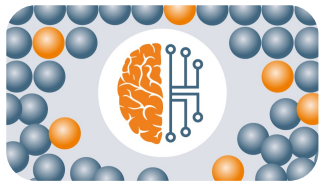
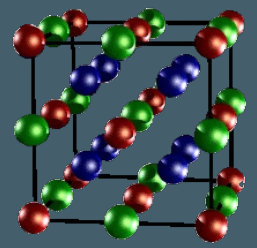


MAX PLANCK INSTITUTE
FOR SUSTAINABLE MATERIALS

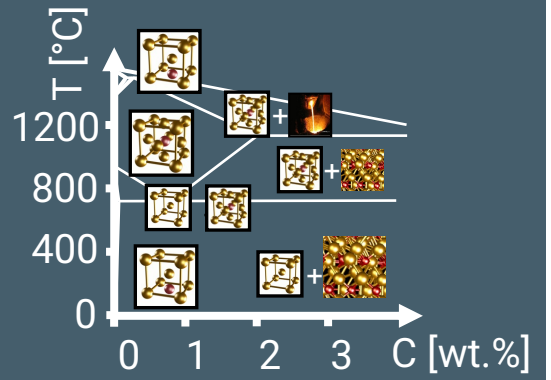
Up-scale Python Functions for High-Performance Computing with Executorlib

Jan Janssen – Group Leader for Materials Informatics



Atomistic Simulation

High-Performance Computing



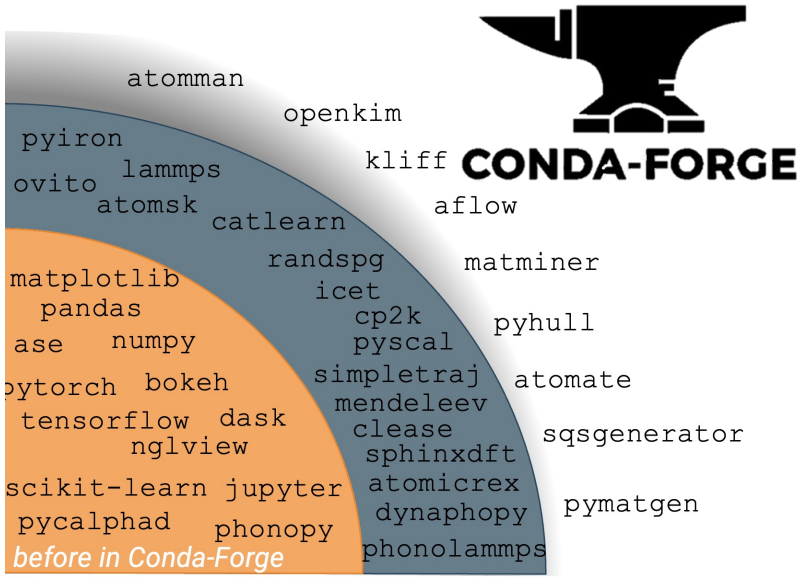
Material Property

2026/01/14 – Workflow Community Talk

Materials Informatics Group

Our Expertise: Simulation Workflows to Predict Sustainable Materials

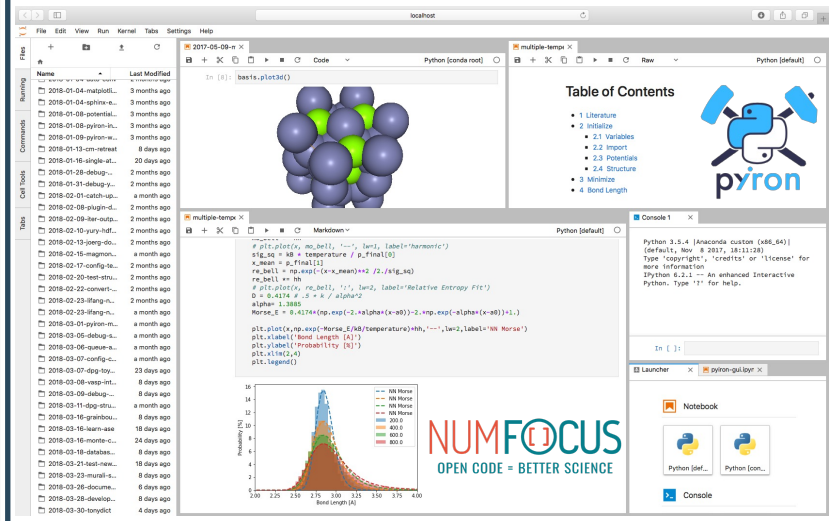
Conda-Forge
Installation of Scientific Software



Maintaining over 1000 materials informatics packages on conda-forge with over 500 million downloads so far.

https://github.com/jan-janssen/conda-forge-contribution

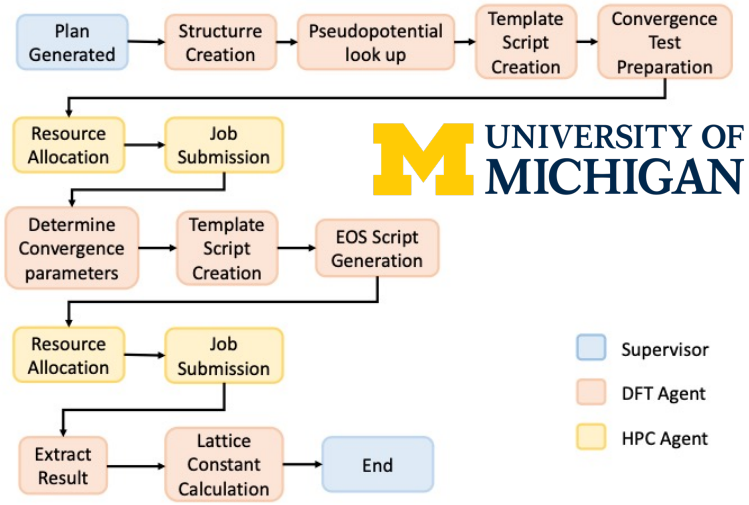
pyiron
Workflow Framework



Jupyter Lab based interface for data-driven materials informatics to enable rapid prototyping and up-scaling.

https://github.com/pyiron - https://pyiron.org

Large Language Model Agents
for atomistic simulations



Reduce hallucination by restricting the large language model to use pre-defined workflows developed by experts.

Z. Wang, H. Huang, H. Zhao, C. Xu, J. Janssen and V. Viswanathan – arXiv 2507.14267 (2025).

Three Levels of Workflow Interoperability in Materials Science

Simplify the exchange of workflows between the different workflow frameworks

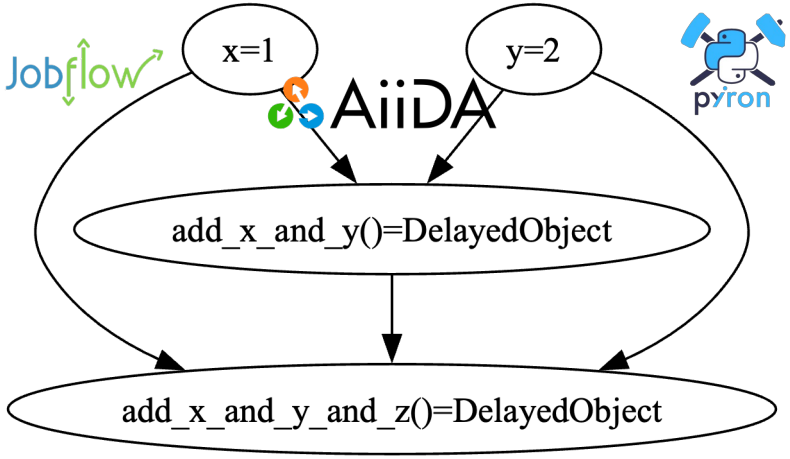
Shared Python Functions

```python
def add_x_and_y(x, y):
    z = x + y
    return z

def add_x_and_y_and_z(x, y, z):
    w = x + y + z
    return w
```

Python functions are defined once and can be reused with different Workflow Managers.

