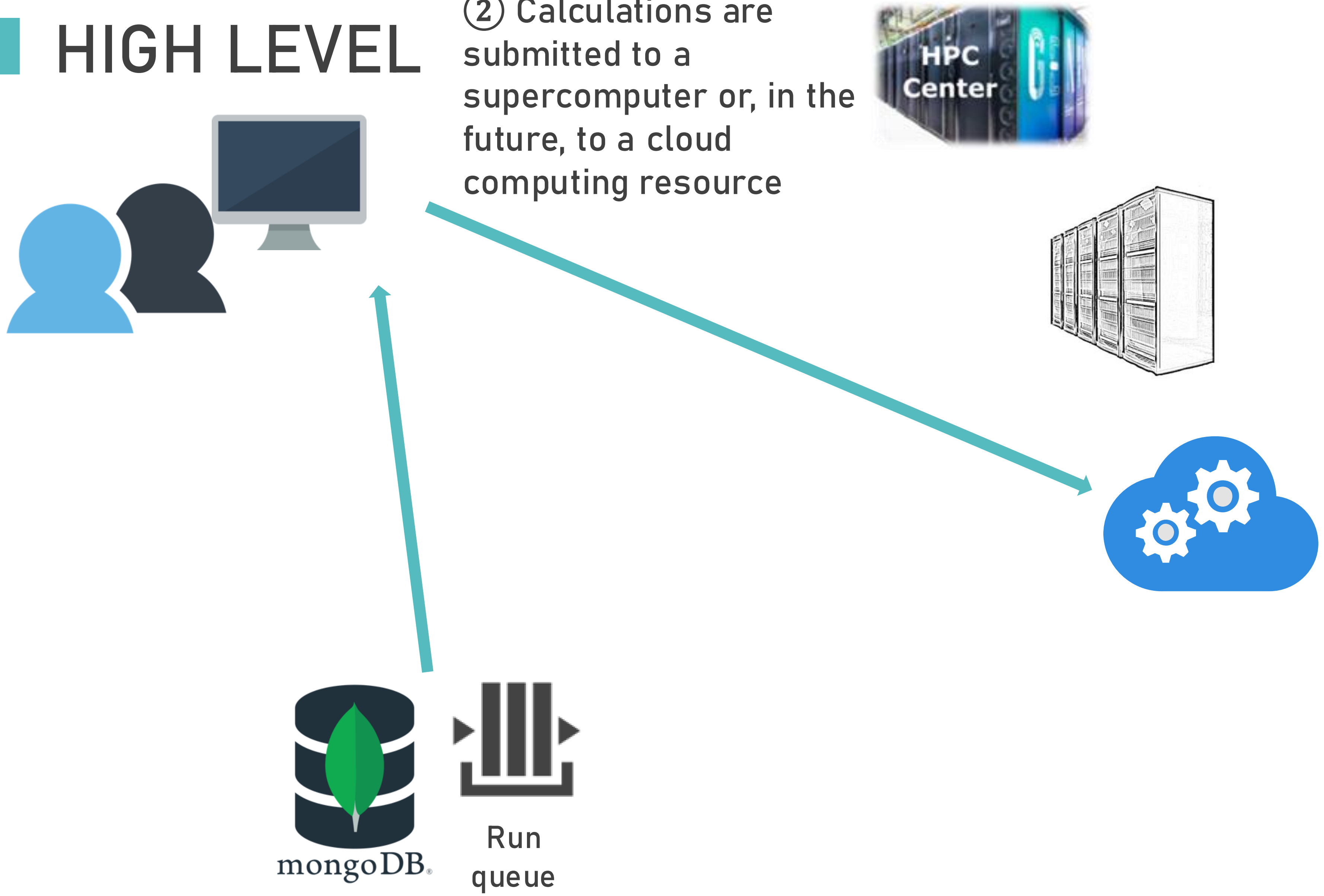
③ Results are brought back by the runner, …