- Easy to implement ✓
- Dependencies are lost ✗

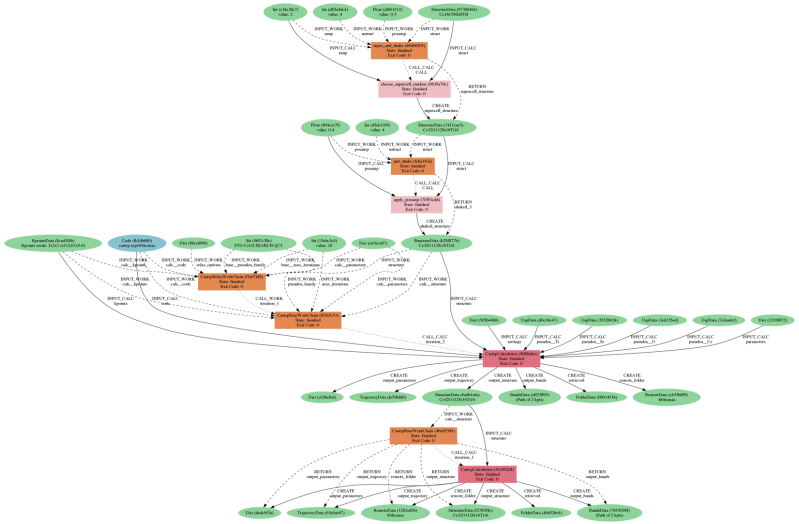
Python Workflow Definition



Export workflow in the Python Workflow Definition to share with different Workflow Frameworks.

- Reproducibility ✓
- Repeated Calculation ✗

Workflow Graph with Data



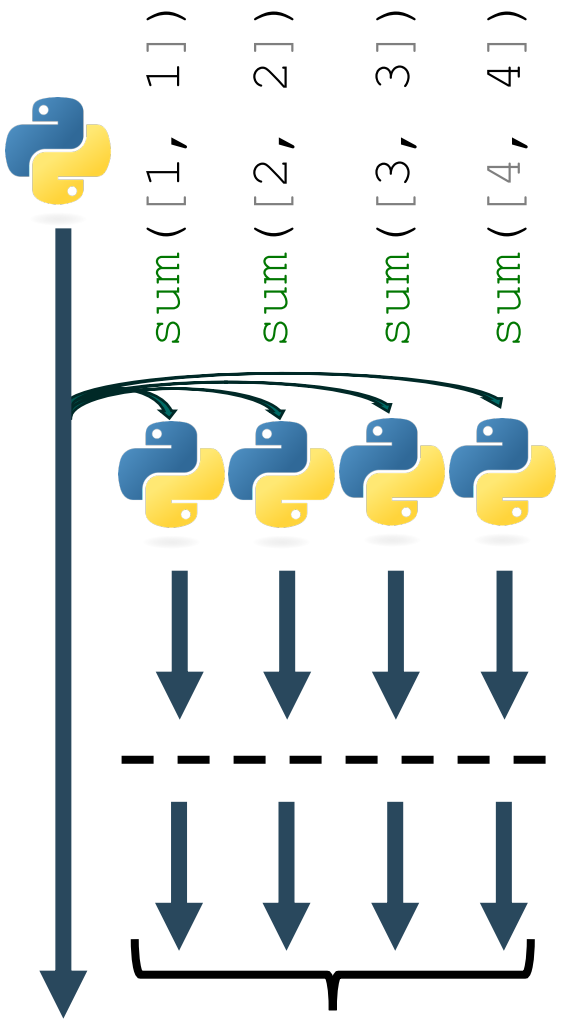
Sharing the provenances between different Workflow Frameworks enables efficient collaboration.

J. Janssen et al., Digital Discovery (2025). - https://github.com/pythonworkflow

Python Standard Library

High-level Interface for Asynchronously Executing Callables

```
from concurrent.futures import ProcessPoolExecutor


with ProcessPoolExecutor() as exe:
    future_lst = []
    for i in range(1, 5):
        future_lst.append(exe.submit(sum, [i, i]))


    # check status
    print([f.done() for f in future_lst])


    # wait for computations to complete
    print([f.result() for f in future_lst])
```



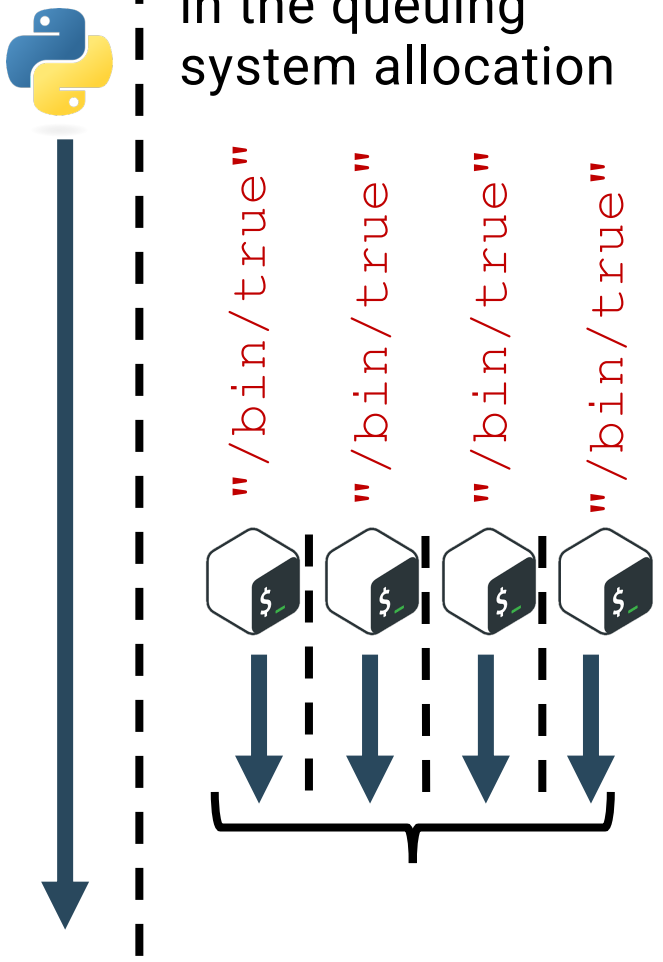
https://docs.python.org/3/library/concurrent.futures.html#processpoolexecutor

Flux Framework

Hierarchical Job Scheduler

```python
import concurrent.futures
import flux.job


jobspec = flux.job.JobspecV1.from_command(
    ["/bin/true"]
)
with flux.job.FluxExecutor() as exe:
    future_lst = []
    for i in range(1, 5):
        future_lst.append(exe.submit(jobspec))


    # wait for computations to complete
    print([f.result() for f in future_lst])
```