and inserted into the database

Large data/files are/can be stored in different type of storage

HIGH LEVEL

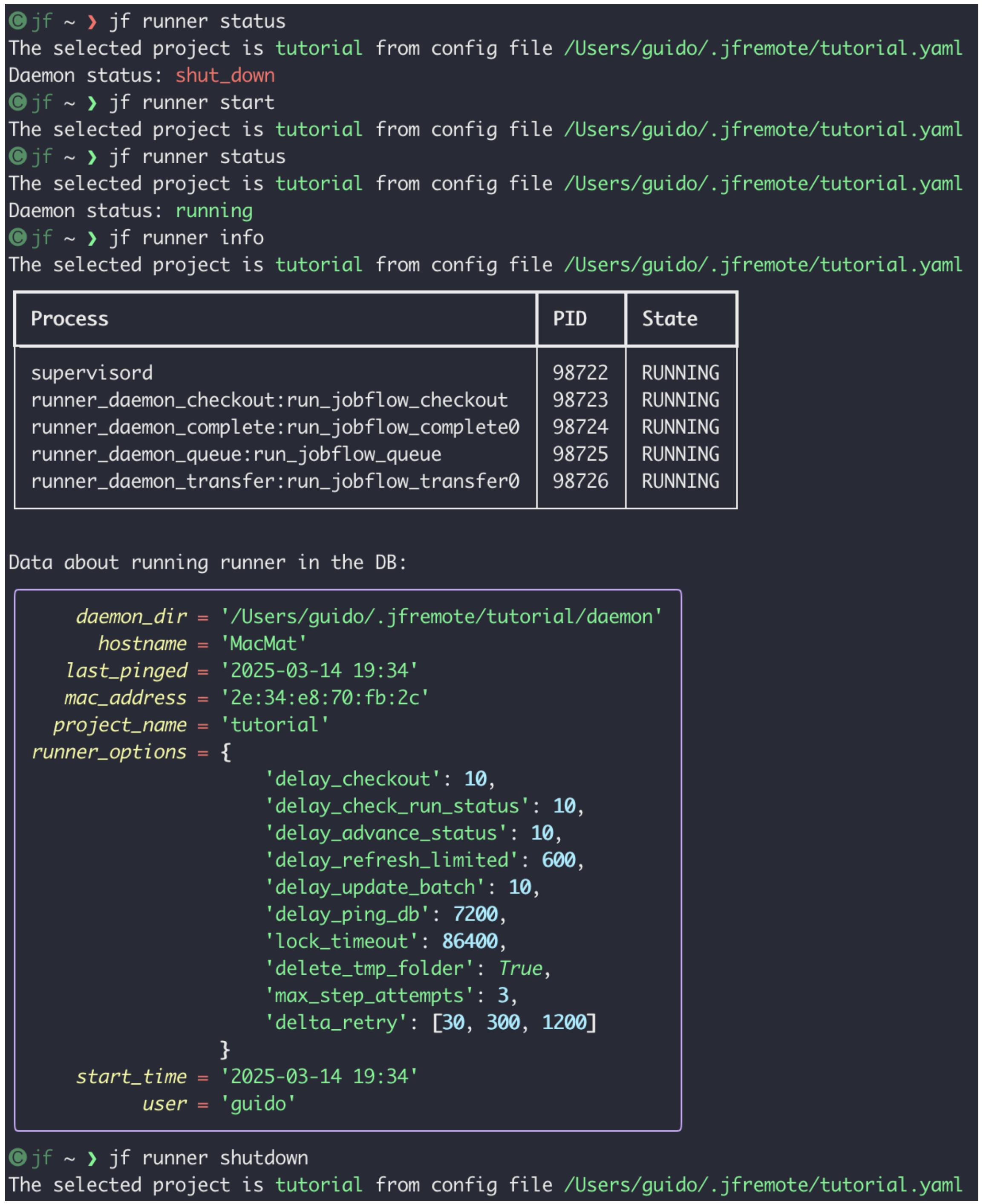
② Calculations are submitted to a supercomputer or, in the future, to a cloud computing resource



CLI - RUNNER

runner: control the Runner

- Start
- Stop
- Status
- Subprocesses information
- Kill



```
jf ~ ❯ jf runner status
The selected project is tutorial from config file /Users/guido/.jfremote/tutorial.yaml
Daemon status: shut_down
jf ~ ❯ jf runner start
The selected project is tutorial from config file /Users/guido/.jfremote/tutorial.yaml
jf ~ ❯ jf runner status
The selected project is tutorial from config file /Users/guido/.jfremote/tutorial.yaml
Daemon status: running
jf ~ ❯ jf runner info
The selected project is tutorial from config file /Users/guido/.jfremote/tutorial.yaml
┌───────────────────────────────────────────────────┬───────┬─────────┐
│ Process                                           │ PID   │ State   │
├───────────────────────────────────────────────────┼───────┼─────────┤
│ supervisord                                       │ 98722 │ RUNNING │
│ runner_daemon_checkout:run_jobflow_checkout       │ 98723 │ RUNNING │
│ runner_daemon_complete:run_jobflow_complete0      │ 98724 │ RUNNING │
│ runner_daemon_queue:run_jobflow_queue             │ 98725 │ RUNNING │
│ runner_daemon_transfer:run_jobflow_transfer0      │ 98726 │ RUNNING │
└───────────────────────────────────────────────────┴───────┴─────────┘

Data about running runner in the DB:
╭───────────────────────────────────────────────────────────╮
│     daemon_dir = '/Users/guido/.jfremote/tutorial/daemon'  │
│       hostname = 'MacMat'                                  │
│    last_pinged = '2025-03-14 19:34'                        │
│    mac_address = '2e:34:e8:70:fb:2c'                       │
│   project_name = 'tutorial'                                │
│ runner_options = {                                         │
│                      'delay_checkout': 10,                 │
│                      'delay_check_run_status': 10,         │
│                      'delay_advance_status': 10,           │
│                      'delay_refresh_limited': 600,         │
│                      'delay_update_batch': 10,             │
│                      'delay_ping_db': 7200,                │
│                      'lock_timeout': 86400,                │
│                      'delete_tmp_folder': True,            │
│                      'max_step_attempts': 3,               │
│                      'delta_retry': [30, 300, 1200]        │
│                  }                                         │
│     start_time = '2025-03-14 19:34'                        │
│           user = 'guido'                                   │
╰───────────────────────────────────────────────────────────╯
jf ~ ❯ jf runner shutdown
The selected project is tutorial from config file /Users/guido/.jfremote/tutorial.yaml
```