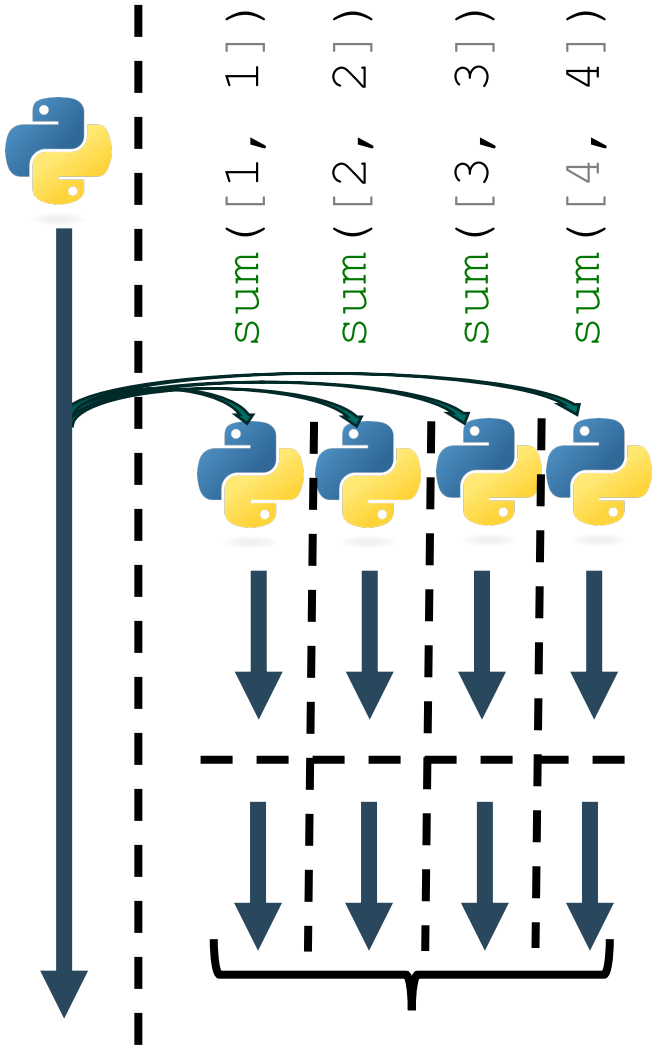
Tasks are distributed in the queuing system allocation



Executorlib

Combines the Python Standard Library Executor Interface with Flux

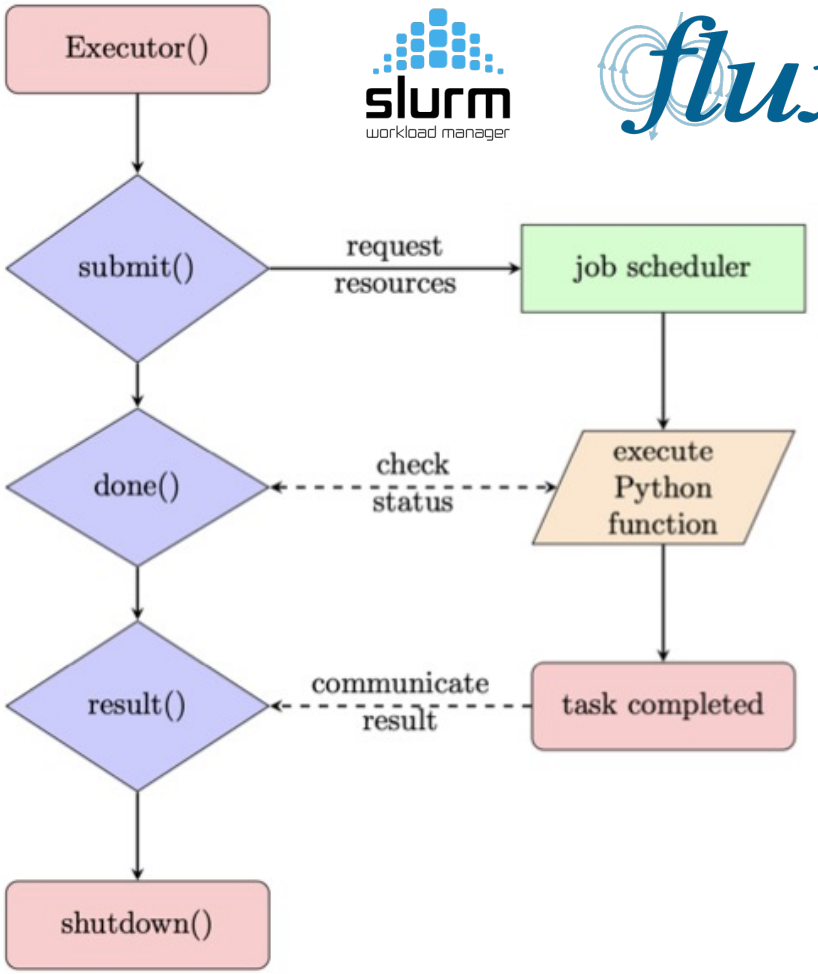
```python
from executorlib import FluxJobExecutor


with FluxJobExecutor() as exe:
    future_lst = []
    for i in range(1, 5):
        future_lst.append(exe.submit(sum, [i, i]))


    # check status
    print([f.done() for f in future_lst])


    # wait for computations to complete
    print([f.result() for f in future_lst])
```



Janssen et al. JOSS, 10, 108 (2025) - https://github.com/pyiron/executorlib

Up-scale your Python Functions with Executorlib

No Database, No Daemon Process, Just Job Schedulers



Executor	Communication	Scheduler
`SingleNodeExecutor`	Socket	
`SlurmClusterExecutor`	File	SLURM
`SlurmJobExecutor`	Socket	SLURM
`FluxClusterExecutor`	File	Flux
`FluxJobExecutor`	Socket	Flux

Janssen et al. JOSS, 10, 108 (2025) - https://github.com/pyiron/executorlib

Message Passing Interface in Python

Assignment of Computational Resources

```
from executorlib import FluxJobExecutor


def calc_mpi(i):
    from mpi4py import MPI
    size = MPI.COMM_WORLD.Get_size()
    rank = MPI.COMM_WORLD.Get_rank()
    return i, size, rank


with FluxJobExecutor() as exe:
    fs = exe.submit(calc_mpi, 3, resource_dict={
        "cores": 2,
        "threads_per_core": 1,
        "gpus_per_core": 1,
        "cwd": "/a/b/c",
    })
```


Janssen et al. JOSS, 10, 108 (2025) - https://github.com/pyiron/executorlib

Cache Results

Beyond the Execution

```
from executorlib import SingleNodeExecutor, get_cache_data


def calc_add(a, b):
    return a + b


with SingleNodeExecutor(cache_directory="./cache") as exe:
    future = 0
    for i in range(1, 4):
        future = exe.submit(calc_add, i, future)
    print(future.result())
```

Cache Results

Beyond the Execution

```
with SingleNodeExecutor(cache_directory="./cache") as exe:
    future = 0
    for i in range(1, 4):
        future = exe.submit(calc_add, i, future)
    print(future.result())
```

```
pandas.DataFrame(get_cache_data(cache_directory="./cache"))
```

	function	input_args	input_kwargs	output	runtime
0	<built-in function sum>	[[1, 1]]	{}	2	0.180522
1	<built-in function sum>	[[2, 2]]	{}	4	0.179688
2	<built-in function sum>	[[3, 3]]	{}	6	0.132677

Janssen et al. JOSS, 10, 108 (2025) - https://github.com/pyiron/executorlib

From Python Functions to Workflows – Dependency Management

Link Python Functions with Different Resource Requirements

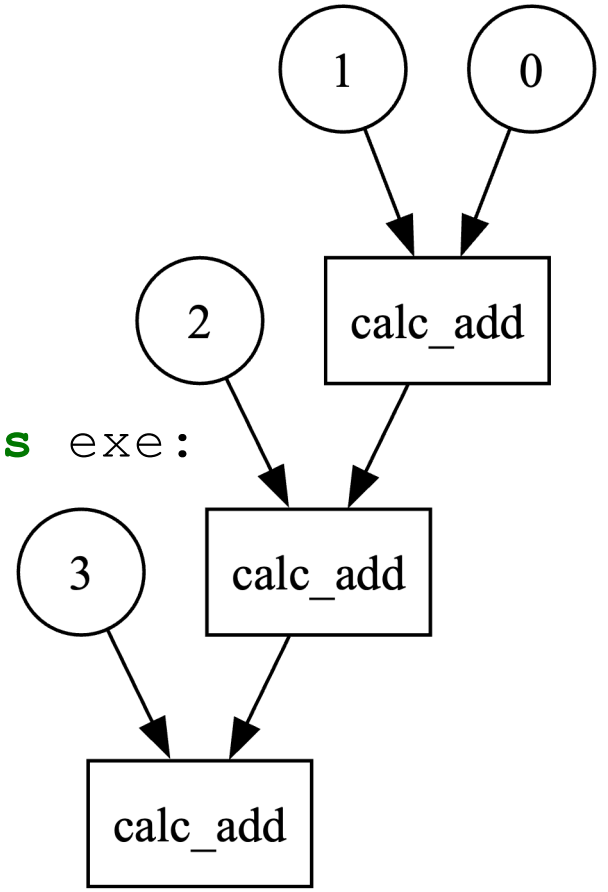
```python
from executorlib import SingleNodeExecutor


def calc_add(a, b):
    return a + b


with SingleNodeExecutor() as exe:
    future = 0
    for i in range(1, 4):
        future = exe.submit(calc_add, i, future)
    print(future.result())
```


Janssen et al. JOSS, 10, 108 (2025) - https://github.com/pyiron/executorlib

From Python Functions to Workflows – Dependency Management

Link Python Functions with Different Resource Requirements

```python
from executorlib import SingleNodeExecutor


def calc_add(a, b):
    return a + b


with SingleNodeExecutor(plot_dependency_graph=True) as exe:
    future = 0
    for i in range(1, 4):
        future = exe.submit(calc_add, i, future)
    print(future.result())
```

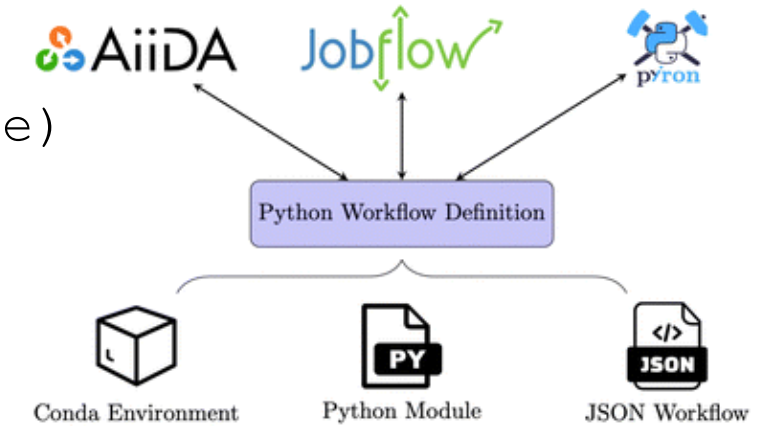


Janssen et al. JOSS, 10, 108 (2025) - https://github.com/pyiron/executorlib

From Python Functions to Workflows – Dependency Management

Link Python Functions with Different Resource Requirements

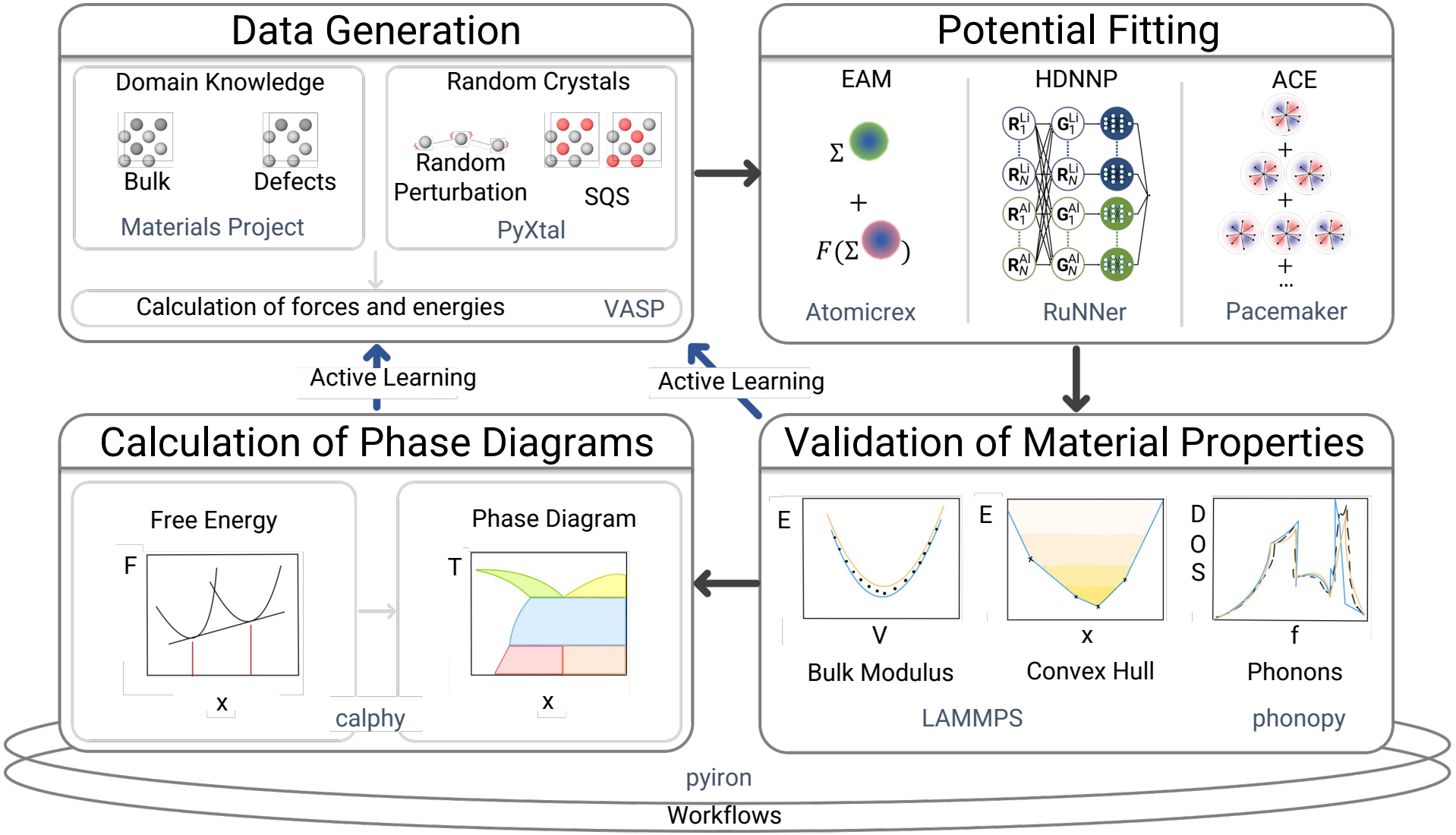
```python
from executorlib import SingleNodeExecutor


def calc_add(a, b):
    return a + b


with SingleNodeExecutor(export_workflow_filename="flow.json") as exe:
    future = 0
    for i in range(1, 4):
        future = exe.submit(calc_add, i, future)
    print(future.result())
```



Janssen et al. JOSS, 10, 108 (2025) - https://github.com/pyiron/executorlib

Temperature Concentration Phase Diagram

Ab-initio Thermodynamics



S. Menon, Y. Lysogorskiy, A. L. M. Knoll, N. Leimeroth, M. Poul, M. Qatar, J. Janssen, M. Mrovec, J. Rohrer, K. Albe, J. Behler, R. Drautz and J. Neugebauer. npj Comput. Mater., 10, 261 (2024)

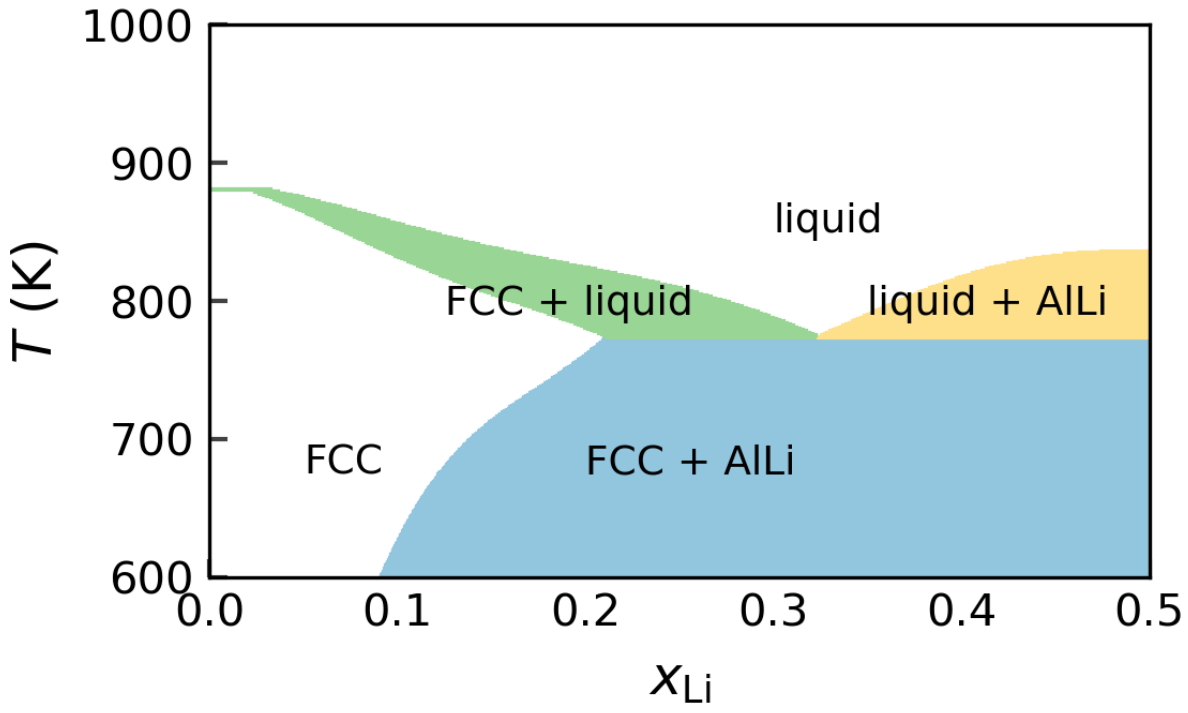
Temperature Concentration Phase Diagram

Ab-initio Thermodynamics

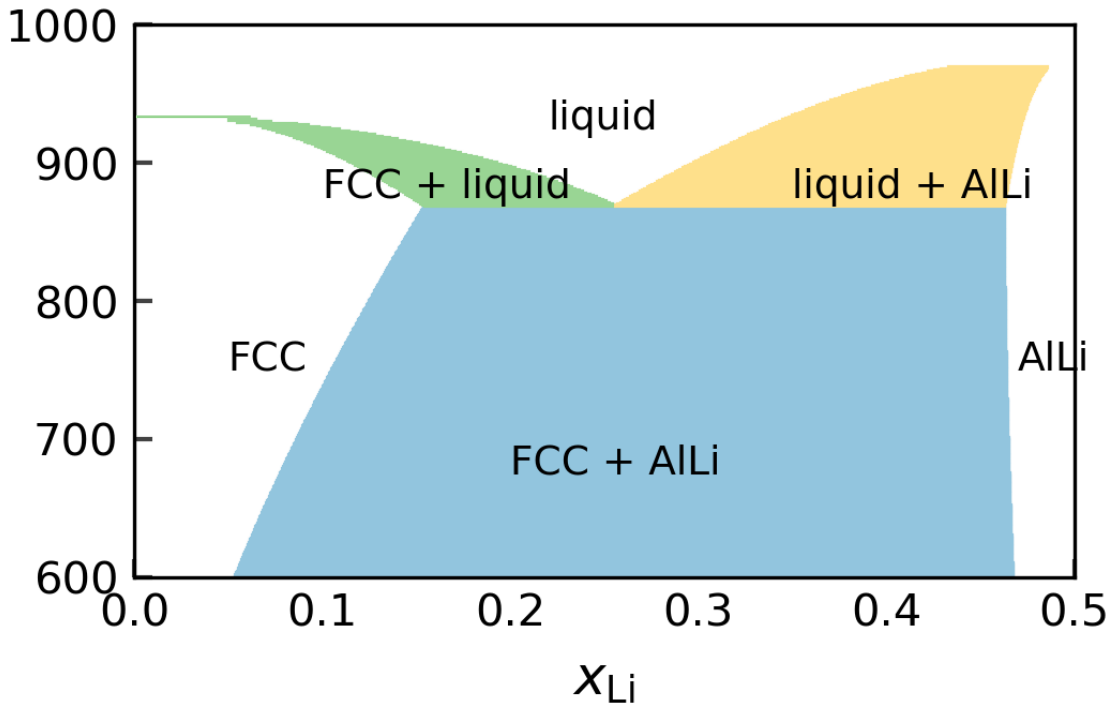
Data Generation

Potential Fitting

Machine-Learned Interatomic Potential



Experiment (CALPHAD)



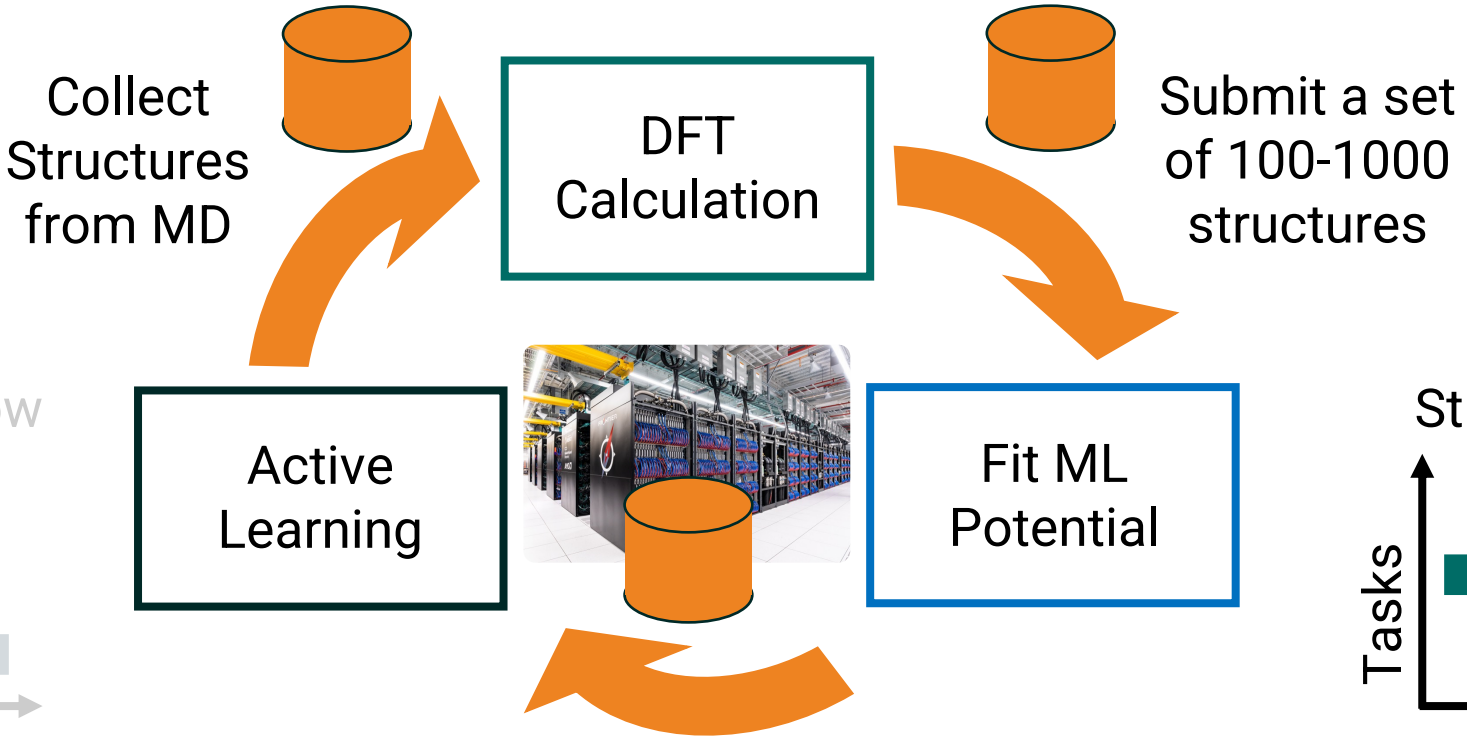
pyiron

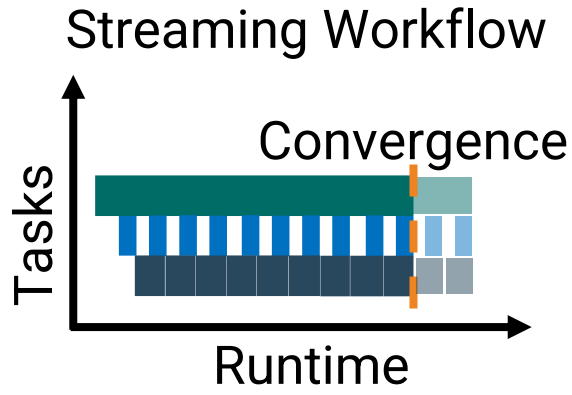
Workflows

S. Menon, Y. Lysogorskiy, A. L. M. Knoll, N. Leimeroth, M. Poul, M. Qatar, J. Janssen, M. Mrovec, J. Rohrer, K. Albe, J. Behler, R. Drautz and J. Neugebauer. npj Comput. Mater., 10, 261 (2024)

Fitting Interatomic Machine Learning Potentials

Maximize the Resource Utilization

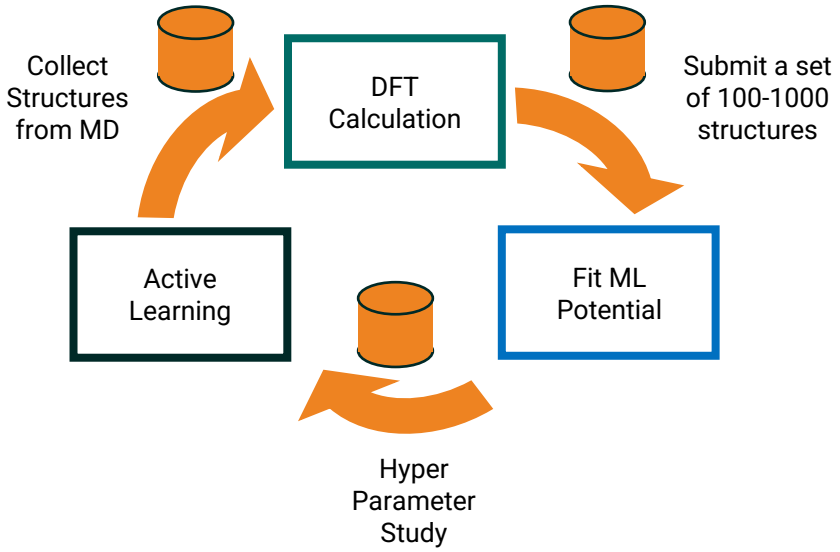


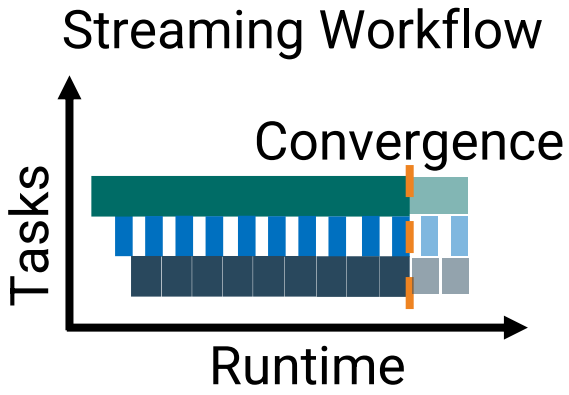


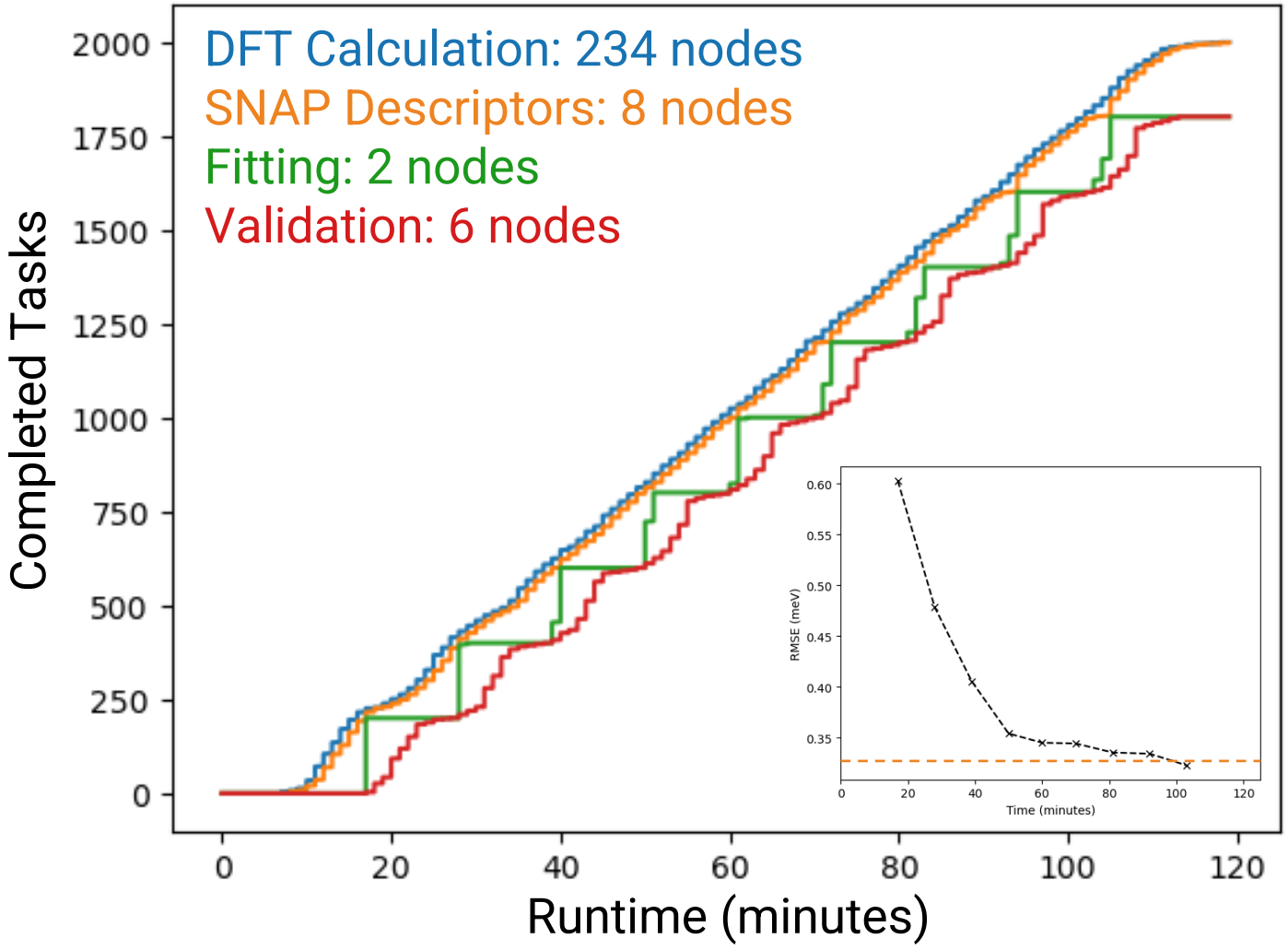
https://executorlib.readthedocs.io - Janssen et al. JOSS, 10, 108 (2025)

Develop a Machine Learning Potential in Two Hours

Testing on Frontier



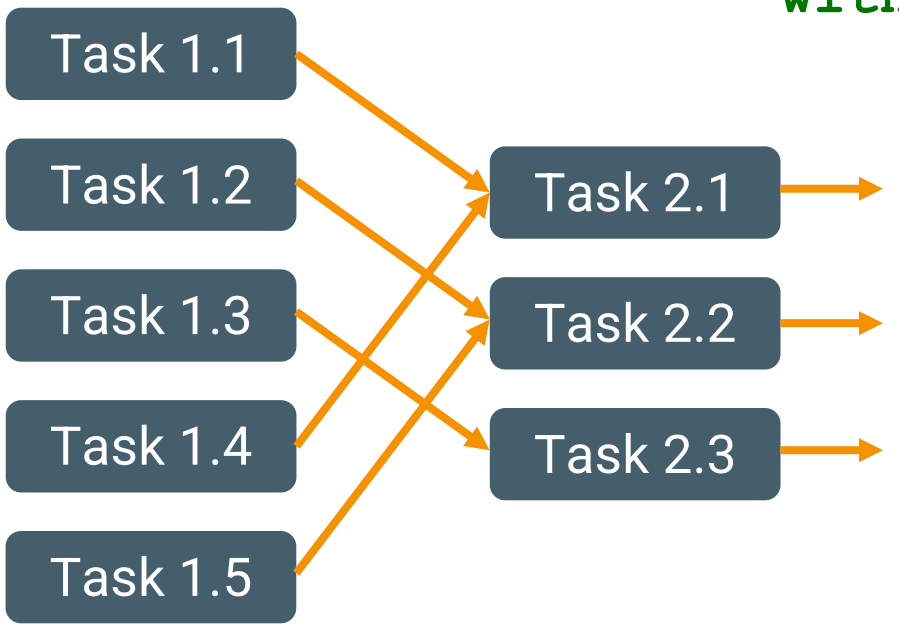




https://executorlib.readthedocs.io - Janssen et al. JOSS, 10, 108 (2025)

Streaming Workflow in Executorlib

From a Group of Tasks Take the Batch of Tasks Which Finishes First




```python
from executorlib import SingleNodeExecutor


with SingleNodeExecutor() as exe:
    task_one_lst = []
    for i in range(…):
        task_one_lst.append(
            exe.submit(task_one, …)
        )

task_two_lst = []
for f in exe.batched(task_one_lst, n=2):
    task_two_lst.append(
        exe.submit(task_two, …)
    )
```


Janssen et al. JOSS, 10, 108 (2025) - https://github.com/pyiron/executorlib

Reusing Data in Memory

Block Allocation

```
from executorlib import FluxJobExecutor

def init_function():
    return {"j": 4, "k": 3, "l": 2}

def calc_with_preload(i, j, k):
    return i + j + k

with FluxJobExecutor(
    max_workers=2,
    resource_dict={"cores": 1},
    init_function=init_function,
    block_allocation=True,
) as exe:
    fs = exe.submit(calc_with_preload, 2, j=5)
```


Janssen et al. JOSS, 10, 108 (2025) - https://github.com/pyiron/executorlib

Hierarchical Task Management

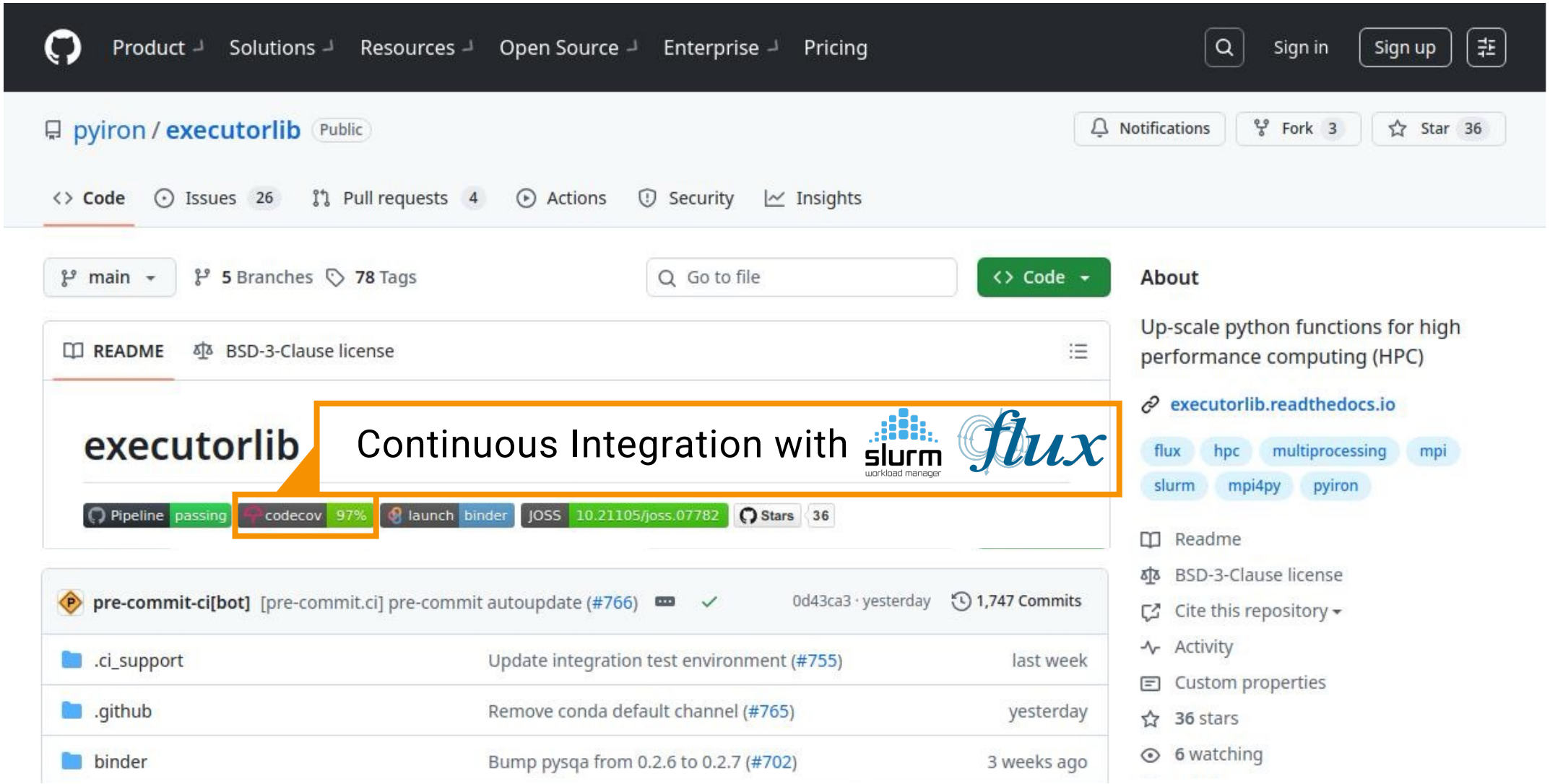
Based on Flux Hierarchical Job Scheduler

```
from executorlib import FluxJobExecutor


def calc_nested():
    from executorlib import FluxJobExecutor
    with FluxJobExecutor() as exe:
        fs = exe.submit(sum, [1, 1])
        return fs.result()


with FluxJobExecutor(flux_executor_nesting=True) as exe:
    future = exe.submit(calc_nested)
    print(future.result())
```


Janssen et al. JOSS, 10, 108 (2025) - https://github.com/pyiron/executorlib

Open-Source Development

From Scientists for Scientists

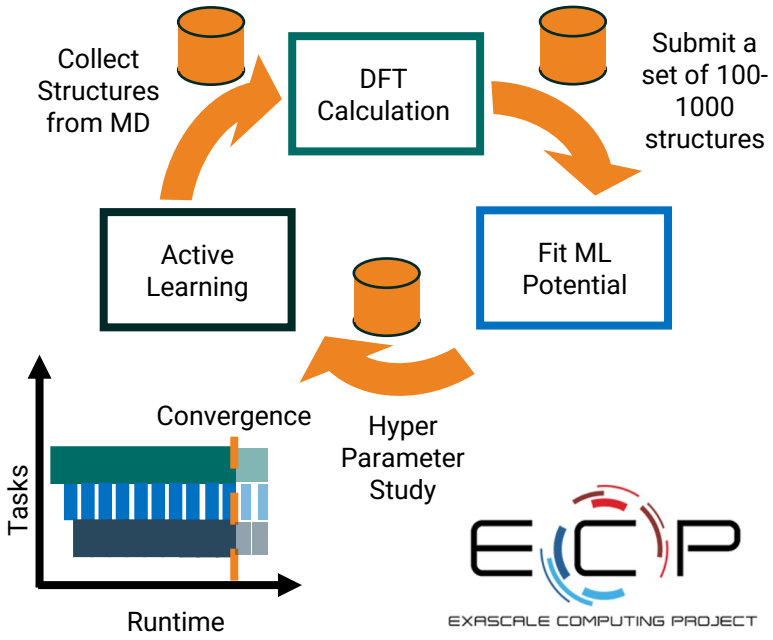


Janssen et al. JOSS, 10, 108 (2025) - https://github.com/pyiron/executorlib

Summary: Up-scale Python Functions with Executorlib – Thank you

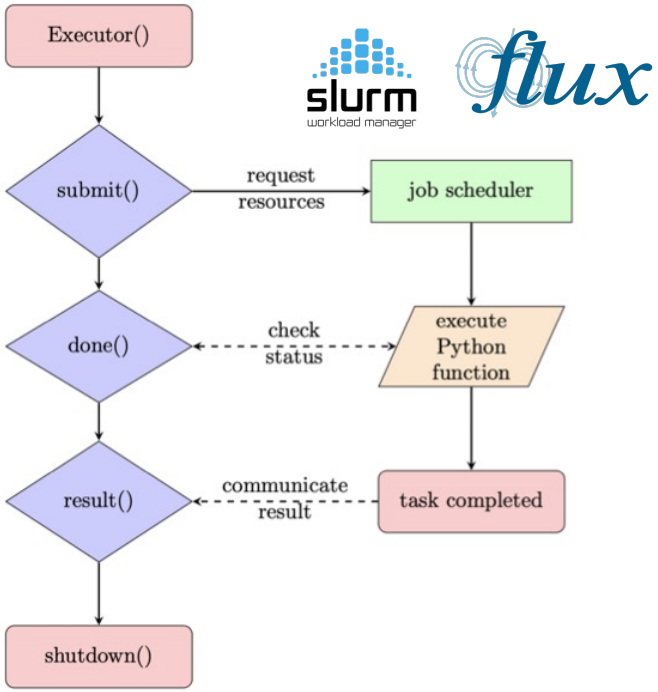
From your local workstation to high-performance computing and the Exascale

Streaming Workflows for Exascale Materials Science



Maximizing the utilization of available computational resources to gain scientific insights as fast as possible.

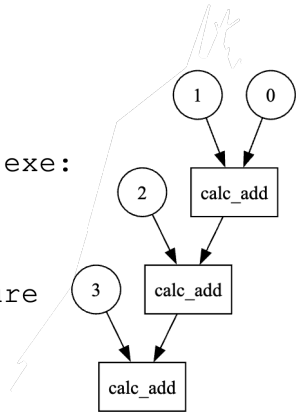
Executorlib For Up-Scaling Python Functions



No database, no daemon process, just a Python interface to high-performance computing job schedulers.

Beyond Python Functions Towards Python Workflows

```python
from executorlib import SingleNodeExecutor

def calc_add(a, b):
    return a + b

with SingleNodeExecutor() as exe:
    future = 0
    for i in range(1, 4):
        future = exe.submit(
            calc_add, i, future
        )
    print(future.result())
```



Follow the interface of the concurrent futures Executor class extended to address scientific computing challenges.

Janssen et al. JOSS, 10, 108 (2025) - https://github.com/pyiron/executorlib